

MATLAB

Vincent ISOZ



Cookbook (M2)

2014-10-18 (V0.8 Revision 101), {oUUID 1.858}

Alphorm
com

DÉMOCRATISER LE SAVOIR

TABLE OF CONTENTS

TABLE OF CONTENTS.....	2
INDEX OF TABLES.....	14
Introduction	15
MATLAB Evolution	16
About the writers.....	21
ISOZ Vincent	21
KOLANI Daname	22
Warning.....	23
Your Opinion Matters!	24
Useful Links	25
General	25
Forums.....	26
Non-Mathworks additional powerful Toolboxes and tools	26
Bibliography.....	27
MATLAB core (without Toolbox).....	28
New main core features (to be complete)	28
MATLAB 2014a (8.3).....	28
MATLAB 2013b (8.2)	28
MATLAB 2013a (8.1).....	28
MATLAB 2012b (8.0)	28
MATLAB 2012a (7.14).....	28
MATLAB 2011b (7.14)	29
MATLAB 2011a (7.12).....	29
MATLAB 2010b (7.11)	29
MATLAB 2010a (7.10/2010)	29
MATLAB 2009b (7.9/2009)	29
MATLAB 2009a (7.9/2009)	30
MATLAB 2008b (7.8/2009)	30
MATLAB 2008a (7.6/2008)	30
MATLAB 2007b (7.5/2007)	30
MATLAB 2007a (7.4/2007)	31
MATLAB 2006b (7.3/2006)	31
MATLAB R14SP3 (7.1/2005)	32
MATLAB R14SP2 (7.0.4/2005)	32
MATLAB R14SP1 (7.0.1/2004)	32
MATLAB R14 (7.0/2004).....	32
MATLAB R13 (6.5/2000).....	33
MATLAB R10 (5.2/1998).....	33
MATLAB R9 (5.1/1997).....	33
MATLAB R8 (5.0/1996).....	33
Cite MATLAB	34
User Interface	34
Use Help	47

Edit a native function source code (if available).....	49
Most asked user preferences	51
General commands.....	53
Execute system commands.....	53
Call R from MATLAB	54
Get MATLAB system information	56
Some dates stuff	61
Manage variables.....	62
Some Command Window controls	64
Use the startup.m file	66
Use the clipboard.....	70
Comments and multiple line commands	70
Save workspace/Use commands diary	71
Arithmetic operations on scalars	74
Operators of comparison	77
Real numbers manipulations	79
Command Windows real numbers formatting and output	81
Arithmetic operations on complex numbers	84
Identify the number type	87
Rounding results.....	89
Set (vectors) operations.....	90
Important practical cases	97
Counting/Removing NaN in cell arrays.....	97
Counting/Removing NaN in (numeric) vectors	98
Finding the closest number in an vector	99
Arithmetic operations and manipulations of vectors	100
Important practical cases	109
Clean a vector.....	109
Build an orthonormal vector base	110
Arithmetic operations and manipulations of matrices	112
Important practical cases	127
Removing NaN in matrices (arrays)	127
Sampling matrix columns.....	128
Creation multiple dimensional matrices.....	129
Manipulating tables (structure array)	130
Mathematical functions	136
Core (elementary) statistical functions.....	149
Generate random variable	159
Use of constants (permanent variables)	163
Working with strings	166
Working with dates	173
Important practical cases	180
Convert year/month/day matrix as dates.....	180
Difference in milliseconds/seconds between 2 dates	181
Simple user interactions	184
Message boxes.....	184
Input	185
Input Dialog.....	186
UI set color	189

UI save.....	190
UI open.....	190
UI get dir	192
Wait bar	194
Question dialog	197
Menu.....	198
List dialog.....	200
Send E-mail	201
Data management	202
MATLAB files (*.mat and *.m).....	202
Manage working paths	205
Browse and manage folders and files.....	211
Import/Export csv files.....	218
Import very clean local csv files (with comas, without headers, without texts, without dates)	218
Import clean local csv files (with semi-colons, with headers, without text, without dates).....	221
Import local csv files (with semi-colons, with headers, with text, with dates)	222
Merge csv files	225
Import web csv files (with semi-colons, with headers, with text, with dates)	227
Import from Yahoo Finance	228
Export to csv file	230
Import/Export xls/xlsx (Microsoft Office Excel) files.....	232
Import xls/xlsx local files	232
Export to xls/xlsx files.....	236
Import data with the wizard	238
Charts.....	245
Demo	245
Data sets	249
Chart wizard	251
Control chart window.....	282
Pick-up points on chart.....	293
C to R ² plots	295
Analytical based plot.....	295
Vector based plot with lines and stems.....	296
Animated vector based plot	299
Vector based plot with various lines, colors and points	301
Vector based multiple plot with legends and titles	303
Horizontal/Vertical lines method.....	305
Vector based plot with error bars.....	306
Vector based semi-log X plot with custom tick space	307
Vector based semi-log Y plot	308
Vector based log-log plot with data cursor tip	309
Subplots.....	311
Greeks letters in legends and titles (LaTeX style)	314
Plots with dates in axis	315
Multiple axis on line plots	317
Magnifier on line plot.....	320
2D vertical/horizontal bar charts.....	321
Multiple series bar chart	321
Stacked bar chart.....	322
Stacked normalized bar chart.....	323
Bar chart with error bars	324
Bar chart with bar having a special a color.....	326
Tornado Chart.....	327

<i>Bar chart with line and multiple axis</i>	328
<i>Two series bar chart with multiple axis</i>	329
<i>Probability histogram with density curve (pdf)</i>	331
Correlation 2D Heatmap	332
3D vertical bar charts	334
2D/3D sector charts (cheese).....	336
2D Surface/Area charts	339
Radial (rose) bar chart/arrow (compass) chart	341
Pareto chart.....	344
Plotmatrix.....	345
3D parametric plots	346
Vector based 3D parametric line plot	346
Animated or non-animated analytic based 3D parametric line plot	347
Animated vector based 3D parametric line plot.....	348
Vector based 3D stem parametric line plot.....	349
Vector based 3D surface parametric plot.....	350
3D surface plots.....	361
ezsurf.....	361
ezsurf.....	362
surf	363
<i>surf with an explicit or implicit function!</i>	371
Animated surf.....	372
mesh	373
surf.....	376
contour3 (3d contours)	377
surf with surf projection	380
contour (2d contours)	383
waterfall	387
2D Gradient (quiver)	388
3D Gradient.....	389
Perpendicular vectors	392
Surface plots of nonuniformly sampled data	393
<i>Cubic (triangle base interpolation)</i>	393
<i>Delaunay (triangular connections)</i>	395
<i>Spline (biharmonic-spline)</i>	395
Image mapping plots	396
Map an image on a 2D plane chart	396
Map an image on a 3D surface chart	397
Sphere plot with data mapping and animated rotation	399
Export animated plots as *.avi or *.gif.....	402
Export to *.avi	402
Export to *.gif	403
Engineering/Simulations	404
Vector field equation analysis	404
Vector field snapshots analysis.....	406
Regression/Interpolation/Extrapolation	425
OLS univariate regression	425
Spline univariate interpolation.....	427
Spline multivariate interpolation	428
Fractals	430
Save chart as image	432
Save on drive.....	432
Copy in the clipboard	434

Analysis	435
Estimate value of an univariate polynom on various points	435
Calculate the roots of an univariate polynom.....	435
Find the local min/max of an univariate function in a range	438
Find the local min/max of a bivariate function	440
1 st Method (very old one)	440
2 nd Method (modern one).....	442
Discrete convolution	442
Integration	445
Pseudo-symbolic integration of polynoms.....	445
Numeric integration.....	445
Numeric trapezoidal integration	446
Double numeric integration	447
Derivation.....	448
Pseudo-symbolic derivation of polynoms.....	448
Numeric derivation.....	449
Differential equations (ODE/PDE)	452
Pendulum (ODE).....	452
Lotka-Volterra (system of ODE)	455
Lorenz attractor (system of ODE)	457
2 body motion (system of ODE).....	459
Heat equation (parapolic PDE).....	463
Fast Fourier Transform (DFT)	467
Image processing.....	471
Get image properties	471
Show an image properly.....	473
Decompose images alpha layers	474
Build an image	475
Plot some basic images manipulations and save image	476
Filter on luminosity value of a given channel	477
Draw on an image and export the result.....	478
Compare images.....	481
Add (superpose) images.....	483
Video processing	486
Get video properties	486
Play video.....	487
Extract a frame	488
Export all video frames in a folder.....	489
Build a video from a list of (pre-processed or not) images	490
Sound processing.....	491
Get sound properties.....	491
Play, Pause (resume), Resume, Stop sound playing	492
Plot a sound spectrum	493
Sound FFT (DFT).....	495
Generate a sound and save it	497
Play sounds in sequence or parallel.....	498
Scripting	499
Script Editor most asked preferences	499
Well commented script.....	501
Define help text	503

Autocompletion	504
Conditional structure (if/switch)	506
if statement	506
switch statement (with if...)	509
Iteration structures	510
For...End iteration structure (and optimization)	510
While...End iteration structure	513
Functions (functions files)	514
One input, One output (OIOO)	514
Multiple inputs, One output (MIOO)	515
One input, Multiple outputs (OIMO)	516
Multiple inputs, Multiple outputs (MIMO)	518
Optional arguments	519
Variables scope	520
Global vs Local variables	520
Persistent variables	523
Anonymous function as input, chart output	524
Recursive functions	526
Local functions (multiple functions in one script)	528
Profile a script	529
Catching errors	531
Multiple callable functions in one *.m file	531
Debugging technics	534
M-lint	535
M-lint real time analyzer	535
M-lint reports	537
Echo on/Echo off	541
For simple scripts	541
For functions scripts	545
Manipulations during a run	546
Debugging (db...) functions	548
Add conditional breakpoints	555
Create a more complex wait bar	557
Import an XML file	558
Export an XML file	561
MATLAB Automation	563
Generate a Microsoft Word report	563
Put values into Microsoft Excel	565
Get values from a MS Excel VBA function	568
Create a ToolBox (App)	570
Install/Uninstall a Toolbox	576
Publish a script (documented script)	580
Optimization tricks	585
Statistically check code efficiency	586
linspace vs sequence	588
Simple arithmetic operation vs bsxfun	589
Create userforms (GUI)	591
Create userforms using GUIDE	591
Edit existing GUI	600
Manage static texts (legends)	601
Manage buttons and associate a callback	606
Create axes area for figures, images,	607
Manage check-boxes and associate a callback	613
Manage radio buttons group	616

Manage pop-up menu and get selected value	622
Manage multiple selection List box	627
Manage sliders	633
Manage tables.....	636
Userform calling another userform	641
Write/Get GUI variables in MATLAB workspace	644
Create userforms using only script	646
Important practical cases	646
Create a GUI that opens a file.....	646
Symbolic calculations (Symbolic Toolbox)	654
New main core features (to be complete)	654
MATLAB 2014a (8.3).....	654
MATLAB 2013b (8.2)	654
MATLAB 2013a (8.1).....	654
MATLAB 2012b (8.0)	654
Limits.....	655
Derivate/Partial Derivate.....	656
Integration	657
Taylor series	657
Heaviside function.....	659
Matrix determinant	661
Integral Transformation.....	661
Fourier Transform	662
Laplace Transform	663
Z Transform.....	665
Jacobien	666
MuPAD	666
Databases (with Database/Datafeed toolbox)	669
Connect to mySQL 5.5.36.....	669
Connect to Microsoft Access 2003/2010 (*.mdb/*.accdb)	674
Connect to Oracle Express 11g	678
Connect to MS SQL Server Express 2008	682
Connect to Yahoo! Finance.....	688
Statistics (with Statistical and Curve Fitting Toolbox).....	689
Datasets	689
Import/Export *.csv files as data set (data frame).....	689
Import *.xls/*.xlsx files as data set (data frame).....	691
Convert structure array to data sets	692
Clean messy or missing data in datasets	694
Select/Remove column (variable)	696
Create subsets and filter	696
Delete rows and clean duplicates	698
Add variables.....	698
Change variable name	699
Change column order	699
Random sample of a dataset.....	700
Sort a dataset	701
Merge datasets.....	702
Plots	703
Histogram/CDF/PDF.....	703

Box-Whisker plot	706
Scatter Group plot	707
QQ plot.....	708
Histogram Scatter.....	709
Copulas.....	710
Gaussian copula.....	710
Descriptive statistics.....	713
Geometric mean	713
Harmonic mean	714
Trimmed mean (M-statistics).....	714
CDF/iCDF/Quantile/Centile.....	716
Skewness/Kurtosis	717
Hypothesis tests.....	720
Tests power, sample size and resolution	720
Power of a Z test.....	720
Sample size of a one sample Z test.....	721
Difference (resolution) of a one sample Z test.....	721
Power of a t test.....	722
Sample size of a one sample t test	722
Sample size of a one sample p test	723
One sample Z-test.....	723
One sample t-test.....	724
One sample t-test with explicit target mean	725
Two sample homoscedastic t-test.....	726
Two sample heteroscedastic t-test (Welch test)	727
Paired t-test.....	728
Chi-square variance test	729
Two sample F-test for equality of variances	730
Two sample Levene test for equality of variances	731
Brown-Forsythe (median).....	731
Absolute Levene.....	733
Quadratic Levene (square).....	734
Test for the median (one sample sign test).....	735
Binomial signed rank test (dixit: median test paired samples).....	736
Two unpaired samples Wilcoxon (Mann-Withney) rank sum test	737
Wilcoxon signed rank test for 1 sample	738
Wilcoxon signed rank test for two paired samples	739
ANOVA	740
One-way fixed factor canonical ANOVA.....	740
Two-way fixed factors canonical ANOVA.....	743
Friedman's test (canonical non-parametric ANOVA based on ranks).....	745
Kruskal-Wallis test (canonical non-parametric one-way ANOVA	747
Adequation tests	750
Chi-square goodness of fit for Poisson distribution	750
Kolmogorov-Smirnov (one sample) adequation test.....	750
Anderson-Darling adequation test	752
Regression Techniques.....	753
Ordinary Least Squares (parametric) univariate regression	753
1 st method (coefficient and constant can not be extracted)	753
2 nd method (coefficient and constant and others can be extracted)	757
3 rd method (using Curve Fitting Toolbox)	759

Manage outliers	765
Residuals and Leverage (residual diagnosis)	769
Regression diagnostics wizard.....	770
Ordinary Least Squares (parametric) multivariate regression (with/without interactions)	
.....	772
1 st method (coefficient and constant can not be extracted)	772
2 nd method (coefficient and constant and others can be extracted)	775
Polynomial multivariate regression using OLS and Curve fit Toolbox.....	777
(Forward) Stepwise regression.....	780
General Linear Models (GLM)	783
Normal regression	783
Log-linear Poisson regression.....	784
Binomial (logistic) regression.....	785
Data Mining/Machine Learning	787
K-means (euclidian full).....	787
Hierarchical ascendant (CAH) Dendrogram	794
CART Classification trees (Gini index based).....	799
k-NN (k-Nearest Neighbors).....	803
Naive Bayes.....	805
Non classical multidimensional scaling	808
Parametric Principal Component Analysis	810
Linear Discriminant Analysis (Fisher Discriminant)	815
Control charts (CC)	822
P attribute CC	824
NP attribute CC	826
C attribute CC.....	828
U attribute CC	830
Xbar-R continuous CC	832
Rbar-R continuous CC	833
Sbar-S continuous CC	835
I-EM continuous CC	836
Moving Average continuous CC.....	839
Design of experiments.....	841
Two-level full factorial design	841
Full factorial design.....	842
Fractional factorial design.....	843
Simulations techniques.....	844
Kernel Smoothing	844
Monte Carlo standard technic	846
Monte Carlo antithetic variables technique.....	847
Monte Carlo with antithetic quasi-random variables technique.....	849
Latin HyperCube	850
Monte Carlo by control variable (variance reduction technique)	852
Monte Carlo importance sampling.....	854
Optimization (with Optimization Toolbox/Global Optimization Toolbox)	
.....	858
Linear optimization using LP Simplex (LPS)	858
LP-S Factory optimization problem (inequality linear system)	858
LP-S Naive (without covariance...) Markowitz parametric portfolio optimization problem (equality linear system without target value)	861

LP-S Naive (without covariance...) Markowitz parametric portfolio optimization problem (equality linear system with target value)	862
Function constrained optimization (FCO)	863
FCO with constraints and without specific target	863
FCO with constraints and specific target	865
FCO with non linear constraints (and specific target).....	866
Quadratic optimization (QO)	868
QO using conjugate gradient method (Q-CG)	868
QO using interior point convex method (Q-IPC).....	870
Genetic algorithms	871
Finance (with Finance/Optimization and Econometrics Toolboxes)	875
Fractional prices	875
Return	875
Maximum Draw Down.....	876
Box-Cox transformation.....	876
Simple capital calculations.....	878
Composed Interest (fixed future value).....	878
Composed Interest (fixed present value).....	879
Amortization and depreciation	880
Straight line depreciation (SLN)	880
Sum of years digit (SYD).....	880
Fixed declining-balance depreciation schedule (DB)	881
Double declining balance (DDB).....	881
Loans and credits.....	882
Payment for a loan based on constant payments and a constant interest rate (PMT)	882
Principal Payment (PPMT)	883
Interest Payment (IPMT).....	884
Payment Present Value (PV).....	885
Plot resume of loan payment.....	886
Annuities	888
Postnumerando actual annuity value (PV)	888
Postnumerando forward annuity value (PV).....	888
Postnumerando number of periods (NPER).....	889
Postnumerando annuity rate (ANNURATE)	889
Corporate finance	890
Net Present Value (NPV)	890
Periodic NPV	890
non-Periodic NPV	890
Internal Rate of Return (IRR).....	891
Periodic IRR.....	891
non-Periodic IRR.....	891
Modified Internal Rate of Return (MIRR)	892
Treasure bonds/securities calculations	893
Clean price of a security (PRICE).....	893
Price of treasury bond/security (TBILLPRICE)	893
Yield of a treasury bond (TBILLYIELD)	894
Discount rate of a security (DISC)	895
Annual yield for a discounted security (YIELDDISC).....	895
Bond duration and Macaulay duration	896

Naive Markowitz portfolio models	897
Optimal short-long portfolio (without transaction costs).....	898
Optimal portfolio with implicit constraints no short sales allowed (without transaction costs).....	900
Optimal portfolio with implicit constraints no short sales allowed (with transaction costs)	905
Get portfolios on the efficient frontier with explicit constraints (without transaction costs).....	906
Equally weighted portfolio (EWP) (without transaction costs)	908
Maximum Sharpe Ratio (max. diversification) portfolio with implicit constraints (without costs)	910
Optimal portfolios for target returns and CVaR minimization (without transaction costs)	912
Optimal portfolios for target risks and CVaR minimization (without transaction costs)	913
Values at risk (VaR).....	914
Parametric relative VaR	914
Parametric absolute VaR.....	914
non-Parametric (absolute) VaR.....	914
Condition (absolute) Normal VaR	914
Options Pricing & Monitoring	917
Pricing European Call & Put.....	917
European Call & Put implied volatility.....	919
European Call & Put delta.....	919
European Option (call/put) Gamma	920
European Option (call/put) Vega	920
Option european lambda	920
Option european theta	921
Option european rho.....	921
Implied volatility surface of an European option.....	921
Volatility smile of an European option	922
CRR binomial European option pricing model.....	927
Time series	929
Autocorrelation.....	929
Partial autocorrelation	932
Moving average MM3 (without ARIMA)	934
Simple Exponential Weighted Moving Average (EWMA)	936
Univariate AR(2)/ARIMA(2,0,0).....	939
Univariate ARIMA(0,0,0)/Moving Average.....	942
Univariate ARIMA(0,1,0)	945
Univariate ARIMA(0,1,1)/Simple Exponential Smoothing (EWMA)	949
Univariate ARIMA(0,2,2)/Holt's double exponential smoothing	951
Univariate ARMA(1,1)/ARIMA(1,0,1)	954
Univariate GARCH(1,1)	957
Applications (without Financial Toolbox)	962
Bond portfolio with duration/convexity targets	962
Brownian motion and Wiener Process	963
Risk-return asset pricing models	965
Capital Asset Pricing Model (CAPM).....	965

Three-factor Fama-French model (FFM).....	966
Fama-McBeth.....	968
Options Pricing.....	971
CRR binomial option pricing model without Financial Toolbox.....	971
Drifted CRR binomial option pricing model without Financial Toolbox.....	972
Jarrow-Rudd binomial option pricing model without Financial Toolbox	974
Jarrow-Rudd binomial risk neutral option pricing model without Financial Toolbox.....	975
Leisen-Reimer binomial option pricing model without Financial Toolbox	977
Tian binomial option pricing model without Financial Toolbox.....	979
Portfolios models.....	981
Global Minimum Variance Efficient Portfolios (GMVP) without Toolboxes using langrangien multiplier	981
Global Minimum Variance Efficient Portfolios (GMVP) using only Optimization Toolbox	982
Portfolio Performance	991
Michaud Optimal Portfolio (bootstrap/resample efficiency).....	993
CVaR portfolio.....	997
Black-Littermann portfolio	1004
Signal processing (Signal processing Toolbox).....	1008
Circuit Analysis (Simulink Toolbox)	1012
Create an RLC circuit.....	1020
Neural Networks (Neural Network Toolbox)	1027
Image Processing (with Image Processing and Acquisition Toolboxes) ..	1039
Play with the image tool wizard	1039
Image as a function	1043
Image histogram	1045
Filters (blur, motion) on an image.....	1049
Crop an image	1053
Convert a *.gif file into an *.avi.....	1054
Overlay images with masks.....	1056
Objects recognition and extraction.....	1060
Without references	1060
With reference	1061
Image filters (convolution).....	1067
Gaussian (blur) filter	1067
1 st method using imfilter()	1067
2 nd method using convn().....	1069
3 rd method using conv2()	1070
External Camera Acquisition	1071
Bioinformatic (with Bioinformatics Toolbox)	1077
Shortest Path.....	1077
Parallel Computing (with Parallel Computing Toolbox)	1079
Clusters computing.....	1079
NVidia GPU computing	1083
Advanced scripting (C#, C++, Compiler Toolbox)	1084
Create a *.p (protected script) file (aka "pcode").....	1084
C MEX file	1085
Create executable (*.exe)	1088
Call MATLAB from C# (VS Express 2010)	1092

Call MATLAB from C++ (VS Express 2010 x64).....	1098
Compile DLL and call from C#	1124
Passing a scalar.....	1124
Passing a constant, vector and matrix	1134
Compile DLL and call from C	1136
Compile JAR and call from JAVA	1140
VHDL	1161

INDEX OF TABLES

Table 1 Standardized color codes.....	290
Table 2 Common dates format syntaxes	316
Tableau 3 ODE solvers	453

Introduction

MATLAB is the abbreviation for *MA*trix *LAB*oratory and is actually the leading environment for technical computing. Originally written in Fortran, by C Moler, MATLAB was designed to facilitate the access to the matrix framework developed for the LINPACK and EISPACK projects. The current version, written in C by the MathWorks Inc., exist in a Professional version and a Student version. Its availability is ensured on multiple platforms: Linux, Mac, Windows and several parallel machines.

MATLAB is a comprehensive and almost easy to use for scientific computing but very expensive for students or independent/small companies (unlike R, Octave, Scilab or other free open source equivalent alternatives). It provides to engineers, researchers and any scientific an interactive system that integrates a lot of scientific numerical computation and visualization algorithms (but can do much better!). It is a powerful, almost open and programmable system allowing especially remarkable gains in productivity and creativity.

MATLAB is relatively complete, open and extensible for calculation and visualization algorithms. It has hundreds (if not thousands, depending on the version and the optional toolbox around the MATLAB core solution) mathematical, scientific and technical functions. The matrix approach allows MATLAB to process data without large size variables and perform numeric and symbolic calculations quickly and reliably. Thanks to the graphics capabilities of MATLAB, it becomes very easy to interactively change the various graphics settings and to adapt them according to our wishes.

The fact that the source code of some functions integrated in MATLAB can be inspected gives sometimes the engineers to develop better custom made solutions (also do Python Scipy...).

MATLAB has its own language that is very fast and allows time savings for Engineers compared to other languages like C, Fortran, Python and the TurboPascal even if the language is horrible to use (non-typed, bad closed lined, etc.). With MATLAB, we can make connections dynamically, to C, .Net or Fortran programs, exchange data with other applications (via DDE: MATLAB server or client) or use MATLAB as analysis and visualization engine.

In terms of development speed, MATLAB is always faster than C/C++. This is to be expected with a higher-level language, just as C/C++ is faster to develop with than assembler.

But a C/C++ programmer with an infinite amount of development time can beat MATLABs speed, in the same way an assembly programmer with an infinite time can beat a C/C++ programmer.

MATLAB also includes a set of tools specific to engineering areas called Toolboxes. Essential for most users, Toolboxes are collections of functions that extend the MATLAB environment to solve specific types of problems. The areas covered are varied and include signal processing, automatic control, system identification, neural networks, fuzzy logic, structure calculation, statistics, database access, etc.

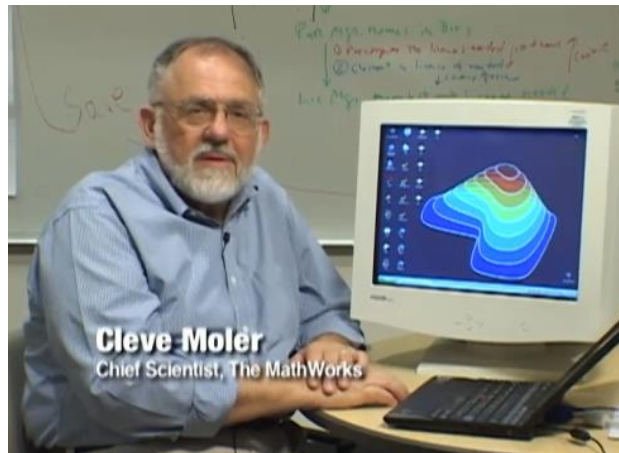
This document can also strongly help you to prepare the following certifications:



Like all my e-books you will found here only subjects that are proven and calculated by hand in the theoretical course.

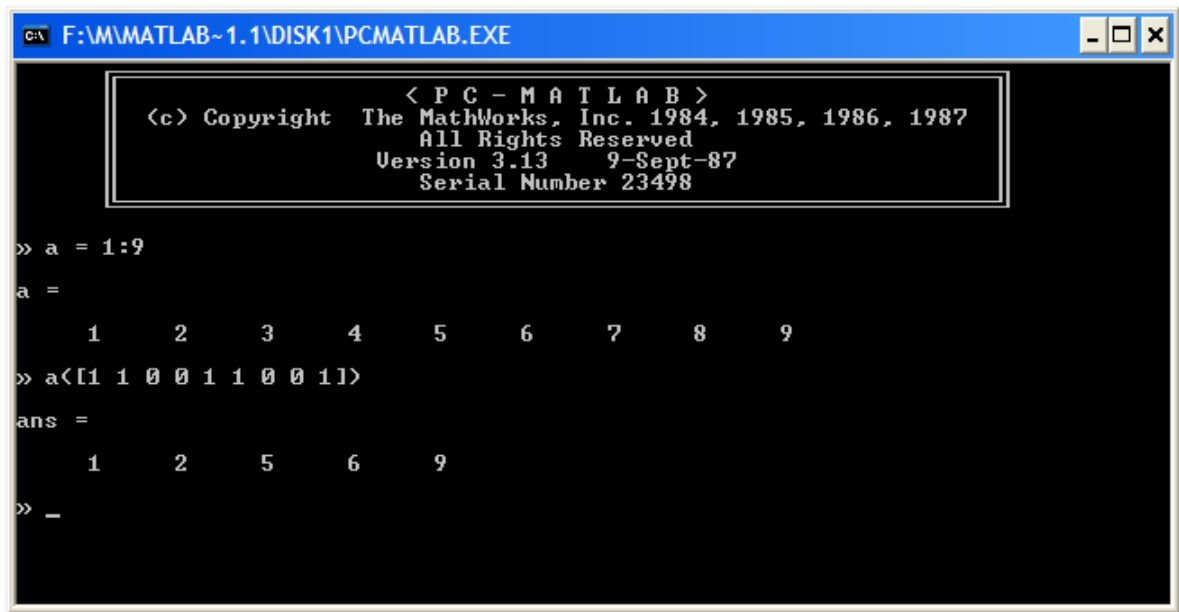
MATLAB Evolution

MATLAB was Programmed by Cleve Moler as an interface for EISPACK & LINPACK.



- 1957: Moler goes to Caltech. Studies numerical Analysis
- 1961: Goes to Stanford. Works with G. Forsythe on Laplacian eigenvalues.
- 1977: 1st edition of MATLAB; 2000 lines of Fortran 80 functions (now more than 8'000 functions)
- 1979: Met with Jack Little in Stanford. Started working on porting it to C
- 1984: Mathworks is founded

Below is a screen shot from MATLAB 3.13:

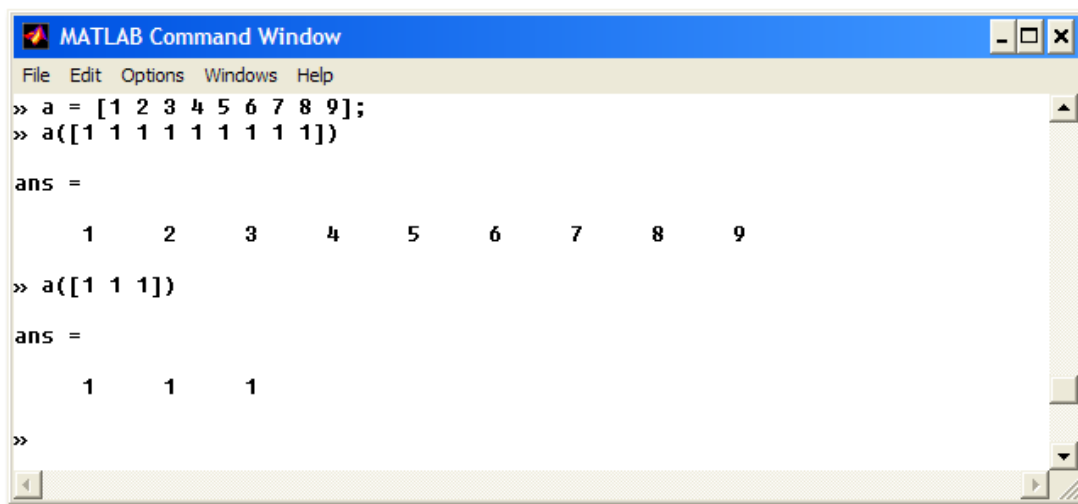


The screenshot shows the MATLAB 3.13 command window. The title bar reads 'F:\M\MATLAB~1.1\DISK1\PCMATLAB.EXE'. A copyright notice is displayed in a box: '< P C - M A T L A B > <c> Copyright The MathWorks, Inc. 1984, 1985, 1986, 1987 All Rights Reserved Version 3.13 9-Sept-87 Serial Number 23498'. The command prompt shows the execution of 'a = 1:9', resulting in a row vector of integers from 1 to 9. Then, the command 'a([1 1 0 0 1 1 0 0 1])' is entered, and the output 'ans = 1 2 5 6 9' is shown.

```
< P C - M A T L A B >
<c> Copyright The MathWorks, Inc. 1984, 1985, 1986, 1987
All Rights Reserved
Version 3.13 9-Sept-87
Serial Number 23498

>> a = 1:9
a =
     1     2     3     4     5     6     7     8     9
>> a([1 1 0 0 1 1 0 0 1])
ans =
     1     2     5     6     9
>> _
```

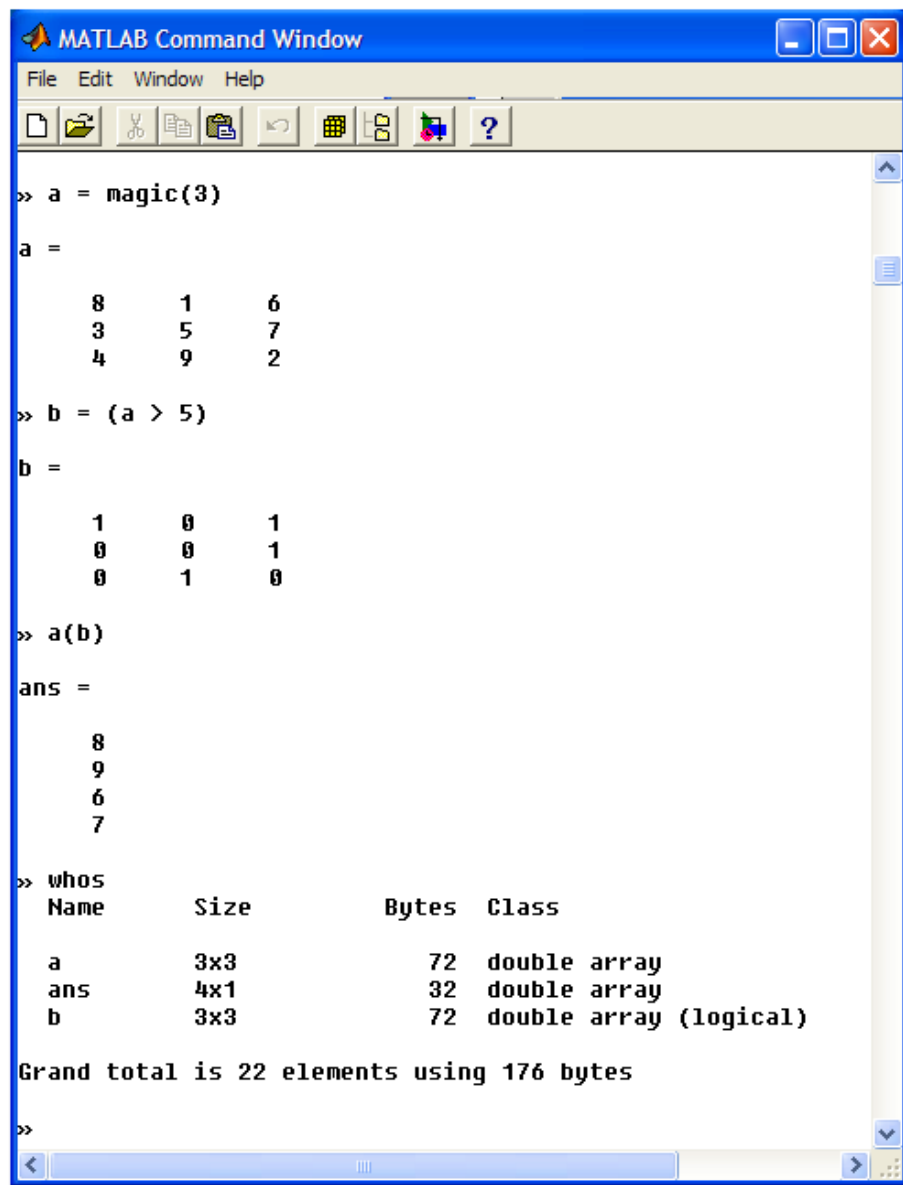
Here's a screen shot of MATLAB 4.2c (1994):



The screenshot shows the MATLAB 4.2c Command Window. The title bar reads 'MATLAB Command Window'. The menu bar includes 'File', 'Edit', 'Options', 'Windows', and 'Help'. The command prompt shows the execution of 'a = [1 2 3 4 5 6 7 8 9];' and 'a([1 1 1 1 1 1 1 1 1])', resulting in a row vector of integers from 1 to 9. Then, the command 'a([1 1 1])' is entered, and the output 'ans = 1 1 1' is shown.

```
MATLAB Command Window
File Edit Options Windows Help
>> a = [1 2 3 4 5 6 7 8 9];
>> a([1 1 1 1 1 1 1 1 1])
ans =
     1     2     3     4     5     6     7     8     9
>> a([1 1 1])
ans =
     1     1     1
>>
```

Here's a screenshot from MATLAB 5.1 in 1997:



The screenshot shows the MATLAB Command Window interface. The title bar reads "MATLAB Command Window". The menu bar includes "File", "Edit", "Window", and "Help". The toolbar contains icons for file operations (new, open, save, print, delete), editing (undo, redo, cut, copy, paste), and other functions (help, zoom, scroll). The command history shows the following sequence of commands and outputs:

```
>> a = magic(3)
a =
     8     1     6
     3     5     7
     4     9     2

>> b = (a > 5)
b =
     1     0     1
     0     0     1
     0     1     0

>> a(b)
ans =
     8
     9
     6
     7

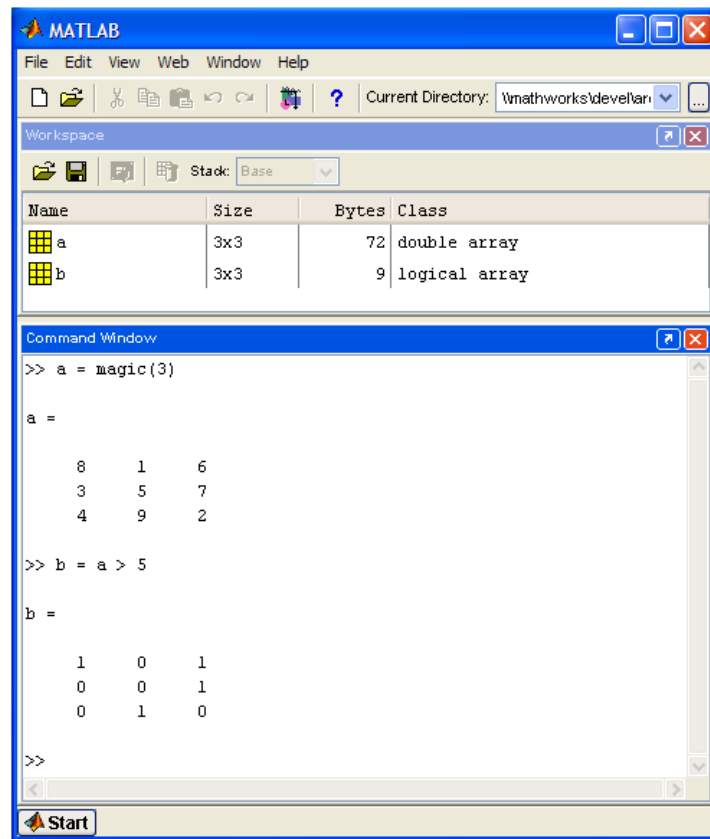
>> whos
Name      Size      Bytes  Class
a         3x3         72  double array
ans       4x1         32  double array
b         3x3         72  double array (logical)

Grand total is 22 elements using 176 bytes

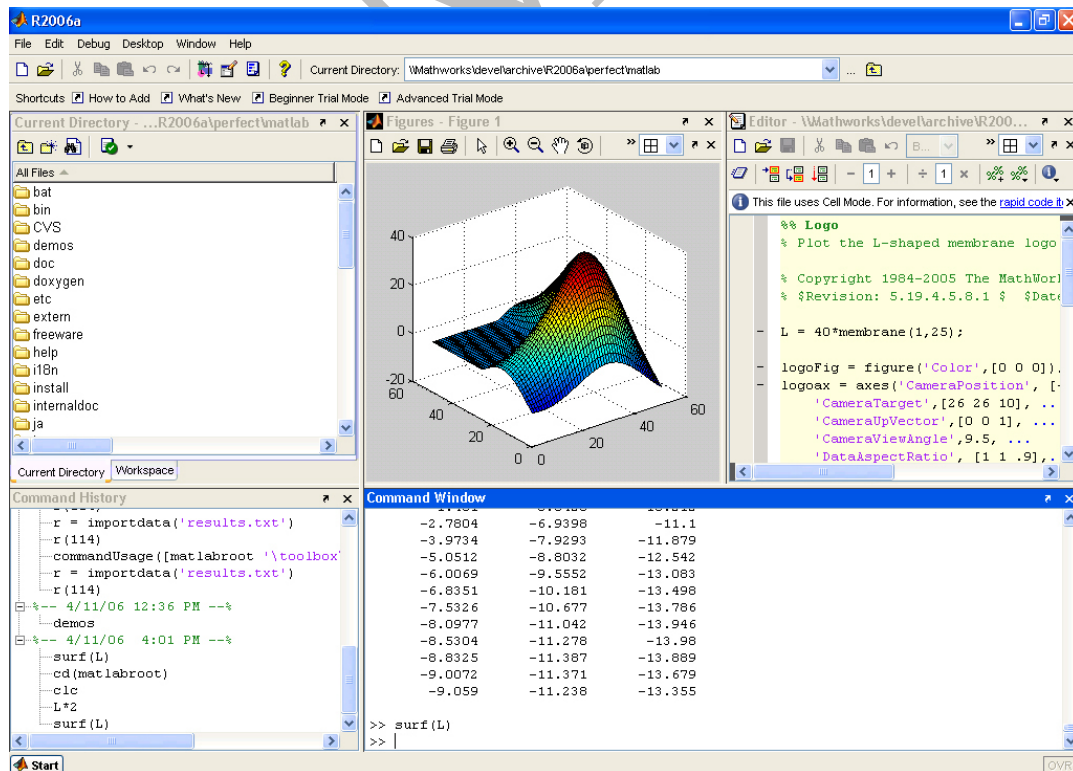
>>
```

The window has a scroll bar on the right and a status bar at the bottom with navigation arrows.

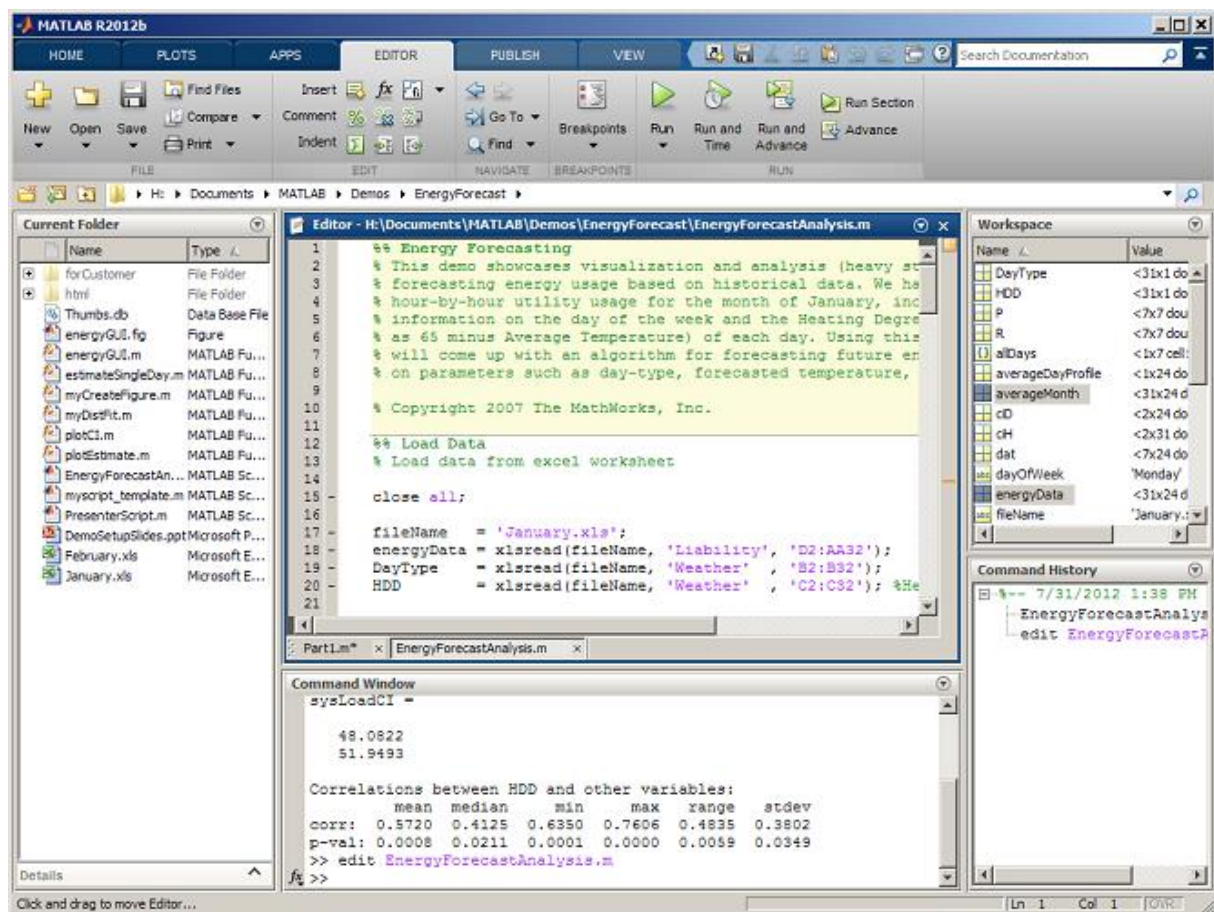
A screenshot from MATLAB 6.5 in 2002:



MATLAB 7 in 2005:



MATLAB 8 in 2012:



About the writers

ISOZ Vincent



Resident until this date in Lausanne (Switzerland)

Trainer: Physicist Engineer HES (B.Sc.)

Year of birth: 1978

E-mail: isoz@sciences.ch

Director of the company Scientific Evolution Sàrl specialized in consulting and training for high-potentials especially quantitative analysts in project management, quality, engineering, finance and decision making. Incidentally, happy freelancer that gathers since young information from articles everywhere and now addicted to everything related to applied mathematics in the management of organization (Businesses, States), TQM (Total Quality Management) with Post-taylorisme and related softwares.

Consultant in applied mathematics tutoring quantitative analysts (graduate and posgraduate) in French, English and German and author of several French and English eBooks in the following areas:

- statistical process control (parametric and nonparametric methods)
- predictive modeling / advanced decision (decision trees, Markov chains)
- Operations Research (quasi-Newton, simplex, genetic algorithms, algorithm GRG)
- data mining (neural networks, PCA, CA, regressions, scoring, clustering, etc.)
- risk modeling in project management and corporate finance (monte-carlo, bootstrapping)
- project management (best practices and theoretical models EFQM+Six Sigma, MS Office Project)
- ISO 9001:2008, 5807:1985, 10015:1999, 31000+31010:2009, 8258:1991, 10017:2003
- Adobe Photoshop and Illustrator
- 12 applications of the Microsoft Office System (Project, Visio, SharePoint, Access, etc.)

At this date I made interventions in over 200 companies, including some of to the Fortune 500 according to the 2009 listing and also in three universities and engineering schools in Switzerland for databases and stochastic risk simulations trainings. I also gave training in several multinational to executives in one to one. Sometimes I give training on softwares like MS Project, MS Visio, MS Access and approximately 20 others softwares that I delegated the training organization to companies specialized in computer softwares trainings (university degree level and below).

Finally, I would advise anyone wishing to really master the subject to read my e-book on Applied Mathematics (~ 4'900 pages).

KOLANI Daname

I'm currently living in Casablanca (Maroc)

Graduate in Finance from the Business School ENCG-Casablanca

Born in 1990, I am from Togo.

E-Mail: daname.kolani@hotmail.com

I am passionate in the field of finance, economics, and other corollary as econometrics, also in statistics & probabilities. Apart, the theories associated with those fields, I have a great interest in tools that underpin all of these areas. I am bilingual (French & English) and my specific competences include:

- Corporate Finance (Investment decision & Financing, Financial Analysis and Diagnosis, Management Control and Company Evaluation...)
- Market finance (financial portfolio management, financial asset valuation, risk management tools ...)

I also have an interest in computational finance skills in R Project, VBA, Excel, MATLAB, EViews & SPSS.

I aspire to get into consulting and thereby contribute to the optimization of financial decisions and the implementation of efficient financial strategy consistent with the business model of companies.

Warning

The content of this document is elaborated by a development process by which volunteers reach a consensus. This process that brings together volunteers, research also the point of view of people interested in the topic of this document. The person in charge of this document administers the process and establishes rules to promote fairness in the consensus approach. It is also responsible for drafting the text, sometime for testing/evaluating or independently verifying the accuracy or completeness of the presented information.

We decline no responsibility for any injury, damage or any other kind, special, incidental, consequential or compensatory, arising from the publication, application or reliance on the content of this document. We make no express or implied warranty on the accuracy or completeness of any information published in this document, and do not guarantee that the information contained in this document meet any specific need or goal of the reader. We do not guarantee the performance of products or services of one manufacturer or vendor solely by virtue of the document content.

By publishing texts, it is not the intention of this document to provide services on behalf of any person or entity or performing any task to be accomplished by any person or entity for the benefit of a third party. Anyone using this document should rely on its own independent judgment or, where that is appropriate, seek the advice of a qualified expert to determine how to exercise reasonable care under all circumstances. The information and standards on the topic covered by this document may be available from other sources that the reader may wish to visit in search of points of view or additional information not covered by the contents of this document.

We have no power in order to enforce compliance with the contents of this document, and we do not undertake to monitor or enforce such compliance. We have no certification, testing or inspection activity of products, designs or installations for safety or health of persons and property. Any certification or other statement of compliance regarding information relating to health or safety of persons and property, mentioned in this document, can not possibly be attributed to the content of this document and remains under the responsibility of the certification center or the concerned reporter.

Your Opinion Matters!

As a reader of this document, you are the critic and the most important commentator. Your opinion counts and it is very interesting for us to know what is good, what can be better and topics that you would like to see be treated.

You can send me an e-mail to share what you liked or disliked in this document to ensure continuous improvement.

If you want to complete this support with a subject that is important to you and for which you have the mathematical proof please do not hesitate to contact me. I will integrate the subject specifying your name and surname.

Note that unfortunately I can not answer to technical engineering questions or business issues for free by email for obvious professional reasons.

E-mail: isoz@sciences.ch

With that, I want to say a huge "THANK YOU" to all of my customers, partners, colleagues, students and friends who with their feedback, return on experience helped me write this e-book.

Useful Links

General

<http://www.sciences.ch> (the companion website containing mathematical proofs)

<http://www.google.com>

<http://www.youtube.com>

<http://www.scribd.com>

<http://www.mathworks.com/products/>

<http://www.mathworks.ch/ch/help/matlab/release-notes.html>

<http://www.mathworks.com/matlabcentral/fileexchange/>

<http://www.mathworks.ch/discovery/>

<http://www.mathworks.ch/discovery/gallery.html>

<http://www.mathworks.com/matlabcentral/newsreader/>

<http://www.undocumentedmatlab.com>

<http://www.imageprocessingblog.com>

<http://www.alphorm.com> (for people that want a part of this eBook as a video training):



Forums

<http://www.mathworks.ch/matlabcentral/answers/>

<http://www.talkstats.com>

<http://matlab.developpez.com>

Non-Mathworks additional powerful Toolboxes and tools

<http://www.mosek.com> (optimization toolbox: non-free)

<http://www.cvxr.com> (another convex optimization toolbox)

http://www.asu.cas.cz/~bezdek/vyzkum/rotating_3d_globe/ (earth globe statistics: free)

<http://www.nag.co.uk/numeric/MB/start.asp> (high speed calculation toolbox: non-free)

<http://www.idquantique.com/random-number-generators/products.html> (quantum rd generator)

Bibliography

Here is a list of books of extraordinary educational and rigor quality I had the chance to have in my hands and that I strongly recommend the acquisition. I've read many others (more than a hundred!) but they are so bad they do not need to be mentioned.

The reader will understand that I recommend strongly to complete the reading of this e-book (non-exhaustive about MATLAB) by the book-list below.



Éléments de mathématiques appliquées / ~4'900 pages / Éditions Sciences.ch / Vincent ISOZ / 3^{ème} édition
ISBN: 978283999327



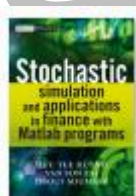
Essentials MATLAB for Engineers and Scientists / 427 pages / ELSEVIER Editions / Brian HAHN & Dan VALENTINE
ISBN: 9780123748836



MATLAB a practical introduction to programming and problem solving / 524 pages / ELSEVIER Editions / Stormy ATTAWAY
ISBN: 9780123850812



Introduction to MATLAB for Engineers / 577 pages / McGraw-Hill Editions / William J. PALM III
ISBN: 9780073534879



Stochastic simulations and applications in finance with MATLAB programs / 356 pages / Wiley Finance Edition / Huu Tue HUYNH, Van SON LAI, Issouf SOUMARÉ
ISBN: 9780470725382



Financial Instruments Toolbox / 2347 pages / Mathworks Edition /
ISBN: -

MATLAB core (without Toolbox)

This chapter is dedicated to all MATLAB topics that don't require any installed Toolboxes!

First thing you must know!: When you install MATLAB on a x64 O.S. then it will automatically install the x64 version of MATLAB. Same behavior if you are on a x86 system.

New main core features (to be complete)

Note that all changes are not documented here! Please refer to the official release notes for a full list.

<http://www.mathworks.ch/ch/help/matlab/release-notes-older.html>

MATLAB 2014a (8.3)

- Core: Pop-up Command History for recalling, viewing, filtering, and searching recently used commands in the Command Window
- Create your own icons when compiling a MATLAB app

MATLAB 2013b (8.2)

- You can now package a custom icon in an app that you create

MATLAB 2013a (8.1)

- New functions *strsplit*
- Option to add separators between controls on the quick access toolbar
- Additional icon choices, auto-scaled thumbnails, and text-formatting options for customizing descriptions of MATLAB apps

MATLAB 2012b (8.0)

- Toolstrip that replaces menus and toolbars in MATLAB Desktop
- Apps gallery that presents apps from the MATLAB product family
- Suggested corrections for mistyped functions and variables in the Command Window
- Data import from delimited and fixed-width text files using Import Tool
- Single-file application packaging as a MATLAB App Installer file for inclusion in the apps gallery

MATLAB 2012a (7.14)

- New functions: *integral*, *integral2*, and *integral3*
- Transpose and Sort Variables in the Variable Editor
- *xlsread* Reads XLSX Files on All Platforms
- *VideoWriter* replace all instances of *avifile*
- *whatsnew* won't work anymore and will not be replaced

MATLAB 2011b (7.14)

- Removed function: *bessel*
- Error messages for each function now provide a link to that function's documentation.
- Under certain conditions, as you rename a variable or function, a tooltip opens. The tooltip indicates that MATLAB can rename all instances of the function or variable in the current file.
- The new Spreadsheet Import Tool allows you to select ranges of data and define rules for handling nonnumeric values

MATLAB 2011a (7.12)

- Renaming Files and Folders in the Current Folder Browser Now Reflected in the Editor

MATLAB 2010b (7.11)

- The following functions are now deprecated (but still work): *isstr*, *str2mat*, *strread*, *textread*.
- We can now customize the date format that the *Current Folder* browser and the *Command History* window use to display dates
- New functions *isrow*, *iscolumn*, and *ismatrix*
- The new *VideoWriter* function allows you to create AVI files on all platforms. As an improvement over the *avifile* function, *VideoWriter* can create files larger than 2 GB.
- For consistency with the new *VideoWriter* class, the *mmreader* class is now called *VideoReader*.
- Replace all existing instances of *wavplay* with *audioplayer* function.

MATLAB 2010a (7.10/2010)

- Create zip file from *Current Folder* panel by right click on selected files
- You can now remove folders from the MATLAB path directly from the *Current Folder* panel by right click and selecting *Remove path*
- Tab completion for local variable and subfunctions names while typing scripts

MATLAB 2009b (7.9/2009)

- Start MATLAB using *-nodisplay* (UNIX) or *-noFigureWindows* (Microsoft Windows) is deprecated. Use *-nodesktop* instead.
- During a search in the Help Aa preview of where the search words were found within the page are highlighted.
- In the *Current Folder* you can use the expandable tree structure to view the contents of the current folder. Display the file type in a column.
- At the bottom of the *Current Folder* there is a small "details" that will give you some information about selected MATLAB files.
- You have now a search tool in the address bar to found a file
- MATLAB now supports PDF as the output format for publishing MATLAB code files.

- This version of MATLAB introduces a new usage for the tilde (~) operator. You can also use the tilde operator to specify unused outputs in a function call, or unused inputs in a function definition. This new syntax can help you avoid confusion in your program code and unnecessary clutter in your workspace.
- In previous releases, the Import Wizard imported only the first populated worksheet in an Excel file. The Import Wizard now allows you to specify the worksheet to import.
- New Plot Selector workspace tool that creates graphs of workspace variables

MATLAB 2009a (7.9/2009)

- Many M-Lint Messages Now Extend to Provide an Explanation and Suggested Action
- If you have installed Excel 2007 (or Excel 2003 with the Compatibility Pack) on your Windows system, the `xlswrite` function exports data to XLSX, XLSB, and XLSM formats
- Dynamic Links in Published Documents

MATLAB 2008b (7.8/2009)

- New functions: *clearvars*, *memory*
- There is a new desktop layout when you select *Desktop* > *Desktop Layout* > *Default*
- New function browser (Microsoft Excel like...) tool
- View syntax hints while entering statements
- *Workspace Browser Toolbar* is now configurable
- The new function `randi` returns random integers from a uniform discrete distribution

MATLAB 2008a (7.6/2008)

- On Microsoft Windows platforms, when MATLAB starts, it automatically adds the My Documents/MATLAB directory (or Documents/MATLAB on Windows Vista™) to the top of the MATLAB search path
- Customize the MATLAB desktop and possibility to customize toolbars
- *Array Editor* renamed to *Variable Editor*
- The MATLAB stand-alone editor (`meditor.exe`) is no longer provided. Instead of the stand-alone editor, you can use the MATLAB Editor
- Specify publishing settings, such as an output directory and file format, that you can save and reuse as a group
- New *Data brushing* tool
- With the new *clearvars* function, you can specify which variables you do not want cleared from memory

MATLAB 2007b (7.5/2007)

- You can now change the MATLAB startup directory on Microsoft Windows platforms using the standard shortcut Start in field.
- New desktop layout: All but *Command Window* minimized
- Code folding feature for collapsing and expanding code

- With the *maxNumCompThreads* function, you can either set the maximum number of computational threads to a specific number, or indicate that you want the setting to be done automatically by MATLAB
- P-code files built with MATLAB 7.5 only work on 7.5 or later. They cannot be used with MATLAB 7.4 or earlier versions.
- When you save a figure, all datatips existing in it are saved along with other annotations.

MATLAB 2007a (7.4/2007)

- Introduction of MATLAB for Macintosh (Intel)
- Double-Clicking associated file type in Explorer now opens file in existing session of MATLAB Software
- Minimize, temporarily Display, and Restore panels in the desktop
- For some types of warnings or errors, M-Lint can apply an automatic fix to the code
- You can now publish an M-file function. When publishing an M-file function, you cannot evaluate the code. This feature effectively allows you to save M-file functions to output formats such as HTML or Microsoft Word documents, with formatting

MATLAB 2006b (7.3/2006)

- Introduction of MATLAB for Windows 64-bit
- The *Workspace* browser includes new columns that automatically display results of common statistical calculations: Minimum, Maximum, Range, Mean, Median, Mode, Variance, and Standard Deviation.
- Use the new *File Comparisons* tool to highlight the differences between two files
- The *beta* function, when used with 3 inputs won't work anymore
- The *gamma* function, when used with 2 inputs won't work anymore
- When using *whos* there is a new column in the output called *Attributes* that identifies values that are sparse, complex, global, or persistent.
- When using *whos* the words "array" and "object" have been dropped from items under the Class heading. Items formerly listed as double array or timer object, for example, are now displayed as double, and timer.
- When using *whos* MATLAB no longer includes summary information showing the "Grand total" of elements and bytes at the bottom of the *whos* output display.
- There is an additional field in the structure returned by *whos*. This new field is '*persistent*' and is set to logical 1 for those variables that are persistent, or logical 0 otherwise.
- Using the new *first* option with the *unique* function returns a vector with elements that represent the lowest indices of unique elements in the input vector.
- Save *Compression* and *Unicode* Options Removed. Instead, save now creates a MAT-file with compression and Unicode character encoding by default.
- The function definition line in a function M-file no longer requires commas separating output variables.
- Printing MATLAB figures has become easier as a result of combining the *Page Setup*, *Print Setup*, and *Print Preview* dialogs into one tabbed *Print Preview* dialog.
- If your mouse has a center scroll wheel, you can use it to zoom in and out of axes on charts, as well as by clicking and/or dragging

- Data Cursor text can now be programmatically modified

MATLAB R14SP3 (7.1/2005)

- The behavior of MATLAB software when started on Microsoft Windows platforms with the *-nodesktop* option has changed. The MATLAB Command Window no longer displays a menu bar or toolbar
- The Editor/Debugger now supports a horizontal or vertical split screen for displaying two different parts of the same document at once.
- You can set a preference to highlight the current line, that is, the line with the caret
- New function *hypot* for the square root of sum of squares
- New function *mod* to find the most frequent values in sample
- The *rand* function supports Mersenne-Twister algorithm
- The *datestr*, *datenum*, and *datevec* functions now support time specification in milliseconds

MATLAB R14SP2 (7.0.4/2005)

- The Installer honors the default installation folder for Windows software, which on most machines is *Program Files*.
- Go directly to a subfunction or nested function within an M-file using the enhanced *Go To* dialog box
- In MATLAB versions prior to R14, date values read into MATLAB from an Excel spreadsheet using *xlsread* were always imported as numeric date values. The R14 and later releases of MATLAB import dates in the format in which they were stored in the Excel file. Dates stored in string or date format are now imported as strings by *xlsread*. Dates stored in numeric format are imported as numeric date values.
- You can display MATLAB output using two new formats: *short eng* and *long eng*.
- The *imwrite* function now supports exporting image data in Graphics Interchange Format (GIF)

MATLAB R14SP1 (7.0.1/2004)

- Nothing new interesting in my point of view...

MATLAB R14 (7.0/2004)

- A 64-bit version of MATLAB is now available for Linux platforms based on AMD64 and Intel EM64T processors. A 64-bit processor provides a very large amount of available memory, up to 2^{64} bytes = 18,446,744,073,709,552, 000 bytes (16 exabytes), enabling you to store a very large amount of information.
- The largest contiguous block of memory under Windows XP to approximately 1.5 GB, equivalent to 180 million double precision values
- New functions: *addtodate*, *intmax*, *intmin*, *isfloat*, *isinteger*, *isscalar*, *isvector*, *recycle*, *restoredefaultpath*, *strtrim*, *textscan*, *xlswrite*, *isstrprop*
- New *feature('memstats')* to see available memory on the system
- Now MATLAB is case-sensitivity in functions and directory names for MS Windows and UNIX (was only case sensitive on UNIX before).

- Anonymous functions give you a quick means of creating simple functions without having to create M-files each time. You can construct an anonymous function either at the MATLAB command line or from within another function or script.
- When converting between serial date numbers, date vectors, and date strings with the *datenum*, *datevec*, and *datestr* functions, you can specify a format for the date string
- You can now specify the range to import with *xlsread*
- MATLAB now includes a function, named *mmfileinfo*, that returns information about the contents of a multimedia file. The file can contain audio data, video data, or both
- From within MATLAB, you can connect to an FTP server to perform remote file operations.
- MATLAB can now consume Simple Object Access Protocol-based (SOAP) Web services with the *createClassFromWSDL* function.

MATLAB R13 (6.5/2000)

- MathWorks products in Release 13 no longer support Microsoft Windows 95.
- This release introduces the Profiler, a new graphical user interface for assessing the performance of your M-files so you can make performance improvements
- Display of line number, column number, and current function in the Editor
- Colormap editor enables interactive colormap editing
- New text properties provide control over the text background
- Redesigned property editor accesses graphics properties with interactive editor

MATLAB R10 (5.2/1998)

- New MATLAB language functions, implementing features such as try/catch error handling
- Support for two new ActiveX technologies: ActiveX control containment and ActiveX Automation client capabilities, so that now MATLAB can both control and be controlled by other ActiveX components
- Support for OpenGL rendering to improve performance dramatically for many visualization application
- Enhanced visualization features, including enhanced camera control, simplified placement of Light objects, tighter and more consistent control of graphics object hiding.

MATLAB R9 (5.1/1997)

- New scatter plot functions

MATLAB R8 (5.0/1996)

- Can't use logical (binary) vector anymore to subselect a vector or matrix
- Integrated M-file editor
- Visual M-file debugger
- M-file performance profiler
- Search path browser/editor
- Workspace browser

- Web-based online Help Desk/documentation viewer
- GUI builder
- New multidimensional arrays
- New User-definable data structures
- New cell arrays: multitype data arrays
- New Character arrays: two bytes per character
- Switch/Case statements
- Import/Export images functionality
- Greeks symbols available in charts
- Dual axis plots
- Three dimensional quiver, ribbon and stem plots
- Pie charts and 3D bar charts
- Extended curve make symbol family

Cite MATLAB

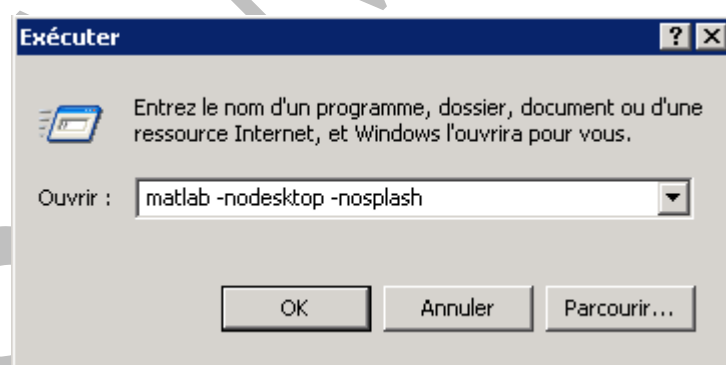
To cite MATLAB (and in this case a toolbox) you can use this:

MATLAB and Statistics Toolbox Release 2012b, The MathWorks, Inc., Natick, Massachusetts, United States.

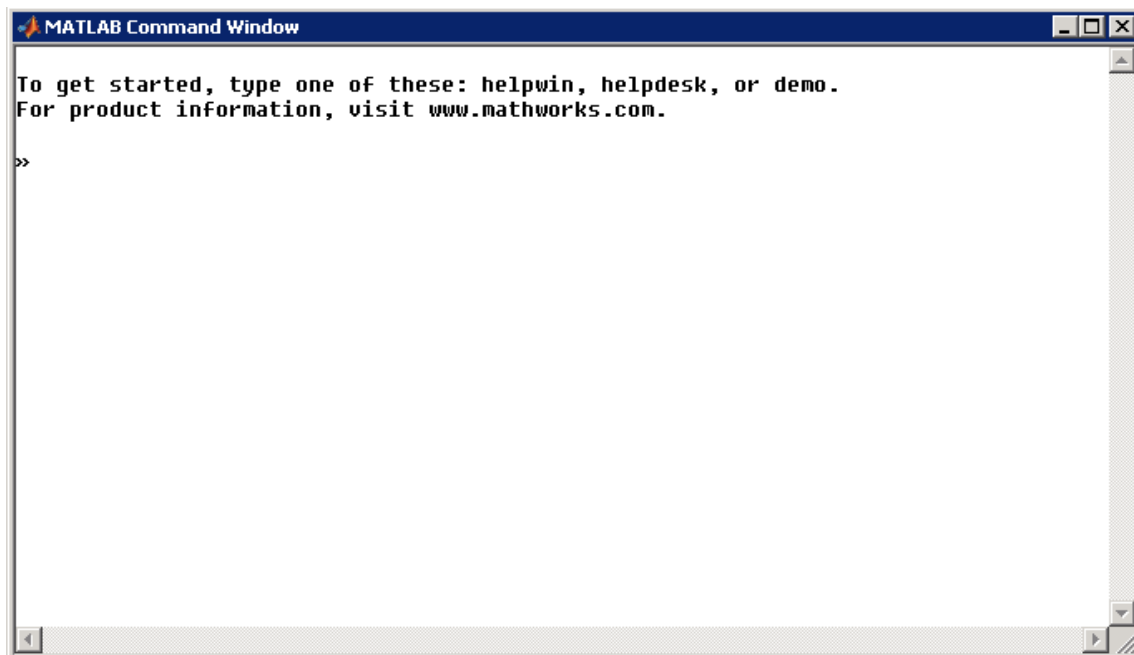
User Interface

The goal of this chapter is to learn the basic of MATLAB without using any Toolboxes!

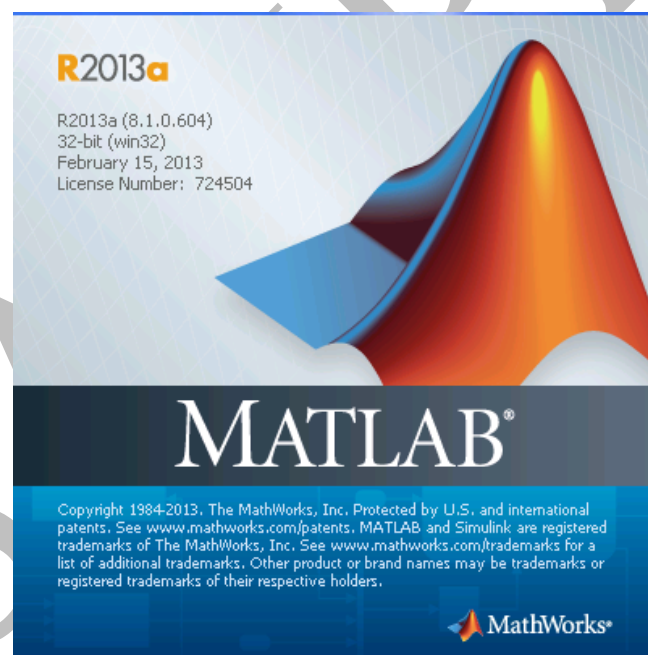
To open MATLAB without the GUI in the MS Windows command prompt type the following:



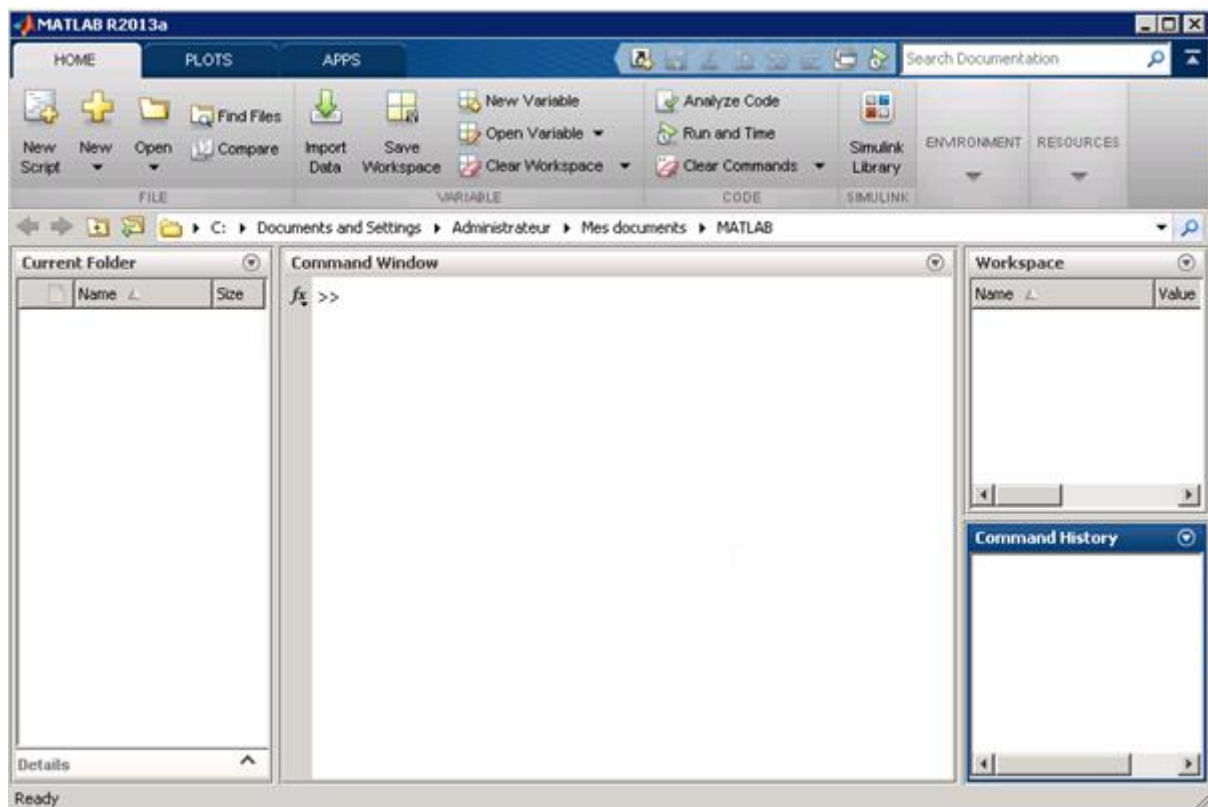
You will get the following old style robust scientific (UNIX like) interface:



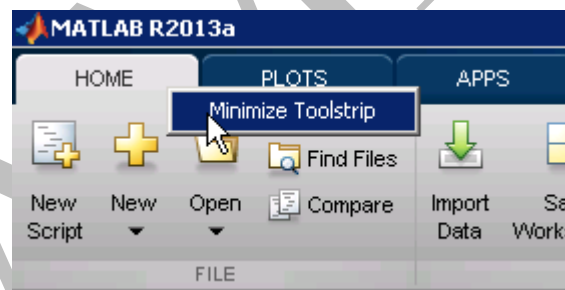
If you run MATLAB by the standard way you will first get the splash screen during a few seconds (here MATLAB 2013a splash screen):



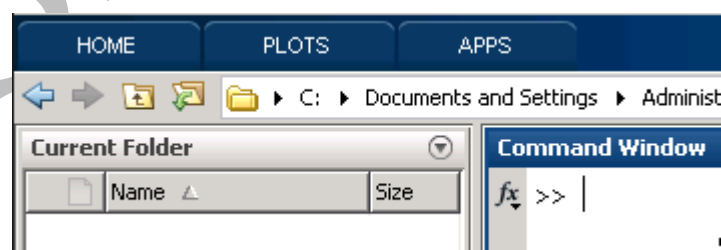
And after you will see the MATLAB GUI open:



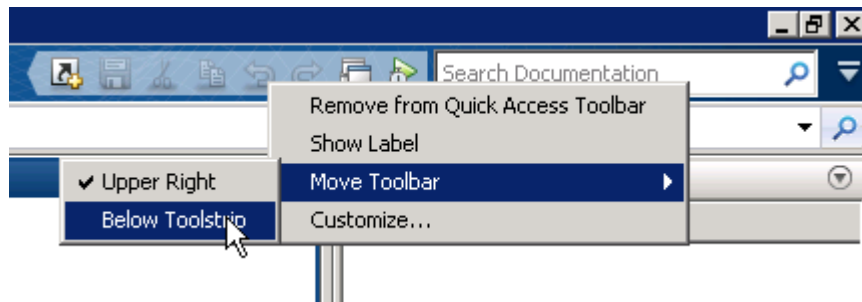
You can minimize the ribbons like in Microsoft Office by double clicking on the register or by making a right click:



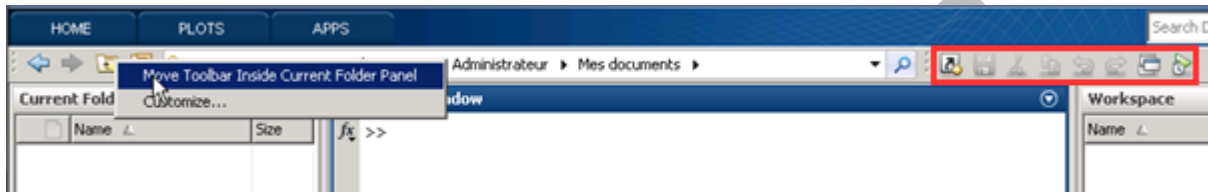
The result will be:



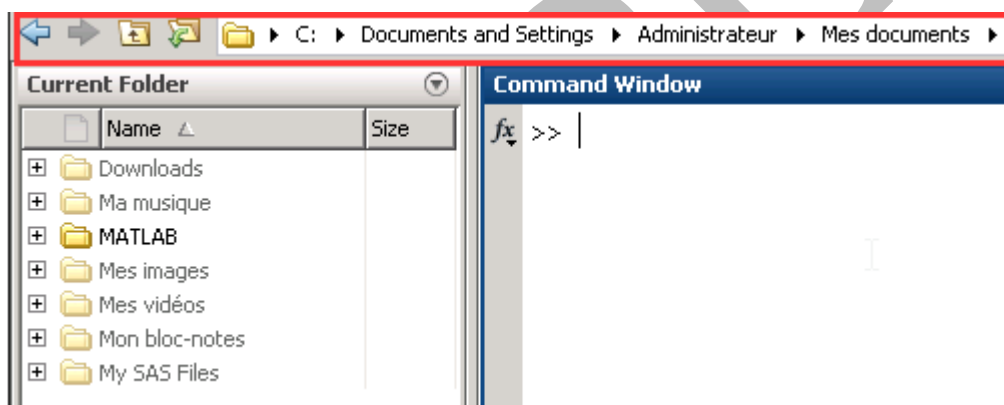
You can also change the position of the **Quick Access Toolbar**:



Or take it back to its previous position:



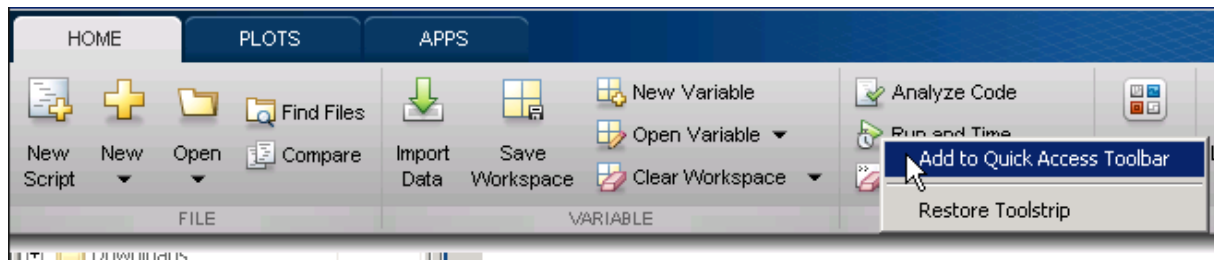
As you can see MATLAB has a **Path browser toolbar**:



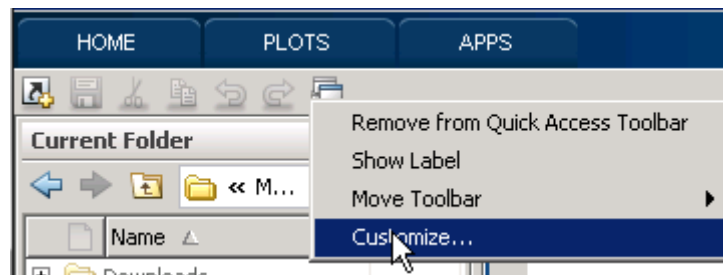
With a right click you can move it into the **Current Folder Panel**:



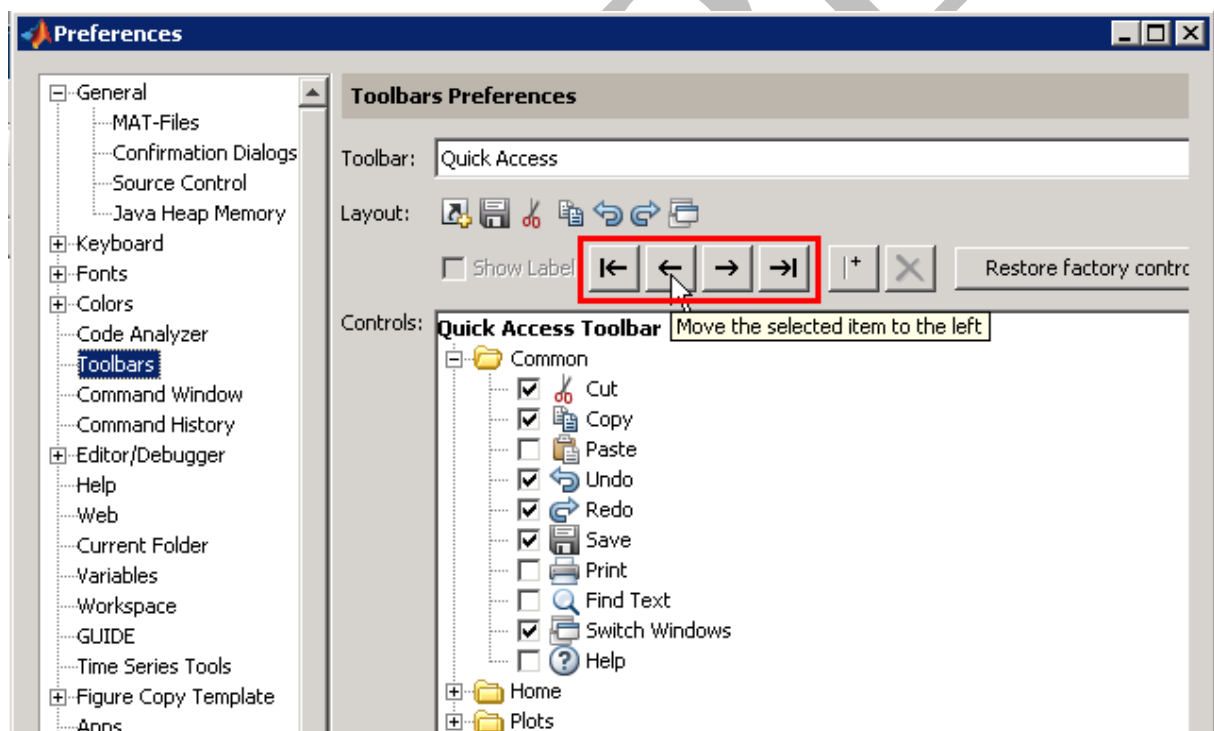
Like in Microsoft Office you can move you favorites buttons to the Quick Access Toolbar by right click and selecting **Add to Quick Access Toolbar**:



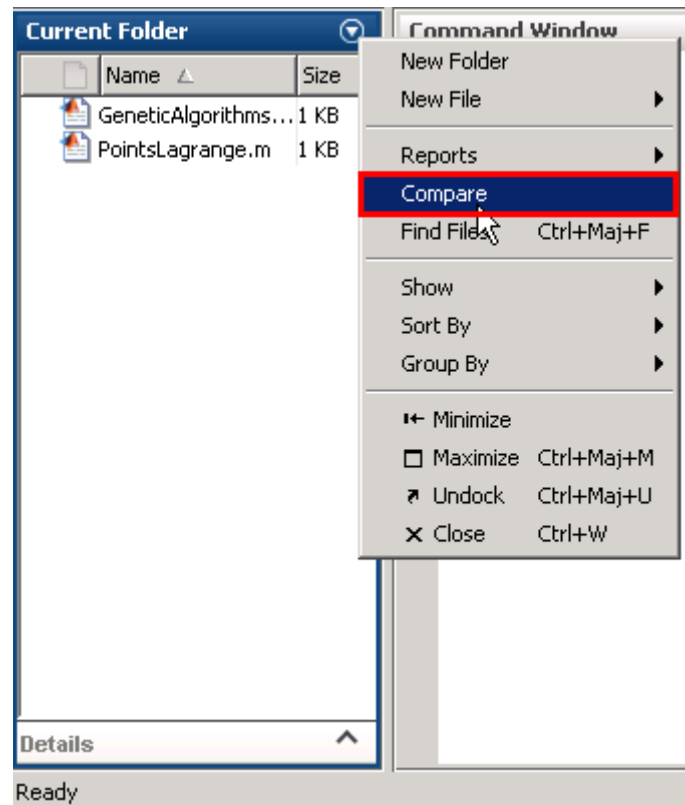
You can also like in Microsoft Office customize the Quick Access Toolbar:



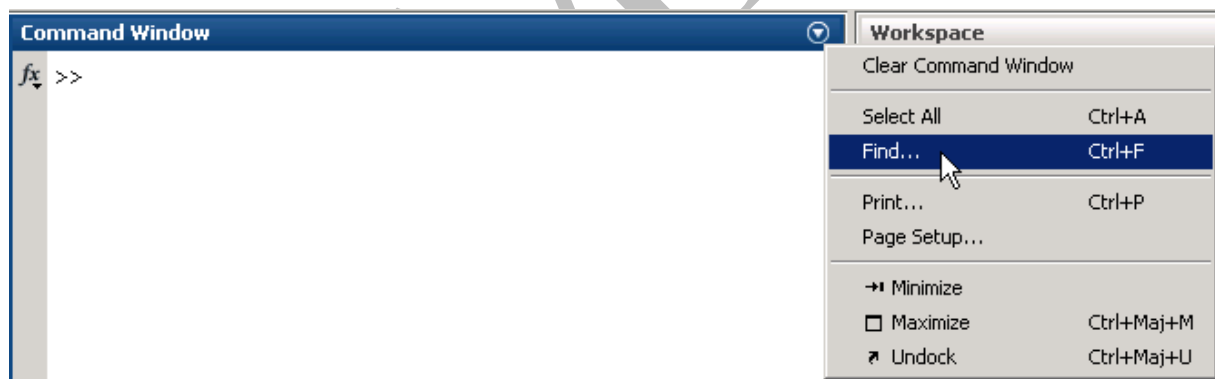
And you will get a windows that looks very similar to the Microsoft Office one:



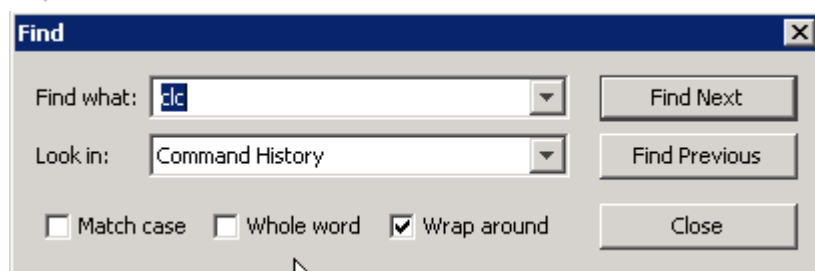
Trough the **Current Folder** panel you can if you wish to compare two script files when you click on the small arrow on the top right corner:



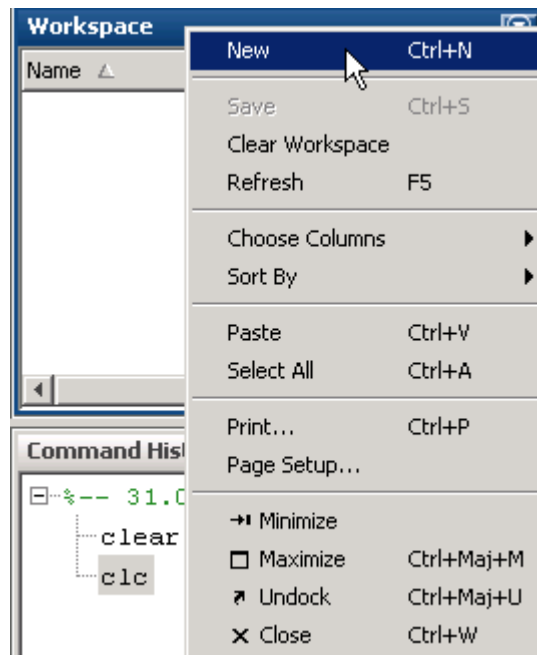
If you click on the small arrow of the top right corner of the **Command Window** you can **Find...** (search) previous type commands. This is very useful!:



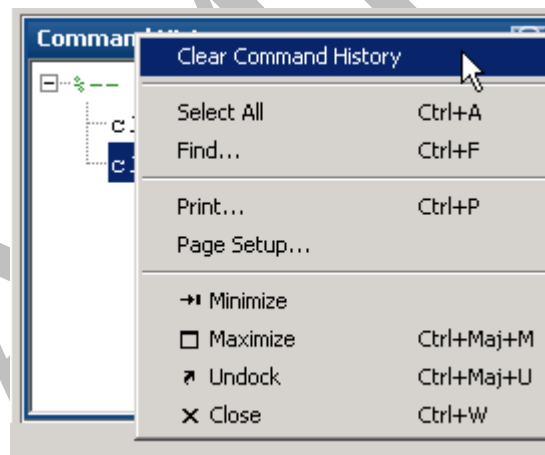
You will get the following famous dialog box on your screen where you can type some keywords to search a previous typed command:



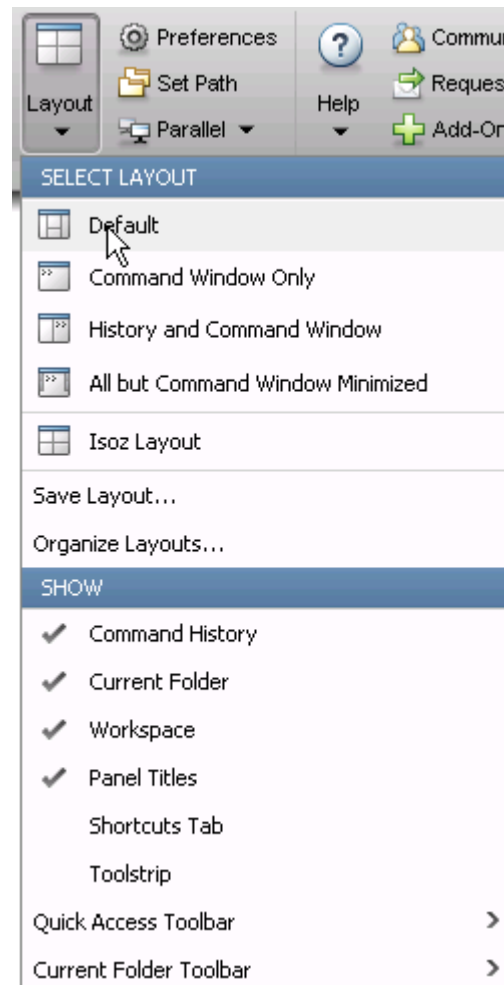
Trough the **Workspace Panel** small icon you can also create new script files (most people use instead **Ctrl+N** for this purpose):



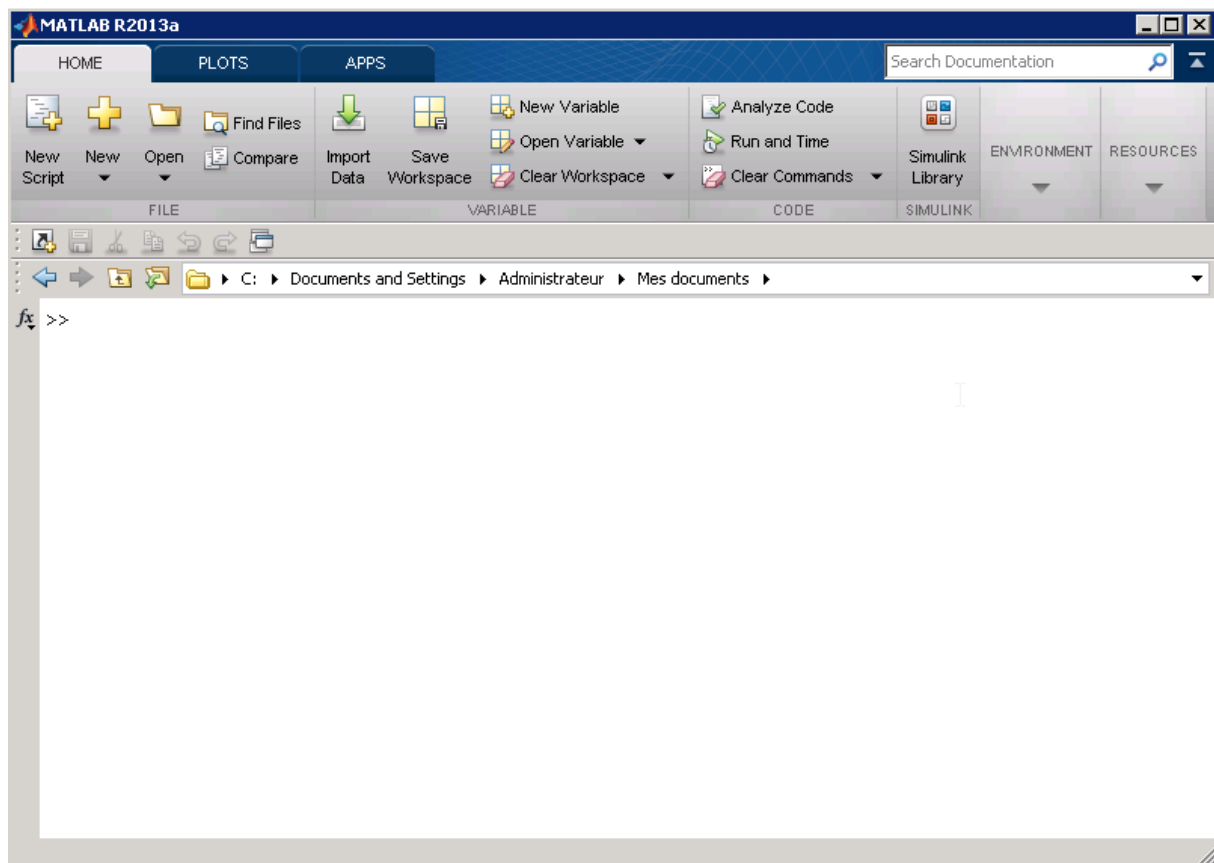
In the **Command History Panel** you can clear the history if you wish to:



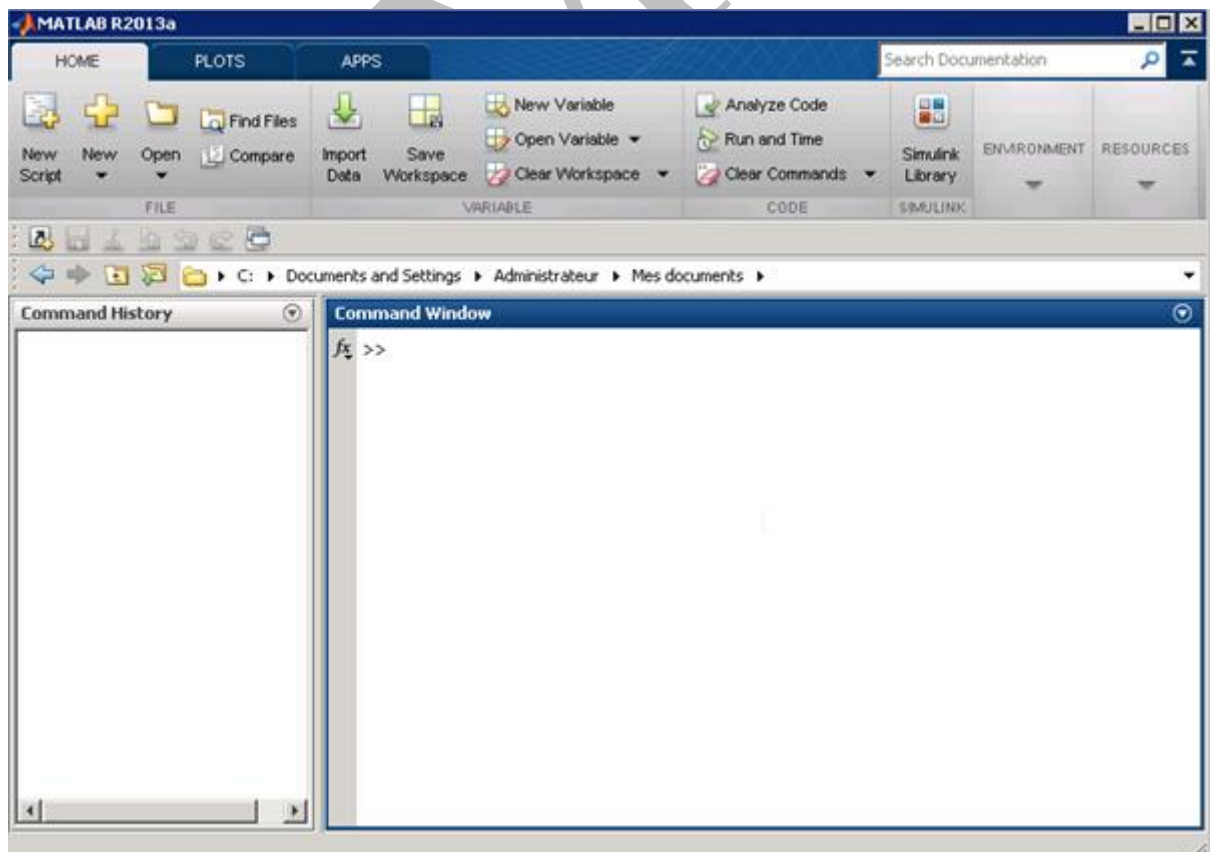
You can also choose between different presentation layout of the GUI:



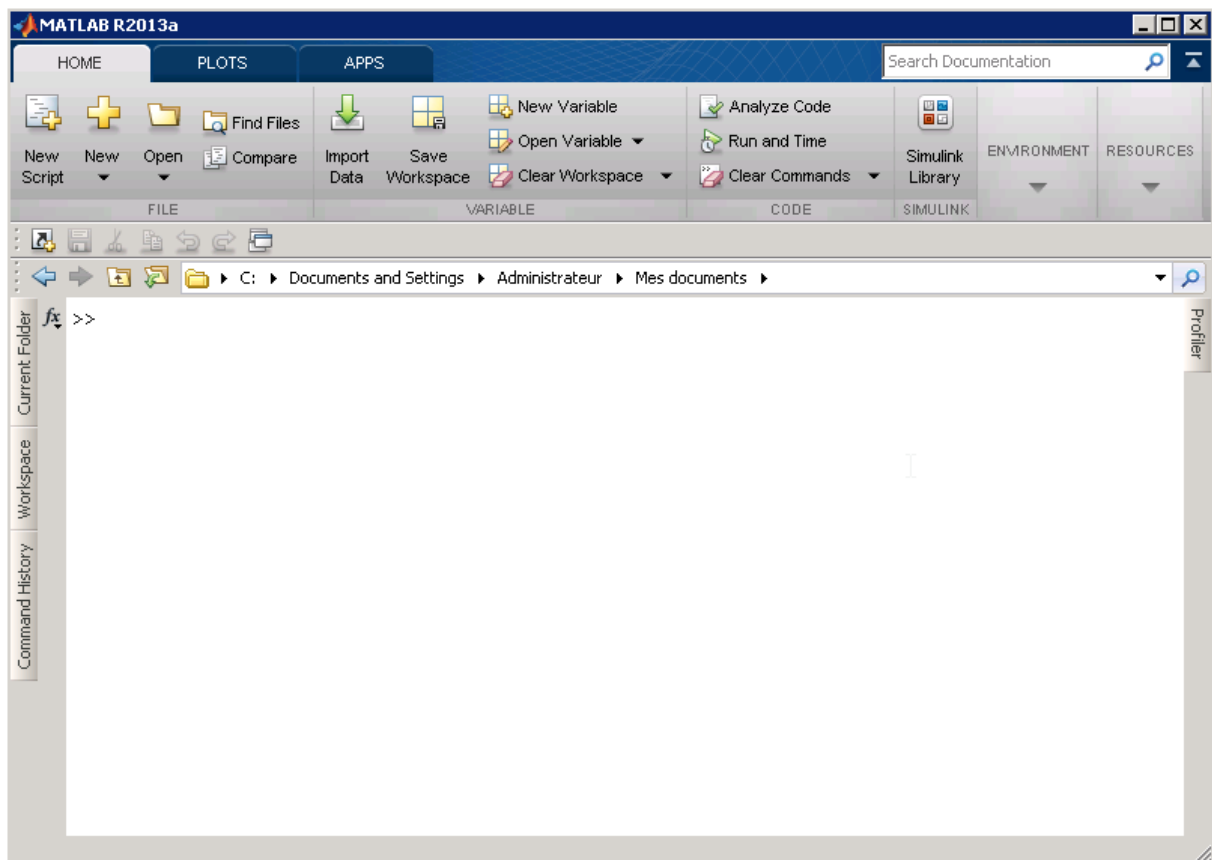
Here below you have for example the **Command Window Only** layout visible:



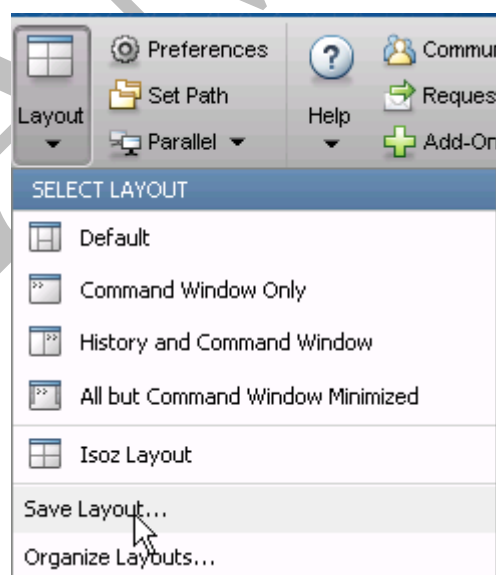
And here the **History and Command Window** layout:



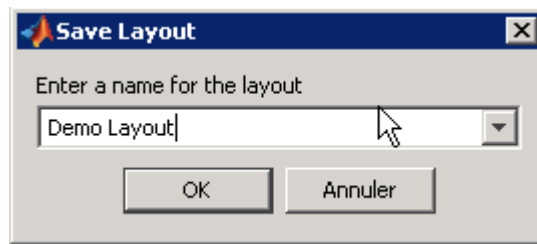
Or the **All but Command Window minimized**:



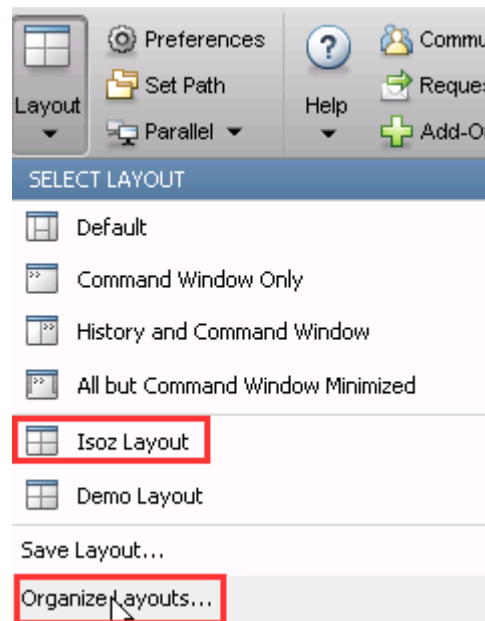
You can also save you customized layout because you can drag & drop the panels with the mouse. For this click on **Save Layout...**:



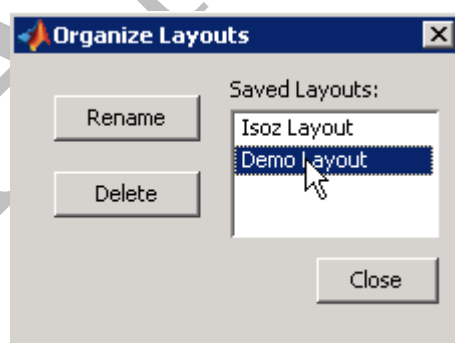
The software will ask you to give a name for the layout:



Once saved you will see your layout in the **Layout** list:



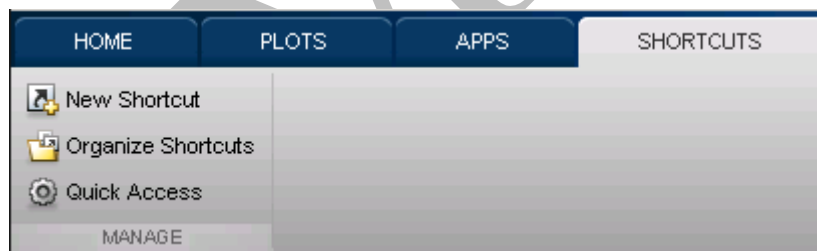
If you click on **Organize Layouts...** you will have the possibility to rename or remove one of the Layout you have created:



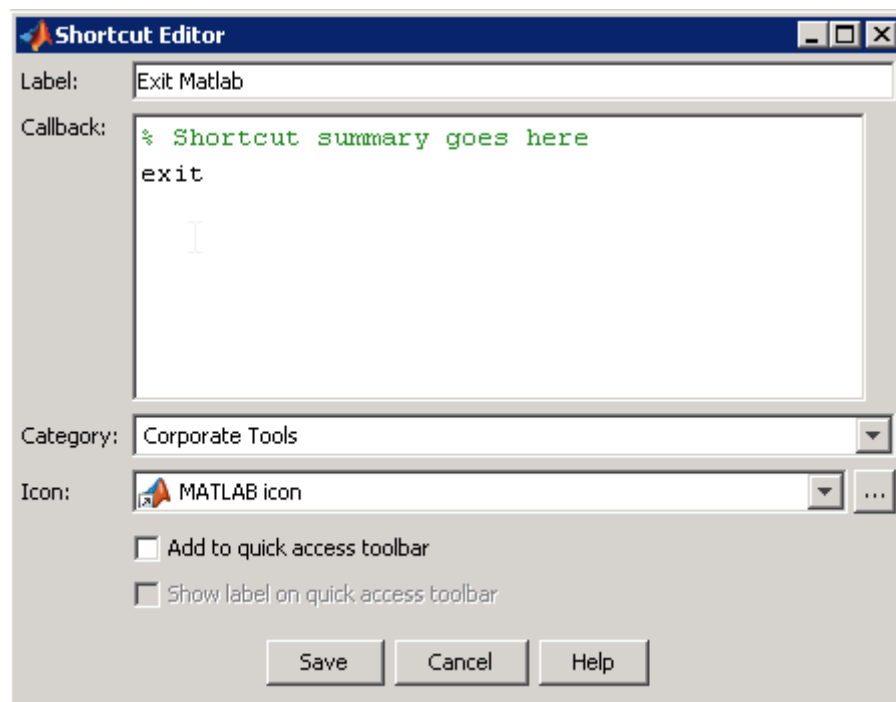
MATLAB has a hidden tab (register) that allows you to create you own shortcuts to activate it click on **Shortcut Tab** in the **Layout** button:



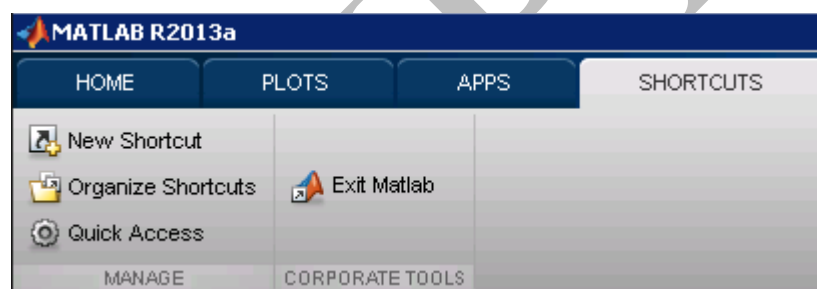
Then you will see a new tab as following:



To create a new shortcut just click on **New Shortcut...**:



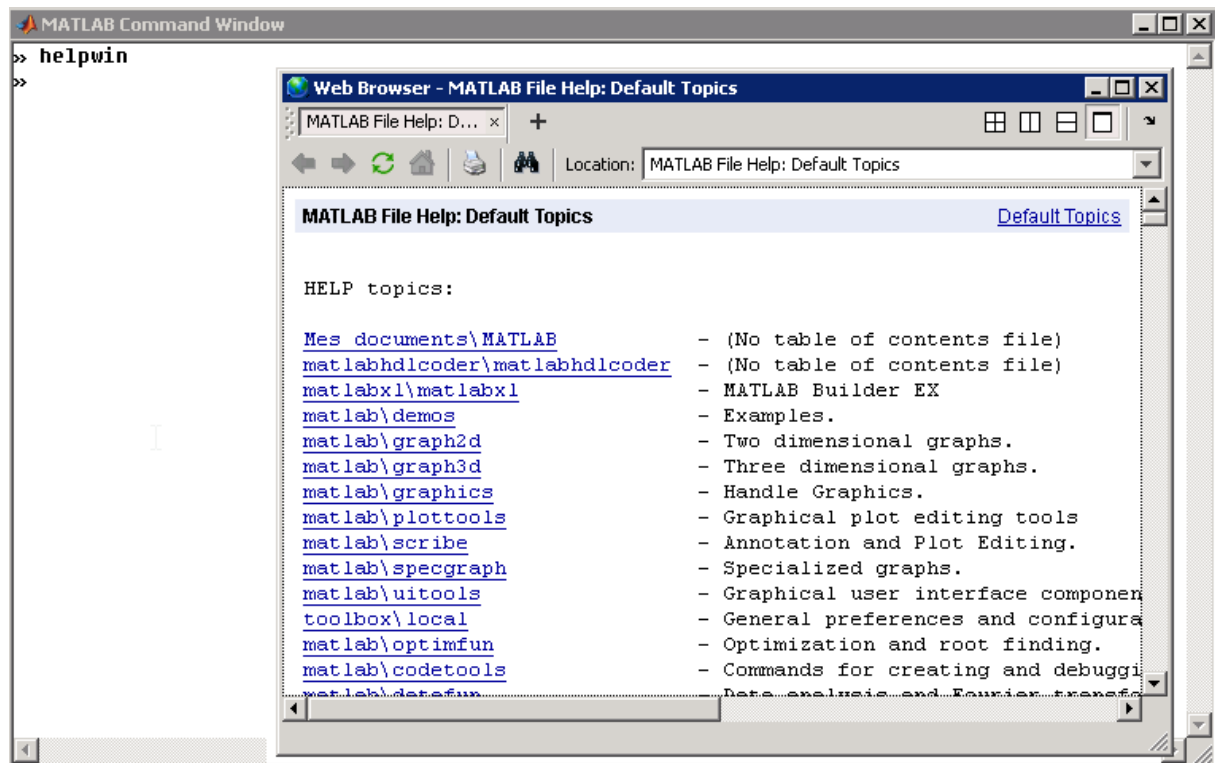
and type the informations you want and click **Save**. Then you will get:



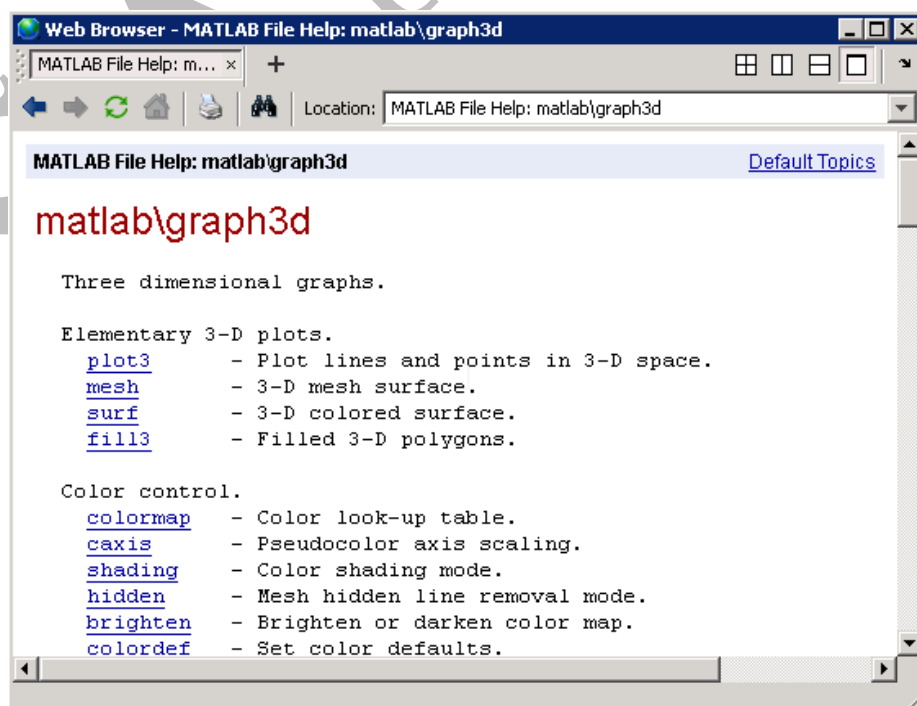
Use Help

Using the help in MATLAB is very important because there is more than 8'000 functions and you can't remember and now everything.

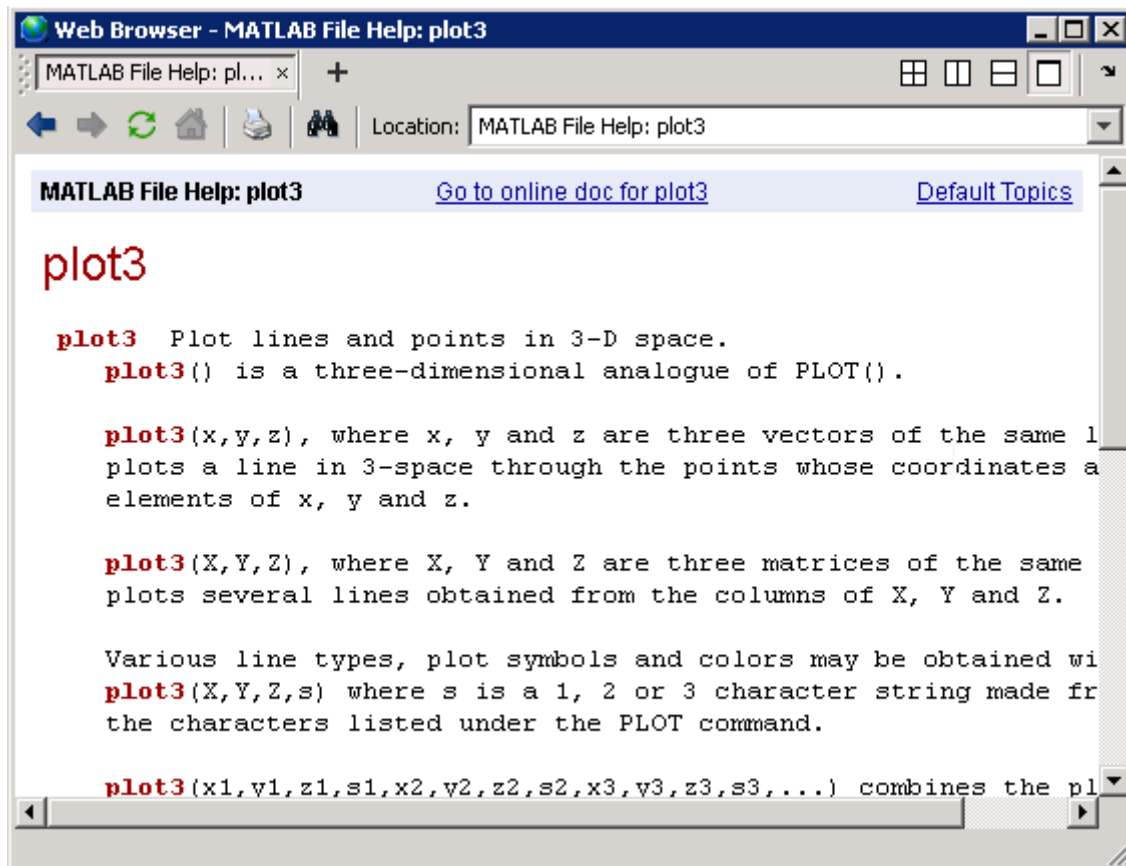
To open the classic help window use the command **helpwin**:



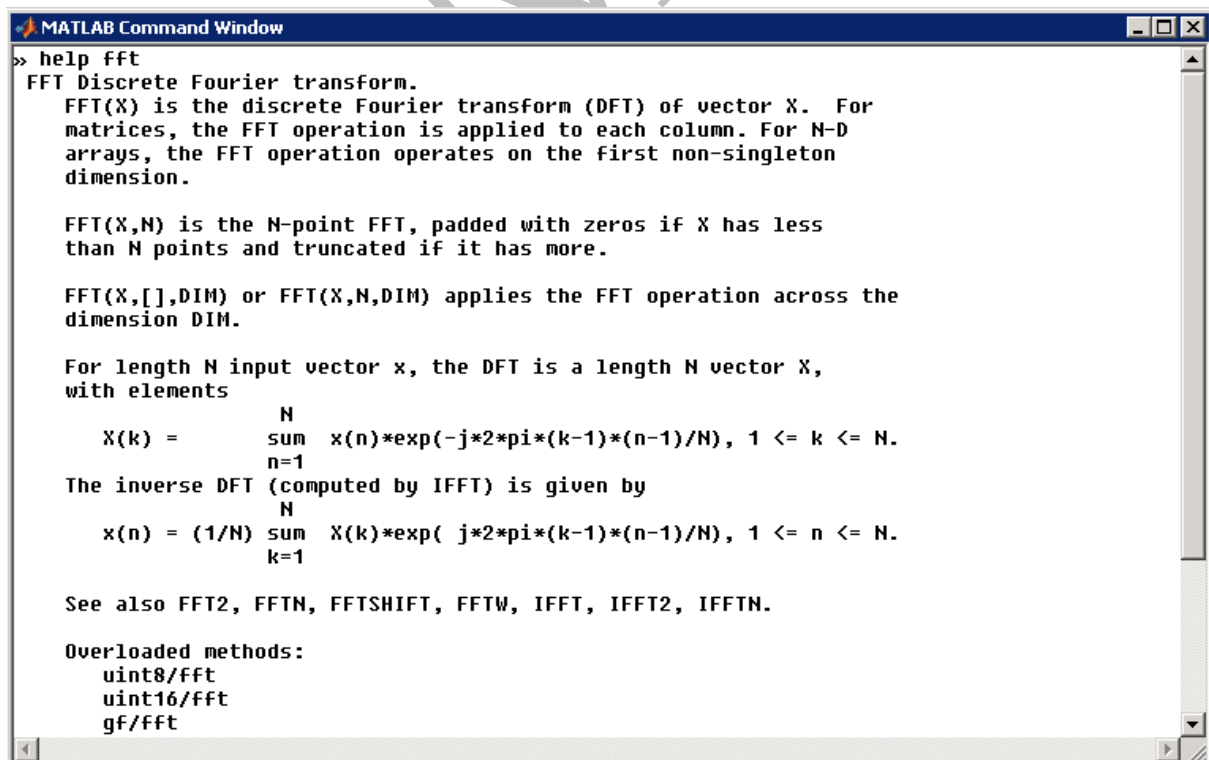
And if you click on one of the subjects you will get something like this:



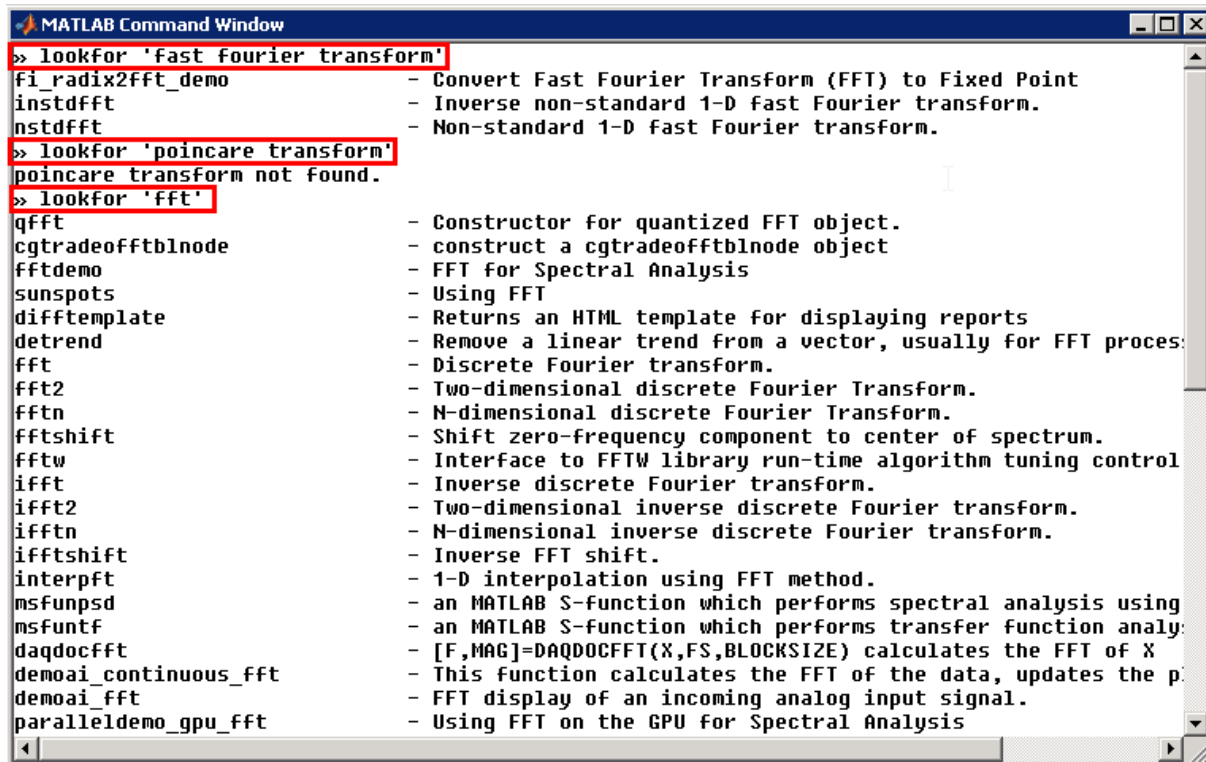
And if you click once again on one of the sub-subjects you will get something like this:



In reality the best way to get a quick answer and help is to use the command **help**:



And if you don't know the name of what you are looking for you can use the commande **lookfor**:

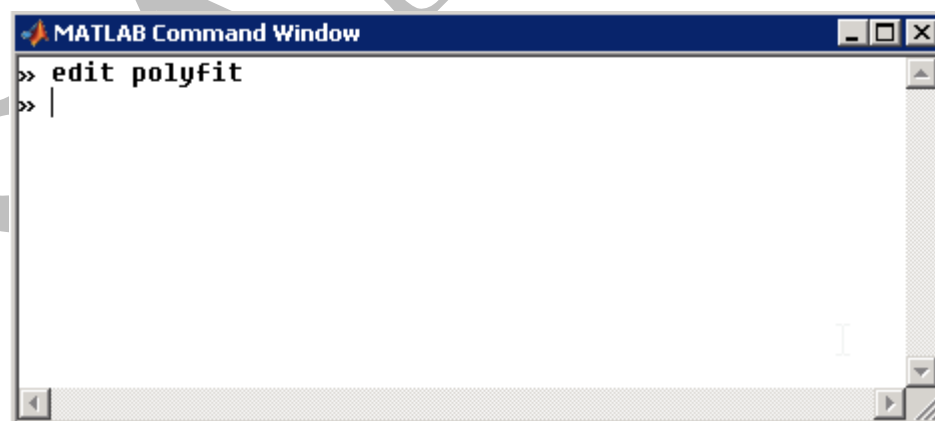


```

MATLAB Command Window
>> lookfor 'fast fourier transform'
fi_radix2fft_demo      - Convert Fast Fourier Transform (FFT) to Fixed Point
instdfft              - Inverse non-standard 1-D fast Fourier transform.
nstdfft               - Non-standard 1-D fast Fourier transform.
>> lookfor 'poincare transform'
poincare transform not found.
>> lookfor 'fft'
qfft                  - Constructor for quantized FFT object.
cgtradeofftblnode    - construct a cgtradeofftblnode object
fftdemo              - FFT for Spectral Analysis
sunspots             - Using FFT
difftemplate         - Returns an HTML template for displaying reports
detrnd               - Remove a linear trend from a vector, usually for FFT proces
fft                  - Discrete Fourier transform.
fft2                 - Two-dimensional discrete Fourier Transform.
fftn                 - N-dimensional discrete Fourier Transform.
fftshift             - Shift zero-frequency component to center of spectrum.
fftw                 - Interface to FFTW library run-time algorithm tuning control
ifft                 - Inverse discrete Fourier transform.
ifft2                - Two-dimensional inverse discrete Fourier transform.
ifftn                - N-dimensional inverse discrete Fourier transform.
ifftshift            - Inverse FFT shift.
interpfft            - 1-D interpolation using FFT method.
msfunpsd             - an MATLAB S-function which performs spectral analysis using
msfuntf              - an MATLAB S-function which performs transfer function analy
daqdocfft            - [F,MAG]=DAQDOCFFT(X,FS,BLOCKSIZE) calculates the FFT of X
demoai_continuous_fft - This function calculates the FFT of the data, updates the p
demoai_fft           - FFT display of an incoming analog input signal.
paralleldemo_gpu_fft - Using FFT on the GPU for Spectral Analysis
  
```

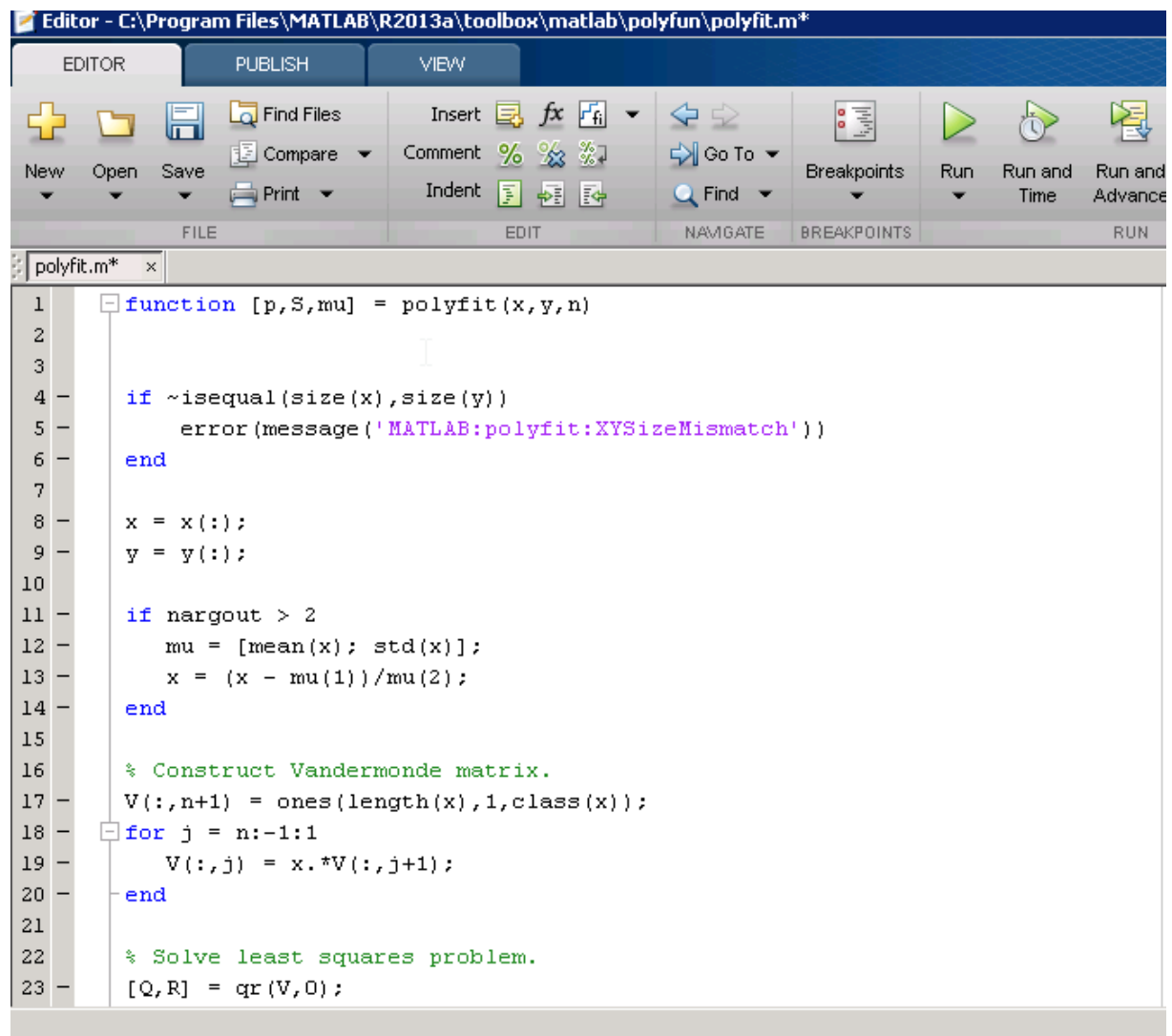
Edit a native function source code (if available)

The source code of some MATLAB functions are available. In some cases this is especially useful so you can made modifications to perform or adapt the existing script to your needs (practice to use only in case of emergency solution!):



```

MATLAB Command Window
>> edit polyfit
>> |
  
```



The image shows the MATLAB Editor window for the file `polyfit.m`. The window has a title bar with the path `C:\Program Files\MATLAB\R2013a\toolbox\matlab\polyfun\polyfit.m*`. Below the title bar are three tabs: **EDITOR**, **PUBLISH**, and **VIEW**. The **EDITOR** tab is active, showing a toolbar with icons for New, Open, Save, Find Files, Compare, Print, Insert, Comment, Indent, Go To, Find, Breakpoints, Run, Run and Time, and Run and Advance. The main editing area displays the following MATLAB code:

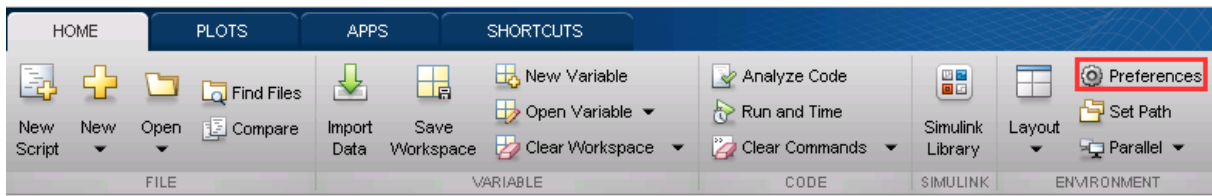
```
1 function [p,S,mu] = polyfit(x,y,n)
2
3
4 if ~isequal(size(x),size(y))
5     error(message('MATLAB:polyfit:XYSIZEMismatch'))
6 end
7
8 x = x(:);
9 y = y(:);
10
11 if nargin > 2
12     mu = [mean(x); std(x)];
13     x = (x - mu(1))/mu(2);
14 end
15
16 % Construct Vandermonde matrix.
17 V(:,n+1) = ones(length(x),1,class(x));
18 for j = n:-1:1
19     V(:,j) = x.*V(:,j+1);
20 end
21
22 % Solve least squares problem.
23 [Q,R] = qr(V,0);
```

Most asked user preferences

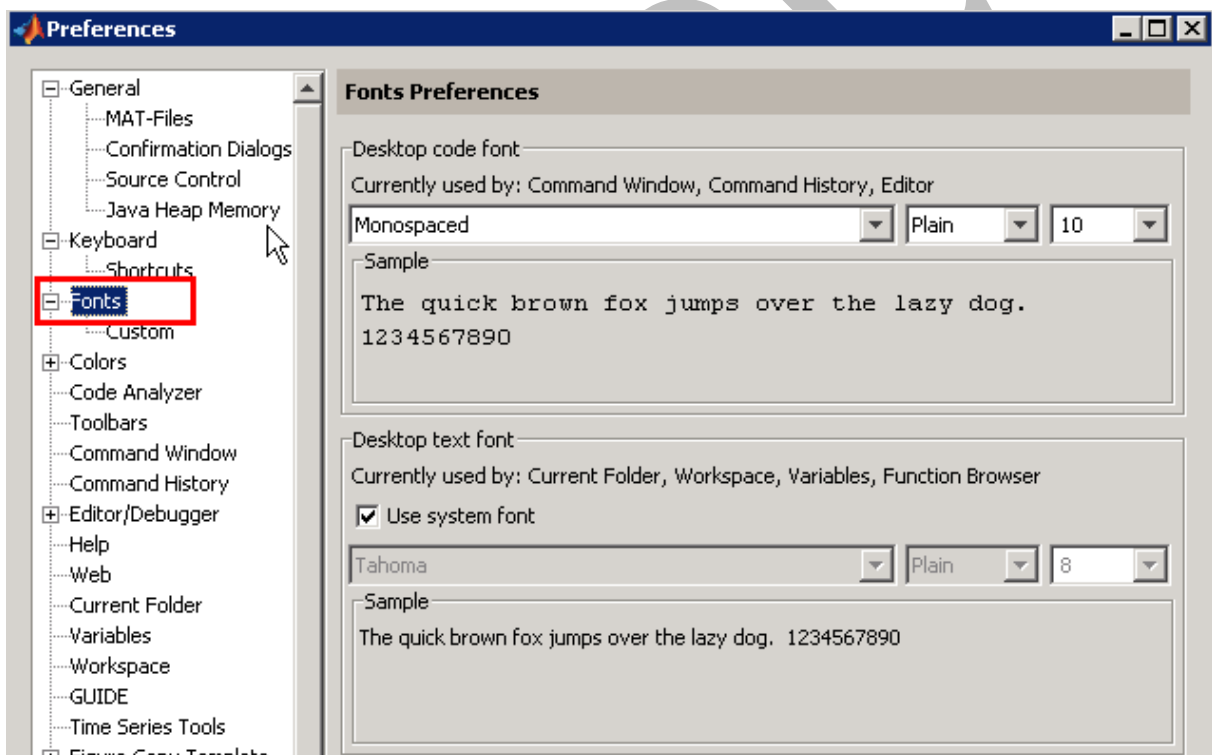
Here we would like to show you only a few MATLAB preferences (system settings) asked by large number of practitioners.

The first setting is how to change the font in the Command Window, Command History and Editor.

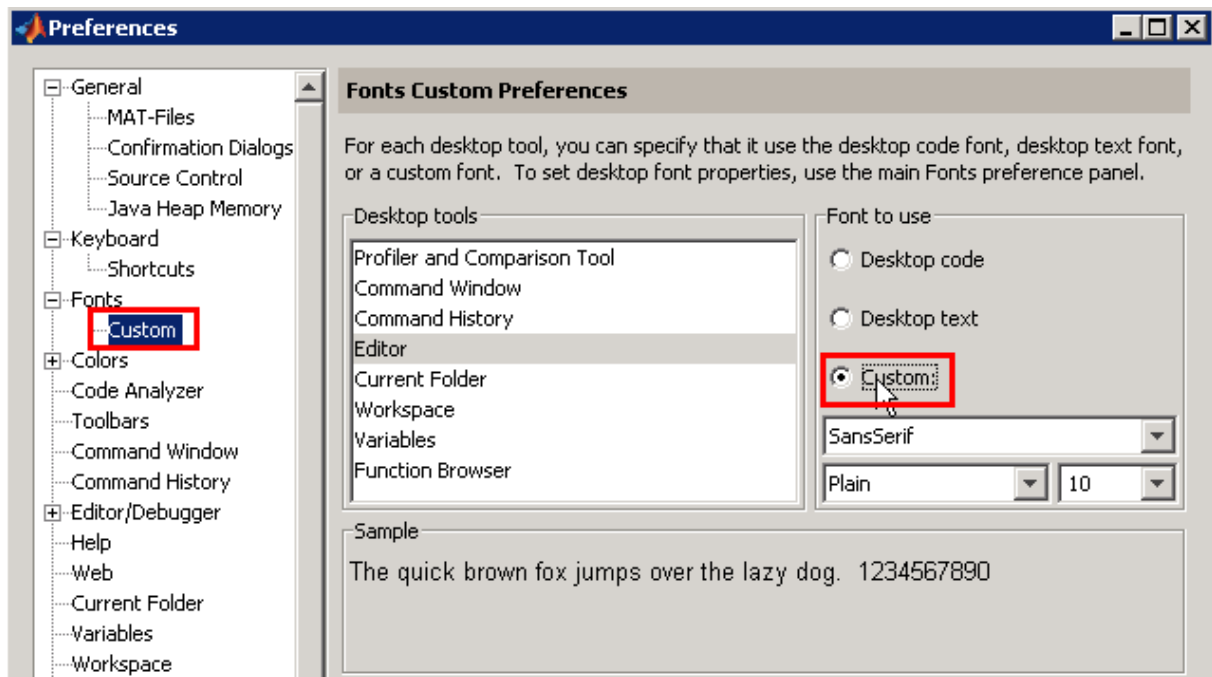
First you go in the tab **Home** and you click on **Preferences**:



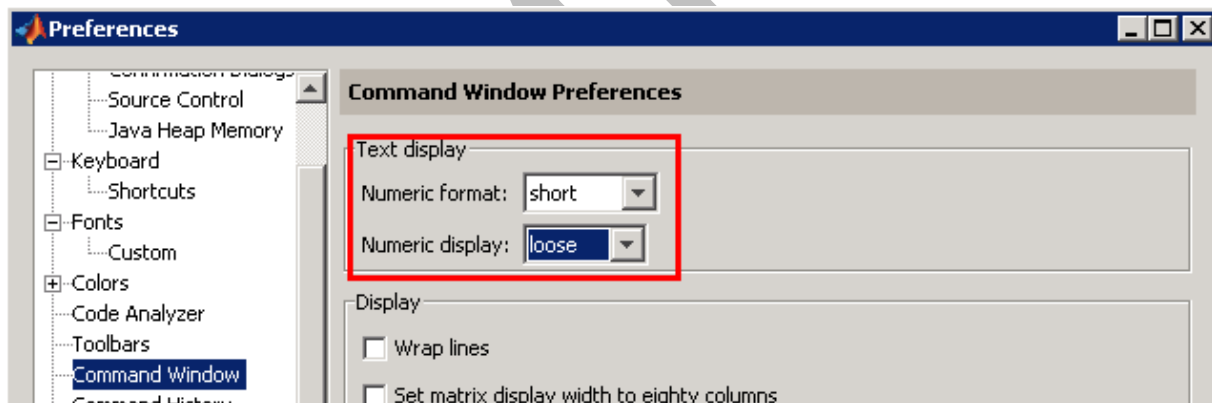
And when the properties windows opens select the **Fonts** section and change what you want the way you want:



If you want to select a special font apart for the Editor you have to click on the **Custom** section inside of **Fonts** as you can see below:



Another annoying point in MATLAB editor for a lot of people is the default display of the **Command Window**. By default format are short and the numeric display is loose but you can change this to moore accurate and optimal choices:

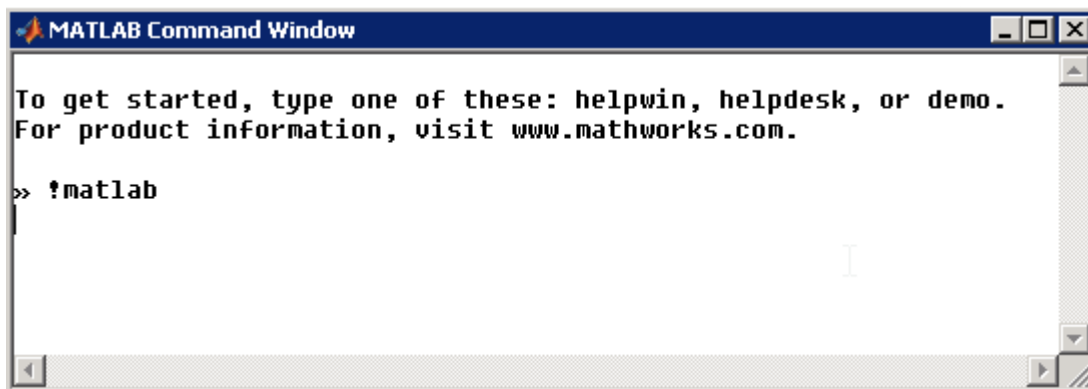


General commands

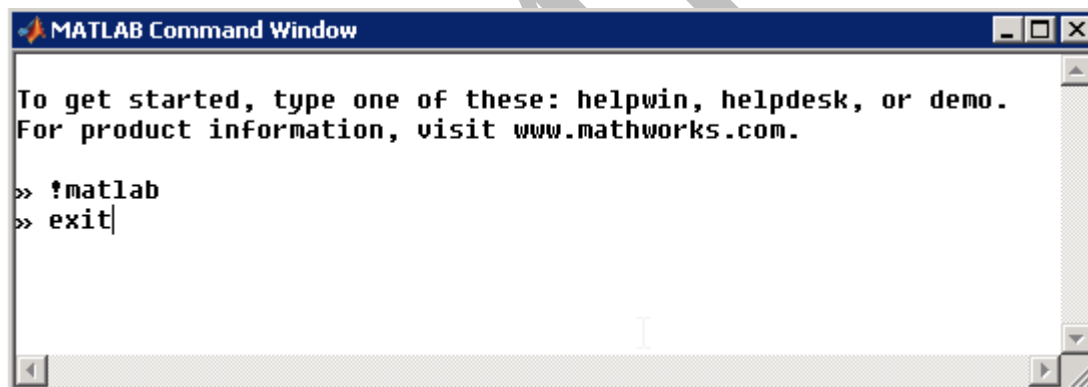
Here we will just take a look to some very general MATLAB commands that are a must know!!

Execute system commands

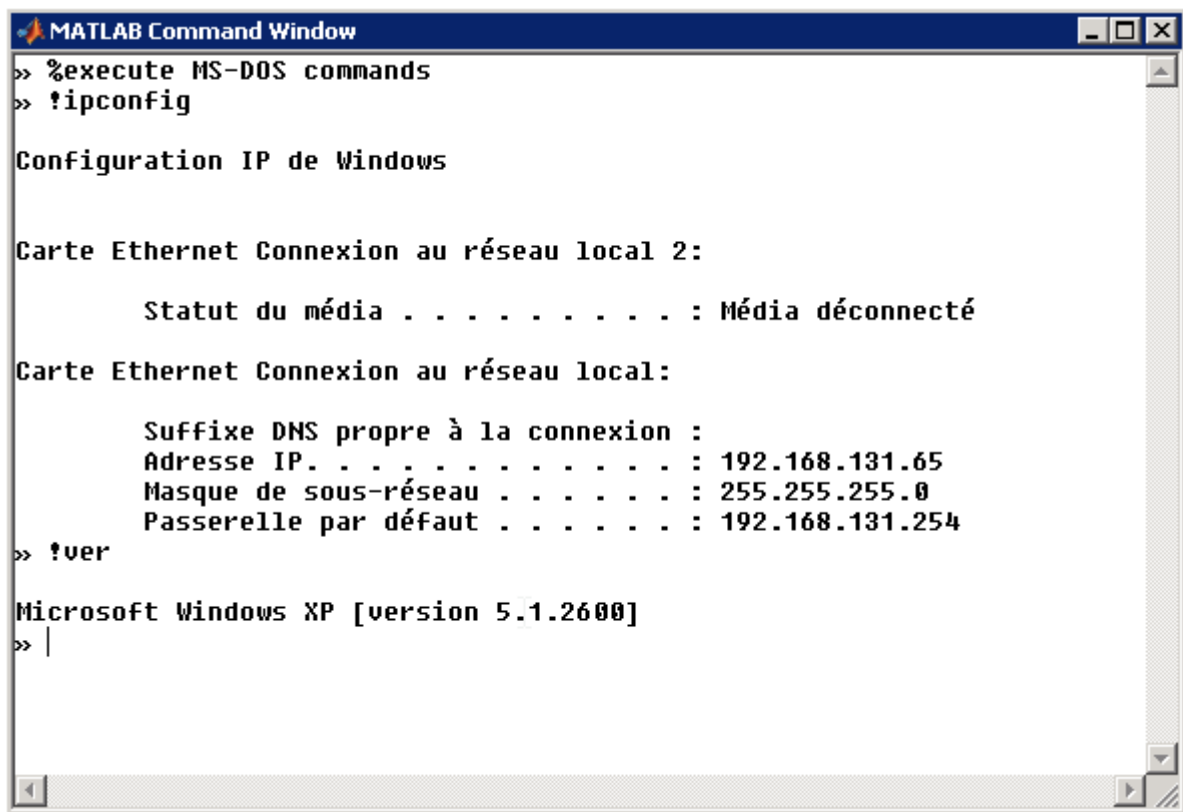
First if you write **!MATLAB** from the old style Command Window you will see the full MATLAB GUI that will open:



You can also close the Command Window or event the MATLAB GUI by typing **exit**:



From the Command Window you can type most MS-DOS commands if you write a exclamation mark ! before the standard MS-DOS command:



```
MATLAB Command Window
>> %execute MS-DOS commands
>> !ipconfig

Configuration IP de Windows

Carte Ethernet Connexion au réseau local 2:

    Statut du média . . . . . : Média déconnecté

Carte Ethernet Connexion au réseau local:

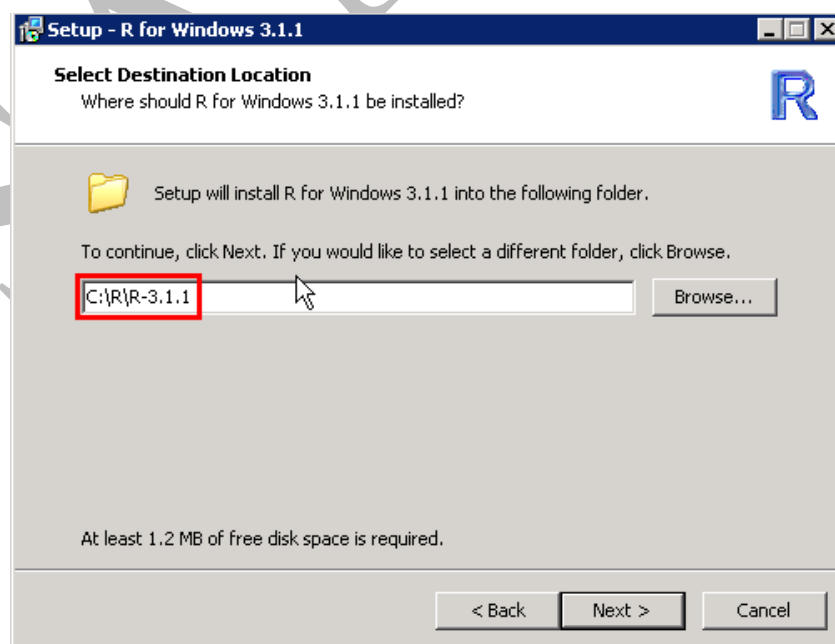
    Suffixe DNS propre à la connexion :
    Adresse IP. . . . . : 192.168.131.65
    Masque de sous-réseau . . . . . : 255.255.255.0
    Passerelle par défaut . . . . . : 192.168.131.254

>> !ver

Microsoft Windows XP [version 5.1.2600]
>> |
```

Call R from MATLAB

With such system commands you can call R software from MATLAB and retrieve results. For this purpose first install R:

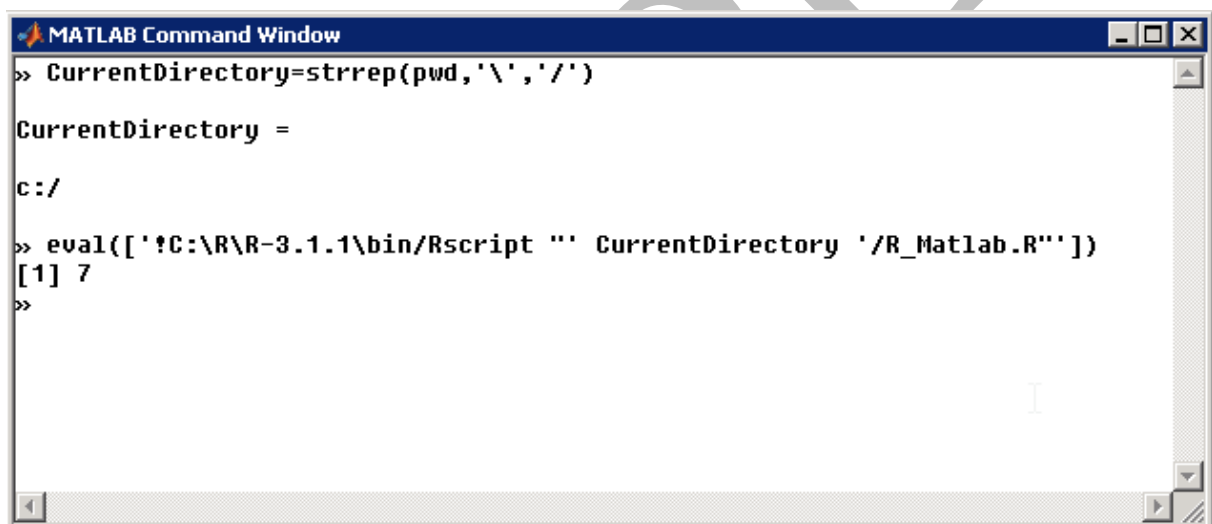


Avoid the usage of spaces in the installation folder!

Once this is done create for example the following small *.R script and take care to save it (for this particular example) in the active MATLAB workpath:

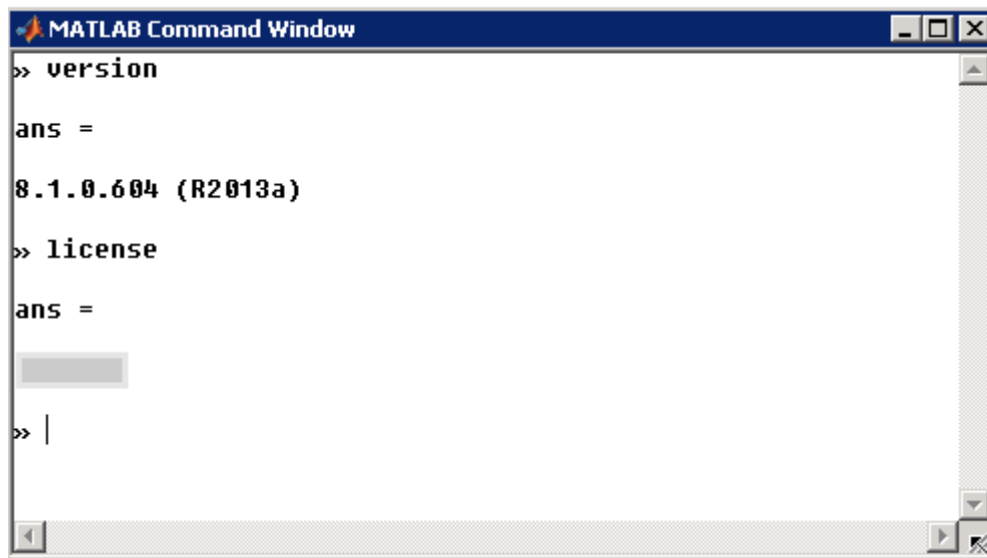


Then you can call R from MATLAB simply by using the following command:



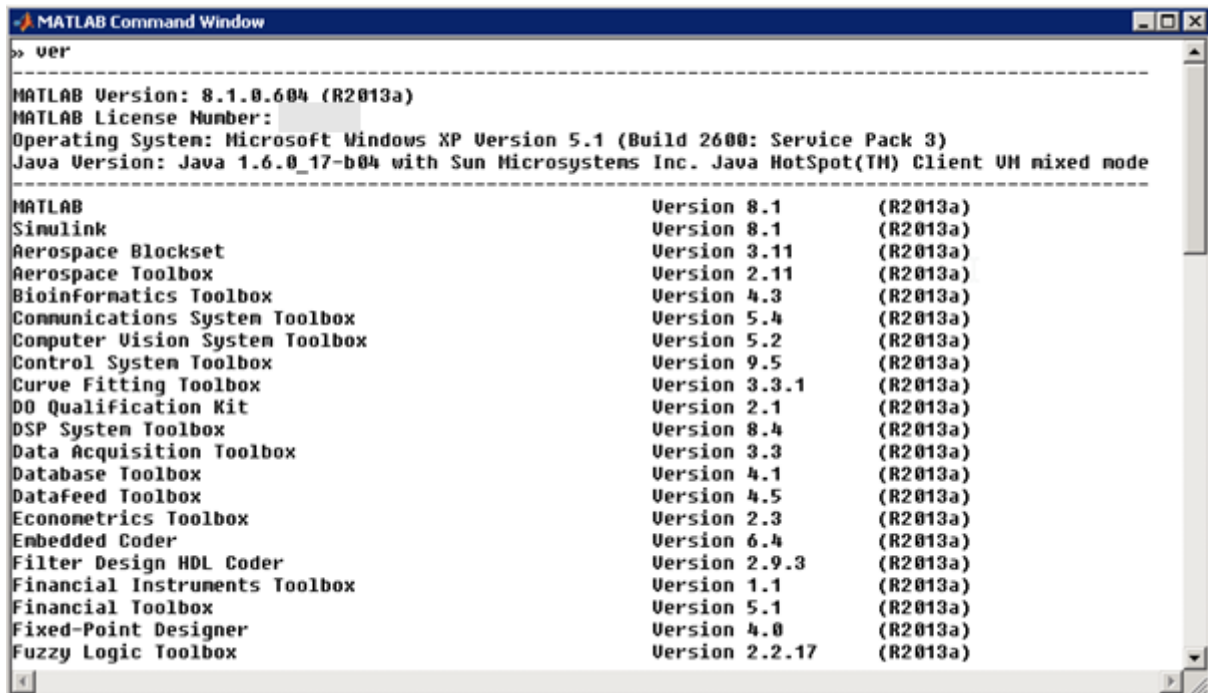
Get MATLAB system information

You can get a small resume of you MATLAB version by typing the command **version** and get the license number by typing the command **license**:

A screenshot of the MATLAB Command Window. The window has a blue title bar with the MATLAB logo and the text "MATLAB Command Window". The command prompt shows the user entering the command 'version', followed by the output 'ans = 8.1.0.604 (R2013a)'. Then, the user enters the command 'license', followed by the output 'ans =' and a small gray rectangular box representing the license number. The command prompt is currently at the '» |' line.

```
» version  
ans =  
8.1.0.604 (R2013a)  
» license  
ans =  
[redacted]  
» |
```

If you want to know the details of your version and the toolboxes available on your MATLAB installation you can use the command **ver**:



```

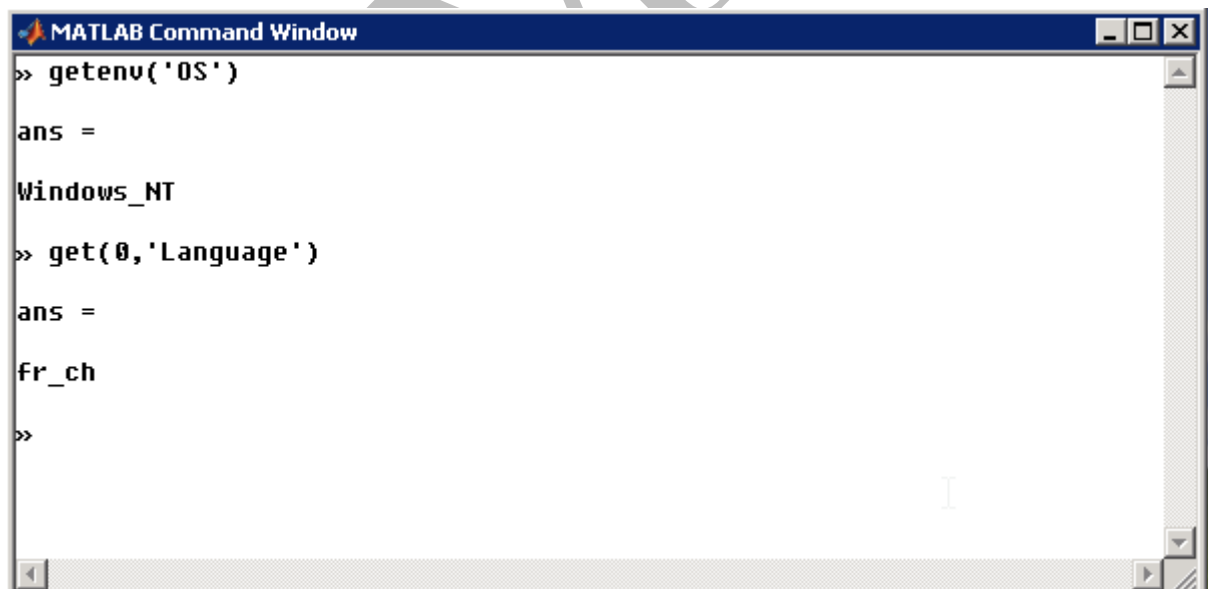
MATLAB Command Window
>> ver

-----
MATLAB Version: 8.1.0.604 (R2013a)
MATLAB License Number:
Operating System: Microsoft Windows XP Version 5.1 (Build 2600: Service Pack 3)
Java Version: Java 1.6.0_17-b04 with Sun Microsystems Inc. Java HotSpot(TM) Client VM mixed mode
-----

MATLAB                        Version 8.1      (R2013a)
Simulink                      Version 8.1      (R2013a)
Aerospace Blockset           Version 3.11     (R2013a)
Aerospace Toolbox            Version 2.11     (R2013a)
Bioinformatics Toolbox        Version 4.3      (R2013a)
Communications System Toolbox Version 5.4      (R2013a)
Computer Vision System Toolbox Version 5.2      (R2013a)
Control System Toolbox        Version 9.5      (R2013a)
Curve Fitting Toolbox         Version 3.3.1    (R2013a)
DO Qualification Kit          Version 2.1      (R2013a)
DSP System Toolbox            Version 8.4      (R2013a)
Data Acquisition Toolbox      Version 3.3      (R2013a)
Database Toolbox              Version 4.1      (R2013a)
Datafeed Toolbox              Version 4.5      (R2013a)
Econometrics Toolbox          Version 2.3      (R2013a)
Embedded Coder                 Version 6.4      (R2013a)
Filter Design HDL Coder        Version 2.9.3    (R2013a)
Financial Instruments Toolbox Version 1.1      (R2013a)
Financial Toolbox              Version 5.1      (R2013a)
Fixed-Point Designer           Version 4.0      (R2013a)
Fuzzy Logic Toolbox           Version 2.2.17   (R2013a)

```

When you develop application it is also not only important to get the MATLAB version to ensure your script will run well but also the operating system name and language. You can get this informations by using the commands **getenv('OS')** and **get(0,'Language')**:



```

MATLAB Command Window
>> getenv('OS')

ans =

Windows_NT

>> get(0,'Language')

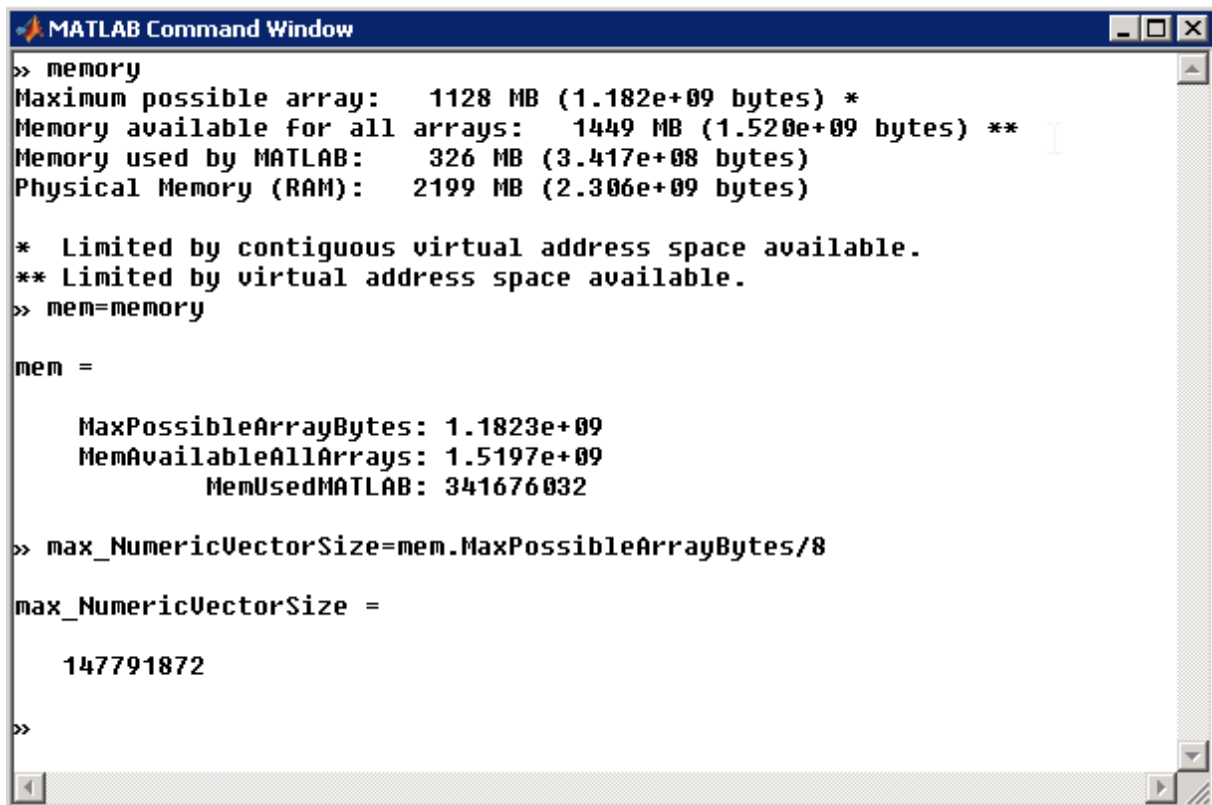
ans =

fr_ch

>>

```

Another very important information that you need when you develop commercial information is the memory available and the biggest possible numeric vector size by using the method **memory** and its **MaxPossibleArrayBytes** property:



```

MATLAB Command Window
>> memory
Maximum possible array: 1128 MB (1.182e+09 bytes) *
Memory available for all arrays: 1449 MB (1.520e+09 bytes) **
Memory used by MATLAB: 326 MB (3.417e+08 bytes)
Physical Memory (RAM): 2199 MB (2.306e+09 bytes)

* Limited by contiguous virtual address space available.
** Limited by virtual address space available.
>> mem=memory

mem =

    MaxPossibleArrayBytes: 1.1823e+09
    MemAvailableAllArrays: 1.5197e+09
    MemUsedMATLAB: 341676032

>> max_NumericVectorSize=mem.MaxPossibleArrayBytes/8

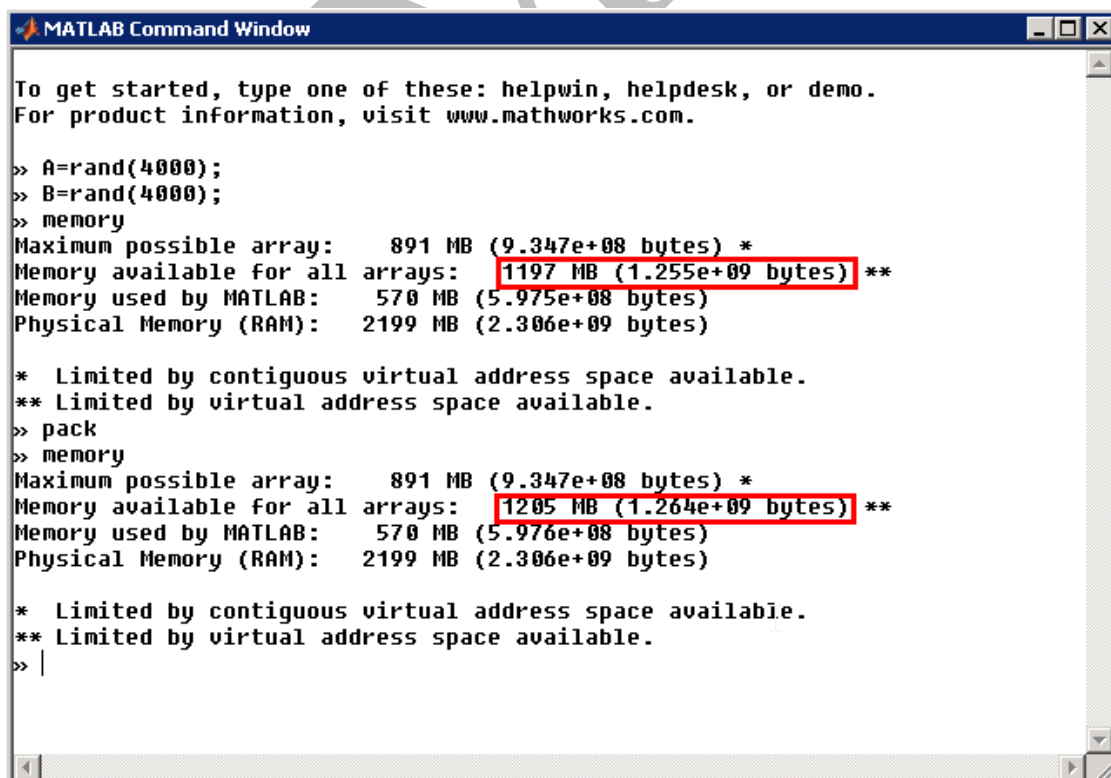
max_NumericVectorSize =

    147791872

>>

```

There is a command name **pack** that can not be used in scripts files but that allows you to defragment the memory. In this way you will clean the memory a little bit and increase also a little bit the execution time of your scripts:



```

MATLAB Command Window
To get started, type one of these: helpwin, helpdesk, or demo.
For product information, visit www.mathworks.com.

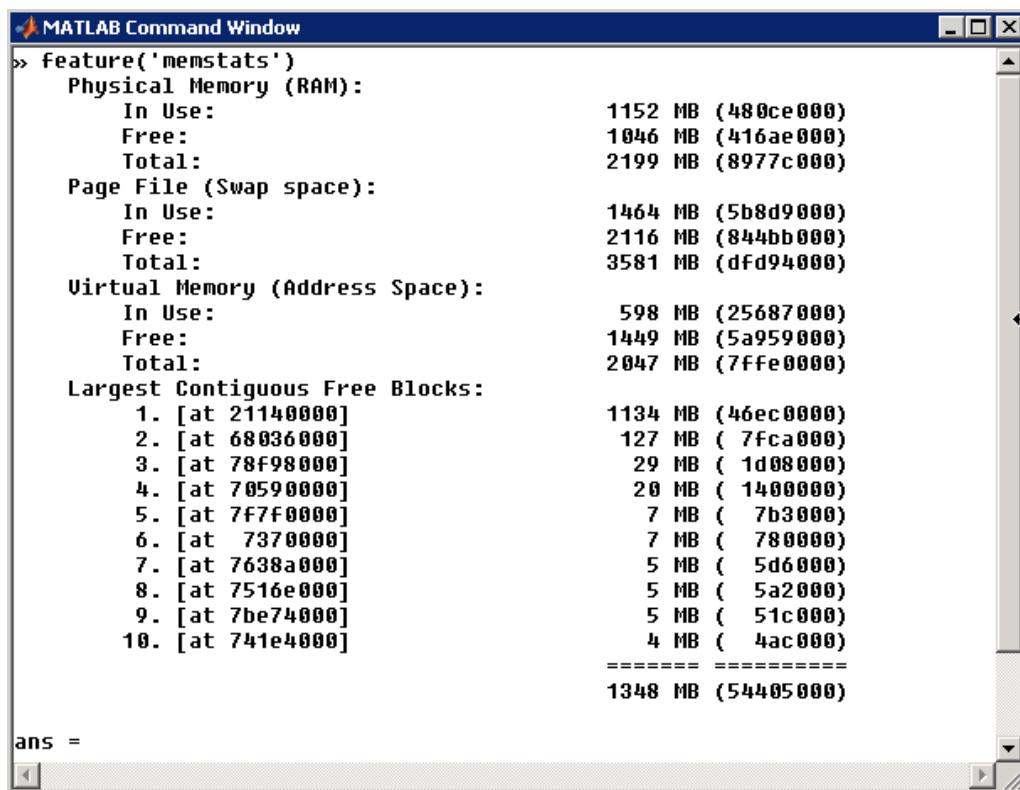
>> A=rand(4000);
>> B=rand(4000);
>> memory
Maximum possible array: 891 MB (9.347e+08 bytes) *
Memory available for all arrays: 1197 MB (1.255e+09 bytes) **
Memory used by MATLAB: 570 MB (5.975e+08 bytes)
Physical Memory (RAM): 2199 MB (2.306e+09 bytes)

* Limited by contiguous virtual address space available.
** Limited by virtual address space available.
>> pack
>> memory
Maximum possible array: 891 MB (9.347e+08 bytes) *
Memory available for all arrays: 1205 MB (1.264e+09 bytes) **
Memory used by MATLAB: 570 MB (5.976e+08 bytes)
Physical Memory (RAM): 2199 MB (2.306e+09 bytes)

* Limited by contiguous virtual address space available.
** Limited by virtual address space available.
>> |

```

There is another way to see memory usage using the command **feature('memstats')**:

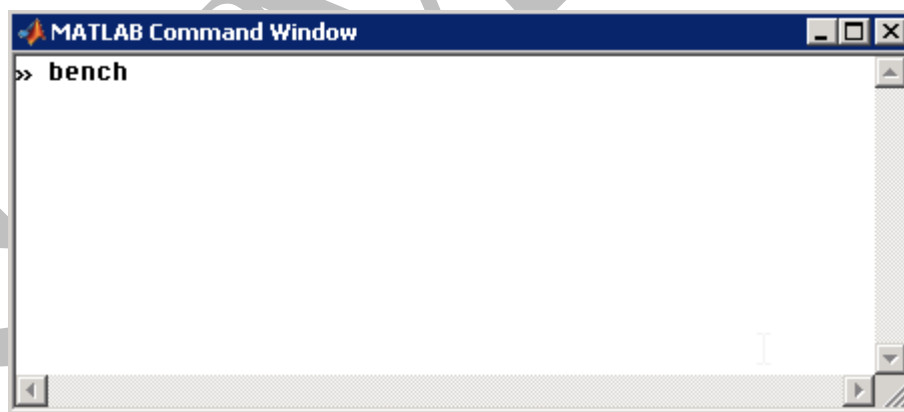


```

>> feature('memstats')
Physical Memory (RAM):
  In Use:          1152 MB (480ce000)
  Free:           1046 MB (416ae000)
  Total:          2199 MB (8977c000)
Page File (Swap space):
  In Use:          1464 MB (5b8d9000)
  Free:           2116 MB (844bb000)
  Total:          3581 MB (dfd94000)
Virtual Memory (Address Space):
  In Use:           598 MB (25687000)
  Free:            1449 MB (5a959000)
  Total:           2047 MB (7ffe0000)
Largest Contiguous Free Blocks:
  1. [at 21140000]    1134 MB (46ec0000)
  2. [at 68036000]    127 MB ( 7fca0000)
  3. [at 78f98000]     29 MB ( 1d080000)
  4. [at 70590000]     20 MB ( 14000000)
  5. [at 7f7f0000]      7 MB ( 7b300000)
  6. [at 73700000]      7 MB ( 78000000)
  7. [at 7638a000]      5 MB ( 5d600000)
  8. [at 7516e000]      5 MB ( 5a200000)
  9. [at 7be74000]      5 MB ( 51c00000)
 10. [at 741e4000]      4 MB ( 4ac00000)
=====
1348 MB (54405000)

ans =
  
```

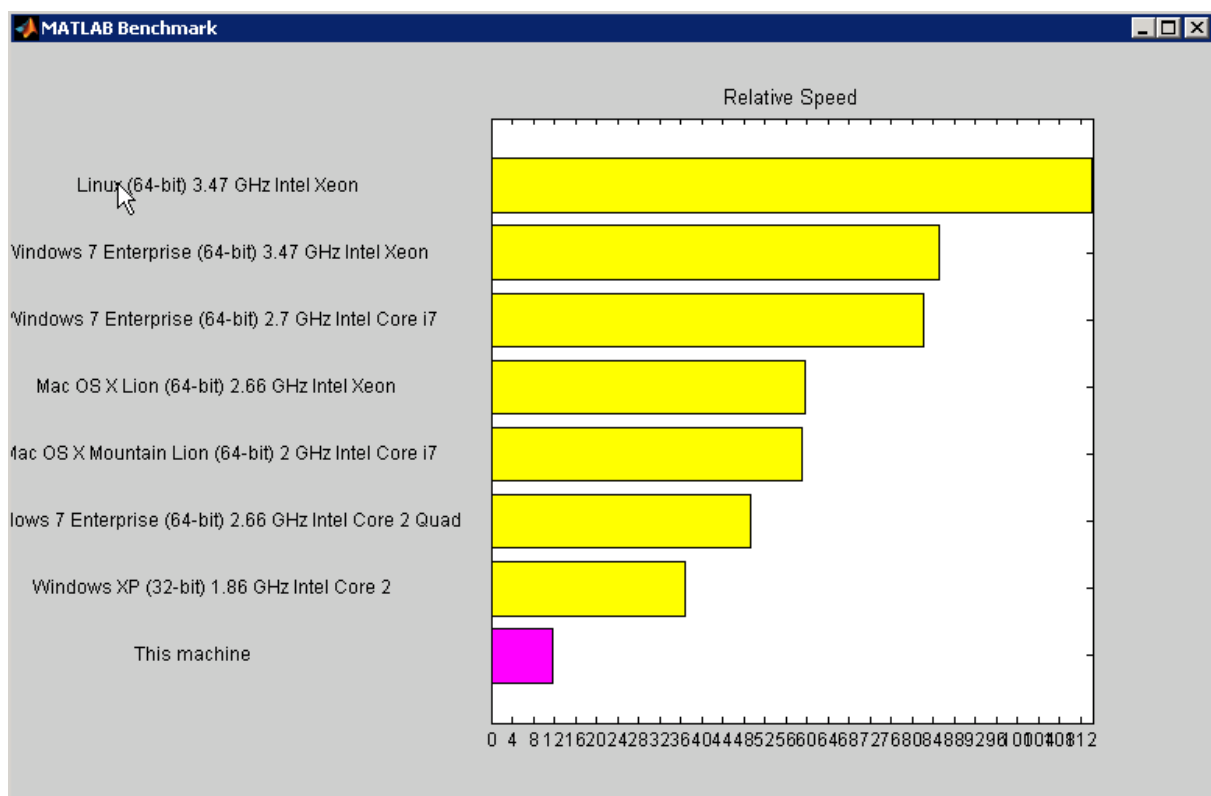
For fun or for technical support with a customer you can run the **bench** command to benchmark your computer performance with typical others systems:



```

>> bench
  
```

You will get a first chart after a few seconds:

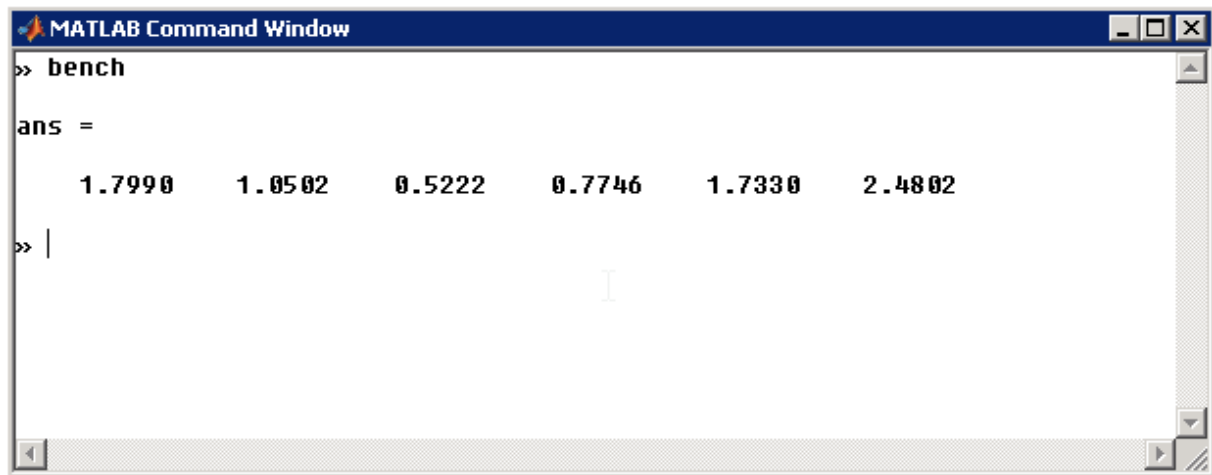


You will also get a figure with a resume just after:

MATLAB Benchmark (times in seconds)						
Computer Type	LU	FFT	ODE	Sparse	2-D	3-D
Linux (64-bit) 3.47 GHz Intel Xeon	0.0626	0.0661	0.1617	0.1161	0.2055	0.0911
Windows 7 Enterprise (64-bit) 3.47 GHz Intel Xeon	0.0701	0.0719	0.1094	0.1297	0.2784	0.7044
Windows 7 Enterprise (64-bit) 2.7 GHz Intel Core i7	0.0857	0.0801	0.0958	0.1319	0.2975	0.7003
Mac OS X Lion (64-bit) 2.66 GHz Intel Xeon	0.0547	0.1278	0.2008	0.1877	0.6670	0.6299
Mac OS X Mountain Lion (64-bit) 2 GHz Intel Core i7	0.0725	0.1292	0.1881	0.1587	0.7203	0.6476
Windows 7 Enterprise (64-bit) 2.66 GHz Intel Core 2 Quad	0.1239	0.2333	0.1561	0.2822	0.4819	0.7390
Windows XP (32-bit) 1.86 GHz Intel Core 2	0.3406	0.3178	0.1883	0.3542	0.5775	0.3601
This machine	1.7990	1.0502	0.5222	0.7746	1.7330	2.4802

Place the cursor near a computer name for system and version details. Before using this data to compare different versions of MATLAB, or to download an updated timing data file, see the help for the bench function by typing "help bench" at the MATLAB prompt.

and in the Command Windows you will see the above blue line available in the variable **ans** so you can manipulate these information later in your own scripts if you want to warn your customer about its low performance computer:



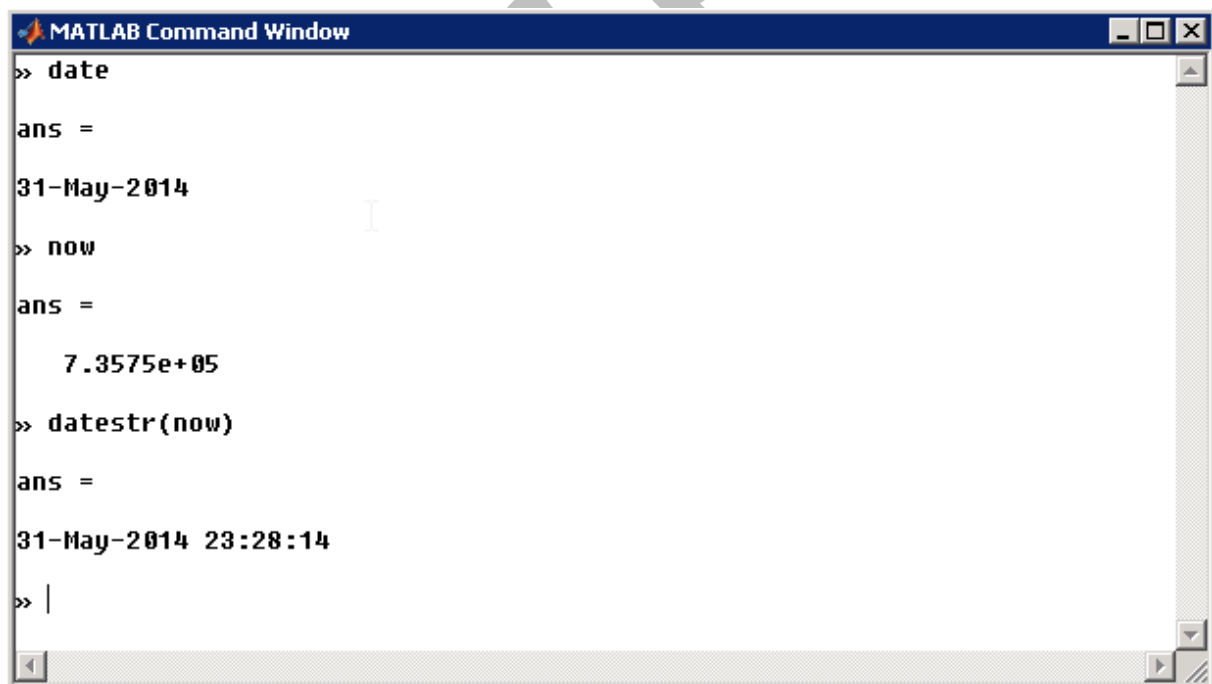
```
MATLAB Command Window
>> bench

ans =

    1.7990    1.0502    0.5222    0.7746    1.7330    2.4802
>> |
```

Some dates stuff

Late we will study dates in details but we can also consider that some dates manipulations command are very general. This is the case of the **date**, **now** and **datestr(now)** commands



```
MATLAB Command Window
>> date

ans =

31-May-2014
>> now

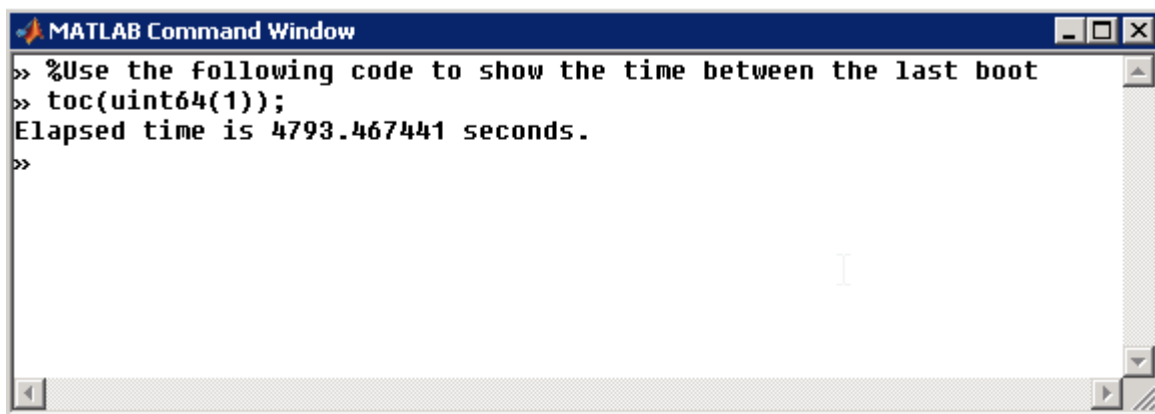
ans =

    7.3575e+05
>> datestr(now)

ans =

31-May-2014 23:28:14
>> |
```

Sometimes your customer will forget to periodically reboot its servers or computer. To check since how many seconds the computer is started you can use the command **toc(uint64(1))**:



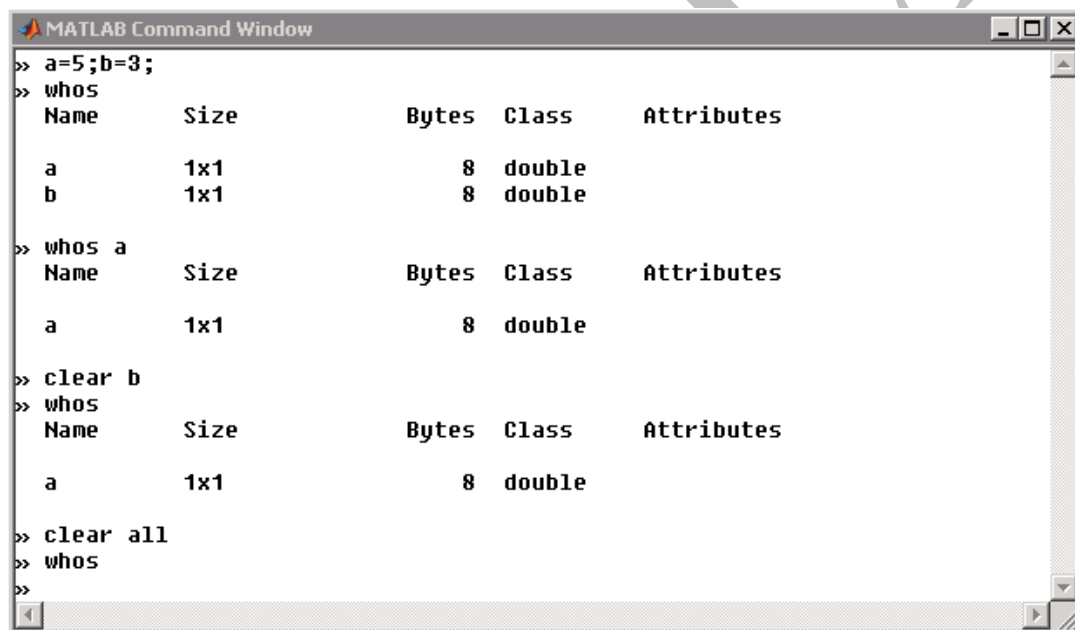
```

MATLAB Command Window
>> %Use the following code to show the time between the last boot
>> toc(uint64(1));
Elapsed time is 4793.467441 seconds.
>>

```

Manage variables

If you create variables in MATLAB and use the command **whos** you will have information about all variables or only some variables of the current workspace. If you use **clear** or **clear all** you can remove one or more variables:

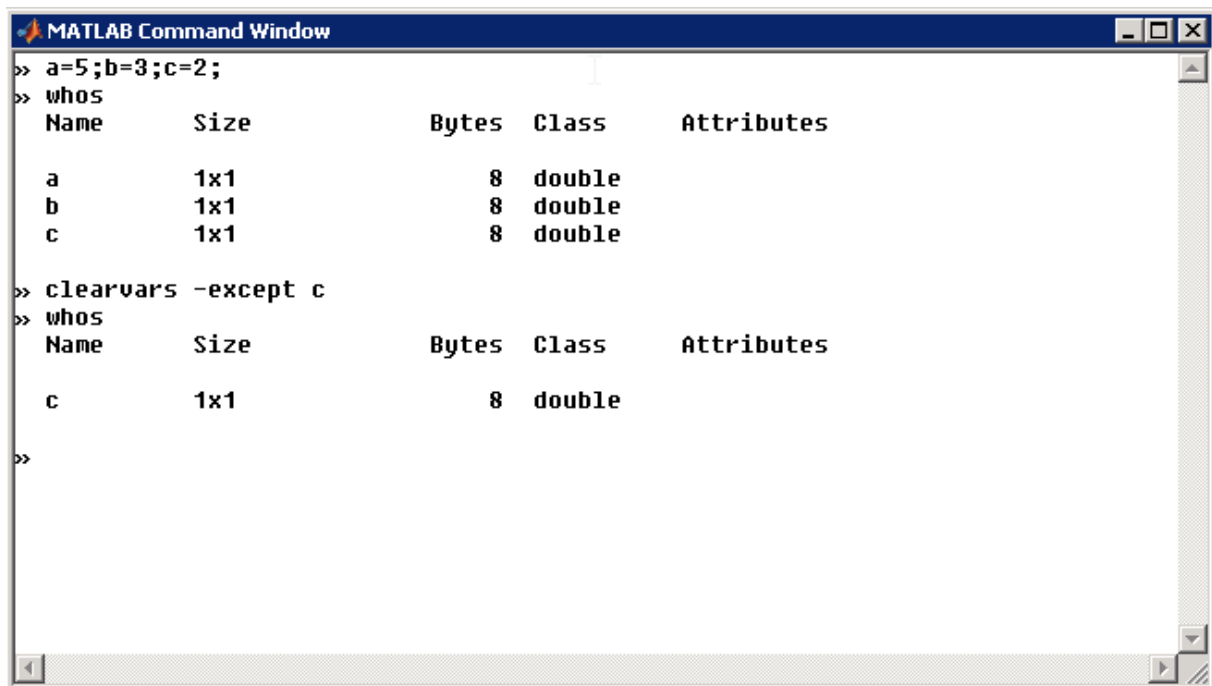


```

MATLAB Command Window
>> a=5;b=3;
>> whos
  Name      Size      Bytes  Class  Attributes
  a         1x1         8  double
  b         1x1         8  double
>> whos a
  Name      Size      Bytes  Class  Attributes
  a         1x1         8  double
>> clear b
>> whos
  Name      Size      Bytes  Class  Attributes
  a         1x1         8  double
>> clear all
>> whos
>>

```

Sometimes when running script you want to delete all variables and keep only the final one. For this you can use the command **clearvars -except**:

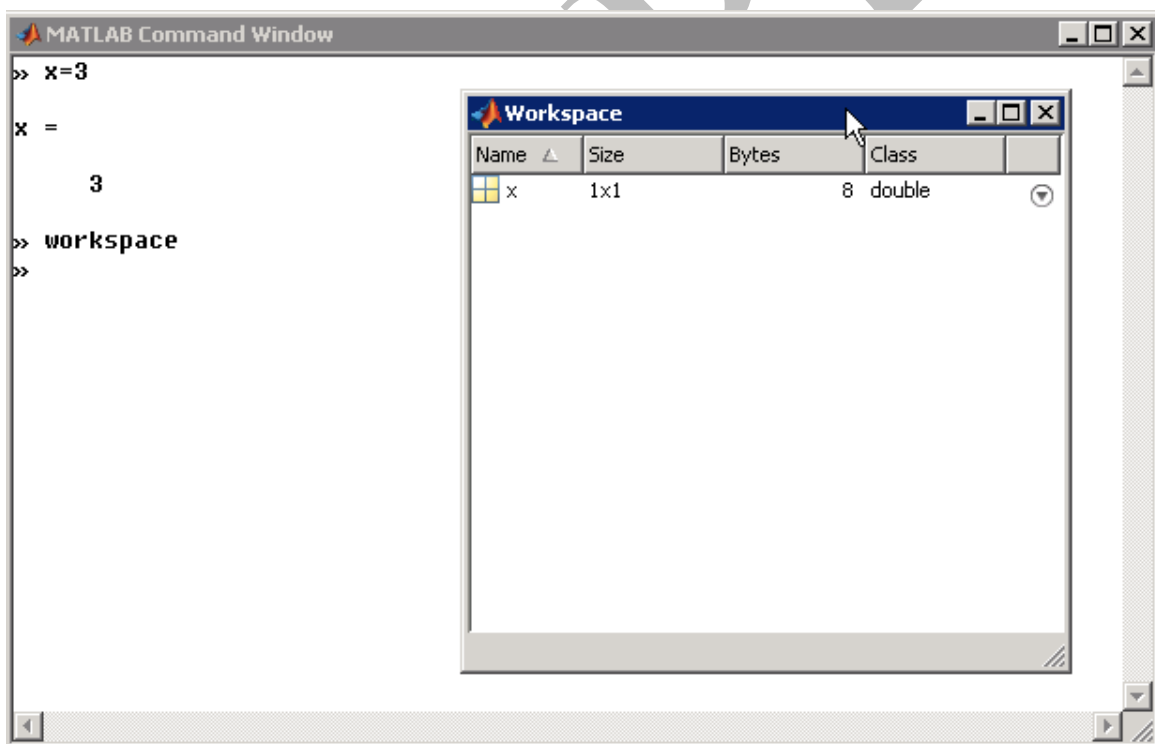


```
>> a=5;b=3;c=2;
>> whos
  Name      Size      Bytes  Class  Attributes
  a         1x1         8  double
  b         1x1         8  double
  c         1x1         8  double

>> clearvars -except c
>> whos
  Name      Size      Bytes  Class  Attributes
  c         1x1         8  double

>>
```

You can also see the variables workspace windows by typing the command **workspace**:

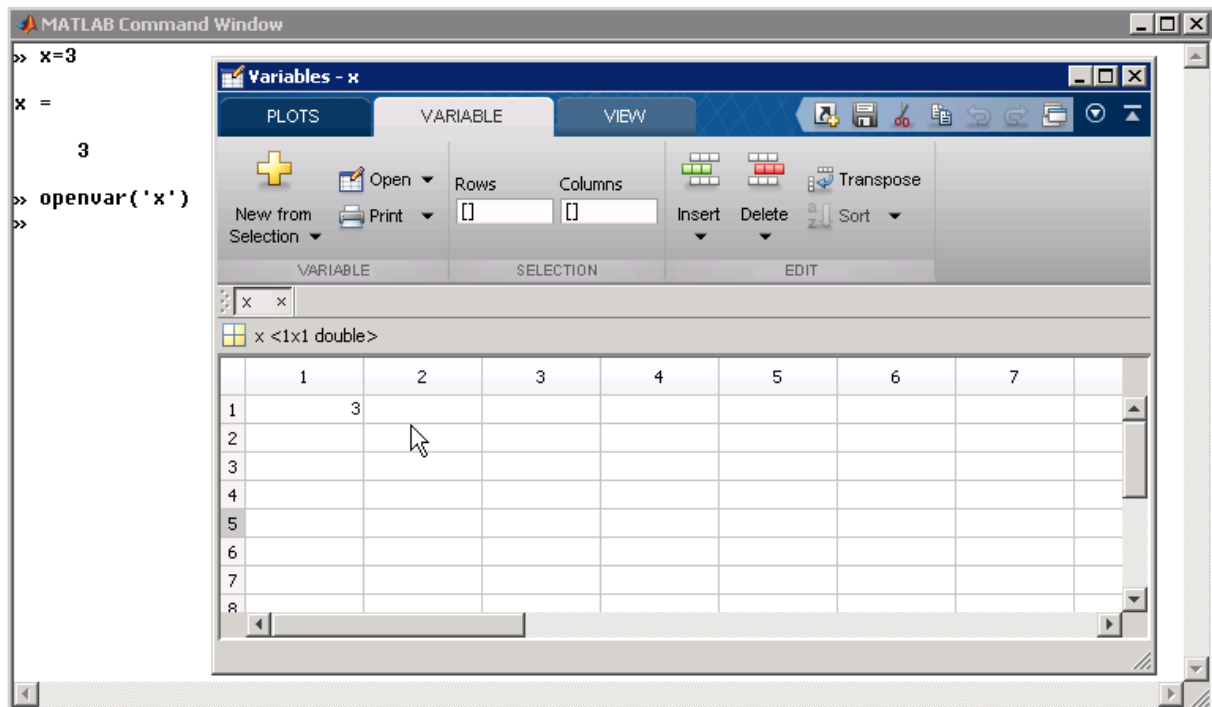


```
>> x=3
x =
     3

>> workspace
>>
```

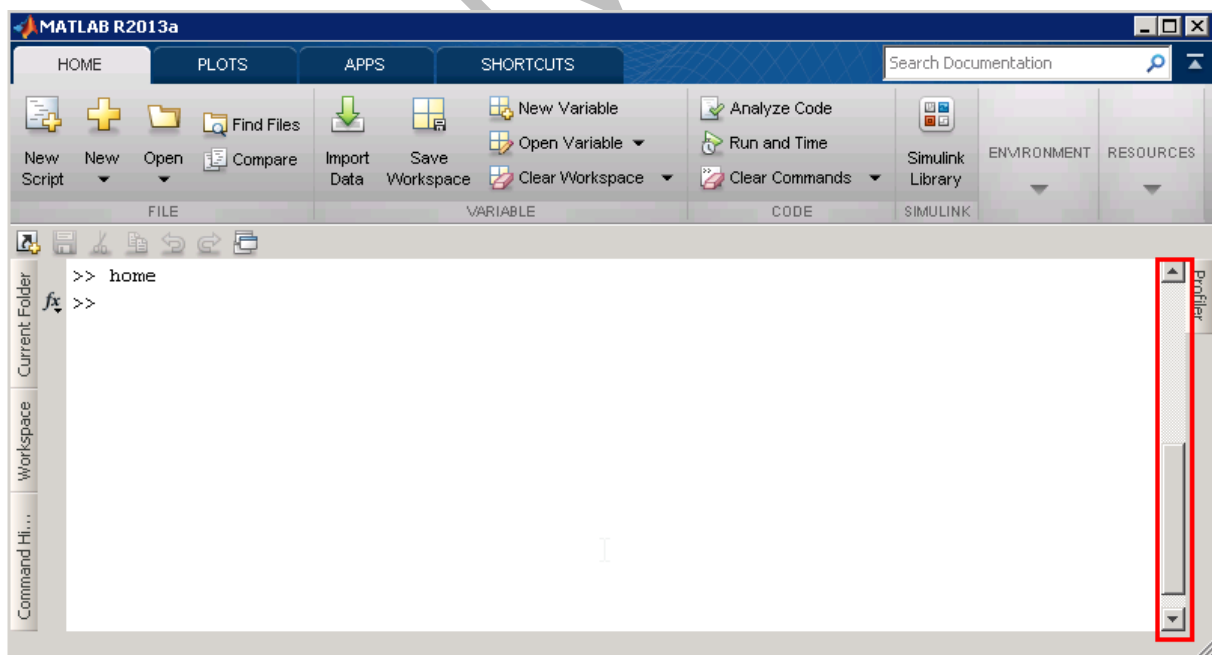
Name	Size	Bytes	Class
x	1x1	8	double

And if you have complex variables (arrays, cell array, dataset...) you can edit some of them by using the command **openvar** and you will have the variable editor that will open.

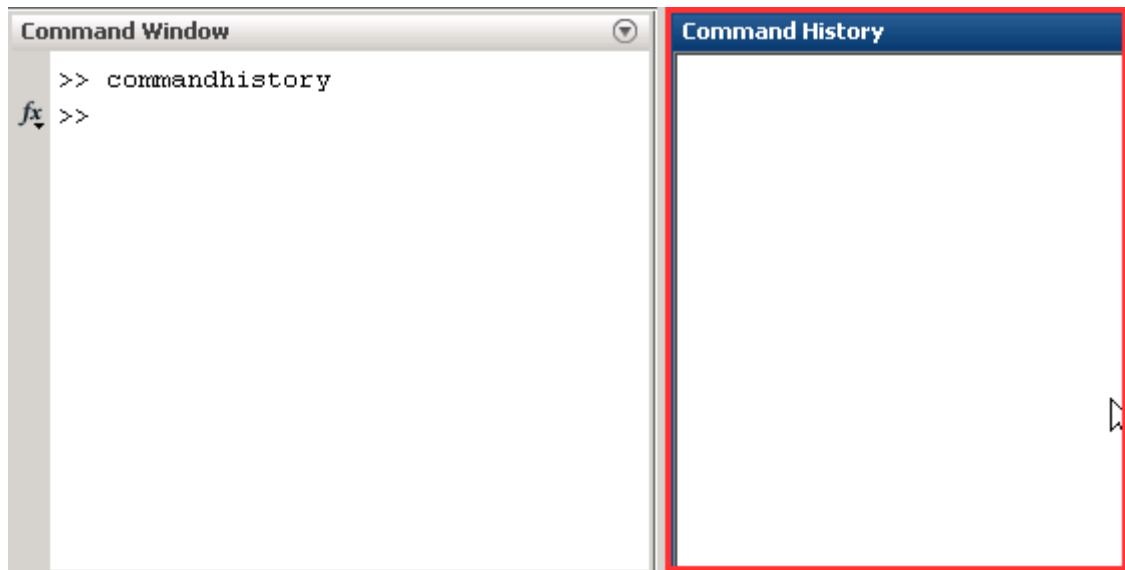


Some Command Window controls

If you are using the MATLAB GUI by typing the command **home** you will get back to the top of the Command Windows (in the old style Command Windows this will just have the same effect as using the command **clc** that clear all previous written commands):



If you type **commandhistory** in the MATLAB GUI this will force the the Command History window to appear (**the *Command History* windows shows only the commands that were typed in the MATLAB GUI not the commands typed in the old style MATLAB Command Window!!!**):

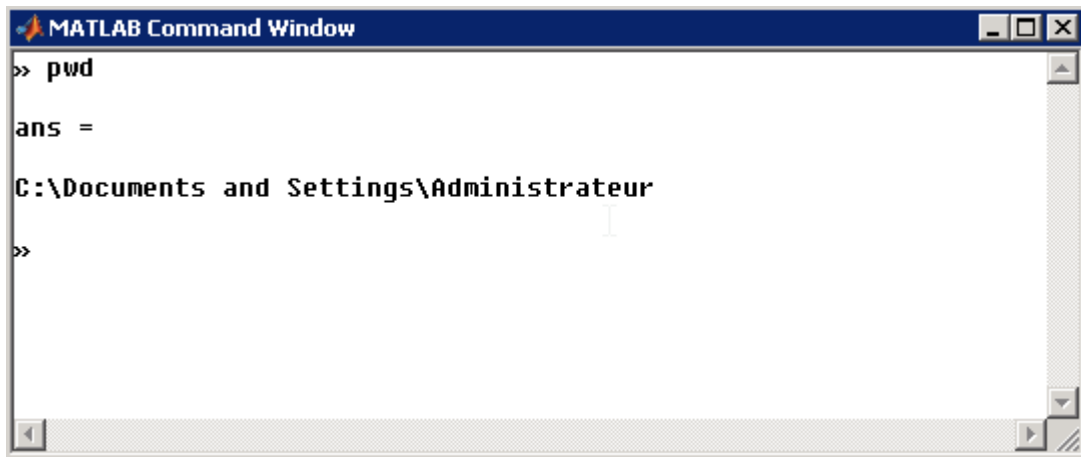


If you want to clear the content of the history without using the mouse but with a simple command you will have to use the following command:

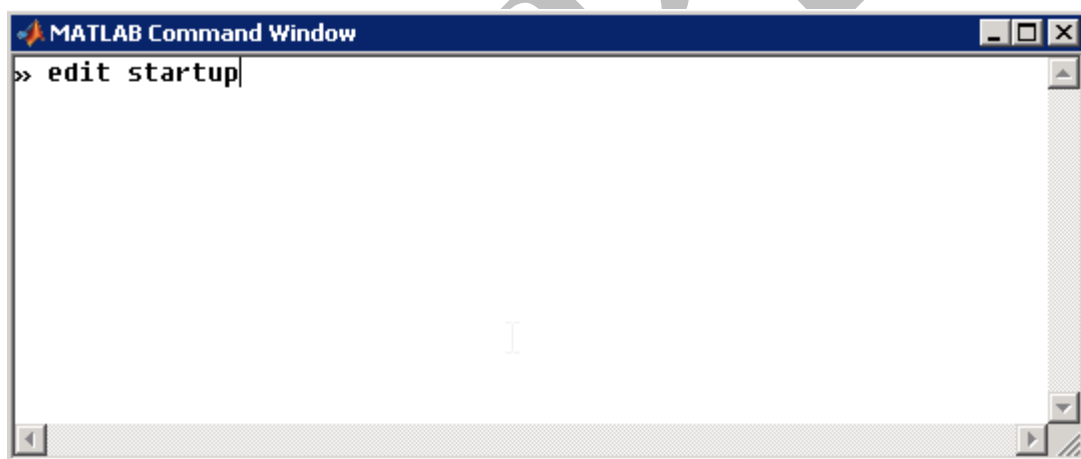


Use the startup.m file

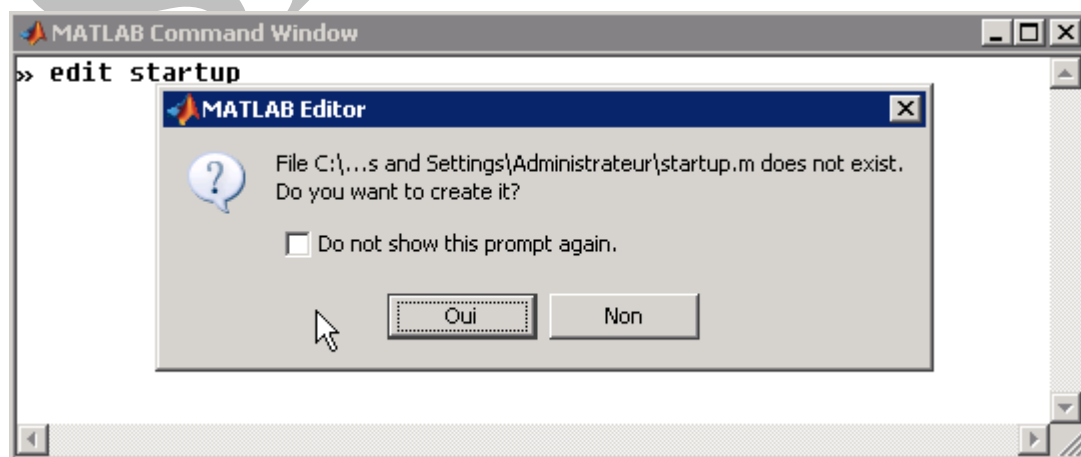
By default MATLAB has a workspace path where you can put a *startup.m* file script that will run each time you will open the software. You can get this path by typing **pwd** (we will see later how to change it):



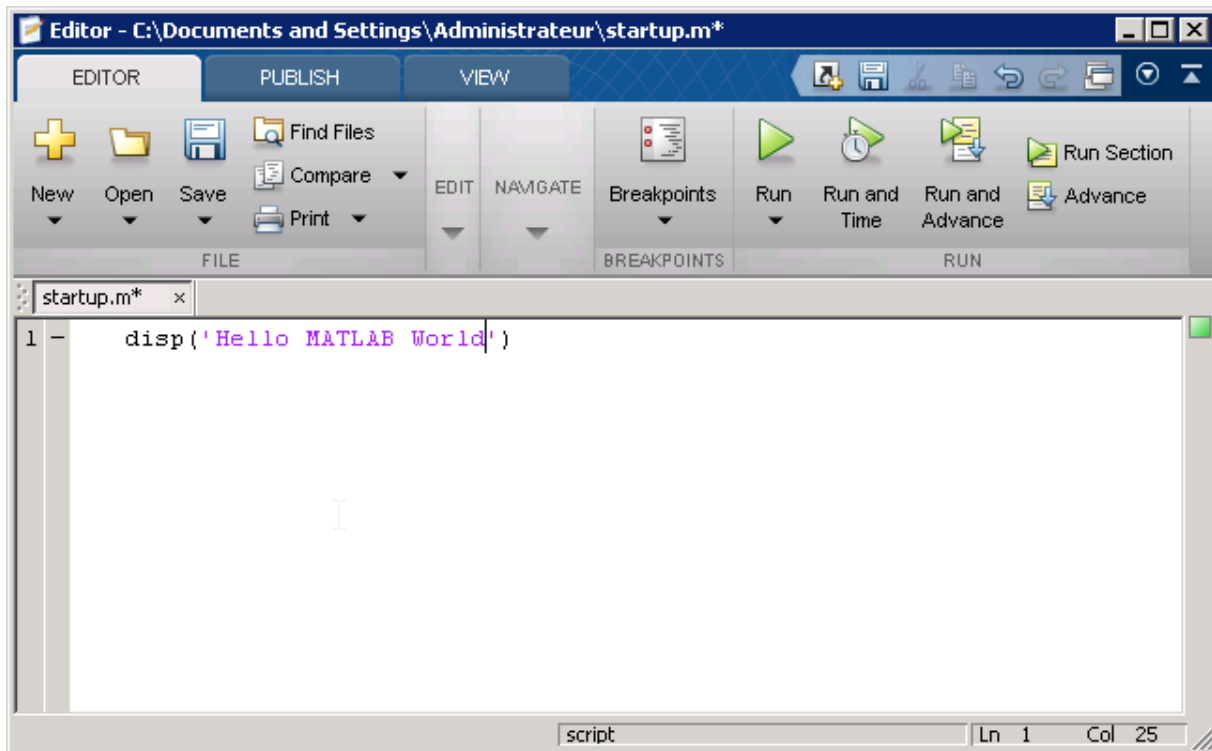
After if you want to create the *startup.m* file for the first time just use the command **edit**:



If the file does not exist, MATLAB will ask you if you want to create it:



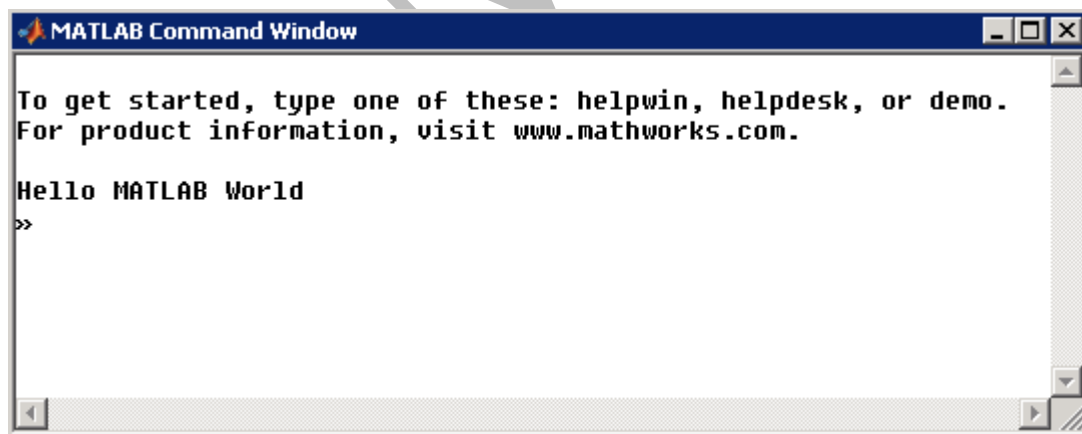
Then the *startup.m* file will open in the MATLAB Script Editor:



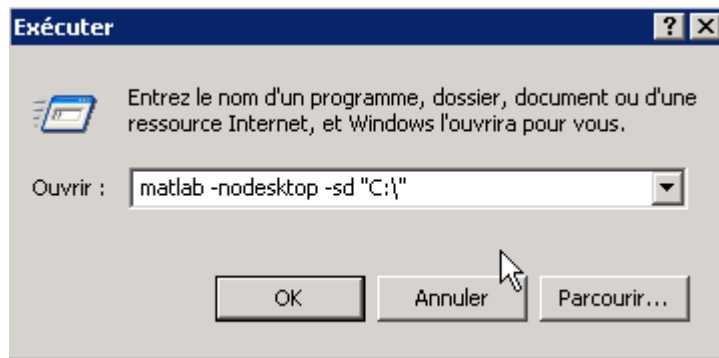
You can type a lot of stuffs in the *startup.m* file...

Remember: You can't put the *pack* command in the *startup.m* or any other script!!!

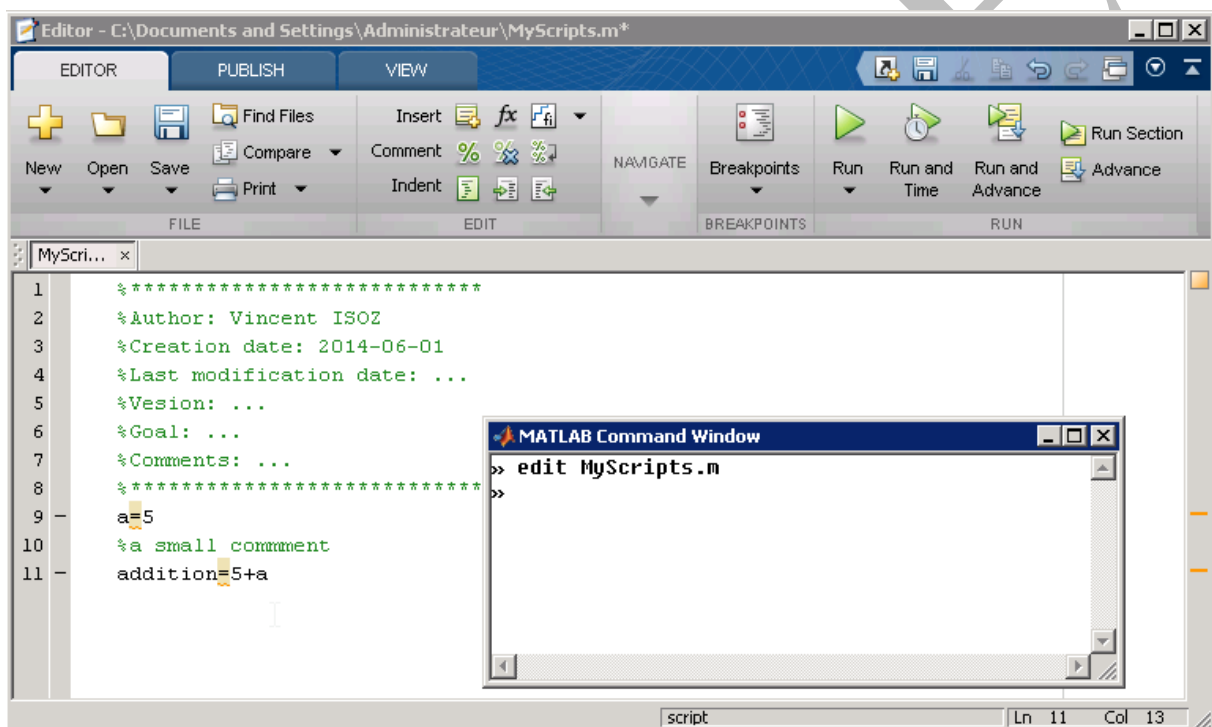
If you save the file and close/reopen MATLAB you will get:



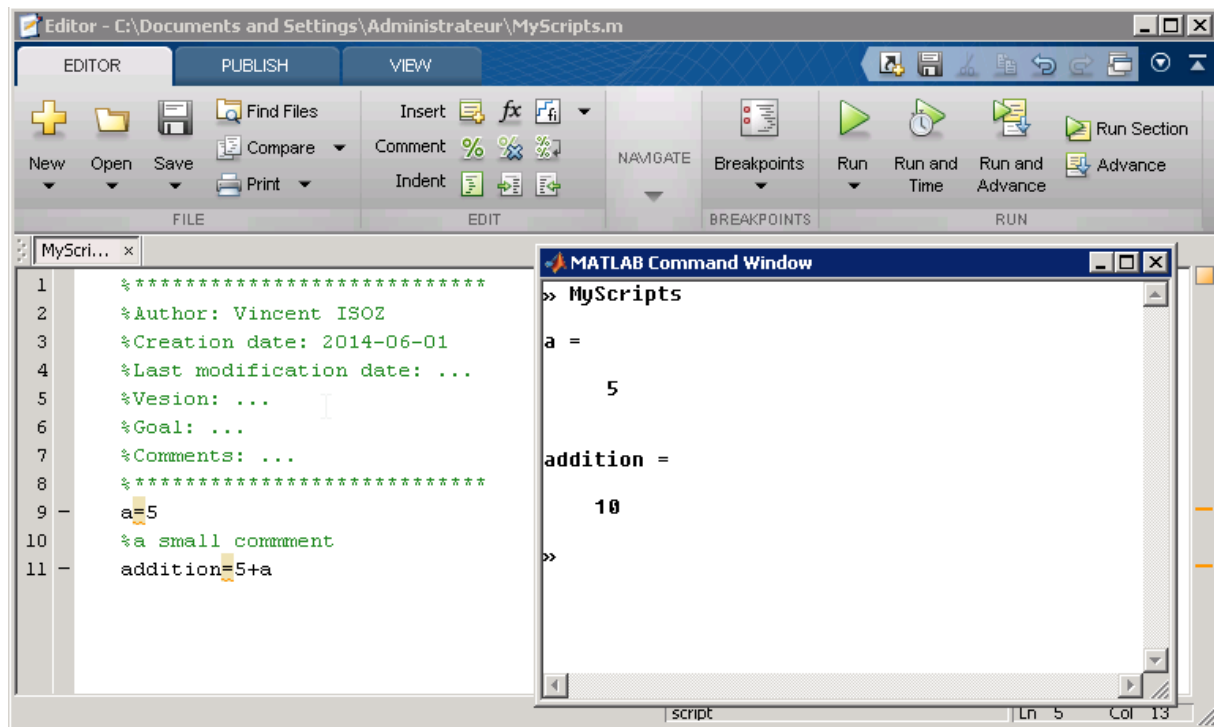
If you want to change the MATLAB default start directory (**sd**) you can change the icon shortcut with the following options:



In the facts you can create and use any *.m script file following the same process. For example if we want to create in the current working path a new file named *myScripts.m* we will write the following:

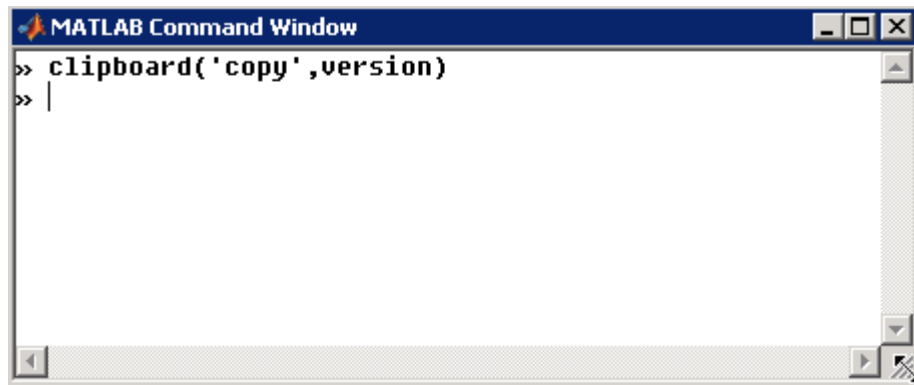


And to run it we will just write the script file name:

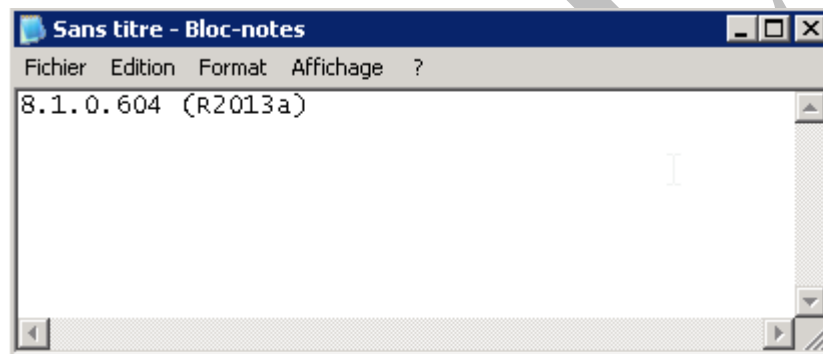


Use the clipboard

You can copy with the command **clipboard** MATLAB information or variables so the user will then be able to make a simple paste in its favorite application:

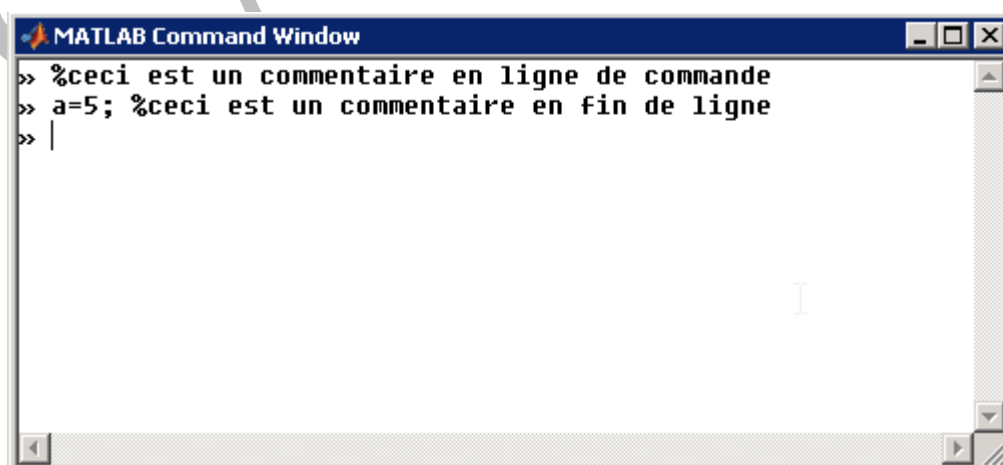


and if we open the Notepad for example and we paste (Ctrl+V) we get:

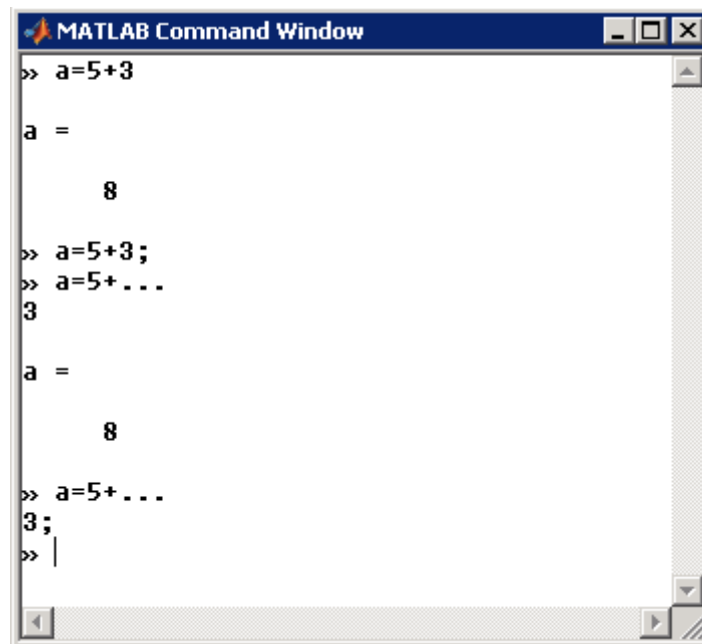


Comments and multiple line commands

Typing comments is very important in any computer language. In MATLAB there are three types of comments (we will see them all later): simple line comment that begin with `%`, simple line title comments that begin with `%%`, and multiple line comments `%{ ... }%`



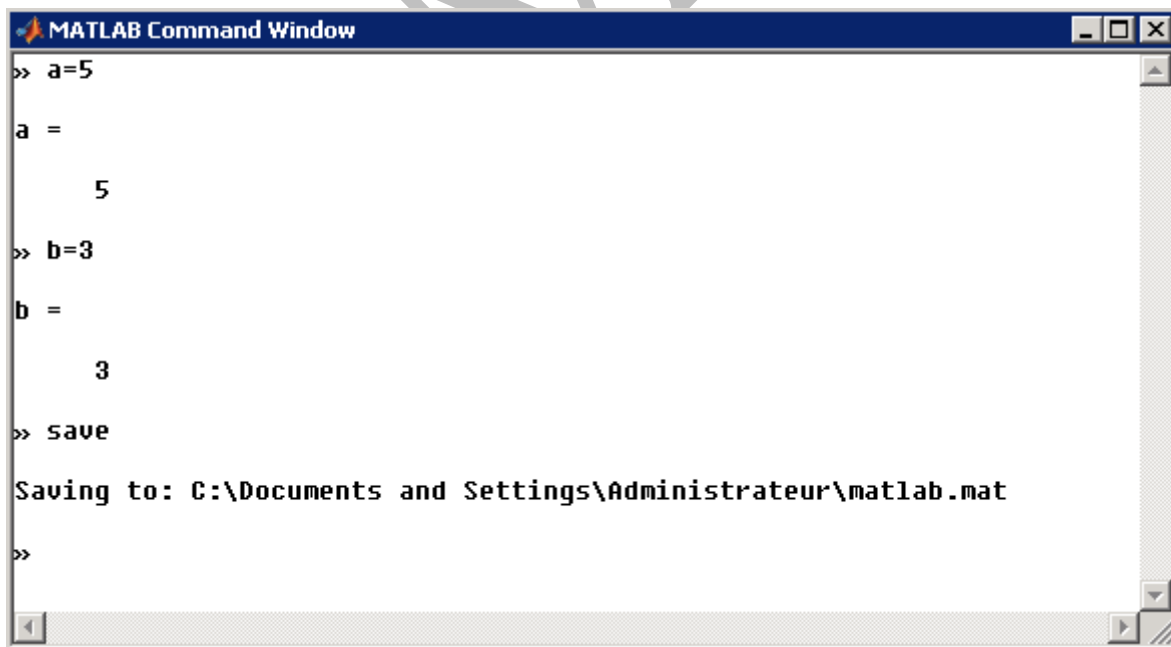
Many times in MATLAB you will have to type very long command. To cut on command on multiple lines just type three dots ... just after a comma or a mathematical operator:



```
MATLAB Command Window
>> a=5+3
a =
    8
>> a=5+3;
>> a=5+...
3
a =
    8
>> a=5+...
3;
>> |
```

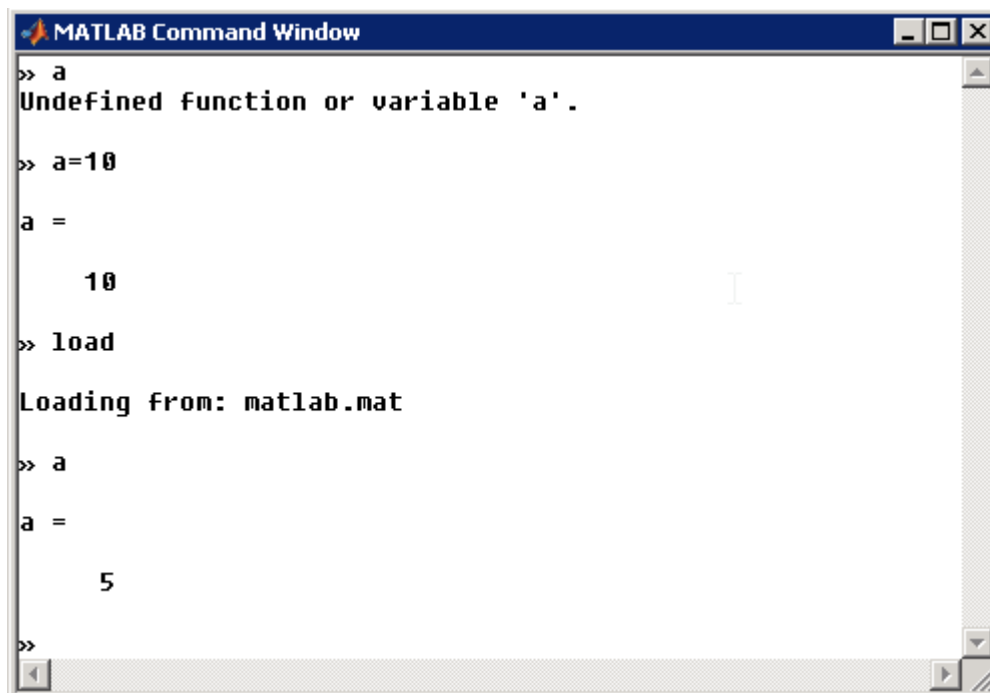
Save workspace/Use commands diary

You can save all variables you have created or imported in a binary *.mat file to recover this informations later. Using the command **save**, you will save all the variables of the workspace by default in the current working path under the default matlab.mat filename:



```
MATLAB Command Window
>> a=5
a =
    5
>> b=3
b =
    3
>> save
Saving to: C:\Documents and Settings\Administrateur\matlab.mat
>>
```

If you make a **clear all** and try to access to one of your previous variable, it will be empty but if you load the previous matlab.mat file using the **load** command the variable will be again accessible:



```
MATLAB Command Window

>> a
Undefined function or variable 'a'.

>> a=10

a =

    10

>> load

Loading from: matlab.mat

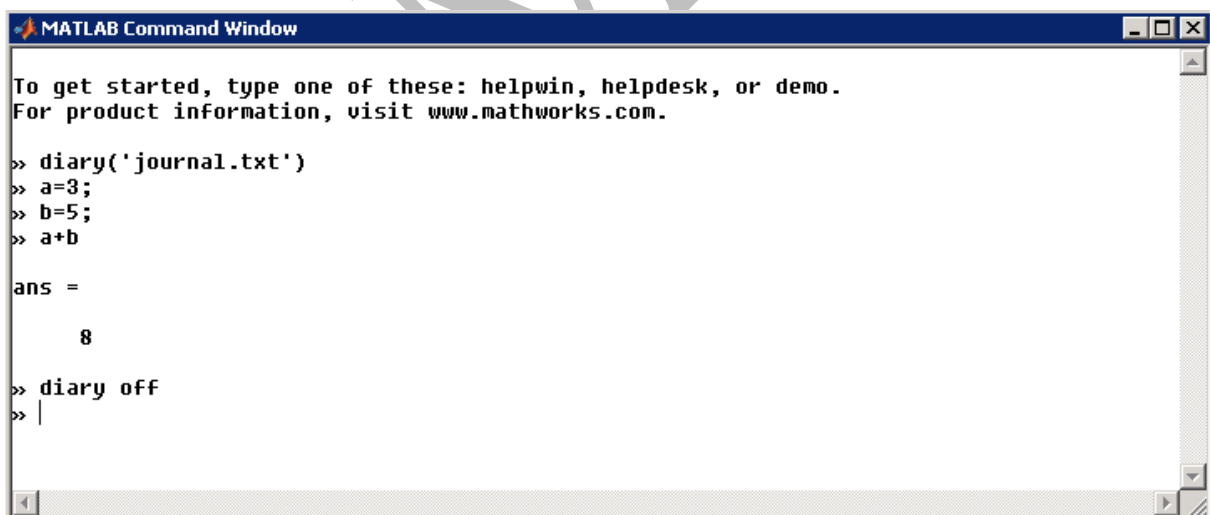
>> a

a =

     5

>>
```

If you want you can ask MATLAB to save in a text file all comands that you type. For this you just have use the command **diary** and specify between bracket the path and filename you want to use for this purpose:



```
MATLAB Command Window

To get started, type one of these: helpwin, helpdesk, or demo.
For product information, visit www.mathworks.com.

>> diary('journal.txt')
>> a=3;
>> b=5;
>> a+b

ans =

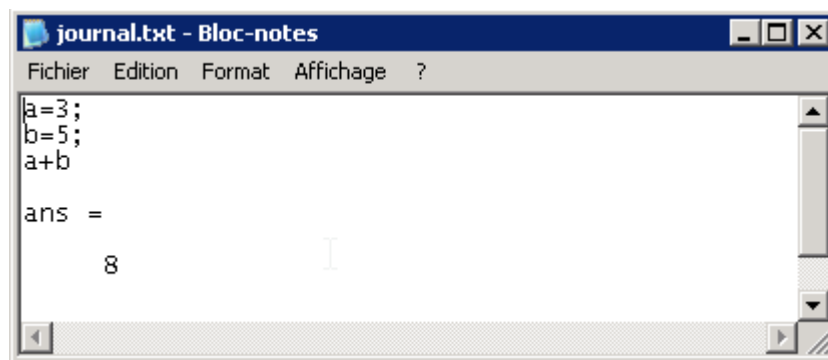
     8

>> diary off
>> |
```

Then you will get a new file on your hard drive:



The content will looks like following:

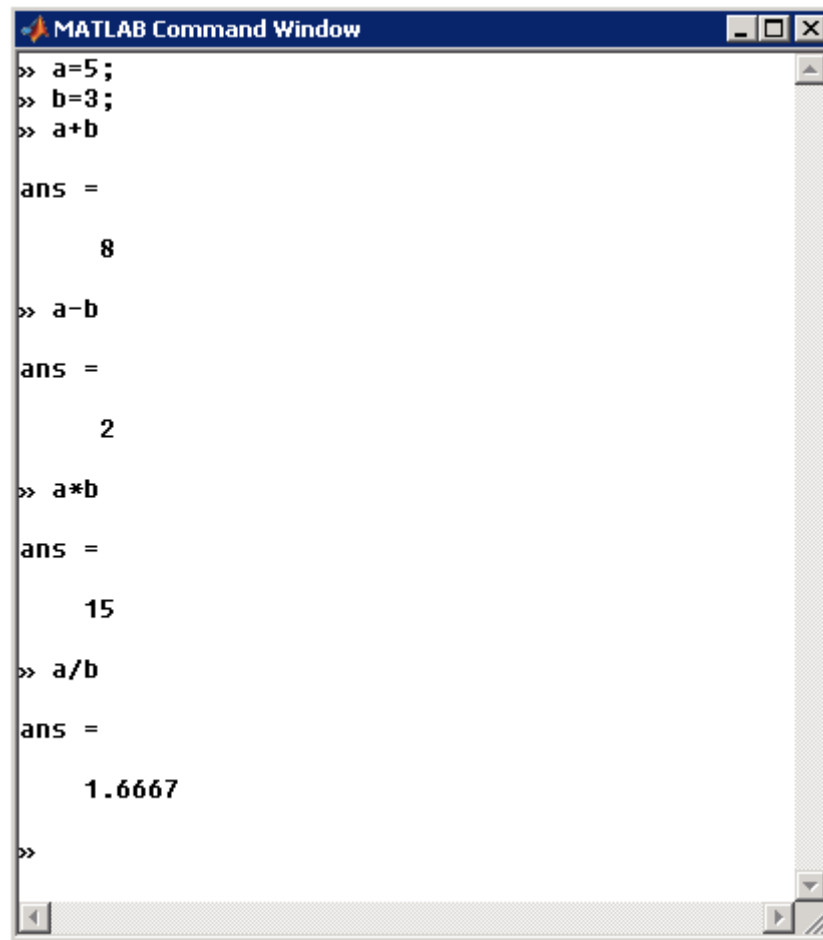


```
journal.txt - Bloc-notes
Fichier  Edition  Format  Affichage  ?
a=3;
b=5;
a+b
ans =
      8
```

Arithmetic operations on scalars

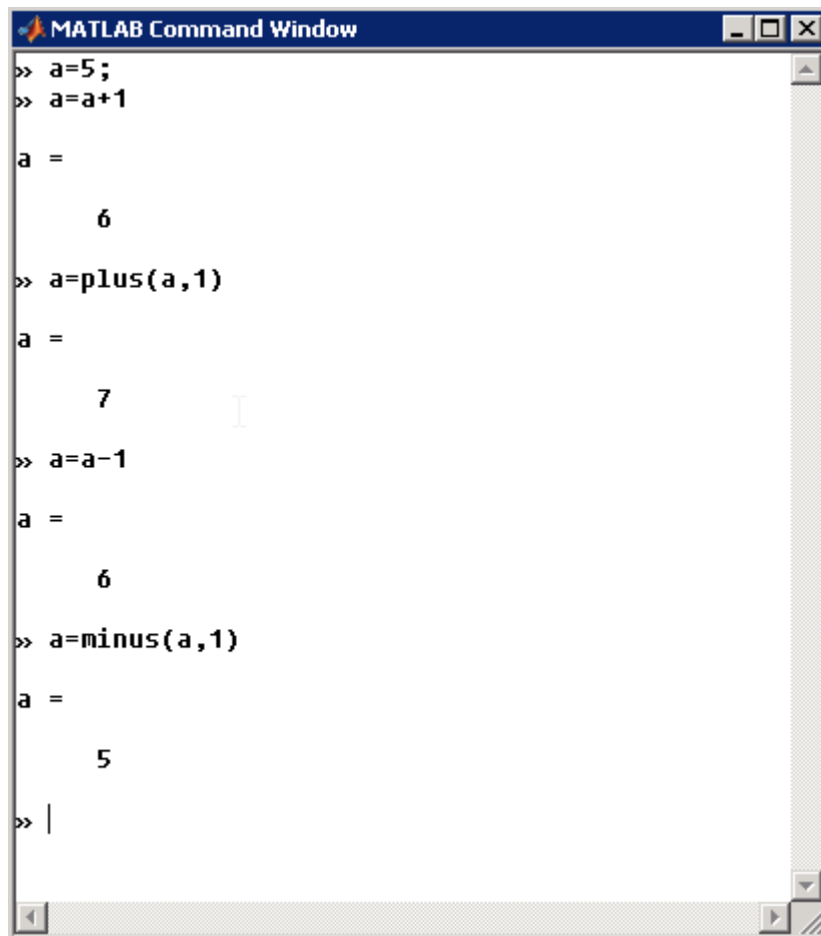
The purpose of this chapter is to have a look to the syntax of very trivial scalar number arithmetic operators in MATLAB.

We first begin with addition, subtraction, multiplication and division:

A screenshot of the MATLAB Command Window. The window has a blue title bar with the MATLAB logo and the text "MATLAB Command Window". The command history shows the following:

```
>> a=5;  
>> b=3;  
>> a+b  
  
ans =  
  
      8  
  
>> a-b  
  
ans =  
  
      2  
  
>> a*b  
  
ans =  
  
     15  
  
>> a/b  
  
ans =  
  
    1.6667  
  
>>
```

There is also another way to add/subtract in MATLAB using the functions **plus()** and **minus()** that is use sometimes by MATLAB users but used a lot when you want to optimize matrices addition/subtraction as we will see it later:



```
MATLAB Command Window

>> a=5;
>> a=a+1

a =

     6

>> a=plus(a,1)

a =

     7

>> a=a-1

a =

     6

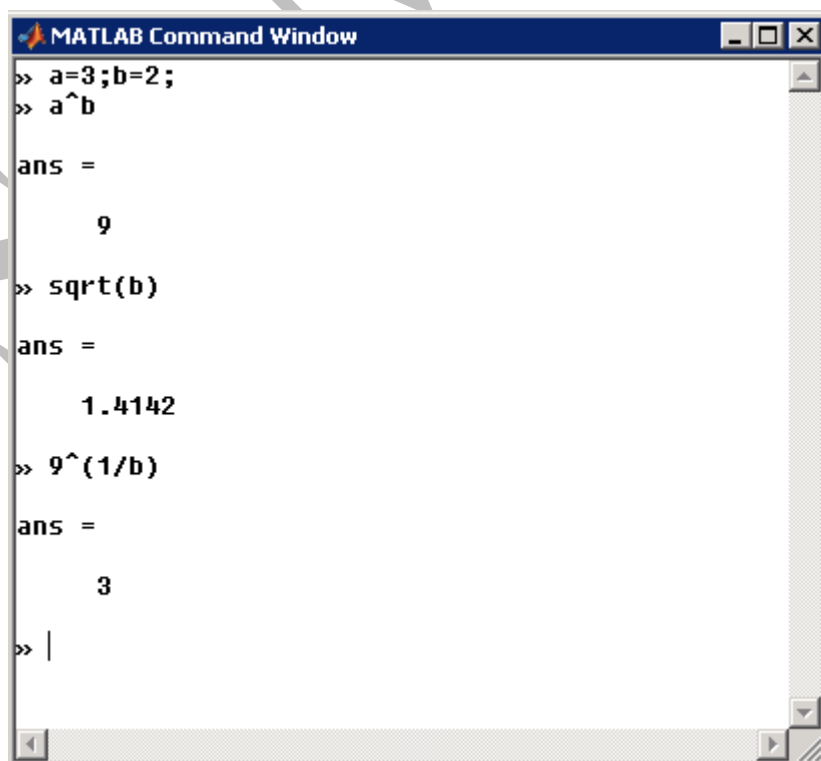
>> a=minus(a,1)

a =

     5

>> |
```

Now we take a look to powers and roots (as you can see the square too can be obtained with the classical **sqrt()** function):



```
MATLAB Command Window

>> a=3;b=2;
>> a^b

ans =

     9

>> sqrt(b)

ans =

  1.4142

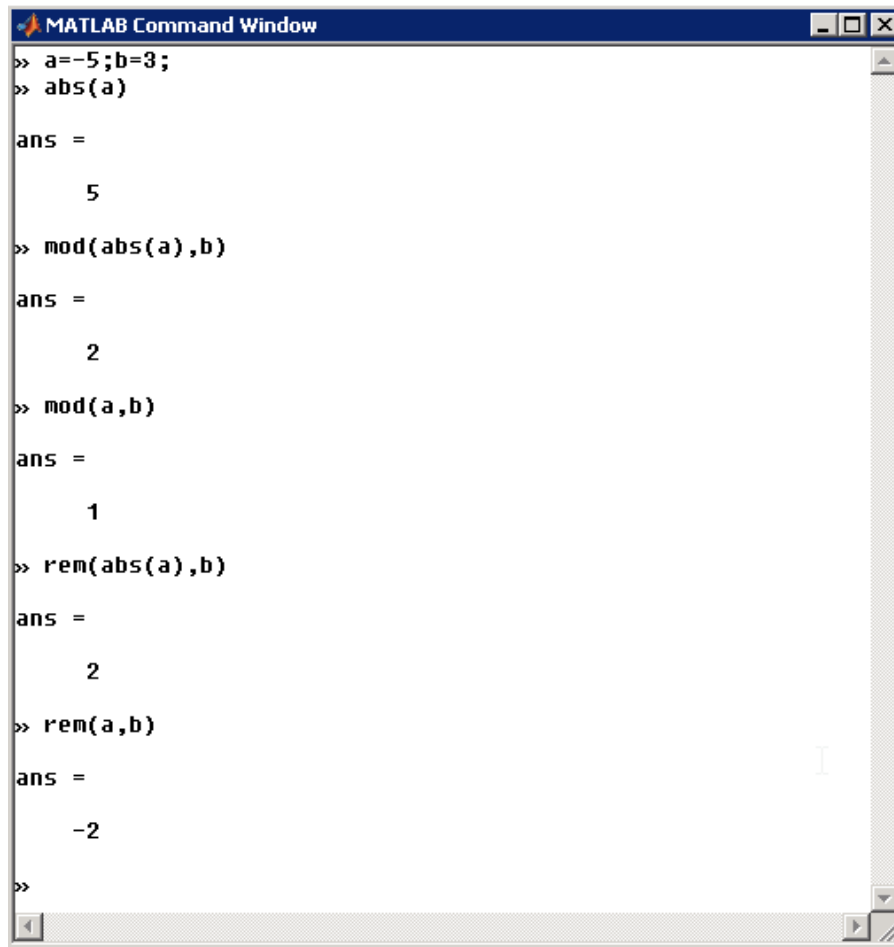
>> 9^(1/b)

ans =

     3

>> |
```

Now we take a look the classic absolute, modulo and remaining calculations whose respective functions are **abs()**, **mod()** and **rem()**:



```
MATLAB Command Window
>> a=-5;b=3;
>> abs(a)

ans =

     5

>> mod(abs(a),b)

ans =

     2

>> mod(a,b)

ans =

     1

>> rem(abs(a),b)

ans =

     2

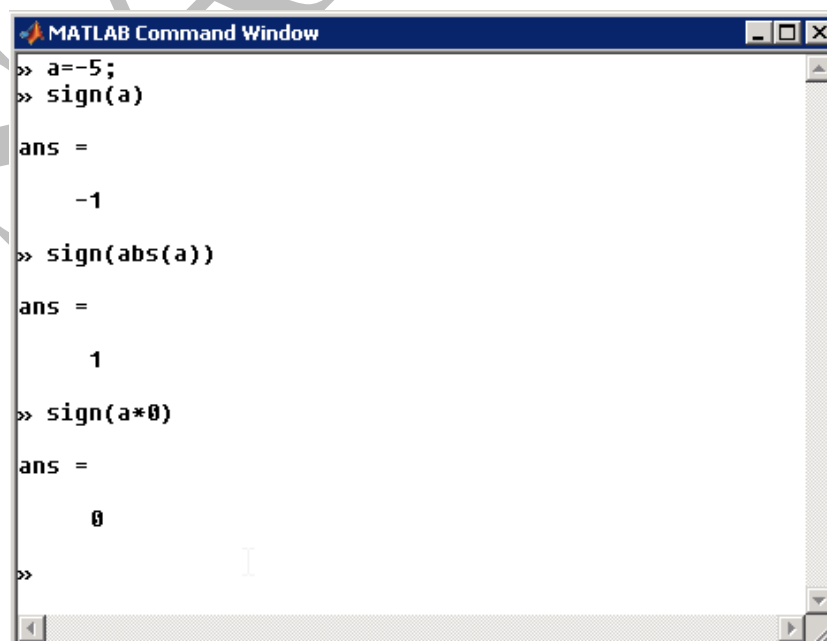
>> rem(a,b)

ans =

    -2

>>
```

We also have the very classic and sometimes useful **signum()** function to get the sign of a function:



```
MATLAB Command Window
>> a=-5;
>> sign(a)

ans =

    -1

>> sign(abs(a))

ans =

     1

>> sign(a*0)

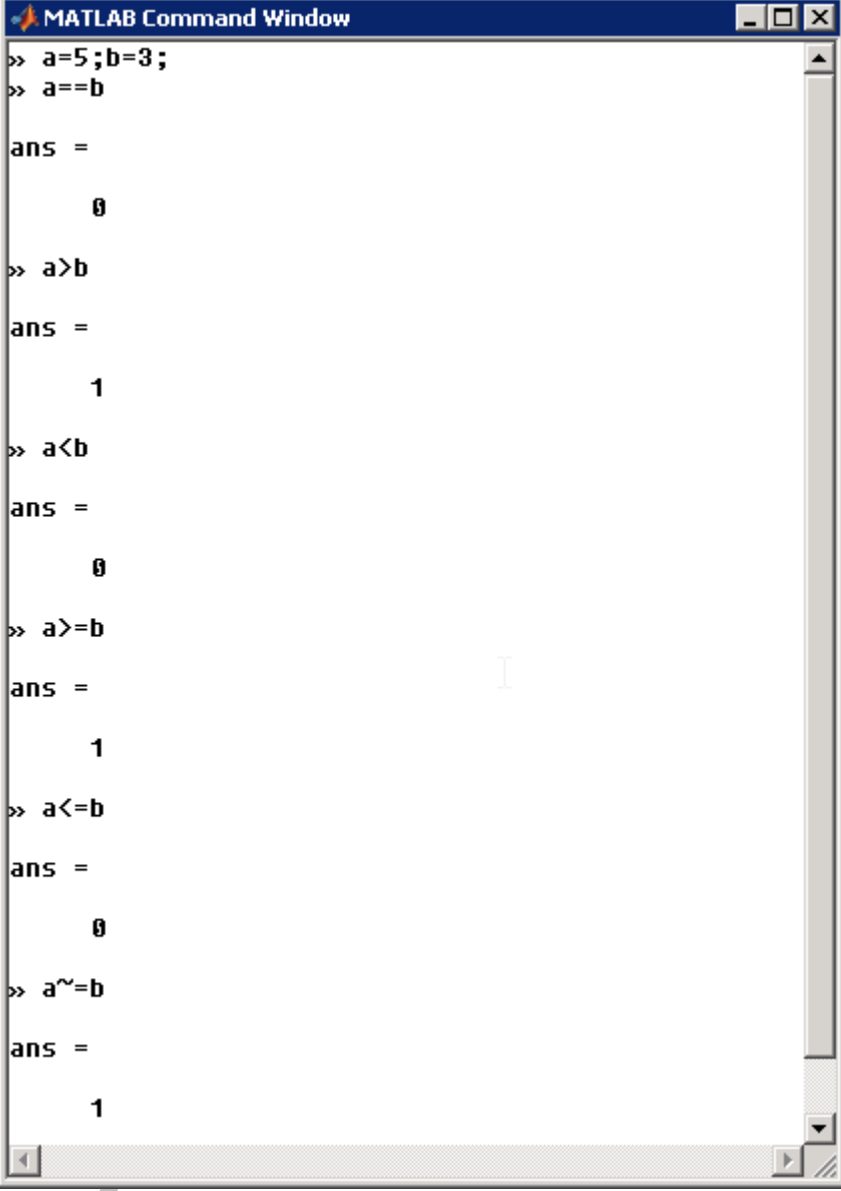
ans =

     0

>>
```

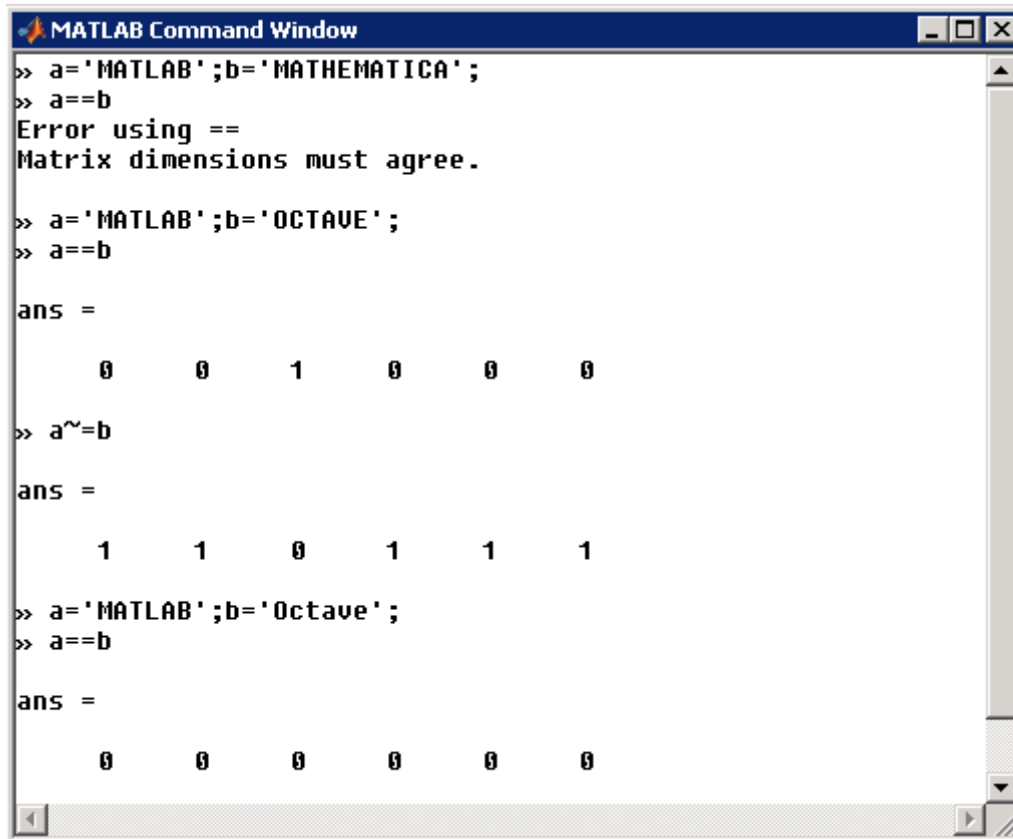
Operators of comparison

All scripting languages have operators of comparisons to compare a lot of different things. Here is the syntax for the MATLAB scripting language operators of comparison:

A screenshot of the MATLAB Command Window. The window has a title bar that says "MATLAB Command Window". Inside, the following commands and their outputs are shown:

```
>> a=5;b=3;  
>> a==b  
  
ans =  
  
    0  
  
>> a>b  
  
ans =  
  
    1  
  
>> a<b  
  
ans =  
  
    0  
  
>> a>=b  
  
ans =  
  
    1  
  
>> a<=b  
  
ans =  
  
    0  
  
>> a~=b  
  
ans =  
  
    1
```

But you try to compare two texts you will get a surprise because in MATLAB everything is matrices (also vectors) in reality texts are also vectors as you can see it below:



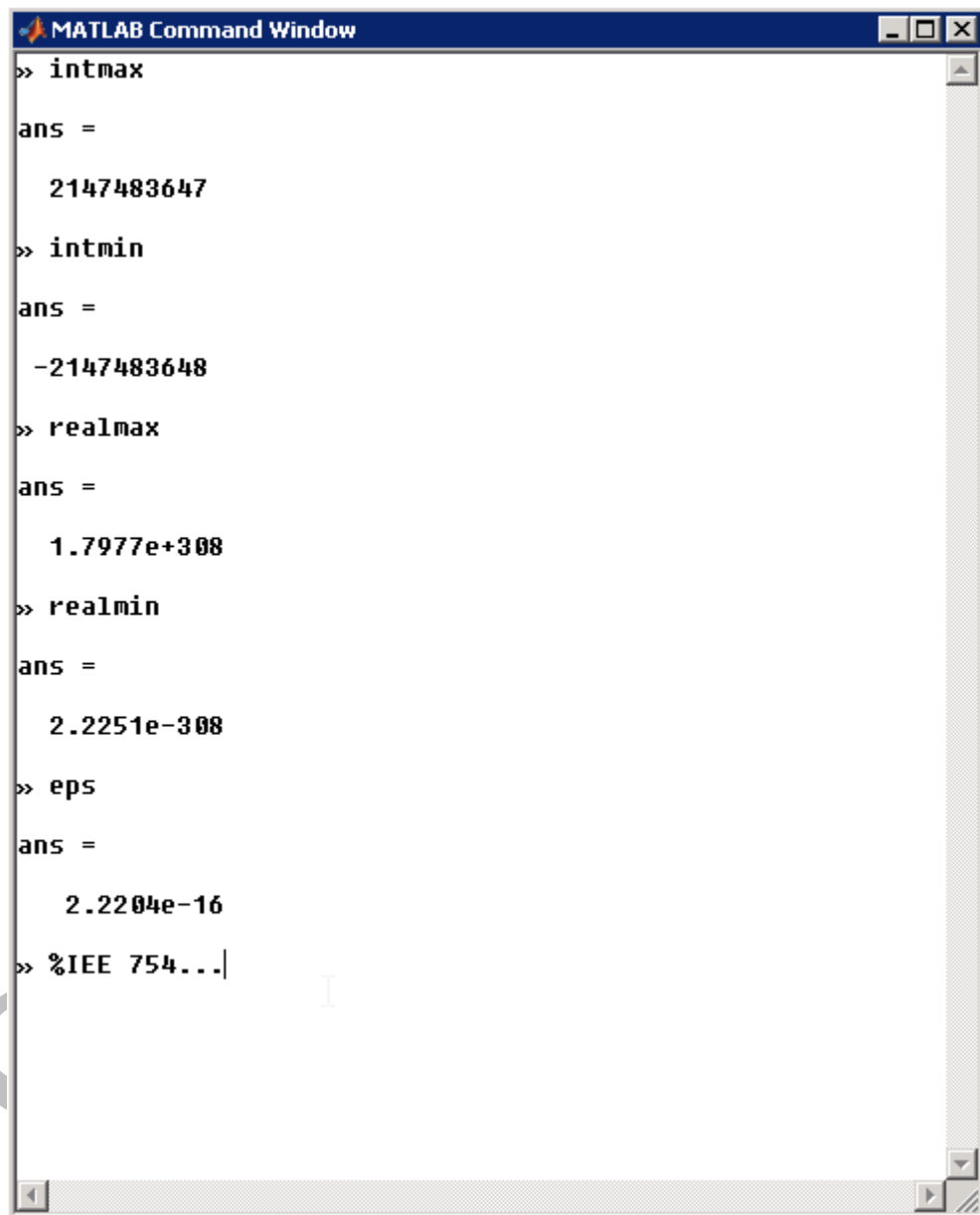
The screenshot shows the MATLAB Command Window with the following commands and outputs:

```
>> a='MATLAB';b='MATHEMATICA';  
>> a==b  
Error using ==  
Matrix dimensions must agree.  
  
>> a='MATLAB';b='OCTAVE';  
>> a==b  
  
ans =  
  
     0     0     1     0     0     0  
  
>> a~=b  
  
ans =  
  
     1     1     0     1     1     1  
  
>> a='MATLAB';b='Octave';  
>> a==b  
  
ans =  
  
     0     0     0     0     0     0
```

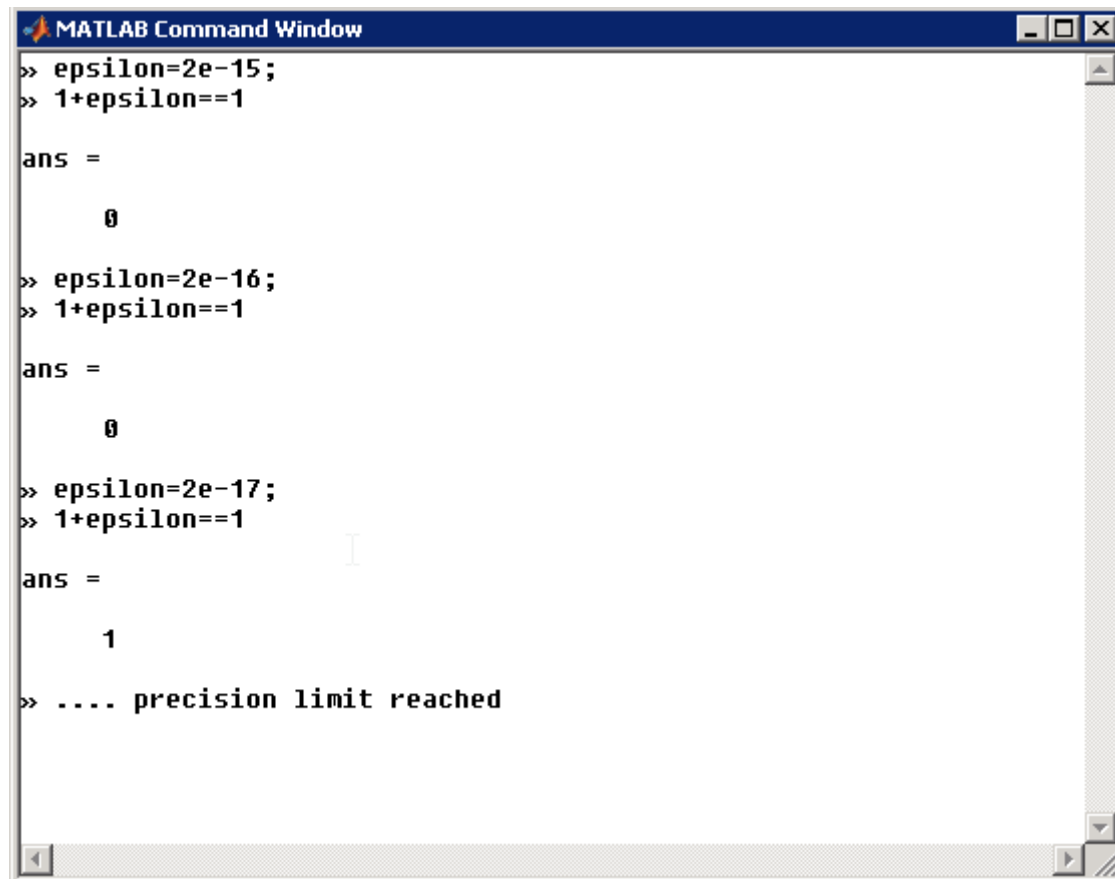
A large, light gray 'SAMPLE' watermark is oriented diagonally across the lower half of the page.

Real numbers manipulations

The first interesting thing when you work with any software is to know the maximum and minimum viewable values that the software can manage. You can get these information in MATLAB by using the commands **intmax**, **intmin**, **realmax**, **realmin**:

A screenshot of the MATLAB Command Window. The window has a blue title bar with the MATLAB logo and the text "MATLAB Command Window". The command history shows the following:
1. Command: `>> intmax`
 Output: `ans = 2147483647`
2. Command: `>> intmin`
 Output: `ans = -2147483648`
3. Command: `>> realmax`
 Output: `ans = 1.7977e+308`
4. Command: `>> realmin`
 Output: `ans = 2.2251e-308`
5. Command: `>> eps`
 Output: `ans = 2.2204e-16`
6. Command: `>> %IEEE 754...`
The cursor is at the end of the last command. The window has standard Windows-style window controls (minimize, maximize, close) in the top right corner and scroll bars on the right and bottom.

But take care! This does not signifies that MATLAB calculates with an accuracy of $2.2251e-308$. In reality he is limited like most software to IEEE double precision as you can see it here below:



```
MATLAB Command Window
>> epsilon=2e-15;
>> 1+epsilon==1

ans =

    0

>> epsilon=2e-16;
>> 1+epsilon==1

ans =

    0

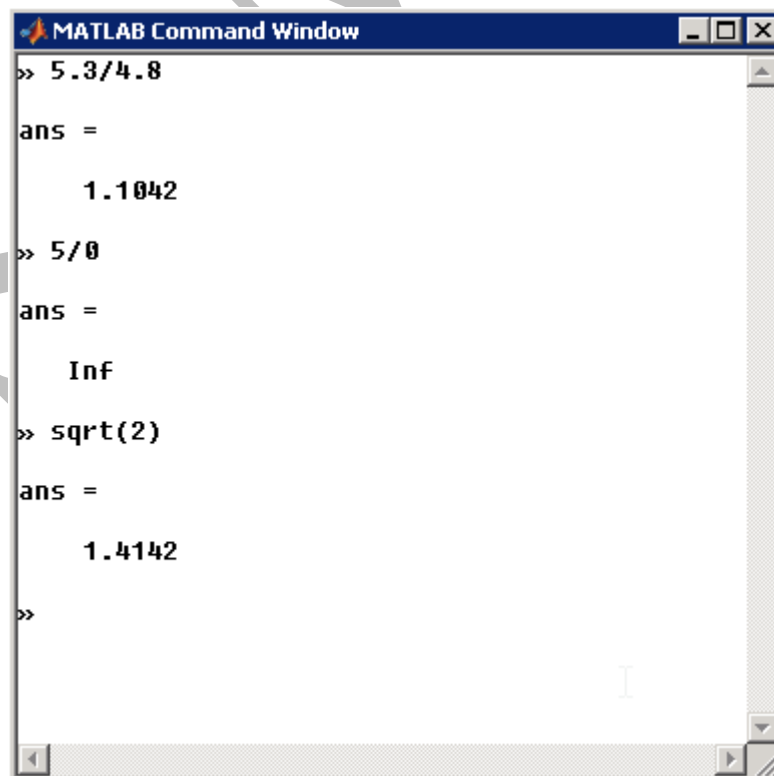
>> epsilon=2e-17;
>> 1+epsilon==1

ans =

    1

>> .... precision limit reached
```

As you can see here below real numbers are otherwise correctly management by MATLAB and we have an important value (that we will use a lot in Optimization toolbox and Symbolic) for infinity that is **Inf** or **-Inf**:



```
MATLAB Command Window
>> 5.3/4.8

ans =

    1.1042

>> 5/0

ans =

    Inf

>> sqrt(2)

ans =

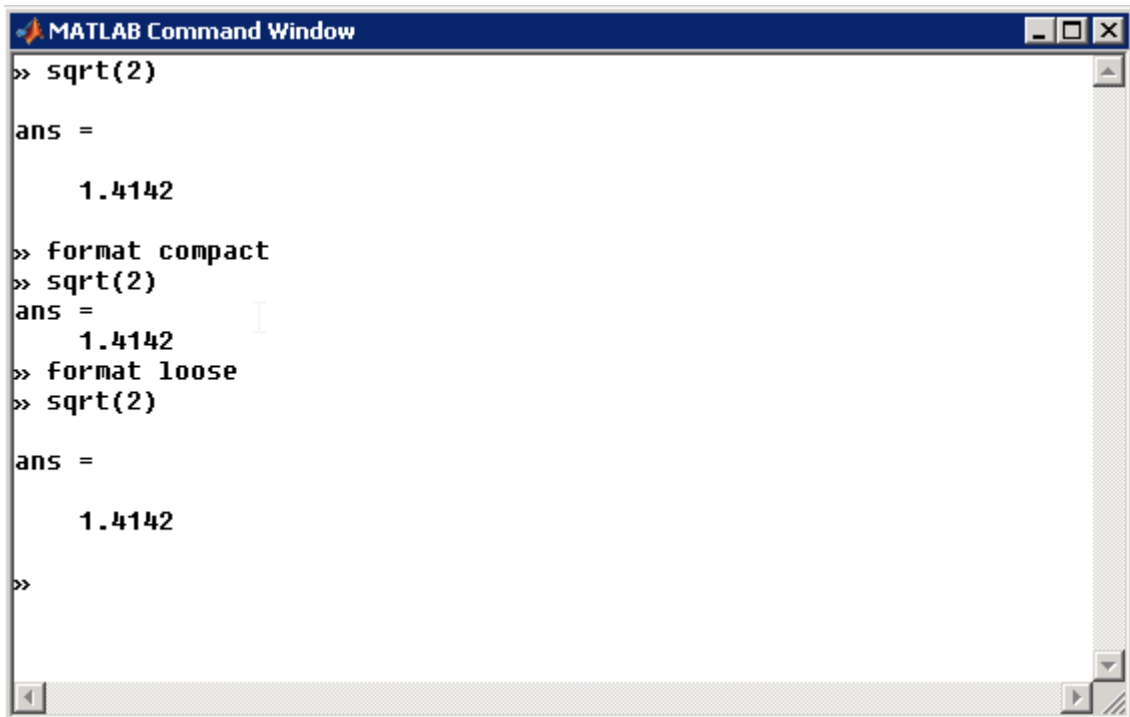
    1.4142

>>
```

Command Windows real numbers formatting and output

When make calculations in MATLAB there is a boring white space between each line. This space is useful when you teach MATLAB but not when you are using it alone at your office.

To remove the white space use the command **format compact**, to reset to the previous state use **format loose**:



```
MATLAB Command Window
>> sqrt(2)

ans =

    1.4142

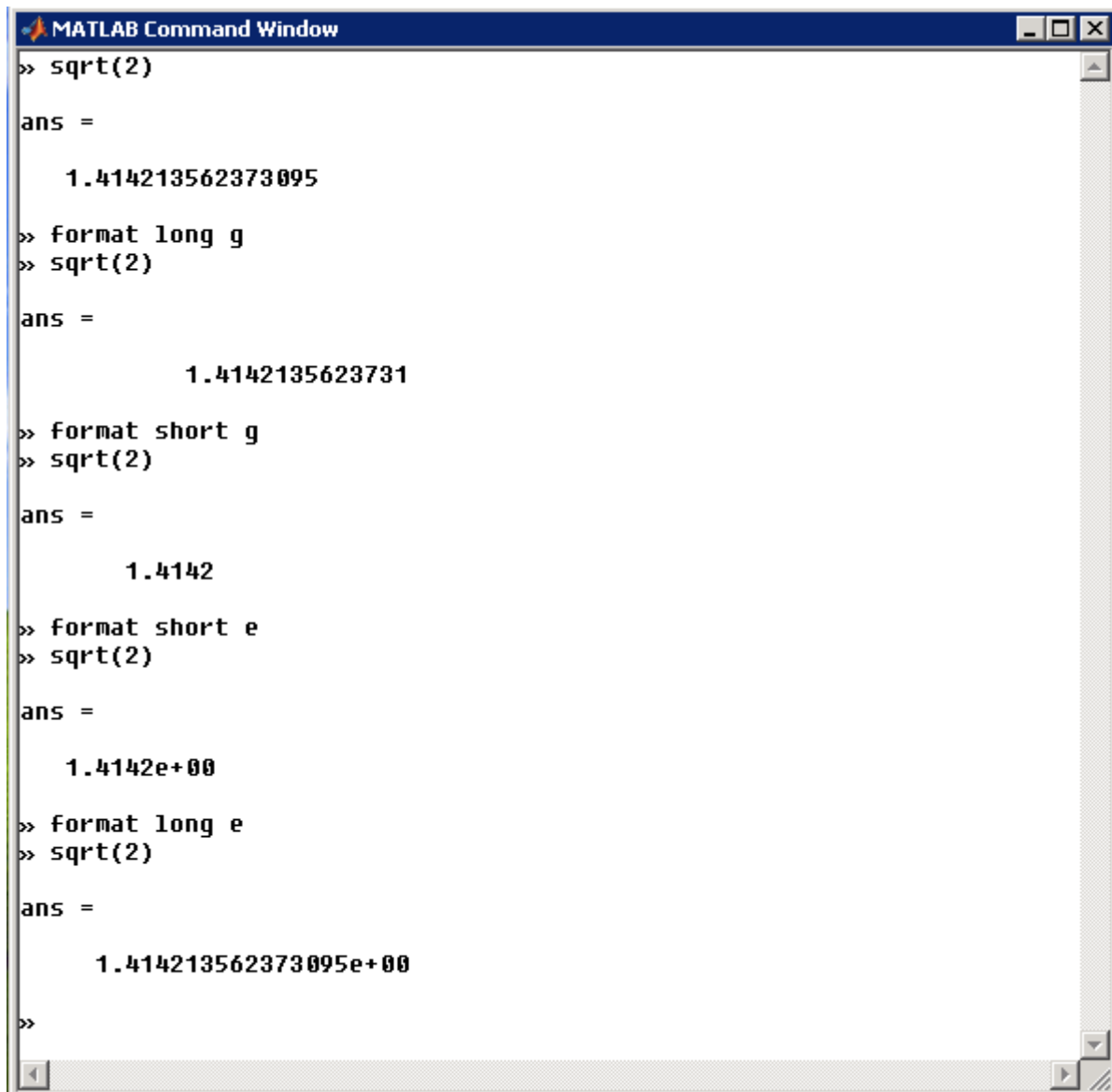
>> format compact
>> sqrt(2)
ans =
    1.4142
>> format loose
>> sqrt(2)

ans =

    1.4142

>>
```

Otherwise you can also choose the precision of number outputs using different commands like **format long g**, **format short g**, **format long e**, **format short e**:



```
MATLAB Command Window

>> sqrt(2)

ans =

    1.414213562373095

>> format long g
>> sqrt(2)

ans =

    1.4142135623731

>> format short g
>> sqrt(2)

ans =

    1.4142

>> format short e
>> sqrt(2)

ans =

    1.4142e+00

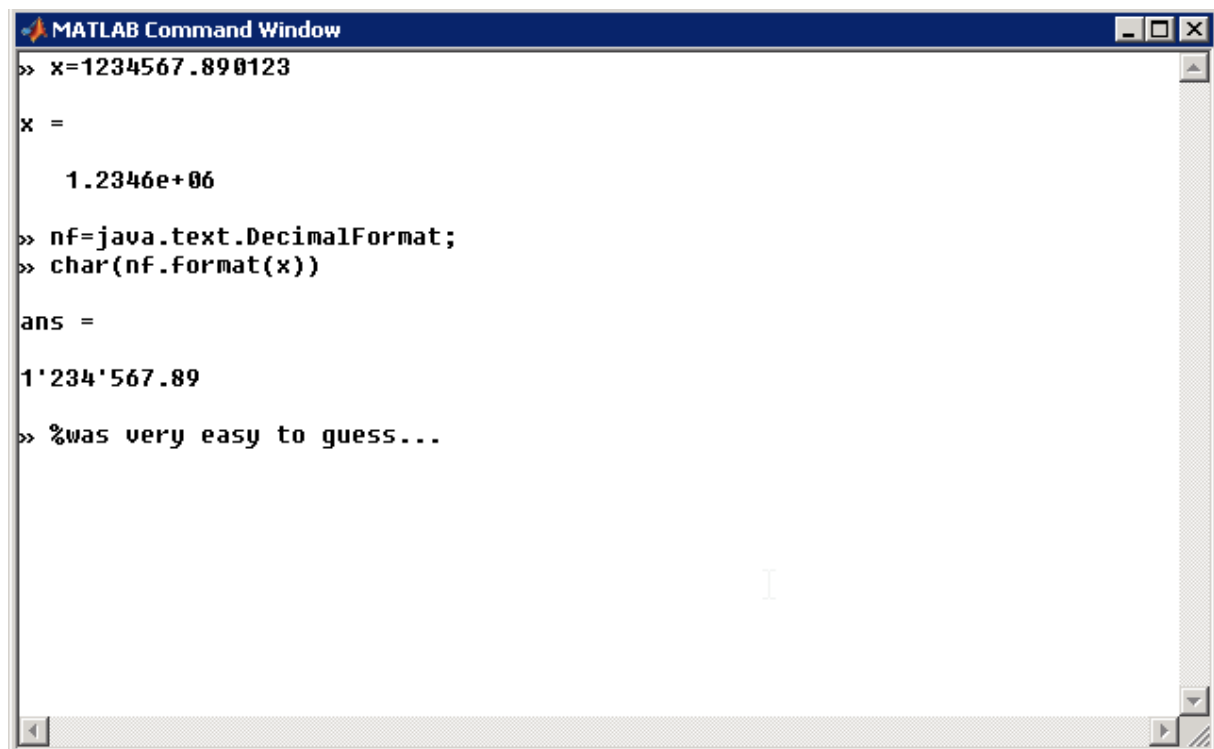
>> format long e
>> sqrt(2)

ans =

    1.414213562373095e+00

>>
```

And what about thousands separators?:



```
MATLAB Command Window
>> x=1234567.890123

x =

    1.2346e+06

>> nf=java.text.DecimalFormat;
>> char(nf.format(x))

ans =

1'234'567.89

>> %was very easy to guess...
```

Arithmetic operations on complex numbers

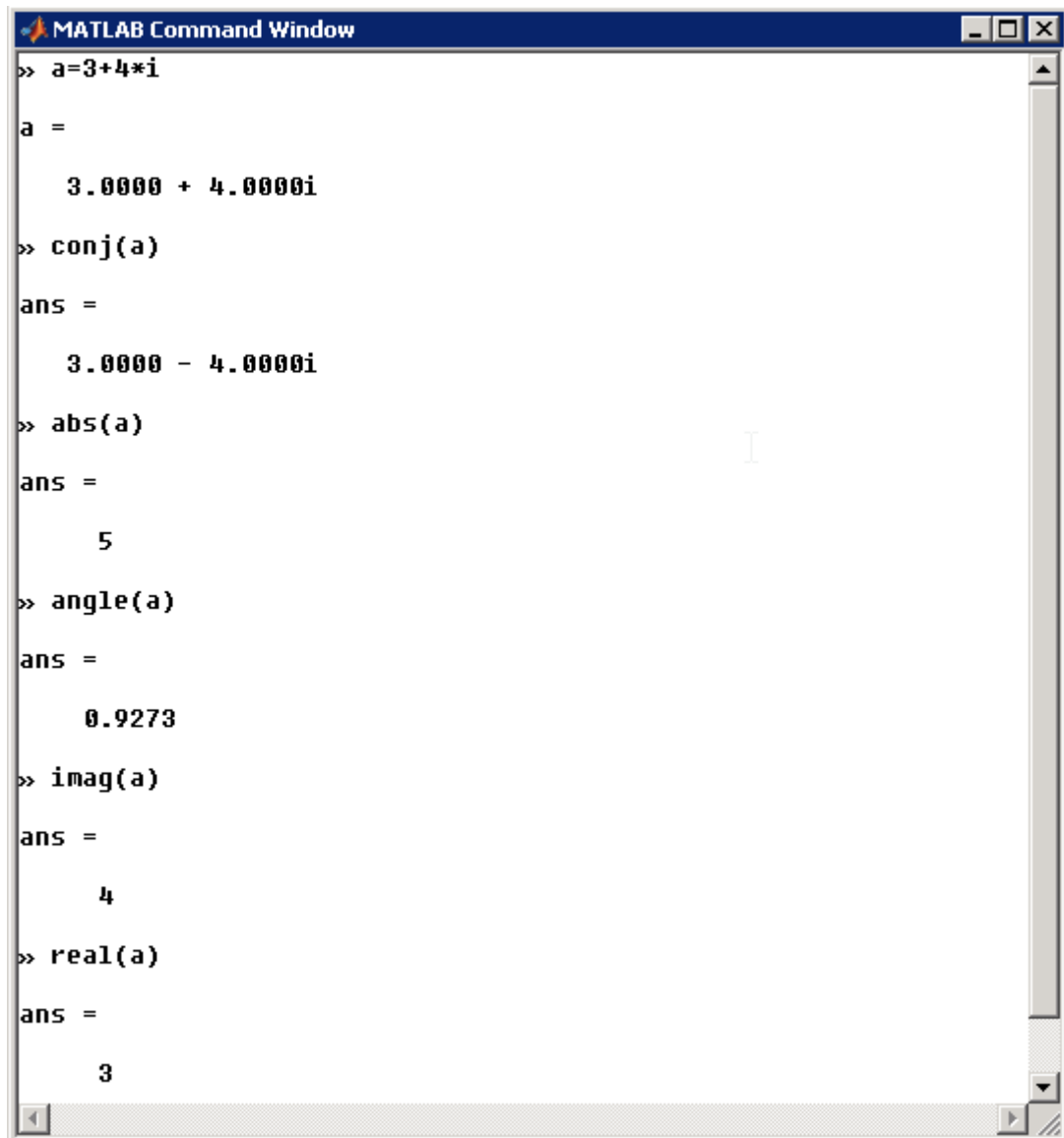
Complex numbers are especially useful in electronics/electrodynamics and physics but you will also have to manage them for example in Finance!

First you must know that MATLAB can use the letters **i** or **j** as complex number imaginary part.

A screenshot of the MATLAB Command Window. The window has a blue title bar with the MATLAB logo and the text "MATLAB Command Window". The command prompt shows two inputs: first, the letter 'i', which results in the output 'ans = 0.0000 + 1.0000i'; second, the letter 'j', which also results in the output 'ans = 0.0000 + 1.0000i'. The window has standard Windows-style window controls (minimize, maximize, close) in the top right corner and a scroll bar on the right side.

```
» i
ans =
    0.0000 + 1.0000i
» j
ans =
    0.0000 + 1.0000i
»
```

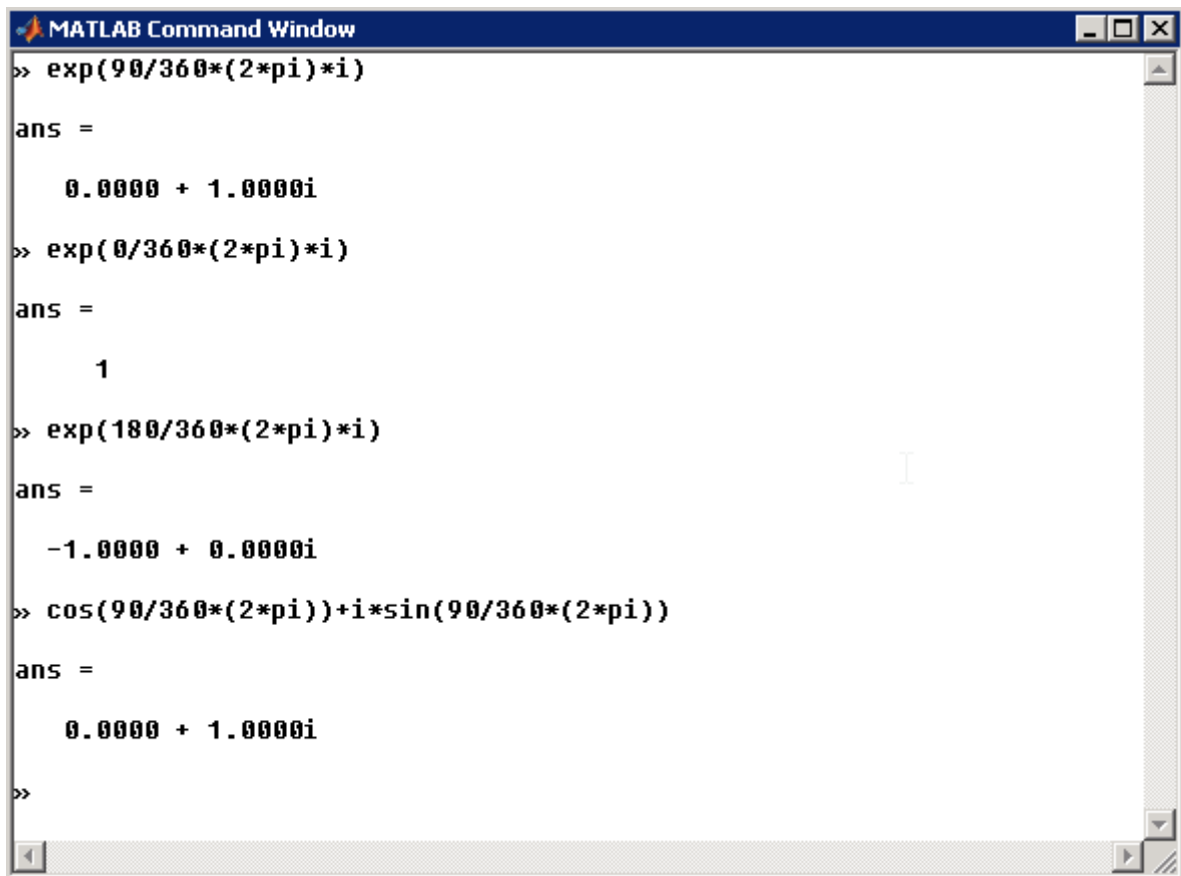
You can otherwise make some very classic calculations on a complex number like addition, multiplication, division, power, root but also get the conjugate with the function **conj()**, the module with the function **abs()**, the angle of the phasor with the function **angle()**, and the imaginary and real part with the functions **imag()** and **real()**:

A screenshot of the MATLAB Command Window. The window has a title bar that says "MATLAB Command Window". Inside, the following commands and their outputs are shown:

```
>> a=3+4*i  
a =  
    3.0000 + 4.0000i  
  
>> conj(a)  
ans =  
    3.0000 - 4.0000i  
  
>> abs(a)  
ans =  
     5  
  
>> angle(a)  
ans =  
    0.9273  
  
>> imag(a)  
ans =  
     4  
  
>> real(a)  
ans =  
     3
```

And for just the classical $+$, $-$, $/$, $*$, $^$ operators can also be applied on complex numbers.

For some charts (especially the **compass!**) it is useful to remember the different technics to get a complex number just from an angle given degrees:

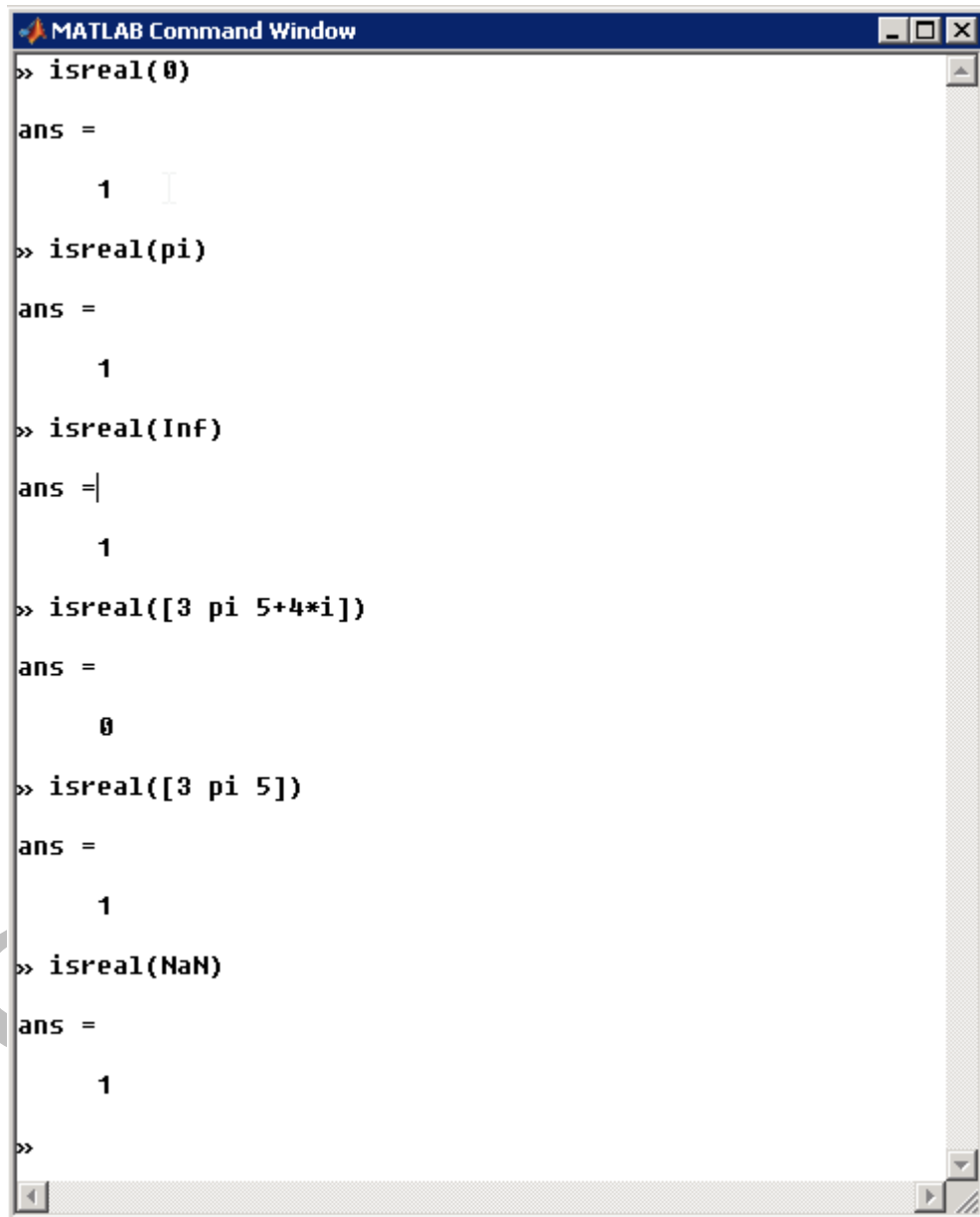
A screenshot of the MATLAB Command Window. The window has a blue title bar with the MATLAB logo and the text "MATLAB Command Window". The command history shows four commands and their corresponding outputs. The first command is `exp(90/360*(2*pi)*i)`, which returns `0.0000 + 1.0000i`. The second command is `exp(0/360*(2*pi)*i)`, which returns `1`. The third command is `exp(180/360*(2*pi)*i)`, which returns `-1.0000 + 0.0000i`. The fourth command is `cos(90/360*(2*pi))+i*sin(90/360*(2*pi))`, which returns `0.0000 + 1.0000i`. The window has a scroll bar on the right and a status bar at the bottom.

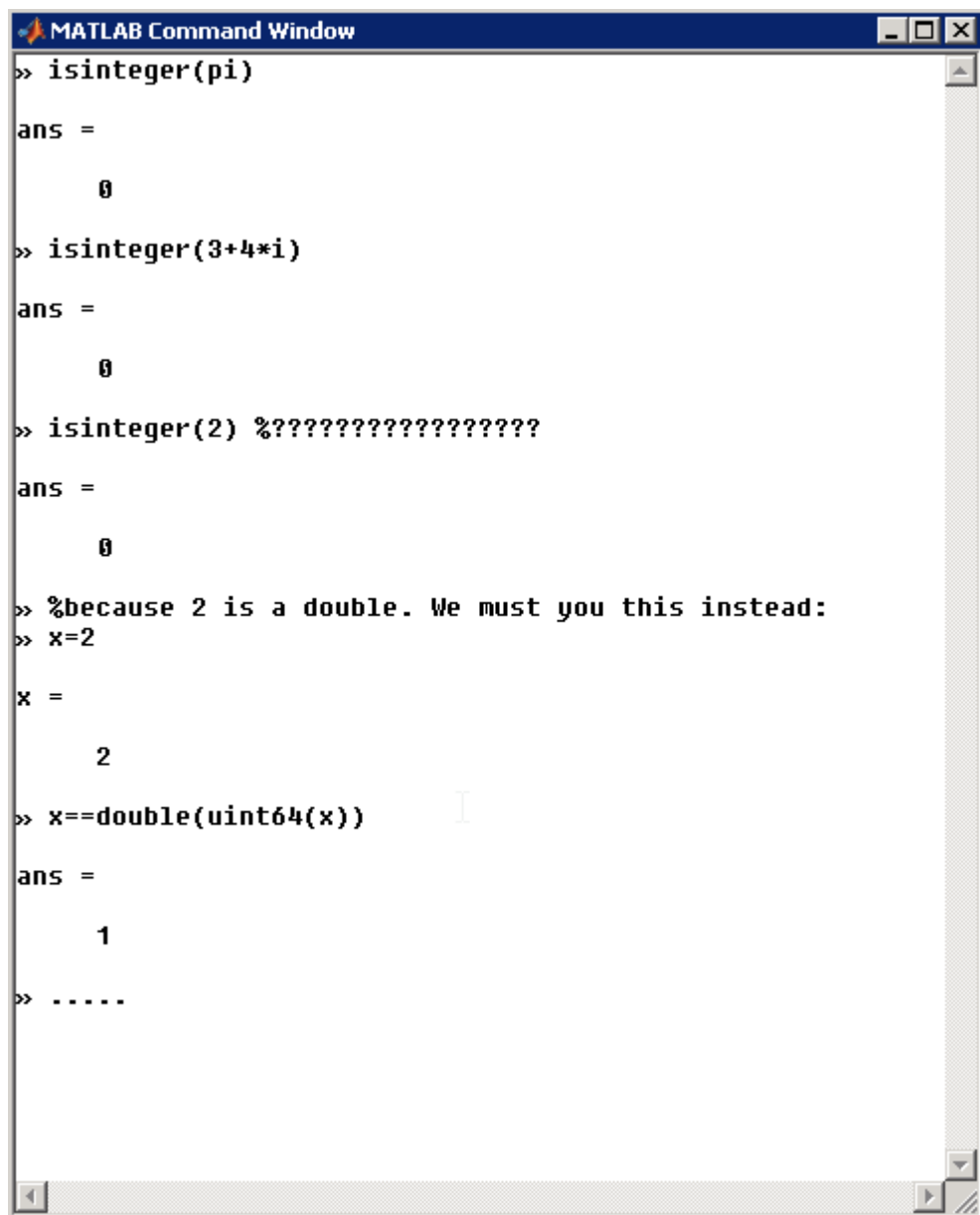
```
MATLAB Command Window
>> exp(90/360*(2*pi)*i)
ans =
    0.0000 + 1.0000i
>> exp(0/360*(2*pi)*i)
ans =
     1
>> exp(180/360*(2*pi)*i)
ans =
   -1.0000 + 0.0000i
>> cos(90/360*(2*pi))+i*sin(90/360*(2*pi))
ans =
    0.0000 + 1.0000i
>>
```


Identify the number type

Many times you will have to face with functions that will return multiple solutions. It is then necessary to know how to determine if a result is a real, integer or complex.

For this purpose you can use the functions **isreal()** or **isinteger()** on scalar or vectors as shown below (**iscomplex()** does not exist!):

A screenshot of the MATLAB Command Window. The window title is "MATLAB Command Window". The command history shows several calls to the `isreal` function. The results are as follows:
- `>> isreal(0)`
 `ans =`
 `1`
- `>> isreal(pi)`
 `ans =`
 `1`
- `>> isreal(Inf)`
 `ans =`
 `1`
- `>> isreal([3 pi 5+4*i])`
 `ans =`
 `0`
- `>> isreal([3 pi 5])`
 `ans =`
 `1`
- `>> isreal(NaN)`
 `ans =`
 `1`
The prompt `>>` is visible at the bottom of the window.

A screenshot of the MATLAB Command Window. The window has a blue title bar with the MATLAB logo and the text "MATLAB Command Window". The command history shows several commands and their outputs. The first command is `isinteger(pi)`, which returns `ans = 0`. The second command is `isinteger(3+4\*i)`, which also returns `ans = 0`. The third command is `isinteger(2) %????????????????????`, which returns `ans = 0`. The fourth command is a comment: `%%because 2 is a double. We must use this instead:`, followed by `x=2`, which returns `x = 2`. The fifth command is `x==double(uint64(x))`, which returns `ans = 1`. The sixth command is `.....`.

```
MATLAB Command Window

>> isinteger(pi)

ans =

    0

>> isinteger(3+4*i)

ans =

    0

>> isinteger(2) %????????????????????

ans =

    0

>> %%because 2 is a double. We must use this instead:
>> x=2

x =

    2

>> x==double(uint64(x))

ans =

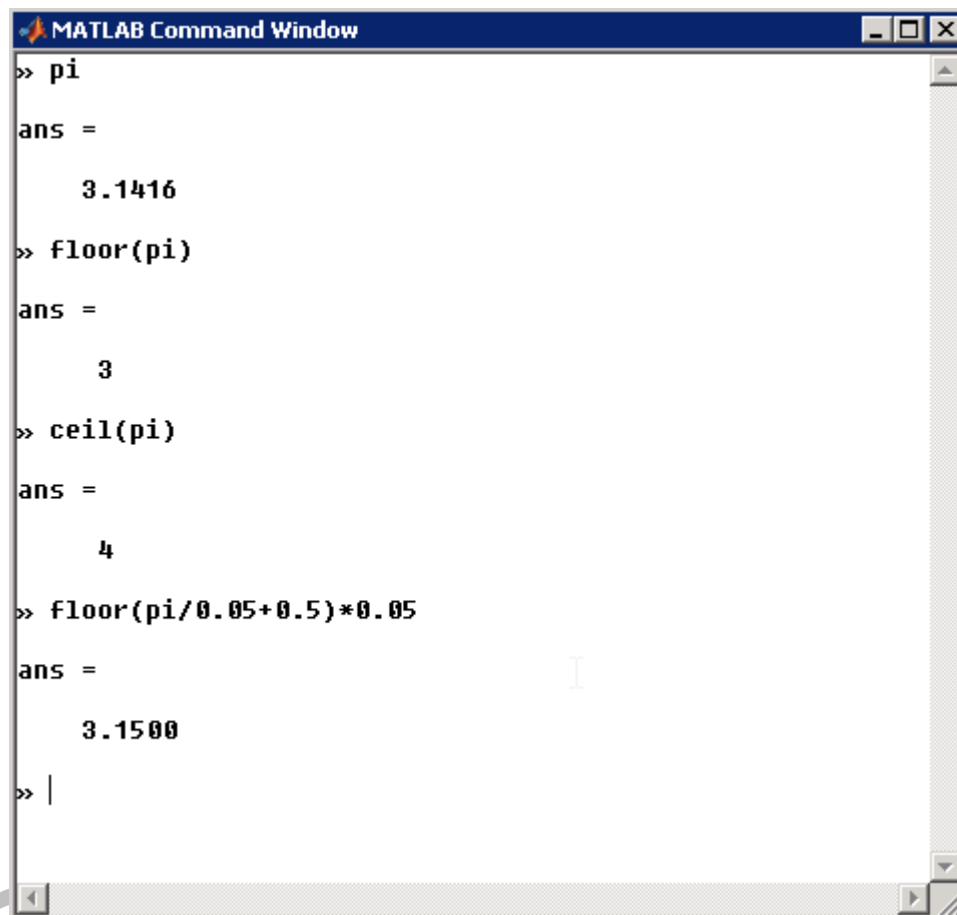
    1

>> .....
```

Rounding results

Rounding results is always something important if you want to generate a nice output on a chart or in an automated report.

For this purpose MATLAB has two important functions **ceil()** and **floor()** that can also be used with the famous equation to round to any multiple we want (in the last example below we round to the nearest 0.05):

A screenshot of the MATLAB Command Window. The window has a title bar that says "MATLAB Command Window". Inside, the command prompt shows several commands and their outputs. The commands are: 'pi', 'floor(pi)', 'ceil(pi)', and 'floor(pi/0.05+0.5)*0.05'. The outputs are: 'ans = 3.1416', 'ans = 3', 'ans = 4', and 'ans = 3.1500'. The cursor is at the end of the last command line.

```
>> pi
ans =
    3.1416

>> floor(pi)
ans =
     3

>> ceil(pi)
ans =
     4

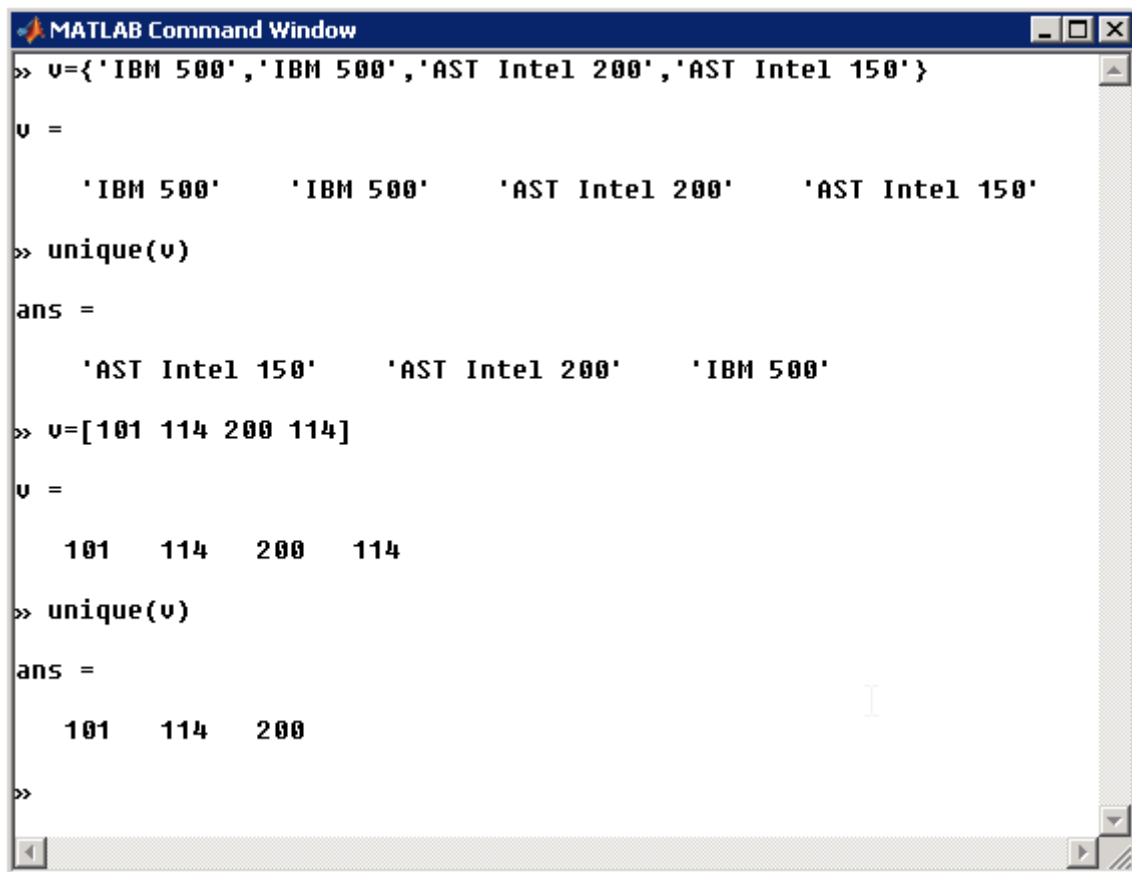
>> floor(pi/0.05+0.5)*0.05
ans =
    3.1500

>> |
```

Set (vectors) operations

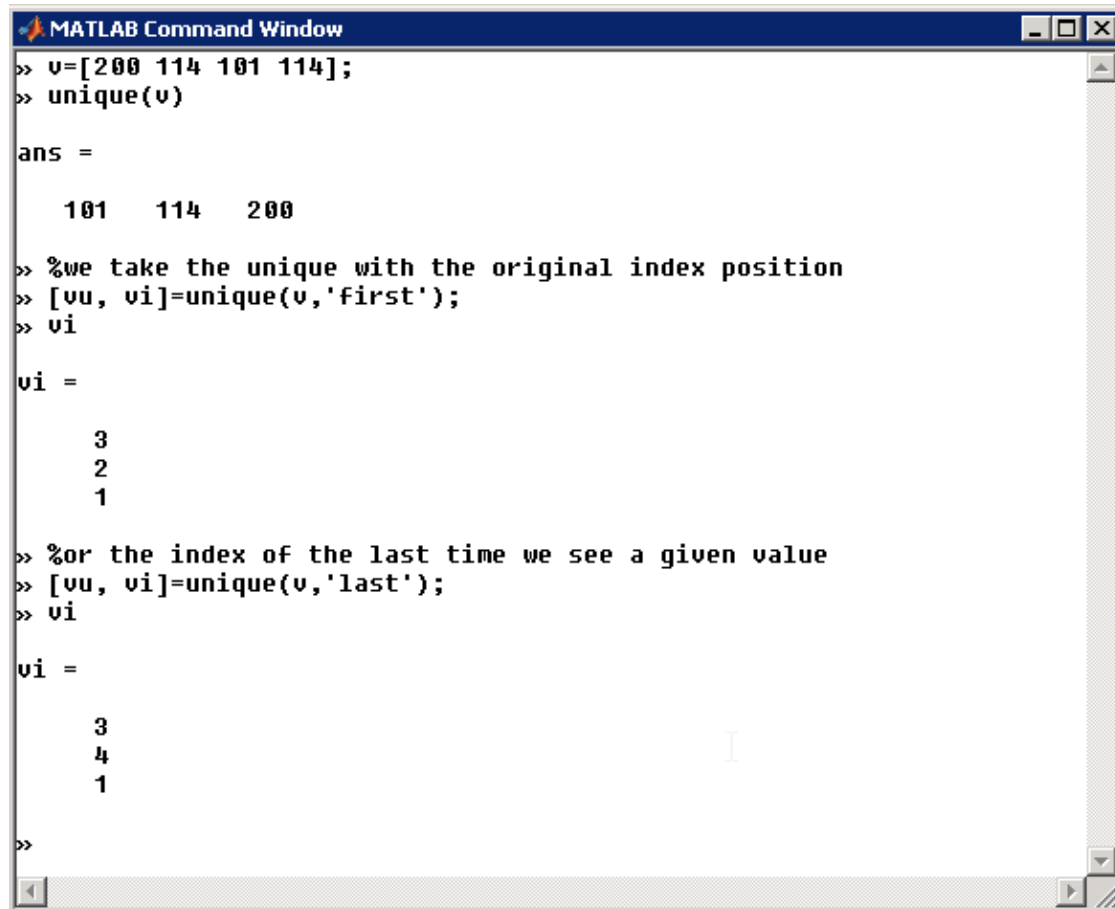
Numerical vectors and string vectors (also name "cell arrays" in MATLAB") are one of the most important objects that any user has to master. Before making calculations with such objects one important prior knowledge to have is to be able to apply set operations on them.

First see how to define a simple string vector (cell array) and a numerical (line) vector and how to extract unique items using the **unique()** function:

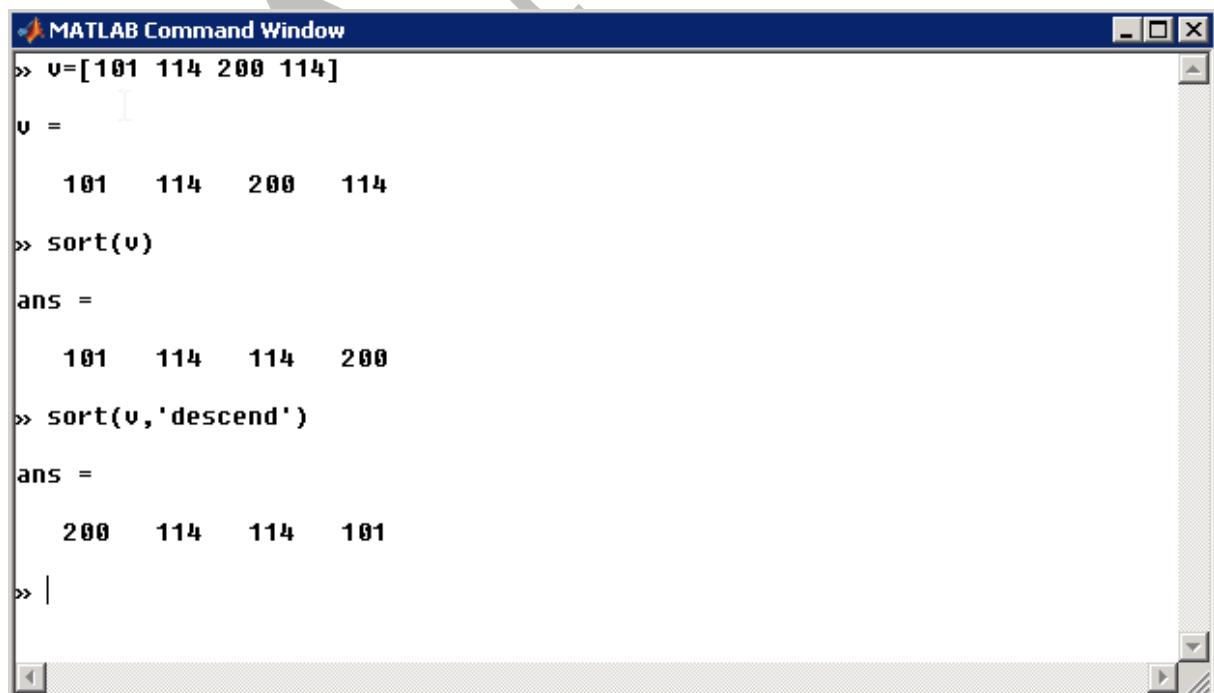


```
MATLAB Command Window
>> v={'IBM 500','IBM 500','AST Intel 200','AST Intel 150'}
v =
    'IBM 500'    'IBM 500'    'AST Intel 200'    'AST Intel 150'
>> unique(v)
ans =
    'AST Intel 150'    'AST Intel 200'    'IBM 500'
>> v=[101 114 200 114]
v =
    101    114    200    114
>> unique(v)
ans =
    101    114    200
>>
```

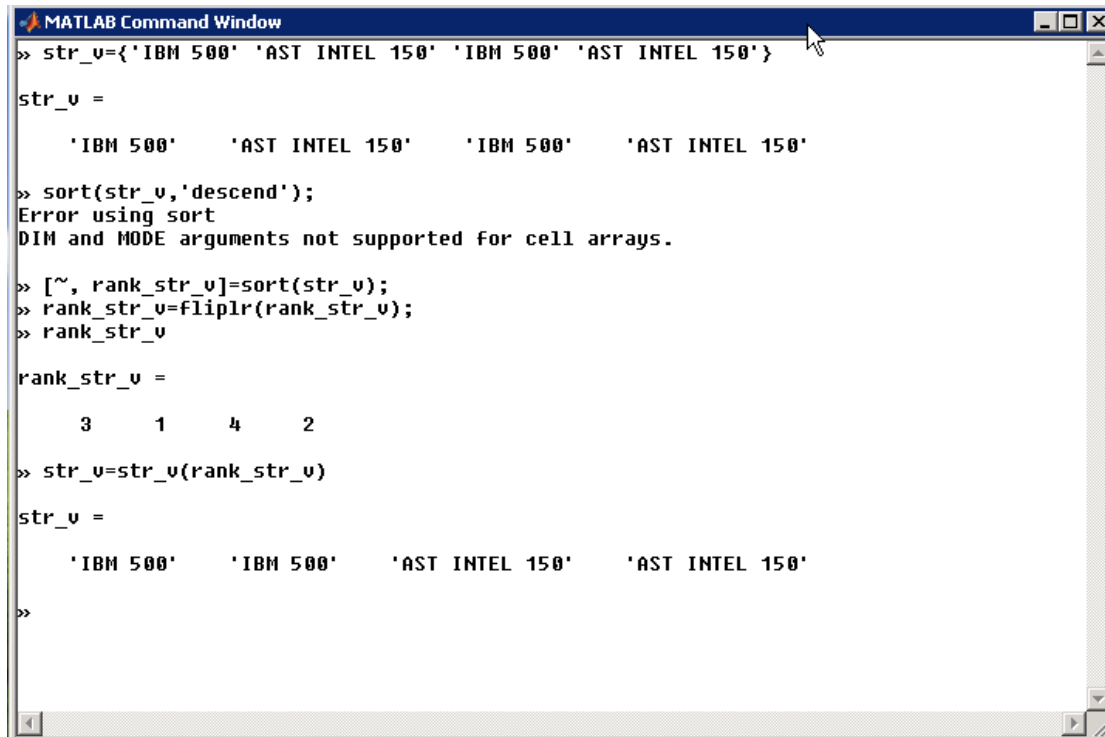
But in facts, the function **unique()** is one of this numerous MATLAB fuctions that returns multiple output. For example we can see below hot to get also the ranks:

A screenshot of the MATLAB Command Window. The window has a blue title bar with the MATLAB logo and the text "MATLAB Command Window". The command history shows: 1. Input: v=[200 114 101 114]; 2. Input: unique(v) 3. Output: ans = 101 114 200 4. Input: %we take the unique with the original index position 5. Input: [vu, vi]=unique(v,'first'); 6. Input: vi 7. Output: vi = 3 2 1 8. Input: %or the index of the last time we see a given value 9. Input: [vu, vi]=unique(v,'last'); 10. Input: vi 11. Output: vi = 3 4 1 12. Input: >> The window has standard Windows-style window controls (minimize, maximize, close) in the top right corner and scroll bars on the right and bottom.

We can also sort in a very easy way numerical vector using the **sort()** function:

A screenshot of the MATLAB Command Window. The window has a blue title bar with the MATLAB logo and the text "MATLAB Command Window". The command history shows: 1. Input: v=[101 114 200 114] 2. Output: v = 101 114 200 114 3. Input: sort(v) 4. Output: ans = 101 114 114 200 5. Input: sort(v,'descend') 6. Output: ans = 200 114 114 101 7. Input: >> | The window has standard Windows-style window controls (minimize, maximize, close) in the top right corner and scroll bars on the right and bottom.

But sorting string vectors (cell array) **in a descending order** is a little bit more complicated. We have to play with multiple functions using **sort()** and **fliplr()** together to inverse the sorting order of the ranks and then use it to get the final string vector:



```

MATLAB Command Window
>> str_v={'IBM 500' 'AST INTEL 150' 'IBM 500' 'AST INTEL 150'}

str_v =

    'IBM 500'    'AST INTEL 150'    'IBM 500'    'AST INTEL 150'

>> sort(str_v,'descend');
Error using sort
DIM and MODE arguments not supported for cell arrays.

>> [~, rank_str_v]=sort(str_v);
>> rank_str_v=fliplr(rank_str_v);
>> rank_str_v

rank_str_v =

     3     1     4     2

>> str_v=str_v(rank_str_v)

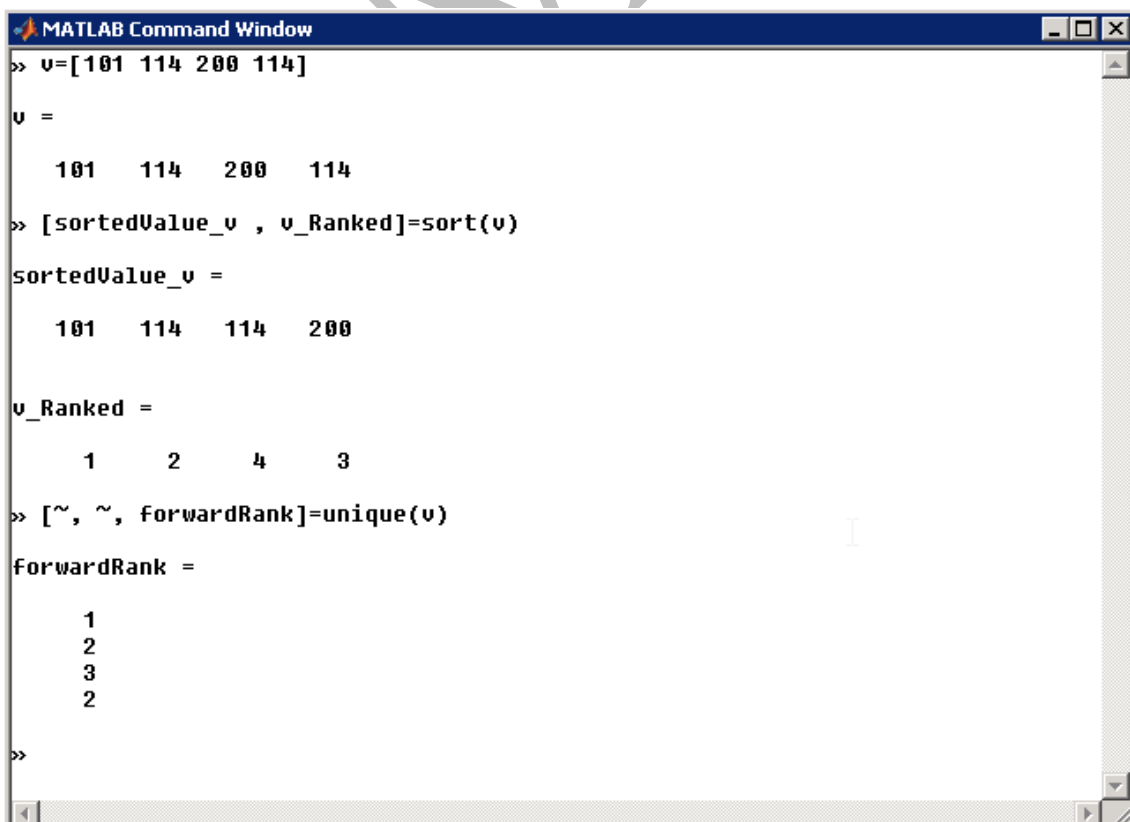
str_v =

    'IBM 500'    'IBM 500'    'AST INTEL 150'    'AST INTEL 150'

>>

```

An interesting thing in rankings is to manage equalities. See how MATLAB can manage that with the function **unique()**:



```

MATLAB Command Window
>> v=[101 114 200 114]

v =

    101    114    200    114

>> [sortedValue_v , v_Ranked]=sort(v)

sortedValue_v =

    101    114    114    200

v_Ranked =

     1     2     4     3

>> [~, ~, forwardRank]=unique(v)

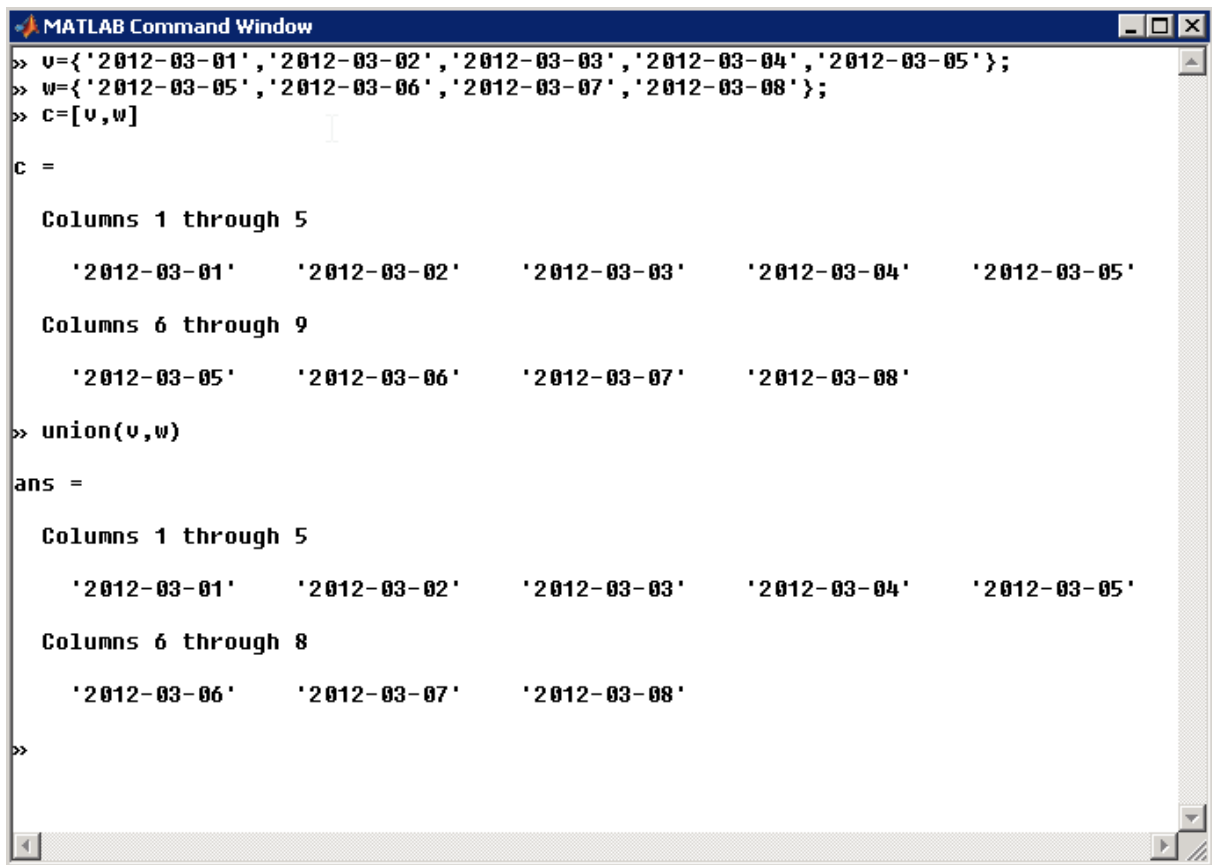
forwardRank =

     1
     2
     3
     2

>>

```

Something also important with vectors (or cell array) is to learn to merge them or to make a union set operation using the function **union()** to avoid duplicates:



```

>> v={'2012-03-01','2012-03-02','2012-03-03','2012-03-04','2012-03-05'};
>> w={'2012-03-05','2012-03-06','2012-03-07','2012-03-08'};
>> c=[v,w]

c =

Columns 1 through 5

    '2012-03-01'    '2012-03-02'    '2012-03-03'    '2012-03-04'    '2012-03-05'

Columns 6 through 9

    '2012-03-05'    '2012-03-06'    '2012-03-07'    '2012-03-08'

>> union(v,w)

ans =

Columns 1 through 5

    '2012-03-01'    '2012-03-02'    '2012-03-03'    '2012-03-04'    '2012-03-05'

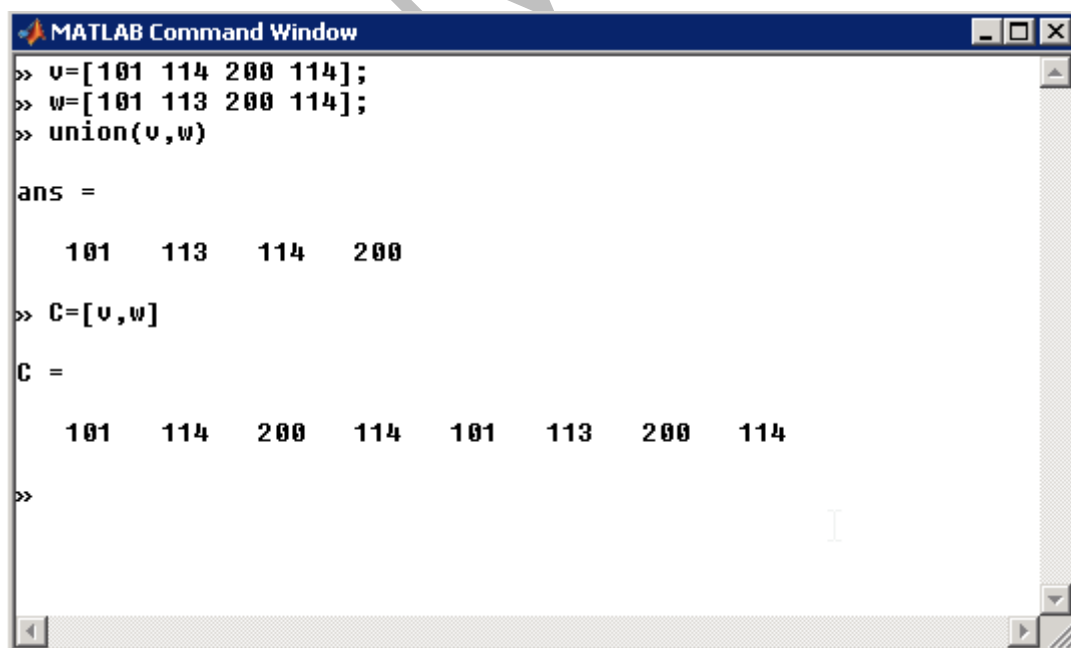
Columns 6 through 8

    '2012-03-06'    '2012-03-07'    '2012-03-08'

>>

```

That works also the same for numeric vectors:



```

>> v=[101 114 200 114];
>> w=[101 113 200 114];
>> union(v,w)

ans =

    101    113    114    200

>> C=[v,w]

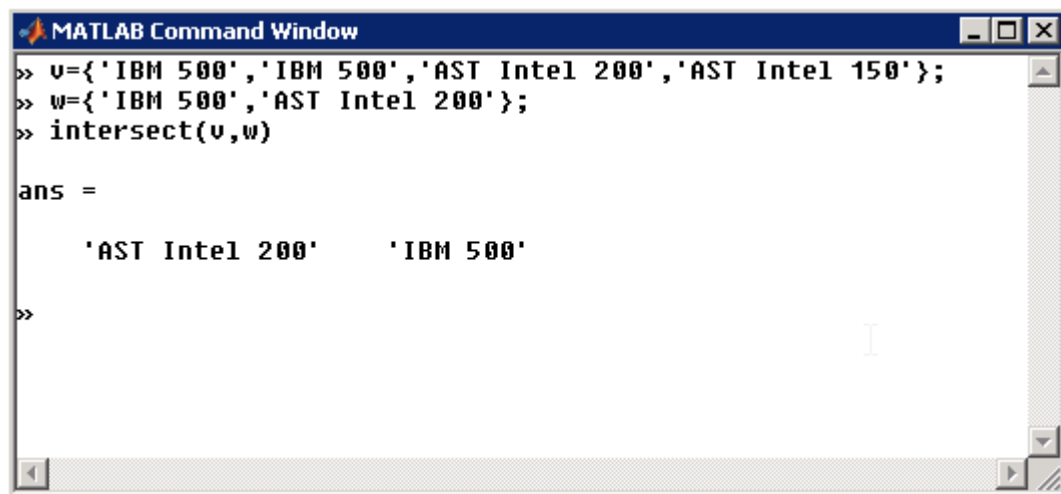
C =

    101    114    200    114    101    113    200    114

>>

```

If you are interested to get the similarities between a given number of vector (numeric or strings) you can use the set operation function **intersect()** as shown below:

A screenshot of the MATLAB Command Window. The title bar reads "MATLAB Command Window". The command prompt shows the following code:

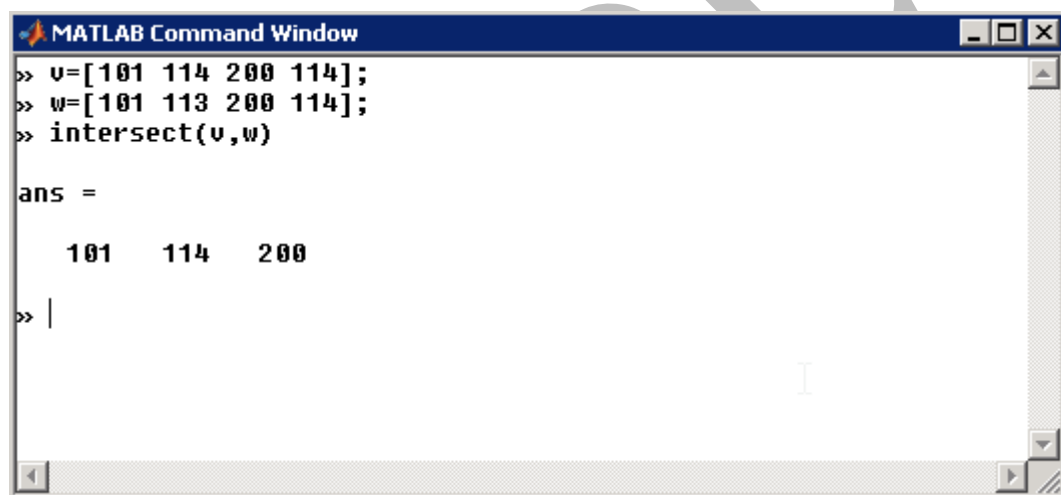
```
>> v={'IBM 500','IBM 500','AST Intel 200','AST Intel 150'};  
>> w={'IBM 500','AST Intel 200'};  
>> intersect(v,w)
```

 The output is displayed as:

```
ans =  
  
    'AST Intel 200'    'IBM 500'
```

 The cursor is at the end of the second line of the output.

And here we see that it works the same for numerical vectors:

A screenshot of the MATLAB Command Window. The title bar reads "MATLAB Command Window". The command prompt shows the following code:

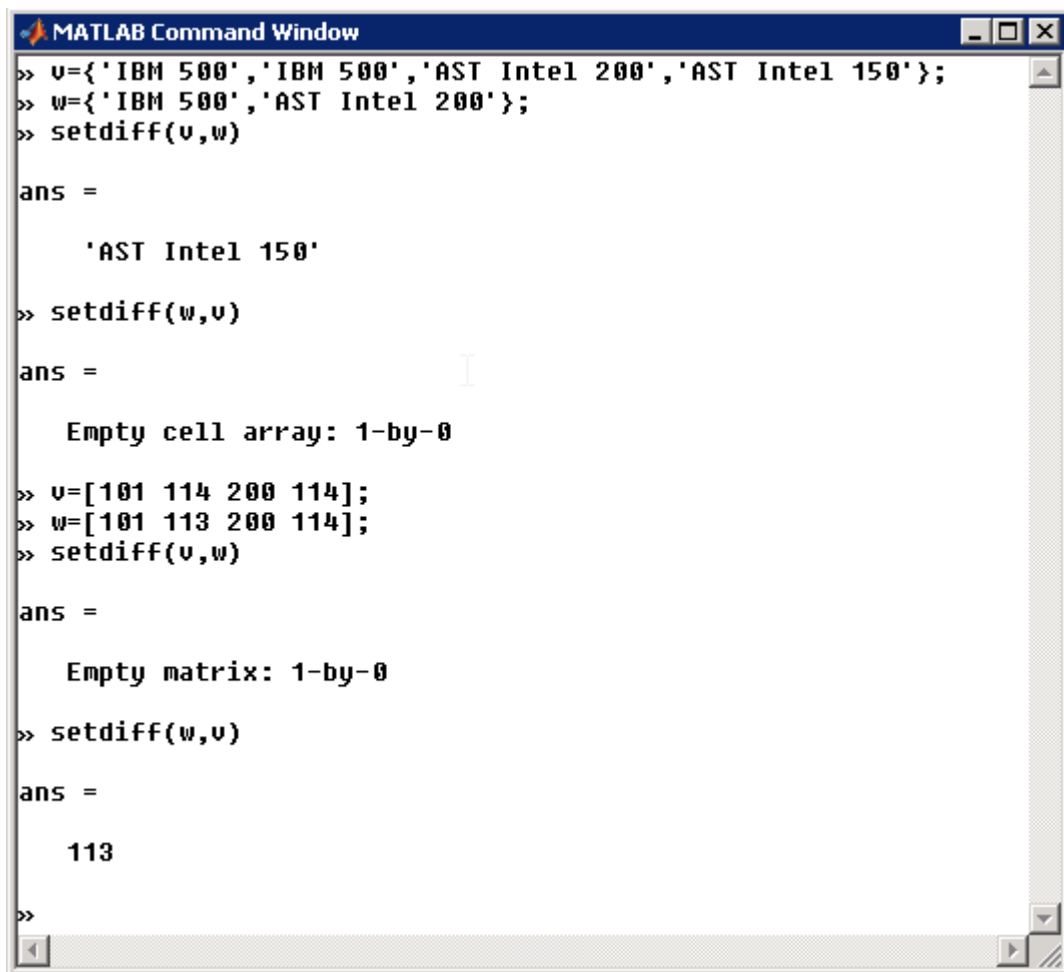
```
>> v=[101 114 200 114];  
>> w=[101 113 200 114];  
>> intersect(v,w)
```

 The output is displayed as:

```
ans =  
  
    101    114    200
```

 The cursor is at the end of the second line of the output.

When we have two vectors (numeric or string) we can be interested to know what differs between them we can use the `setdiff()` set function. **Don't forget that it is not a commutative operator!:**



```
MATLAB Command Window
>> v={'IBM 500','IBM 500','AST Intel 200','AST Intel 150'};
>> w={'IBM 500','AST Intel 200'};
>> setdiff(v,w)

ans =

    'AST Intel 150'

>> setdiff(w,v)

ans =

Empty cell array: 1-by-0

>> v=[101 114 200 114];
>> w=[101 113 200 114];
>> setdiff(v,w)

ans =

Empty matrix: 1-by-0

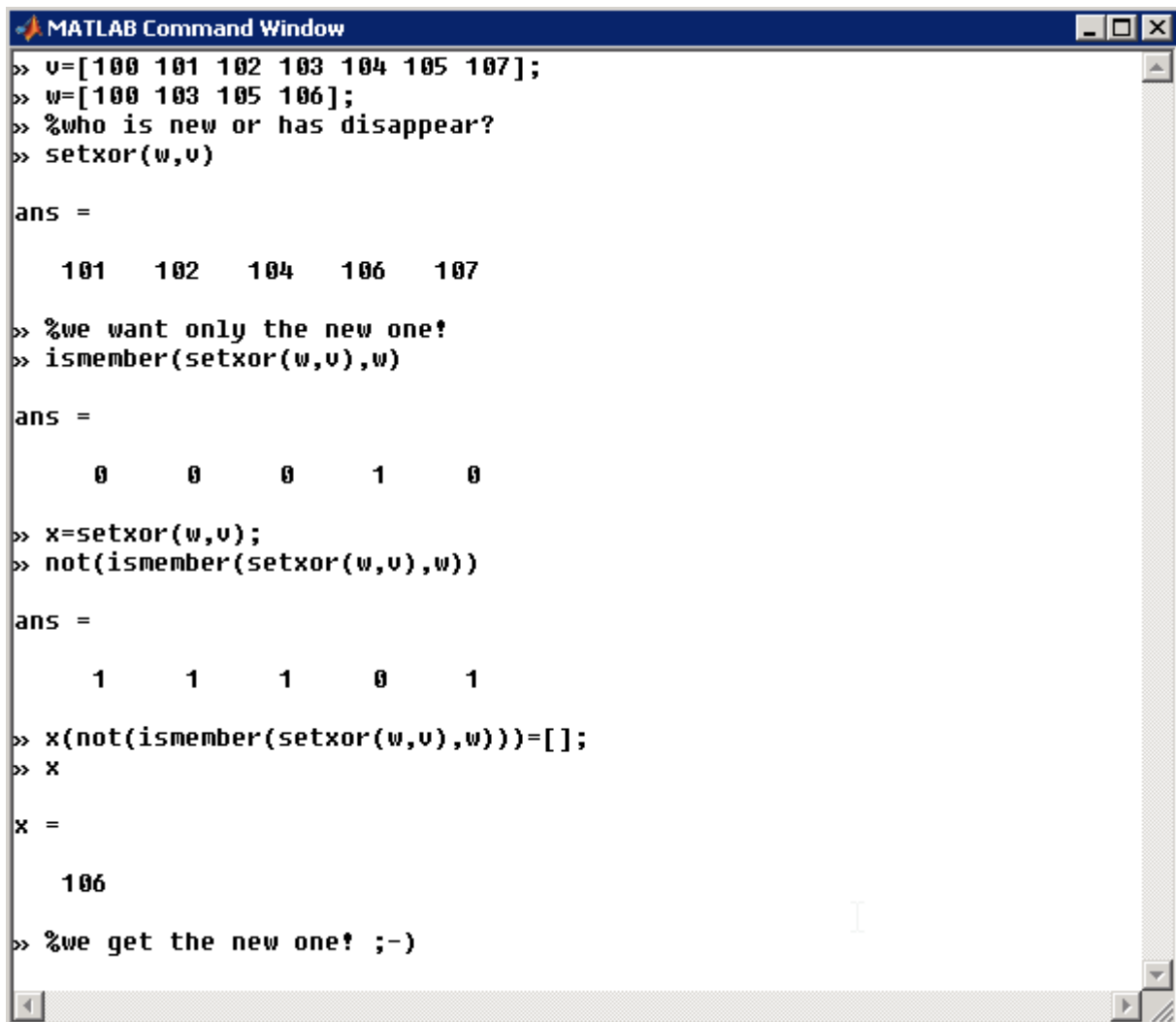
>> setdiff(w,v)

ans =

    113

>>
```

We can also use the **setxor()** set operator function to investigate what has change between two vectors (numeric or string) as you can see below:



```
MATLAB Command Window
>> u=[100 101 102 103 104 105 107];
>> w=[100 103 105 106];
>> %who is new or has disappear?
>> setxor(w,u)

ans =

    101    102    104    106    107

>> %we want only the new one!
>> ismember(setxor(w,u),w)

ans =

     0     0     0     1     0

>> x=setxor(w,u);
>> not(ismember(setxor(w,u),w))

ans =

     1     1     1     0     1

>> x(not(ismember(setxor(w,u),w)))=[];
>> x

x =

    106

>> %we get the new one! ;-)
```

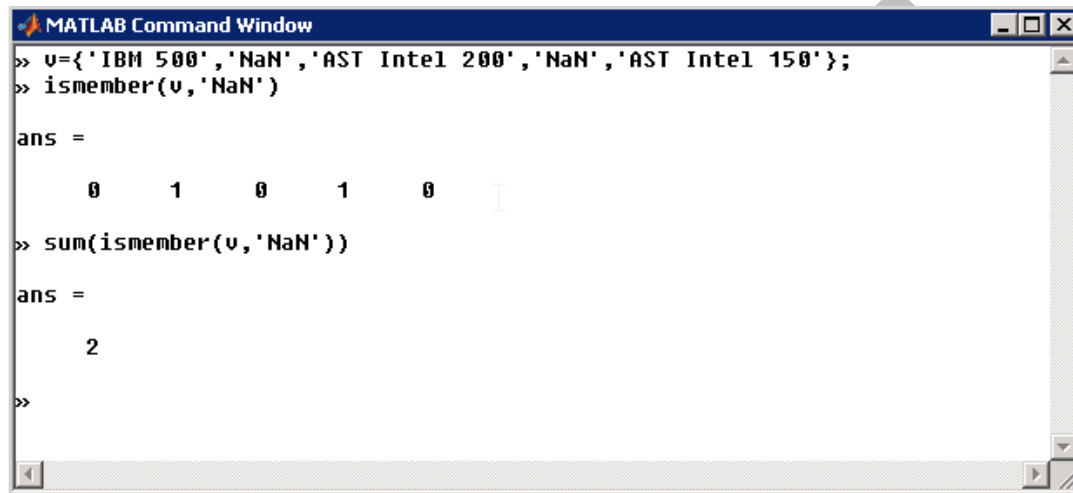
Important practical cases

This like a FAQ (frequently asked question) for which MATLAB doesn't seem to have built-in functions.

Counting/Removing NaN in cell arrays

The first typical question is to remove NaN values in a cell array.

How many NaN do we have? Use for this the **ismember()** and **sum()** functions together:



```
MATLAB Command Window
>> v={'IBM 500','NaN','AST Intel 200','NaN','AST Intel 150'};
>> ismember(v,'NaN')

ans =

     0     1     0     1     0

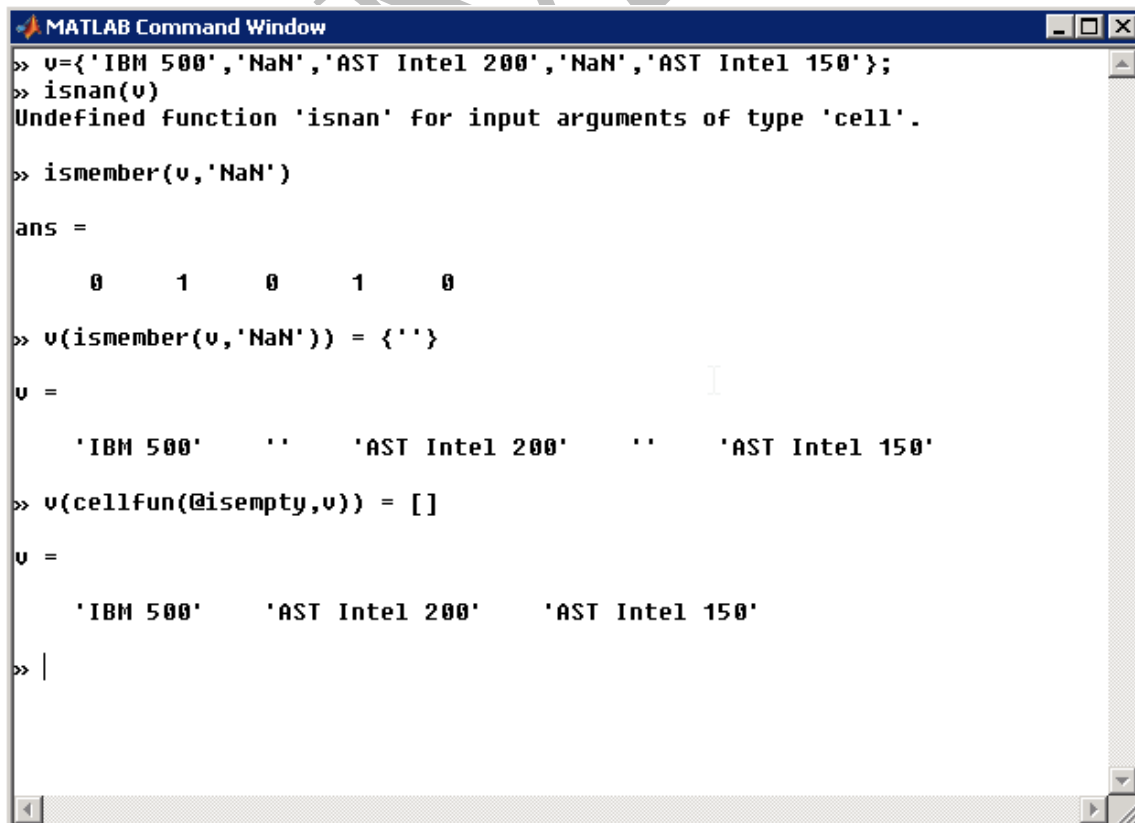
>> sum(ismember(v,'NaN'))

ans =

     2

>>
```

And now comes the tricky part... how to clean the vector using **cellfun()** function:



```
MATLAB Command Window
>> v={'IBM 500','NaN','AST Intel 200','NaN','AST Intel 150'};
>> isnan(v)
Undefined function 'isnan' for input arguments of type 'cell'.
>> ismember(v,'NaN')

ans =

     0     1     0     1     0

>> v(ismember(v,'NaN')) = {''}
v =

    'IBM 500'      ''    'AST Intel 200'      ''    'AST Intel 150'

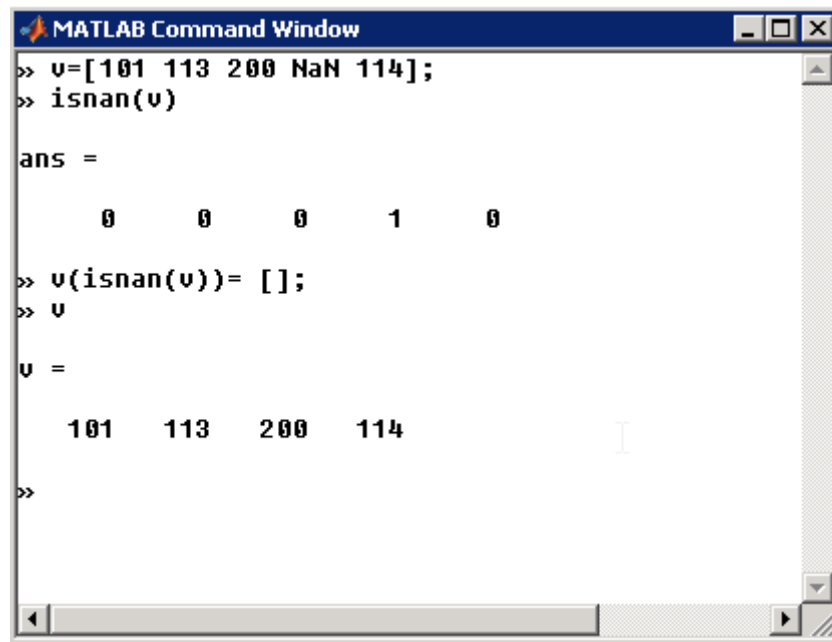
>> v(cellfun(@isempty,v)) = []
v =

    'IBM 500'    'AST Intel 200'    'AST Intel 150'

>> |
```

Counting/Removing NaN in (numeric) vectors

There are multiple ways to remove NaN in numeric vectors. There are all easier than with cell array. We will just use **isnan()** function:



```
MATLAB Command Window
>> v=[101 113 200 NaN 114];
>> isnan(v)

ans =

     0     0     0     1     0

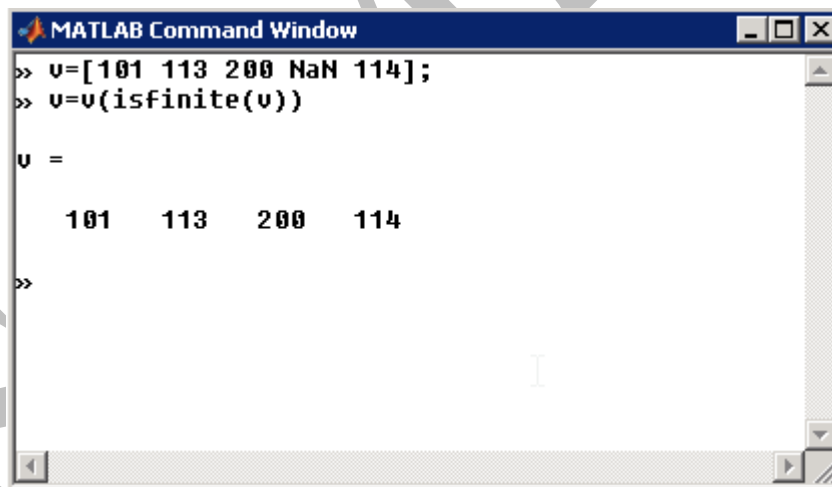
>> v(isnan(v))= [];
>> v

v =

    101    113    200    114

>>
```

Or **isfinite()**:



```
MATLAB Command Window
>> v=[101 113 200 NaN 114];
>> v=v(isfinite(v))

v =

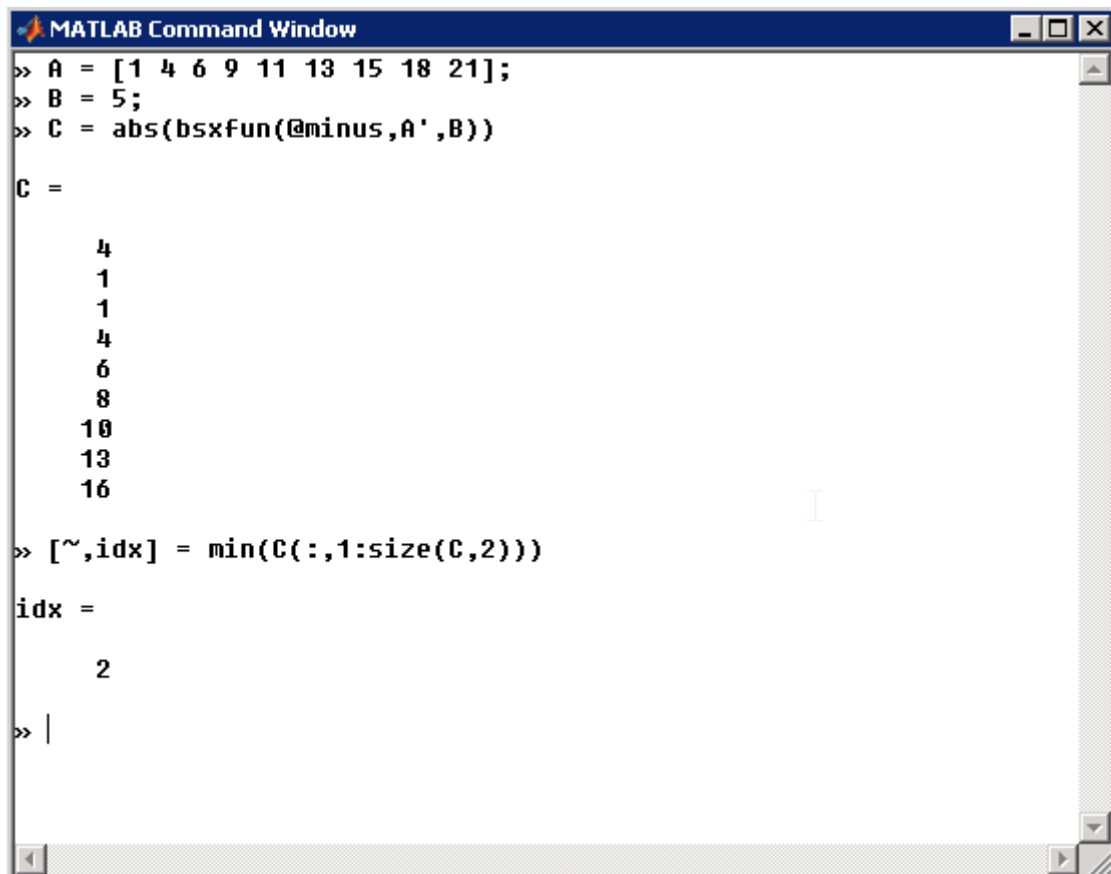
    101    113    200    114

>>
```

Finding the closest number in an vector

Here we will see how to find the position of the closes value in a numeric vector. This is very useful in finance. For this purpose we will use a trick well know in a lot of programming languages.

The presence of the function **bsxfun()** is just to make the thing faster (see page 589 on fast scripting tricks):



```
MATLAB Command Window
>> A = [1 4 6 9 11 13 15 18 21];
>> B = 5;
>> C = abs(bsxfun(@minus,A',B))

C =

     4
     1
     1
     4
     6
     8
    10
    13
    16

>> [~,idx] = min(C(:,1:size(C,2)))
idx =

     2

>> |
```

The screenshot shows a MATLAB Command Window with the following code and output:

```
>> A = [1 4 6 9 11 13 15 18 21];
>> B = 5;
>> C = abs(bsxfun(@minus,A',B))

C =

     4
     1
     1
     4
     6
     8
    10
    13
    16

>> [~,idx] = min(C(:,1:size(C,2)))
idx =

     2

>> |
```

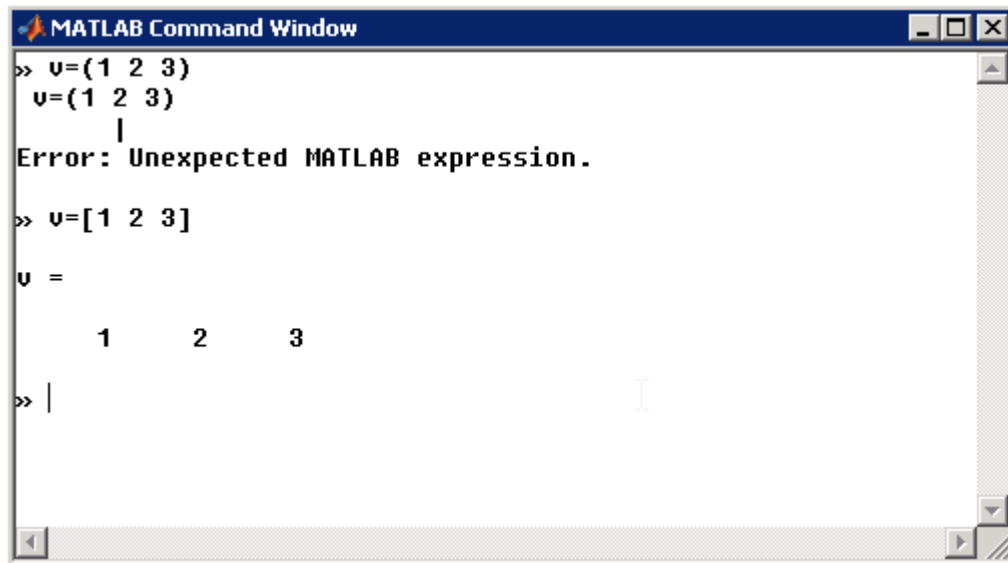
The output shows the absolute differences between the elements of vector A and the scalar B. The minimum value is 1, which occurs at indices 2 and 3. The variable idx is set to 2, indicating the first occurrence of the minimum value.

Arithmetic operations and manipulations of vectors

As we said before numerical vectors are one of the most important objects that any user has to master.

Let's see here the most common prior knowledge you must have before doing more complicated stuff (algorithms).

First you must remember that a vector will not be defined with parenthesis but with brackets:



```
MATLAB Command Window
>> v=(1 2 3)
v=(1 2 3)
|
Error: Unexpected MATLAB expression.

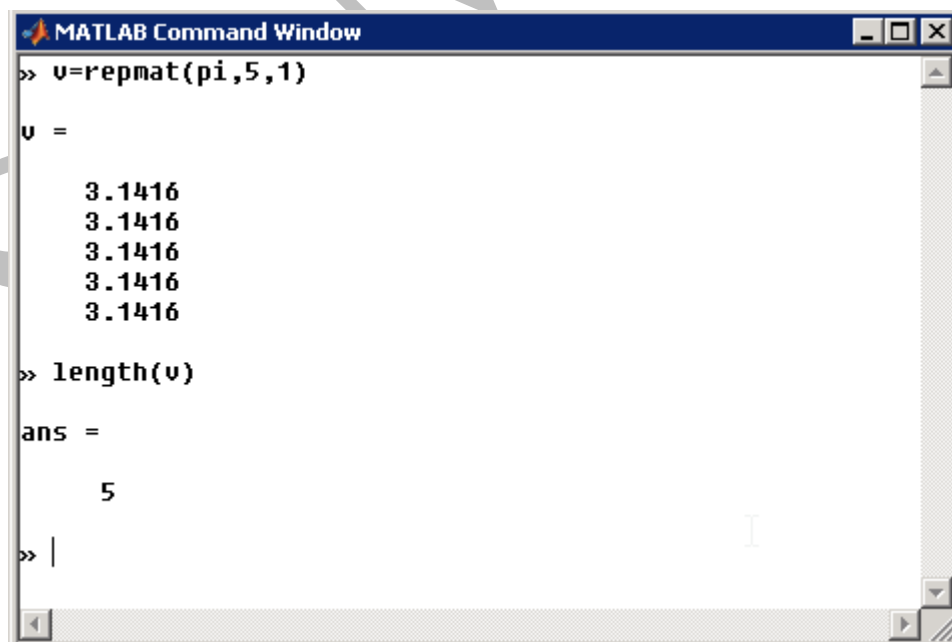
>> v=[1 2 3]

v =

     1     2     3

>> |
```

Another very useful function is to repeat a given value some times in a vector using **repmat()** and also to get a vector **length()**:



```
MATLAB Command Window
>> v=repmat(pi,5,1)

v =

    3.1416
    3.1416
    3.1416
    3.1416
    3.1416

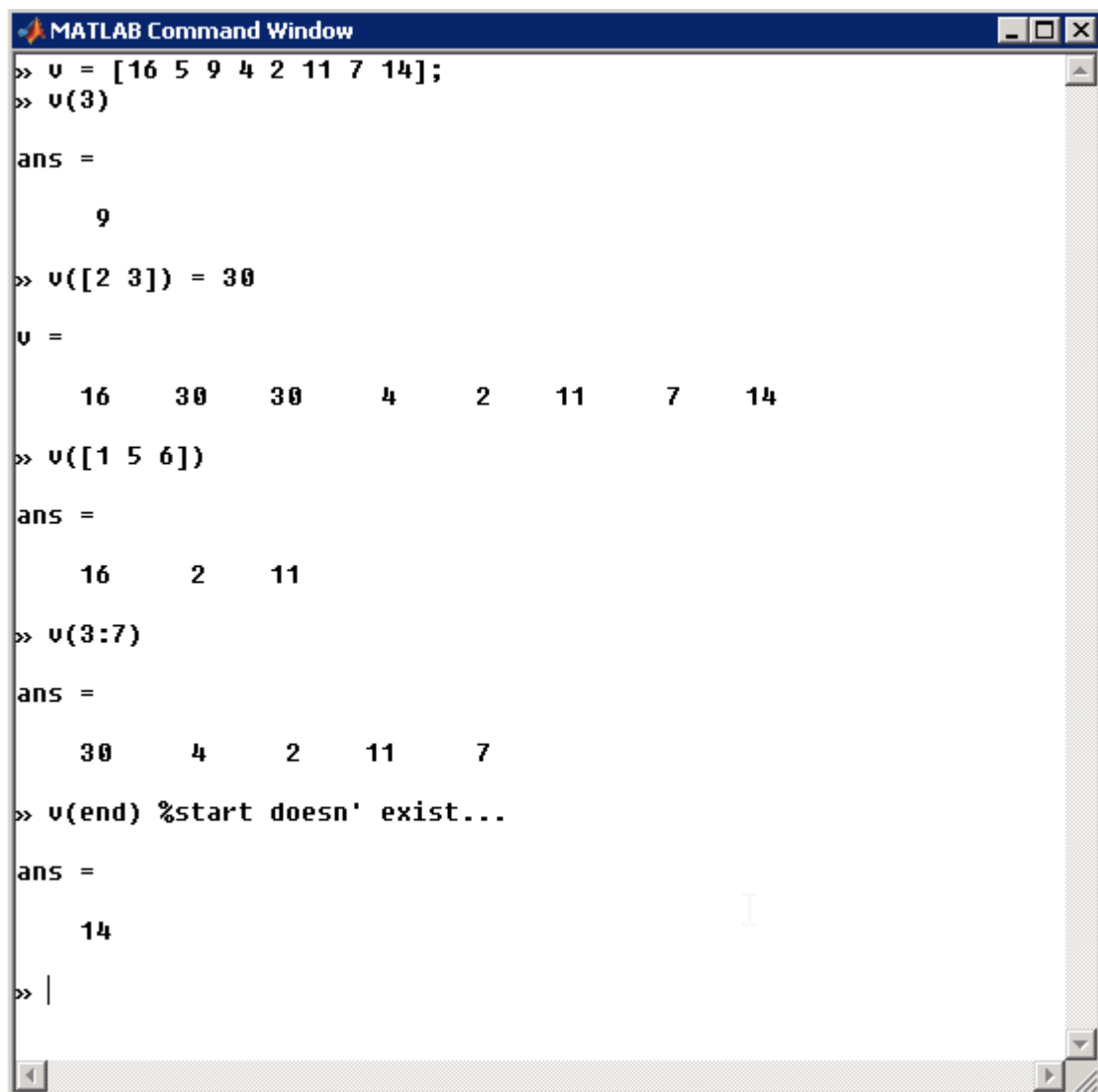
>> length(v)

ans =

     5

>> |
```

Extracting values from a vector or subsets is also something very important:

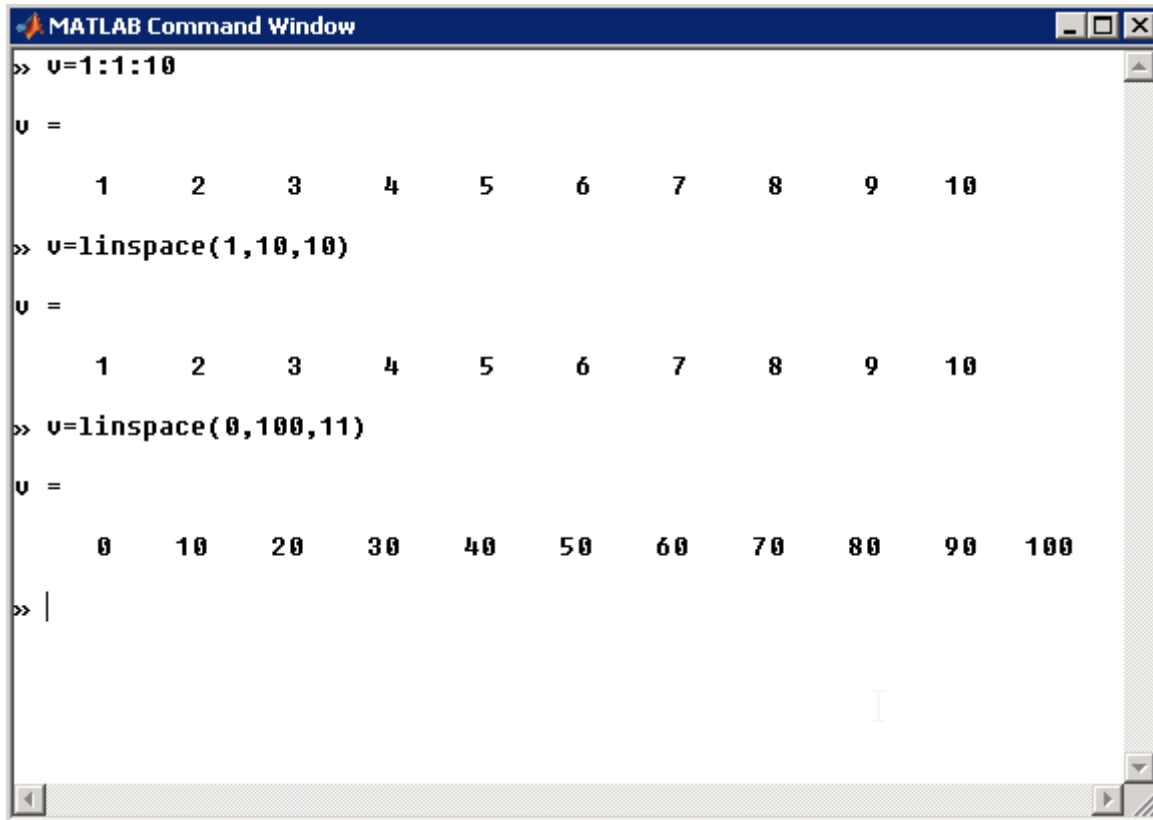


A screenshot of the MATLAB Command Window showing a series of commands and their outputs. The window has a blue title bar with the MATLAB logo and the text 'MATLAB Command Window'. The command history is as follows:

```
>> v = [16 5 9 4 2 11 7 14];  
>> v(3)  
  
ans =  
  
    9  
  
>> v([2 3]) = 30  
  
v =  
  
    16    30    30     4     2    11     7    14  
  
>> v([1 5 6])  
  
ans =  
  
    16     2    11  
  
>> v(3:7)  
  
ans =  
  
    30     4     2    11     7  
  
>> v(end) %start doesn't exist...  
  
ans =  
  
    14  
  
>> |
```

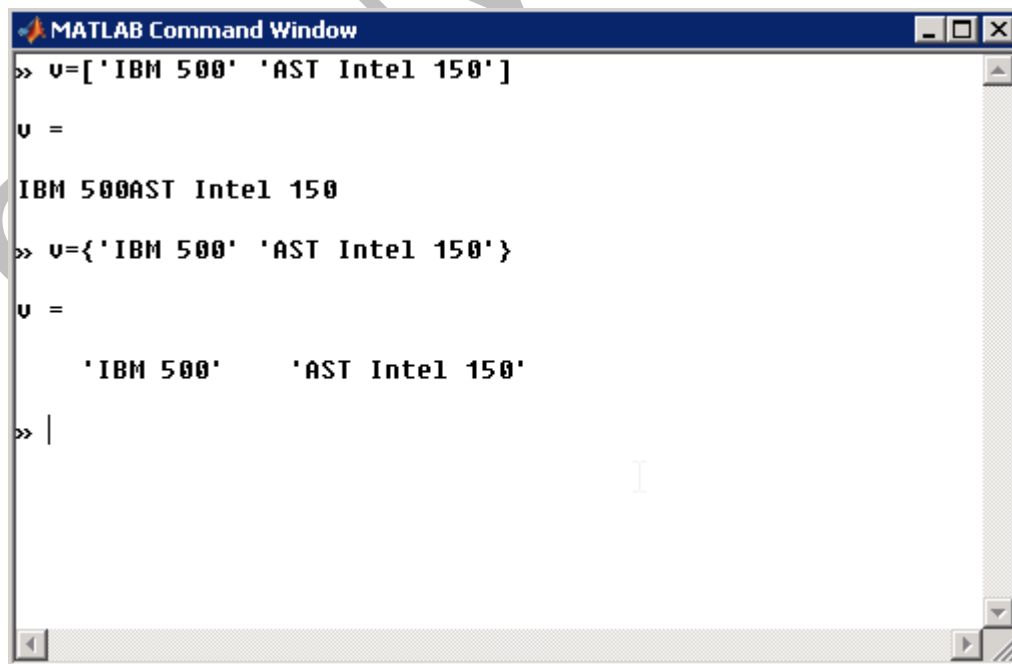
The output shows the vector `v` being modified in place. The first command creates `v` with values [16, 5, 9, 4, 2, 11, 7, 14]. The second command extracts the value at index 3, which is 9. The third command replaces the values at indices 2 and 3 with 30. The fourth command extracts the values at indices 1, 5, and 6, which are 16, 2, and 11. The fifth command extracts the values from index 3 to 7, which are 30, 4, 2, 11, and 7. The sixth command extracts the value at the end of the vector, which is 14.

Something also very very important is to be able to create vector ranges using **linspace()** function or not:



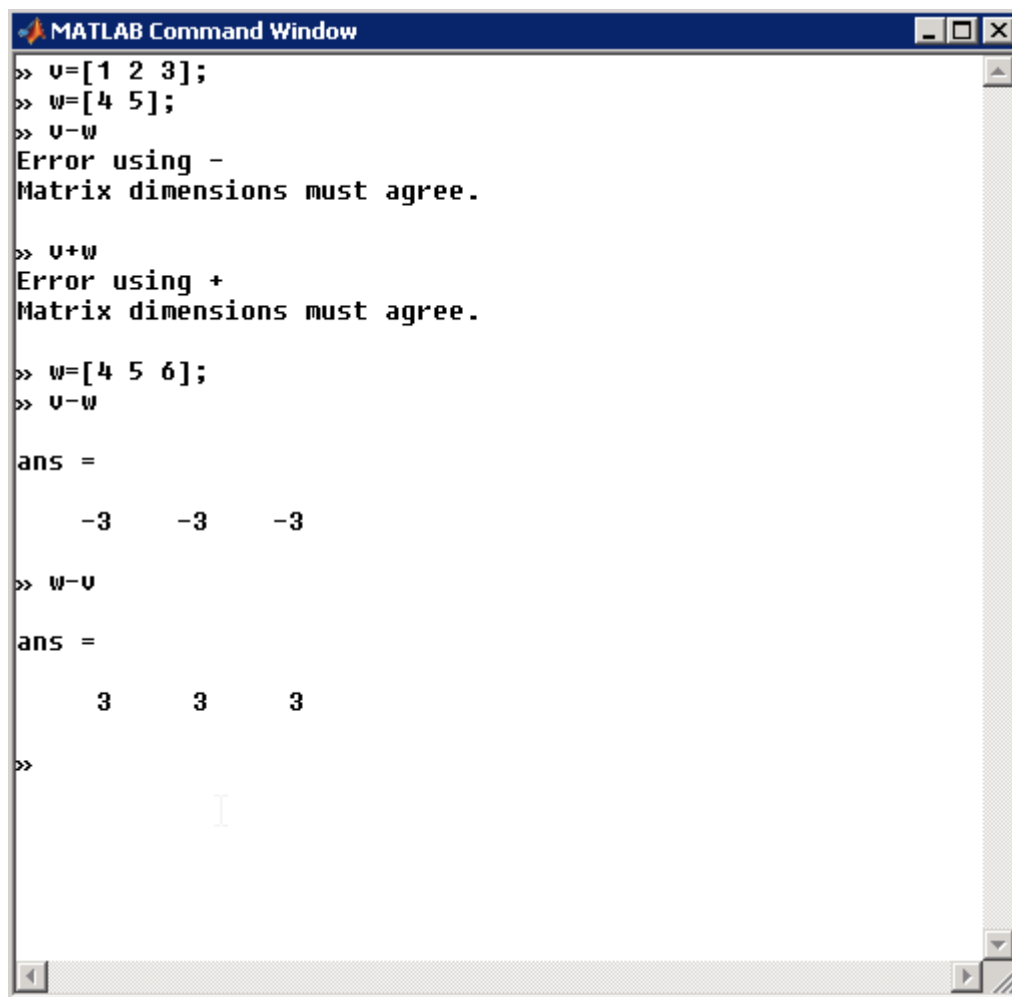
```
MATLAB Command Window
>> v=1:1:10
v =
     1     2     3     4     5     6     7     8     9    10
>> v=linspace(1,10,10)
v =
     1     2     3     4     5     6     7     8     9    10
>> v=linspace(0,100,11)
v =
     0    10    20    30    40    50    60    70    80    90   100
>> |
```

Remember that you can't put string in vectors but only in cell array that's why the definition of the first use `[ ]` and the second `{ }`:



```
MATLAB Command Window
>> v=['IBM 500' 'AST Intel 150']
v =
IBM 500AST Intel 150
>> v={'IBM 500' 'AST Intel 150'}
v =
    'IBM 500'    'AST Intel 150'
>> |
```


You always have to respect vectors dimensions rules that you have seen during your study in vector analysis and linear algebra even in MATLAB ;-P :



```
MATLAB Command Window
>> u=[1 2 3];
>> w=[4 5];
>> u-w
Error using -
Matrix dimensions must agree.

>> u+w
Error using +
Matrix dimensions must agree.

>> w=[4 5 6];
>> u-w

ans =

    -3    -3    -3

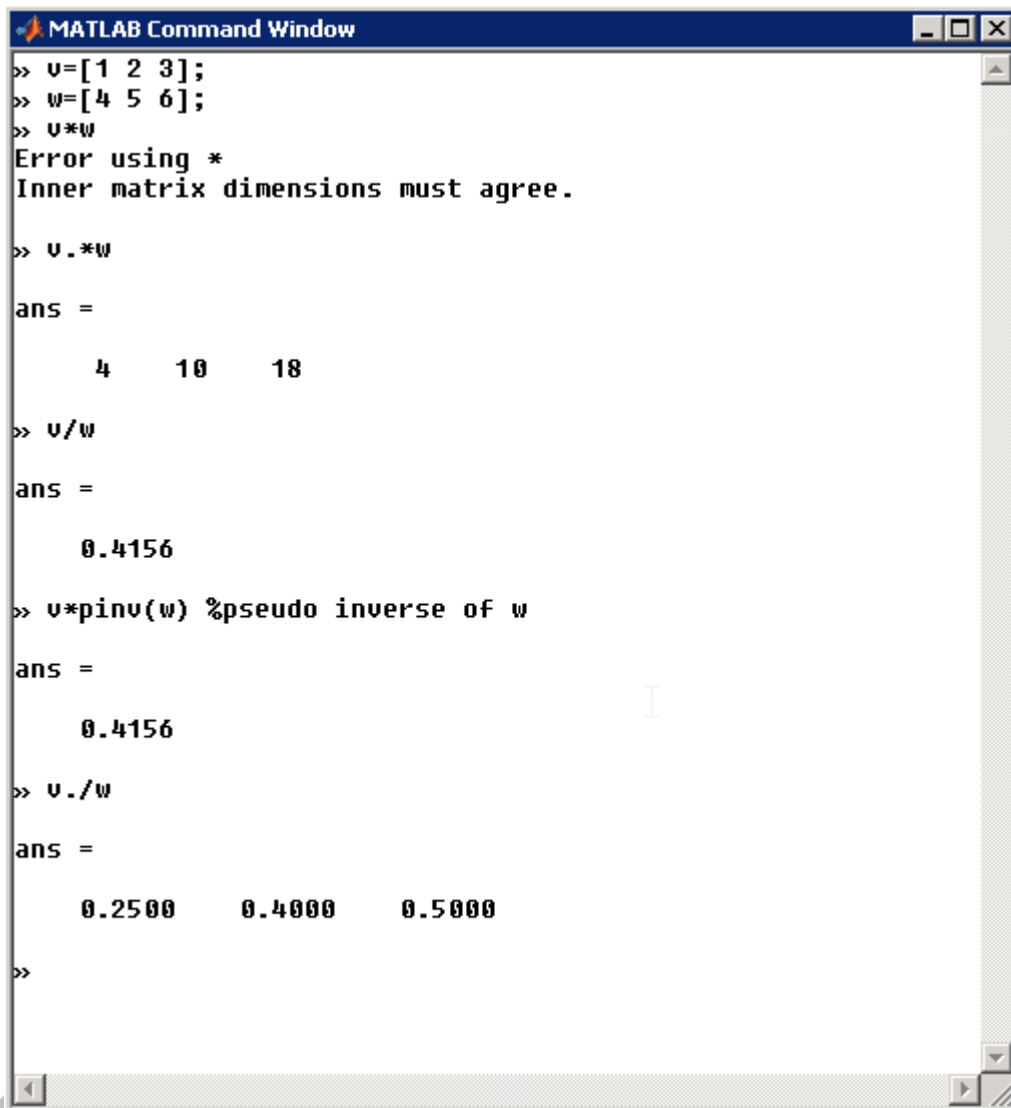
>> w-u

ans =

     3     3     3

>>
```

There is something very important to note: in MATLAB if you precede the multiplication symbol, division symbol or power symbol by a point (**.***, **./**, **.^X**) it means that you are doing a component by component operation:

A screenshot of the MATLAB Command Window. The window title is "MATLAB Command Window". The command history shows: 1. `v=[1 2 3];` 2. `w=[4 5 6];` 3. `v\*w` which results in an error: "Error using * Inner matrix dimensions must agree." 4. `v.\*w` which results in `ans = 4 10 18`. 5. `v/w` which results in `ans = 0.4156`. 6. `v\*pinv(w) %pseudo inverse of w` which results in `ans = 0.4156`. 7. `v./w` which results in `ans = 0.2500 0.4000 0.5000`. 8. A final prompt `>>` is shown at the bottom.

```
>> v=[1 2 3];
>> w=[4 5 6];
>> v*w
Error using *
Inner matrix dimensions must agree.

>> v.*w

ans =

     4     10     18

>> v/w

ans =

     0.4156

>> v*pinv(w) %pseudo inverse of w

ans =

     0.4156

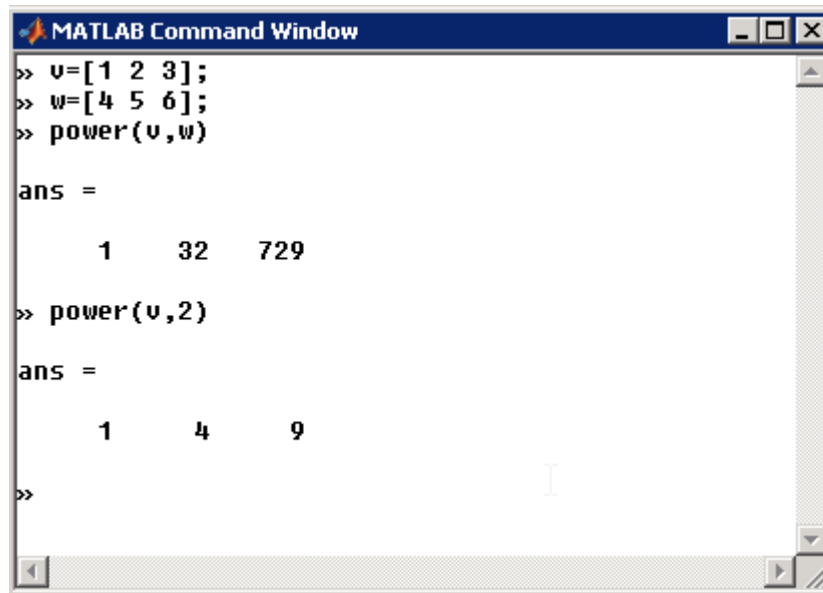
>> v./w

ans =

     0.2500     0.4000     0.5000

>>
```

The **power()** function apply to a vector with a vector as exponent is equivalent to a component by component **.^X** power:

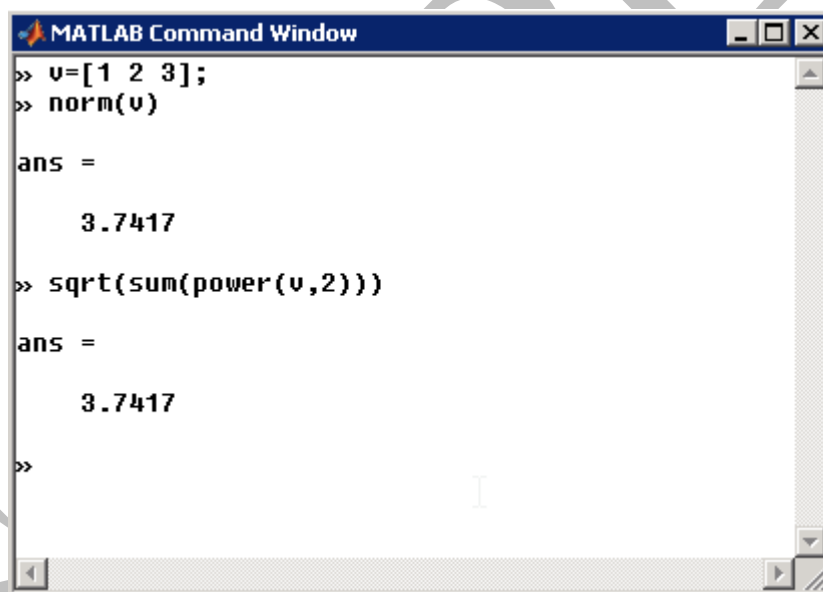


```
MATLAB Command Window
>> v=[1 2 3];
>> w=[4 5 6];
>> power(v,w)

ans =
     1    32   729
>> power(v,2)

ans =
     1     4     9
>>
```

We also have the function **norm()** to calculate the euclidian norm of a vector (we make also quickly the check...):

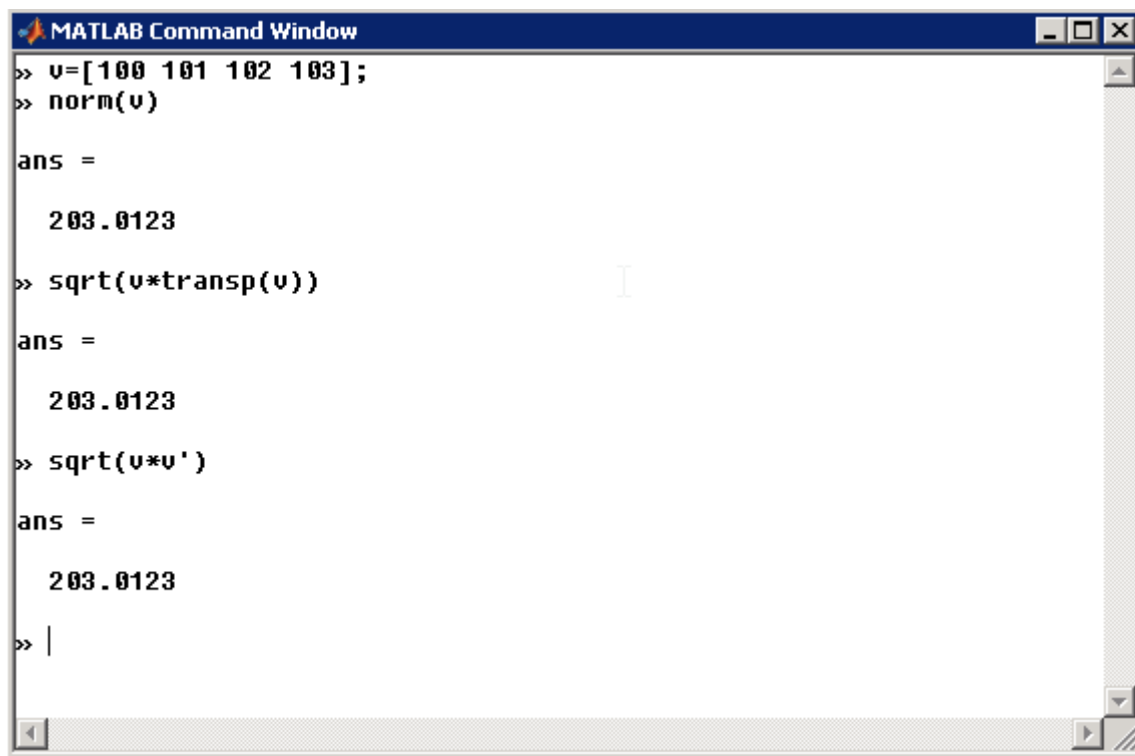


```
MATLAB Command Window
>> v=[1 2 3];
>> norm(v)

ans =
    3.7417
>> sqrt(sum(power(v,2)))

ans =
    3.7417
>>
```

Or some other possible syntaxes for the norm that you can sometimes see in scripts:



```
MATLAB Command Window
>> v=[100 101 102 103];
>> norm(v)

ans =

    203.0123

>> sqrt(v*transp(v))

ans =

    203.0123

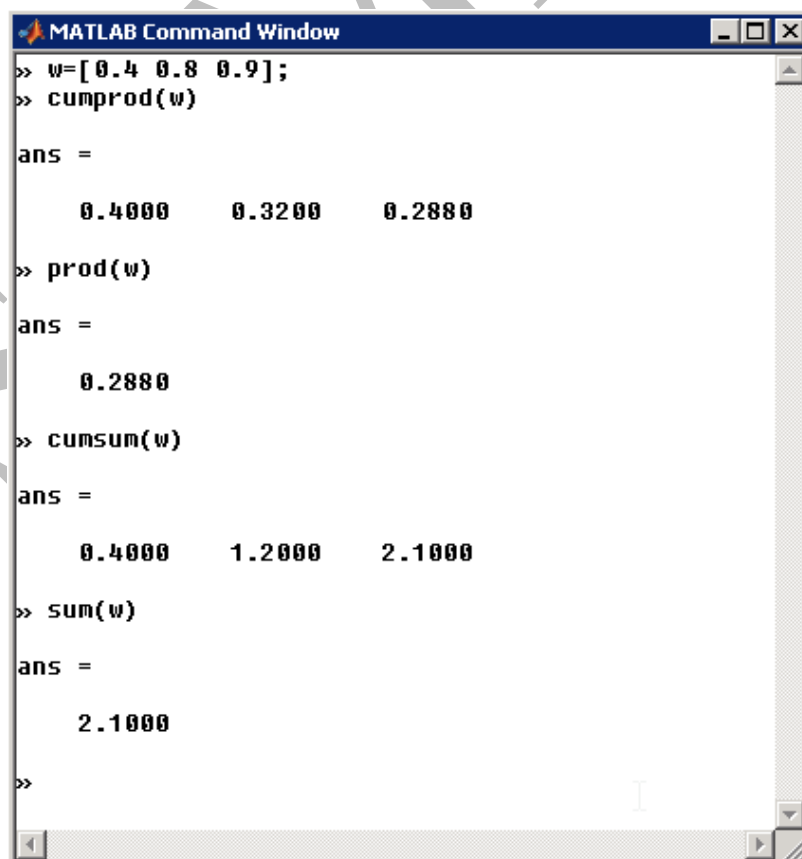
>> sqrt(v*v')

ans =

    203.0123

>> |
```

If you work with probabilities vector the **cumprod()** function will be very useful for you, and if you work in the field of finance you will appreciate the **cumsum()** function:



```
MATLAB Command Window
>> w=[0.4 0.8 0.9];
>> cumprod(w)

ans =

    0.4000    0.3200    0.2880

>> prod(w)

ans =

    0.2880

>> cumsum(w)

ans =

    0.4000    1.2000    2.1000

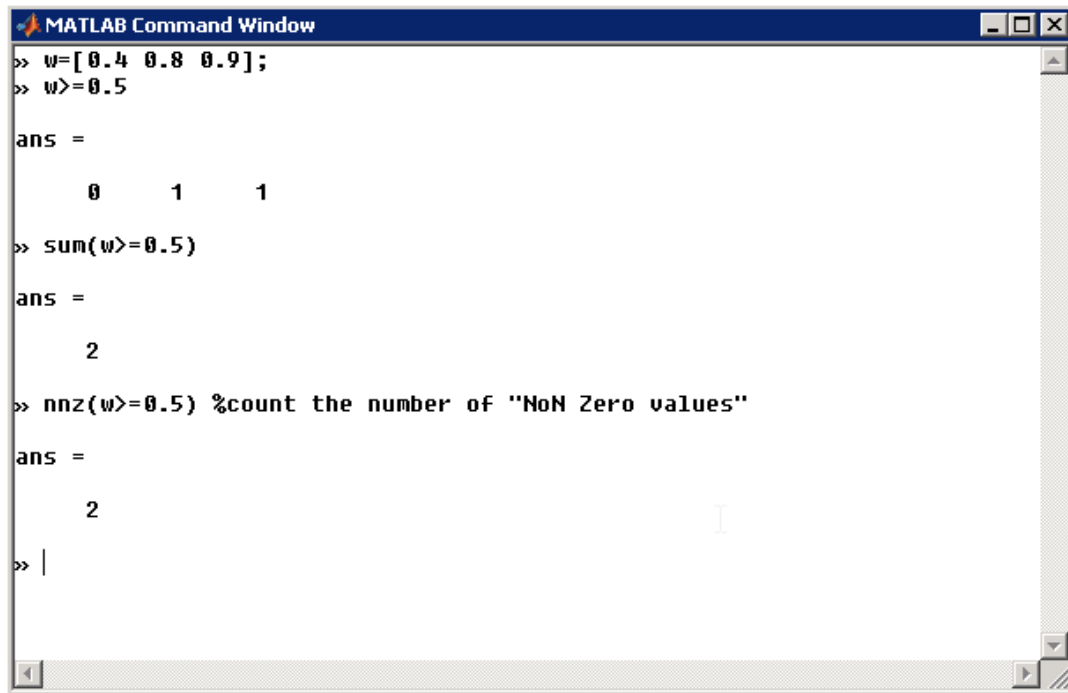
>> sum(w)

ans =

    2.1000

>>
```

We can also apply logical operations on vectors and especially for example non zero values with the function **nnz()**:



```
MATLAB Command Window
>> w=[0.4 0.8 0.9];
>> w>=0.5

ans =

     0     1     1

>> sum(w>=0.5)

ans =

     2

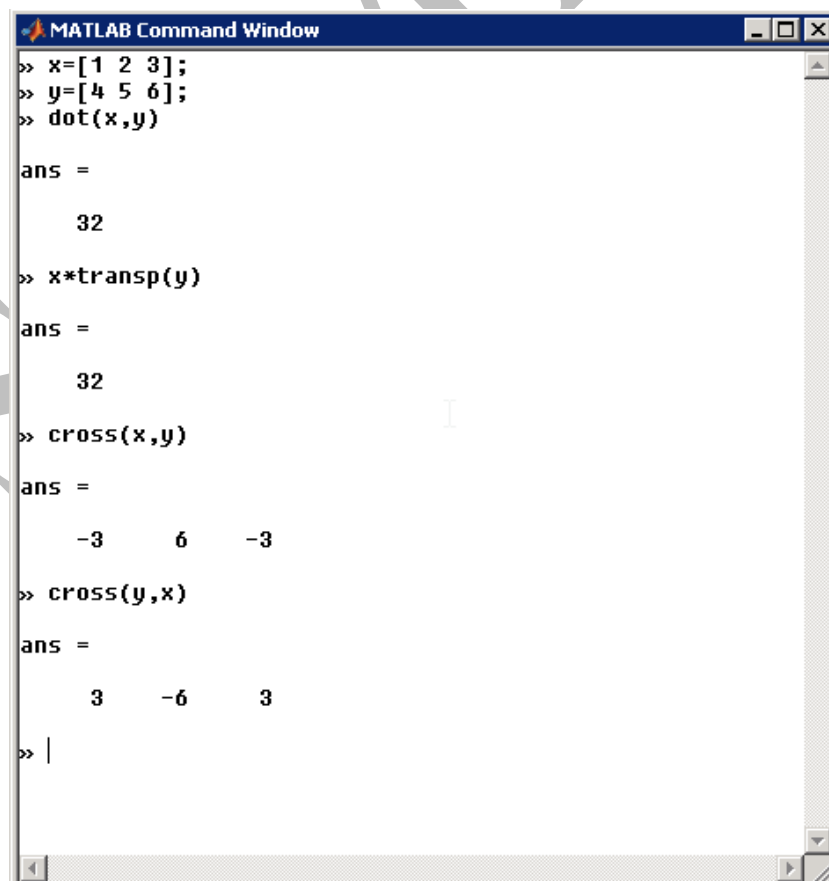
>> nnz(w>=0.5) %count the number of "NoN Zero values"

ans =

     2

>> |
```

We can also have a look at the dot product and cross product with the functions **dot()** and **cross()** that are very useful in physics:



```
MATLAB Command Window
>> x=[1 2 3];
>> y=[4 5 6];
>> dot(x,y)

ans =

    32

>> x*transp(y)

ans =

    32

>> cross(x,y)

ans =

    -3     6    -3

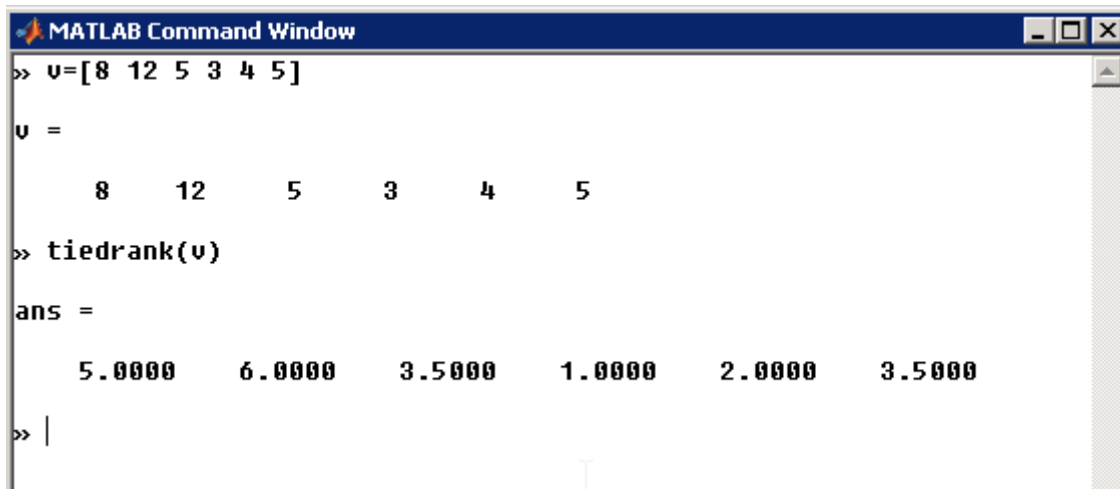
>> cross(y,x)

ans =

     3    -6     3

>> |
```

You certainly remember that the function **unique()** returns also the rank, but there is also a function named **tiedrank()** that returns another type of ranking method keeping all original values:



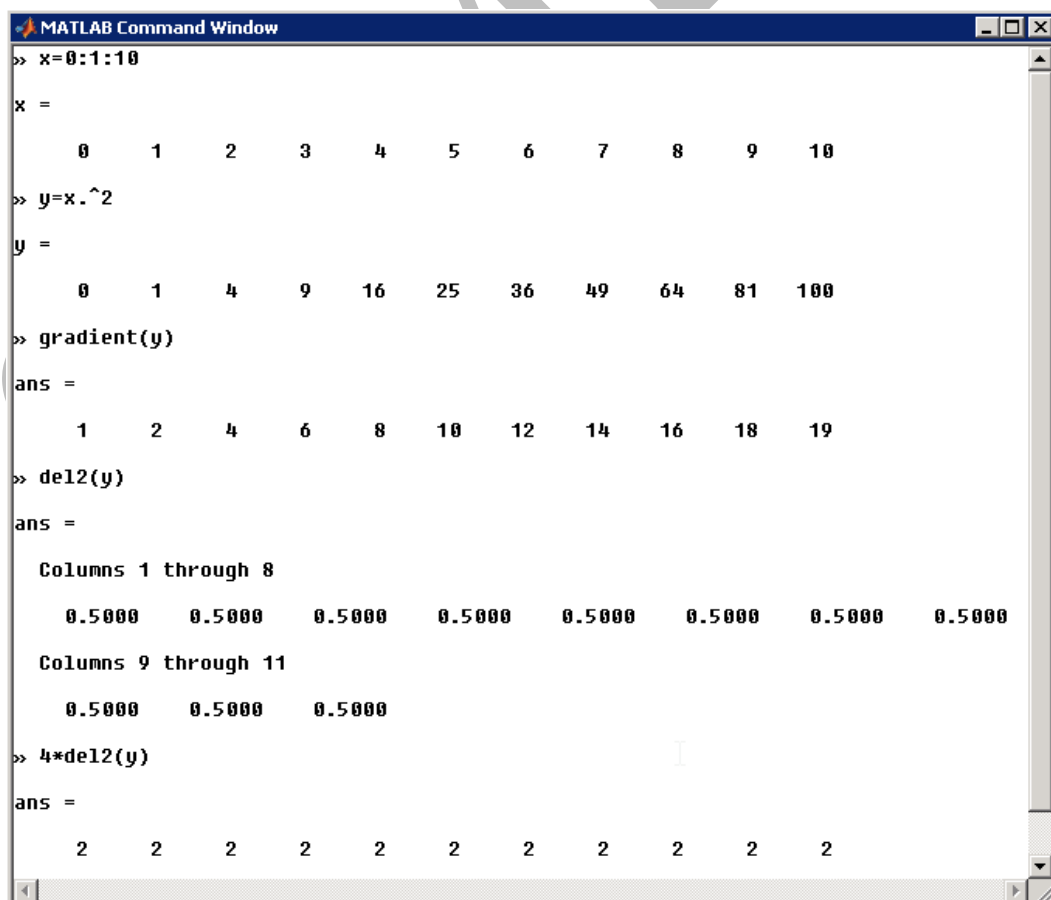
```

MATLAB Command Window
>> v=[8 12 5 3 4 5]
v =
     8     12     5     3     4     5
>> tiedrank(v)
ans =
     5.0000     6.0000     3.5000     1.0000     2.0000     3.5000
>> |

```

Now take a look the **gradient()** function by checking that it's effectively giving back the derivate and the laplacien with the function **del2()** giving back the second order derivate with a simple naive example.

As you can see below **there is a trap** with MATLAB when you use **del2()**. You have to multiply the result by 4 to get the standard laplacien:



```

MATLAB Command Window
>> x=0:1:10
x =
     0     1     2     3     4     5     6     7     8     9    10
>> y=x.^2
y =
     0     1     4     9    16    25    36    49    64    81   100
>> gradient(y)
ans =
     1     2     4     6     8    10    12    14    16    18    19
>> del2(y)
ans =
Columns 1 through 8
     0.5000     0.5000     0.5000     0.5000     0.5000     0.5000     0.5000     0.5000
Columns 9 through 11
     0.5000     0.5000     0.5000
>> 4*del2(y)
ans =
     2     2     2     2     2     2     2     2     2     2     2

```

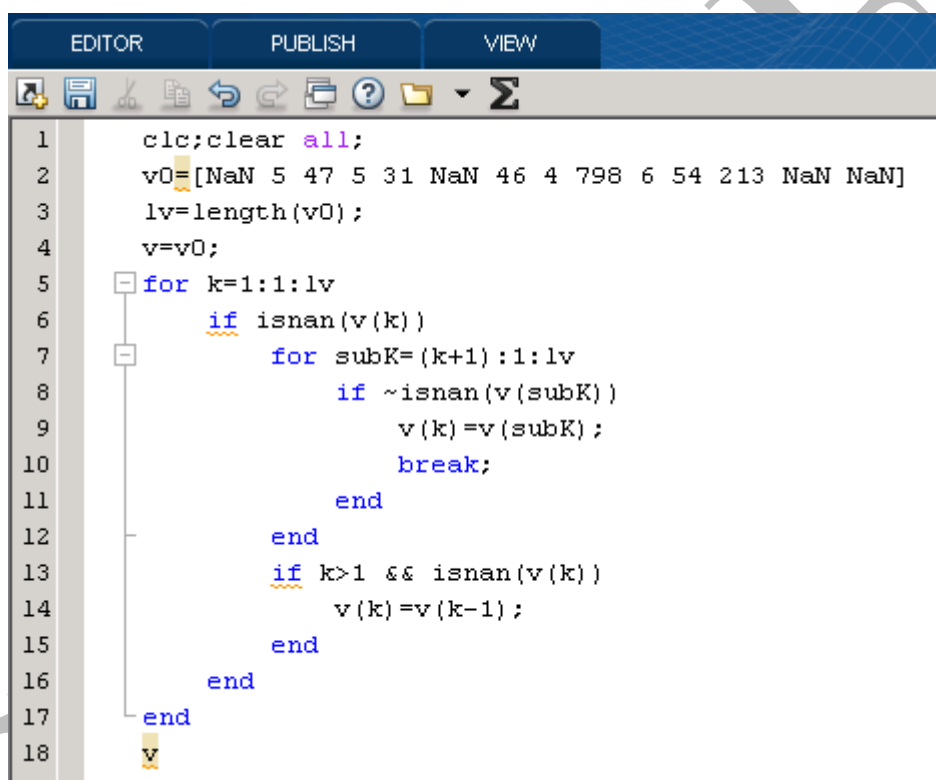
Important practical cases

This like a FAQ (frequently asked question) for which MATLAB doesn't seem to have built-in functions.

Clean a vector

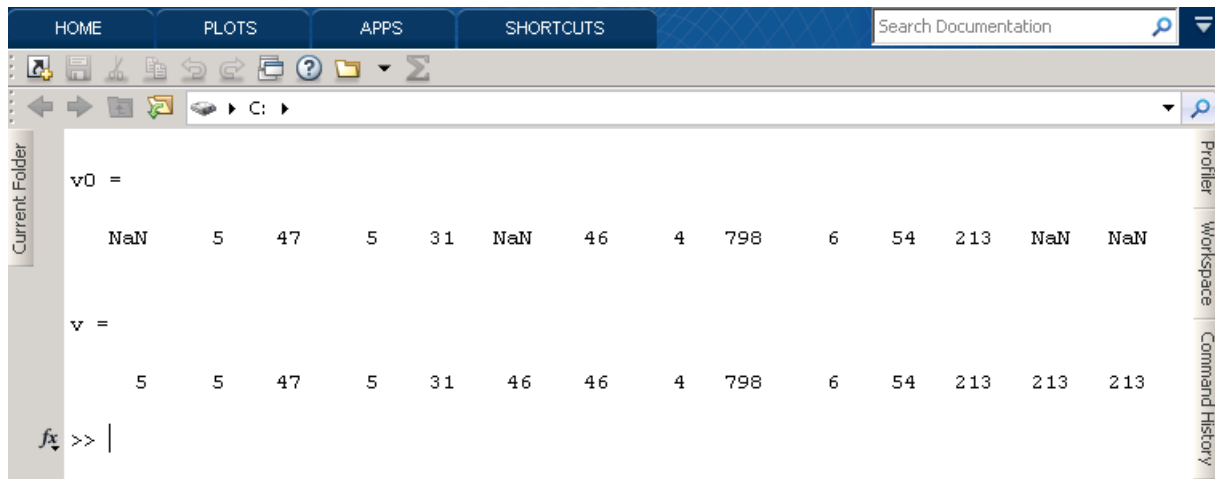
Consider we have a return vector retrieved from a financial feed. Some values can be missing. In case of missing value at the beginning, the rule is to take the next available one... otherwise we take the previous day value.

Consider this practical case as a challenge! The fastest code written today by one of my student is the following:



```
1      clc;clear all;
2      v0=[NaN 5 47 5 31 NaN 46 4 798 6 54 213 NaN NaN]
3      lv=length(v0);
4      v=v0;
5      for k=1:1:lv
6          if isnan(v(k))
7              for subK=(k+1):1:lv
8                  if ~isnan(v(subK))
9                      v(k)=v(subK);
10                     break;
11                 end
12             end
13             if k>1 && isnan(v(k))
14                 v(k)=v(k-1);
15             end
16         end
17     end
18     v
```

This gives:



Build an orthonormal vector base

So, given vector

$$v = \begin{pmatrix} a \\ b \\ c \end{pmatrix}$$

in $\mathbb{R}^3$, it follows that any point (x, y, z) that satisfies:

$$ax + by + cz = 0$$

(i.e. a point that lives on plane whose normal vector is v) will define components of a vector that u is orthogonal to v .

One such vector is:

$$u = \begin{pmatrix} 0 \\ -c \\ b \end{pmatrix}, u = \begin{pmatrix} -c \\ 0 \\ a \end{pmatrix}$$

you can choose whatever you prefer, but it cannot be:

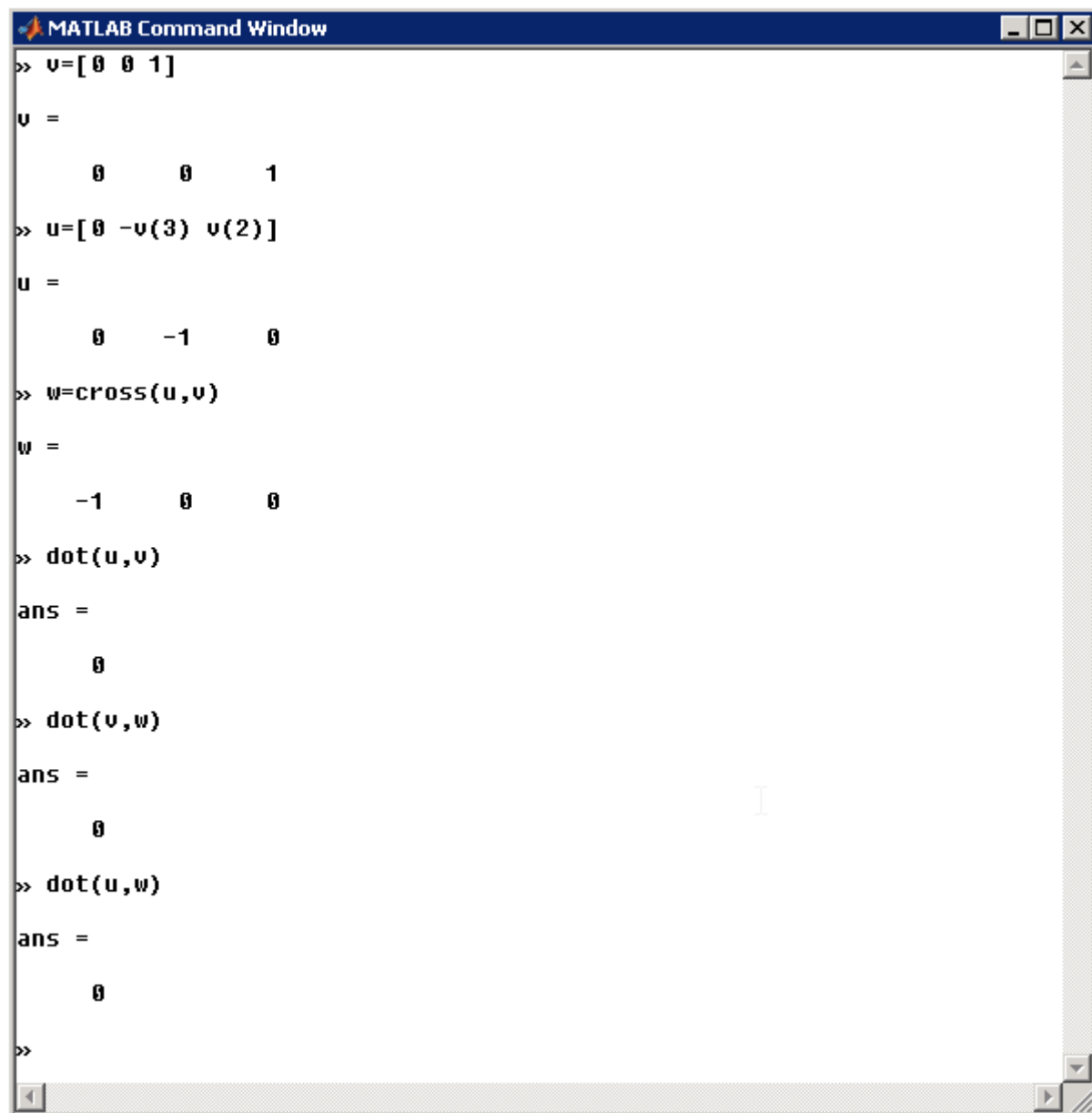
$$u = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Now, given u and v , compute the third vector that is orthogonal to both u and v with

$$w = u \times v$$

To make an orthonormal basis just divide each vector by its norm!

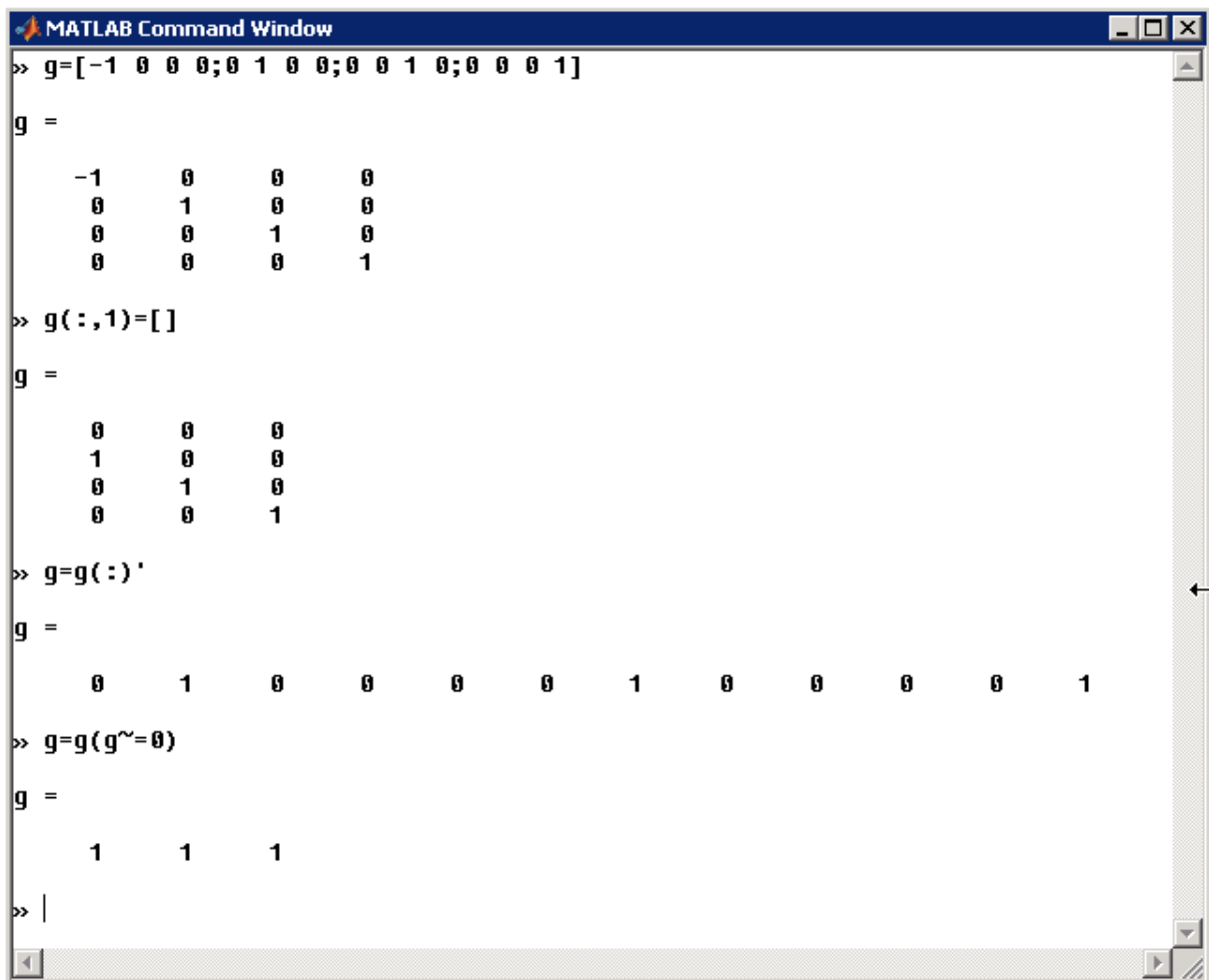
Demo:

A screenshot of the MATLAB Command Window. The window has a blue title bar with the MATLAB logo and the text "MATLAB Command Window". The command history shows the following:
1. Command: `v=[0 0 1]`
Output: `v =` followed by a row vector `0 0 1`.
2. Command: `u=[0 -v(3) v(2)]`
Output: `u =` followed by a row vector `0 -1 0`.
3. Command: `w=cross(u,v)`
Output: `w =` followed by a row vector `-1 0 0`.
4. Command: `dot(u,v)`
Output: `ans =` followed by the value `0`.
5. Command: `dot(v,w)`
Output: `ans =` followed by the value `0`.
6. Command: `dot(u,w)`
Output: `ans =` followed by the value `0`.
The window ends with a prompt `>>`. A large, light gray watermark "SCI" is visible across the bottom half of the image.

Arithmetic operations and manipulations of matrices

Master the manipulation of matrices is even more important than vectors. Let us see some manipulations with this type of objects.

We begin something simple with the Minkowsky matrix (see chapter of Relativity of my applied mathematics e-book) by removing some parts of it and getting the diagonal by excluding first the usage of the **diag()** function (see next page) :



```

MATLAB Command Window
>> g=[-1 0 0 0;0 1 0 0;0 0 1 0;0 0 0 1]
g =
    -1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1

>> g(:,1)=[]
g =
     0     0     0
     1     0     0
     0     1     0
     0     0     1

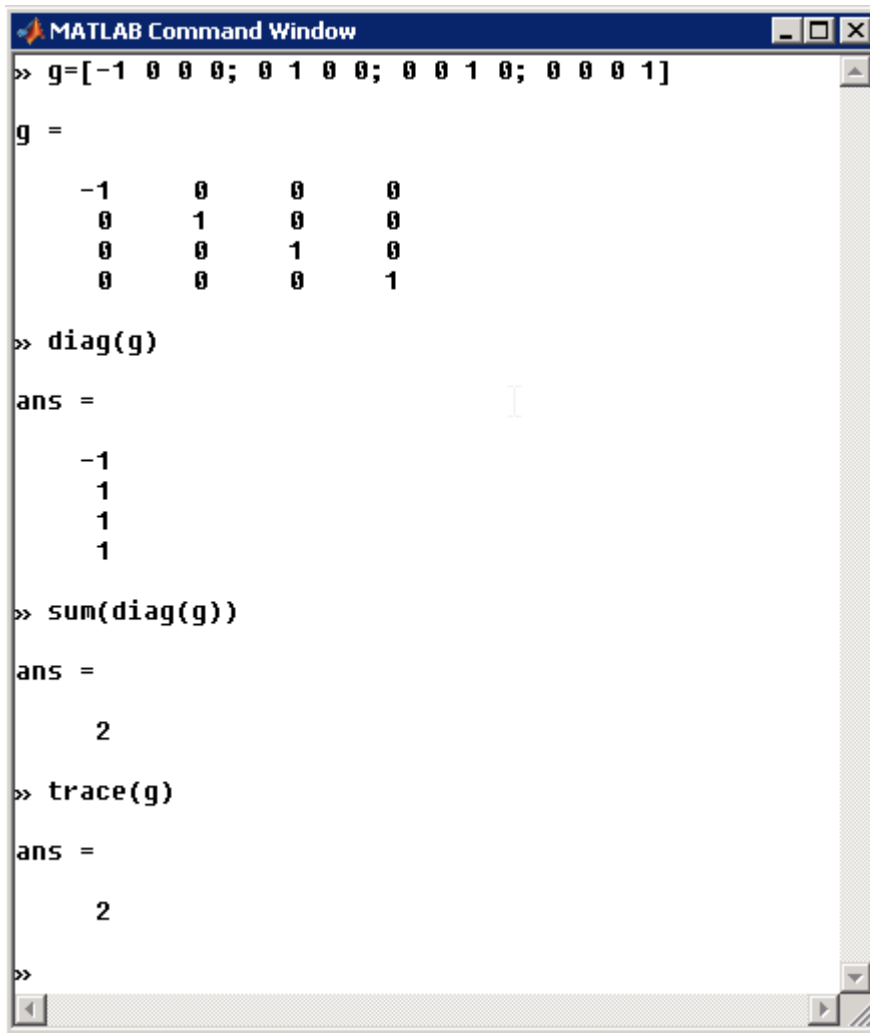
>> g=g(:) '
g =
     0     1     0     0     0     0     1     0     0     0     0     1

>> g=g(g~=0)
g =
     1     1     1

>> |

```

Now let us see that we can use the **diag()** function or **trace()** function that are very useful in advanced physics and also in statistics (when you build your own confusion matrix):

A screenshot of the MATLAB Command Window. The window title is "MATLAB Command Window". The command prompt shows the definition of a 4x4 matrix g: g=[-1 0 0 0; 0 1 0 0; 0 0 1 0; 0 0 0 1]. The output shows the matrix g as a 4x4 grid. Then, the command diag(g) is entered, and the output is a column vector containing the diagonal elements: -1, 1, 1, 1. Next, the command sum(diag(g)) is entered, and the output is 2. Finally, the command trace(g) is entered, and the output is also 2. The window has standard Windows-style window controls (minimize, maximize, close) in the top right corner and scroll bars on the right and bottom.

```
>> g=[-1 0 0 0; 0 1 0 0; 0 0 1 0; 0 0 0 1]

g =

    -1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1

>> diag(g)

ans =

    -1
     1
     1
     1

>> sum(diag(g))

ans =

     2

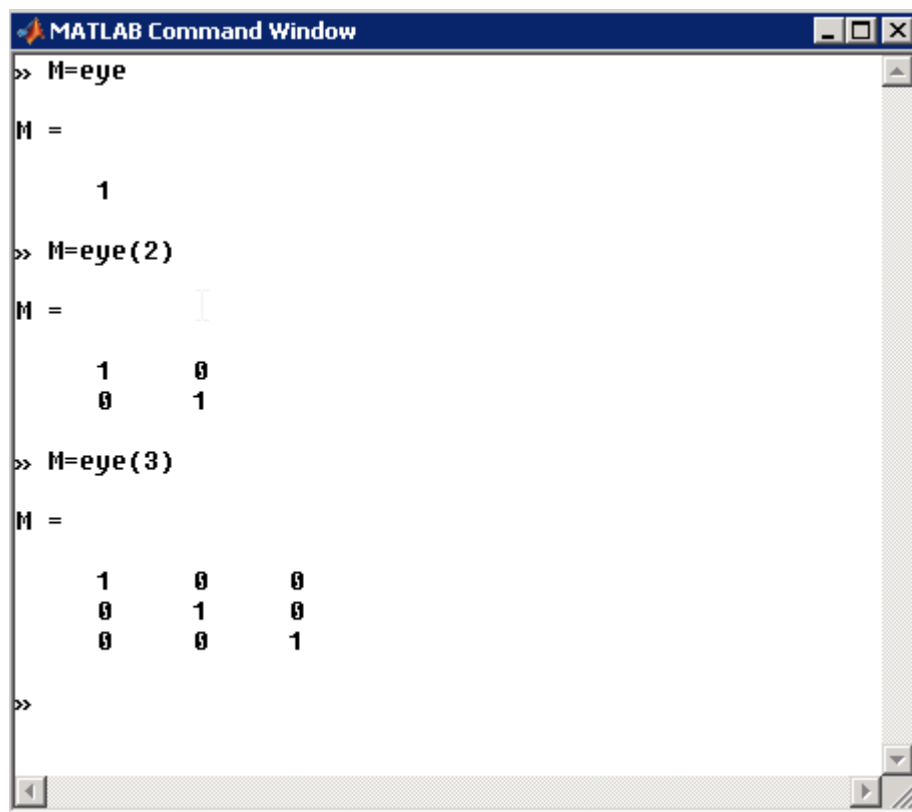
>> trace(g)

ans =

     2

>>
```

Sometimes it can be useful to generate a diagonal unit matrix. You can do that very easily with the **eye()** function:



```
>> M=eye

M =

     1

>> M=eye(2)

M =

     1     0
     0     1

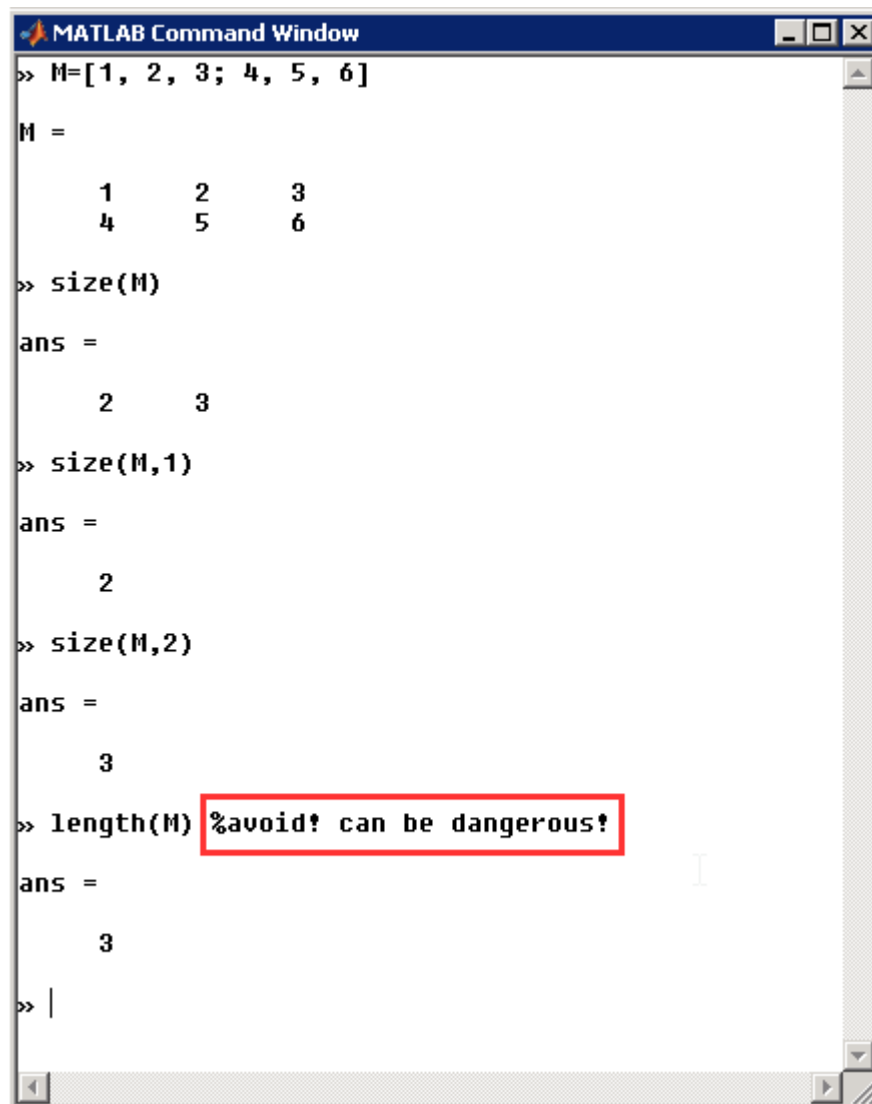
>> M=eye(3)

M =

     1     0     0
     0     1     0
     0     0     1

>>
```

Now have a look why it is **dangerous** to use the **length()** function instead of the **size()** function to get dimensions of a matrix:



A screenshot of the MATLAB Command Window. The window title is "MATLAB Command Window". The command prompt shows the following sequence of commands and outputs:

```
>> M=[1, 2, 3; 4, 5, 6]

M =

     1     2     3
     4     5     6

>> size(M)

ans =

     2     3

>> size(M,1)

ans =

     2

>> size(M,2)

ans =

     3

>> length(M) %avoid! can be dangerous!

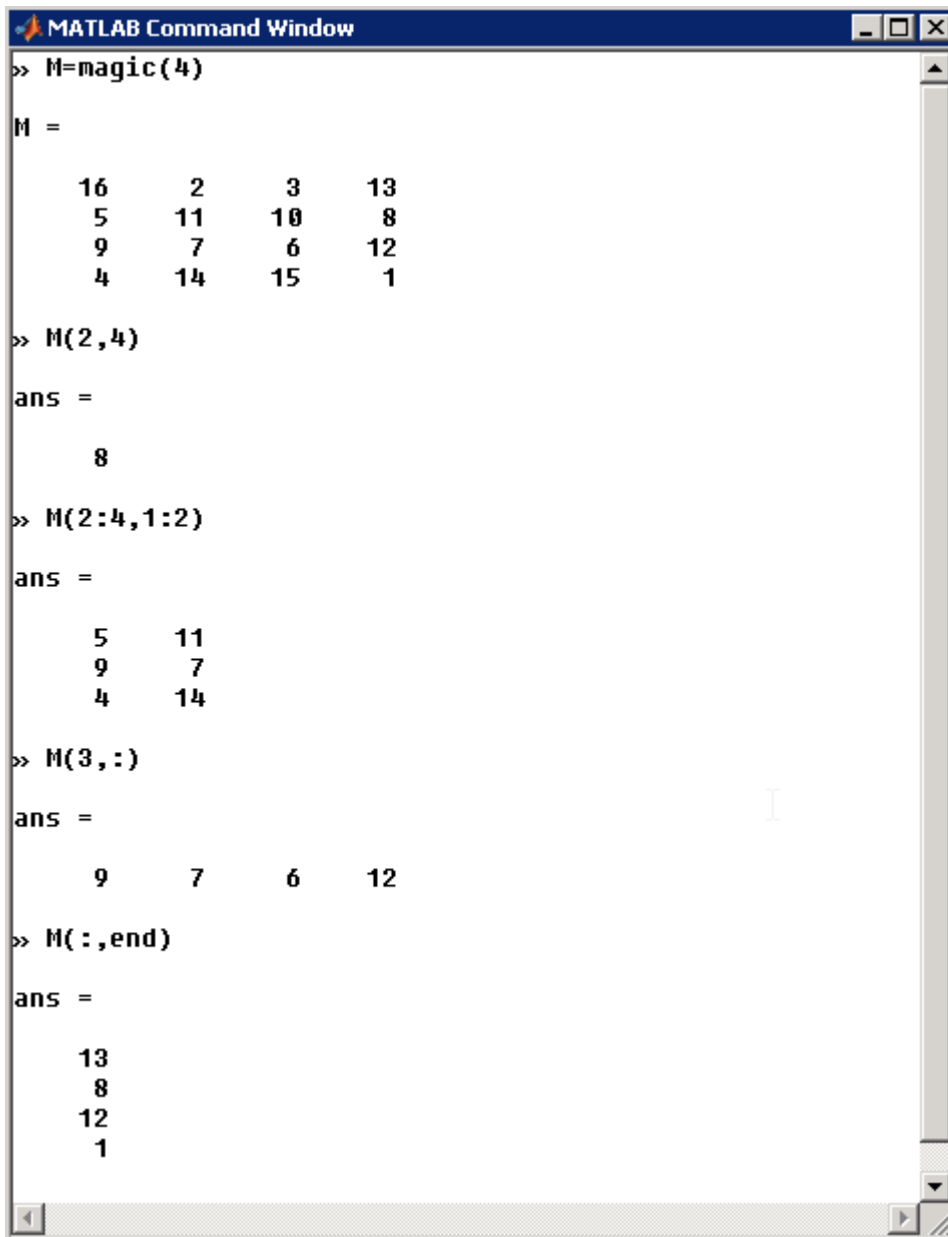
ans =

     3

>> |
```

The output for `length(M)` is 3. A red rectangular box highlights the text `%avoid! can be dangerous!` next to the `length(M)` command. A large, faint "SA" watermark is visible across the bottom half of the image.

And now some stuff about extracting some parts of a matrix:

A screenshot of the MATLAB Command Window. The window has a blue title bar with the MATLAB logo and the text "MATLAB Command Window". The command history shows the following:
1. Command: `>> M=magic(4)`
 Output: `M =` followed by a 4x4 matrix:

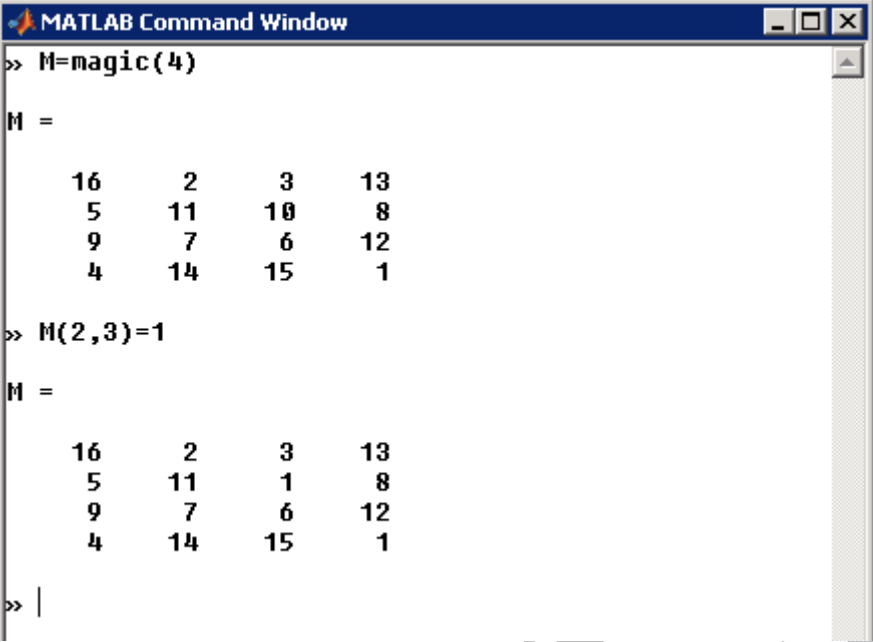
16	2	3	13
5	11	10	8
9	7	6	12
4	14	15	1

2. Command: `>> M(2,4)`
 Output: `ans =` followed by the value `8`.
3. Command: `>> M(2:4,1:2)`
 Output: `ans =` followed by a 3x2 matrix:

5	11
9	7
4	14

4. Command: `>> M(3,:)`
 Output: `ans =` followed by a 1x4 row vector: `9 7 6 12`.
5. Command: `>> M(:,end)`
 Output: `ans =` followed by a 4x1 column vector: `13 8 12 1`.
The window includes standard Windows-style window controls (minimize, maximize, close) in the top right and scroll bars on the right and bottom.

Or change the value of a given component:



```

>> M=magic(4)

M =

    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1

>> M(2,3)=1

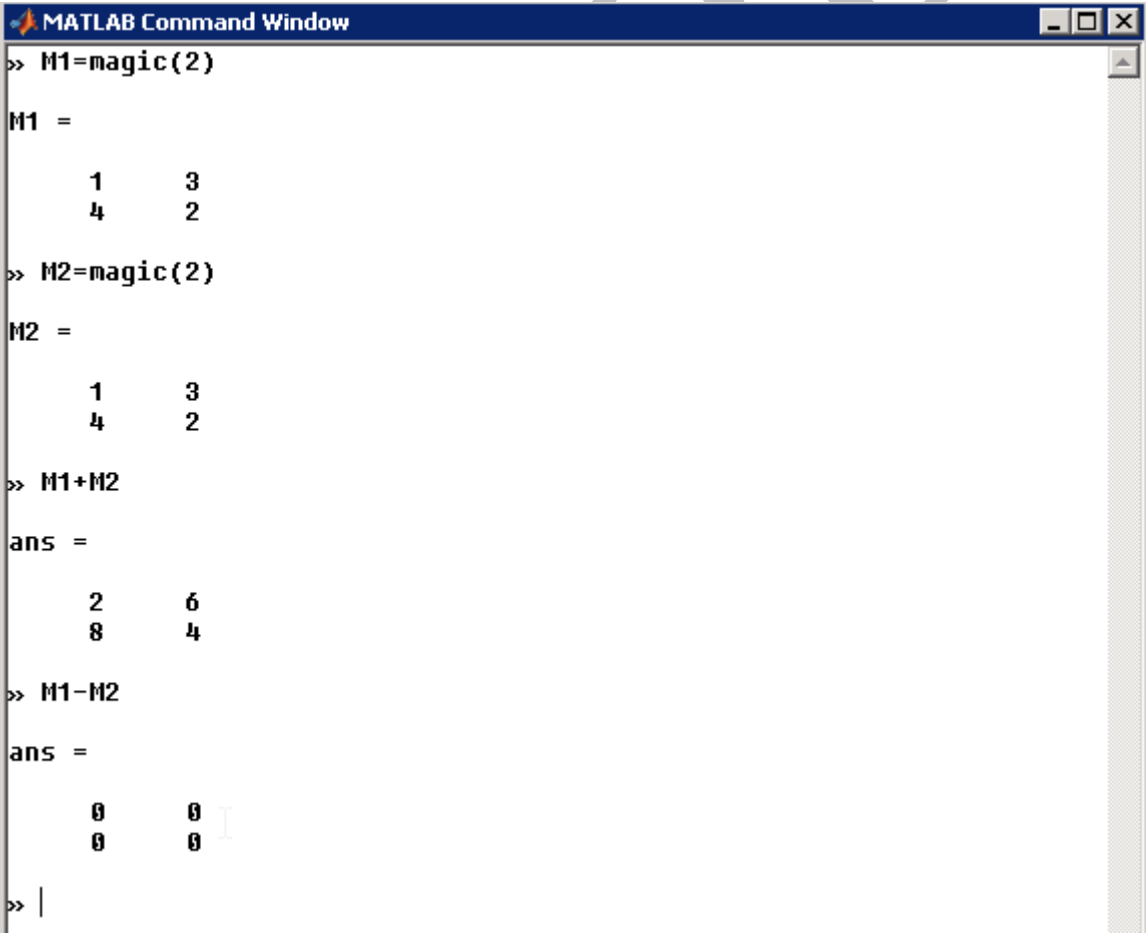
M =

    16     2     3    13
     5    11     1     8
     9     7     6    12
     4    14    15     1

>> |

```

Otherwise we can check that addin matrices of the same dimension works well:



```

>> M1=magic(2)

M1 =

     1     3
     4     2

>> M2=magic(2)

M2 =

     1     3
     4     2

>> M1+M2

ans =

     2     6
     8     4

>> M1-M2

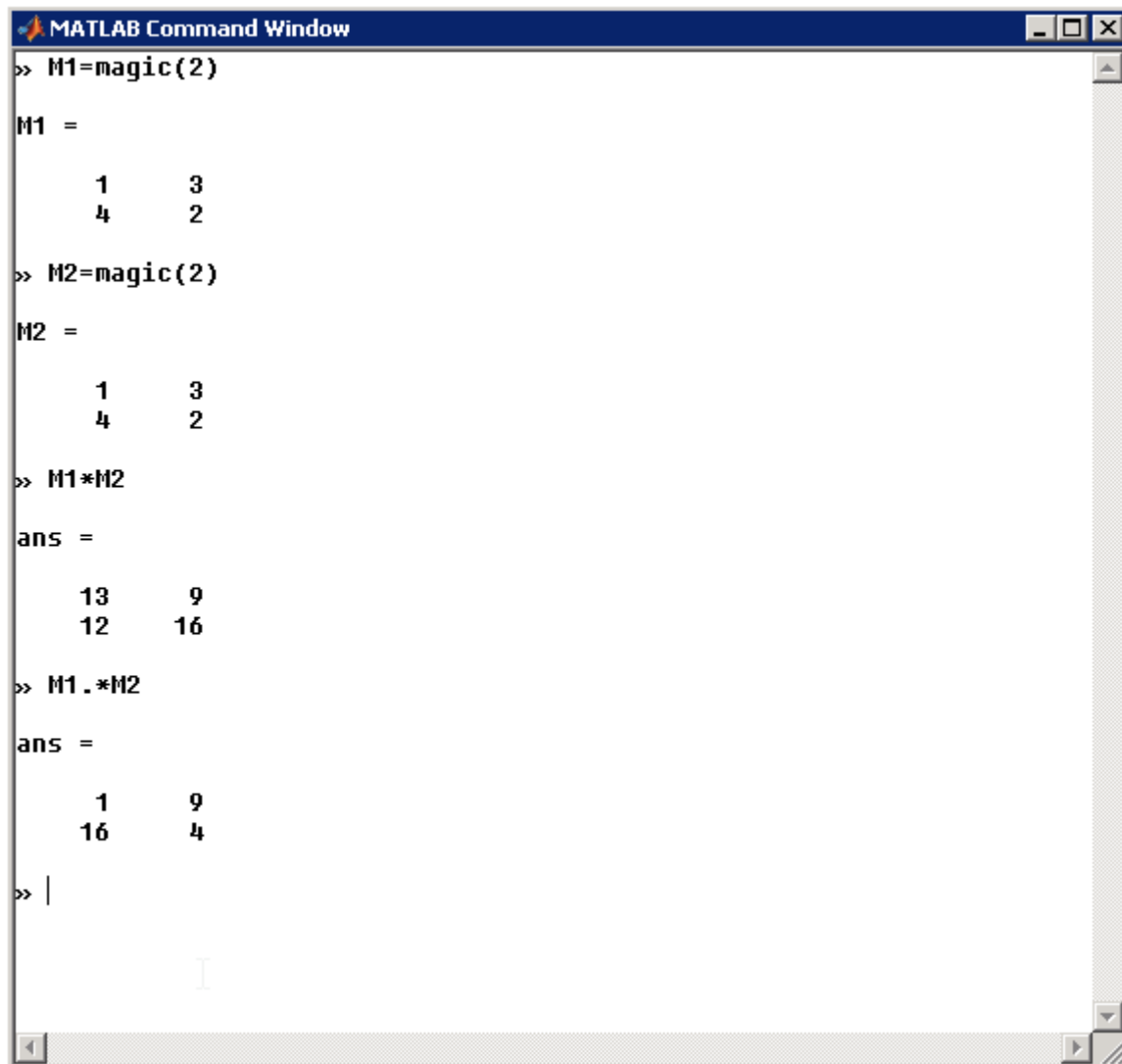
ans =

     0     0
     0     0

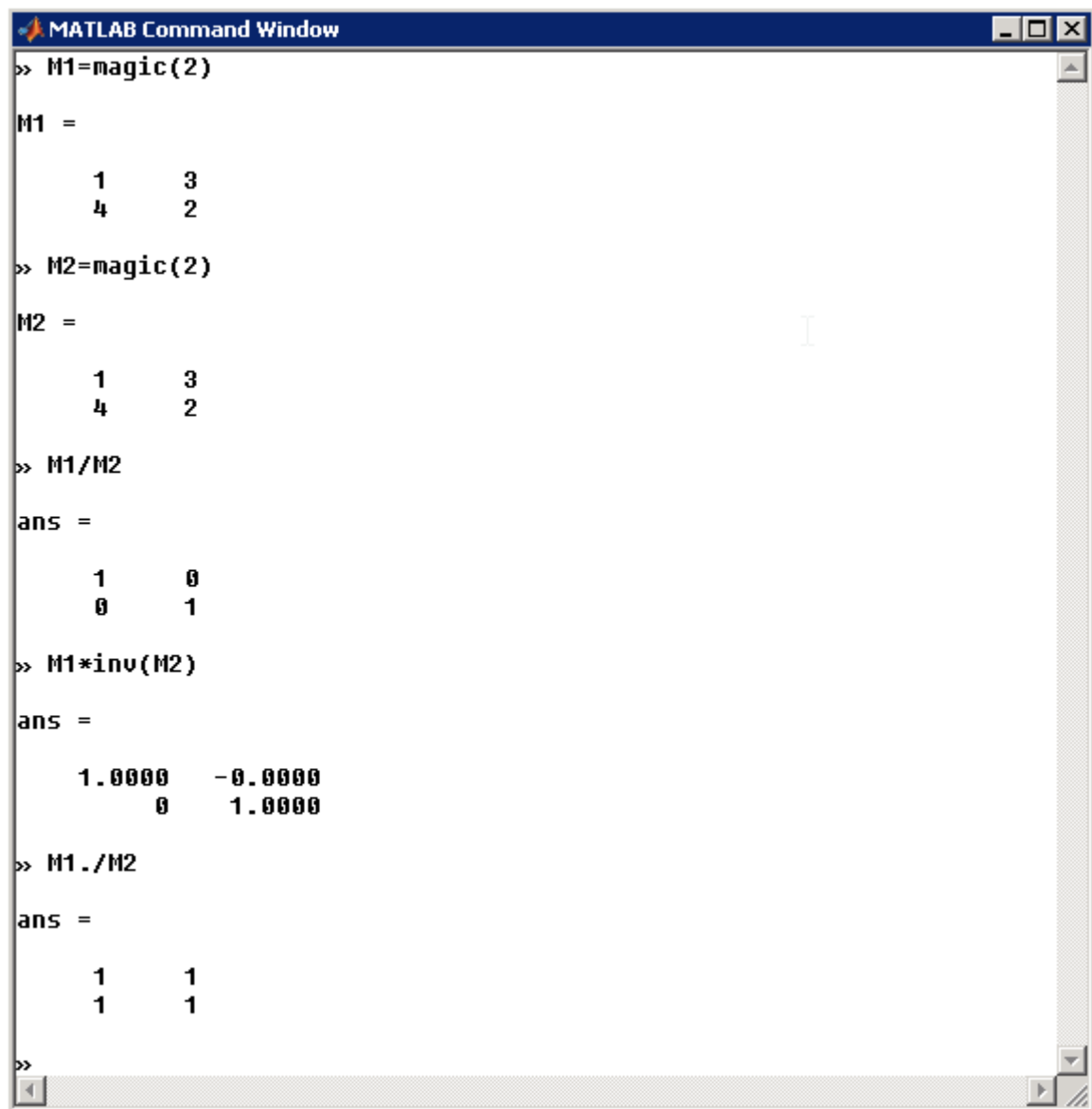
>> |

```

And also matrices multiplication if you respect linear algebra rules or that you can also make a component by component multiplication like for vectors:

A screenshot of the MATLAB Command Window. The window has a blue title bar with the MATLAB logo and the text "MATLAB Command Window". The command history shows the following:
1. Command: `M1=magic(2)`
Output: `M1 =` followed by a 2x2 matrix $\begin{bmatrix} 1 & 3 \\ 4 & 2 \end{bmatrix}$.
2. Command: `M2=magic(2)`
Output: `M2 =` followed by a 2x2 matrix $\begin{bmatrix} 1 & 3 \\ 4 & 2 \end{bmatrix}$.
3. Command: `M1*M2`
Output: `ans =` followed by a 2x2 matrix $\begin{bmatrix} 13 & 9 \\ 12 & 16 \end{bmatrix}$.
4. Command: `M1.*M2`
Output: `ans =` followed by a 2x2 matrix $\begin{bmatrix} 1 & 9 \\ 16 & 4 \end{bmatrix}$.
The prompt `>> |` is visible at the bottom of the window, indicating it is ready for the next command.

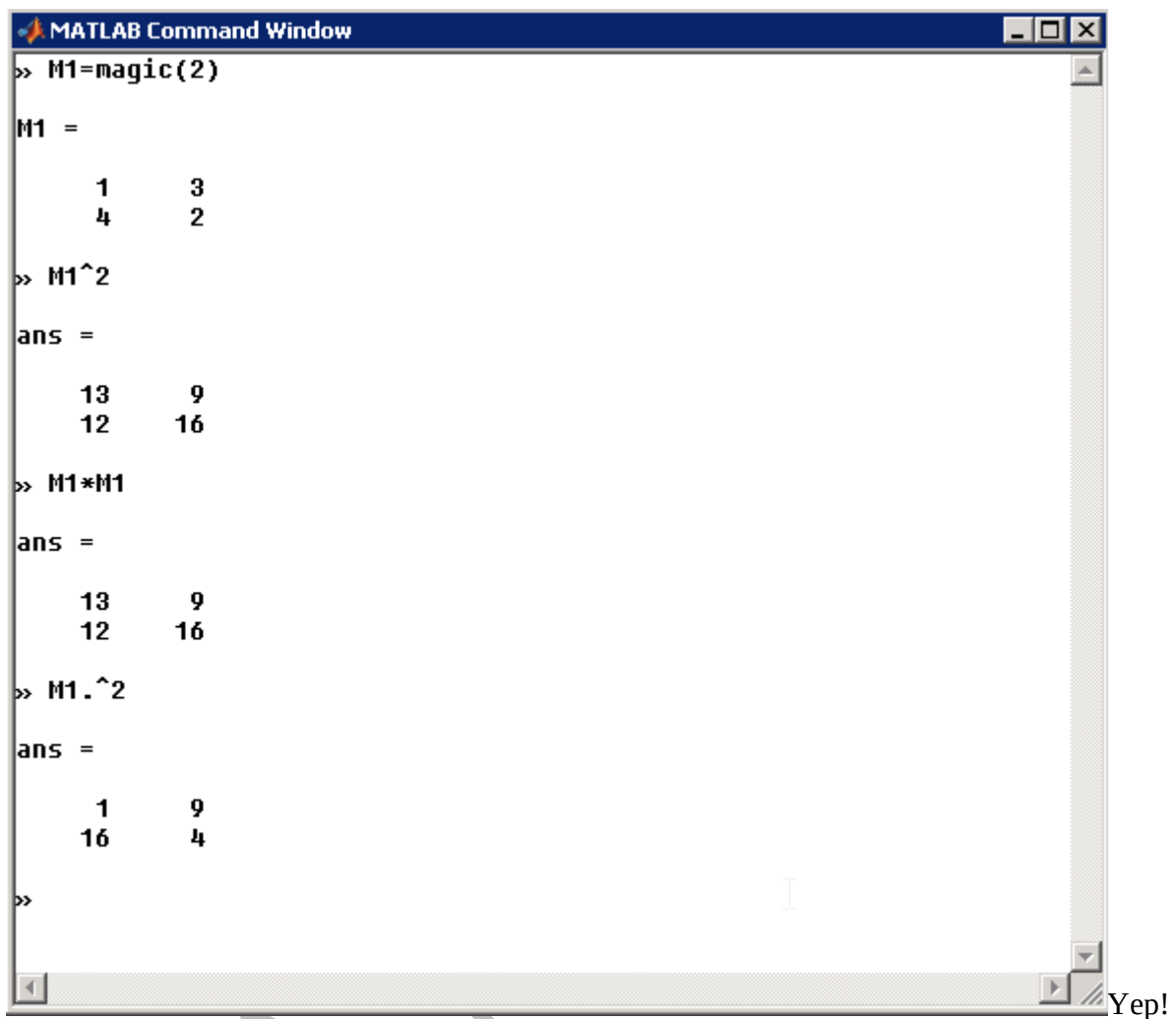
We can also take a look to what happens with division and if it's corresponding with the inverse through the function `inv()`:

A screenshot of the MATLAB Command Window. The window has a blue title bar with the MATLAB logo and the text "MATLAB Command Window". The command history shows the following:
1. Command: `M1=magic(2)`
Output: `M1 =` followed by a 2x2 matrix $\begin{bmatrix} 1 & 3 \\ 4 & 2 \end{bmatrix}$.
2. Command: `M2=magic(2)`
Output: `M2 =` followed by the same 2x2 matrix $\begin{bmatrix} 1 & 3 \\ 4 & 2 \end{bmatrix}$.
3. Command: `M1/M2`
Output: `ans =` followed by a 2x2 identity matrix $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$.
4. Command: `M1*inv(M2)`
Output: `ans =` followed by a 2x2 identity matrix with decimal formatting: $\begin{bmatrix} 1.0000 & -0.0000 \\ 0 & 1.0000 \end{bmatrix}$.
5. Command: `M1./M2`
Output: `ans =` followed by a 2x2 matrix of ones: $\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$.
The window ends with a prompt `>>`.

```
MATLAB Command Window
>> M1=magic(2)
M1 =
     1     3
     4     2
>> M2=magic(2)
M2 =
     1     3
     4     2
>> M1/M2
ans =
     1     0
     0     1
>> M1*inv(M2)
ans =
    1.0000   -0.0000
         0    1.0000
>> M1./M2
ans =
     1     1
     1     1
>>
```

Everything seems all right and standard.

Let's have a look with multiplication and power to see if everything is also standard:



The screenshot shows the MATLAB Command Window with the following commands and outputs:

```
>> M1=magic(2)

M1 =

     1     3
     4     2

>> M1^2

ans =

    13     9
    12    16

>> M1*M1

ans =

    13     9
    12    16

>> M1.^2

ans =

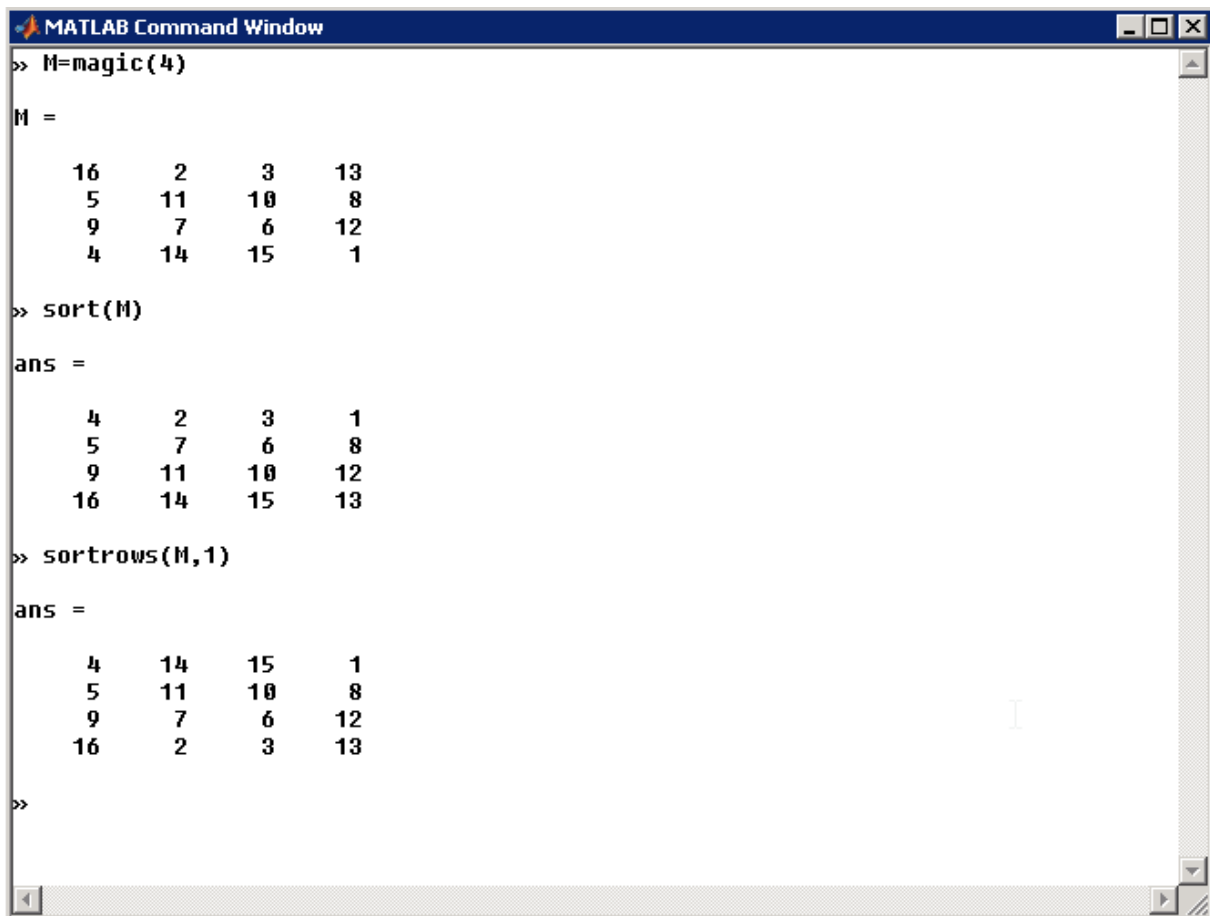
     1     9
    16     4

>>
```

At the bottom right of the window, the text "Yep!" is visible.

Everything seems fine (standard)!

And what about sorting a matrix by columns with the function `sort()` or by columns with the function `sortrows()`:



```

>> M=magic(4)

M =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1

>> sort(M)

ans =
     4     2     3     1
     5     7     6     8
     9    11    10    12
    16    14    15    13

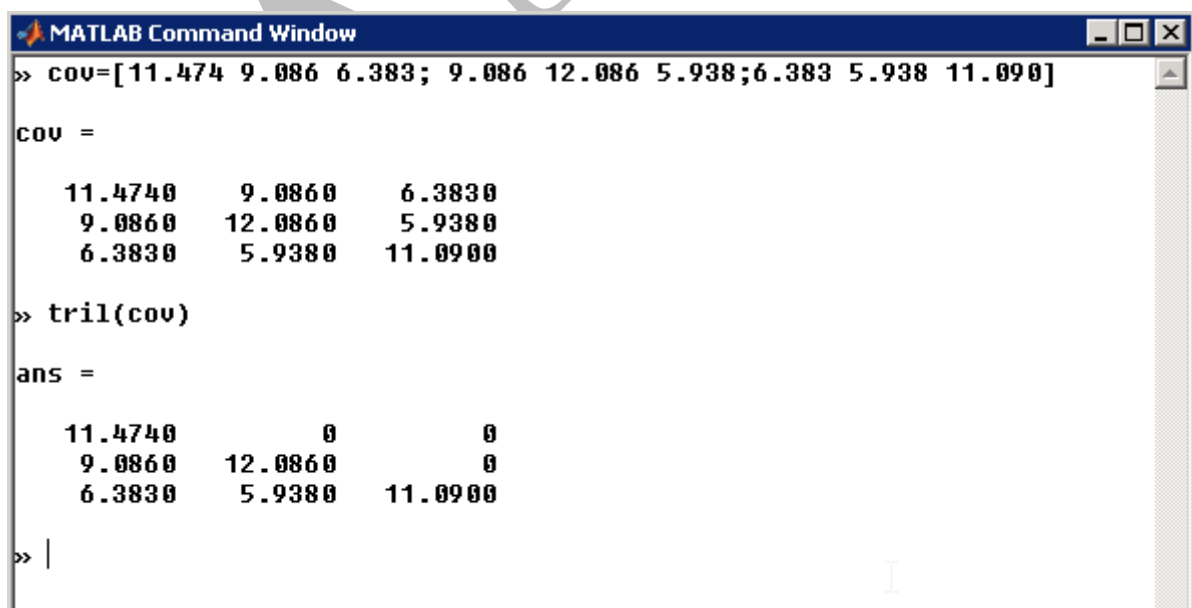
>> sortrows(M,1)

ans =
     4    14    15     1
     5    11    10     8
     9     7     6    12
    16     2     3    13

>>

```

Now suppose we have a covariance matrix and we want too keep only the triangular inferior part. Thus we can use the function `tril()`:



```

>> cov=[11.474 9.086 6.383; 9.086 12.086 5.938;6.383 5.938 11.090]

cov =
    11.4740    9.0860    6.3830
     9.0860    12.0860    5.9380
     6.3830     5.9380    11.0900

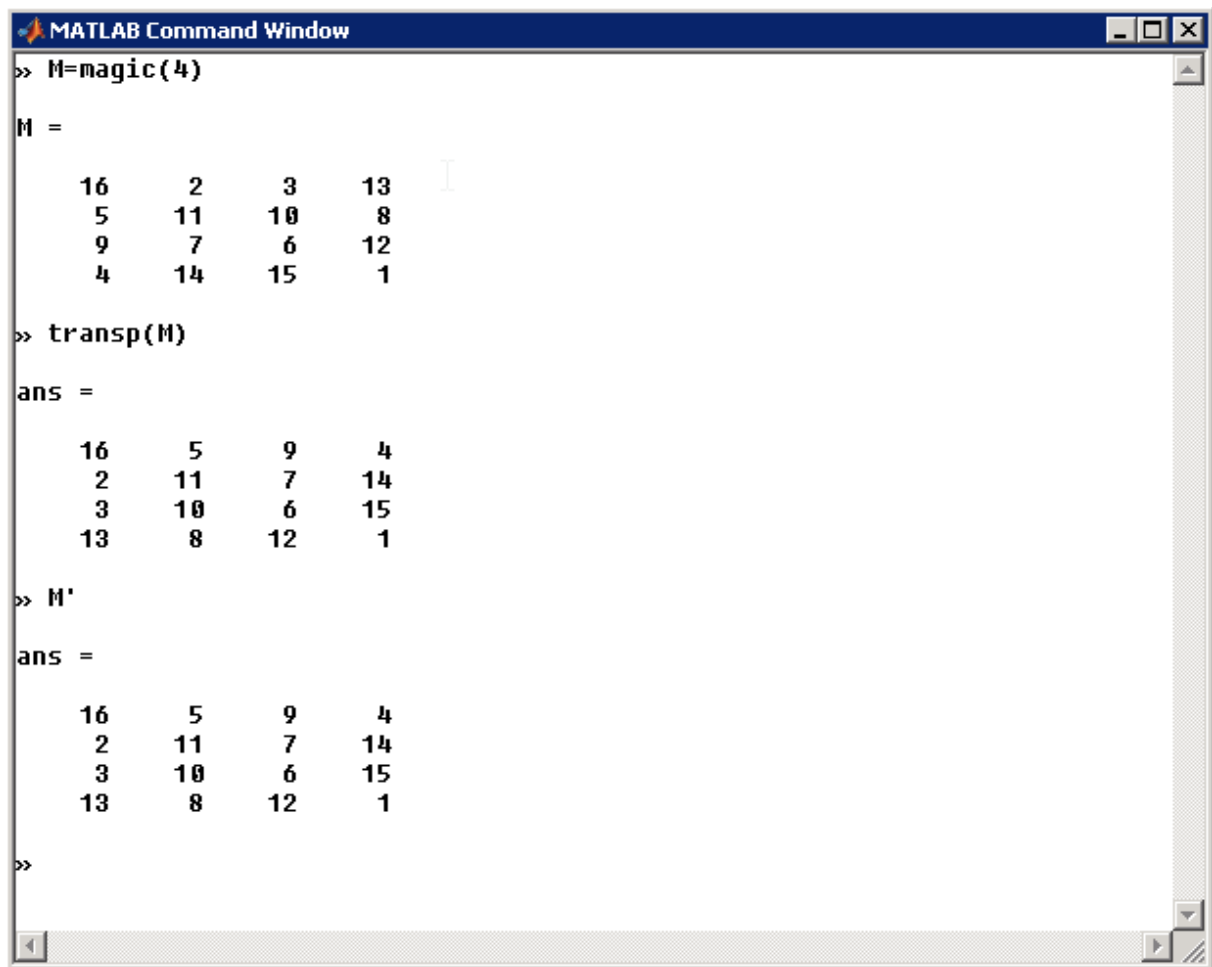
>> tril(cov)

ans =
    11.4740         0         0
     9.0860    12.0860         0
     6.3830     5.9380    11.0900

>> |

```

And have also a check to the tradition transposition with the function `transp()` or the apostroph `'`:



```
MATLAB Command Window
>> M=magic(4)

M =

    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1

>> transp(M)

ans =

    16     5     9     4
     2    11     7    14
     3    10     6    15
    13     8    12     1

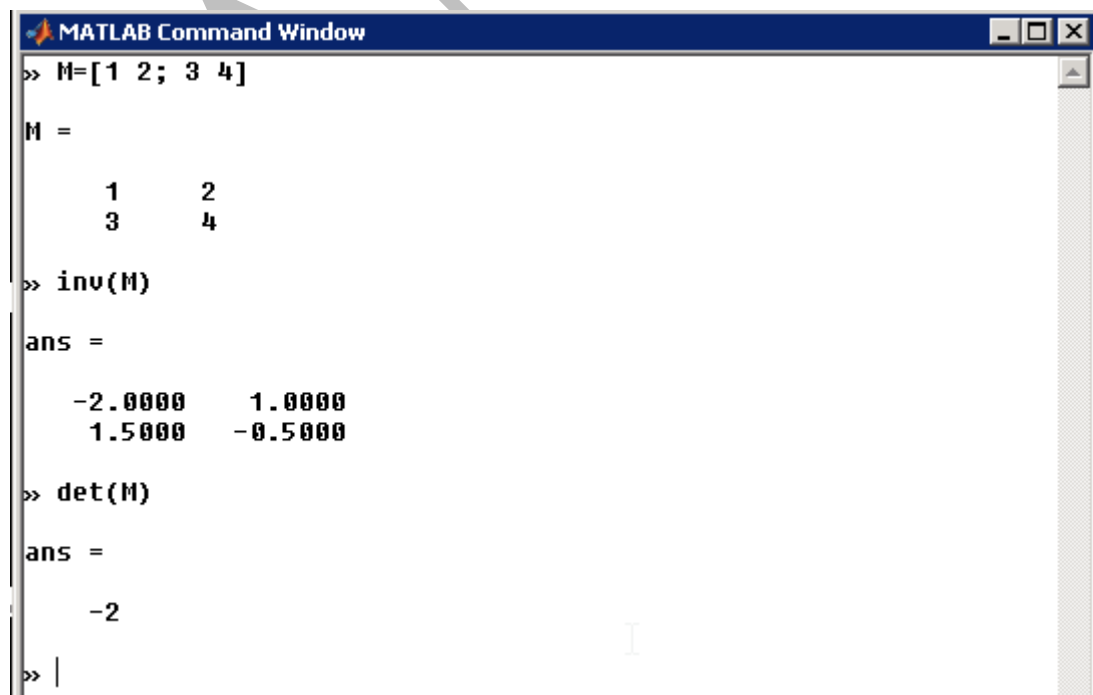
>> M'

ans =

    16     5     9     4
     2    11     7    14
     3    10     6    15
    13     8    12     1

>>
```

And we can also check that the inverse and determinant using the function **det()** are corresponding to what we have calculate by hand in the theoretical course:



```
MATLAB Command Window
>> M=[1 2; 3 4]

M =

     1     2
     3     4

>> inv(M)

ans =

   -2.0000    1.0000
    1.5000   -0.5000

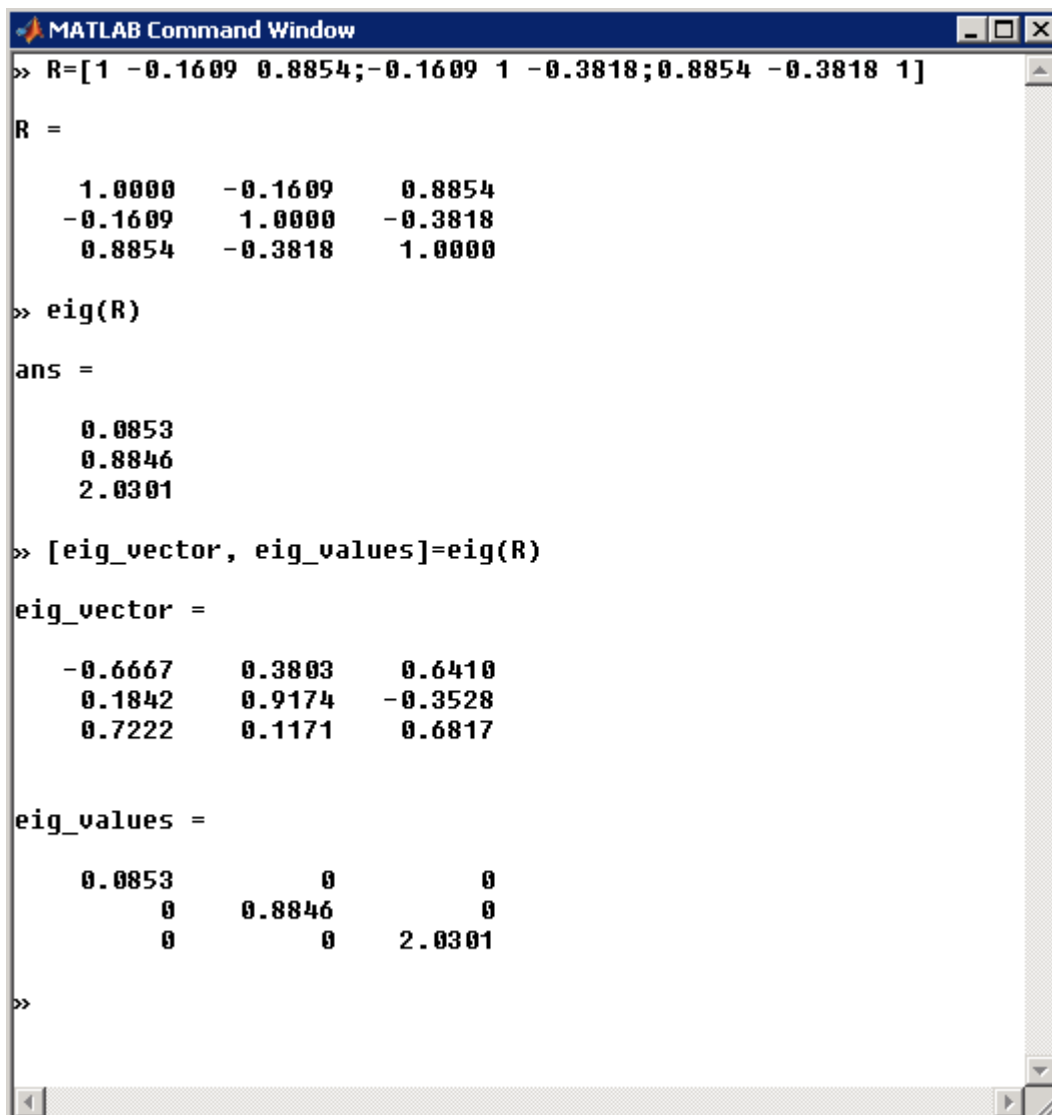
>> det(M)

ans =

   -2

>> |
```

Now have a look the correlation matrix we have caculated by hand during our study of the principal component analysis proof and let us see if we obtain the same eigenvalues and eigenvectors by using the function **eig()**:



```
MATLAB Command Window
>> R=[1 -0.1609 0.8854;-0.1609 1 -0.3818;0.8854 -0.3818 1]

R =

    1.0000   -0.1609    0.8854
   -0.1609    1.0000   -0.3818
    0.8854   -0.3818    1.0000

>> eig(R)

ans =

    0.0853
    0.8846
    2.0301

>> [eig_vector, eig_values]=eig(R)

eig_vector =

   -0.6667    0.3803    0.6410
    0.1842    0.9174   -0.3528
    0.7222    0.1171    0.6817

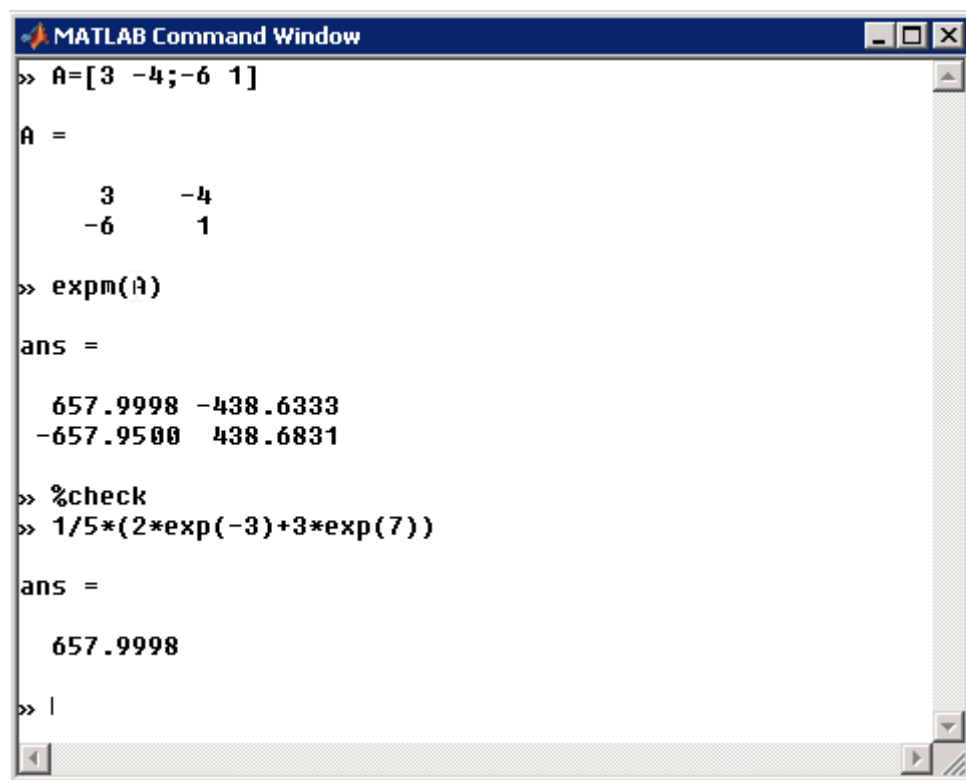
eig_values =

    0.0853     0     0
         0    0.8846     0
         0     0    2.0301

>>
```

Everything is in conformity with what we calculated.

During the course on Integration and Differential Calculus we have seen that sometimes it is required to solve ODE by using matrix exponentiation. Let us have a look if MATLAB does the job correctly by taking the same example as in the theoretical course:

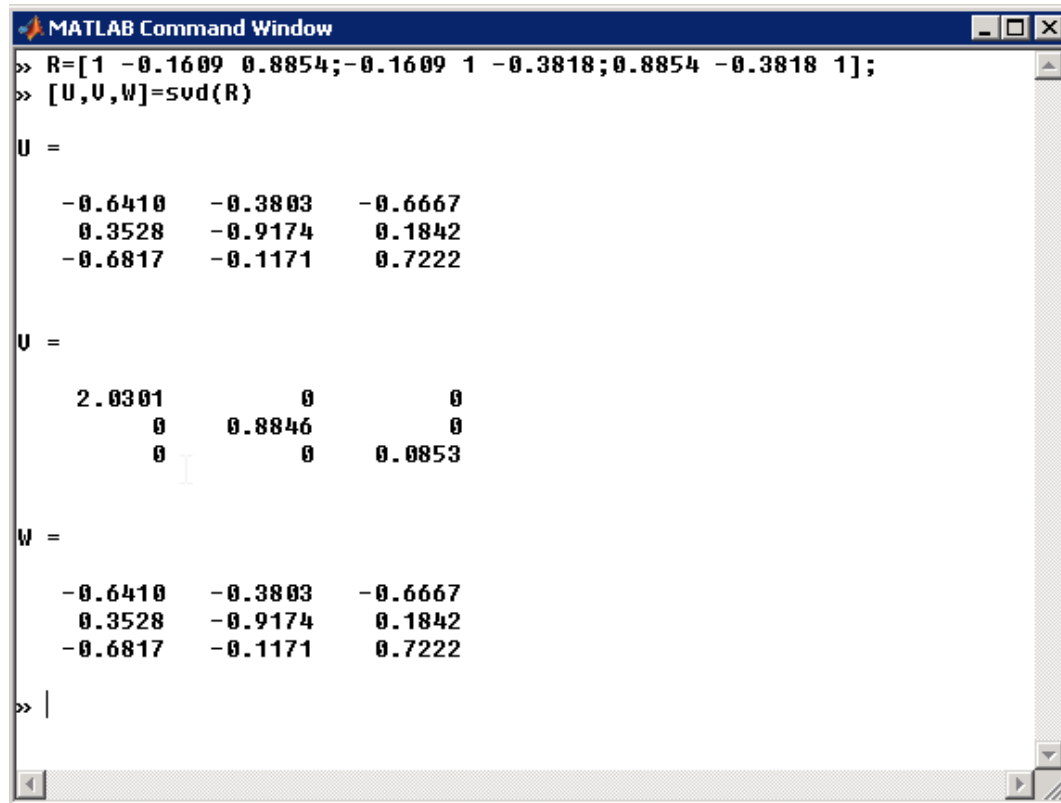


```
MATLAB Command Window
>> A=[3 -4;-6 1]
A =
     3     -4
    -6      1
>> expm(A)
ans =
    657.9998   -438.6333
   -657.9500    438.6831
>> %check
>> 1/5*(2*exp(-3)+3*exp(7))
ans =
    657.9998
>> |
```

So we obtain the same results and can validate what MATLAB returns!

In the training on Numerical Methods we have seen that the singular value decomposition has an important place in various statistical methods.

Using the correlation matrix calculated by hand in the theoretical training let us see if we obtain the same decomposition as calculated by hand by using the function `svd()`:



```

MATLAB Command Window
>> R=[1 -0.1609 0.8854;-0.1609 1 -0.3818;0.8854 -0.3818 1];
>> [U,S,W]=svd(R)

U =

   -0.6410   -0.3803   -0.6667
    0.3528   -0.9174    0.1842
   -0.6817   -0.1171    0.7222

S =

    2.0301         0         0
         0    0.8846         0
         0         0    0.0853

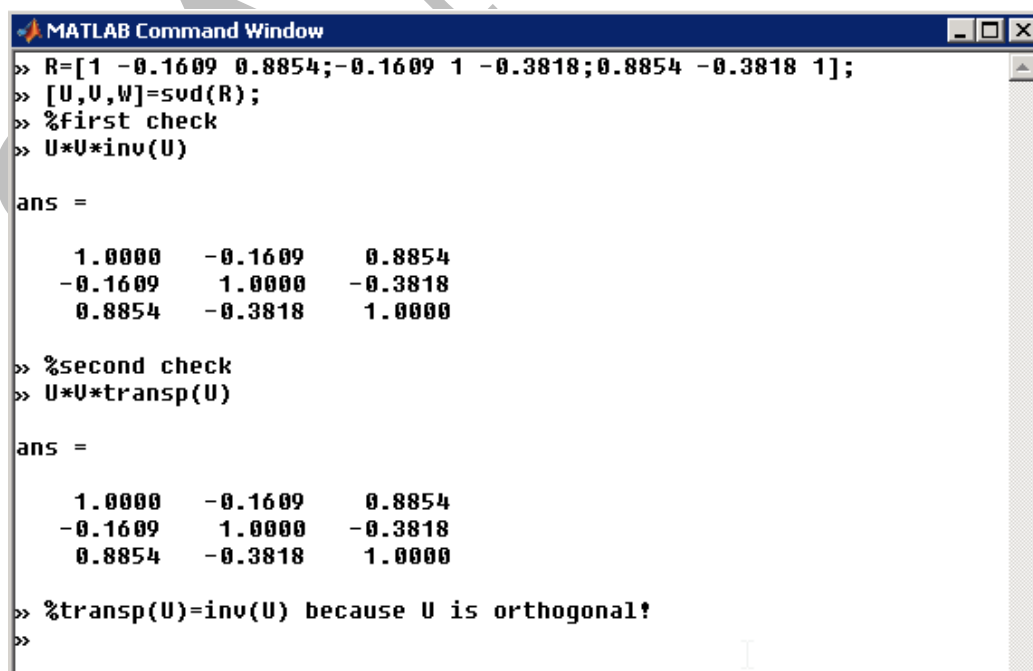
W =

   -0.6410   -0.3803   -0.6667
    0.3528   -0.9174    0.1842
   -0.6817   -0.1171    0.7222

>> |

```

We can also see for fun if it is correct by inverting the calculation:



```

MATLAB Command Window
>> R=[1 -0.1609 0.8854;-0.1609 1 -0.3818;0.8854 -0.3818 1];
>> [U,S,W]=svd(R);
>> %first check
>> U*S*inv(U)

ans =

    1.0000   -0.1609    0.8854
   -0.1609    1.0000   -0.3818
    0.8854   -0.3818    1.0000

>> %second check
>> U*S*transp(U)

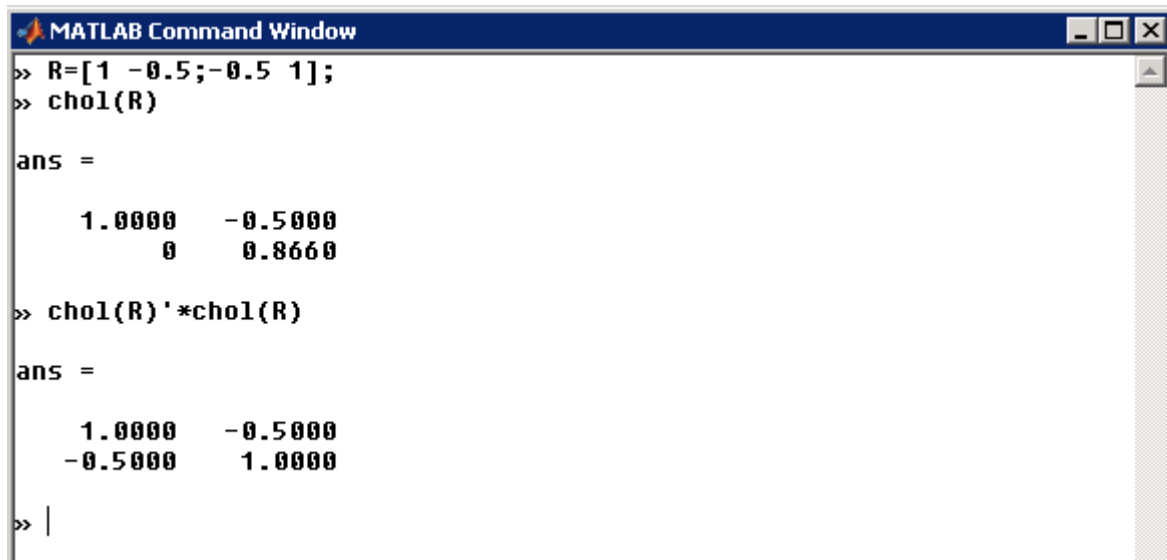
ans =

    1.0000   -0.1609    0.8854
   -0.1609    1.0000   -0.3818
    0.8854   -0.3818    1.0000

>> %transp(U)=inv(U) because U is orthogonal!
>>

```

During our study of the bivariate normal distribution we have seen the importance of the cholesy distribution. We can see if we obtain the same result as in the theoretical course by using the same correlation matrix and the using the function **chol()**:



```

MATLAB Command Window
>> R=[1 -0.5;-0.5 1];
>> chol(R)

ans =

    1.0000    -0.5000
         0     0.8660

>> chol(R)'*chol(R)

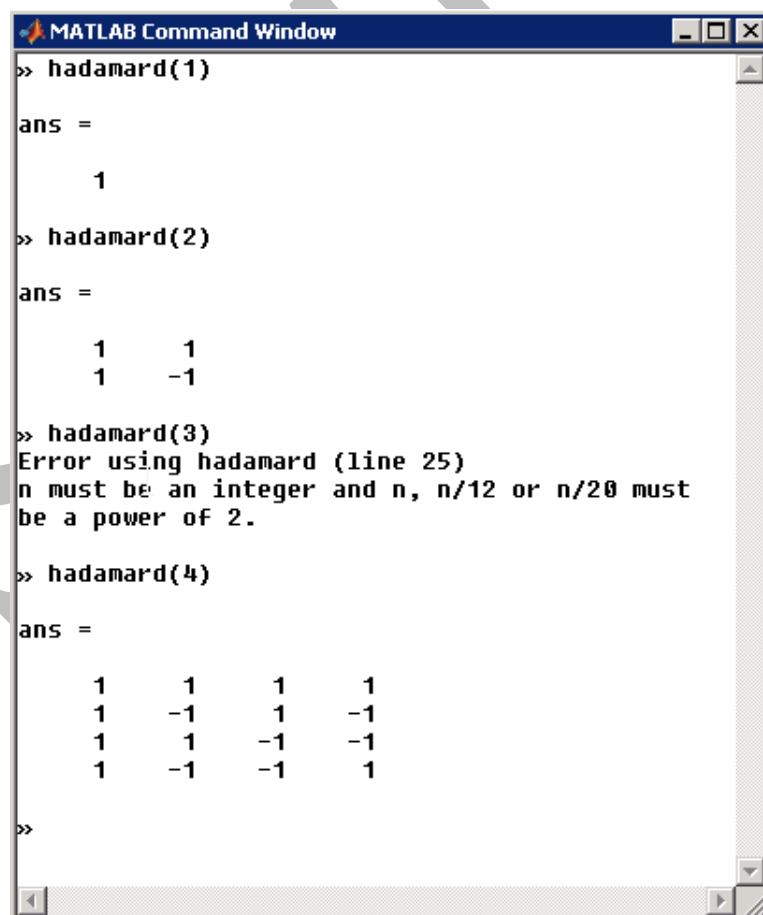
ans =

    1.0000    -0.5000
   -0.5000     1.0000

>> |

```

During the training on Design Of Experiment we have seen the importance of Hadamard matrices. Let us see if MATLAB respect the rules by using the function **hadamard()**:



```

MATLAB Command Window
>> hadamard(1)

ans =

     1

>> hadamard(2)

ans =

     1     1
     1    -1

>> hadamard(3)
Error using hadamard (line 25)
n must be an integer and n, n/12 or n/20 must
be a power of 2.

>> hadamard(4)

ans =

     1     1     1     1
     1    -1     1    -1
     1     1    -1    -1
     1    -1    -1     1

>>

```

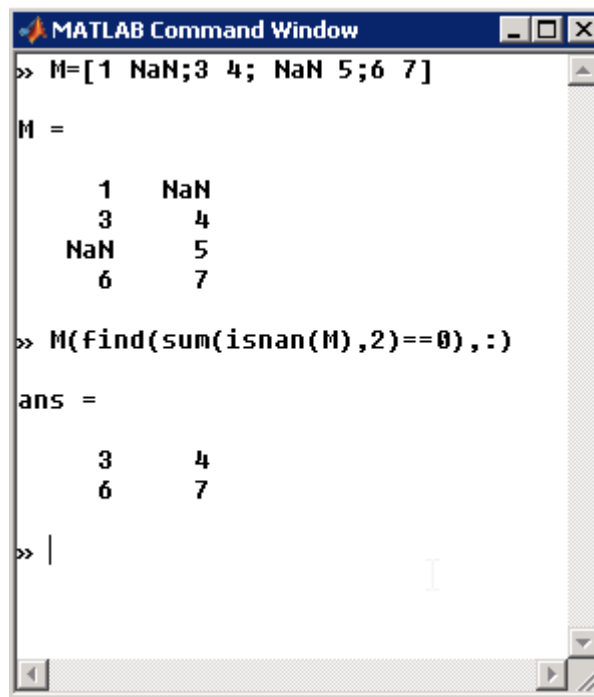
Everything is fine!

Important practical cases

This like a FAQ (frequently asked question) for which MATLAB doesn't seem to have built-in functions.

Removing NaN in matrices (arrays)

This is a typical request in finance but in the facts it could be better to interpolate using less or more advanced techniques the missing values (simple smoothing, or moving average, or spline, or...):



A screenshot of the MATLAB Command Window. The window title is "MATLAB Command Window". The command prompt shows the following sequence of commands and outputs:

```
>> M=[1 NaN;3 4; NaN 5;6 7]

M =

     1     NaN
     3         4
    NaN         5
     6         7

>> M(find(sum(isnan(M),2)==0), :)

ans =

     3         4
     6         7

>> |
```

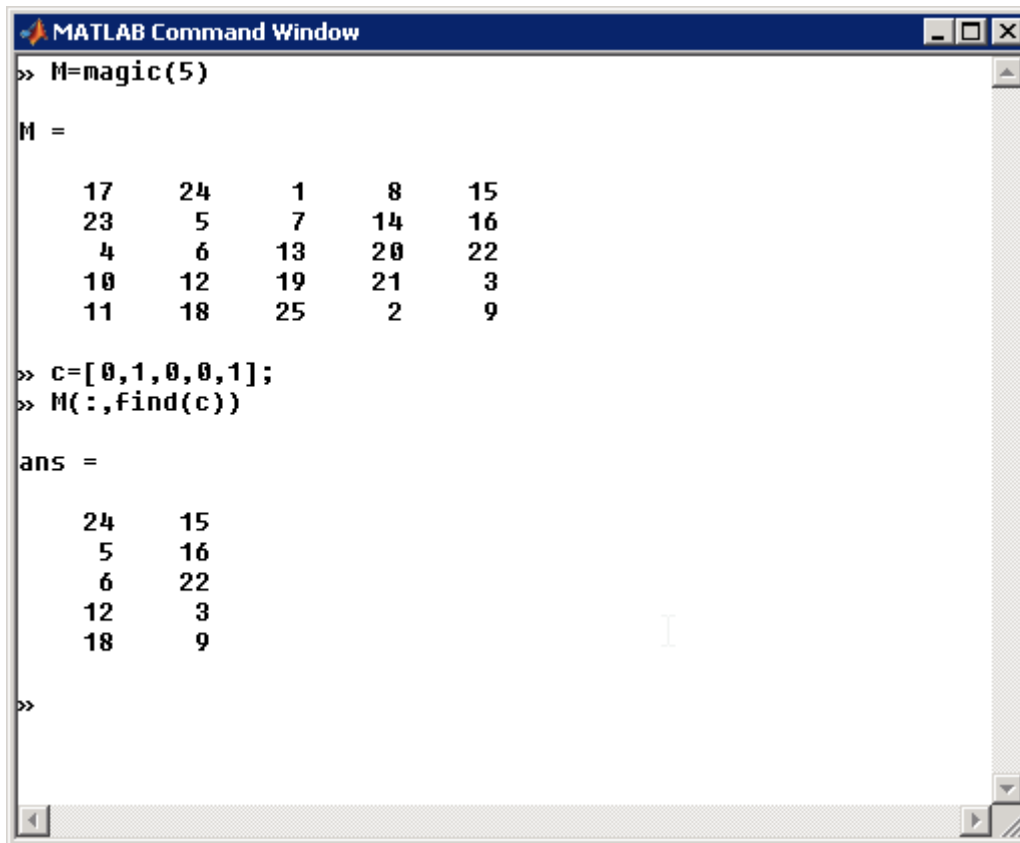
The matrix M is a 4x2 matrix with the following values:

1	NaN
3	4
NaN	5
6	7

The command `M(find(sum(isnan(M),2)==0), :)` returns the rows of M where the sum of NaN values is zero, which are the rows [3, 4] and [6, 7].

Sampling matrix columns

In finance you have sometimes thousands of assets and you want to check your algorithms applied on multiples combinations of assets returns. Suppose we have such a matrix and that we have in our possession a vector (or a set of vectors) that makes the combinations unique (typically using Hadamard matrices). Then the trick is:

A screenshot of the MATLAB Command Window. The window title is "MATLAB Command Window". The command prompt shows the execution of two commands. The first command is `M=magic(5)`, which creates a 5x5 magic square matrix M. The output shows the matrix M with its elements. The second command is `c=[0,1,0,0,1];` followed by `M(:,find(c))`. The output shows the resulting 5x2 matrix, which contains the columns of M corresponding to the non-zero elements in c (columns 2 and 5 of M).

```
>> M=magic(5)

M =

    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9

>> c=[0,1,0,0,1];
>> M(:,find(c))

ans =

    24    15
     5    16
     6    22
    12     3
    18     9

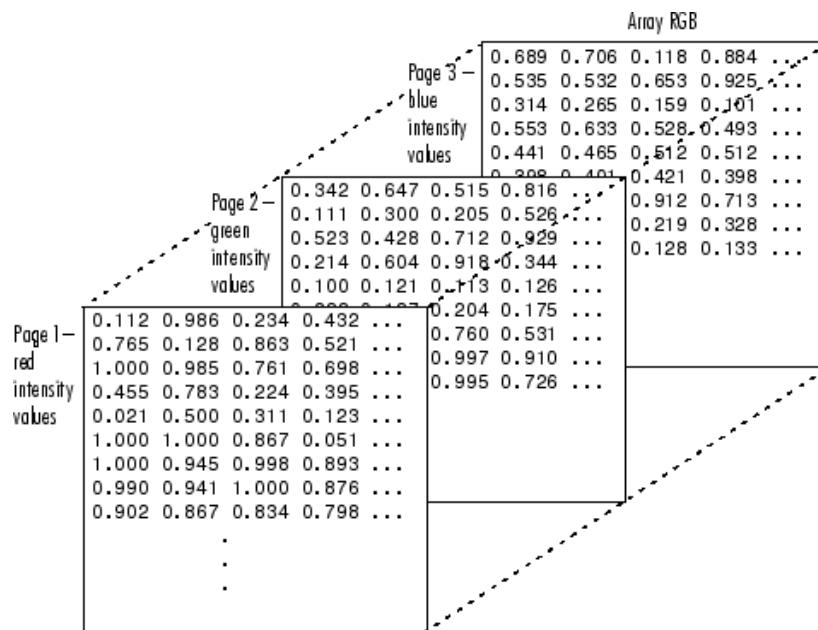
>>
```

Creation multiple dimensional matrices

When you are working in MATLAB with images, by defaults they will be described by multiples dimensional array. This also sometimes happen with measures made by engineers.

Then it is important to know how it is structured and how to create such objects.

First here is an example on how an RGB image is structured in MATLAB:



Now see how to build yourself such a multidimensional array:

```

>> M=rand(4,4)

M =

    0.9593    0.2575    0.2435    0.2511
    0.5472    0.8407    0.9293    0.6160
    0.1386    0.2543    0.3500    0.4733
    0.1493    0.8143    0.1966    0.3517

>> M(:,:,2)=rand(4,4)

M(:,:,1) =

    0.9593    0.2575    0.2435    0.2511
    0.5472    0.8407    0.9293    0.6160
    0.1386    0.2543    0.3500    0.4733
    0.1493    0.8143    0.1966    0.3517

M(:,:,2) =

    0.8308    0.2858    0.5678    0.7792
    0.5853    0.7572    0.0759    0.9340
    0.5497    0.7537    0.0540    0.1299
    0.9172    0.3804    0.5308    0.5688
  
```

Workspace

Name	Value	Min	Bytes	Max
M	<4x4x2 double>	0.0540	256	0.9593

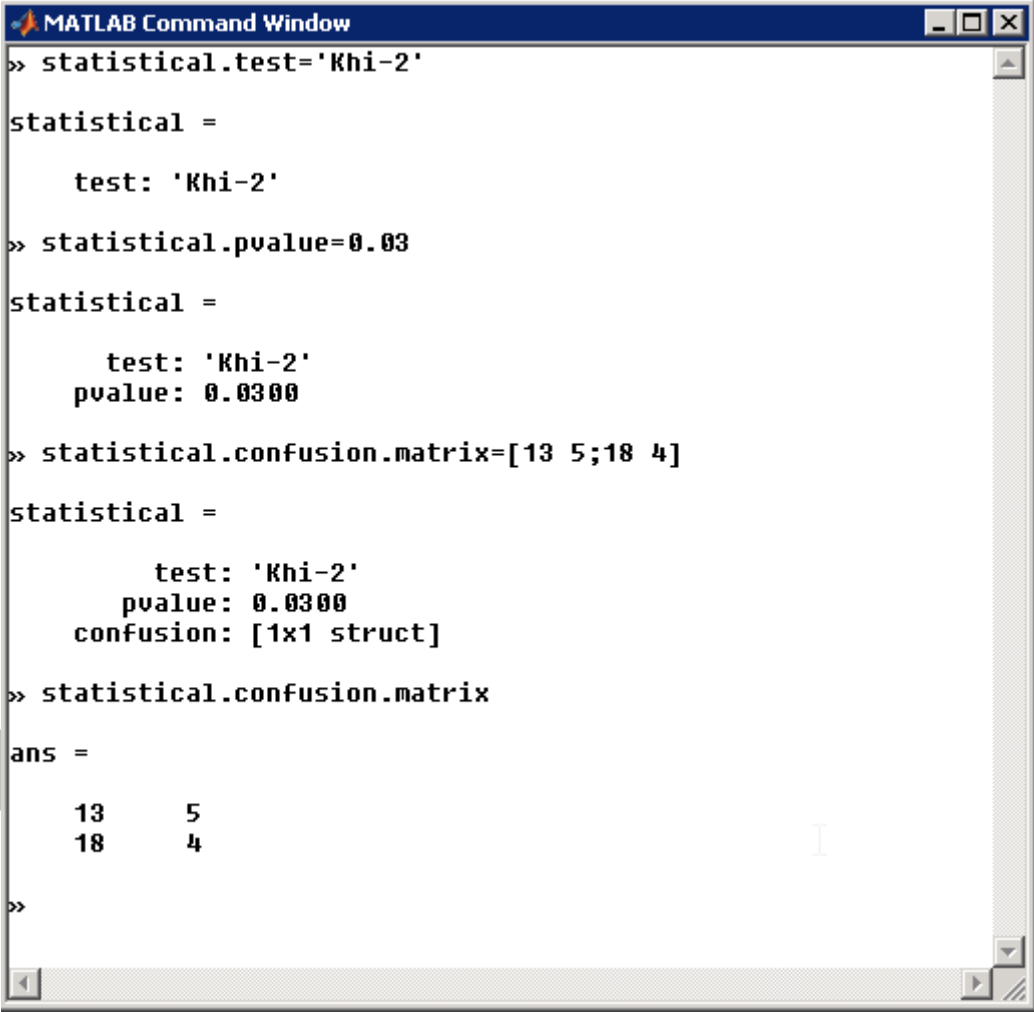
Manipulating tables (structure array)

You can since a few years work with tables structures in MATLAB like we do it for example in the R software. This is especially useful when you create your own scripts (see Scripting chapter on page 499).

You have to know that some native MATLAB functions are by default structured arrays) and we already have meet such one when we have study the **mem** command.

Let us make a typical example well knows in R.

Imagine we have build a statistical function that returns a structure named *statistical* with different properties containing some values:

A screenshot of the MATLAB Command Window. The window has a blue title bar with the MATLAB logo and the text 'MATLAB Command Window'. The command history shows three steps of building a structure array named 'statistical'.

```
>> statistical.test='Khi-2'

statistical =

    test: 'Khi-2'

>> statistical.pvalue=0.03

statistical =

    test: 'Khi-2'
    pvalue: 0.0300

>> statistical.confusion.matrix=[13 5;18 4]

statistical =

    test: 'Khi-2'
    pvalue: 0.0300
    confusion: [1x1 struct]

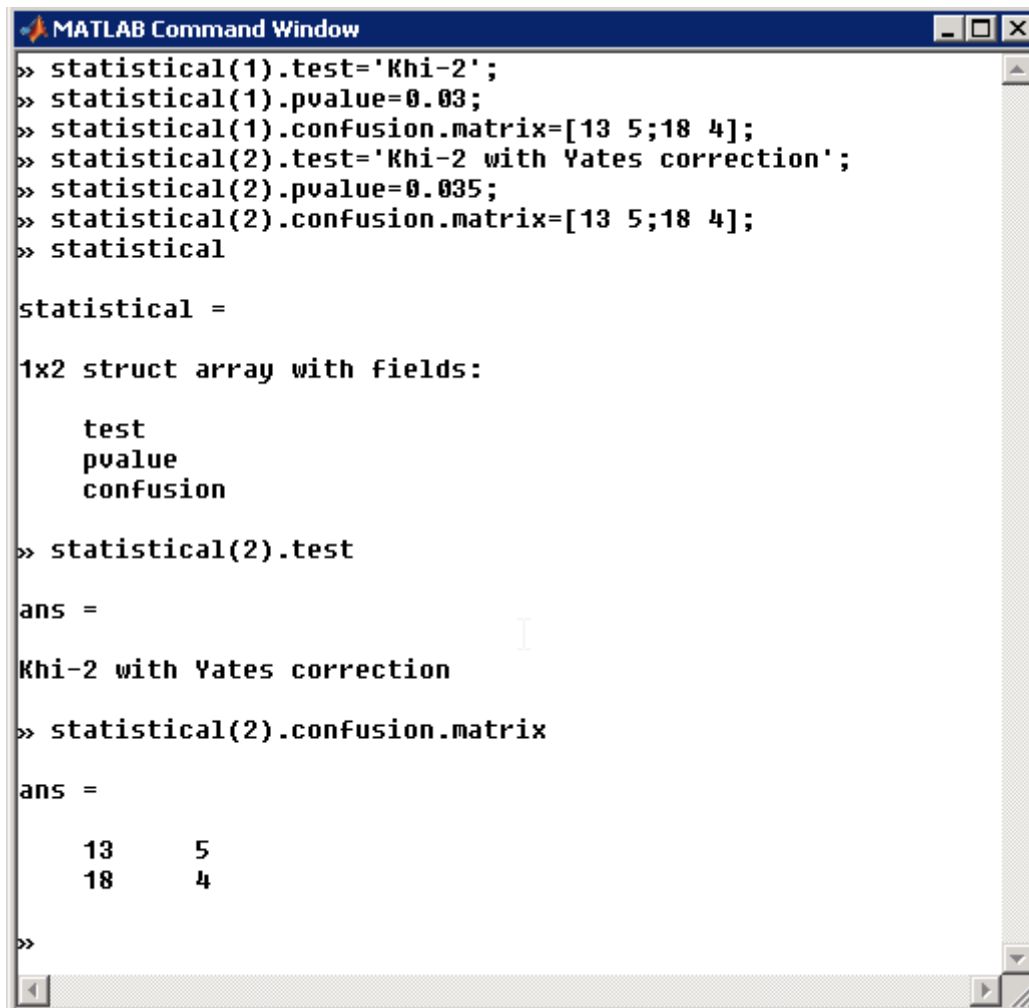
>> statistical.confusion.matrix

ans =

    13     5
    18     4

>>
```

Now let us see how to access to the different properties values and to different property values:



```
MATLAB Command Window

>> statistical(1).test='Khi-2';
>> statistical(1).pvalue=0.03;
>> statistical(1).confusion.matrix=[13 5;18 4];
>> statistical(2).test='Khi-2 with Yates correction';
>> statistical(2).pvalue=0.035;
>> statistical(2).confusion.matrix=[13 5;18 4];
>> statistical

statistical =

1x2 struct array with fields:

    test
   pvalue
 confusion

>> statistical(2).test

ans =

Khi-2 with Yates correction

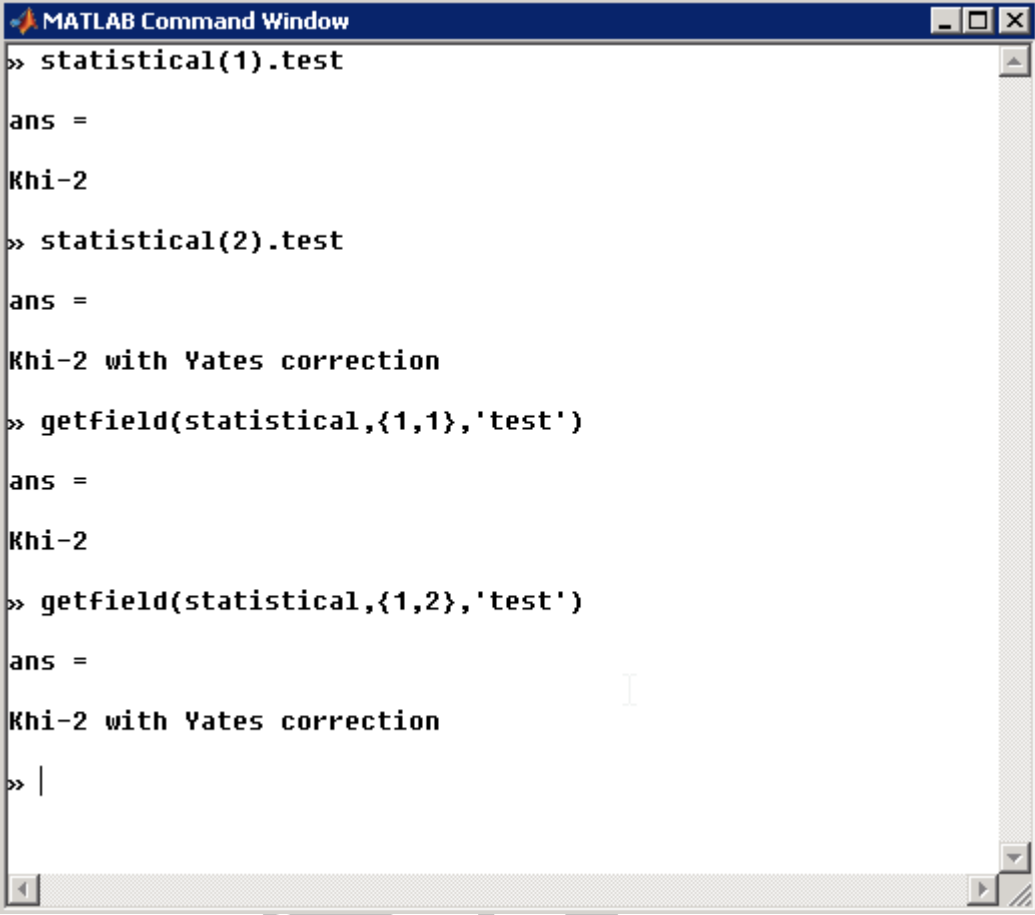
>> statistical(2).confusion.matrix

ans =

    13     5
    18     4

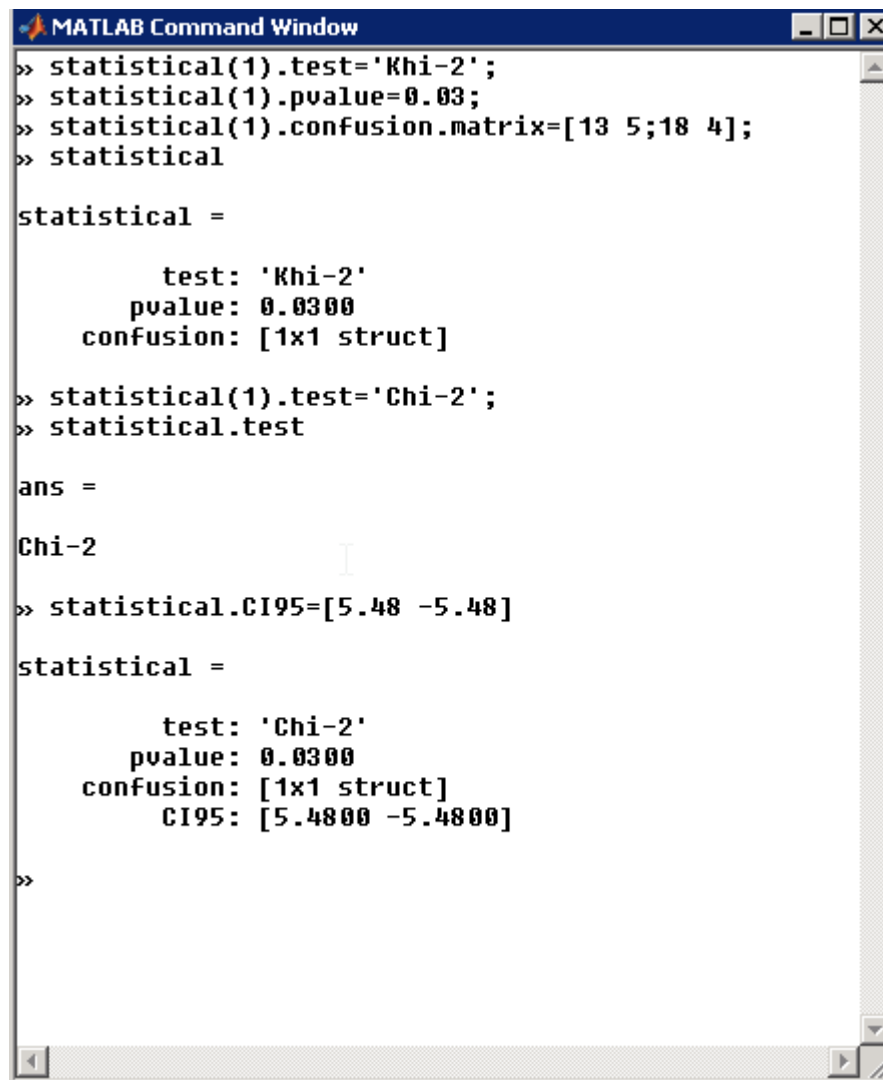
>>
```

You must also know that there is another way to access to properties values:

A screenshot of the MATLAB Command Window. The window has a blue title bar with the MATLAB logo and the text "MATLAB Command Window". The command history shows the following:
1. Command: `statistical(1).test`
Output: `ans =`
`Khi-2`
2. Command: `statistical(2).test`
Output: `ans =`
`Khi-2 with Yates correction`
3. Command: `getfield(statistical,{1,1},'test')`
Output: `ans =`
`Khi-2`
4. Command: `getfield(statistical,{1,2},'test')`
Output: `ans =`
`Khi-2 with Yates correction`
The cursor is at the prompt `>> |` on the next line. A large, light gray "SAMPLE" watermark is diagonally across the image.

```
MATLAB Command Window
>> statistical(1).test
ans =
Khi-2
>> statistical(2).test
ans =
Khi-2 with Yates correction
>> getfield(statistical,{1,1},'test')
ans =
Khi-2
>> getfield(statistical,{1,2},'test')
ans =
Khi-2 with Yates correction
>> |
```

We can at anytime change the value of each of the properties:

A screenshot of the MATLAB Command Window. The window title is 'MATLAB Command Window'. The command history shows several lines of code: 'statistical(1).test='Khi-2''; 'statistical(1).pvalue=0.03; 'statistical(1).confusion.matrix=[13 5;18 4]; and 'statistical'. The output shows 'statistical =' followed by a struct with fields: 'test: 'Khi-2'', 'pvalue: 0.0300', and 'confusion: [1x1 struct]'. Then, the command 'statistical(1).test='Chi-2'' is entered, followed by 'statistical.test'. The output shows 'ans =' followed by 'Chi-2'. Finally, the command 'statistical.CI95=[5.48 -5.48]' is entered. The output shows 'statistical =' followed by a struct with fields: 'test: 'Chi-2'', 'pvalue: 0.0300', 'confusion: [1x1 struct]', and 'CI95: [5.4800 -5.4800]'. The prompt '»' is visible at the bottom.

```
» statistical(1).test='Khi-2';
» statistical(1).pvalue=0.03;
» statistical(1).confusion.matrix=[13 5;18 4];
» statistical

statistical =

    test: 'Khi-2'
    pvalue: 0.0300
    confusion: [1x1 struct]

» statistical(1).test='Chi-2';
» statistical.test

ans =

Chi-2

» statistical.CI95=[5.48 -5.48]

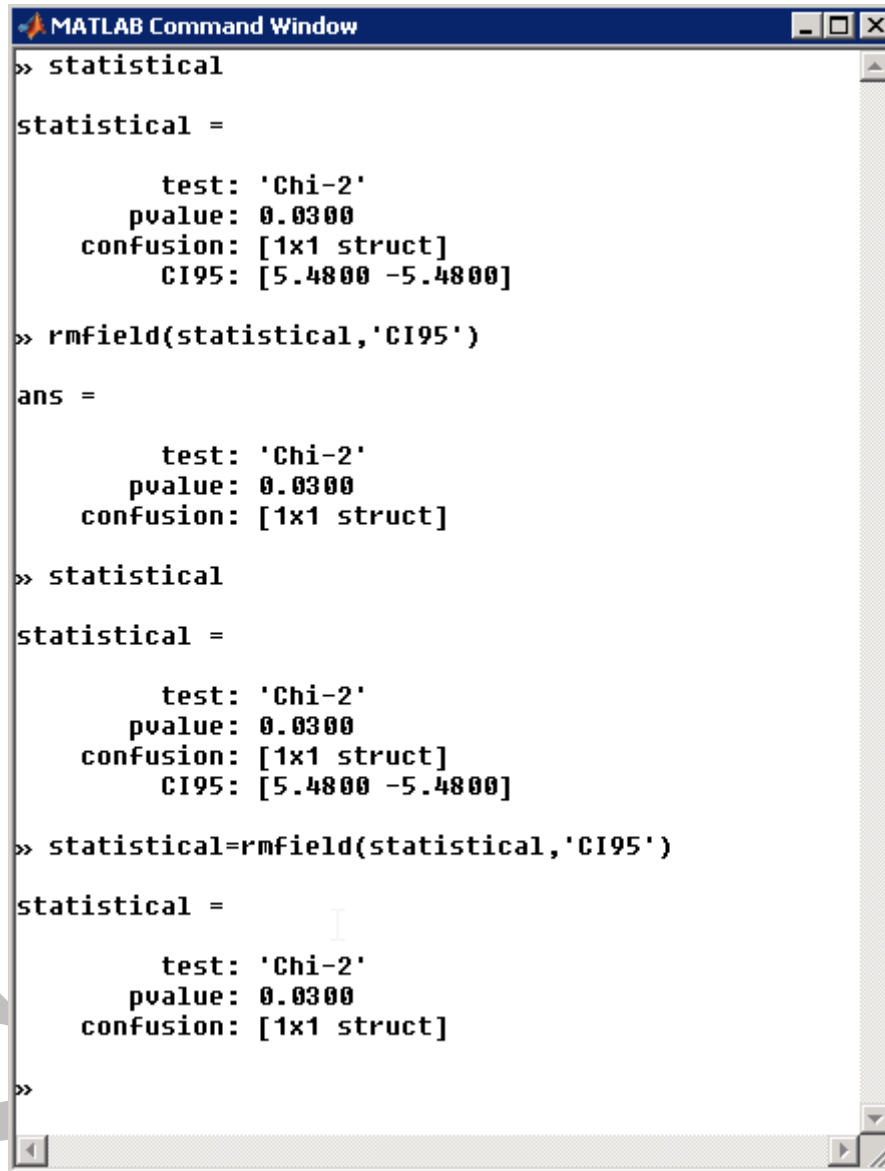
statistical =

    test: 'Chi-2'
    pvalue: 0.0300
    confusion: [1x1 struct]
    CI95: [5.4800 -5.4800]

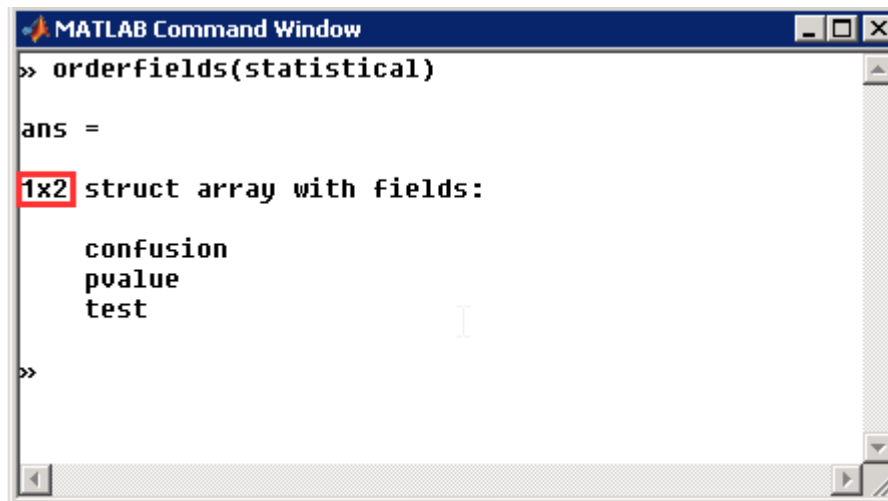
»
```

If you write the name of the structure you will by default get only the first dimension properties.

Using the function **rmfield()** we can temporarily hide a property when we call a structured array:

A screenshot of the MATLAB Command Window. The window title is "MATLAB Command Window". The command prompt shows the following sequence of commands and outputs:
1. Command: `>> statistical`
Output: `statistical =`
 test: 'Chi-2'
 pvalue: 0.0300
 confusion: [1x1 struct]
 CI95: [5.4800 -5.4800]
2. Command: `>> rmfield(statistical,'CI95')`
Output: `ans =`
 test: 'Chi-2'
 pvalue: 0.0300
 confusion: [1x1 struct]
3. Command: `>> statistical`
Output: `statistical =`
 test: 'Chi-2'
 pvalue: 0.0300
 confusion: [1x1 struct]
 CI95: [5.4800 -5.4800]
4. Command: `>> statistical=rmfield(statistical,'CI95')`
Output: `statistical =`
 test: 'Chi-2'
 pvalue: 0.0300
 confusion: [1x1 struct]
5. Command: `>>`

We can get the list of the names of the structured array using the function **orderfields()** and at the same time we get the number of dimensions:



```
MATLAB Command Window
>> orderfields(statistical)

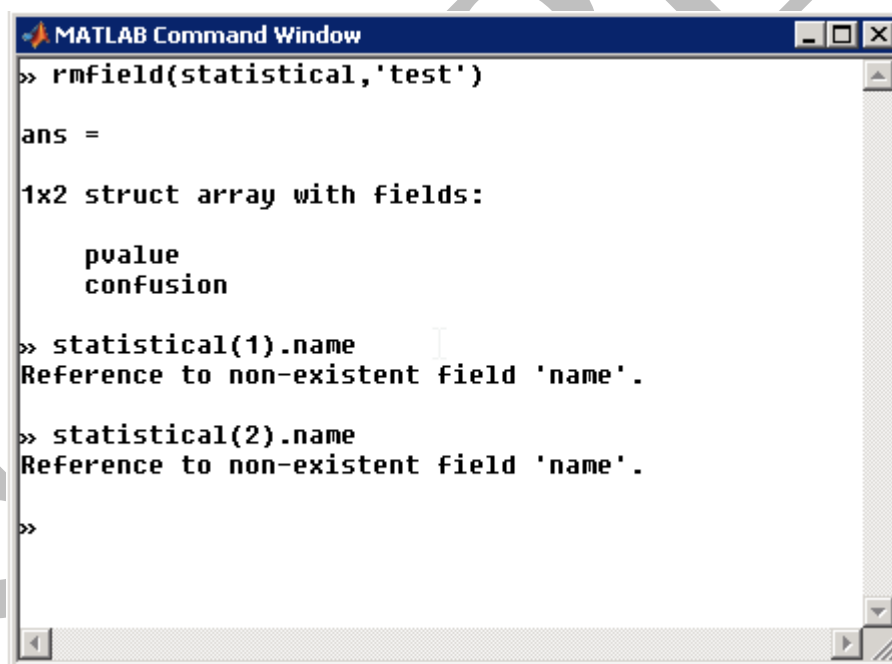
ans =

1x2 struct array with fields:

    confusion
    pvalue
    test

>>
```

Strange behaviour... if we remove a field containing a string this will be definitely removed at the contrary of a property containing a matrix:



```
MATLAB Command Window
>> rmfield(statistical,'test')

ans =

1x2 struct array with fields:

    pvalue
    confusion

>> statistical(1).name
Reference to non-existent field 'name'.

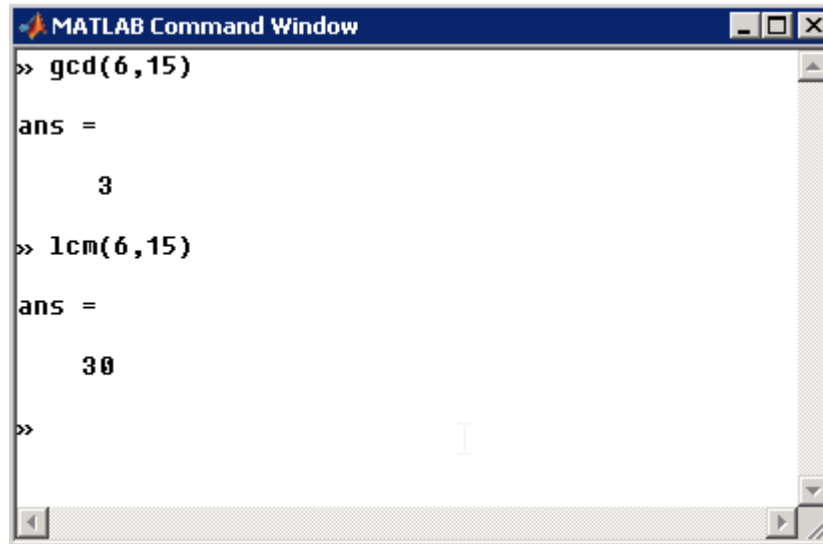
>> statistical(2).name
Reference to non-existent field 'name'.

>>
```

Mathematical functions

The purpose now is to check that MATLAB has some of the mathematical function that we used we saw the Applied Mathematics e-book.

Let us start by the greatest common divisor with the function **gcd()** and also the lower common multiple with the function **lcm()**:



```
MATLAB Command Window
>> gcd(6,15)

ans =

     3

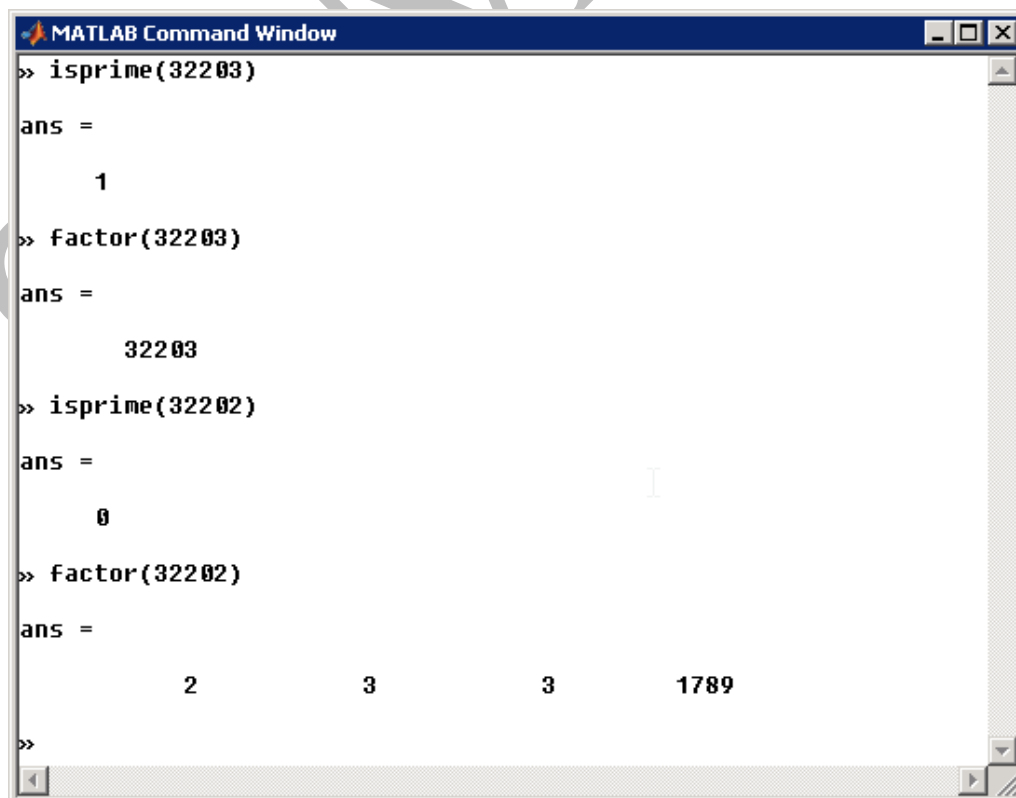
>> lcm(6,15)

ans =

    30

>>
```

We also know that in R.S.A. cryptography that the manipulation of prime number is important. For this purpose MATLAB has some prime relative functions like **isprime()**, and **factor()**:



```
MATLAB Command Window
>> isprime(32203)

ans =

     1

>> factor(32203)

ans =

    32203

>> isprime(32202)

ans =

     0

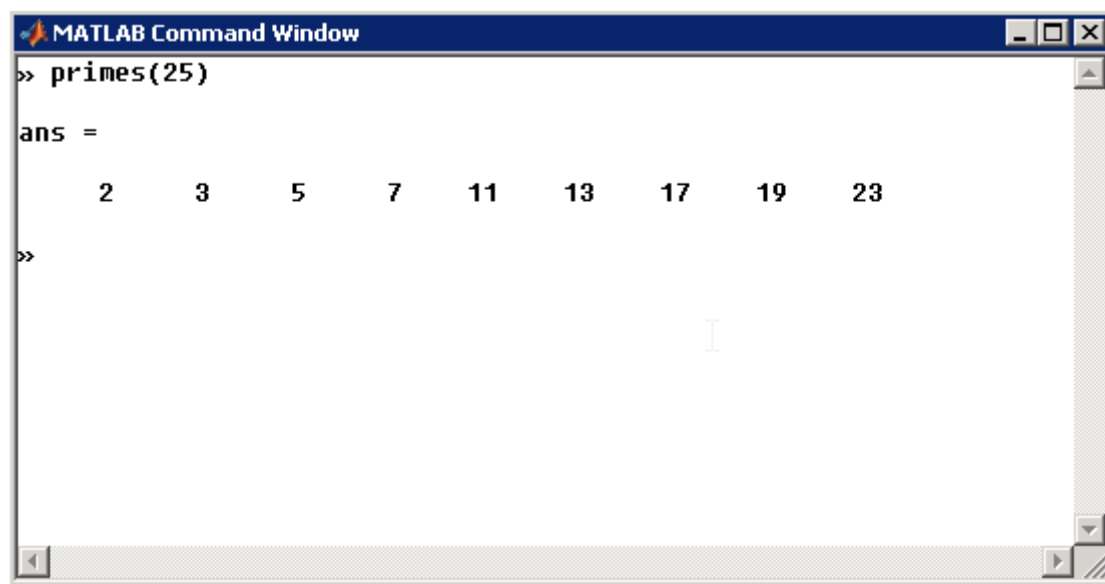
>> factor(32202)

ans =

     2     3     3    1789

>>
```

and also **primes()**:

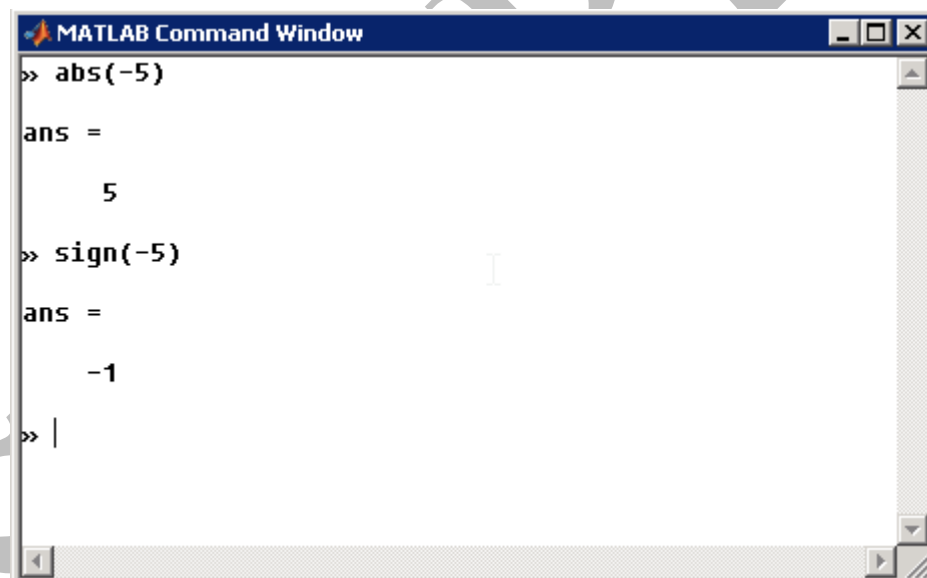


```
MATLAB Command Window
>> primes(25)

ans =

     2     3     5     7    11    13    17    19    23
>>
```

For sure we have the absolute value that we have already seen twice before with the function **abs()** and the signum function **sign()** that is still useful in numerous applied field:



```
MATLAB Command Window
>> abs(-5)

ans =

     5

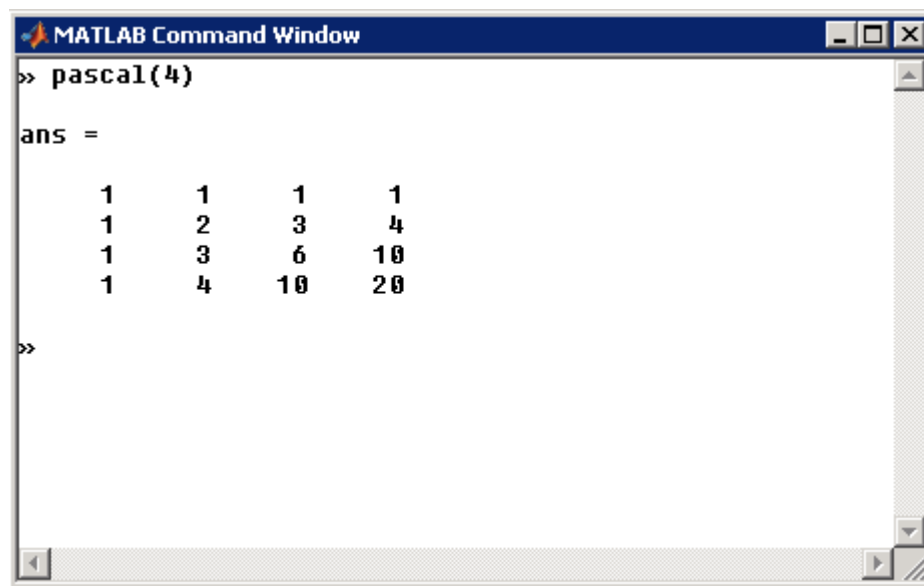
>> sign(-5)

ans =

    -1

>> |
```

For highschool students you can get the Pascal triangle with the function **pascal()** for fun...:

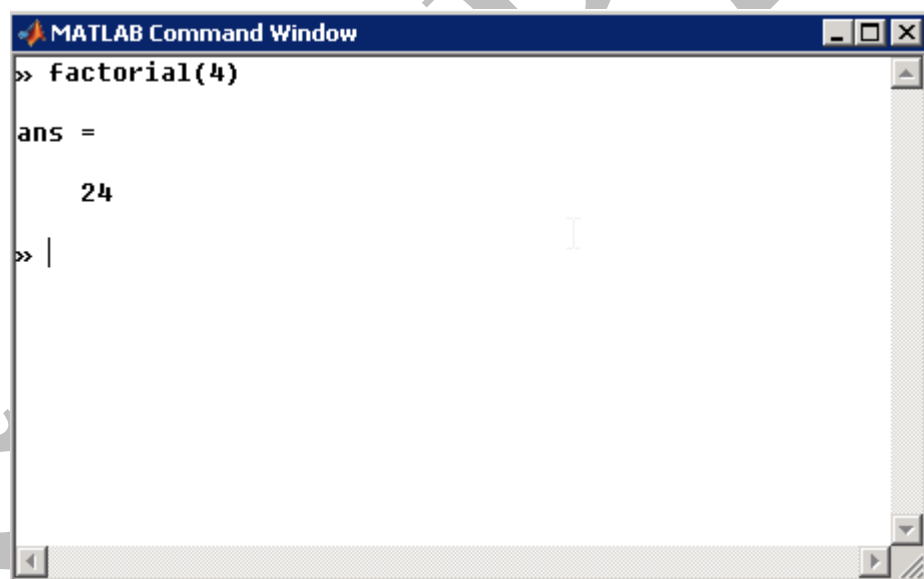


```
MATLAB Command Window
>> pascal(4)

ans =

     1     1     1     1
     1     2     3     4
     1     3     6    10
     1     4    10    20
```

And our tradition **factorial()** function very useful in statistics, number theory and quantum physics:

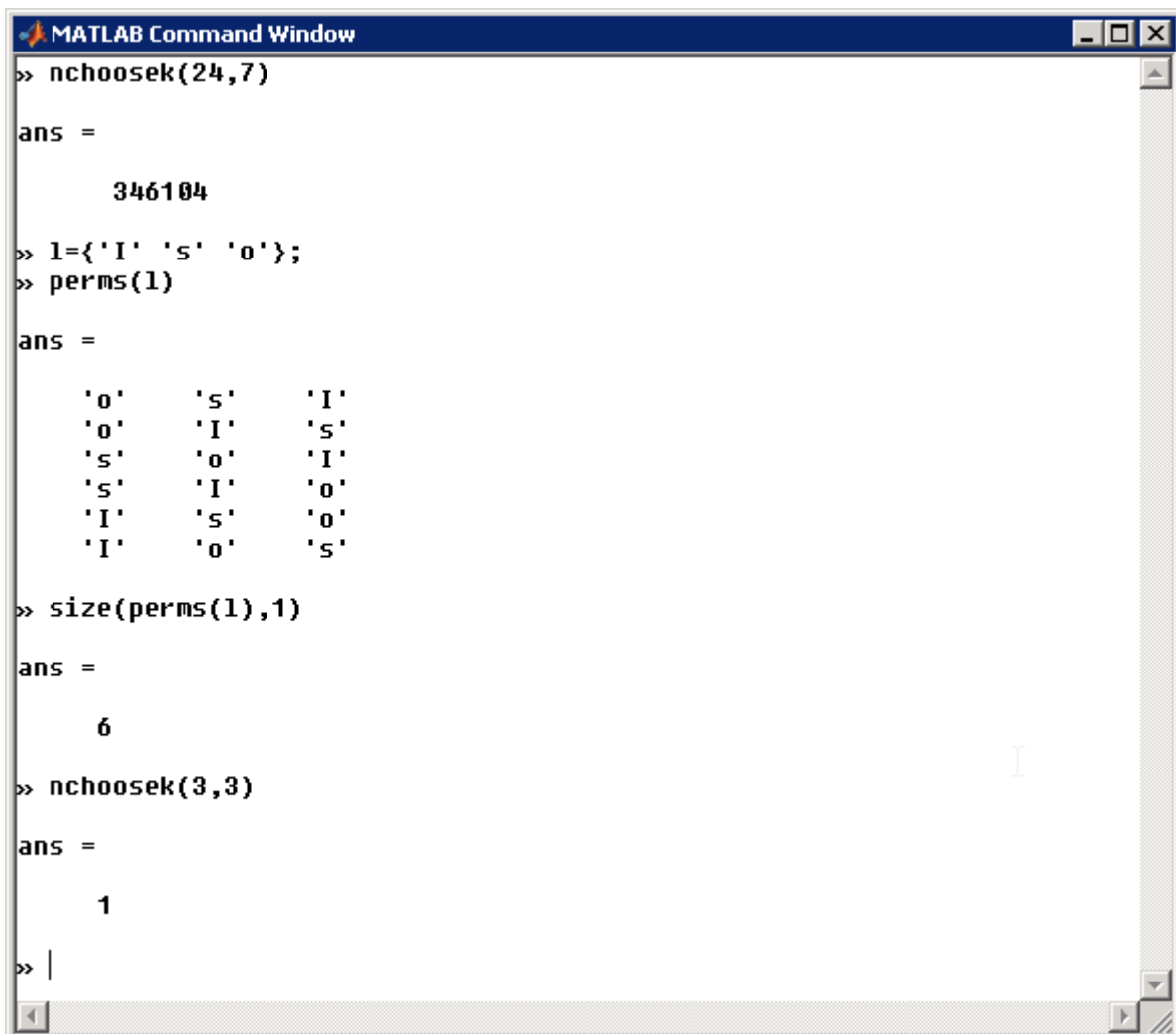


```
MATLAB Command Window
>> factorial(4)

ans =

    24
```

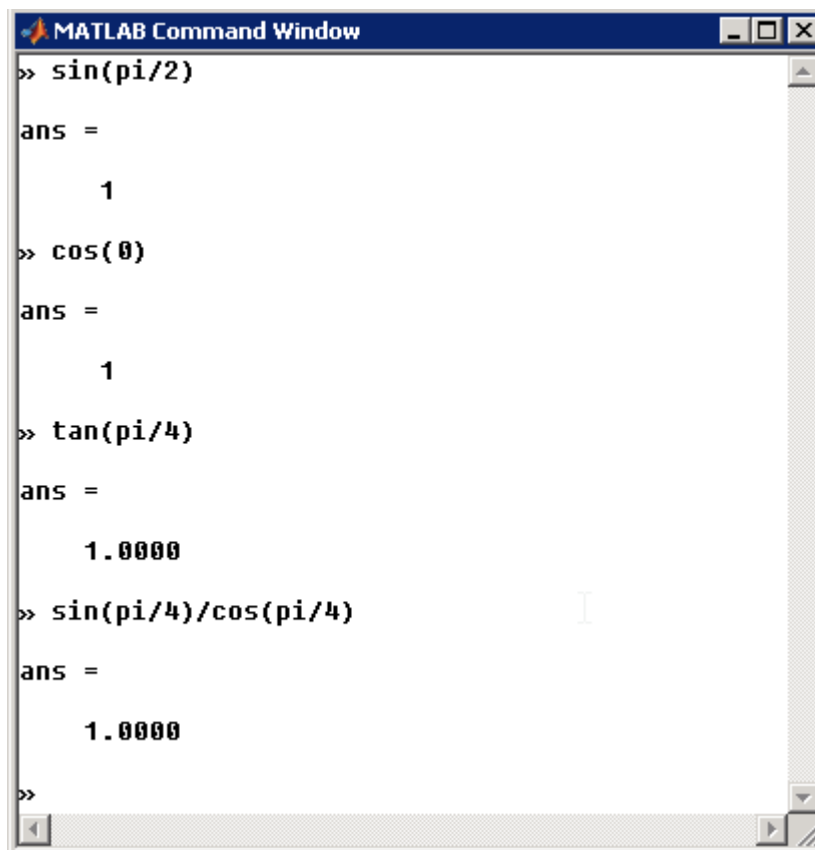
Another important function in statistics is the combinatorics that you can get with **nchoosek()** and for high school students they can play with also with the **perms()** function to see all possible permutations using the sample examples as in the theoretical course:

A screenshot of the MATLAB Command Window. The window title is "MATLAB Command Window". The command history shows the following:
1. Command: `nchoosek(24,7)`
Output: `ans = 346104`
2. Command: `l={'I' 'S' 'O'};`
3. Command: `perms(l)`
Output: `ans =` followed by a 6x3 matrix of characters:

'O'	'S'	'I'
'O'	'I'	'S'
'S'	'O'	'I'
'S'	'I'	'O'
'I'	'S'	'O'
'I'	'O'	'S'

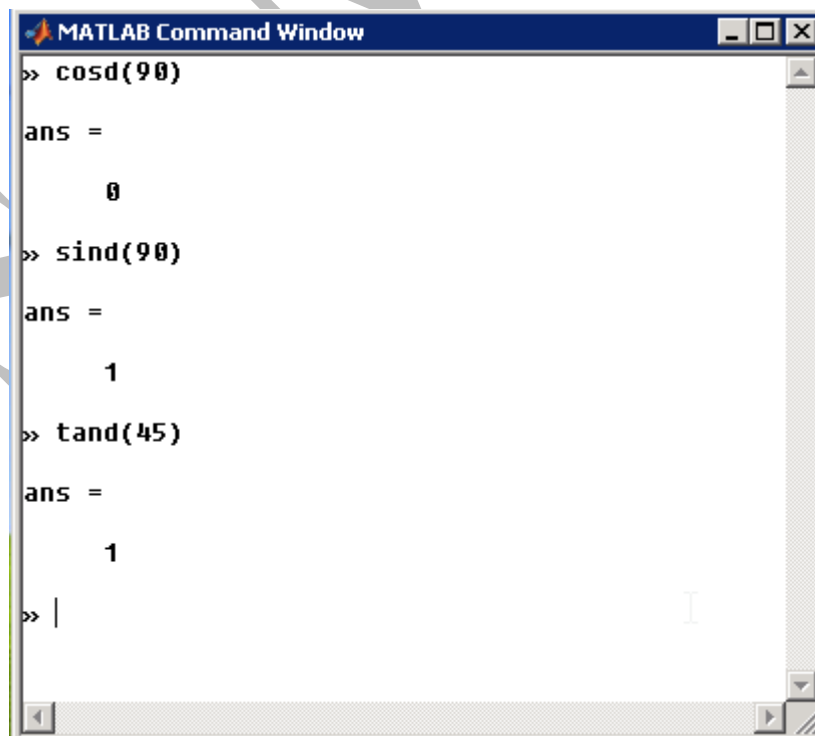
4. Command: `size(perms(l),1)`
Output: `ans = 6`
5. Command: `nchoosek(3,3)`
Output: `ans = 1`
The prompt `>> |` is visible at the bottom.

Now some trigonometry based on radians with the functions **sin()**, **cos()**, **tan()**:



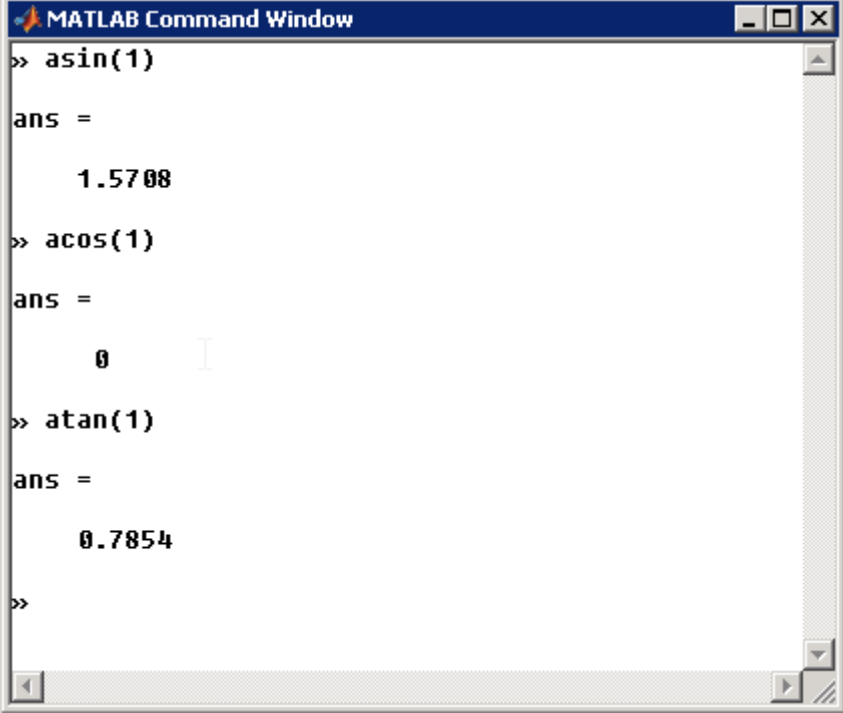
```
MATLAB Command Window
>> sin(pi/2)
ans =
    1
>> cos(0)
ans =
    1
>> tan(pi/4)
ans =
    1.0000
>> sin(pi/4)/cos(pi/4)
ans =
    1.0000
>>
```

For people that like to work in degrees instead of radians you have the functions **cosd()**, **sind()** and **tand()** that are useful when you work with latitudes and longitudes:



```
MATLAB Command Window
>> cosd(90)
ans =
    0
>> sind(90)
ans =
    1
>> tand(45)
ans =
    1
>> |
```

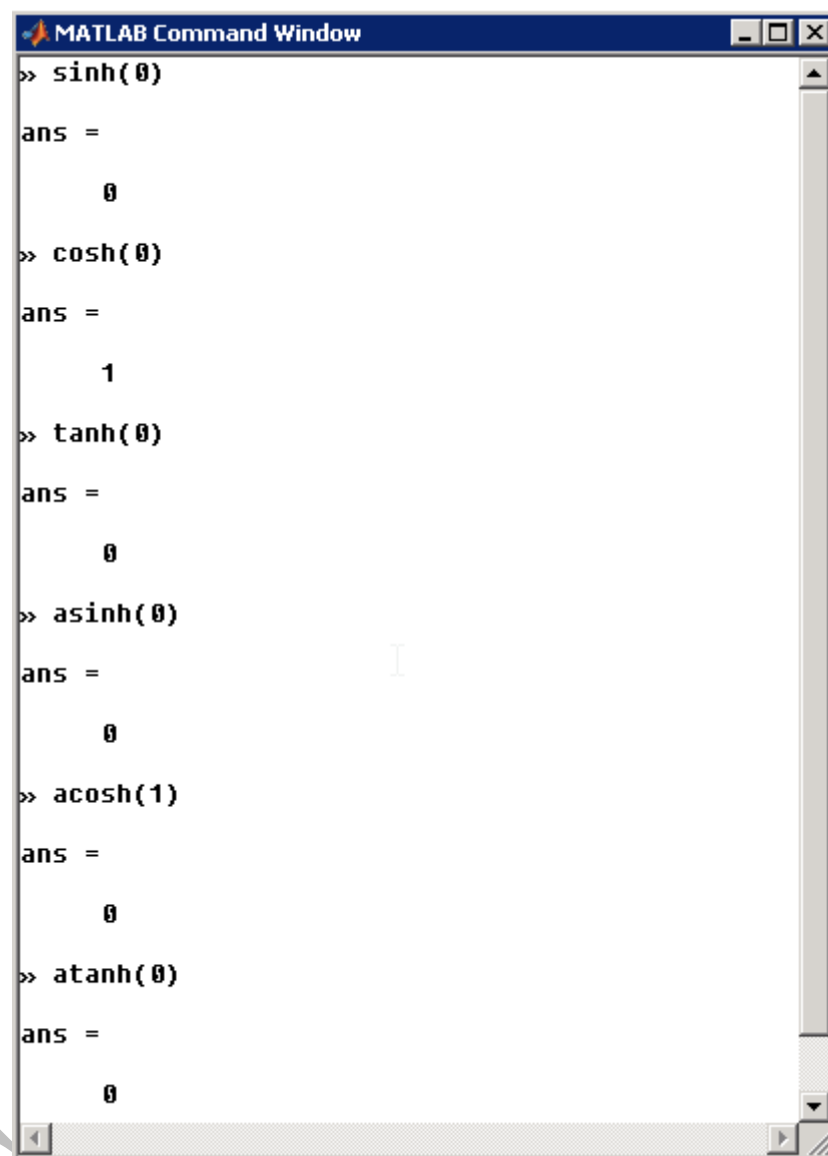
let us continue in radians with some inverse trigonometric functions in radians like **asin()**, **acos()** or **atan()**:



```
MATLAB Command Window
>> asin(1)
ans =
    1.5708
>> acos(1)
ans =
     0
>> atan(1)
ans =
    0.7854
>>
```

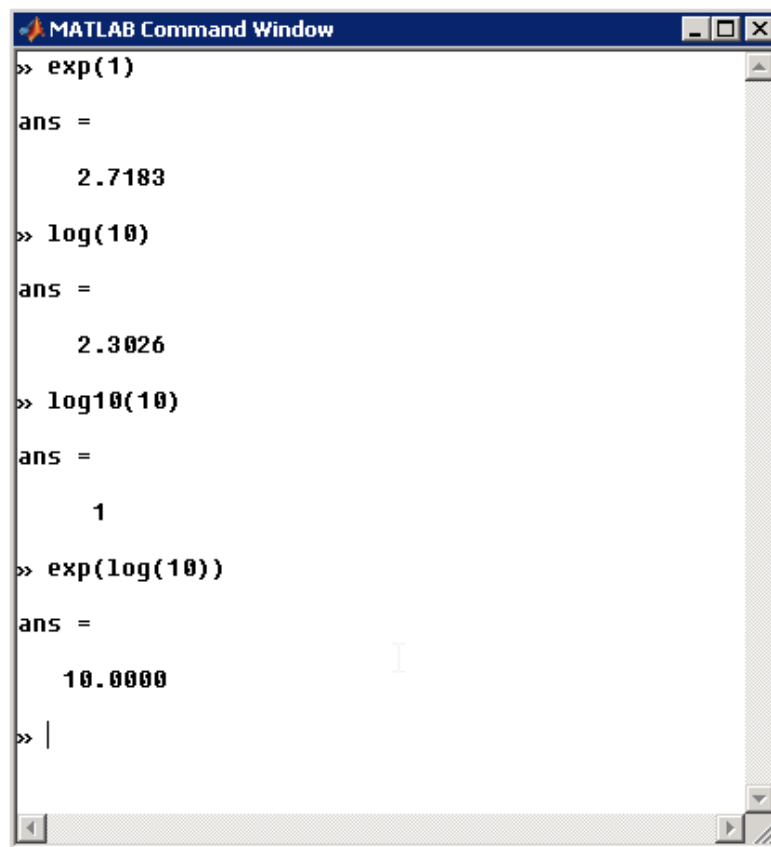
A screenshot of the MATLAB Command Window. The window has a blue title bar with the MATLAB logo and the text "MATLAB Command Window". The command history shows three commands: `asin(1)`, `acos(1)`, and `atan(1)`. The results are displayed as `ans = 1.5708`, `ans = 0`, and `ans = 0.7854` respectively. The cursor is at the end of the last command line.

And for sure we also have the hyperbolic normal and inverse trigonometric functions **sinh()**, **cosh()**, **tanh()**, **asinh()**, **acosh()**, **atanh()**:



```
MATLAB Command Window
>> sinh(0)
ans =
    0
>> cosh(0)
ans =
    1
>> tanh(0)
ans =
    0
>> asinh(0)
ans =
    0
>> acosh(1)
ans =
    0
>> atanh(0)
ans =
    0
```

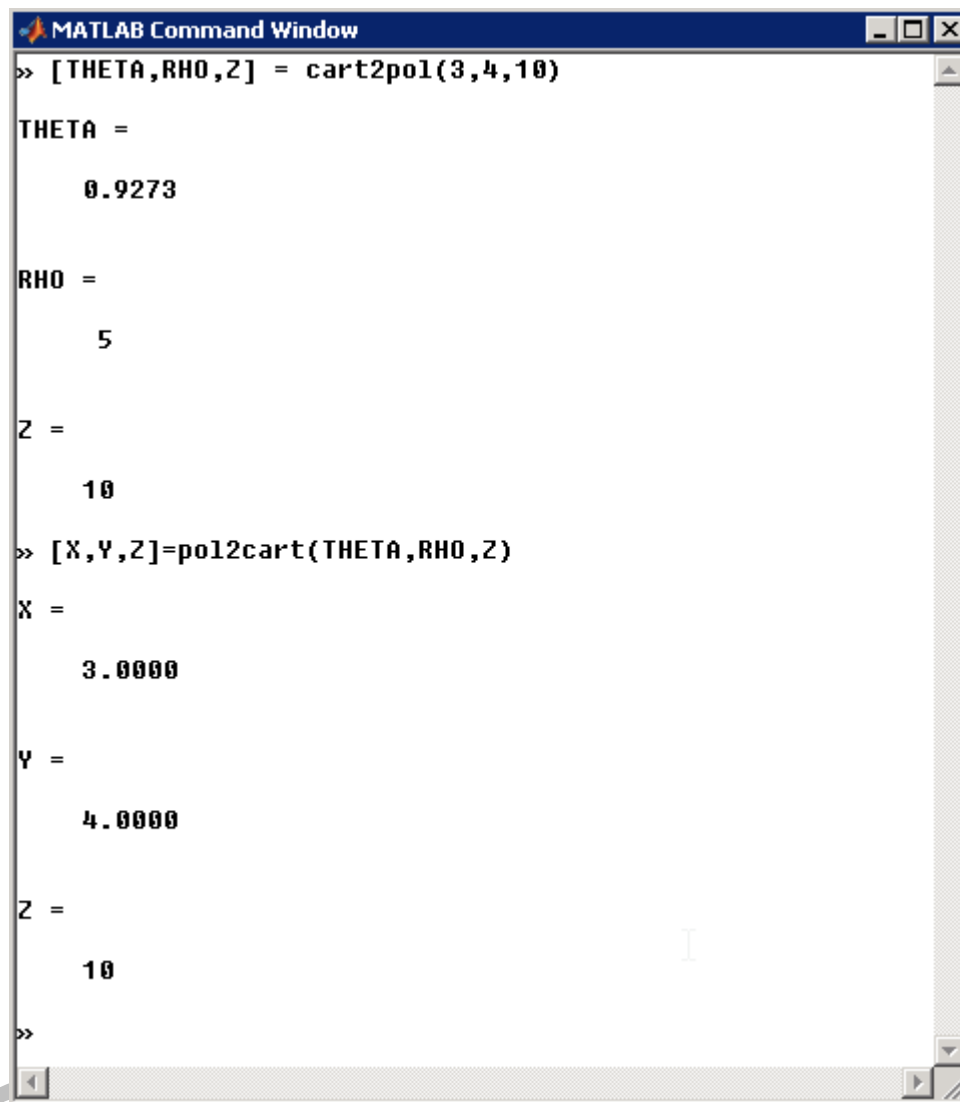

Some other very classic and useful mathematical functions are `exp( )` for the exponential, `log( )` for the natural logarithm, `log10( )` for the logarithm in base 10:



```
MATLAB Command Window
>> exp(1)
ans =
    2.7183
>> log(10)
ans =
    2.3026
>> log10(10)
ans =
    1
>> exp(log(10))
ans =
   10.0000
>> |
```

The image shows a MATLAB Command Window with a blue title bar. It contains a series of commands and their outputs. The commands are `exp(1)`, `log(10)`, `log10(10)`, and `exp(log(10))`. The outputs are `ans = 2.7183`, `ans = 2.3026`, `ans = 1`, and `ans = 10.0000` respectively. The window has standard Windows-style window controls (minimize, maximize, close) in the top right corner and scrollbars on the right and bottom.

Sometimes it is useful to some charts to know how to transform cartesian coordinates into polar coordinates. This can easily be done thanks to the function **cart2pol()** or also the inverse by using the function **pol2cart()**:



```
MATLAB Command Window
>> [THETA,RHO,Z] = cart2pol(3,4,10)

THETA =

    0.9273

RHO =

     5

Z =

    10

>> [X,Y,Z]=pol2cart(THETA,RHO,Z)

X =

    3.0000

Y =

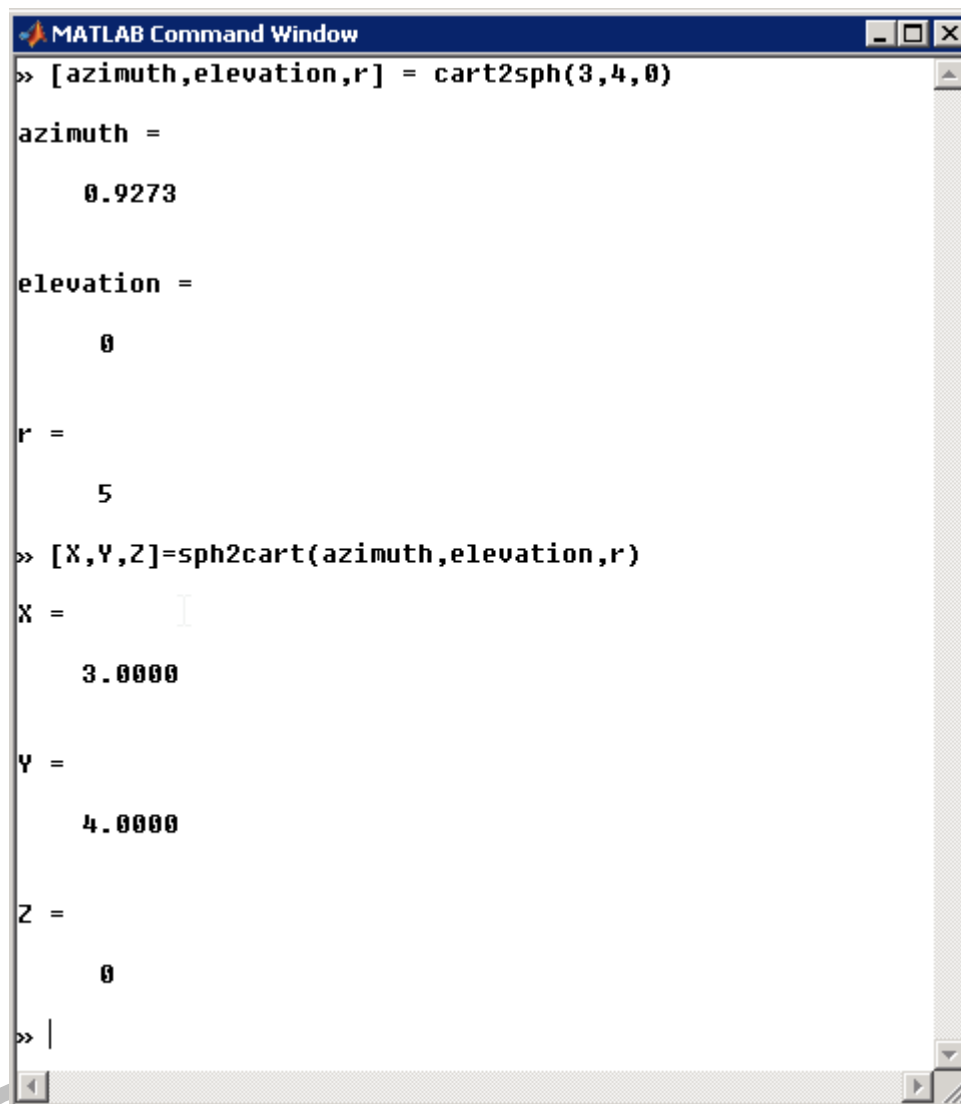
    4.0000

Z =

    10

>>
```

In astronomy, or celestial mechanics and also in atomistic in can be useful to know how to convert cartesian coordinates to spherical coordinates using the function **cart2sph()** or the inverse operation using **sph2cart()**:



```

MATLAB Command Window
>> [azimuth,elevation,r] = cart2sph(3,4,0)

azimuth =

    0.9273

elevation =

    0

r =

    5

>> [X,Y,Z]=sph2cart(azimuth,elevation,r)

X =

    3.0000

Y =

    4.0000

Z =

    0

>> |
  
```

Now let us see a famous example using complex number and trigonometric function by plotting one of the 3d orbital of the hydrogenoid atom:

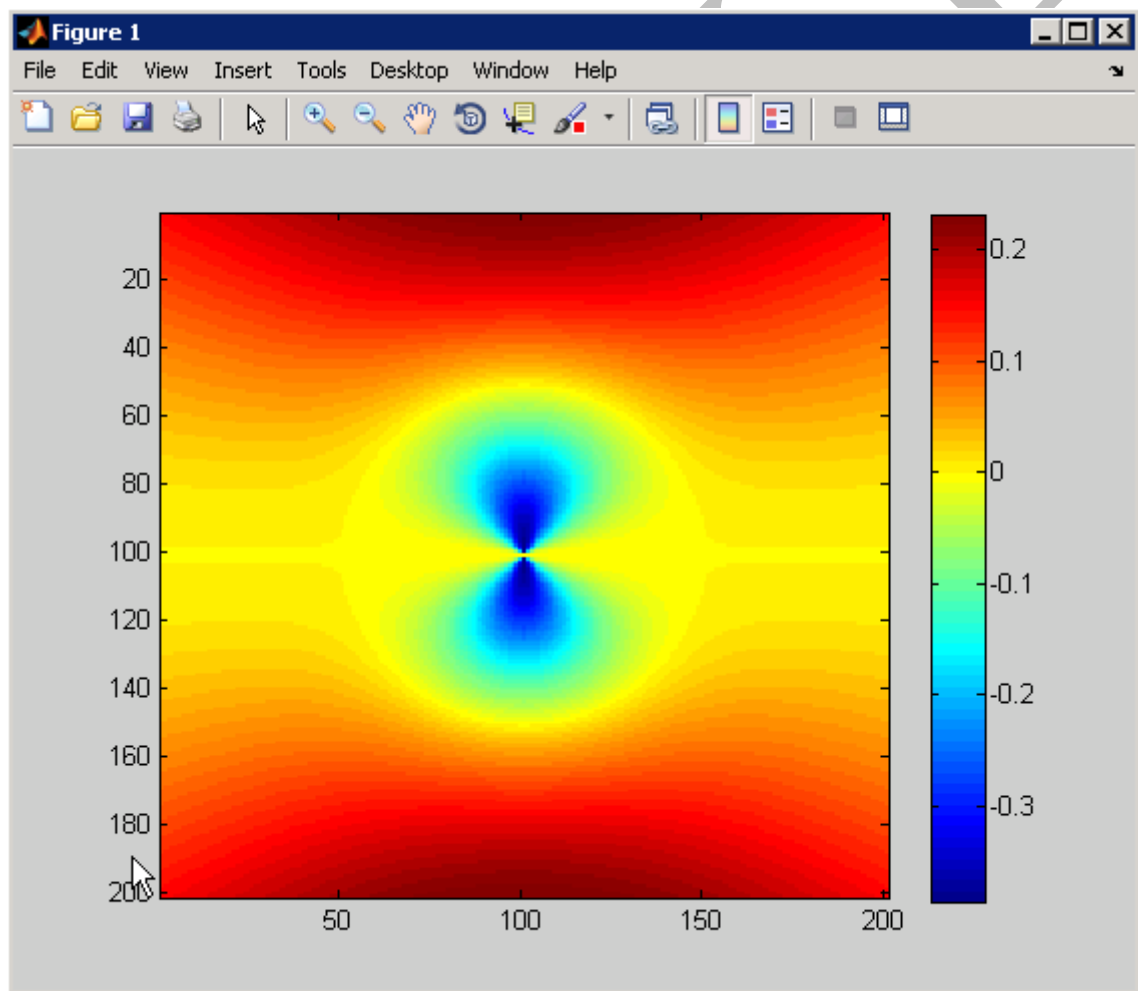
$$Y_{2,2} = \sqrt{\frac{15}{32\pi}} \sin^2 \theta e^{2i\phi}$$

```

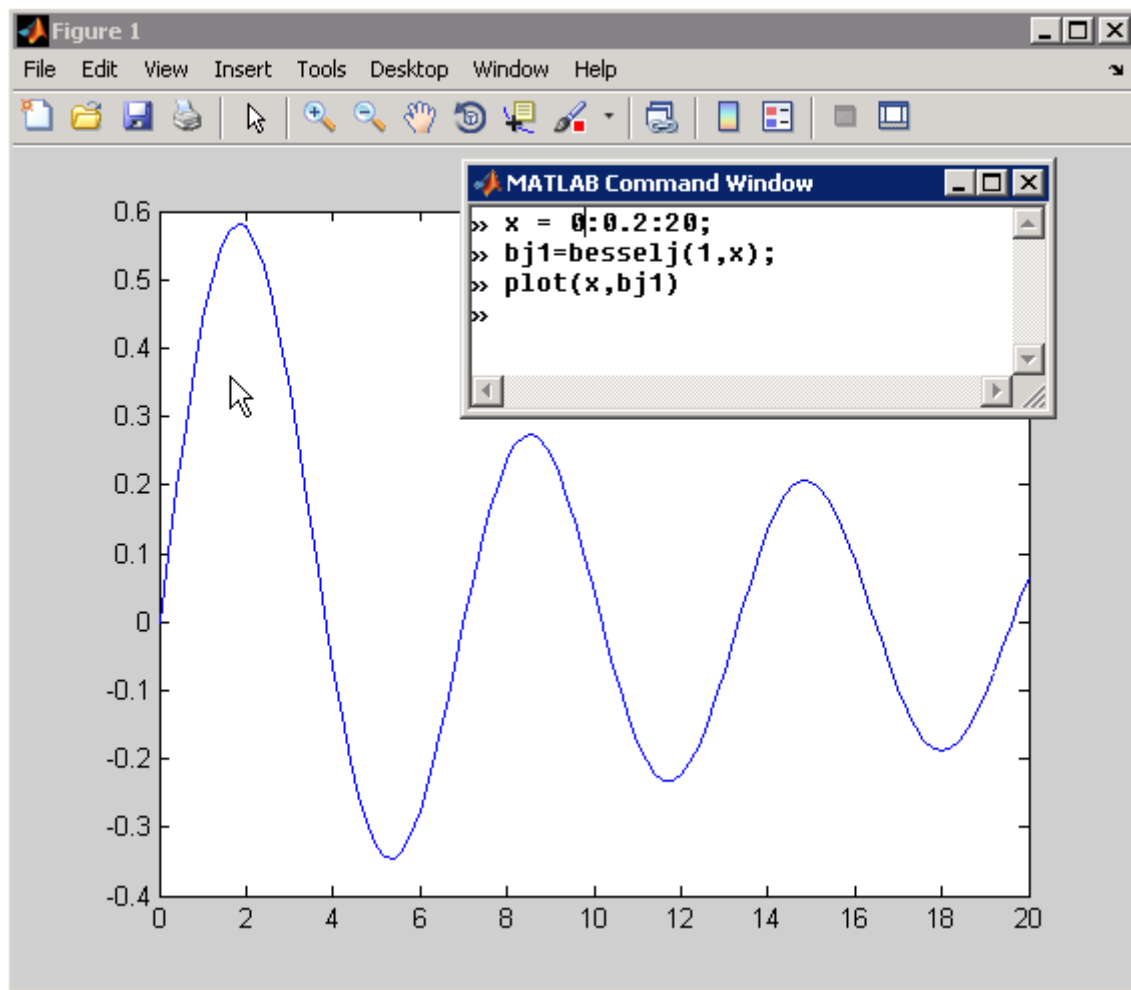
EDITOR PUBLISH VIEW
1 %probability densities
2 - pd_psi_210 = @(r,theta,phi) real(exp(2*i*phi)).*(sin(theta).^2)*sqrt(15/(32*pi));
3
4 %configuring the range
5 - [x, y, z] = meshgrid(-1:.01:1,-1:.01:1,-0.5:.01:0.5);
6 - [theta,phi,r] = cart2sph(x,y,z);
7
8 %create array of probability density magnitudes
9 - psi_210_vals = pd_psi_210(r,theta,phi);
10
11 %plotting
12 - imagesc(psi_210_vals(:,:,1)); %x-y plane
13 - colorbar

```

That gives:



We have seen in the Optic course and Nuclear course that the Bessel J function was very useful. Let us take a look if it's available in MATLAB using the function **besselj()**:

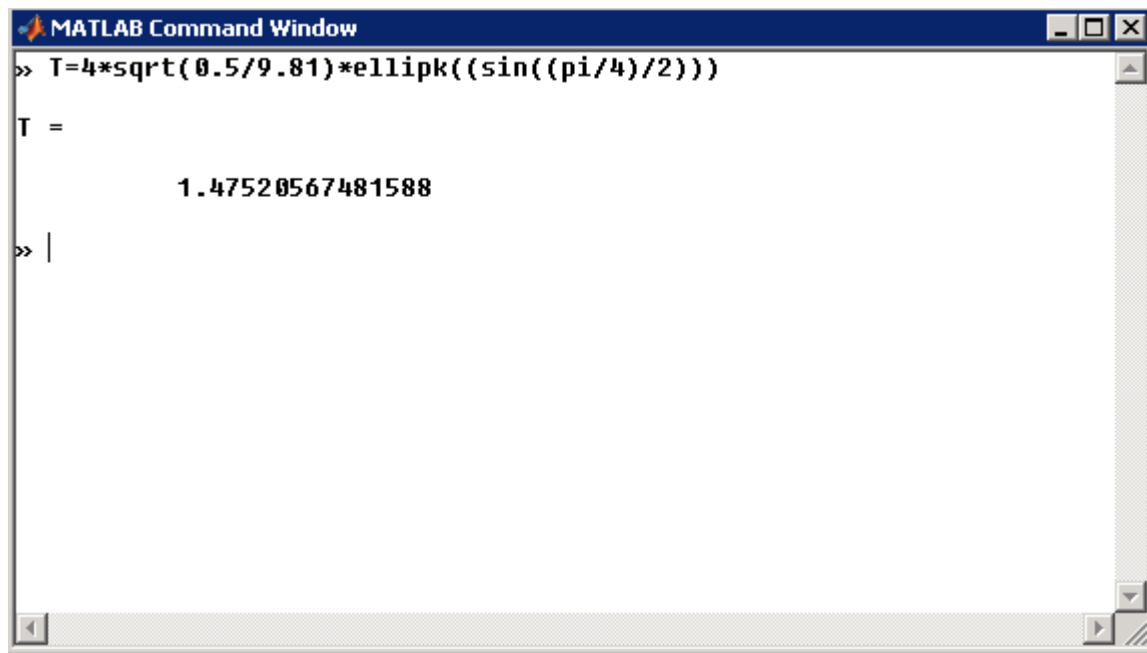


We have also seen in the Statistics course and Thermodynamics course a function named "erf". Let us see if this function is really available in MATLAB using the **erf()** function:

The image shows the 'MATLAB Command Window' with the following interaction:

```
>> erf(Inf)/2  
  
ans =  
  
    0.5000  
  
>> |
```

During our study of nuclear magnetic resonance bird cage we have seen that K elliptic functions were useful. Let us have a look with a typical academic example base on the pendulum using the function **ellipk()**:



```

MATLAB Command Window
>> T=4*sqrt(0.5/9.81)*ellipk((sin((pi/4)/2)))

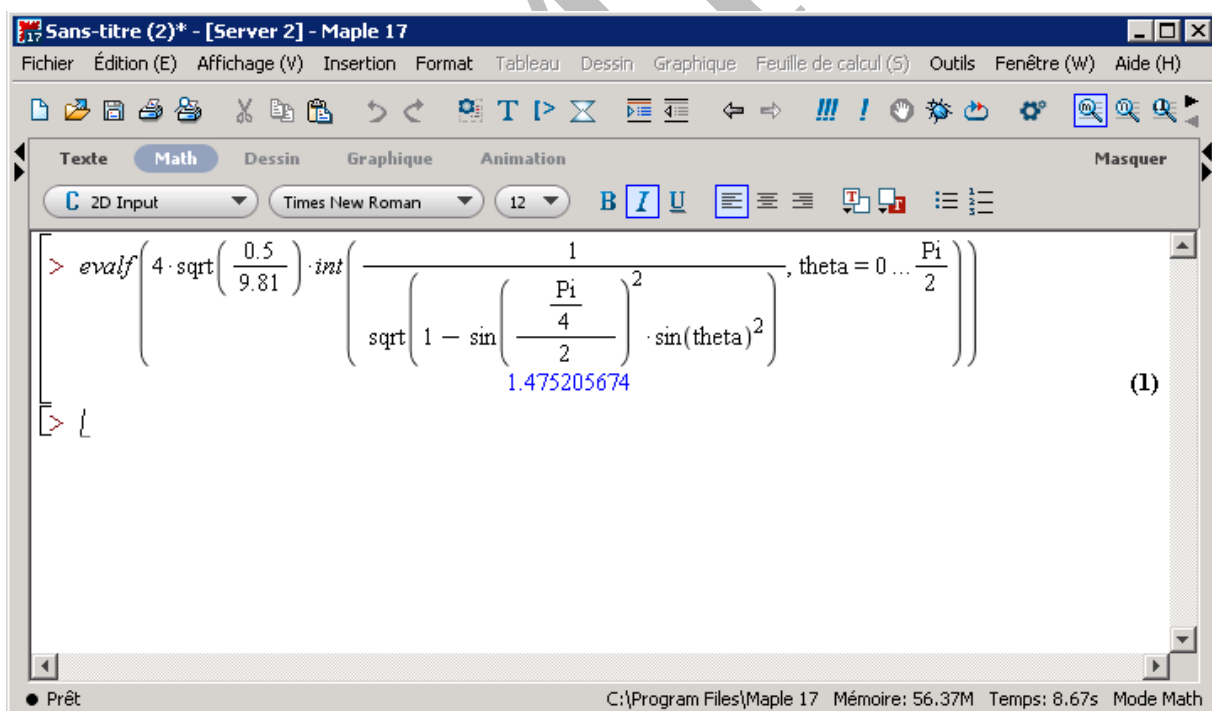
T =

    1.47520567481588

>> |

```

Is the result very accurate? Let us check that with Maple 17:



```

Sans-titre (2)* - [Server 2] - Maple 17
Fichier Édition (E) Affichage (V) Insertion Format Tableau Dessin Graphique Feuille de calcul (S) Outils Fenêtre (W) Aide (H)

[Icons]

Texte Math Dessin Graphique Animation Masquer
2D Input Times New Roman 12 B I U [Icons]

> evalf(4*sqrt(0.5/9.81)*int(1/sqrt(1-sin(Pi/4/2)^2*sin(theta)^2), theta = 0 ... Pi/2))

1.475205674 (1)

> |

```

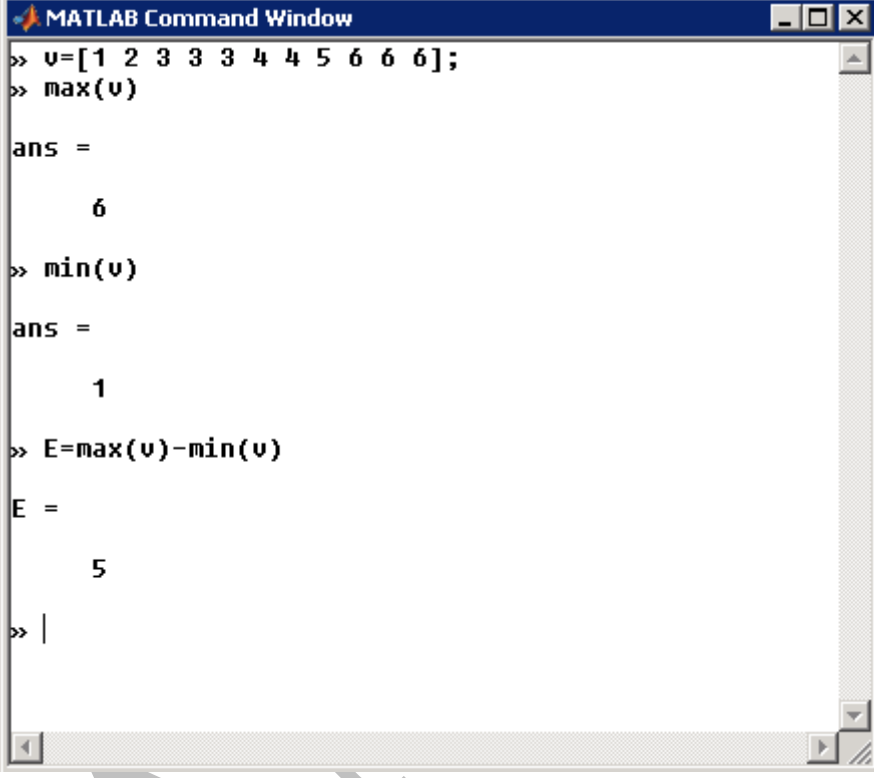
● Prêt C:\Program Files\Maple 17 Mémoire: 56.37M Temps: 8.67s Mode Math

Core (elementary) statistical functions

MATLAB is very limited in native advanced statistical functions if you do not have additional Toolbox like the Statistical Toolbox, Econometrics Toolbox or Forecasting Toolbox.

The purpose here is to see what is available in the MATLAB core version.

First take a look the **max()**, **min()** function and calculate also the range of a vector:



```
MATLAB Command Window
>> v=[1 2 3 3 3 4 4 5 6 6 6];
>> max(v)

ans =

     6

>> min(v)

ans =

     1

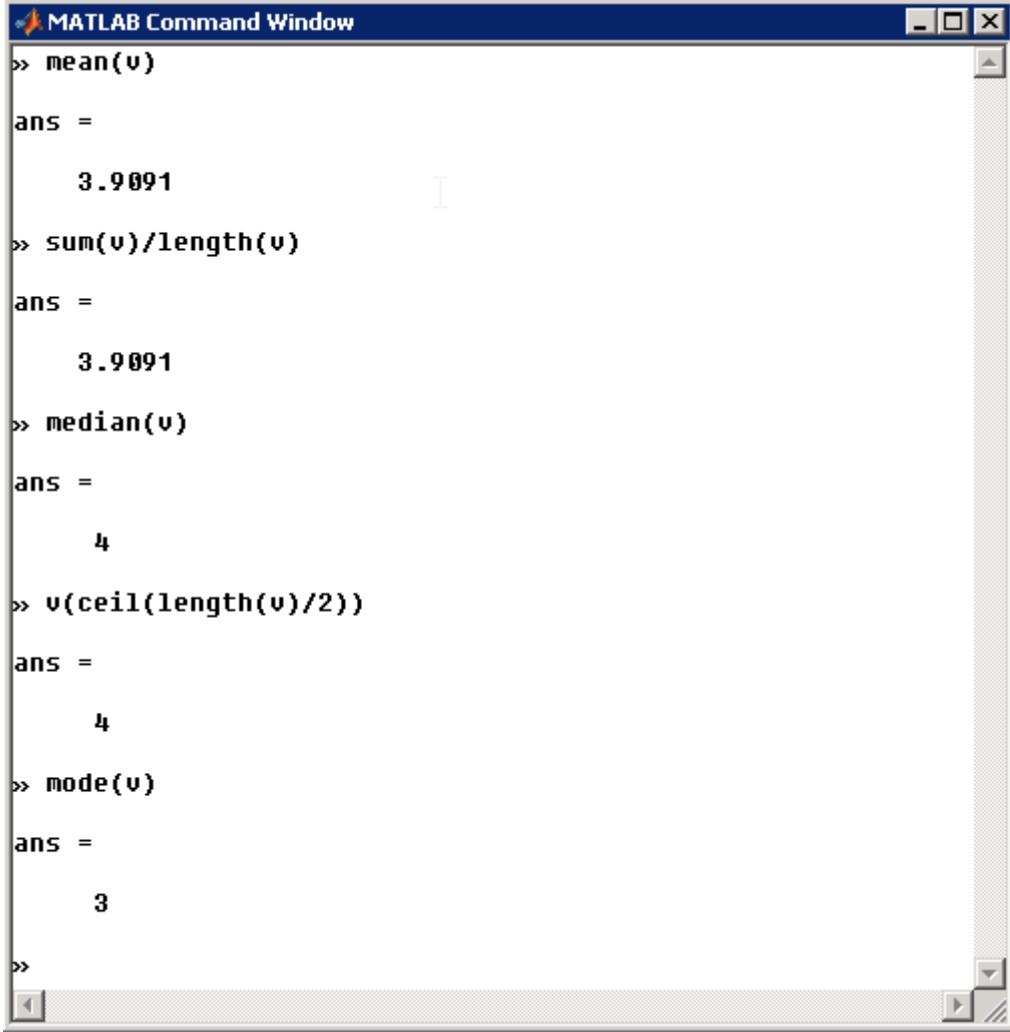
>> E=max(v)-min(v)

E =

     5

>> |
```

Now let us see if the arithmetical mean is available and correct with the **mean()** function... and also the median with the **median()** function and the mode with the function **mode()**:

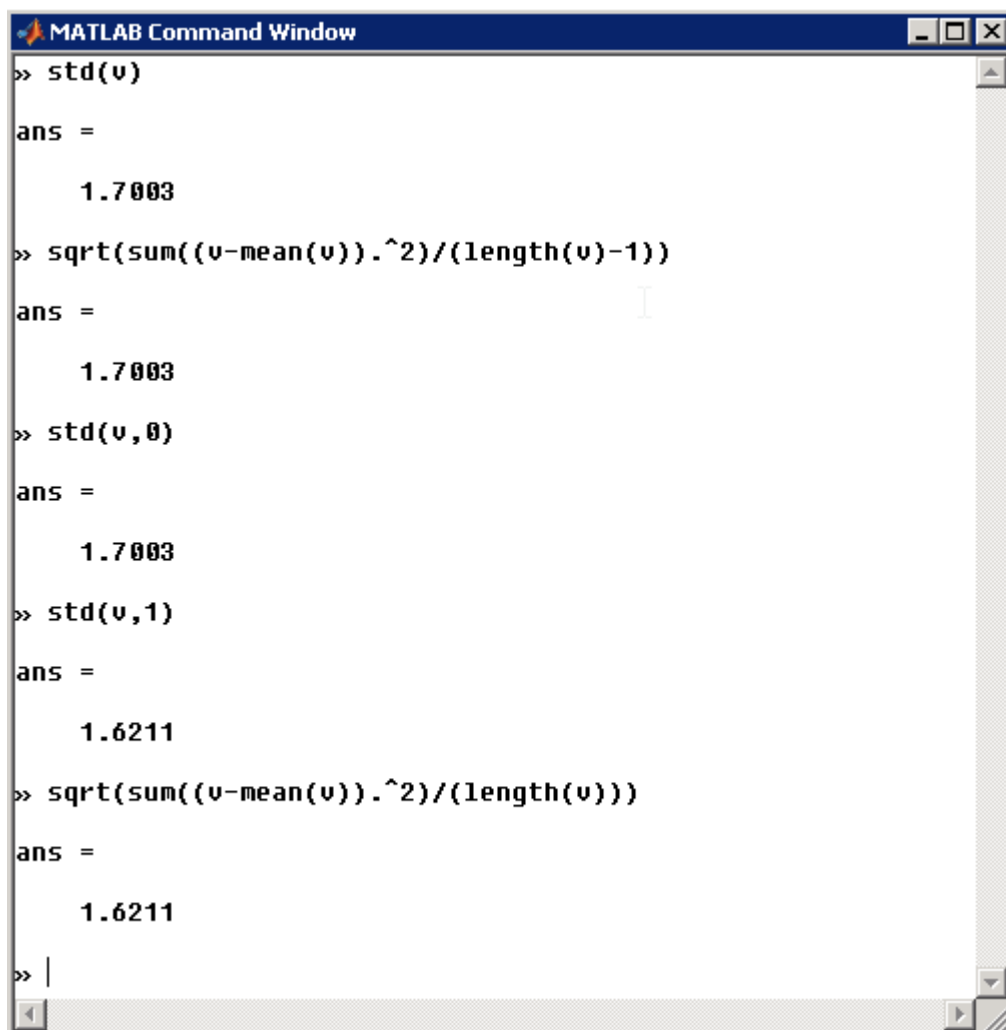


```
MATLAB Command Window
>> mean(v)
ans =
    3.9091
>> sum(v)/length(v)
ans =
    3.9091
>> median(v)
ans =
     4
>> v(ceil(length(v)/2))
ans =
     4
>> mode(v)
ans =
     3
>>
```

The screenshot shows the MATLAB Command Window with the following commands and outputs:

- `mean(v)` returns `ans = 3.9091`.
- `sum(v)/length(v)` returns `ans = 3.9091`.
- `median(v)` returns `ans = 4`.
- `v(ceil(length(v)/2))` returns `ans = 4`.
- `mode(v)` returns `ans = 3`.

We must have a look the standard deviation and be sure that MATLAB use by default the non biased estimator of the standard deviation with the function **std()** and that we can choose the type of standard deviation we want (biased are non-biased):

A screenshot of the MATLAB Command Window. The window has a blue title bar with the MATLAB logo and the text "MATLAB Command Window". The command history shows several commands and their outputs. The first command is `>> std(v)`, which returns `ans = 1.7003`. The second command is `>> sqrt(sum((v-mean(v)).^2)/(length(v)-1))`, which also returns `ans = 1.7003`. The third command is `>> std(v,0)`, which returns `ans = 1.7003`. The fourth command is `>> std(v,1)`, which returns `ans = 1.6211`. The fifth command is `>> sqrt(sum((v-mean(v)).^2)/(length(v)))`, which also returns `ans = 1.6211`. The cursor is at the end of the last command line.

```
>> std(v)
ans =
    1.7003

>> sqrt(sum((v-mean(v)).^2)/(length(v)-1))
ans =
    1.7003

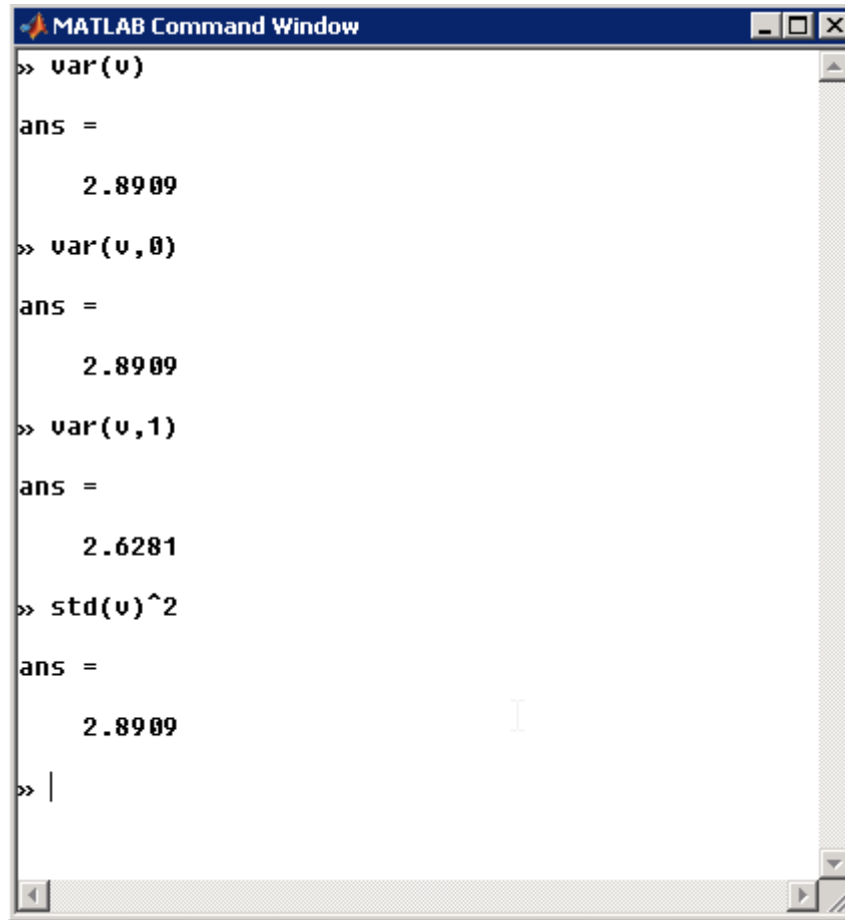
>> std(v,0)
ans =
    1.7003

>> std(v,1)
ans =
    1.6211

>> sqrt(sum((v-mean(v)).^2)/(length(v)))
ans =
    1.6211

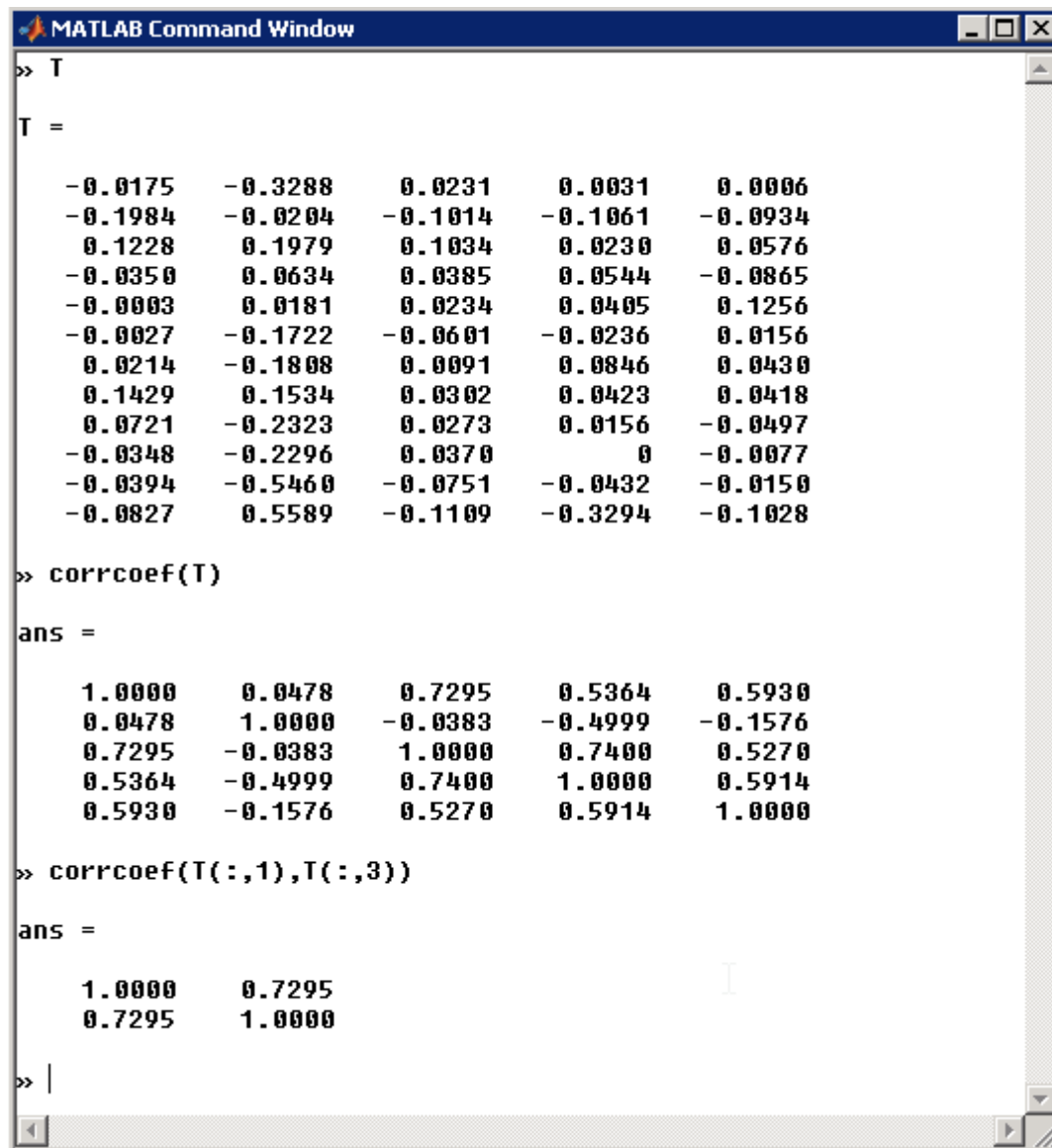
>> |
```

We can also check that the variance have the same flag with the function **var()** to have the non-biased and the biased version:

A screenshot of the MATLAB Command Window. The window has a blue title bar with the MATLAB logo and the text "MATLAB Command Window". The command history shows four commands and their outputs. The first command is `var(v)`, which returns `ans = 2.8909`. The second command is `var(v,0)`, which also returns `ans = 2.8909`. The third command is `var(v,1)`, which returns `ans = 2.6281`. The fourth command is `std(v)^2`, which returns `ans = 2.8909`. The prompt `>> |` is visible at the bottom of the command history.

```
MATLAB Command Window
>> var(v)
ans =
    2.8909
>> var(v,0)
ans =
    2.8909
>> var(v,1)
ans =
    2.6281
>> std(v)^2
ans =
    2.8909
>> |
```

The correlation matrix is by default the non biased correlation matrix when you use the function **corrcoef()** on a whole matrix or on one of its subset:



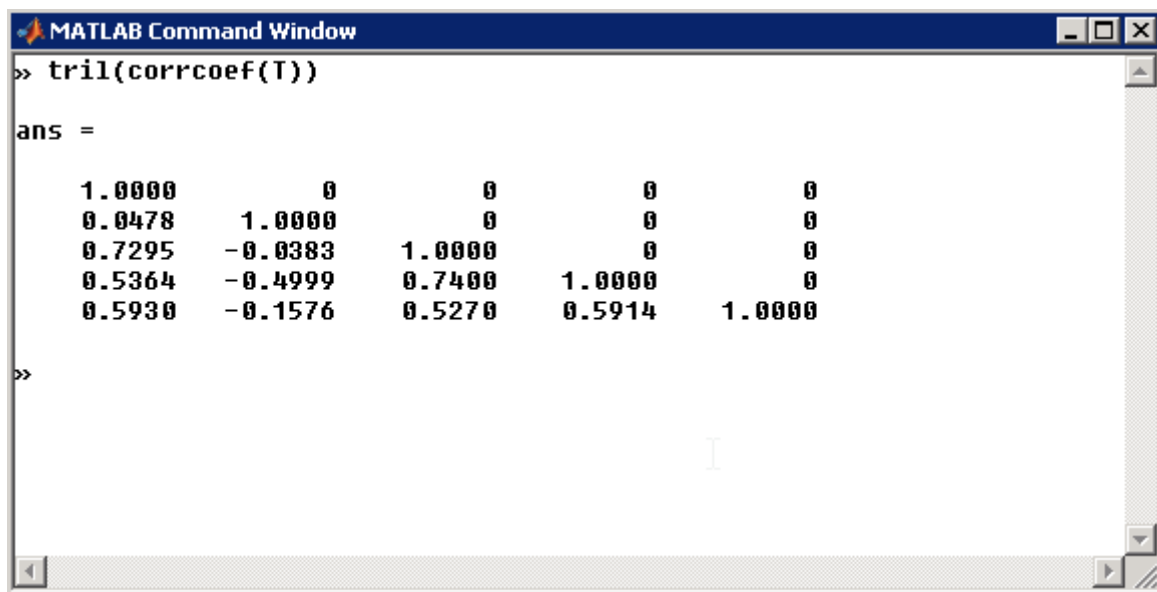
```

MATLAB Command Window
>> T
T =
    -0.0175    -0.3288     0.0231     0.0031     0.0006
    -0.1984    -0.0204    -0.1014    -0.1061    -0.0934
     0.1228     0.1979     0.1034     0.0230     0.0576
    -0.0350     0.0634     0.0385     0.0544    -0.0865
    -0.0003     0.0181     0.0234     0.0405     0.1256
    -0.0027    -0.1722    -0.0601    -0.0236     0.0156
     0.0214    -0.1808     0.0091     0.0846     0.0430
     0.1429     0.1534     0.0302     0.0423     0.0418
     0.0721    -0.2323     0.0273     0.0156    -0.0497
    -0.0348    -0.2296     0.0370         0    -0.0077
    -0.0394    -0.5460    -0.0751    -0.0432    -0.0150
    -0.0827     0.5589    -0.1109    -0.3294    -0.1028

>> corrcoef(T)
ans =
    1.0000     0.0478     0.7295     0.5364     0.5930
     0.0478     1.0000    -0.0383    -0.4999    -0.1576
     0.7295    -0.0383     1.0000     0.7400     0.5270
     0.5364    -0.4999     0.7400     1.0000     0.5914
     0.5930    -0.1576     0.5270     0.5914     1.0000

>> corrcoef(T(:,1),T(:,3))
ans =
    1.0000     0.7295
     0.7295     1.0000
  
```

For sure we can apply the **tril()** function together with the **corrcoef()** function if we want:



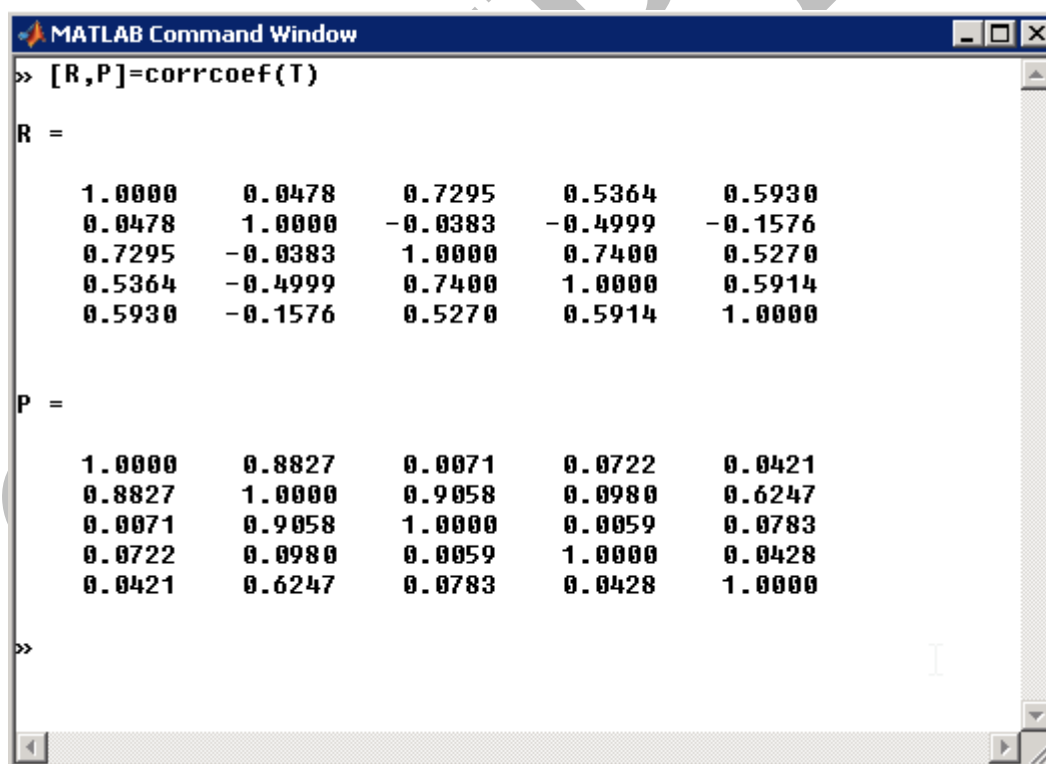
```

>> tril(corrcoef(T))

ans =

    1.0000         0         0         0         0
    0.0478    1.0000         0         0         0
    0.7295   -0.0383    1.0000         0         0
    0.5364   -0.4999    0.7400    1.0000         0
    0.5930   -0.1576    0.5270    0.5914    1.0000
  
```

Or by extracting the second output of the function **corrcoef()** we can get the p-value of the Pearson test with by default a 5% threshold:



```

>> [R,P]=corrcoef(T)

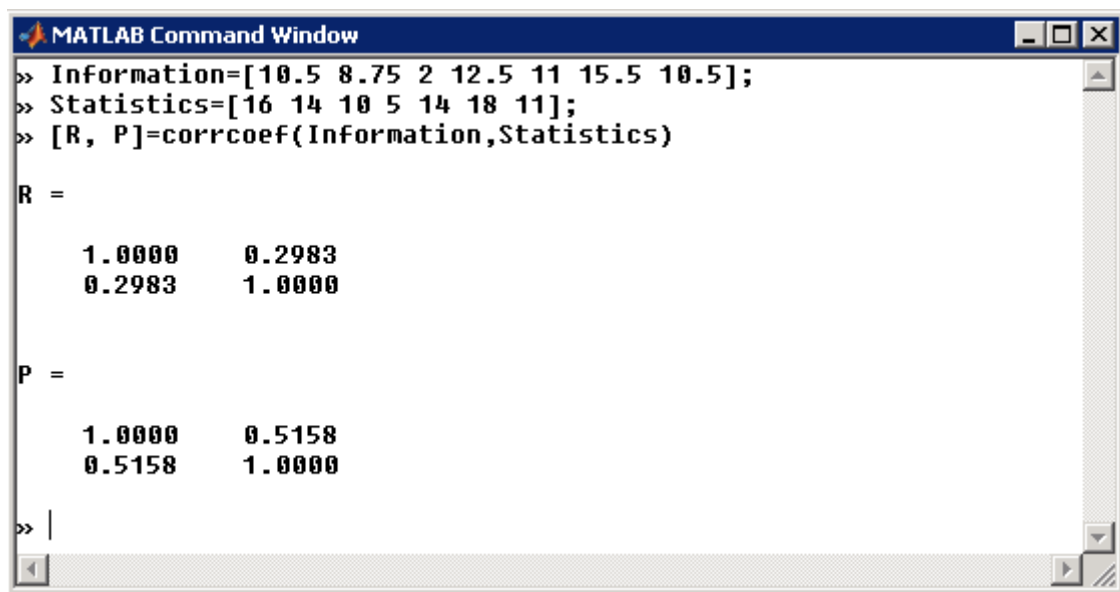
R =

    1.0000    0.0478    0.7295    0.5364    0.5930
    0.0478    1.0000   -0.0383   -0.4999   -0.1576
    0.7295   -0.0383    1.0000    0.7400    0.5270
    0.5364   -0.4999    0.7400    1.0000    0.5914
    0.5930   -0.1576    0.5270    0.5914    1.0000

P =

    1.0000    0.8827    0.0071    0.0722    0.0421
    0.8827    1.0000    0.9058    0.0980    0.6247
    0.0071    0.9058    1.0000    0.0059    0.0783
    0.0722    0.0980    0.0059    1.0000    0.0428
    0.0421    0.6247    0.0783    0.0428    1.0000
  
```

Let us check if the result corresponds to calculation made by hand in the Statistics course:

A screenshot of the MATLAB Command Window. The window title is "MATLAB Command Window". The command prompt shows the following code:

```
>> Information=[10.5 8.75 2 12.5 11 15.5 10.5];  
>> Statistics=[16 14 10 5 14 18 11];  
>> [R, P]=corrcoef(Information,Statistics)
```

 The output shows two matrices, R and P, each displayed as a 2x2 matrix. Matrix R is:

1.0000	0.2983
0.2983	1.0000

 Matrix P is:

1.0000	0.5158
0.5158	1.0000

 The command prompt ends with

```
>> |
```

```
MATLAB Command Window  
>> Information=[10.5 8.75 2 12.5 11 15.5 10.5];  
>> Statistics=[16 14 10 5 14 18 11];  
>> [R, P]=corrcoef(Information,Statistics)  
  
R =  
  
    1.0000    0.2983  
    0.2983    1.0000  
  
P =  
  
    1.0000    0.5158  
    0.5158    1.0000  
  
>> |
```

Yep! We get it!

We can also have the lower and upper limits of the 95% confidence interval:

```

>> [R,P,RLO,RUP]=corrcoef(T)

R =
    1.0000    0.0478    0.7295    0.5364    0.5930
    0.0478    1.0000   -0.0383   -0.4999   -0.1576
    0.7295   -0.0383    1.0000    0.7400    0.5270
    0.5364   -0.4999    0.7400    1.0000    0.5914
    0.5930   -0.1576    0.5270    0.5914    1.0000

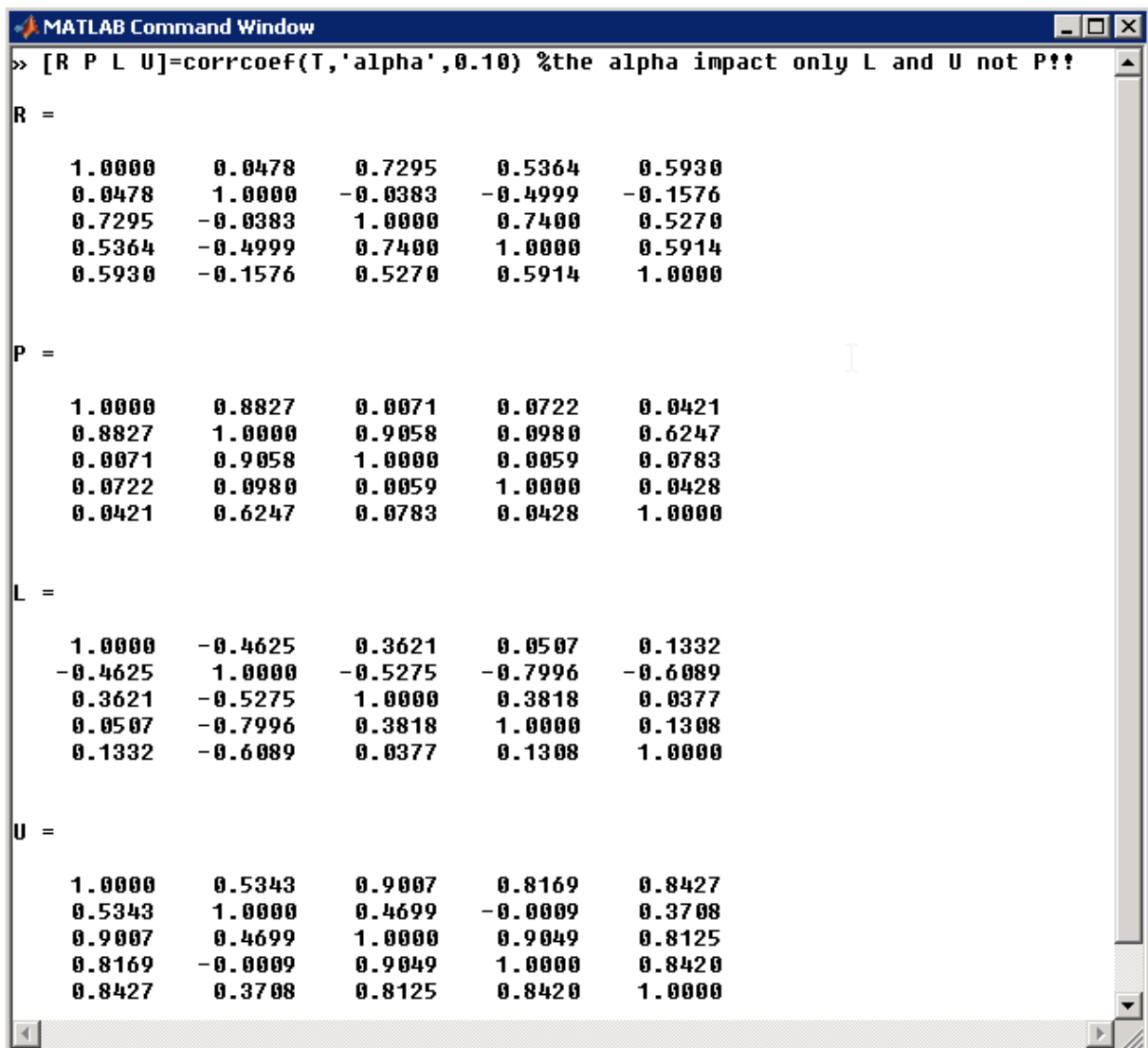
P =
    1.0000    0.8827    0.0071    0.0722    0.0421
    0.8827    1.0000    0.9058    0.0980    0.6247
    0.0071    0.9058    1.0000    0.0059    0.0783
    0.0722    0.0980    0.0059    1.0000    0.0428
    0.0421    0.6247    0.0783    0.0428    1.0000

RLO =
    1.0000   -0.5409    0.2676   -0.0542    0.0289
   -0.5409    1.0000   -0.5991   -0.8344   -0.6708
    0.2676   -0.5991    1.0000    0.2887   -0.0673
   -0.0542   -0.8344    0.2887    1.0000    0.0265
    0.0289   -0.6708   -0.0673    0.0265    1.0000

RUP =
    1.0000    0.6051    0.9187    0.8490    0.8706
    0.6051    1.0000    0.5476    0.1038    0.4577
    0.9187    0.5476    1.0000    0.9222    0.8453
    0.8490    0.1038    0.9222    1.0000    0.8700
    0.8706    0.4577    0.8453    0.8700    1.0000

```

Or chose another value for the alpha threshold as the default one of 5%:



```

MATLAB Command Window
>> [R P L U]=corrcoef(T,'alpha',0.10) %the alpha impact only L and U not P!!

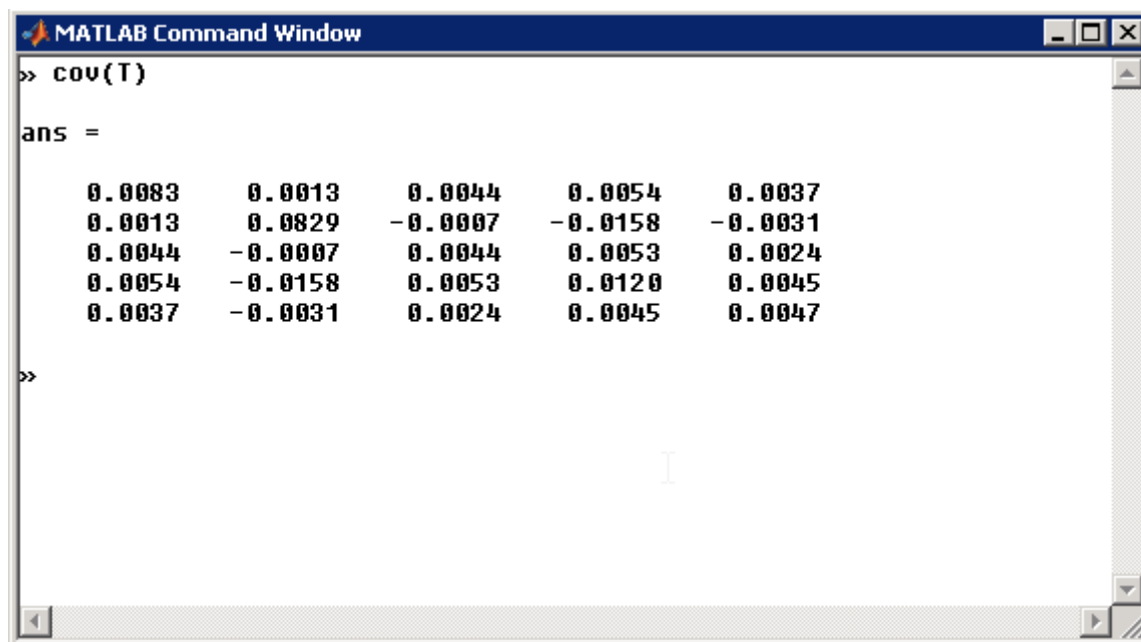
R =
    1.0000    0.0478    0.7295    0.5364    0.5930
    0.0478    1.0000   -0.0383   -0.4999   -0.1576
    0.7295   -0.0383    1.0000    0.7400    0.5270
    0.5364   -0.4999    0.7400    1.0000    0.5914
    0.5930   -0.1576    0.5270    0.5914    1.0000

P =
    1.0000    0.8827    0.0071    0.0722    0.0421
    0.8827    1.0000    0.9058    0.0980    0.6247
    0.0071    0.9058    1.0000    0.0059    0.0783
    0.0722    0.0980    0.0059    1.0000    0.0428
    0.0421    0.6247    0.0783    0.0428    1.0000

L =
    1.0000   -0.4625    0.3621    0.0507    0.1332
   -0.4625    1.0000   -0.5275   -0.7996   -0.6089
    0.3621   -0.5275    1.0000    0.3818    0.0377
    0.0507   -0.7996    0.3818    1.0000    0.1308
    0.1332   -0.6089    0.0377    0.1308    1.0000

U =
    1.0000    0.5343    0.9007    0.8169    0.8427
    0.5343    1.0000    0.4699   -0.0009    0.3708
    0.9007    0.4699    1.0000    0.9049    0.8125
    0.8169   -0.0009    0.9049    1.0000    0.8420
    0.8427    0.3708    0.8125    0.8420    1.0000
  
```

The non biased variance-covariance matrix will be calculated thanks to the **cov()** function:



```
MATLAB Command Window
>> cov(T)

ans =

    0.0083    0.0013    0.0044    0.0054    0.0037
    0.0013    0.0029   -0.0007   -0.0158   -0.0031
    0.0044   -0.0007    0.0044    0.0053    0.0024
    0.0054   -0.0158    0.0053    0.0120    0.0045
    0.0037   -0.0031    0.0024    0.0045    0.0047

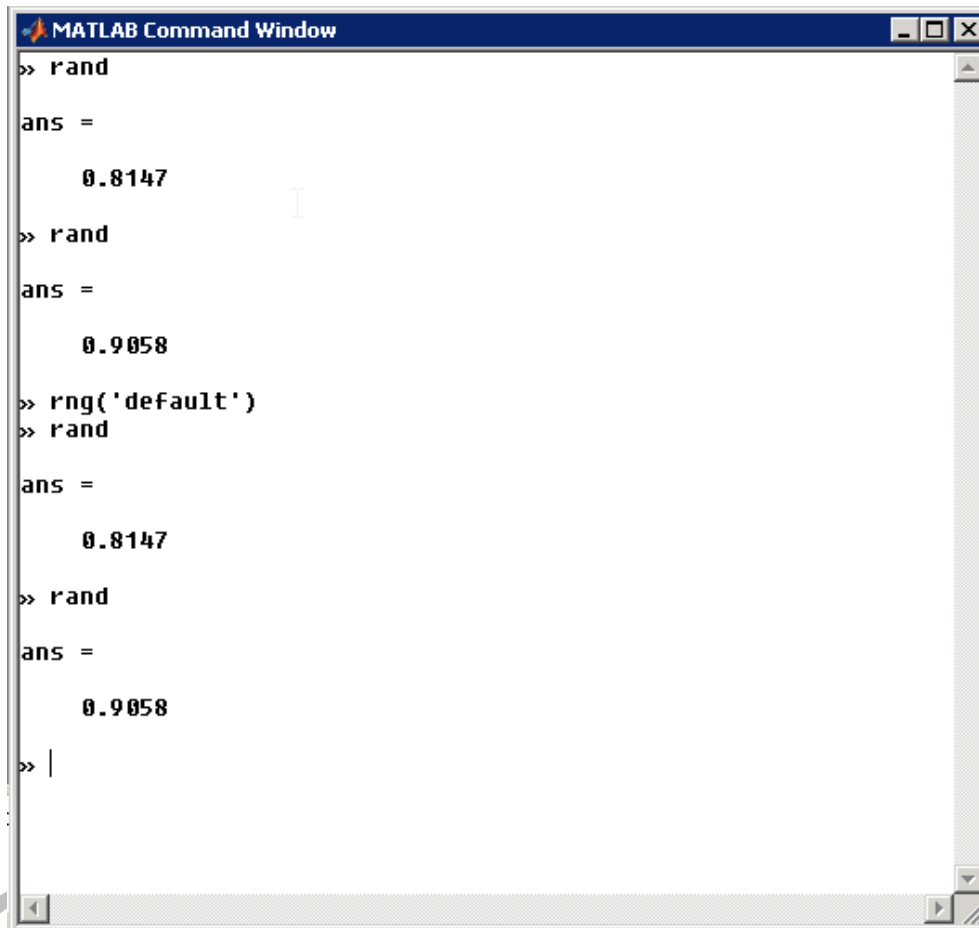
>>
```

It's a shame that there is no option that let us choose if we want the biased or non-biased covariance matrix!

Generate random variable

Generate and manage random variables in MATLAB is a huge subject especially if you don't have the Statistical Toolbox... We will see now the idea behind what is available in the core version of MATLAB.

First the **rand** command by default will give you a uniform distributed random variable. If you want to restart the random generator use the command **rng('default')**:



```
MATLAB Command Window
>> rand
ans =
    0.8147

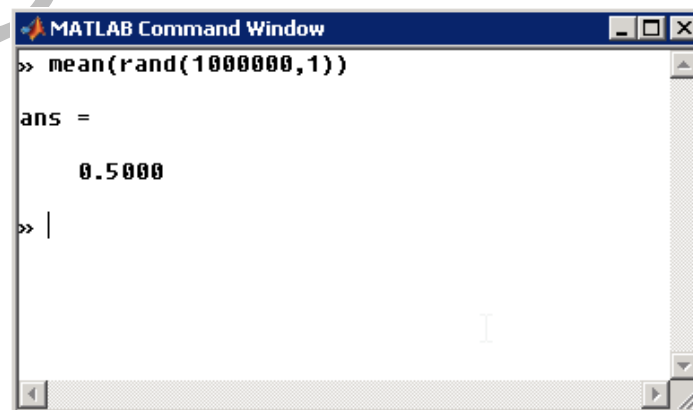
>> rand
ans =
    0.9058

>> rng('default')
>> rand
ans =
    0.8147

>> rand
ans =
    0.9058

>> |
```

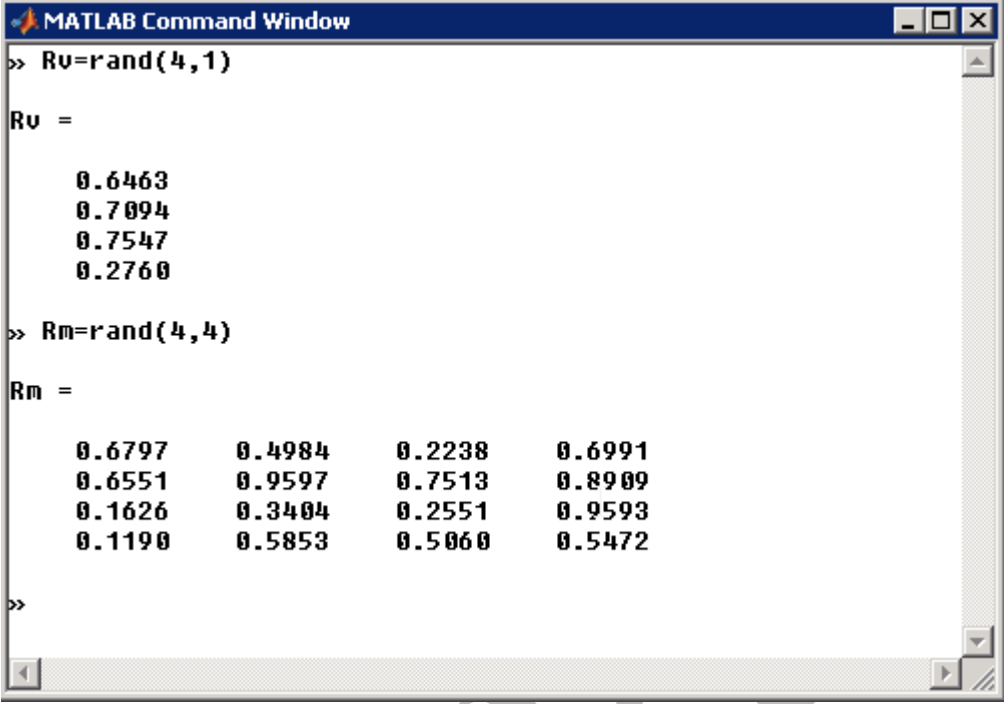
The rand is also a function **rand()**. You can pass it the number of rows and columns of random variables that you want:



```
MATLAB Command Window
>> mean(rand(1000000,1))
ans =
    0.5000

>> |
```

We can generate then vector columns but also matrices of random values:

A screenshot of the MATLAB Command Window. The title bar reads "MATLAB Command Window". The command prompt shows two commands: `Rv=rand(4,1)` and `Rm=rand(4,4)`. The output for `Rv` is a column vector with four values: 0.6463, 0.7094, 0.7547, and 0.2760. The output for `Rm` is a 4x4 matrix with values ranging from 0.1190 to 0.9593.

```
>> Rv=rand(4,1)

Rv =

    0.6463
    0.7094
    0.7547
    0.2760

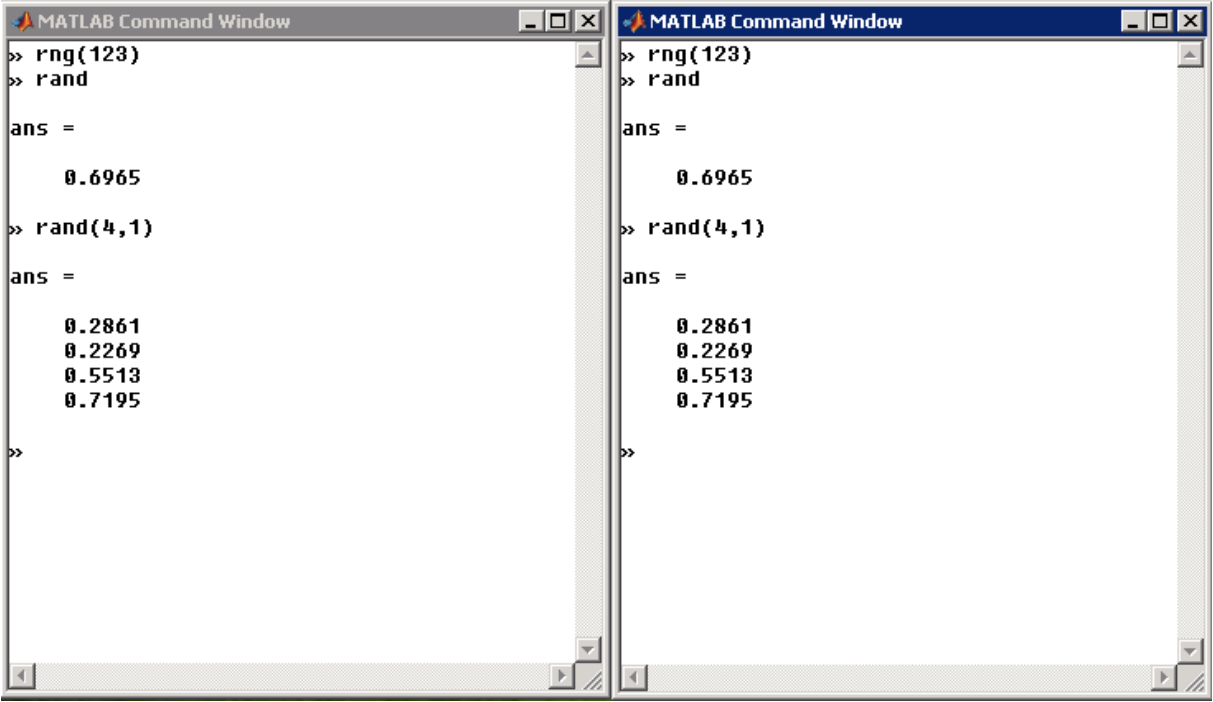
>> Rm=rand(4,4)

Rm =

    0.6797    0.4984    0.2238    0.6991
    0.6551    0.9597    0.7513    0.8909
    0.1626    0.3404    0.2551    0.9593
    0.1190    0.5853    0.5060    0.5472

>>
```

We can define a status for the random generator to be able to replicate the randomized values using the function `rng()`:

Two side-by-side screenshots of the MATLAB Command Window. Both windows show the same sequence of commands: `rng(123)`, `rand`, and `rand(4,1)`. The outputs are identical in both windows, demonstrating that setting the seed with `rng(123)` ensures reproducible random values. The first window shows the output for `rand` as 0.6965 and for `rand(4,1)` as a column vector [0.2861; 0.2269; 0.5513; 0.7195].

```
>> rng(123)
>> rand

ans =

    0.6965

>> rand(4,1)

ans =

    0.2861
    0.2269
    0.5513
    0.7195

>>
```

```
>> rng(123)
>> rand

ans =

    0.6965

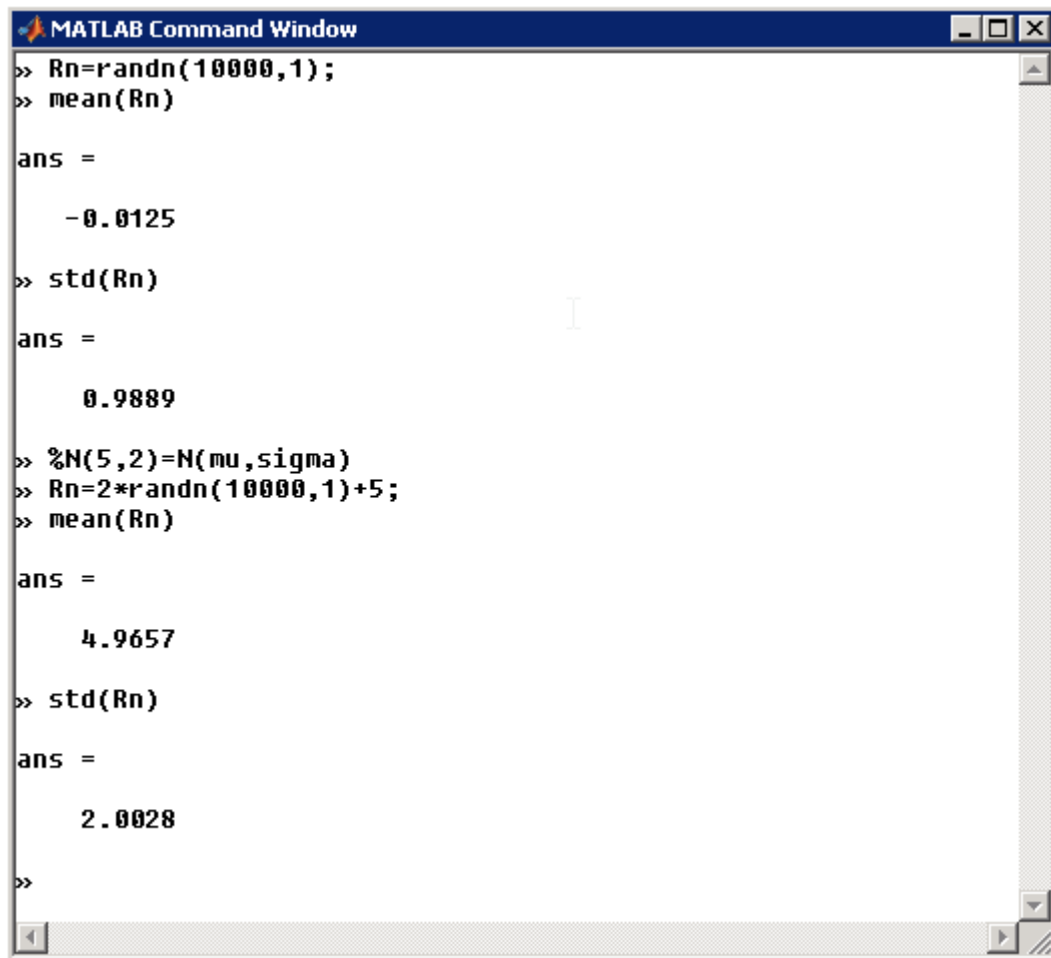
>> rand(4,1)

ans =

    0.2861
    0.2269
    0.5513
    0.7195

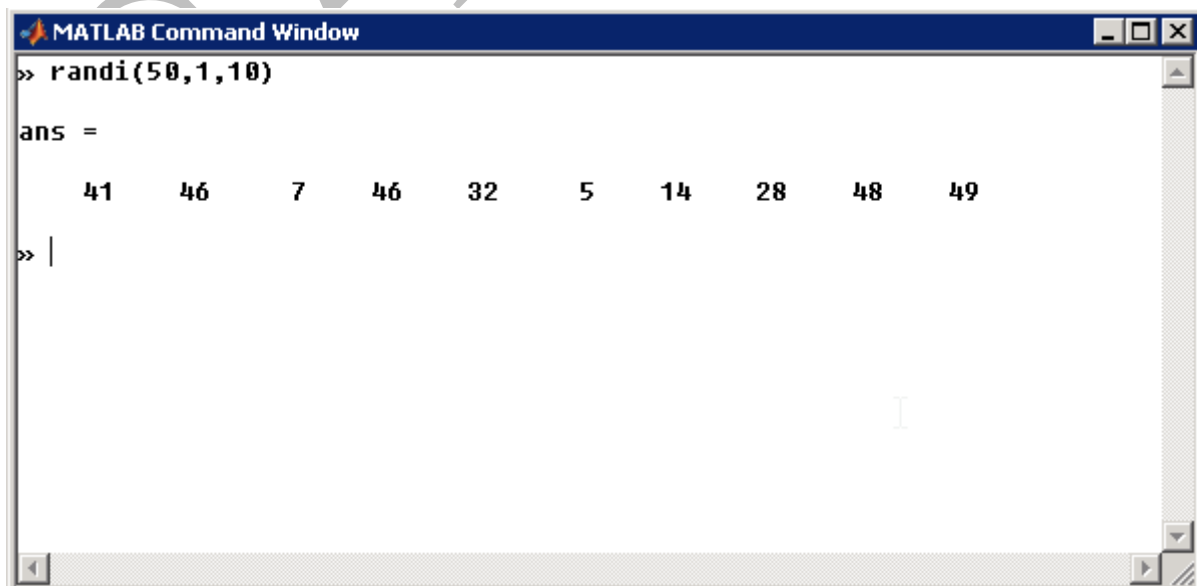
>>
```

The core version of MATLAB has also the possibility to generate and random Normal centered reduced variable with the function **randn()** and if you use the you mathematical knowledge in the fact you can generate any Normal random variable:

A screenshot of the MATLAB Command Window. The window has a blue title bar with the MATLAB logo and the text "MATLAB Command Window". The command history shows the following:

```
>> Rn=randn(10000,1);  
>> mean(Rn)  
  
ans =  
  
    -0.0125  
  
>> std(Rn)  
  
ans =  
  
    0.9889  
  
>> %N(5,2)=N(mu,sigma)  
>> Rn=2*randn(10000,1)+5;  
>> mean(Rn)  
  
ans =  
  
    4.9657  
  
>> std(Rn)  
  
ans =  
  
    2.0028  
  
>>
```

You can also generate random integer value between two ranges (very useful!) with the function **randi()**:

A screenshot of the MATLAB Command Window. The window has a blue title bar with the MATLAB logo and the text "MATLAB Command Window". The command history shows the following:

```
>> randi(50,1,10)  
  
ans =  
  
    41    46     7    46    32     5    14    28    48    49  
  
>> |
```

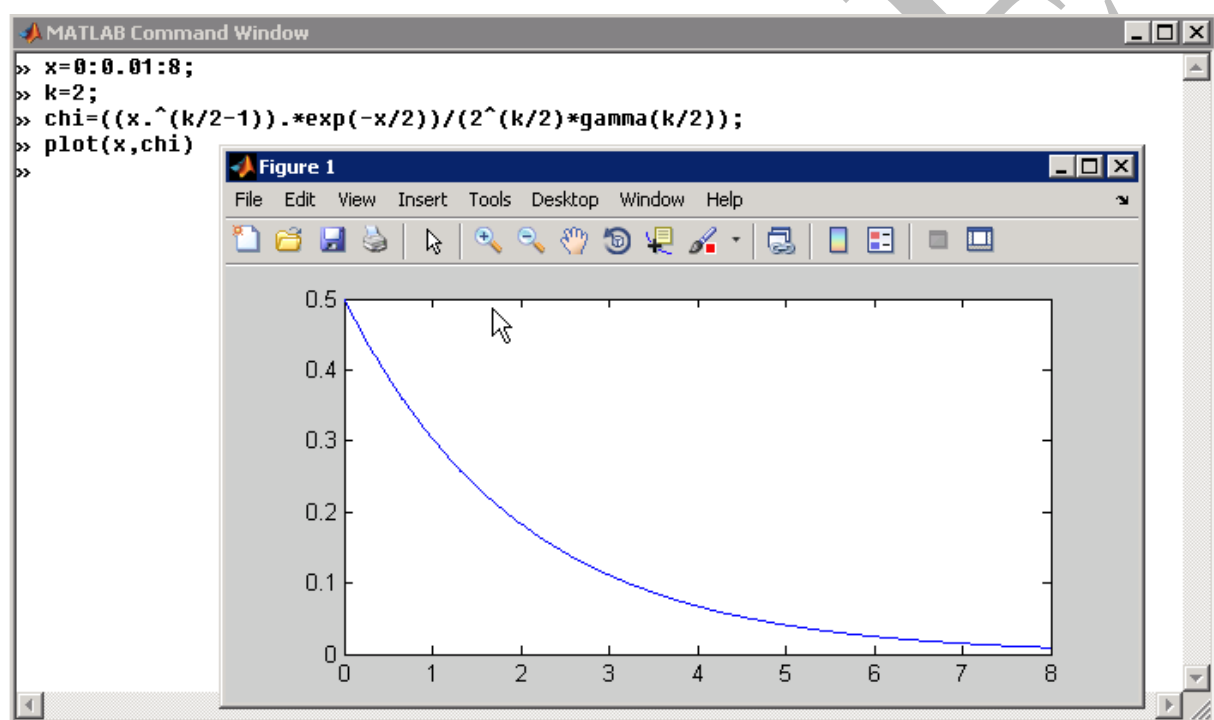
But what can you do if you want simulate a random variable that is out of the box of the MATLAB core version?

Then you just have to play with you knowledge of statistics. Imagine we want of example to simulate a chi-square random variable.

We know that:

$$\chi_k^2 = f(x, k)_{[0+\infty[} = \frac{x^{k/2-1} e^{-x/2}}{2^{k/2} \Gamma\left(\frac{k}{2}\right)}$$

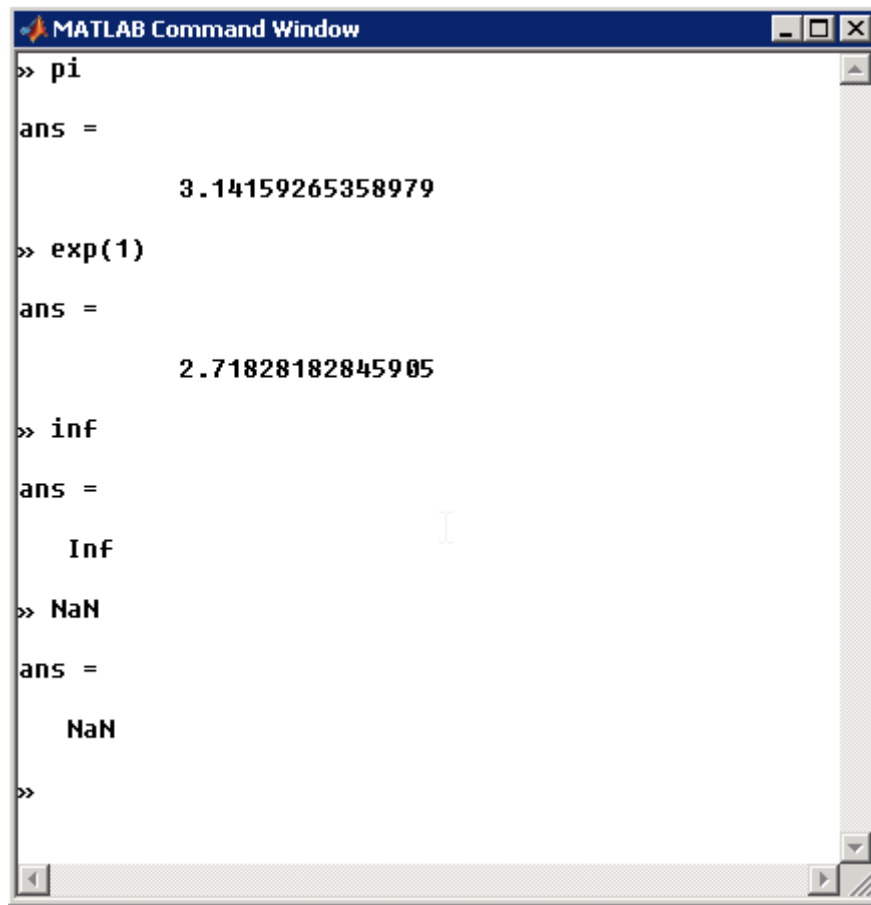
Thus:



and so on for any other rand variable...

Use of constants (permanent variables)

In MATLAB there are a few mathematical constants only:



```
MATLAB Command Window

>> pi
ans =
    3.14159265358979

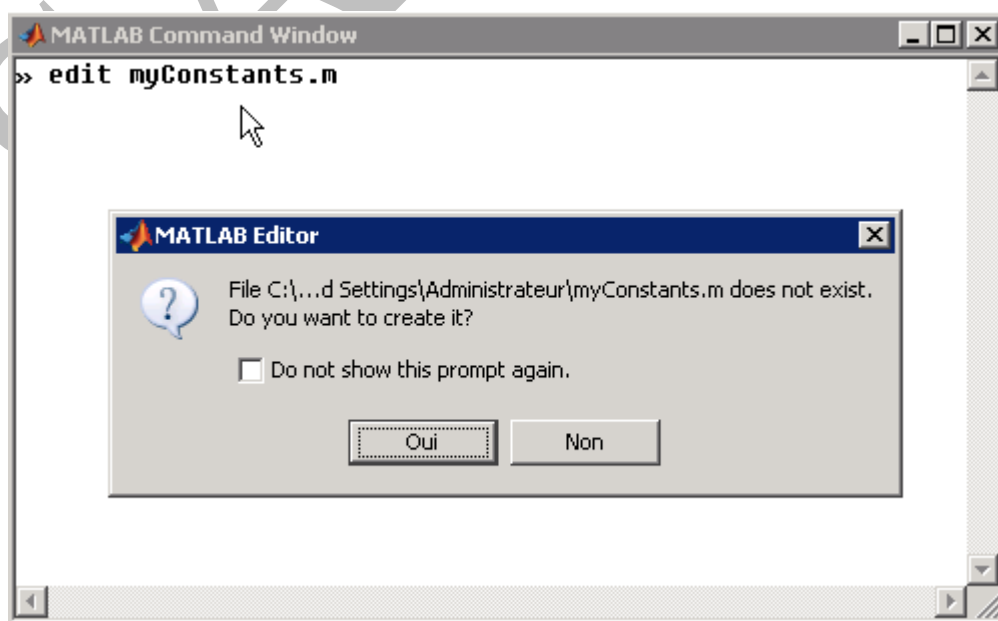
>> exp(1)
ans =
    2.71828182845905

>> inf
ans =
    Inf

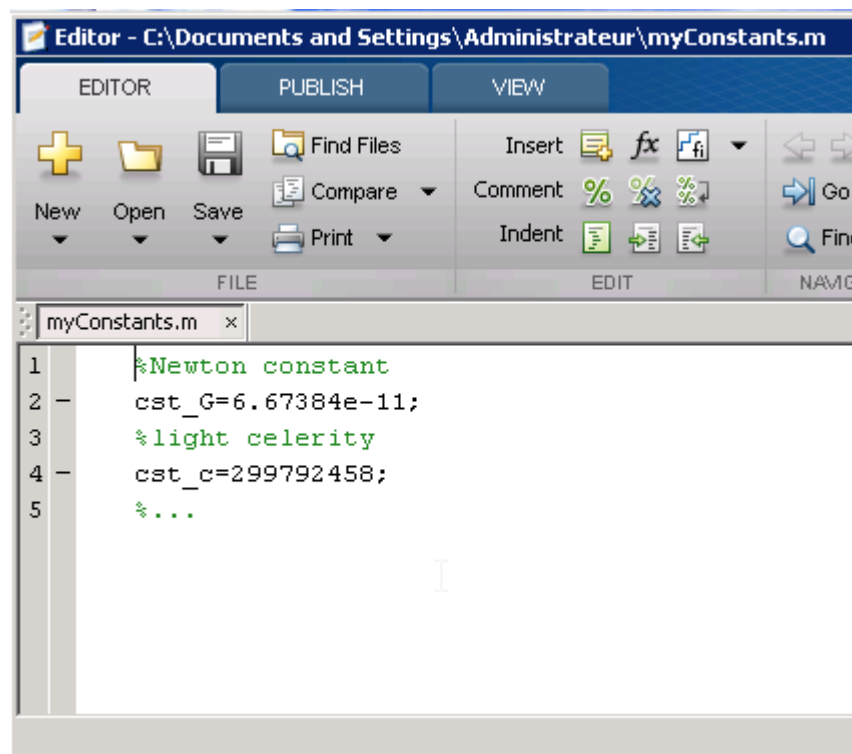
>> NaN
ans =
    NaN

>>
```

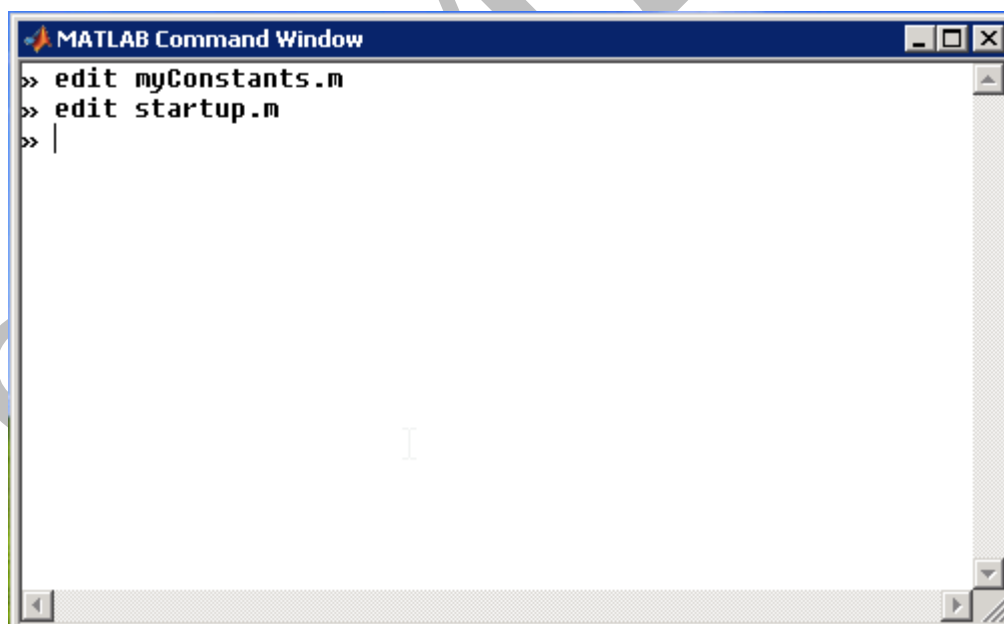
If you are a physicist this is a an issue. A workaround is to create a new script file (the name doesn't really matter):



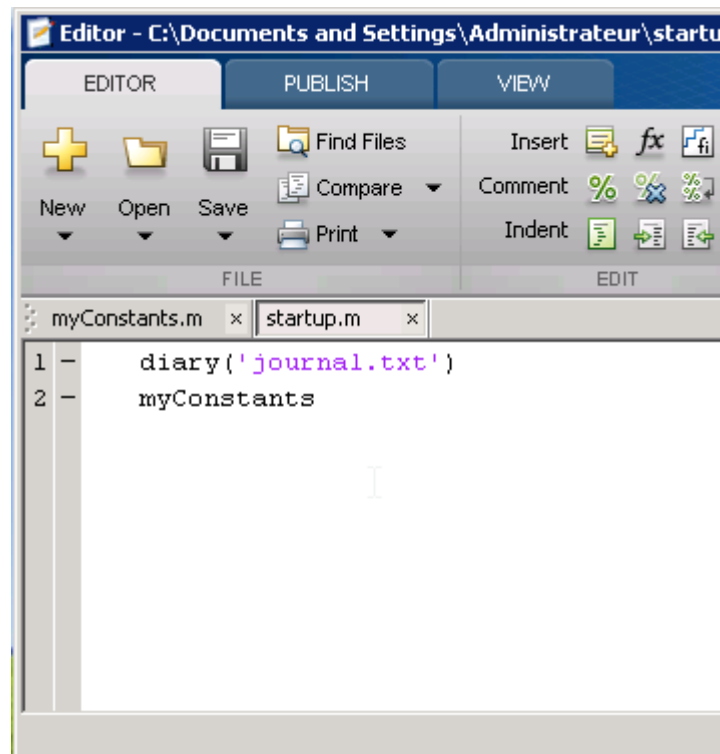
Type in all the constants you use for your work:



Once saved, open the file *startup.m*.



And call your script with your constants just by writing the script file name (if you don't specify the path that means that your file is in a MATLAB path):

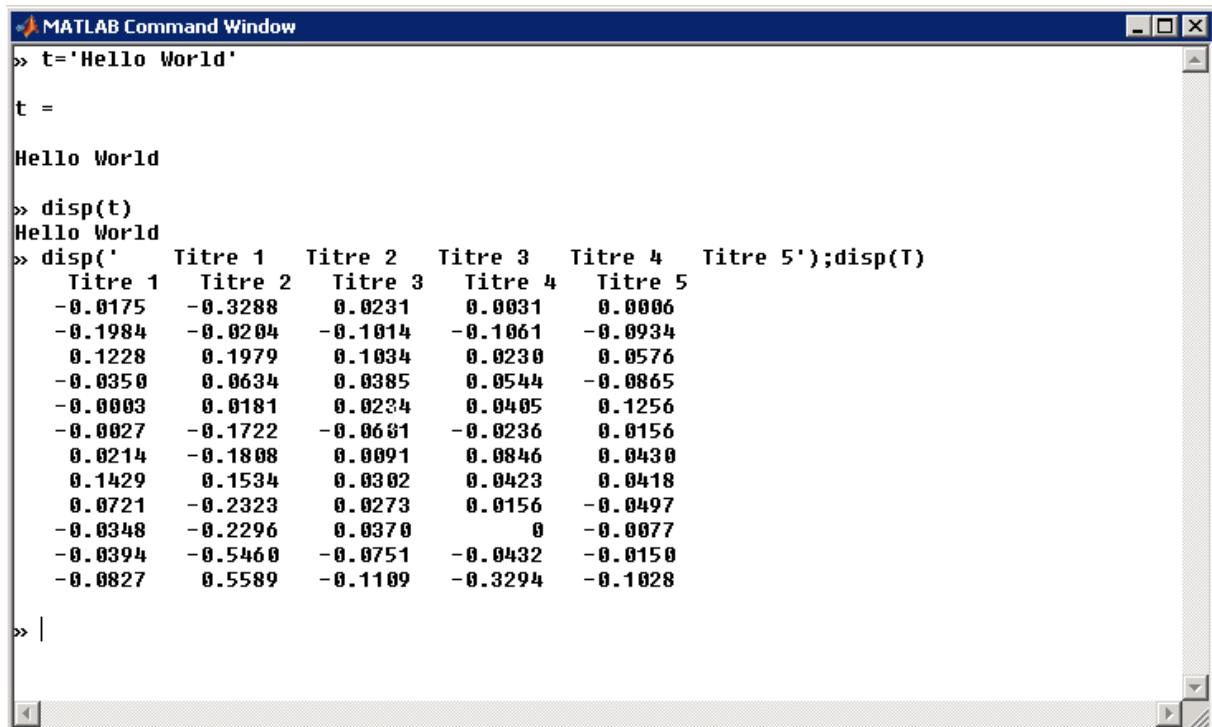


And the job is done!

Working with strings

Working with strings is very important especially when you work with charts or that you automate reporting.

One of the first point to learn is to know different ways to add title rows to a matrix using a small trick with the function **disp()**:



```

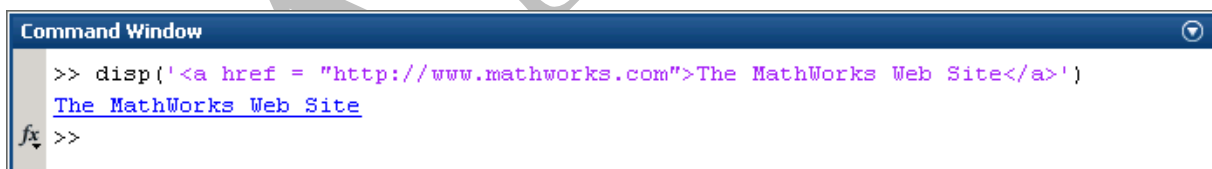
>> t='Hello World'

t =

Hello World

>> disp(t)
Hello World
>> disp('    Titre 1    Titre 2    Titre 3    Titre 4    Titre 5');disp(T)
    Titre 1    Titre 2    Titre 3    Titre 4    Titre 5
-0.0175   -0.3288    0.0231    0.0031    0.0006
-0.1984   -0.0204   -0.1014   -0.1061   -0.0934
 0.1228    0.1979    0.1034    0.0230    0.0576
-0.0350    0.0634    0.0385    0.0544   -0.0865
-0.0003    0.0181    0.0234    0.0405    0.1256
-0.0027   -0.1722   -0.0601   -0.0236    0.0156
 0.0214   -0.1808    0.0091    0.0846    0.0430
 0.1429    0.1534    0.0302    0.0423    0.0418
 0.0721   -0.2323    0.0273    0.0156   -0.0497
-0.0348   -0.2296    0.0370         0   -0.0077
-0.0394   -0.5460   -0.0751   -0.0432   -0.0150
-0.0827    0.5589   -0.1109   -0.3294   -0.1028
  
```

Or if you want to display a link (very useful for documentation):

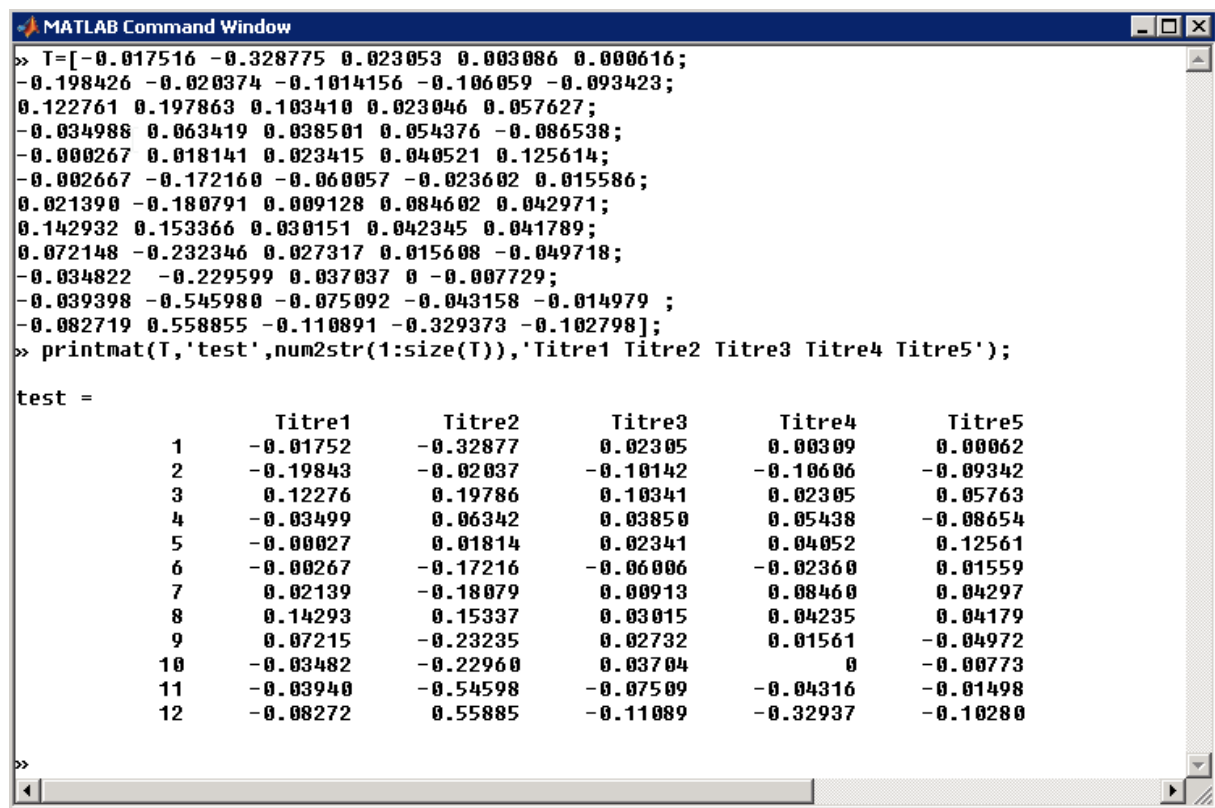


```

>> disp('<a href = "http://www.mathworks.com">The MathWorks Web Site</a>')
The MathWorks Web Site
fx >>
  
```

Note: This doesn't work in the old style Command Window!!!

There is another official way to do the same using the function **printmat()**:



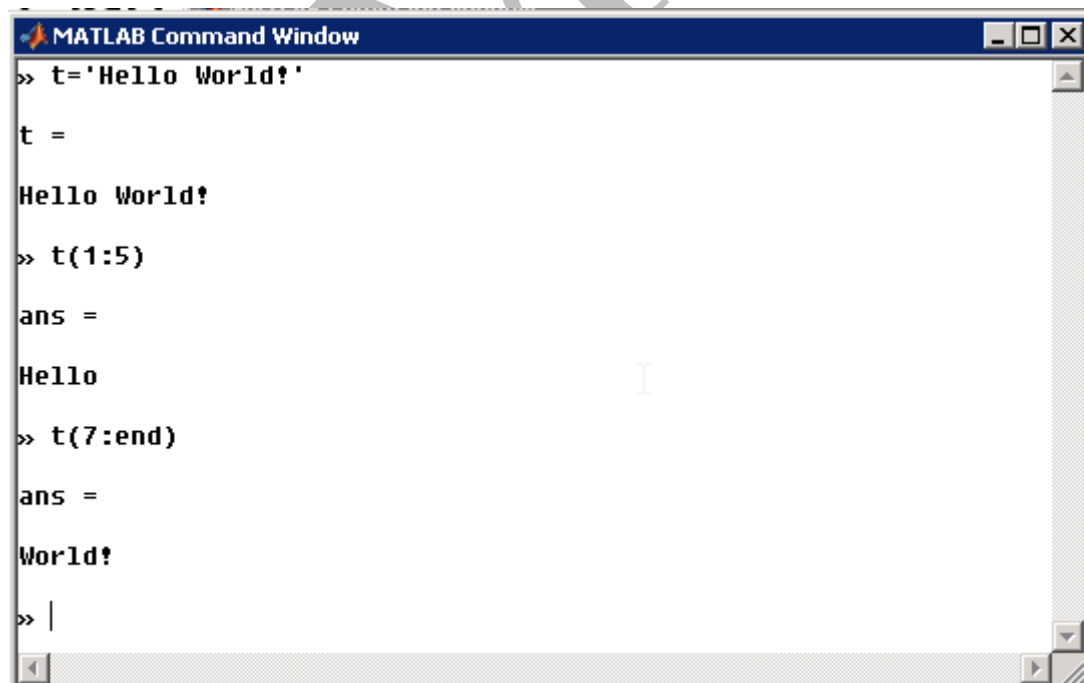
```

>> T=[-0.017516 -0.328775 0.023053 0.003086 0.000616;
-0.198426 -0.020374 -0.1014156 -0.106059 -0.093423;
0.122761 0.197863 0.103410 0.023046 0.057627;
-0.034988 0.063419 0.038501 0.054376 -0.086538;
-0.000267 0.018141 0.023415 0.040521 0.125614;
-0.002667 -0.172160 -0.060057 -0.023602 0.015586;
0.021390 -0.180791 0.009128 0.084602 0.042971;
0.142932 0.153366 0.030151 0.042345 0.041789;
0.072148 -0.232346 0.027317 0.015608 -0.049718;
-0.034822 -0.229599 0.037037 0 -0.007729;
-0.039398 -0.545980 -0.075092 -0.043158 -0.014979 ;
-0.082719 0.558855 -0.110891 -0.329373 -0.102798];
>> printmat(T,'test',num2str(1:size(T)),'Titre1 Titre2 Titre3 Titre4 Titre5');

test =

      1      Titre1      Titre2      Titre3      Titre4      Titre5
      2      -0.01752      -0.32877      0.02305      0.00309      0.00062
      3      -0.19843      -0.02037      -0.10142      -0.10606      -0.09342
      4      0.12276      0.19786      0.10341      0.02305      0.05763
      5      -0.03499      0.06342      0.03850      0.05438      -0.08654
      6      -0.00027      0.01814      0.02341      0.04052      0.12561
      7      -0.00267      -0.17216      -0.06006      -0.02360      0.01559
      8      0.02139      -0.18079      0.00913      0.08460      0.04297
      9      0.14293      0.15337      0.03015      0.04235      0.04179
     10      0.07215      -0.23235      0.02732      0.01561      -0.04972
     11      -0.03482      -0.22960      0.03704      0          -0.00773
     12      -0.03940      -0.54598      -0.07509      -0.04316      -0.01498
     13      -0.08272      0.55885      -0.11089      -0.32937      -0.10280
  
```

If we want to extract a substring of string remember that strings are also vectors by default:



```

>> t='Hello World!'

t =

Hello World!

>> t(1:5)

ans =

Hello

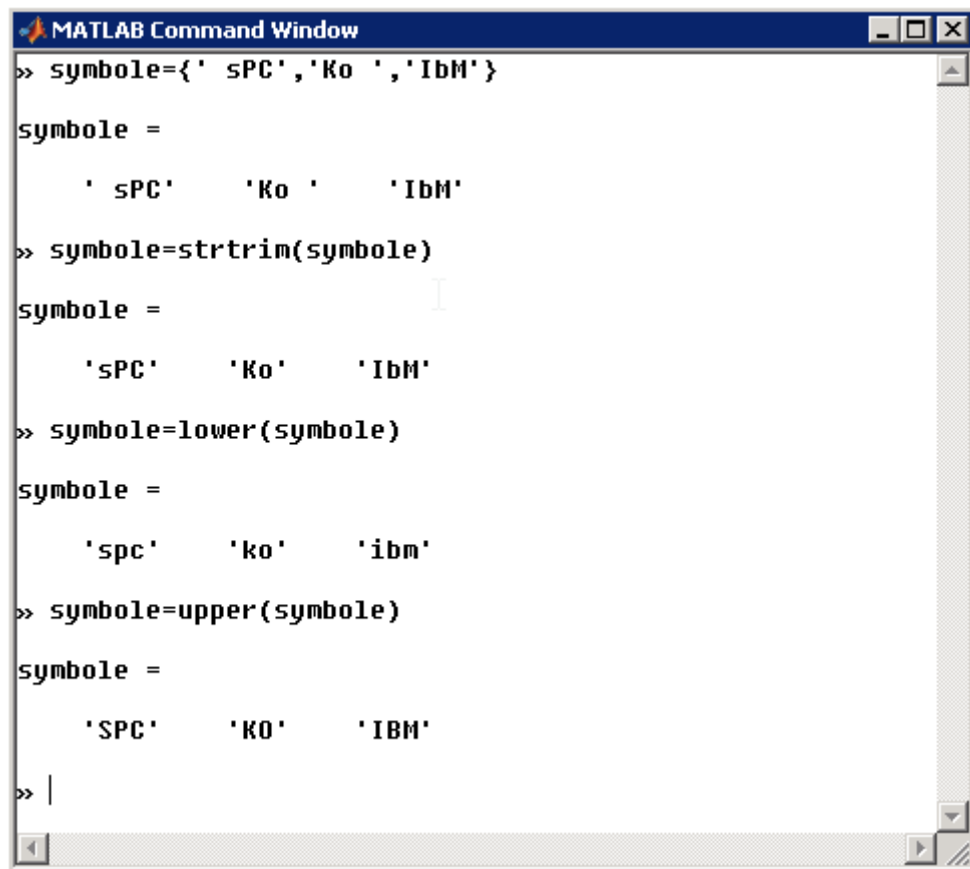
>> t(7:end)

ans =

World!
  
```

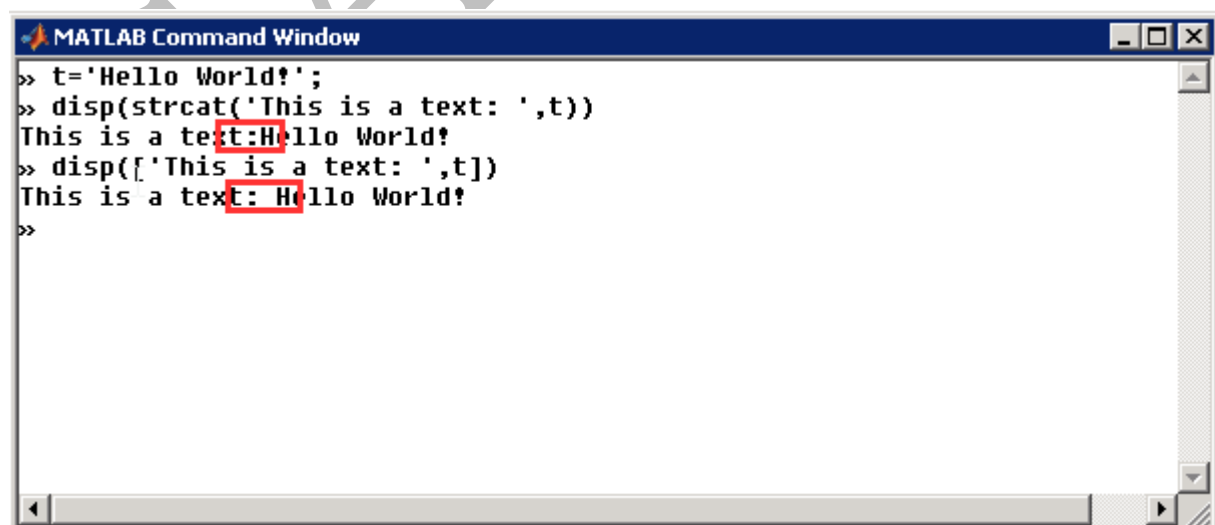
Now what about manipulating cell array that by default are made to contain strings?

First we can remove non desired spaces using the **strtrim()** function or force lower or uppercase respectively with the functions **lower()** or **upper()**:



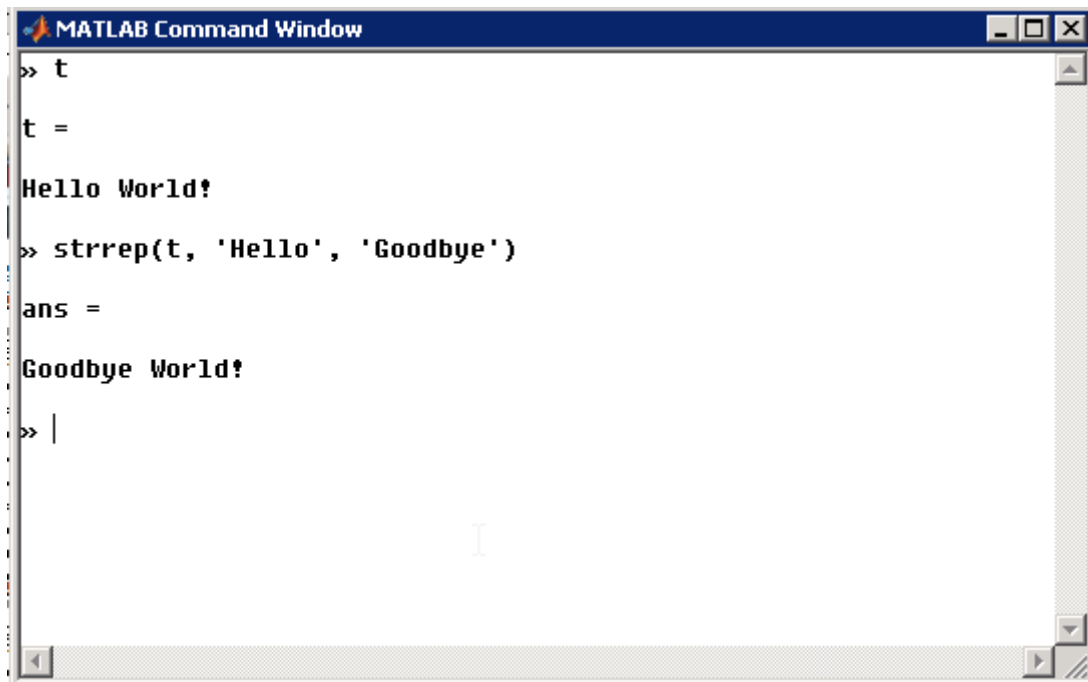
```
MATLAB Command Window
>> symbole={' sPC','Ko ','IbM'}
symbole =
    ' sPC'    'Ko '    'IbM'
>> symbole=strtrim(symbole)
symbole =
    'sPC'    'Ko'    'IbM'
>> symbole=lower(symbole)
symbole =
    'spc'    'ko'    'ibm'
>> symbole=upper(symbole)
symbole =
    'SPC'    'KO'    'IBM'
>> |
```

A typical operation is to concatenate strings together. If you do that using the both functions **disp()** and **strcat()** the blank spaces will be removed as you can see below. But if you use a vector of strings then everything will be alright!:



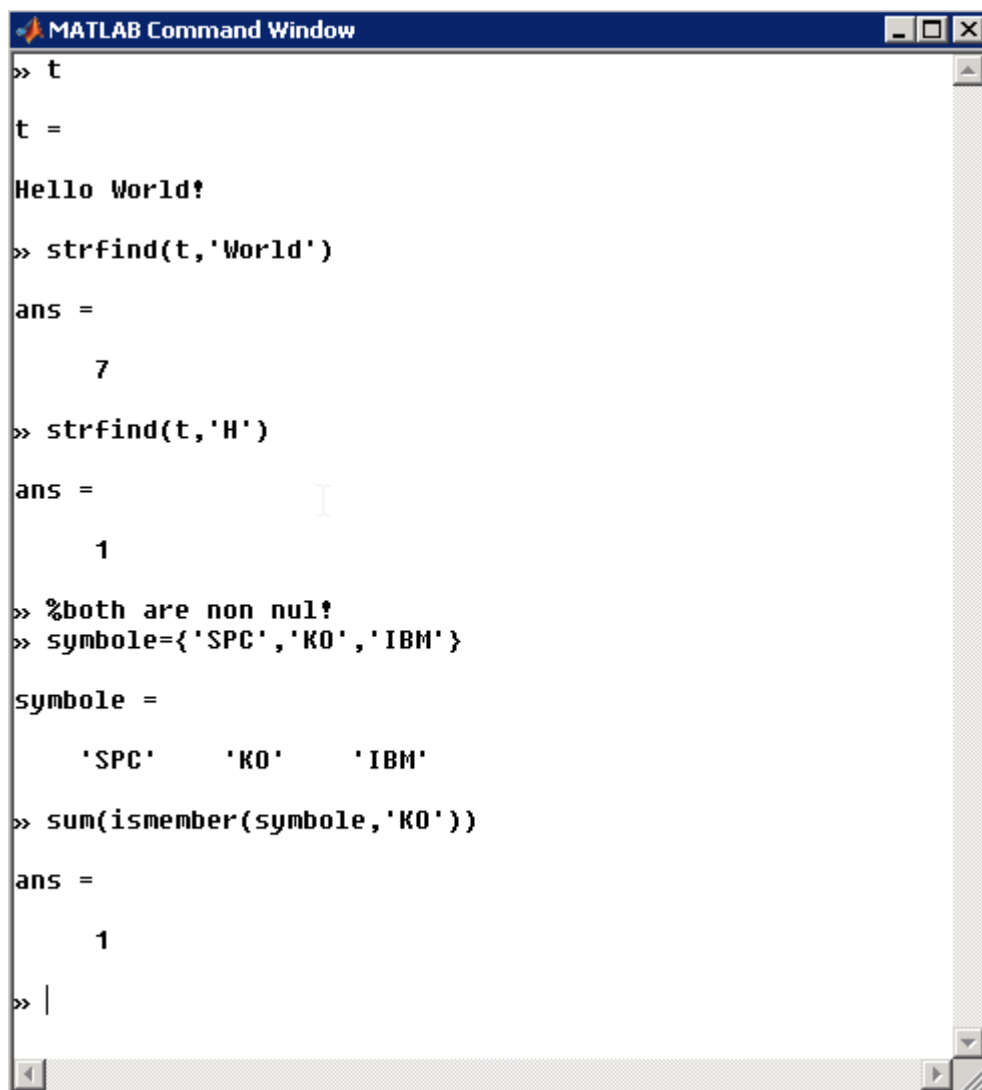
```
MATLAB Command Window
>> t='Hello World!';
>> disp(strcat('This is a text: ',t))
This is a text:Hello World!
>> disp(['This is a text: ',t])
This is a text: Hello World!
>>
```

Something also useful sometimes is to replace a word by another one using the function **strrep()**:

A screenshot of the MATLAB Command Window. The window has a blue title bar with the MATLAB logo and the text "MATLAB Command Window". The command history shows: first, the variable 't' is assigned the string 'Hello World!'; second, the command 'strrep(t, 'Hello', 'Goodbye')' is entered, resulting in the output 'ans = Goodbye World!'. The prompt '» |' is visible at the bottom of the command history.

```
» t
t =
Hello World!
» strrep(t, 'Hello', 'Goodbye')
ans =
Goodbye World!
» |
```

Sometimes it can be useful to search a character or a word in a string or even in if it exists in a cell array. You can achieve this respectively thanks to the functions **strfind()** and **ismember()**:



```
MATLAB Command Window

>> t

t =

Hello World!

>> strfind(t,'World')

ans =

     7

>> strfind(t,'H')

ans =

     1

>> %both are non nul!
>> symbole={'SPC','KO','IBM'}

symbole =

    'SPC'    'KO'    'IBM'

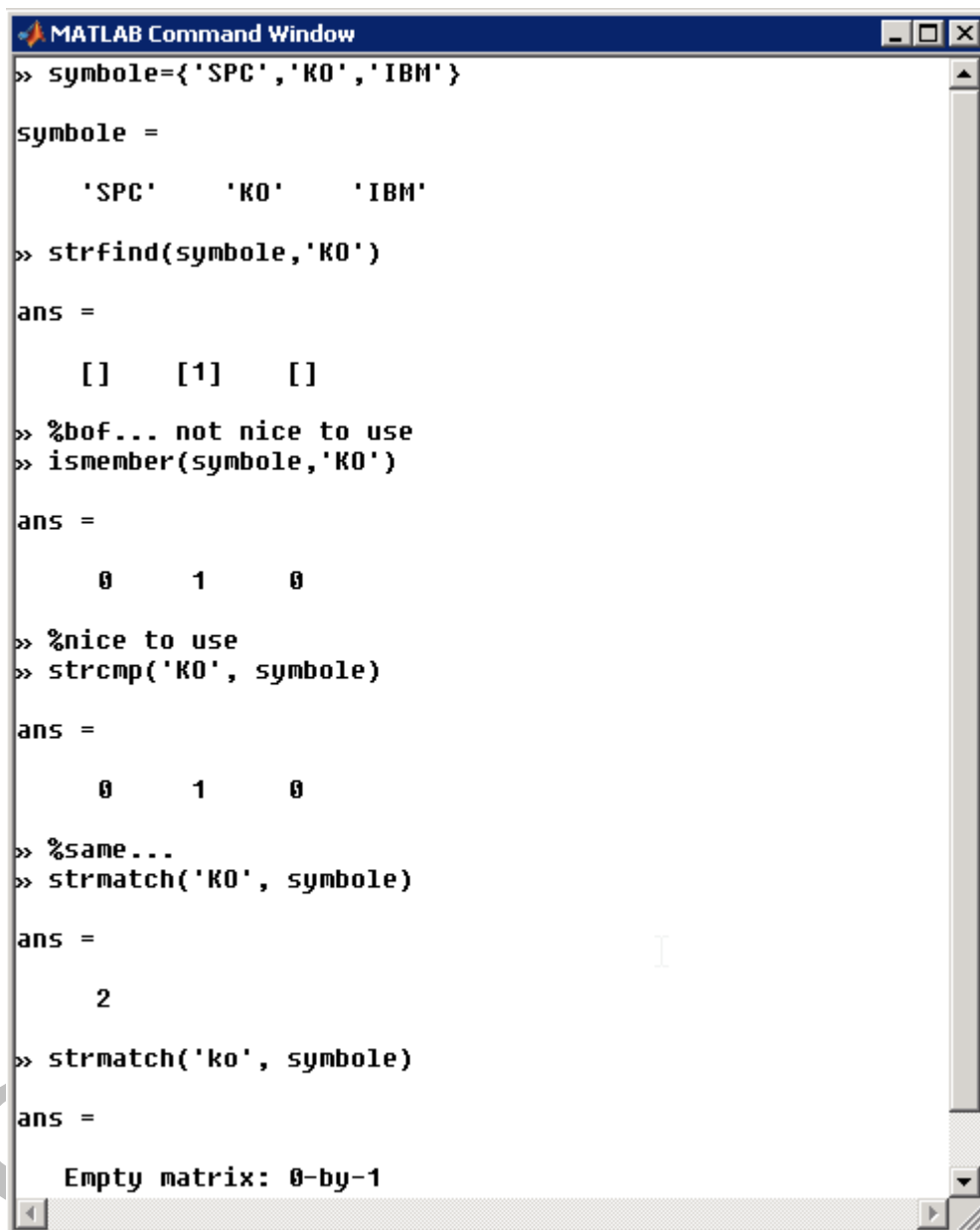
>> sum(ismember(symbole,'KO'))

ans =

     1

>> |
```

As you can see we can also use **strfind()** to search in a cell array but output is not very nice. Otherwise the functions **strcmp()** and **strmatch()** do almost the same with the difference that **strmatch()** is case sensitive:

A screenshot of the MATLAB Command Window. The window has a blue title bar with the MATLAB logo and the text "MATLAB Command Window". The command history shows several lines of code and their corresponding outputs. The code starts with creating a cell array 'symbole' containing 'SPC', 'KO', and 'IBM'. Then, 'strfind' is used to search for 'KO', resulting in a cell array of indices. A comment '%bof... not nice to use' is followed by 'ismember', which returns a logical array. Another comment '%nice to use' is followed by 'strcmp', which also returns a logical array. A third comment '%same...' is followed by 'strmatch' with 'KO', which returns the index 2. Finally, 'strmatch' is used with 'ko' (lowercase), which returns an empty matrix, indicating no case-sensitive match was found.

```
>> symbole={'SPC','KO','IBM'}

symbole =

    'SPC'    'KO'    'IBM'

>> strfind(symbole,'KO')

ans =

     []     [1]     []

>> %bof... not nice to use
>> ismember(symbole,'KO')

ans =

     0     1     0

>> %nice to use
>> strcmp('KO', symbole)

ans =

     0     1     0

>> %same...
>> strmatch('KO', symbole)

ans =

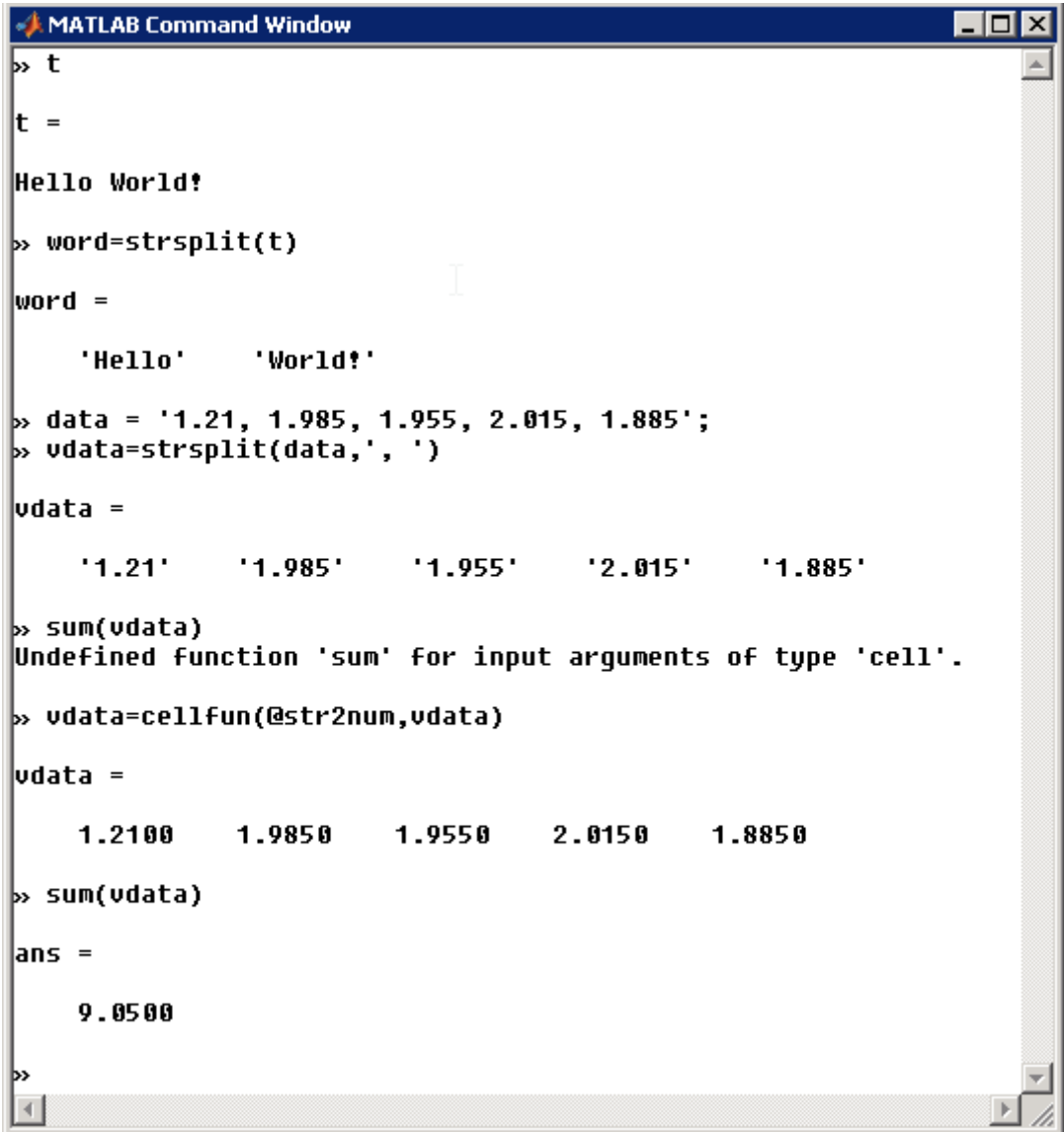
     2

>> strmatch('ko', symbole)

ans =

Empty matrix: 0-by-1
```

The function **strsplit()** is also sometimes useful when you create GUI and a field where people can input multiples values at once. The problem is that you must not forget to convert the values to numerics using after the cell array function **str2num()**:



```
MATLAB Command Window

>> t

t =

Hello World!

>> word=strsplit(t)

word =

    'Hello'    'World!'

>> data = '1.21, 1.985, 1.955, 2.015, 1.885';
>> vdata=strsplit(data,',')

vdata =

    '1.21'    '1.985'    '1.955'    '2.015'    '1.885'

>> sum(vdata)
Undefined function 'sum' for input arguments of type 'cell'.

>> vdata=cellfun(@str2num,vdata)

vdata =

    1.2100    1.9850    1.9550    2.0150    1.8850

>> sum(vdata)

ans =

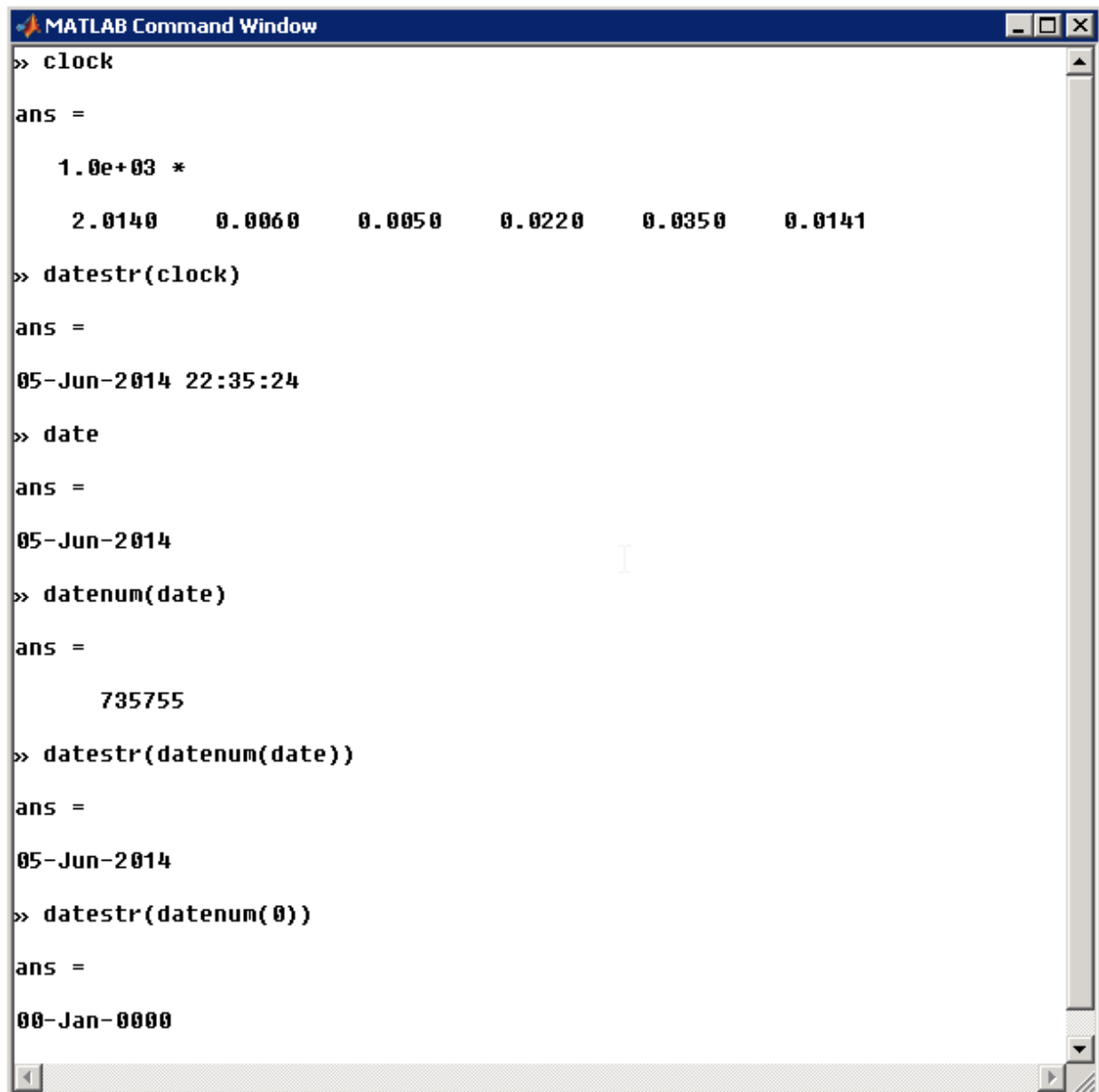
    9.0500

>>
```

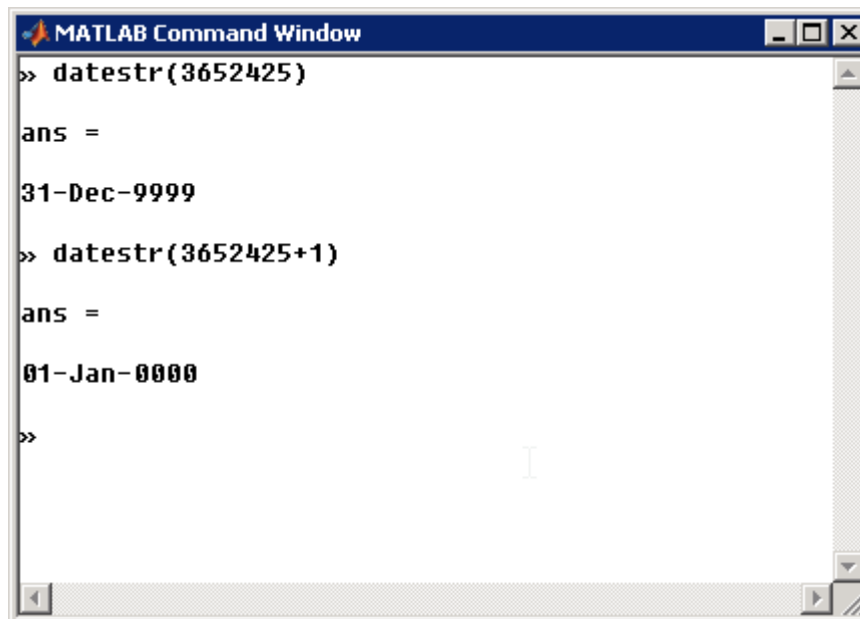
Working with dates

Working with dates is something also very important because most softwares and peoples around the world doesn't respect ISO norms...

Let us see first the different way to obtain the actual time using the commands **clock** and **date** together with the functions **datestr()** and **datenum()**:

A screenshot of the MATLAB Command Window. The window has a blue title bar with the MATLAB logo and the text "MATLAB Command Window". The command history shows the following:
1. Command: `>> clock`
Output: `ans =`
`1.0e+03 *`
`2.0140 0.0060 0.0050 0.0220 0.0350 0.0141`
2. Command: `>> datestr(clock)`
Output: `ans =`
`05-Jun-2014 22:35:24`
3. Command: `>> date`
Output: `ans =`
`05-Jun-2014`
4. Command: `>> datenum(date)`
Output: `ans =`
`735755`
5. Command: `>> datestr(datenum(date))`
Output: `ans =`
`05-Jun-2014`
6. Command: `>> datestr(datenum(0))`
Output: `ans =`
`00-Jan-0000`

We can also have a look at the biggest and smallest date available in MATLAB by playing with **datestr()**:



```
MATLAB Command Window
>> datestr(3652425)

ans =

31-Dec-9999

>> datestr(3652425+1)

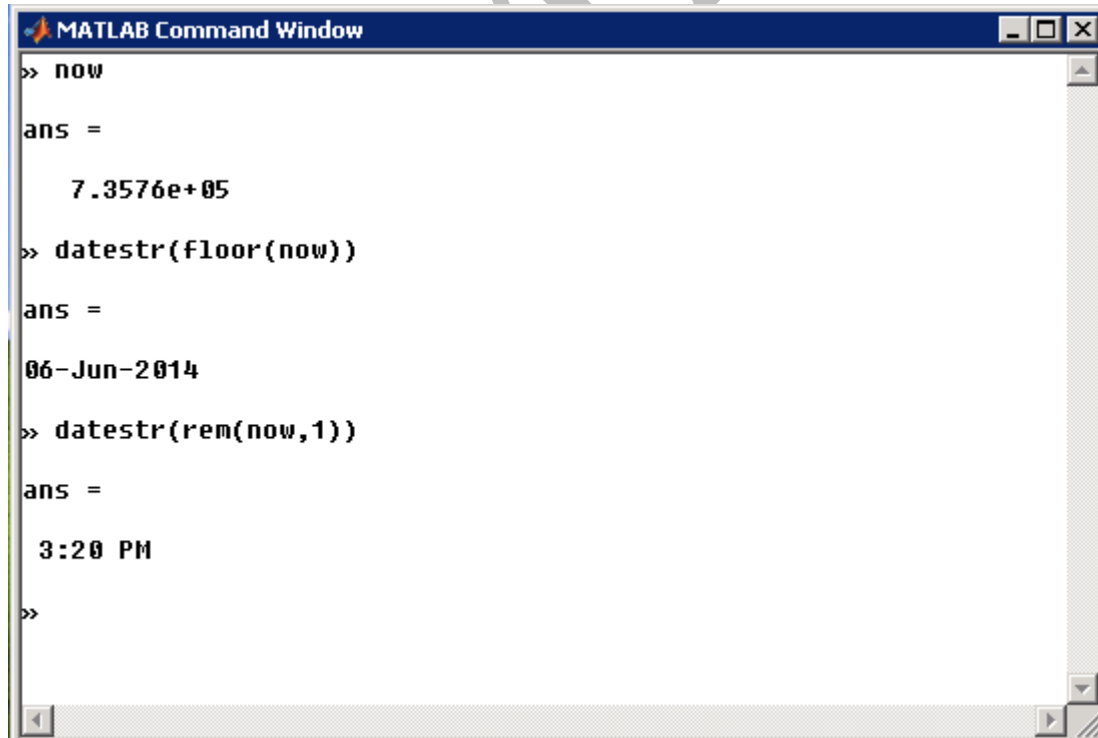
ans =

01-Jan-0000

>>
```

These poor limits can be problematic for astronomers...!

We can also play with the command **now** to get the actual date and time and if you want to get only the date or only the time you will have to play with the functions **floor()** and **rem()**:



```
MATLAB Command Window
>> now

ans =

7.3576e+05

>> datestr(floor(now))

ans =

06-Jun-2014

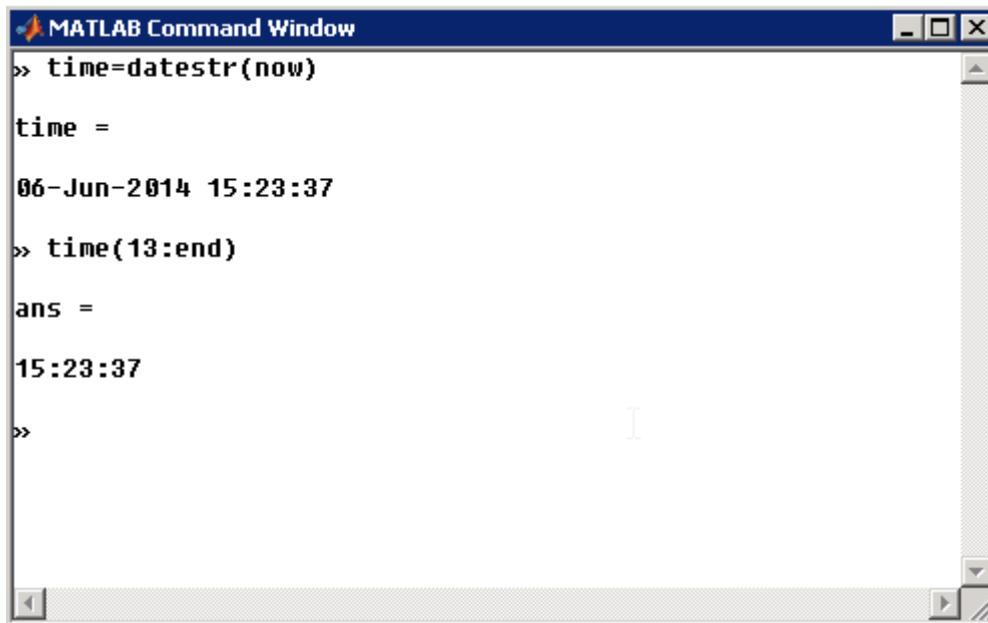
>> datestr(rem(now,1))

ans =

3:20 PM

>>
```


Here you can see the real full output of the command **now** and another way to get the time + the seconds:



```

MATLAB Command Window
>> time=datestr(now)

time =

06-Jun-2014 15:23:37

>> time(13:end)

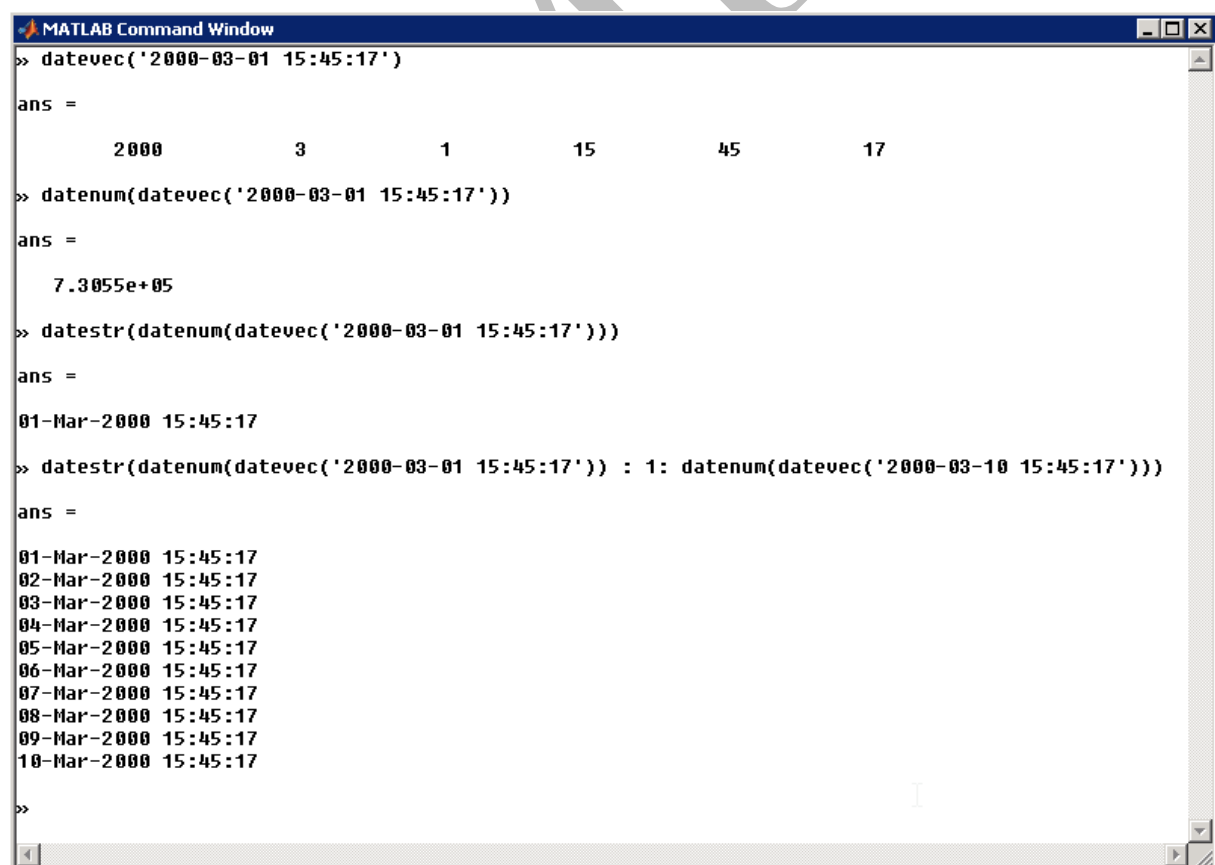
ans =

15:23:37

>>

```

It is also very interesting to see what happens when you pass a date through the function **datevec()** and also how to build a sequence of dates using **datevec()** in association with **datenum()** and **datestr()**:



```

MATLAB Command Window
>> datevec('2000-03-01 15:45:17')

ans =

    2000         3         1        15        45        17

>> datenum(datevec('2000-03-01 15:45:17'))

ans =

7.3055e+05

>> datestr(datenum(datevec('2000-03-01 15:45:17'))

ans =

01-Mar-2000 15:45:17

>> datestr(datenum(datevec('2000-03-01 15:45:17')) : 1: datenum(datevec('2000-03-10 15:45:17')))

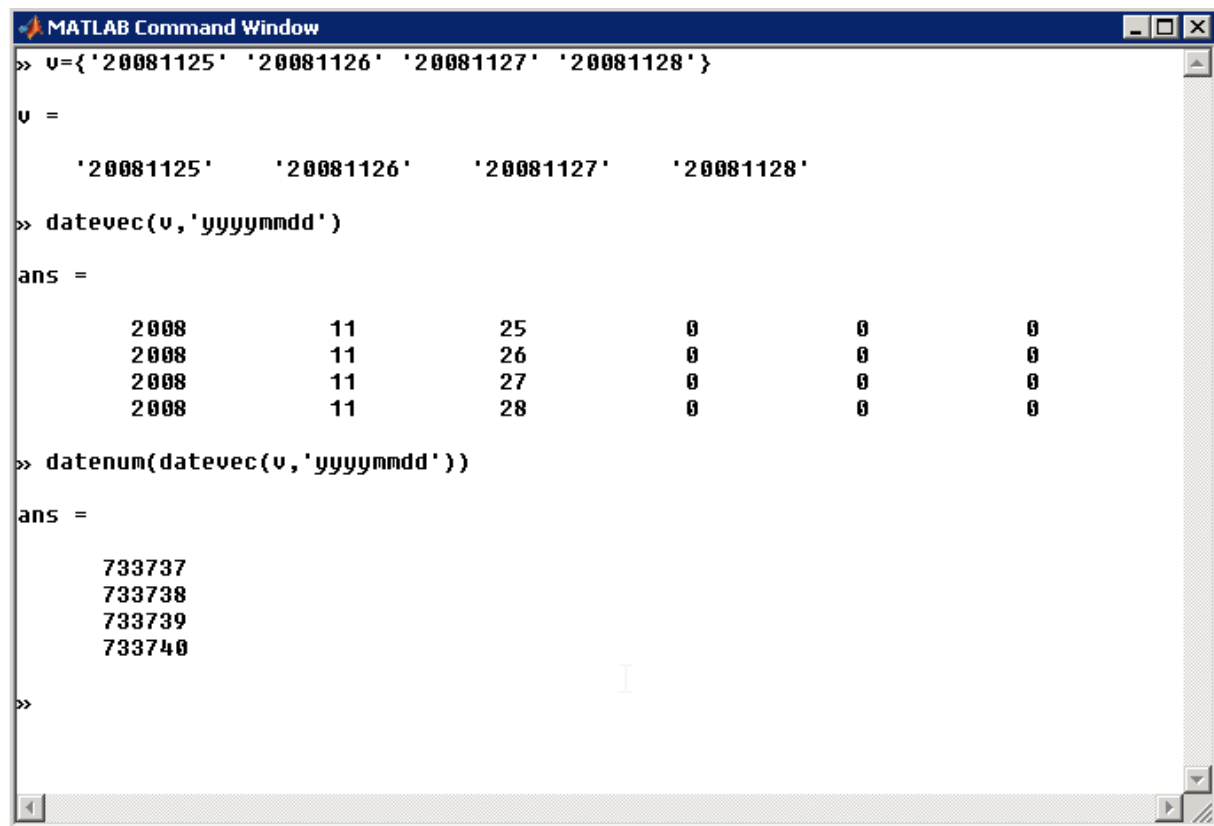
ans =

01-Mar-2000 15:45:17
02-Mar-2000 15:45:17
03-Mar-2000 15:45:17
04-Mar-2000 15:45:17
05-Mar-2000 15:45:17
06-Mar-2000 15:45:17
07-Mar-2000 15:45:17
08-Mar-2000 15:45:17
09-Mar-2000 15:45:17
10-Mar-2000 15:45:17

>>

```

Even when the dates are given in some horrible format you can still give a formatting argument to the function **datevec()** so that it can recognize the date pattern of you original datas:



```
MATLAB Command Window
>> v={'20081125' '20081126' '20081127' '20081128'}

v =
    '20081125'    '20081126'    '20081127'    '20081128'

>> datevec(v,'yyyymmdd')

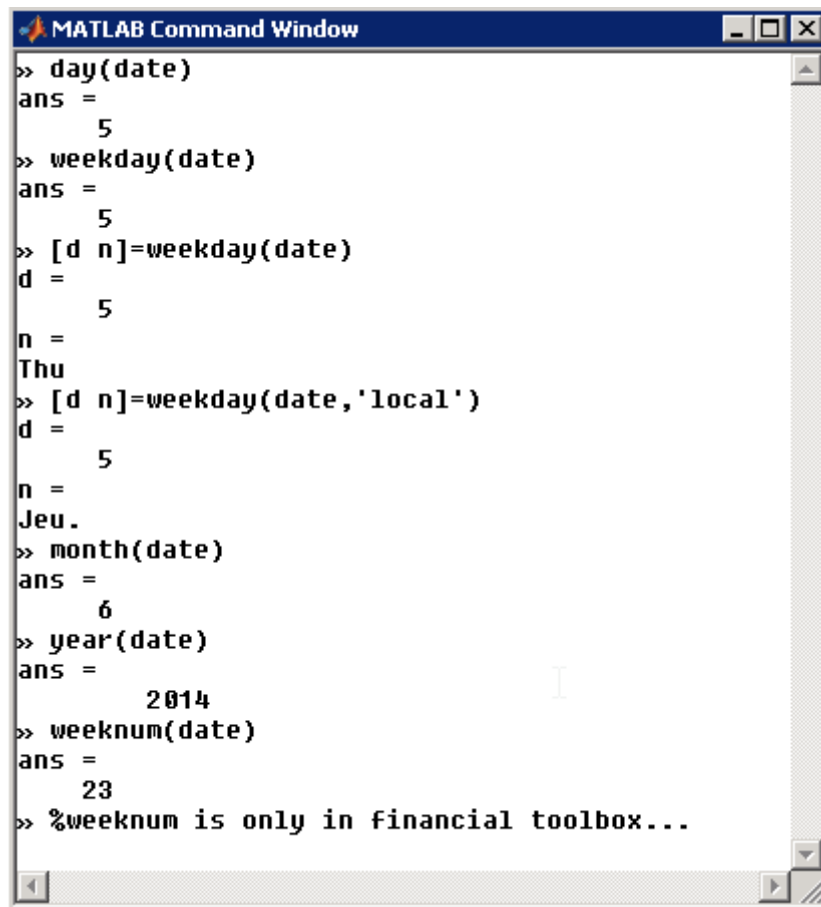
ans =
    2008     11     25         0         0         0
    2008     11     26         0         0         0
    2008     11     27         0         0         0
    2008     11     28         0         0         0

>> datenum(datevec(v,'yyyymmdd'))

ans =
    733737
    733738
    733739
    733740

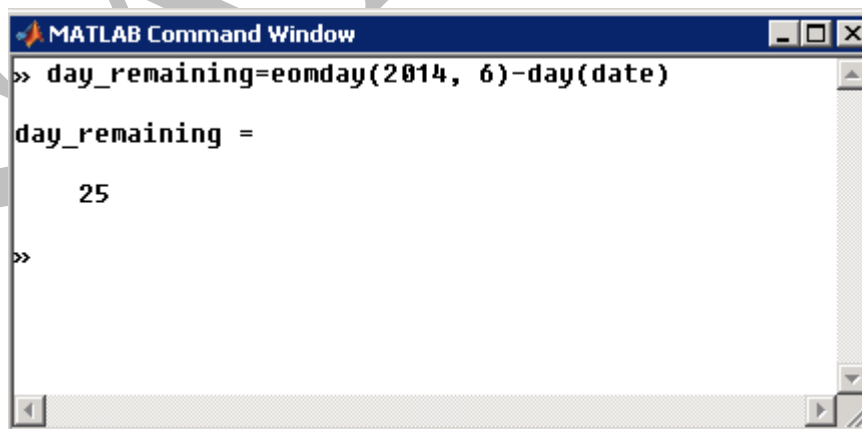
>>
```

Like in Microsoft Excel or any other spreadsheet software the core version of MATLAB has some elementary extraction dates functions like **day()**, **weekday()**, **month()** and **year()**:

A screenshot of the MATLAB Command Window. The window has a blue title bar with the MATLAB logo and the text "MATLAB Command Window". The command history shows several commands and their outputs. The first command is >> day(date), which returns ans = 5. The second command is >> weekday(date), which returns ans = 5. The third command is >> [d n]=weekday(date), which returns d = 5 and n = Thu. The fourth command is >> [d n]=weekday(date,'local'), which returns d = 5 and n = Jeu.. The fifth command is >> month(date), which returns ans = 6. The sixth command is >> year(date), which returns ans = 2014. The seventh command is >> weeknum(date), which returns ans = 23. The eighth command is >> %weeknum is only in financial toolbox...

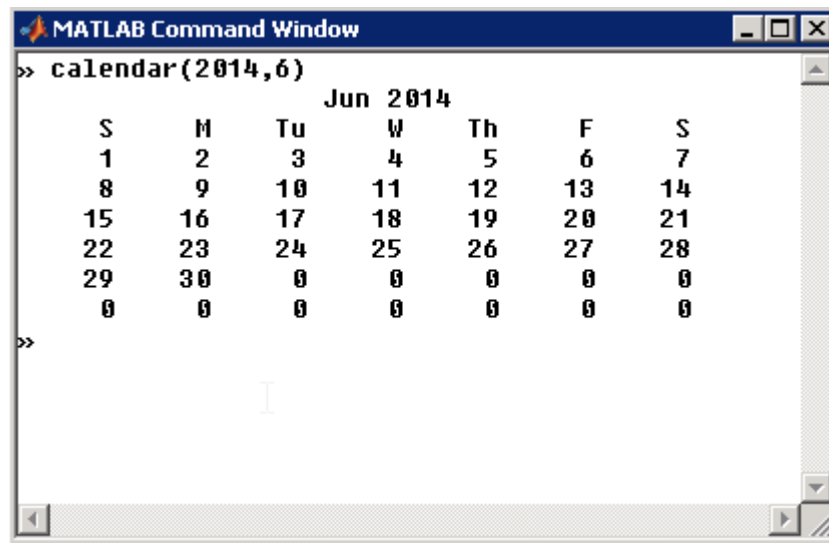
```
>> day(date)
ans =
     5
>> weekday(date)
ans =
     5
>> [d n]=weekday(date)
d =
     5
n =
Thu
>> [d n]=weekday(date,'local')
d =
     5
n =
Jeu.
>> month(date)
ans =
     6
>> year(date)
ans =
    2014
>> weeknum(date)
ans =
    23
>> %weeknum is only in financial toolbox...
```

Another typical calculation is to calculate the number of days remaining until the end of the month using the function **eomday()**:

A screenshot of the MATLAB Command Window. The window has a blue title bar with the MATLAB logo and the text "MATLAB Command Window". The command history shows a single command and its output. The command is >> day_remaining=eomday(2014, 6)-day(date), which returns day_remaining = 25. The prompt >> is shown on the next line.

```
>> day_remaining=eomday(2014, 6)-day(date)
day_remaining =
    25
>>
```

If you need it you can generate small textual calendar in the *Command Window* using the function **calendar()**:

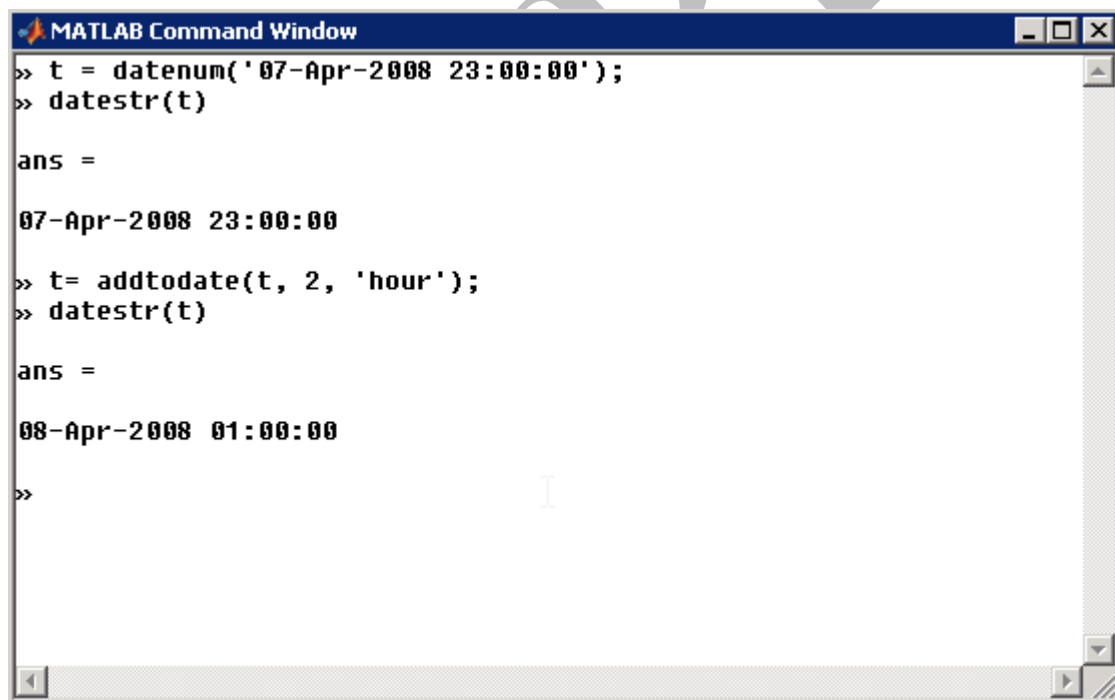


```
>> calendar(2014,6)
```

Jun 2014						
S	M	Tu	W	Th	F	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	0	0	0	0	0
0	0	0	0	0	0	0

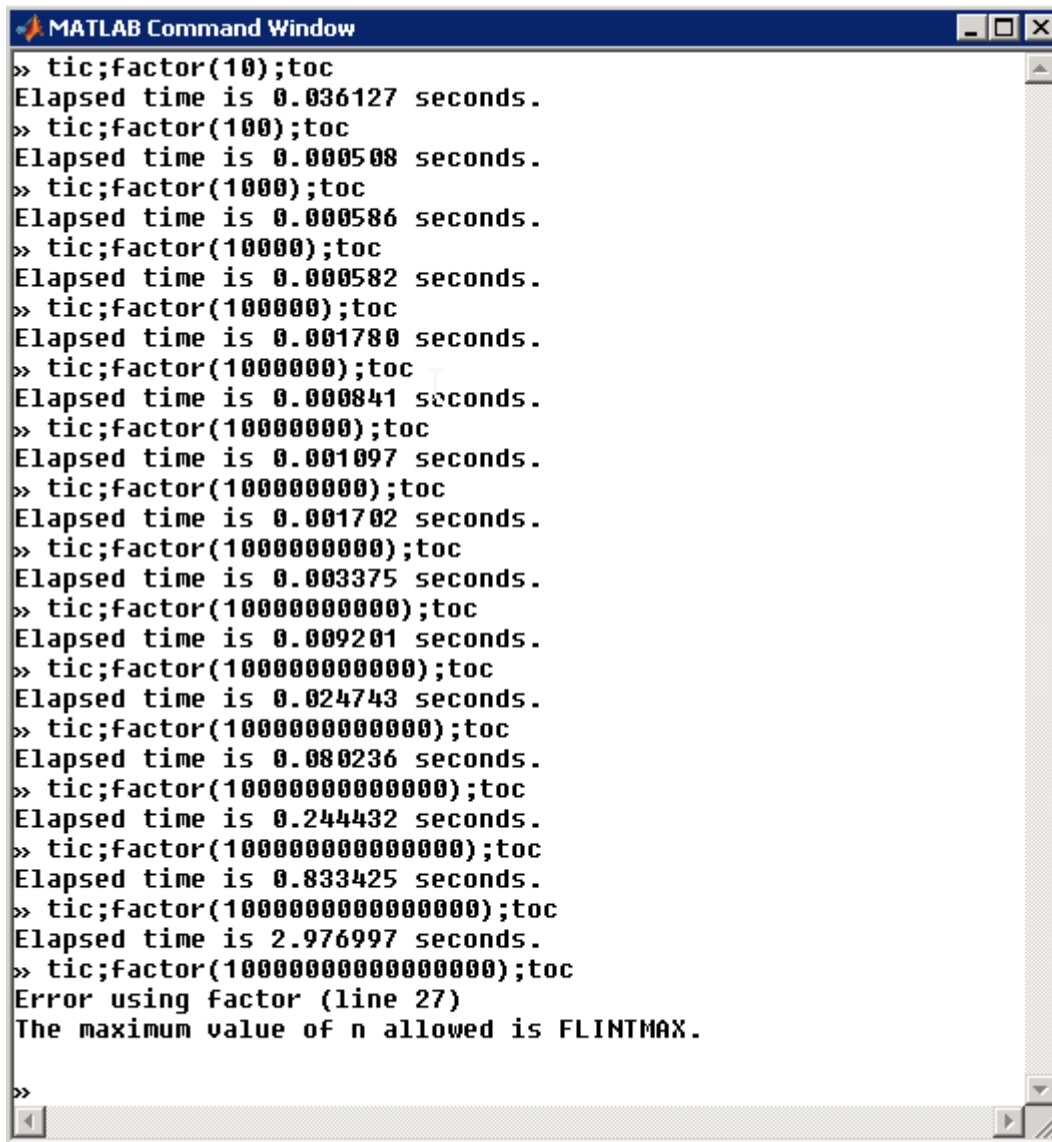
```
>>
```

There is also an important function to add a given time unit to a given date using the function **addtodate()**:



```
>> t = datenum('07-Apr-2008 23:00:00');  
>> datestr(t)  
  
ans =  
  
07-Apr-2008 23:00:00  
  
>> t= addtodate(t, 2, 'hour');  
>> datestr(t)  
  
ans =  
  
08-Apr-2008 01:00:00  
  
>>
```

A famous function when you want to test the efficiency of your code is to use the **tic** and **toc** commands:



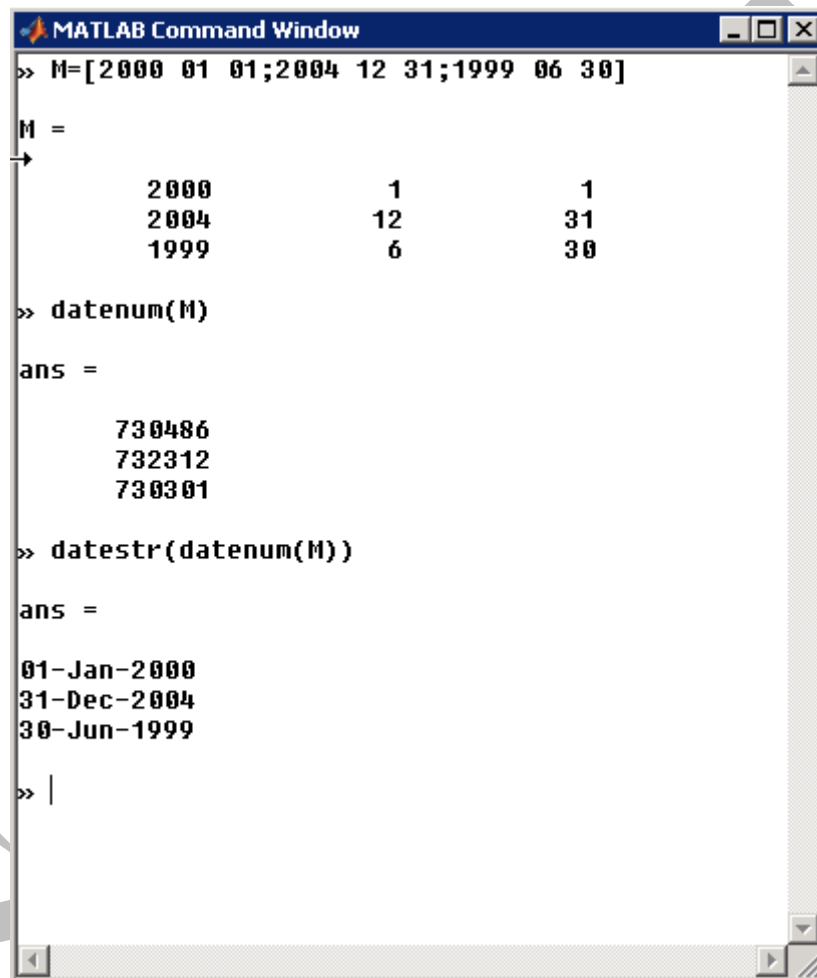
```
MATLAB Command Window
>> tic;factor(10);toc
Elapsed time is 0.036127 seconds.
>> tic;factor(100);toc
Elapsed time is 0.000508 seconds.
>> tic;factor(1000);toc
Elapsed time is 0.000586 seconds.
>> tic;factor(10000);toc
Elapsed time is 0.000582 seconds.
>> tic;factor(100000);toc
Elapsed time is 0.001780 seconds.
>> tic;factor(1000000);toc
Elapsed time is 0.000841 seconds.
>> tic;factor(10000000);toc
Elapsed time is 0.001097 seconds.
>> tic;factor(100000000);toc
Elapsed time is 0.001702 seconds.
>> tic;factor(1000000000);toc
Elapsed time is 0.003375 seconds.
>> tic;factor(10000000000);toc
Elapsed time is 0.009201 seconds.
>> tic;factor(100000000000);toc
Elapsed time is 0.024743 seconds.
>> tic;factor(1000000000000);toc
Elapsed time is 0.080236 seconds.
>> tic;factor(10000000000000);toc
Elapsed time is 0.244432 seconds.
>> tic;factor(100000000000000);toc
Elapsed time is 0.833425 seconds.
>> tic;factor(1000000000000000);toc
Elapsed time is 2.976997 seconds.
>> tic;factor(10000000000000000);toc
Error using factor (line 27)
The maximum value of n allowed is FLINTMAX.
>>
```

Important practical cases

This like a FAQ (frequently asked question) for which MATLAB doesn't seem to have built-in functions.

Convert year/month/day matrix as dates

Suppose we have a matrix with 3 columns, each column contains respectively years, months and days. To convert this into standard dates it is in fact very easy. We just have to use the function **datenum()** and **datestr()**:



```
MATLAB Command Window
>> M=[2000 01 01;2004 12 31;1999 06 30]

M =

    2000     1     1
    2004    12    31
    1999     6    30

>> datenum(M)

ans =

    730486
    732312
    730301

>> datestr(datenum(M))

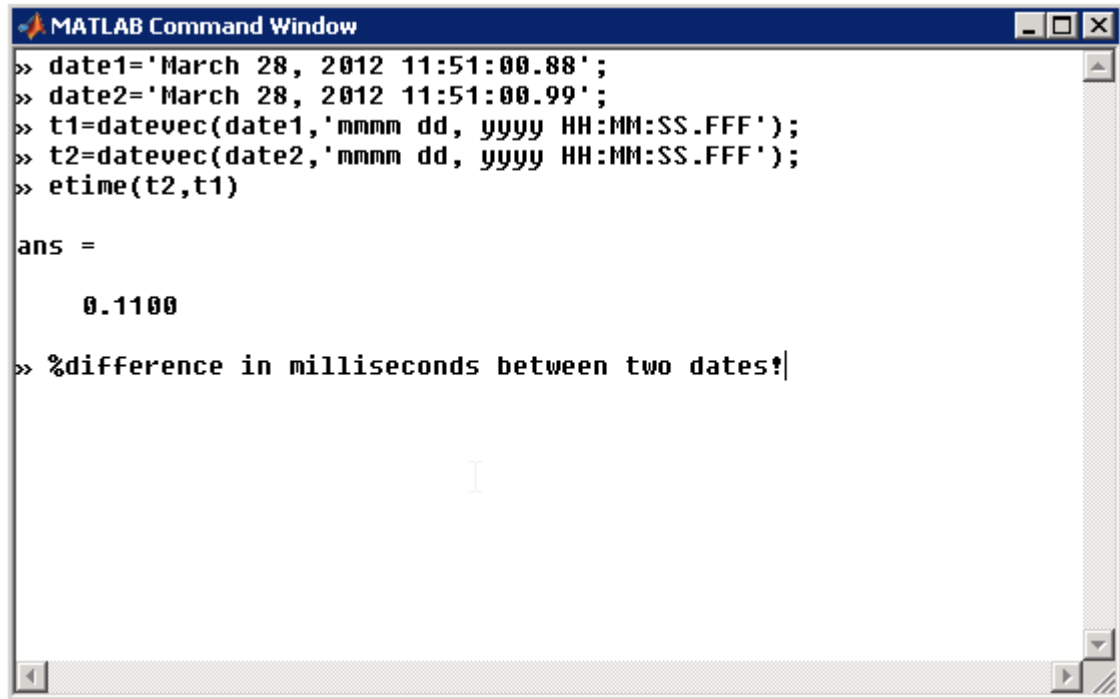
ans =

01-Jan-2000
31-Dec-2004
30-Jun-1999

>> |
```

Difference in milliseconds/seconds between 2 dates

In high frequency trading we need to calculate the number of milliseconds between two times. Supposing that we have two dates in a non-normalized format, we want to calculate the milliseconds between the both times. For this we will use the functions **etime()**:



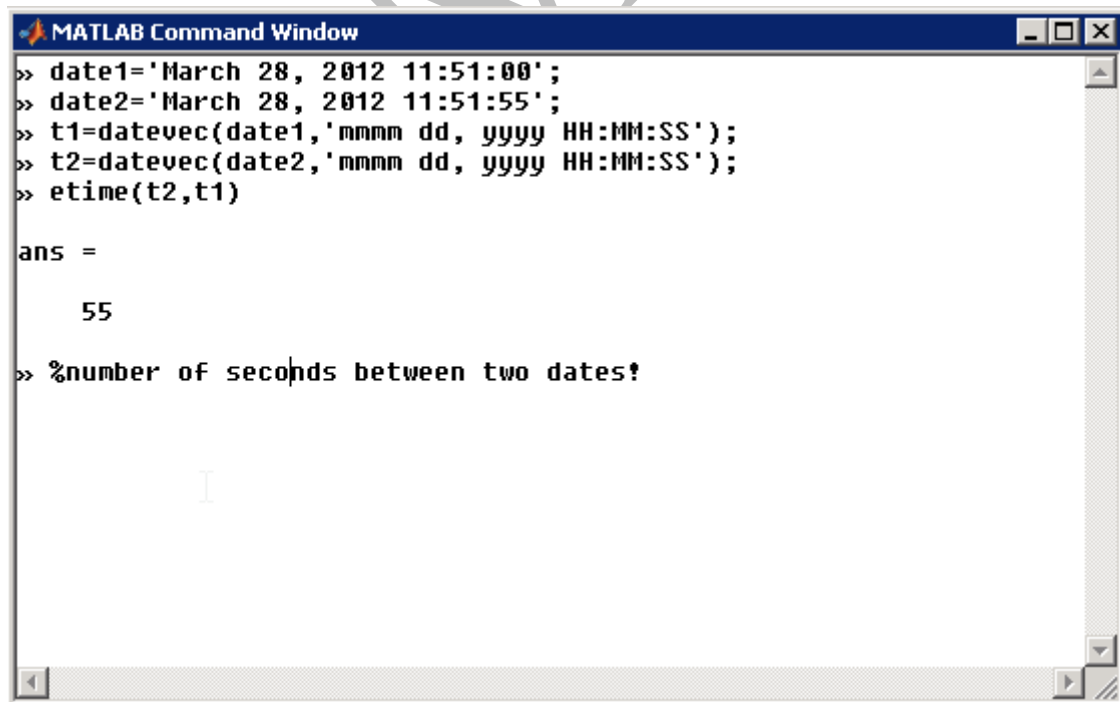
```
MATLAB Command Window
>> date1='March 28, 2012 11:51:00.88';
>> date2='March 28, 2012 11:51:00.99';
>> t1=datevec(date1,'mmm dd, yyyy HH:MM:SS.FFF');
>> t2=datevec(date2,'mmm dd, yyyy HH:MM:SS.FFF');
>> etime(t2,t1)

ans =

    0.1100

>> %difference in milliseconds between two dates!
```

Or if we are just interested to the number of seconds:



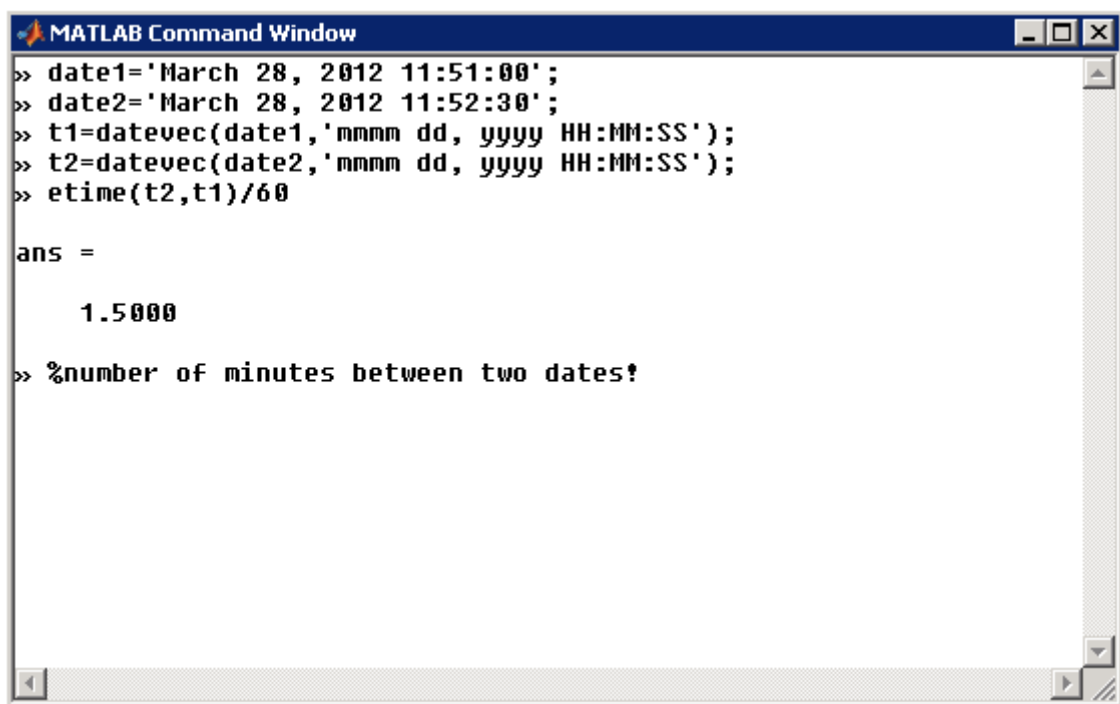
```
MATLAB Command Window
>> date1='March 28, 2012 11:51:00';
>> date2='March 28, 2012 11:51:55';
>> t1=datevec(date1,'mmm dd, yyyy HH:MM:SS');
>> t2=datevec(date2,'mmm dd, yyyy HH:MM:SS');
>> etime(t2,t1)

ans =

    55

>> %number of seconds between two dates!
```

or to the number of minutes by just dividing by 60...:

A screenshot of the MATLAB Command Window. The window has a blue title bar with the MATLAB logo and the text "MATLAB Command Window". The command history shows:

```
>> date1='March 28, 2012 11:51:00';  
>> date2='March 28, 2012 11:52:30';  
>> t1=datevec(date1,'mmm dd, yyyy HH:MM:SS');  
>> t2=datevec(date2,'mmm dd, yyyy HH:MM:SS');  
>> etime(t2,t1)/60
```

 The output is:

```
ans =  
  
    1.5000
```

 Below the output, there is a comment:

```
>> %number of minutes between two dates!
```

```

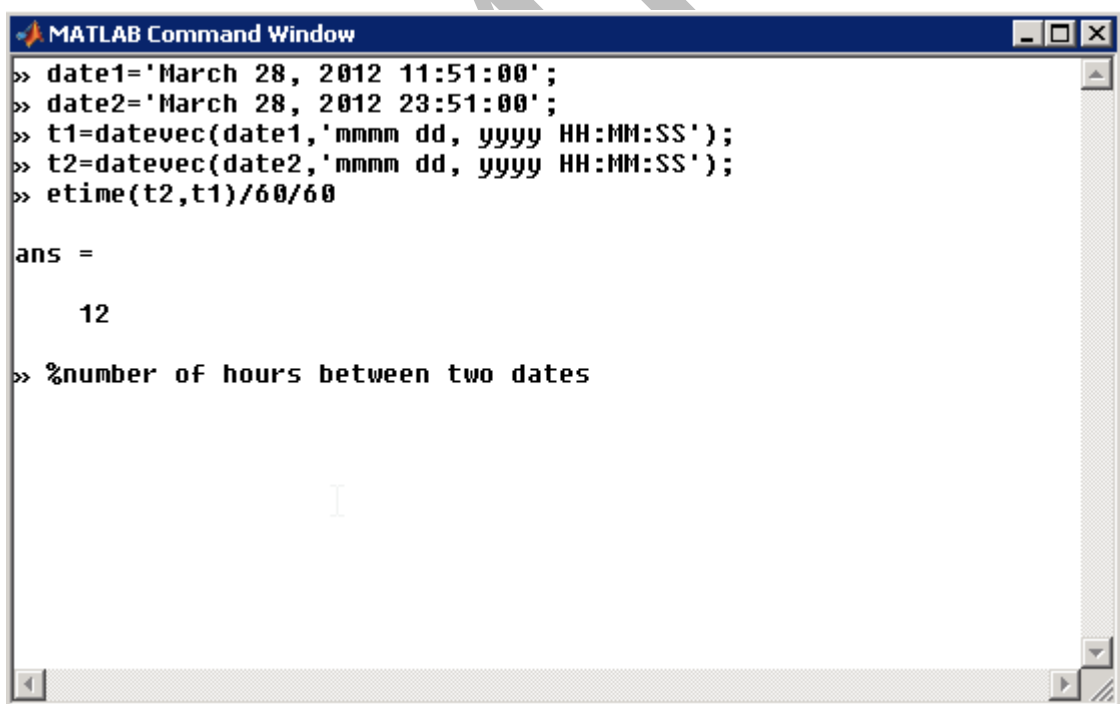
MATLAB Command Window
>> date1='March 28, 2012 11:51:00';
>> date2='March 28, 2012 11:52:30';
>> t1=datevec(date1,'mmm dd, yyyy HH:MM:SS');
>> t2=datevec(date2,'mmm dd, yyyy HH:MM:SS');
>> etime(t2,t1)/60

ans =

    1.5000

>> %number of minutes between two dates!
```

Or the number of hours by dividing once again by 60...:

A screenshot of the MATLAB Command Window. The window has a blue title bar with the MATLAB logo and the text "MATLAB Command Window". The command history shows:

```
>> date1='March 28, 2012 11:51:00';  
>> date2='March 28, 2012 23:51:00';  
>> t1=datevec(date1,'mmm dd, yyyy HH:MM:SS');  
>> t2=datevec(date2,'mmm dd, yyyy HH:MM:SS');  
>> etime(t2,t1)/60/60
```

 The output is:

```
ans =  
  
    12
```

 Below the output, there is a comment:

```
>> %number of hours between two dates
```

```

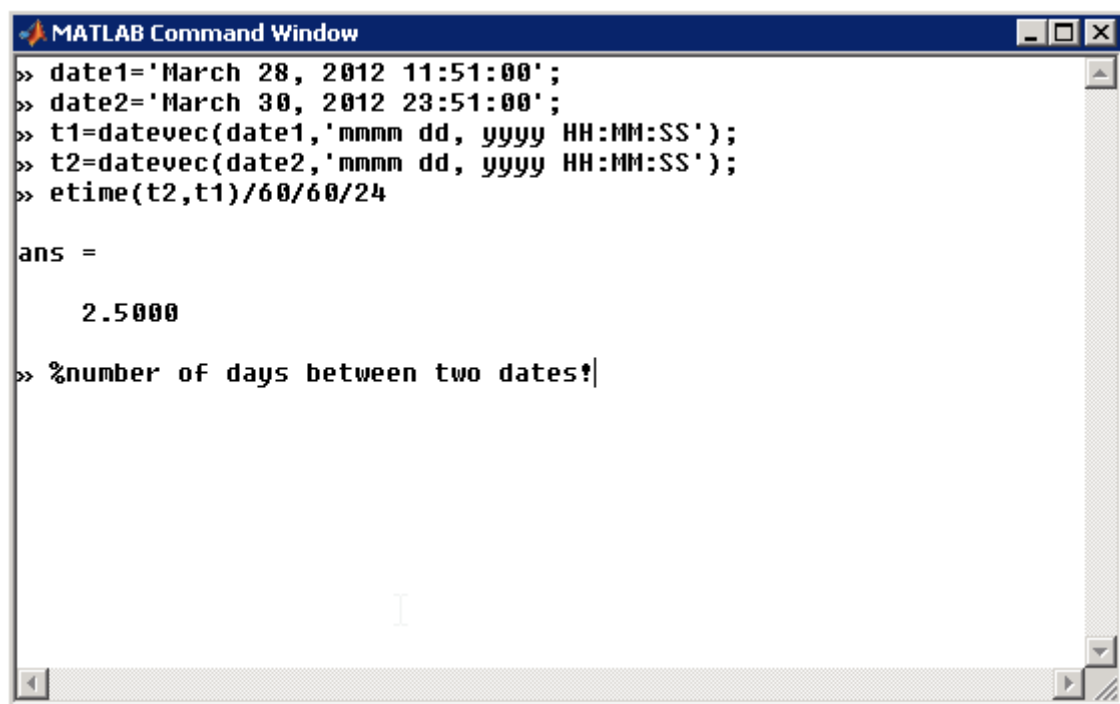
MATLAB Command Window
>> date1='March 28, 2012 11:51:00';
>> date2='March 28, 2012 23:51:00';
>> t1=datevec(date1,'mmm dd, yyyy HH:MM:SS');
>> t2=datevec(date2,'mmm dd, yyyy HH:MM:SS');
>> etime(t2,t1)/60/60

ans =

    12

>> %number of hours between two dates
```


Or the number of day by dividing by 24 the number of hours...:

A screenshot of the MATLAB Command Window. The window has a blue title bar with the MATLAB logo and the text "MATLAB Command Window". The command history shows the following code:

```
>> date1='March 28, 2012 11:51:00';  
>> date2='March 30, 2012 23:51:00';  
>> t1=datevec(date1,'mmm dd, yyyy HH:MM:SS');  
>> t2=datevec(date2,'mmm dd, yyyy HH:MM:SS');  
>> etime(t2,t1)/60/60/24
```

The output is

```
ans =  
  
    2.5000
```

Below the output, the prompt

```
>> %number of days between two dates!
```

 is visible. The window has standard Windows-style window controls (minimize, maximize, close) in the top right corner and a scroll bar on the right side.

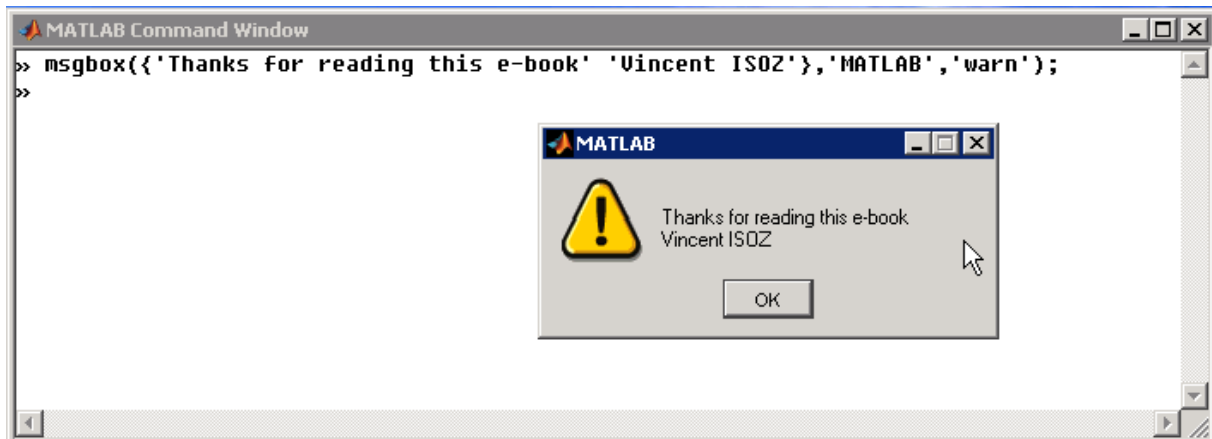
Now for months and years it begins to be a little bit more complicated without Toolbox or scripting...

Simple user interactions

Message boxes

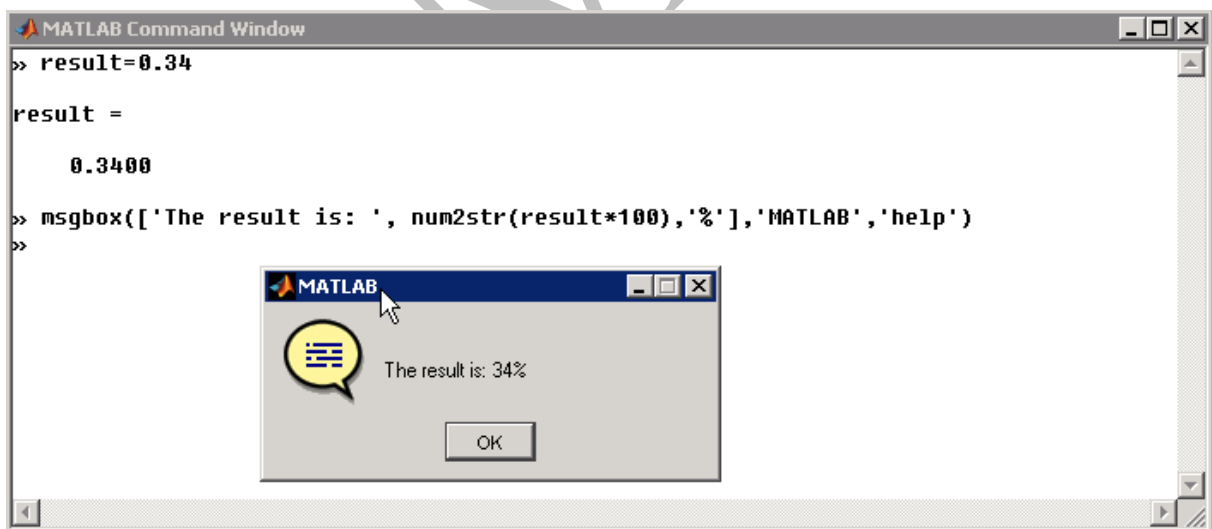
Later when we will learn to write scripts it will be necessary to interact with the user. The most common elementary function for this purpose is to use the function `msgbox()` that has three different logos (`warn`, `help`, `error`).

Here is a demo first with the warn logo:



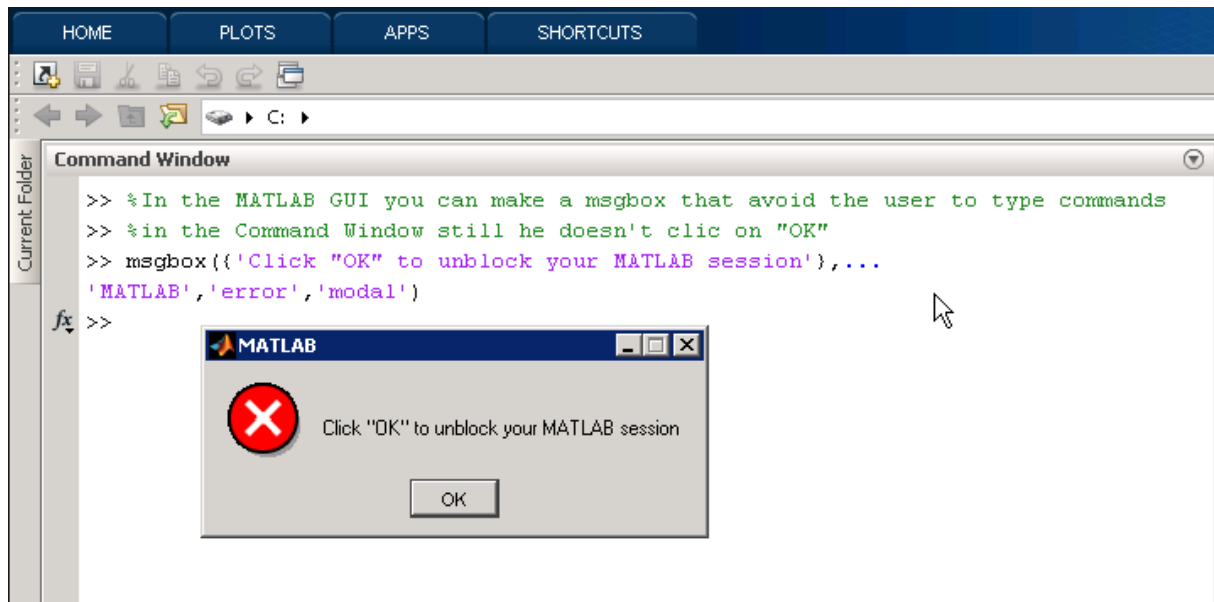
Remark: It is possible to customize the icon with an external image but at this step it is not a good idea because **the image won't follow** your script if you don't create a package (toolbox).

Now an example with the help logo and the result of a calculation:



Remark: You can't add nice equations in a message box without the Symbolic Toolbox...

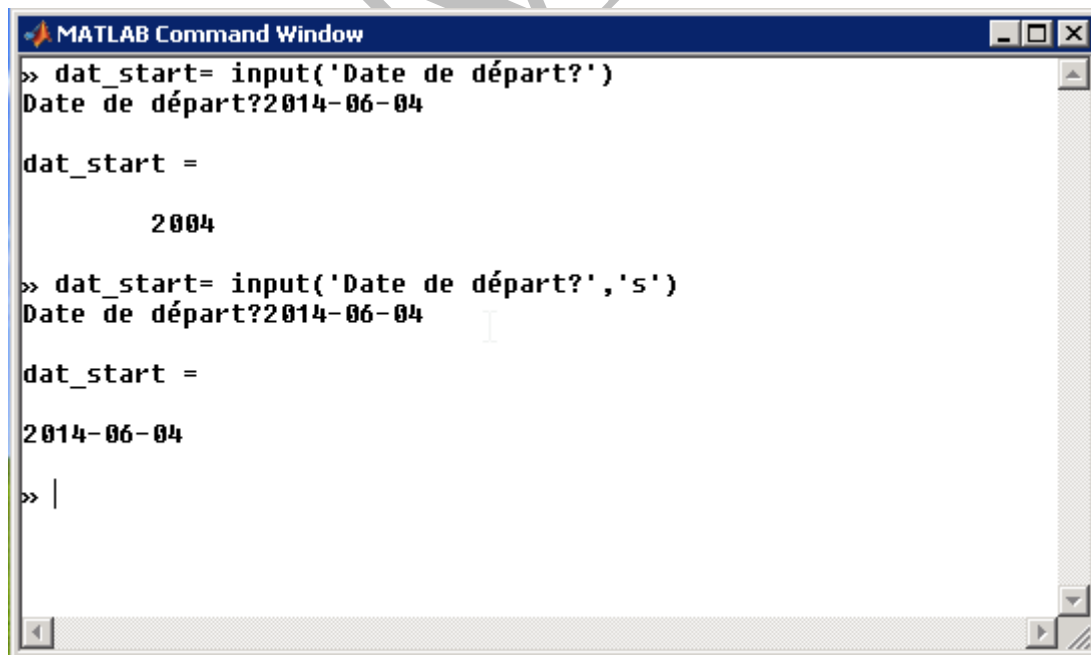
The user can still write behind the message box in the leight Command Window. To block the user make the message *modal* as following but **it works only in the MATLAB GUI...**:



All others user interactions that we will se below (inputdlg, listdlg, etc.) are "modal" in the MATLAB GUI by default!

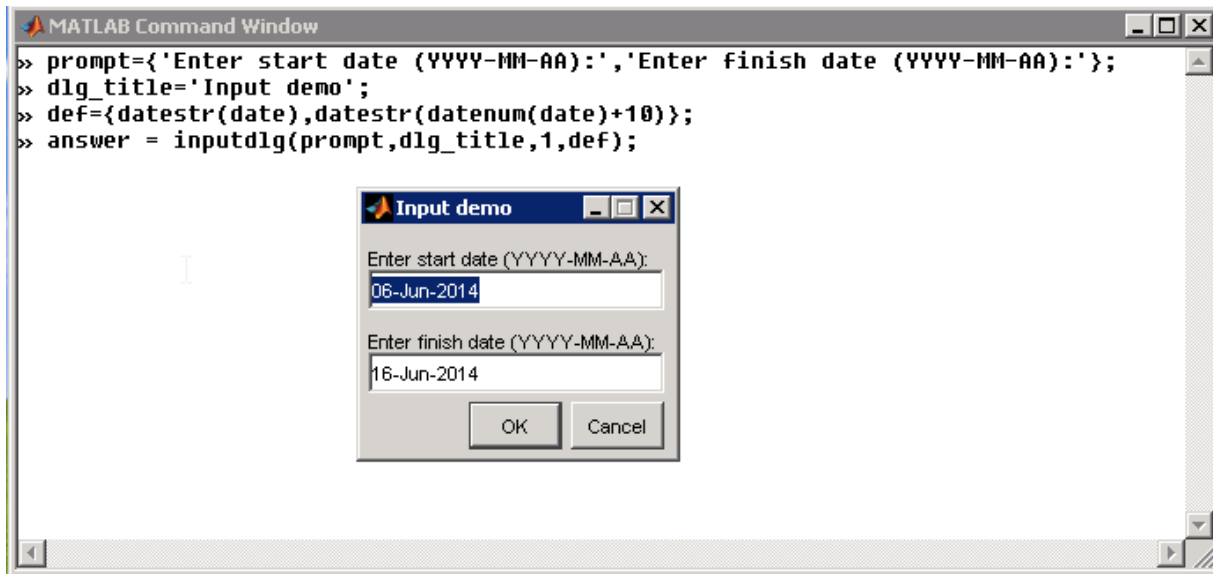
Input

Now let us see a very basic way to ask the user for an input using the Command Windows and the function **input()**. See also the trap specific to dates that you have to define as string input:

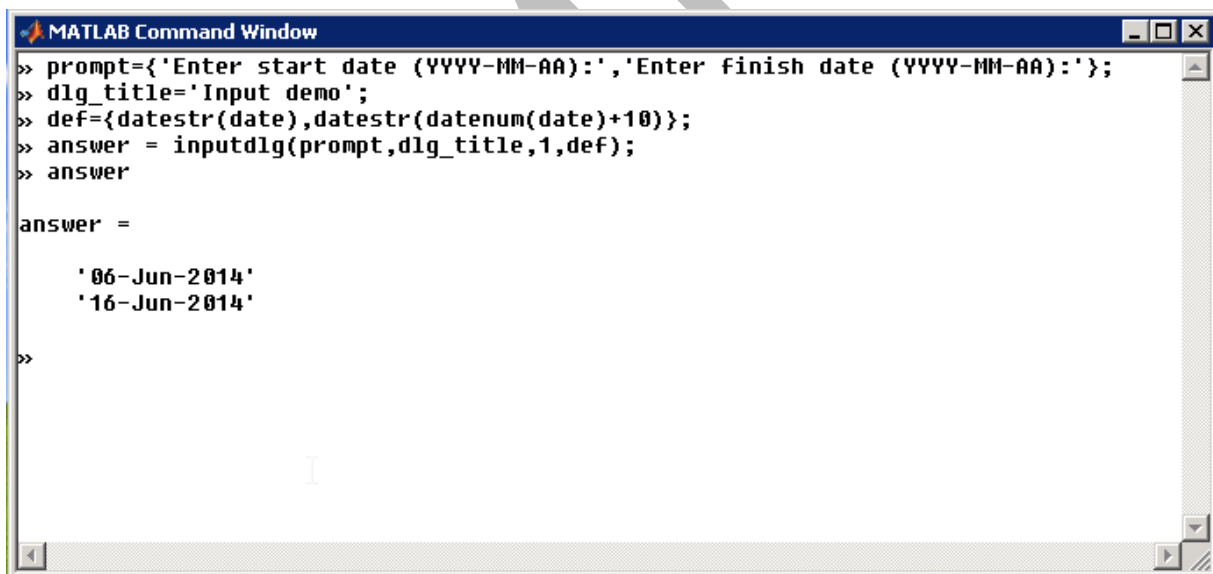


Input Dialog

The previous `input()` function needs the *Command Windows* to be open. This is not a good thing. A much more interesting alternative is to use the `inputdlg()` function:



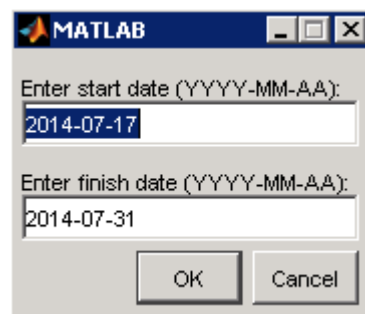
And here you can see the result of the input:



But it's still not right (see the dates! the format does not correspond). We can correct this using our knowledge on dates functions:

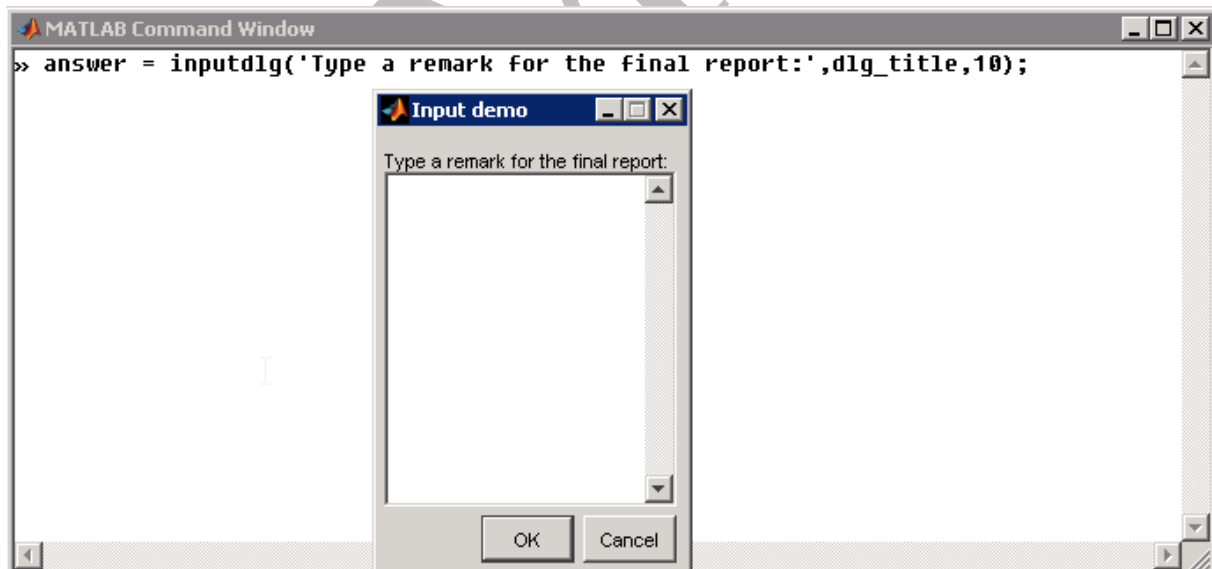
```
1 prompt={'Enter start date (YYYY-MM-AA):','Enter finish date (YYYY-MM-AA):'};  
2 def={datestr(date,'yyyy-mm-dd'),...  
3     datestr(addtodate(now,eomday(year(date),month(date))-day(date),'day'),'yyyy-mm-dd')};  
4 answer=inputdlg(prompt,'MATLAB',1,def);  
5
```

Then we get:

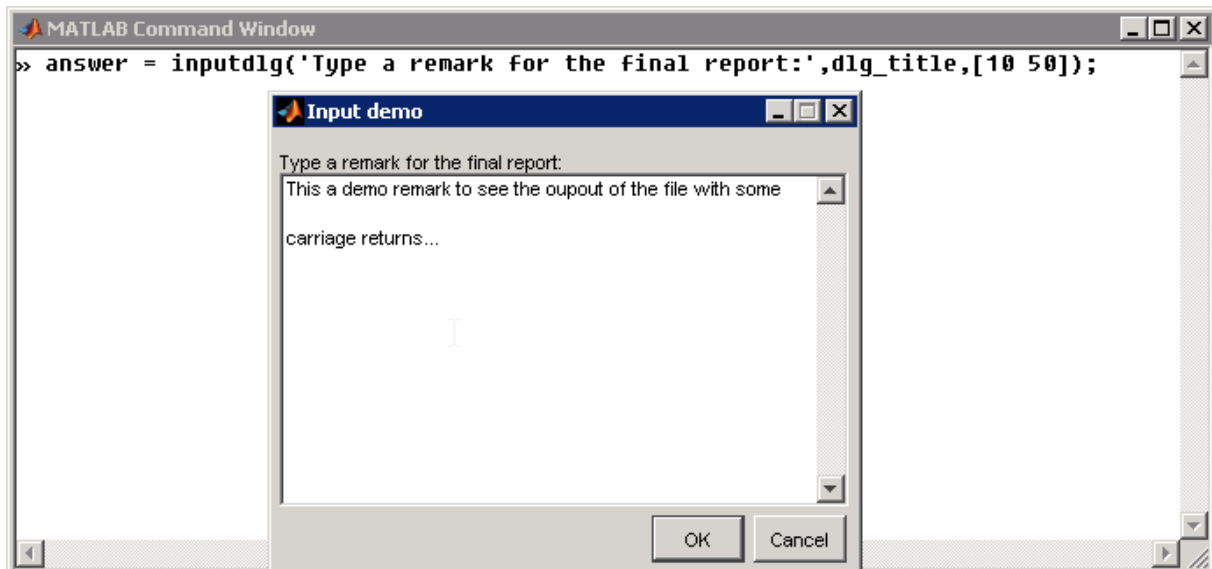


That's it!

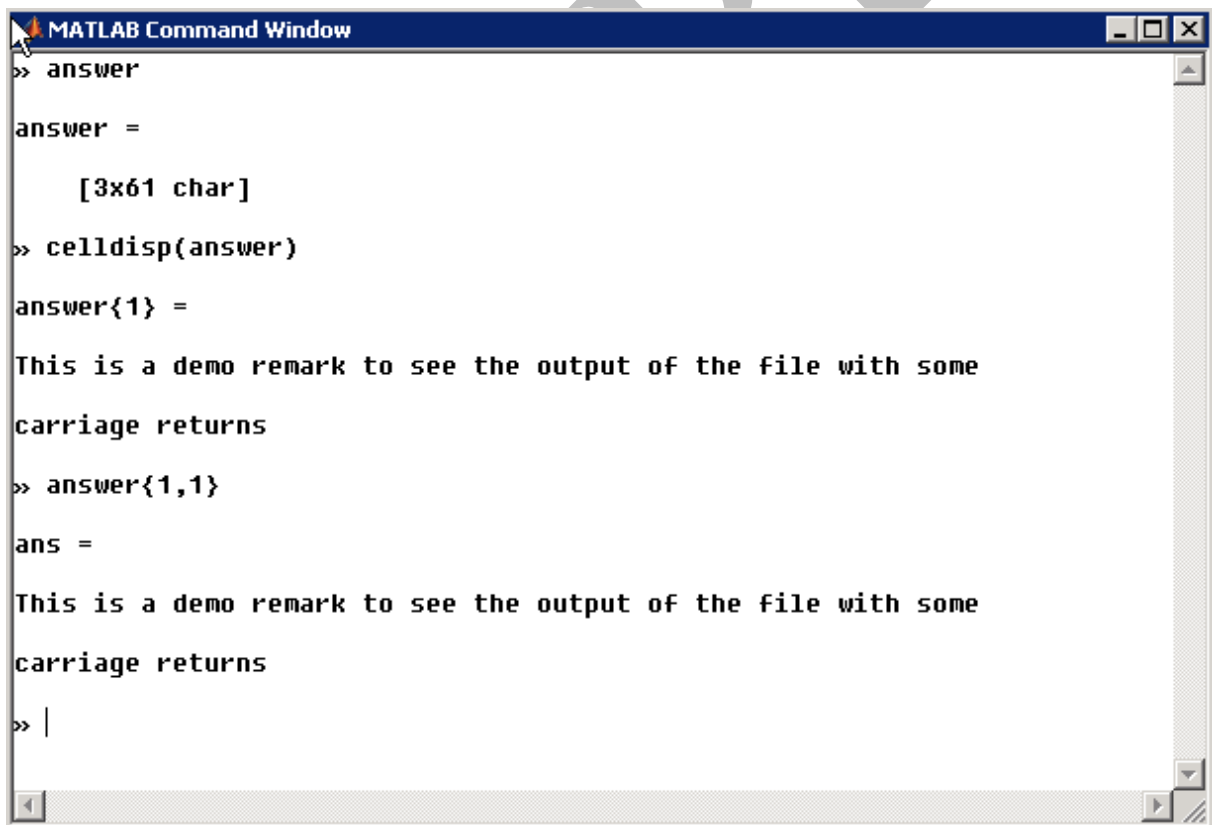
Let us see what is the purpose of the last argument of the **inputdlg()** function:



Or we can also control the width of the inputdlg field:

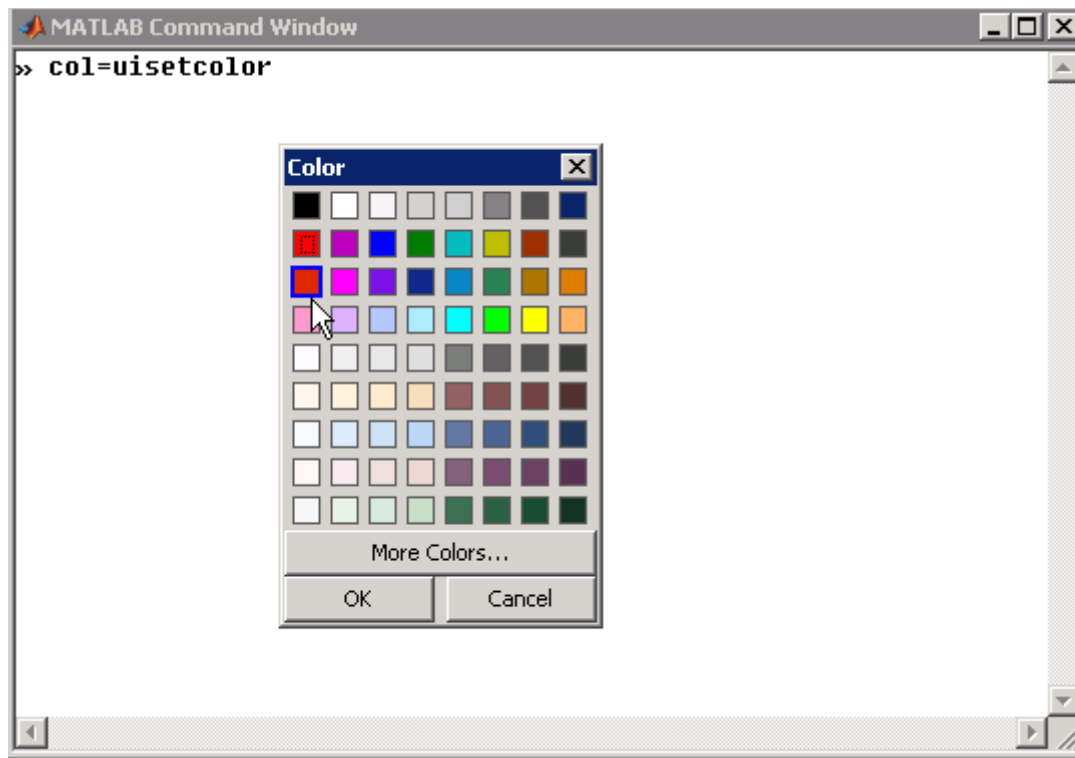


As you can see the input written by the user will be in a cell array we can see the content of this cell array by using a classic technic or by using the **celldisp()** function:

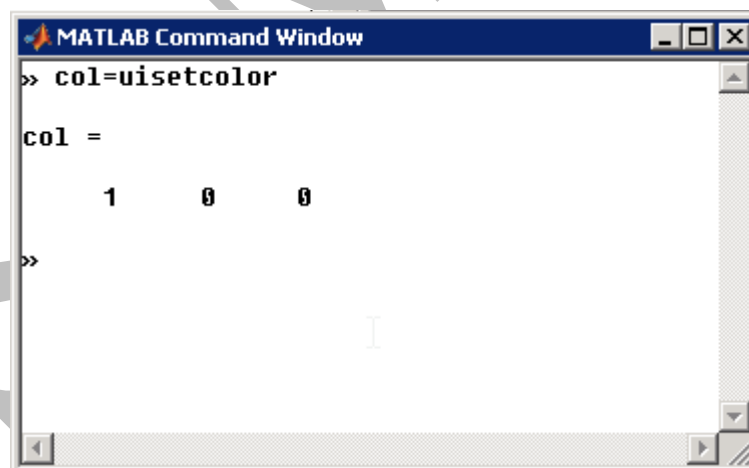


UI set color

Another very useful user interaction is the **uisetcolor** command that gives the possibility to the user to choose a color when you build automated charts:



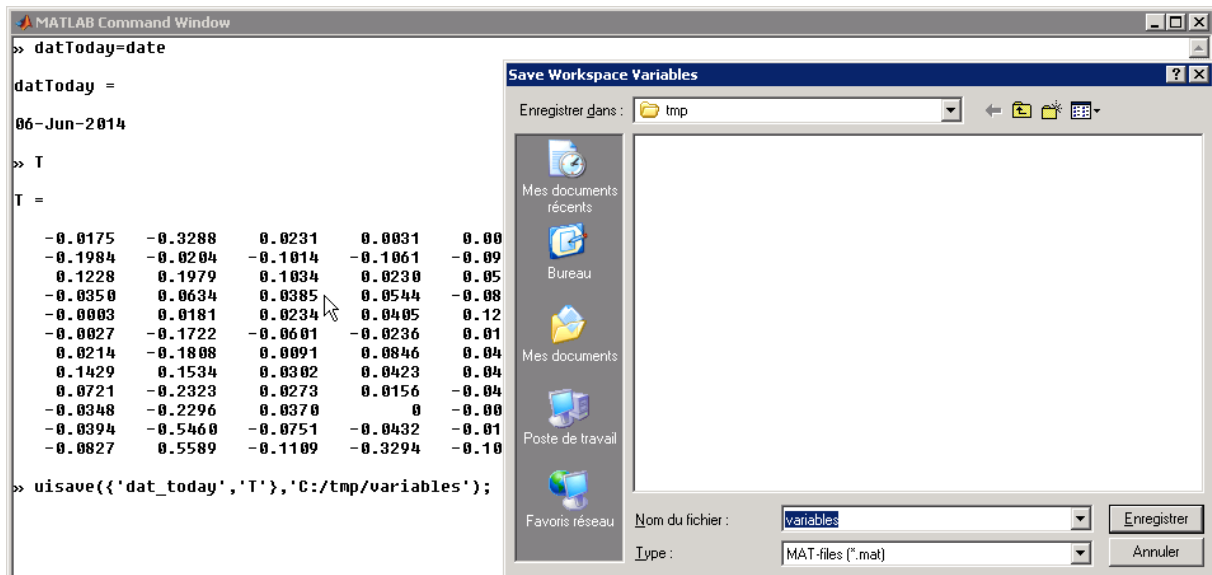
When the user choose a color the result will be the following:



UI save

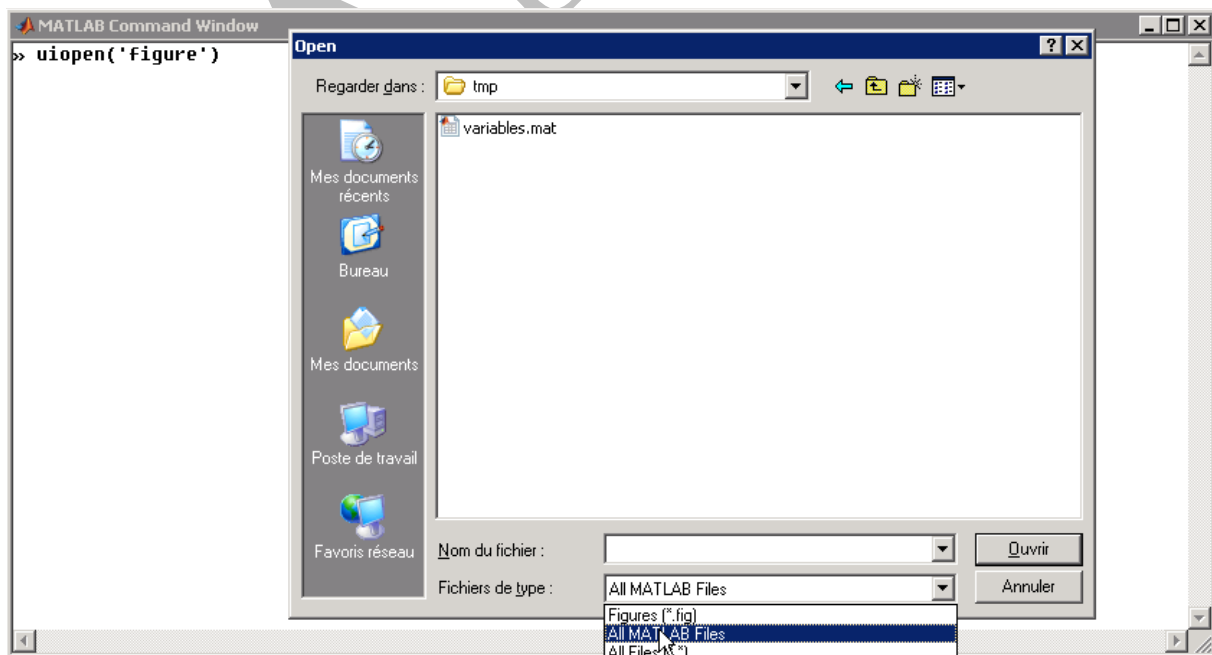
As for the previous interactions options you will also need with some scripts to save all variables of the current MATLAB session in a *.mat file.

For this we can use the **uisave()** function:

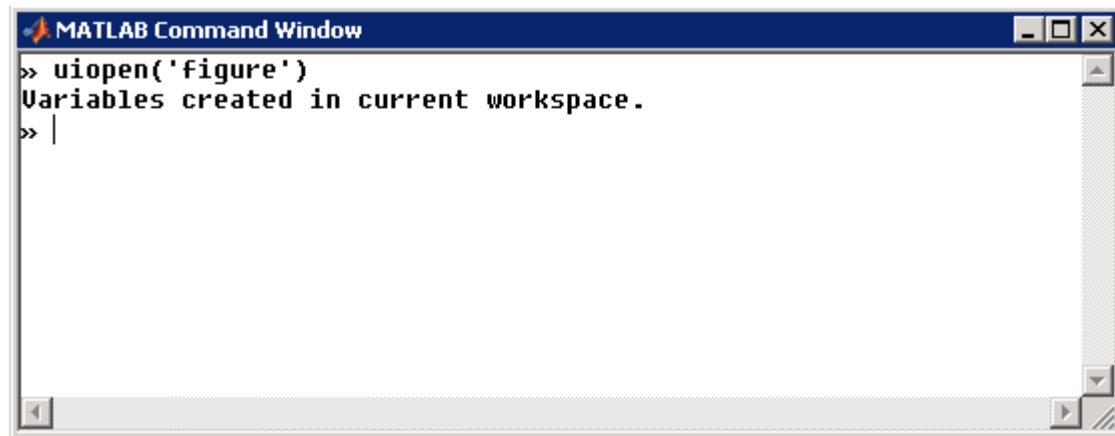


UI open

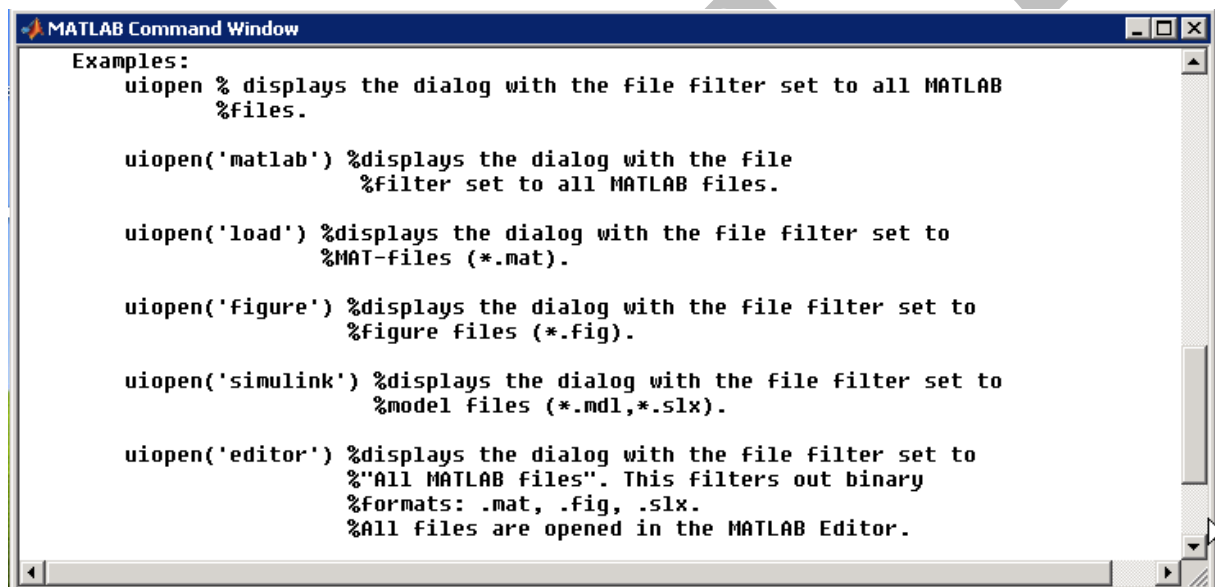
It can also be sometimes useful to ask the user interactively which *.mat or *.fig he wants to open with the command **uiopen()**. The 'figure' doesn't mean we want the user to be able to open only figures but just that MATLAB show us the dialog window to open files with by default the *.fig activated:



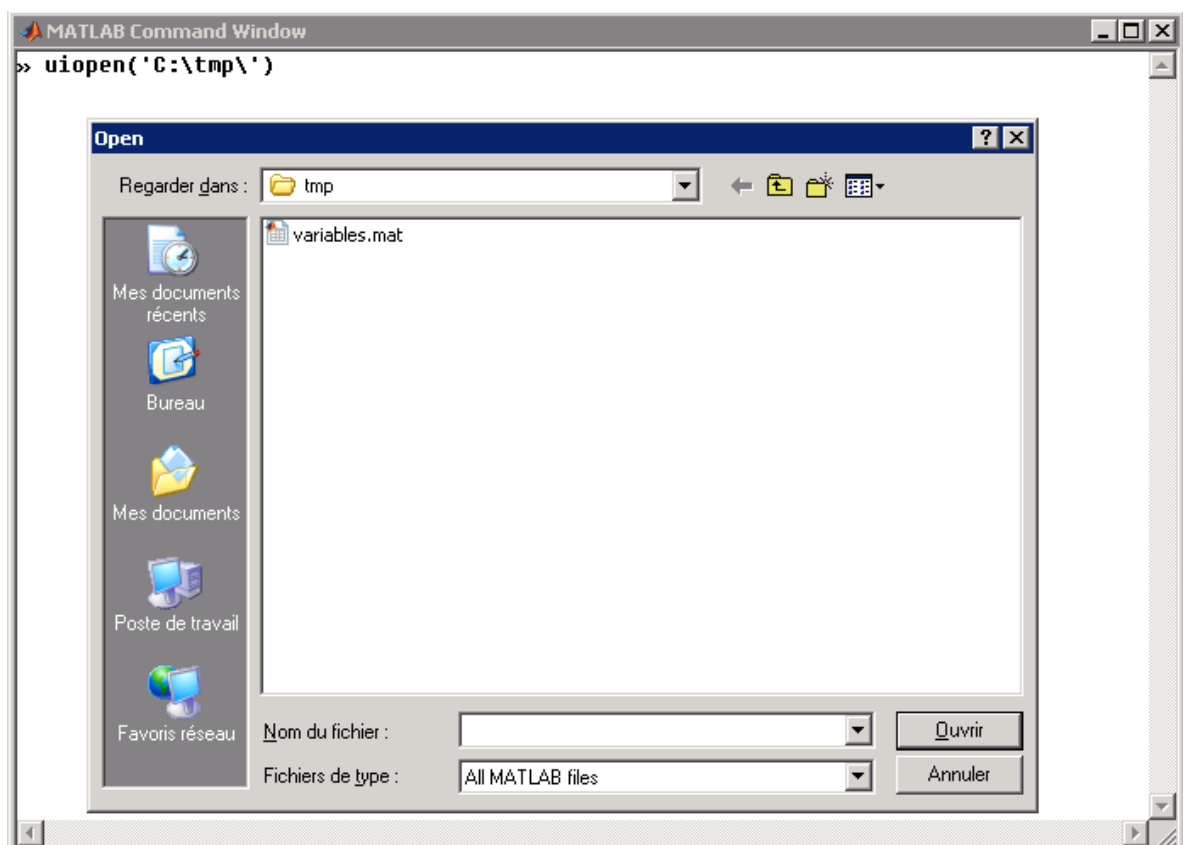
If we select a *.mat file instead of a *.fig it will work:



The **uiopen()** function has also other arguments to filter the visible filetypes. As you can see in the help of the command:

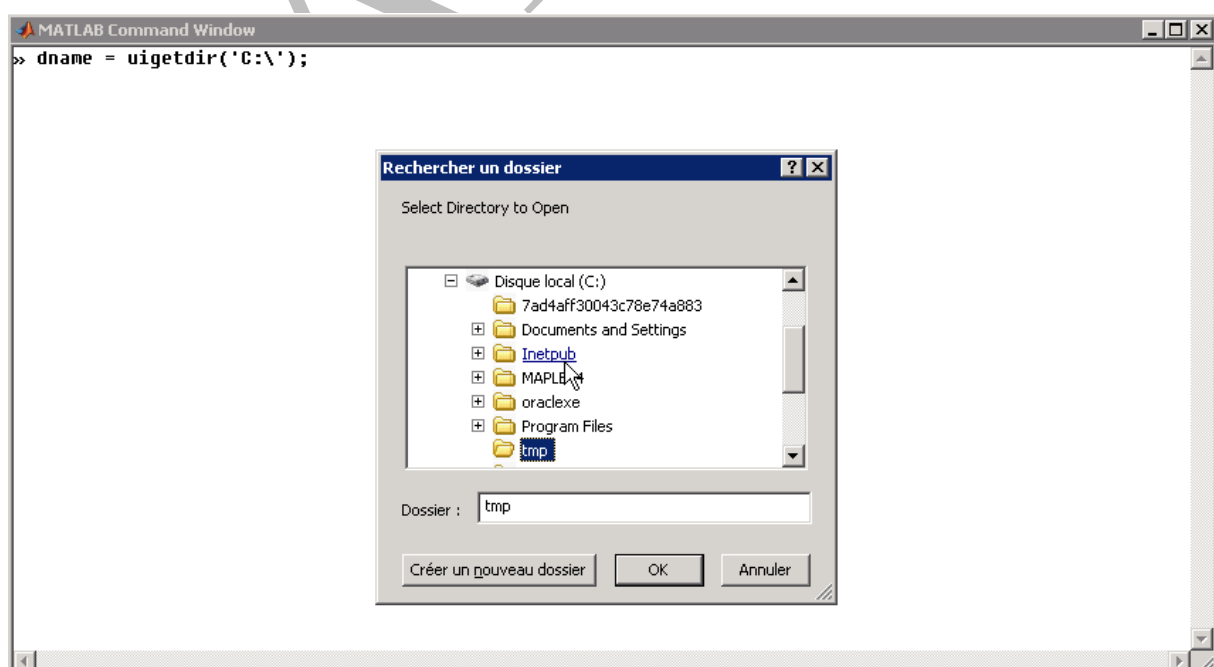


We can also force the open dialog window to show a given folder by default:



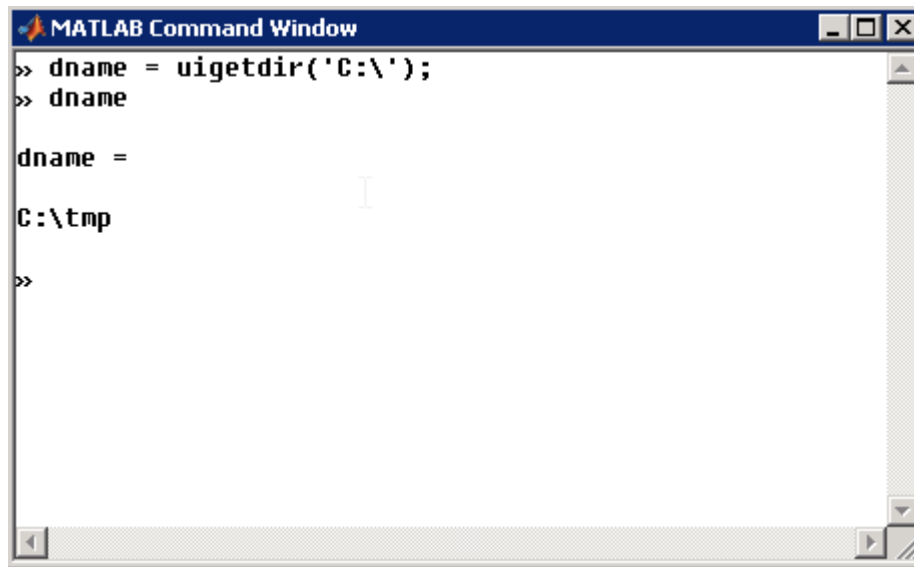
UI get dir

The need to ask the user a target director with the function **uigetdir()** is something very common when you develop MATLAB applications to ask the user where he wants to store the reports:



SAMPLE

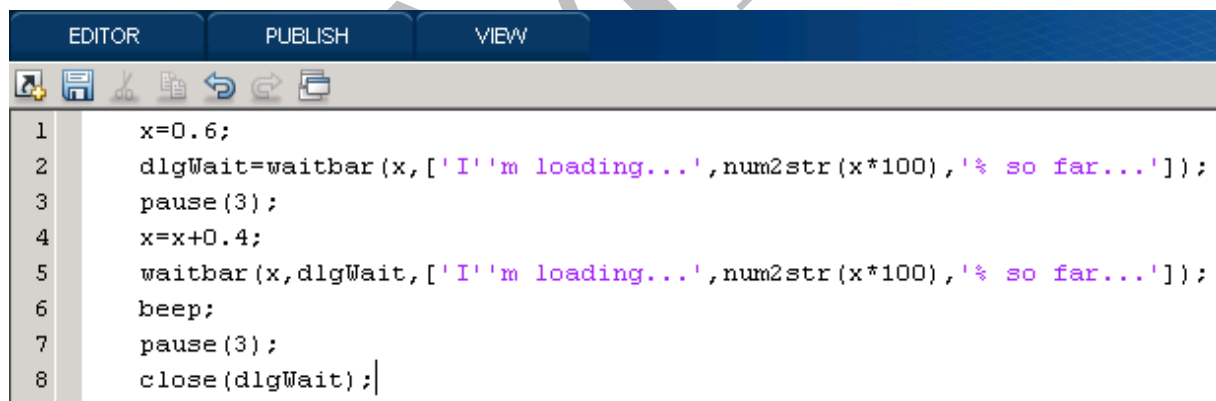
As you can see the result will be a simple string:



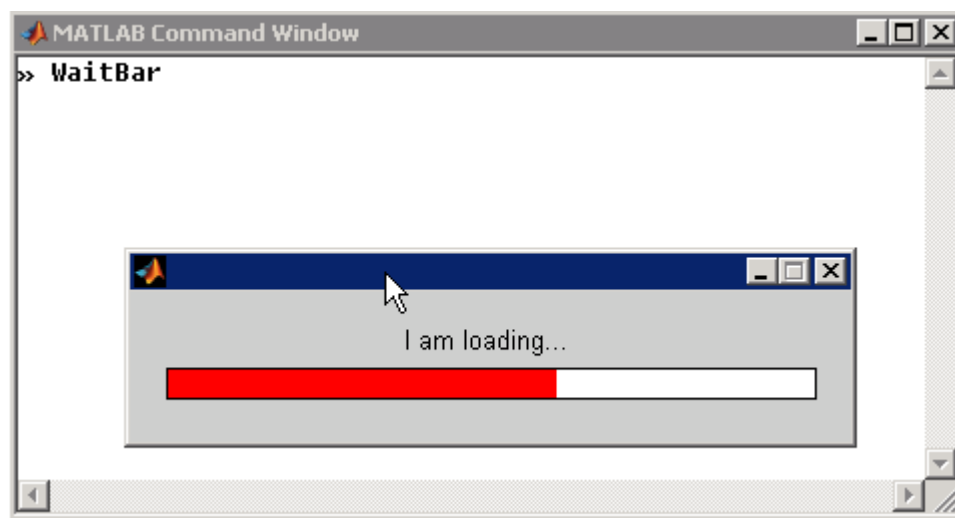
Wait bar

A functionality that is requested a lot by MATLAB users but slows down scripts a lot is to show a wait bar to the user of your application.

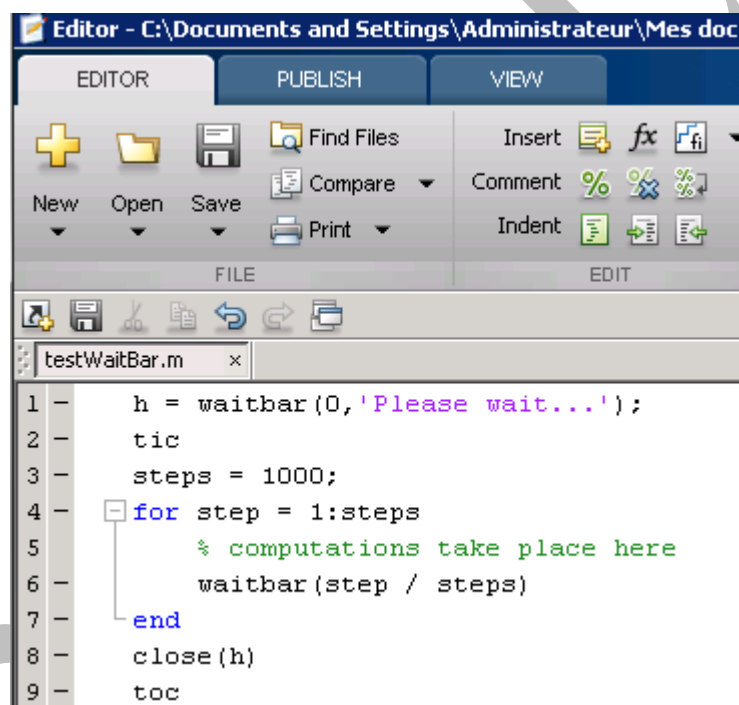
Below you can see a very simple stupid usage of the **waitbar()** function that starts at 60% then after 3 seconds jumps to 60%+40% and does a **beep** and after close:



The waitbar will look like this:

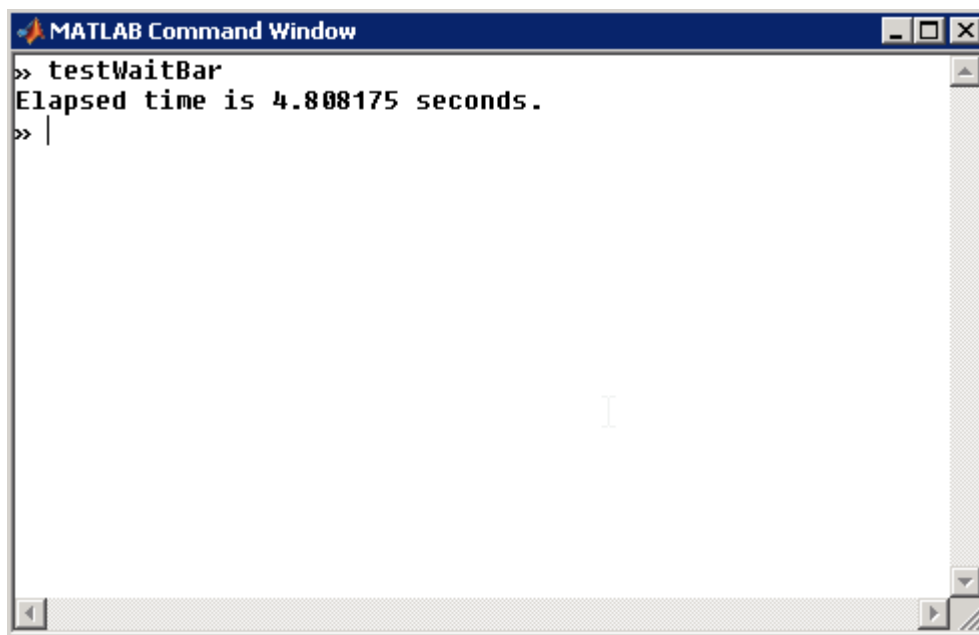


And here is a more real typical example of the use of the wait bar in a loop:

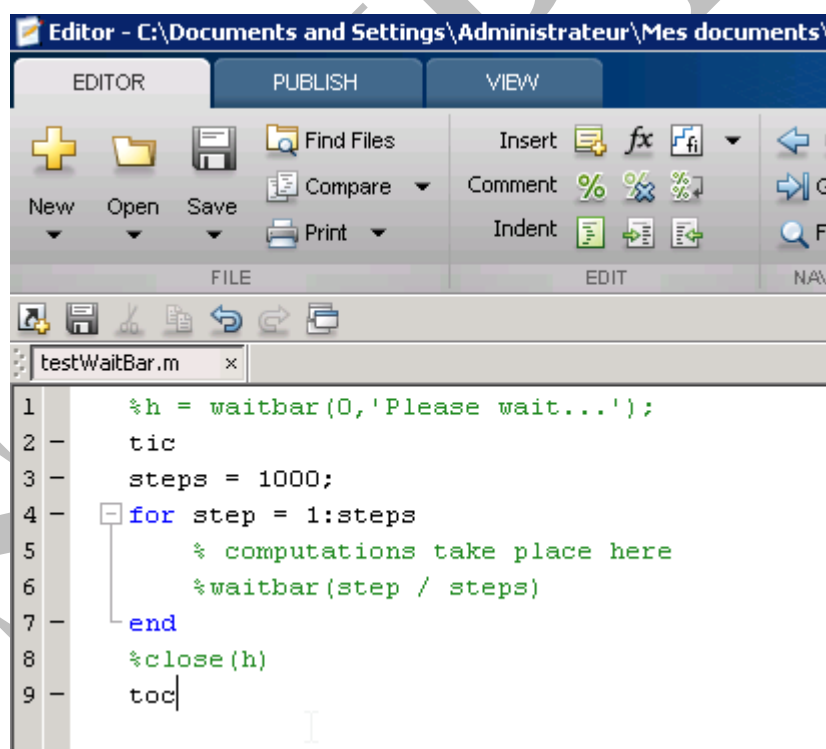
A screenshot of the MATLAB Editor window. The window title is "Editor - C:\Documents and Settings\Administrateur\Mes documents". The editor shows a script named "testWaitBar.m". The script contains the following code:

```
1 - h = waitbar(0, 'Please wait...');
2 - tic
3 - steps = 1000;
4 - for step = 1:steps
5 -     % computations take place here
6 -     waitbar(step / steps)
7 - end
8 - close(h)
9 - toc
```

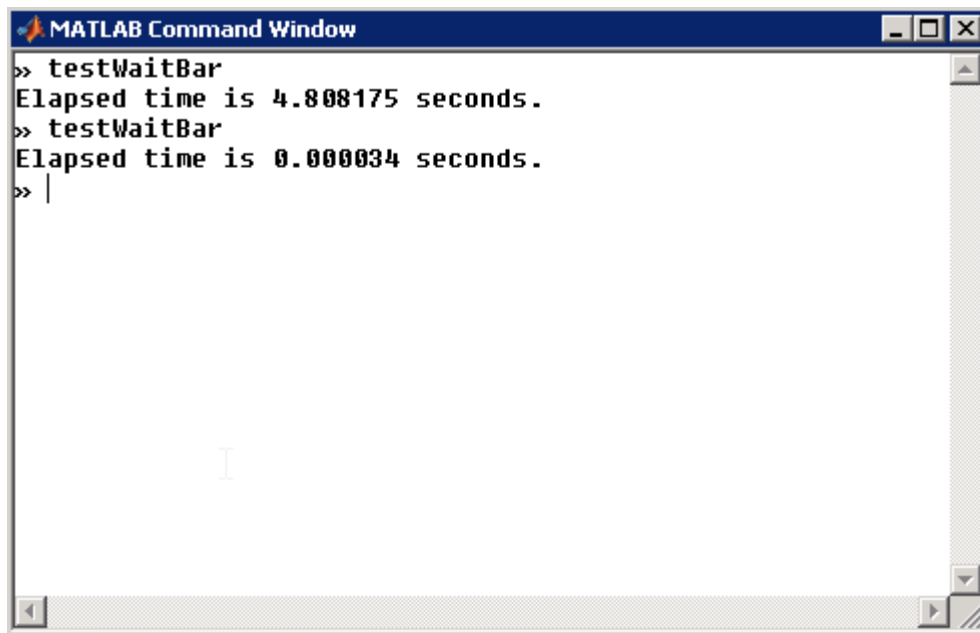
And here you can see the execution time with a wait bar:



And now let us deactivate the wait bar:



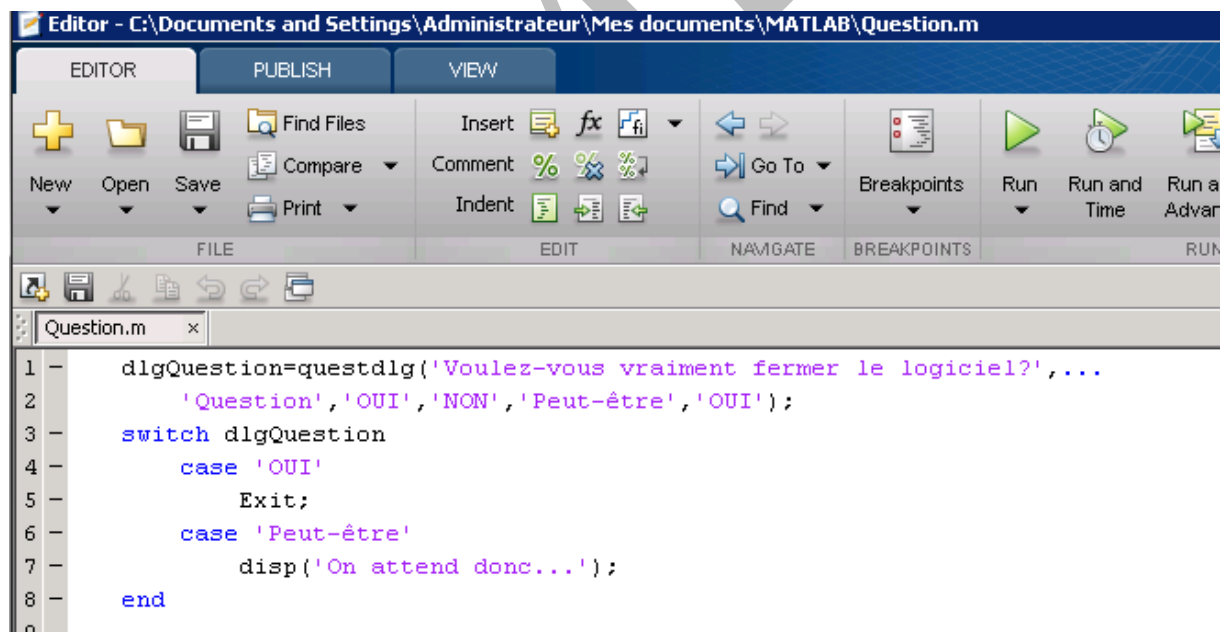
and see the execution time...:



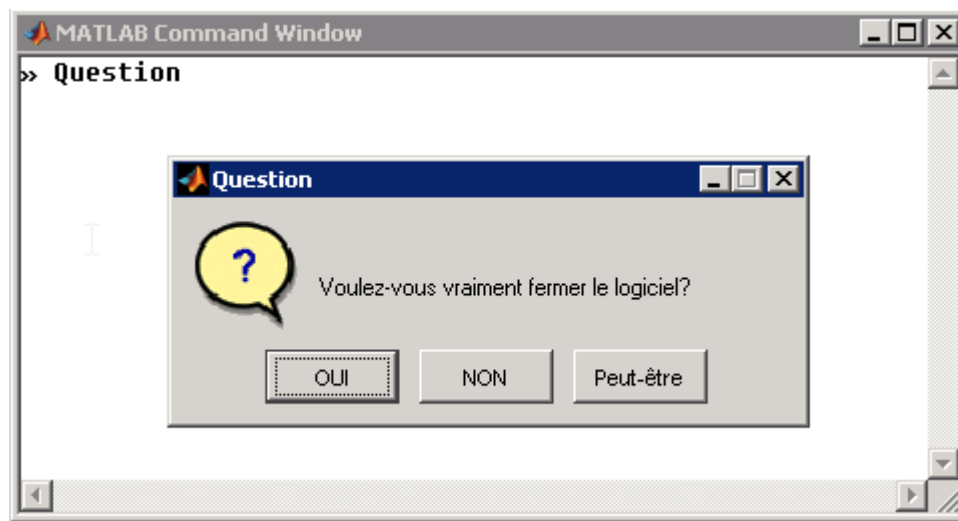
no comment...

Question dialog

Another typical dialog is to ask a question to the user and retrieve its answer. This can easily be done thanks to the function `questdlg()`:



This will result in:



and the answer will be automatically managed by the *switch... case....* structure!

Menu

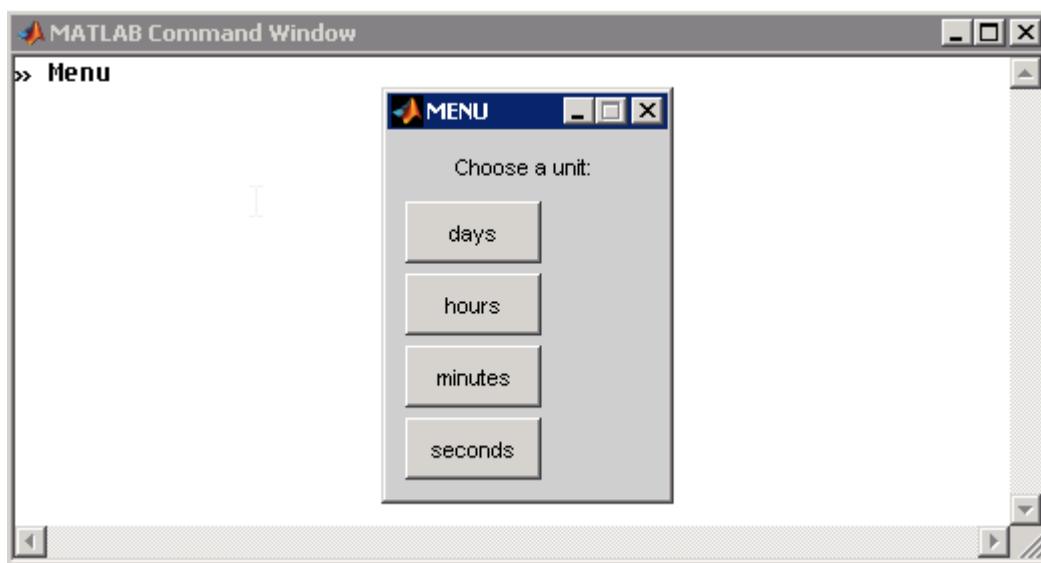
When you make charts it is many times useful to ask the users the unit scale he wants to see for the axes or the export format for automated reports. For this purpose one possibility is to create a menu using the function... **menu()**:

```

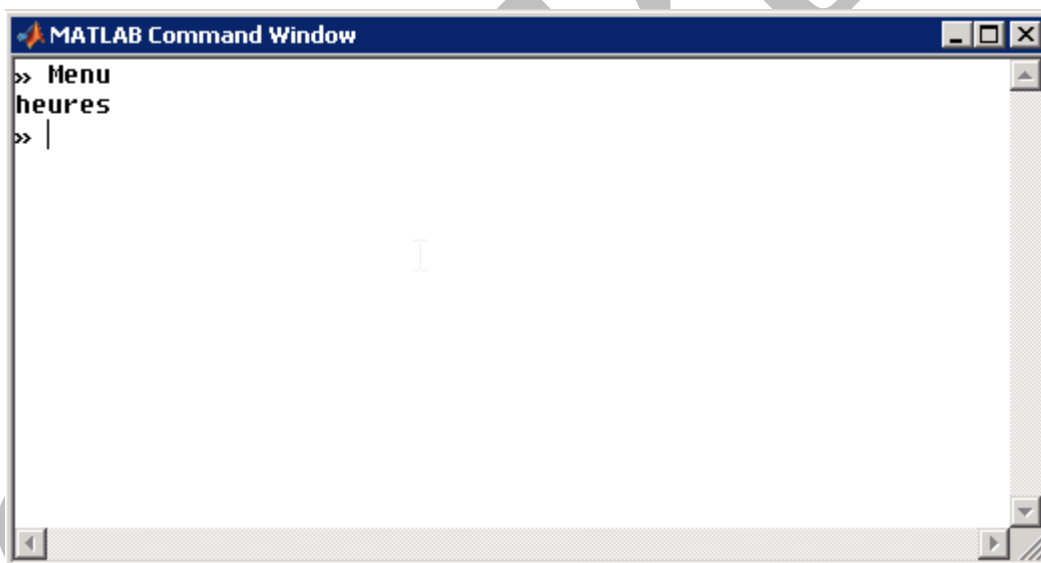
1 -   dlgMenu=menu('Choose a unit: ','days','hours','minutes','seconds');
2 -   switch dlgMenu
3 -       case 1
4 -           disp('Jours')
5 -       case 2
6 -           disp('heures')
7 -       case 3
8 -           disp('minutes')
9 -       case 4
10 -          disp('secondes')
11 -   end

```


This will give:

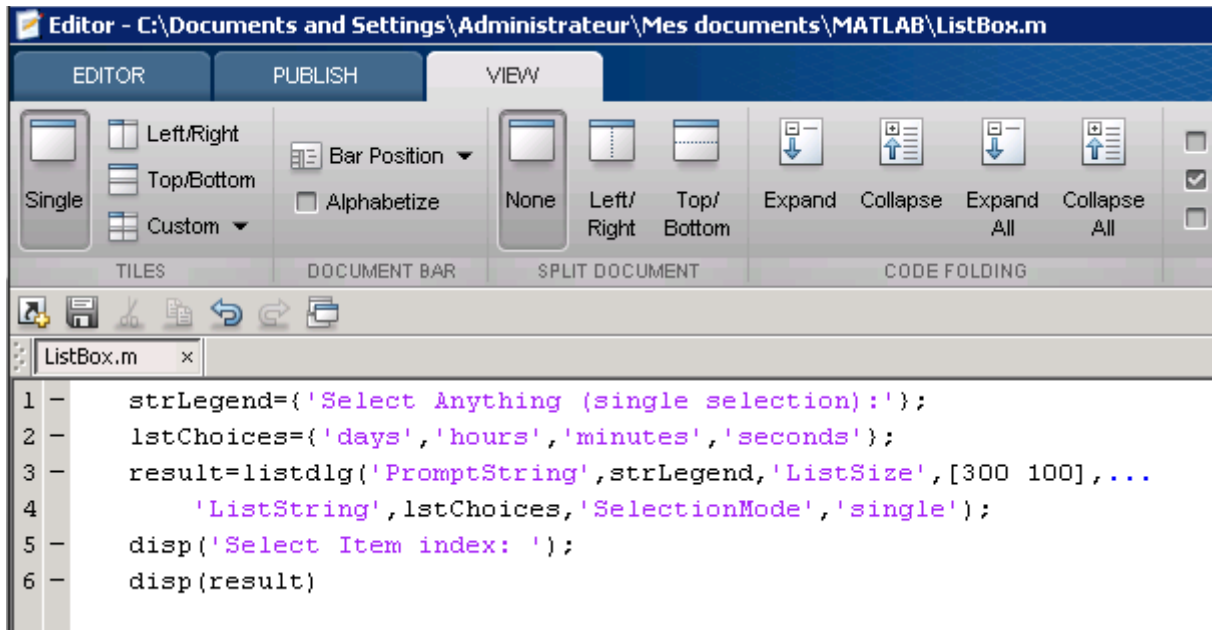


and the answer will be automatically managed by the *switch... case....* structure as you can see it below:

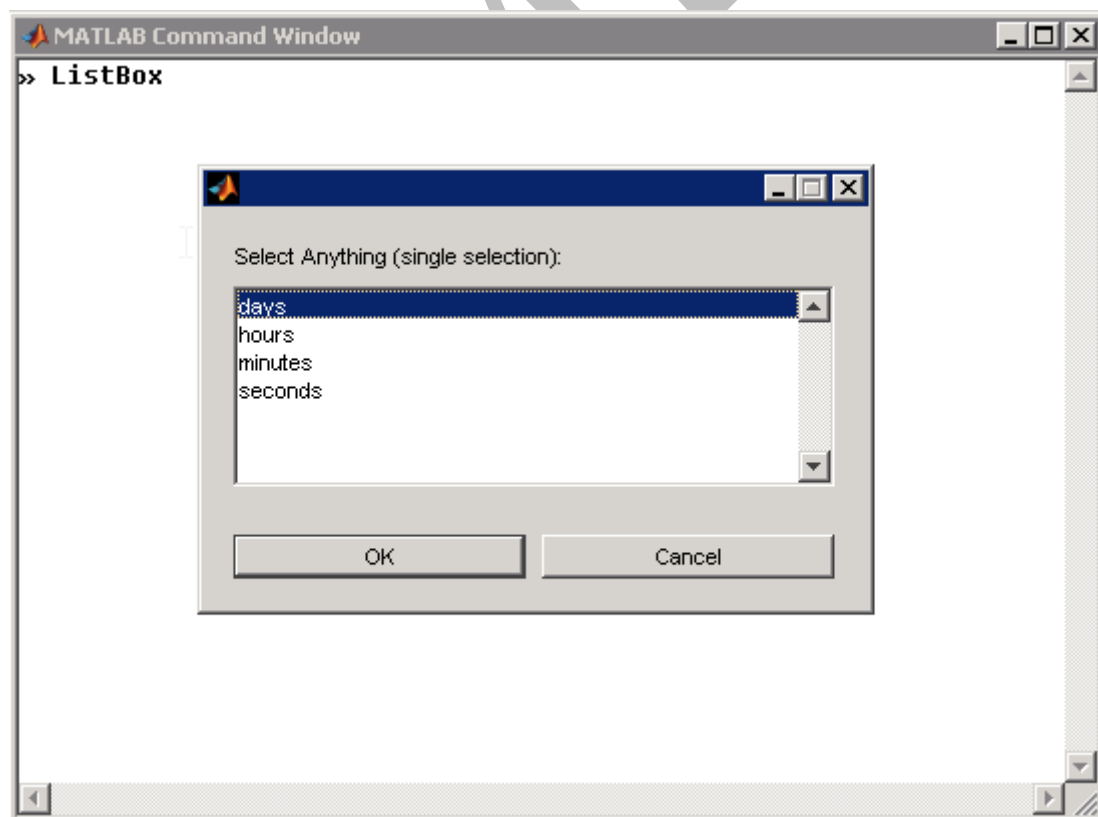


List dialog

An alternative to the menu is to create a list dialog using the function **listdlg()** but this last option has something more... the user can make multiples choices if you want to let him do it.



The result will look like this:



And as for the menu, this function returns the choosen index numbers:



Send E-mail

Another very common request is to send automatically e-mail through MATLAB to warn financial portfolio manager of some events. Here is a typical script that you can use and customize depending on your needs that is based on the **sendmail()** core function:

```

EDITOR PUBLISH VIEW
1 % Define these variables appropriately:
2 mail_address = 'isoz@sciences.ch'; % My E-Mail Address
3 user_name = 'isoz'; % My Username
4 password = 'mypassword'; % My E-Mail Password
5 smtp_server = 'mail.infomaniak.net'; % My SMTP Server
6
7 % Then this code will set up the preferences properly:
8 setpref('Internet', 'E_mail', mail_address);
9 setpref('Internet', 'SMTP_Username', user_name);
10 setpref('Internet', 'SMTP_Password', password);
11 setpref('Internet', 'SMTP_Server', smtp_server);
12 props = java.lang.System.getProperties;
13 props.setProperty('mail.smtp.auth', 'true');
14 props.setProperty('mail.smtp.socketFactory.class', 'javax.net.ssl.SSLSocketFactory');
15 props.setProperty('mail.smtp.socketFactory.port', '465');
16
17 % Send the email with carriage return in body text (10) and attachments
18 sendmail('mymail@somedomain.com', 'Mail from MATLAB', ...
19 ['Line1 of message' 10 'Line2 of message' 10 ...
20 'Line3 of message' 10 'Line4 of message'], ...
21 {'report.doc', 'chart.jpg'})

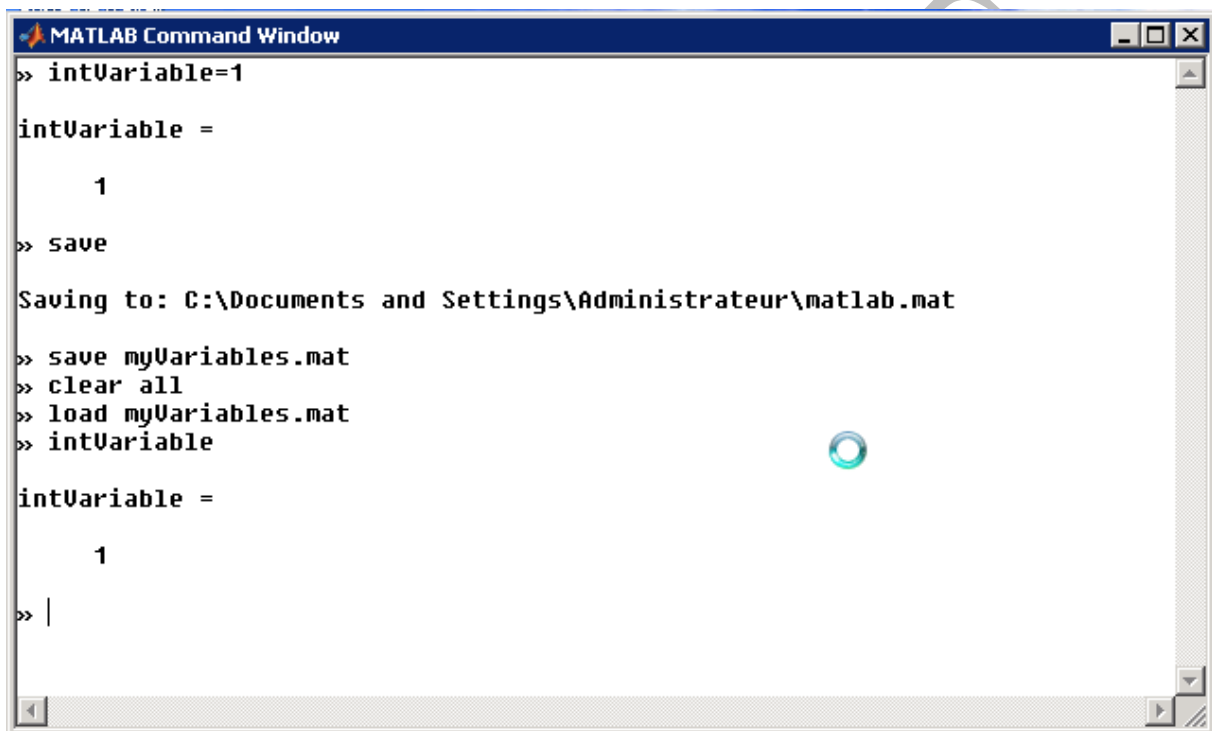
```

Data management

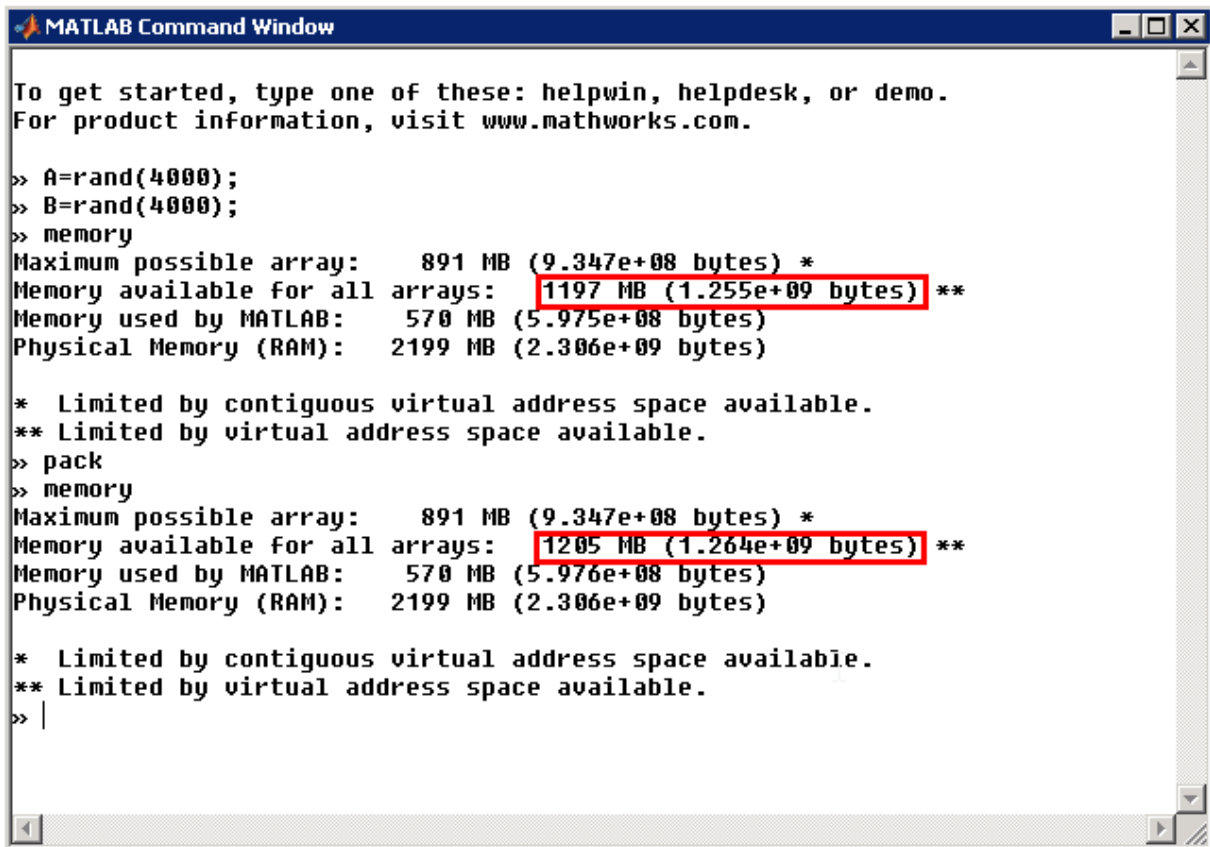
We will see here important stuff about data management in MATLAB without the use of any toolboxes like managing variables workspaces, importing/exporting *.csv or *.xls, *.xlsx files.

MATLAB files (*.mat and *.m)

This part is more a refresh than anything else on how to save a current variable workspace and reload it again:

A screenshot of the MATLAB Command Window. The window title is "MATLAB Command Window". The command history shows the following sequence of commands and outputs:
1. Command: `>> intVariable=1`
Output: `intVariable =`
 `1`
2. Command: `>> save`
Output: `Saving to: C:\Documents and Settings\Administrateur\matlab.mat`
3. Command: `>> save myVariables.mat`
4. Command: `>> clear all`
5. Command: `>> load myVariables.mat`
6. Command: `>> intVariable`
Output: `intVariable =`
 `1`
The command prompt is currently at `>> |`. A large, faint "SA" watermark is visible in the background of the image.

And also how to take a look to the memory usage and defragment it:



```
MATLAB Command Window

To get started, type one of these: helpwin, helpdesk, or demo.
For product information, visit www.mathworks.com.

>> A=rand(4000);
>> B=rand(4000);
>> memory
Maximum possible array:    891 MB (9.347e+08 bytes) *
Memory available for all arrays: 1197 MB (1.255e+09 bytes) **
Memory used by MATLAB:    570 MB (5.975e+08 bytes)
Physical Memory (RAM):    2199 MB (2.306e+09 bytes)

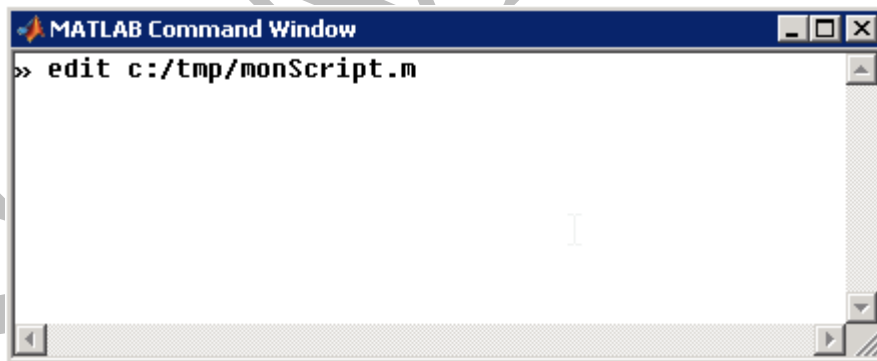
* Limited by contiguous virtual address space available.
** Limited by virtual address space available.

>> pack
>> memory
Maximum possible array:    891 MB (9.347e+08 bytes) *
Memory available for all arrays: 1205 MB (1.264e+09 bytes) **
Memory used by MATLAB:    570 MB (5.976e+08 bytes)
Physical Memory (RAM):    2199 MB (2.306e+09 bytes)

* Limited by contiguous virtual address space available.
** Limited by virtual address space available.

>> |
```

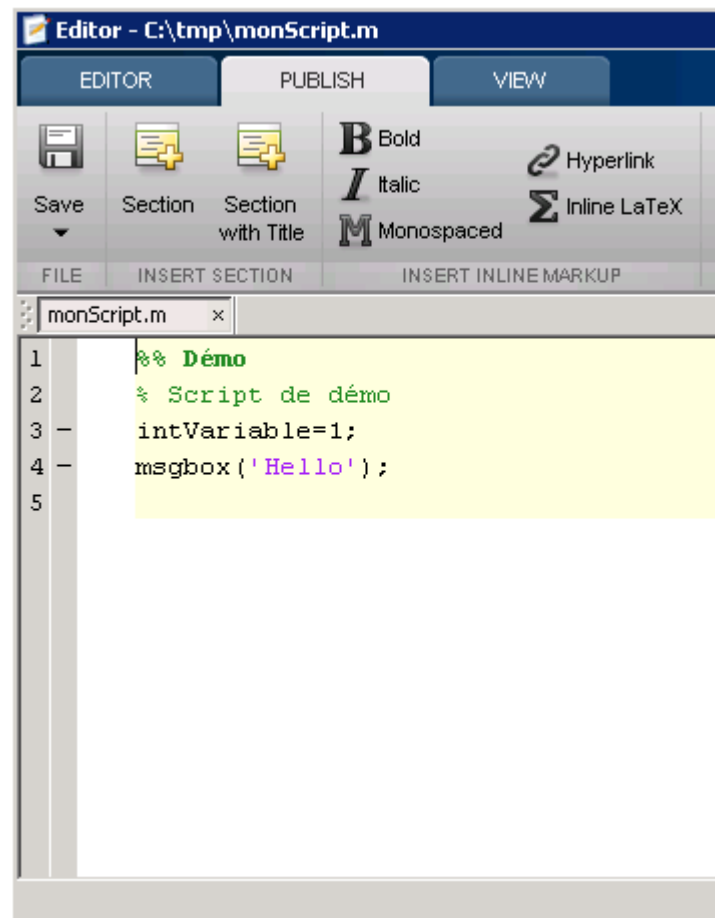
If we create a script:



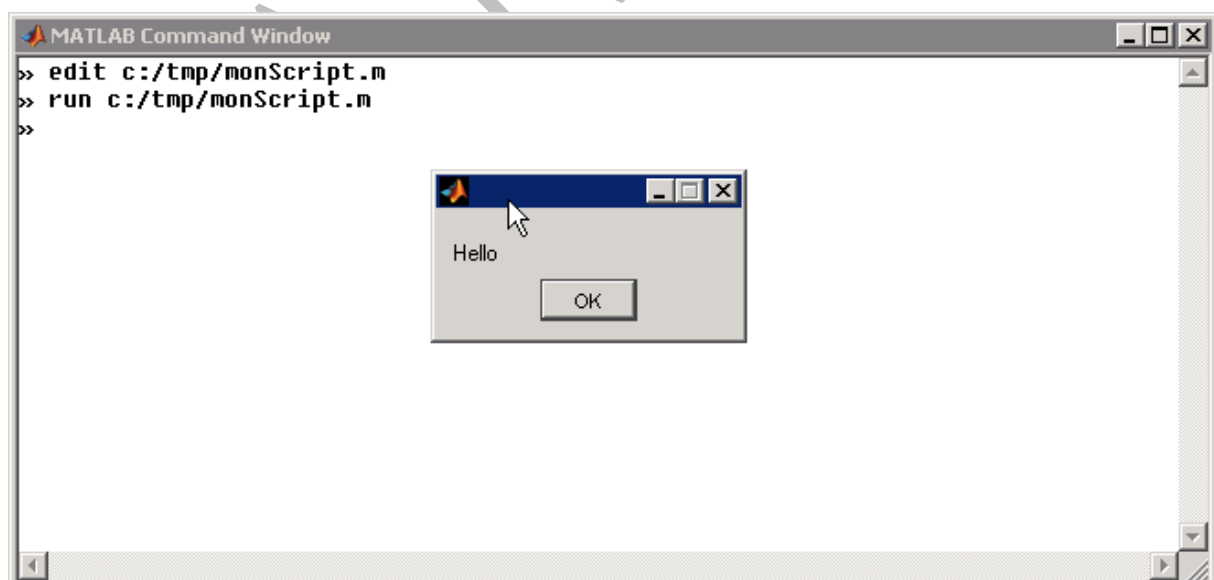
```
MATLAB Command Window

>> edit c:/tmp/monScript.m
```

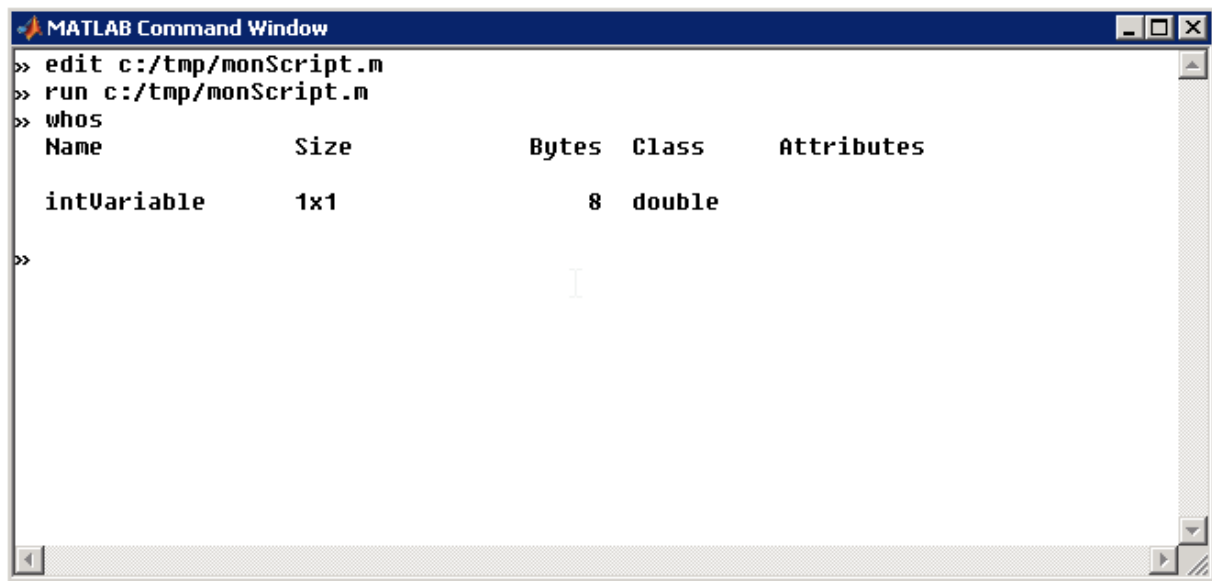
and that we write the following very simple stuff:



And that we run it afterwards:



Then we can see that the variable *intVariable* is now in the workspace:



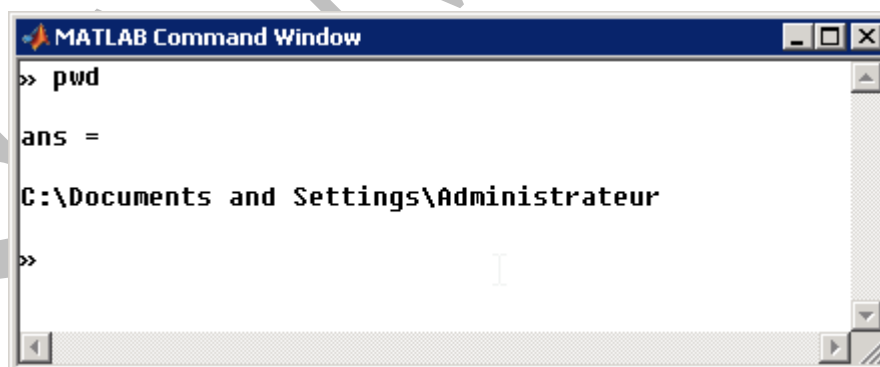
```
MATLAB Command Window
>> edit c:/tmp/monScript.m
>> run c:/tmp/monScript.m
>> whos
Name                Size          Bytes  Class   Attributes
intVariable         1x1              8  double
```

This is almost logic and not surprising that a variable into a script that is not a function can also be accessible from the current workspace.

Manage working paths

When you run a script or a function by default MATLAB will look into all defined working paths.

For refresh remember that to see the actual main working path you have to use the command **pwd**:

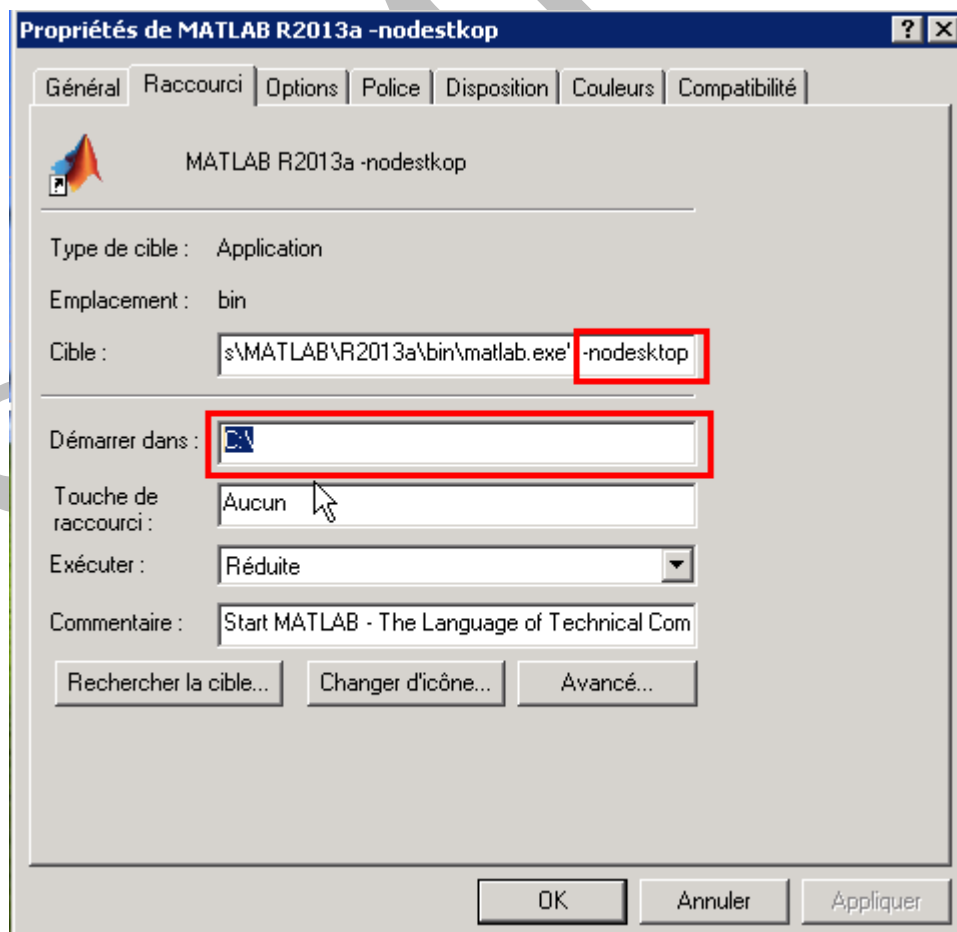


```
MATLAB Command Window
>> pwd
ans =
C:\Documents and Settings\Administrateur
```

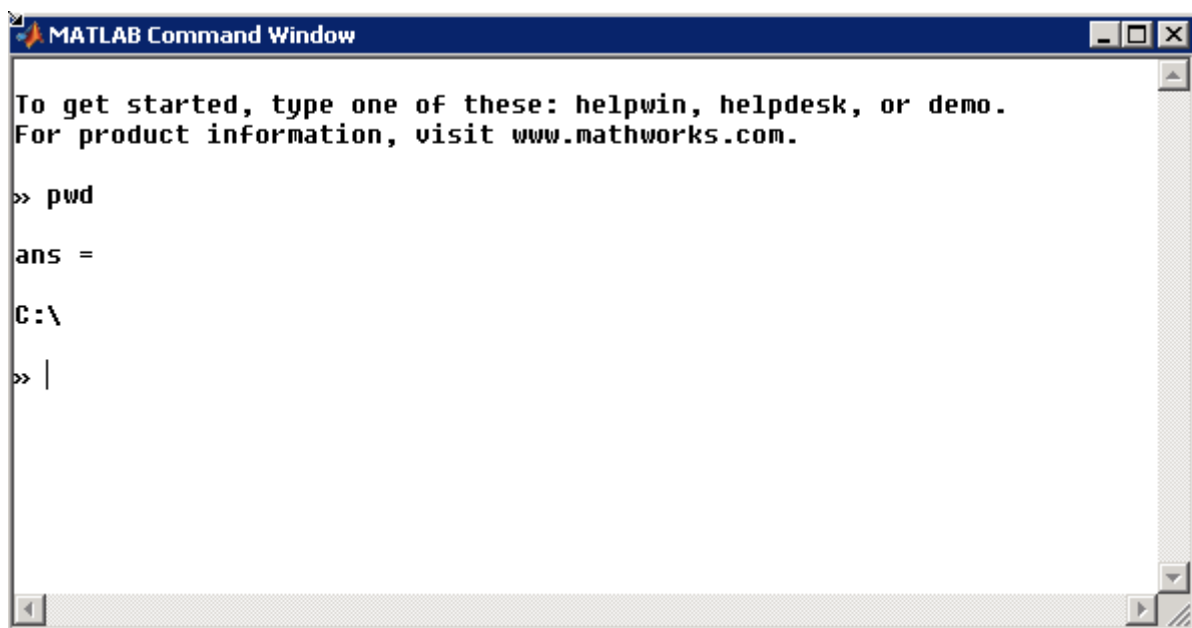
If you want to change this default path that is used especially for the *startup.m* and *finish.m* script files you how to customize the MATLAB shortcut as shown on the next page:



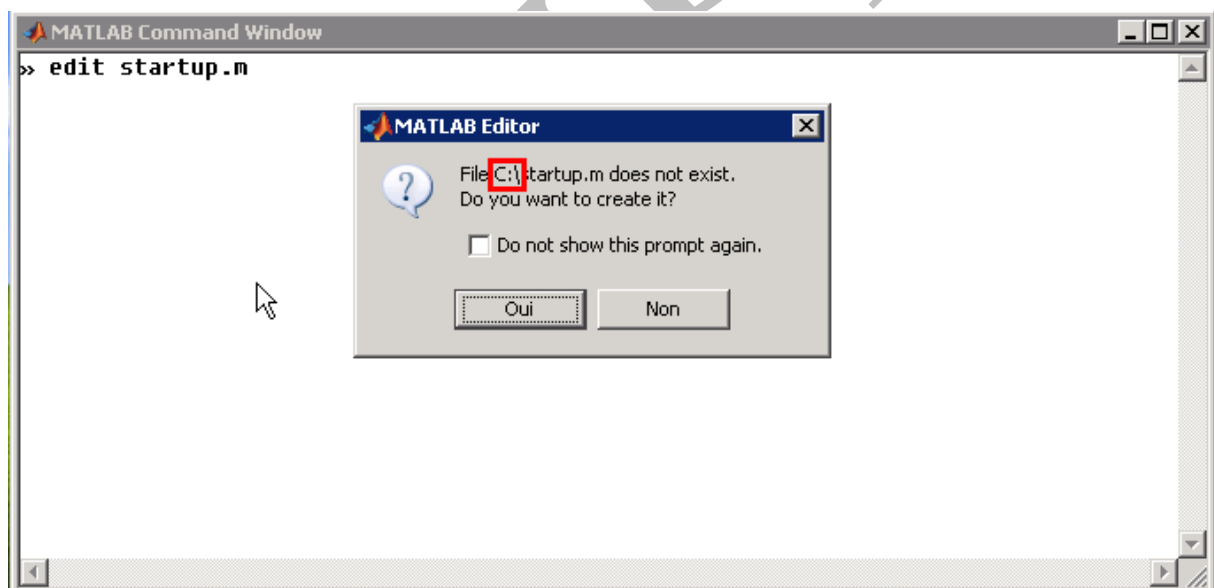
Make a right click on the shortcut and go into its properties and after in the shortcut register change the property *Start in* (the *-nodesktop* is facultative):



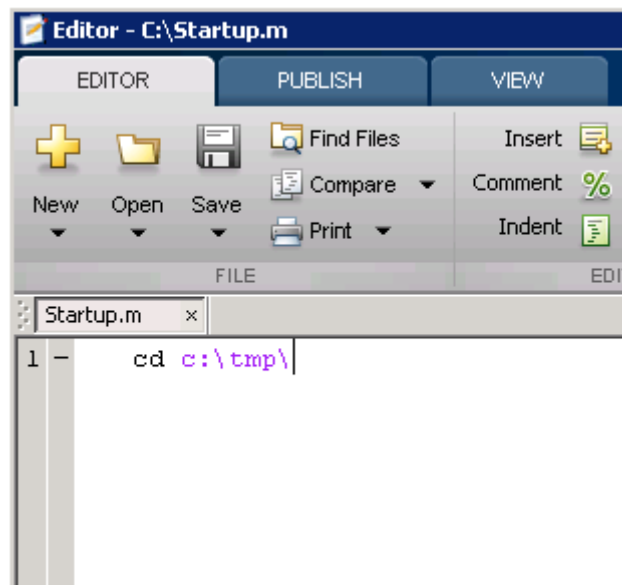
The you will see that the current main working path has changed:



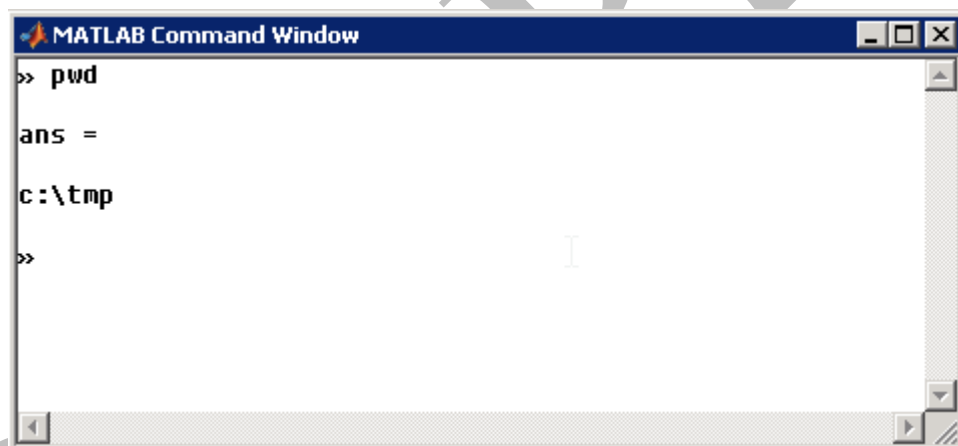
Now that the default working path is change you can create or move your *startup.m* file into it:



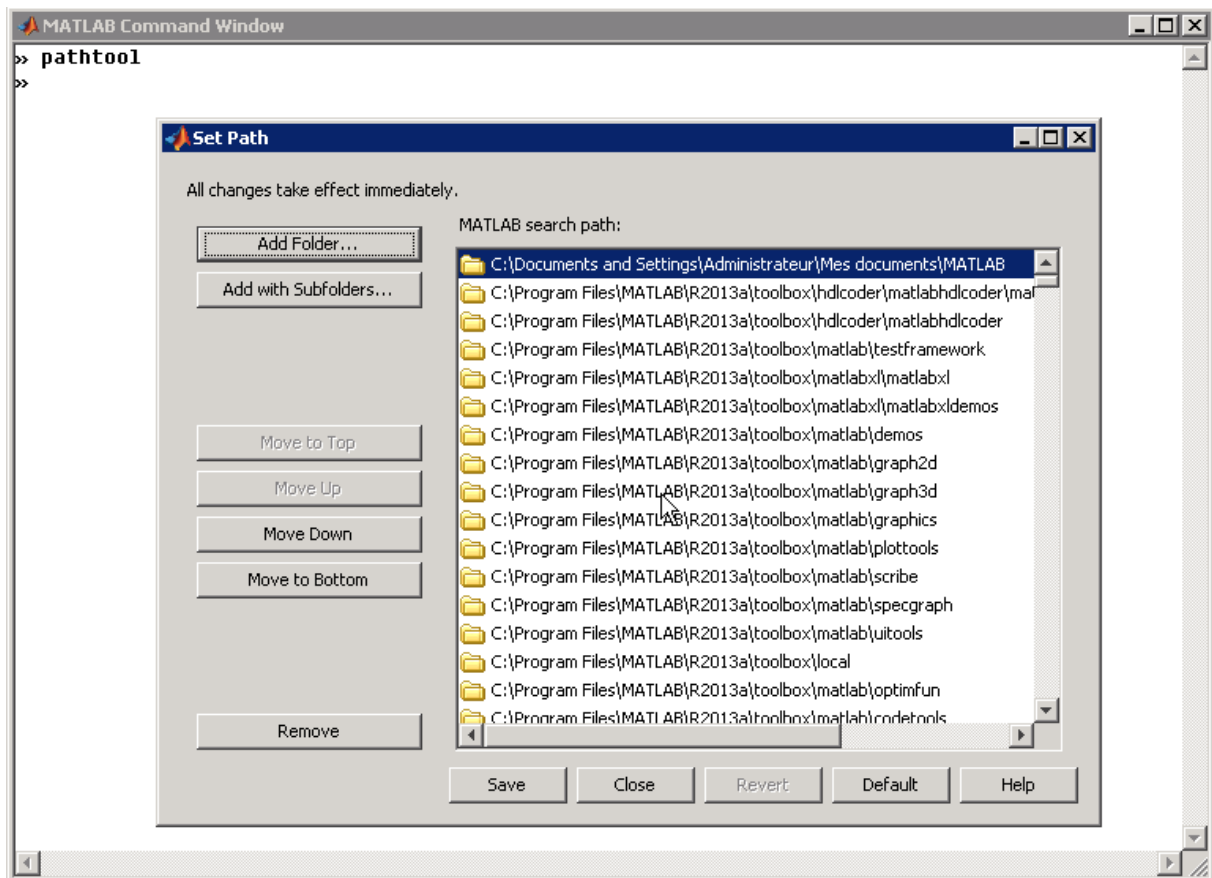
And write whatever you want:



and then if you close and restart MATLAB you will see that the *startup.m* file will be executed automatically from its new location:



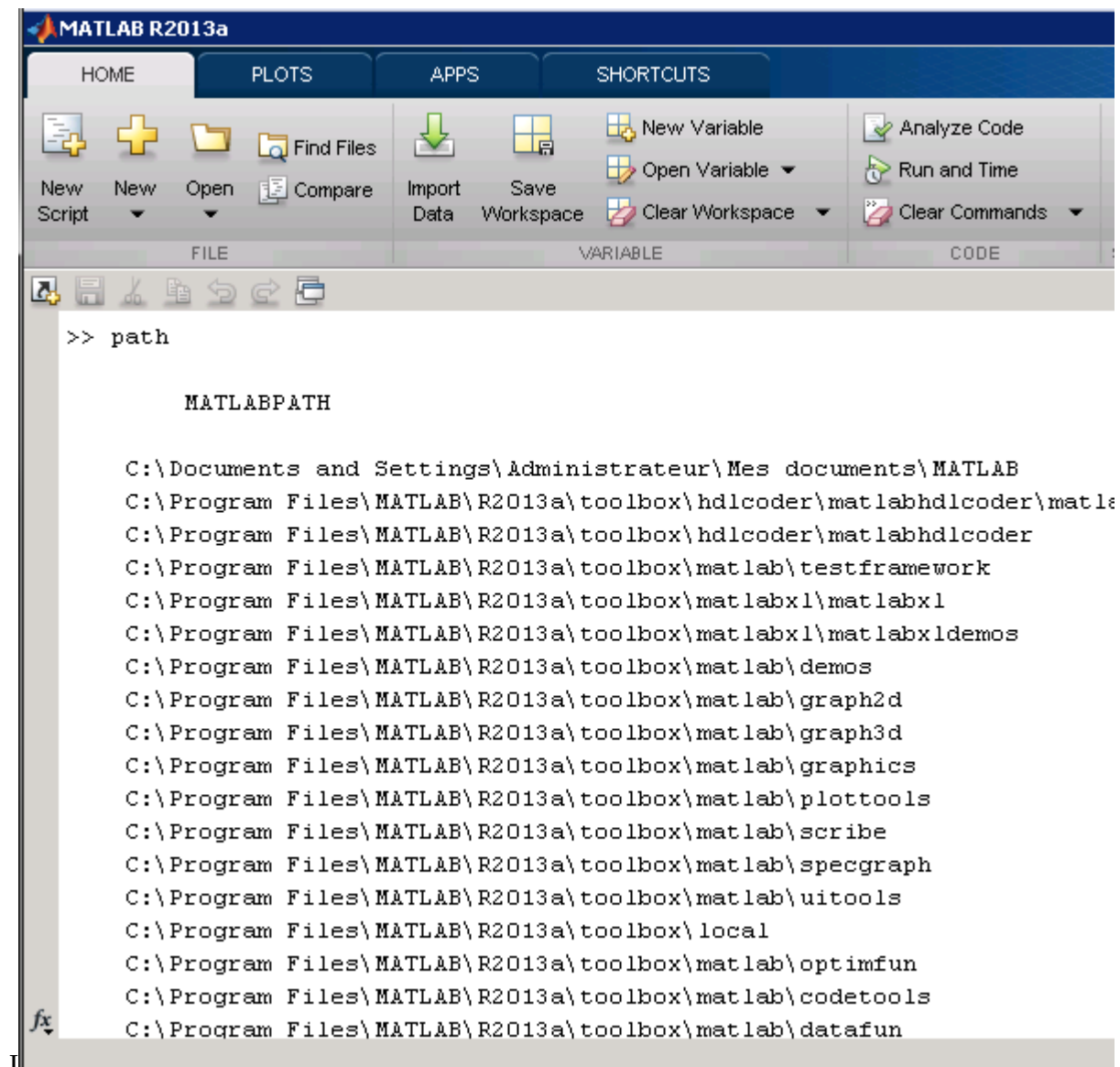
You can also manage working paths with a convenient userfriendly tool called the "path tool" by typing **pathtool** in the *Command Window*:



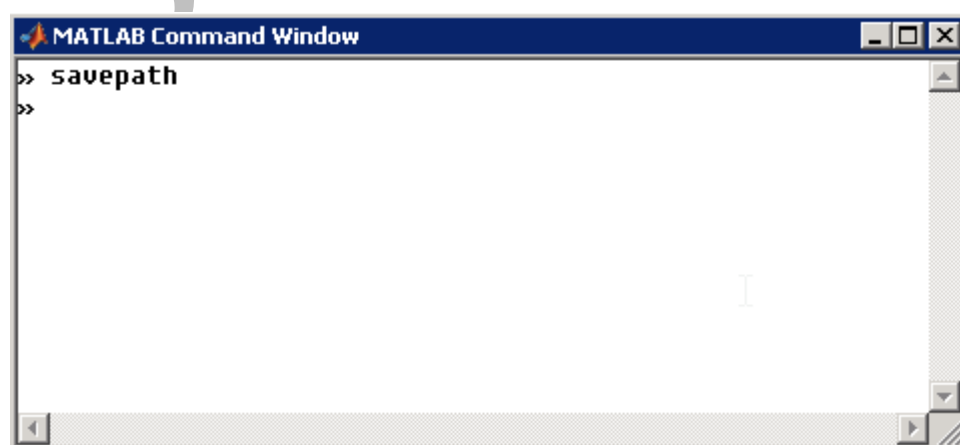
It is very easy to use thus there is nothing special to say. If you prefer to remove a Toolbox by a command you can still use:



If you type the command **path** you will see the list of all active paths in you MATLAB:



and if you want to save the status of your news paths for a future session of MATLAB you will have to not forget to type the command **savepath**:

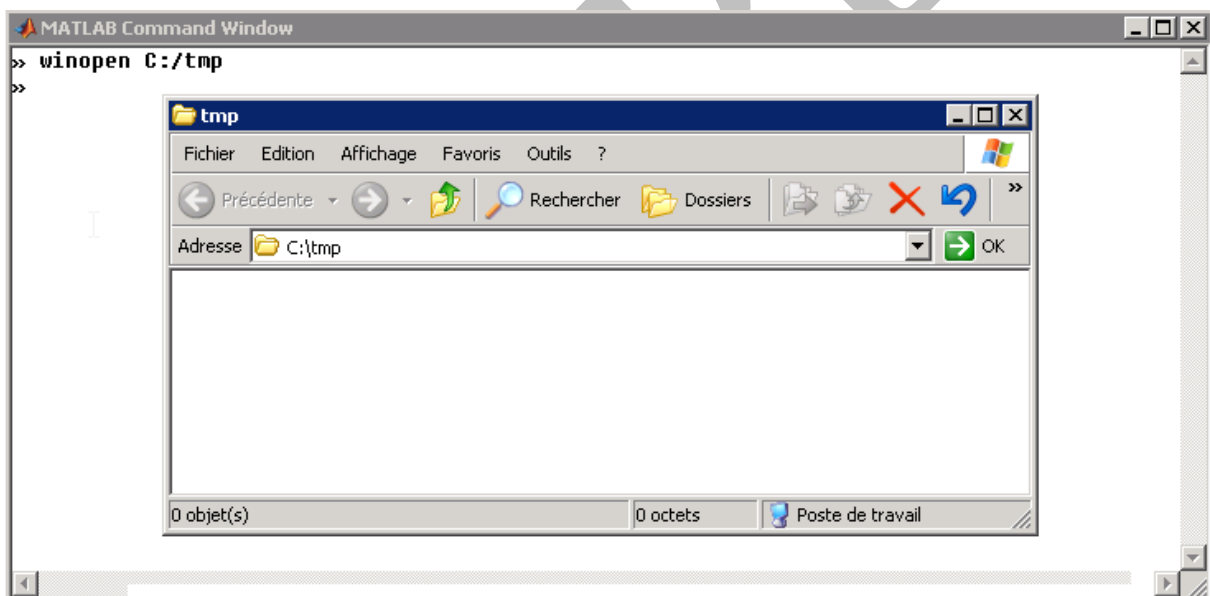


At any moment you can restore MATLAB default factory working paths if necessary by typing the command **restoredefaultpath**:

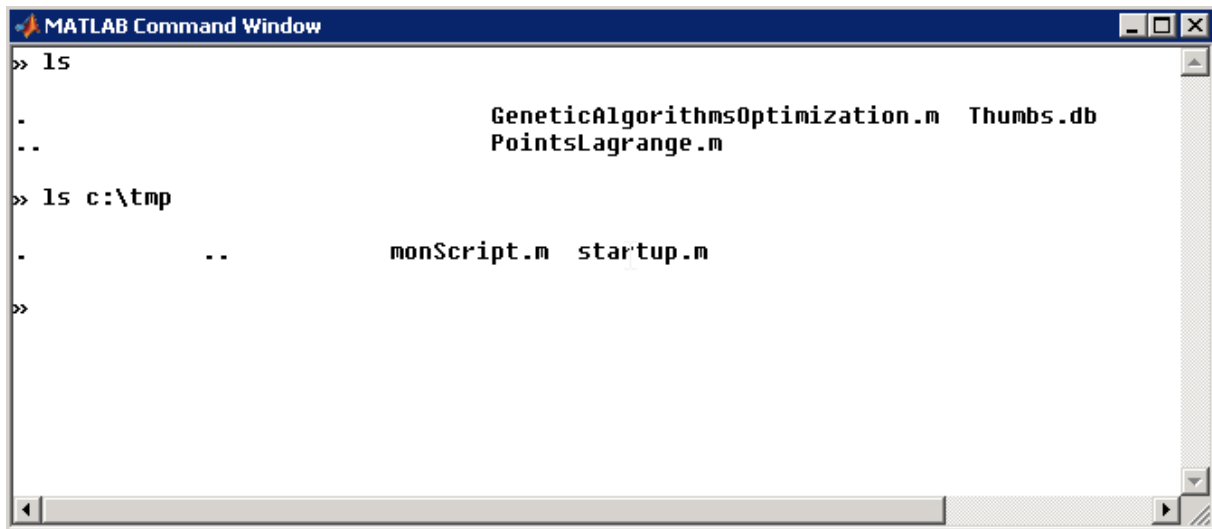


Browse and manage folders and files

When you create scripts that autogenerate reports it can be useful to open for the user the folder where all reports were saved. You can easily do that thanks to the command **winopen**:



To list all the files in a given folder just use the command **ls** like in UNIX:

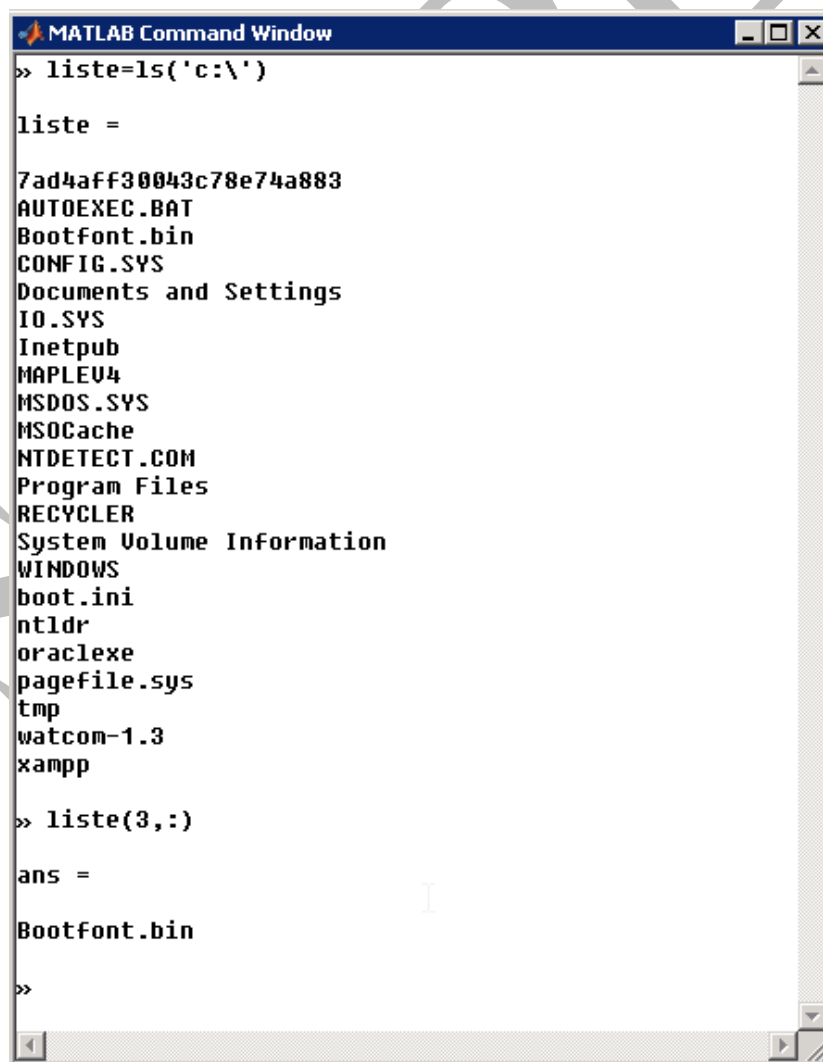


```
MATLAB Command Window
>> ls
.
..
GeneticAlgorithmsOptimization.m  Thumbs.db
PointsLagrange.m

>> ls c:\tmp
.
..
monScript.m  startup.m

>>
```

You can also store the list of files and folders in a vector to use it later for treatment purposes in your future scripts:



```
MATLAB Command Window
>> liste=ls('c:\')

liste =

7ad4aff30043c78e74a883
AUTOEXEC.BAT
Bootfont.bin
CONFIG.SYS
Documents and Settings
IO.SYS
Inetpub
MAPLEV4
MSDOS.SYS
MSOCache
NTDETECT.COM
Program Files
RECYCLER
System Volume Information
WINDOWS
boot.ini
ntldr
oraclexe
pagefile.sys
tmp
watcom-1.3
xampp

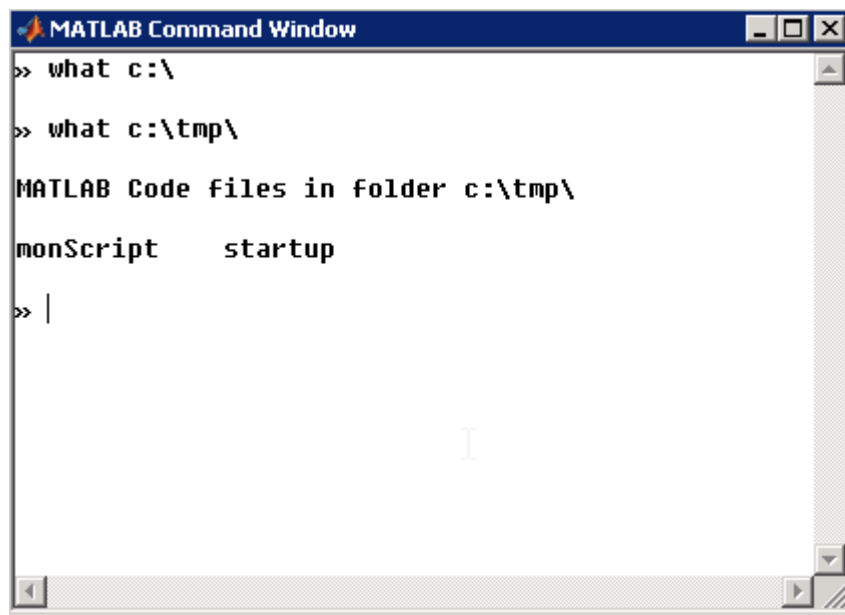
>> liste(3,:)

ans =

Bootfont.bin

>>
```

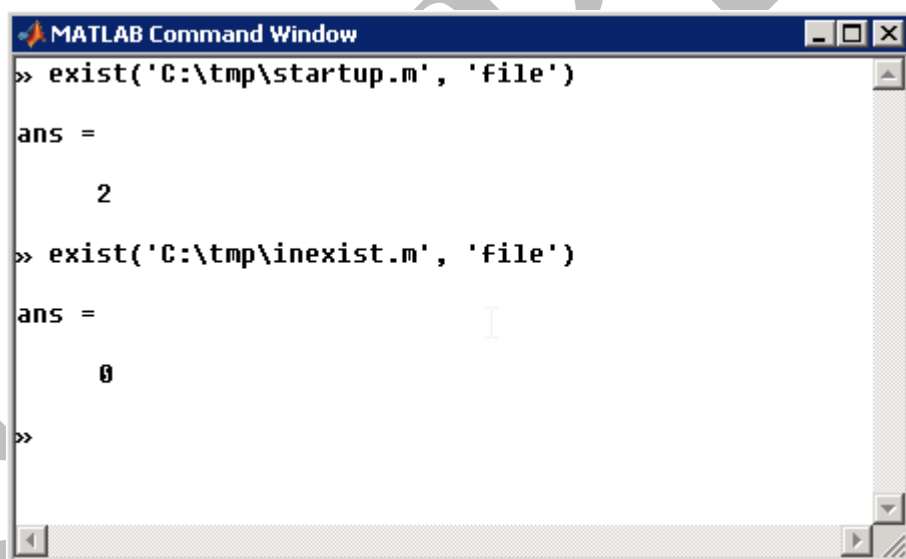
If you use the command **what** instead of **ls** you will get only MATLAB files type:



```
MATLAB Command Window
>> what c:\
>> what c:\tmp\

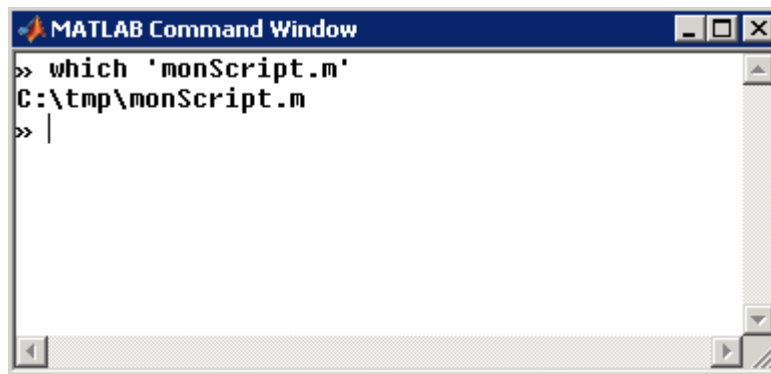
MATLAB Code files in folder c:\tmp\
monScript    startup
>> |
```

You can also check if a file already exists or not:



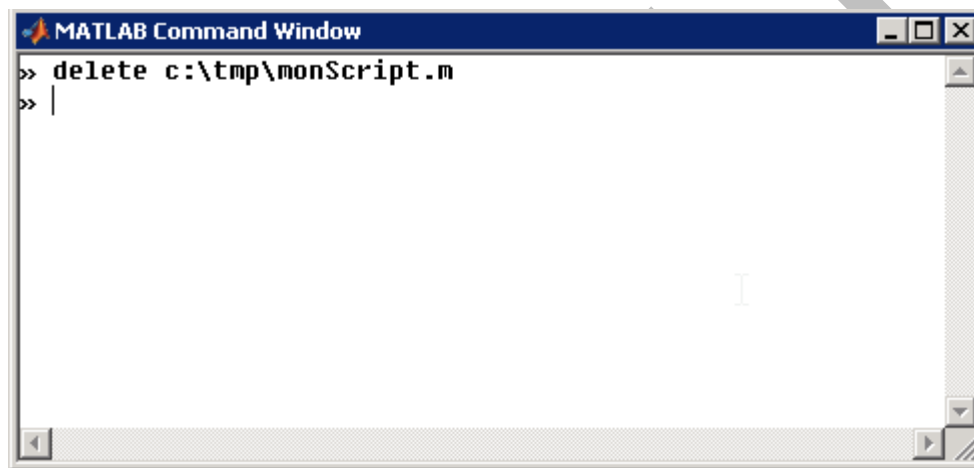
```
MATLAB Command Window
>> exist('C:\tmp\startup.m', 'file')
ans =
     2
>> exist('C:\tmp\inexist.m', 'file')
ans =
     0
>>
```

If you use the command **which** with a **file name** or a **function name** this command will give you in which working path is stored the corresponding script:

A screenshot of the MATLAB Command Window. The title bar reads "MATLAB Command Window". The command prompt shows the command `>> which 'monScript.m'` followed by the output `C:\tmp\monScript.m`. The cursor is on the line `>> |`.

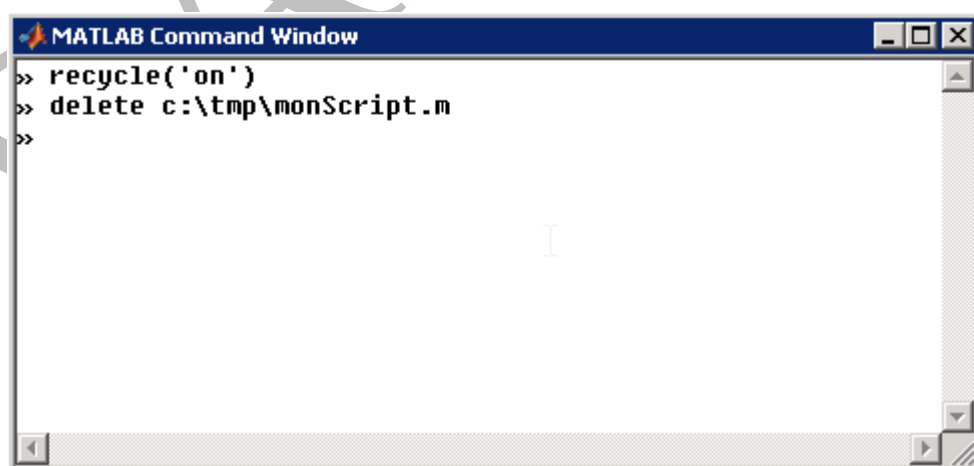
```
>> which 'monScript.m'
C:\tmp\monScript.m
>> |
```

You can also delete a file using the command **delete**:

A screenshot of the MATLAB Command Window. The title bar reads "MATLAB Command Window". The command prompt shows the command `>> delete c:\tmp\monScript.m` followed by the cursor on the line `>> |`.

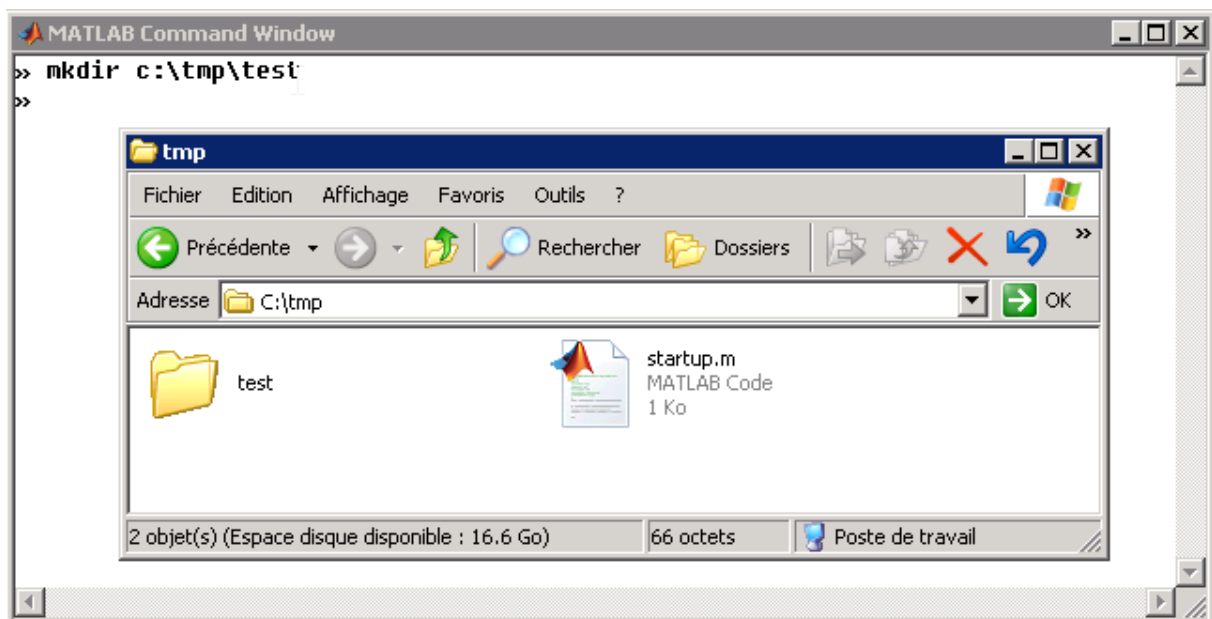
```
>> delete c:\tmp\monScript.m
>> |
```

The problem is that the file will be definitively deleted. If you want to activate the recycle bin you have to activate it by using the command **recycle('on')**:

A screenshot of the MATLAB Command Window. The title bar reads "MATLAB Command Window". The command prompt shows two commands: `>> recycle('on')` and `>> delete c:\tmp\monScript.m`, followed by the cursor on the line `>> |`.

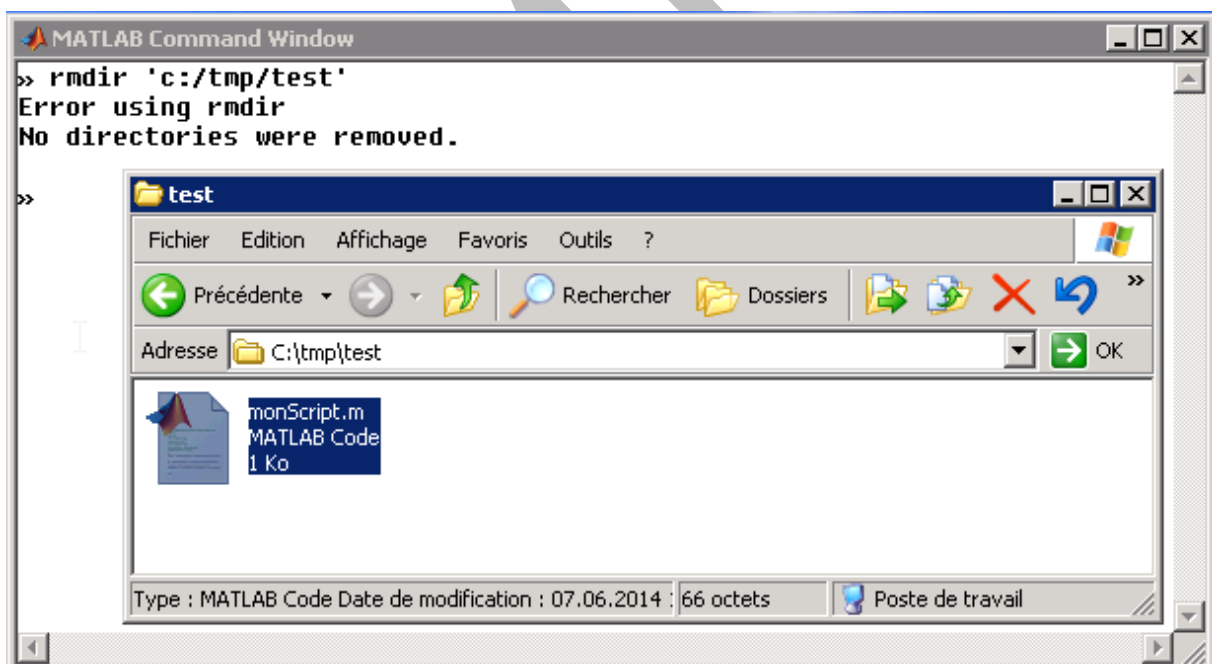
```
>> recycle('on')
>> delete c:\tmp\monScript.m
>> |
```


You can also create folders by using the UNIX **mkdir** command:

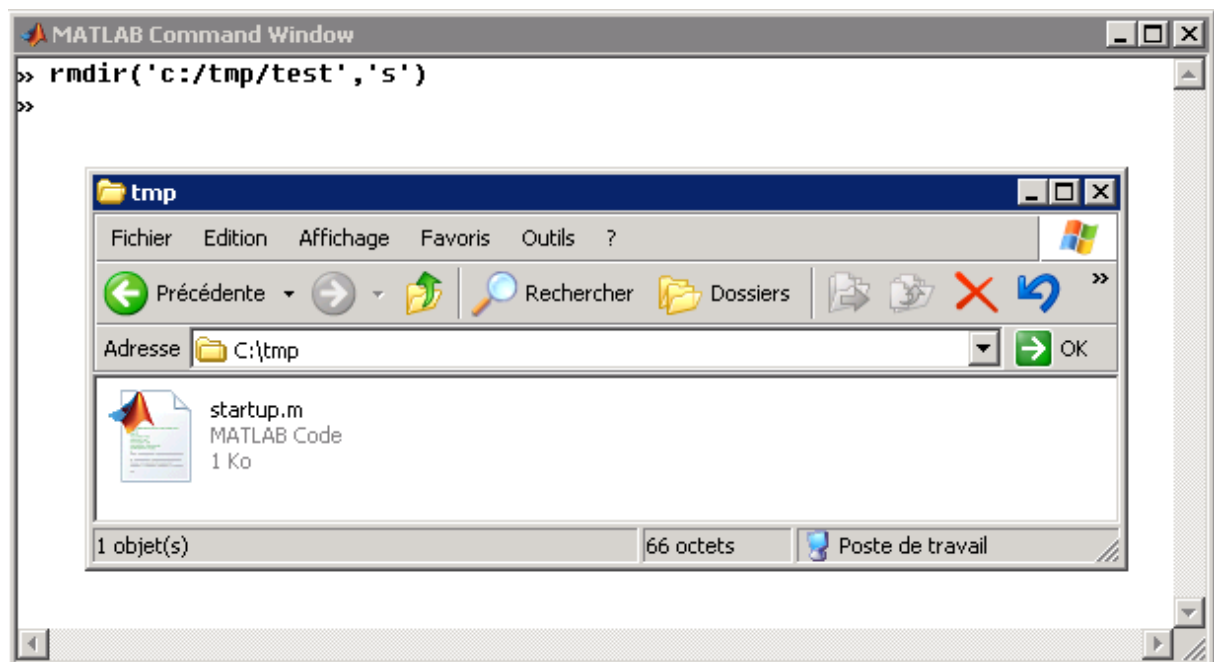


Or remove (delete) a folder by using the UNIX **rmdir** command.

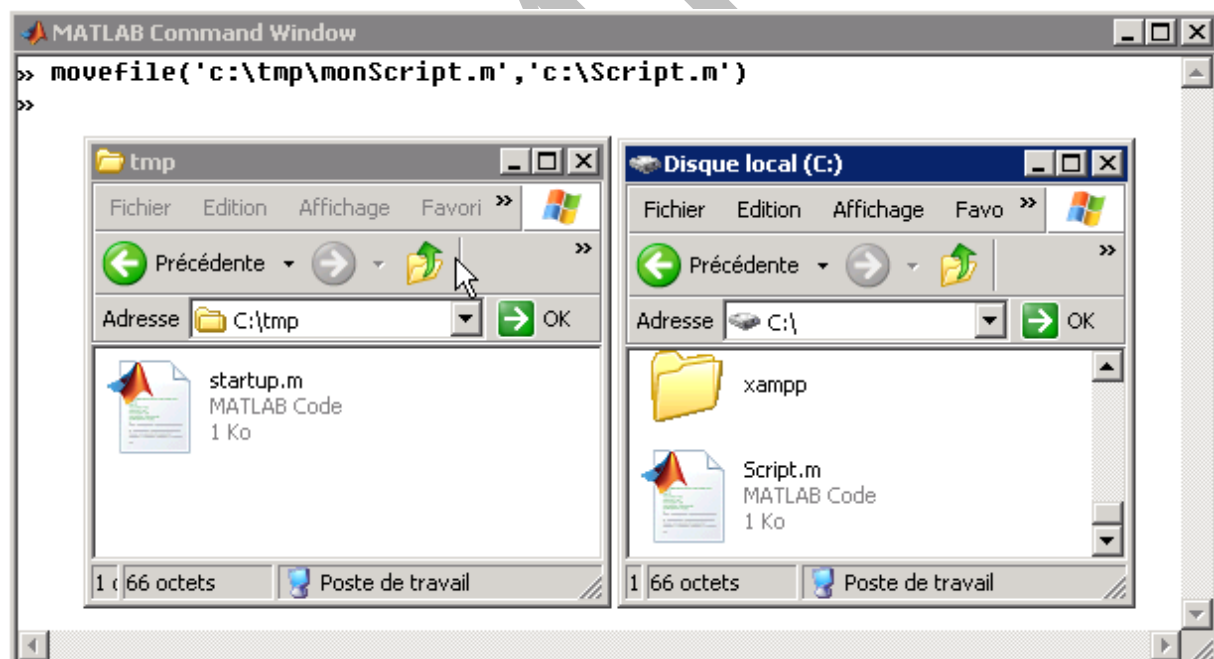
But there is a small trap. If you try to simply use the command you will get an **error message**:



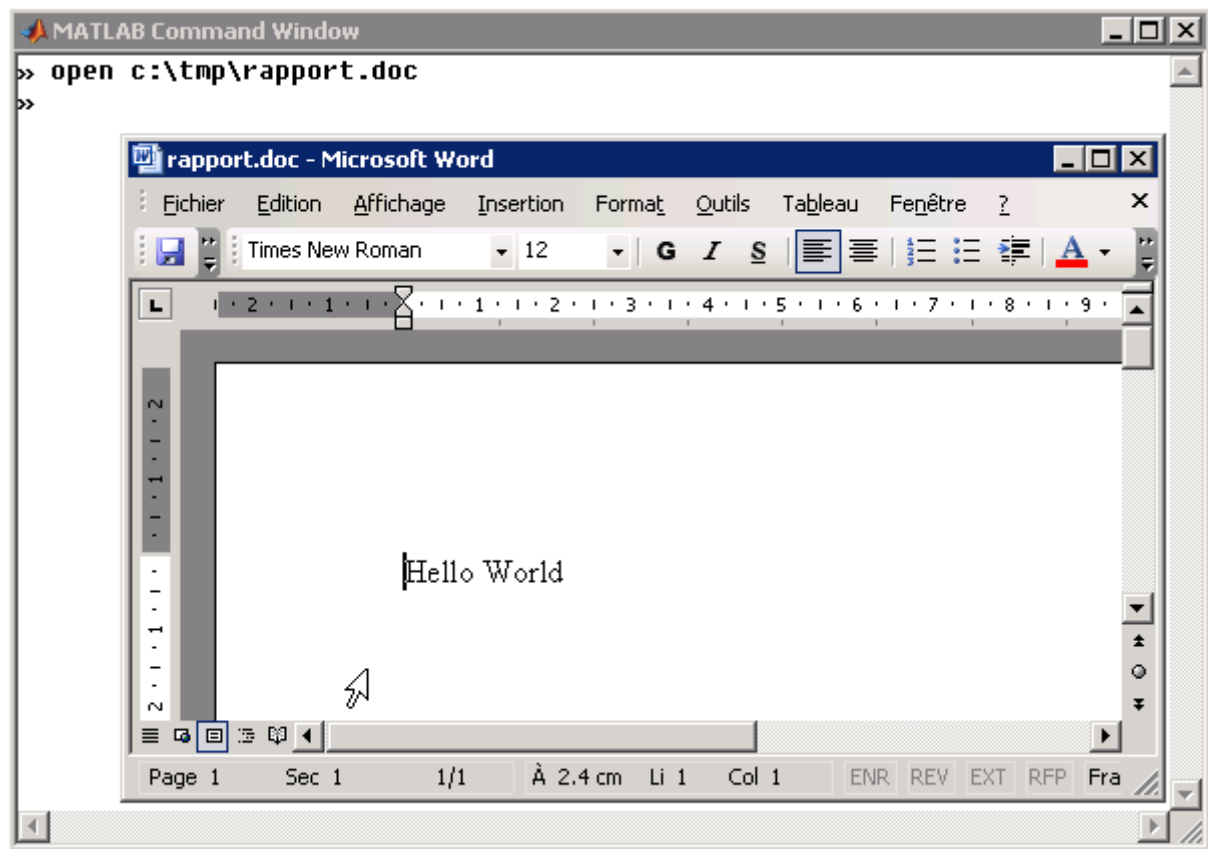
To avoid this you have to pass the parameter 's' to the command like this:



You can also need when you develop applications with MATLAB to move files from a folder to the other one. For this purpose you can just use the function **movefile()**:



And if instead of opening a folder with reports you want to open the reports themselves just use the command **open**:



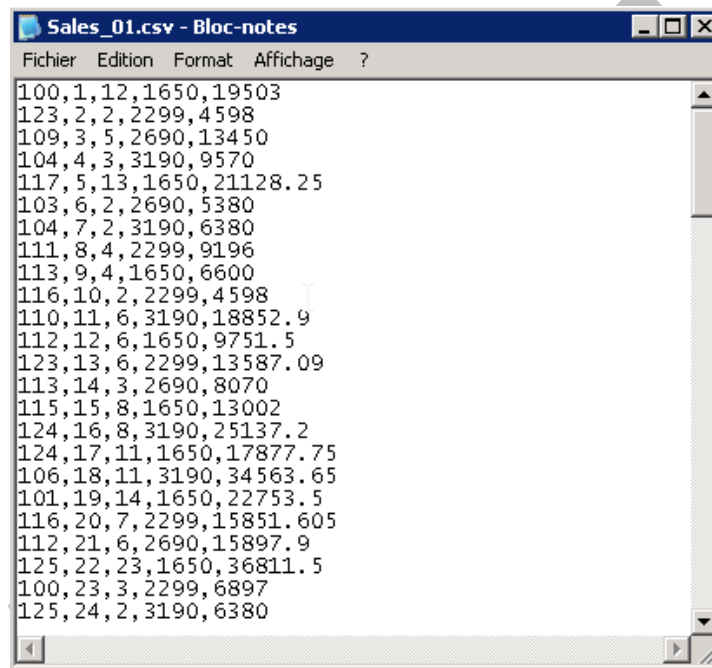
Import/Export csv files

We will see here how to import *.csv files without using the import wizard of MATLAB but just by using the *Command Window*.

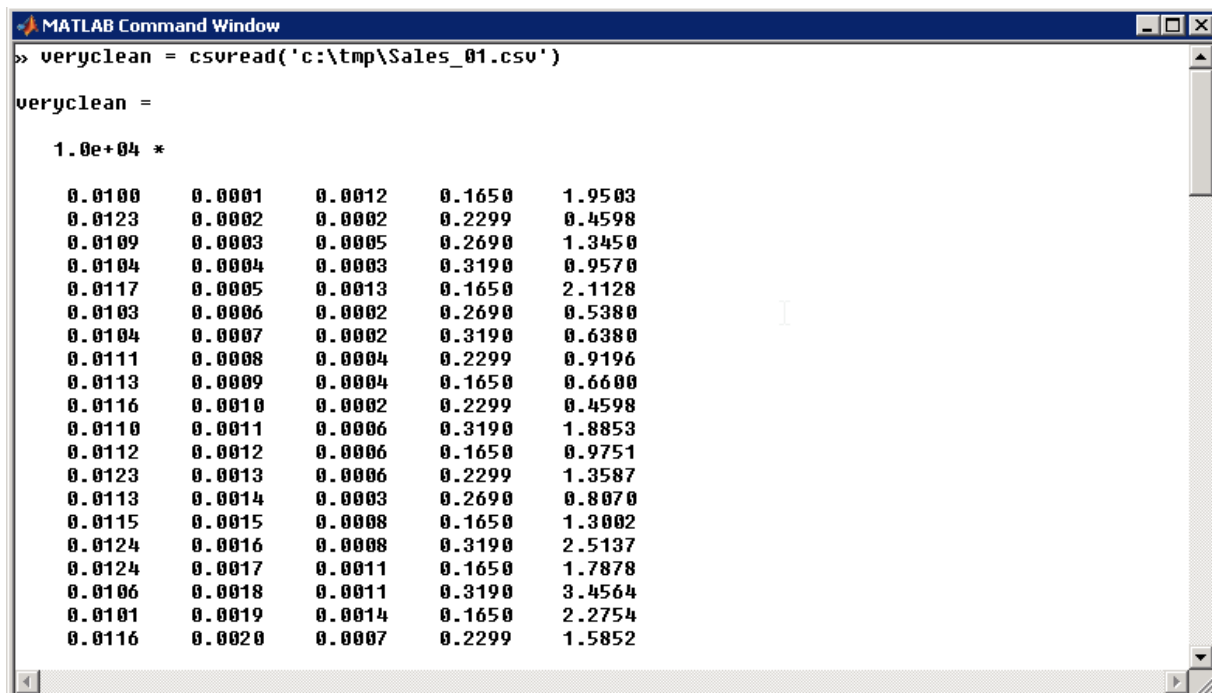
Import very clean local csv files (with comas, without headers, without texts, without dates)

The command `csvread()` can't import a file with semi-colons or with headers or with texts or with dates.

You must have a file with comas and numerical values only:



Then you just have to use the command `csvread()` to import the datas into a matrix:



```

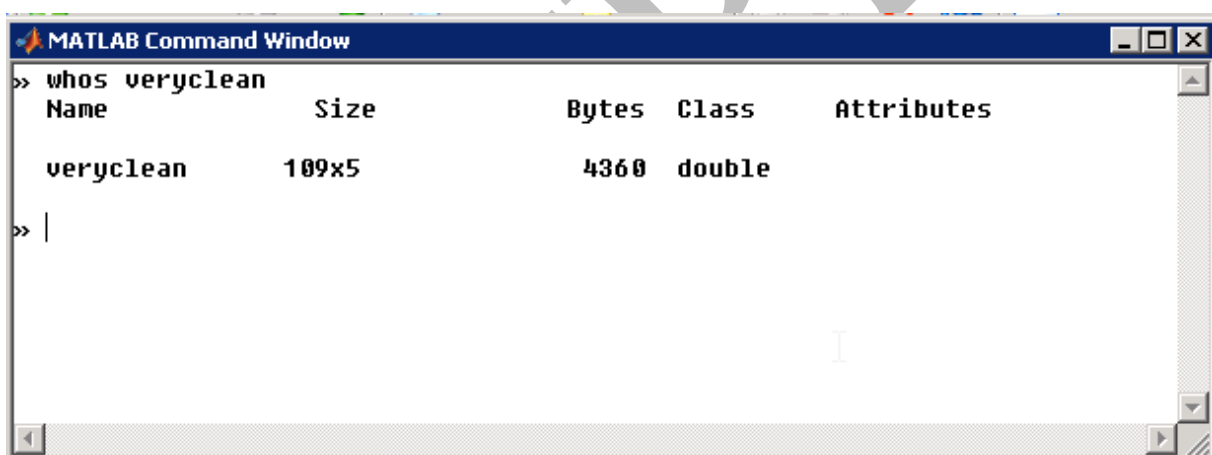
MATLAB Command Window
>> veryclean = csvread('c:\tmp\Sales_01.csv')

veryclean =

    1.0e+04 *
    0.0100    0.0001    0.0012    0.1650    1.9503
    0.0123    0.0002    0.0002    0.2299    0.4598
    0.0109    0.0003    0.0005    0.2690    1.3450
    0.0104    0.0004    0.0003    0.3190    0.9570
    0.0117    0.0005    0.0013    0.1650    2.1128
    0.0103    0.0006    0.0002    0.2690    0.5380
    0.0104    0.0007    0.0002    0.3190    0.6380
    0.0111    0.0008    0.0004    0.2299    0.9196
    0.0113    0.0009    0.0004    0.1650    0.6600
    0.0116    0.0010    0.0002    0.2299    0.4598
    0.0110    0.0011    0.0006    0.3190    1.8853
    0.0112    0.0012    0.0006    0.1650    0.9751
    0.0123    0.0013    0.0006    0.2299    1.3587
    0.0113    0.0014    0.0003    0.2690    0.8070
    0.0115    0.0015    0.0008    0.1650    1.3002
    0.0124    0.0016    0.0008    0.3190    2.5137
    0.0124    0.0017    0.0011    0.1650    1.7878
    0.0106    0.0018    0.0011    0.3190    3.4564
    0.0101    0.0019    0.0014    0.1650    2.2754
    0.0116    0.0020    0.0007    0.2299    1.5852

```

As you can see its a matrix:



```

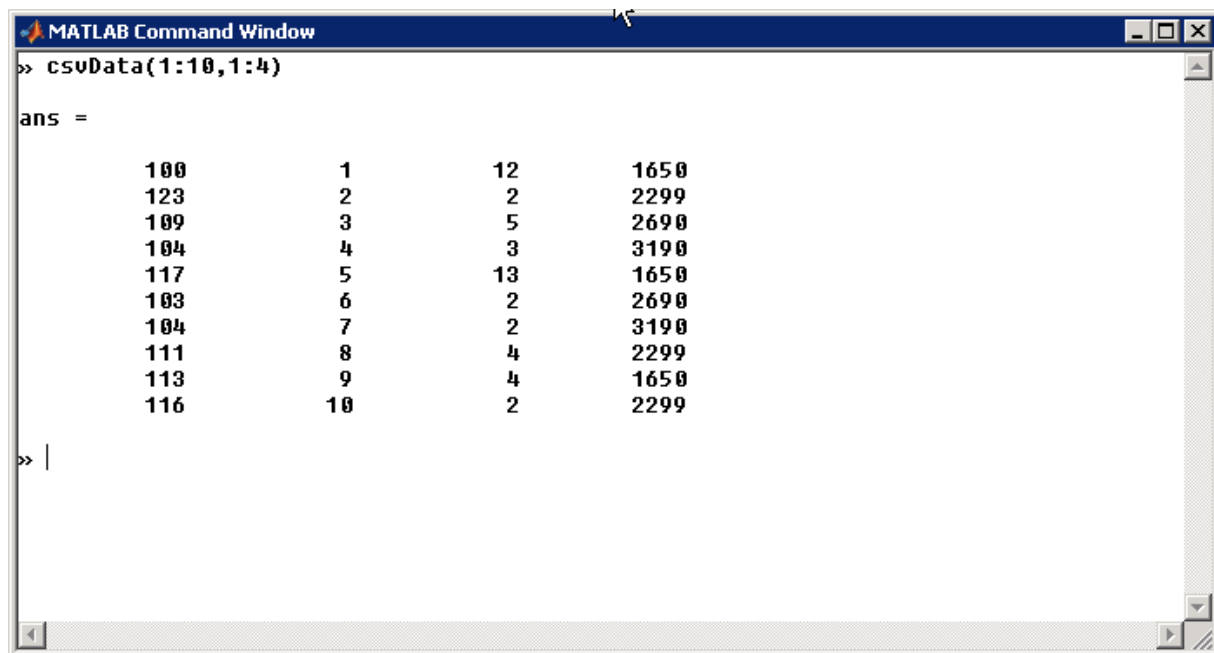
MATLAB Command Window
>> whos veryclean

Name          Size          Bytes    Class    Attributes

veryclean     109x5          4360    double

```

And you can like any other matrices take a subset:



```
MATLAB Command Window
>> csvData(1:10,1:4)

ans =

    100     1    12    1650
    123     2     2    2299
    109     3     5    2690
    104     4     3    3190
    117     5    13    1650
    103     6     2    2690
    104     7     2    3190
    111     8     4    2299
    113     9     4    1650
    116    10     2    2299

>> |
```

Import clean local csv files (with semi-colons, with headers, without text, without dates)

Now we will see that the function **importdata()** can manager a few more things than **csvread()** but still can't import files with strings or dates.

Consider the following file:

```

"N.Client";"Activité";"N.de.Commande";"Date.de.commande";"Article";"Quantité"
"1";"100";"Assurances";"1";"03.01.2000";"Compaq Presario 100";"12;1650";"1.50%";"15"
"2";"123";"Machines/outils";"2";"03.01.2000";"IBM 500";"2;2299";"0.00%";"4598";"oui"
"3";"109";"Éducation";"3";"03.01.2000";"AST Intel 150";"5;2690";"0.00%";"13450";"oui"
"4";"104";"Éducation";"4";"03.01.2000";"AST Intel 200";"3;3190";"0.00%";"9570";"oui"
"5";"117";"Banques";"5";"04.01.2000";"Compaq Presario 100";"13;1650";"1.50%";"21128"
"6";"103";"Assurances";"6";"04.01.2000";"AST Intel 150";"2;2690";"0.00%";"5380";"oui"
"7";"104";"Éducation";"7";"04.01.2000";"AST Intel 200";"2;3190";"0.00%";"6380";"oui"
"8";"111";"Alimentaire";"8";"04.01.2000";"IBM 500";"4;2299";"0.00%";"9196";"oui"
"9";"113";"Construction";"9";"04.01.2000";"Compaq Presario 100";"4;1650";"0.00%";"6"
"10";"116";"Pharmaceutique";"10";"04.01.2000";"IBM 500";"2;2299";"0.00%";"4598";"oui"
"11";"110";"Distribution";"11";"05.01.2000";"AST Intel 200";"6;3190";"1.50%";"18852"
"12";"112";"Machines/outils";"12";"05.01.2000";"Compaq Presario 100";"6;1650";"1.5"
"13";"123";"Machines/outils";"13";"05.01.2000";"IBM 500";"6;2299";"1.50%";"13587.05"
"14";"113";"Construction";"14";"05.01.2000";"AST Intel 150";"3;2690";"0.00%";"8070";"
"15";"115";"Distribution";"15";"05.01.2000";"Compaq Presario 100";"8;1650";"1.50%"
"16";"124";"Éducation";"16";"05.01.2000";"AST Intel 200";"8;3190";"1.50%";"25137.2;"
"17";"124";"Éducation";"17";"05.01.2000";"Compaq Presario 100";"11;1650";"1.50%";"1"
"18";"106";"Construction";"18";"05.01.2000";"AST Intel 200";"11;3190";"1.50%";"3456"
"19";"101";"Construction";"19";"05.01.2000";"Compaq Presario 100";"14;1650";"1.50%"
"20";"116";"Pharmaceutique";"20";"06.01.2000";"IBM 500";"7;2299";"1.50%";"15851.605"
"21";"112";"Machines/outils";"21";"06.01.2000";"AST Intel 150";"6;2690";"1.50%";"15"
"22";"125";"Construction";"22";"06.01.2000";"Compaq Presario 100";"23;1650";"3.00%"
"23";"100";"Assurances";"23";"06.01.2000";"IBM 500";"3;2299";"0.00%";"6897";"oui"
"24";"125";"Construction";"24";"06.01.2000";"AST Intel 200";"2;3190";"0.00%";"6380";"
"25";"104";"Éducation";"25";"07.01.2000";"AST Intel 150";"12;2690";"1.50%";"31795.8"
"26";"126";"Machines/outils";"26";"07.01.2000";"AST Intel 150";"24;2690";"3.00%";"6"
"27";"121";"Pharmaceutique";"27";"07.01.2000";"IBM 500";"8;2299";"1.50%";"18116.12;"

```

The result will be easy to manage because it results in a structure:

```

>> clean=importdata('c:\tmp\Sales_02.csv',';',1);
>> clean

clean =

    data: [1089x6 double]
    textdata: {'Ncommande' 'NClient' 'Activité' 'Article' 'PrixParPièce' 'Rabais'}
    colheaders: {'Ncommande' 'NClient' 'Activité' 'Article' 'PrixParPièce' 'Rabais'}

>> clean.colheaders

ans =

    'Ncommande'    'NClient'    'Activité'    'Article'    'PrixParPièce'    'Rabais'

>> clean.data

ans =

    1.0e+04 *

    0.0001    0.0100    0.0001    0.0012    0.1650    1.9503
    0.0002    0.0123    0.0002    0.0002    0.2299    0.4598
    0.0003    0.0109    0.0003    0.0005    0.2690    1.3450
    0.0004    0.0104    0.0004    0.0003    0.3190    0.9570
    0.0005    0.0117    0.0005    0.0013    0.1650    2.1128
    0.0006    0.0103    0.0006    0.0002    0.2690    0.5380
    0.0007    0.0104    0.0007    0.0002    0.3190    0.6380
    0.0008    0.0111    0.0008    0.0004    0.2299    0.9196

```

Import local csv files (with semi-colons, with headers, with text, with dates)

The most easy way to import typical csv files is to use the function **fopen()** associated with **textscan()**. Consider as example the following file:

```

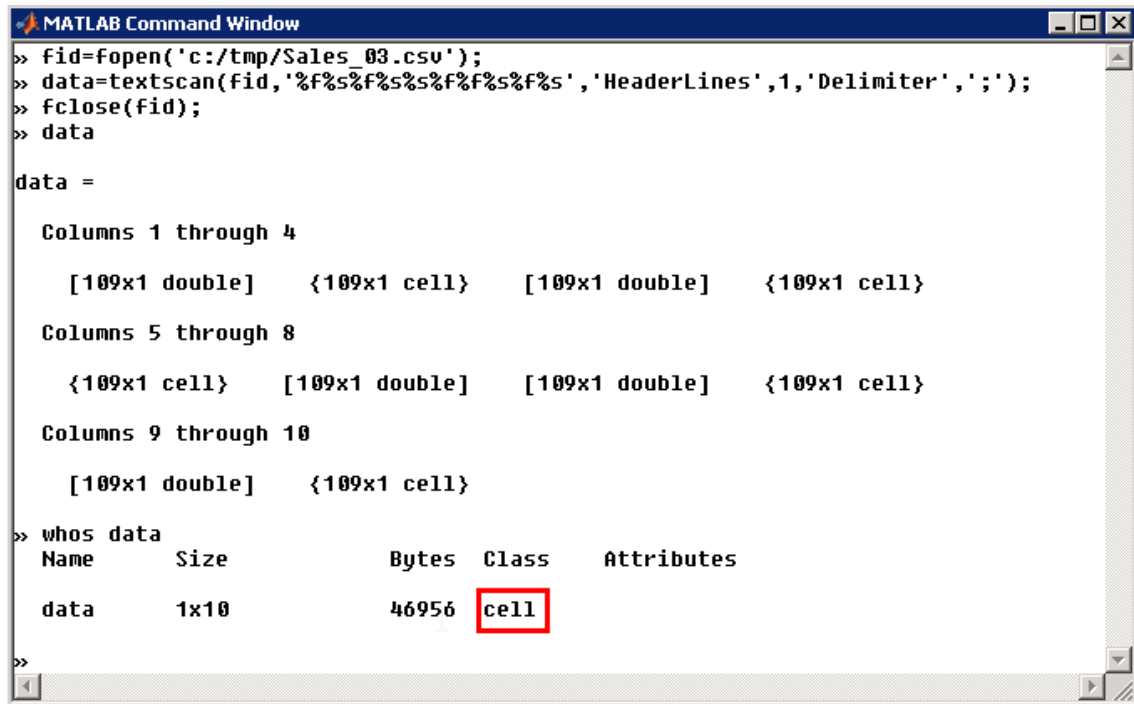
Client;Activite;NCommande;DateDeCommande;Article;Quantite;PrixParPiece;Rai
100;Assurances;1;03.01.2000;Compaq Presario 100;12;1650;1.50%;19503;oui
123;Machines/outils;2;03.01.2000;IBM 500;2;2299;0.00%;4598;oui
109;Éducation;3;03.01.2000;AST Intel 150;5;2690;0.00%;13450;oui
104;Éducation;4;03.01.2000;AST Intel 200;3;3190;0.00%;9570;oui
117;Banques;5;04.01.2000;Compaq Presario 100;13;1650;1.50%;21128.25;oui
103;Assurances;6;04.01.2000;AST Intel 150;2;2690;0.00%;5380;oui
104;Éducation;7;04.01.2000;AST Intel 200;2;3190;0.00%;6380;oui
111;Alimentaire;8;04.01.2000;IBM 500;4;2299;0.00%;9196;oui
113;Construction;9;04.01.2000;Compaq Presario 100;4;1650;0.00%;6600;oui
116;Pharmaceutique;10;04.01.2000;IBM 500;2;2299;0.00%;4598;oui
110;Distribution;11;05.01.2000;AST Intel 200;6;3190;1.50%;18852.9;oui
112;Machines/outils;12;05.01.2000;Compaq Presario 100;6;1650;1.50%;9751.5;
123;Machines/outils;13;05.01.2000;IBM 500;6;2299;1.50%;13587.09;oui
113;Construction;14;05.01.2000;AST Intel 150;3;2690;0.00%;8070;oui
115;Distribution;15;05.01.2000;Compaq Presario 100;8;1650;1.50%;13002;oui
124;Éducation;16;05.01.2000;AST Intel 200;8;3190;1.50%;25137.2;oui
124;Éducation;17;05.01.2000;Compaq Presario 100;11;1650;1.50%;17877.75;oui
106;Construction;18;05.01.2000;AST Intel 200;11;3190;1.50%;34563.65;oui
101;Construction;19;05.01.2000;Compaq Presario 100;14;1650;1.50%;22753.5;
116;Pharmaceutique;20;06.01.2000;IBM 500;7;2299;1.50%;15851.605;Non
112;Machines/outils;21;06.01.2000;AST Intel 150;6;2690;1.50%;15897.9;oui
125;Construction;22;06.01.2000;Compaq Presario 100;23;1650;3.00%;36811.5;
100;Assurances;23;06.01.2000;IBM 500;3;2299;0.00%;6897;oui
125;Construction;24;06.01.2000;AST Intel 200;2;3190;0.00%;6380;oui
104;Éducation;25;07.01.2000;AST Intel 150;12;2690;1.50%;31795.8;oui
126;Machines/outils;26;07.01.2000;AST Intel 150;24;2690;3.00%;62623.2;oui
121;Pharmaceutique;27;07.01.2000;IBM 500;8;2299;1.50%;18116.12;Non

```


Now the commands below tells **textscan()** how is each row in your csv file to be converted to. **%f** means "64 bit double" and **%s** means "string" so for example:

```
'%f%f%f%s%s'
```

means "3 doubles followed by 2 strings". For tab separated files the delimiter is **char(9)**



```

>> fid=fopen('c:/tmp/Sales_03.csv');
>> data=textscan(fid,'%F%s%f%s%f%s%f%s','HeaderLines',1,'Delimiter',';');
>> fclose(fid);
>> data

data =

Columns 1 through 4

    [109x1 double]    {109x1 cell}    [109x1 double]    {109x1 cell}

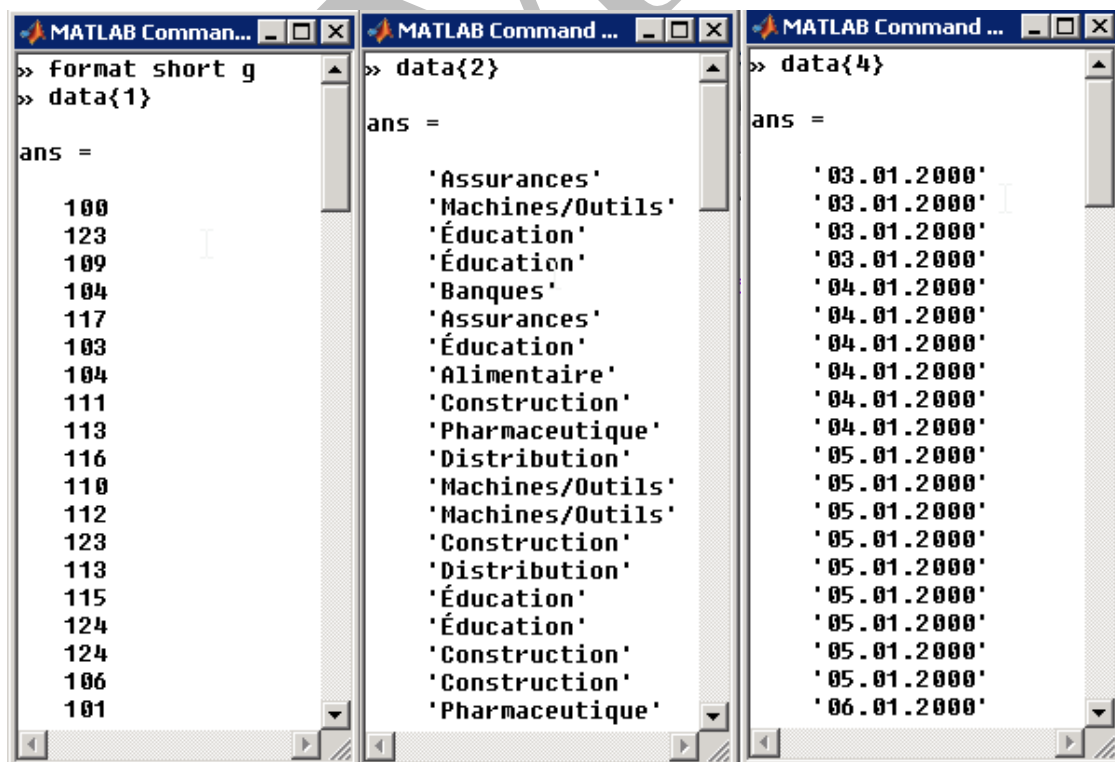
Columns 5 through 8

    {109x1 cell}    [109x1 double]    [109x1 double]    {109x1 cell}

Columns 9 through 10

    [109x1 double]    {109x1 cell}

>> whos data
Name      Size      Bytes  Class  Attributes
data      1x10      46956  cell
  
```



```

>> format short g
>> data{1}

ans =

    100
    123
    109
    104
    117
    103
    104
    111
    113
    116
    110
    112
    123
    113
    115
    124
    124
    106
    101

>> data{2}

ans =

'Assurances'
'Machines/Outils'
'Éducation'
'Éducation'
'Banques'
'Assurances'
'Éducation'
'Alimentaire'
'Construction'
'Pharmaceutique'
'Distribution'
'Machines/Outils'
'Machines/Outils'
'Construction'
'Distribution'
'Éducation'
'Éducation'
'Construction'
'Construction'
'Pharmaceutique'

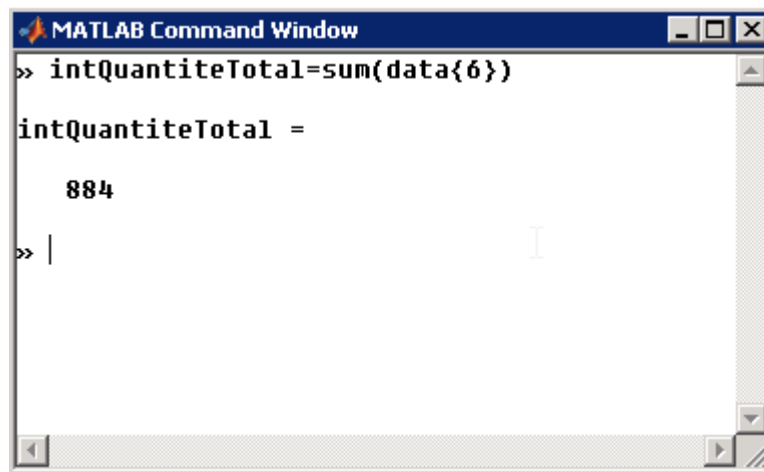
>> data{4}

ans =

'03.01.2000'
'03.01.2000'
'03.01.2000'
'03.01.2000'
'04.01.2000'
'04.01.2000'
'04.01.2000'
'04.01.2000'
'04.01.2000'
'04.01.2000'
'05.01.2000'
'05.01.2000'
'05.01.2000'
'05.01.2000'
'05.01.2000'
'05.01.2000'
'05.01.2000'
'05.01.2000'
'05.01.2000'
'06.01.2000'
  
```

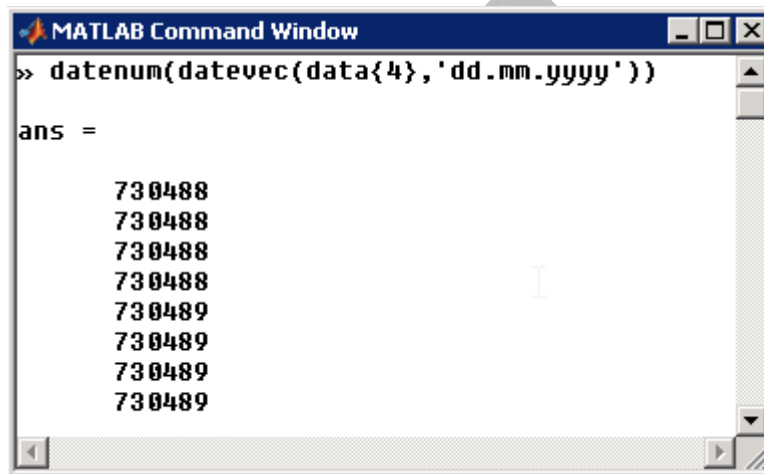
As you can see it the content is imported as a cell array!

We can style make calculations on numerical columns:

A screenshot of the MATLAB Command Window. The title bar reads "MATLAB Command Window". The command prompt shows the command `intQuantiteTotal=sum(data{6})` being entered. Below the command, the output is displayed as `intQuantiteTotal =` followed by the value `884`. The cursor is positioned at the end of the command line.

```
>> intQuantiteTotal=sum(data{6})  
  
intQuantiteTotal =  
  
    884  
  
>> |
```

And convert dates strings into real dates using classical conversion technics:

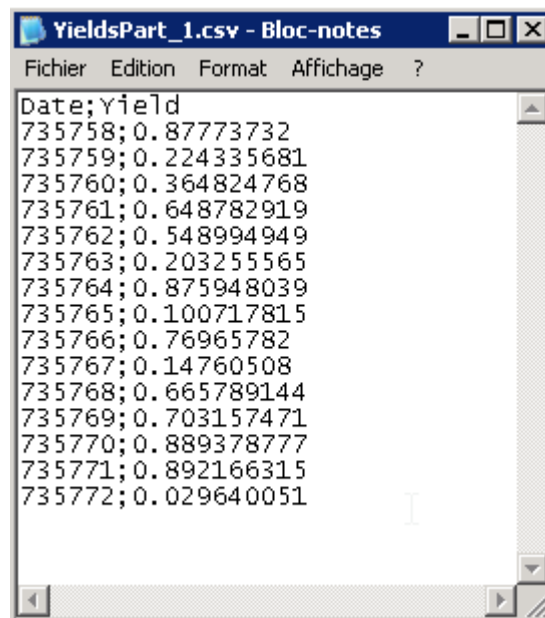
A screenshot of the MATLAB Command Window. The title bar reads "MATLAB Command Window". The command prompt shows the command `datenum(datevec(data{4}, 'dd.mm.yyyy'))` being entered. Below the command, the output is displayed as `ans =` followed by a column of eight values: `730488`, `730488`, `730488`, `730488`, `730489`, `730489`, `730489`, and `730489`. The cursor is positioned at the end of the command line.

```
>> datenum(datevec(data{4}, 'dd.mm.yyyy'))  
  
ans =  
  
    730488  
    730488  
    730488  
    730488  
    730489  
    730489  
    730489  
    730489
```

Merge csv files

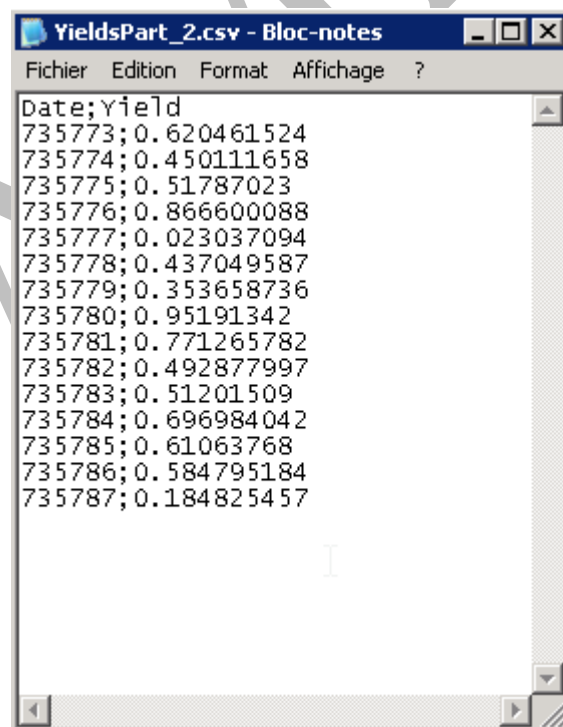
You can normally always merge various files content.

Suppose we have the following file:



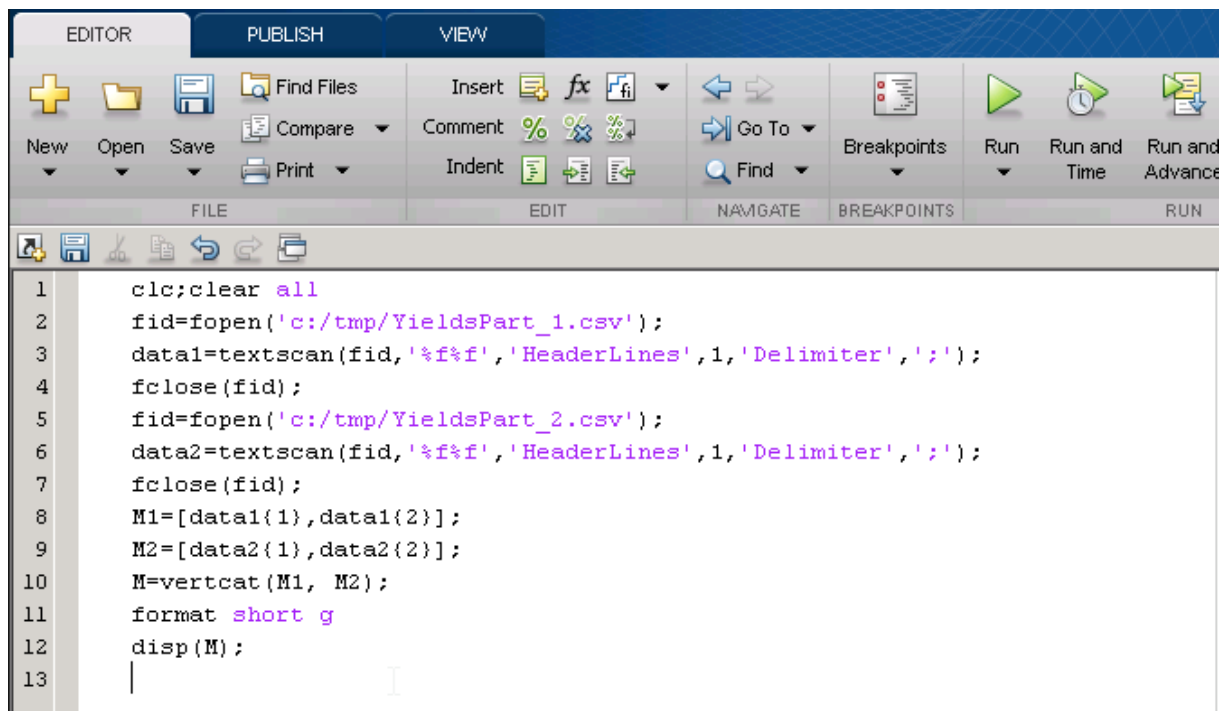
```
YieldsPart_1.csv - Bloc-notes
Fichier  Edition  Format  Affichage  ?
Date;Yield
735758;0.87773732
735759;0.224335681
735760;0.364824768
735761;0.648782919
735762;0.548994949
735763;0.203255565
735764;0.875948039
735765;0.100717815
735766;0.76965782
735767;0.14760508
735768;0.665789144
735769;0.703157471
735770;0.889378777
735771;0.892166315
735772;0.029640051
```

and another one:



```
YieldsPart_2.csv - Bloc-notes
Fichier  Edition  Format  Affichage  ?
Date;Yield
735773;0.620461524
735774;0.450111658
735775;0.51787023
735776;0.866600088
735777;0.023037094
735778;0.437049587
735779;0.353658736
735780;0.95191342
735781;0.771265782
735782;0.492877997
735783;0.51201509
735784;0.696984042
735785;0.61063768
735786;0.584795184
735787;0.184825457
```

We import the data and merge them together using classical matrices technics:

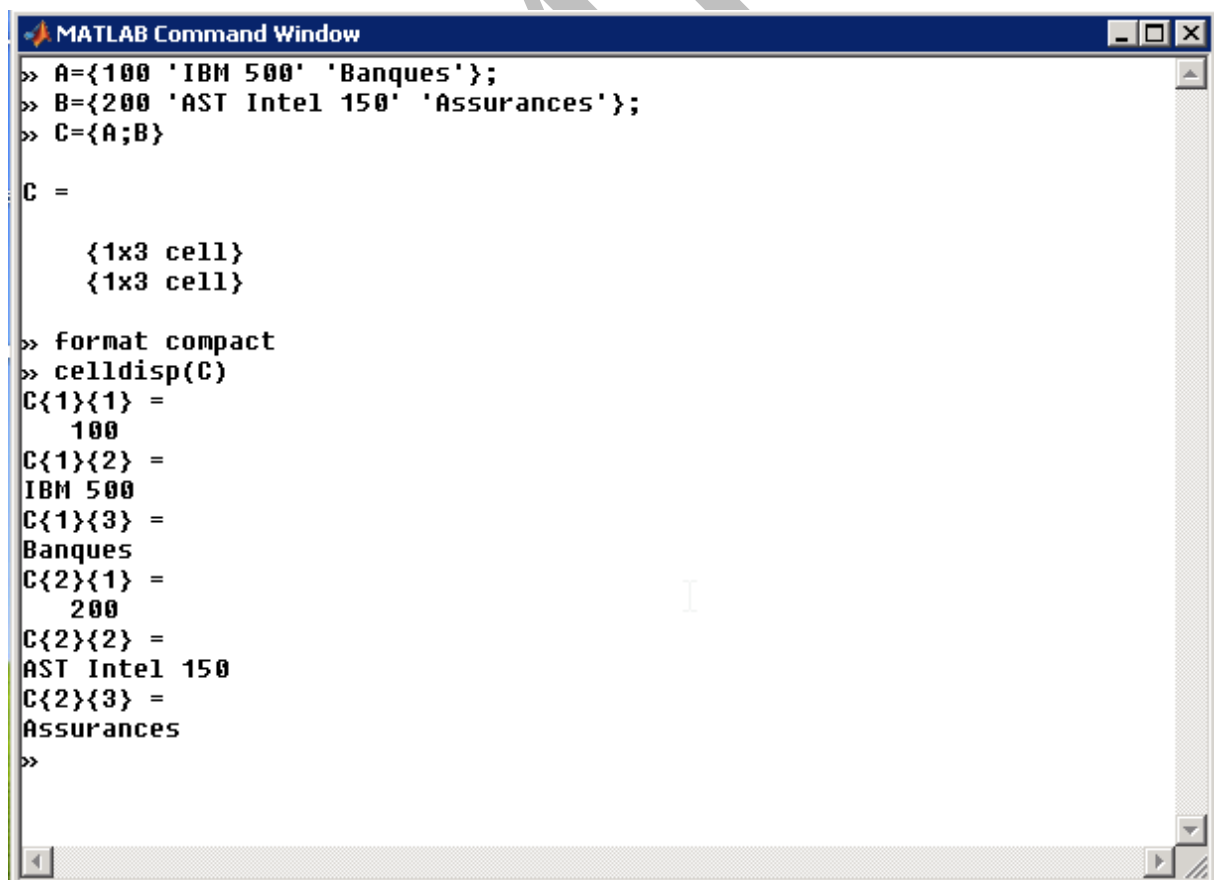


```

1  clc;clear all
2  fid=fopen('c:/tmp/YieldsPart_1.csv');
3  data1=textscan(fid,'%f%f','HeaderLines',1,'Delimiter',';');
4  fclose(fid);
5  fid=fopen('c:/tmp/YieldsPart_2.csv');
6  data2=textscan(fid,'%f%f','HeaderLines',1,'Delimiter',';');
7  fclose(fid);
8  M1=[data1{1},data1{2}];
9  M2=[data2{1},data2{2}];
10 M=vertcat(M1, M2);
11 format short g
12 disp(M);
13

```

The technic is the same for cell arrays. Here is a small example as refresh:



```

>> A={100 'IBM 500' 'Banques'};
>> B={200 'AST Intel 150' 'Assurances'};
>> C={A;B}

C =

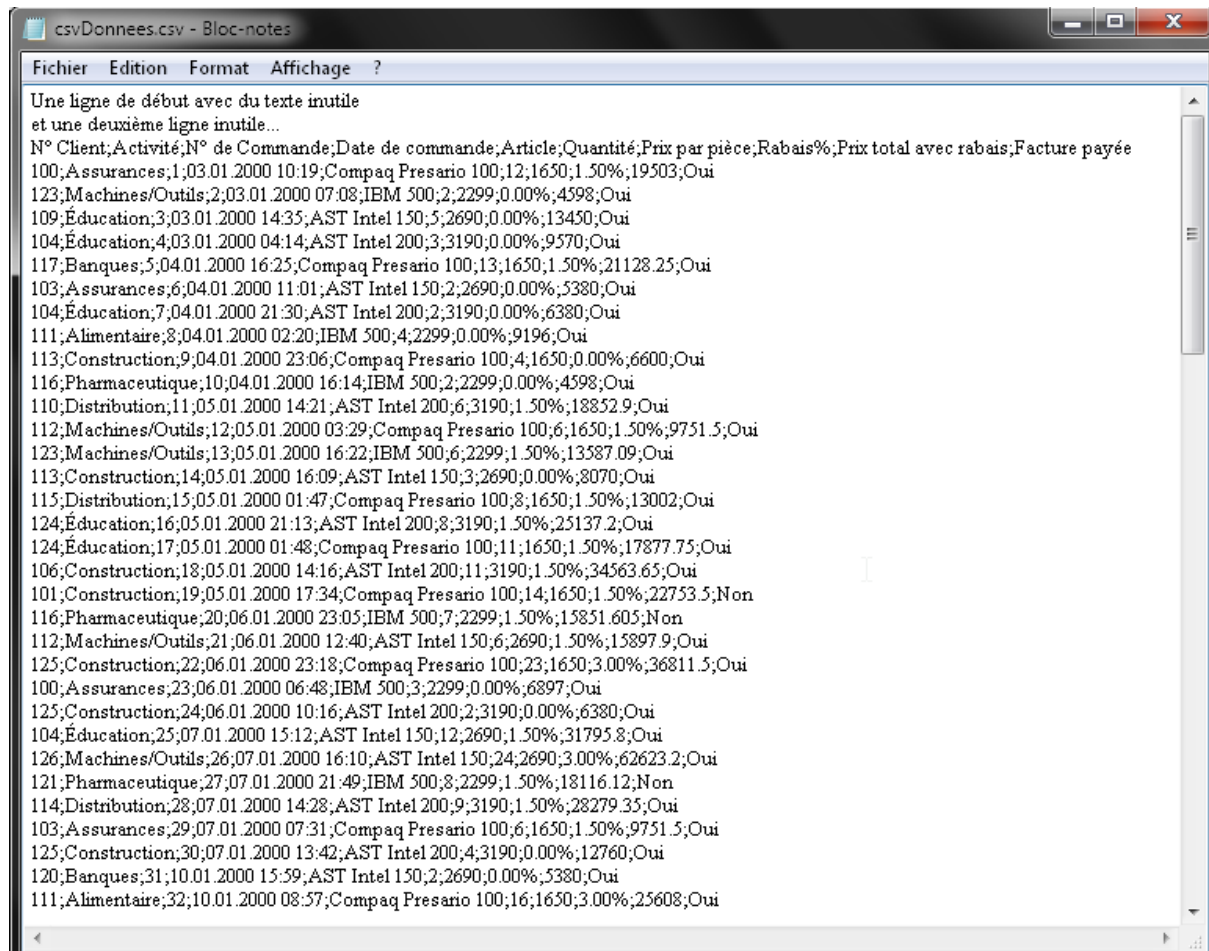
    {1x3 cell}
    {1x3 cell}

>> format compact
>> celldisp(C)
C{1}{1} =
    100
C{1}{2} =
    IBM 500
C{1}{3} =
    Banques
C{2}{1} =
    200
C{2}{2} =
    AST Intel 150
C{2}{3} =
    Assurances
>>

```

Import web csv files (with semi-colons, with headers, with text, with dates)

One very common task in academics labs is to import csv files from an URL. Suppose we have the following file on an external web server:

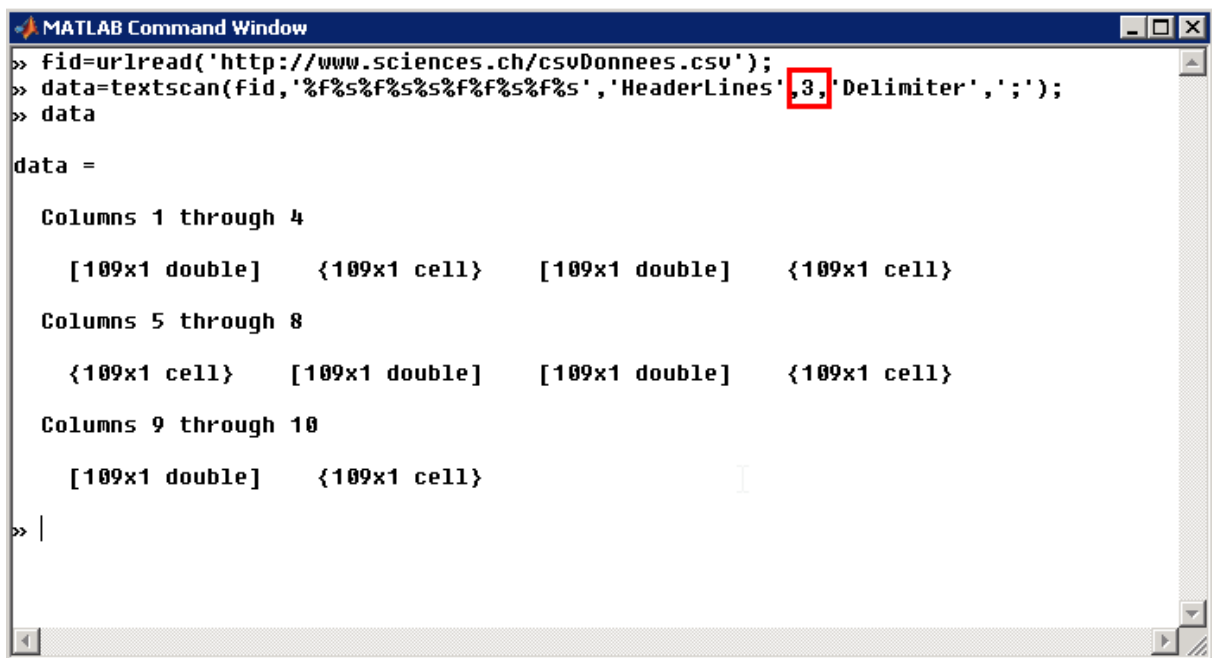


```

Une ligne de début avec du texte inutile
et une deuxième ligne inutile...
N° Client;Activité;N° de Commande;Date de commande;Article;Quantité;Prix par pièce;Rabais%;Prix total avec rabais;Facture payée
100;Assurances;1;03.01.2000 10:19;Compaq Presario 100;12;1650;1.50%;19503;Oui
123;Machines/Outils;2;03.01.2000 07:08;IBM 500;2;2299;0.00%;4598;Oui
109;Éducation;3;03.01.2000 14:35;AST Intel 150;5;2690;0.00%;13450;Oui
104;Éducation;4;03.01.2000 04:14;AST Intel 200;3;3190;0.00%;9570;Oui
117;Banques;5;04.01.2000 16:25;Compaq Presario 100;13;1650;1.50%;21128.25;Oui
103;Assurances;6;04.01.2000 11:01;AST Intel 150;2;2690;0.00%;5380;Oui
104;Éducation;7;04.01.2000 21:30;AST Intel 200;2;3190;0.00%;6380;Oui
111;Alimentaire;8;04.01.2000 02:20;IBM 500;4;2299;0.00%;9196;Oui
113;Construction;9;04.01.2000 23:06;Compaq Presario 100;4;1650;0.00%;6600;Oui
116;Pharmaceutique;10;04.01.2000 16:14;IBM 500;2;2299;0.00%;4598;Oui
110;Distribution;11;05.01.2000 14:21;AST Intel 200;6;3190;1.50%;18852.9;Oui
112;Machines/Outils;12;05.01.2000 03:29;Compaq Presario 100;6;1650;1.50%;9751.5;Oui
123;Machines/Outils;13;05.01.2000 16:22;IBM 500;6;2299;1.50%;13587.09;Oui
113;Construction;14;05.01.2000 16:09;AST Intel 150;3;2690;0.00%;8070;Oui
115;Distribution;15;05.01.2000 01:47;Compaq Presario 100;8;1650;1.50%;13002;Oui
124;Éducation;16;05.01.2000 21:13;AST Intel 200;8;3190;1.50%;25137.2;Oui
124;Éducation;17;05.01.2000 01:48;Compaq Presario 100;11;1650;1.50%;17877.75;Oui
106;Construction;18;05.01.2000 14:16;AST Intel 200;11;3190;1.50%;34563.65;Oui
101;Construction;19;05.01.2000 17:34;Compaq Presario 100;14;1650;1.50%;22753.5;Non
116;Pharmaceutique;20;06.01.2000 23:05;IBM 500;7;2299;1.50%;15851.605;Non
112;Machines/Outils;21;06.01.2000 12:40;AST Intel 150;6;2690;1.50%;15897.9;Oui
125;Construction;22;06.01.2000 23:18;Compaq Presario 100;23;1650;3.00%;36811.5;Oui
100;Assurances;23;06.01.2000 06:48;IBM 500;3;2299;0.00%;6897;Oui
125;Construction;24;06.01.2000 10:16;AST Intel 200;2;3190;0.00%;6380;Oui
104;Éducation;25;07.01.2000 15:12;AST Intel 150;12;2690;1.50%;31795.8;Oui
126;Machines/Outils;26;07.01.2000 16:10;AST Intel 150;24;2690;3.00%;62623.2;Oui
121;Pharmaceutique;27;07.01.2000 21:49;IBM 500;8;2299;1.50%;18116.12;Non
114;Distribution;28;07.01.2000 14:28;AST Intel 200;9;3190;1.50%;28279.35;Oui
103;Assurances;29;07.01.2000 07:31;Compaq Presario 100;6;1650;1.50%;9751.5;Oui
125;Construction;30;07.01.2000 13:42;AST Intel 200;4;3190;0.00%;12760;Oui
120;Banques;31;10.01.2000 15:59;AST Intel 150;2;2690;0.00%;5380;Oui
111;Alimentaire;32;10.01.2000 08:57;Compaq Presario 100;16;1650;3.00%;25608;Oui

```

To import it we just use the function **urlread()** associated to the function **textscan()** and we get a cell array:



```

MATLAB Command Window
>> fid=urlread('http://www.sciences.ch/csvDonnees.csv');
>> data=textscan(fid,'%f%s%f%s%f%s%f%s','HeaderLines',3,'Delimiter',';');
>> data

data =

Columns 1 through 4

    [109x1 double]    {109x1 cell}    [109x1 double]    {109x1 cell}

Columns 5 through 8

    {109x1 cell}    [109x1 double]    [109x1 double]    {109x1 cell}

Columns 9 through 10

    [109x1 double]    {109x1 cell}

>> |

```

Import from Yahoo Finance

Without scripting or without Toolboxes (especially the Mathworks Datafeed Toolbox) importing data from yahoo finance is annoying.

First see some descriptions:

<https://code.google.com/p/yahoo-finance-managed/wiki/csvQuotesDownload>

And here is an example:



```

MATLAB Command Window
>> %last value Google
>> fid=urlread('http://download.finance.yahoo.com/d/quotes.csv?s=%40%5EDJI,G00G&f=ns11op')
fid =
"Google Inc.", "G00G", 567.88, 569.99, 568.77

>> %many values Apple
>> fid=urlread('http://ichart.finance.yahoo.com/table.csv?s=aapl&a=00&b=01&c=2011&d=03&e=04&f=&ignore=.csv');
>> size(fid)
ans =
     1    49274

>> fid(1,1:10)
ans =
Date,Open,
>> fid(1,1:100)
ans =
Date,Open,High,Low,Close,Volume,Adj Close
2014-08-11,95.27,96.08,94.84,95.99,36490300,95.99
2014-08-
>> %we need a script to clean this text vector...

```

Here is the most esthetic way I have found until today for this purpose:

```

data.csv - Bloc-notes
Fichier Edition Format Affichage ?
Date,Open,High,Low,Close,volume,Adj Close2014-08-29,102.86,102.90,102.20,102.50,44567000,102.
000,94.652014-08-04,96.37,96.58,95.17,95.59,39958000,95.122014-08-01,94.90,96.62,94.81,96.13
14,95.99,94.10,95.97,56468000,95.502014-07-03,93.67,94.10,93.20,94.03,22891800,93.562014-07-
7,87484600,91.772014-06-05,646.20,649.37,642.61,647.35,75951400,92.022014-06-04,637.44,647.8
5.54,72899400,83.232014-05-08,588.25,594.41,586.40,587.99,57574300,83.582014-05-07,595.25,59
14,519.61,67929400,73.452014-04-10,530.68,532.24,523.17,523.48,59913000,74.002014-04-09,522.
5.85,526.74,49886200,74.462014-03-14,528.79,530.89,523.00,524.69,59299800,74.172014-03-13,53
545.61,545.99,65062900,77.182014-02-14,542.47,545.98,541.21,543.99,68231100,76.902014-02-13,
99,550.07,540.42,549.07,82131700,77.152014-01-17,551.48,552.07,539.90,540.67,106684900,75.971
,545.43,551.61,544.82,549.02,109103400,77.152013-12-19,549.50,550.00,543.73,544.46,80077200,7
13-11-22,519.52,522.16,518.53,519.80,55931400,73.042013-11-21,517.60,521.21,513.67,521.14,651
2013-10-28,529.04,531.00,523.21,529.88,137610200,74.032013-10-25,531.32,533.23,525.11,525.96,
.3902013-10-01,478.45,489.14,478.38,487.96,88470900,68.172013-09-30,477.25,481.66,474.41,476.
9091900,69.192013-09-04,499.56,502.24,496.28,498.69,86258200,69.672013-09-03,493.10,500.60,4
61.01,63944300,64.402013-08-07,463.80,467.00,461.77,464.98,74714500,64.532013-08-06,468.02,4
1,426.51,69890800,59.192013-07-11,422.95,428.25,421.17,427.29,81573100,59.302013-07-10,419.6
,428.50,430.05,67966500,59.692013-06-13,432.50,437.14,428.75,435.96,71458100,60.512013-06-12
5,440.09,431.01,433.26,106976100,60.132013-05-16,423.24,437.85,418.90,434.58,150801000,60.312
2013-04-22,392.64,402.20,391.27,398.67,107480100,54.972013-04-19,387.97,399.60,385.10,390.53,
0,63.582013-03-25,464.69,469.95,461.78,463.58,125283900,63.922013-03-22,454.58,462.10,453.11
444.57,146837600,61.302013-02-26,443.82,451.54,437.66,448.97,125374900,61.902013-02-25,453.8
0,462.60,454.50,456.83,104288800,62.622013-01-29,458.50,460.20,452.12,458.27,142789500,62.821
312013-01-02,553.82,555.00,541.63,549.03,140129500,75.262012-12-31,510.53,535.40,509.00,532.
8.79,261159500,73.862012-12-04,581.80,581.80,572.13,575.85,139267100,78.942012-12-03,593.65,
,574.54,555.55,558.00,198412200,76.492012-11-06,590.23,590.74,580.09,582.85,93729300,79.53201
2012-10-09,638.65,640.49,623.55,635.85,209649300,86.762012-10-08,646.88,647.56,636.11,638.17,
149590000,93.192012-09-12,666.85,669.90,656.00,669.79,178058300,91.392012-09-11,665.11,670.1
30.50,636.34,63633500,86.832012-08-15,631.30,634.00,627.75,630.83,64335600,86.082012-08-14,6
614.44,603.70,604.30,99367800,82.112012-07-19,611.28,615.35,606.00,614.32,109215400,83.47201
22,579.04,582.19,575.42,582.10,71117900,79.092012-06-21,585.44,588.22,577.44,577.67,81587800,
502012-05-25,564.59,565.85,558.47,562.29,82126800,76.402012-05-24,575.87,576.50,561.23,565.32
3,152749800,79.092012-04-30,597.80,598.40,583.00,583.98,126536200,79.342012-04-27,605.07,606

```

```

MATLAB Command Window
>> url='http://ichart.finance.yahoo.com/table.csv?s=aapl&a=00&b=01&c=2011&d=03&e=04&f=&ignore=.csv';
>> fid=urlread(url);
>> urlwrite(url,'C:\tmp\data.csv');
>> mydata=dataset('File','C:\tmp\data.csv','Delimiter','');
Warning: Variable names were modified to make them valid MATLAB identifiers.
> In genvalidnames at 81
   In @dataset\private\setvarnames at 40
   In dataset.readFile at 237
   In dataset.dataset>dataset.dataset at 351
>> mydata(1:10,1:6)

ans =

    Date      Open      High      Low      Close      Volume
    '2014-08-29'    102.86    102.9    102.2    102.5    4.4567e+07
    '2014-08-28'    101.59    102.78    101.56    102.25    6.839e+07
    '2014-08-27'    101.02    102.57    100.7    102.13    4.6827e+07
    '2014-08-26'    101.42    101.5    100.86    100.89    3.312e+07
    '2014-08-25'    101.79    102.17    101.28    101.54    4.0145e+07
    '2014-08-22'    100.29    101.47    100.19    101.32    4.4102e+07
    '2014-08-21'    100.57    100.94    100.11    100.58    3.3422e+07
    '2014-08-20'    100.44    101.09    99.95    100.57    5.2613e+07
    '2014-08-19'    99.41    100.68    99.32    100.53    6.9275e+07
    '2014-08-18'    98.49    99.37    97.98    99.16    4.7572e+07

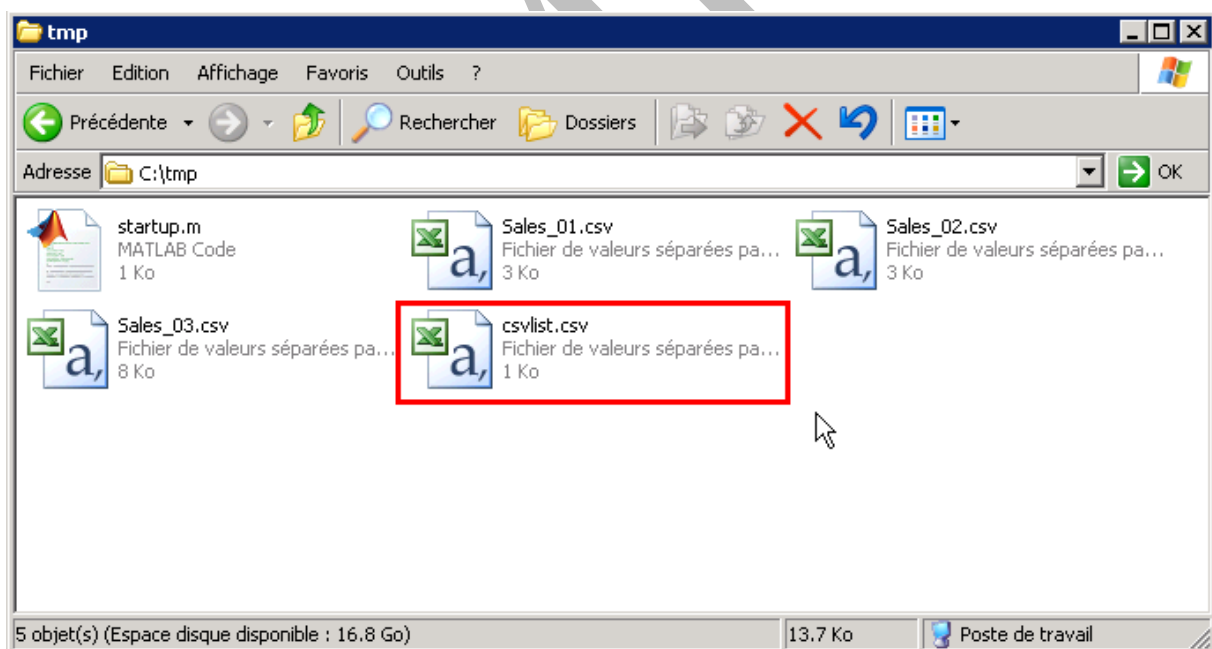
```

Export to csv file

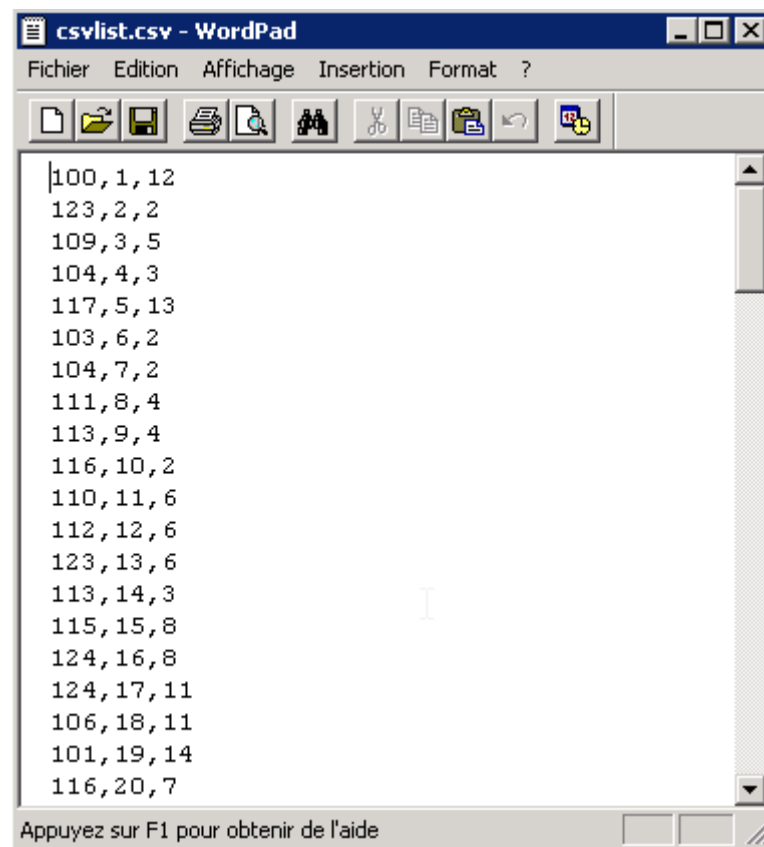
Now also something very important is to export manipulated data to a *.csv file. This can be done easily by using the function **csvwrite()**:



The file will then be created:



And we will have the correct content:



```
100,1,12
123,2,2
109,3,5
104,4,3
117,5,13
103,6,2
104,7,2
111,8,4
113,9,4
116,10,2
110,11,6
112,12,6
123,13,6
113,14,3
115,15,8
124,16,8
124,17,11
106,18,11
101,19,14
116,20,7
```

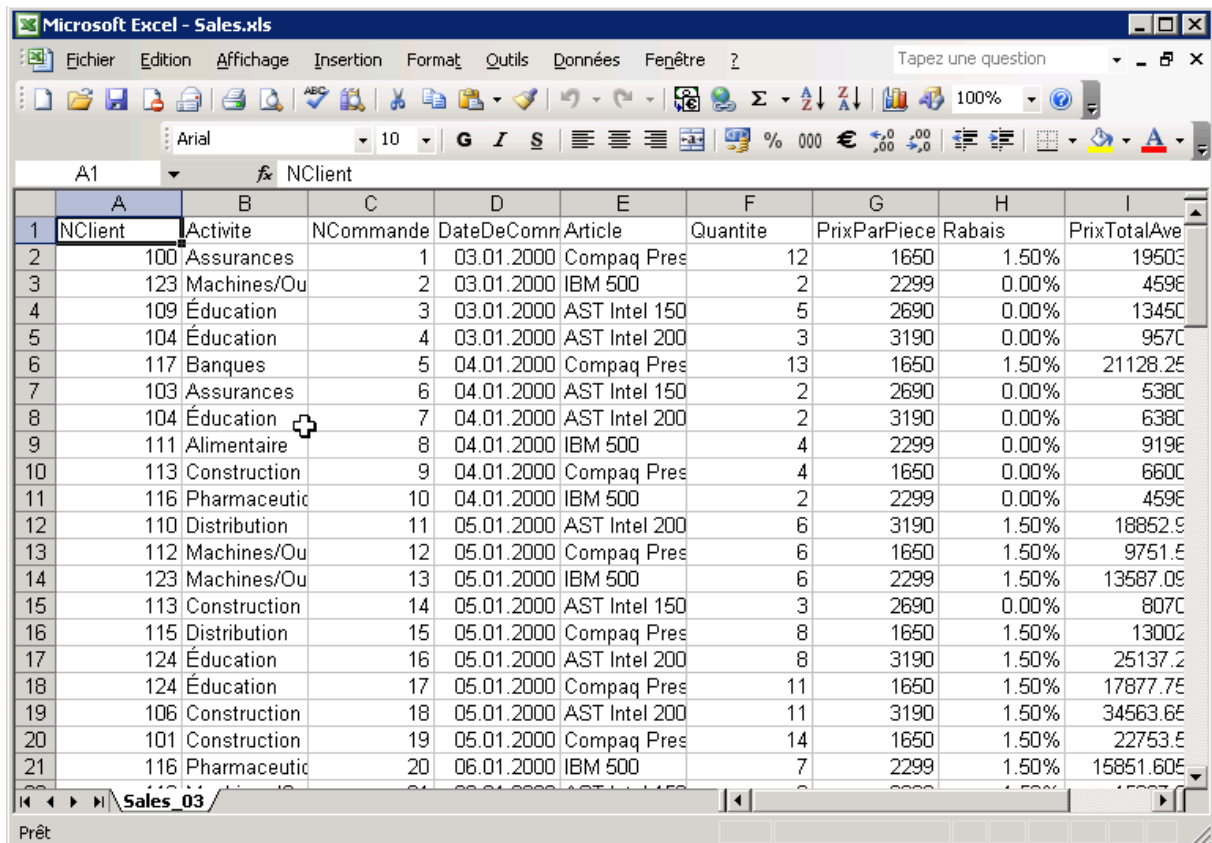
Appuyez sur F1 pour obtenir de l'aide

Import/Export xls/xlsx (Microsoft Office Excel) files

We will see here how to import *.xls or *.xlsx files without using the import wizard of MATLAB but just by using the *Command Window*.

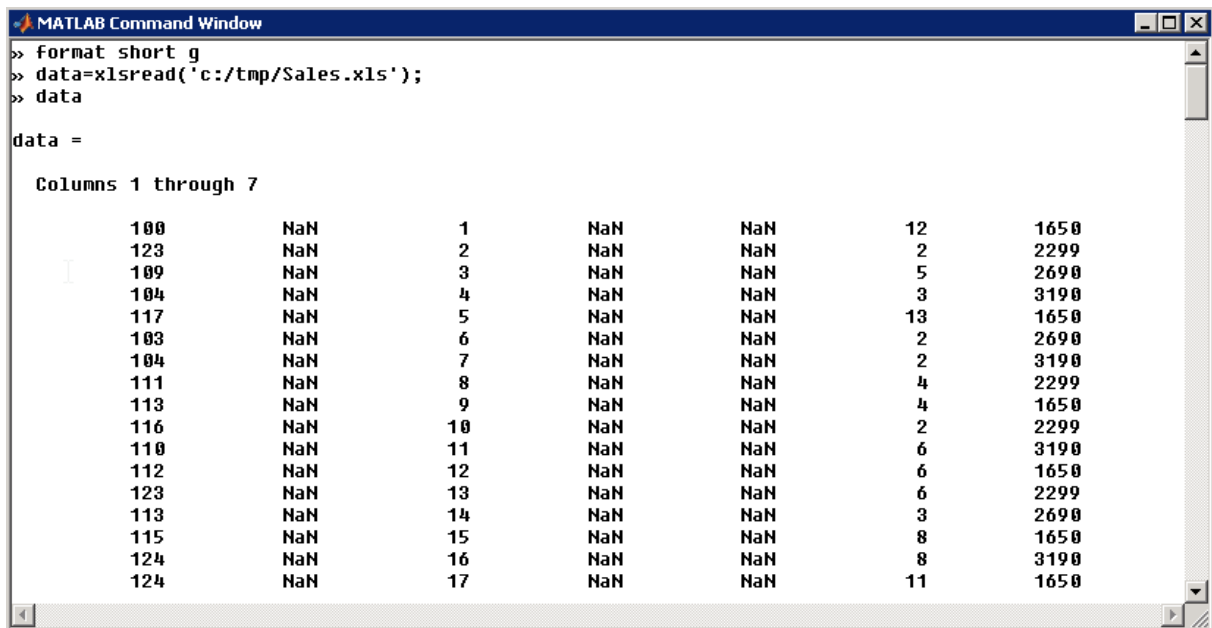
Import xls/xlsx local files

Suppose we have the following Microsoft Excel file:



	A	B	C	D	E	F	G	H	I
	NClient	Activite	NCommande	DateDeComn	Article	Quantite	PrixParPiece	Rabais	PrixTotalAve
1	100	Assurances	1	03.01.2000	Compaq Pres	12	1650	1.50%	19500
2	123	Machines/Ou	2	03.01.2000	IBM 500	2	2299	0.00%	4598
3	109	Éducation	3	03.01.2000	AST Intel 150	5	2690	0.00%	13450
4	104	Éducation	4	03.01.2000	AST Intel 200	3	3190	0.00%	9570
5	117	Banques	5	04.01.2000	Compaq Pres	13	1650	1.50%	21128.25
6	103	Assurances	6	04.01.2000	AST Intel 150	2	2690	0.00%	5380
7	104	Éducation	7	04.01.2000	AST Intel 200	2	3190	0.00%	6380
8	111	Alimentaire	8	04.01.2000	IBM 500	4	2299	0.00%	9196
9	113	Construction	9	04.01.2000	Compaq Pres	4	1650	0.00%	6600
10	116	Pharmaceutic	10	04.01.2000	IBM 500	2	2299	0.00%	4598
11	110	Distribution	11	05.01.2000	AST Intel 200	6	3190	1.50%	18852.9
12	112	Machines/Ou	12	05.01.2000	Compaq Pres	6	1650	1.50%	9751.5
13	123	Machines/Ou	13	05.01.2000	IBM 500	6	2299	1.50%	13587.09
14	113	Construction	14	05.01.2000	AST Intel 150	3	2690	0.00%	8070
15	115	Distribution	15	05.01.2000	Compaq Pres	8	1650	1.50%	13002
16	124	Éducation	16	05.01.2000	AST Intel 200	8	3190	1.50%	25137.2
17	124	Éducation	17	05.01.2000	Compaq Pres	11	1650	1.50%	17877.75
18	106	Construction	18	05.01.2000	AST Intel 200	11	3190	1.50%	34563.65
19	101	Construction	19	05.01.2000	Compaq Pres	14	1650	1.50%	22753.5
20	116	Pharmaceutic	20	06.01.2000	IBM 500	7	2299	1.50%	15851.605
21	110	Distribution	21	06.01.2000	AST Intel 150	3	2690	0.00%	8070

To import simple the content but without titles and string columns we can simply use the **xlsread()** function:



```

MATLAB Command Window
>> format short g
>> data=xlsread('c:/tmp/Sales.xls');
>> data

data =

Columns 1 through 7

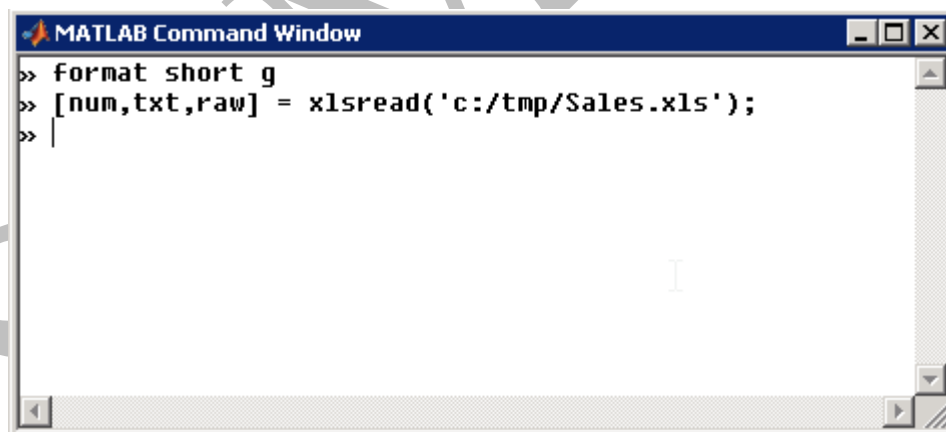
    100      NaN         1      NaN      NaN         12      1650
    123      NaN         2      NaN      NaN         2      2299
    109      NaN         3      NaN      NaN         5      2690
    104      NaN         4      NaN      NaN         3      3190
    117      NaN         5      NaN      NaN        13      1650
    103      NaN         6      NaN      NaN         2      2690
    104      NaN         7      NaN      NaN         2      3190
    111      NaN         8      NaN      NaN         4      2299
    113      NaN         9      NaN      NaN         4      1650
    116      NaN        10      NaN      NaN         2      2299
    110      NaN        11      NaN      NaN         6      3190
    112      NaN        12      NaN      NaN         6      1650
    123      NaN        13      NaN      NaN         6      2299
    113      NaN        14      NaN      NaN         3      2690
    115      NaN        15      NaN      NaN         8      1650
    124      NaN        16      NaN      NaN         8      3190
    124      NaN        17      NaN      NaN        11      1650

```

If necessary you can specify the sheet to read by writing:

```
myData = xlsread('c:/tmp/Sales.xls','Sheet1').
```

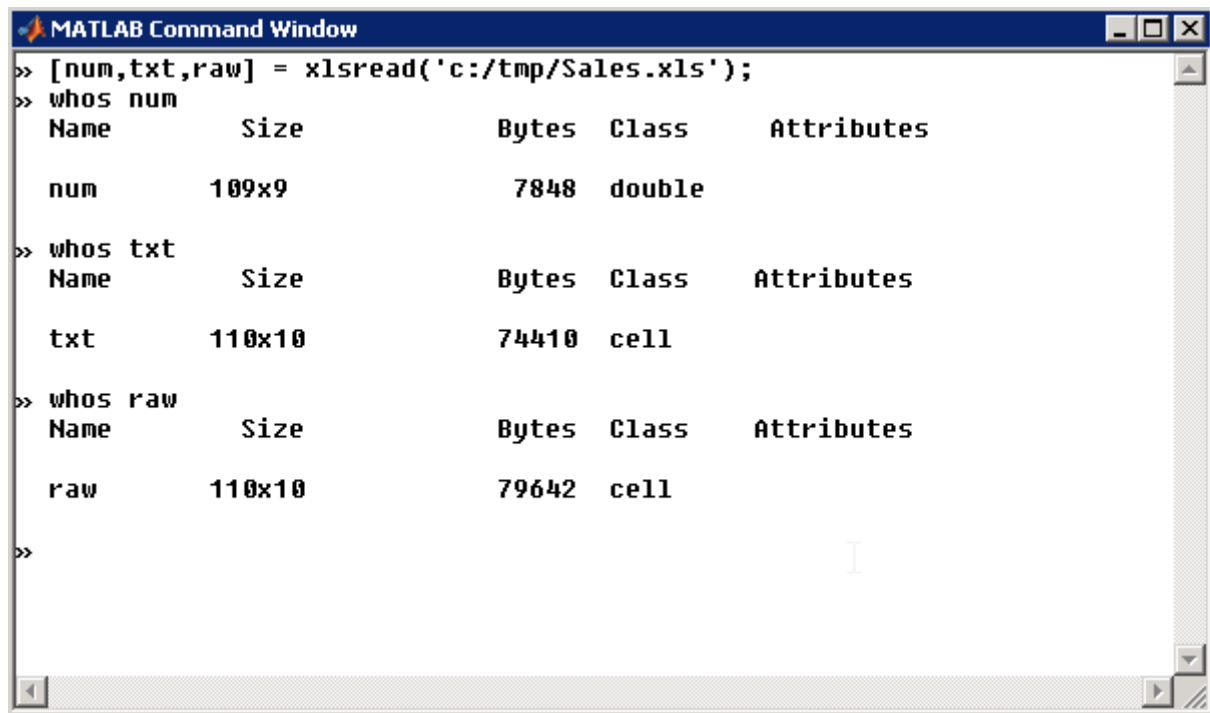
This is an issue... but in fact there is a workaround because the function **xlsread()** returns multiples outputs as you can see below:



```

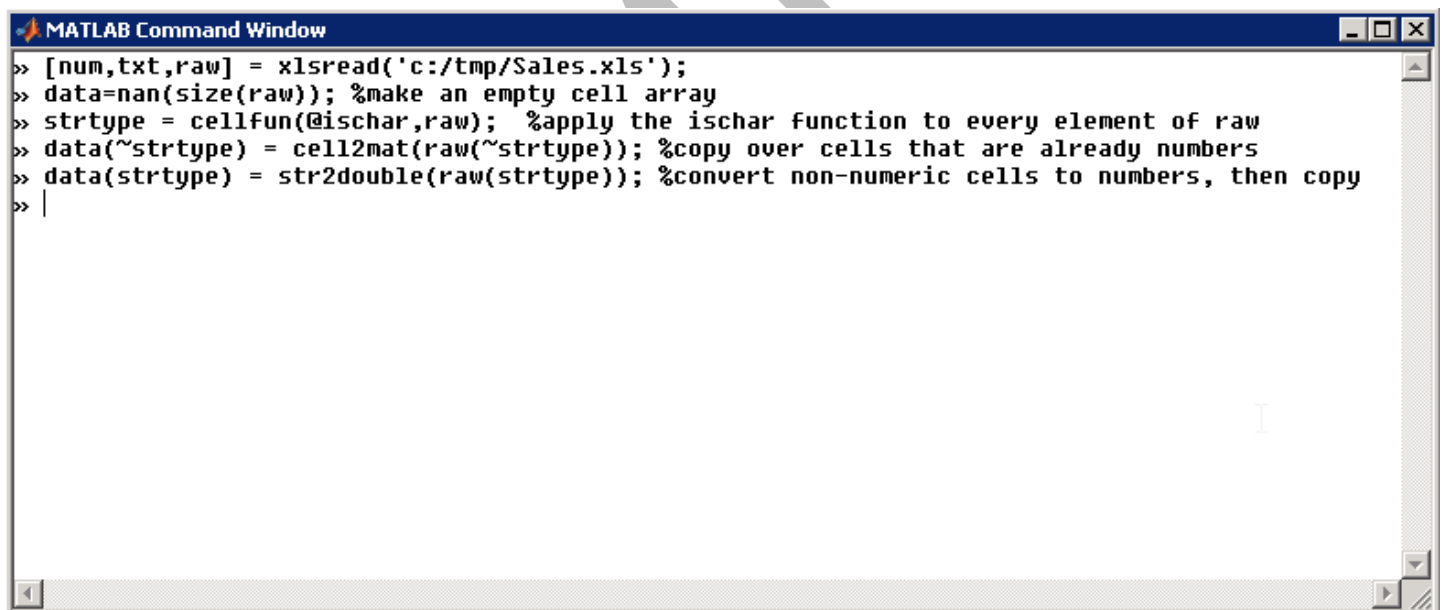
MATLAB Command Window
>> format short g
>> [num,txt,row] = xlsread('c:/tmp/Sales.xls');
>> |

```



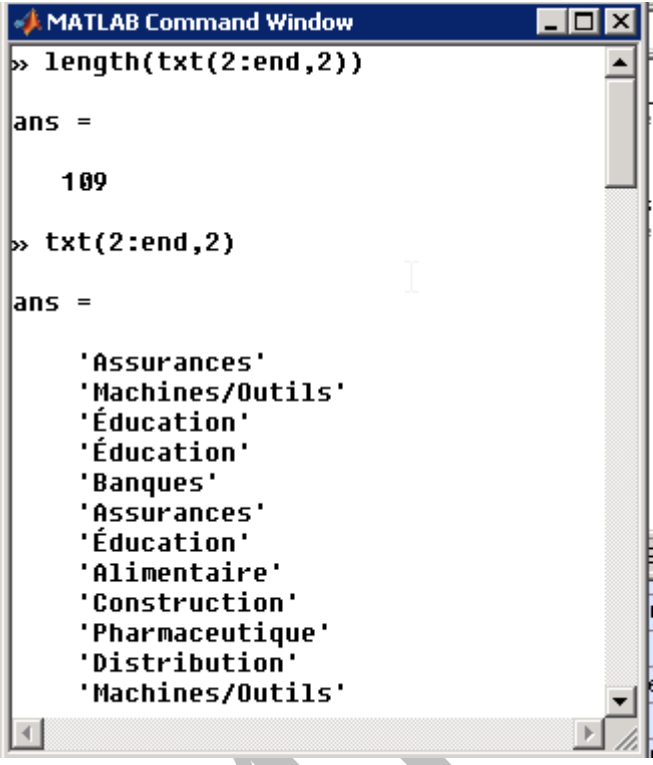
```
MATLAB Command Window
>> [num,txt,row] = xlsread('c:/tmp/Sales.xls');
>> whos num
Name      Size      Bytes  Class  Attributes
num       109x9       7848   double
>> whos txt
Name      Size      Bytes  Class  Attributes
txt       110x10      74410  cell
>> whos row
Name      Size      Bytes  Class  Attributes
row       110x10      79642  cell
>>
```

Now we will use some tricks that are not very intuitive to merge quickly everything in a single cell array:



```
MATLAB Command Window
>> [num,txt,row] = xlsread('c:/tmp/Sales.xls');
>> data=nan(size(row)); %make an empty cell array
>> strtype = cellfun(@ischar,row); %apply the ischar function to every element of row
>> data(~strtype) = cell2mat(row(~strtype)); %copy over cells that are already numbers
>> data(strtype) = str2double(row(strtype)); %convert non-numeric cells to numbers, then copy
>> |
```

Otherwise you can access to the content of the *text* cell array if you don't want to clean the data:

A screenshot of the MATLAB Command Window. The first command is `length(txt(2:end,2))`, which returns `ans = 109`. The second command is `txt(2:end,2)`, which returns a cell array of 10 rows, each containing a category name: 'Assurances', 'Machines/Outils', 'Éducation', 'Éducation', 'Banques', 'Assurances', 'Éducation', 'Alimentaire', 'Construction', 'Pharmaceutique', 'Distribution', and 'Machines/Outils'.

```
MATLAB Command Window
>> length(txt(2:end,2))

ans =

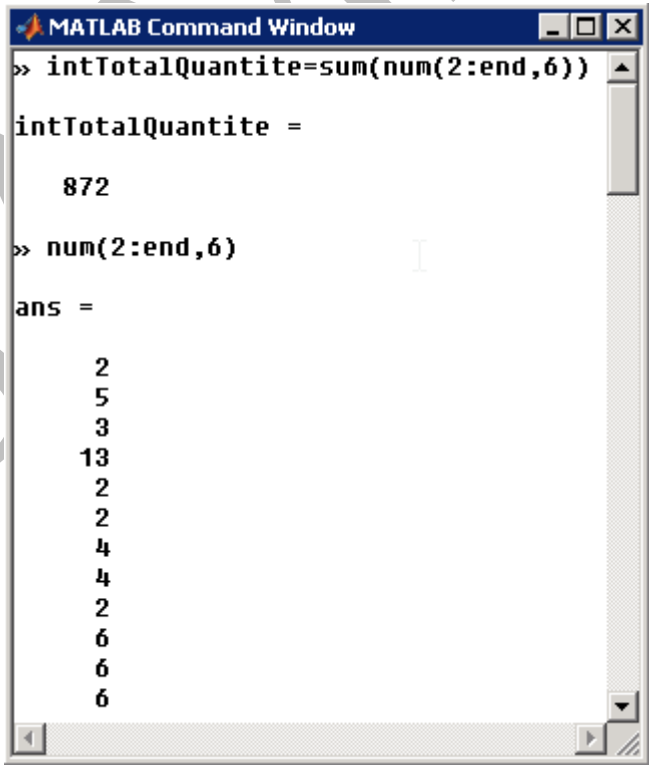
    109

>> txt(2:end,2)

ans =

    'Assurances'
    'Machines/Outils'
    'Éducation'
    'Éducation'
    'Banques'
    'Assurances'
    'Éducation'
    'Alimentaire'
    'Construction'
    'Pharmaceutique'
    'Distribution'
    'Machines/Outils'
```

Or directly to the *num* content for calculations:

A screenshot of the MATLAB Command Window. The first command is `intTotalQuantite=sum(num(2:end,6))`, which returns `intTotalQuantite = 872`. The second command is `num(2:end,6)`, which returns a column vector of 12 numerical values: 2, 5, 3, 13, 2, 2, 4, 4, 2, 6, 6, and 6.

```
MATLAB Command Window
>> intTotalQuantite=sum(num(2:end,6))

intTotalQuantite =

    872

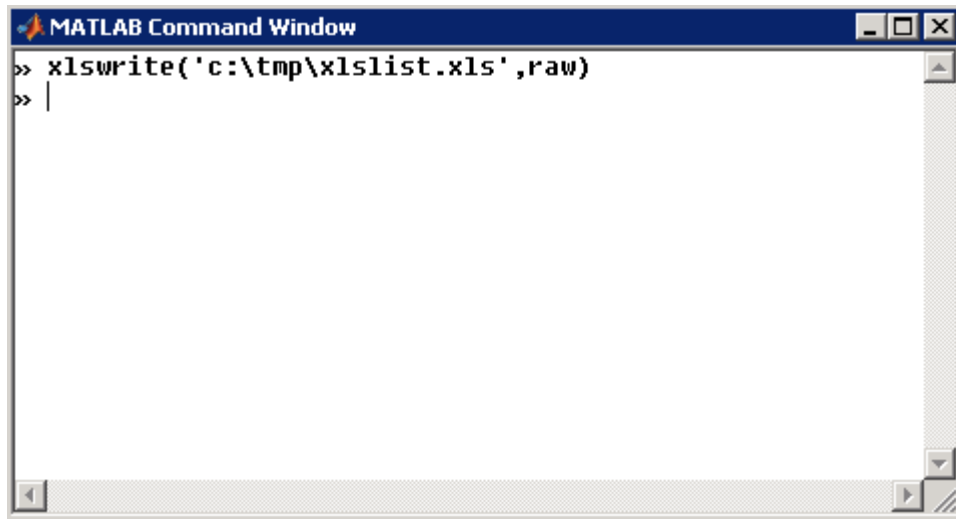
>> num(2:end,6)

ans =

     2
     5
     3
    13
     2
     2
     4
     4
     2
     6
     6
     6
```

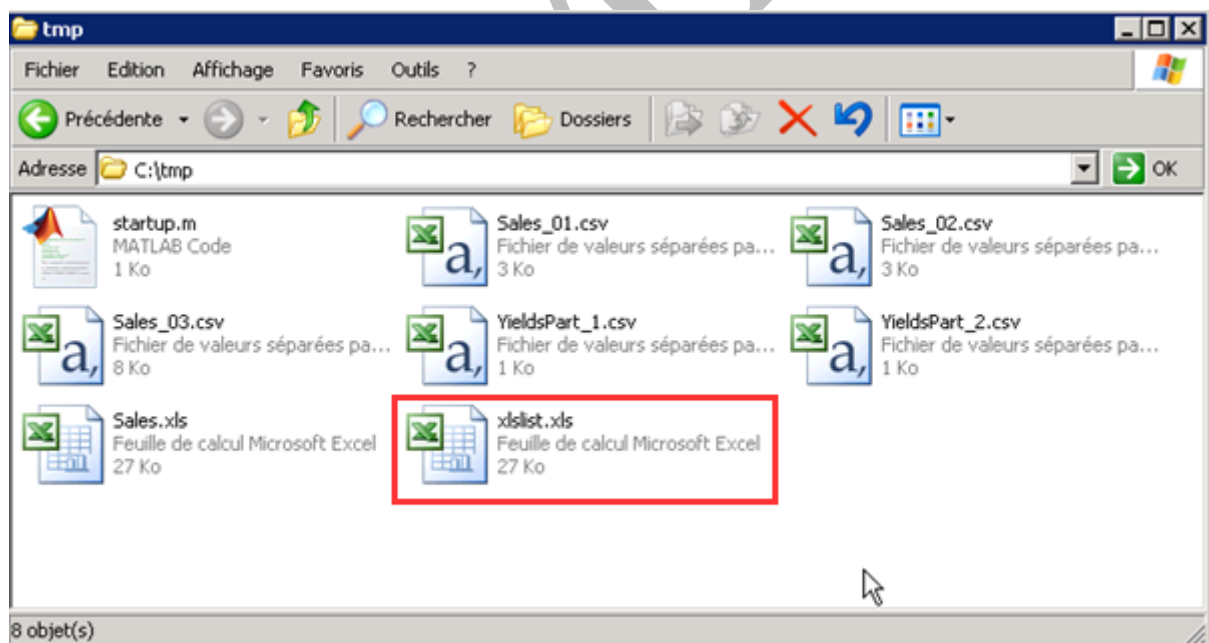
Export to xls/xlsx files

Now also something very important is to export manipulated data to a *.csv file. This can be done easily by using the function **xlswrite()**:



Note: To write in a specific sheet in a specific range just write:

The file will then be created:



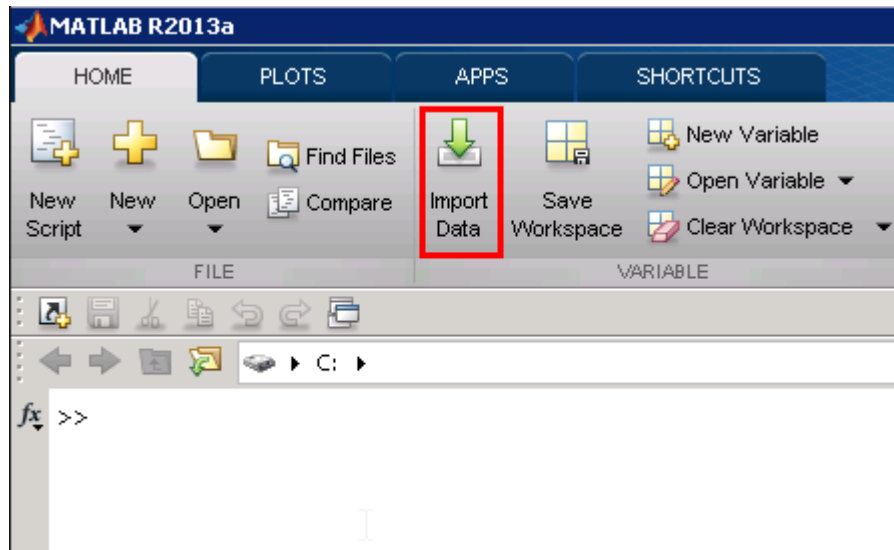
And we will have the correct content:

	A	B	C	D	E	F	G	H	I
	NClient	Activite	NCommande	DateDeComm	Article	Quantite	PrixParPiece	Rabais	PrixTotalAve
2	100	Assurances	1	03.01.2000	Compaq Pres	12	1650	0.015	19503
3	123	Machines/Ou	2	03.01.2000	IBM 500	2	2299	0	4598
4	109	Éducation	3	03.01.2000	AST Intel 150	5	2690	0	13450
5	104	Éducation	4	03.01.2000	AST Intel 200	3	3190	0	9570
6	117	Banques	5	04.01.2000	Compaq Pres	13	1650	0.015	21128.25
7	103	Assurances	6	04.01.2000	AST Intel 150	2	2690	0	5380
8	104	Éducation	7	04.01.2000	AST Intel 200	2	3190	0	6380
9	111	Alimentaire	8	04.01.2000	IBM 500	4	2299	0	9196
10	113	Construction	9	04.01.2000	Compaq Pres	4	1650	0	6600
11	116	Pharmaceutic	10	04.01.2000	IBM 500	2	2299	0	4598
12	110	Distribution	11	05.01.2000	AST Intel 200	6	3190	0.015	18852.9
13	112	Machines/Ou	12	05.01.2000	Compaq Pres	6	1650	0.015	9751.5
14	123	Machines/Ou	13	05.01.2000	IBM 500	6	2299	0.015	13587.09
15	113	Construction	14	05.01.2000	AST Intel 150	3	2690	0	8070
16	115	Distribution	15	05.01.2000	Compaq Pres	8	1650	0.015	13002
17	124	Éducation	16	05.01.2000	AST Intel 200	8	3190	0.015	25137.2
18	124	Éducation	17	05.01.2000	Compaq Pres	11	1650	0.015	17877.75
19	106	Construction	18	05.01.2000	AST Intel 200	11	3190	0.015	34563.65
20	101	Construction	19	05.01.2000	Compaq Pres	14	1650	0.015	22753.5
21	116	Pharmaceutic	20	06.01.2000	IBM 500	7	2299	0.015	15851.605

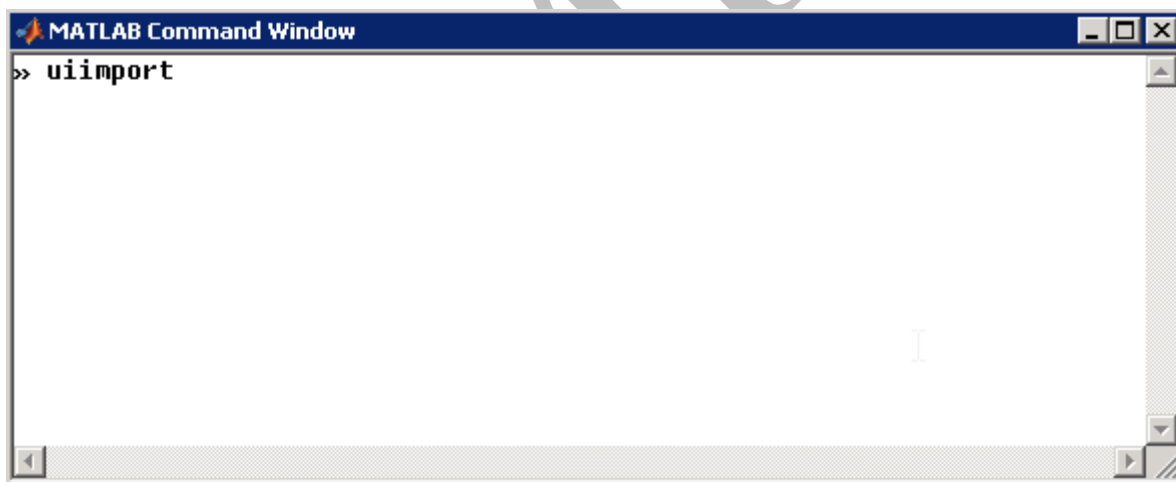
Import data with the wizard

There are some people that don't like to use the terminal to import data, they prefer to import using a WYSIWYG wizard.

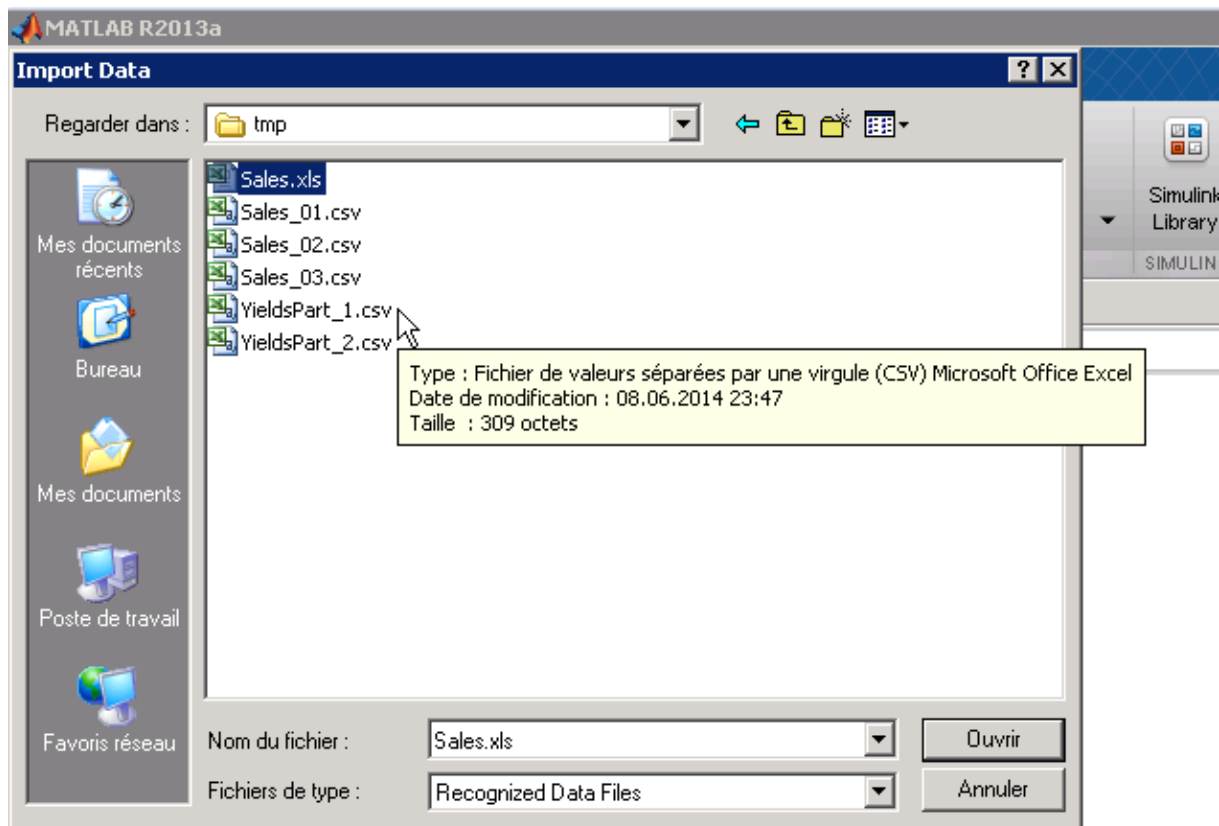
Thus in the **Home** ribbon click on the button **Import Data**:



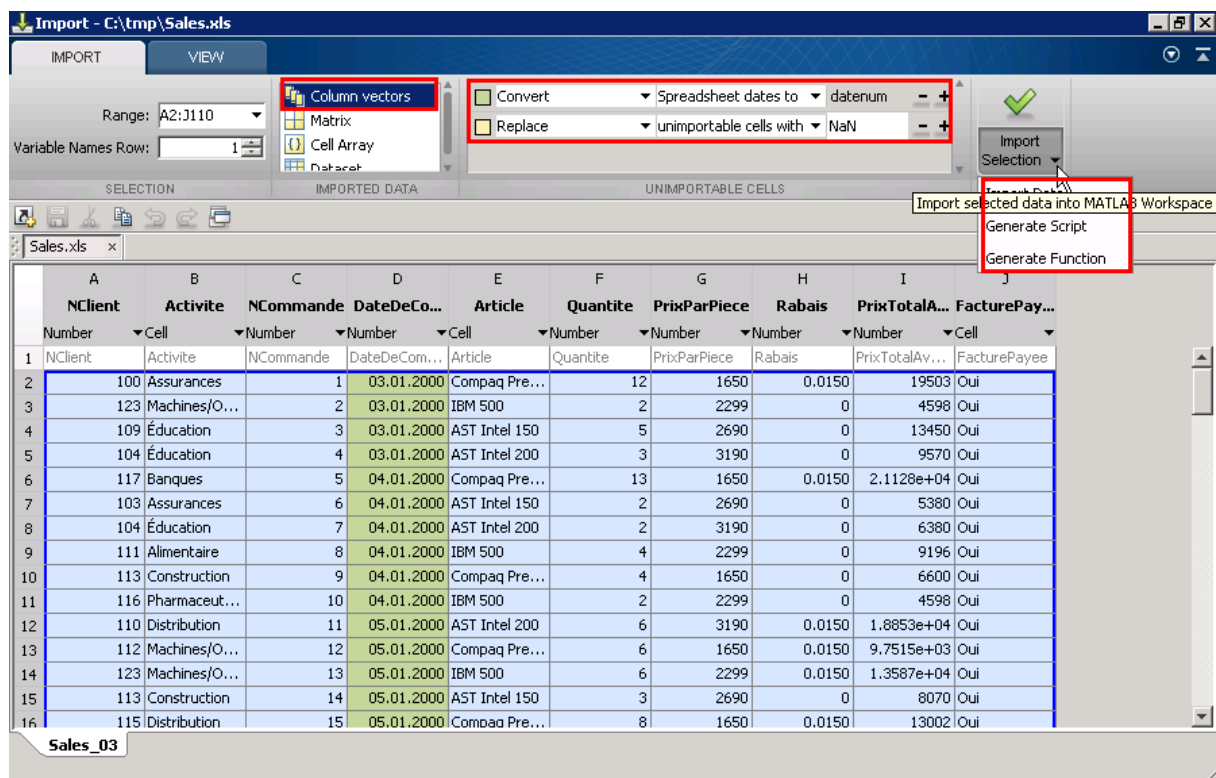
Or if you want to start this wizard from the terminal use the command **uiimport**:



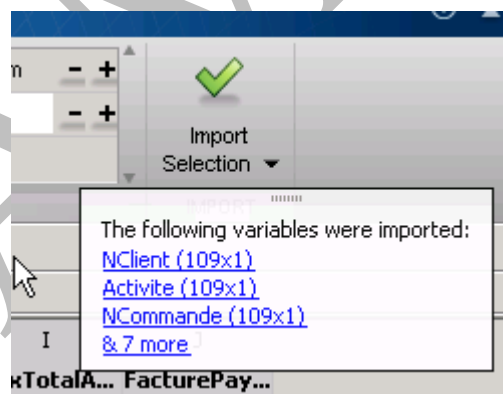
When the wizard start the first thing it ask you is to choose the file with the datas you want to import:



Once the data selected you get the **Import** Window with different option to import the selection as you can see below:

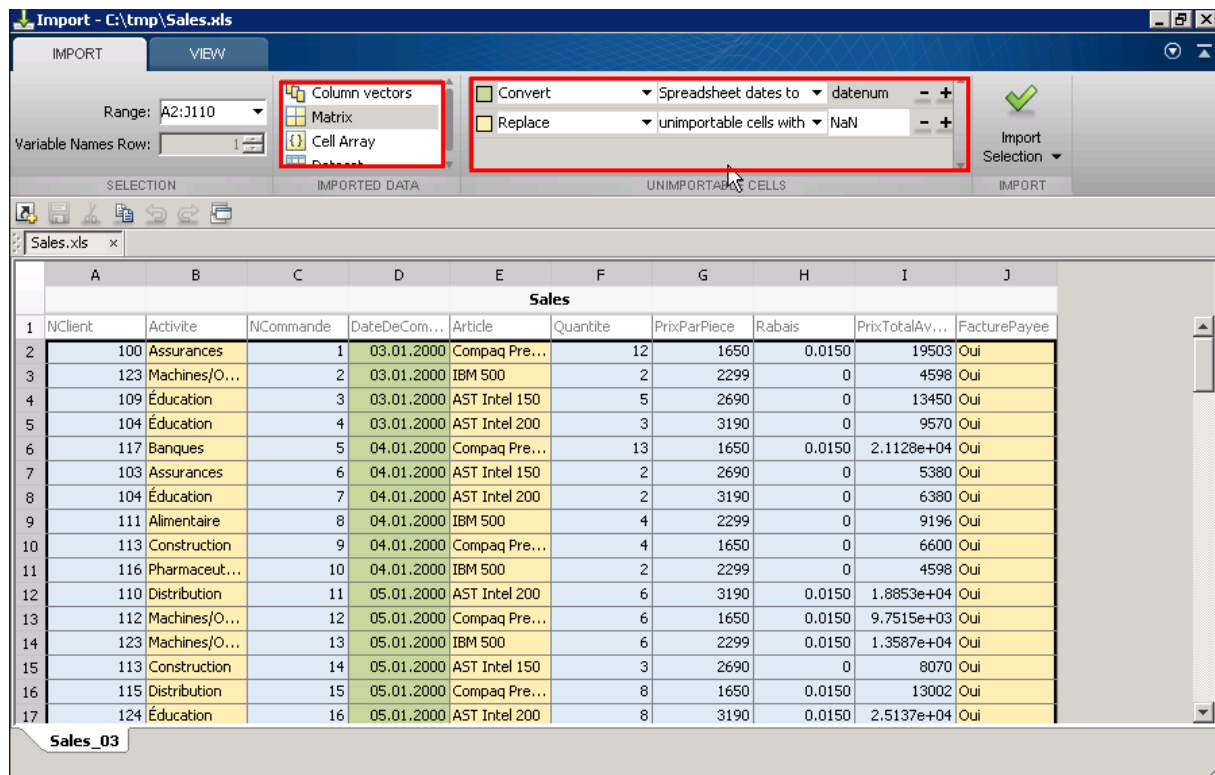


If you select the option **Import Data** from the button **Import Selection** you will get first a small message:

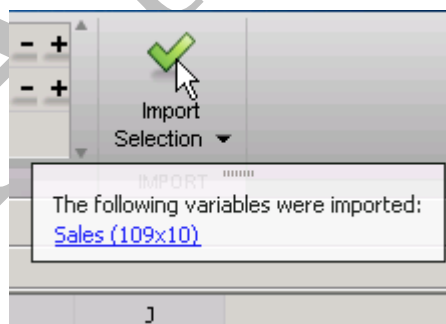


Resulting in imported 10 vectors in 10 separated variables.

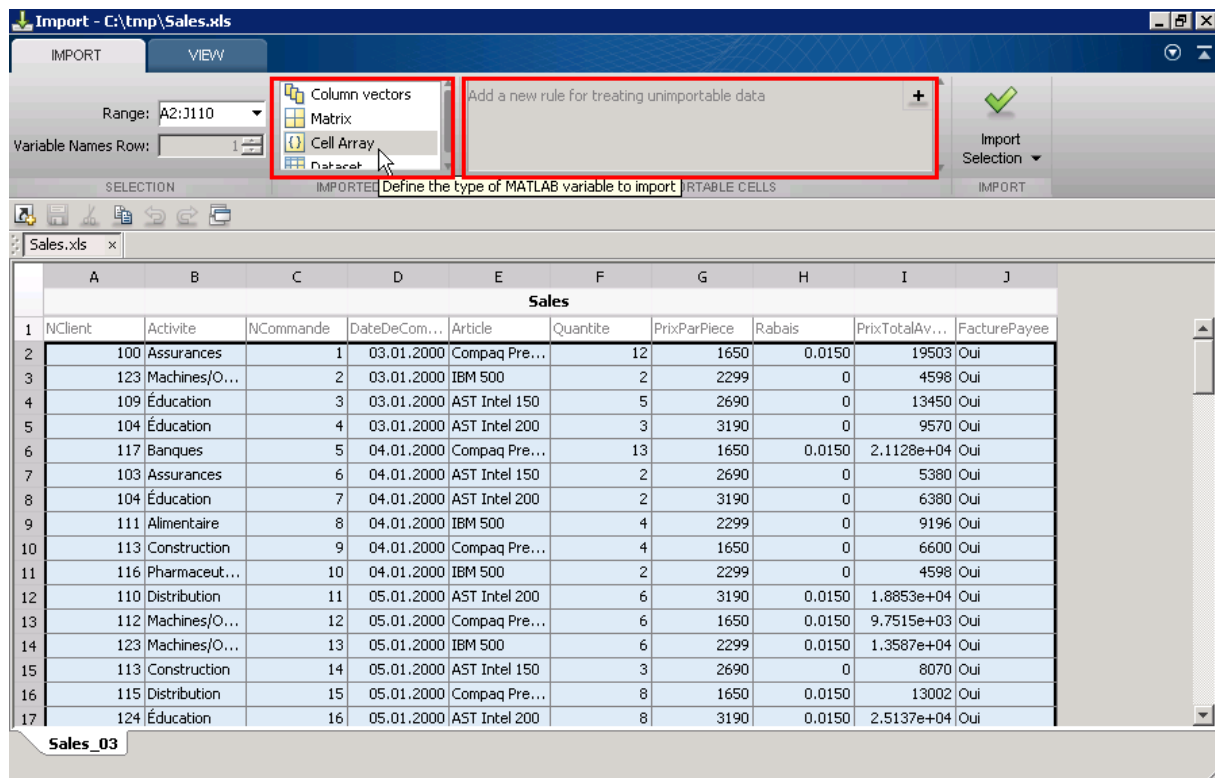
See carefully that depending on the type of variable you want to import your data in (**Column Vectors**, **Matrix**, **Cell Array** or **Dataset**) the colors in the sheet will change to inform you to what will be converted (in green) and what will be replaced (in yellow) as you can see below:



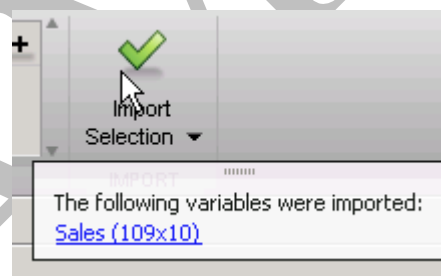
If you choose for example the structure **Matrix** and validate the import you will get a unique matrix with 10 columns but some columns will all be with NaN (the columns with strings that are in yellow above)



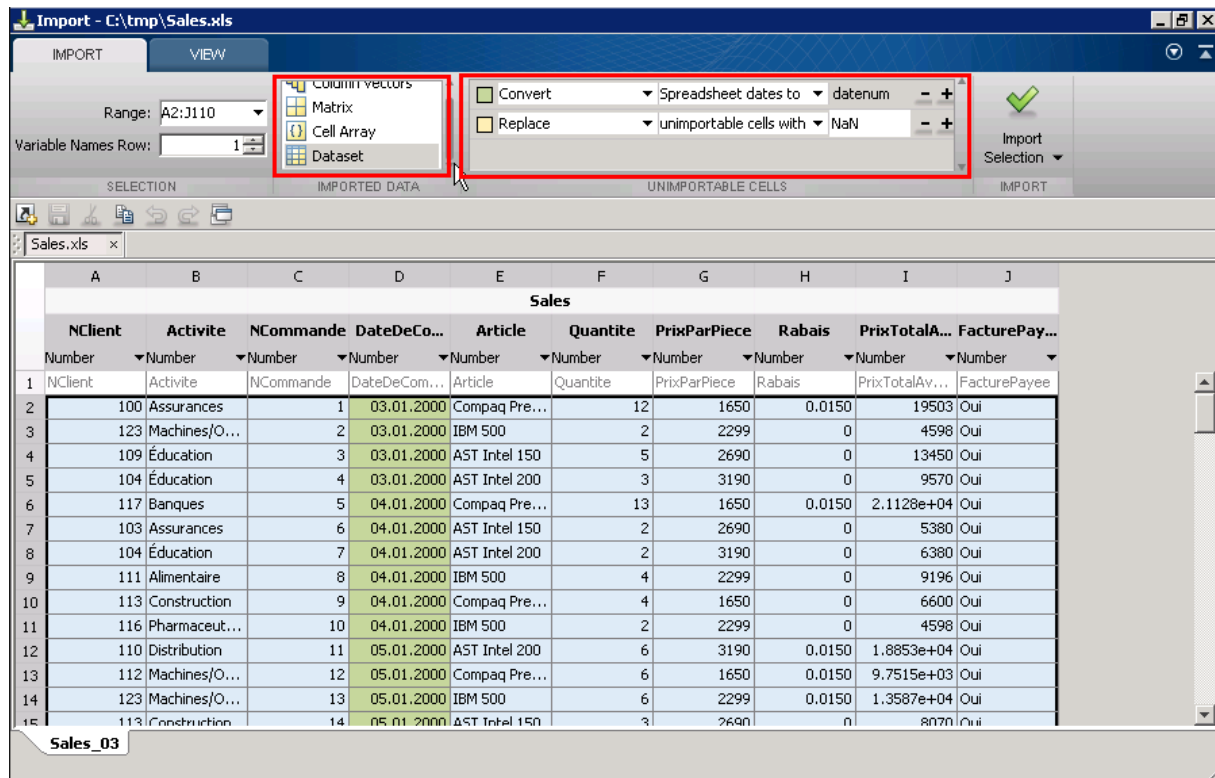
If you choose **Cell Array** as structure for sure... everything can be imported as you can see it below:



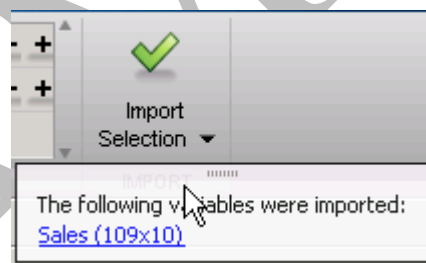
And once you validate you can see that only one variable will be created:



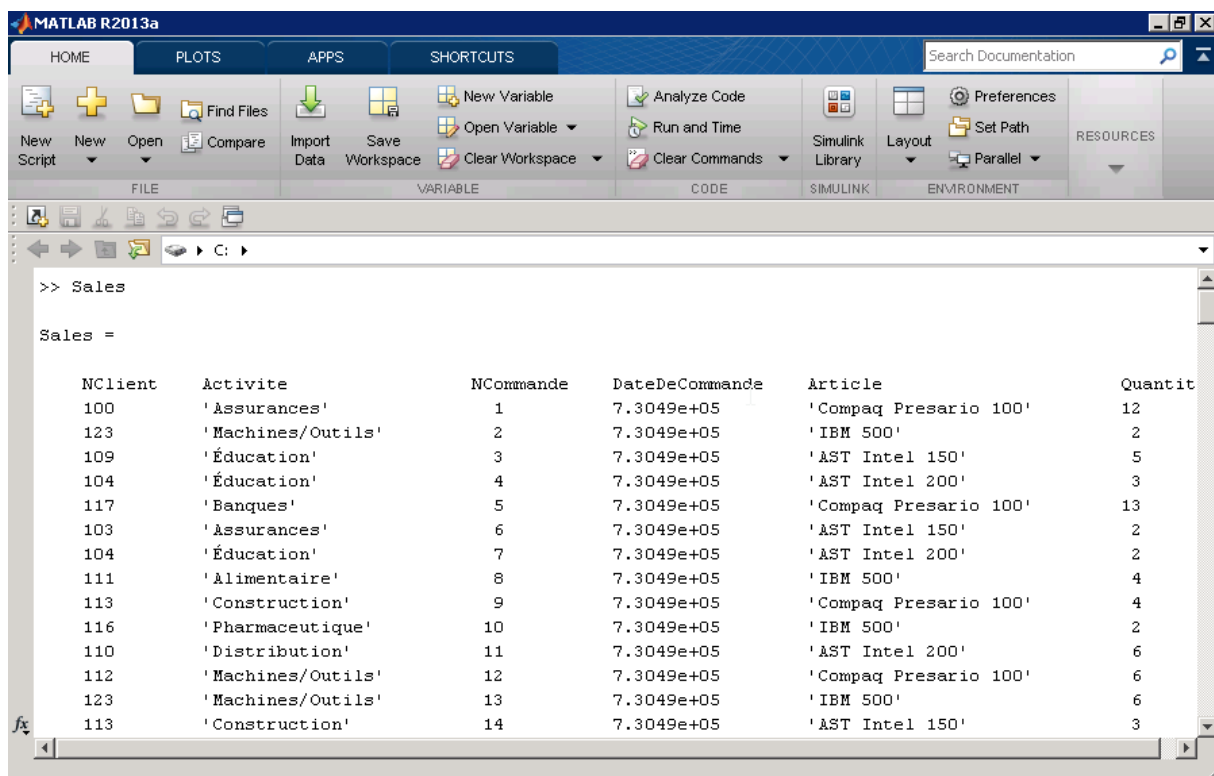
If you select **Dataset** you will not be able to import dates if you don't choose to convert them as string:



A dataset will also create only one unique variable in the workspace:



The result will look like this when you import as a dataset:



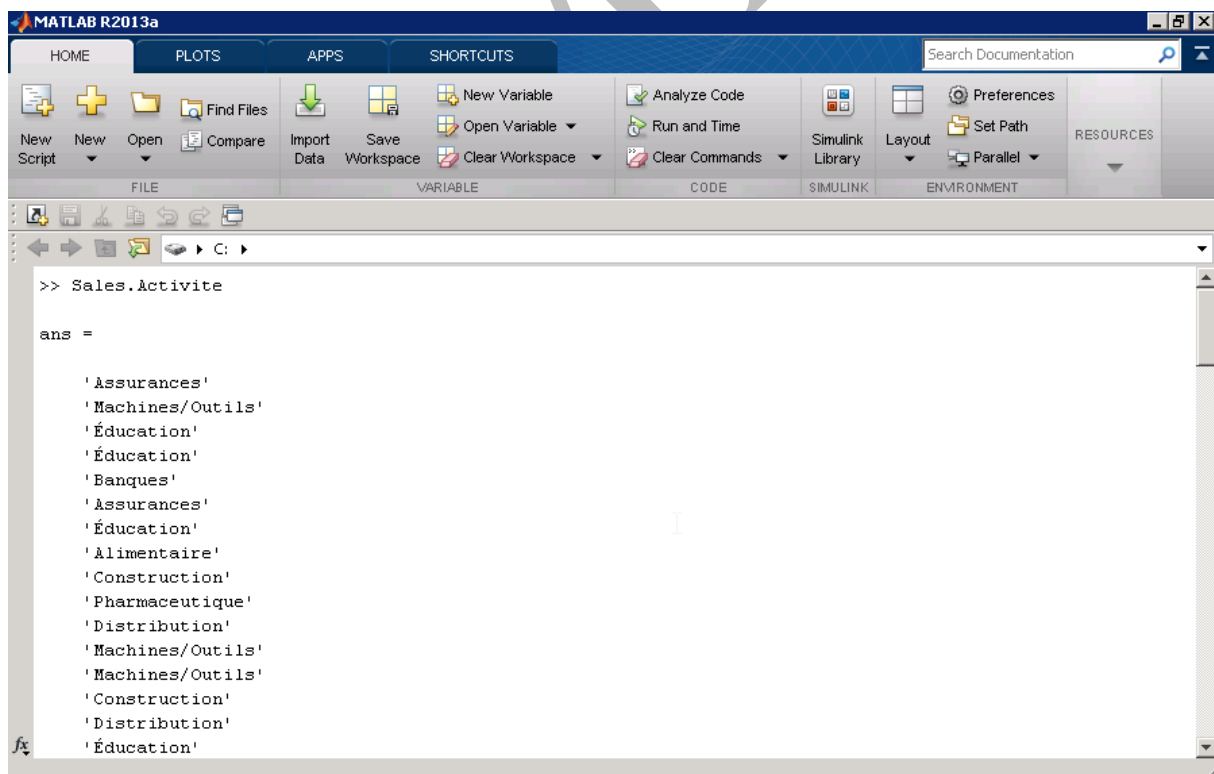
```

>> Sales

Sales =

    NClient    Activite    NCommande    DateDeCommande    Article    Quantite
    _____    _____    _____    _____    _____    _____
    100        'Assurances'         1        7.3049e+05    'Compaq Presario 100'    12
    123        'Machines/Outils'    2        7.3049e+05    'IBM 500'                2
    109        'Education'         3        7.3049e+05    'AST Intel 150'          5
    104        'Education'         4        7.3049e+05    'AST Intel 200'          3
    117        'Banques'           5        7.3049e+05    'Compaq Presario 100'    13
    103        'Assurances'         6        7.3049e+05    'AST Intel 150'          2
    104        'Education'         7        7.3049e+05    'AST Intel 200'          2
    111        'Alimentaire'       8        7.3049e+05    'IBM 500'                4
    113        'Construction'     9        7.3049e+05    'Compaq Presario 100'    4
    116        'Pharmaceutique'   10       7.3049e+05    'IBM 500'                2
    110        'Distribution'     11       7.3049e+05    'AST Intel 200'          6
    112        'Machines/Outils'  12       7.3049e+05    'Compaq Presario 100'    6
    123        'Machines/Outils'  13       7.3049e+05    'IBM 500'                6
    113        'Construction'   14       7.3049e+05    'AST Intel 150'          3
  
```

To access to a column the syntax is very similar to R:



```

>> Sales.Activite

ans =

    'Assurances'
    'Machines/Outils'
    'Education'
    'Education'
    'Banques'
    'Assurances'
    'Education'
    'Alimentaire'
    'Construction'
    'Pharmaceutique'
    'Distribution'
    'Machines/Outils'
    'Machines/Outils'
    'Construction'
    'Distribution'
    'Education'
  
```

Charts

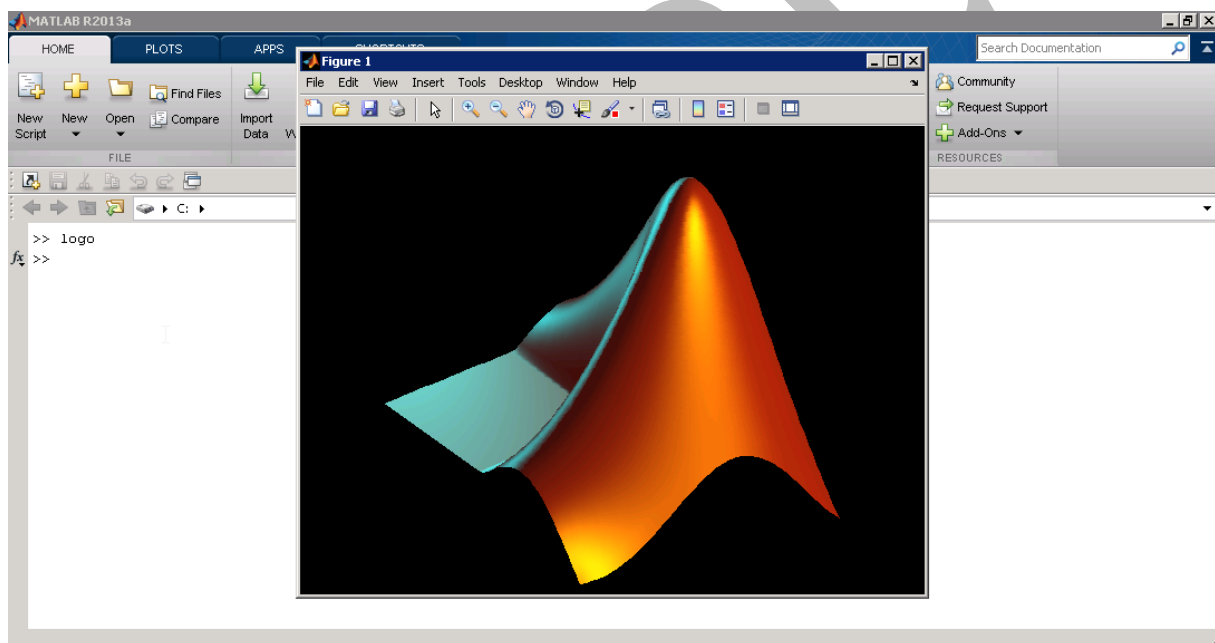
A lot of people love charts and put charts everywhere in their job. You must first remember that for sure "one image says one thousand words" but in mathematics you can apply this quote only for 1D, 2D, 3D and 4D charts but after... you have to go back to the reading of pure mathematical values.

We will see in this chapter a lot of classical, typical charts in physics, signal processing and finance that don't require any toolboxes. The main purpose of this chapter in fact is to show a lot of "hidden" tips in MATLAB and answer to most common questions of the users following my trainings.

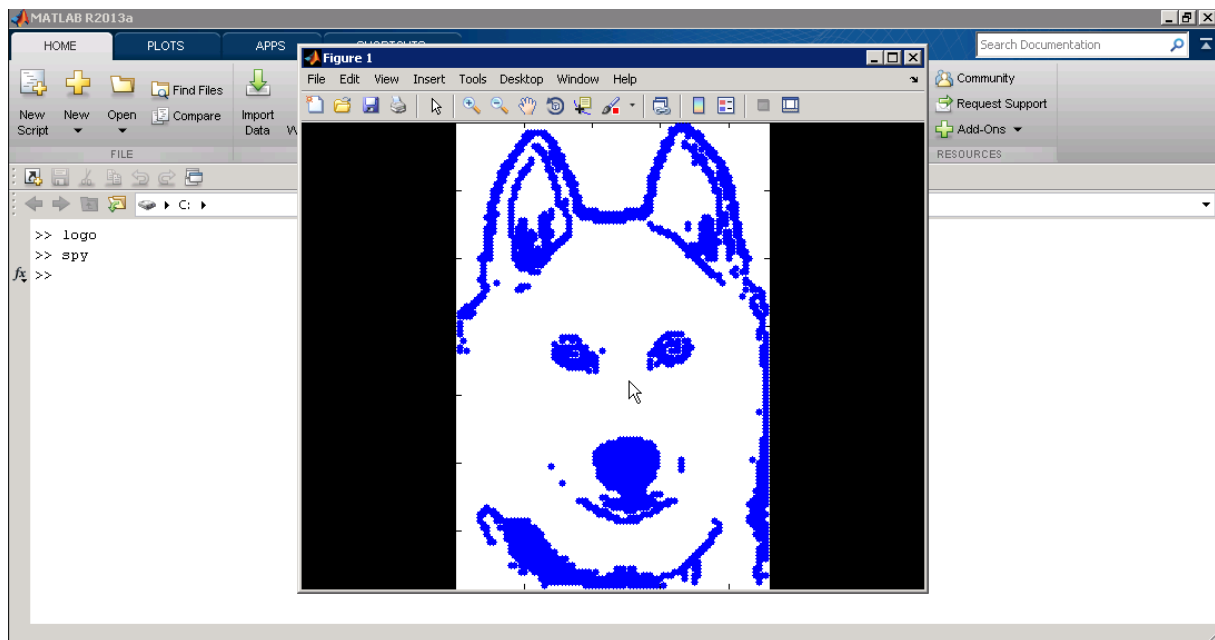
But first some fun...

Demo

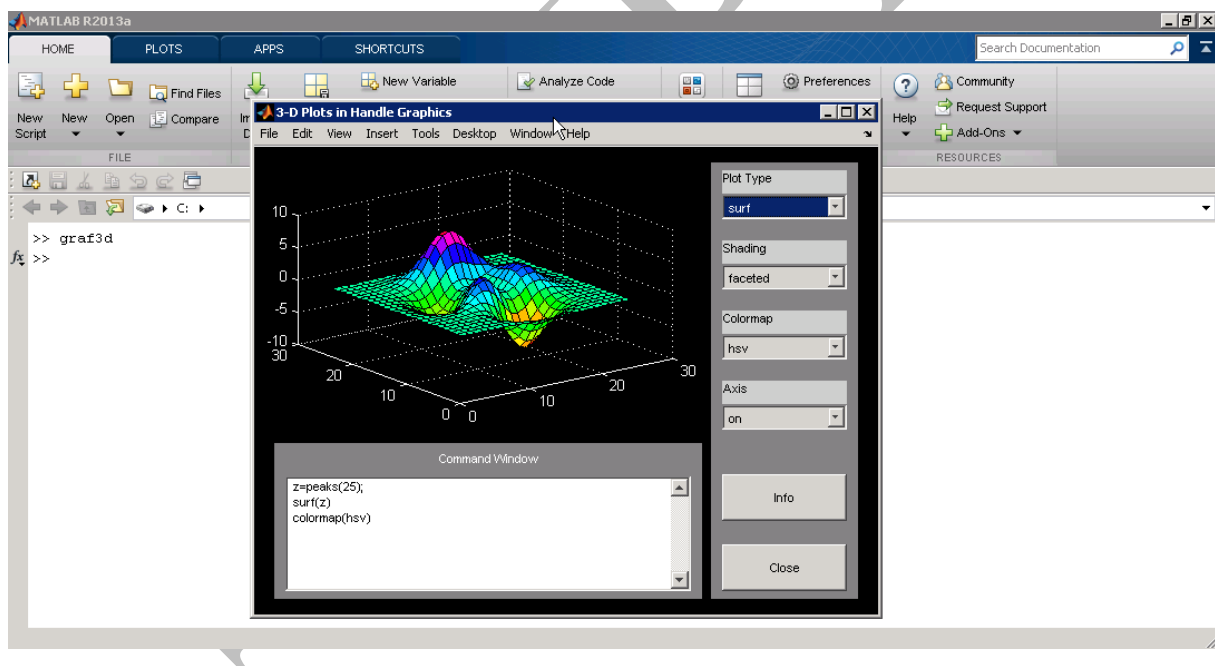
In MATLAB there are some demo charts that can be useful to learn the basics alone. For example if you type the command **logo** you will get...:



And if you type the command **spy**:

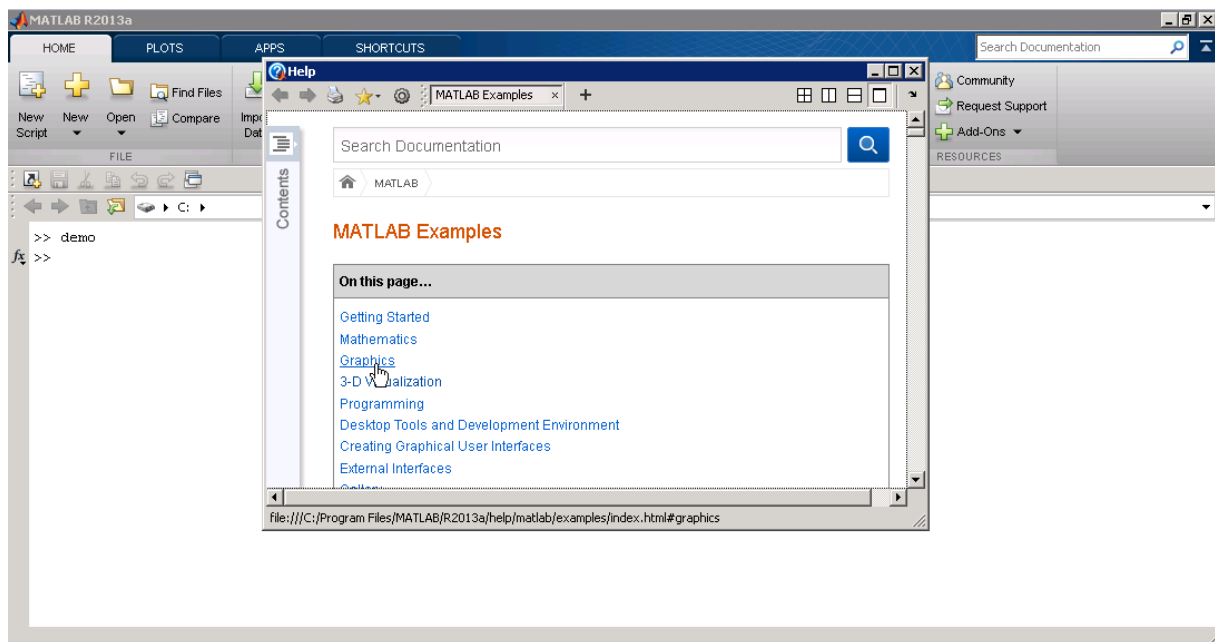


Or **graf3d**:

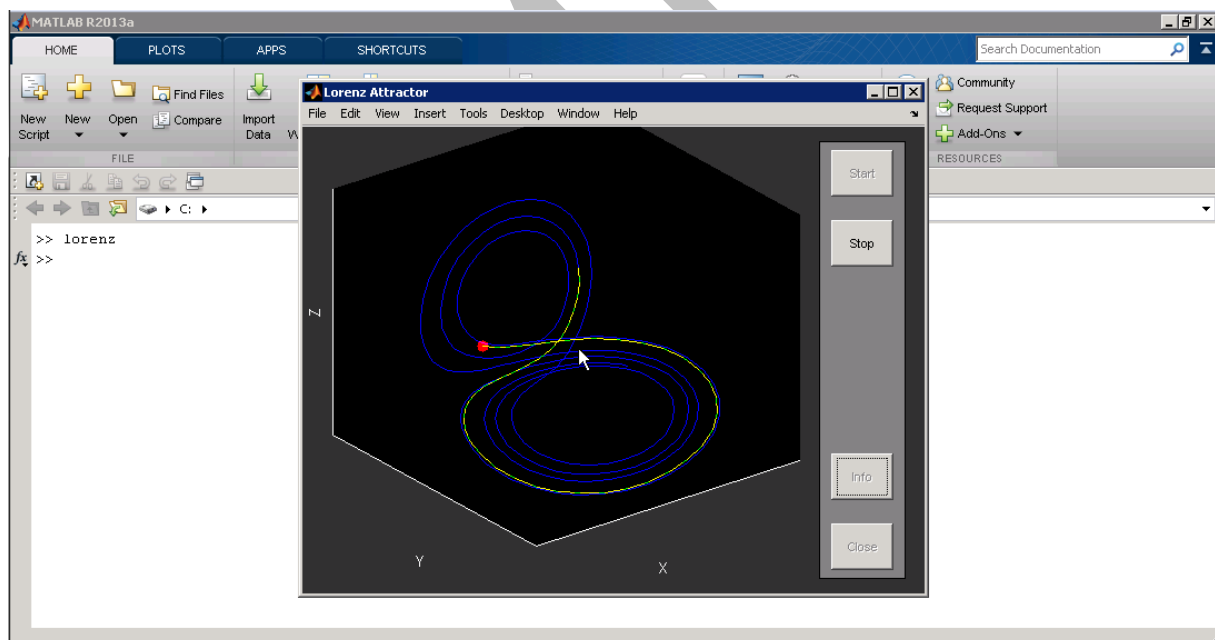


Each of these charts can be edit and you have access to the source code to see how they were builded.

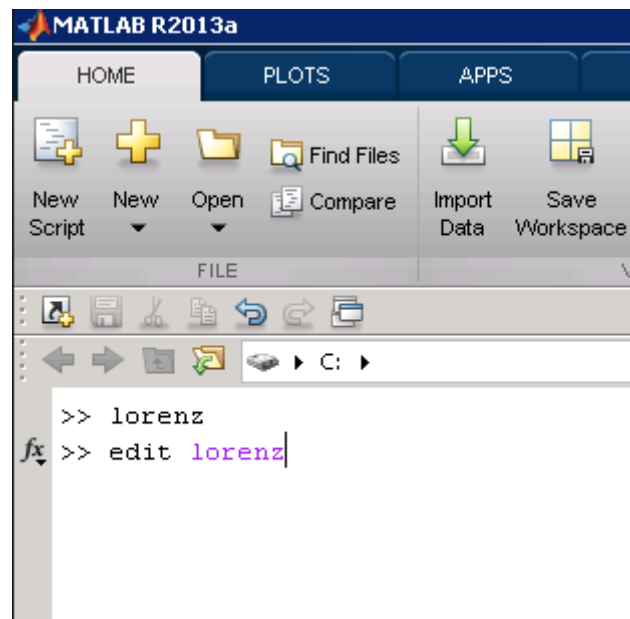
If you type the command **demo** you will see a section *Graphics*:



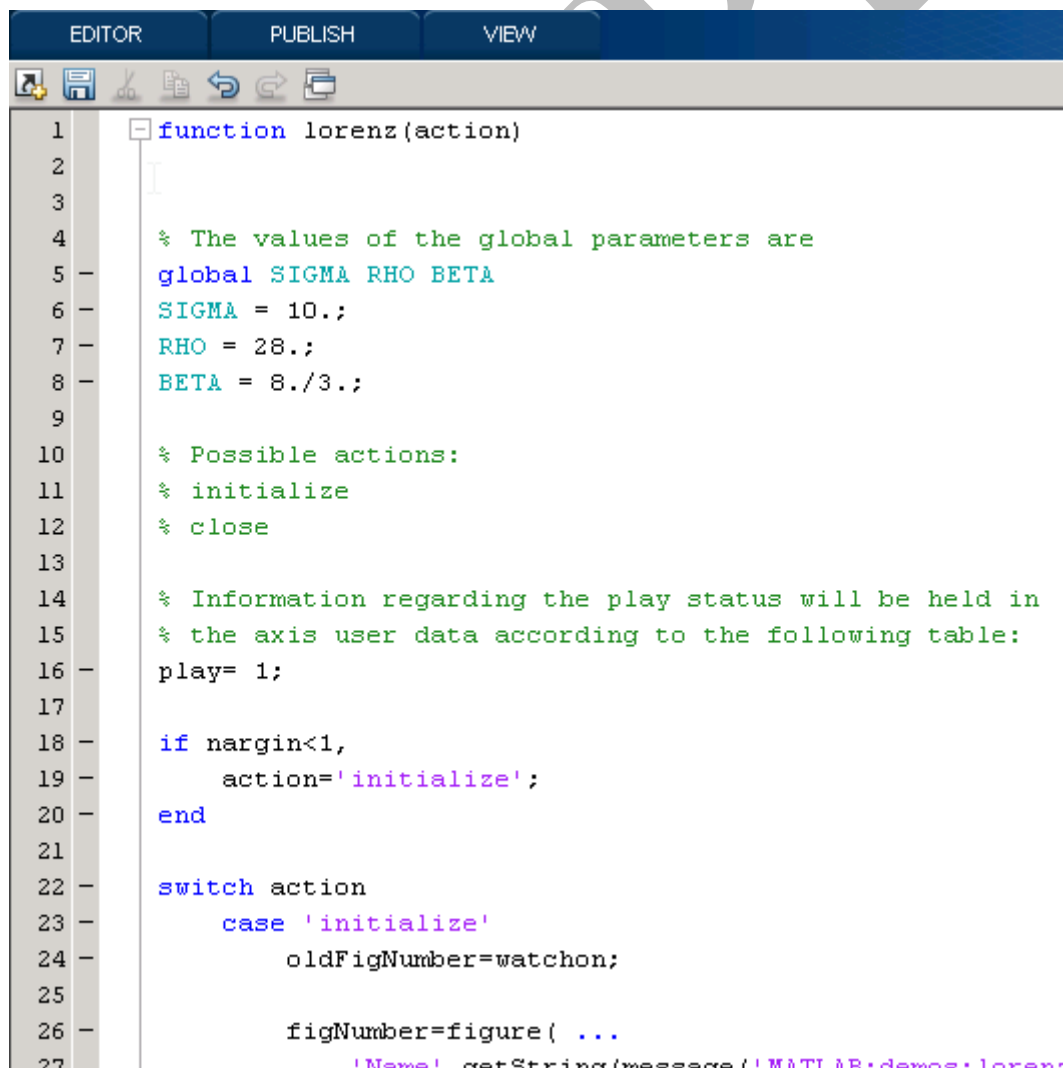
with a lot of possible demo charts commands. For example the famous Lorenz attractor using the command **lorenz**:



And as says before you can see the code source by using the well know command **edit**:

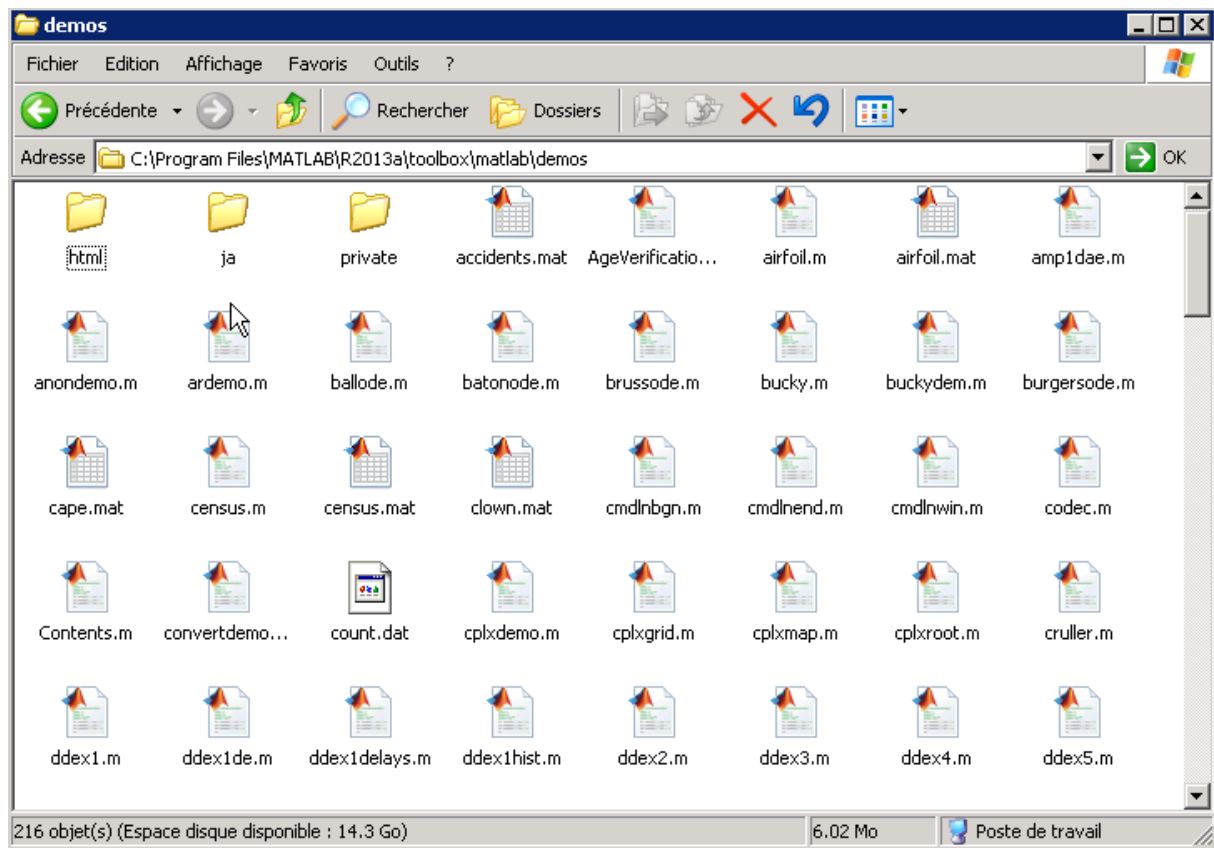


Thus in this particular example you will get:

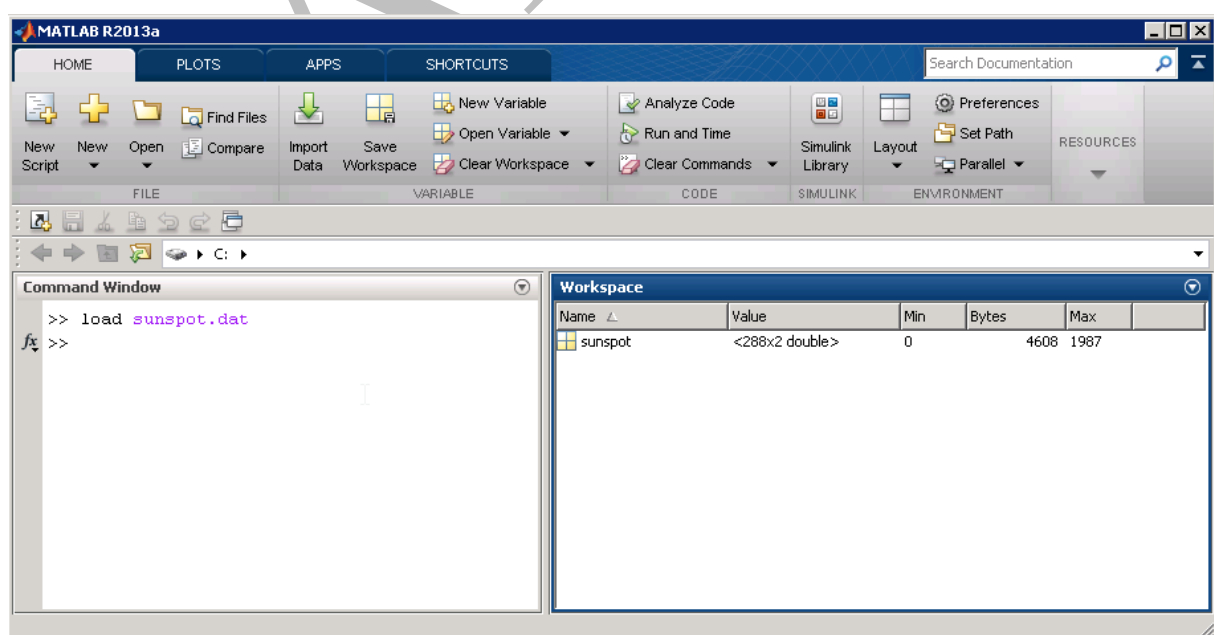


Data sets

Otherwise if you want to play to create you own charts but you do not have any datasets to play with, MATLAB provides a few datasets in the folder *demos* of your hard drive:



For example the small *sunspot* data set can be loaded with the simple well know command **load**:



Or for a more complicated example you can use the *wind* data set (we will use it a lot later in this book):

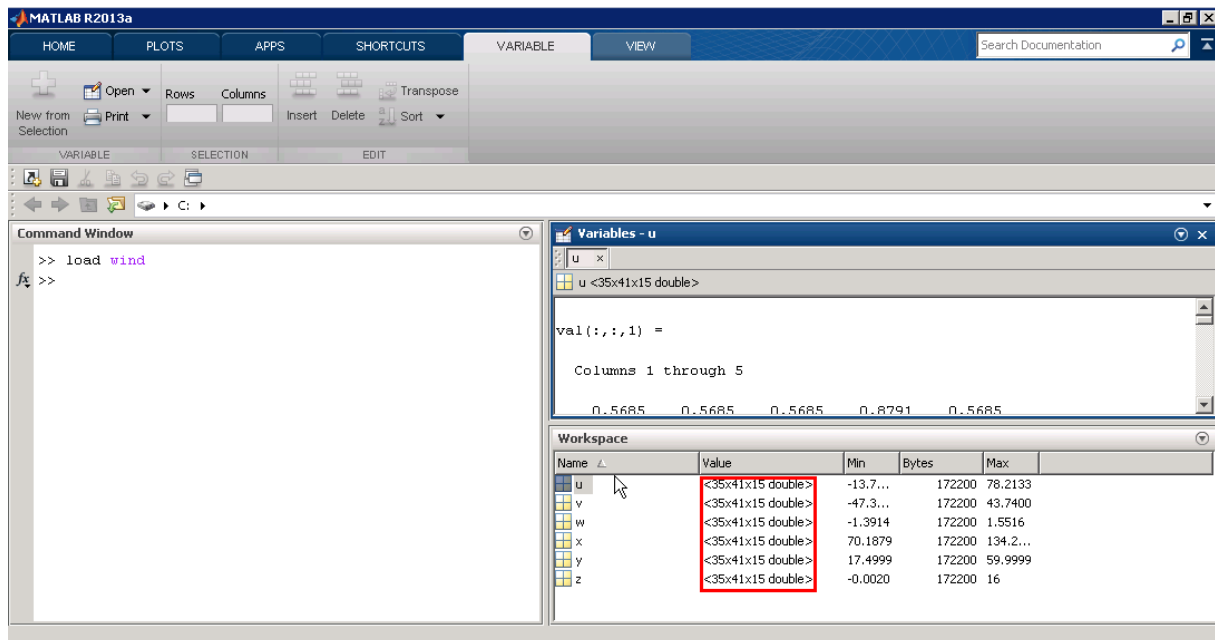
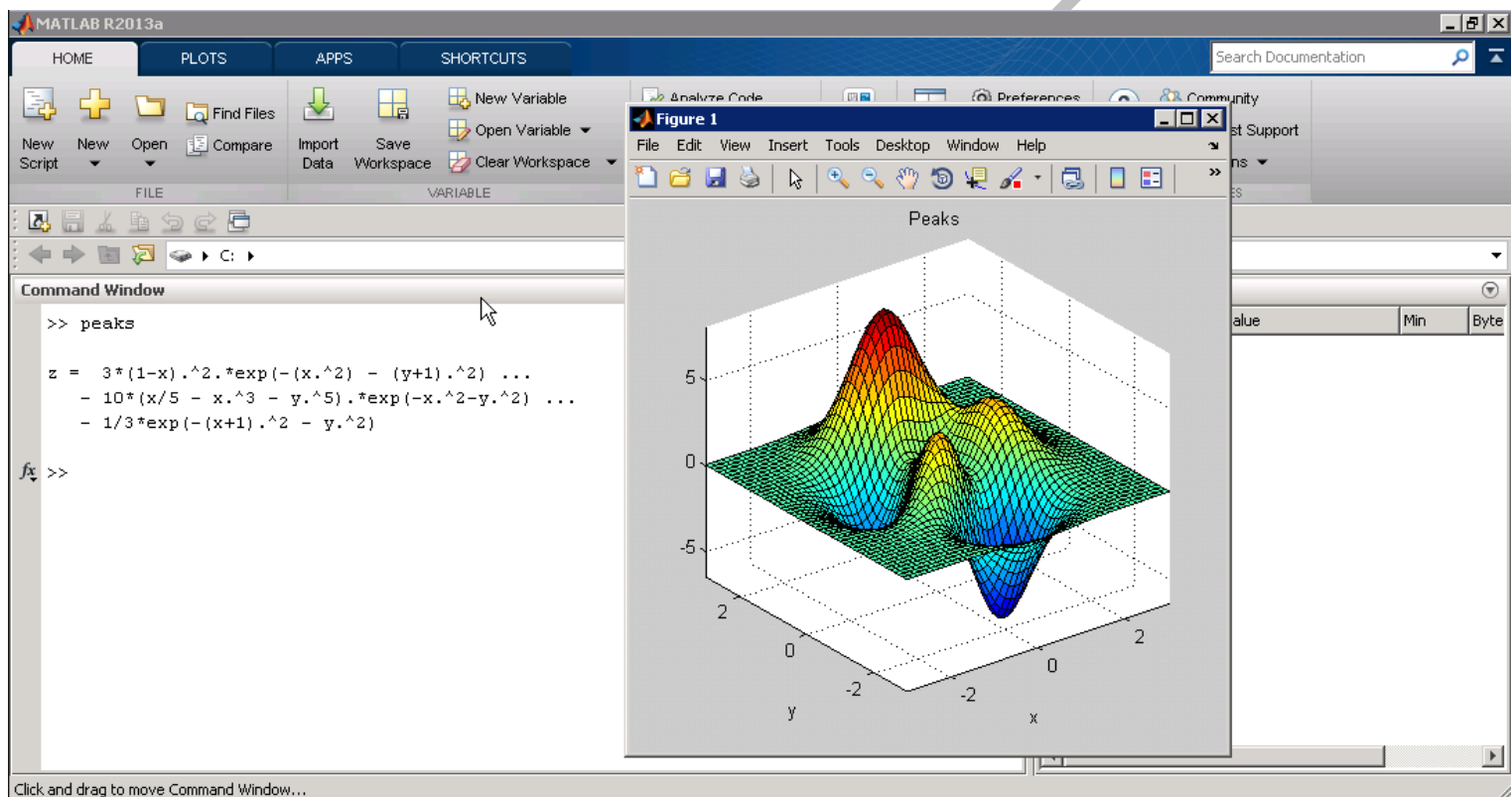


Chart wizard

For users that don't like to write commands or script to create charts there is a chart wizard in MATLAB but for your information... you will be quickly limited if you learn only this way of doing charts.

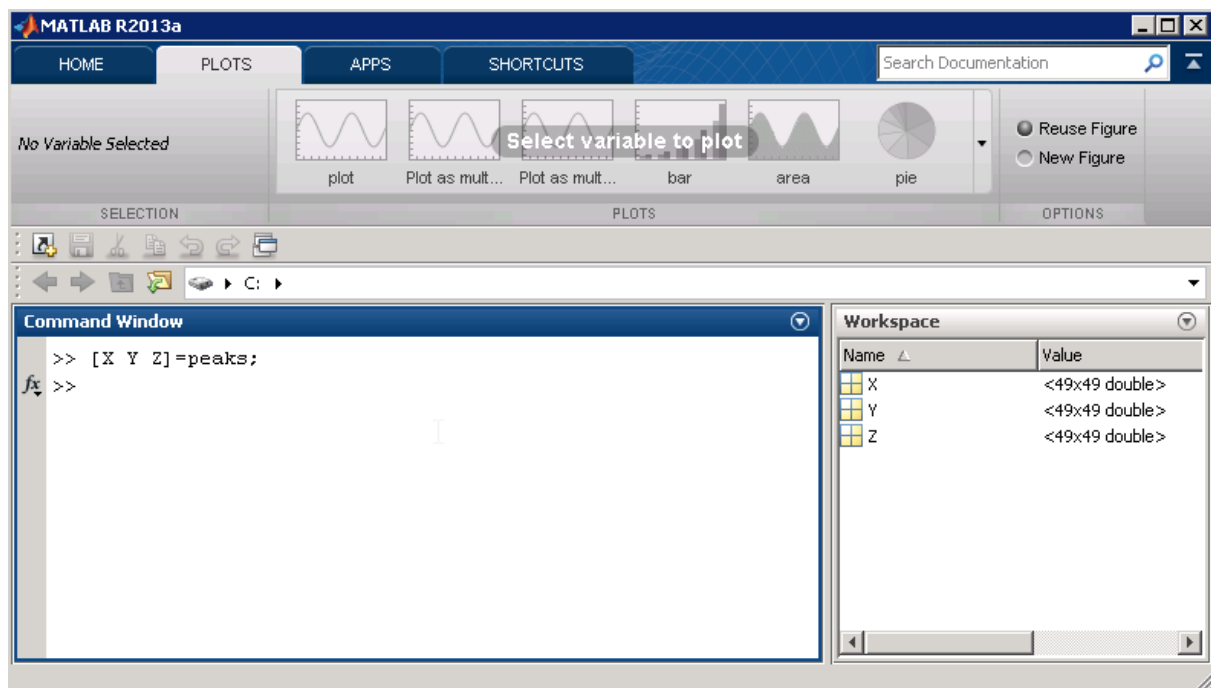
But this wizard is still sometimes useful to remember a simple code or command that we have forgotten and when we don't have an Internet access in our company to search for some help or documentation on Google. This is way it is still useful to have a quick look at it.

For this purpose we will play with the old famous command **peaks**:

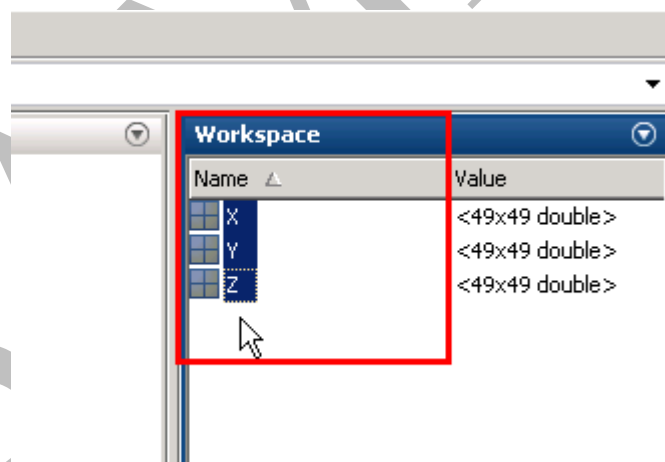


OK let us start!

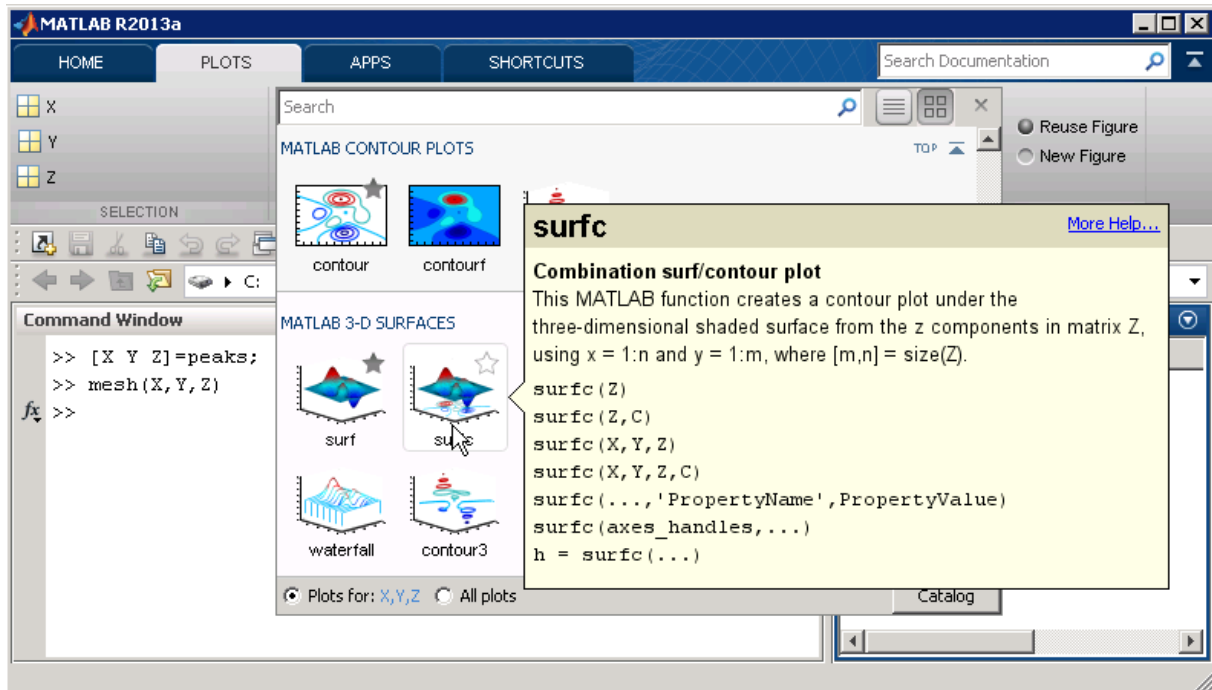
First we import separately the matrices of the peak function as following:



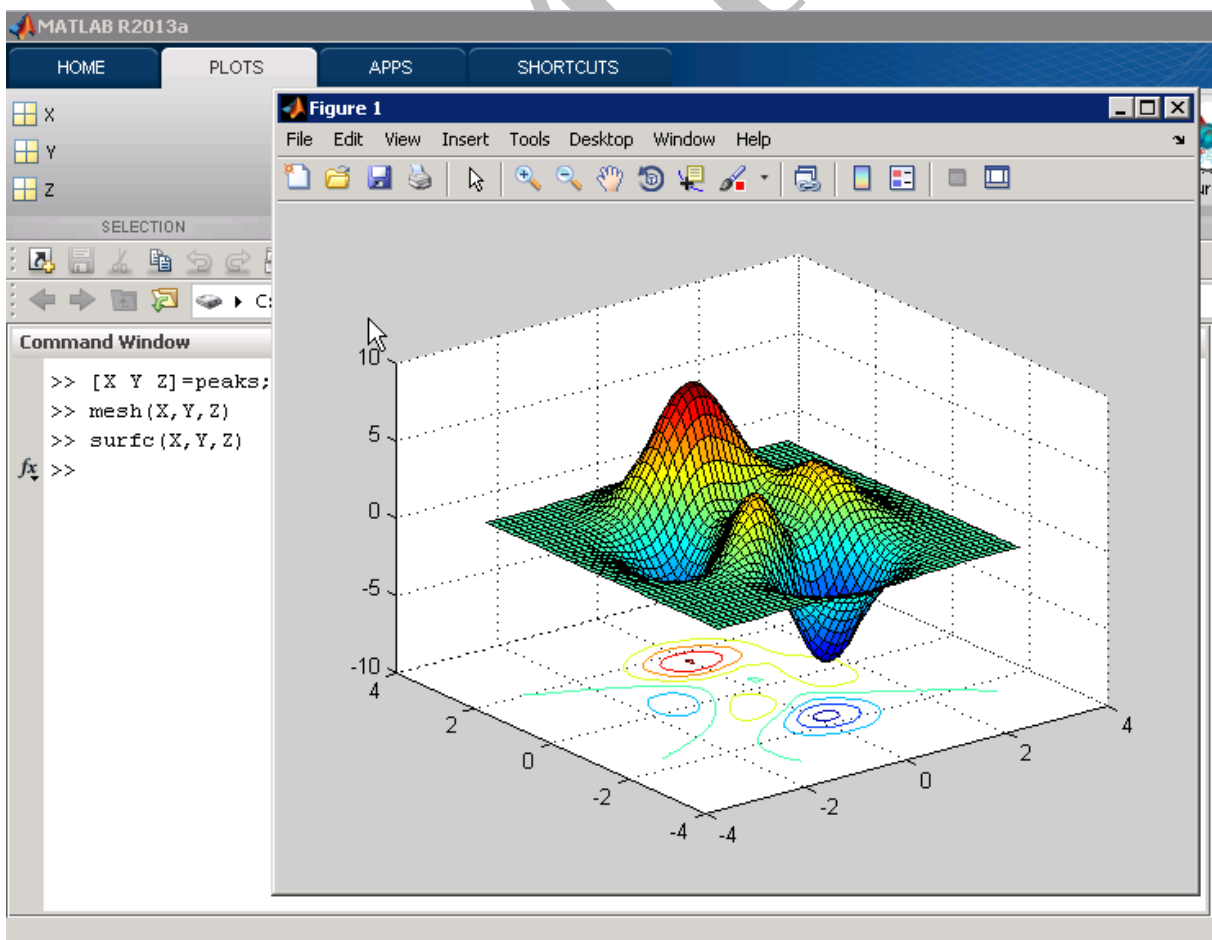
As you can see we have three matrices (this is commonly named a "mesh grid" in the MATLAB world):



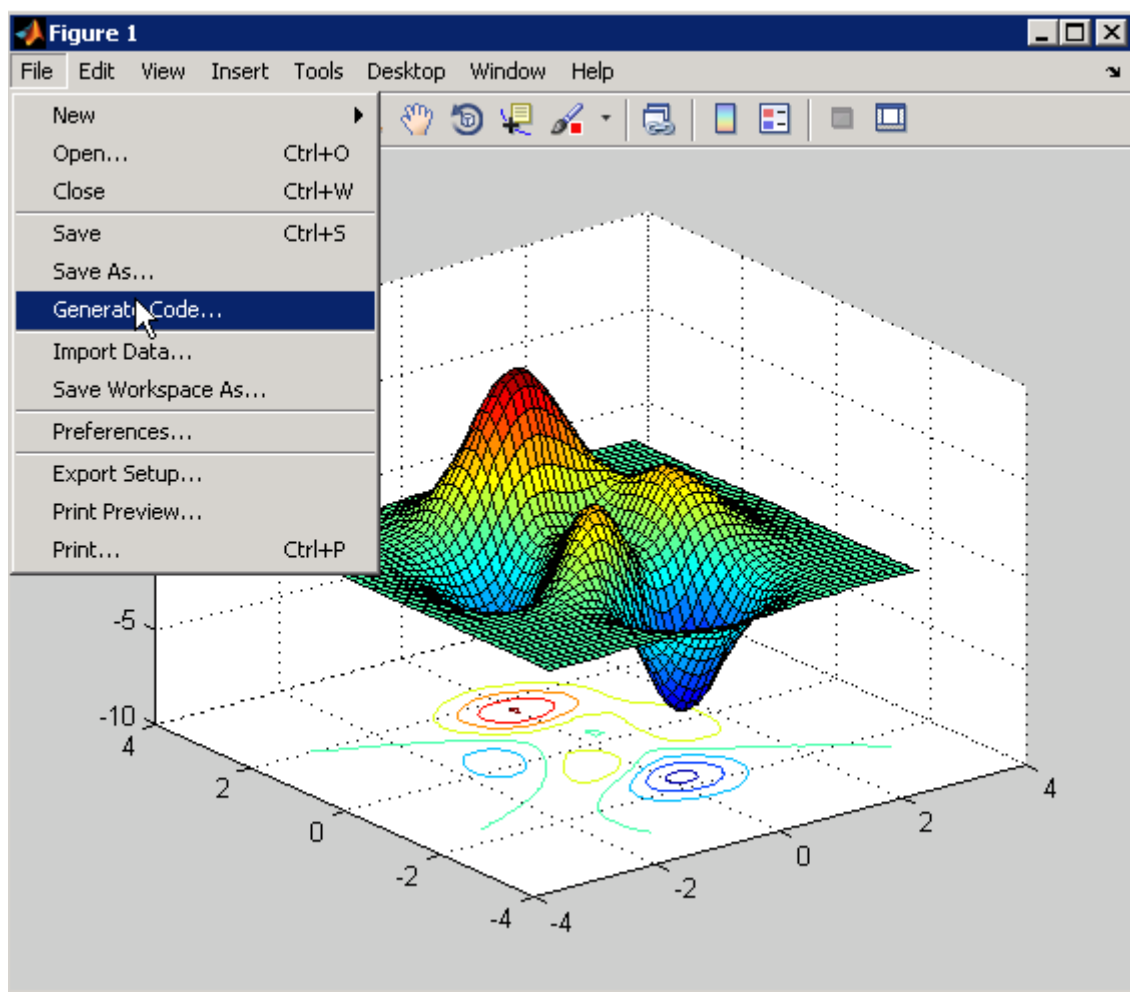
Once this done you go in the ribbon **PLOTS** and select for example the **surf** MATLAB 3-D surfaces:



You will get the following result:

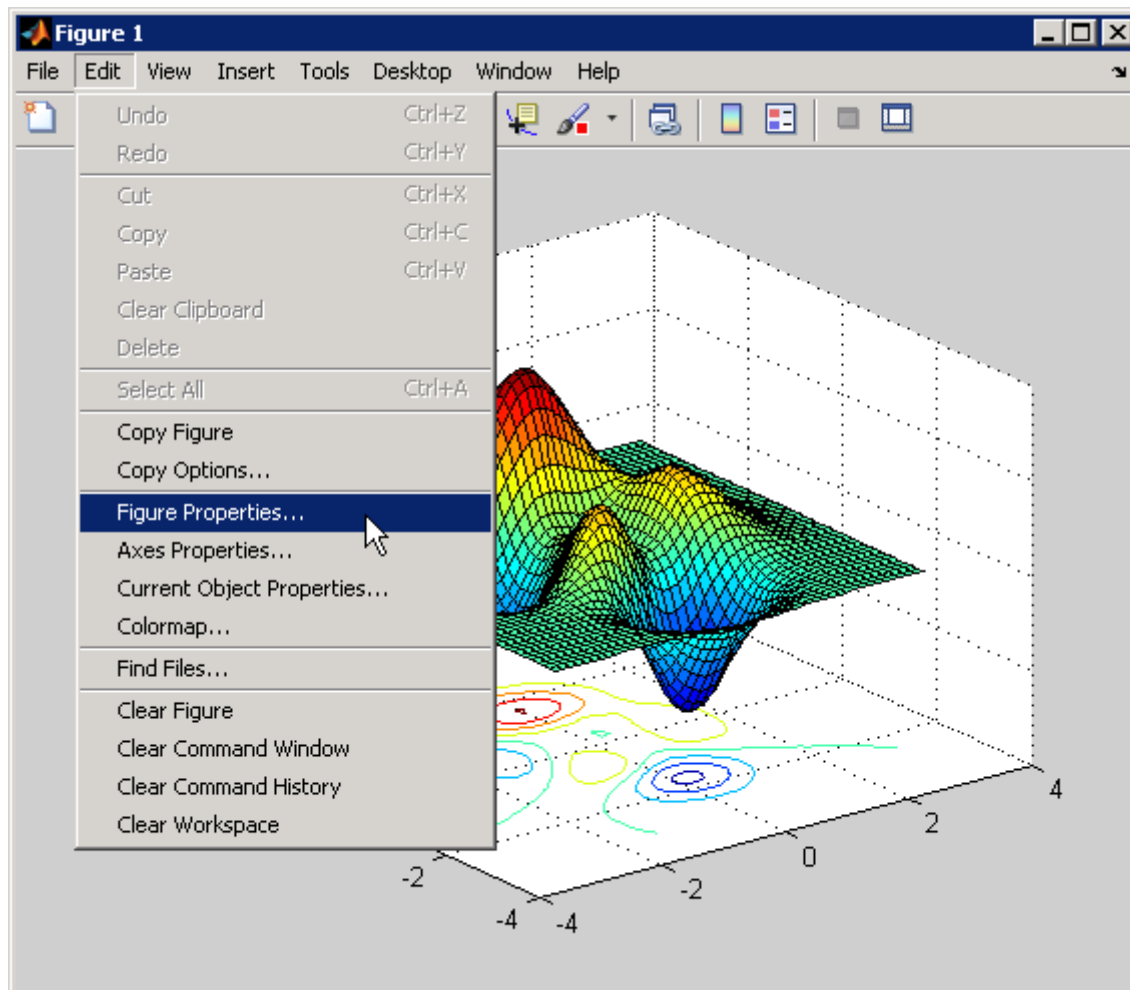


Now if you go in the menu **File/Generate Code...**:

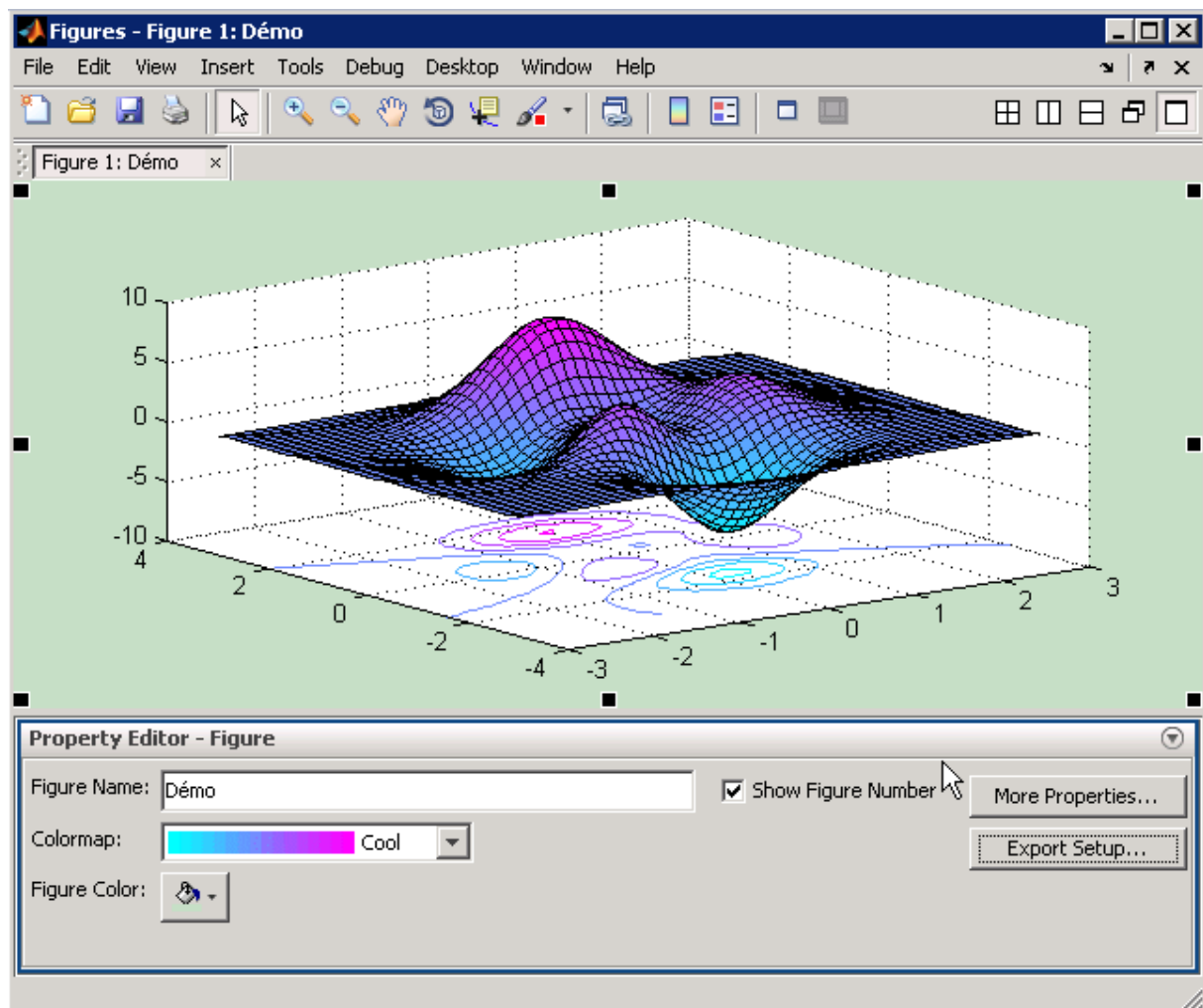


MATLAB will give you automatically all the (awful) equivalent script that can reproduce this chart later automatically (I did not make a screenshot of this code because is very awful).

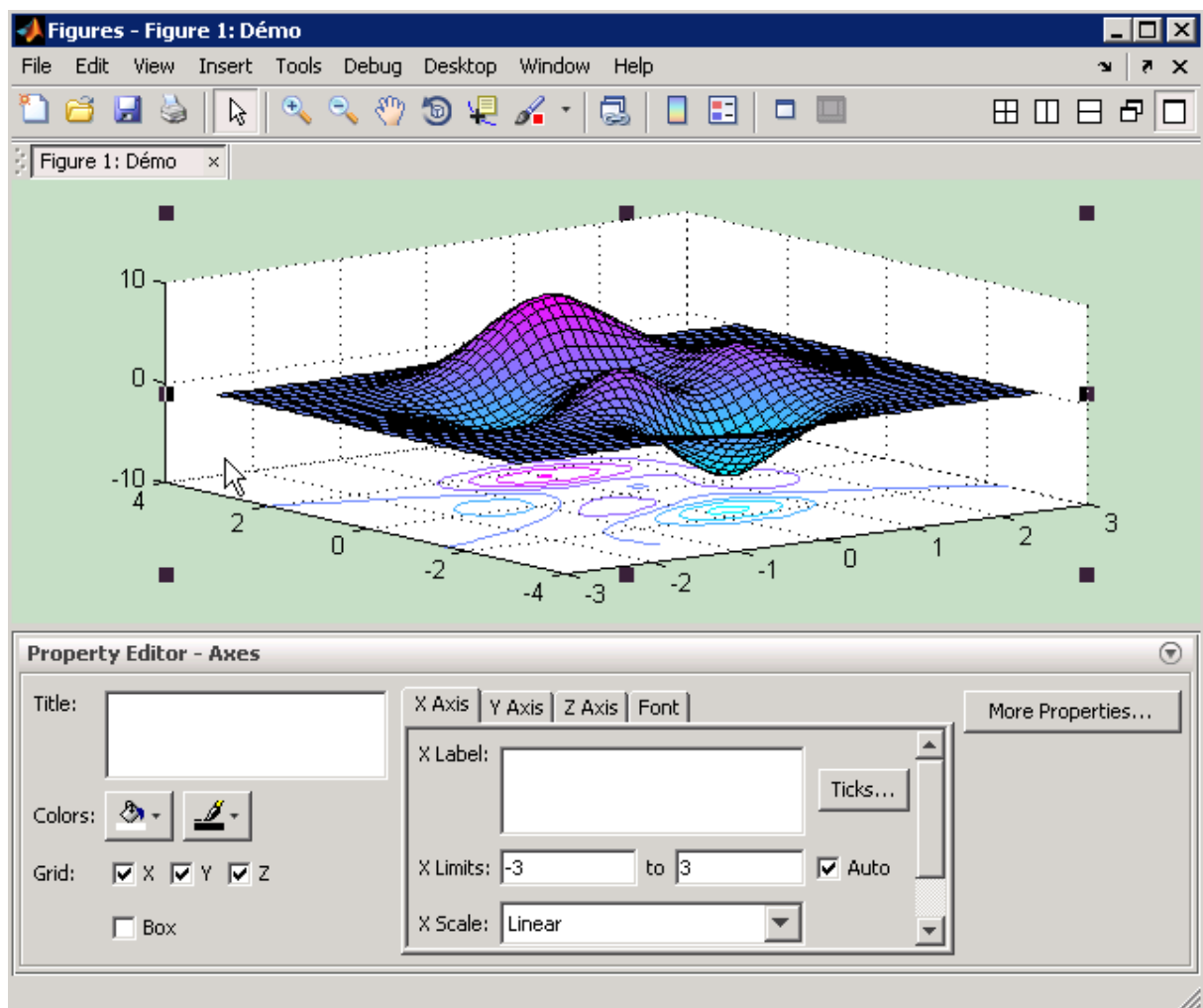
Now to customize any chart, when you have the menu visible (because it can be hidden), one possibility is to go on the menu **Edit/Figure Properties...**:



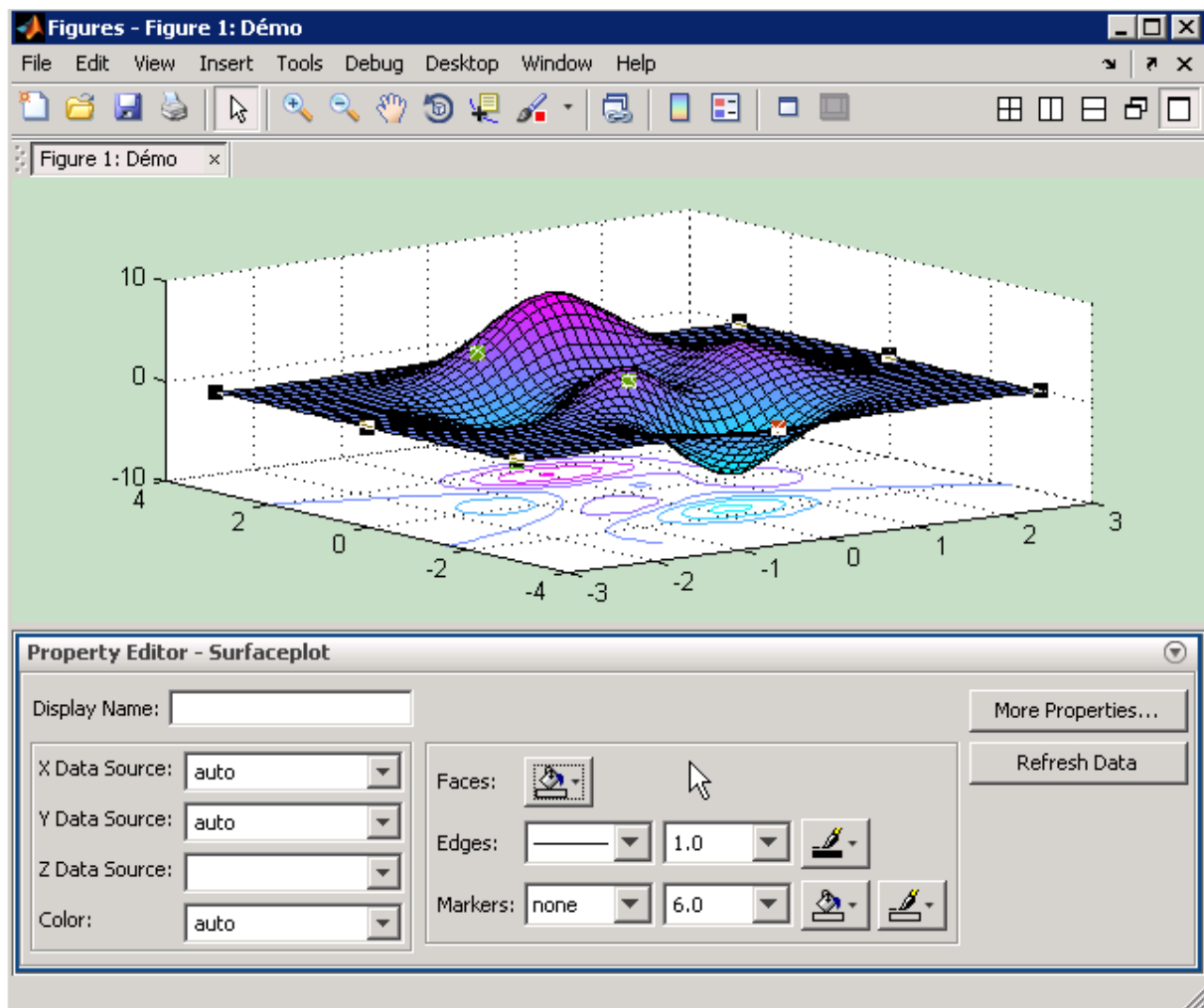
Then the chart window will change a little bit and you will have hundreds of options to customize the chart as for example (see below) the background color of the chart or its colormap and also if you want the figure number to be visible or not in the title bar of the windows, etc.:



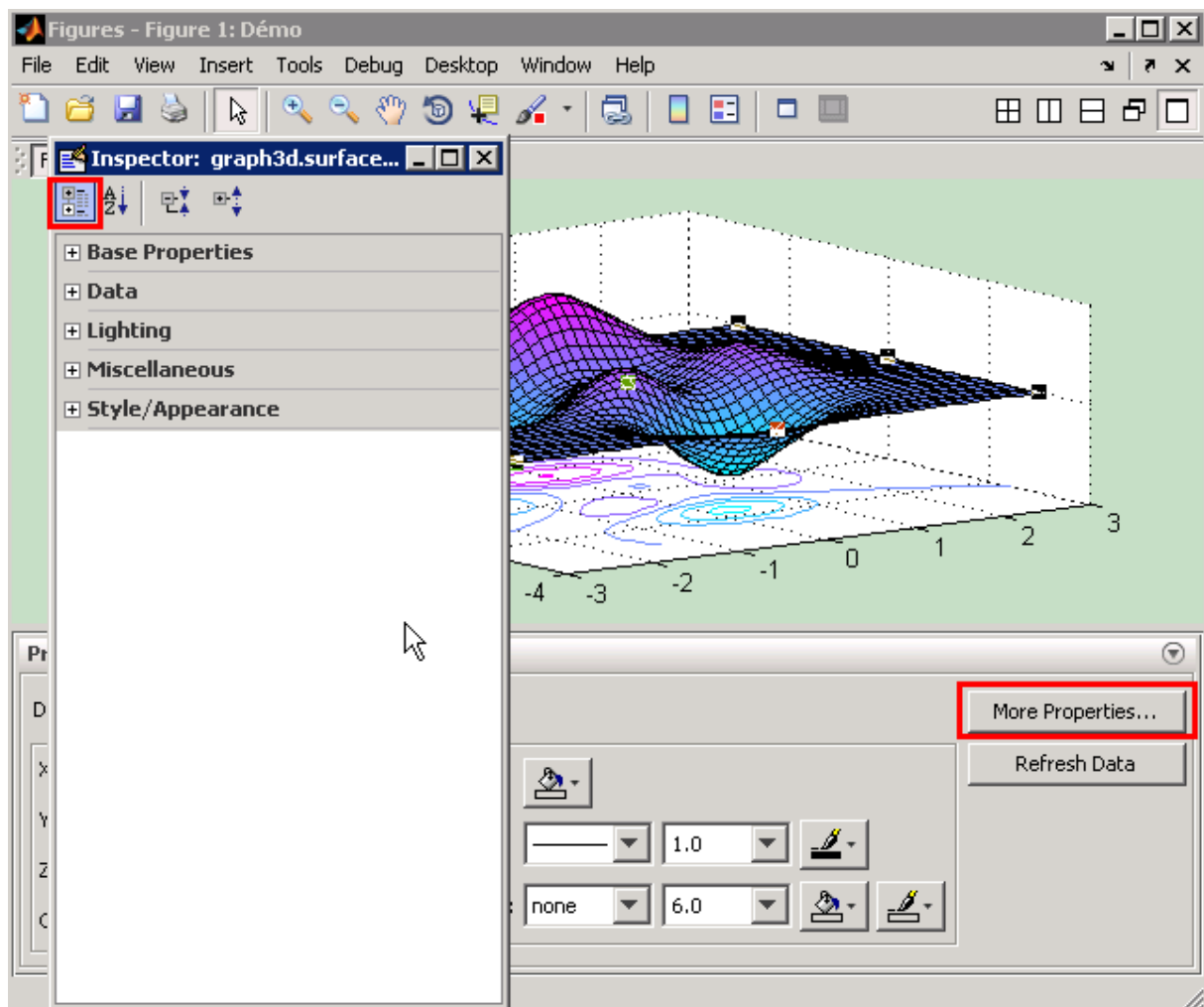
If you double click on the axis, the bottom part will change with other customization options that are sometimes very useful and important (for the people that don't like to script everything themselves!):



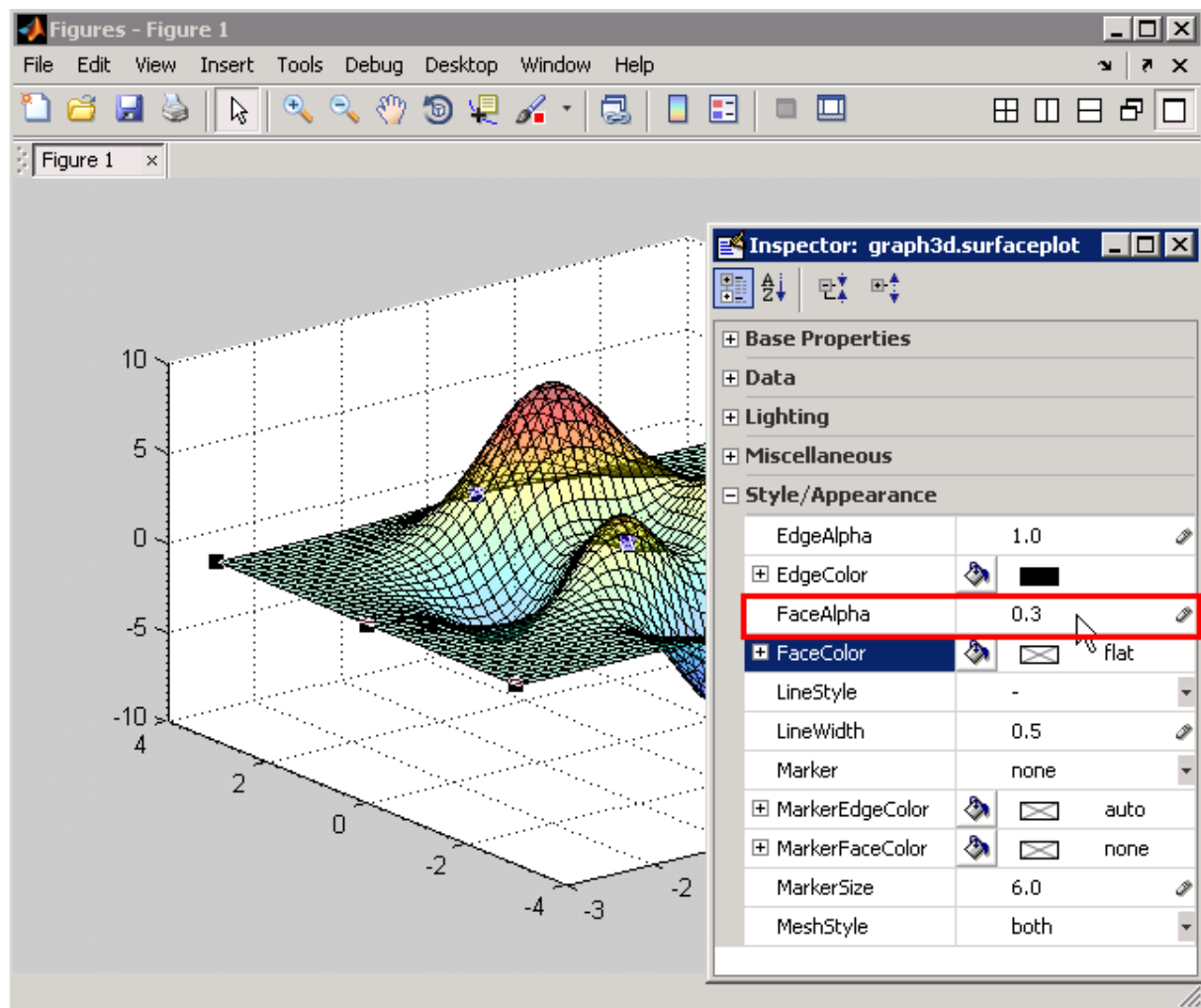
Or if you double click on the surface you can also change a few options:



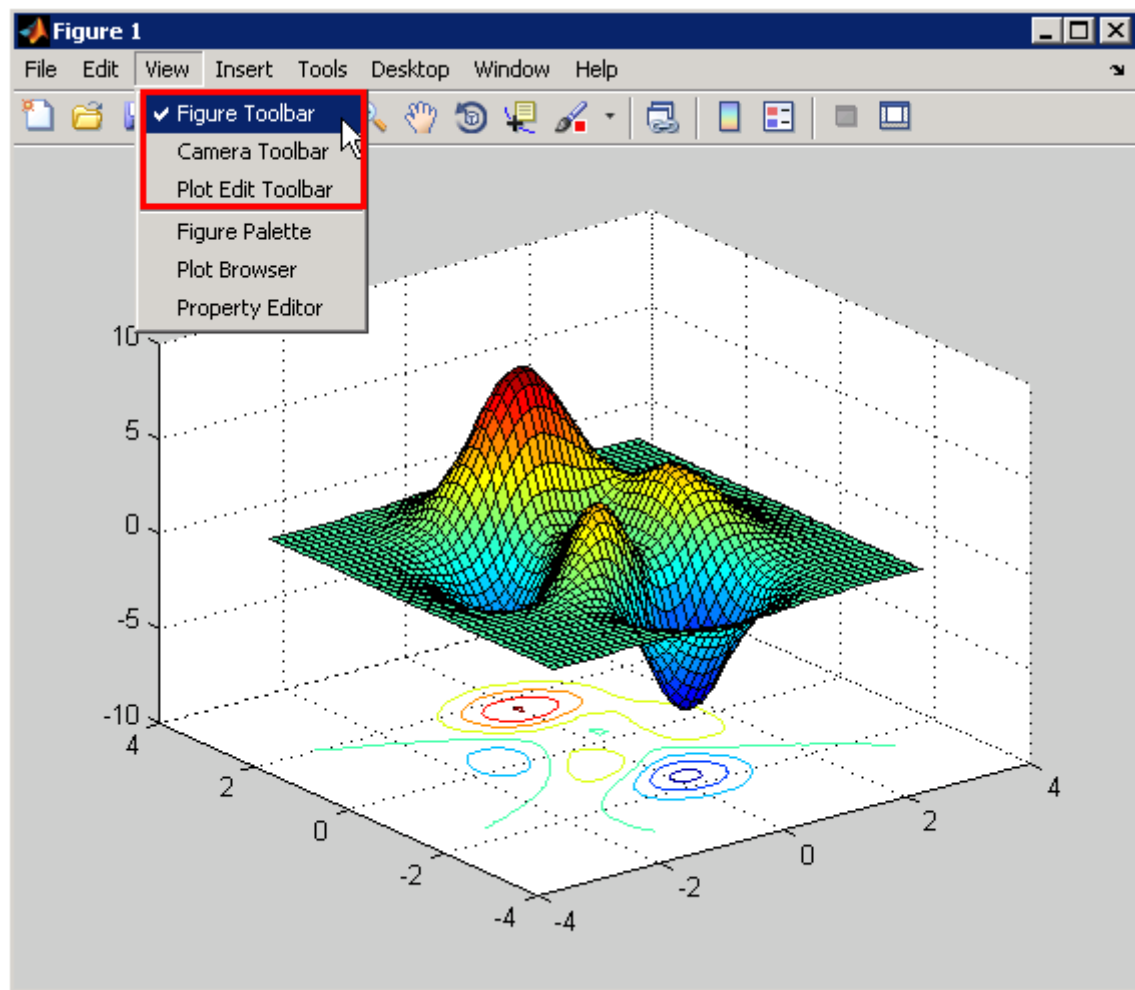
If you click on the button **More Properties...** you will see a small **Inspector** window:



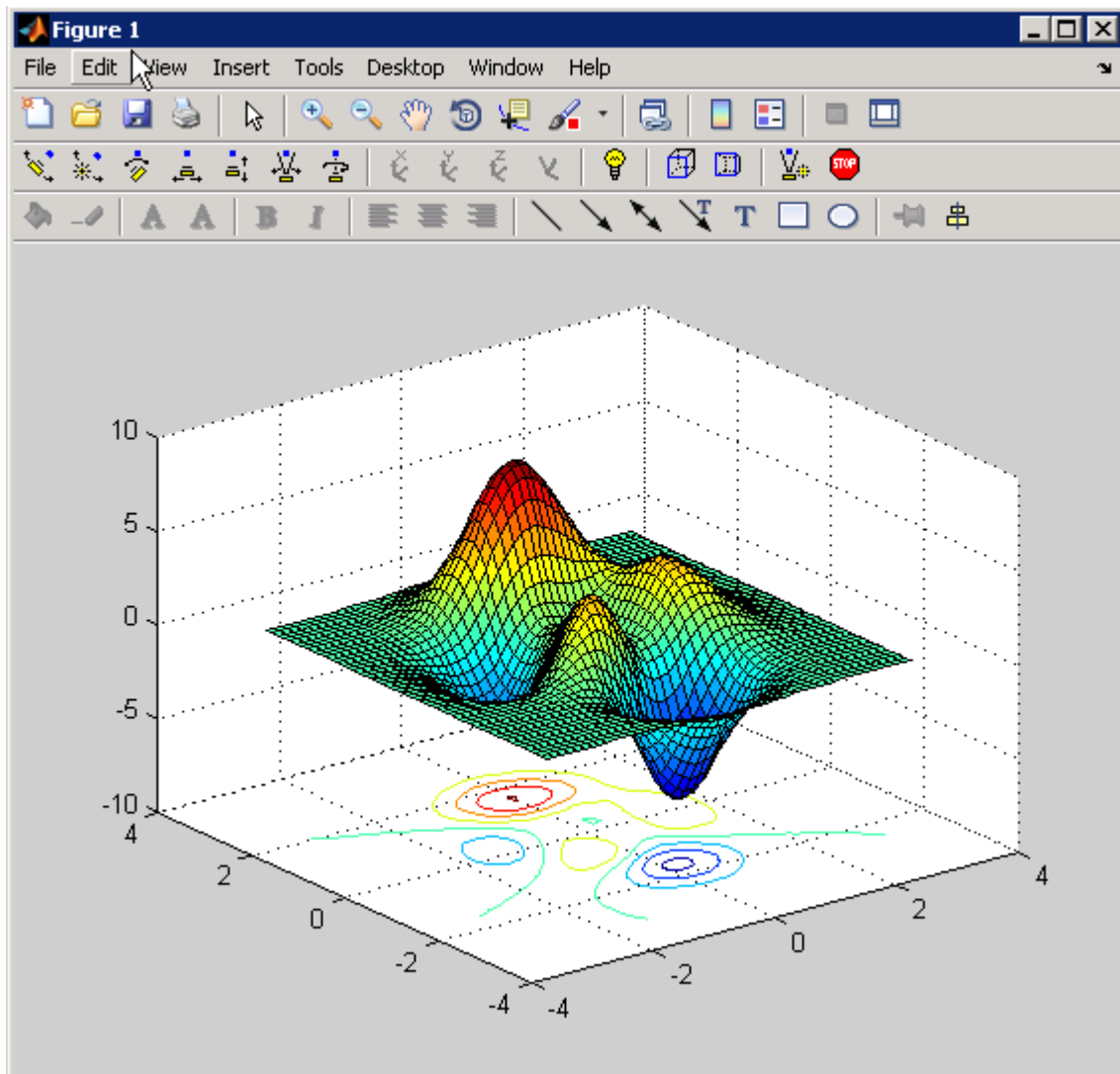
Where you can change almost a hundred of properties for your charts (for example we changed here the transparency of the surface by playing the **FaceAlpha** property):



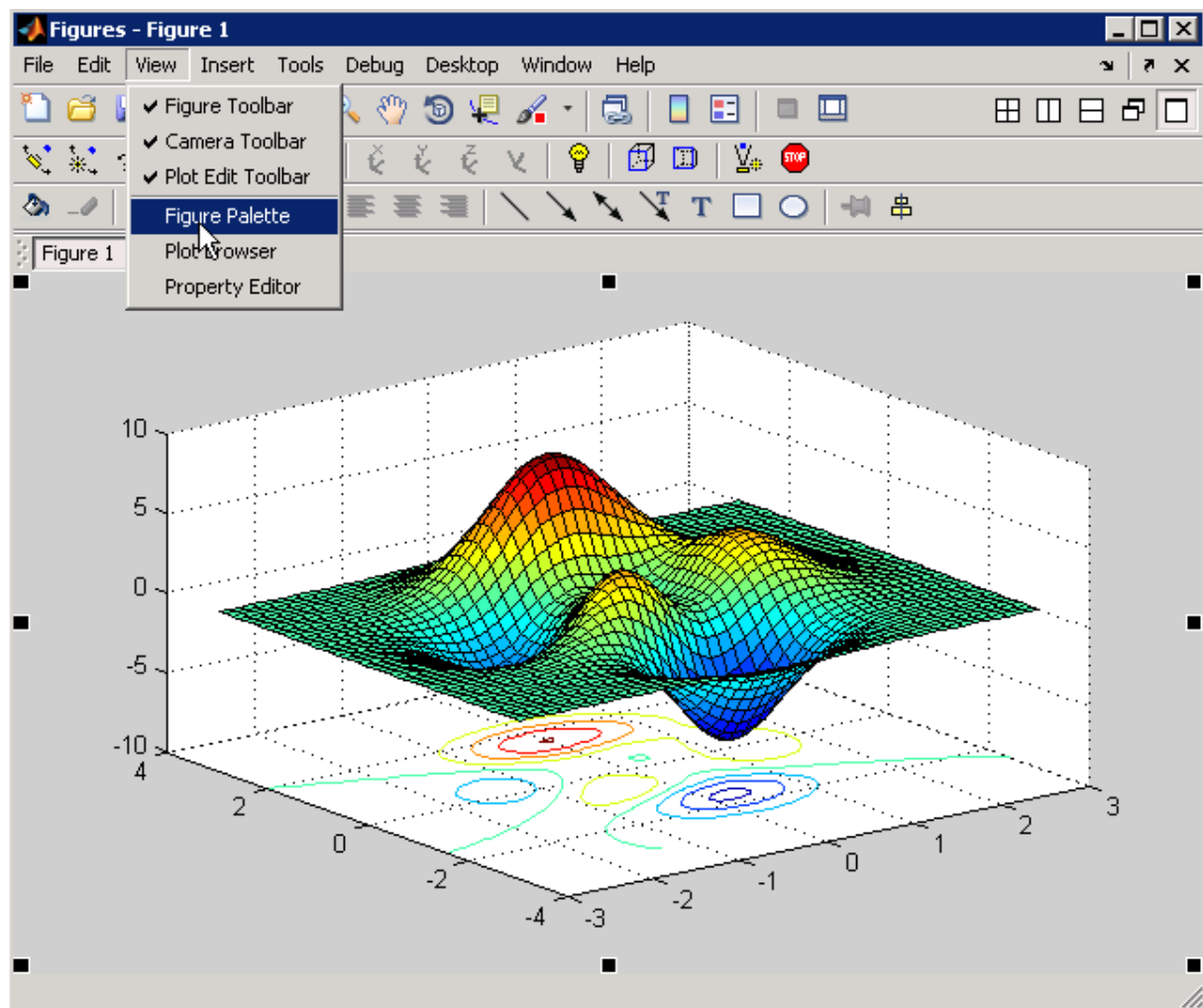
Ok, now see something else a structural or cosmetic properties. Have a look at the available toolbars:



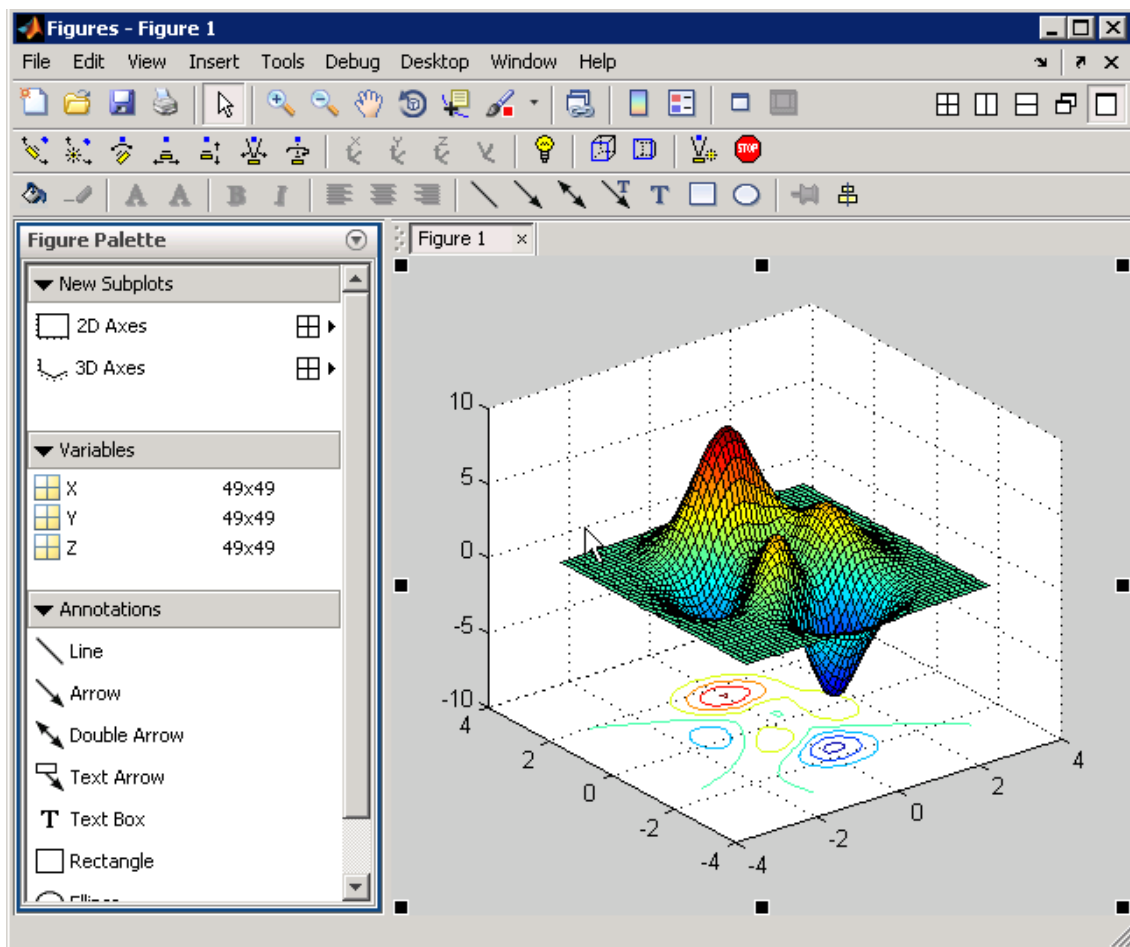
If you activate all the toolbars the result will looks like following:



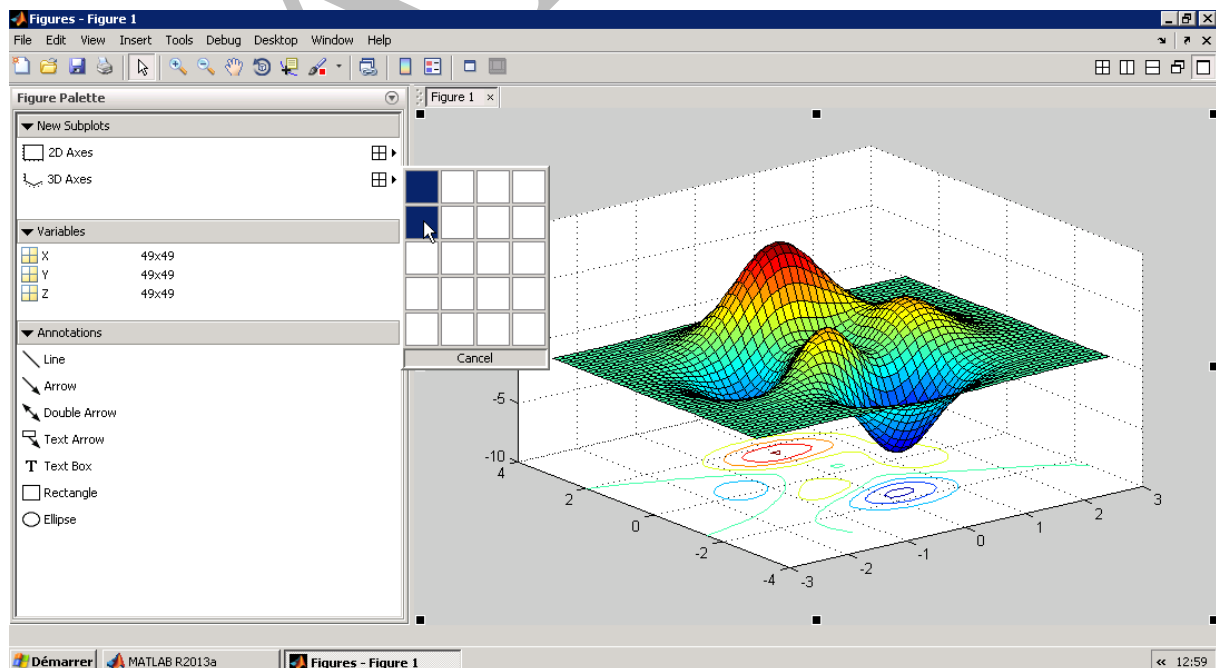
There is also a **Figure Palette** especially useful to create subplots with the wizard:



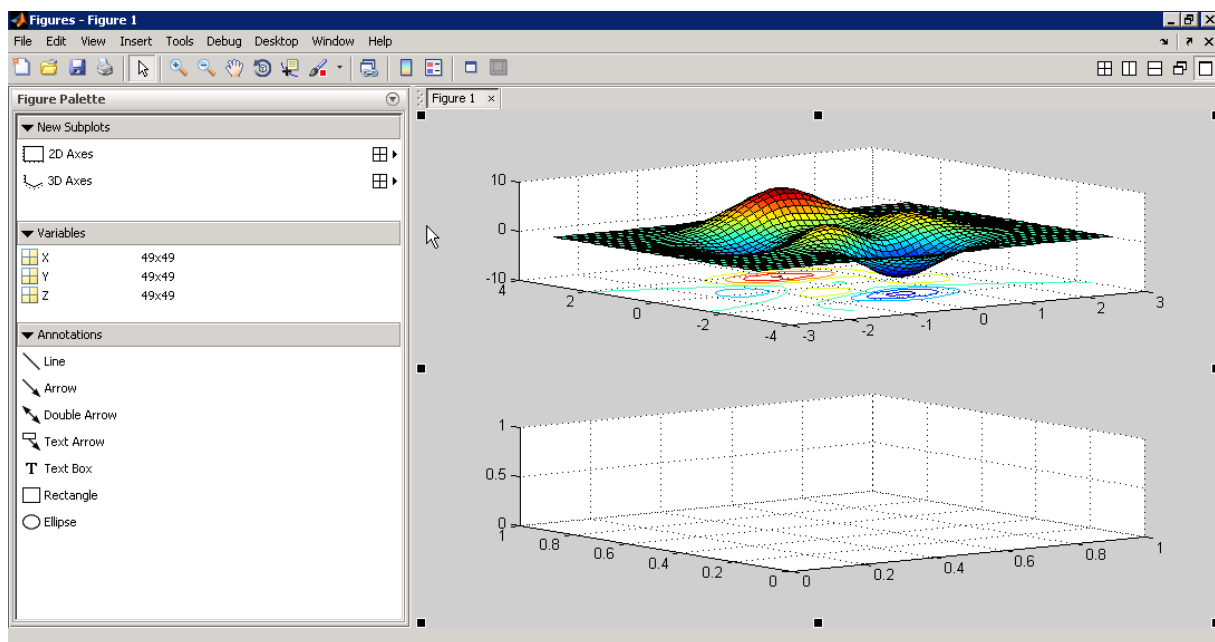
The **Figure Palette** looks like this:



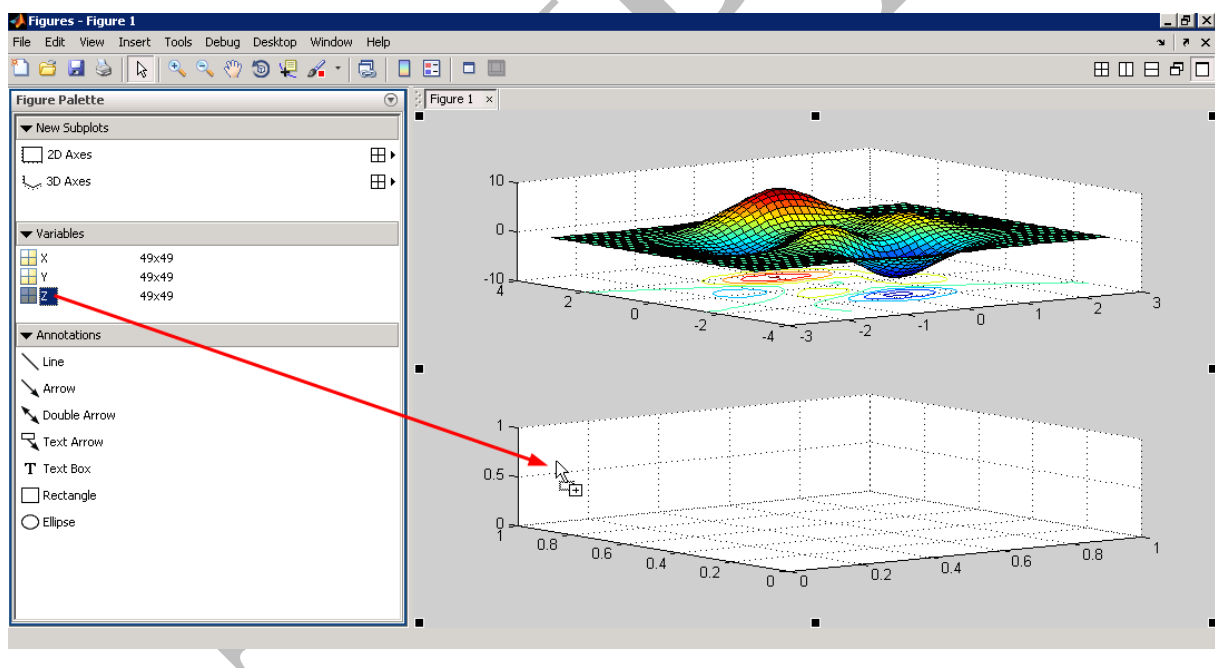
If you click on 3D axes arrows of the **Figure Palette** panel you will be able to create a few subplots because (you can also mix 3D and 2D subplots in the same figure!):



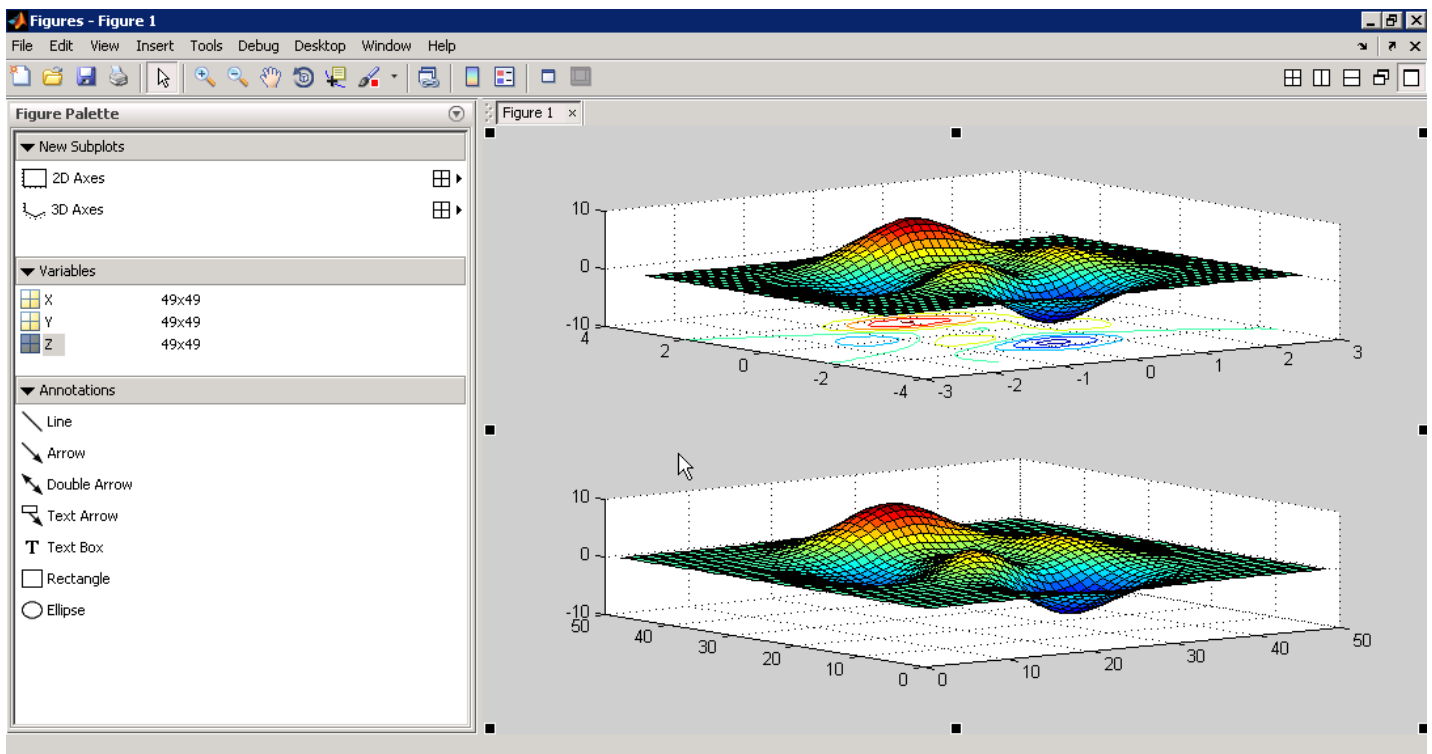
For example with an additional toolbox the result looks like this:



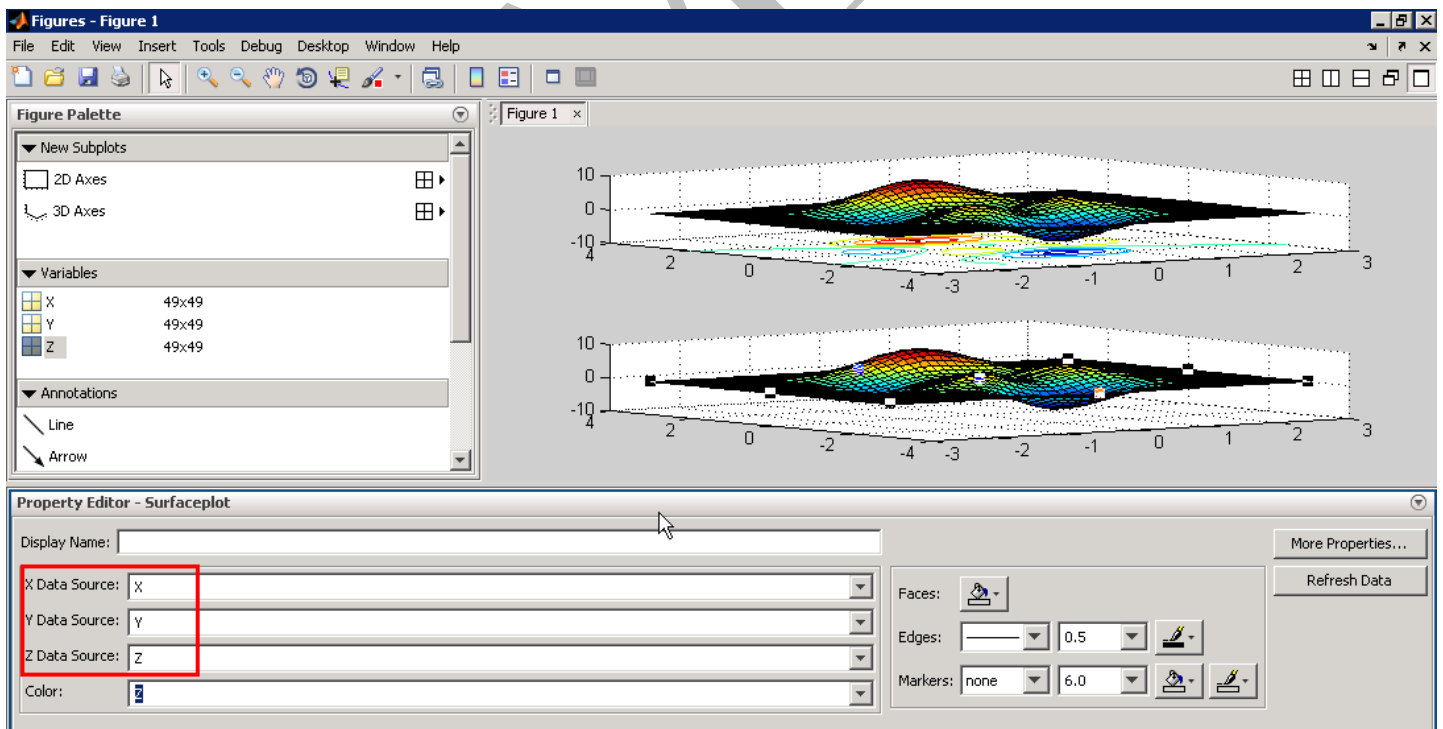
Then you just have to drag the components that you want to plot in this new 3D or 2D subplot:



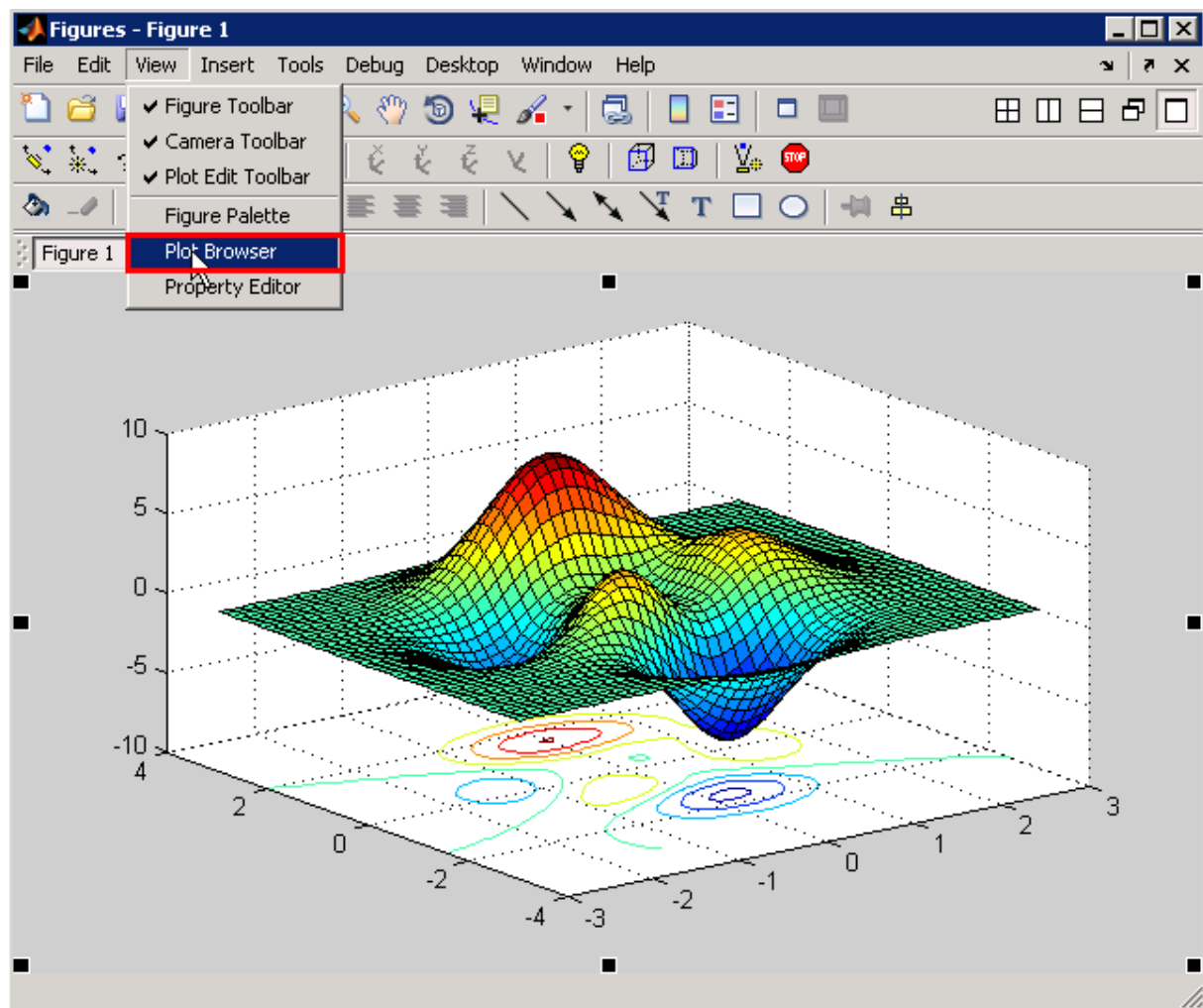
Here for example we just reproduced the above one (for sure this example is almost stupid...):



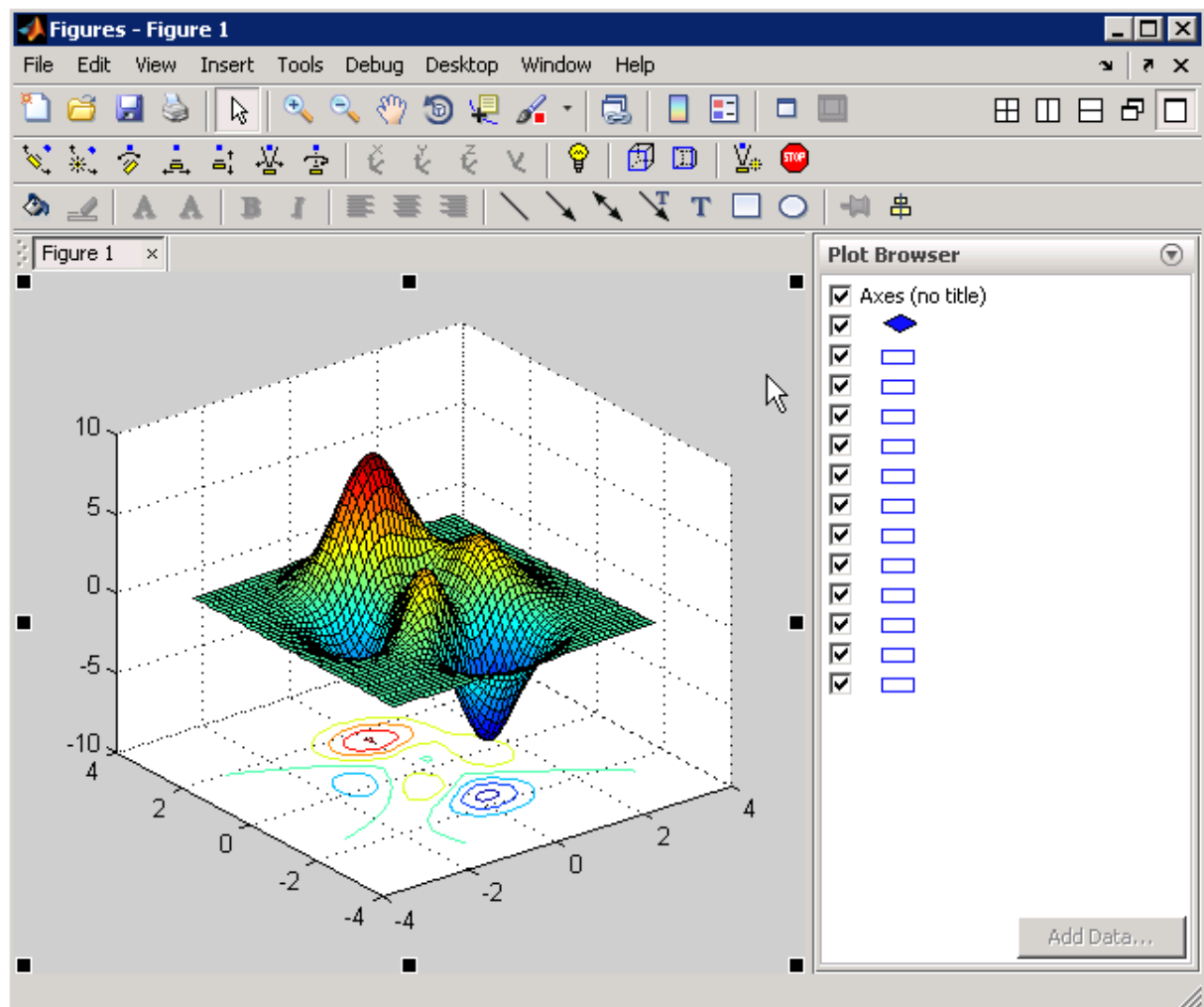
Now if you double click on the surface you can always change after the source values for the second plot to have the correct X, Y and Z components as you can see below:



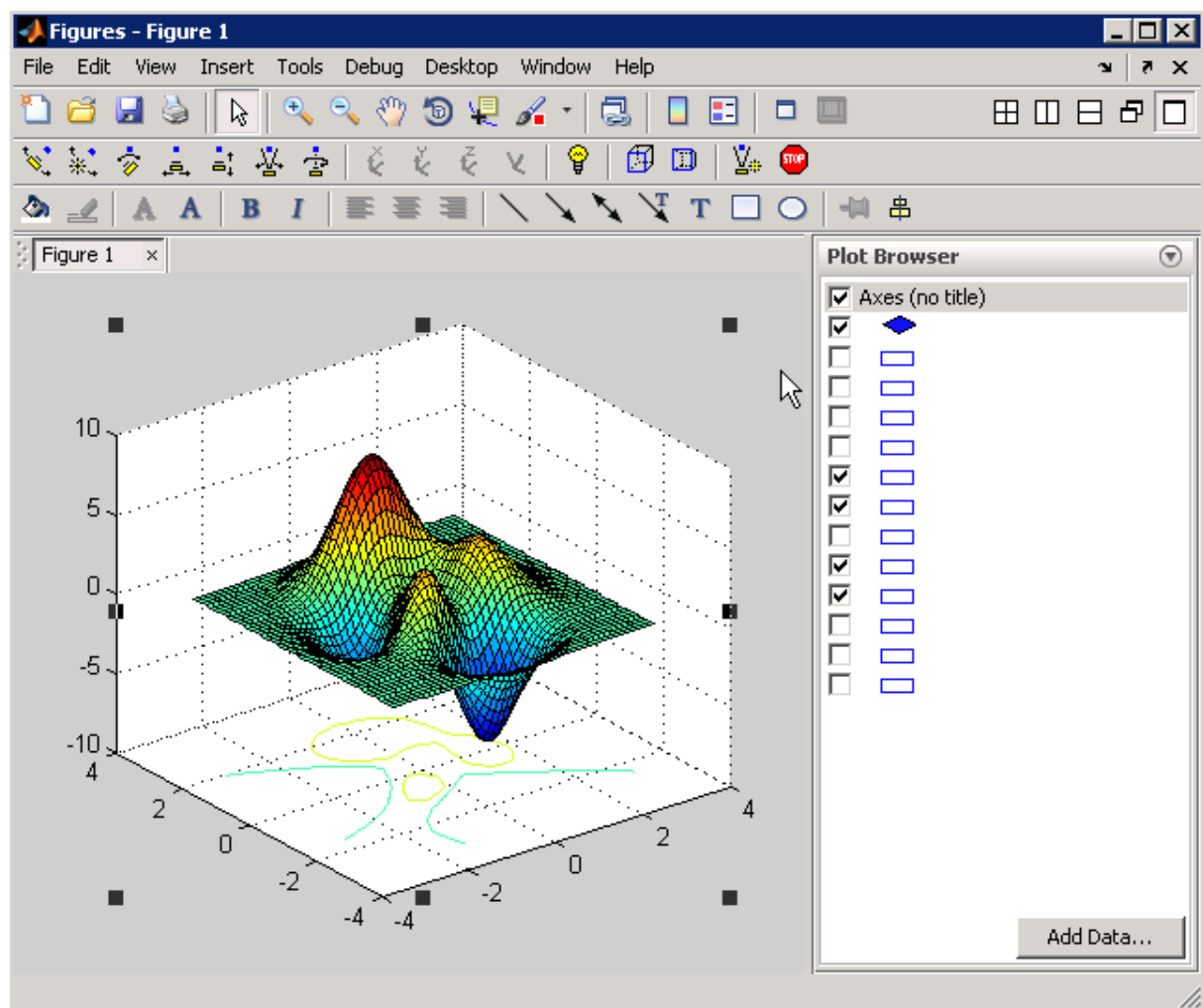
Now take a look the **Plot Browser**:



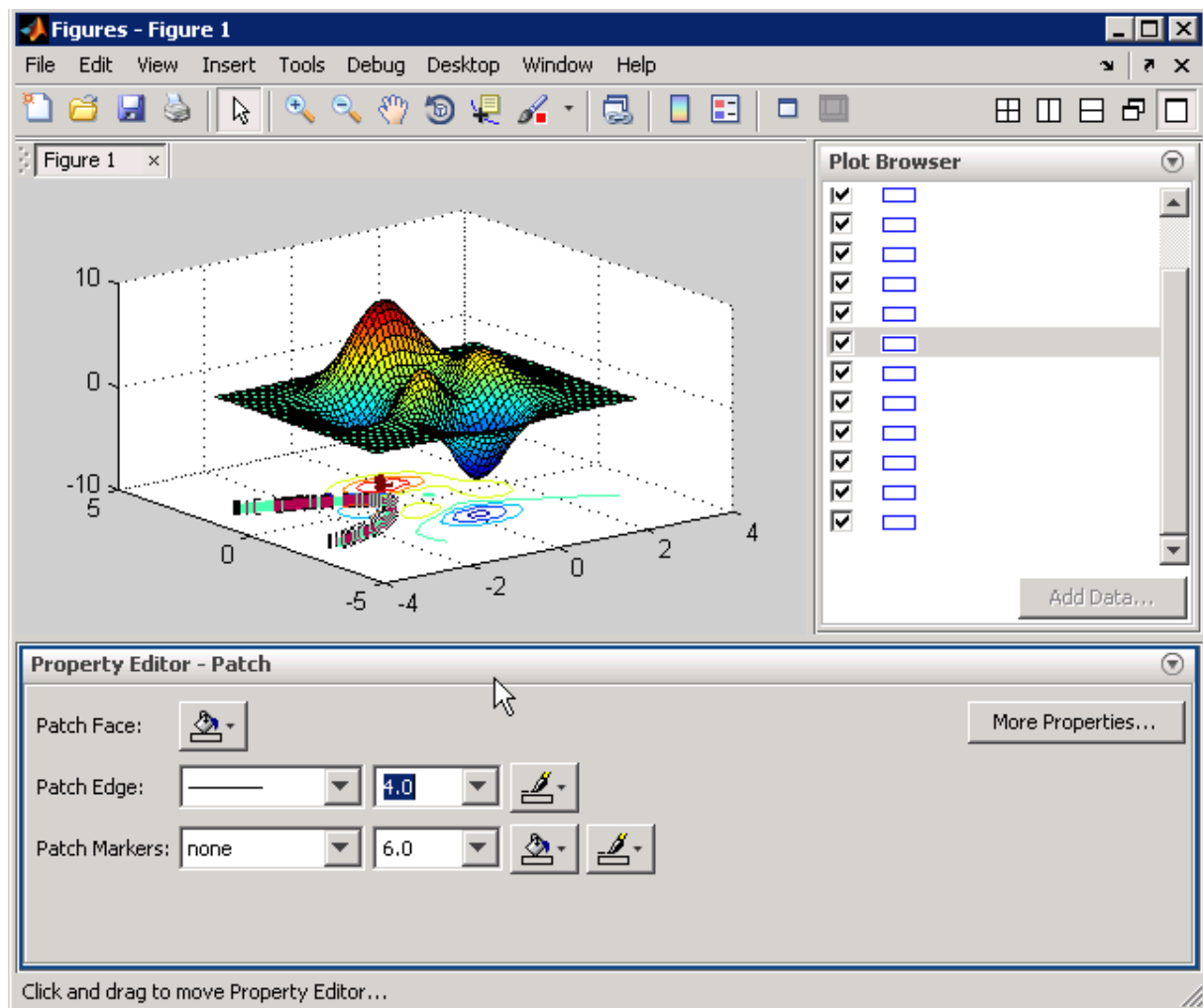
This browser works in a similar way as layers for Adobe Photoshop but with a very negative point: **you can't give a name or rename the different "layer"...**



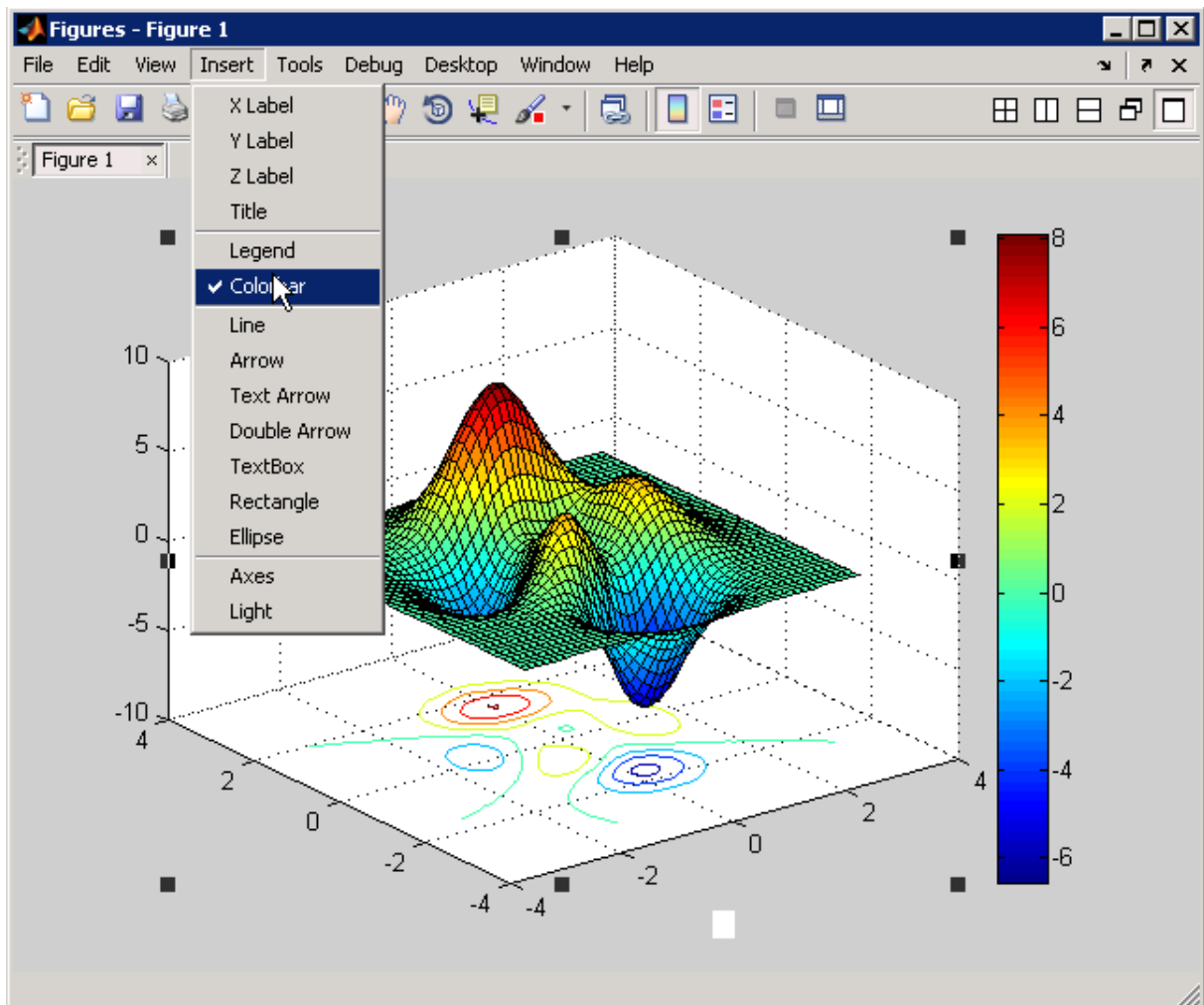
Below we have for example disable some projected isoclines:



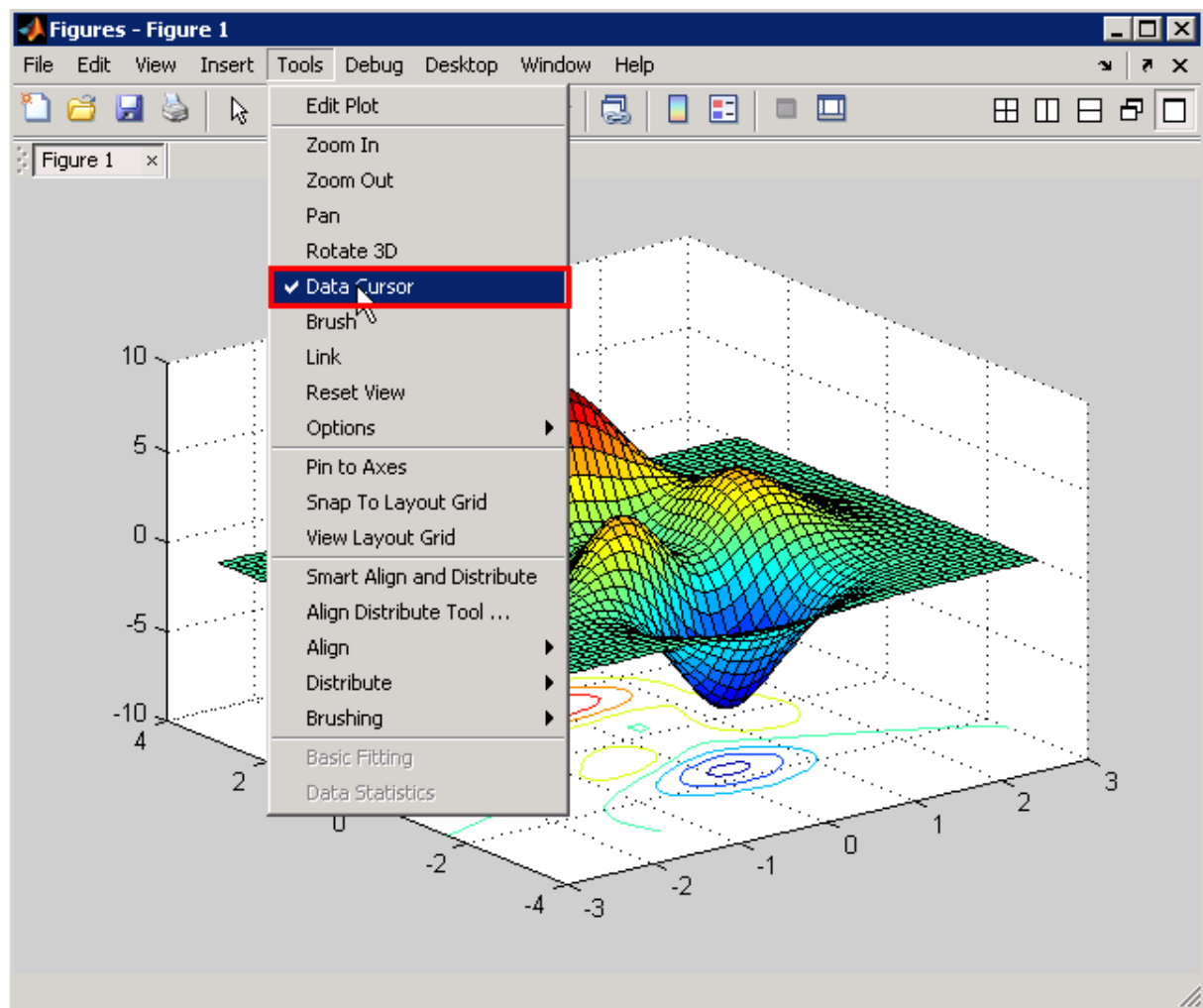
By double click on one of the layers you will open the bottom **Property Editor** if it is not already activated:



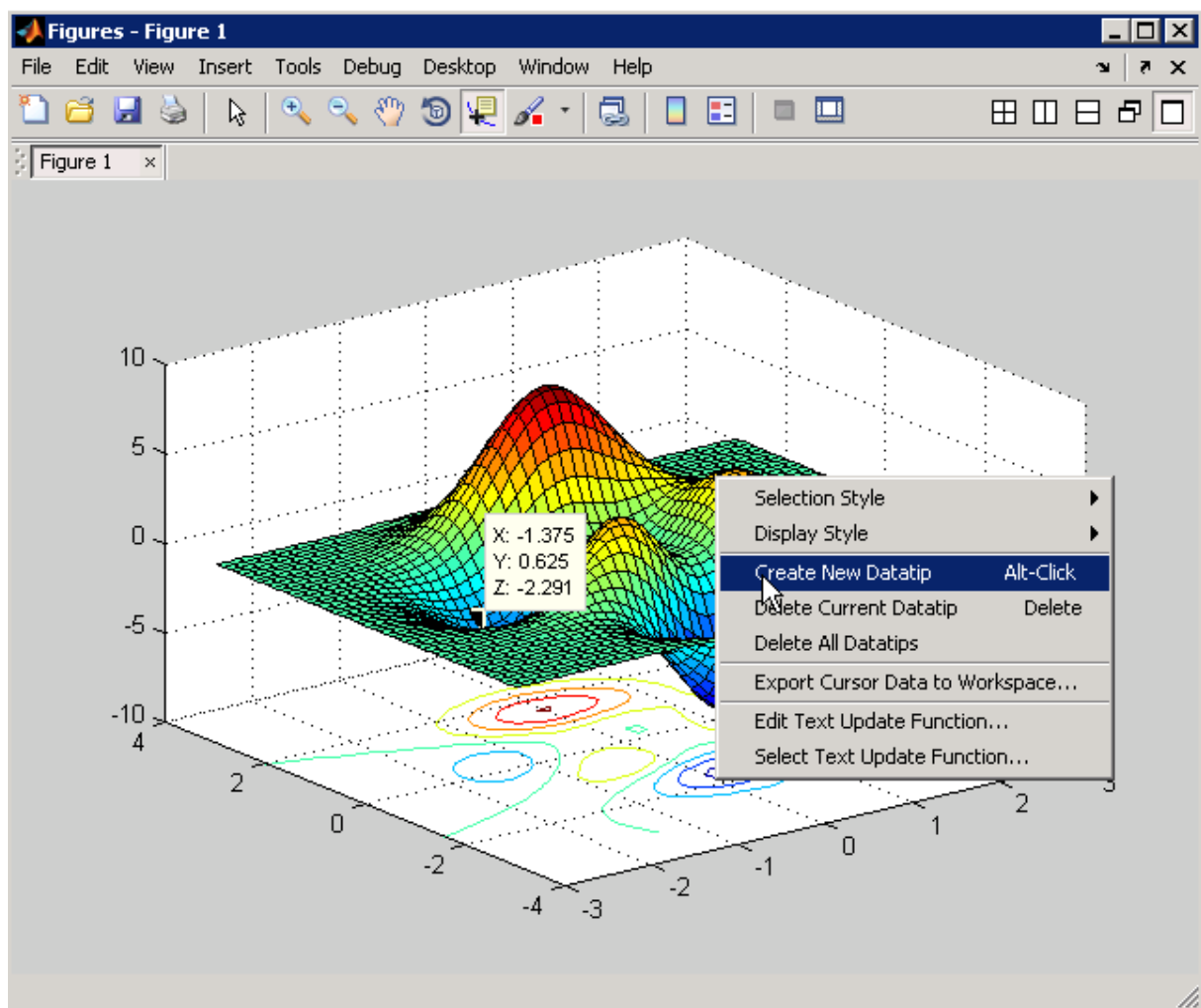
In the menu **Insert** the first option that people that don't like to create chart using scripting almost always use is the **Colorbar** (the other options are too easy to guess to make examples):



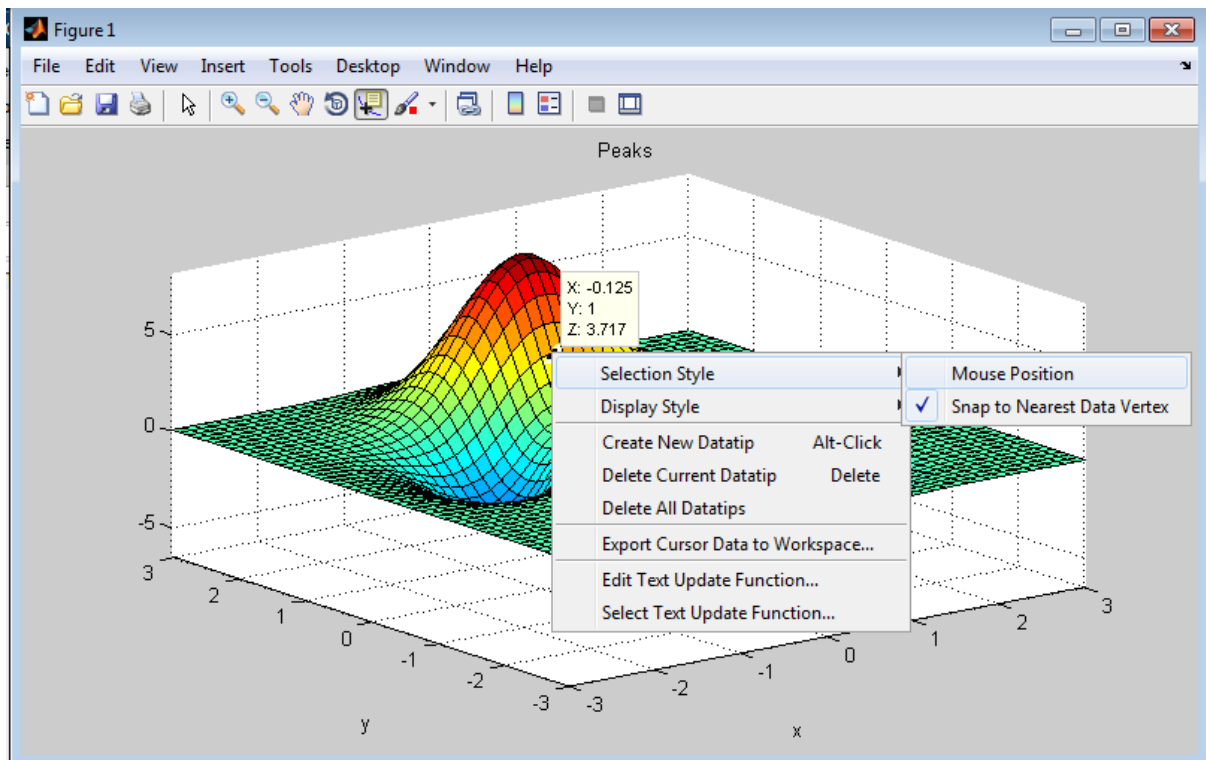
Now take a look to the very useful option **Data Cursor** of the menu **Tools**:



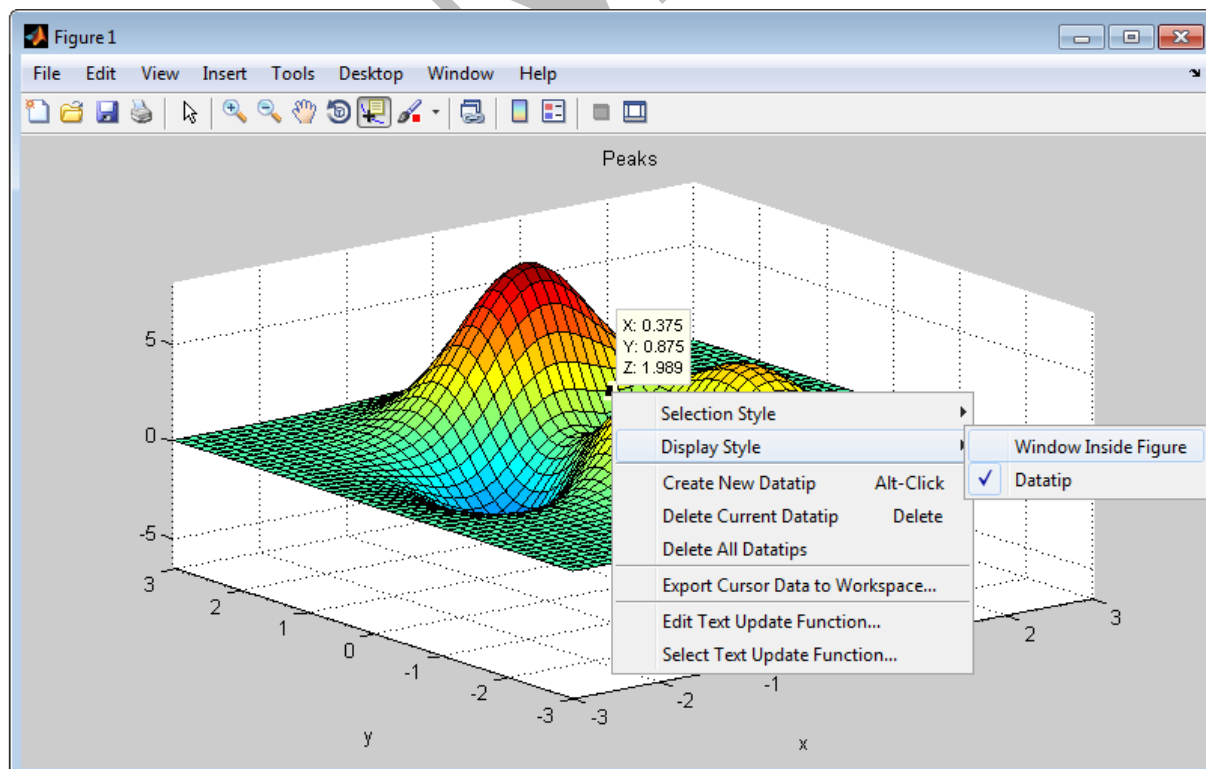
This tool gives you the possibility to add data tooltips on the chart (as many as you want):



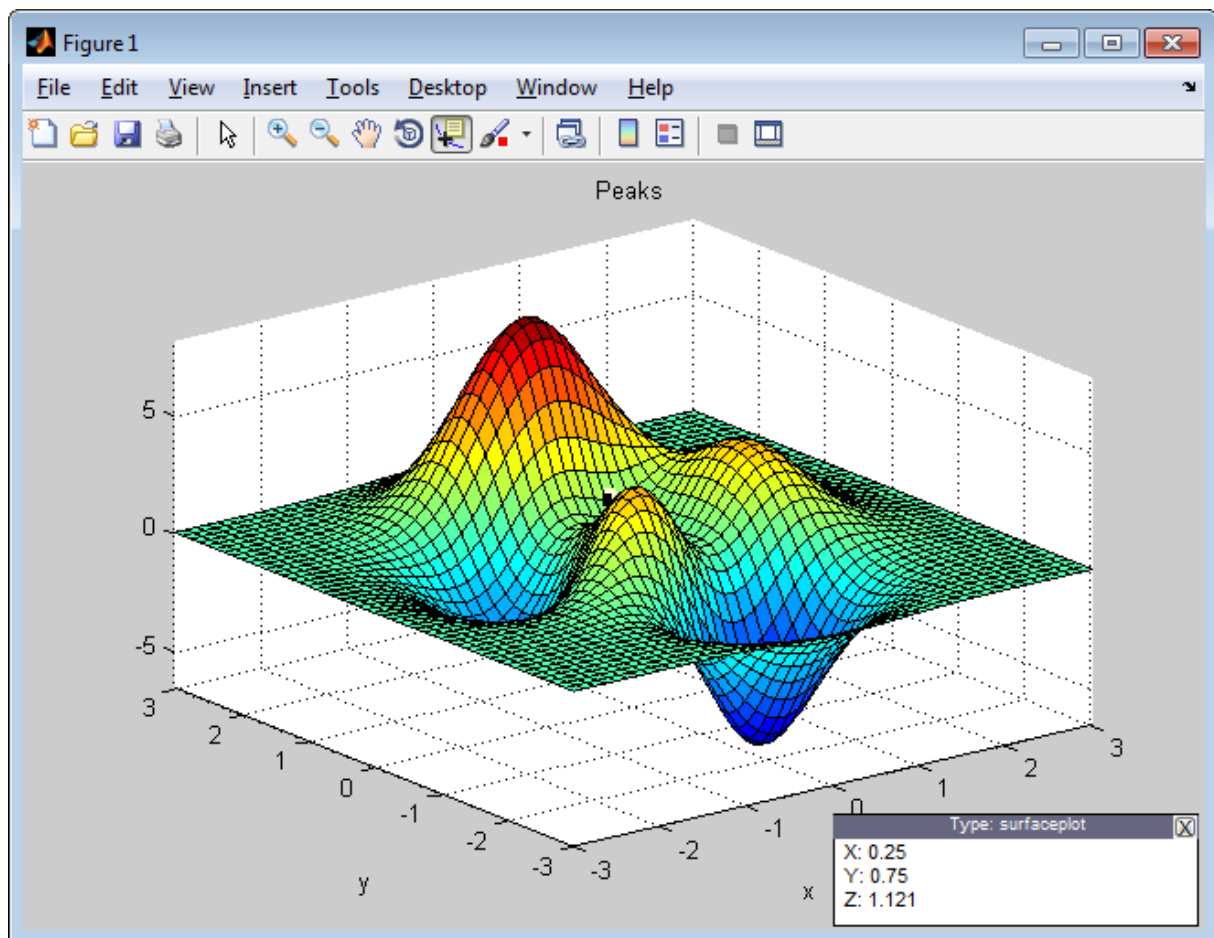
There are also a few options for these data tip. For example by default it is on Snap to **Nearest Data Vertex** but you can choose to click anywhere you want on the (surface) plot by selecting **Mouse Position**:



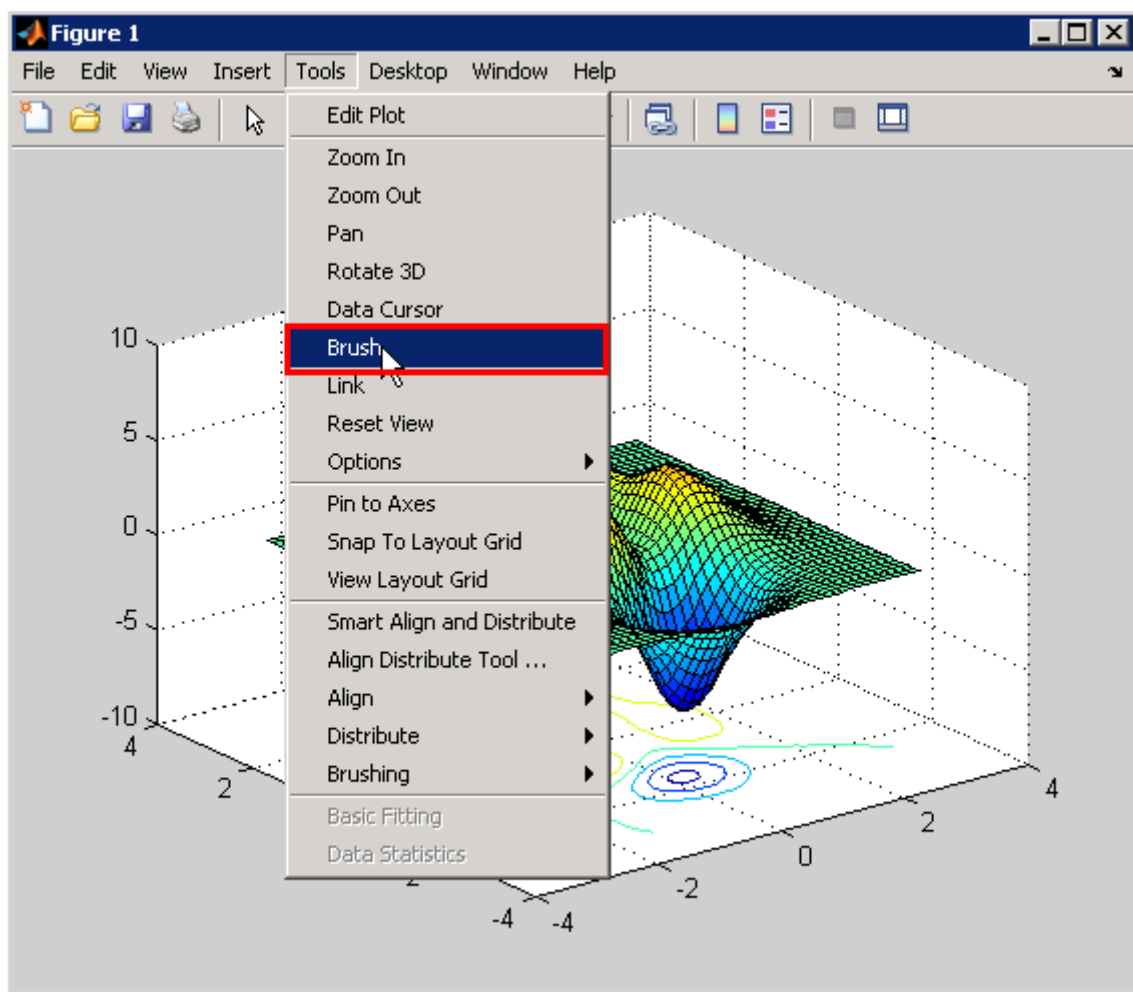
And there are also different ways to show the data type. You can choose for example the see the coordinates as a **Window Inside the Figure**:



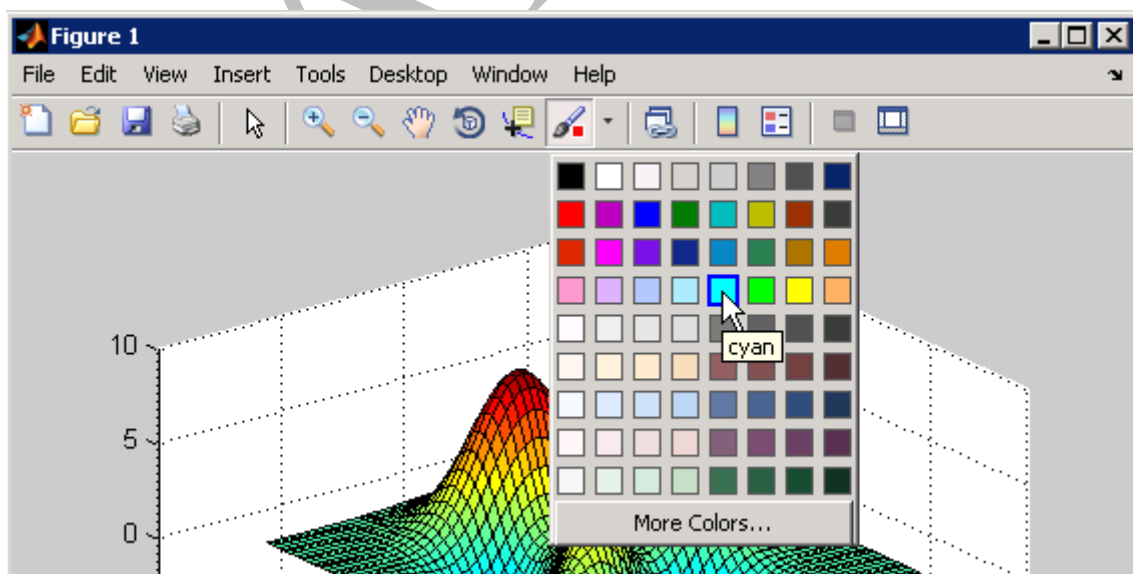
This will result to the following (see the bottom right corner):



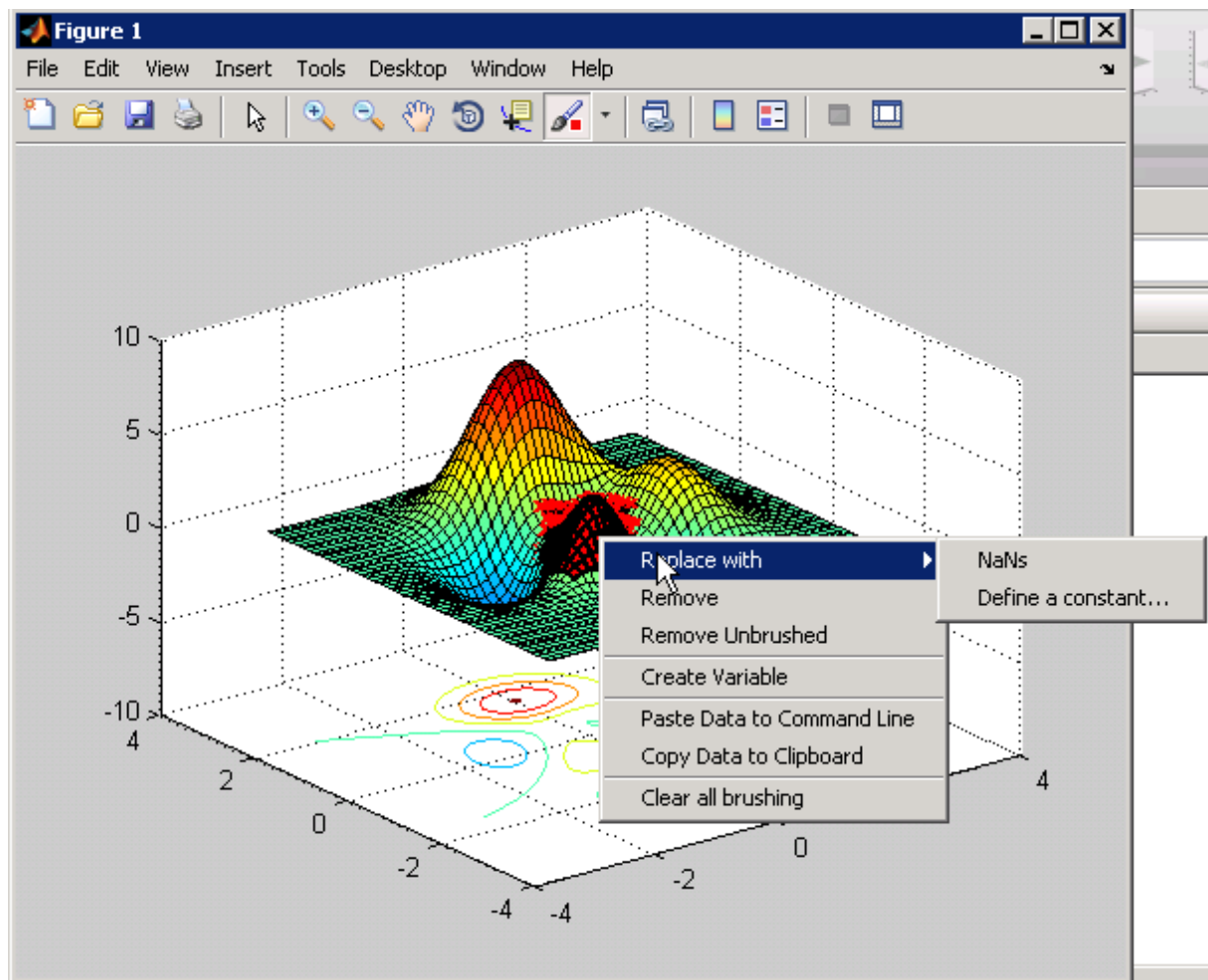
Now just take a look to the **Brush** option:



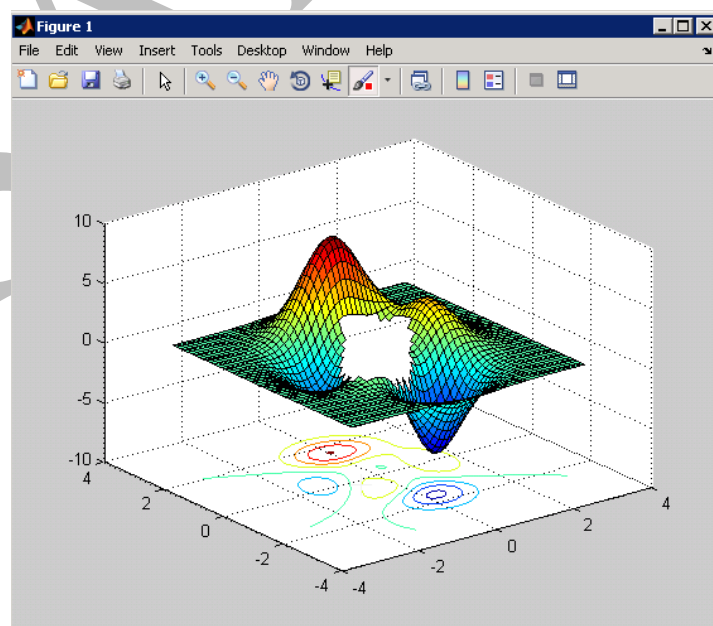
That you can also find in the main toolbar and choose a color:



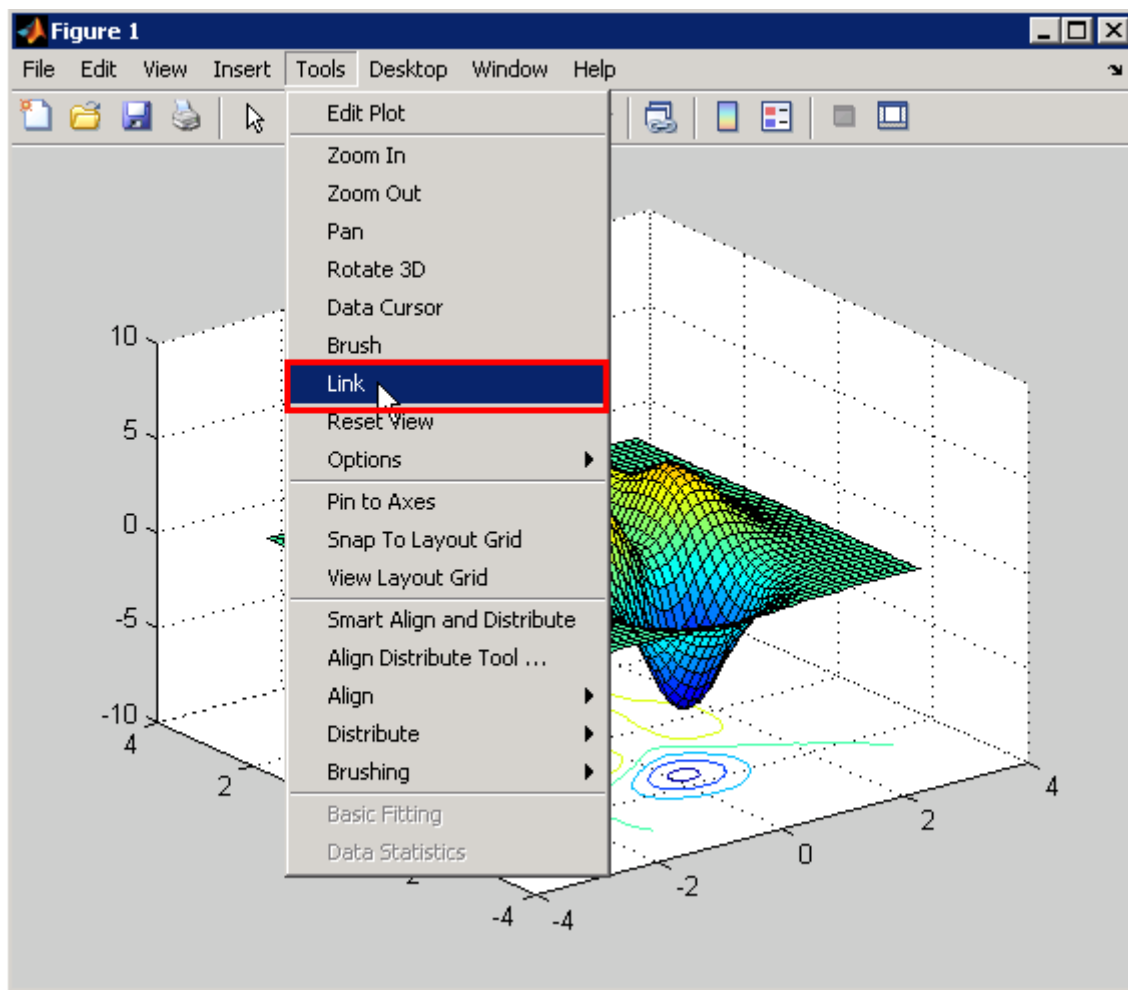
Once you have finished to paint you can also make a right click and choose some option:



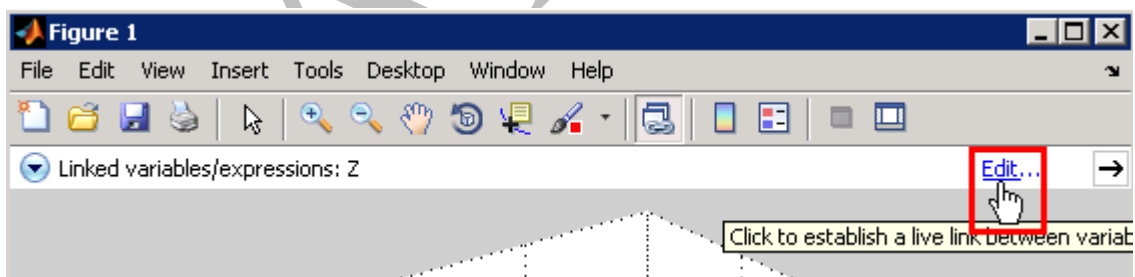
You can see below what happens when you select **Replace with/NaNs** for example:



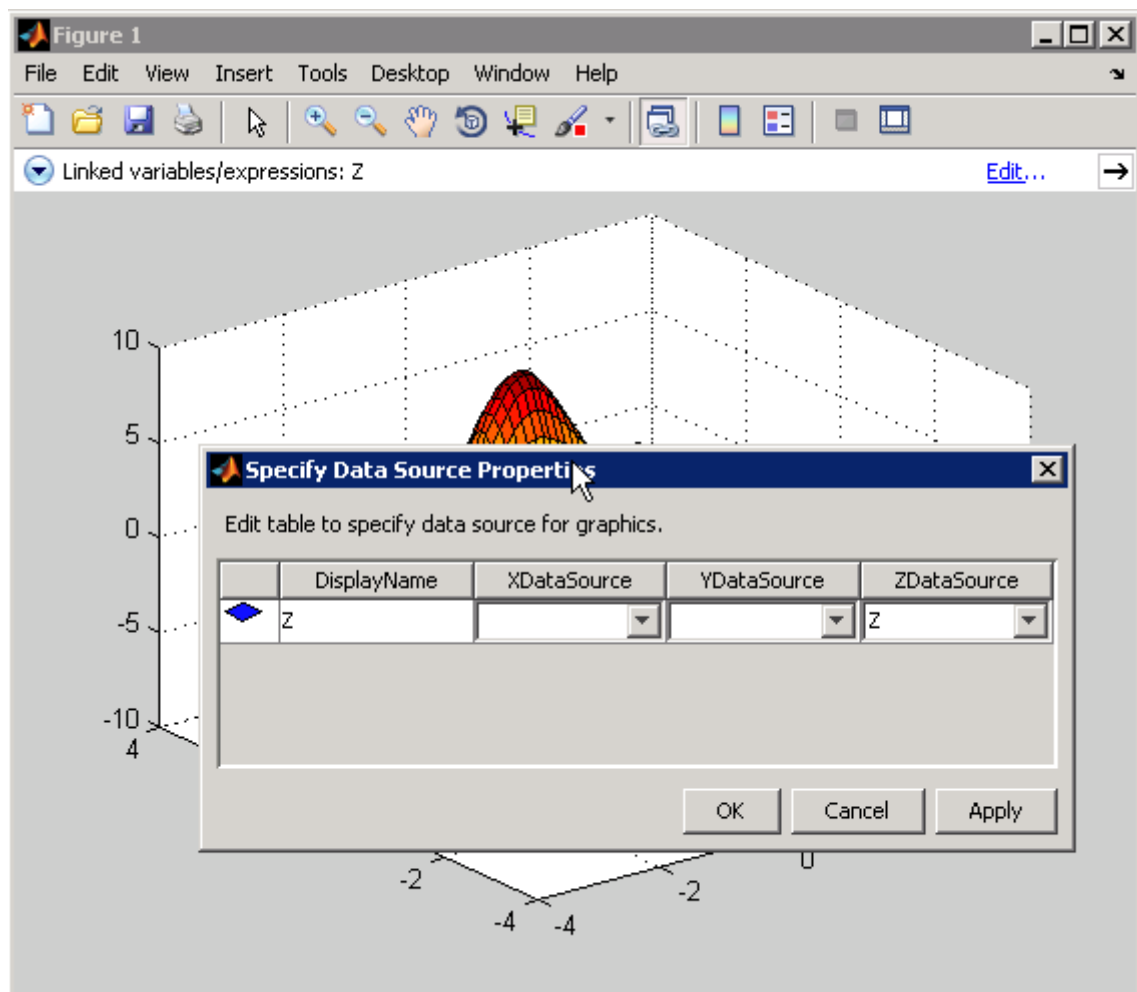
Now have a look to the option named **Link**:



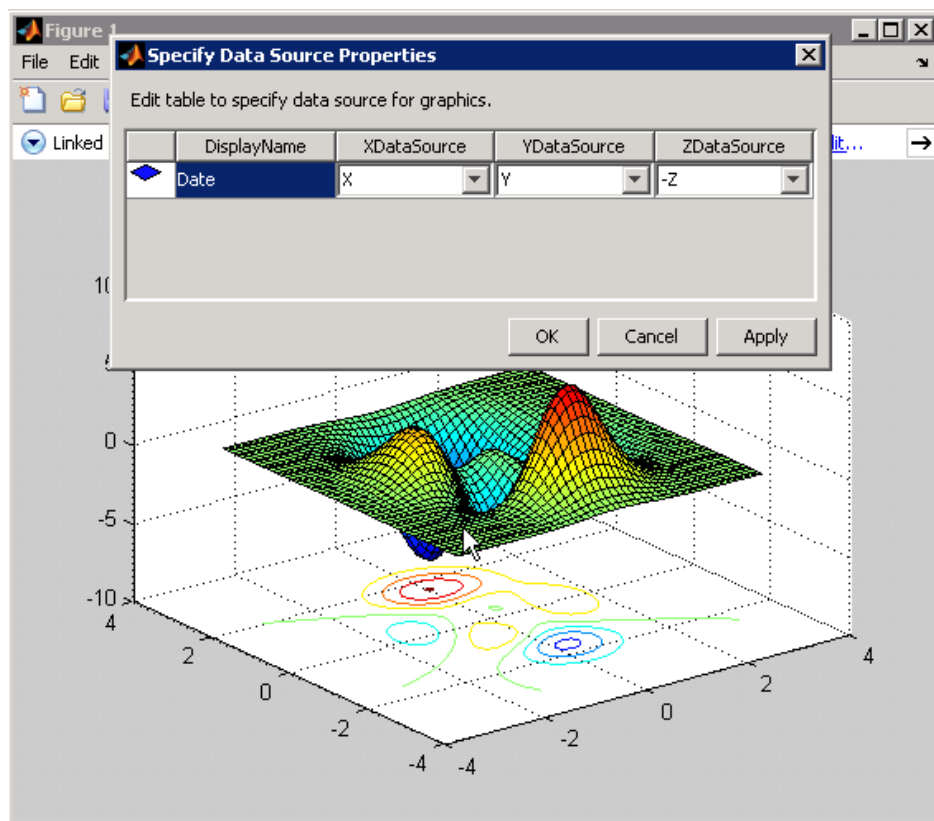
You will see then a small white band appear to the top. Click on **Edit...**:



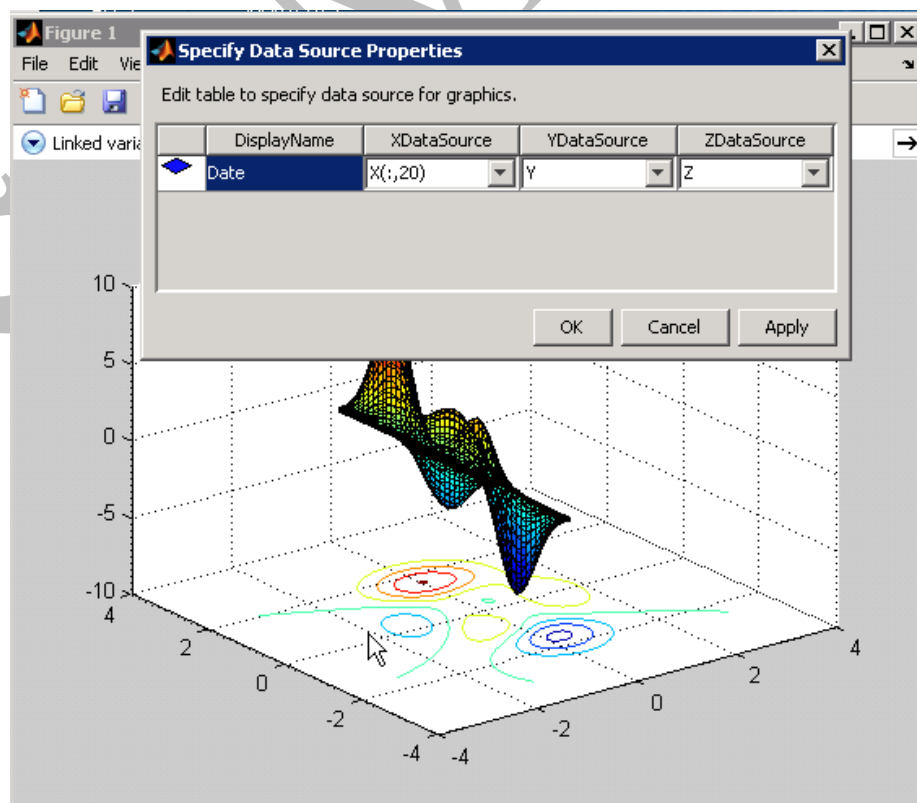
And whaa... what a surprise (it's ironic...) we just get an option to change the data source variable names to plot the chart... (same as we already saw at the beginning with this wizard):



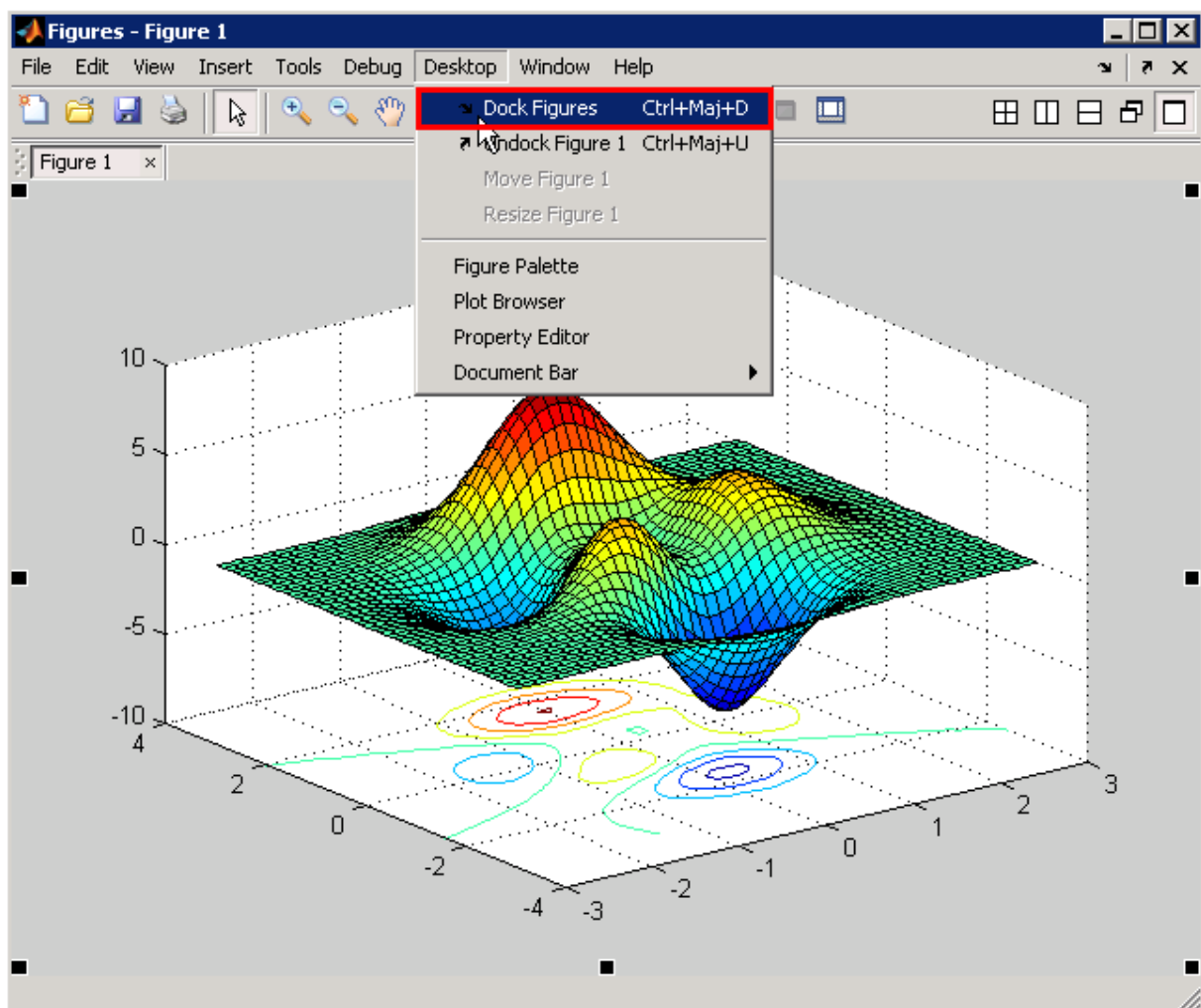
We can check for sure that it works well by defining the variables sources and playing with one of them by taking the opposite sign. Then the plotted surface will be reversed:



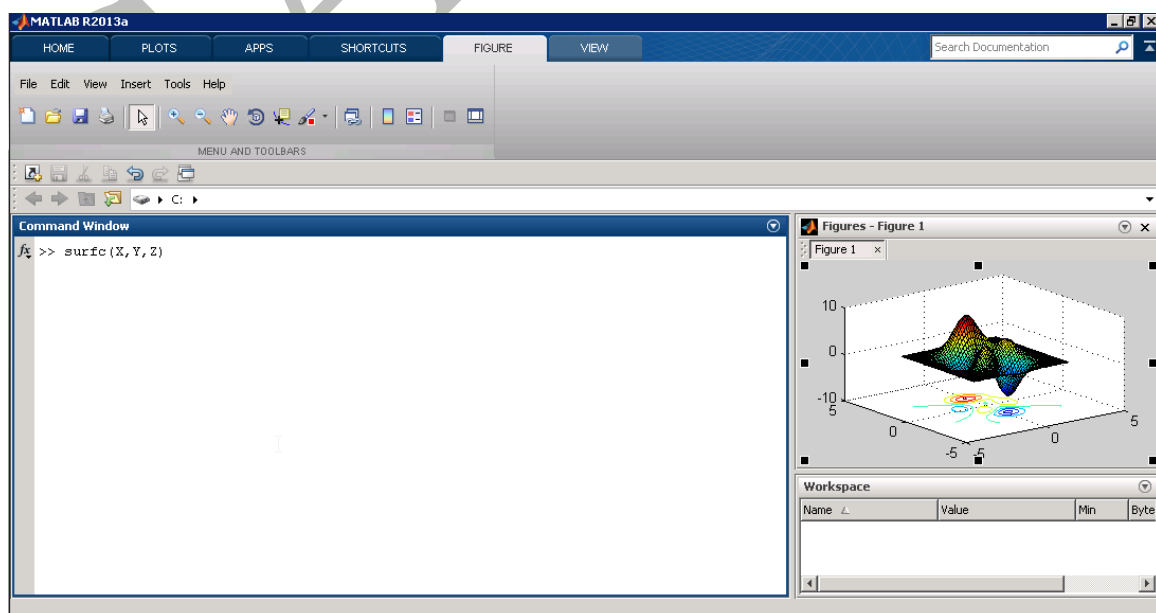
Or imagine that we are in flat world ;-):



Now a look to the option in the menu **Desktop/Dock Figures**:



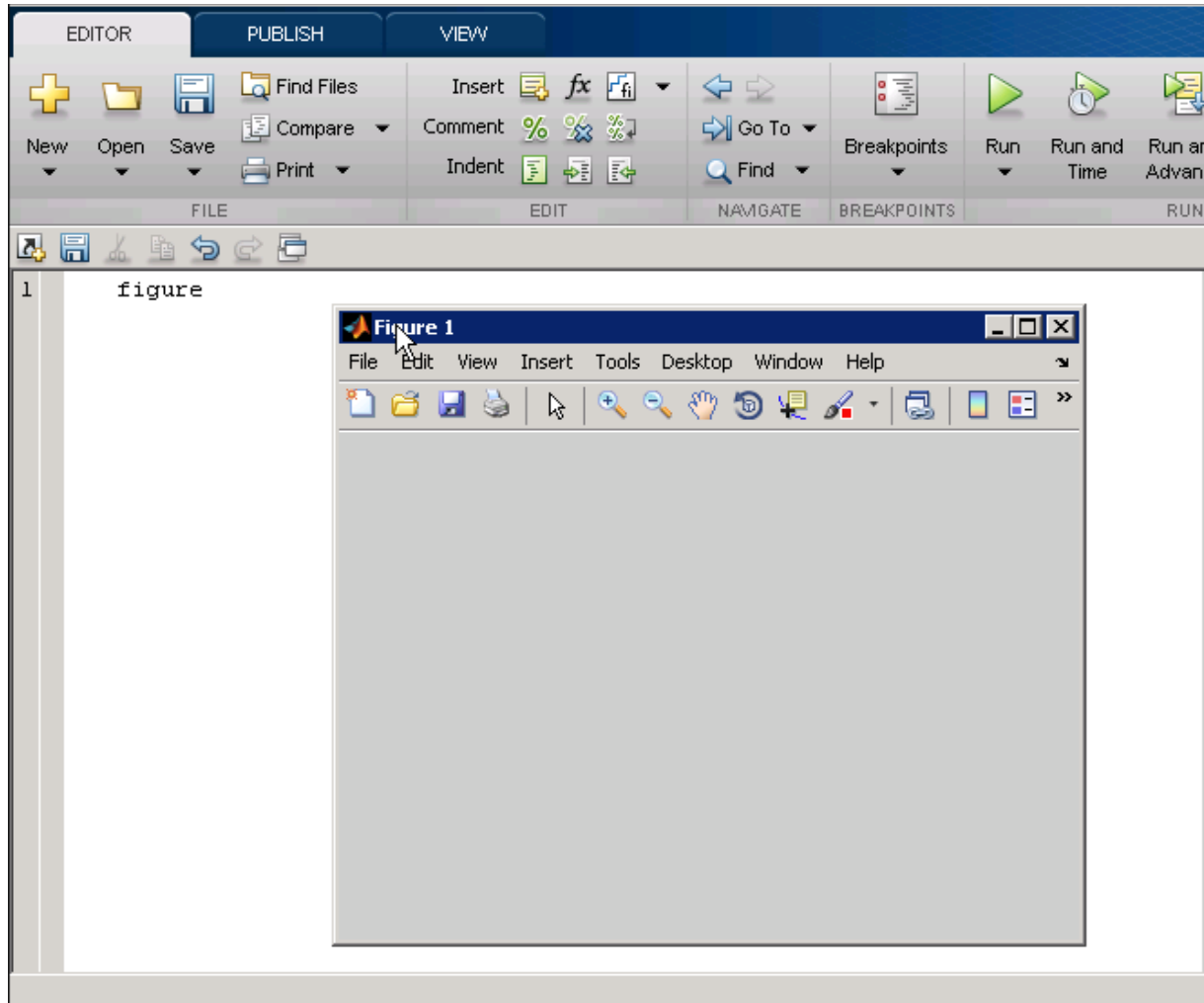
This will result in something that can be sometimes very useful! You have the chart and the command Windows easily always on the screen:



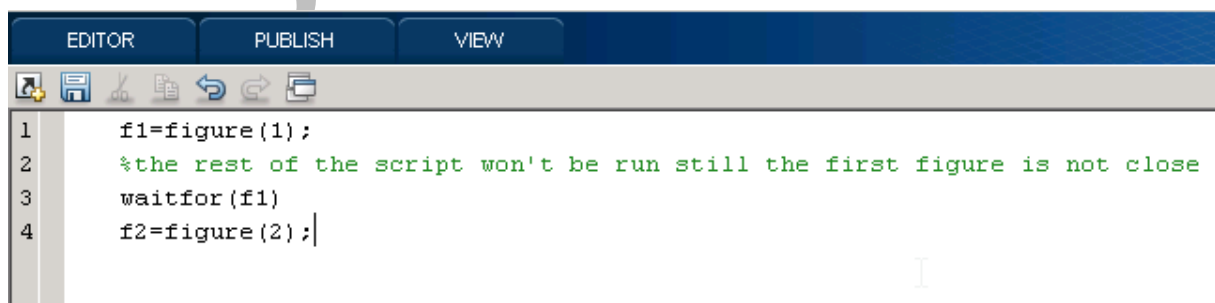
Control chart window

OK now it's time to begin to learn how to create charts and charts figure without any wizards.

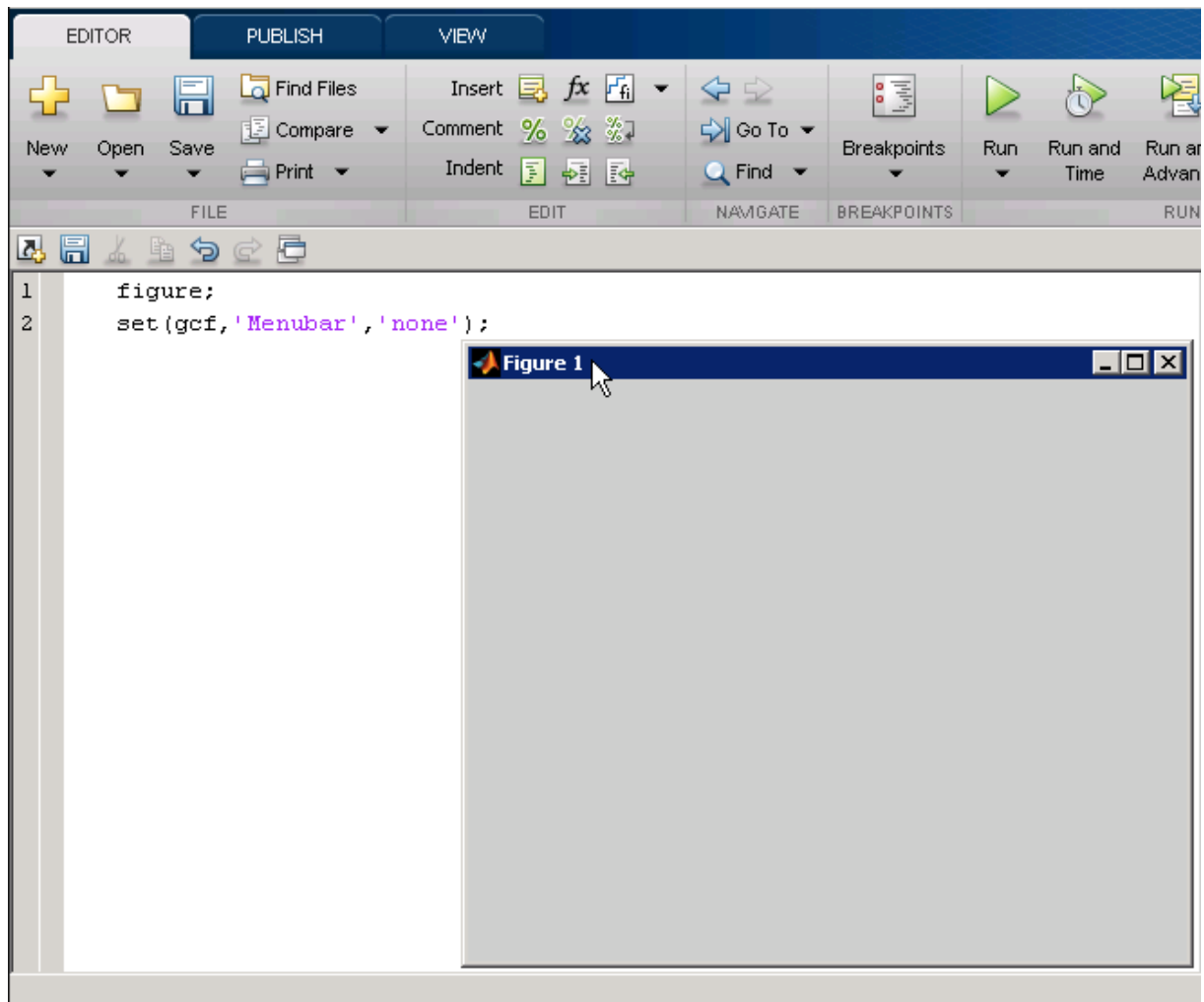
First to create an empty chart figure you can just use the command **figure** as below:



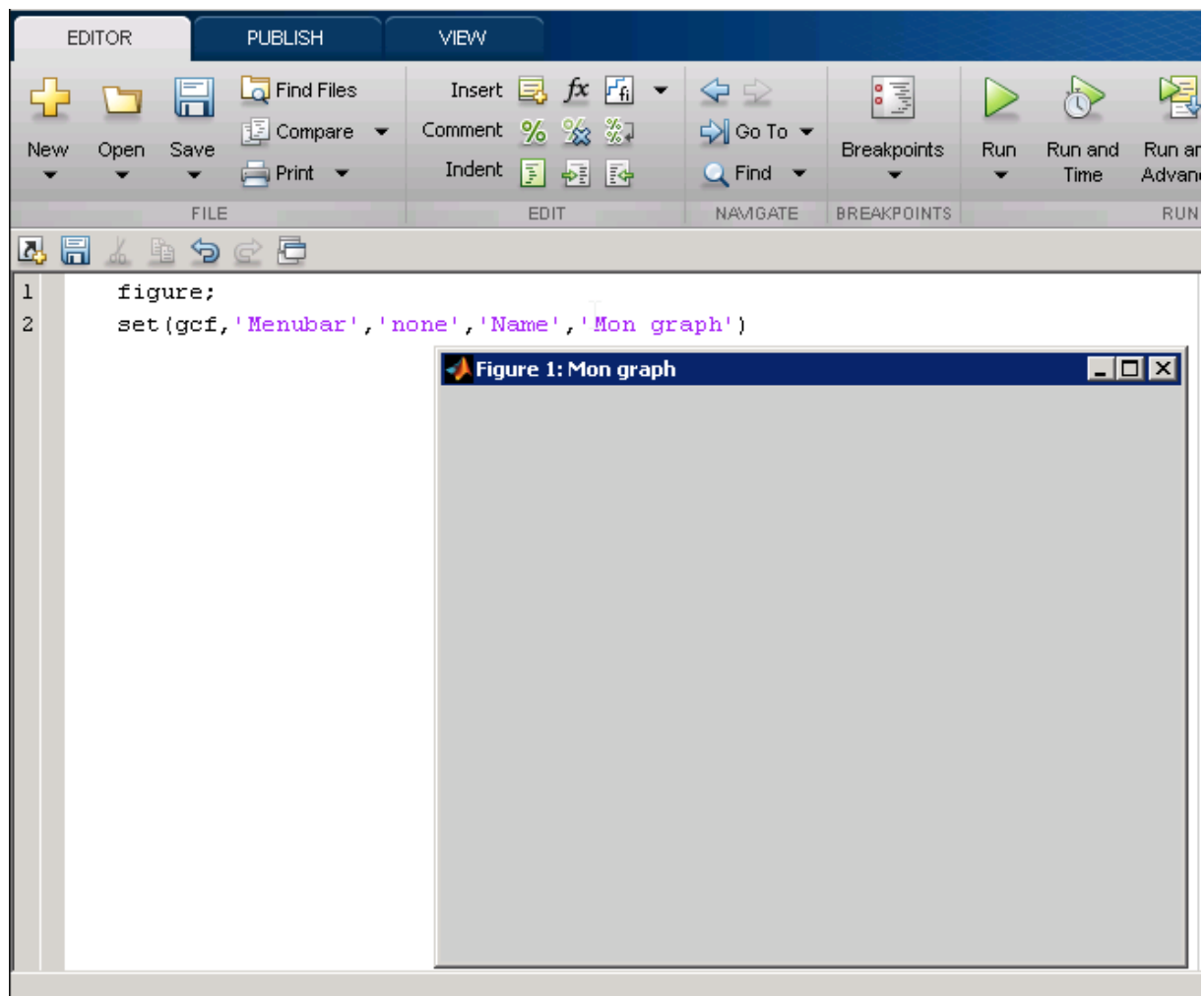
If you have to open multiple charts figure, using the function **waitfor()** you can be sure that the rest of the script won't be run until the first figure chart is not closed:



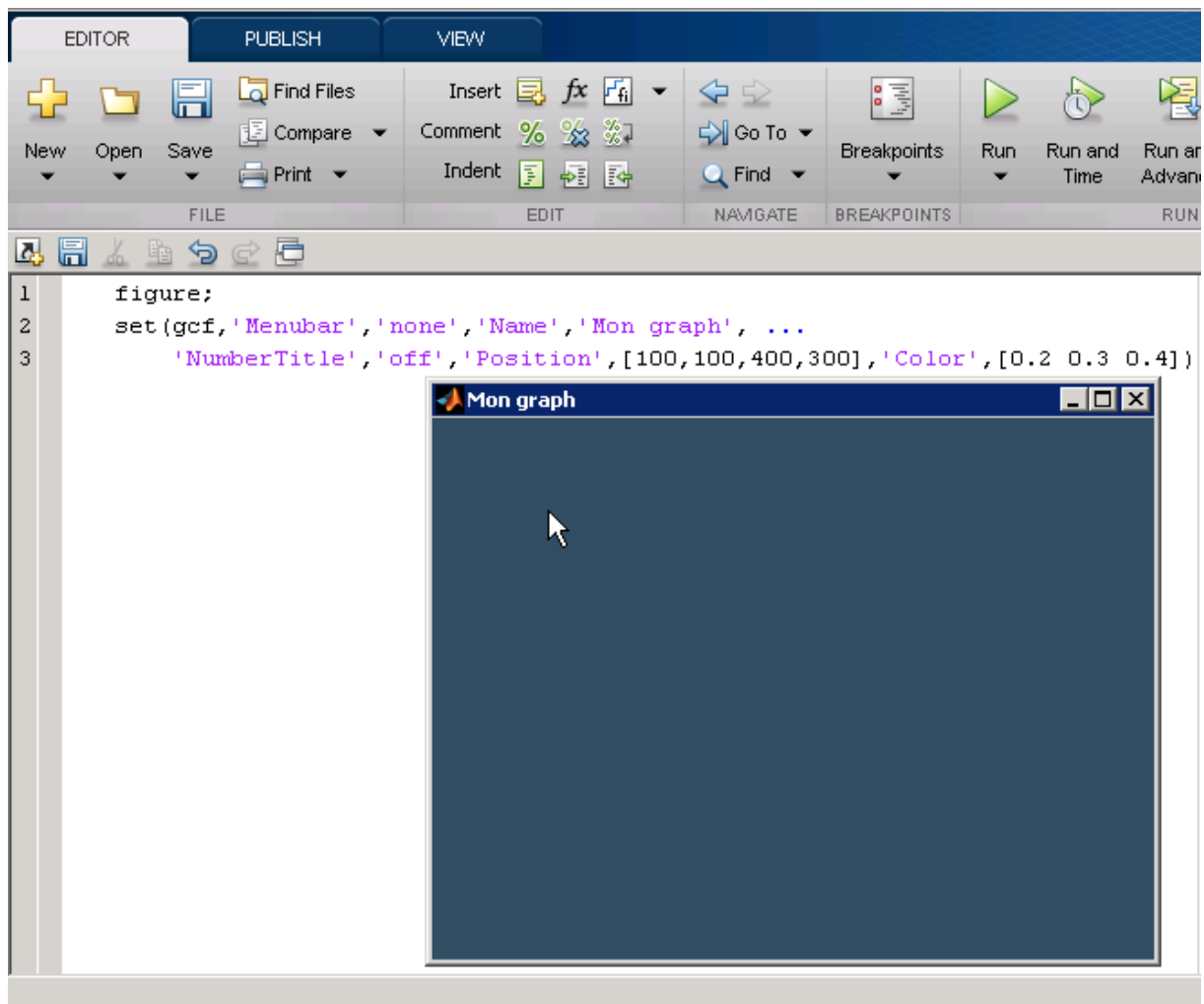
By defining some options on the active figure (**gcf**) you can remove the *Menu bar* of the figure:



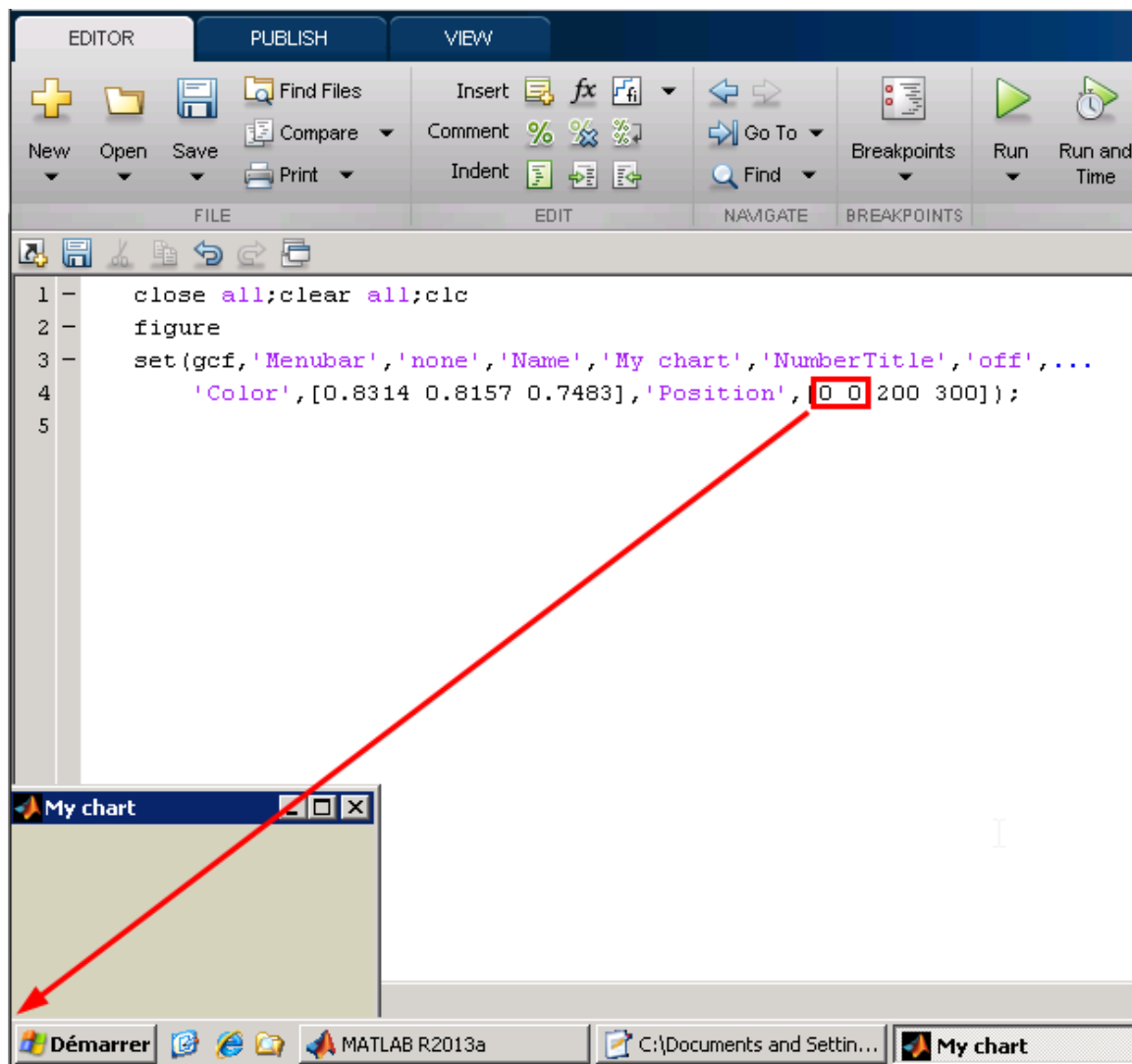
You can also with the same object properties easily define the name in the title bar of the figure:



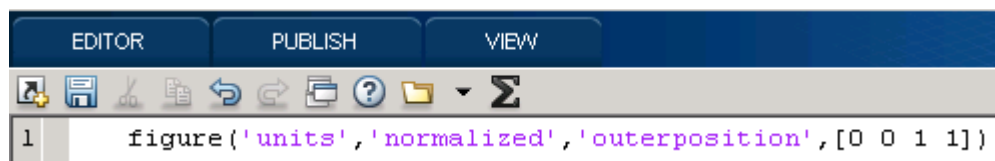
Or also deactivate the numbering of the figure that was visible on the title bar, change it's default **Position** (the 0,0 position is in the bottom left corner!) and it's default background **Color**:



And as you can see it here as a proof, the 0,0 position is in the bottom left corner:



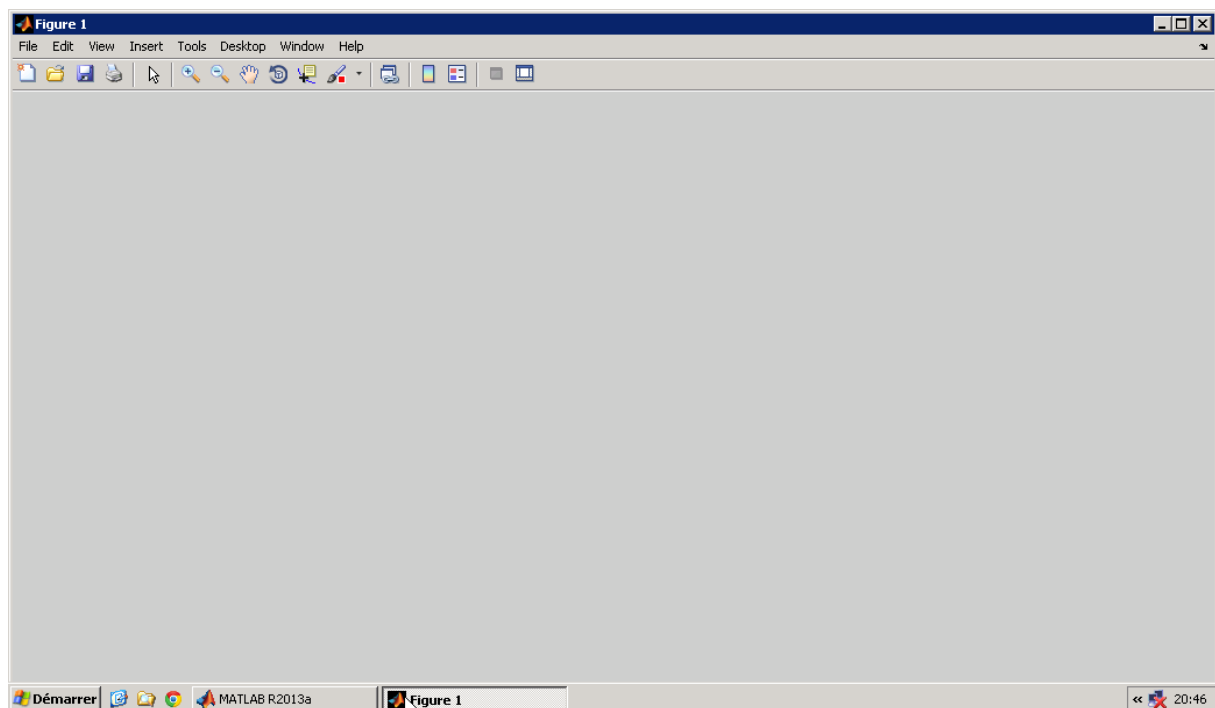
We can also make a full screen chart window using the following syntax:



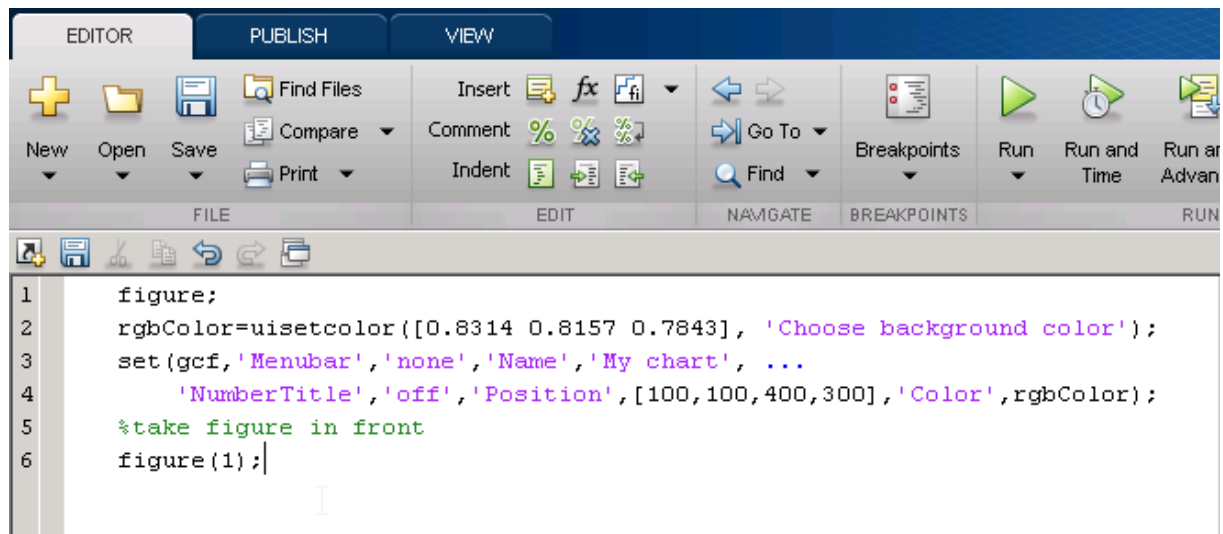
```
1 figure('units','normalized','outerposition',[0 0 1 1])
```

The image shows a snippet of the MATLAB Editor interface. At the top, there are three tabs: 'EDITOR', 'PUBLISH', and 'VIEW'. Below the tabs is a toolbar with various icons for file operations, editing, and execution. The code editor area shows a single line of MATLAB code: `figure('units','normalized','outerposition',[0 0 1 1])`. The line number '1' is visible on the left margin.

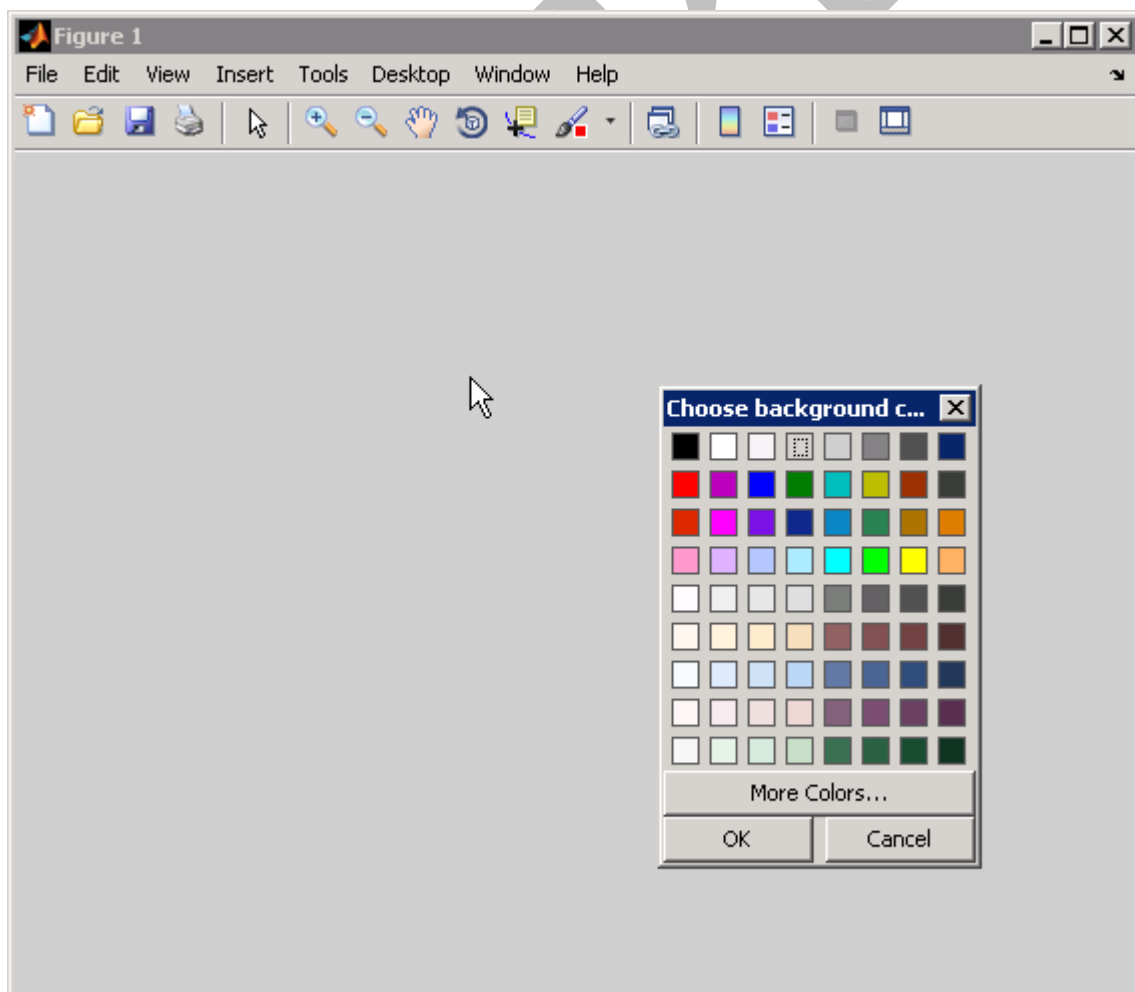
Then we get:



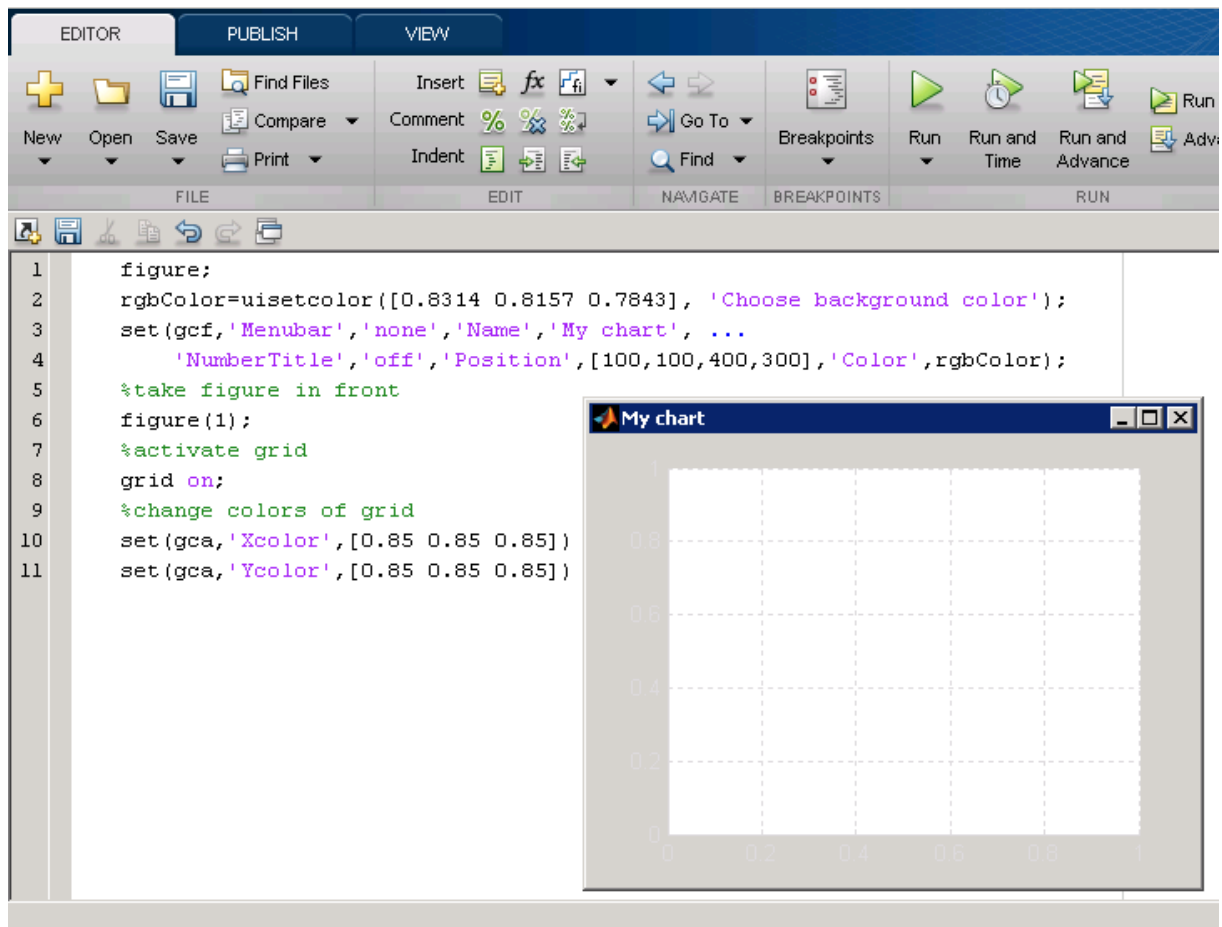
Using what we learned about User Interactions (see page 184), we can also ask the user to choose any color with the **uiscolor** function (in this case with a recommended color activated by default):



When you run the script the result will look like this:

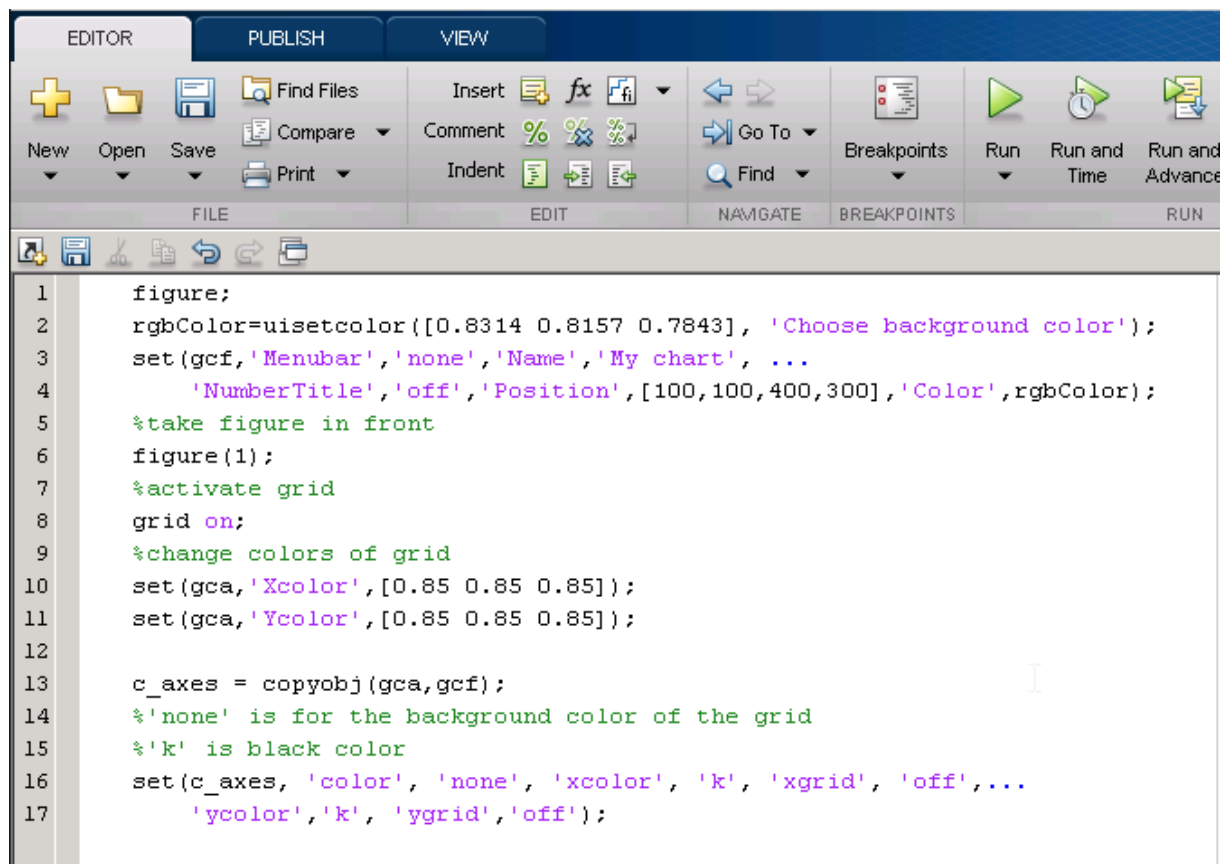


Now let us put a grid, using the command **grid on** and at the same time changing the grid color thanks to the object **gca**:



The problem is that when we customize the grid, we also change the axes ticks and labels in the same color... thus see next page what we have to add:

Here is what we have to add to recover back the axes in a more common color...:



```

1  figure;
2  rgbColor=uisetcolor([0.8314 0.8157 0.7843], 'Choose background color');
3  set(gcf,'MenuBar','none','Name','My chart', ...
4      'NumberTitle','off','Position',[100,100,400,300],'Color',rgbColor);
5  %take figure in front
6  figure(1);
7  %activate grid
8  grid on;
9  %change colors of grid
10 set(gca,'Xcolor',[0.85 0.85 0.85]);
11 set(gca,'Ycolor',[0.85 0.85 0.85]);
12
13 c_axes = copyobj(gca,gcf);
14 %'none' is for the background color of the grid
15 %'k' is black color
16 set(c_axes, 'color', 'none', 'xcolor', 'k', 'xgrid', 'off',...
17     'ycolor', 'k', 'ygrid', 'off');

```

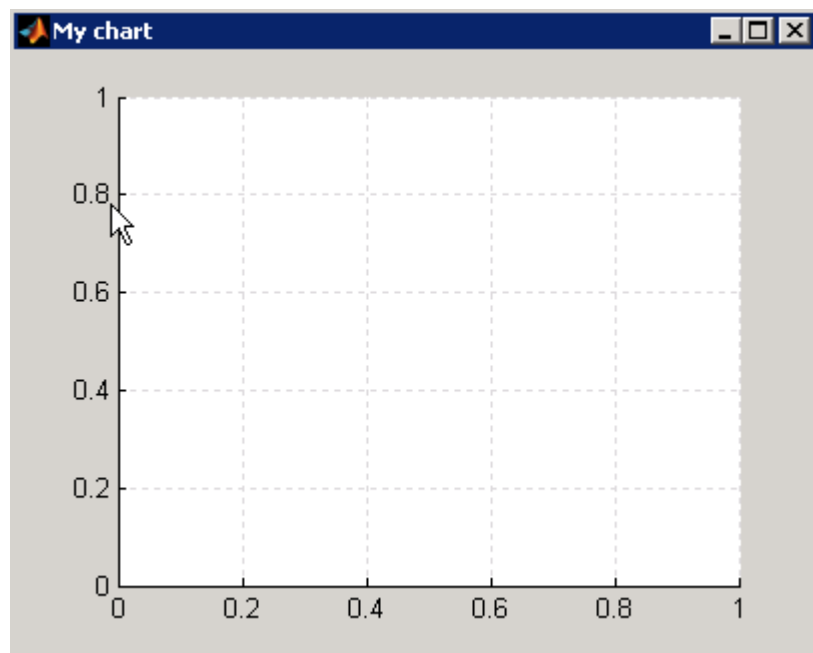
Not very intuitive isn't it? The result is visible on the next page.

For the colors here is a small table as refresh:

RGB Value	Short Name	Long Name
[1 1 0]	y	yellow
[1 0 1]	m	magenta
[0 1 1]	c	cyan
[1 0 0]	r	red
[0 1 0]	g	green
[0 0 1]	b	blue
[1 1 1]	w	white
[0 0 0]	k	black

Table 1 Standardized color codes

Here is the result:



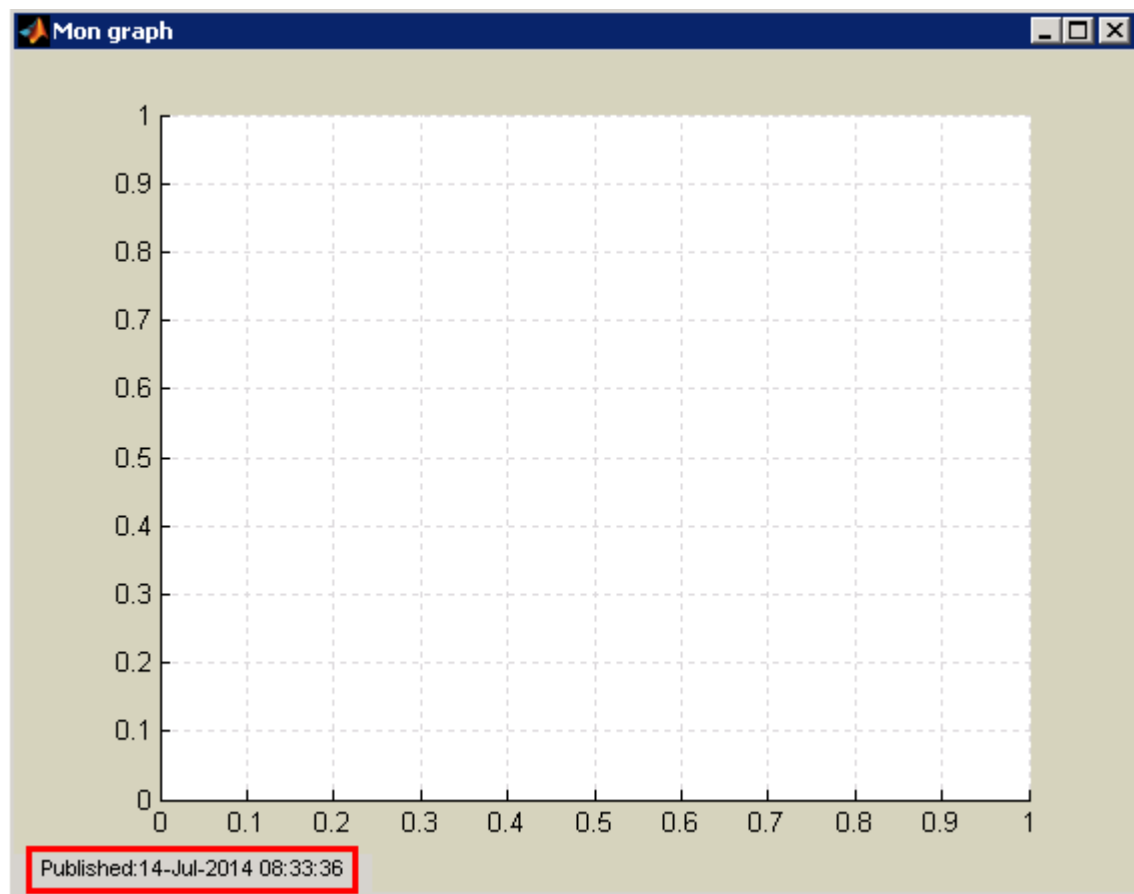
Now a very common request by engineers is to put like in Minitab the actual date and time (or any other date) in the corner of the chart figure. We can do this using a text type **uicontrol**:

```

EDITOR   PUBLISH   VIEW
1 - close all;clear all;clc
2 - figure
3 - set(gcf,'Menubar','none','Name','Mon graph','NumberTitle','off',...
4 -     'Color',[0.8314 0.8157 0.7483]);
5
6 - grid on;
7
8 - set(gca,'Xcolor',[0.85 0.85 0.85]);
9 - set(gca,'Ycolor',[0.85 0.85 0.85]);
10 - setAxes=copyobj(gca,gcf);
11 - set(setAxes,'color','none','Xcolor','k','xgrid','off','Ycolor','k','ygrid','off');
12 - MyBox=uicontrol('style','text');
13 - set(MyBox,'String',strcat('Published: ',datestr(clock)),'Position',[0, 0, 180, 20])
14

```

The result will look like the following:



Pick-up points on chart

We have removed all the menus and toolbar of our chart figure then a very common question is how can we easily and efficiently for post analysis make that the user can click on the chart an retrieve coordinates in the workspace.

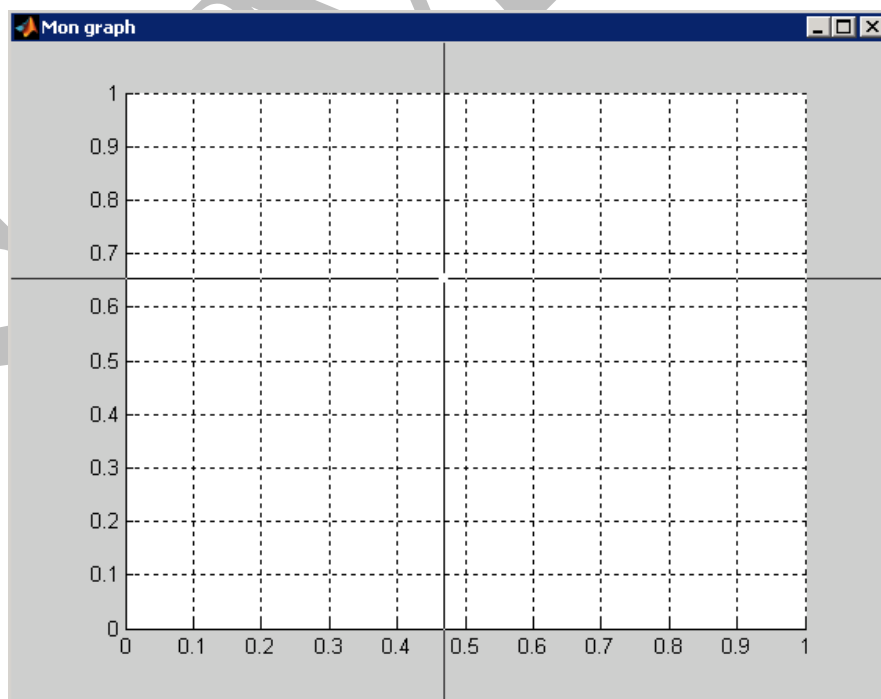
For this purpose you can use the **ginput()** function by specifying the number of points to retrieve in the workspace:

```

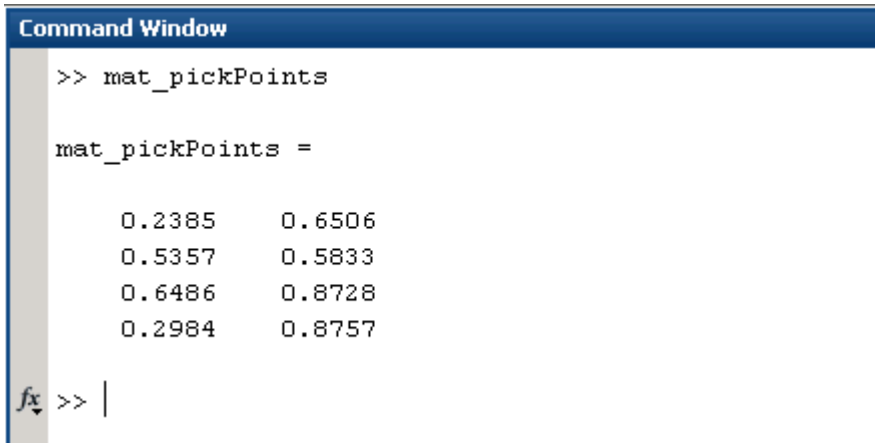
1  figure
2  set(gcf,'Menubar','none','Name','Mon graph','NumberTitle','off',...
3      'Color',[0.8 0.8 0.8])
4
5  grid on;
6  %get 4 points
7  mat_pickPoints=ginput(4);
8
9  set(gca,'Xcolor',[0.85 0.85 0.85])
10 set(gca,'Ycolor',[0.85 0.85 0.85])
11 c_axes=copyobj(gca,gcf);
12 set(c_axes,'color','none','xcolor','k','xgrid','off','ycolor','k','ygrid','off');

```

This will result in a sniper target tool to pick up the coordinates as you can see below on this empty chart:



And you will get in the workspace a new matrix variable with all coordinates that you can use for multiple purposes:



```
Command Window

>> mat_pickPoints

mat_pickPoints =

    0.2385    0.6506
    0.5357    0.5833
    0.6486    0.8728
    0.2984    0.8757

fx >> |
```

A screenshot of the MATLAB Command Window. The title bar is blue and says "Command Window". The window shows the command `>> mat_pickPoints` being entered. Below the command, the output is displayed: `mat_pickPoints =` followed by a 4x2 matrix of values: 0.2385, 0.6506; 0.5357, 0.5833; 0.6486, 0.8728; 0.2984, 0.8757. At the bottom, the prompt `fx >> |` is visible with a cursor.

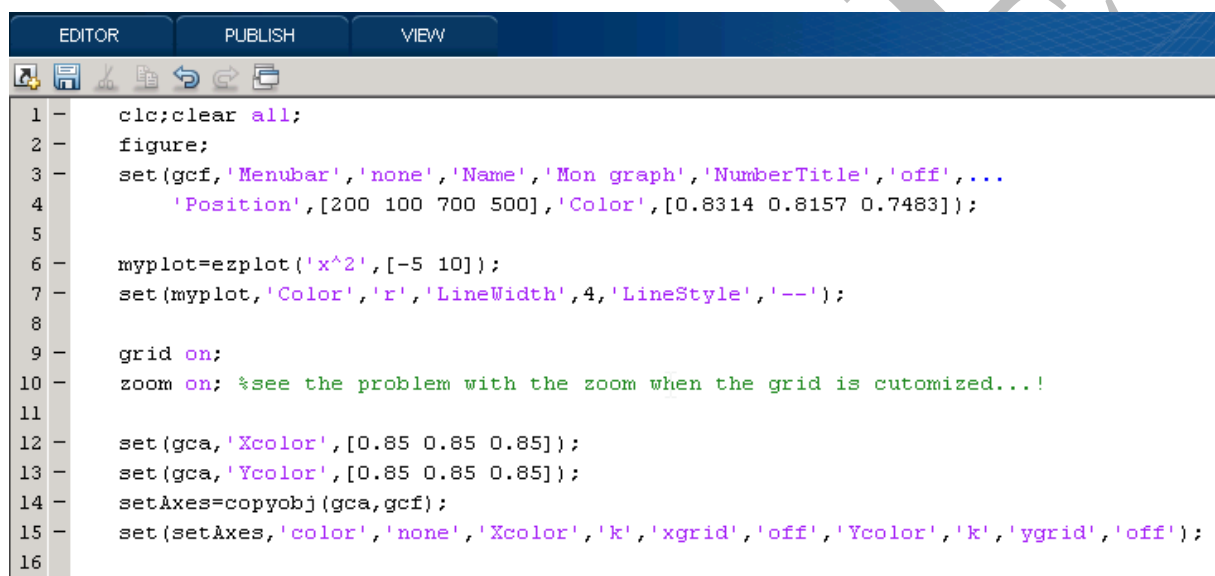
C to $\mathbb{R}^2$ plots

OK now we begin a big chapter important for students but also for professional practitioners of MATLAB.

The purpose of this chapter is not only to see how to make $\mathbb{C} \rightarrow \mathbb{R}^2$ but also to give a lot of tips and tricks relatively to the numerous questions of people following my trainings.

Analytical based plot

We start with a very academic example first where many student know the expression of a function a have to simple plot it by typing its expression. You can do this quite easily with the function **ezplot()**:

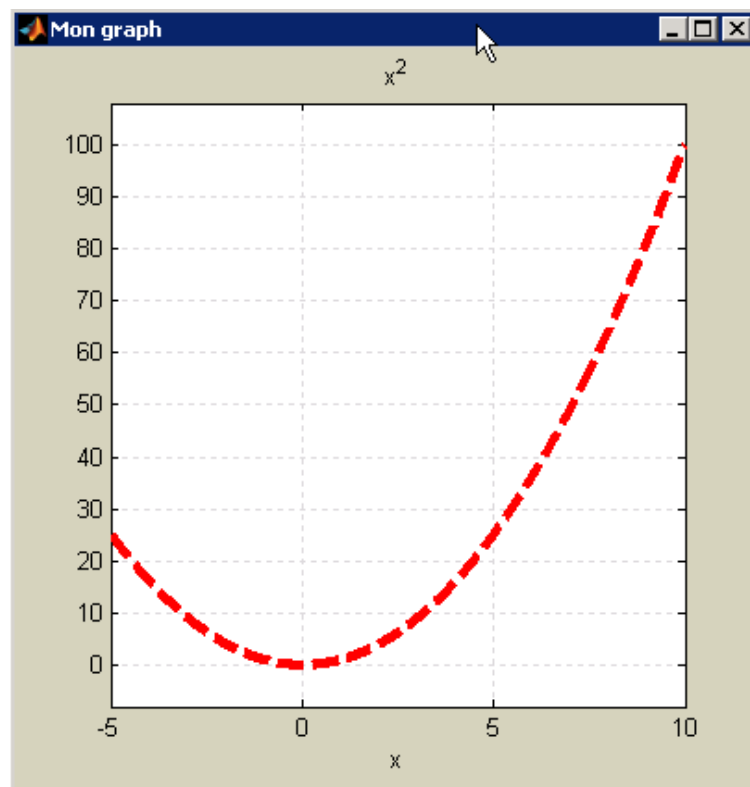


```

1 -   clc;clear all;
2 -   figure;
3 -   set(gcf,'MenuBar','none','Name','Mon graph','NumberTitle','off',...
4 -       'Position',[200 100 700 500],'Color',[0.8314 0.8157 0.7483]);
5
6 -   myplot=ezplot('x^2',[-5 10]);
7 -   set(myplot,'Color','r','LineWidth',4,'LineStyle','--');
8
9 -   grid on;
10 -  zoom on; %see the problem with the zoom when the grid is customized...!
11
12 -  set(gca,'Xcolor',[0.85 0.85 0.85]);
13 -  set(gca,'Ycolor',[0.85 0.85 0.85]);
14 -  setAxes=copyobj(gca,gcf);
15 -  set(setAxes,'color','none','Xcolor','k','xgrid','off','Ycolor','k','ygrid','off');
16

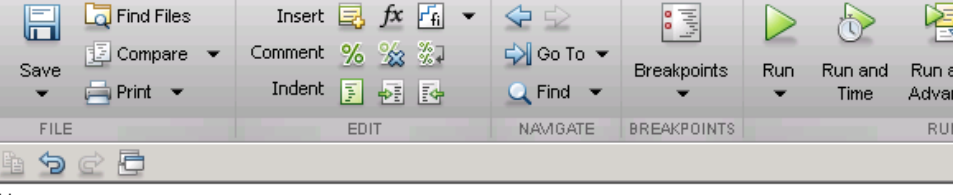
```

The result will be the following (see the title that is added automatically):



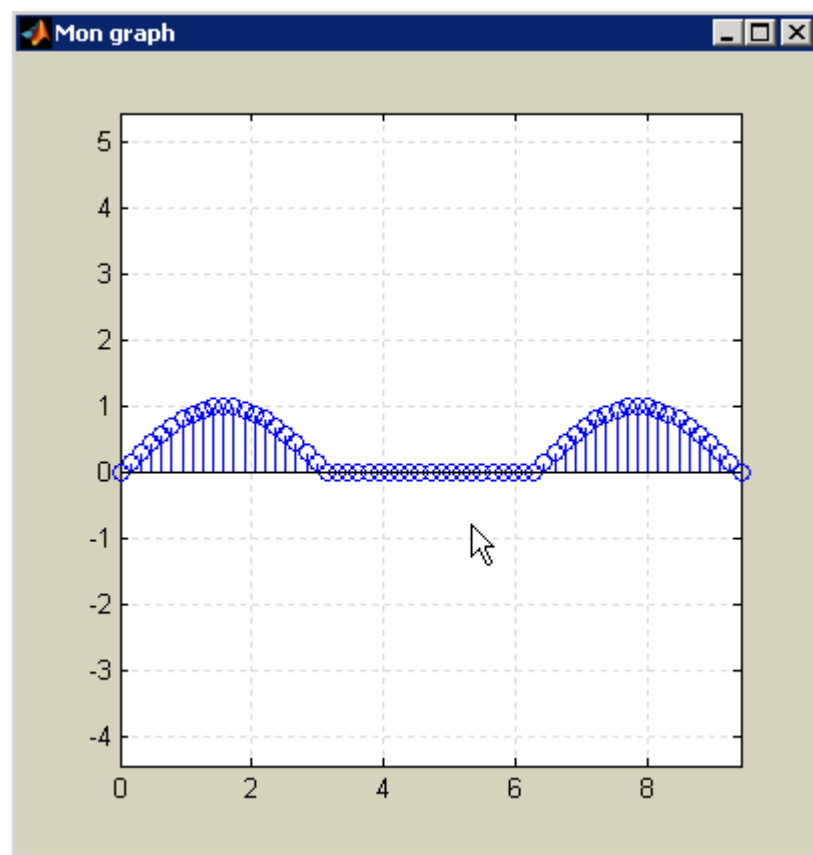
Vector based plot with lines and stems

Sometimes and especially in signal processing you can't use analytical expression to plot something. This is what I call "vector based plots". We can make a typical example of this with a simple trigonometric function, a threshold filter condition and using the **stem()** plot function that is very common in signal processing and see how the chart will look like by adding or not the line **axis equal**:



```
1 figure;
2 set(gcf, 'MenuBar', 'none', 'Name', 'Mon graph', 'NumberTitle', 'off', ...
3     'Position', [100 100 400 400], 'Color', [0.8314 0.8157 0.7483]);
4 x = 0 : pi/20 : 3 * pi;
5 y = sin(x);
6 y = y .* (y > 0); %set negative values of sin(x) to zero
7 plot(x, y);
8 stem(x,y); %stem is used a lot for FFT
9 grid on;
10 axis equal;
11 set(gca, 'Xcolor', [0.85 0.85 0.85]);
12 set(gca, 'Ycolor', [0.85 0.85 0.85]);
13 setAxes=copypobj(gca,gcf);
14 set(setAxes, 'color', 'none', 'Xcolor', 'k', 'xgrid', 'off', 'Ycolor', 'k', 'ygrid', 'off');
```

The result will be:



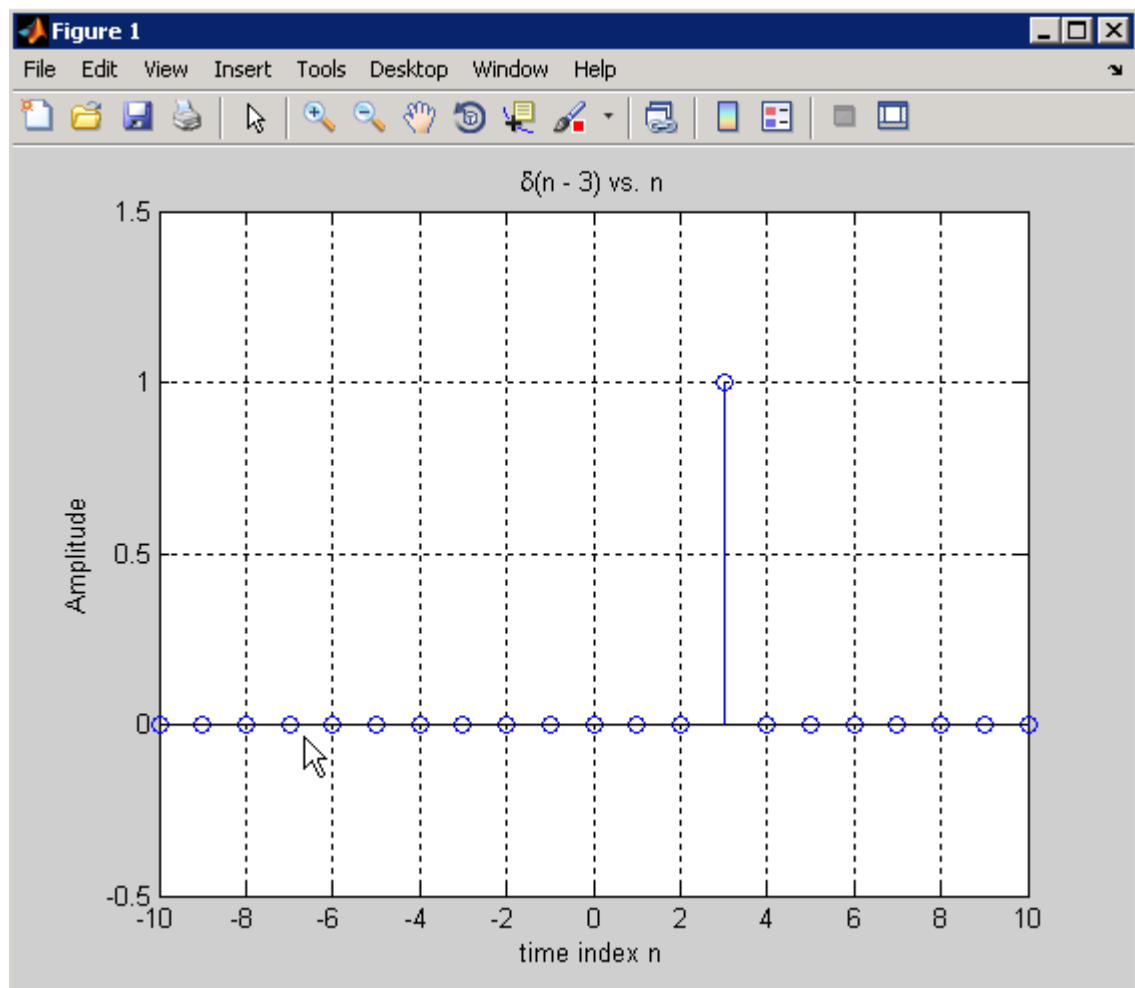
Or a more concrete example:

```

EDITOR    PUBLISH    VIEW
[Icons]
1      n = -10:1:10;
2      k = [(n - 3) == 0];
3      stem(n,k)
4      title('\delta(n - 3) vs. n')
5      xlabel('time index n')
6      ylabel('Amplitude')
7      a = min(n);
8      b = max(n);
9      c = min(k) - .5;
10     d = max(k) + .5;
11     axis([a b c d]); grid on;

```

That gives and Dirac delta on $n = 3$:



Animated vector based plot

A lot of schools and some financial institutions ask me sometimes how to animate a 2D plots (in financial institutions they like this to animate forecasts...).

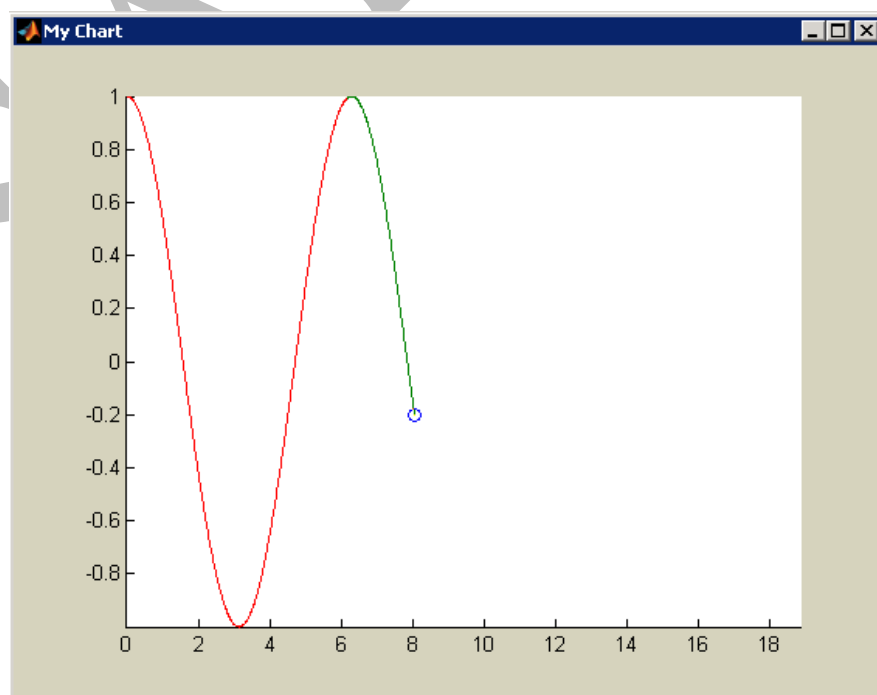
Let us see a first method (default native poor one...) using the command **comet()** and solving at the same time the fact that the plot disappear one the animation is finished:

```

1  -  clc;clear all;
2  -  figure;
3  -  set(gcf,'Menubar','none','Name','Mon graph','NumberTitle','off',...
4  -      'Position',[200 100 700 500],'Color',[0.8314 0.8157 0.7483]);
5
6  -  grid on;
7  -  t=0:0.01:6*pi;
8  -  x=t;y=cos(t);
9  -  comet(x,y); %the problem is that the line disappears at the end...
10
11  -  %then we need after a normal plot
12  -  plot(x,y,'Color','r')
13  -  xlim([0 19]); %otherwise it doesn't correspond to comet axis... (stupid behaviour!)
14  -  ylabel('y');
15  -  xlabel('t');
16  -  grid on;
17  -  title('Animated 2D plot: cos(t)','fontweigh','bold','fontsize',16);
18
19  -  set(gca,'Xcolor',[0.85 0.85 0.85]);
20  -  set(gca,'Ycolor',[0.85 0.85 0.85]);
21  -  setAxes=copyobj(gca,gcf);
22  -  set(setAxes,'color','none','Xcolor','k','xgrid','off','Ycolor','k','ygrid','off');
23

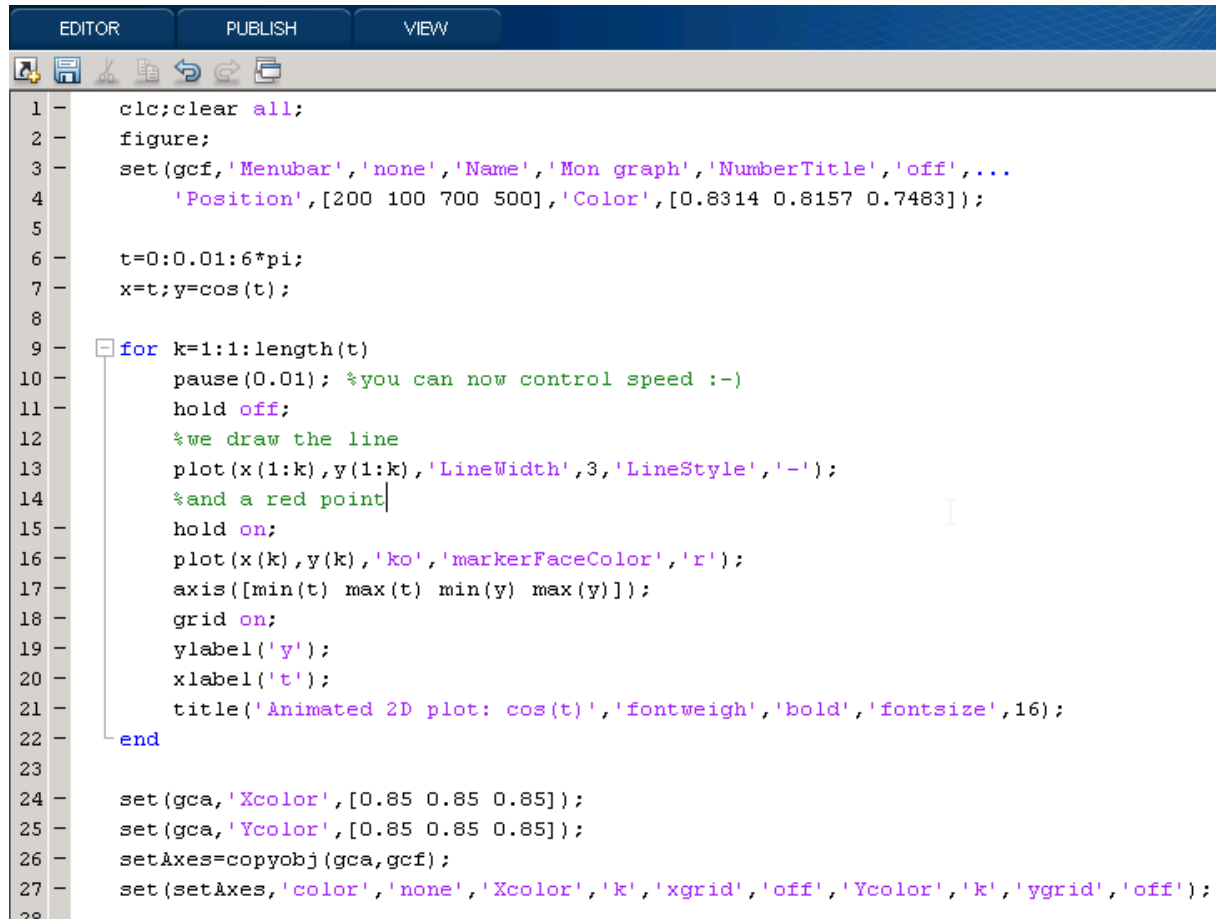
```

The result will looks like:



To control speed you can edit the code of the *comet* function by adding a *pause(2)* for example at the right place but then it will work only on your computer. The best way is the following:

Second method using a loop with the functions **pause()** and **plot()** to control the speed of the animation:

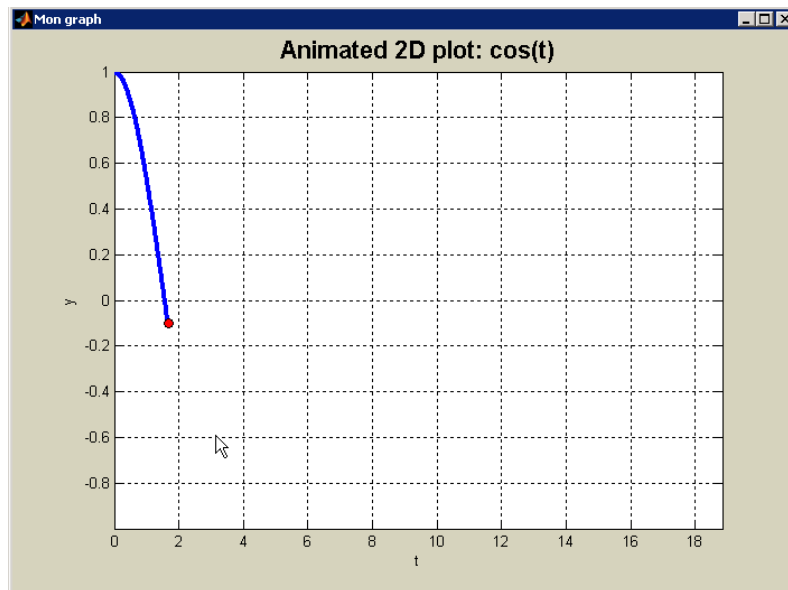


```

1 -   clc;clear all;
2 -   figure;
3 -   set(gcf,'Menubar','none','Name','Mon graph','NumberTitle','off',...
4 -       'Position',[200 100 700 500],'Color',[0.8314 0.8157 0.7483]);
5 -
6 -   t=0:0.01:6*pi;
7 -   x=t;y=cos(t);
8 -
9 -   for k=1:length(t)
10 -       pause(0.01); %you can now control speed :-)
11 -       hold off;
12 -       %we draw the line
13 -       plot(x(1:k),y(1:k),'LineWidth',3,'LineStyle','-');
14 -       %and a red point
15 -       hold on;
16 -       plot(x(k),y(k),'ko','markerFaceColor','r');
17 -       axis([min(t) max(t) min(y) max(y)]);
18 -       grid on;
19 -       ylabel('y');
20 -       xlabel('t');
21 -       title('Animated 2D plot: cos(t)','fontweigh','bold','fontsize',16);
22 -   end
23 -
24 -   set(gca,'Xcolor',[0.85 0.85 0.85]);
25 -   set(gca,'Ycolor',[0.85 0.85 0.85]);
26 -   setAxes=copyobj(gca,gcf);
27 -   set(setAxes,'color','none','Xcolor','k','xgrid','off','Ycolor','k','ygrid','off');
28 -

```

The result will looks like this:



Vector based plot with various lines, colors and points

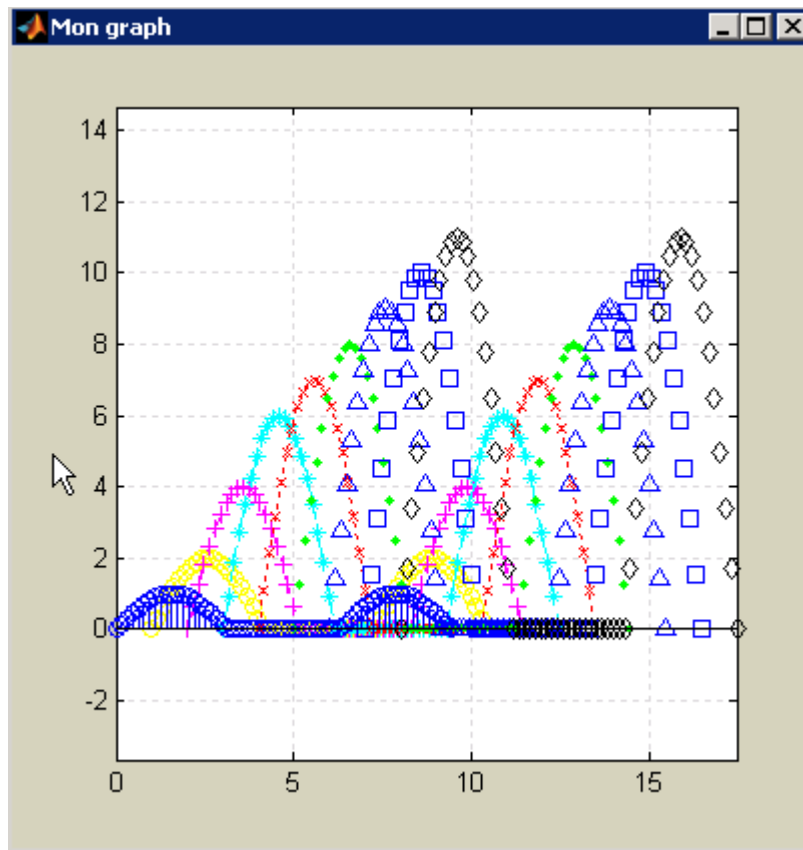
Depending on the subject you are studying you will need to have multiple lines plots into only one plot area. Then it is important to have an overview of some possible line plots styles.

Here we plot a vector based function but now using the function **plot()** and we added also a **stem()** for fun:

```

1 - figure;
2 - title('Vector based plot');
3 - set(gcf, 'Menubar', 'none', 'Name', 'Mon graph', 'NumberTitle', 'off', ...
4 -     'Position', [100 100 400 400], 'Color', [0.8314 0.8157 0.7483]);
5 - x = 0 : pi/20 : 3 * pi;
6 - y = sin(x);
7 - y = y .* (y > 0); %set negative values of sin(x) to zero
8 - plot(x, y);
9 - hold on;
10 - plot(x+1, 2*y, '-yo');
11 - plot(x+2, 4*y, '-.m+');
12 - plot(x+3, 6*y, '--c*');
13 - plot(x+4, 7*y, 'rx');
14 - plot(x+5, 8*y, '.g');
15 - plot(x+6, 9*y, '^b');
16 - plot(x+7, 10*y, 'sb');
17 - plot(x+8, 11*y, 'kd');
18 - stem(x, y); %stem is used a lot for FFT
19 - grid on;
20 - axis equal;
21 - set(gca, 'Xcolor', [0.85 0.85 0.85]);
22 - set(gca, 'Ycolor', [0.85 0.85 0.85]);
23 - set&axes=copyobj(gca, gcf);
24 - set(set&axes, 'color', 'none', 'Xcolor', 'k', 'xgrid', 'off', 'Ycolor', 'k', 'ygrid', 'off');

```



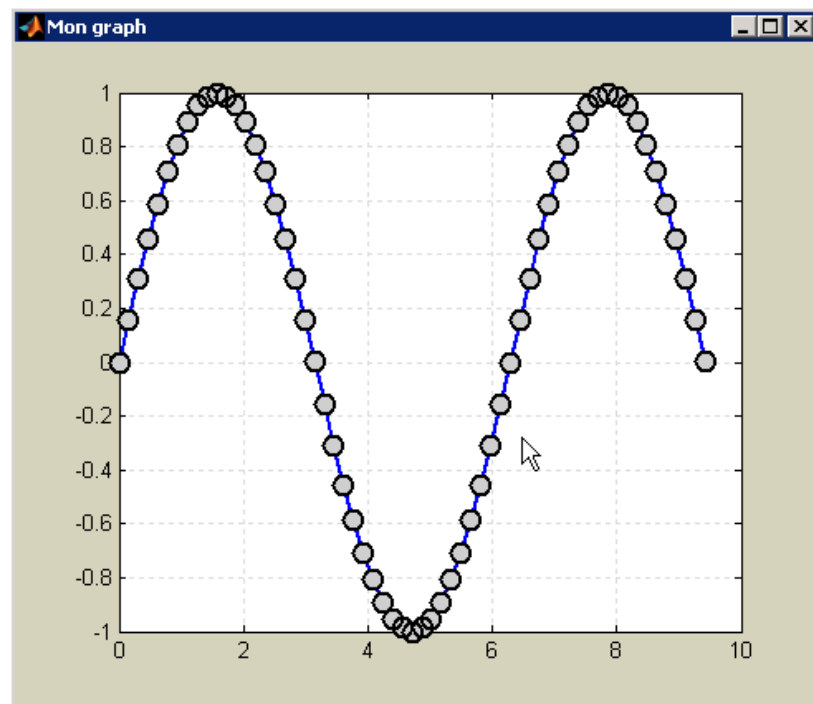
And for line width and some other extras we play with the **plot()** function arguments:

```

EDITOR PUBLISH VIEW
1 - figure;
2 - set(gcf,'MenuBar','none','Name','Mon graph','NumberTitle','off',...
3 -     'Position',[100 100 400 400],'Color',[0.8314 0.8157 0.7483]);
4 -
5 - x=0:pi/20:3*pi;
6 - y=sin(x);
7 - plot(x,y,'-bo','LineWidth',2,'MarkerSize',10,'MarkerEdgeColor','k',...
8 -     'MarkerFaceColor',[0.8 0.8 0.8])
9 -
10 - grid on;
11 - set(gca,'Xcolor',[0.85 0.85 0.85]);
12 - set(gca,'Ycolor',[0.85 0.85 0.85]);
13 - setAxes=copyobj(gca,gcf);
14 - set(setAxes,'color','none','Xcolor','k','xgrid','off','Ycolor','k','ygrid','off');

```

See the result next page.



Vector based multiple plot with legends and titles

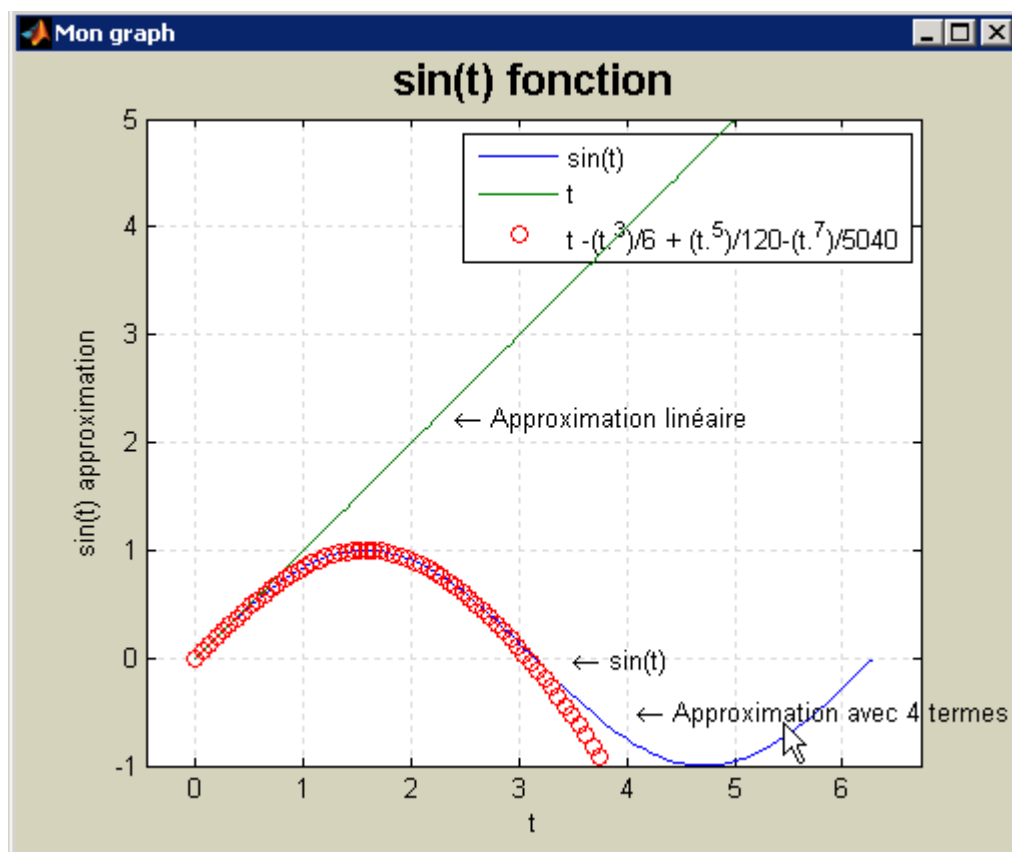
Instead than typing multiple times `plot()`, `plot()`, `plot()` etc. you can directly in only one `plot()` function pass multiple plot vectors. At the same time the purpose of this example is to show add axes labels with `xlabel()` and `ylabel()`, plot title with `title()`, plot legends with `legend()` and static text with `text()` and dynamic text with `gtext()` on the plot:

```

1 -   clc; clear all
2 -   figure;
3 -   title('Vector based plot');
4 -   set(gcf,'Menubar','none','Name','Mon graph','NumberTitle','off',...
5 -       'Position',[100 100 500 400],'Color',[0.8314 0.8157 0.7483]);
6 -   t = linspace(0, 2*pi, 100);
7 -   y1 = sin(t);
8 -   y2 = t;
9 -   y3 = t - (t.^3)/6 + (t.^5)/120 - (t.^7)/5040;
10 -  plot(t, y1, t, y2, '-', t, y3, 'o');
11 -  grid on;
12 -  axis([0 5 -1 5]);
13 -  xlabel('t');
14 -  ylabel('sin(t) approximation');
15 -  title('sin(t) function','fontweight','bold','fontsize',16);
16 -  text(3.5,0, '\leftarrow sin(t)');
17 -  legend('sin(t)', 't', 't - (t.^3)/6 + (t.^5)/120 - (t.^7)/5040');
18 -  gtext('\leftarrow Approximation linéaire');
19 -  gtext('\leftarrow Approximation avec 4 termes');
20 -  axis equal;
21 -  set(gca,'Xcolor',[0.85 0.85 0.85]);
22 -  set(gca,'Ycolor',[0.85 0.85 0.85]);
23 -  setAxes=copyobj(gca,gcf);
24 -  set(setAxes,'color','none','Xcolor','k','xgrid','off','Ycolor','k','ygrid','off');

```

This will result in:



Thanks to **ginput()** the texts will be placed where the user has click on the chart.

Horizontal/Vertical lines method

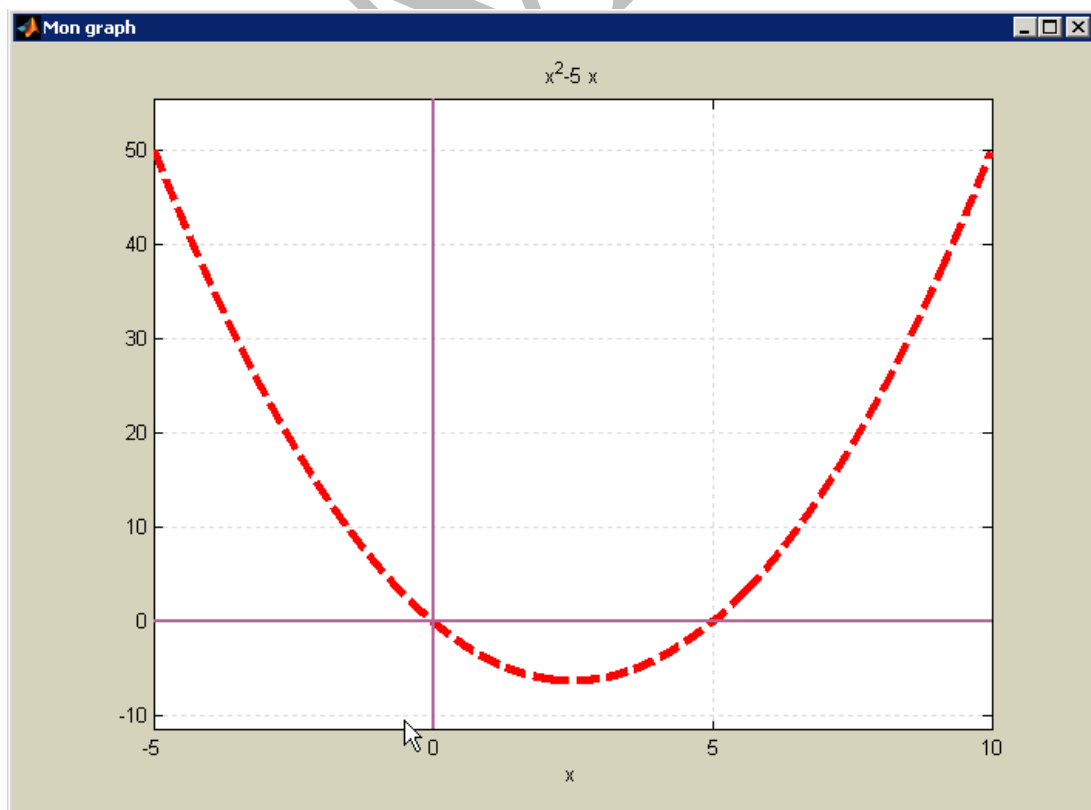
A very very common question to almost all jobs specializations is how to add a vertical/horizontal line on chart. This can easily be done thanks to the **graph2d.constantline()** function:

```

1  -  clc;clear all;
2  -  figure;
3  -  set(gcf,'Menubar','none','Name','Mon graph','NumberTitle','off',...
4  -      'Position',[200 100 700 500],'Color',[0.8314 0.8157 0.7483]);
5
6  -  myplot=ezplot('x^2-5*x',[-5 10]);
7  -  set(myplot,'Color','r','LineWidth',4,'LineStyle','--');
8  -  grid on;
9
10 -  %vertical line
11 -  hx=graph2d.constantline(0,'Color',[0.7 0.4 0.6],'LineWidth',2);
12 -  changedependvar(hx,'x');
13 -  %horizontal line
14 -  hy=graph2d.constantline(0,'Color',[0.7 0.4 0.6],'LineWidth',2);
15 -  changedependvar(hy,'y');
16
17 -  set(gca,'Xcolor',[0.85 0.85 0.85]);
18 -  set(gca,'Ycolor',[0.85 0.85 0.85]);
19 -  setAxes=copyobj(gca,gcf);
20 -  set(setAxes,'color','none','Xcolor','k','xgrid','off','Ycolor','k','ygrid','off');
21

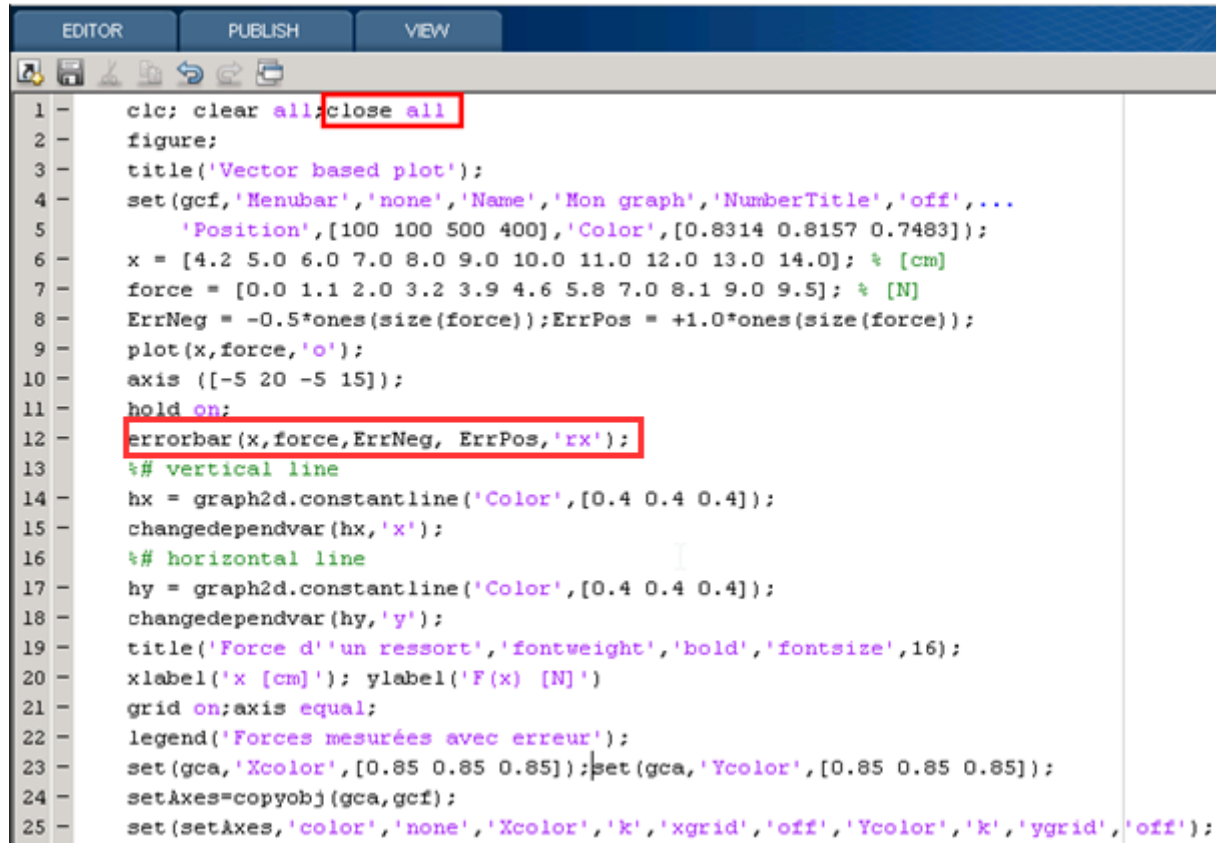
```

The result will look as following:



Vector based plot with error bars

Ok now see how we do a dot plot with error bars in MATLAB using the function **errorbar()**. At the same time see that at the beginning we close all open charts to be sure that our will be the only one opened in this MATLAB sessions:

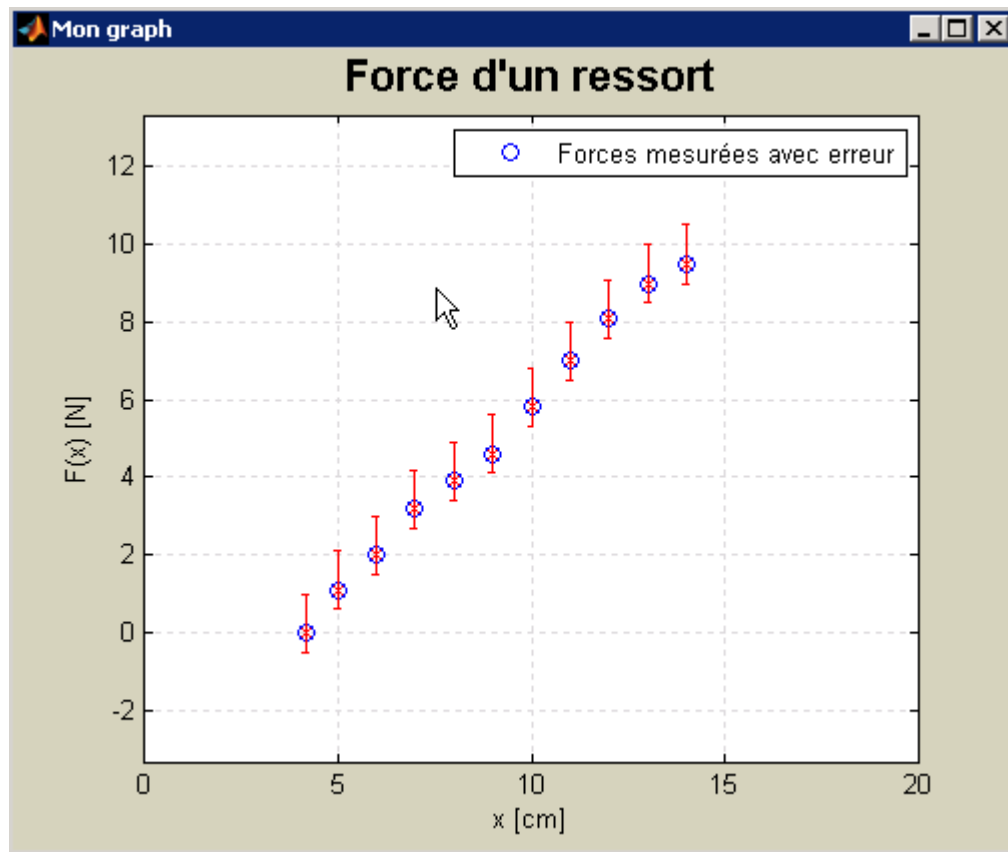


```

1 - clc; clear all; close all;
2 - figure;
3 - title('Vector based plot');
4 - set(gcf,'Menubar','none','Name','Mon graph','NumberTitle','off',...
5 -     'Position',[100 100 500 400],'Color',[0.8314 0.8157 0.7483]);
6 - x = [4.2 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 13.0 14.0]; % [cm]
7 - force = [0.0 1.1 2.0 3.2 3.9 4.6 5.8 7.0 8.1 9.0 9.5]; % [N]
8 - ErrNeg = -0.5*ones(size(force)); ErrPos = +1.0*ones(size(force));
9 - plot(x,force,'o');
10 - axis ([-5 20 -5 15]);
11 - hold on;
12 - errorbar(x,force,ErrNeg, ErrPos,'rx');
13 - %# vertical line
14 - hx = graph2d.constantline('Color',[0.4 0.4 0.4]);
15 - changedependvar(hx,'x');
16 - %# horizontal line
17 - hy = graph2d.constantline('Color',[0.4 0.4 0.4]);
18 - changedependvar(hy,'y');
19 - title('Force d'un ressort','fontweight','bold','fontsize',16);
20 - xlabel('x [cm]'); ylabel('F(x) [N]')
21 - grid on; axis equal;
22 - legend('Forces mesurées avec erreur');
23 - set(gca,'Xcolor',[0.85 0.85 0.85]); set(gca,'Ycolor',[0.85 0.85 0.85]);
24 - setAxes=copyobj(gca,gcf);
25 - set(setAxes,'color','none','Xcolor','k','xgrid','off','Ycolor','k','ygrid','off');

```

See the result on next page.



Vector based semi-log X plot with custom tick space

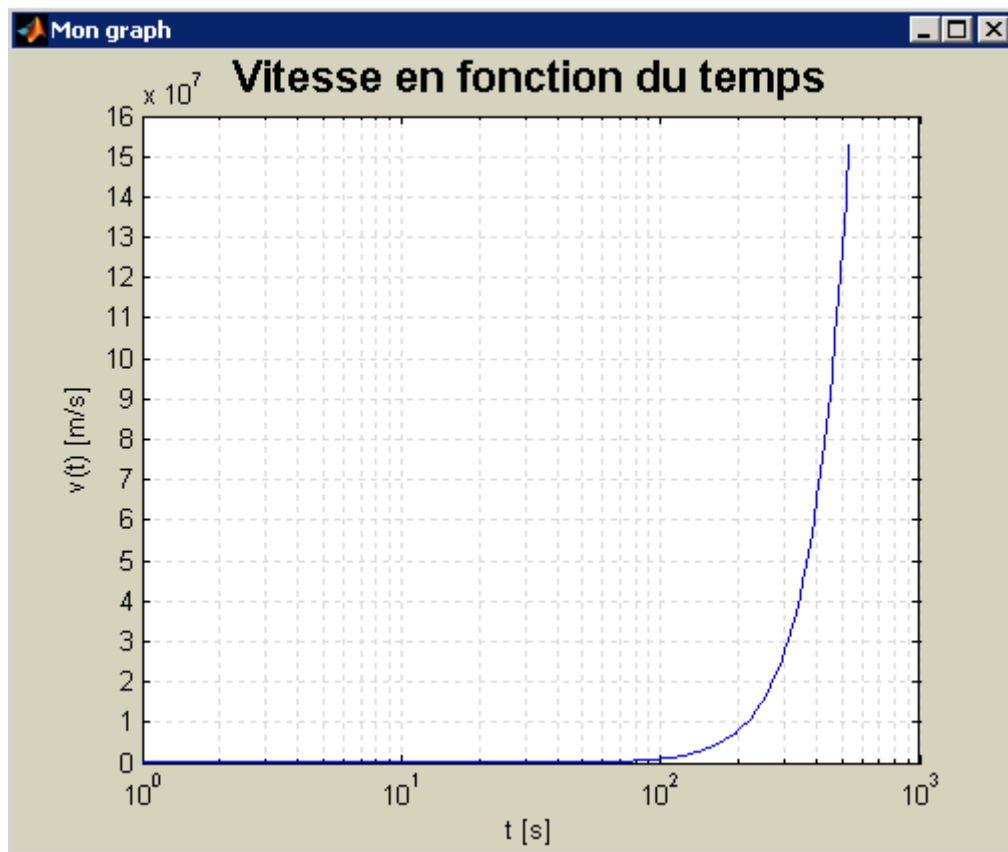
Now see the famous semi-log plots used a lot by physicist and electronic engineers using the the function **semilogx()** and see how we can control the axes Y tick space at the same time using the **YTick** property of the active chart figure:

```

1 -  clc; clear all; close all
2 -  figure;
3 -  title('Vector based plot');
4 -  set(gcf,'MenuBar','none','Name','Mon graph','NumberTitle','off',...
5 -      'Position',[100 100 500 400],'Color',[0.8314 0.8157 0.7483]);
6 -  t = linspace(0, 2*pi, 200);
7 -  x = exp(t);
8 -  y = 100 + exp(3*t);
9 -  semilogx(x, y);
10 - title('Vitesse en fonction du temps','fontweight','bold','fontsize',16);
11 - xlabel('t [s]'); ylabel('v(t) [m/s]')
12 - grid on;
13 - set(gca,'YTick',[0:100000000:2000000000])
14 - %axis equal; must be removed when using log scale!
15 - set(gca,'Xcolor',[0.85 0.85 0.85]); set(gca,'Ycolor',[0.85 0.85 0.85]);
16 - set(gca,'copyobj',gca,gcf);
17 - set(setaxes,'color','none','Xcolor','k','xgrid','off','Ycolor','k','ygrid','off');
18 -

```

This is how the result looks like:



Vector based semi-log Y plot

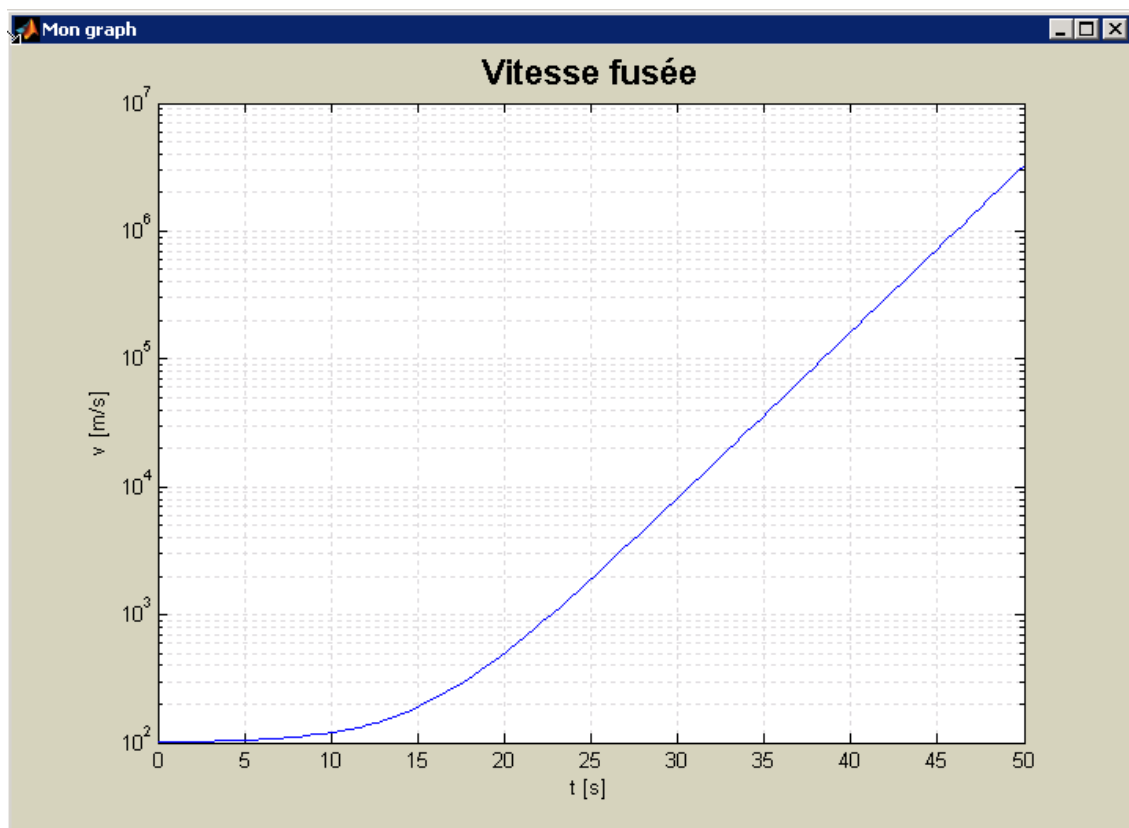
And in the same idea but using with the Y axes using the **semilogy()** function:

```

EDITOR PUBLISH VIEW
1 - clc;clear all;close all;
2 - figure;
3 - set(gcf,'Menubar','none','Name','Mon graph','NumberTitle','off',...
4 - 'Position',[200 -50 800 800],'Color',[0.8314 0.8157 0.7483]);
5
6 - t=linspace(0,50,100);
7 - v=100+exp(0.3*t);
8 - semilogy(t,v);
9
10 - xlabel('t [s]');
11 - ylabel('v [m/s]');
12 - title('Vitesse fusée','fontweight','bold','fontsize',16);
13
14 - grid on;
15 - set(gca,'Xcolor',[0.85 0.85 0.85]);
16 - set(gca,'Ycolor',[0.85 0.85 0.85]);
17 - setAxes=copyobj(gca,gcf);
18 - set(setAxes,'color','none','Xcolor','k','xgrid','off','Ycolor','k','ygrid','off');
19
20 - datacursormode on;

```

This is how the result looks like:



Vector based log-log plot with data cursor tip

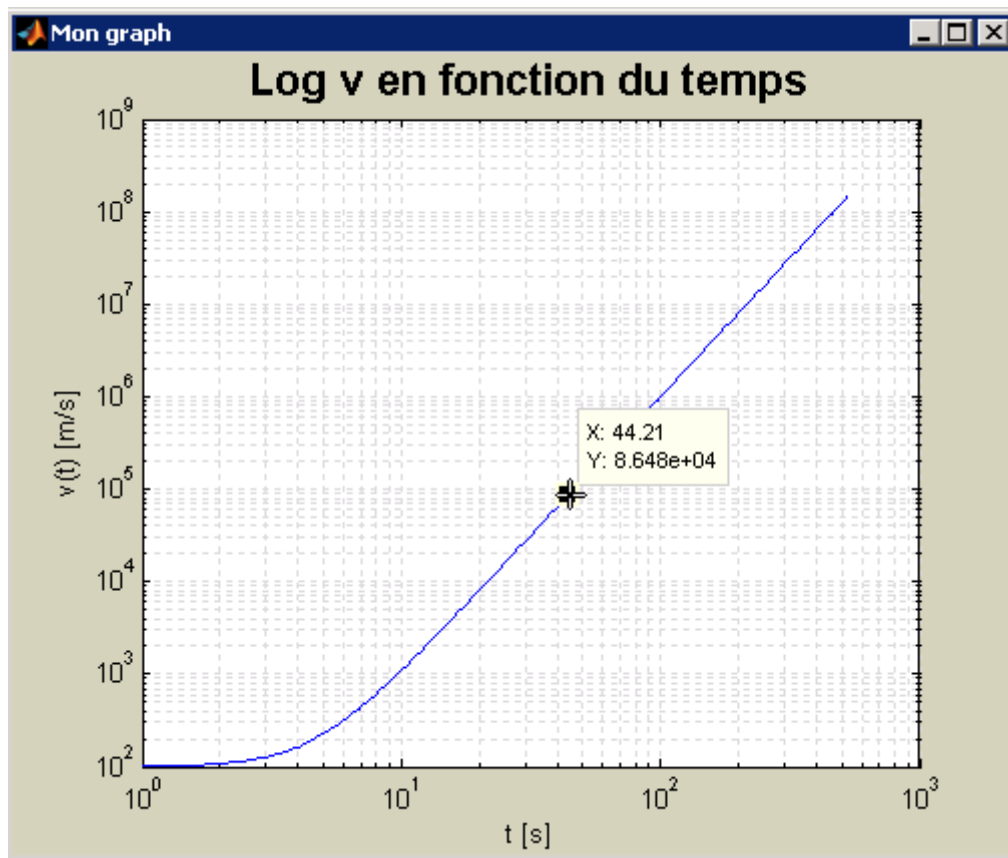
This case is rare... using a log-log plot thanks to the function `loglog()`. What is instead interesting in this chart is that we show here for the first time that you can give to your user/customer the possibility to play with the data cursor by using the command **`datacursormode on`**:

```

1 -   clc; clear all; close all
2 -   figure;
3 -   title('Vector based plot');
4 -   set(gcf, 'Menubar', 'none', 'Name', 'Mon graph', 'NumberTitle', 'off', ...
5 -         'Position', [100 100 500 400], 'Color', [0.8314 0.8157 0.7483]);
6 -   t = linspace(0, 2*pi, 200);
7 -   x = exp(t);
8 -   y = 100 + exp(3*t);
9 -   loglog(x, y);
10 -  title('Log v en fonction du temps', 'fontweight', 'bold', 'fontsize', 16);
11 -  xlabel('t [s]'); ylabel('v(t) [m/s]')
12 -  grid on;
13 -  %axis equal; must be removed when using log scale!
14 -  set(gca, 'Xcolor', [0.85 0.85 0.85]); set(gca, 'Ycolor', [0.85 0.85 0.85]);
15 -  setAxes=copyobj(gca, gcf);
16 -  set(setAxes, 'color', 'none', 'Xcolor', 'k', 'xgrid', 'off', 'Ycolor', 'k', 'ygrid', 'off');
17 -
18 -   datacursormode on

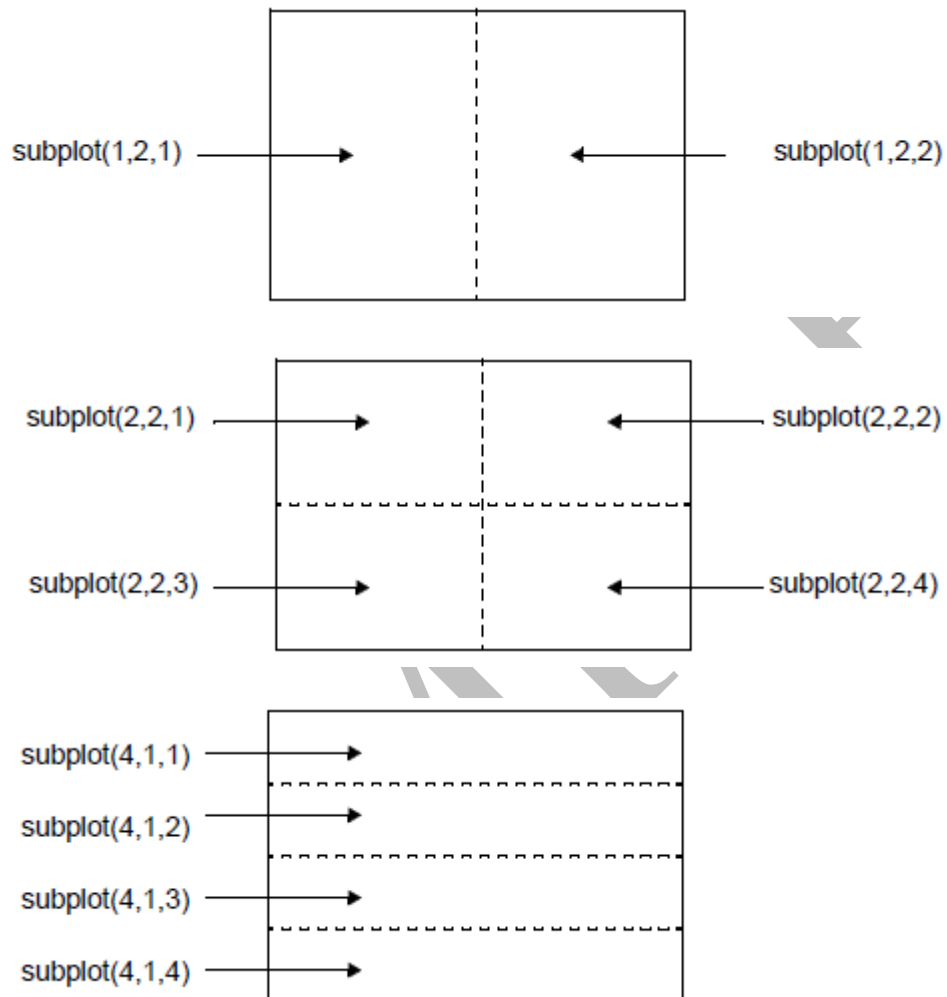
```

Now with the data cursor mode even if there are no toolbars anymore the user/customer can still click on any point of the chart to get some data tips on the screen:

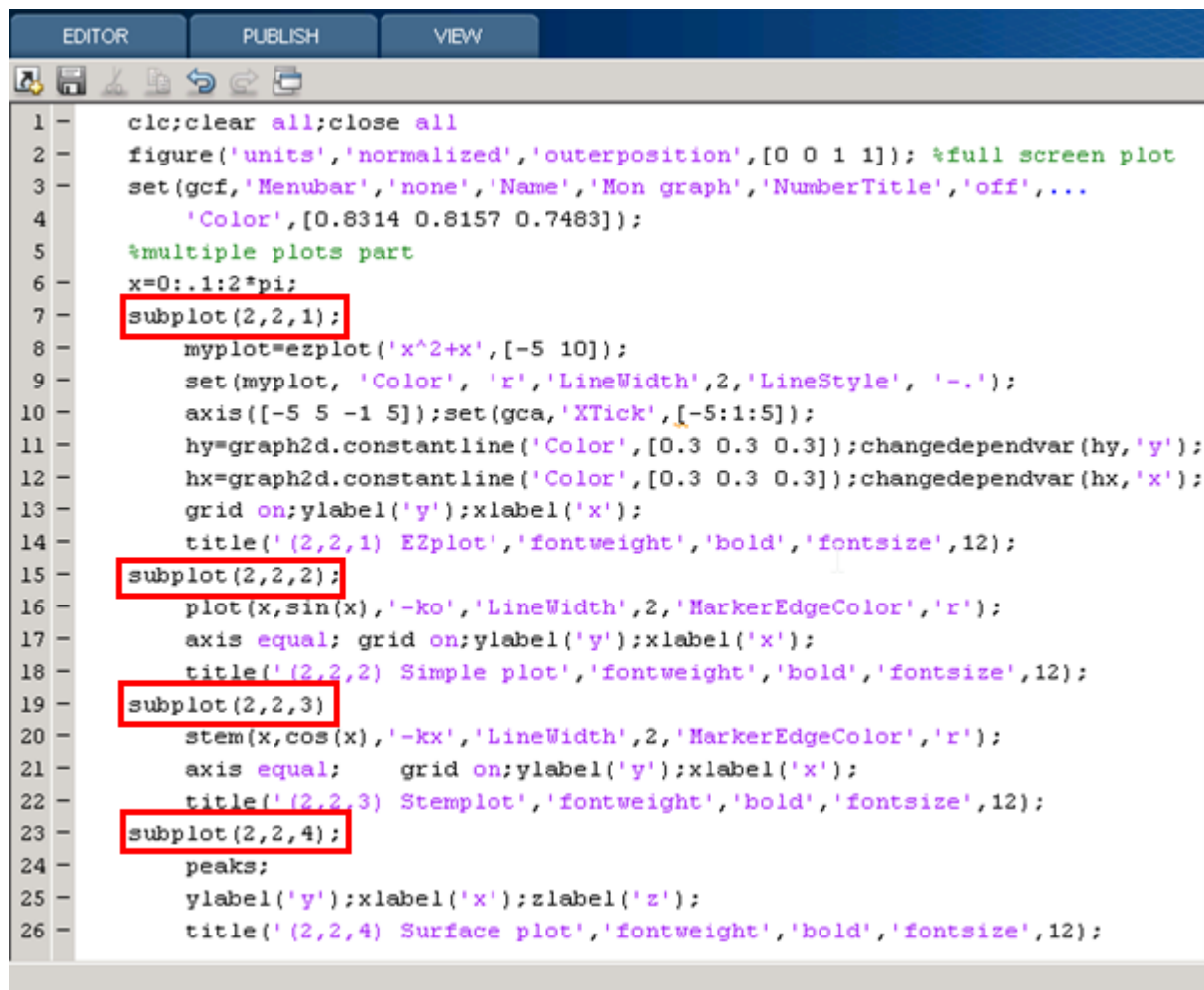


Subplots

Having multiple windows to show multiple charts can be very boring. You have the possibility in MATLAB as in R to create multiple plots window using the function **subplot(rows, columns, cell number)**. The main ideas is the following:



Here is an example:

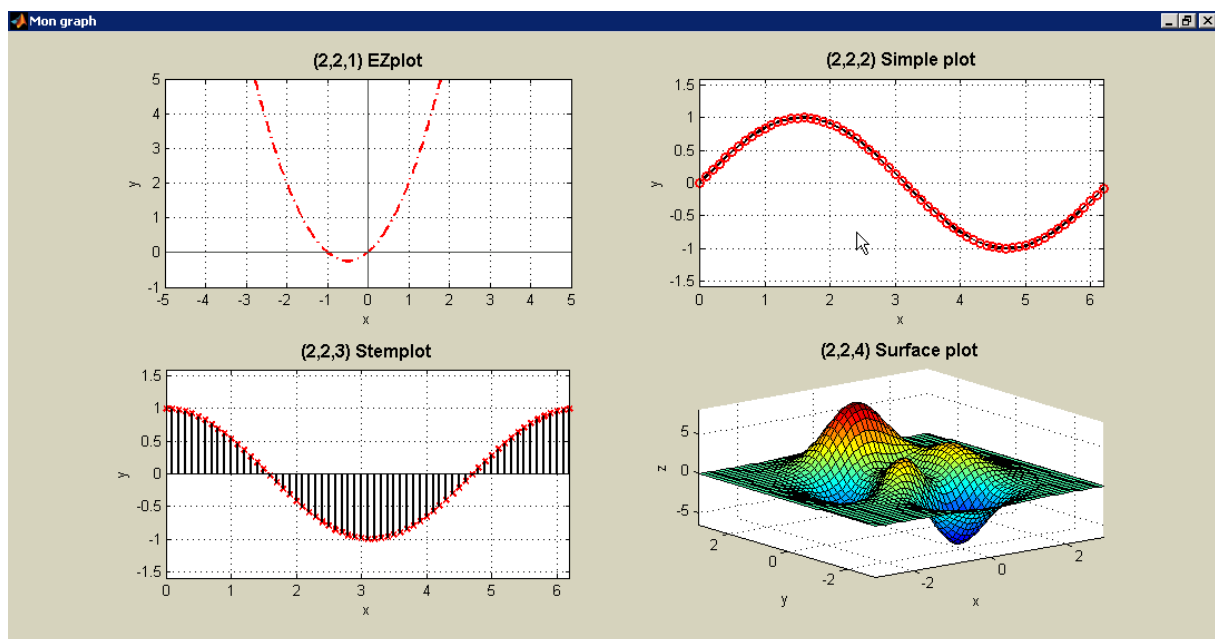


```

1 -  clc;clear all;close all
2 -  figure('units','normalized','outerposition',[0 0 1 1]); %full screen plot
3 -  set(gcf,'MenuBar','none','Name','Mon graph','NumberTitle','off',...
4 -    'Color',[0.8314 0.8157 0.7483]);
5 -  %multiple plots part
6 -  x=0:.1:2*pi;
7 -  subplot(2,2,1);
8 -      myplot=ezplot('x^2+x',[-5 10]);
9 -      set(myplot, 'Color', 'r','LineWidth',2,'LineStyle', '-.');
10 -      axis([-5 5 -1 5]);set(gca,'XTick',[-5:1:5]);
11 -      hy=graph2d.constantline('Color',[0.3 0.3 0.3]);changedependvar(hy,'y');
12 -      hx=graph2d.constantline('Color',[0.3 0.3 0.3]);changedependvar(hx,'x');
13 -      grid on;ylabel('y');xlabel('x');
14 -      title('(2,2,1) EZplot','fontweight','bold','fontsize',12);
15 -  subplot(2,2,2);
16 -      plot(x,sin(x),'-ko','LineWidth',2,'MarkerEdgeColor','r');
17 -      axis equal; grid on;ylabel('y');xlabel('x');
18 -      title('(2,2,2) Simple plot','fontweight','bold','fontsize',12);
19 -  subplot(2,2,3)
20 -      stem(x,cos(x),'-kx','LineWidth',2,'MarkerEdgeColor','r');
21 -      axis equal; grid on;ylabel('y');xlabel('x');
22 -      title('(2,2,3) Stemplot','fontweight','bold','fontsize',12);
23 -  subplot(2,2,4);
24 -      peaks;
25 -      ylabel('y');xlabel('x');zlabel('z');
26 -      title('(2,2,4) Surface plot','fontweight','bold','fontsize',12);

```

This will result to:



Greeks letters in legends and titles (LaTeX style)

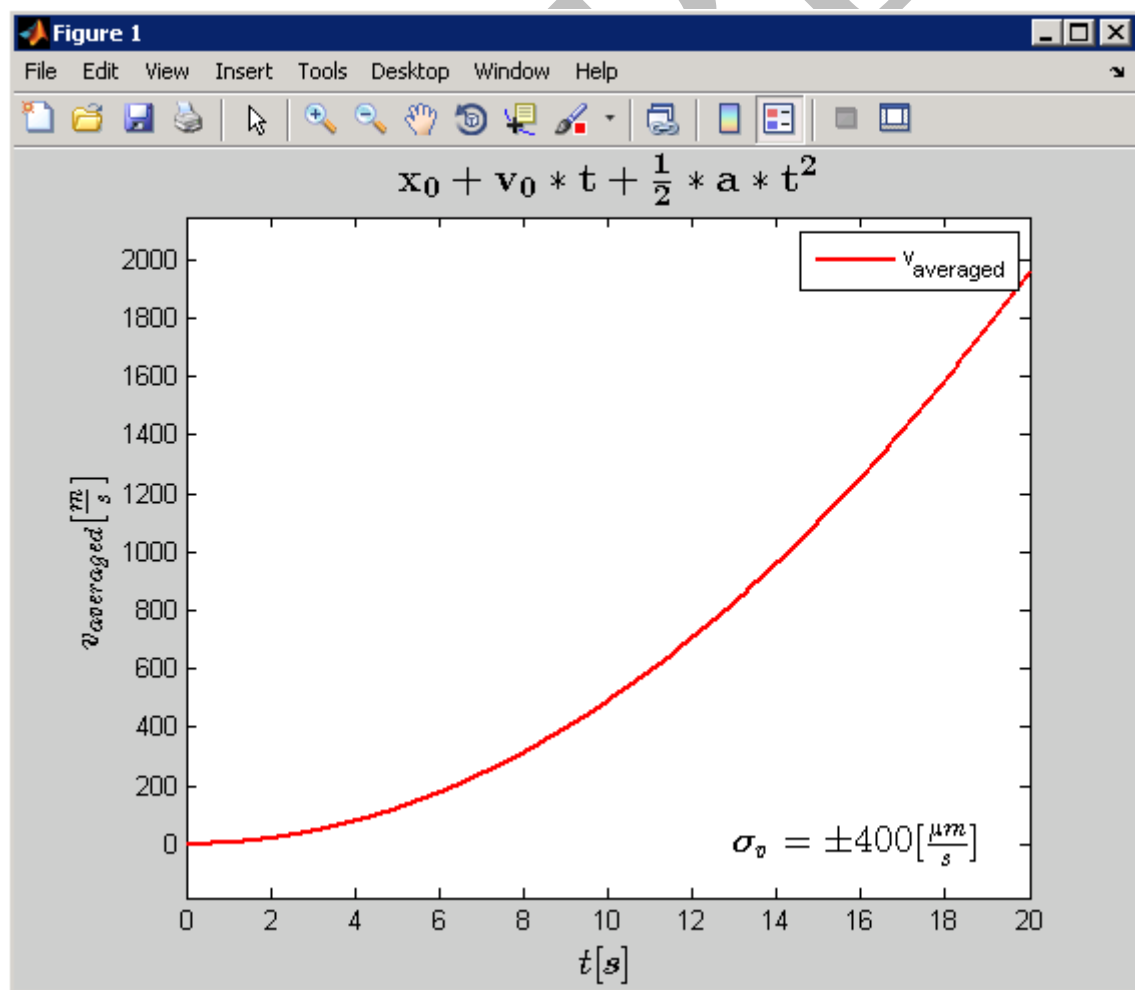
A very important option in any mathematical software is the ability to use LaTeX to write equations or mathematical symbols for different outputs. This is also possible with MATLAB by using the **"interpreter"**. You can see 5 examples just below:

```

1 clear all;close all;clc
2 x0=0;v0=0;a=9.81;
3 myplot=ezplot(sprintf('%f+%f*t+0.5*f*t^2',x0,v0,a),[0 20]);
4 set(myplot,'color','r','LineWidth',2,'LineStyle','-');
5 ylabel('$v_{averaged} [\frac{m}{s}]$','interpreter','latex','fontsize',14);
6 xlabel('$t [s]$','interpreter','latex','fontsize',14);
7 legend('$v_{averaged}$','interpreter','latex','fontsize',14);
8 text(13,0,'$\sigma_v=\pm 400 [\frac{\mu m}{s}]$','interpreter','latex','fontsize',14);
9 title('$\bf(x_0+v_0*t+\frac{1}{2}*a*t^2)$','interpreter','latex','fontsize',16)

```

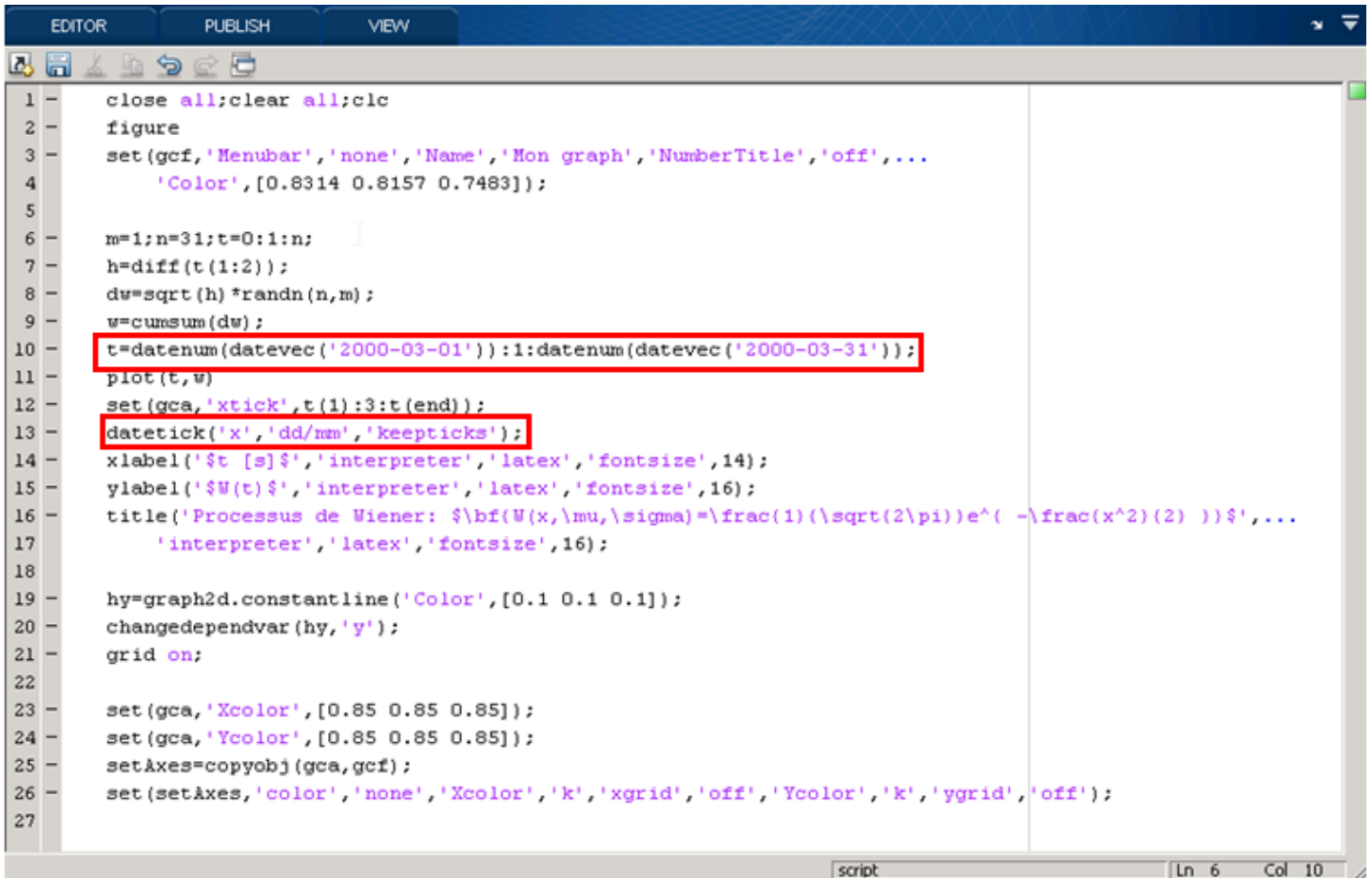
This will give the following result:



Plots with dates in axis

Managing dates is almost in all softwares an Art because of the lack of respect of international norms. Even if you or your customers respect international norms it is still a nightmare to manage in international finance. There are books (PDFs) of 40 pages explaining how to manager dates and times in international trading... then I let you imagine...

Here is an example with a date vector created by hand. As you can see we use it we use a function named **datetick()** to choose the format of the dates:



```

1 - close all;clear all;clc
2 - figure
3 - set(gcf,'MenuBar','none','Name','Mon graph','NumberTitle','off',...
4 -     'Color',[0.8314 0.8157 0.7483]);
5 -
6 - m=1;n=31;t=0:1:n;
7 - h=diff(t(1:2));
8 - dw=sqrt(h)*randn(n,m);
9 - w=cumsum(dw);
10 - t=datetime(datevec('2000-03-01')):1:datetime(datevec('2000-03-31'));
11 - plot(t,w)
12 - set(gca,'xtick',t(1):3:t(end));
13 - datetick('x','dd/mm','keepticks');
14 - xlabel('$t$ [s]','interpreter','latex','fontsize',14);
15 - ylabel('$W(t)$','interpreter','latex','fontsize',16);
16 - title('Processus de Wiener: $\bf(W(x,\mu,\sigma)=\frac{1}{\sqrt{2}\pi})e^{-(\frac{x^2}{2})}$',...
17 -     'interpreter','latex','fontsize',16);
18 -
19 - hy=graph2d.constantline('Color',[0.1 0.1 0.1]);
20 - changedependvar(hy,'y');
21 - grid on;
22 -
23 - set(gca,'Xcolor',[0.85 0.85 0.85]);
24 - set(gca,'Ycolor',[0.85 0.85 0.85]);
25 - setAxes=copyobj(gca,gcf);
26 - set(setAxes,'color','none','Xcolor','k','xgrid','off','Ycolor','k','ygrid','off');
27 -

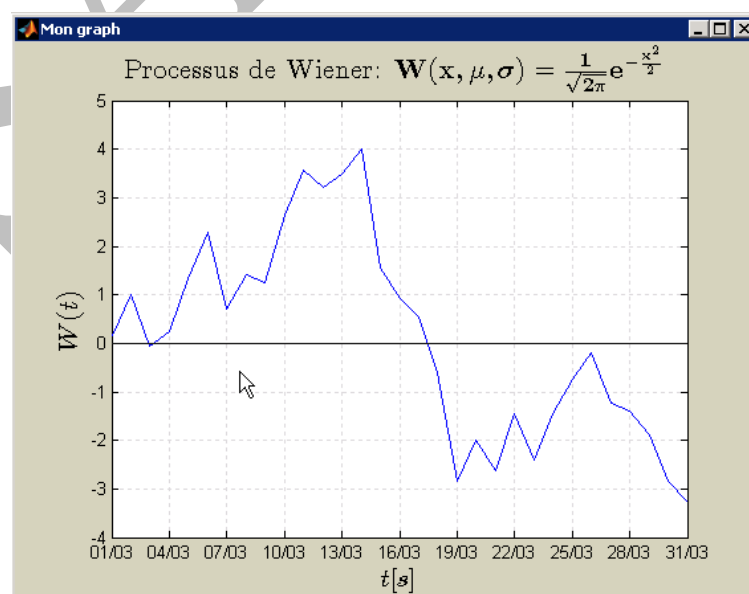
```

Here is a table with the most common used dates format:

Symbolic Identifier	Description	Example
YYYY	Year in full	1990, 2002
YY	Year in two digits	90, 02
QQ	Quarter year using letter Q and one digit	Q1
mmmm	Month using full name	March, December
mmm	Month using first three letters	Mar, Dec
mm	Month in two digits	03, 12
m	Month using capitalized first letter	M, D
dddd	Day using full name	Monday, Tuesday
ddd	Day using first three letters	Mon, Tue
dd	Day in two digits	05, 20
d	Day using capitalized first letter	M, T
HH	Hour in two digits (no leading zeros when symbolic identifier AM or PM is used)	05, 5 AM
MM	Minute in two digits	12, 02
SS	Second in two digits	07, 59
FFF	Millisecond in three digits	057
AM or PM	AM or PM inserted in date string	3:45:02 PM

Table 2 Common dates format syntaxes

The result will as following:



Multiple axis on line plots

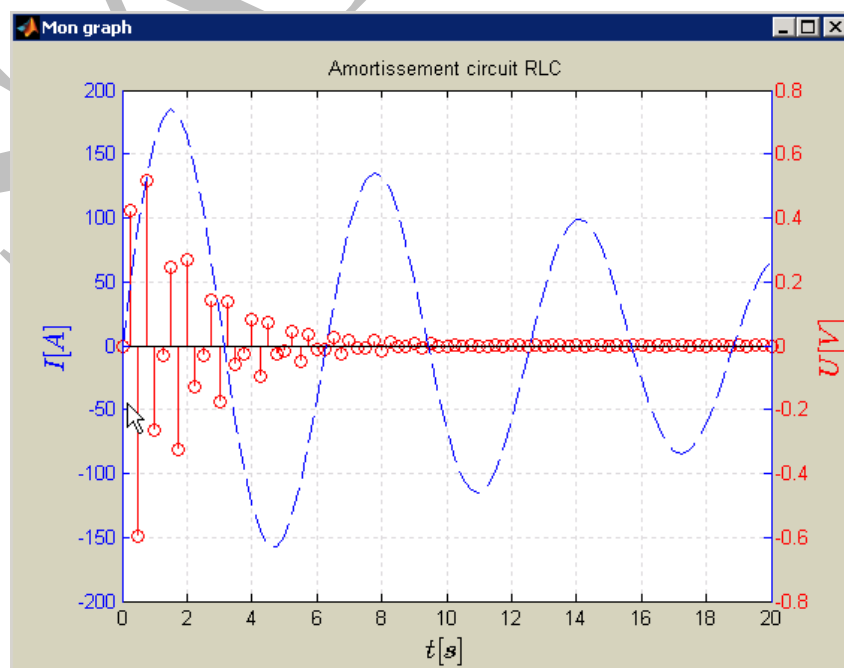
Multiple axes are a recurrent request in business usage or in physics and electronics. We will see here we can manage this in different ways. But first we create a simple double Y axes plot with the function **plotyy()** and we change the color of the axes:

```

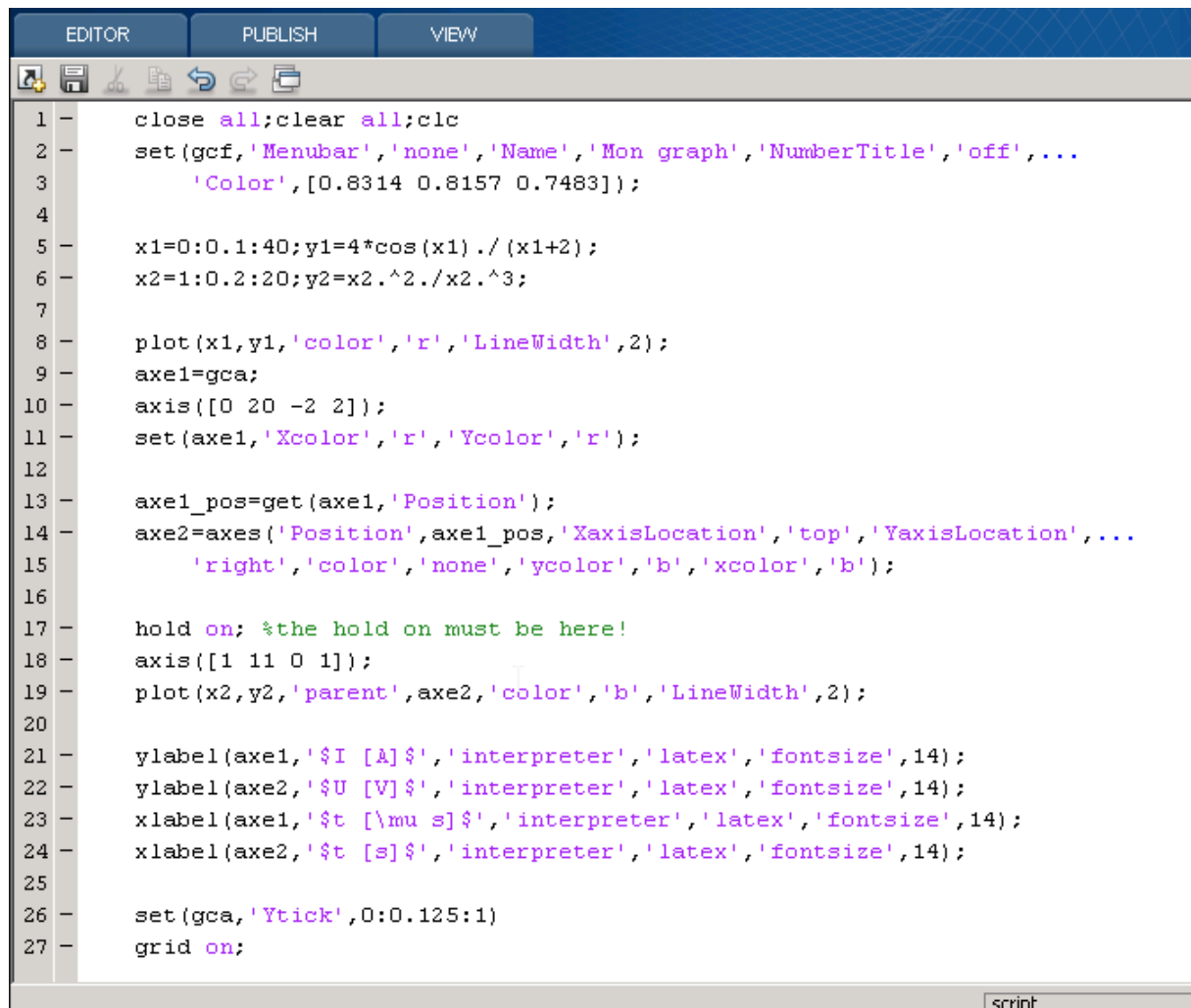
1 - close all;clear all;clc
2 - figure
3 - set(gcf,'Menubar','none','Name','Mon graph','NumberTitle','off',...
4 -     'Color',[0.8314 0.8157 0.7483]);
5 -
6 - t=0:0.25:20;
7 - I=200*exp(-0.05*t).*sin(t);
8 - U=0.8*exp(-0.5*t).*sin(10*t);
9 - [ax, line1, line2]=plotyy(t,I,t,U,'plot','stem');
10 -
11 - set(line1,'Color','b','LineStyle','--');
12 - set(line2,'Color','r');
13 - set(ax(2),'ylabel','r');
14 -
15 - title('Amortissement circuit RLC');
16 - xlabel('$t$ [s]','interpreter','latex','fontSize',14);
17 - ylabel(ax(1),'$I$ [A]','interpreter','latex','fontSize',14);
18 - ylabel(ax(2),'$U$ [V]','interpreter','latex','fontSize',14);
19 -
20 - grid on;
21 -
22 - set(gca,'Xcolor',[0.85 0.85 0.85]);
23 - set(gca,'Ycolor',[0.85 0.85 0.85]);
24 - setAxes=copyobj(gca,gcf);
25 - set(setAxes,'color','none','Xcolor','k','xgrid','off','Ycolor','b','ygrid','off');

```

The results will looks as following:



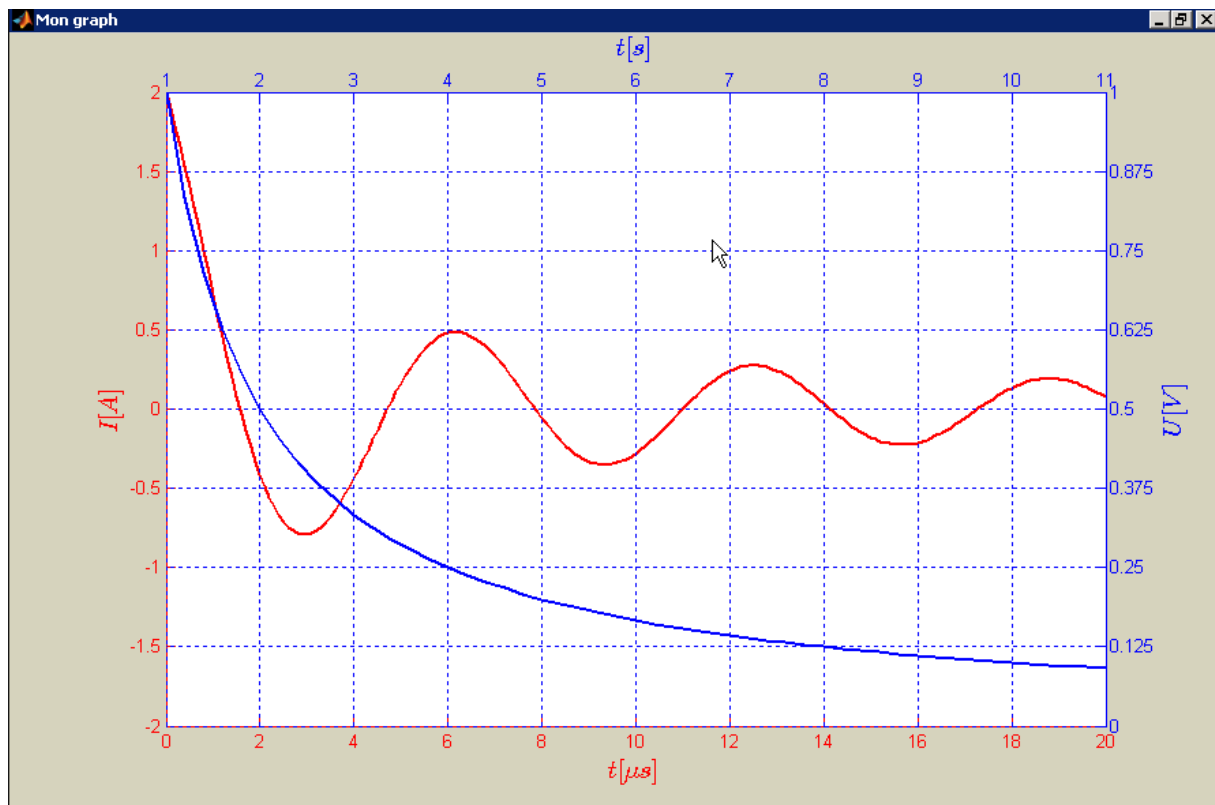
If we want a rare 4 axes plot (until today only one customer in business asked me for that...) you have to use a lot of MATLAB tricks that are almost impossible to guess in my point of view and the adjustment of tick labels is always a pain in the ass:



```

1 - close all;clear all;clc
2 - set(gcf,'MenuBar','none','Name','Mon graph','NumberTitle','off',...
3 -     'Color',[0.8314 0.8157 0.7483]);
4 -
5 - x1=0:0.1:40;y1=4*cos(x1)./(x1+2);
6 - x2=1:0.2:20;y2=x2.^2./x2.^3;
7 -
8 - plot(x1,y1,'color','r','LineWidth',2);
9 - axel=gca;
10 - axis([0 20 -2 2]);
11 - set(axel,'Xcolor','r','Ycolor','r');
12 -
13 - axel_pos=get(axel,'Position');
14 - axe2=axes('Position',axel_pos,'AxisLocation','top','YaxisLocation',...
15 -     'right','color','none','ycolor','b','xcolor','b');
16 -
17 - hold on; %the hold on must be here!
18 - axis([1 11 0 1]);
19 - plot(x2,y2,'parent',axe2,'color','b','LineWidth',2);
20 -
21 - ylabel(axel,'$I [A]$', 'interpreter','latex','fontsize',14);
22 - ylabel(axe2,'$U [V]$', 'interpreter','latex','fontsize',14);
23 - xlabel(axel,'$t [\mu s]$', 'interpreter','latex','fontsize',14);
24 - xlabel(axe2,'$t [s]$', 'interpreter','latex','fontsize',14);
25 -
26 - set(gca,'Ytick',0:0.125:1)
27 - grid on;

```

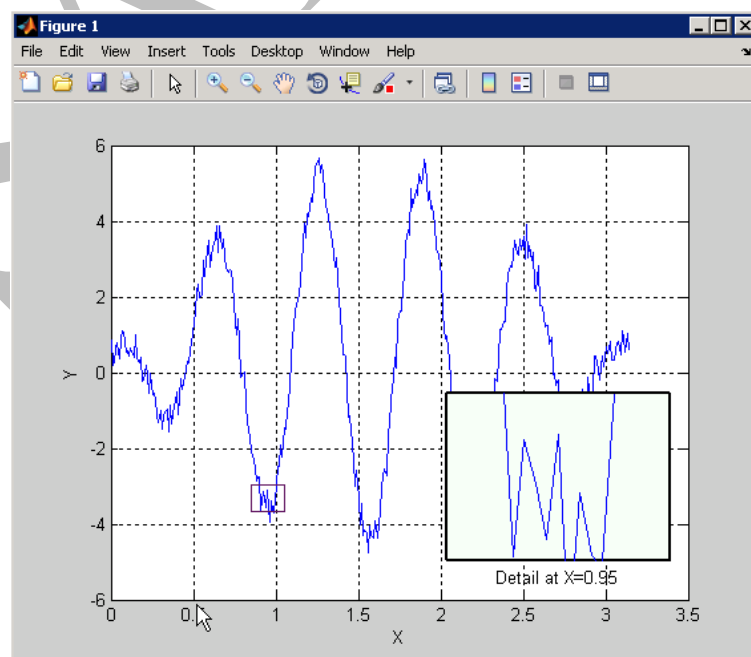
Magnifier on line plot

Create a magnifier is a wonderful recapitulative exercise for almost all what we have seen until now about charts in MATLAB. Let see how to do this and the result:

```

1  % Create a dummy function
2  x = 0:0.01:pi;
3  y = 5*sin(x).*cos(10*x) + rand(size(x));
4  plot(x, y); grid on;
5  xlabel('X');
6  ylabel('Y');
7
8  % Specify the position and the size of the rectangle
9  x_r = 0.95; y_r = -3.3; w_r = 0.2; h_r = 0.7;
10 rectangle('Position', [x_r-w_r/2, y_r-h_r/2, w_r, h_r], ...
11           'EdgeColor', [0.4, 0.1, 0.4]);
12
13 % We specify the position and the size of the second axis
14 % The position is in percentage of the whole plot area
15 x_a = 0.58; y_a = 0.18; w_a = 0.3; h_a = 0.3;
16 ax = axes('Units', 'Normalized', ...
17           'Position', [x_a, y_a, w_a, h_a], ...
18           'XTick', [], ... %no ticks
19           'YTick', [], ... %no ticks
20           'Box', 'on', ...
21           'LineWidth', 2, ...
22           'Color', [0.95, 0.99, 0.95]);
23 hold on;
24 plot(x, y);
25 lbl=xlabel(strcat('Detail at X=',num2str(x_r)));
26 axis([x_r-w_r/2, x_r+w_r/2, y_r-h_r/2, y_r+h_r/2]);

```



2D vertical/horizontal bar charts

2D vertical bar charts are used a lot in business, statistics and finance. We will see here the most common usage of this type of charts and once again give a lot of tips to customize them because it is not as easy as in a spreadsheet software like Microsoft Excel.

Multiple series bar chart

First we begin with a very common task... Plot multiple series of data on a charts using the function **bar()** whose last parameter is the distance between the groups:

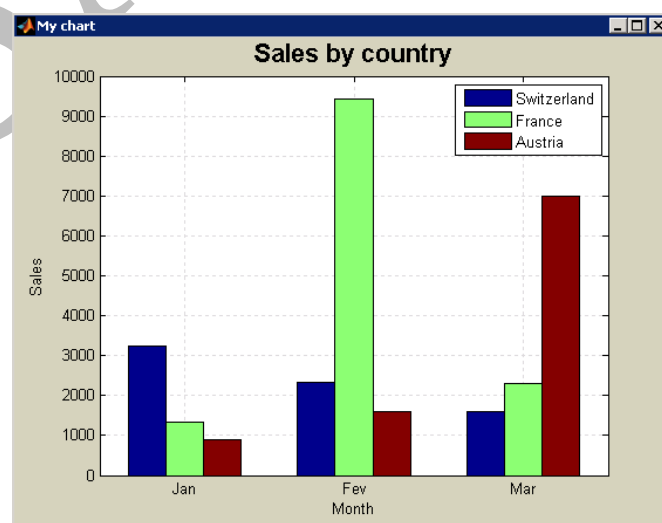
```

1 - close all;clear all;clc
2 - figure
3 - set(gcf,'Menubar','none','Name','My chart','NumberTitle','off',...
4 -     'Color',[0.8314 0.8157, 0.7483]);
5
6 - Switzerland=[3233 2323 1600];
7 - France=[1340 9432 2304];
8 - Austria=[900 1594 7000];
9 - bar(1:3,[Switzerland' France' Austria'],1)
10 - set(gca,'Xtick',1:3,'xTickLabel',{'Jan', 'Fev', 'Mar'});
11 - legend('Switzerland','France','Austria');
12 - xlabel('Month');ylabel('Sales');
13 - title('Sales by country','fontweight','bold','fontsize',16);
14
15 - grid on
16
17 - set(gca,'Xcolor',[0.85 0.85 0.85]);
18 - set(gca,'Ycolor',[0.85 0.85 0.85]);
19 - setAxes=copyobj(gca,gcf);
20 - set(setAxes,'color','none','Xcolor','k','xgrid','off','Ycolor','k','ygrid','off');

```

You can see at the end the trick to keep the major axis visible after we have customized the grid (this is a small difficulty with bar charts...).

The result looks like this:



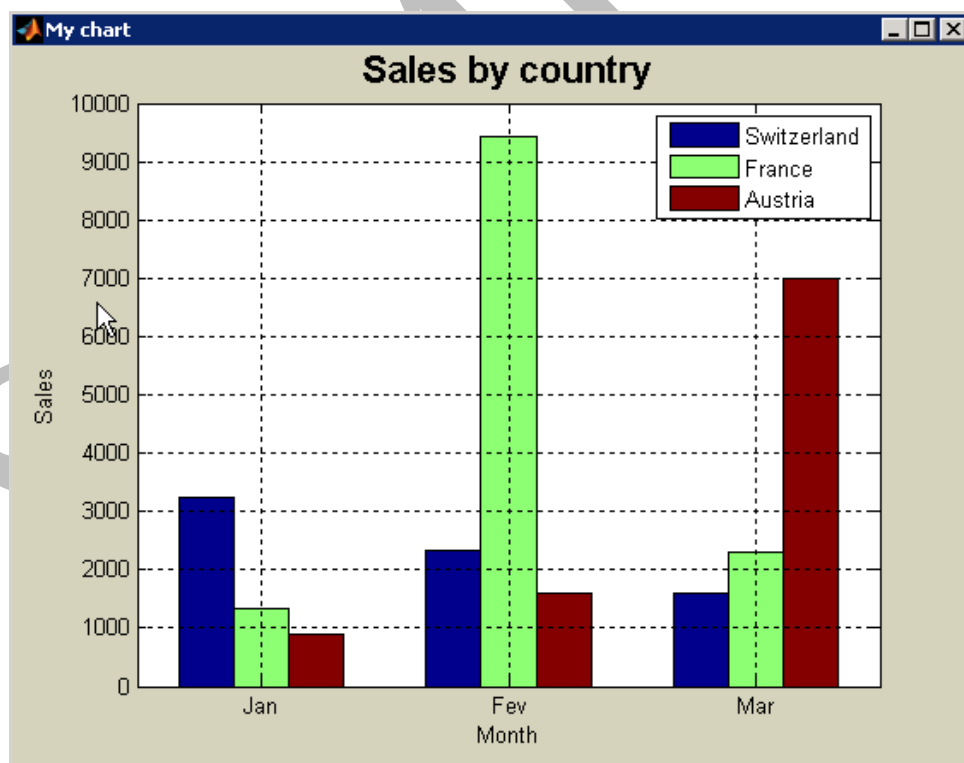
If you want the gridline on the top you just need to write the following:

```

1  close all;clear all;clc
2  figure
3  set(gcf,'Menubar','none','Name','My chart','NumberTitle','off',...
4      'Color',[0.8314 0.8157, 0.7483]);
5
6  Switzerland=[3233 2323 1600];
7  France=[1340 9432 2304];
8  Austria=[900 1594 7000];
9  bar(1:3,[Switzerland' France' Austria'],1)
10 set(gca,'Xtick',1:3,'xTickLabel',{'Jan', 'Fev', 'Mar'});
11 legend('Switzerland','France','Austria');
12 xlabel('Month');ylabel('Sales');
13 title('Sales by country','fontweight','bold','fontsize',16);
14
15 grid on
16 set(gca,'Layer','top');
17

```

You will get:



Stacked bar chart

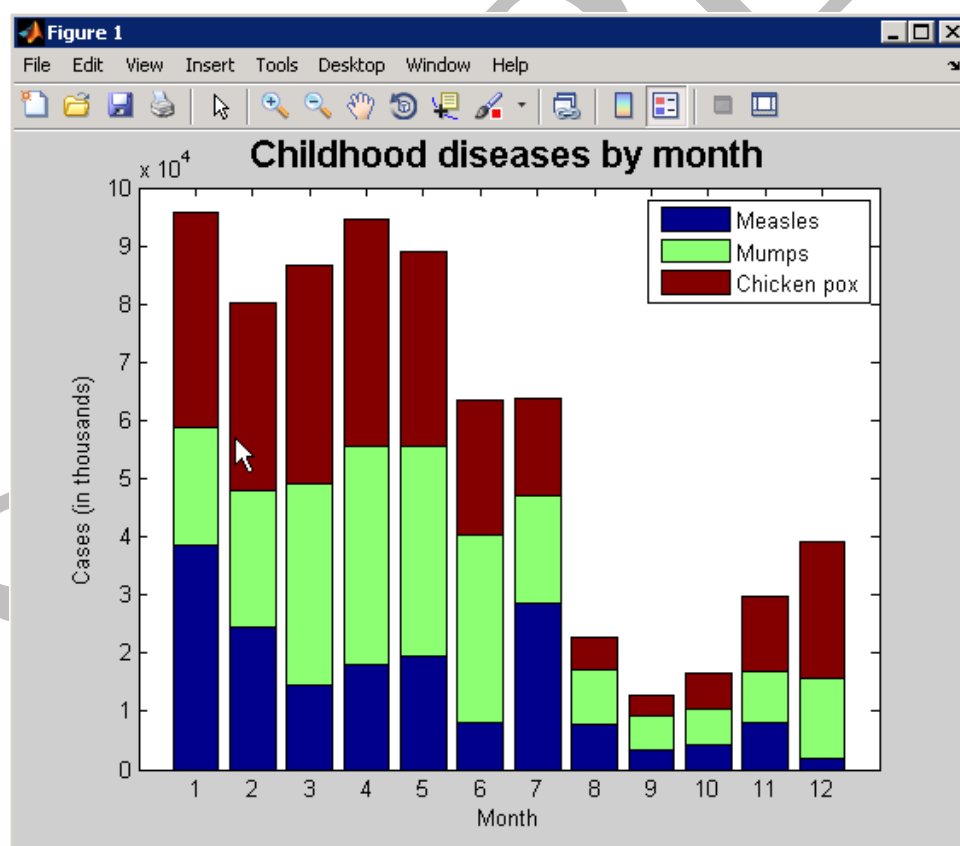
Another typical common chart in spreadsheet softwares is to create a stacked bar chart. This is very easy we just have to change the last parameter of the function `bar()` as you can see below:

```

1  %plot 2d histogramme groupé
2  %Pour normaliser il faut diviser (il n'existe pas de bar chart normalisé
3  %dans Matlab)
4  measles = [38556 24472 14556 18060 19549 8122 28541 7880 3283 4135 7953 1884];
5  mumps = [20178 23536 34561 37395 36072 32237 18597 9408 6005 6268 8963 13882];
6  chickenPox = [37140 32169 37533 39103 33244 23269 16737 5411 3435 6052 12825 23332];
7  figure;
8  bar(1:12, [measles' mumps' chickenPox'], 'stack');
9  axis([0 13 0 100000]);
10 title('Childhood diseases by month','fontweight','bold','fontsize',16);
11 xlabel('Month');
12 ylabel('Cases (in thousands)');
13 legend('Measles', 'Mumps', 'Chicken pox');
14 [im_hatch,colorlist] = applyhatch_pluscolor(1,'-x',1,[1 0 1 1]);

```

The result will be:



Stacked normalized bar chart

Even more frequent as stacked bar chart and use a lot in finance especially in the field of portfolio allocation, we have the stacked normalized bar chart.

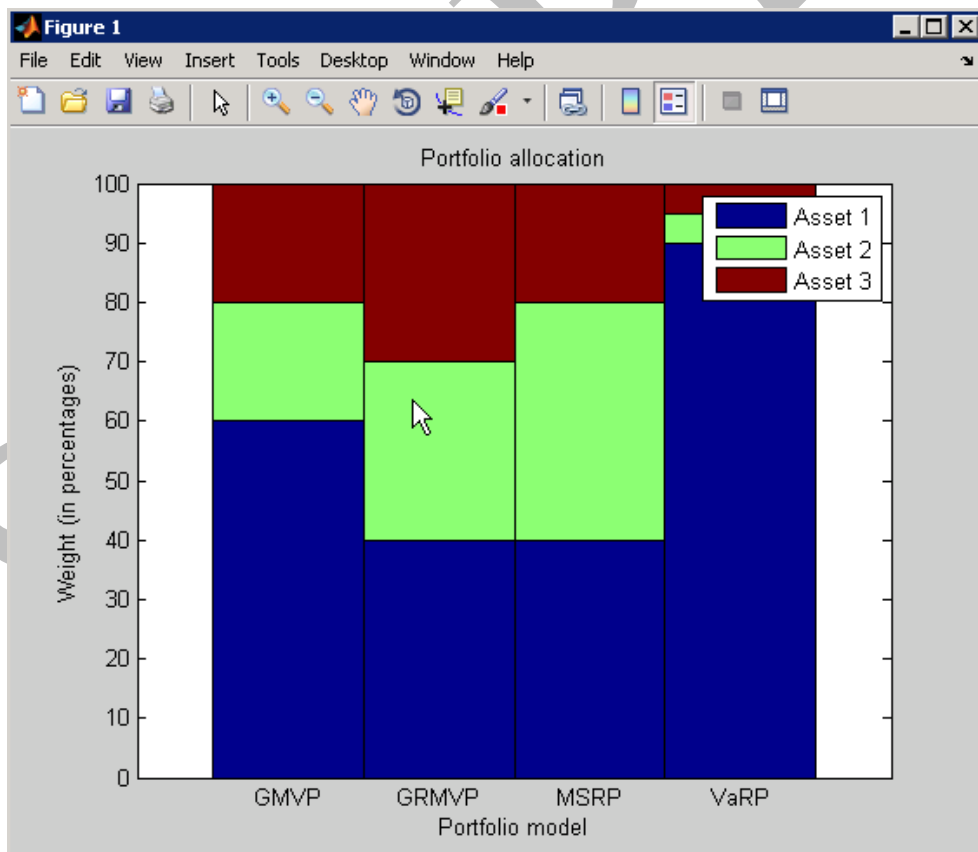
Because this type of chart doesn't exist in MATLAB you will always have to normalized your vectors yourself (simple arithmetic). The only trick to know otherwise is to give the parameter "1" to the **bar()** function to have a null distance between column bars:

```

1  asset_1 = [0.6 0.4 0.4 0.9]';
2  asset_2 = [0.2 0.3 0.4 0.05]';
3  asset_3 = [0.2 0.3 0.2 0.05]';
4  figure;
5  bar(1:4, [asset_1*100 asset_2*100 asset_3*100], 1, 'stack');
6  axis([0 5 0 100]);
7  title('Portfolio allocation');
8  xlabel('Portfolio model');
9  ylabel('Weight (in percentages)');
10 legend('Asset 1', 'Asset 2', 'Asset 3');
11 set(gca, 'XTickLabel', {'GMVP', 'GRMVP', 'MSRP', 'VaRP'})

```

The result will look as following:



Bar chart with error bars

Now a very common bar chart in physics and statistics is this with error bars at the top of the bars. To do this it is simple if we mix our knowledge of the function **bar()** and **errorbar()**

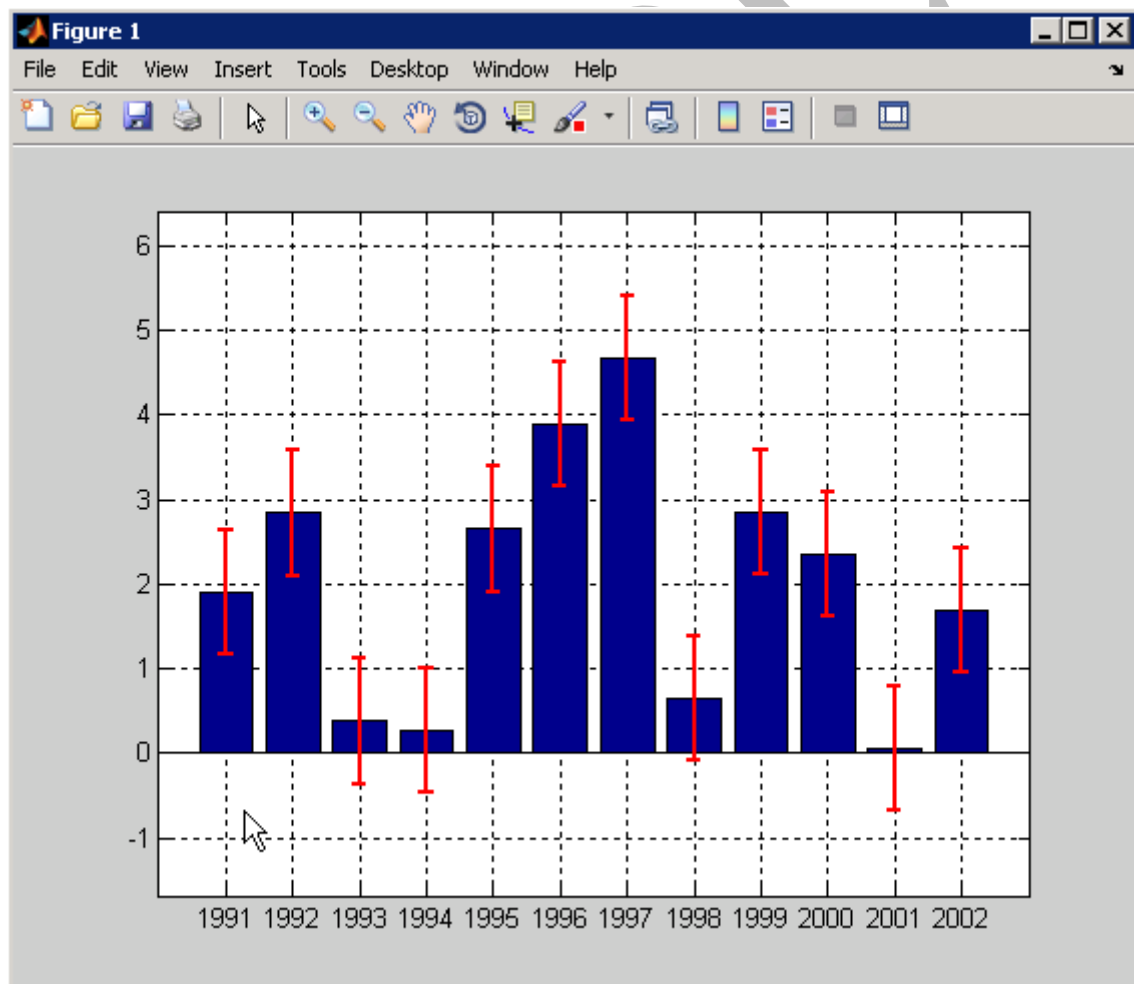
but we have to add a line at the end to manage an awful behavior of MATLAB with X and Y in such a situation:

```

1 - clear all;clc;close all
2 - intYears=12;
3 - y=5*rand(intYears,1);
4 - e=std(y/2)*ones(size(y));
5 - x=1991:1:1991+intYears-1;
6 - bar(x,y);
7 - hold on;
8 - grid on;
9 - h=errorbar(x,y,e);
10 - set(h(1),'color','r','LineStyle','none','LineWidth',2);
11 - %we are obliged to manage axes range otherwise the chart is sometimes awful
12 - ylim([min(y-e)-1 max(y+e)+1]);xlim([(1991)-1 (1991+intYears-1)+1])

```

The result looks like:



Bar chart with bar having a special a color

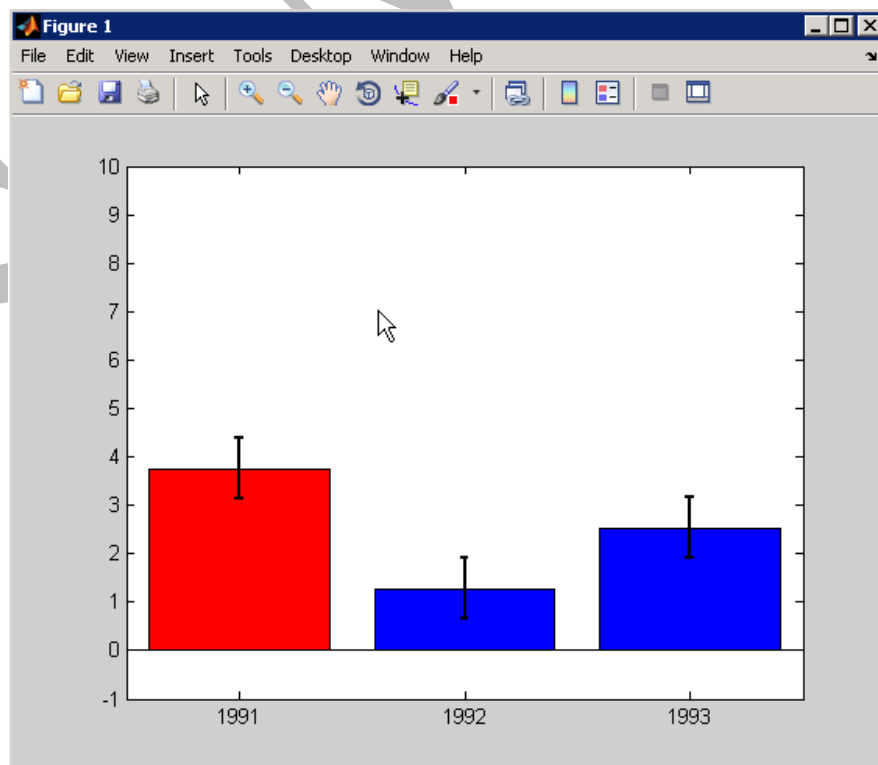
To change the color of only one bar the manipulation also difficult to guess. We have to build the bar chart bar by bar... as you can see here below:

```

1 - clear all;clc;close all
2 - y=5*rand(3,1);
3 - e=std(y/2)*ones(size(y));
4 - x=1991:1993;
5 - for i=1:length(y)
6 -     if y(i)>3.5
7 -         colorcode = 'r';
8 -     else
9 -         colorcode = 'b';
10 -    end
11 -    bar(i, y(i), colorcode);
12 -    hold on;
13 - end
14 - set(gca,'Xtick',1:3,'XTickLabel',x)
15 - h=errorbar(1:3,y,e);
16 - set(h(1),'color','k','LineStyle','none','LineWidth',2);
17 - ylim([-1 10]);

```

The result will looks as following:



Tornado Chart

Tornados chart are used a lot in finance this is why we have to see how to create such a chart.

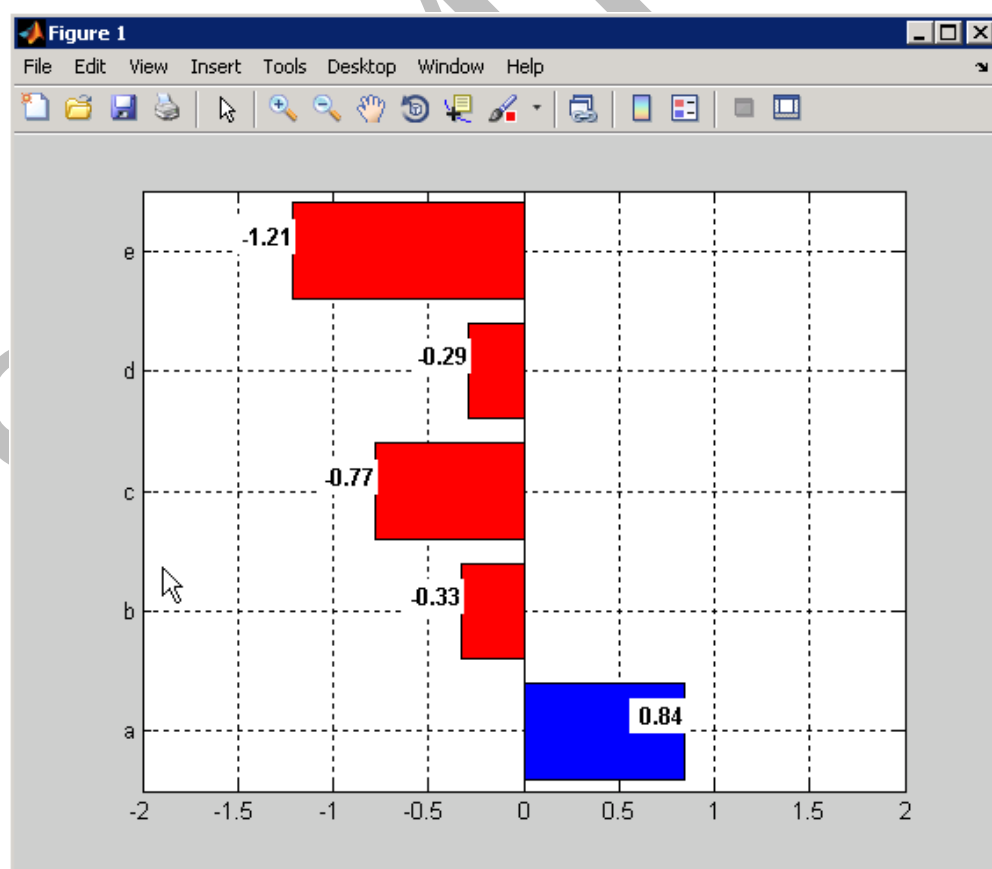
First we will have to use the **barh()** function after to use a colormap trick for negative/positive values et set ticks manually and also to position values "manually" on the top of the bars as you can see below:

```

1 - y=rand(5,1)*3-1.5;
2
3 - hb=barh(y);
4 - set(get(hb,'children'),'cdata', sign(y));
5 - colormap([1 0 0; 0 0 1]); % red & blue in rgb
6 - set(gca,'Ytick',1:5,'YTickLabel',{'a'; 'b'; 'c'; 'd'; 'e'})
7 - text(y,1:5,num2str(y,'%0.2f'),'HorizontalAlignment','right',...
8     'VerticalAlignment','bottom','FontWeight','bold',...
9     'BackgroundColor',[1 1 1])
10 xlim([-2 2]);
11 - grid on;

```

The result will be:



Bar chart with line and multiple axis

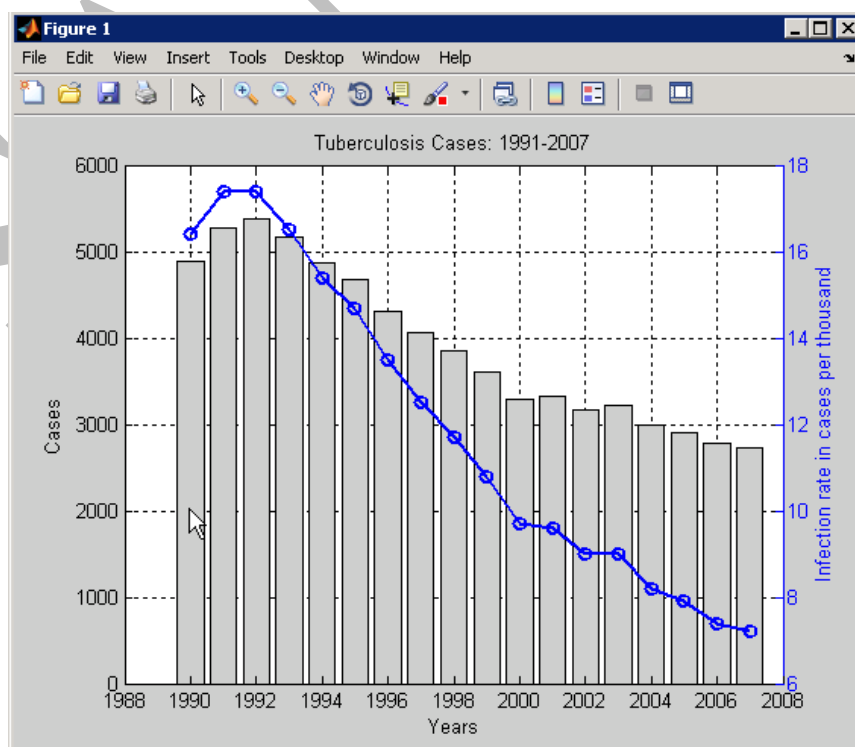
Sometimes users want to mix a chart line with a line chart for readiness purposes. For this purpose you can use the function **plotyy()** that not only management multiple axes but also multiple chart type as you can see below:

```

1  TBdata = [1990 4889 16.4; 1991 5273 17.4; 1992 5382 17.4; 1993 5173 16.5;
2           1994 4860 15.4; 1995 4675 14.7; 1996 4313 13.5; 1997 4059 12.5;
3           1998 3855 11.7; 1999 3608 10.8; 2000 3297 9.7; 2001 3332 9.6;
4           2002 3169 9.0; 2003 3227 9.0; 2004 2989 8.2; 2005 2903 7.9;
5           2006 2779 7.4; 2007 2725 7.2];
6
7  years = TBdata(:,1);
8  cases = TBdata(:,2);
9  rate = TBdata(:,3);
10
11  [ax, h1, h2] = plotyy(years, cases, years, rate, 'bar', 'plot');
12
13  set(h1, 'FaceColor', [0.8, 0.8, 0.8]);
14  set(h2, 'LineWidth', 2, 'Marker', 'o');
15  grid on;
16
17  title('Tuberculosis Cases: 1991-2007');
18  xlabel('Years');
19
20  set(get(ax(1), 'Ylabel'), 'String', 'Cases');
21  set(get(ax(2), 'Ylabel'), 'String', 'Infection rate in cases per thousand');

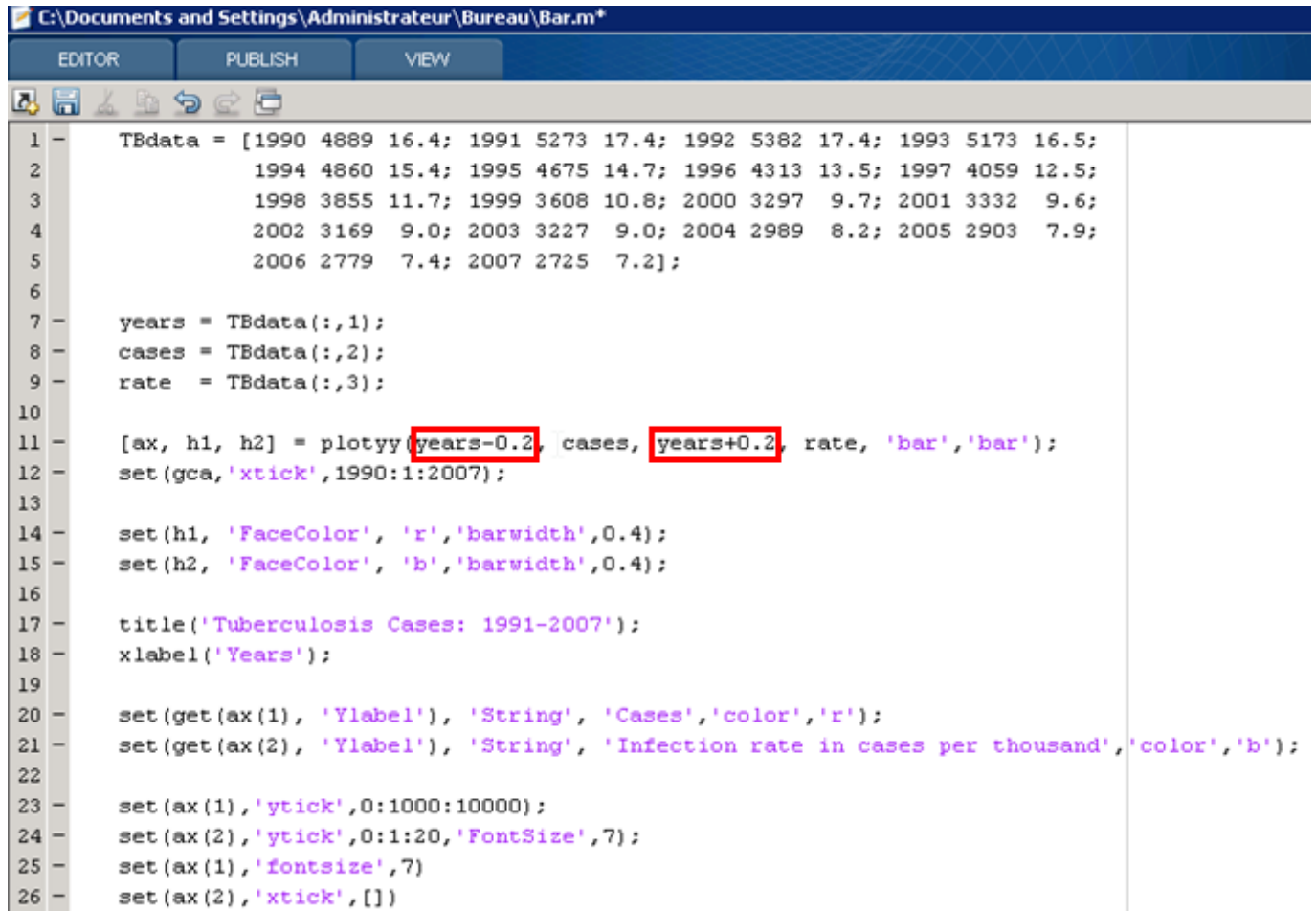
```

The result will be:



Two series bar chart with multiple axis

Another variant of the previous example is a multiple axis plot with two bar charts. For sure we will still use the **plotyy()** but MATLAB has a problem managing the same abscises values. Then we have to use a trick:

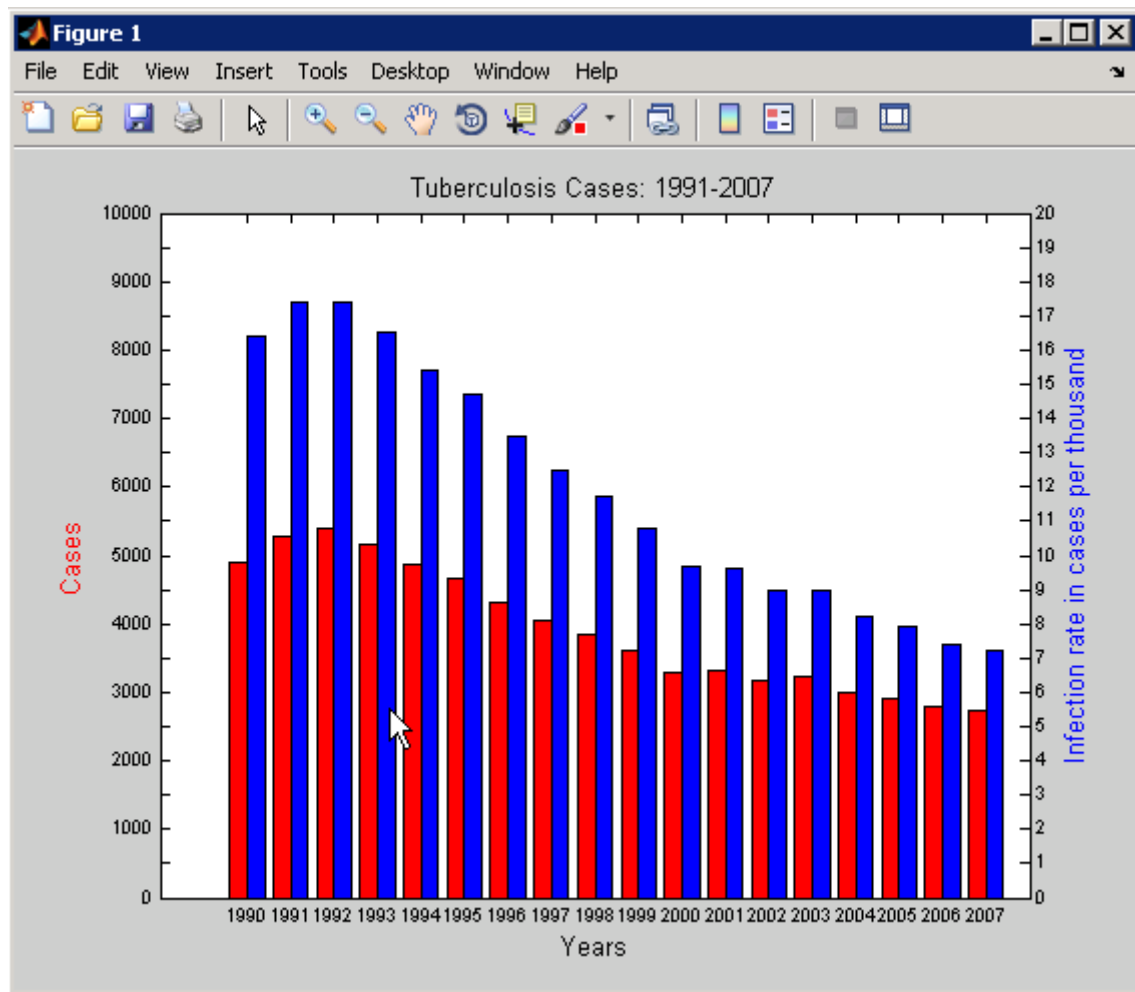


```

1 - TBdata = [1990 4889 16.4; 1991 5273 17.4; 1992 5382 17.4; 1993 5173 16.5;
2 -           1994 4860 15.4; 1995 4675 14.7; 1996 4313 13.5; 1997 4059 12.5;
3 -           1998 3855 11.7; 1999 3608 10.8; 2000 3297 9.7; 2001 3332 9.6;
4 -           2002 3169 9.0; 2003 3227 9.0; 2004 2989 8.2; 2005 2903 7.9;
5 -           2006 2779 7.4; 2007 2725 7.2];
6 -
7 - years = TBdata(:,1);
8 - cases = TBdata(:,2);
9 - rate = TBdata(:,3);
10 -
11 - [ax, h1, h2] = plotyy(years-0.2, cases, years+0.2, rate, 'bar','bar');
12 - set(gca,'xtick',1990:1:2007);
13 -
14 - set(h1, 'FaceColor', 'r','barwidth',0.4);
15 - set(h2, 'FaceColor', 'b','barwidth',0.4);
16 -
17 - title('Tuberculosis Cases: 1991-2007');
18 - xlabel('Years');
19 -
20 - set(get(ax(1), 'Ylabel'), 'String', 'Cases','color','r');
21 - set(get(ax(2), 'Ylabel'), 'String', 'Infection rate in cases per thousand','color','b');
22 -
23 - set(ax(1),'ytick',0:1000:10000);
24 - set(ax(2),'ytick',0:1:20,'FontSize',7);
25 - set(ax(1),'fontSize',7);
26 - set(ax(2),'xtick',[])

```

The result is on next page:



Probability histogram with density curve (pdf)

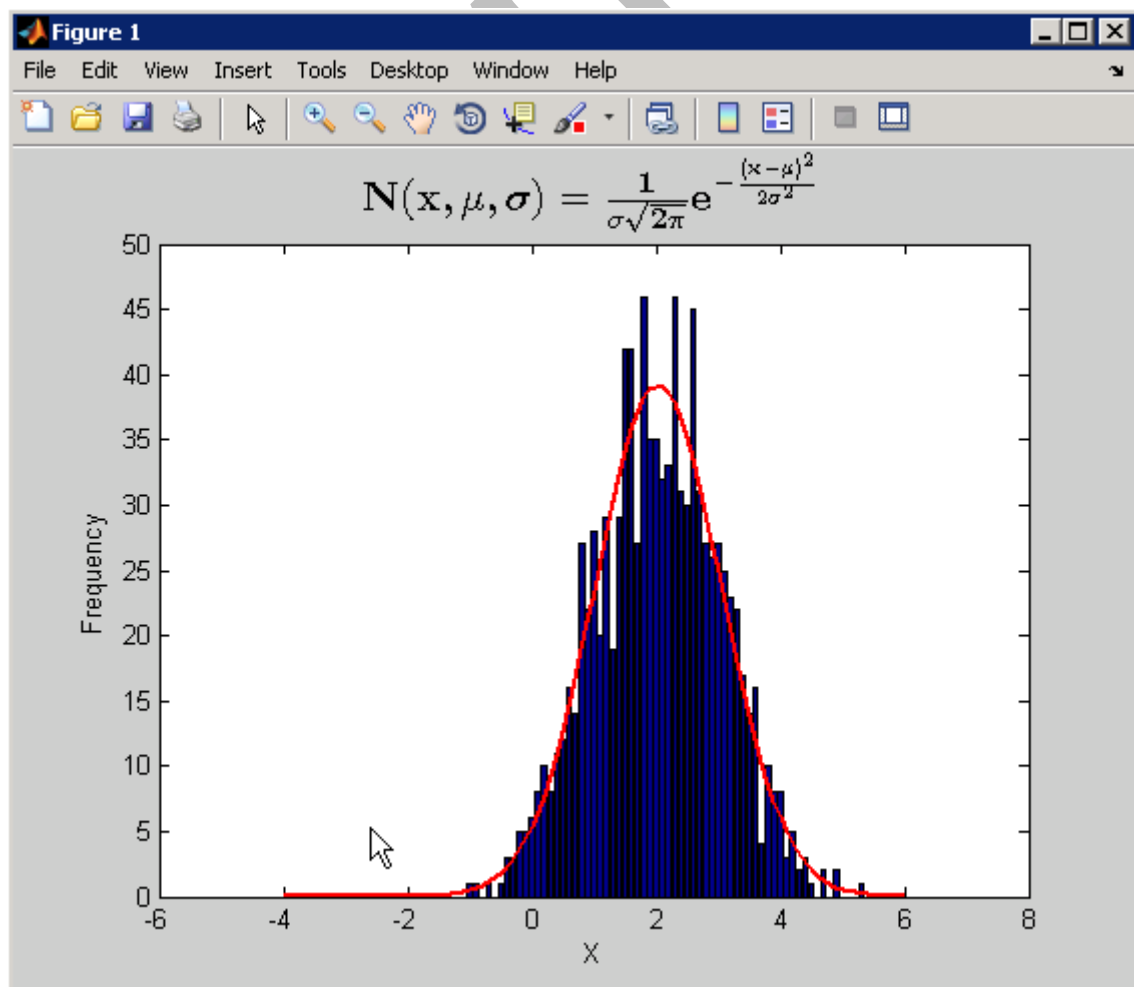
This is one of the most used bar chart type in Statistics. To make a PDF (probability density function) of a random variable we need to use the two functions **hist()** and **bar()** together:

```

1 %variable aléatoire normale centrée d'espérance 2 et d'écart-type unitaire
2 y=randn(1000,1) + 2; %le +2 est pour décaler la moyenne à la valeur de 2
3 x=-4:0.1:6;
4 n=hist(y,x);
5 bar(x,n);
6 MEAN=mean(y);
7 STD=sqrt(mean((y-MEAN).^2));
8 f=(1/(STD*sqrt(2*pi)))*exp(-0.5*((x-MEAN)/STD).^2);
9 f=f*sum(n)/sum(f);
10 hold on
11 plot(x,f,'r','LineWidth',2);
12 xlabel('X');ylabel('Frequency');
13 title('$\bf(N(x,\mu,\sigma)=\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{(x-\mu)^2}{2\sigma^2}})$',...
14 'interpreter','latex','fontsize',16);

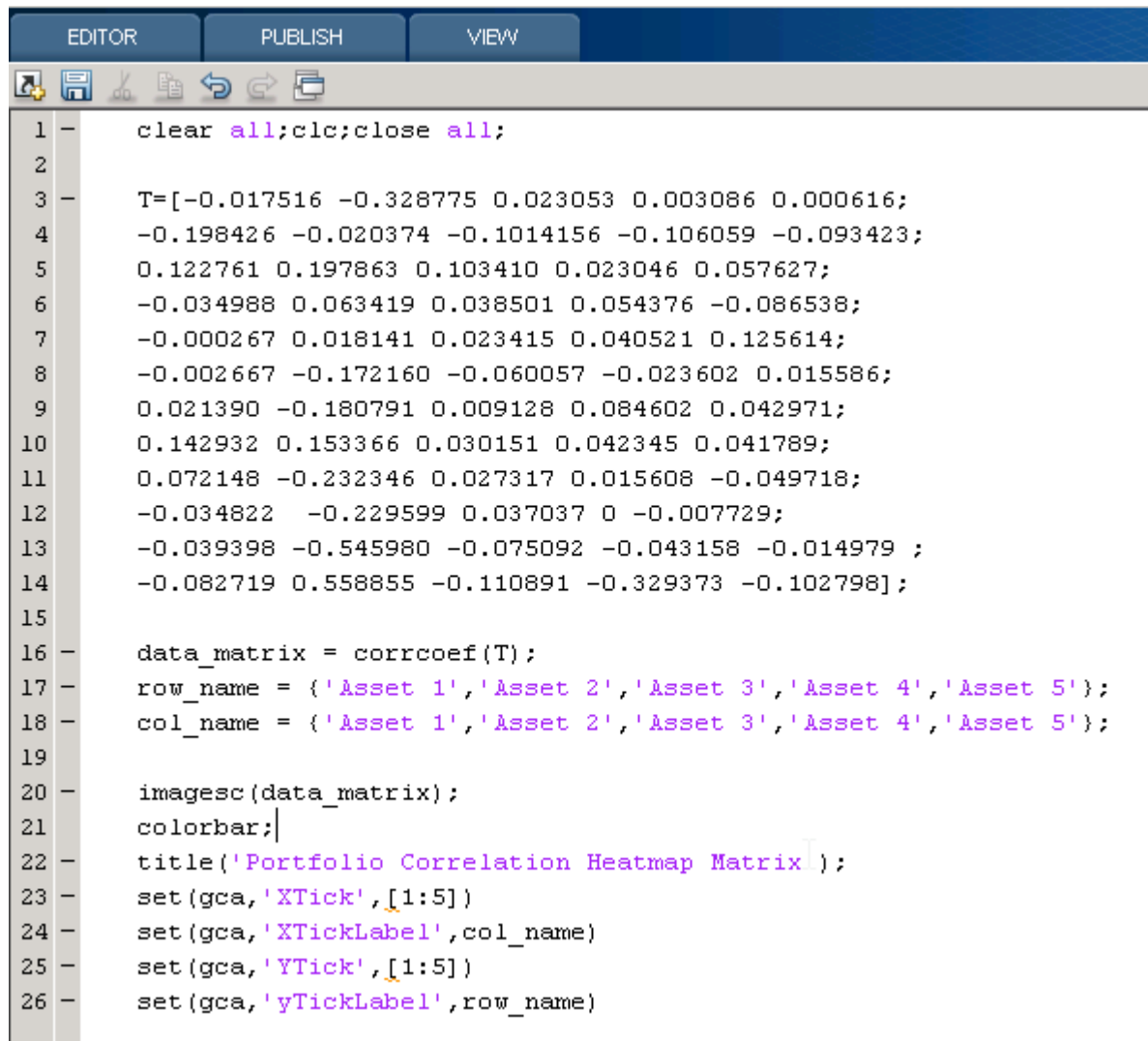
```

The result will be:



Correlation 2D Heatmap

This is a well know standard in Finance. We have to use a trick of image processing with the function `image()` to get the result we want but MATLAB is not as flexible as R for this type of job:

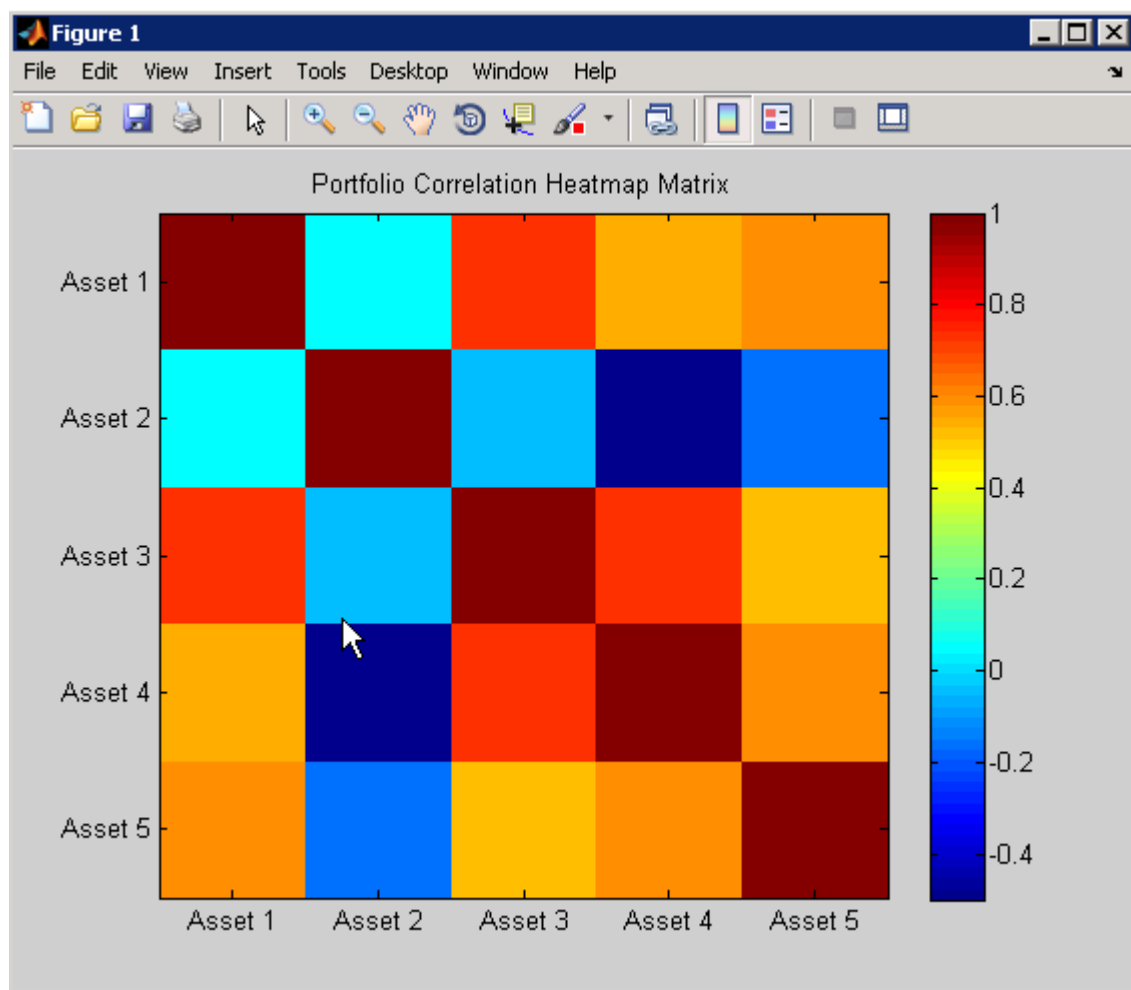


```

1 - clear all;clc;close all;
2
3 - T=[-0.017516 -0.328775 0.023053 0.003086 0.000616;
4 -   -0.198426 -0.020374 -0.1014156 -0.106059 -0.093423;
5 -   0.122761 0.197863 0.103410 0.023046 0.057627;
6 -   -0.034988 0.063419 0.038501 0.054376 -0.086538;
7 -   -0.000267 0.018141 0.023415 0.040521 0.125614;
8 -   -0.002667 -0.172160 -0.060057 -0.023602 0.015586;
9 -   0.021390 -0.180791 0.009128 0.084602 0.042971;
10 -  0.142932 0.153366 0.030151 0.042345 0.041789;
11 -  0.072148 -0.232346 0.027317 0.015608 -0.049718;
12 -  -0.034822 -0.229599 0.037037 0 -0.007729;
13 -  -0.039398 -0.545980 -0.075092 -0.043158 -0.014979 ;
14 -  -0.082719 0.558855 -0.110891 -0.329373 -0.102798];
15
16 - data_matrix = corrcoef(T);
17 - row_name = {'Asset 1','Asset 2','Asset 3','Asset 4','Asset 5'};
18 - col_name = {'Asset 1','Asset 2','Asset 3','Asset 4','Asset 5'};
19
20 - imagesc(data_matrix);
21 - colorbar;
22 - title('Portfolio Correlation Heatmap Matrix');
23 - set(gca,'XTick',[1:5])
24 - set(gca,'XTickLabel',col_name)
25 - set(gca,'YTick',[1:5])
26 - set(gca,'YTickLabel',row_name)

```

The result is on next page:



3D vertical bar charts

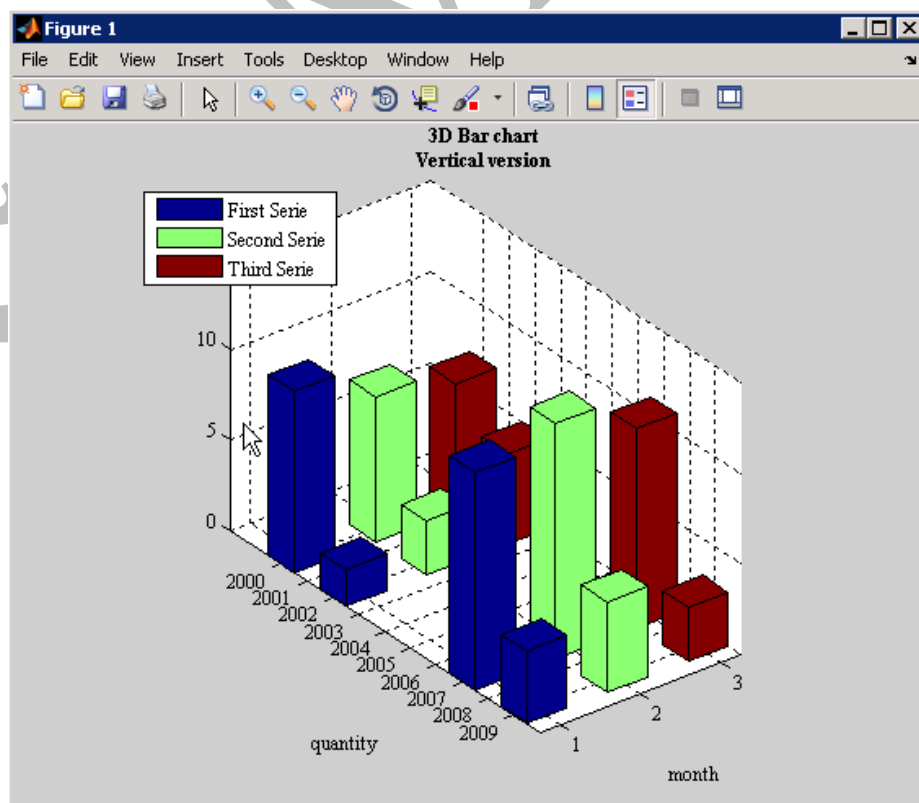
I'm not a fan of this type of chart then we will go quickly through. The idea is quite simple but like always, here we use the **bar3()** function and some tricks to manage ticks correctly with MATLAB (for sure... same can be done horizontally...):

```

1  x=[2 4 9 11];
2  mat=[10 8 7; 2 3 5; 12 13 11; 4 5 3];
3  bar3(x,mat,0.5,'detached');
4  set(gca, 'ytick', [1:1:10]);
5  set(gca, 'yticklabel', (2000:1:2009))
6  set(gca,'FontSize',9,'FontName','Times');
7  title({'3D Bar chart','Vertical version'},'FontWeight','bold');
8  xlabel('month');
9  ylabel('year');
10 ylabel('quantity');
11 legend('First Serie','Seconde Serie','Third Serie');
12 legend(gca,'location','NorthWest')
    rotate3d on
  
```

You can see also the rotate3d command to let the user rotate the chart even if no toolbars are available on the chart.

The result will be:



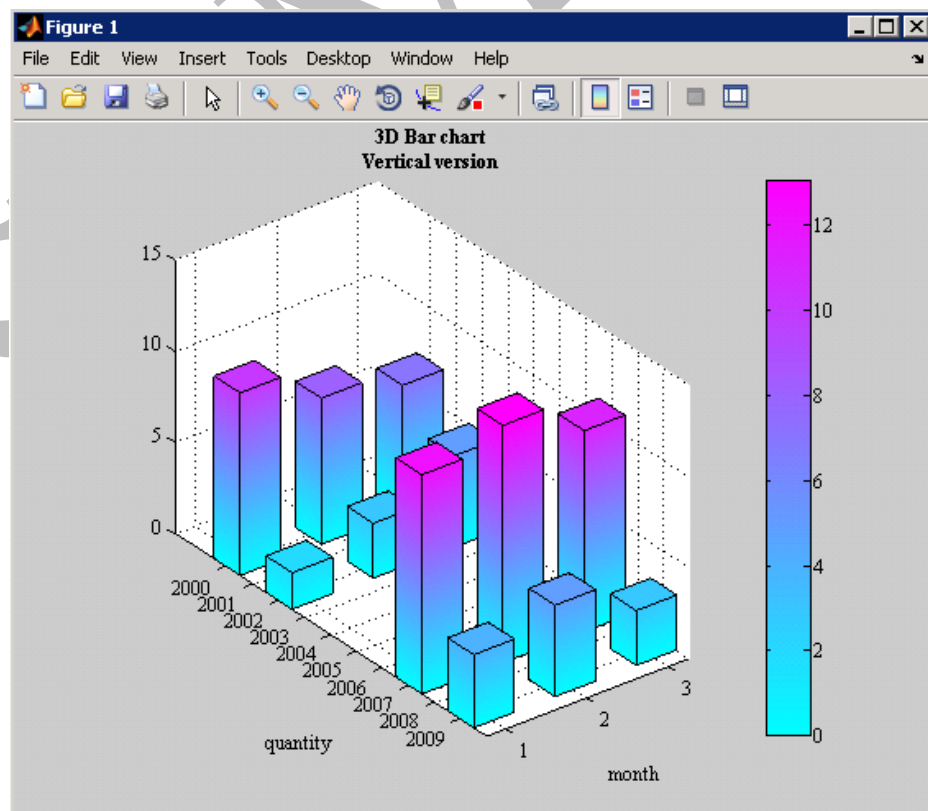
You can make the result a little bit more prettier by using a lot of MATLAB hidden tricks:

```

1      x=[2 4 9 11];
2      mat=[10 8 7; 2 3 5; 12 13 11; 4 5 3];
3      h=bar3(x,mat,0.5,'detached');
4      colormap cool
5      colorbar
6      shading interp
7      for i = 1:length(h)
8          zdata = get(h(i),'ZData');
9          set(h(i),'CData',zdata)
10         % Add back edge color removed by interpolating shading
11         set(h,'EdgeColor','k')
12     end
13     set(gca, 'ytick', [2:1:11]);
14     set(gca, 'yticklabel', (2000:1:2009))
15     set(gca,'FontSize',9,'FontName','Times');
16     title({'3D Bar chart','Vertical version'},'FontWeight','bold');
17     xlabel('month');
18     ylabel('year');
19     ylabel('quantity');

```

The result will be:



2D/3D sector charts (cheese)

Ok another typical type of chart that is not critical but I strongly recommend you to go on the website of MATLAB file exchange to take a look to the free beautiful 3D sector chart Toolbox.

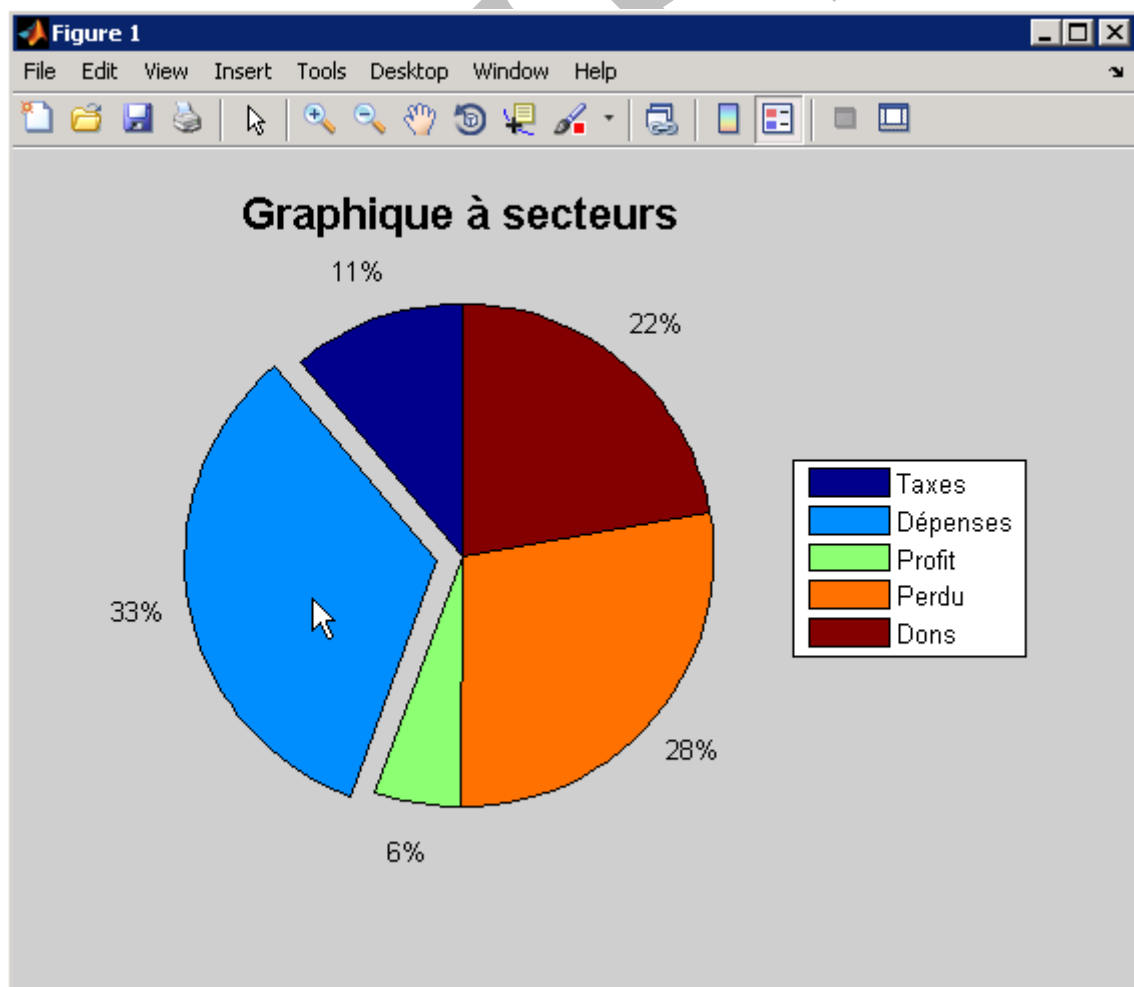
But here see what are the basics to know first using the function **pie()** and the trick to add the values to the different sectors:

```

EDITOR PUBLISH VIEW
1 x = [1 3 0.5 2.5 2];
2 labels = ('Taxes', 'Dépenses', 'Profit', 'Perdu', 'Dons');
3 explode = [0 1 0 0 0];
4 h=pie(x,explode);
5 legend(findobj(h, 'type','patch'), labels, 'Location','EastOutside')
6 colormap jet;
7 title('Graphique à secteurs','fontweight','bold','fontsize',16);

```

The result is:



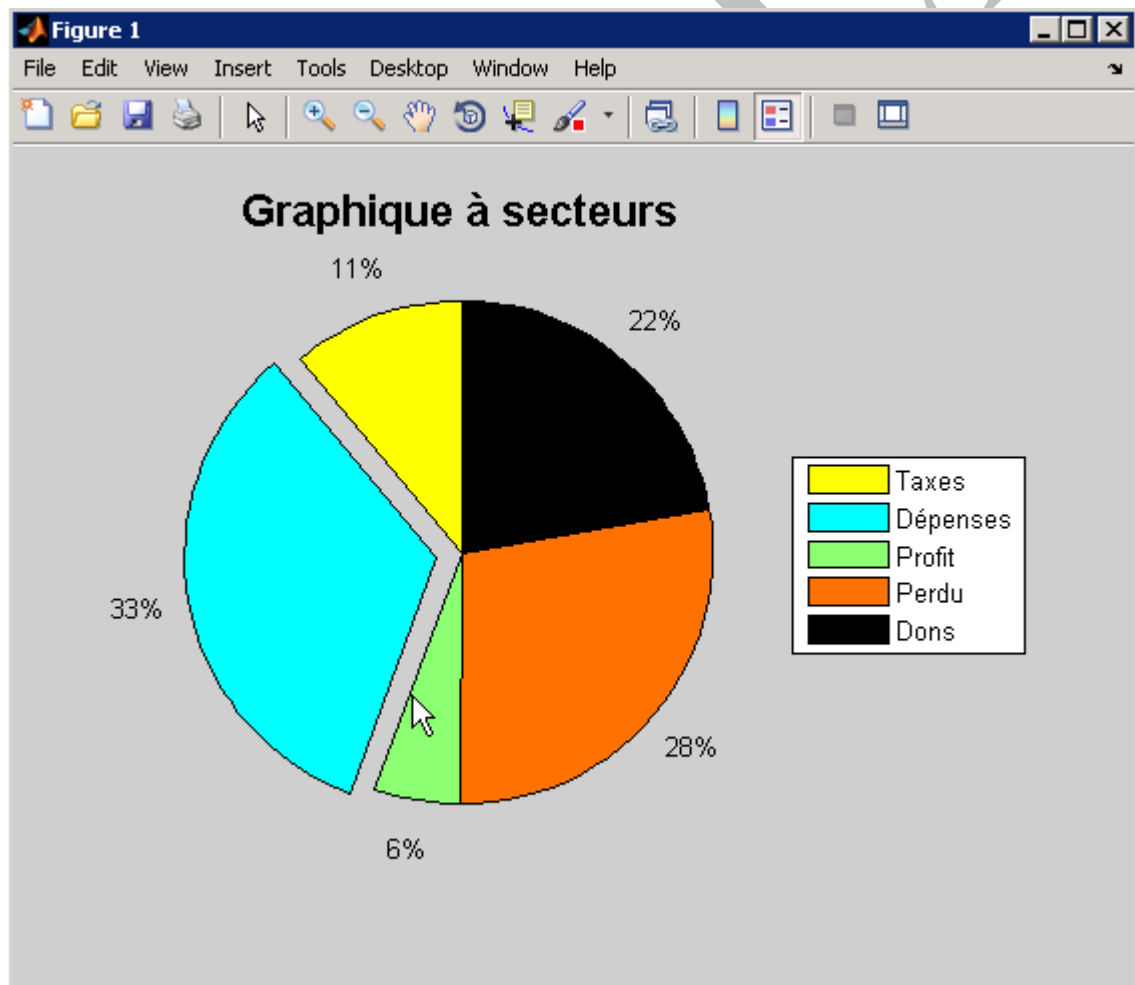
Or if we want to control the color for each sector part this is once again difficult to guess:

```

1  clear all;clc;close all
2  x = [1 3 0.5 2.5 2]; %5 values!
3  labels = {'Taxes','Dépenses','Profit','Perdu','Dons'};
4  explode = [0 1 0 0 0];
5  h=pie(x,explode);
6  legend(findobj(h, 'type','patch'), labels, 'Location','EastOutside')
7  %the h vector contains 10 components (2 times 5!)
8  set(h(1),'FaceColor','y') %this is taxis (1-2)
9  set(h(3),'FaceColor','c') %this is Dépenses (3-4)
10 set(h(9),'FaceColor','k') %this is Dons (9-10)
11 title('Graphique à secteurs','fontweight','bold','fontsize',16);

```

The result is:



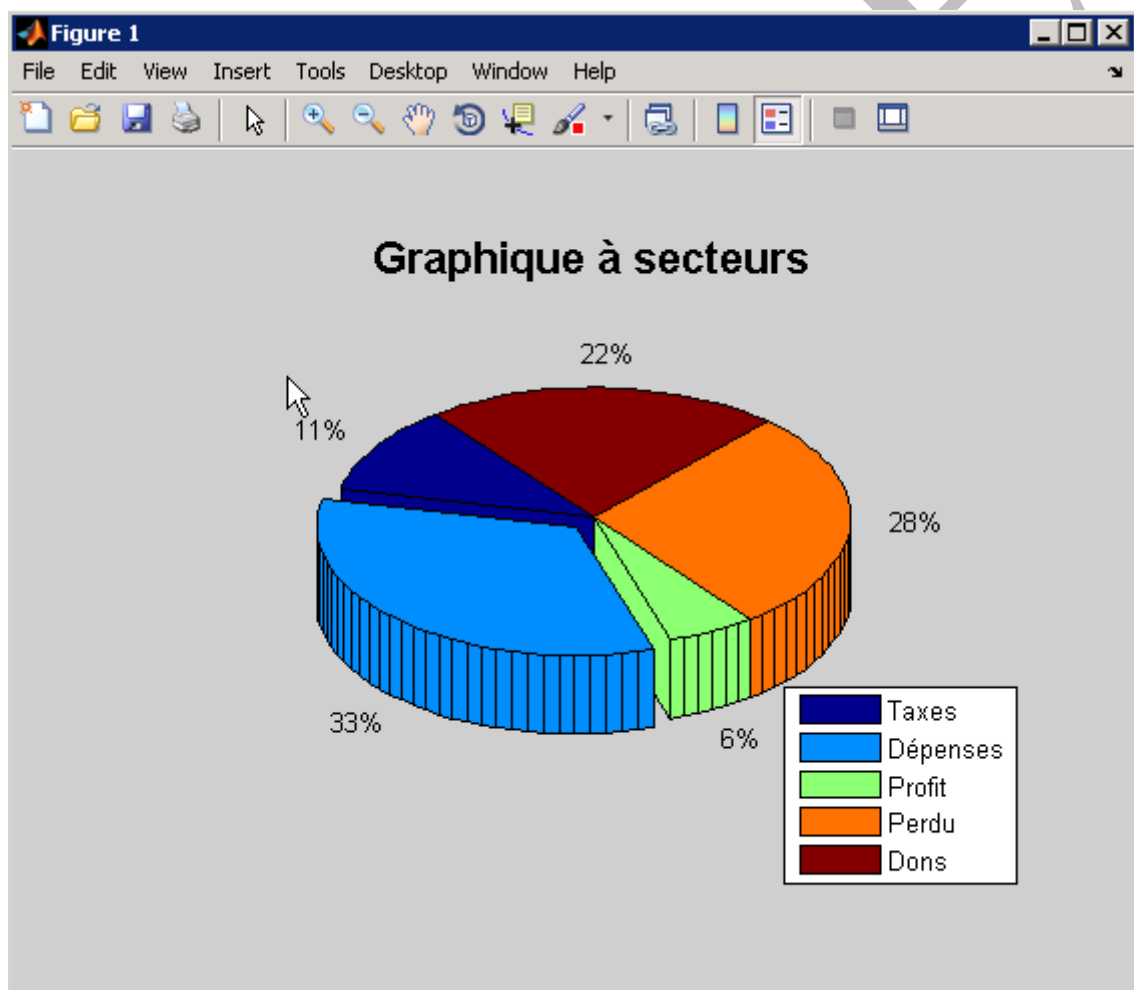
And in 3D with the function **pie3()**:

```

1 - close all;clear all;clc
2 - explode=[0 1 0 0 0];
3 - labels={'Taxes','Dépenses','Profit','Perdu','Dons'};
4 - x=[1 3 0.5 2.5 2];
5 - pie3(x,explode);
6 - legend(labels,'Location','SouthEast');
7 - title('Graphique à secteurs','fontweight','bold','fontsize',16);

```

This will give:



2D Surface/Area charts

Area charts are not used a lot as they are in a first place but they are used a lot in Finance and Econometrics to create custom time series confidence interval (the default one of MATLAB or almost horrible and a too limited).

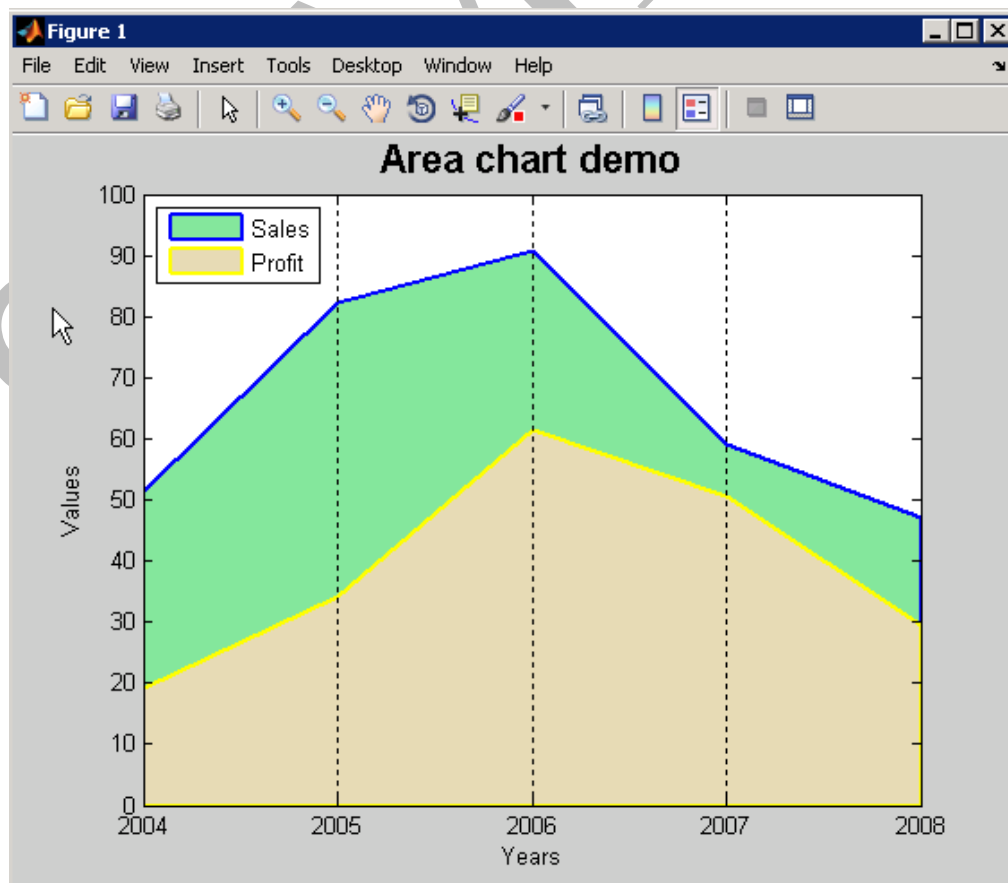
First let us see a very basic example:

```

1  clear all;close all;clc
2  x = 2004:2008;
3  sales = [51.6 82.4 90.8 59.1 47.0];
4  profits = [19.3 34.2 61.4 50.5 29.4];
5  area(x,sales,'FaceColor',[.5 .9 .6],'EdgeColor','b','LineWidth',2);
6  hold on;
7  area(x,profits,'FaceColor',[.9 .85 .7],'EdgeColor','y','LineWidth',2);
8  legend({'Sales','Profit'}, 'Location','NorthWest');
9  set(gca,'XTick',x);
10 set(gca,'XGrid','on');
11 set(gca,'Layer','top');
12 title('Area chart demo','fontWeight','bold','fontSize',16);
13 xlabel('Years');ylabel('Values');

```

The result will be:



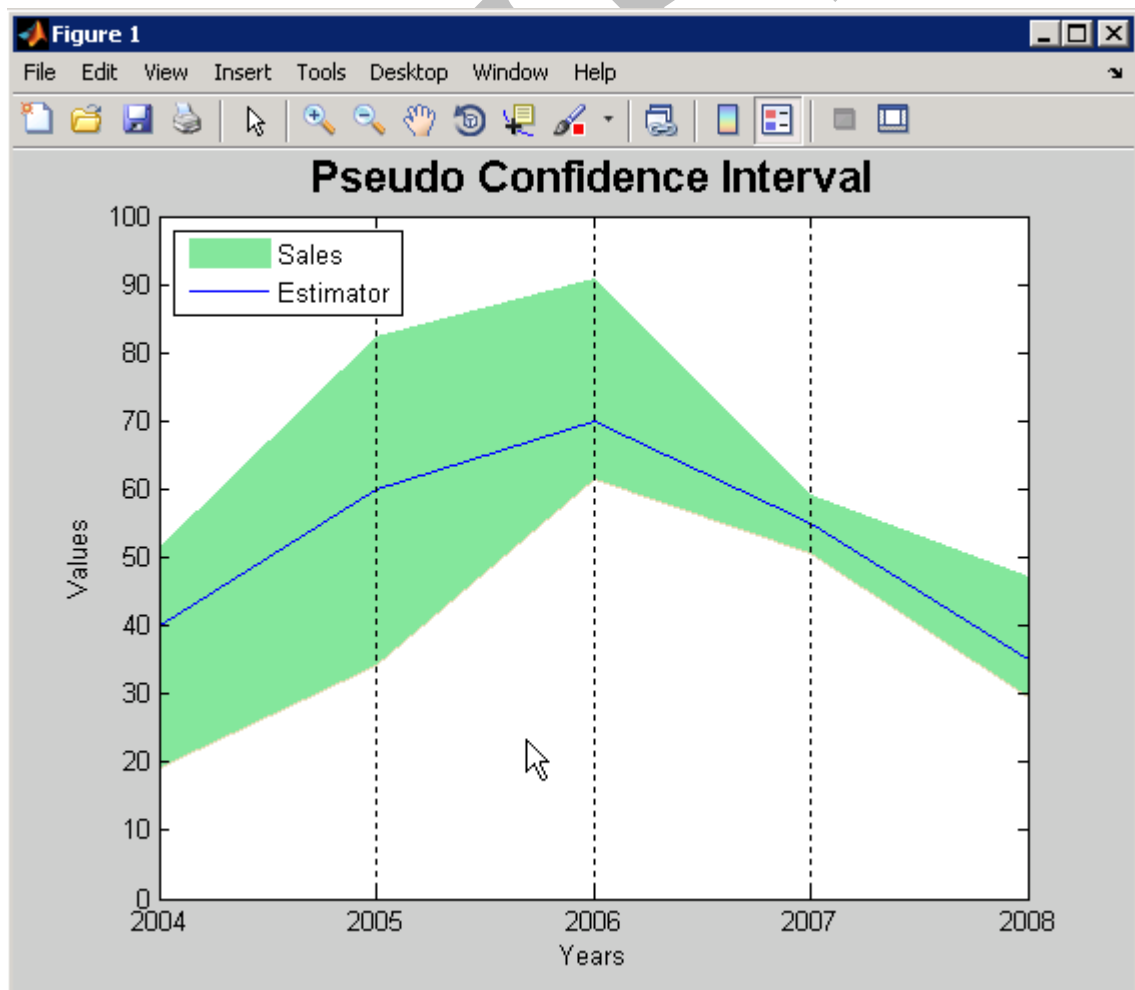
Now a much more interesting usage of area charts is to use them as pseudo-confidence interval. Here is a small example of the trick that mixed the function **area()** and **plot()** that you can complicate much more with multiple confidence intervals not just one!

```

1  x = 2004:2008;
2  sales = [51.6 82.4 90.8 59.1 47.0];
3  profits = [19.3 34.2 61.4 50.5 29.4];
4  forecast = [40 60 70 55 35];
5  h1=area(x,sales,'FaceColor',[.5 .9 .6],'EdgeColor',[.5 .9 .6]);
6  hold on;
7  h2=area(x,profits,'FaceColor',[1 1 1],'EdgeColor',[.9 .85 .7]);
8  h3=plot(x,forecast);
9  legend([h1 h3],{'Sales','Estimator'}, 'Location', 'NorthWest');
10 set(gca,'XTick',x);
11 set(gca,'XGrid','on');
12 set(gca,'Layer','top');
13 title('Pseudo Confidence Interval','fontweight','bold','fontsize',16);
14 xlabel('Years');ylabel('Values');

```

The result will be:



Radial (rose) bar chart/arrow (compass) chart

The radial bar chart is especially used in spatial statistics and electronic.

Caution! Even if the **rose()** function give as output a chart in degrees, the input vector must be in radians!

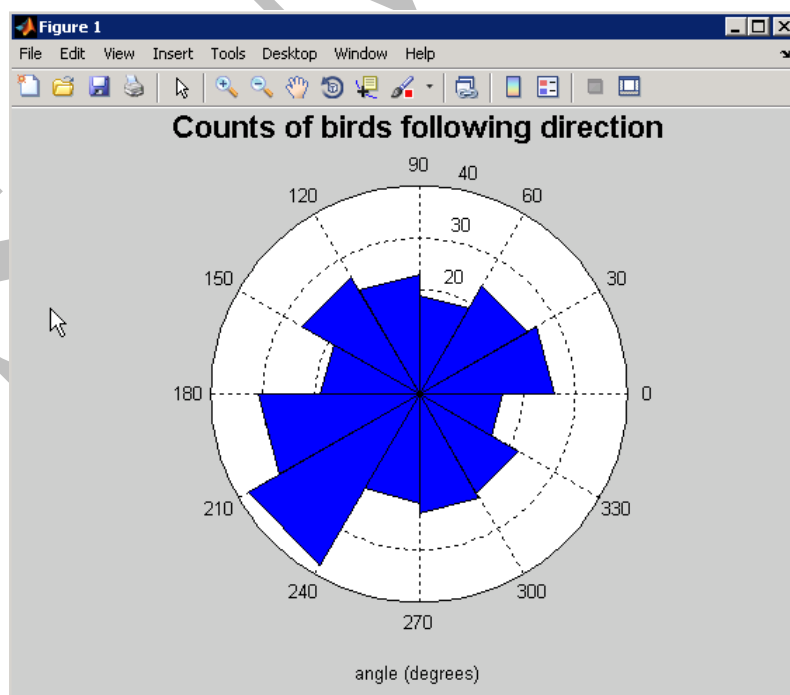
Even if the chat below is a simple example, it is false because we did not convert the angles in radians... but that doesn't really matter because the purpose is just to see the syntax of the function **rose()**:

```

1 clear all;close all;clc
2 load sunspot.dat
3 %we take the second column of the dataset that contains the counts
4 theta = sunspot(:,2);
5 %we divide by 12 bins (360°/12=30°)
6 rose_plot=rose(theta,12);
7 title('Counts of birds following direction','fontweight',...
8       'bold','fontsize',16)
9 xlabel('angle (degrees)');
10 %we change the colors of the bars
11 x = get(rose_plot,'XData') ;
12 y = get(rose_plot,'YData') ;
13 rose_plot = patch(x, y, 'b');

```

The result is:



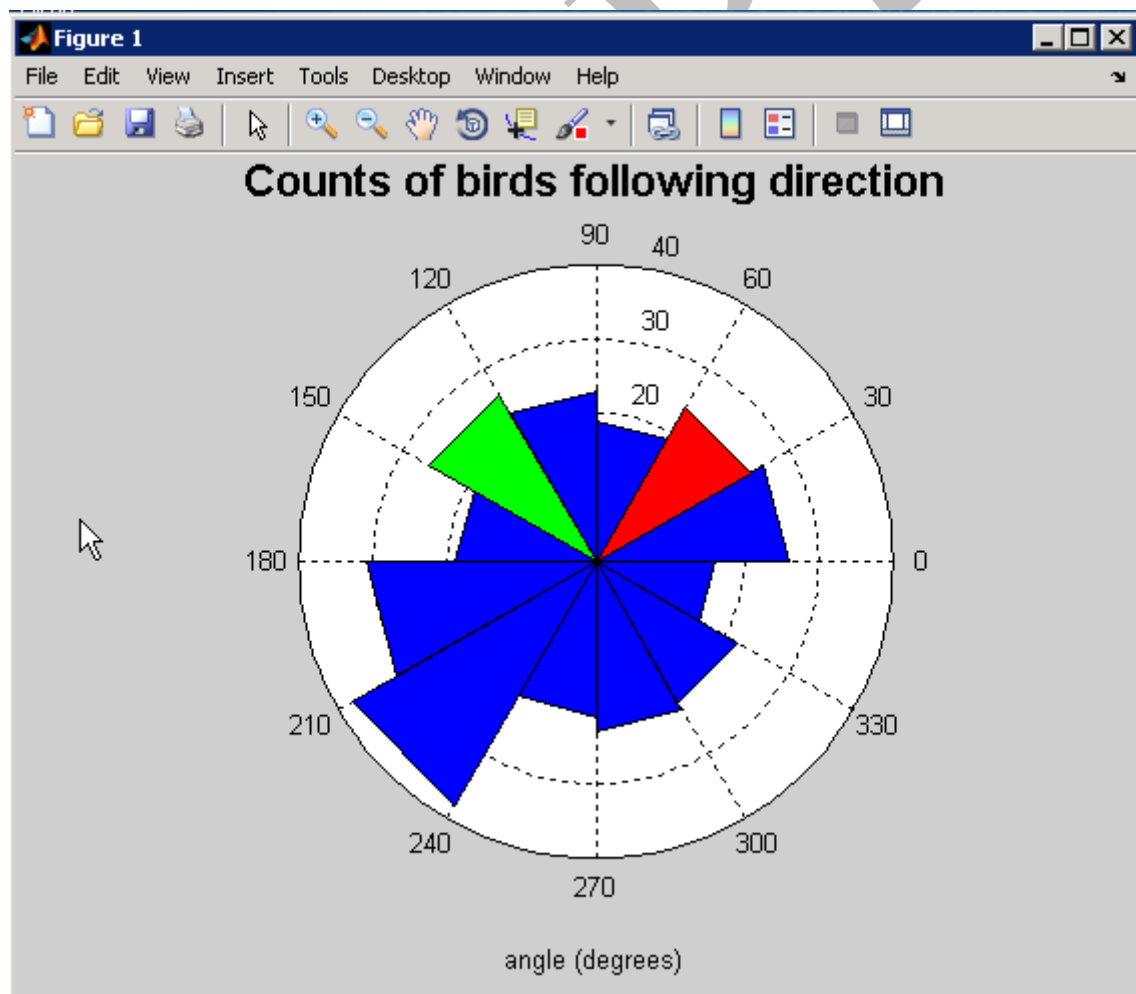
And because each vector has 48 components that mean that each triangle is in reality a rectangle with 4 coordinates. Thus to change the color of only one rectangle we have to play a little bit:

```

EDITOR PUBLISH VIEW
load sunspot.dat
theta=sunspot(:,2);
rose_plot=rose(theta,12);
title('Counts of birds following direction','fontweight',...
      'bold','fontsize',16)
xlabel('angle (degrees)')
x=get(rose_plot,'XData');
y=get(rose_plot,'YData');
rose_plot=patch(x,y,'b');
rose_plot=patch(x(5:9),y(5:9),'r');
rose_plot=patch(x(16:20),y(16:20),'g');

```

The result will be:



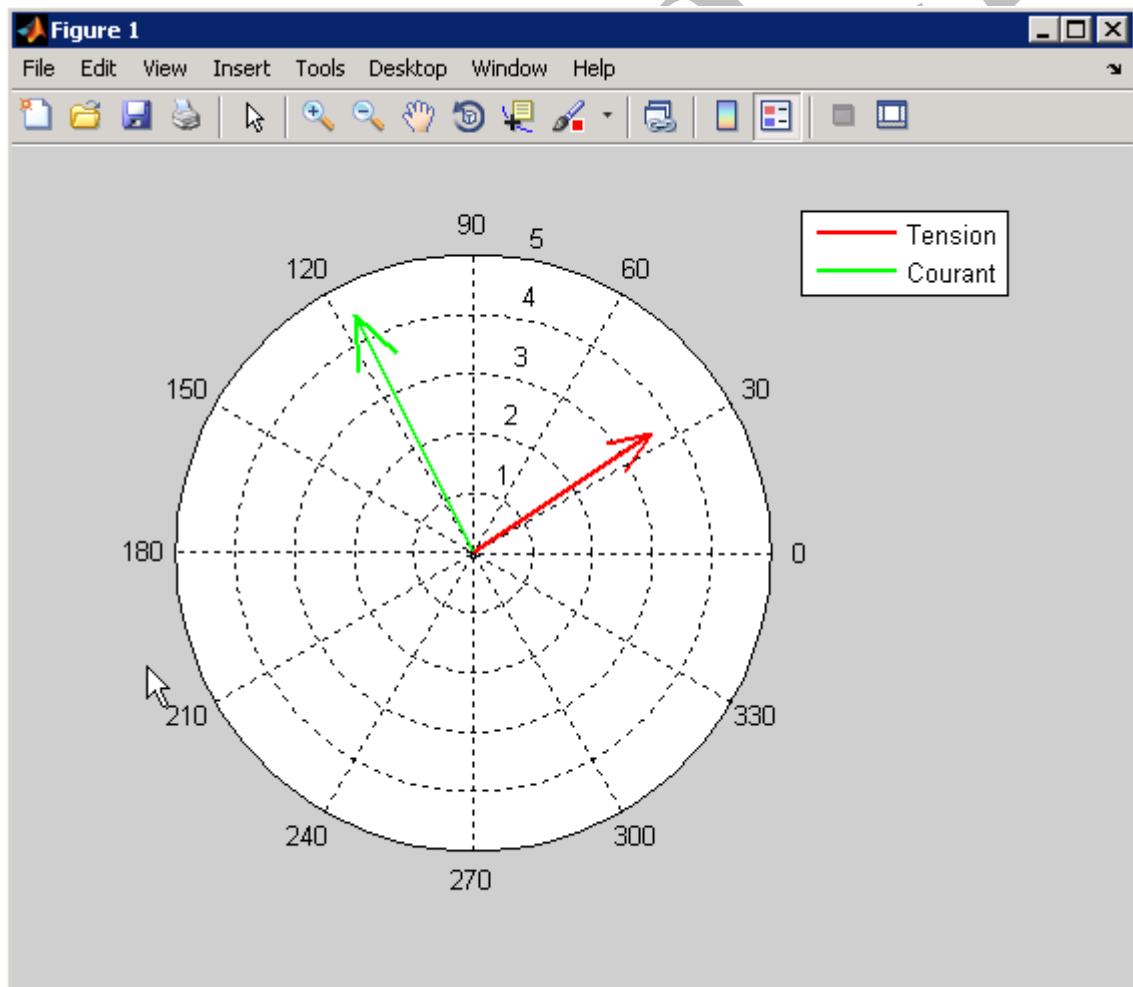
And now for electronic engineers see how we can plot phase vectors using the **compass()** function and some tricks to change the color of the arrows...:

```

EDITOR PUBLISH VIEW
1 close all;clear all;clc
2 Z=[3+2*1i;-2+4*1i];
3 color = [1 0 0;0 1 0]; %red and green
4 h=compass(Z);
5 set(h,{'Color'},num2cell(color,2),'LineWidth',2);
6 str={'Tension' 'Courant'};
7 legend(h,str,'Location','BestOutside');

```

The result will be:



Pareto chart

Pareto charts in MATLAB are very limited and basics as in most Statistical softwares by default. Then you will have to do a lot of additions to get something professional.

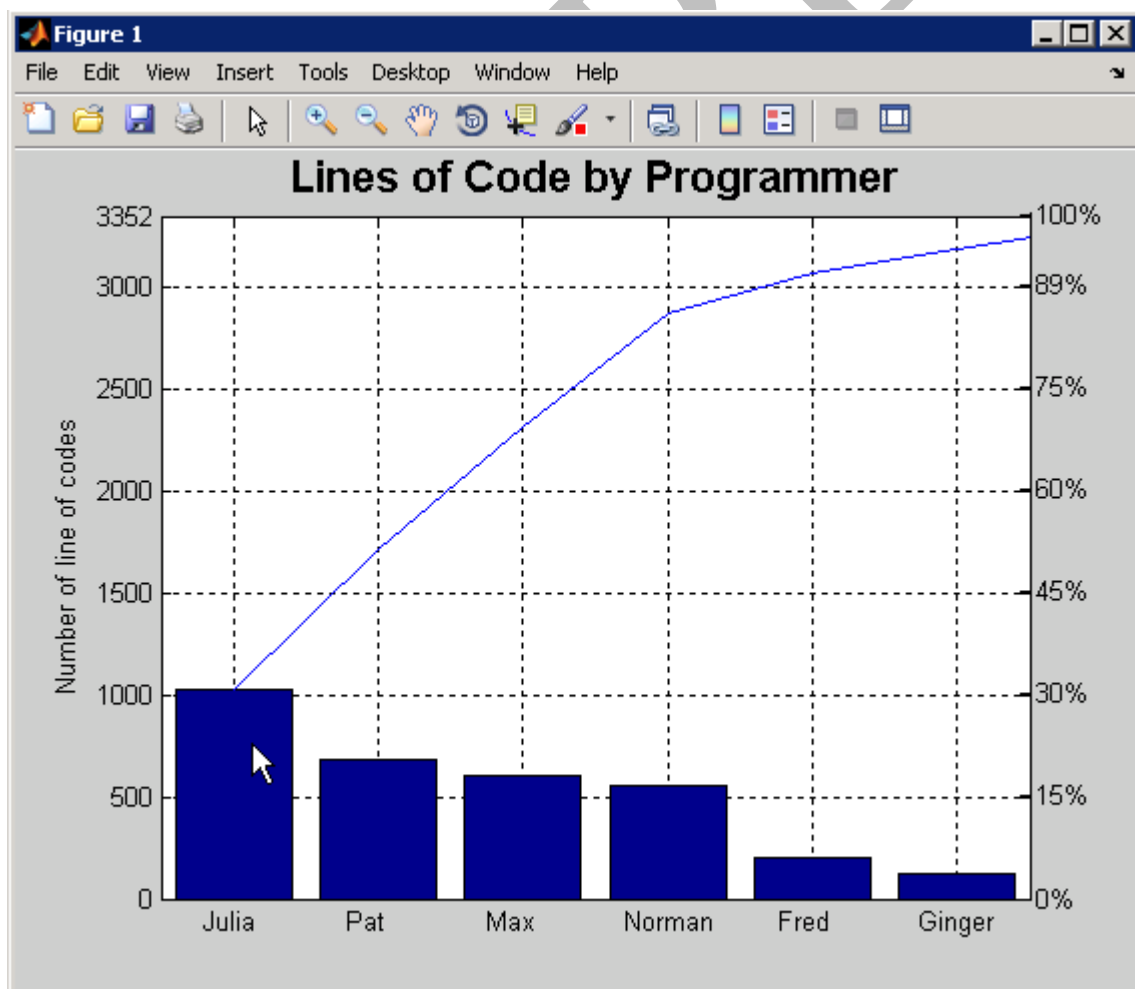
But let us see the default possibilities of MATLAB about this type of chart using the function **pareto()**:

```

1  codelines = [200 120 555 608 1024 101 57 687];
2  coders = {'Fred','Ginger','Norman','Max','Julia','Wally','Heidi','Pat'};
3  pareto(codelines, coders)
4  title('Lines of Code by Programmer','fontweight',...
5        'bold','fontsize',16)
6  ylabel('Number of line of codes');
7  grid on

```

And the result will be...:



Plotmatrix

Matrix plots are very disappointing in MATLAB in comparison of R... The only one thing that we can do (as far as I know) are histograms on the diagonal and dot charts otherwise.

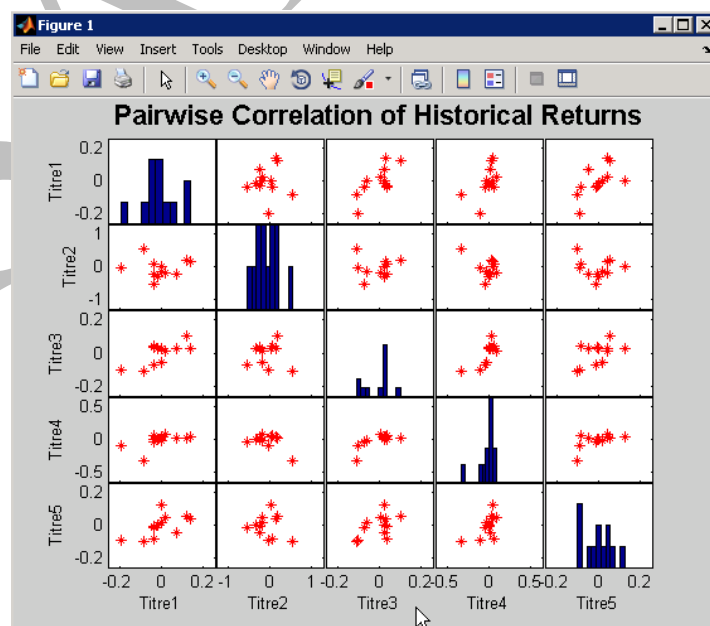
Let us see this with our famous small dataset of 5 assets and by using the function **plotmatrix()**:

```

1  T=[-0.017516 -0.328775 0.023053 0.003086 0.000616;
2  -0.198426 -0.020374 -0.1014156 -0.106059 -0.093423;
3  0.122761 0.197863 0.103410 0.023046 0.057627;
4  -0.034988 0.063419 0.038501 0.054376 -0.086538;
5  -0.000267 0.018141 0.023415 0.040521 0.125614;
6  -0.002667 -0.172160 -0.060057 -0.023602 0.015586;
7  0.021390 -0.180791 0.009128 0.084602 0.042971;
8  0.142932 0.153366 0.030151 0.042345 0.041789;
9  0.072148 -0.232346 0.027317 0.015608 -0.049718;
10 -0.034822 -0.229599 0.037037 0 -0.007729;
11 -0.039398 -0.545980 -0.075092 -0.043158 -0.014979 ;
12 -0.082719 0.558855 -0.110891 -0.329373 -0.102798];
13 [~,ax]=plotmatrix(T,'*r');
14 title('Pairwise Correlation of Historical Returns','fontweight',...
15       'bold','fontsize',16);
16 names={'Titre1', 'Titre2', 'Titre3', 'Titre4', 'Titre5'};
17 for i = 1:length(names)
18     ylabel(ax(i,1),names{i});
19     xlabel(ax(end,i),names{i});
20 end

```

The result will be:



3D parametric plots

3D parametric plots are especially used by engineers that work on spatial rockets or with drones. We can see also some biologists plotting the trajectory of a tracked bird. Otherwise it is more a plot family use in academics fields for pedagogical purposes.

Vector based 3D parametric line plot

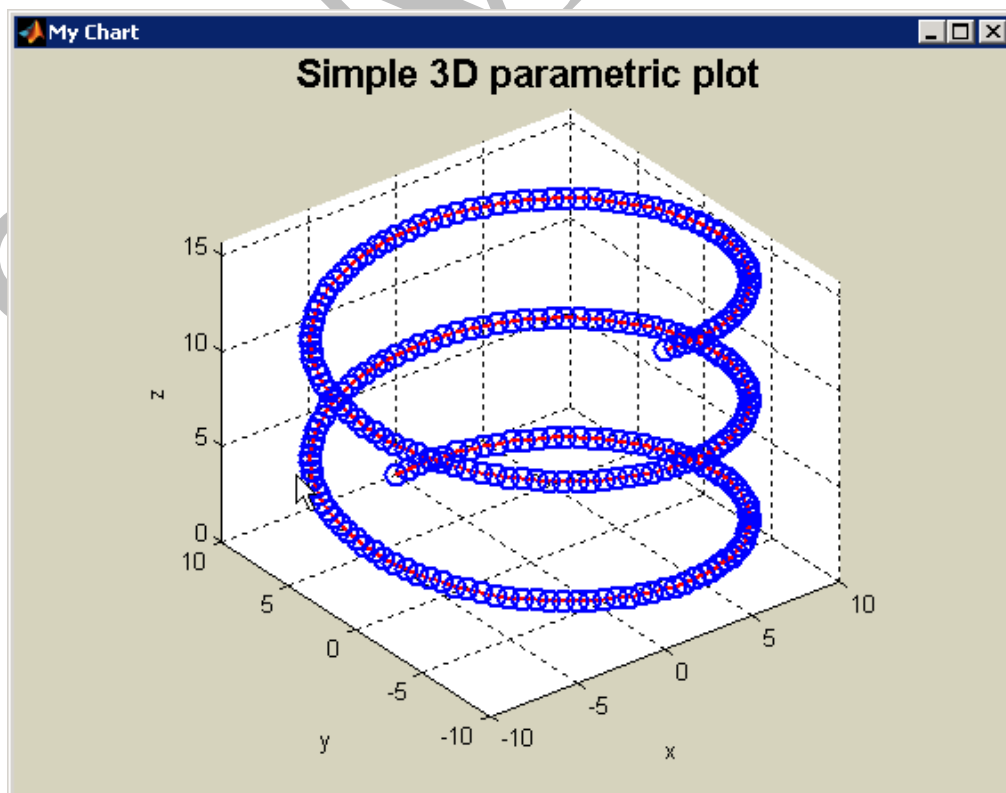
Let us begin with the function **plot3()** that can be useful to plot a parametric line in space with or without points visible on the line when you have vector of points in the time:

```

1  clear all;close all;clc
2  figure;
3  set(gcf,'Menubar','none','Name','My Chart','NumberTitle','off',...
4    'Color',[0.8314 0.8157 0.7483]);
5  t = 0:pi/50:5*pi;
6  st = 10*sin(t);
7  ct = 10*cos(t);
8  plot3(st,ct,t,'-ro','LineWidth',2,'Markersize',10,'MarkerEdgeColor','b')
9  grid on;
10 xlabel('x');ylabel('y');zlabel('z')
11 title('Simple 3D parametric plot','fontweight','bold','fontsize',16);
12 axis equal;

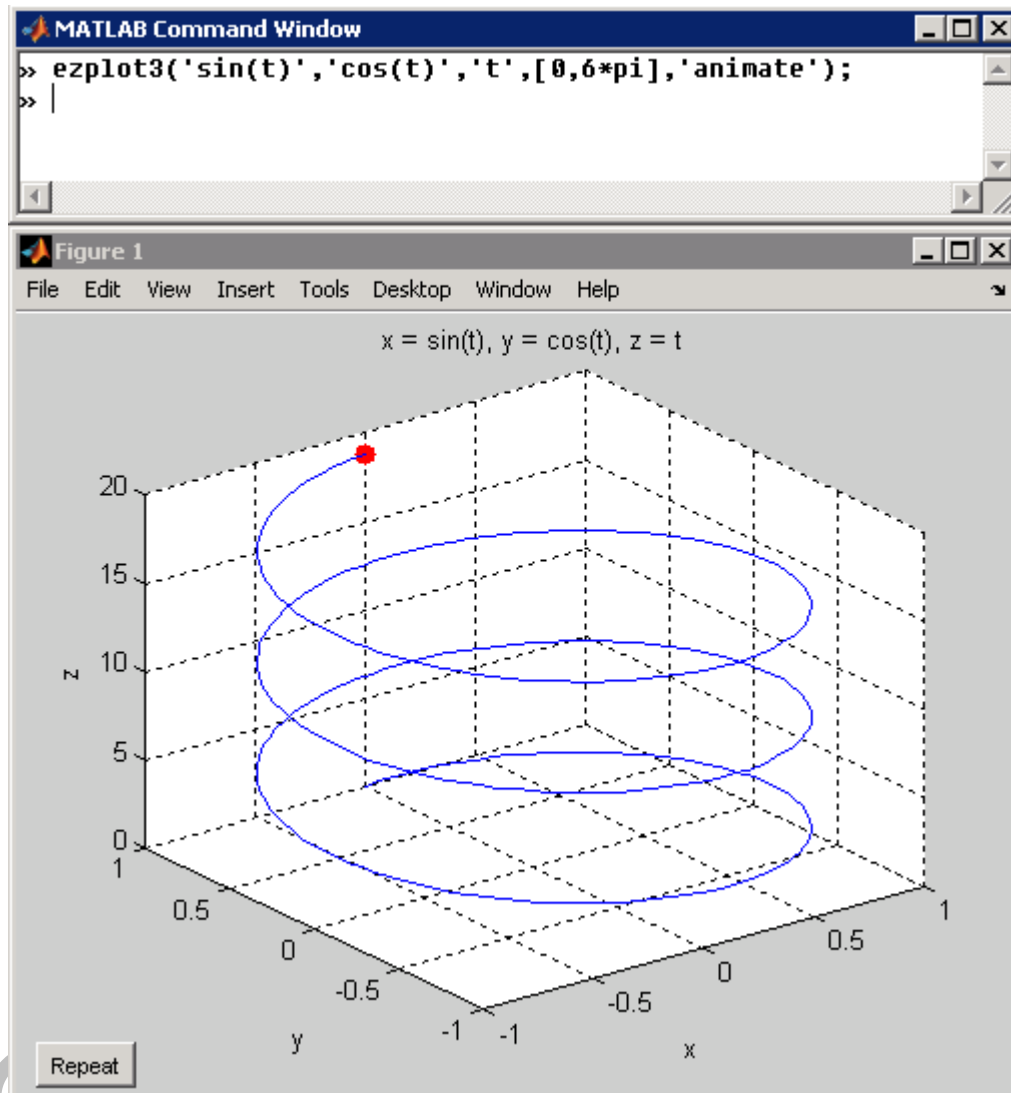
```

The result will be:



Animated or non-animated analytic based 3D parametric line plot

If you want to animate an analytical parametric plot expression (typical in academics classes) you can easily do it by using the function **ezplot3()**:



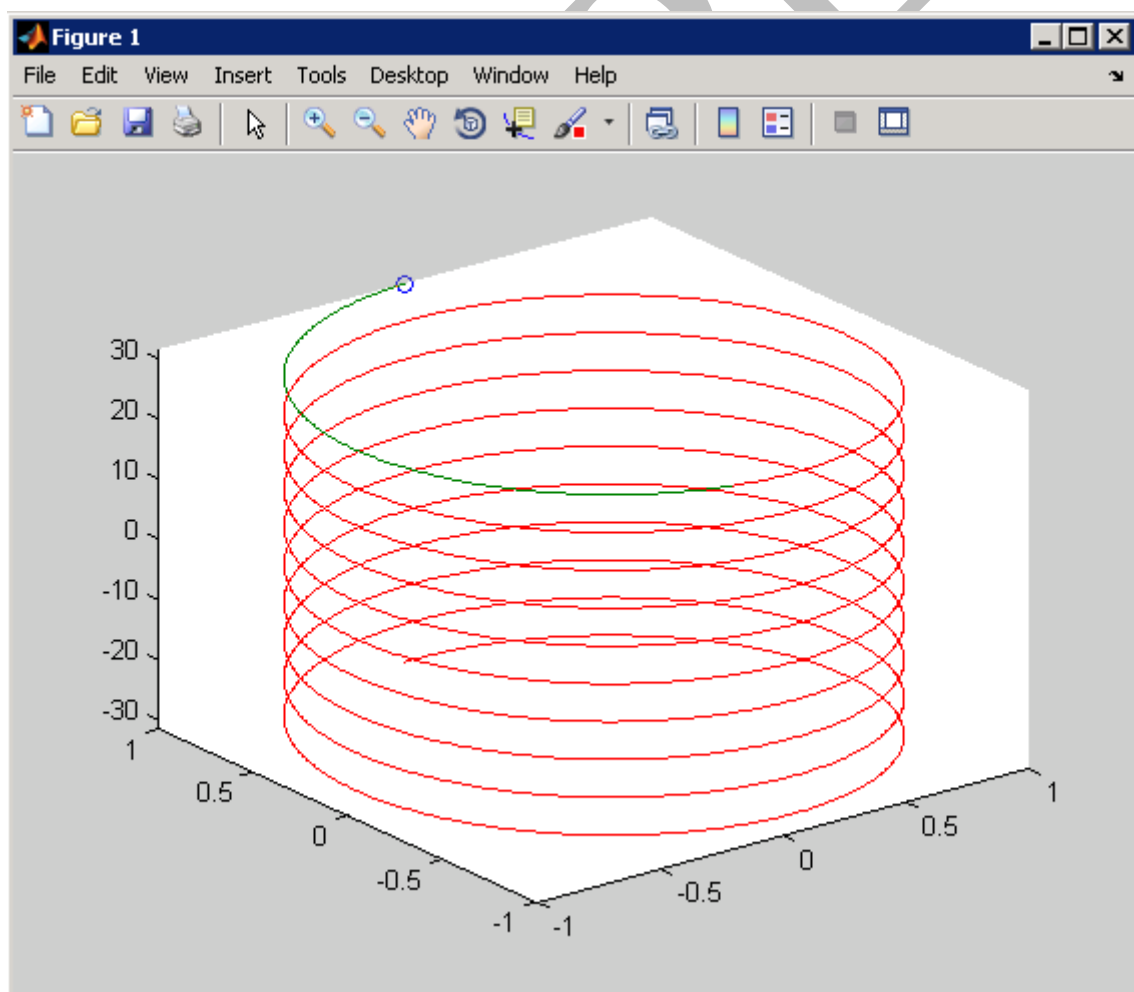
See the **Repeat** button on the bottom left corner!

Animated vector based 3D parametric line plot

If you don't have an analytical expression but only vector with the points of evolution in space-time the **comet3()** function can show you the animated plot by doing a linear interpolation (when possible...) between each for the fluidity of the animation:

```
EDITOR PUBLISH VIEW
1 %same thing with comet3 but it use a linear interpolation
2 %and we can see that by increasing the step
3 t = -10*pi:pi/250:10*pi;
4 comet3(sin(t),cos(t),t);
5 %axis and title will appear after animation
6 xlabel('x');ylabel('y');zlabel('z')
7 title('x=sin(t),y=cos(t),z=t');
```

The result will be:

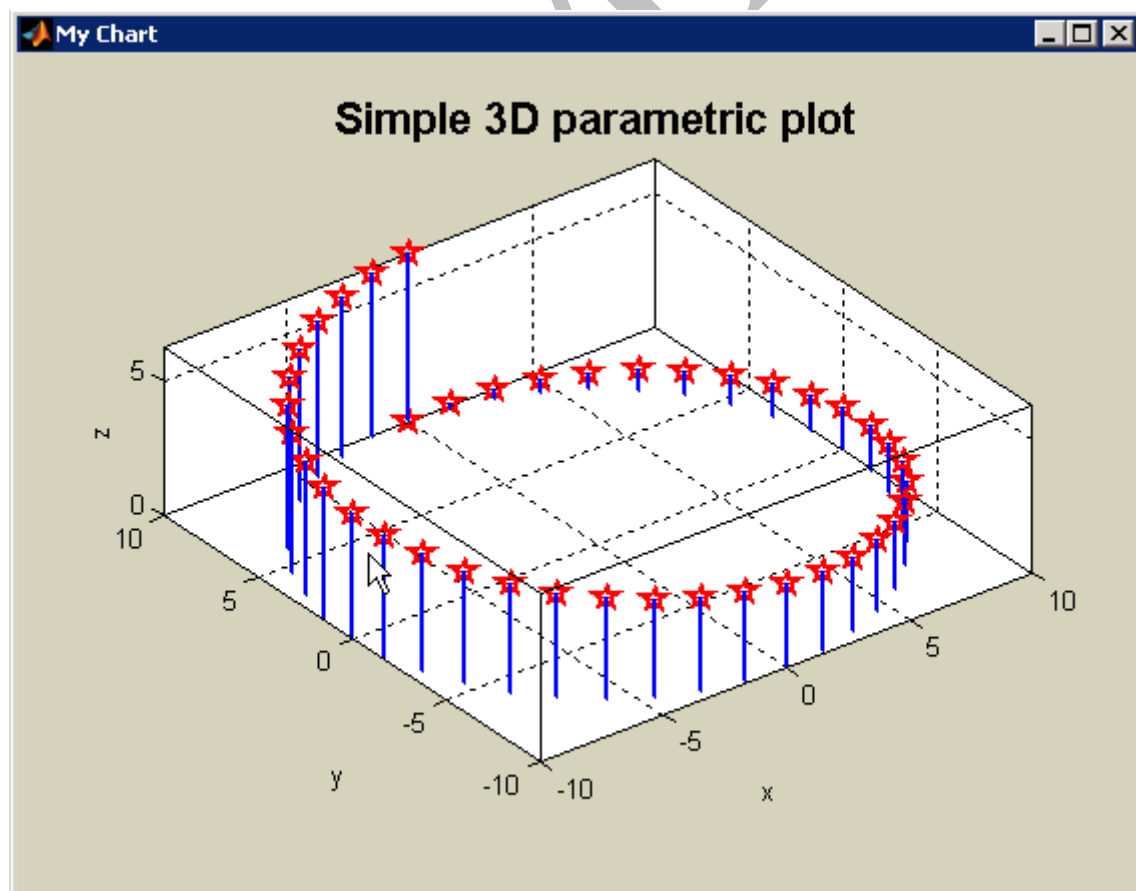


Vector based 3D stem parametric line plot

This plot is especially used in rocket science to see over what the rocket was on the ground (or on the water in most cases...) on in GPS tracking. For this purpose you use the function `stem3()`:

```
1 clear all;close all;clc
2 figure;
3 set(gcf,'Menubar','none','Name','My Chart','NumberTitle','off',...
4     'Color',[0.8314 0.8157 0.7483]);
5 t = 0:pi/20:2*pi;
6 st = 10*sin(t);
7 ct = 10*cos(t);
8 stem3(st,ct,t,'bp','LineWidth',2,'Markersize',10,'MarkerEdgeColor','r')
9 grid on;
10 xlabel('x');ylabel('y');zlabel('z')
11 title('Simple 3D parametric plot','fontweight','bold','fontsize',16);
12 axis equal;
```

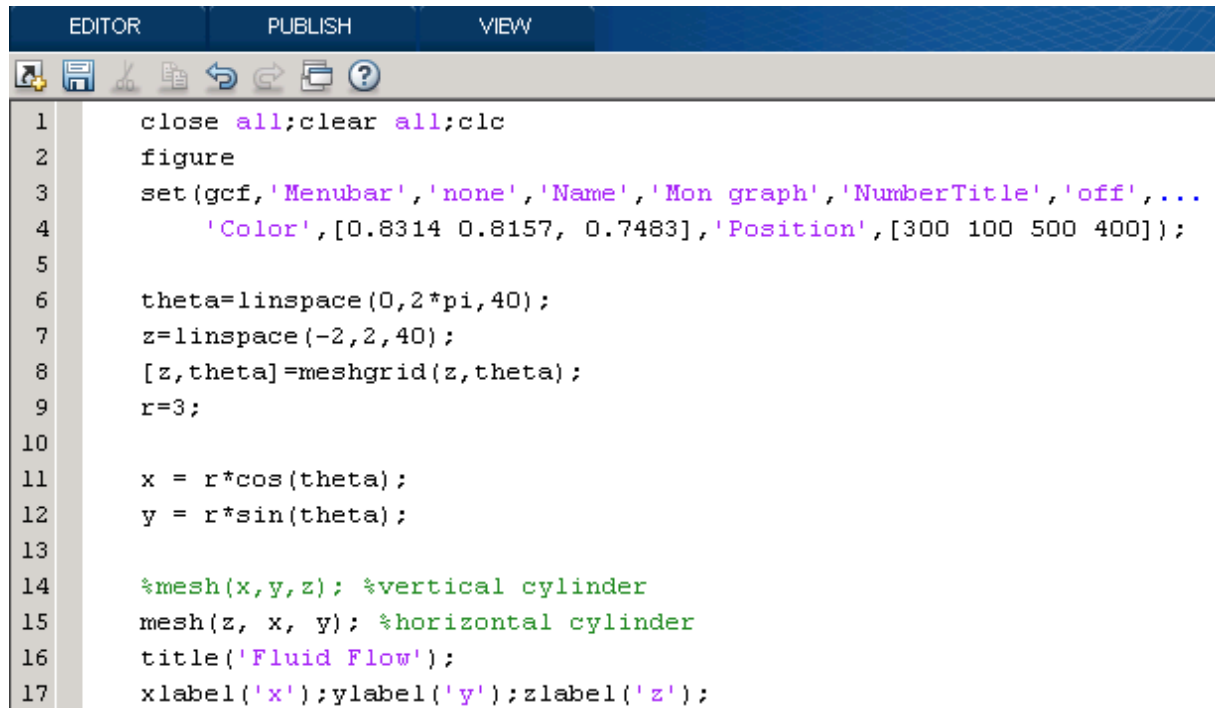
The result will look like this (ToDo: I still have to add an image on the ground when I will have the time):



Vector based 3D surface parametric plot

This type of plots is mainly academics but I saw them sometimes used in General Relativity (but still for academic pedagogical purposes).

Let us begin with the most simple parametric plot (a cylinder) to deeply understand how to manipulate the **meshgrid()** and **mesh()** functions that are also sometimes used in Finance:

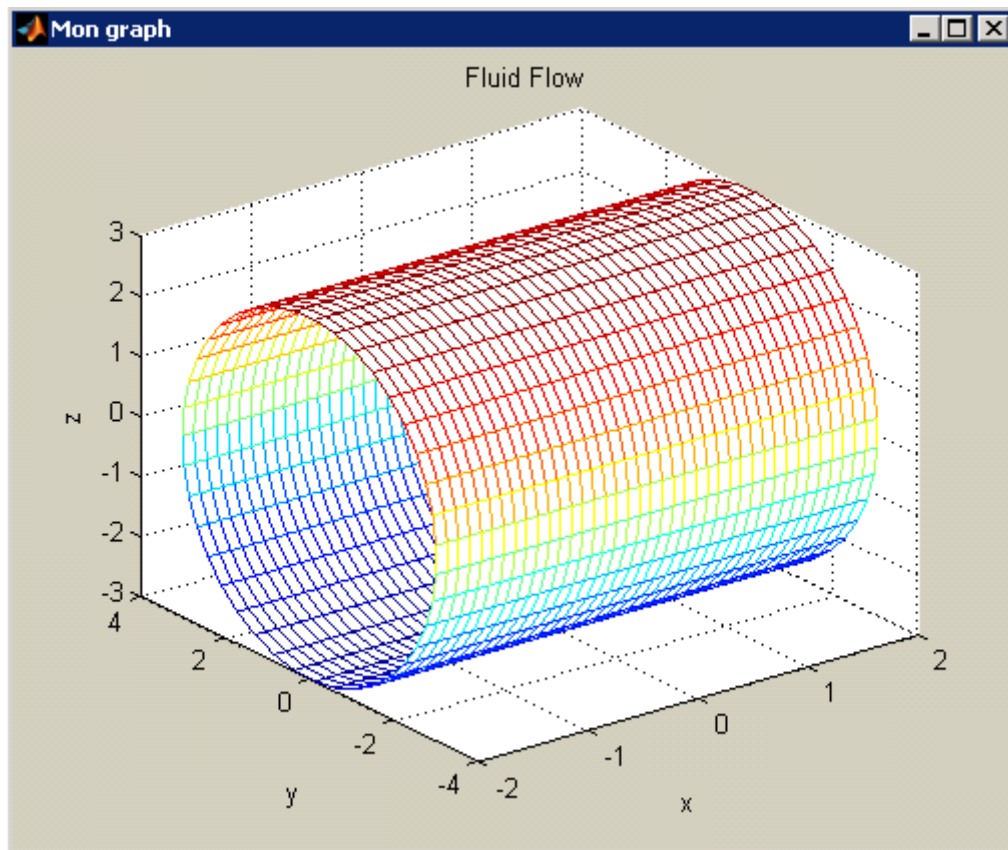


```

1      close all;clear all;clc
2      figure
3      set(gcf,'Menubar','none','Name','Mon graph','NumberTitle','off',...
4          'Color',[0.8314 0.8157, 0.7483],'Position',[300 100 500 400]);
5
6      theta=linspace(0,2*pi,40);
7      z=linspace(-2,2,40);
8      [z,theta]=meshgrid(z,theta);
9      r=3;
10
11      x = r*cos(theta);
12      y = r*sin(theta);
13
14      %mesh(x,y,z); %vertical cylinder
15      mesh(z, x, y); %horizontal cylinder
16      title('Fluid Flow');
17      xlabel('x');ylabel('y');zlabel('z');

```

That will give:



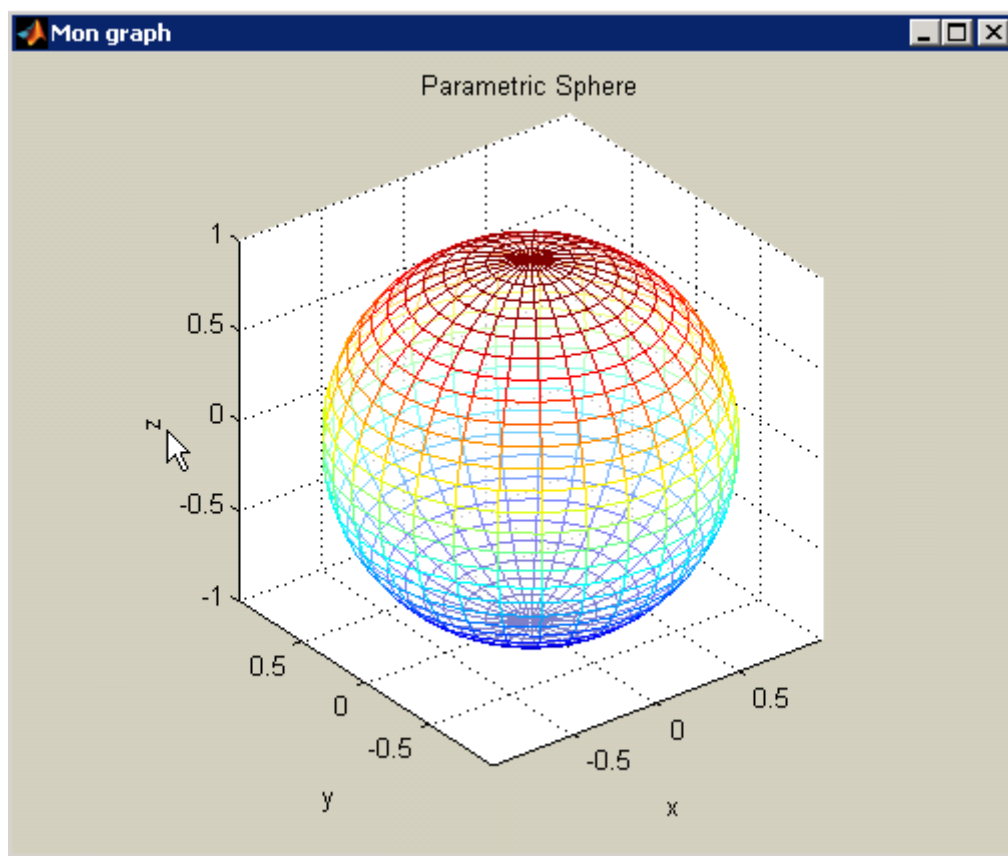
And what about a simple sphere with transparency on the surface to be able to see trough:

```

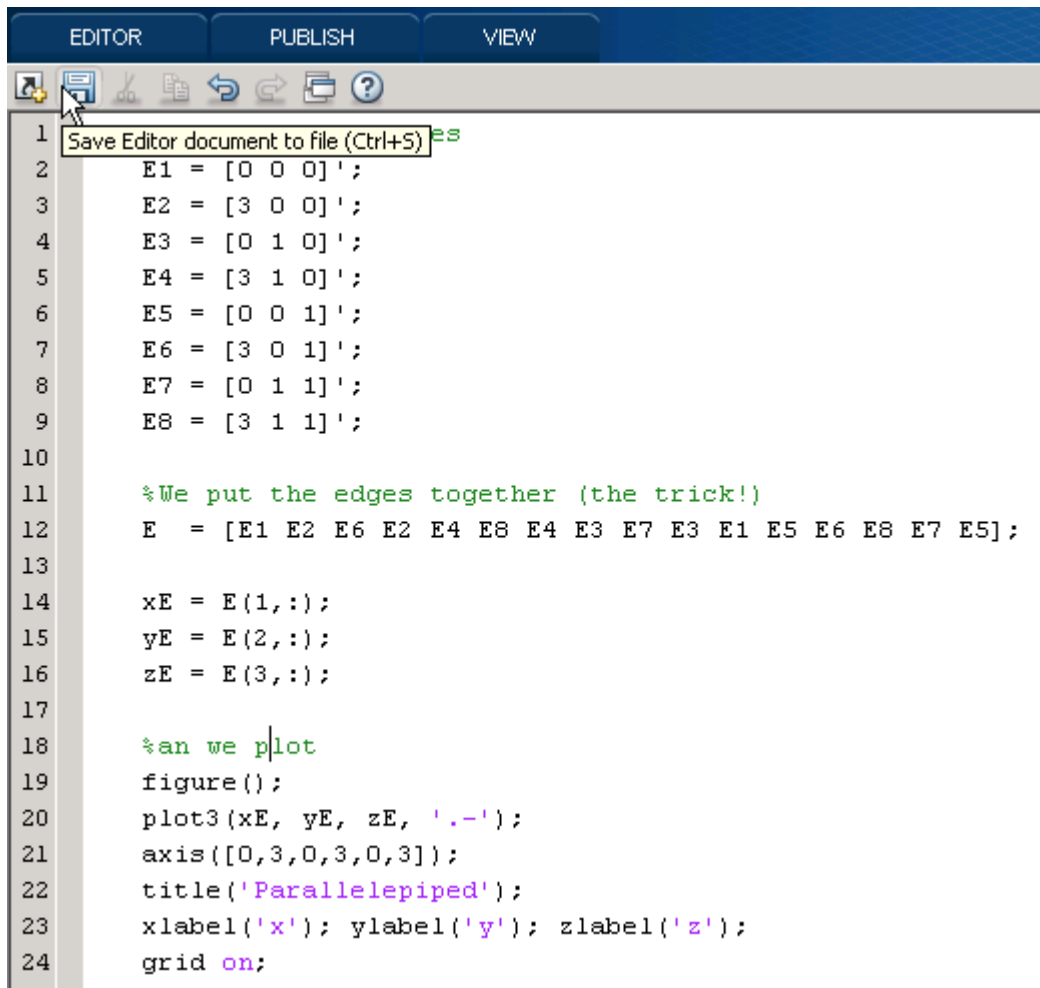
EDITOR    PUBLISH    VIEW
[Icons]
1 - close all;clear all;clc
2 - figure
3 - set(gcf,'Menubar','none','Name','Mon graph','NumberTitle','off',...
4 -     'Color',[0.8314 0.8157, 0.7483],'Position',[300 100 500 400]);
5
6 - phi=linspace(0,pi,30);
7 - theta=linspace(0,2*pi,30);
8 - [phi,theta]=meshgrid(phi,theta);
9
10 - x=sin(phi).*cos(theta);
11 - y=sin(phi).*sin(theta);
12 - z=cos(phi);
13
14 - mhndl=mesh(x,y,z);
15 - set(mhndl,'FaceAlpha',0.5);
16 - axis equal;
17
18 - title('Parametric Sphere');
19 - xlabel('x');ylabel('y');zlabel('z');

```

The result will be:



Something that seems to be easy but has no built-in function in MATLAB as far as I know is to create a parallelepiped (useful for physicists and chemists because a lot of probes have this shape):

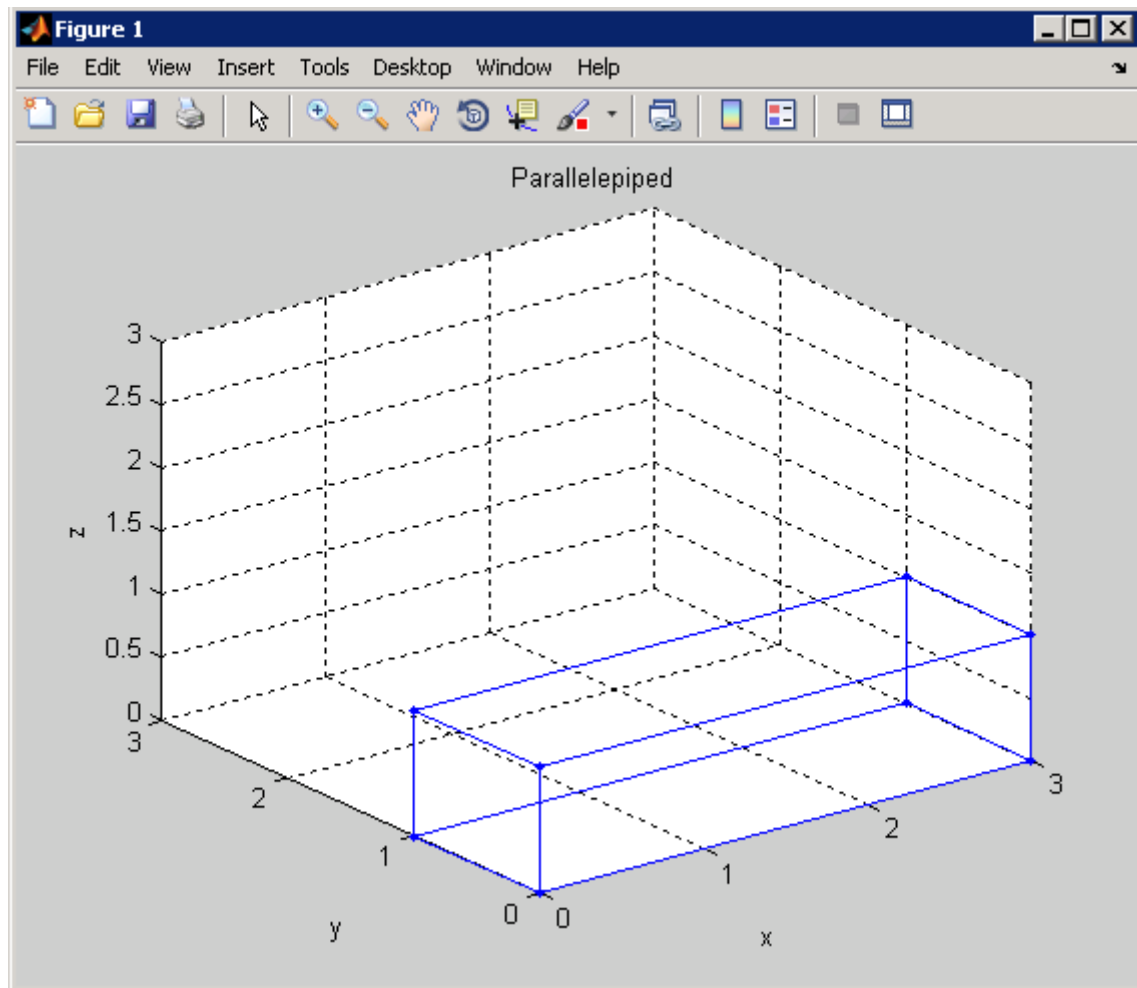


```

1  Save Editor document to file (Ctrl+S)
2  E1 = [0 0 0]';
3  E2 = [3 0 0]';
4  E3 = [0 1 0]';
5  E4 = [3 1 0]';
6  E5 = [0 0 1]';
7  E6 = [3 0 1]';
8  E7 = [0 1 1]';
9  E8 = [3 1 1]';
10
11  %We put the edges together (the trick!)
12  E = [E1 E2 E6 E2 E4 E8 E4 E3 E7 E3 E1 E5 E6 E8 E7 E5];
13
14  xE = E(1,:);
15  yE = E(2,:);
16  zE = E(3,:);
17
18  %an we plot
19  figure();
20  plot3(xE, yE, zE, '.-');
21  axis([0,3,0,3,0,3]);
22  title('Parallelepiped');
23  xlabel('x'); ylabel('y'); zlabel('z');
24  grid on;

```

This will give:



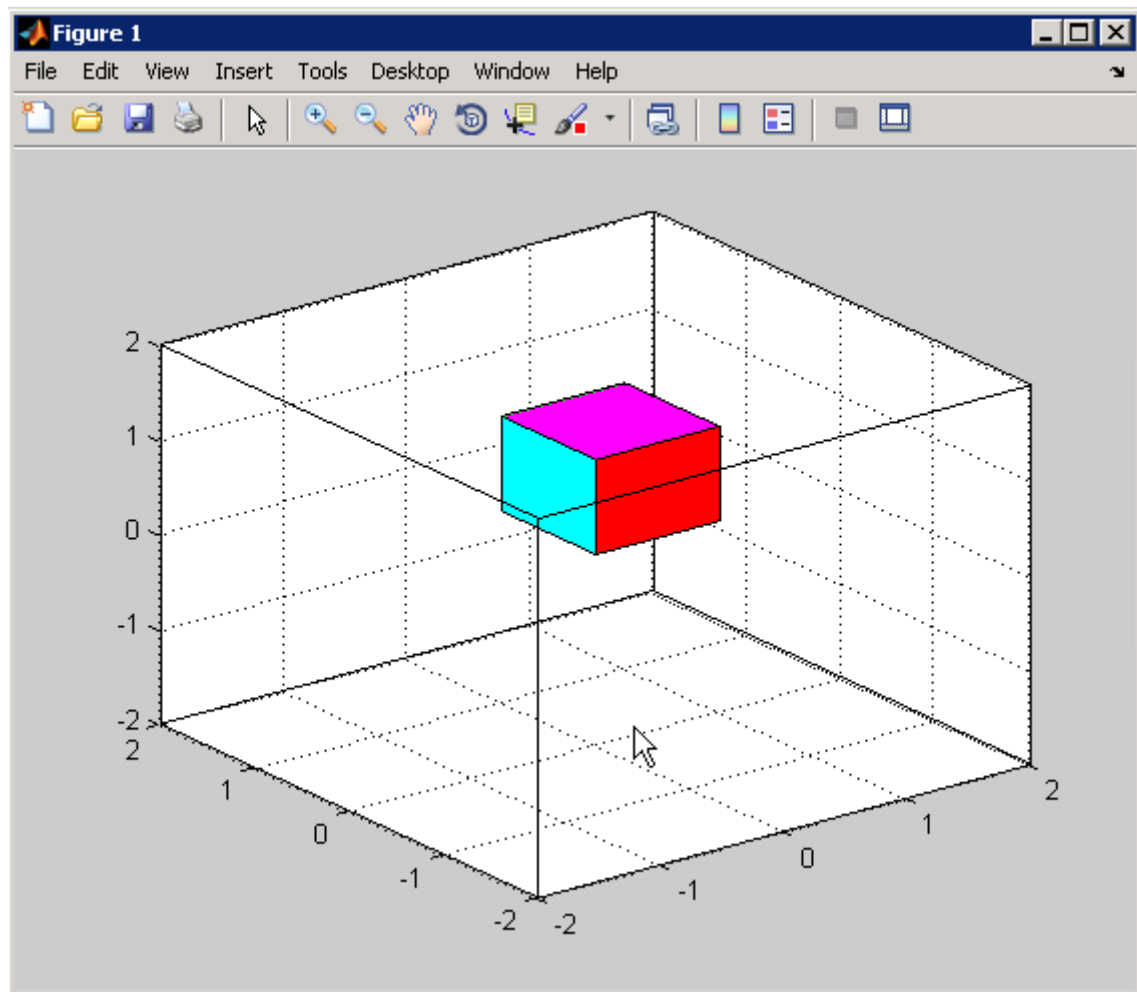
Or in another way using the function **patch()**:

```

EDITOR    PUBLISH    VIEW
1      %vector with the 8 points of the parallelepiped
2      vm=[0 0 0; 1 0 0; 1 1 0; 0 1 0; 0 0 1; 1 0 1; 1 1 1; 0 1 1];
3
4      %because vm must be a row vector for each of the 6 faces we will reference
5      %the four building points of vm
6      fm=[1 2 6 5; 2 3 7 6; 3 4 8 7; 4 1 5 8; 1 2 3 4; 5 6 7 8];
7
8      patch('Vertices',vm,'Faces',fm, 'FaceVertexCData',hsv(6),'FaceColor','flat');
9      axis([-2 2 -2 2 -2 2]);
10     grid on;box on;

```

The result will be:



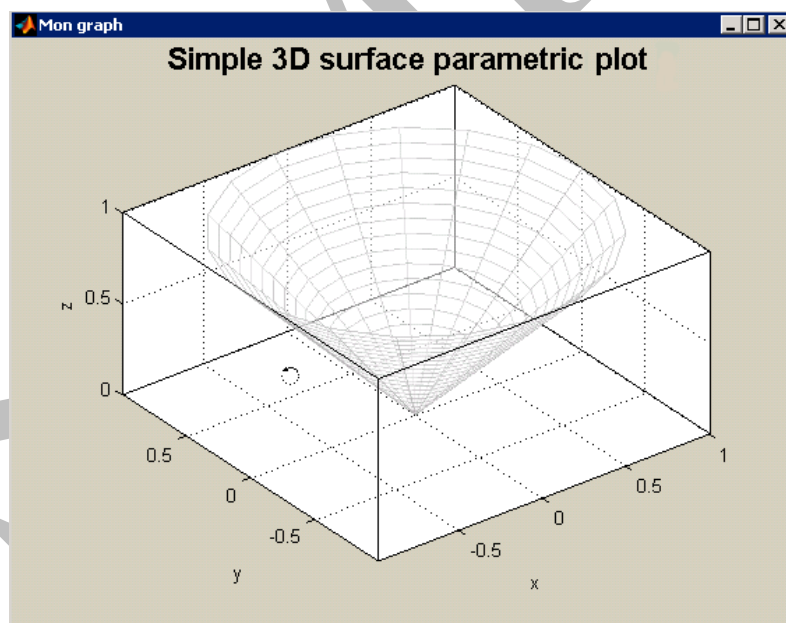
Now have a look of the logical continuity that is the cone but by adding transparency to the surface and the mesh:

```

EDITOR PUBLISH VIEW
1 close all;clear all;clc
2 figure
3 set(gcf,'MenuBar','none','Name','Mon graph','NumberTitle','off',...
4     'Color',[0.277 0.181 0.8157 0.7483]);
5
6 r=linspace(0,1,20);
7 theta=linspace(0,2*pi,20);
8 %and r and theta becomes now a 20x20 matrix
9 [r,theta]=meshgrid(r,theta);
10
11 %we change from cylindric to cart. coordinates
12 x=r.*cos(theta);
13 y=r.*sin(theta);
14 z=r;
15
16 mhndl=mesh(x,y,z);
17 set(mhndl,'EdgeColor',[.6,.6,.6],'FaceAlpha',0.5,'EdgeAlpha',0.5);
18 xlabel('x');ylabel('y');zlabel('z');
19 title('Simple 3D surface parametric plot','fontweight','bold','fontsize',16);
20 axis equal;
21 box on;
22 rotate3d on;

```

The result will look like following:



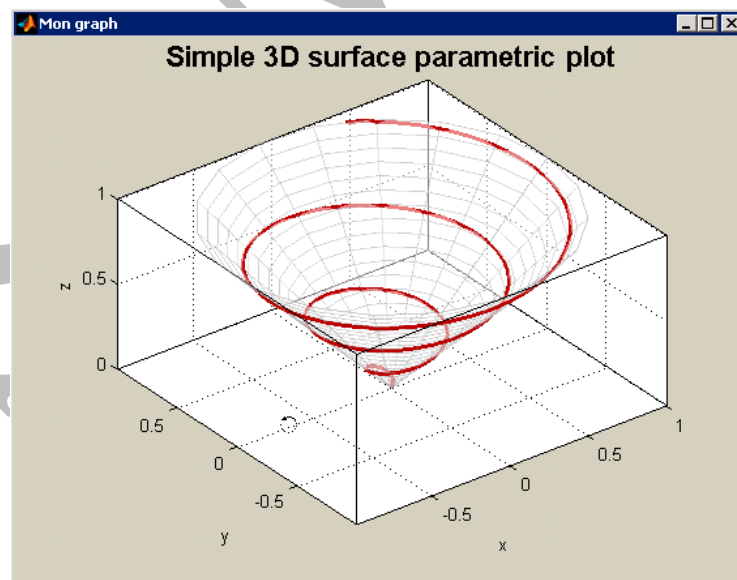
Continue now with a little bit more sophisticated version near of what I saw once in General Relativity (a path on a curve surface) using a mix of the functions **meshgrid()**, **mesh()** and **line()**:

```

1  close all;clear all;clc
2  figure
3  set(gcf,'Menubar','none','Name','Mon graph','NumberTitle','off',...
4      'Color',[0.8314 0.8157 0.7483]);
5
6  r=linspace(0,1,20);
7  theta=linspace(0,2*pi,20);
8  [r,theta]=meshgrid(r,theta);
9
10 x=r.*cos(theta);y=r.*sin(theta);z=r;
11 mhn1=mesh(x,y,z);
12 set(mhn1,'EdgeColor',[.6,.6,.6],'FaceAlpha',0.5,'EdgeAlpha',0.5);
13
14 %now we add a line on the surface and this can be done only if the
15 %surface and the line trigonometric function have the same amplitude
16 % (=1 in this example)
17 r=linspace(0,1,200);
18 x=r.*cos(20*r);y=r.*sin(20*r);z=r;
19
20 surfLine=line(x,y,z);
21 set(surfLine,'Color',[.625,0,0],'LineWidth',2)
22
23 xlabel('x');ylabel('y');zlabel('z');
24 title('Simple 3D surface parametric plot','fontweight','bold','fontsize',16);
25 axis equal;
26 box on;
27 rotate3d on;

```

The result will look like:

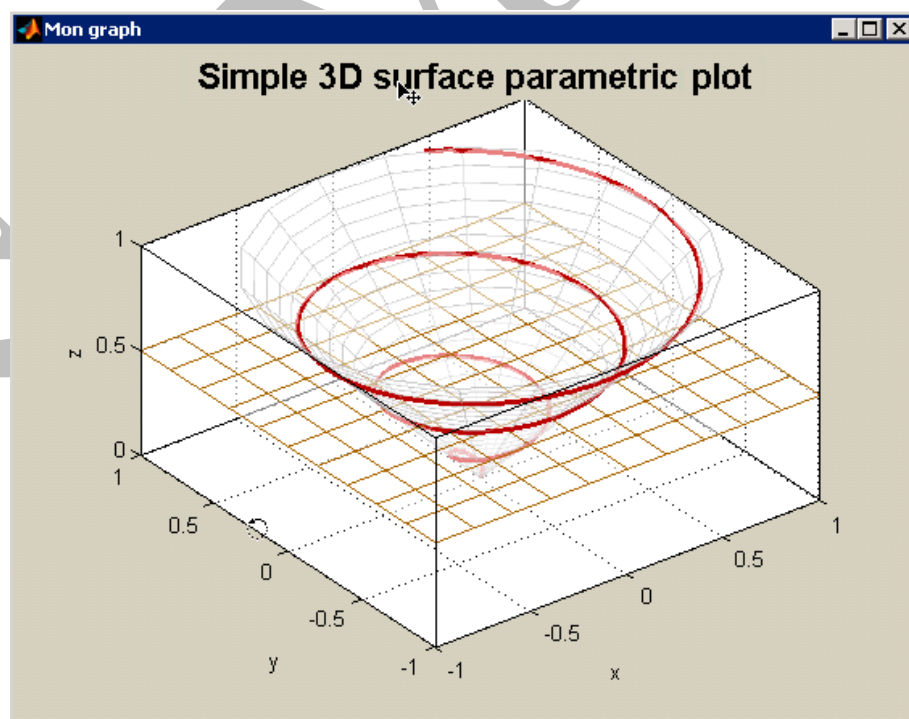


And now just for practice a composition of academics stuff (cone, line, plane):

```

1  close all;clear all;clc
2  figure
3  set(gcf,'Menubar','none','Name','Mon graph','NumberTitle','off',...
4      'Color',[0.8314 0.8157 0.7483]);
5
6  %----- cone
7  r=linspace(0,1,20);
8  theta=linspace(0,2*pi,20);
9  [r,theta]=meshgrid(r,theta);
10 x=r.*cos(theta);y=r.*sin(theta);z=r;
11 mhn1=mesh(x,y,z);
12 set(mhn1,'EdgeColor',[.6,.6,.6],'FaceAlpha',0.5,'EdgeAlpha',0.5);
13 %----- line
14 r=linspace(0,1,200);
15 x=r.*cos(20*r);y=r.*sin(20*r);z=r;
16 surfLine=line(x,y,z);
17 set(surfLine,'Color',[.625,0,0],'LineWidth',2)
18 %----- now we add a square plane at middle height
19 hold on
20 [x,y]=meshgrid(-1:0.2:1);
21 z=0.5*ones(size(x));
22 plane=mesh(x,y,z);
23 set(plane,'EdgeColor',[0.6,0.4,0],'FaceAlpha',0.5)
24
25 xlabel('x');ylabel('y');zlabel('z');
26 title('Simple 3D surface parametric plot','fontweight','bold','fontsize',16);
27 axis equal;box on;rotate3d on;

```

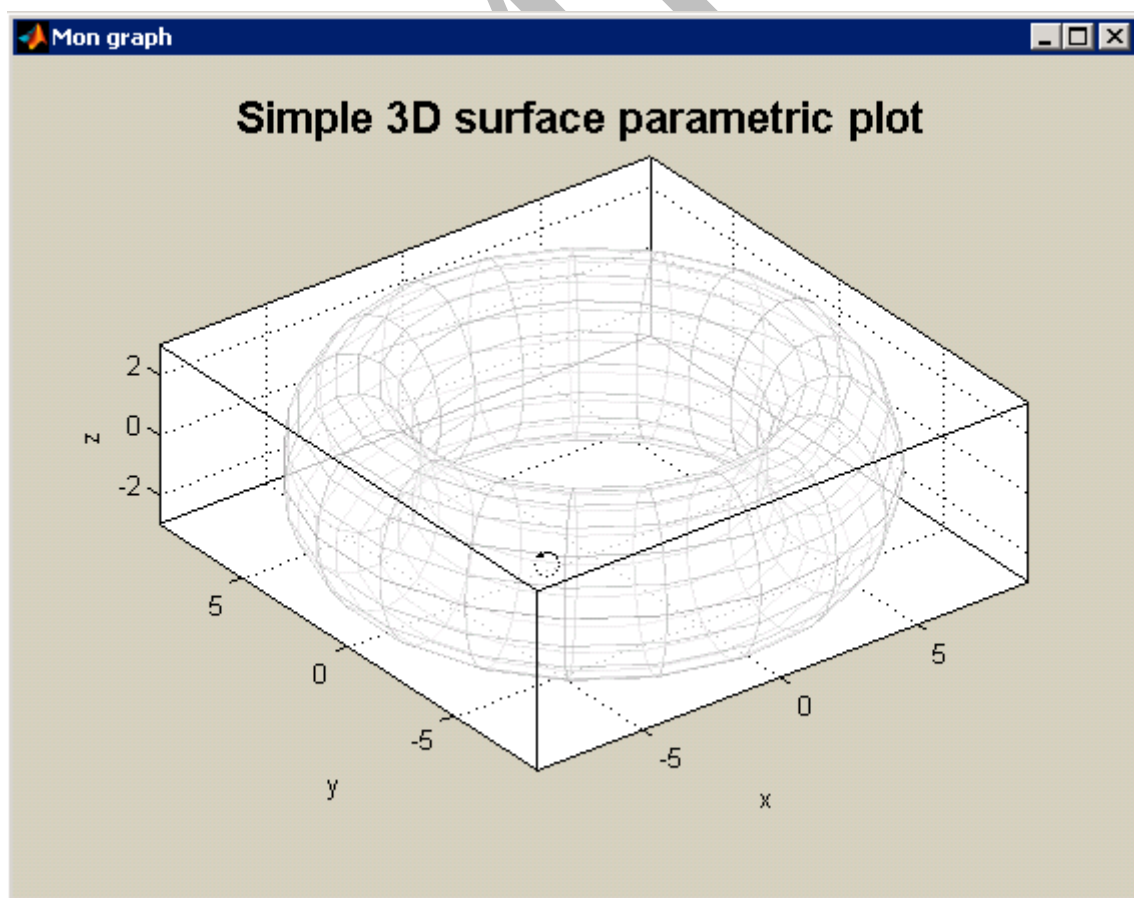


This is also a very academic example to practice with a little bit more complicate surface (a tore):

```

1  close all;clear all;clc
2  figure
3  set(gcf,'MenuBar','none','Name','Mon graph','NumberTitle','off',...
4      'Color',[0.8314 0.8157 0.7483]);
5
6  a=7;b=2;c=3;
7  u=linspace(0,2*pi,20);
8  v=linspace(0,2*pi,20);
9  [u,v]=meshgrid(u,v);
10 x=(a+b*cos(u)).*cos(v);
11 y=(a+b*cos(u)).*sin(v);
12 z=c*sin(u);
13
14 tore=mesh(x,y,z);
15 set(tore,'EdgeColor',[.6,.6,.6],'FaceAlpha',0.5,'EdgeAlpha',0.5);
16 axis equal
17
18 xlabel('x');ylabel('y');zlabel('z');|
19 title('Simple 3D surface parametric plot','fontweight','bold','fontsize',16);
20 axis equal;box on;rotate3d on;

```

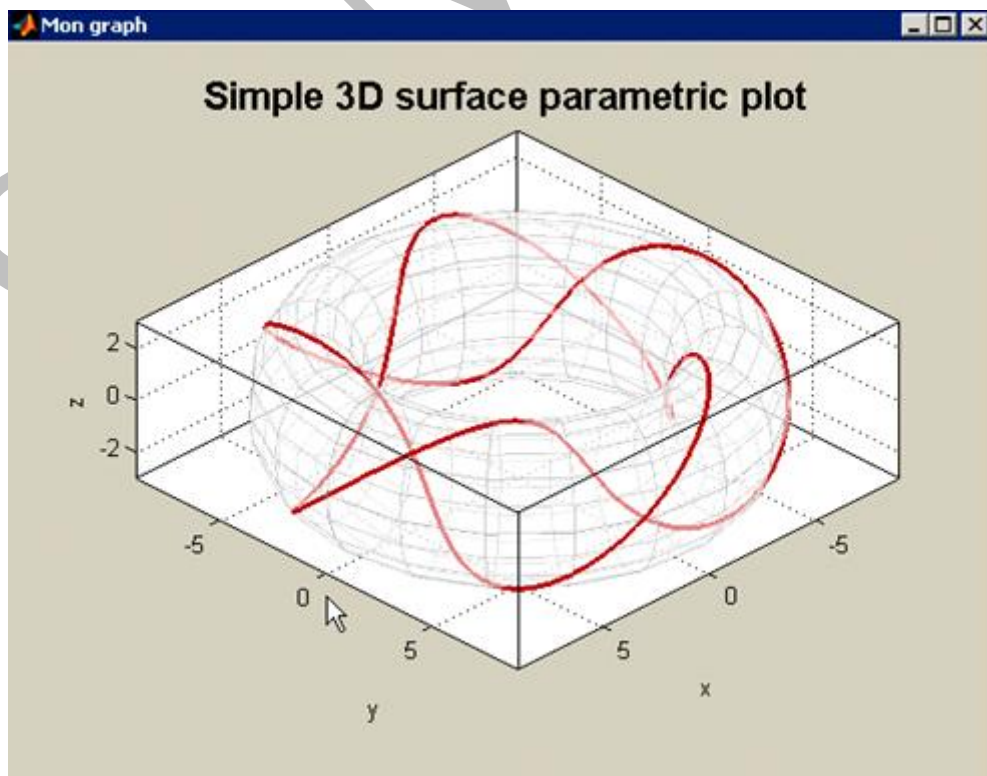


And always for practice purposes:

```

1  close all;clear all;clc
2  figure
3  set(gcf,'Menubar','none','Name','Mon graph','NumberTitle','off',...
4      'Color',[0.8314 0.8157 0.7483]);
5
6  %----- the tore
7  a=7;b=2;c=3;
8  u=linspace(0,2*pi,20);
9  v=linspace(0,2*pi,20);
10 [u,v]=meshgrid(u,v);
11 x=(a+b*cos(u)).*cos(v);
12 y=(a+b*cos(u)).*sin(v);
13 z=c*sin(u);
14 tore=mesh(x,y,z);
15 set(tore,'EdgeColor',[.6,.6,.6],'FaceAlpha',0.5,'EdgeAlpha',0.5);
16
17 %----- a line on the tore
18 t=linspace(0,2*pi,200);
19 x=(a+b*cos(5*t)).*cos(2*t);
20 y=(a+b*cos(5*t)).*sin(2*t);
21 z=c*sin(5*t);
22 surfLine=line(x,y,z);
23 set(surfLine,'Color',[.625,0,0],'LineWidth',2)
24
25 xlabel('x');ylabel('y');zlabel('z');
26 title('Simple 3D surface parametric plot','fontweight','bold','fontsize',16);
27 axis equal;box on;rotate3d on;view(135,30);

```



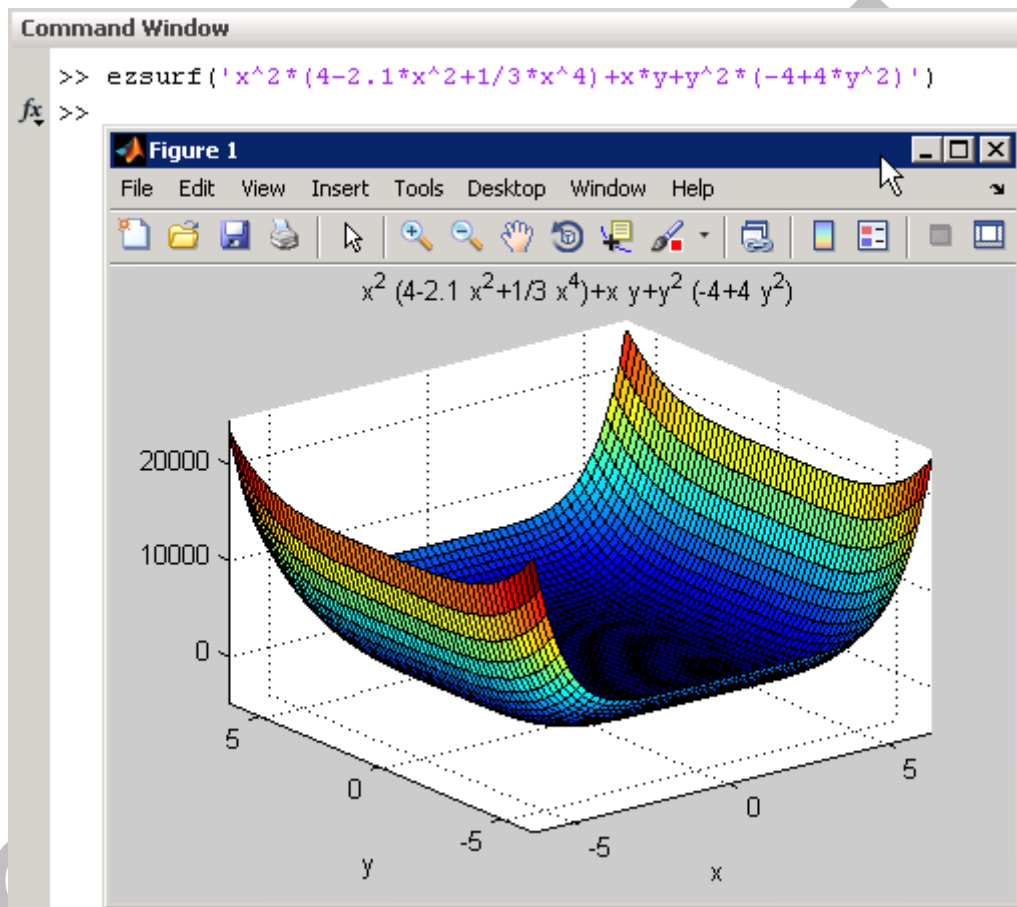
3D surface plots

I think it is not necessary to give examples of fields in which 3D surface charts are useful....

Then we start directly.

ezsurf

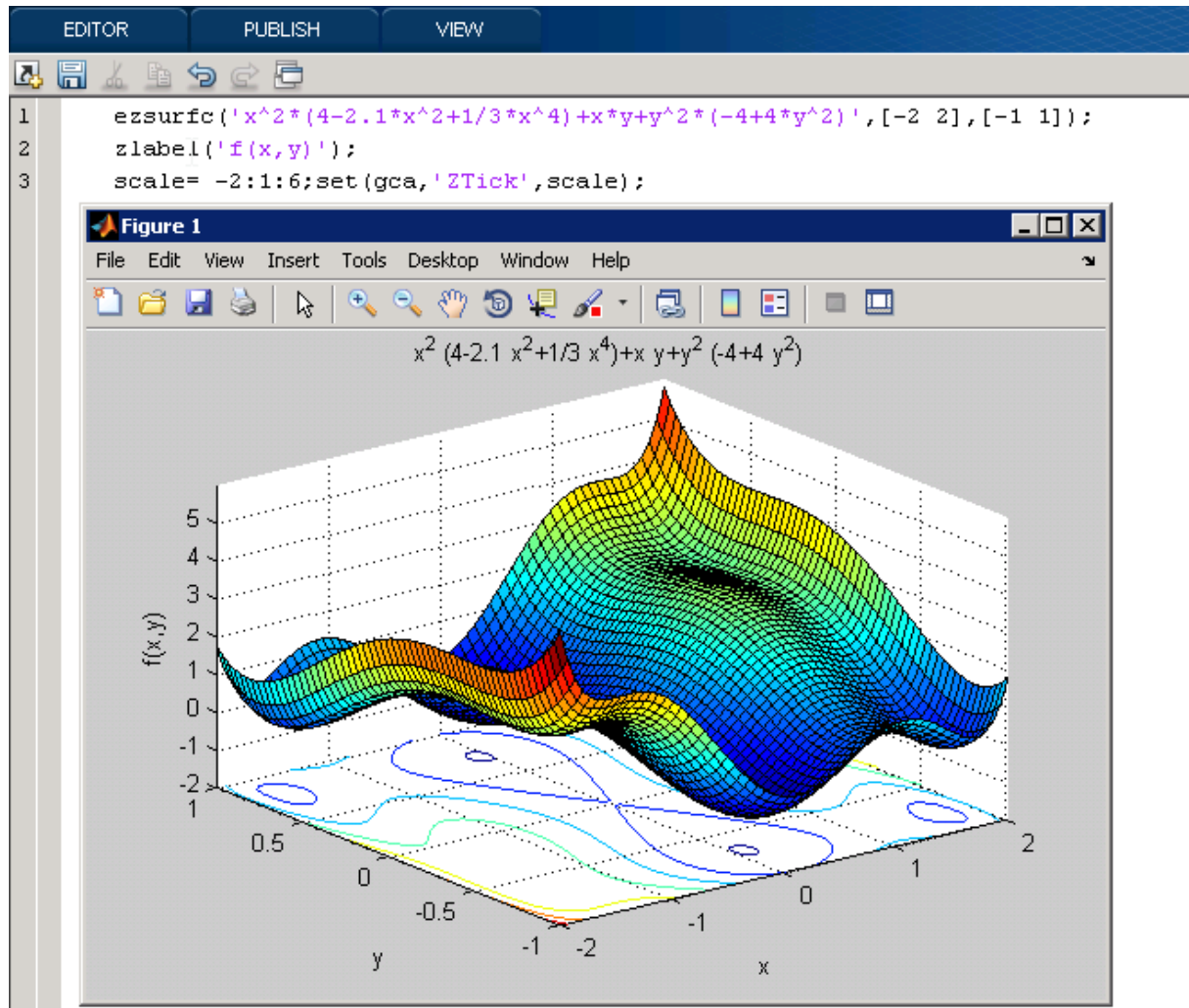
If you want to plot an analytical surface plot expression (typical in academics classes) you can just simply use the function **ezsurf()**:



ezsurf

The command **ezsurf()** also plot an analytical 3D surface plot expression but also add isoclines projected on the *XY* plane.

Let us see that with a few interesting customizations of the chart at the same time to learn also something more than just a simple function:



surf

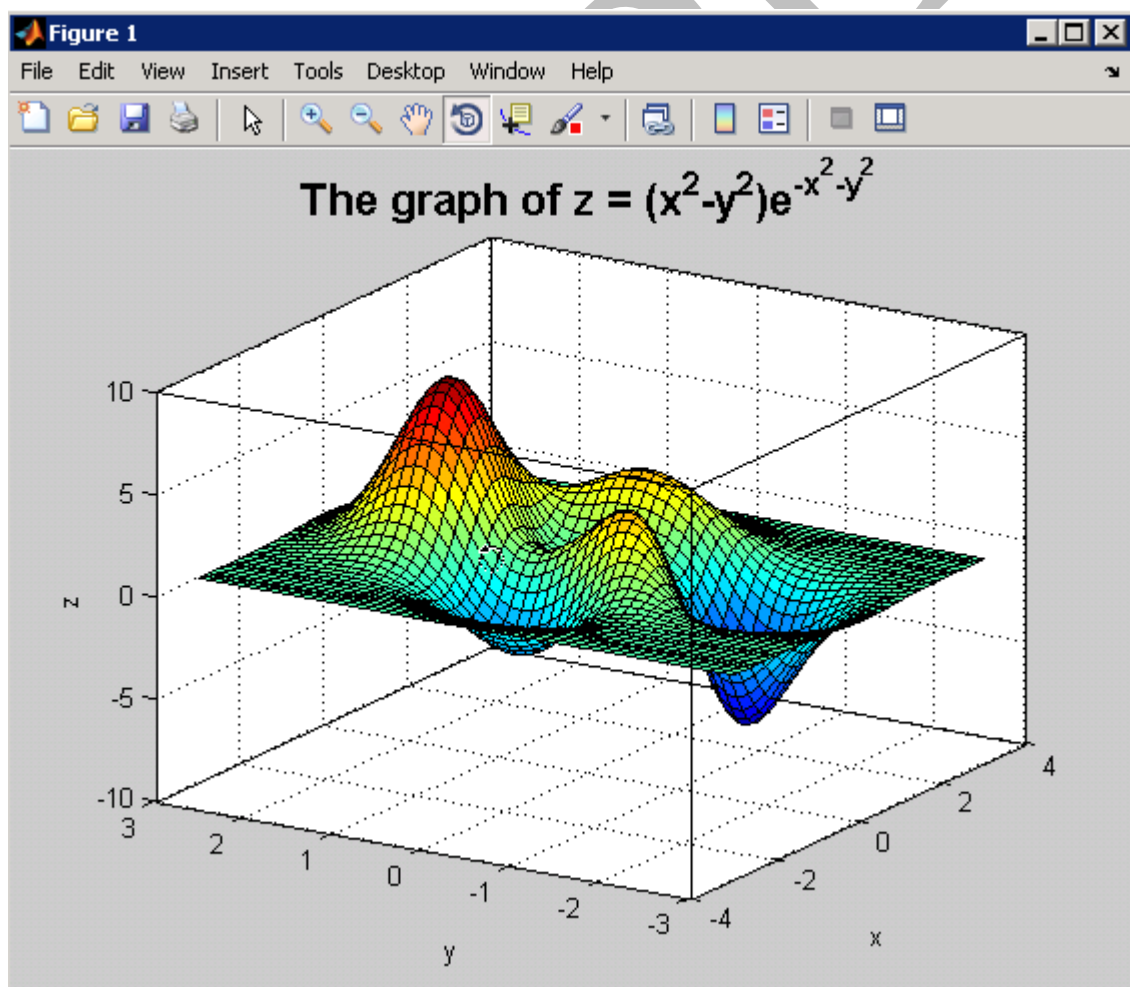
The **surf()** function will be useful when you have not an analytical expression but a grid of measures points (**don't forget that x, y, z must be matrices!**) and at the same time let us have a look to the command **view()**:

```

1      [x, y, z]=peaks;
2      surf(x,y,z);
3      title('The graph of  $z = (x^2-y^2)e^{-x^2-y^2}$ ','fontweight',...
4          'bold','fontsize',16);
5      box on;
6      xlabel('x');ylabel('y');zlabel('z');
7      rotate3d on;view(-58,24);

```

The result is the same as **ezsurf()** excepted that the function is not analytically expressed:



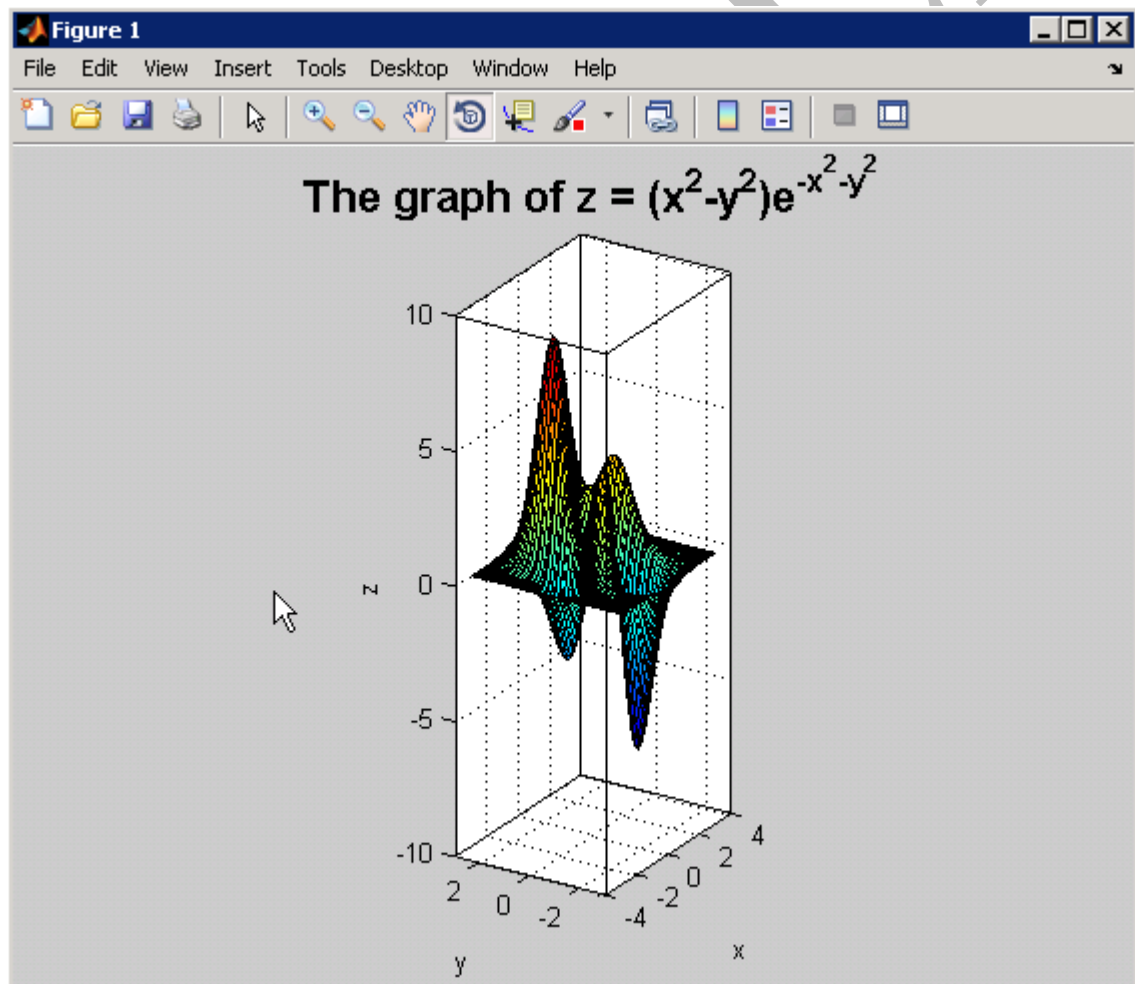
As you can see on the previous chart axes are also not proportional. We can use the same technic as for 2D chart to respect them:

```

1      [x, y ,z]=peaks;
2      surf(x,y,z);
3      title('The graph of  $z = (x^2-y^2)e^{-x^2-y^2}$ ','fontweight',...
4          'bold','fontsize',16);
5      box on;
6      xlabel('x');ylabel('y');zlabel('z');
7      rotate3d on;view(-58,24);
8      axis equal;
9      axis([-4 4 -3 3 -10 10]);

```

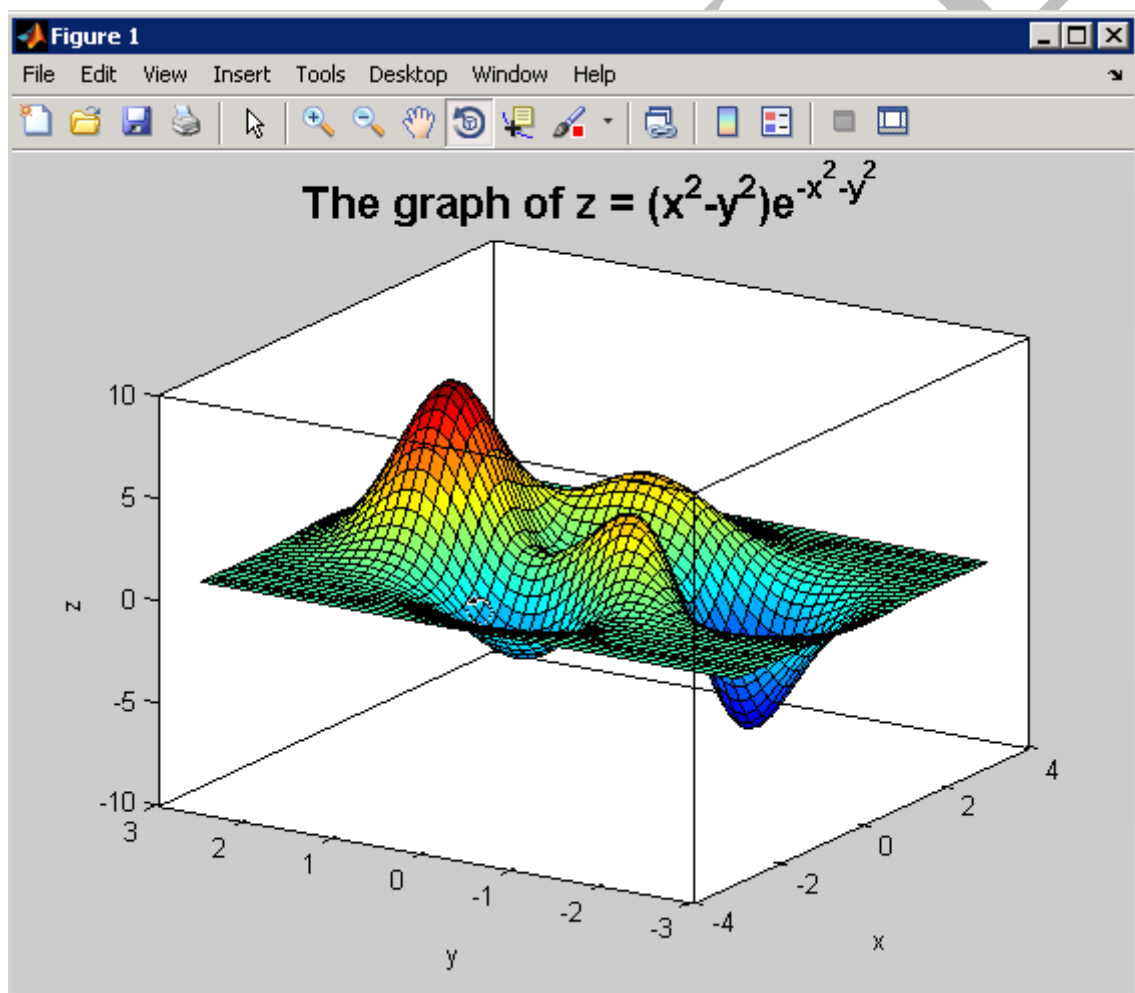
The result will be less pretty but more respective to real proportions:



Now see the result without grid just by curiosity:

```
EDITOR PUBLISH VIEW
1 [x, y, z]=peaks;
2 surf(x,y,z);
3 title('The graph of  $z = (x^2-y^2)e^{-x^2-y^2}$ ','fontweight',...
4     'bold','fontsize',16);
5 box on;
6 xlabel('x');ylabel('y');zlabel('z');
7 rotate3d on;view(-58,24);
8 grid off;
```

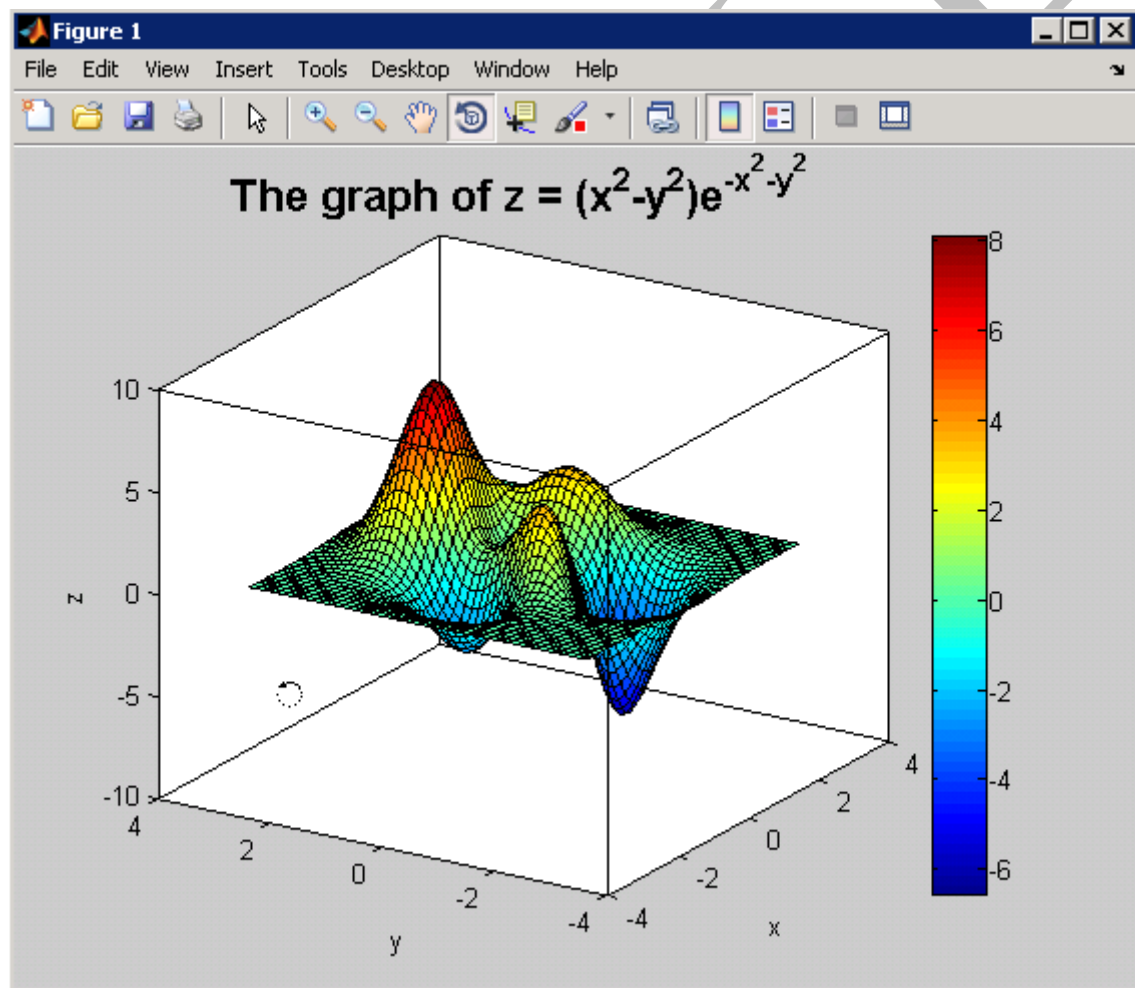
The result will be:



And how does it look like with a **colorbar**?:

```
1 [x, y, z]=peaks;  
2 surf(x,y,z);  
3 title('The graph of  $z = (x^2-y^2)e^{-x^2-y^2}$ ','fontweight',...  
4     'bold','fontsize',16);  
5 box on;  
6 xlabel('x');ylabel('y');zlabel('z');  
7 rotate3d on;view(-58,24);  
8 grid off;  
9 colorbar;
```

Not to bad



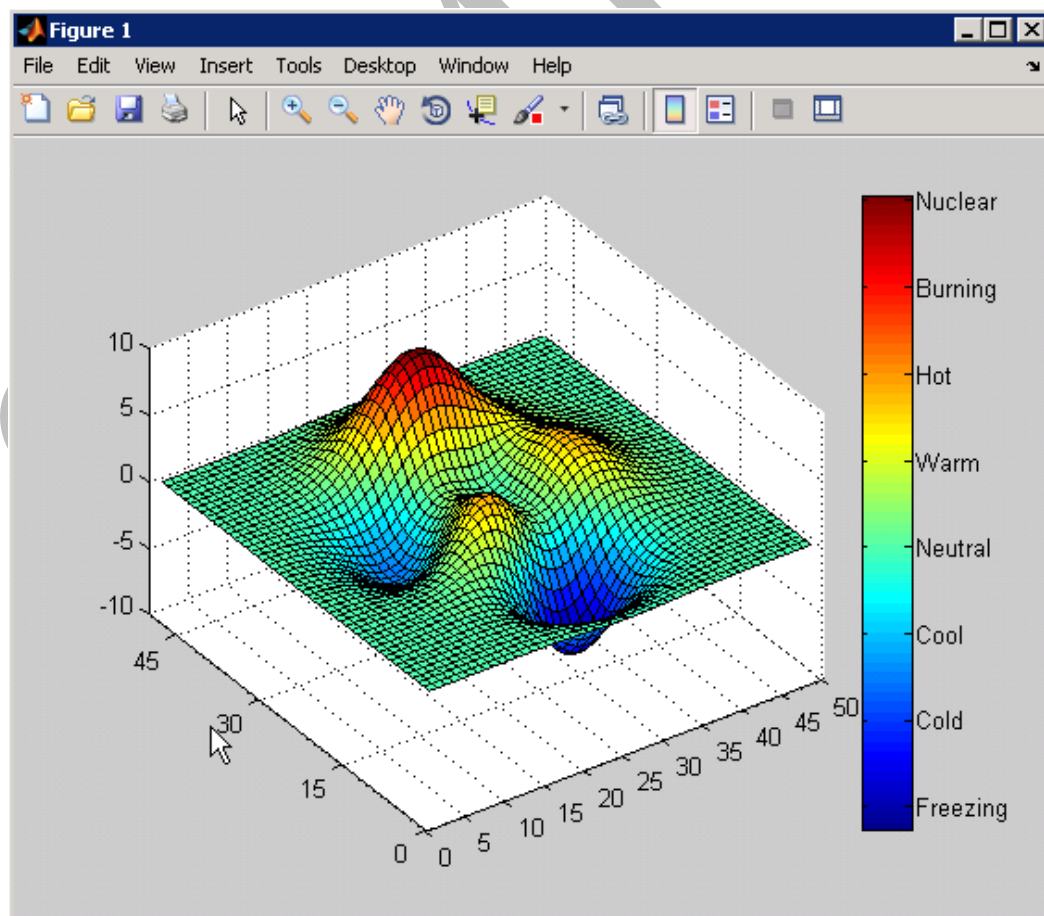
In reality (in practice...) we need sometimes words instead of value next to colorbars. See the trick:

```

1      clear all;close all;clc
2
3      surface(peaks)
4      colorbar('YTickLabel',...           % set labels to the colorbar
5          {'Freezing','Cold','Cool','Neutral',...
6          'Warm','Hot','Burning','Nuclear'})
7      view(-35,45)
8
9      number_panels = 0:5:50;
10     number_batteries = 0:15:50;
11
12     set(gca,'XTick',number_panels)      % set Xticks
13     set(gca,'YTick',number_batteries ) % set Yticks
14     grid on

```

This will give the expected result:



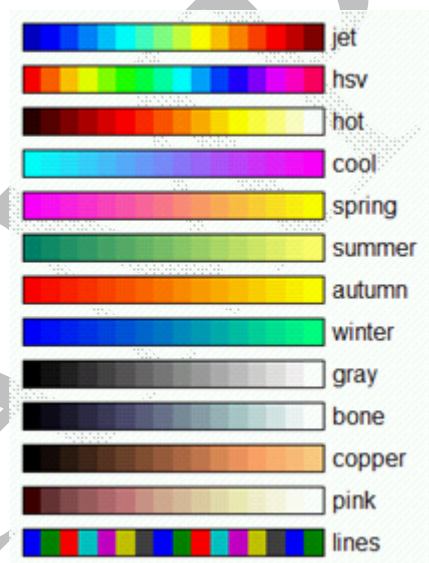
And if we want to invert the colors of the colorbar? We can use the function **flipud()** for this purpose:

```

EDITOR PUBLISH VIEW
1 [x, y, z]=peaks;
2 surf(x,y,z);
3 title('The graph of z = (x^2-y^2)e^(-x^2-y^2)','fontweight',...
4     'bold','fontsize',16);
5 box on;
6 xlabel('x');ylabel('y');zlabel('z');
7 rotate3d on;view(-58,24);
8 grid off;
9 colorbar;
10 colormap(flipud(colormap));
11

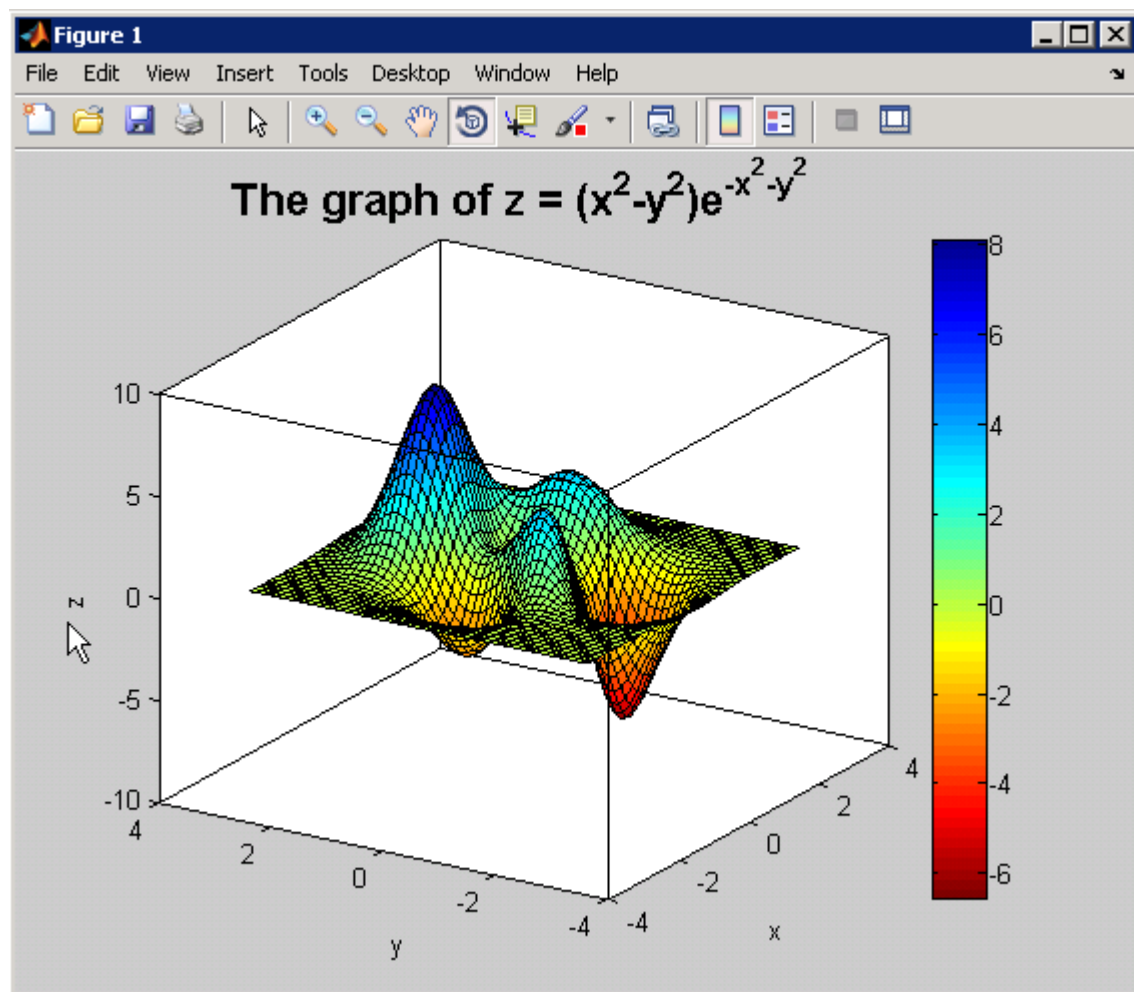
```

You can change the colors style of the colorbar by writing for example **colorbar('hot')**. Different styles exists:



Afficher les isoclines/équipotentielles en 2D ou 3D

The result will be as expected:



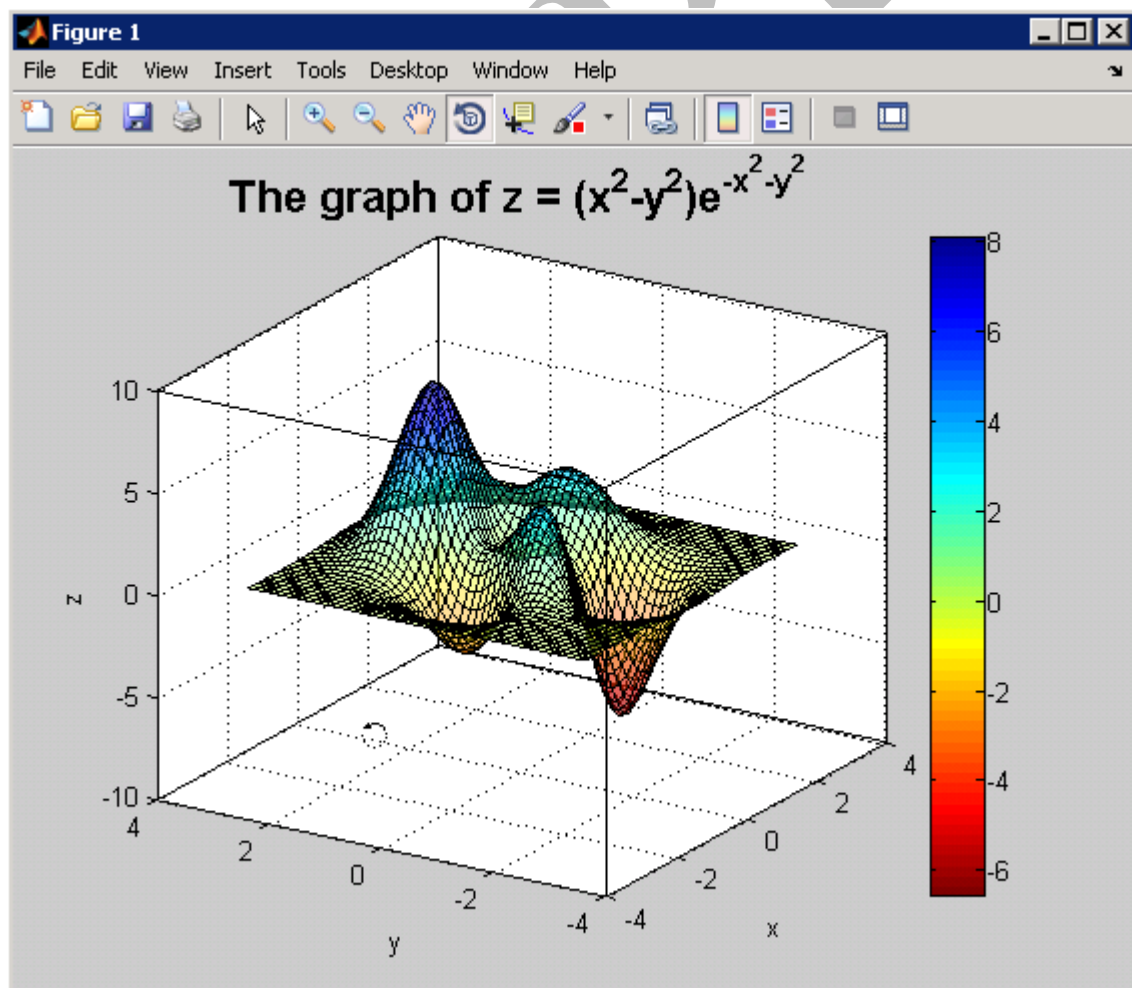
And what about transparency if we play with the function **alpha()**:

```

EDITOR PUBLISH VIEW
1 - [x, y, z]=peaks;
2 - surf(x,y,z);
3 - alpha(.4);
4 - title('The graph of  $z = (x^2-y^2)e^{-x^2-y^2}$ ', 'fontweight', ...
5 -     'bold', 'fontsize', 16);
6 - box on;
7 - xlabel('x'); ylabel('y'); zlabel('z');
8 - rotate3d on; view(-58, 24);
9 - grid on;
10 - colorbar;
11 - colormap(flipud(colormap));
12 - %to see trough the grid
13 - hidden off;

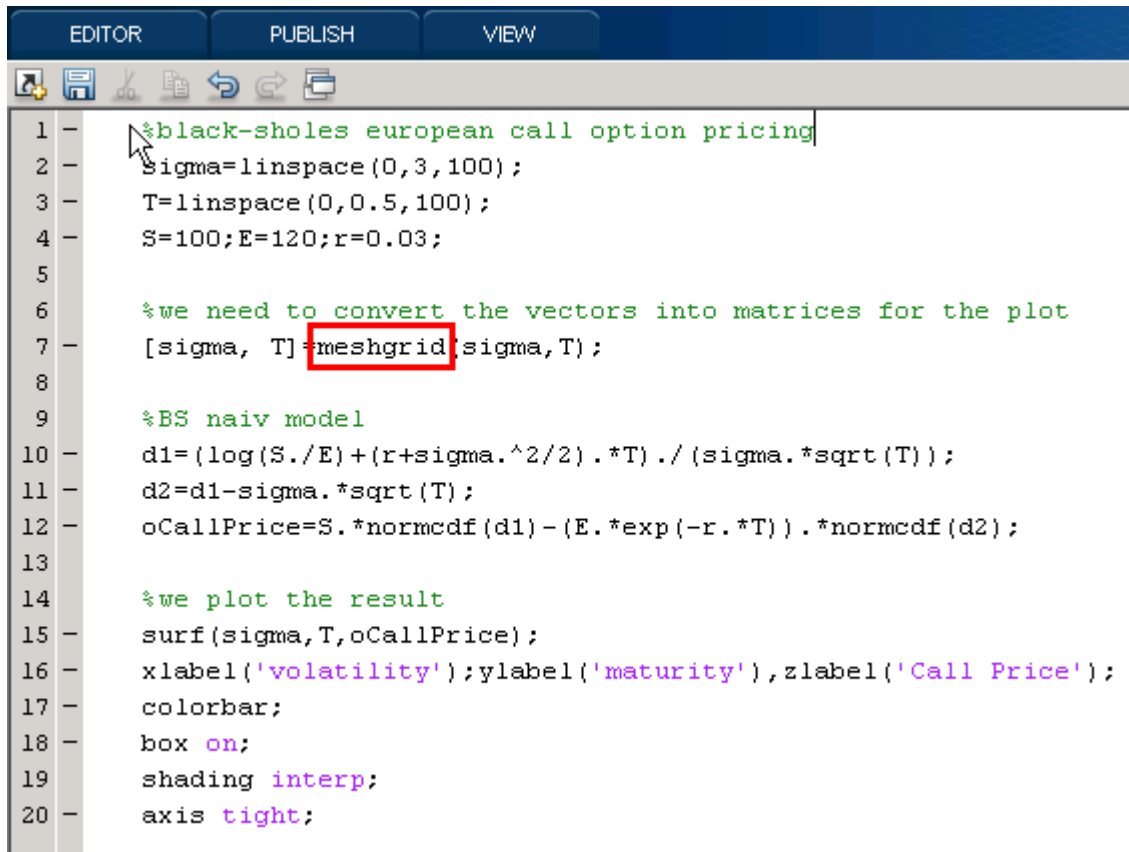
```

We can see trough!:

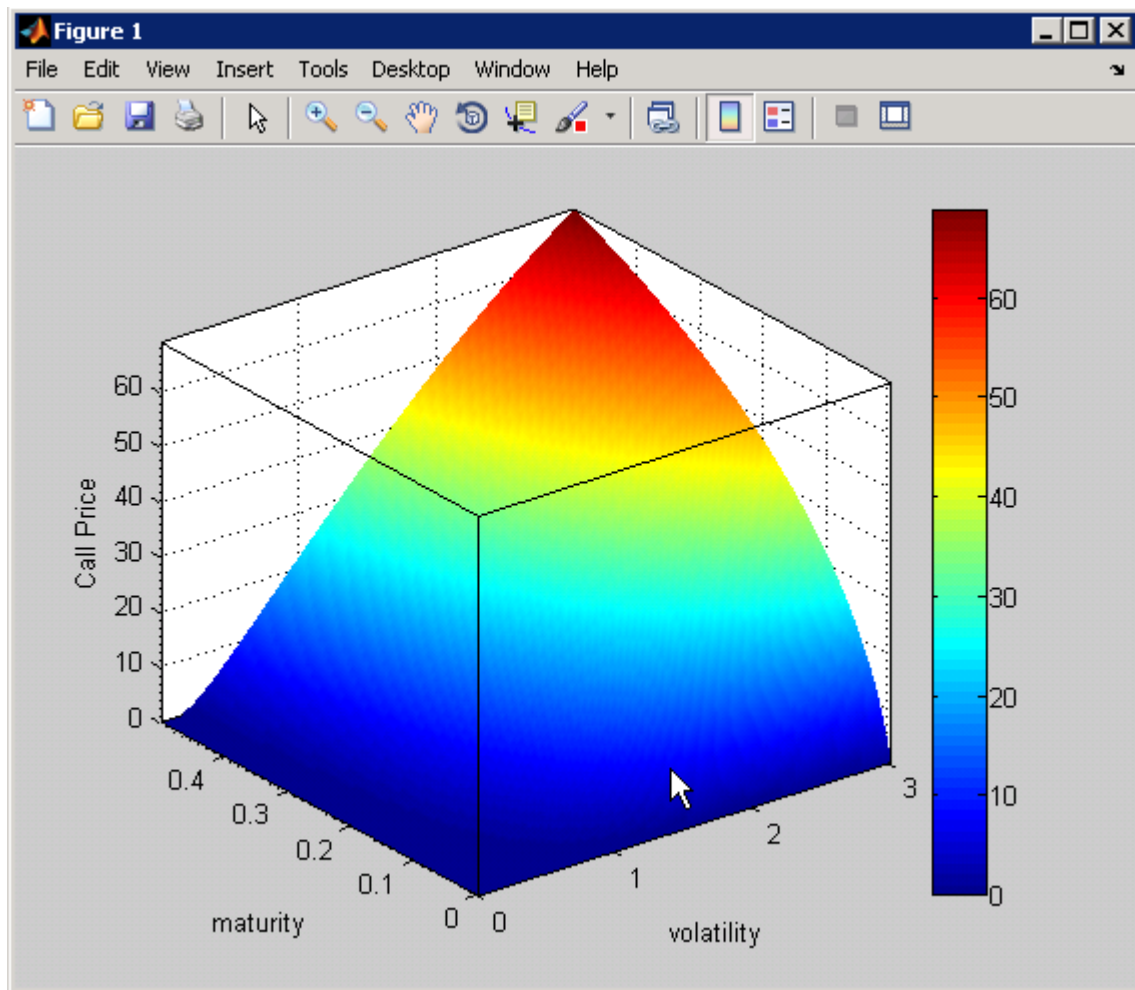


surf with an explicit or implicit function!

If you have an old version of MATLAB you won't be able to use **ezsurf()**. Then here is method combining **meshgrid()** and **surf()** when you have a known function or only measures points to plot a surface and at the same time see how the command **shading interp** smooth the colors (for people working in laboratories and that have two vectors of parameters and one vector of measure you will have to wait a little bit):



```
1 - %black-sholes european call option pricing
2 - sigma=linspace(0,3,100);
3 - T=linspace(0,0.5,100);
4 - S=100;E=120;r=0.03;
5
6 - %we need to convert the vectors into matrices for the plot
7 - [sigma, T]=meshgrid(sigma,T);
8
9 - %BS naiv model
10 - d1=(log(S./E)+(r+sigma.^2/2).*T)./(sigma.*sqrt(T));
11 - d2=d1-sigma.*sqrt(T);
12 - oCallPrice=S.*normcdf(d1)-(E.*exp(-r.*T)).*normcdf(d2);
13
14 - %we plot the result
15 - surf(sigma,T,oCallPrice);
16 - xlabel('volatility');ylabel('maturity'),zlabel('Call Price');
17 - colorbar;
18 - box on;
19 - shading interp;
20 - axis tight;
```



Animated surf

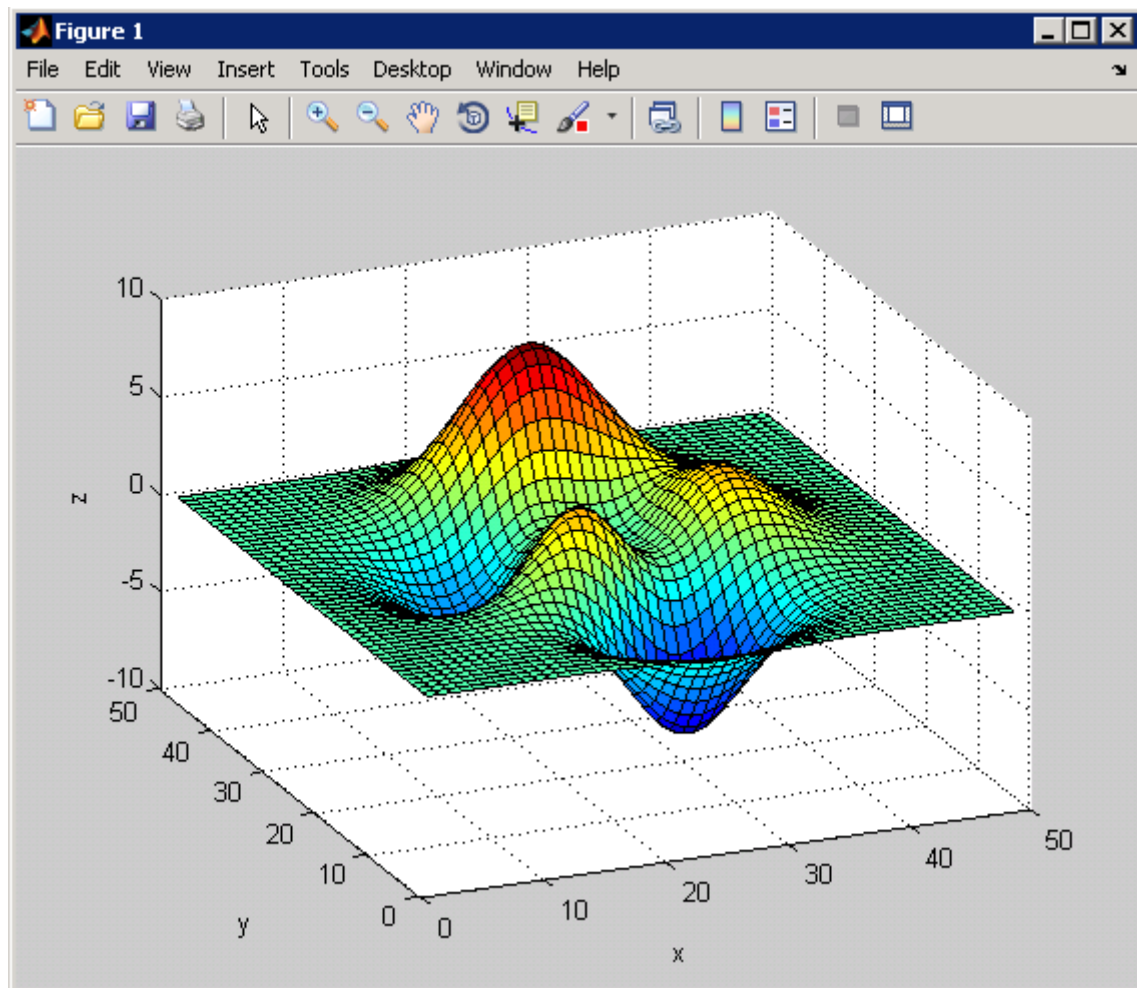
For users that want to animate a rotating chart for their MS PowerPoint slides or their website (we will see later how to export the result as a *.gif) you can play with a loop and use the function **view()** and the command **drawnow**:

```

1      clear all;close all;clc
2      Z=peaks;surf(Z);
3      % Keep axes constant
4      axis([0 50 0 50 -10 10])
5      % Define title and labels for reference
6      xlabel('x'); ylabel('y'); zlabel('z')
7      % Modify azimuth (horizontal rotation) and update drawing
8      for az = -37.5 : .5 : 30
9          view(az, 30)
10         drawnow
11     end

```

For sure you won't be able to see the animation work in this PDF:



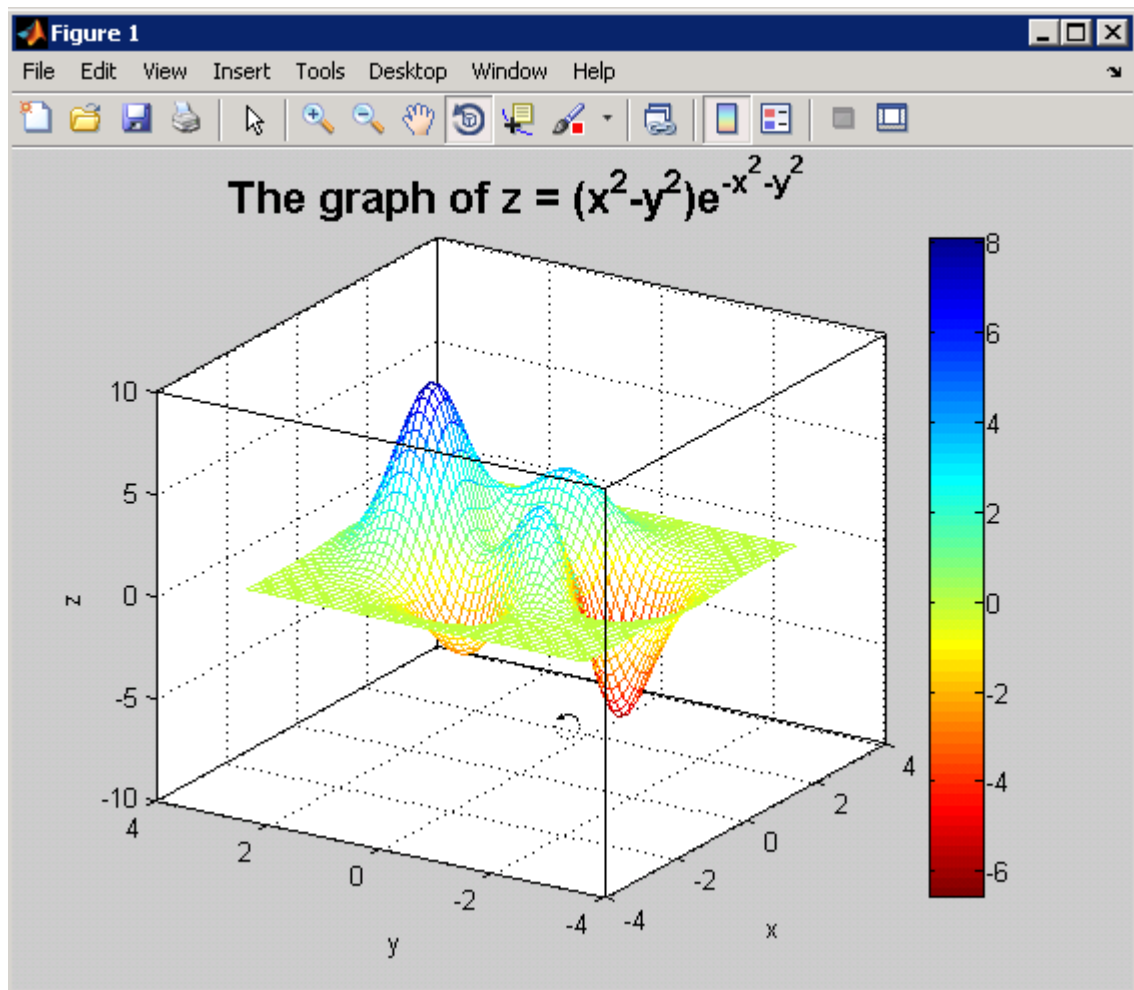
mesh

The **mesh()** function is very important because it fits like for surf() not a surface but a surface grid:

```

EDITOR    PUBLISH    VIEW
[Icons]
1      [x, y, z]=peaks;
2      mesh(x,y,z);
3      title('The graph of z = (x^2-y^2)e^(-x^2-y^2)','fontweight',...
4          'bold','fontsize',16);
5      box on;
6      xlabel('x');ylabel('y');zlabel('z');
7      rotate3d on;view(-58,24);
8      grid on;
9      colorbar;
10     colormap(flipud(colormap));
11

```



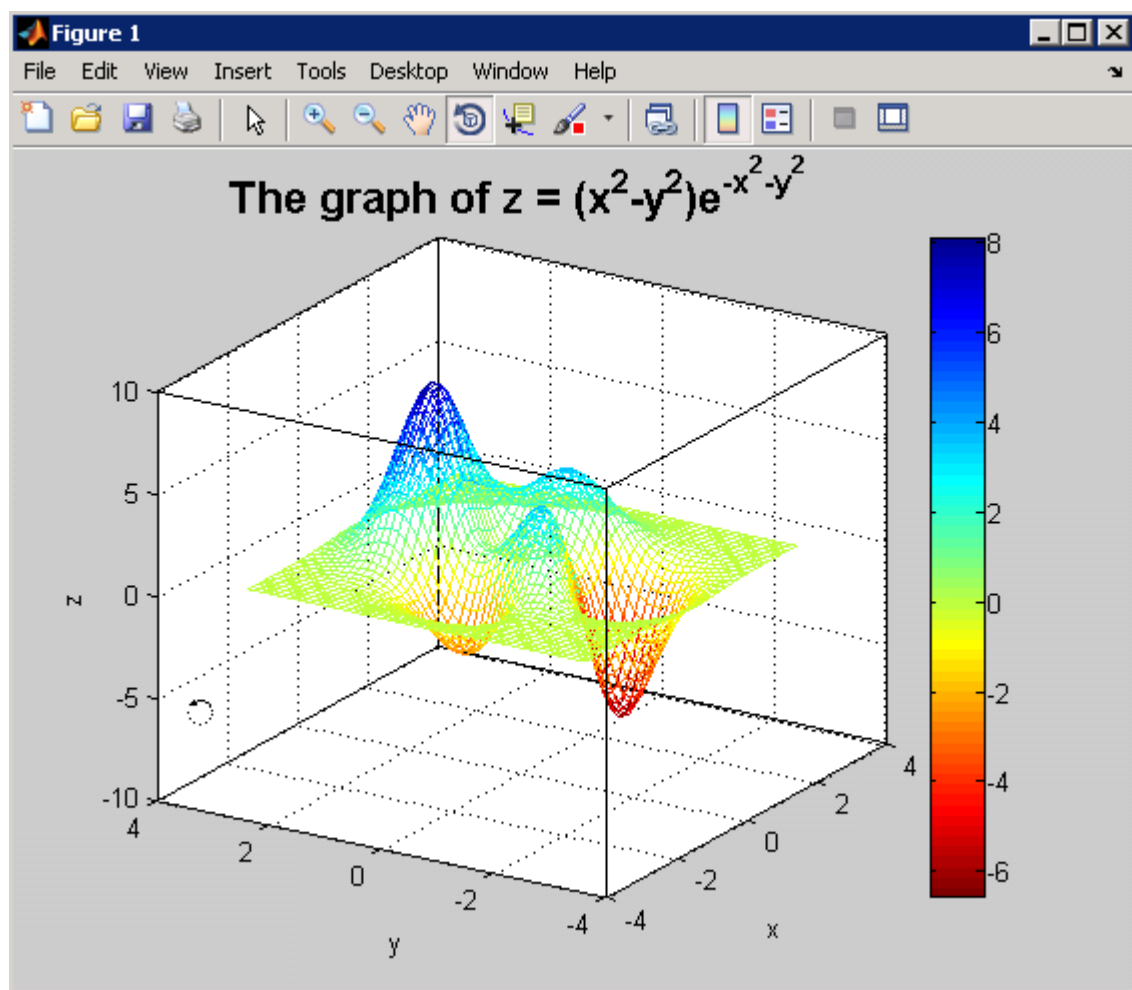
The grid is visible but we can't see trough. If we add **hidden off**:

```

EDITOR PUBLISH VIEW
1 [x, y, z]=peaks;
2 mesh(x,y,z);
3 title('The graph of z = (x^2-y^2)e^{-x^2-y^2}','fontweight',...
4     'bold','fontsize',16);
5 box on;
6 xlabel('x');ylabel('y');zlabel('z');
7 rotate3d on;view(-58,24);
8 grid on;
9 colorbar;
10 colormap(flipud(colormap));
11 %to see trough the grid
12 hidden off;

```


The result is as expected:



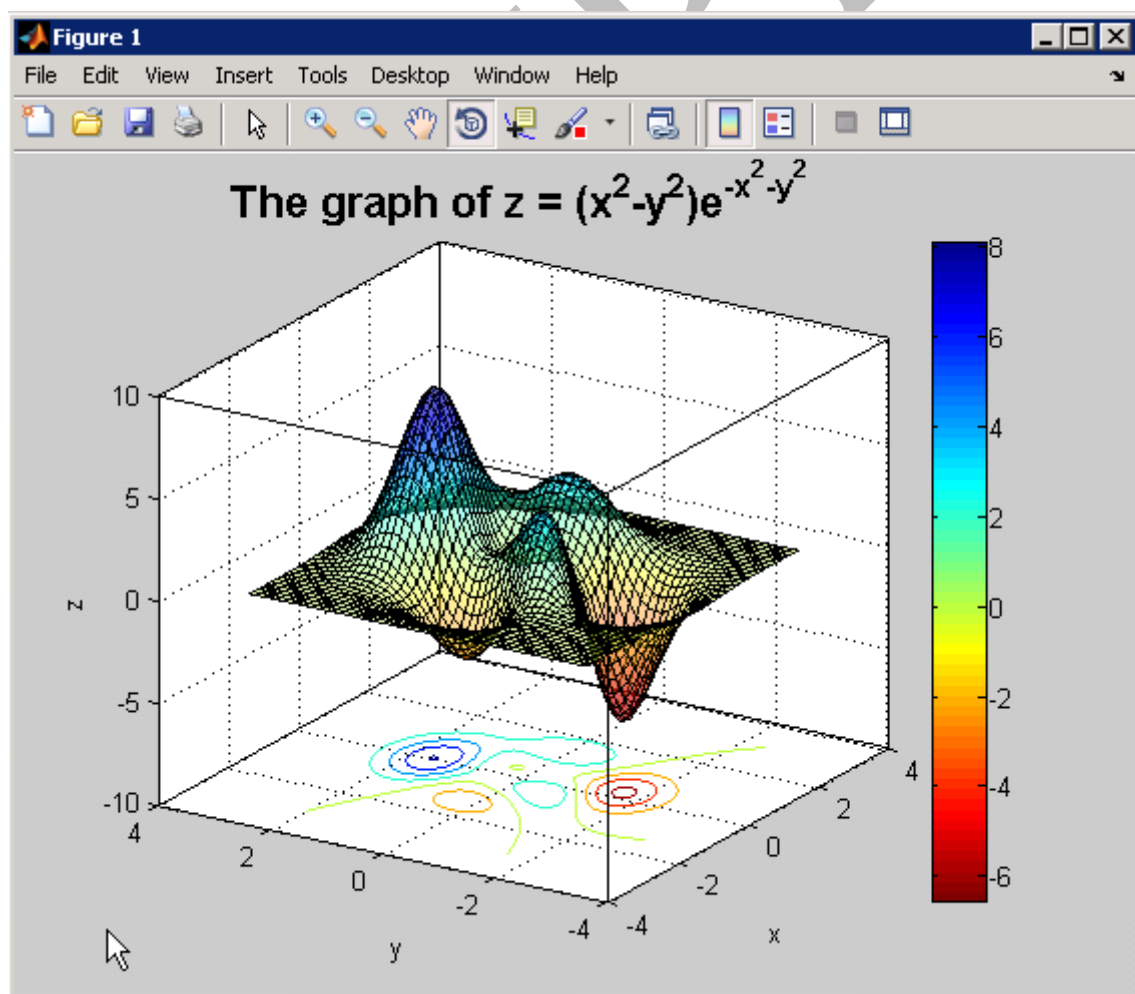
surf

For the same reason as the function **surf()** exists, you can use **surfc()** to have isoclines on the bottom plane:

```

1 - [x, y, z]=peaks;
2 - surfc(x,y,z);
3 - alpha(.4);
4 - title('The graph of  $z = (x^2-y^2)e^{-x^2-y^2}$ ','fontweight',...
5 -       'bold','fontsize',16);
6 - box on;
7 - xlabel('x');ylabel('y');zlabel('z');
8 - rotate3d on;view(-58,24);
9 - grid on;
10 - colorbar;
11 - colormap(flipud(colormap));
12 - %to see trough the grid
13 - hidden off;

```



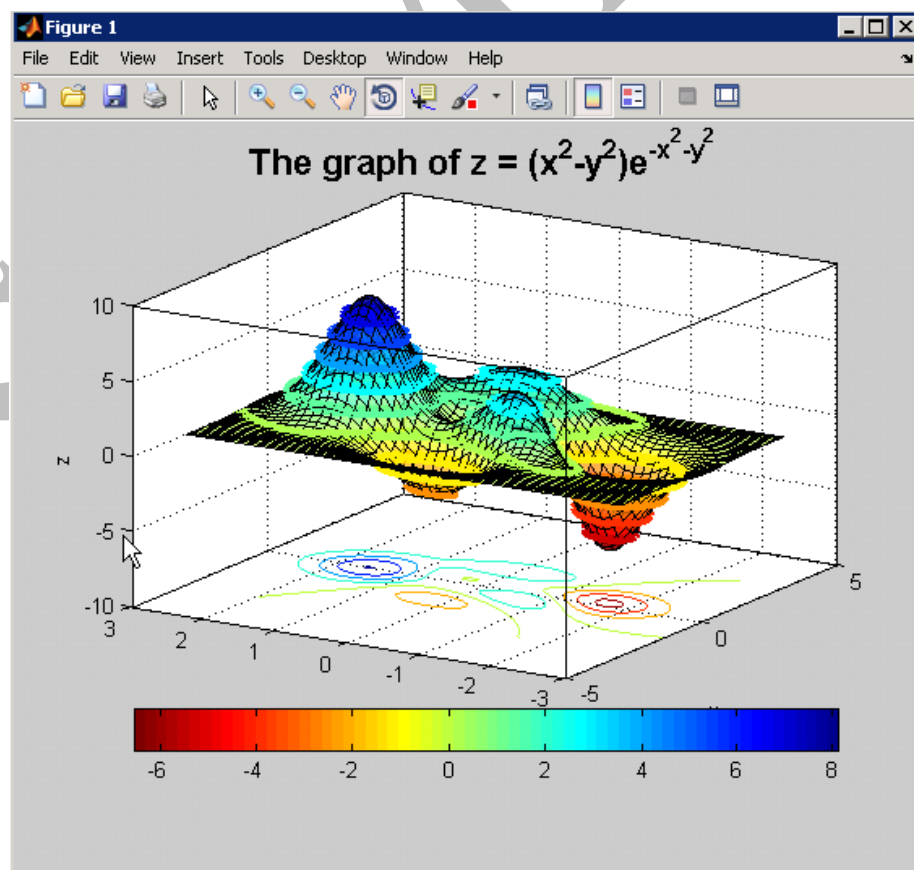
contour3 (3d contours)

The function **contour3()** is especially used by geophysicists and geometricians:

```

EDITOR PUBLISH VIEW
1 - [x, y, z]=peaks;
2 - surf(x,y,z);
3 - hold on
4 - %contours with 10 isoclines
5 - [c,h]=contour3(x,y,z,10);
6 - set(h,'linewidth',5);
7 - alpha(0.7);
8 - title('The graph of z = (x^2-y^2)e^{-x^2-y^2}','fontweight',...
9 - 'bold','fontsize',16);
10 - box on;
11 - xlabel('x');ylabel('y');zlabel('z');
12 - rotate3d on;view(-58,24);
13 - grid on;
14 - colorbar('location','southoutside');
15 - colormap(flipud(colormap));
16 - %to see trough the grid
17 - hidden off;

```

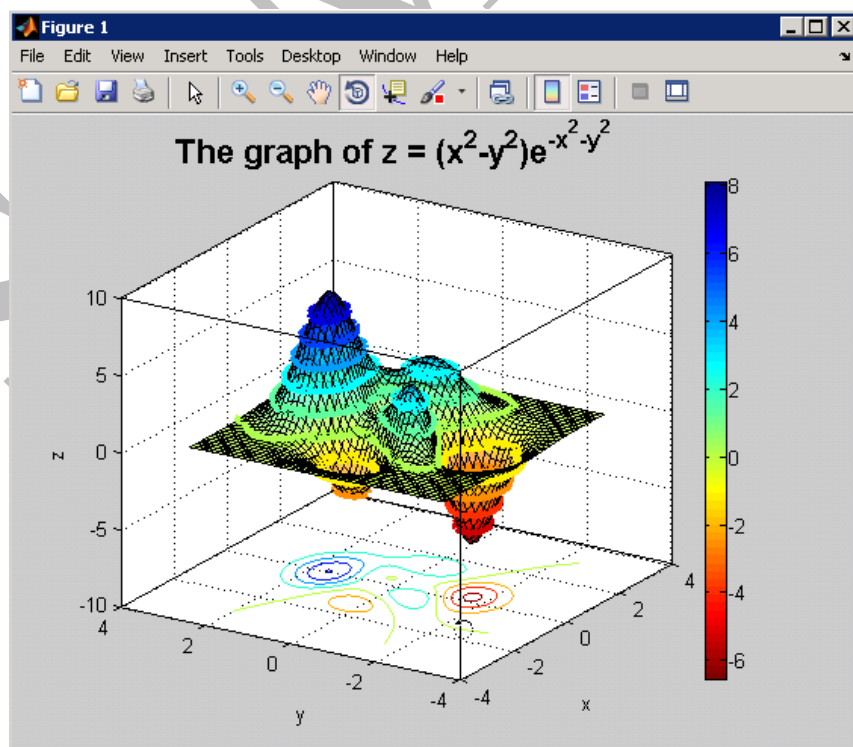


Now see the trick... to control the colorbar width (almost impossible to guess):

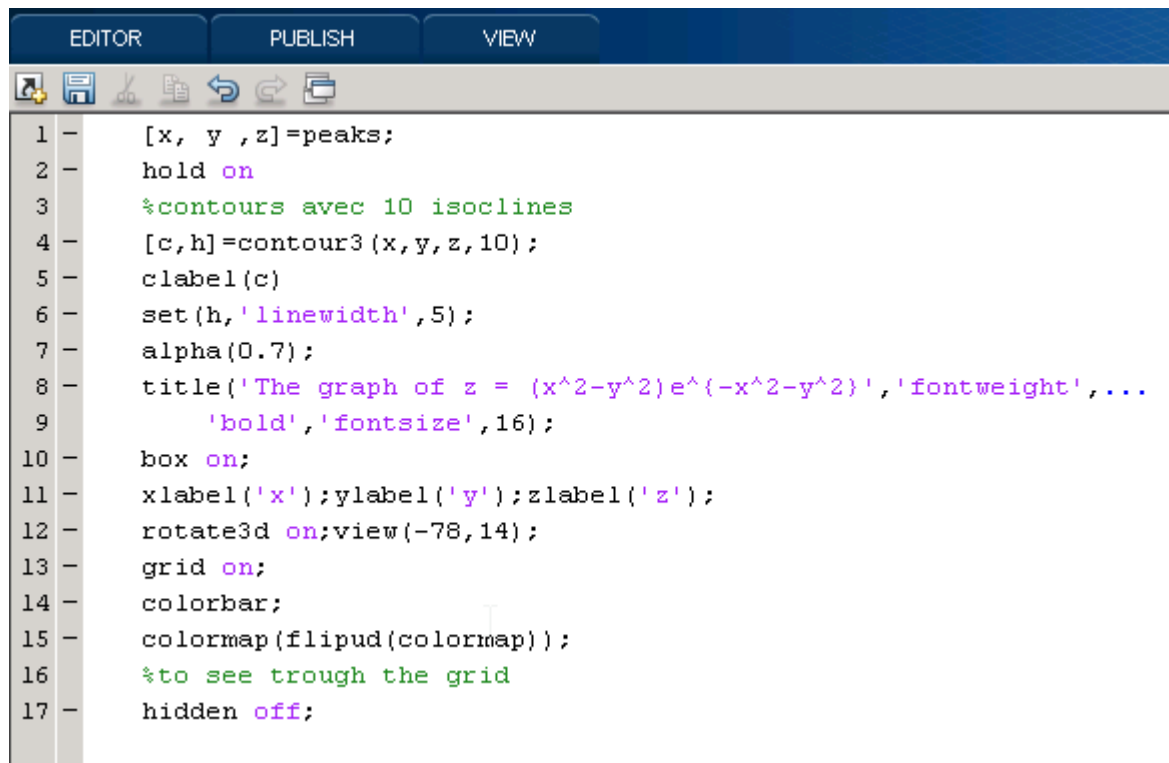
```

1 - [x, y, z]=peaks;
2 - surfc(x,y,z);
3 - hold on
4 - %contours with 10 isoclines
5 - [c,h]=contour3(x,y,z,10);
6 - set(h,'linewidth',5);
7 - alpha(0.7);
8 - title('The graph of  $z = (x^2-y^2)e^{-x^2-y^2}$ ','fontweight',...
9 -       'bold','fontsize',16);
10 - box on;
11 - xlabel('x');ylabel('y');zlabel('z');
12 - rotate3d on;view(-58,24);
13 - grid on;
14 - c =colorbar;
15 - axpos = get(gca,'Position');
16 - cpos = get(c,'Position');
17 - cpos(3) = 0.5*cpos(3);
18 - set(c,'Position',cpos)
19 - set(gca,'Position',axpos)
20 - colormap(flipud(colormap));
21 - %to see trough the grid
22 - hidden off;

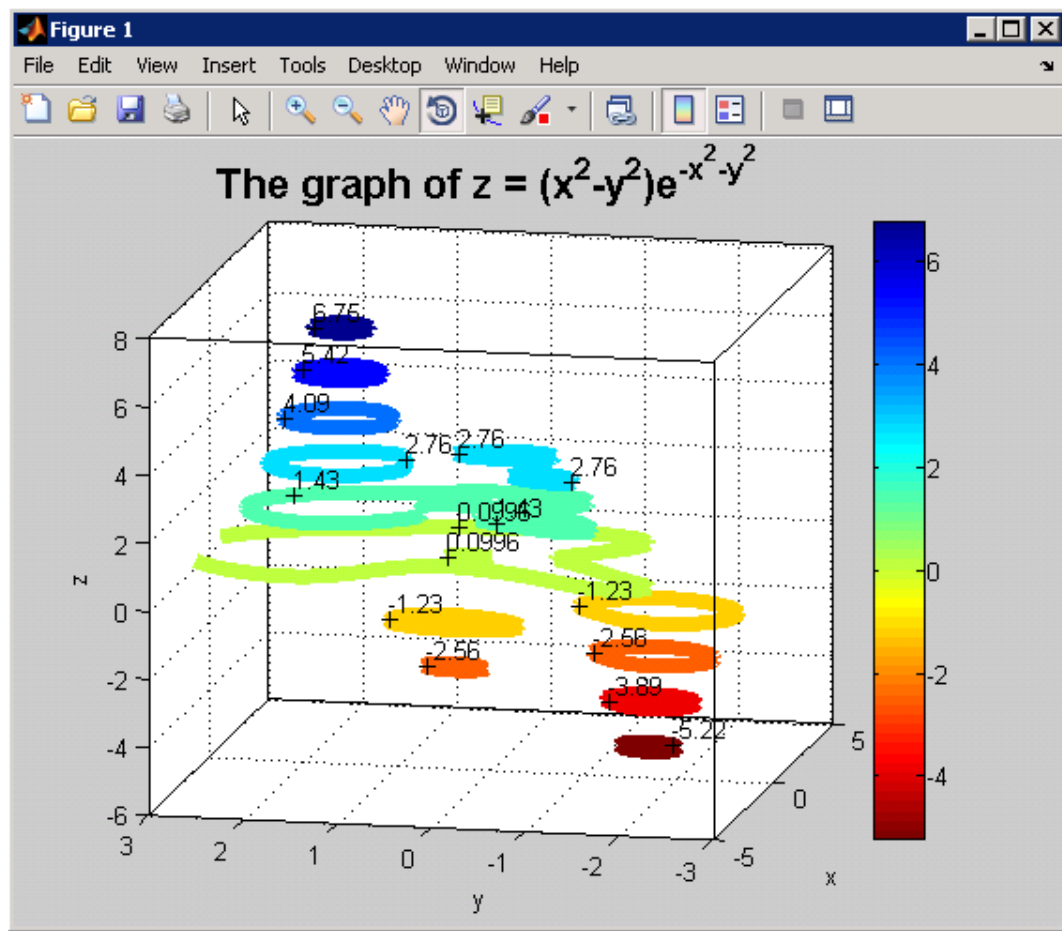
```



Most time geophysicists and geometricians don't want the surface but just the isoclines. Then it is very easy. You just remove the **surf()** or **surfc()** function of your script and to make the values of the contours on the chart you use the function **clabel()**:

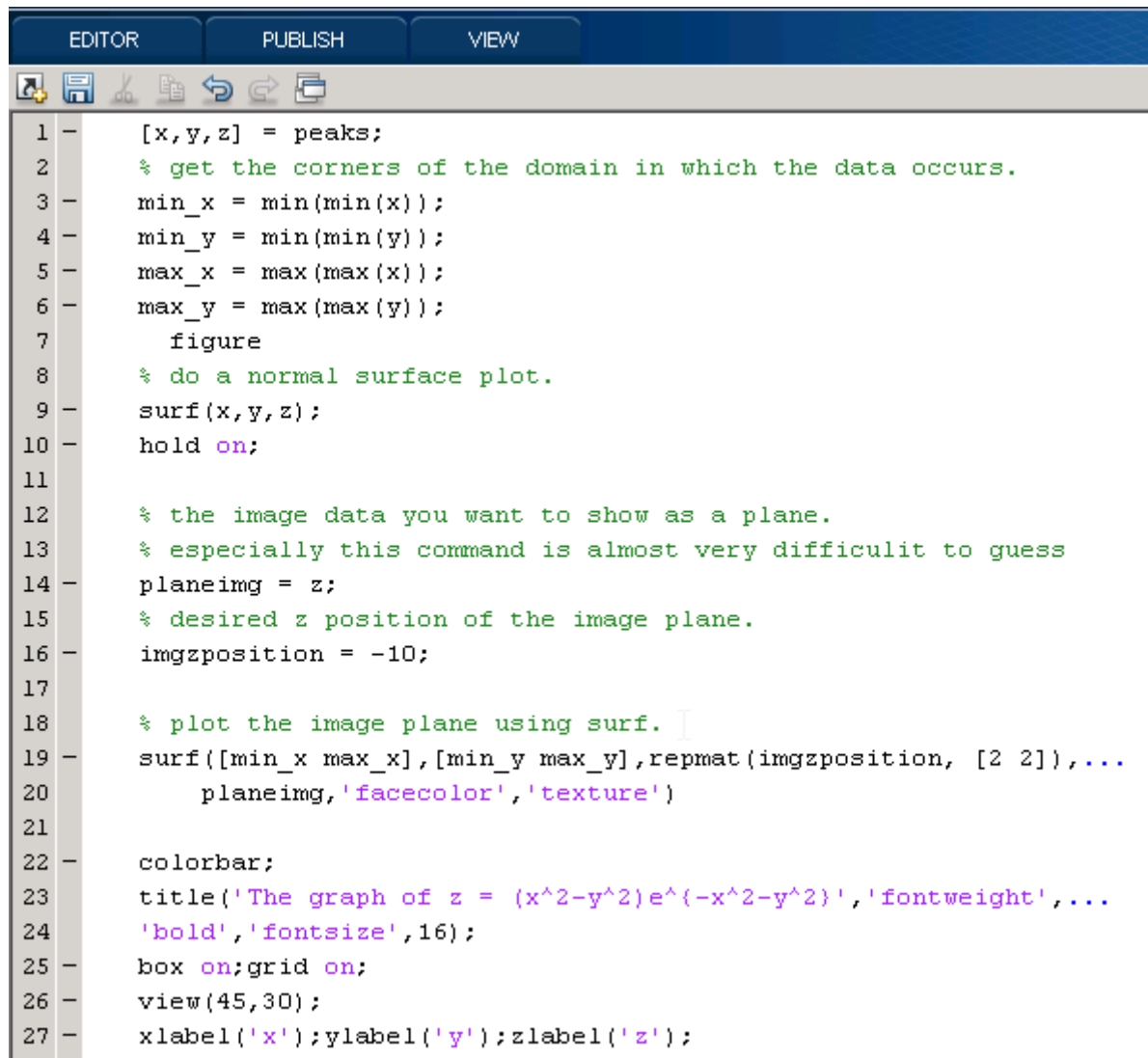


```
1 - [x, y, z]=peaks;
2 - hold on
3 - %contours avec 10 isoclines
4 - [c,h]=contour3(x,y,z,10);
5 - clabel(c)
6 - set(h,'linewidth',5);
7 - alpha(0.7);
8 - title('The graph of z = (x^2-y^2)e^{-x^2-y^2}','fontweight',...
9 -       'bold','fontsize',16);
10 - box on;
11 - xlabel('x');ylabel('y');zlabel('z');
12 - rotate3d on;view(-78,14);
13 - grid on;
14 - colorbar;
15 - colormap(flipud(colormap));
16 - %to see through the grid
17 - hidden off;
```



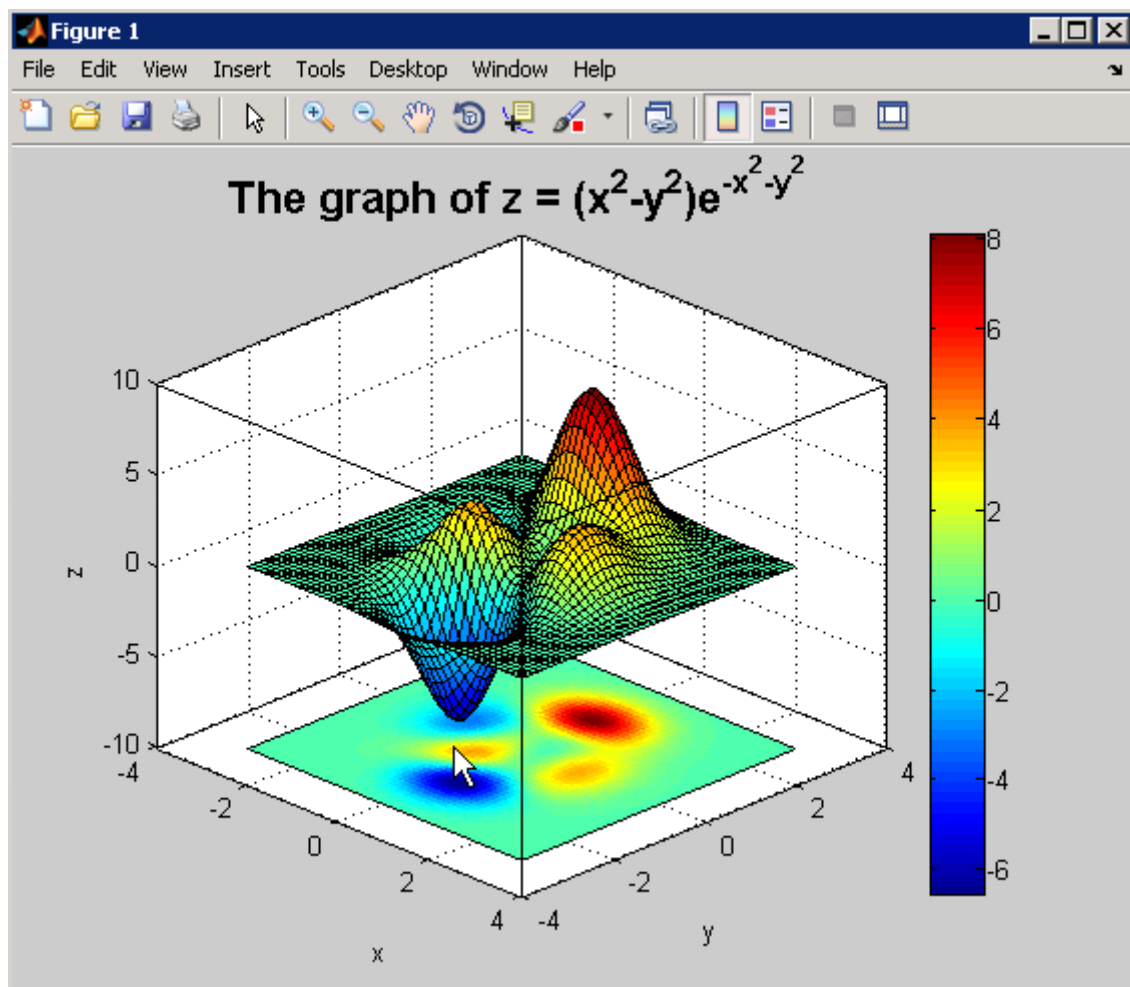
surf with surf projection

Many geophysicists and physicists need to have a surface plot with at the same time a surface density plot projection. This options does not exists in MATLAB then we have to use a trick to get the expected result:



```
1 - [x,y,z] = peaks;
2 - % get the corners of the domain in which the data occurs.
3 - min_x = min(min(x));
4 - min_y = min(min(y));
5 - max_x = max(max(x));
6 - max_y = max(max(y));
7 - figure
8 - % do a normal surface plot.
9 - surf(x,y,z);
10 - hold on;
11
12 - % the image data you want to show as a plane.
13 - % especially this command is almost very difficult to guess
14 - planeimg = z;
15 - % desired z position of the image plane.
16 - imgzposition = -10;
17
18 - % plot the image plane using surf.
19 - surf([min_x max_x],[min_y max_y],repmat(imgzposition, [2 2]),...
20 -     planeimg,'facecolor','texture')
21
22 - colorbar;
23 - title('The graph of  $z = (x^2 - y^2)e^{-(x^2 - y^2)}$ ','fontweight',...
24 -     'bold','fontsize',16);
25 - box on;grid on;
26 - view(45,30);
27 - xlabel('x');ylabel('y');zlabel('z');
```

The result is on the next page:



contour (2d contours)

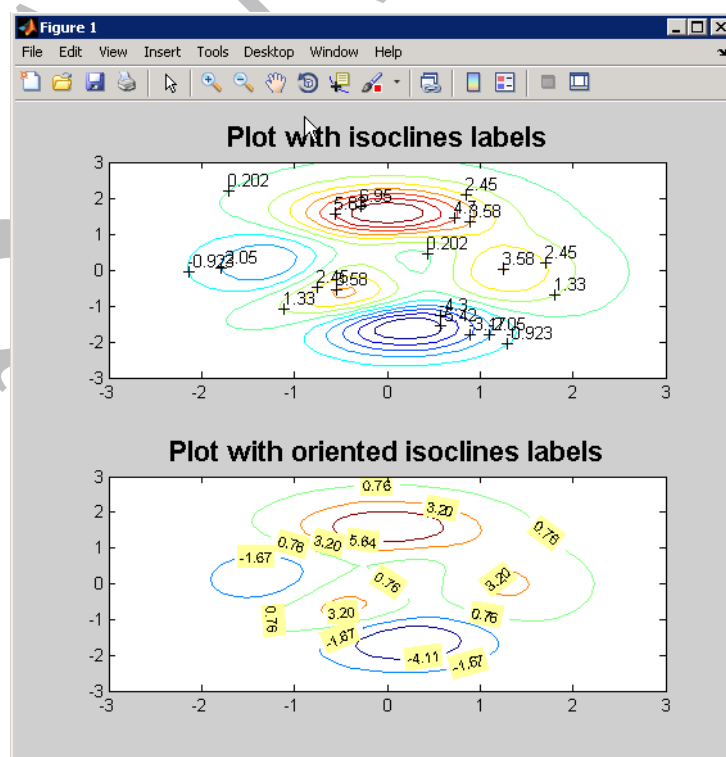
The function **contour()** works in the same idea as the **contour3()** that we was before, the difference is that this first function do the job in a projected XY plane:

```

1 subplot(2,1,1);
2 [X,Y,Z]=peaks;
3 %12 contour lines
4 C=contour(X,Y,Z,12);
5 clabel(C)
6 title('Plot with isoclines labels','fontweight','bold','fontsize',16)
7
8 subplot(2,1,2)
9 %5 oriented contour lines
10 [C,h]=contour(X,Y,Z,5);
11 textx=clabel(C,h,'fontsize',8)
12 set(textx,'BackgroundColor',[1 1 .6])
13 %loop for decimal precision control of labels...
14 for i=1:size(textx)
15     textstr=get(textx(i),'String');
16     textnum=str2double(textstr);
17     textstrnew=sprintf('%0.2f',textnum);
18     set(textx(i),'String',textstrnew);
19 end
20 title('Plot with oriented isoclines labels','fontweight',...
21       'bold','fontsize',16)

```

The result looks like:



Note: It seems that we can't actually control easily the number of labels that appears automatically on a isocline (contour)

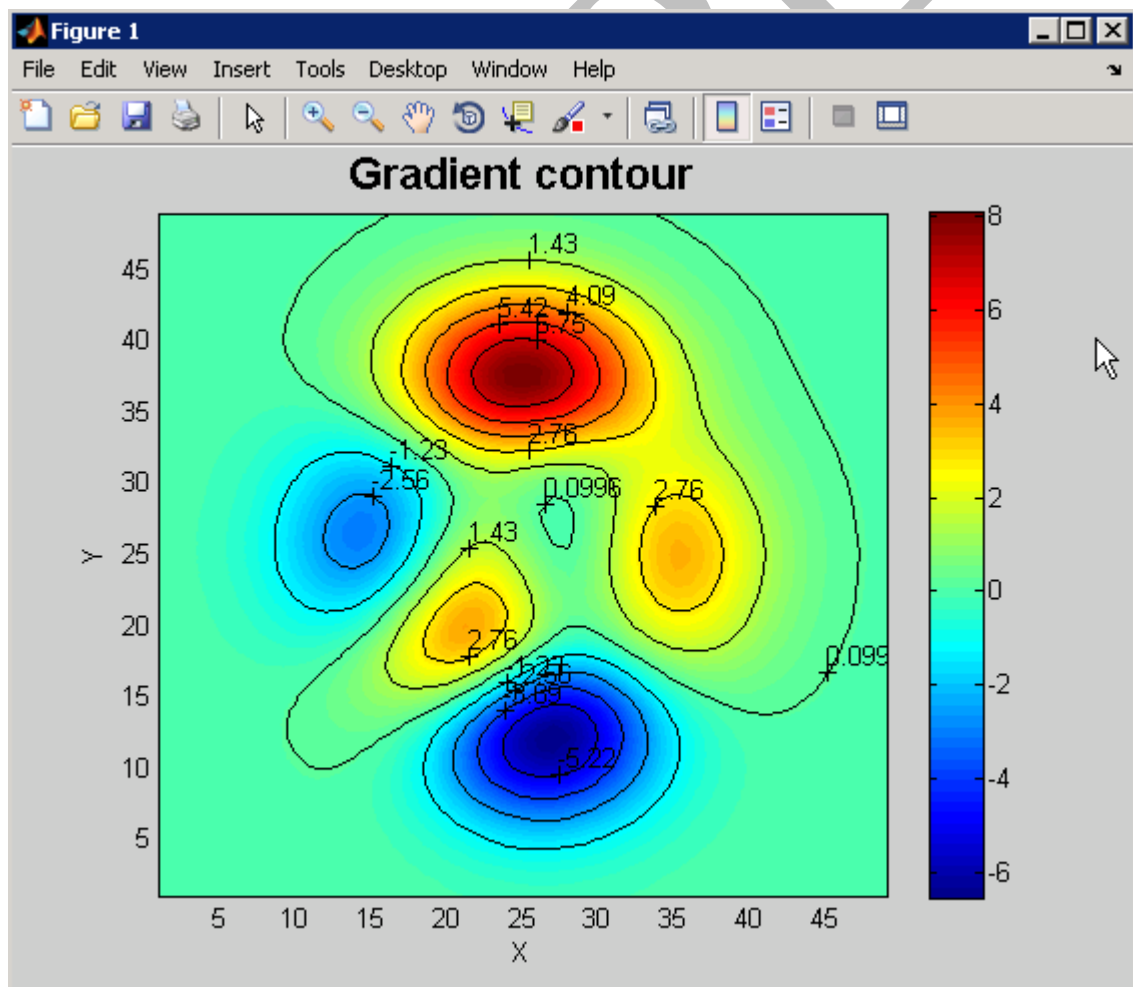
On a more pretty way using a colorbar and also a shading interpolation:

```

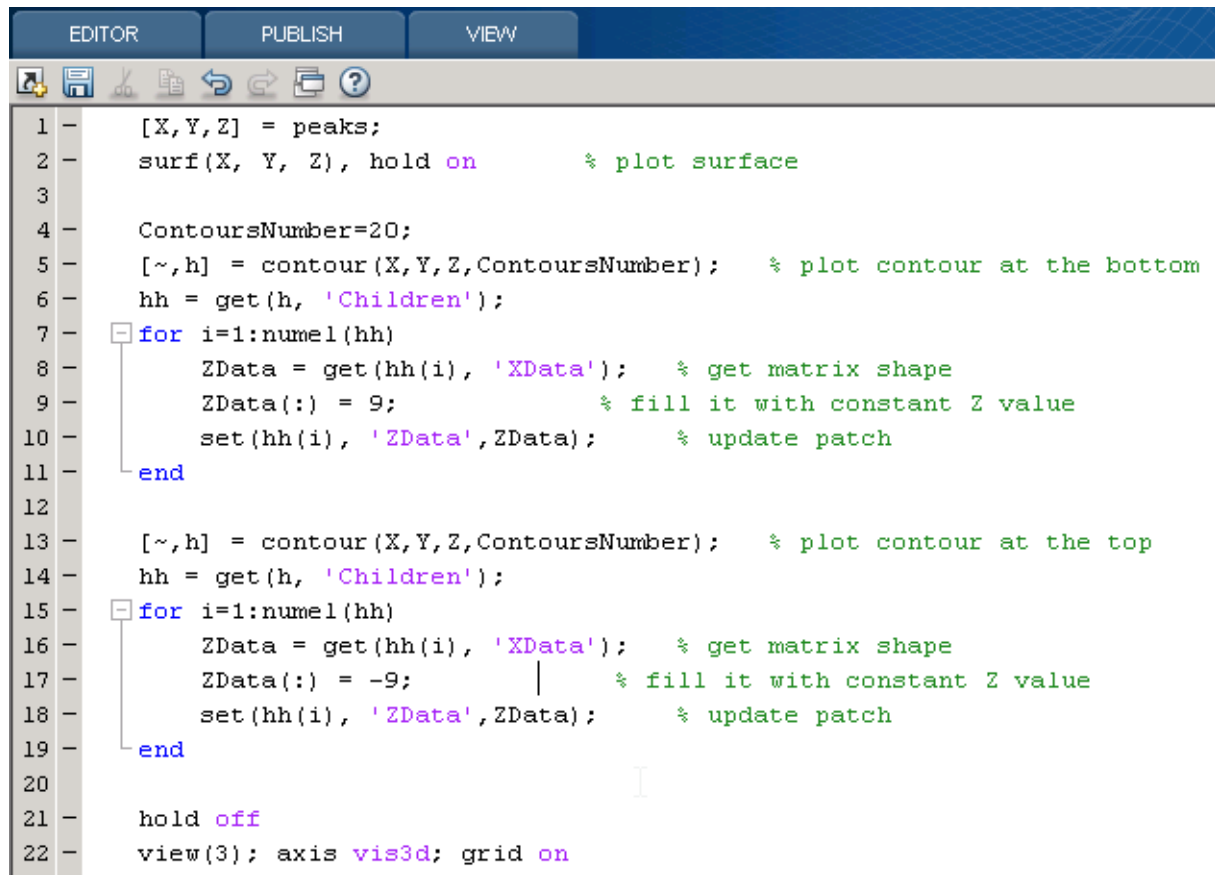
1  h=peaks;
2  pcolor(h);
3  colorbar;
4  grid on;
5  shading interp;
6  hold on;
7  C=contour(h,10,'k');
8  clabel(C);
9  title('Gradient contour','fontweight','bold','fontsize',16);
10 xlabel('X'),ylabel('Y')

```

This will give (ToDo: found how to activate the grid...):



2D contour can also be used to facilitate the readiness of a 3D plot as you can see below:

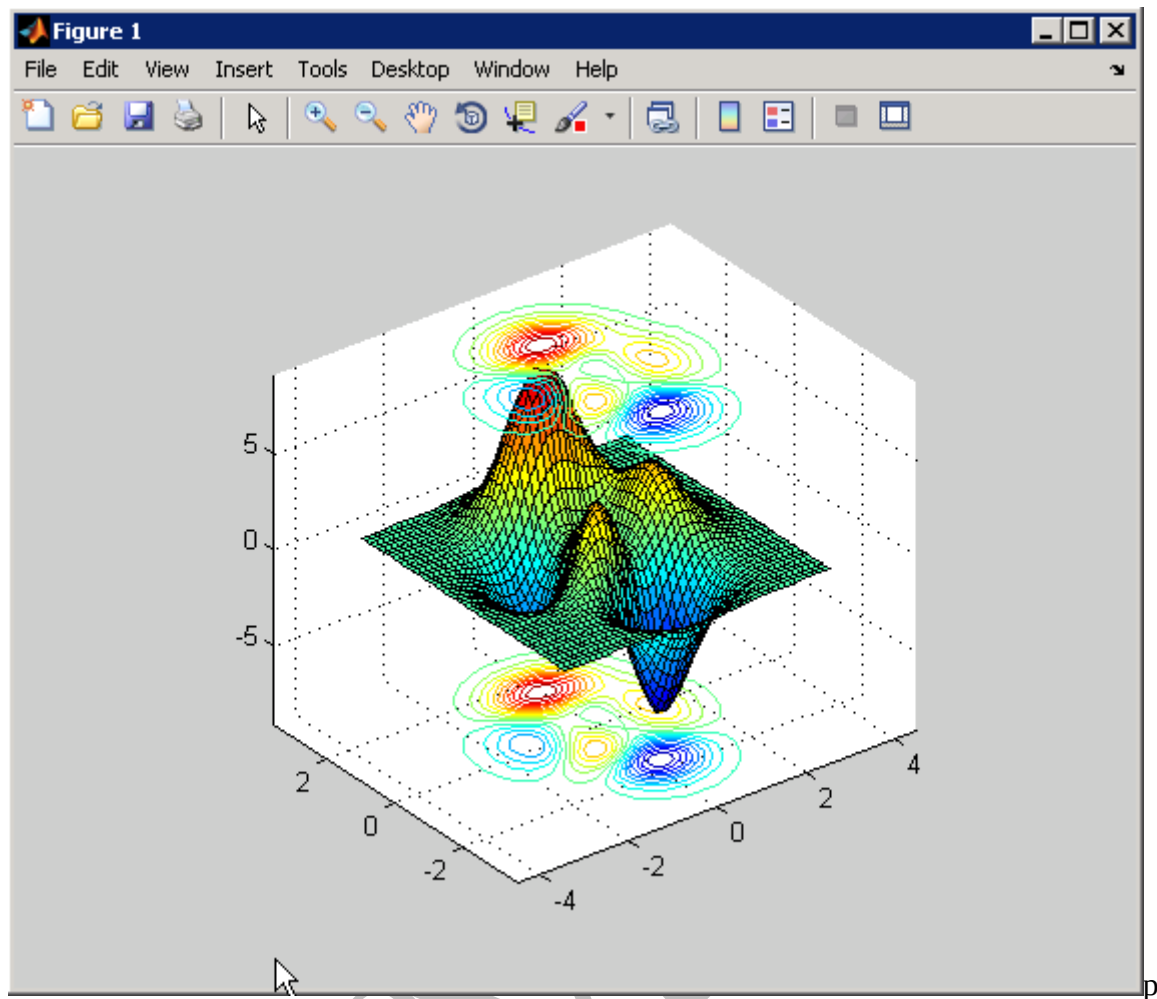


```

1 - [X,Y,Z] = peaks;
2 - surf(X, Y, Z), hold on           % plot surface
3
4 - ContoursNumber=20;
5 - [~,h] = contour(X,Y,Z,ContoursNumber); % plot contour at the bottom
6 - hh = get(h, 'Children');
7 - for i=1:numel(hh)
8 -     ZData = get(hh(i), 'ZData'); % get matrix shape
9 -     ZData(:) = 9;                % fill it with constant Z value
10 -    set(hh(i), 'ZData',ZData);    % update patch
11 - end
12
13 - [~,h] = contour(X,Y,Z,ContoursNumber); % plot contour at the top
14 - hh = get(h, 'Children');
15 - for i=1:numel(hh)
16 -     ZData = get(hh(i), 'ZData'); % get matrix shape
17 -     ZData(:) = -9;               % fill it with constant Z value
18 -     set(hh(i), 'ZData',ZData);  % update patch
19 - end
20
21 - hold off
22 - view(3); axis vis3d; grid on

```

That will give:



waterfall

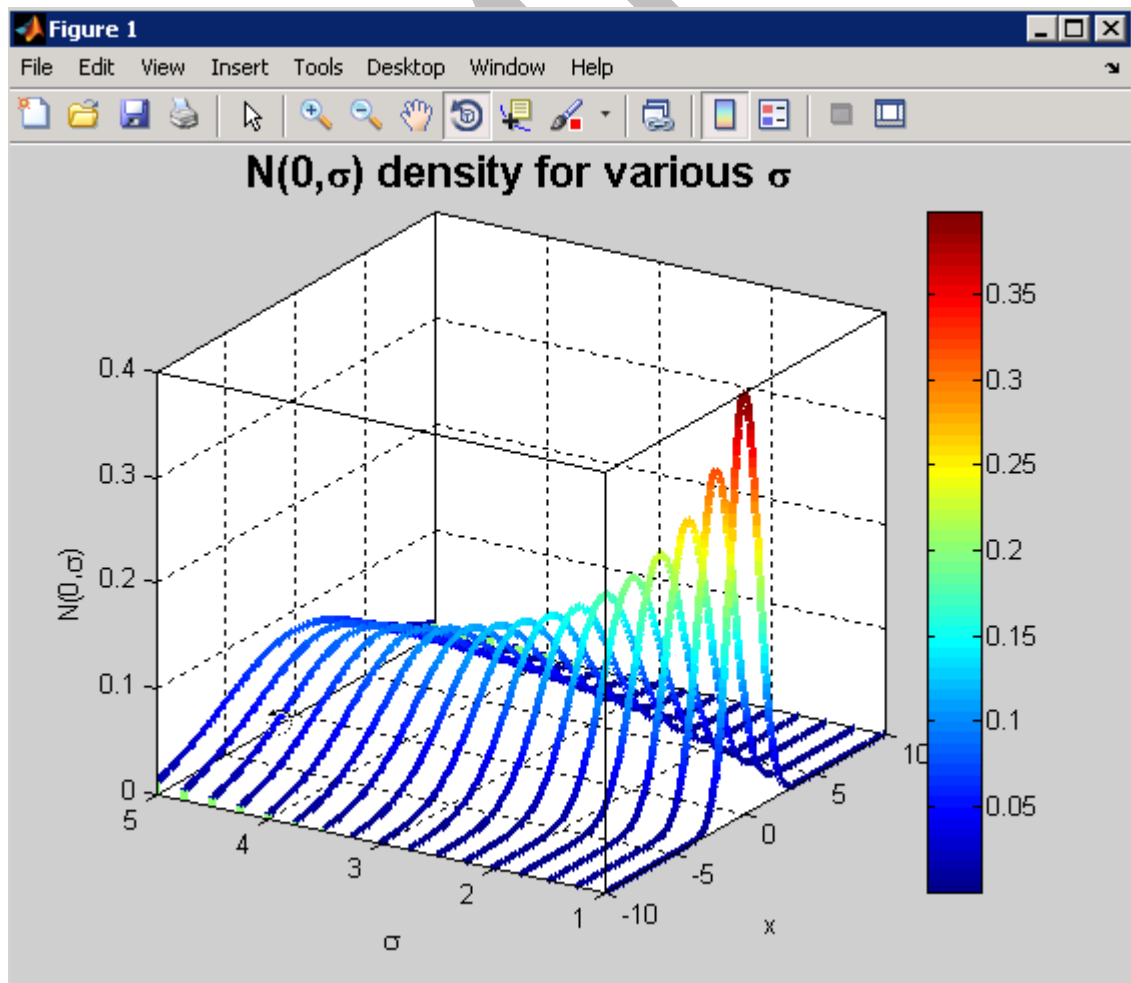
The **waterfall()** function is especially used in statistics to show the hypothesized density distribution function evolution of a regression or in finance to show how a distribution change with the time:

```

1 - dsig = 0.25; dx = 0.05; mu = 0;
2 - [X,SIGMA] = meshgrid(-10:dx:10,1:dsig:5);
3 - Z = exp(-(X-mu).^2./(2*SIGMA.^2))./sqrt(2*pi*SIGMA.^2);
4 - f=waterfall(X,SIGMA,Z);
5 - set(f, 'LineWidth',3);
6 - title('N(0,\sigma) density for various \sigma','fontweight','bold','fontsize',16)
7
8 - box on;
9 - xlabel('x');ylabel('\sigma');zlabel('N(0,\sigma)')
10 - rotate3d on;view(-58,24);
11 - grid on;
12 - colorbar;
13 - %to see trough the grid
14 - hidden off;

```

The result will be:



2D Gradient (quiver)

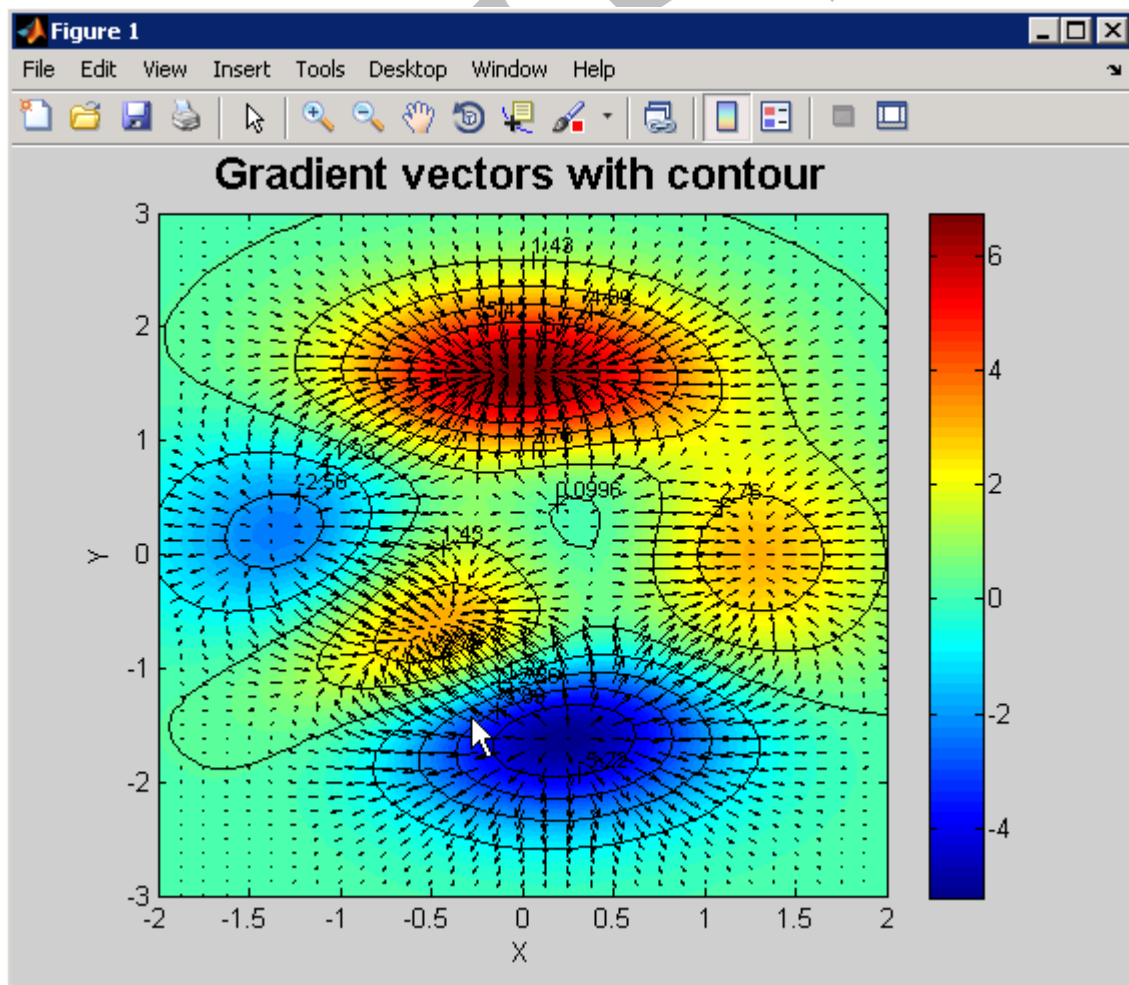
Gradient is mostly used by physicists in electrodynamics and various field implying gaz, liquids or plasmas. To obtain a gradient plot in MATLAB we have to mix the functions **gradient()** and **quiver()**:

```

1  [X,Y,Z]=peaks;
2  [DX,DY]=gradient(Z);
3  [C,h]=contour(X,Y,Z,10,'k');
4  texth=clabel(C,'fontsize',8);
5  hold on;
6  quiver(X,Y,DX,DY,2,'k');
7  pcolor(X,Y,Z);
8  shading interp;
9  colorbar;
10 axis([-2 2 -3 3]);
11 title('Gradient vectors with contour','fontweight','bold','fontsize',16);
12 xlabel('X');ylabel('Y');

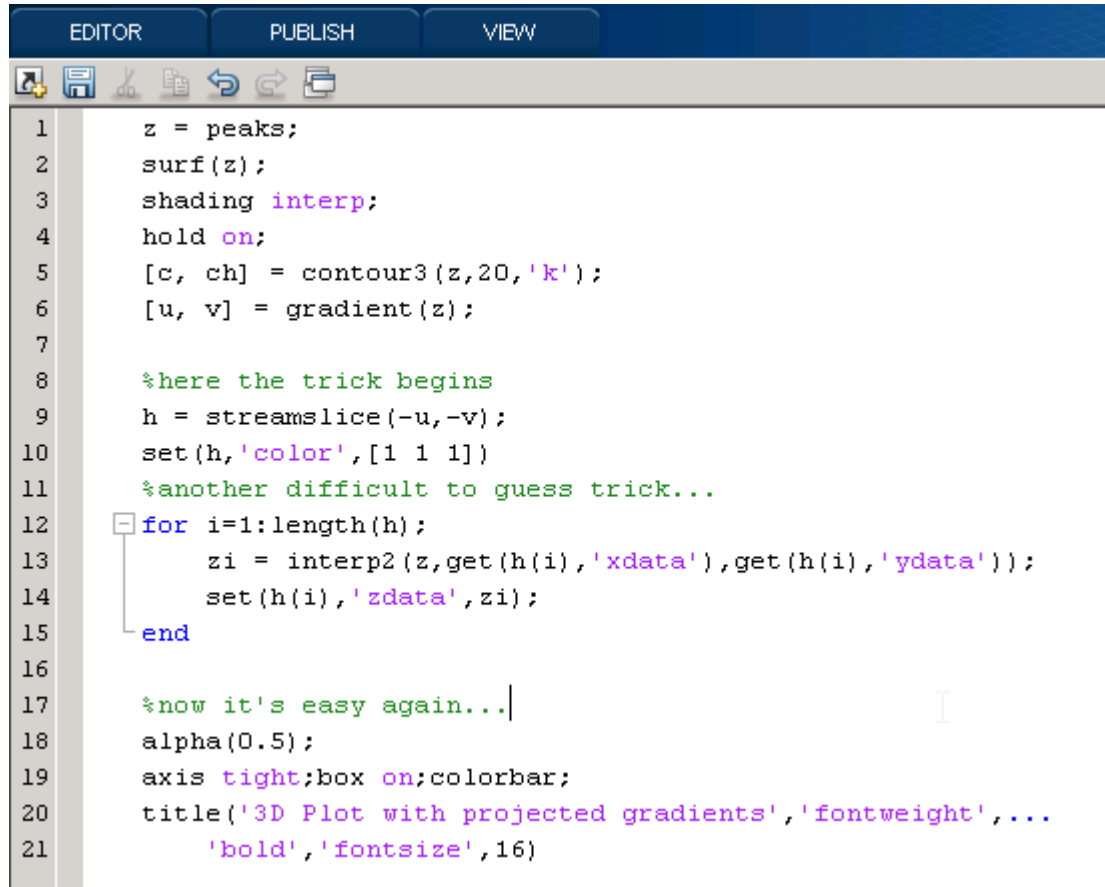
```

The result will be:



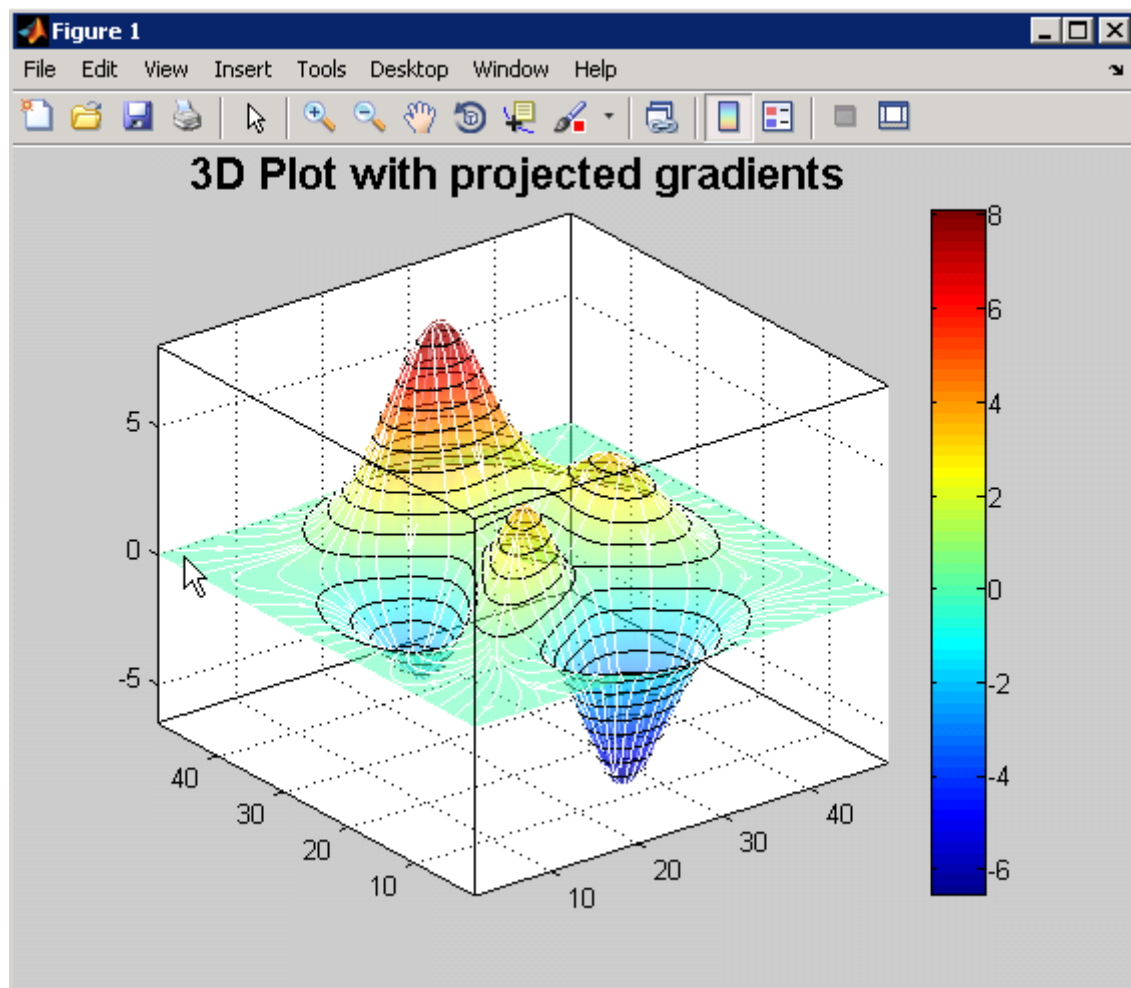
3D Gradient

A 3D gradient is most of time much more useful than a 2D one but the problem in MATLAB (as far as I know) is that there are no simple functions to get that. We need to apply a cook receipt using in a subtle way the **interp2()** function to get the expected result



```
1      z = peaks;
2      surf(z);
3      shading interp;
4      hold on;
5      [c, ch] = contour3(z,20,'k');
6      [u, v] = gradient(z);
7
8      %here the trick begins
9      h = streamslice(-u,-v);
10     set(h,'color',[1 1 1])
11     %another difficult to guess trick...
12     for i=1:length(h);
13         zi = interp2(z,get(h(i),'xdata'),get(h(i),'ydata'));
14         set(h(i),'zdata',zi);
15     end
16
17     %now it's easy again...|
18     alpha(0.5);
19     axis tight;box on;colorbar;
20     title('3D Plot with projected gradients','fontweight',...
21         'bold','fontsize',16)
```

You can see the result on the next page (see the white lines that are the gradients perpendicular to the isoclines):



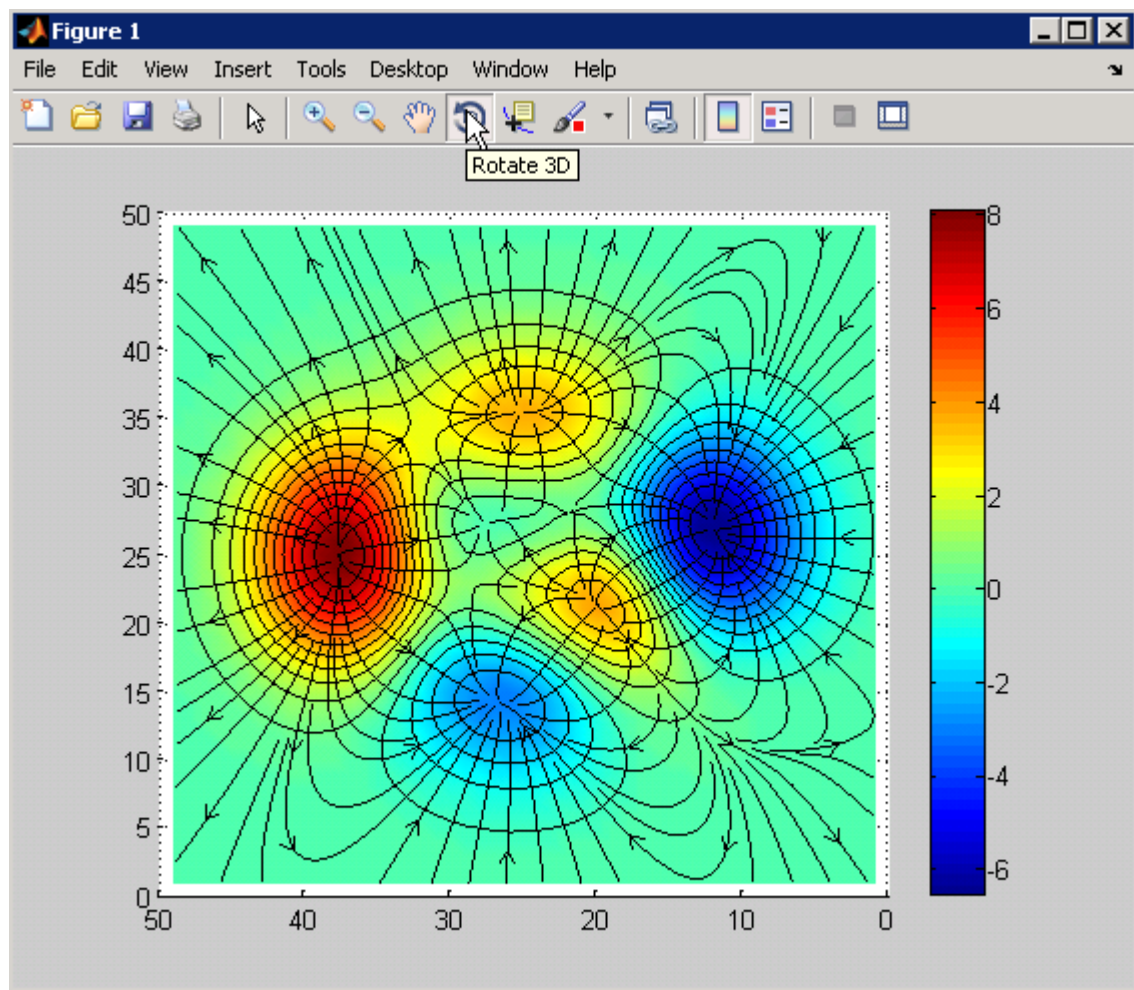
We can also play with the perspective:

```

EDITOR    PUBLISH    VIEW
1 -      z=peaks;
2 -      surf(z);
3 -      shading interp;
4 -      hold on;
5 -      [c,ch]=contour3(z,20,'k');
6 -      [u,v]=gradient(z);
7 -      h=streamslice(-u,-v);
8 -      set(h,'Color',[0 0 0]);
9 -      colorbar;
10
11 -  for k=1:length(h)
12 -      zk=interp2(z,get(h(k),'xdata'),get(h(k),'ydata'));
13 -      set(h(k),'zdata',zk);
14 -  end
15
16 -      rotate3d on;
17 -      view(-90,90);

```


To get:



Perpendicular vectors

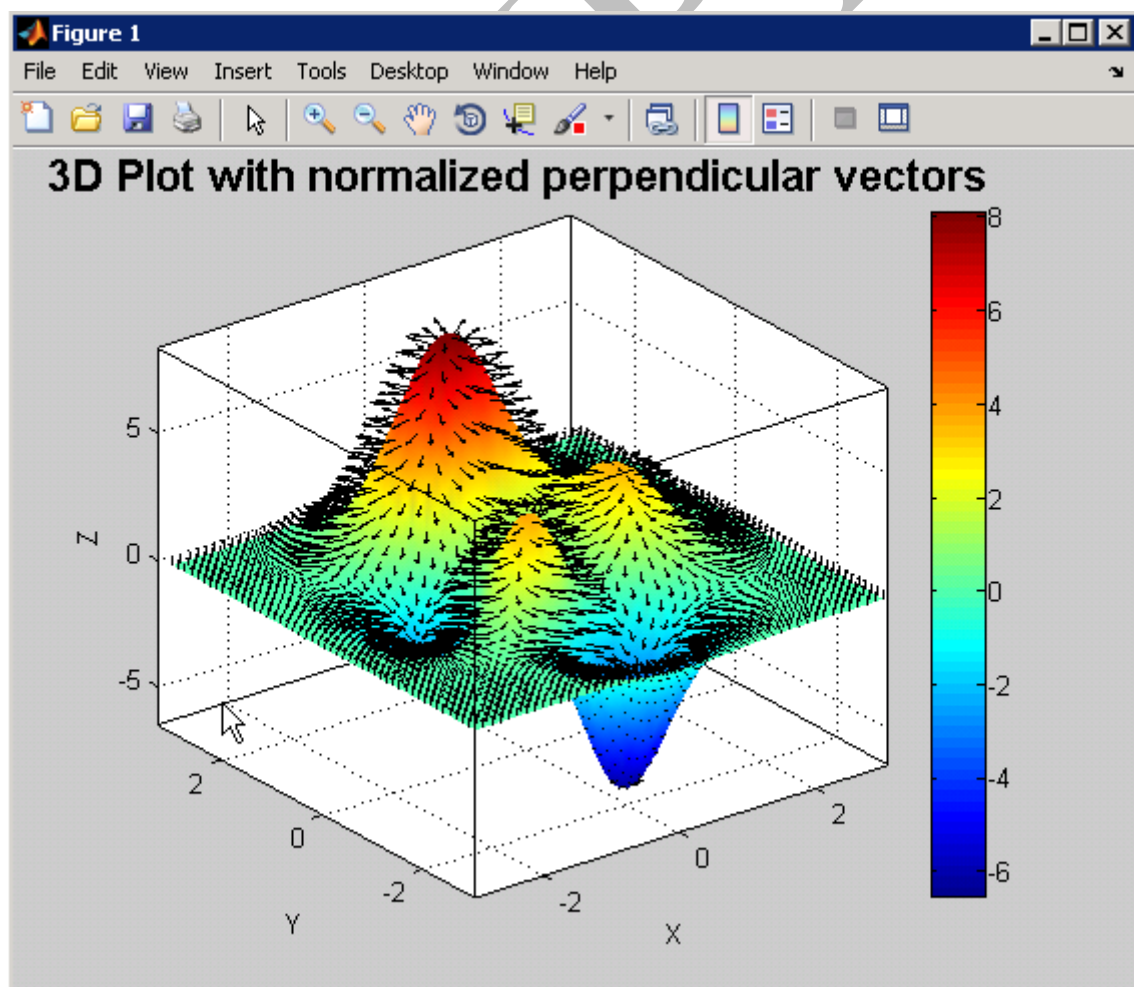
Until today I never saw anyone using this type of chart then I will just a simple example of the **surfnorm()** function associated with **quiver3()** this last one being used a lot in fluid mechanics:

```

1      [X,Y,Z]=peaks;
2      [U,V,W]= surfnorm(X,Y,Z);
3      surf(X,Y,Z)
4      hold('on')
5      quiver3(X,Y,Z, U,V,W, 1, 'k');
6      colorbar;
7      shading interp;
8      axis tight;box on;colorbar;
9      title('3D Plot with normalized perpendicular vectors','fontweight',...
10          'bold','fontsize',16)
11      xlabel('X');ylabel('Y');zlabel('Z');

```

The result will be:



Surface plots of nonuniformly sampled data

This is the most important example of 3D surface plot for people working in laboratories because most of time we have three vector and for each $\{x, y\}$ we have a z .

Consider:

```

EDITOR PUBLISH VIEW
1 %we generate something random
2 x = rand(100,1)*16 - 8;
3 y = rand(100,1)*16 - 8;
4 r = sqrt(x.^2 + y.^2) + eps;
5 z = sin(r)./r;

```

Cubic (triangle base interpolation)

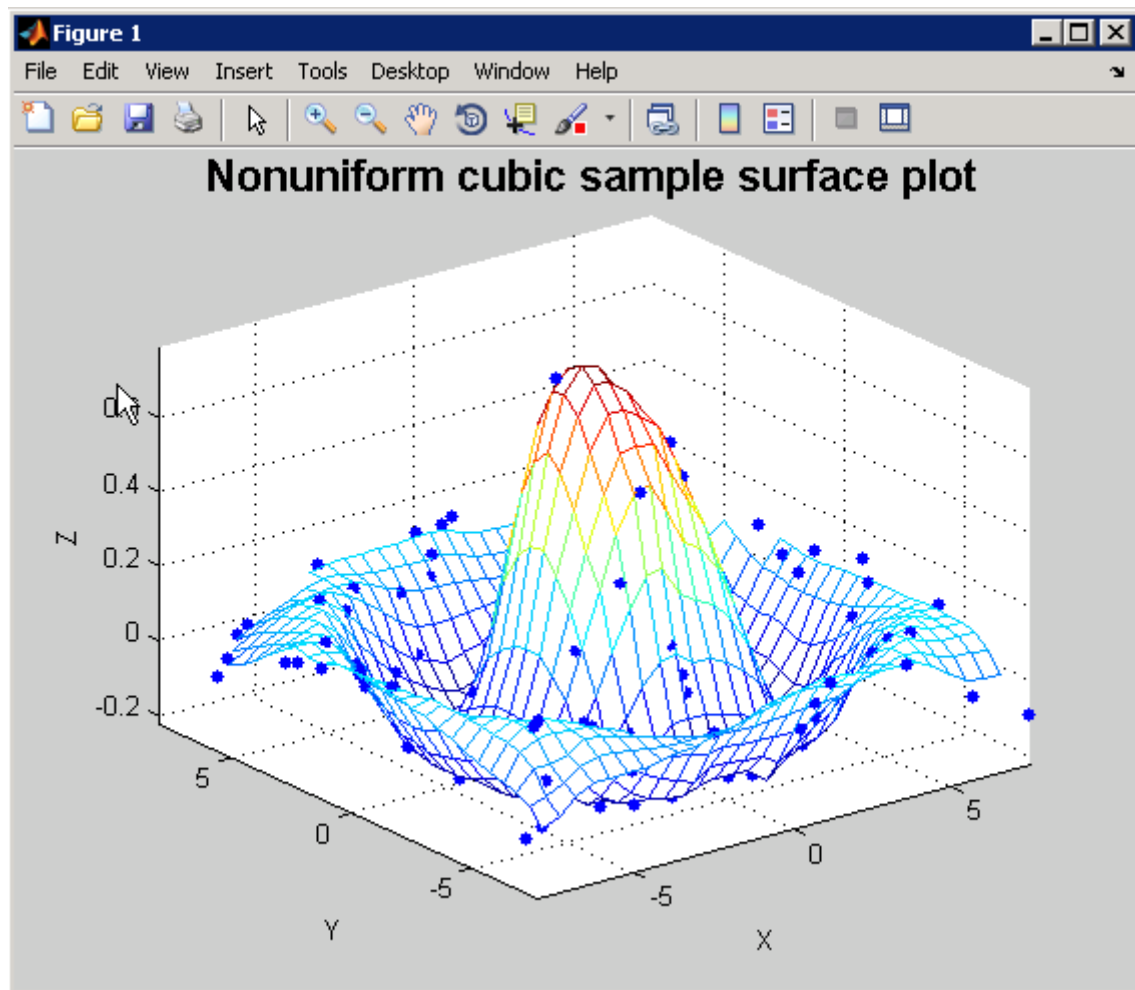
The 1st method is to transform your non-uniform vectors into a uniform grid thanks to the combination of **linspace()** and **meshgrid()**. After the difficult trick is to use the function **griddata()** to prepare a **mesh()**. Finally we use a **plot3()** to plot the points:

```

EDITOR PUBLISH VIEW
7 %we construct the ranges and the number of steps for the interpolation
8 xlin = linspace(min(x),max(x),30);
9 ylin = linspace(min(y),max(y),30);
10
11 %Now we use these points to generate a uniformly spaced grid.
12 [X,Y] = meshgrid(xlin,ylin);
13
14 %The key to this process is to use griddata to interpolate the values of
15 %the function at the uniformly spaced points, based on the values of the
16 %function at the original data points (which are random in this example).
17 %This statement uses a triangle-based cubic interpolation to generate
18 %the new data.
19 Z = griddata(x,y,z,X,Y,'cubic'); %linear, cubic (see Matlab help for others)
20
21 mesh(X,Y,Z) %interpolated
22 axis tight; hold on
23 plot3(x,y,z,'.','MarkerSize',15) %nonuniform
24
25 xlabel('X');ylabel('Y'),zlabel('Z');
26 title('Nonuniform cubic sample surface plot','fontweight',...
27       'bold','fontsize',16)

```

The result will look like (see that we can have a grid that has more vertices than the original vectors sample points!):



Note: 'cubic' doesn't mean "(bi-)cubic spline"! if you edit the code of the function griddata you will be able to see that:

```

235 %-----
236 function vq = cubic(x,y,v,xq,yq)
237 %TRIANGLE Triangle-based cubic interpolation
238
239 % Reference: T. Y. Yang, "Finite Element Structural Analysis",
240 % Prentice Hall, 1986. pp. 446-449.
241 %
242 % Reference: David F. Watson, "Contouring: A guide
243 % to the analysis and display of spacial data", Pergamon, 1994.
244
245 % Triangulate the data
246
247 [x, y, v] = mergepoints2D(x,y,v);
248
249 dt = delaunayTriangulation(x,y);
250 scoredWarnOff = warning('off', 'MATLAB:triangulation:EmptvTri2DWarnId');

```

Delaunay (triangular connections)

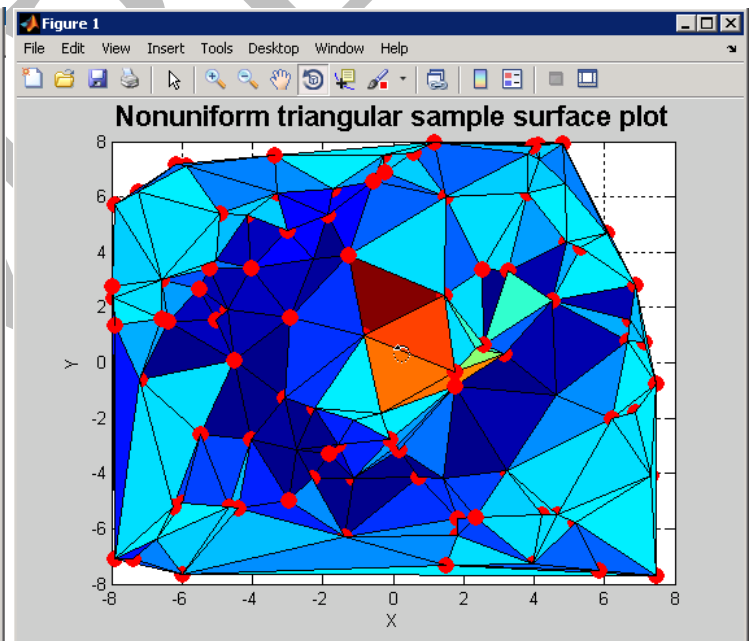
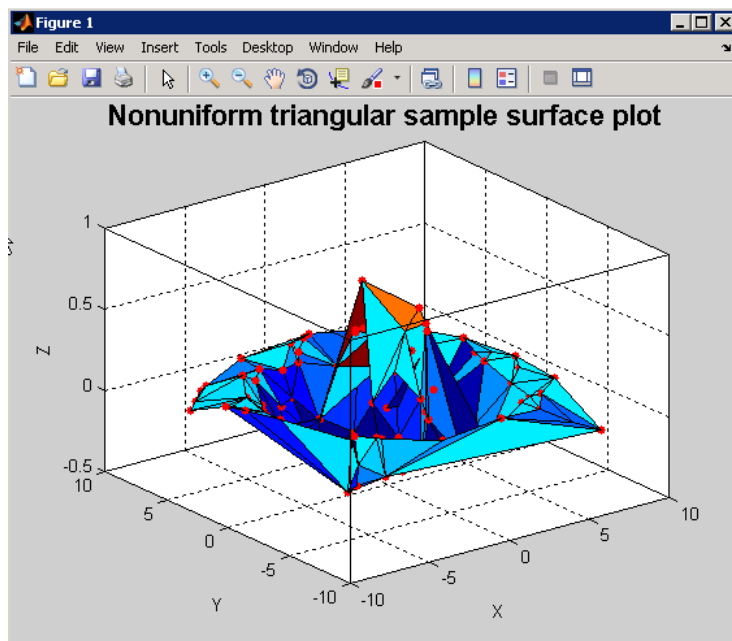
The 2nd method below using the function **delaunay()** and **trisurf()** is easier to implement (shorter) but also very difficult to guess:

```

29     p_h=plot3(x,y,z,'r','MarkerSize',15);
30     hold on;grid on;box on;
31     tri=delaunay(x,y);
32     trisurf(tri,x,y,z);
33
34     xlabel('X');ylabel('Y'),zlabel('Z');
35     title('Nonuniform triangular sample surface plot','fontweight',...
36           'bold','fontsize',16)
37

```

The result will be (see that the grid has the same number of vertices as the original vectors sample points!):

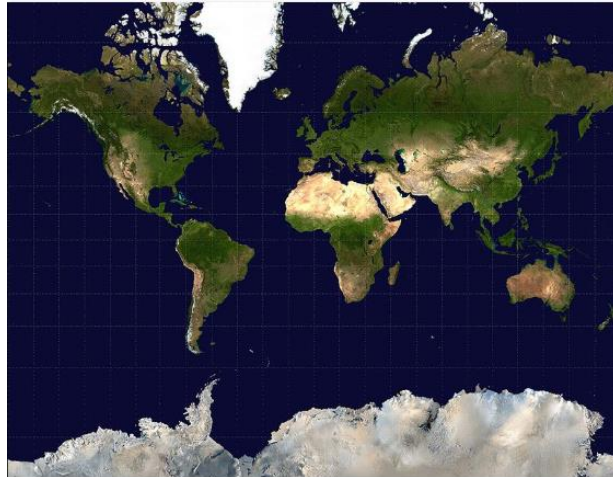
*Spline (biharmonic-spline)*

For an example see page 428.

Image mapping plots

Map an image on a 2D plane chart

A typical use of mapping images in 2D plots is for statistical representations on any area in the World. For the following example, we will use the following image:



<http://upload.wikimedia.org/wikipedia/commons/thumb/7/74/Mercator-projection.jpg/773px-Mercator-projection.jpg>

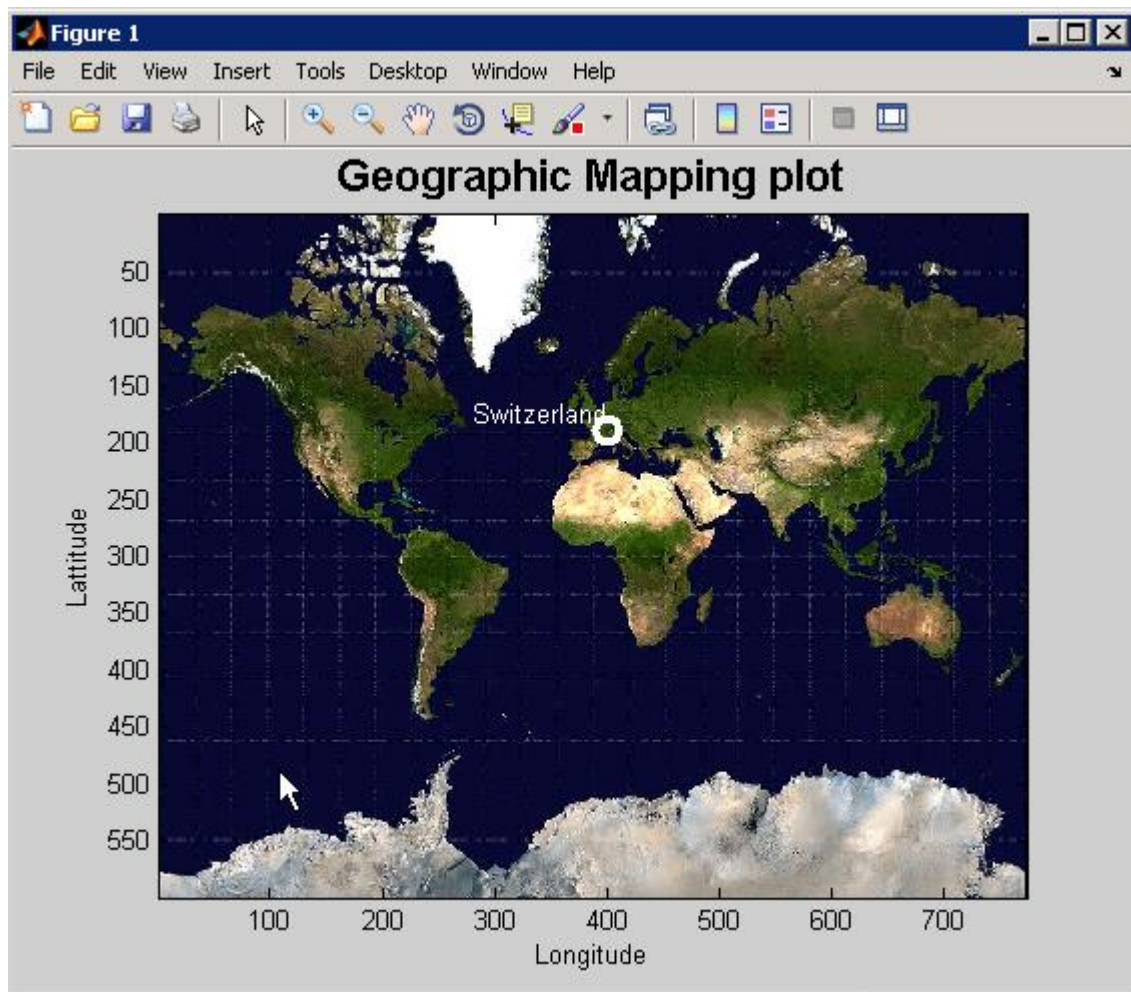
After we write the following script using the functions **imread()** and **imagesc()**:

```

EDITOR PUBLISH VIEW
1  img = imread('http://upload.wikimedia.org/wikipedia/commons/thumb/7/74/Merc
2  x=400;y=190;
3  labels='Switzerland';
4  figure;
5  imagesc(img);
6  xlabel('Longitude');
7  ylabel('Latitude');
8  hold on
9  plot(x,y, 'bo', 'MarkerSize',10, 'LineWidth',3,'color','white');
10 title('Geographic Mapping plot','fontweight',...
11      'bold','fontsize',16)
12 text(x,y, labels, 'Color','w', 'VerticalAlign',...
13      'bottom', 'HorizontalAlign','right')

```

The result is visible on next page:



Map an image on a 3D surface chart

A typical use of mapping images on 3D surfaces is for pedagogical purposes. Here is an well know example in General Relativity using the following image:



<http://d1jqu7g1y74ds1.cloudfront.net/wp-content/uploads/2010/08/The-Brightest-of-Stars.jpg>

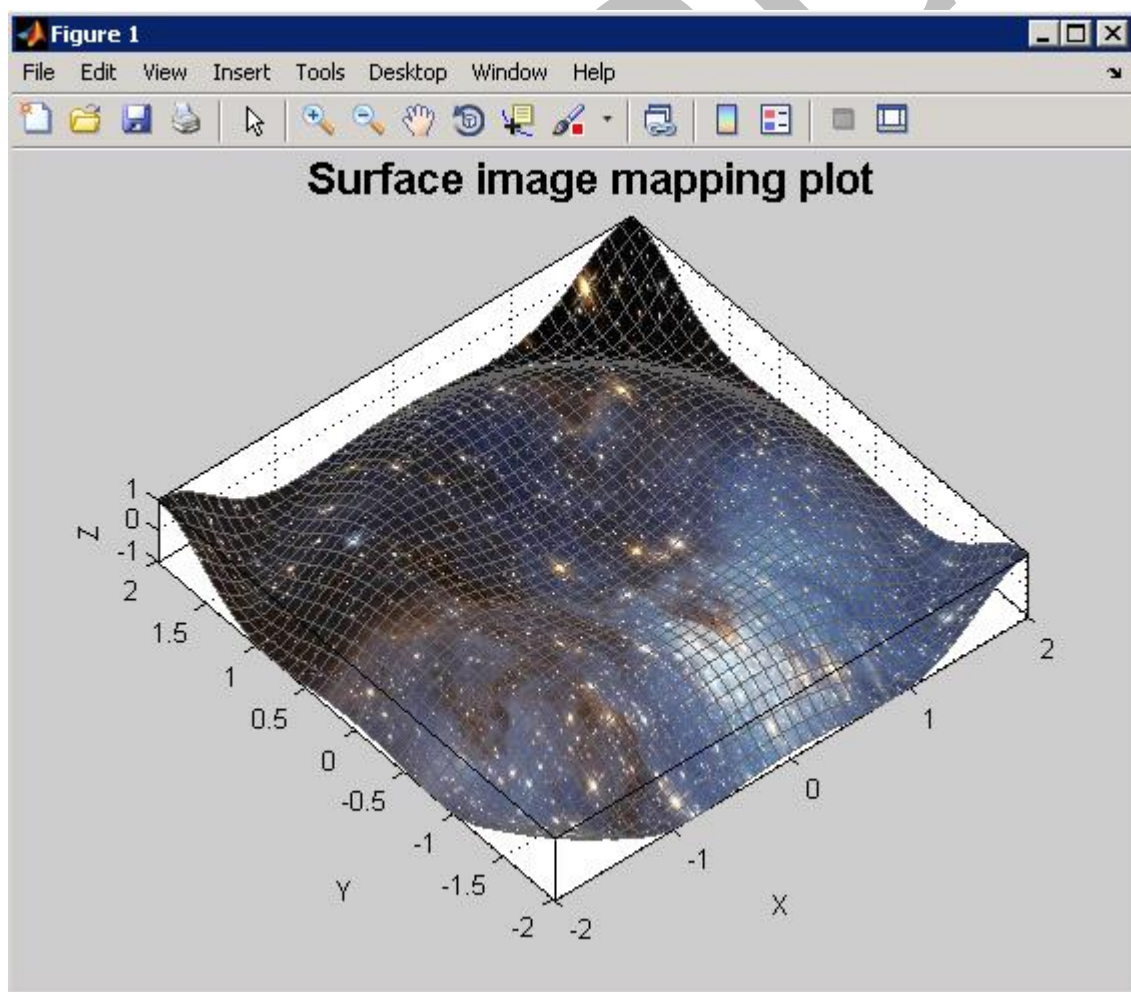
The tip afterwards is to play with the fourth **texturemap** parameter of the function **surf()**:

```

1  X=-2:0.1:2;
2  Y=X;
3  [x,y] = meshgrid(X, Y);
4  z = sin(x.^2 + y.^2);
5  a=imread('http://d1jq7g1y74ds1.cloudfront.net/wp-content/uploads/2010/08/T
6  s=surf(x,y,z,a,'FaceColor','texturemap');
7  shading interp
8  set(s, 'EdgeColor', [0.4 0.4 0.4])
9  view([-40 82]);
10 box on;
11 xlabel('X');ylabel('Y');zlabel('Z');
12 title('Surface image mapping plot','fontweight',...
13       'bold','fontsize',16)

```

The result will be:



Sphere plot with data mapping and animated rotation

To be able to plot a sphere and map an image on it is especially useful for meteorologists, geophysicists and spatial engineer. For this example we will use a demo data mat of MATLAB:

Command Window		Workspace				
>> load topo		Name	Value	Min	Bytes	Max
fx >>		topo	<180x360 double>	-7473	518400	5731
		topolegend	[1,90,0]	0	24	90
		topomap1	<64x3 double>	0	1536	1
		topomap2	<128x3 double>	0	3072	1

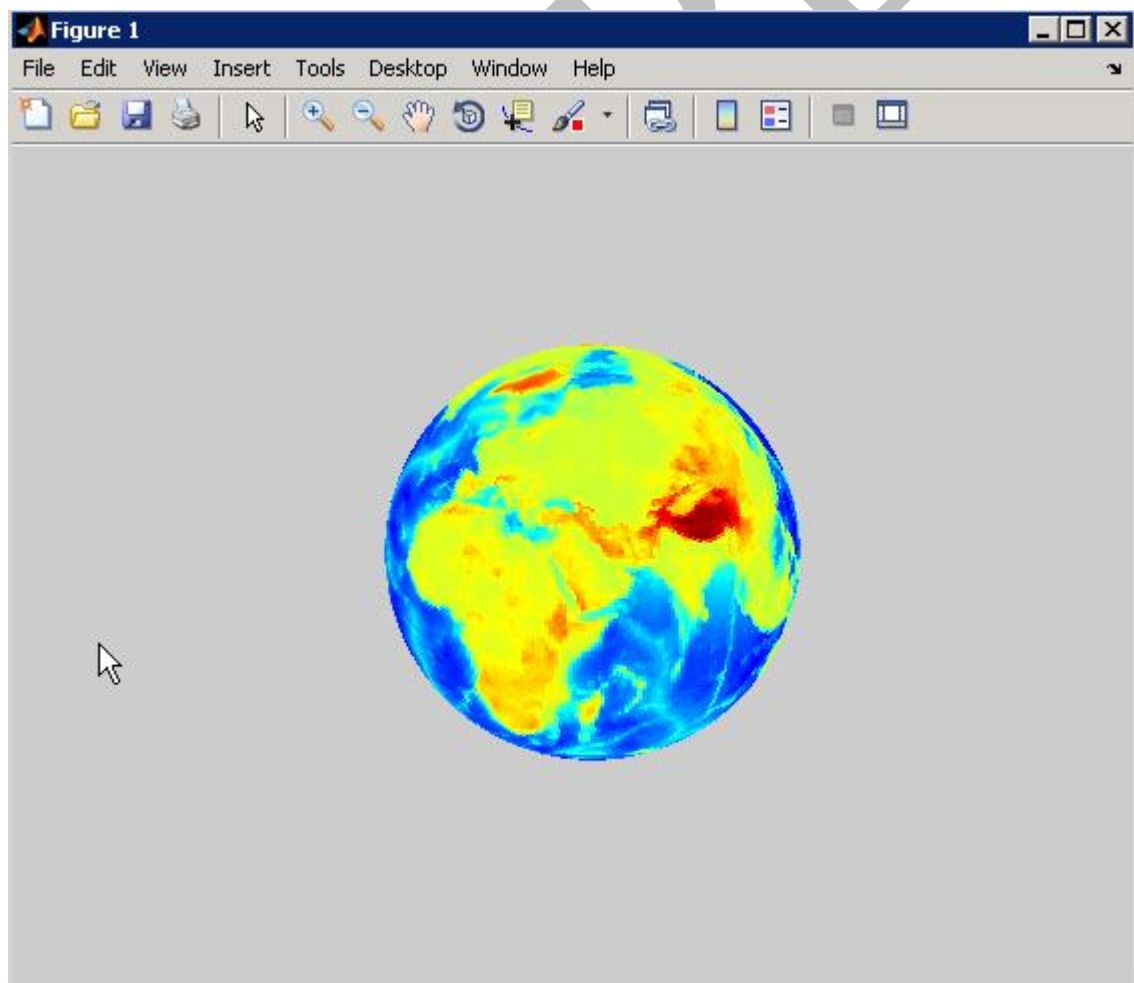
The content of the variable *topo* looks as following:

topo <180x360 double>							
	1	2	3	4	5	6	7
1	2783	2783	2783	2783	2783	2783	2783
2	2736	2736	2736	2736	2736	2736	2736
3	2743	2742	2734	2734	2734	2734	2734
4	2685	2684	2683	2683	2683	2683	2683
5	2682	2682	2682	2682	2683	2693	2710
6	2663	2666	2682	2682	2688	2737	2771
7	2664	2690	2691	2697	2745	2788	2841
8	2681	2690	2741	2750	2823	2919	2982
9	2489	2499	2560	2568	2651	2744	2793
10	2691	2745	2786	2874	2962	3023	3073
11	2702	2751	2817	2907	3011	3072	3132
12	2848	2930	3021	3110	3181	3226	3255
13	2822	2878	2946	3132	3237	3256	3314

Now let us plot using the functions **sphere()** and **surface()**:

```
EDITOR PUBLISH VIEW
1      n=300;
2      load topo;
3      [x,y,z]=sphere(n);
4      sh=surface(x,y,z,...
5              'facecolor','texturemap',...
6              'cdata',topo,'EdgeColor','none','LineStyle','none');
7
8      axis off;axis equal;
9      view(3); %see face
10     for i=1:20 %you can change the "20" to the value you want
11         rotate(sh,[0,0,1],1);
12         pause(1);
13     end
```

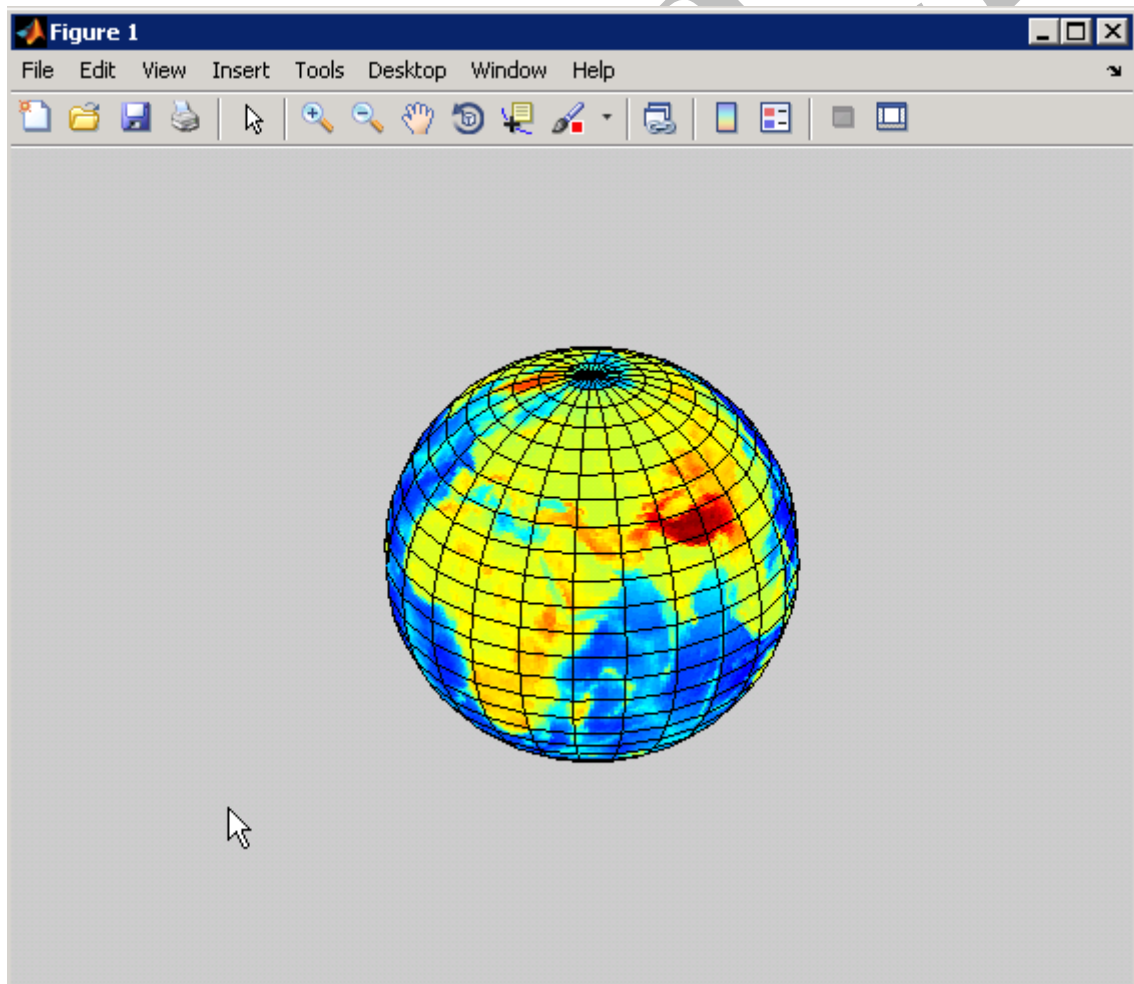
This is how the result will look like:



Now to have something more "geographic" let us draw a sphere with 24 meshes lines:

```
EDITOR PUBLISH VIEW
1 n=24;
2 load topo;
3 [x,y,z]=sphere(n);
4 sh=surface(x,y,z,'facecolor','texturemap','cdata',topo);
5 axis off;axis equal;
6 view(3); %see face
7 for i=1:20 %you can change the "20" to the value you want
8     rotate(sh,[0,0,1],1);
9     pause(1);
10 end
```

This will give:



Nice isn't it ;-)

Export animated plots as *.avi or *.gif

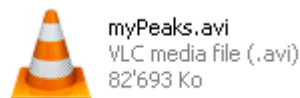
Export to *.avi

It is very useful for PowerPoint presentation to export as *.avi or for TV documentary purposes. For this, we use the function **getframe()** and **movie2avi()**:

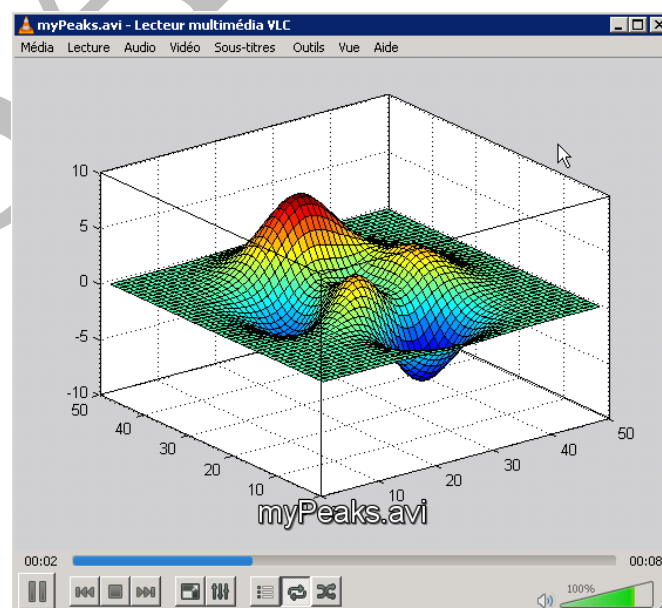
```

EDITOR PUBLISH VIEW
1 %Animated chart with *.avi as output
2 clear all;close all;clc
3 nFrames = 120;
4 mov(1:nFrames) = struct('cdata', [], 'colormap', []);
5 Z = peaks; surf(Z);
6 % Keep axes constant
7 axis([0 50 0 50 -10 10])
8 box on;
9 set(gca, 'nextplot', 'replacechildren');
10 for k = 1:nFrames
11     surf(sin(2*pi*k/nFrames)*Z,Z)
12     mov(k) = getframe(gcf);
13 end
14 movie2avi(mov, 'c:\myPeaks.avi', 'compression', 'None');
```

The result will create a file on your hard drive:



And when you will open it you will get a high quality video:



Export to *.gif

Export a chart animation as a *.gif is very useful for PowerPoint presentations and especially for Web pages.

The example below use the function **struct()**, **getframe()**, **frame2im()**, **rgb2im()** and **imwrite()** to export an animated **peaks** plot:

```

EDITOR PUBLISH VIEW
1 %Animated *.gif output (most used case)
2 clear all;close all;clc
3 nFrames = 20;
4 Z=peaks;surf(Z);
5 box on;
6 axis([0 50 0 50 -10 10])
7 set(gca,'NextPlot','replaceChildren');
8 % Preallocate the struct array for the struct returned by getframe
9 F(nFrames) = struct('cdata',[],'colormap',[]);
10 % Record the movie
11 for j = 1:nFrames
12     surf(.01+sin(2*pi*j/nFrames)*Z,Z)
13     F(j) = getframe;
14     %the next lines are to add the frame to the gif (difficult to guess...)
15     F = getframe(gcf); %first we get the frame from the chat
16     RGB = frame2im(F); %we convert the frame in true color
17     [IND,map] = rgb2ind(RGB, 255); %we convert in 255 indexed colors gif
18     outfile = 'c:\myPeaks.gif';
19     if j==1
20         imwrite(IND,map,outfile,'gif','DelayTime',0,'loopcount',inf);
21     else
22         imwrite(IND,map,outfile,'gif','DelayTime',0,'writemode','append');
23     end
24 end

```

This will create *.gif a file that you can insert on your webpage or PowerPoint presentation:

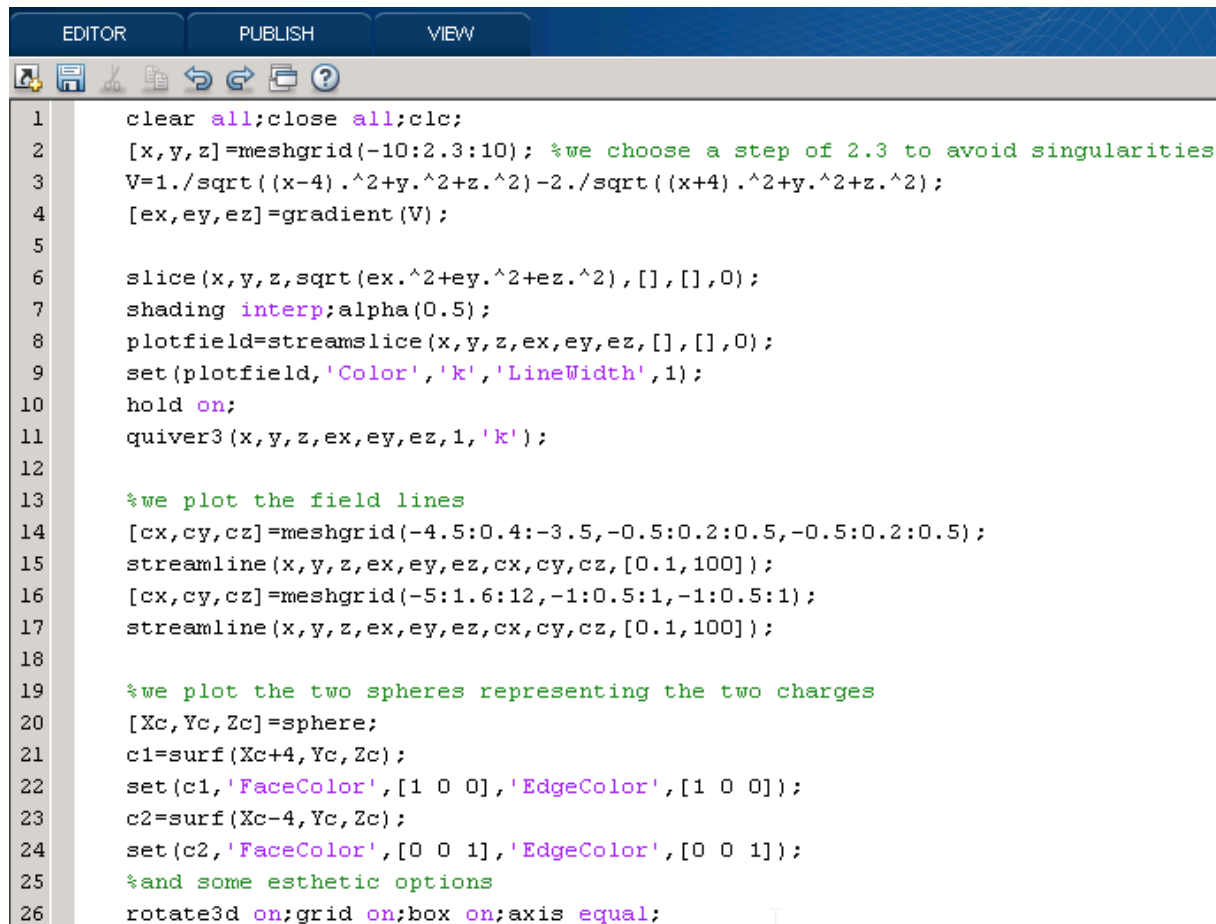


Engineering/Simulations

Vector filed equation analysis

This is a very academic example of an electric dipole a MATLAB. In the facts it is very easy to get such an analysis with Maple but with MATLAB this is very a pain in the a... because we have to manager singularities and play with the meshgrid steps to get a relevant result.

Thus consider the following example:

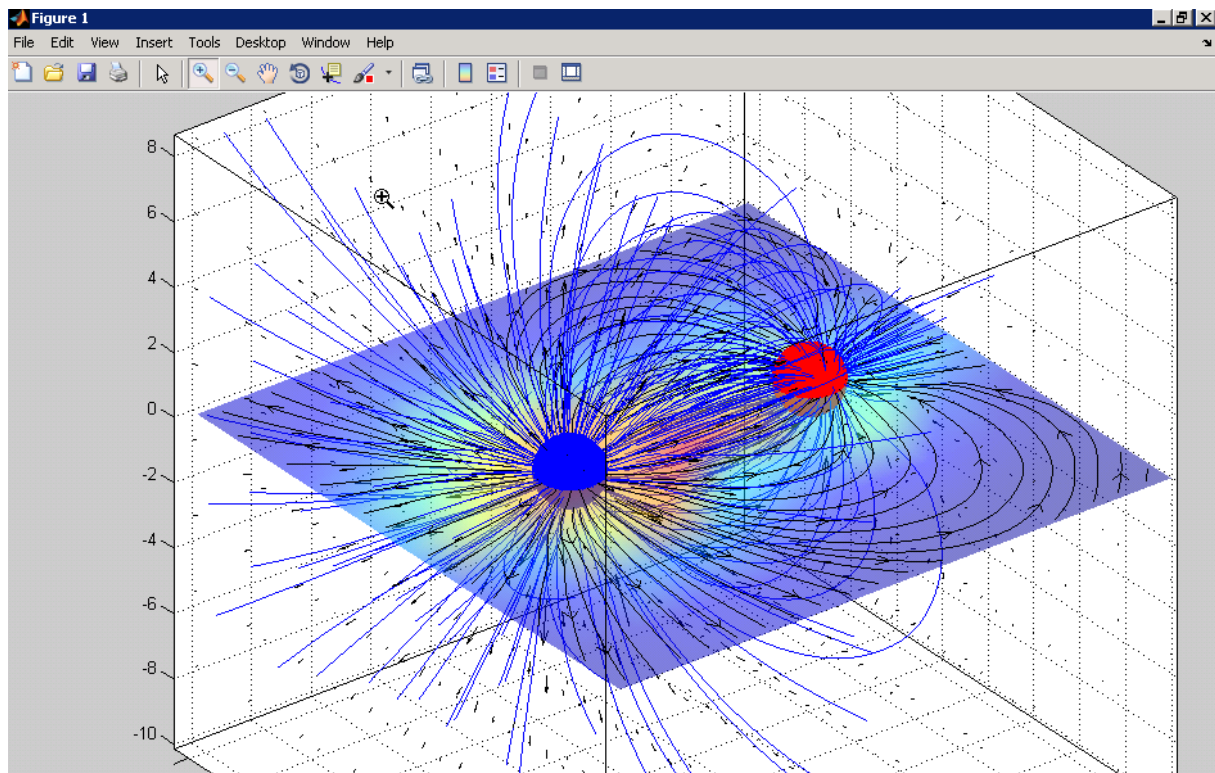


```

1 clear all;close all;clc;
2 [x,y,z]=meshgrid(-10:2.3:10); %we choose a step of 2.3 to avoid singularities
3 V=1./sqrt((x-4).^2+y.^2+z.^2)-2./sqrt((x+4).^2+y.^2+z.^2);
4 [ex,ey,ez]=gradient(V);
5
6 slice(x,y,z,sqrt(ex.^2+ey.^2+ez.^2),[],[],0);
7 shading interp;alpha(0.5);
8 plotfield=streamslice(x,y,z,ex,ey,ez,[],[],0);
9 set(plotfield,'Color','k','LineWidth',1);
10 hold on;
11 quiver3(x,y,z,ex,ey,ez,1,'k');
12
13 %we plot the field lines
14 [cx,cy,cz]=meshgrid(-4.5:0.4:-3.5,-0.5:0.2:0.5,-0.5:0.2:0.5);
15 streamline(x,y,z,ex,ey,ez,cx,cy,cz,[0.1,100]);
16 [cx,cy,cz]=meshgrid(-5:1.6:12,-1:0.5:1,-1:0.5:1);
17 streamline(x,y,z,ex,ey,ez,cx,cy,cz,[0.1,100]);
18
19 %we plot the two spheres representing the two charges
20 [Xc,Yc,Zc]=sphere;
21 c1=surf(Xc+4,Yc,Zc);
22 set(c1,'FaceColor',[1 0 0],'EdgeColor',[1 0 0]);
23 c2=surf(Xc-4,Yc,Zc);
24 set(c2,'FaceColor',[0 0 1],'EdgeColor',[0 0 1]);
25 %and some esthetic options
26 rotate3d on;grid on;box on;axis equal;

```

The result will be after a small zoom (see on next page):



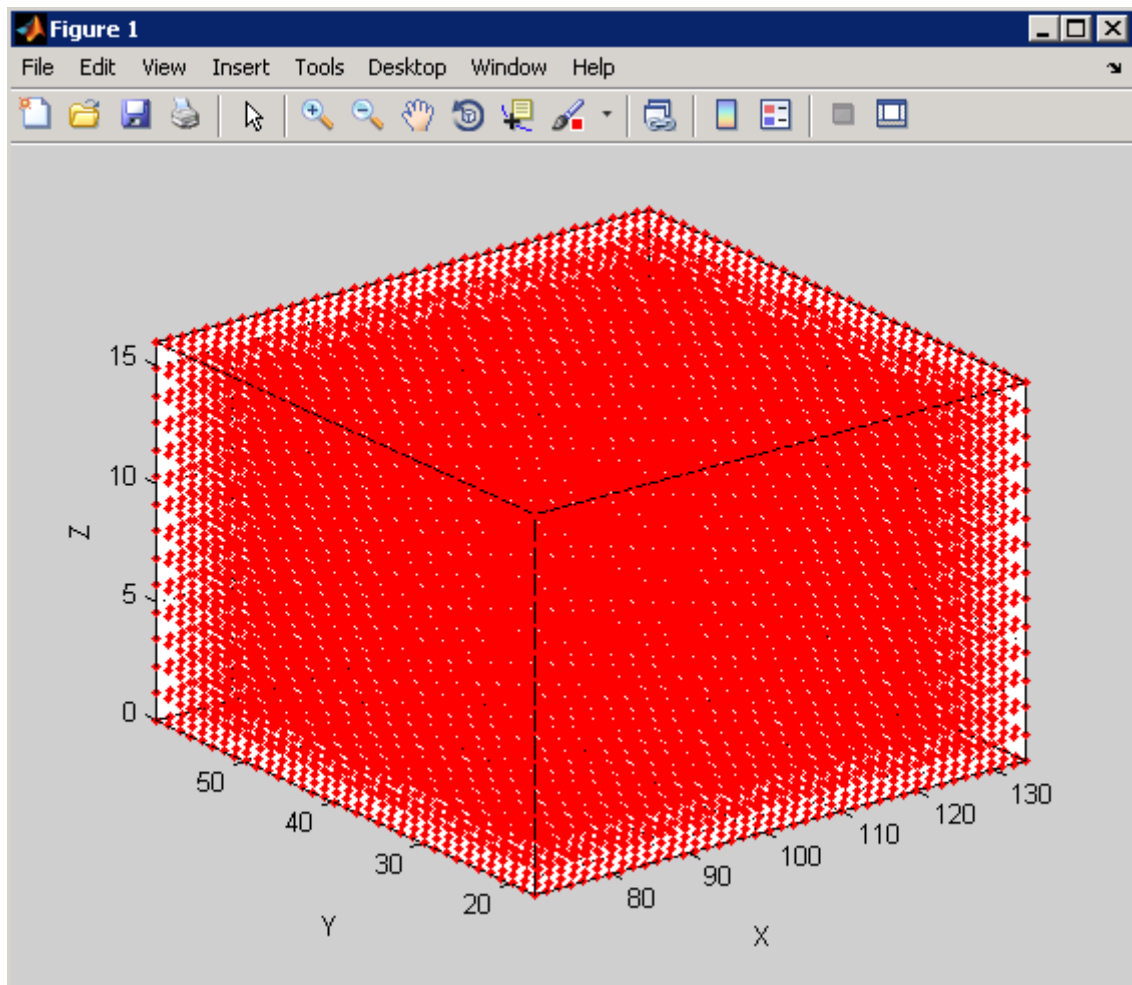
Vector field snapshots analysis

You have to imagine that we have measured (or calculated) in a spatial constant (x, y, z) grid (of size 35×41×15) the velocity of the wind.

Command Window		Workspace			
>> load wind fx >>		Name ▲	Value	Min	Bytes
		u	<35x41x15 double>	-13.7...	172200
		v	<35x41x15 double>	-47.3...	172200
		w	<35x41x15 double>	-1.3914	172200
		x	<35x41x15 double>	70.1879	172200
		y	<35x41x15 double>	17.4999	172200
		z	<35x41x15 double>	-0.0020	16

We can see here below the grid measure box:

EDITOR	PUBLISH	VIEW
<pre> 1 clear all;clc;close all 2 load wind 3 x=x(:,:);y=y(:,:);z=z(:,:); 4 plot3(x,y,z,'.r') 5 axis([min(x(:)) max(x(:)) min(y(:)) max(y(:)) min(z(:)) max(z(:))]) 6 grid on;box on; 7 </pre>		



Now we take a look closer to velocities following the notation seen in fluid mechanics theory:

$$(v_x, v_y, v_z) = (u, v, w)$$

Because in our situation we have a grid of $35 \times 41 \times 15$ we have also so the same quantity of (v_x, v_y, v_z) measures:

u	<35x41x15 double>
v	<35x41x15 double>
w	<35x41x15 double>
x	<35x41x15 double>
y	<35x41x15 double>
z	<35x41x15 double>

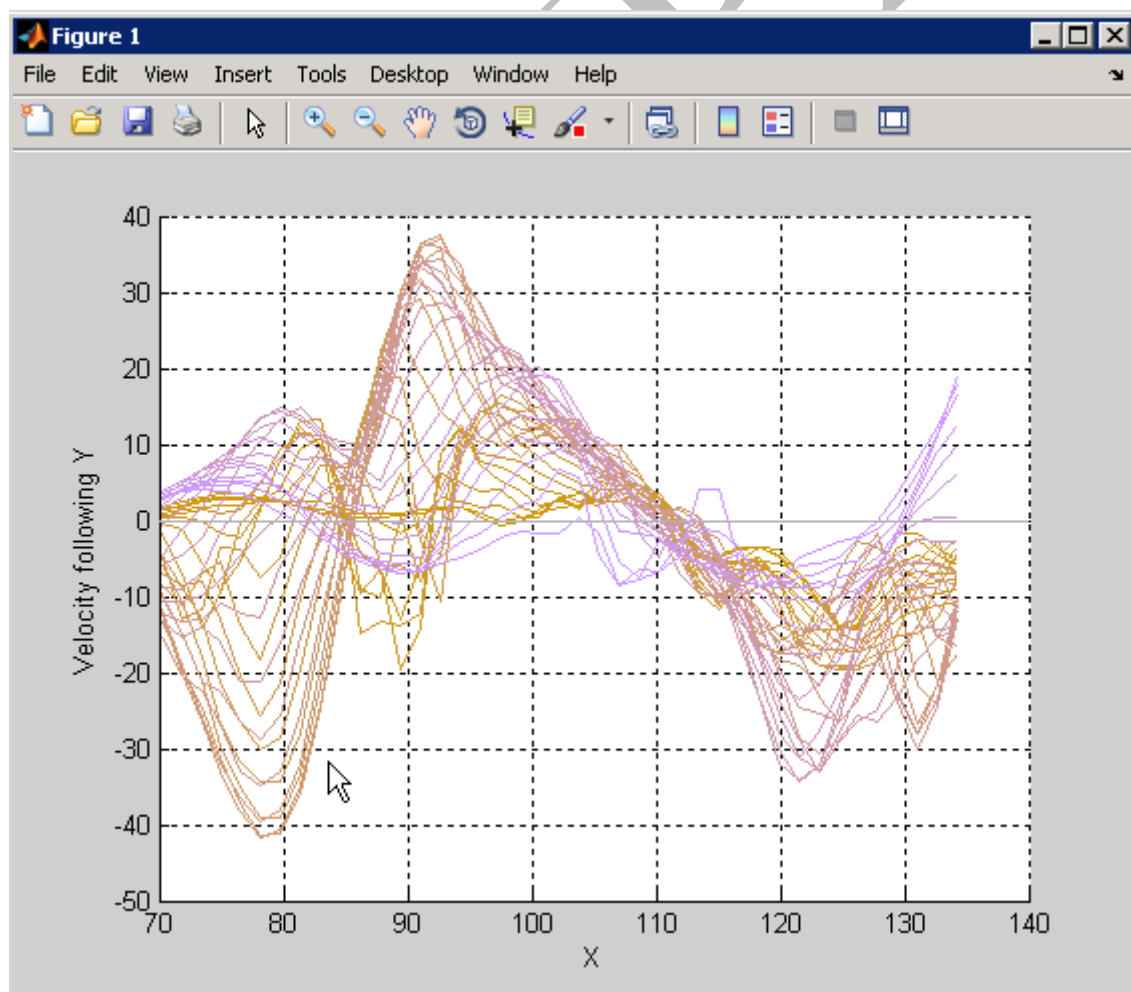
We can now take a special look at the velocity following Y:

```

EDITOR PUBLISH VIEW
1 clear all;clc;close all
2 %by curiosity we take a look of the velocity following Y, for a given Z
3 %measure line and multiples X
4 Z=7; %This fixed the Z measure line height (empirical choice)
5 load wind
6 for coord=1:1:35
7     %remember that "v" is the velocity following "Y"
8     v_plot=v(coord,:,Z);
9     x_plot=x(coord,:,Z);
10    hold on;
11    grid on;
12    plot(x_plot,v_plot,'Color',[0.8 0.6 coord/35])
13 end
14 %the zero line
15 hy = graph2d.constantline(0, 'Color',[.7 .7 .7]);
16 changedependvar(hy,'y');
17 xlabel('X');ylabel('Velocity following Y');

```

This will give:



We can see above a speed inversion when X is between $[70,80]$.

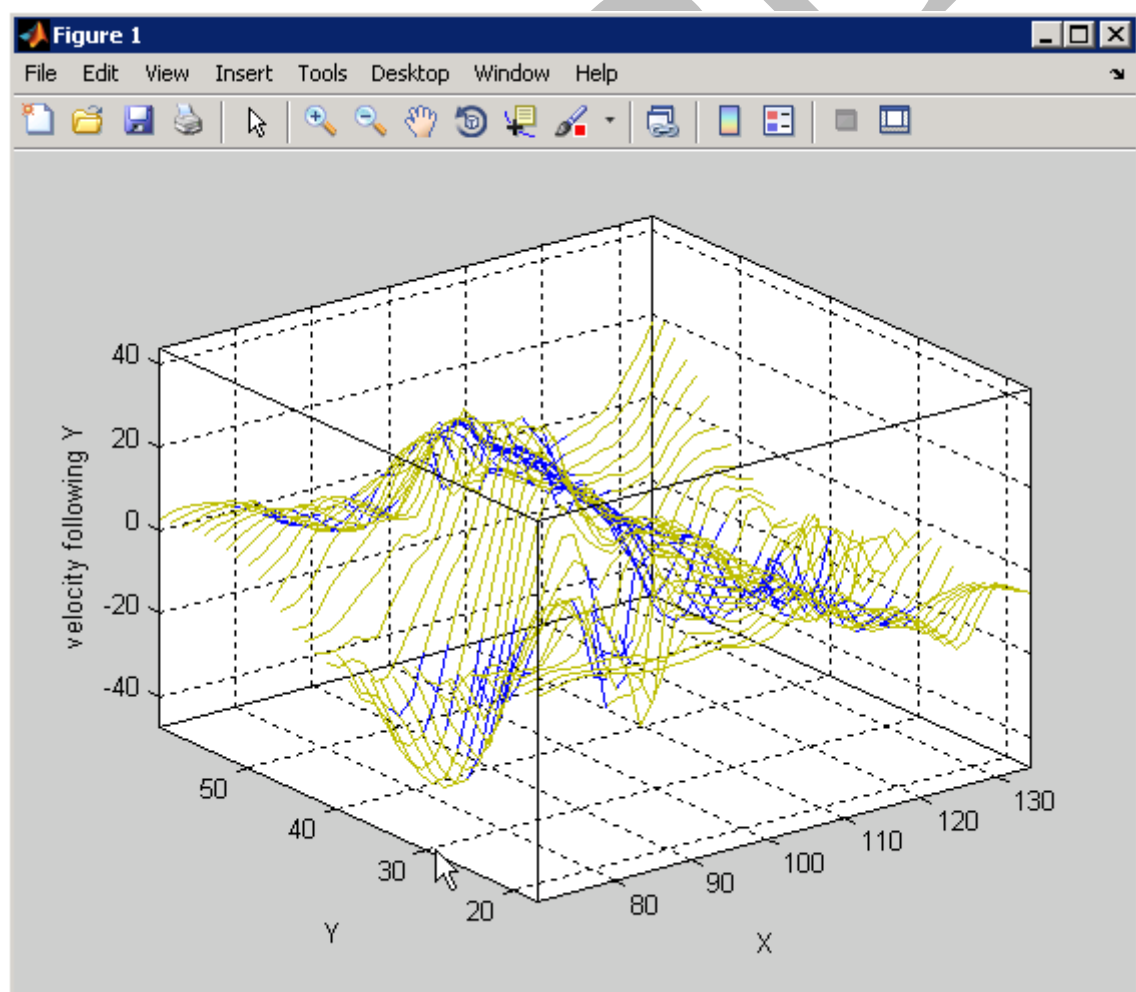
Or in 3D:

```

1  clear all;clc;close all;
2  load wind;
3  Z=7;
4  for coord=1:1:35
5      plot3(x(coord,:),1,repmat(y(coord,1,Z),41),v(coord,:,Z));
6      hold on;
7  end
8  axis([min(x(:)) max(x(:)) min(y(:)) max(y(:)) min(v(:)) max(v(:))])
9  grid on;box on;
10 xlabel('X');ylabel('Y');zlabel('velocity following Y');

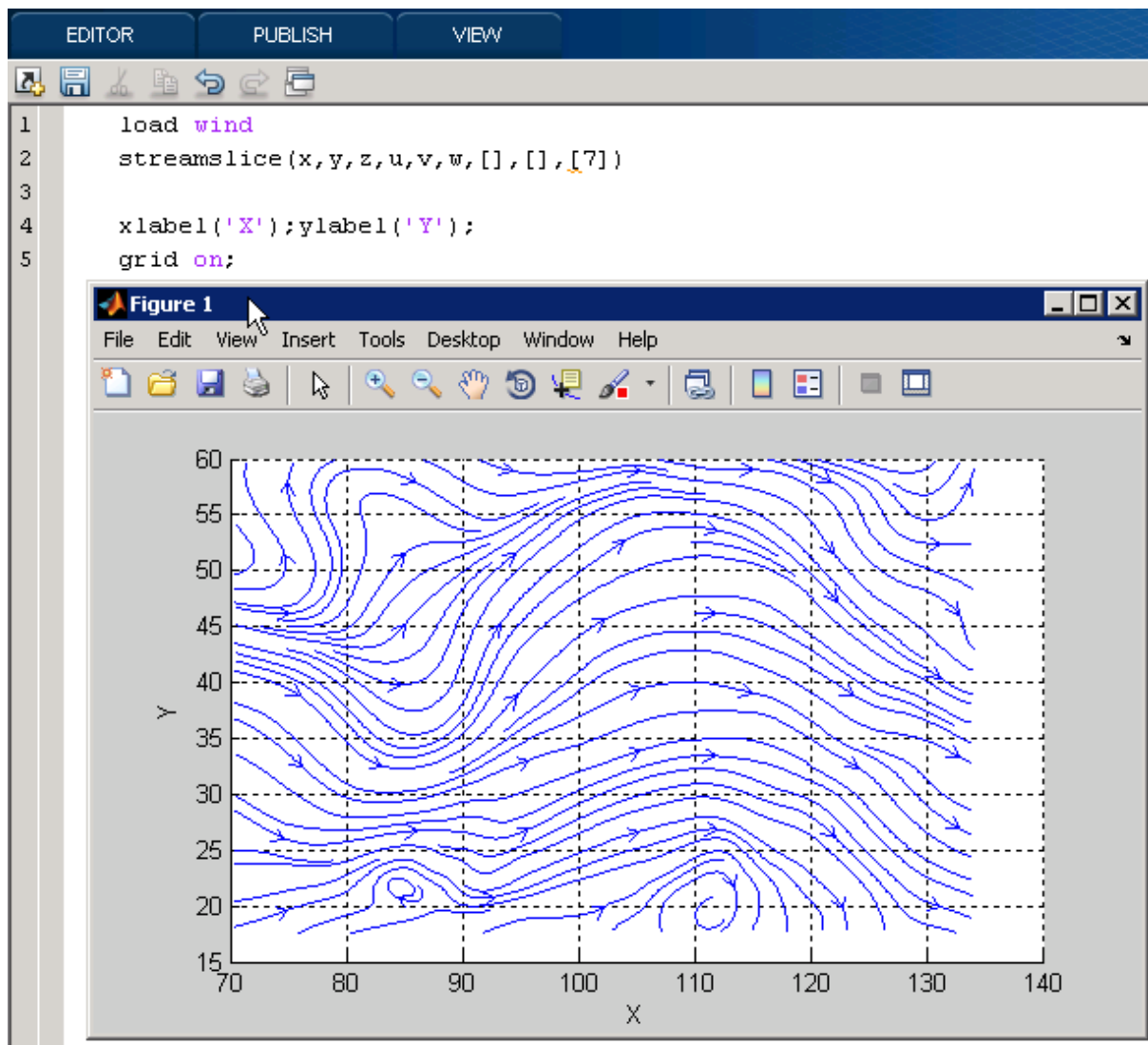
```

This will give:



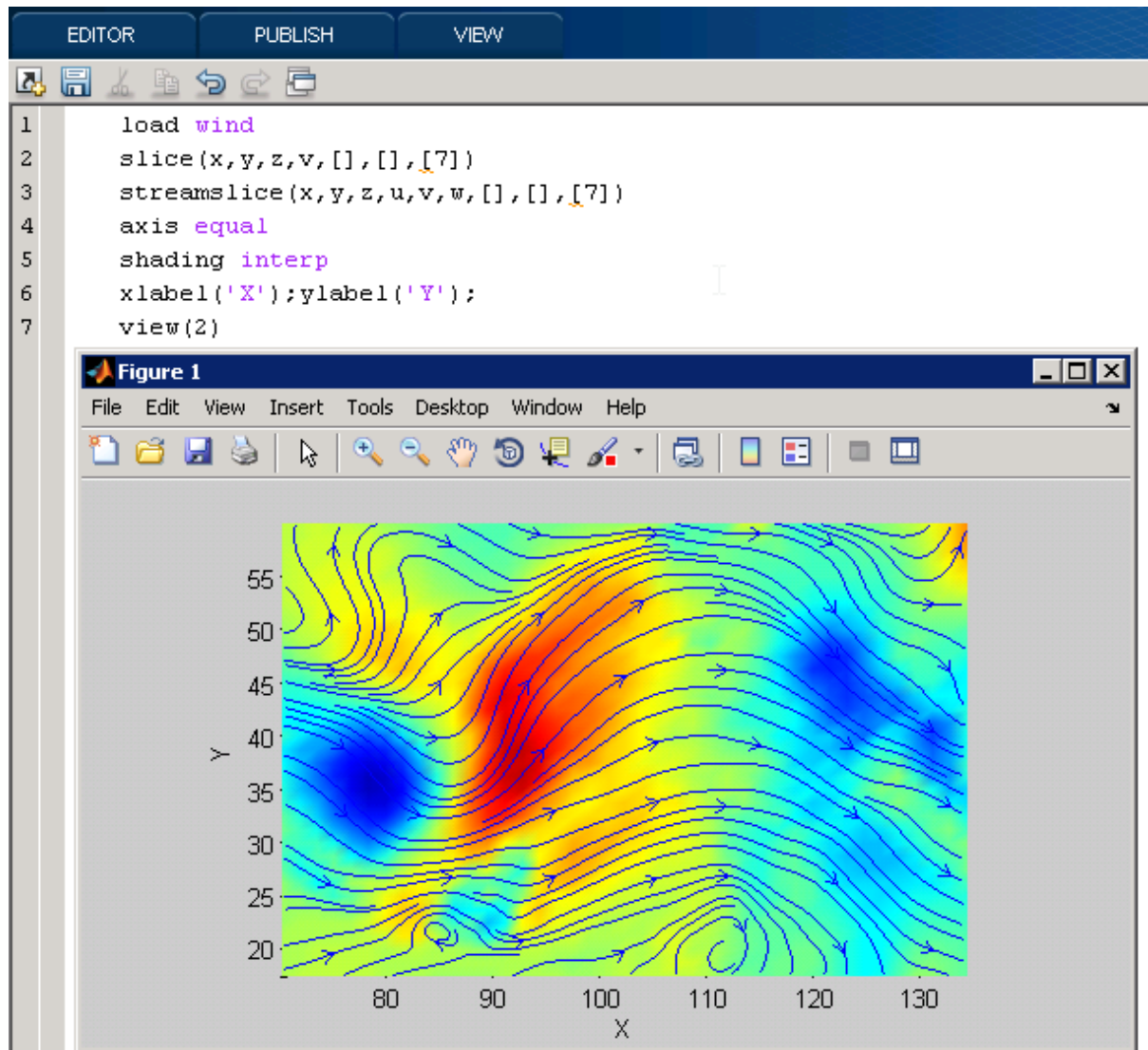
ToDo: make the same faster with waterfall

We can see a profile for the chosen Z using the function **streamslice()**:



Thus we understand better the inversion of the velocity following X in the range [70,90].

Or in another way:



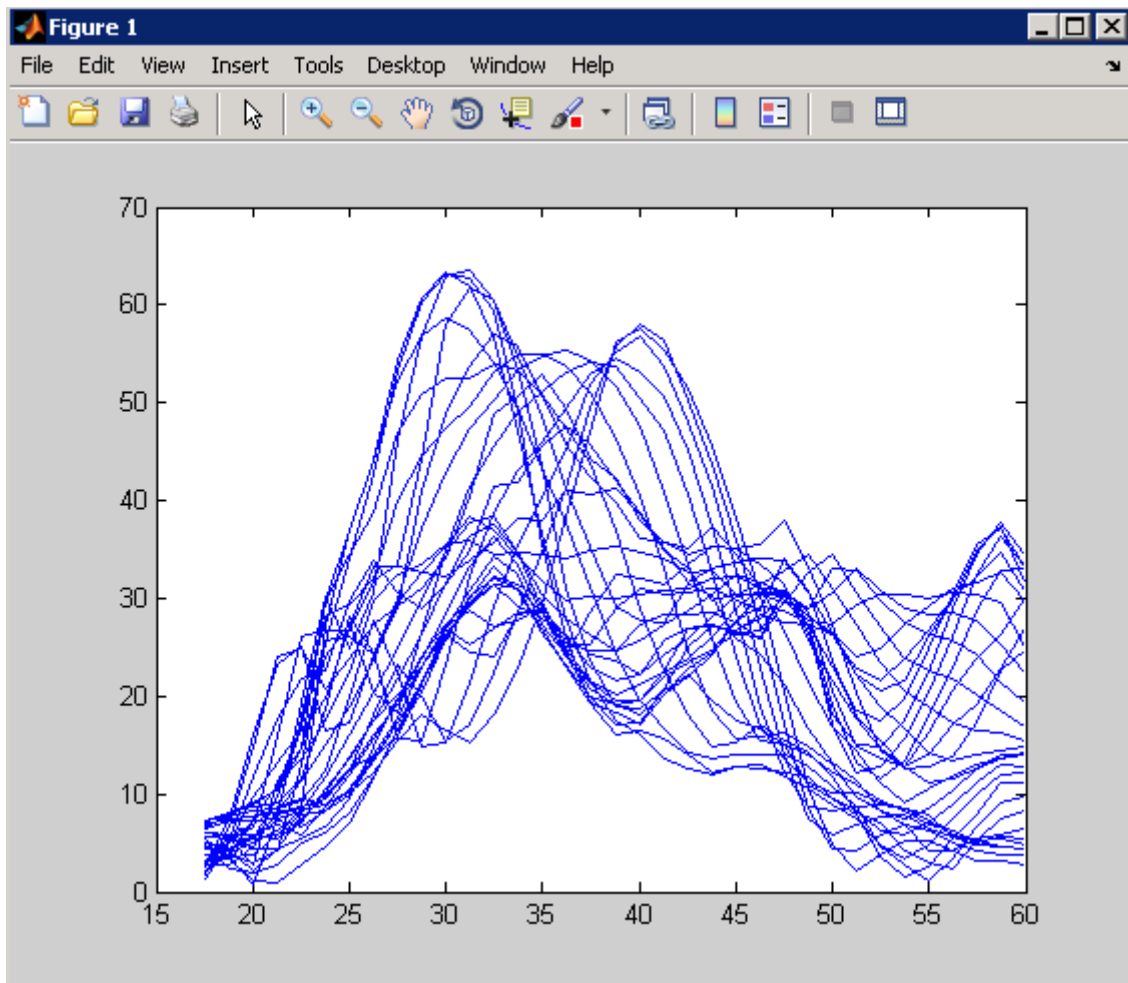
That helps to understand the amplitude of velocity!

We can also see the same thing by taking the norm of the velocity in the (x, y) plane for the same Z for each Y:

```

EDITOR PUBLISH VIEW
1 clear all;clc;close all
2 Z=7;
3 for coord=1:1:31
4     load wind
5     u=u(:,coord,Z);v=v(:,coord,Z);y=y(:,coord,Z);
6     spd = sqrt(u.*u + v.*v);
7     plot(y,spd)
8     hold on
9 end

```

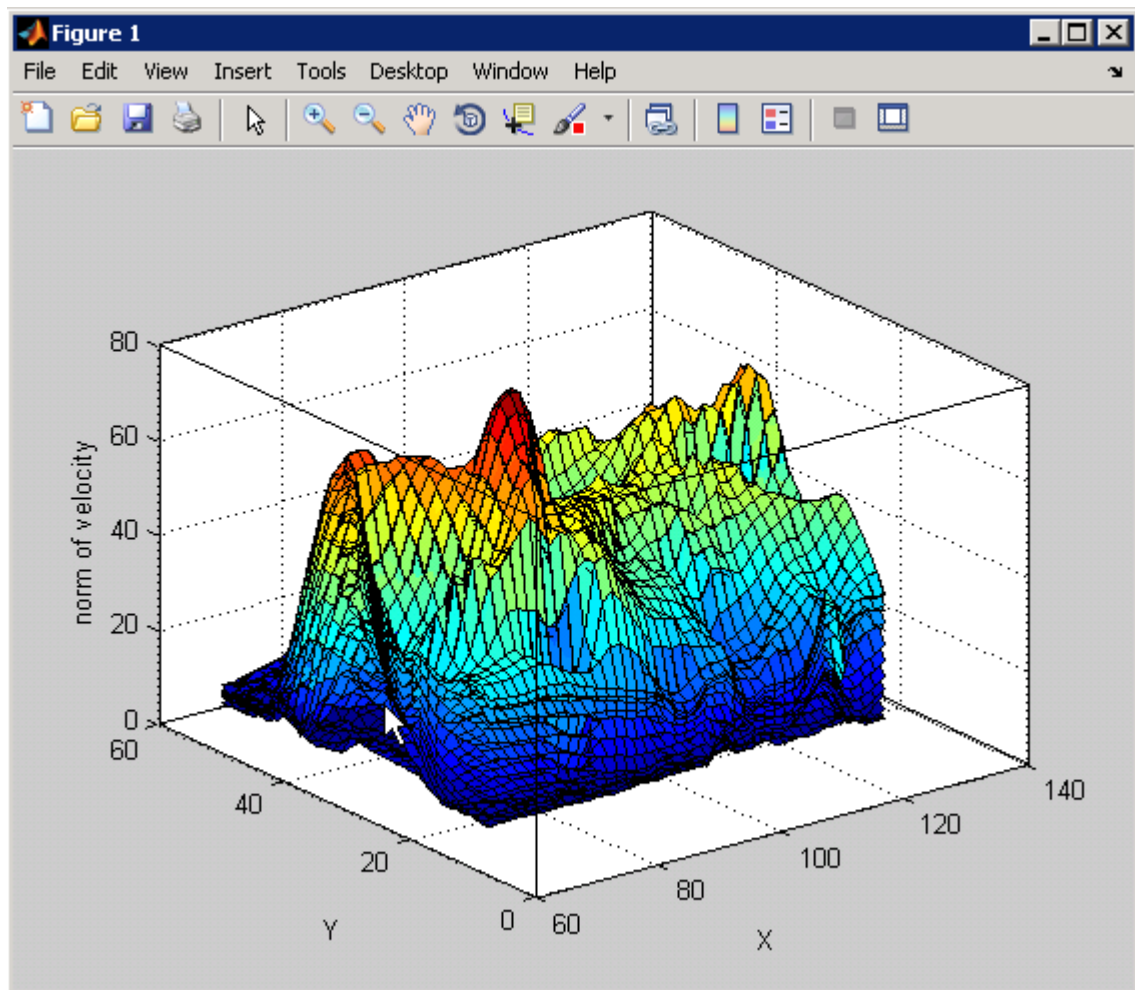


We can also take a close look to the norm of the velocity for different Z and all (x, y):

```

EDITOR    PUBLISH    VIEW
+-----+
1  clear all;clc;close all
2  for k=1:15
3      for coord=1:31
4          load wind
5          u=u(:,:,k);v=v(:,:,k);w=w(:,:,k);
6          x=x(coord,:,k);y=y(:,coord,k);
7          spd = sqrt(u.*u + v.*v+w.*w);
8          surf(x,y,spd)
9          hold on
10         end
11     end
12     grid on;box on;
13     xlabel('X');ylabel('Y');zlabel('norm of velocity');

```



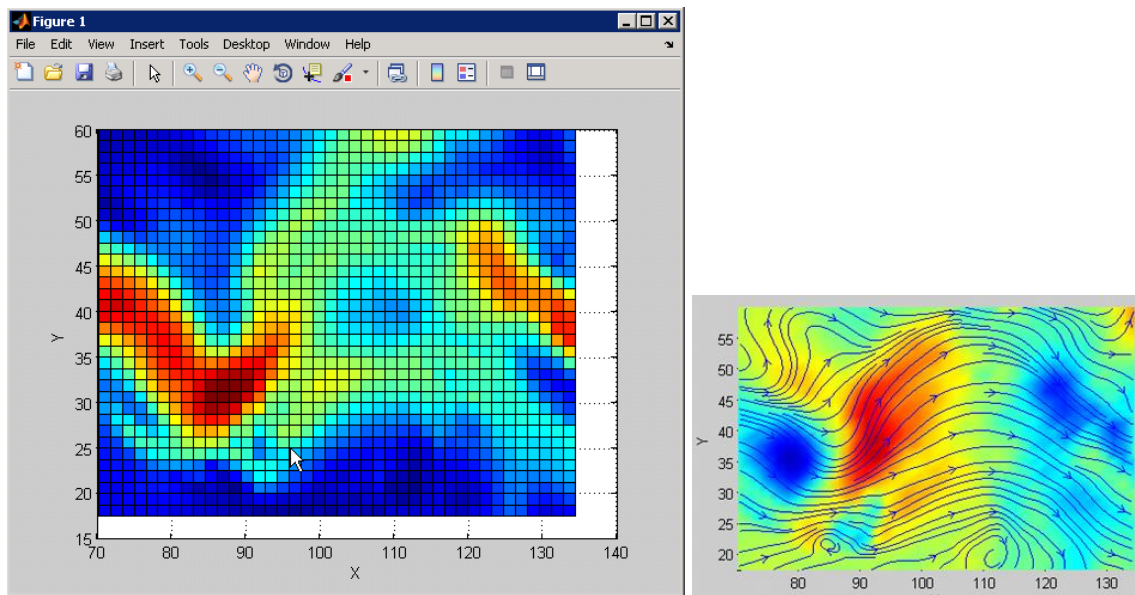
As we can see: deeper we are, slower is the velocity. In the borders velocity is also low. This corresponds to the streamslice!

But if we focus only on the 7th level of Z we see that:

```

EDITOR    PUBLISH    VIEW
[Icons]
1      clear all;clc;close all
2      for k=7
3          for coord=1:31
4              load wind
5              u=u(:,:,k);v=v(:,:,k);w=w(:,:,k);
6              x=x(coord,:,k);y=y(:,coord,k);
7              spd = sqrt(u.*u + v.*v+w.*w);
8              surf(x,y,spd)
9              hold on
10         end
11     end
12     grid on;box on;
13     xlabel('X');ylabel('Y');zlabel('norm of velocity');
14     view(2)

```

Then we see that the streamslice take (for sure...) in comparison the global amplitude for all Z!

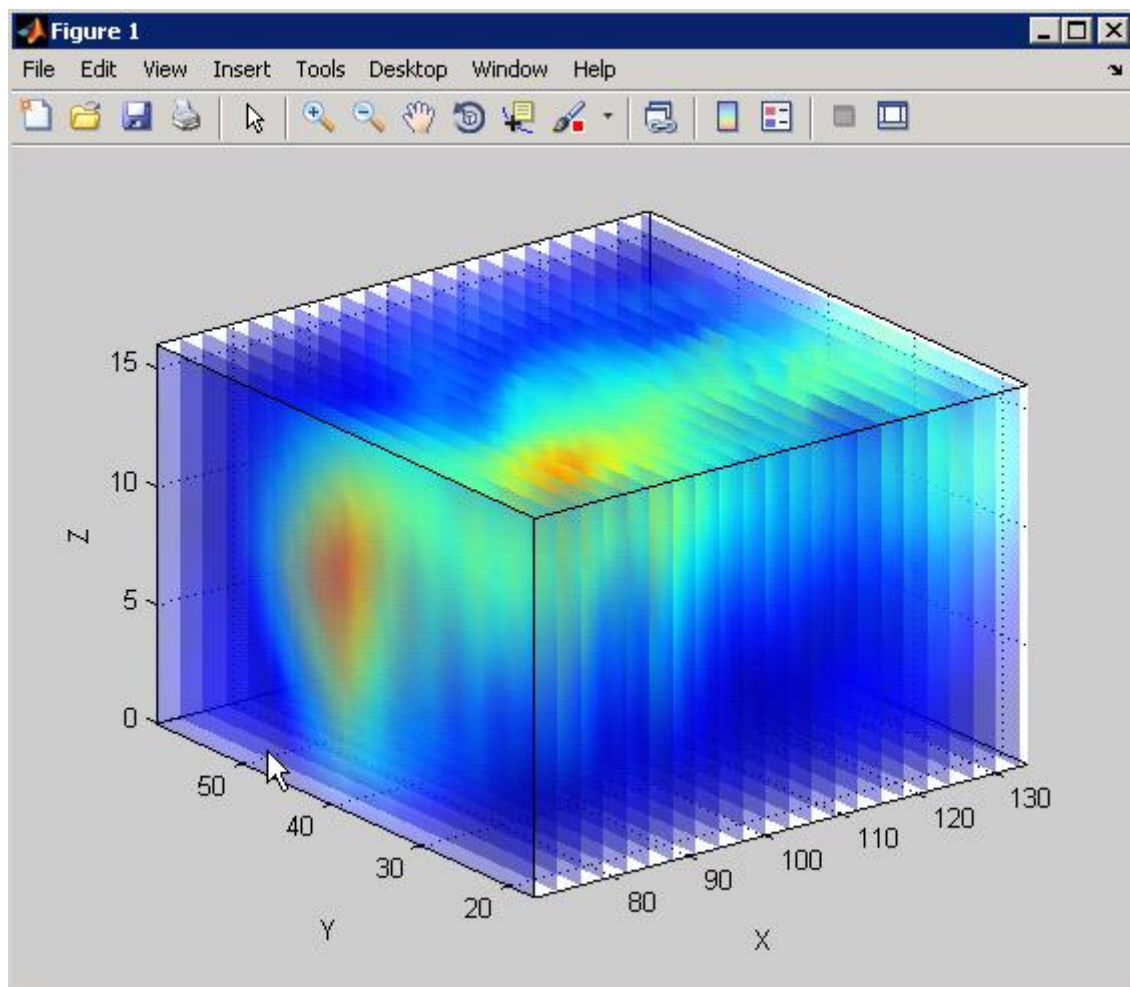
In a more advanced way we can see the norm of the velocity in 3D with interpolation using the function `slice()`:

```

EDITOR    PUBLISH    VIEW
1      clear all;clc;close all
2      load wind;
3      xmin=min(x(:));xmax=max(x(:));
4      wind_speed=sqrt(u.^2+v.^2+w.^2);
5      hsurfaces=slice(x,y,z,wind_speed,[xmin:3:xmax],[[]],[[]]);
6      set(hsurfaces,'FaceColor','interp','EdgeColor','none');
7      alpha(0.4)
8      grid on;box on;
9      xlabel('X');ylabel('Y');zlabel('Z');
10     axis([min(x(:)) max(x(:)) min(y(:)) max(y(:)) min(z(:)) max(z(:))]);

```

We can see an expected result: the velocity is low in the borders of the box a high velocities are concentrated in the center.

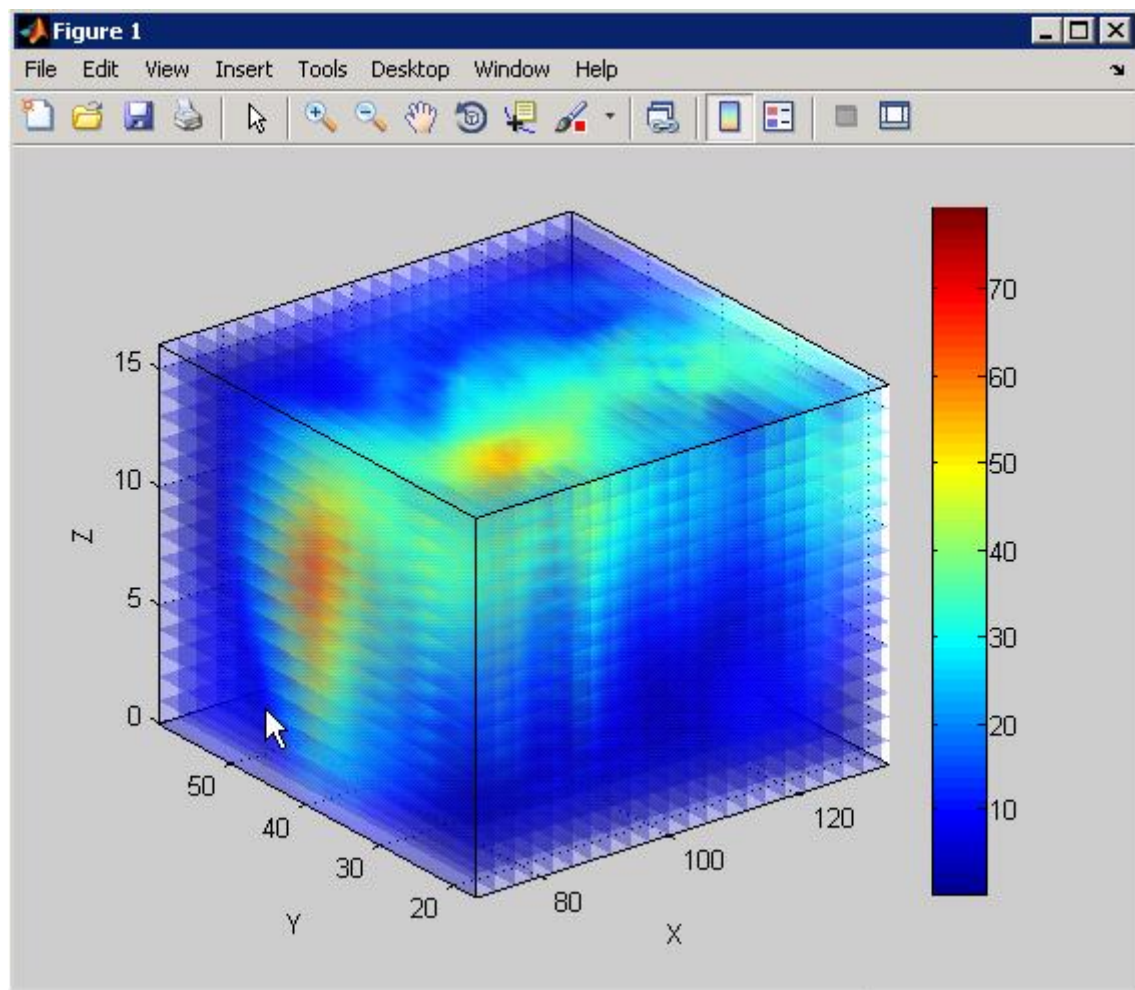


Or if you prefer something more "dense":

```

EDITOR    PUBLISH    VIEW
1      clear all;clc;close all
2      load wind;
3      zmin=min(z(:));zmax=max(z(:));
4      xmin=min(x(:));xmax=max(x(:));
5      wind_speed=sqrt(u.^2+v.^2+w.^2);
6      hsurfaces=slice(x,y,z,wind_speed,[xmin:3:xmax],[zmin:1:zmax]);
7      set(hsurfaces,'FaceColor','interp','EdgeColor','none');
8      alpha(0.3);
9      colorbar;
10     grid on;box on;
11     xlabel('X');ylabel('Y');zlabel('Z');
12     axis([min(x(:)) max(x(:)) min(y(:)) max(y(:)) min(z(:)) max(z(:))]);

```

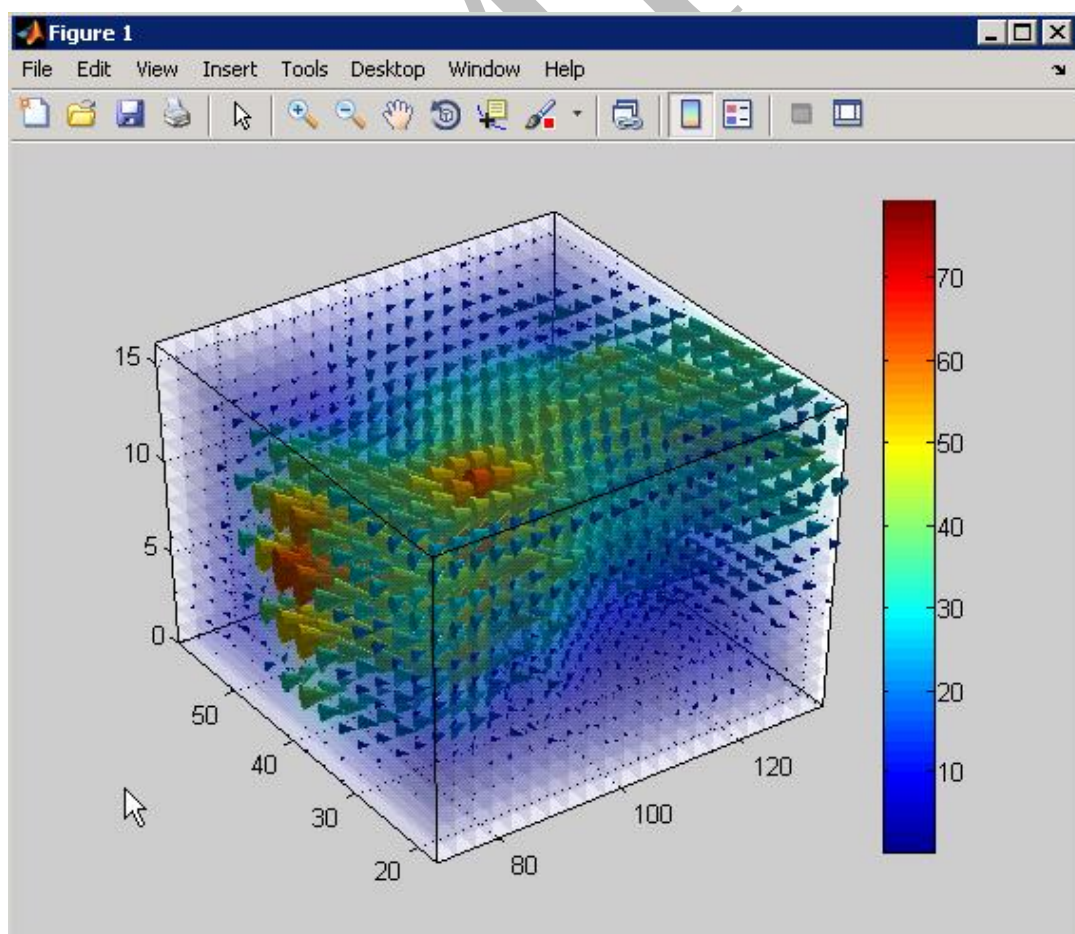


Or in a different way using the function **coneplot()**:

```

1  clear all;clc;close all
2  load wind;
3  xmin = min(x(:));xmax = max(x(:));ymin = min(y(:));ymax = max(y(:));
4  zmin = min(z(:));zmax = max(z(:));
5  wind_speed = sqrt(u.^2 + v.^2 + w.^2);
6  hsurfaces=slice(x,y,z,wind_speed,[xmin:3:xmax],[ymin:3:ymax],[zmin:1:zmax]);
7  set(hsurfaces,'FaceColor','interp','EdgeColor','none');
8  alpha(0.1);
9  colors = wind_speed;
10 scale = 4;
11 [cx, cy, cz] = meshgrid(xmin:3:xmax,ymin:3:ymax,zmin:3:zmax);
12 c = coneplot(x,y,z,u,v,w,cx,cy,cz,scale,colors);
13 set(c,'EdgeColor','none');
14 view([-35 40]);
15 camproj perspective;
16 grid on;box on;
17 light;lighting flat;
18 colorbar;
19 xlabel('X');ylabel('Y');zlabel('Z');
20 axis([min(x(:)) max(x(:)) min(y(:)) max(y(:)) min(z(:)) max(z(:))]);

```

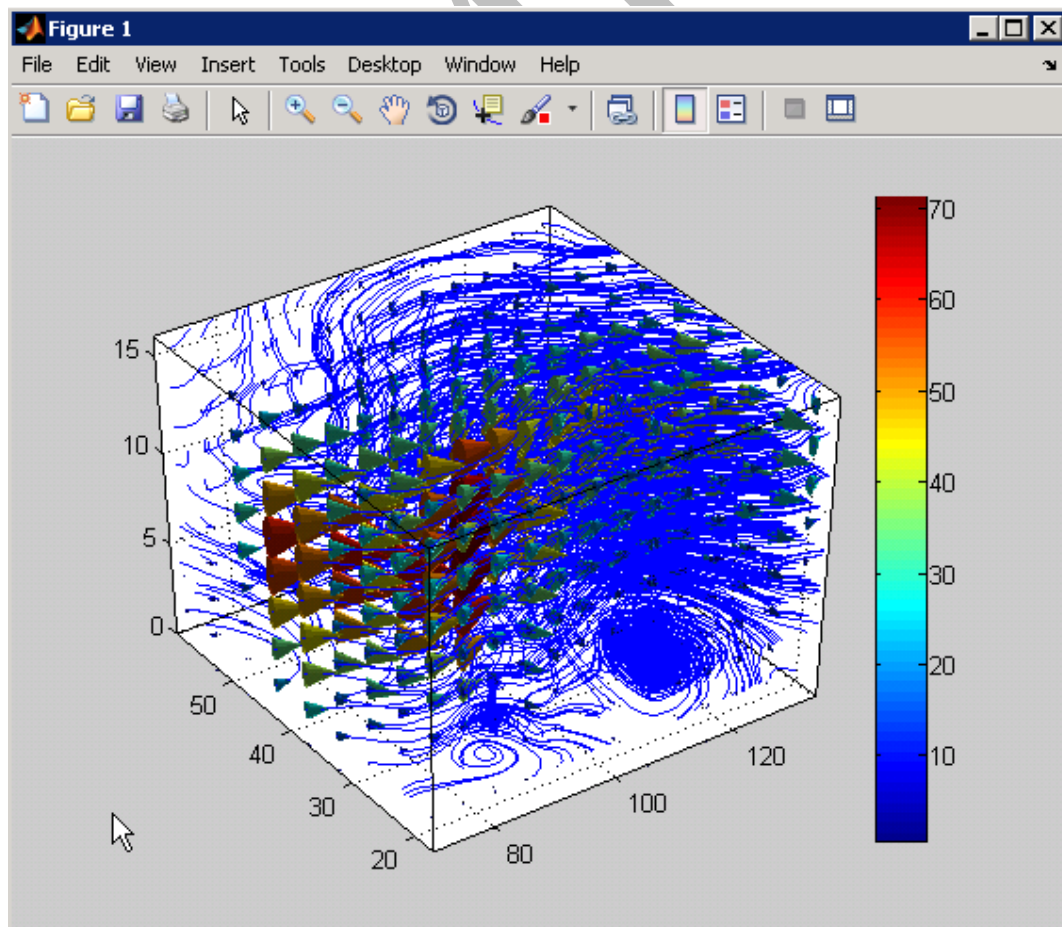


Or something a little bit more interesting using the function **streamline()**:

```

1  clear all;clc;close all
2  load wind;
3  xmin = min(x(:));xmax = max(x(:));ymin = min(y(:));ymax = max(y(:));
4  zmin = min(z(:));zmax = max(z(:));
5  wind_speed = sqrt(u.^2 + v.^2 + w.^2);
6  colorbar('vertical');
7  colors = wind_speed;
8  scale=4;
9  [cx, cy, cz] = meshgrid(xmin:5:xmax,ymin:5:ymax,zmin:2:zmax);
10 c = coneplot(x,y,z,u,v,w,cx,cy,cz,scale,colors);
11 set(c,'EdgeColor','none');
12 h = streamline(x,y,z,u,v,w,cx,cy,cz);
13 view([-35 40]);
14 camproj perspective;
15 grid on;box on;
16 light;lighting flat;
17 colorbar;
18 xlabel('X');ylabel('Y');zlabel('Z');
19 axis([min(x(:)) max(x(:)) min(y(:)) max(y(:)) min(z(:)) max(z(:))]);

```

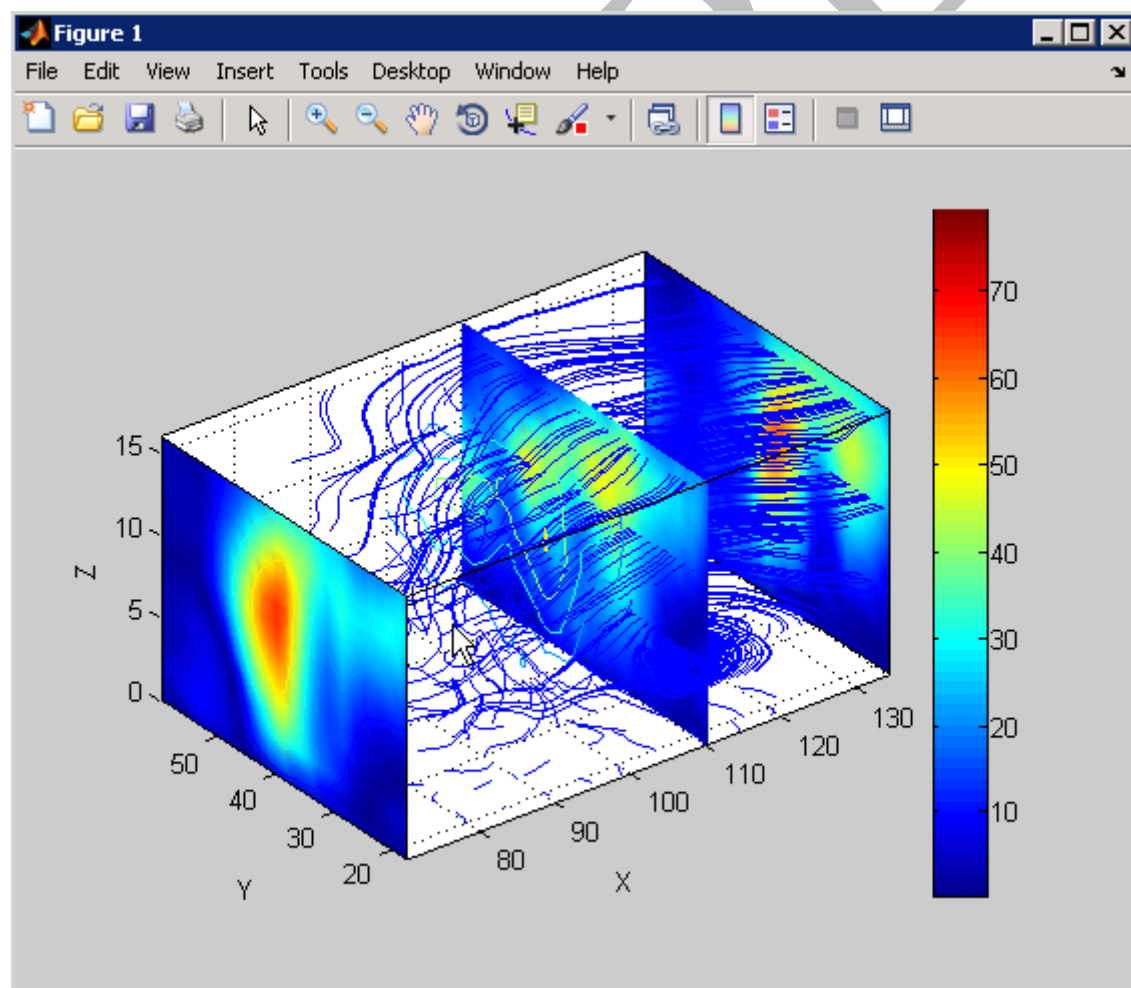


Or with another style using the function **contourslice()**:

```

1 clear all;clc;close all
2 load wind;
3 xmin=min(x(:));xmax=max(x(:));ymax=max(y(:));zmin=min(z(:));
4 wind_speed=sqrt(u.^2+v.^2+w.^2);
5 hsurfaces=slice(x,y,z,wind_speed,[xmin,110,xmax],[],[]);
6 set(hsurfaces,'FaceColor','interp','EdgeColor','none');
7 hcont=contourslice(x,y,z,wind_speed,[xmin,100,xmax],[],[]);
8 [sx,sy,sz]=meshgrid(80:5:120,20:5:50,0:5:15);
9 hlines=streamline(x,y,z,u,v,w,sx,sy,sz);
10 set(hlines,'LineWidth',1,'Color','b');
11 xlabel('X');ylabel('Y');zlabel('Z');
12 view(3); daspect([2,2,1]); axis tight;
13 grid on;box on;
14 colorbar;
15 axis([min(x(:)) max(x(:)) min(y(:)) max(y(:)) min(z(:)) max(z(:))]);

```



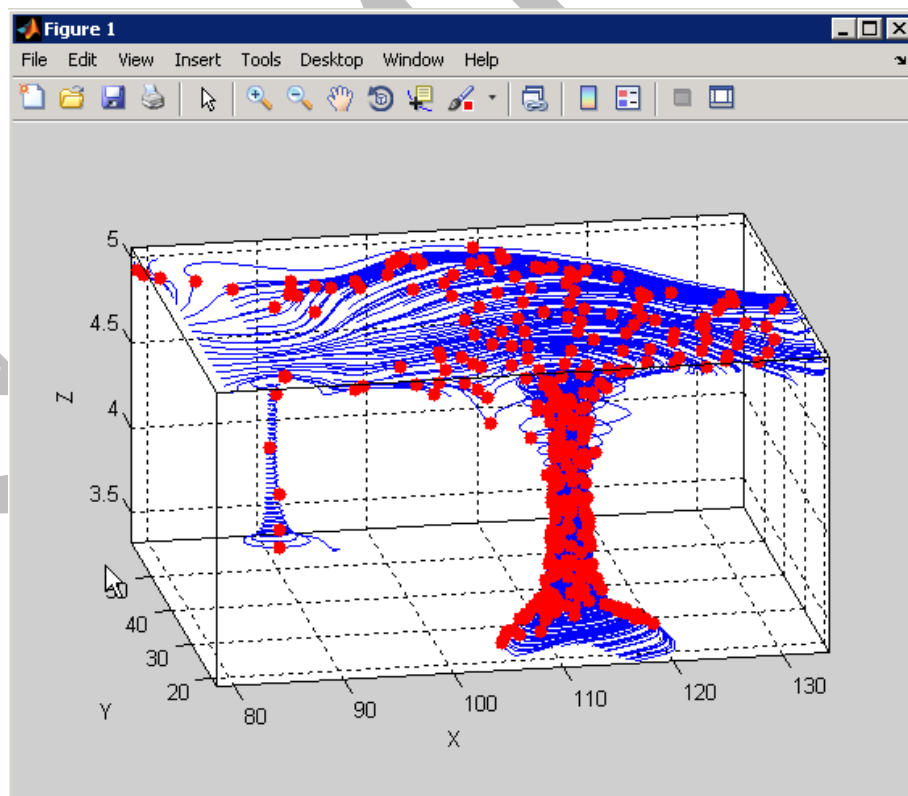
We can also animate that stuff starting from a given height using the function **interpstreamspeed()** and **streamparticles()**:

```

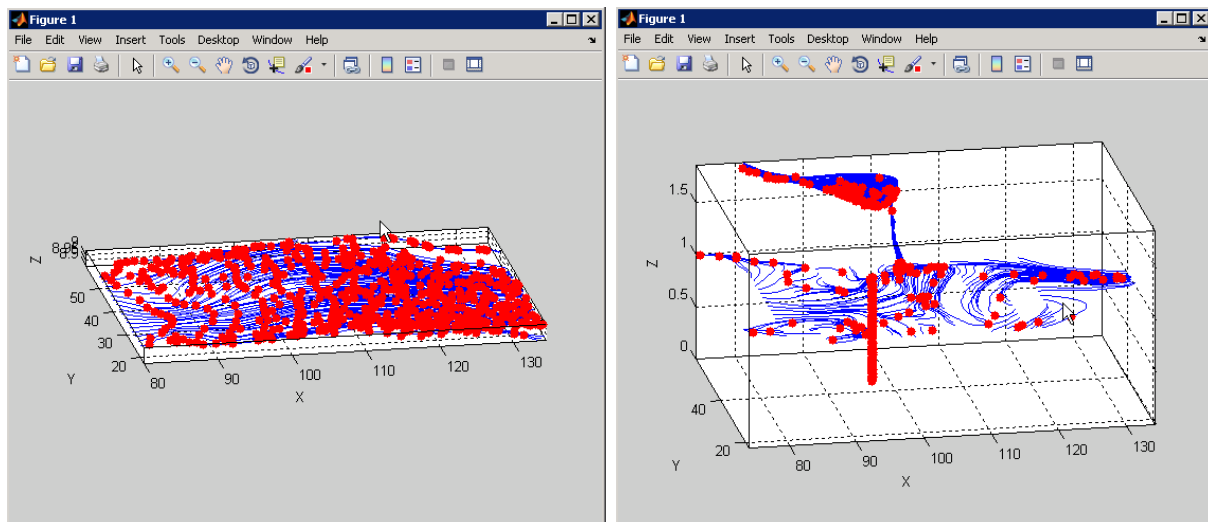
1  clear all;clc;close all
2  step=5;
3  Zheight=5;
4  load wind;
5  [sx,sy,sz]=meshgrid(80:step:120,20:2:50,Zheight);
6  verts=stream3(x,y,z,u,v,w,sx,sy,sz);
7  sl=streamline(verts);
8  view(-10.5,18);
9  daspect([2 2 0.125]);
10 axis tight;box on;grid on
11 iverts =interpstreamspeed(x,y,z,u,v,w,verts,0.05);
12 xlabel('X');ylabel('Y');zlabel('Z');
13 set(gca,'drawmode','fast'); streamparticles(iverts,15,'Animate',10,...
14     'ParticleAlignment','on','MarkerEdgeColor','none','MarkerFaceColor',...
15     'red','Marker','o');
16 axis([min(x(:)) max(x(:)) min(y(:)) max(y(:)) min(z(:)) max(z(:))]);

```

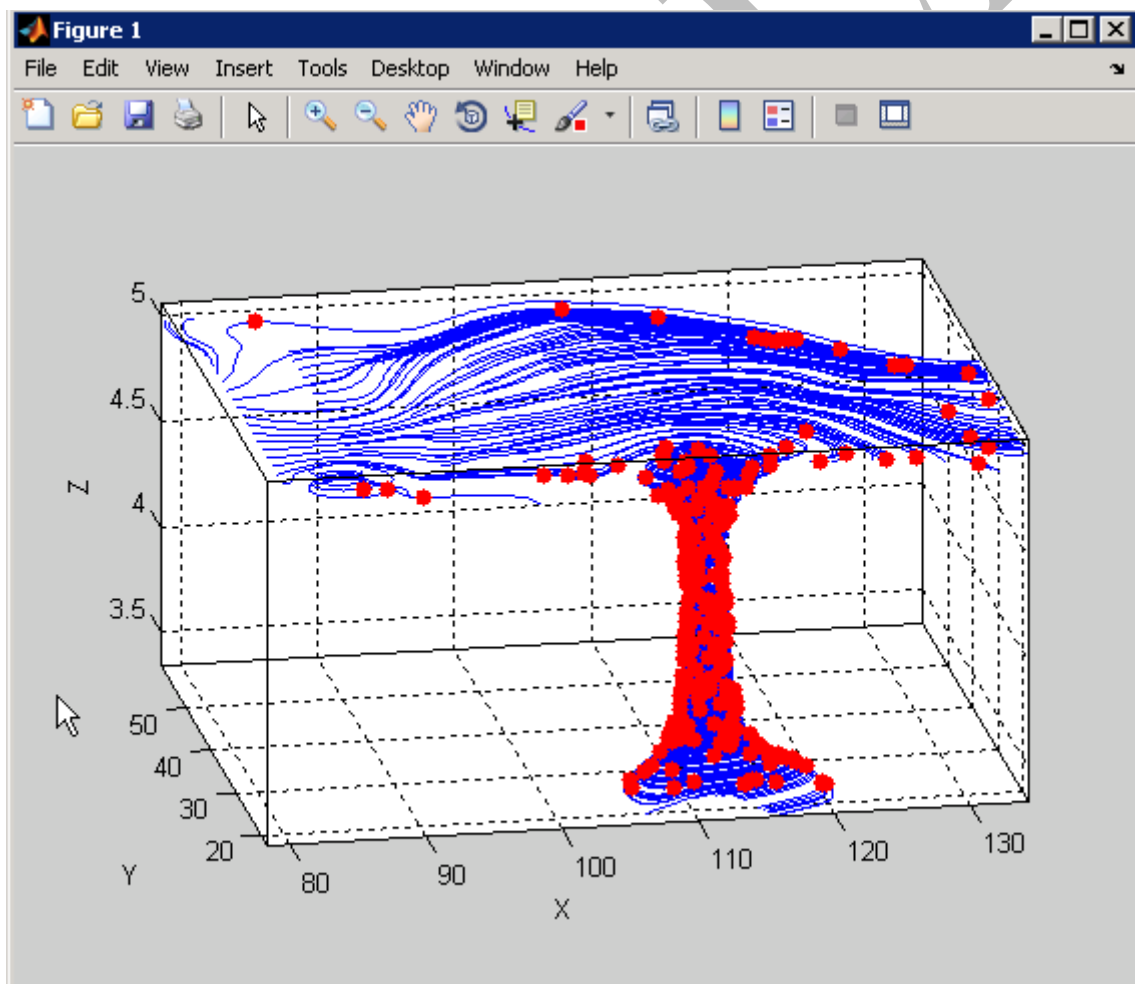
This will give:



If we change de *Zheight* to 9 on the left figure below (or to 1 on the figure on the right below) the result is quite impressive in comparison:



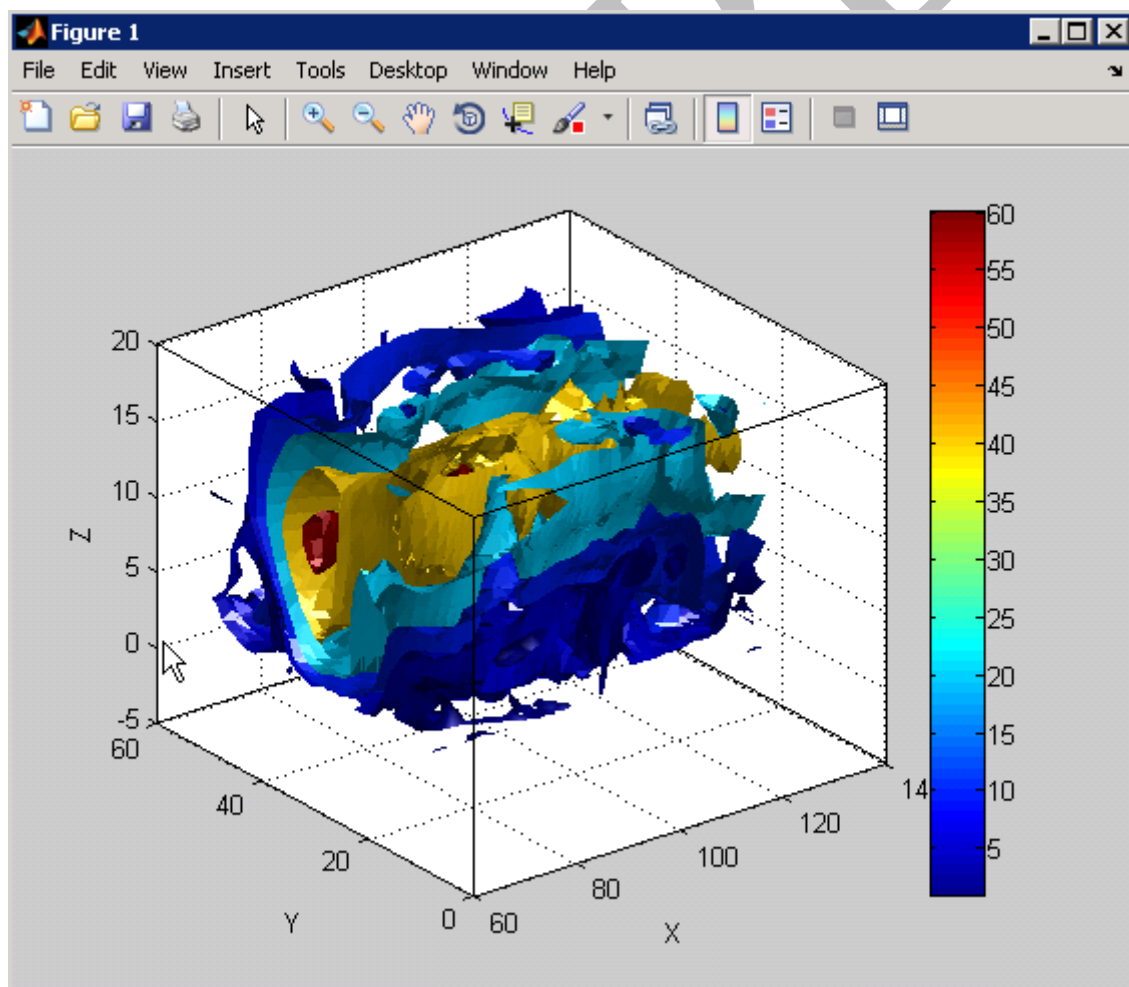
Or for the same *Zheight* (5) and with a *steps* of 6 we also have a surprise (for sure... to big step hide information):



Or we can also play with the function **isosurfaces()**:

```
EDITOR PUBLISH VIEW
1 load wind;
2 spd = sqrt(u.*u + v.*v + w.*w);
3 % Create isosurface patch
4 isosurface(x, y, z, spd,1);
5 hold on;
6 isosurface(x, y, z, spd,5);
7 isosurface(x, y, z, spd,10);
8 isosurface(x, y, z, spd,20);
9 isosurface(x, y, z, spd,40);
10 isosurface(x, y, z, spd,60);
11 colorbar;
12 grid on;box on;
13 xlabel('X');ylabel('Y');zlabel('Z');
```

To get:



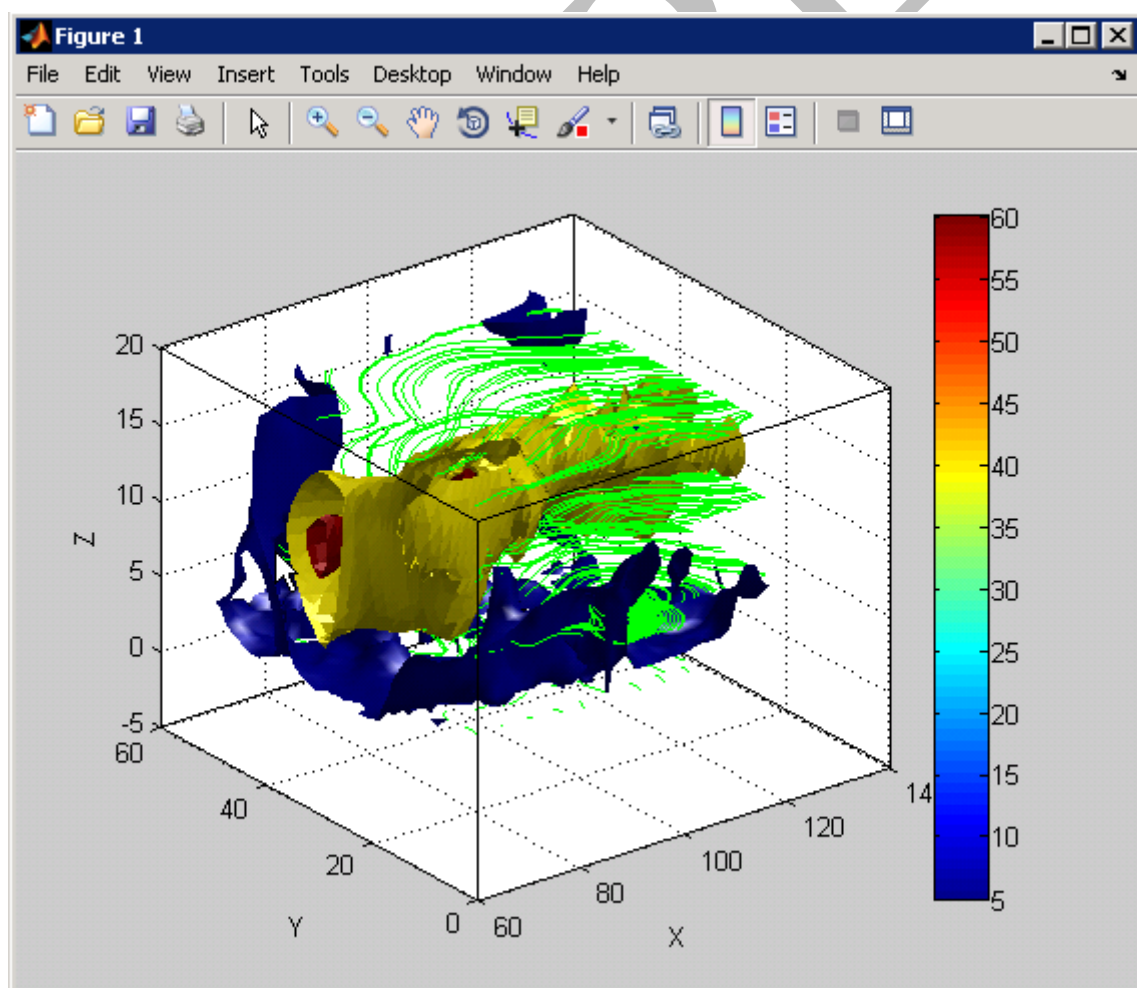
We can also add streamlines to this:

```

EDITOR PUBLISH VIEW
1 load wind;
2 spd = sqrt(u.*u + v.*v + w.*w);
3 % Create isosurface patch
4 isosurface(x, y, z, spd,5);
5 hold on;
6 isosurface(x, y, z, spd,40);
7 isosurface(x, y, z, spd,60);
8 [sx,sy,sz]=meshgrid(80:5:120,20:5:50,0:5:15);
9 h2 = streamline(x,y,z,u,v,w,sx,sy,sz);
10 set(h2,'LineWidth',1, 'Color', 'g');
11 colorbar;
12 grid on;box on;
13 xlabel('X');ylabel('Y');zlabel('Z');

```

This will give:



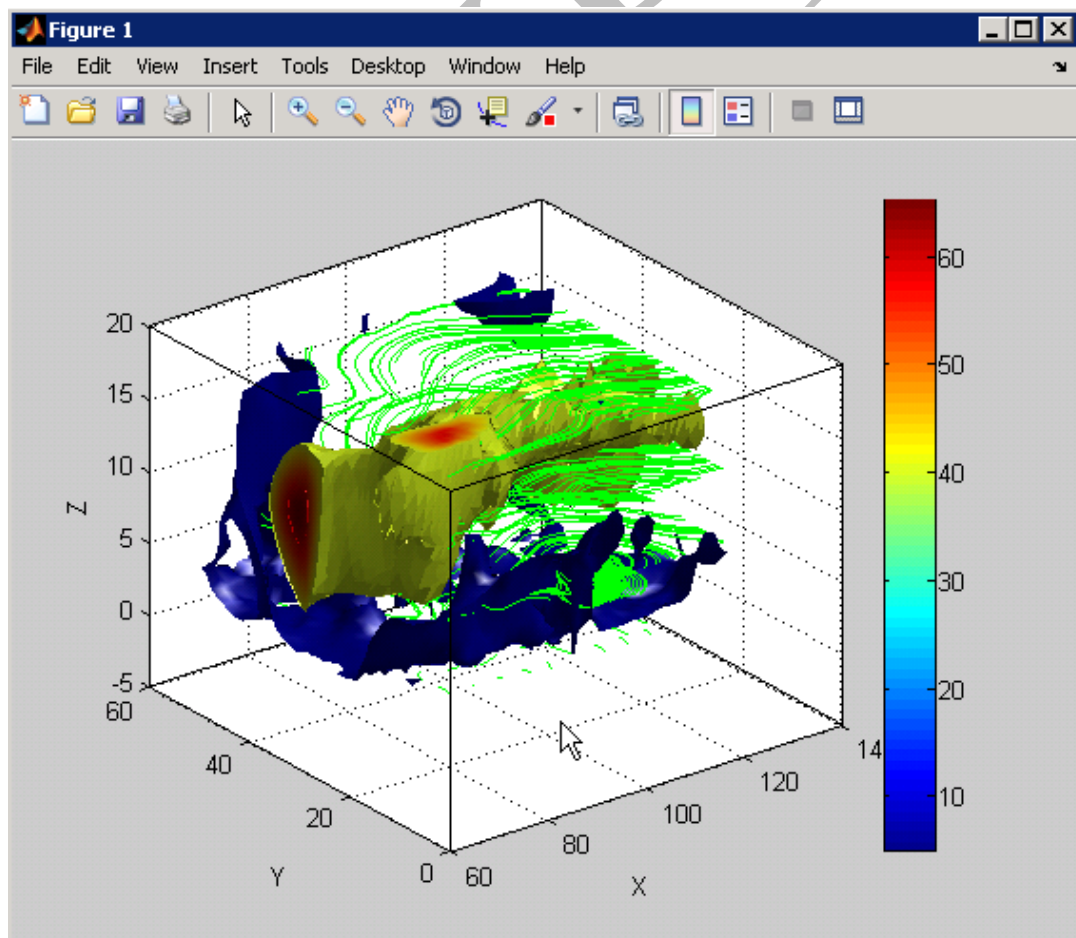
And we can close the holes of a given isosurface using the function **isocaps()**:

```

1      load wind;
2      spd = sqrt(u.*u + v.*v + w.*w);
3      % Create isosurface patch
4      isosurface(x, y, z, spd,5);
5      hold on;
6      isosurface(x, y, z, spd,40);
7      %we close the above isosurface
8      isocaps(x, y, z, spd, 40);
9      isosurface(x, y, z, spd,60);
10     [sx,sy,sz]=meshgrid(80:5:120,20:5:50,0:5:15);
11     h2 = streamline(x,y,z,u,v,w,sx,sy,sz);
12     set(h2,'LineWidth',1, 'Color', 'g');
13     colorbar;
14     grid on;box on;
15     xlabel('X');ylabel('Y');zlabel('Z');

```

This will give you:



etc.

Regression/Interpolation/Extropolation

OLS univariate regression

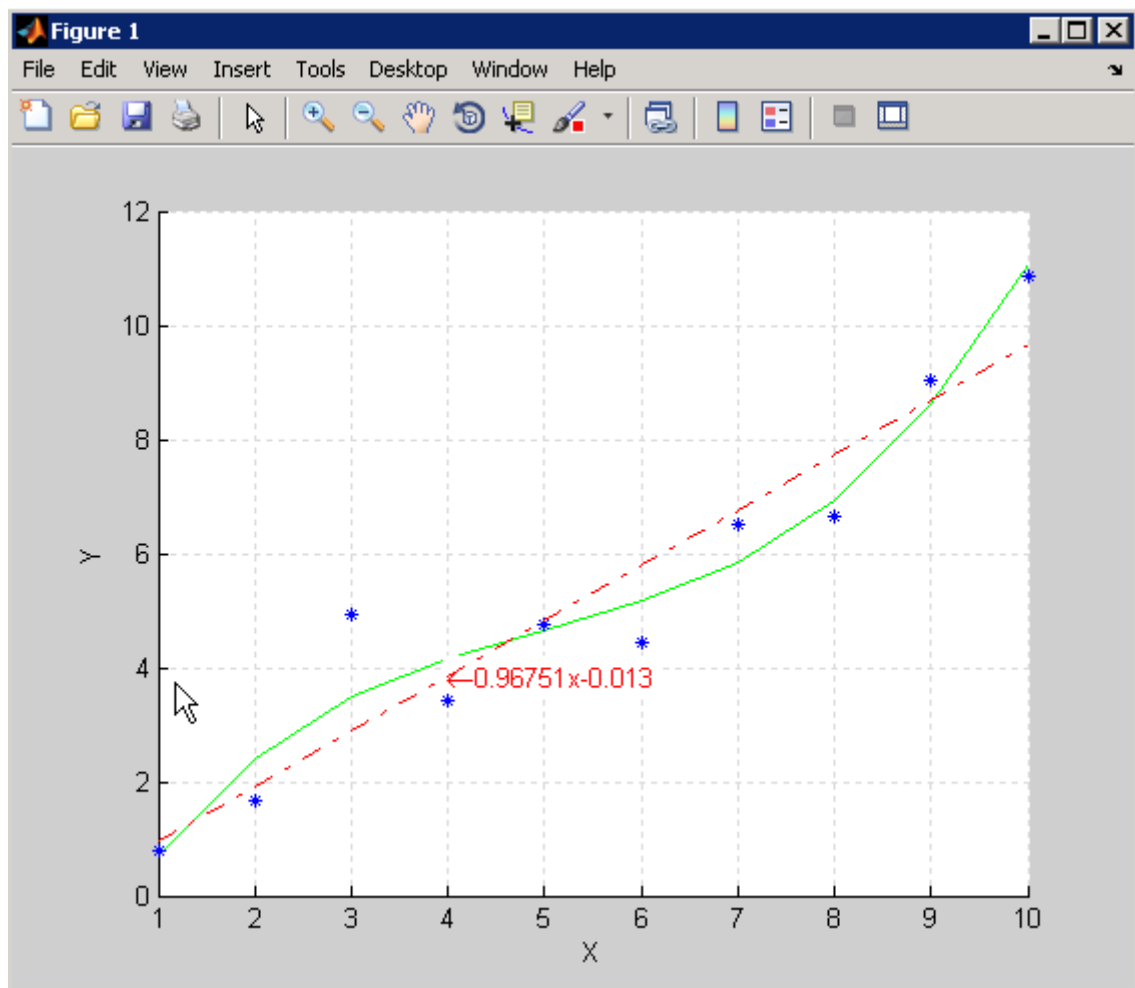
These are some simple OLS regression examples without statistical reports (because you do not need the Statistical Toolbox) using the **polyfit()** function for the regression itself and after a simple **plot()** for the chart.

```

1  %First a simple OLS regression
2  x = 1:10;
3  y = x + randn(1,10);
4  scatter(x,y,25,'b','*')
5  P_line = polyfit(x,y,1);
6  yfit = P_line(1)*x+P_line(2);
7  hold on
8  plot(x,yfit,'r-.');
9  xlabel('X');ylabel('Y');
10 grid on;
11 set(gca,'Xcolor',[0.85 0.85 0.85])
12 set(gca,'Ycolor',[0.85 0.85 0.85])
13 c_axes = copyobj(gca,gcf);
14 set(c_axes,'color','none','xcolor','k','xgrid','off',...
15      'ycolor','k','ygrid','off');
16
17 %And now the same with a third order polynom
18 P_poly = polyfit(x,y,3);
19 yfit = P_poly(1)*x.^3+P_poly(2)*x.^2+P_poly(3)*x+P_poly(4);
20 plot(x,yfit,'-g');
21 P_poly(1);P_poly(2);P_poly(3);
22
23 %we add some texts (att the end to be overs the other plots)
24 text(4,polyval(P_line,4),...
25      strcat('\leftarrow',num2str(P_line(1)),'x',...
26      sprintf('%+3.3f',P_line(2)),['Color','r','BackgroundColor',[1 1 1]]);

```

The result is visible on the next page:



Spline univariate interpolation

Don't forget that with spline interpolation the main definition support must be monotonic increasing and extrapolate before or after the first/last point is a non-sense (this is also why you can't use it finance to extrapolate the first and last missing values of a return vector!):

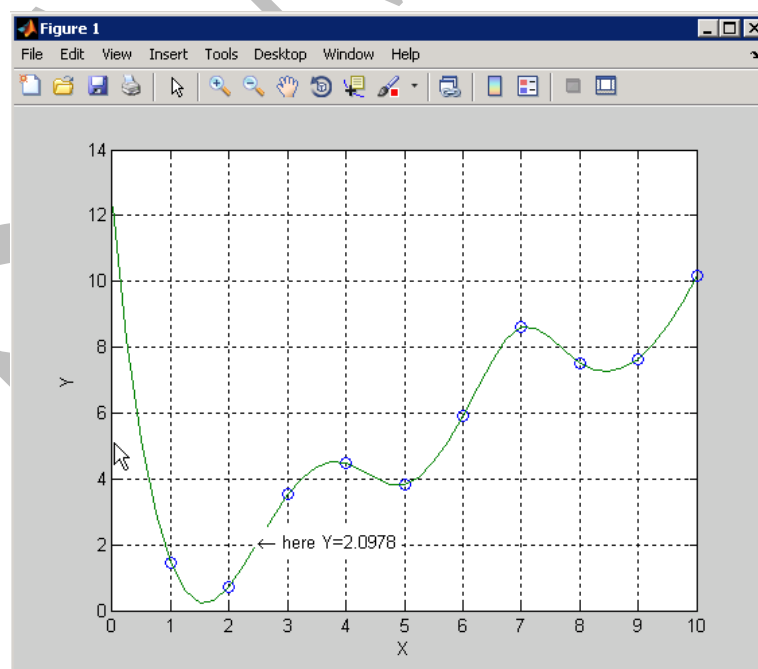
Here is an example with a random vector and the function **spline()**:

```

EDITOR PUBLISH VIEW
1 x = 1:10;
2 y = x + randn(1,10);
3 xx = 0:.25:10;
4 spline_int = spline(x,y,xx);
5 plot(x,y,'o',xx,spline_int);
6
7 %get a value at a given point
8 point_of_interest=2.5;
9 Y_spline=spline(x,y,point_of_interest);
10
11 text(point_of_interest,Y_spline,...
12      strcat('\leftarrow here Y=',num2str(Y_spline)),...
13      'backgroundColor','w');
14 grid on;
15 xlabel('X'),ylabel('Y');

```

The result will be:

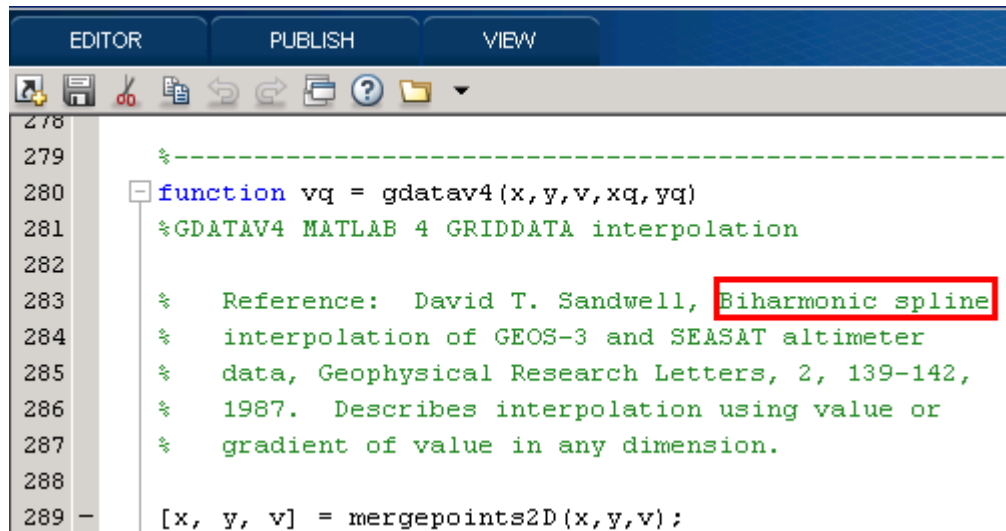


Note: For surface spline interpolation see page 393.

Spline multivariate interpolation

At the page we already saw different technique to interpolate non-uniform vectors measurement points (cubic and Delaunay)

Now have a look to the comparison of the cubic method and spline method call `v4` in MATLAB because as described in the source code of the **`griddata()`** function we have:



```
278
279
280 function vq = gdatav4(x,y,v,xq,yq)
281 %GDATAV4 MATLAB 4 GRIDDATA interpolation
282
283 % Reference: David T. Sandwell, Biharmonic spline
284 % interpolation of GEOS-3 and SEASAT altimeter
285 % data, Geophysical Research Letters, 2, 139-142,
286 % 1987. Describes interpolation using value or
287 % gradient of value in any dimension.
288
289 [x, y, v] = mergepoints2D(x,y,v);
```

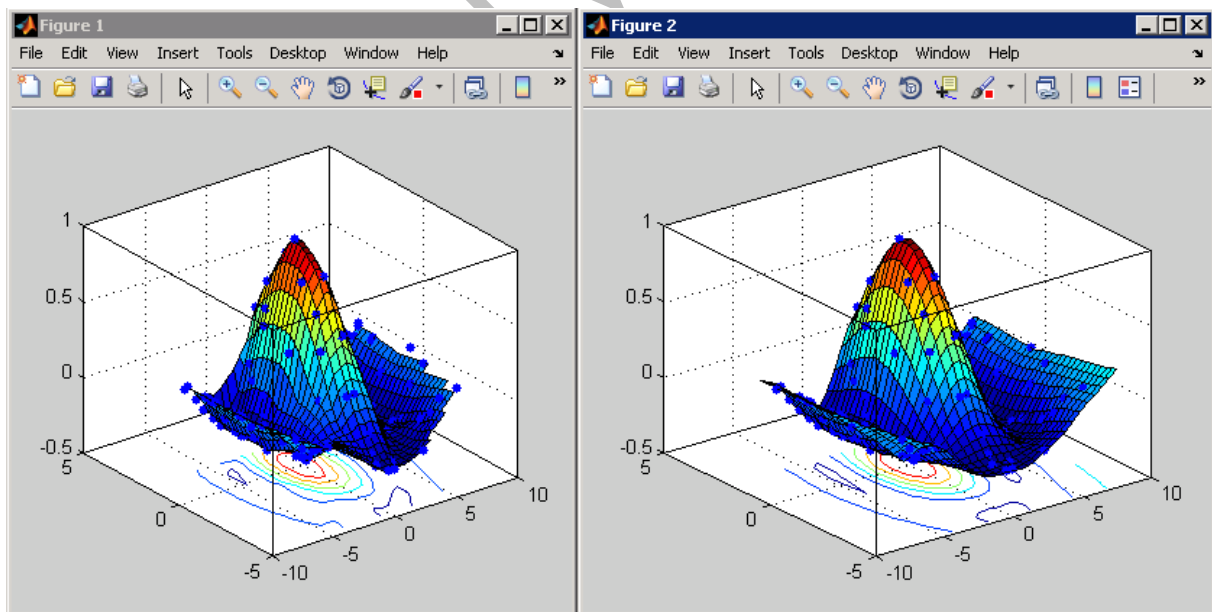
Let us see a comparison example:

```

1 - close all;clear all;clc
2
3 - x=rand(100,1)*16-8;
4 - y=rand(100,1)*7-5;
5 - z=sin(sqrt(x.^2+y.^2))./(sqrt(x.^2+y.^2)+eps);
6
7 - figure(1) %for cubic method
8 - xlin=linspace(min(x),max(x),30);
9 - ylin=linspace(min(y),max(y),30);
10
11 - [X,Y]=meshgrid(xlin,ylin);
12 - Z=griddata(x,y,z,X,Y,'cubic');
13 - surfc(X,Y,Z);
14 - hold on;
15 - plot3(x,y,z,'.','MarkerSize',15);
16 - box on;
17
18 - figure(2) %for spline method
19 - [X,Y]=meshgrid(xlin,ylin);
20 - Z=griddata(x,y,z,X,Y,'v4');
21 - surfc(X,Y,Z);
22 - hold on;
23 - plot3(x,y,z,'.','MarkerSize',15);
24 - box on;

```

This will give (left cubic, right spline):



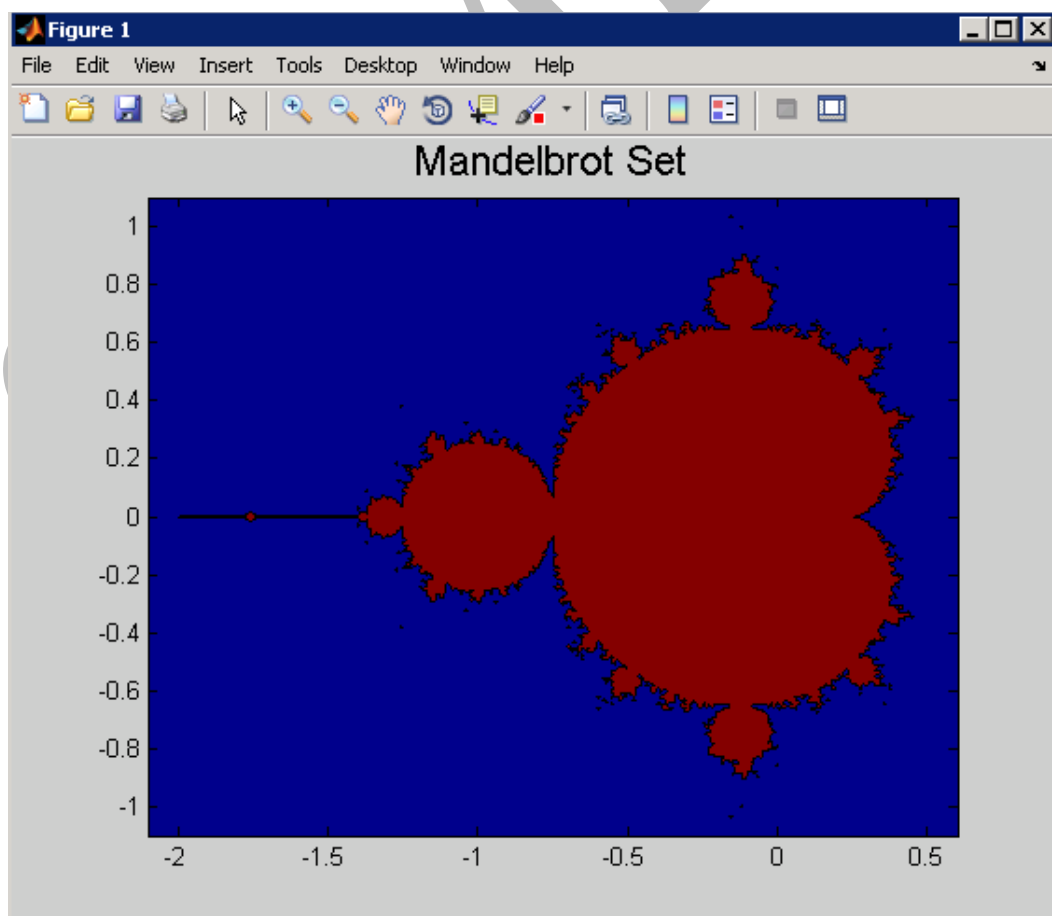
As you can see the spline method is more smoothed.

Fractals

Fractal are use by some financial institutions to model volatility. Then it seems to me useful to first go back to the basics of fractals with the Mandelbrot set:

```
EDITOR PUBLISH VIEW
1 h=waitbar(0,'Computing...');
2 x=linspace(-2.1,0.6,301);
3 y=linspace(-1.1,1.1,301);
4 [X,Y]=meshgrid(x,y);
5 C=complex(X,Y);
6 Z_max=1e6;it_max=50;
7 Z = C;
8 for k = 1:it_max
9     Z = Z.^2 + C;
10    waitbar(k/it_max)
11 end
12 close(h)
13 contourf(x,y,abs(Z)<Z_max,1)
14 title('Mandelbrot Set','FontSize',16)
```

The result will be:

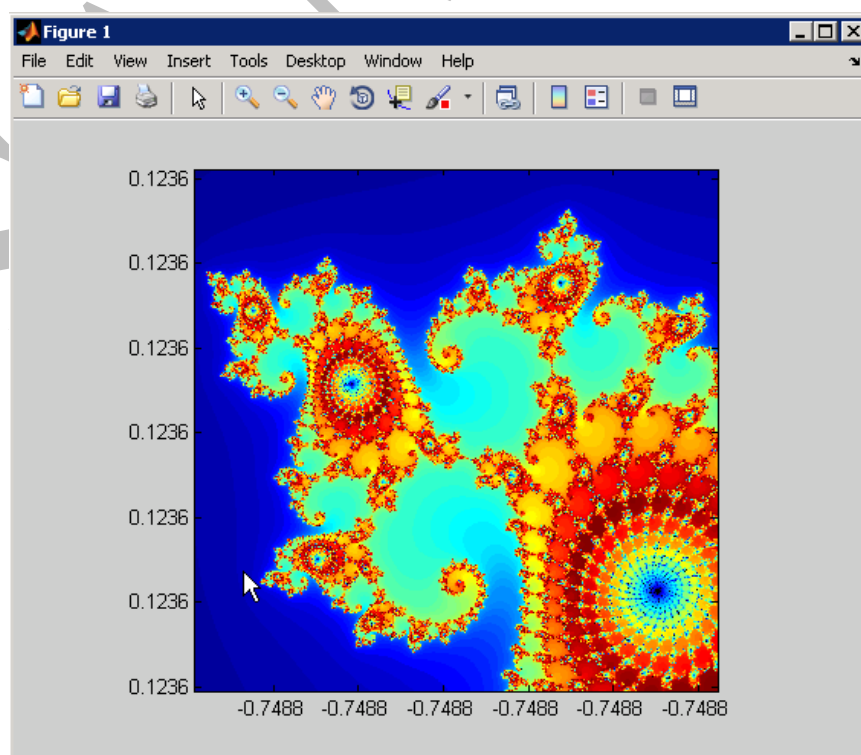


This last figure is a little bit too much old style. Here is a much more pretty way to have the results:

```

EDITOR PUBLISH VIEW
1 maxIterations=500;
2 gridSize=1000;
3 %we can play with those limits
4 xlim = [-0.748766713922161, -0.748766707771757];
5 ylim = [ 0.123640844894862, 0.123640851045266];
6
7 x= linspace( xlim(1), xlim(2), gridSize );
8 y= linspace( ylim(1), ylim(2), gridSize );
9 [xGrid,yGrid] = meshgrid(x,y);
10 z0=xGrid+1i*yGrid;
11 count=zeros(size(z0));
12
13 z = z0;
14 for n = 0:maxIterations
15     z = z.*z + z0;
16     inside=abs(z)<=10;
17     count=count+inside;
18 end
19 count = log( count+1 );
20
21 imagesc(x,y,count);
22 axis image
23 colormap( [jet();flipud( jet() );0 0 0] );|

```



Save chart as image

Save on drive

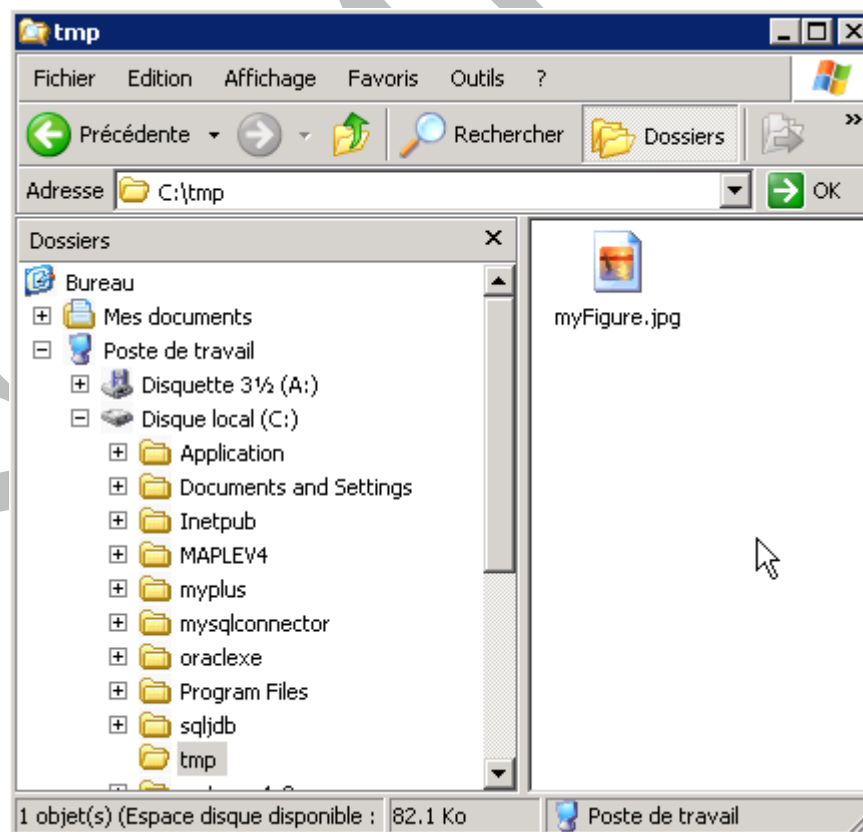
Instead of saving a chart manually as an image it is much more interesting to automate the process. For this purpose we use the command `saveas()`:

```

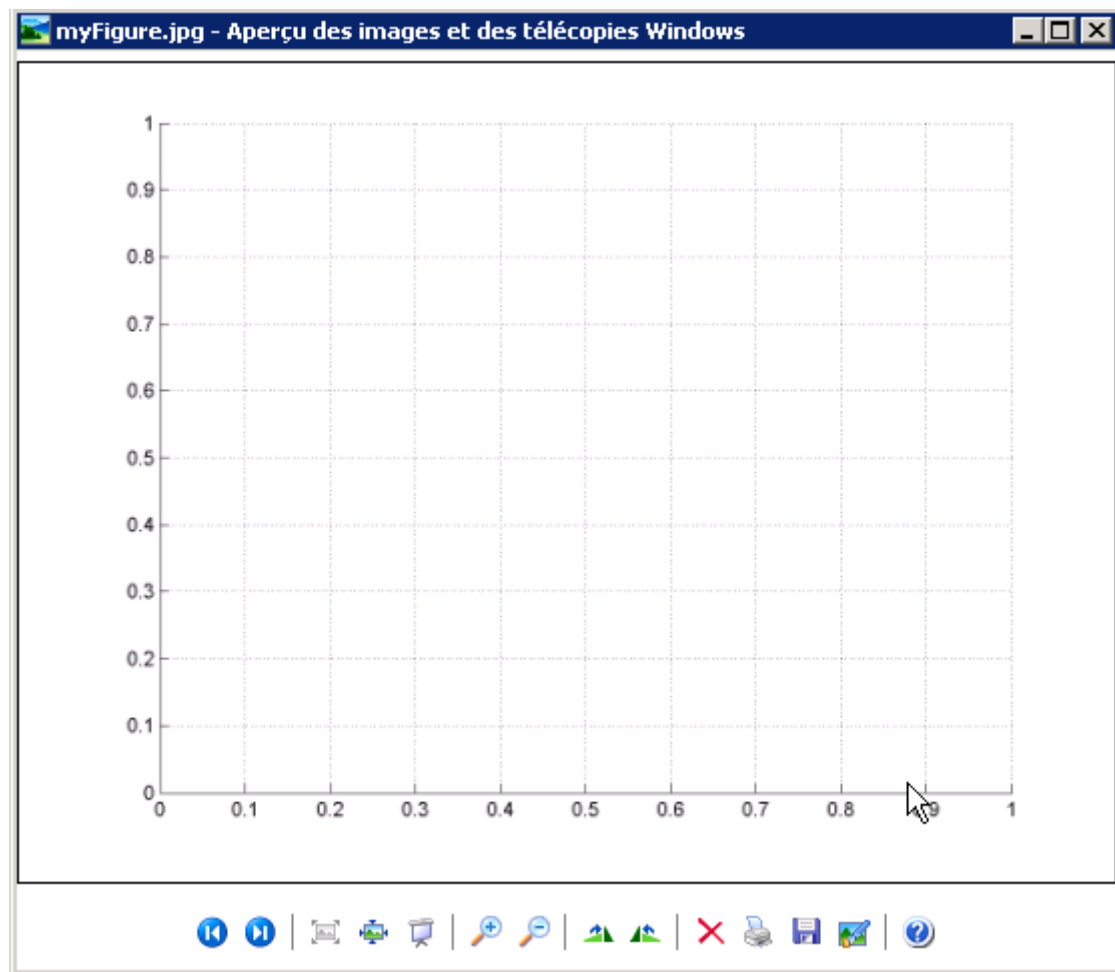
EDITOR PUBLISH VIEW
1 - close all;clear all;clc
2 - f=figure
3 - set(gcf,'Menubar','none','Name','Mon graph','NumberTitle','off',...
4   'Color',[0.8314 0.8157 0.7483]);
5
6 - grid on;
7 - saveas(f,'C:\tmp\myFigure.jpg');
8
9 - set(gca,'Xcolor',[0.85 0.85 0.85]);
10 - set(gca,'Ycolor',[0.85 0.85 0.85]);
11 - setAxes=copyobj(gca,gcf);
12 - set(setAxes,'color','none','Xcolor','k','xgrid','off','Ycolor','k','ygrid','off');
13 - MyBox=uicontrol('style','text');

```

This will create a file:

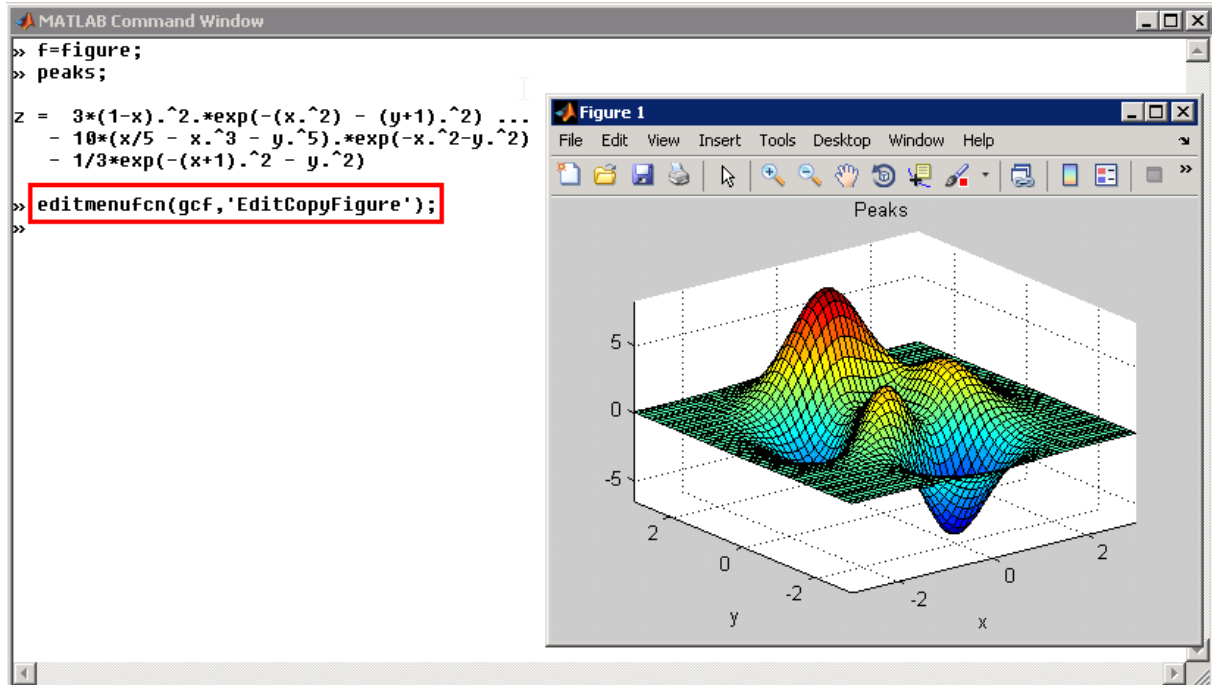


With the following content (empty chart because nothing was plotted):



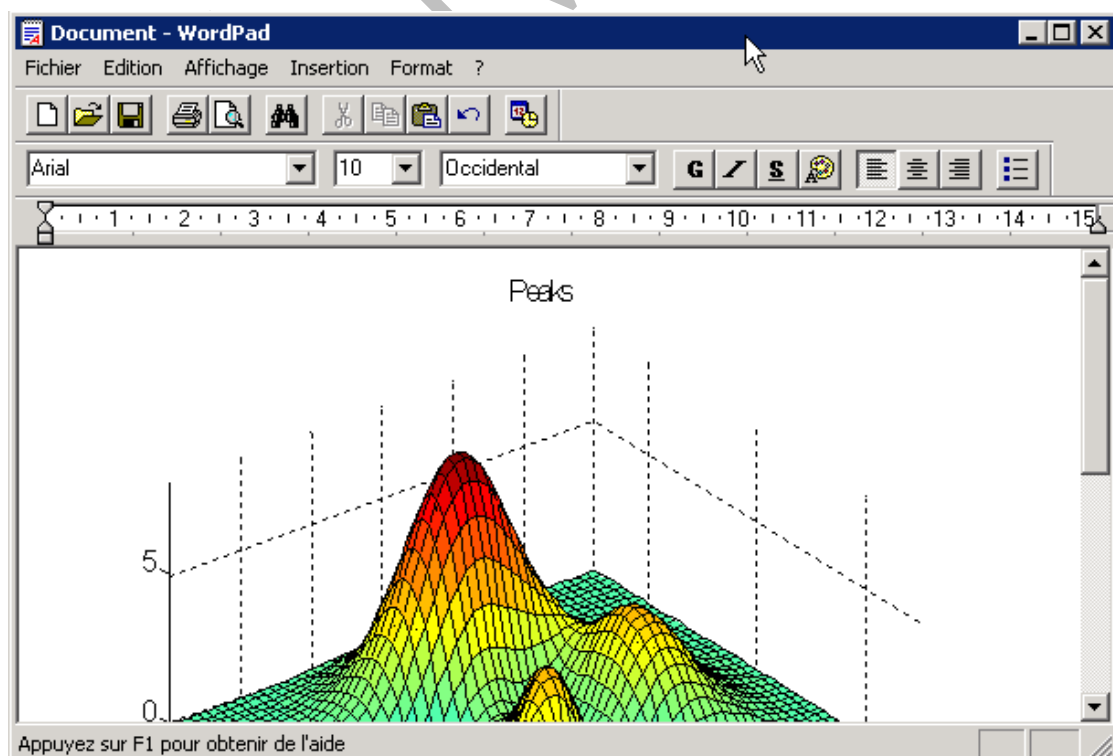
Copy in the clipboard

Most of time the menus and buttons of a chart figure are removed from the figure Windows. Then we have to found also a way to automatize the action to take the figure in the clipboard. For this purpose consider the small following example:



The whole trick resides in the use of the callback function **editmenufcn()**.

Ater the user just has to open the software he wants and make a Ctrl+V:

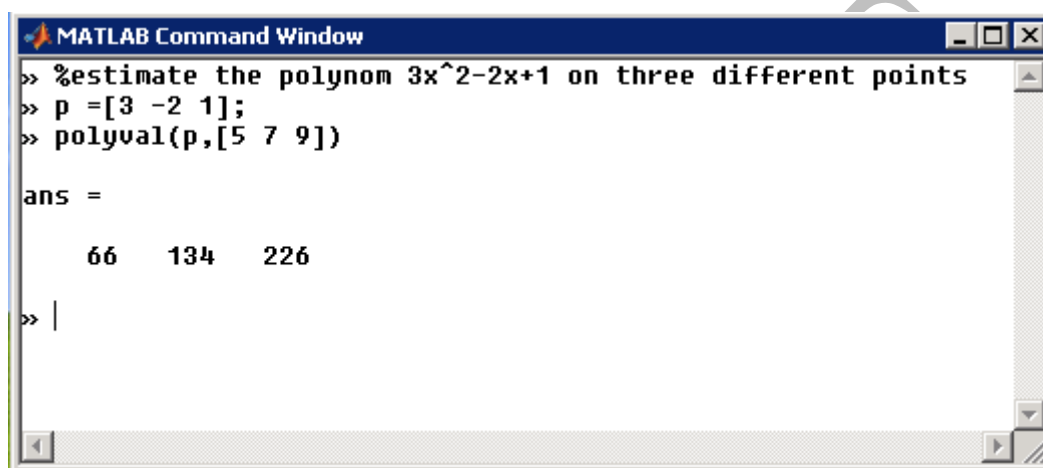


Analysis

This chapter is dedicated to topics related to "math analysis" field. It concern almost all areas of physics, mathematics, engineering, finance, signal (audio/video) processing, etc.

Estimate value of an univariate polynom on various points

Let us begin with something very easy (high school level) that we already have discovered during our study of MATLAB chart. The estimation of a value of a polynom on one or many points using the function **polyval()**:



```

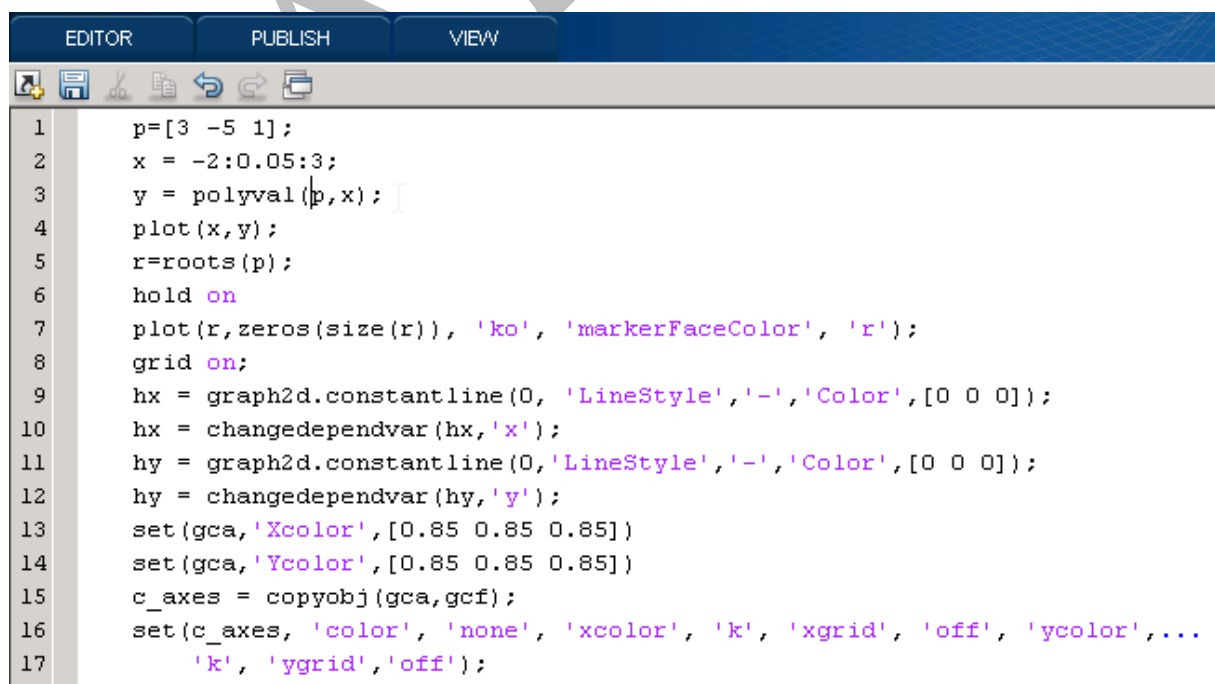
MATLAB Command Window
>> %estimate the polynom 3x^2-2x+1 on three different points
>> p = [3 -2 1];
>> polyval(p,[5 7 9])

ans =

    66    134    226
>> |
  
```

Calculate the roots of an univariate polynom

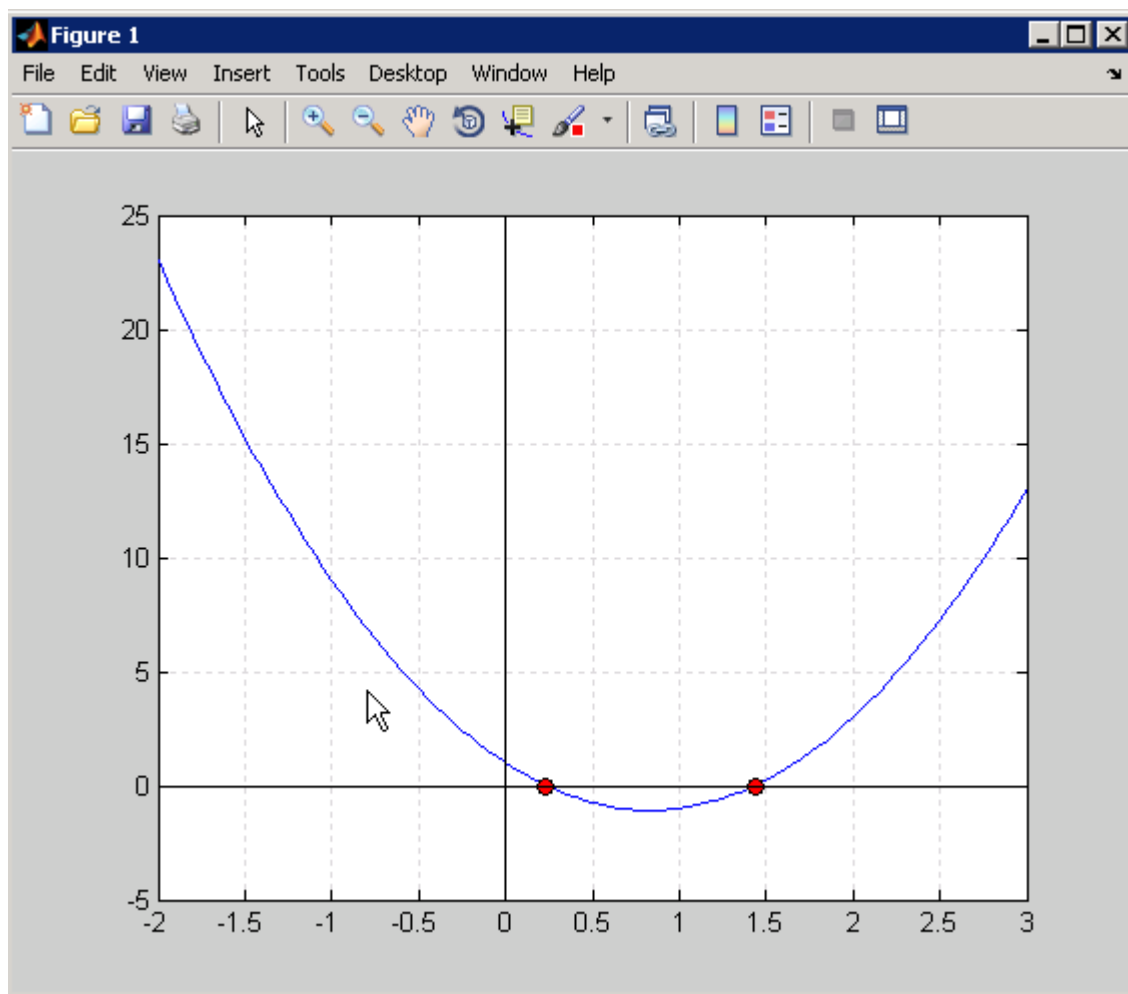
Always at high school level and also something we already used during our study of MATLAB chart, how to calculate the roots of a polynom. There are two methods to do this. The first one is visual and use the functions **polyval()** and **roots()**:



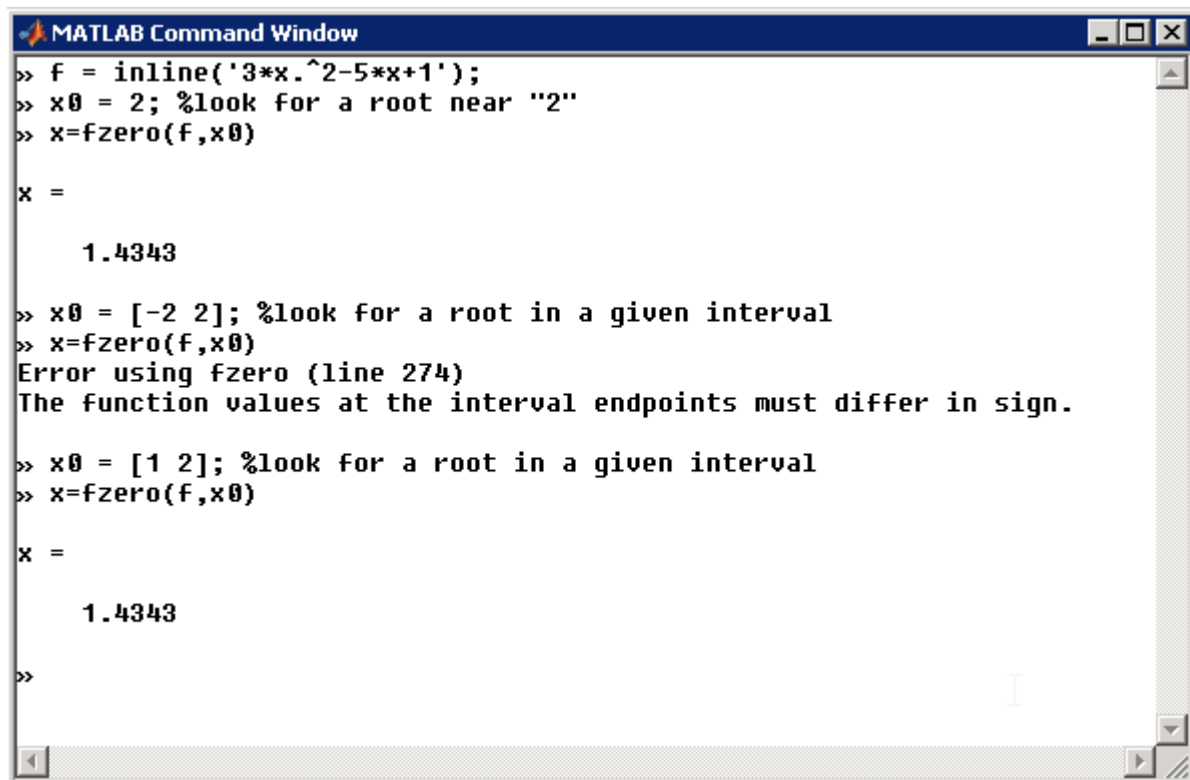
```

EDITOR PUBLISH VIEW
1 p=[3 -5 1];
2 x = -2:0.05:3;
3 y = polyval(p,x);
4 plot(x,y);
5 r=roots(p);
6 hold on
7 plot(r,zeros(size(r)), 'ko', 'markerFaceColor', 'r');
8 grid on;
9 hx = graph2d.constantline(0, 'LineStyle','-','Color',[0 0 0]);
10 hx = changedependvar(hx,'x');
11 hy = graph2d.constantline(0,'LineStyle','-','Color',[0 0 0]);
12 hy = changedependvar(hy,'y');
13 set(gca,'Xcolor',[0.85 0.85 0.85])
14 set(gca,'Ycolor',[0.85 0.85 0.85])
15 c_axes = copyobj(gca,gcf);
16 set(c_axes, 'color', 'none', 'xcolor', 'k', 'xgrid', 'off', 'ycolor',...
17         'k', 'ygrid', 'off');
  
```

This give us a pretty result to see:



The second method is less visual and more "approximative".



```
MATLAB Command Window
>> f = inline('3*x.^2-5*x+1');
>> x0 = 2; %look for a root near "2"
>> x=fzero(f,x0)

x =

    1.4343

>> x0 = [-2 2]; %look for a root in a given interval
>> x=fzero(f,x0)
Error using fzero (line 274)
The function values at the interval endpoints must differ in sign.

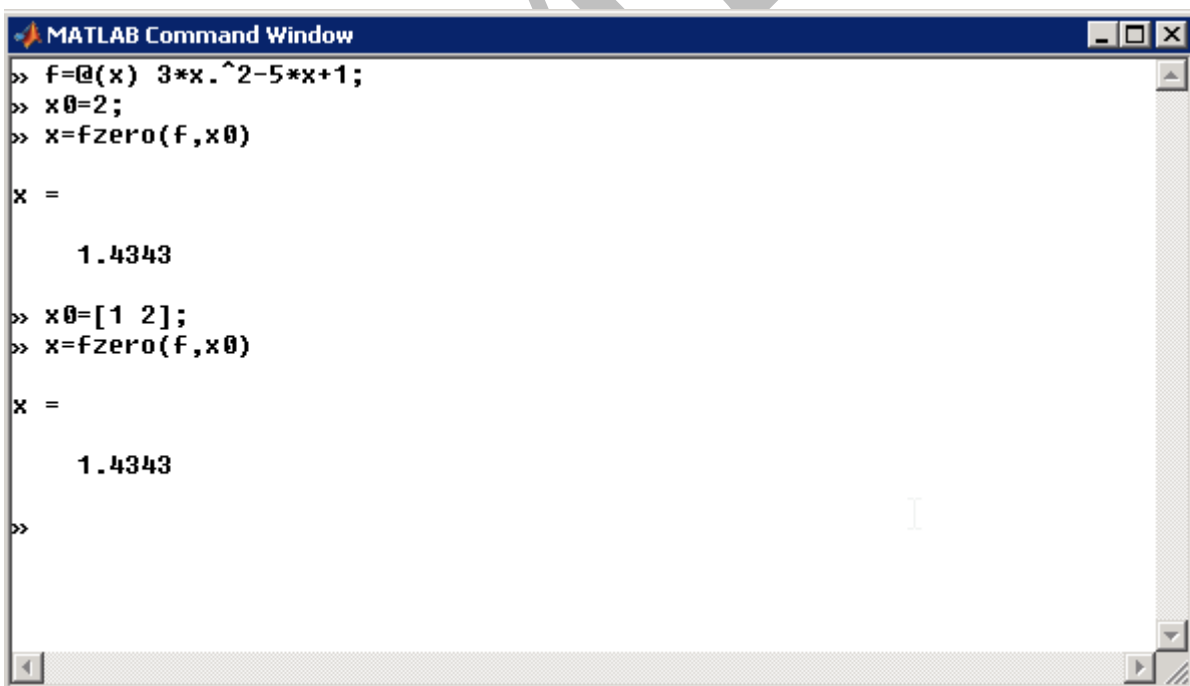
>> x0 = [1 2]; %look for a root in a given interval
>> x=fzero(f,x0)

x =

    1.4343

>>
```

Second method based on "modern notation" with the function **fzero()** that don't use Newton-Raphson method FYI...:



```
MATLAB Command Window
>> f=@(x) 3*x.^2-5*x+1;
>> x0=2;
>> x=fzero(f,x0)

x =

    1.4343

>> x0=[1 2];
>> x=fzero(f,x0)

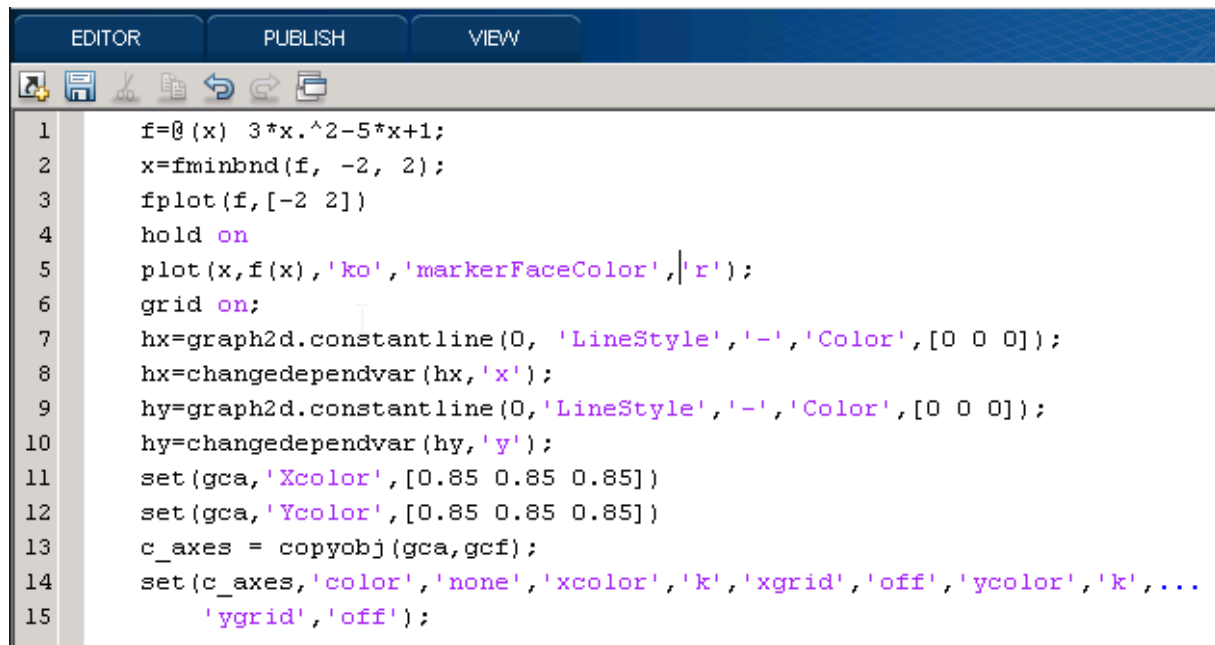
x =

    1.4343

>>
```

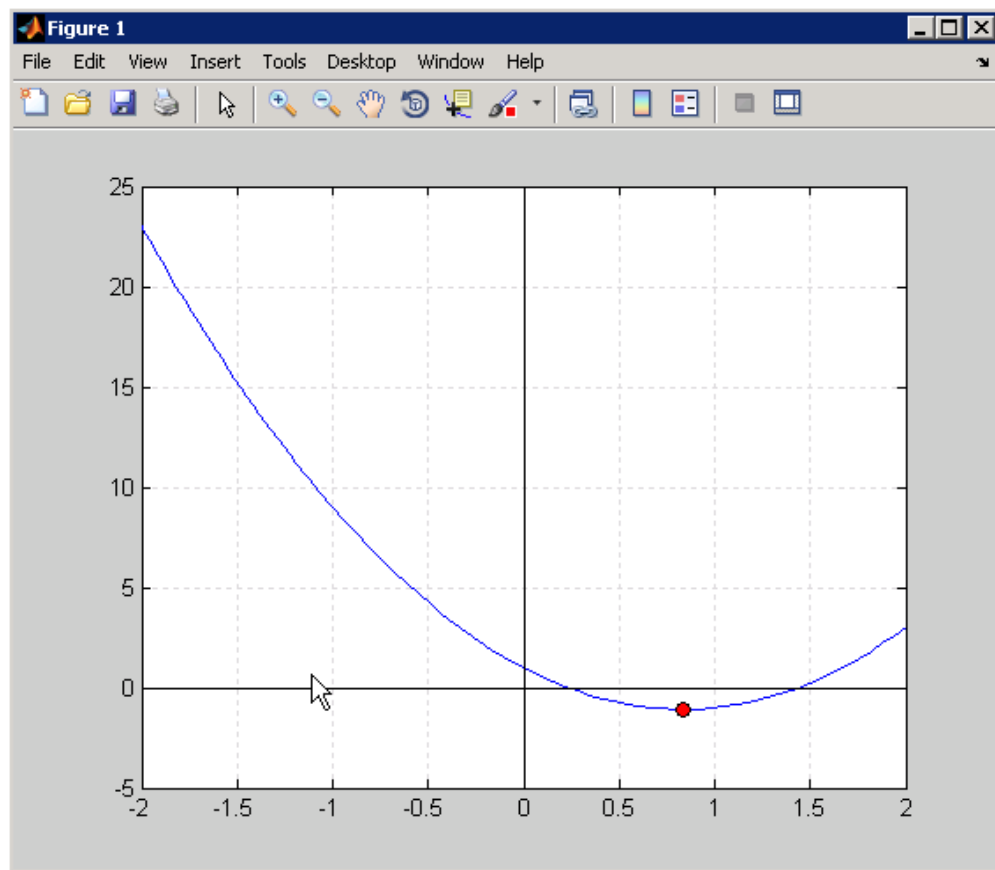
Find the local min/max of an univariate function in a range

The idea here is the same as **fzero()** at the difference that we search a local minima. Why "local"? Because the function **fminbnd()**, for "function minimum bounded", search for only one minima and the smallest unique minima and does not return all minimas of a function (there exist free toolbox for this last purpose):

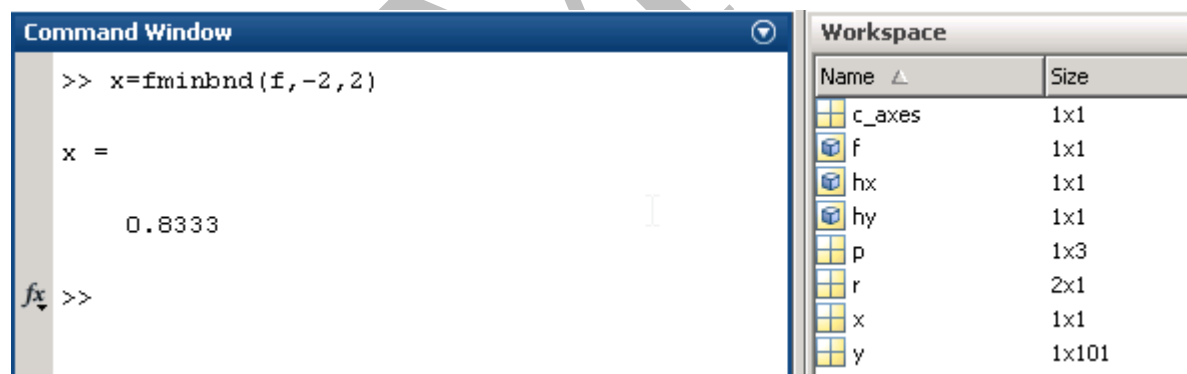


```
1 f=@(x) 3*x.^2-5*x+1;
2 x=fminbnd(f, -2, 2);
3 fplot(f,[-2 2])
4 hold on
5 plot(x,f(x),'ko','markerFaceColor','r');
6 grid on;
7 hx=graph2d.constantline(0, 'LineStyle','-','Color',[0 0 0]);
8 hx=changedependvar(hx,'x');
9 hy=graph2d.constantline(0,'LineStyle','-','Color',[0 0 0]);
10 hy=changedependvar(hy,'y');
11 set(gca,'Xcolor',[0.85 0.85 0.85])
12 set(gca,'Ycolor',[0.85 0.85 0.85])
13 c_axes = copyobj(gca,gcf);
14 set(c_axes,'color','none','xcolor','k','xgrid','off','ycolor','k',...
15      'ygrid','off');
```

This will give:



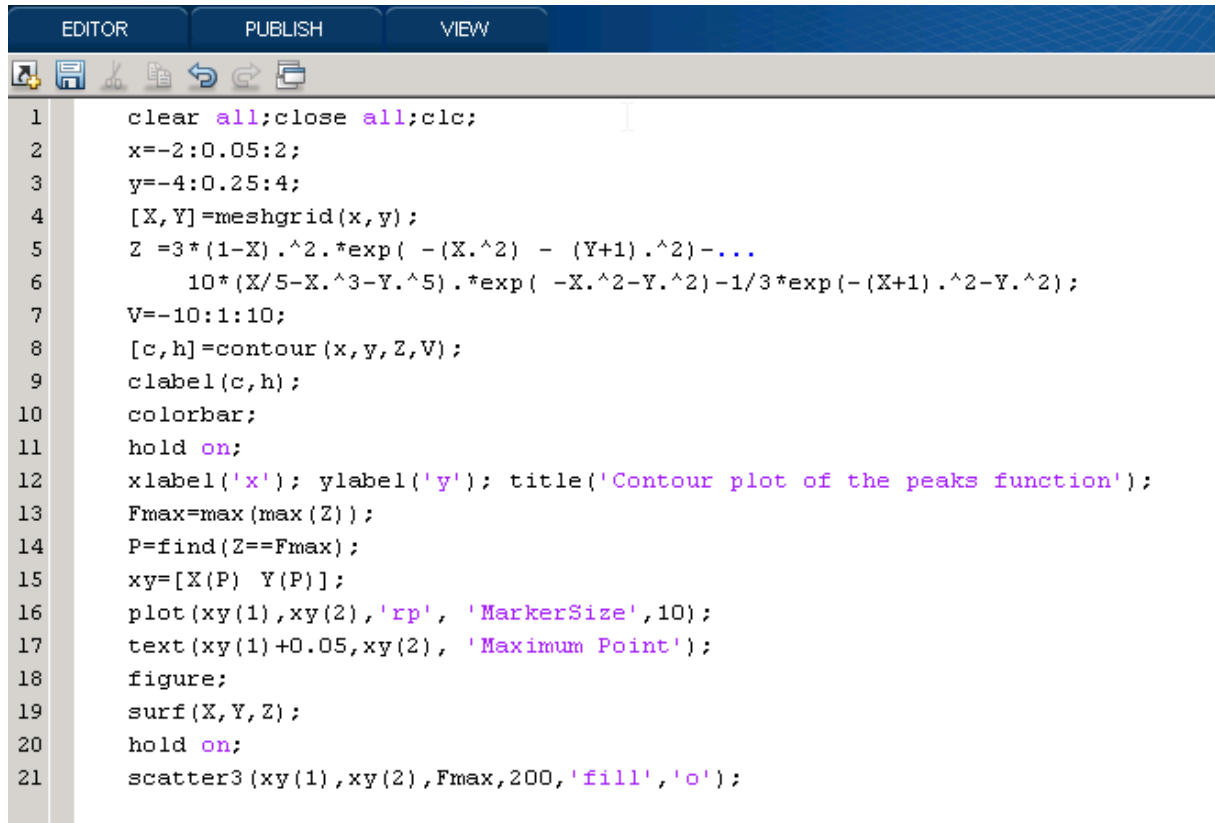
In the *Command Window* this will look like:



Find the local min/max of a bivariate function

1st Method (very old one)

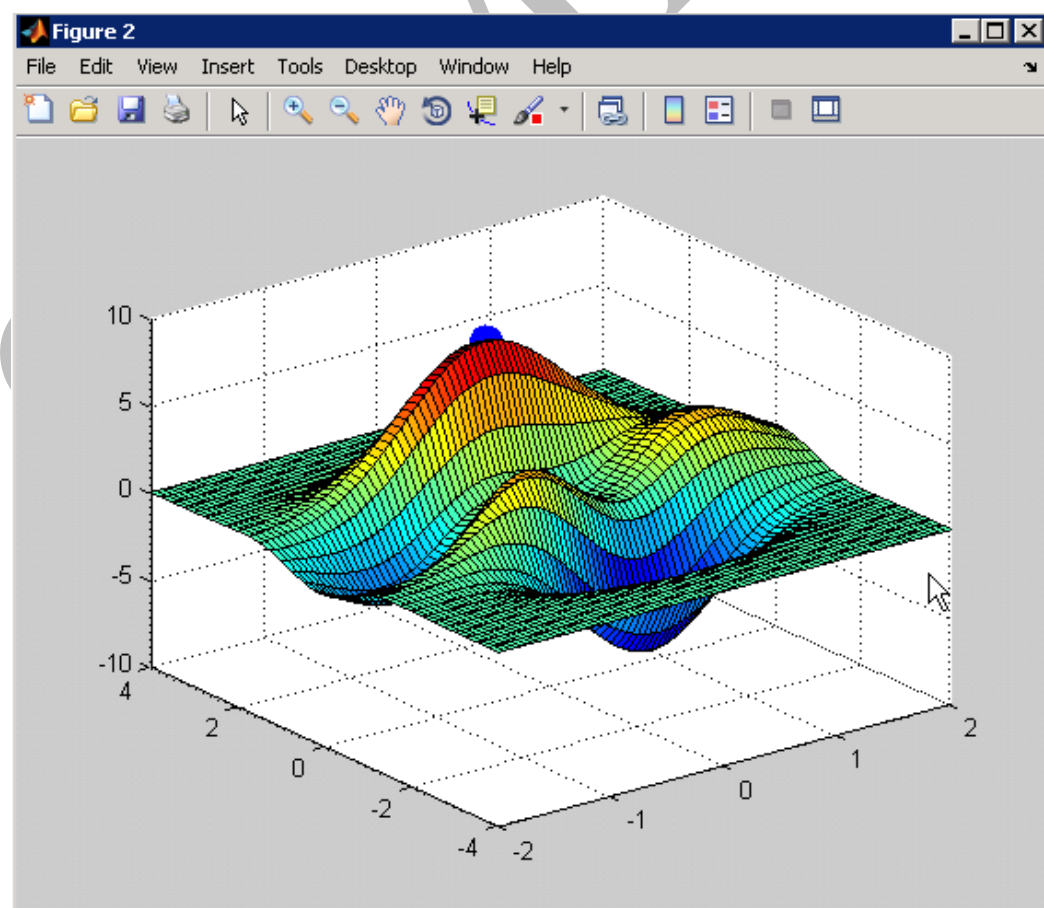
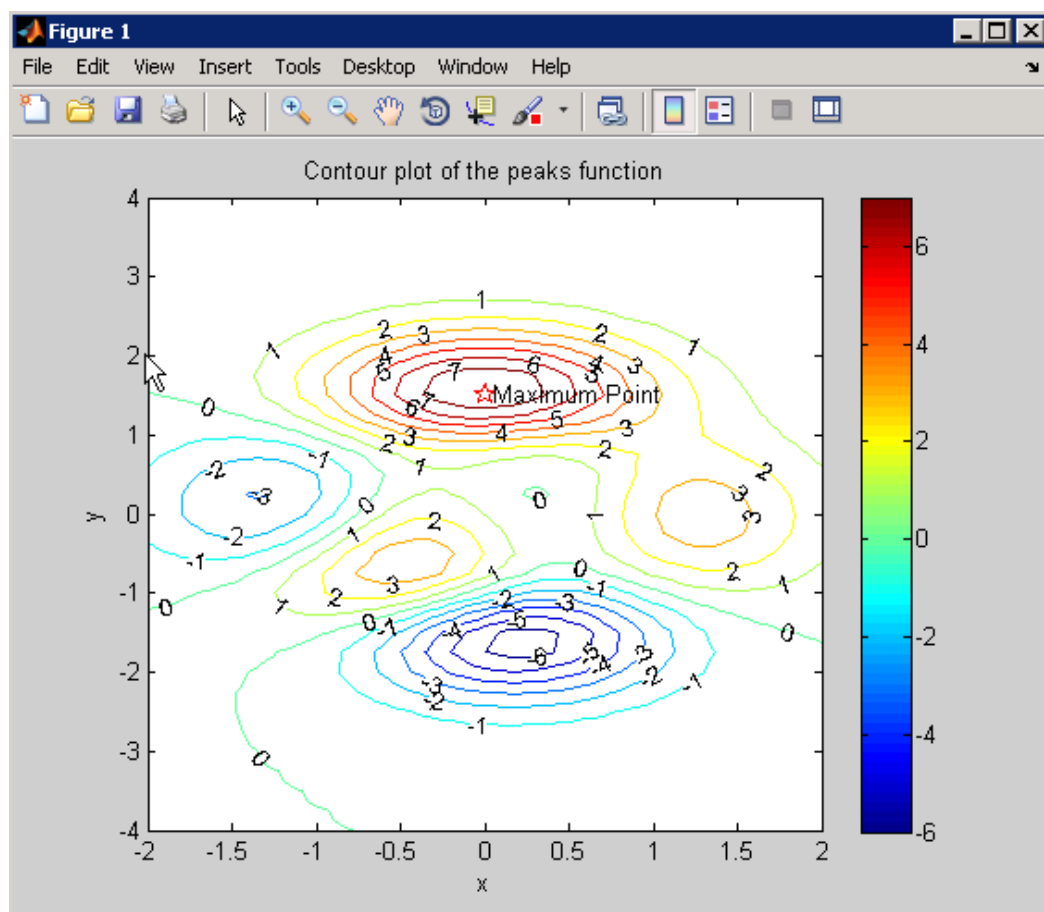
There were no functions at the beginning of MATLAB as far I as know to get the global minimum/maximum of a bivariate function. Thus we were obliged to use a trick using the **max()** and **find()** functions:



```

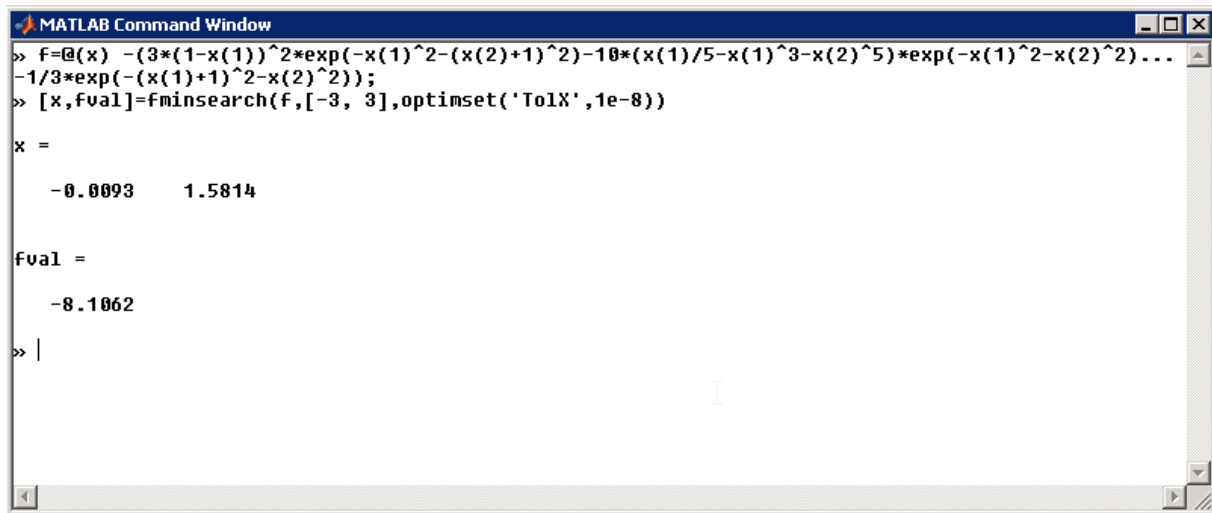
1  clear all;close all;clc;
2  x=-2:0.05:2;
3  y=-4:0.25:4;
4  [X,Y]=meshgrid(x,y);
5  Z =3*(1-X).^2.*exp(-(X.^2) -(Y+1).^2)-...
6      10*(X/5-X.^3-Y.^5).*exp(-X.^2-Y.^2)-1/3*exp(-(X+1).^2-Y.^2);
7  V=-10:1:10;
8  [c,h]=contour(x,y,Z,V);
9  clabel(c,h);
10 colorbar;
11 hold on;
12 xlabel('x'); ylabel('y'); title('Contour plot of the peaks function');
13 Fmax=max(max(Z));
14 P=find(Z==Fmax);
15 xy=[X(P) Y(P)];
16 plot(xy(1),xy(2),'rp','MarkerSize',10);
17 text(xy(1)+0.05,xy(2),'Maximum Point');
18 figure;
19 surf(X,Y,Z);
20 hold on;
21 scatter3(xy(1),xy(2),Fmax,200,'fill','o');
  
```

The result is visible on the next page with two various possibilities:



2nd Method (modern one)

Here we just use the function **fminsearch()** as following:



```

MATLAB Command Window
>> f=@(x) -(3*(1-x(1))^2*exp(-x(1)^2-(x(2)+1)^2)-10*(x(1)/5-x(1)^3-x(2)^5)*exp(-x(1)^2-x(2)^2)...
-1/3*exp(-(x(1)+1)^2-x(2)^2));
>> [x,fval]=fminsearch(f,[-3, 3],optimset('TolX',1e-8))

x =

    -0.0093    1.5814

fval =

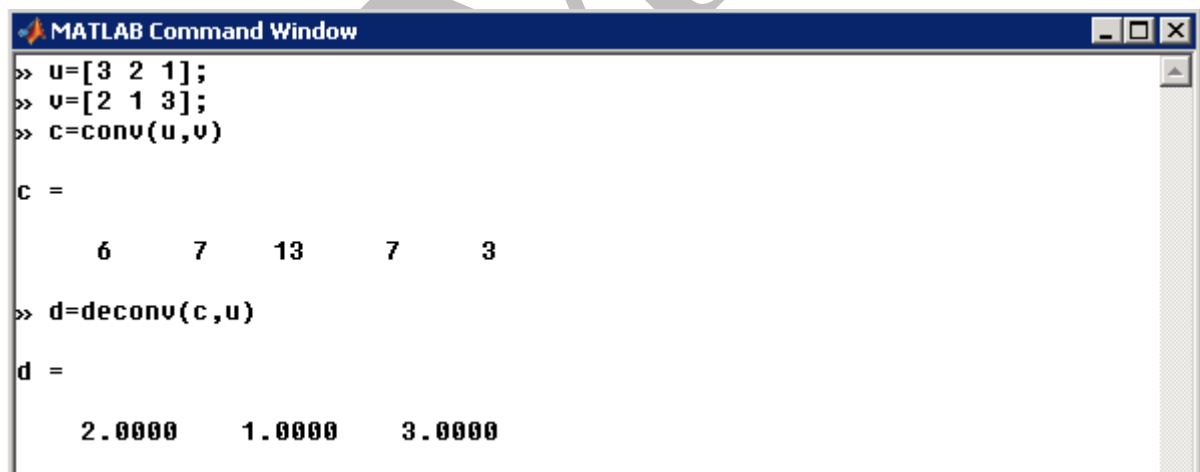
    -8.1062
>> |

```

Discrete convolution

Convolution is a used a lot in signal (and also sound/images) processing to modify some original images to get new ones or to add echo/filters to sounds. Here we will two very elementary examples.

First we control the basics with the function **conv()** and **deconv()**:



```

MATLAB Command Window
>> u=[3 2 1];
>> v=[2 1 3];
>> c=conv(u,v)

c =

     6     7    13     7     3

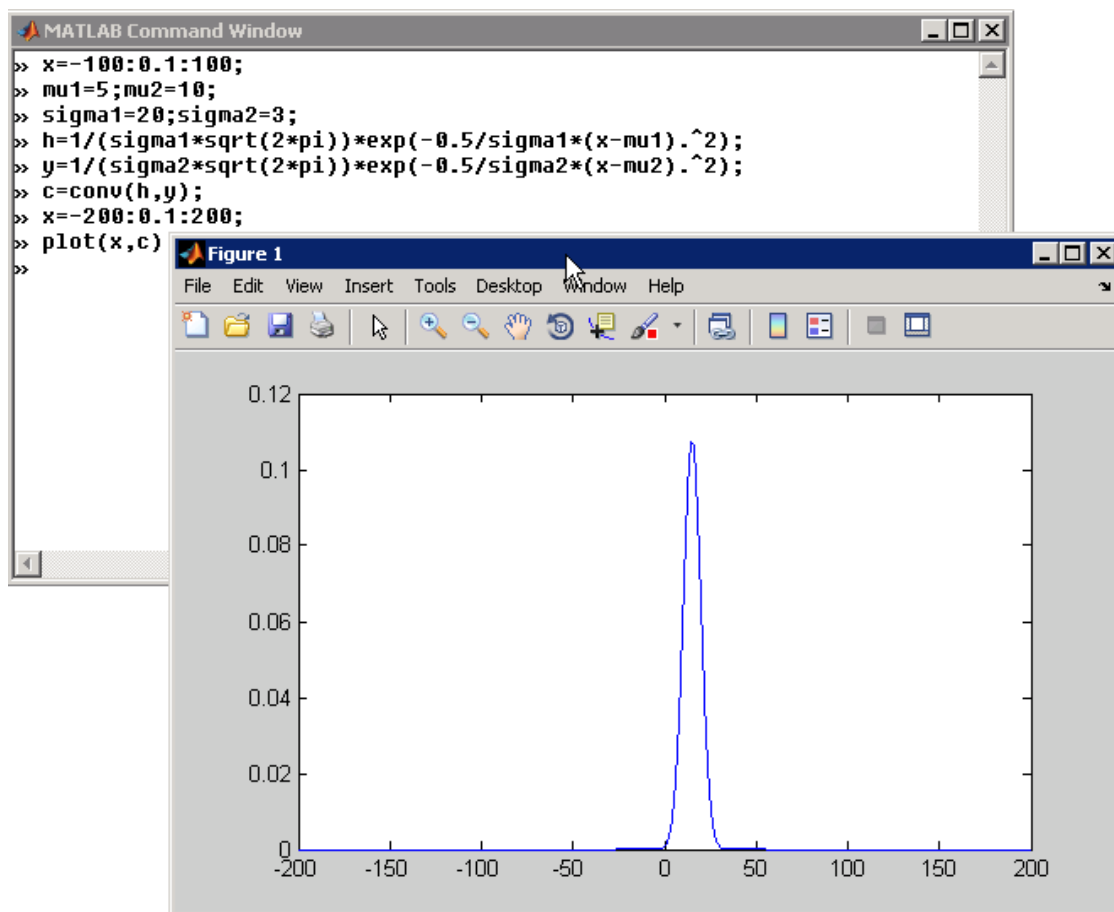
>> d=deconv(c,u)

d =

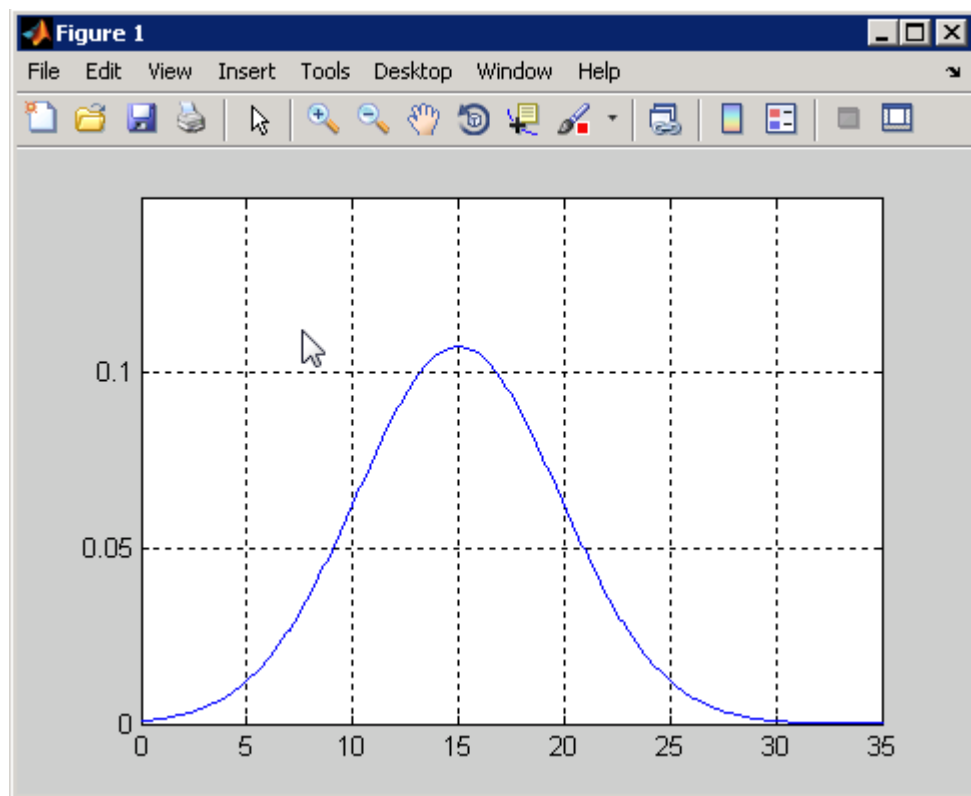
     2.0000     1.0000     3.0000

```

We can also check that the convolution of two Normal random variables is equal to their sum:



And if we zoom , we get:



We must forget de deconvolution in this case!

A study of the U.S. Army Statistical Department has shown that the convolution of random variable is faster than Monte-Carlo but only for two or three random variable. Over 3 random variables, MC is faster.

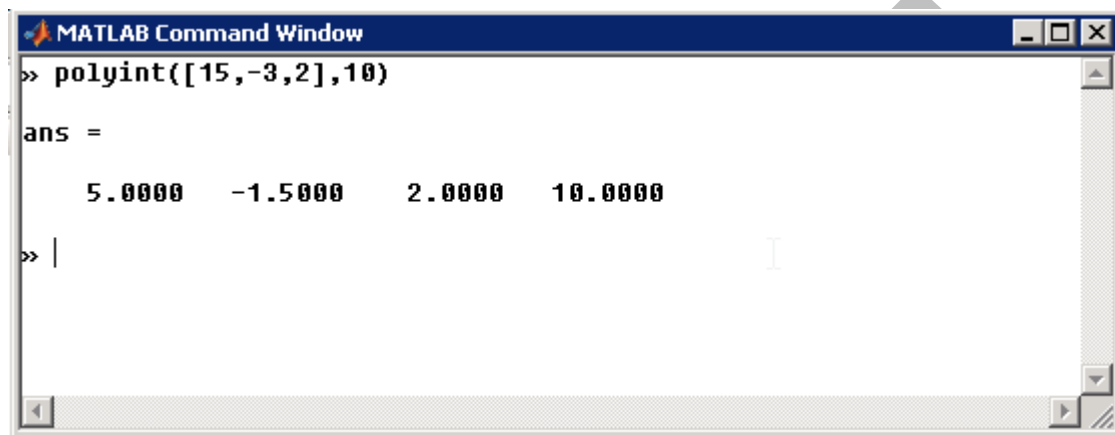
ToDo: Calculate the mean and variance of this convolution

Integration

Integration is not only useful in physics or in pure mathematics but also a lot in engineering finance. Let us see first some very easy applications and further in the chapter about the Statistical Toolbox we will deepen this field using Monte-Carlo technics.

Pseudo-symbolic integration of polynoms

The most easiest integration that can be done in MATLAB is certainly using the function **polyint()** to integrate a simple polynomial. The last argument is the constant of integration...:



```
MATLAB Command Window
>> polyint([15,-3,2],10)

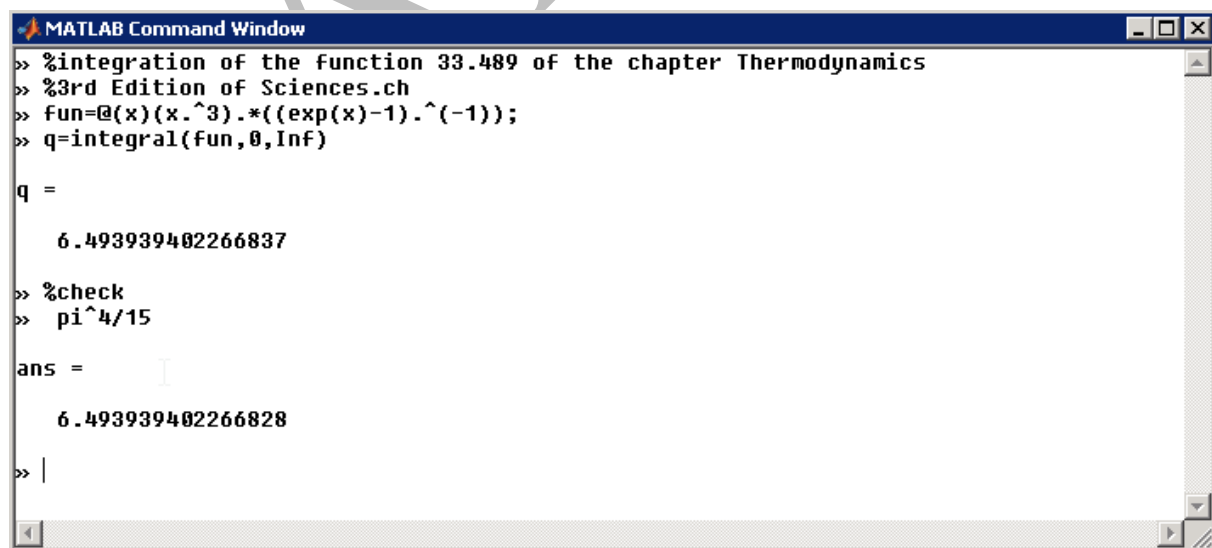
ans =

    5.0000   -1.5000    2.0000   10.0000
>> |
```

Sometimes I think that this function is really a joke...

Numeric integration

Here it begins to be much more interesting! We the function **integral()** we can calculate the integral of many definite functions. We will use an example that made us a lot of trouble in the theoretical chapter of Thermodynamics:



```
MATLAB Command Window
>> %integration of the function 33.489 of the chapter Thermodynamics
>> %3rd Edition of Sciences.ch
>> fun=@(x)(x.^3).*((exp(x)-1).^(-1));
>> q=integral(fun,0,Inf)

q =

    6.493939402266837

>> %check
>> pi^4/15

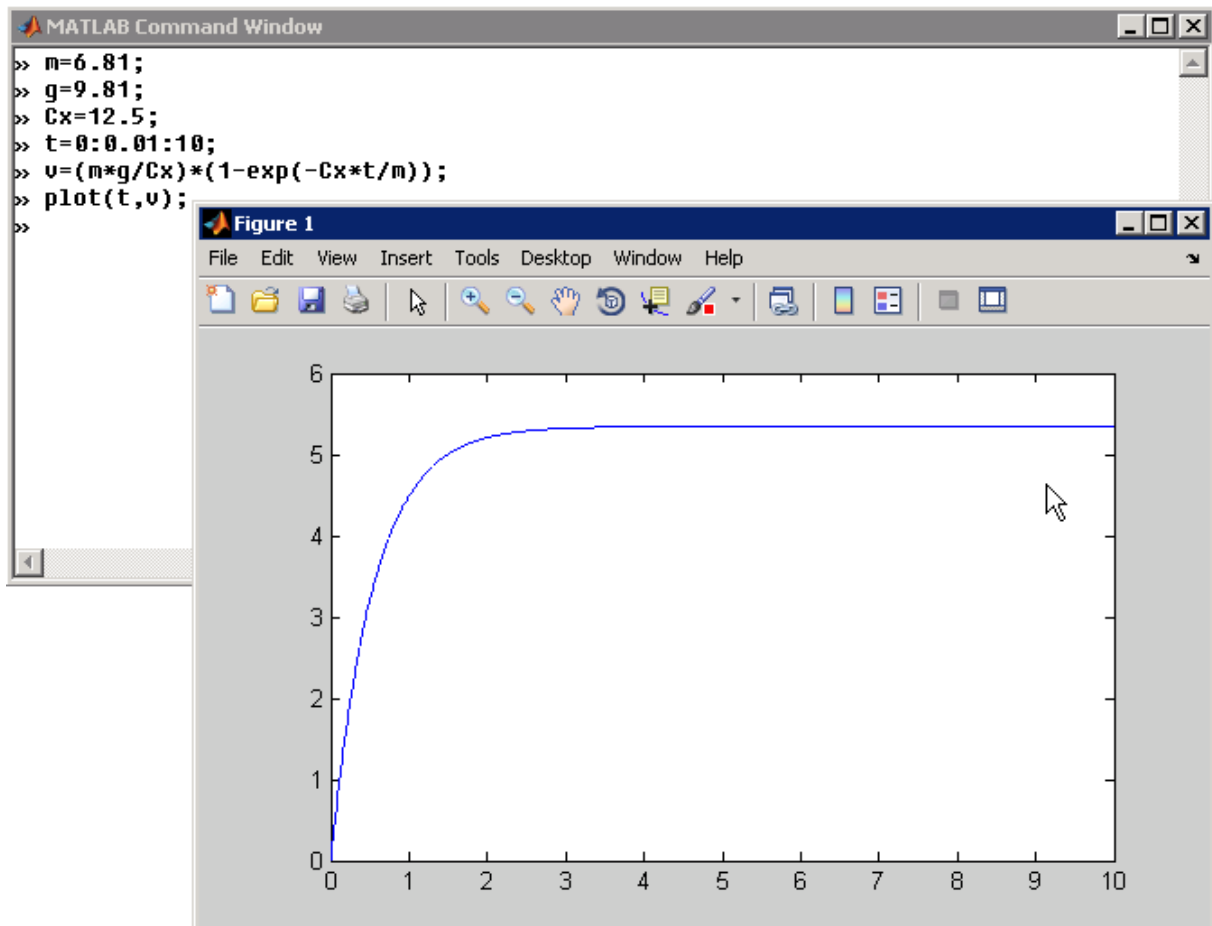
ans =

    6.493939402266828
>> |
```

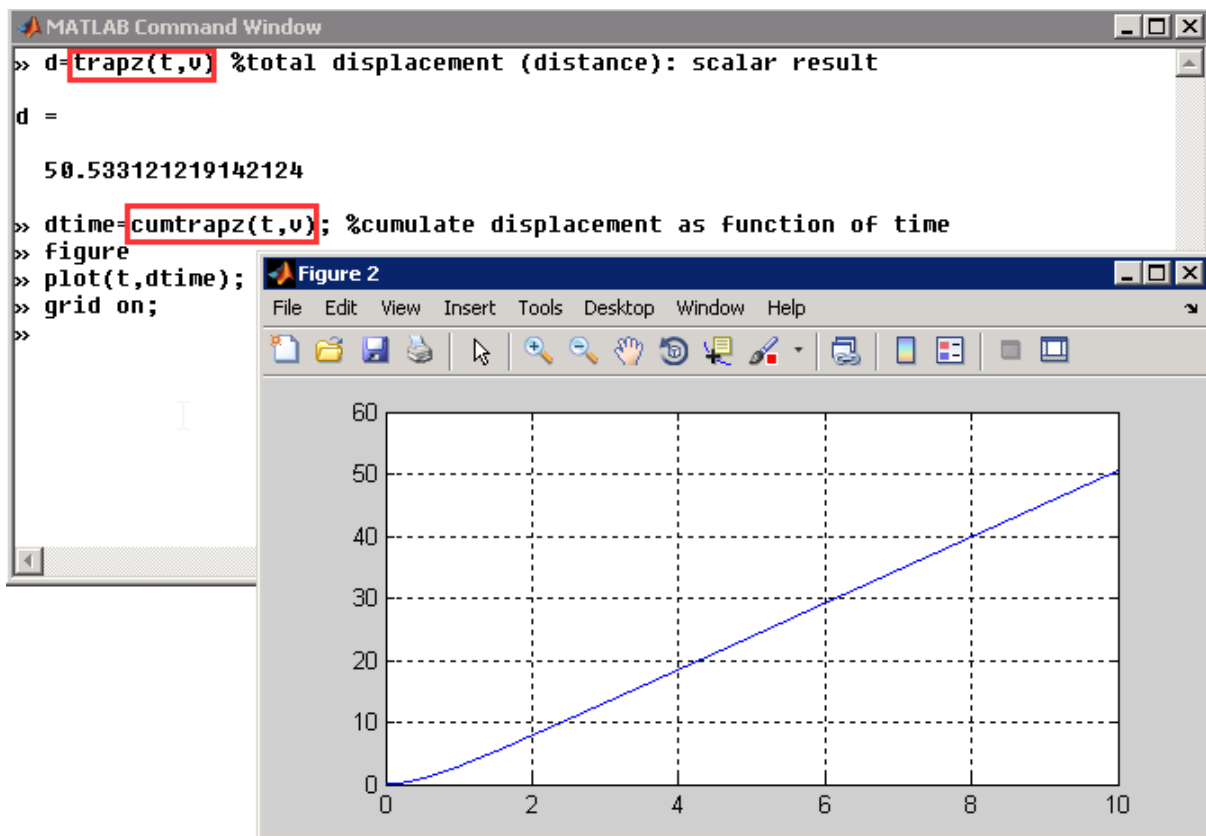
Numeric trapezoidal integration

Even more important: if the function is not explicitly known, we can integrate a vector defined function.

For the example consider the famous academic following example:



And now we calculate the total or cumulated displacement over the time by integration with **trapz()** and **cumtrapz()** thanks to the fact that we know the velocity at each moment:



Double numeric integration

Use a lot in statistics and finance the double integration for explicitly know functions can be done thanks to **integral2()**:

```

MATLAB Command Window
>> mu1=3;mu2=2;
>> sigma1=sqrt(10);sigma2=sqrt(5);
>> rho=5/sqrt(5*10);
>> pdf=@(x,y) (1/(2*pi*sqrt(sigma1^2*sigma2^2*(1-rho^2))))*exp((-1/(2*(1-rho^2)))*(((x-mu1)/sigma1).^2+..
((y-mu2)/sigma2).^2-2*rho*((x-mu1)/sigma1).*((y-mu2)/sigma2)));
>> integral2(pdf,-3,4,-5,2)

ans =

    0.4007
>> integral2(pdf,-Inf,Inf,-Inf,Inf)

ans =

    1.0000
>> |

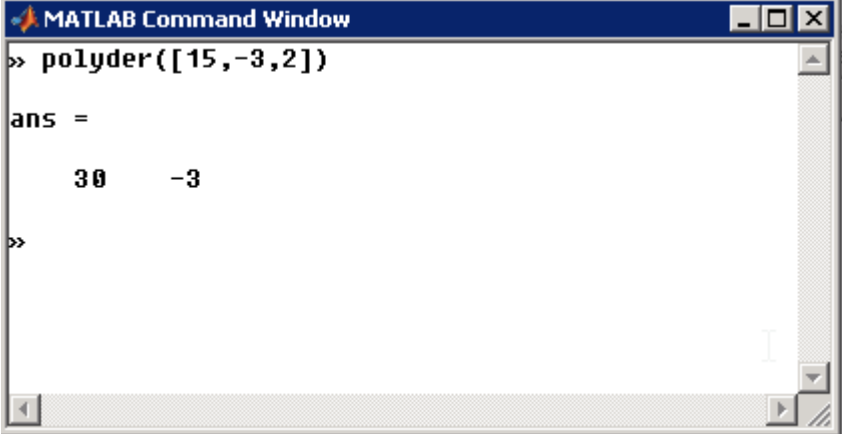
```

Derivation

Derivation is with MATLAB mainly used in mechanics or physics and only sometimes in finance. We will see now some typical function to calculate derivate of explicit expressions or vector measures.

Pseudo-symbolic derivation of polynoms

The most easiest integration that can be done in MATLAB is certainly using the function **polyder()** to derivate a simple polynom:

A screenshot of the MATLAB Command Window. The title bar reads "MATLAB Command Window". The command prompt shows the command `>> polyder([15,-3,2])`. The output is `ans =` followed by a row vector `30 -3`. The window has standard MATLAB window controls (minimize, maximize, close) in the top right corner and scrollbars on the right and bottom.

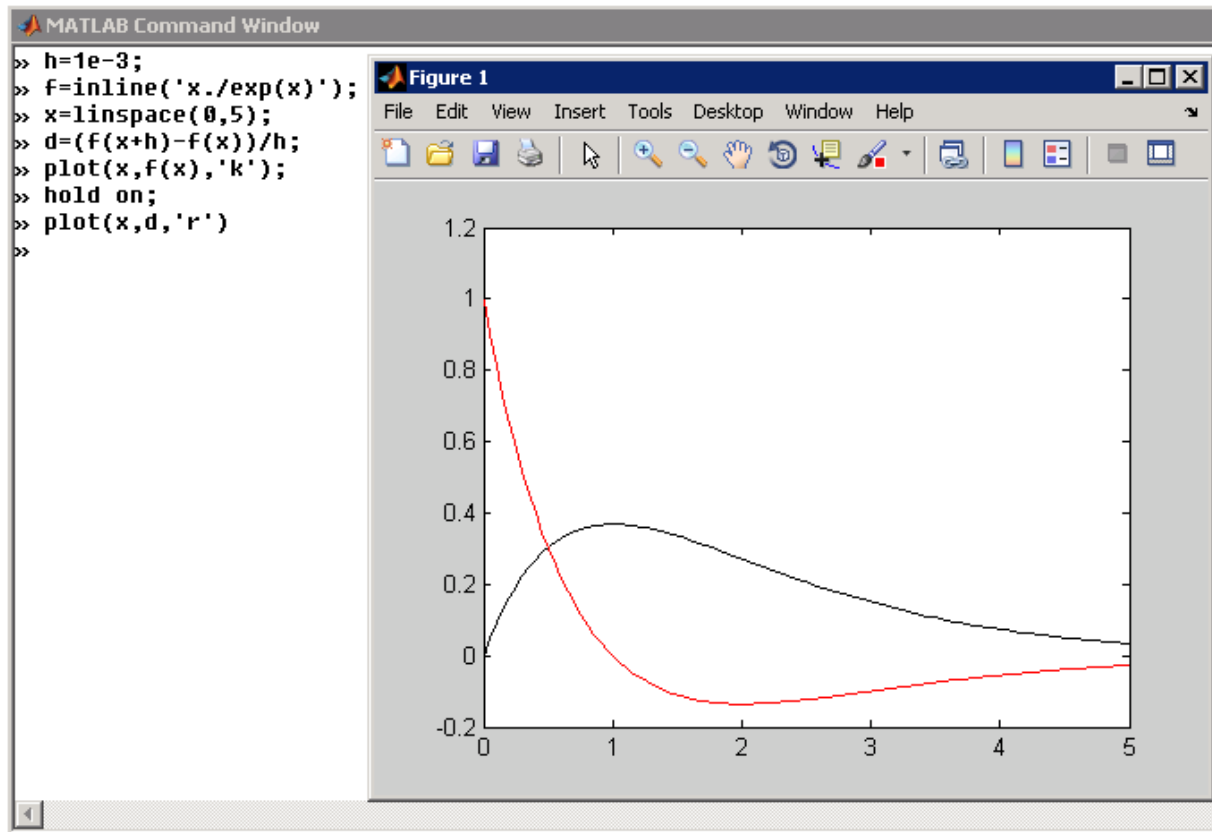
```
>> polyder([15,-3,2])  
  
ans =  
    30    -3  
  
>>
```

Sometimes I think also that this function is really a joke...

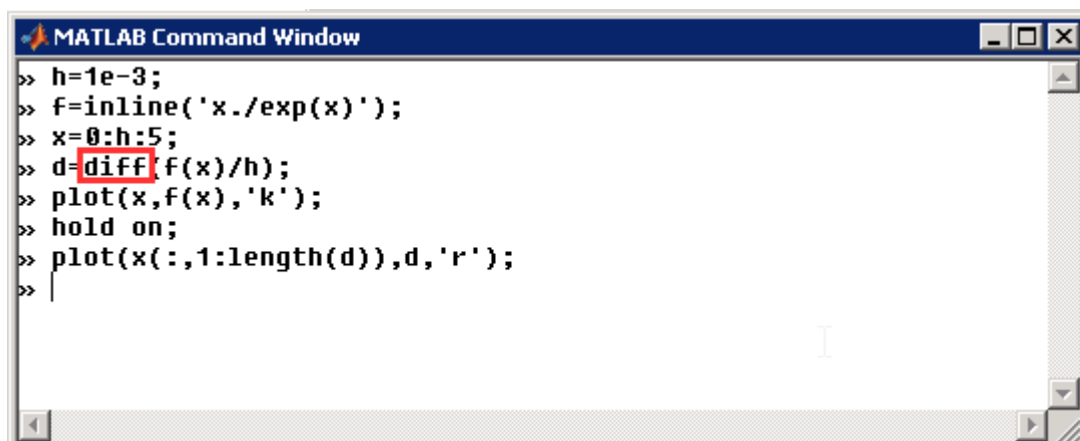
Numeric derivation

There a different way to do a numeric derivation in MATLAB (like in many others softwares) and it is always interesting to study this subject of derivation that has a lot of subtilities.

First begin with a simple right derivate with a large accuracy of $1e-3$ and using no special functions:

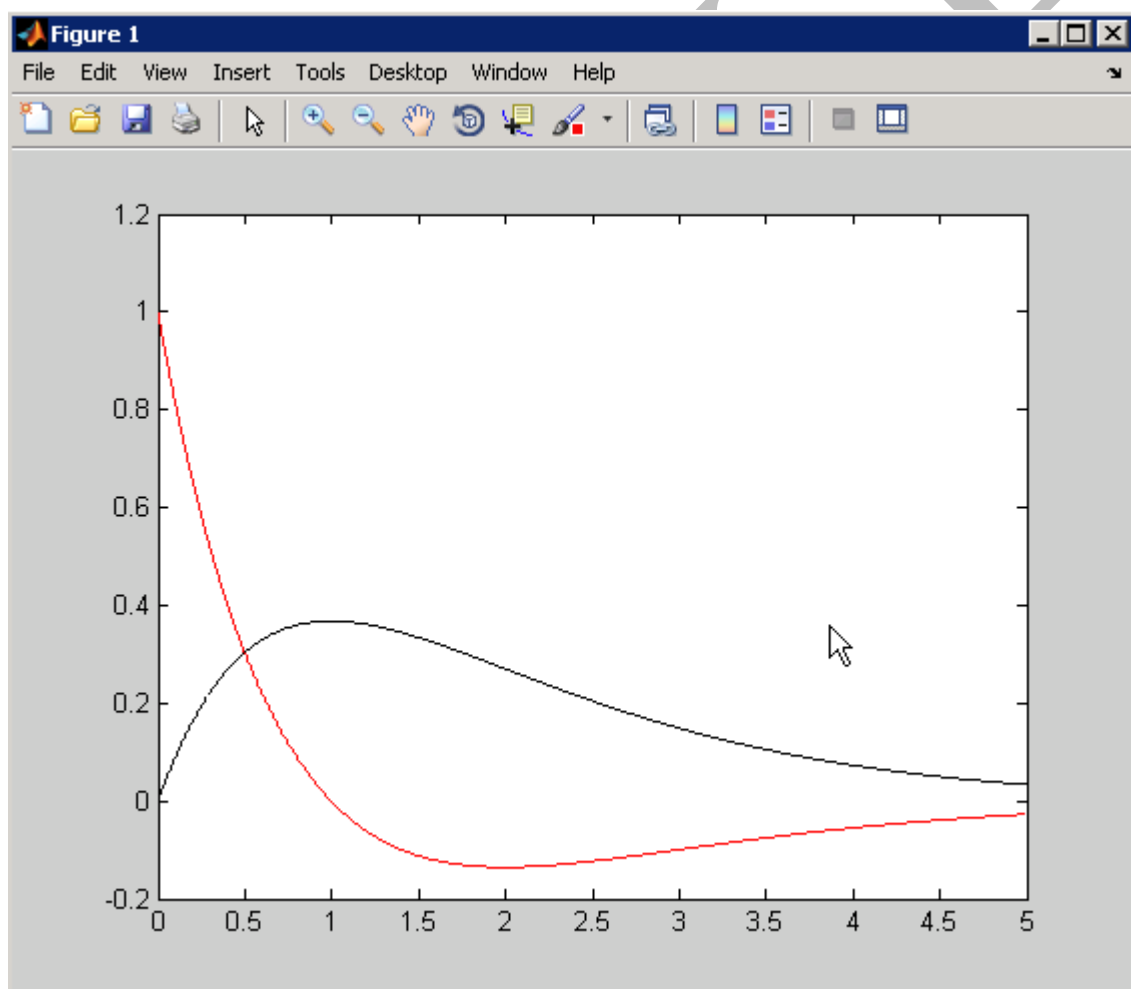


Now see the "official" method for derivate using the function **diff()**...:

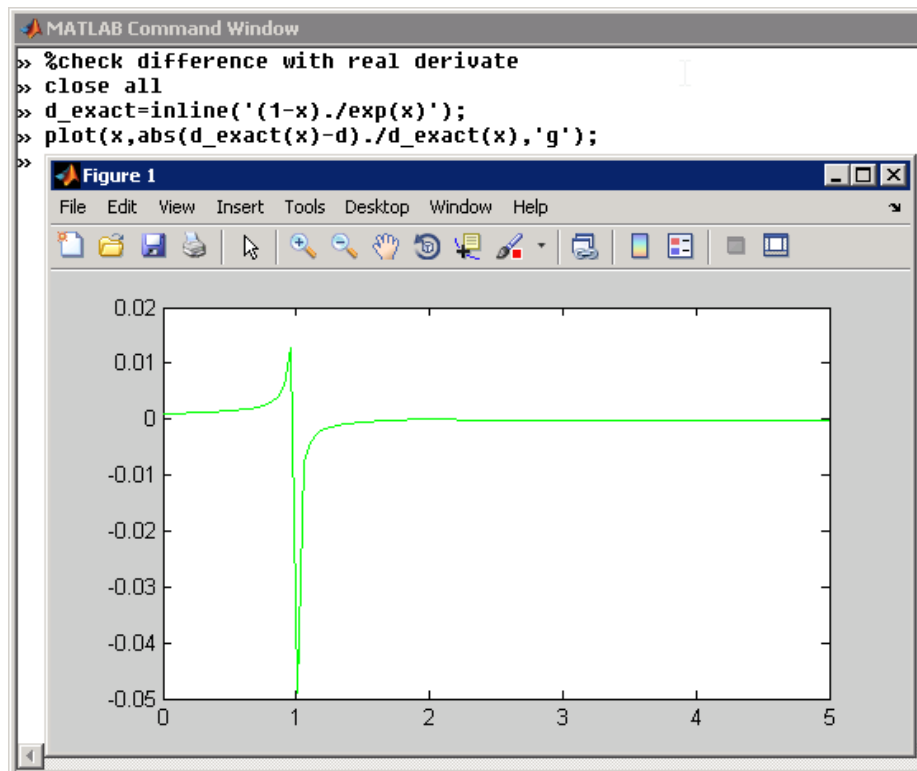


```
>> h=1e-3;  
>> f=inline('x./exp(x)');  
>> x=0:h:5;  
>> d=diff(f(x)/h);  
>> plot(x,f(x),'k');  
>> hold on;  
>> plot(x(:,1:length(d)),d,'r');  
>>
```

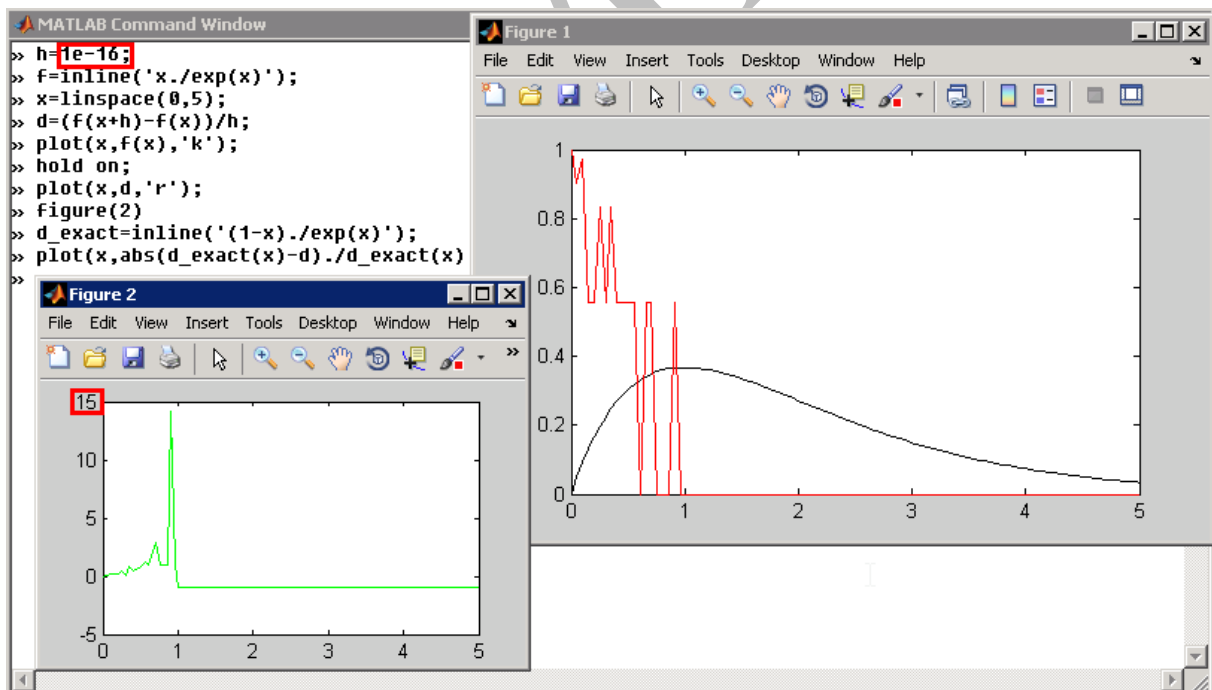
That gives:



Now see the difference between the reald derivate and the one calculate numerically:



Take care! Due to rounding error and truncatanation this method is not accurate for very small values of h (of the order of eps) for any function! (the smallest is not always the better...) as you can see below:



Differential equations (ODE/PDE)

Like everything else in the e-book, all the examples here are taken from my e-book "Éléments de Mathématiques Appliquées" where you can find the detailed proofs of the different ODE/PDE solutions.

Before to see the example you must know that it seems to be an obligation to have the ODE/PDE in an external *.m file.

Pendulum (ODE)

Let us begin with the pendulum ODE. Because the approach in MATLAB is far different for the notation of pure mathematics we will write the equations to make you understand the things better (we hope so...).

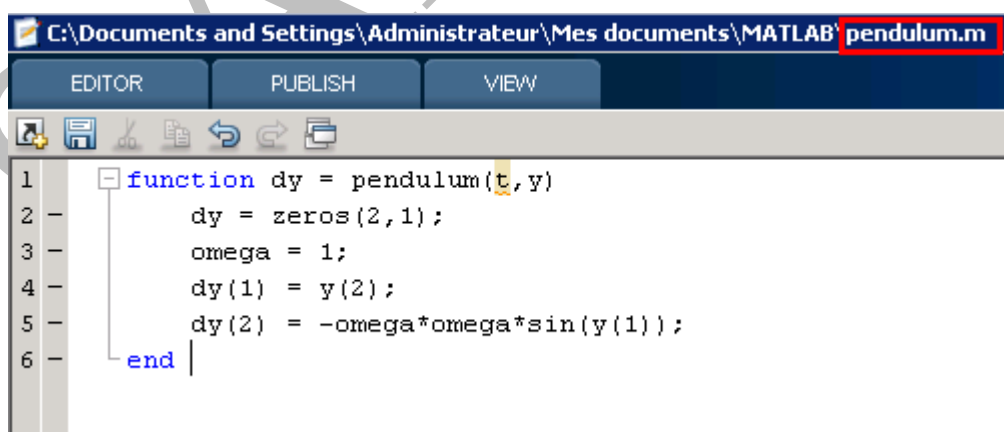
Remember that for the simple pendulum we get in the Mechanic theoretic course for small angles the following 2nd order non linear ODE (the sign in front of the sinus doesn't really matter):

$$\frac{d^2\alpha}{dt} = -\omega_0^2 \sin \alpha \Leftrightarrow \begin{matrix} x=\alpha \\ y=\frac{dx}{dt} \end{matrix} \frac{d}{dt} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} y \\ -\omega_0^2 \sin x \end{bmatrix}$$

with the initial condition that x and y are at the beginning both equal to zero. In MATLAB we need to rewrite this equation into two first order equations.

MATLAB ODE function will always, as far as I know, always return the components of the **red** vector.

In MATLAB we have to write the ODE system in a specific file and define the initial values of the derivate:



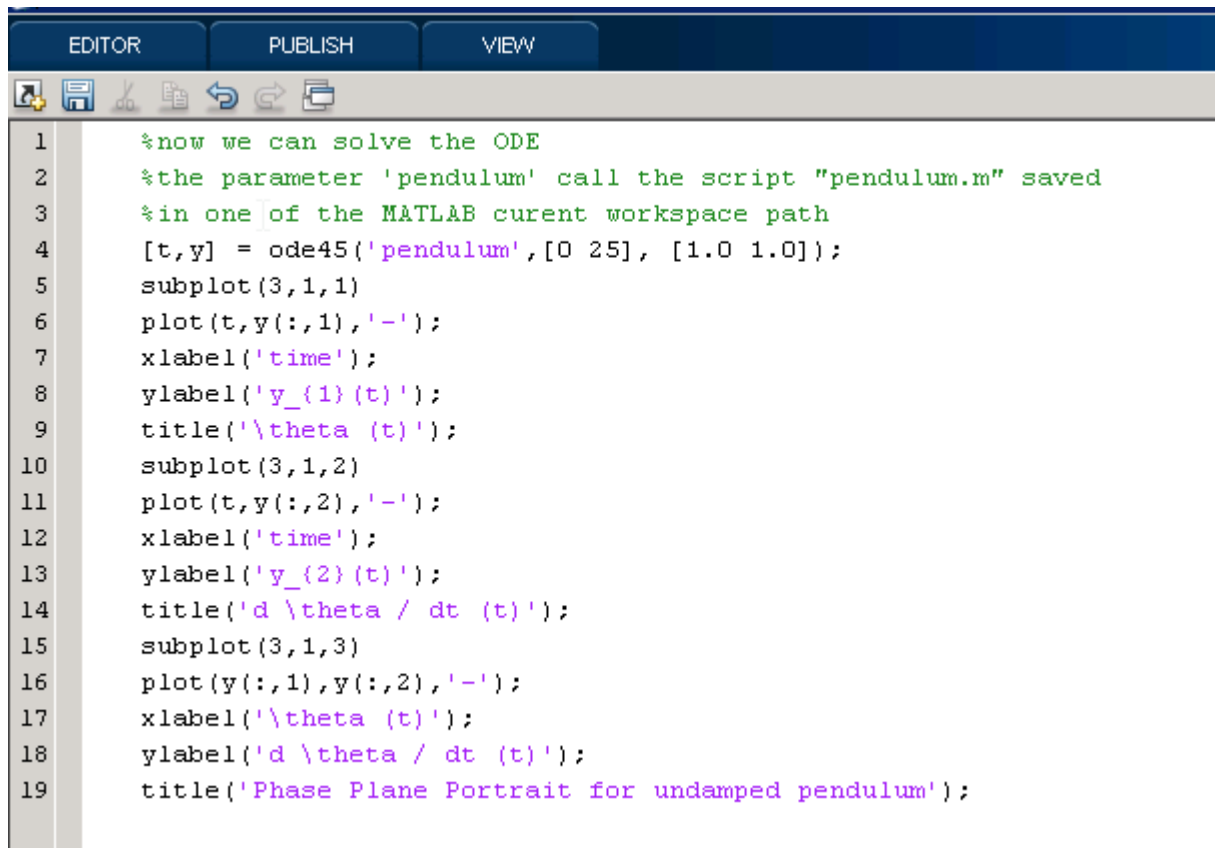
The screenshot shows a MATLAB editor window with the title bar "C:\Documents and Settings\Administrateur\Mes documents\MATLAB\pendulum.m". The window has tabs for "EDITOR", "PUBLISH", and "VIEW". The code in the editor is as follows:

```

1 function dy = pendulum(t,y)
2     dy = zeros(2,1);
3     omega = 1;
4     dy(1) = y(2);
5     dy(2) = -omega*omega*sin(y(1));
6 end

```

And after only we can solve it between times $[0, 25]$ and initial position and pulsation using the **ode45()** solver without any special parameters:



```

1      %now we can solve the ODE
2      %the parameter 'pendulum' call the script "pendulum.m" saved
3      %in one of the MATLAB curent workspace path
4      [t,y] = ode45('pendulum',[0 25], [1.0 1.0]);
5      subplot(3,1,1)
6      plot(t,y(:,1),'-');
7      xlabel('time');
8      ylabel('y_{1}(t)');
9      title('\theta (t)');
10     subplot(3,1,2)
11     plot(t,y(:,2),'-');
12     xlabel('time');
13     ylabel('y_{2}(t)');
14     title('d \theta / dt (t)');
15     subplot(3,1,3)
16     plot(y(:,1),y(:,2),'-');
17     xlabel('\theta (t)');
18     ylabel('d \theta / dt (t)');
19     title('Phase Plane Portrait for undamped pendulum');

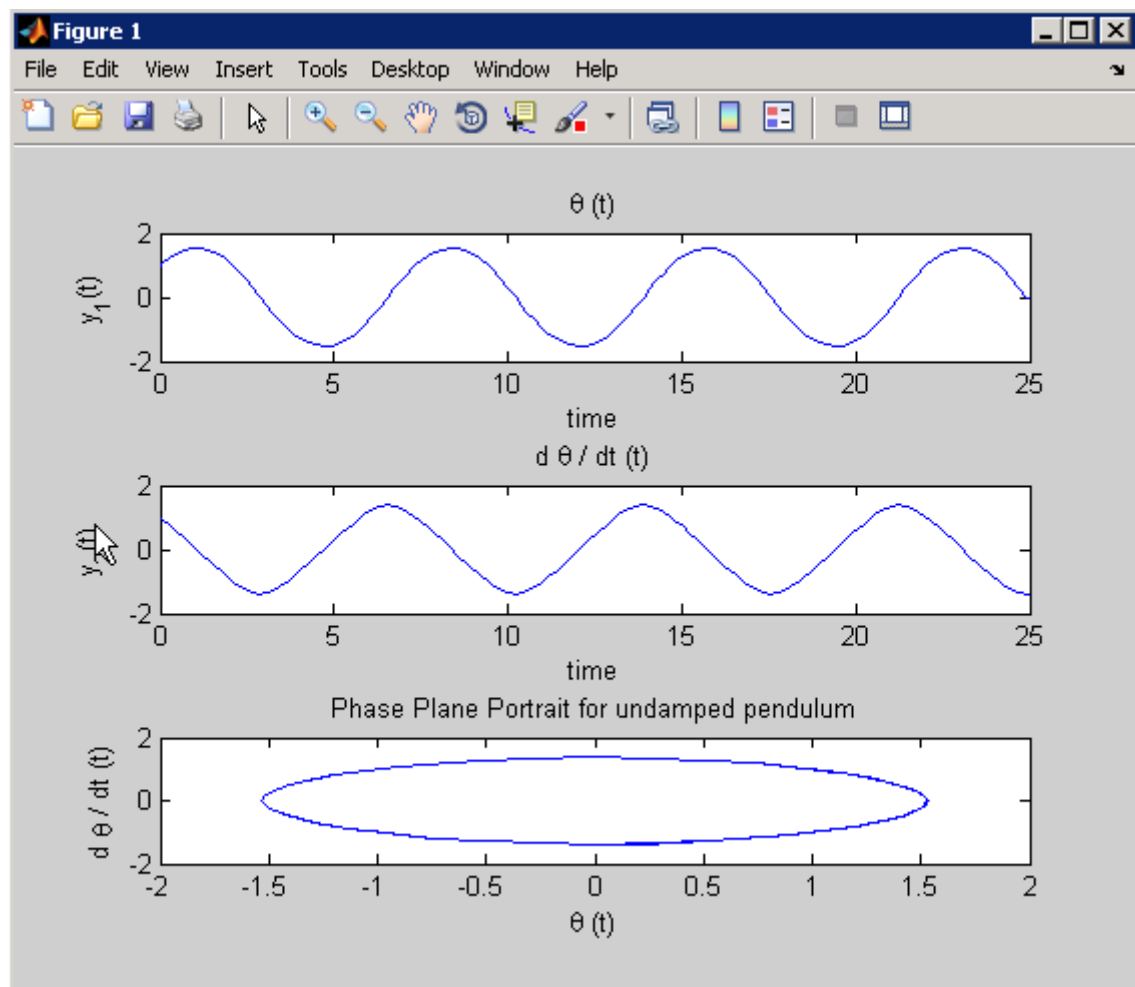
```

Here is a resume of different MATLAB ODE solver:

Solver	Problem Type	Order of Accuracy	When to Use
ode45	Nonstiff	Medium	Most of the time. This should be the first solver you try.
ode23	Nonstiff	Low	For problems with crude error tolerances or for solving moderately stiff problems.
ode113	Nonstiff	Low to high	For problems with stringent error tolerances or for solving computationally intensive problems.
ode15s	Stiff	Low to medium	If ode45 is slow because the problem is stiff.
ode23s	Stiff	Low	If using crude error tolerances to solve stiff systems and the mass matrix is constant.
ode23t	Moderately Stiff	Low	For moderately stiff problems if you need a solution without numerical damping.
ode23tb	Stiff	Low	If using crude error tolerances to solve stiff systems.

Tableau 3 ODE solvers

The result is visible on next page:



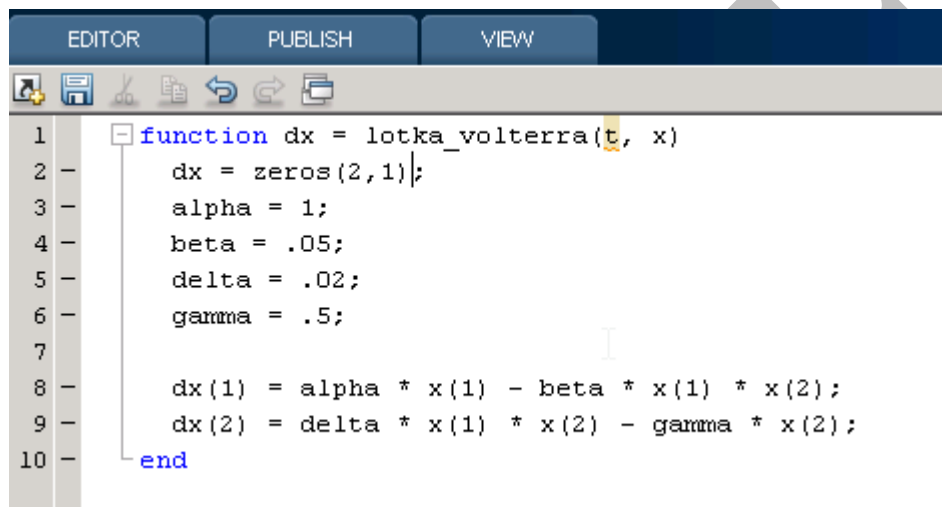
Lotka-Volterra (system of ODE)

Various assumptions during our study of Population Dynamics took us to the following system of ODE:

$$\frac{dN}{dt} = rN - cNP$$

$$\frac{dP}{dt} = -mP + bNP$$

This will be rewritten in MATLAB into the following form for some given values of the constants of the system:

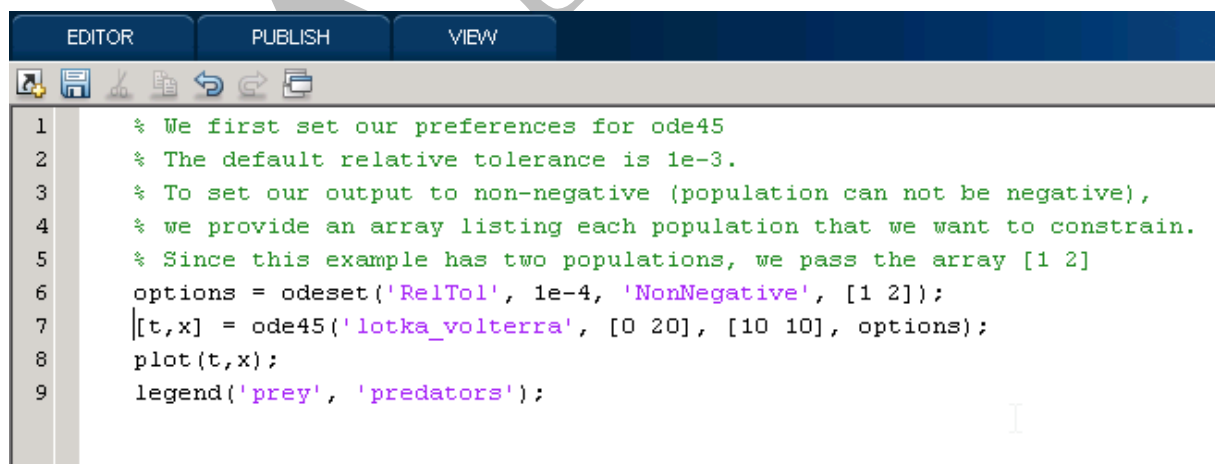


```

1  function dx = lotka_volterra(t, x)
2  -     dx = zeros(2,1);
3  -     alpha = 1;
4  -     beta = .05;
5  -     delta = .02;
6  -     gamma = .5;
7
8  -     dx(1) = alpha * x(1) - beta * x(1) * x(2);
9  -     dx(2) = delta * x(1) * x(2) - gamma * x(2);
10 - end

```

Now solving this using once again **ode45()** solver but also the function **odeset()** to define the Relative Error Tolerance and to say that both variable are not negatives:

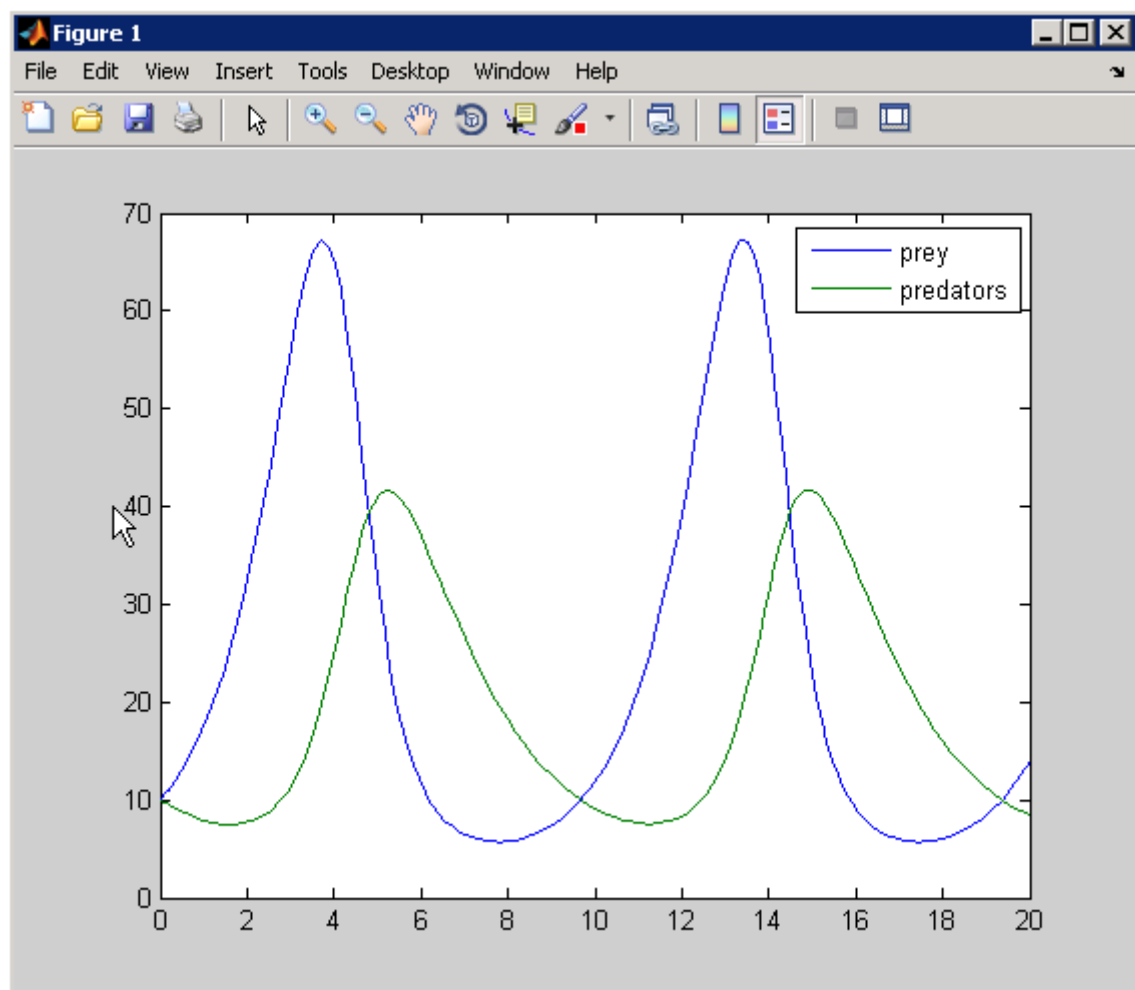


```

1  % We first set our preferences for ode45
2  % The default relative tolerance is 1e-3.
3  % To set our output to non-negative (population can not be negative),
4  % we provide an array listing each population that we want to constrain.
5  % Since this example has two populations, we pass the array [1 2]
6  options = odeset('RelTol', 1e-4, 'NonNegative', [1 2]);
7  [t,x] = ode45('lotka_volterra', [0 20], [10 10], options);
8  plot(t,x);
9  legend('prey', 'predators');

```

So we get:



Lorenz attractor (system of ODE)

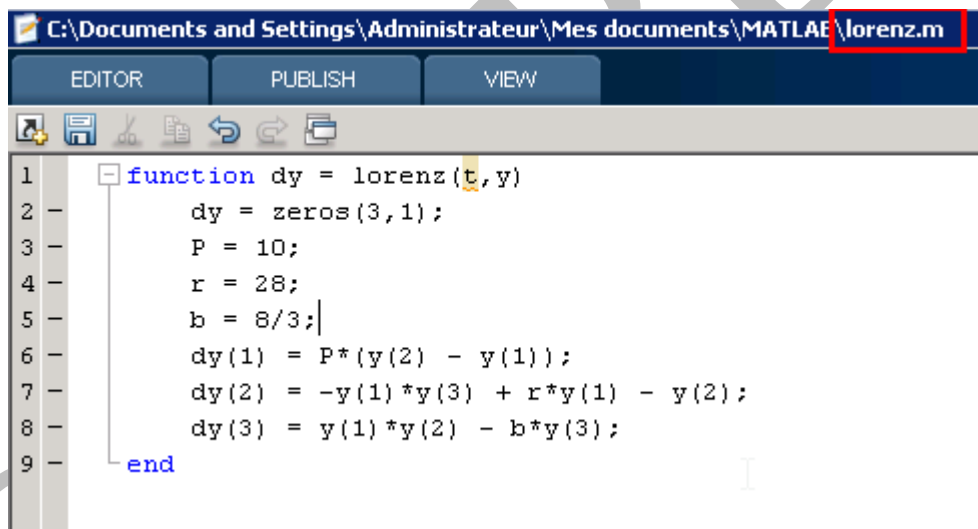
After many developments we get in the theoretical course of Meteorology Engineering the following system of ODE for the convection of a fluid:

$$\begin{aligned}\frac{dv}{dt} &= -\Gamma v + \varepsilon g \frac{T_3}{2} \\ \frac{dT_3}{dt} &= -KT_3 + \frac{vT_1}{r} - \frac{vT_4}{r} \\ \frac{dT_4}{dt} &\cong -KT_4 + \frac{vT_3}{r}\end{aligned}$$

And we get after some simplifications:

$$\begin{aligned}X' &= \text{Pr}(Y - X) \\ Y' &= rX - Y - XZ \\ Z' &= -Z + YX\end{aligned}$$

We will write this in MATLAB into the following way:



The screenshot shows a MATLAB editor window with the following code:

```

C:\Documents and Settings\Administrateur\Mes documents\MATLAB\lorenz.m
EDITOR PUBLISH VIEW
1 function dy = lorenz(t,y)
2     dy = zeros(3,1);
3     P = 10;
4     r = 28;
5     b = 8/3;
6     dy(1) = P*(y(2) - y(1));
7     dy(2) = -y(1)*y(3) + r*y(1) - y(2);
8     dy(3) = y(1)*y(2) - b*y(3);
9 end
  
```

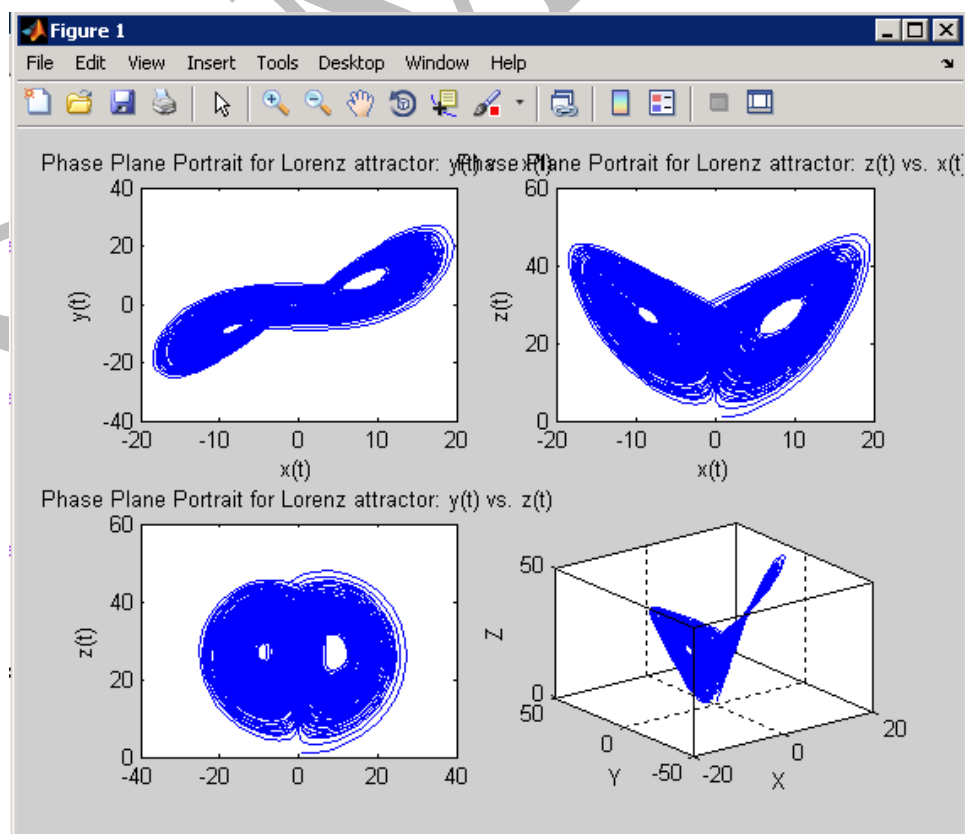
And after we solve it again using `ode45()`:

```

1      [t,y] = ode45('lorenz',[0 250],[1.0 1.0 1.0]);
2      subplot(221)
3      plot(y(:,1),y(:,2),'-');
4      xlabel('x(t)');
5      ylabel('y(t)');
6      title('Phase Plane Portrait for Lorenz attractor: y(t) vs. x(t)');
7      subplot(222)
8      plot(y(:,1),y(:,3),'-');
9      xlabel('x(t)');
10     ylabel('z(t)');
11     title('Phase Plane Portrait for Lorenz attractor: z(t) vs. x(t)');
12     subplot(223)
13     plot(y(:,2),y(:,3),'-');
14     ylabel('y(t)');
15     ylabel('z(t)');
16     title('Phase Plane Portrait for Lorenz attractor: y(t) vs. z(t)');
17     subplot(224)
18     plot3(y(:,1),y(:,2),y(:,3))
19     grid on;box on;
20     xlabel('X');ylabel('Y');zlabel('Z');

```

We get the following famous plots:



2 body motion (system of ODE)

Consider 2 point masses, with masses m_1 and m_2 , described by the two-dimensional coordinate positions:

$$\begin{aligned}(x_1(t), y_1(t)) \\ (x_2(t), y_2(t))\end{aligned}$$

The gravitational force between both of these masses is given by:

$$\frac{Gm_1m_2}{r_{12}^2} = \frac{Gm_1m_2}{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

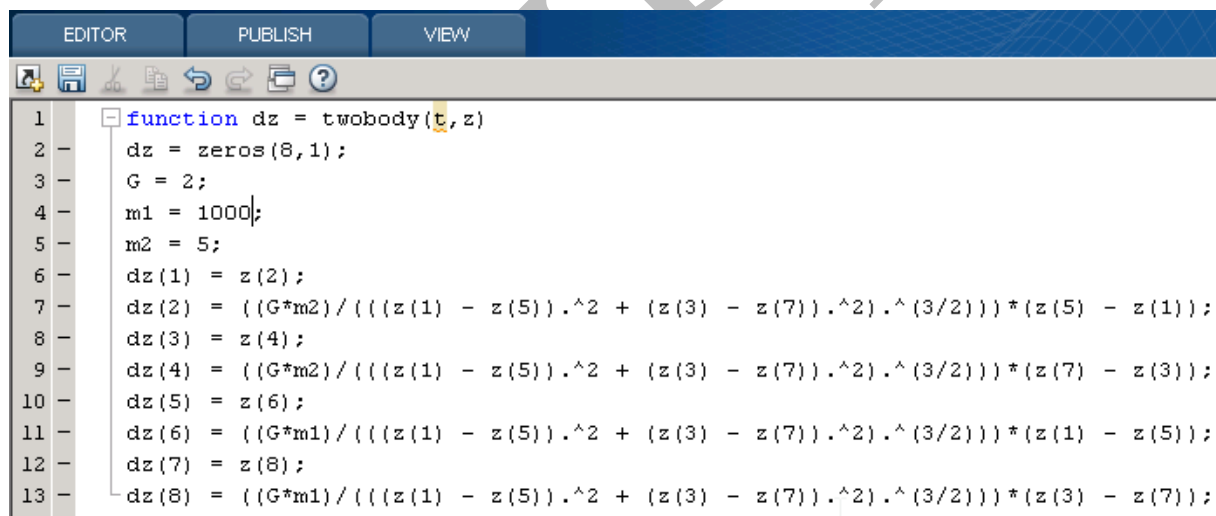
If we apply Newton's 2nd law for both masses in each coordinate direction (x and y), we get a set of 4 coupled differential equations:

$$\begin{aligned}m_1 \frac{dx_1^2}{dt^2} &= \frac{Gm_1m_2}{r_{12}^3} (x_2 - x_1) \\ m_1 \frac{dy_1^2}{dt^2} &= \frac{Gm_1m_2}{r_{12}^3} (y_2 - y_1) \\ m_2 \frac{dx_2^2}{dt^2} &= \frac{Gm_1m_2}{r_{12}^3} (x_1 - x_2) \\ m_2 \frac{dy_2^2}{dt^2} &= \frac{Gm_1m_2}{r_{12}^3} (y_1 - y_2)\end{aligned}$$

To solve these equations in MATLAB, we'll need to rewrite each of the 4 equations into two first order differential equations, thereby giving us a system of 8 differential equations to solve:

$$\begin{aligned}\frac{dx_1}{dt} &= u_1 \\ \frac{du_1}{dt} &= \frac{Gm_2}{r_{12}^3}(x_2 - x_1) \\ \frac{dy_1}{dt} &= v_1 \\ \frac{dv_1}{dt} &= \frac{Gm_2}{r_{12}^3}(y_2 - y_1) \\ \frac{dx_2}{dt} &= u_2 \\ \frac{du_2}{dt} &= \frac{Gm_1}{r_{12}^3}(x_1 - x_2) \\ \frac{dy_2}{dt} &= v_2 \\ \frac{dv_2}{dt} &= \frac{Gm_1}{r_{12}^3}(y_1 - y_2)\end{aligned}$$

In MATLAB this gives:



```

1  function dz = twobody(t,z)
2  -   dz = zeros(8,1);
3  -   G = 2;
4  -   m1 = 1000;
5  -   m2 = 5;
6  -   dz(1) = z(2);
7  -   dz(2) = ((G*m2)/(((z(1) - z(5)).^2 + (z(3) - z(7)).^2).^(3/2)))*(z(5) - z(1));
8  -   dz(3) = z(4);
9  -   dz(4) = ((G*m2)/(((z(1) - z(5)).^2 + (z(3) - z(7)).^2).^(3/2)))*(z(7) - z(3));
10 -   dz(5) = z(6);
11 -   dz(6) = ((G*m1)/(((z(1) - z(5)).^2 + (z(3) - z(7)).^2).^(3/2)))*(z(1) - z(5));
12 -   dz(7) = z(8);
13 -   dz(8) = ((G*m1)/(((z(1) - z(5)).^2 + (z(3) - z(7)).^2).^(3/2)))*(z(3) - z(7));

```

And now we solve again with `ode45()` between times `[0, 25]` and for initial values:

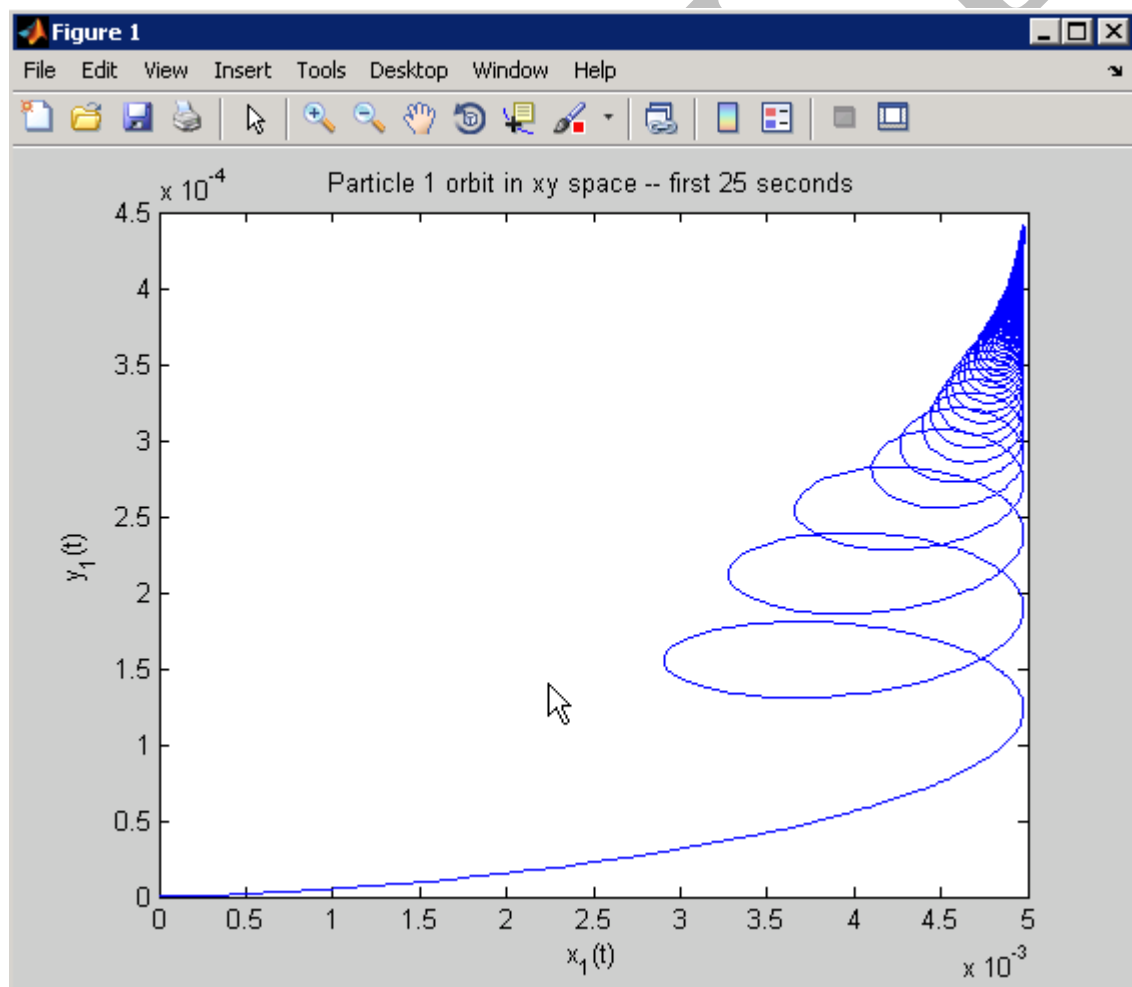
$$x_1(0), u_1(0), y_1(0), v_1(0), x_2(0), u_2(0), y_2(0), v_2(0)$$

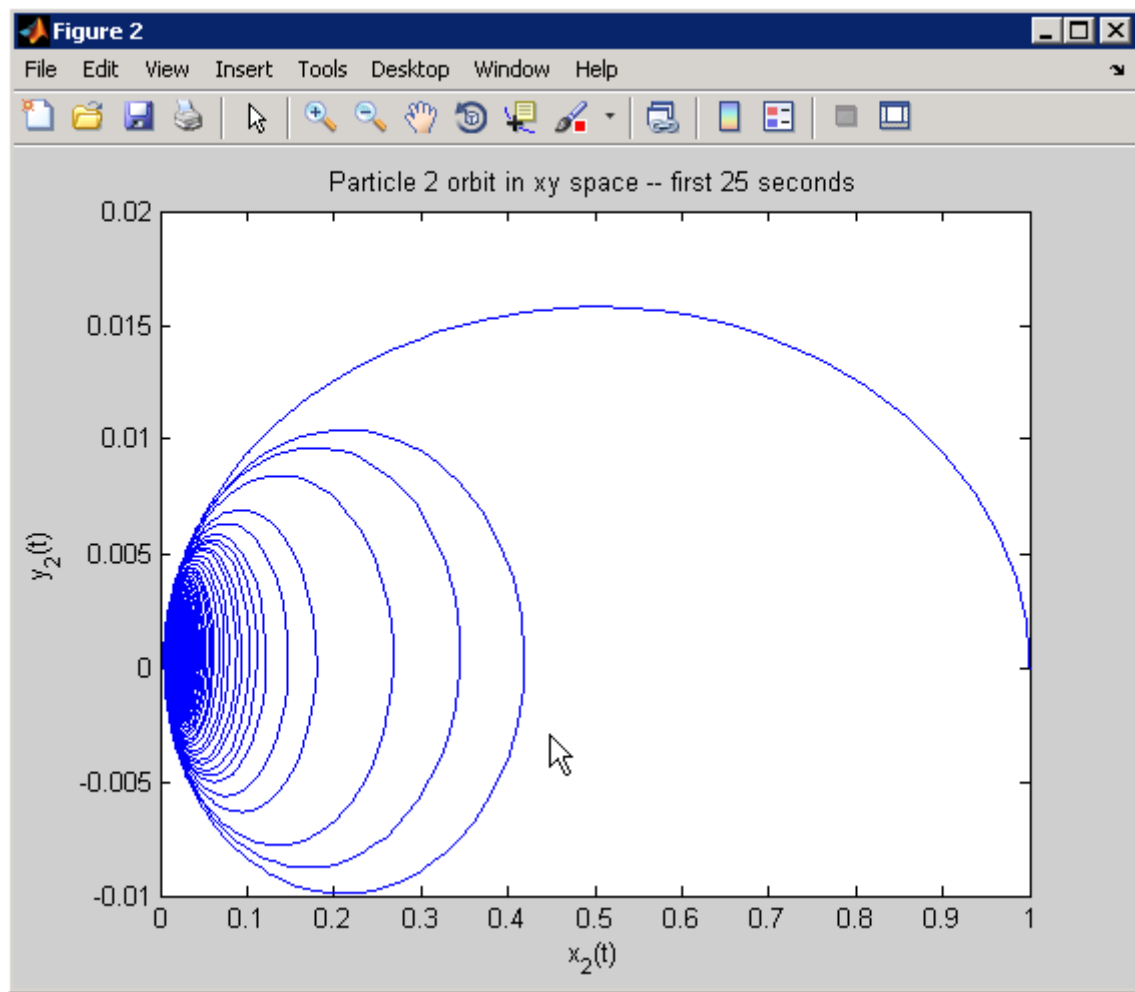
```

1 - clear all;close all;clc
2 - [t,z] = ode45('twobody',[0 25], [0 0 0 0 1 0 0 1]);
3 - plot(z(:,1),z(:,3),'-');
4 - xlabel('x_{1}(t)');
5 - ylabel('y_{1}(t)');
6 - title('Particle 1 orbit in xy space -- first 25 seconds');
7 - figure;
8 - plot(z(:,5),z(:,7),'-');
9 - xlabel('x_{2}(t)');
10 - ylabel('y_{2}(t)');
11 - title('Particle 2 orbit in xy space -- first 25 seconds');

```

And as output we get the following charts:





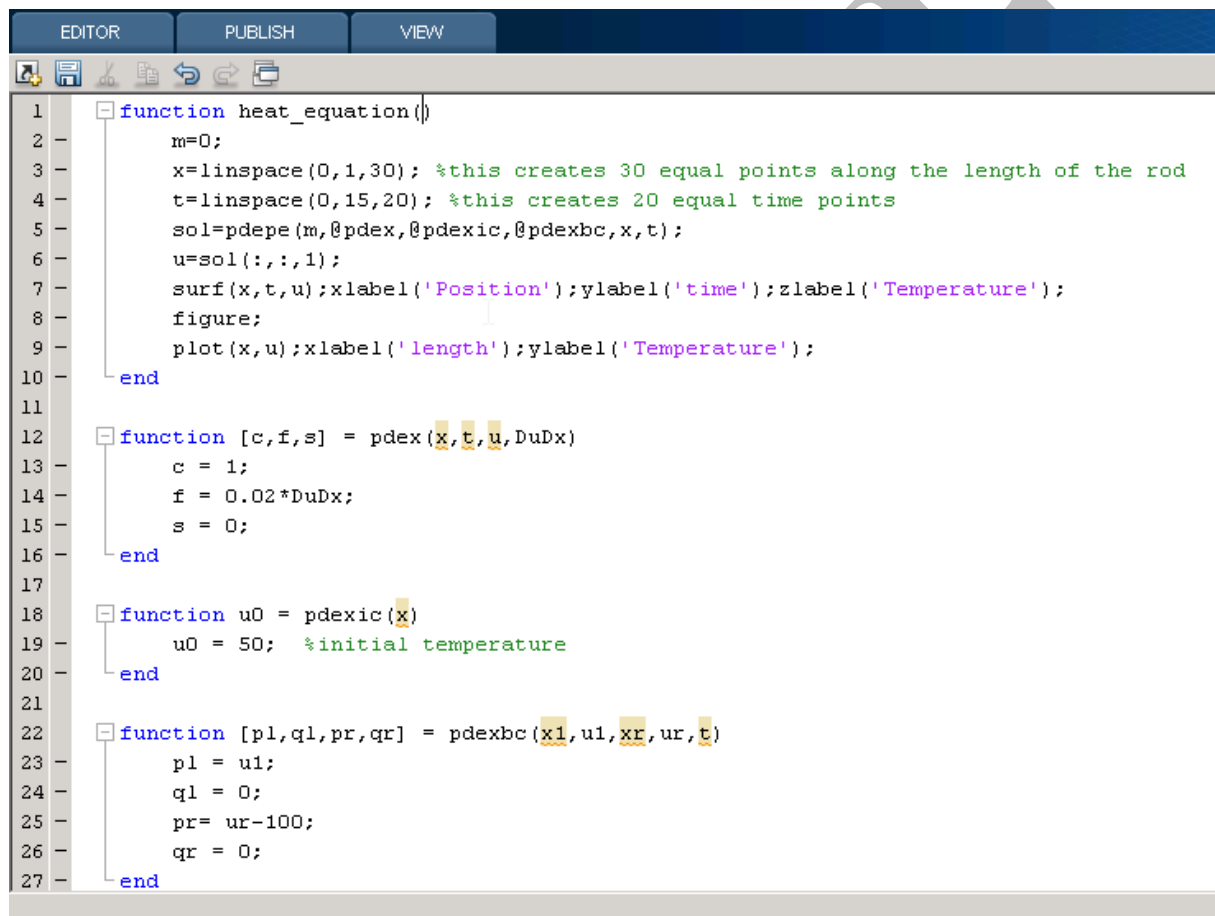
Heat equation (parabolic PDE)

Working with PDE in MATLAB is like in Maple... something horrible to write and time consuming to search the right syntax.

Let us begin with the famous heat equation that we have studied and proved in the course of Thermodynamics:

$$\frac{\partial T(x,t)}{\partial t} - \frac{\kappa}{\rho c_p} \frac{\partial^2 T(x,t)}{\partial x^2} = 0$$

using the function **pdepe()** (the explanations are given after):



```

1 function heat_equation()
2     m=0;
3     x=linspace(0,1,30); %this creates 30 equal points along the length of the rod
4     t=linspace(0,15,20); %this creates 20 equal time points
5     sol=pdepe(m,@pdex,@pdexic,@pdexbc,x,t);
6     u=sol(:, :, 1);
7     surf(x,t,u); xlabel('Position'); ylabel('time'); zlabel('Temperature');
8     figure;
9     plot(x,u); xlabel('length'); ylabel('Temperature');
10 end
11
12 function [c,f,s] = pdex(x,t,u,DuDx)
13     c = 1;
14     f = 0.02*DuDx;
15     s = 0;
16 end
17
18 function u0 = pdexic(x)
19     u0 = 50; %initial temperature
20 end
21
22 function [pl,q1,pr,qr] = pdexbc(x1,u1,xr,ur,t)
23     pl = u1;
24     q1 = 0;
25     pr = ur-100;
26     qr = 0;
27 end
  
```

For **pdepe()** to be able to interpret the PDE the equation must in MATLAB written into the general form of:

$$c\left(x,t,u,\frac{\partial u}{\partial x}\right)\frac{\partial u}{\partial t} = x^{-m} \frac{\partial}{\partial x}\left(x^m f\left(x,t,u,\frac{\partial u}{\partial x}\right)\right) + s\left(x,t,u,\frac{\partial u}{\partial x}\right)$$

where c is a diagonal matrix, f is called the "flux term", and s is called the "source term". Placing our PDE in the form as shown above, starting from (using the coefficient of the above example):

$$\frac{\partial u}{\partial t} = 0.02 \frac{\partial^2 u}{\partial x^2}$$

it becomes:

$$1 \cdot \left(\frac{\partial u}{\partial t} \right) = \left(0.02 \frac{\partial}{\partial x} \right) \frac{\partial u}{\partial x} + 0$$

Therefore:

$$c \left(x, t, u, \frac{\partial u}{\partial x} \right) = 1$$

$$f \left(x, t, u, \frac{\partial u}{\partial x} \right) = 0.02 \frac{\partial u}{\partial x}$$

$$m = 0$$

$$s \left(x, t, u, \frac{\partial u}{\partial x} \right) = 0$$

This is why we have in *pdex*:

```
function [c,f,s] = pdex(x,t,u,DuDx)
    c = 1;
    f = 0.02*DuDx;
    s = 0;
end
```

Next is **pdexic()**. This is the function that evaluates the initial conditions of the PDE. Since the initial temperature distribution for this example is:

$$u_0(x) = 50$$

the function has the following form:

```
function u0 = pdexic(x)
    u0 = 50; %initial temperature
end
```

If we were to write an initial condition function for a piecewise (triangular) function, the function would be written as follows:

```
function u0=pdexic(x)
    If x<1
        u0=x;
    Else
        u0=1-x;
    end
```

The function **pdexbc()** is the function that evaluates the boundary conditions. Again as for **pdepe()** to be able to evaluate these conditions they must be put into a form that it can be interpreted by MATLAB. The general form for the boundary conditions is:

$$p(x,t,u) + q(x,t) f\left(x,t,u, \frac{\partial u}{\partial x}\right)$$

With the boundary conditions of our example being:

$$u(0,t) = u_l = 0 \quad \text{and} \quad u(1,t) = u_r = 100$$

We need to find the values for pl and ql . Where pl and ql are the functions in the general form for **pdepe()** of the left boundary conditions:

$$u(0,t) + 0 \cdot \frac{\partial u}{\partial x}(0,t) = 0$$

This gives:

$$pl = ul$$

Where ul is the temperature function at the left boundary, and:

$$ql = 0$$

Next we find the **pdepe()** form for pr and qr , where these are the values for the right boundary condition:

$$u(1,t) + 0 \cdot \frac{\partial u}{\partial x}(1,t) = 100$$

Therefore:

$$pr = ur - 100$$

And:

$$qr = 0$$

This function is written as follows:

```
function [pl,ql,pr,qr] = pdexbc(xl,ul,xr,ur,t)
    pl = ul;
    ql = 0;
    pr = ur-100;
    qr = 0;
end
```

For an example of the Newmann conditions, where:

$$u(0,t) = 0 \quad \text{and} \quad u(1,t) = 0$$

are the boundary conditions. We find the **pdepe()** form for pl and ql :

$$u(0,t) + 1 \cdot \frac{\partial u}{\partial x}(0,t) = 0$$

Therefore, we have :

$$pl = 0, ql = 0$$

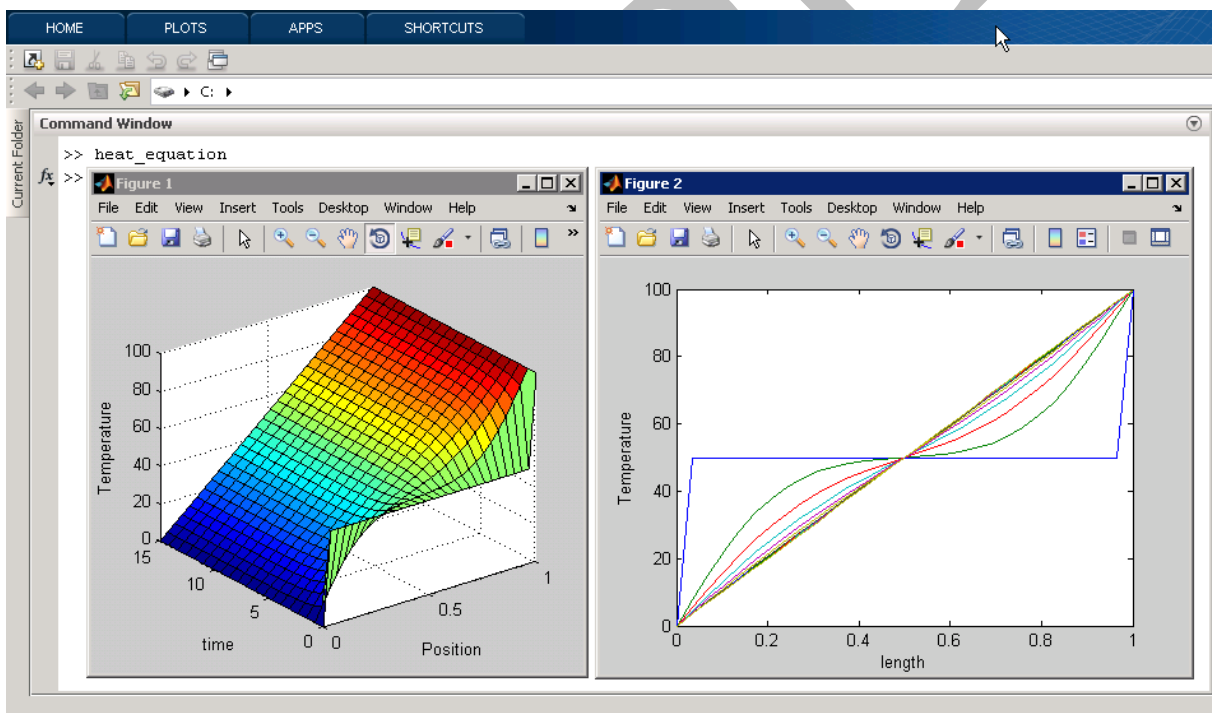
Evaluating the right boundary:

$$u(1,t) + 1 \cdot \frac{\partial u}{\partial x}(1,t) = 0$$

gives:

$$pr = 0, qr = 0$$

The result we get the function *heat_equation* is:



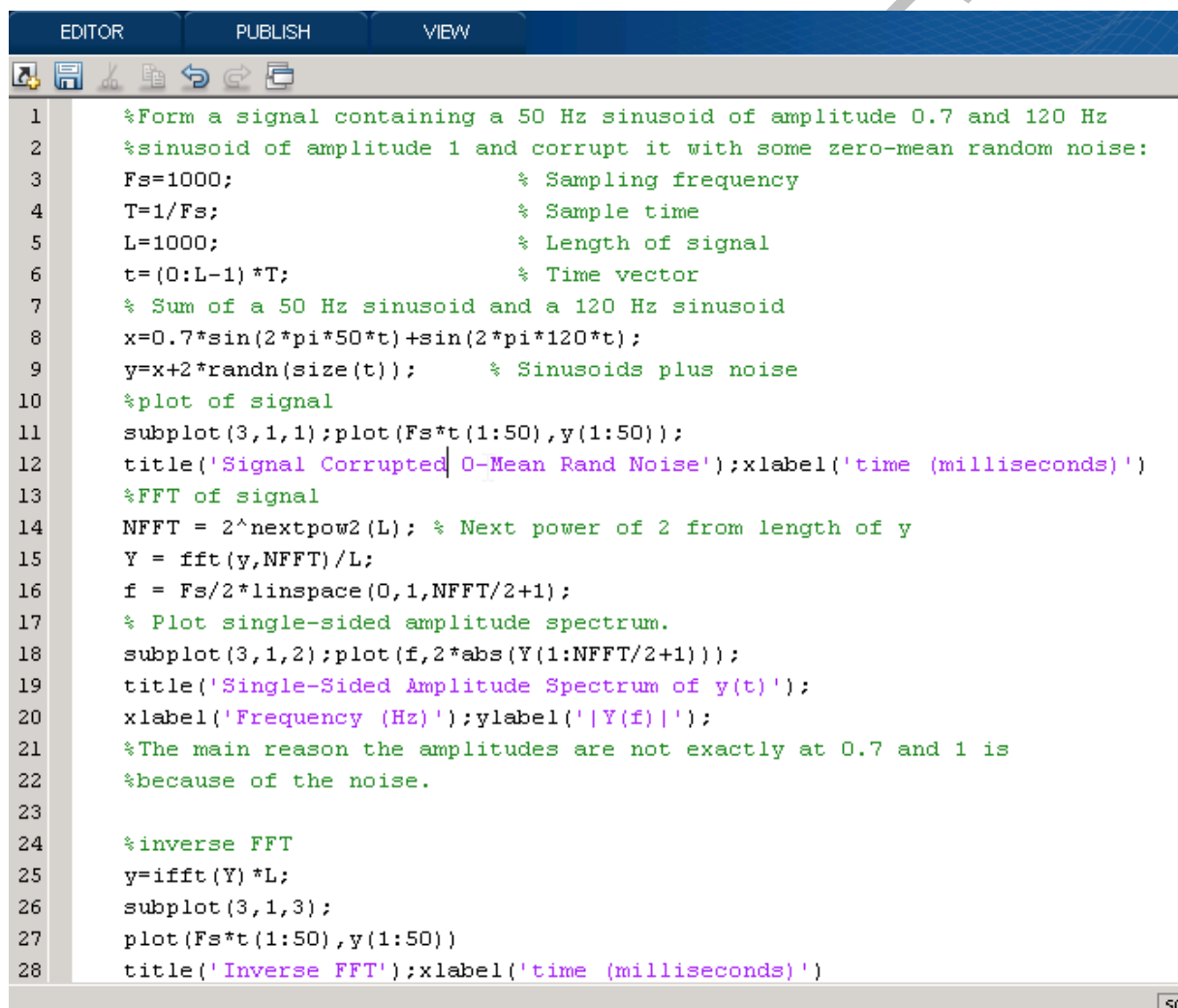
Fast Fourier Transform (DFT)

The FFT¹ is used a lot in signal processing but also in finance to identify periodicity of a time series.

The function `fft()` is not easy (intuitive) to implement to get what we saw in the theoretical course.

Note: We recommend the user to see a more real example on sound processing at page 495

Then see below how we can get the expected result:



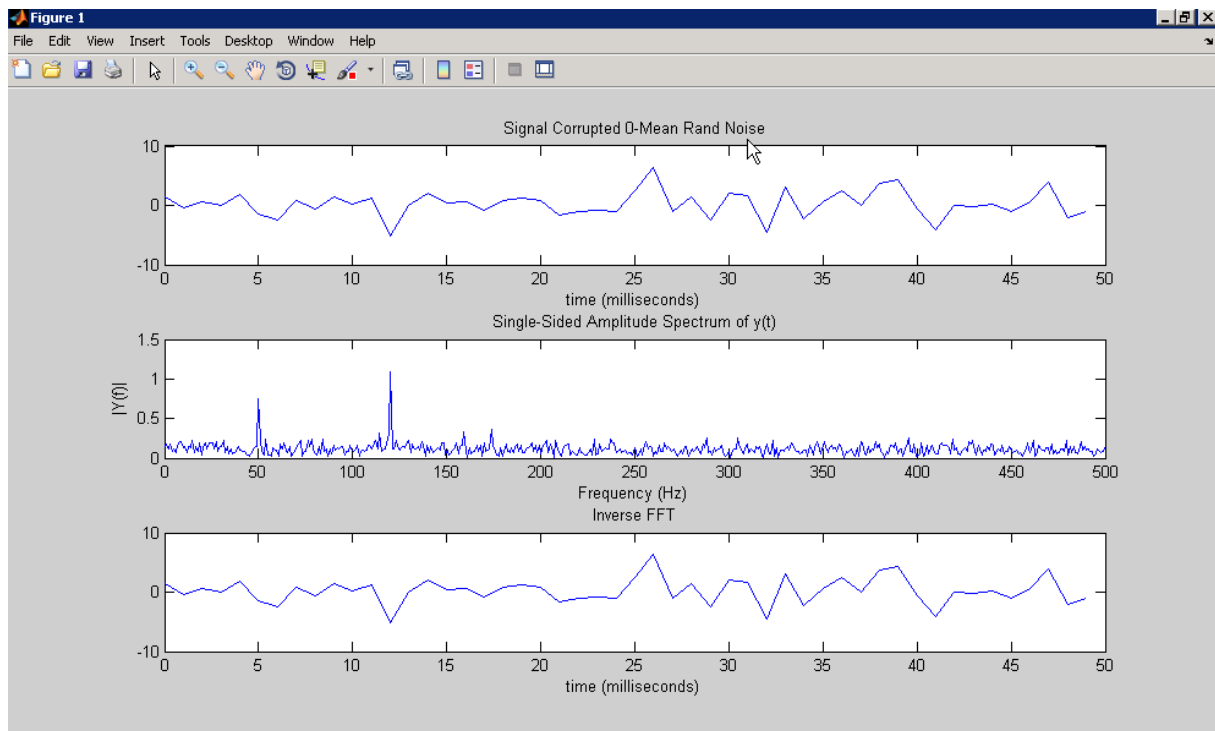
```

1 %Form a signal containing a 50 Hz sinusoid of amplitude 0.7 and 120 Hz
2 %sinusoid of amplitude 1 and corrupt it with some zero-mean random noise:
3 Fs=1000; % Sampling frequency
4 T=1/Fs; % Sample time
5 L=1000; % Length of signal
6 t=(0:L-1)*T; % Time vector
7 % Sum of a 50 Hz sinusoid and a 120 Hz sinusoid
8 x=0.7*sin(2*pi*50*t)+sin(2*pi*120*t);
9 y=x+2*randn(size(t)); % Sinusoids plus noise
10 %plot of signal
11 subplot(3,1,1);plot(Fs*t(1:50),y(1:50));
12 title('Signal Corrupted 0-Mean Rand Noise');xlabel('time (milliseconds)')
13 %FFT of signal
14 NFFT = 2^nextpow2(L); % Next power of 2 from length of y
15 Y = fft(y,NFFT)/L;
16 f = Fs/2*linspace(0,1,NFFT/2+1);
17 % Plot single-sided amplitude spectrum.
18 subplot(3,1,2);plot(f,2*abs(Y(1:NFFT/2+1)));
19 title('Single-Sided Amplitude Spectrum of y(t)');
20 xlabel('Frequency (Hz)');ylabel('|Y(f)|');
21 %The main reason the amplitudes are not exactly at 0.7 and 1 is
22 %because of the noise.
23
24 %inverse FFT
25 y=ifft(Y)*L;
26 subplot(3,1,3);
27 plot(Fs*t(1:50),y(1:50))
28 title('Inverse FFT');xlabel('time (milliseconds)')

```

In the second subplot you have the frequencies **and** the amplitudes of the original signal!

¹ FFT is the name for *any* efficient algorithm that can compute the DFT in about $\Theta(n \log n)$ time, instead of $\Theta(n^2)$ time. There are several existing FFT algorithms.



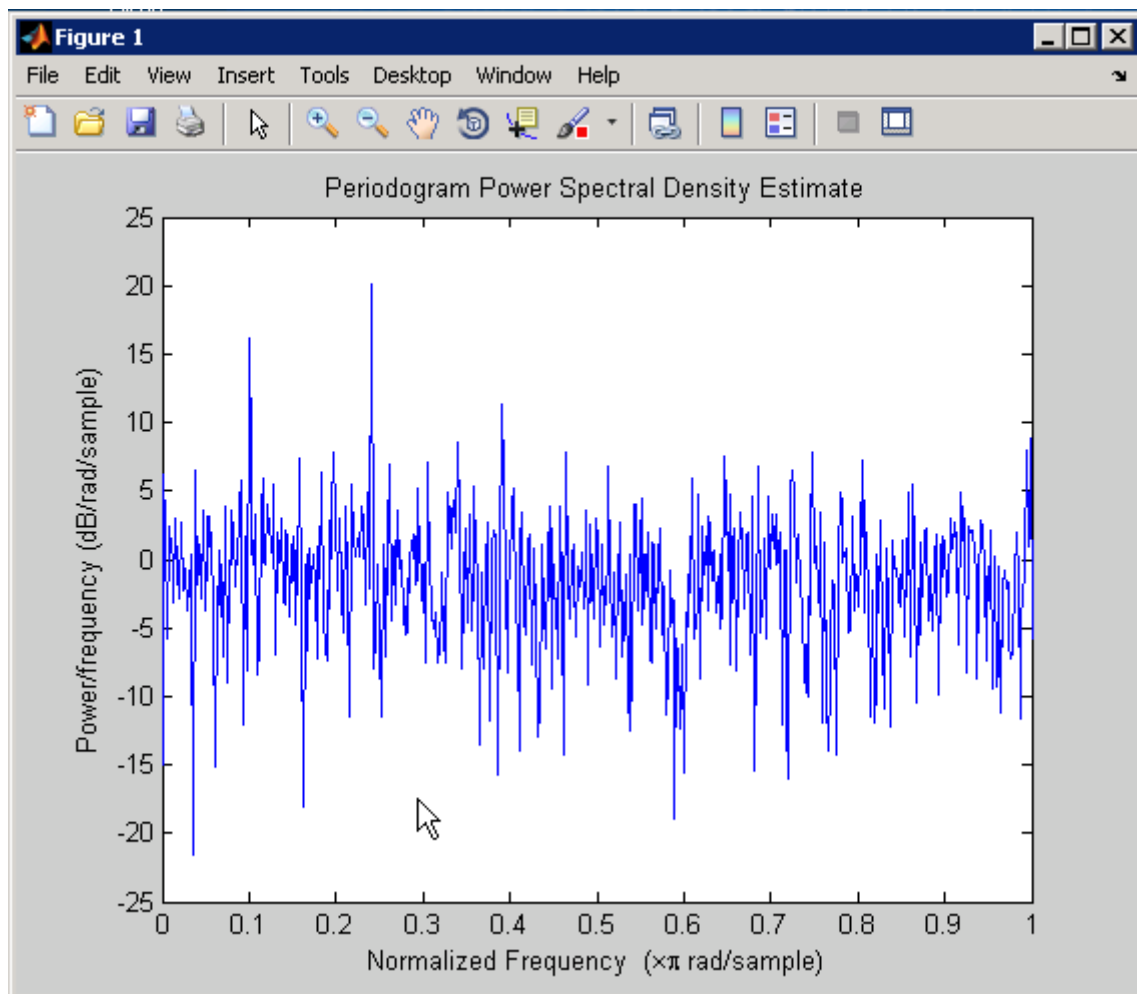
Some people prefer to use the periodogram but this command is reserved to the Signal Processing Toolbox:

```

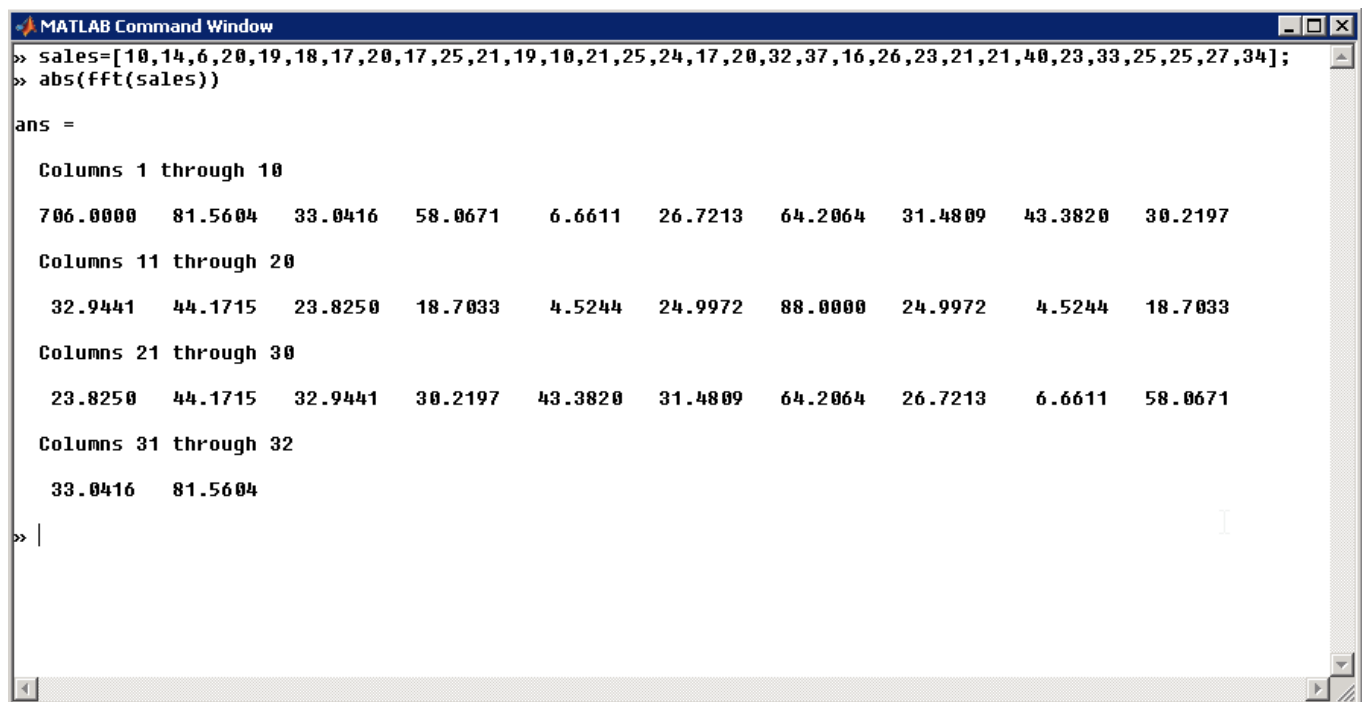
EDITOR   PUBLISH   VIEW
[Icons]
1      Fs=1000;
2      T=1/Fs;
3      L=1000;
4      t=(0:L-1)*T;
5      x=0.7*sin(2*pi*50*t)+sin(2*pi*120*t);
6      y=x+2*randn(size(t));
7
8      periodogram(y)
9      grid off;
10     %the first normalized frequency is 50/500=0.1
11     %the second normalized frequency is 120/500=0.24

```

The result is:



And now have a look if for the statistical test of periodicity that we have studied in the theoretical training we get the same coefficients (module of complex numbers) as for R and Microsoft Excel:



```

MATLAB Command Window
>> sales=[10,14,6,20,19,18,17,20,17,25,21,19,10,21,25,24,17,20,32,37,16,26,23,21,21,40,23,33,25,25,27,34];
>> abs(fft(sales))

ans =

Columns 1 through 10
706.0000    81.5604    33.0416    58.0671     6.6611    26.7213    64.2064    31.4809    43.3820    30.2197

Columns 11 through 20
32.9441    44.1715    23.8250    18.7033     4.5244    24.9972    88.0000    24.9972     4.5244    18.7033

Columns 21 through 30
23.8250    44.1715    32.9441    30.2197    43.3820    31.4809    64.2064    26.7213     6.6611    58.0671

Columns 31 through 32
33.0416    81.5604
  
```

So we get exactly the expected values!

Image processing

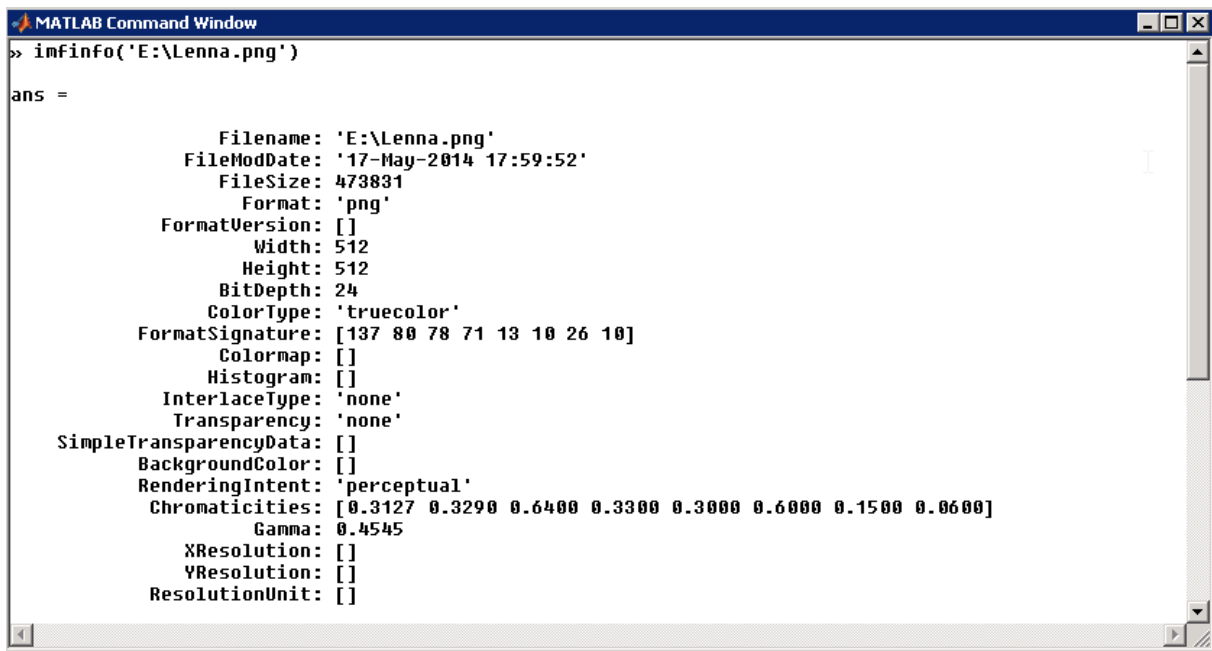
Without the Image Processing Toolbox working with images in MATLAB is possible but requires a lot of manipulations of matrices to obtain simple results. Thus, the purpose of this chapter will be to see simple operations that we can get without too much pain using native core function.

Get image properties

The basics of working with images are to retrieve information concerning the image file itself. Consider the following image:



Using the function `iminfo()` we get:

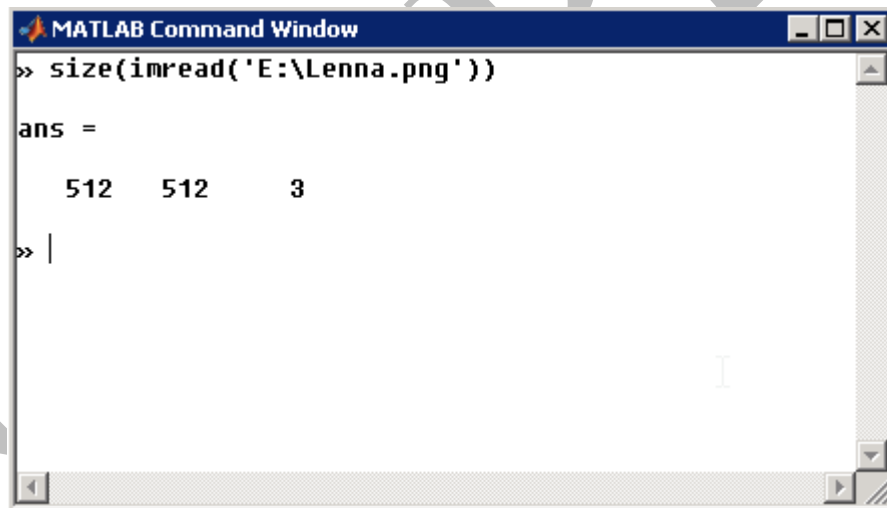


```
MATLAB Command Window
>> imfinfo('E:\Lenna.png')

ans =

    Filename: 'E:\Lenna.png'
   FileModDate: '17-May-2014 17:59:52'
    FileSize: 473831
      Format: 'png'
  FormatVersion: []
        Width: 512
        Height: 512
    BitDepth: 24
    ColorType: 'truecolor'
FormatSignature: [137 80 78 71 13 10 26 10]
    Colormap: []
   Histogram: []
InterlaceType: 'none'
  Transparency: 'none'
SimpleTransparencyData: []
BackgroundColors: []
RenderingIntent: 'perceptual'
Chromaticities: [0.3127 0.3290 0.6400 0.3300 0.3300 0.6000 0.1500 0.0600]
          Gamma: 0.4545
  XResolution: []
  YResolution: []
ResolutionUnit: []
```

The most important property for a lot of images effects and manipulations is the size of the image that we can simple get using the function **size()**:



```
MATLAB Command Window
>> size(imread('E:\Lenna.png'))

ans =

    512    512     3

>> |
```

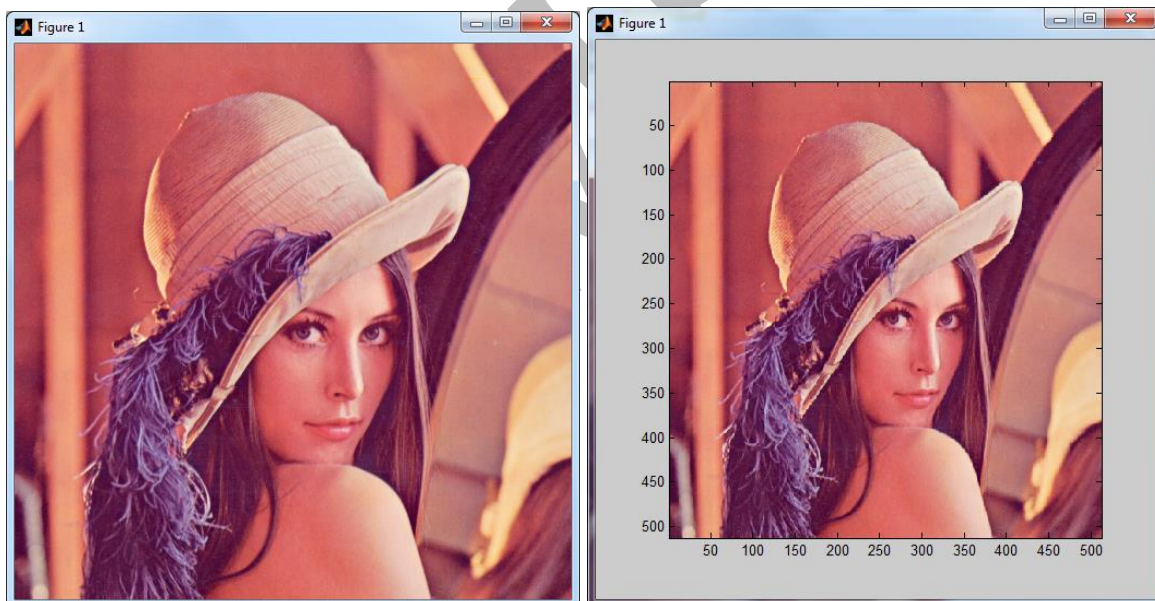
Show an image properly

Show properly and image is important for the end user. For this purpose consider the following script:

```
EDITOR PUBLISH VIEW
1  img=imread('Lenna.png');
2
3  figure;
4  imshow(img); %
5  set(gcf,'menubar','none');
6
7  figure
8  image(img);
9  set(gcf,'menubar','none');
```

Caution: `imshow()` belongs to the Image Processing Toolbox.

With this we get for the first figure (left) on the second figure (right):

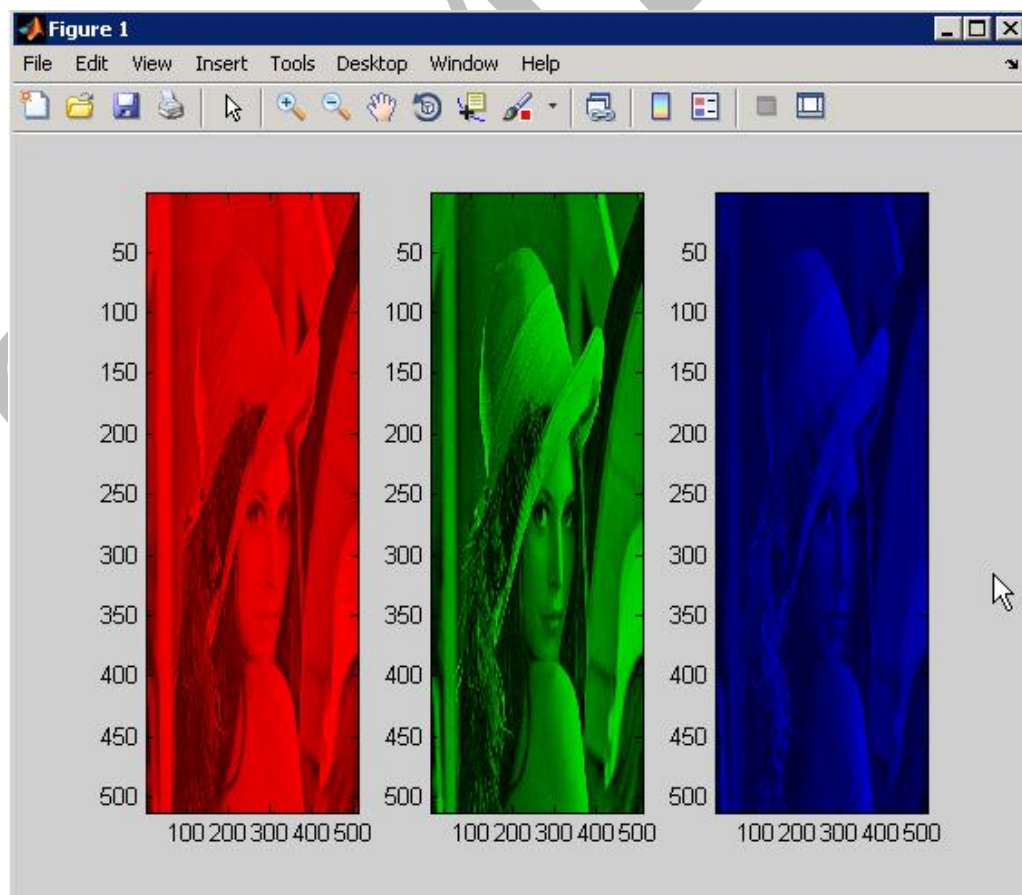


Decompose images alpha layers

This is a very good exercise to understand how image files are built on the basis of three matrices containing respectively the RGB color channels:

```
EDITOR PUBLISH VIEW
1  img=imread('E:\Lenna.png');
2  imgrouge=uint8(zeros(size(img)));
3  imgrouge(:,:,1) = img(:,:,1);
4  subplot(1,3,1);
5  image(imgrouge)
6  imgvert= uint8(zeros(size(img)));
7  imgvert(:,:,2)=img(:,:,2);
8  subplot(1,3,2);
9  image(imgvert)
10 imgbleu=uint8(zeros(size(img)));
11 imgbleu(:,:,3)=img(:,:,3);
12 subplot(1,3,3);
13 image(imgbleu);
```

The result will be:

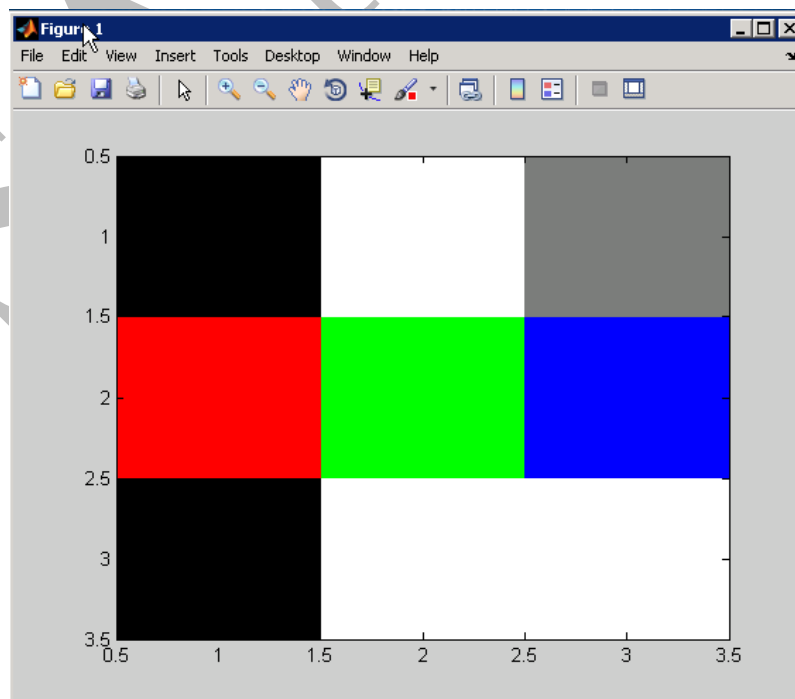


Build an image

To understand how a three layer image works a very good exercise is to build its own RGB image and also to see how MATLAB behaves in some extreme cases:

```
1      clear all;close all;clc
2
3      img=uint8(zeros(3,3,3));
4
5      %first row
6      img(1,1,:)=0; %pure black
7      img(1,2,:)=255; %pure white
8      img(1,3,:)=125; %pure gray
9
10     %second row
11     img(2,1,1)=255; %pure red
12     img(2,2,2)=255; %pure green
13     img(2,3,3)=255; %pure blue
14
15     %third row: extrem cases to see the behavior
16     img(3,1,:)=-30;
17     img(3,2,:)=270;
18     img(3,3,:)=Inf;
19
20     image(img);
```

This will give:



Plot some basic images manipulations and save image

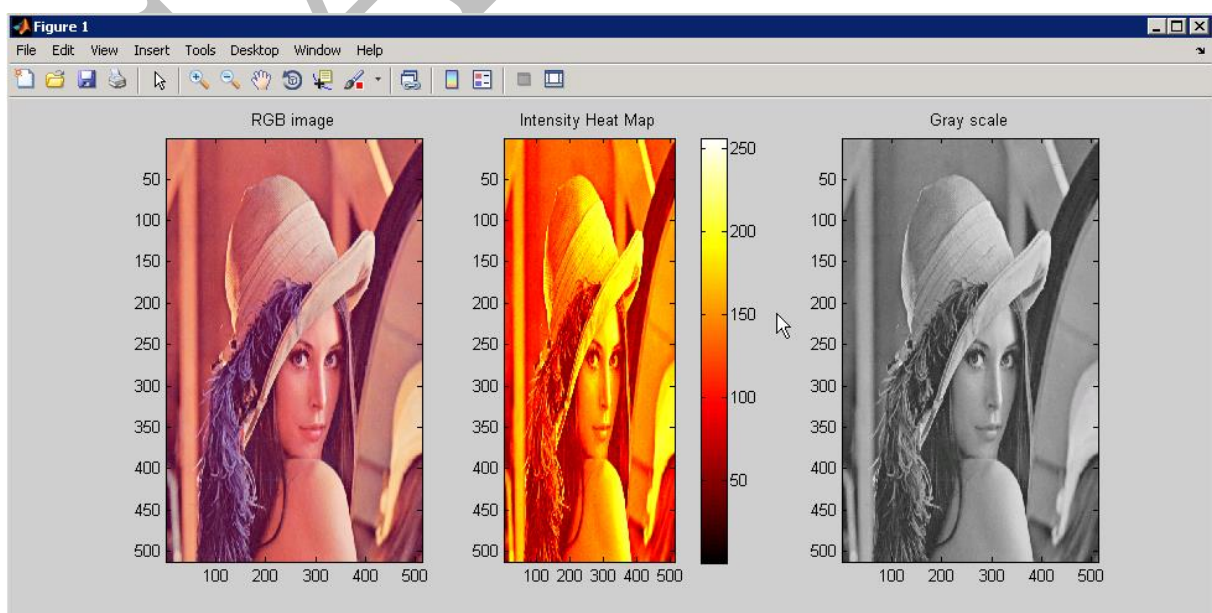
We will see here below some basics images manipulations (show only some color channel and convert to gray without using Image Processing Toolbox) of colors and plots using the function **imread()**, **image()** and finally **imwrite()** to save the transformed image on the computer:

```

1 subplot(1,3,1);
2 rgb = imread('E:\Lenna.png');
3 image(rgb) %image with full colors
4 title('RGB image')
5
6 subplot(1,3,2);
7 %image with average colors of the 3 alpha channel
8 im = mean(rgb,3);
9 image(im);
10 colorbar;
11 title('Intensity Heat Map');
12 colormap(hot(256));
13 imwrite(im,hot(256),'E:\LennaHeat.jpg','jpg');
14
15 subplot(1,3,3);
16 rgb=imread('E:\Lenna.png');
17 %image in gray scale
18 imgray=uint8(repmat(mean(rgb,3),[1,1,3]));
19 image(imgray)

```

The result of the multiple plot will be:



Filter on luminosity value of a given channel

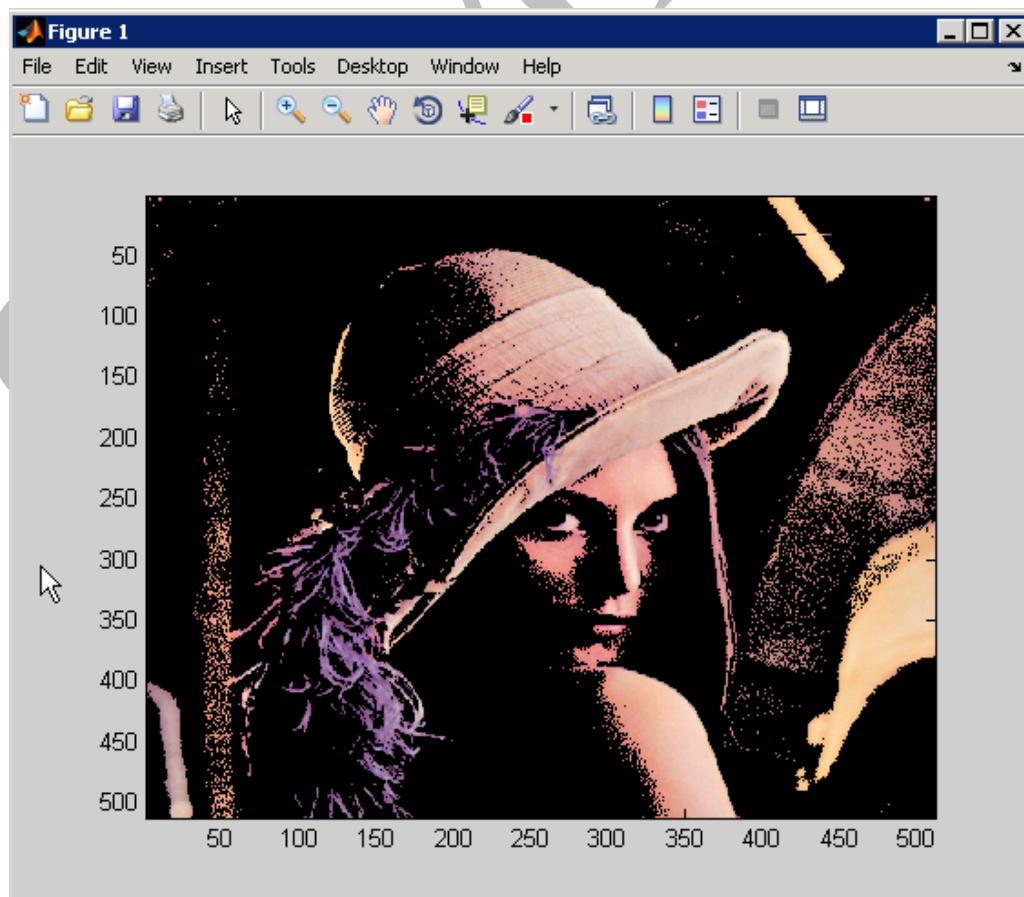
This is also a very academic useful example to understand how MATLAB works with images. The example can be understood only by reading the script:

```

1  clear all;close all;clc
2  img=imread('Lenna.png');
3
4  s=size(img);
5
6  img_post=uint8(zeros(s));
7  treshold=125;
8
9  for k=1:s(1) %for each column
10
11      for l=1:s(2) %and for each row of the column
12          if img(k,l,3)>treshold %filter the third (blue layer)
13              img_post(k,l,:)=img(k,l,:); %keep the pixel color info on the 3 layers
14          end
15      end
16  end
17
18
19  imshow(img_post)

```

The final result will be:



Draw on an image and export the result

Something very important in image pattern identification is to be able (later in this document) to recognize automatically geometries on a photo and to draw something around to put them in evidence for the user and after to save it on the computer.

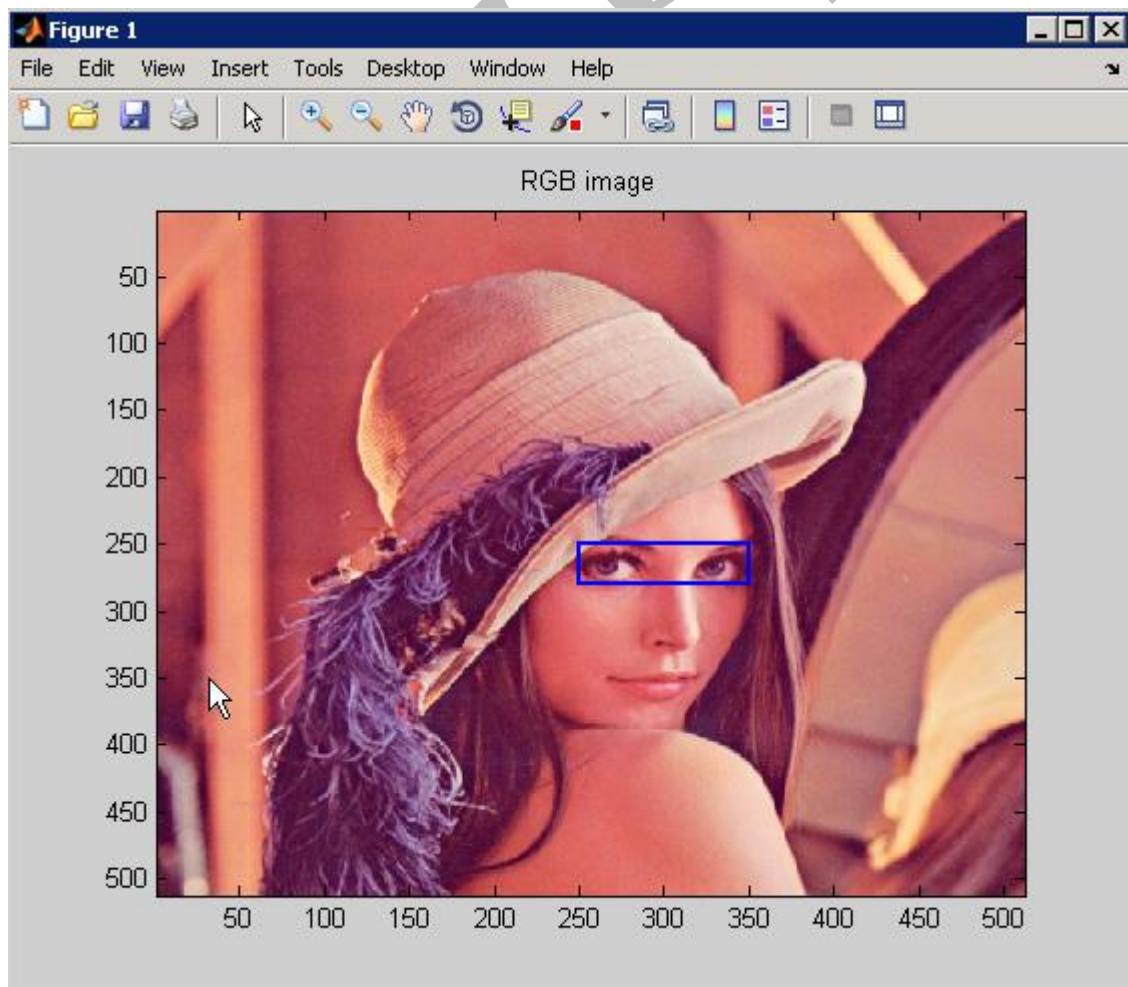
For this purpose we use the functions **rectangle()**, **getframe()** and **frame2im()**:

```

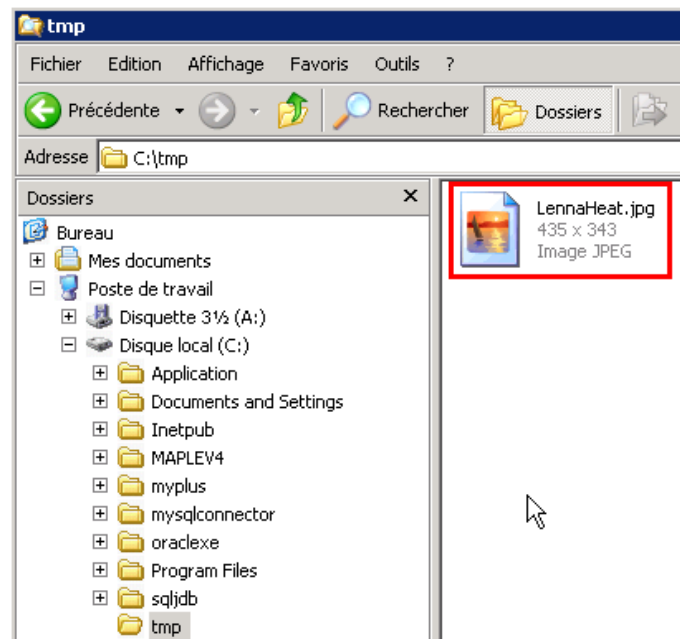
EDITOR PUBLISH VIEW
1  rgb=imread('E:\Lenna.png');
2  image(rgb);
3  title('RGB image');
4  hold on;
5  rectangle('position',[250 250 100 30],'LineWidth',2, 'EdgeColor','b')
6  %and here we export the image
7  f=getframe(gca);
8  [rgb] = frame2im(f);
9  imwrite(rgb,'C:\tmp\LennaHeat.jpg','jpg');

```

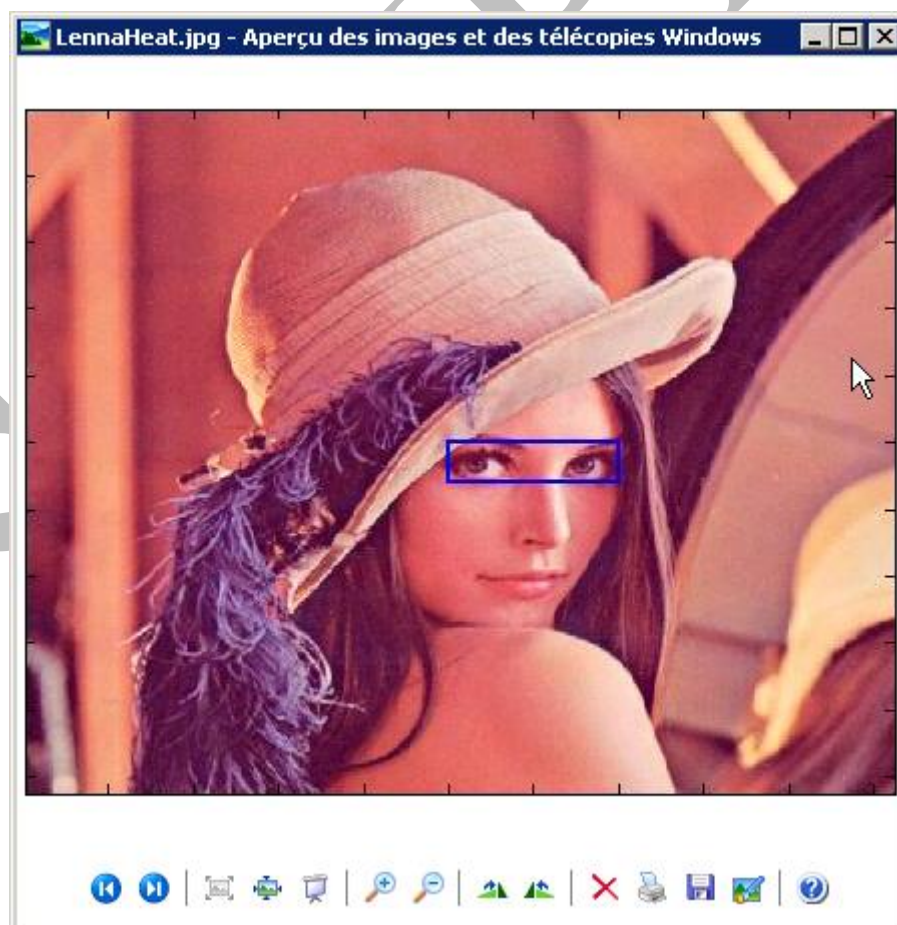
Thus we get:



And it will create a file:



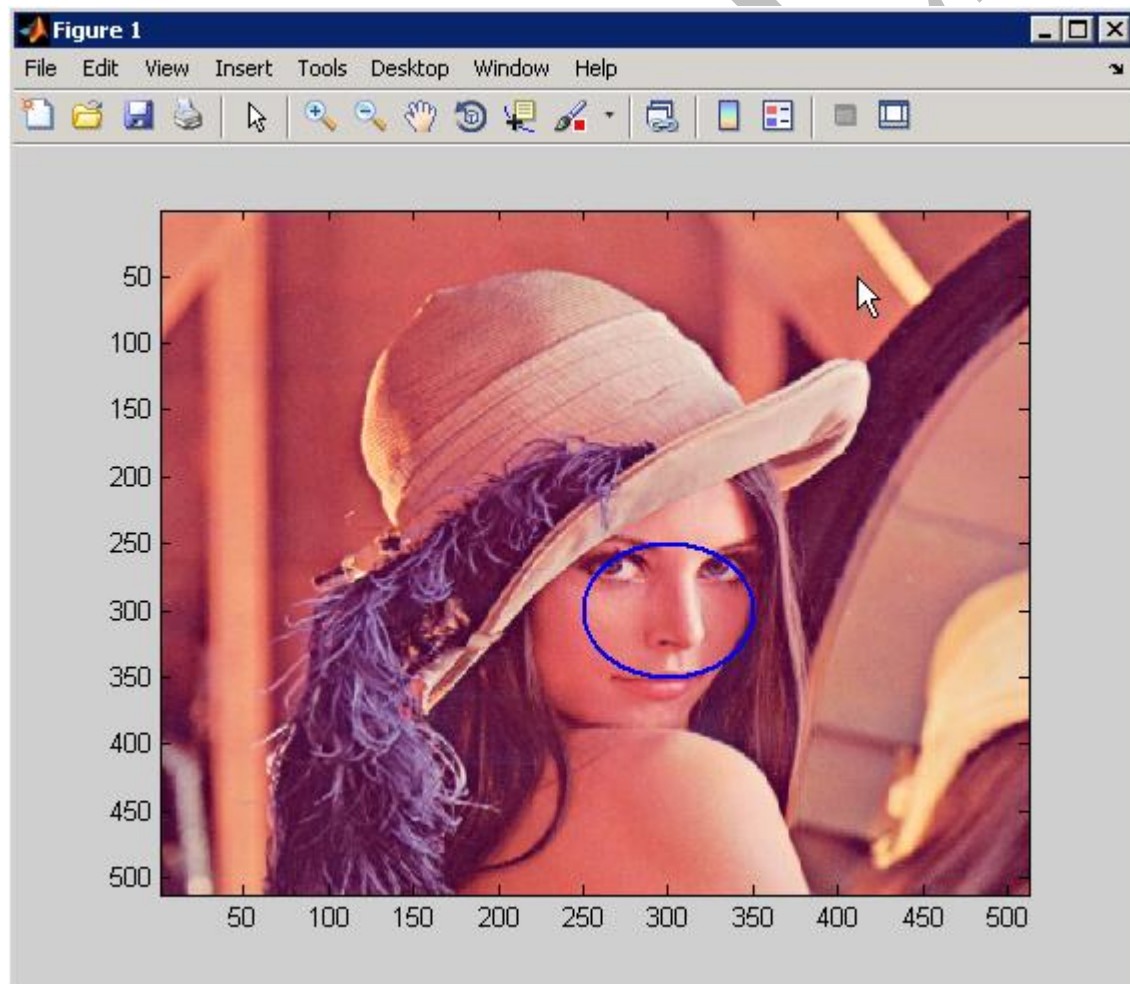
who's preview gives:



Or if you want to draw a circle or an ellipse:

```
EDITOR PUBLISH VIEW
1 - clear all;close all;clc
2
3 - img=imread('Lenna.png');
4 - image(img);
5 - rectangle('position',[250 250 100 100],'LineWidth',2,...
6         'EdgeColor','b','Curvature',[1 1]);
7
8 - f=getframe(gca);
9 - img=frame2im(f);
10 - imwrite(img,'LennaReconnaissance.jpg');
```

That will give:

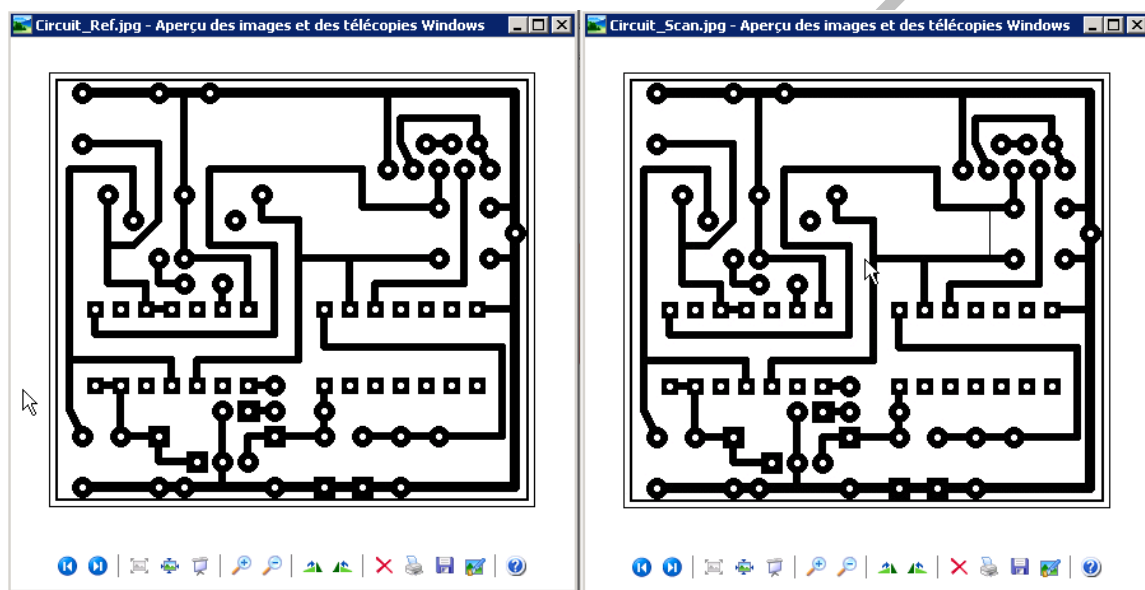


Compare images

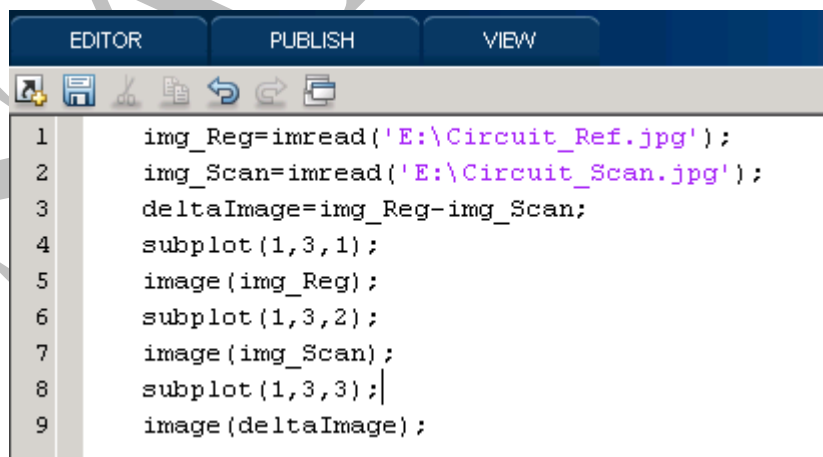
The comparison of images is a very important subject in quality control of production processes to compare pieces to a reference one and rejecting the too bad one.

Caution! The images (when you don't have the Image Processing Toolbox) must have the same size!

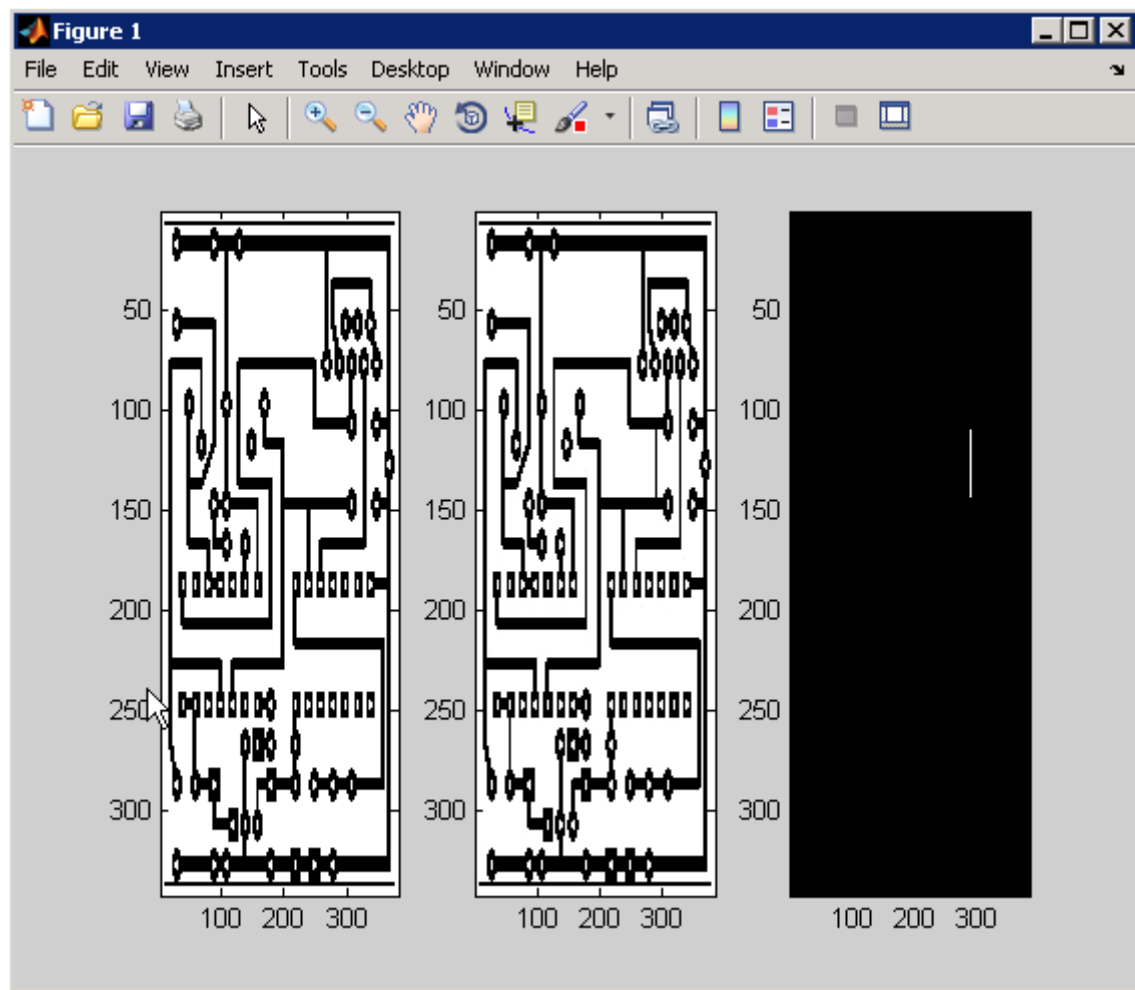
For the example consider the two images below. The reference is on the left, the tested one is on the right (for the conversion of an image in black-white see the chapter of Image Processing Toolbox on page 1039):



The comparison simply use a subtraction of the matrices of the both images:



The plotted result is on the right. After we just have to make an arithmetic sum of all pixels to know if we have to significate difference or not:



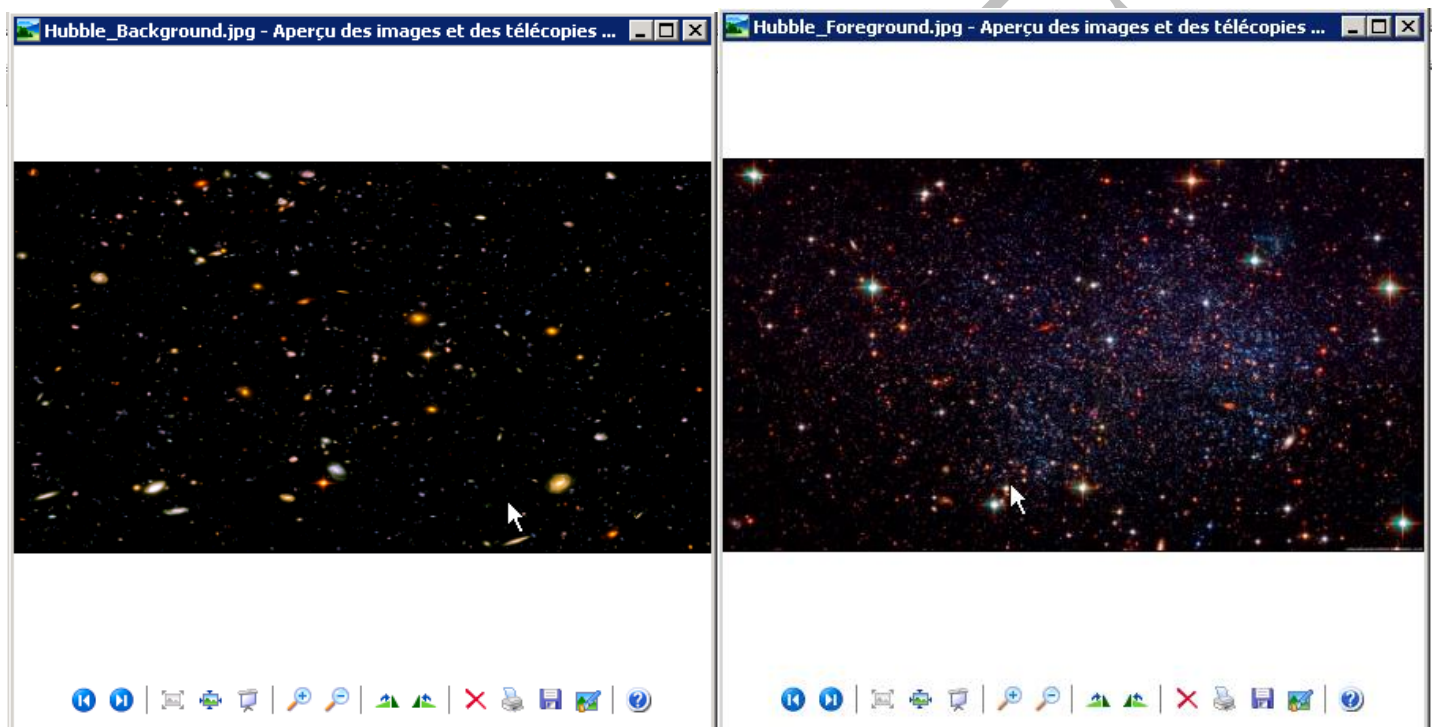
Add (superpose) images

The superposition of images is useful in many scientific fields. One of the most common after medical purposes is astronomy when you have more than one image of a same sky area using different spectrum.

Caution! The images (when you don't have the Image Processing Toolbox) must have the same size!

Caution! The black color is transparent by default for MATLAB!

Suppose we have the following images:



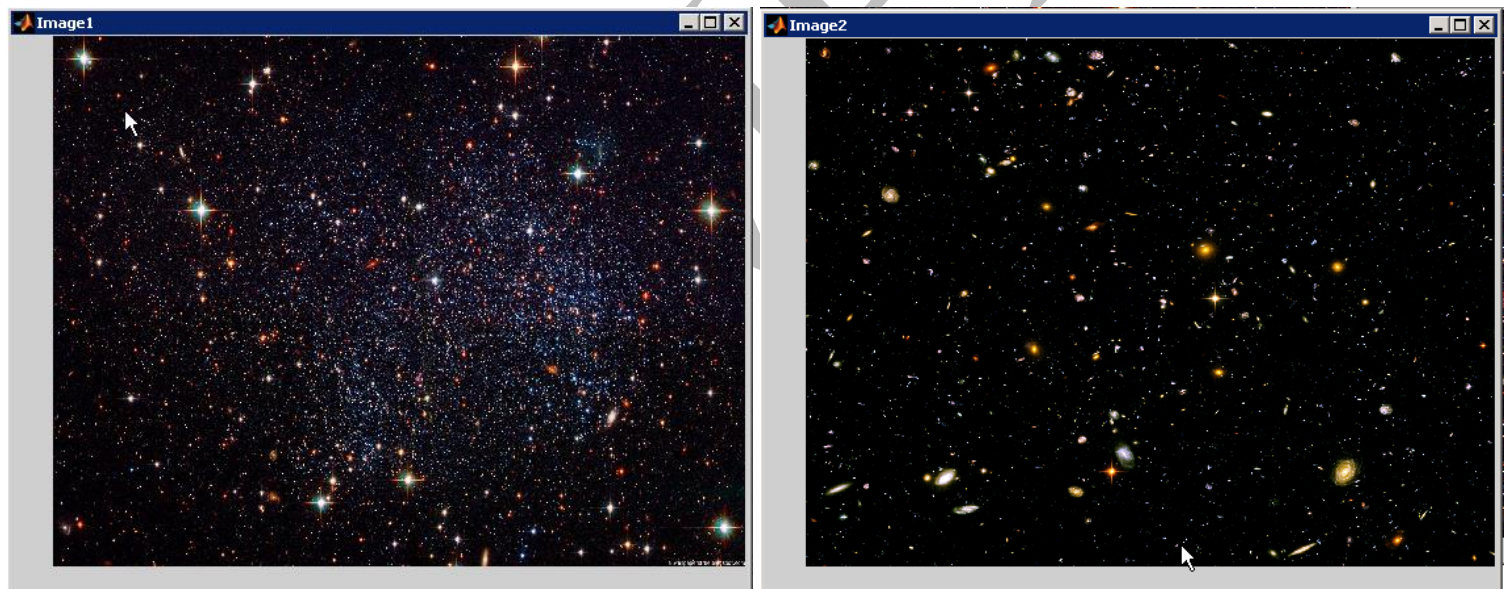
To merge them we just make the sum of the matrices of the both images because black is transparent (0,0,0) by default:


```

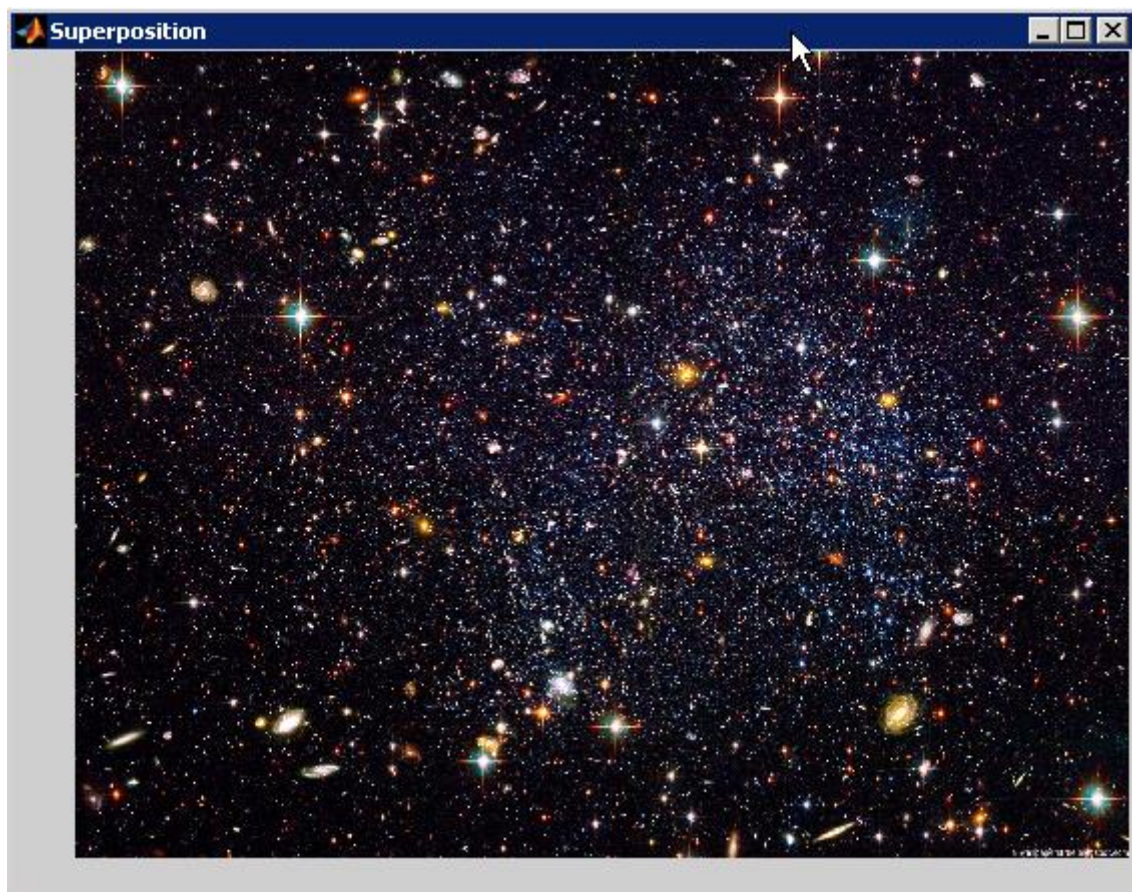
EDITOR PUBLISH VIEW
1  im1=imread('E:\Hubble_Foreground.jpg');
2  im2=imread('E:\Hubble_Background.jpg');
3  ImageSum=im1+im2;
4  image(im1);
5  set(gcf,'Menubar','none','Name','Image1','NumberTitle','off');
6  set(gca,'LooseInset',get(gca,'TightInset'))
7  axis off;
8  figure;
9  image(im2);
10 set(gcf,'Menubar','none','Name','Image2','NumberTitle','off');
11 set(gca,'LooseInset',get(gca,'TightInset'));
12 axis off;
13 figure;
14 image(ImageSum);
15 set(gcf,'Menubar','none','Name','Superposition','NumberTitle','off');
16 set(gca,'LooseInset',get(gca,'TightInset'));
17 axis off;

```

Then we get first for the two original images:



And for the final result (**ToDo**: find a way to remove MATLAB plot margins...):



Video processing

Without the Image Processing Toolbox working with videos in MATLAB is also possible but requires a lot of manipulations of matrices on each video frame to obtain simple results. Thus, the purpose of this chapter will be to see simple operations that we can get without too much pain using native core function.

Get video properties

The basics of working with videos are to retrieve information concerning the video file itself. Consider the following video:



We get the properties by using the function `mmfileinfo()`:

```
MATLAB Command Window
>> mmfileinfo('E:\clock.avi')

ans =

    Filename: 'clock.avi'
      Path: 'E:\'
   Duration: 12
     Audio: [1x1 struct]
     Video: [1x1 struct]

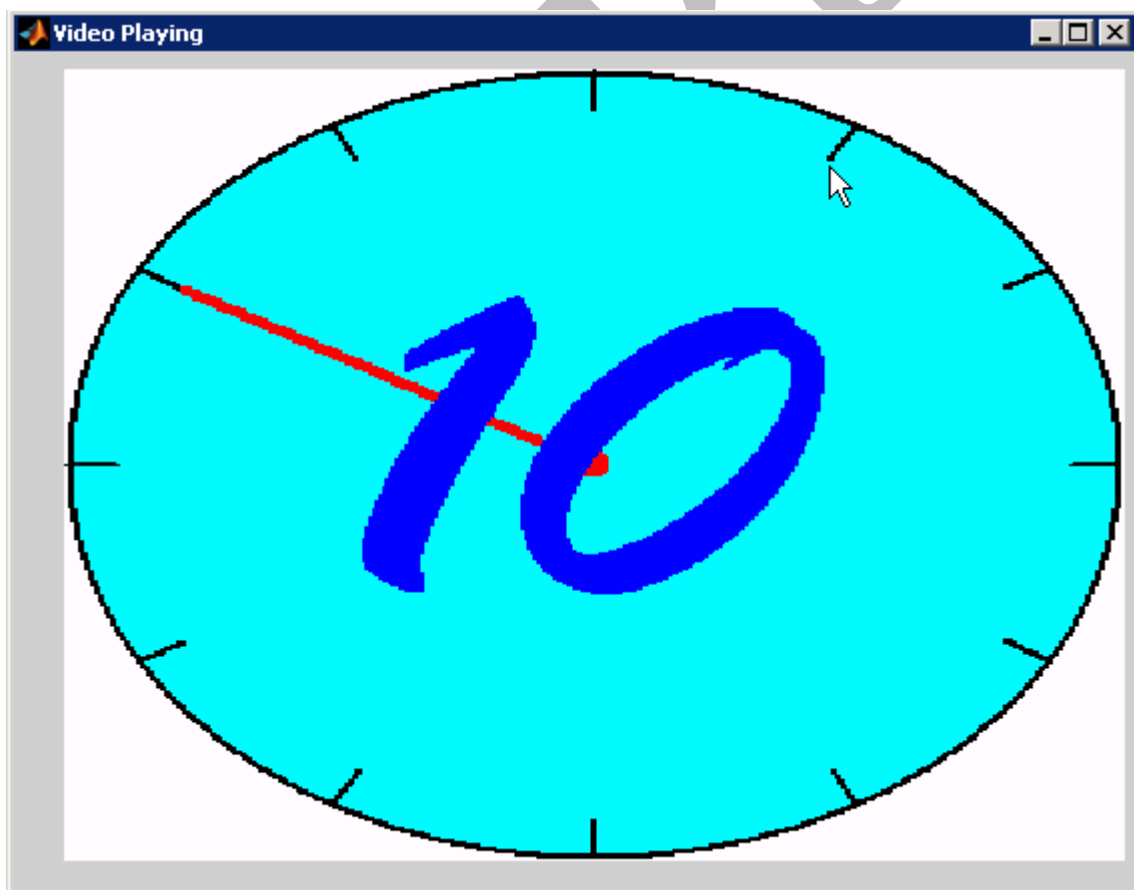
>> |
```


Play video

Without the Image and Visio Processing Toolbox is almost a pain in the a... to play a video. To play we will need to usage the function **VideoReader()**:

```
EDITOR PUBLISH VIEW
1 set(gcf,'Menubar','none','Name','Video Playing','NumberTitle','off');
2 set(gca,'LooseInset',get(gca,'TightInset'))
3 vid=VideoReader('E:\clock.avi');
4 numFrames = vid.NumberOfFrames;
5 n=numFrames;
6 for i=1:n
7     frames = read(vid,i);
8     im(i)=image(frames);
9     axis off;
10 end
```

So the video will play in a MATLAB figure (**ToDo**: found how to remove margins):

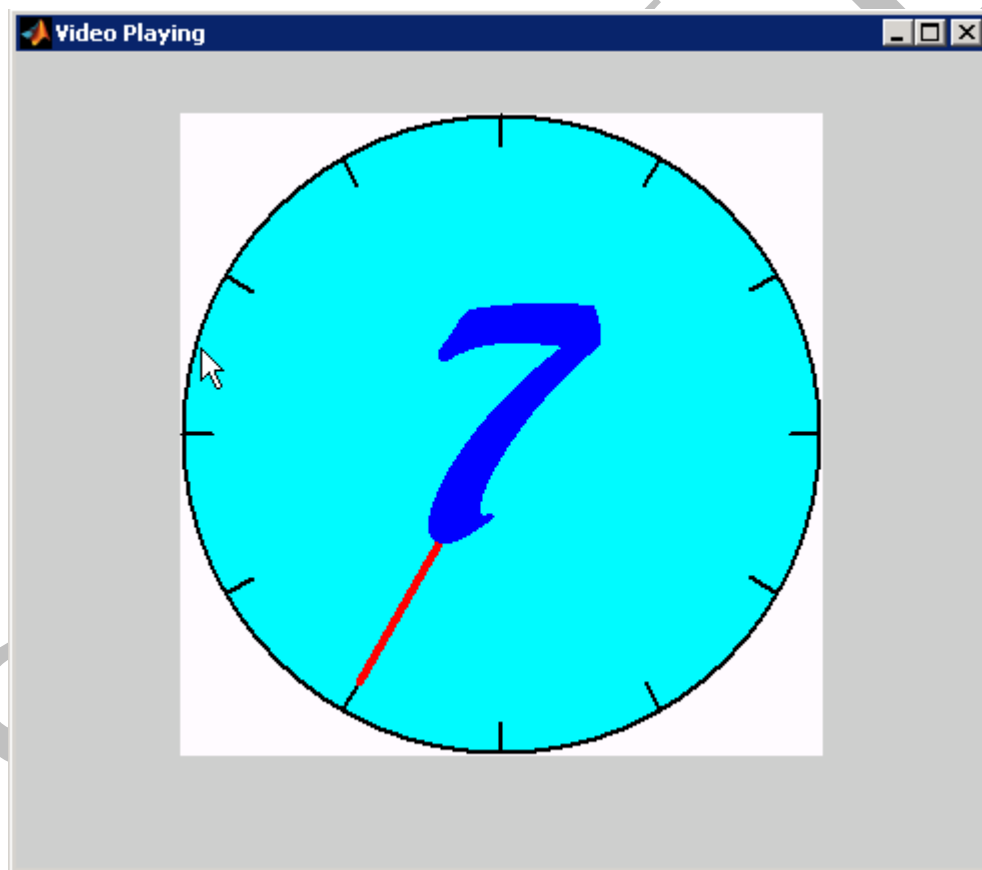


Extract a frame

Extracting a frame from a video is one of the main thing you must learn to do post-processing and this can be done with the conjunctions of the functions **VideoReader()** and **read()**:

```
EDITOR PUBLISH VIEW
1 vid=VideoReader('E:\clock.avi');
2 im=read(vid,7);
3 imshow(im);
4 set(gcf,'MenuBar','none','Name','Video Playing','NumberTitle','off');
5 set(gca,'LooseInset',get(gca,'TightInset'));
```

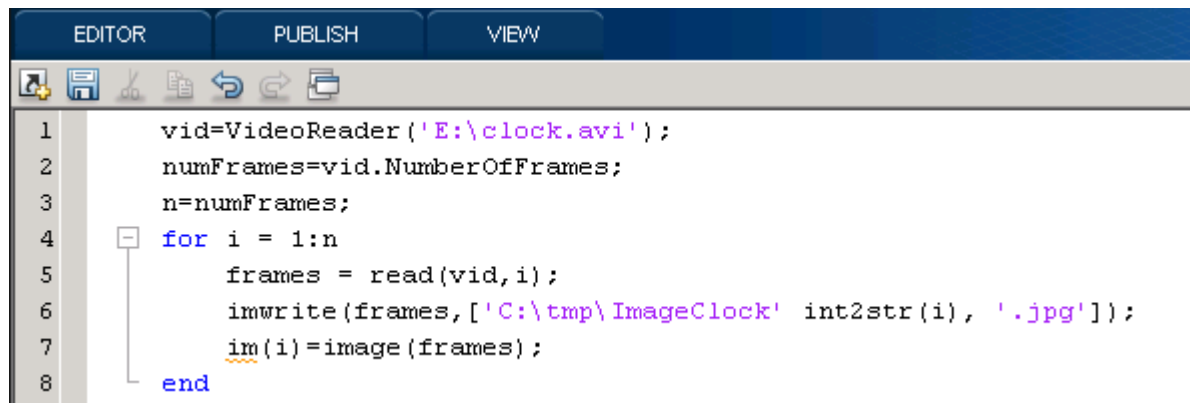
So we get for the extraction of the 7th frame:



Export all video frames in a folder

Something also useful sometimes is to decompose a video into multiple frames (images) in a folder for postprocessing purposes.

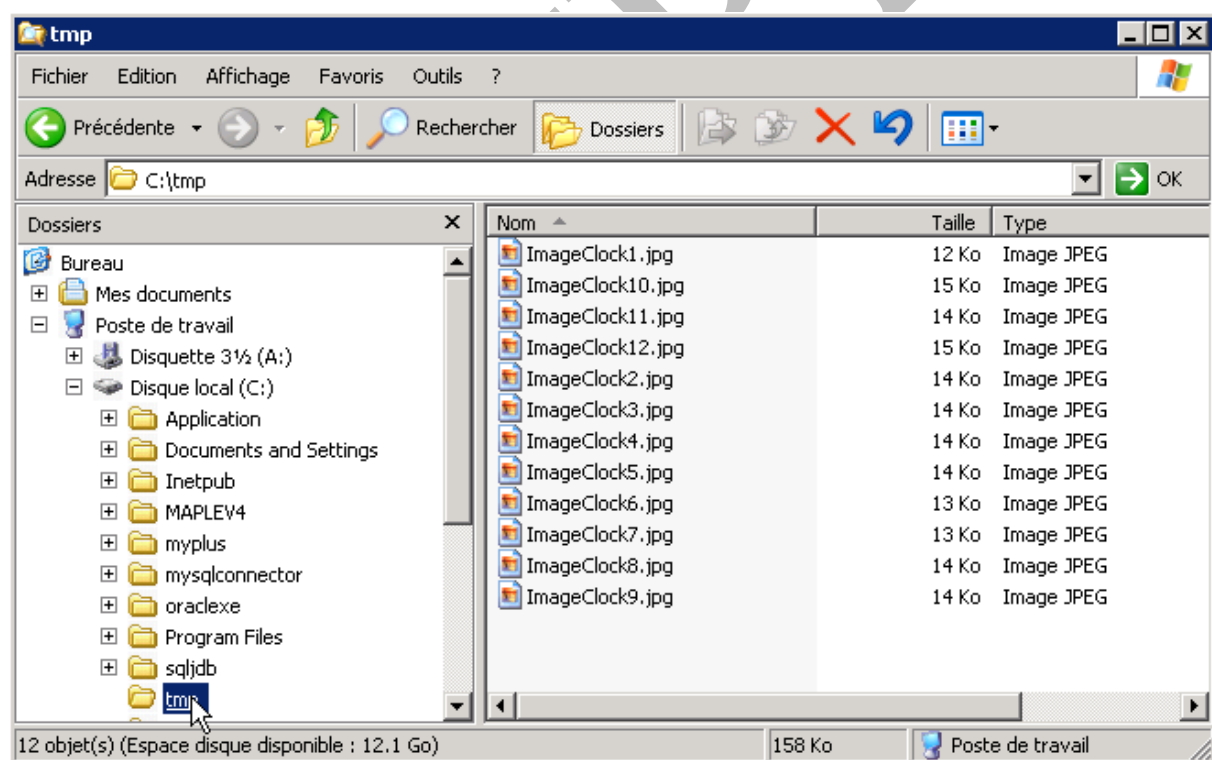
For this purpose we will use the property **NumberOfFrames** of the method **VideoReader()** and after the well know **imwrite()** function:



```

1      vid=VideoReader('E:\clock.avi');
2      numFrames=vid.NumberOfFrames;
3      n=numFrames;
4      for i = 1:n
5          frames = read(vid,i);
6          imwrite(frames,['C:\tmp\ImageClock' int2str(i), '.jpg']);
7          im(i)=image(frames);
8      end
  
```

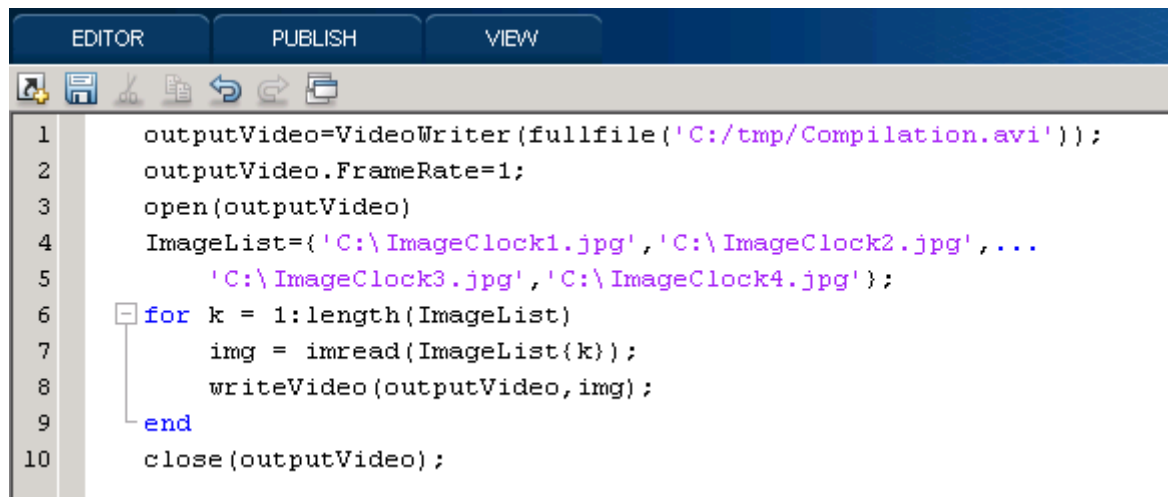
This will simply give the expected result:



Build a video from a list of (pre-processed or not) images

In the example before we saw how to export frames into a folder for preprocessing. Let us see now how to do the opposite: import images to build a video!

For this purpose we create a new file video using the function **VideoWriter()** and after we define the images sequence into the file using the function **writeVideo()**:

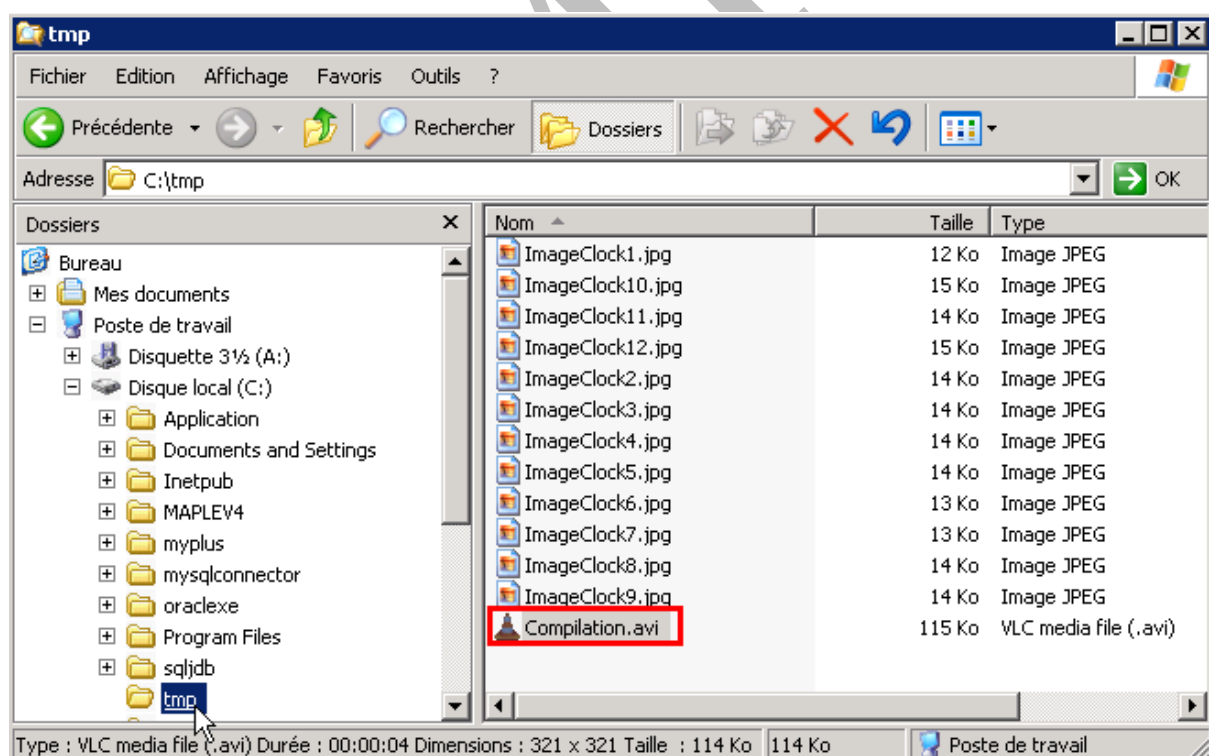


```

1  outputVideo=VideoWriter(fullfile('C:/tmp/Compilation.avi'));
2  outputVideo.FrameRate=1;
3  open(outputVideo)
4  ImageList={'C:\ImageClock1.jpg','C:\ImageClock2.jpg',...
5            'C:\ImageClock3.jpg','C:\ImageClock4.jpg'};
6  for k = 1:length(ImageList)
7      img = imread(ImageList{k});
8      writeVideo(outputVideo,img);
9  end
10 close(outputVideo);

```

As you can see below we get a new video file:



And that's it!

Sound processing

Caution! Nothing will appear on the screen with the example below (no charts, no player control).

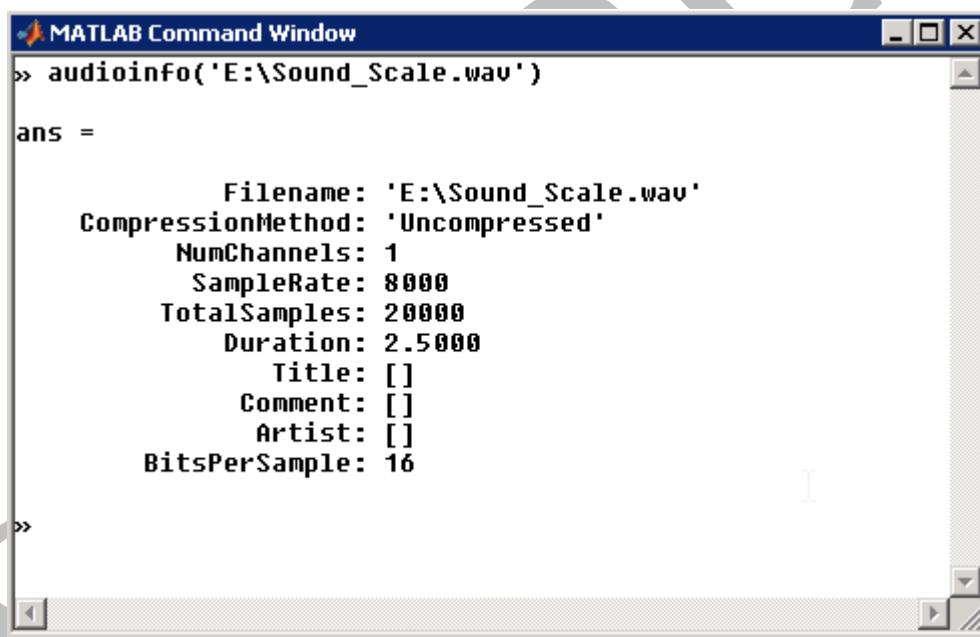
Without the Signal Processing Toolbox working with sounds (signals) in MATLAB is also possible but requires a lot of manipulations of vectors to obtain simple results. Thus, the purpose of this chapter will be to see simple operations that we can get without too much pain using native core function.

Get sound properties

The basics of working with sounds/signals are to retrieve information concerning the signal file itself. Consider the following *.wav file:

 Sound_Scale.wav

To get information about the content we use the function **audioinfo()**:



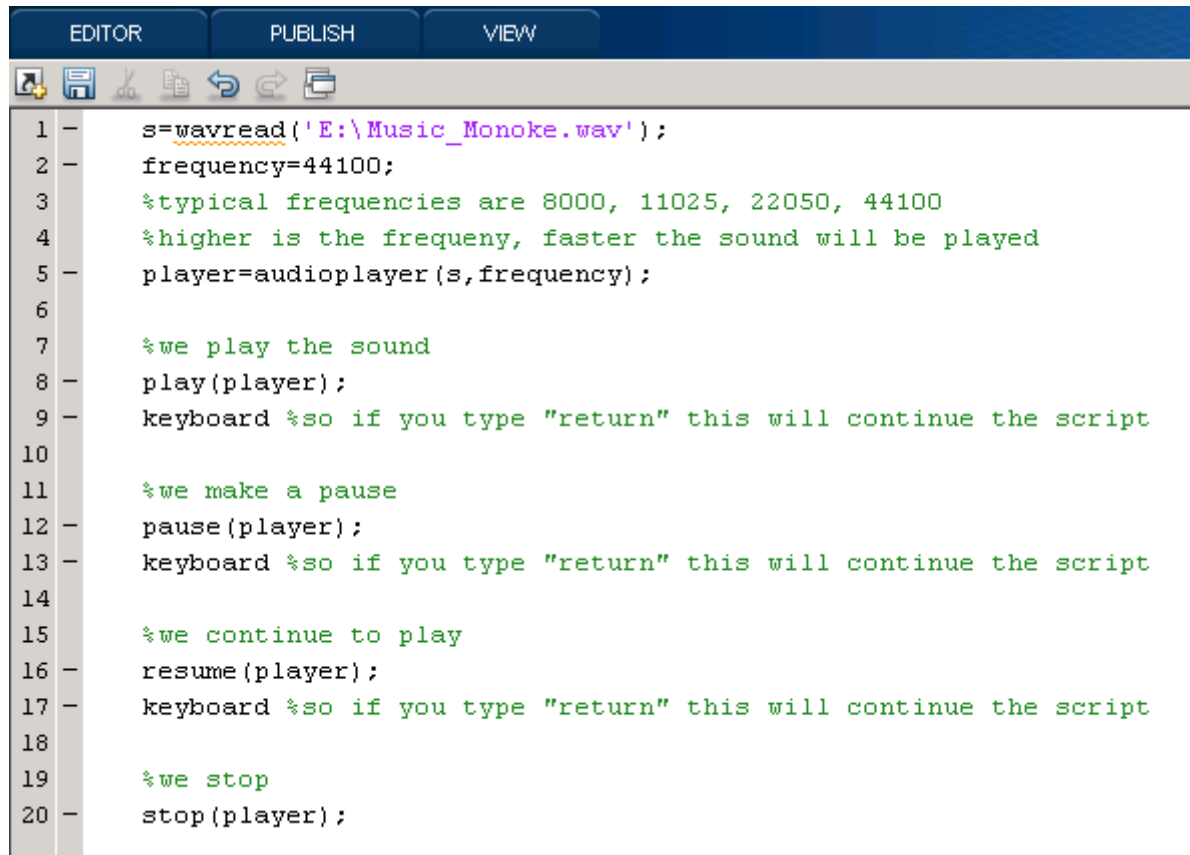
```
MATLAB Command Window
>> audioinfo('E:\Sound_Scale.wav')

ans =

        Filename: 'E:\Sound_Scale.wav'
    CompressionMethod: 'Uncompressed'
        NumChannels: 1
         SampleRate: 8000
    TotalSamples: 20000
         Duration: 2.5000
           Title: []
          Comment: []
          Artist: []
    BitsPerSample: 16
```

Play, Pause (resume), Resume, Stop sound playing

How using the functions **audioplayer()**, **play()**, **pause()**, **resume()** and **stop()** we will see how to play, pause, continue and stop playing a sound/signal:



```
1 - s=wavread('E:\Music_Monoke.wav');
2 - frequency=44100;
3 - %typical frequencies are 8000, 11025, 22050, 44100
4 - %higher is the frequency, faster the sound will be played
5 - player=audioplayer(s,frequency);
6
7 - %we play the sound
8 - play(player);
9 - keyboard %so if you type "return" this will continue the script
10
11 - %we make a pause
12 - pause(player);
13 - keyboard %so if you type "return" this will continue the script
14
15 - %we continue to play
16 - resume(player);
17 - keyboard %so if you type "return" this will continue the script
18
19 - %we stop
20 - stop(player);
```

Plot a sound spectrum

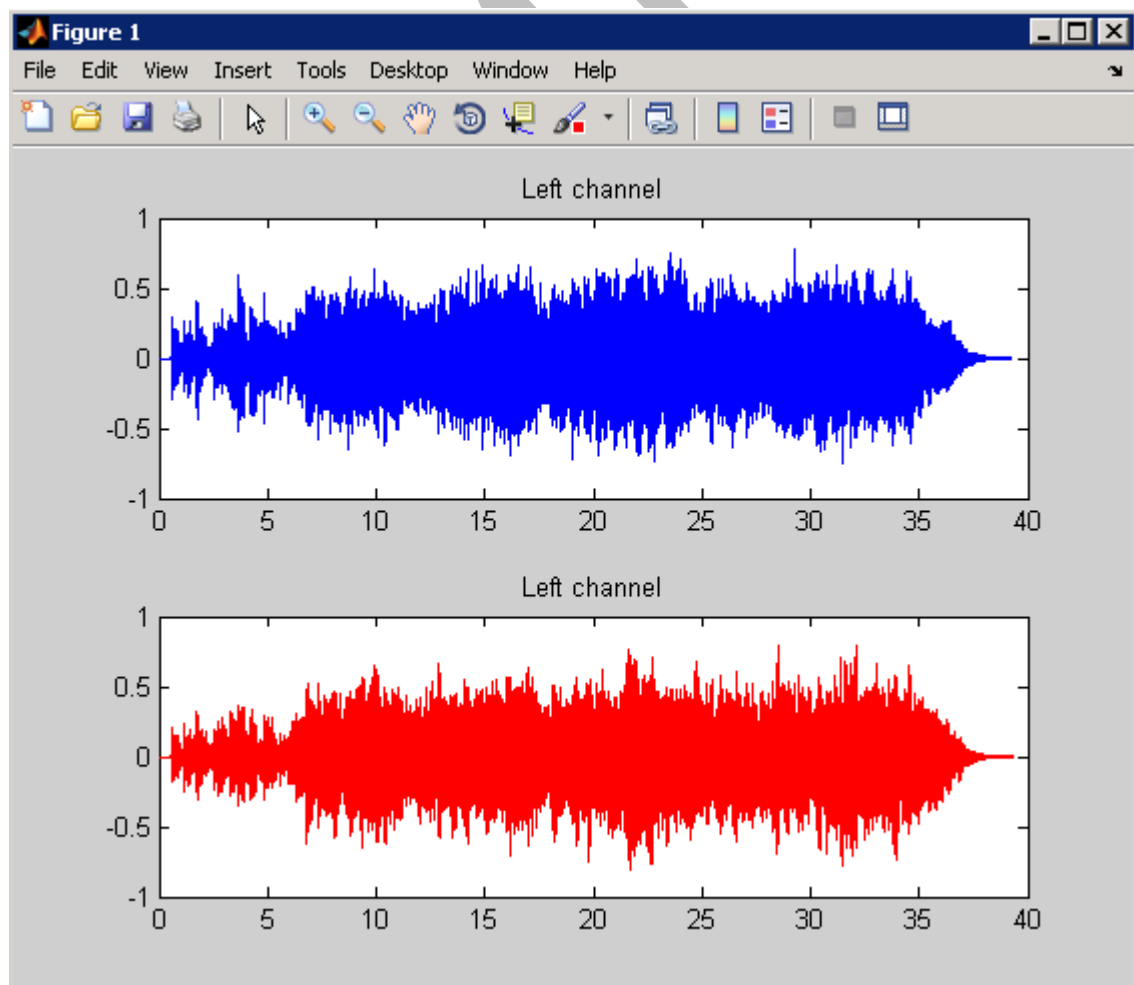
Plot a sound spectrum is one of the most common requests of MATLAB signal processing engineers. Without Toolbox the beginning of the procedure is not necessarily intuitive because it required to use the sampling frequency that we can retrieve using the **wavread()** function as you can see below

```

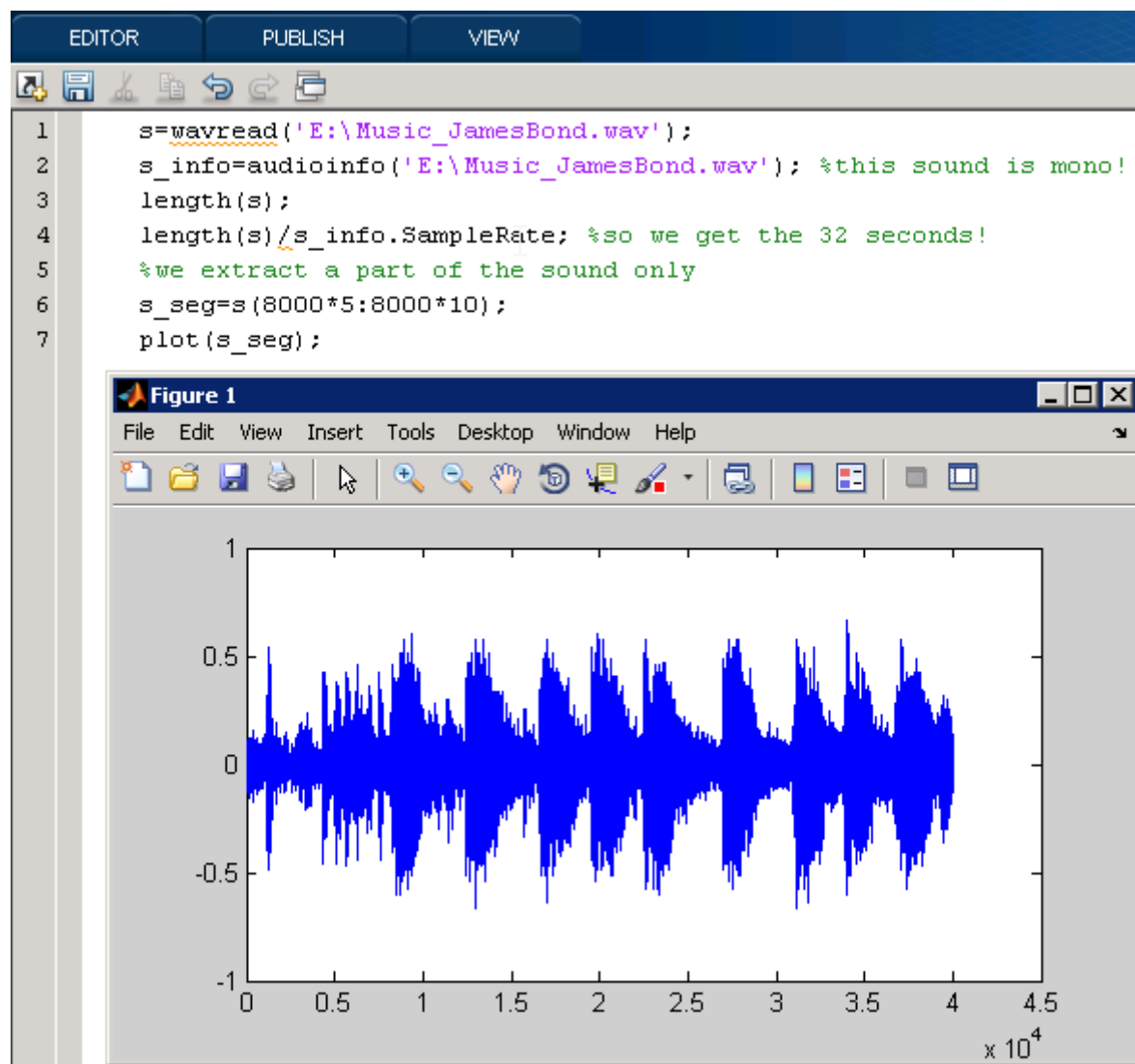
EDITOR PUBLISH VIEW
1 - [wave,fs]=wavread('E:\Music_Monoke.wav');
2 - t=0:1/fs:(length(wave)-1)/fs;
3 - %this sound is stereo the the "wave" is a matrix with two column
4 - %this is why the plot has two spectrum (one in blue the other in red)
5 - subplot(2,1,1)
6 - plot(t, wave(:,1));
7 - title('Left channel');
8 - subplot(2,1,2)
9 - plot(t, wave(:,2), 'r');
10 - title('Left channel');

```

The we get of a stereo sound signal:



Or another way to do that:



Sound FFT (DFT)

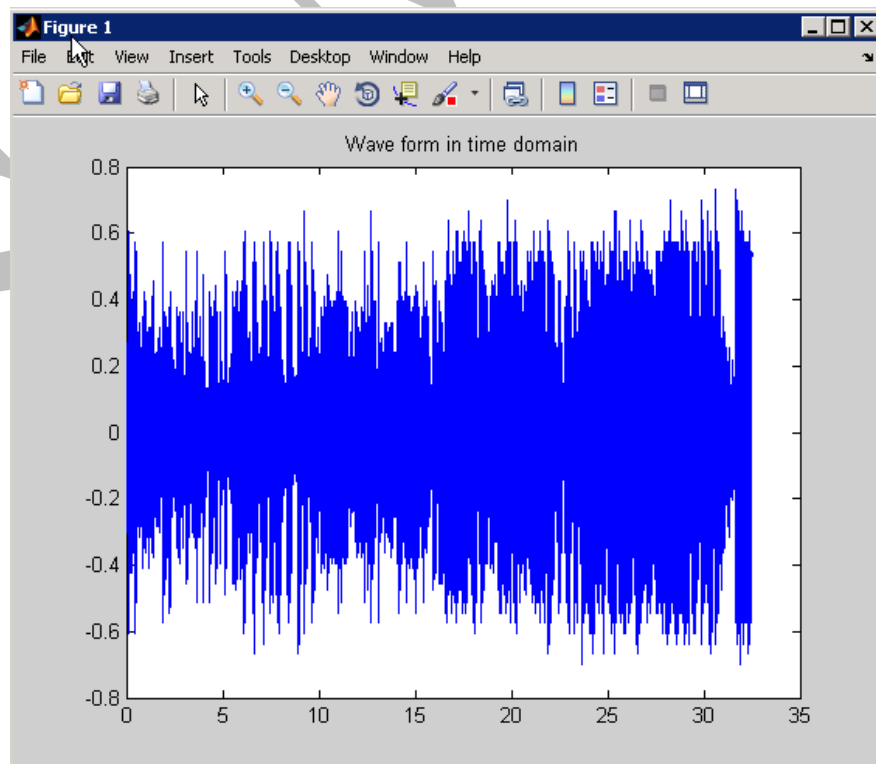
We have already seen how to make an FFT with a random signal (see page 467). The only purpose here is to see a simple way to the same with a sound signal using once again the function `fft()` and `nextpow2()`:

```

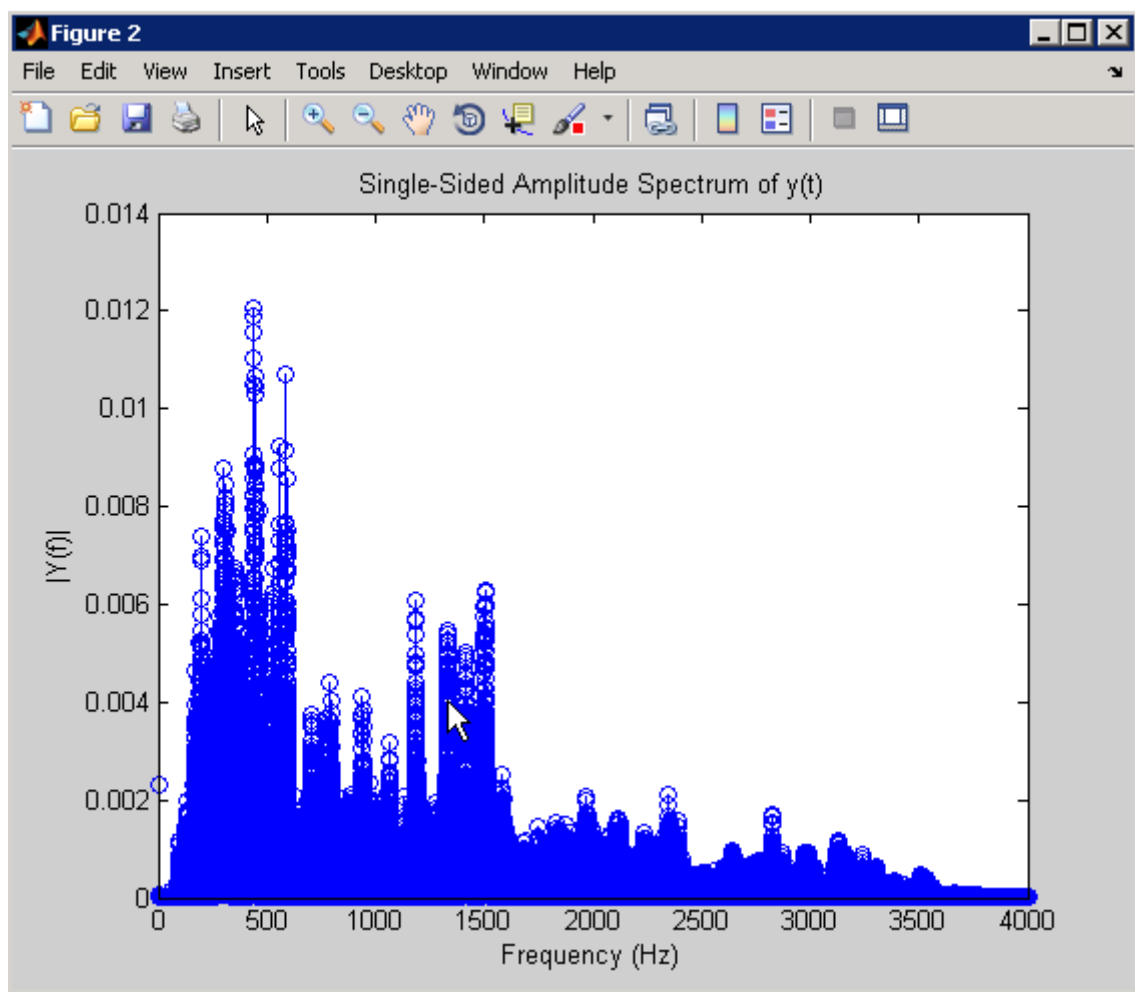
1  y=wavread('E:\Music_JamesBond.wav');
2  Fs=8000; %Sampling frequency
3  T=1/Fs; %Sample time
4  L=length(y); %Length of signal
5  t=(0:L-1)*T; %Time vector
6  NFFT=2^nextpow2(L); %Next power of 2 from length of y
7  Y=fft(y,NFFT)/L;
8  f=Fs/2*linspace(0,1,NFFT/2+1);
9  figure(1)
10 plot(t,y)
11 title('Wave form in time domain')
12 %Plot single-sided amplitude spectrum.
13 figure(2)
14 stem(f,2*abs(Y(1:NFFT/2+1)))
15 title('Single-Sided Amplitude Spectrum of y(t)')
16 xlabel('Frequency (Hz)')
17 ylabel('|Y(f)|');

```

The original sound signal is:

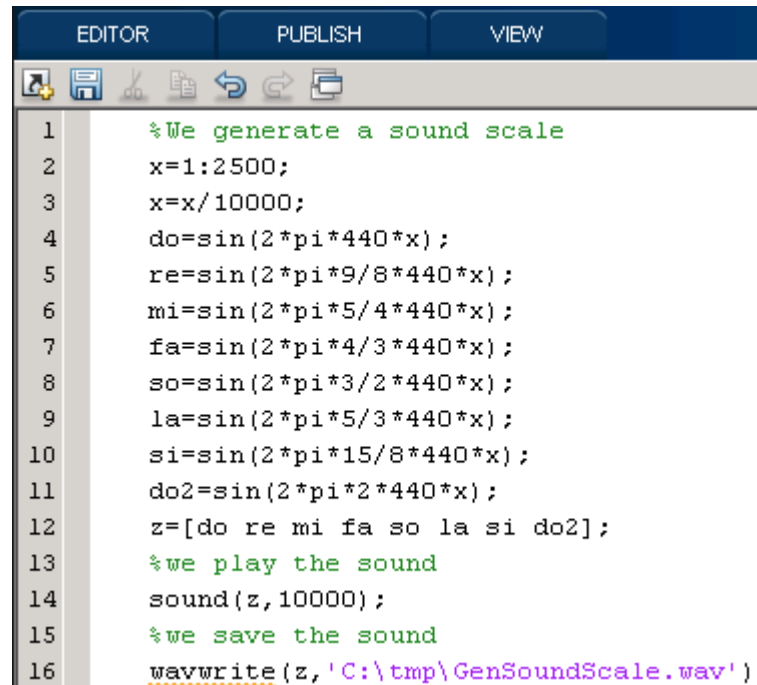


And the corresponding FFT (DFT) gives:



Generate a sound and save it

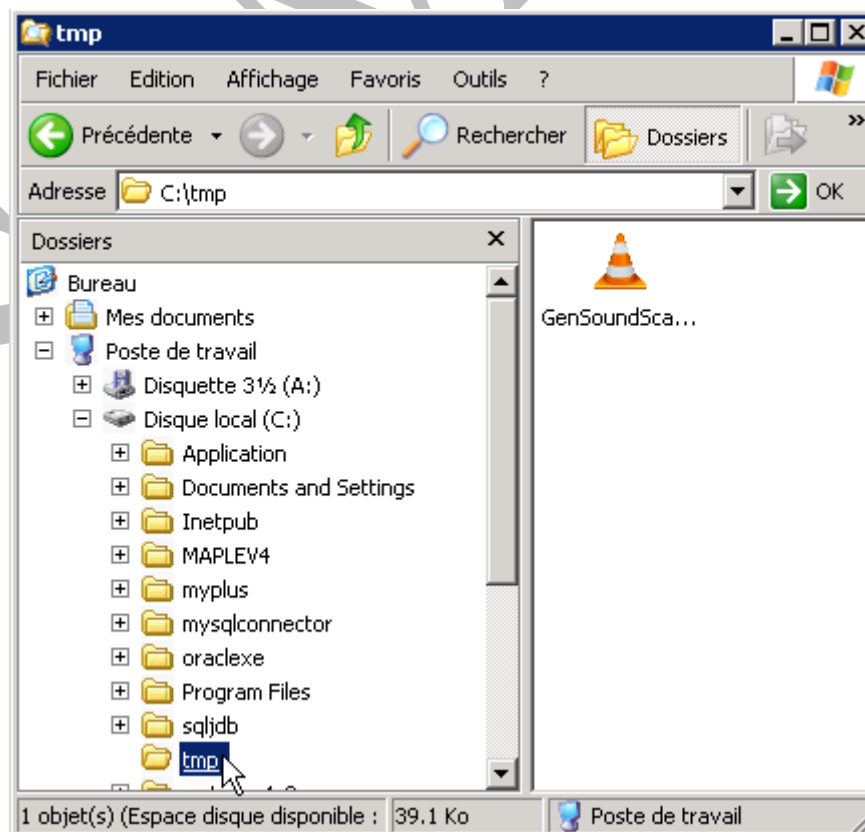
Create artificial sounds is a major subject and business in CAM (Computer Assisted Music). We will see here a small academic example on how to play a sound scale using the function `sound()` and the function `wavwrite()` to save it on the computer:



```

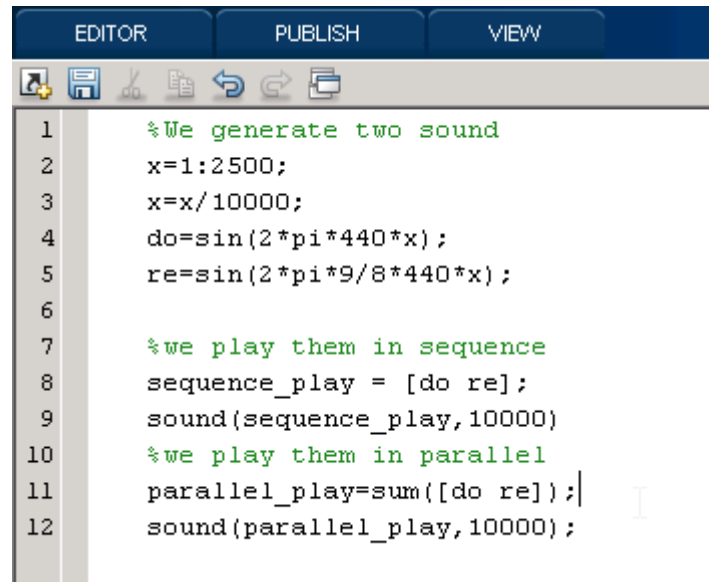
1      %We generate a sound scale
2      x=1:2500;
3      x=x/10000;
4      do=sin(2*pi*440*x);
5      re=sin(2*pi*9/8*440*x);
6      mi=sin(2*pi*5/4*440*x);
7      fa=sin(2*pi*4/3*440*x);
8      so=sin(2*pi*3/2*440*x);
9      la=sin(2*pi*5/3*440*x);
10     si=sin(2*pi*15/8*440*x);
11     do2=sin(2*pi*2*440*x);
12     z=[do re mi fa so la si do2];
13     %we play the sound
14     sound(z,10000);
15     %we save the sound
16     wavwrite(z,'C:\tmp\GenSoundScale.wav')
  
```

This will create a file as you can see below:



Play sounds in sequence or parallel

In the example before we have play a sound in sequence (serial) but what about playing two or more sounds in parallel using the function **sound()**?:



```
1  %We generate two sound
2  x=1:2500;
3  x=x/10000;
4  do=sin(2*pi*440*x);
5  re=sin(2*pi*9/8*440*x);
6
7  %we play them in sequence
8  sequence_play = [do re];
9  sound(sequence_play,10000)
10 %we play them in parallel
11 parallel_play=sum([do re]);
12 sound(parallel_play,10000);
```

Scripting

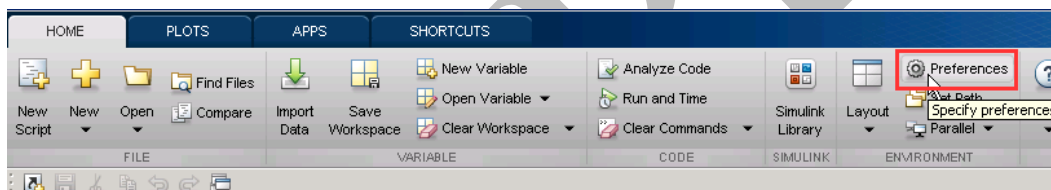
Caution! New releases of MATLAB have autocompletion/intellisense for built-in functions but not for contextual variables/functions that you create yourself!

MATLAB script is the major subject of MATLAB as for any scientific tool. Scripts gives you the possibility to automate analysis, trading orders, machine behaviors, etc. The most important fact in MATLAB is that you can at the end compile you script in a C++ ou C# *.dll and use the resulting code in your own applications.

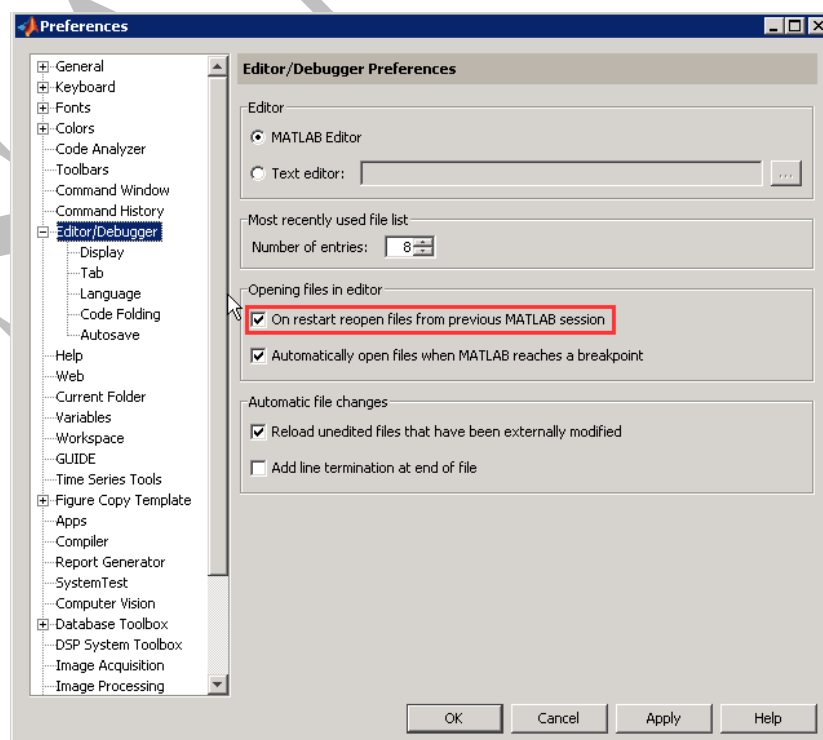
The MATLAB script language is not a programming language, you do not have a difference between routine and functions and the variables are not signed but you must always keep in mind the purpose of the R&D engineer is not to become and computer scientist but to have a RAD (rapid application development) solution to not loose to much time on computer and focus more on lab stuff.

Script Editor most asked preferences

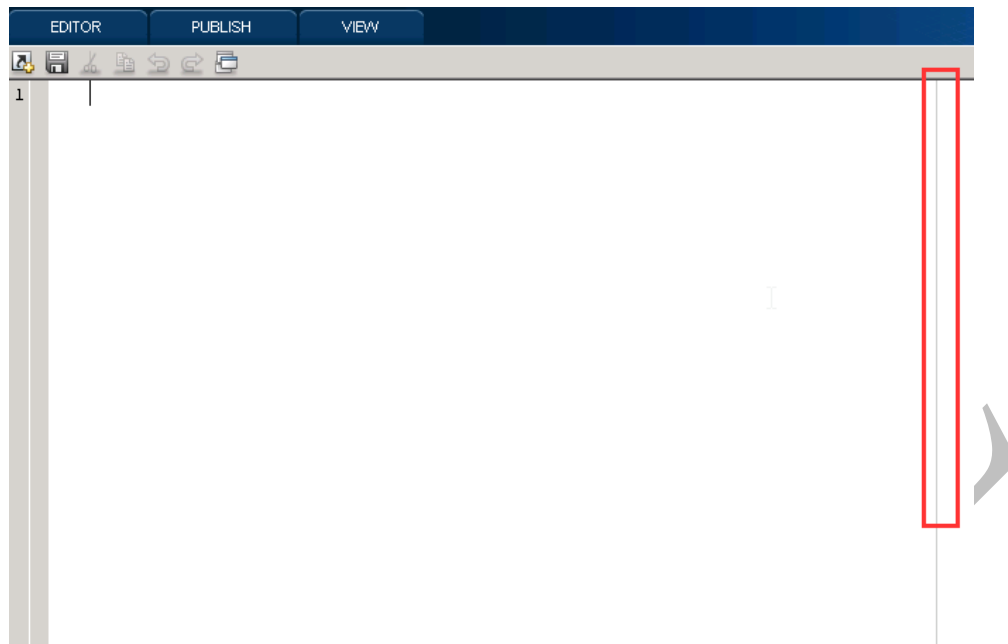
Let us first begin by the most asked preferences questions of MATLAB users about the script editor. In the **Home** tab click on **Preferences**:



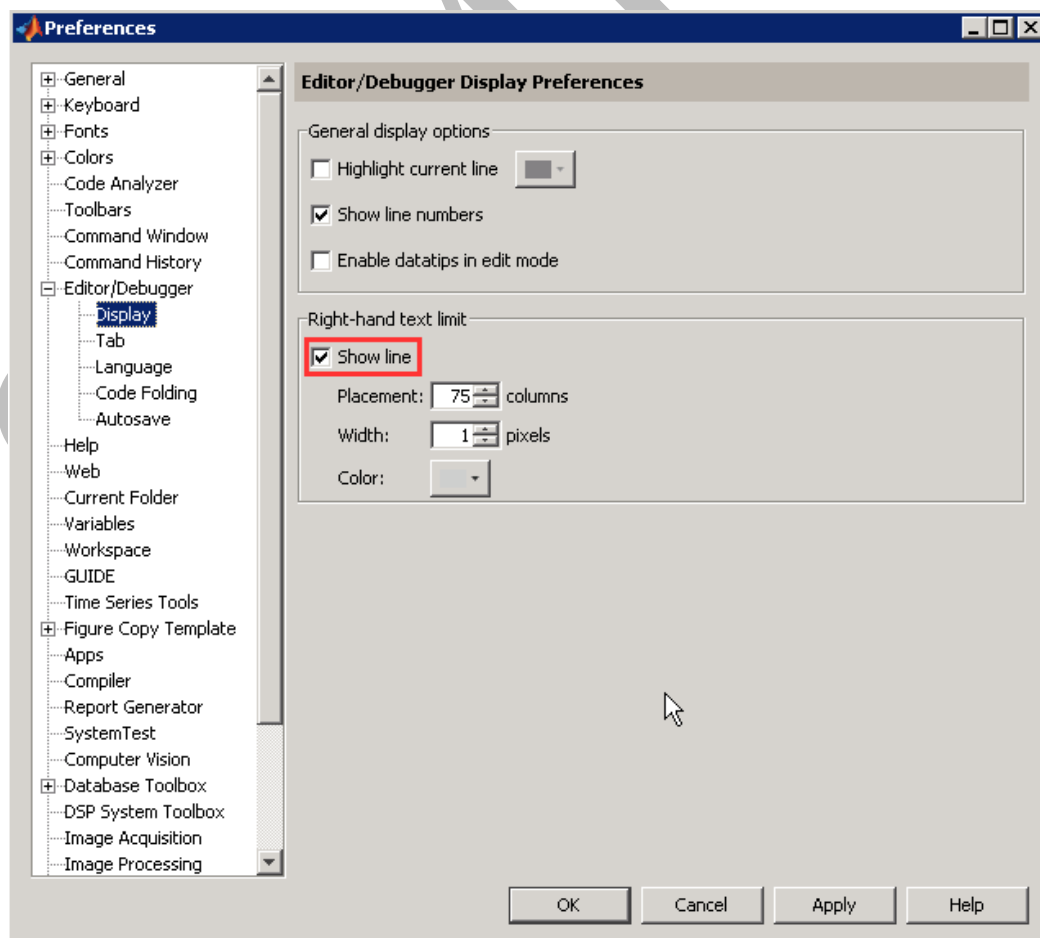
In the Editor/Debugger section you can choose or not to have last script files to be reopen at each new MATLAB session:



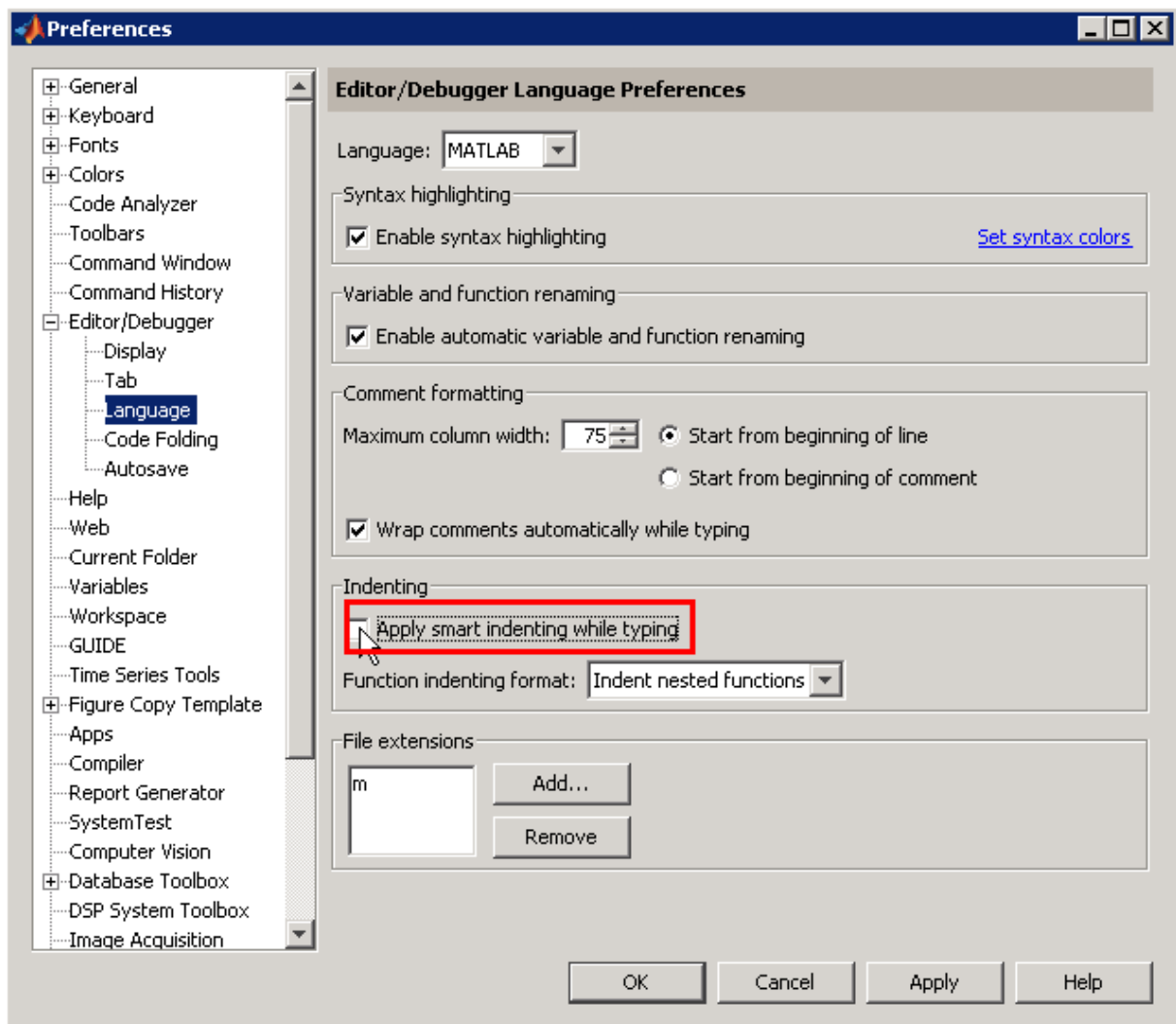
Now something boring in the script editor is the page break line on the right for people that print scripts on paper or on PDF:



but most user don't print anymore... Then let us see how to remove this trough the section **Editor/Debugger** in the subsection **Display** you can deactivate **Show line**:

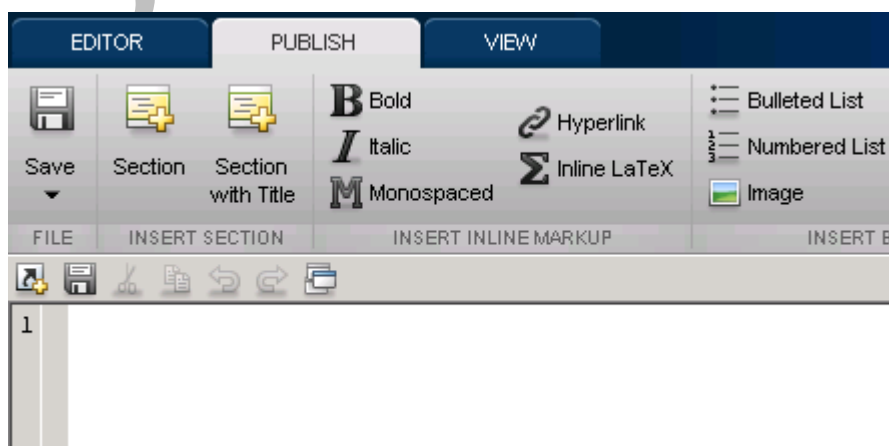


And if you don't want auto-indenting (auto-tabulation) while you are typing MATLAB script just change the following:

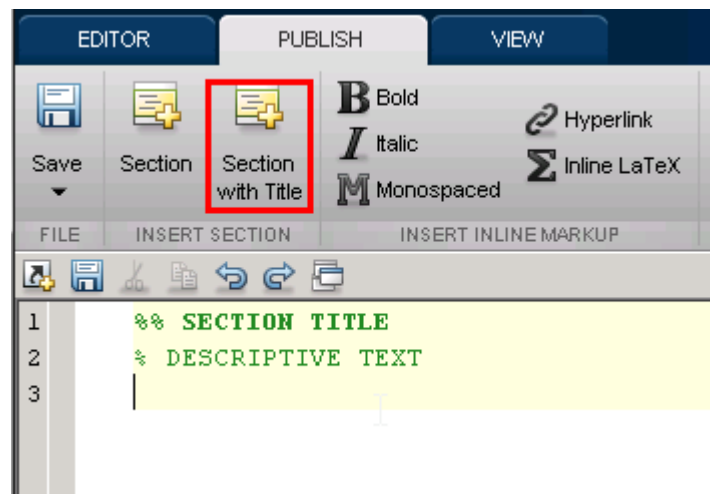


Well commented script

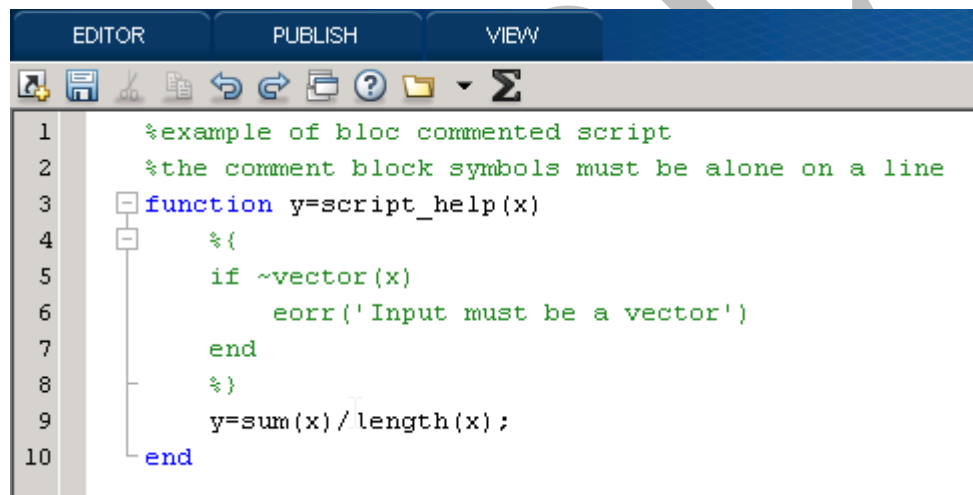
Comment script is a very important process in the field of engineering. Everything that is relative to comments is in the register **Publish**:



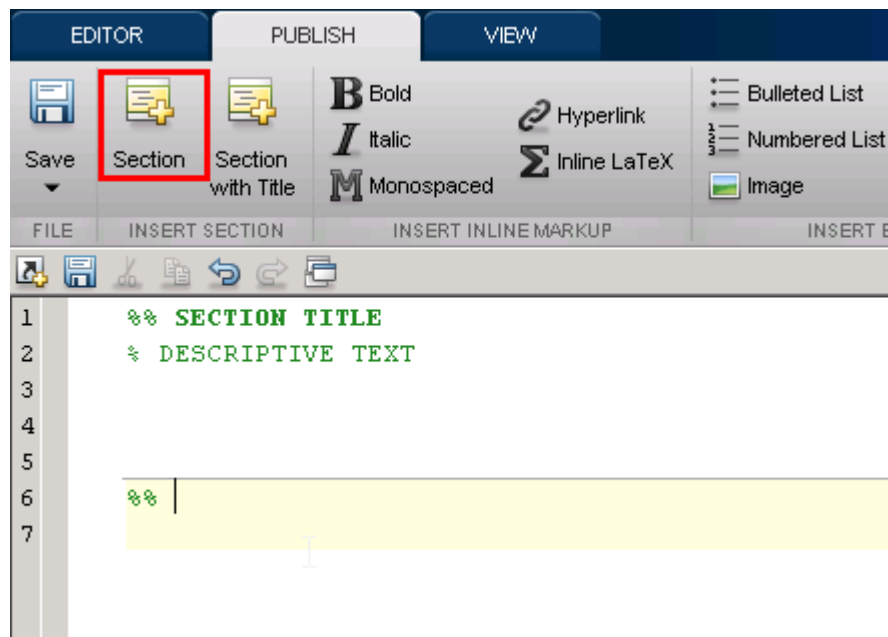
The first thing to do when you create a new script is to insert a **Section with Title**:



Thus you see that to comment a line we just simply write % at the beginning of the line or we use the keyboard shortcut **Ctrl+R**. You can also comment a block of script using %{ ... %}:



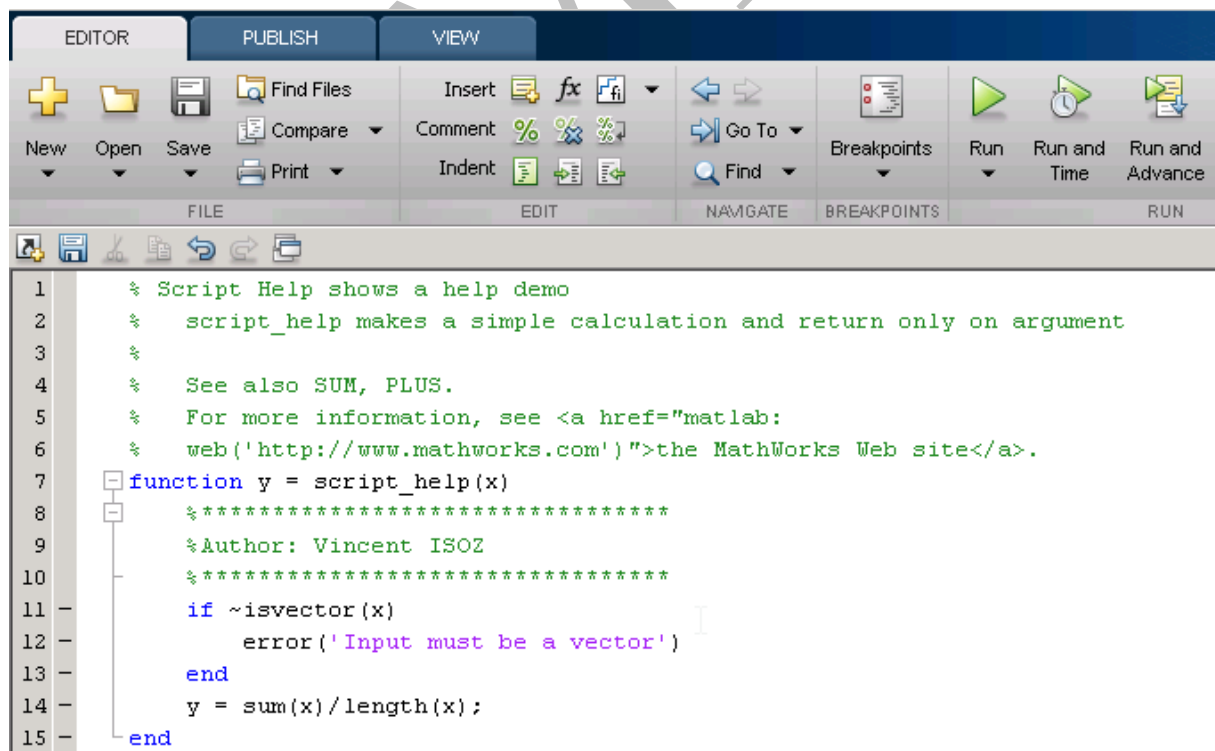
After you can insert anywhere a **Section** (also called "cell") in the code:



Note: If you press **Ctrl+Enter** to execute a code only the active section (active cell) will be run.

Define help text

In MATLAB by default the first comments at the top of the scripts will be use by the **help** command:



As you can see here is demo using simple texts and hyperlinks:

```

Command Window

>> help script_help
Script Help shows a help demo
    script_help makes a simple calculation and return only on argument

See also sum, plus.
for more information, see the MathWorks Web site.

fx >>

```

As you can see nothing is especially difficult. But in facts what we did is false!!

Yes the good method is the following (the script is not the same as above but this is not the subject, just focus on the header part!):

```

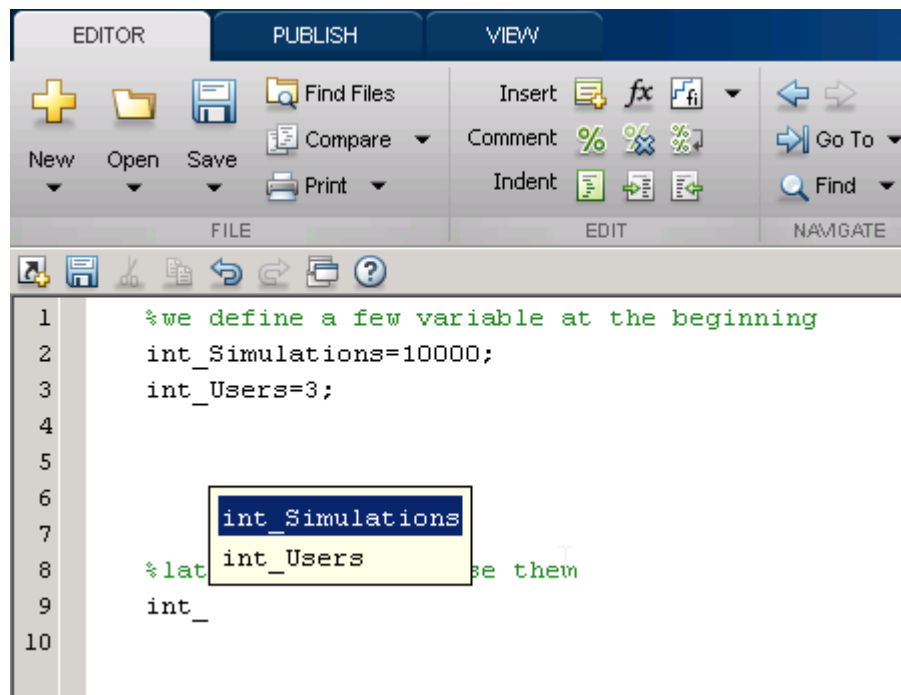
EDITOR PUBLISH VIEW
1      %%Main title of our demo script to study help syntax
2      %   script_help is a demo script to see how to correctly use MATLAB
3      %   comments
4      %
5      %   Example:
6      %
7      %   ScriptDemo(x) Then you get the following result
8      %
9      %   For more information, see <a href="www.mathworks.com">Alphorm</a>
10     %   See also: SUM, PLUS
11
12     %   Copyright 2014 Mathworks
13
14     %   $Author: V. ISOZ $       $Date: 2014/08/07 18:23:52 $       $Revision: 0.1 $
15
16     function y=script_help(x)
17     -   x=5; %this is a constant for...
18     -   %{
19     -       if ~isvector(x)
20     -           error('Input must be a vector');
21     -       end
22     -   %}
23     -   Y=sum(x)/length(x);
24     -   end

```

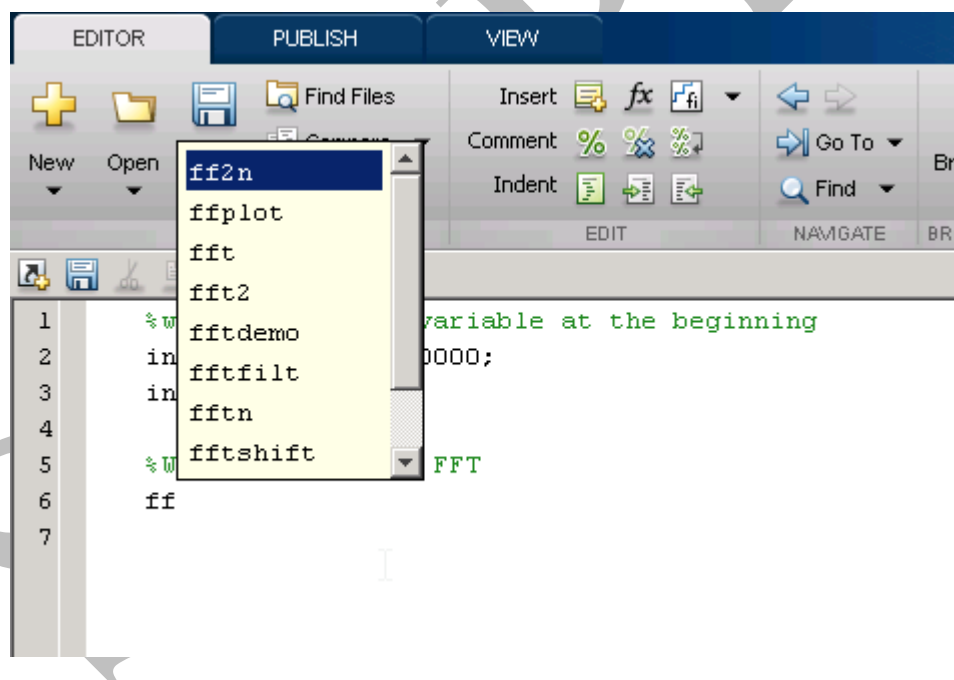
Since the last versions of MATLAB it seem that the Copyright and Author metadata doesn't work in the publishing tool anymore.

Autocompletion

If you want to use autocompletion in MATLAB begin to write the variable name/function you want and press the **TAB** button on your keyboard:



Also the same thing works for native functions:



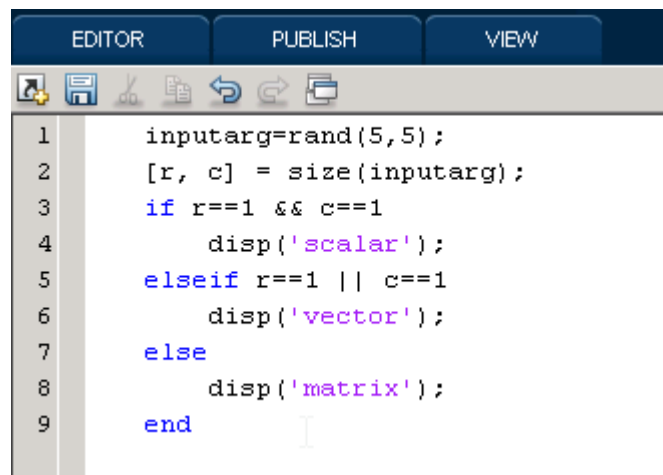
Move with the arrows of you keyboard and press **ENTER** to validate (or **TAB** again).

Conditional structure (if/switch)

Note: The purpose of this document is not to make an introduction to algorithmic. If you don't know how to make programming/scripts first follow an algorithmic course or read a book on the subject.

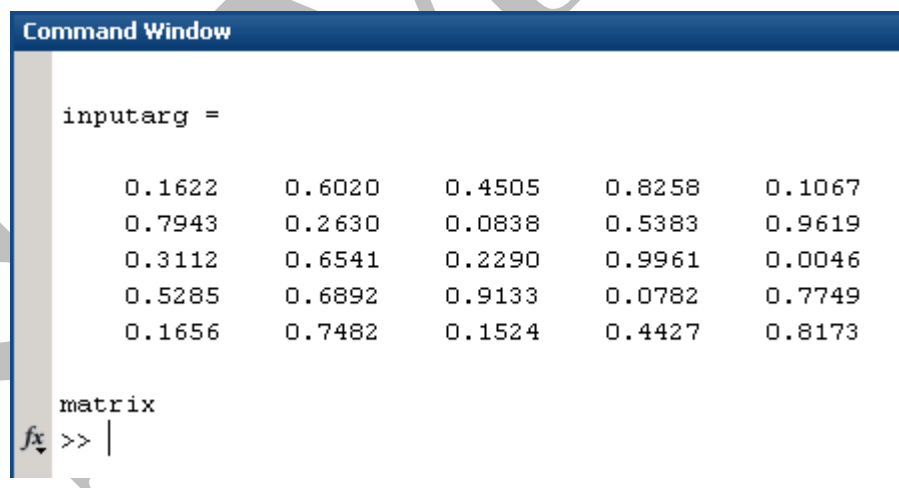
if statement

Let us see the most simple and complete example of an **If... Elseif...Else...End** MATLAB statement syntax with OR (||) and AND (&&) operators=



```
1 inputarg=rand(5,5);
2 [r, c] = size(inputarg);
3 if r==1 && c==1
4     disp('scalar');
5 elseif r==1 || c==1
6     disp('vector');
7 else
8     disp('matrix');
9 end
```

As you can see if we run the script (**Ctrl+Enter**), we get:



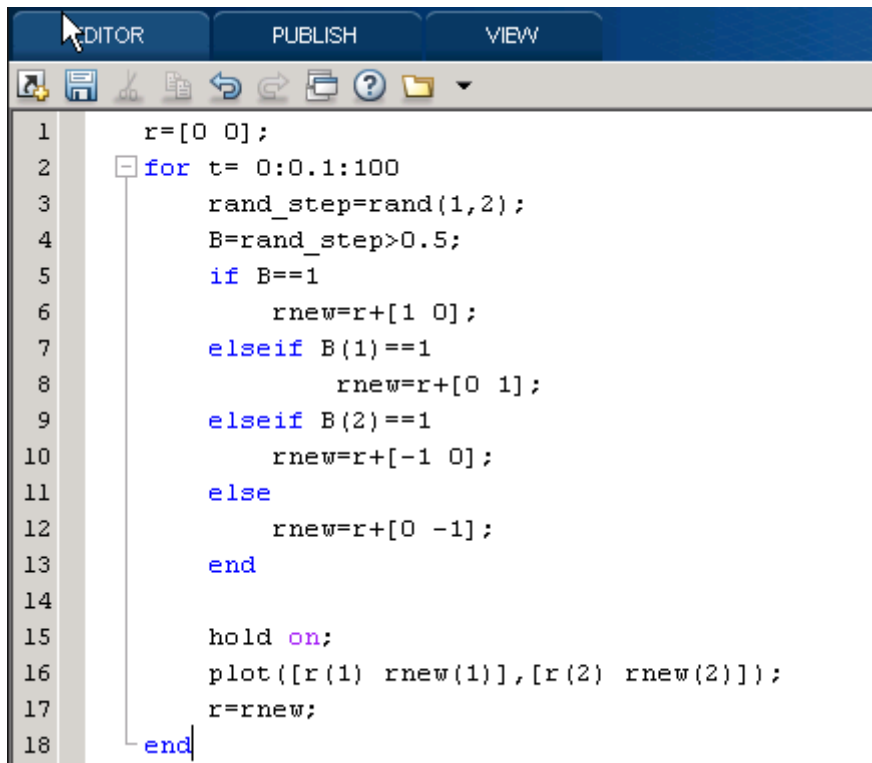
```
Command Window

inputarg =

    0.1622    0.6020    0.4505    0.8258    0.1067
    0.7943    0.2630    0.0838    0.5383    0.9619
    0.3112    0.6541    0.2290    0.9961    0.0046
    0.5285    0.6892    0.9133    0.0782    0.7749
    0.1656    0.7482    0.1524    0.4427    0.8173

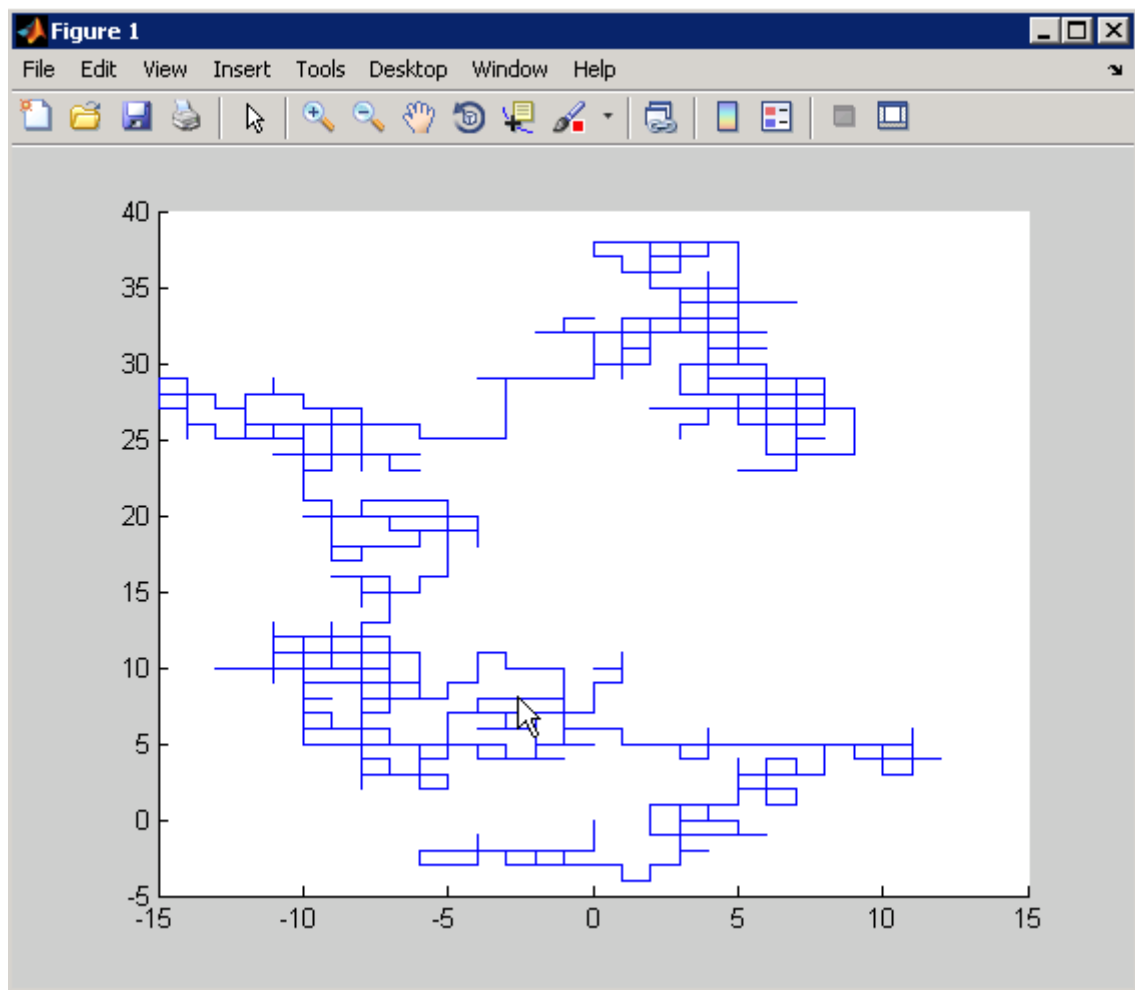
matrix
fx >> |
```

And well another well know good exercise for applying if statement is to plot a random walk (brownian motion):



```
1      r=[0 0];
2      for t= 0:0.1:100
3          rand_step=rand(1,2);
4          B=rand_step>0.5;
5          if B==1
6              rnew=r+[1 0];
7          elseif B(1)==1
8              rnew=r+[0 1];
9          elseif B(2)==1
10             rnew=r+[-1 0];
11          else
12             rnew=r+[0 -1];
13          end
14
15          hold on;
16          plot([r(1) rnew(1)], [r(2) rnew(2)]);
17          r=rnew;
18      end
```

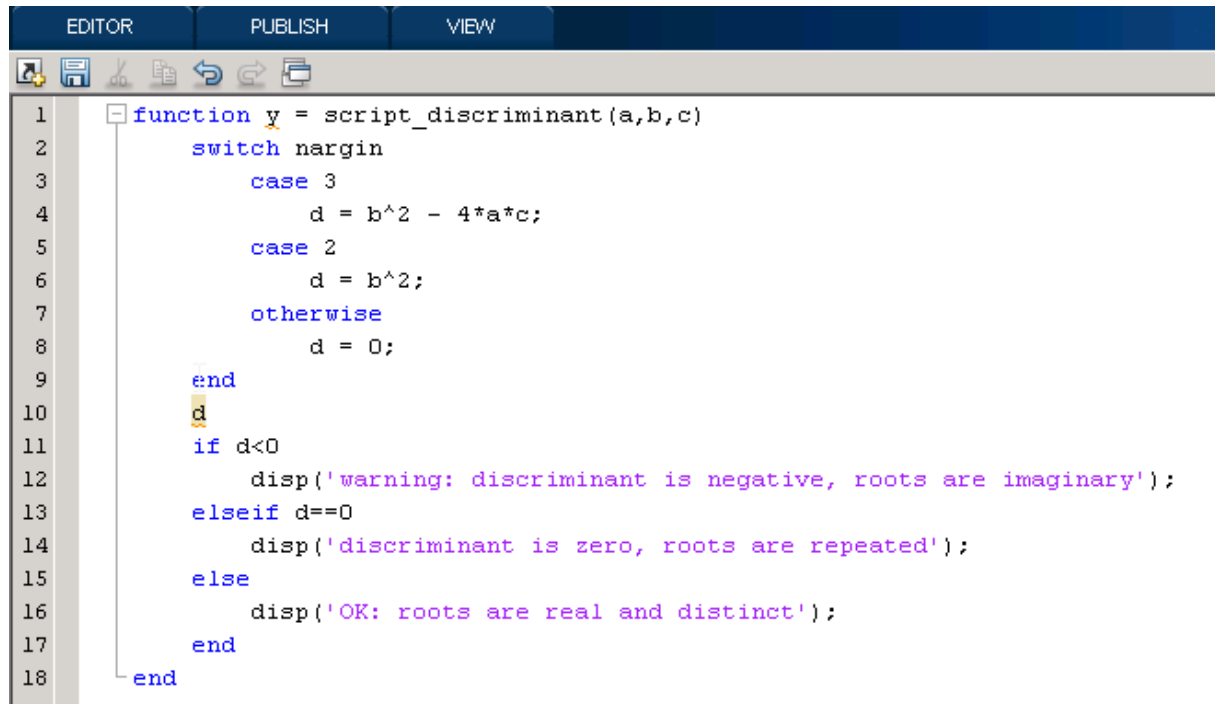
This will result in:



switch statement (with if...)

The **Switch...Case...Otherwise...End** statement is known to be faster as the If statement in most case and gives you also the possibility to have a more readable script.

Here is a very interesting example introducing at the same time the famous **nargin** MATLAB command:

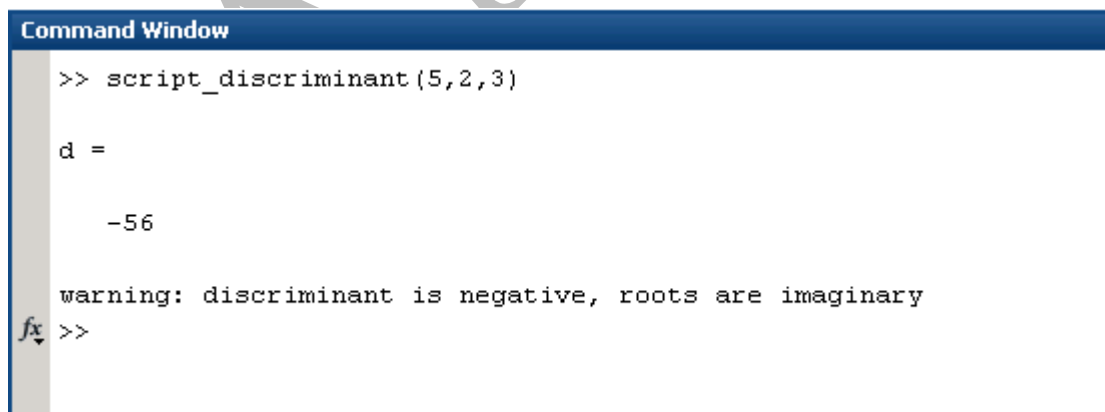


```

1  function y = script_discriminant(a,b,c)
2      switch nargin
3          case 3
4              d = b^2 - 4*a*c;
5          case 2
6              d = b^2;
7          otherwise
8              d = 0;
9      end
10     d
11     if d<0
12         disp('warning: discriminant is negative, roots are imaginary');
13     elseif d==0
14         disp('discriminant is zero, roots are repeated');
15     else
16         disp('OK: roots are real and distinct');
17     end
18 end

```

As you can see above, mix **Switch** with **If** in the same function/script is not a major issue (logic...). The result of this script if you call the function will be:



```

>> script_discriminant(5,2,3)

d =

    -56

warning: discriminant is negative, roots are imaginary
fx >>

```

Iteration structures

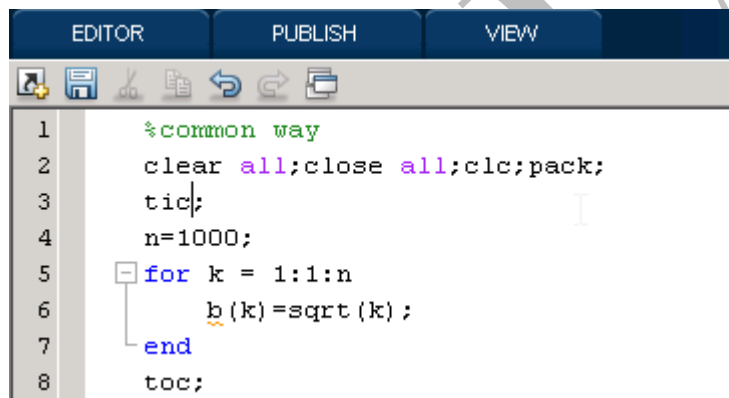
Iteration structures are well known in algorithmic and thus in any programming language. There is just a trap with MATLAB: **even if some MATLAB native functions facilitates the cleaning/treatment of vectors and matrices, sometimes the traditional For...End are faster if you test them with a tic... toc!**

You must also know that there is no **For each...** iteration structure in MATLAB (at least actually) and thus you can not iterate across objects collection.

For...End iteration structure (and optimization)

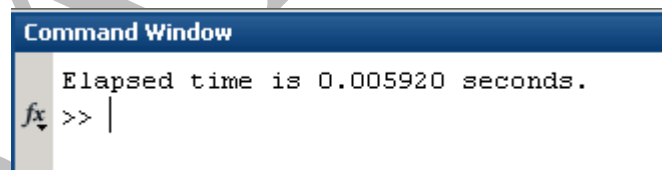
In MATLAB we almost always use loop as in other programming/script languages. But the fact that MATLAB is Matrix based sometimes changes the deal with classic programming/script languages.

Let us see below a very easy example that shows us this paradigm:



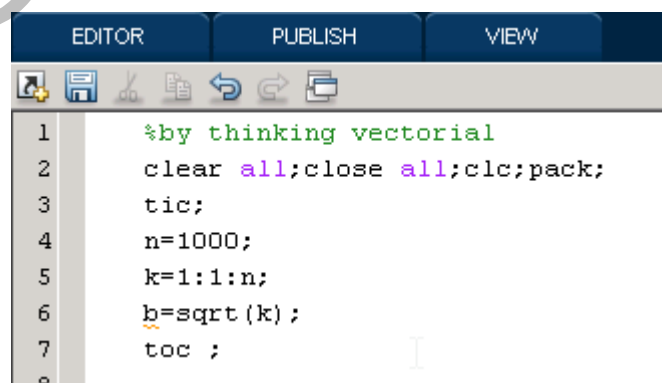
```
1 %common way
2 clear all;close all;clc;pack;
3 tic;
4 n=1000;
5 for k = 1:1:n
6     b(k)=sqrt(k);
7 end
8 toc;
```

and if we run the script (**Ctrl+Enter**) we get:



```
Command Window
Elapsed time is 0.005920 seconds.
fx >> |
```

Now if we write the same thing but in a "vectorial" way we have:



```
1 %by thinking vectorial
2 clear all;close all;clc;pack;
3 tic;
4 n=1000;
5 k=1:1:n;
6 b=sqrt(k);
7 toc ;
8
```


and if we run the script (**Ctrl+Enter**) we get:

```
Command Window
Elapsed time is 0.006796 seconds.
fx >>
```

Take care to the statistical bias of the interpretation of only one run! You have to run a code a thousand of times to have a statistical distribution of the running time and after do a Student-*t* test (or a non-parametric median test) to compare the efficiency of the code.

Now what happens if we preallocate the vectors=:

```
EDITOR PUBLISH VIEW
1 %better by pre-allocating
2 tic;
3 n=1000;
4 a=zeros(1,n);
5 b=zeros(1,n);
6 a = 1:1:n;
7 b = sqrt(a);
8 toc;
```

and if we run the script (**Ctrl+Enter**) we get:

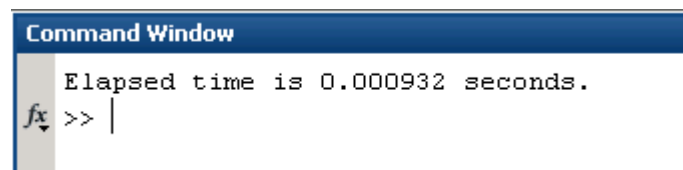
```
Command Window
Elapsed time is 0.008364 seconds.
fx >> |
```

Seems to be much better...

And what happens if we create empty vectors only?:

```
EDITOR PUBLISH VIEW
1 %even better than befor by pre-allocating with empty vectors
2 clear all;close all;clc;pack;
3 tic ;
4 n=1000;|
5 a=[];
6 b=[];
7 a = 1 : 1 : n ;
8 b = sqrt(a) ;
9 toc ;
10
```

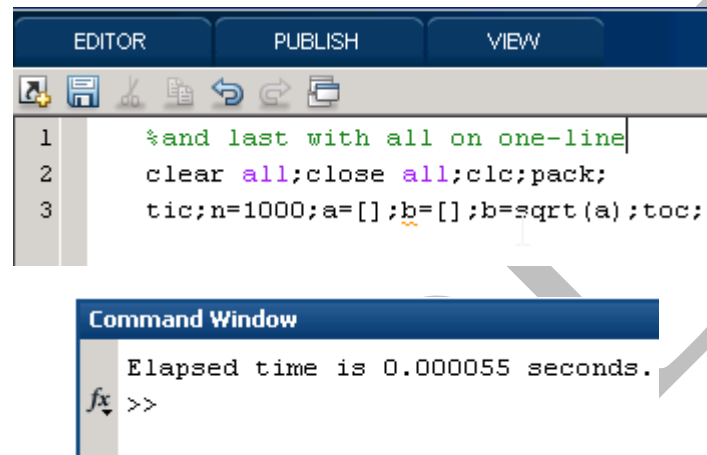
and if we run the script (**Ctrl+Enter**) we get:



```
Command Window
Elapsed time is 0.000932 seconds.
fx >> |
```

wouah! Not to bad...

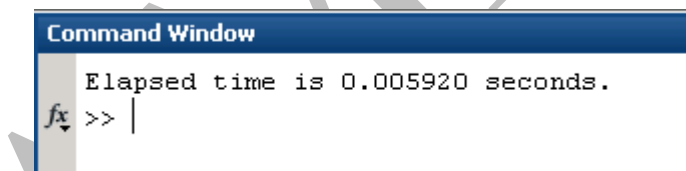
And what happens if we write a maximum of stuff only on one line?:



```
EDITOR PUBLISH VIEW
1 %and last with all on one-line
2 clear all;close all;clc;pack;
3 tic;n=1000;a=[];b=[];b=sqrt(a);toc;

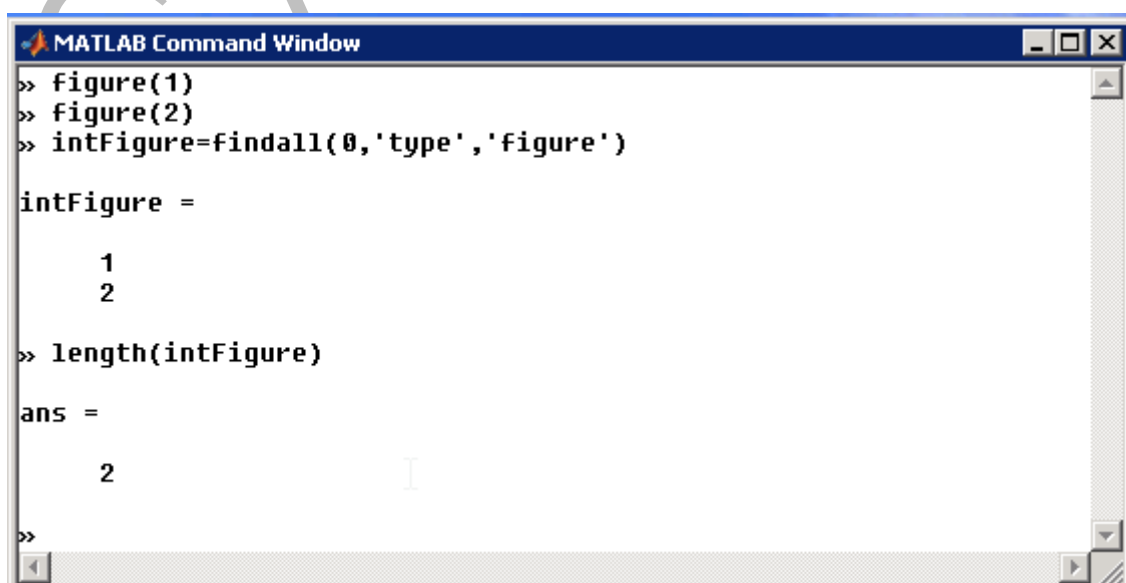
Command Window
Elapsed time is 0.000055 seconds.
fx >>
```

wouhahah! In comparison with the first script we are **1000 times faster!**



```
Command Window
Elapsed time is 0.005920 seconds.
fx >> |
```

A last remark... If you want to loop on chart figures you will sometimes need to know how much figure are open. For this purpose you can use the following syntax:



```
MATLAB Command Window
>> figure(1)
>> figure(2)
>> intFigure=findall(0,'type','figure')

intFigure =

     1
     2

>> length(intFigure)

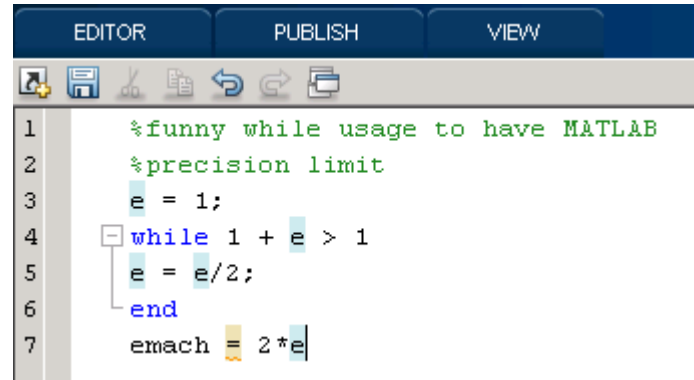
ans =

     2

>>
```

While...End iteration structure

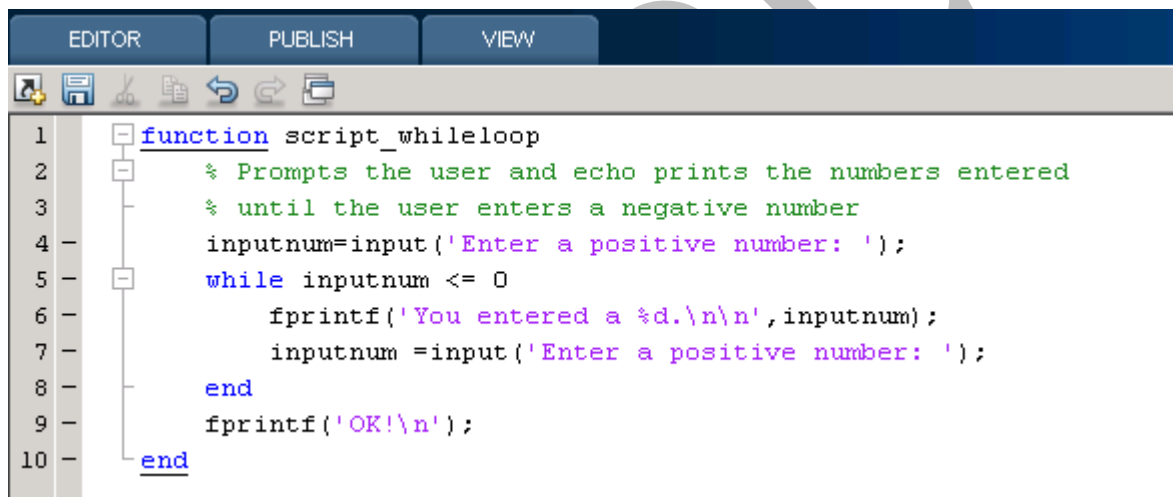
For...End iteration structures are not always adapted to all situations. Before that we see such a real example let us see the syntax of a **While...End** structure with a funny well know example of the precision limit on any programming/script language:



```

1      %funny while usage to have MATLAB
2      %precision limit
3      e = 1;
4      while 1 + e > 1
5          e = e/2;
6      end
7      emach = 2*e
  
```

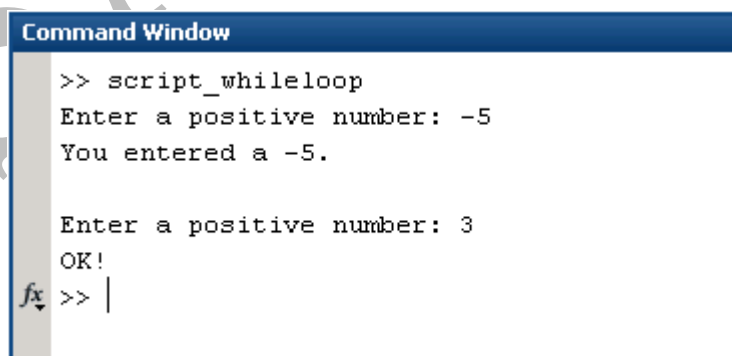
Or the very common case when you don't have developed a GUI (Graphic User Interface):



```

1      function script_whileloop
2      % Prompts the user and echo prints the numbers entered
3      % until the user enters a negative number
4      inputnum=input('Enter a positive number: ');
5      while inputnum <= 0
6          fprintf('You entered a %d.\n\n',inputnum);
7          inputnum =input('Enter a positive number: ');
8      end
9      fprintf('OK!\n');
10     end
  
```

If we call the function we get something working well:



```

Command Window

>> script_whileloop
Enter a positive number: -5
You entered a -5.

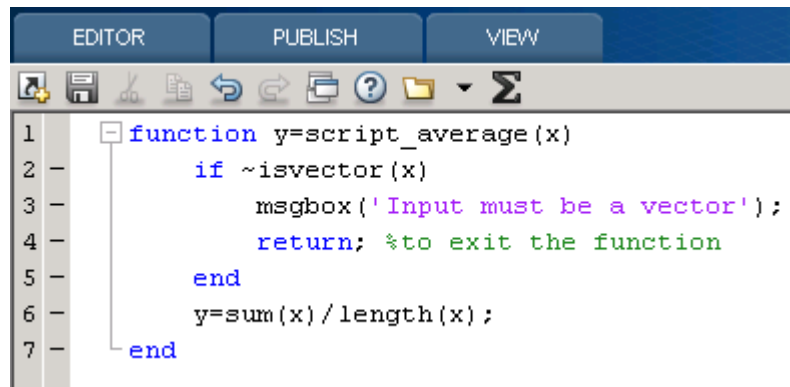
Enter a positive number: 3
OK!
fx >> |
  
```

Functions (functions files)

In some example seen before you have certainly noted that when you create a function, **the *.m file must have the same name as the function itself!**

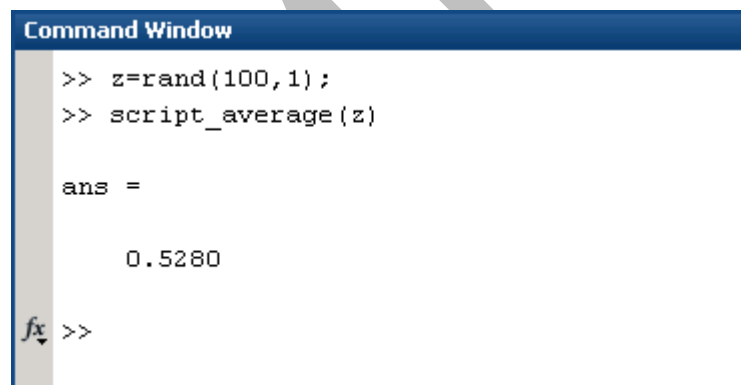
One input, One output (OI00)

Now write one of the most simple example that you can imagine with a function: something that takes in input only one information and that returns also only one output:



```
1 function y=script_average(x)
2     if ~isvector(x)
3         msgbox('Input must be a vector');
4         return; %to exit the function
5     end
6     y=sum(x)/length(x);
7 end
```

If you call the function you simply get:



```
Command Window

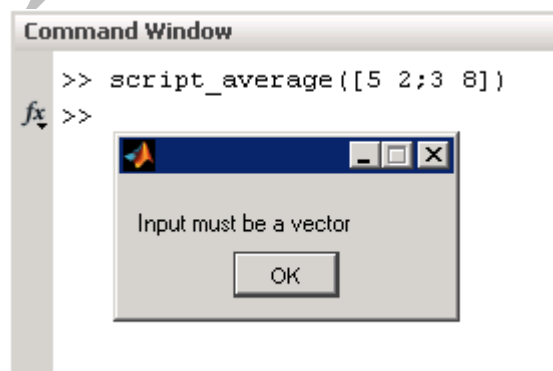
>> z=rand(100,1);
>> script_average(z)

ans =

    0.5280

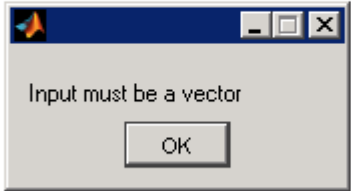
fx >>
```

Nothing new! And if generate an error:



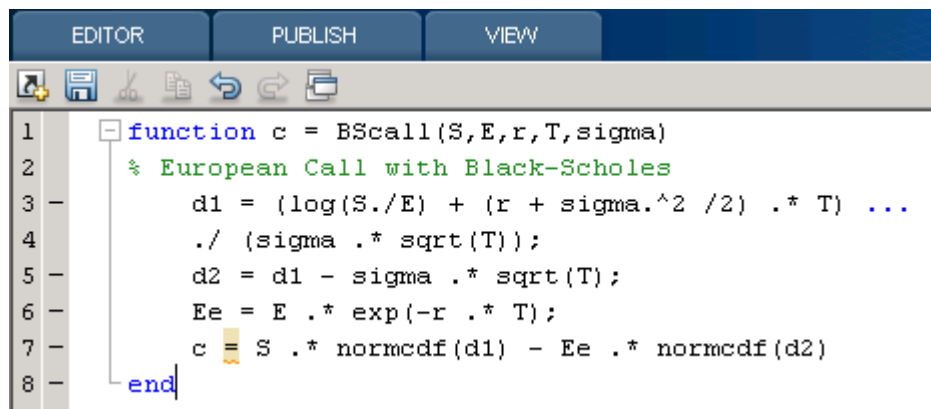
```
Command Window

>> script_average([5 2;3 8])
fx >>
```



Multiple inputs, One output (M100)

The level just above of the previous example is to have multiple inputs and to return one output. Here is also a well know worldwide example in Finance of such a situation:

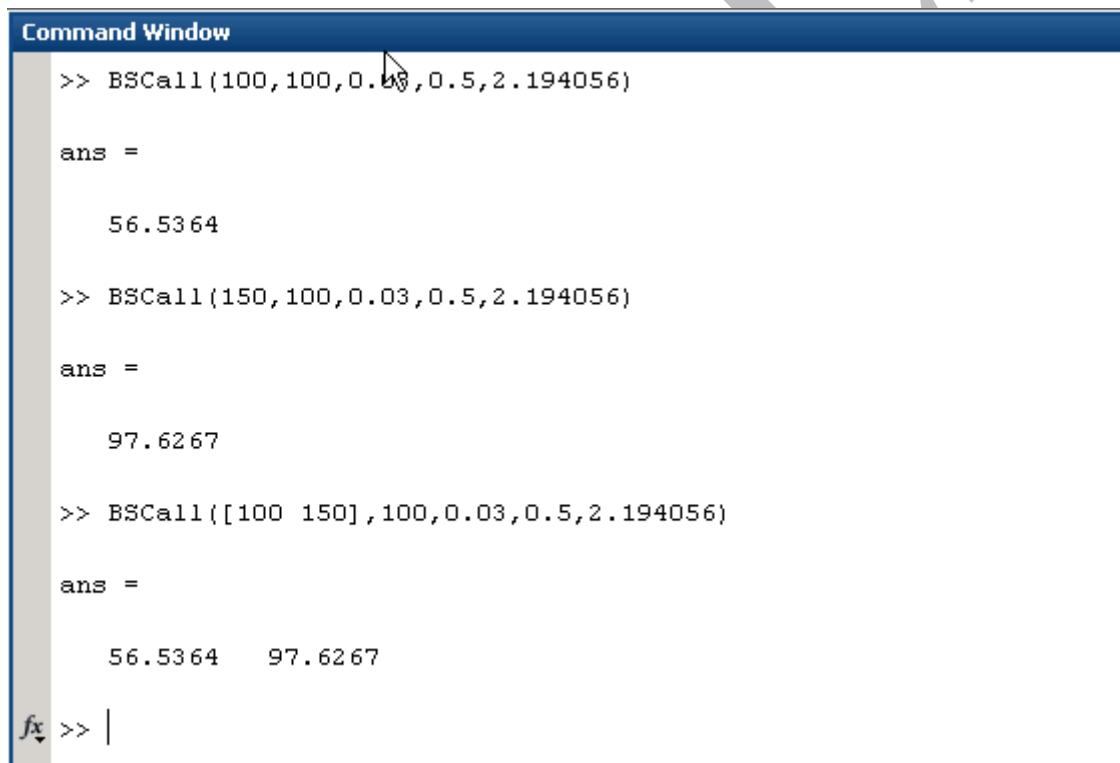


```

1 function c = BScall(S,E,r,T,sigma)
2     % European Call with Black-Scholes
3     d1 = (log(S./E) + (r + sigma.^2 /2) .* T) ...
4         ./ (sigma .* sqrt(T));
5     d2 = d1 - sigma .* sqrt(T);
6     Ee = E .* exp(-r .* T);
7     c = S .* normcdf(d1) - Ee .* normcdf(d2)
8 end

```

So we get:



```

>> BScall(100,100,0.03,0.5,2.194056)

ans =

    56.5364

>> BScall(150,100,0.03,0.5,2.194056)

ans =

    97.6267

>> BScall([100 150],100,0.03,0.5,2.194056)

ans =

    56.5364    97.6267

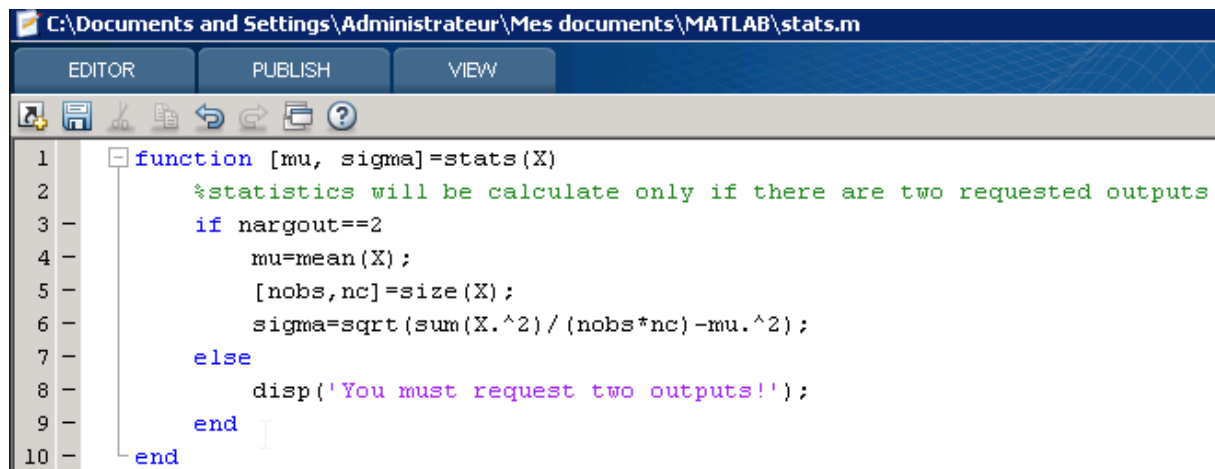
fx >> |

```

As you can see, the inputs can also be vectors!

One input, Multiple outputs (OIMO)

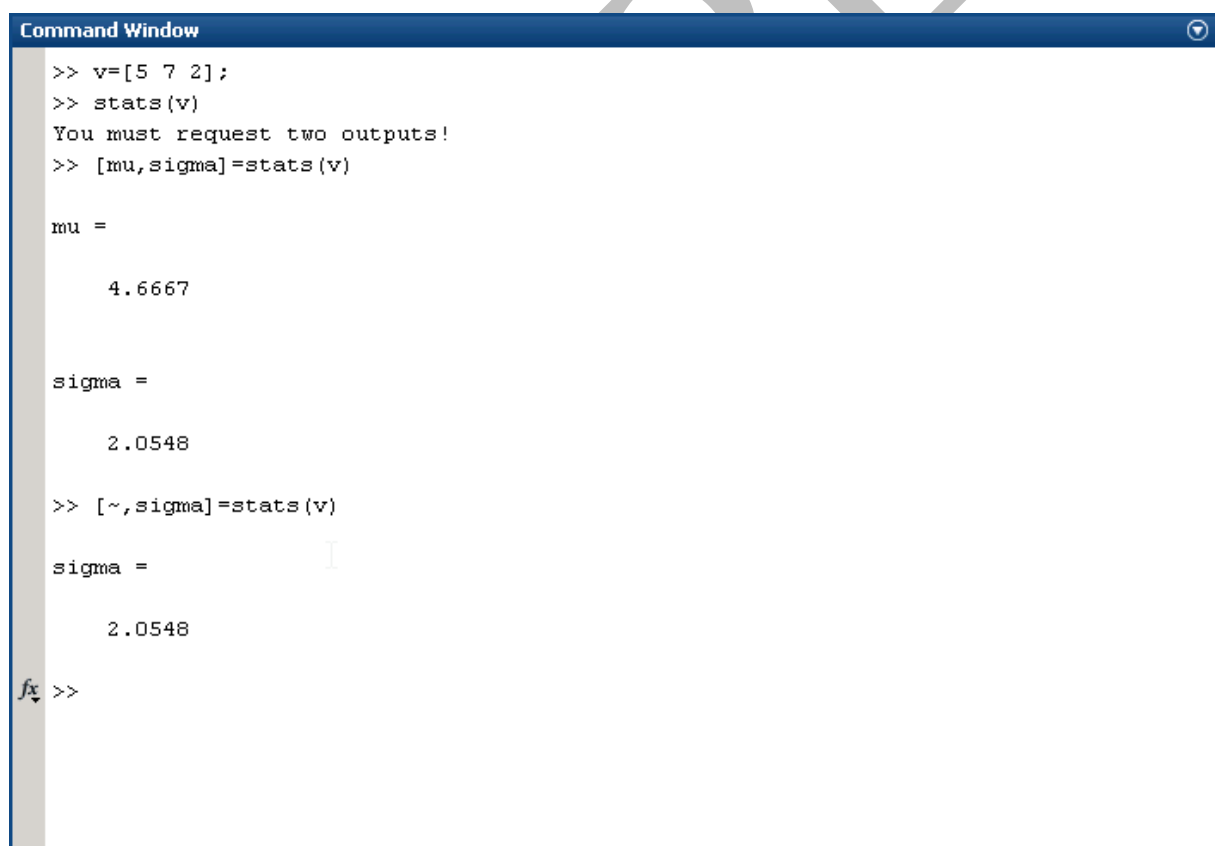
This is also a famous example introducing the **nargout** MATLAB command that detects how many outputs the user request for the function:



```

C:\Documents and Settings\Administrateur\Mes documents\MATLAB\stats.m
EDITOR PUBLISH VIEW
1 function [mu, sigma]=stats(X)
2     %statistics will be calculate only if there are two requested outputs
3     if nargout==2
4         mu=mean(X);
5         [nobs,nc]=size(X);
6         sigma=sqrt(sum(X.^2)/(nobs*nc)-mu.^2);
7     else
8         disp('You must request two outputs!');
9     end
10 end
  
```

The result will be:



```

Command Window
>> v=[5 7 2];
>> stats(v)
You must request two outputs!
>> [mu,sigma]=stats(v)

mu =

    4.6667

sigma =

    2.0548

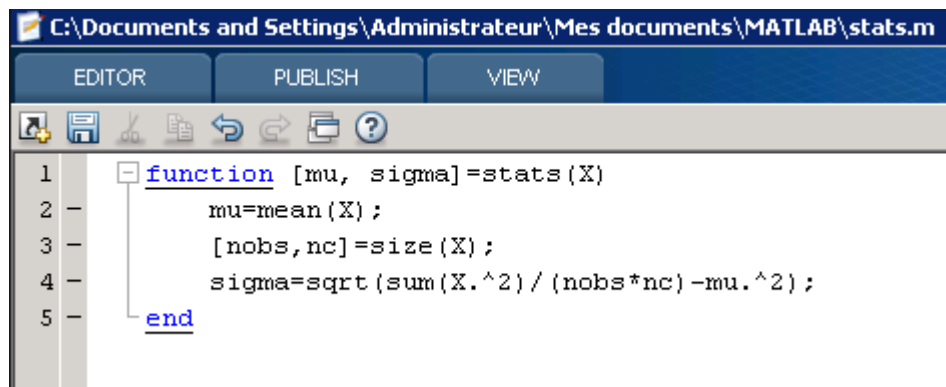
>> [~,sigma]=stats(v)

sigma =

    2.0548

fx >>
  
```

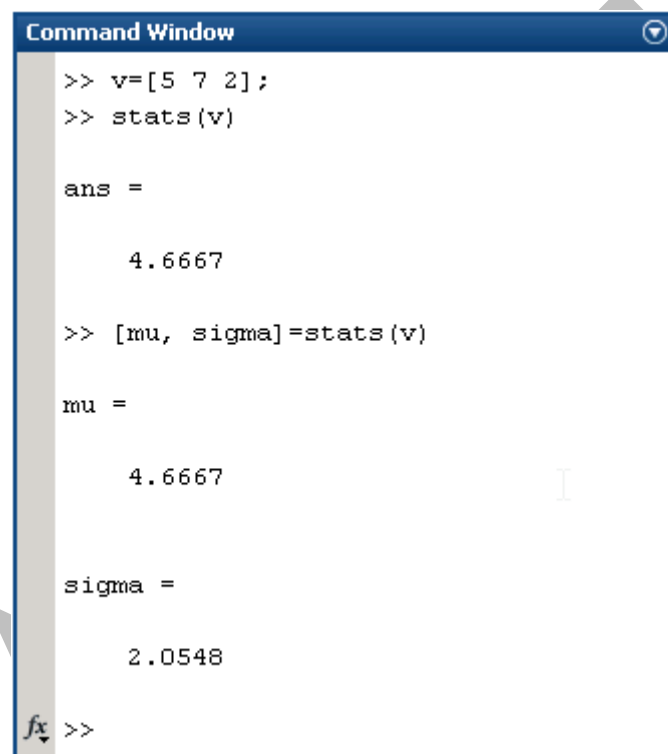
And if we simplify the code:



The image shows a MATLAB Editor window titled 'C:\Documents and Settings\Administrateur\Mes documents\MATLAB\stats.m'. The window has tabs for 'EDITOR', 'PUBLISH', and 'VIEW'. Below the tabs is a toolbar with icons for file operations and help. The main area contains the following code:

```
1 function [mu, sigma]=stats(X)
2     mu=mean(X);
3     [nobs,nc]=size(X);
4     sigma=sqrt(sum(X.^2)/(nobs*nc)-mu.^2);
5 end
```

We get:



The image shows a MATLAB Command Window with the following session:

```
>> v=[5 7 2];
>> stats(v)

ans =

    4.6667

>> [mu, sigma]=stats(v)

mu =

    4.6667

sigma =

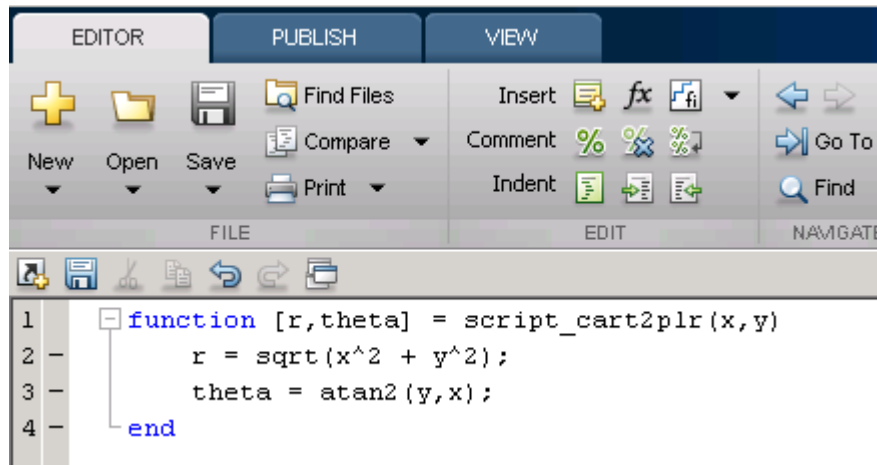
    2.0548

fx >>
```

Nothing very suprising.

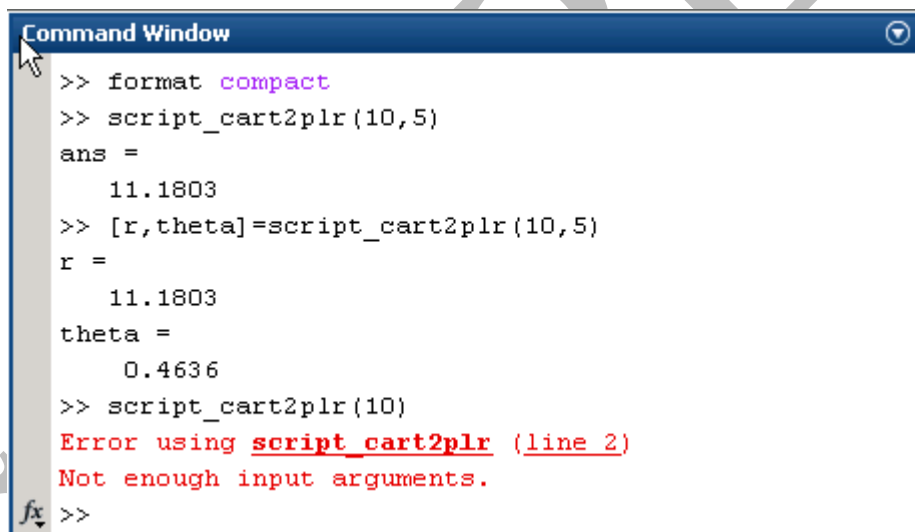
Multiple inputs, Multiple outputs (MIMO)

The way to use multiple inputs to get multiple output is by extension very easy. Consider the following well know academic example:



```
1 function [r,theta] = script_cart2plr(x,y)
2     r = sqrt(x^2 + y^2);
3     theta = atan2(y,x);
4 end
```

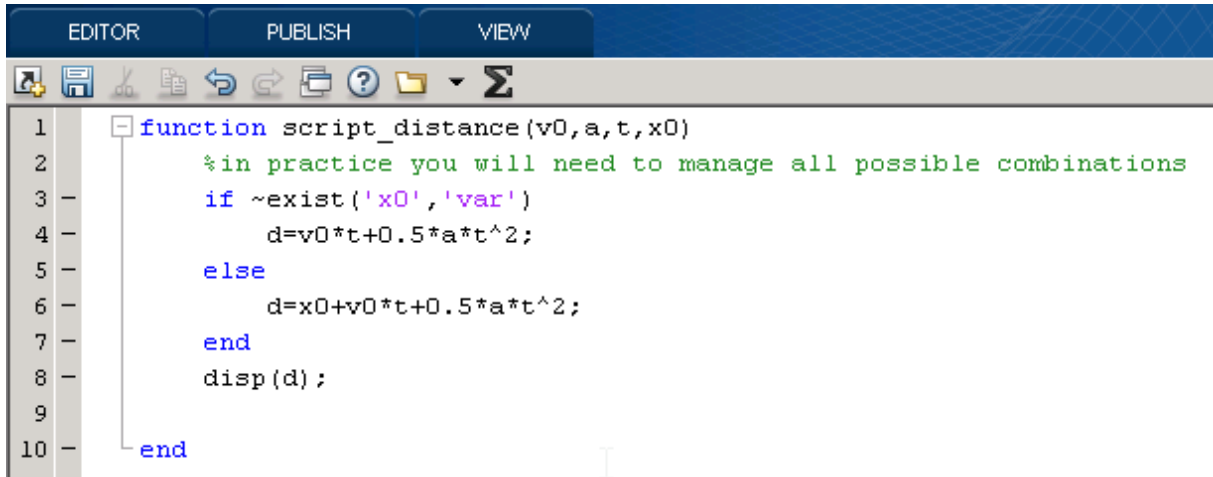
The result will be:



```
>> format compact
>> script_cart2plr(10,5)
ans =
    11.1803
>> [r,theta]=script_cart2plr(10,5)
r =
    11.1803
theta =
     0.4636
>> script_cart2plr(10)
Error using script_cart2plr (line 2)
Not enough input arguments.
fx >>
```

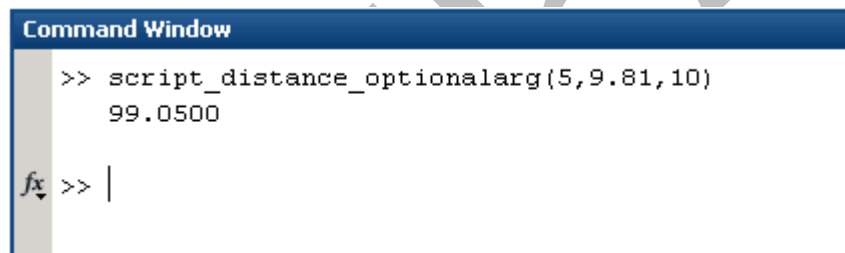

Optional arguments

We saw before that we can use **nargin** and **nargout** to manage/detect respectively input and output arguments. If you don't like how this both functions work you can use instead the function **exist()**:



```
1 function script_distance(v0,a,t,x0)
2     %in practice you will need to manage all possible combinations
3     if ~exist('x0','var')
4         d=v0*t+0.5*a*t^2;
5     else
6         d=x0+v0*t+0.5*a*t^2;
7     end
8     disp(d);
9
10 end
```

As you can see, if we forget the fourth argument the function still work:



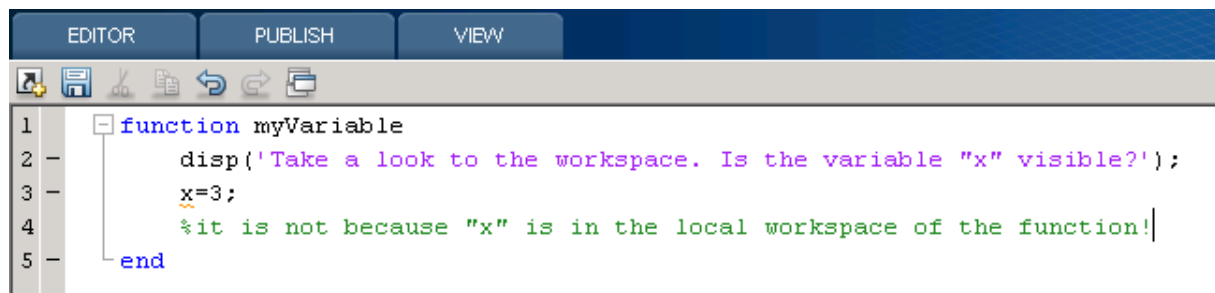
```
Command Window
>> script_distance_optionalarg(5,9.81,10)
99.0500
fx >> |
```

Variables scope

Most variables in the majority of script are local but in some special case it is useful to manage the scopes of variables.

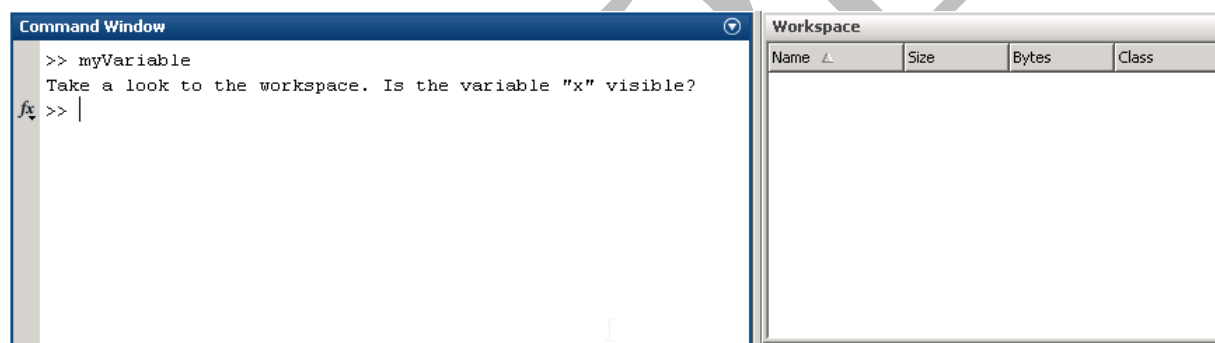
Global vs Local variables

The first example is to see what happens when you create a variable into a function:



```
1 function myVariable
2     disp('Take a look to the workspace. Is the variable "x" visible?');
3     x=3;
4     %it is not because "x" is in the local workspace of the function!
5 end
```

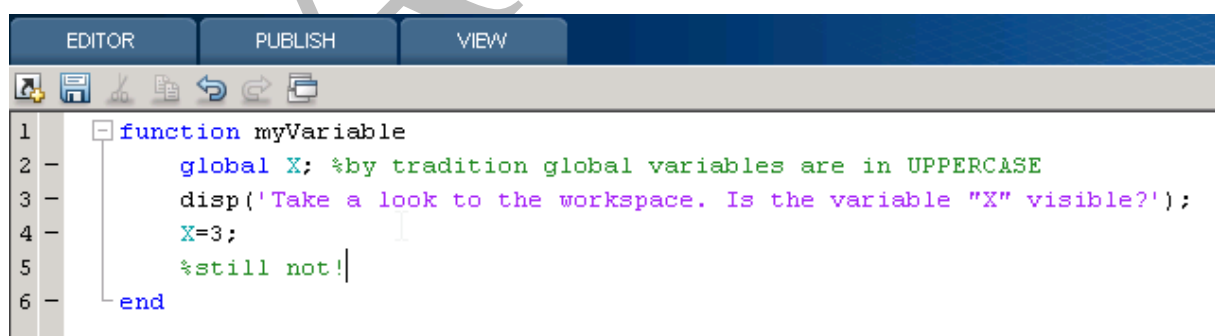
You will see that the variable is hopefully by default not visible in the MATLAB workspace. This is why we call such a variable a "ultra-local variable":



```
>> myVariable
Take a look to the workspace. Is the variable "x" visible?
fx >> |
```

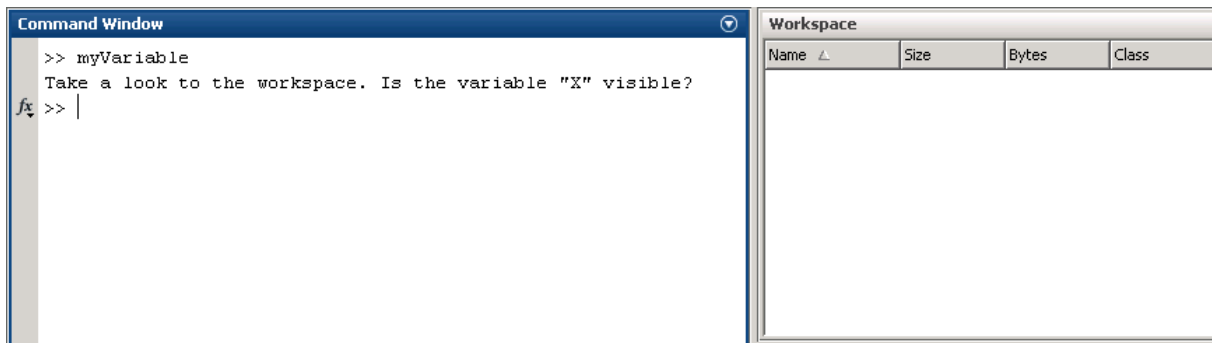
Name	Size	Bytes	Class
------	------	-------	-------

Now see what happens if we declare the variable using the statement **global**:

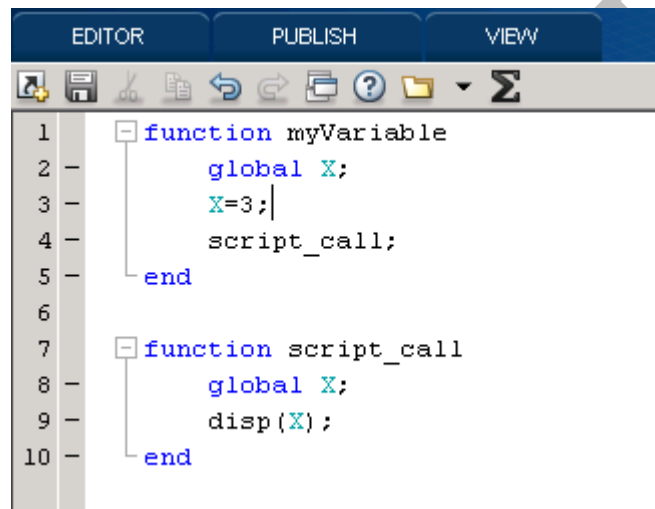


```
1 function myVariable
2     global X; %by tradition global variables are in UPPERCASE
3     disp('Take a look to the workspace. Is the variable "X" visible?');
4     X=3;
5     %still not!
6 end
```

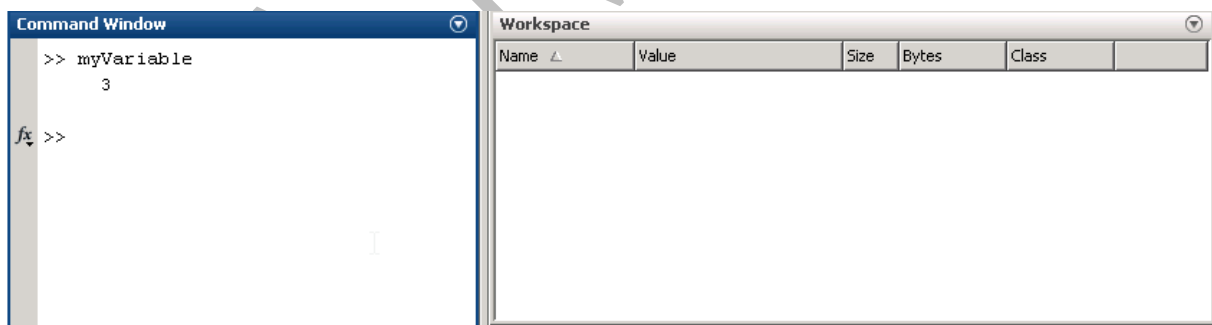
As you can see the variable is still (hopefully) not visible in the MATLAB workspace:



Then what is the purpose? The purpose is that the variable value can still be accessed by another function:

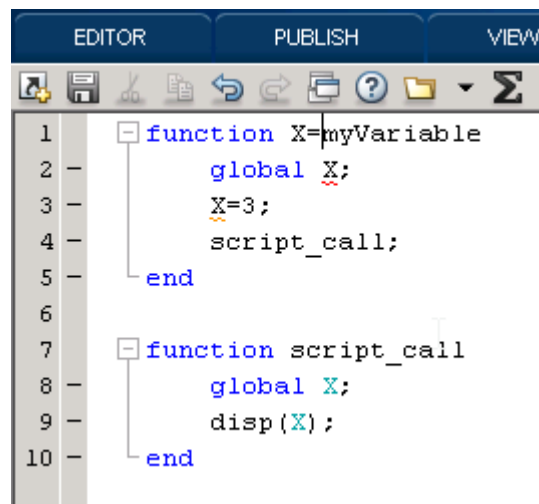


And we get:



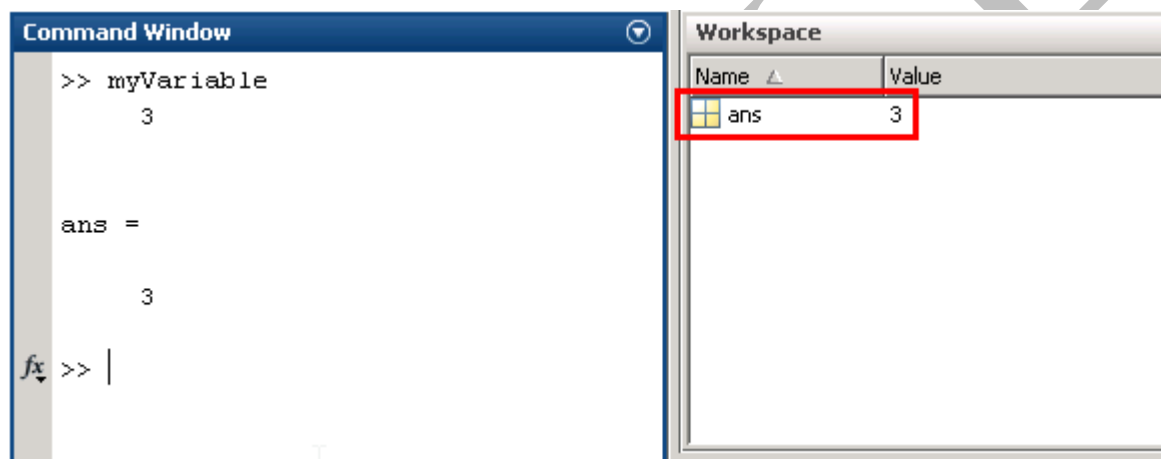
Then, what we call a "global variable" is a variable that is global at the level of functions workspace but will by default never be visible in the MATLAB workspace for security purposes (to not erase variables having the same name).

If you want you can change the script as following:



```
1  function X=myVariable
2      global X;
3      X=3;
4      script_call;
5  end
6
7  function script_call
8      global X;
9      disp(X);
10 end
```

Then when you run it you will get:



Command Window

```
>> myVariable
3

ans =

3

fx >> |
```

Workspace

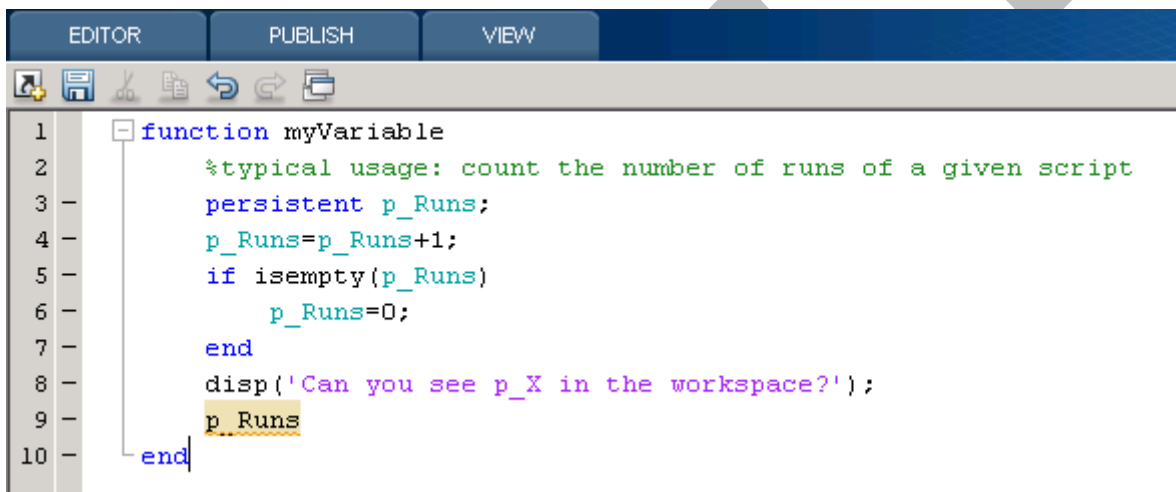
Name	Value
ans	3

Persistent variables

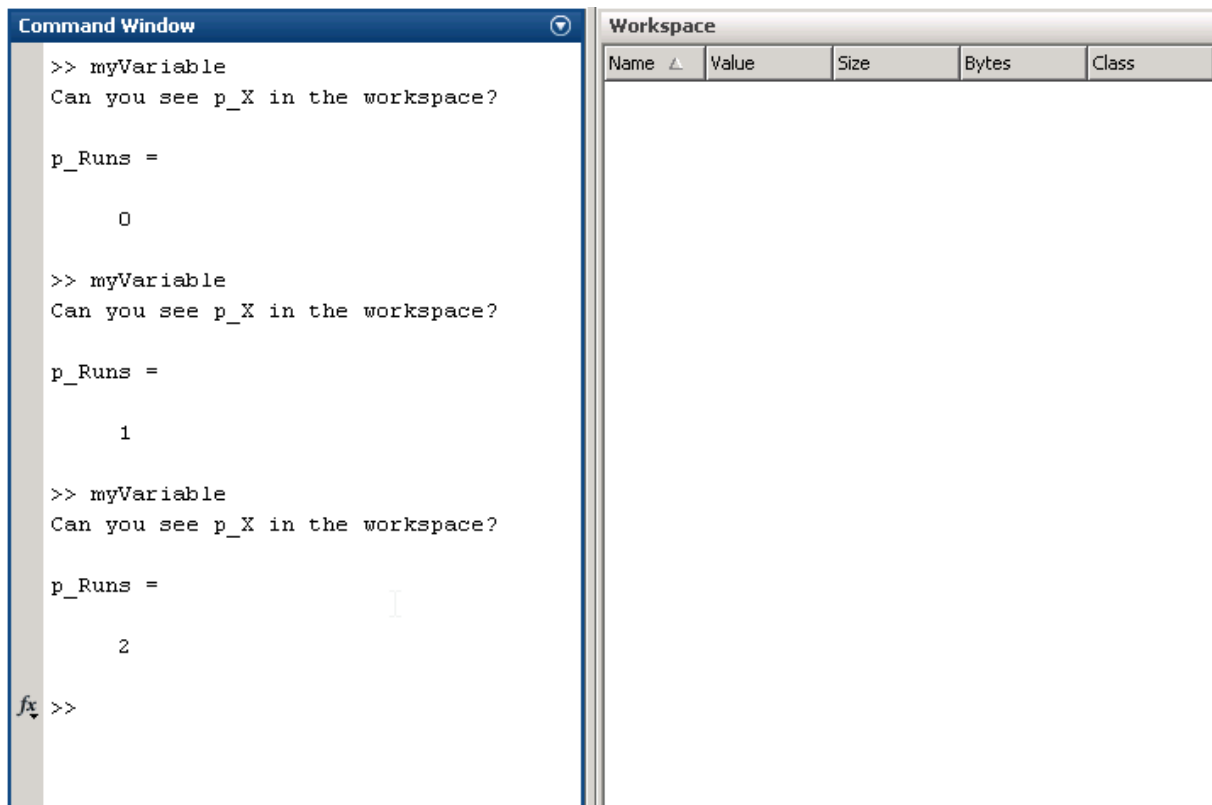
Some comments are in order:

1. You can declare and use persistent variables in a M-file function only (**this is on of the main difference with a Global Variable that can be modified by the user in the MATLAB GUI**).
2. Only the function in which the persistent variable is declared is allowed access to the variable (this is specific to persistent variables). At the opposite of a Global variable that is Global AND persistent
3. When the function exits, MATLAB does not clear the persistent variable from memory, so it retains its value from one function call to the next (this is the same behavior as global variables)

To declare a persistent variable we use the statement **persistent**:



```
1  function myVariable
2      %typical usage: count the number of runs of a given script
3      persistent p_Runs;
4      p_Runs=p_Runs+1;
5      if isempty(p_Runs)
6          p_Runs=0;
7      end
8      disp('Can you see p_X in the workspace?');
9      p_Runs
10 end
```

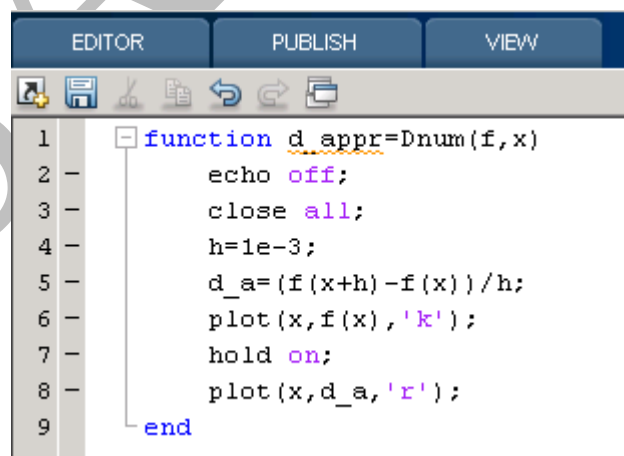


As you can see below, the variable is never visible in the MATLAB workspace!

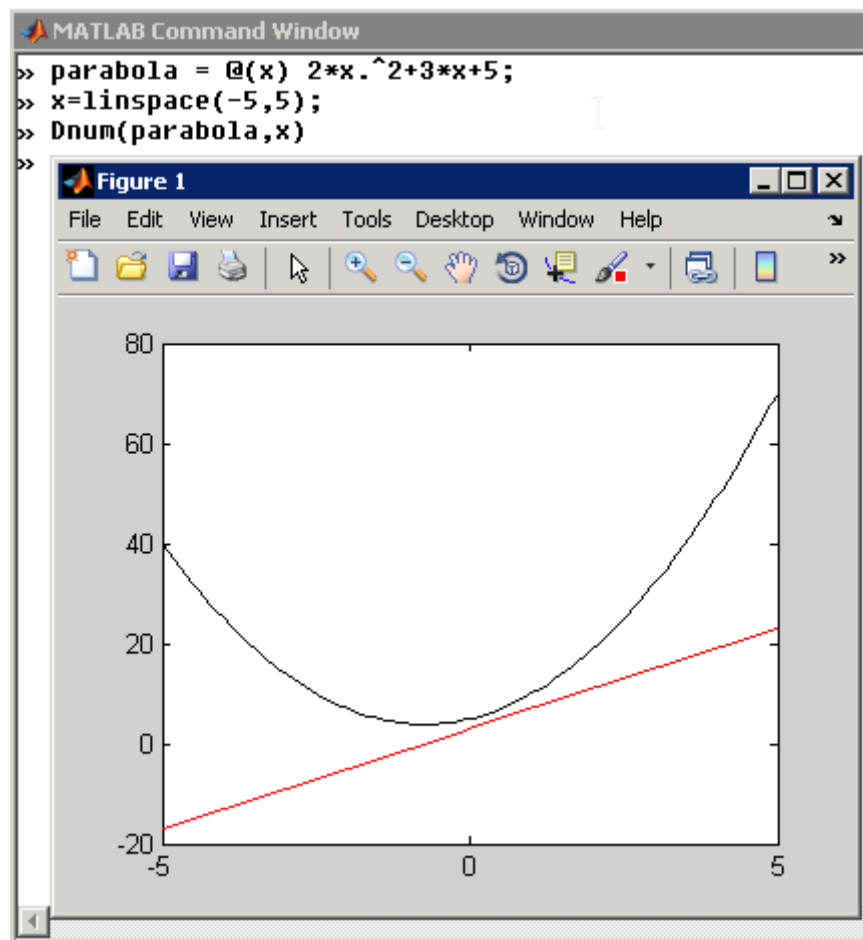
Anonymous function as input, chart output

One very interesting and important thing to know is that you can also pass functions as argument.

Consider the following simple example that just calculate the derivate of any function in a range defined by an input vector:



Then we can call this function as following using the anonymous @ function statement:

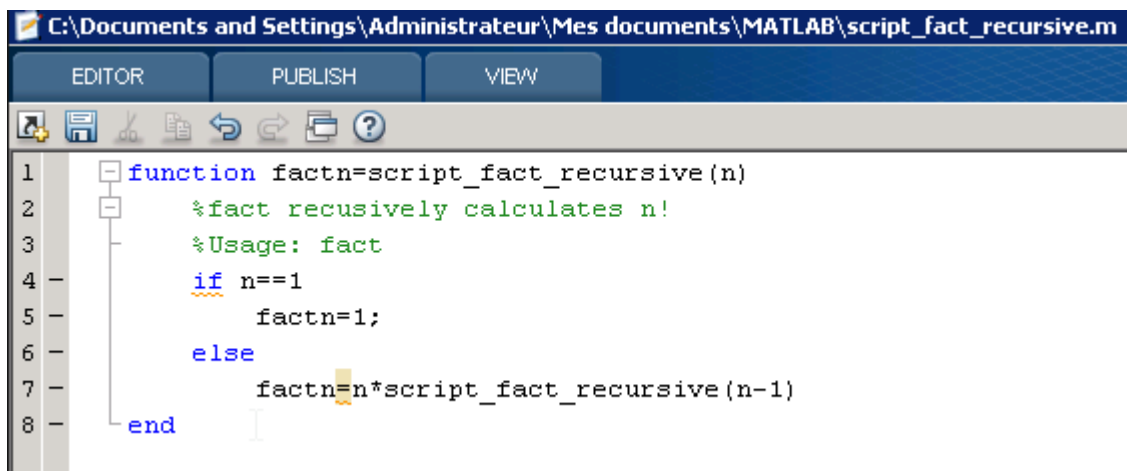


Works well :-)

Recursive functions

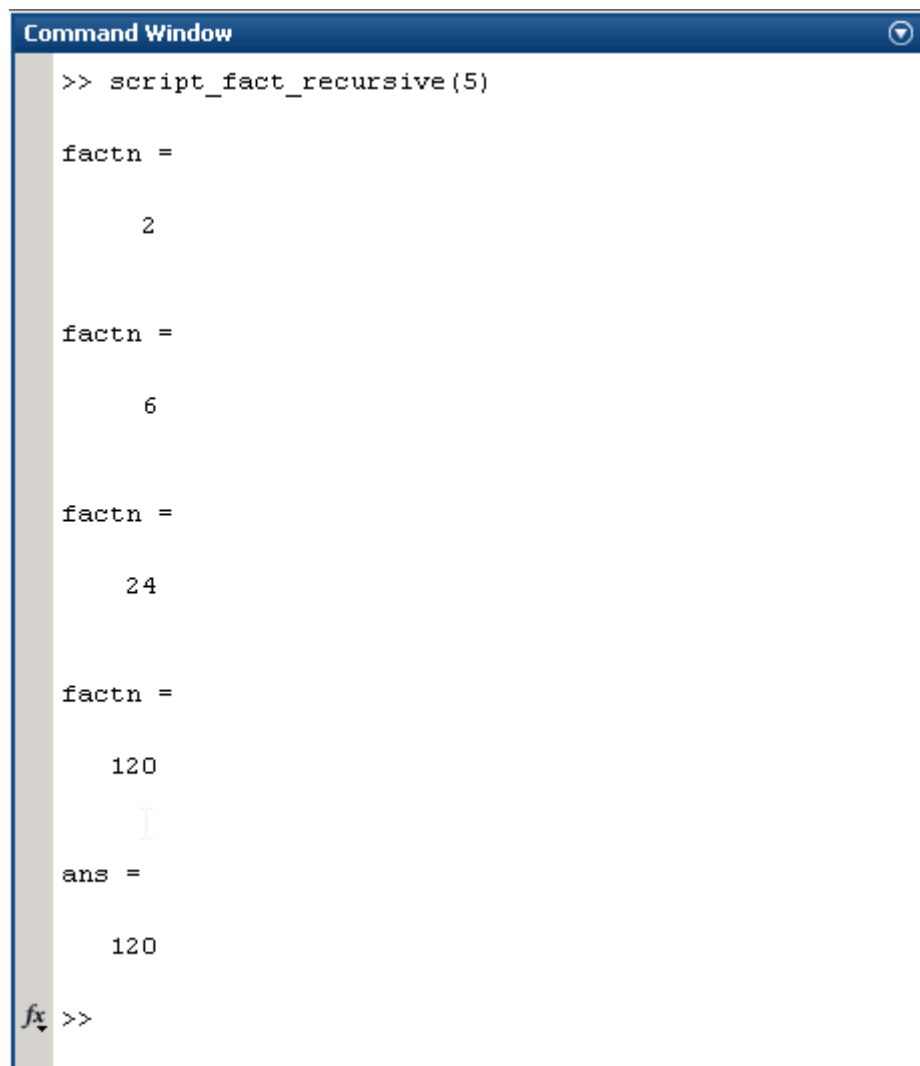
Another useful type of function (especially in finance for example with binomial/trinomial trees and Data Mining for hierarchical clustering) are recursive function that call themselves to compute the next values.

As example see the famous academic example below where we calculate the factorial using recursivity:



```
C:\Documents and Settings\Administrateur\Mes documents\MATLAB\script_fact_recursive.m
EDITOR PUBLISH VIEW
1 function factn=script_fact_recursive(n)
2     %fact recusively calculates n!
3     %Usage: fact
4     if n==1
5         factn=1;
6     else
7         factn=n*script_fact_recursive(n-1)
8     end
```

Thus we get:



The image shows a MATLAB Command Window with a blue title bar labeled "Command Window". The window contains the following text: a command prompt ">>" followed by the function call "script_fact_recursive(5)", and then four lines of output. Each line shows a variable "factn" being assigned a value: 2, 6, 24, and 120. The final line shows the variable "ans" assigned the value 120. At the bottom, there is a prompt "fx >>" with a small cursor icon. A large, light gray watermark "SA" is visible across the lower half of the image.

```
>> script_fact_recursive(5)

factn =

     2

factn =

     6

factn =

    24

factn =

   120

ans =

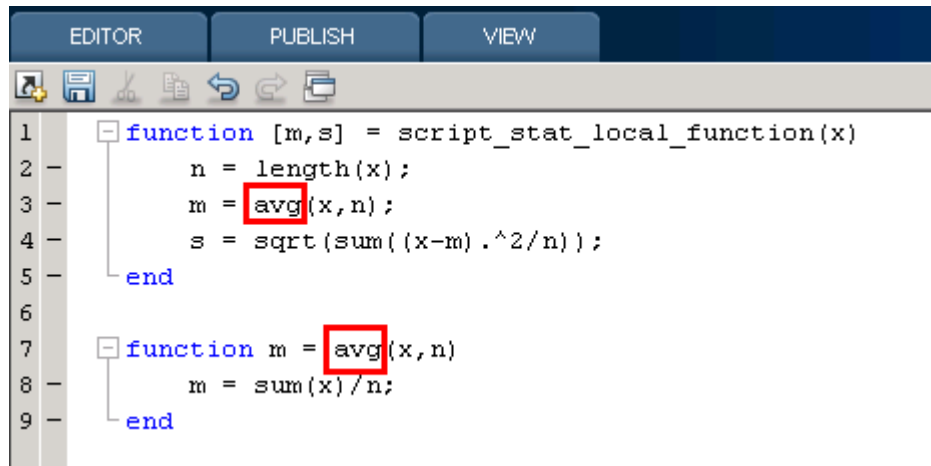
   120

fx >>
```

Local functions (multiple functions in one script)

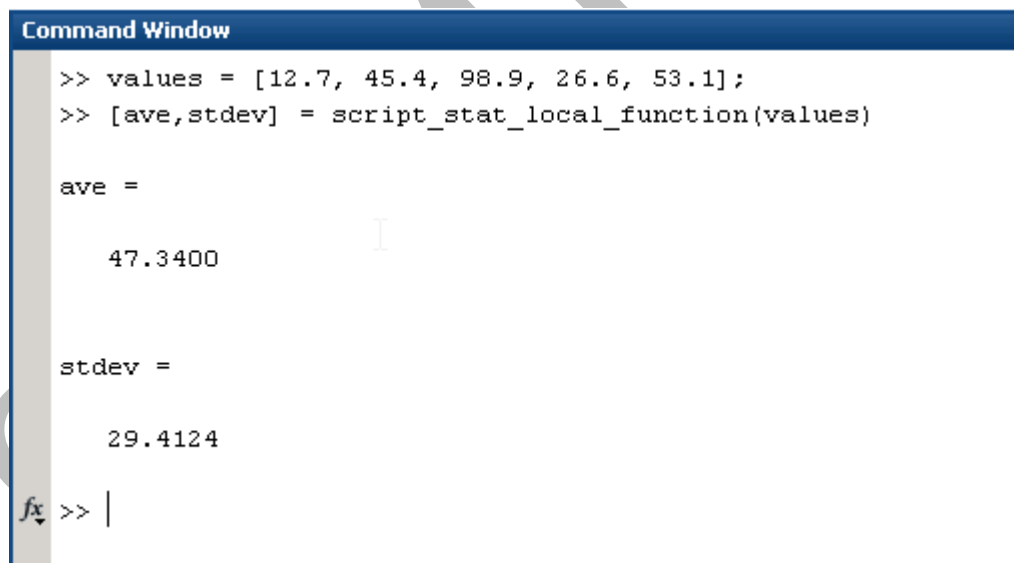
As in many scripting/programming languages it must be possible in a same M-file to have multiple functions.

As you can see this works well but the **only one condition** is that the main callable function to be at the top of the script:



```
1 function [m,s] = script_stat_local_function(x)
2     n = length(x);
3     m = avg(x,n);
4     s = sqrt(sum((x-m).^2/n));
5 end
6
7 function m = avg(x,n)
8     m = sum(x)/n;
9 end
```

As you can see this works well:



```
Command Window
>> values = [12.7, 45.4, 98.9, 26.6, 53.1];
>> [ave,stdev] = script_stat_local_function(values)

ave =

    47.3400

stdev =

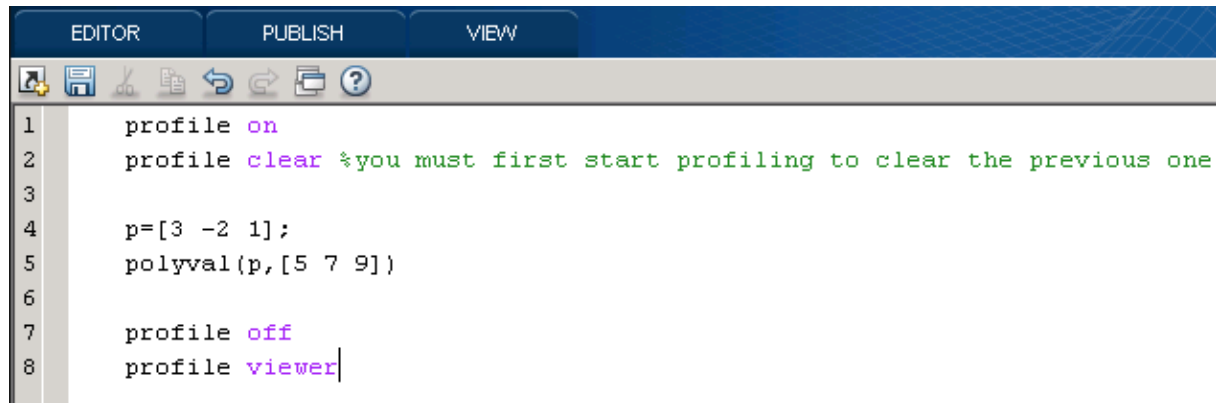
    29.4124

fx >> |
```

Profile a script

Profiling a script is a very useful functionality when you want to see where your code has low performance pieces.

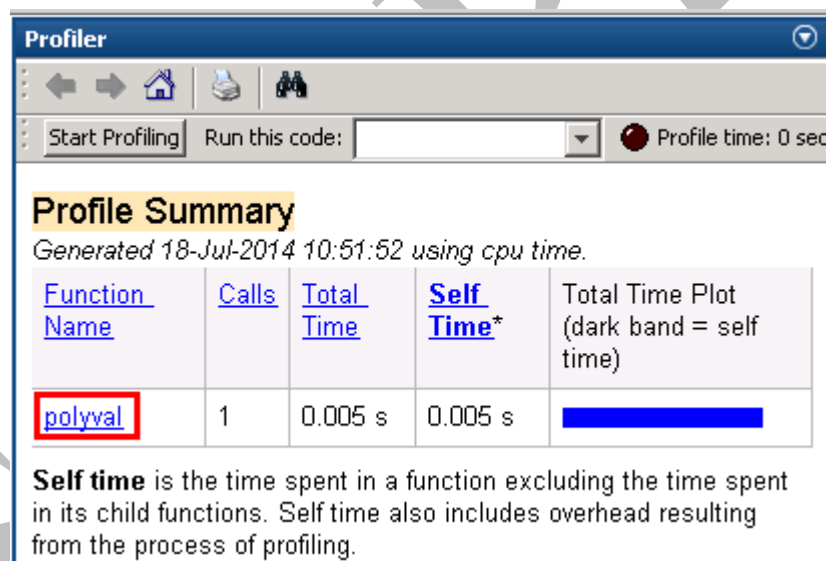
To use this functionality we will need the commands **profile on**, **profile clear**, **profile off** and finally **profile viewer**:



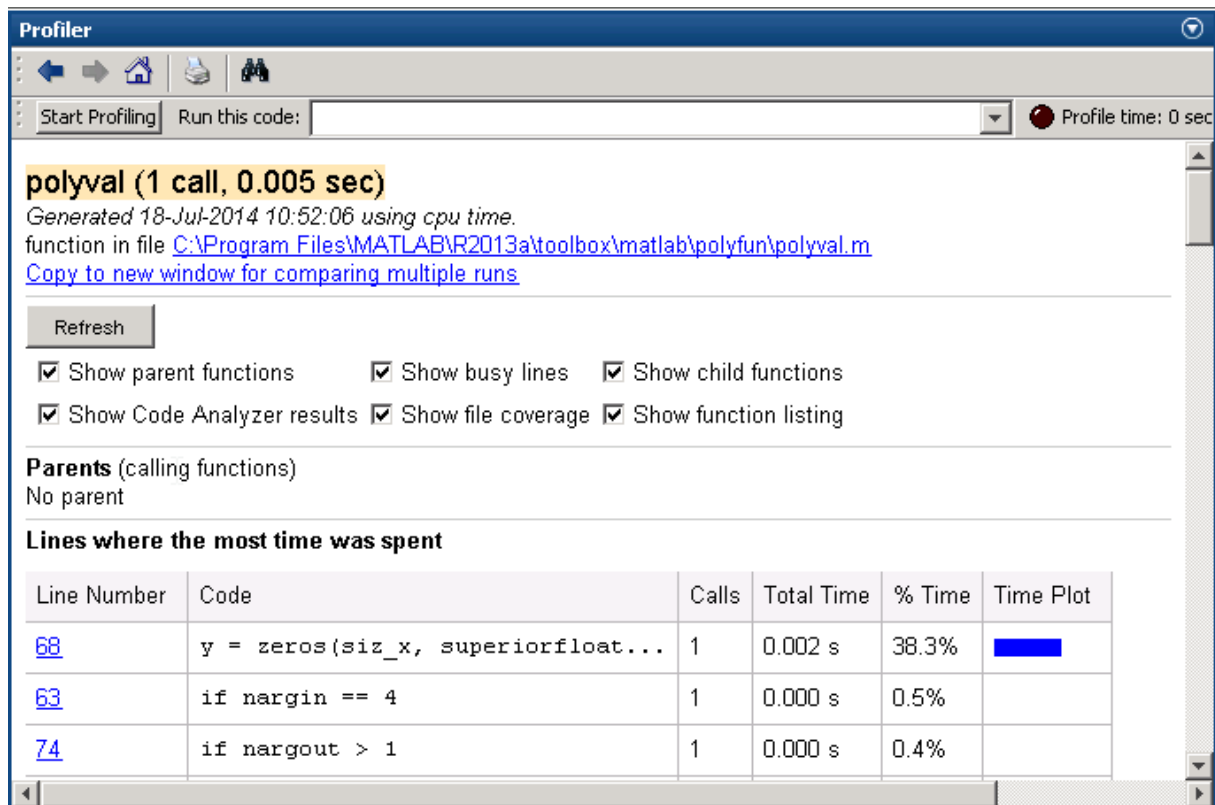
```

1  profile on
2  profile clear %you must first start profiling to clear the previous one
3
4  p=[3 -2 1];
5  polyval(p,[5 7 9])
6
7  profile off
8  profile viewer
  
```

Then you will see a new panel that evaluates the execution time of the function **polyval()**:



If you click on the [polyval](#) link you get more details about what happens inside this function:

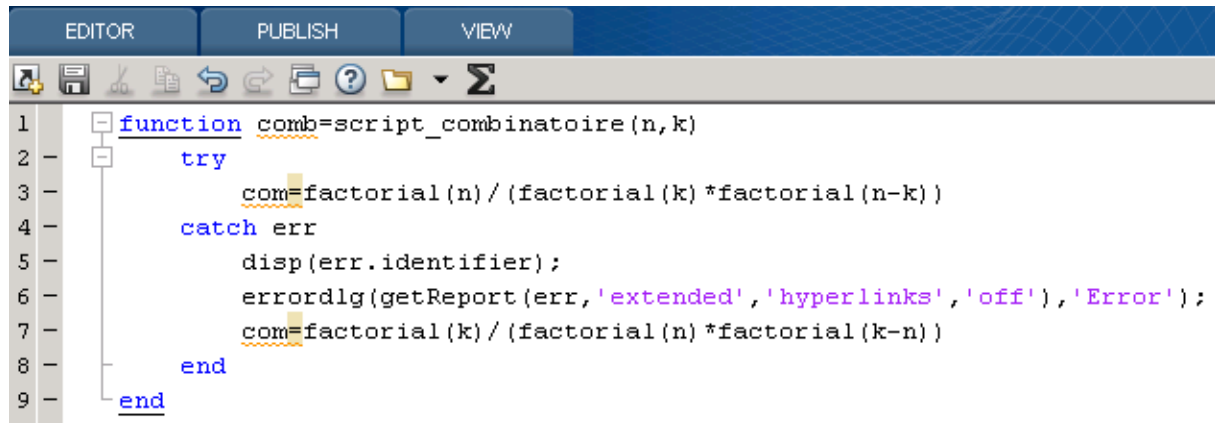


Simple but very useful!

Catching errors

Catching errors is one of the most important job of a professional scripting/programming language developer.

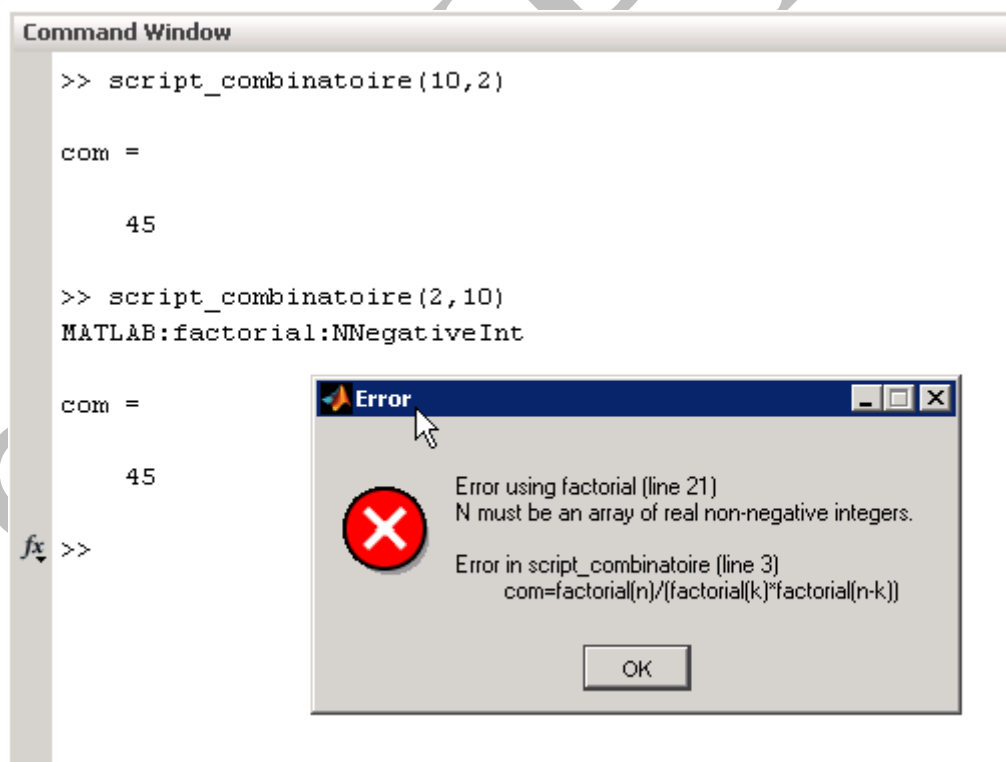
The syntax of MATLAB for this purpose is similar to many modern programming languages using the **Try...Catch...** structure as you can see below in this simple example:



```

1  function comb=script_combinatoire(n,k)
2      try
3          com=factorial(n)/(factorial(k)*factorial(n-k))
4      catch err
5          disp(err.identifier);
6          errordlg(getReport(err,'extended','hyperlinks','off'),'Error');
7          com=factorial(k)/(factorial(n)*factorial(k-n))
8      end
9  end
  
```

The result will look like following:



```

>> script_combinatoire(10,2)

com =

    45

>> script_combinatoire(2,10)
MATLAB:factorial:NNegativeInt

com =

    45
fx >>
  
```

Error

Error using factorial (line 21)
N must be an array of real non-negative integers.

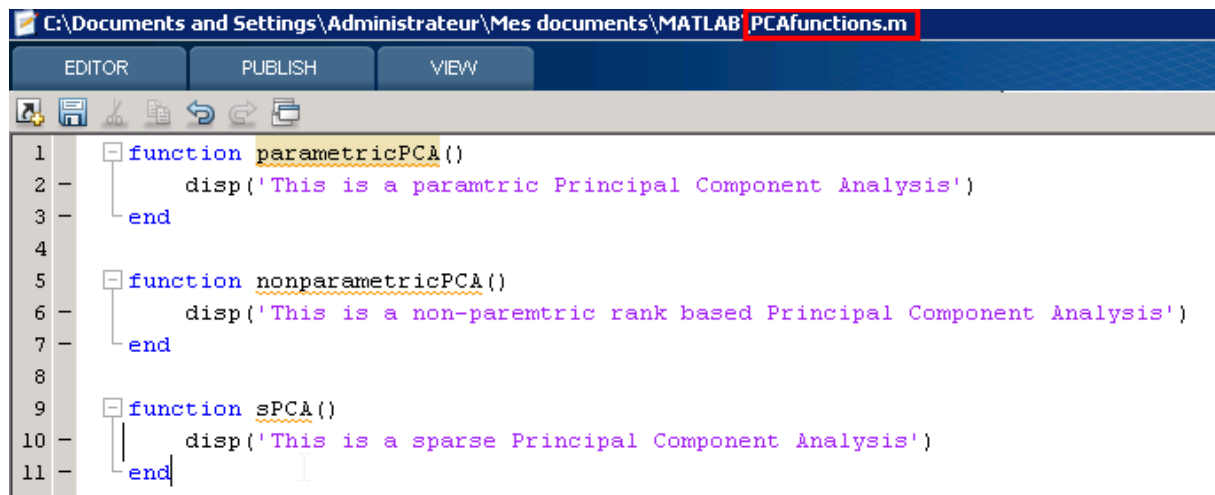
Error in script_combinatoire (line 3)
com=factorial(n)/(factorial(k)*factorial(n-k))

OK

Multiple callable functions in one *.m file

By default the only function accessible from outside is the first function whose name is the same as the script file name.

Here is an example of what we could do intuitively:



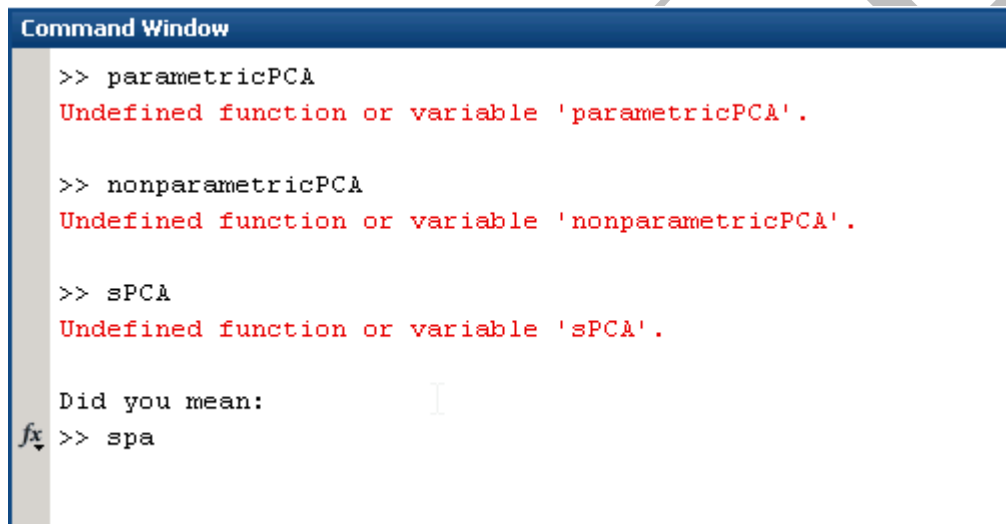
The image shows a MATLAB Editor window with the file 'PCAfunctions.m' open. The code contains three separate function definitions: 'parametricPCA()', 'nonparametricPCA()', and 'sPCA()'. Each function has a single line of code that displays a message using the 'disp' function. The functions are not yet called from the Command Window.

```

1 function parametricPCA()
2     disp('This is a parametric Principal Component Analysis')
3 end
4
5 function nonparametricPCA()
6     disp('This is a non-parametric rank based Principal Component Analysis')
7 end
8
9 function sPCA()
10    disp('This is a sparse Principal Component Analysis')
11 end

```

But the result will not be the expected one:



The image shows the MATLAB Command Window. Three commands have been entered: 'parametricPCA', 'nonparametricPCA', and 'sPCA'. Each command results in an error message: 'Undefined function or variable 'parametricPCA'.' and so on. The window also suggests a correction: 'Did you mean: spa'.

```

>> parametricPCA
Undefined function or variable 'parametricPCA'.

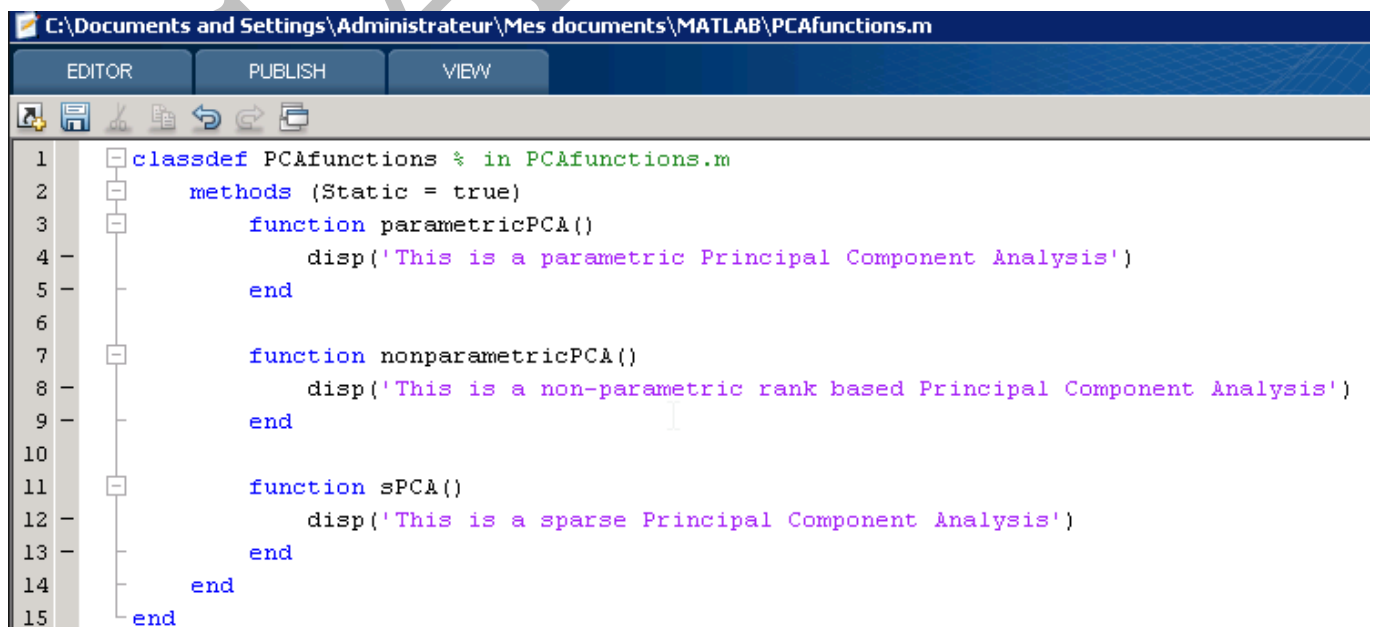
>> nonparametricPCA
Undefined function or variable 'nonparametricPCA'.

>> sPCA
Undefined function or variable 'sPCA'.

Did you mean:
spa

```

To be able to have multiple functions in a same file you have to use classes a following:



The image shows a MATLAB Editor window with the file 'PCAfunctions.m' open. The code uses the 'classdef' keyword to define a class 'PCAfunctions' within the same file. Inside the class, three methods are defined: 'parametricPCA()', 'nonparametricPCA()', and 'sPCA()'. Each method has a single line of code that displays a message using the 'disp' function. The class is defined with 'Static = true'.

```

1 classdef PCAfunctions % in PCAfunctions.m
2     methods (Static = true)
3         function parametricPCA()
4             disp('This is a parametric Principal Component Analysis')
5         end
6
7         function nonparametricPCA()
8             disp('This is a non-parametric rank based Principal Component Analysis')
9         end
10
11        function sPCA()
12            disp('This is a sparse Principal Component Analysis')
13        end
14    end
15 end

```

After in any script or in the *Command Window* you just write:

```
Command Window
>> import PCAfunctions.*
>> parametricPCA
This is a paramtric Principal Component Analysis
>> nonparametricPCA
This is a non-parametric rank based Principal Component Analysis
>> sPCA
This is a sparse Principal Component Analysis
fx >> |
```

At the same time this is a first introduction to OOP (Object Oriented Programming) in MATLAB.

Debugging technics

Debugging is an important part of the job of any computer scientist. As we know one of the axioms of computing science is that almost all scripts have a least one potential bug (that is not necessary due to the code itself!)

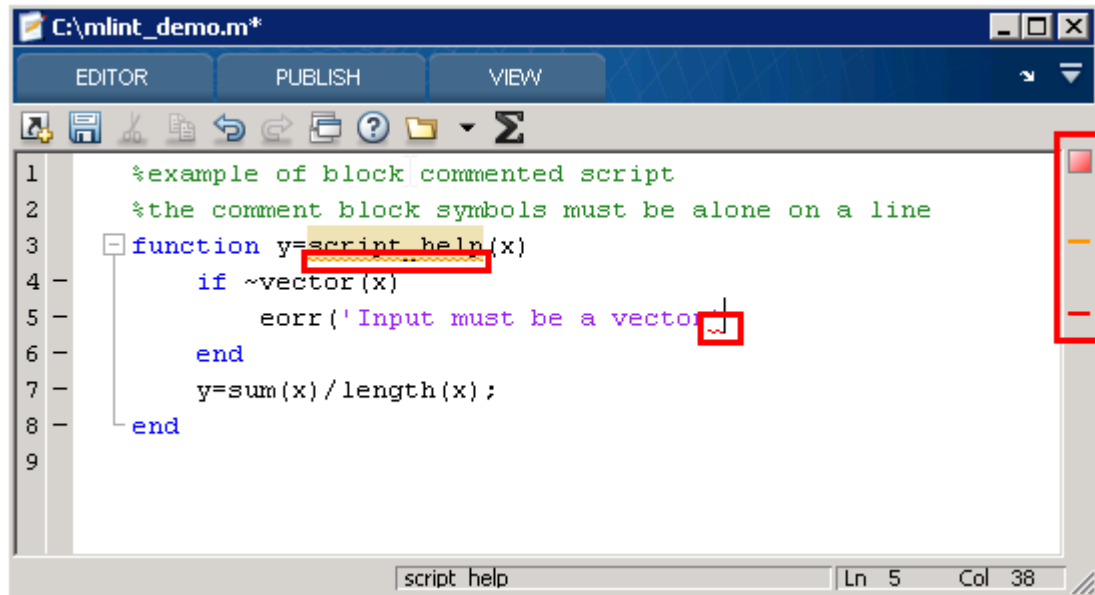
There are different commands for this purpose in MATLAB and let us see the most important of them.

SAMPLE

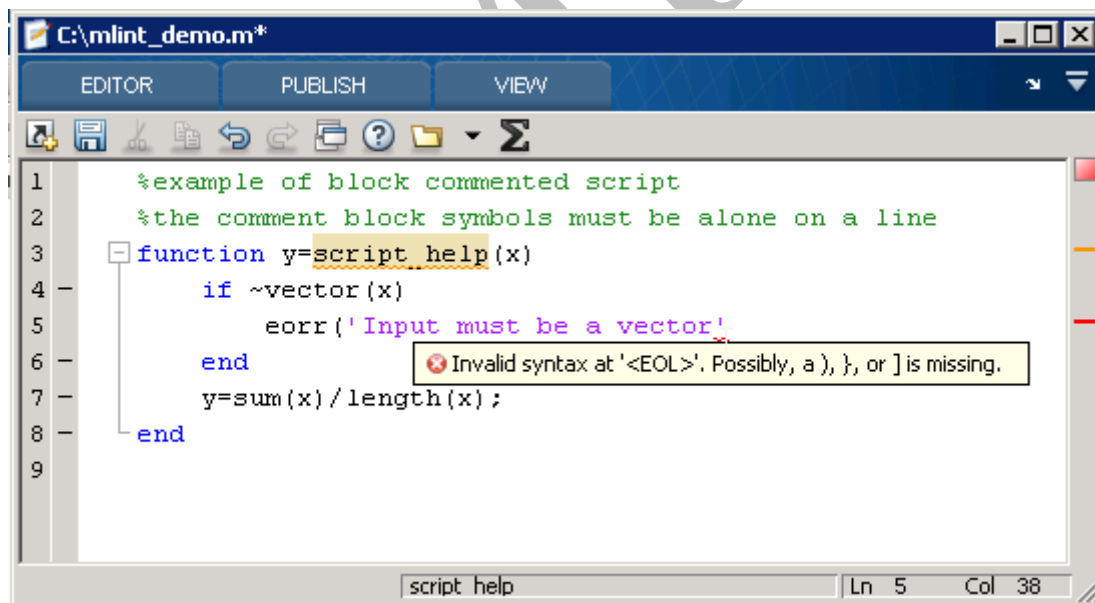
M-lint

M-lint real time analyzer

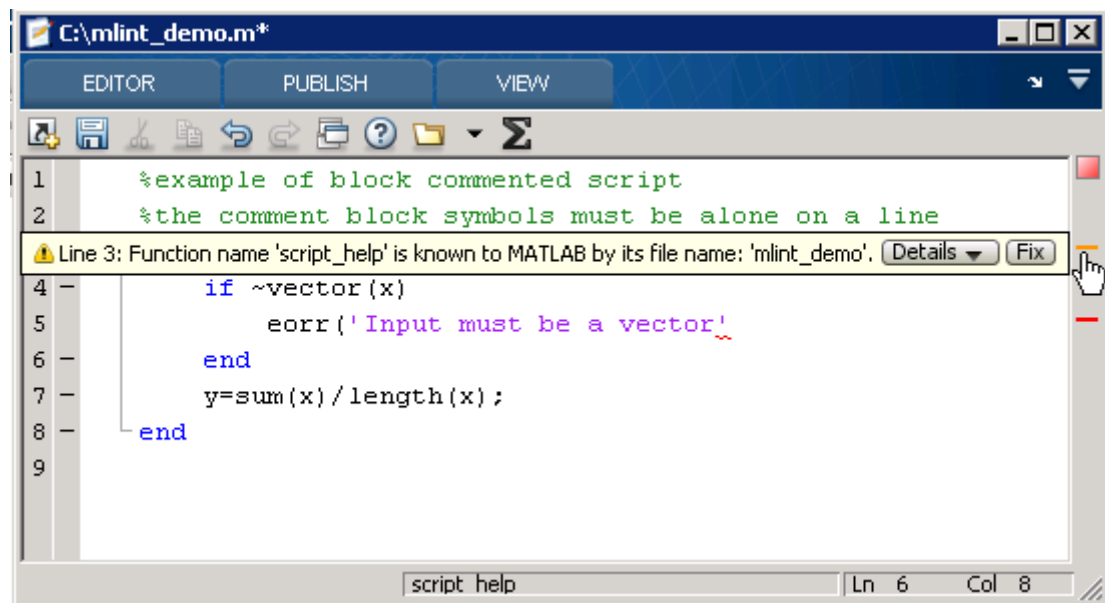
The M-lint real time analyzer is easy to recognize. Consider the following function and see:



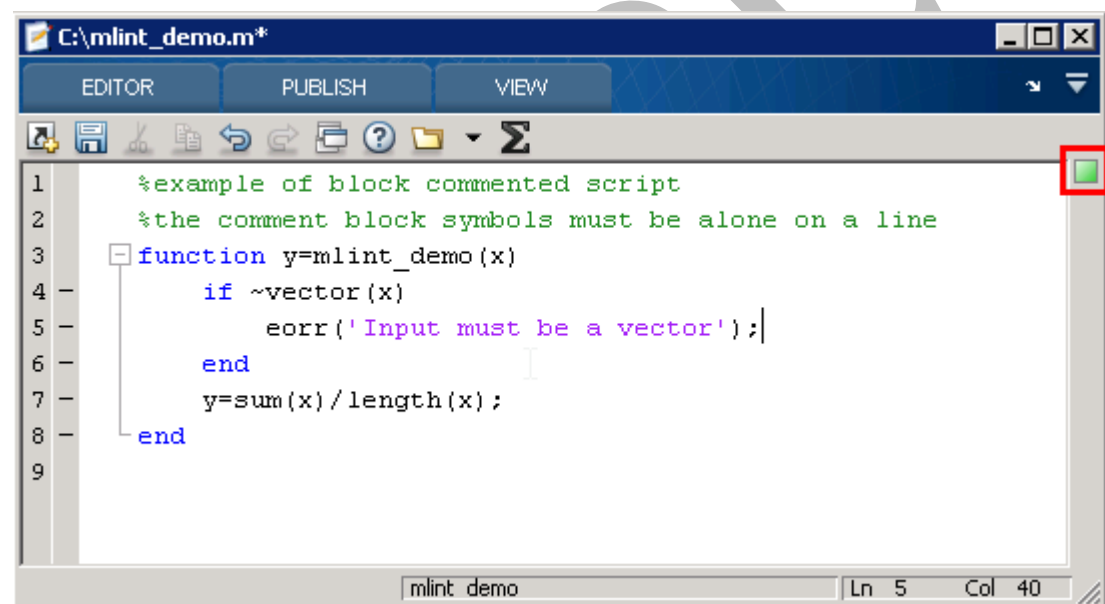
Going with the mouse on the colors we get for example:



Or:

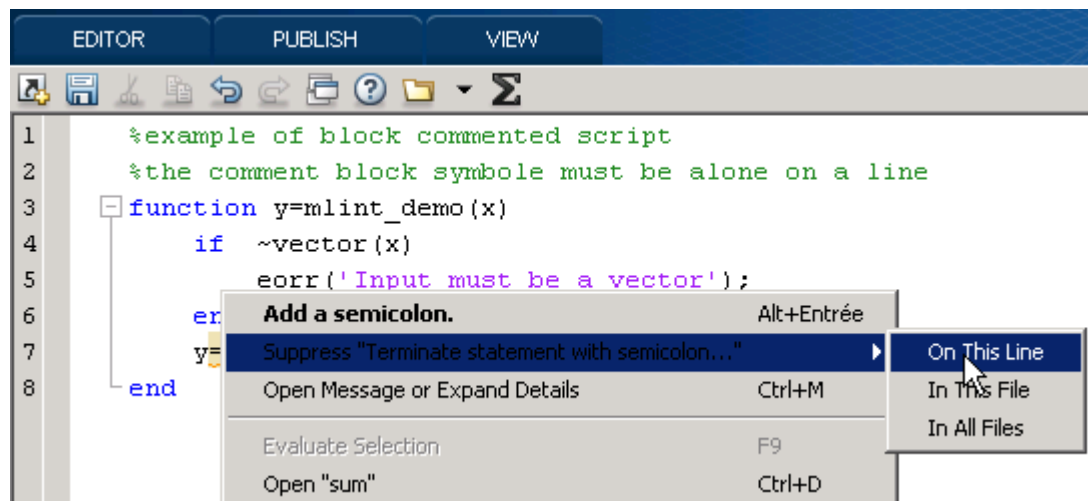


If the system thinks... that there are no errors you get a green square:

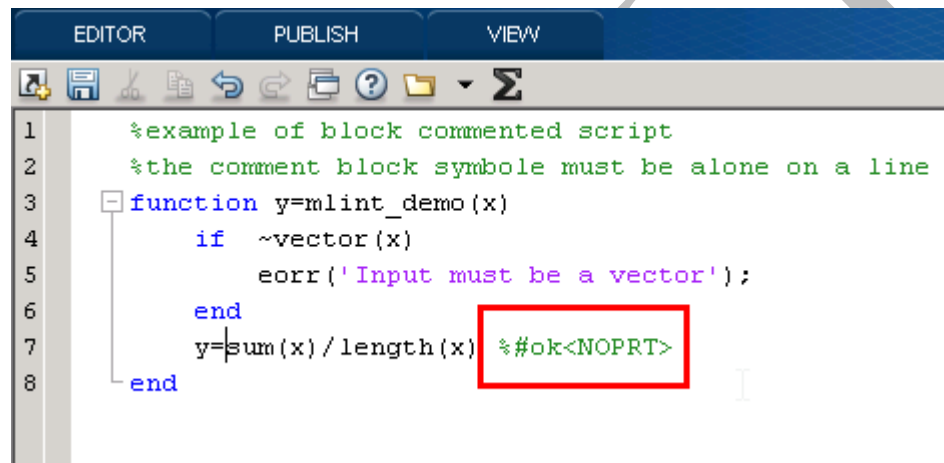


but as you can see there is still an error in reality...

If we ask the M-lint analyzer to ignore something:

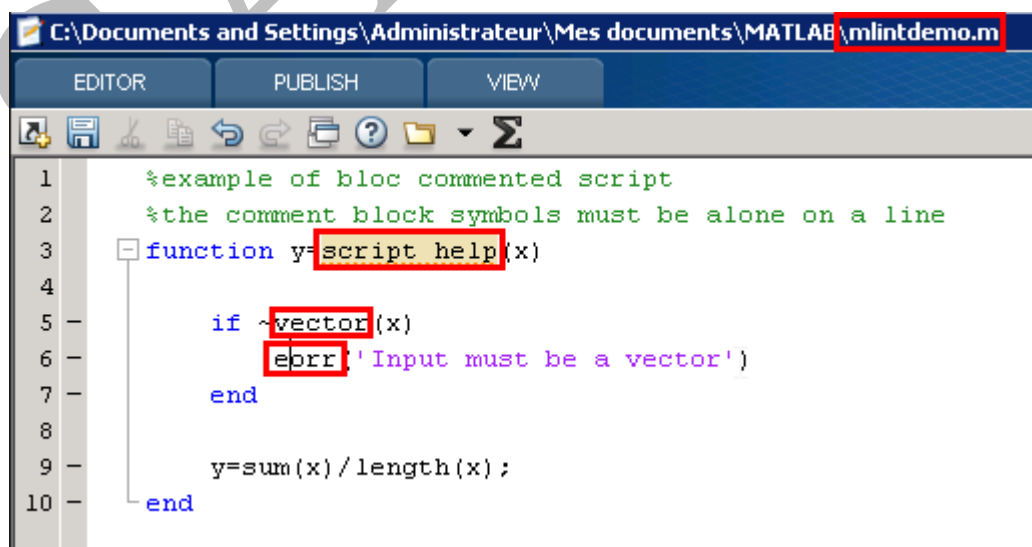


we get:

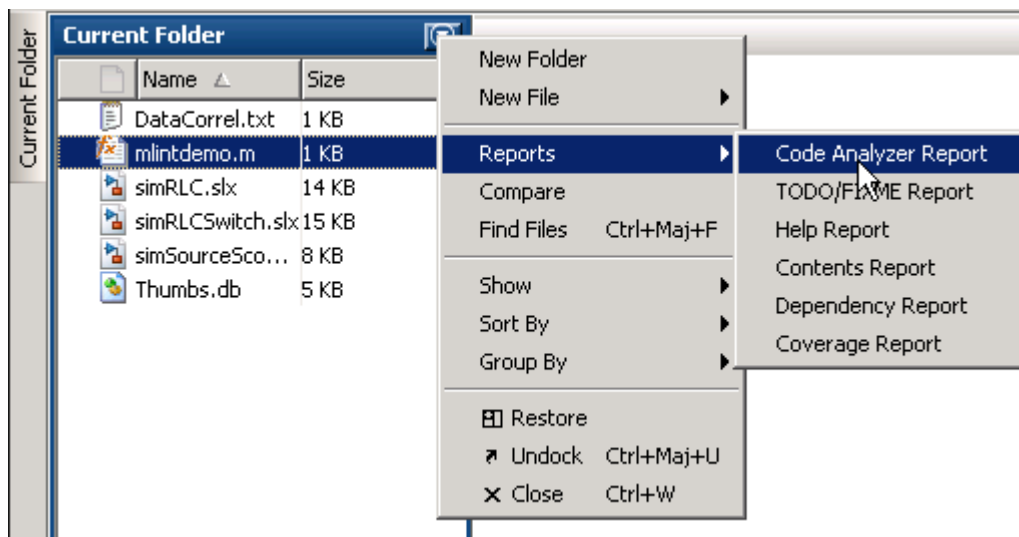


M-lint reports

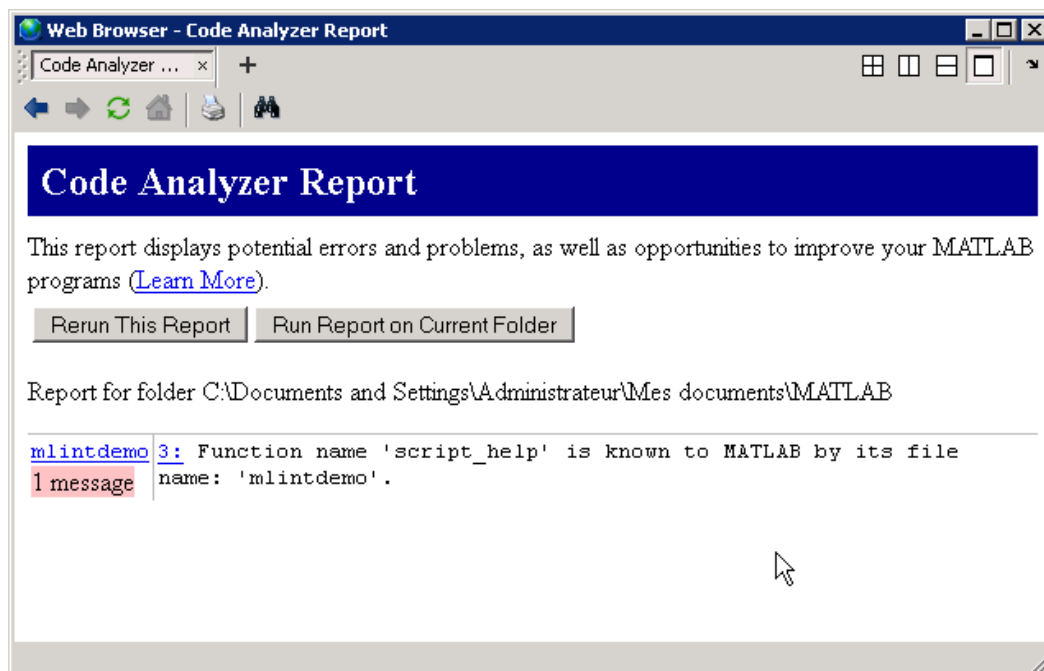
Consider the following small script with multiple incoherencies and errors:



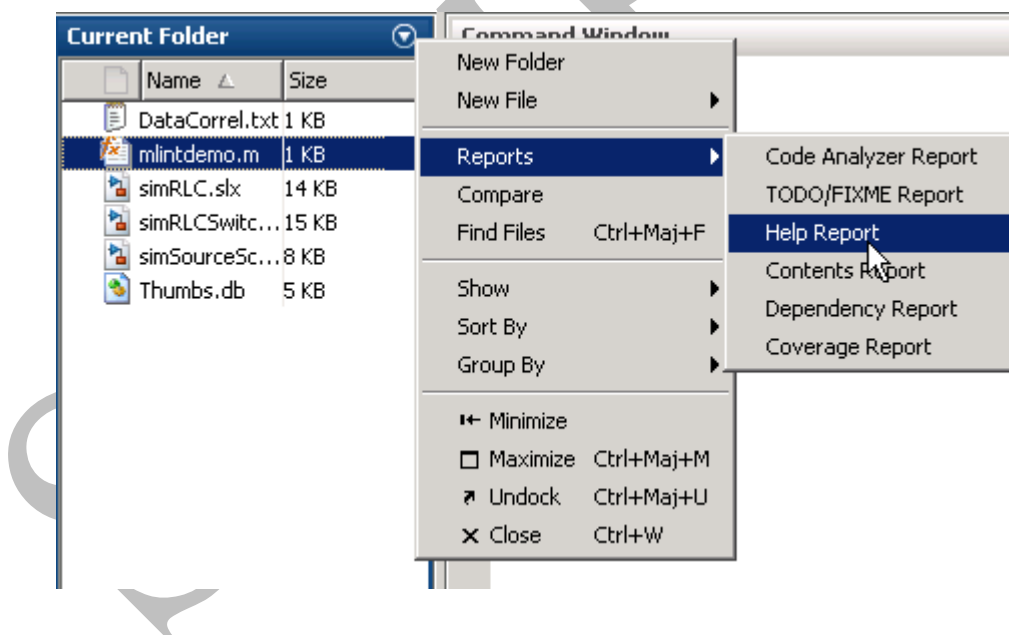
Now take a look the M-Lint **Code Analyzer Report**:



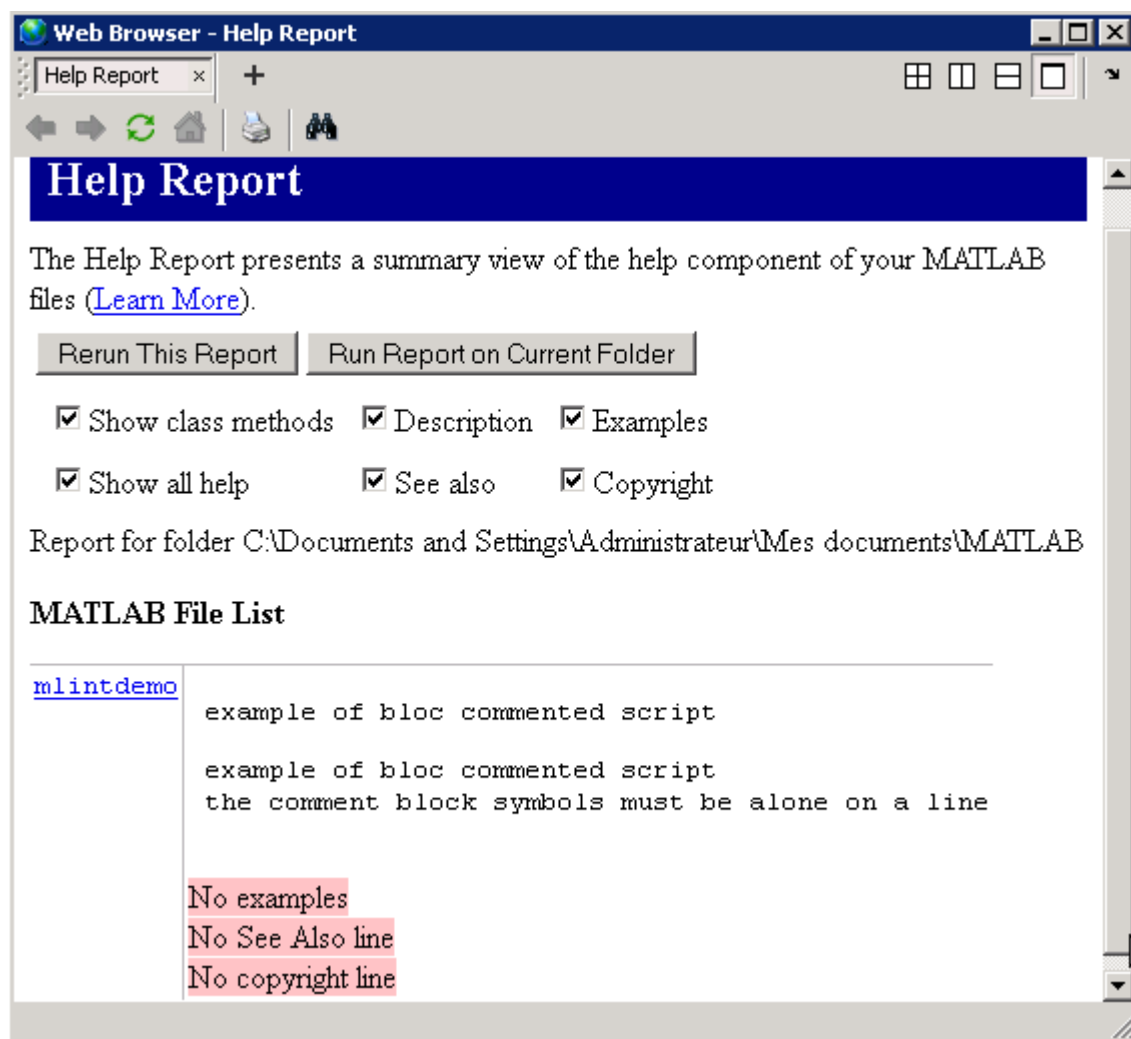
This will give a poor result with a lack in error detection:



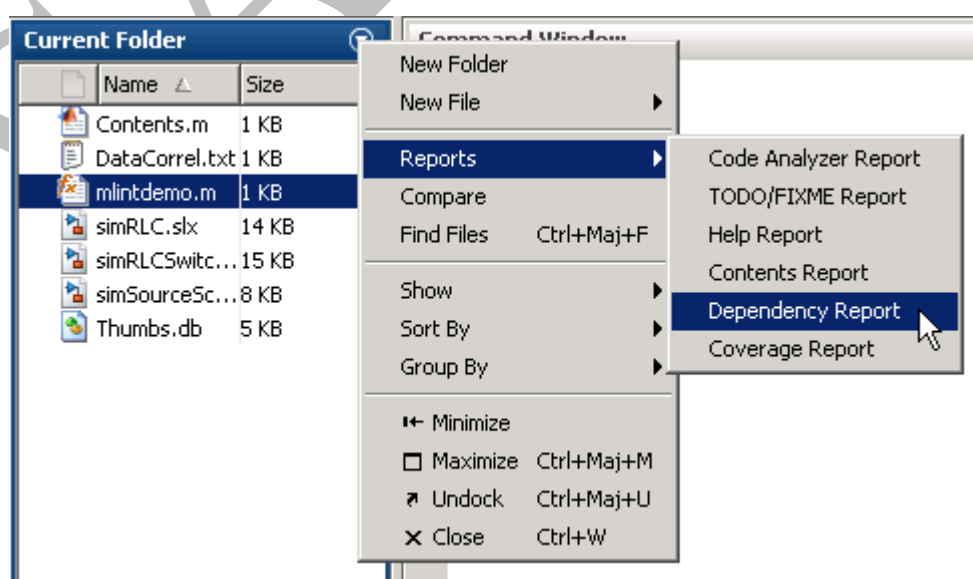
Now consider the **Help Report**:



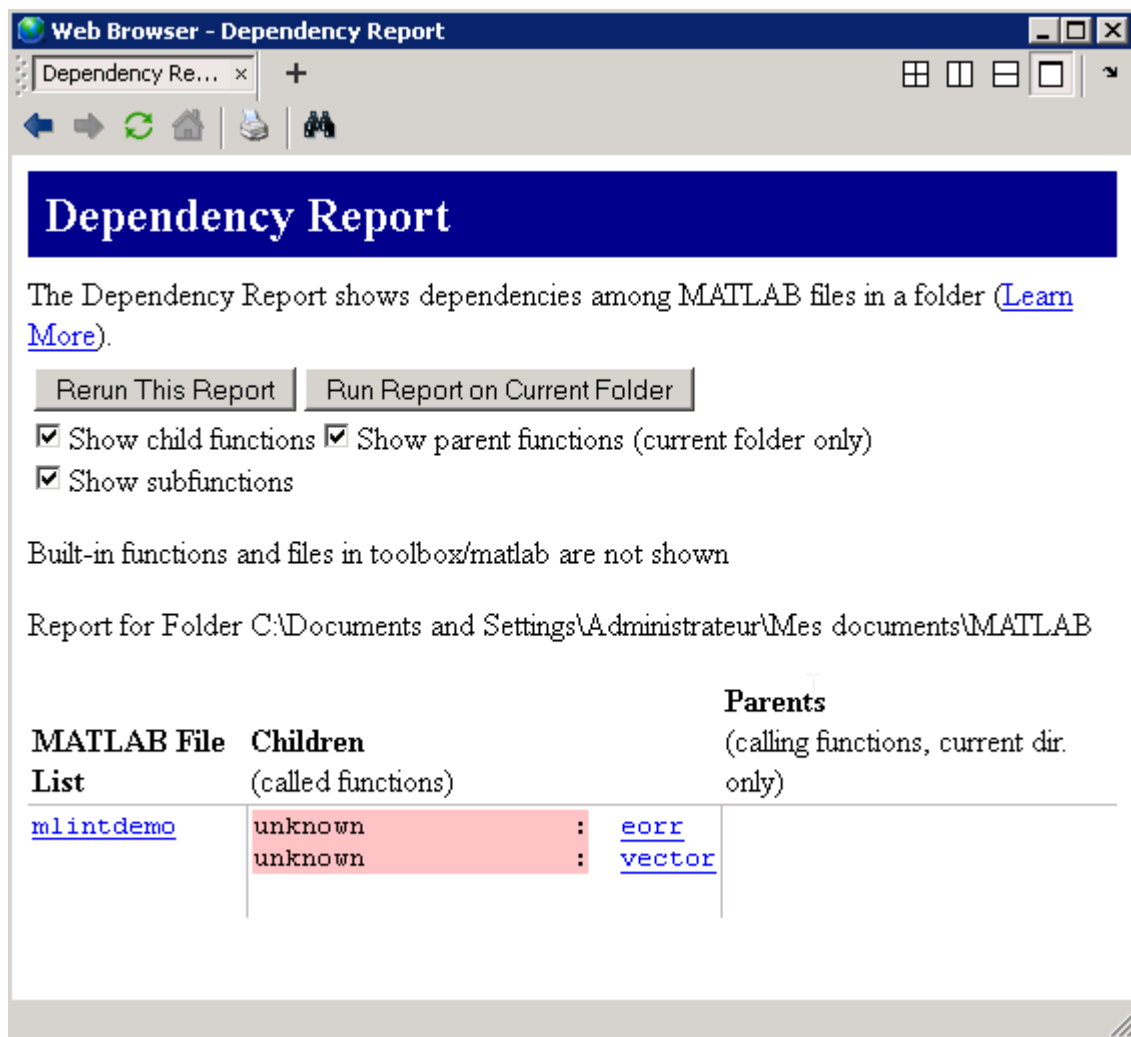
When your script is not correctly documented you get:



Now have a look the **Dependency Report**:



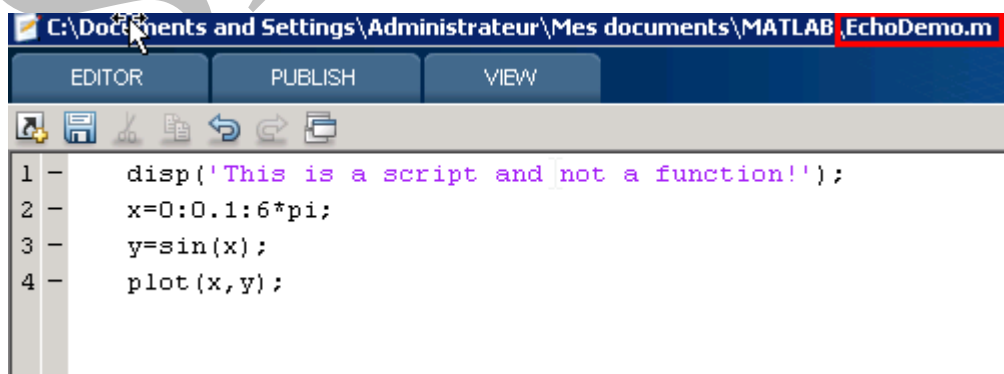
And now we get the error indication about our two misspelled functions:



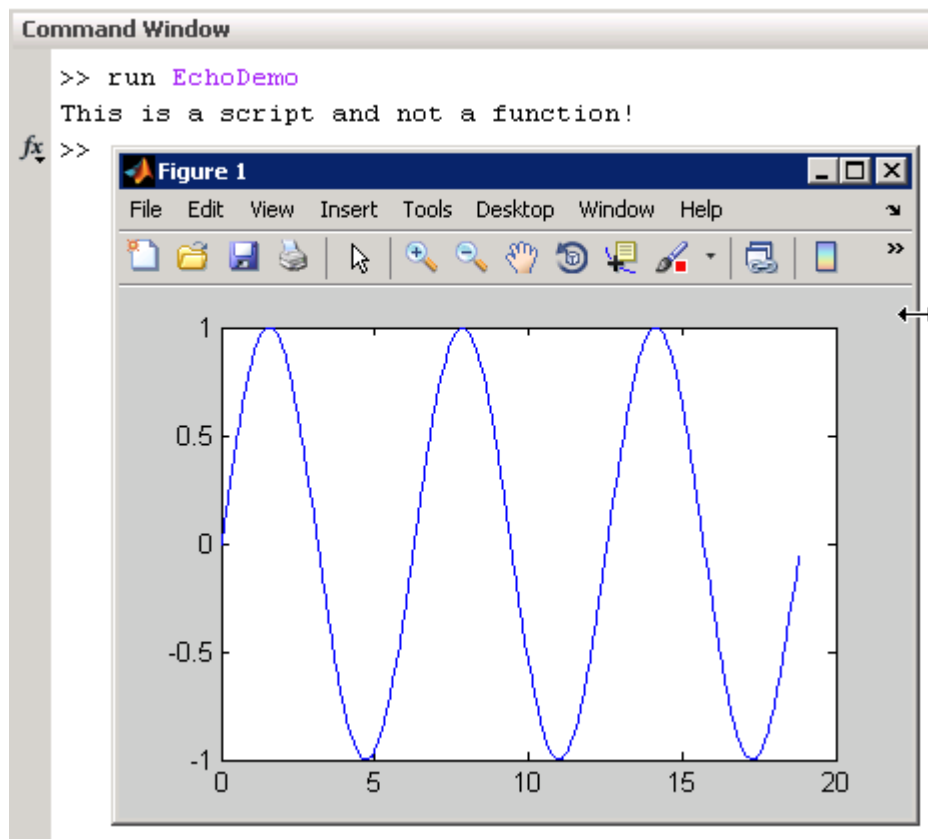
Echo on/Echo off

For simple scripts

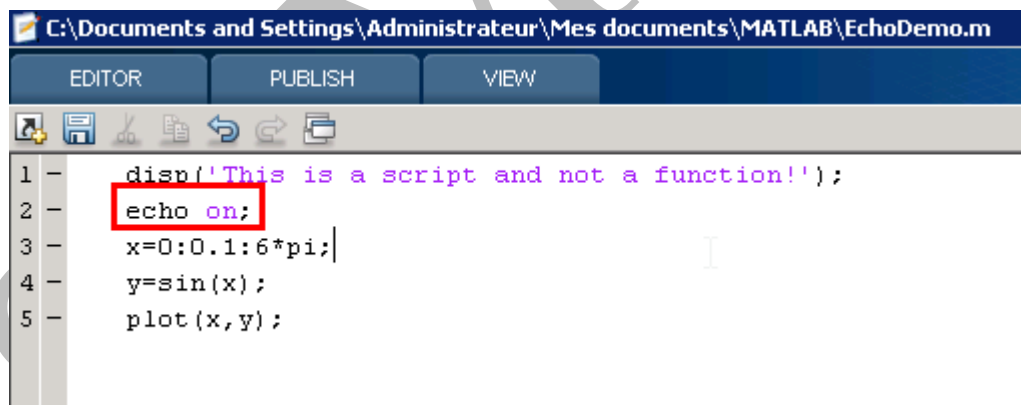
Suppose we have the following script:



We run it:



We want to see the inside of the script when we run it without having to open/edit the script file. For this purpose we can use the **echo on** function:

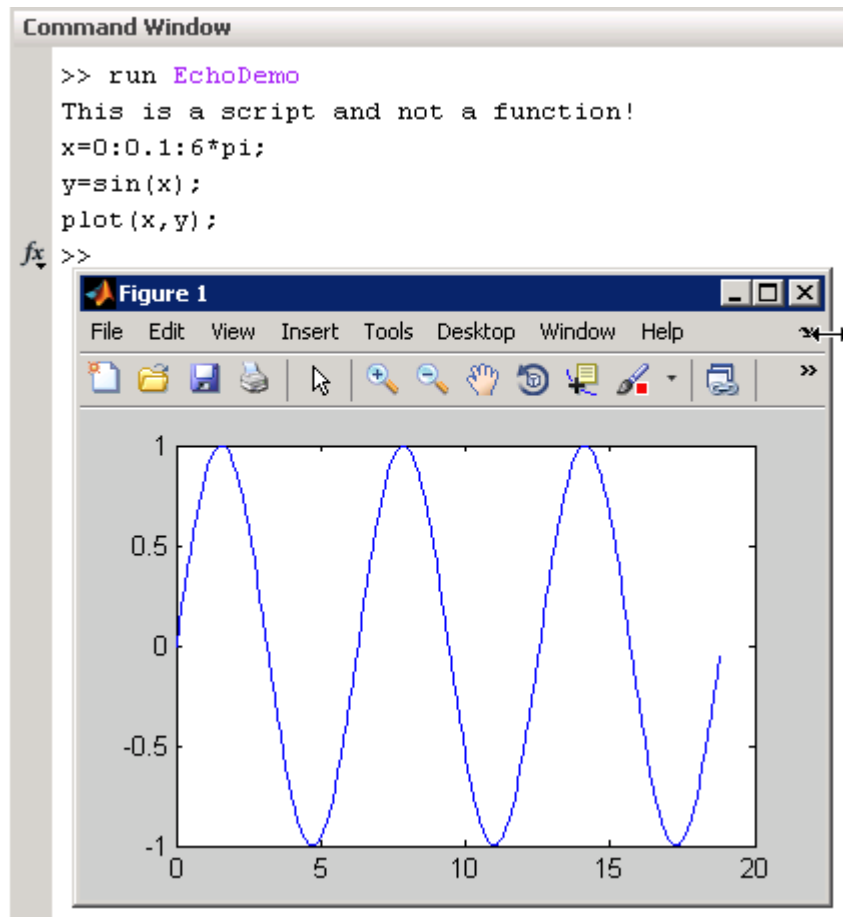


The image shows a MATLAB Editor window titled `C:\Documents and Settings\Administrateur\Mes documents\MATLAB\EchoDemo.m`. The window has tabs for EDITOR, PUBLISH, and VIEW. The script content is as follows:

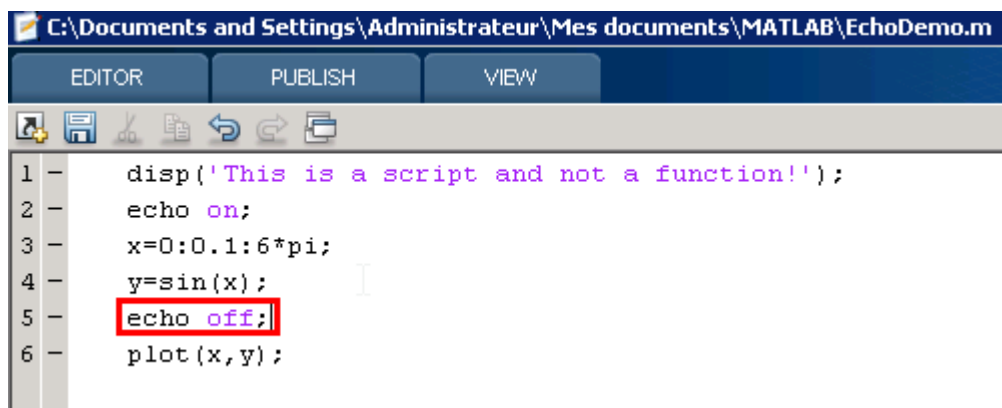
```
1 - disp('This is a script and not a function!');
2 - echo on;
3 - x=0:0.1:6*pi;
4 - y=sin(x);
5 - plot(x,y);
```

The line `echo on;` is highlighted with a red box.

We run it and as you can see it, all the details are printed out:

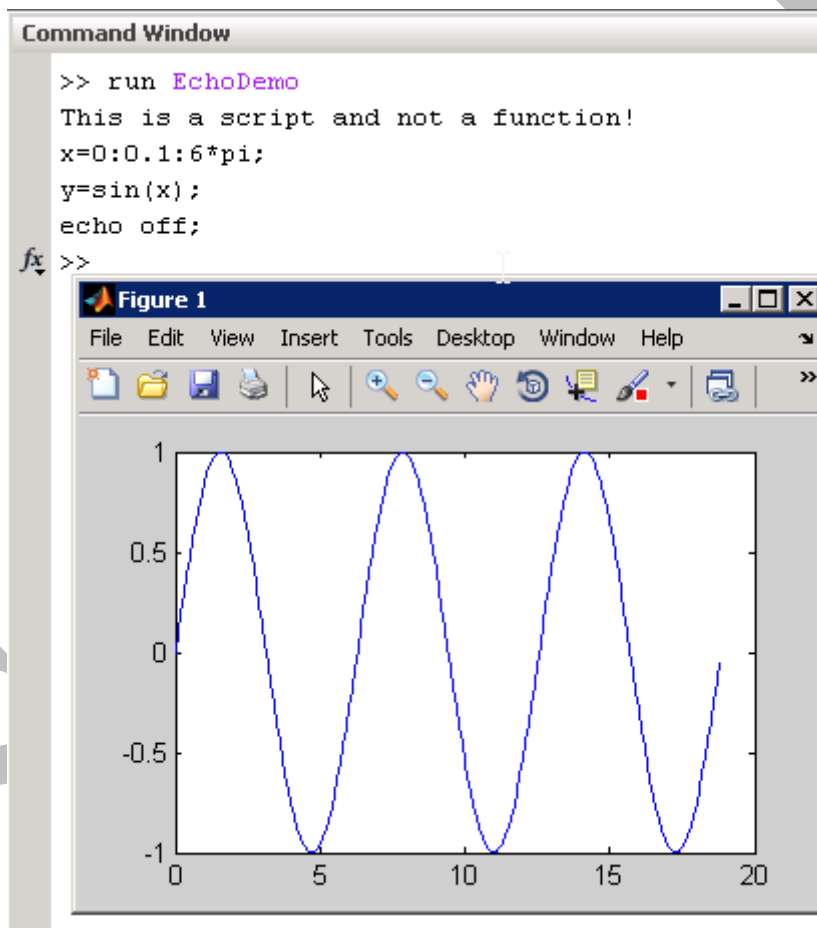


Or if we cancel the **echo on** in the script using the **echo off** command:



```
C:\Documents and Settings\Administrateur\Mes documents\MATLAB\EchoDemo.m
EDITOR PUBLISH VIEW
1 - disp('This is a script and not a function!');
2 - echo on;
3 - x=0:0.1:6*pi;
4 - y=sin(x);
5 - echo off;
6 - plot(x,y);
```

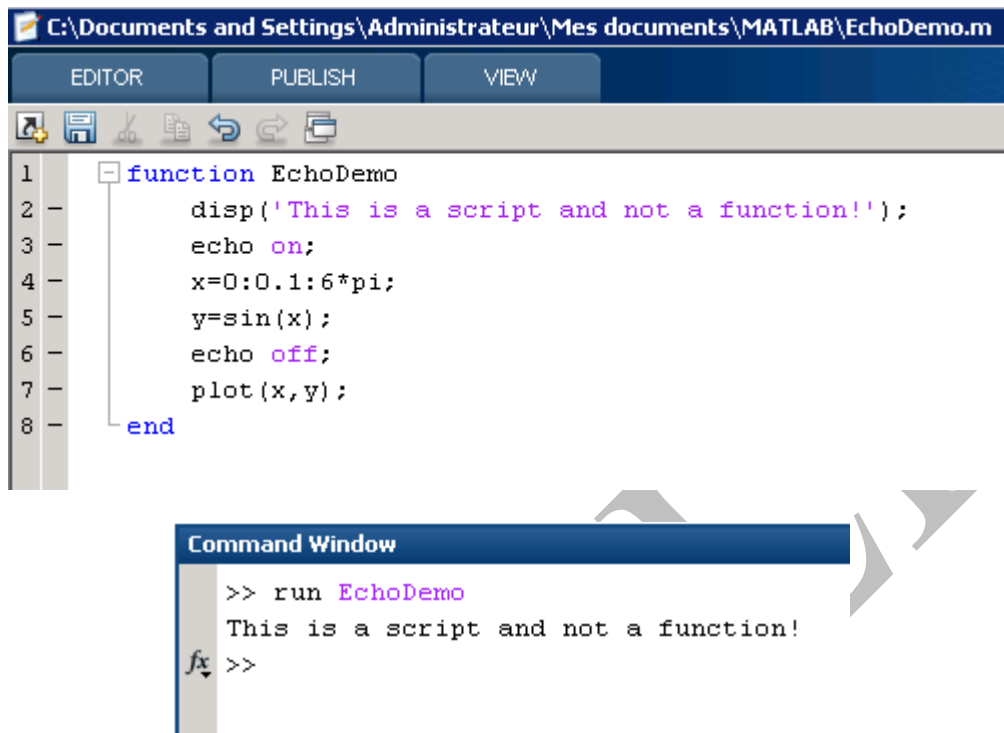
Then we run it again:



Works well!

For functions scripts

If we change the previous script into a function we see that the **echo on** has no effect:



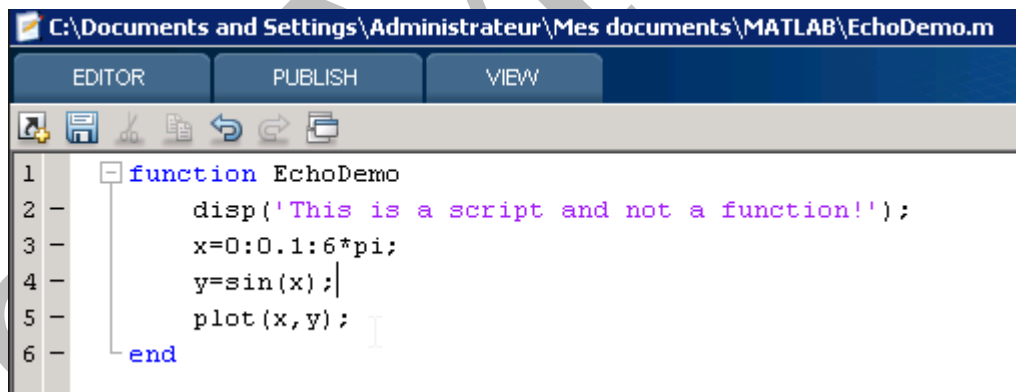
The image shows the MATLAB Editor window with a file named 'EchoDemo.m'. The code is as follows:

```
1 function EchoDemo
2     disp('This is a script and not a function!');
3     echo on;
4     x=0:0.1:6*pi;
5     y=sin(x);
6     echo off;
7     plot(x,y);
8 end
```

The Command Window shows the execution of the function:

```
>> run EchoDemo
This is a script and not a function!
fx >>
```

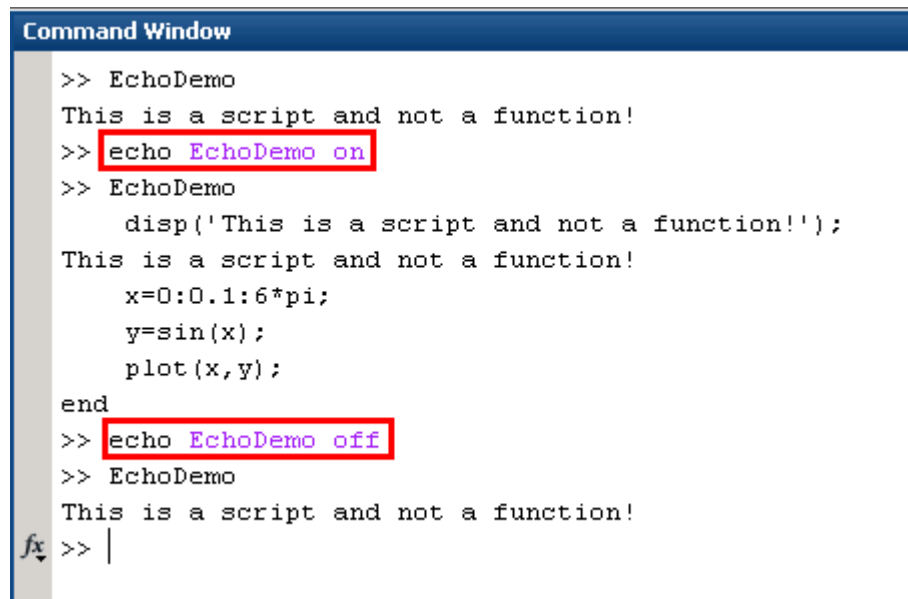
Then we can remove the **echo on** and **echo off** because they are useless:



The image shows the MATLAB Editor window with the same file 'EchoDemo.m'. The code has been modified to remove the 'echo on' and 'echo off' commands:

```
1 function EchoDemo
2     disp('This is a script and not a function!');
3     x=0:0.1:6*pi;
4     y=sin(x);
5     plot(x,y);
6 end
```

To activate the echo on a function we have to use the command **echo nameOfFunction on** from the *Command Window*:



```

Command Window

>> EchoDemo
This is a script and not a function!
>> echo EchoDemo on
>> EchoDemo
    disp('This is a script and not a function!');
This is a script and not a function!
    x=0:0.1:6*pi;
    y=sin(x);
    plot(x,y);
end
>> echo EchoDemo off
>> EchoDemo
This is a script and not a function!
fx >> |

```

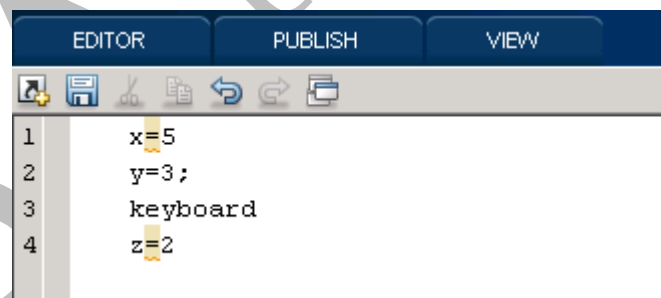
And to stop to echo this function each time we run it, we just write **echo nameOfFunction off**.

Manipulations during a run

When you run a script you can't type commands while the script is running. If you write a **pause(...)** statement in the script this won't help because you won't be able to execute immediately what you write in the *Command Window*.

The only way to stop a code and the same time to write something in *Command Window* is to use de **keyboard** command.

For example if you execute this script:



```

EDITOR    PUBLISH    VIEW
1         x=5
2         y=3;
3         keyboard
4         z=2

```

You will get:



```

Command Window

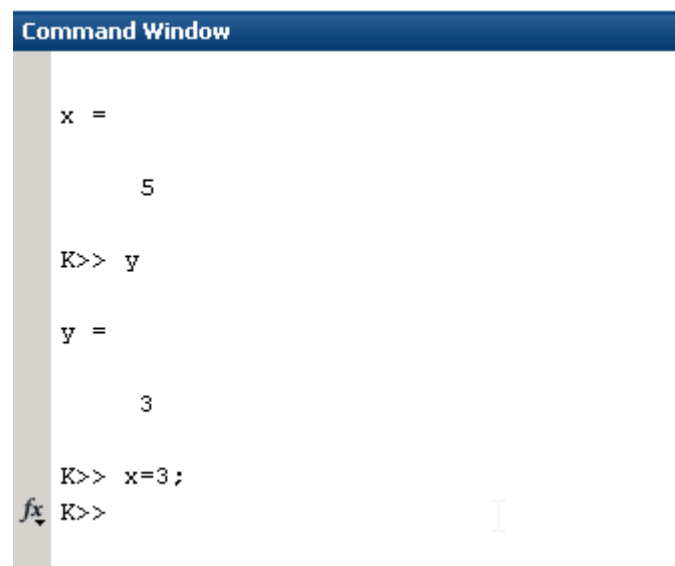
x =

    5

fx K>>

```

Then you can type what you want and execute it immediately:



```
Command Window

x =

    5

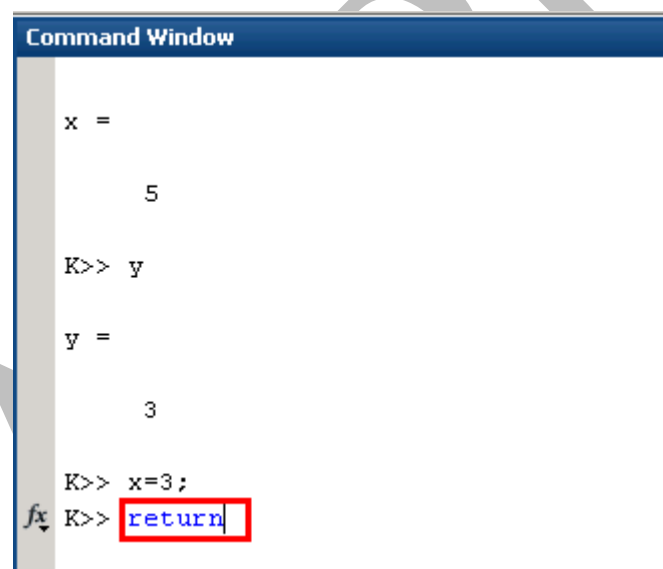
K>> y

y =

    3

K>> x=3;
fx K>>
```

To continue to run the script, you just type **return**:



```
Command Window

x =

    5

K>> y

y =

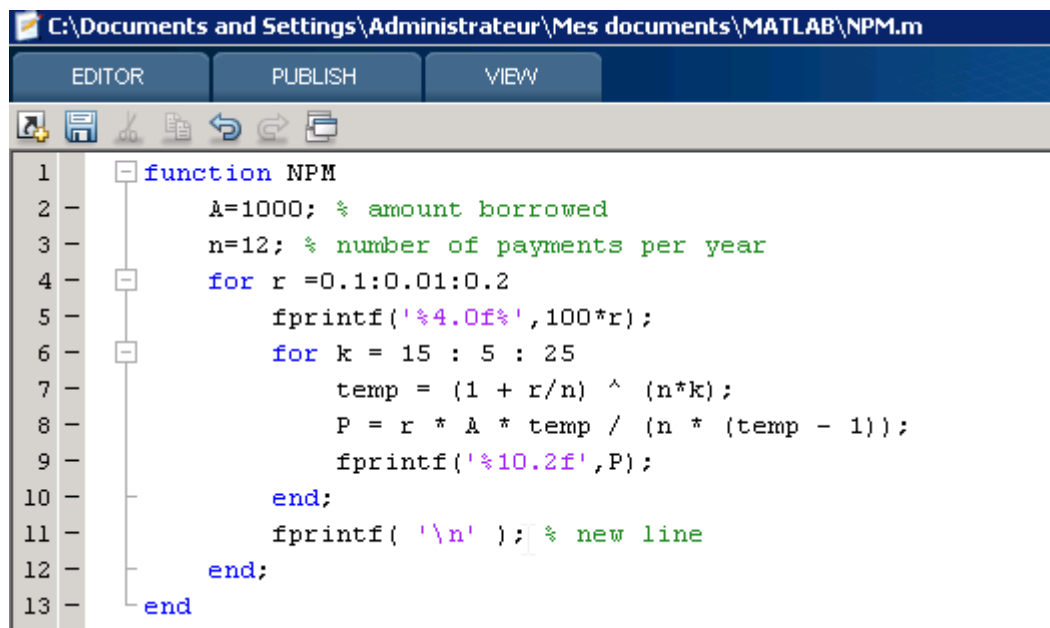
    3

K>> x=3;
fx K>> return
```

And that's it.

Debugging (db...) functions

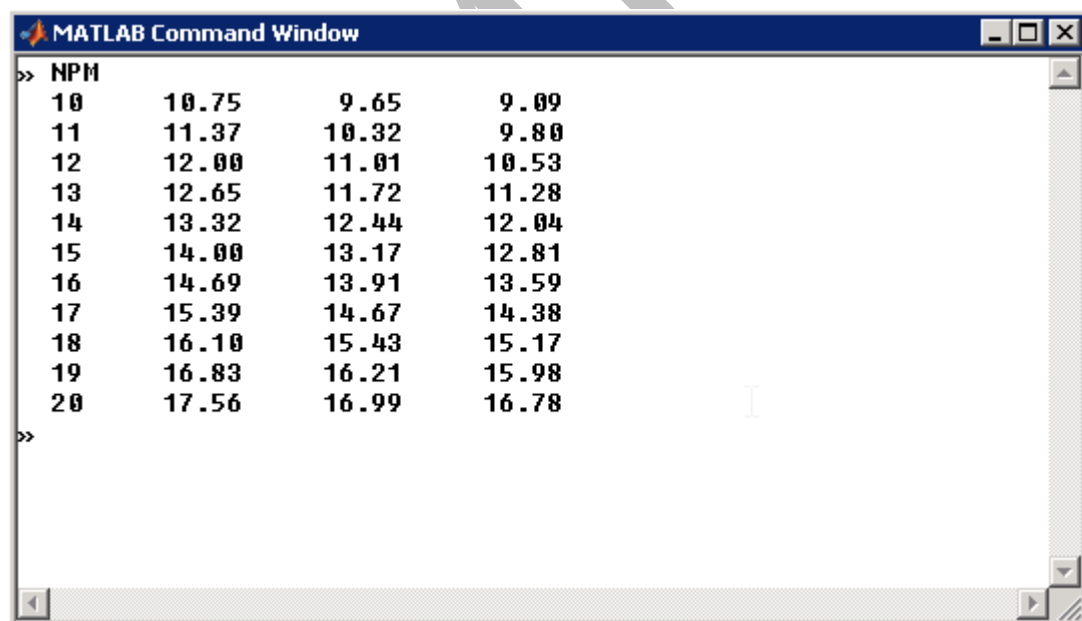
Consider the following script:



```

C:\Documents and Settings\Administrateur\Mes documents\MATLAB\NPM.m
EDITOR PUBLISH VIEW
1 function NPM
2     A=1000; % amount borrowed
3     n=12; % number of payments per year
4     for r =0.1:0.01:0.2
5         fprintf('%4.0f%',100*r);
6         for k = 15 : 5 : 25
7             temp = (1 + r/n) ^ (n*k);
8             P = r * A * temp / (n * (temp - 1));
9             fprintf('%10.2f',P);
10        end;
11        fprintf( '\n' ); % new line
12    end;
13 end
  
```

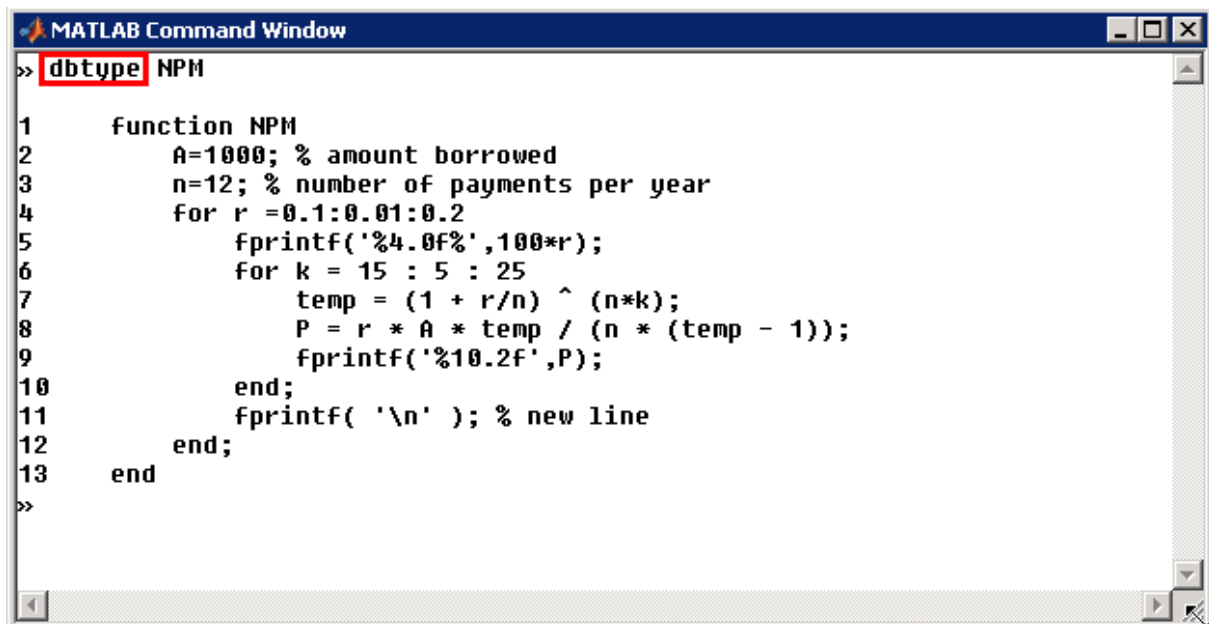
If you run it you will simply get:



```

MATLAB Command Window
>> NPM
10    10.75    9.65    9.09
11    11.37    10.32    9.80
12    12.00    11.01    10.53
13    12.65    11.72    11.28
14    13.32    12.44    12.04
15    14.00    13.17    12.81
16    14.69    13.91    13.59
17    15.39    14.67    14.38
18    16.10    15.43    15.17
19    16.83    16.21    15.98
20    17.56    16.99    16.78
>>
  
```

Now if we use the command **dbtype** we get a result that is similar to **echo on** with the difference that we see the line number and also the tabulation structure of the original script:

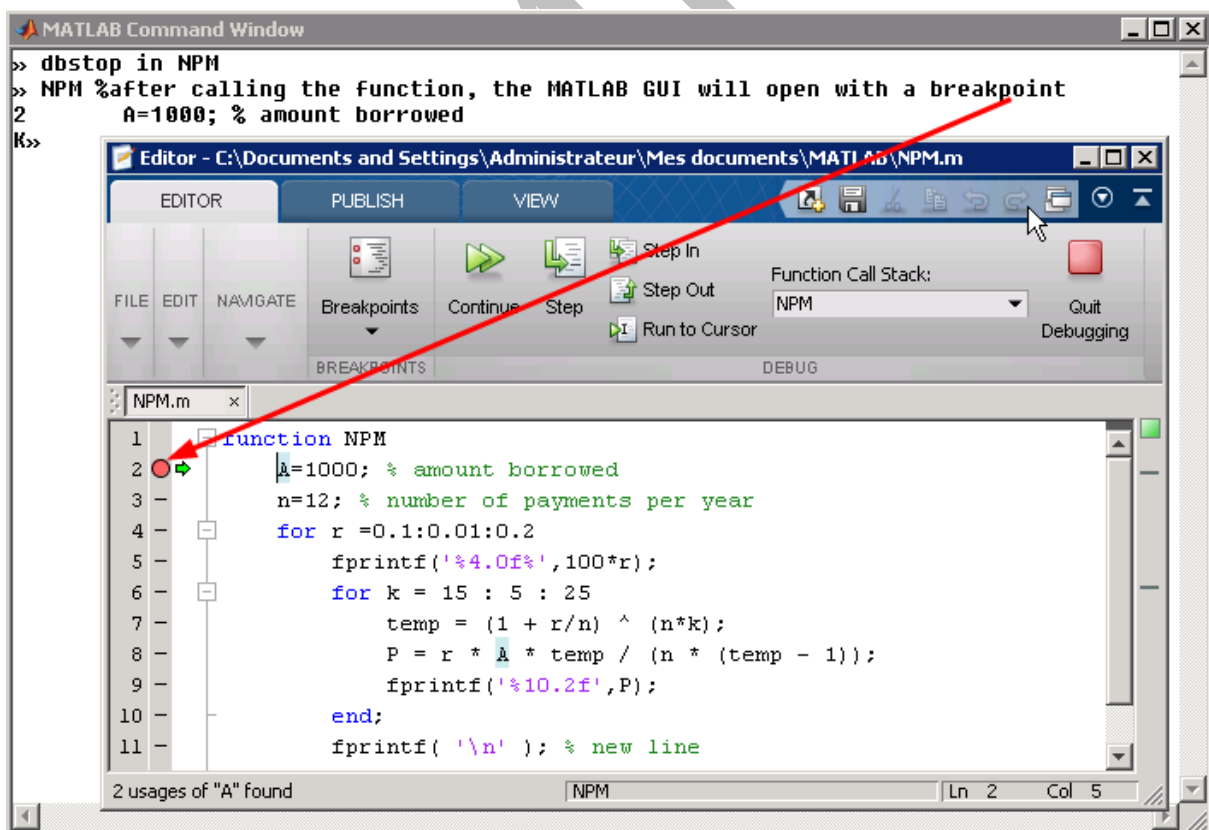


```

>> dbtype NPM

1      function NPM
2          A=1000; % amount borrowed
3          n=12; % number of payments per year
4          for r = 0.1:0.01:0.2
5              fprintf('%4.0f%',100*r);
6              for k = 15 : 5 : 25
7                  temp = (1 + r/n) ^ (n*k);
8                  P = r * A * temp / (n * (temp - 1));
9                  fprintf('%10.2f',P);
10             end;
11             fprintf( '\n' ); % new line
12         end;
13     end
>>
  
```

If we use the command **dbstop** the function will be executed line by line with the keyboard mode activated and **the scripts editor automatically opens**:



```

>> dbstop in NPM
>> NPM %after calling the function, the MATLAB GUI will open with a breakpoint
2      A=1000; % amount borrowed
K>>
  
```

Editor - C:\Documents and Settings\Administrateur\Mes documents\MATLAB\NPM.m

EDITOR PUBLISH VIEW

FILE EDIT NAVIGATE Breakpoints Continue Step Step In Step Out Run to Cursor Function Call Stack: NPM Quit Debugging

BREAKPOINTS DEBUG

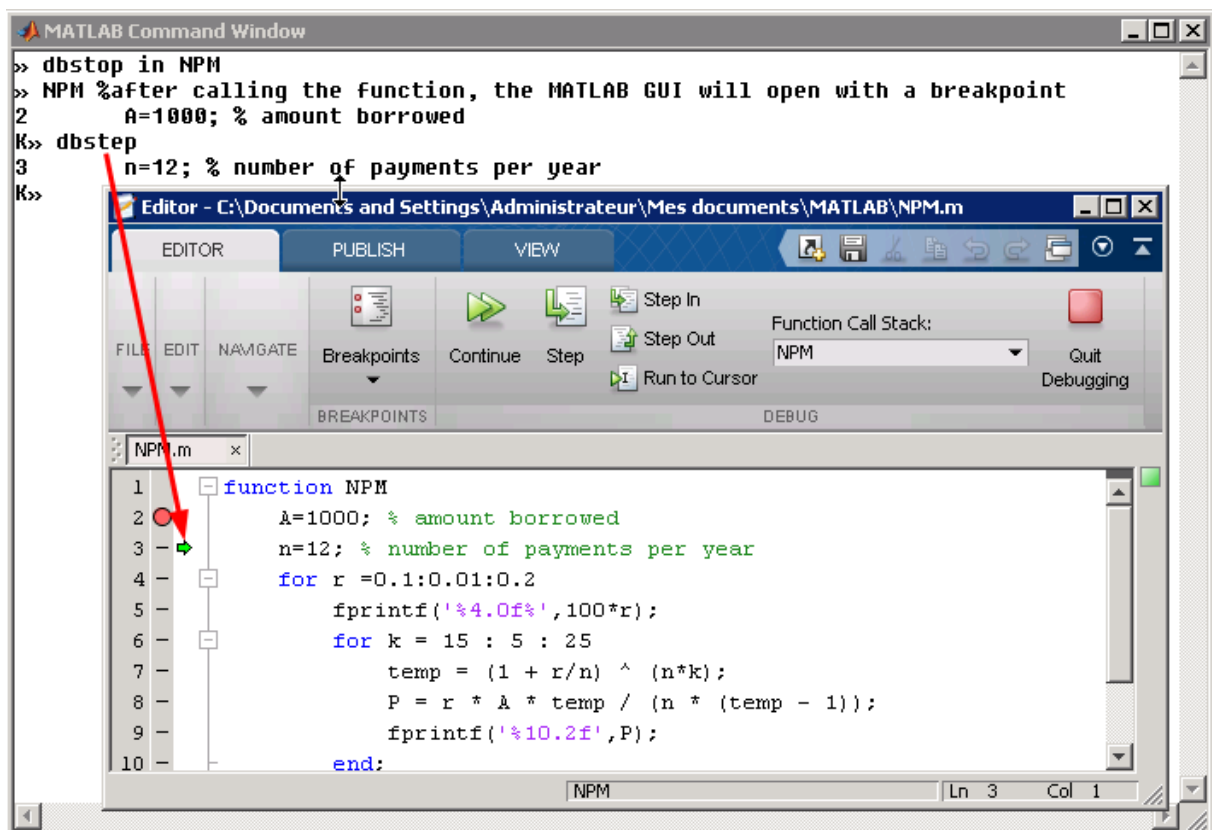
NPM.m x

```

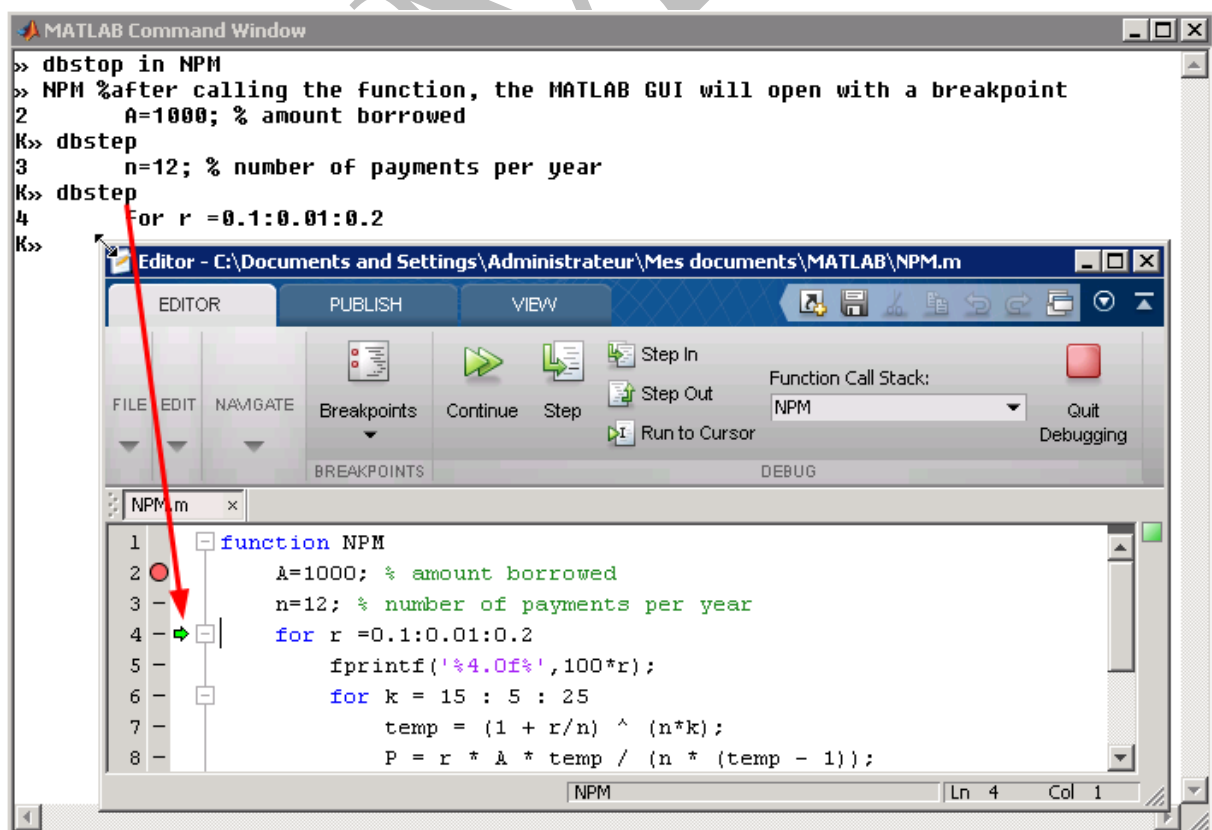
1      function NPM
2      A=1000; % amount borrowed
3      n=12; % number of payments per year
4      for r = 0.1:0.01:0.2
5          fprintf('%4.0f%',100*r);
6          for k = 15 : 5 : 25
7              temp = (1 + r/n) ^ (n*k);
8              P = r * A * temp / (n * (temp - 1));
9              fprintf('%10.2f',P);
10         end;
11         fprintf( '\n' ); % new line
  
```

2 usages of "A" found | NPM | Ln 2 Col 5

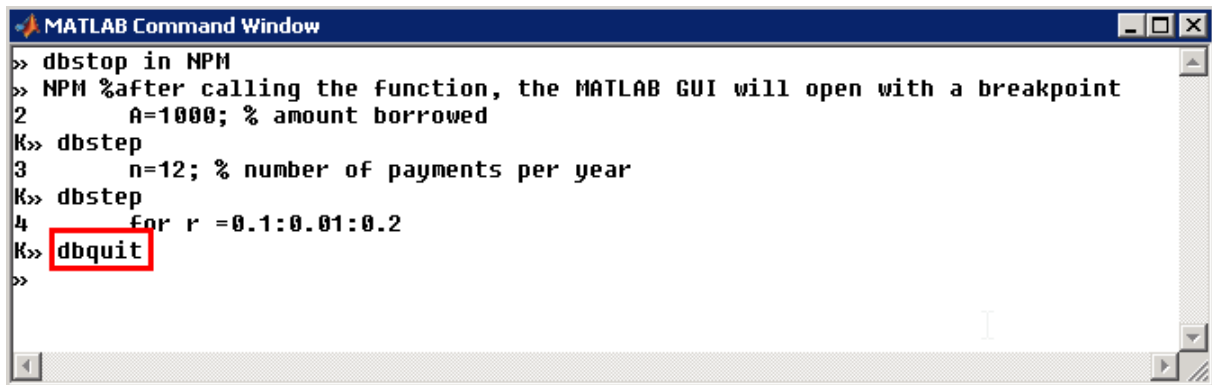
To go further we write **dbstep** and the next line of the function will be executed and show:



and so on...:

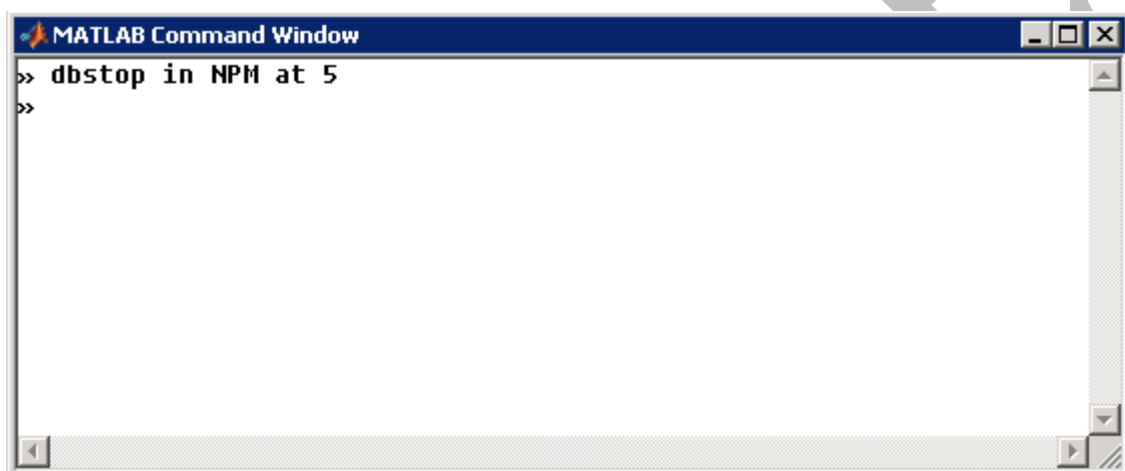


And if we want to stop the debugging and the script we use the command **dbquit**:



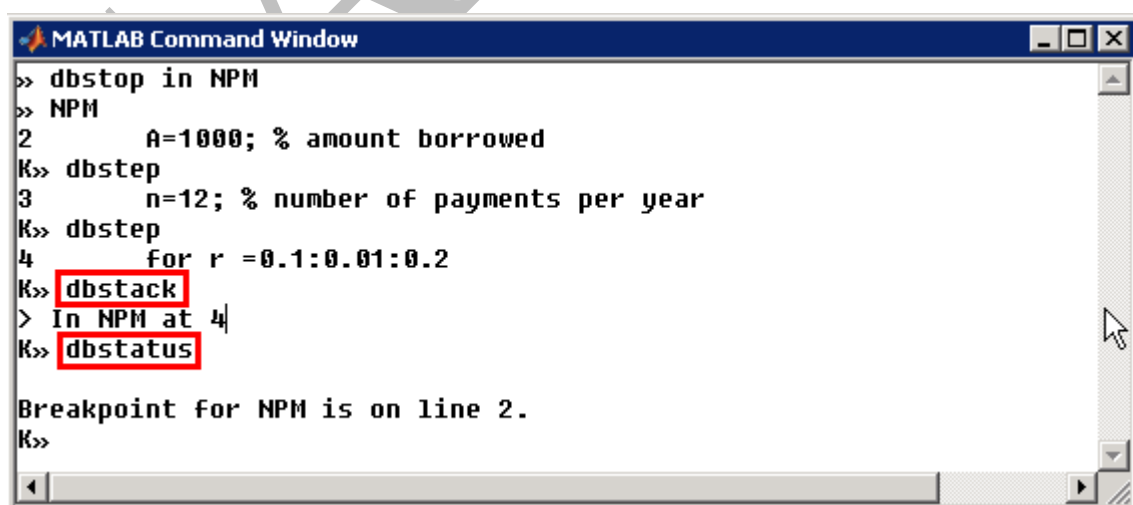
```
MATLAB Command Window
>> dbstop in NPM
>> NPM %after calling the function, the MATLAB GUI will open with a breakpoint
2      A=1000; % amount borrowed
K>> dbstep
3      n=12; % number of payments per year
K>> dbstep
4      for r = 0.1:0.01:0.2
K>> dbquit
>>
```

You can also put a break point at a given desired line:



```
MATLAB Command Window
>> dbstop in NPM at 5
>>
```

The command **dbstack** gives you the possibility to know where you are in the debugging of your script by returning the line number and **dbstatus** returns you the line where the first breakpoint has started the debugging:

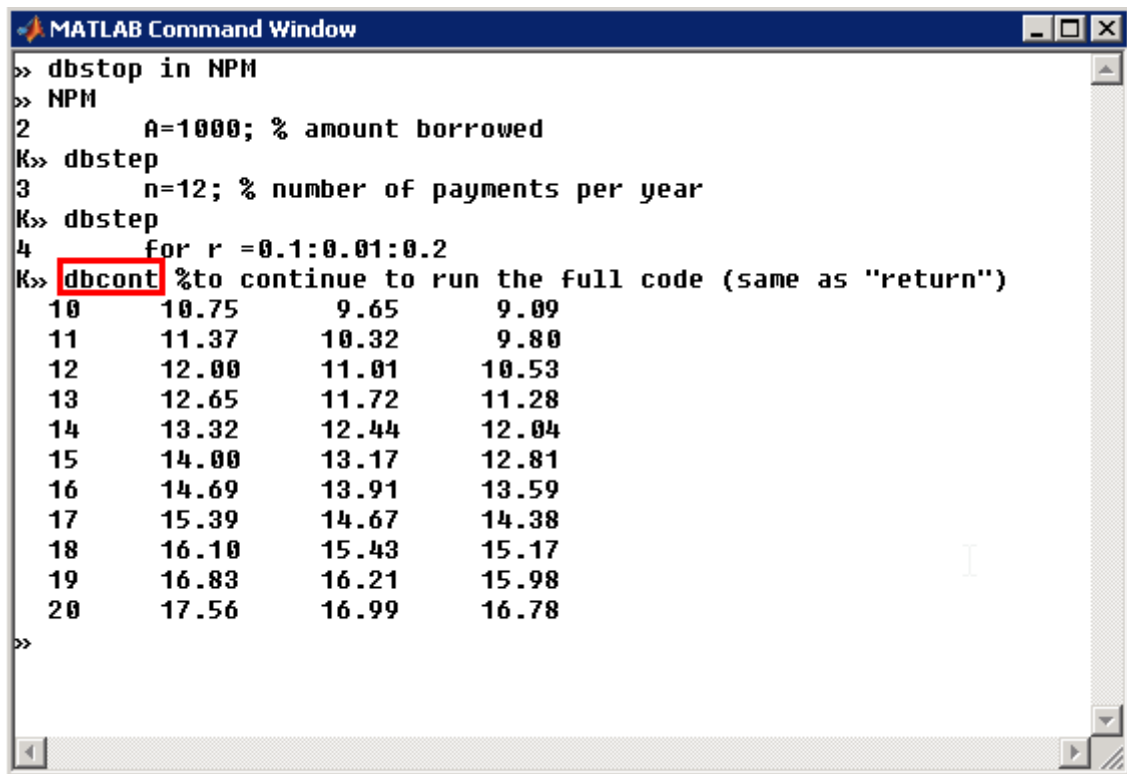


```
MATLAB Command Window
>> dbstop in NPM
>> NPM
2      A=1000; % amount borrowed
K>> dbstep
3      n=12; % number of payments per year
K>> dbstep
4      for r = 0.1:0.01:0.2
K>> dbstack
> In NPM at 4
K>> dbstatus

Breakpoint for NPM is on line 2.
K>>
```

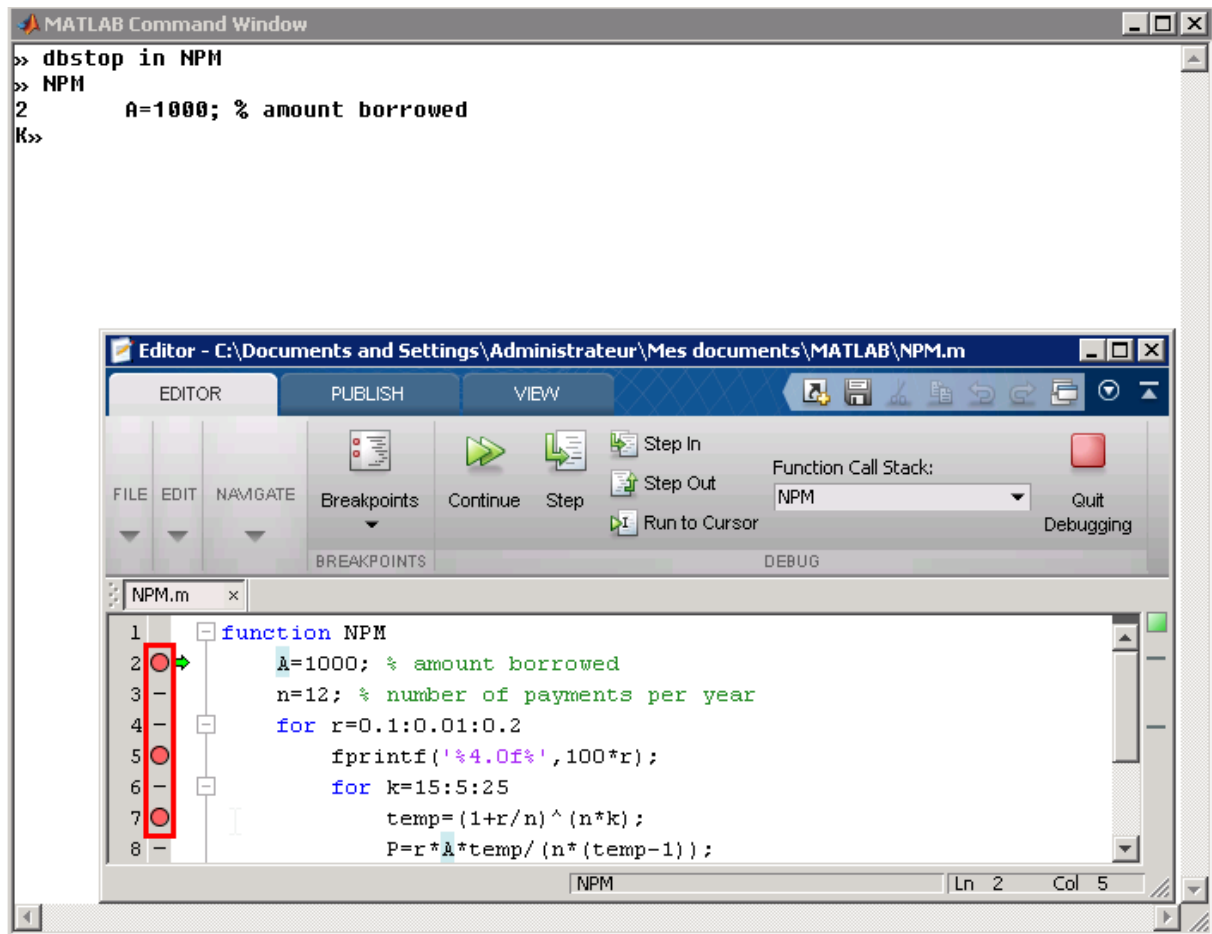
This two last function are not used a lot.

Now when you debug a script you want sometimes to run the script until the end without stopping it again. Then you just have to use the command **dbcont**:

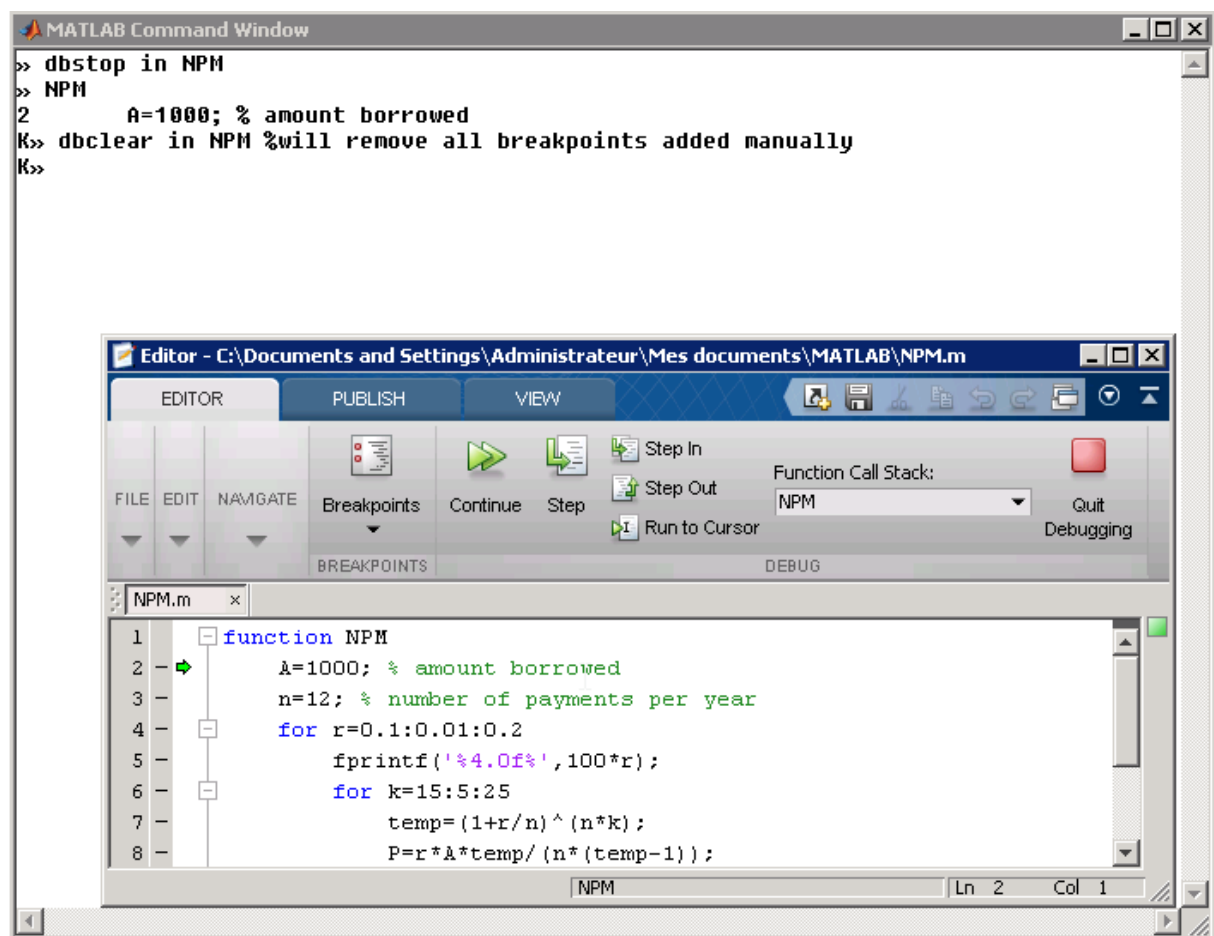


```
MATLAB Command Window
>> dbstop in NPM
>> NPM
2      A=1000; % amount borrowed
K>> dbstep
3      n=12; % number of payments per year
K>> dbstep
4      for r = 0.1:0.01:0.2
K>> dbcont %to continue to run the full code (same as "return")
10     10.75     9.65     9.09
11     11.37     10.32     9.80
12     12.00     11.01    10.53
13     12.65     11.72    11.28
14     13.32     12.44    12.04
15     14.00     13.17    12.81
16     14.69     13.91    13.59
17     15.39     14.67    14.38
18     16.10     15.43    15.17
19     16.83     16.21    15.98
20     17.56     16.99    16.78
>>
```

Now have a closer look to the breakpoint that **dbstop** add automatically on line number 2 of the script and let us add other breakpoints manually as you can see here below:



If you use the command **dbclear**, all breakpoints will be deleted:

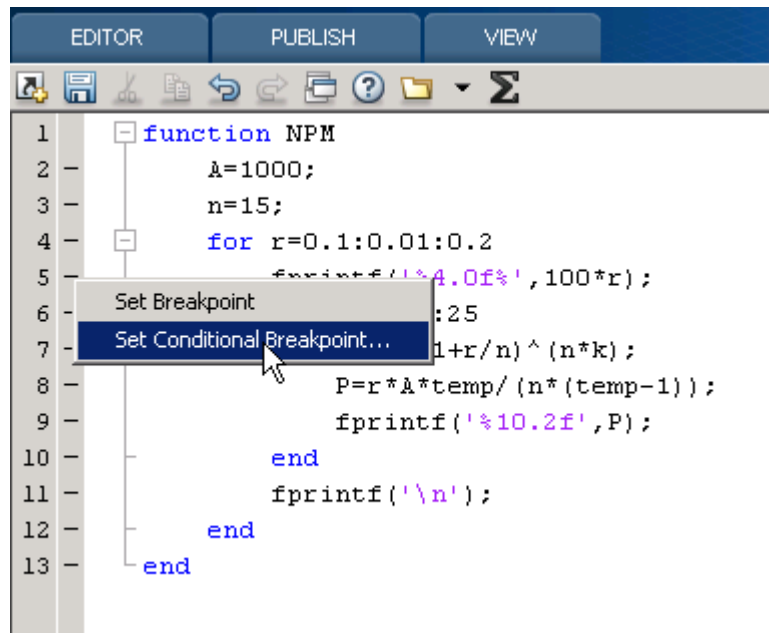


You can use the **s=dbstatus** syntax and then save **s** to save the current breakpoints to a MAT-file. At a later time, you can load **s** and restore the breakpoints using **dbstop(s)**.

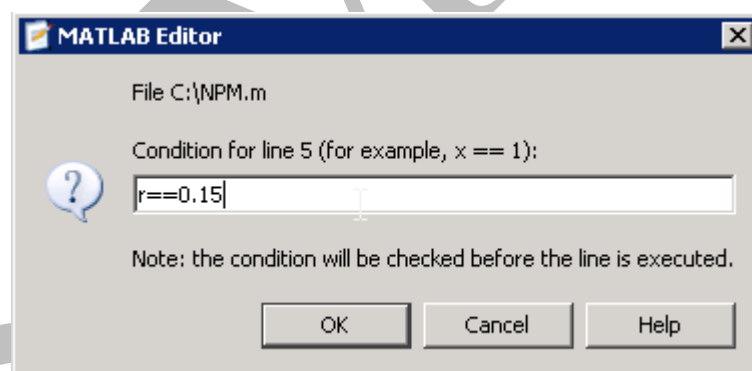
Add conditional breakpoints

This is in my point of view the most important debugging manipulation. To define such a breakpoint proceed as following:

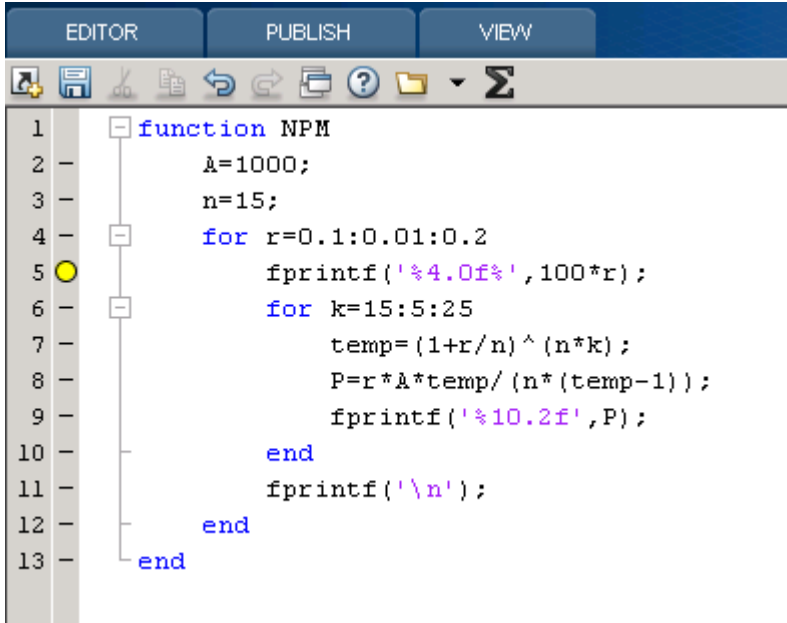
Make a right click at the line level where you want a condition breakpoint:



Then you type what you want:



And when you validate by **OK**, you get a yellow breakpoint:



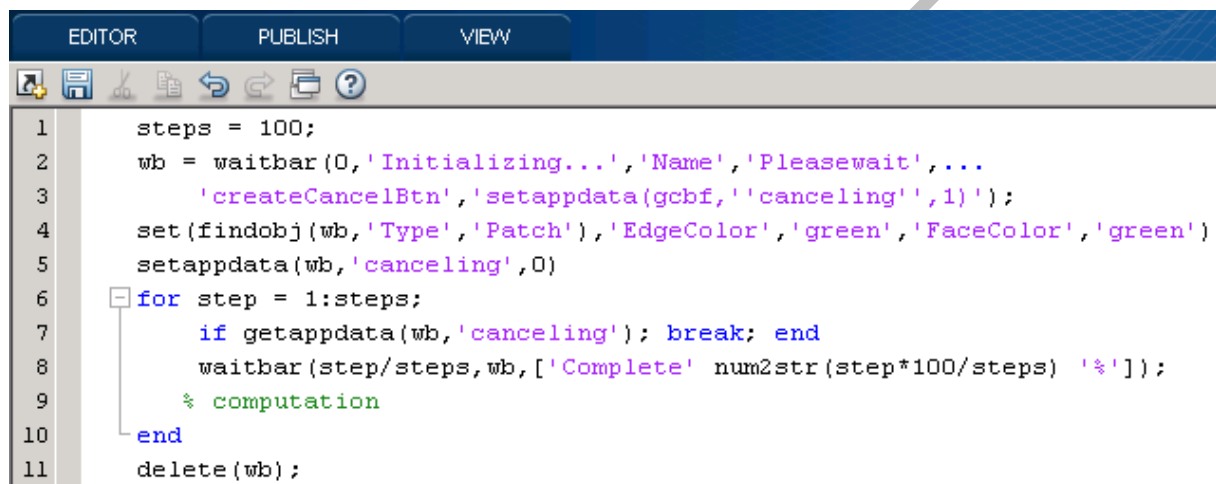
```
1 function NPM
2     A=1000;
3     n=15;
4     for r=0.1:0.01:0.2
5         fprintf('%4.0f%',100*r);
6         for k=15:5:25
7             temp=(1+r/n)^(n*k);
8             P=r*A*temp/(n*(temp-1));
9             fprintf('%10.2f',P);
10        end
11        fprintf('\n');
12    end
13 end
```

Create a more complex wait bar

A lot of MATLAB users and some developers... absolutely wants a progress bar even if it slow down dramatically the running time of the a script as we already saw it (see page 194).

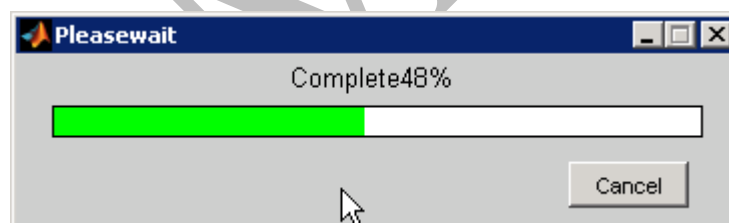
The problem that we saw the first time with the **waitbar()** function is that it is very limited because we could not change the color of the progress bar and not add a button to cancel the whole running script.

Now that we are studying scripts it is time to introduce a more complex **waitbar()** and as you can see below, the syntax is not very intuitive:



```
1  steps = 100;
2  wb = waitbar(0, 'Initializing...', 'Name', 'Pleasewait', ...
3      'createCancelBtn', 'setappdata(gcf, 'canceling', 1)');
4  set(findobj(wb, 'Type', 'Patch'), 'EdgeColor', 'green', 'FaceColor', 'green')
5  setappdata(wb, 'canceling', 0)
6  for step = 1:steps;
7      if getappdata(wb, 'canceling'); break; end
8      waitbar(step/steps, wb, ['Complete' num2str(step*100/steps) '%']);
9      % computation
10 end
11 delete(wb);
```

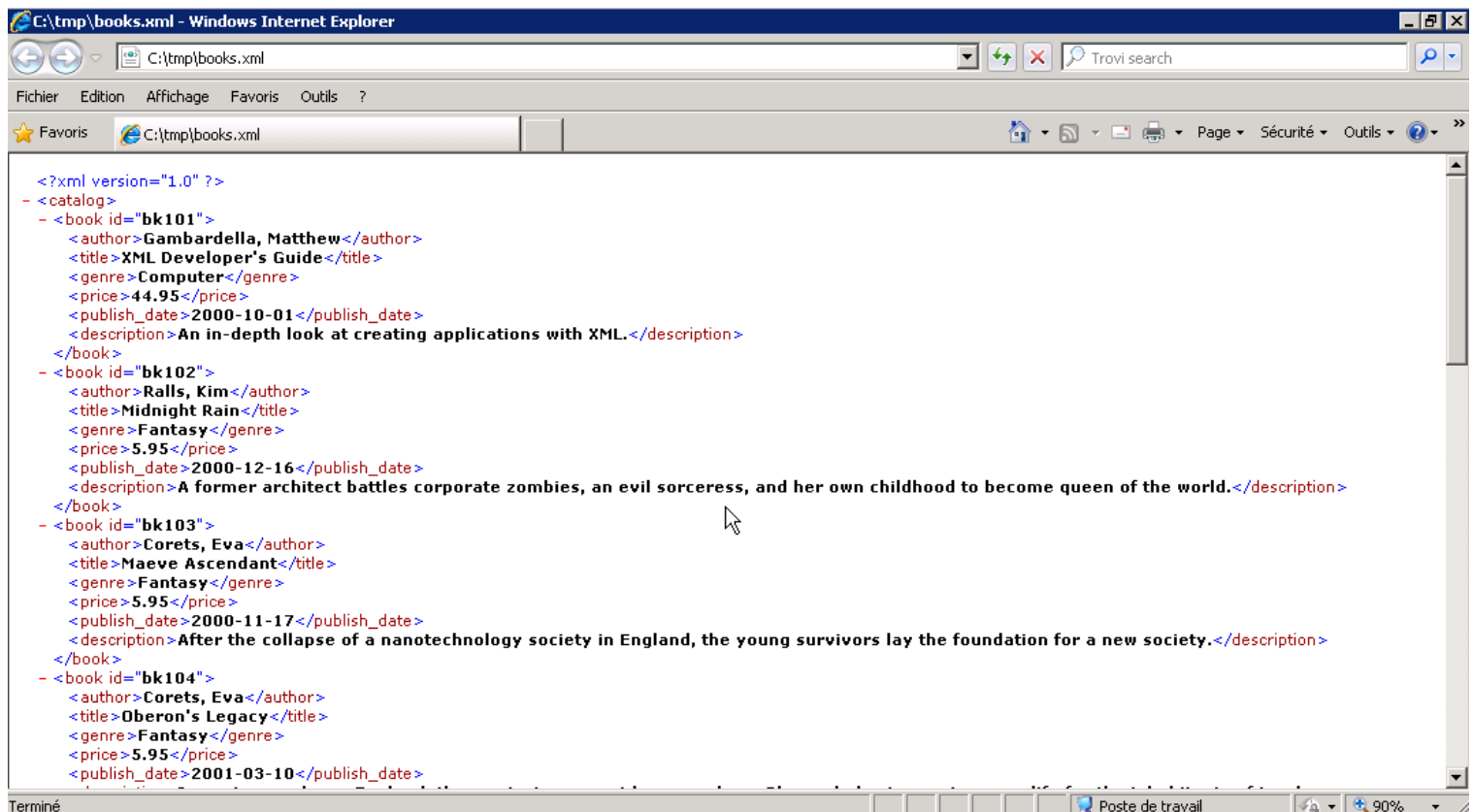
This will result in:



Import an XML file

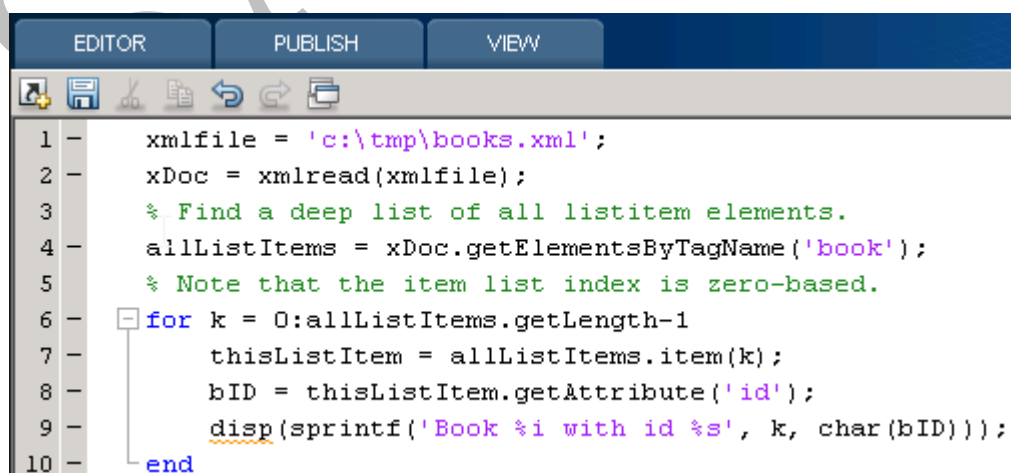
Importing XML feed is something always more and more frequent in computing developments. Because the MATLAB import wizard is not able at this day to import XML files we have waited until now to introduce the script to do such a job.

For this purpose consider the following typical XML file:

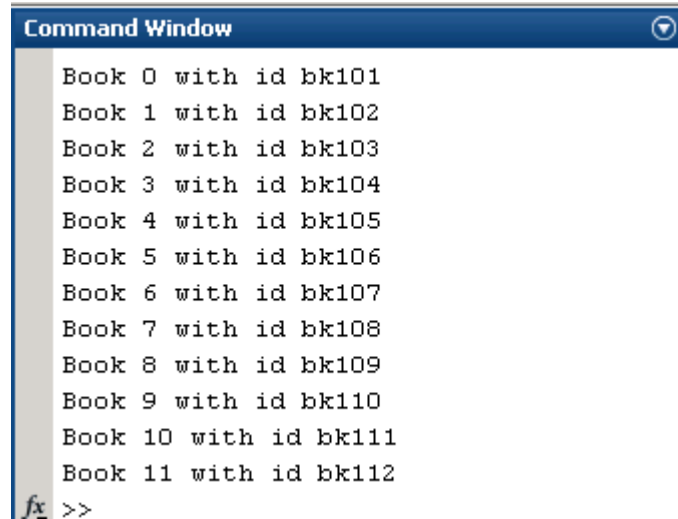


And the following script:

xmlfile='c:\tmp



The result will be:

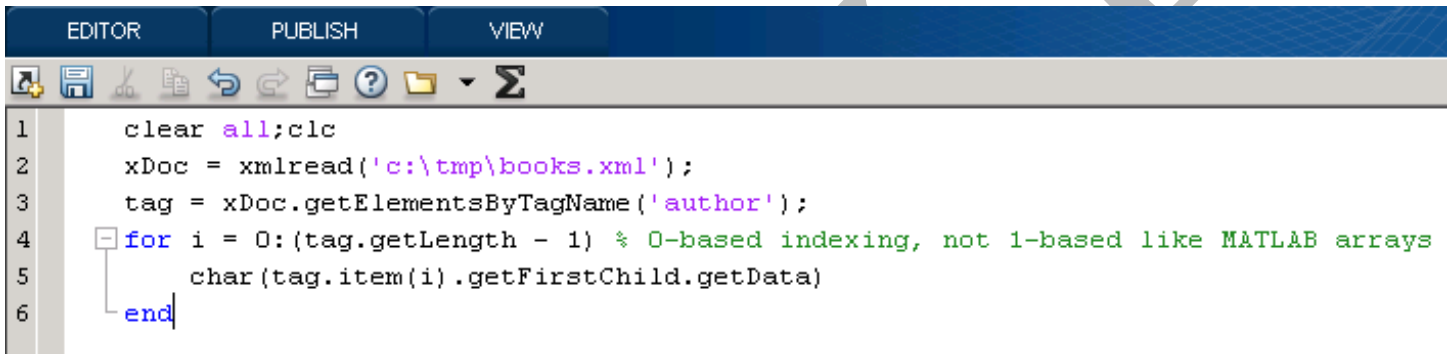


```
Command Window

Book 0 with id bk101
Book 1 with id bk102
Book 2 with id bk103
Book 3 with id bk104
Book 4 with id bk105
Book 5 with id bk106
Book 6 with id bk107
Book 7 with id bk108
Book 8 with id bk109
Book 9 with id bk110
Book 10 with id bk111
Book 11 with id bk112

fx >>
```

And now to read tag values and not only attributes:



```
EDITOR PUBLISH VIEW

1 clear all;clc
2 xDoc = xmlread('c:\tmp\books.xml');
3 tag = xDoc.getElementsByTagName('author');
4 for i = 0:(tag.getLength - 1) % 0-based indexing, not 1-based like MATLAB arrays
5     char(tag.item(i).getFirstChild.getData)
6 end
```

It gives as expected:

```
Command Window

ans =

Gambardella, Matthew

ans =

Ralls, Kim

ans =

Corets, Eva

ans =

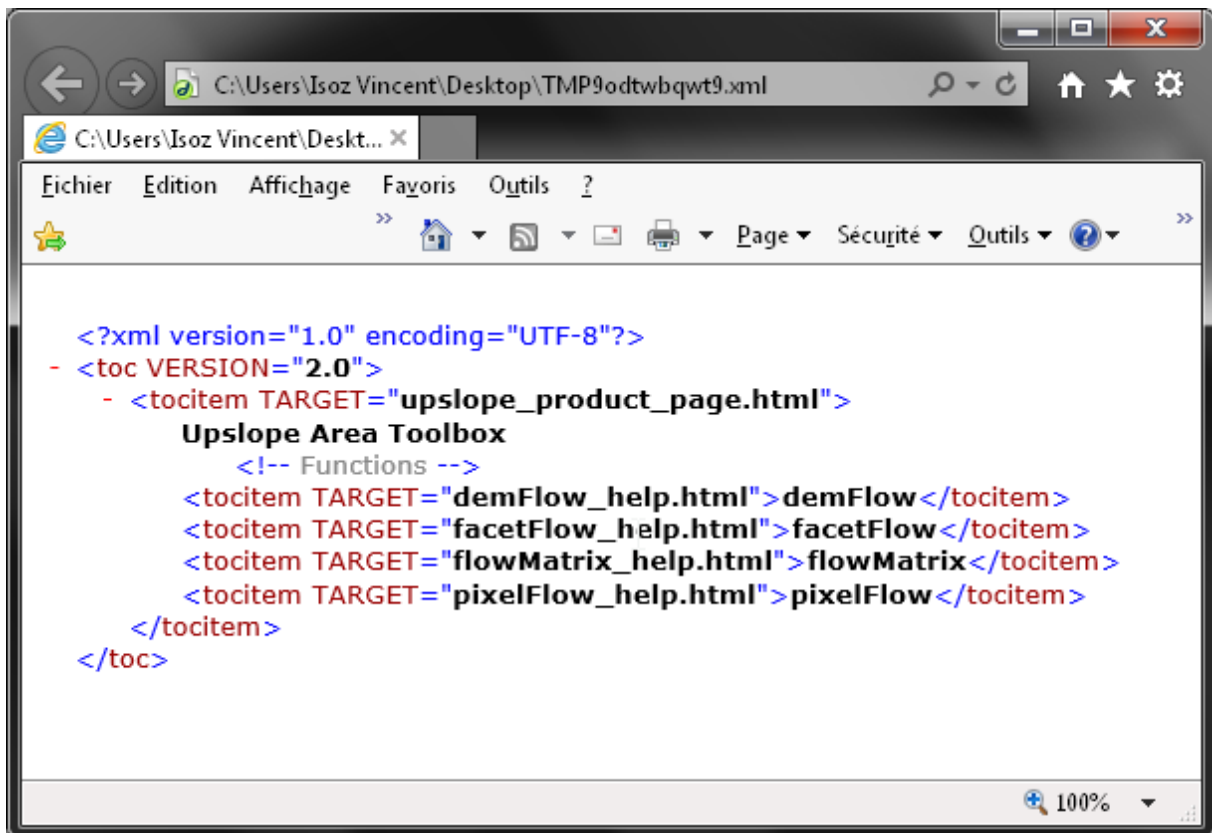
Corets, Eva

ans =

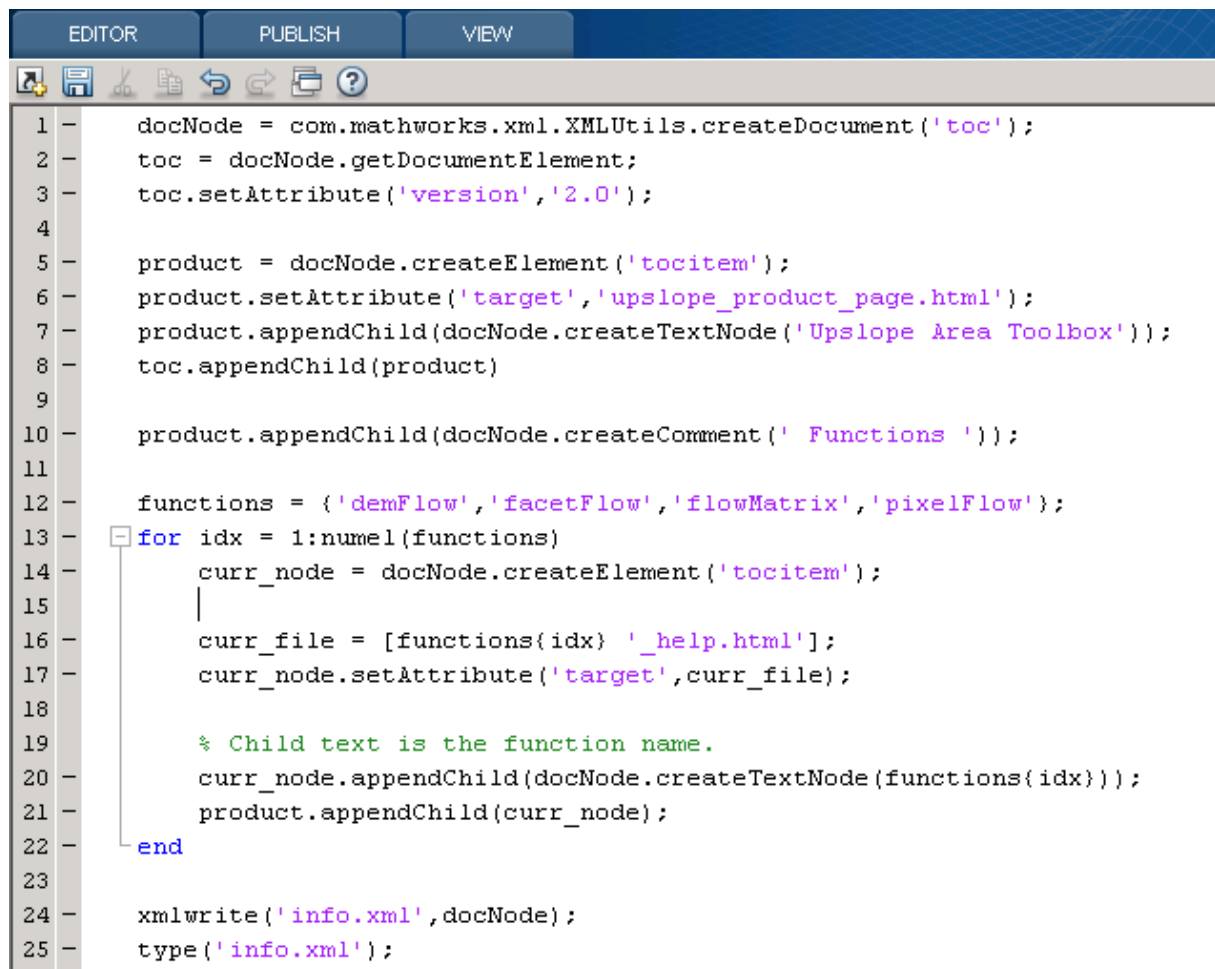
fx Corets, Eva
```

Export an XML file

Suppose we want to export some datas to create the following XML file:



The script will be the following using `xmlwrite()`:



The image shows a screenshot of the MATLAB Editor window. The window has a dark blue header bar with three tabs: 'EDITOR', 'PUBLISH', and 'VIEW'. Below the header is a toolbar with icons for file operations (new, open, save, print, copy, paste, undo, redo, zoom in, zoom out, help) and a search icon. The main area displays a MATLAB script with line numbers 1 through 25 on the left. The script is for creating an XML document for a table of contents. It starts by creating a document element 'toc' and setting its version to '2.0'. Then, it creates a 'tocitem' element 'product' and sets its 'target' attribute to 'upslope_product_page.html'. It appends a text node 'Upslope Area Toolbox' to 'product' and appends 'product' to the 'toc' element. Next, it appends a comment ' Functions ' to 'product'. Then, it defines a list of functions: 'demFlow', 'facetFlow', 'flowMatrix', and 'pixelFlow'. It enters a 'for' loop to iterate over these functions. For each function, it creates a 'tocitem' element 'curr_node', sets its 'target' attribute to the function name followed by '_help.html', appends a text node with the function name to 'curr_node', and appends 'curr_node' to 'product'. The loop ends with 'end'. Finally, it writes the XML document to 'info.xml' and displays its type.

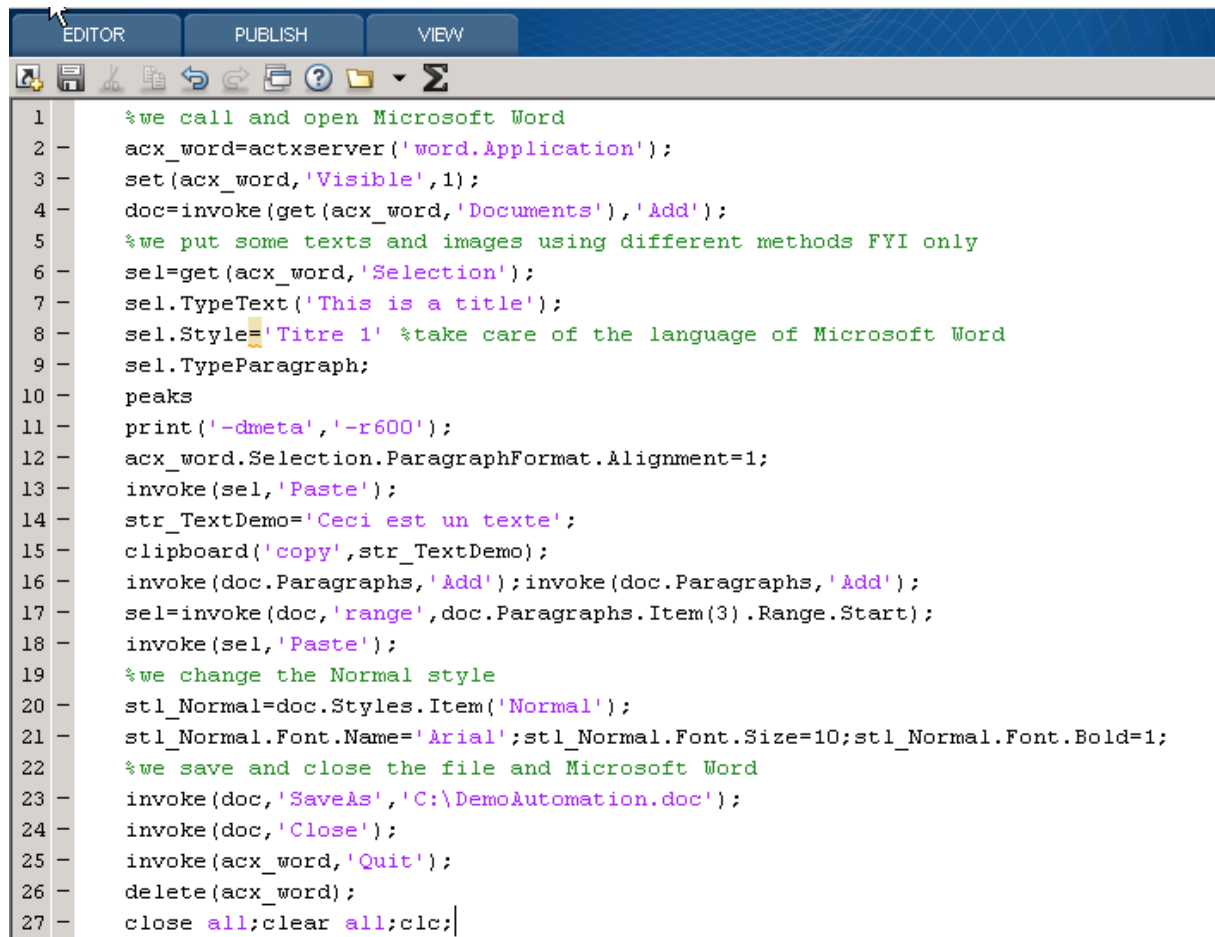
```
1 - docNode = com.mathworks.xml.XMLUtils.createDocument('toc');
2 - toc = docNode.getDocumentElement;
3 - toc.setAttribute('version','2.0');
4 -
5 - product = docNode.createElement('tocitem');
6 - product.setAttribute('target','upslope_product_page.html');
7 - product.appendChild(docNode.createTextNode('Upslope Area Toolbox'));
8 - toc.appendChild(product)
9 -
10 - product.appendChild(docNode.createComment(' Functions '));
11 -
12 - functions = {'demFlow','facetFlow','flowMatrix','pixelFlow'};
13 - for idx = 1:numel(functions)
14 -     curr_node = docNode.createElement('tocitem');
15 -     |
16 -     curr_file = [functions{idx} '_help.html'];
17 -     curr_node.setAttribute('target',curr_file);
18 -
19 -     % Child text is the function name.
20 -     curr_node.appendChild(docNode.createTextNode(functions{idx}));
21 -     product.appendChild(curr_node);
22 - end
23 -
24 - xmlwrite('info.xml',docNode);
25 - type('info.xml');
```

MATLAB Automation

Generate a Microsoft Word report

Generating automated Microsoft Word report is something very common.

Here is an example of most common Microsoft Word Automation tasks:

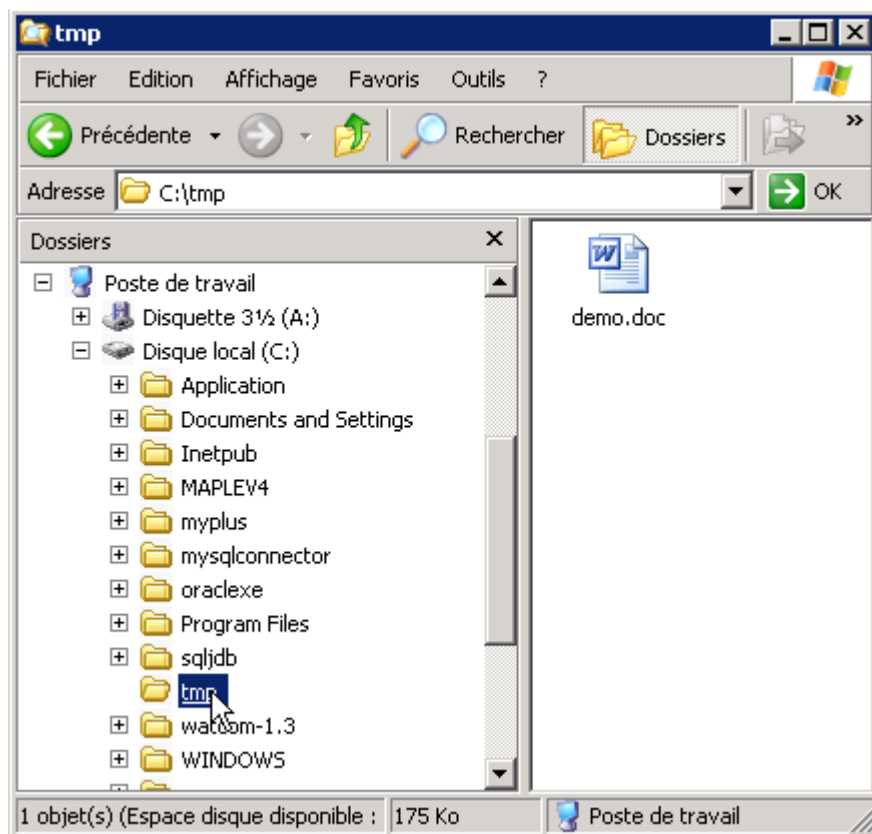


```

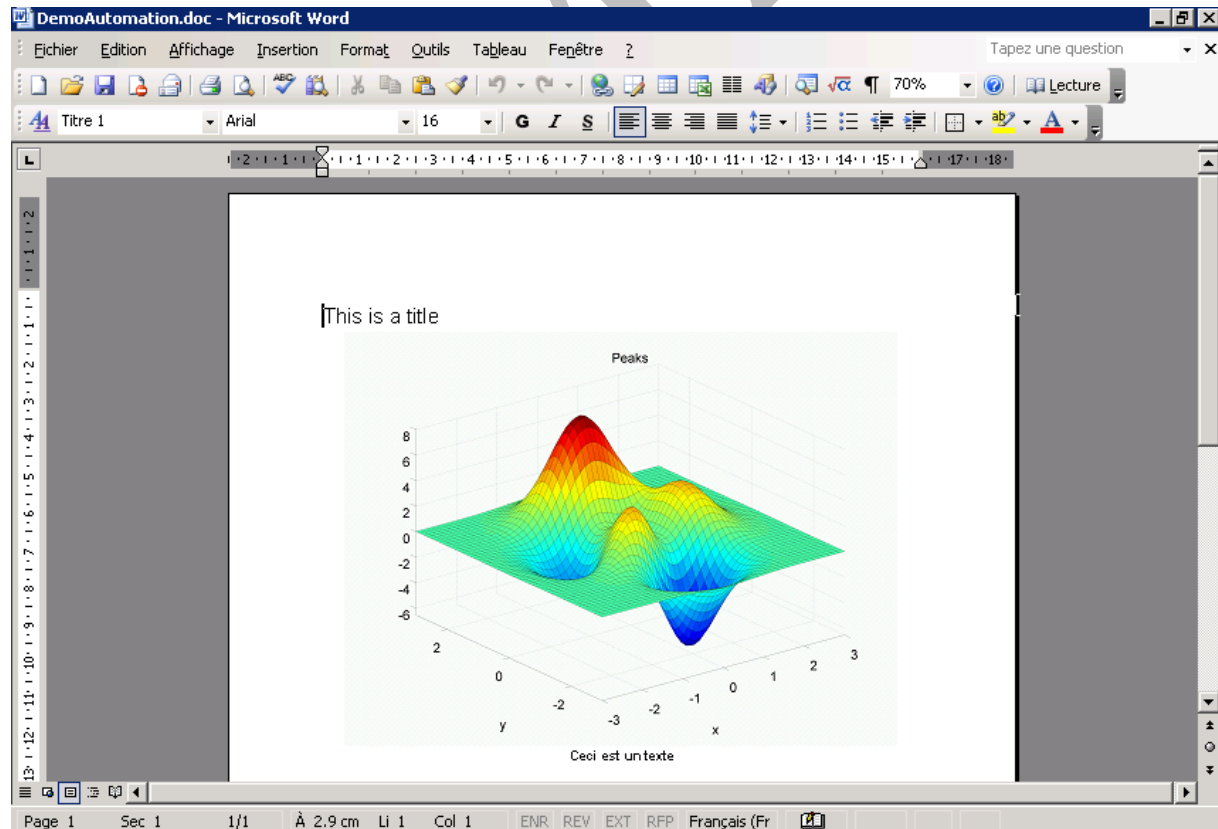
1  %we call and open Microsoft Word
2  - acx_word=actxserver('word.Application');
3  - set(acx_word,'Visible',1);
4  - doc=invoke(get(acx_word,'Documents'),'Add');
5  %we put some texts and images using different methods FYI only
6  - sel=get(acx_word,'Selection');
7  - sel.TypeText('This is a title');
8  - sel.Style='Titre 1' %take care of the language of Microsoft Word
9  - sel.TypeParagraph;
10 - peaks
11 - print('-dmeta','-r600');
12 - acx_word.Selection.ParagraphFormat.Alignment=1;
13 - invoke(sel,'Paste');
14 - str_TextDemo='Ceci est un texte';
15 - clipboard('copy',str_TextDemo);
16 - invoke(doc.Paragraphs,'Add');invoke(doc.Paragraphs,'Add');
17 - sel=invoke(doc,'range',doc.Paragraphs.Item(3).Range.Start);
18 - invoke(sel,'Paste');
19 %we change the Normal style
20 - stl_Normal=doc.Styles.Item('Normal');
21 - stl_Normal.Font.Name='Arial';stl_Normal.Font.Size=10;stl_Normal.Font.Bold=1;
22 %we save and close the file and Microsoft Word
23 - invoke(doc,'SaveAs','C:\DemoAutomation.doc');
24 - invoke(doc,'Close');
25 - invoke(acx_word,'Quit');
26 - delete(acx_word);
27 - close all;clear all;clc;

```

This will create the file:



And paste the active MATLAB figure with a legend into the document:

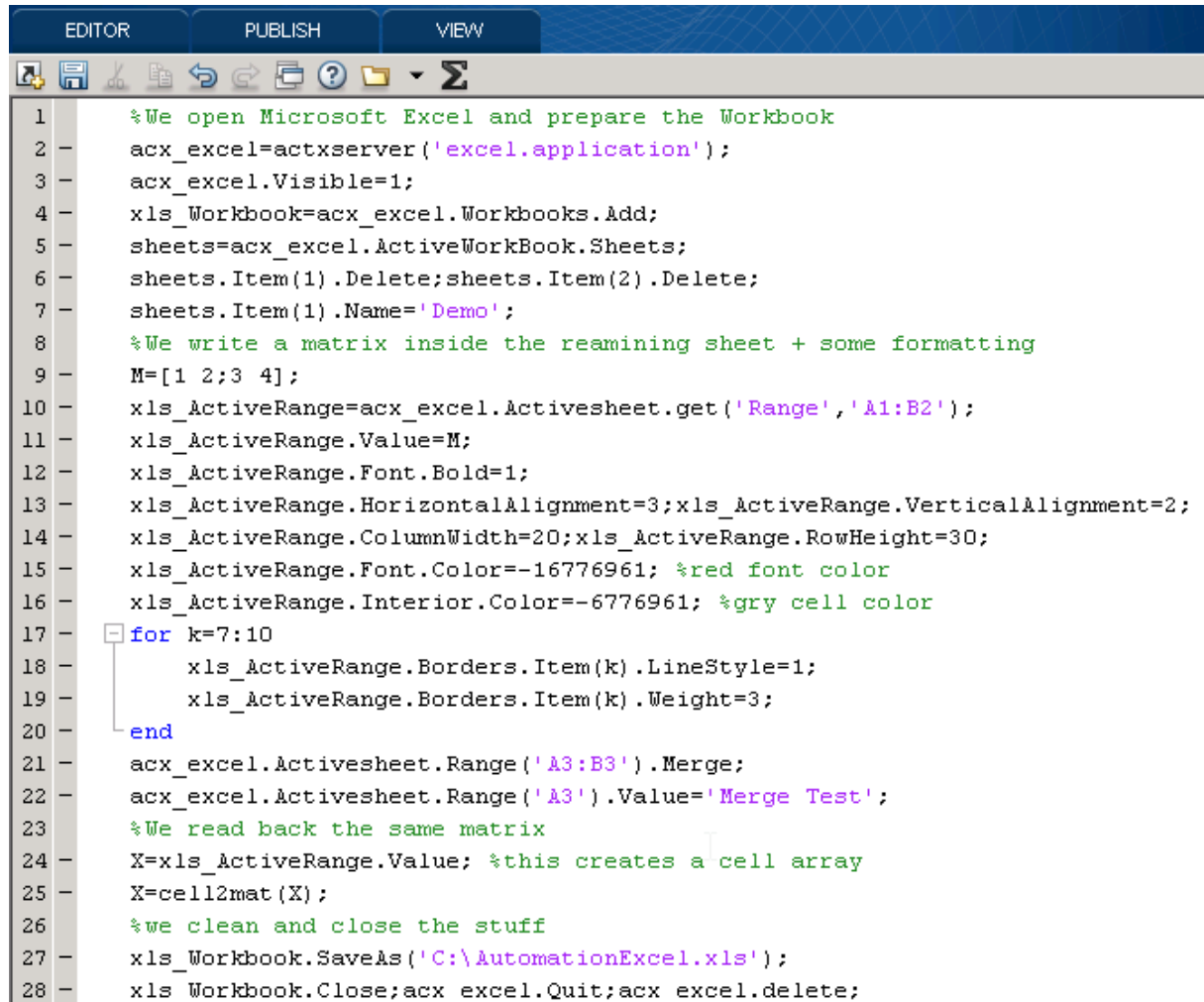


That's it?

Put values into Microsoft Excel

Another very common application of automation is to retrieve MATLAB results into Microsoft Excel.

Here is a Microsoft Excel Automation example with some of the most typical tasks:

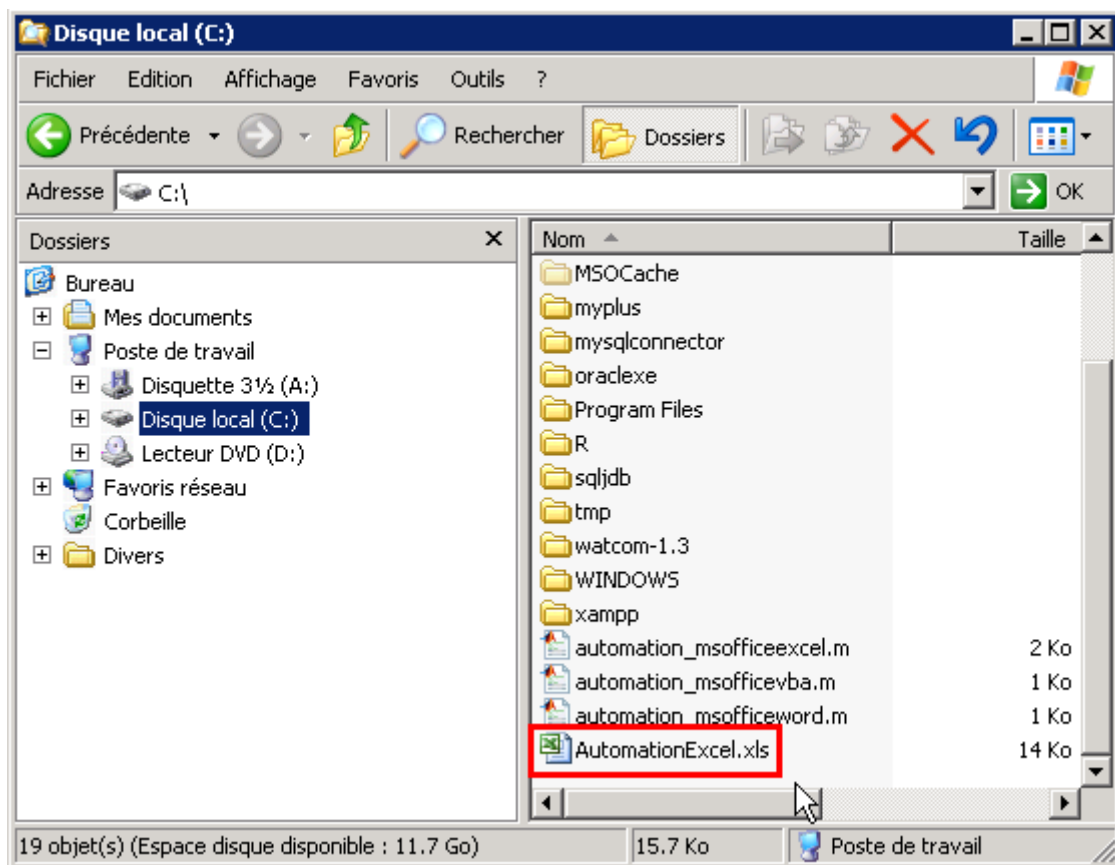


```

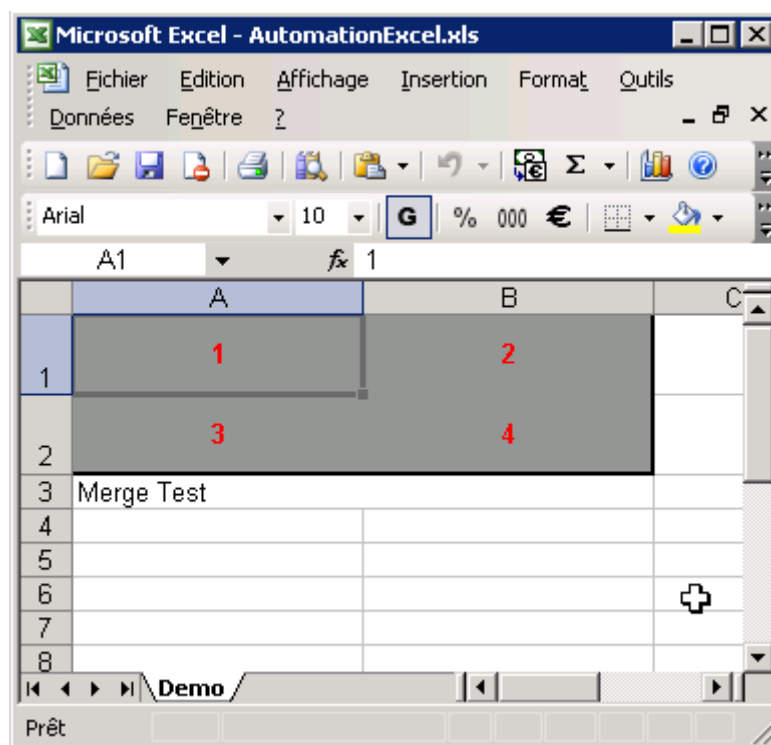
1  %We open Microsoft Excel and prepare the Workbook
2  -  acx_excel=actxserver('excel.application');
3  -  acx_excel.Visible=1;
4  -  xls_Workbook=acx_excel.Workbooks.Add;
5  -  sheets=acx_excel.ActiveWorkBook.Sheets;
6  -  sheets.Item(1).Delete;sheets.Item(2).Delete;
7  -  sheets.Item(1).Name='Demo';
8  %We write a matrix inside the reamining sheet + some formatting
9  -  M=[1 2;3 4];
10 -  xls_ActiveRange=acx_excel.Activesheet.get('Range','A1:B2');
11 -  xls_ActiveRange.Value=M;
12 -  xls_ActiveRange.Font.Bold=1;
13 -  xls_ActiveRange.HorizontalAlignment=3;xls_ActiveRange.VerticalAlignment=2;
14 -  xls_ActiveRange.ColumnWidth=20;xls_ActiveRange.RowHeight=30;
15 -  xls_ActiveRange.Font.Color=-16776961; %red font color
16 -  xls_ActiveRange.Interior.Color=-6776961; %gry cell color
17 -  for k=7:10
18 -      xls_ActiveRange.Borders.Item(k).LineStyle=1;
19 -      xls_ActiveRange.Borders.Item(k).Weight=3;
20 -  end
21 -  acx_excel.Activesheet.Range('A3:B3').Merge;
22 -  acx_excel.Activesheet.Range('A3').Value='Merge Test';
23 %We read back the same matrix
24 -  X=xls_ActiveRange.Value; %this creates a cell array
25 -  X=cell2mat(X);
26 %we clean and close the stuff
27 -  xls_Workbook.SaveAs('C:\AutomationExcel.xls');
28 -  xls_Workbook.Close;acx_excel.Quit;acx_excel.delete;

```

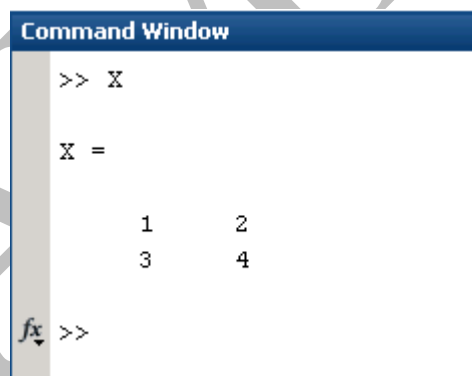
This will create a file as you can see:



With the following content:



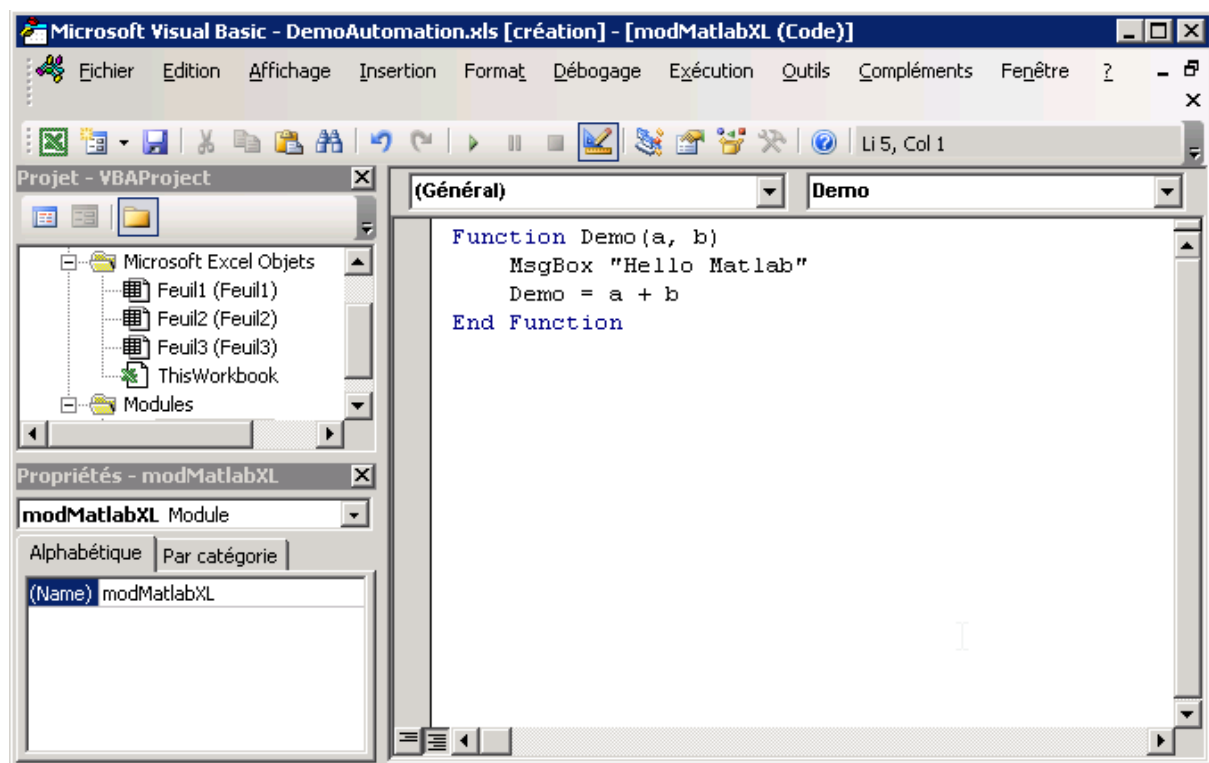
And because our example has also read the values that very into the Microsoft Excel sheet to retrieve them into a MATLAB variable X we have:



Get values from a MS Excel VBA function

A lot of people develop MS Excel functions and because of time and budget constraints financial engineers can't convert VBA functions to MATLAB script. Then an easy way is to call VBA functions from MS Excel.

Consider the following Excel file named *DemoAutomation.xls*, with the module *modMatlabXL* and a function inside *Demo*:

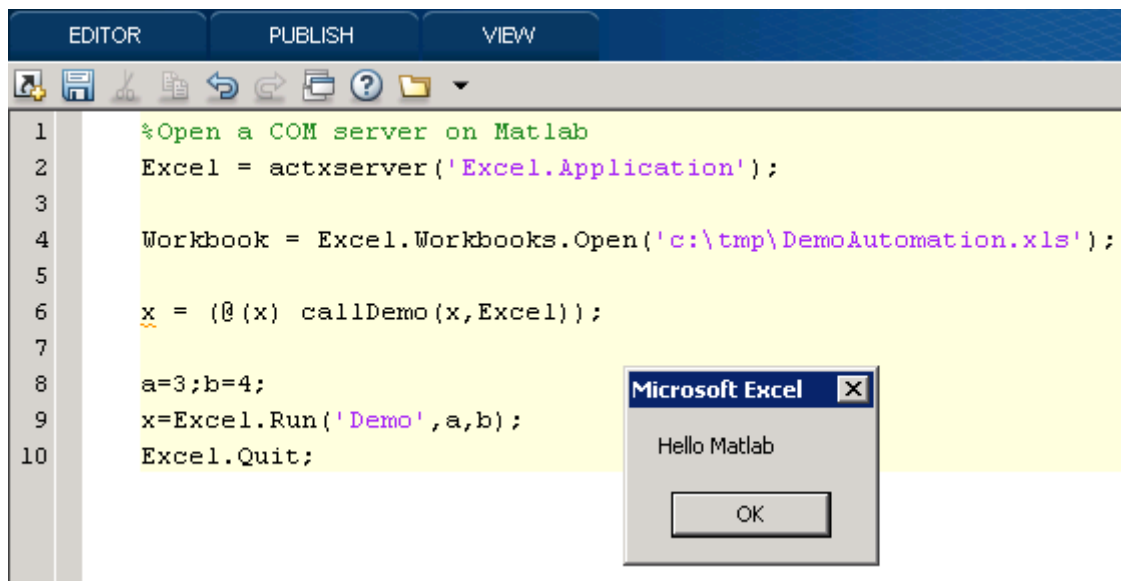


Now the script to call the VBA function from MATLAB is the following:

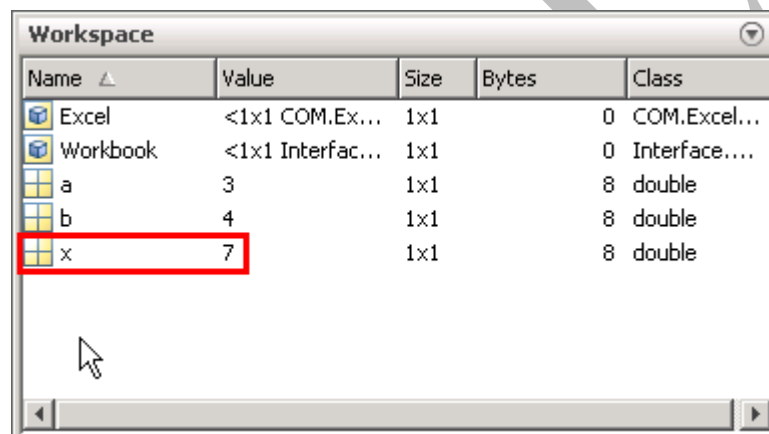
The screenshot shows a MATLAB script editor with the following code:

```
1 %Open a COM server on Matlab
2 Excel = actxserver('Excel.Application');
3
4 Workbook = Excel.Workbooks.Open('c:\tmp\DemoAutomation.xls');
5
6 x = (@(x) callDemo(x,Excel));
7
8 a=3;b=4;
9 x=Excel.Run('Demo',a,b);
10 Excel.Quit;
```

If we run this code, we get:



And in the workspace:

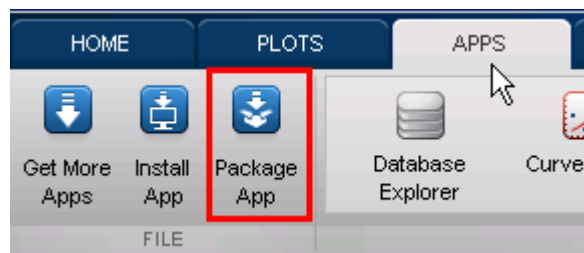


Gotcha!

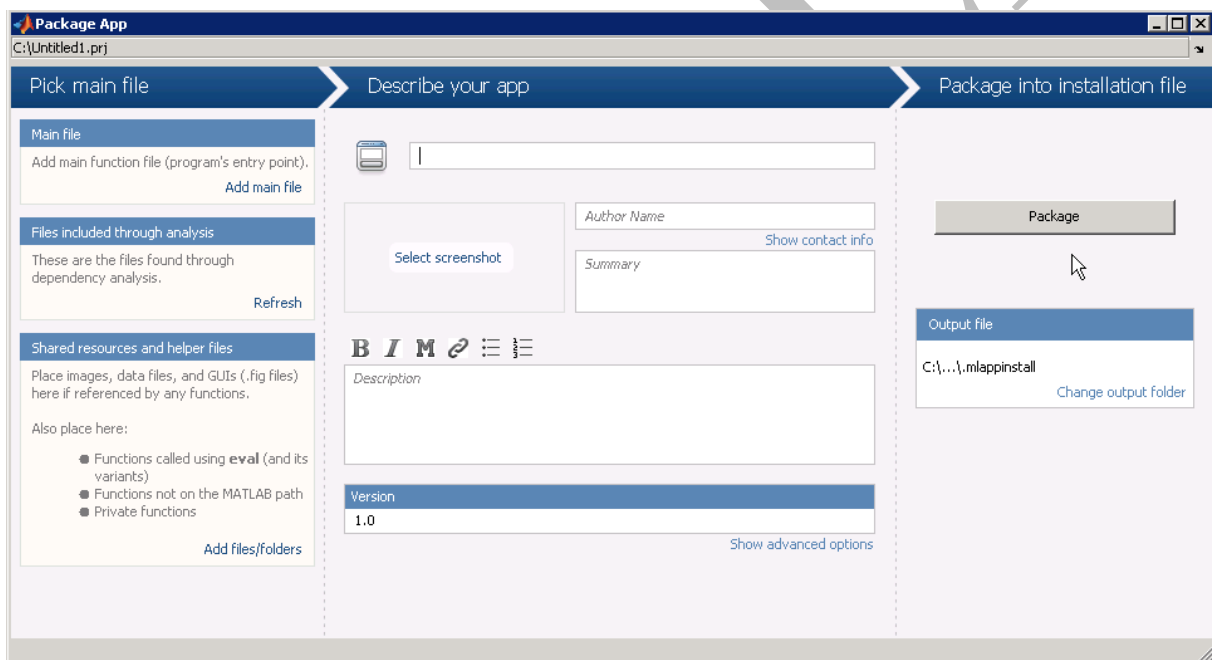
Create a Toolbox (App)

Doesn't matter if you are a professional MATLAB developer or not, is much more elegant to share or sell a complicated MATLAB script as Toolbox (or "App") than as simple multiple *.m script files.

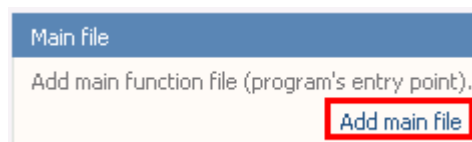
To create a package from your own script you go first then the App ribbons and click on **Package App**:



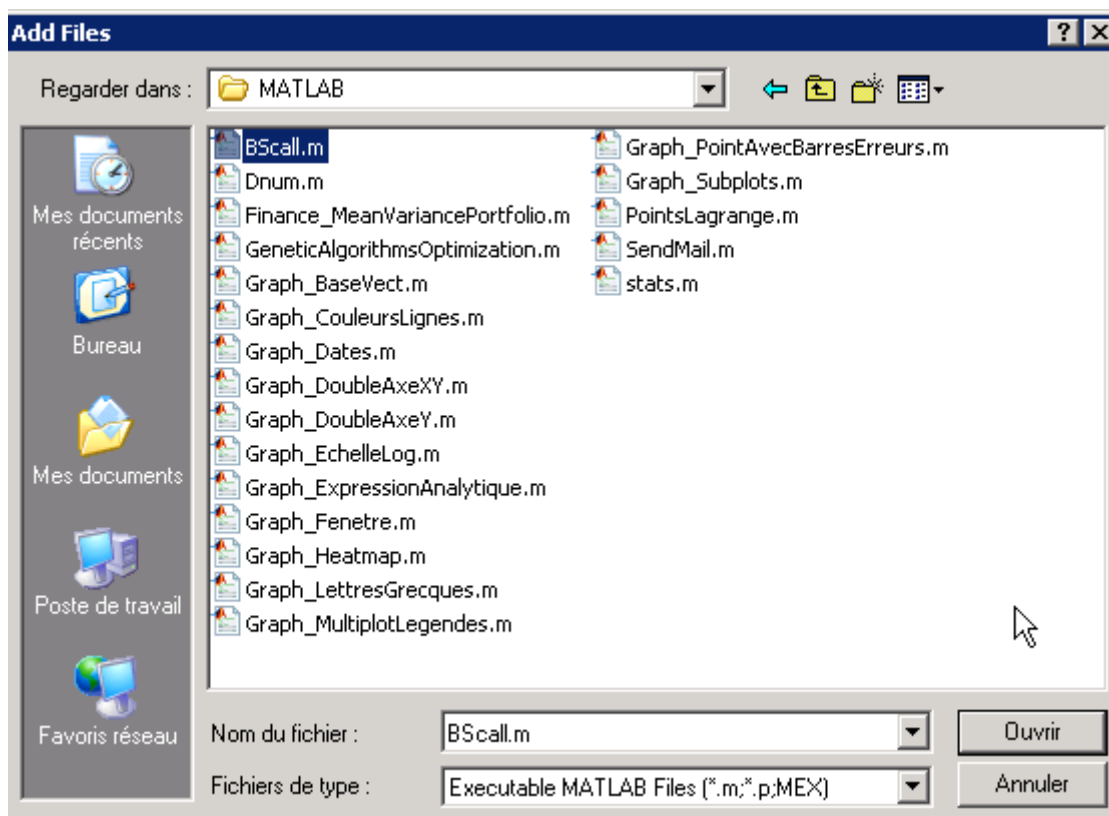
The following wizard will appear on your screen:



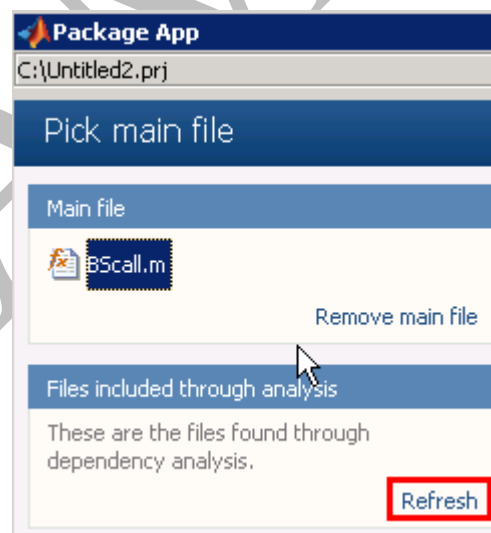
In the upper left corner of the wizard you can see the following. Click on **Add main file**:



Browse for the main file of your script:

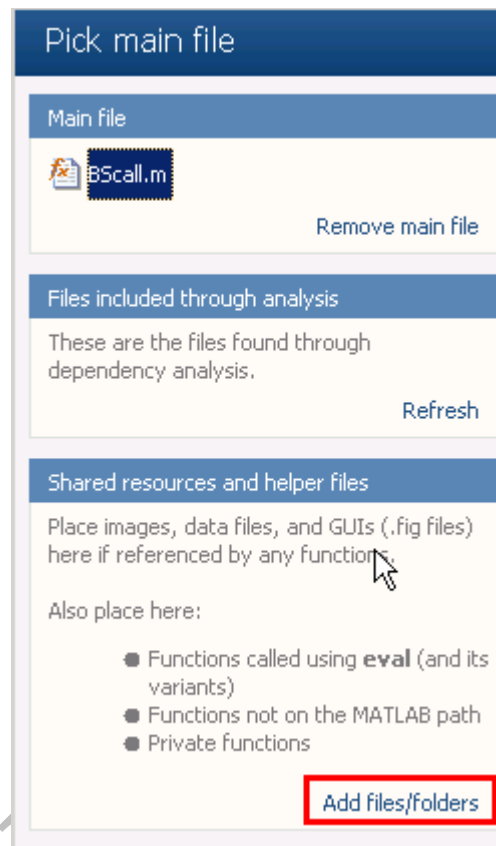


Once this is done you can ask MATLAB to include automatically dependante script files to the main one by clickin on **Refresh**:

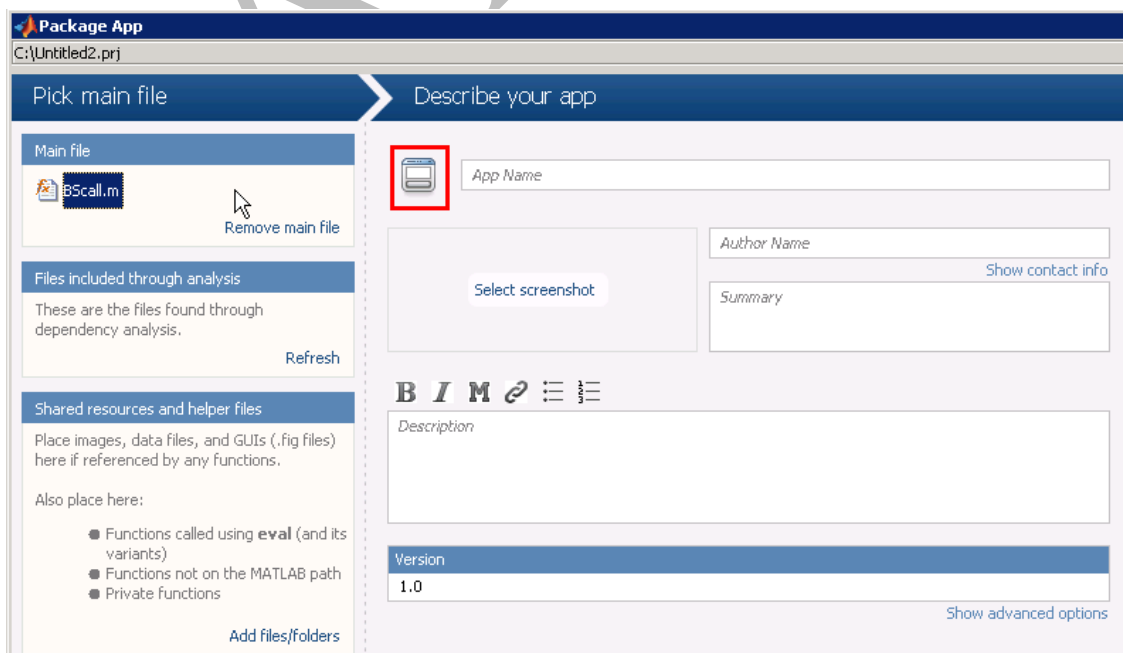


In this case we don't have any dependant files then nothing will appear.

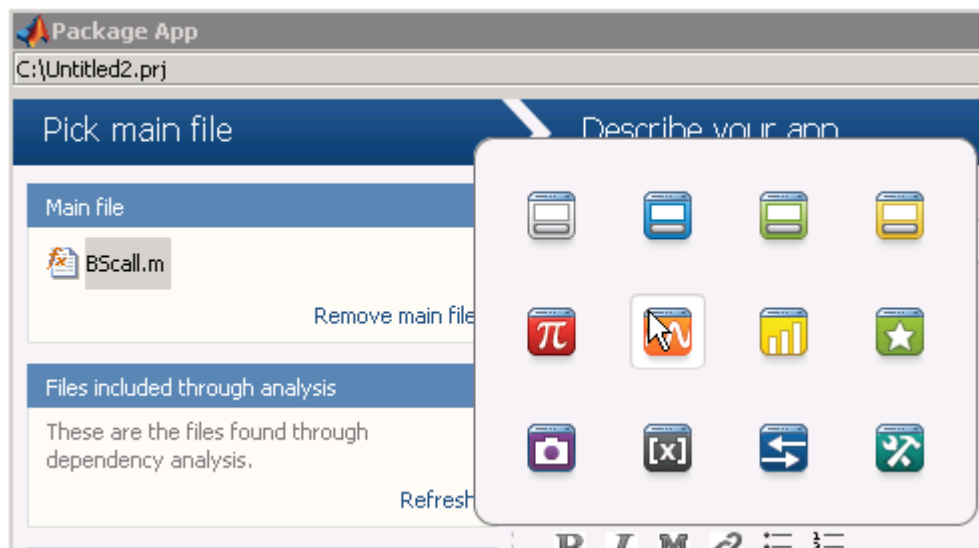
In the third area MATLAB ask you if you have or not any images, data files or GUIs that you would like to associate with you Toolbox:



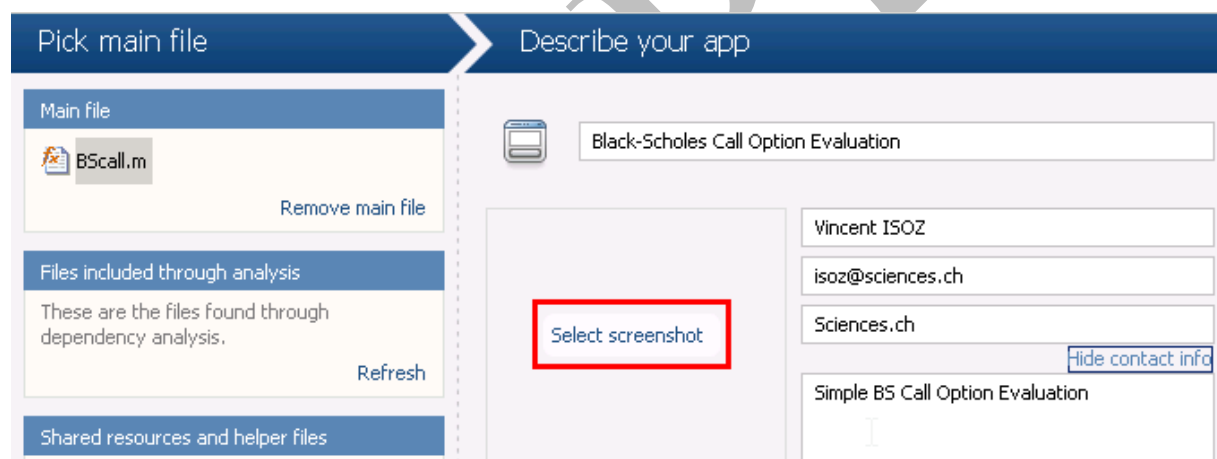
Once the first step finish if you click now on the small icon visible below and rounded by a red rectangle:



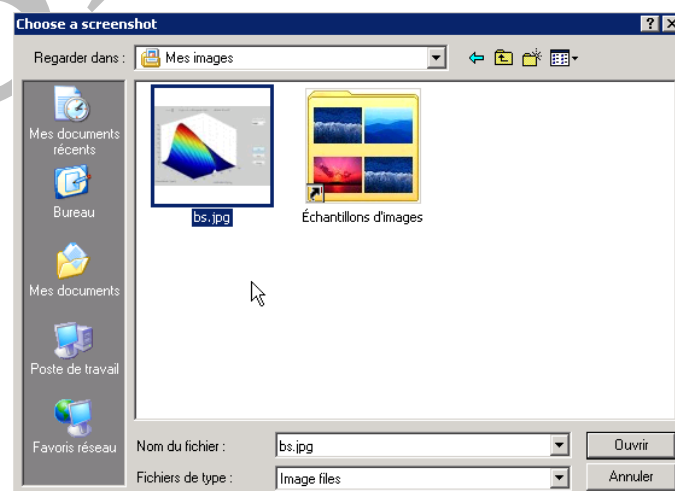
MATLAB ask you which icon you want to use for your application (you can't create your own icon with the wizard):



After this you type a title for your App (Toolbox), write some informations about you and your App (Toolbox) and also select a screenshot as preview example:



If we click on **Select screenshot**, we get:



Once the image selected you get:

Pick main file | **Describe your app**

Main file
 BScall.m
 Remove main file

Files included through analysis
 These are the files found through dependency analysis.
 Refresh

Shared resources and helper files
 Place images, data files, and GUIs (.fig files) here if referenced by any functions.
 Also place here:
 • Functions called using `eval` (and its variants)
 • Functions not on the MATLAB path
 • Private functions
 Add files/folders

App Title: Black-Scholes Call Option Evaluation

App Author: Vincent ISOZ

Email: isoz@sciences.ch

Website: Sciences.ch
 Hide contact info

App Description: Simple BS Call Option Evaluation

Version: 1.0

Show advanced options

If you click on **Show advanced options** you get:

Version: 1.0

Products needed: Statistics Toolbox, MATLAB

Supported platforms: All

Hide advanced options

To finish, we click on the button **Package**:

Package App
 C:\Black-Scholes Call Option Evaluation.pr

Pick main file | **Describe your app** | **Package into installation file**

Main file: BScall.m (Remove main file)

Files included through analysis: Refresh

Shared resources and helper files:

App Title: Black-Scholes Call Option Evaluation

App Author: Vincent ISOZ (App author)

Email: isoz@sciences.ch

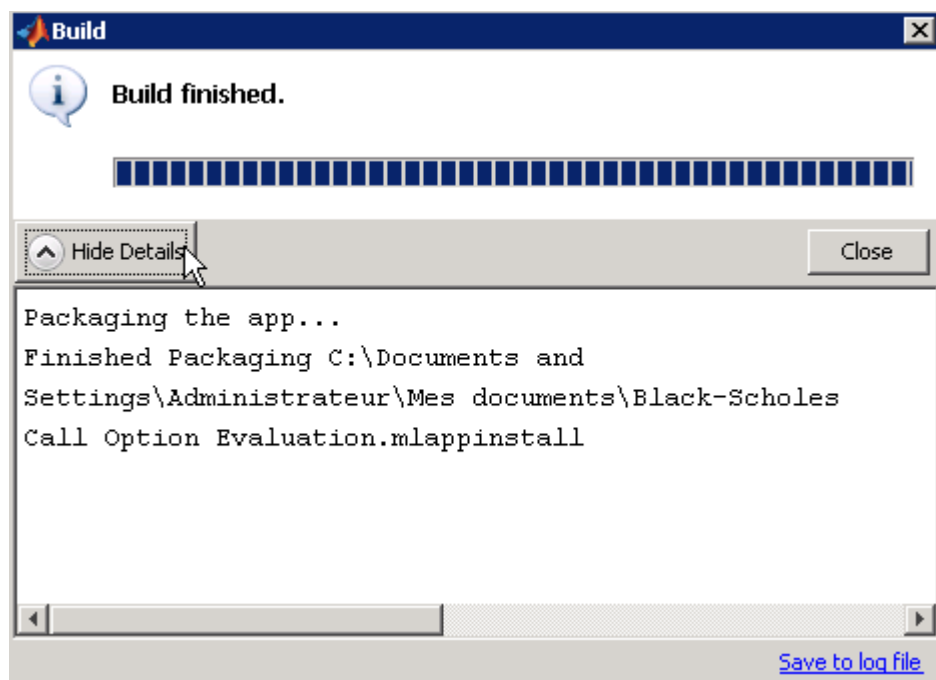
Website: Sciences.ch (Hide contact info)

App Description: Simple BS Call Option Evaluation

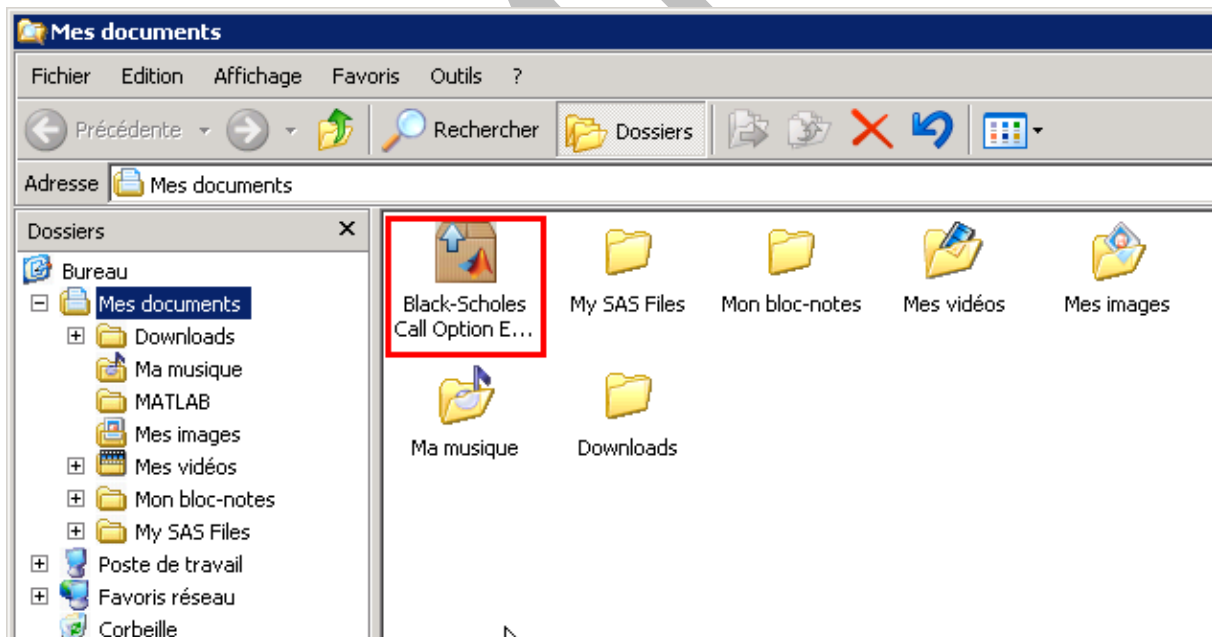
Package: [Highlighted]

Output file: C:\...\Black-Scholes Call Option Evaluation.pr (Change output folder)

The the builder opens:

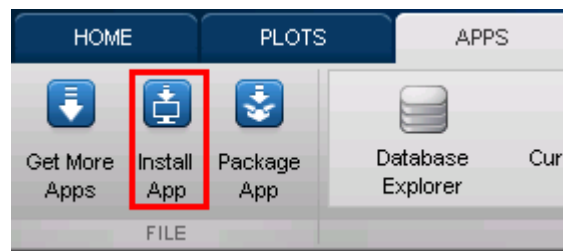


This will create an App in the current workpath folder:

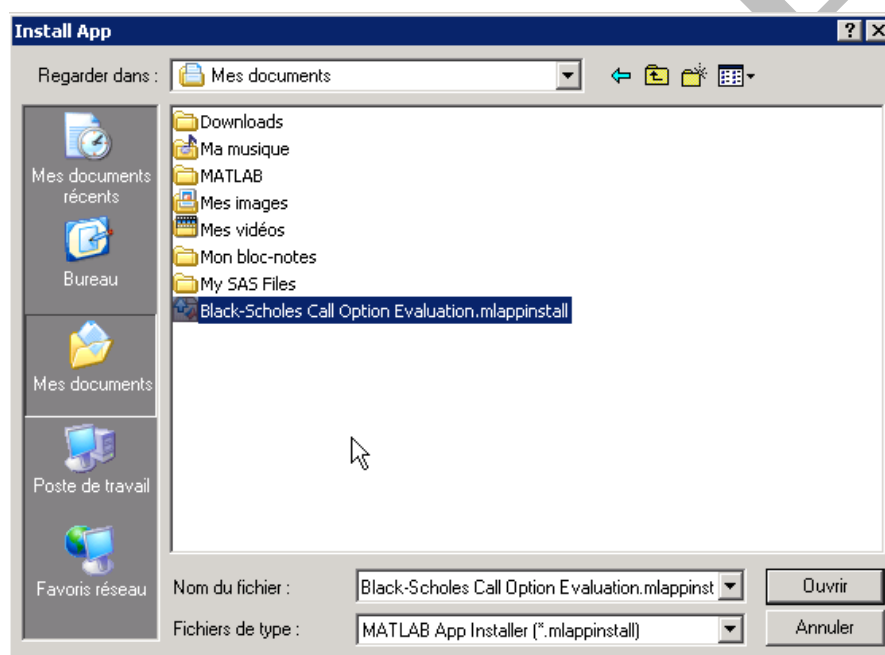


Install/Uninstall a Toolbox

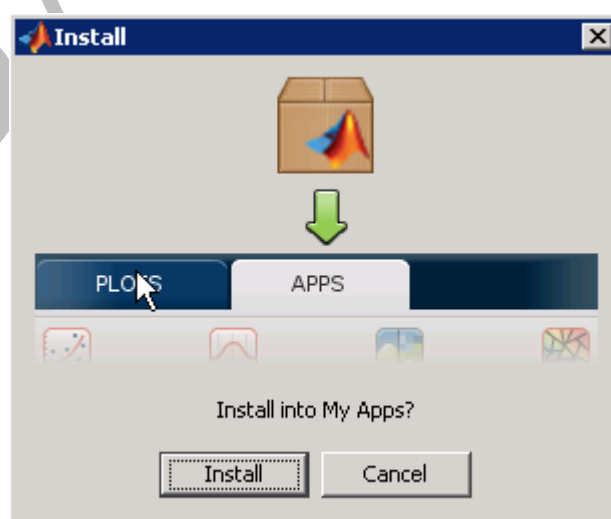
Now once you know how to create an App (Toolbox) you must also know how to install a new App:



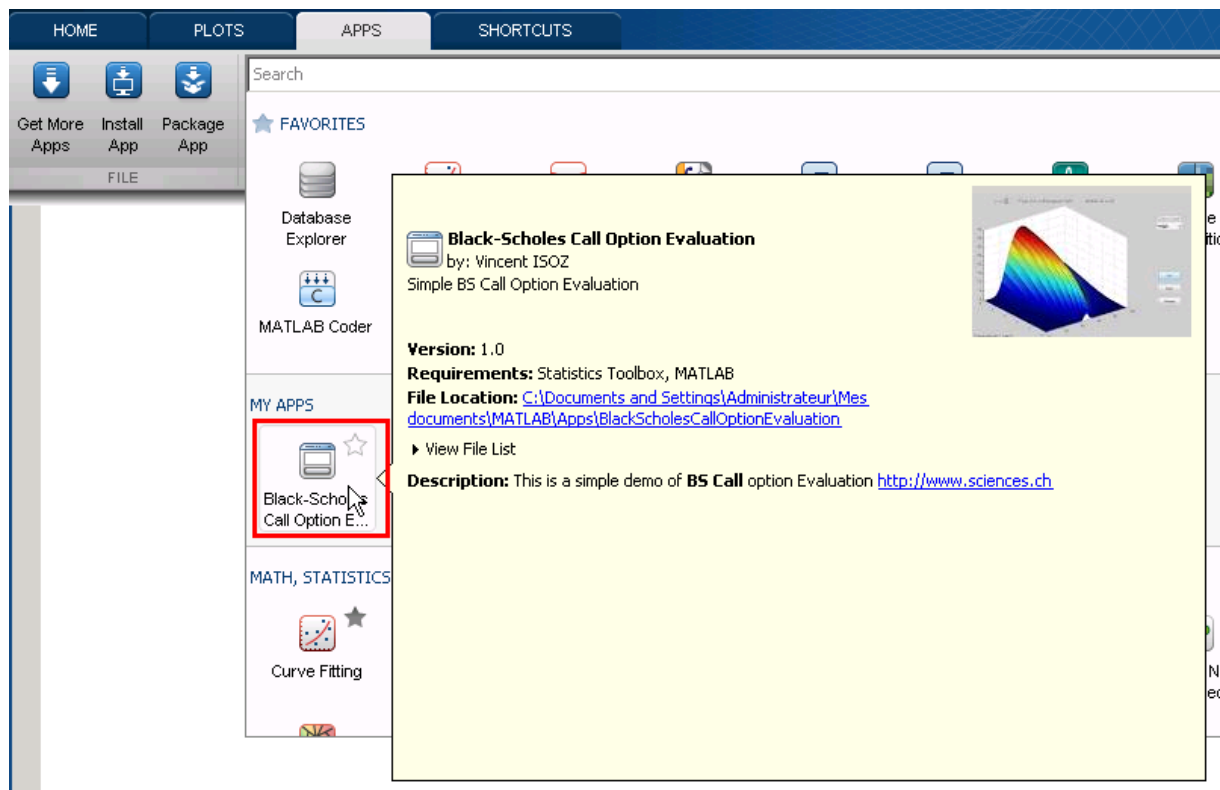
You have to browse where you *.mlappinstall file resides:



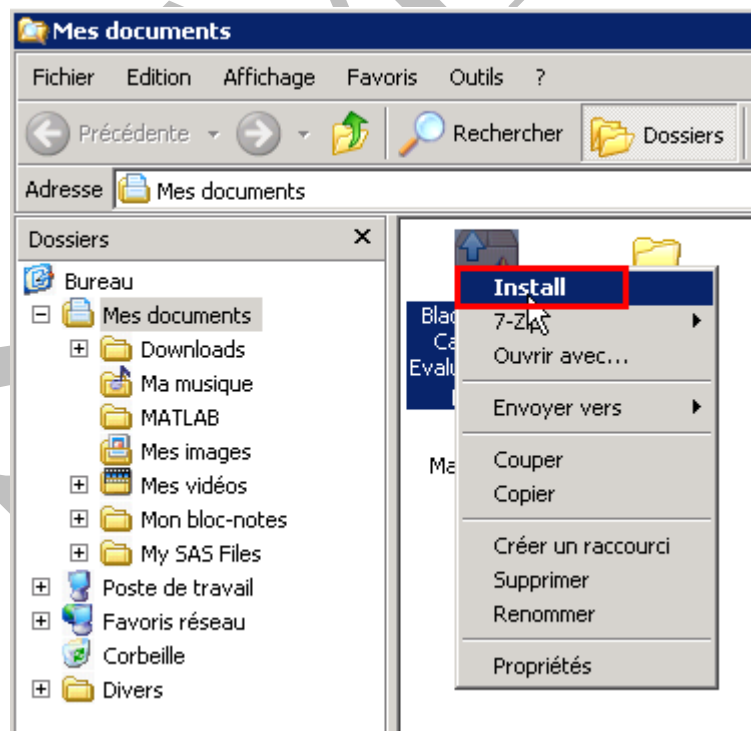
Once you select the app the following screen appears:



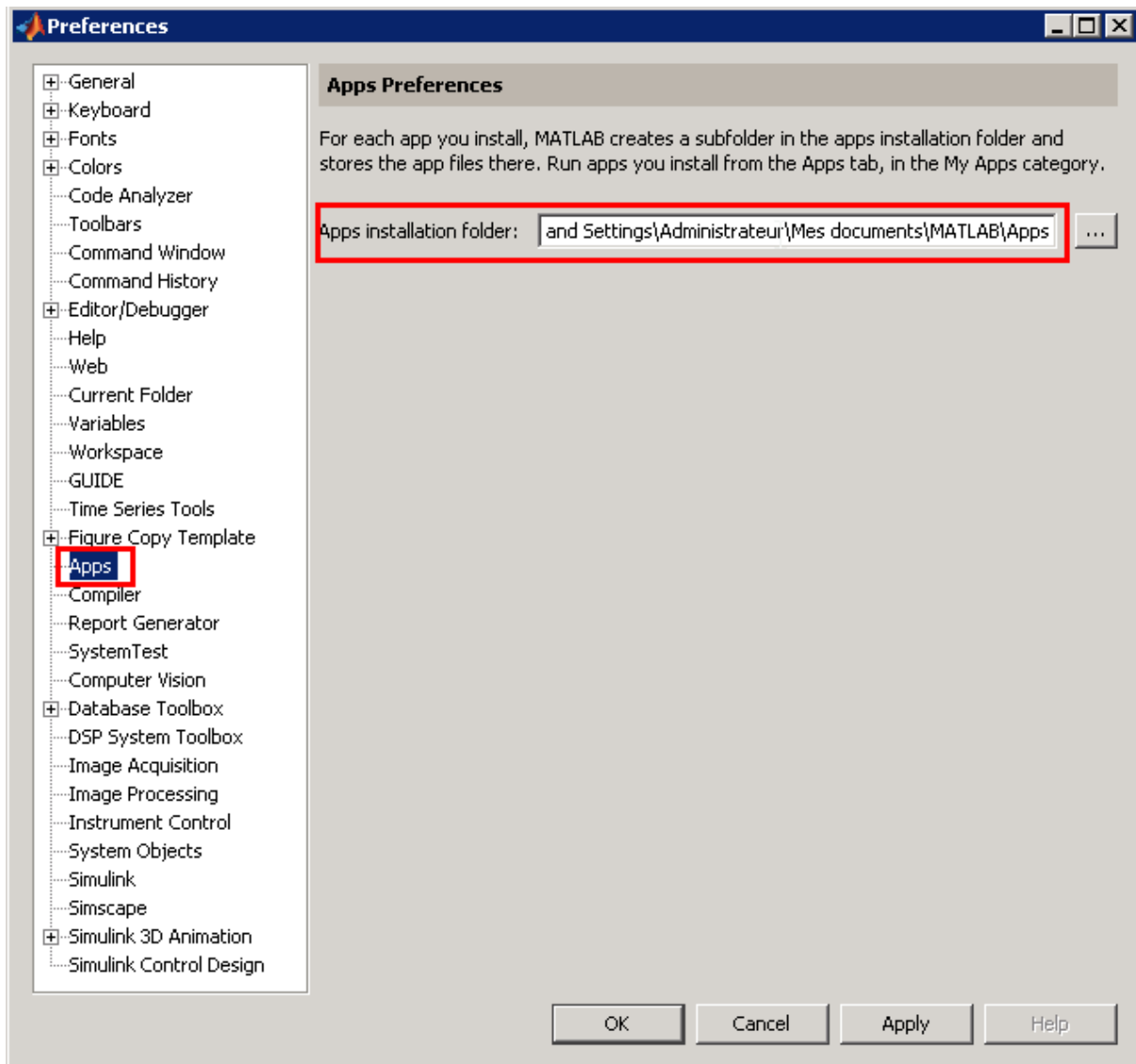
Once the installation finished, you get:



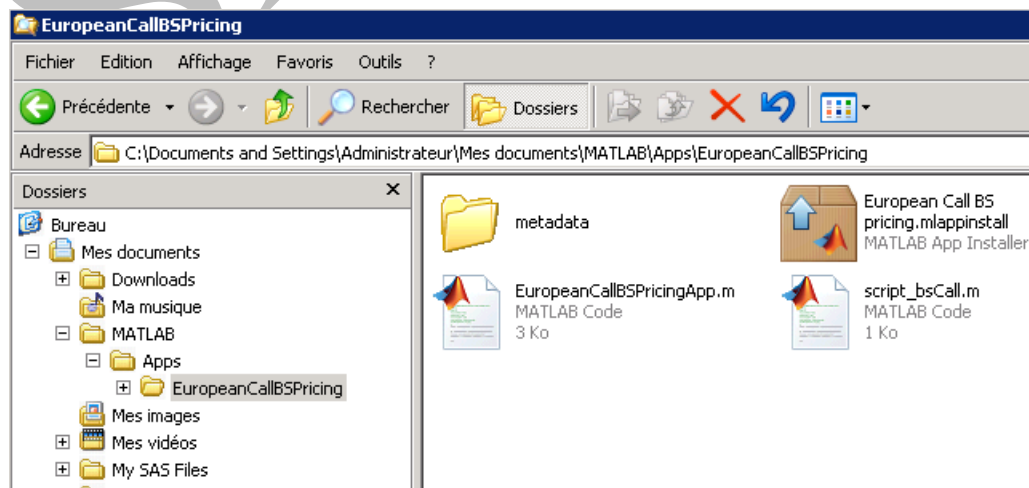
Or an alternative to install is to make a right click on the package and select **Install**:



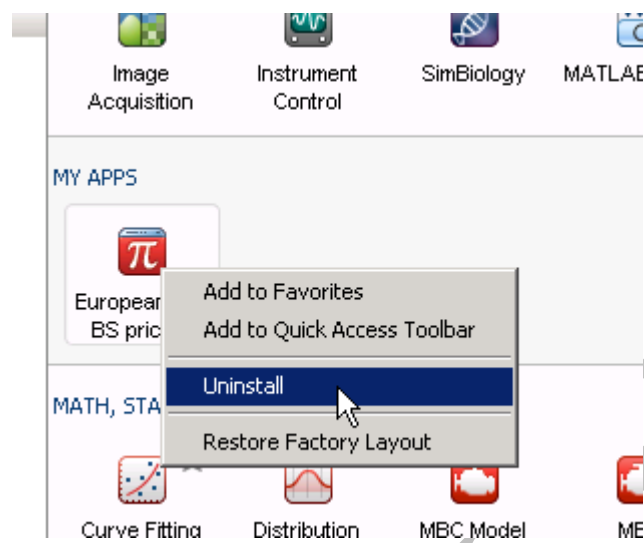
If you want to change the installation folder you just have to go through the **Preferences** of the software:



And this is how the installation folder looks like:

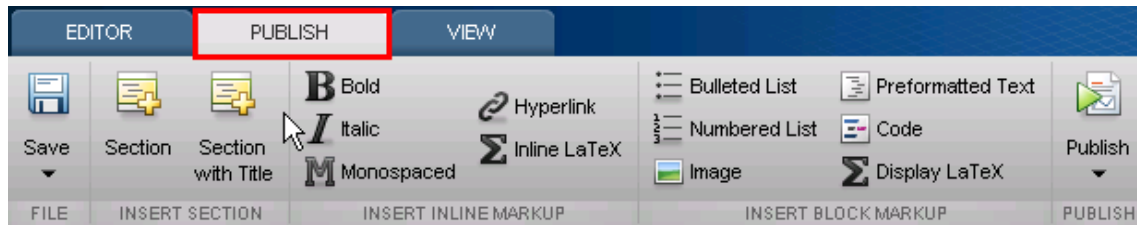


Uninstall a Toolbox is quite easy:

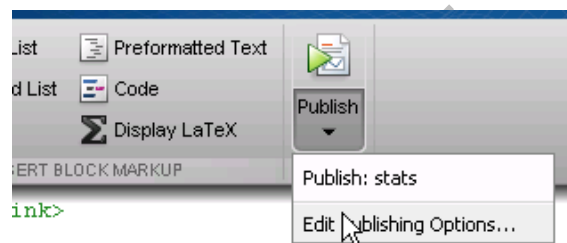


Publish a script (documented script)

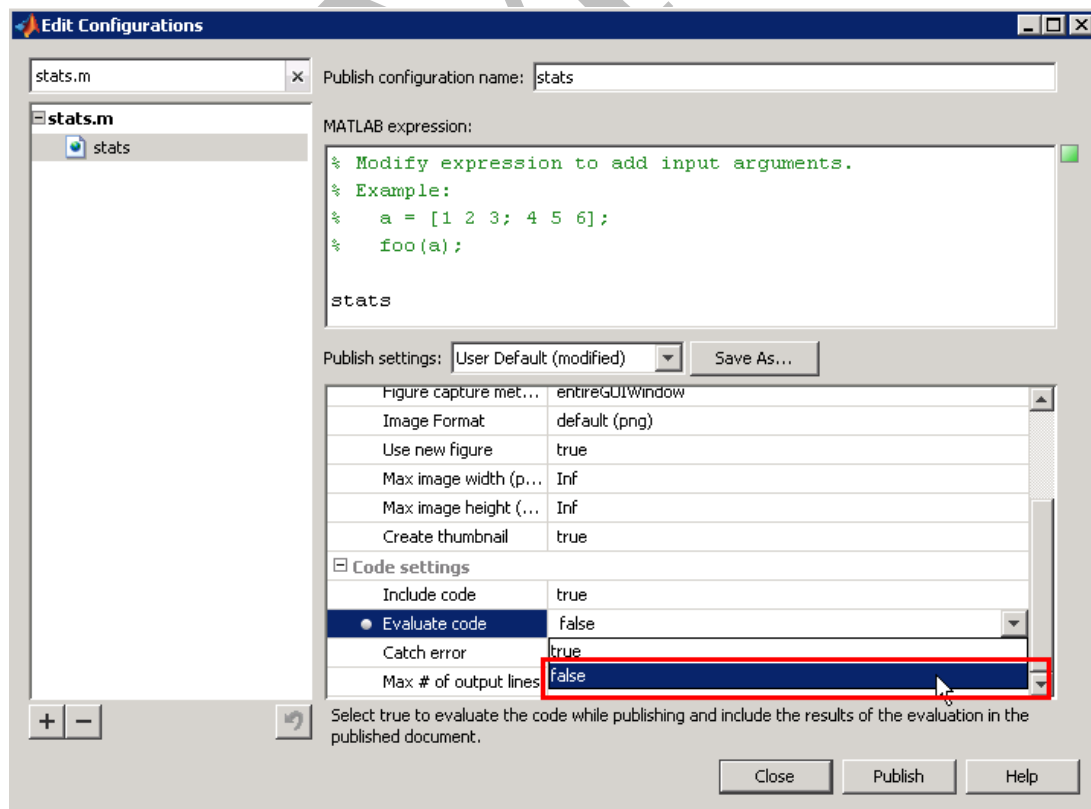
Once you have finished to write a script, an obligation in some industrial fields is to create a documentation of your script. There are some interesting options in the register **PUBLISH** for this purpose:



To publish the file you click on the button Publish and I strongly recommend you to edit some properties first:



One important function is the **Evaluate code** property because if you let this option on **true** (default value) then the whole script will be run each time you publish your code:



Then I strongly recommend first to change this property to false.

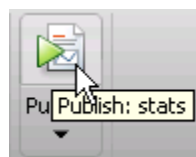
After as example consider the following script (see that there is a link and also a LaTeX equation!):

```

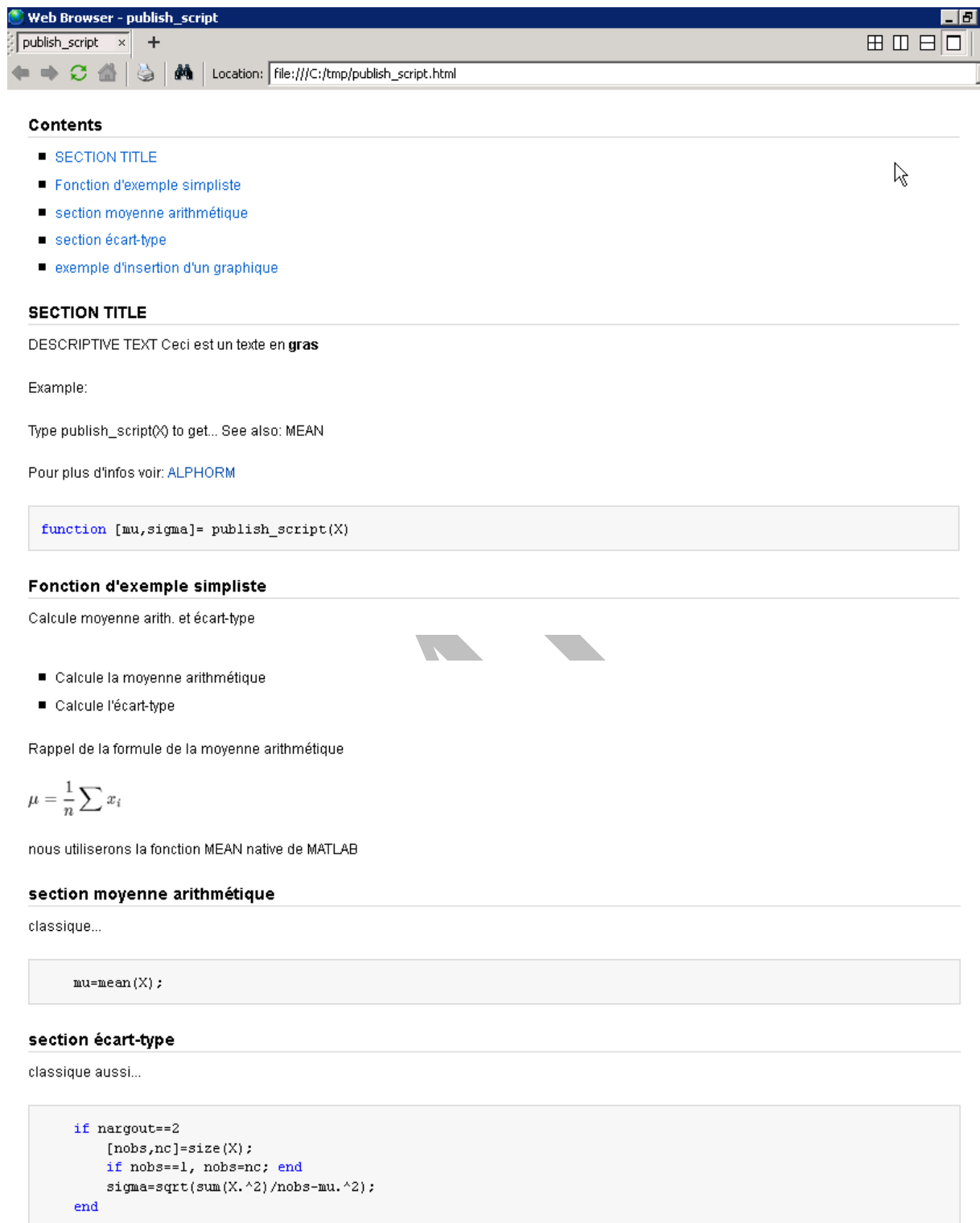
1  %% SECTION TITLE
2  % DESCRIPTIVE TEXT
3  % Ceci est un texte en *gras*
4  %
5  % Example:
6  %
7  % Type publish_script(X) to get...
8  % See also: MEAN
9  %
10 % Pour plus d'infos voir: <http://www.alphorm.com ALPHORM>
11
12 function [mu,sigma]= publish_script(X)
13 %% Fonction d'exemple simpliste
14 % Calcule moyenne arith. et écart-type
15 %%
16 % * Calcule la moyenne arithmétique
17 % * Calcule l'écart-type
18 %
19 % Rappel de la formule de la moyenne arithmétique
20 %
21 %%
22 %  $\mu = \frac{1}{n} \sum x_i$ 
23 %
24 % nous utiliserons la fonction MEAN native de MATLAB
25
26 %% section moyenne arithmétique
27 % classique...
28
29 mu=mean(X);
30
31 %% section écart-type
32 % classique aussi...
33 if nargin==2
34     [nobs,nc]=size(X);
35     if nobs==1, nobs=nc; end
36     sigma=sqrt(sum(X.^2)/nobs-mu.^2);
37 end
38 %% exemple d'insertion d'un graphique
39 % <<peaks.jpg>>
end

```

Now if we **Publish** the code:



The result will look like following:



Contents

- [SECTION TITLE](#)
- [Fonction d'exemple simpliste](#)
- [section moyenne arithmétique](#)
- [section écart-type](#)
- [exemple d'insertion d'un graphique](#)

SECTION TITLE

DESCRIPTIVE TEXT Ceci est un texte en **gras**

Example:

Type `publish_script()` to get... See also: [MEAN](#)

Pour plus d'infos voir: [ALPHORM](#)

```
function [mu,sigma]= publish_script(X)
```

Fonction d'exemple simpliste

Calcule moyenne arith. et écart-type

- Calcule la moyenne arithmétique
- Calcule l'écart-type

Rappel de la formule de la moyenne arithmétique

$$\mu = \frac{1}{n} \sum x_i$$

nous utiliserons la fonction MEAN native de MATLAB

section moyenne arithmétique

classique...

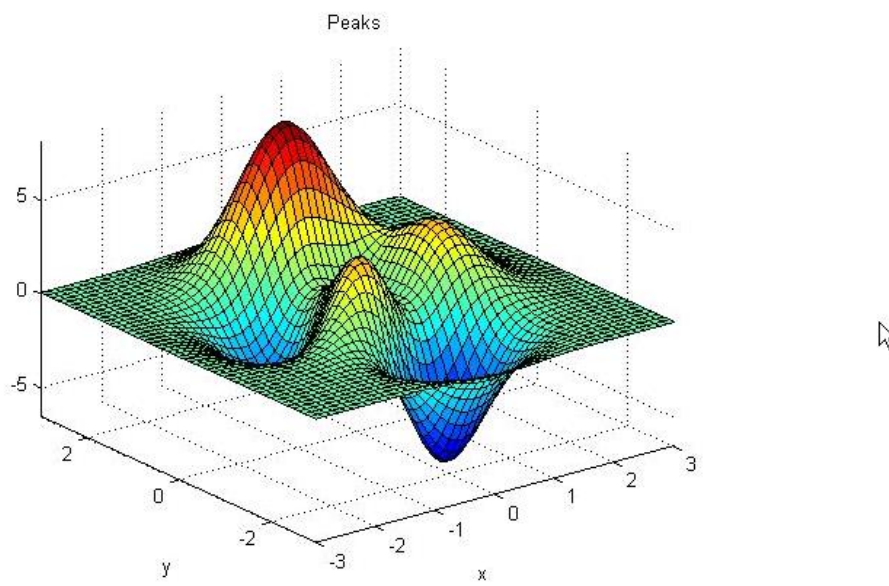
```
mu=mean(X);
```

section écart-type

classique aussi...

```
if nargin==2
    [nobs,nc]=size(X);
    if nobs==1, nobs=nc; end
    sigma=sqrt(sum(X.^2)/nobs-mu.^2);
end
```

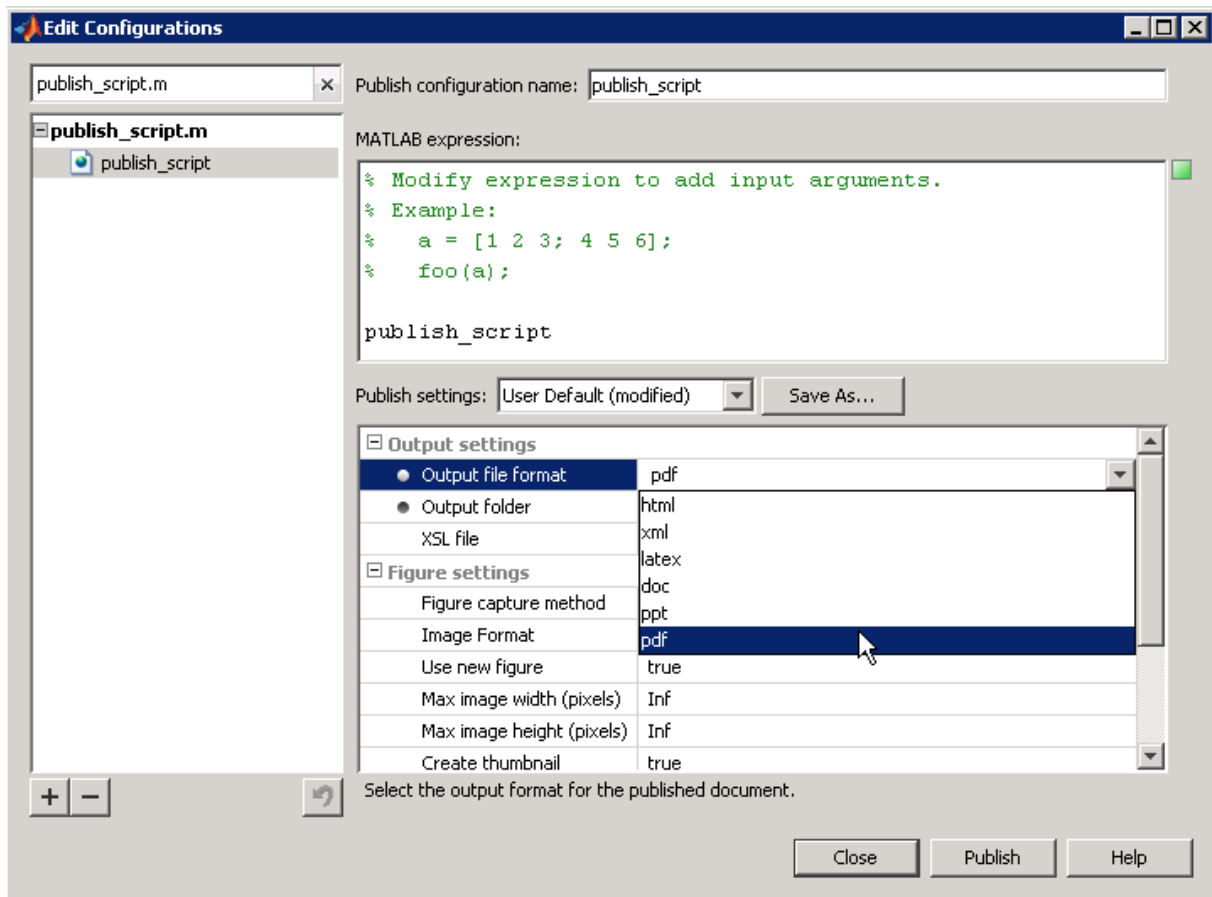

exemple d'insertion d'un graphique



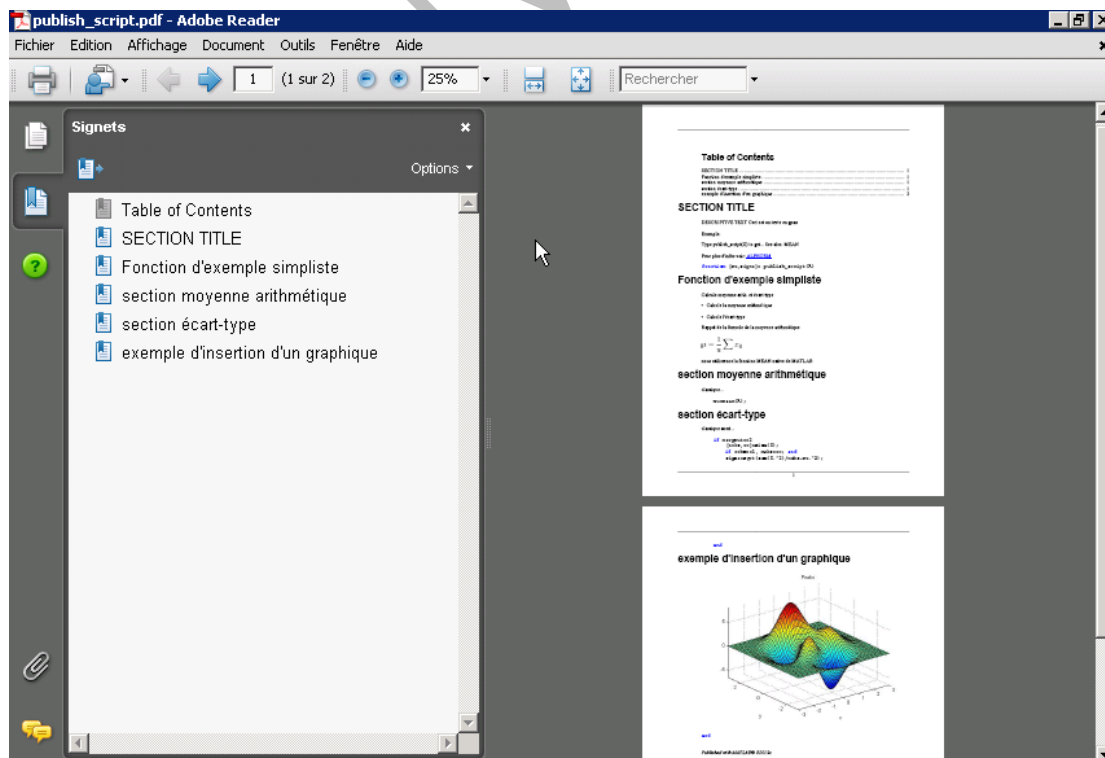
```
end
```

Published with MATLAB® R2013a

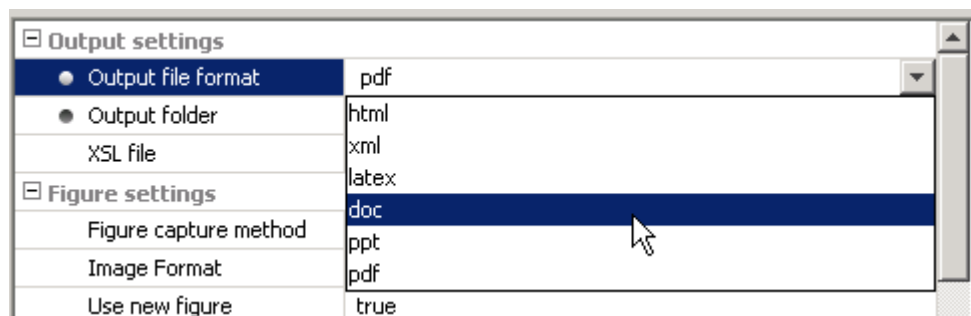
If we change the output option to a PDF:



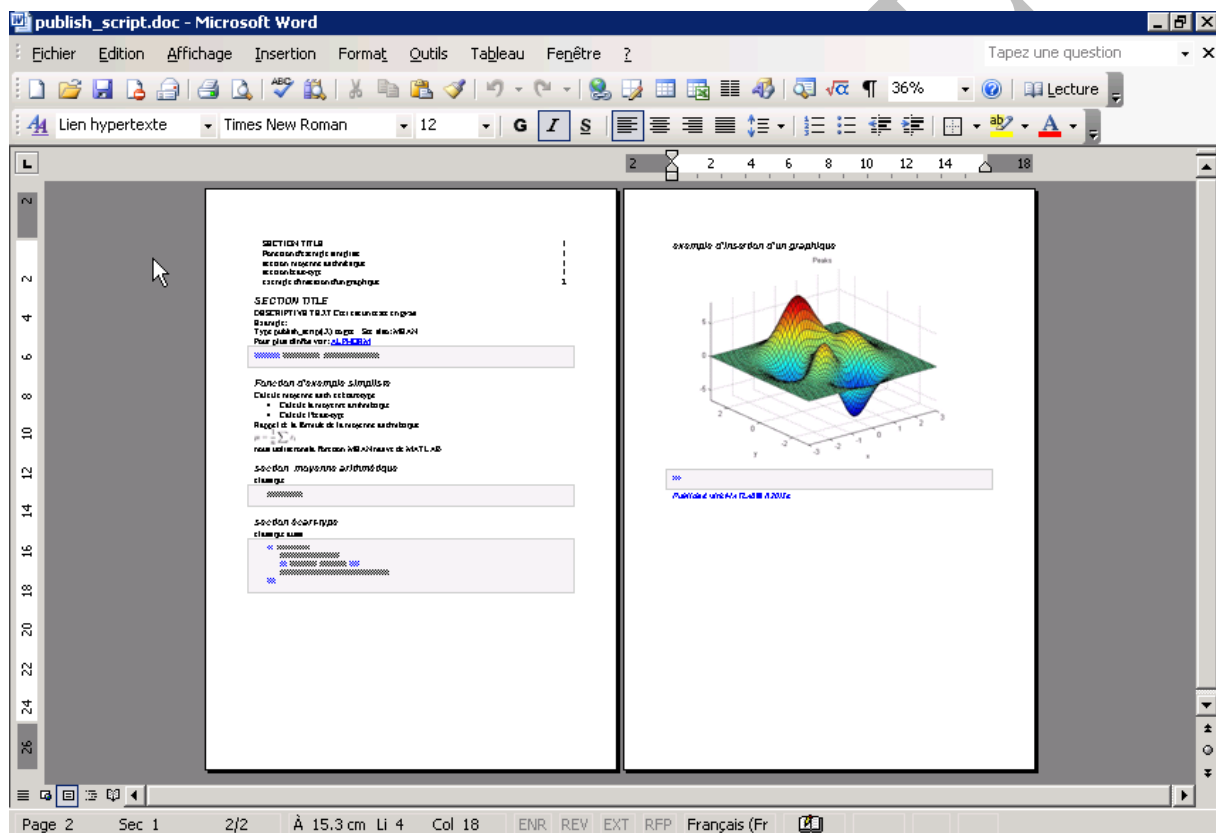
We get we get a very clean result with a table of content and clickable links:



And if we change the output to *.doc:



we also get a very clean output with a real table of contents, styles, clickable links, etc.:



Caution!!! If you choose to run the code in the **Publish** feature property, you will see only the charts at the end of the publication that were still open after the execution finished (you won't see closed charts!).

Optimization tricks

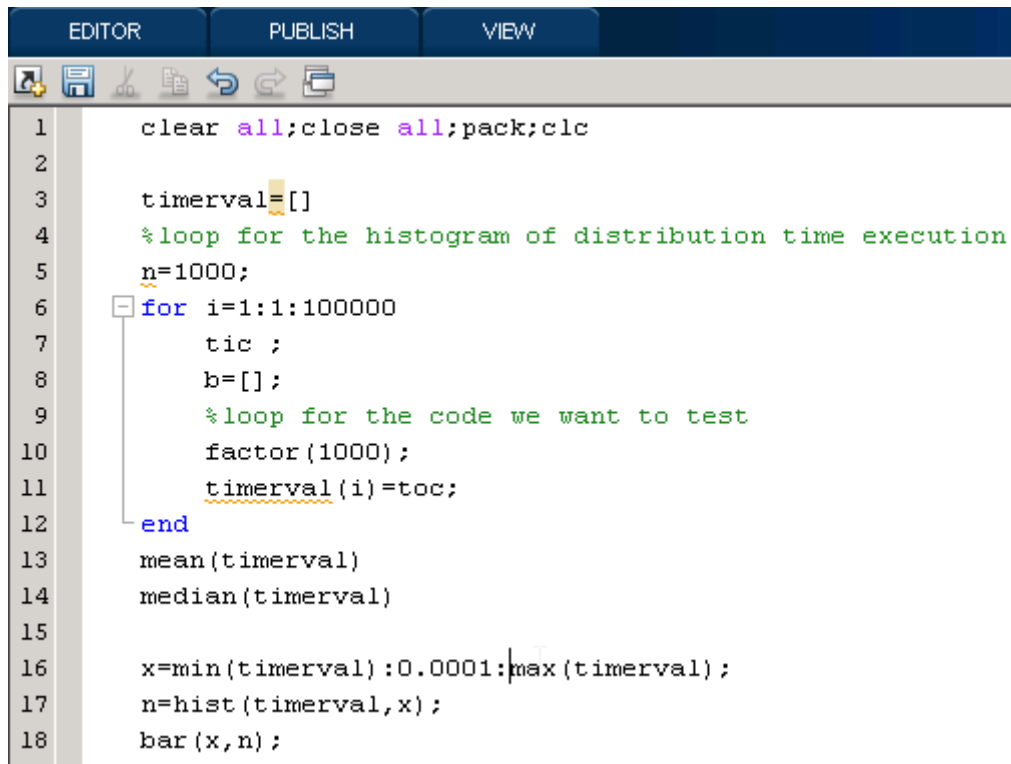
In advanced finance (hyperfrequency trading) and spatial applications, the performance of a script a major factor (with the robustness...).

The purpose of this small chapter is to show you some simple optimization tricks.

Statistically check code efficiency

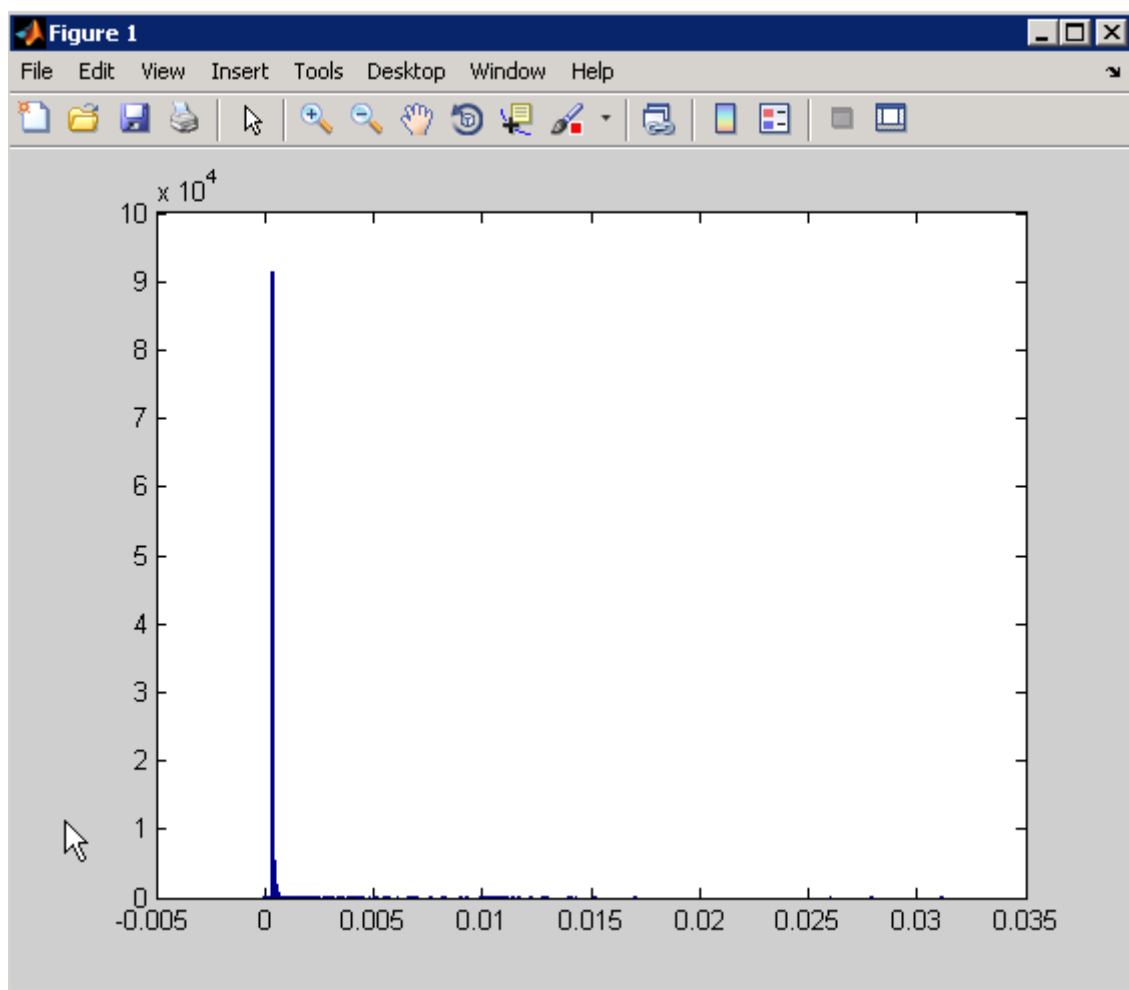
When you test the efficiency of a script you must not run it just one time, but thousands of times to get a statistical distribution of running time.

Here is such an example with the famous example (in cryptography) of the **factor()** function:



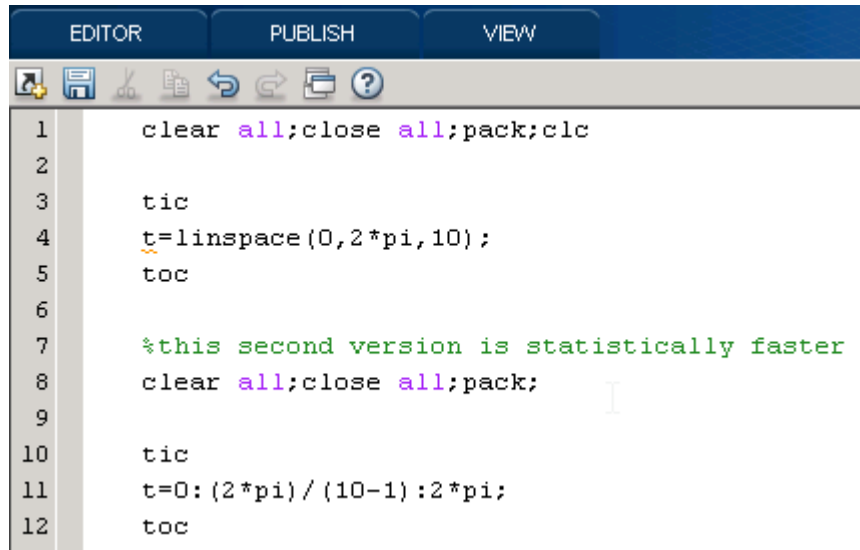
```
1      clear all;close all;pack;clc
2
3      timerval=[]
4      %loop for the histogram of distribution time execution
5      n=1000;
6      for i=1:1:100000
7          tic ;
8          b=[];
9          %loop for the code we want to test
10         factor(1000);
11         timerval(i)=toc;
12     end
13     mean(timerval)
14     median(timerval)
15
16     x=min(timerval):0.0001:max(timerval);
17     n=hist(timerval,x);
18     bar(x,n);
```

The result will look like following:



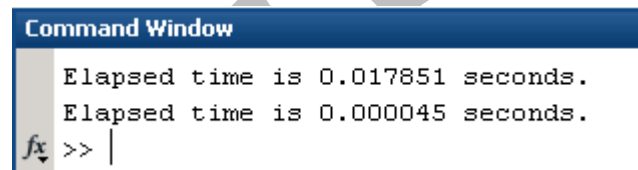
linspace vs sequence

A lot of people use the **linspace()** function to create sequence of values but in the facts this **function** is a little bit slow as you can see below:



```
1 clear all;close all;pack;clc
2
3 tic
4 t=linspace(0,2*pi,10);
5 toc
6
7 %this second version is statistically faster
8 clear all;close all;pack;
9
10 tic
11 t=0:(2*pi)/(10-1):2*pi;
12 toc
```

The running time differences gives:



```
Command Window
Elapsed time is 0.017851 seconds.
Elapsed time is 0.000045 seconds.
fx >> |
```

Simple arithmetic operation vs bsxfun

Another difficulty trick to know is to use the broadcasting **bsxfun()** instead of simple arithmetic operation (and some others) in MATLAB.

To understand the subject first see and read the following example:

```

Command Window
>> a=rand(3,3)

a =

    0.7011    0.5478    0.4418
    0.4847    0.8017    0.0320
    0.1305    0.6626    0.4883

>> b=1:3;b=b';repmat(b,1,3)

ans =

     1     1     1
     2     2     2
     3     3     3

>> repmat(b,1,3)+a %but this is slow

ans =

    1.7011    1.5478    1.4418
    2.4847    2.8017    2.0320
    3.1305    3.6626    3.4883

fx >> |

```

But this is much faster:

```

Command Window
>> bsxfun(@plus,a,b) %@plus = "+"

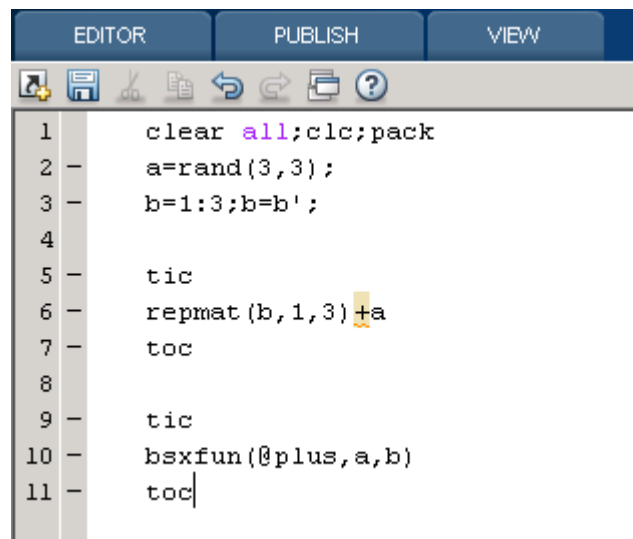
ans =

    1.7011    1.5478    1.4418
    2.4847    2.8017    2.0320
    3.1305    3.6626    3.4883

fx >> |

```

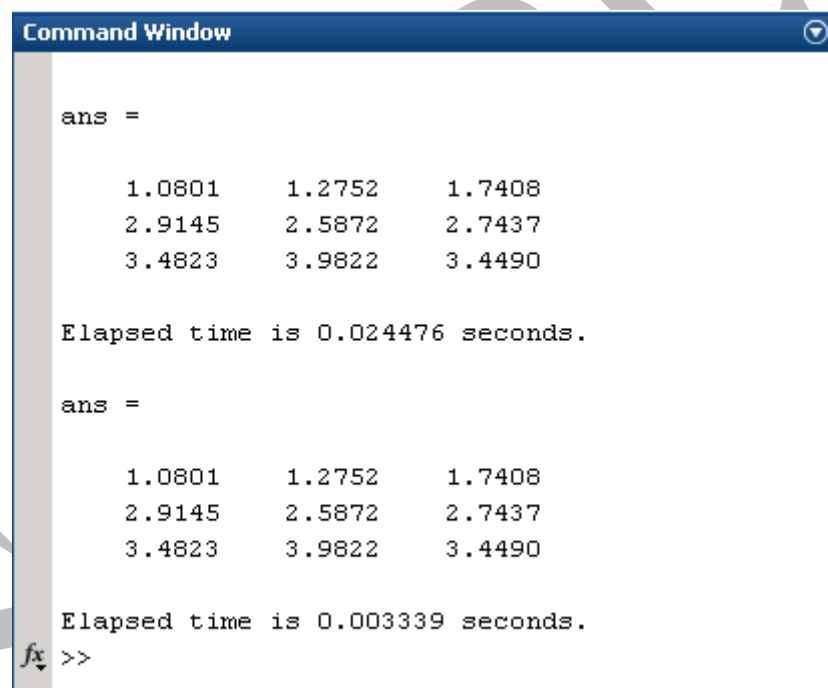
Let us now compare the performance between a classic addition and a sum using **bsxfun**:



```

1      clear all;clc;pack
2      a=rand(3,3);
3      b=1:3;b=b';
4
5      tic
6      repmat(b,1,3)+a
7      toc
8
9      tic
10     bsxfun(@plus,a,b)
11     toc
  
```

The result will be:



```

Command Window

ans =

    1.0801    1.2752    1.7408
    2.9145    2.5872    2.7437
    3.4823    3.9822    3.4490

Elapsed time is 0.024476 seconds.

ans =

    1.0801    1.2752    1.7408
    2.9145    2.5872    2.7437
    3.4823    3.9822    3.4490

Elapsed time is 0.003339 seconds.
fx >>
  
```

The most used operators of **bsxfun** are **@plus**, **@minus**, **@times**, **@power**, **@max**, **@min**.

Create userforms (GUI)

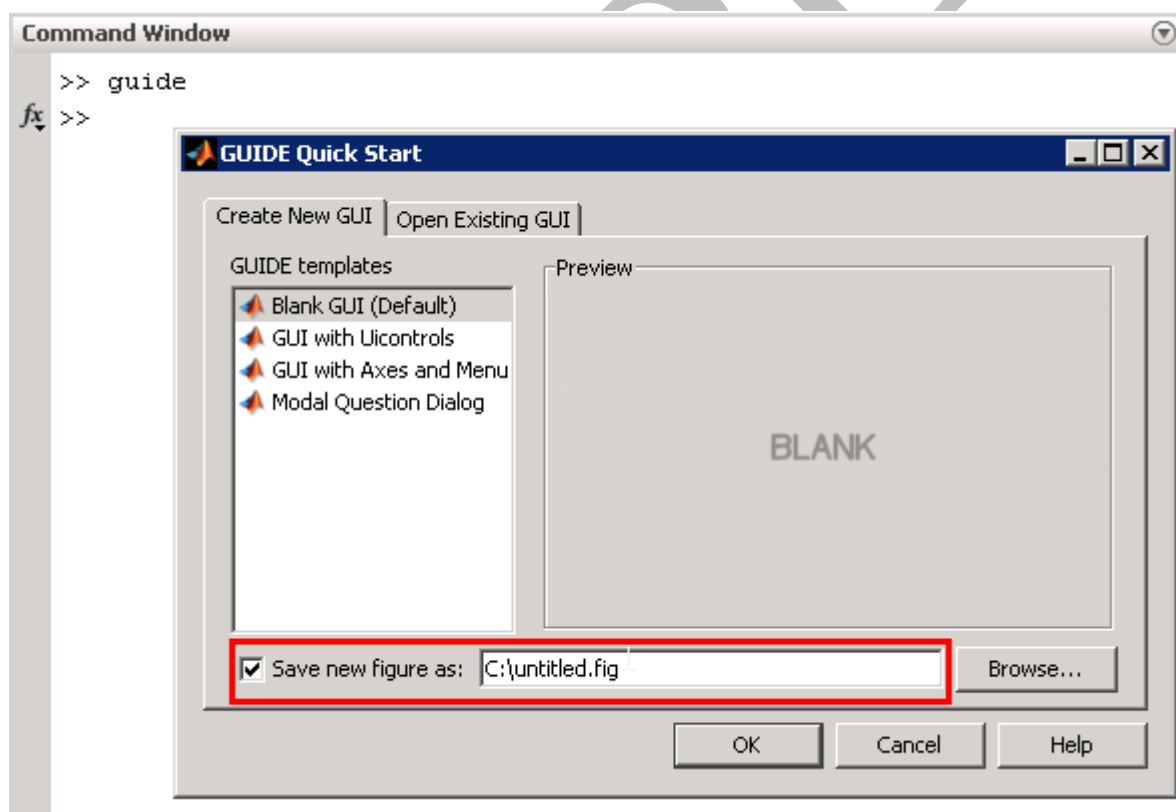
Like coding (doesn't matter which language) non-mathematical stuff is very boring in my point of view, create GUI is also a much more boring job that could better be given to a student or a designer rather than a senior scientist having a PhD or a Master or even an undergraduate level.

GUI have a lot of possibilities like in almost all scripting/programming languages this is why this section/chapter about GUI will be written very slowly and perhaps never completed. Hope you understand what I mean...

Create userforms using GUIDE

Create GUI in MATLAB is a huge subject! There a 300 hundred pages book only about this tool! The purpose of this PDF will not be to make you a professional of GUI creation but just to give you an idea of how it works.

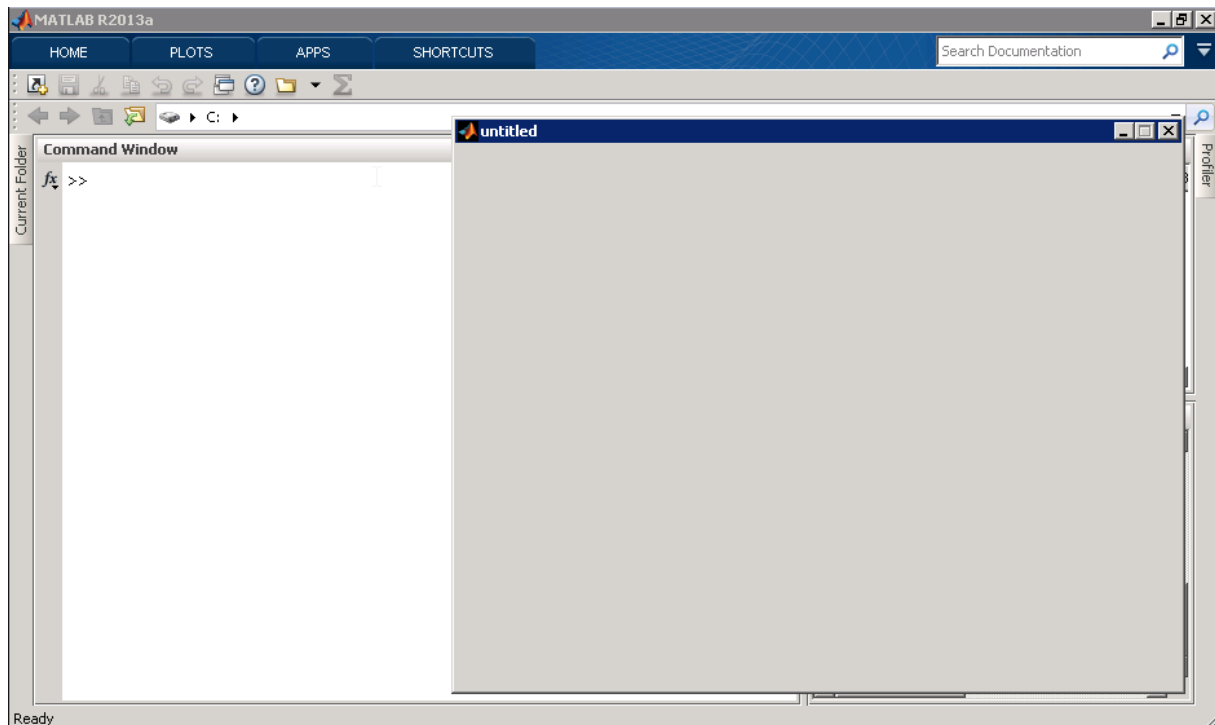
First thing to know is how to start the guide builder by typing **guide** in the *Command Window*:



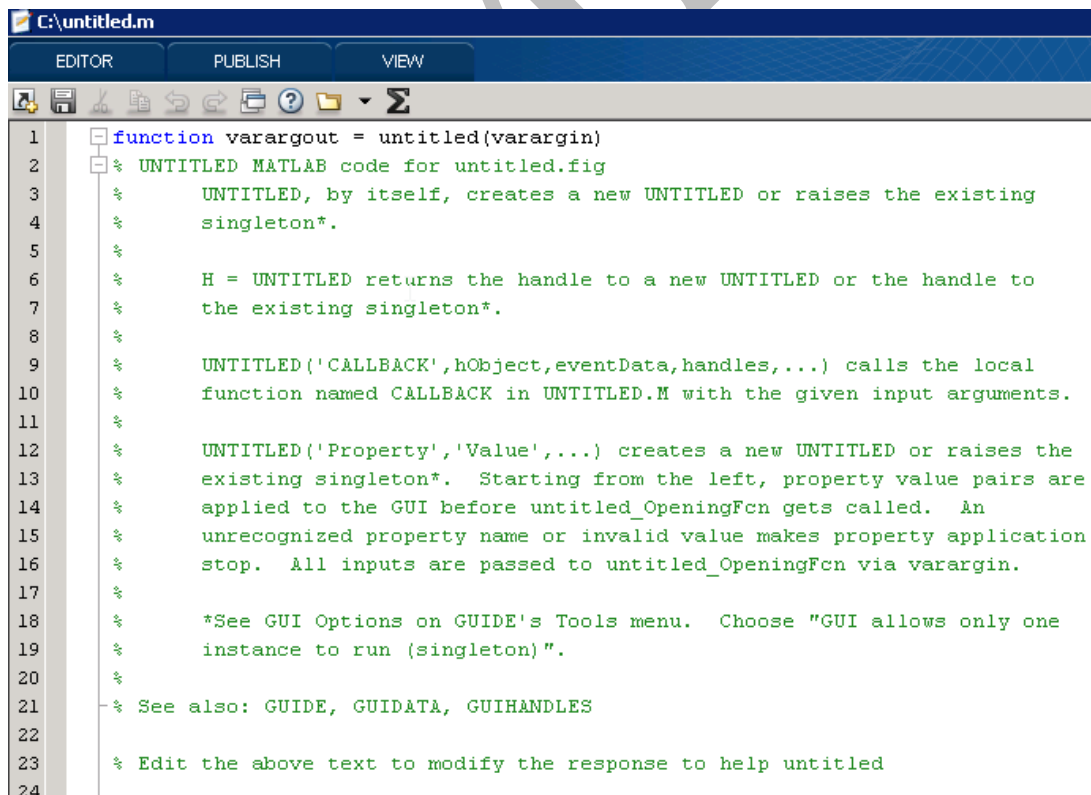
When you save you get two files:



When MATLAB is closed and you double click on the first (*.fig) you simply get MATLAB that open with the GUI:



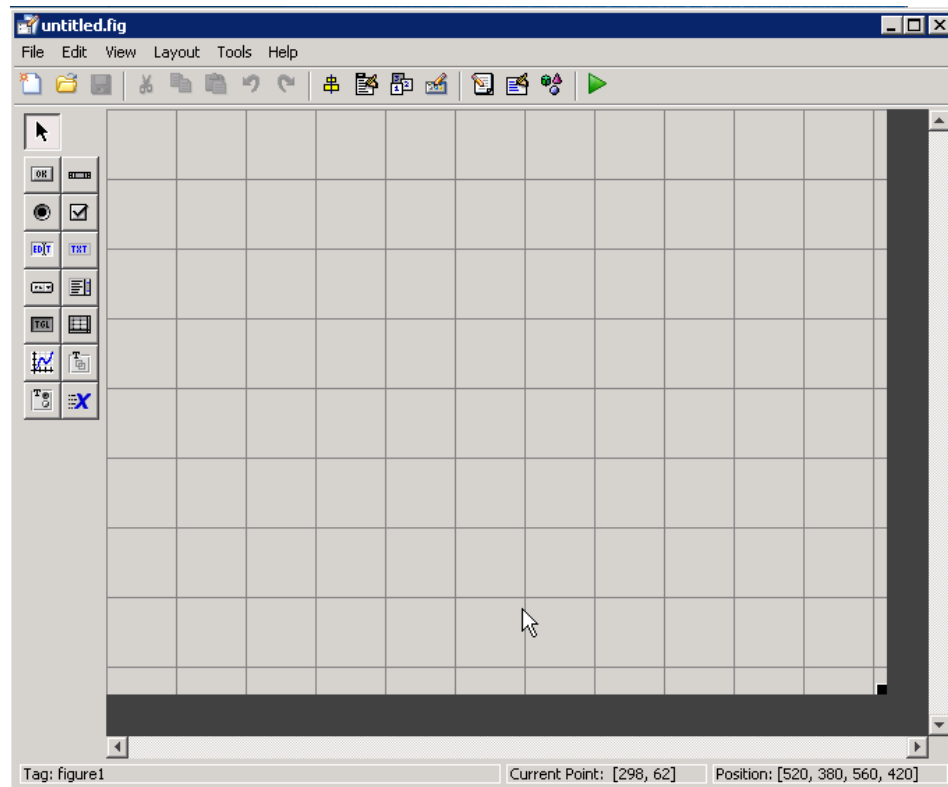
If you double click on the *.m file you get (this is only a part of the screenshot):



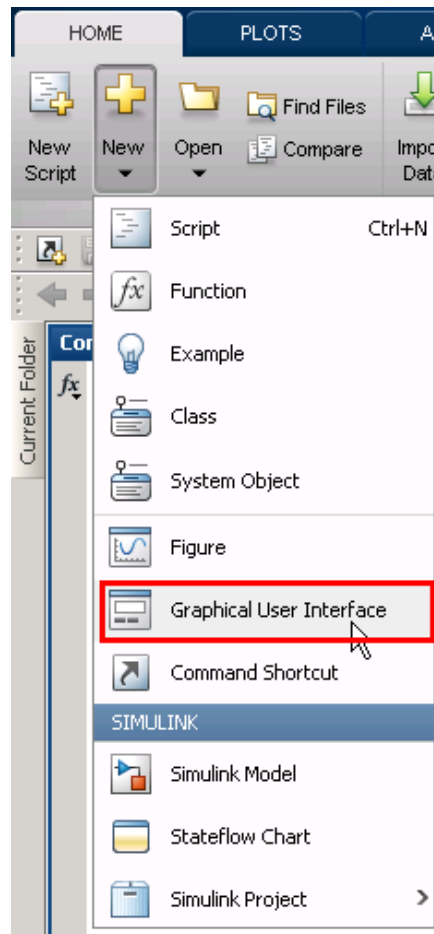
You can't run this file! If you remove the *.m file the GUI stills work because it is empty! But in reality never delete the *.m file associated to the figure because it will be used for callbacks!

Also don't forget to make **Error Management!!!**

And we specify the figure file name in the field **Save figure as** and once validated, we get:

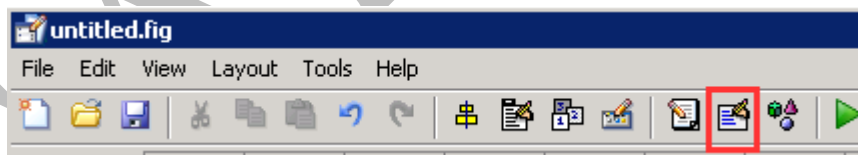


Another way to create a new GUI is to use the mouse:

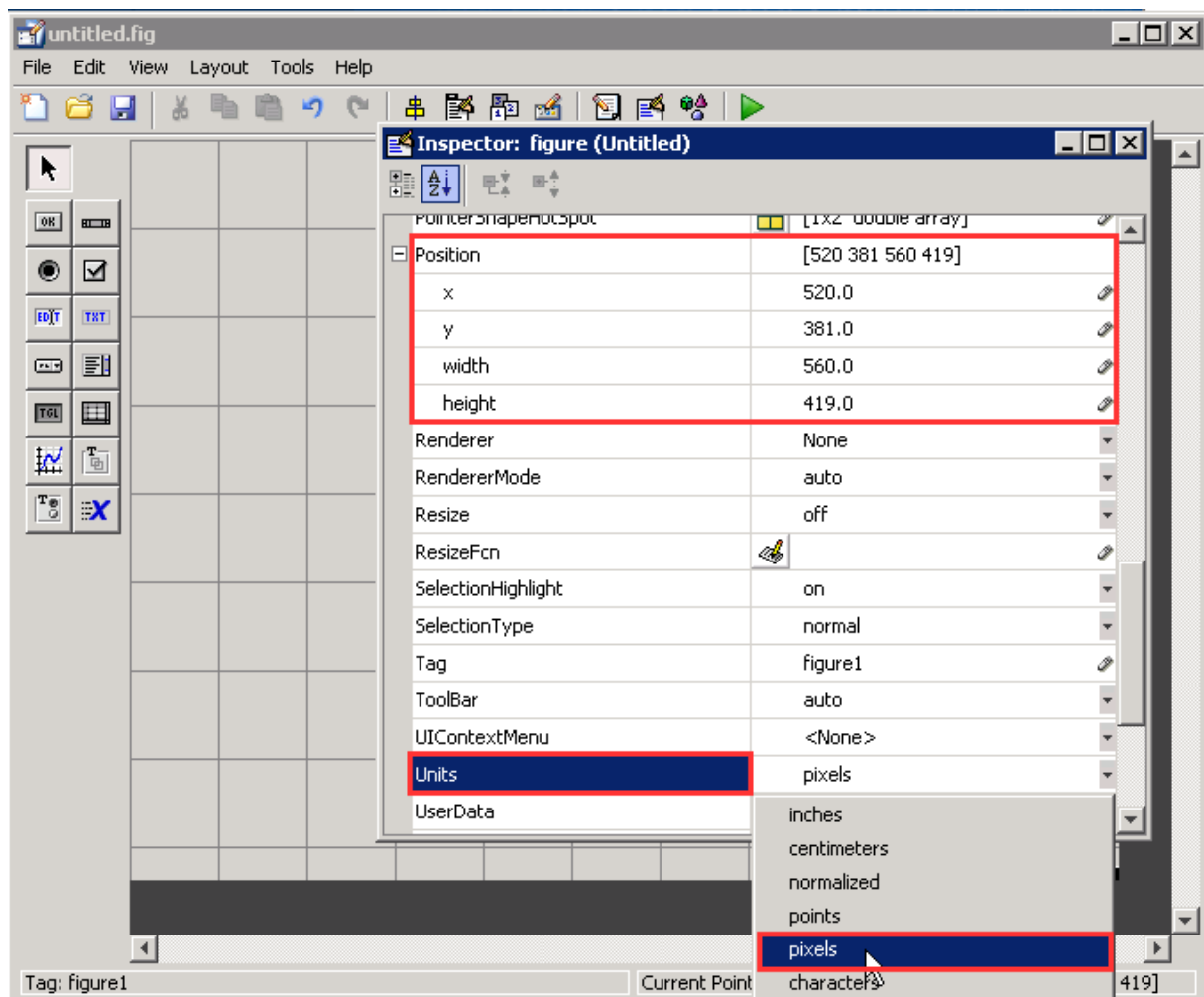


You can resize the userform by clicking on the lower-right corner and resize it.

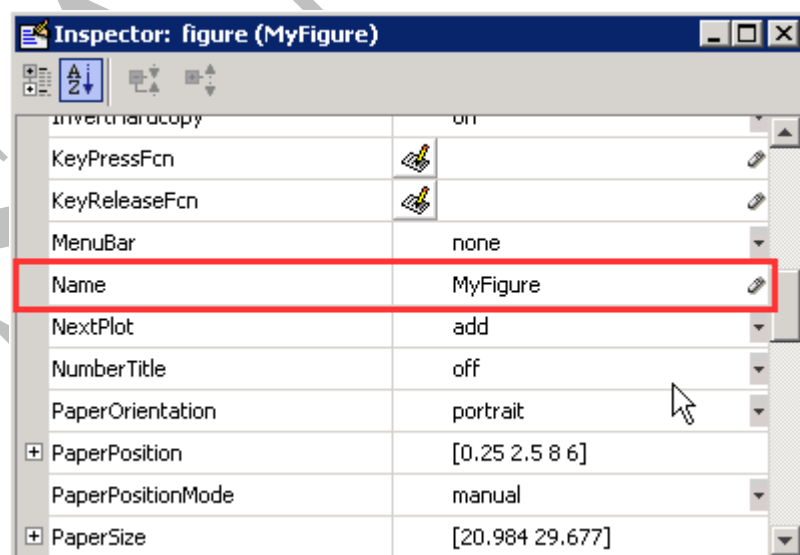
Now to change the size and initial position with precision of a userform you can click on the userform **Inspector** button:



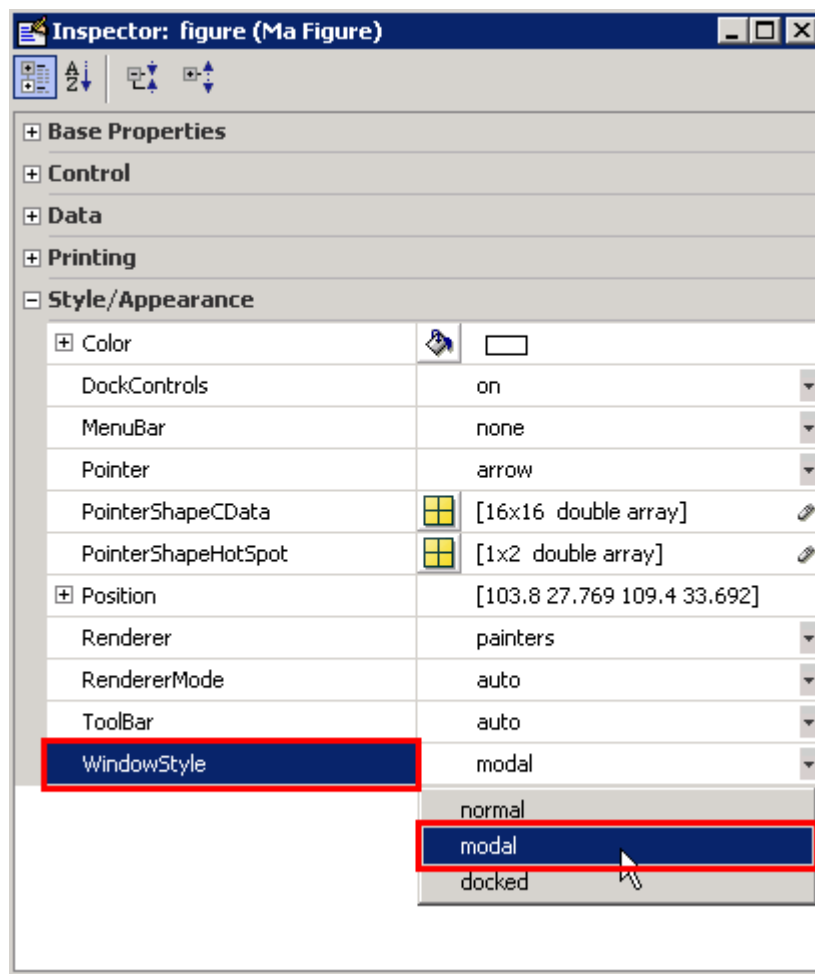
Then by scrolling down in the small window that will appear you will have the possibility to choose the measurement **Units** and also the **Position** properties:



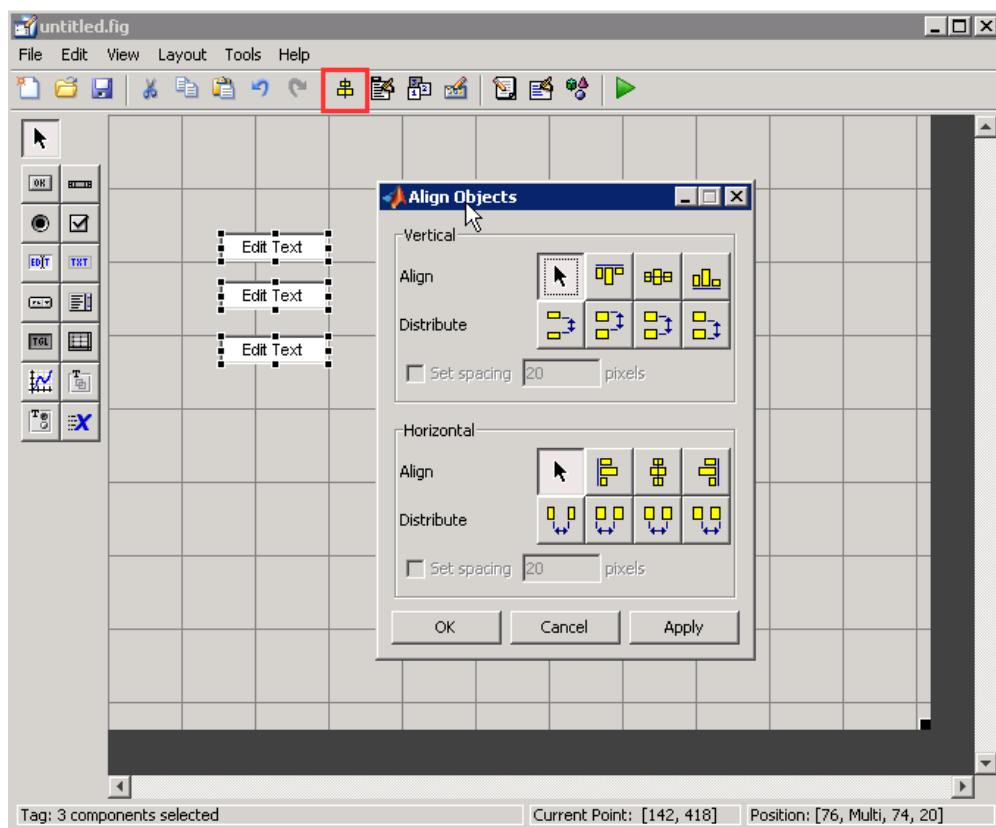
To rename the GUI:



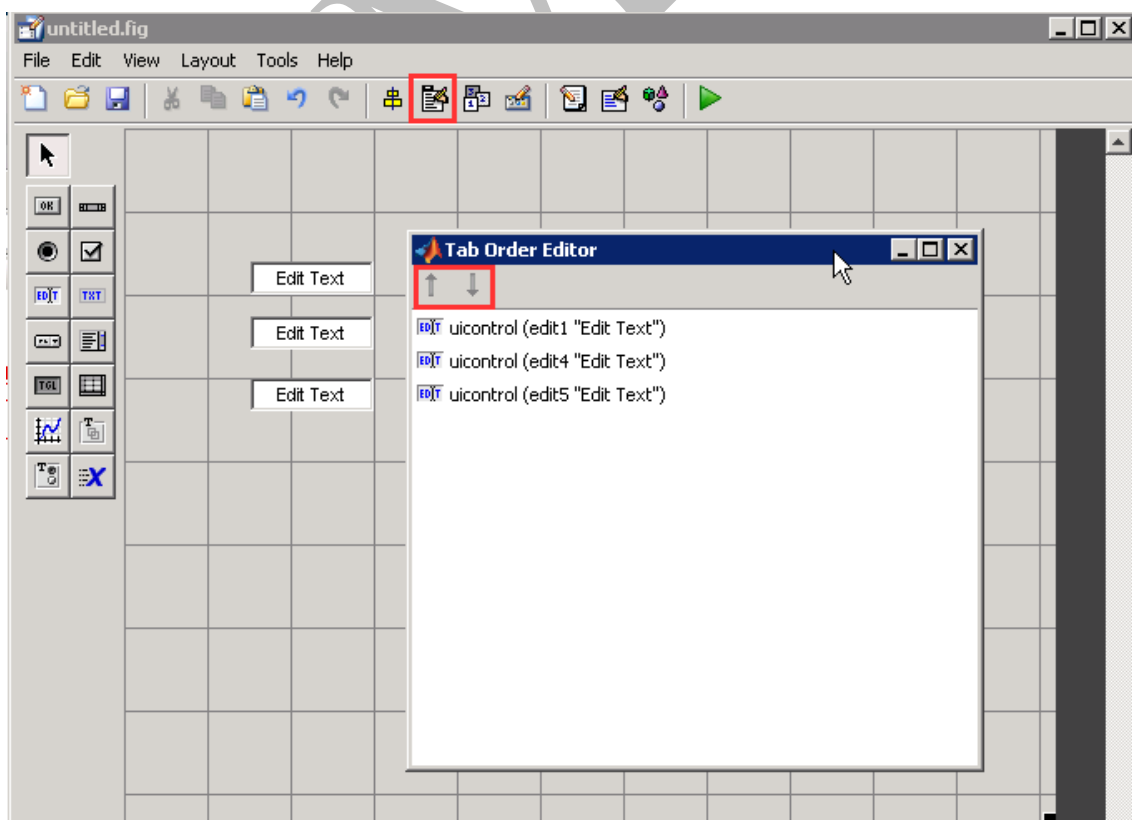
To make the GUI modal (so that the user can not activate another window behind):



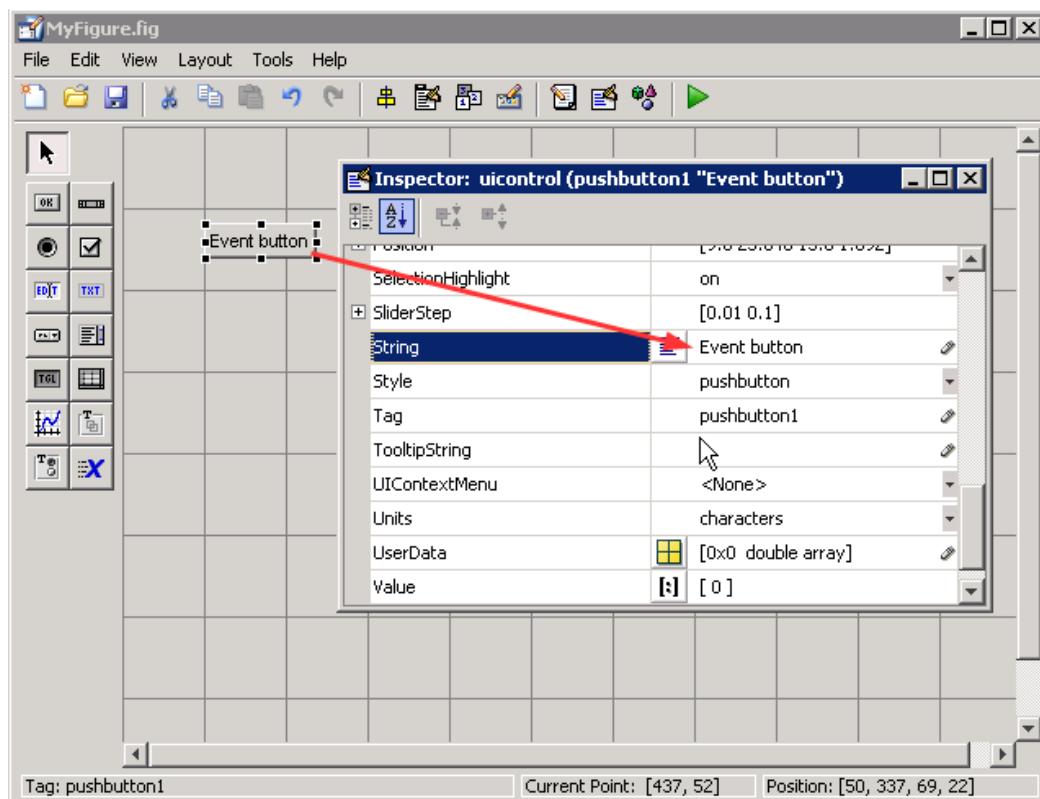
To align objects:



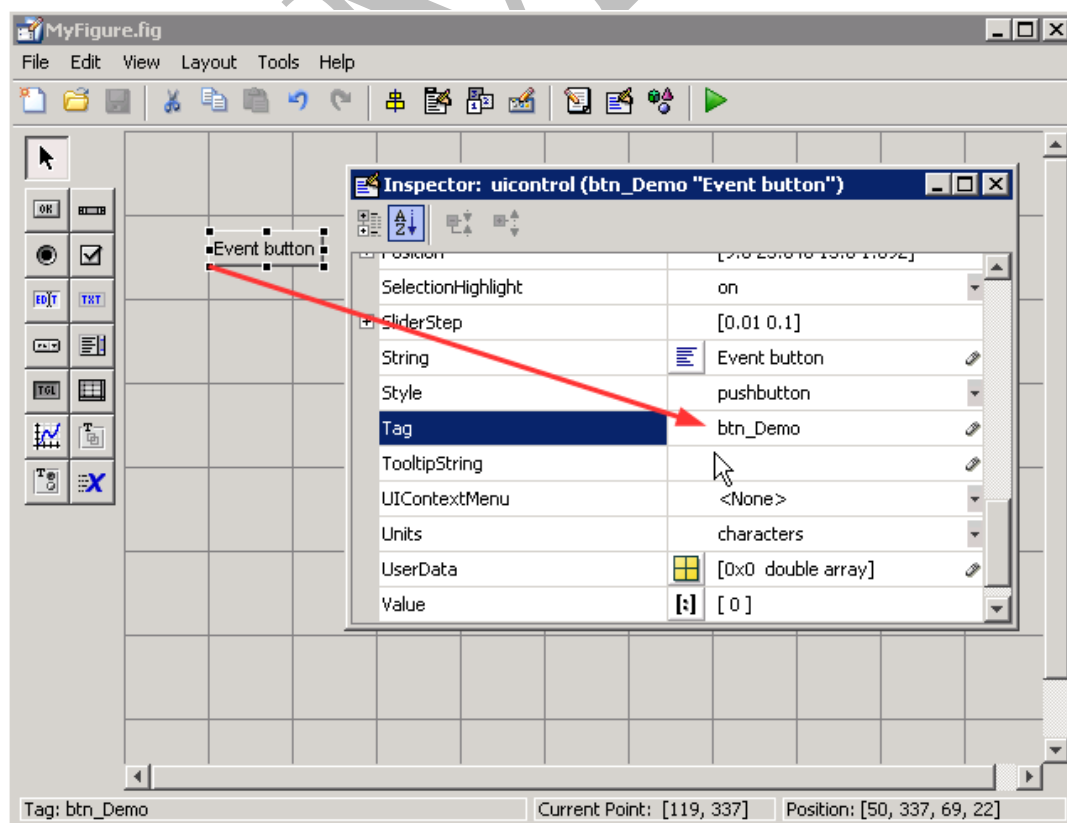
To choose tab order:



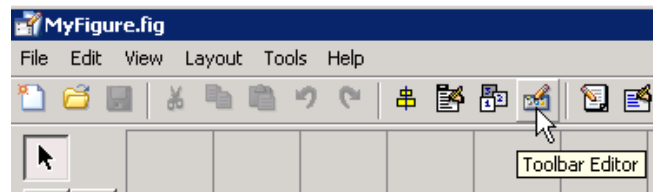
Change the legend name of a button:



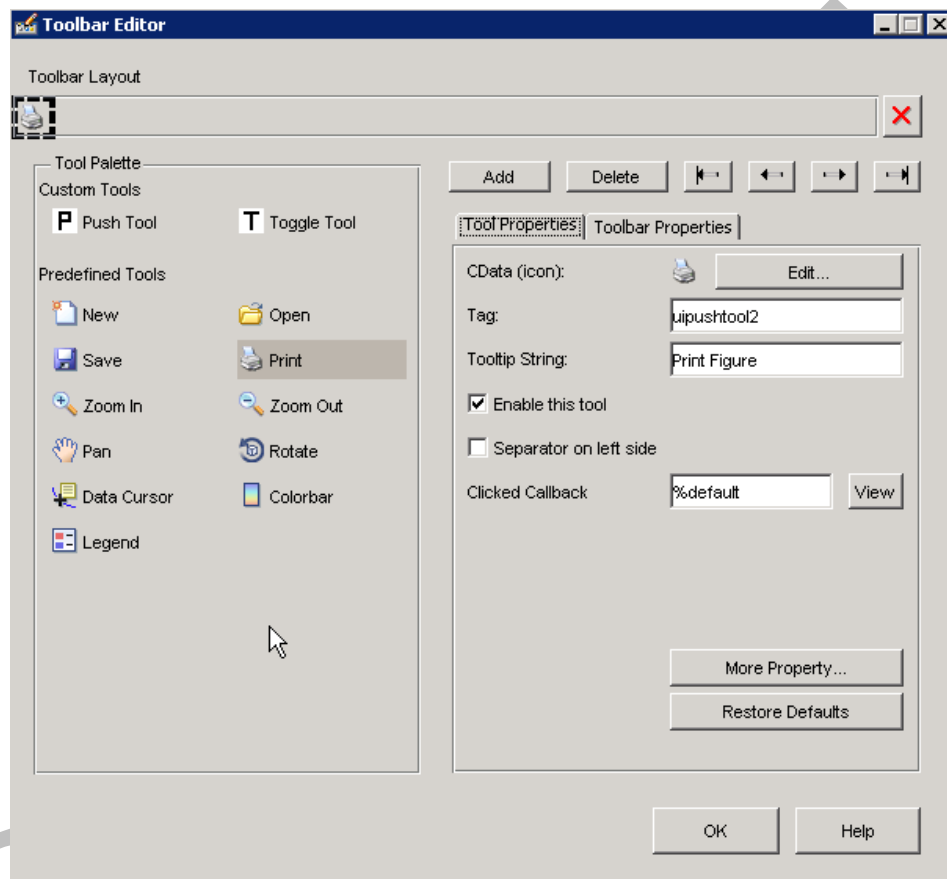
Change the technical name of the button (name also used for the event code) respecting Leszinky-Reddick nomenclature if possible:



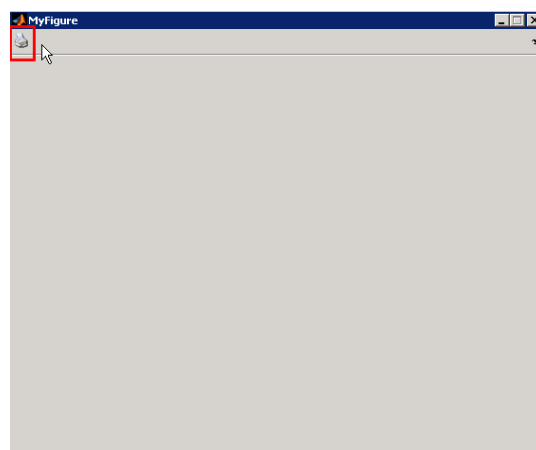
If you want to create toolbars on you figure this is easy. Just click on the **Toolbar Editor** Button:



And just drag and drop to buttons you want in the toolbar:

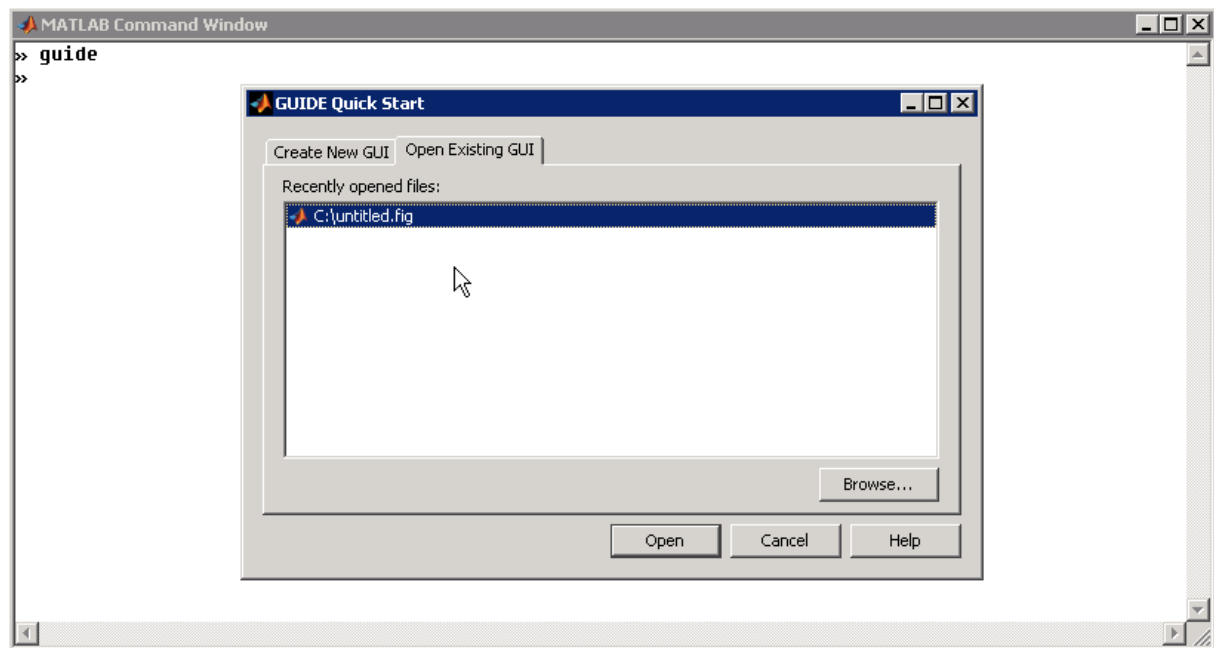


As you can see, it works well:



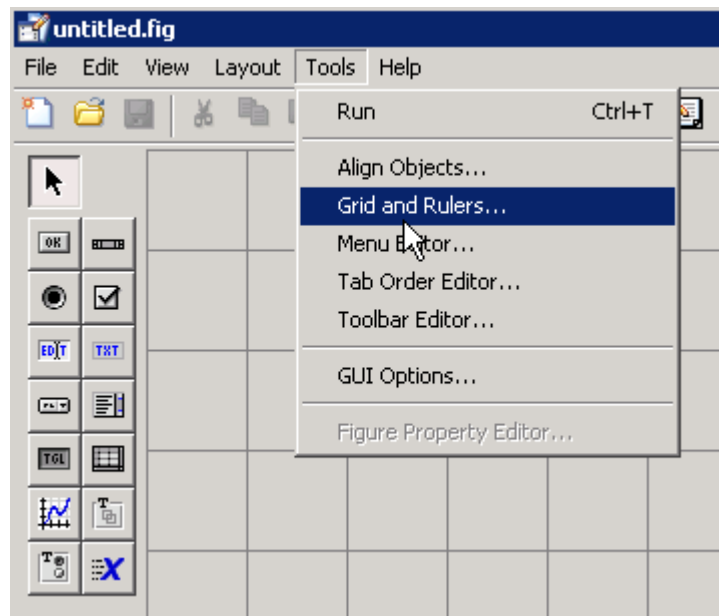
Edit existing GUI

To edit an existing GUI we use the same technic as when we create one:

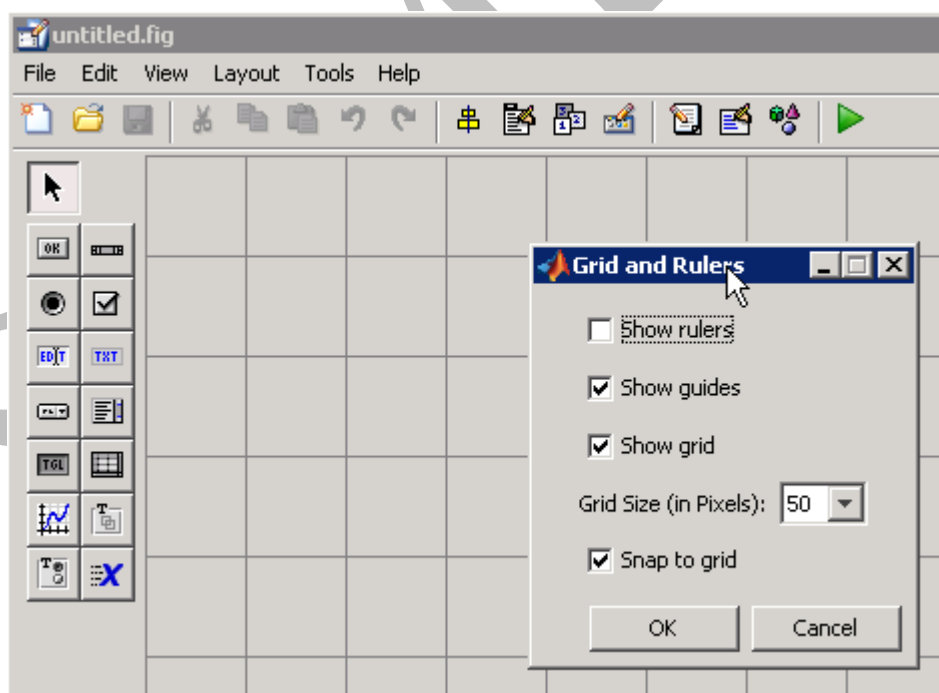


Manage static texts (legends)

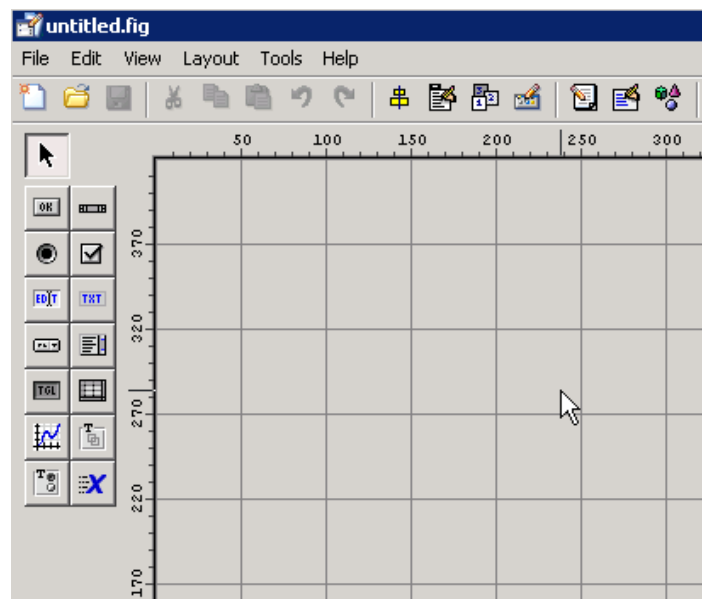
Add **Static Texts**  components is very easy you just need to draw them like on any Microsoft Office software. To quickly align them you can use the Grid and Rulers options:



So you get:



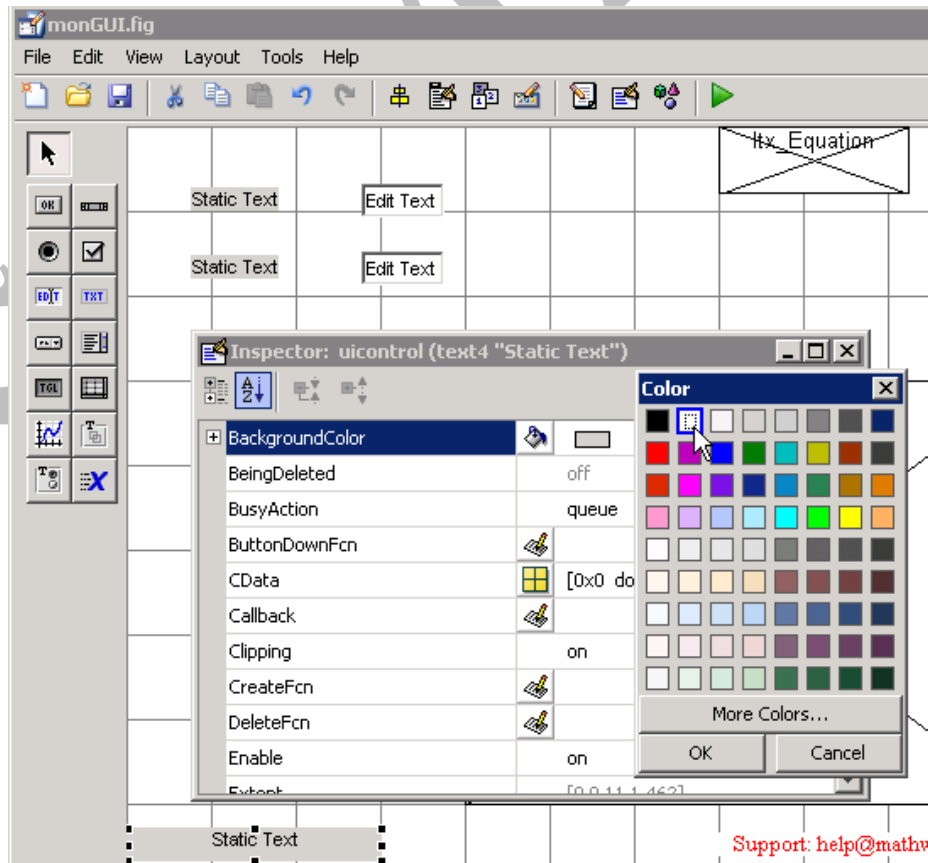
Activate **Show rulers** and the final result will be:



By dragging the rulers you can create guides that will greatly help you to align objects proactively.

Once you have added some **Static Text**  and **Edit Text**  components, let us have a look now to the most used properties.

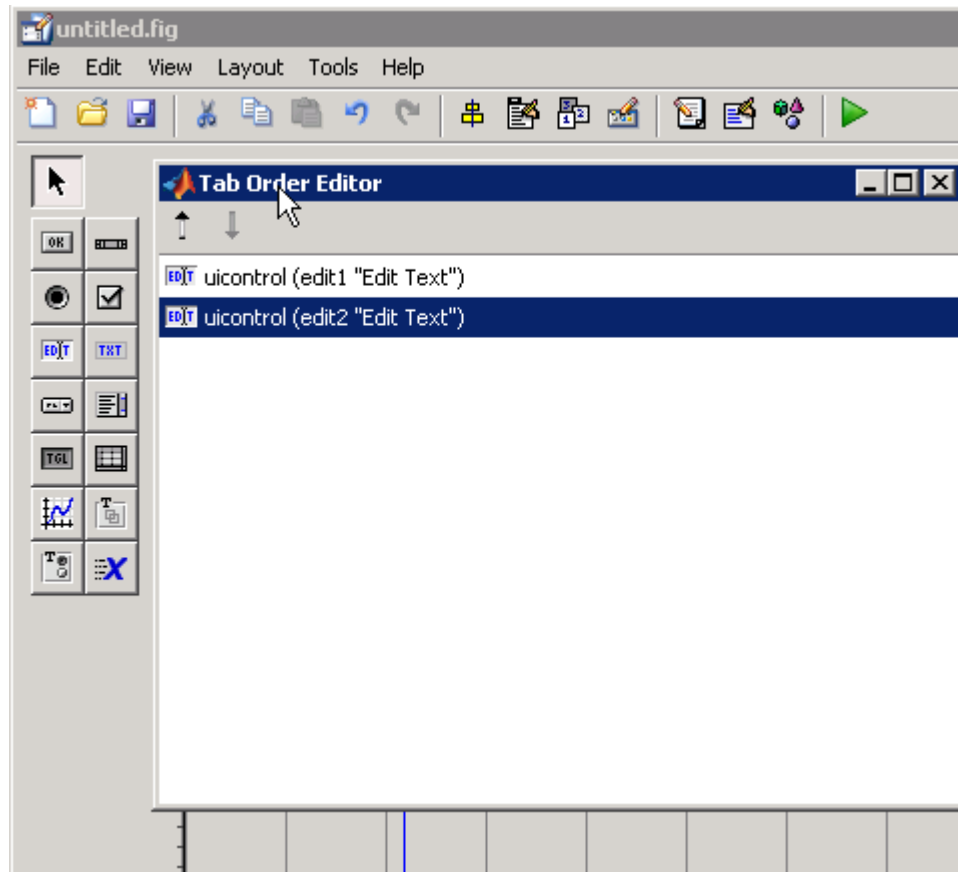
First we change the background color with the property **BackgroundColor**:



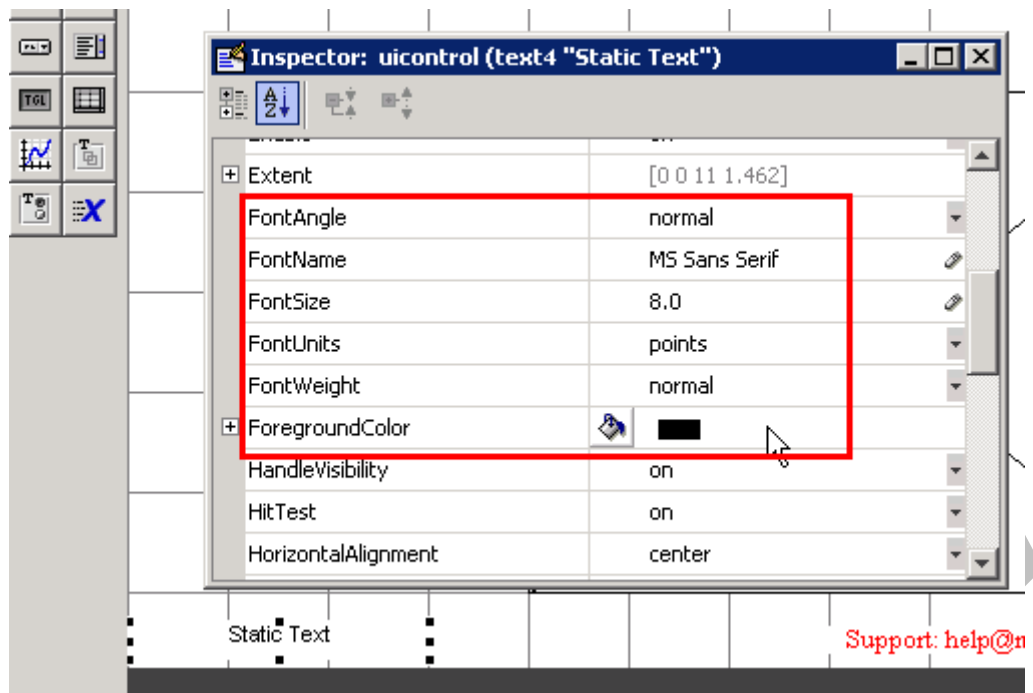
Another important stuff for **Edit Text**  components is to define Tabulation order:



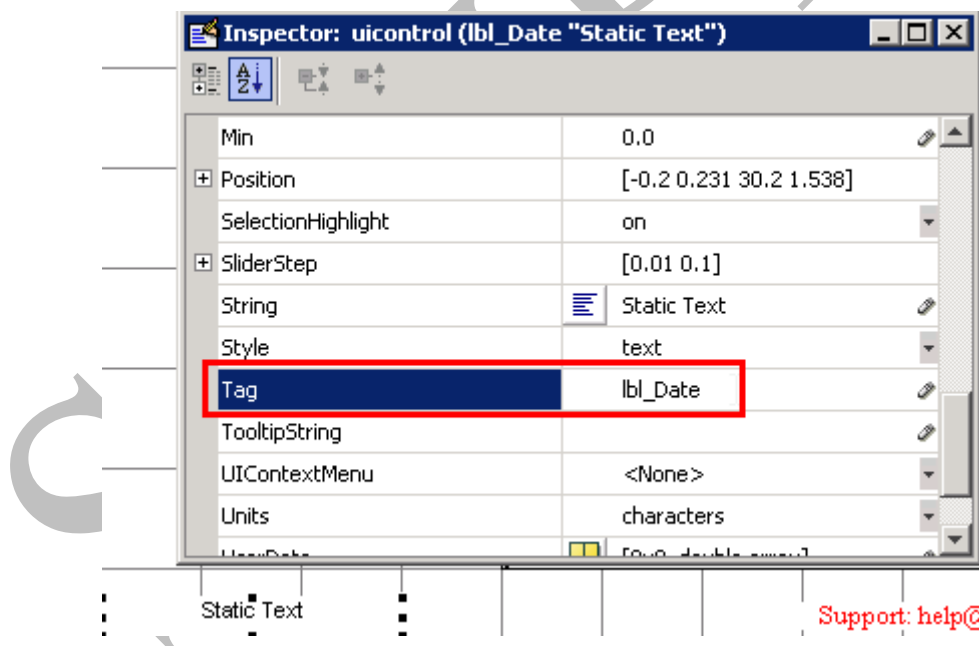
So you get:



You can also change classical font properties but you have to see that you can not mix in the same **Static Text** component different text styles:



As always, we change the component **Tag** property respecting Leszinky-Reddick nomenclature:



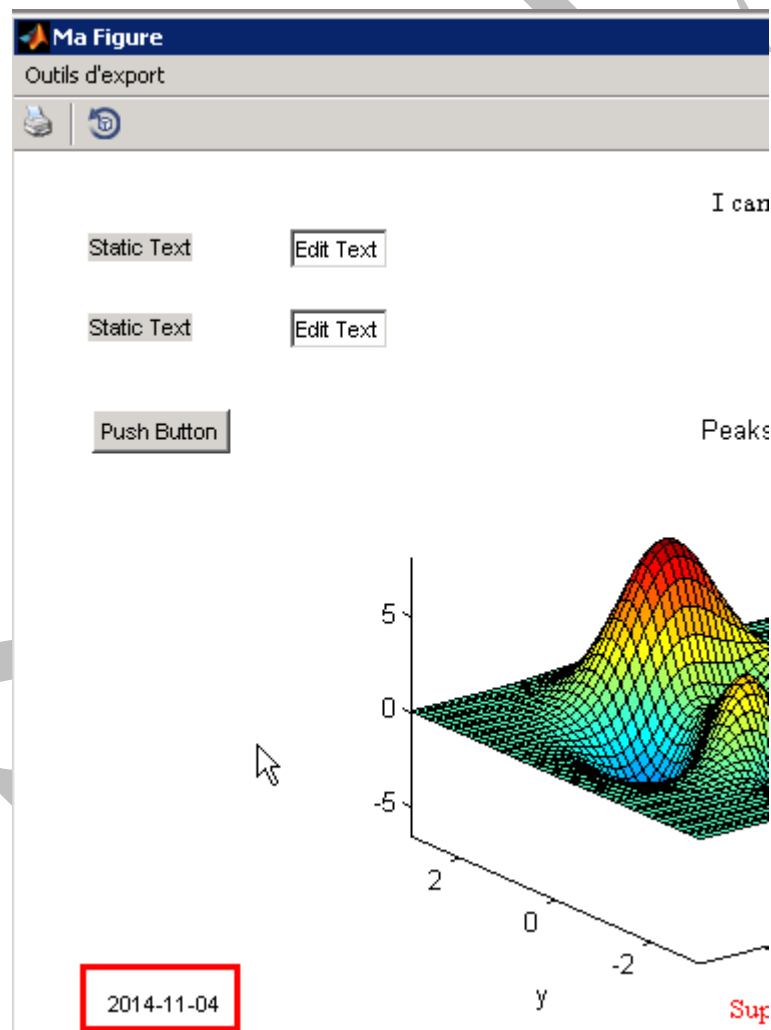
Now we go in the script editor associate to the GUI and the *OpeningFcn* event (callback) we add the following:

```

40
47 % --- Executes just before monGUI is made visible.
48 function monGUI_OpeningFcn(hObject, eventdata, handles, varargin)
49 % This function has no output args, see OutputFcn.
54
55 % Choose default command line output for monGUI
56 handles.output = hObject;
57
58 % Update handles structure
59 guidata(hObject, handles);
60
61 set(handles.lbl_Date,'String',cellstr(datestr(date, 'yyyy-mm-dd')));

```

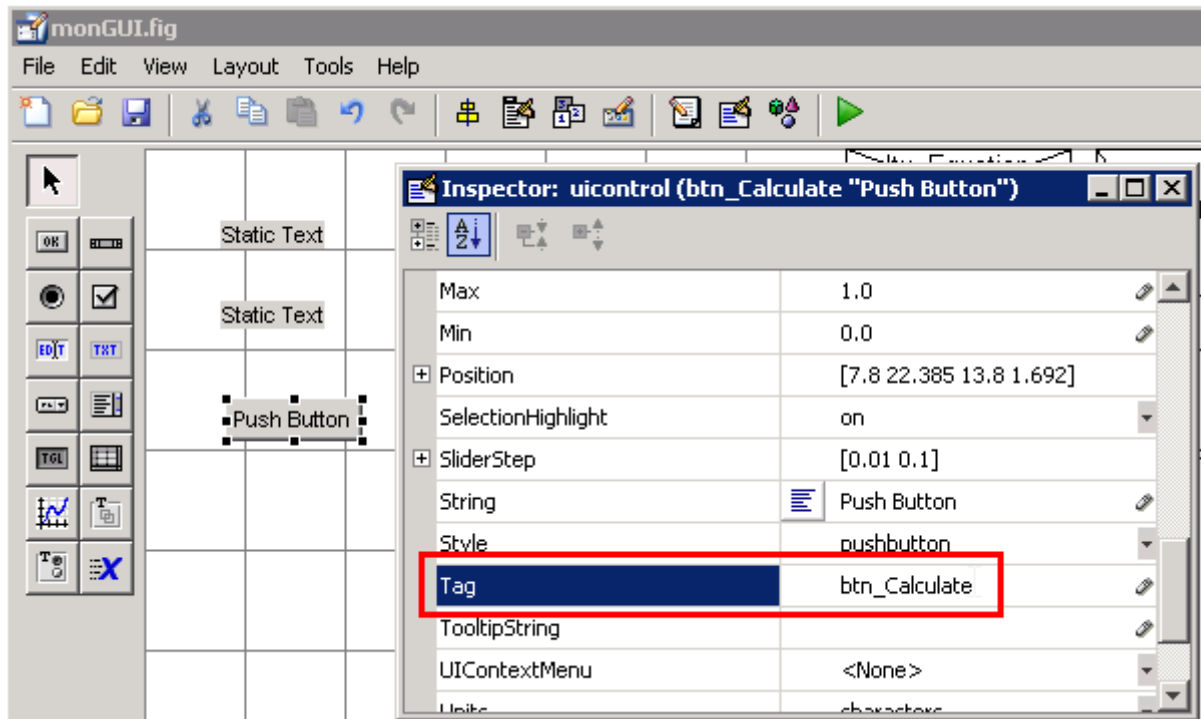
Once this done, we you or ur user open the GUI they get:



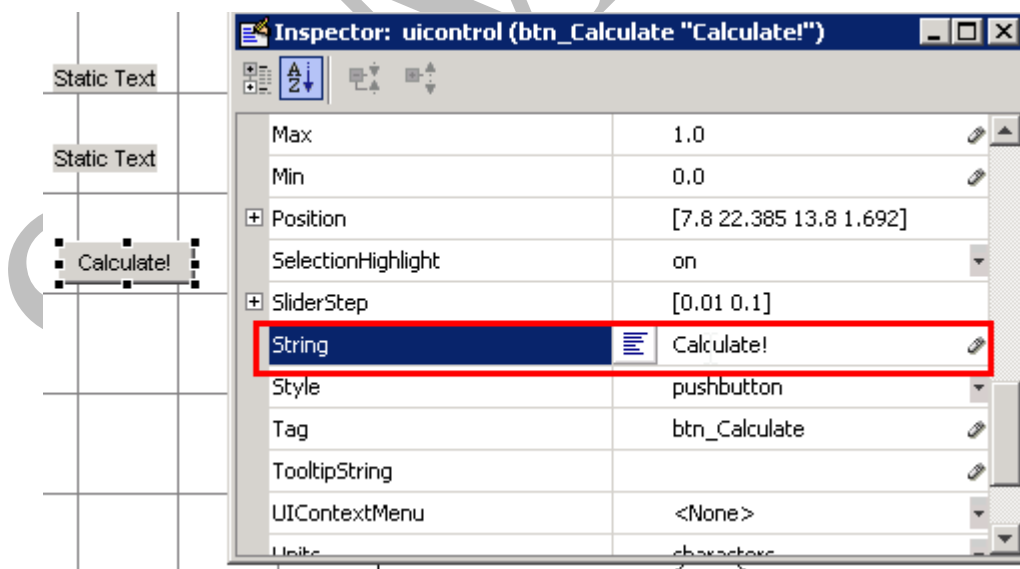
Manage buttons and associate a callback

Adding a button on a GUI is very easy. In fact what interest us is to associate a callback to it.

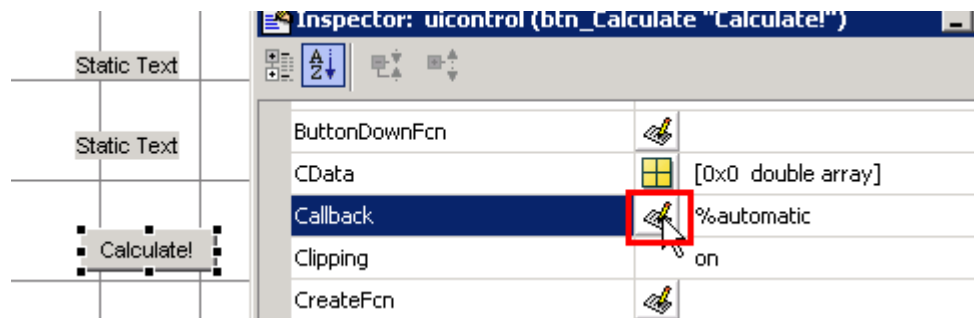
First rename as always the button **Tag** property following Leszinky-Reddick nomenclature:



Change also the string in the button to be more explicit....:



And after we change it's associated **Callback** (event) property:



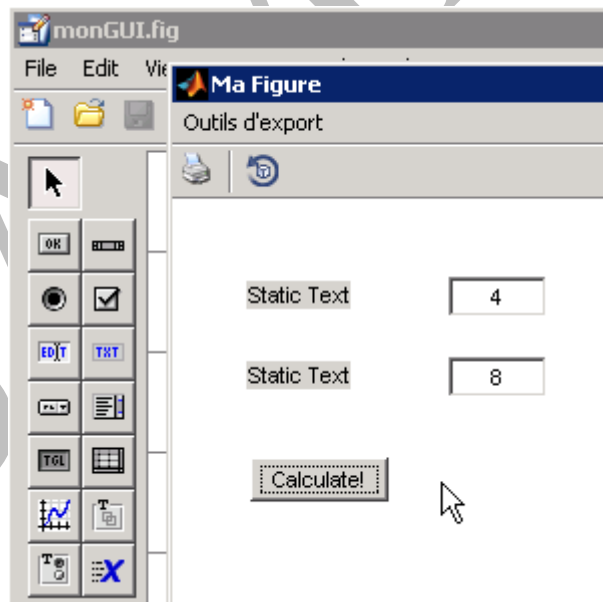
And we add a simple multiplication where we read the value from the first **Edit** field (*edit1*) and we write the result in the second one (*edit2*):

```

148
149 % --- Executes on button press in btn_Calculate.
150 function btn_Calculate_Callback(hObject, eventdata, handles)
151 % hObject    handle to btn_Calculate (see GCBO)
152 % eventdata  reserved - to be defined in a future version of MATLAB
153 % handles    structure with handles and user data (see GUIDATA)
154
155 sng_Value=str2double(get(handles.edit1,'String'))*2;
156 set(handles.edit2,'String',sng_Value);

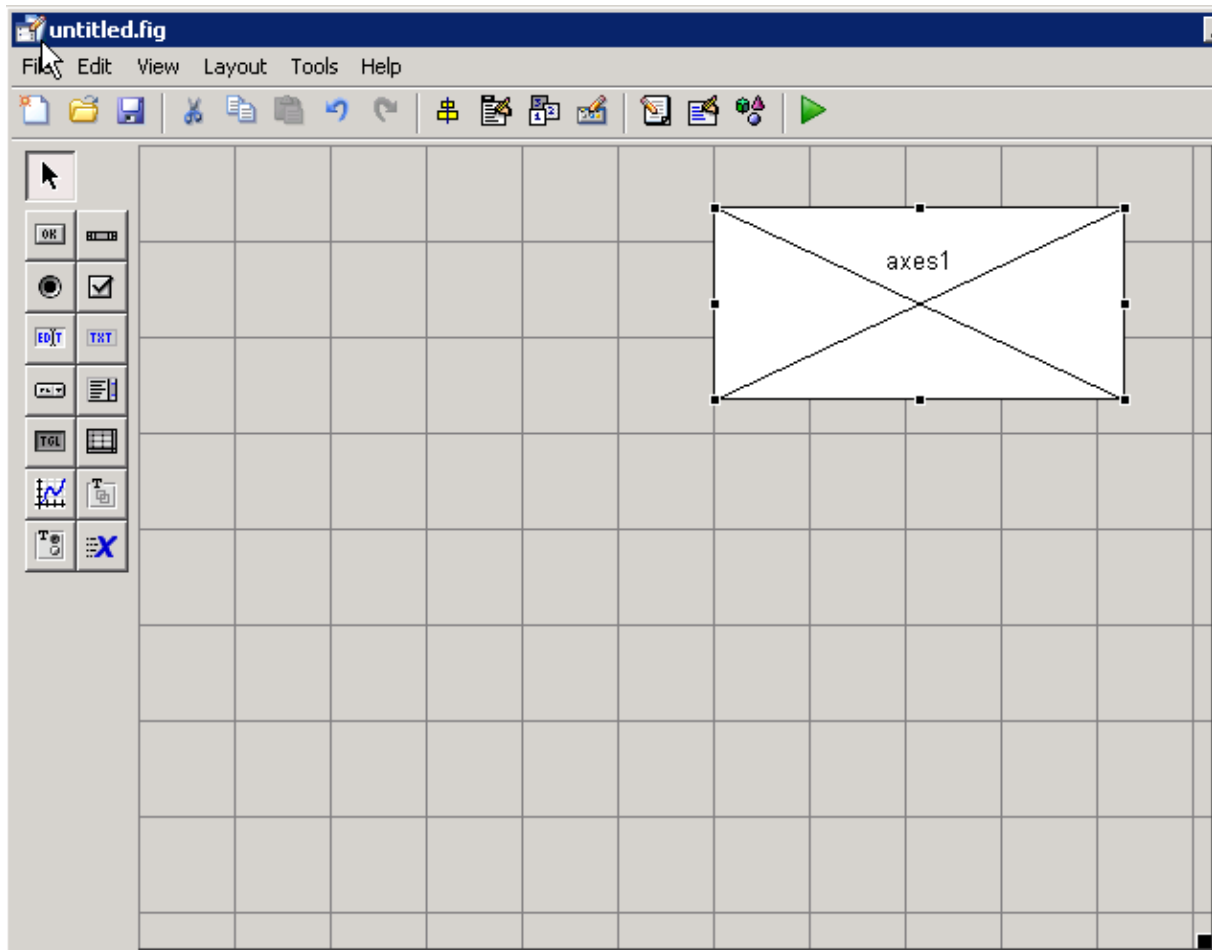
```

So the results will look like:

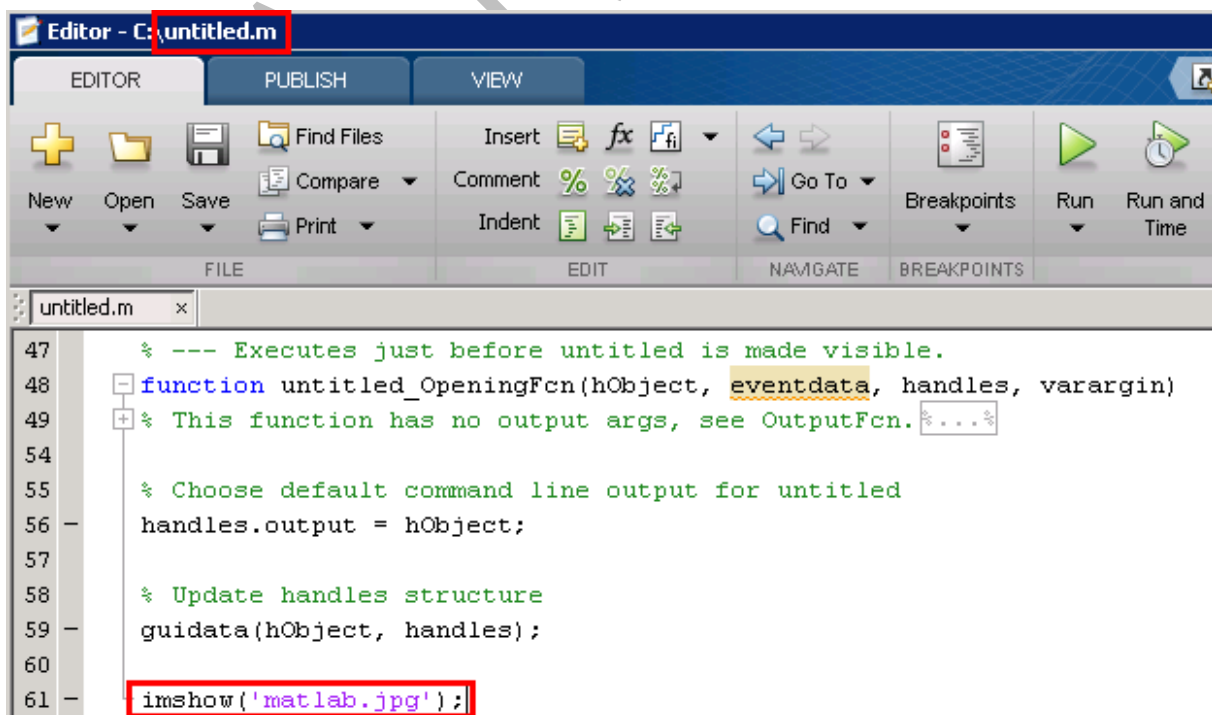


Create axes area for figures, images,

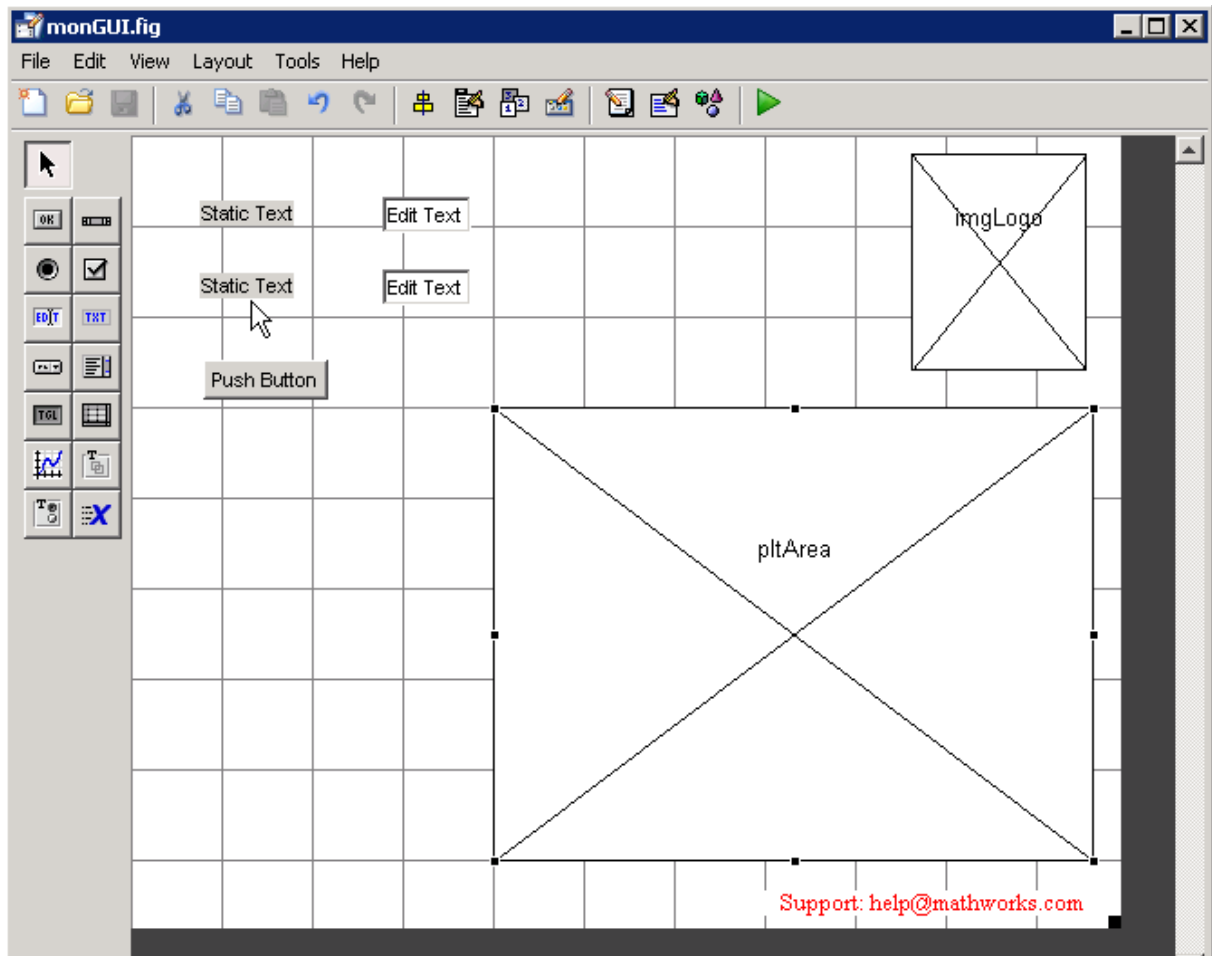
You have for this subject to consider two scenarios. The first is that you only have one unique axes object on your userform like here:



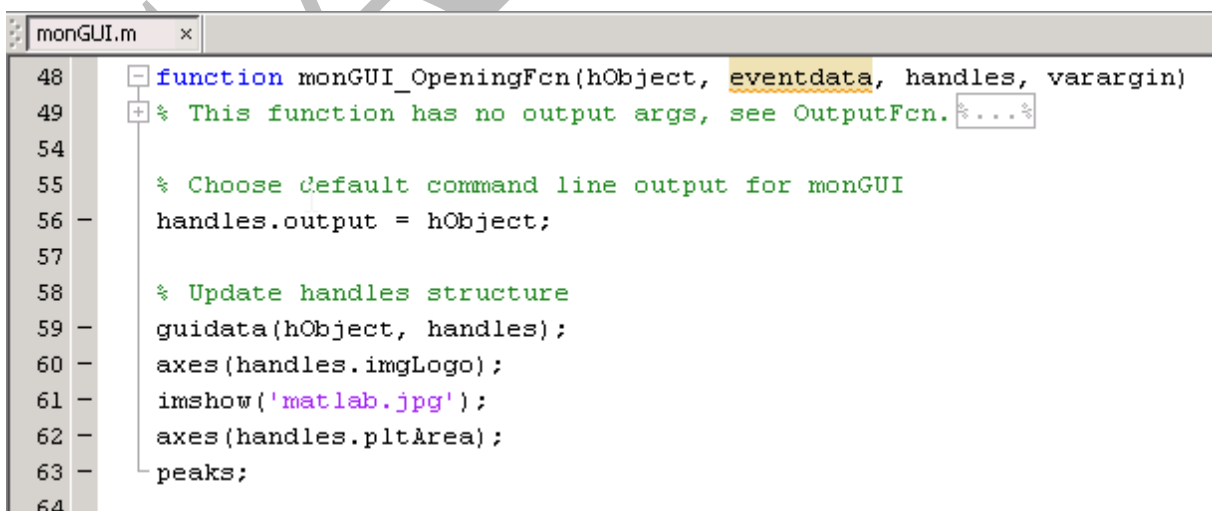
Then to fill it with an image it's quite easy. You just go in the associated script file and add in the *OpeningFcn* event the following line:



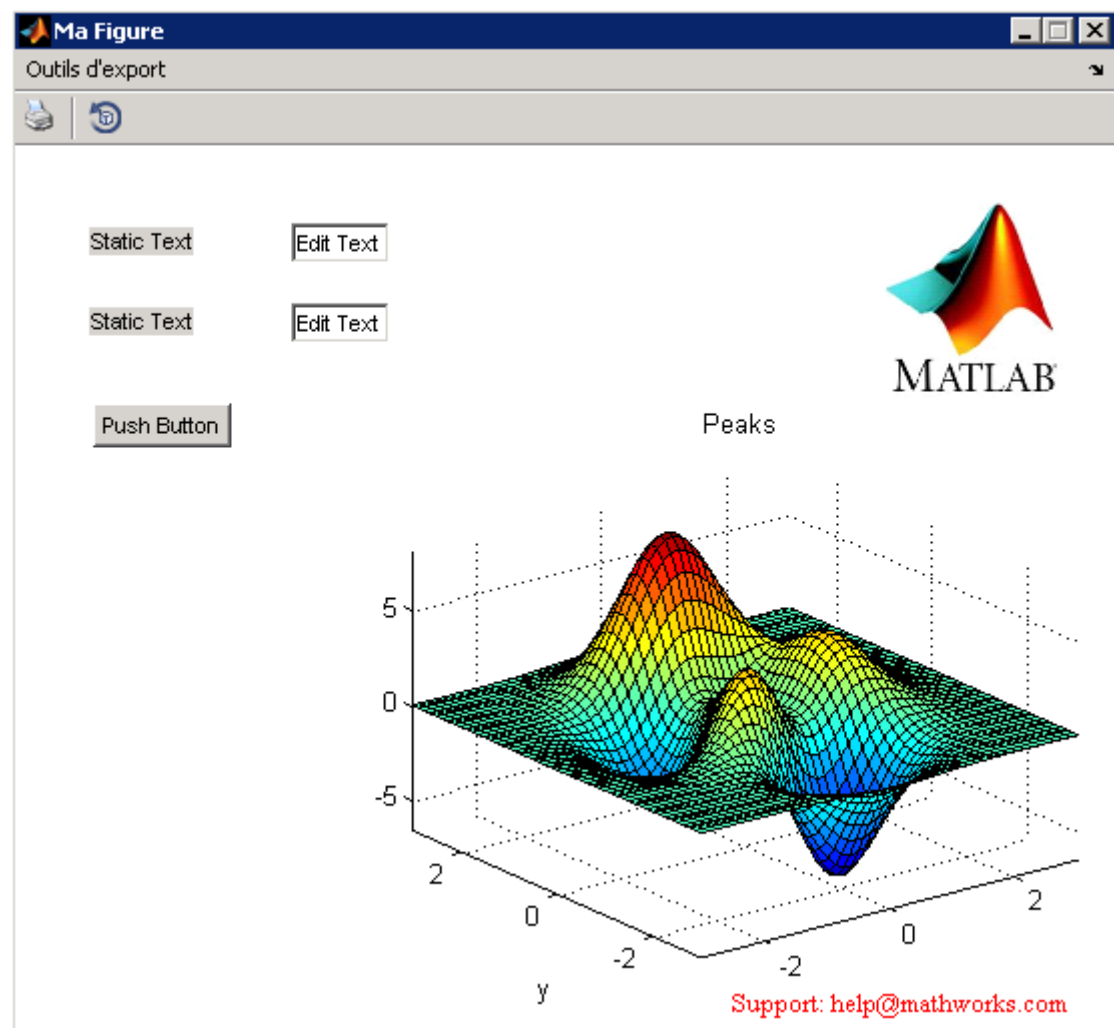
When there are more than one axes area it is not much more difficult. Consider first the following situation where we change the **Tag** property of the two axes components:



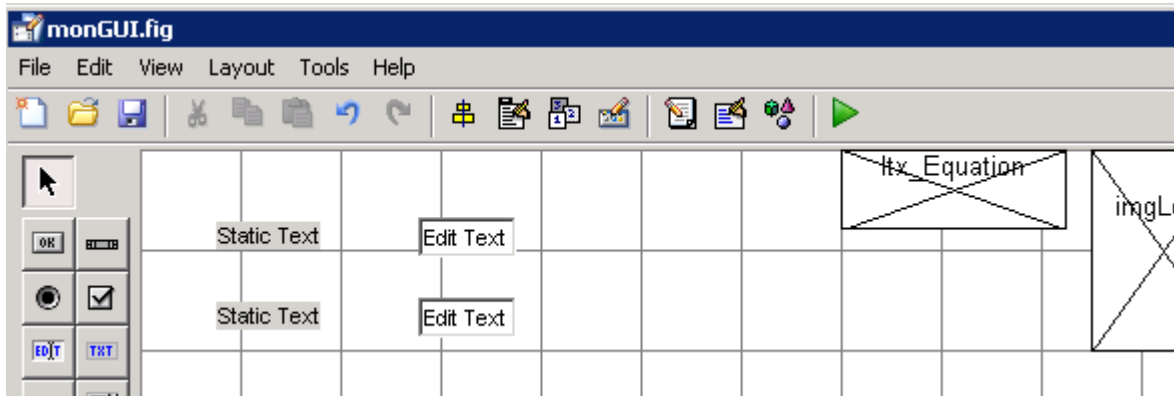
We just change the script as following:



And we get:



The use of axes seems also the only way actually to write LaTeX on a GUI. First add an axes component where you want and resize it and change its **Tag** property (here we chose *ltx_Equation*):



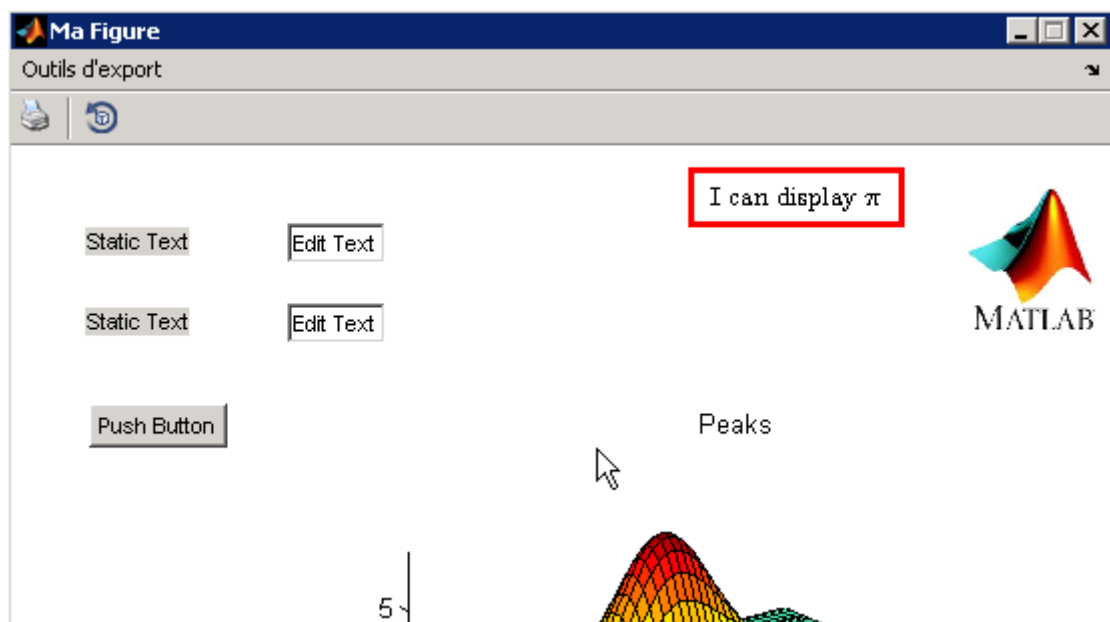
Once this is done, the idea in this particular example will be to write a LaTeX text once the GUI is opened. For this purpose we go in the script editor associated to the GUI and we write in the *OpeningFcn* event (callback) the following:

```

monGUI.m
47 % --- Executes just before monGUI is made visible.
48 function monGUI_OpeningFcn(hObject, eventdata, handles, varargin)
49 % This function has no output args, see OutputFcn.
54
55 % Choose default command line output for monGUI
56 handles.output = hObject;
57
58 % Update handles structure
59 guidata(hObject, handles);
60
61 axes(handles.ltx_Equation);
62 axis off;
63 text(0,0.5,'I can display $\pi$', 'interpreter', 'latex', ...
64 'horiz', 'left', 'vert', 'middle')

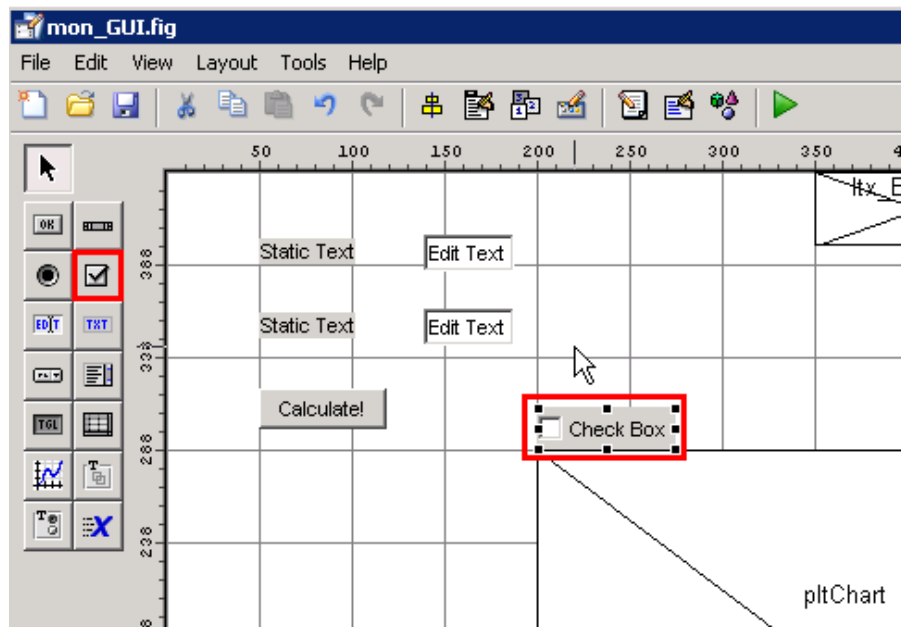
```

Once this is done, when we or the user opens the GUI, we get:

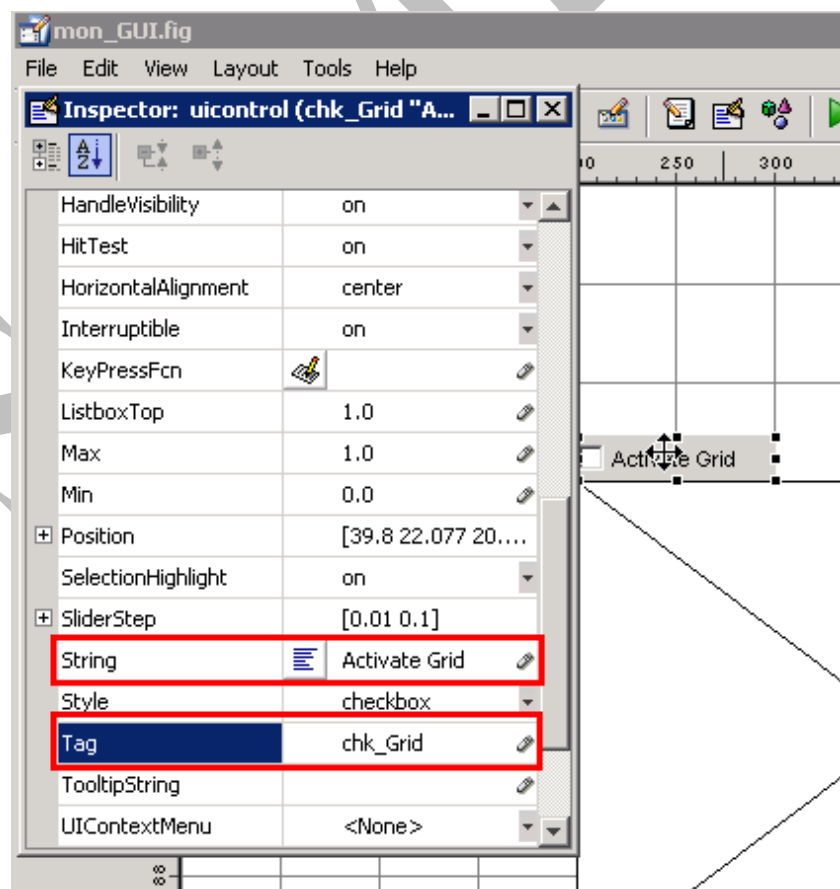


Manage check-boxes and associate a callback

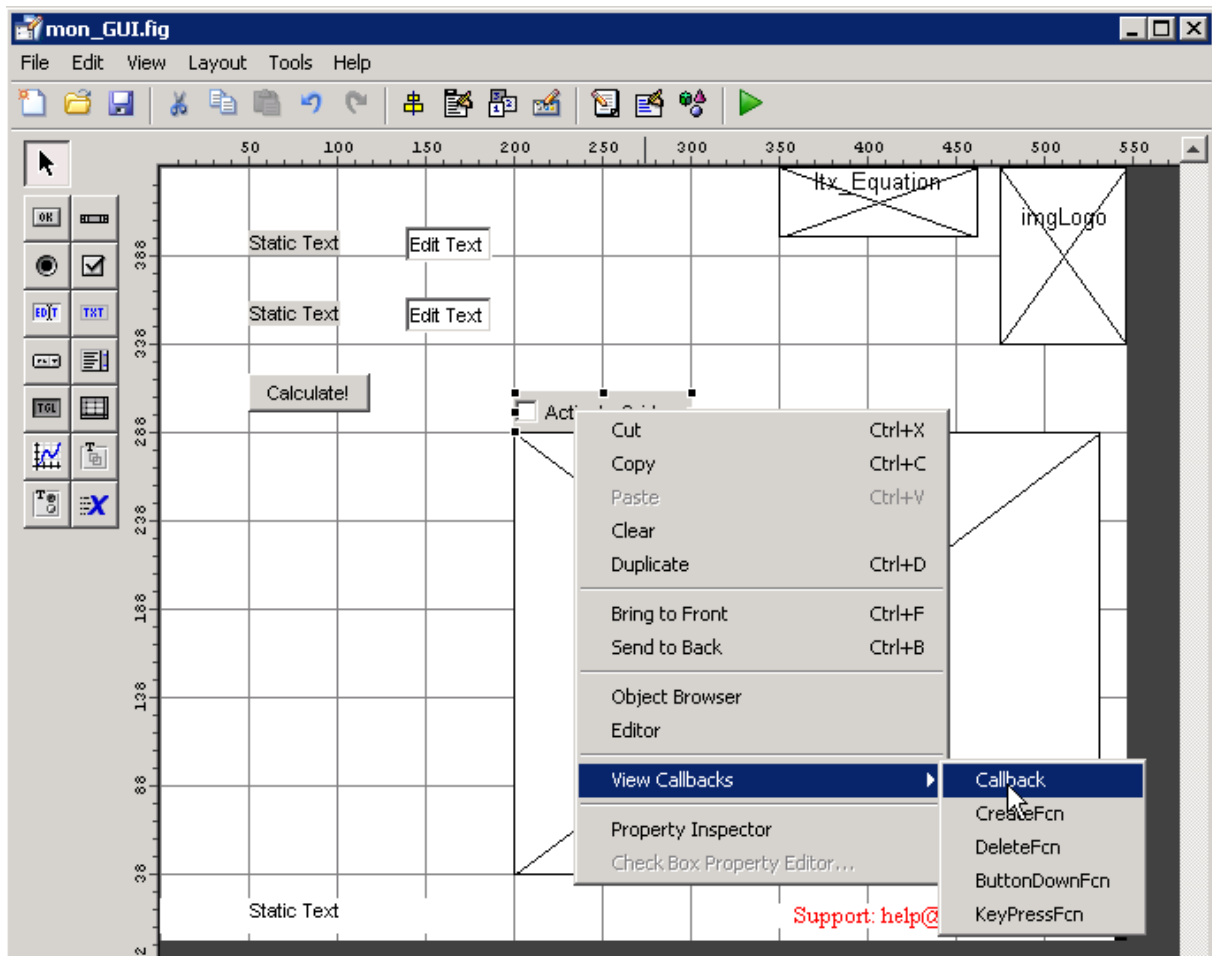
We will see the idea of using **Checkboxes**  by a very simple case. First add such a component:



We change the classic stuff:



Now we go to the Callback of the checkbox:



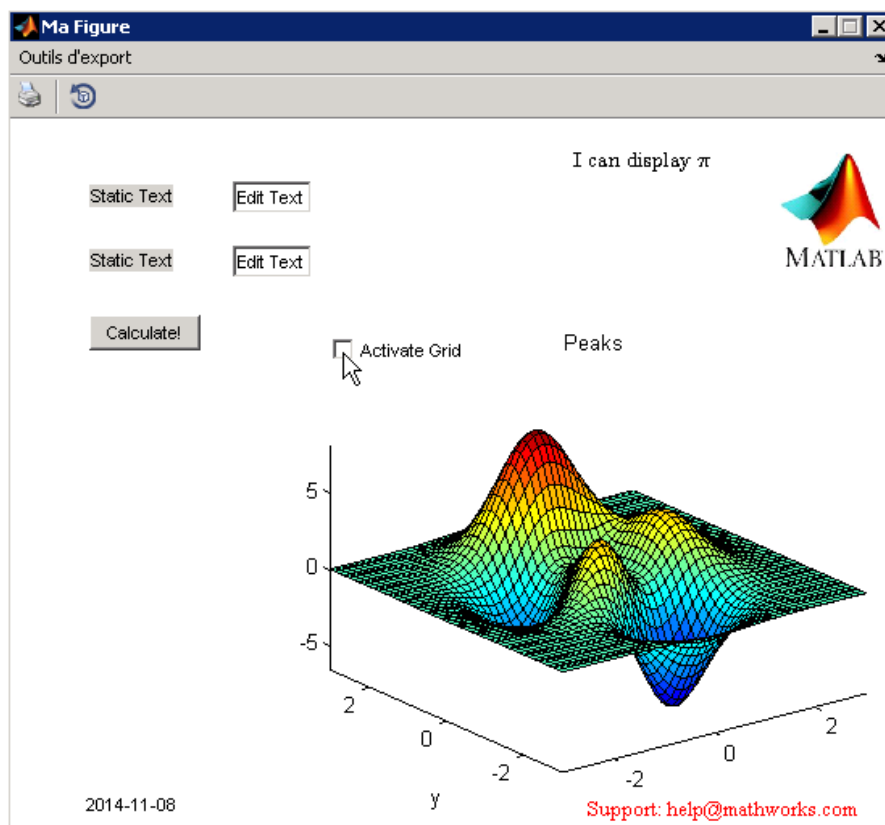
And so we write:

```

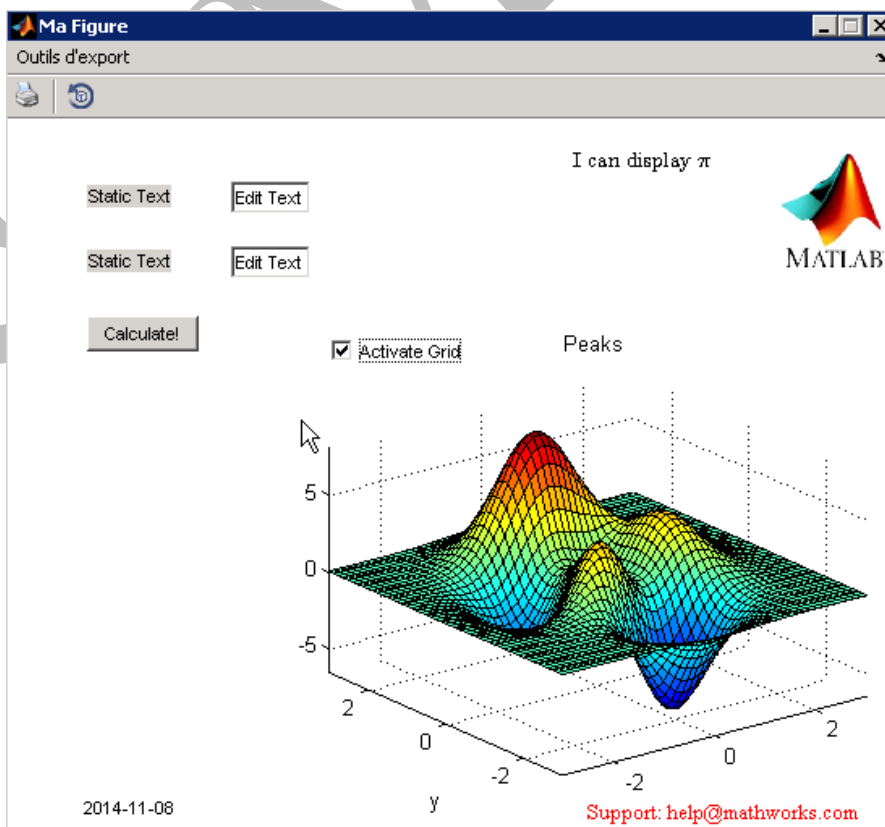
mon_GUI.m
156 - set(handles.edit2,'String',sng_Value);
157
158
159 % --- Executes on button press in chk_Grid.
160 function chk_Grid_Callback(hObject, eventdata, handles)
161 % hObject    handle to chk_Grid (see GCBO)
162 % eventdata  reserved - to be defined in a future version of MATLAB
163 % handles    structure with handles and user data (see GUIDATA)
164
165 % Hint: get(hObject,'Value') returns toggle state of chk_Grid
166 - chk_Grid_Status=get(hObject,'Value');
167 - if chk_Grid_Status==1
168 -     axes(handles.pltChart);
169 -     grid on;
170 - else
171 -     axes(handles.pltChart);
172 -     grid off;
173 - end

```


And now if we execute the GUI we get:

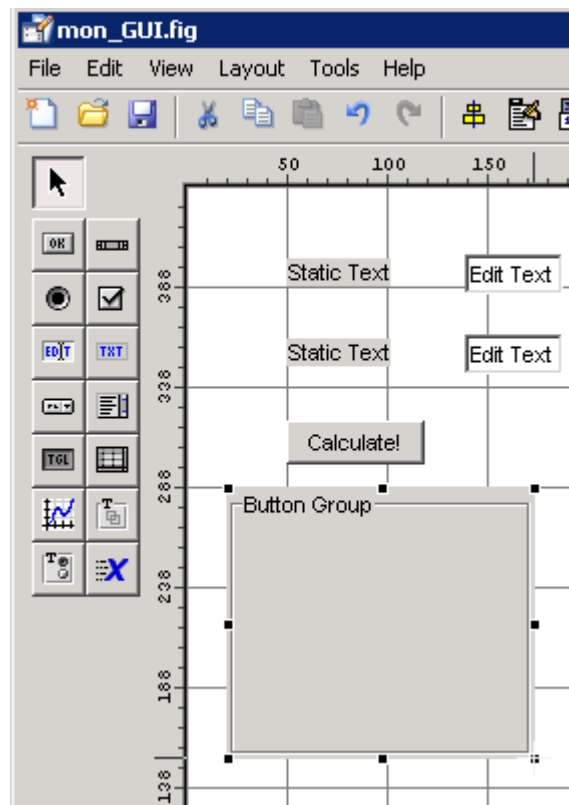


And with the checkbox activated:

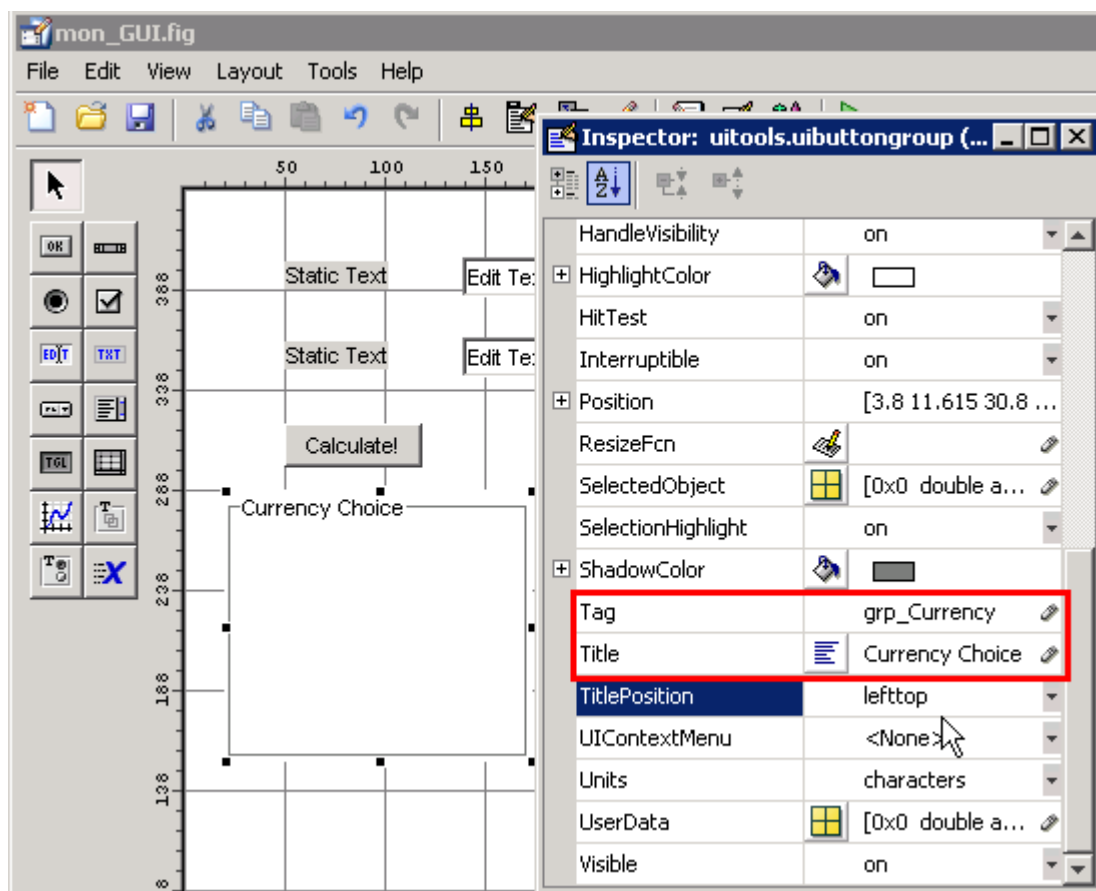


Manage radio buttons group

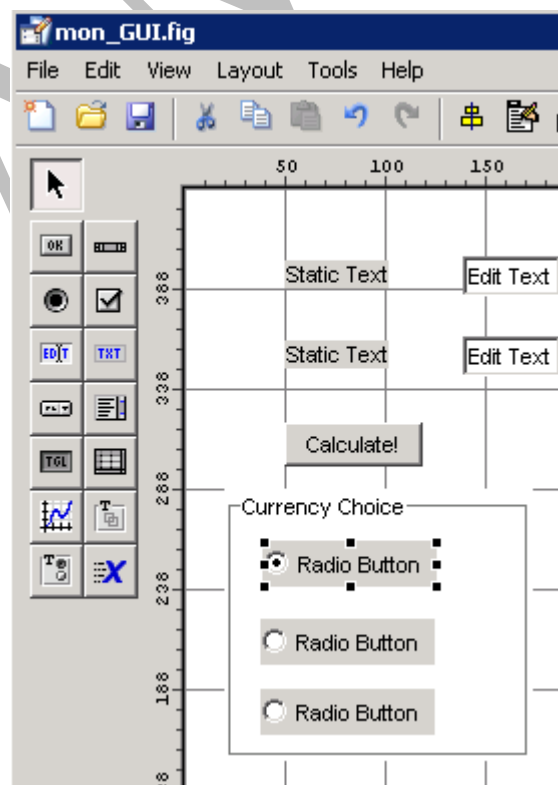
Another famous type or components are **Radio buttons** . A good practice before inserting them is to add a **Buttons group**  before. So add this one first:



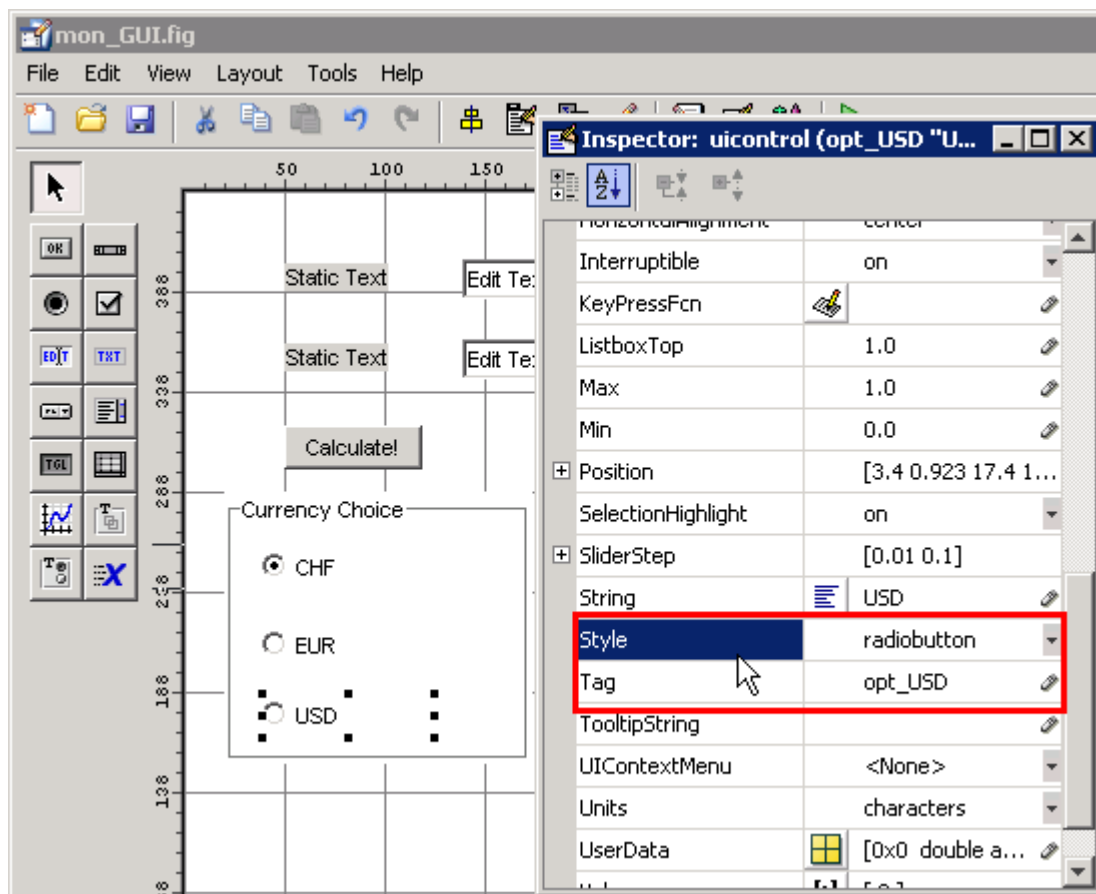
Change its **BackgroundColor**, **Title** and **Tag** properties:



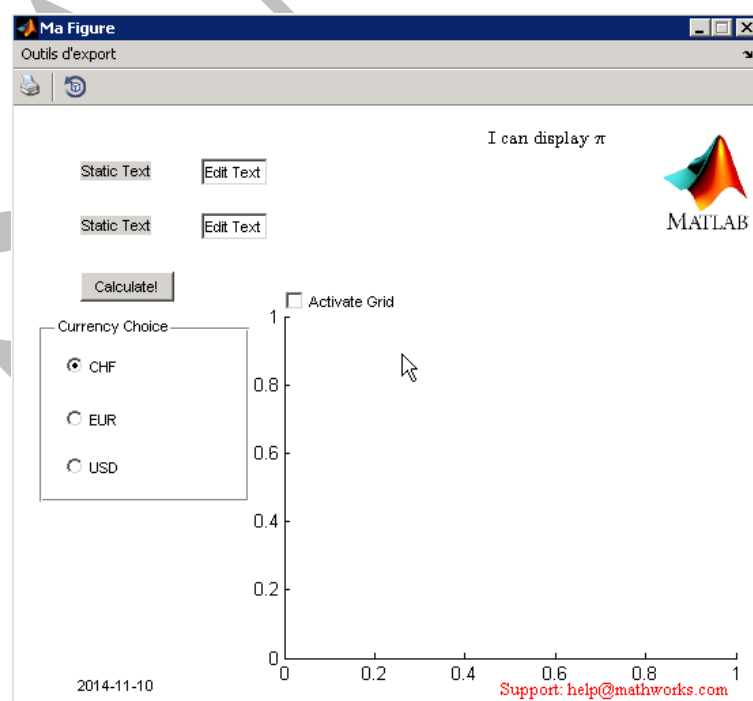
Once this is done add for example three radio buttons:



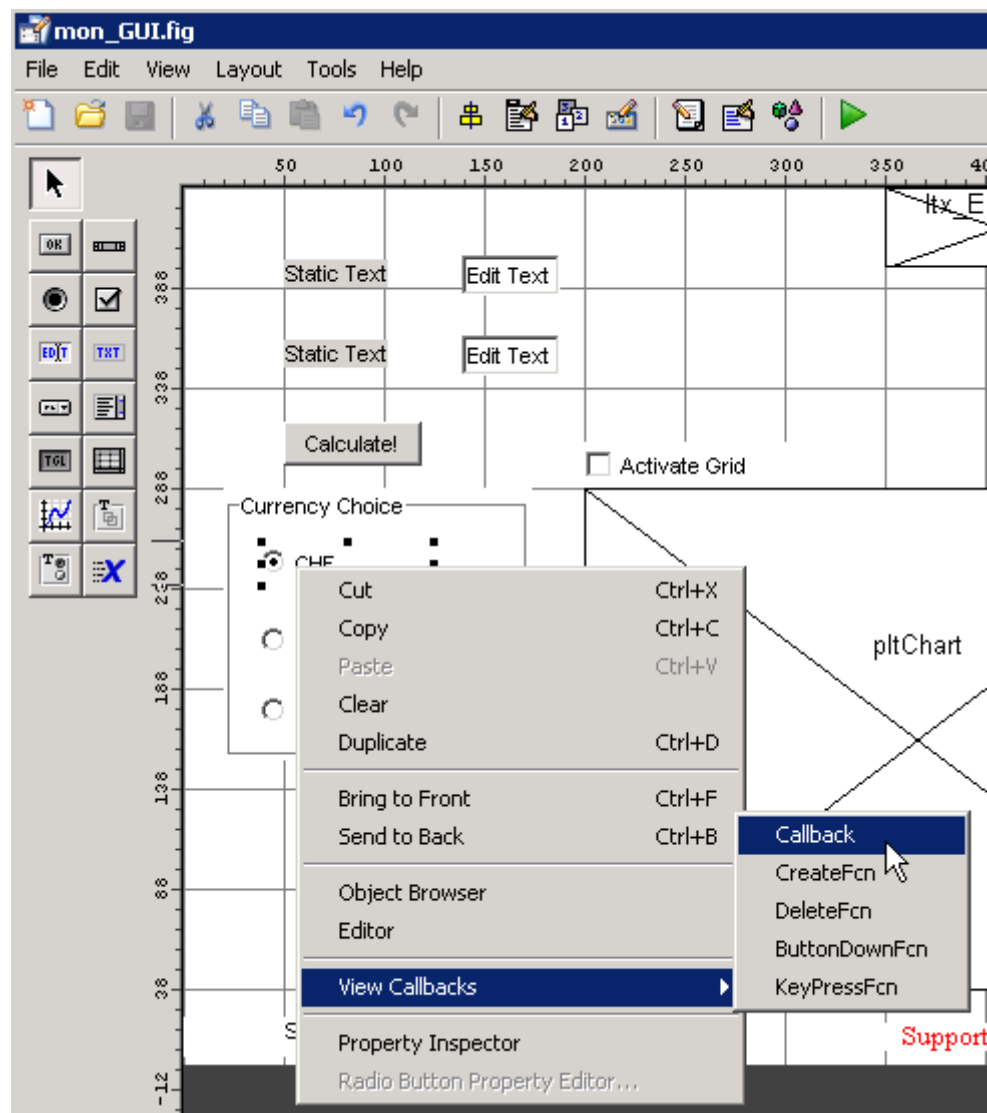
And also change their **BackgroundColor**, **Title** and **Tag** properties:



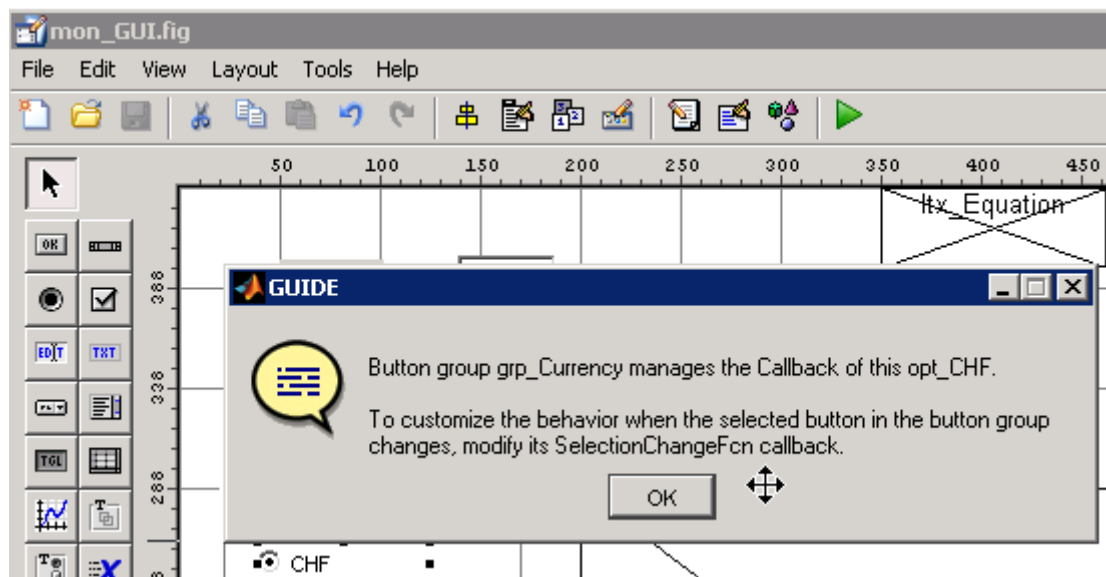
The first thing you can already see if you execute the GUI is that you can't activate all radio buttons at the same time because of the Button group area:



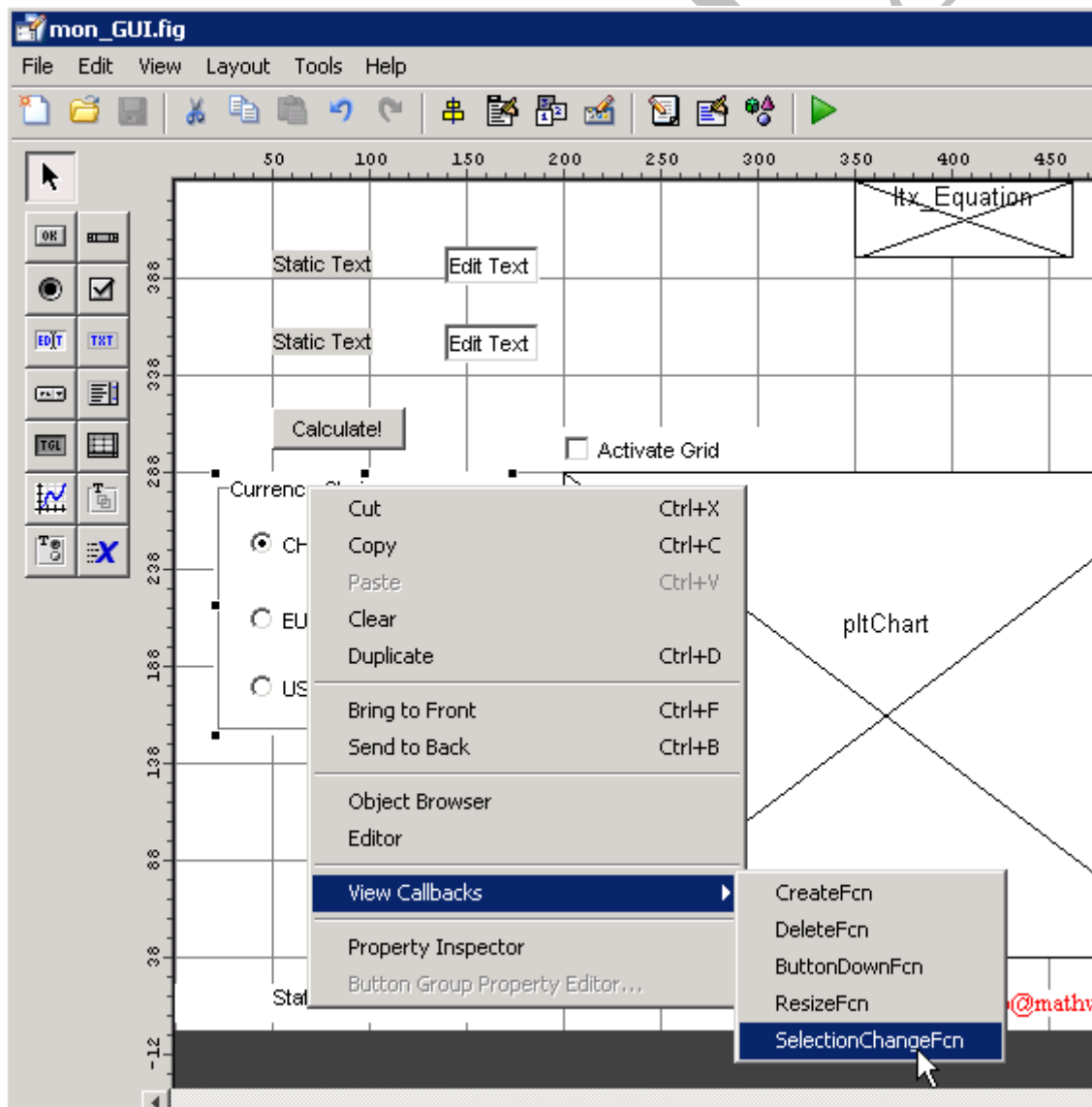
Now we will associate a Callback to the radiobuttons. Here we show only for the first one because the ideas are exactly the same for the others:



You will have:



You click on **OK** and we take as recommended the **SelectionChangeFcn** callback:



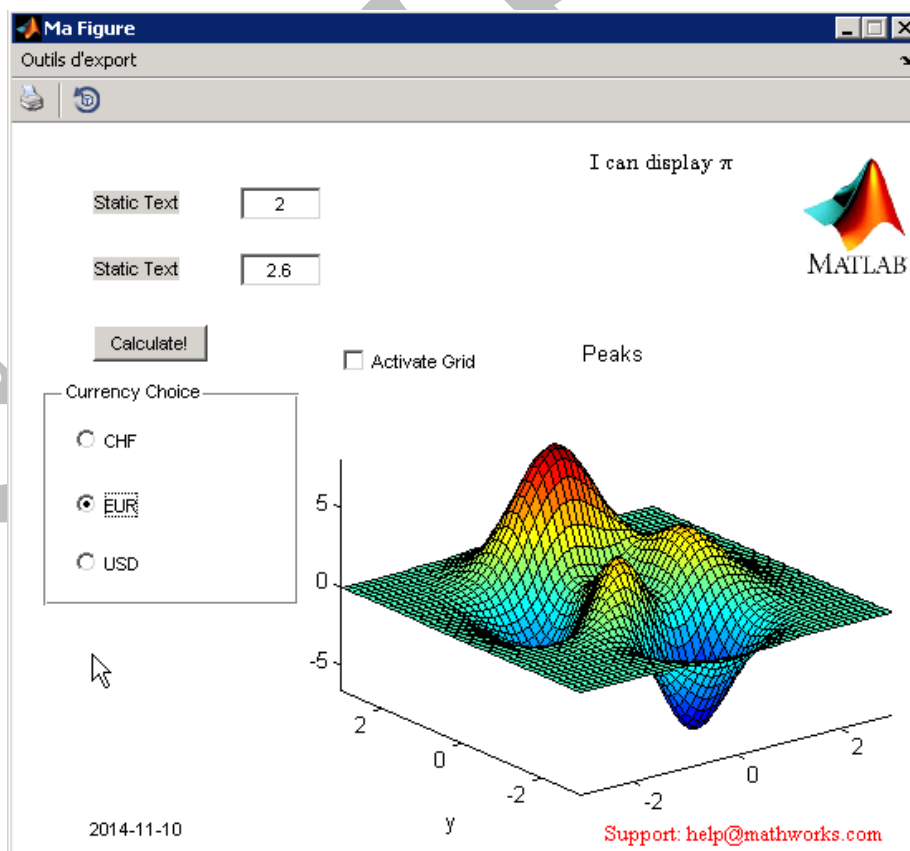
And we write:

```

174
175
176 % --- Executes when selected object is changed in grp_Currency.
177 function grp_Currency_SelectionChangeFcn(hObject, eventdata, handles)
178 % hObject    handle to the selected object in grp_Currency
179 % eventdata  structure with the following fields (see UIBUTTONGROUP)
180 %   EventName: string 'SelectionChanged' (read only)
181 %   OldValue: handle of the previously selected object or empty if none was selected
182 %   NewValue: handle of the currently selected object
183 % handles    structure with handles and user data (see GUIDATA)
184 switch get(eventdata.NewValue,'Tag') % Get Tag of selected object.
185 case 'opt_CHF'
186     sng_Value=str2double(get(handles.edit1,'String'))*1.2;
187 case 'opt_EUR'
188     sng_Value=str2double(get(handles.edit1,'String'))*1.3;
189 case 'opt_USD'
190     sng_Value=str2double(get(handles.edit1,'String'))*1;
191 end
192 set(handles.edit2,'String',sng_Value);
193

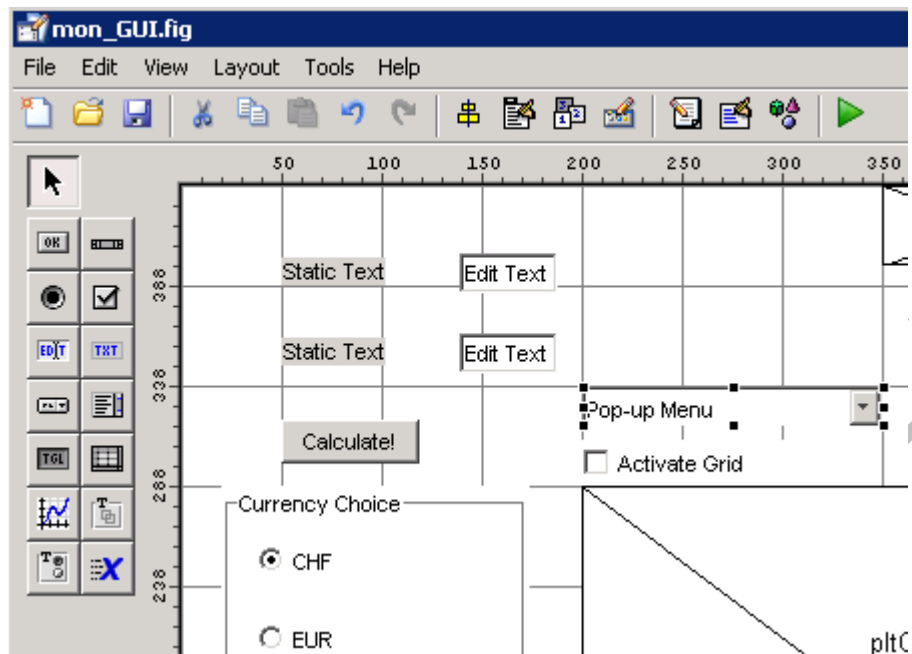
```

and so we get the expected result:

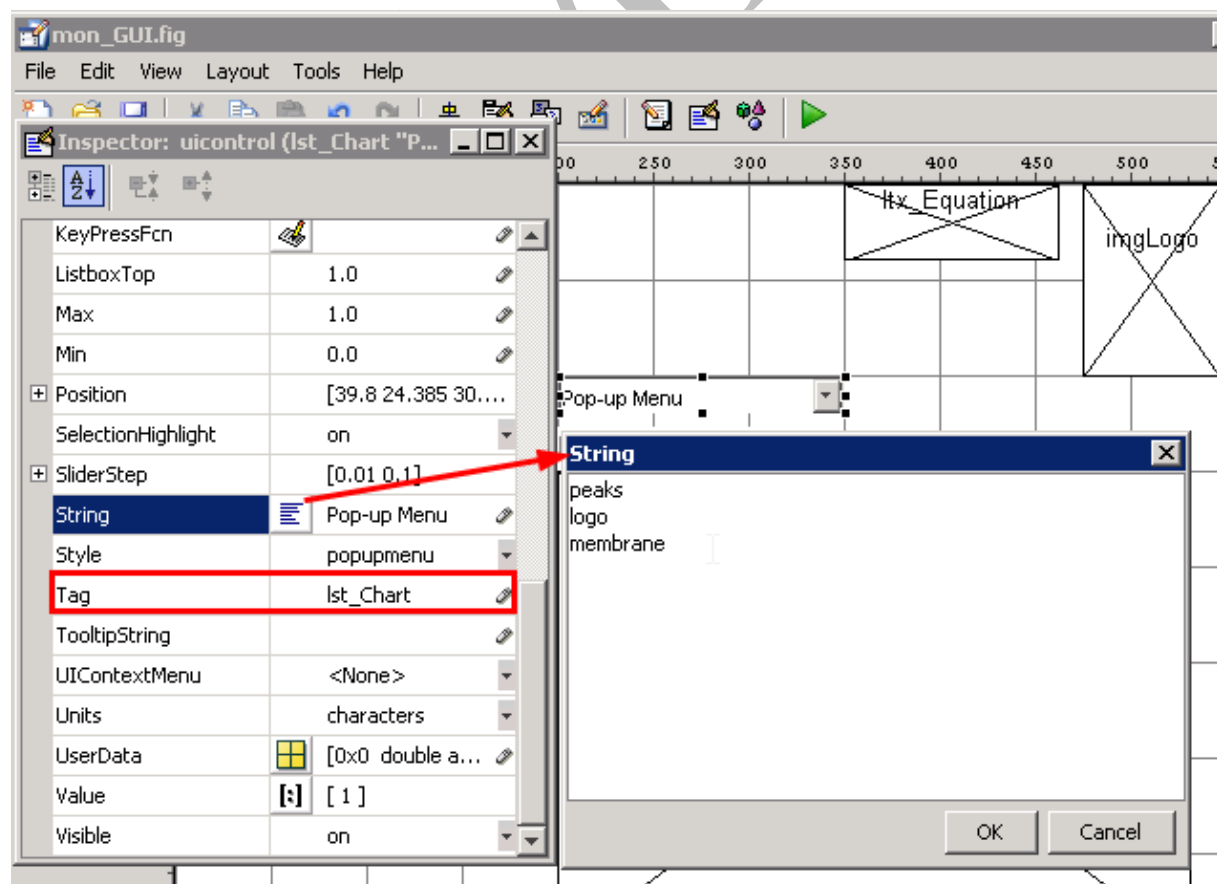


Manage pop-up menu and get selected value

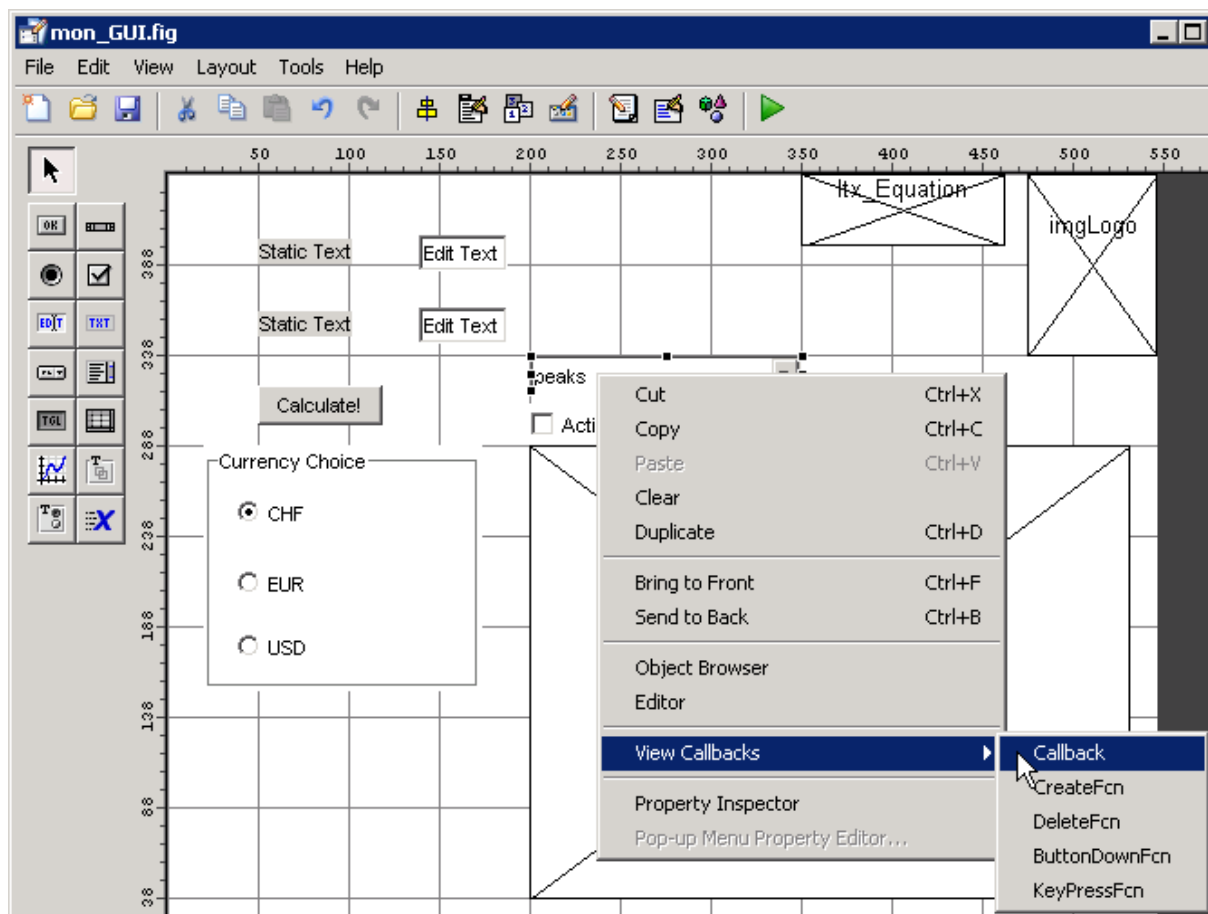
We want to choose in a static combo-box a selection of charts. For this purpose add pop-up menu  control in the GUI:



Open its properties and change the **String** and **Tag** properties:



Once done, open its **Callback**:



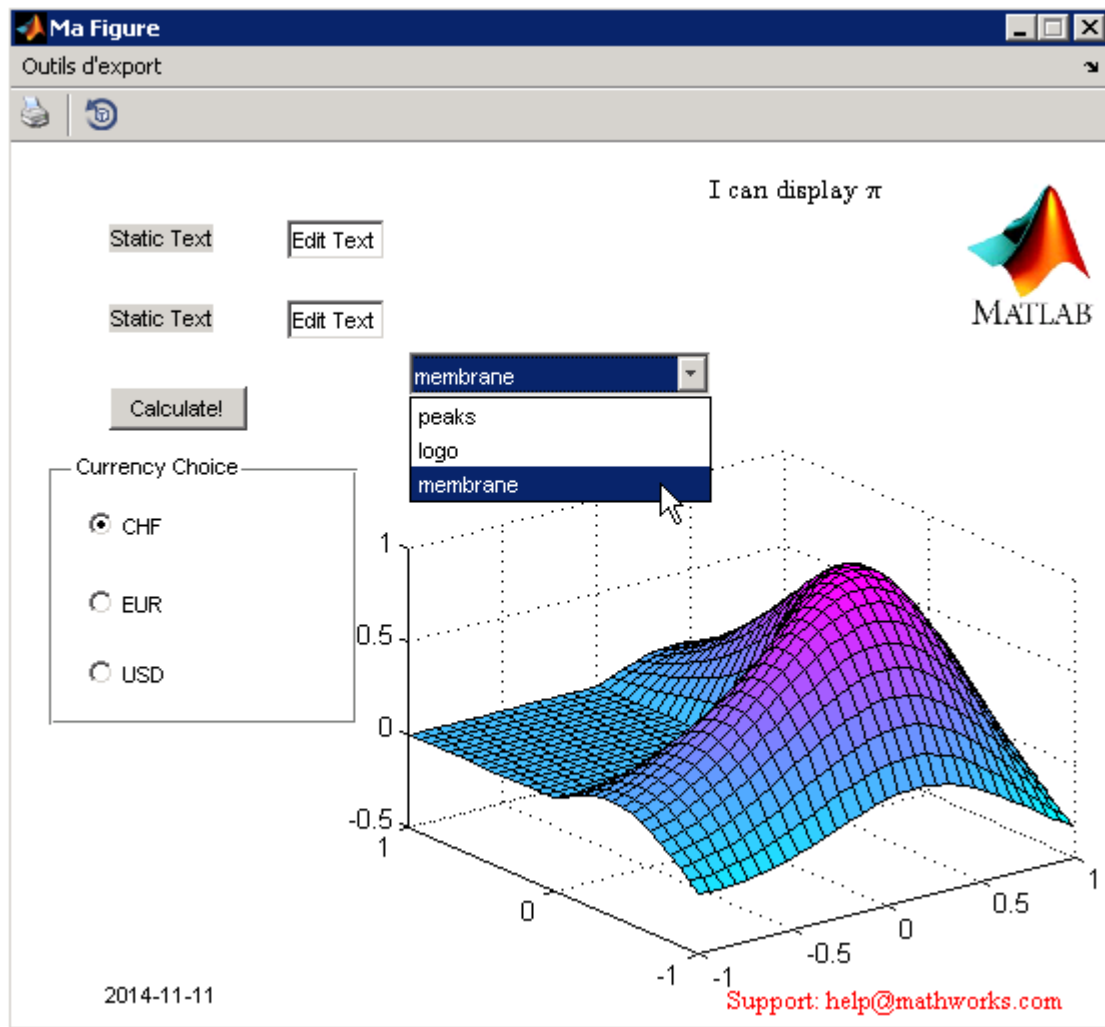
and add the following code:

```

194
195 % --- Executes on selection change in 1st_Chart.
196 function 1st_Chart_Callback(hObject, eventdata, handles)
197 % hObject handle to 1st_Chart (see GCBO)
198 % eventdata reserved - to be defined in a future version of MATLAB
199 % handles structure with handles and user data (see GUIDATA)
200
201 % Hints: contents = cellstr(get(hObject,'String')) returns 1st_Chart contents as cell array
202 % contents(get(hObject,'Value')) returns selected item from 1st_Chart
203
204 val=get(hObject,'Value');
205 str=get(hObject,'String');
206 switch str(val)
207     case 'peaks'
208         axes(handles.pltChart);
209         peaks;
210     case 'membrane'
211         axes(handles.pltChart);
212         membrane;
213     case 'logo'
214         axes(handles.pltChart);
215         logo;
216 end

```

Now if as always we save the changes and run the GUI we get:



And if you want to fill dynamically the pop-up menu, here is what you need to use for example when the GUI loads:

```
mon_GUI.m x
47 % --- Executes just before monGUI is made visible.
48 function monGUI_OpeningFcn(hObject, eventdata, handles, varargin)
49 % This function has no output args, see OutputFcn.
50
51 % Choose default command line output for monGUI
52 handles.output = hObject;
53
54 % Update handles structure
55 guidata(hObject, handles);
56
57 set(handles.lbl_Date,'String',cellstr(datestr(date, 'yyyy-mm-dd')));
58
59 axes(handles.ltx_Equation);
60 axis off;
61 text(0,0.5,'I can display $\pi$', 'interpreter','latex',...
62 'horiz','left','vert','middle')
63
64 axes(handles.imgLogo);
65 imshow('matlab.jpg');
66 axes(handles.pltChart);
67 peaks;
68 grid off;
69 string_list{1}='peaks';
70 string_list{2}='logo';
71 string_list{3}='membrane';
72 set(handles.lst_Chart,'String',string_list);
```

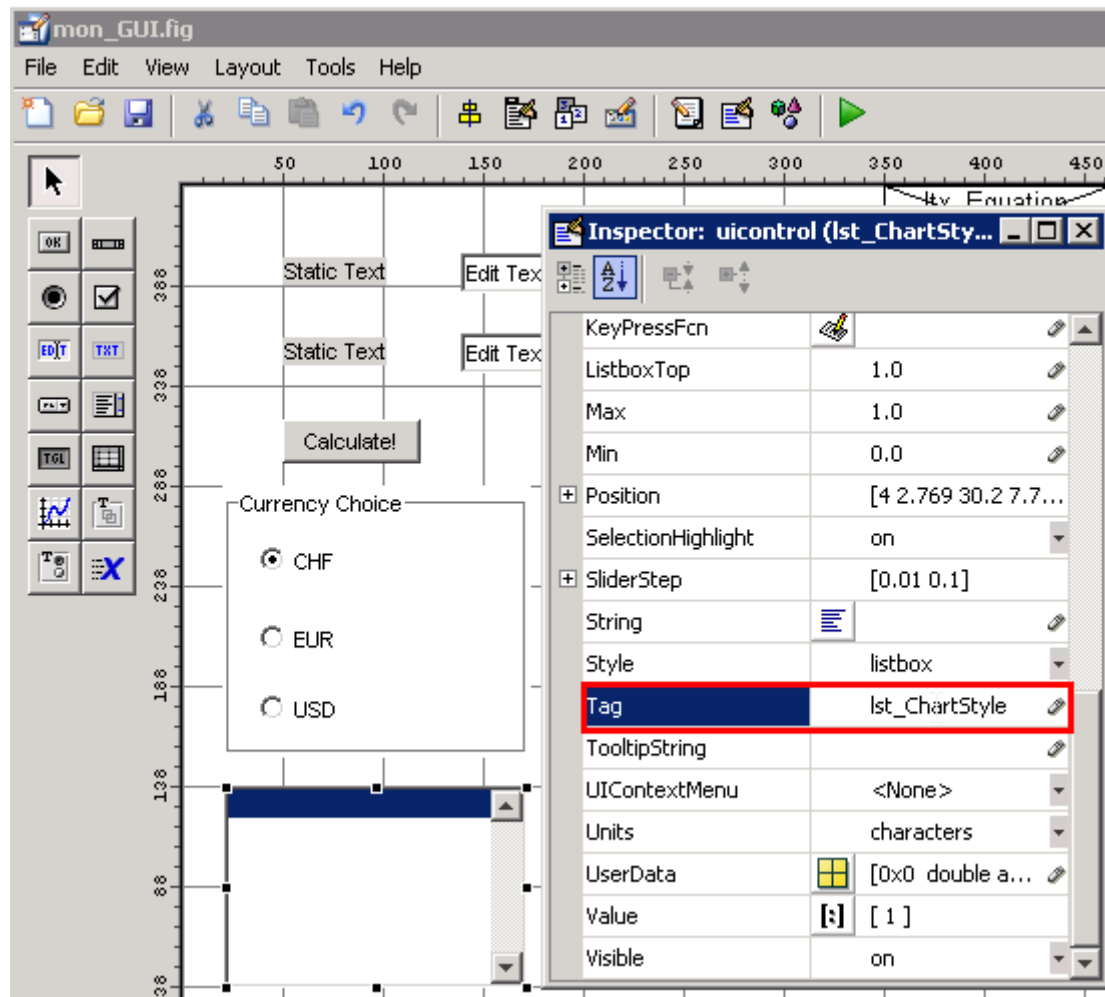
and for sure you can automate this with a loop! Or if you prefer:

```
mon_GUI.m* x
47 % --- Executes just before monGUI is made visible.
48 function monGUI_OpeningFcn(hObject, eventdata, handles, varargin)
49 % This function has no output args, see OutputFcn.
50
51 % Choose default command line output for monGUI
52 handles.output = hObject;
53
54 % Update handles structure
55 guidata(hObject, handles);
56
57 set(handles.lbl_Date,'String',cellstr(datestr(date, 'yyyy-mm-dd')));
58
59 axes(handles.ltx_Equation);
60 axis off;
61 text(0,0.5,'I can display $\pi$', 'interpreter', 'latex', ...
62 'horiz', 'left', 'vert', 'middle')
63
64 axes(handles.imgLogo);
65 imshow('matlab.jpg');
66 axes(handles.pltChart);
67 peaks;
68 grid off;
69 set(handles.lst_Chart, 'String', {'peaks', 'logo', 'membrane'});
70
```

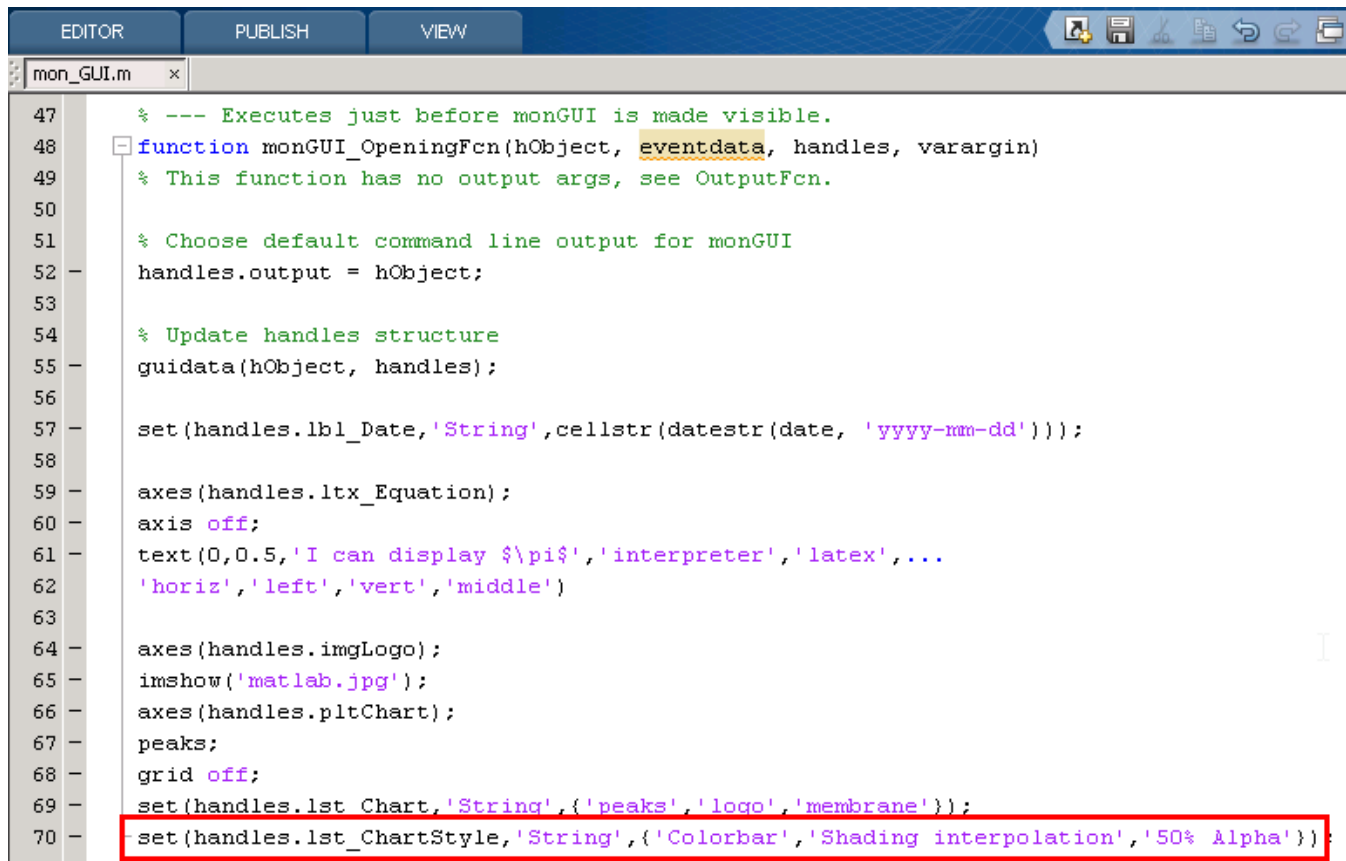
So when we run the GUI we also get the same thing as before!

Manage multiple selection List box

For the multiple selection List box  we will directly make a dynamic one (we already saw before how to manage static stuff). First create such a component following exactly the same steps as we did for the pop-menu (simple list box):



Once this is done we fill it dynamically on the GUI open event callback:

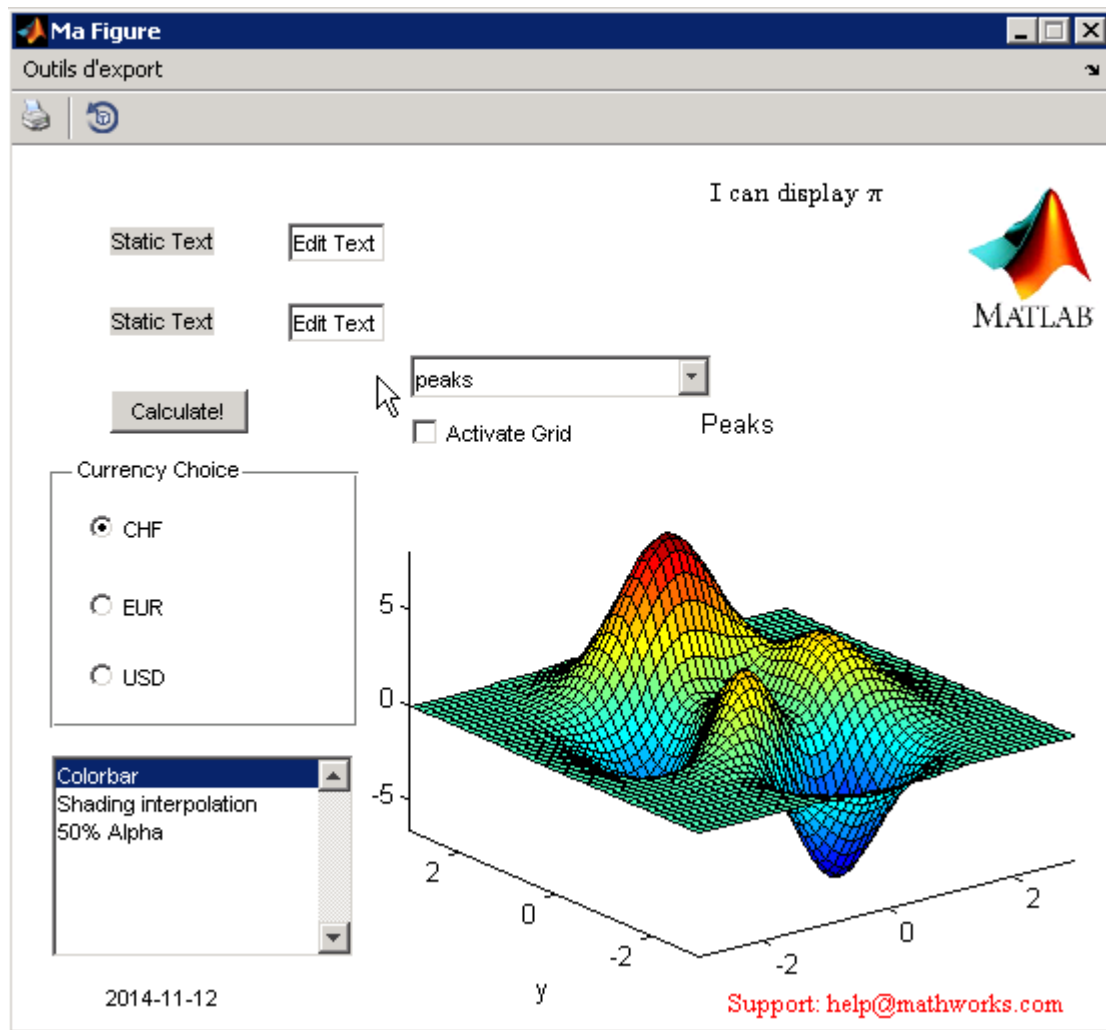


```

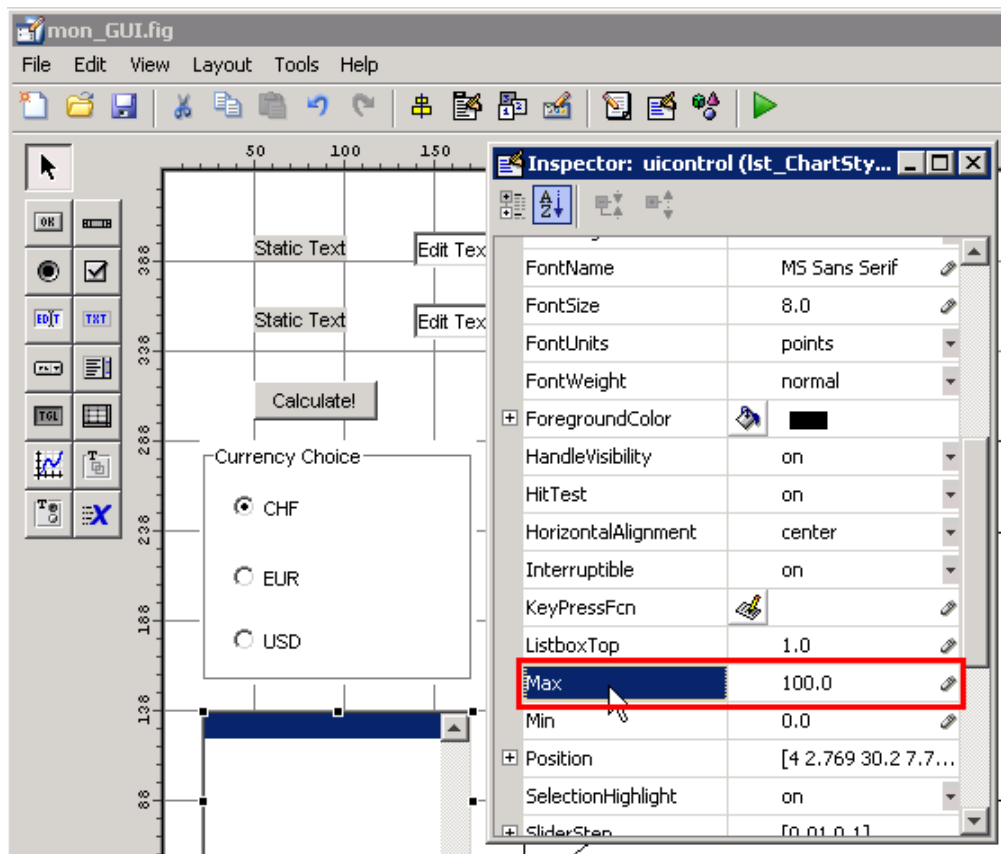
47 % --- Executes just before monGUI is made visible.
48 function monGUI_OpeningFcn(hObject, eventdata, handles, varargin)
49 % This function has no output args, see OutputFcn.
50
51 % Choose default command line output for monGUI
52 handles.output = hObject;
53
54 % Update handles structure
55 guidata(hObject, handles);
56
57 set(handles.lbl_Date,'String',cellstr(datestr(date, 'yyyy-mm-dd')));
58
59 axes(handles.ltx_Equation);
60 axis off;
61 text(0,0.5,'I can display $\pi$', 'interpreter', 'latex', ...
62 'horiz', 'left', 'vert', 'middle')
63
64 axes(handles.imgLogo);
65 imshow('matlab.jpg');
66 axes(handles.pltChart);
67 peaks;
68 grid off;
69 set(handles.lst_Chart,'String',{'peaks','logo','membrane'});
70 set(handles.lst_ChartStyle,'String',{'Colorbar','Shading interpolation','50% Alpha'});

```

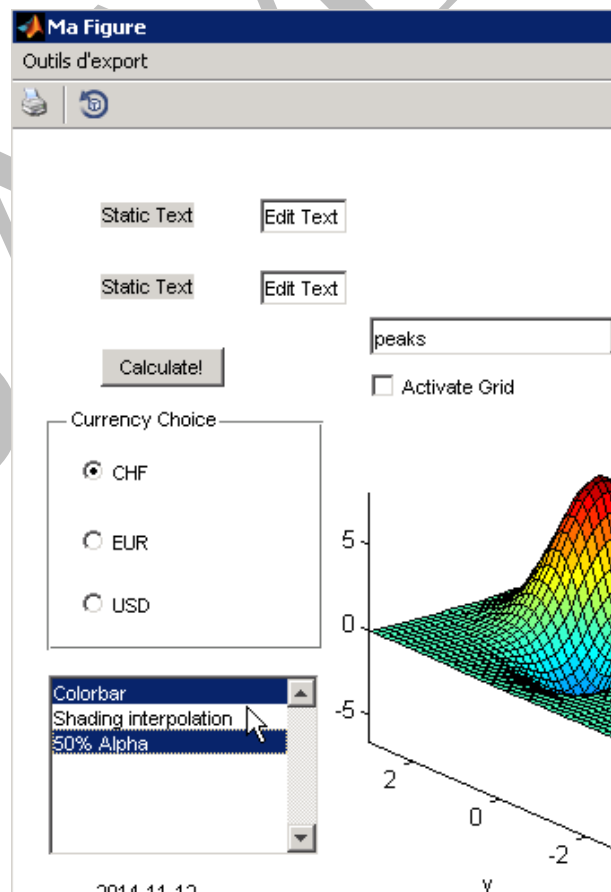
If we already run the GUI we get:



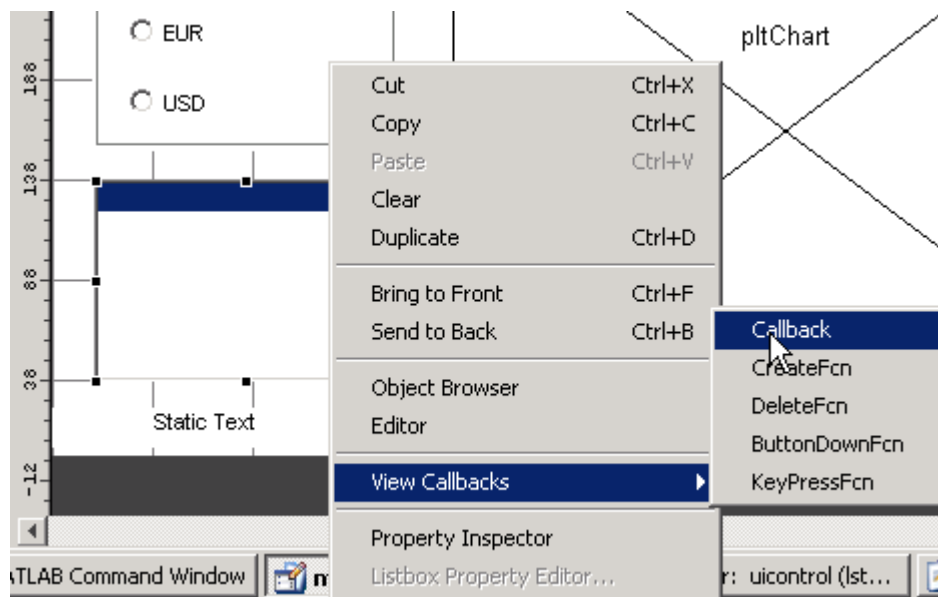
To make multiple selection possible you have to change the **Min** and **Max** properties. The difference corresponds to the number of possible selections. For example, even if we have only three choices we can put:



So:



Seems to work well! We continue now to define a callback on this list box:

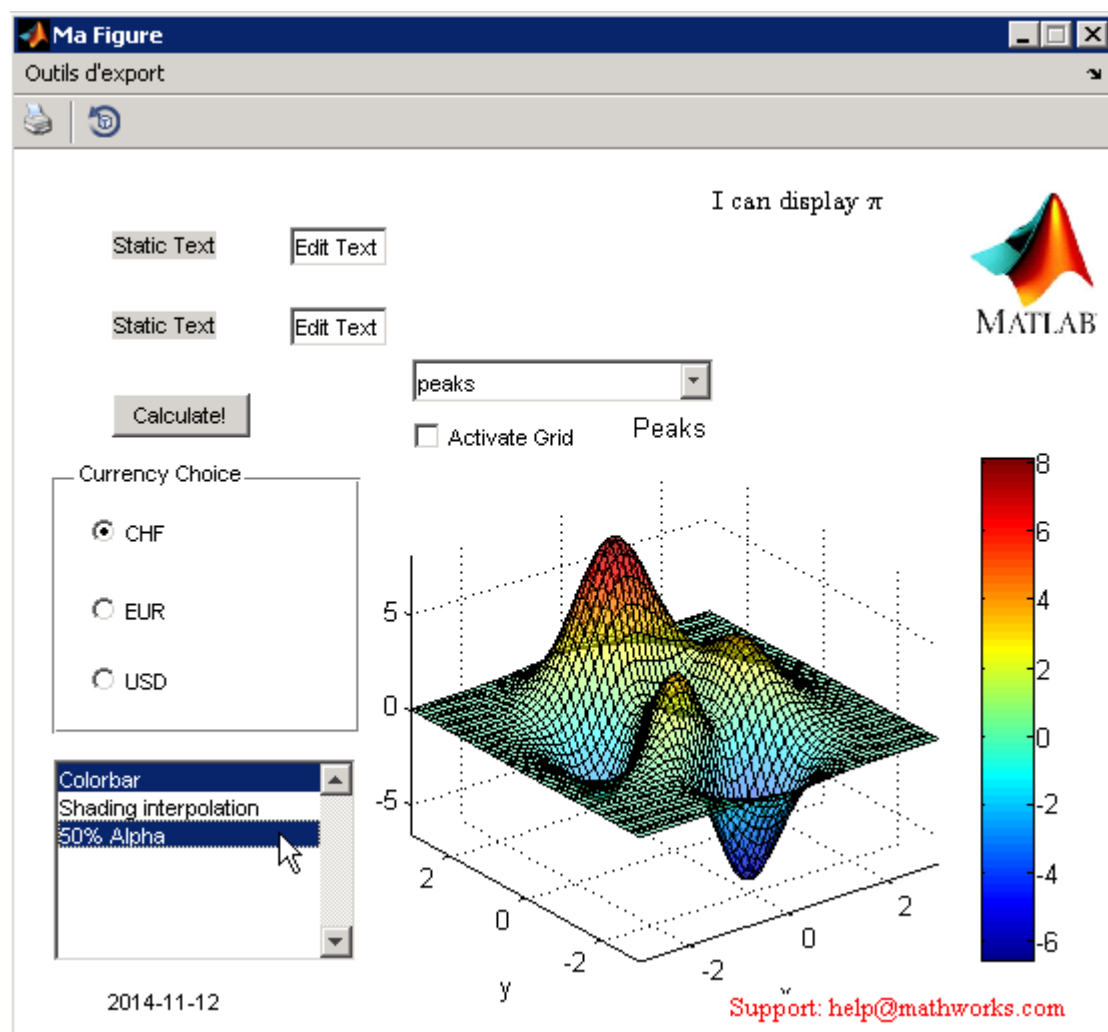


And now we write the callback for the List Box:

```

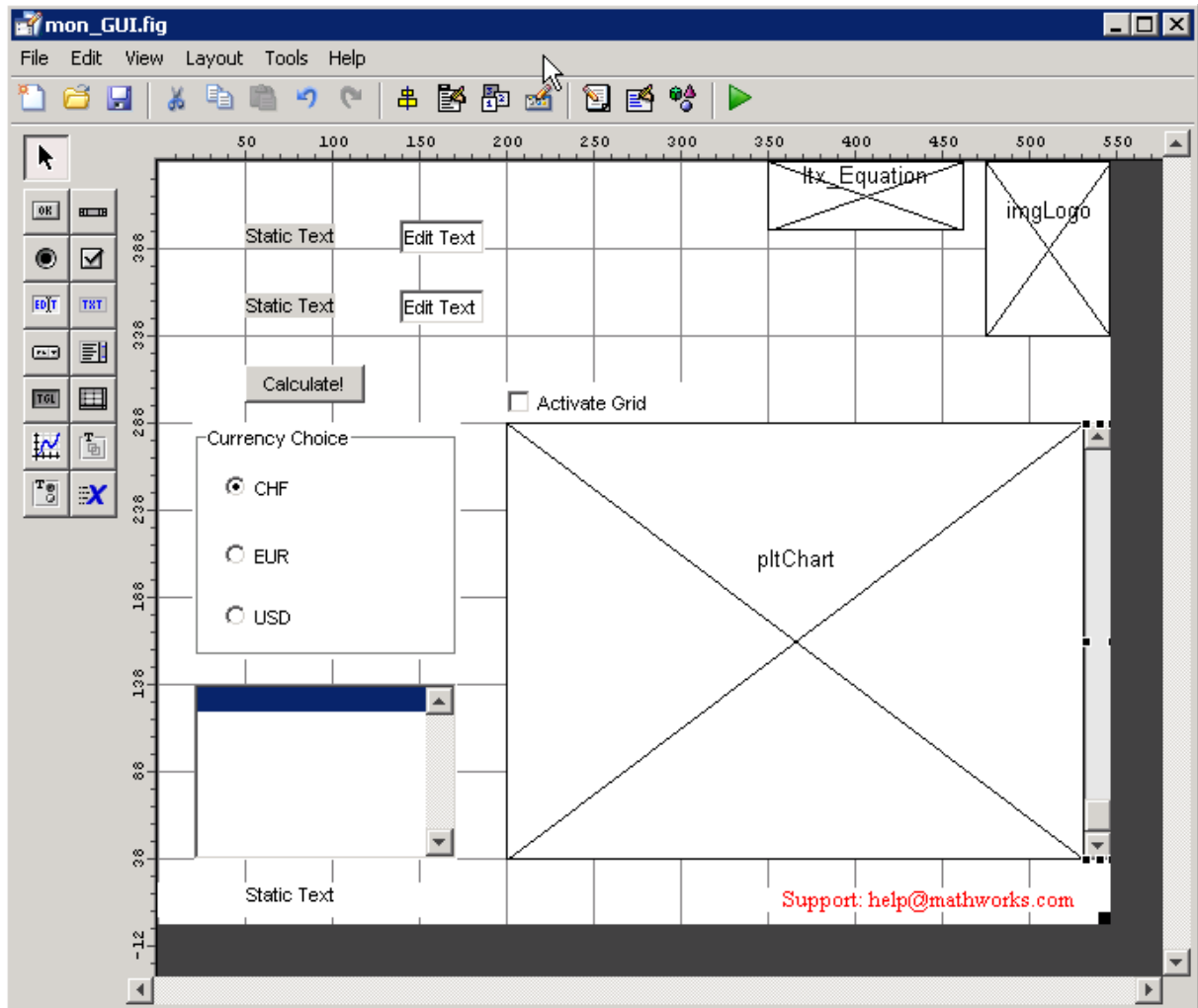
227 % --- Executes on selection change in lst_ChartStyle.
228 function lst_ChartStyle_Callback(hObject, eventdata, handles)
229 % hObject      handle to lst_ChartStyle (see GCBO)
230 % eventdata    reserved - to be defined in a future version of MATLAB
231 % handles      structure with handles and user data (see GUIDATA)
232
233 % Hints: contents = cellstr(get(hObject,'String')) returns lst_ChartStyle contents as cell array
234 %          contents{get(hObject,'Value')} returns selected item from lst_ChartStyle
235 str = get(hObject,{'String','Value'});
236 str = str(1)(str(2));
237 peaks;
238 if find(ismember(str,'Colorbar'))
239     axes(handles.pltChart);
240     colorbar;
241 end
242 if find(ismember(str,'Shading interpolation'))
243     axes(handles.pltChart);
244     shading interp;
245 end
246 if find(ismember(str,'50% Alpha'))
247     axes(handles.pltChart);
248     alpha(0.5);
249 end
  
```

This works well as we can see it:

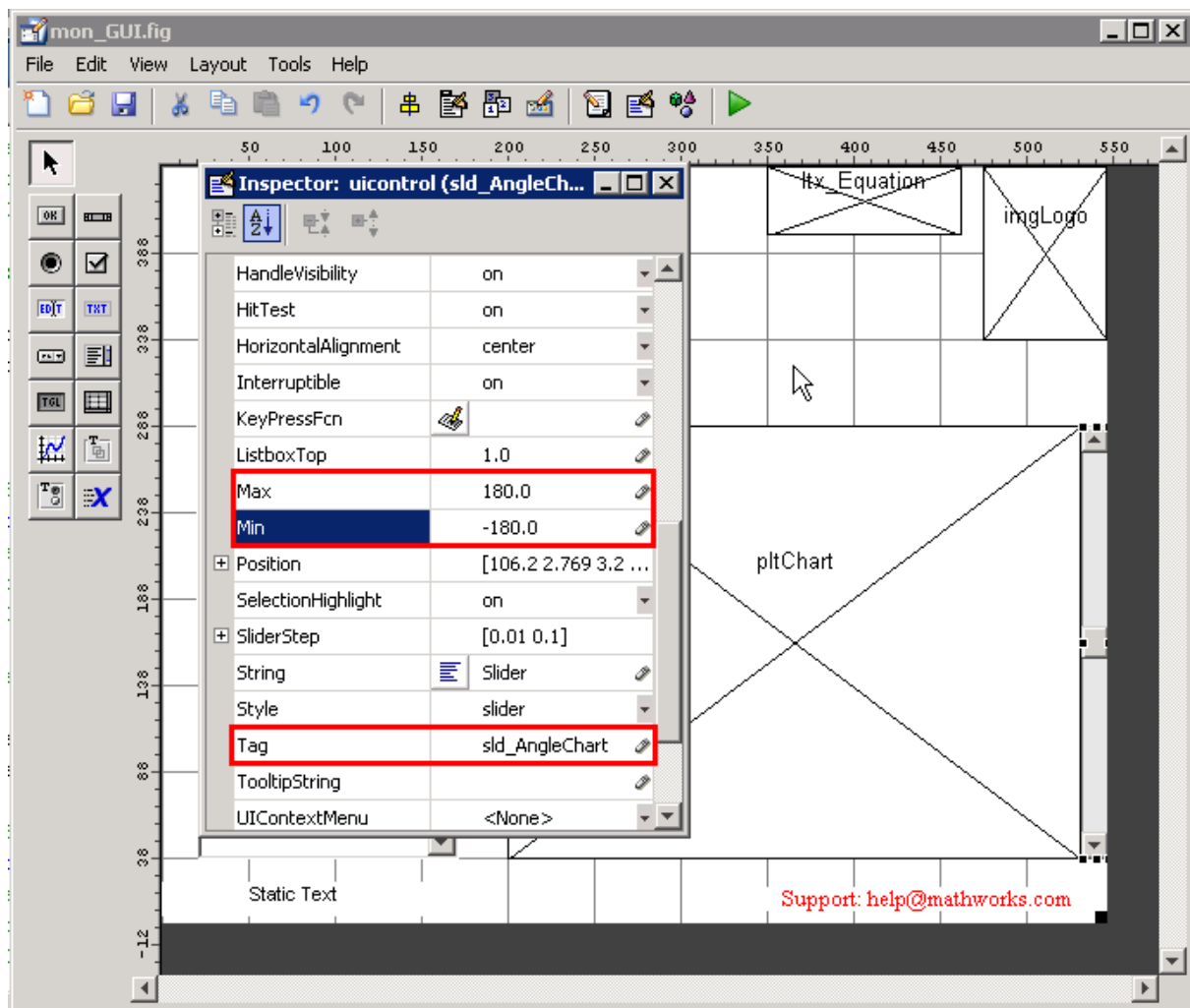


Manage sliders

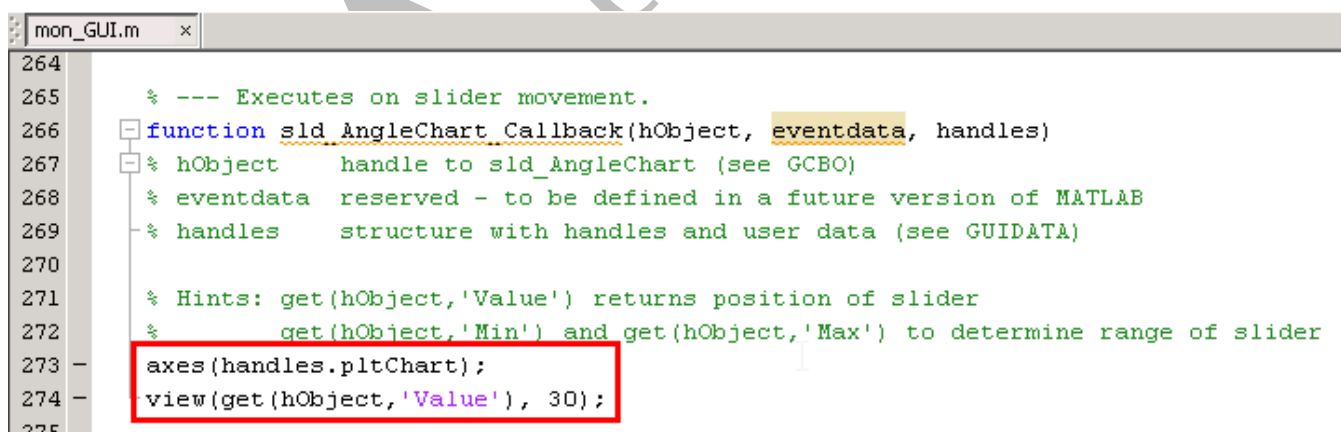
Sliders  are very useful for all sort of simulations or chart manipulation. Let us have a look of how to use them. Add one right from the chart as following:



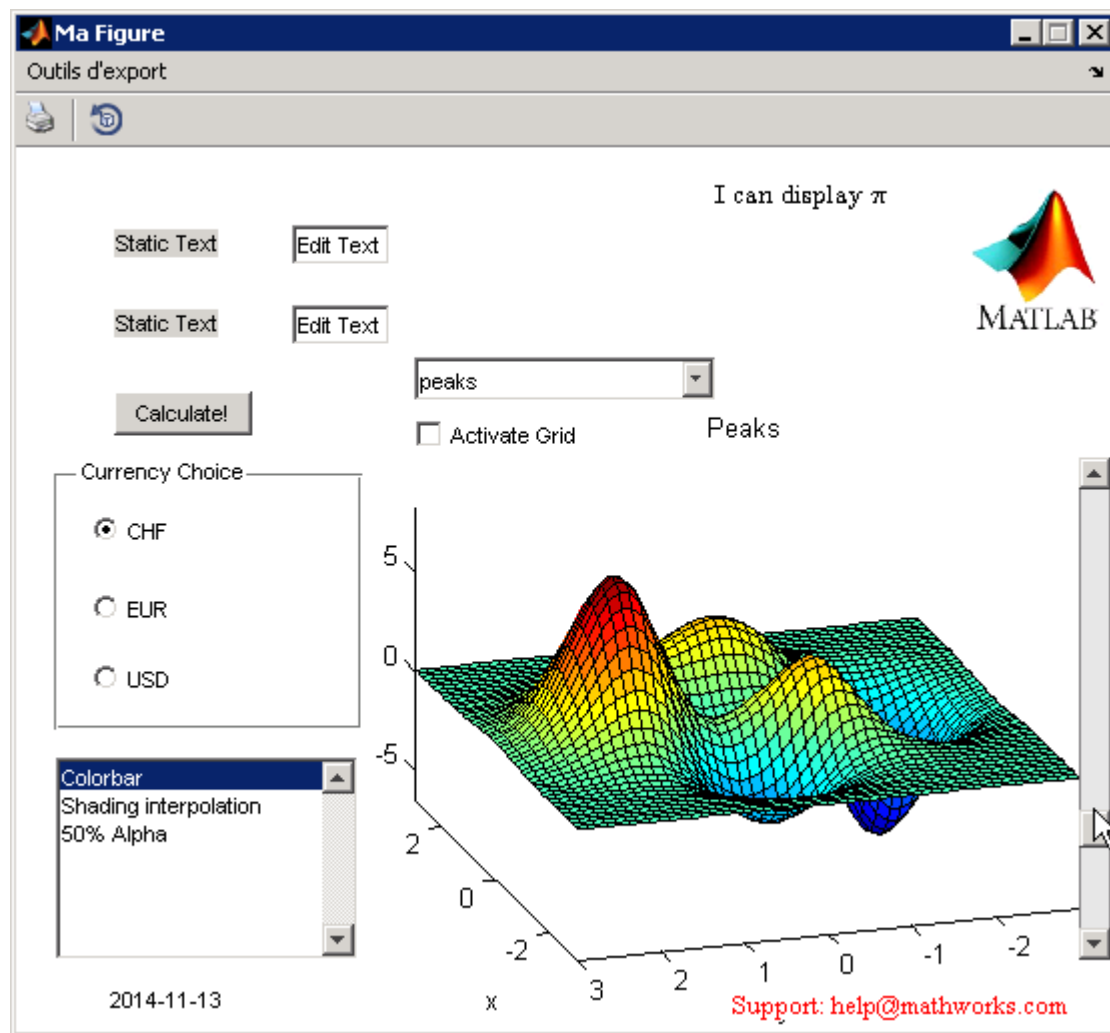
Open the property inspector and change the **Tag**, **Max** and **Min** properties:



Now we open the callback of the slider to write:

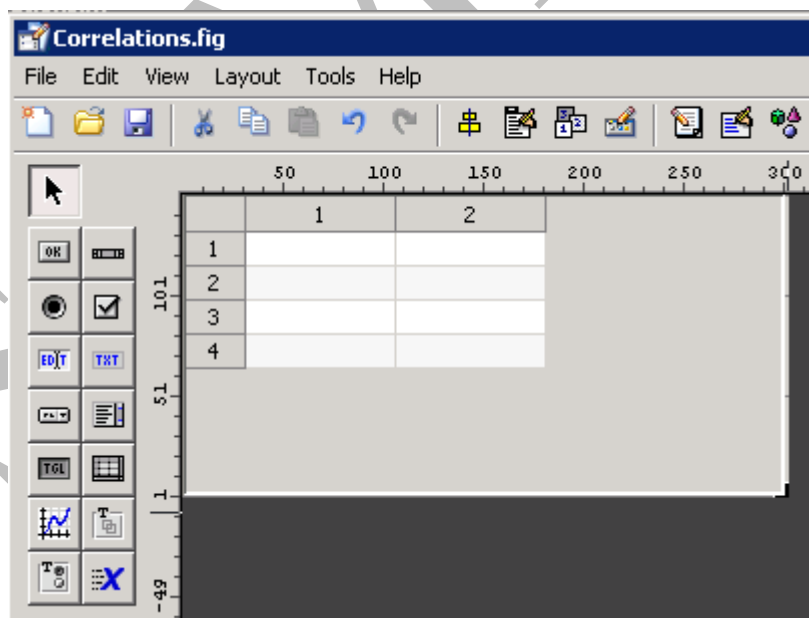
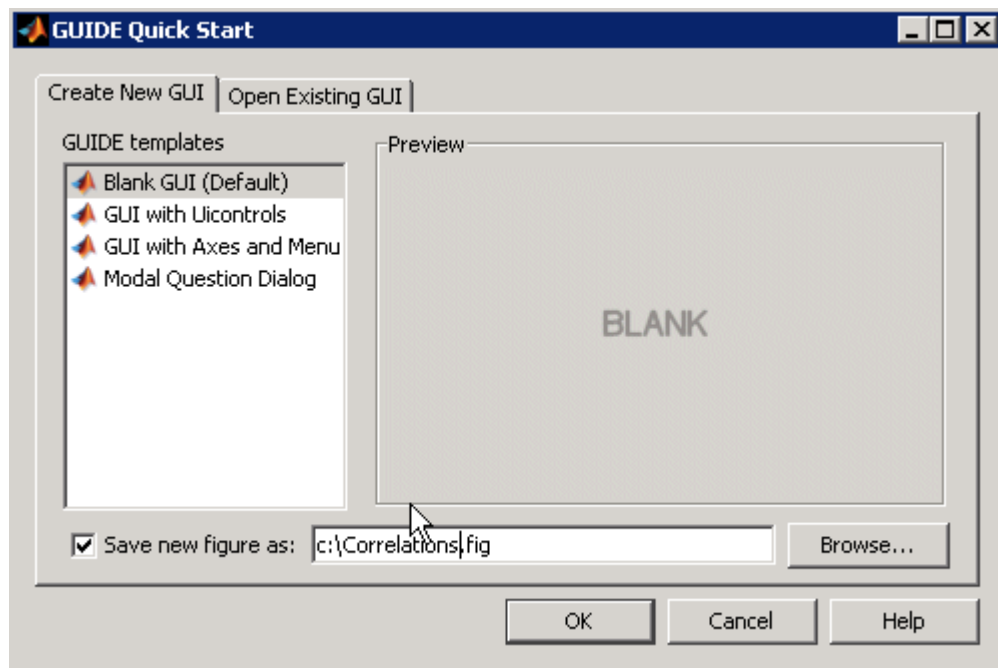


And if we run the GUI you can check that the slider works well:



Manage tables

We can do a lot of things with MATLAB GUI tables: tables of values, tables of values with check boxes, tables of listboxes, etc. Here we will focus only on the basics.



MATLAB Command Window

```
>> c=[1,0,0;0,1,0;0,0,1]
```

c =

1	0	0
0	1	0
0	0	1

```
>> |
```

Correlations.fig

File Edit View Layout Tools Help

Inspector: uitable

ColumnFormat	[0x0 double array]
ColumnName	numbered
ColumnWidth	[1x1 char array]
CreateFcn	
Data	[4x2 cell array]
DeleteFcn	
Enable	on

Table Property Editor

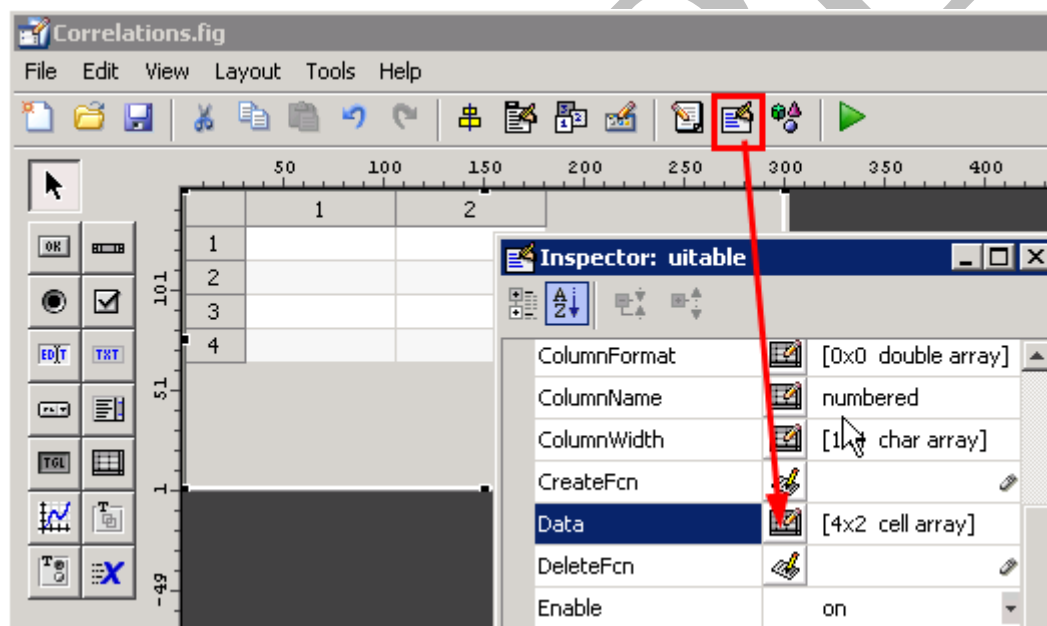
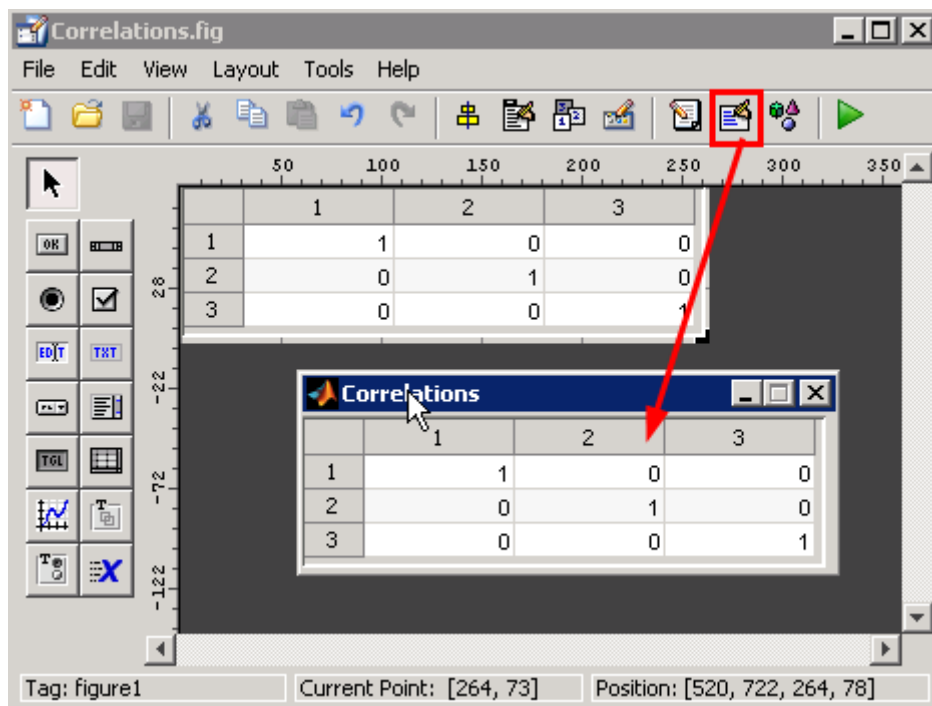
Columns
Rows
Data
Colors

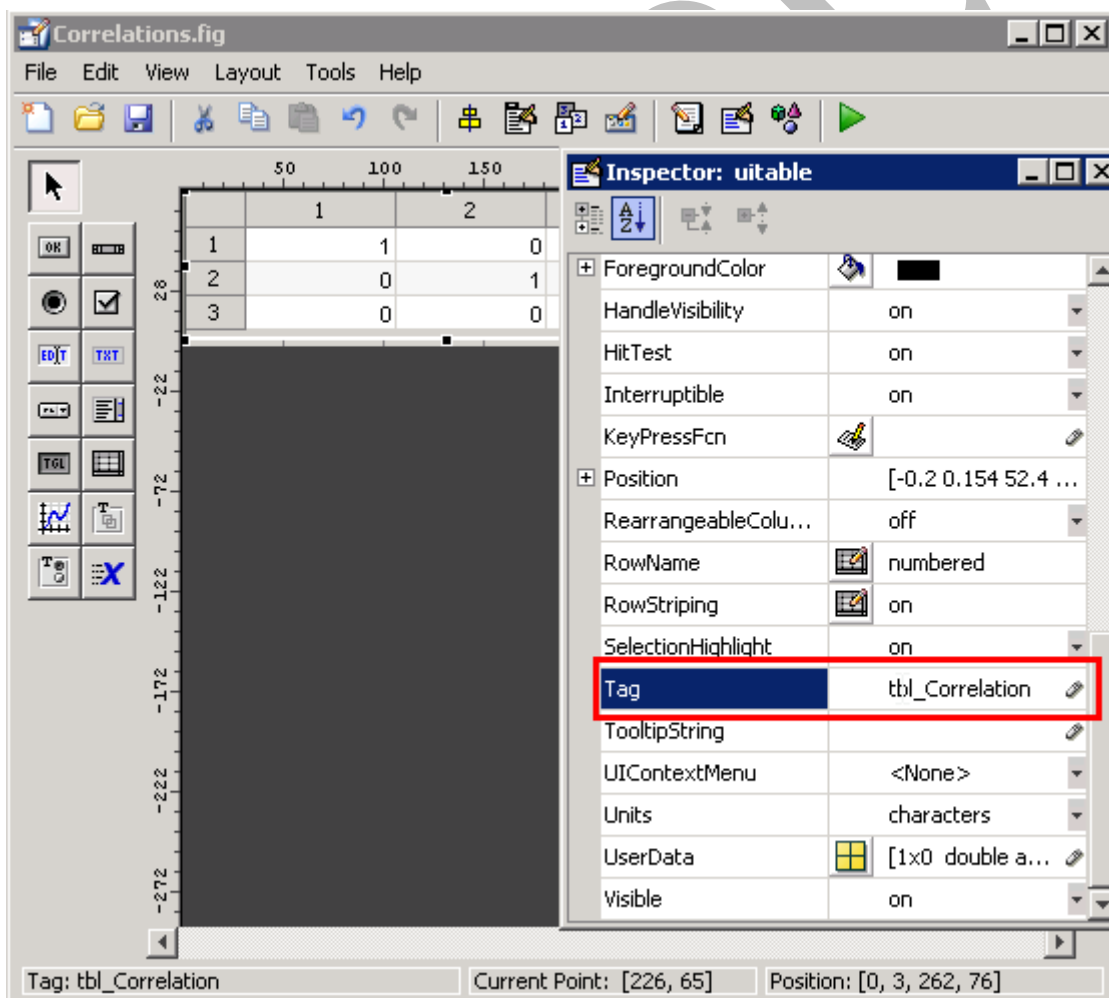
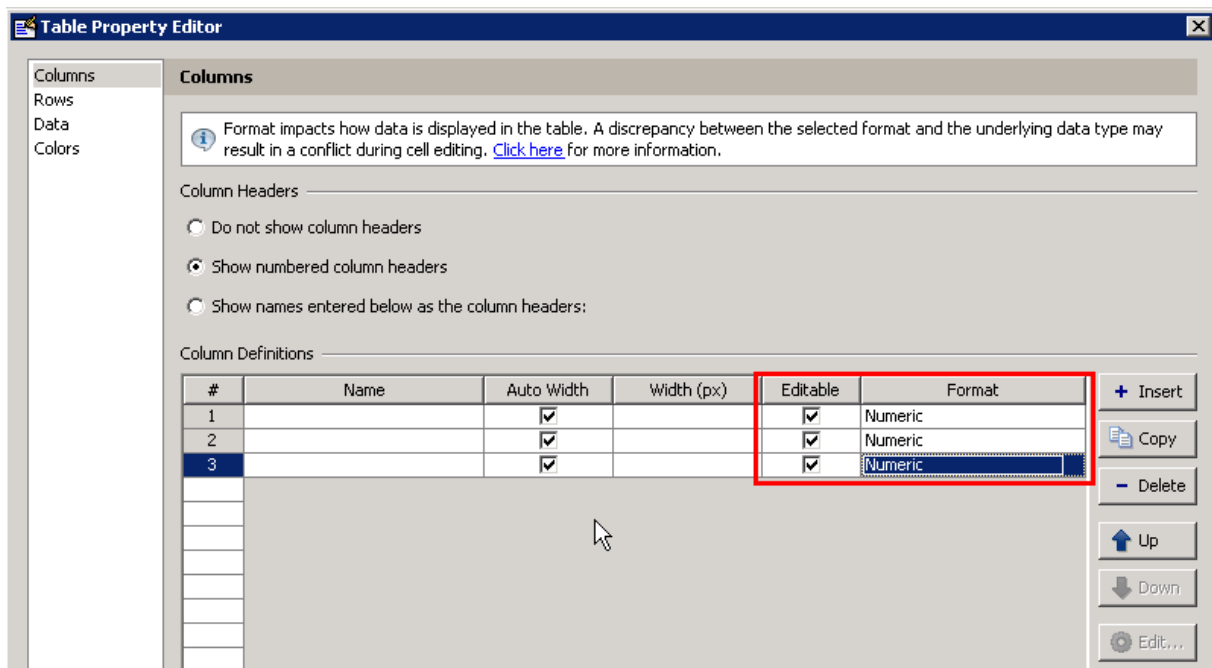
Data

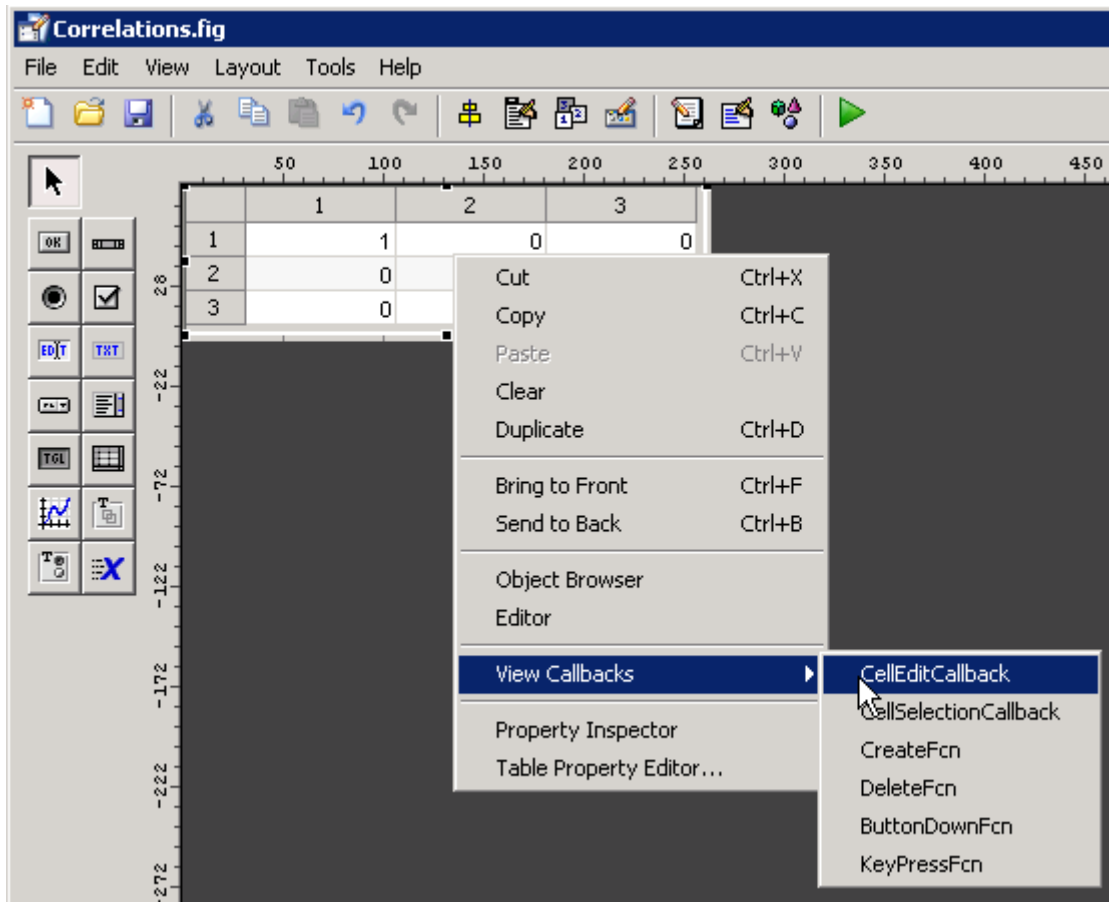
Loading data from a workspace variable creates a static copy of the data. Change the table.

☐ Do not set data value and leave the table empty
☐ Keep the current data value
☒ Change data value to the selected workspace variable below

Name	Value	Size	Class
c	[1,0,0;0,1,0;0,0,1]	3x3	double



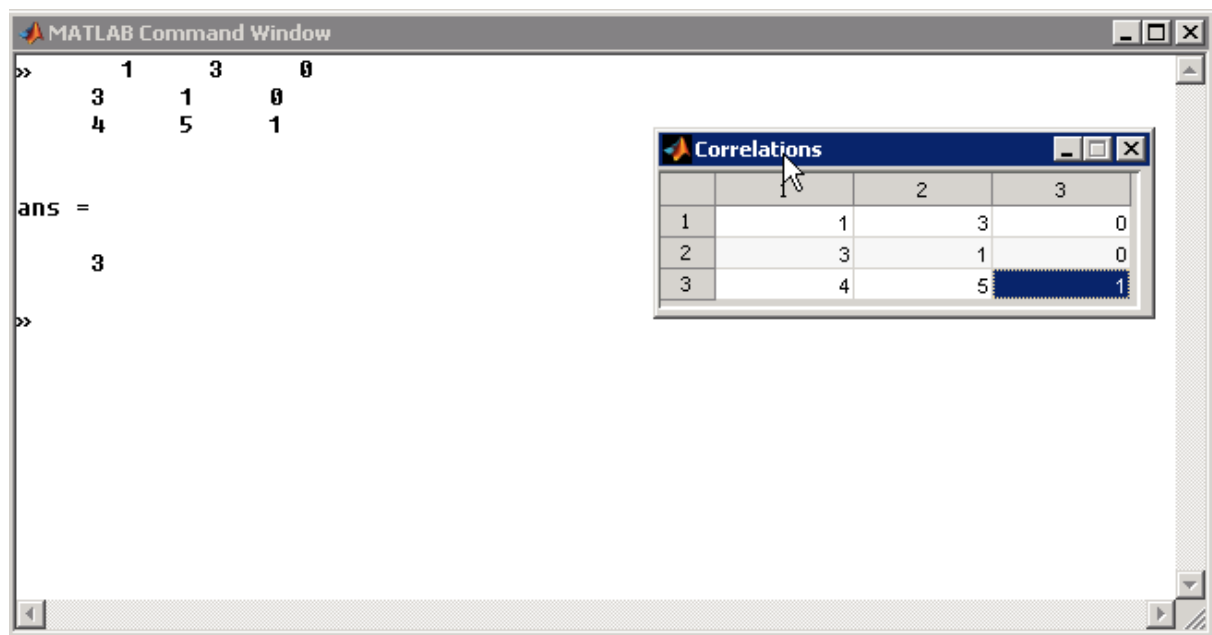




```

76
77 % --- Executes when entered data in editable cell(s) in tbl Correlation.
78 function tbl_Correlation_CellEditCallback(hObject, eventdata, handles)
79 % hObject    handle to tbl_Correlation (see GCBO)
80 % eventdata  structure with the following fields (see UITABLE)
81 %   Indices: row and column indices of the cell(s) edited
82 %   PreviousData: previous data for the cell(s) edited
83 %   EditData: string(s) entered by the user
84 %   NewData: EditData or its converted form set on the Data property. Empty if Data was not changed
85 %   Error: error string when failed to convert EditData to appropriate value for Data
86 % handles    structure with handles and user data (see GUIDATA)
87 data=get(hObject,'data');
88 disp(data); %display the whole matrix in the Command Window
89 data(1,2) %show only the first line, second column component
90

```

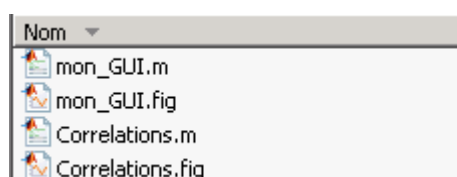


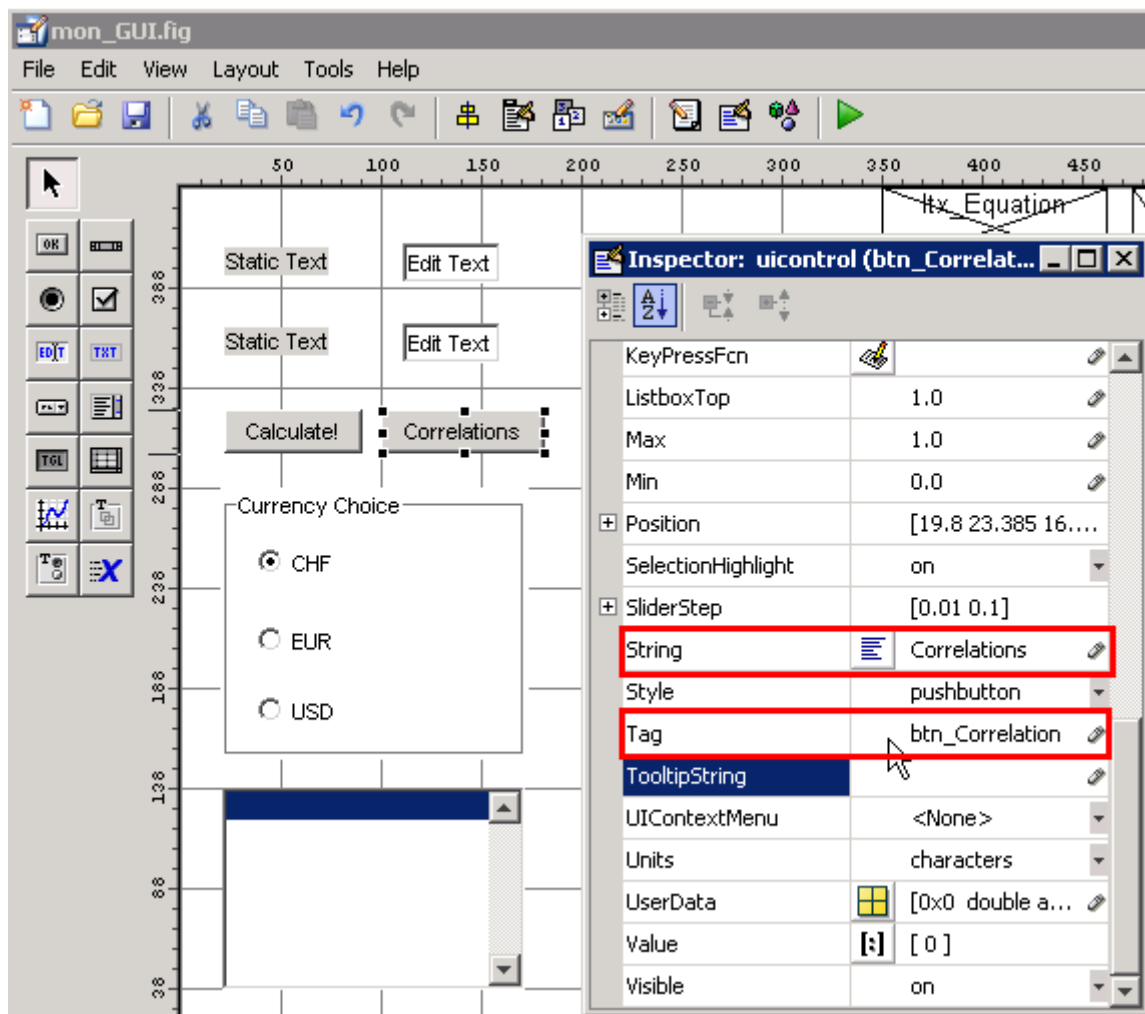
```
Correlations.m
47 % --- Executes just before Correlations is made visible.
48 function Correlations_OpeningFcn(hObject, eventdata, handles, varargin)
49 % This function has no output args, see OutputFcn.
50 % hObject    handle to figure
51 % eventdata  reserved - to be defined in a future version of MATLAB
52 % handles    structure with handles and user data (see GUIDATA)
53 % varargin   command line arguments to Correlations (see VARARGIN)
54
55 % Choose default command line output for Correlations
56 handles.output = hObject;
57
58 % Update handles structure
59 guidata(hObject, handles);
60 set(handles.tbl_Correlation, 'Data', [3,0,0;0,3,0;0,0,3])
61
```

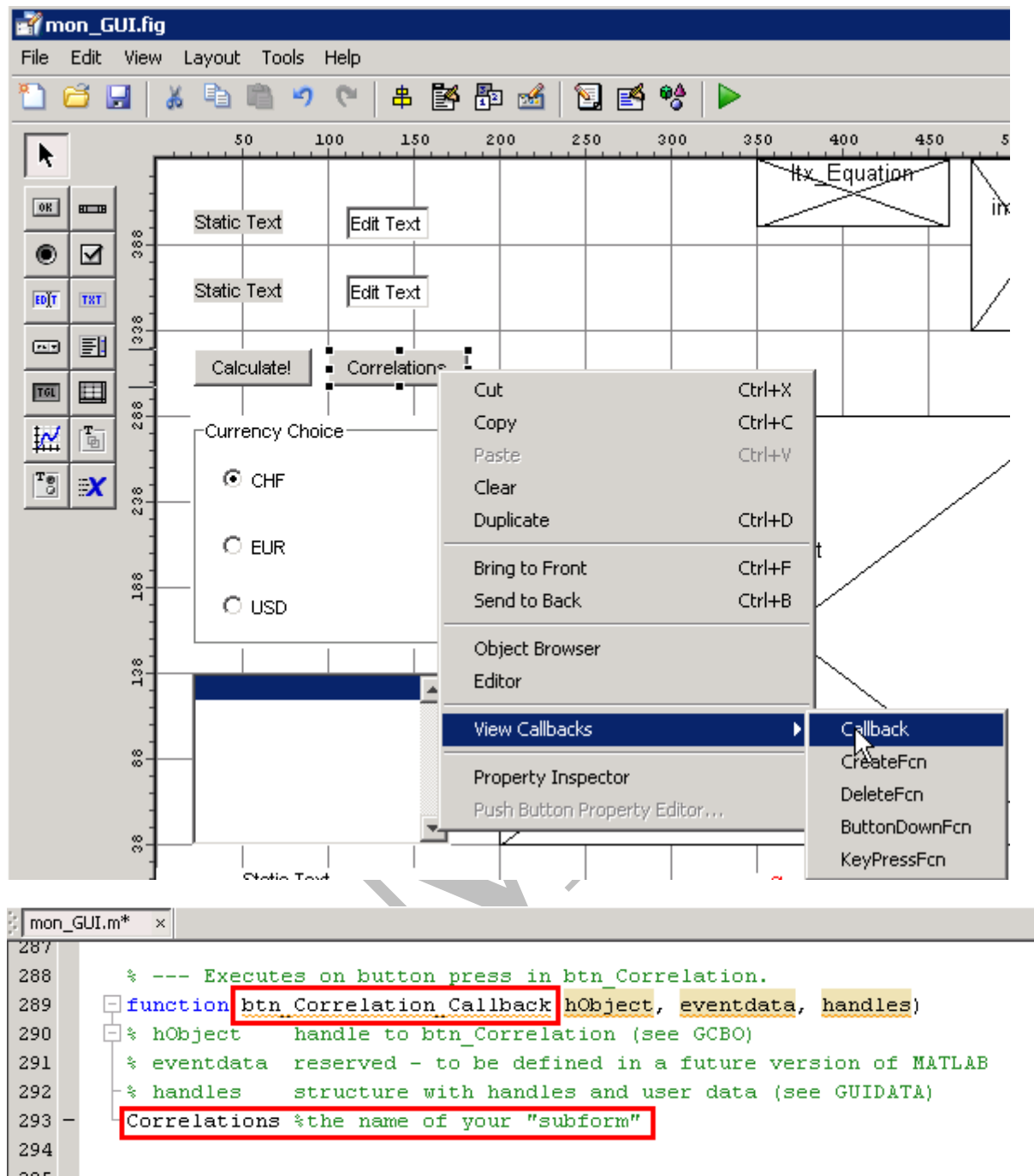
Correlations

	1	2	3
1	3	0	0
2	0	3	0
3	0	0	3

Userform calling another userform





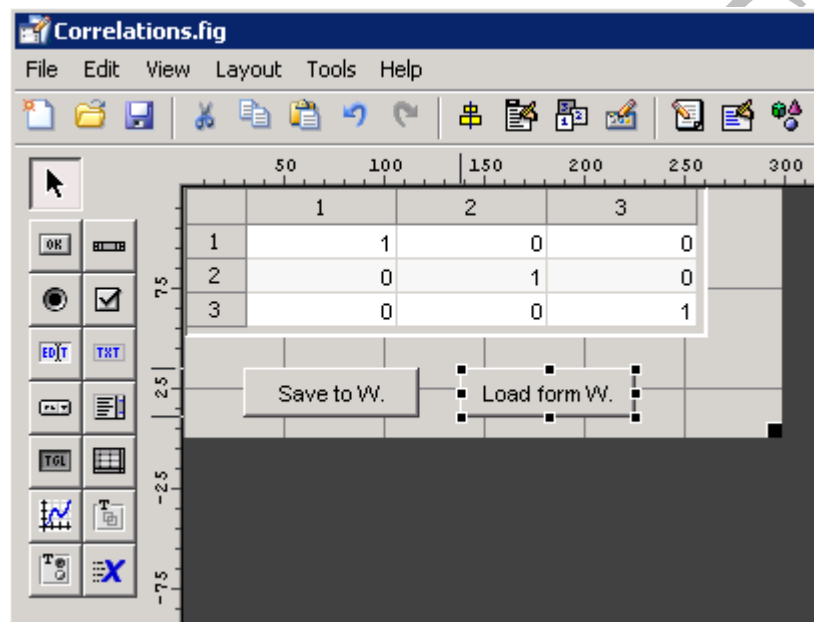


Write/Get GUI variables in MATLAB workspace

There exist methods to pass variables between GUI² without writing the variables in the workspace but I prefer this last solution for multiple reasons:

1. The user can change the variable on the fly from the *Command Window* if he prefers
2. The user can save all variables in a MAT-file to reload them later or for other use
3. The script is less horrible than the one

Suppose we want to be able to save or load our correlation matrix to the Workspace. First create two buttons *btn_SaveW* and *btn_LoadW* as following:



² To pass variables between Callbacks in the same GUI just add *global myvariable_name*; at the beginning of each callback where you need to access your variable and add at the end of the callback that fills you variables the command *guidata(hObject,handles)* to maintain the variable in the memory (see the example in the text).

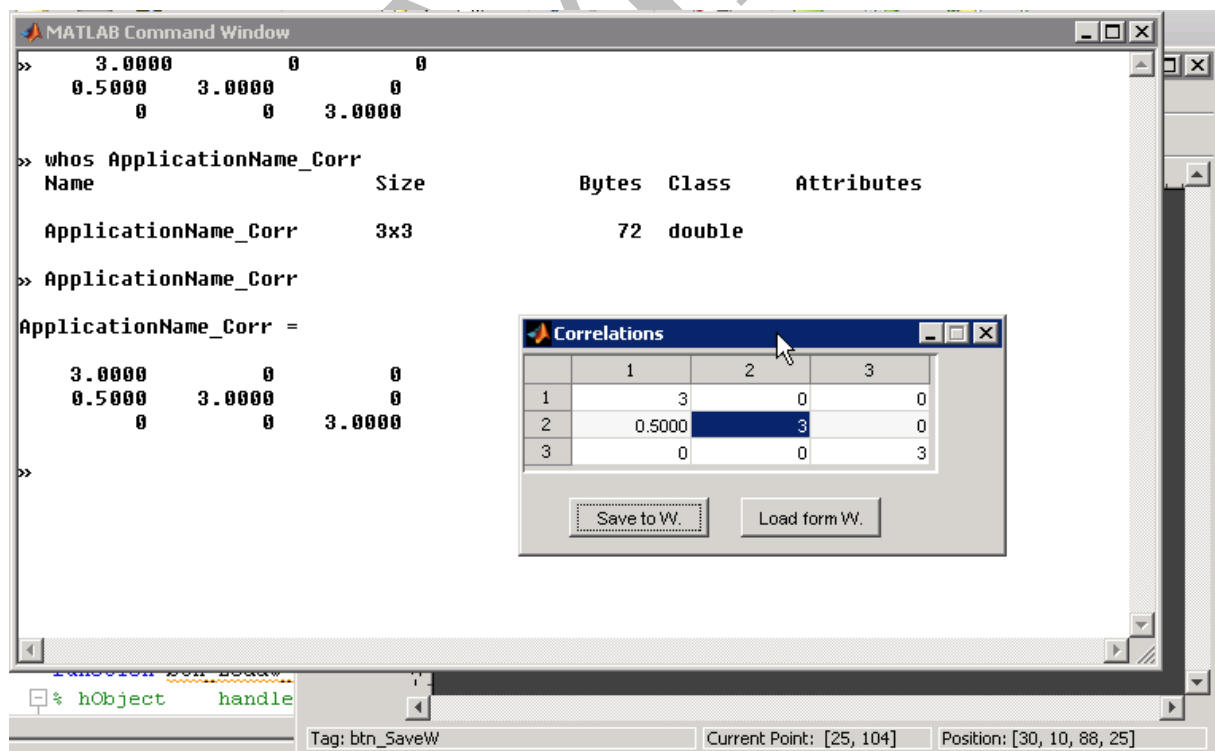
After change the all **Callbacks** as following using global variable and especially use the functions **assignin()** and **evalin()** to write and read in the workspace:

```

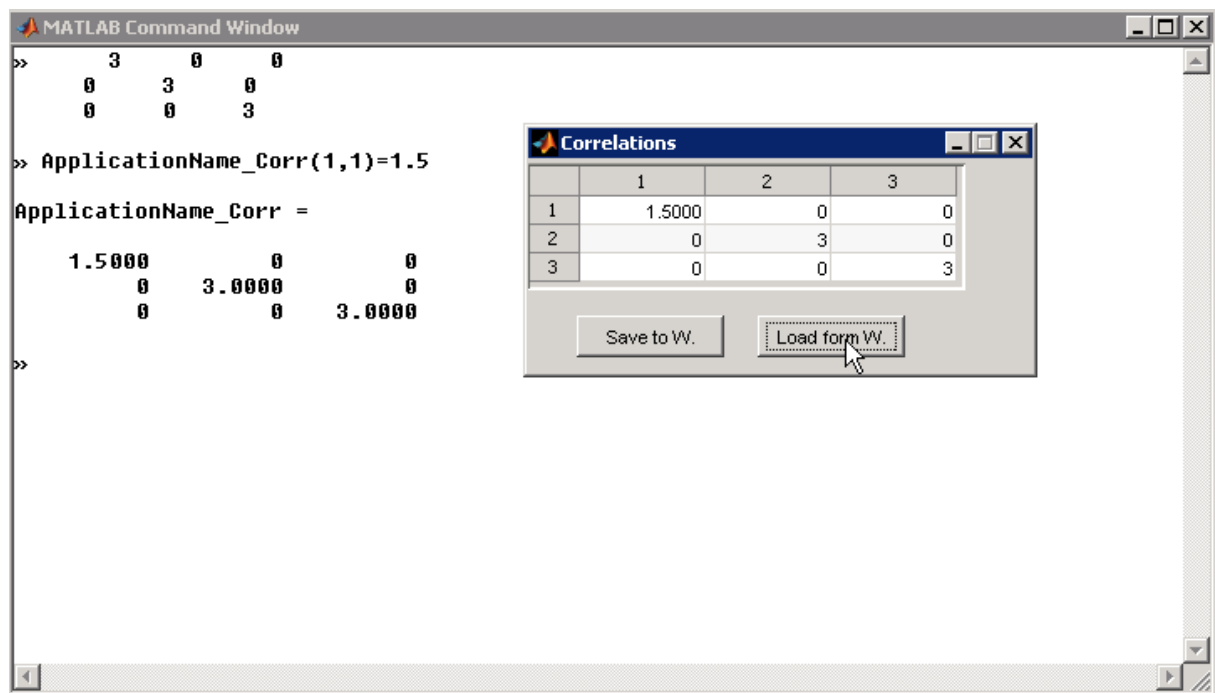
Correlations.m x
76
77 % --- Executes when entered data in editable cell(s) in tbl Correlation.
78 function tbl_Correlation_CellEditCallback(hObject, eventdata, handles)
79 % hObject    handle to tbl_Correlation (see GCBO) %...%
80
81 global data;
82 data=get(hObject,'data');
83 guidata(hObject,handles);
84
85 % --- Executes on button press in btn_SaveW.
86 function btn_SaveW_Callback(hObject, eventdata, handles)
87 % hObject    handle to btn_SaveW (see GCBO) %...%
88
89 global data;
90 assignin('base','ApplicationName_Corr',data);
91 disp(data)
92
93 % --- Executes on button press in btn_LoadW.
94 function btn_LoadW_Callback(hObject, eventdata, handles)
95 % hObject    handle to btn_LoadW (see GCBO) %...%
96
97 set(handles.tbl_Correlation,'Data',evalin('base','ApplicationName_Corr'))
98
99

```

When you click on the button *Save to W.* you can see and control in the **Command Window** that everything is working:



And also you can change the variable in the workspace and after click on the button *Load from W.* to see that everything works well:

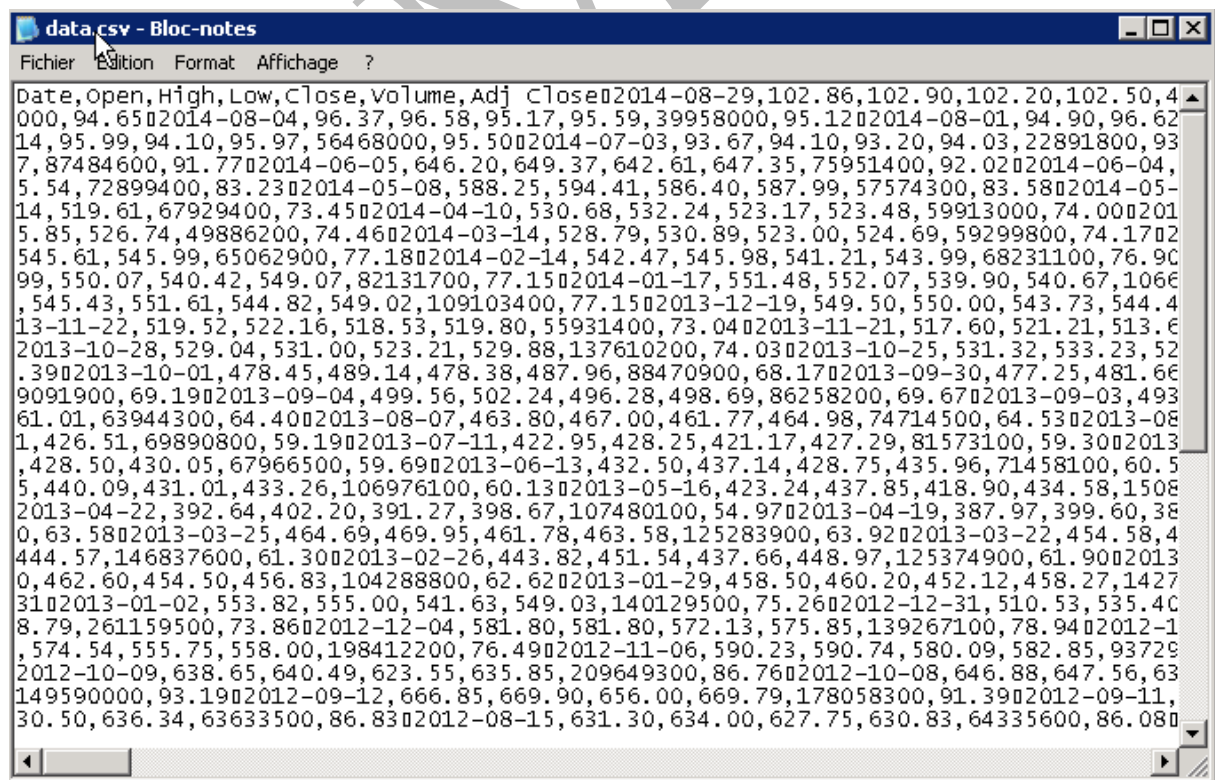


Create userforms using only script

Important practical cases

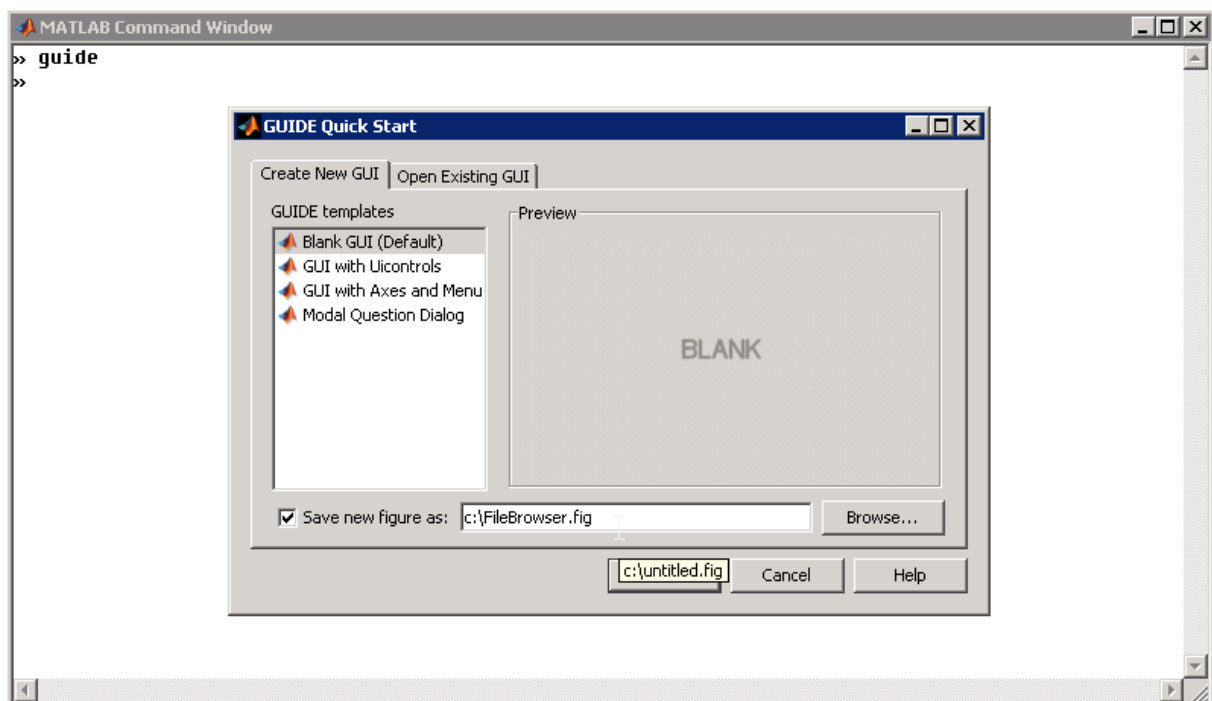
Create a GUI that opens a file

The idea will be to see how it looks like to load such a file:

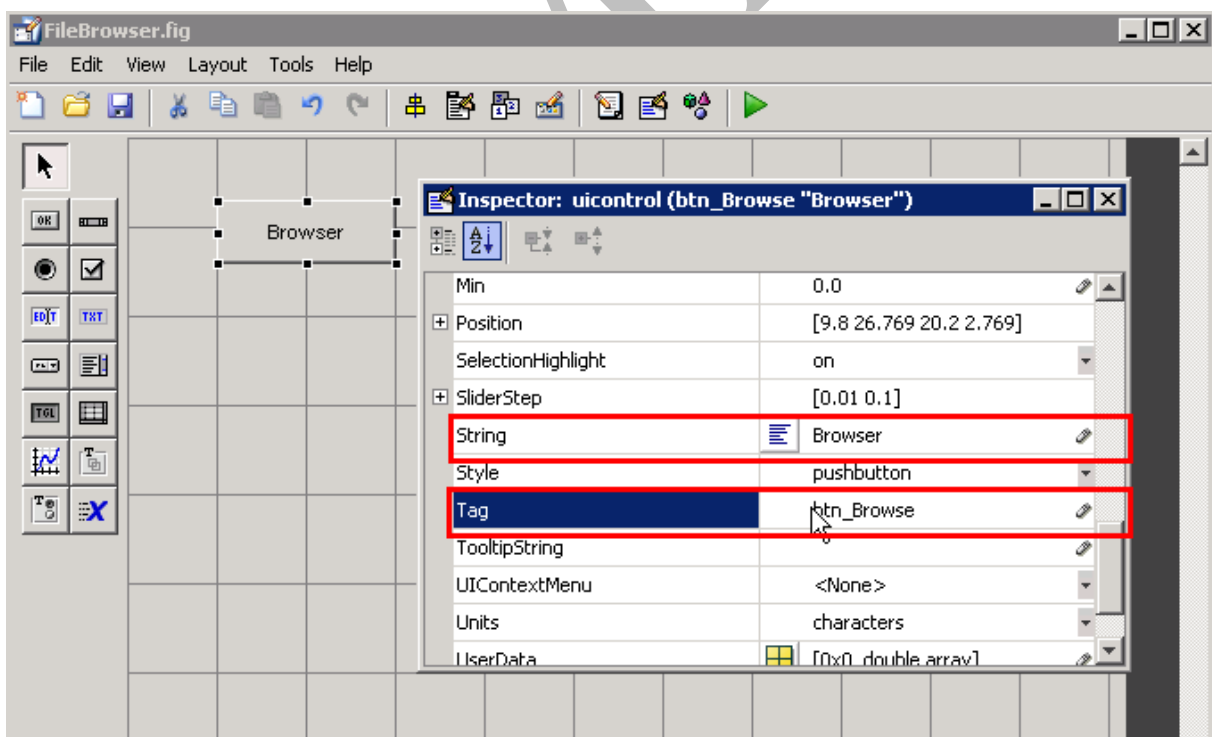


in a MATLAB GUI **Static text** area.

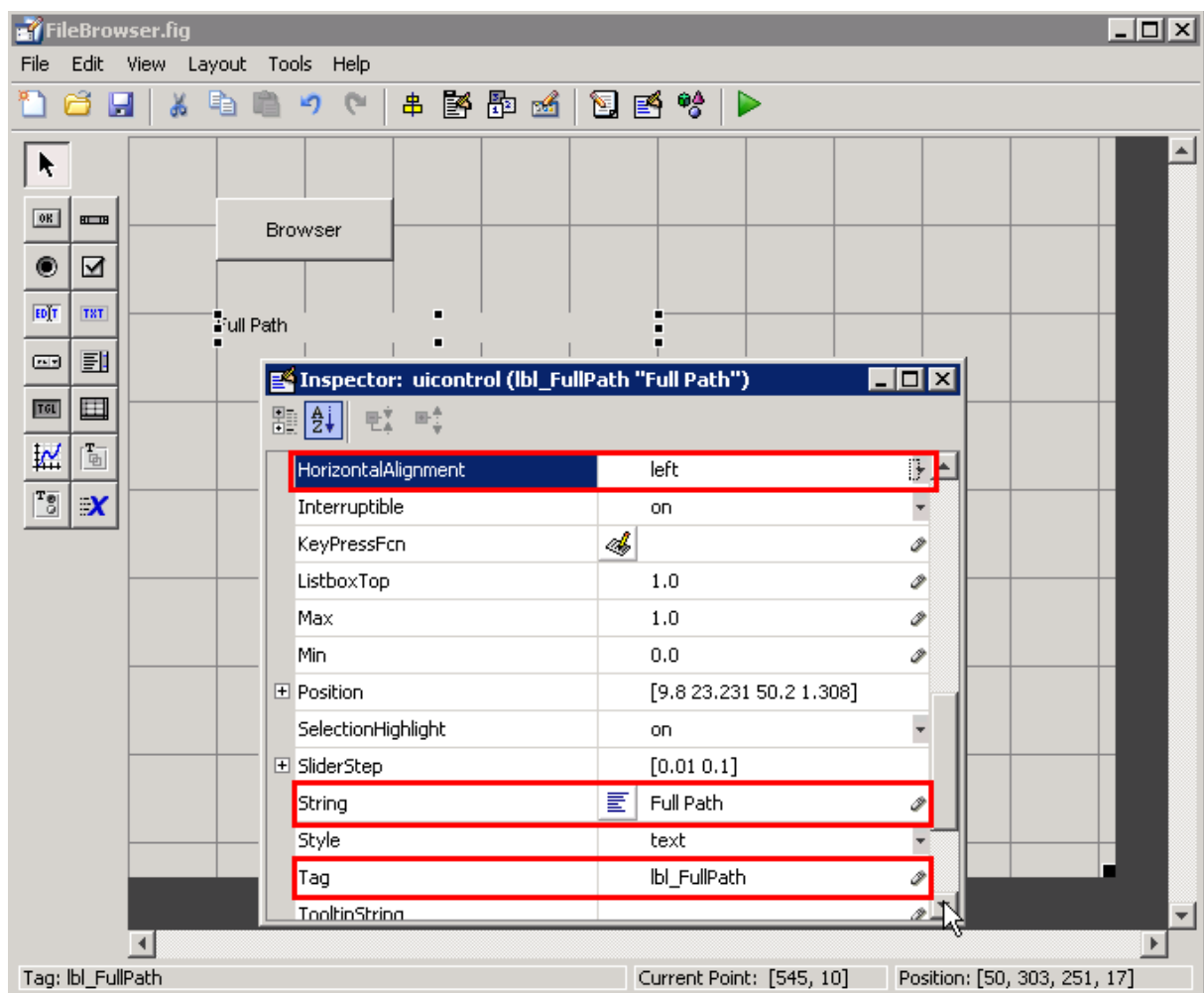
First we create the GUI:



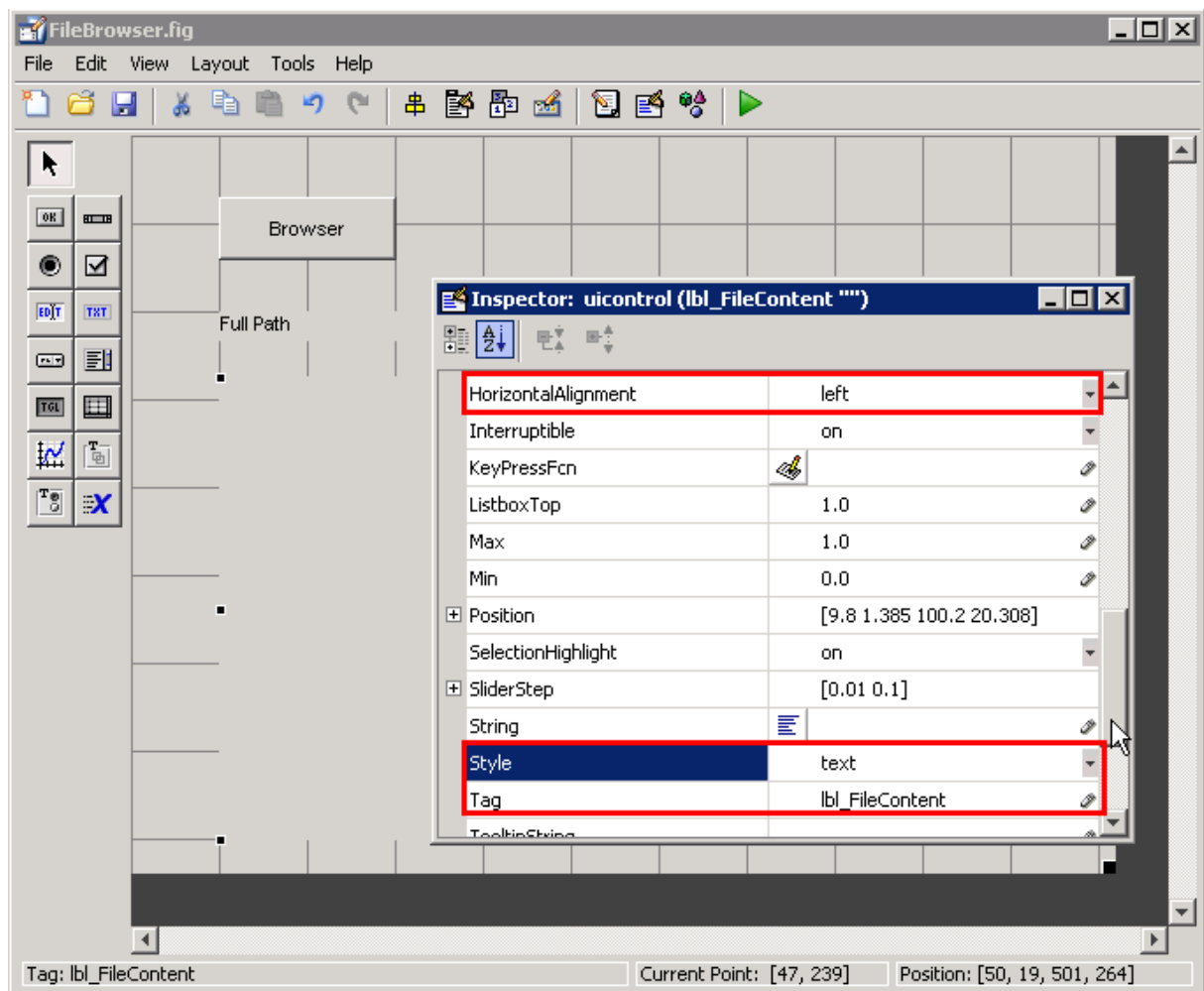
We change the **String** property and the **Tag**:



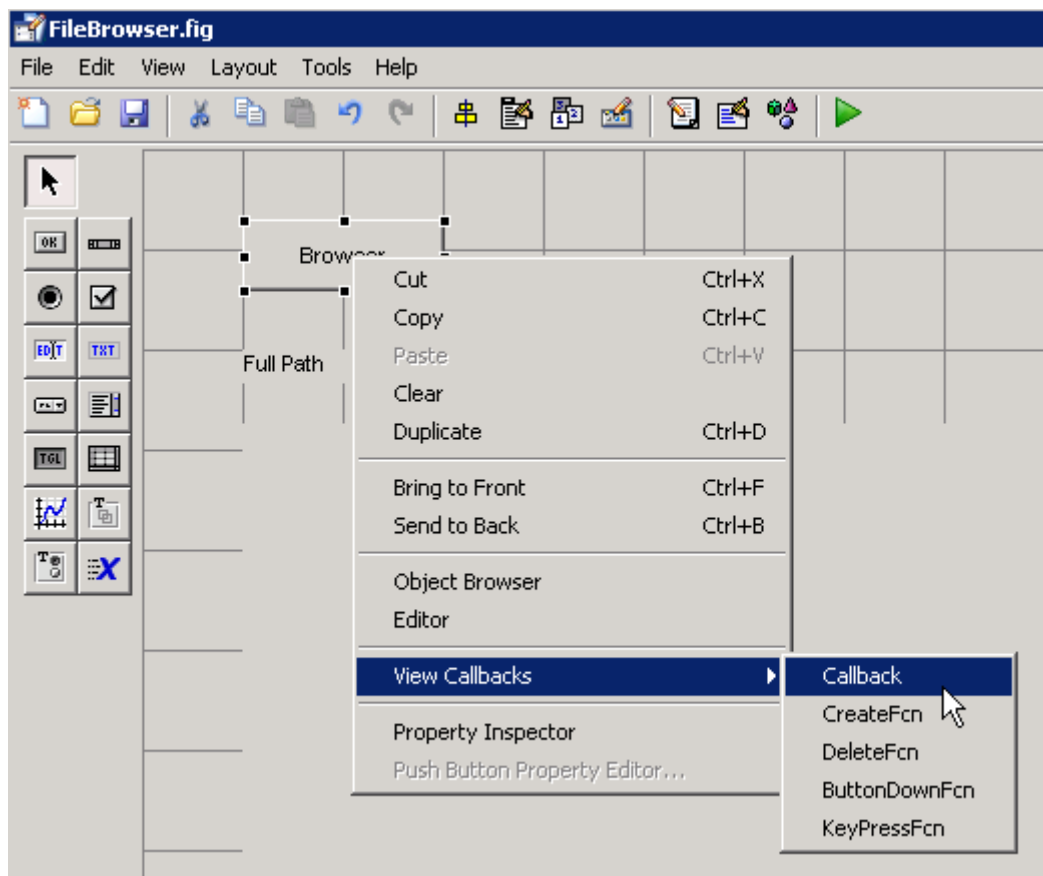
We add a **Static text** area and change a few standard stuff:



We add the main Static text area afterwards:



Now we click make the **Callback** script of the main button visible:



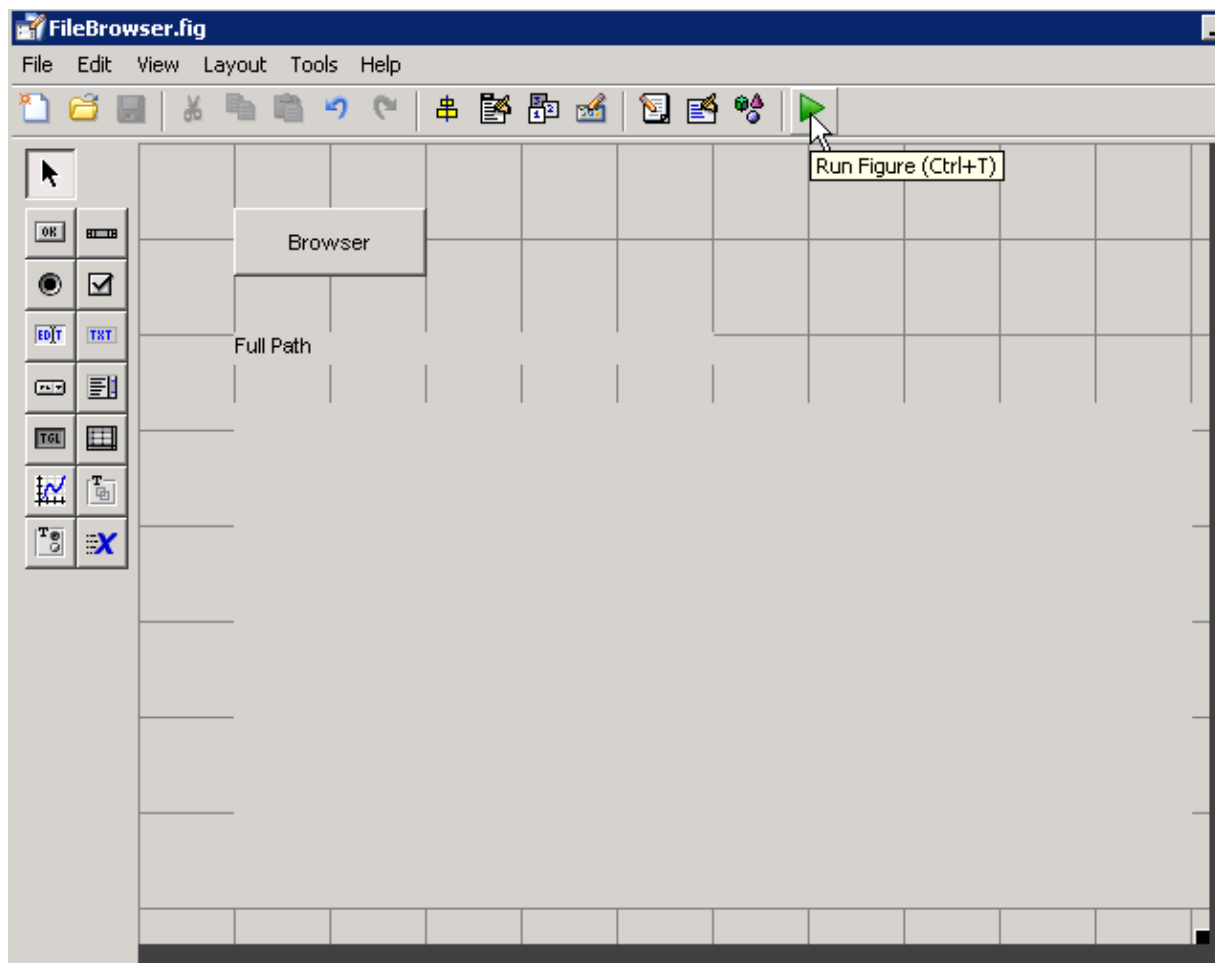
And we add the following:

```

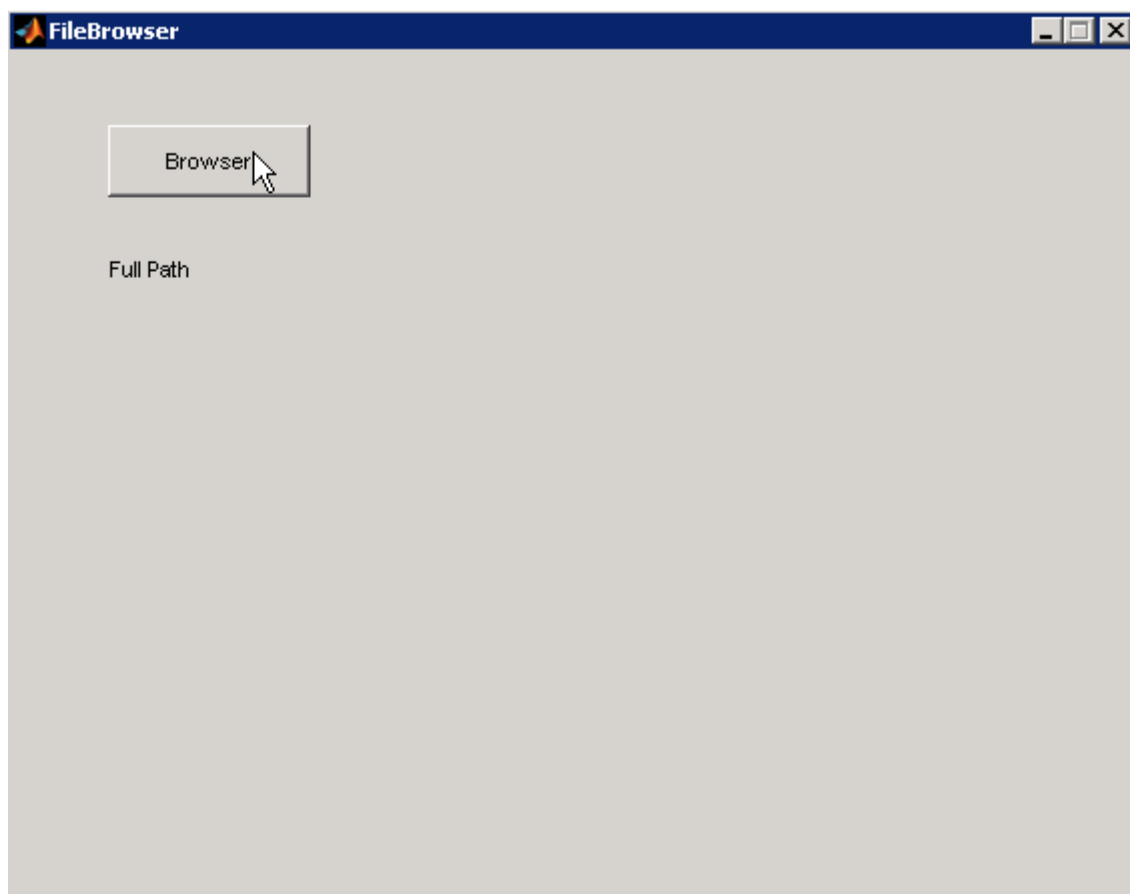
71
72 % Get default command line output from handles structure
73 varargout{1} = handles.output;
74
75
76 % --- Executes on button press in btn_Browse.
77 function btn_Browse_Callback(hObject, eventdata, handles)
78 % hObject    handle to btn_Browse (see GCBO)
79 % eventdata  reserved - to be defined in a future version of MATLAB
80 % handles    structure with handles and user data (see GUIDATA)
81 [filename pathname]=uigetfile({'*.csv'},'File Selector');
82 str_fullpathname=strcat(pathname,filename);
83 set(handles.lbl_FullPath,'String',str_fullpathname);
84 str_Content=fileread(str_fullpathname);
85 set(handles.lbl_FileContent,'String',str_Content);

```

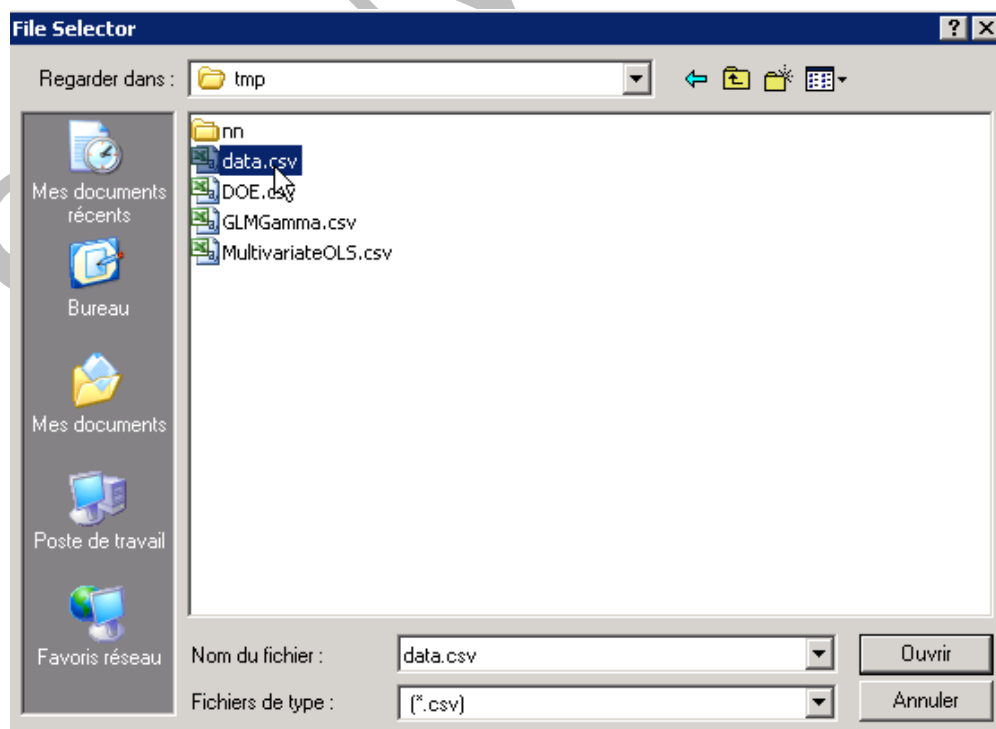
Now we can test it:



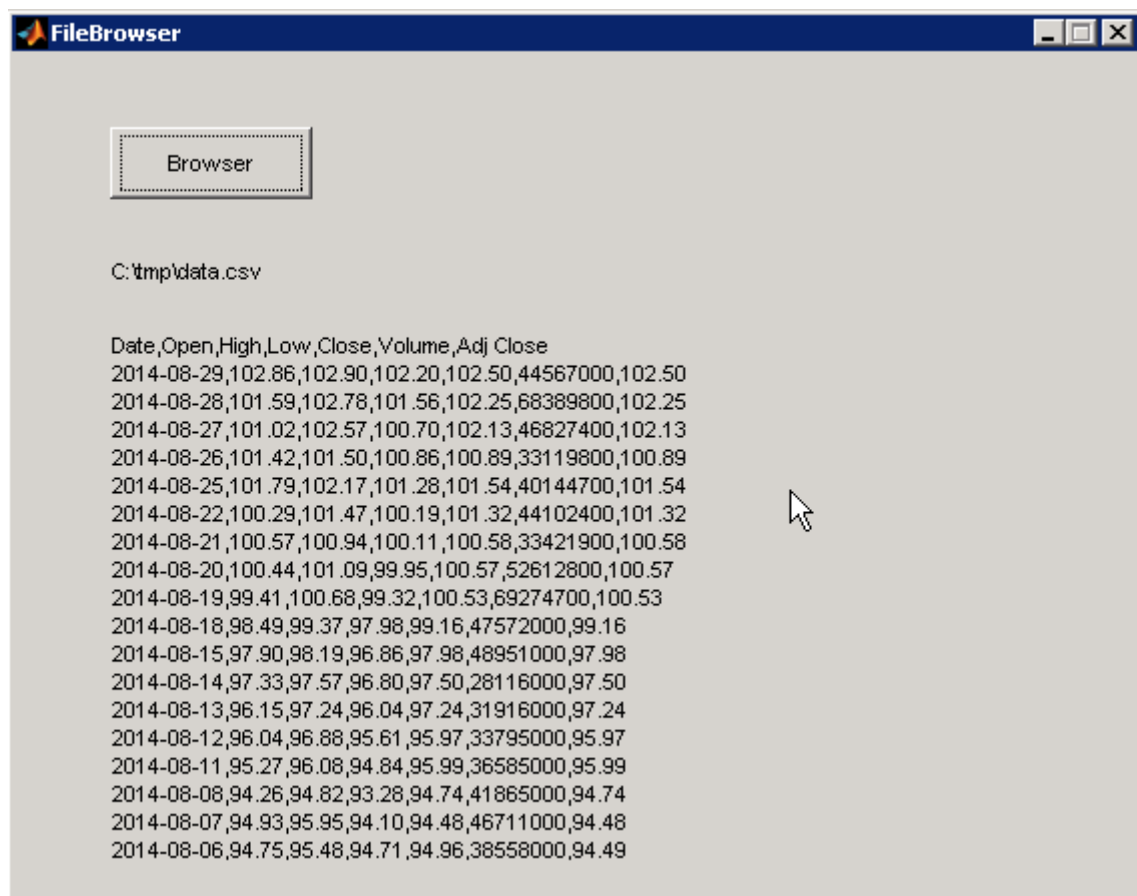
Once executed it looks like this:



When we click on *Browse* we get as expected:



Once the *.csv file choosed we have the the result we wanted:



Symbolic calculations (Symbolic Toolbox)

Now that we have finished with the core basic functions of MATLAB, it is time to focus on the first Toolbox in order of importance that is the: Symbolic Toolbox.

You will see in this chapter the most common commands use in business and engineering and also discover the similarities between MATLAB and Maple about the subjects covered in my book of Applied Mathematics.

New main core features (to be complete)

Note that all changes are not documented here! Please refer to the official release notes for a full list.

<http://www.mathworks.ch/ch/help/symbolic/release-notes.html>

MATLAB 2014a (8.3)

- Quickly compute cosinus/sinus hyperbolic
- max and min for finding the largest and smallest elements of a symbolic array

MATLAB 2013b (8.2)

- *cumsum* and *cumprod* functions for computing cumulative sums and products
- close function to close MuPAD notebooks from MATLAB

MATLAB 2013a (8.1)

- Linear algebra functions for computing matrix factorizations (*lu*, *chol*), pseudoinverse, orthogonal basis, and adjoint
- *sign* function for computing signs of numbers

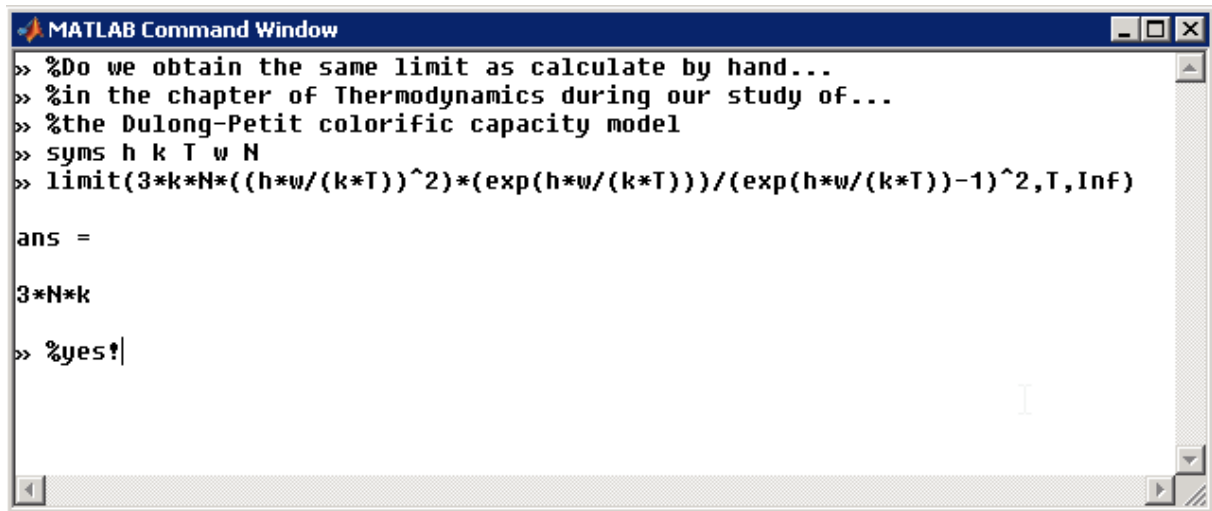
MATLAB 2012b (8.0)

- matrix analysis functions for characteristic *charpoly*
- *poles* function for determining the poles of an expression
- Functions for converting linear systems of equations to matrix form $AX=B$ (*equationsToMatrix*) and solving matrix equations (*linsolve*)
- symbolic functions for describing pulses: *rectangularPulse* and *triangularPulse*

Limits

The first important used command and one of the most simple is the **limit()** function to calculate the... limit O_o

For this purpose we will also need to learn how to create variable or constants symbols using the command **syms**:



```

MATLAB Command Window
>> %Do we obtain the same limit as calculate by hand...
>> %in the chapter of Thermodynamics during our study of...
>> %the Dulong-Petit colorific capacity model
>> syms h k T w N
>> limit(3*k*N*((h*w/(k*T))^2)*(exp(h*w/(k*T)))/(exp(h*w/(k*T))-1)^2,T,Inf)

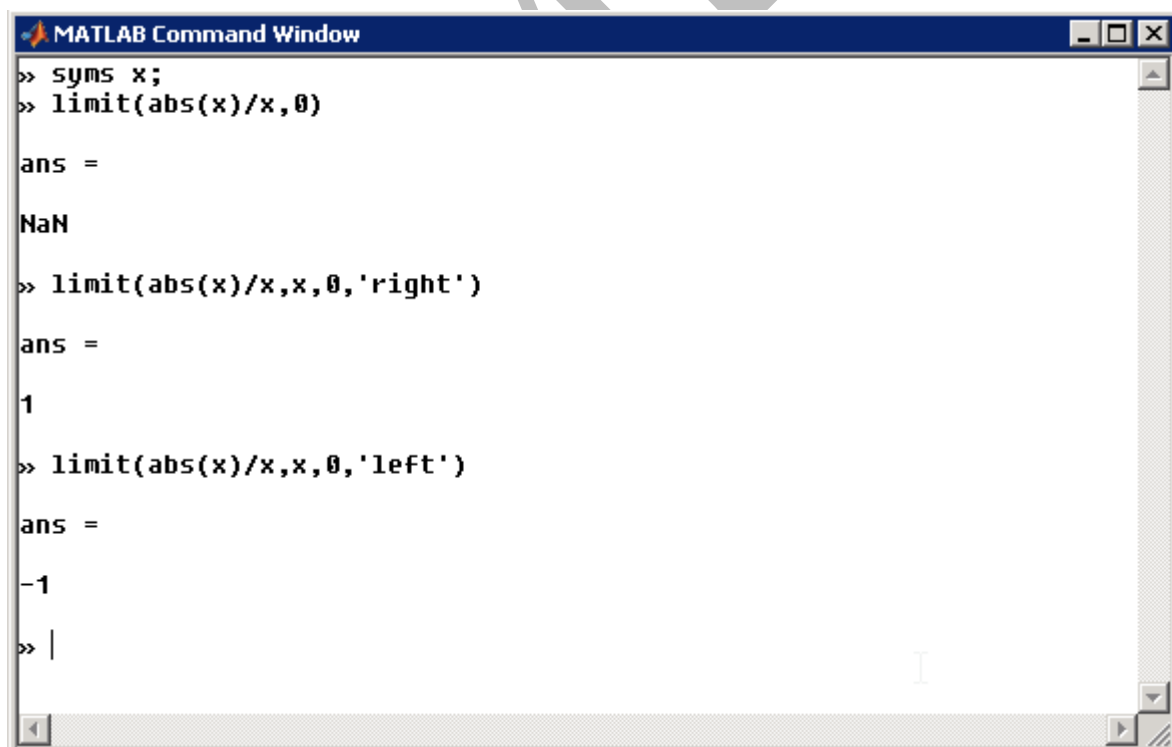
ans =

3*N*k

>> %yes!

```

Or another well know case (trap) about limits:



```

MATLAB Command Window
>> syms x;
>> limit(abs(x)/x,0)

ans =

NaN

>> limit(abs(x)/x,x,0,'right')

ans =

1

>> limit(abs(x)/x,x,0,'left')

ans =

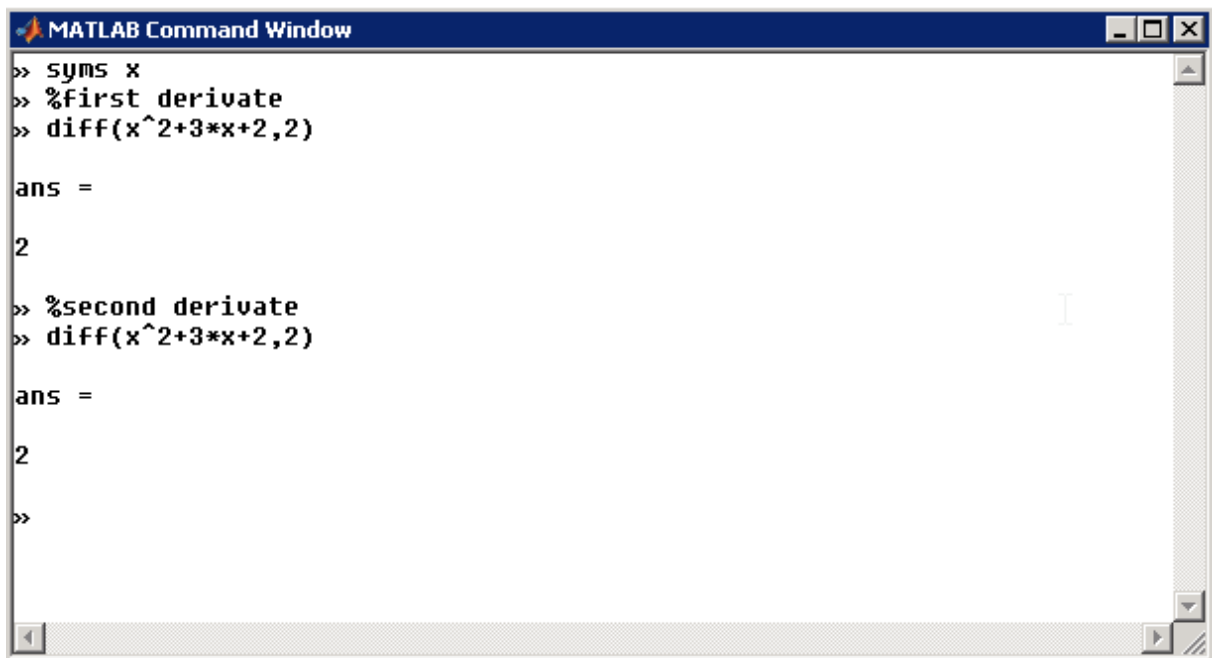
-1

>> |

```

Derivate/Partial Derivate

As we saw with the core version of MATLAB, derivation was limited to pseudo-symbolic polynomial only. With the Symbolic Toolbox we can derivate almost all classical functions and estimate their value on a given point using the function **diff()**:



```
MATLAB Command Window
>> syms x
>> %first derivate
>> diff(x^2+3*x+2,2)

ans =

2

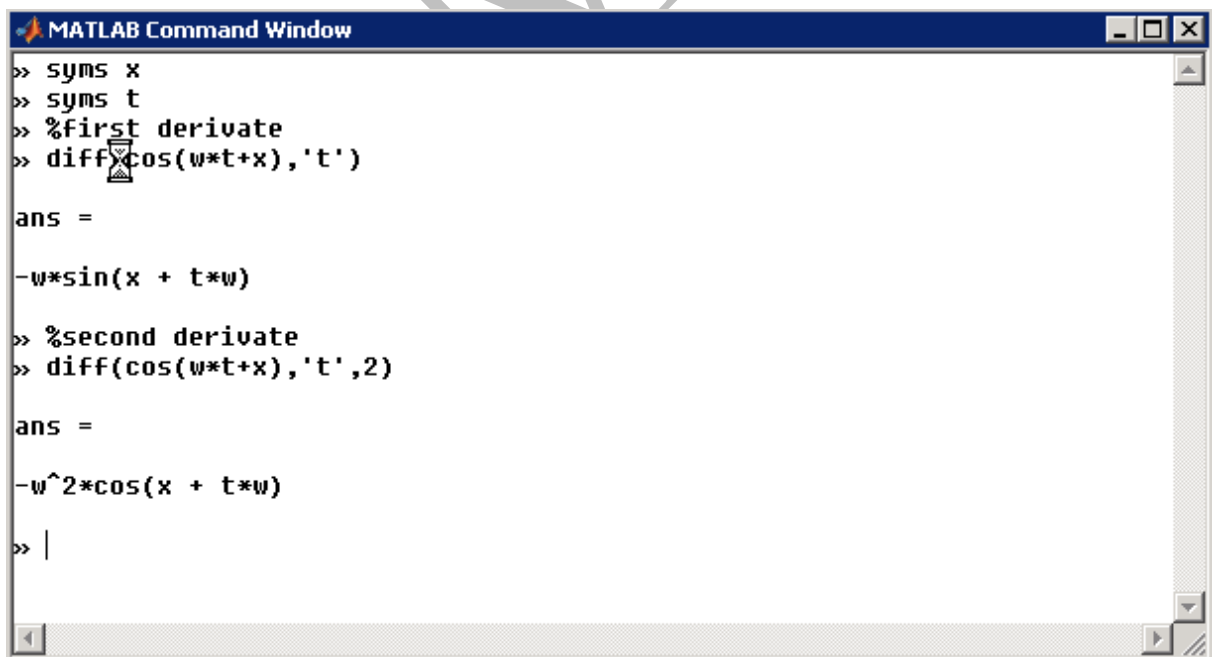
>> %second derivate
>> diff(x^2+3*x+2,2)

ans =

2

>>
```

Or if we want a pure symbolic derivate without being interested to the evaluation on a given point we have (always using the **diff()** function):



```
MATLAB Command Window
>> syms x
>> syms t
>> %first derivate
>> diff(cos(w*t+x),'t')

ans =

-w*sin(x + t*w)

>> %second derivate
>> diff(cos(w*t+x),'t',2)

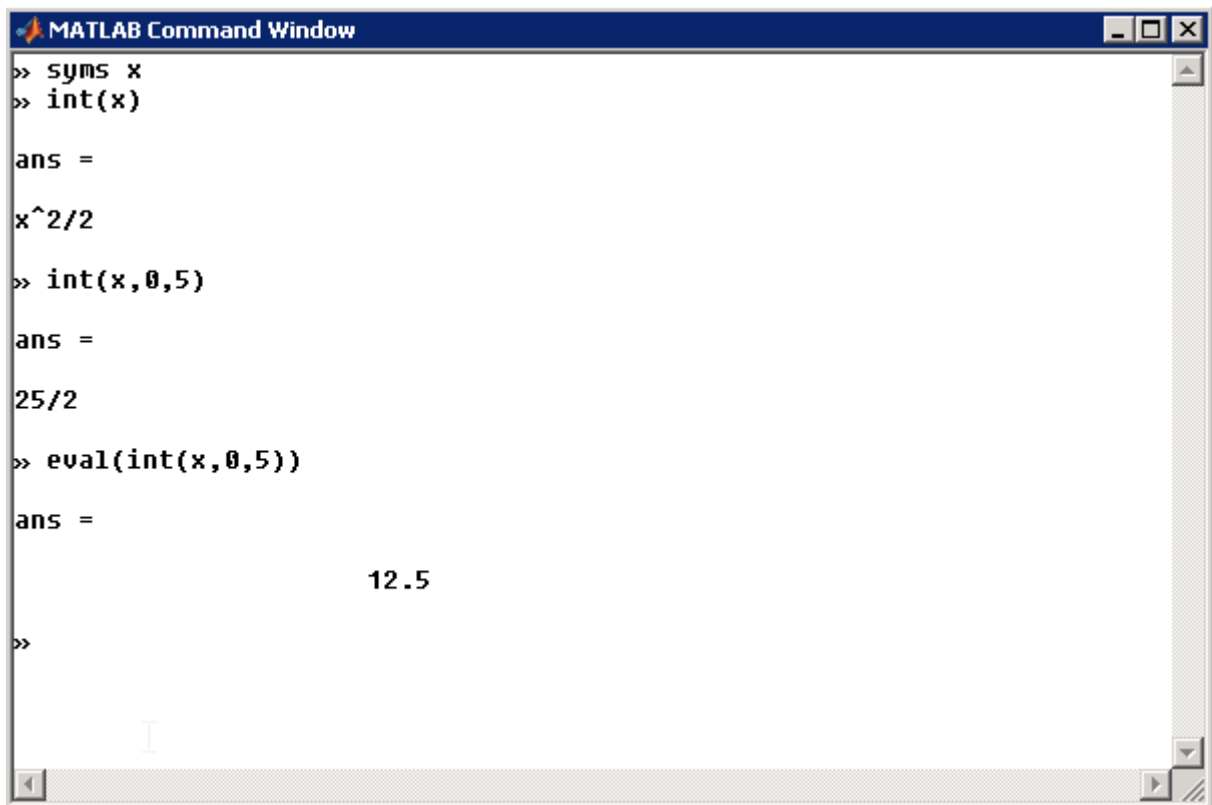
ans =

-w^2*cos(x + t*w)

>> |
```

Integration

As we saw with the core version of MATLAB, derivation was limited to pseudo-symbolic polynomials only. With the Symbolic Toolbox we can integrate almost all classical functions and estimate their value on a given point using the function `int()` alone or associated with `eval()`:

A screenshot of the MATLAB Command Window. The window has a blue title bar with the MATLAB logo and the text "MATLAB Command Window". The command history shows: `>> syms x`, `>> int(x)`, `ans =`, `x^2/2`, `>> int(x,0,5)`, `ans =`, `25/2`, `>> eval(int(x,0,5))`, `ans =`, `12.5`, and `>>`. The output `12.5` is displayed in a larger font size than the other text.

```
>> syms x
>> int(x)

ans =

x^2/2
>> int(x,0,5)

ans =

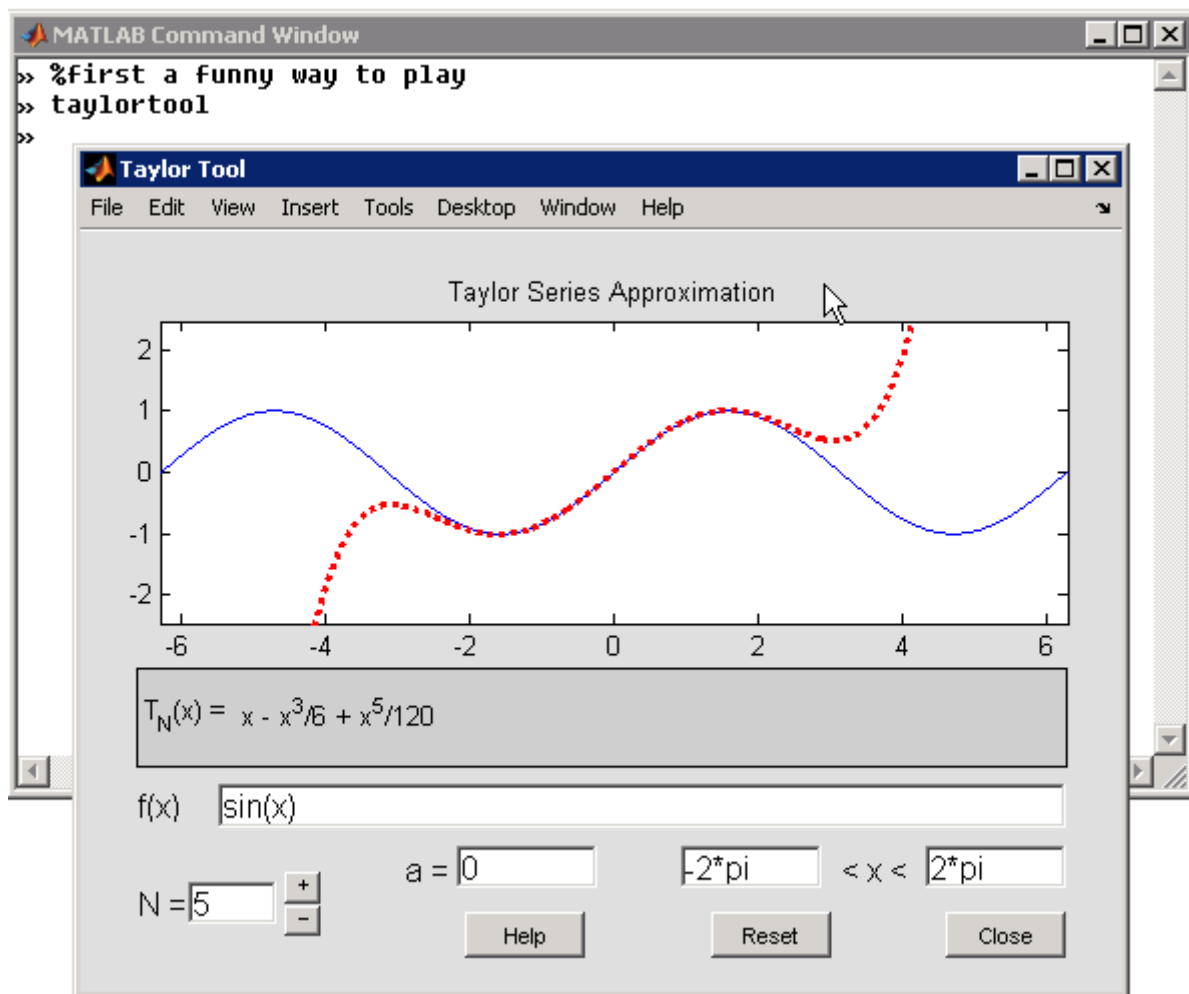
25/2
>> eval(int(x,0,5))

ans =

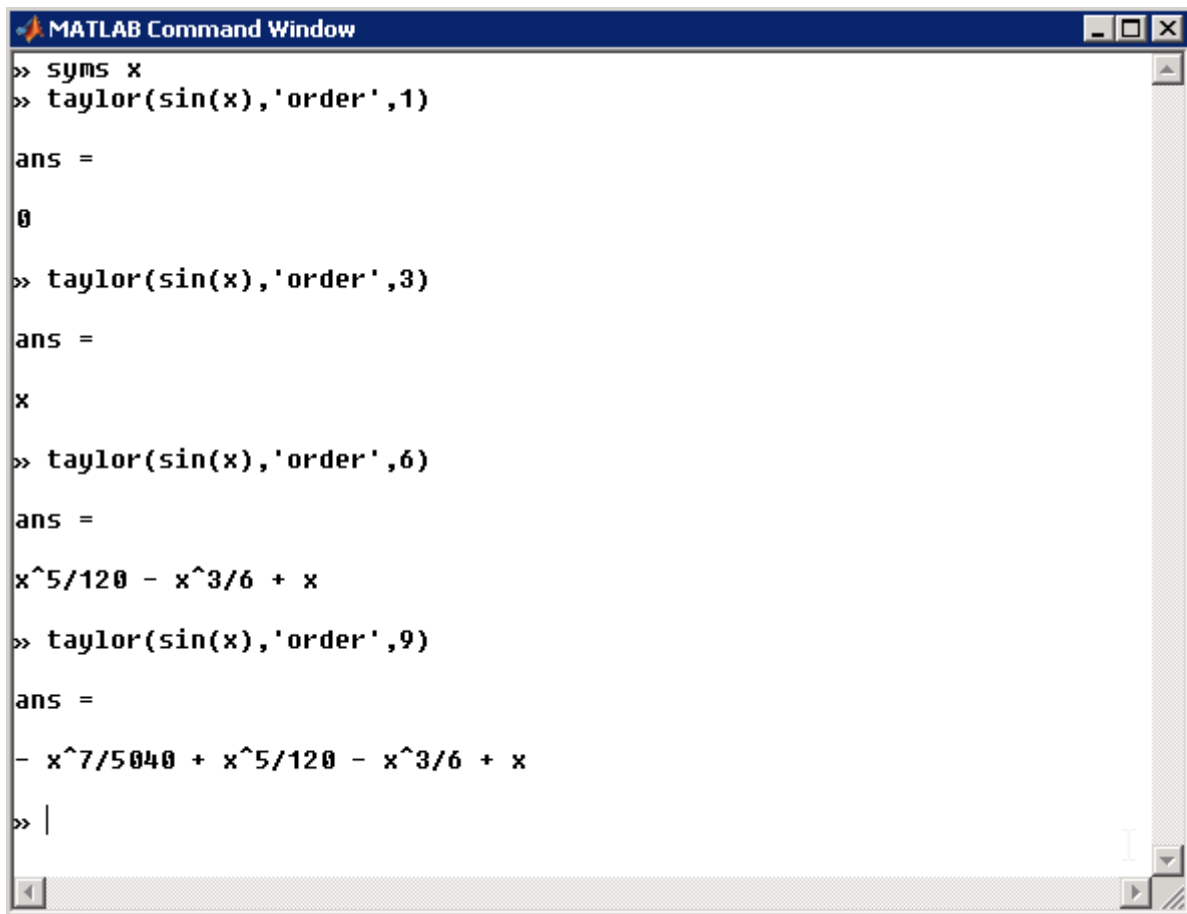
12.5
>>
```

Taylor series

The fact that Taylor series expansions are not available in the core version of MATLAB is a huge and boring limitation. At least, it is available in the Symbolic Toolbox and there are two ways of using it. The first one is through a GUI wizard called through the command `taylorTool`:



The way to get directly in the *Command Windows* the Taylor expansion is by using the function **taylor()** with some trivial arguments:



```
MATLAB Command Window
>> syms x
>> taylor(sin(x),'order',1)

ans =

0

>> taylor(sin(x),'order',3)

ans =

x

>> taylor(sin(x),'order',6)

ans =

x^5/120 - x^3/6 + x

>> taylor(sin(x),'order',9)

ans =

- x^7/5040 + x^5/120 - x^3/6 + x

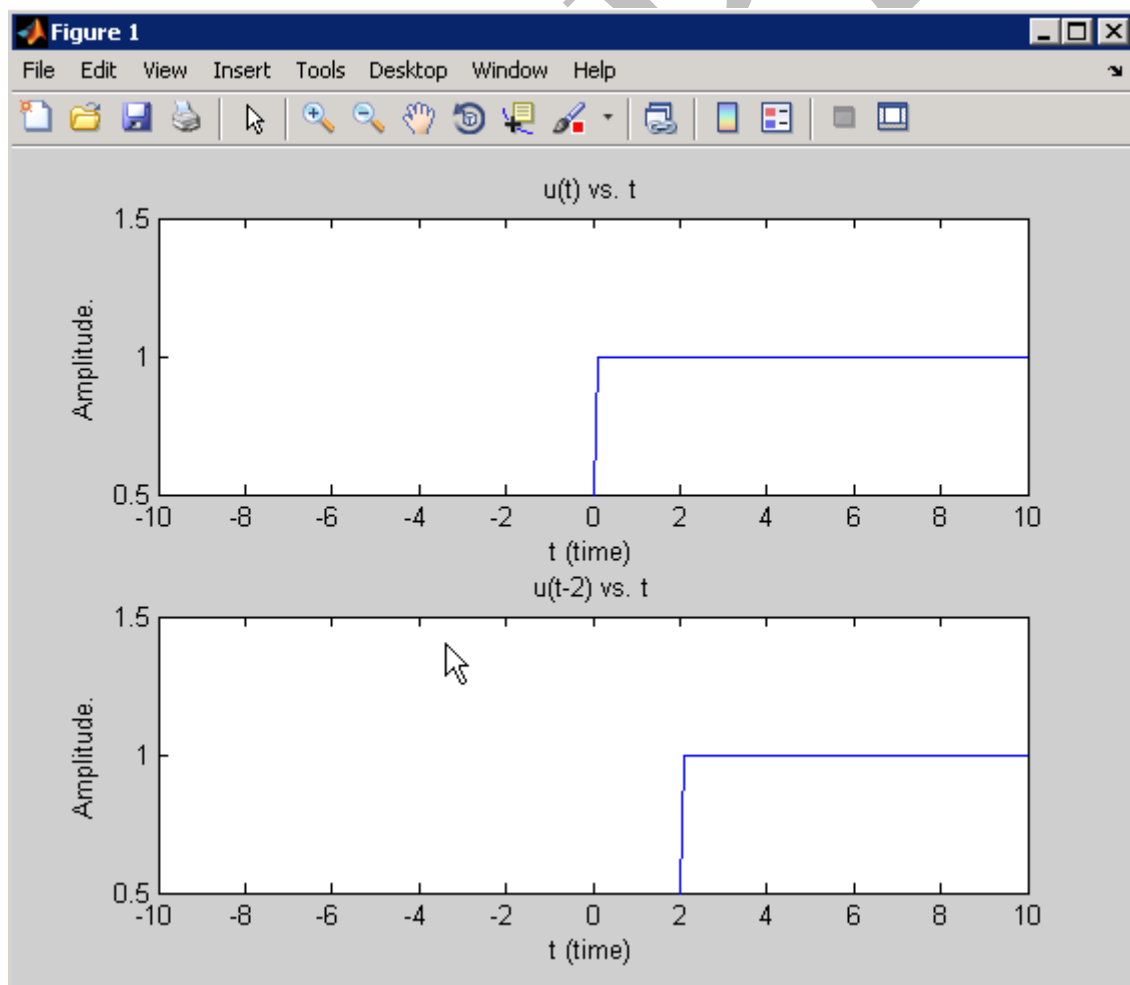
>> |
```

Heaviside function

If you have the symbolic toolbox you can use the **heaviside()** function to get a step function (very useful in electronics):

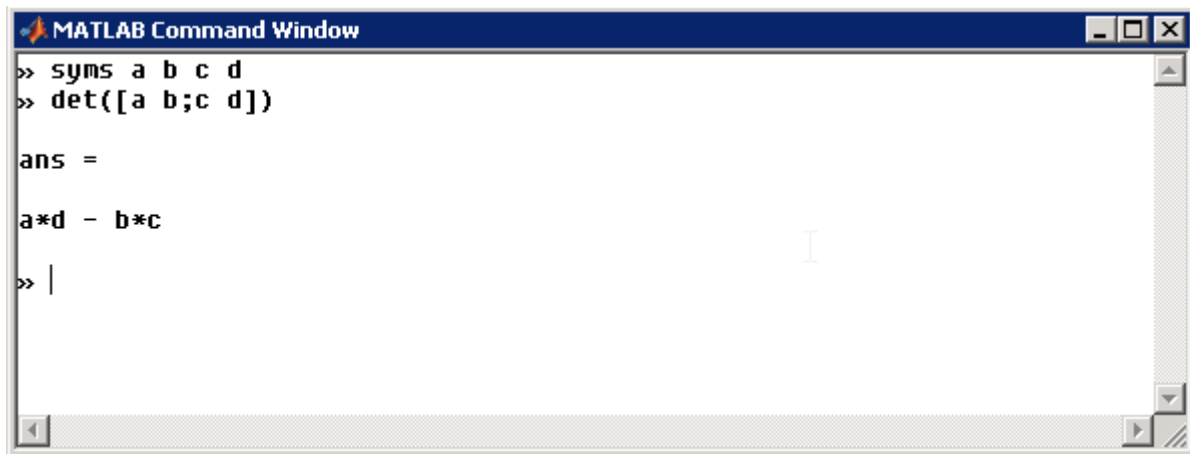
```
EDITOR PUBLISH VIEW
1 x = -10:0.1:10;
2 stepfun = heaviside(x);
3 subplot(2, 1, 1);
4 plot(x, stepfun)
5 xlabel('t (time)')
6 title('u(t) vs. t')
7 ylabel('Amplitude.')
8 axis([-10 10 ,0.5 1.5])
9 subplot(2, 1, 2);
10 stepfun = heaviside(x-2);
11 plot(x, stepfun)
12 axis([-10 10 ,0.5 1.5])
13 title('u(t-2) vs. t')
14 ylabel('Amplitude. ');
15 xlabel('t (time)');
```

So we get:



Matrix determinant

We saw that in the core MATLAB version it was very easy to get the determinant of a numeric matrix with the function **det()**. By passing to the same function variables instead of numerical values we can also get the determinant:

A screenshot of the MATLAB Command Window. The window has a blue title bar with the MATLAB logo and the text "MATLAB Command Window". The command prompt shows two lines of input: ">> syms a b c d" and ">> det([a b;c d])". The output shows "ans =" followed by the symbolic expression "a*d - b*c". The cursor is on a new line after the output.

```
>> syms a b c d
>> det([a b;c d])

ans =

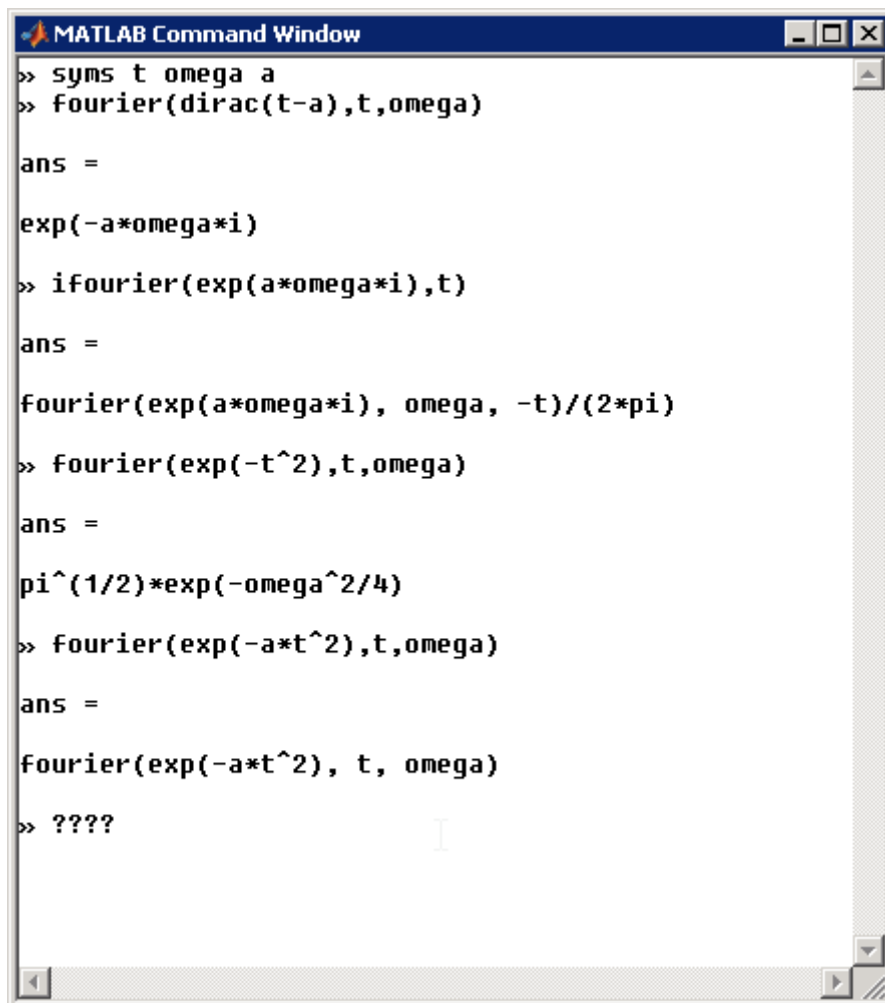
a*d - b*c
>> |
```

Integral Transformation

In the theoretical course we have seen how powerful are Fourier, Laplace or Z transformations to solve differential equations, or for calculating series values or for signal processing. The purpose here is just to make an example of each to see that we obtain the same results as those calculated by hand.

Fourier Transform

Have a look to some Fourier transformation we calculated by hand in the theoretical course:



```

MATLAB Command Window
>> syms t omega a
>> fourier(dirac(t-a),t,omega)

ans =

exp(-a*omega*i)
>> ifourier(exp(a*omega*i),t)

ans =

fourier(exp(a*omega*i), omega, -t)/(2*pi)
>> fourier(exp(-t^2),t,omega)

ans =

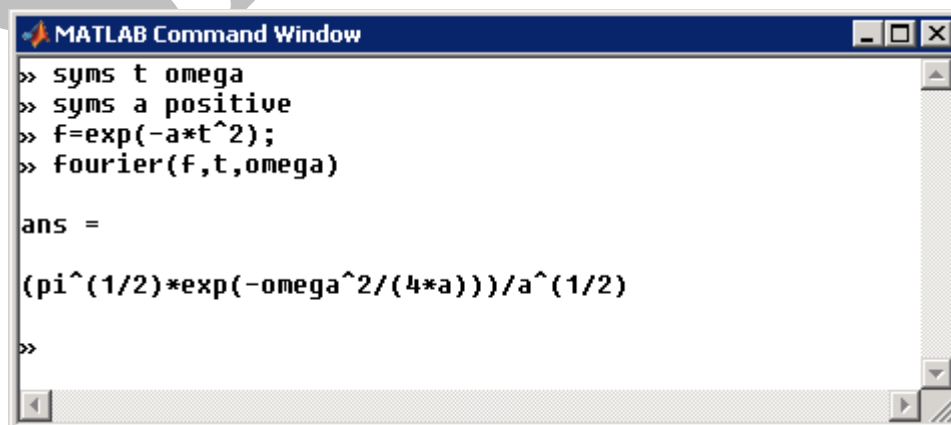
pi^(1/2)*exp(-omega^2/4)
>> fourier(exp(-a*t^2),t,omega)

ans =

fourier(exp(-a*t^2), t, omega)
>> ????

```

Everything seems ok excepted the last one (heat equation Fourier Transform) that MATLAB seems not be able to compute. But in reality as we know it, it is because of the sign of a . Thus:



```

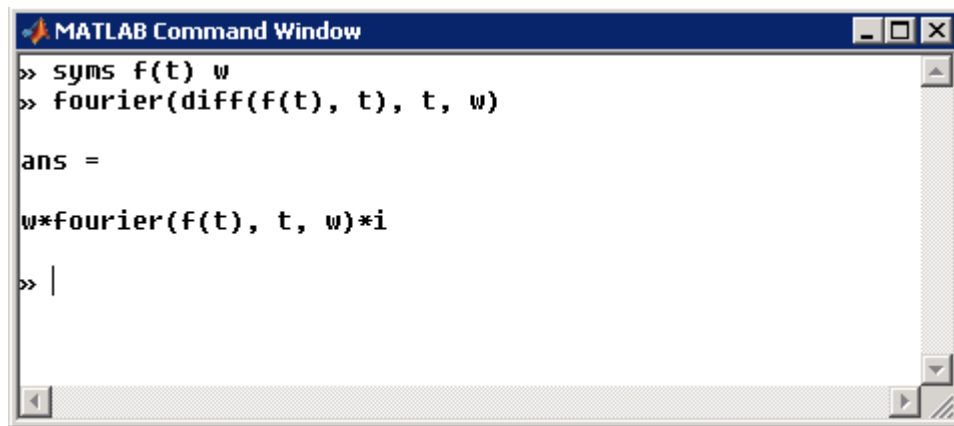
MATLAB Command Window
>> syms t omega
>> syms a positive
>> f=exp(-a*t^2);
>> fourier(f,t,omega)

ans =

(pi^(1/2)*exp(-omega^2/(4*a)))/a^(1/2)
>>

```

We have also the famous result useful for resolving PDE:



```

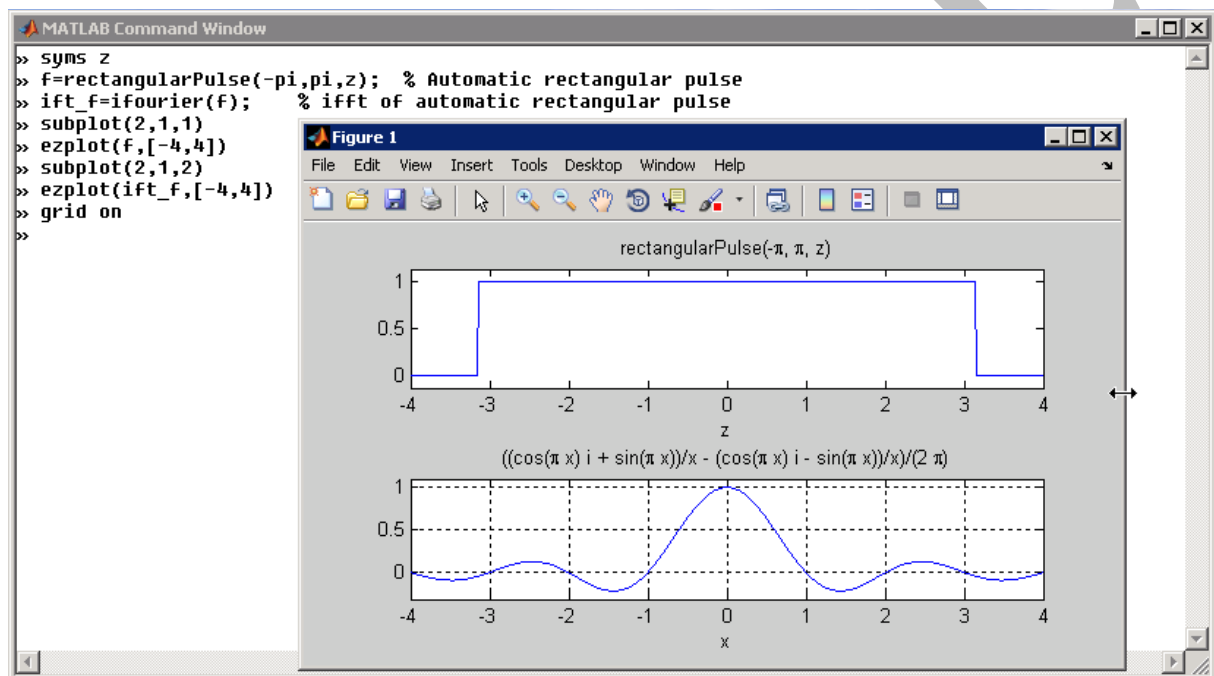
>> syms f(t) w
>> fourier(diff(f(t), t), t, w)

ans =

w*fourier(f(t), t, w)*i
>> |

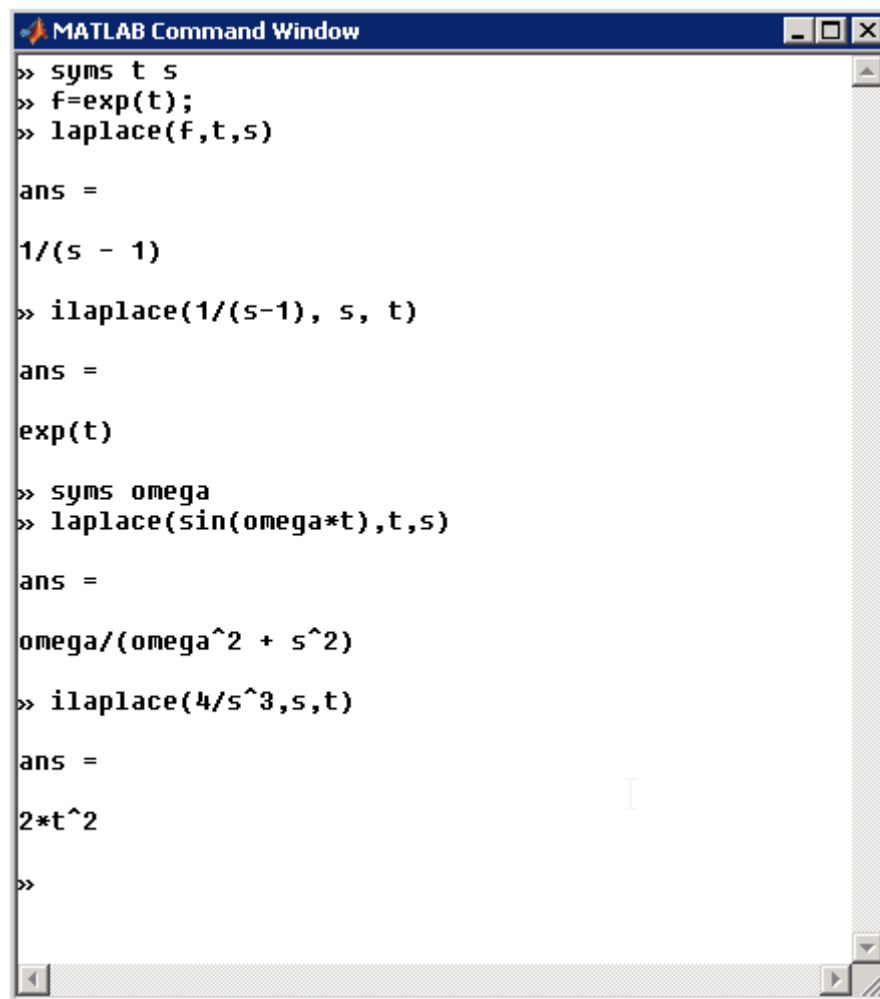
```

And another famous typical case that we have calculated by hand in the theoretical course using the function **rectangularPulse()** also from the Symbolic Math Toolbox.



Laplace Transform

Have a look to some Laplace transformation we calculated by hand in the theoretical course:

A screenshot of the MATLAB Command Window. The window has a blue title bar with the MATLAB logo and the text "MATLAB Command Window". The command history shows three separate operations. The first operation defines symbolic variables t and s, assigns f = exp(t), and computes the Laplace transform, resulting in 1/(s - 1). The second operation computes the inverse Laplace transform of 1/(s-1), resulting in exp(t). The third operation defines a symbolic variable omega, computes the Laplace transform of sin(omega*t), resulting in omega/(omega^2 + s^2), and then computes the inverse Laplace transform of 4/s^3, resulting in 2*t^2. The cursor is at the end of the last command line.

```
>> syms t s
>> f=exp(t);
>> laplace(f,t,s)

ans =

1/(s - 1)

>> ilaplace(1/(s-1), s, t)

ans =

exp(t)

>> syms omega
>> laplace(sin(omega*t),t,s)

ans =

omega/(omega^2 + s^2)

>> ilaplace(4/s^3,s,t)

ans =

2*t^2

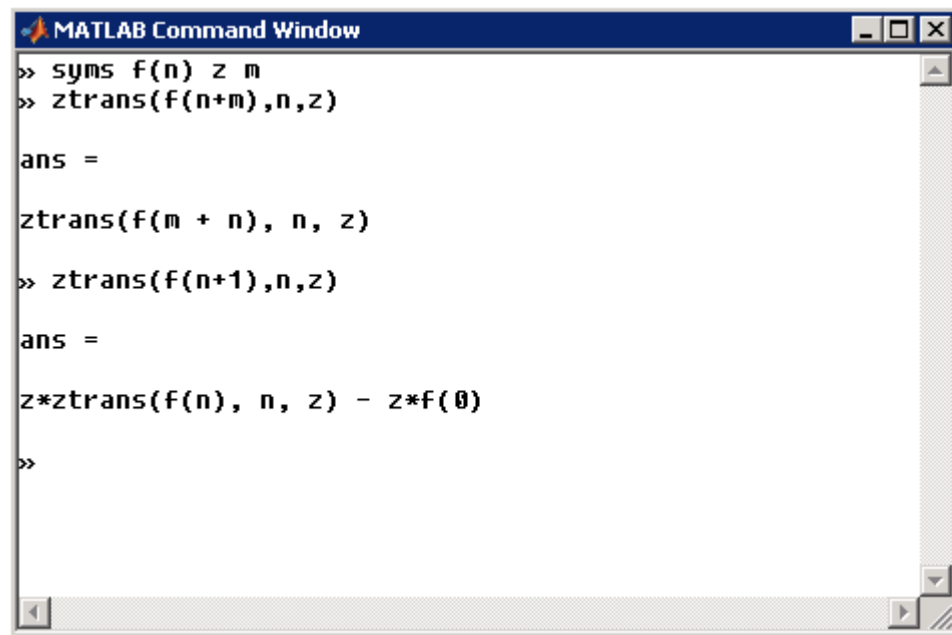
>>
```

Everything is OK here!

Z Transform

For the Z transform we will focus only on the practical financial case of this transformation that we have study during the theoretical course.

Let's have a look:

A screenshot of the MATLAB Command Window. The window title is "MATLAB Command Window". The command prompt shows two commands: `>> syms f(n) z m` and `>> ztrans(f(n+m),n,z)`. The output is `ans = ztrans(f(m + n), n, z)`. Then, the command `>> ztrans(f(n+1),n,z)` is entered, and the output is `ans = z*ztrans(f(n), n, z) - z*f(0)`. The prompt `>>` is shown at the bottom.

```
>> syms f(n) z m
>> ztrans(f(n+m),n,z)

ans =

ztrans(f(m + n), n, z)
>> ztrans(f(n+1),n,z)

ans =

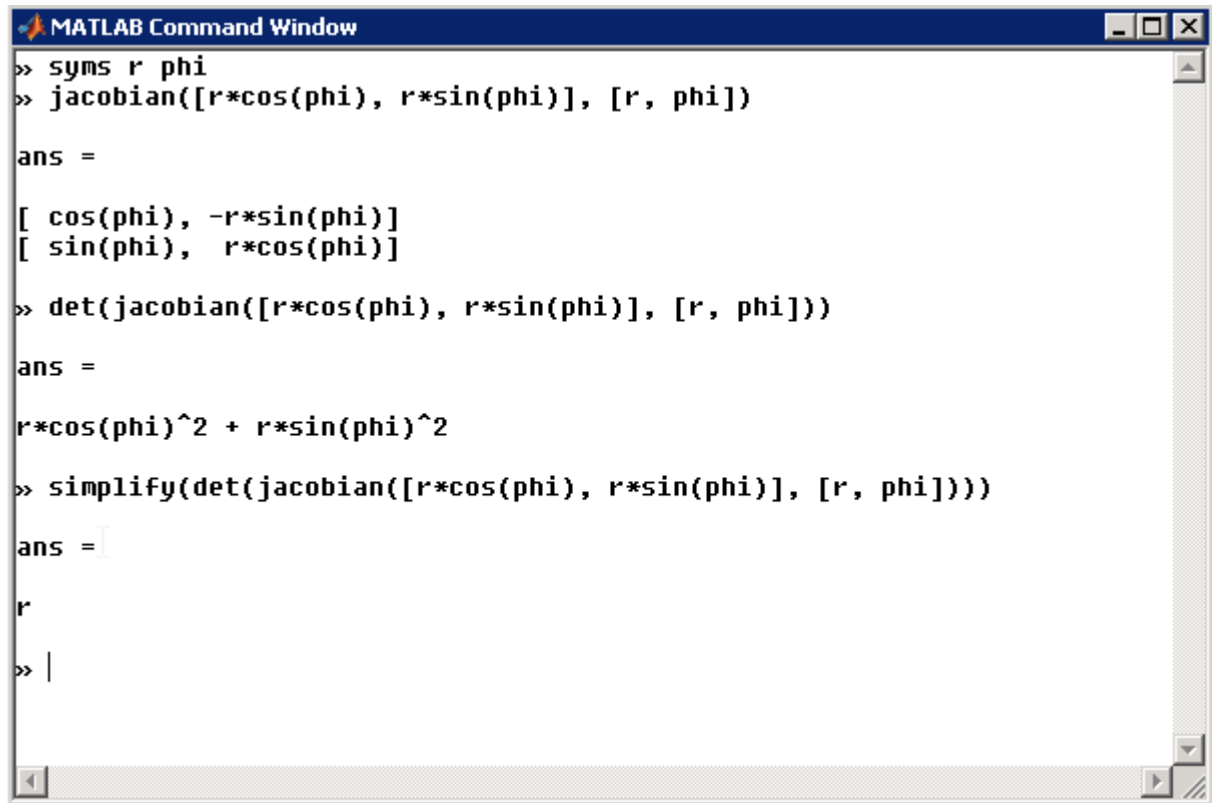
z*ztrans(f(n), n, z) - z*f(0)
>>
```

As we can see, MATLAB seems not to be able to manager a general result for any m but if we chose a special value than the result is ok.

Jacobien

As we saw in the course of applied mathematics the jacobian has an important place in some integrals and even sometimes in some statistical tests.

Let us see if we can found back the same result as the one calculate by hand using the function **jacobian()**:

A screenshot of the MATLAB Command Window. The window has a blue title bar with the MATLAB logo and the text "MATLAB Command Window". The command history shows the following:

```
>> syms r phi
>> jacobian([r*cos(phi), r*sin(phi)], [r, phi])
```

 The output is:

```
ans =
[ cos(phi), -r*sin(phi)]
[ sin(phi),  r*cos(phi)]
```

 Then, the user enters:

```
>> det(jacobian([r*cos(phi), r*sin(phi)], [r, phi]))
```

 The output is:

```
ans =
r*cos(phi)^2 + r*sin(phi)^2
```

 Finally, the user enters:

```
>> simplify(det(jacobian([r*cos(phi), r*sin(phi)], [r, phi])))
```

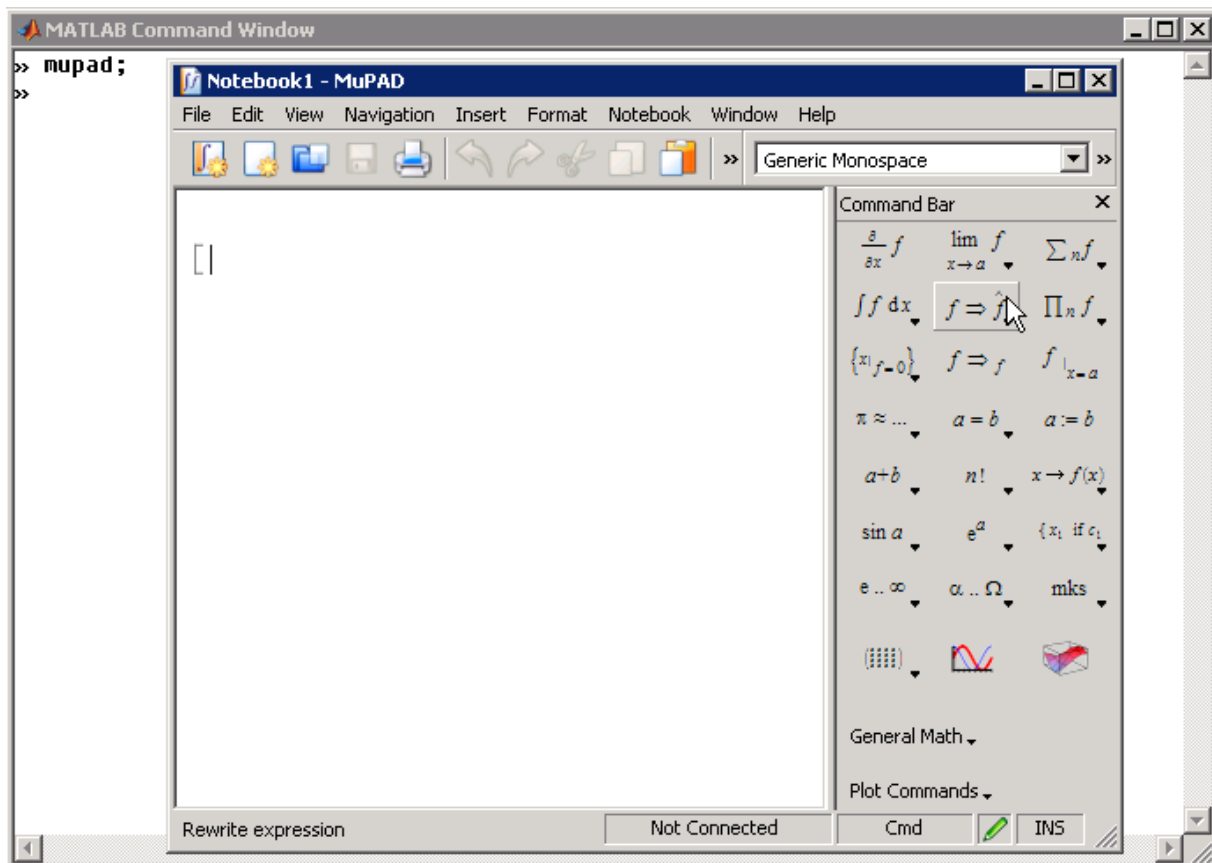
 The output is:

```
ans =
r
```

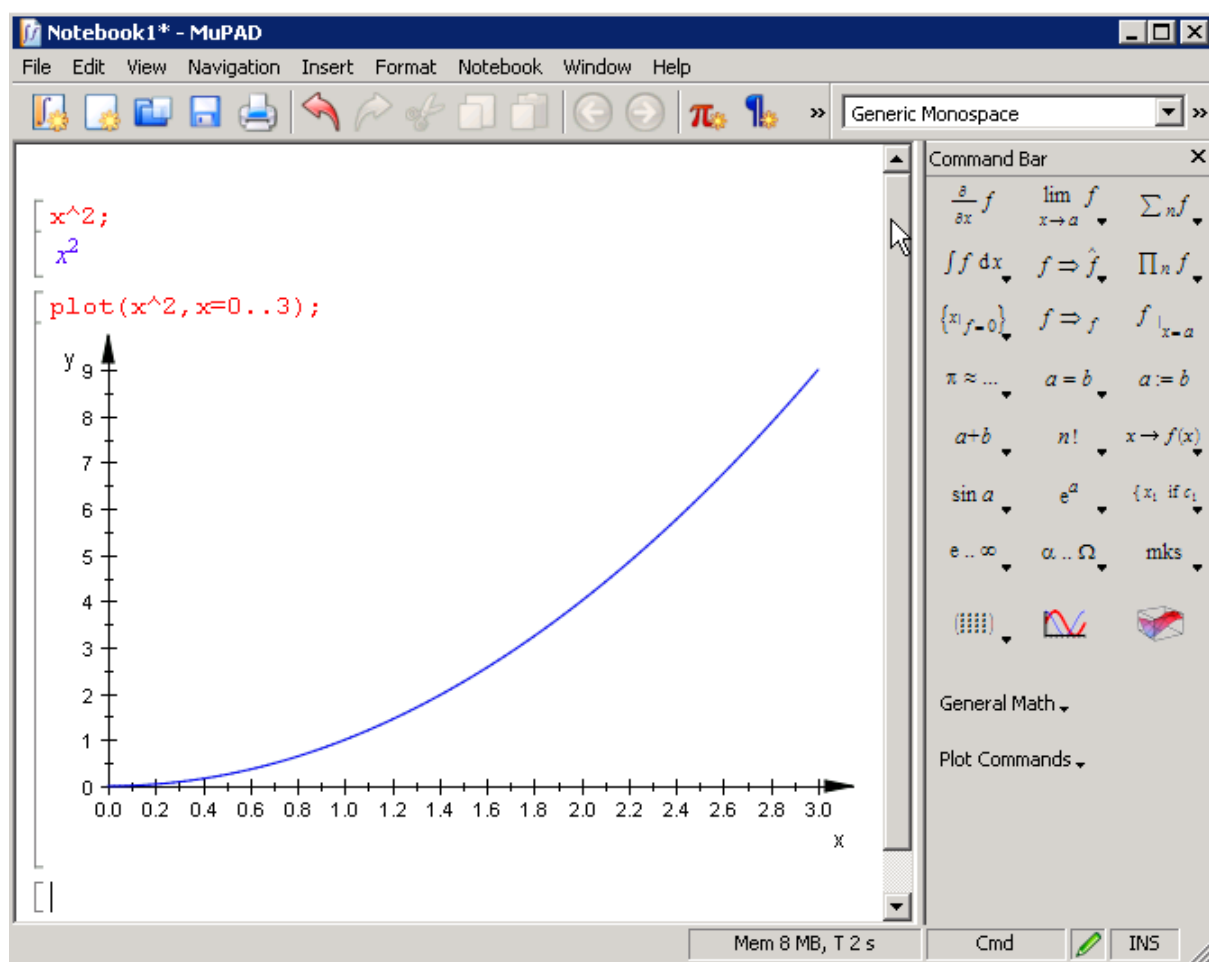
 The cursor is at the end of the last command line.

MuPAD

The MuPAD is a Maple like interface in MATLAB to make symbolic calculation and charting. To start it just type **mupad** in the *Command Window*:



And now we just make a small example and as you can see it is very similar to Maple (at least for basics features):



Databases (with Database/Datafeed toolbox)

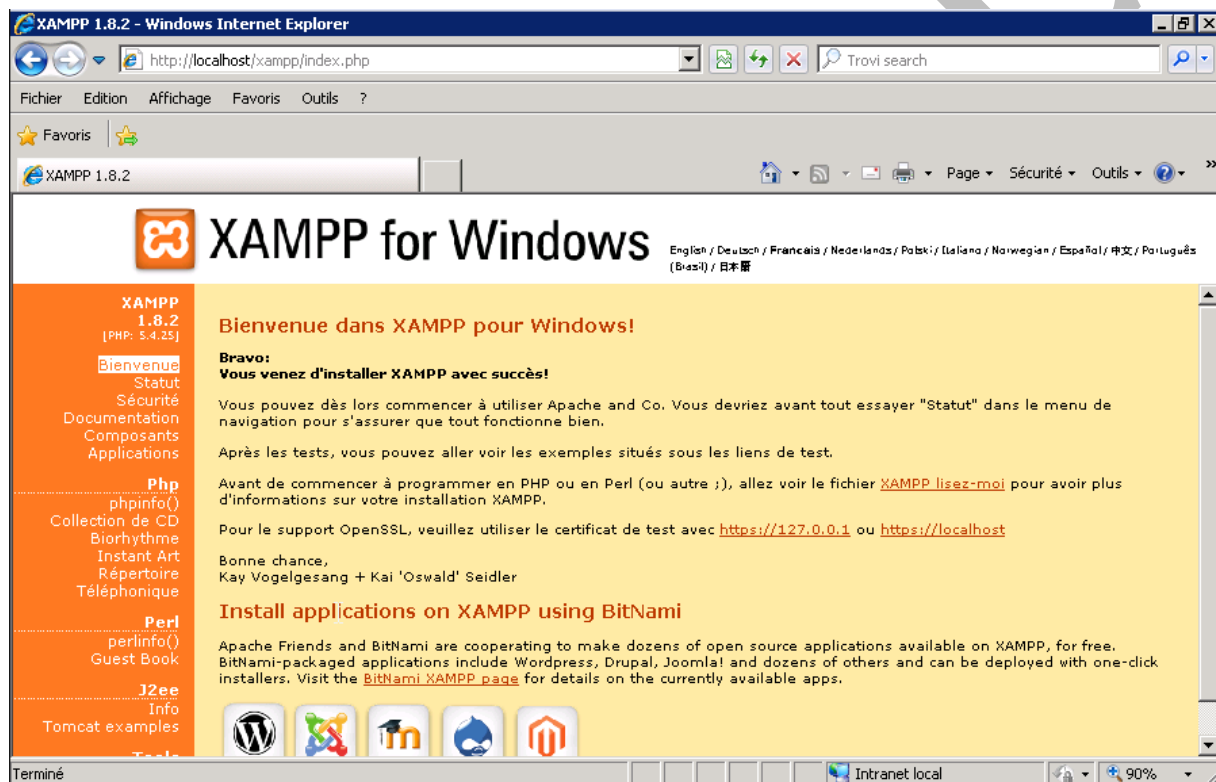
<http://www.mathworks.ch/ch/help/database/release-notes.html>

Access to external databases is a major topic in almost all statistical softwares and especially for practitioners in Quantitative Finance, Data Mining and Machine Learning (Big Data).

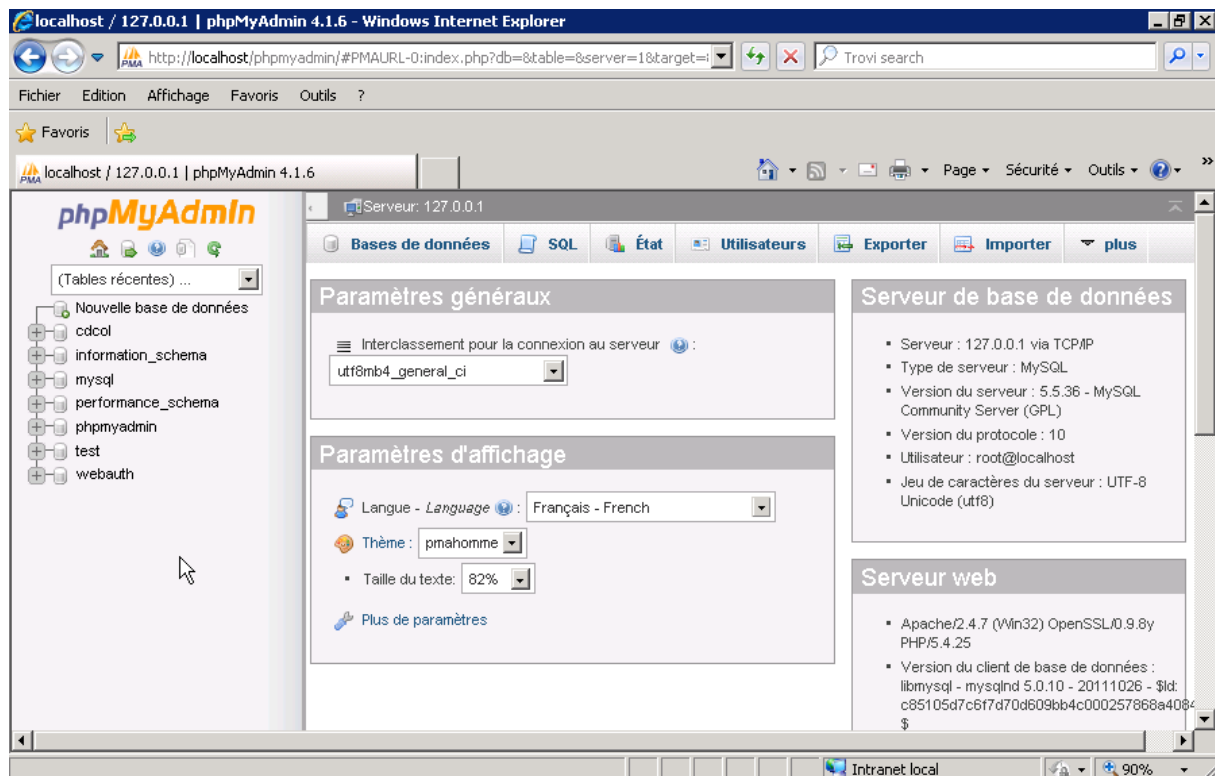
We will see here the most common example to access and retrieve datas form mySQL, Microsoft Access, Oracle Express and SQL Server Express.

Connect to mySQL 5.5.36

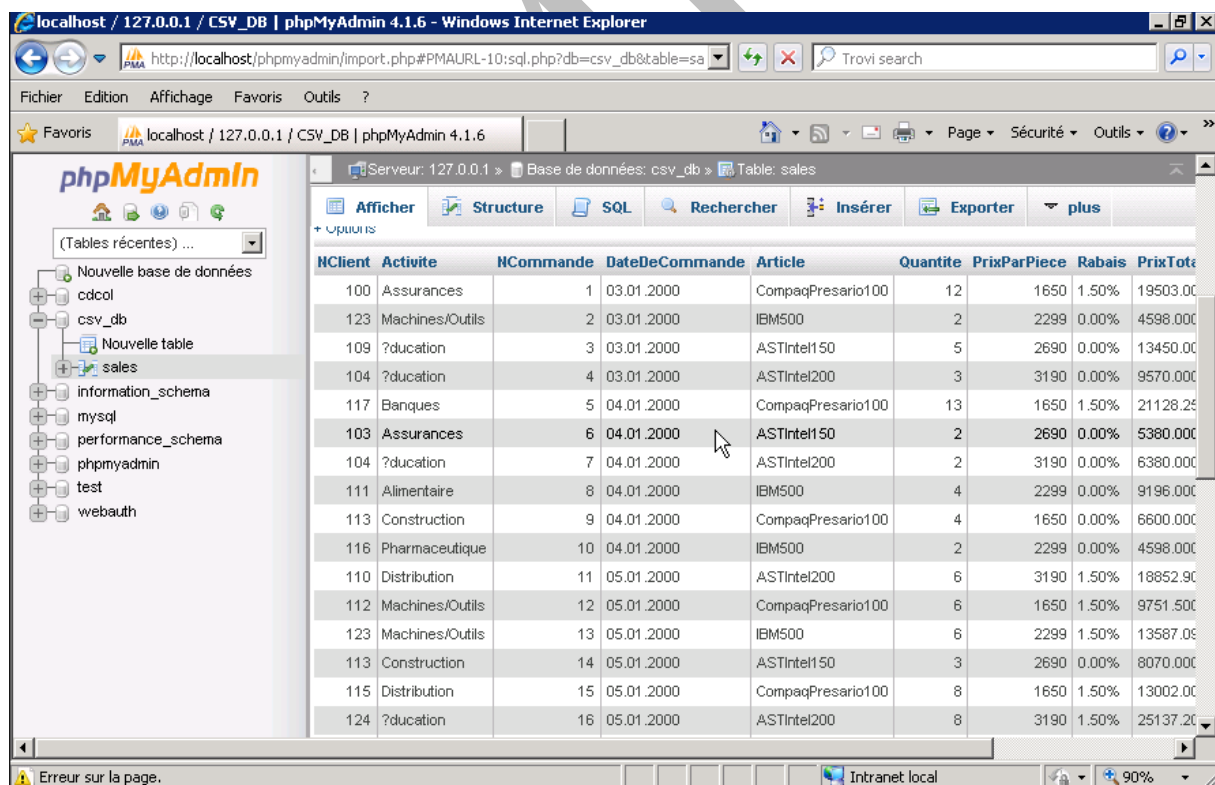
We will begin with mySQL that was for this example installed automatically with XAMPP server:



Then if you click on the mySQL link you get:

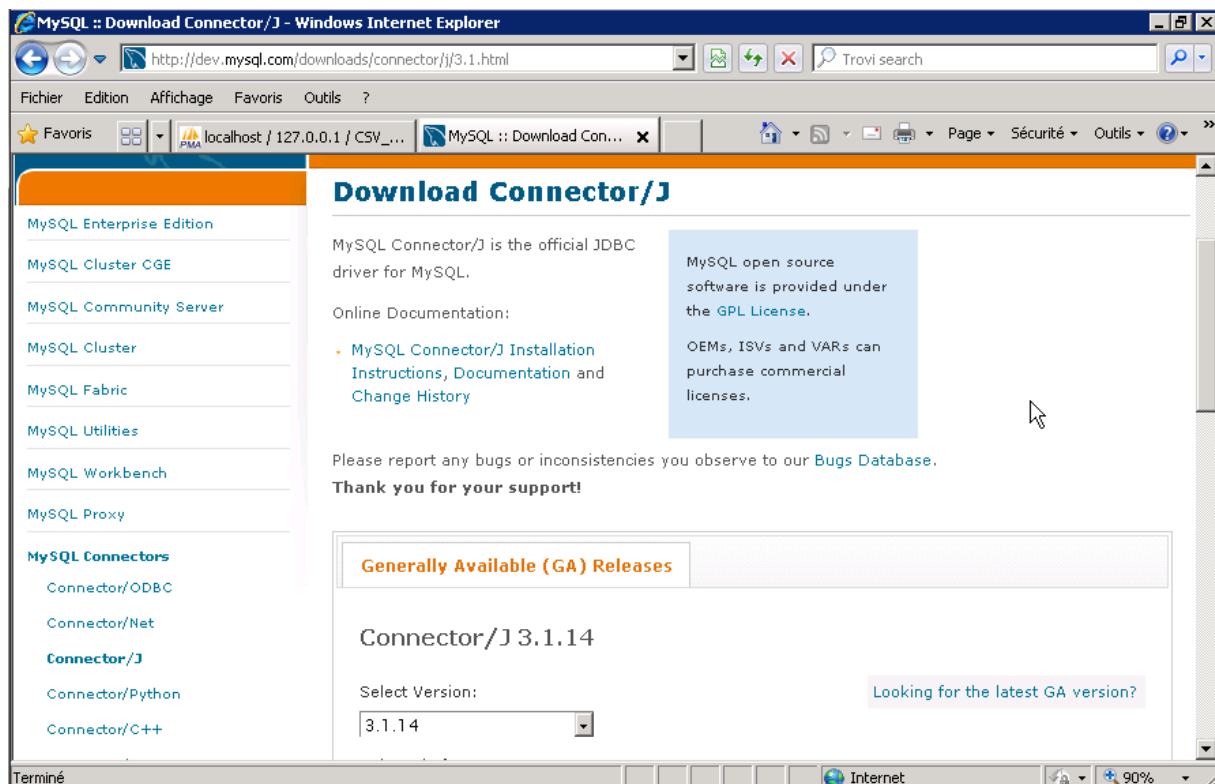


For the example we will import the *sales* table to MATLAB:

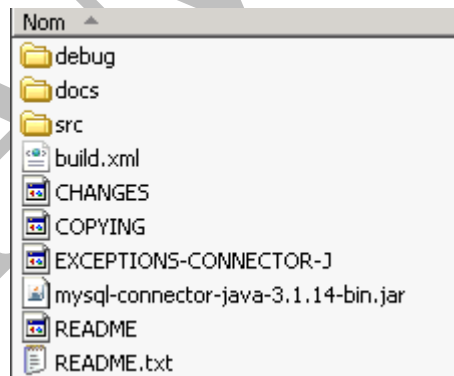


The first thing to do is to install the MySQL JDBC connector of the website:

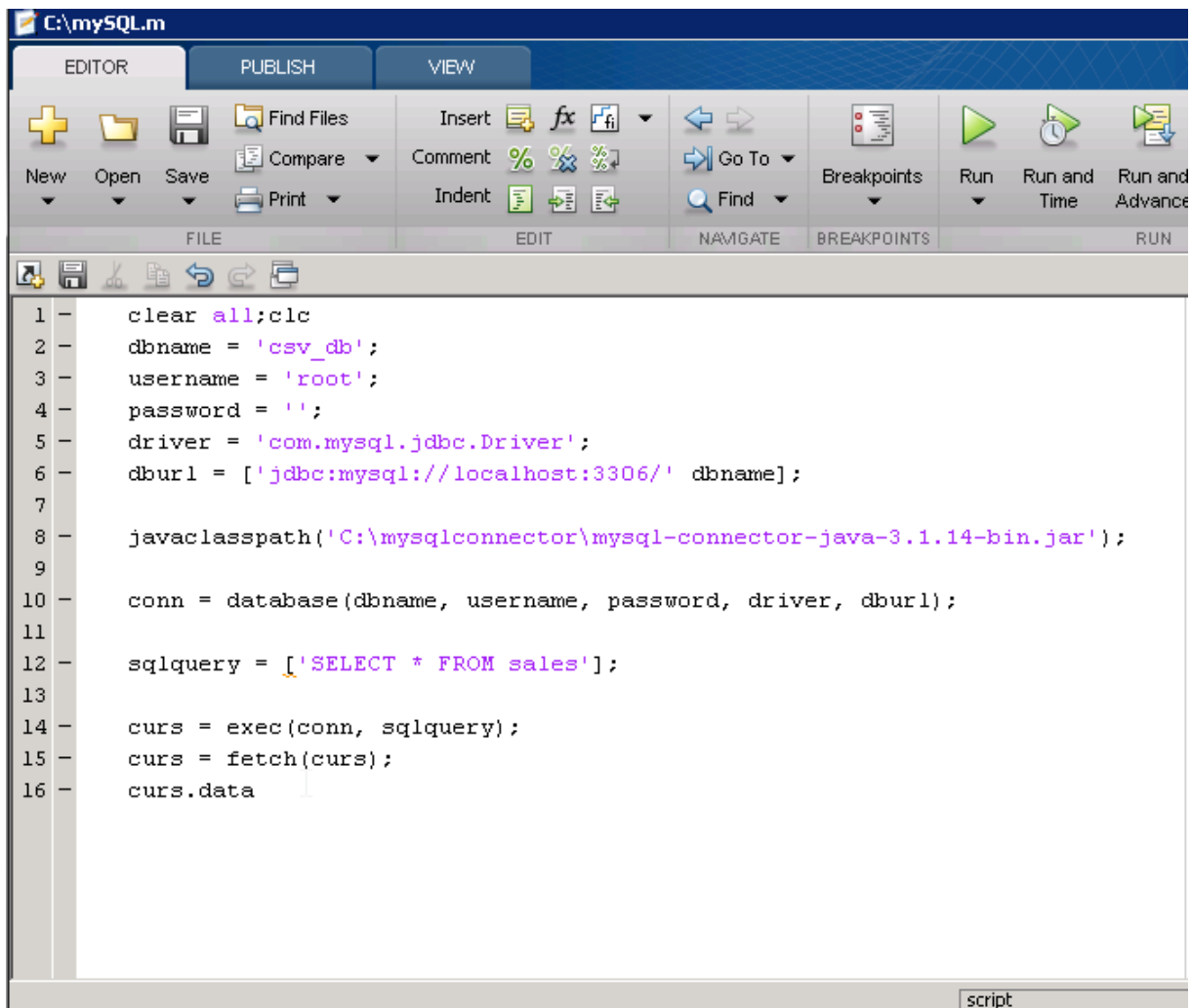
<http://dev.mysql.com/download/connector/j/3.1.html>



Once downloaded and the zip file extracted you get a folder with the following files and subfolders:

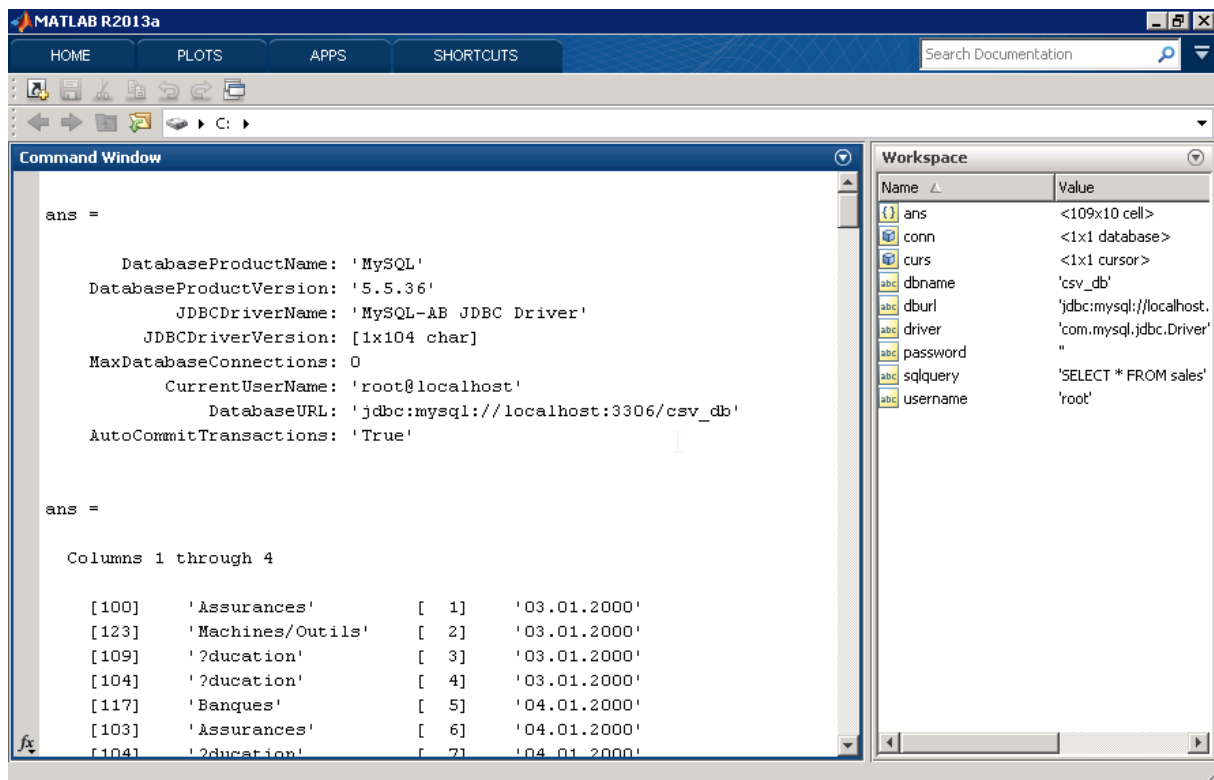


Once the connector unzipped we write the following script:



```
1 - clear all;clc
2 - dbname = 'csv_db';
3 - username = 'root';
4 - password = '';
5 - driver = 'com.mysql.jdbc.Driver';
6 - dburl = ['jdbc:mysql://localhost:3306/' dbname];
7
8 - javaclasspath('C:\mysqlconnector\mysql-connector-java-3.1.14-bin.jar');
9
10 - conn = database(dbname, username, password, driver, dburl);
11
12 - sqlquery = ['SELECT * FROM sales'];
13
14 - curs = exec(conn, sqlquery);
15 - curs = fetch(curs);
16 - curs.data
```

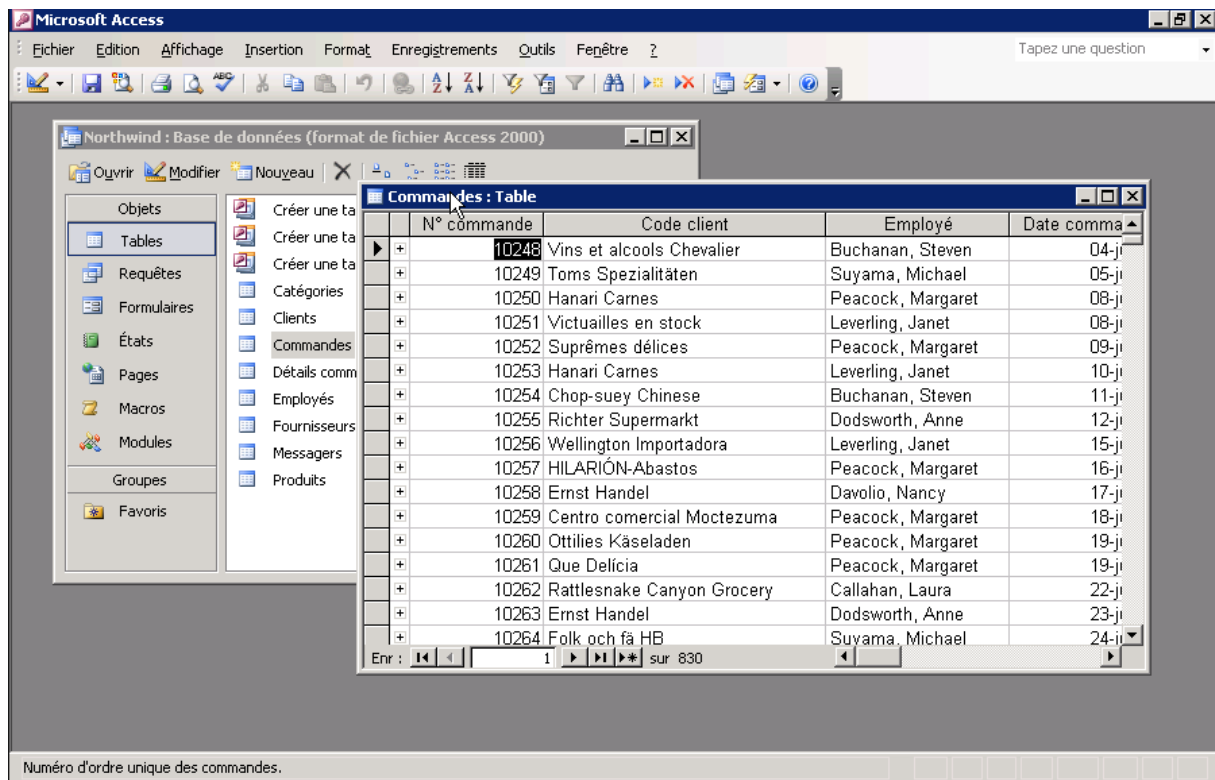
Once the code is executed the datas are imported in a cell array (**ToDo**: found a way to directly import the data into a dataset).



Connect to Microsoft Access 2003/2010 (*.mdb/*.accdb)

Retrieve data from Microsoft Access is something very common even Fortune 500 companies use Microsoft Access with MATLAB...

To start the first example let us see how to connect a *.mdb (Microsoft DataBase file) with MATLAB. Consider for this purpose the following Northwind.mdb file and especially the table named *Commandes*:



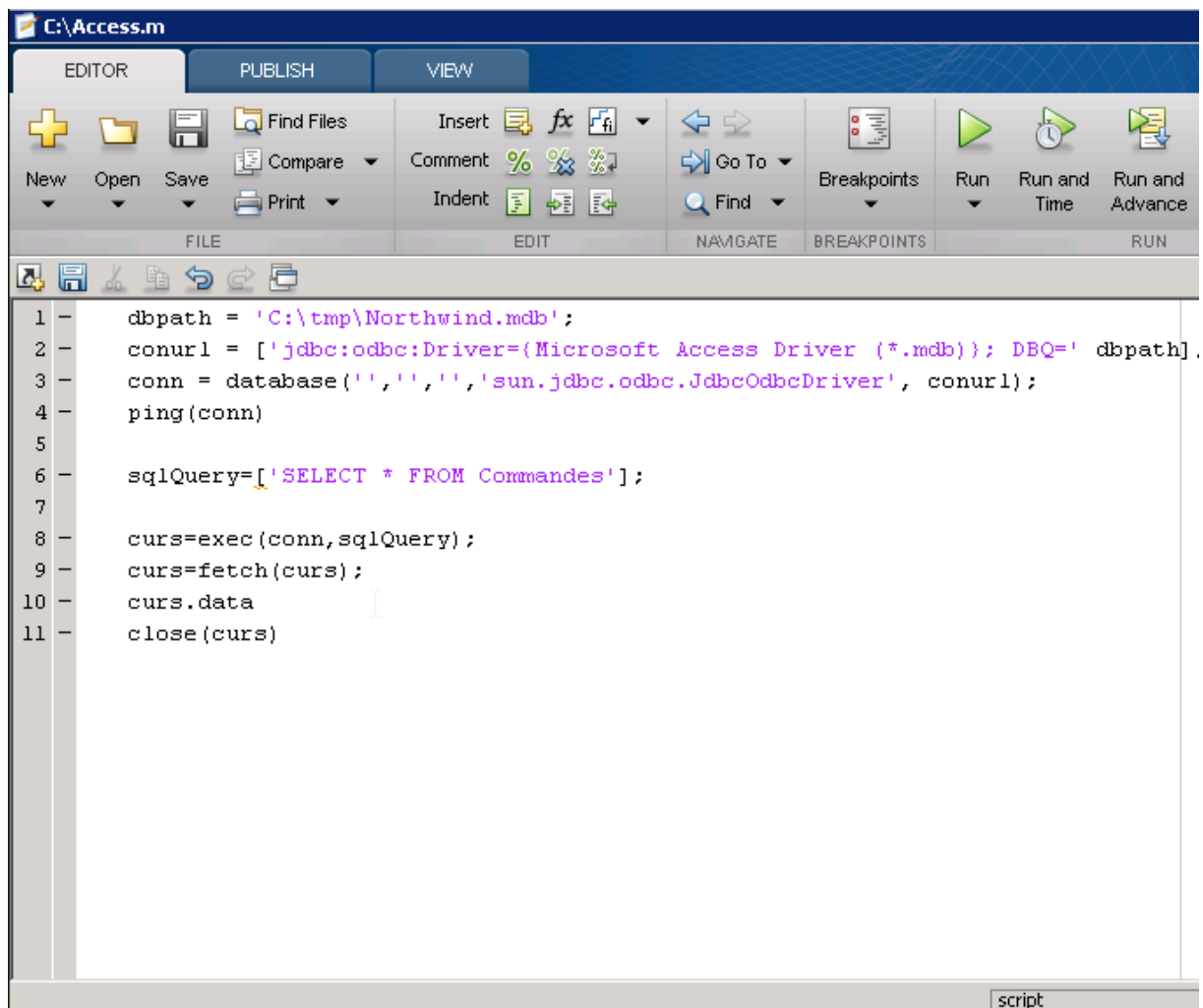
N° commande	Code client	Employé	Date commande
10248	Vins et alcools Chevalier	Buchanan, Steven	04-jr
10249	Toms Spezialitäten	Suyama, Michael	05-jr
10250	Hanari Carnes	Peacock, Margaret	08-jr
10251	Victuailles en stock	Leverling, Janet	08-jr
10252	Suprêmes délices	Peacock, Margaret	09-jr
10253	Hanari Carnes	Leverling, Janet	10-jr
10254	Chop-suey Chinese	Buchanan, Steven	11-jr
10255	Richter Supermarkt	Dodsworth, Anne	12-jr
10256	Wellington Importadora	Leverling, Janet	15-jr
10257	HILARIÓN-Abastos	Peacock, Margaret	16-jr
10258	Ernst Handel	Davolio, Nancy	17-jr
10259	Centro comercial Moctezuma	Peacock, Margaret	18-jr
10260	Ottilies Käseladen	Peacock, Margaret	19-jr
10261	Que Delícia	Peacock, Margaret	19-jr
10262	Rattlesnake Canyon Grocery	Callahan, Laura	22-jr
10263	Ernst Handel	Dodsworth, Anne	23-jr
10264	Folk och få HB	Suyama, Michael	24-jr

To retrieve data from such a table you need the following script but you must first be sure that you have at least the Java Runtime installed:

<https://www.java.com/fr/download/>

and the Microsoft Access ODBC Driver:

<http://www.microsoft.com/en-us/download/details.aspx?id=13255>



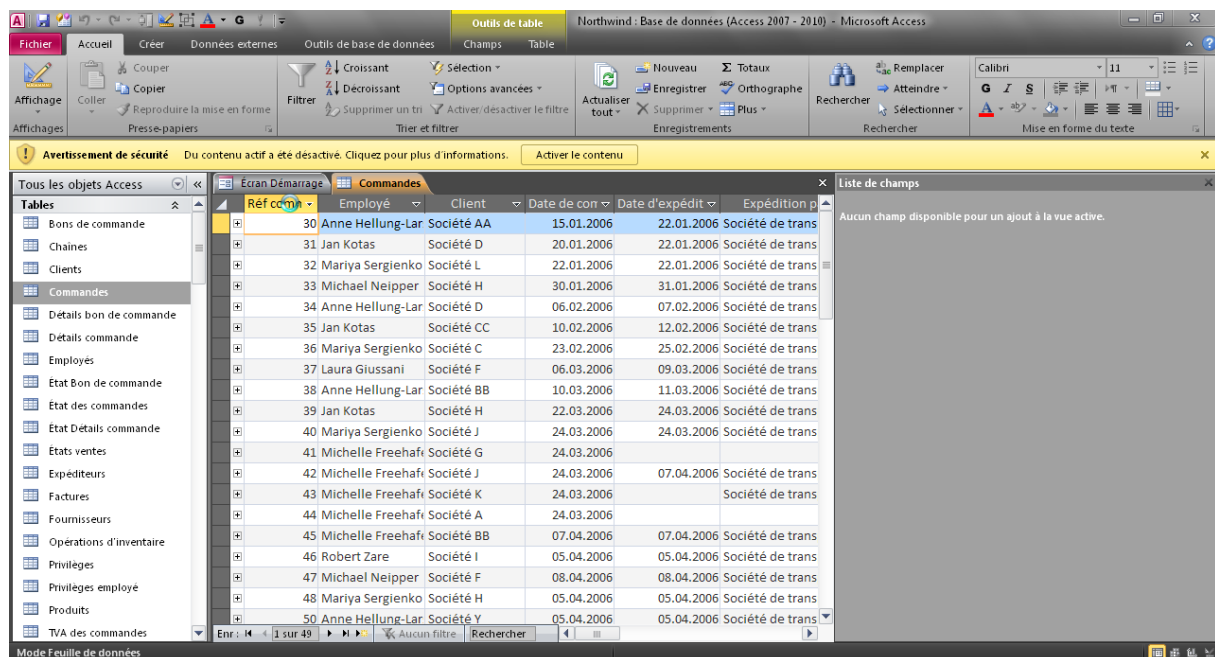
The image shows the MATLAB script editor window titled 'C:\Access.m'. The window has a menu bar with 'EDITOR', 'PUBLISH', and 'VIEW' tabs. Below the menu bar is a toolbar with icons for 'New', 'Open', 'Save', 'Find Files', 'Compare', 'Print', 'Insert', 'Comment', 'Indent', 'Go To', 'Find', 'Breakpoints', 'Run', 'Run and Time', and 'Run and Advance'. The script editor area contains the following MATLAB code:

```
1 - dbpath = 'C:\tmp\Northwind.mdb';
2 - conurl = ['jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)}; DBQ=' dbpath];
3 - conn = database('','','sun.jdbc.odbc.JdbcOdbcDriver', conurl);
4 - ping(conn)
5
6 - sqlQuery=['SELECT * FROM Commandes'];
7
8 - curs=exec(conn,sqlQuery);
9 - curs=fetch(curs);
10 - curs.data
11 - close(curs)
```

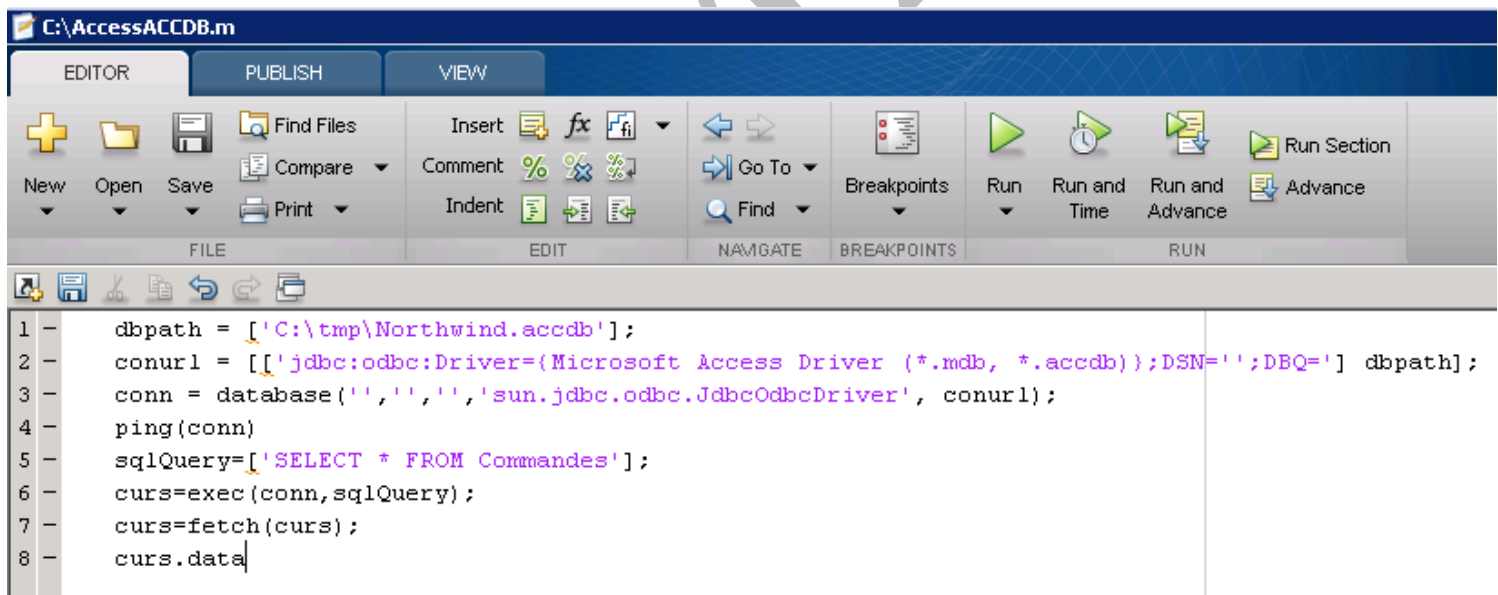
The script is saved as 'script' in the bottom right corner of the editor window.

The imported result will also be a cell array (**ToDo**: found a way to directly import the data into a dataset).

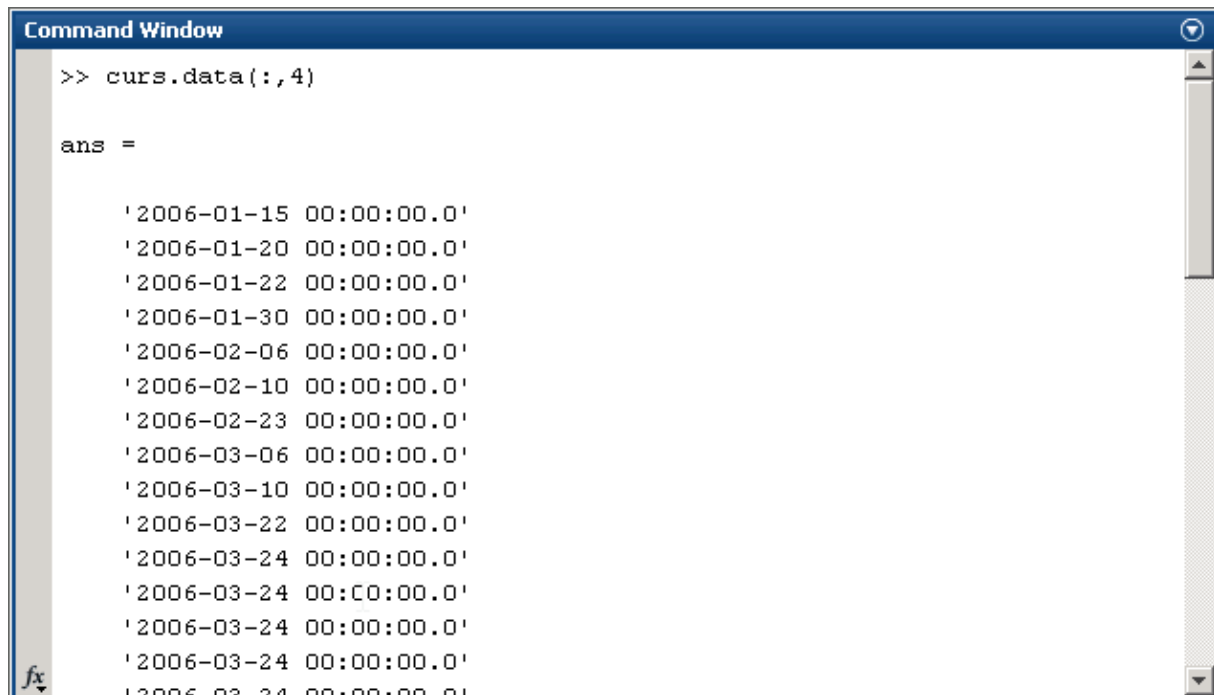
For *.accdb file (it can help to have ODBC driver installed and an up to date for Microsoft Access 2007/2010/2013 version and sometimes even having the MS Access runtime installed!) with the same table:



The following script will make the job (you must also have the Java Runtime and the Microsoft Access ODBC Driver installed!):



and the result will always be (**ToDo**: found a way to directly import the data into a dataset):



```
>> curs.data(:,4)

ans =

    '2006-01-15 00:00:00.0'
    '2006-01-20 00:00:00.0'
    '2006-01-22 00:00:00.0'
    '2006-01-30 00:00:00.0'
    '2006-02-06 00:00:00.0'
    '2006-02-10 00:00:00.0'
    '2006-02-23 00:00:00.0'
    '2006-03-06 00:00:00.0'
    '2006-03-10 00:00:00.0'
    '2006-03-22 00:00:00.0'
    '2006-03-24 00:00:00.0'
    '2006-03-24 00:C0:00.0'
    '2006-03-24 00:00:00.0'
    '2006-03-24 00:00:00.0'
    '2006-03-24 00:00:00.0'
```

Connect to Oracle Express 11g

Oracle is in my point of view the most powerful database worldwide (and also my preferred DBMS).

To see how to retrieve data from Oracle (Express) 11g we will use the native *DEMO_ORDERS* table:

The screenshot shows the Oracle Application Express (APEX) interface. The browser address bar displays the URL: 127.0.0.1:8080/apex/f?p=4500:1001:764133614418094::NO::: The page title is "ORACLE Application Express" and the user is logged in as "Welcome ISOZ (Logout)". The navigation menu includes "Home", "Application Builder", "SQL Workshop", "Team Development", and "Administration". The breadcrumb trail shows "Home > SQL Workshop > Object Browser". The "Schema" dropdown is set to "ISOZ".

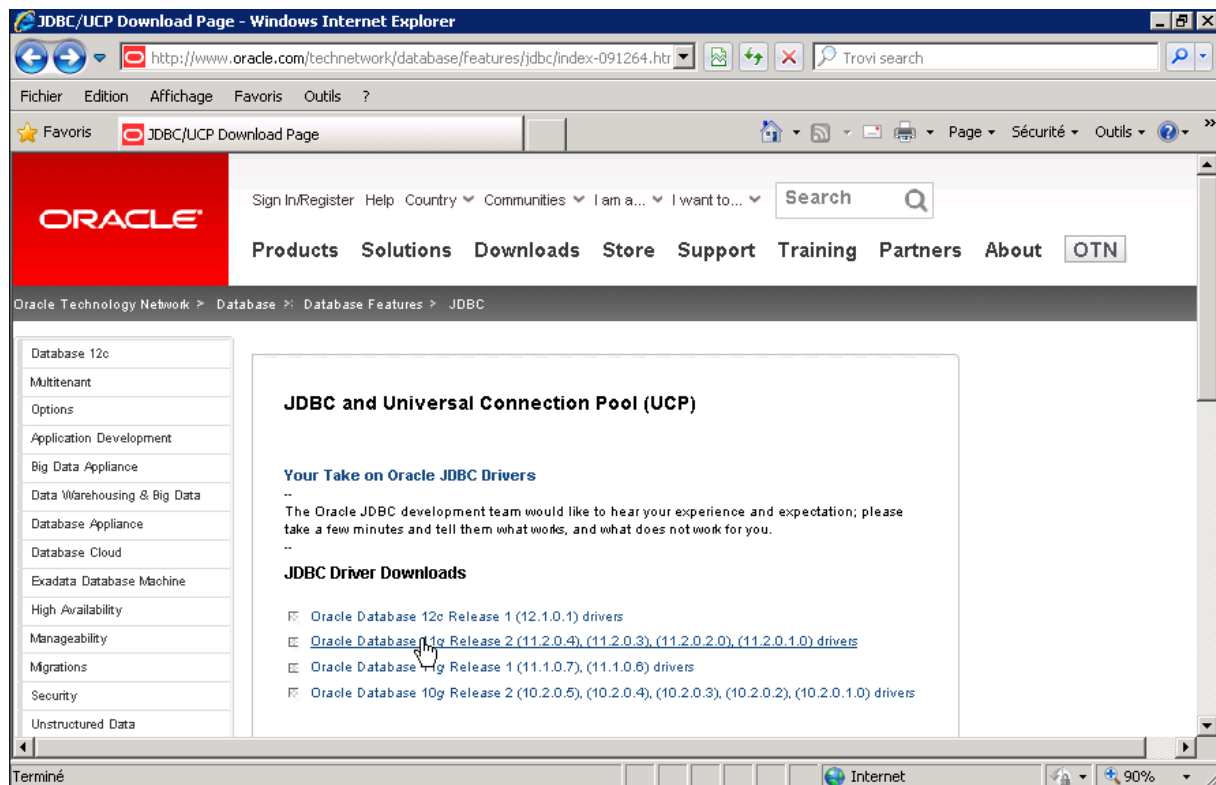
In the "Object Browser" pane on the left, the "Tables" tab is selected, and the "DEMO_ORDERS" table is highlighted. The main pane displays the table structure for "DEMO_ORDERS". The table has the following columns:

Column Name	Data Type	Nullable	Default	Primary Key
ORDER_ID	NUMBER	No	-	1
CUSTOMER_ID	NUMBER	No	-	-
ORDER_TOTAL	NUMBER(8,2)	Yes	-	-
ORDER_TIMESTAMP	DATE	Yes	-	-
USER_ID	NUMBER	Yes	-	-

Below the table structure, there is a "Download" link. The page also includes tabs for "Table", "Data", "Indexes", "Model", "Constraints", "Grants", "Statistics", "UI Defaults", "Triggers", "Dependencies", and "SQL".

First we will need to download the Oracle JDBC driver on the Oracle website:

<http://www.oracle.com/technetwork/database/features/jdbc/index-091264.html>

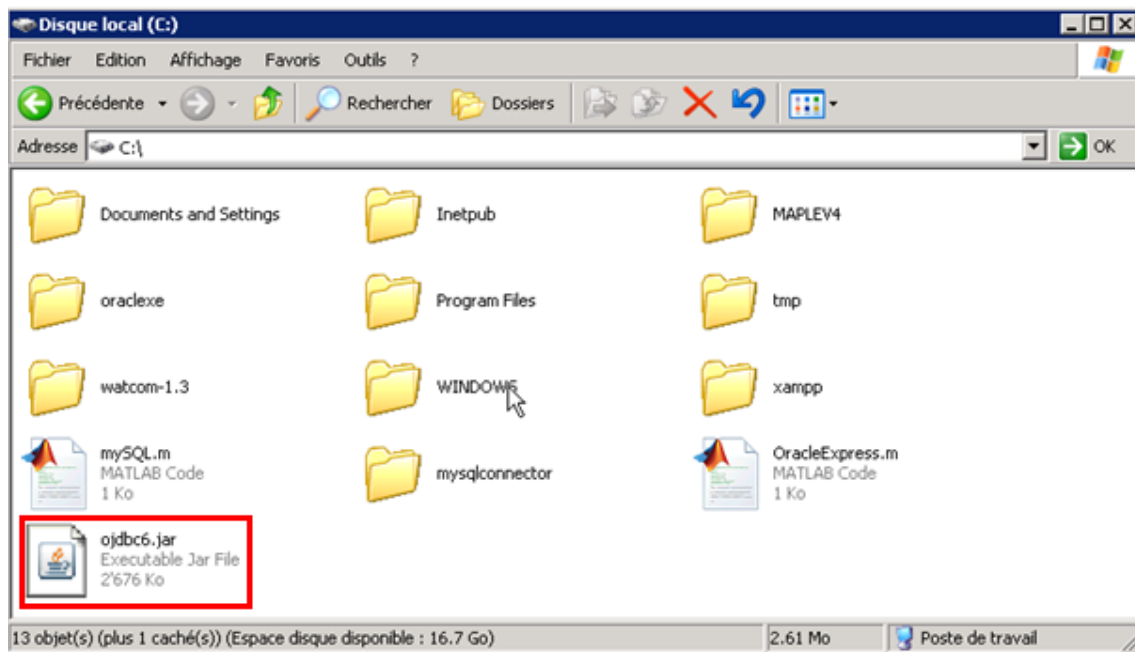


You click on the link **Oracle Database 11g Release 2 (11.2.0) drivers** as you can see above. After you get:

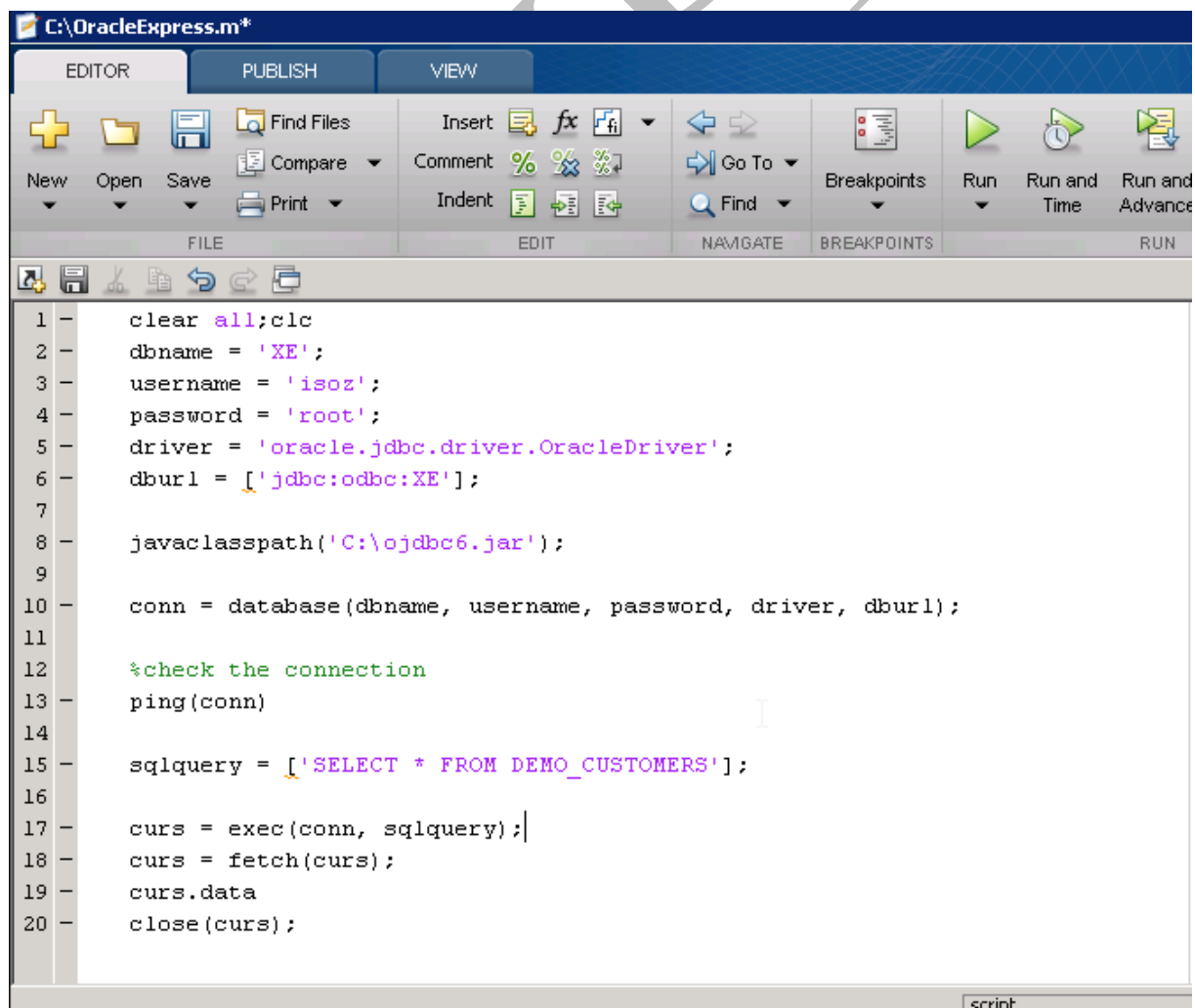


and you click on the **ojdbc6.jar** file to download.

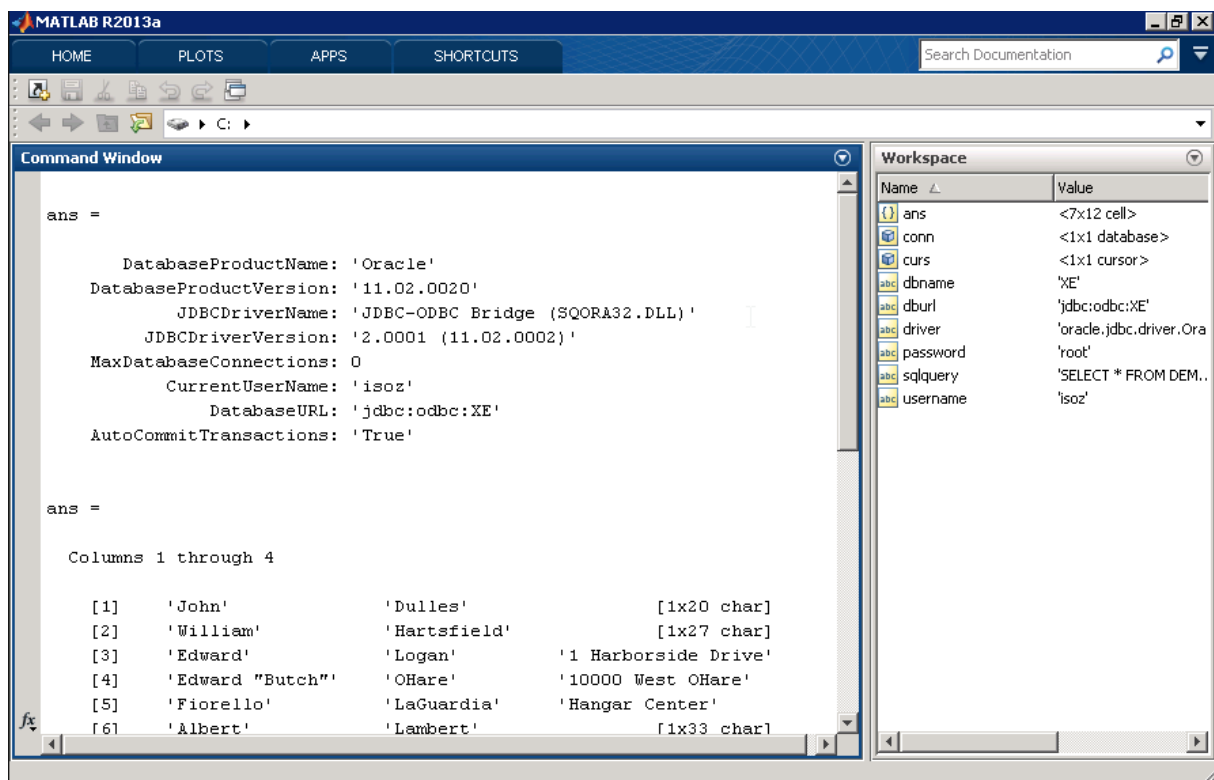
Once downloaded just put the file where you want:



and write the following script (**ToDo**: found a way to directly import the data into a dataset):



The result will also be cell array structure:



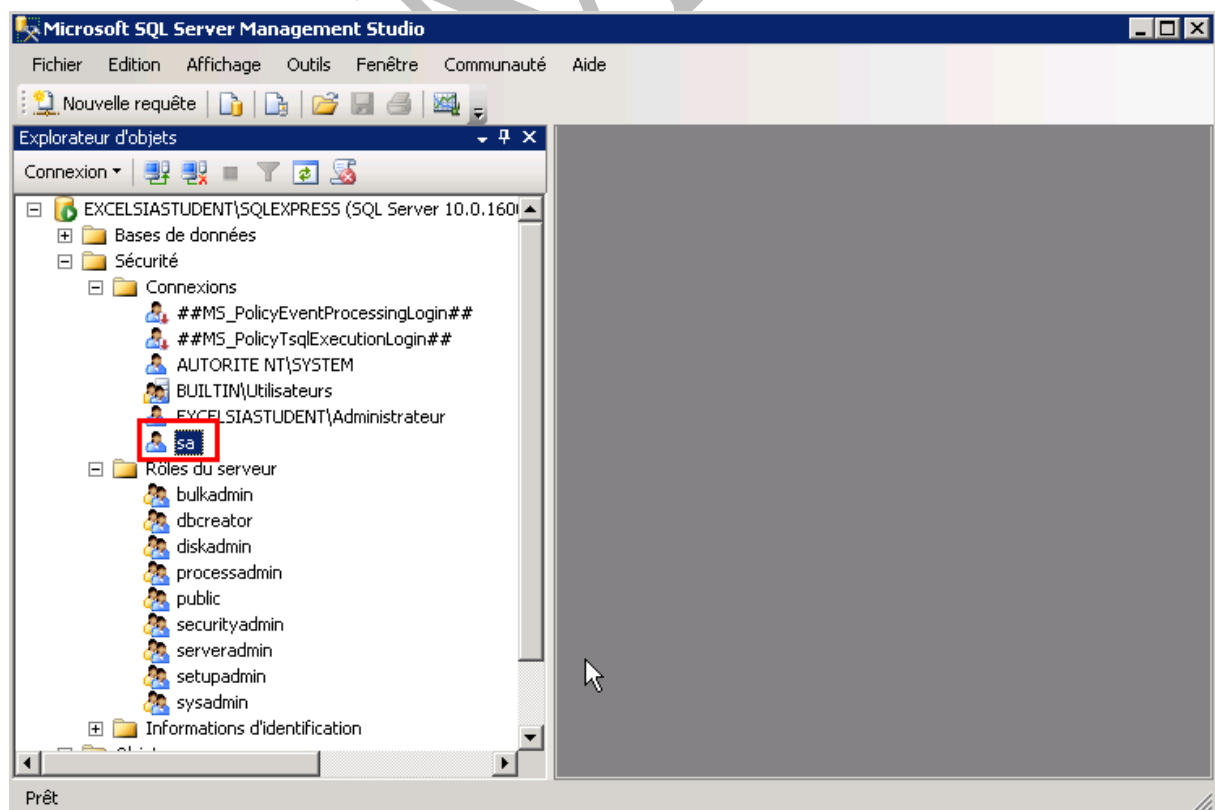
Connect to MS SQL Server Express 2008

Even if it is far away to be the best DBMS, Microsoft SQL Server (Express or not) is largely deployed across the world and even in financial institutions.

For this example we will use the default SQLEXPRES server name instance:



Below you can see the existing system administrator account inside of **SQL Server Express Management Studio** that we will have to use later:



For the example we will use the *Northwind* database and especially the table *dbo.Orders*:

The screenshot shows the Microsoft SQL Server Management Studio interface. The left pane displays the 'Explorateur d'objets' (Object Explorer) with the 'Northwind' database selected. The central pane shows a SQL query titled 'SQLQuery3.sql ...istrateur (52)' with the following text:

```

/***** Script de la commande SelectTopNRows à partir de SSMS *****/
SELECT TOP 1000 [OrderID]
, [CustomerID]
, [EmployeeID]
, [OrderDate]
, [RequiredDate]
, [ShippedDate]
, [ShipVia]
, [Freight]
, [ShipName]
, [ShipAddress]
, [ShipCity]

```

The bottom pane shows the 'Résultats' (Results) tab with a table of 830 rows. The first 7 rows are visible:

	OrderID	CustomerID	EmployeeID	OrderDate	RequiredDate	ShippedDate	ShipVia	Freight	ShipName
1	10248	VINET	5	1996-07-04 00:00:00.000	1996-08-01 00:00:00.000	1996-07-16 00:00:00.000	3	32.38	Vins et alcools Chevalier
2	10249	TOMSP	6	1996-07-05 00:00:00.000	1996-08-16 00:00:00.000	1996-07-10 00:00:00.000	1	11.61	Toms Spezialitäten
3	10250	HANAR	4	1996-07-08 00:00:00.000	1996-08-05 00:00:00.000	1996-07-12 00:00:00.000	2	65.83	Hanari Carnes
4	10251	VICTE	3	1996-07-08 00:00:00.000	1996-08-05 00:00:00.000	1996-07-15 00:00:00.000	1	41.34	Victuailles en stock
5	10252	SUPRD	4	1996-07-09 00:00:00.000	1996-08-06 00:00:00.000	1996-07-11 00:00:00.000	2	51.30	Suprêmes délices
6	10253	HANAR	3	1996-07-10 00:00:00.000	1996-07-24 00:00:00.000	1996-07-16 00:00:00.000	2	58.17	Hanari Carnes
7	10254	CHOPS	5	1996-07-11 00:00:00.000	1996-08-08 00:00:00.000	1996-07-23 00:00:00.000	2	22.98	Chou-sien Chinese

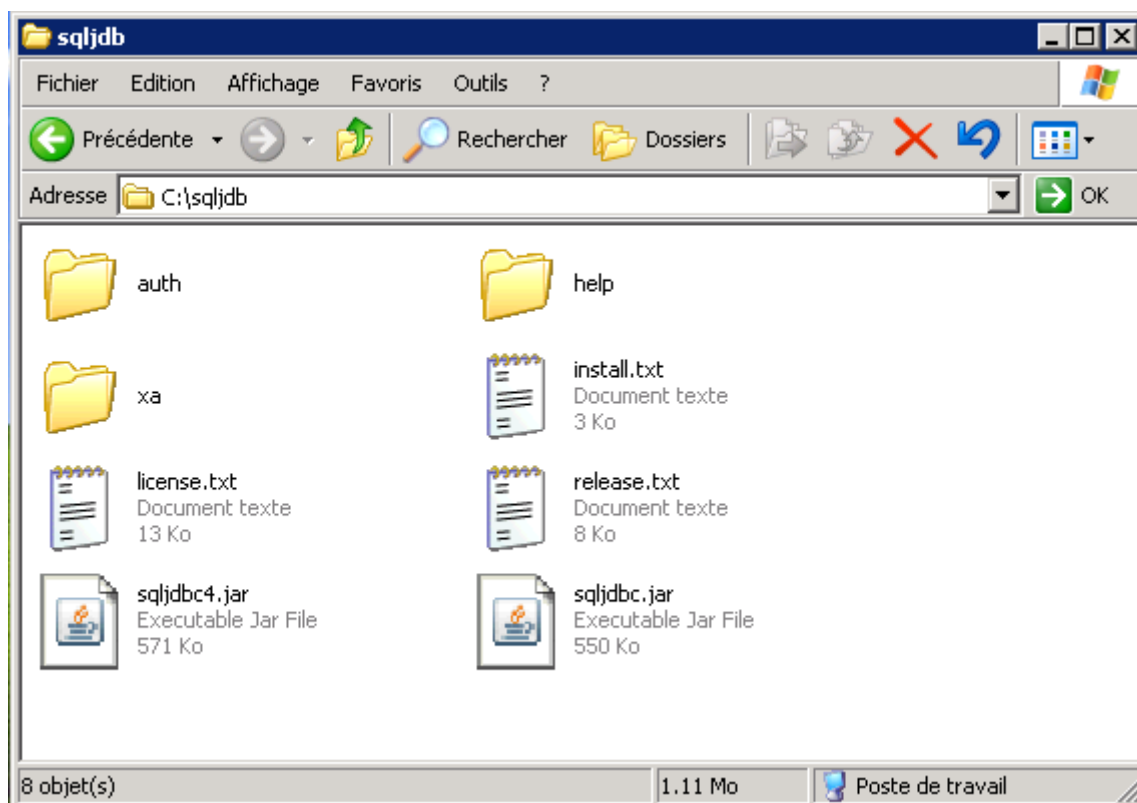
The status bar at the bottom indicates 'Exécution de requête réussie.' (Query execution successful).

After we need to download the JDBC 4.0 Microsoft SQL Server Driver:

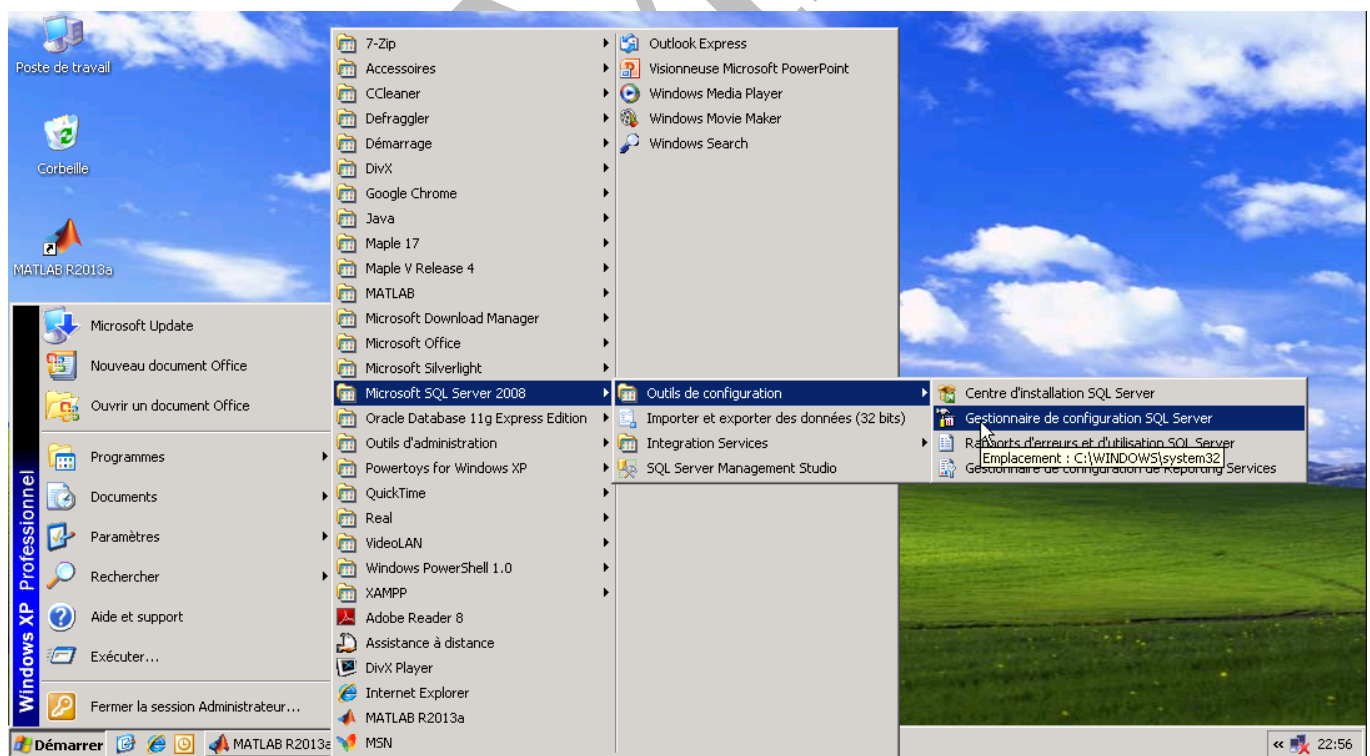
<http://www.microsoft.com/fr-fr/download/details.aspx?id=11774>

The screenshot shows the Microsoft Download Center page for the 'Pilote JDBC 4.0 Microsoft pour SQL Server'. The page is in French. It features a search bar, navigation links (Acheter, Produits, Catégories, Support, Sécurité), and a large download button labeled 'Télécharger'. Below the button, there is a section titled 'Mises à jour gratuites pour PC' (Free updates for PC) with a list of updates: 'Correctifs de sécurité' (Security updates), 'Mises à jour logicielles' (Software updates), and 'Service Packs'.

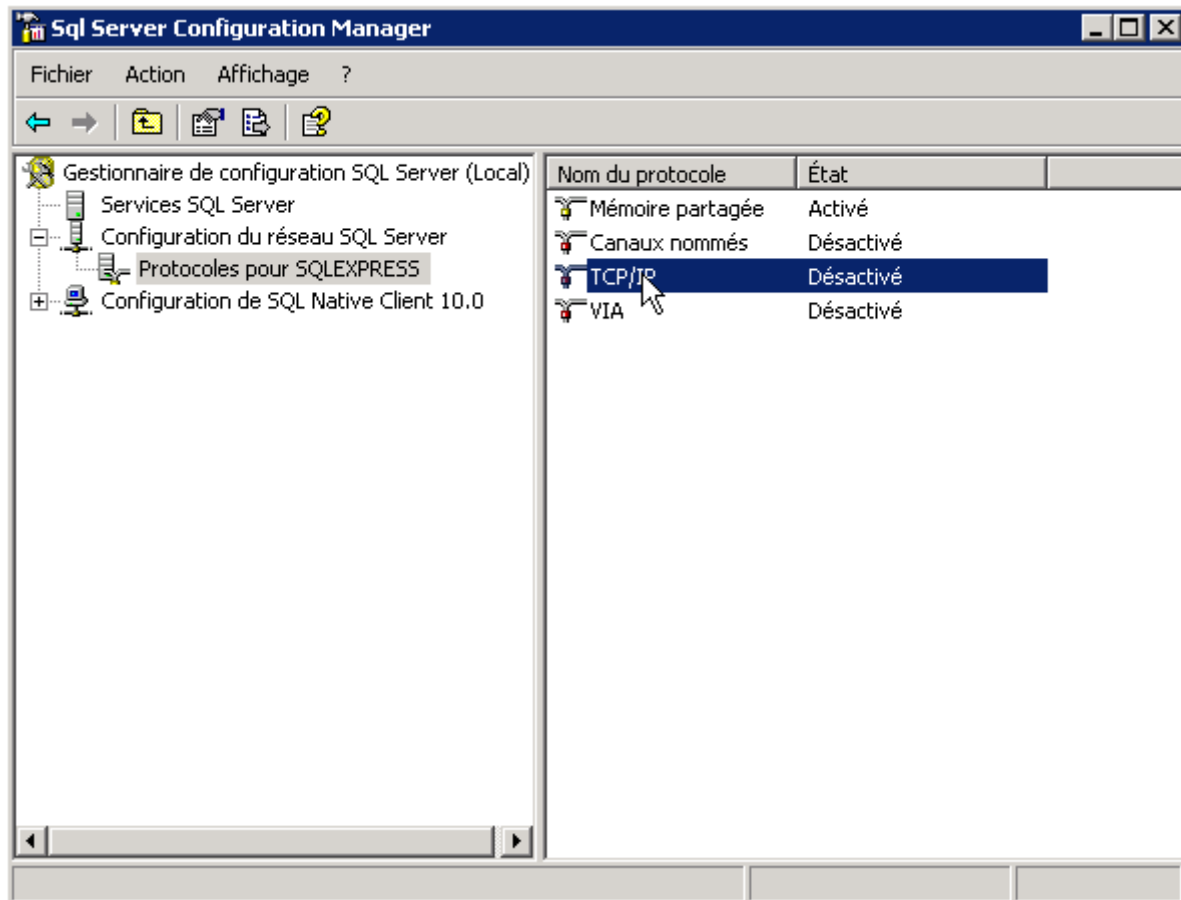
Once de file download and extracte you will get the following folder with a few various files:



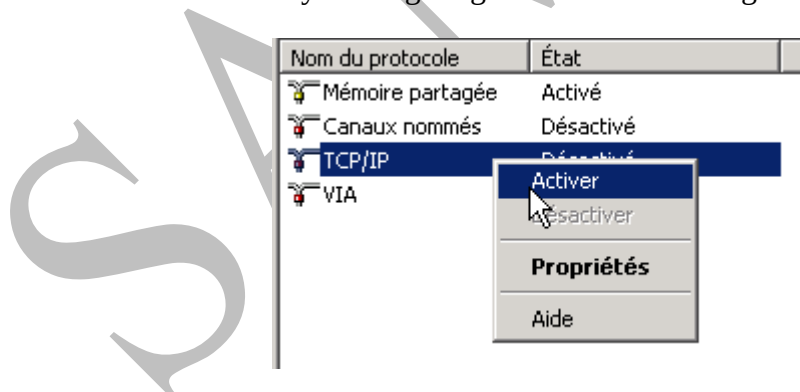
Now we open the **SQL Server Configuration Manager**:



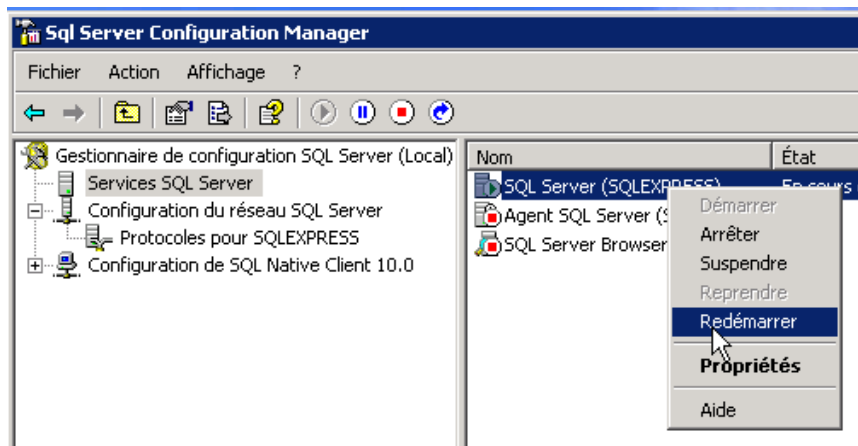
In the windows that will open we select the SQL **Protocoles** branch and on the right the **TCP/IP** protocol:



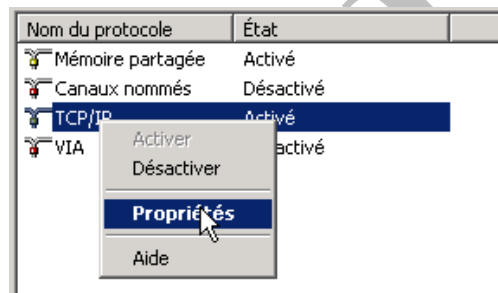
First we need to activate it by making a right click and selecting **Activate**:



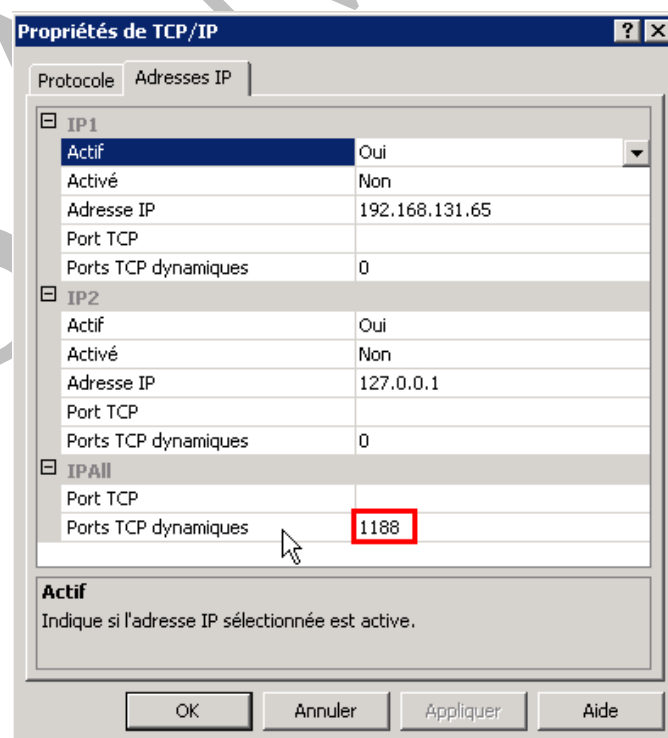
Once the TCP/IP is activated you must restart the SQL Server in the **Services** branch by right clicking and selecting **Restart**:



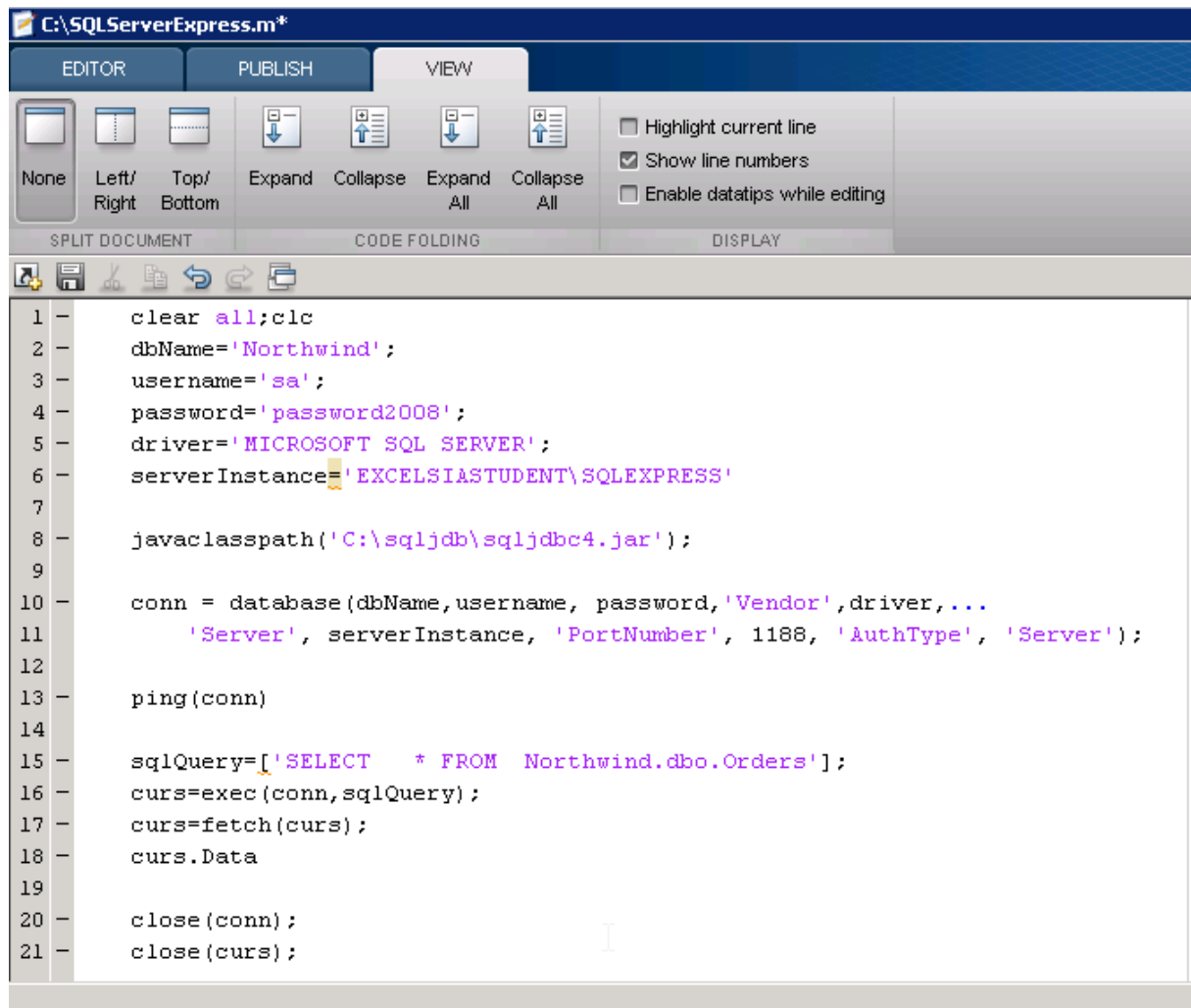
Once this is done we go back to the TCP/IP protocol and we open the **Properties**:



The following windows will open and it is important to take note of the **TCP dynamic port** or to write a given one if the field is empty:



Now we write the following script to retrieve the data (**ToDo**: found a way to directly import the data into a dataset):

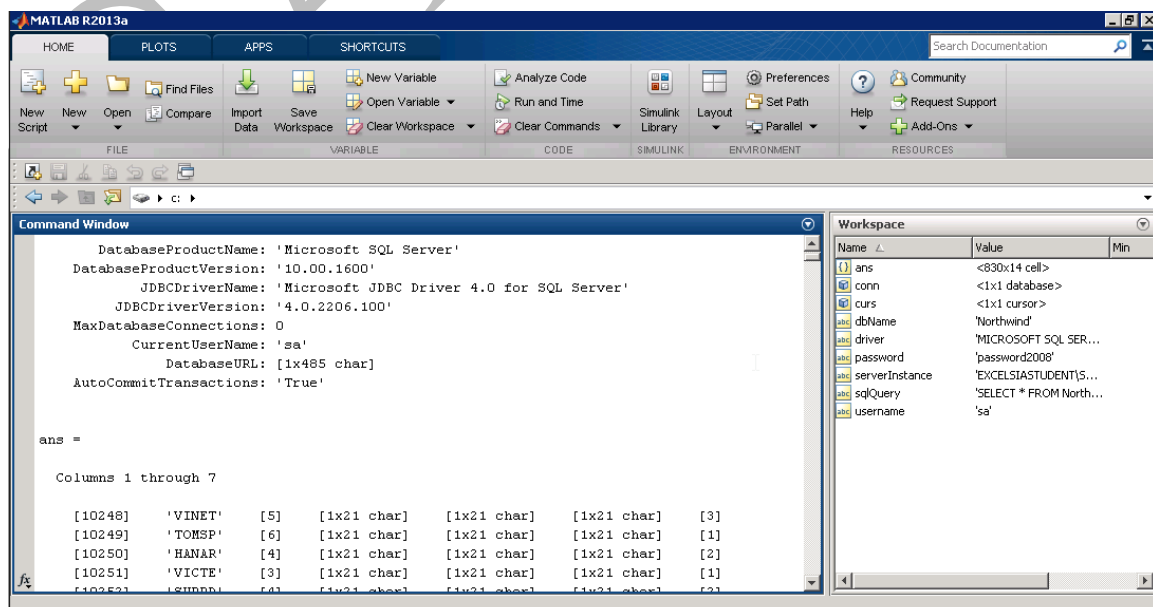


```

1 - clear all;clc
2 - dbName='Northwind';
3 - username='sa';
4 - password='password2008';
5 - driver='MICROSOFT SQL SERVER';
6 - serverInstance='EXCELSIASTUDENT\SQLEXPRESS'
7
8 - javaclasspath('C:\sqljdbc\sqljdbc4.jar');
9
10 - conn = database(dbName,username, password,'Vendor',driver,...
11 - 'Server', serverInstance, 'PortNumber', 1188, 'AuthType', 'Server');
12
13 - ping(conn)
14
15 - sqlQuery=['SELECT * FROM Northwind.dbo.Orders'];
16 - curs=exec(conn,sqlQuery);
17 - curs=fetch(curs);
18 - curs.Data
19
20 - close(conn);
21 - close(curs);

```

And the result will also be a cell array structure:



Command Window:

```

DatabaseProductName: 'Microsoft SQL Server'
DatabaseProductVersion: '10.00.1600'
JDBCDriverName: 'Microsoft JDBC Driver 4.0 for SQL Server'
JDBCDriverVersion: '4.0.2206.100'
MaxDatabaseConnections: 0
CurrentUserName: 'sa'
DatabaseURL: [1x485 char]
AutoCommitTransactions: 'True'

ans =

Columns 1 through 7

[10248] 'VINET' [5] [1x21 char] [1x21 char] [1x21 char] [3]
[10249] 'TOMSP' [6] [1x21 char] [1x21 char] [1x21 char] [1]
[10250] 'HANAR' [4] [1x21 char] [1x21 char] [1x21 char] [2]
[10251] 'VICTE' [3] [1x21 char] [1x21 char] [1x21 char] [1]
[10252] 'SUPDD' [4] [1x21 char] [1x21 char] [1x21 char] [2]

```

Workspace:

Name	Value	Min
ans	<830x14 cell>	
conn	<1x1 database>	
curs	<1x1 cursor>	
dbName	'Northwind'	
driver	'MICROSOFT SQL SER...	
password	'password2008'	
serverInstance	'EXCELSIASTUDENT(S...	
sqlQuery	'SELECT * FROM North...	
username	'sa'	

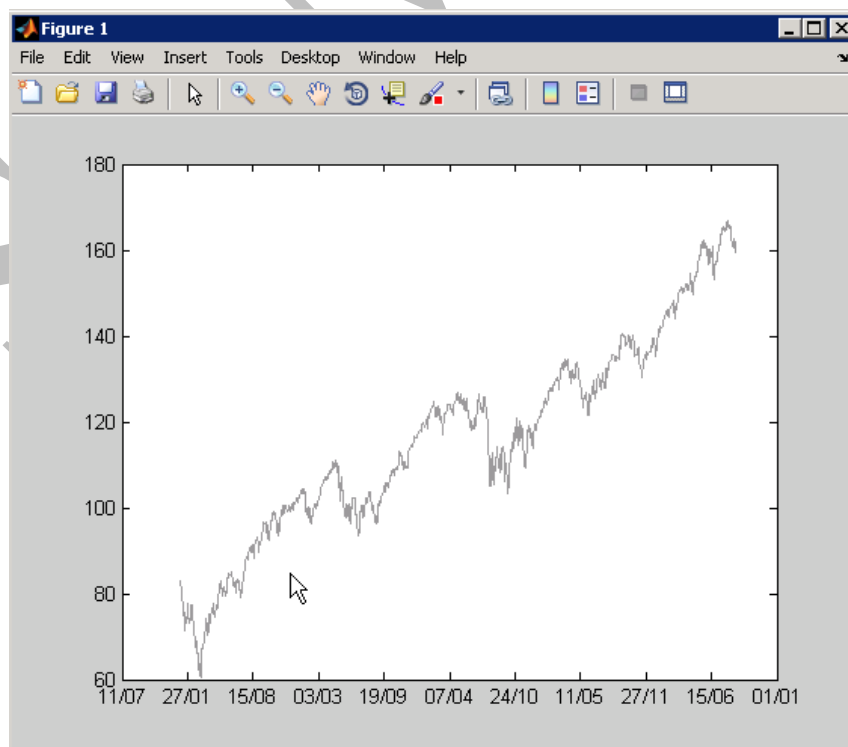
Connect to Yahoo! Finance

Here we use the `fetch()` function of the database 2013a toolbox to retrieve data directly from Yahoo! Finance into MATLAB and plot them:

```

1  close all; clear all; clc;
2
3  date_from=datetime('Jan 1 2009');
4  date_to=datetime('Aug 27 2013');
5
6  stock='SPY';
7
8  %we retrieve the close values (with the dates in the first column)
9  adjClose = fetch(yahoo,stock,'adj close',date_from,date_to);
10 %we retrieve the dividend values (with the dates in the first column)
11 div = fetch(yahoo,stock,date_from,date_to,'v');
12
13 %we calculate the returns
14 returns=(adjClose(2:end,2)./adjClose(1:end-1,2)-1);
15
16 % We plot adjusted close price of and mark days when dividends
17 % have been announced
18 plot(adjClose(:,1),adjClose(:,2),'color',[0.6 0.6 0.6])
19 datetick('x','dd/mm','keepticks');
```

The result gives (we already saw how to format dates in the chart chapter):



Statistics (with Statistical and Curve Fitting Toolbox)

<http://www.mathworks.ch/ch/help/stats/release-notes.html>

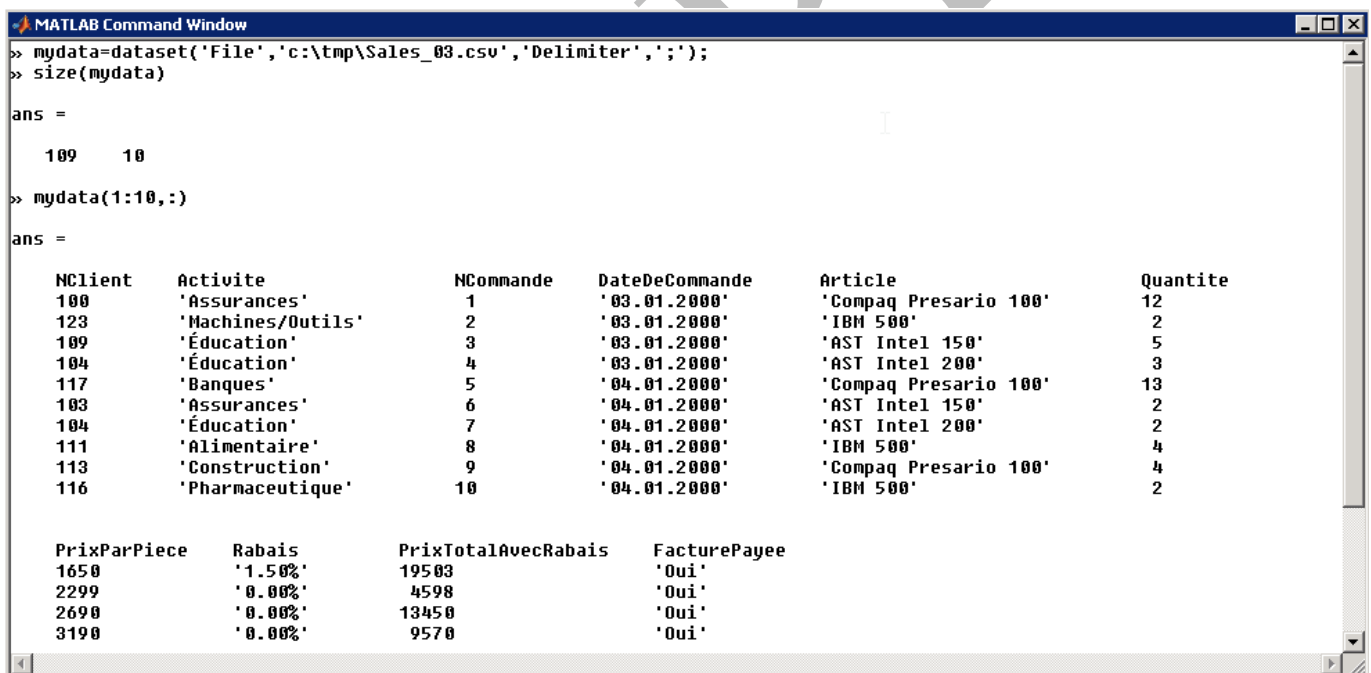
We will focus now on the Statistics Toolbox of MATLAB but only on the mathematical tools and theorems that we have studied and proved in details during the theoretical course on statistics/data mining and machine learning.

Datasets

The first thing to know with the Statistical Toolbox is that you have the possibility to import or to convert external data to a structure named "dataset" that is similar in many points to the dataframes in R and can manage both: strings and numerical values (also including dates).

Import/Export *.csv files as data set (data frame)

Let us first see how to import directly as dataset using the function `dataset()` data from a *.csv file:



```

MATLAB Command Window
>> mydata=dataset('File','c:\tmp\Sales_03.csv','Delimiter',';');
>> size(mydata)

ans =

    109    10

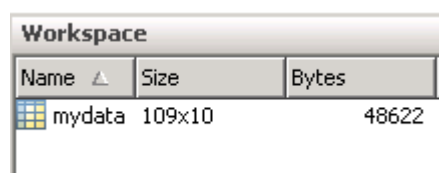
>> mydata(1:10,:)

ans =

    NClient    Activite    NCommande    DateDeCommande    Article    Quantite
    100      'Assurances'         1      '03.01.2000'      'Compaq Presario 100'         12
    123      'Machines/Outils'       2      '03.01.2000'      'IBH 500'                     2
    109      'Education'             3      '03.01.2000'      'AST Intel 150'                5
    104      'Education'             4      '03.01.2000'      'AST Intel 200'                3
    117      'Banques'               5      '04.01.2000'      'Compaq Presario 100'        13
    103      'Assurances'             6      '04.01.2000'      'AST Intel 150'                2
    104      'Education'             7      '04.01.2000'      'AST Intel 200'                2
    111      'Alimentaire'            8      '04.01.2000'      'IBH 500'                     4
    113      'Construction'           9      '04.01.2000'      'Compaq Presario 100'         4
    116      'Pharmaceutique'        10      '04.01.2000'      'IBH 500'                     2

    PrixParPiece    Rabais    PrixTotalAvecRabais    FacturePayee
    1650            '1.50%'        19503                'Oui'
    2299            '0.00%'         4598                'Oui'
    2690            '0.00%'        13450                'Oui'
    3190            '0.00%'         9570                'Oui'
  
```

And here this is how such a structure looks like in the variables workspace:



If you double click you can easily edit all data:

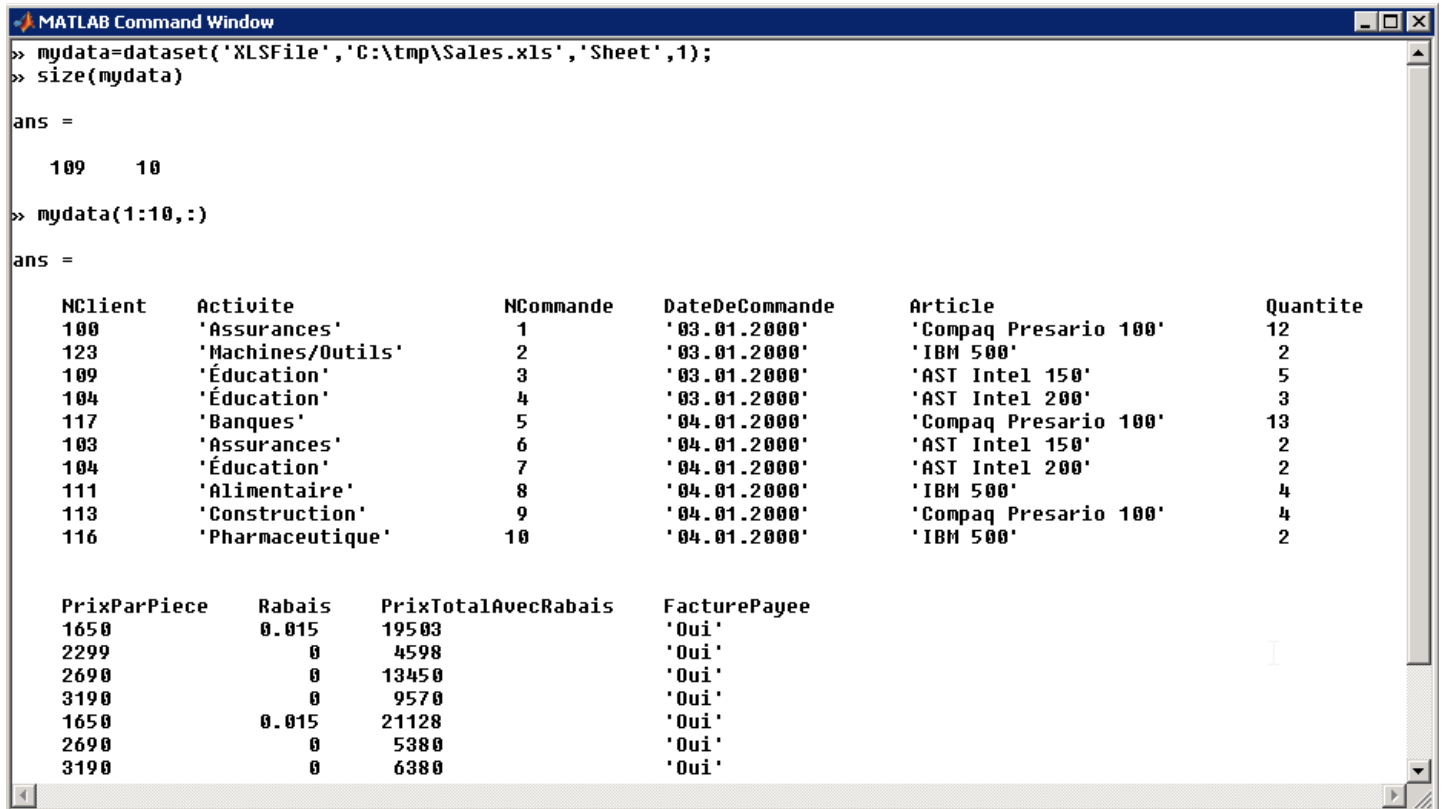
	1	2	3	4	5	6	7	8	9	10
	NClient	Activite	NCommande	DateDeCommande	Article	Quantite	PrixParPiece	Rabais	PrixTotalAvecRabais	FacturePayee
1	100	'Assurances'	1	'03.01.2000'	'Compaq Pre...	12	1650	'1.50%'	19503	'Oui'
2	123	'Machines/O...	2	'03.01.2000'	'IBM 500'	2	2299	'0.00%'	4598	'Oui'
3	109	'Éducation'	3	'03.01.2000'	'AST Intel 150'	5	2690	'0.00%'	13450	'Oui'
4	104	'Éducation'	4	'03.01.2000'	'AST Intel 200'	3	3190	'0.00%'	9570	'Oui'
5	117	'Banques'	5	'04.01.2000'	'Compaq Pre...	13	1650	'1.50%'	2.1128e+04	'Oui'
6	103	'Assurances'	6	'04.01.2000'	'AST Intel 150'	2	2690	'0.00%'	5380	'Oui'
7	104	'Éducation'	7	'04.01.2000'	'AST Intel 200'	2	3190	'0.00%'	6380	'Oui'
8	111	'Alimentaire'	8	'04.01.2000'	'IBM 500'	4	2299	'0.00%'	9196	'Oui'
9	113	'Construction'	9	'04.01.2000'	'Compaq Pre...	4	1650	'0.00%'	6600	'Oui'
10	116	'Pharmaceut...	10	'04.01.2000'	'IBM 500'	2	2299	'0.00%'	4598	'Oui'
11	110	'Distribution'	11	'05.01.2000'	'AST Intel 200'	6	3190	'1.50%'	1.8853e+04	'Oui'
12	112	'Machines/O...	12	'05.01.2000'	'Compaq Pre...	6	1650	'1.50%'	9.7515e+03	'Oui'
13	123	'Machines/O...	13	'05.01.2000'	'IBM 500'	6	2299	'1.50%'	1.3587e+04	'Oui'
14	113	'Construction'	14	'05.01.2000'	'AST Intel 150'	3	2690	'0.00%'	8070	'Oui'
15	115	'Distribution'	15	'05.01.2000'	'Compaq Pre...	8	1650	'1.50%'	13002	'Oui'
16	124	'Éducation'	16	'05.01.2000'	'AST Intel 200'	8	3190	'1.50%'	2.5137e+04	'Oui'

To export a dataset to a *.csv file you just have to use the function **export()**:

```
>> export(mydata, 'File', 'c:\Sales.csv', 'Delimiter', ';', 'WriteVarNames', true)
>>
```

Import *.xls/*.xlsx files as data set (data frame)

Let us now see how to import directly as dataset using the function **dataset()** data from as *.xls or *.xlsx file (Microsoft Excel):



```

MATLAB Command Window
>> mydata=dataset('XLSFile','C:\tmp\Sales.xls','Sheet',1);
>> size(mydata)

ans =

    109    10

>> mydata(1:10,:)

ans =

    NClient    Activite    NCommande    DateDeCommande    Article    Quantite
    100      'Assurances'         1    '03.01.2000'    'Compaq Presario 100'    12
    123      'Machines/Outils'     2    '03.01.2000'    'IBM 500'                2
    109      'Éducation'           3    '03.01.2000'    'AST Intel 150'          5
    104      'Éducation'           4    '03.01.2000'    'AST Intel 200'          3
    117      'Banques'              5    '04.01.2000'    'Compaq Presario 100'    13
    103      'Assurances'           6    '04.01.2000'    'AST Intel 150'          2
    104      'Éducation'           7    '04.01.2000'    'AST Intel 200'          2
    111      'Alimentaire'          8    '04.01.2000'    'IBM 500'                4
    113      'Construction'         9    '04.01.2000'    'Compaq Presario 100'    4
    116      'Pharmaceutique'      10    '04.01.2000'    'IBM 500'                2

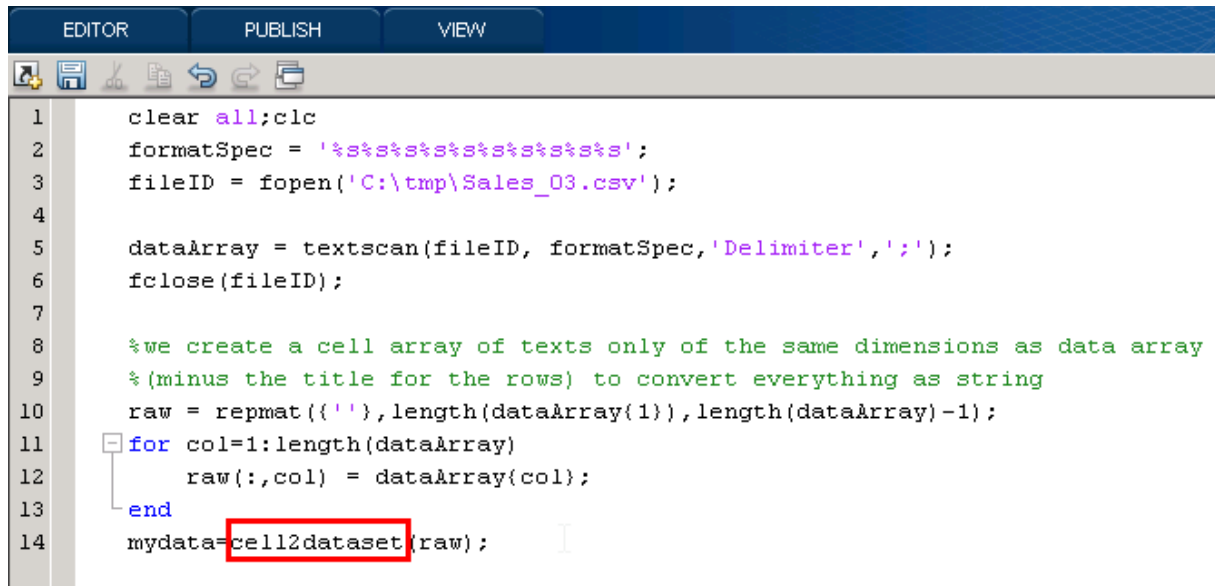
    PrixParPiece    Rabais    PrixTotalAvecRabais    FacturePayee
    1650            0.015    19503                'Oui'
    2299             0        4598                'Oui'
    2690             0       13450                'Oui'
    3190             0       9570                 'Oui'
    1650            0.015    21128                'Oui'
    2690             0       5380                 'Oui'
    3190             0       6380                 'Oui'
  
```

This works well!

Convert structure array to data sets

Sometimes (especially for databases) you can not (as far as I know) directly import as dataset structure. Then it is useful to know how to convert a cell array to a dataset.

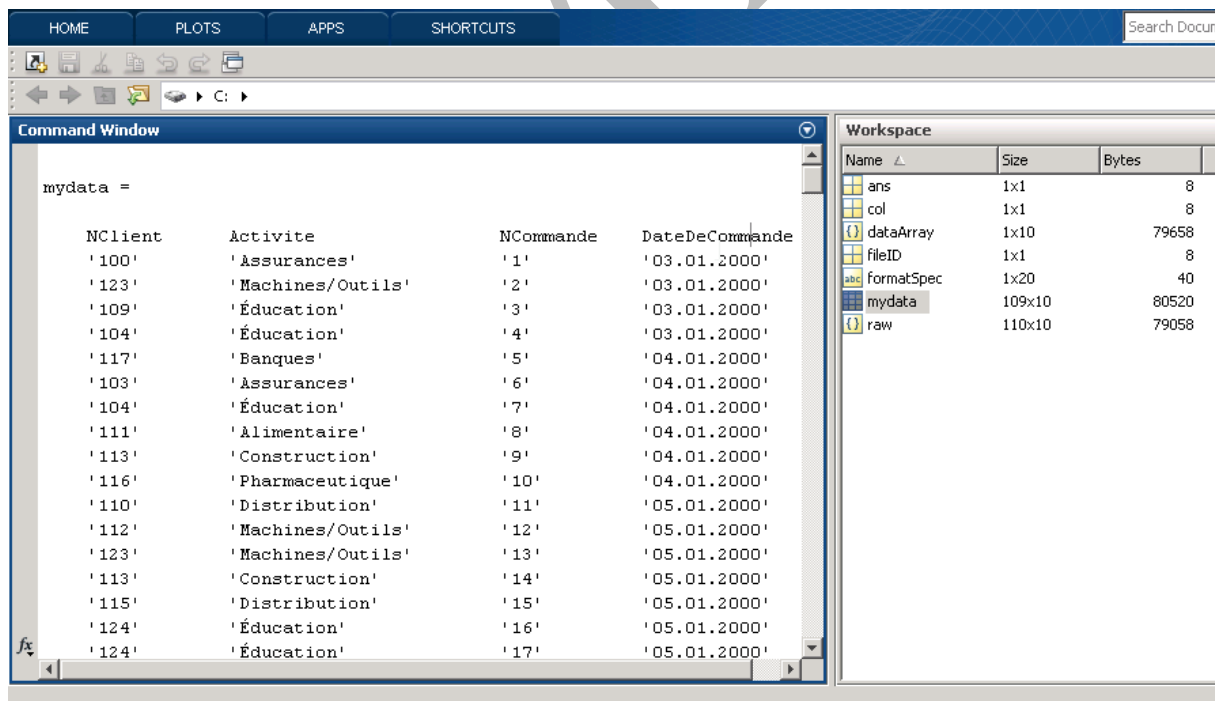
For this purpose we need to loop across the columns (**ToDo**: find an easier way to do that):



```

1 clear all;clc
2 formatSpec = '%s%s%s%s%s%s%s%s';
3 fileID = fopen('C:\tmp\Sales_03.csv');
4
5 dataArray = textscan(fileID, formatSpec, 'Delimiter', ';');
6 fclose(fileID);
7
8 %we create a cell array of texts only of the same dimensions as data array
9 % (minus the title for the rows) to convert everything as string
10 raw = repmat({' '}, length(dataArray{1}), length(dataArray)-1);
11 for col=1:length(dataArray)
12     raw(:,col) = dataArray{col};
13 end
14 mydata=cell2dataset(raw);
  
```

The result will be almost as expected!:



Command Window

```

mydata =

    NClient    Activite    NCommande    DateDeCommande
    '100'      'Assurances'    '1'        '03.01.2000'
    '123'      'Machines/Outils'    '2'        '03.01.2000'
    '109'      'Éducation'    '3'        '03.01.2000'
    '104'      'Éducation'    '4'        '03.01.2000'
    '117'      'Banques'    '5'        '04.01.2000'
    '103'      'Assurances'    '6'        '04.01.2000'
    '104'      'Éducation'    '7'        '04.01.2000'
    '111'      'Alimentaire'    '8'        '04.01.2000'
    '113'      'Construction'    '9'        '04.01.2000'
    '116'      'Pharmaceutique'    '10'       '04.01.2000'
    '110'      'Distribution'    '11'       '05.01.2000'
    '112'      'Machines/Outils'    '12'       '05.01.2000'
    '123'      'Machines/Outils'    '13'       '05.01.2000'
    '113'      'Construction'    '14'       '05.01.2000'
    '115'      'Distribution'    '15'       '05.01.2000'
    '124'      'Éducation'    '16'       '05.01.2000'
    '124'      'Éducation'    '17'       '05.01.2000'
  
```

Workspace

Name	Size	Bytes
ans	1x1	8
col	1x1	8
dataArray	1x10	79658
fileID	1x1	8
formatSpec	1x20	40
mydata	109x10	80520
raw	110x10	79058

But we still have to convert the numeric string values to real pure numeric values with the function `str2double()`:

Command Window

```
>> mydata.NClient = str2double(mydata.NClient);
>> mydata(1:10,1:5)
```

ans =

NClient	Activite	NCommande	DateDeCommande	Article
100	'Assurances'	'1'	'03.01.2000'	'Compaq Presar
123	'Machines/Outils'	'2'	'03.01.2000'	'IBM 500'
109	'Éducation'	'3'	' '	'AST Intel 150
104	'Éducation'	'4'	'03.01.2000'	'AST Intel 200
117	'Banques'	'5'	'04.01.2000'	'Compaq Presar
103	'Assurances'	'6'	'04.01.2000'	'AST Intel 150
104	'Éducation'	'7'	'04.01.2000'	'AST Intel 200
111	'Alimentaire'	'8'	'04.01.2000'	'IBM 500'
113	'Construction'	'9'	'04.01.2000'	'Compaq Presar
116	'Pharmaceutique'	'10'	'04.01.2000'	'IBM 500'

Then you can do as script of this:

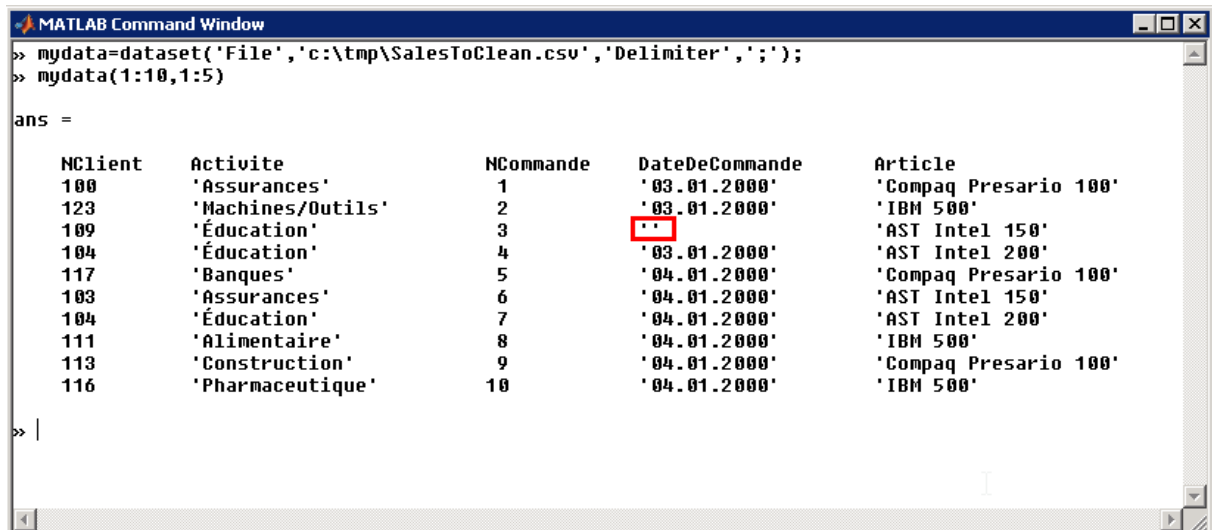
EDITOR PUBLISH VIEW

```
1 clear all;clc
2 formatSpec='%s%s%s%s%s%s%s%s';
3 fileID=fopen('C:\tmp\Sales_03.csv');
4 dataArray=textscan(fileID,formatSpec,'Delimiter',';');
5 fclose(fileID);
6 raw= repmat({' '},length(dataArray{1}),length(dataArray)-1);
7 for col=1:length(dataArray)
8     raw(:,col)=dataArray{col};
9 end
10 mydata=cell2dataset(raw);
11 bol_dataset=ismissing(mydata);
12 cleanData=mydata(~any(bol_dataset,2),:);
13 cleanData.Day=day(datenum(cleanData.DateDeCommande,'dd.mm.yyyy'));
14 cleanData.Quantite=str2double(cleanData.Quantite);
15 clearvars -except cleanData;
```

Clean messy or missing data in datasets

Cleanings messing or missing datas is a major topic in finance and there are a lot of technics for this job. One of the most basic one is just to remove the problematic rows.

For the example consider the following dataset:



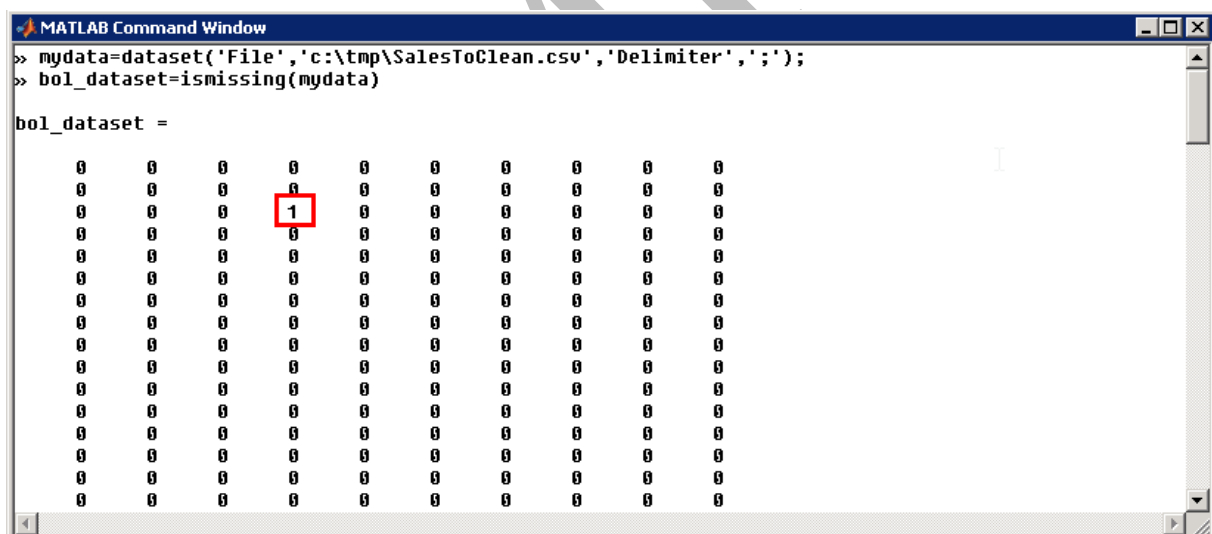
```

MATLAB Command Window
>> mydata=dataset('File','c:\tmp\SalesToClean.csv','Delimiter',';');
>> mydata(1:10,1:5)

ans =

    NClient    Activite    NCommande    DateDeCommande    Article
    100        'Assurances'        1          '03.01.2000'      'Compaq Presario 100'
    123        'Machines/Outils'    2          '03.01.2000'      'IBM 500'
    109        'Éducation'          3          '03.01.2000'      'AST Intel 150'
    104        'Éducation'          4          '03.01.2000'      'AST Intel 200'
    117        'Banques'            5          '04.01.2000'      'Compaq Presario 100'
    103        'Assurances'        6          '04.01.2000'      'AST Intel 150'
    104        'Éducation'          7          '04.01.2000'      'AST Intel 200'
    111        'Alimentaire'        8          '04.01.2000'      'IBM 500'
    113        'Construction'       9          '04.01.2000'      'Compaq Presario 100'
    116        'Pharmaceutique'    10         '04.01.2000'      'IBM 500'
  
```

To remove the missing data rows we will first use the `ismissing()` function:



```

MATLAB Command Window
>> mydata=dataset('File','c:\tmp\SalesToClean.csv','Delimiter',';');
>> bol_dataset=ismissing(mydata)

bol_dataset =

     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     1     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
  
```


And after we use the **any()** function to extract only problematic rows or to eliminate problematic rows:

```

>> messyData=mydata(any(bol_dataset,2),:);

messyData =

    NClient    Activite    NCommande    DateDeCommande    Article    Quantite
    109        'Éducation'        3        ..        'AST Intel 150'        5

    PrixParPiece    Rabais    PrixTotalAvecRabais    FacturePayee
    2690        '0.00%'        13450        'Oui'

>> cleanData=mydata(~any(bol_dataset,2),:);
>> cleanData(1:10,1:5)

ans =

    NClient    Activite    NCommande    DateDeCommande    Article
    100        'Assurances'        1    '03.01.2000'    'Compaq Presario 100'
    123        'Machines/Outils'        2    '03.01.2000'    'IBM 500'
    104        'Éducation'        4    '03.01.2000'    'AST Intel 200'
    117        'Banques'        5    '04.01.2000'    'Compaq Presario 100'
    103        'Assurances'        6    '04.01.2000'    'AST Intel 150'
    104        'Éducation'        7    '04.01.2000'    'AST Intel 200'
    111        'Alimentaire'        8    '04.01.2000'    'IBM 500'
    113        'Construction'        9    '04.01.2000'    'Compaq Presario 100'
    116        'Pharmaceutique'        10    '04.01.2000'    'IBM 500'
    110        'Distribution'        11    '05.01.2000'    'AST Intel 200'
  
```

One another typical cleaning process is to reformat dates. Here is the beginning of such an example:

```

>> cleanData.Month=month(datetime(cleanData.DateDeCommande,'dd.mm.yyyy'));
>> cleanData(1:3,:)

ans =

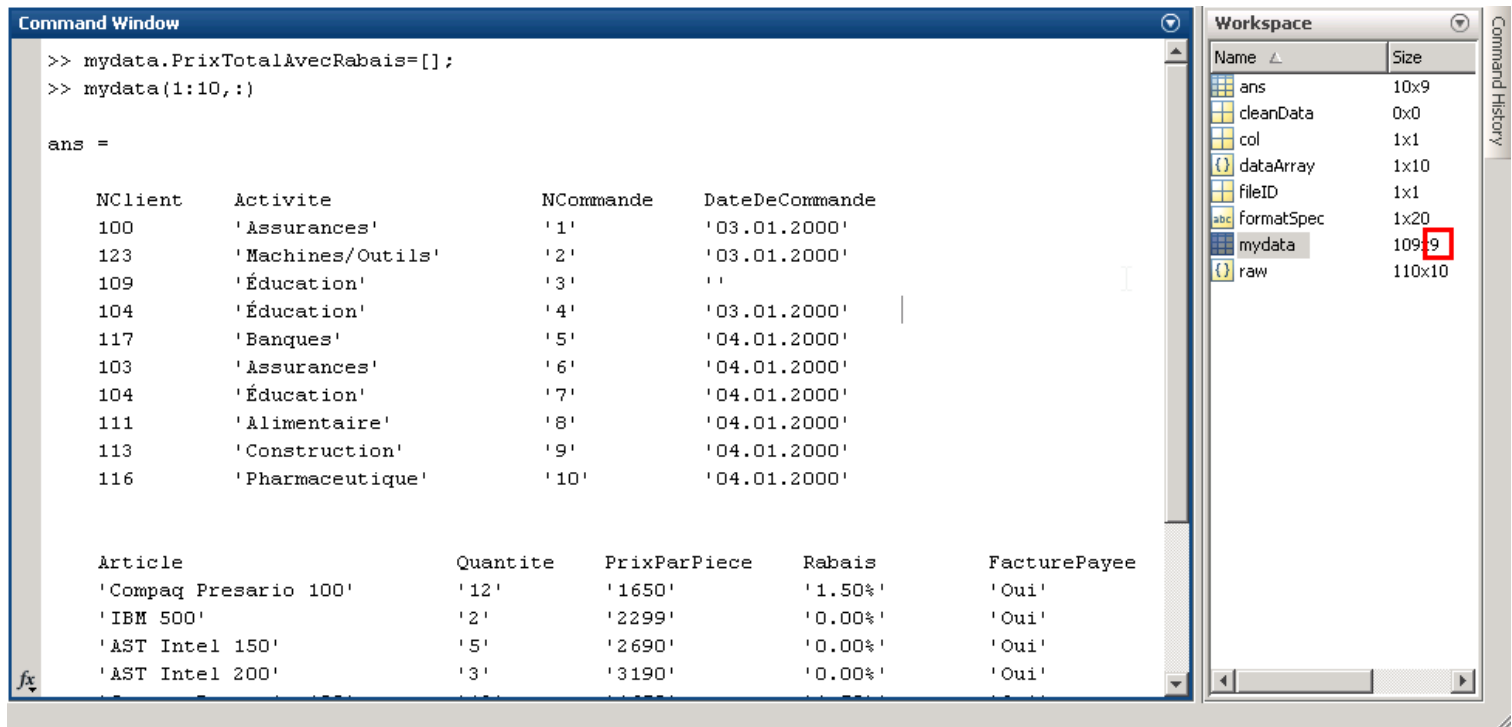
    NClient    Activite    NCommande    DateDeCommande
    '100'        'Assurances'        '1'    '03.01.2000'
    '123'        'Machines/Outils'        '2'    '03.01.2000'
    '104'        'Éducation'        '4'    '03.01.2000'

    Article    Quantite    PrixParPiece    Rabais
    'Compaq Presario 100'    '12'    '1650'    '1.50%'
    'IBM 500'    '2'    '2299'    '0.00%'
    'AST Intel 200'    '3'    '3190'    '0.00%'

    PrixTotalAvecRabais    FacturePayee    Month
    '19503'    'Oui'    1
    '4598'    'Oui'    1
    '9570'    'Oui'    1
  
```

Select/Remove column (variable)

One very common question about dataset is how it is possible to remove on column (variable). For this consider we want to remove the column *PrixTotalAvecRabais*. The syntax will be the following:



The screenshot shows the MATLAB Command Window and Workspace. In the Command Window, the following commands are entered:

```
>> mydata.PrixTotalAvecRabais=[];
>> mydata(1:10,:)
```

The output shows the first 10 rows of the dataset, with columns NClient, Activite, NCommande, and DateDeCommande. The variable `ans` is assigned the result of the command.

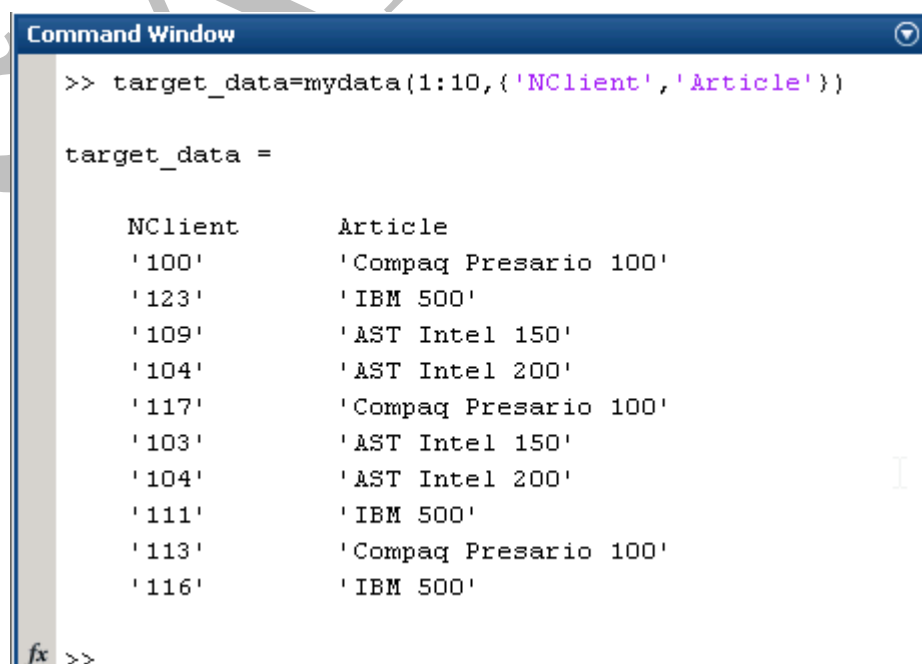
The Workspace window on the right shows the following variables:

Name	Size
ans	10x9
cleanData	0x0
col	1x1
dataArray	1x10
fileID	1x1
formatSpec	1x20
mydata	109x9
raw	110x10

The `mydata` variable is highlighted in the workspace, and its size is 109x9, indicating that the column `PrixTotalAvecRabais` has been successfully removed.

Create subsets and filter

Another very common question is how to select a subset of rows and columns of a dataset. For this the syntax is simply the following:



The screenshot shows the MATLAB Command Window with the following command:

```
>> target_data=mydata(1:10,{'NClient','Article'})
```

The output shows the first 10 rows of the dataset, with columns NClient and Article. The variable `target_data` is assigned the result of the command.

```
target_data =
```

NClient	Article
'100'	'Compaq Presario 100'
'123'	'IBM 500'
'109'	'AST Intel 150'
'104'	'AST Intel 200'
'117'	'Compaq Presario 100'
'103'	'AST Intel 150'
'104'	'AST Intel 200'
'111'	'IBM 500'
'113'	'Compaq Presario 100'
'116'	'IBM 500'

Or if you need to create a subset through a filter, the syntax is the following for numerical values:

```
Command Window
>> target_data=mydata(mydata.Quantite==10,:)

target_data =

    NClient    Activite    NCommande    DateCommande
    '118'      'Éducation'    '61'        '18.01.2000'
    '101'      'Construction' '76'        '21.01.2000'

    Article    Quantite    PrixParPiece    Rabais
    'IBM 500'   10         2299            '1.50%'
    'AST Intel 200' 10         3190            '1.50%'

    FacturePayee    PrixTotal
    'Oui'           22990
    'Non'           31900

fx >> |
```

For strings this is a little bit more complicated (ToDo: found if there is not a more simple way to do this job):

```
Command Window
>> mydata(find(strncmp(mydata.Activite, 'Banques', length('Banques'))),3:end)

ans =

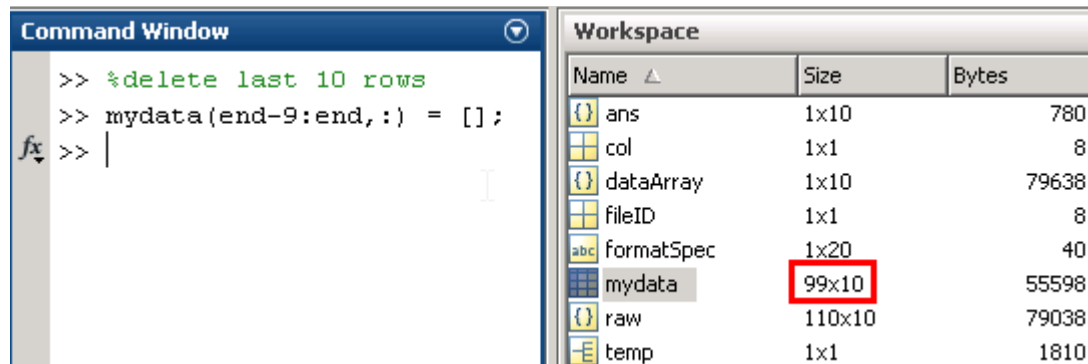
    NCommande    DateCommande    Article    Quantite
    '5'         '04.01.2000'    'Compaq Presario 100' 13
    '31'        '10.01.2000'    'AST Intel 150'        2
    '54'        '17.01.2000'    'IBM 500'              5
    '59'        '18.01.2000'    'IBM 500'              9
    '67'        '19.01.2000'    'IBM 500'              12
    '68'        '19.01.2000'    'AST Intel 150'        5
    '86'        '25.01.2000'    'Compaq Presario 100' 19
    '87'        '25.01.2000'    'IBM 500'              6

    PrixParPiece    Rabais    FacturePayee    PrixTotal
    1650            '1.50%'    'Oui'           21450
    2690            '0.00%'    'Oui'           5380
    2299            '0.00%'    'Oui'           11495
    2299            '1.50%'    'Oui'           20691
    2299            '1.50%'    'Non'           27588
    2690            '0.00%'    'Non'           13450
    1650            '3.00%'    'Non'           31350
    2299            '1.50%'    'Non'           13794

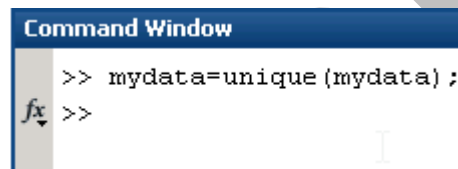
fx
```

Delete rows and clean duplicates

Delete rows in a dataset is exactly similar to delete rows in a matrix or vector. The syntax is just the following:

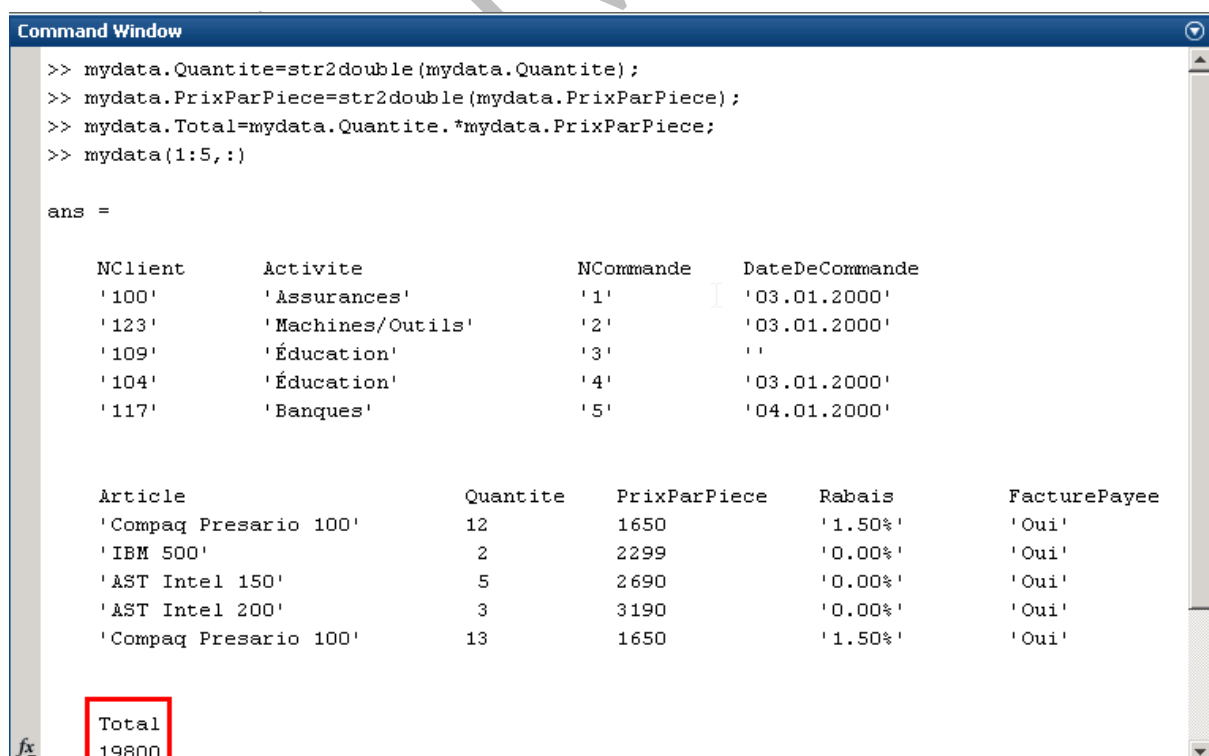


The function to clean the duplicates of a dataset is the same as for cleaning duplicates in a vector or matrix. It use the **unique()** function:



Add variables

One of the most common simple manipulation is to add new variable Consider the case where we want to add a new column who's results is the arithmetic operation of other columns:



Change variable name

Change a variable name could be very easy but the manipulation is a little bit strange to in MATLAB. We have get the old name before by using the property (or Method?) **VarNames** before we can be able to change the name.



The screenshot shows a MATLAB Command Window with the following text:

```
>> temp=get(mydata);
>> temp.VarNames

ans =

Columns 1 through 6

    'NClient'    'Activite'    'NCommande'    'DateDeCommande'    'Article'    'Quantite'

Columns 7 through 10

    'PrixParPiece'    'Rabais'    'FacturePayee'    'PrixTotal'

>> mydata.Properties.VarNames{4} = 'DateCommande';
>> temp=get(mydata);
>> temp.VarNames

ans =

Columns 1 through 6

    'NClient'    'Activite'    'NCommande'    'DateCommande'    'Article'    'Quantite'

Columns 7 through 10
```

In both outputs, the variable name 'DateDeCommande' is highlighted with a red box. In the second output, it has been changed to 'DateCommande'.

Change column order

ToDo... (search a better way than recreating a new dataset...)

Random sample of a dataset

Sampling in quality control is an important step of statistical inference. For this purpose the method is the same as the one for vectors using the functions **randsample()** and **length()** together:

```
Command Window
>> mydata(randsample(length(mydata),3), :)

ans =

    NClient      Activite      NCommande      DateCommande
    '122'        'Distribution'      '49'          '13.01.2000'
    '111'        'Alimentaire'      '80'          '24.01.2000'
    '113'        'Construction'      '95'          '27.01.2000'

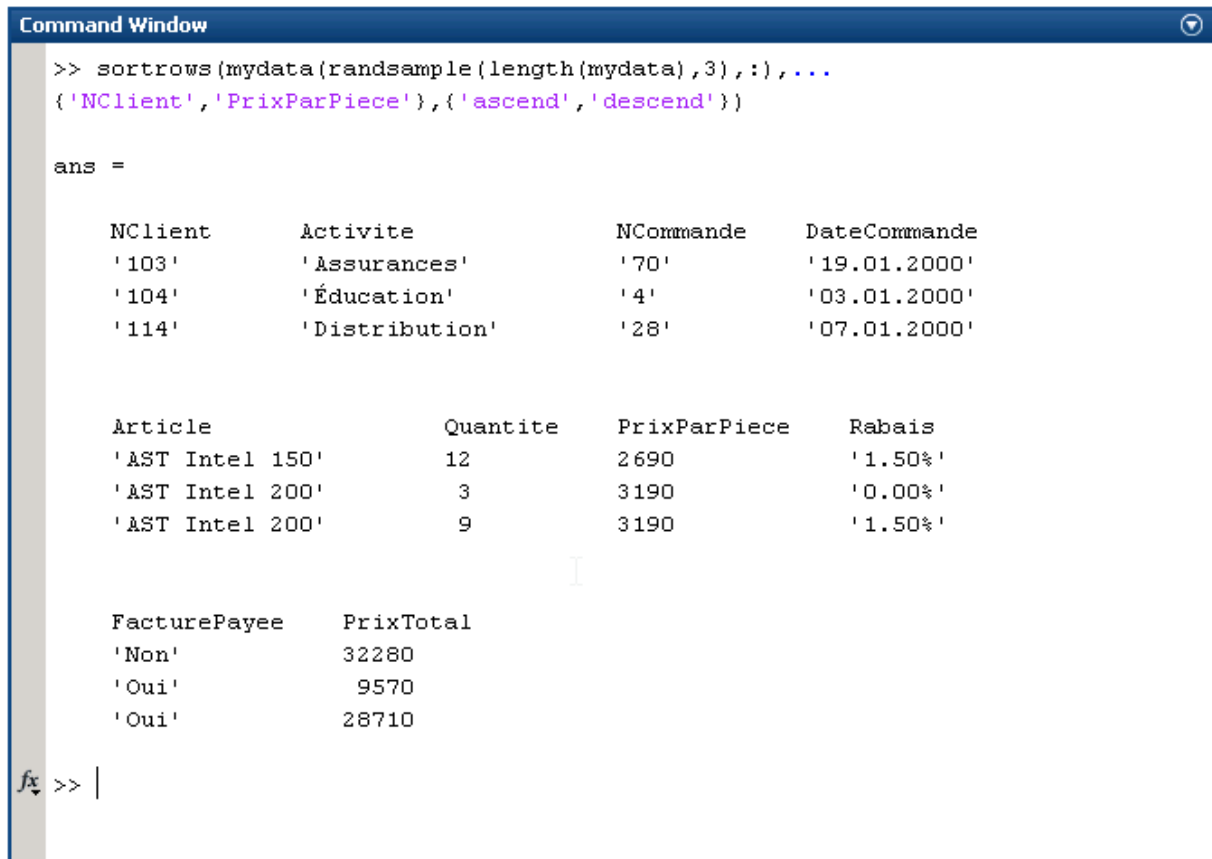
    Article      Quantite      PrixParPiece      Rabais
    'AST Intel 200'      7          3190          '1.50%'
    'AST Intel 200'      2          3190          '0.00%'
    'IBM 500'          6          2299          '1.50%'

    FacturePayee      PrixTotal
    'Oui'          22330
    'Non'          6380
    'Non'          13794

fx >> |
```

Sort a dataset

The command to sort a dataset in MATLAB is very simple. You just have to use the commande **sortrows()**:



```
>> sortrows(mydata(randsample(length(mydata),3),:),...
{'NClient','PrixParPiece'},{'ascend','descend'})

ans =
```

NClient	Activite	NCommande	DateCommande
'103'	'Assurances'	'70'	'19.01.2000'
'104'	'Éducation'	'4'	'03.01.2000'
'114'	'Distribution'	'28'	'07.01.2000'

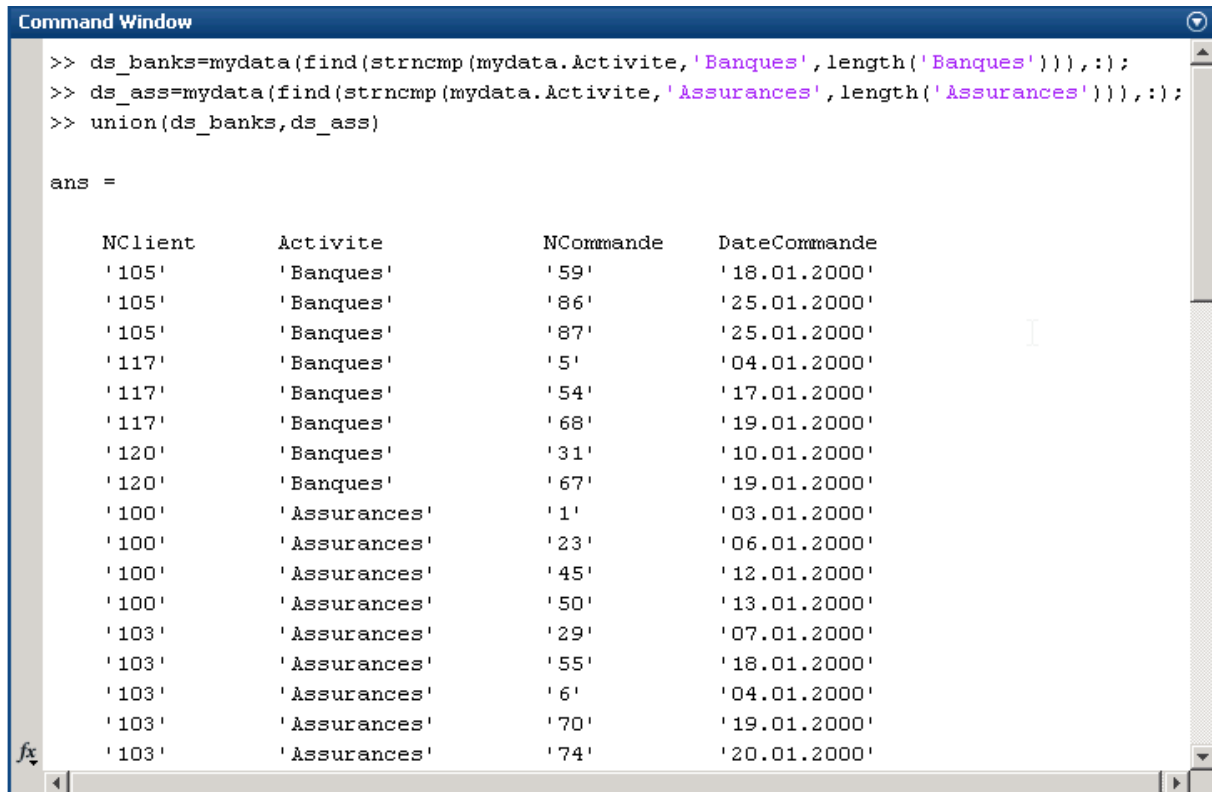
Article	Quantite	PrixParPiece	Rabais
'AST Intel 150'	12	2690	'1.50%'
'AST Intel 200'	3	3190	'0.00%'
'AST Intel 200'	9	3190	'1.50%'

FacturePayee	PrixTotal
'Non'	32280
'Oui'	9570
'Oui'	28710

```
fx >> |
```

Merge datasets

Another typical is to be able to merge two datasets having the same number of columns and same order of columns. For this purpose we simply use the functions **union()** as for vectors:



The image shows a MATLAB Command Window with the following code and output:

```
>> ds_banks=mydata(find(strcmp(mydata.Activite,'Banques'),length('Banques')),:);
>> ds_ass=mydata(find(strcmp(mydata.Activite,'Assurances'),length('Assurances')),:);
>> union(ds_banks,ds_ass)

ans =
```

NClient	Activite	NCommande	DateCommande
'105'	'Banques'	'59'	'18.01.2000'
'105'	'Banques'	'86'	'25.01.2000'
'105'	'Banques'	'87'	'25.01.2000'
'117'	'Banques'	'5'	'04.01.2000'
'117'	'Banques'	'54'	'17.01.2000'
'117'	'Banques'	'68'	'19.01.2000'
'120'	'Banques'	'31'	'10.01.2000'
'120'	'Banques'	'67'	'19.01.2000'
'100'	'Assurances'	'1'	'03.01.2000'
'100'	'Assurances'	'23'	'06.01.2000'
'100'	'Assurances'	'45'	'12.01.2000'
'100'	'Assurances'	'50'	'13.01.2000'
'103'	'Assurances'	'29'	'07.01.2000'
'103'	'Assurances'	'55'	'18.01.2000'
'103'	'Assurances'	'6'	'04.01.2000'
'103'	'Assurances'	'70'	'19.01.2000'
'103'	'Assurances'	'74'	'20.01.2000'

Plots

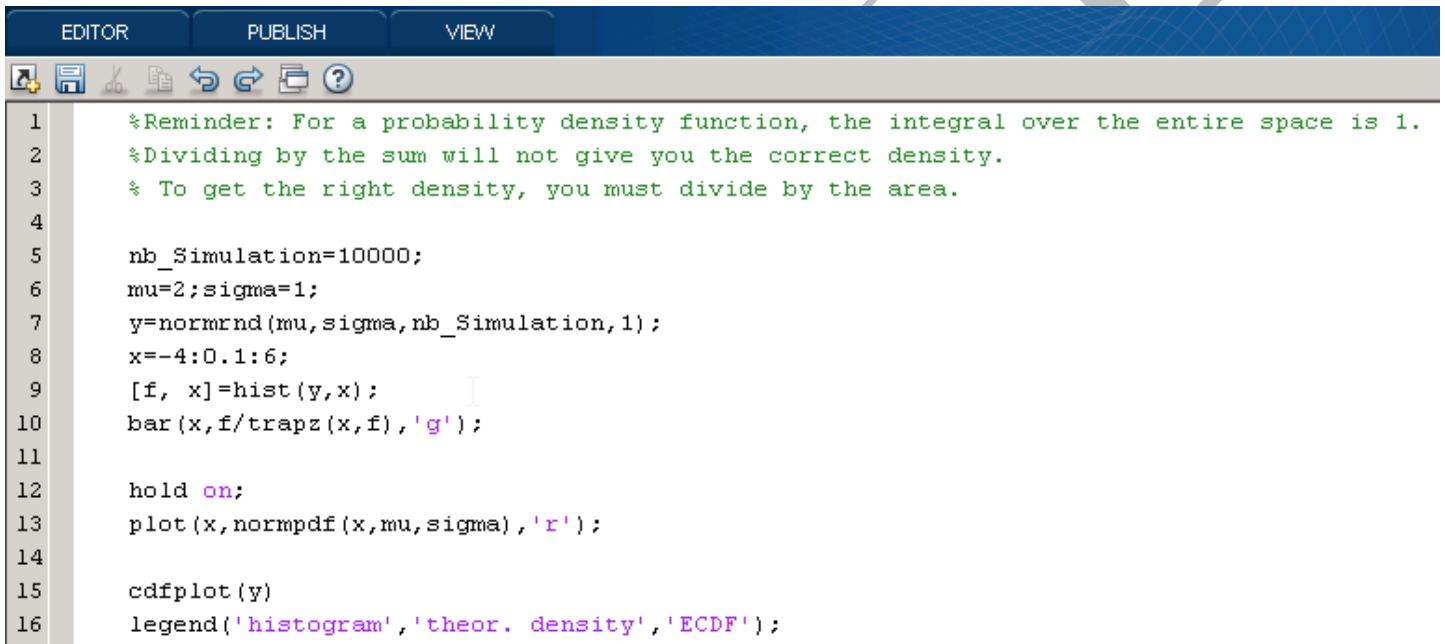
We have study during a very (to much) long time possible charts using only the core version of MATLAB (see page 245).

In this chapter we will focus on charts that can be done especially with the Statistical Toolbox.

Histogram/CDF/PDF

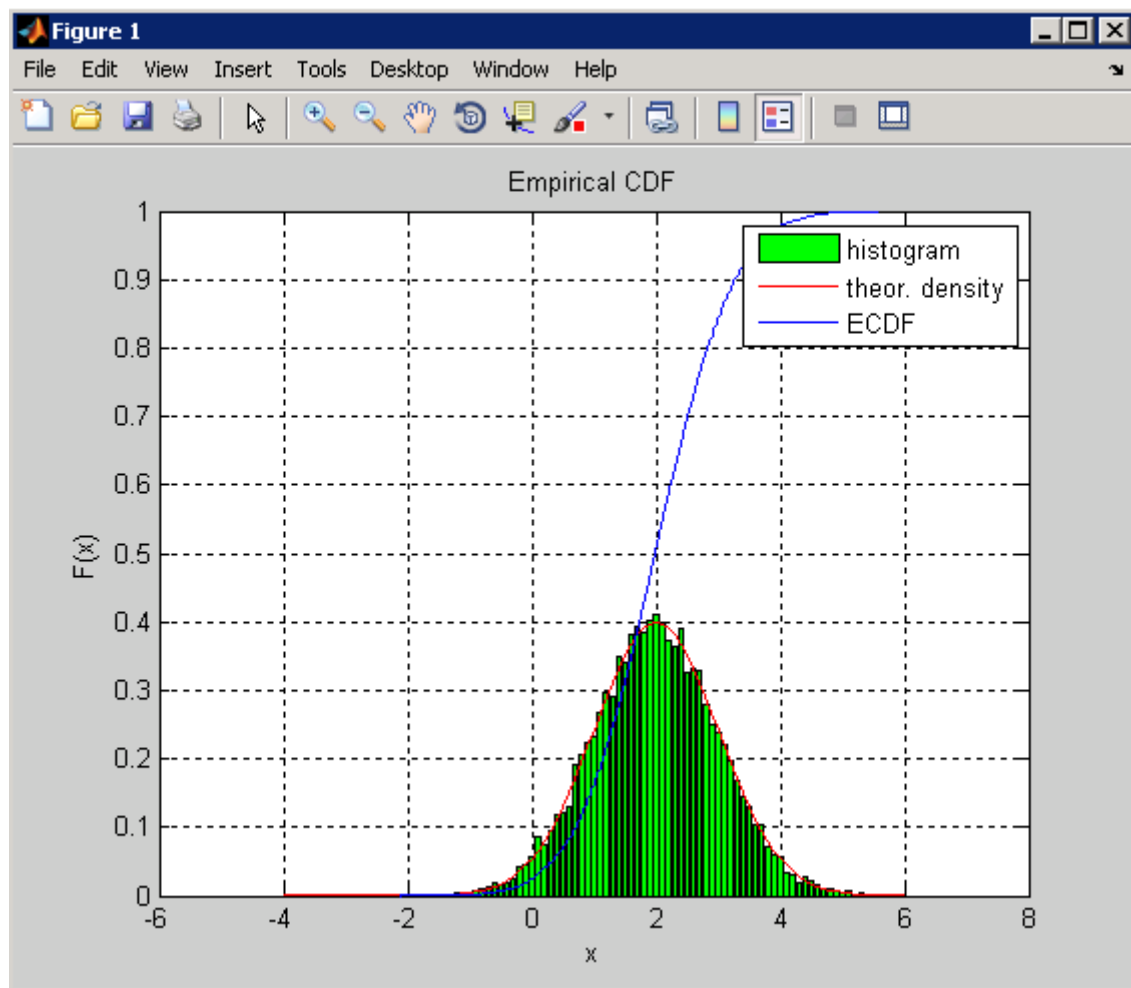
Create histograms is a very common task asked by many statistical softwares users even if it is not the best way to analyze the shape of data because of the bias of bin width.

The first method used the combination of functions **hist()**, **bar()**, **trapz()** and **cdfplot()** to plot on the same chart a histogram, a CDF and a PDF:



```
1 %Reminder: For a probability density function, the integral over the entire space is 1.
2 %Dividing by the sum will not give you the correct density.
3 % To get the right density, you must divide by the area.
4
5 nb_Simulation=10000;
6 mu=2;sigma=1;
7 y=normrnd(mu,sigma,nb_Simulation,1);
8 x=-4:0.1:6;
9 [f, x]=hist(y,x);
10 bar(x,f/trapz(x,f),'g');
11
12 hold on;
13 plot(x,normpdf(x,mu,sigma),'r');
14
15 cdfplot(y)
16 legend('histogram','theor. density','ECDF');
```

The result is visible on the next page.



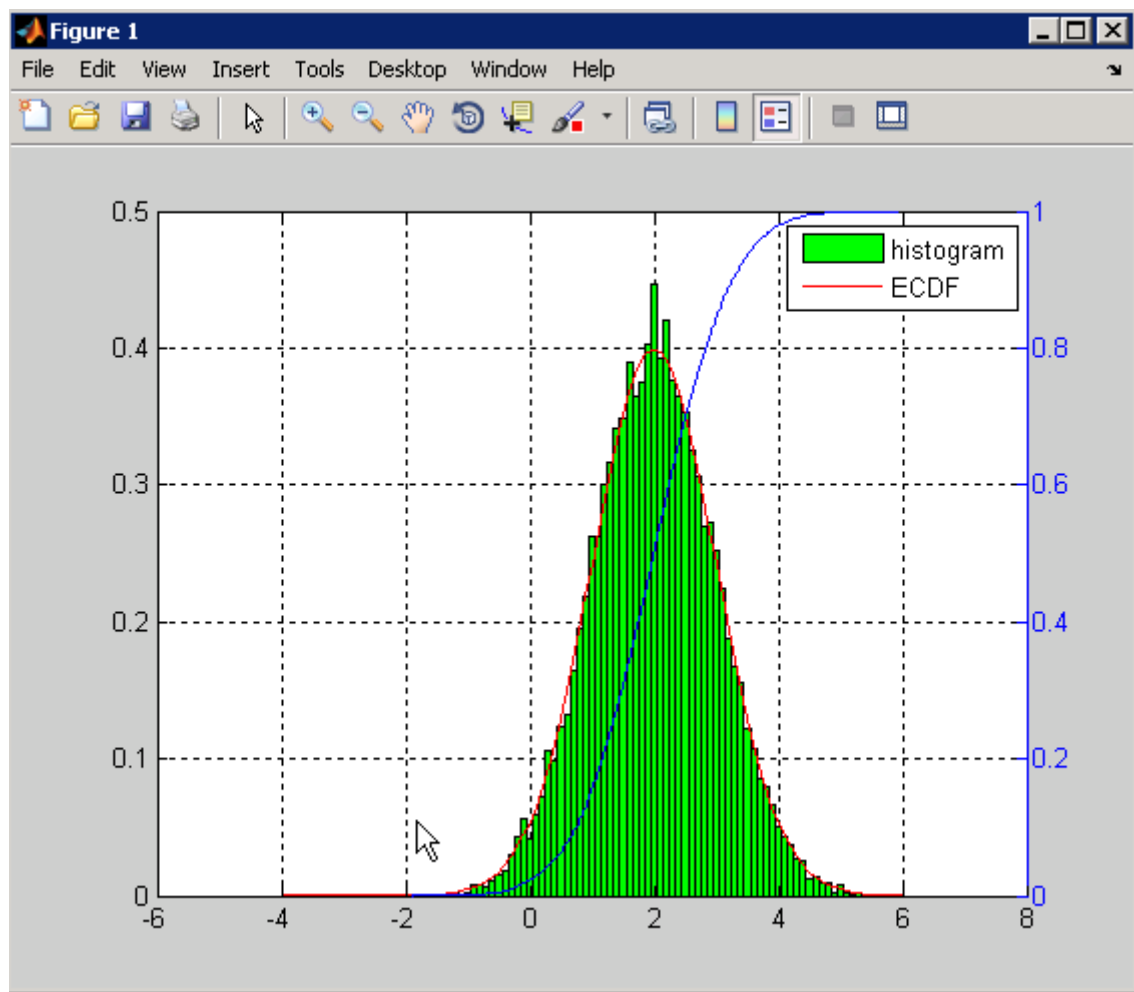
The second method is a variant using `ecdf()` instead of using a plot of the function with `cdf()`:

```

EDITOR    PUBLISH    VIEW
1      %Reminder: For a probability density function, the integral over the entire space is 1.
2      %Dividing by the sum will not give you the correct density.
3      % To get the right density, you must divide by the area.
4
5      nb_Simulation=10000;
6      mu=2;sigma=1;
7      y=normrnd(mu,sigma,nb_Simulation,1);
8      x=-4:0.1:6;
9      n=hist(y,x);
10
11     [y2,x2] = ecdf(y);
12     [y1,x1] = hist(y,x);
13     plotyy(x1,y1,x2,y2,@(x,y)bar(x,y/trapz(x,y),1,'g'),'stairs');
14     hold on;
15     plot(x,normpdf(x,mu,sigma),'r');
16     legend('histogram','ECDF');
17     grid on;

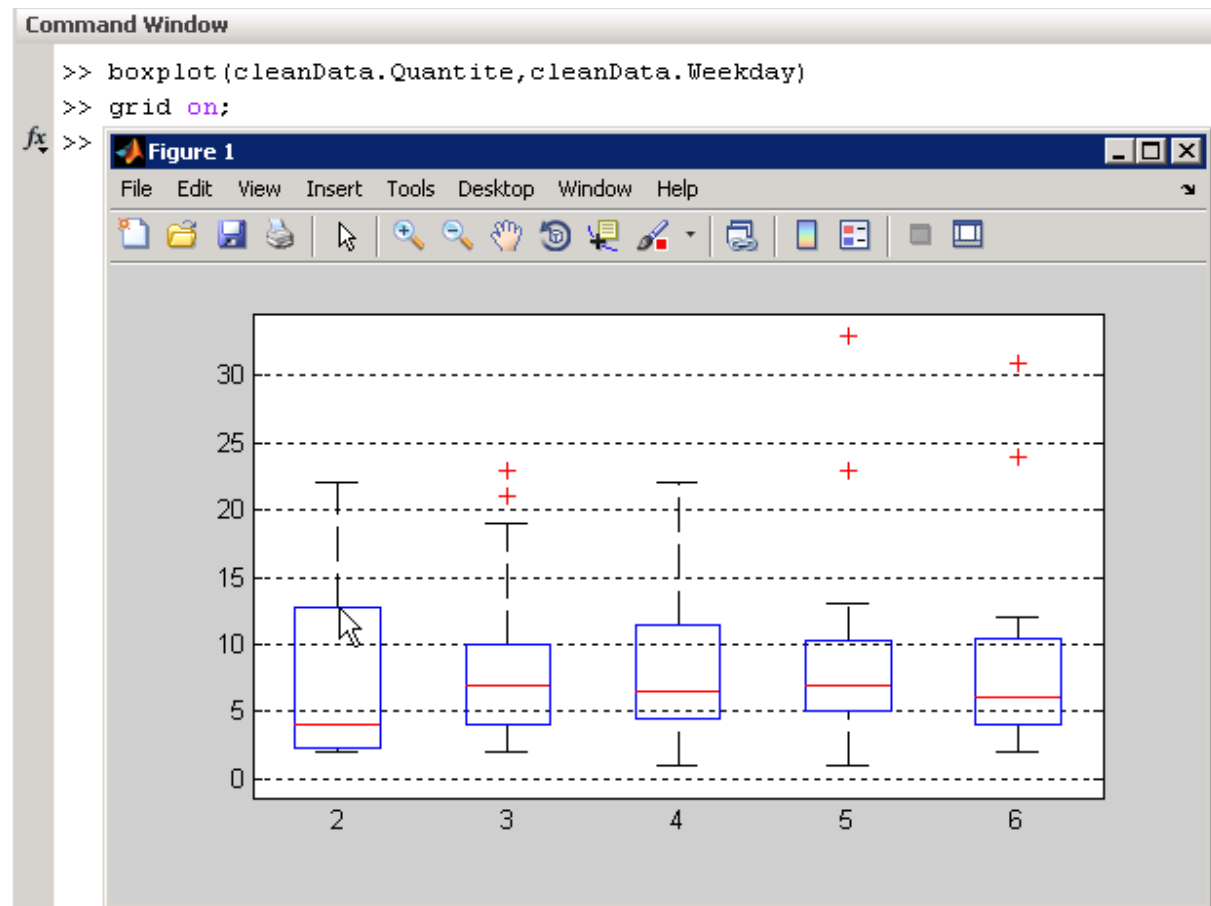
```

The result will be:



Box-Whisker plot

Compare to R, the Box-Whisker plot in MATLAB are very poor in possibilities and fine tuning but let us see an example using the function **boxplot()**:



Scatter Group plot

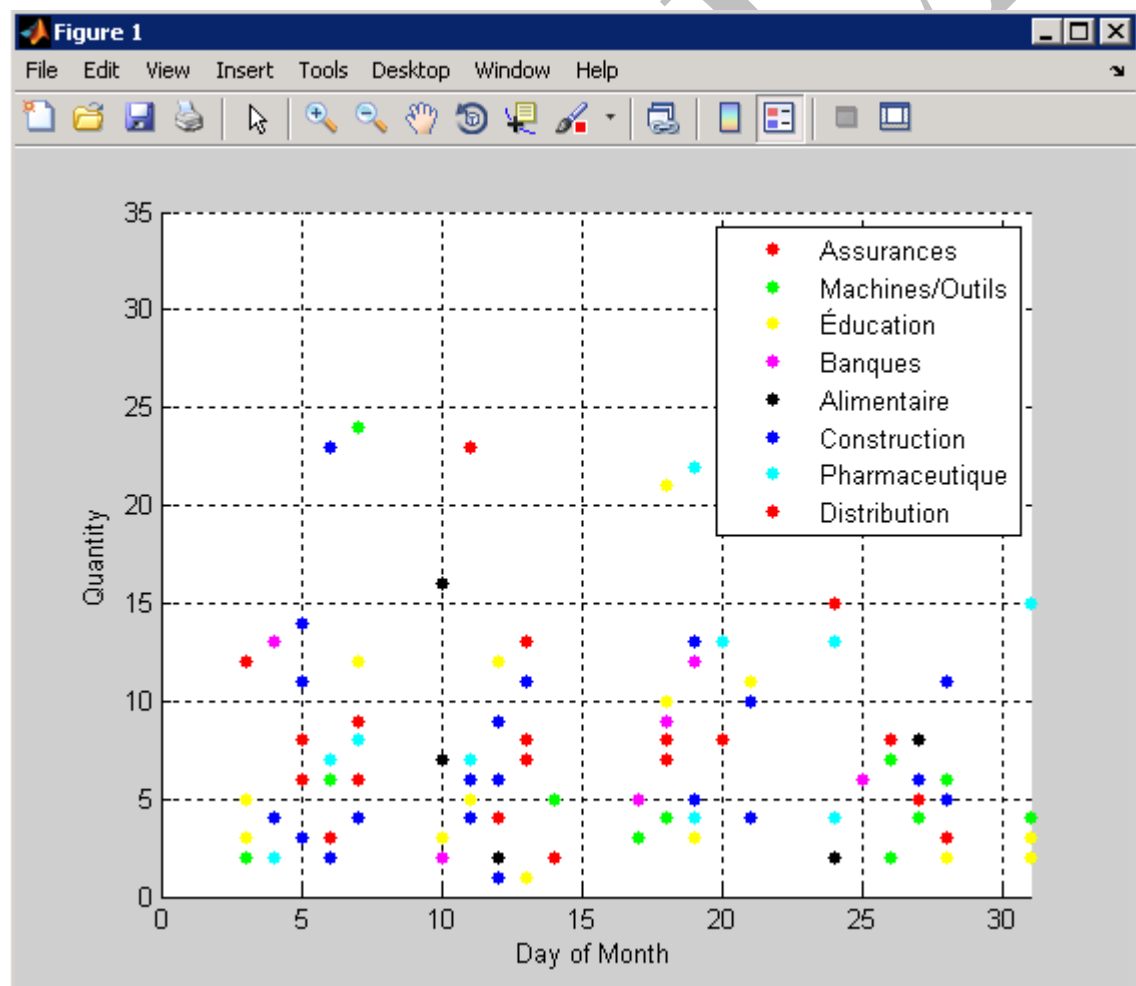
To simplify the management of multiple dot plots with datasets, MATLAB Statistical Toolbox has a **gscatter()** function to simplify a little bit the fact that we often need to plot groups of points with different colors:

```

1  gscatter(cleanData.Day, cleanData.Quantite, cleanData.Activite, 'rgymkbc')
2  %there is not enough color for the last group of Activite...
3  %this is why gscatter sucks
4  xlabel('Day of Month'); ylabel('Quantity');
5  xlim([0 31]);
6  grid on;

```

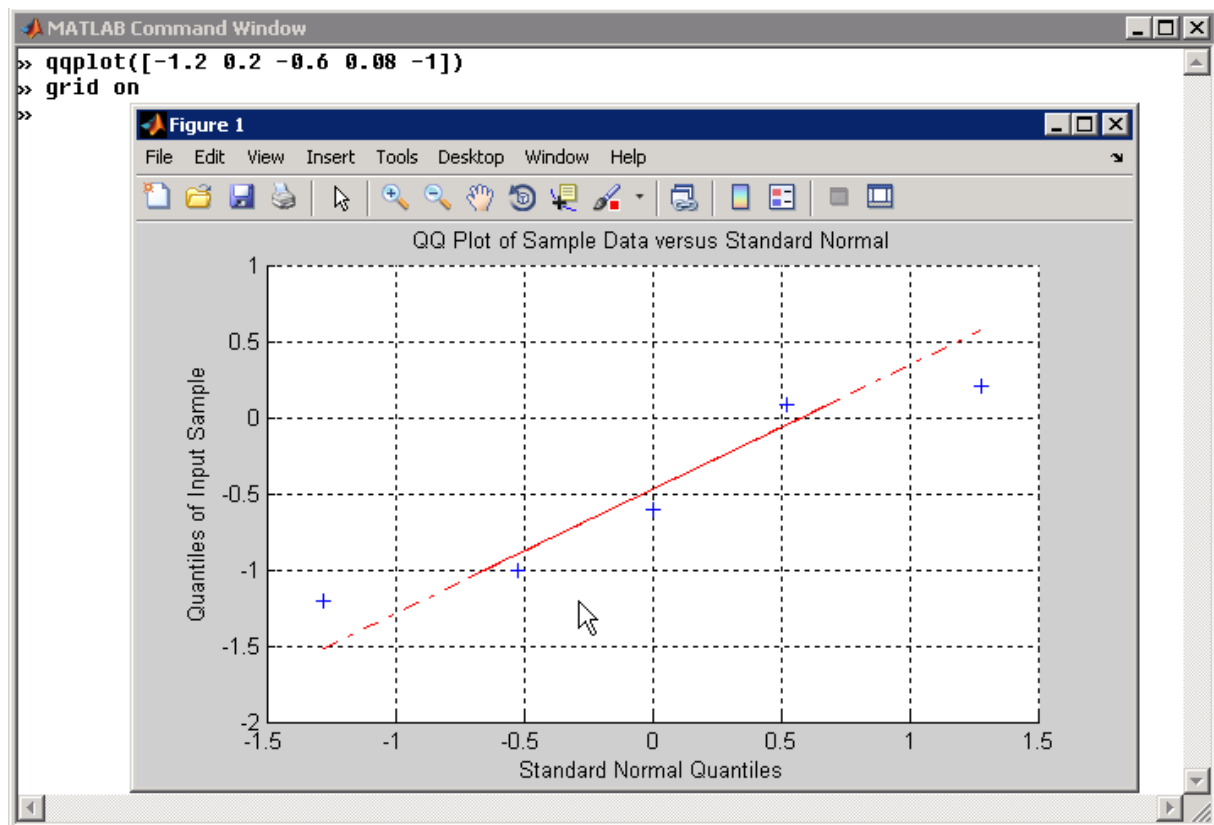
The result will be:



QQ plot

QQ plot are often used to check qualitatively the normality of data. For this purpose we have a **qqplot()** function in the MATLAB Statistical Toolbox:

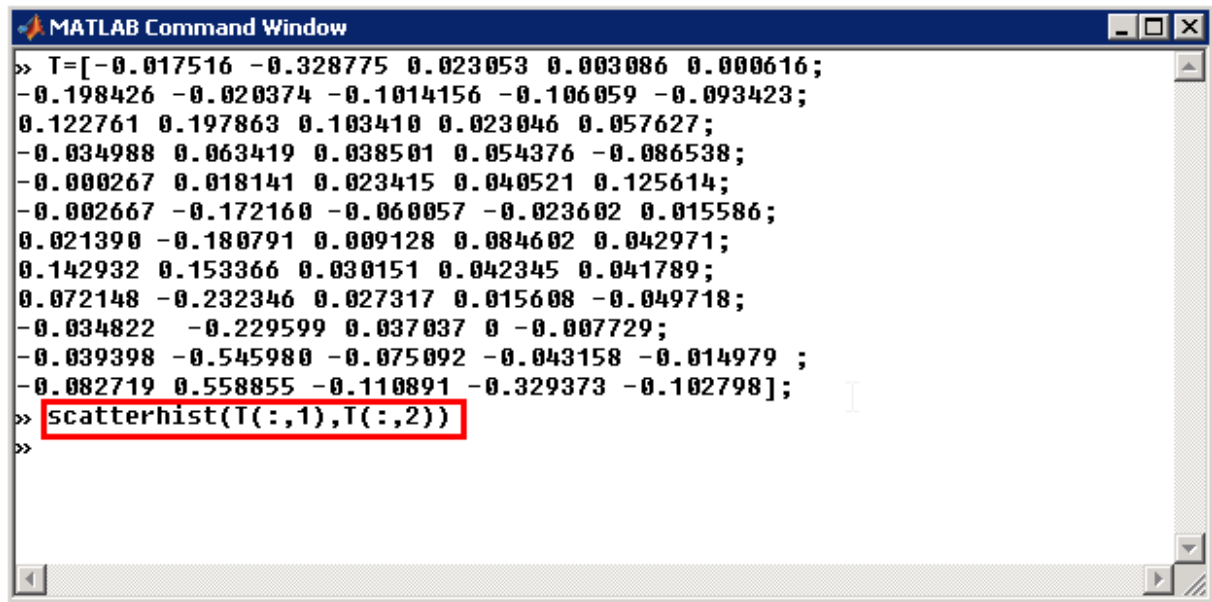
The main purpose of this example is just to compare the result with those of Minitab and R using the same data:



Histogram Scatter

Histogram scatters are very useful plots in finance and very flexible in R. But has such a plot with the function `scatterhist()` but the result is very hard to customize in comparison of R.

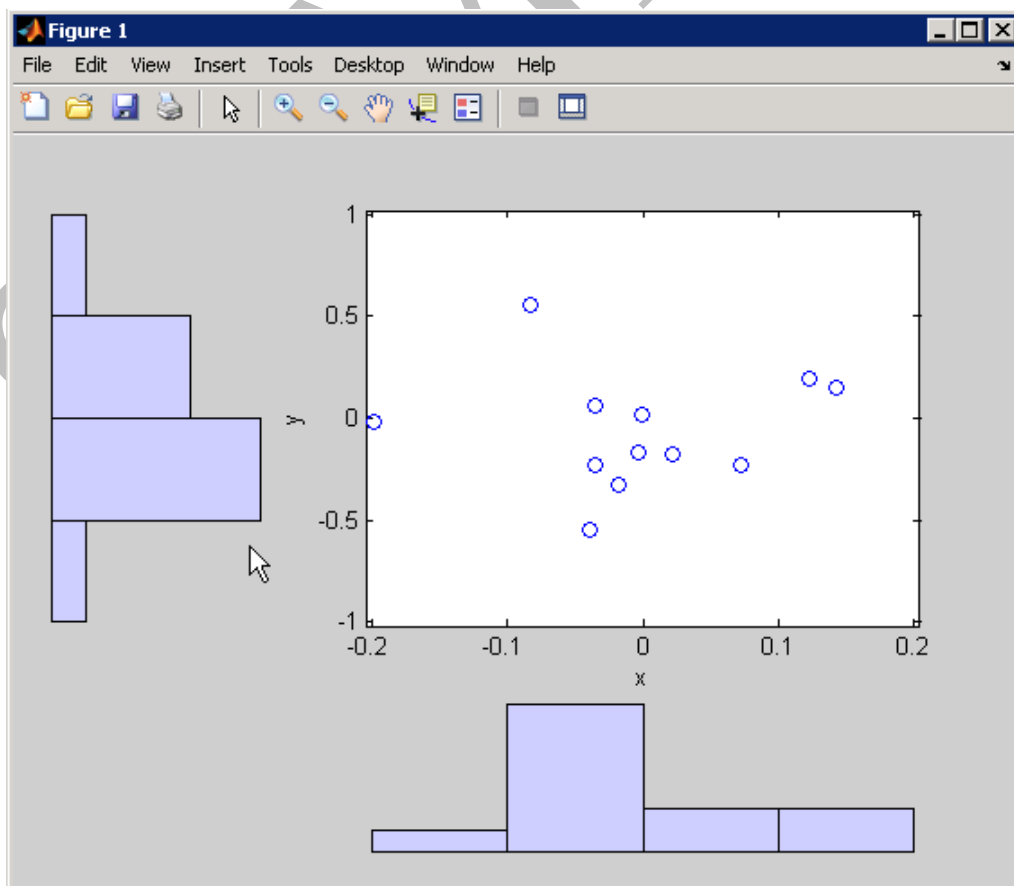
Using our asset data sample:



```

MATLAB Command Window
>> T=[-0.017516 -0.328775 0.023053 0.003086 0.000616;
-0.198426 -0.020374 -0.1014156 -0.106059 -0.093423;
0.122761 0.197863 0.103410 0.023046 0.057627;
-0.034988 0.063419 0.038501 0.054376 -0.086538;
-0.000267 0.018141 0.023415 0.040521 0.125614;
-0.002667 -0.172160 -0.060057 -0.023602 0.015586;
0.021390 -0.180791 0.009128 0.084602 0.042971;
0.142932 0.153366 0.030151 0.042345 0.041789;
0.072148 -0.232346 0.027317 0.015608 -0.049718;
-0.034822 -0.229599 0.037037 0 -0.007729;
-0.039398 -0.545980 -0.075092 -0.043158 -0.014979 ;
-0.082719 0.558855 -0.110891 -0.329373 -0.102798];
>> scatterhist(T(:,1),T(:,2))
>>
  
```

We get:



Copulas

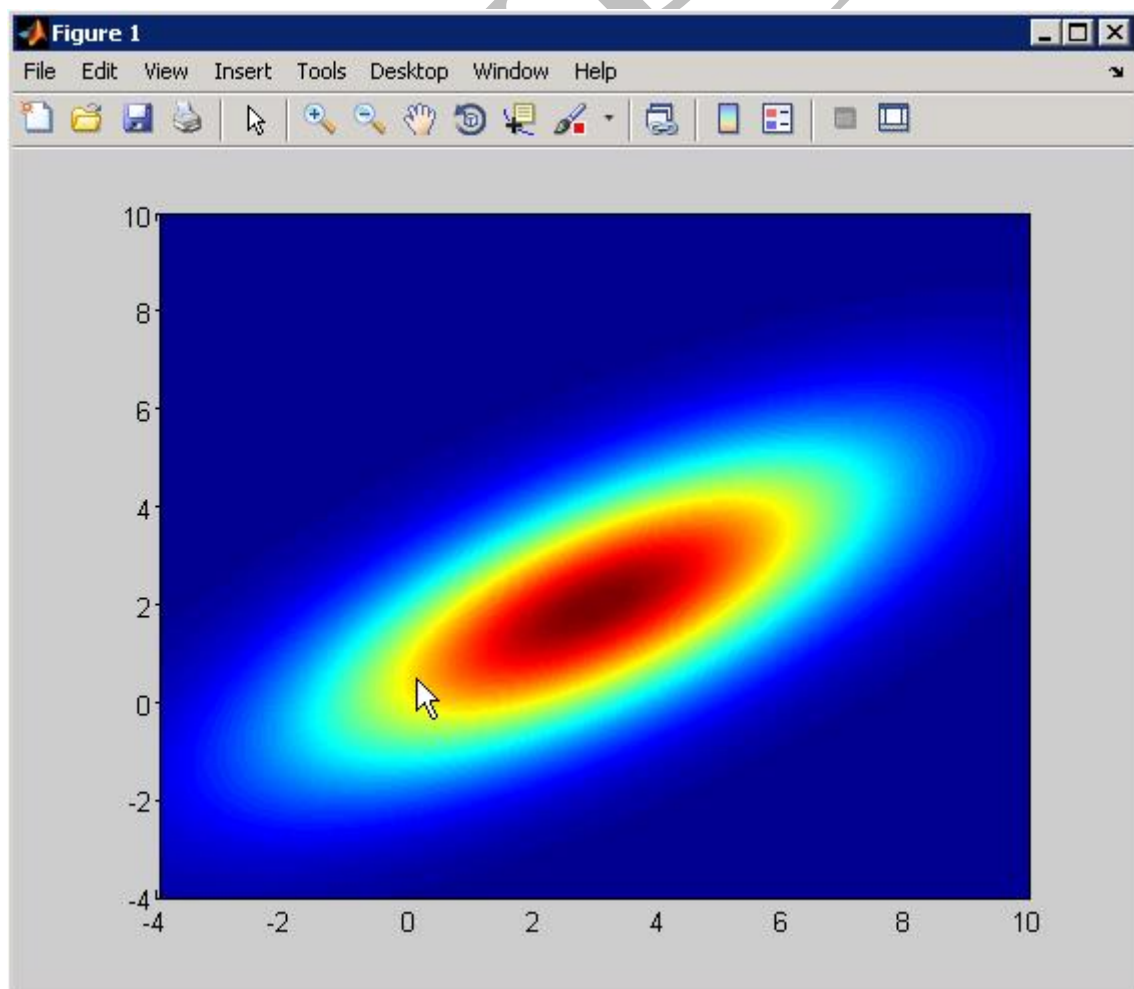
Copulas is a major subject if finance and because in the theoretical course we made only the proof of the Gaussian copula and its properties we will focus only on this one...

Gaussian copula

First, to simply simulate the same Gaussian copula as in the theoretical we will use the functions **mvnpdf()** and **pcolor()** of the Statistical Toolbox:

```
EDITOR PUBLISH VIEW
[X,Y]=meshgrid(-4:.05:10,-4:.05:10);
M=[3,2];
V1=[10 5;5 5];
Z1=mvnpdf([X(:),Y(:)],M,V1);
pcolor(X,Y,reshape(Z1,length(X),length(X)));
shading interp;
```

The result will be (exactly the same as the one obtained with R and Maple):

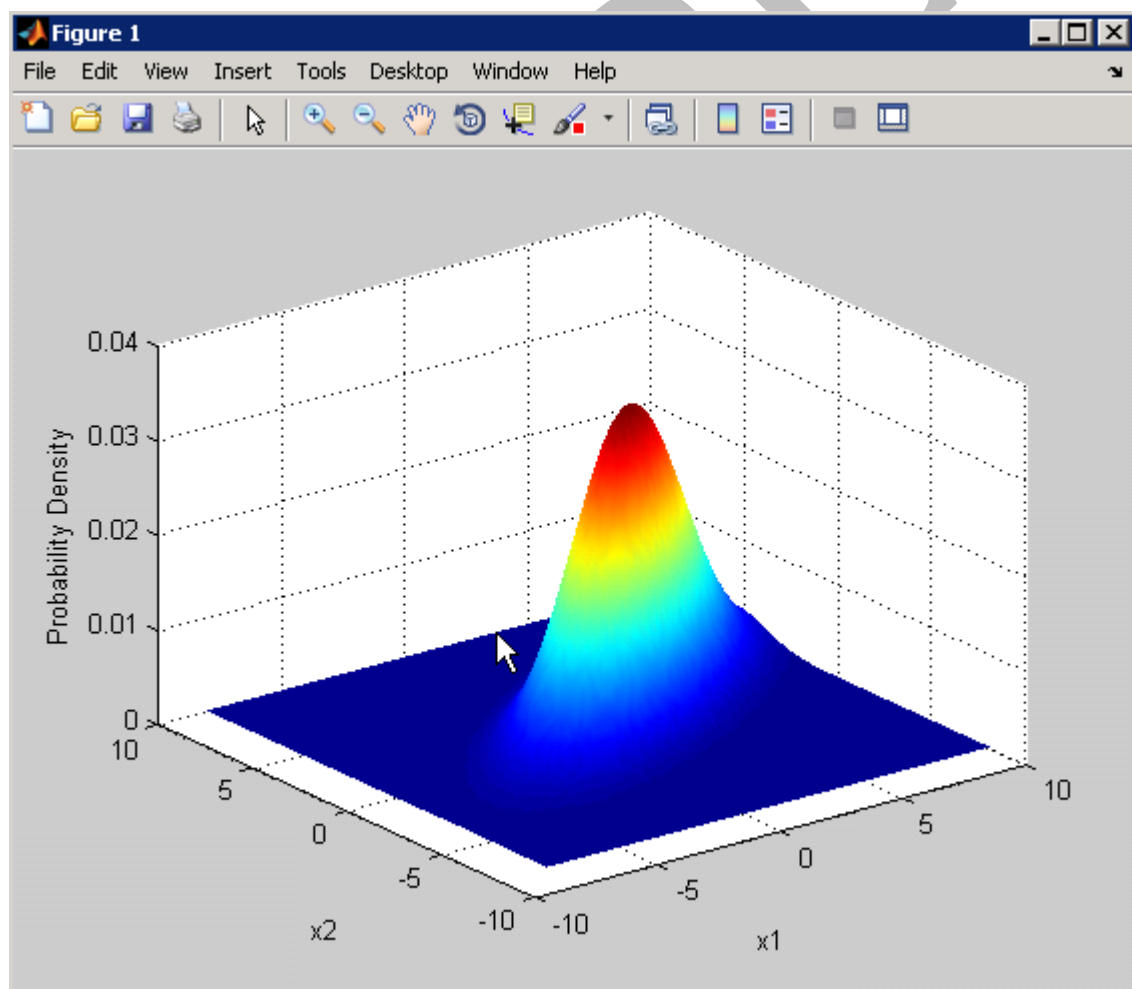


And what about the 3D view?

This is a hard to guess script using the function **reshape()** in addition to the previous script:

```
EDITOR PUBLISH VIEW
1 mu = [3 2];
2 Sigma = [10 5;5 5];
3 x1 = -8:.2:10;
4 x2 = -8:.2:10;
5 [X1,X2] = meshgrid(x1,x2);
6 F = mvnpdf([X1(:) X2(:)],mu,Sigma);
7 F = reshape(F,length(x2),length(x1));
8 surf(x1,x2,F);
9 xlabel('x1'); ylabel('x2'); zlabel('Probability Density');
10 shading interp;
```

We get the following result:



This is in adequation with what we get with R and Maple.

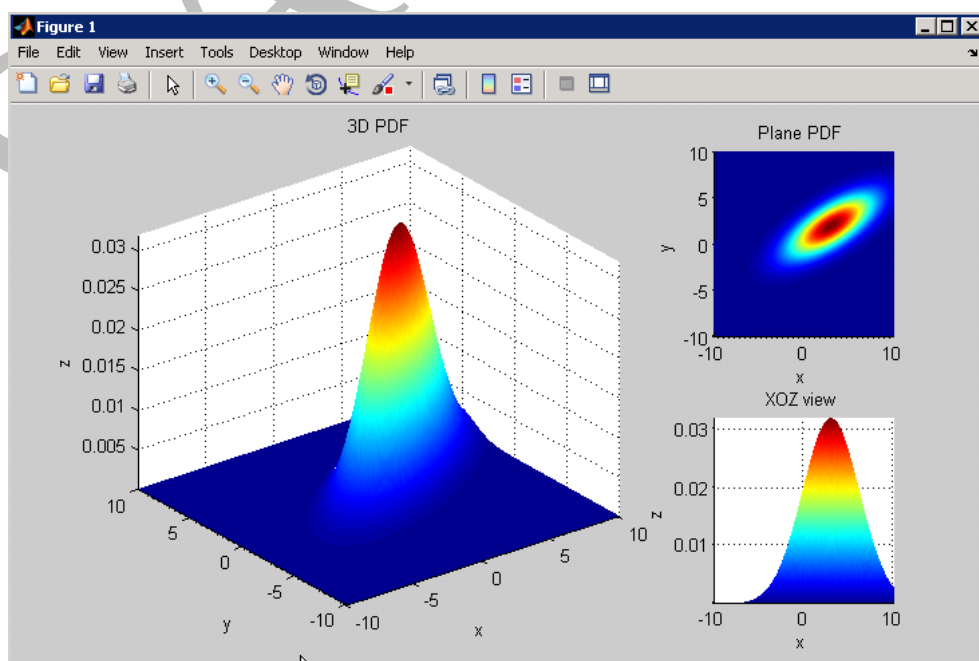
We can mix all the stuff below to get a global view using a subplot:

```

1  clc;clear;close all
2
3  mu=[3 2];
4  Sigma=[10 5;5 5];
5  [X,Y]=meshgrid(-10:0.1:10,-10:0.1:10);
6  p=mvnpdf([X(:) Y(:)],mu,Sigma);
7  p=reshape(p,size(X));
8
9  set(gcf,'Position',get(gcf,'Position').*[1 1 1.3 1])
10
11  subplot(2,3,[1 2 4 5])
12  surf(X,Y,p),axis tight,title('3D PDF')
13  xlabel('x');ylabel('y');zlabel('z');
14  shading interp;
15
16  subplot(2,3,3)
17  surf(X,Y,p),view(2),axis tight,title('Plane PDF')
18  xlabel('x');ylabel('y')
19  shading interp;
20
21  subplot(2,3,6)
22  surf(X,Y,p),view([0 0]),axis tight,title('XOZ view')
23  xlabel('x');zlabel('z')
24  shading interp;

```

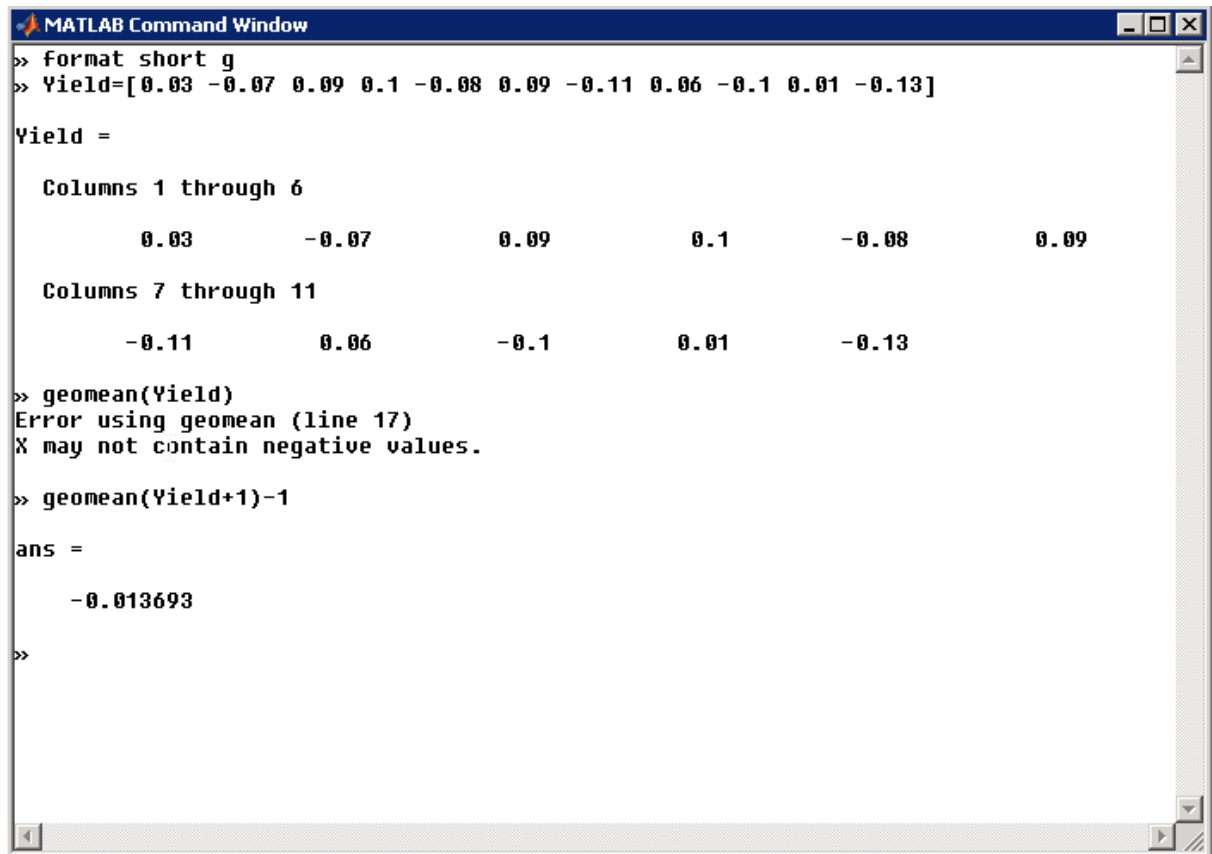
The result will be:



Descriptive statistics

Now that we have seen the basic charts available in the Statistical Toolbox we will see some very elementary statistical calculations.

Geometric mean



```
MATLAB Command Window
>> format short g
>> Yield=[0.03 -0.07 0.09 0.1 -0.08 0.09 -0.11 0.06 -0.1 0.01 -0.13]

Yield =

    Columns 1 through 6
           0.03        -0.07           0.09           0.1        -0.08           0.09
    Columns 7 through 11
        -0.11           0.06        -0.1           0.01        -0.13

>> geomean(Yield)
Error using geomean (line 17)
X may not contain negative values.

>> geomean(Yield+1)-1

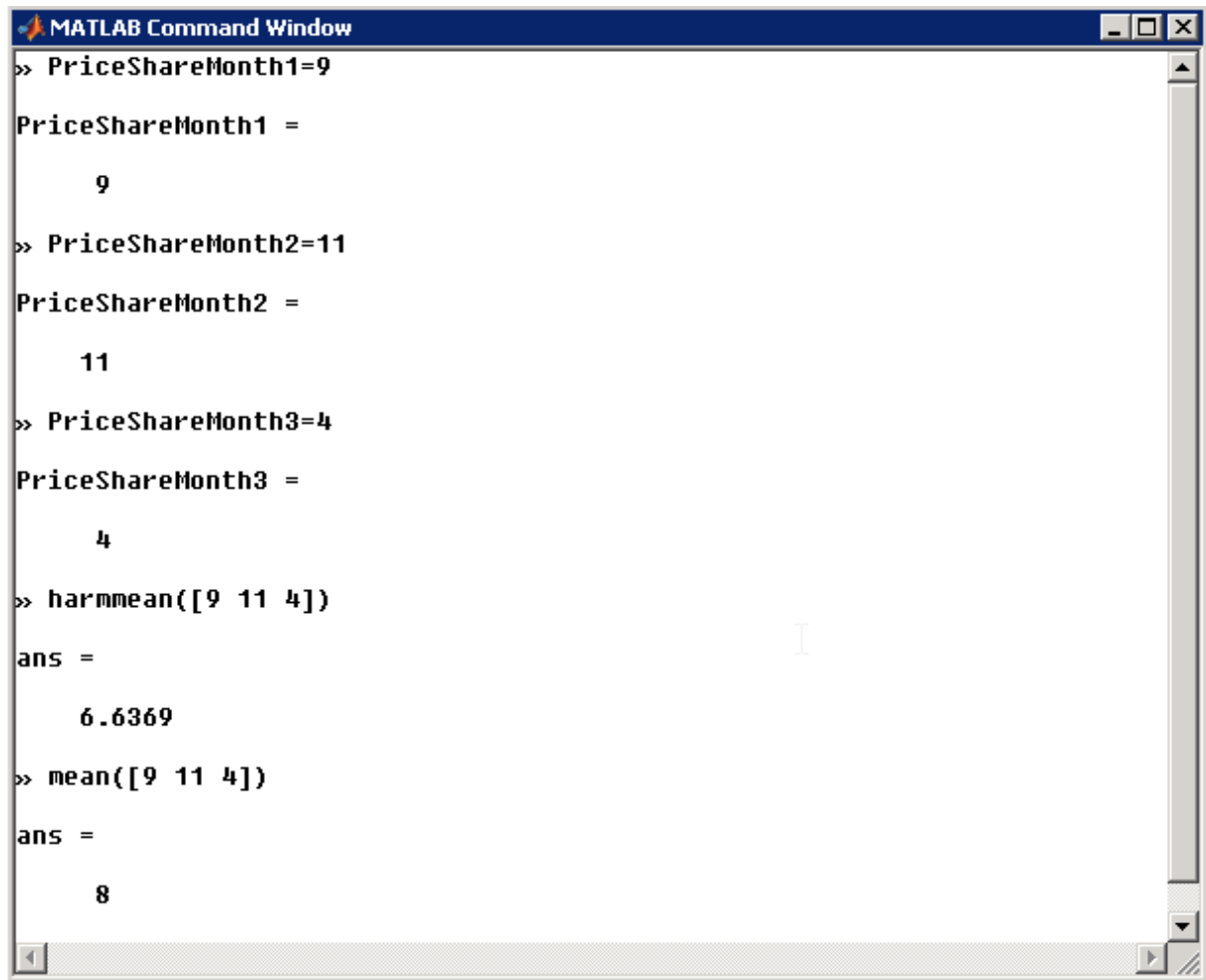
ans =

    -0.013693

>>
```

A screenshot of the MATLAB Command Window. The window title is "MATLAB Command Window". The command prompt shows the following sequence of commands and outputs:
1. `>> format short g`
2. `>> Yield=[0.03 -0.07 0.09 0.1 -0.08 0.09 -0.11 0.06 -0.1 0.01 -0.13]`
3. The output `Yield =` is displayed, showing the matrix values in two rows:
 - Row 1 (Columns 1 through 6): 0.03, -0.07, 0.09, 0.1, -0.08, 0.09
 - Row 2 (Columns 7 through 11): -0.11, 0.06, -0.1, 0.01, -0.13
4. The command `>> geomean(Yield)` is entered, resulting in an error message: `Error using geomean (line 17) X may not contain negative values.`
5. The command `>> geomean(Yield+1)-1` is entered, resulting in the output `ans = -0.013693`.
6. The prompt `>>` is shown at the bottom, indicating the command window is ready for further input.

Harmonic mean

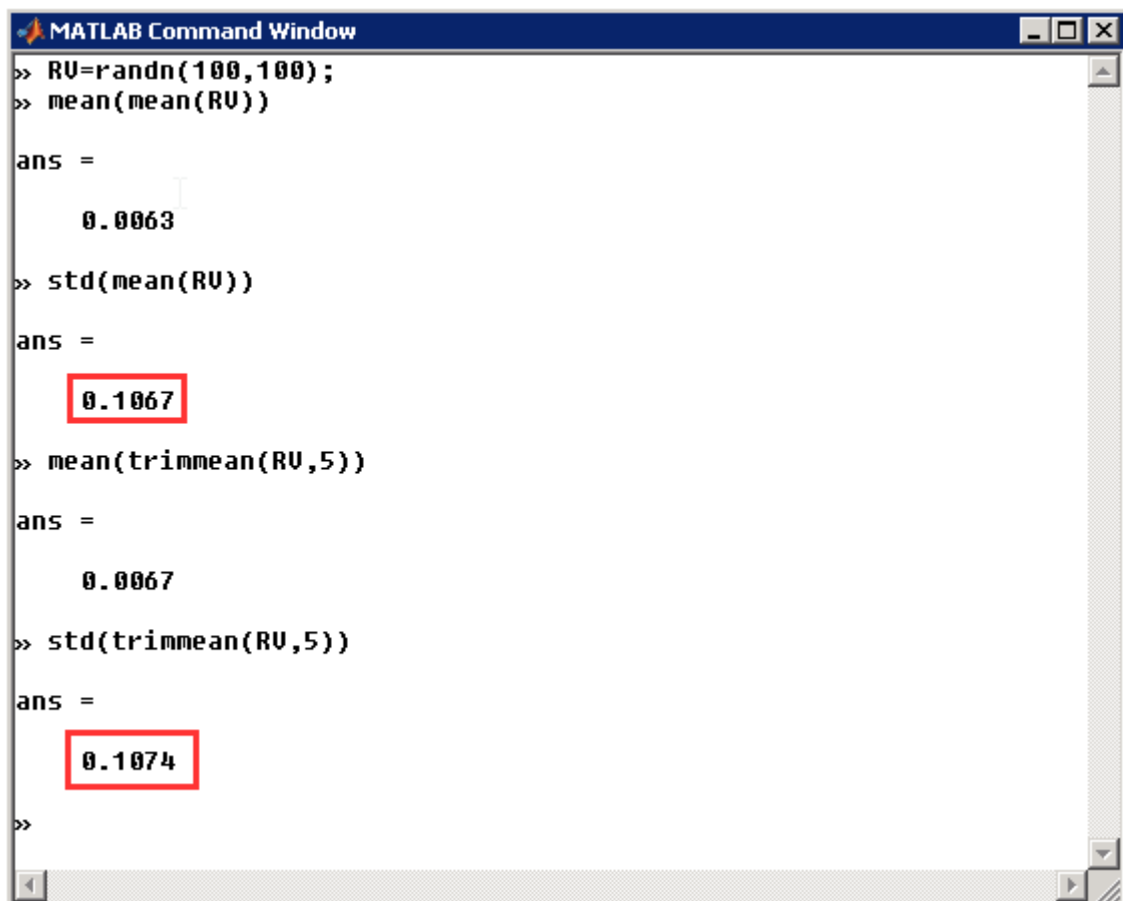
A screenshot of the MATLAB Command Window. The window has a blue title bar with the text 'MATLAB Command Window'. The command history shows the following: 'PriceShareMonth1=9' followed by 'PriceShareMonth1 =' and the value '9'; 'PriceShareMonth2=11' followed by 'PriceShareMonth2 =' and the value '11'; 'PriceShareMonth3=4' followed by 'PriceShareMonth3 =' and the value '4'. Then, the command 'harmmean([9 11 4])' is entered, resulting in 'ans = 6.6369'. Finally, the command 'mean([9 11 4])' is entered, resulting in 'ans = 8'.

```
MATLAB Command Window
>> PriceShareMonth1=9
PriceShareMonth1 =
    9
>> PriceShareMonth2=11
PriceShareMonth2 =
   11
>> PriceShareMonth3=4
PriceShareMonth3 =
    4
>> harmmean([9 11 4])
ans =
   6.6369
>> mean([9 11 4])
ans =
    8
```

Trimmed mean (M-statistics)

In comparison of SPSS as far as I know MATLAB Statistical Toolbox is very poor in the integration of well know M-Statistics. The only integrated one seems to be the trimmed mean....

Then let us see how to use it with a trimmed of 5%:



A screenshot of the MATLAB Command Window. The window has a blue title bar with the MATLAB logo and the text "MATLAB Command Window". The command history shows the following:
1. `>> RV=randn(100,100);`
2. `>> mean(mean(RV))`
The output is `ans = 0.0063`.
3. `>> std(mean(RV))`
The output is `ans = 0.1067`, which is highlighted with a red rectangular box.
4. `>> mean(trimmean(RV,5))`
The output is `ans = 0.0067`.
5. `>> std(trimmean(RV,5))`
The output is `ans = 0.1074`, which is also highlighted with a red rectangular box.
The window ends with `>>`. A large, light gray "SAMPLE" watermark is oriented diagonally across the lower half of the page.

```
MATLAB Command Window
>> RV=randn(100,100);
>> mean(mean(RV))

ans =

    0.0063

>> std(mean(RV))

ans =

    0.1067

>> mean(trimmean(RV,5))

ans =

    0.0067

>> std(trimmean(RV,5))

ans =

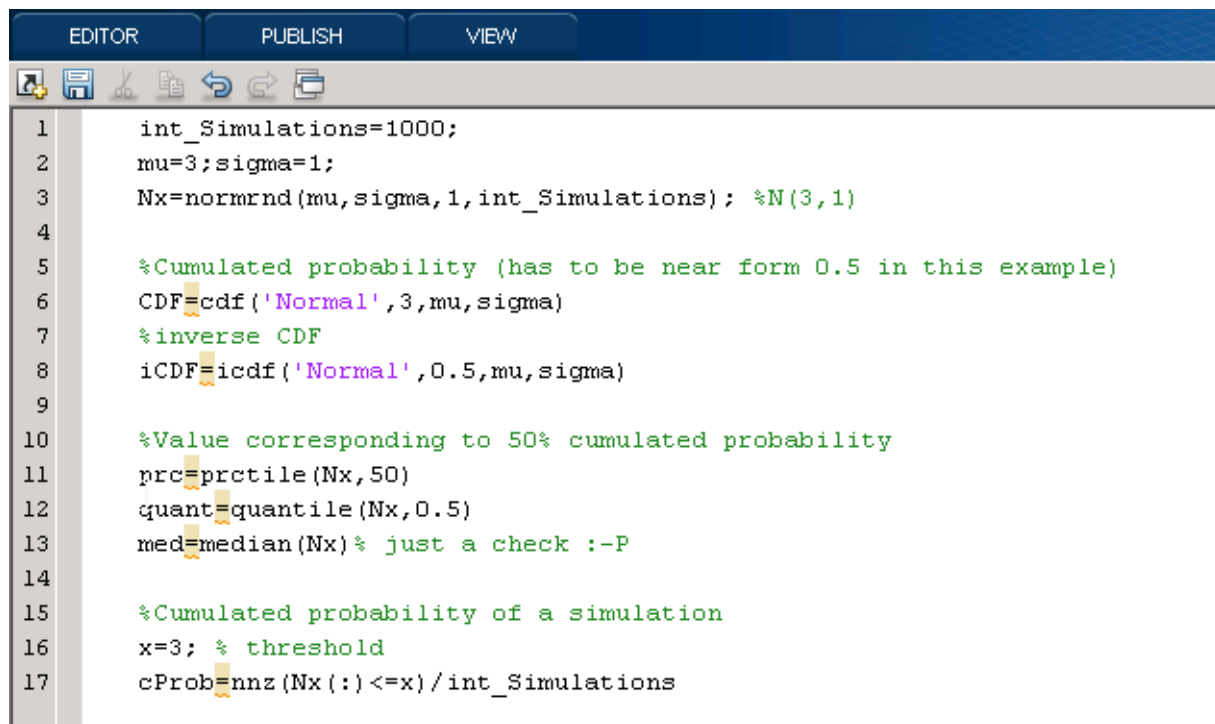
    0.1074

>>
```

CDF/iCDF/Quantile/Centile

Cumulated probability **cdf()**, inverse cumulated probability **icdf()** and also percentile **prctile()**, **quantile()** and **median()** are use a lot in finance and in general in stochastic simulations for various purposes (Value At Risk, Treshold values of a sample, etc.).

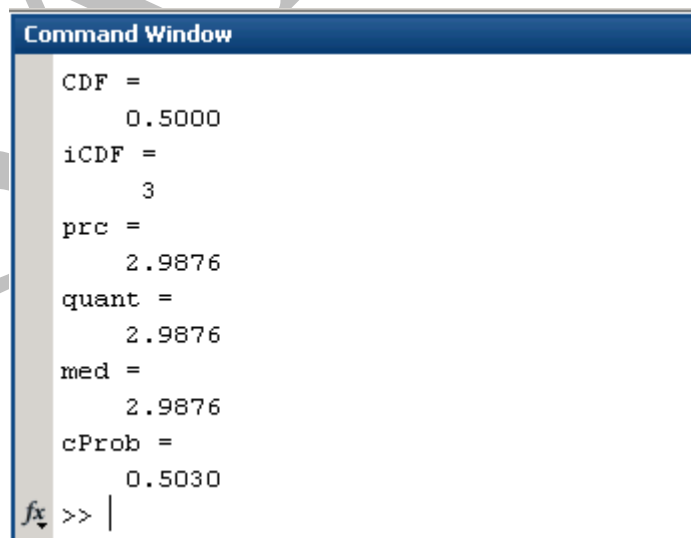
Let us see a very academic example used to check the validity of the different aspect of those functions:



```

1      int_Simulations=1000;
2      mu=3;sigma=1;
3      Nx=normrnd(mu,sigma,1,int_Simulations); %N(3,1)
4
5      %Cumulated probability (has to be near form 0.5 in this example)
6      CDF=cdf('Normal',3,mu,sigma)
7      %inverse CDF
8      iCDF=icdf('Normal',0.5,mu,sigma)
9
10     %Value corresponding to 50% cumulated probability
11     prc=prctile(Nx,50)
12     quant=quantile(Nx,0.5)
13     med=median(Nx)% just a check :-P
14
15     %Cumulated probability of a simulation
16     x=3; % threshold
17     cProb=nnz(Nx(:)<=x)/int_Simulations
  
```

The result will give:



```

Command Window

CDF =
    0.5000

iCDF =
     3

prc =
    2.9876

quant =
    2.9876

med =
    2.9876

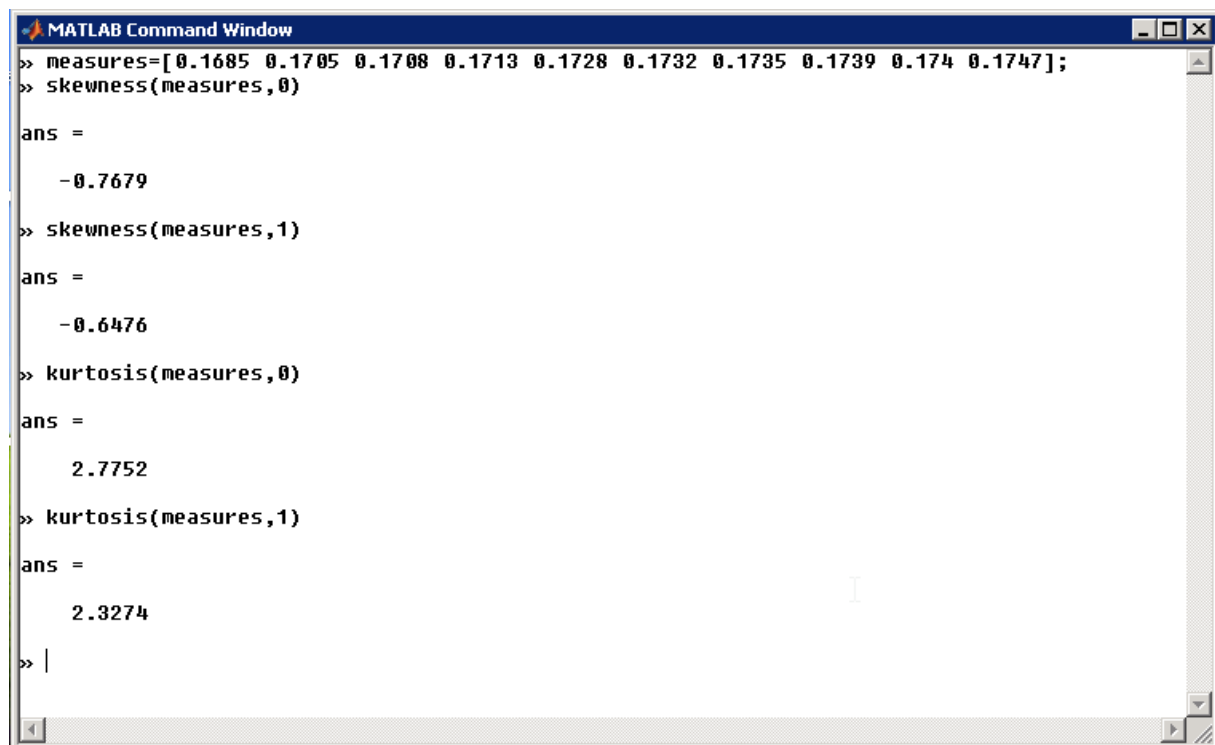
cProb =
    0.5030

fx >> |
  
```

Skewness/Kurtosis

Since there are many methods to calculate the kurtosis (flatness) and the skewness³ (asymmetry), I still want to treat them separately from the rest.

Consider the following example:



```

MATLAB Command Window
>> measures=[0.1685 0.1705 0.1708 0.1713 0.1728 0.1732 0.1735 0.1739 0.174 0.1747];
>> skewness(measures,0)

ans =

    -0.7679

>> skewness(measures,1)

ans =

    -0.6476

>> kurtosis(measures,0)

ans =

     2.7752

>> kurtosis(measures,1)

ans =

     2.3274

>> |
  
```

In MATLAB, we have then the following rules::

- If the **skewness** is zero, the distribution is symmetric (this does not mean that the symmetry occurs on a peak of the distribution because in the case of a bimodal distribution the axis of symmetry can be balanced between the two modal values). If the skewness is positive, the distribution (modal/median) looked right (or there are extreme right values). If the skewness is negative, the distribution (modal/median) is leaning to the left (or there are extreme left values)
- If the **kurtosis** is equal to three (**platikurtic**) then flattening is similar to a Normal distribution. If the value is greater than 3, then the distribution of interest (**leptokurtic**) is then higher than that of a Normal distribution with an equal mean (conversely - **mesokurtic** - if kurtosis is obviously less than 3)

³ At least 11 methods to calculate the Skewness. See the study of Tabor, J. (2010), "Investigating the Investigative Task: Testing for Skewness - An Investigation of Different Test Statistics and their Power to Detect Skewness," Journal of Statistics Education, 18, 1-13. www.amstat.org/publications/jse/v18n2/tabor.pdf

For information and for comparison given the different more common ways to calculate skewness and kurtosis (the first one is called "Fisher-Pearson skewness coefficient" and the second "adjusted Fisher-Pearson standardized moment coefficient"):

$$\gamma_1 = \frac{\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^3}{\left(\sqrt{\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2} \right)^3}$$

$$\gamma_2 = \gamma_1 \sqrt{\frac{n}{(n-1)(n-2)}}$$

$$\gamma_3 = \gamma_1 \left(\frac{n-1}{n} \right)^{3/2}$$

and:

$$\kappa_1 = \frac{\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^4}{\left(\sqrt{\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2} \right)^4}$$

$$\kappa_2 = \frac{(N-1)((N+1)\kappa_1 - 3(N+1))}{(N-2)(N-3)} + 3$$

$$\kappa_3 = \frac{N(N+1)}{(N-1)(N-2)(N-3)} \kappa_1 - \frac{3(N-1)^2}{(N-2)(N-3)}$$

$$\kappa_4 = \kappa_1 - 3$$

$$\kappa_5 = (\kappa_4 + 3) \left(1 - \frac{1}{N} \right)^2 - 3$$

We have by comparison with MATLAB, R, Mathematica, Minitab, IMB SPSS Statistics and Microsoft Excel:

Skewness	R	$-0.647(\gamma_1), -0.77(\gamma_2), -0.552(\gamma_3)$
	MATLAB	$-0.647(\gamma_1), -0.77(\gamma_2)$
	Mathematica	$-0.647(\gamma_1)$
	Minitab, SPSS, Excel	$-0.77(\gamma_3)$
Kurtosis	R	$-1.114(\kappa_5), -0.22(\kappa_3), -0.67(\kappa_4)$
	MATLAB	$2.327(\kappa_1), 2.775(\kappa_2)$
	Mathematica	$2.327(\kappa_1)$
	Minitab, SPSS, Excel	$-0.22(\kappa_3)$

Hypothesis tests

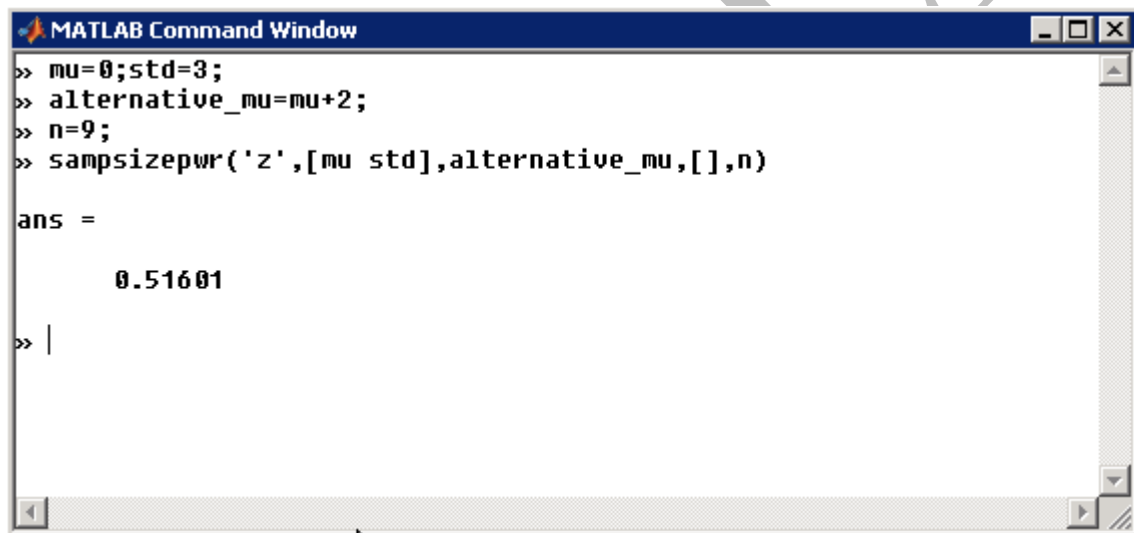
In this chapter we will also use the same data as in the Microsoft Excel, Minitab, SPSS and R training to be able to compare the results with the different e-books and videos tutorials. The purpose is also to compare the results with those obtained by hand during the proof of the different mathematical models!

Tests power, sample size and resolution

We have seen (proved) in the theoretic course the importance of the calculation of the a priori or a posteriori calculation of the power of some statistical tests and use the mathematical results to also be able to determine the necessary sample size or the resolution of a test.

Power of a Z test

First have a look if we found with MATLAB the same power for a Z-test as the one calculated by hand or with Minitab. For this purpose we use the function `sampsizepwr()`:

A screenshot of the MATLAB Command Window. The window has a blue title bar with the MATLAB logo and the text "MATLAB Command Window". The command prompt shows the following code:

```
>> mu=0;std=3;  
>> alternative_mu=mu+2;  
>> n=9;  
>> sampsizepwr('z',[mu std],alternative_mu,[],n)
```

 The output is displayed as

```
ans =  
  
    0.51601
```

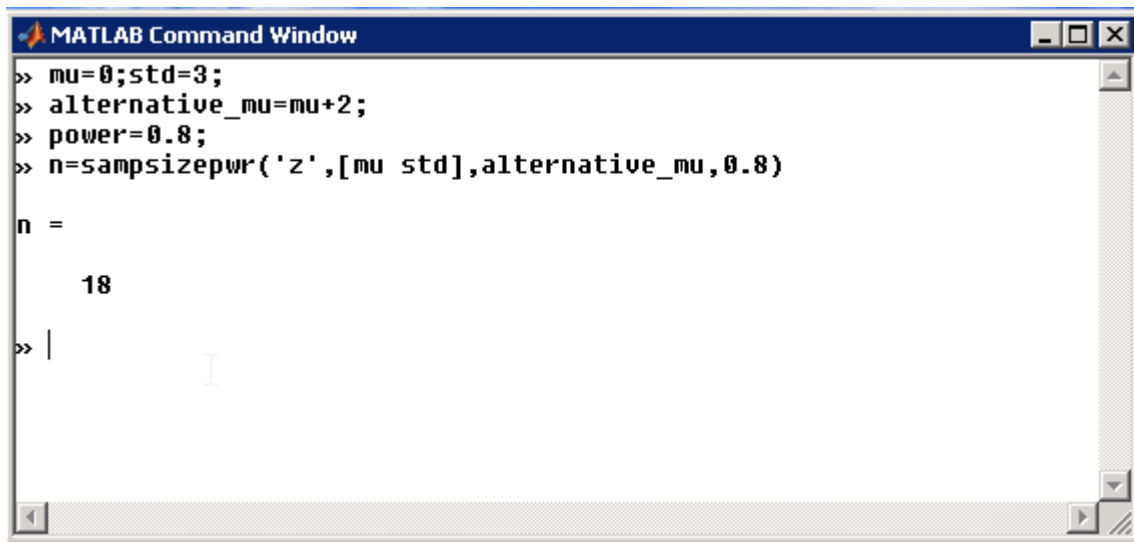
 followed by a new command prompt

```
>> |
```

We get exactly the same result! That's a good thing!

Sample size of a one sample Z test

And what about the sample size? For this purpose we also use the function **sampsizepwr()**:

A screenshot of the MATLAB Command Window. The title bar reads "MATLAB Command Window". The command history shows:

```
>> mu=0;std=3;  
>> alternative_mu=mu+2;  
>> power=0.8;  
>> n=sampsizepwr('z',[mu std],alternative_mu,0.8)
```

 The output is:

```
n =  
  
    18
```

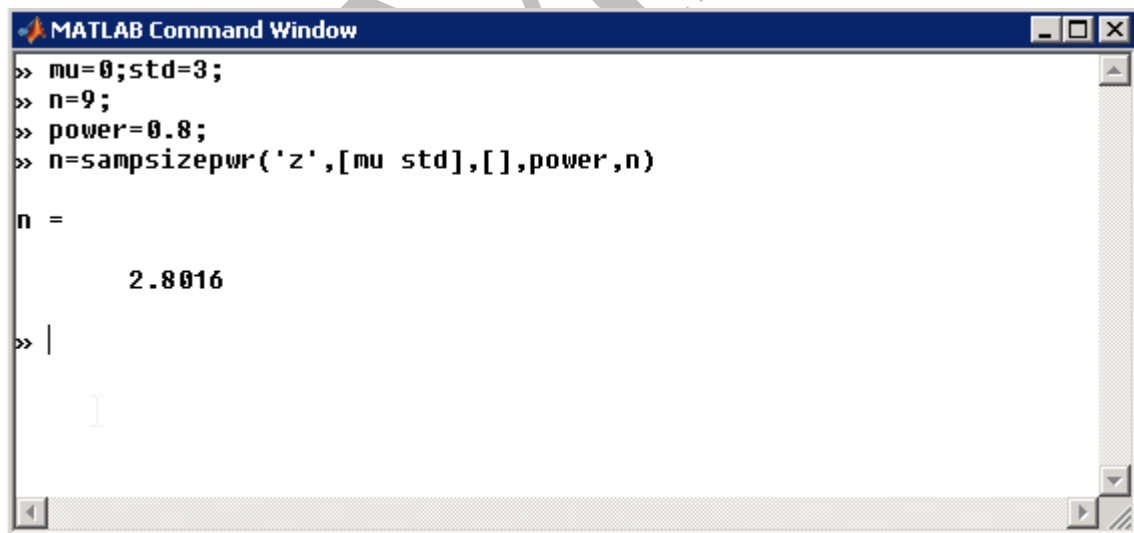
 A cursor is visible on the line following the output.

```
>> |
```

We also get the same result!

Difference (resolution) of a one sample Z test

And what about the difference (resolution) of the test? For this purpose we also use the function **sampsizepwr()**:

A screenshot of the MATLAB Command Window. The title bar reads "MATLAB Command Window". The command history shows:

```
>> mu=0;std=3;  
>> n=9;  
>> power=0.8;  
>> n=sampsizepwr('z',[mu std],[],power,n)
```

 The output is:

```
n =  
  
    2.8016
```

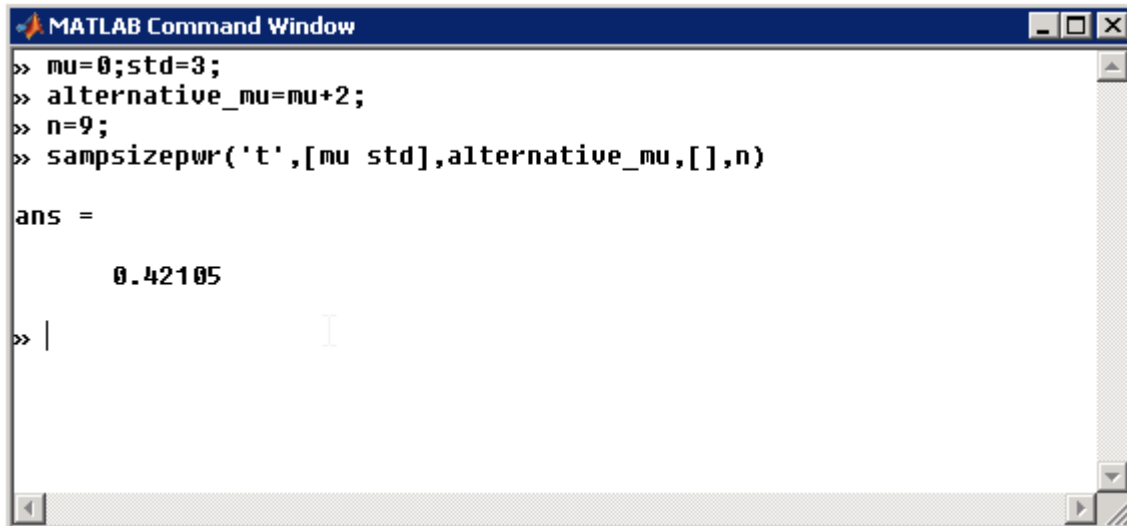
 A cursor is visible on the line following the output.

```
>> |
```

We also get the same result!

Power of a t test

In the theoretical statistic course we also made the proofs for the t-test. Then have a look if we found the same results as the one calculated by hand and in Minitab always by using the function **sampsizepwr**(:



```
MATLAB Command Window
>> mu=0;std=3;
>> alternative_mu=mu+2;
>> n=9;
>> sampsizepwr('t',[mu std],alternative_mu,[],n)

ans =

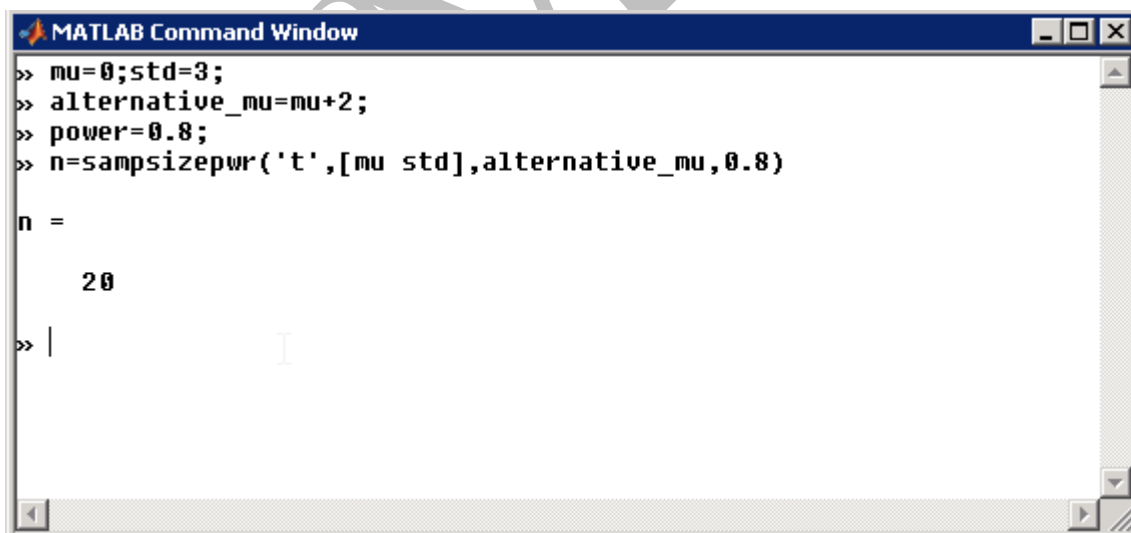
    0.42105

>> |
```

We also get the same result!

Sample size of a one sample t test

And what about the sample size? For this purpose we also use the function **sampsizepwr**():



```
MATLAB Command Window
>> mu=0;std=3;
>> alternative_mu=mu+2;
>> power=0.8;
>> n=sampsizepwr('t',[mu std],alternative_mu,0.8)

n =

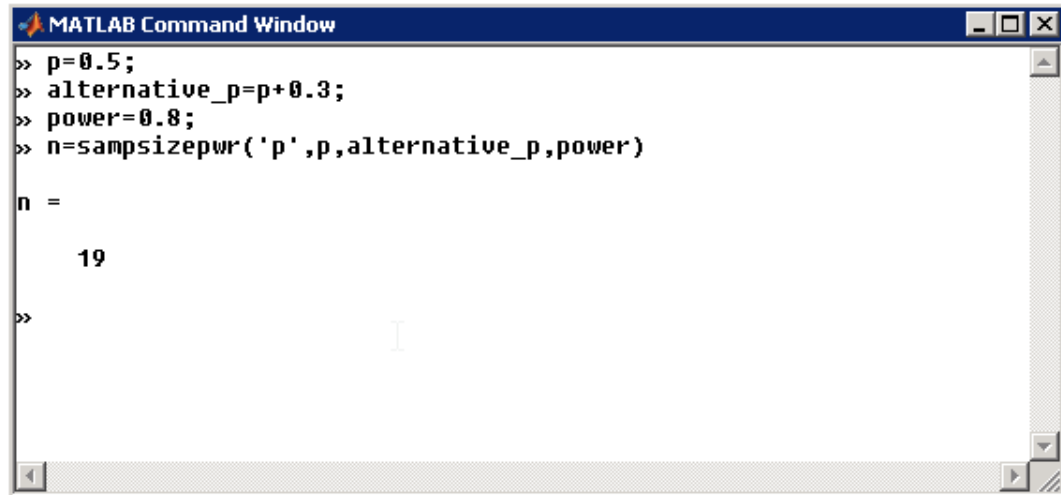
    20

>> |
```

We also get the same result!

Sample size of a one sample p test

In the theoretical statistic course we also made the proofs for the p-test but only for the sample size. Then have a look if we found the same results as the one calculated by hand and in Minitab always by using the function **sampsizepwr**(:



```
MATLAB Command Window
>> p=0.5;
>> alternative_p=p+0.3;
>> power=0.8;
>> n=sampsizepwr('p',p,alternative_p,power)

n =

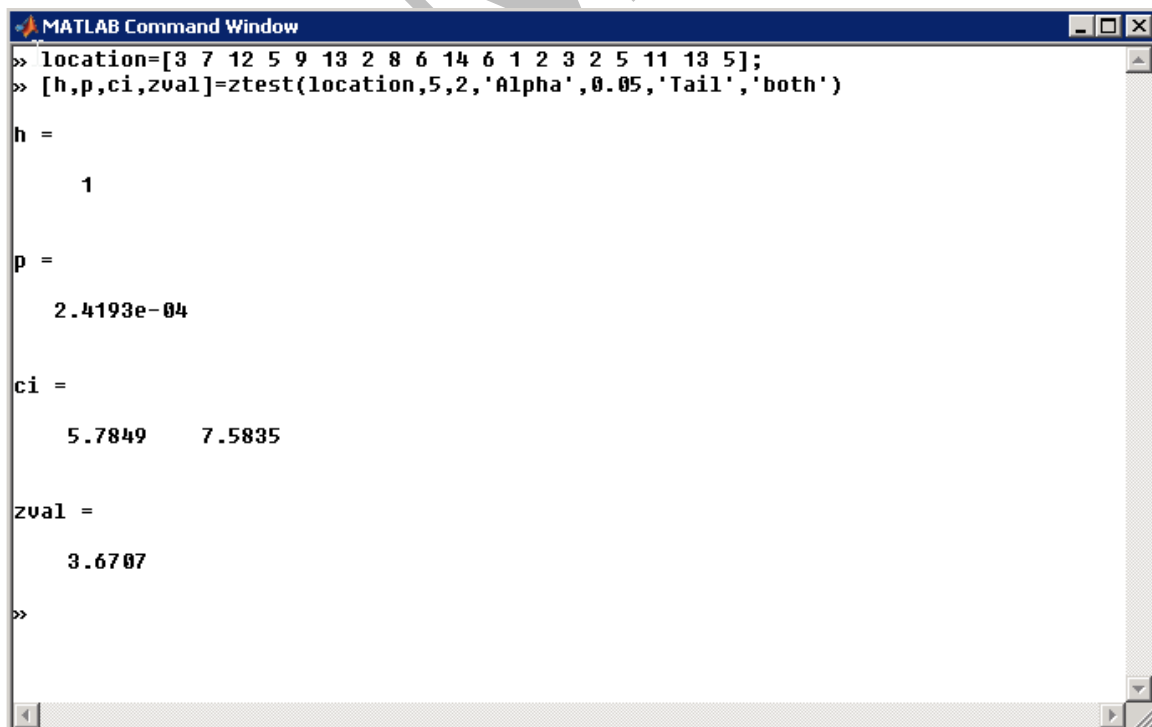
    19

>>
```

That's right!

One sample Z-test

Using the function **ztest**() let us see if we found the same p-value, confidence interval and z value as in Minitab, Excel and the calculations made by hand after the theoretical proof in the statistic course:



```
MATLAB Command Window
>> location=[3 7 12 5 9 13 2 8 6 14 6 1 2 3 2 5 11 13 5];
>> [h,p,ci,zval]=ztest(location,5,2,'Alpha',0.05,'Tail','both')

h =

     1

p =

 2.4193e-04

ci =

    5.7849    7.5835

zval =

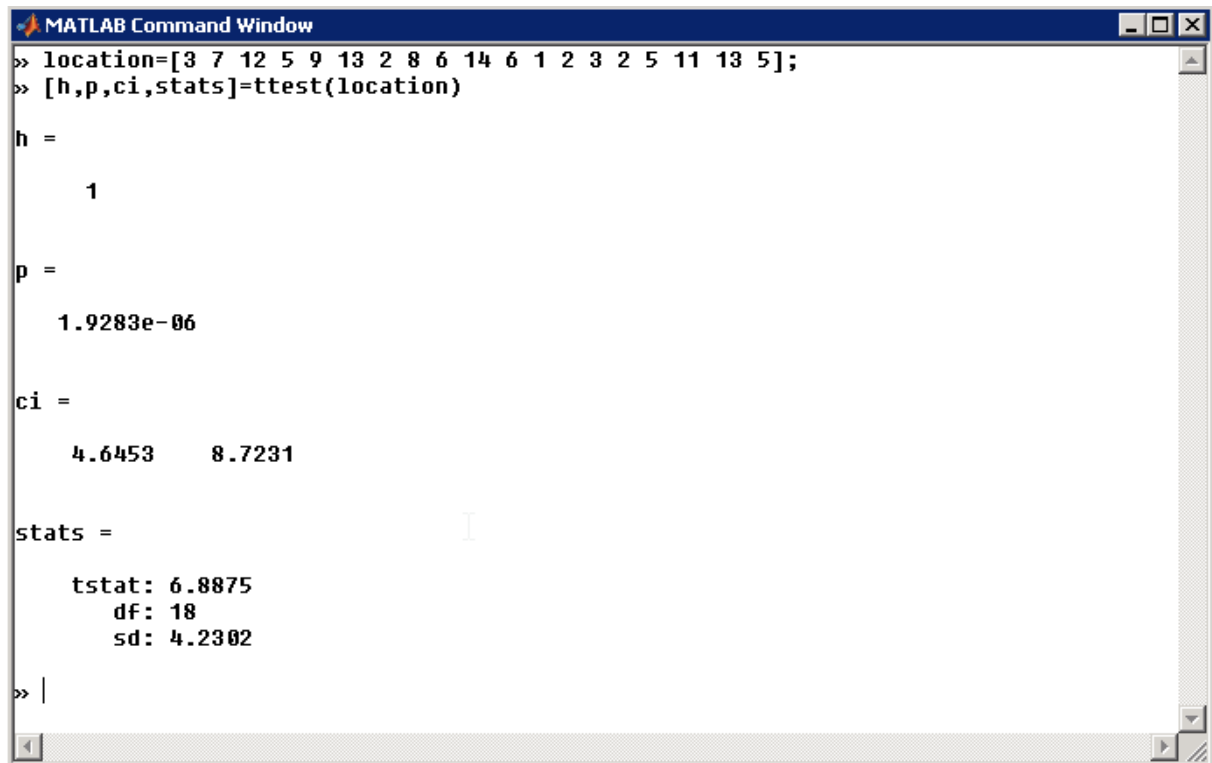
    3.6707

>>
```

Yep! Everything is OK!

One sample t-test

Using the function `ttest()` let us see if we found the same p-value, confidence interval and z value as in Minitab, Excel and the calculations made by hand after the theoretical proof in the statistic course:

A screenshot of the MATLAB Command Window. The window has a blue title bar with the MATLAB logo and the text "MATLAB Command Window". The command prompt shows two lines of code: `location=[3 7 12 5 9 13 2 8 6 14 6 1 2 3 2 5 11 13 5];` and `[h,p,ci,stats]=ttest(location)`. The output shows `h = 1`, `p = 1.9283e-06`, `ci = 4.6453 8.7231`, and `stats =` followed by a sub-structure: `tstat: 6.8875`, `df: 18`, and `sd: 4.2302`. The cursor is at the end of the second line of code.

```
MATLAB Command Window
>> location=[3 7 12 5 9 13 2 8 6 14 6 1 2 3 2 5 11 13 5];
>> [h,p,ci,stats]=ttest(location)

h =

     1

p =

 1.9283e-06

ci =

 4.6453    8.7231

stats =

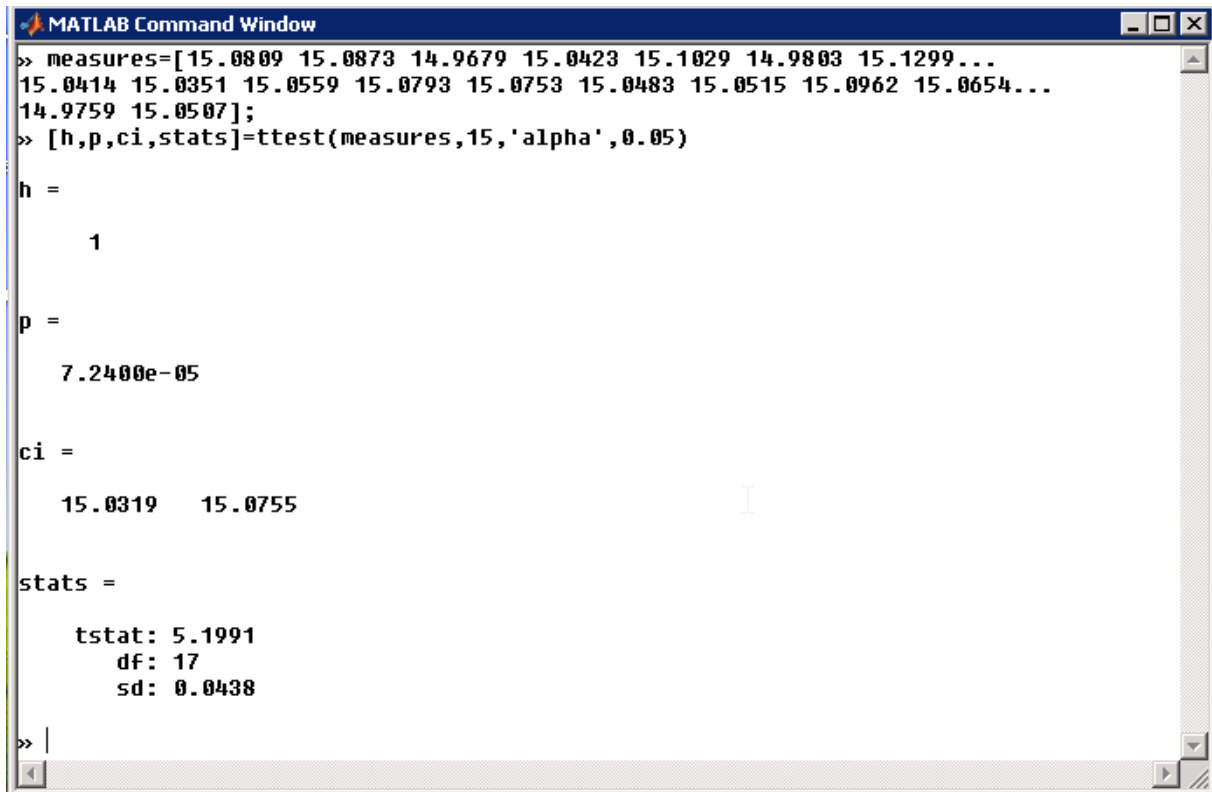
    tstat: 6.8875
         df: 18
         sd: 4.2302

>> |
```

Yep! Everything is OK!

One sample t-test with explicit target mean

And what about the one sample t-test with explicit target mean? Let us check if we found the same result as the one calculated by hand and with Minitab and R?

A screenshot of the MATLAB Command Window. The window title is "MATLAB Command Window". The command prompt shows the following code:

```
>> measures=[15.0809 15.0873 14.9679 15.0423 15.1029 14.9803 15.1299...  
15.0414 15.0351 15.0559 15.0793 15.0753 15.0483 15.0515 15.0962 15.0654...  
14.9759 15.0507];  
>> [h,p,ci,stats]=ttest(measures,15,'alpha',0.05)
```

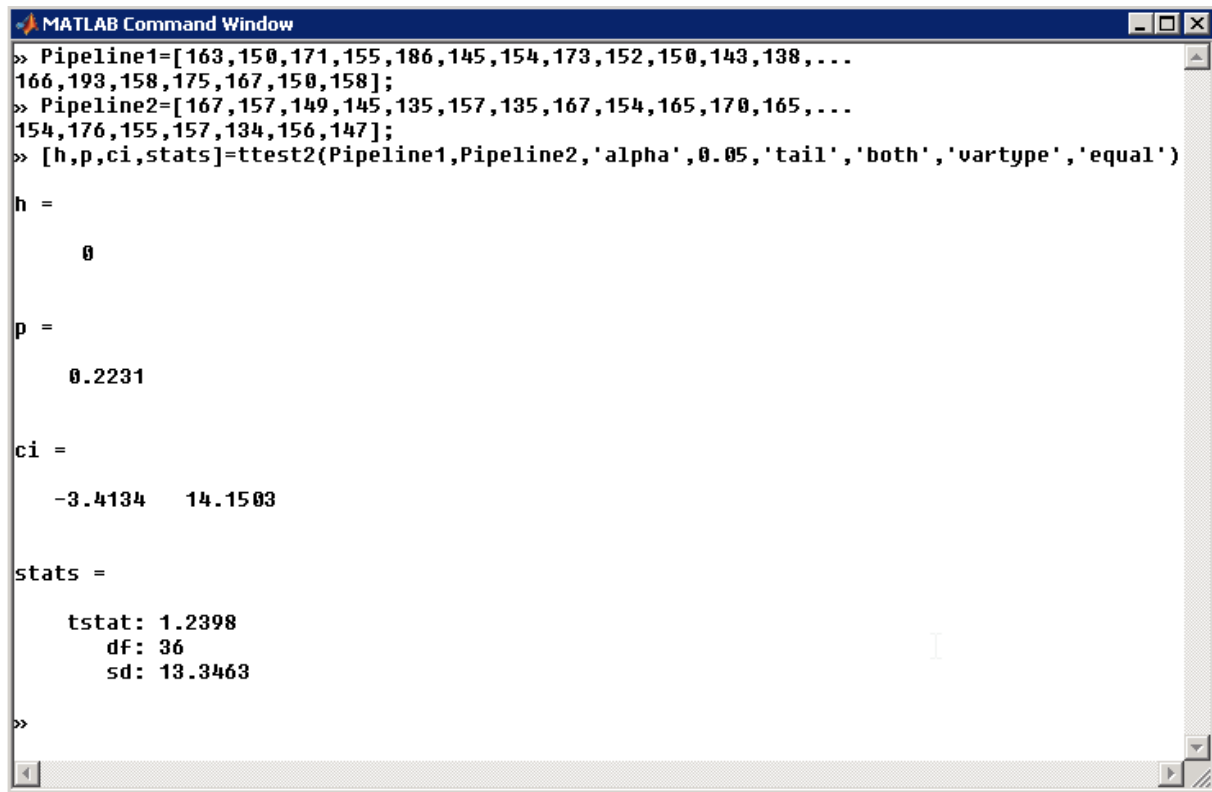
 The output shows:

```
h =  
  
    1  
  
p =  
  
 7.2400e-05  
  
ci =  
  
 15.0319    15.0755  
  
stats =  
  
    tstat: 5.1991  
         df: 17  
         sd: 0.0438  
  
>> |
```

Yes we have the same result!

Two sample homoscedastic t-test

He have proved during the theoretic statistical course how to compare the mean of two unpaired samples under the hypothesis of homoscedasticity. The purpose now is to check if we obtain the same result as by hand on Minitab and R using the function **testt2()**:



```
MATLAB Command Window
>> Pipeline1=[163,150,171,155,186,145,154,173,152,150,143,138,...
166,193,158,175,167,150,158];
>> Pipeline2=[167,157,149,145,135,157,135,167,154,165,170,165,...
154,176,155,157,134,156,147];
>> [h,p,ci,stats]=testt2(Pipeline1,Pipeline2,'alpha',0.05,'tail','both','vartype','equal')

h =

     0

p =

     0.2231

ci =

    -3.4134    14.1503

stats =

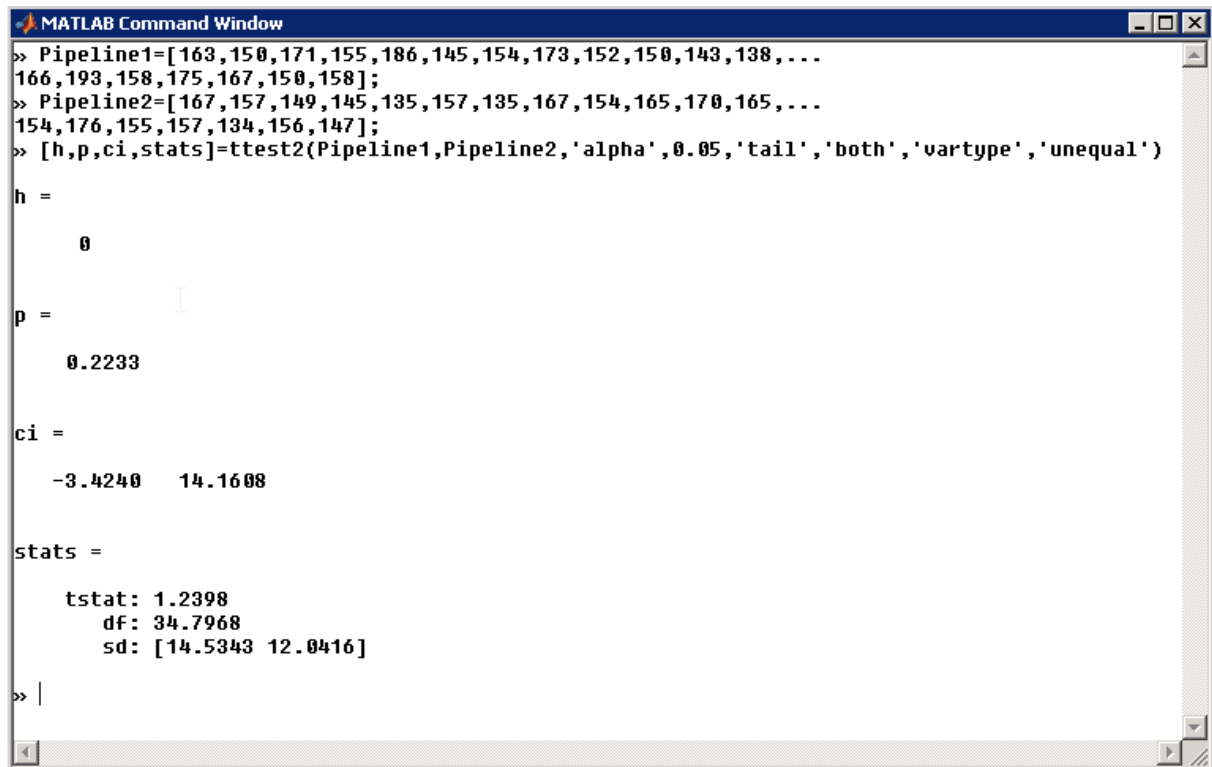
    tstat: 1.2398
         df: 36
         sd: 13.3463

>>
```

...and yes! Everyhting correspond!

Two sample heteroscedastic t-test (Welch test)

He have also proved in the theoretical course how to manage the case if the two samples don't have the same standard deviation. Then the puprose here is to check of we get the same results as those obtained by hand or with Minitab and R:



```
MATLAB Command Window
>> Pipeline1=[163,150,171,155,186,145,154,173,152,150,143,138,...
166,193,158,175,167,150,158];
>> Pipeline2=[167,157,149,145,135,157,135,167,154,165,170,165,...
154,176,155,157,134,156,147];
>> [h,p,ci,stats]=ttest2(Pipeline1,Pipeline2,'alpha',0.05,'tail','both','vartype','unequal')

h =
    0

p =
    0.2233

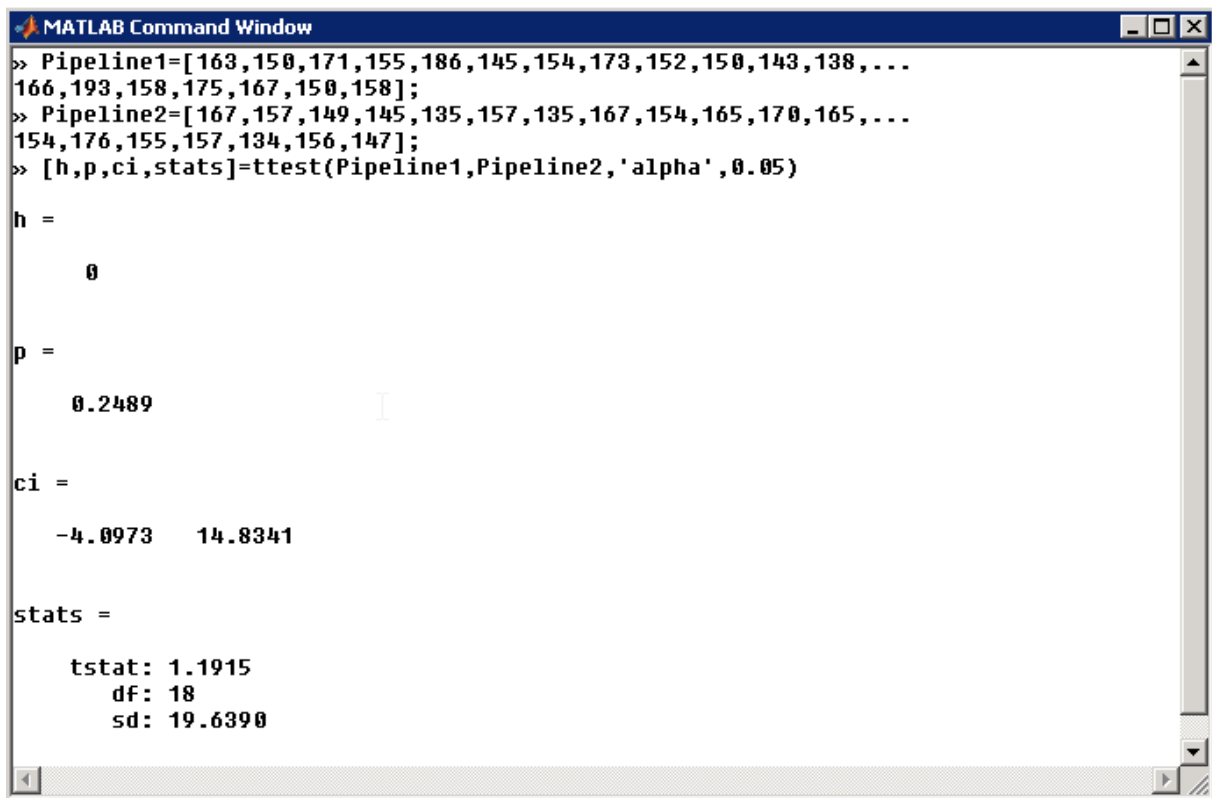
ci =
   -3.4240    14.1608

stats =
    tstat: 1.2398
         df: 34.7968
         sd: [14.5343 12.0416]

>> |
```

Paired t-test

We have also proved in the theoretical course how to compare the mean of two samples if the samples are paired. We will see now if we obtain the same calculations as by hand and Minitab and R:

A screenshot of the MATLAB Command Window. The window has a blue title bar with the MATLAB logo and the text "MATLAB Command Window". The command prompt shows two vectors, Pipeline1 and Pipeline2, being defined. Pipeline1 is [163, 150, 171, 155, 186, 145, 154, 173, 152, 150, 143, 138, ..., 166, 193, 158, 175, 167, 150, 158]. Pipeline2 is [167, 157, 149, 145, 135, 157, 135, 167, 154, 165, 170, 165, ..., 154, 176, 155, 157, 134, 156, 147]. Then, the command [h,p,ci,stats]=ttest(Pipeline1,Pipeline2,'alpha',0.05) is executed. The output shows h = 0, p = 0.2489, ci = [-4.0973, 14.8341], and stats = tstat: 1.1915, df: 18, sd: 19.6390.

```
MATLAB Command Window
>> Pipeline1=[163,150,171,155,186,145,154,173,152,150,143,138,...
166,193,158,175,167,150,158];
>> Pipeline2=[167,157,149,145,135,157,135,167,154,165,170,165,...
154,176,155,157,134,156,147];
>> [h,p,ci,stats]=ttest(Pipeline1,Pipeline2,'alpha',0.05)

h =

     0

p =

    0.2489

ci =

   -4.0973    14.8341

stats =

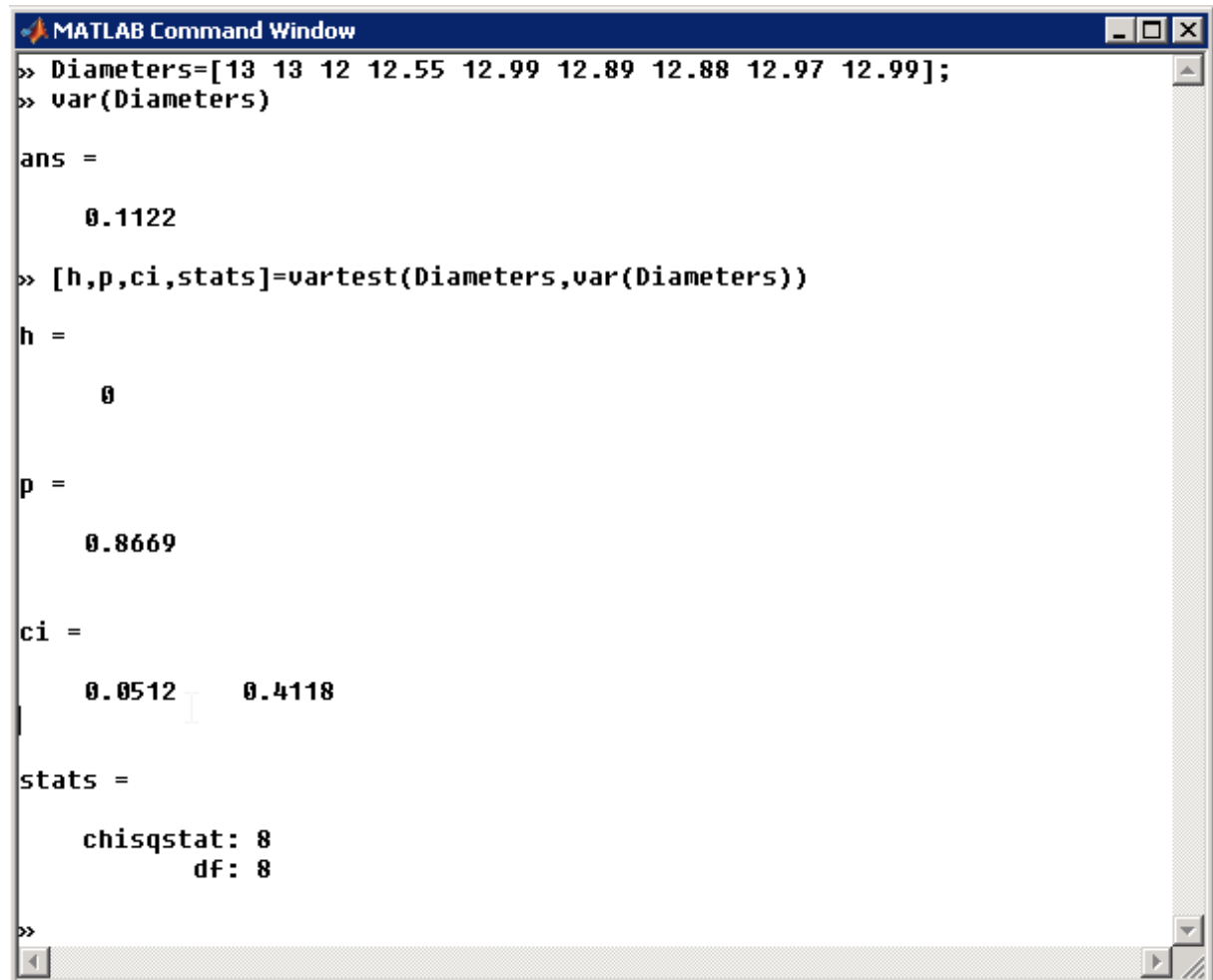
    tstat: 1.1915
         df: 18
         sd: 19.6390
```

And the answer is: Yes! We obtain the same results.

Chi-square variance test

When we finished studying elementary tests for the comparisons of mean in the theoretical statistic course we started (and finished...) proofs on the comparison and interval of confidence of the variance. The first test we saw was the chi-square variance test and can be calculated in MATLAB with the function **vartest()**.

We take the same values as in theoretical course (and R+Minitab course) to check if we obtain the same results:

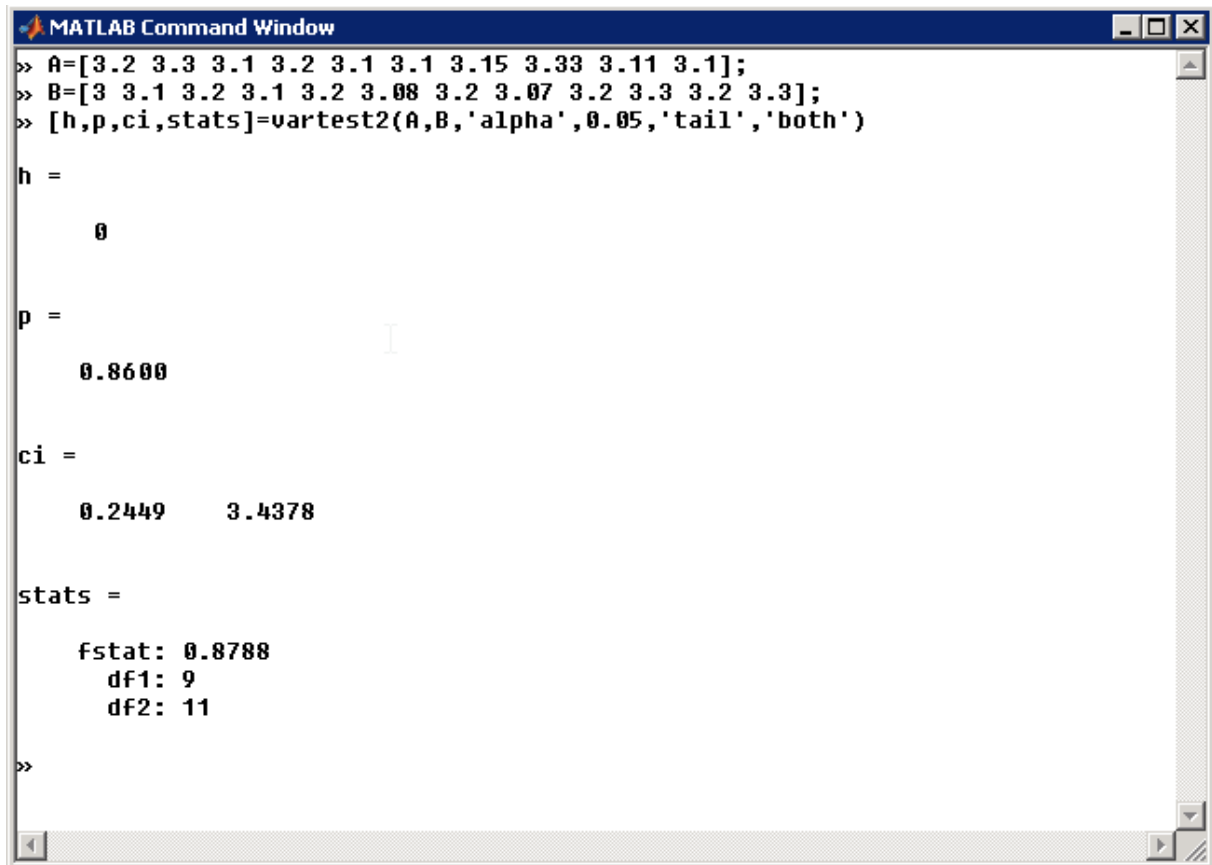
A screenshot of the MATLAB Command Window. The window title is "MATLAB Command Window". The command prompt shows the following sequence of commands and outputs:
1. Command: `Diameters=[13 13 12 12.55 12.99 12.89 12.88 12.97 12.99];`
2. Command: `var(Diameters)`
3. Output: `ans =`
4. Output: `0.1122`
5. Command: `[h,p,ci,stats]=vartest(Diameters,var(Diameters))`
6. Output: `h =`
7. Output: `0`
8. Output: `p =`
9. Output: `0.8669`
10. Output: `ci =`
11. Output: `0.0512 0.4118`
12. Output: `stats =`
13. Output: `chisqstat: 8`
14. Output: `df: 8`
The window has a scroll bar on the right and a status bar at the bottom.

And that's it!

Two sample F-test for equality of variances

And what about the ratio of two variances named "Fisher test" as we proved it in the theoretical statistic course?

Taking the same values as for calculated by hand and Minitab and R, we get in MATLAB using the function **vartest2()**:



```
MATLAB Command Window
>> A=[3.2 3.3 3.1 3.2 3.1 3.1 3.15 3.33 3.11 3.1];
>> B=[3 3.1 3.2 3.1 3.2 3.08 3.2 3.07 3.2 3.3 3.2 3.3];
>> [h,p,ci,stats]=vartest2(A,B,'alpha',0.05,'tail','both')

h =

     0

p =

    0.8600

ci =

    0.2449    3.4378

stats =

    fstat: 0.8788
     df1: 9
     df2: 11

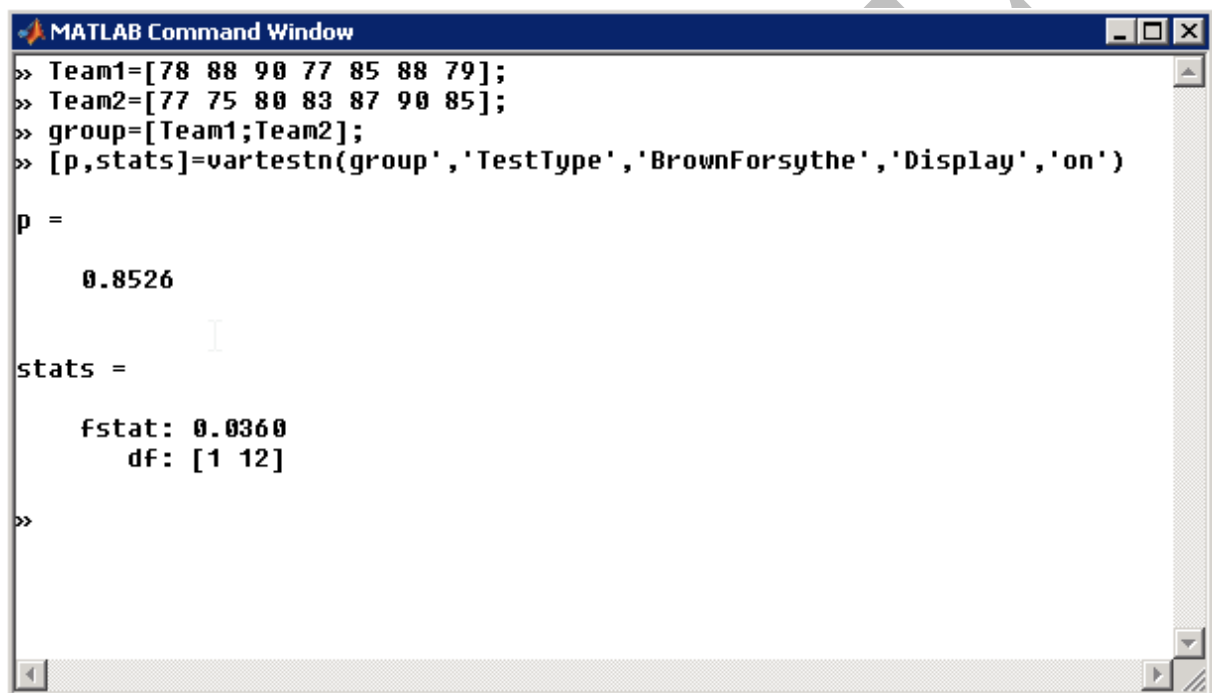
>>
```

Two sample Levene test for equality of variances

We have discussed in the theoretical statistic course about some problems of robustness of the test of the comparison of variance. Different empirical model of robustness were proposed to solving this issue (almost fifty tests classified under the name Test for Homogeneity of the Variance or "THV"). We will see here some of the most famous one available in MATLAB.

Brown-Forsythe (median)

We first begin with the Brown-Forsythe one because this is the only test we checked by making the calculations "by hand" and also with Minitab and R. In MATLAB using the function **vartestn()** with the corresponding arguments we get:



```
MATLAB Command Window
>> Team1=[78 88 90 77 85 88 79];
>> Team2=[77 75 80 83 87 90 85];
>> group=[Team1;Team2];
>> [p,stats]=vartestn(group,'TestType','BrownForsythe','Display','on')

p =

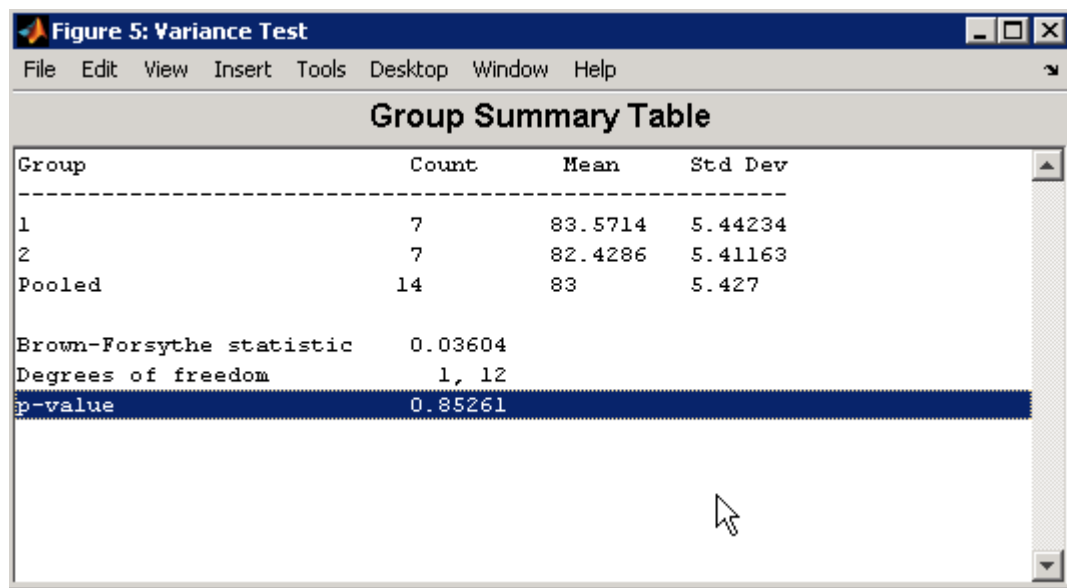
    0.8526

stats =

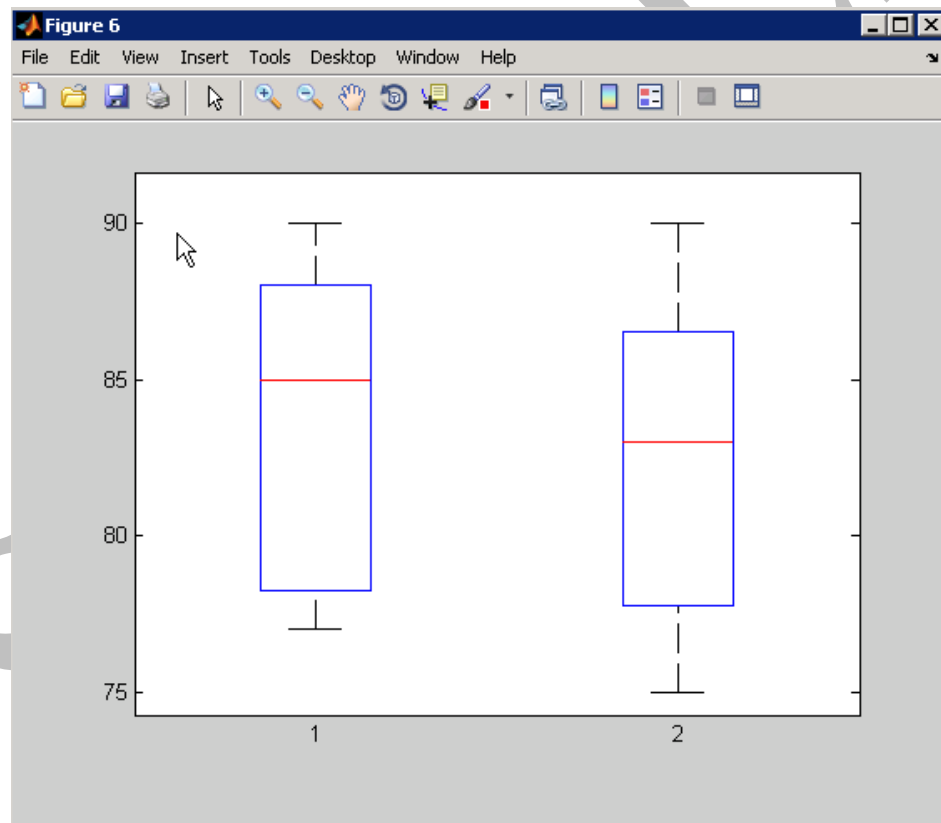
    fstat: 0.0360
         df: [1 12]

>>
```

And the following figures open:



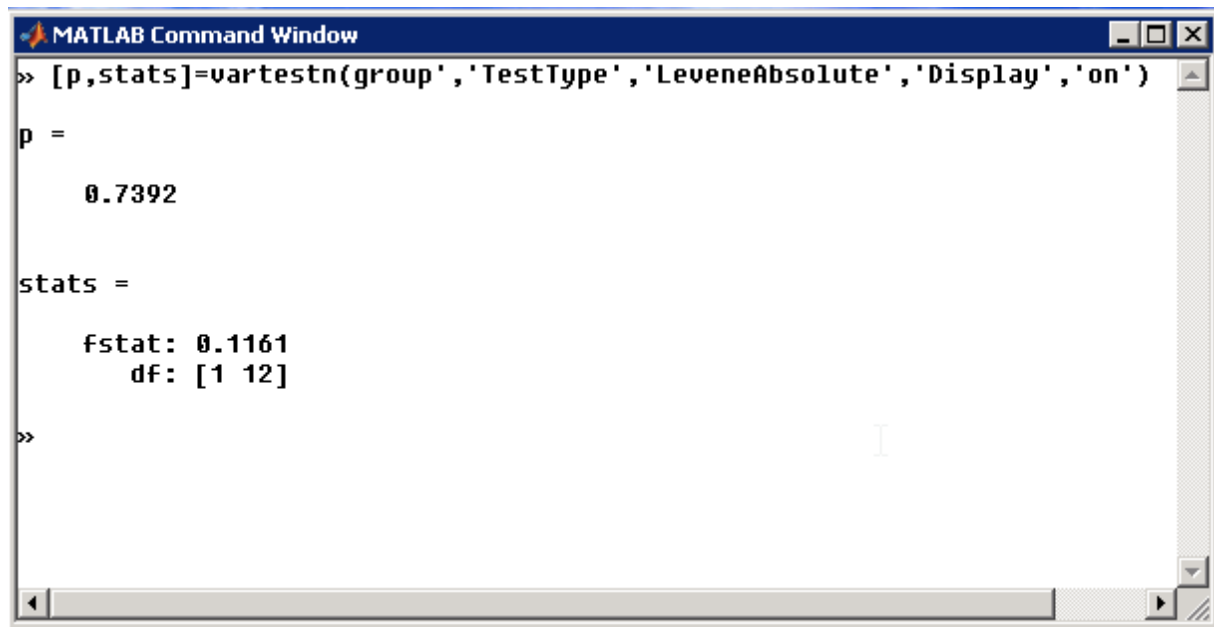
And also the following chart:



We get exactly the same results as the one calculated by hand and with Minitab and R.

Absolute Levene

And is the Levene absolute test available in MATLAB? Yes:



```

MATLAB Command Window
>> [p,stats]=vartestn(group','TestType','LeveneAbsolute','Display','on')

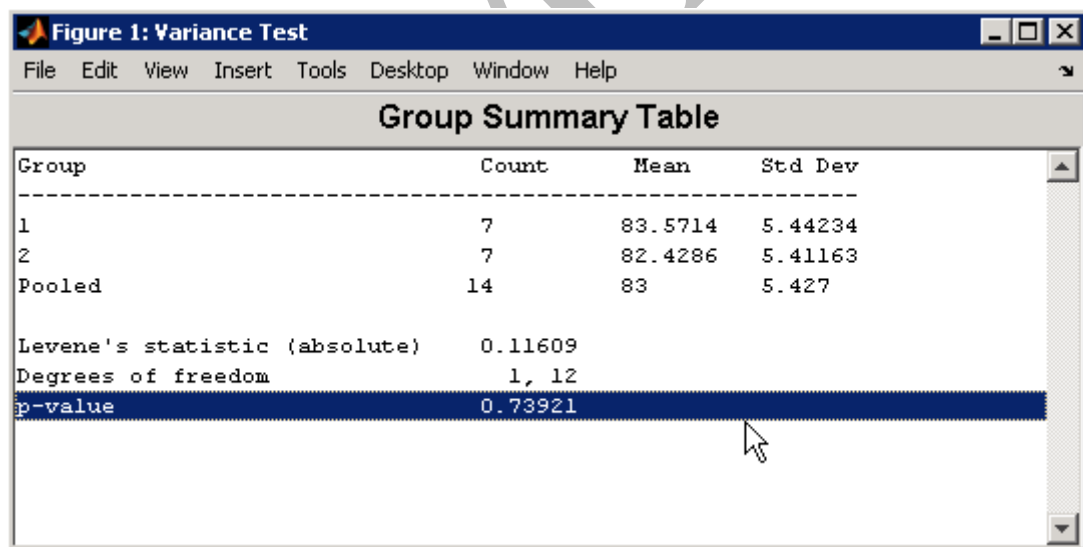
p =

    0.7392

stats =

    fstat: 0.1161
     df: [1 12]
>>
  
```

So we get:

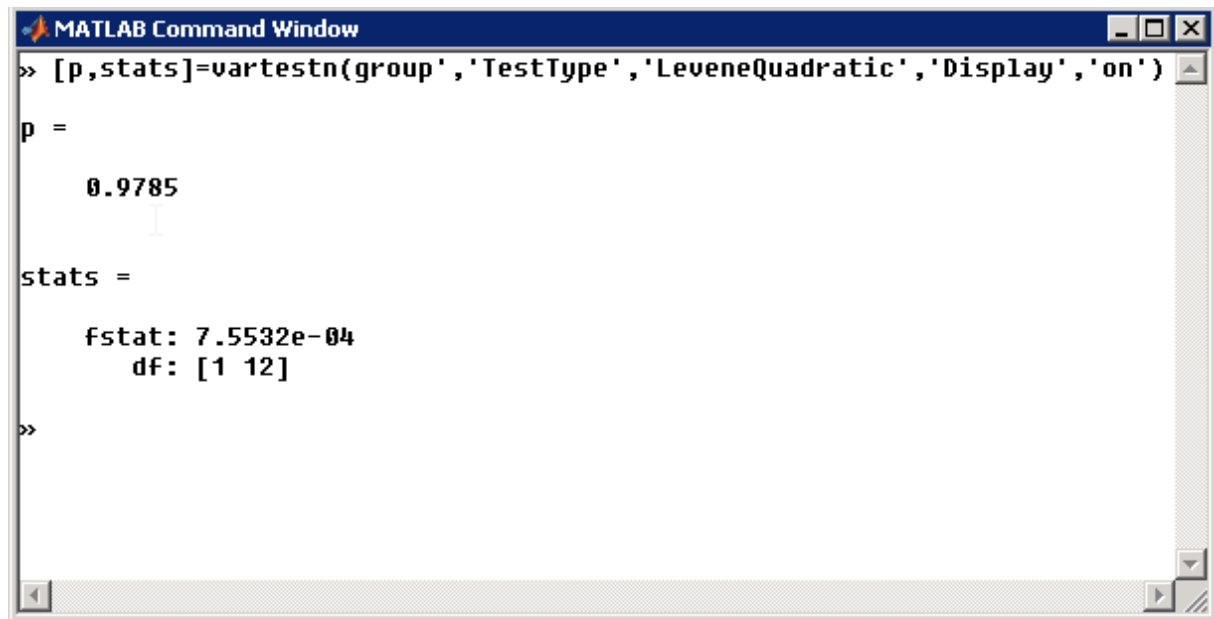


Group	Count	Mean	Std Dev
1	7	83.5714	5.44234
2	7	82.4286	5.41163
Pooled	14	83	5.427

Levene's statistic (absolute)	0.11609
Degrees of freedom	1, 12
p-value	0.73921

Quadratic Levene (square)

And what about the Leven square test? Yes we have it:



```

MATLAB Command Window
>> [p,stats]=vartestn(group','TestType','LeveneQuadratic','Display','on')

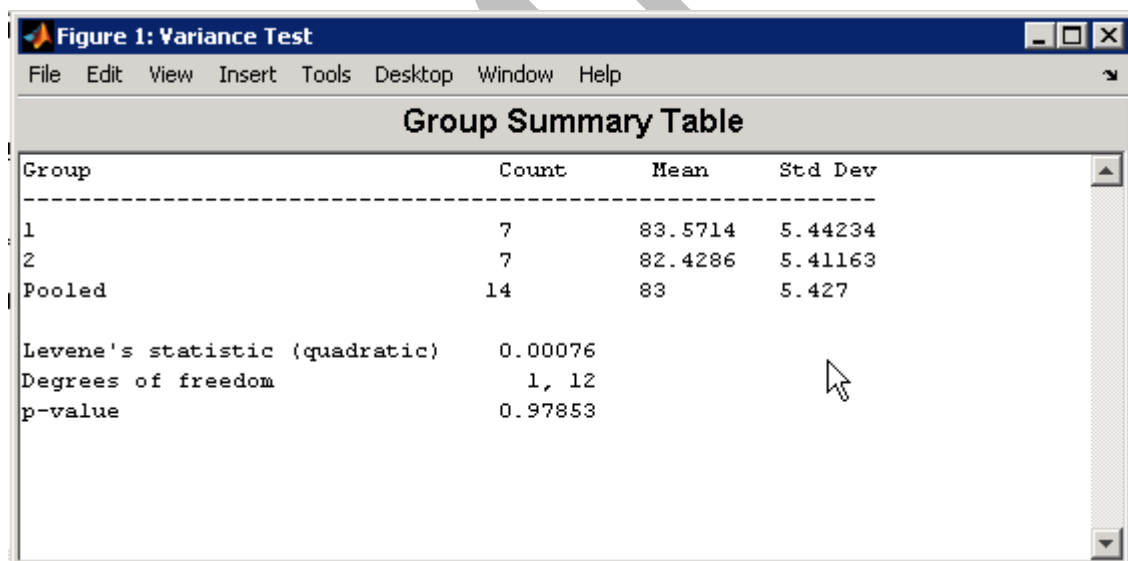
p =

    0.9785

stats =

    fstat: 7.5532e-04
     df: [1 12]
>>
  
```

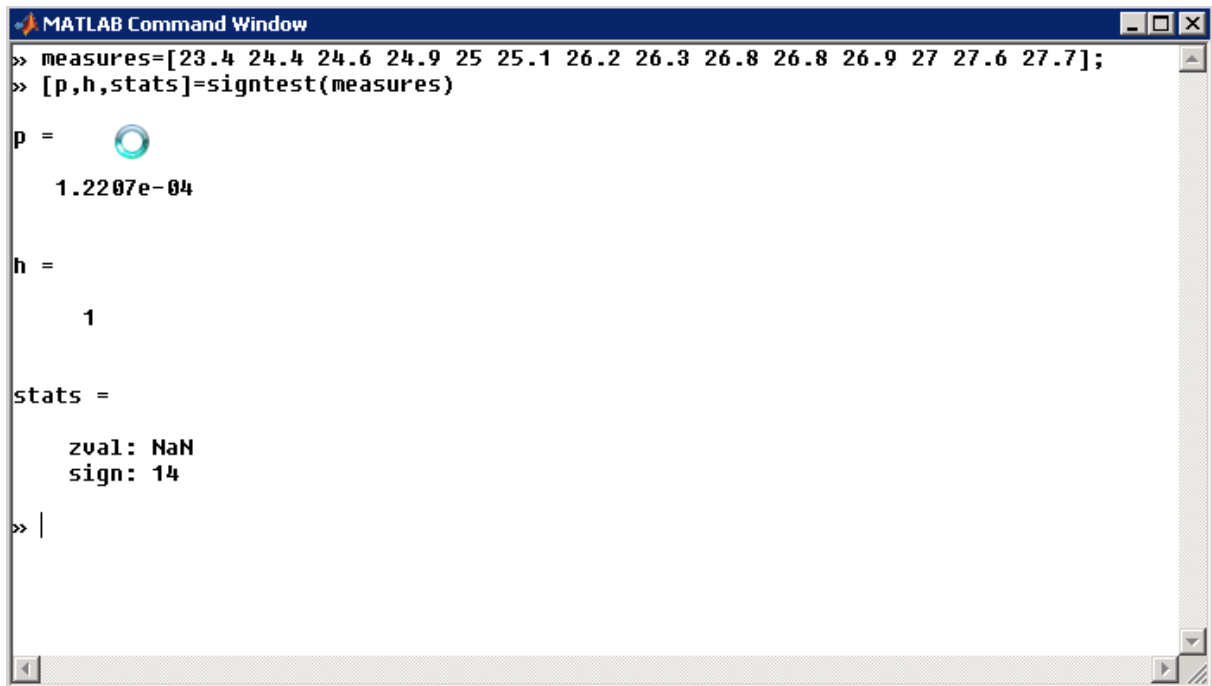
This give us also as output:



We can conclude here that at the opposite of Minitab and R we have much more THV tests available in MATLAB.

Test for the median (one sample sign test)

Now have a look to a very important stuff: the median test with it's confidence interval! For this purpose we will use the **signtest()** function of MATLAB:



```
MATLAB Command Window
>> measures=[23.4 24.4 24.6 24.9 25 25.1 26.2 26.3 26.8 26.8 26.9 27 27.6 27.7];
>> [p,h,stats]=signtest(measures)

p =
    1.2207e-04

h =
     1

stats =
    zval: NaN
    sign: 14

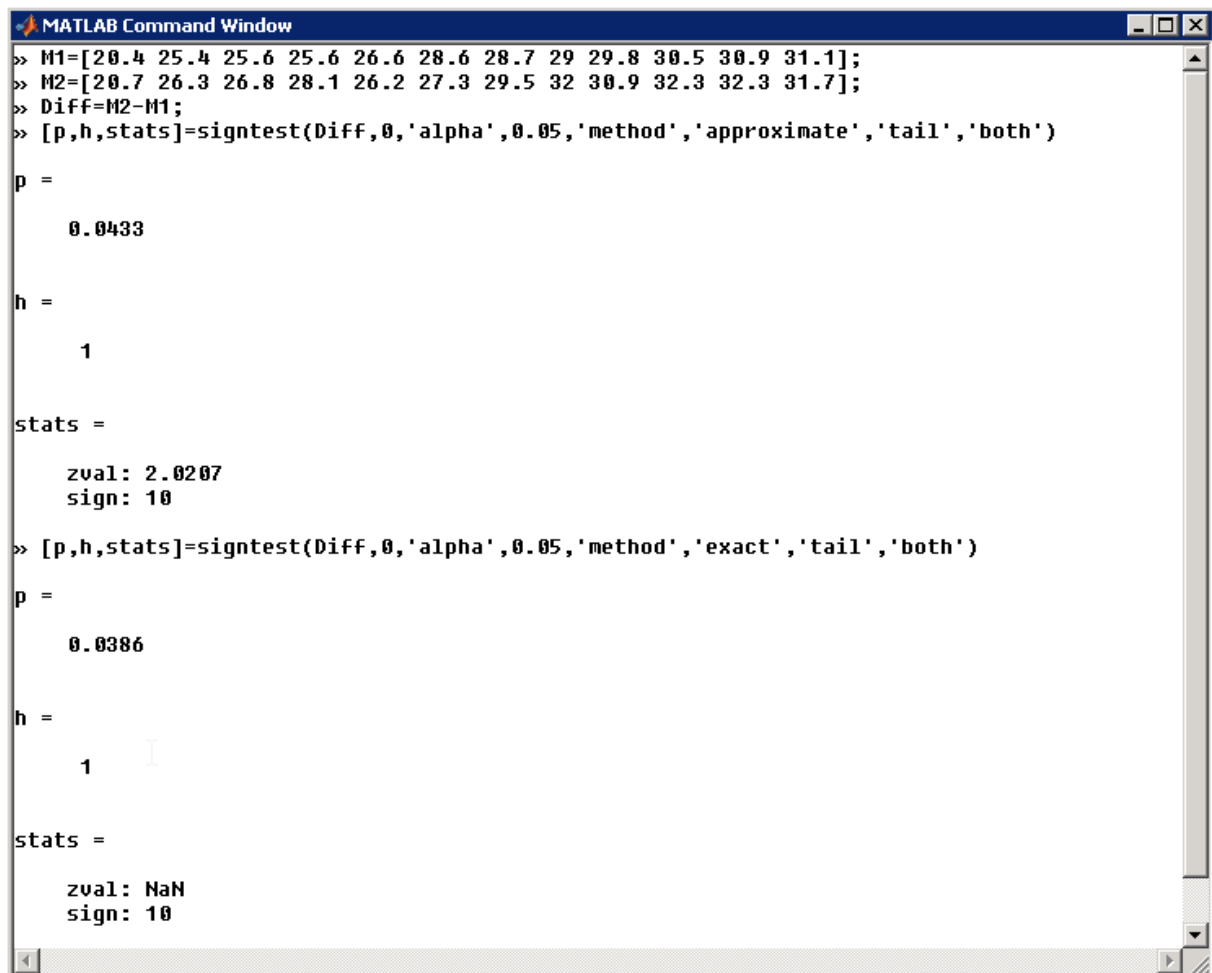
>> |
```

The values are OK and corresponding with those calculated by hand but the very disappointing stuff is that we don't get any confidence interval :-)

For sure, at the opposite of Minitab 15 we get a p-value but at the opposite of R we don't get any C.I.

Binomial signed rank test (dixit: median test paired samples)

We proved in the theoretical statistic course a possible test form comparing the median of two distributions based on the sign of the differences of the two samples. The purpose is to see now if we get the same result as the one calculated by hand or with Minitab.



```
MATLAB Command Window
>> M1=[20.4 25.4 25.6 25.6 26.6 28.6 28.7 29 29.8 30.5 30.9 31.1];
>> M2=[20.7 26.3 26.8 28.1 26.2 27.3 29.5 32 30.9 32.3 32.3 31.7];
>> Diff=M2-M1;
>> [p,h,stats]=signtest(Diff,0,'alpha',0.05,'method','approximate','tail','both')

p =
    0.0433

h =
     1

stats =
    zval: 2.0207
    sign: 10

>> [p,h,stats]=signtest(Diff,0,'alpha',0.05,'method','exact','tail','both')

p =
    0.0386

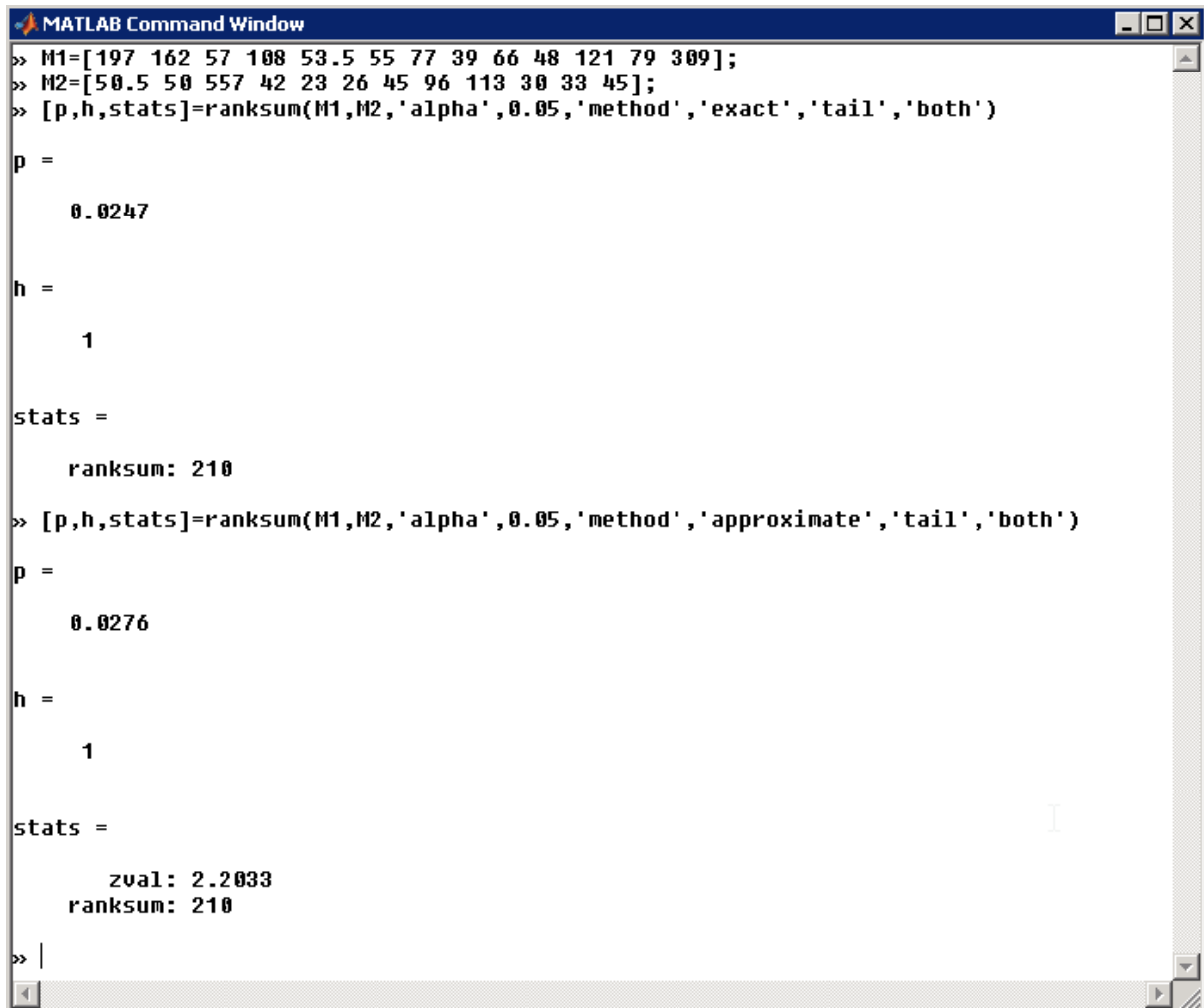
h =
     1

stats =
    zval: NaN
    sign: 10
```

As we can see the second result is in perfect adequation with Minitab and thus with the values calculated by hand in the theoretical course. The very disappointing stuff is that we don't get any confidence interval :-)

Two unpaired samples Wilcoxon (Mann-Withney) rank sum test

Now the purpose is to see if we found the same result with MATLAB as the one with Minitab and this calculated by hand using the function **ranksum()**:



```
MATLAB Command Window
>> M1=[197 162 57 108 53.5 55 77 39 66 48 121 79 309];
>> M2=[50.5 50 557 42 23 26 45 96 113 30 33 45];
>> [p,h,stats]=ranksum(M1,M2,'alpha',0.05,'method','exact','tail','both')

p =

    0.0247

h =

     1

stats =

    ranksum: 210

>> [p,h,stats]=ranksum(M1,M2,'alpha',0.05,'method','approximate','tail','both')

p =

    0.0276

h =

     1

stats =

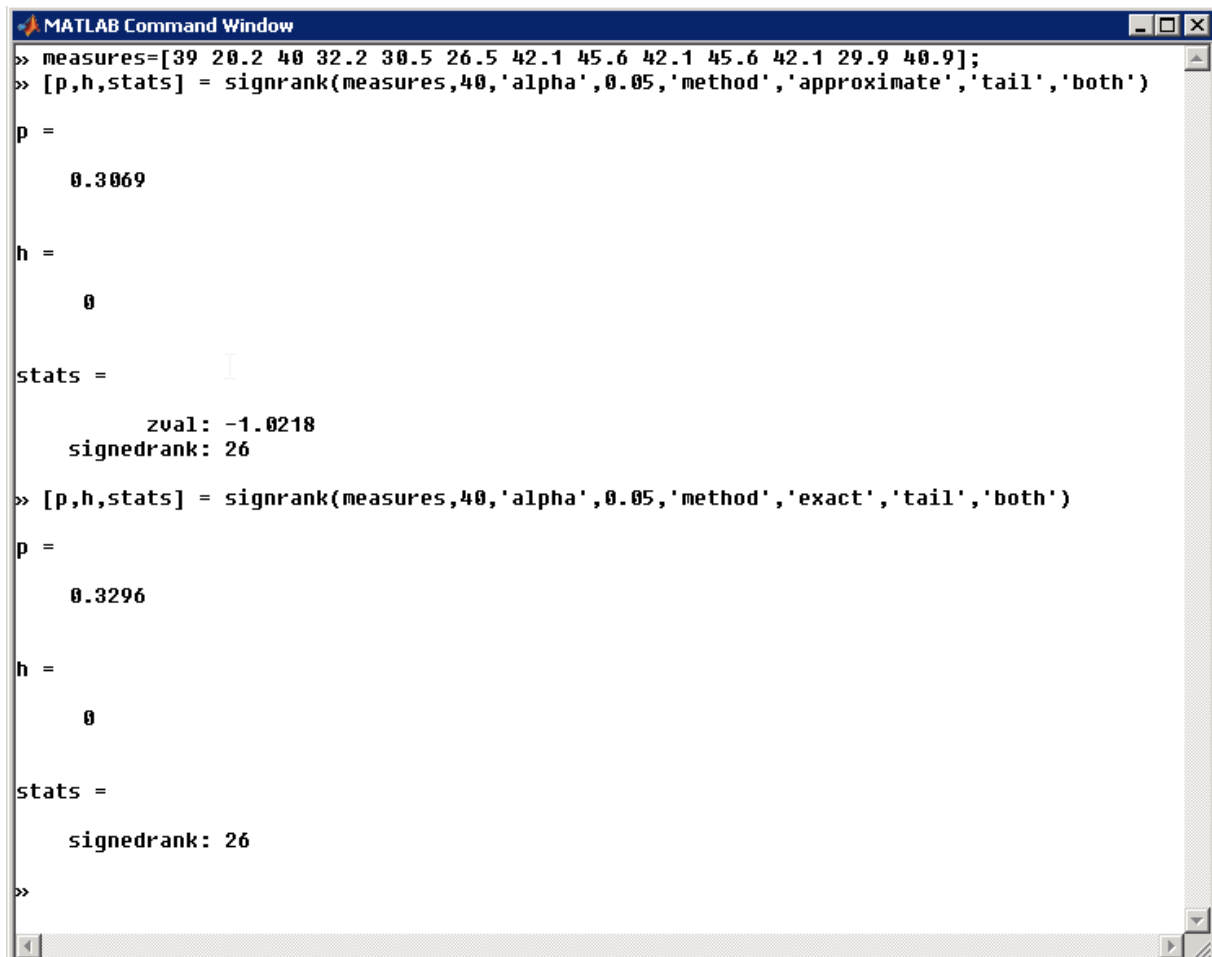
    zval: 2.2033
    ranksum: 210

>> |
```

We get exactly the same result as with Minitab (for the approximate version) and the same small difference as with the calculation made my hand in the theoretical statistic course.

Wilcoxon signed rank test for 1 sample

We also proved in the theoretical statistic course another way to make the test about the median using ranks. The Wilcoxon signed rank test is under the function name **signrank()** in MATLAB:



```
MATLAB Command Window
>> measures=[39 20.2 40 32.2 30.5 26.5 42.1 45.6 42.1 45.6 42.1 29.9 40.9];
>> [p,h,stats] = signrank(measures,40,'alpha',0.05,'method','approximate','tail','both')

p =

    0.3069

h =

     0

stats =

    zval: -1.0218
    signedrank: 26

>> [p,h,stats] = signrank(measures,40,'alpha',0.05,'method','exact','tail','both')

p =

    0.3296

h =

     0

stats =

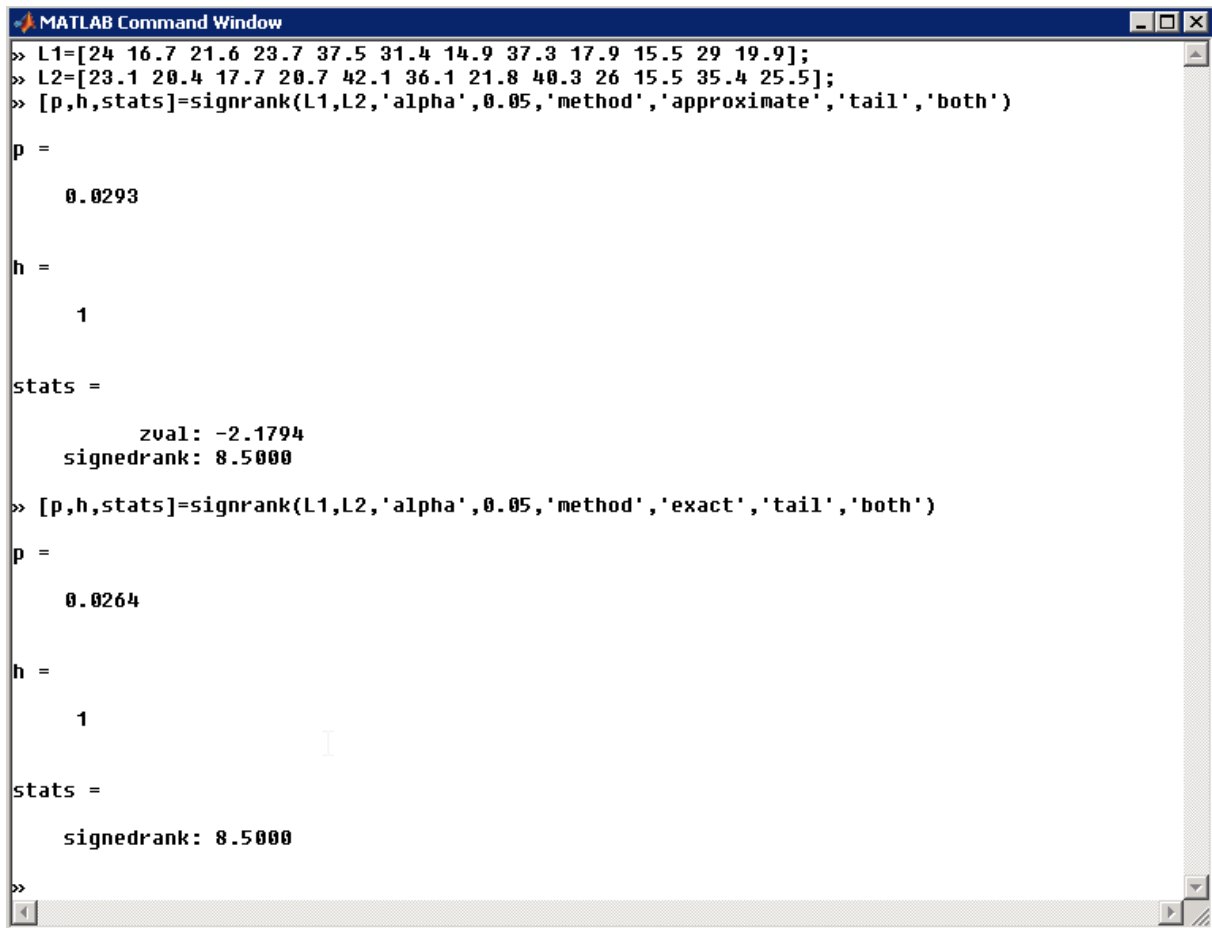
    signedrank: 26

>>
```

The exact method corresponds to what we get with Minitab. The approximate method corresponds to what we get in the calculation by hand.

Wilcoxon signed rank test for two paired samples

And what about the Wilcoxon signed rank test for two paired samples compared to the theoretical statistic course



```
MATLAB Command Window
>> L1=[24 16.7 21.6 23.7 37.5 31.4 14.9 37.3 17.9 15.5 29 19.9];
>> L2=[23.1 20.4 17.7 20.7 42.1 36.1 21.8 40.3 26 15.5 35.4 25.5];
>> [p,h,stats]=signrank(L1,L2,'alpha',0.05,'method','approximate','tail','both')

p =

    0.0293

h =

     1

stats =

    zval: -2.1794
    signedrank: 8.5000

>> [p,h,stats]=signrank(L1,L2,'alpha',0.05,'method','exact','tail','both')

p =

    0.0264

h =

     1

stats =

    signedrank: 8.5000

>>
```

None of the p-values correspond to Minitab or even to R and same thing for the comparison to the values calculate by hand... But the conclusion are the same (we reject the null hypothesis).

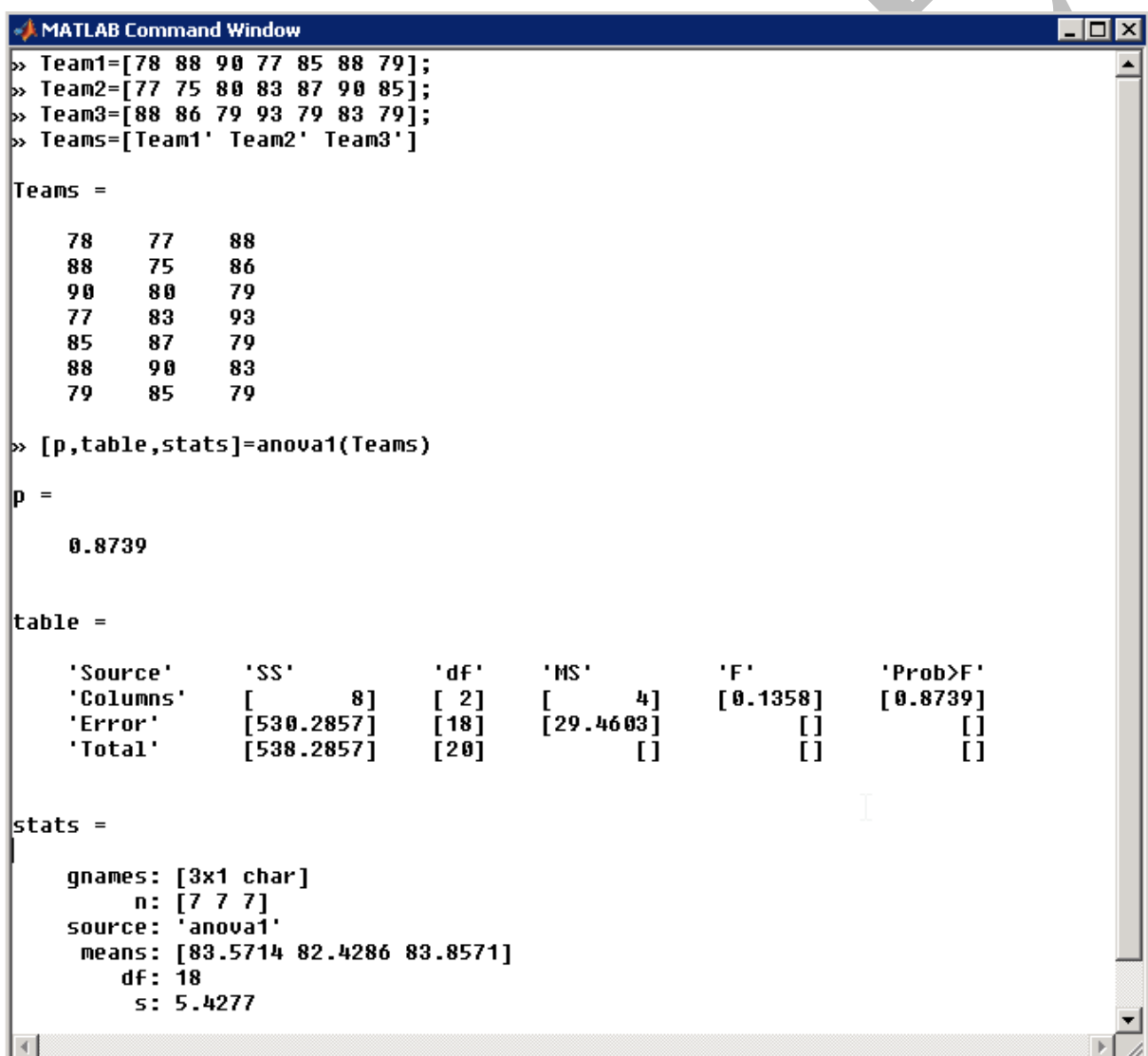
ANOVA

ANOVA (ANalysis Of VAriance) is probably one of the biggest subjects (if it is not the biggest) in the statistic field. You can find books of more than a thousand of pages presenting a lot of different ANOVA models. The purpose here is just to see the ANOVA models that are used and proved at the Master graduate level in most schools worldwide.

One-way fixed factor canonical ANOVA

The one-way fixed factor canonical ANOVA is the simplest existing ANOVA model. The purpose here will be like always to see if we obtain the same results as the one calculated by hand and with Minitab and R.

For this purpose, we will use the function `anova1()` of MATLAB:



```

MATLAB Command Window
>> Team1=[78 88 90 77 85 88 79];
>> Team2=[77 75 80 83 87 90 85];
>> Team3=[88 86 79 93 79 83 79];
>> Teams=[Team1' Team2' Team3']

Teams =

    78    77    88
    88    75    86
    90    80    79
    77    83    93
    85    87    79
    88    90    83
    79    85    79

>> [p,table,stats]=anova1(Teams)

p =

    0.8739

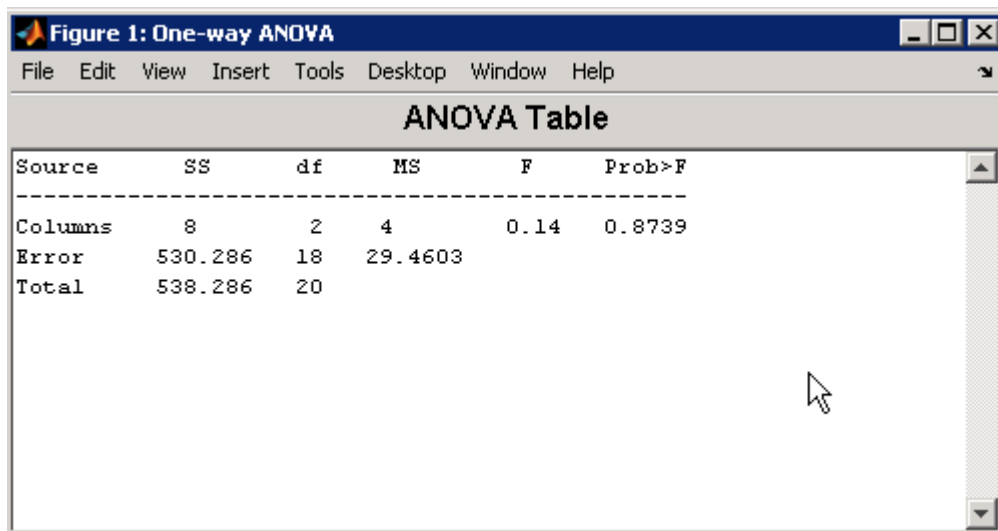
table =

    'Source'    'SS'    'df'    'MS'    'F'    'Prob>F'
    'Columns'   [     8]   [  2]   [    4]   [0.1358] [0.8739]
    'Error'     [530.2857] [ 18] [29.4603]   []   []
    'Total'     [538.2857] [ 20]   []   []   []

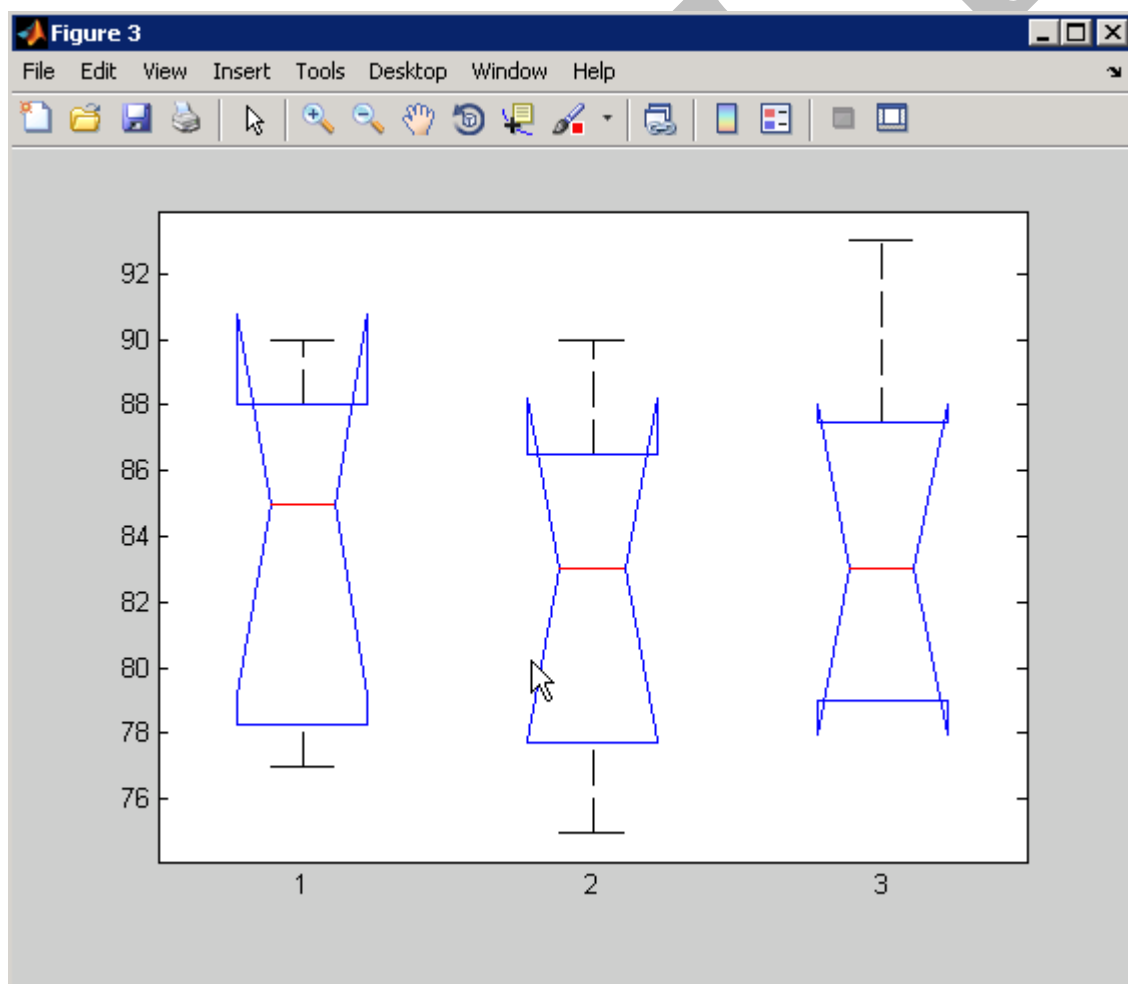
stats =

    gnames: [3x1 char]
         n: [7 7 7]
    source: 'anova1'
     means: [83.5714 82.4286 83.8571]
         df: 18
         s: 5.4277
  
```

This window opens automatically:

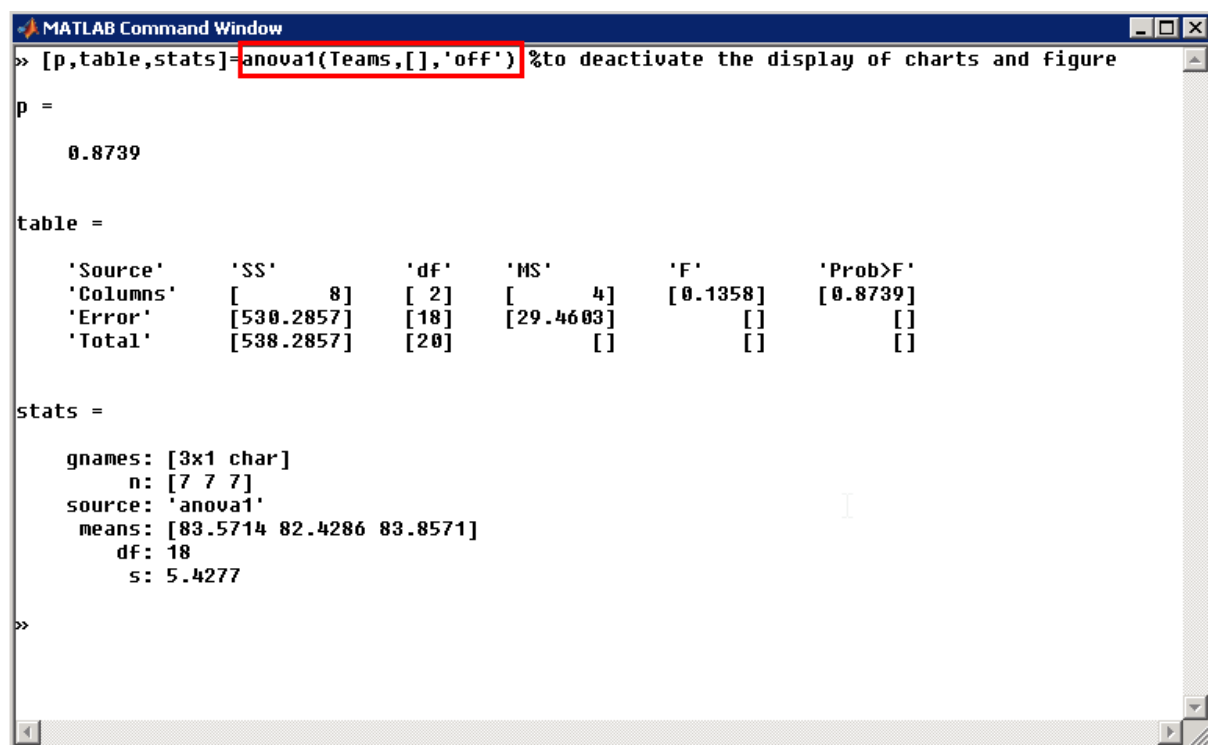


And also this box-plot chart opens also automatically:



What we can conclude is that we get exactly the same results as with Minitab, R and the calculations made by hand.

If we don't want the outputs figure we just have to change the command as following:



The screenshot shows a MATLAB Command Window with the following content:

```
>> [p,table,stats]=anova1(Teams,[],'off') %to deactivate the display of charts and figure
```

The output is as follows:

p =

0.8739

table =

'Source'	'SS'	'df'	'MS'	'F'	'Prob>F'
'Columns'	[8]	[2]	[4]	[0.1358]	[0.8739]
'Error'	[530.2857]	[18]	[29.4603]	[]	[]
'Total'	[538.2857]	[20]	[]	[]	[]

stats =

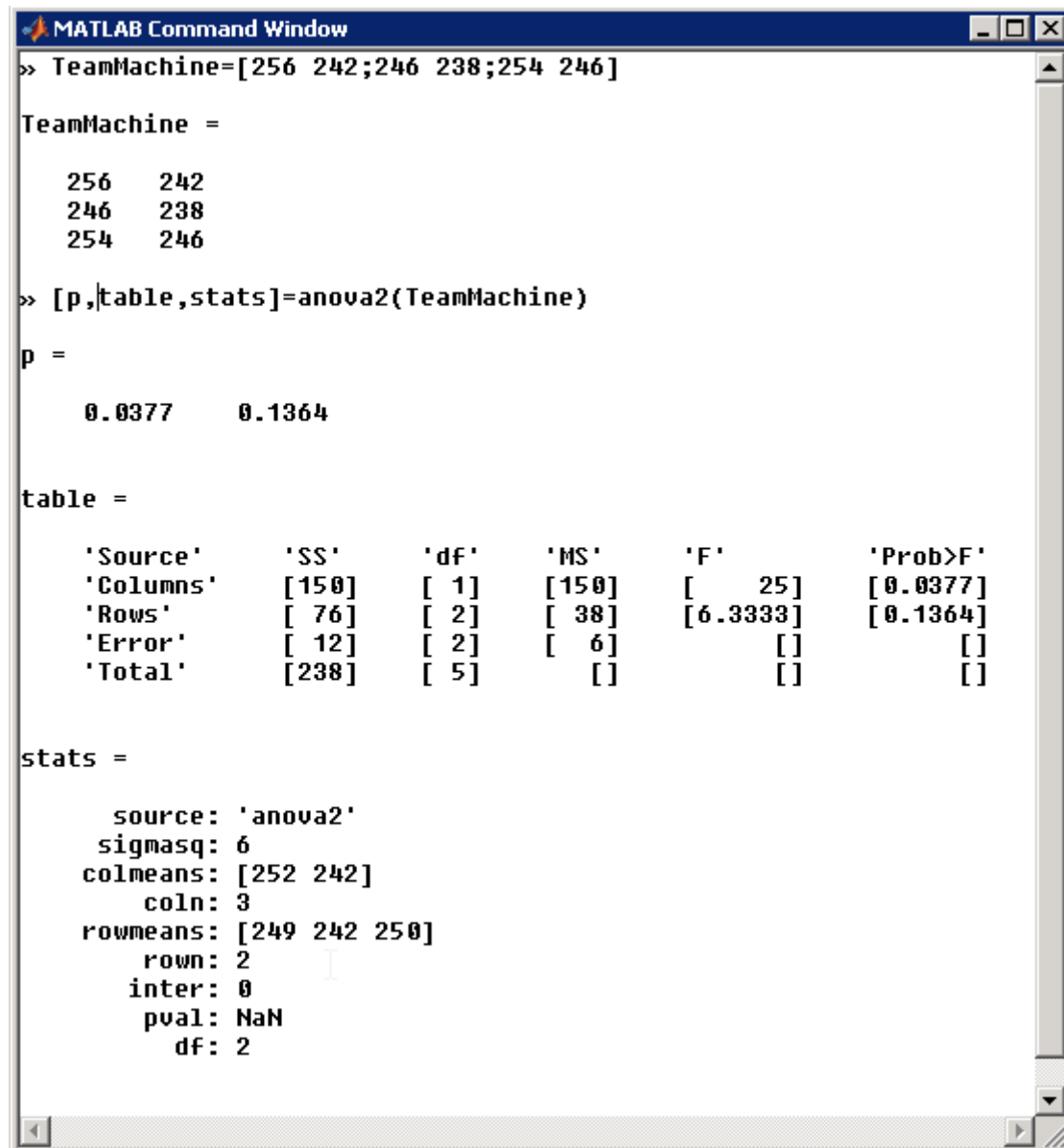
gnames: [3x1 char]
n: [7 7 7]
source: 'anova1'
means: [83.5714 82.4286 83.8571]
df: 18
s: 5.4277

>>

Two-way fixed factors canonical ANOVA

Now have a small look to the two way fixed factors ANOVA without interaction with the small (unrealistic) example we used during the theoretical statistic course to check the correspondence with Minitab, R and the calculations made by hand.

For this purpose we will use the function `anova2()` of MATLAB:



```

>> TeamMachine=[256 242;246 238;254 246]

TeamMachine =

    256    242
    246    238
    254    246

>> [p,table,stats]=anova2(TeamMachine)

p =

    0.0377    0.1364

table =

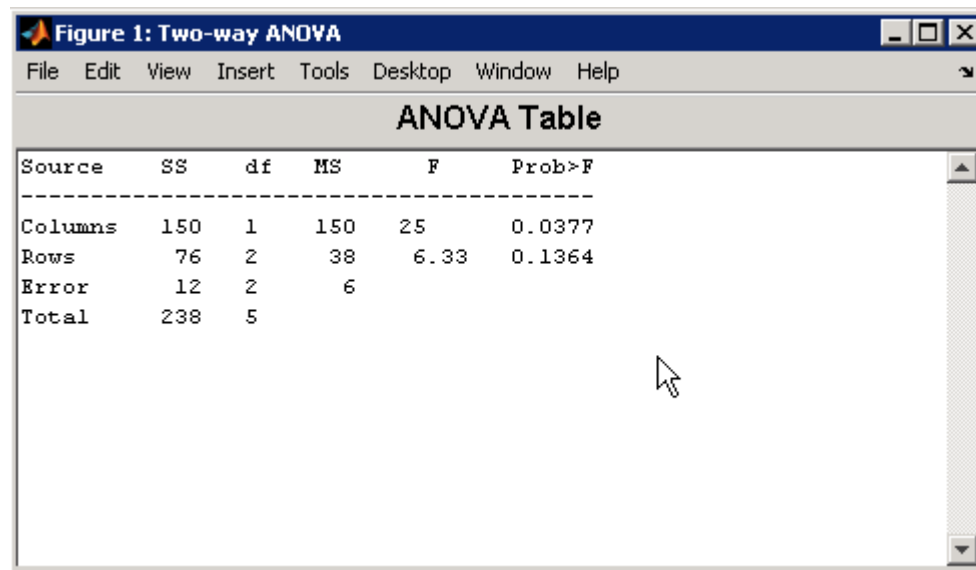
    'Source'    'SS'    'df'    'MS'    'F'    'Prob>F'
    'Columns'   [150]   [ 1]   [150]   [    25] [0.0377]
    'Rows'      [ 76]   [ 2]   [ 38]   [6.3333] [0.1364]
    'Error'     [ 12]   [ 2]   [  6]   [    ]   [    ]
    'Total'     [238]   [ 5]   [    ]   [    ]   [    ]

stats =

    source: 'anova2'
    sigmasq: 6
    colmeans: [252 242]
    coln: 3
    rowmeans: [249 242 250]
    rown: 2
    inter: 0
    pval: NaN
    df: 2

```

This window will open automatically:



And so we can see that we get the same result as Minitab, R and the calculations made by hand.

If we want to deactivate the figure we change the function a little bit:

```

>> [p,table,stats]=anova2(TeamMachine,1,'off')

p =

    0.0377    0.1364

table =

    'Source'    'SS'    'df'    'MS'    'F'    'Prob>F'
    'Columns'   [150]   [ 1]   [150]   [ 25]   [0.0377]
    'Rows'      [ 76]   [ 2]   [ 38]   [6.3333] [0.1364]
    'Error'     [ 12]   [ 2]   [ 6]    []      []
    'Total'     [238]   [ 5]    []      []      []

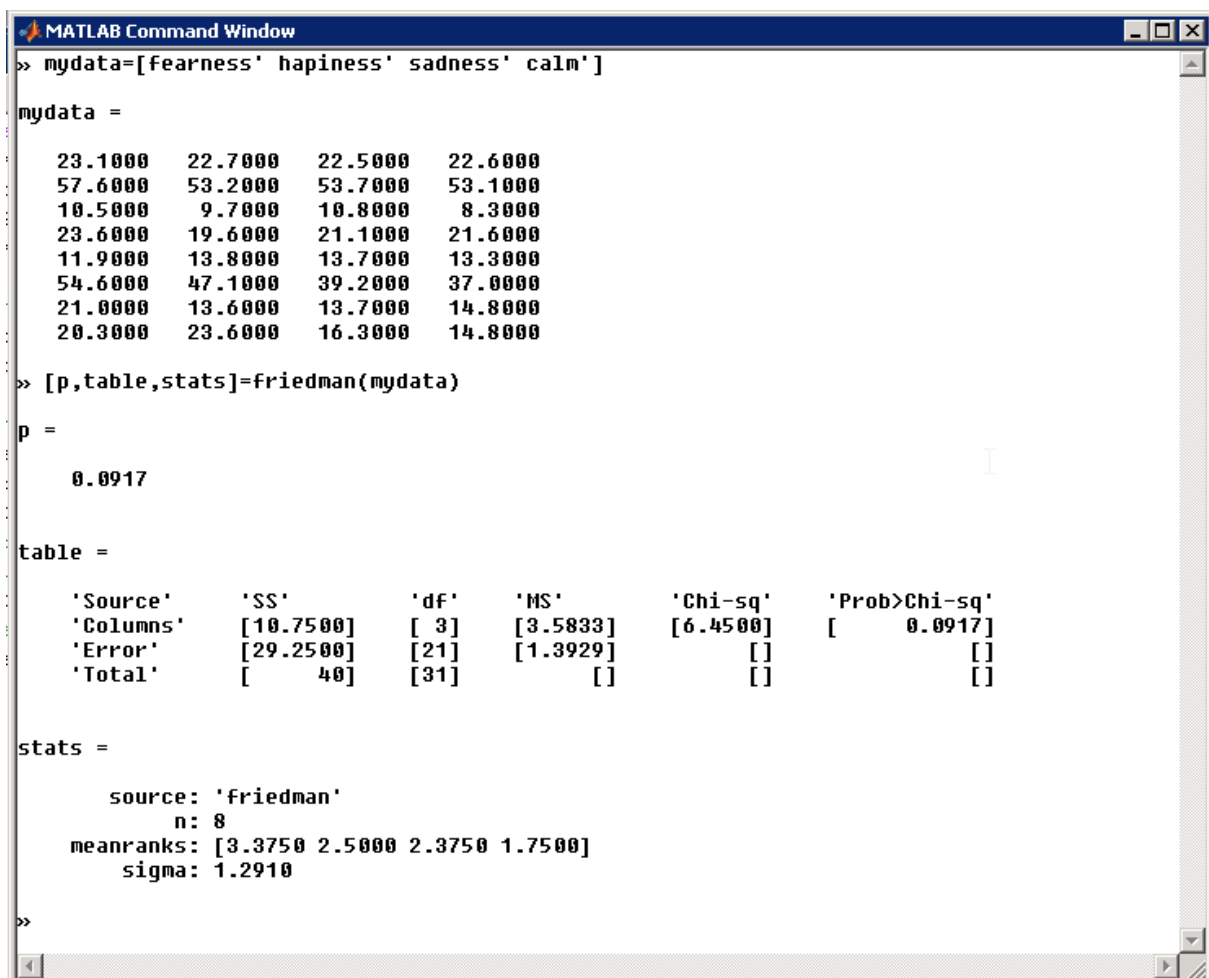
stats =

    source: 'anova2'
    sigmasq: 6
    colmeans: [252 242]
    coln: 3
    rowmeans: [249 242 250]
    rown: 2
  
```

Friedman's test (canonical non-parametric ANOVA based on ranks)

We saw in the theoretical statistic course that the parametric ANOVA can sometimes not be robust to non-normality. For this purpose we have pseudo-proved a way to develop and ranked base non-parametric ANOVA named "Friedman's test".

We will see now using the same datas as the calculations made by hand in MS Excel and also R + Minitab to see if we get the same results using the function **friedman()**:



```

MATLAB Command Window
>> mydata=[fearness' hapiness' sadness' calm']

mydata =

    23.1000    22.7000    22.5000    22.6000
    57.6000    53.2000    53.7000    53.1000
    10.5000     9.7000    10.8000     8.3000
    23.6000    19.6000    21.1000    21.6000
    11.9000    13.8000    13.7000    13.3000
    54.6000    47.1000    39.2000    37.0000
    21.0000    13.6000    13.7000    14.8000
    20.3000    23.6000    16.3000    14.8000

>> [p,table,stats]=friedman(mydata)

p =

    0.0917

table =

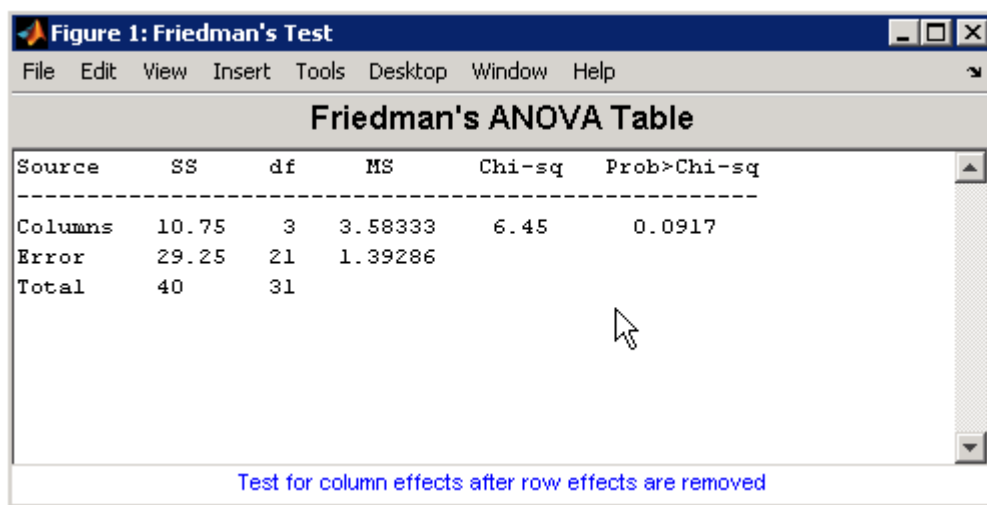
    'Source'      'SS'      'df'      'MS'      'Chi-sq'      'Prob>Chi-sq'
    'Columns'     [10.7500]   [ 3]     [3.5833]    [6.4500]      [ 0.0917]
    'Error'       [29.2500]  [21]     [1.3929]     []            []
    'Total'       [ 40]      [31]     []           []            []

stats =

    source: 'friedman'
         n: 8
  meanranks: [3.3750 2.5000 2.3750 1.7500]
        sigma: 1.2910
  
```

We can see that we get the same values as in theoretical course and more details in comparison of Minitab.

The following windows will also open automatically:



If we don't want to see this figure we just have to change the way we use the function:

```

MATLAB Command Window
>> [p, table, stats]=friedman(mydata,1,'off')
p =
    0.0917
→
table =
    'Source'    'SS'    'df'    'MS'    'Chi-sq'    'Prob>Chi-sq'
    'Columns'   [10.7500] [ 3]    [3.5833]   [6.4500]    [ 0.0917]
    'Error'     [29.2500] [21]    [1.3929]   []          []
    'Total'     [ 40]    [31]    []         []          []

stats =
    source: 'friedman'
         n: 8
  meanranks: [3.3750 2.5000 2.3750 1.7500]
         sigma: 1.2910
>> |
  
```

Kruskal-Wallis test (canonical non-parametric one-way ANOVA)

Once again we will use the original data of the publication of Kruskal and Wallis to see if we get the same results as in the theoretical course:

Standard		Modification 1		Modification 2	
Output	Rank	Output	Rank	Output	Rank
340	5	339	4	347	10
345	9	333	2	343	7
330	1	344	8	349	11
342	6			355	12
338	3				
<i>n</i>		<i>n</i>		<i>n</i>	
5		3		4	
<i>R</i>		<i>R</i>		<i>R</i>	
24		14		40	
<i>R</i> ² / <i>n</i>		65.333		400.	
				Sum	
				12	
				78	
				580.533	

$$\text{Checks : } \sum n = N = 12$$

$$\sum R = \frac{1}{2}N(N+1) = 78$$

$$H = \frac{12 \times 580.533}{12 \times 13} - 3 \times 13 = 5.656 \simeq \chi^2(2)$$

from (1.2)

$$\Pr[\chi^2(2) \geq 5.656] = 0.059$$

from [9] or [13]

$$\Pr[H(5, 4, 3) \geq 5.656] = 0.049$$

from Table 6.1

Thus, using the function **kruskalwallis()** we have:

```

MATLAB Command Window
>> Machine={'Stand' 'Stand' 'Stand' 'Stand' 'Stand' ...
'Modif1' 'Modif1' 'Modif1' 'Modif2' 'Modif2' 'Modif2' 'Modif2'};
>> Production=[340 345 330 342 338 339 333 344 347 343 349 355];
>> [p,tbl,stats] = kruskalwallis(Production,Machine)

p =

    0.0591

tbl =

    'Source'    'SS'    'df'    'MS'    'Chi-sq'    'Prob>Chi-sq'
    'Groups'    [73.5333]    [ 2]    [36.7667]    [5.6564]    [    0.0591]
    'Error'     [69.4667]    [ 9]    [ 7.7185]    []         []
    'Total'     [   143]    [11]    []         []         []

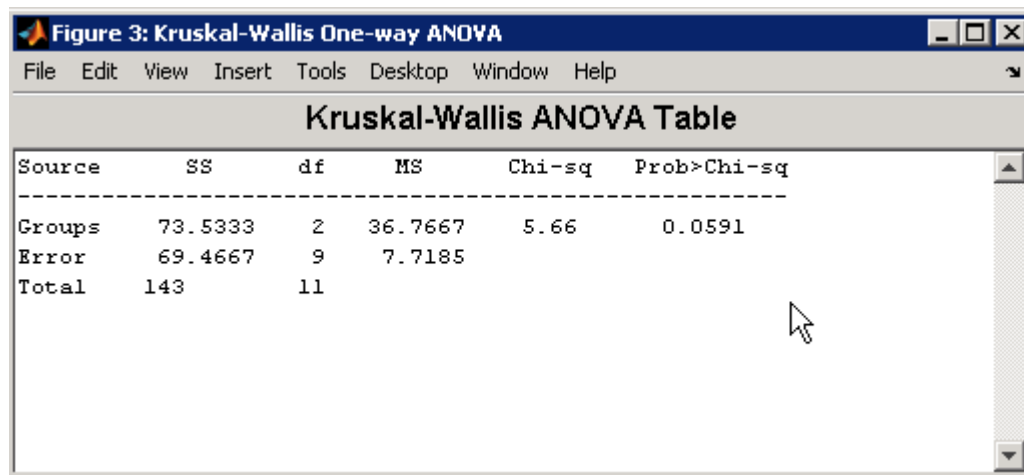
stats =

    gnames: {3x1 cell}
         n: [5 3 4]
    source: 'kruskalwallis'
meanranks: [4.8000 4.6667 10]
      sumt: 0
>>

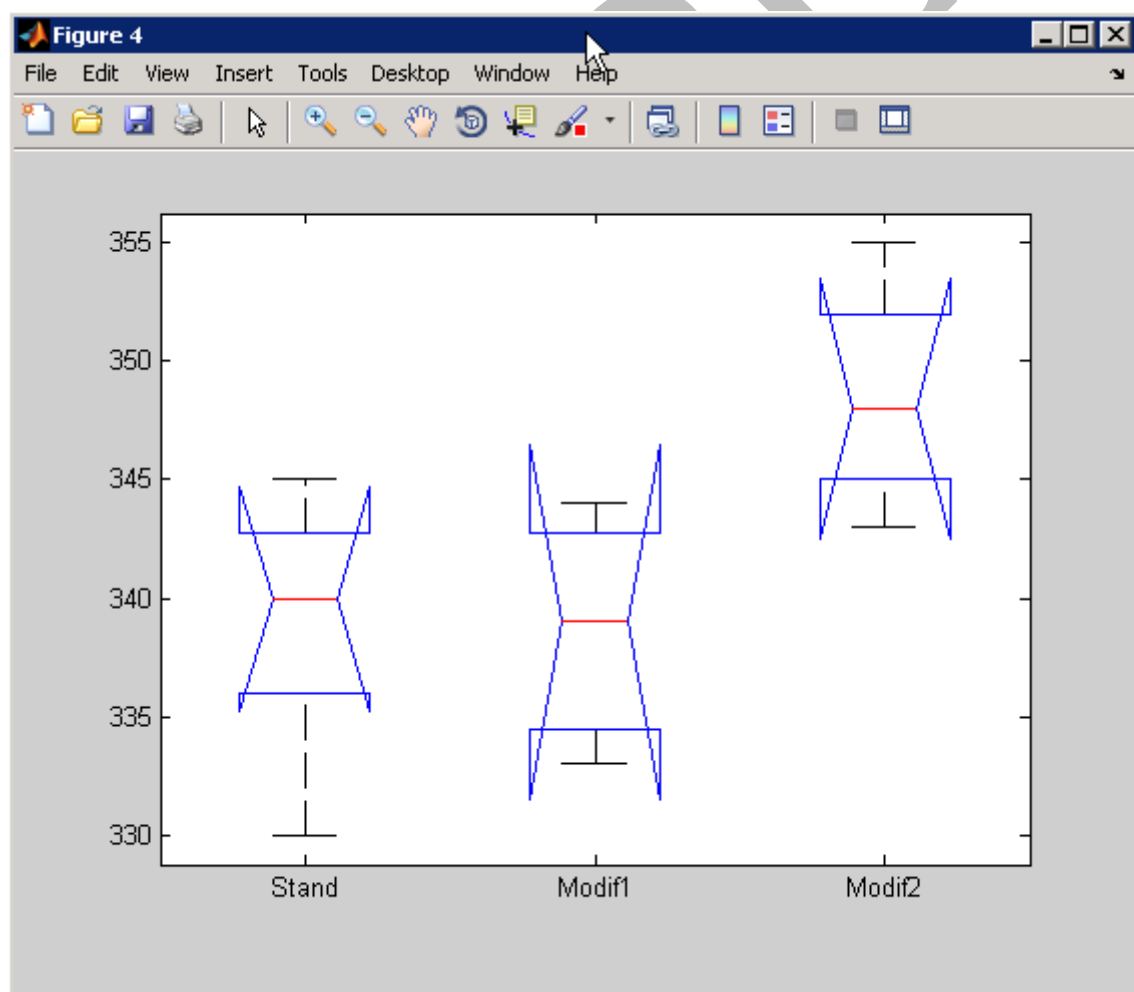
```

As we can see it, we get the same results as the theoretical one and as Minitab and R.

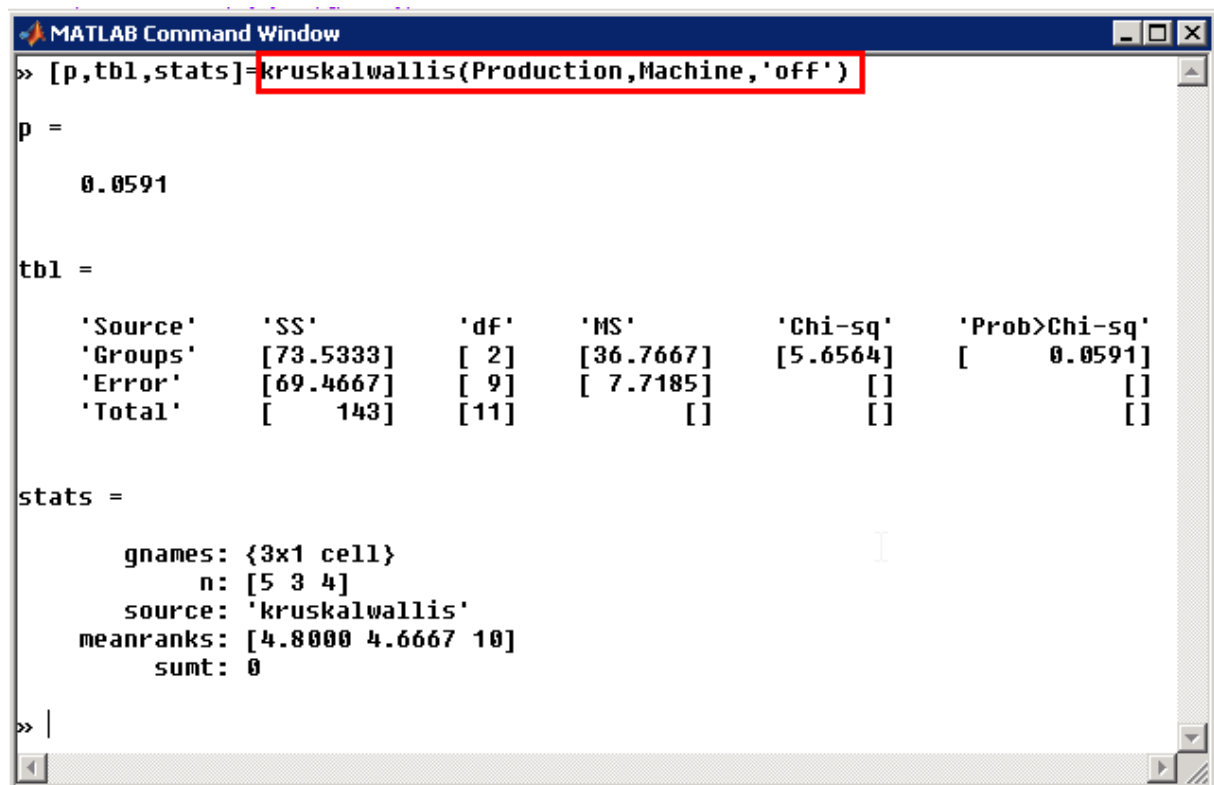
We also have the following figure that will open automatically:



And the following chart:



If we don't want this two previous windows we just have to add a parameter to the function as following:



```
MATLAB Command Window
>> [p,tbl,stats]=kruskalwallis(Production,Machine,'off')

p =

    0.0591

tbl =

    'Source'    'SS'    'df'    'MS'    'Chi-sq'    'Prob>Chi-sq'
    'Groups'    [73.5333]    [ 2]    [36.7667]    [5.6564]    [    0.0591]
    'Error'     [69.4667]    [ 9]    [ 7.7185]    [        ]    [        ]
    'Total'     [   143]    [11]    [        ]    [        ]    [        ]

stats =

    gnames: {3x1 cell}
           n: [5 3 4]
    source: 'kruskalwallis'
meanranks: [4.8000 4.6667 10]
       sumt: 0

>> |
```

Adequation tests

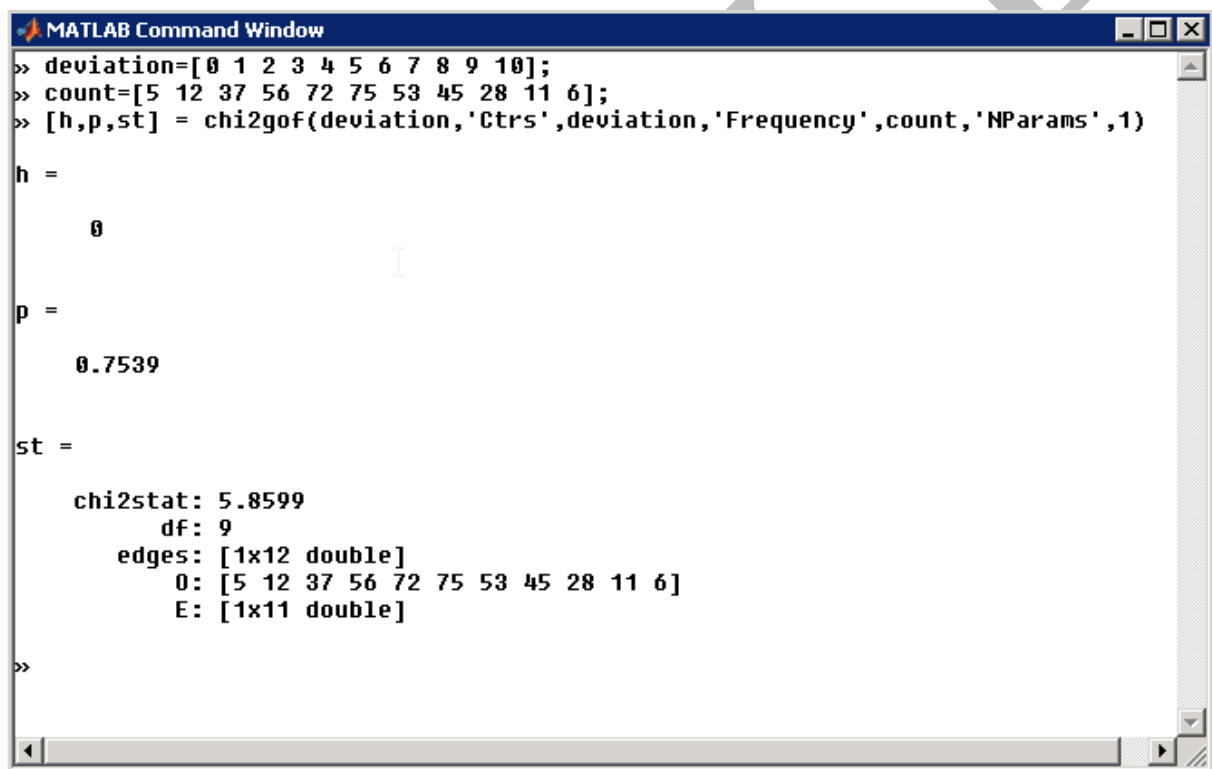
We have seen the important of adequation test in the theoretical course. For its price MATLAB is very poor in possibilities for adequations... for example there is no **native** ranking of best adequation of the forty must well know distributions... Thus I also recommend using the small software of the company:

<http://www.mathwave.com>

for quick fitting among a lot of theoretical distribution (the remark was the same for Minitab, R and SPSS).

Chi-square goodness of fit for Poisson distribution

We begin with the first discrete continuous test we proved in the theoretical course using the function **chi2gof()** and the same valules as in theoretical course Minitab and R:



```

MATLAB Command Window
>> deviation=[0 1 2 3 4 5 6 7 8 9 10];
>> count=[5 12 37 56 72 75 53 45 28 11 6];
>> [h,p,st] = chi2gof(deviation,'Ctrs',deviation,'Frequency',count,'NParams',1)

h =

     0

p =

    0.7539

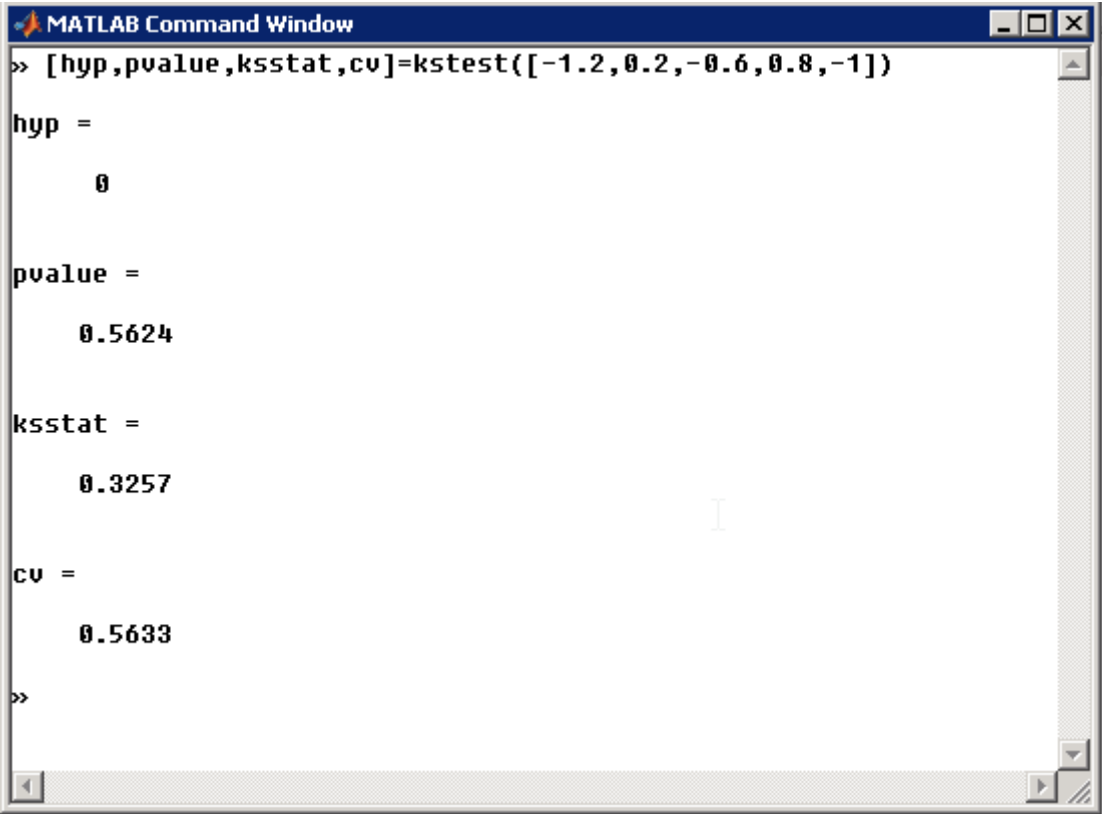
st =

    chi2stat: 5.8599
         df: 9
        edges: [1x12 double]
               0: [5 12 37 56 72 75 53 45 28 11 6]
               E: [1x11 double]
>>
  
```

We can see that the output is much more simple as in Minitab and a little bit different (but the conclusion is the same).

Kolmogorov-Smirnov (one sample) adequation test

Now we take a look to the first adequation continuous test we proved in the theoretical course using the function **kstest()**:



```
MATLAB Command Window
>> [hyp,pvalue,ksstat,cv]=kstest([-1.2,0.2,-0.6,0.8,-1])

hyp =
     0

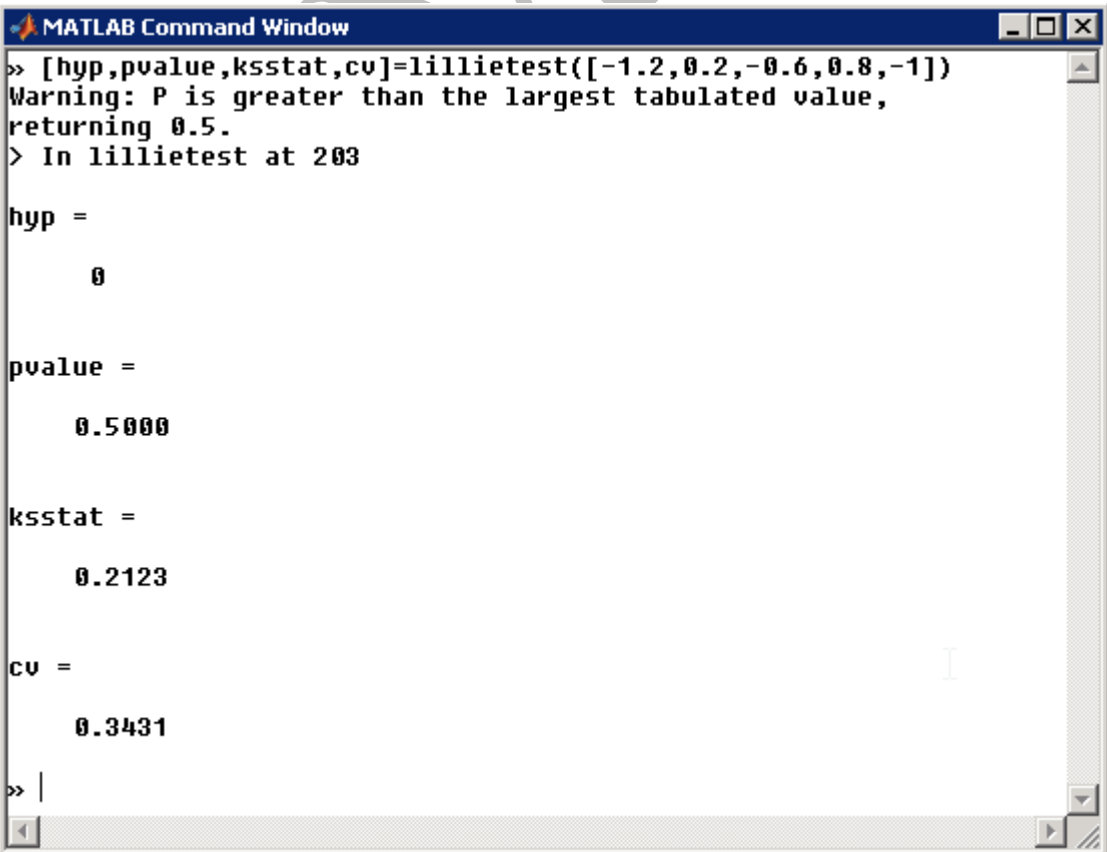
pvalue =
     0.5624

ksstat =
     0.3257

cv =
     0.5633

>>
```

We get the same distance KS distance and p-value (0.3257) as R and the calculations made by hand but not the same as Minitab. To get the same distance as the one of Minitab we must use the Lilliefors variant with the function **lillietest()**:



```
MATLAB Command Window
>> [hyp,pvalue,ksstat,cv]=lillietest([-1.2,0.2,-0.6,0.8,-1])
Warning: P is greater than the largest tabulated value,
returning 0.5.
> In lillietest at 203

hyp =
     0

pvalue =
     0.5000

ksstat =
     0.2123

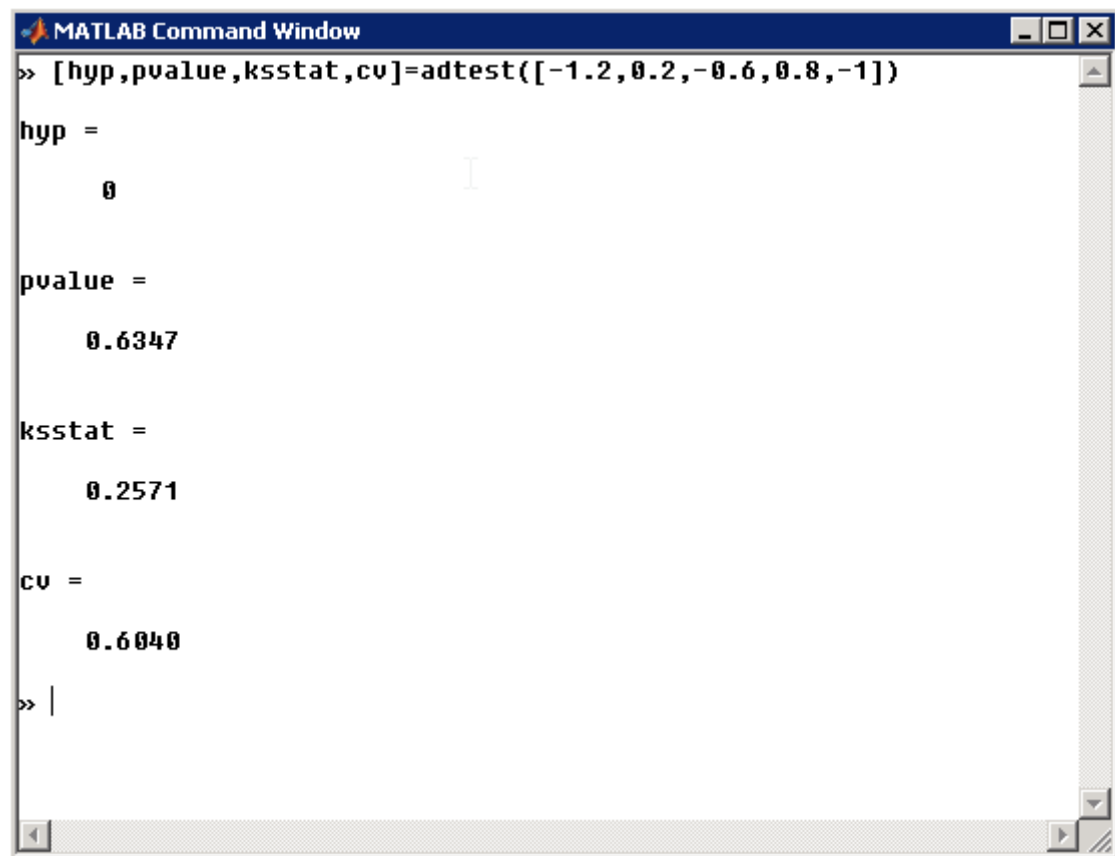
cv =
     0.3431

>> |
```

Se we see that now the distance corresponds to the one given by Minitab.

Anderson-Darling adequation test

Depending on the number of values we have some adequation tests are more powerful than others. Thus it is important to not work only with one adequation test. Another test we proved in the theoretical course is the Anderson-Darling test. Let us see if using the function **adtest()** we found the same results as in the theoretical statistic course:



```
MATLAB Command Window
>> [hyp,pvalue,ksstat,cv]=adtest([-1.2,0.2,-0.6,0.8,-1])

hyp =
     0

pvalue =
     0.6347

ksstat =
     0.2571

cv =
     0.6040

>> |
```

We get the same distance (0.2571) as Minitab and the calculations made by hand but not the same as R (0.636). The p-value is not the same as R (0.6058) and nore Minitab (0.535) but we know why...

Regression Techniques

As we know, regression is a subject also as big as the ANOVA. There are books of thousand of pages presenting the most common regression techniques models and even with thousand pages there are not exhaustive.

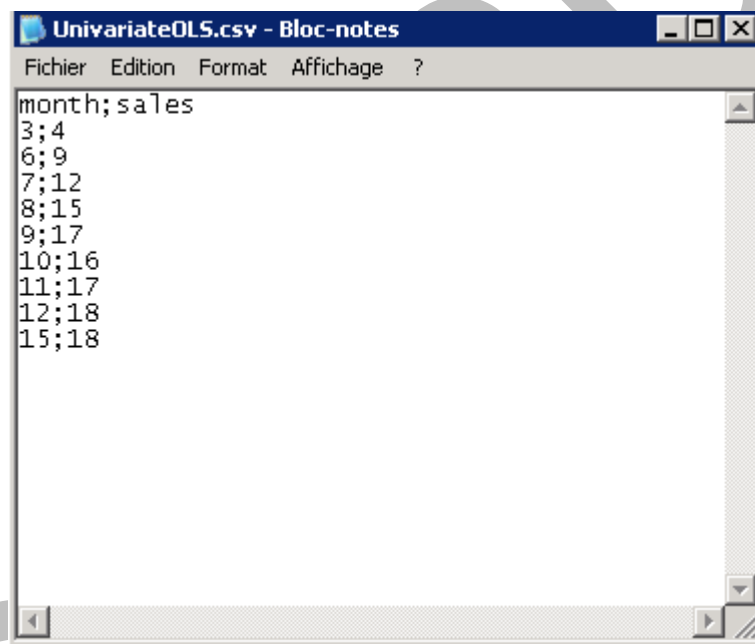
We will first begin by the most simple models in the order we normally study the at school and university.

Ordinary Least Squares (parametric) univariate regression

The parametric Ordinary Least Square regression is a undergraduate technic then it is normal that we begin with this one.

1st method (coefficient and constant can not be extracted)

There exist multiple ways with the statistical toolbox to make an univariate parametric OLS regression. For this purpose consider the same data as in theoretical course:



By using the function `LinearModel.fit()` we get the classic univariate coefficients analysis table:

```

Command Window
>> mydata=dataset('File','C:\tmp\UnivariateOLS.csv','Delimiter',';');
%in 2013a use LinearModel.fit
%in 2013b use fitlm
mdl = LinearModel.fit(mydata,'linear');
>> mdl

mdl =

Linear regression model:
    sales ~ 1 + month

Estimated Coefficients:

```

	Estimate	SE	tStat	pValue
(Intercept)	3.02	2.151	1.404	0.20309
month	1.22	0.22412	5.4436	0.00096257

```

Number of observations: 9, Error degrees of freedom: 7
Root Mean Squared Error: 2.24
R-squared: 0.809, Adjusted R-Squared 0.782
F-statistic vs. constant model: 29.6, p-value = 0.000963
fx >> |

```

All values corresponds with the one obtained in the theoretical course. If we want to specify explicitly the equation of the regression we can do this easily like in R to be able later to build a more complex model:

```

Command Window

>> mdl = LinearModel.fit(mydata, 'sales~1+month')

mdl =

Linear regression model:
    sales ~ 1 + month

Estimated Coefficients:
               Estimate      SE      tStat      pValue
    (Intercept)      3.02      2.151      1.404      0.20309
    month           1.22      0.22412     5.4436     0.00096257

Number of observations: 9, Error degrees of freedom: 7
Root Mean Squared Error: 2.24
R-squared: 0.809, Adjusted R-Squared 0.782
F-statistic vs. constant model: 29.6, p-value = 0.000963
fx >>

```

Thanks to this notation we can for example easily for the constant to be zero:

```

Command Window

>> LinearModel.fit(mydata, 'sales~-1+month') %force origin=0

ans =

Linear regression model:
    sales ~ month

Estimated Coefficients:
               Estimate      SE      tStat      pValue
    month       1.5151      0.082429     18.38     7.9029e-08

Number of observations: 9, Error degrees of freedom: 8
Root Mean Squared Error: 2.37
fx >> |

```

We can also get the famous ANOVA table of the regression by using the function **anova()** with in bracket the variable containing our linear model:

```
Command Window
>> anova mdl

ans =

      SumSq    DF    MeanSq    F    pValue
month    148.84    1    148.84    29.633    0.00096257
Error     35.16    7     5.0229
```

Or we can also predict values using the model by using the function **predict()**:

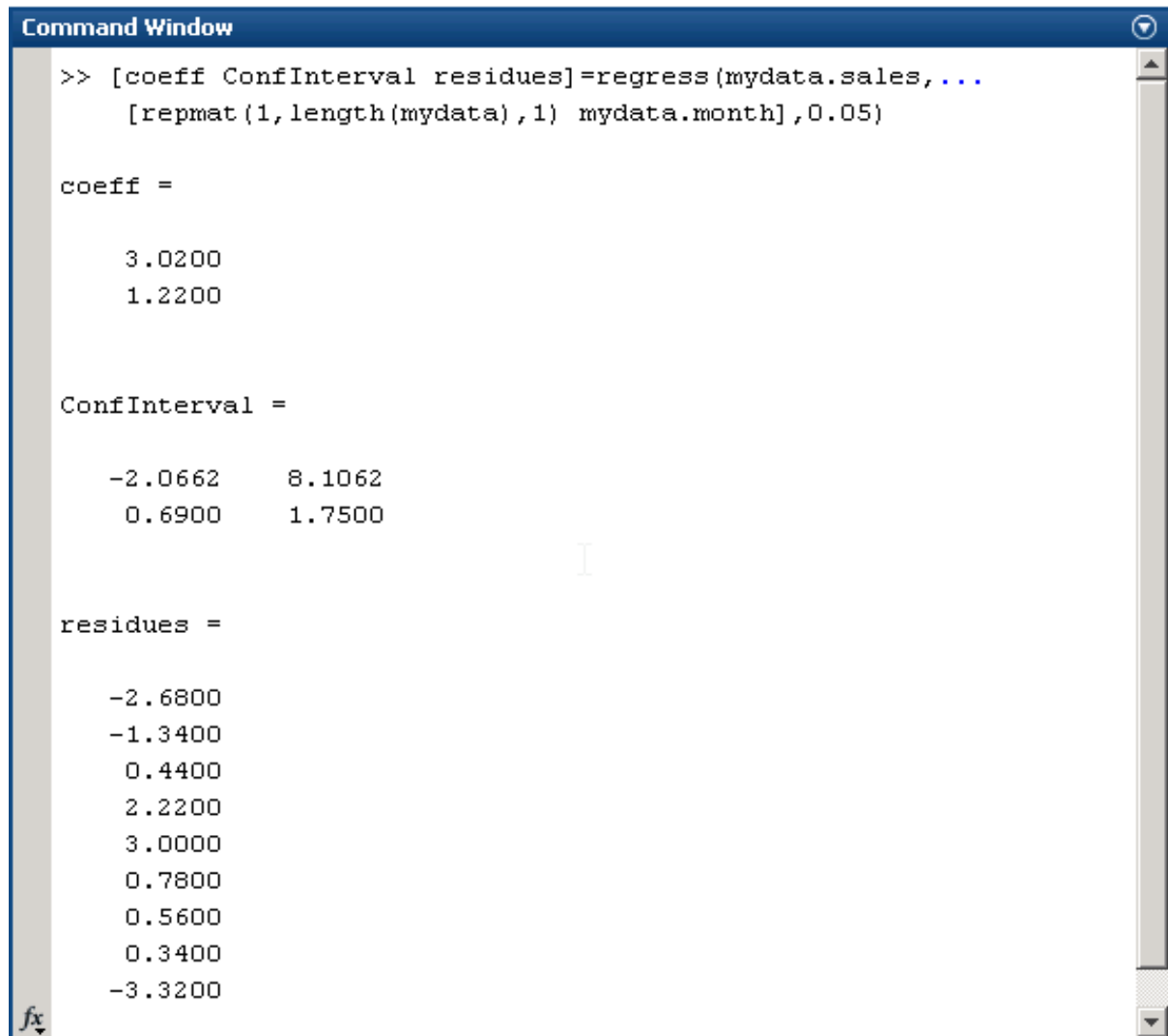
```
Command Window
>> predict(mdl,3)

ans =

    6.6800
```

2nd method (coefficient and constant and others can be extracted)

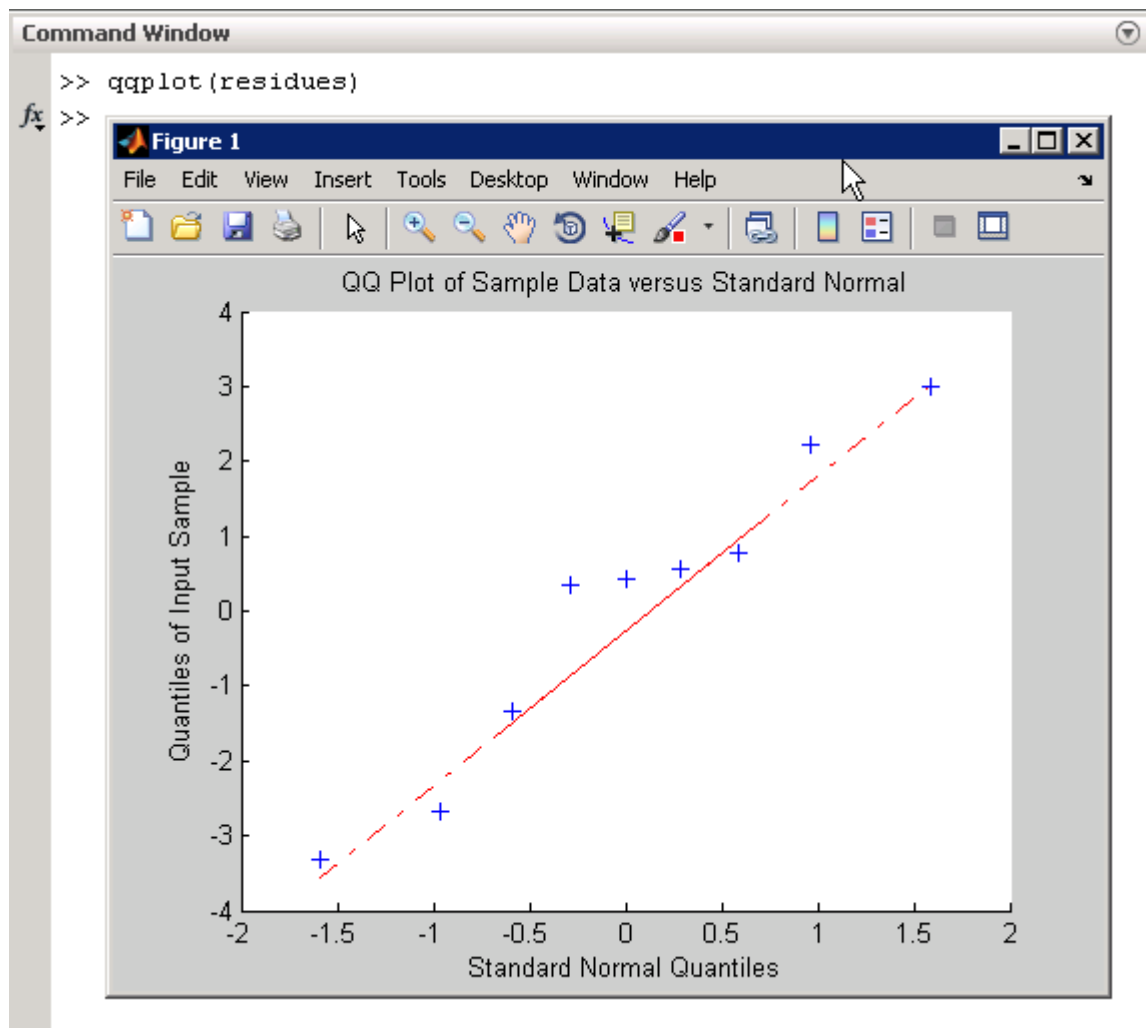
The problem as far as I know with the first method is that you can't extract only the coefficient of the model and this is a major issue in financial application. As workaround I recommend the usage of the function **regress()** with a syntax that is not intuitive but that in fact simply use the explicit matrix of coefficient that we have seen in the theoretical cours:

A screenshot of the MATLAB Command Window. The title bar says "Command Window". The command entered is: `>> [coeff ConfInterval residues]=regress(mydata.sales,...
[repmat(1,length(mydata),1) mydata.month],0.05)`. The output shows three variables: `coeff =` with values `3.0200` and `1.2200`; `ConfInterval =` with a 2x2 matrix of values: `-2.0662 8.1062` and `0.6900 1.7500`; and `residues =` with a column vector of 10 values: `-2.6800`, `-1.3400`, `0.4400`, `2.2200`, `3.0000`, `0.7800`, `0.5600`, `0.3400`, and `-3.3200`.

```
>> [coeff ConfInterval residues]=regress(mydata.sales,...  
    [repmat(1,length(mydata),1) mydata.month],0.05)  
  
coeff =  
  
    3.0200  
    1.2200  
  
ConfInterval =  
  
   -2.0662    8.1062  
    0.6900    1.7500  
  
residues =  
  
   -2.6800  
   -1.3400  
    0.4400  
    2.2200  
    3.0000  
    0.7800  
    0.5600  
    0.3400  
   -3.3200
```

So in this way it is very easy as you can see it to extract information of the linear model!

Afterwards if we want to analyse the residues normality in a qualitative way we can do this for example with a simple **qqplot()**:



or with an adequation test (see above for this last subject)

```
Command Window

>> [rejectH0 pValue]=kstest(residues)

rejectH0 =

    0

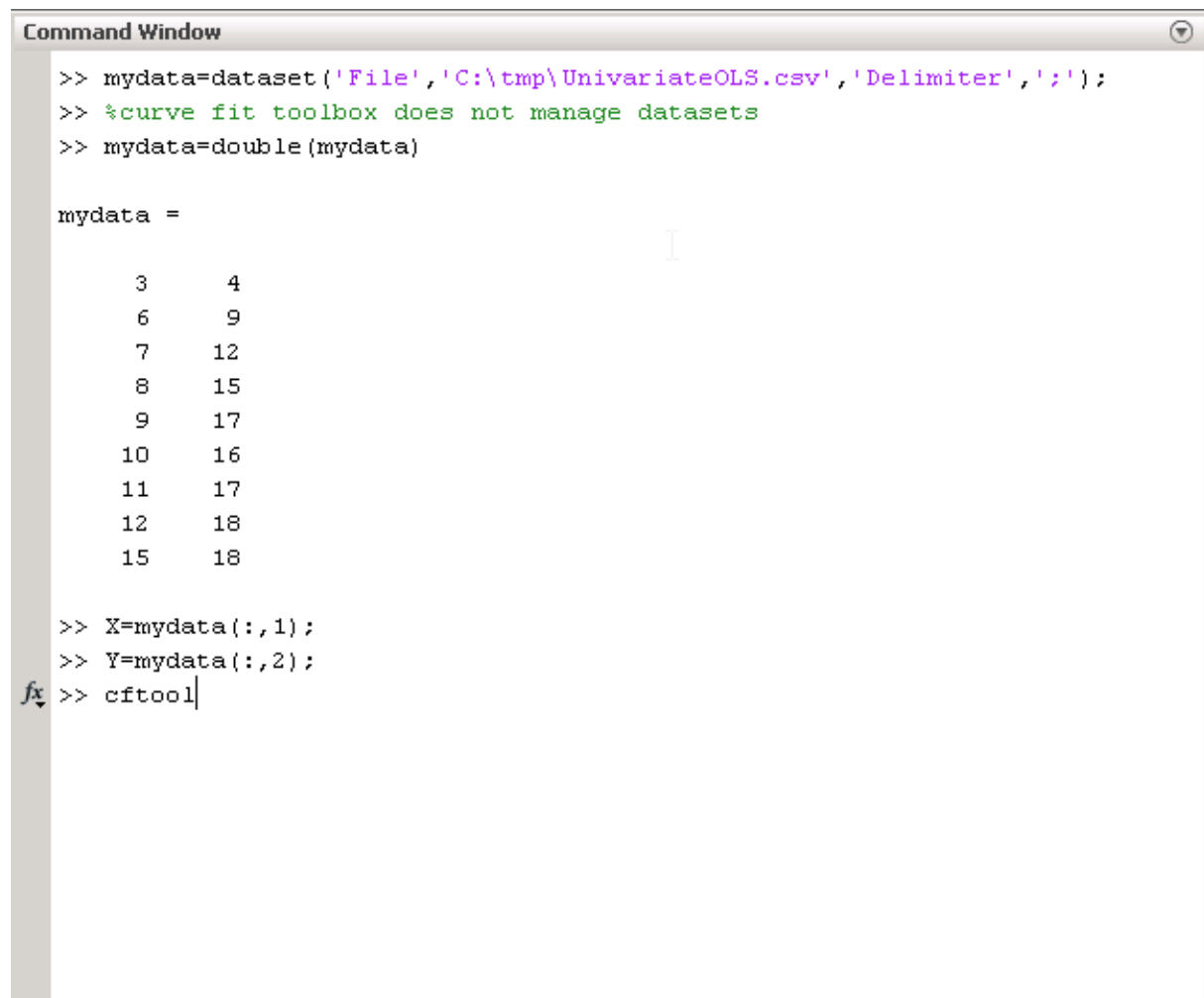
pValue =

    0.32537895067878

fx >> |
```


3rd method (using Curve Fitting Toolbox)

For those that don't like to use too much MATLAB commands you can use the curve fitting toolbox wizard by typing the command **cftool**:



```
Command Window

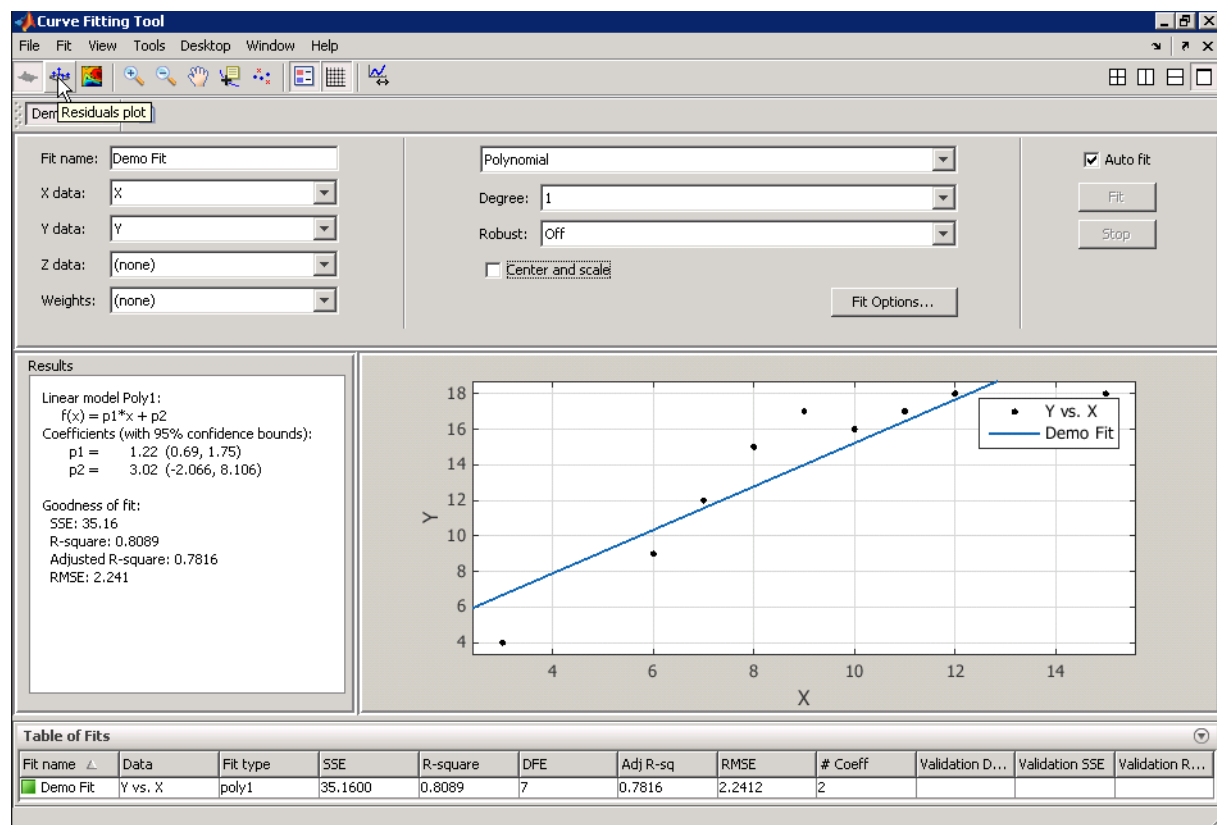
>> mydata=dataset('File','C:\tmp\UnivariateOLS.csv','Delimiter',';');
>> %curve fit toolbox does not manage datasets
>> mydata=double(mydata)

mydata =

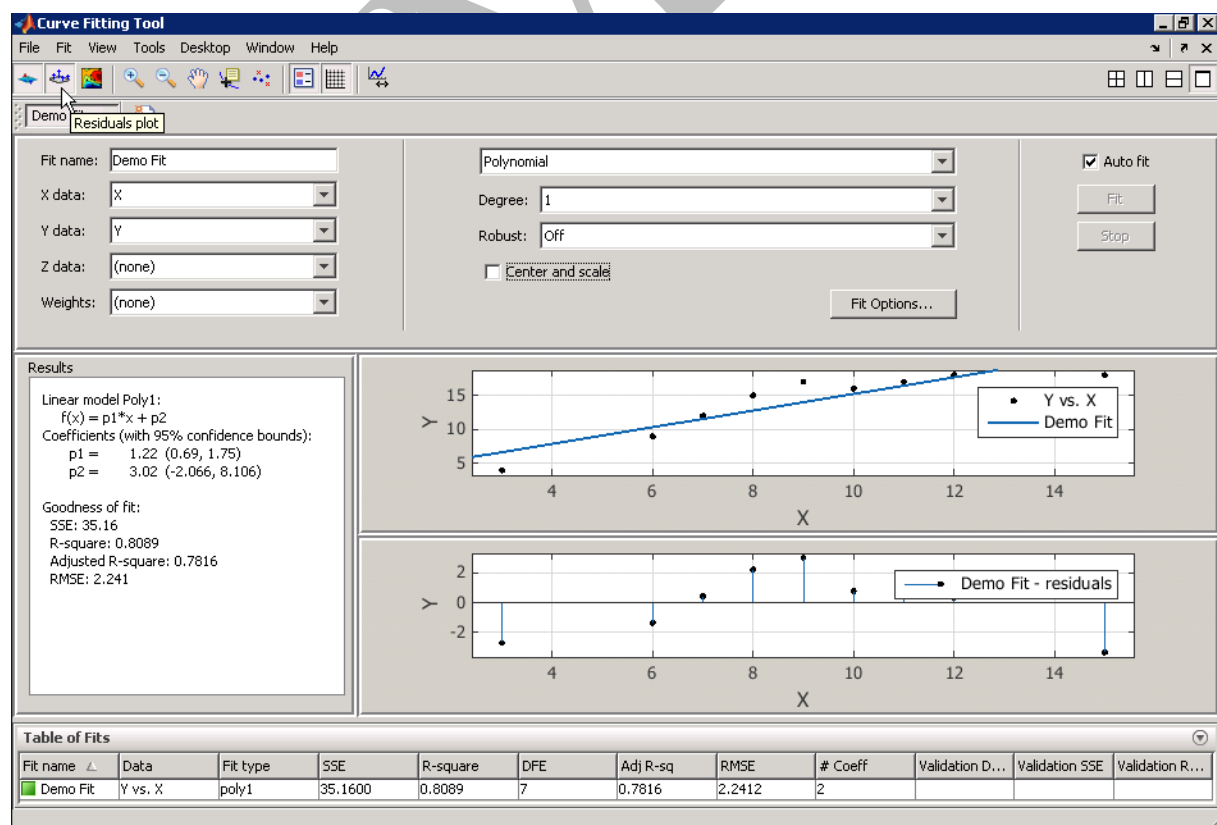
     3     4
     6     9
     7    12
     8    15
     9    17
    10    16
    11    17
    12    18
    15    18

>> X=mydata(:,1);
>> Y=mydata(:,2);
fx >> cftool|
```

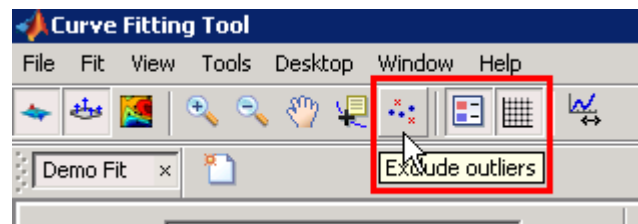
Then you will get:



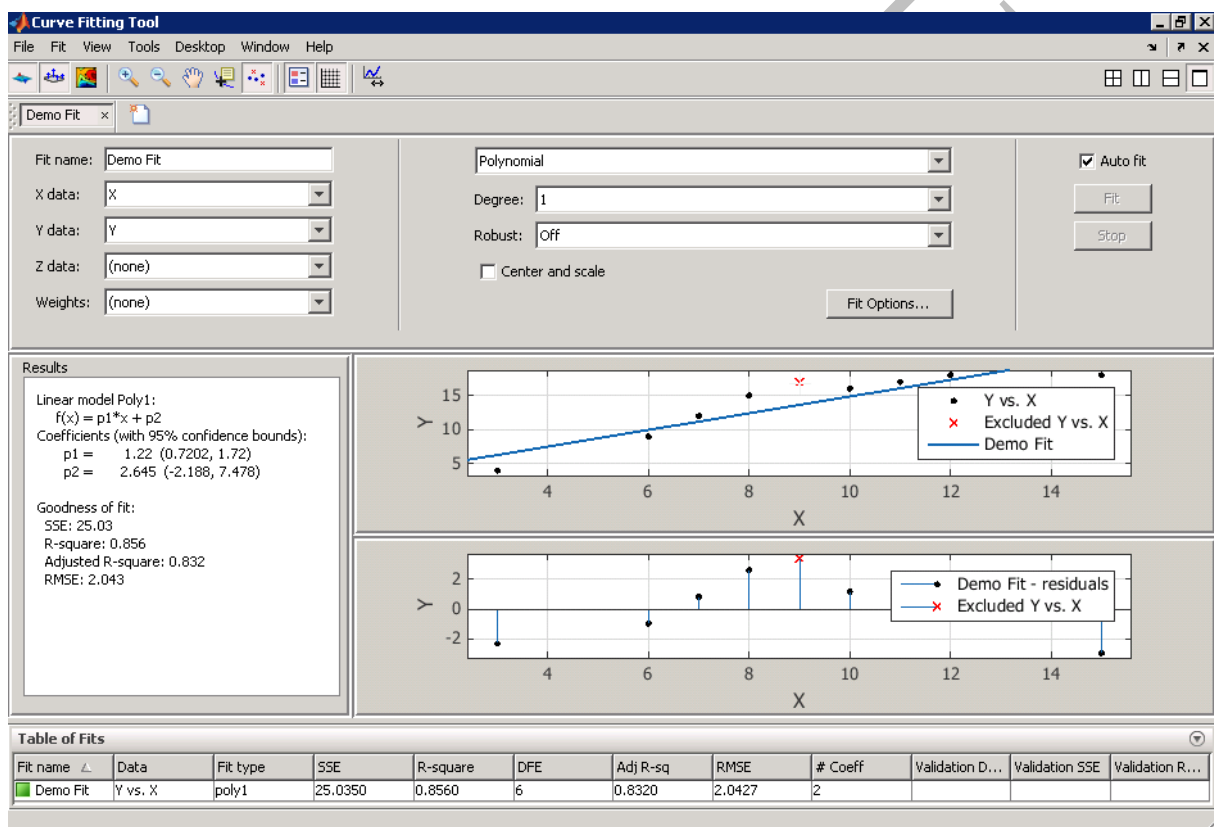
If you click on the button **Residuals plots** you get:



Now a more interesting stuff to focus on is the management of outliers. If you click on the button **Exclude outliers**:

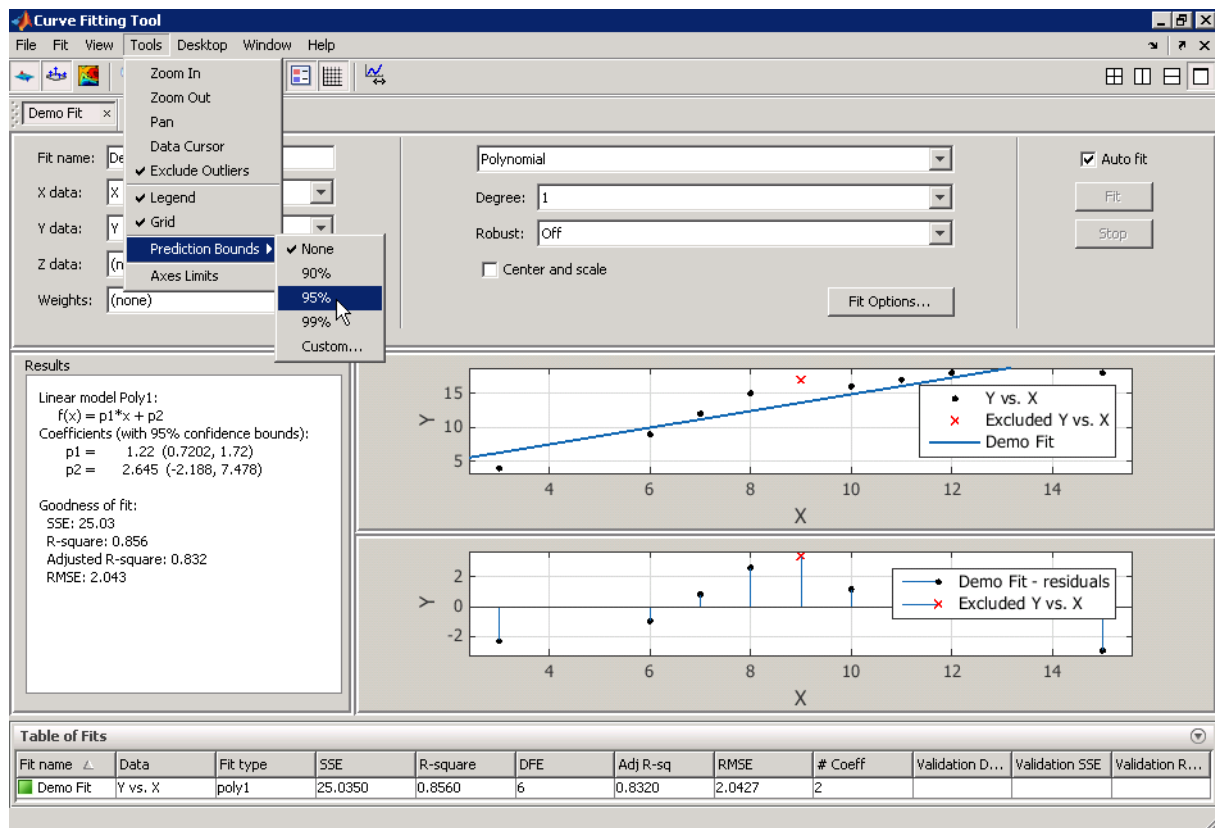


Once activated you have to click on outliers points to exclude them (not very scientific way...):

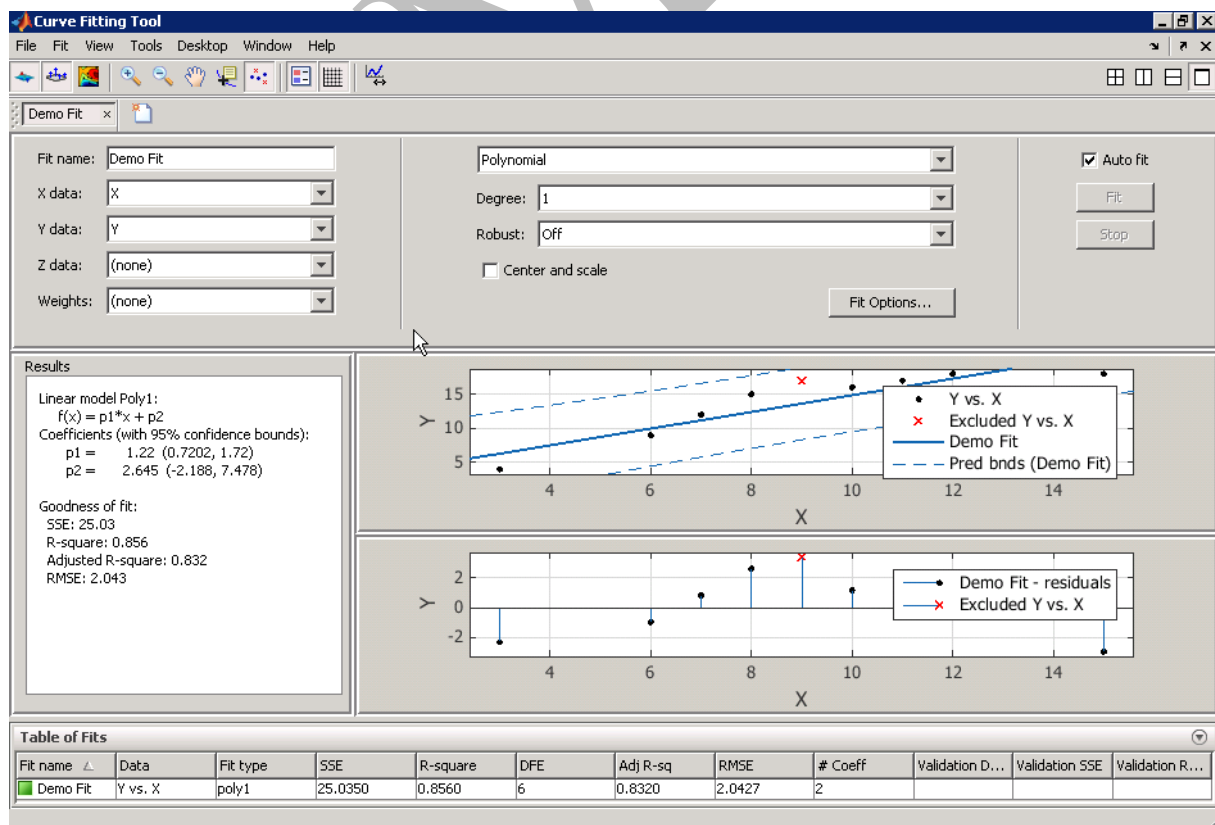


Thus the outlier elimination method here is a joke when we know the number of statistical techniques dedicated to this purpose.

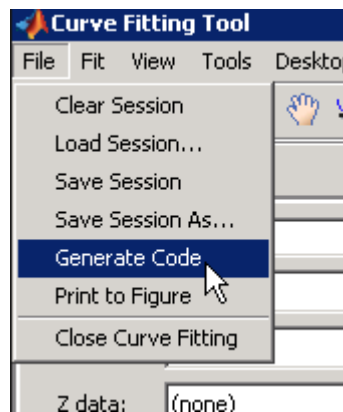
With this tool we also have the possibility to show the prediction bounds trough the menu **Tools/Prediction Bounds/95%**:



The result will be:



If you want to get the MATLAB script for all the previous manipulations you just have to go in the menu **File/Generate Code**:



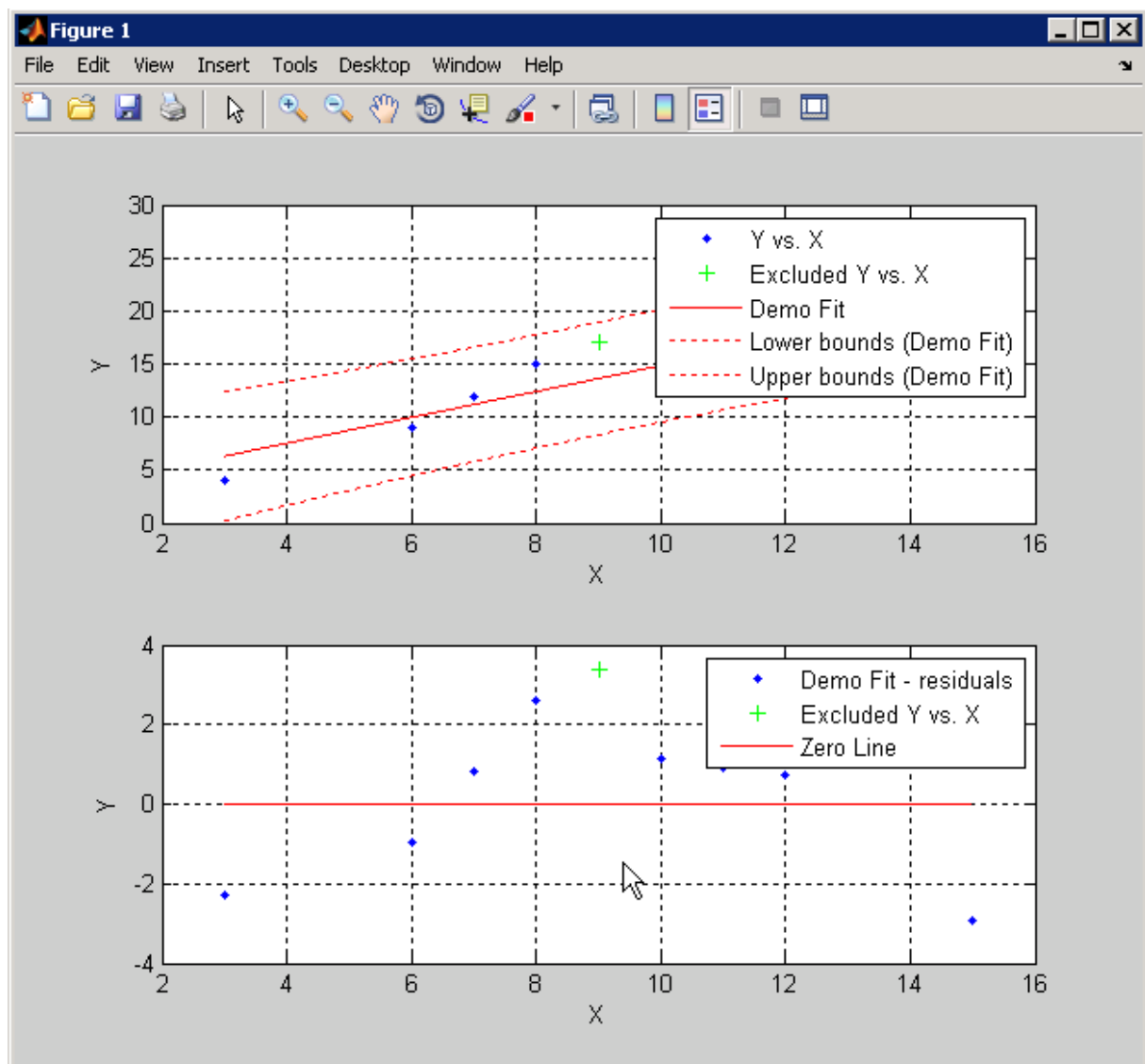
The result will be:

```

EDITOR    PUBLISH    VIEW
1      [xData, yData] = prepareCurveData( X, Y );
2
3      % Set up fittype and options.
4      ft = fittype( 'poly1' );
5      opts = fitoptions( ft );
6      opts.Lower = [-Inf -Inf];
7      opts.Upper = [Inf Inf];
8      ex = excludedata( xData, yData, 'Indices', 5 );
9      opts.Exclude = ex;
10
11     % Fit model to data.
12     [fitresult, gof] = fit( xData, yData, ft, opts );
13
14     % Plot fit with data.
15     subplot( 2, 1, 1 );
16     h = plot( fitresult, xData, yData, ex, 'predobs' );
17     legend( h, 'Y vs. X', 'Excluded Y vs. X', 'Demo Fit',...
18           'Lower bounds (Demo Fit)', 'Upper bounds (Demo Fit)',...
19           'Location', 'NorthEast' );
20     % Label axes
21     xlabel( 'X' );
22     ylabel( 'Y' );
23     grid on
24
25     % Plot residuals.
26     subplot( 2, 1, 2 );
27     h = plot( fitresult, xData, yData, ex, 'residuals' );
28     legend( h, 'Demo Fit - residuals', 'Excluded Y vs. X', 'Zero Line', 'Location', 'NorthEast' );
29     % Label axes
30     xlabel( 'X' );ylabel( 'Y' );grid on

```

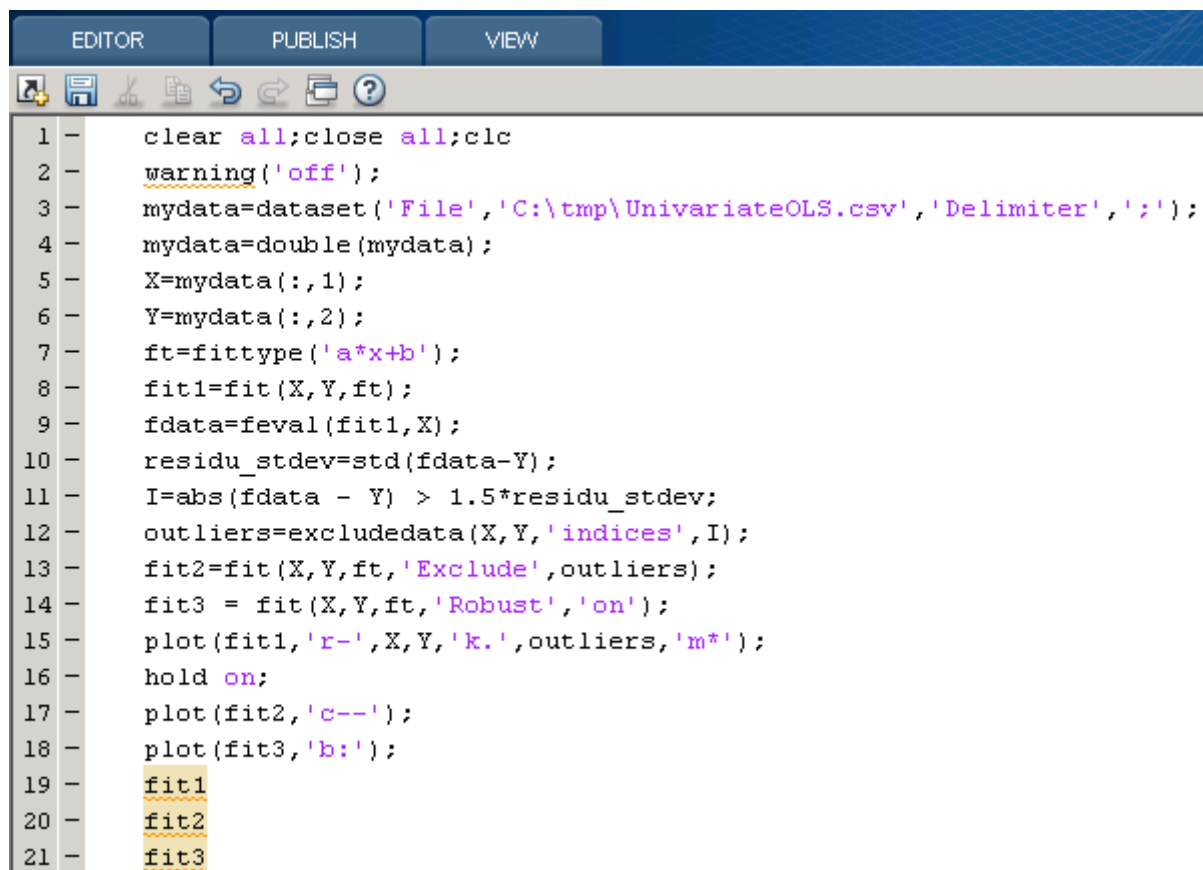
And if you run it you will get:



Manage outliers

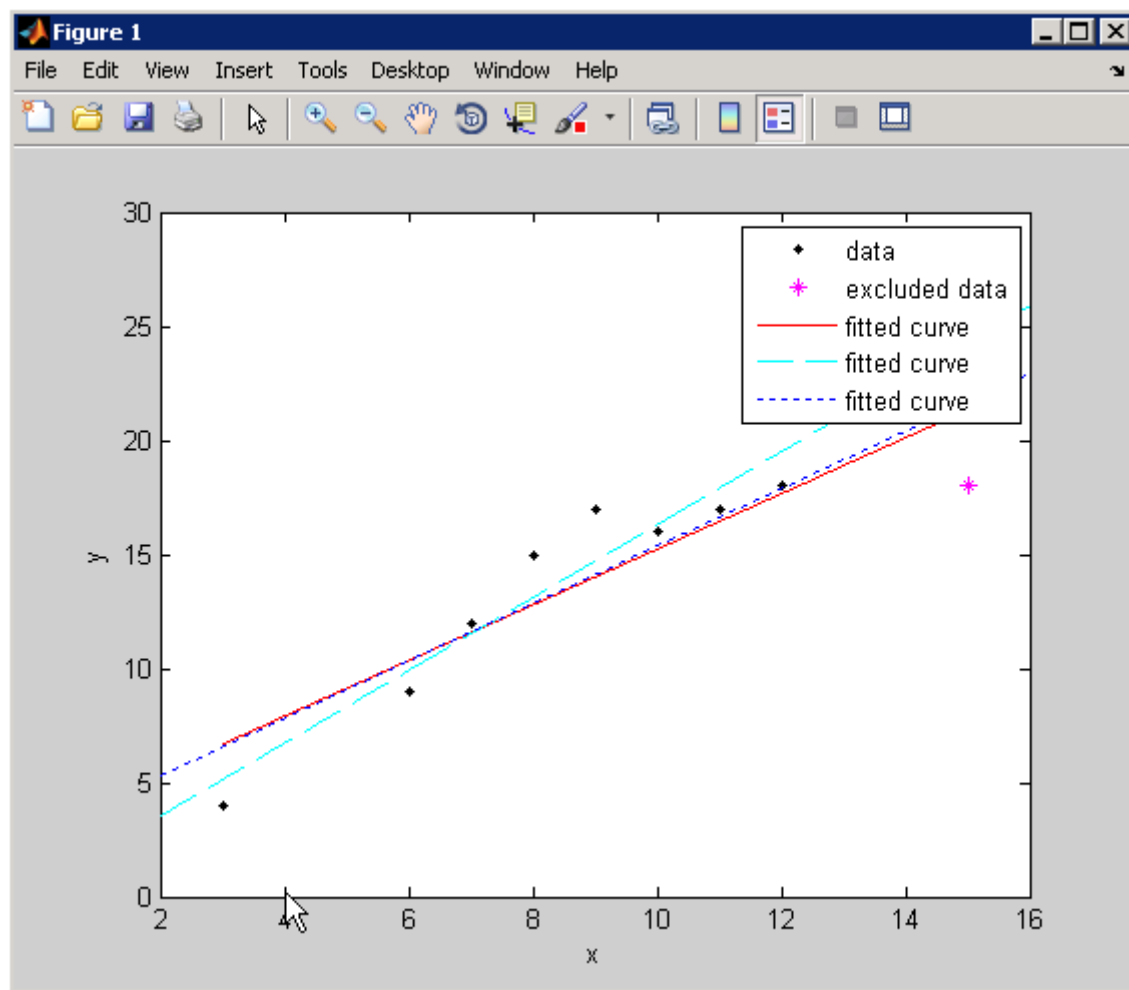
Manager outlier is a very important subject. Some trainings are given only on this subject during almost a whole day.

Here we show only a very basic example using the core function **feval()** with the strong hypothesis of gaussian distributed residues to eliminate the outliers using the curve fitting toolbox functions **excludedata()** and **fit()** with various parameters:

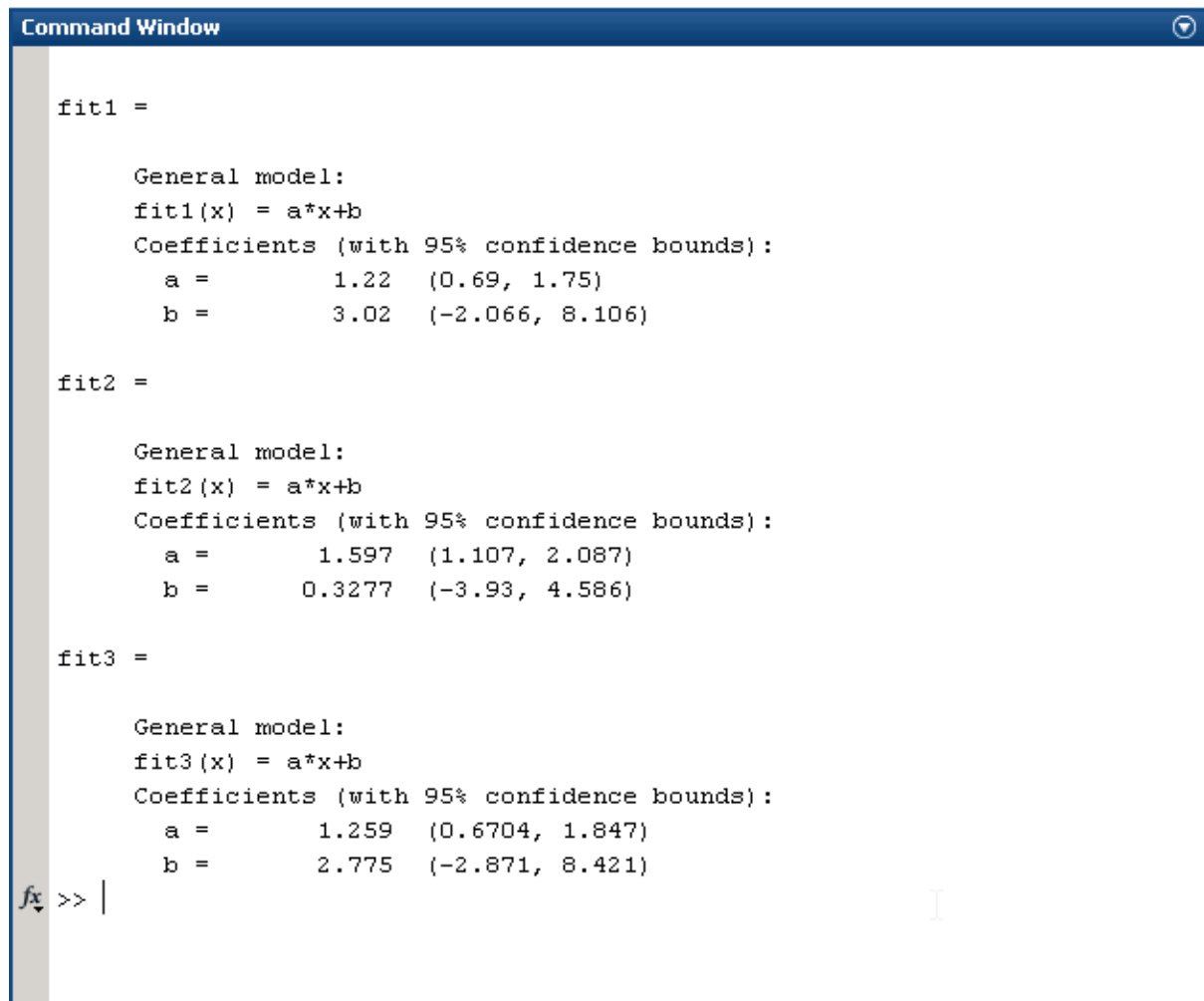
A screenshot of the MATLAB Editor window. The window has a dark blue header bar with 'EDITOR', 'PUBLISH', and 'VIEW' tabs. Below the header is a toolbar with icons for file operations and editing. The main area shows a script with 21 lines of MATLAB code. The code performs the following steps: 1. Clear all variables and close all figures. 2. Turn off warnings. 3. Load data from 'C:\tmp\UnivariateOLS.csv' using 'dataset' with a semicolon as a delimiter. 4. Convert the data to double. 5. Extract the first column as X and the second as Y. 6. Define a linear fit type 'a*x+b'. 7. Perform a standard linear fit. 8. Evaluate the fit to get fdata. 9. Calculate the standard deviation of the residuals. 10. Identify outliers where the absolute residual is greater than 1.5 times the standard deviation. 11. Exclude the outliers from the data. 12. Perform a fit excluding the outliers. 13. Perform a robust fit. 14. Plot the original data (red 'x'), the standard fit (cyan dashed line), the excluded data (black 'k'), and the outliers (magenta asterisks). 15. Hold the plot. 16. Plot the fit excluding outliers (cyan dashed line). 17. Plot the robust fit (blue line). 18. Plot the standard fit (cyan dashed line). 19. Plot the standard fit (cyan dashed line). 20. Plot the standard fit (cyan dashed line). 21. Plot the standard fit (cyan dashed line).

```
1 - clear all;close all;clc
2 - warning('off');
3 - mydata=dataset('File','C:\tmp\UnivariateOLS.csv','Delimiter',';');
4 - mydata=double(mydata);
5 - X=mydata(:,1);
6 - Y=mydata(:,2);
7 - ft=fitttype('a*x+b');
8 - fit1=fit(X,Y,ft);
9 - fdata=feval(fit1,X);
10 - residu_stdev=std(fdata-Y);
11 - I=abs(fdata - Y) > 1.5*residu_stdev;
12 - outliers=excludedata(X,Y,'indices',I);
13 - fit2=fit(X,Y,ft,'Exclude',outliers);
14 - fit3 = fit(X,Y,ft,'Robust','on');
15 - plot(fit1,'r-',X,Y,'k.',outliers,'m*');
16 - hold on;
17 - plot(fit2,'c--');
18 - plot(fit3,'b:');
19 - fit1
20 - fit2
21 - fit3
```

So at the end we have a visual of three different regressions, in red the parametric regression including outliers, in dashed cyan the parametric regression without outliers and in dashed blue the non-parametric regression:



And in the Command Window we have:



```
Command Window

fit1 =

General model:
fit1(x) = a*x+b
Coefficients (with 95% confidence bounds):
    a =      1.22   (0.69, 1.75)
    b =      3.02  (-2.066, 8.106)

fit2 =

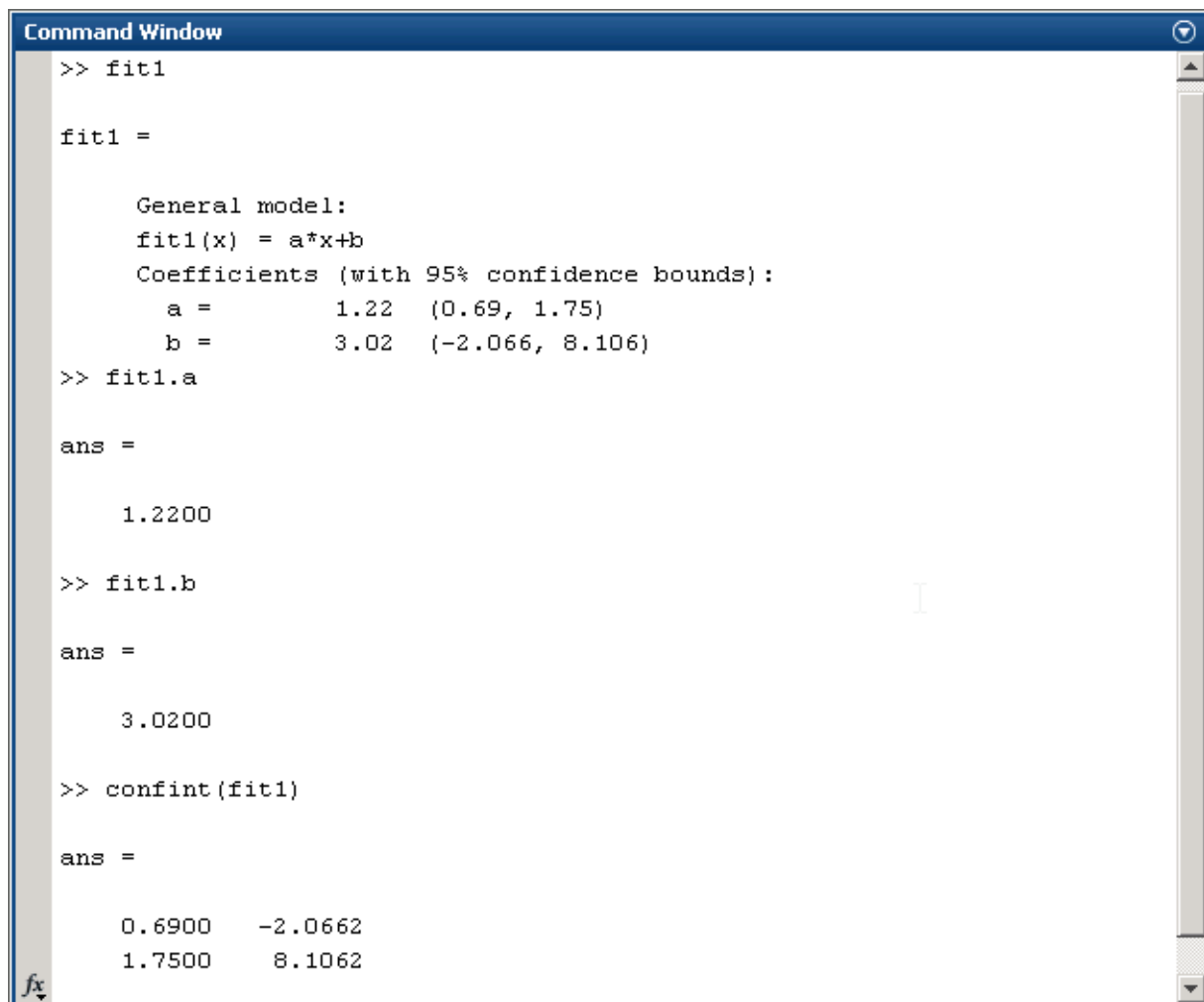
General model:
fit2(x) = a*x+b
Coefficients (with 95% confidence bounds):
    a =      1.597  (1.107, 2.087)
    b =      0.3277 (-3.93, 4.586)

fit3 =

General model:
fit3(x) = a*x+b
Coefficients (with 95% confidence bounds):
    a =      1.259  (0.6704, 1.847)
    b =      2.775  (-2.871, 8.421)

fx >> |
```

And if we need to extract some of the coefficients:



```
>> fit1

fit1 =

    General model:
    fit1(x) = a*x+b
    Coefficients (with 95% confidence bounds):
        a =      1.22   (0.69, 1.75)
        b =      3.02   (-2.066, 8.106)

>> fit1.a

ans =

    1.2200

>> fit1.b

ans =

    3.0200

>> confint(fit1)

ans =

    0.6900   -2.0662
    1.7500    8.1062
```

Residuals and Leverage (residual diagnosis)

We have proved in the theoretical course how to calculate the leverage of the residuals of a linear regression. Let us see if we obtain the same results as the one calculated by hand with our simple example.

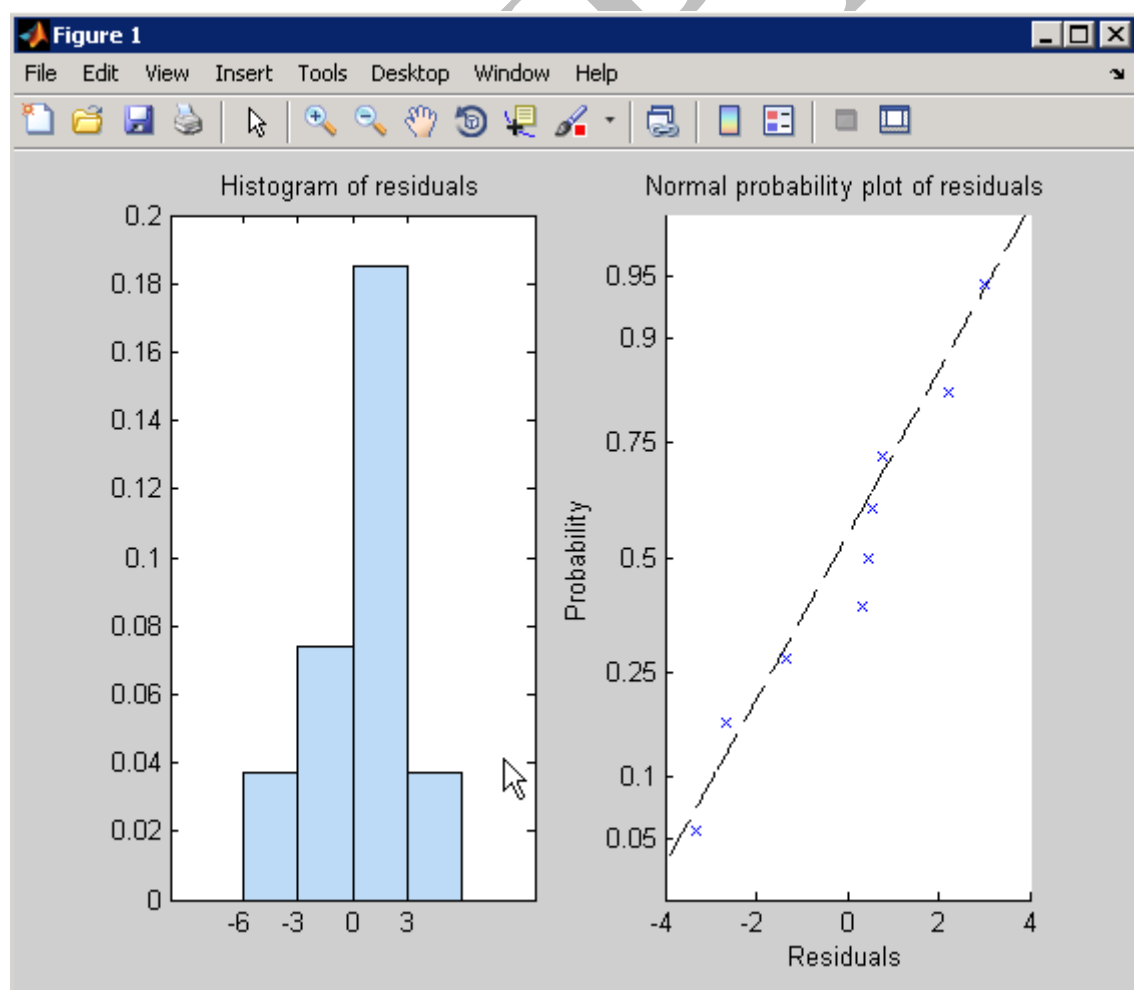
First we plot the residuals distribution and Henry plot using the function **plotResiduals()**:

```

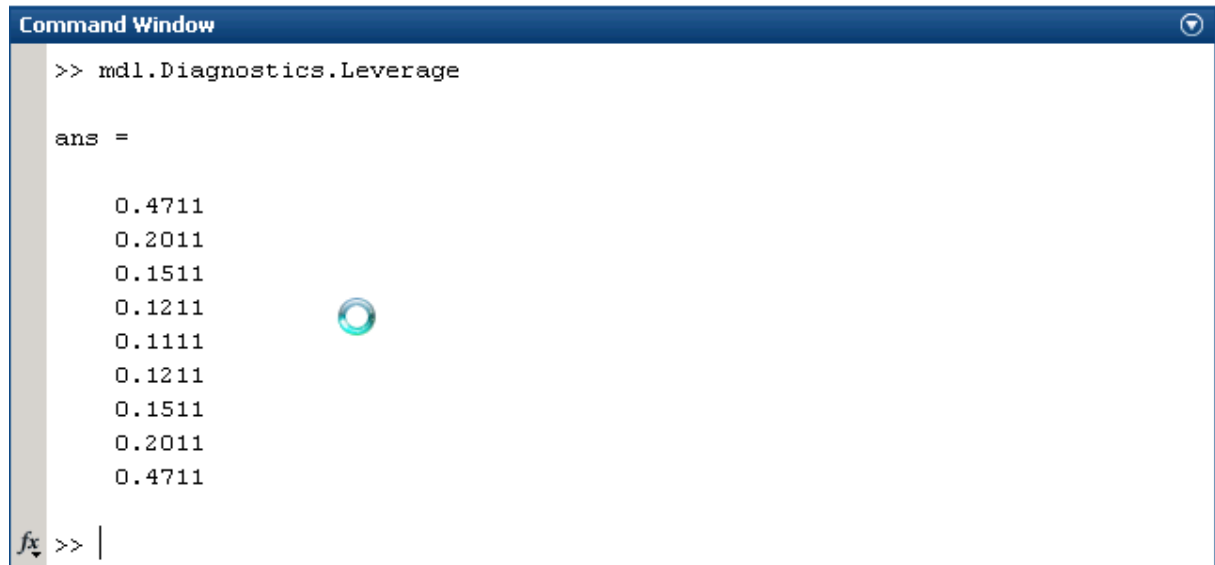
EDITOR    PUBLISH    VIEW
1  mydata=dataset('File','C:\tmp\UnivariateOLS.csv','Delimiter',';');
2  mdl=LinearModel.fit(mydata,'linear');
3  subplot(1,2,1)
4  plotResiduals(mdl)
5  subplot(1,2,2)
6  plotResiduals(mdl,'probability')
7  mdl.Diagnostics.Leverage|

```

So for the plots we get:



And for the diagnostics leverage we get using the function property Leverage of the method Diagnostics trough**Diagnostics.Leverage**:



```

Command Window

>> mdl.Diagnostics.Leverage

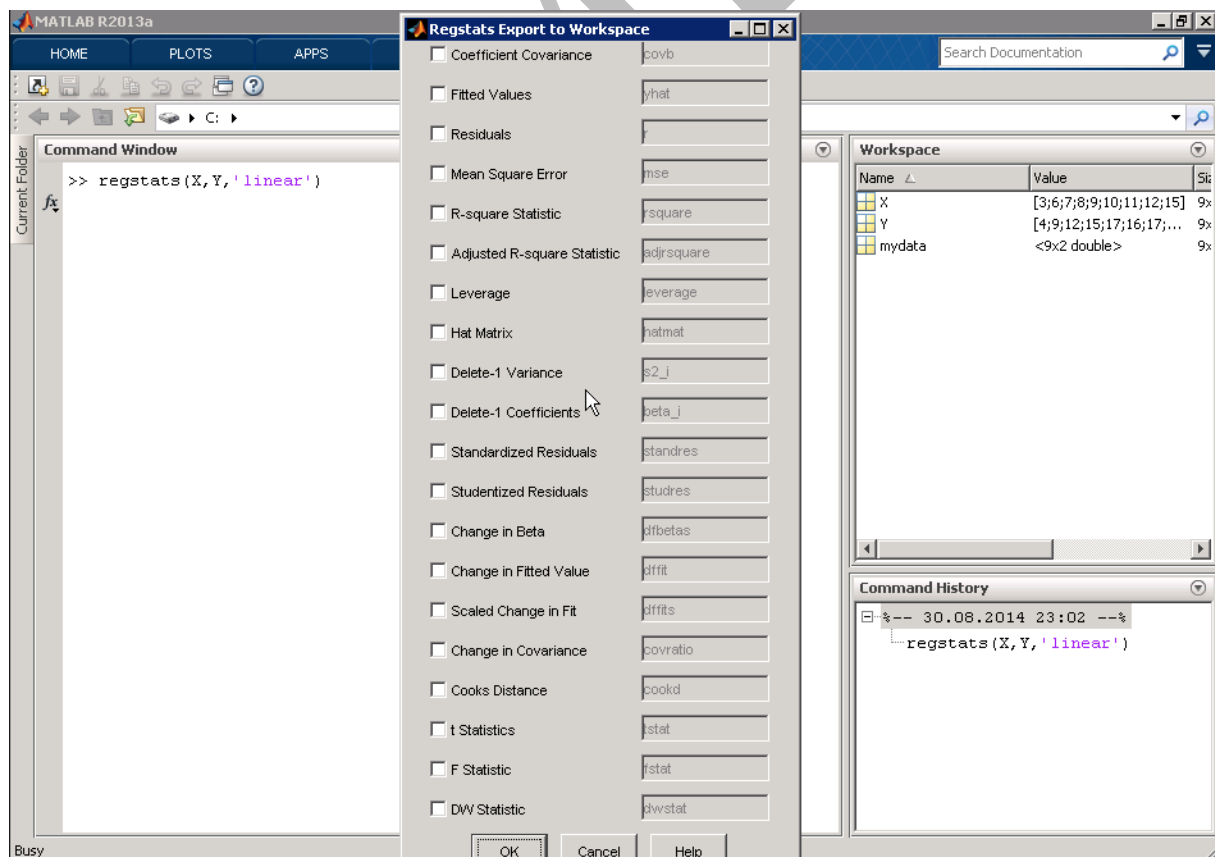
ans =

    0.4711
    0.2011
    0.1511
    0.1211
    0.1111
    0.1211
    0.1511
    0.2011
    0.4711
  
```

This is corresponding to what we get in the theoretical course.

Regression diagnostics wizard

We can get a lot of others diagnostics using the **regstats()** function wizard:



If we select everything we get in the MATLAB variable workspace the following:

Workspace				
Name ▲	Value	Size	Bytes	Class
X	[3;6;7;8;9;10;11;12;15]	9x1	72	double
Y	[4;9;12;15;17;16;17;18;18]	9x1	72	double
adjrsquare	0.7816	1x1	8	double
beta_j	<2x9 double>	2x9	144	double
cookd	[0.3994;0.0079;0.0147;0.0756;0.1644;0.0034;4.9081e-06;0.0068;0.6323]	9x1	72	double
covb	[3.2112;-0.2077;-0.2077;0.0148]	2x2	32	double
covratio	[3.4803;1.7827;1.4848;1.0772;0.9200;1.5496;1.6204;1.6706;0.1224]	9x1	72	double
dfbetas	<2x9 double>	2x9	144	double
dffit	[1.1947;0.1034;-0.1032;-0.2194;-0.3790;-0.0500;0.0021;0.0859;0.8269]	9x1	72	double
dffits	[0.8536;0.1170;-0.1609;-0.3938;-0.6115;-0.0771;0.0029;0.1086;2.0088]	9x1	72	double
dwstat	<1x1 struct>	1x1	264	struct
fstat	<1x1 struct>	1x1	792	struct
hatmat	<9x9 double>	9x9	648	double
leverage	[0.6546;0.2470;0.1329;0.1165;0.1600;0.1329;0.1600;0.1981;0.1981]	9x1	72	double
mse	2.7298	1x1	8	double
mydata	<9x2 double>	9x2	144	double
r	[0.6304;0.3152;-0.6739;-1.6630;-1.9891;-0.3261;0.0109;0.3478;3.3478]	9x1	72	double
rsquare	0.8089	1x1	8	double
s2_j	[2.9930;3.1628;3.0975;2.6630;2.3997;3.1643;3.1848;3.1596;0.8554]	9x1	72	double
standres	[0.6492;0.2199;-0.4380;-1.0709;-1.3136;-0.2119;0.0072;0.2351;2.2627]	9x1	72	double
studres	[0.6200;0.2043;-0.4112;-1.0842;-1.4010;-0.1969;0.0066;0.2185;4.0421]	9x1	72	double
tstat	<1x1 struct>	1x1	692	struct
yhat	[2.3696;5.6848;7.6739;9.6630;10.9891;10.3261;10.9891;11.6522;11.6522]	9x1	72	double

Ordinary Least Squares (parametric) multivariate regression (with/without interactions)

Ok now that we have finished our small overview of the univariate parametric linear regression techniques we saw during the theoretical course we will now focus on some of the multivariate techniques we have studied in details.

We will for the MATLAB examples, as always until now, use the same data as the one used in the theoretical course.

1st method (coefficient and constant can not be extracted)

We begin first with the tests the student-t test for slopes and constants using the following datas:

Month	Costs	CostA	CostB	CostC
1	44439	515	541	928
2	43936	929	692	711
3	44464	800	710	824
4	41533	979	675	758
5	46343	1165	1147	635
6	44922	651	939	901
7	43203	847	755	580
8	43000	942	908	589
9	40967	630	738	682
10	48582	1113	1175	1050
11	45003	1086	1075	984
12	44303	843	640	828
13	42070	500	752	708
14	44353	813	989	804
15	45968	1190	823	904
16	47781	1200	1108	1120
17	43202	731	590	1065
18	44074	1089	607	1132
19	44610	786	513	839

To obtain the coefficient and student t-test the first method use the function **LinearModel.fit()**:

```

1 - mydata=dataset('File','C:\tmp\MultivariateOLS.csv','Delimiter',';');
2 - mydata=mat2dataset([mydata.CostA mydata.CostB mydata.CostC mydata.Costs],...
3 -     'VarNames',{'CostA','CostB','CostC','Costs'});
4 - %simple linear model (without interactions)
5 - mdl = LinearModel.fit(mydata,'linear');|

```

We get then the following:

```
Command Window

mdl =

Linear regression model:
    Costs ~ 1 + CostA + CostB + CostC

Estimated Coefficients:

            Estimate      SE      tStat      pValue
(Intercept)    35103    1837.2    19.106    6.112e-12
CostA           2.066     1.665     1.2408    0.23373
CostB           4.1764    1.6813    2.4841    0.025288
CostC           4.7906    1.7893    2.6774    0.017223

Number of observations: 19, Error degrees of freedom: 15
Root Mean Squared Error: 1.25e+03
R-squared: 0.645, Adjusted R-Squared 0.575
F-statistic vs. constant model: 9.1, p-value = 0.00113
fx >>
```

And to obtain the ANOVA of the regression we just use the function **anova()** as we already saw it for the univariate regression:

```
Command Window

>> anova(mdl)

ans =

            SumSq      DF      MeanSq      F      pValue
CostA    2.4164e+06      1    2.4164e+06    1.5397    0.23373
CostB    9.6843e+06      1    9.6843e+06    6.1706    0.025288
CostC    1.125e+07      1    1.125e+07    7.1682    0.017223
Error    2.3541e+07     15    1.5694e+06

fx >> |
```

As for the univariate regression the we can explicitly write the model we are looking for:

```

Command Window
>> %and now with interactions (quadratic)
>> mdl=LinearModel.fit(mydata, 'Costs~1+CostA+CostB+CostC+CostA*CostB')

mdl =

Linear regression model:
    Costs ~ 1 + CostC + CostA*CostB

Estimated Coefficients:

```

	Estimate	SE	tStat	pValue
(Intercept)	45507	5909.1	7.7012	2.1269e-06
CostA	-8.2599	5.8238	-1.4183	0.17798
CostB	-8.7341	7.1917	-1.2145	0.24466
CostC	4.0788	1.7067	2.3899	0.03147
CostA:CostB	0.01322	0.0071883	1.8391	0.087205

```

Number of observations: 19, Error degrees of freedom: 14
Root Mean Squared Error: 1.16e+03
R-squared: 0.714, Adjusted R-Squared 0.633
F-statistic vs. constant model: 8.76, p-value = 0.000929
fx >>

```


2nd method (coefficient and constant and others can be extracted)

Now have a look to the second method that a in my point of view more useful because we can access directly to the coefficients, confidence interval and residues of the model. This second method use also as for the univariate regression the function **regress()**:

```

Command Window

>> format long g
>> X=[mydata.CostA mydata.CostB mydata.CostC ones(length(mydata),1)];
>> [coeff ConfInterval residues]=regress(mydata.Costs,X)

coeff =

    2.06595329599594
    4.17635553110151
    4.79064103726411
   35102.9004487403

ConfInterval =

   -1.48287136062231    5.61477795261419
    0.592850514346559    7.75986054785645
    0.976804034023853    8.60447804050436
   31186.9439838019   39018.8569136788

residues =

   1567.01037891478
    617.645134262442
    795.636272675249
   -2042.81461526004
   1000.92710858829
    886.48858788868

```

If we want to change the model by adding an interaction we just have to write the input matrix as following:

```

Command Window

>> format long g
>> X=[mydata.CostA mydata.CostB mydata.CostC mydata.CostA.*mydata.CostB ones(length(mydata),1)];
>> [coeff ConfInterval residues]=regress(mydata.Costs,X)

coeff =

    -8.25991226944437
    -8.73406200196337
     4.07879041751122
     0.013219889266409
    45507.0898903344

ConfInterval =

    -20.7507480687893     4.23092352990059
    -24.158626582437     6.69050257851026
     0.418369690375435     7.73921114464701
    -0.00219746795492074     0.0286372464877387
    32833.3399678668     58180.8398128019

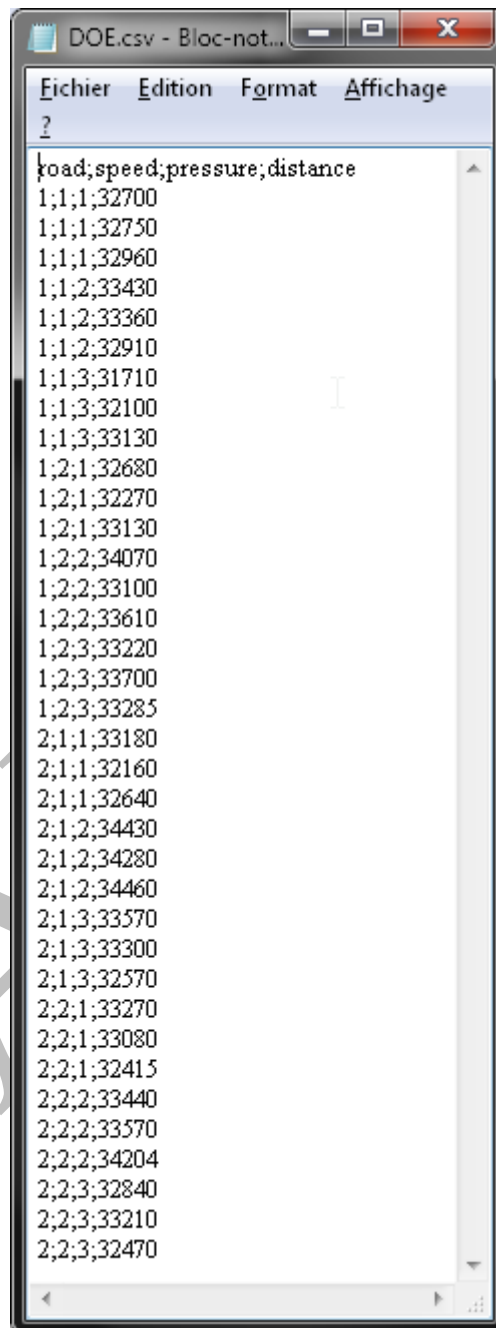
residues =

    442.51551608069
    747.675753569783
    896.203539265538
    1018.68888815101

```

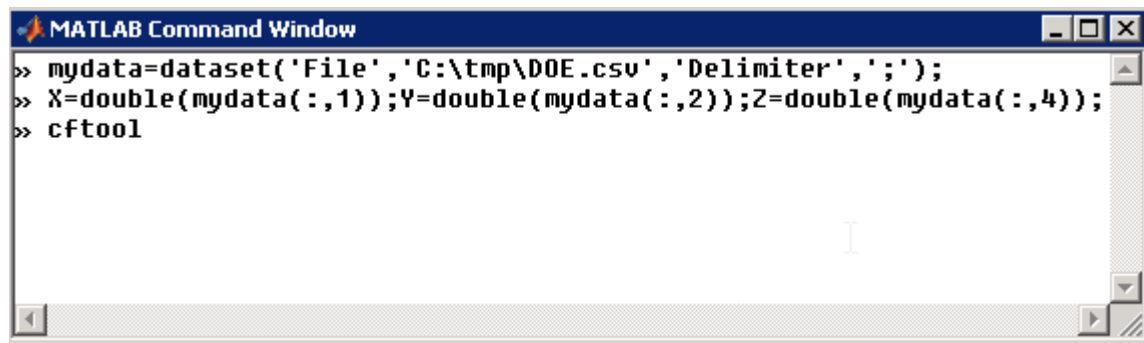
Polynomial multivariate regression using OLS and Curve fit Toolbox

We will see here that the curvefit tool is limited to 2 explanatory variables... Consider the following dataset (the same as in the theoretical course):



```
load,speed,pressure,distance
1,1,1,32700
1,1,1,32750
1,1,1,32960
1,1,2,33430
1,1,2,33360
1,1,2,32910
1,1,3,31710
1,1,3,32100
1,1,3,33130
1,2,1,32680
1,2,1,32270
1,2,1,33130
1,2,2,34070
1,2,2,33100
1,2,2,33610
1,2,3,33220
1,2,3,33700
1,2,3,33285
2,1,1,33180
2,1,1,32160
2,1,1,32640
2,1,2,34430
2,1,2,34280
2,1,2,34460
2,1,3,33570
2,1,3,33300
2,1,3,32570
2,2,1,33270
2,2,1,33080
2,2,1,32415
2,2,2,33440
2,2,2,33570
2,2,2,34204
2,2,3,32840
2,2,3,33210
2,2,3,32470
```

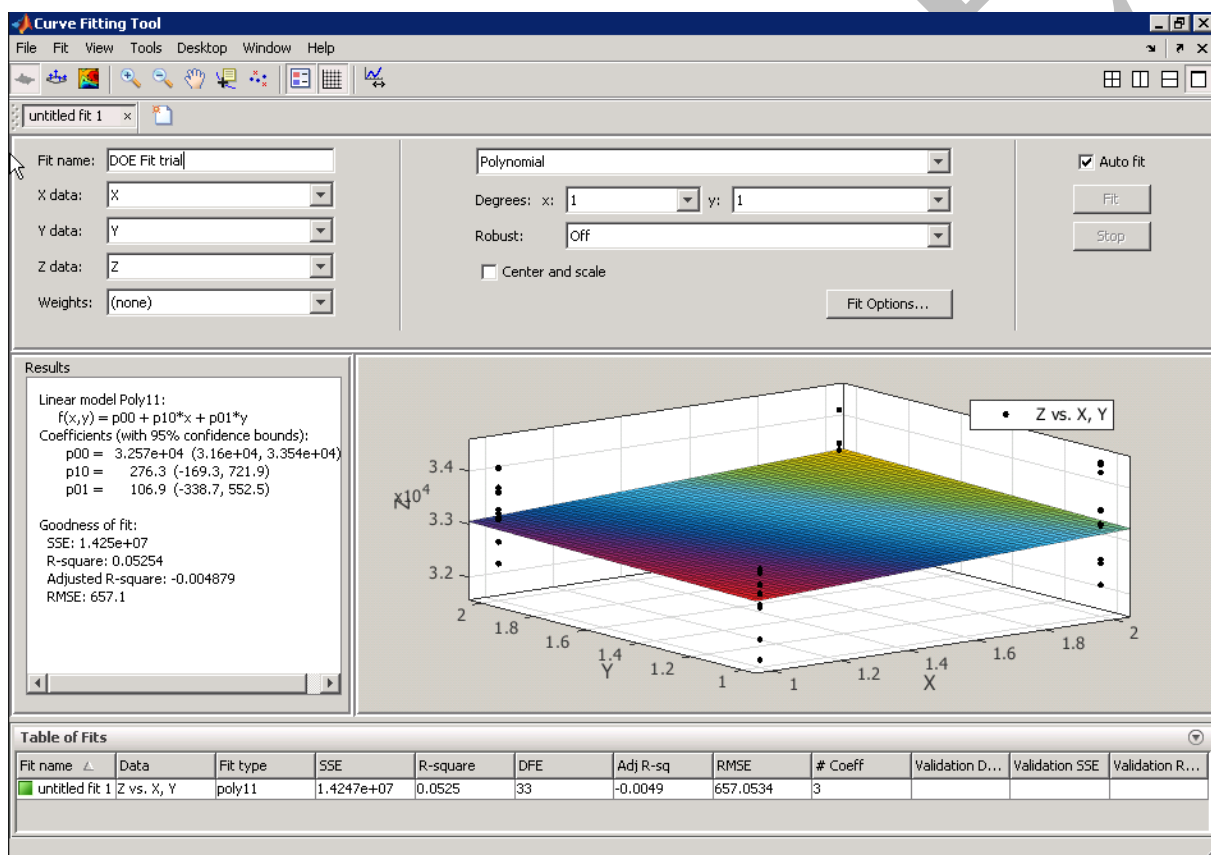
We import the data using the same technic as always and we launch the curve fitting toolbox by typing **cftool**:



```

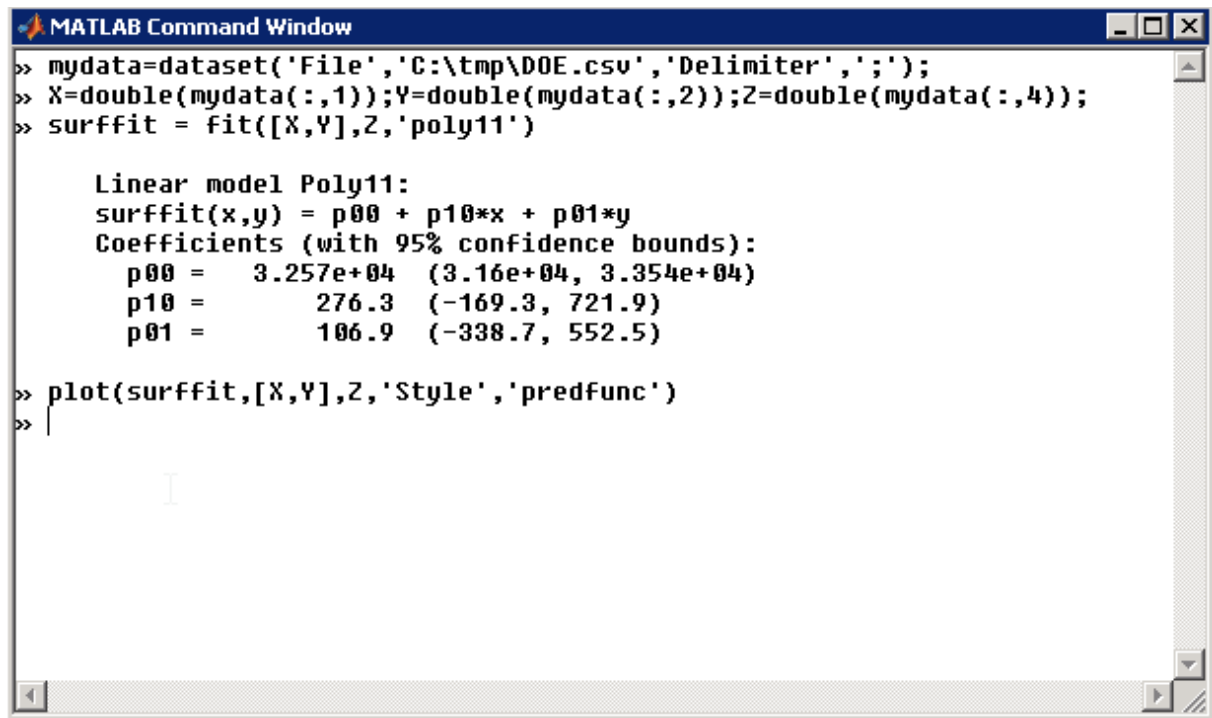
>> mydata=dataset('File','C:\tmp\DOE.csv','Delimiter',';');
>> X=double(mydata(:,1));Y=double(mydata(:,2));Z=double(mydata(:,4));
>> cftool
  
```

The following appears:



As you can see the tool is limited to two explanatory variables and also without interaction. In my point of view this makes the tool almost useless but perhaps there is an unknown tip to me to manager more than two explanatory variables.

We can otherwise the same stuff by using the function **fit()** and by indicating we want a first order polynom on x and y this is why we write the argument *poly11*:



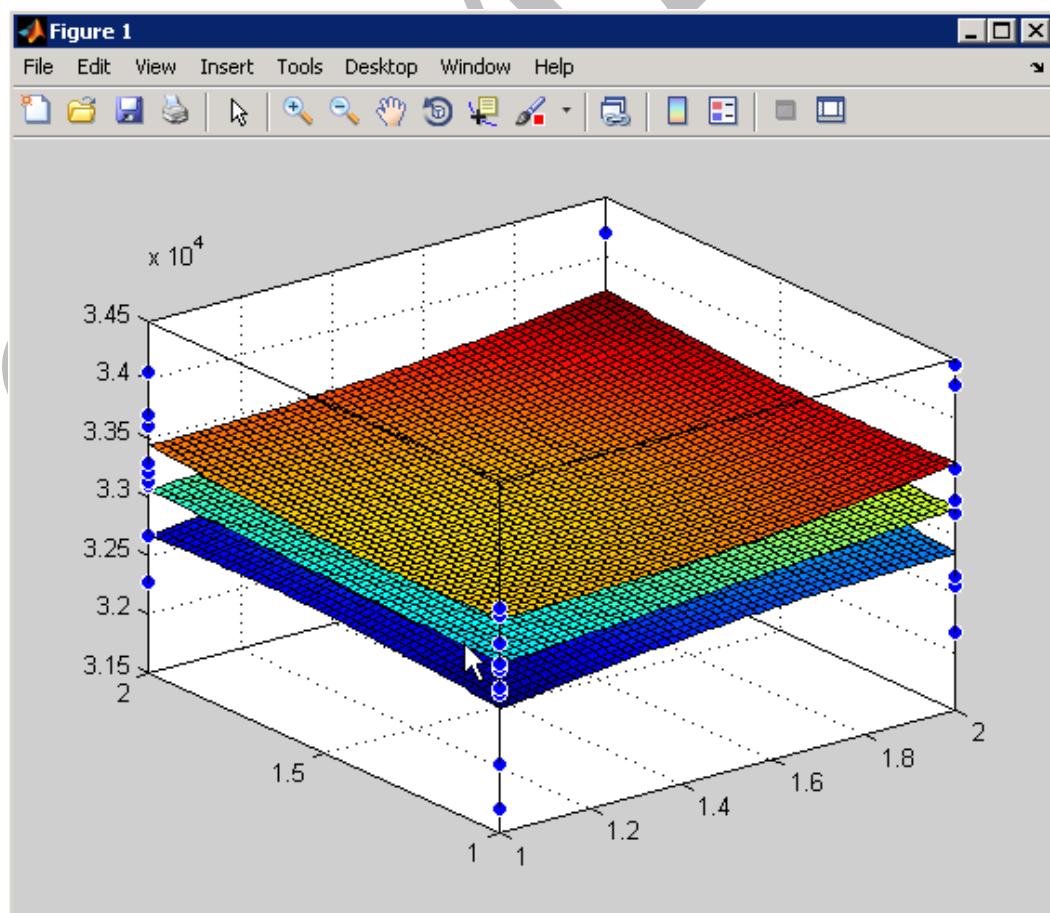
```

MATLAB Command Window
>> mydata=dataset('File','C:\tmp\DOE.csv','Delimiter',';');
>> X=double(mydata(:,1));Y=double(mydata(:,2));Z=double(mydata(:,4));
>> surffit = fit([X,Y],Z,'poly11')

Linear model Poly11:
surffit(x,y) = p00 + p10*x + p01*y
Coefficients (with 95% confidence bounds):
p00 = 3.257e+04 (3.16e+04, 3.354e+04)
p10 = 276.3 (-169.3, 721.9)
p01 = 106.9 (-338.7, 552.5)

>> plot(surffit,[X,Y],Z,'Style','predfunc')
>>
  
```

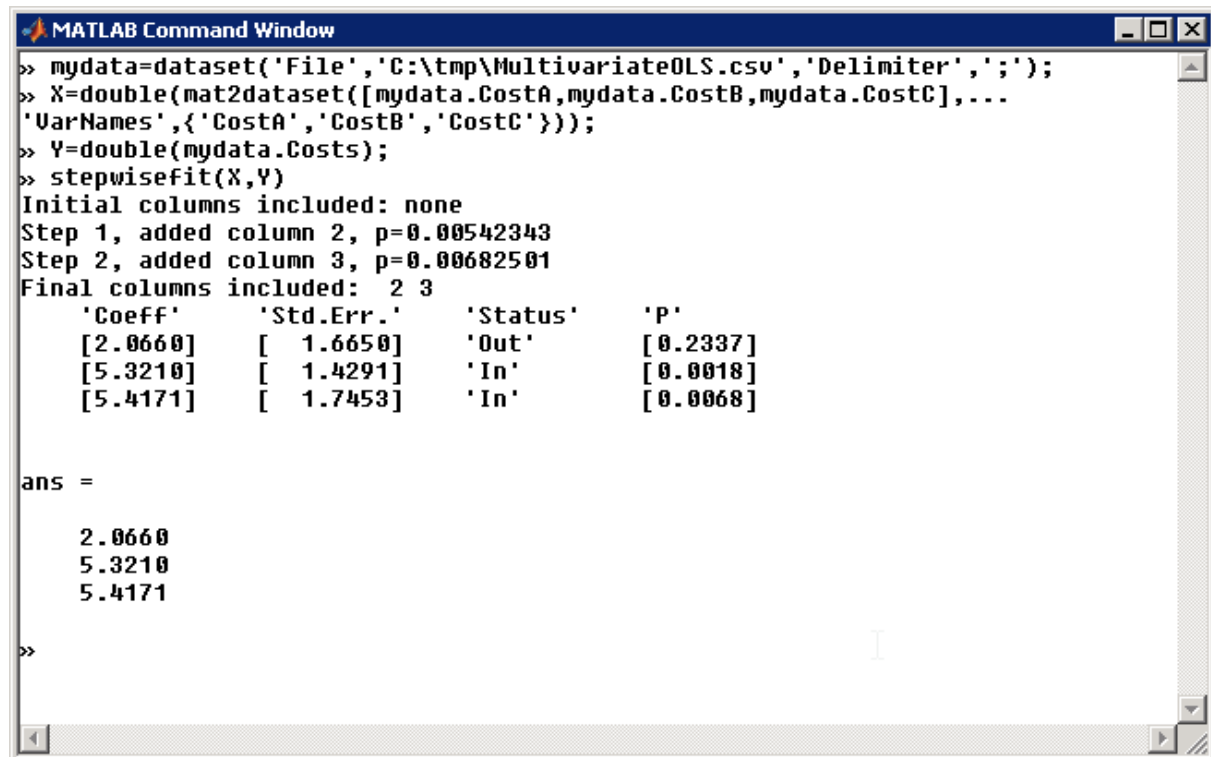
We the get the coefficients, the confidence intervals and also the plot of the function itself:



(Forward) Stepwise regression

Stepwise regression is a must know method to choose the most accurate explanatory variable and to not get too much degrees of freedom that biased the R squares.

For this purpose we use the **stepwisefit()** function:



```

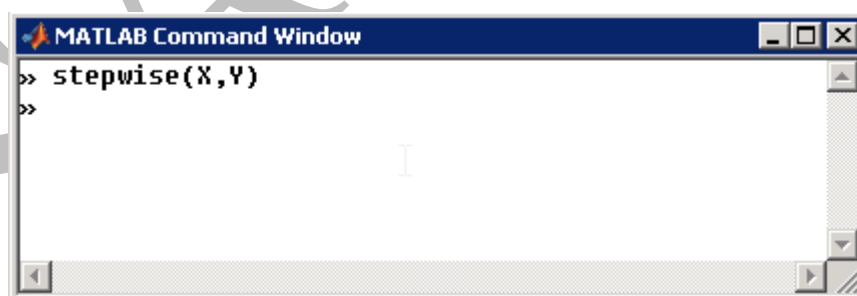
>> mydata=dataset('File','C:\tmp\MultivariateOLS.csv','Delimiter',';');
>> X=double(mat2dataset([mydata.CostA,mydata.CostB,mydata.CostC],...
'VarNames',{'CostA','CostB','CostC'}));
>> Y=double(mydata.Costs);
>> stepwisefit(X,Y)
Initial columns included: none
Step 1, added column 2, p=0.00542343
Step 2, added column 3, p=0.00682501
Final columns included: 2 3
    'Coeff'    'Std.Err.'    'Status'    'p'
    [2.0660]    [ 1.6650]    'Out'       [0.2337]
    [5.3210]    [ 1.4291]    'In'        [0.0018]
    [5.4171]    [ 1.7453]    'In'        [0.0068]

ans =

    2.0660
    5.3210
    5.4171
  
```

As we can see it we get the same results as in the theoretical course.

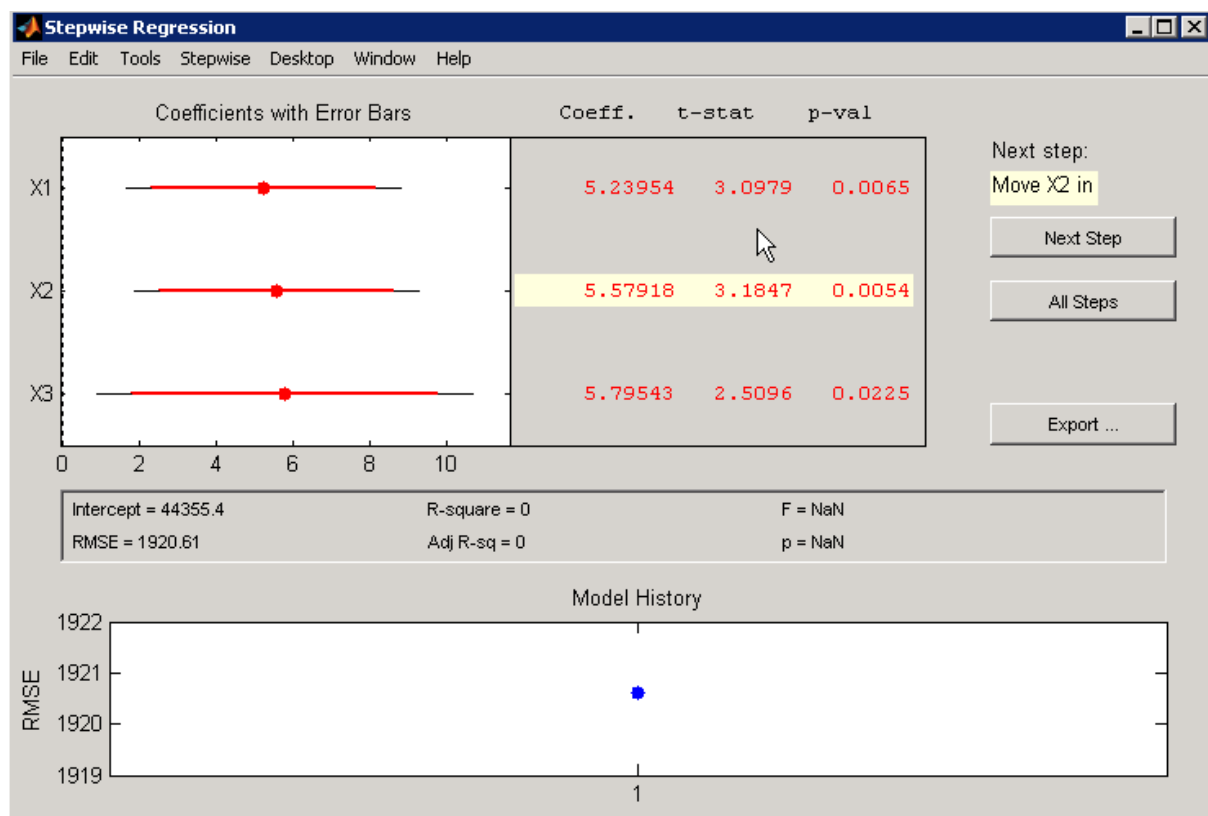
Now if instead of **stepwisefit()** function we use **stepwise()** function:



```

>> stepwise(X,Y)
>>
  
```

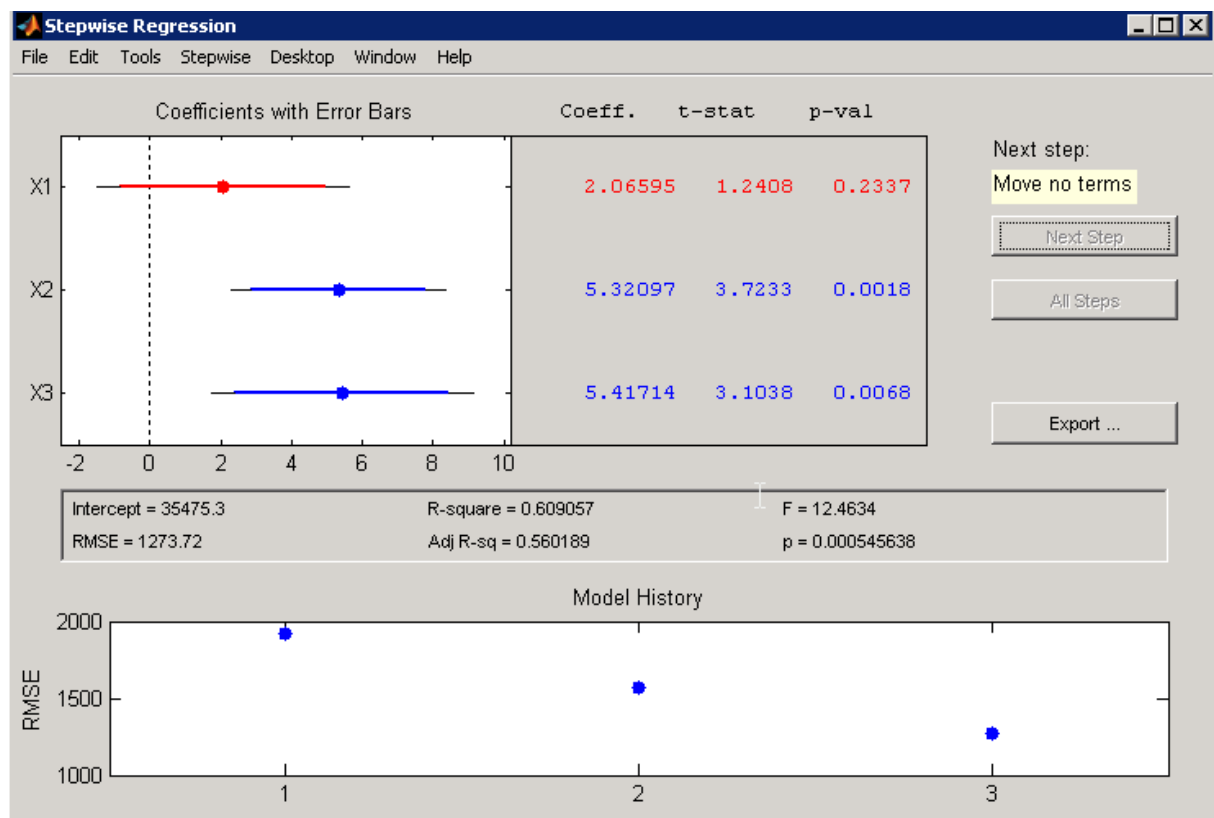
We get a figure that resume the situation step by step with a wizard. This is the first step:



This is the second step when you click on **Next step**:



And this is the last one:

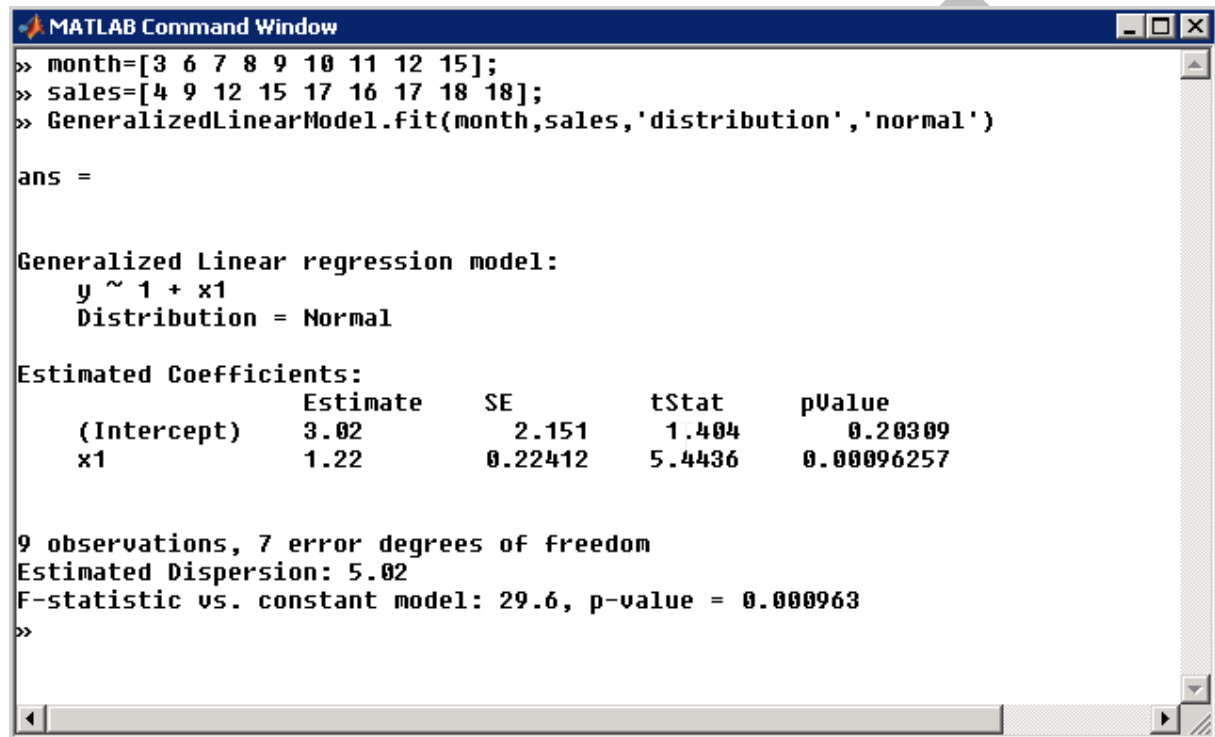


General Linear Models (GLM)

Ok this is a very big subject that has needed a lot of time in the theoretical course. We haven't proved all the post-hoc tests of all these regression models but we saw what was the idea behind them.

Normal regression

Using the function **GeneralizedLinearModel.fit()** consider first the Normal parametric regression model:



```
MATLAB Command Window
>> month=[3 6 7 8 9 10 11 12 15];
>> sales=[4 9 12 15 17 16 17 18 18];
>> GeneralizedLinearModel.fit(month,sales,'distribution','normal')

ans =

Generalized Linear regression model:
  y ~ 1 + x1
  Distribution = Normal

Estimated Coefficients:
              Estimate      SE      tStat      pValue
(Intercept)    3.02         2.151     1.404     0.20309
x1             1.22         0.22412    5.4436    0.00096257

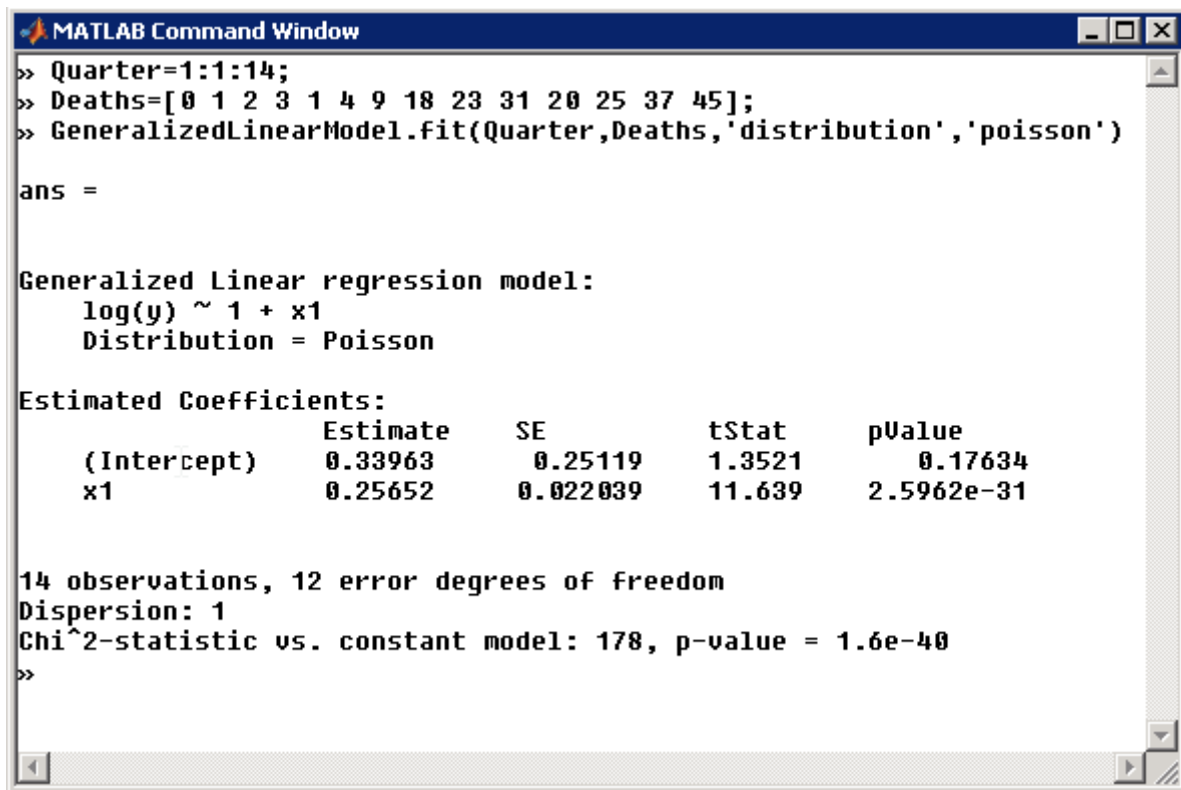
9 observations, 7 error degrees of freedom
Estimated Dispersion: 5.02
F-statistic vs. constant model: 29.6, p-value = 0.000963
>>
```

The results are as expected.

Log-linear Poisson regression

The log-linear Poisson regression model was not easy to prove in the theoretical course. We saw that we have to take especially care of model assumption because Poisson mean is equal to its variance. But apart of this, let us take a look if we obtain the same results as the calculation made by hand using the same data and the function

GeneralizedLinearModel.fit():



```

>> Quarter=1:1:14;
>> Deaths=[0 1 2 3 1 4 9 18 23 31 20 25 37 45];
>> GeneralizedLinearModel.fit(Quarter,Deaths,'distribution','poisson')

ans =

Generalized Linear regression model:
    log(y) ~ 1 + x1
    Distribution = Poisson

Estimated Coefficients:
               Estimate      SE      tStat      pValue
    (Intercept)    0.33963    0.25119    1.3521    0.17634
             x1      0.25652    0.022039   11.639   2.5962e-31

14 observations, 12 error degrees of freedom
Dispersion: 1
Chi^2-statistic vs. constant model: 178, p-value = 1.6e-40
>>

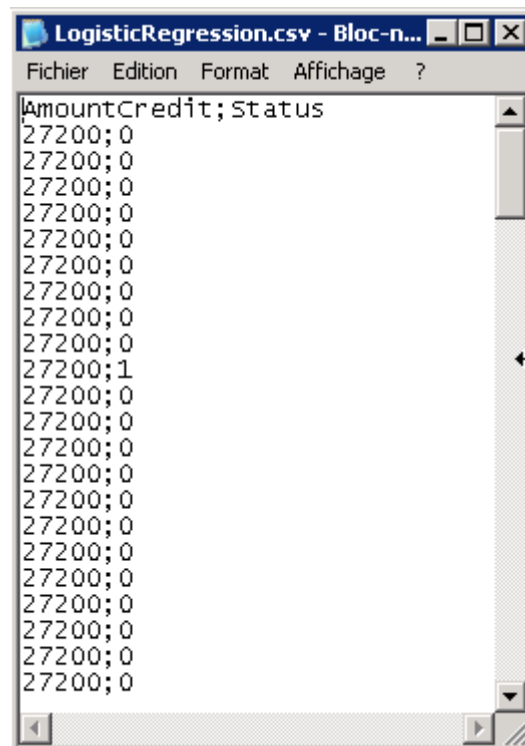
```

As we see it we obtain the same results!

Binomial (logistic) regression

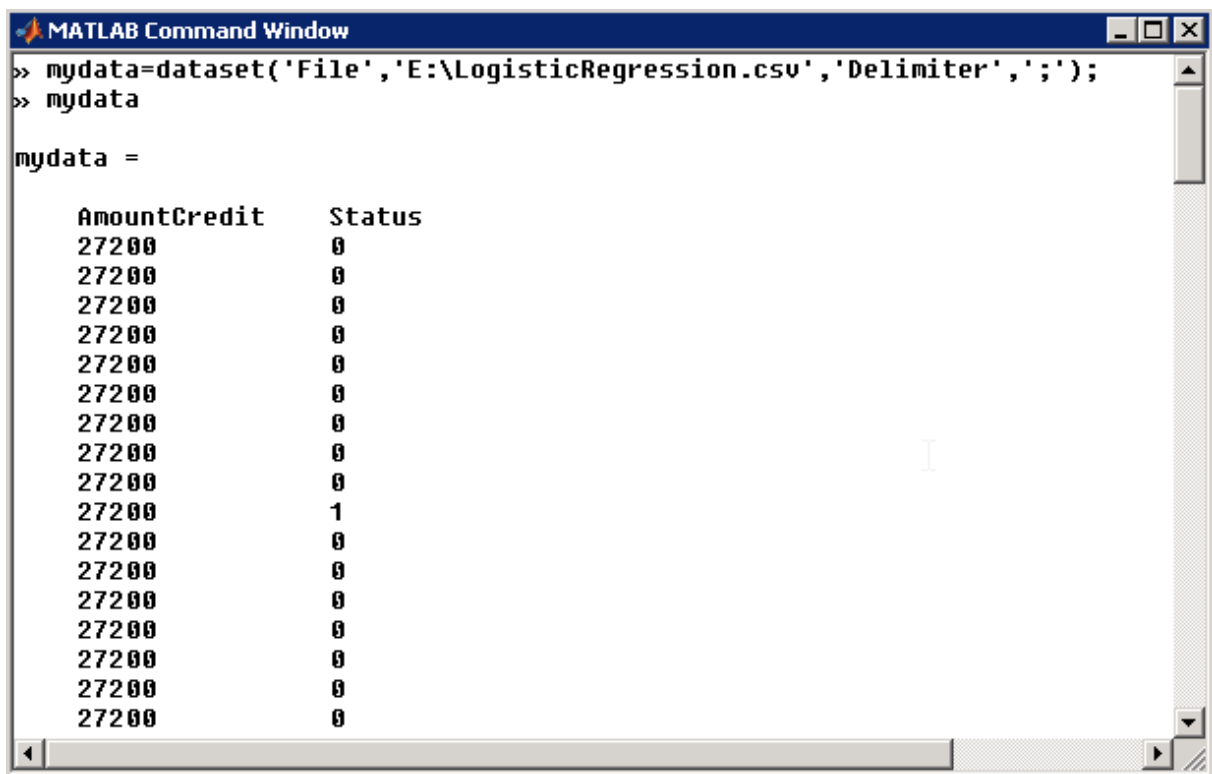
The binomial logistic regression is the second most important GLM, used a lot in Credit Risk or Social Sciences we have discussed and make a lot of proofs about this model in the theoretical course. We did not prove all the corresponding confidence interval tests but the major results also this is why we can still see how to use it in MATLAB as we did it for R and Minitab.

Consider the same datas:



```
LogisticRegression.csv - Bloc-n...
Fichier Edition Format Affichage ?
AmountCredit;Status
27200;0
27200;0
27200;0
27200;0
27200;0
27200;0
27200;0
27200;0
27200;0
27200;0
27200;1
27200;0
27200;0
27200;0
27200;0
27200;0
27200;0
27200;0
27200;0
27200;0
27200;0
27200;0
```

First we load the data as we always do:



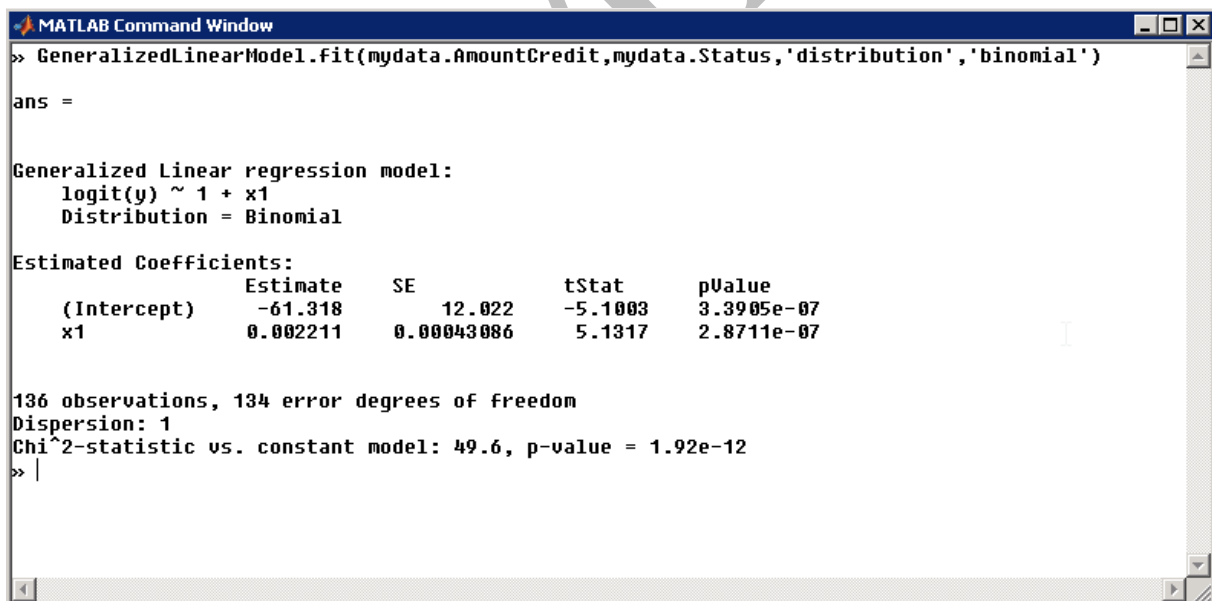
```

MATLAB Command Window
>> mydata=dataset('File','E:\LogisticRegression.csv','Delimiter',';');
>> mydata

mydata =

    AmountCredit    Status
    27200          0
    27200          0
    27200          0
    27200          0
    27200          0
    27200          0
    27200          0
    27200          0
    27200          0
    27200          1
    27200          0
    27200          0
    27200          0
    27200          0
    27200          0
    27200          0
    27200          0
  
```

And we regress using once again the function **GeneralizedLinearModel.fit()**:



```

MATLAB Command Window
>> GeneralizedLinearModel.fit(mydata.AmountCredit,mydata.Status,'distribution','binomial')

ans =

Generalized Linear regression model:
logit(y) ~ 1 + x1
Distribution = Binomial

Estimated Coefficients:
              Estimate      SE      tStat      pValue
(Intercept)   -61.318       12.022   -5.1003   3.3905e-07
x1             0.002211     0.00043086    5.1317   2.8711e-07

136 observations, 134 error degrees of freedom
Dispersion: 1
Chi^2-statistic vs. constant model: 49.6, p-value = 1.92e-12
>> |
  
```

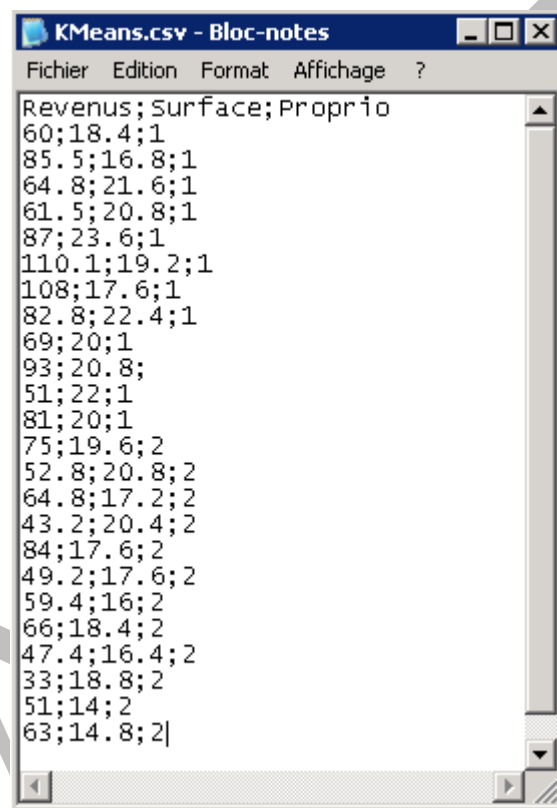
And the estimate and the coefficient corresponds to what we get by hand and in the other softwares.

Data Mining/Machine Learning

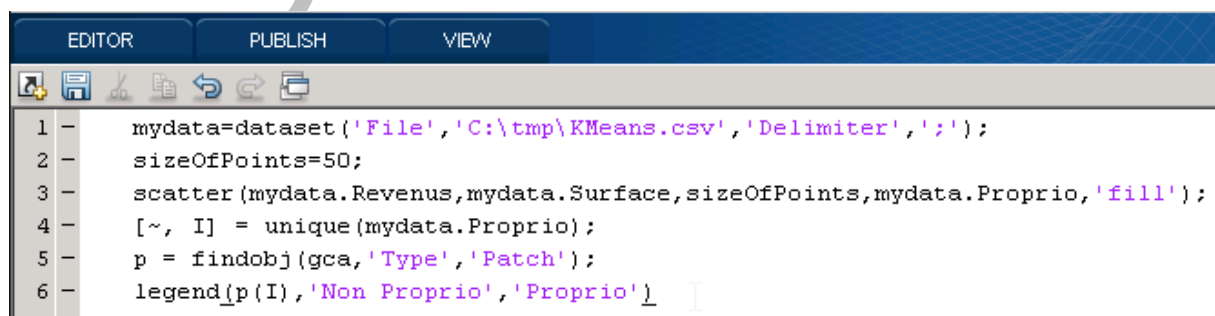
As we saw in the Numerical Methods theoretical course Data Mining/Machine learning is a very big subject. Even if my favorite software for this purpose remains at this day Tanagra first and R in second and RapidMiner in the third position let us have a look to the possibility of some MATLAB core functions in this area.

K-means (euclidian full)

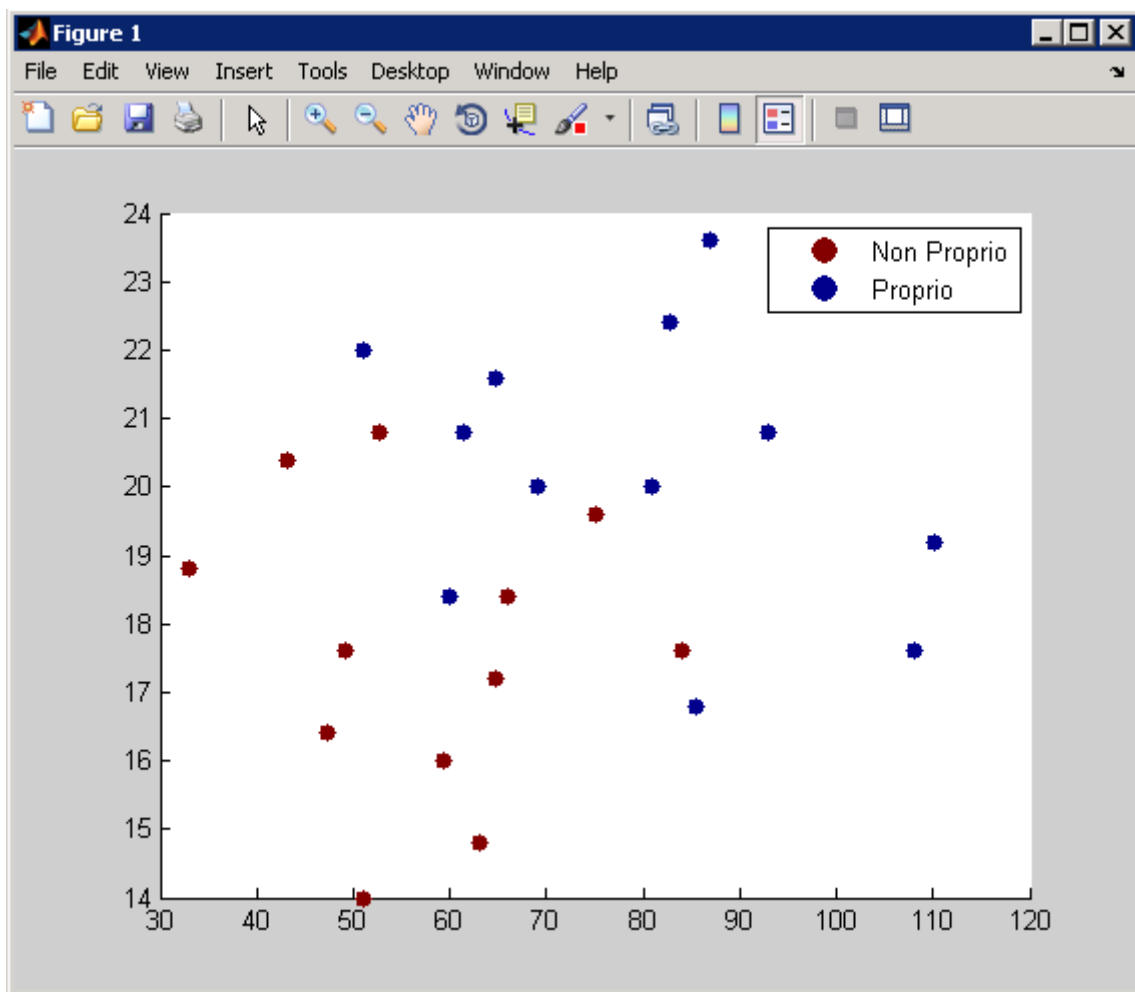
The k-means algorithm is the most easy machine learning algorithm implemented in MATLAB. To see if we found with MATLAB the same results as calculations made by hand and with Minitab, Tanagra and R we will use the same dataset:



First we simply plots the actual values of the dataset:



We can guess that a pattern exists:



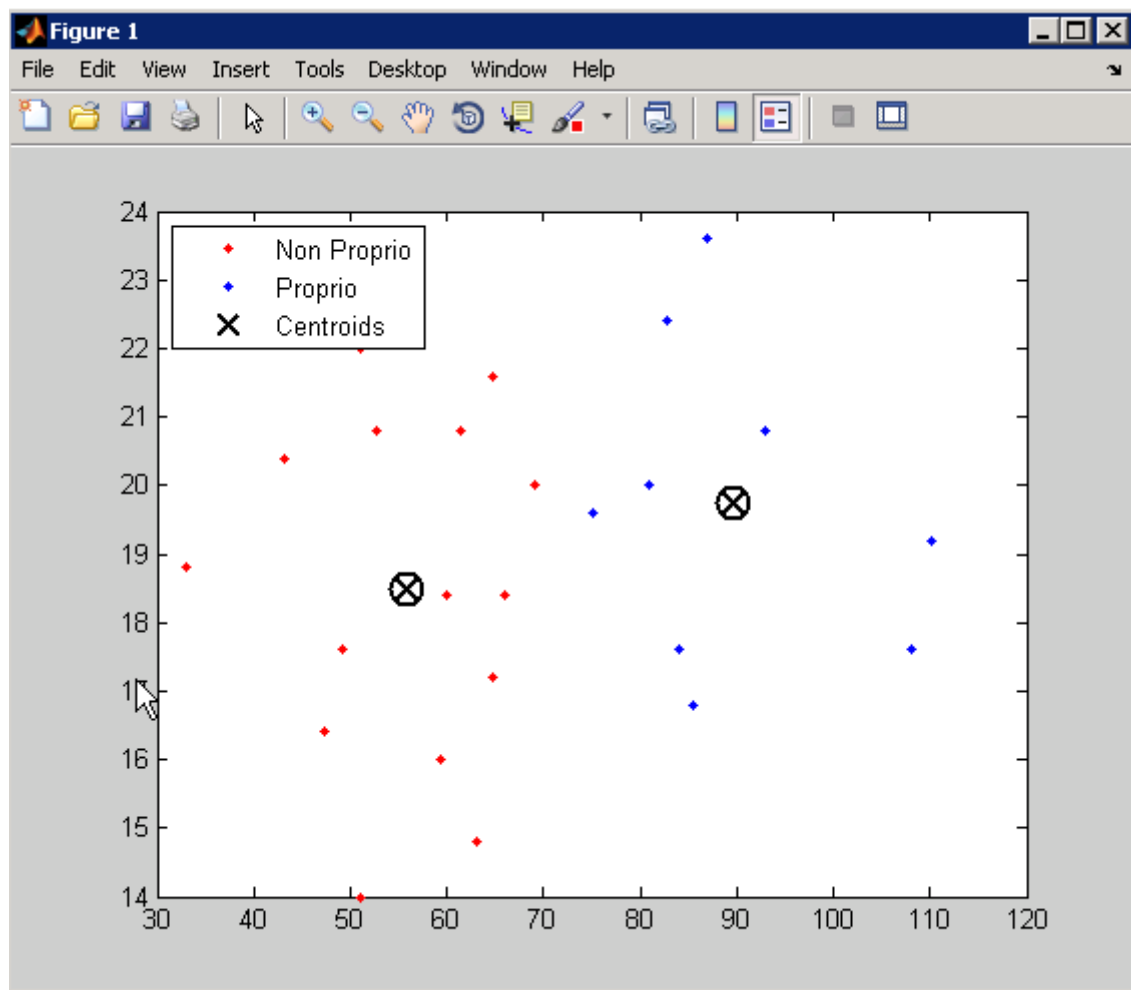
Now we use the **kmeans()** with implicitly the euclidian distance to found two clusters:

```

EDITOR    PUBLISH    VIEW
1 - mydata=dataset('File','C:\tmp\KMeans.csv','Delimiter',';');
2 - sizeOfPoints=50;
3 - scatter(mydata.Revenus,mydata.Surface,sizeOfPoints,mydata.Proprio,'fill');
4 - [~, I] = unique(mydata.Proprio);
5 - p = findobj(gca,'Type','Patch');
6 - legend(p(I),'Non Proprio','Proprio')
7
8 - Clusters=2
9 - data=[mydata.Revenus,mydata.Surface];
10 - [idx,ctr]=kmeans(data,Clusters)
11 - plot(data(idx==1,1),data(idx==1,2),'r.','MarkerSize',12)
12 - hold on
13 - plot(data(idx==2,1),data(idx==2,2),'b.','MarkerSize',12)
14 - plot(ctr(:,1),ctr(:,2),'kx','MarkerSize',12,'LineWidth',2)
15 - plot(ctr(:,1),ctr(:,2),'ko','MarkerSize',12,'LineWidth',2)
16 - legend('Non Proprio','Proprio','Centroids','Location','NW')
17 - hold off

```

and script also plot now the two centroids with what will be consider as a *Proprio* and what will not be considered as a *Proprio*:



Another thing we can do is to get directly the classification from the original dataset and the coordinates of the centroids:

```
Command Window
>> [idx, ctrs] = kmeans(data, 2)

idx =

     2
     1
     2
     2
     1
     1
     1
     1
     1
     2
     1
     2
     1
     1
     2
     2
     2
     1
     2
     2
     2
     2
     2
     2
     2

ctrs =

    89.6000    19.7333
    55.7400    18.4800
```

Take care!!! The order of the centroids will change randomly at each execution!

ToDo: Add here a confusion matrix using **confusionmat()**

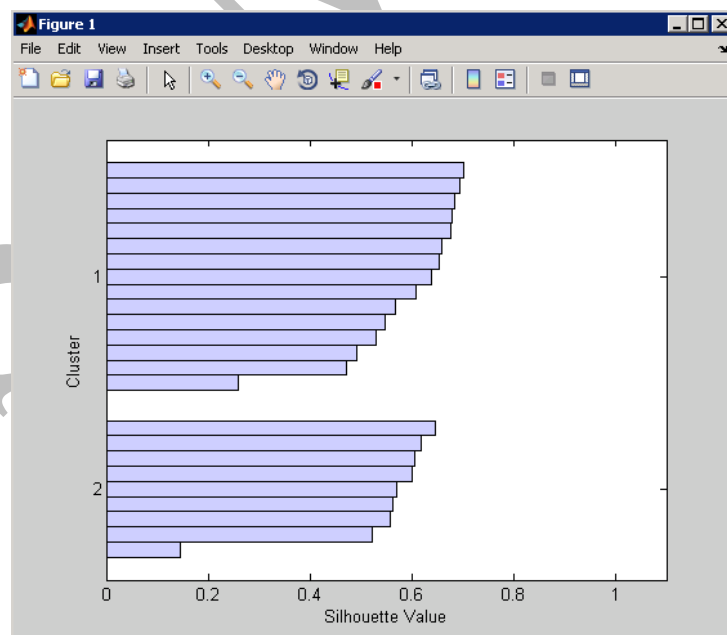
And what about the silhouette of the algorithm. For this purpose we use the function **silhouette()**:

```

1 - mydata=dataset('File','C:\tmp\KMeans.csv','Delimiter',';');
2 - sizeOfPoints=50;
3 - scatter(mydata.Revenus,mydata.Surface,sizeOfPoints,mydata.Proprio,'fill');
4 - [~, I] = unique(mydata.Proprio);
5 - p = findobj(gca,'Type','Patch');
6 - legend(p(I),'Non Proprio','Proprio')
7
8 - Clusters=2
9 - data=[mydata.Revenus,mydata.Surface];
10 - [idx,ctrs]=kmeans(data,Clusters)
11 - plot(data(idx==1,1),data(idx==1,2),'r.','MarkerSize',12)
12 - hold on
13 - plot(data(idx==2,1),data(idx==2,2),'b.','MarkerSize',12)
14 - plot(ctrs(:,1),ctrs(:,2),'kx','MarkerSize',12,'LineWidth',2)
15 - plot(ctrs(:,1),ctrs(:,2),'ko','MarkerSize',12,'LineWidth',2)
16 - legend('Non Proprio','Proprio','Centroids','Location','NW')
17 - hold off
18
19 - %silhouette qualitative cluster quality segmentation
20 - [silh3,h] = silhouette(data,idx,'city');
21 - set(get(gca,'Children'),'FaceColor',[.8 .8 1])
22 - xlabel('Silhouette Value')
23 - ylabel('Cluster')

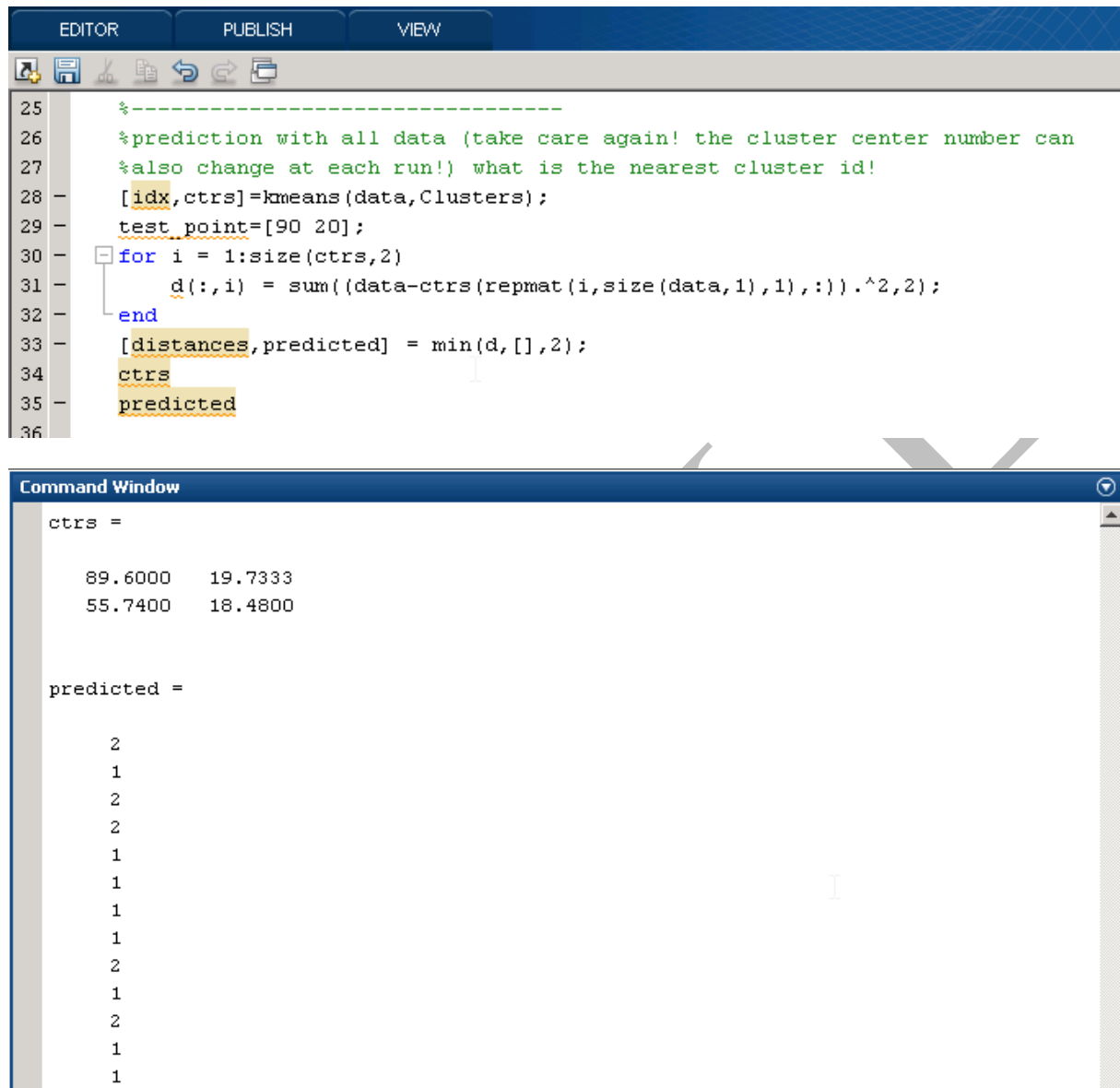
```

And we get:



Each bar represents the affectation of a point to the centroid and the height of the bar represents its normalized distance.

We can check that the system use the euclidian distance by calculating ourself the affectation of each point at each centroid:



The image shows a MATLAB Editor window with a script and a Command Window showing the results of the script.

Editor Window:

```

25 %-----
26 %prediction with all data (take care again! the cluster center number can
27 %also change at each run!) what is the nearest cluster id!
28 [idx, ctrs]=kmeans(data, Clusters);
29 test_point=[90 20];
30 for i = 1:size(ctrs,2)
31     d(:,i) = sum((data-ctrs(repmat(i,size(data,1),1),:)).^2,2);
32 end
33 [distances, predicted] = min(d,[],2);
34 ctrs
35 predicted
36

```

Command Window:

```

ctrs =

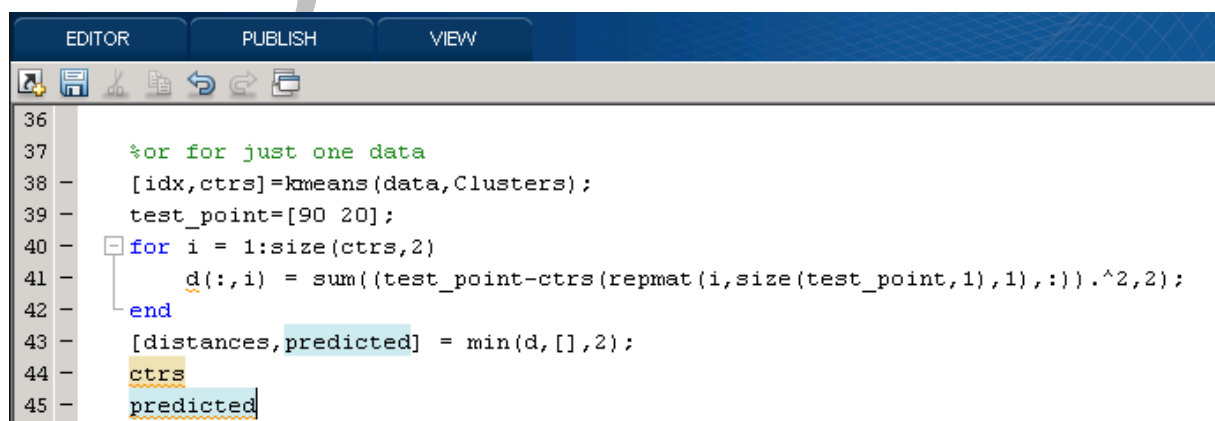
    89.6000    19.7333
    55.7400    18.4800

predicted =

     2
     1
     2
     2
     1
     1
     1
     1
     1
     2
     1
     2
     1
     1

```

Or much more interesting: We do the samething with a new point!:



The image shows a MATLAB Editor window with a script for testing a single point.

```

36
37 %or for just one data
38 [idx, ctrs]=kmeans(data, Clusters);
39 test_point=[90 20];
40 for i = 1:size(ctrs,2)
41     d(:,i) = sum((test_point-ctrs(repmat(i,size(test_point,1),1),:)).^2,2);
42 end
43 [distances, predicted] = min(d,[],2);
44 ctrs
45 predicted

```

Here is the whole part script for classification of new points for those that want a global overview of the last steps:

```

EDITOR PUBLISH VIEW
37 %or a prediction for just one data
38 clear all;close all;clc
39 mydata=dataset('File','C:\tmp\KMeans.csv','Delimiter',';');
40 Clusters=2;
41 data=[mydata.Revenus,mydata.Surface];
42 [idx,ctr]=kmeans(data,Clusters);
43 test_point=[90 20];
44 for i = 1:size(ctr,2)
45     d(:,i) = sum((test_point-ctr(repmat(i,size(test_point,1),1),:)).^2,2);
46 end
47 [distances,predicted] = min(d,[],2);
48 ctr
49 predicted

```

If you want to be able to reproduce the same results at each run thus seed the random generator:

```

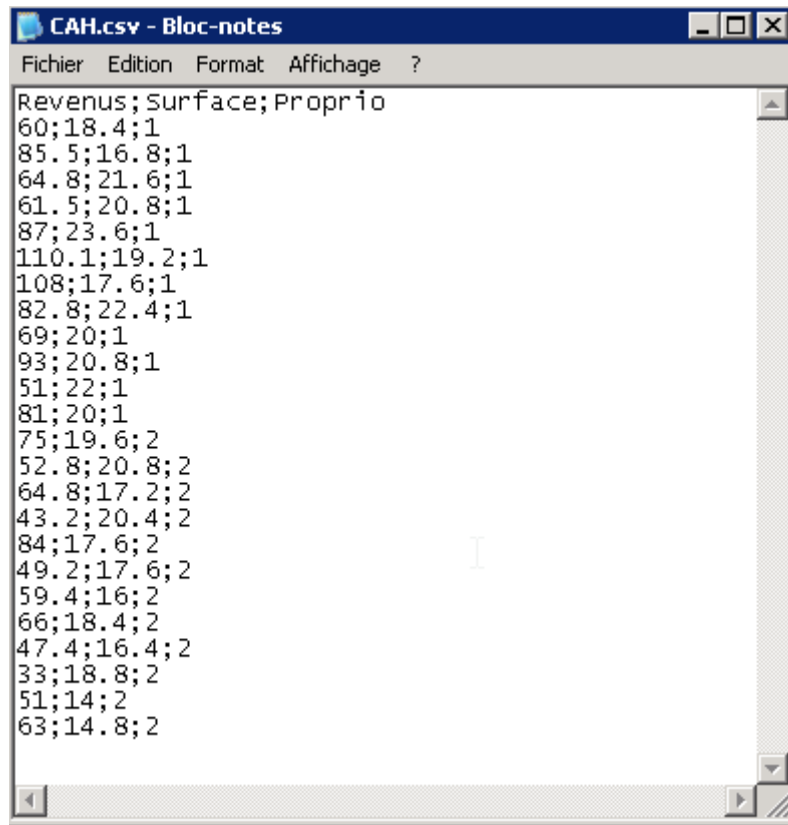
37 %or a prediction for just one data
38 clear all;close all;clc
39 rand('seed',0); %to keekp the same centroid order at each run!
40 mydata=dataset('File','C:\tmp\KMeans.csv','Delimiter',';');
41 Clusters=2;
42 data=[mydata.Revenus,mydata.Surface];
43 [idx,ctr]=kmeans(data,Clusters);
44 test_point=[90 20];
45 for i = 1:size(ctr,2)
46     d(:,i) = sum((test_point-ctr(repmat(i,size(test_point,1),1),:)).^2,2);
47 end
48 [distances,predicted] = min(d,[],2);
49 ctr
50 predicted

```

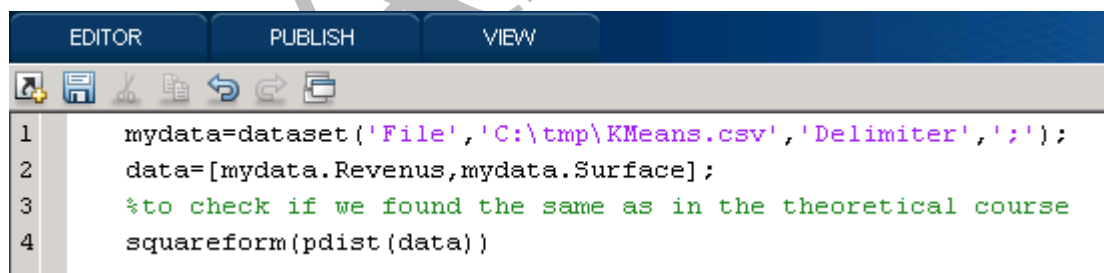
Hierarchical ascendant (CAH) Dendrogram

CAH Dendrogram was an easy part of the Data Mining course: easy to define, boring to calculate but without using complex mathematics...

We will also use the same data as in the theoretical course



First we see that in MATLAB using the functions **pdist()** and **squareform()** together:



So we can see if get the same matrix as the one calculated by hand (yes this was boring...):

Command Window							
ans =							
Columns 1 through 7							
0	25.5501	5.7689	2.8302	27.4962	50.1064	48.0067	
25.5501	0	21.2492	24.3311	6.9635	24.7168	22.5142	
5.7689	21.2492	0	3.3956	22.2899	45.3635	43.3848	
2.8302	24.3311	3.3956	0	25.6533	48.6263	46.6100	
27.4962	6.9635	22.2899	25.6533	0	23.5153	21.8403	
50.1064	24.7168	45.3635	48.6263	23.5153	0	2.6401	
48.0067	22.5142	43.3848	46.6100	21.8403	2.6401	0	
23.1482	6.2169	18.0178	21.3600	4.3681	27.4869	25.6531	
9.1411	16.8074	4.4944	7.5425	18.3565	41.1078	39.0738	
33.0872	8.5000	28.2113	31.5000	6.6212	17.1747	15.3375	
9.6933	34.8897	13.8058	10.5683	36.0355	59.1663	57.1696	
21.0609	5.5218	16.2788	19.5164	6.9971	29.1110	27.1065	
15.0479	10.8669	10.3942	13.5532	12.6491	35.1023	33.0606	
7.5895	32.9437	12.0266	8.7000	34.3144	57.3223	55.2927	
4.9477	20.7039	4.4000	4.8836	23.1041	45.3441	43.2019	
16.9186	42.4529	21.6333	18.3044	43.9167	66.9108	64.8605	
24.0133	1.7000	19.6122	22.7264	6.7082	26.1490	24.0000	
10.8296	36.3088	16.1047	12.7094	38.2732	60.9210	58.8000	
2.4739	26.1123	7.7795	5.2393	28.6273	50.8009	48.6263	
6.0000	19.5655	3.4176	5.1000	21.6342	44.1073	42.0076	
12.7577	38.1021	18.1604	14.7706	40.2492	62.7625	60.6119	
27.0030	52.5381	31.9230	28.5701	54.2129	77.1010	75.0096	

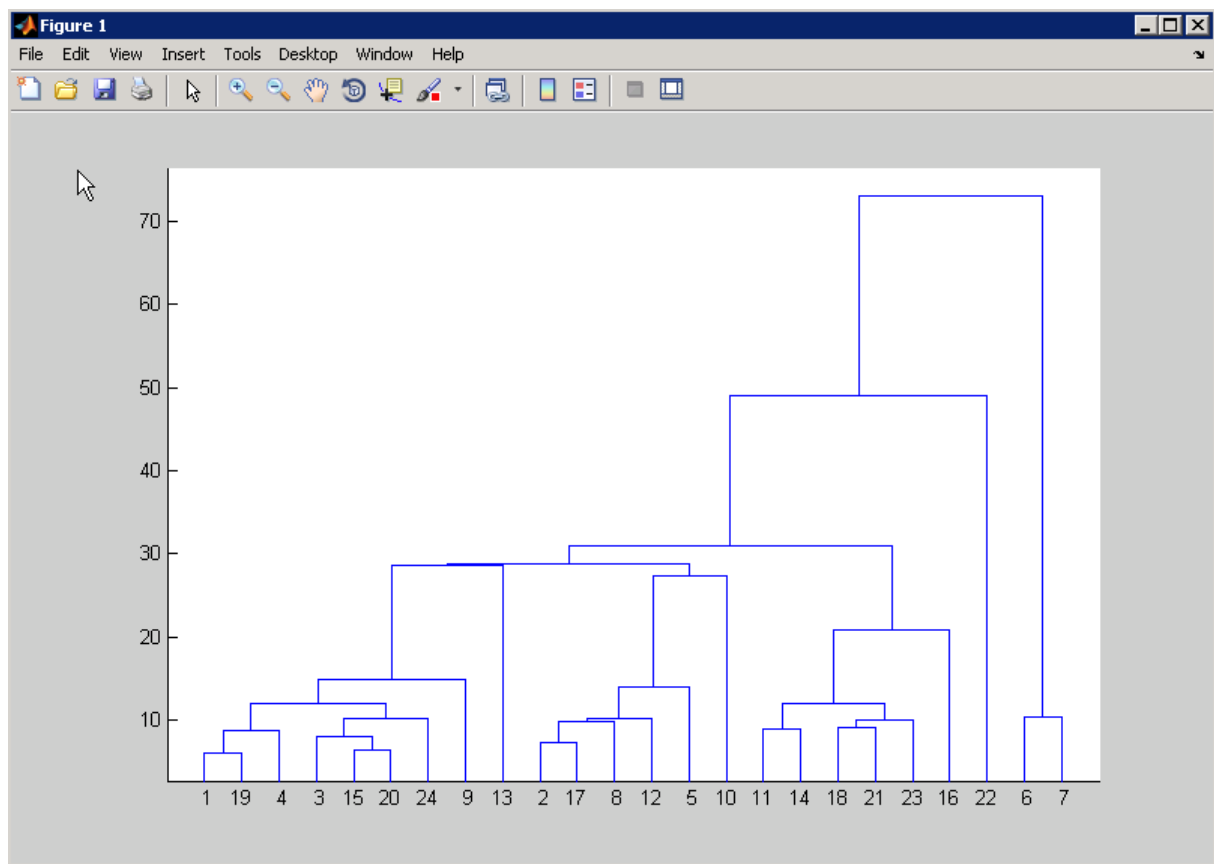
Now if we use the **dendrogram()** function:

```

EDITOR PUBLISH VIEW
1 mydata=dataset('File','C:\tmp\KMeans.csv','Delimiter',';');
2 data=[mydata.Revenus,mydata.Surface];
3 %to check if we found the same as in the theoretical course
4 squareform(pdist(data))
5 dendrogram(linkage(squareform(pdist(data))))

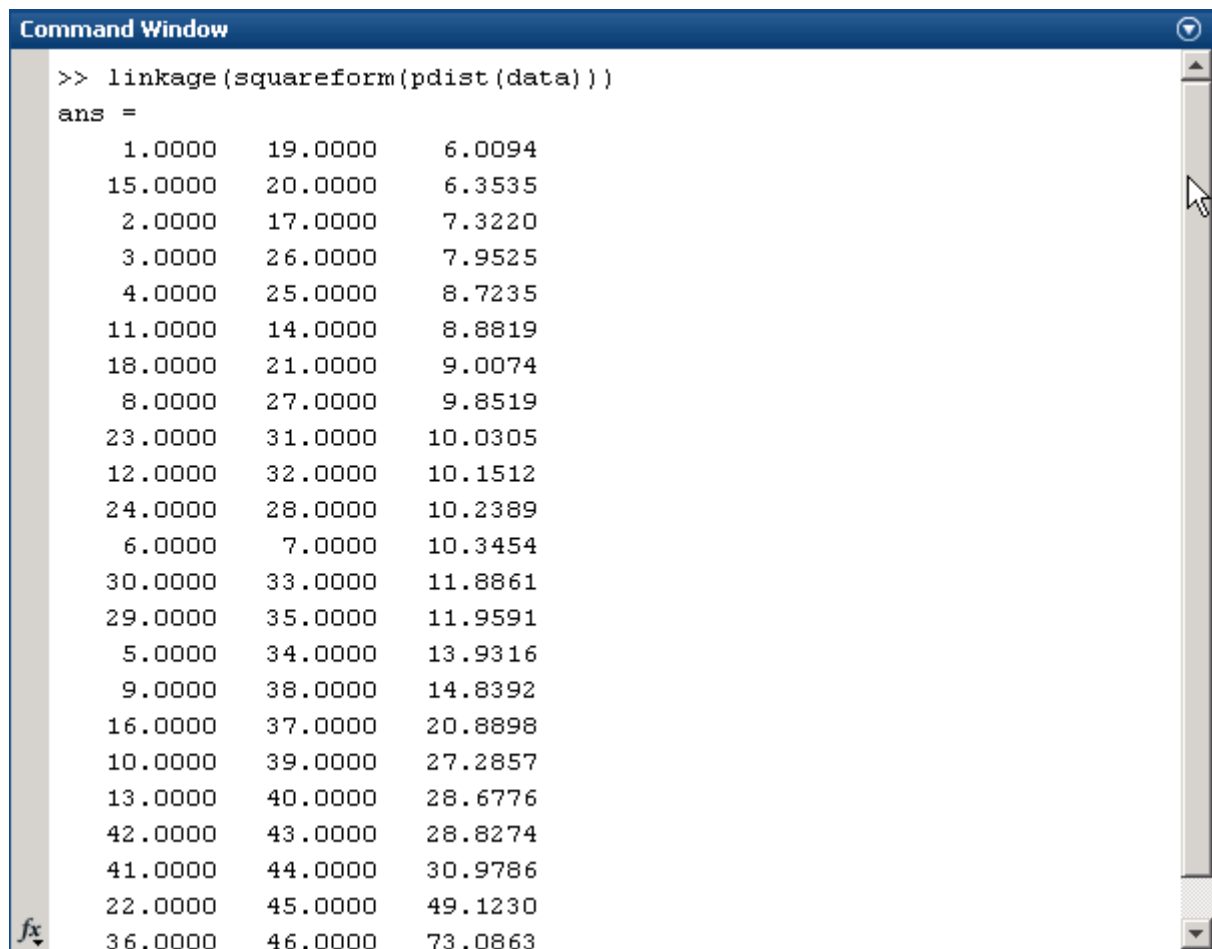
```

We get:



So we get the same thing as in the theoretical course! Nice!

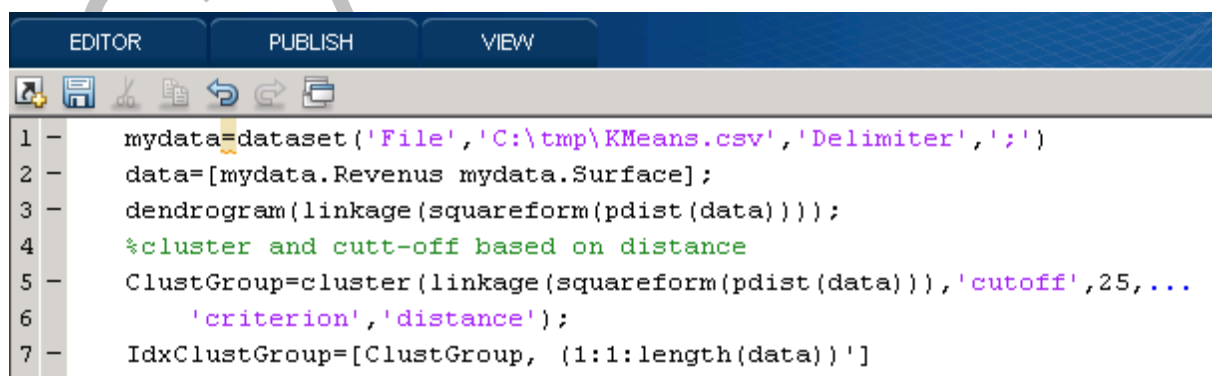
Now have a look to what we call the linkages. Each row represents a couple at the bottom of the tree. But the distance shown are a little bit strange. They do not correspond with what we get in the theoretical course (that were perfectly corresponding with the distance given by Minitab):



```

Command Window
>> linkage(squareform(pdist(data)))
ans =
    1.0000    19.0000     6.0094
   15.0000    20.0000     6.3535
     2.0000    17.0000     7.3220
     3.0000    26.0000     7.9525
     4.0000    25.0000     8.7235
   11.0000    14.0000     8.8819
   18.0000    21.0000     9.0074
     8.0000    27.0000     9.8519
   23.0000    31.0000    10.0305
   12.0000    32.0000    10.1512
   24.0000    28.0000    10.2389
     6.0000     7.0000    10.3454
   30.0000    33.0000    11.8861
   29.0000    35.0000    11.9591
     5.0000    34.0000    13.9316
     9.0000    38.0000    14.8392
   16.0000    37.0000    20.8898
   10.0000    39.0000    27.2857
   13.0000    40.0000    28.6776
   42.0000    43.0000    28.8274
   41.0000    44.0000    30.9786
   22.0000    45.0000    49.1230
   36.0000    46.0000    73.0863
  
```

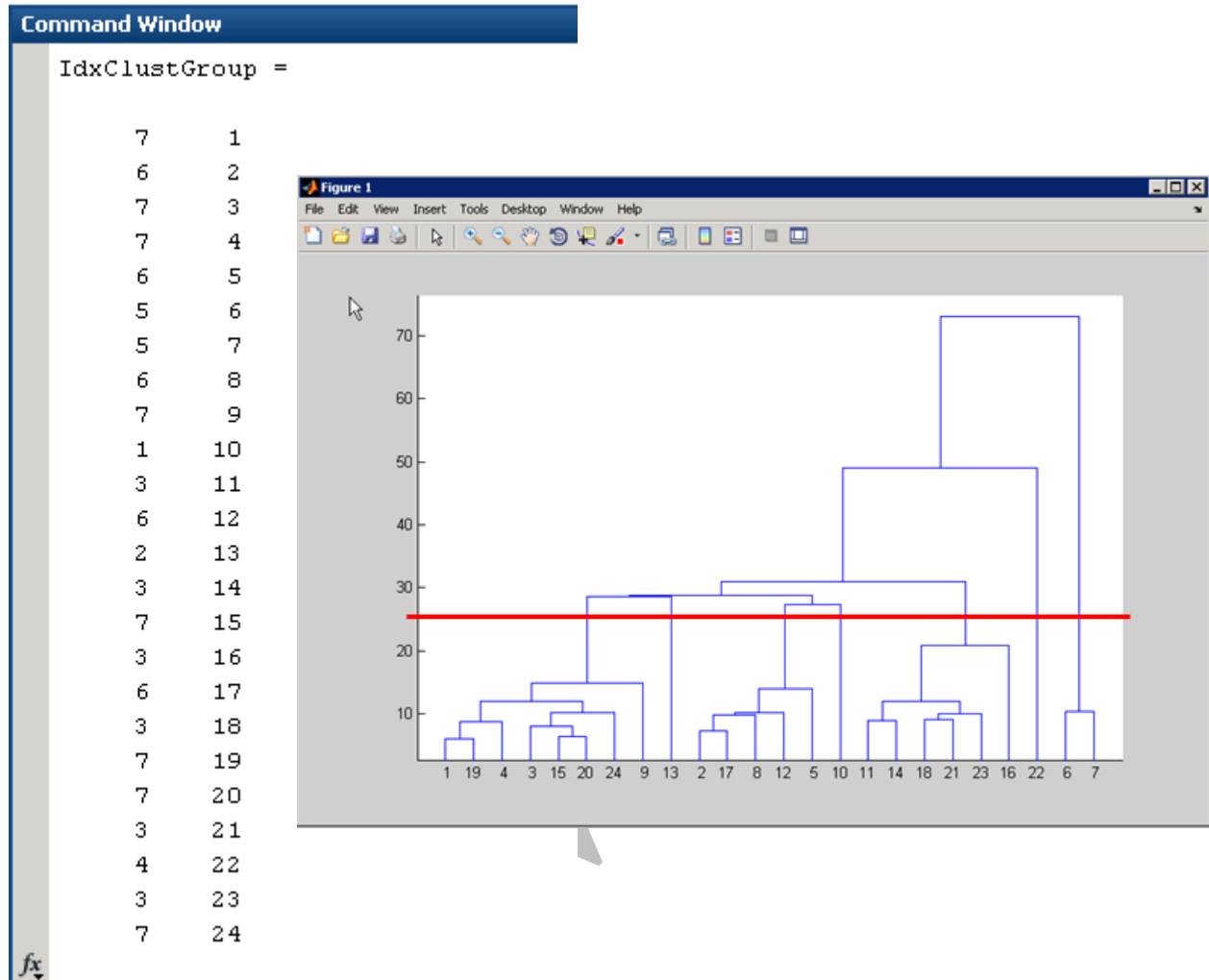
Now using the function **cluster()** with a given **cutoff** high we can calculate row data belongs to which category:



```

EDITOR PUBLISH VIEW
1 - mydata=dataset('File','C:\tmp\KMeans.csv','Delimiter',';')
2 - data=[mydata.Revenus mydata.Surface];
3 - dendrogram(linkage(squareform(pdist(data)))));
4 - %cluster and cutt-off based on distance
5 - ClustGroup=cluster(linkage(squareform(pdist(data))), 'cutoff',25,...
6 -     'criterion','distance');
7 - IdxClustGroup=[ClustGroup, (1:1:length(data))']
  
```

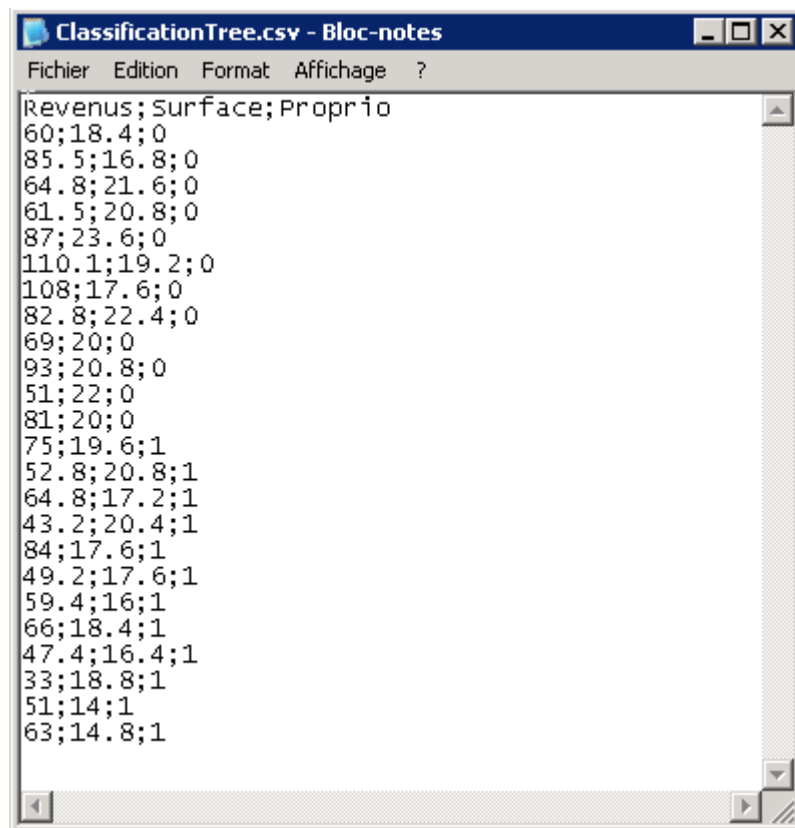
Following this cluster rule with this empirical cluster cutoff high we get (the second column contains the originals row data line number this is why there are 24 rows) the following result in the *Command Window*. Focus on the fact that at the cutoff of 25, as you can see on the chart, there are 7 clusters (seven branch) numbered for right to left:



CART Classification trees (Gini index based)

In the Data Mining course we also see how to create a classification system based on a technique called "CART" (Classification And Regression Tree) using the empirical Gini index.

First we will use the same data:



We import the data as usual in a dataset, we define the *Proprio* column as being logical (binary) with the function **logical()** et after we run the **classregtree()** function with the *gdi* parameter to say that we want an Gini index classification:

```

1  mydata=dataset('File','C:\tmp\KMeans.csv','Delimiter',';');
2  data=[mydata.Surface,mydata.Revenus];
3  Proprio=logical(mydata.Proprio);
4  tree_full=classregtree(data,Proprio,'names',{'Surface' 'Revenus'},...
5  'splitcriterion','gdi','minleaf',1)
6  view(tree_full)

```

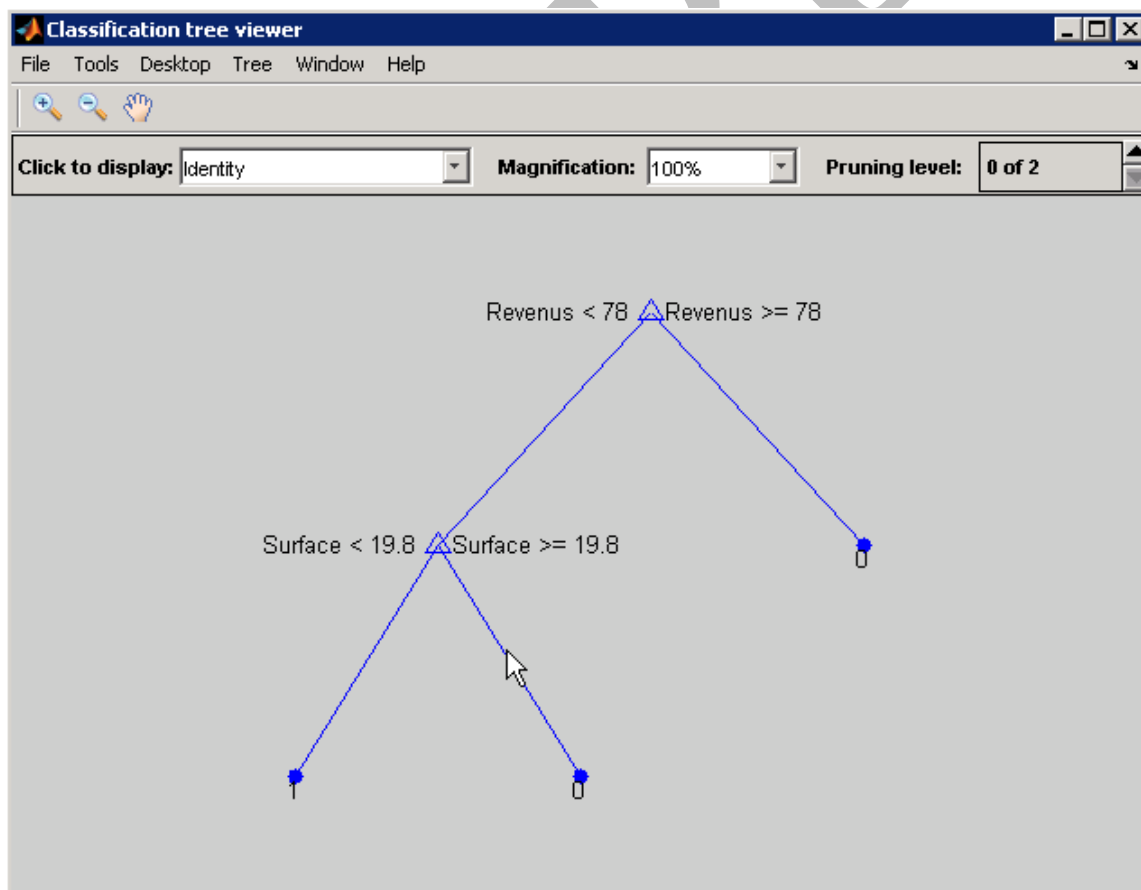
To see the result we use the function `view()` on the tree object:

```

Command Window
>> mydata=dataset('File','C:\tmp\KMeans.csv','Delimiter',';');
data=[mydata.Surface,mydata.Revenus];
Proprio=logical(mydata.Proprio);
tree_full=classregtree(data,Proprio,'names',{'Surface' 'Revenus'},...
    'splitcriterion','gdi','minleaf',1)
view(tree_full)
tree_full =
Decision tree for classification
1  if Revenus<78 then node 2 elseif Revenus>=78 then node 3 else 0
2  if Surface<19.8 then node 4 elseif Surface>=19.8 then node 5 else 1
3  class = 0
4  class = 1
5  class = 0
fx >>

```

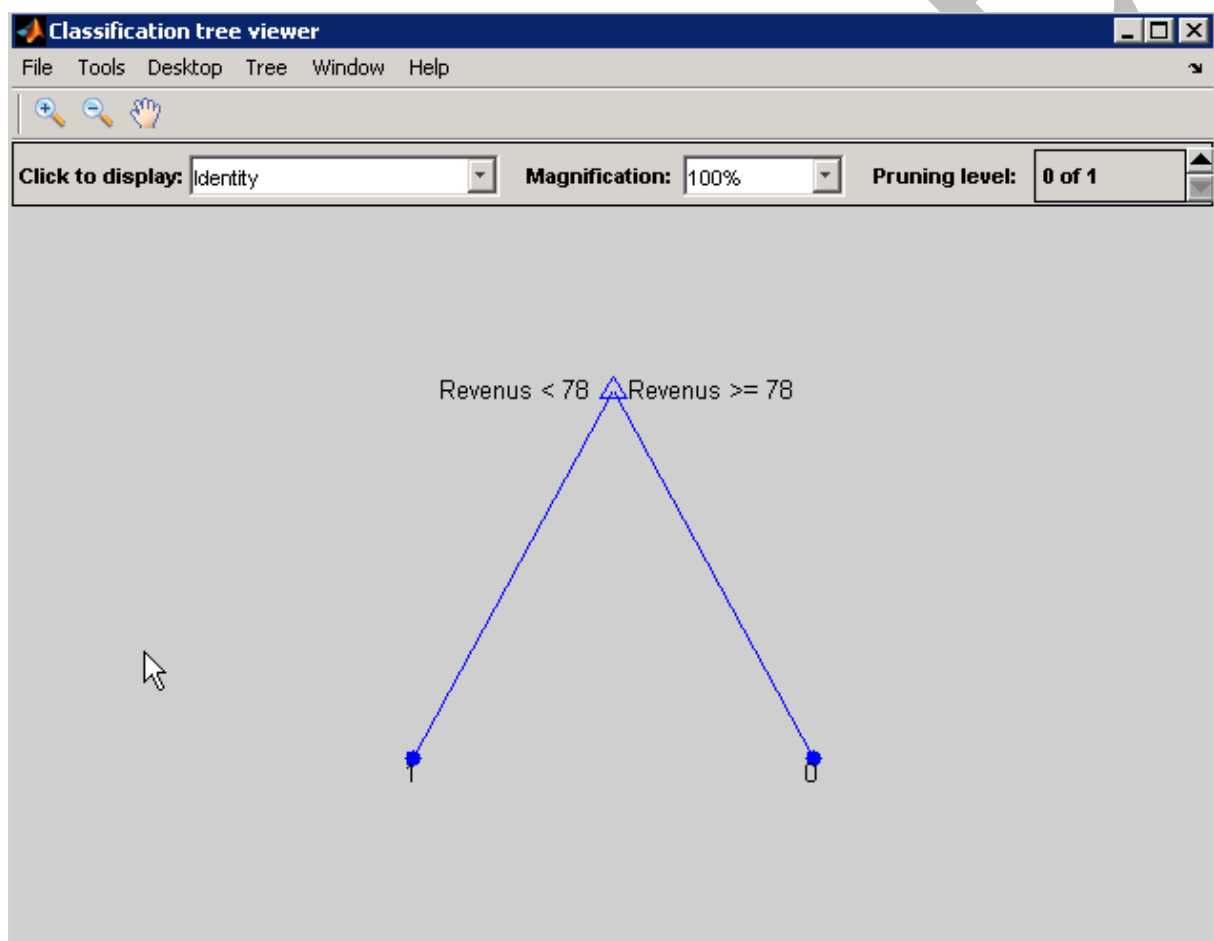
we also get the following tree:



The last value of 19.8 correspond perfectly with what we get in the theoretical course but not the second value of 78 (in the theoretical course we get 84.75)... I have to investigate where this difference comes from.

```
EDITOR PUBLISH VIEW
mydata=dataset('File','C:\tmp\KMeans.csv','Delimiter',';');
data=[mydata.Surface,mydata.Revenus];
Proprio=logical(mydata.Proprio);
tree_full=classregtree(data,Proprio,'names',{'Surface' 'Revenus'},...
'splitcriterion','gdi','minleaf',1)
view(tree_full)
%some tree pruning for fun!
tree_pruned = prune(tree_full,'level',1);
view(tree_pruned)
```

We can make tree pruning if we want:



We can also predict the classification for a new input but the given output is false in comparison to the drawn tree above and I must investigate why!:

```
Command Window

>> eval(tree_full,[75 19])

ans =

    '0'

fx >> |
```

But for the whole dataset we get something good (??? O_o ??):

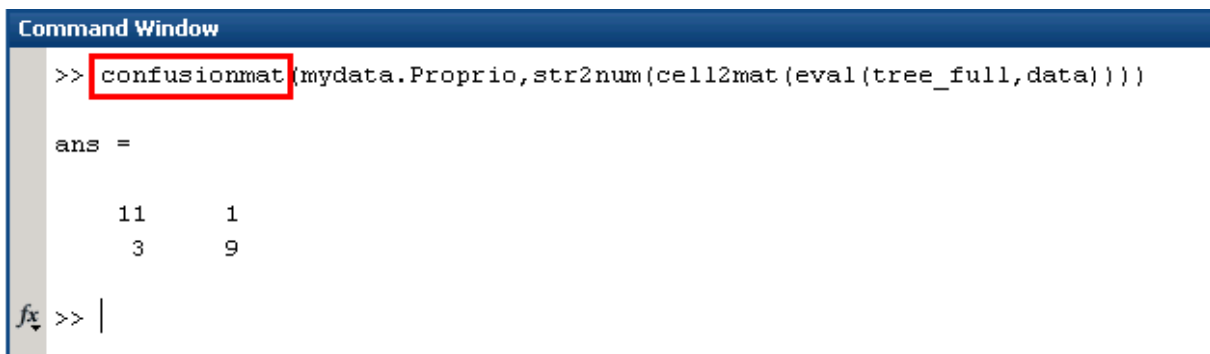
```
Command Window

>> [mydata.Proprio str2num(cell2mat(eval(tree_full,data)))]

ans =

     0     1
     0     0
     0     0
     0     0
     0     0
     0     0
     0     0
     0     0
     0     0
     0     0
     0     0
     0     0
     1     1
     1     0
     1     1
     1     0
     1     0
     1     1
     1     1
     1     1
     1     1
     1     1
     1     1
```

We can use the function **confusionmat()** to quickly get the confusion matrix:



```
Command Window
>> confusionmat(mydata.Proprio, str2num(cell2mat(eval(tree_full, data))))

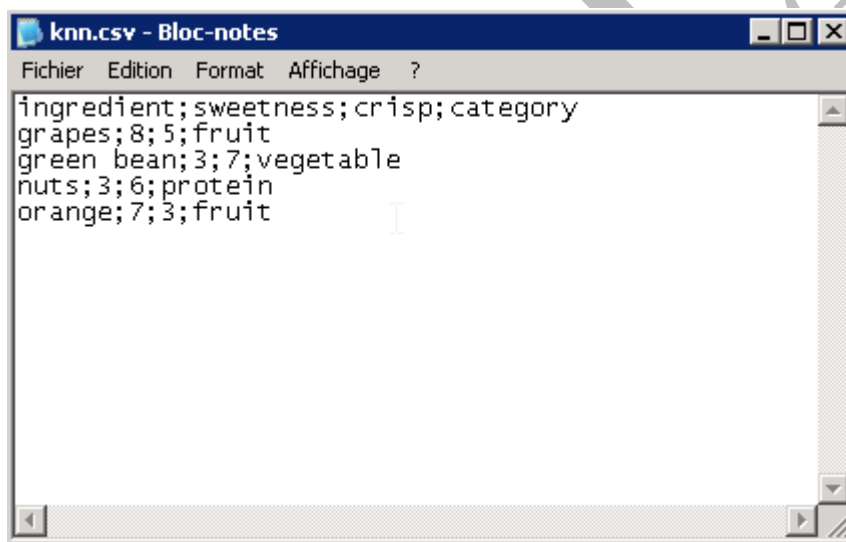
ans =

    11     1
     3     9

fx >> |
```

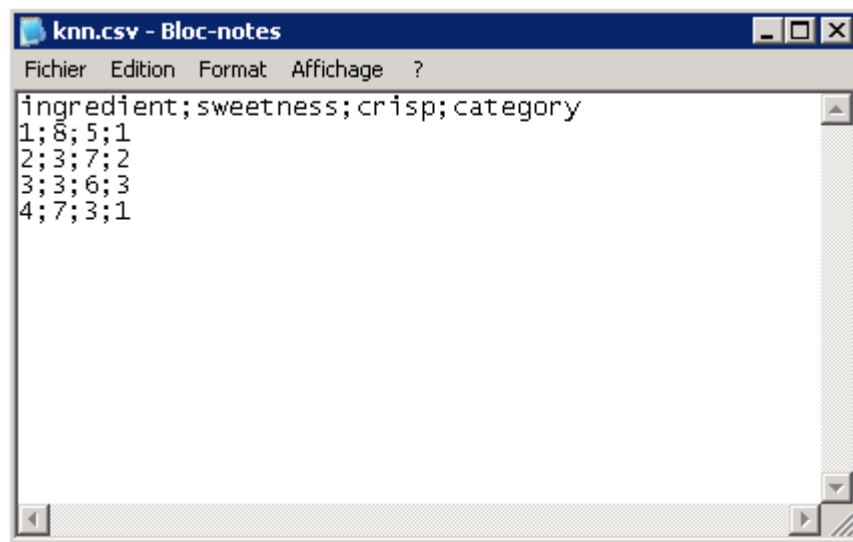
k-NN (k-Nearest Neighbors)

K Nearest Neighbors is another Machine Learning technic we have study in the theoretical course. We will also use the same small dataset to control if we get the same results as the one calculated by hand:



```
knn.csv - Bloc-notes
Fichier Edition Format Affichage ?
ingredient;sweetness;crisp;category
grapes;8;5;fruit
green bean;3;7;vegetable
nuts;3;6;protein
orange;7;3;fruit
```

We must first convert all non-numeric values into numeric because MATLAB doesn't handle (actually...) string:



After we just need to call the function **knnclassify()** and we plot on the same chart the model vs the original (in circles):

```

EDITOR PUBLISH VIEW
%we will supposed that the content is already randomized
mydata=dataset('File','C:\tmp\knn.csv','Delimiter',';');
%we won't do a training and test set (see Naive Bayes example for this)
training_data=[mydata.sweetness mydata.crisp];

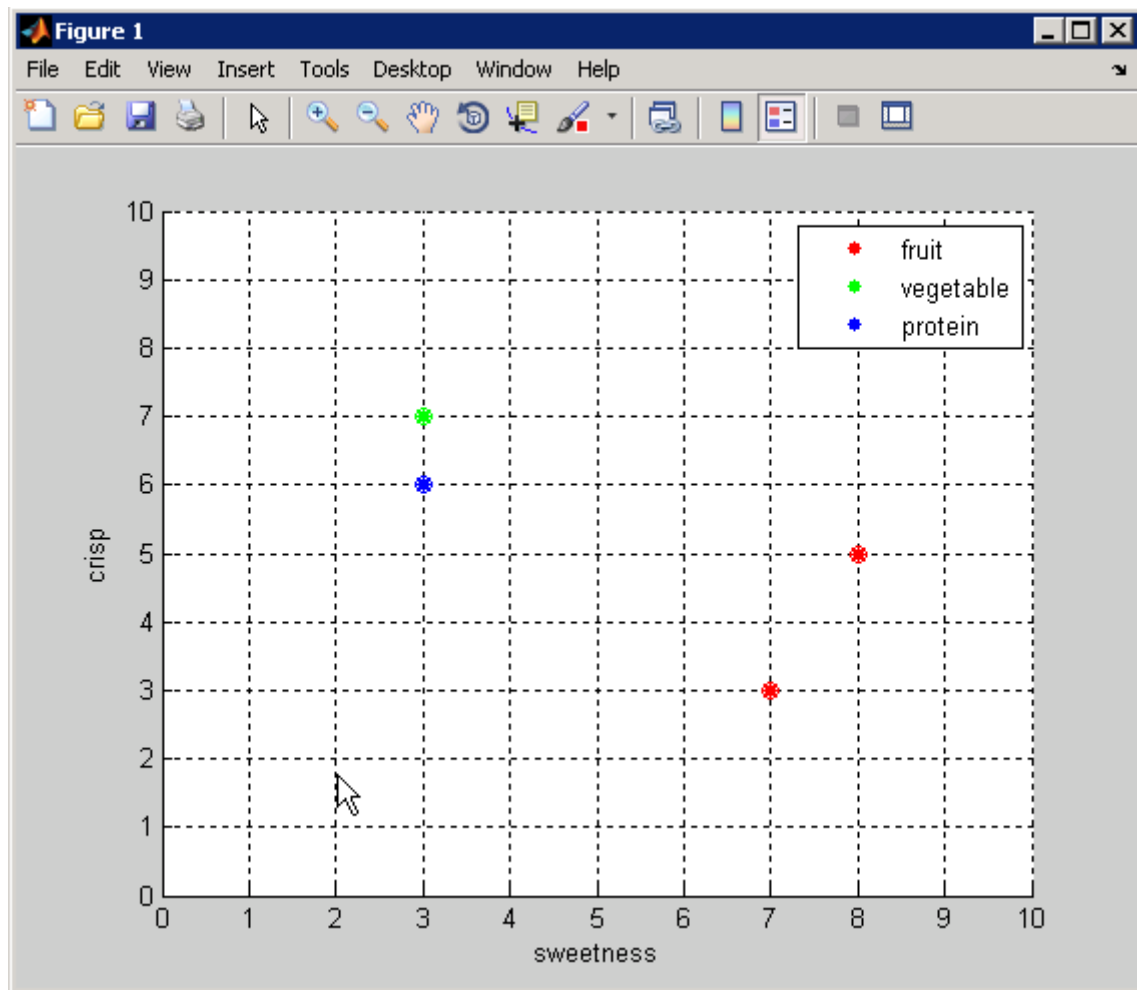
%with the sweetness 6 and crisp 0 classification?
number_nn=1;
knnclassify([6 0],training_data,mydata.category,number_nn)

%and with all data themselves?
knn=knnclassify(training_data,training_data,mydata.category,number_nn);
comparison=[mydata.category knn]

%and a plot
gscatter(training_data(:,1),training_data(:,2),mydata.category);
xlim([0 10]);ylim([0 10]);

hold on;grid on;
gscatter(training_data(:,1),training_data(:,2),knn,'rgb','o');
legend('fruit','vegetable','protein');
xlabel('sweetness');ylabel('crisp');

```



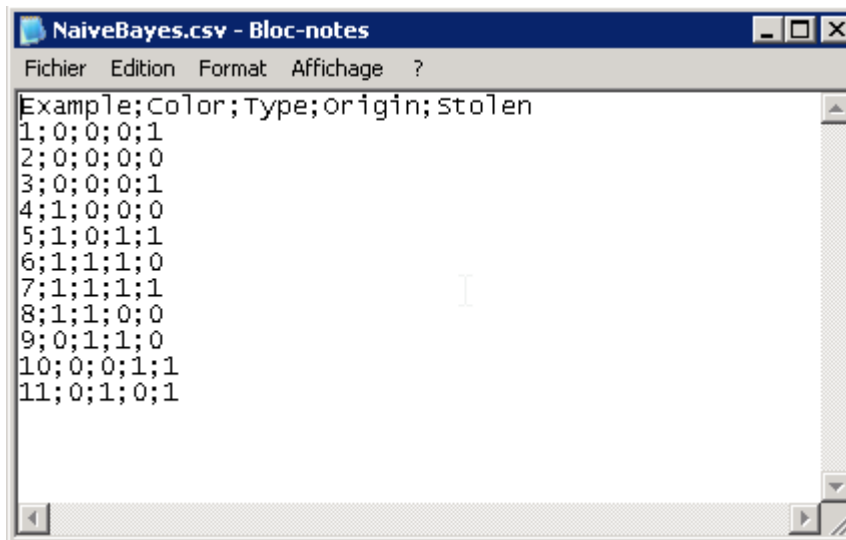
As we can see the prediction model is perfect in this case.

Naive Bayes

Now have a look to the naive Bayes machine learning algorithm to check also if we get the same results as in the theoretical course. For this we will use the same data:

```
NaiveBayes.csv - Bloc-notes
Fichier  Edition  Format  Affichage  ?
Example;Color;Type;Origin;Stolen
1;Red;Sports;Domestic;Yes
2;Red;Sports;Domestic;No
3;Red;Sports;Domestic;Yes
4;Yellow;Sports;Domestic;No
5;Yellow;Sports;Imported;Yes
6;Yellow;SUV;Imported;No
7;Yellow;SUV;Imported;Yes
8;Yellow;SUV;Domestic;No
9;Red;SUV;Imported;No
10;Red;Sports;Imported;Yes
11;Red;SUV;Domestic;Yes
```

We must convert into numeric because MATLAB doesn't handle (actually...) strings:



Now we use the **NaiveBayes()** function after importing, randomizing and sampling the original data:

```

1  mydata=dataset('File','C:\tmp\NaiveBayes.csv','Delimiter',';');
2  %we randomize the sample
3  ord=randperm(size(mydata.Example,1));
4  Color=mydata.Color(ord,:);
5  Type=mydata.Type(ord,:);
6  Origin=mydata.Origin(ord,:);
7  Stolen=mydata.Stolen(ord,:);
8
9  %we split into training/testing
10 data=[ord' Stolen, Color, Type, Origin]
11 training_set=data(:,2:5); %we take everything even if it's stupid
12 testing_set=data(:,2:5); %we also take everything even if it's stupid
13
14 %train model using Multinomial model because there is a row with only '0'!
15 naivB=NaiveBayes.fit(training_set,Stolen,'Distribution','mn')
16
17 %prediction
18 Prediction_Stolen=naivB.predict(testing_set);
19
20 %comparison model/reality
21 [Stolen Prediction_Stolen]
22 %confusion matrix
23 confusionmat(Stolen,Prediction_Stolen)

```

We use after the property **predict** and the function **confusionmat()** to see how performs the algorithm:

Command Window

```
data =
```

```
     6     0     1     1     1
     2     0     0     0     0
    10     1     0     0     1
     3     1     0     0     0
     9     0     0     1     1
    11     1     0     1     0
     4     0     1     0     0
     8     0     1     1     0
     1     1     0     0     0
     7     1     1     1     1
     5     1     1     0     1
```

```
naivB =
```

```
Naive Bayes classifier with 2 classes for 4 dimensions.  
Feature Distribution(s):mn  
Classes:0, 1
```

```
ans =
```

```
     0     0
     0     1
     1     1
     1     1
     0     0
     1     1
     0     0
     0     0
     1     1
     1     1
     1     1
```

```
ans =
```

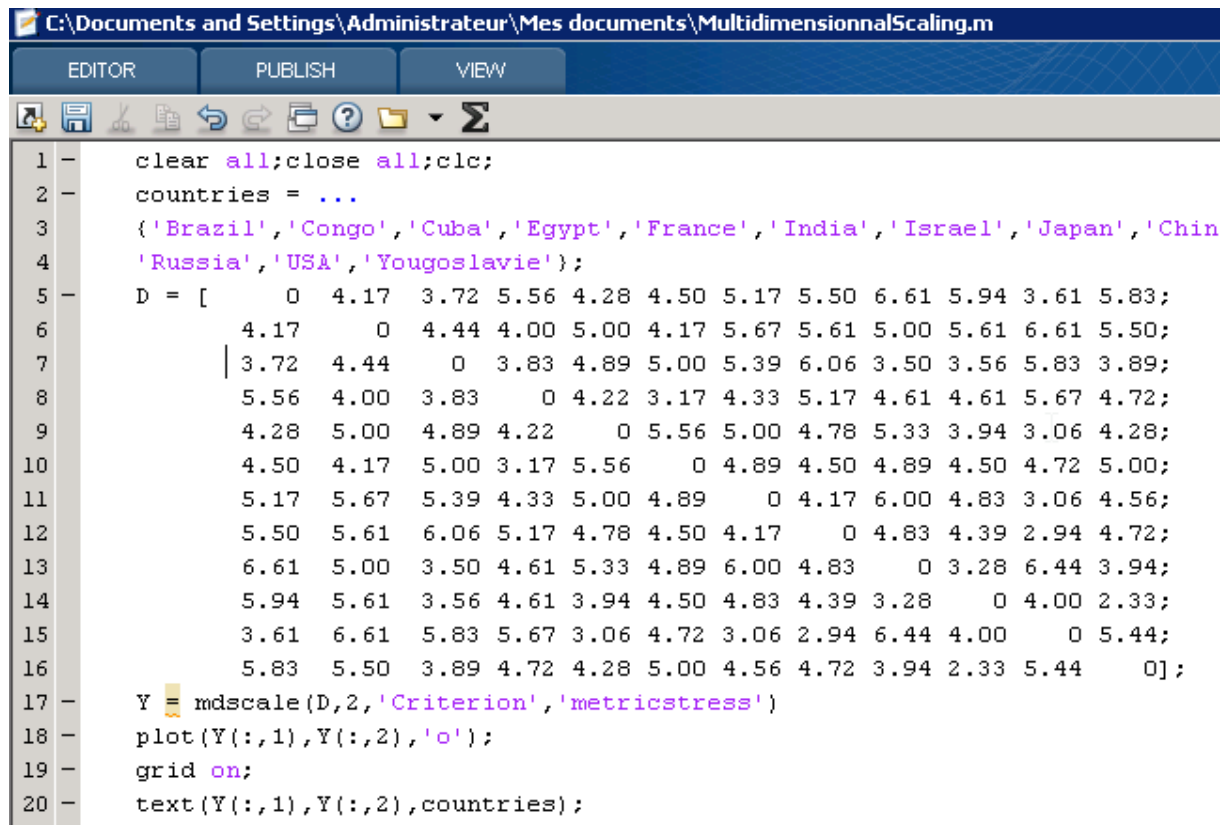
```
     4     1
     0     6
```

We can't compare with the theoretical course because of the sampling but it is difficult to imagine that anything can be false...

Non classical multidimensional scaling

As always the purpose here is to see if we get the same results as in theoretical course and also compare it to the results obtained with R and Microsoft Excel.

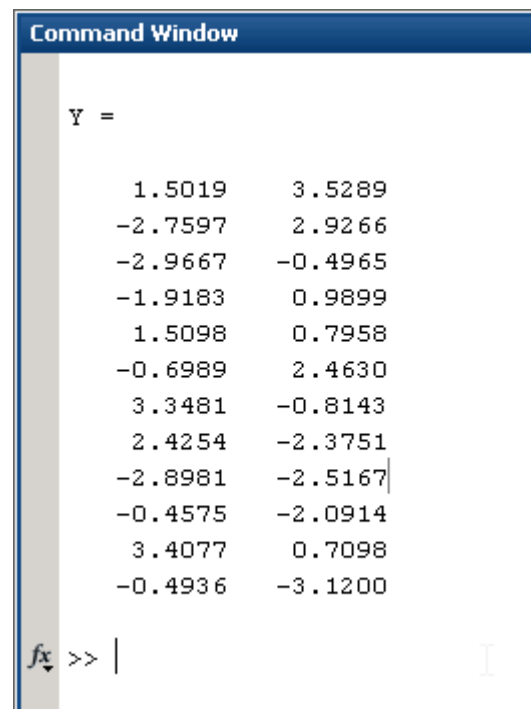
For this purpose we will use the same dissimilarity matrix as in the theoretical course with the function `mdscale()`:



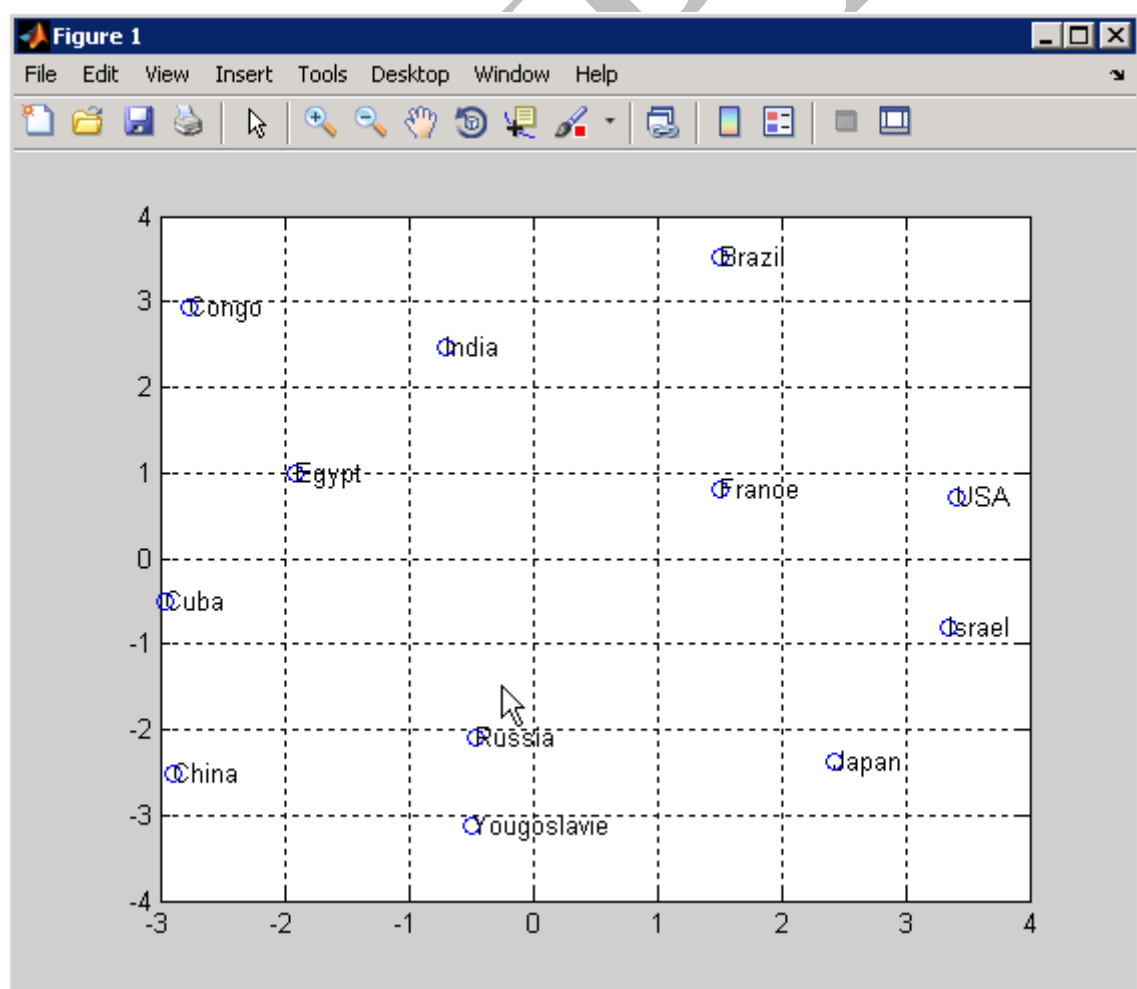
```

C:\Documents and Settings\Administrateur\Mes documents\MultidimensionnalScaling.m
EDITOR PUBLISH VIEW
1 - clear all;close all;clc;
2 - countries = ...
3   {'Brazil','Congo','Cuba','Egypt','France','India','Israel','Japan','Chin
4   'Russia','USA','Yougoslavie'};
5 - D = [
6         0  4.17  3.72  5.56  4.28  4.50  5.17  5.50  6.61  5.94  3.61  5.83;
7         4.17  0  4.44  4.00  5.00  4.17  5.67  5.61  5.00  5.61  6.61  5.50;
8         3.72  4.44  0  3.83  4.89  5.00  5.39  6.06  3.50  3.56  5.83  3.89;
9         5.56  4.00  3.83  0  4.22  3.17  4.33  5.17  4.61  4.61  5.67  4.72;
10        4.28  5.00  4.89  4.22  0  5.56  5.00  4.78  5.33  3.94  3.06  4.28;
11        4.50  4.17  5.00  3.17  5.56  0  4.89  4.50  4.89  4.50  4.72  5.00;
12        5.17  5.67  5.39  4.33  5.00  4.89  0  4.17  6.00  4.83  3.06  4.56;
13        5.50  5.61  6.06  5.17  4.78  4.50  4.17  0  4.83  4.39  2.94  4.72;
14        6.61  5.00  3.50  4.61  5.33  4.89  6.00  4.83  0  3.28  6.44  3.94;
15        5.94  5.61  3.56  4.61  3.94  4.50  4.83  4.39  3.28  0  4.00  2.33;
16        3.61  6.61  5.83  5.67  3.06  4.72  3.06  2.94  6.44  4.00  0  5.44;
17        5.83  5.50  3.89  4.72  4.28  5.00  4.56  4.72  3.94  2.33  5.44  0];
18 - Y = mdscale(D,2,'Criterion','metricstress')
19 - plot(Y(:,1),Y(:,2),'o');
20 - grid on;
21 - text(Y(:,1),Y(:,2),countries);
  
```

And we get:



And:

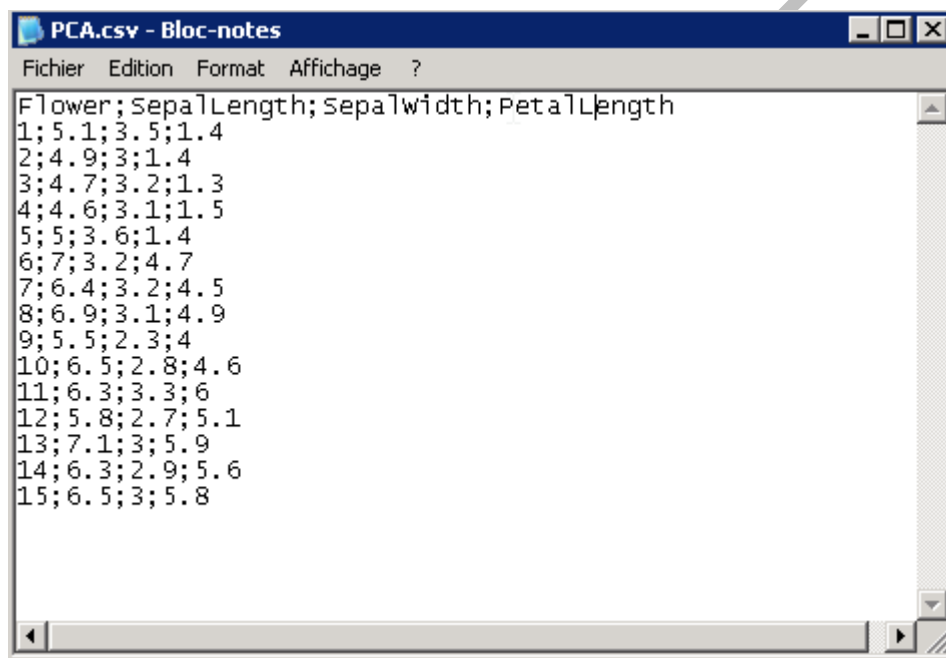


Thus the results are near for the third method used in the R training and far away from the calculations made with Microsoft Excel. But the interpretation of clusters still remains the same.

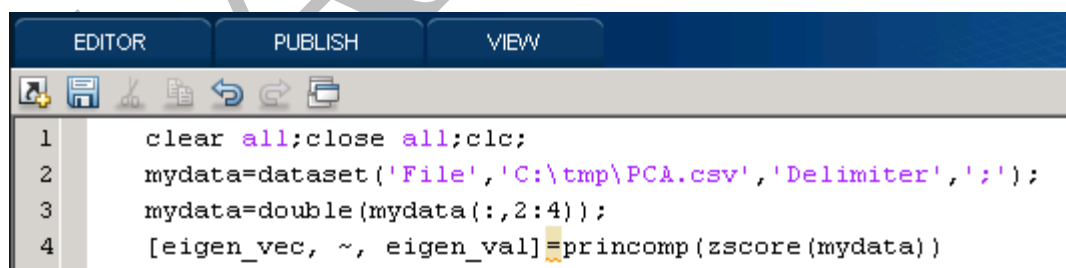
Parametric Principal Component Analysis

As we saw it in the theoretical statistic course, PCA is a very important tool in a lot of different domains. The proof of the parametric model was very long but the results powerful.

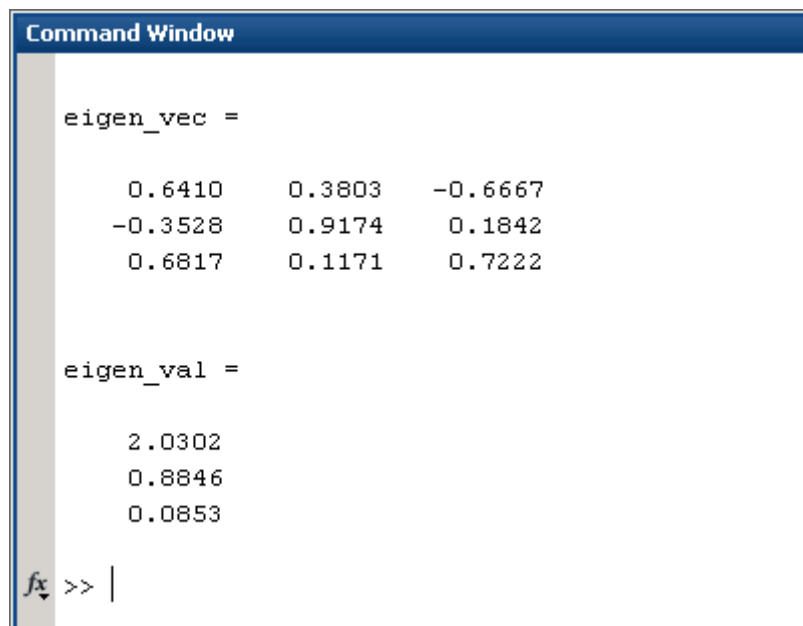
In the MATLAB exercise we will as always use exactly the same data set as the one used in the theoretical course. That is: a subset of the Fisher Iris set:



After we just need to import the data and use the function **princomp()**:



We get exactly the same eigen vectors and eigen values as the calculations made by hand:



```
Command Window

eigen_vec =

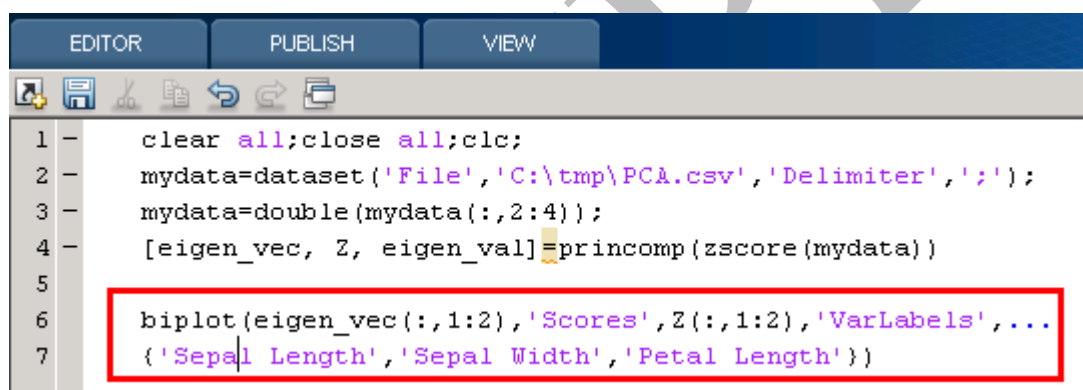
    0.6410    0.3803   -0.6667
   -0.3528    0.9174    0.1842
    0.6817    0.1171    0.7222

eigen_val =

    2.0302
    0.8846
    0.0853

fx >> |
```

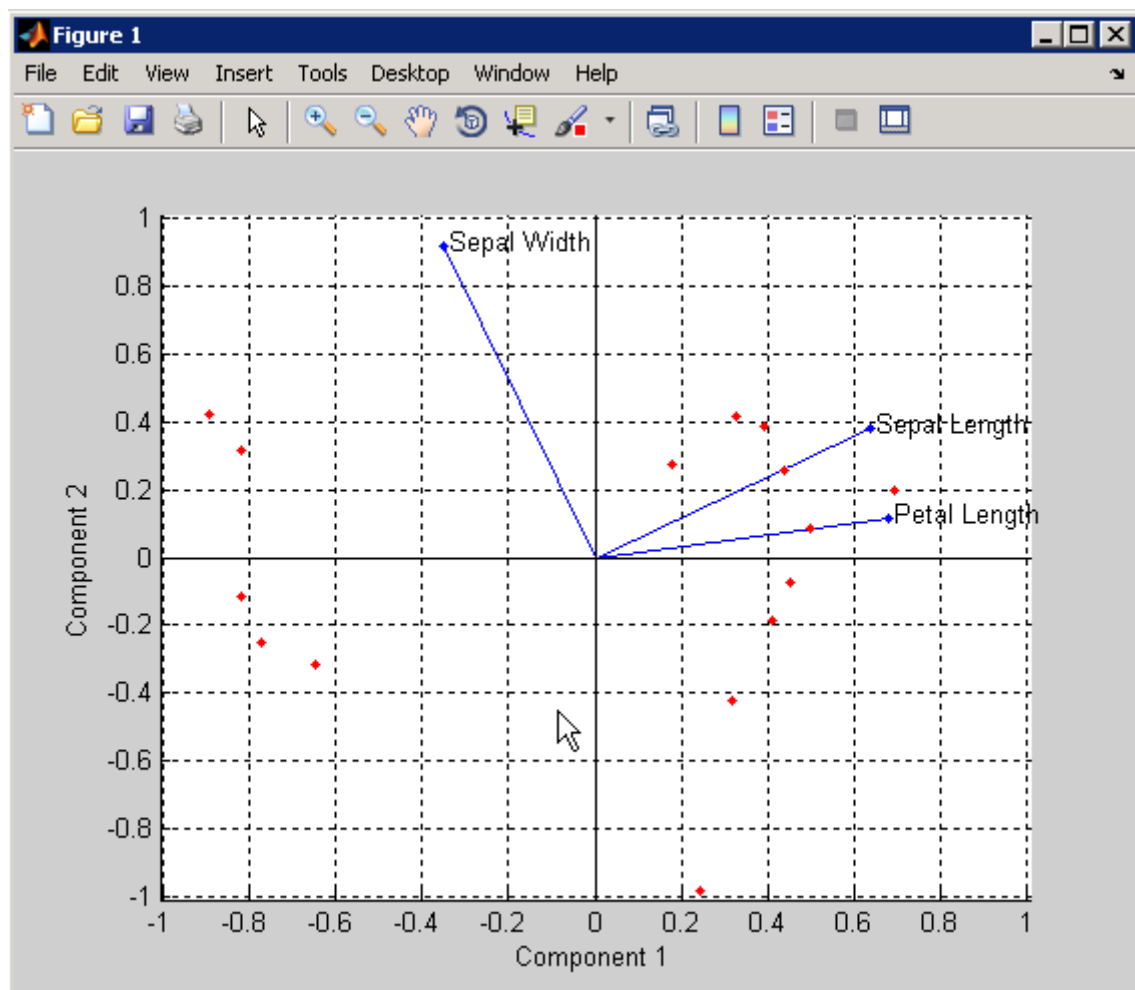
We can also plot the famous biplot using the **biplot()** function:



```
EDITOR PUBLISH VIEW

1 - clear all;close all;clc;
2 - mydata=dataset('File','C:\tmp\PCA.csv','Delimiter',';');
3 - mydata=double(mydata(:,2:4));
4 - [eigen_vec, Z, eigen_val]=princomp(zscore(mydata))
5
6 - biplot(eigen_vec(:,1:2),'Scores',Z(:,1:2),'VarLabels',...
7 - {'Sepal Length','Sepal Width','Petal Length'})
```

We get the following factorial plan that has the same scatter point structure as in the theoretical course. But the values of the scatter points are not the same as the one of Minitab and R or even as the one calculated by hand but the vectors are corresponding to the one we calculated by hand (but not corresponding with Minitab and R):



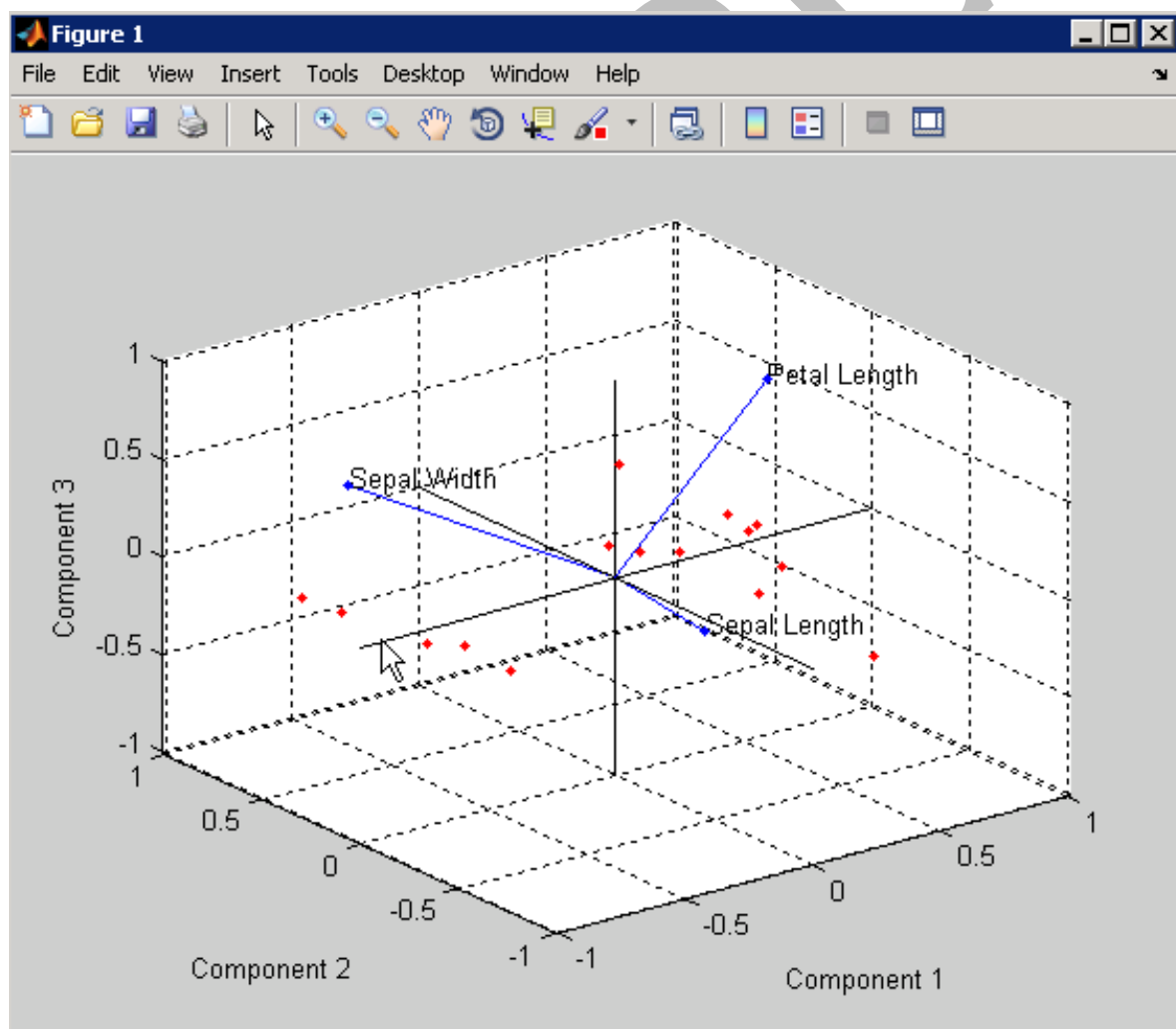
And if you want all the stuff in 3D:

```

1 - clear all;close all;clc;
2 - mydata=dataset('File','C:\tmp\PCA.csv','Delimiter',';');
3 - mydata=double(mydata(:,2:4));
4 - [eigen_vec, Z, eigen_val]=princomp(zscore(mydata))
5
6 - biplot(eigen_vec(:,1:2),'Scores',Z(:,1:2),'VarLabels',...
7 - {'Sepal Length','Sepal Width','Petal Length'})
8
9 - biplot(eigen_vec(:,1:3),'Scores',Z(:,1:3),'VarLabels',...
10 - {'Sepal Length','Sepal Width','Petal Length'})

```

We get:



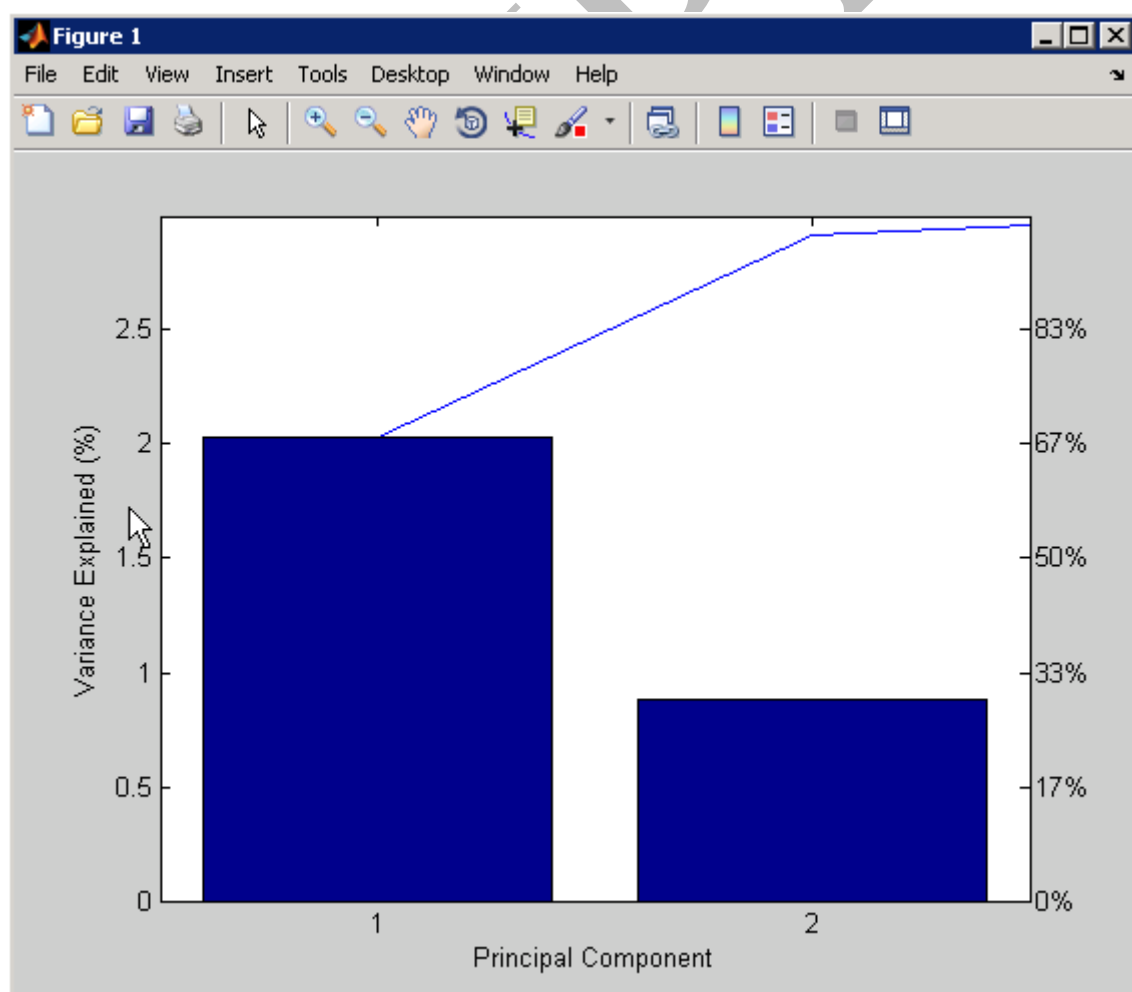
We can also add the famous Pareto eigenvalues ponderation:

```

1 - clear all;close all;clc;
2 - mydata=dataset('File','C:\tmp\PCA.csv','Delimiter',';');
3 - mydata=double(mydata(:,2:4));
4 - [eigen_vec, Z, eigen_val]=princomp(zscore(mydata))
5
6 - biplot(eigen_vec(:,1:2),'Scores',Z(:,1:2),'VarLabels',...
7         {'Sepal Length','Sepal Width','Petal Length'})
8
9 - biplot(eigen_vec(:,1:3),'Scores',Z(:,1:3),'VarLabels',...
10        {'Sepal Length','Sepal Width','Petal Length'})
11
12 - pareto(eigen_val);
13 - xlabel('Principal Component');
14 - ylabel('Variance Explained (%)');

```

So we get:

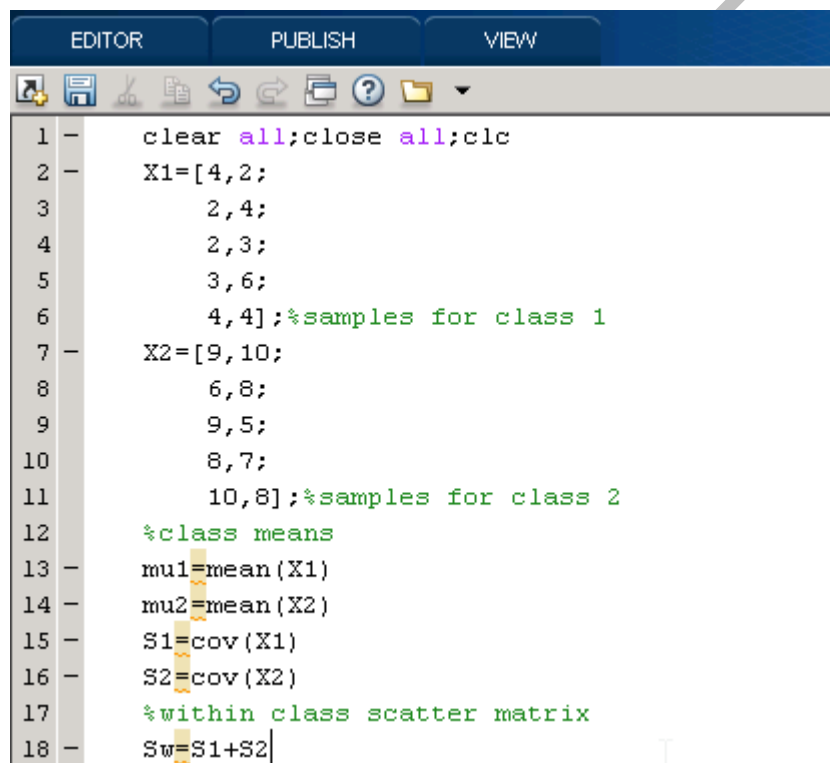


Linear Discriminant Analysis (Fisher Discriminant)

The purpose here will be to see how we can get with MATLAB the LDA result the we have proved in the data mining theoretical course using the Eigenvector Projection method (at the opposite of the Likelihood LDA method).

As far as I know there is no LDA integrated analysis in MATLAB 2013a using this method. This is why we will have to calculate everything ourself.

We type in our small data set values and make also the small mean, covariance and within class scatter matrix calculations:



```
1 - clear all;close all;clc
2 - X1=[4,2;
3     2,4;
4     2,3;
5     3,6;
6     4,4];%samples for class 1
7 - X2=[9,10;
8     6,8;
9     9,5;
10    8,7;
11    10,8];%samples for class 2
12 %class means
13 - mu1=mean(X1)
14 - mu2=mean(X2)
15 - S1=cov(X1)
16 - S2=cov(X2)
17 %within class scatter matrix
18 - Sw=S1+S2
```

This give us (see next page):

```

Command Window

mu1 =

    3.0000    3.8000

mu2 =

    8.4000    7.6000

S1 =

    1.0000   -0.2500
   -0.2500    2.2000

S2 =

    2.3000   -0.0500
   -0.0500    3.3000

fx >> |

```

This are the same results as those calculated by hand. After we add the part to calculate the LDA projection:

```

EDITOR PUBLISH VIEW
1 - clear all;close all;clc
2 - X1=[4,2;
3 -     2,4;
4 -     2,3;
5 -     3,6;
6 -     4,4];%samples for class 1
7 - X2=[9,10;
8 -     6,8;
9 -     9,5;
10 -    8,7;
11 -    10,8];%samples for class 2
12 %class means
13 - mu1=mean(X1);mu2=mean(X2);
14 - S1=cov(X1);S2=cov(X2);
15 %within class scatter matrix
16 - Sw=S1+S2
17 %between-class scatter matrix
18 - Sb=(mu1-mu2)'*(mu1-mu2)
19 %computing the LDA projection
20 - invSw=inv(Sw)
21 - invSw_by_Sb=invSw*Sb

```

This give us also the same results as those calculated by hand in the theoretical course:

```
Command Window

Sw =

    3.3000   -0.3000
   -0.3000    5.5000

Sb =

   29.1600   20.5200
   20.5200   14.4400

invSw =

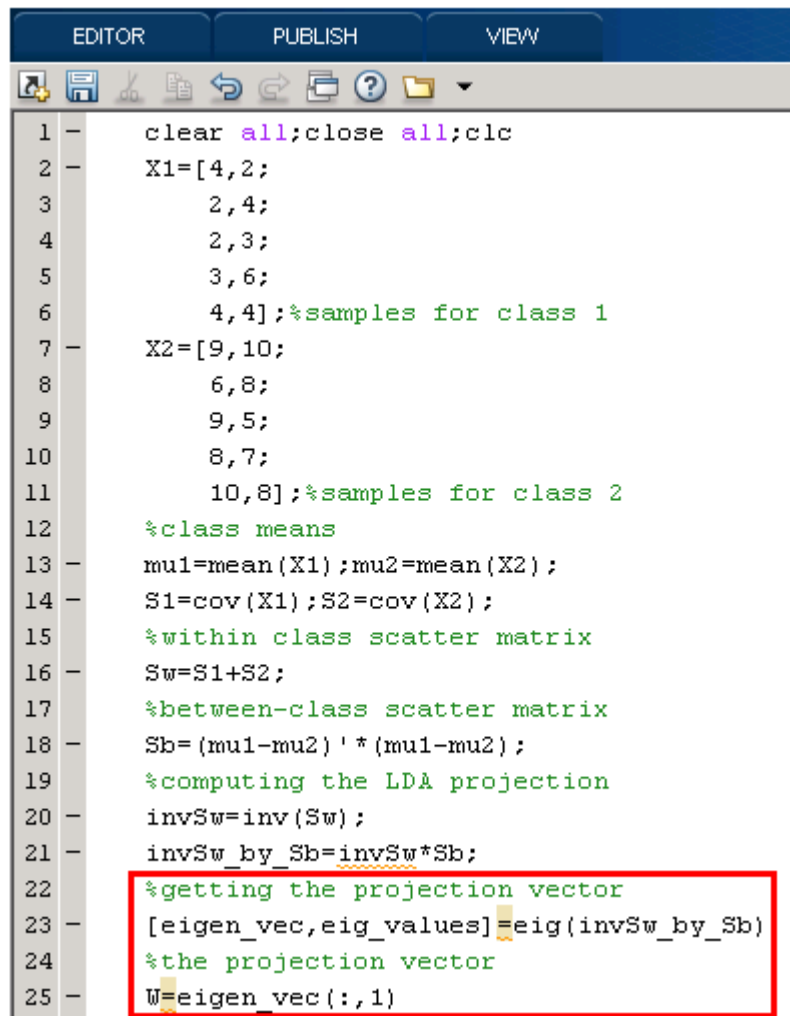
    0.3045    0.0166
    0.0166    0.1827

invSw_by_Sb =

    9.2213    6.4890
    4.2339    2.9794

fx >> |
```

And to finish we calculate the eigenvectors and eigenvalues and also the projection vector:



```

1 - clear all;close all;clc
2 - X1=[4,2;
3 -     2,4;
4 -     2,3;
5 -     3,6;
6 -     4,4];%samples for class 1
7 - X2=[9,10;
8 -     6,8;
9 -     9,5;
10 -    8,7;
11 -    10,8];%samples for class 2
12 %class means
13 - mu1=mean(X1);mu2=mean(X2);
14 - S1=cov(X1);S2=cov(X2);
15 %within class scatter matrix
16 - Sw=S1+S2;
17 %between-class scatter matrix
18 - Sb=(mu1-mu2)'*(mu1-mu2);
19 %computing the LDA projection
20 - invSw=inv(Sw);
21 - invSw_by_Sb=invSw*Sb;
22 %getting the projection vector
23 - [eigen_vec,eig_values]=eig(invSw_by_Sb)
24 %the projection vector
25 - W=eigen_vec(:,1)
  
```

This give us also the same results as those calculated by hand:

```

Command Window

eigen_vec =

    0.9088   -0.5755
    0.4173    0.8178

eig_values =

    12.2007         0
         0    0.0000

W =

    0.9088
    0.4173

fx >> |

```

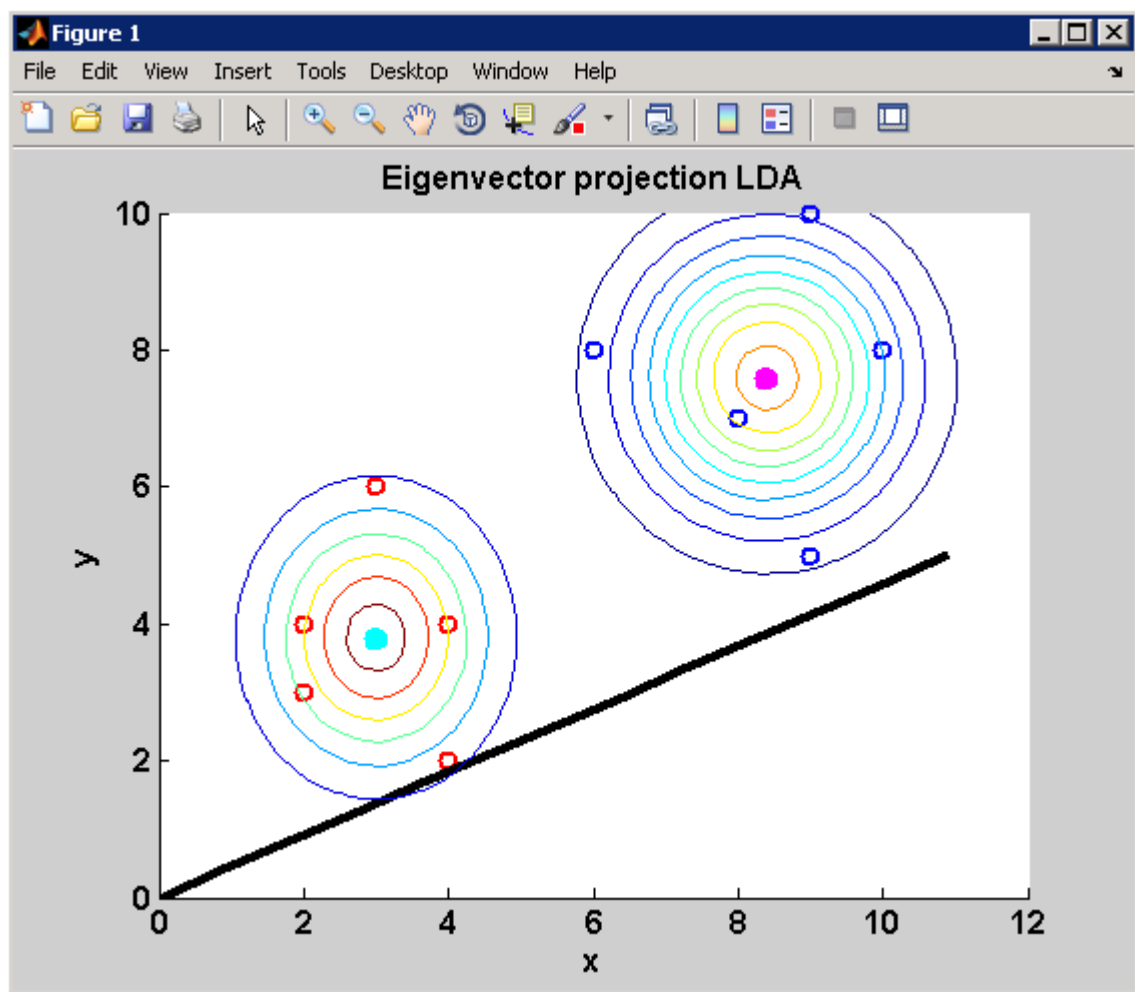
Finally we just need to plot:

```

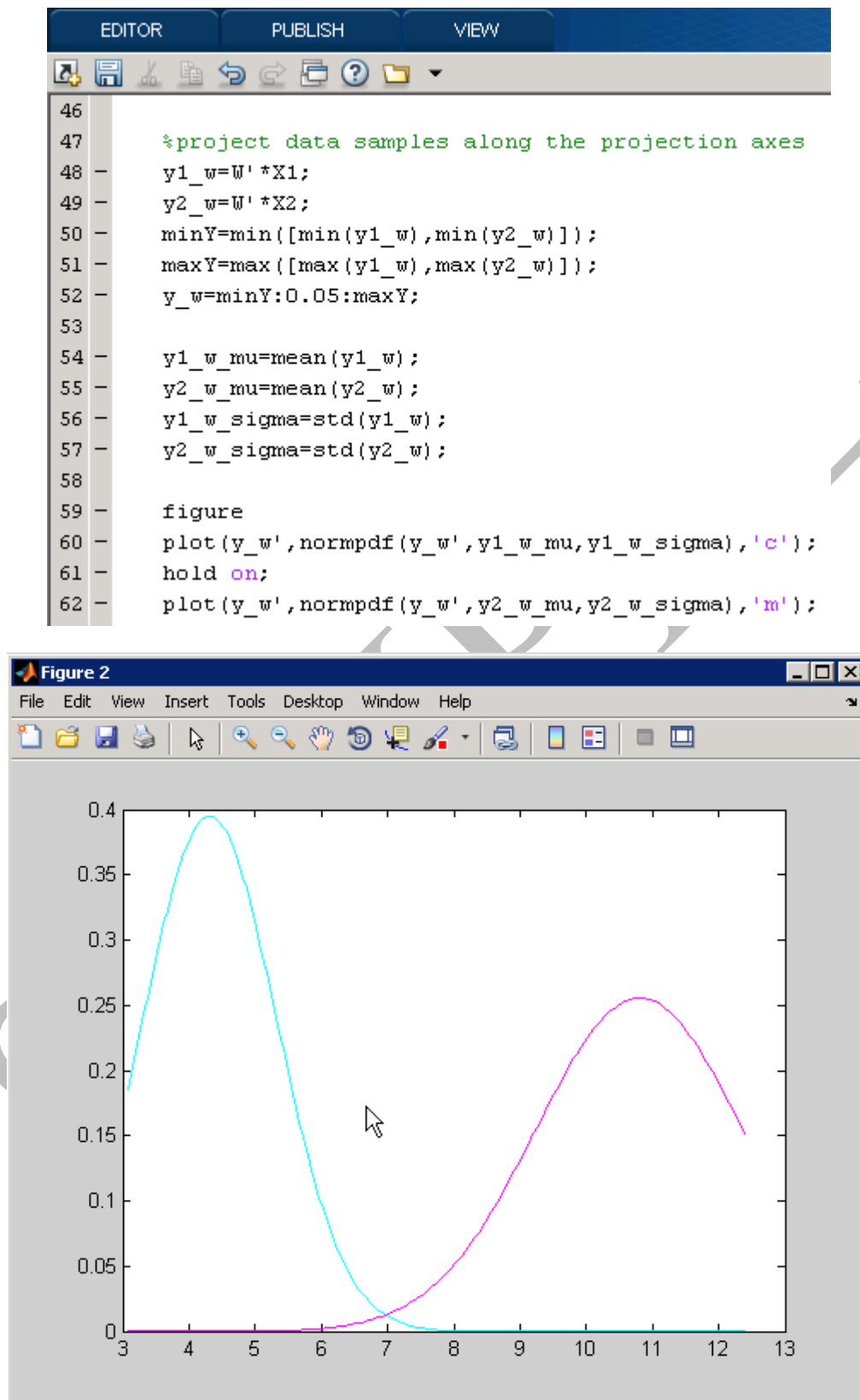
EDITOR PUBLISH VIEW
%we plot the stuff
26 - hfig=figure;
27 - X1=X1';X2=X2';
28 - axes1=axes('Parent',hfig,'Fontweight','bold','FontSize',12);|
29 - hold('all');
30 - xlabel('X_1');ylabel('X_2');
31 -
32 -
33 - scatter(X1(1,:),X1(2:),'r','LineWidth',2,'Parent',axes1);
34 - scatter(X2(1,:),X2(2:),'b','LineWidth',2,'Parent',axes1);
35 - plot(mean(X1(1,:)),mean(X1(2,:)),'co','MarkerSize',8,'MarkerEdgeColor','c',...
36 -      'Color','c','LineWidth',2,'MarkerFaceColor','c')
37 - plot(mean(X2(1,:)),mean(X2(2,:)),'mo','MarkerSize',8,'MarkerEdgeColor','m',...
38 -      'Color','m','LineWidth',2,'MarkerFaceColor','m')
39 - t=0:12;
40 - line_x=t.*W(1);
41 - line_y=t.*W(2);
42 - plot(line_x,line_y,'k-','LineWidth',3);
43 - ezcontour(@(x,y) mvnpdf([x y], mu1, [std(X1(1,:)) 0;0 std(X1(2,:))]), xlim(), ylim())
44 - ezcontour(@(x,y) mvnpdf([x y], mu2, [std(X2(1,:)) 0;0 std(X2(2,:))]), xlim(), ylim())
45 - title('Eigenvector projection LDA');

```

We get the famous LDA Gaussian density projection chart:



Or using another famous other representation we have:

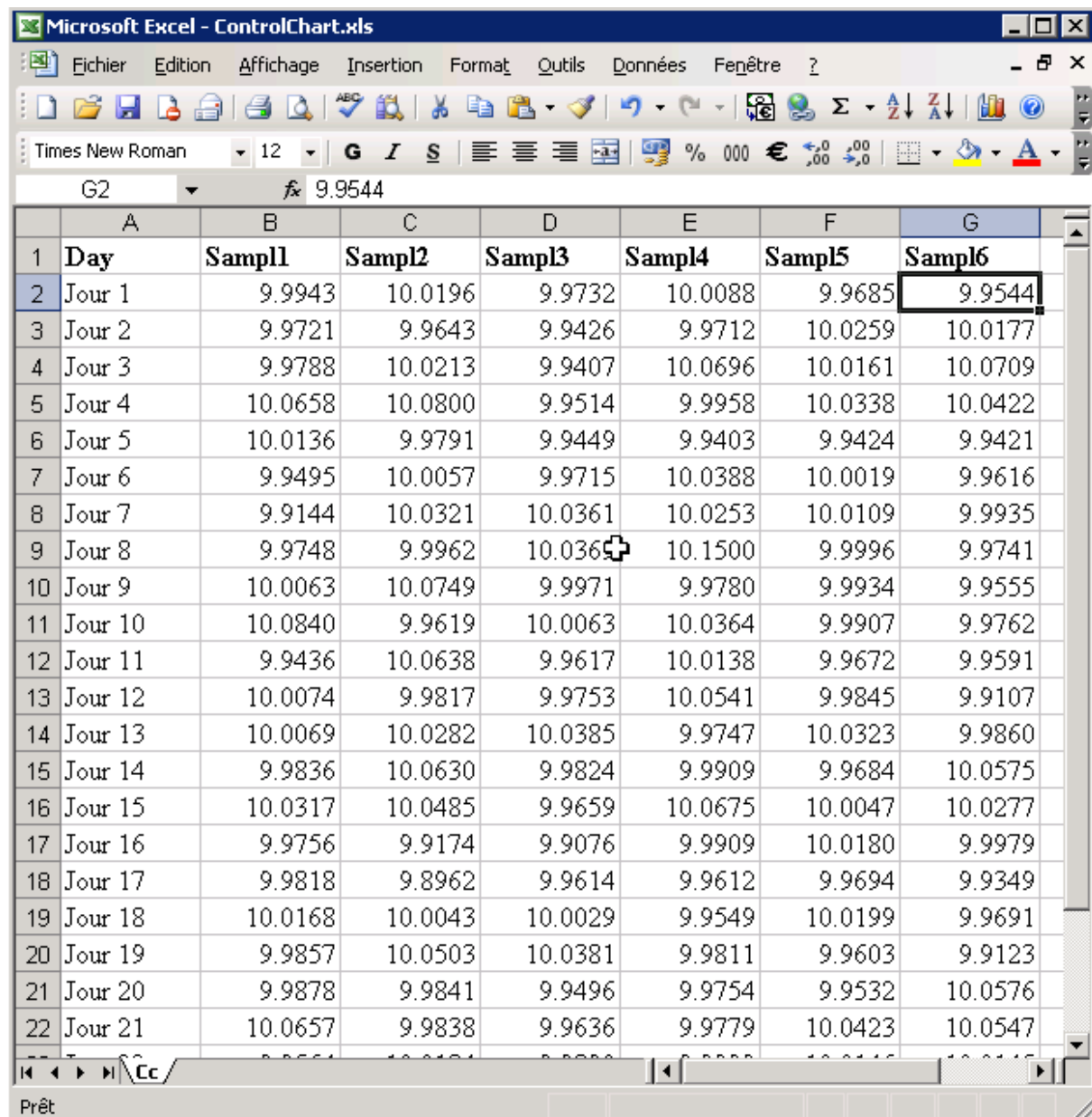


Everything corresponds for sure to calculation made by hand in the theoretical course.

Control charts (CC)

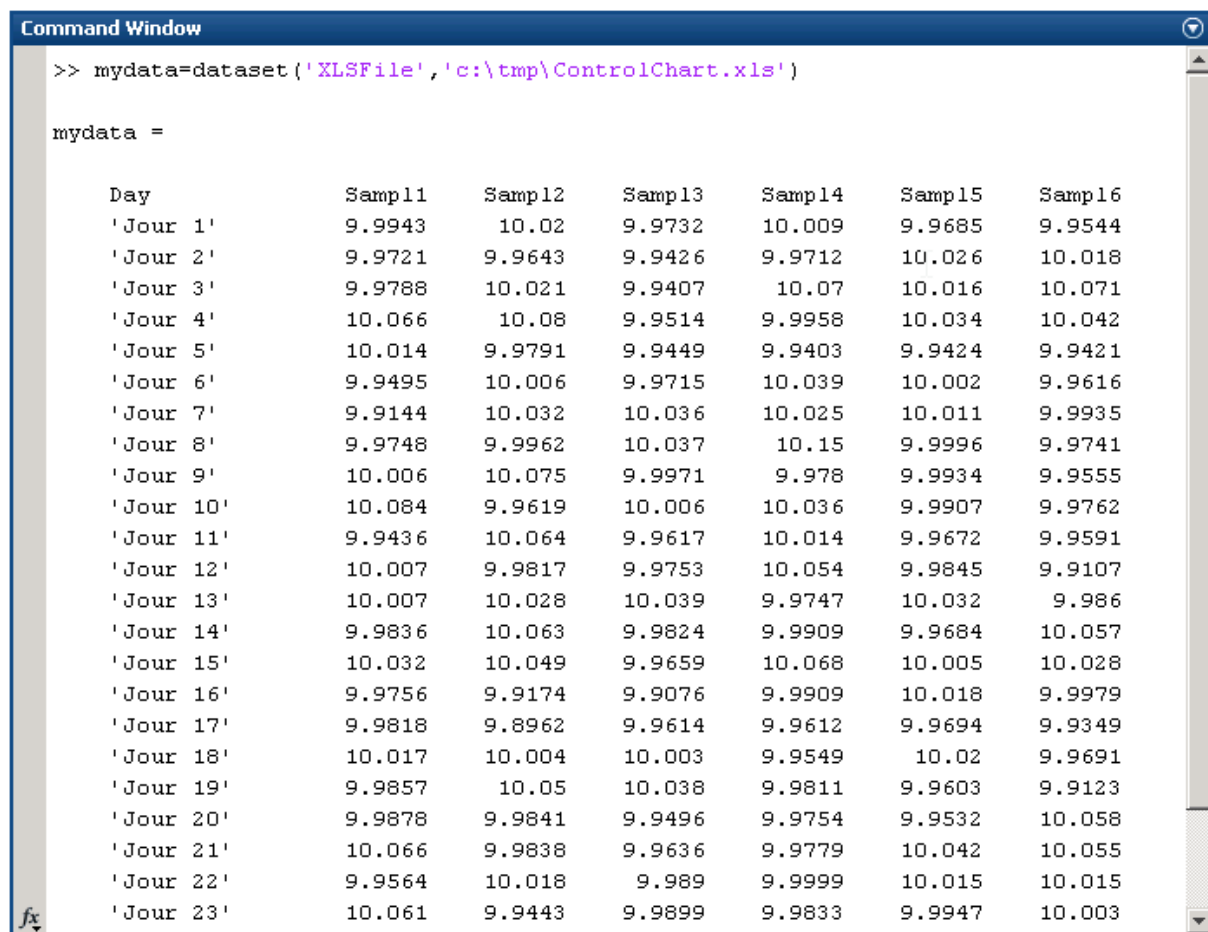
This is a big and important subject in quantitative quality control. MATLAB is not as flexible as Minitab but has at least the (very) minimum we can expect of an engineering software with a Statistical Toolbox.

To study the continuous control chart we will use the same data as in the theoretical course. This is to say:



	A	B	C	D	E	F	G
1	Day	Sampl1	Sampl2	Sampl3	Sampl4	Sampl5	Sampl6
2	Jour 1	9.9943	10.0196	9.9732	10.0088	9.9685	9.9544
3	Jour 2	9.9721	9.9643	9.9426	9.9712	10.0259	10.0177
4	Jour 3	9.9788	10.0213	9.9407	10.0696	10.0161	10.0709
5	Jour 4	10.0658	10.0800	9.9514	9.9958	10.0338	10.0422
6	Jour 5	10.0136	9.9791	9.9449	9.9403	9.9424	9.9421
7	Jour 6	9.9495	10.0057	9.9715	10.0388	10.0019	9.9616
8	Jour 7	9.9144	10.0321	10.0361	10.0253	10.0109	9.9935
9	Jour 8	9.9748	9.9962	10.0365	10.1500	9.9996	9.9741
10	Jour 9	10.0063	10.0749	9.9971	9.9780	9.9934	9.9555
11	Jour 10	10.0840	9.9619	10.0063	10.0364	9.9907	9.9762
12	Jour 11	9.9436	10.0638	9.9617	10.0138	9.9672	9.9591
13	Jour 12	10.0074	9.9817	9.9753	10.0541	9.9845	9.9107
14	Jour 13	10.0069	10.0282	10.0385	9.9747	10.0323	9.9860
15	Jour 14	9.9836	10.0630	9.9824	9.9909	9.9684	10.0575
16	Jour 15	10.0317	10.0485	9.9659	10.0675	10.0047	10.0277
17	Jour 16	9.9756	9.9174	9.9076	9.9909	10.0180	9.9979
18	Jour 17	9.9818	9.8962	9.9614	9.9612	9.9694	9.9349
19	Jour 18	10.0168	10.0043	10.0029	9.9549	10.0199	9.9691
20	Jour 19	9.9857	10.0503	10.0381	9.9811	9.9603	9.9123
21	Jour 20	9.9878	9.9841	9.9496	9.9754	9.9532	10.0576
22	Jour 21	10.0657	9.9838	9.9636	9.9779	10.0423	10.0547

We import the data as usual in MATLAB:



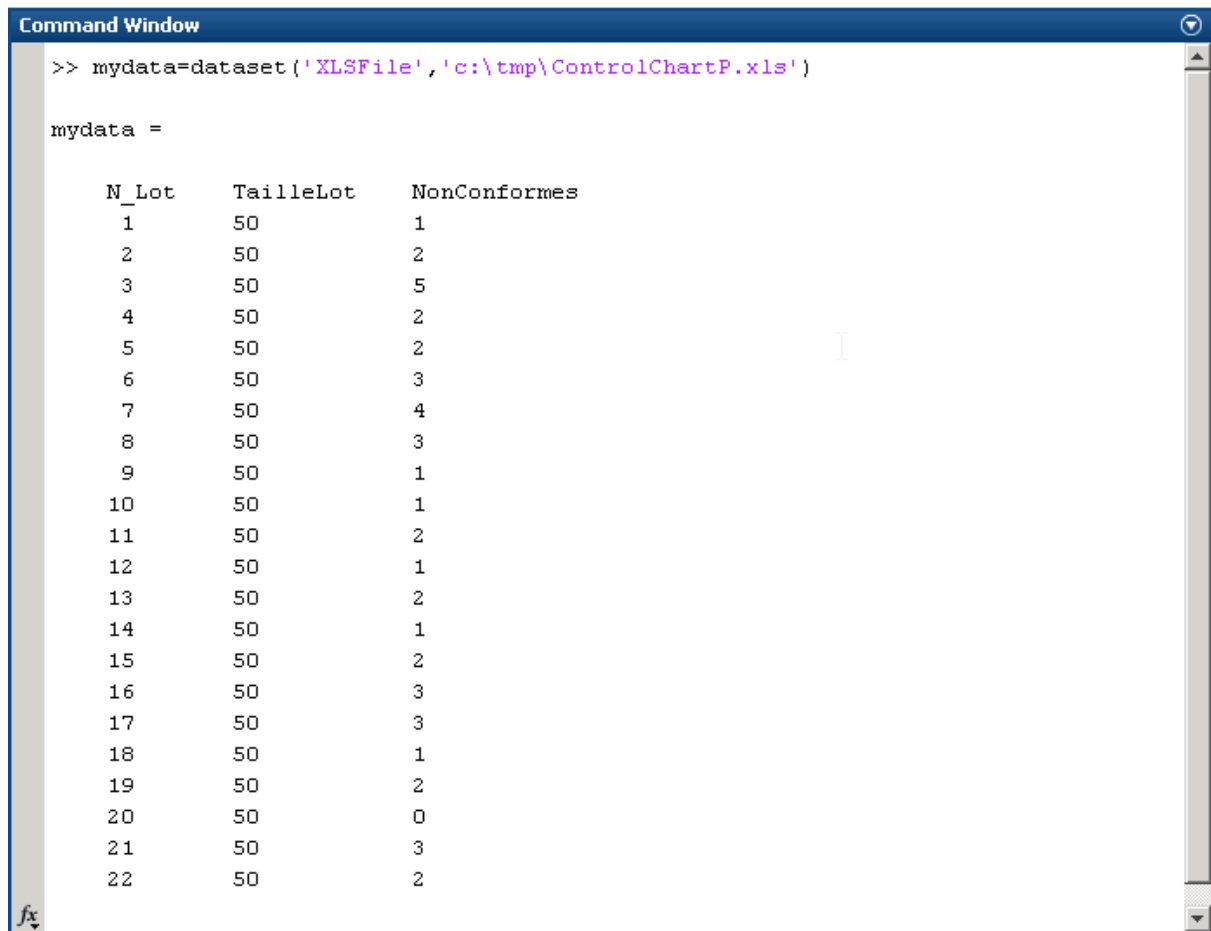
```
>> mydata=dataset('XLSFile','c:\tmp\ControlChart.xls')

mydata =
```

Day	Sampl1	Sampl2	Sampl3	Sampl4	Sampl5	Sampl6
'Jour 1'	9.9943	10.02	9.9732	10.009	9.9685	9.9544
'Jour 2'	9.9721	9.9643	9.9426	9.9712	10.026	10.018
'Jour 3'	9.9788	10.021	9.9407	10.07	10.016	10.071
'Jour 4'	10.066	10.08	9.9514	9.9958	10.034	10.042
'Jour 5'	10.014	9.9791	9.9449	9.9403	9.9424	9.9421
'Jour 6'	9.9495	10.006	9.9715	10.039	10.002	9.9616
'Jour 7'	9.9144	10.032	10.036	10.025	10.011	9.9935
'Jour 8'	9.9748	9.9962	10.037	10.15	9.9996	9.9741
'Jour 9'	10.006	10.075	9.9971	9.978	9.9934	9.9555
'Jour 10'	10.084	9.9619	10.006	10.036	9.9907	9.9762
'Jour 11'	9.9436	10.064	9.9617	10.014	9.9672	9.9591
'Jour 12'	10.007	9.9817	9.9753	10.054	9.9845	9.9107
'Jour 13'	10.007	10.028	10.039	9.9747	10.032	9.986
'Jour 14'	9.9836	10.063	9.9824	9.9909	9.9684	10.057
'Jour 15'	10.032	10.049	9.9659	10.068	10.005	10.028
'Jour 16'	9.9756	9.9174	9.9076	9.9909	10.018	9.9979
'Jour 17'	9.9818	9.8962	9.9614	9.9612	9.9694	9.9349
'Jour 18'	10.017	10.004	10.003	9.9549	10.02	9.9691
'Jour 19'	9.9857	10.05	10.038	9.9811	9.9603	9.9123
'Jour 20'	9.9878	9.9841	9.9496	9.9754	9.9532	10.058
'Jour 21'	10.066	9.9838	9.9636	9.9779	10.042	10.055
'Jour 22'	9.9564	10.018	9.989	9.9999	10.015	10.015
'Jour 23'	10.061	9.9443	9.9899	9.9833	9.9947	10.003

P attribute CC

The proportion control chart is one of the most easy to understand CC in the theoretical point of view (and also to read). We will also here use the same data as in the theoretical course to check if we get the same results:

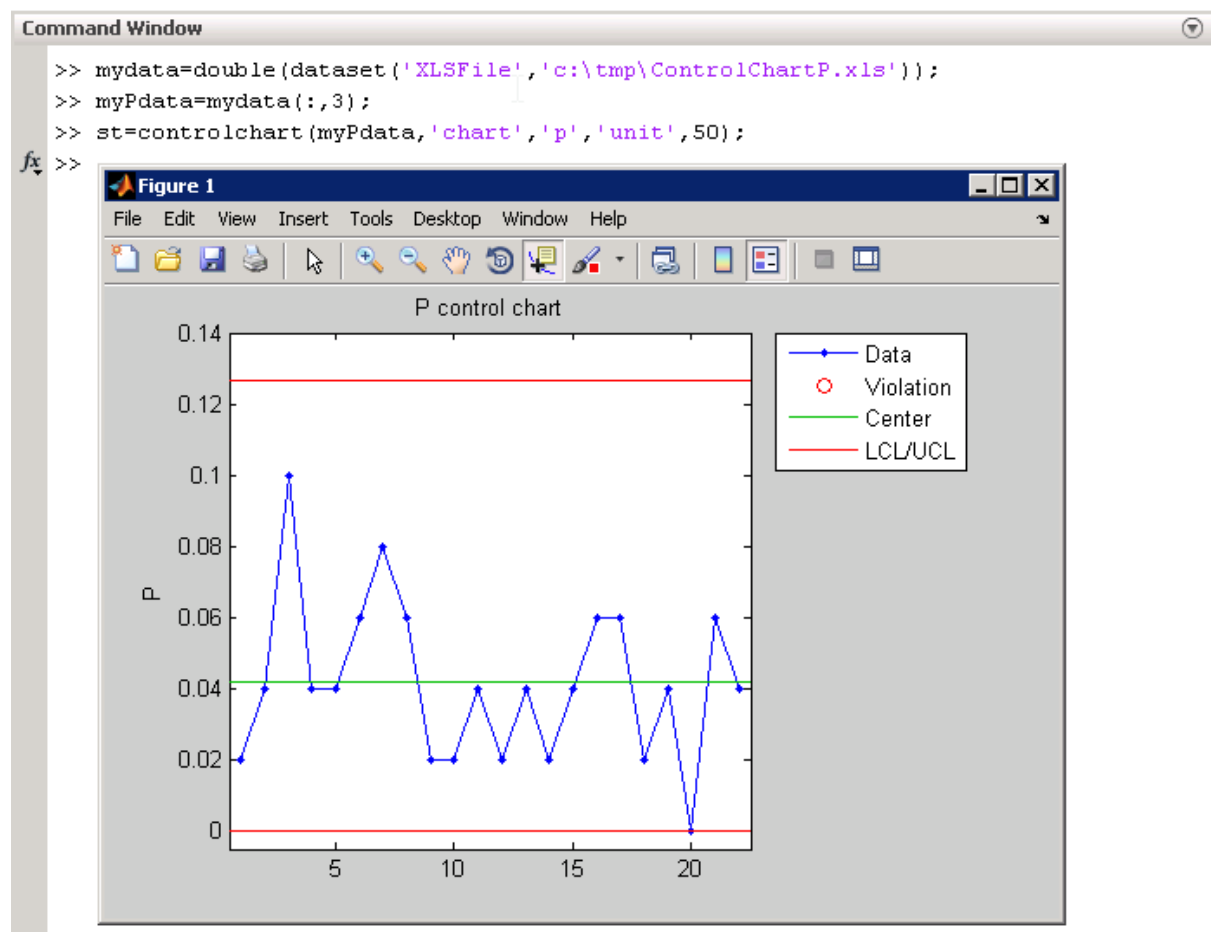
A screenshot of the MATLAB Command Window. The title bar says "Command Window". The command prompt shows the command: `>> mydata=dataset('XLSFile','c:\tmp\ControlChartP.xls')`. Below the command, the output shows the variable `mydata` as a table with three columns: `N_Lot`, `TailleLot`, and `NonConformes`. The data is displayed as a table with 22 rows. A large, faint "STAT" watermark is visible in the background of the image.

```
>> mydata=dataset('XLSFile','c:\tmp\ControlChartP.xls')

mydata =

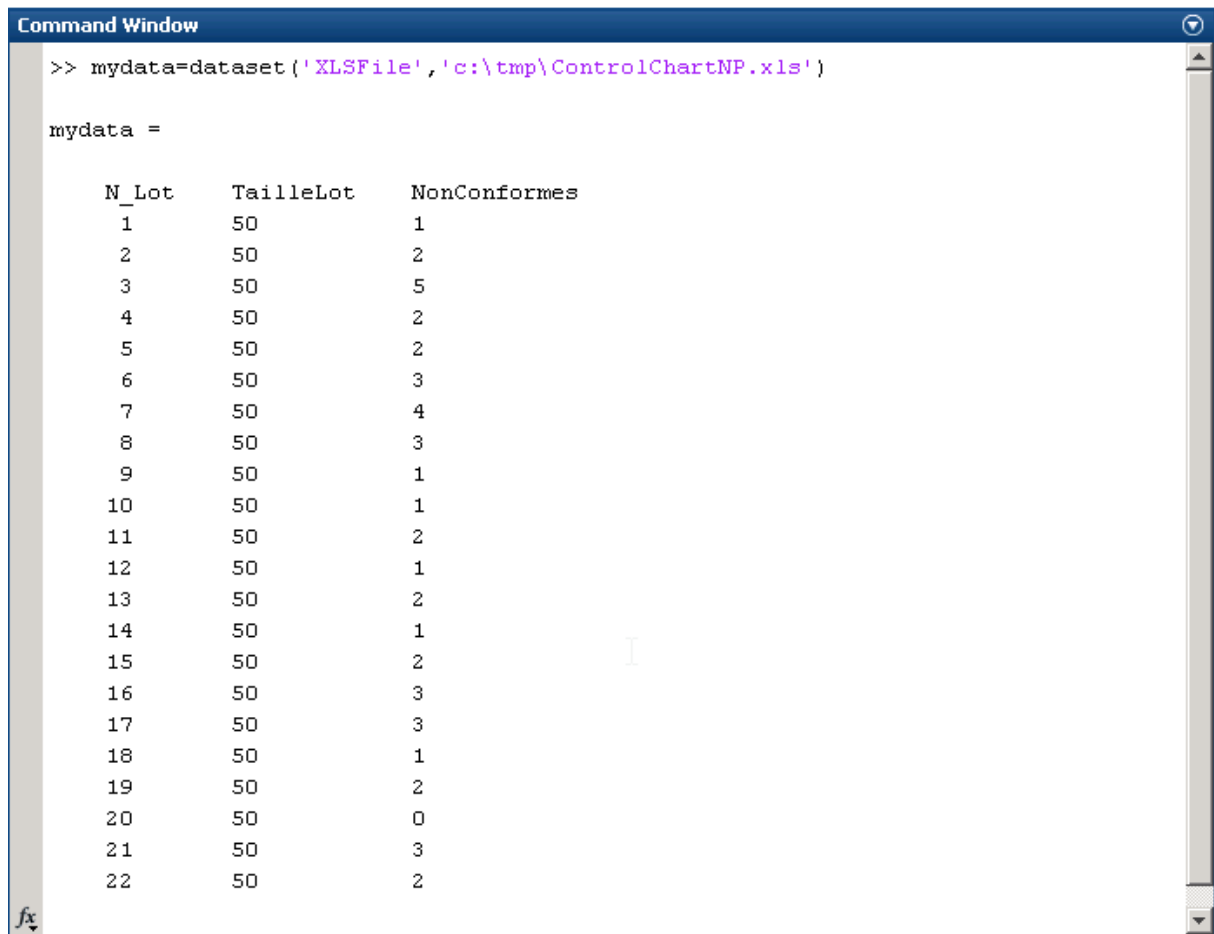
    N_Lot    TailleLot    NonConformes
    1         50          1
    2         50          2
    3         50          5
    4         50          2
    5         50          2
    6         50          3
    7         50          4
    8         50          3
    9         50          1
   10         50          1
   11         50          2
   12         50          1
   13         50          2
   14         50          1
   15         50          2
   16         50          3
   17         50          3
   18         50          1
   19         50          2
   20         50          0
   21         50          3
   22         50          2
```

Using once again (...) the function `controlchart()` we get:



Here also we get the same values as in the theoretical course except for the first three point.

NP attribute CC

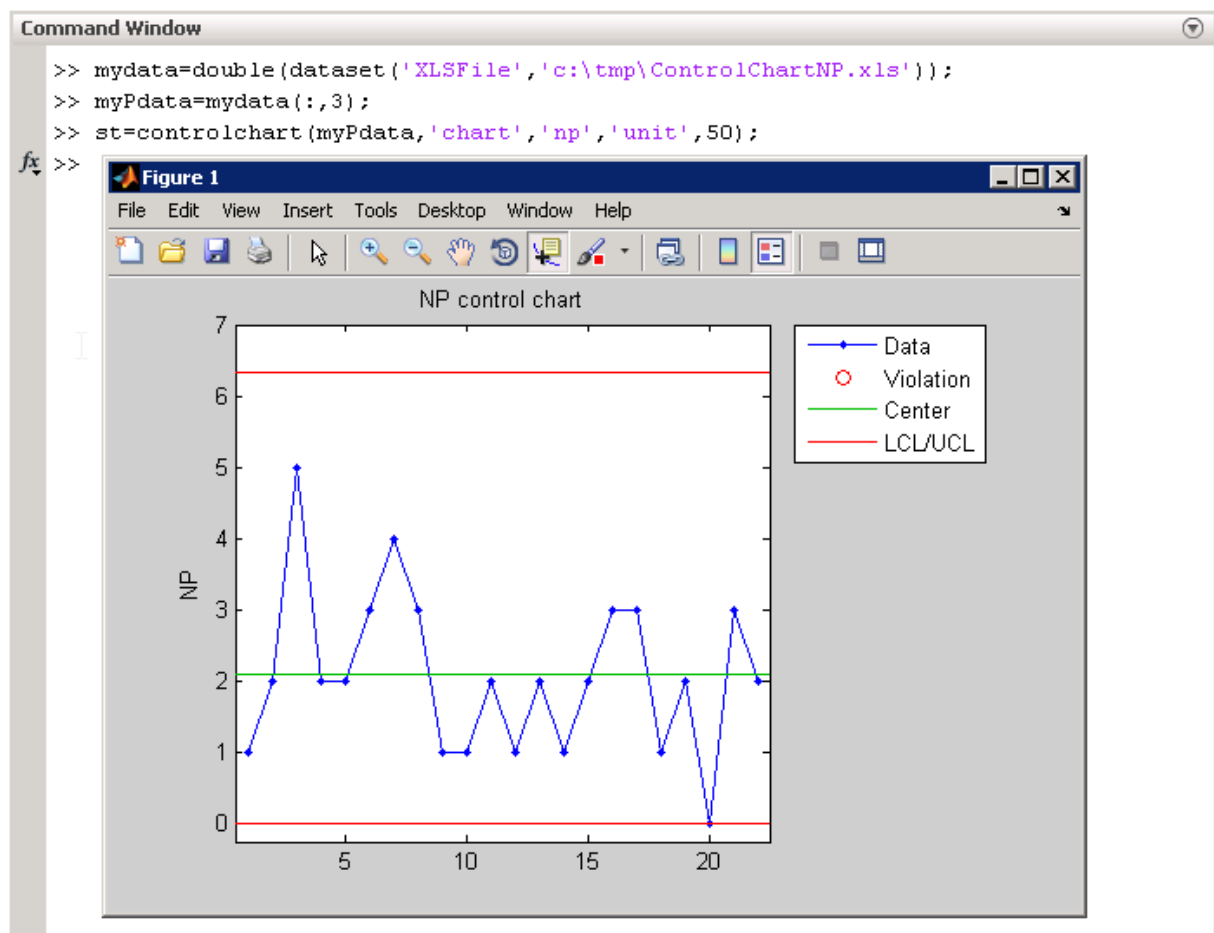


The image shows a MATLAB Command Window with a dark blue title bar. The command `>> mydata=dataset('XLSFile','c:\tmp\ControlChartNP.xls')` has been executed. Below the command, the output `mydata =` is followed by a table with three columns: `N_Lot`, `TailleLot`, and `NonConformes`. The table contains 22 rows of data. A large, light gray watermark with the word "SAVED" is oriented diagonally across the lower half of the image.

```
>> mydata=dataset('XLSFile','c:\tmp\ControlChartNP.xls')

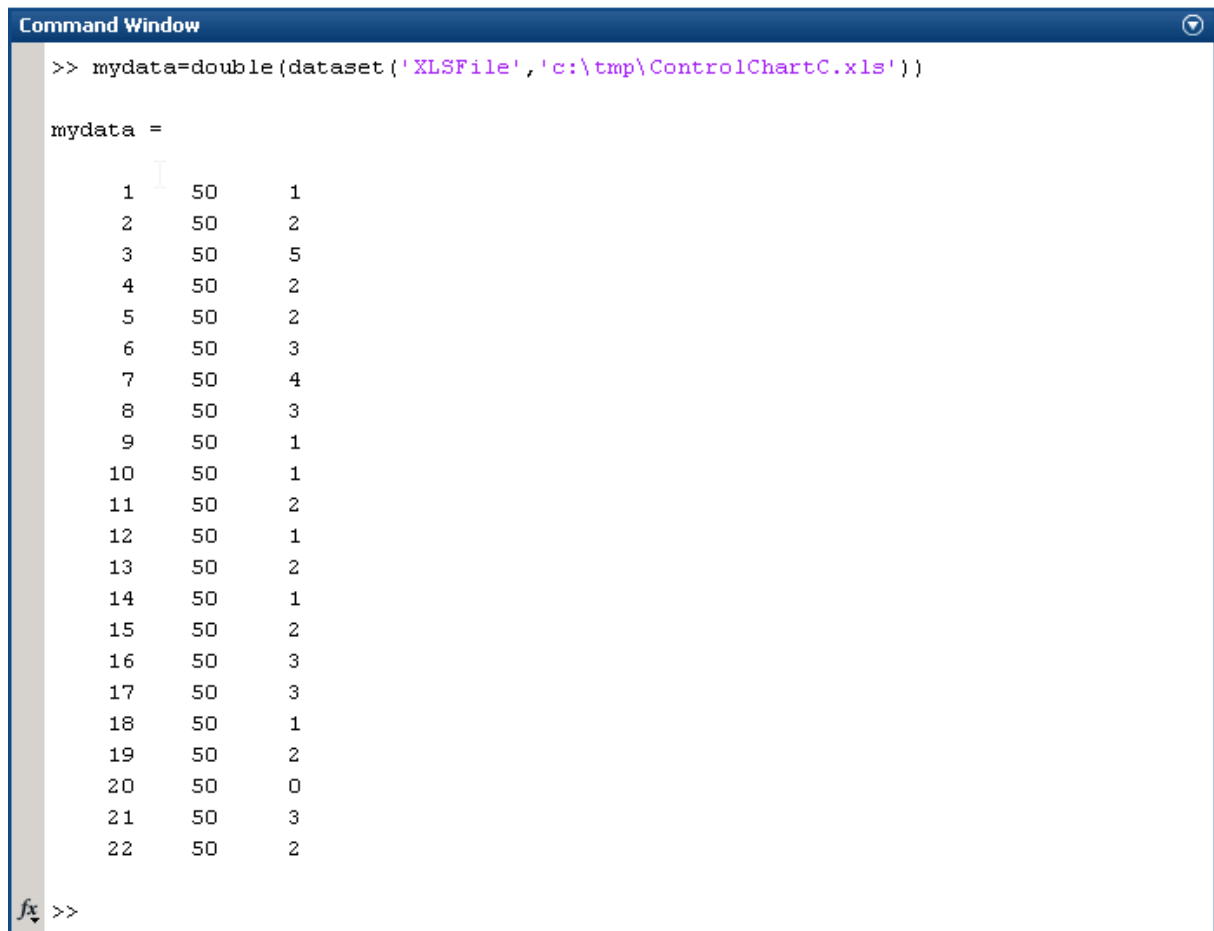
mydata =

      N_Lot  TailleLot  NonConformes
      1      50        1
      2      50        2
      3      50        5
      4      50        2
      5      50        2
      6      50        3
      7      50        4
      8      50        3
      9      50        1
     10      50        1
     11      50        2
     12      50        1
     13      50        2
     14      50        1
     15      50        2
     16      50        3
     17      50        3
     18      50        1
     19      50        2
     20      50        0
     21      50        3
     22      50        2
```



Nice but the lower limit can be interpreted as false...

C attribute CC



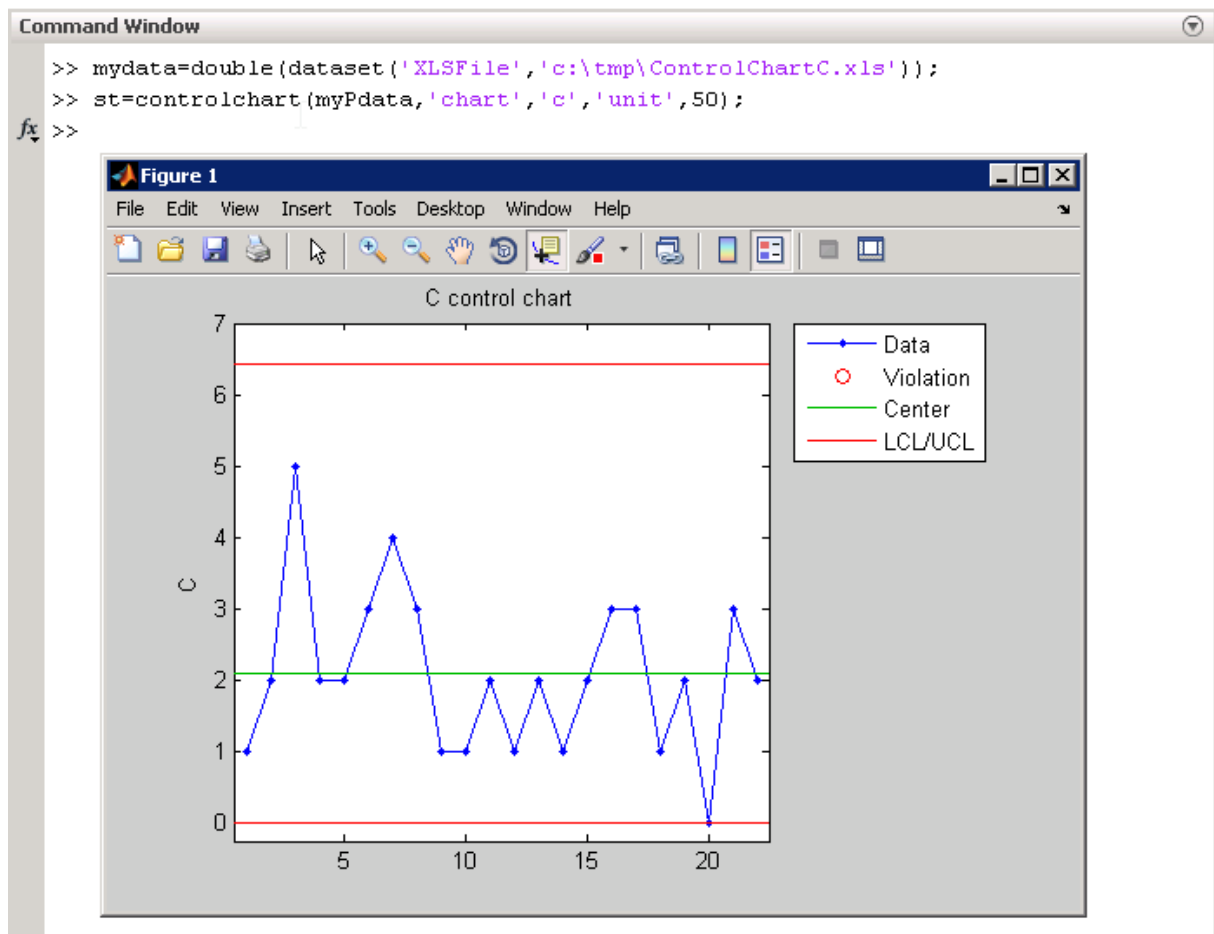
The image shows a MATLAB Command Window with a dark blue title bar. The command `>> mydata=double(dataset('XLSFile','c:\tmp\ControlChartC.xls'))` has been executed. The output displays the variable `mydata` as a 22x3 matrix. The first column contains integers from 1 to 22, the second column contains the value 50 repeated 22 times, and the third column contains integers from 1 to 22, with the last value being 2. A large, light gray watermark with the word "SAMPLE" is oriented diagonally across the lower half of the page.

```
>> mydata=double(dataset('XLSFile','c:\tmp\ControlChartC.xls'))

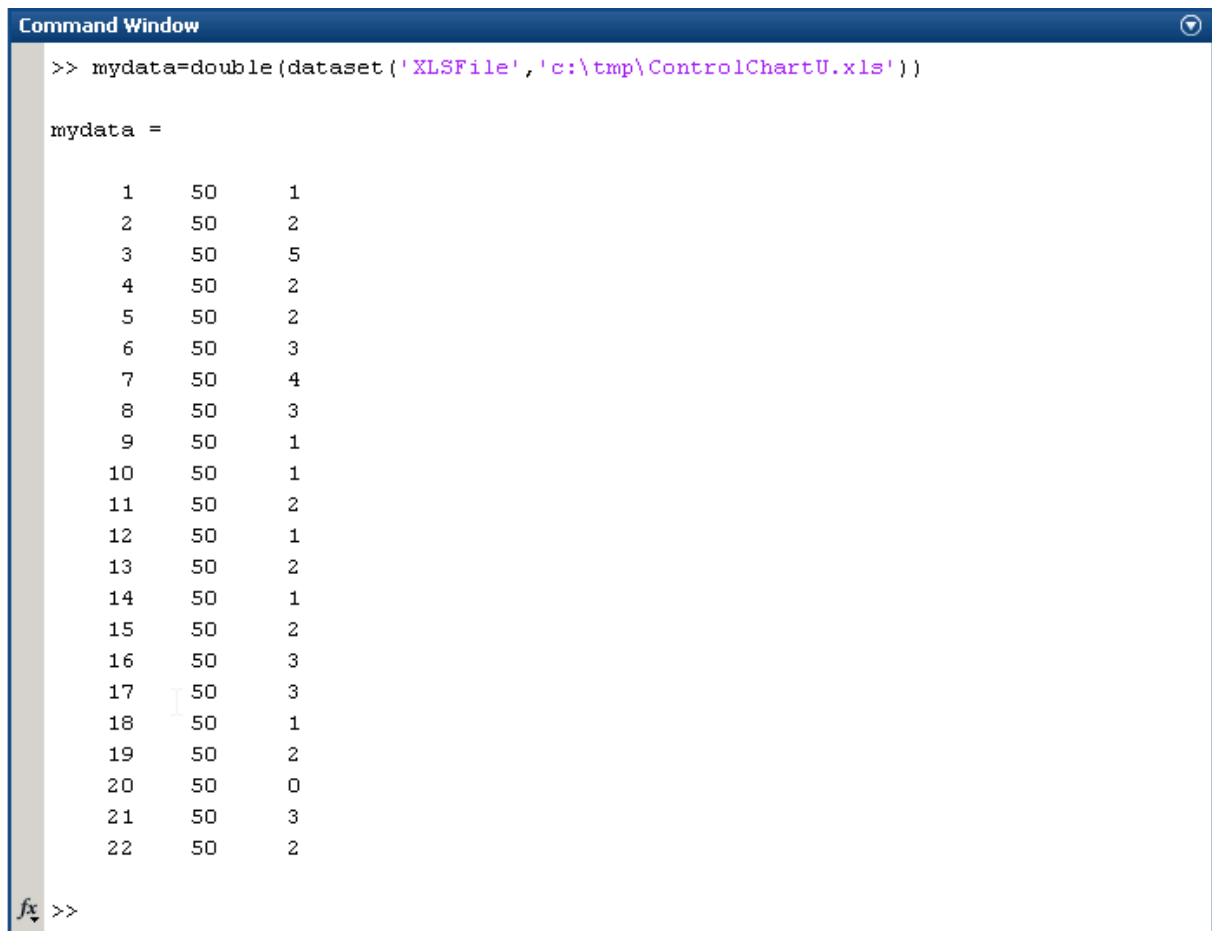
mydata =

     1     50     1
     2     50     2
     3     50     5
     4     50     2
     5     50     2
     6     50     3
     7     50     4
     8     50     3
     9     50     1
    10     50     1
    11     50     2
    12     50     1
    13     50     2
    14     50     1
    15     50     2
    16     50     3
    17     50     3
    18     50     1
    19     50     2
    20     50     0
    21     50     3
    22     50     2

fx >>
```



U attribute CC



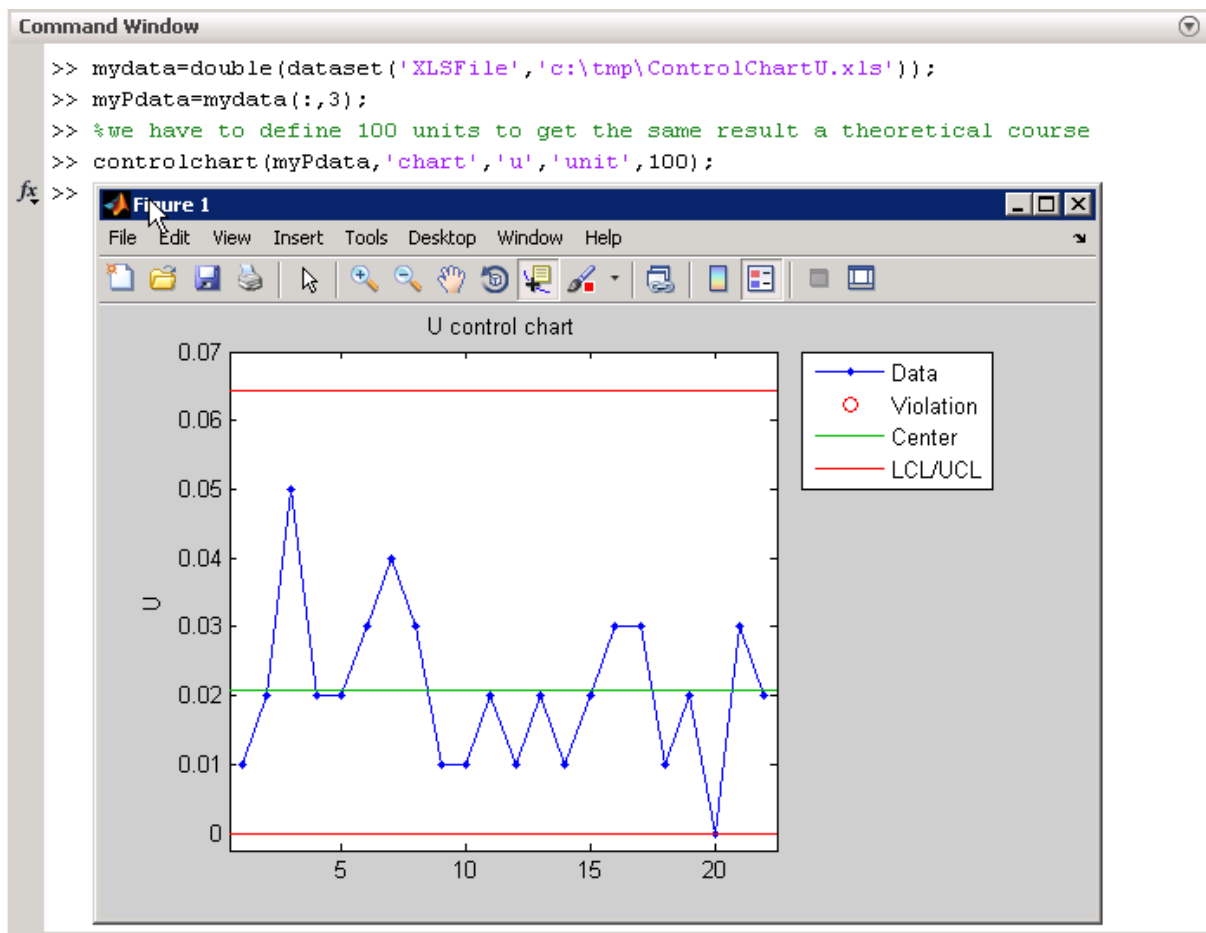
The image shows a MATLAB Command Window with a dark blue title bar. The command `>> mydata=double(dataset('XLSFile','c:\tmp\ControlChartU.xls'))` has been executed. The output displays the variable `mydata` as a 22x3 matrix. The first column contains integers from 1 to 22, the second column contains the value 50 repeated 22 times, and the third column contains integers from 1 to 22 with the last value being 2. A large, light gray 'SAMPLE' watermark is oriented diagonally across the lower half of the page.

```
>> mydata=double(dataset('XLSFile','c:\tmp\ControlChartU.xls'))

mydata =

     1     50     1
     2     50     2
     3     50     5
     4     50     2
     5     50     2
     6     50     3
     7     50     4
     8     50     3
     9     50     1
    10     50     1
    11     50     2
    12     50     1
    13     50     2
    14     50     1
    15     50     2
    16     50     3
    17     50     3
    18     50     1
    19     50     2
    20     50     0
    21     50     3
    22     50     2

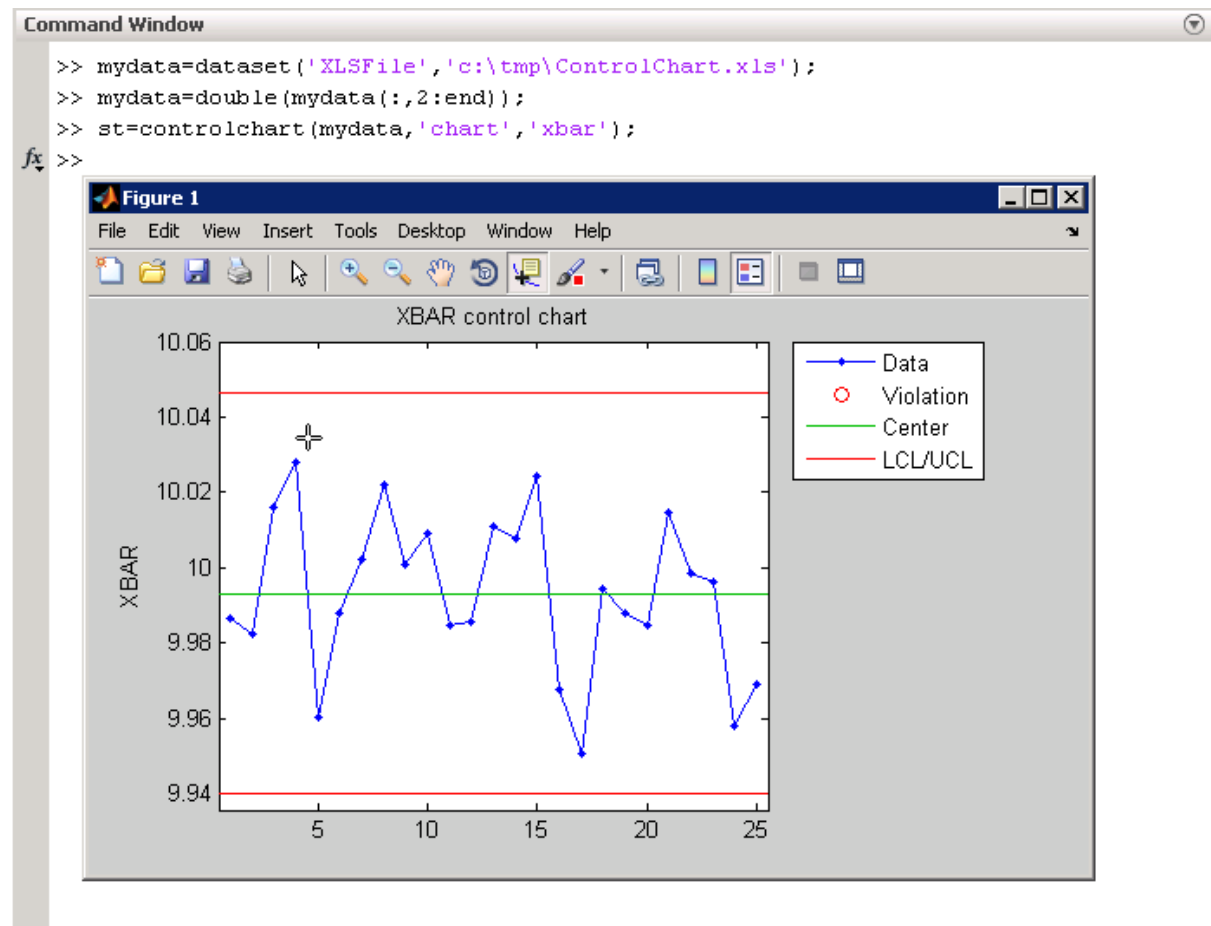
fx >>
```

Nice but the lower limit can also be interpreted as false...

Xbar-R continuous CC

We begin with the most famous control chart the Xbar-R. Once we have imported the data we use the function **controlchart()** and we get:

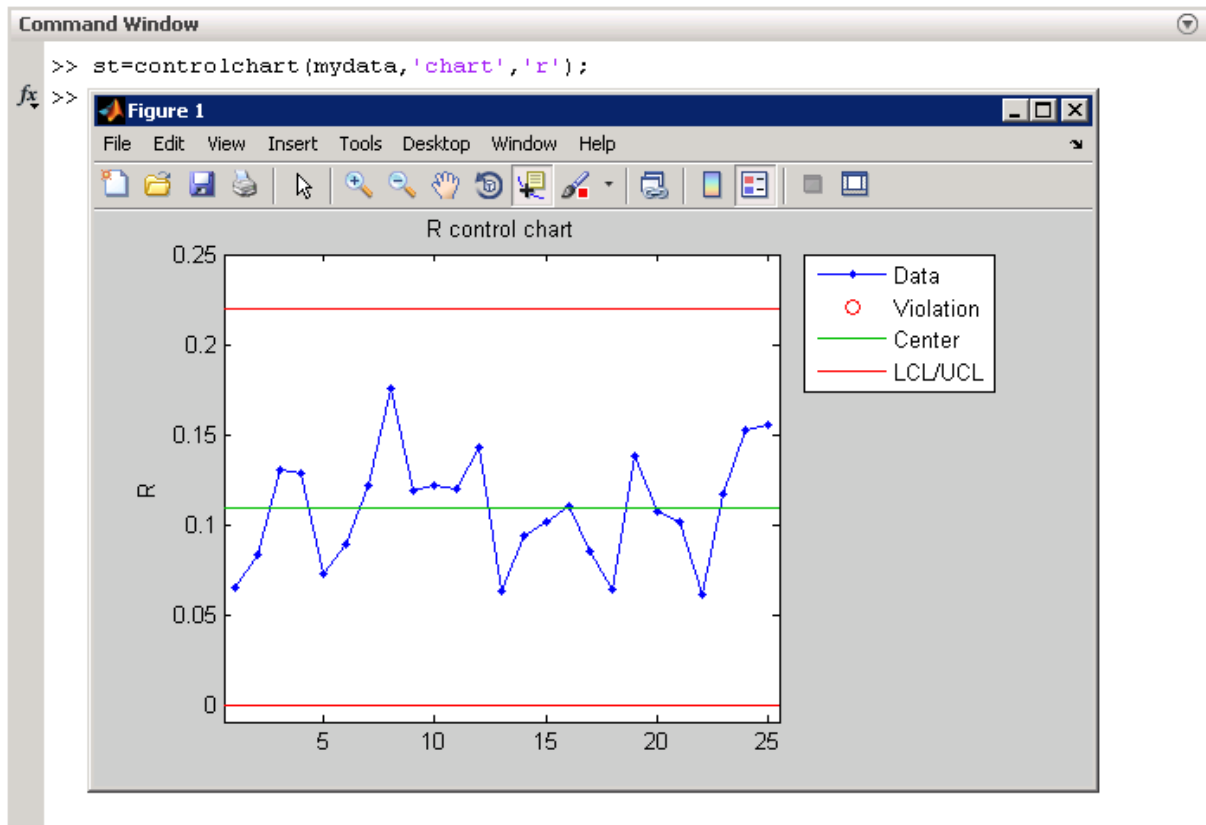


and we can see that we get exactly the same result as the one calculated by hand in the theoretical Industrial Engineering course.

Rbar-R continuous CC

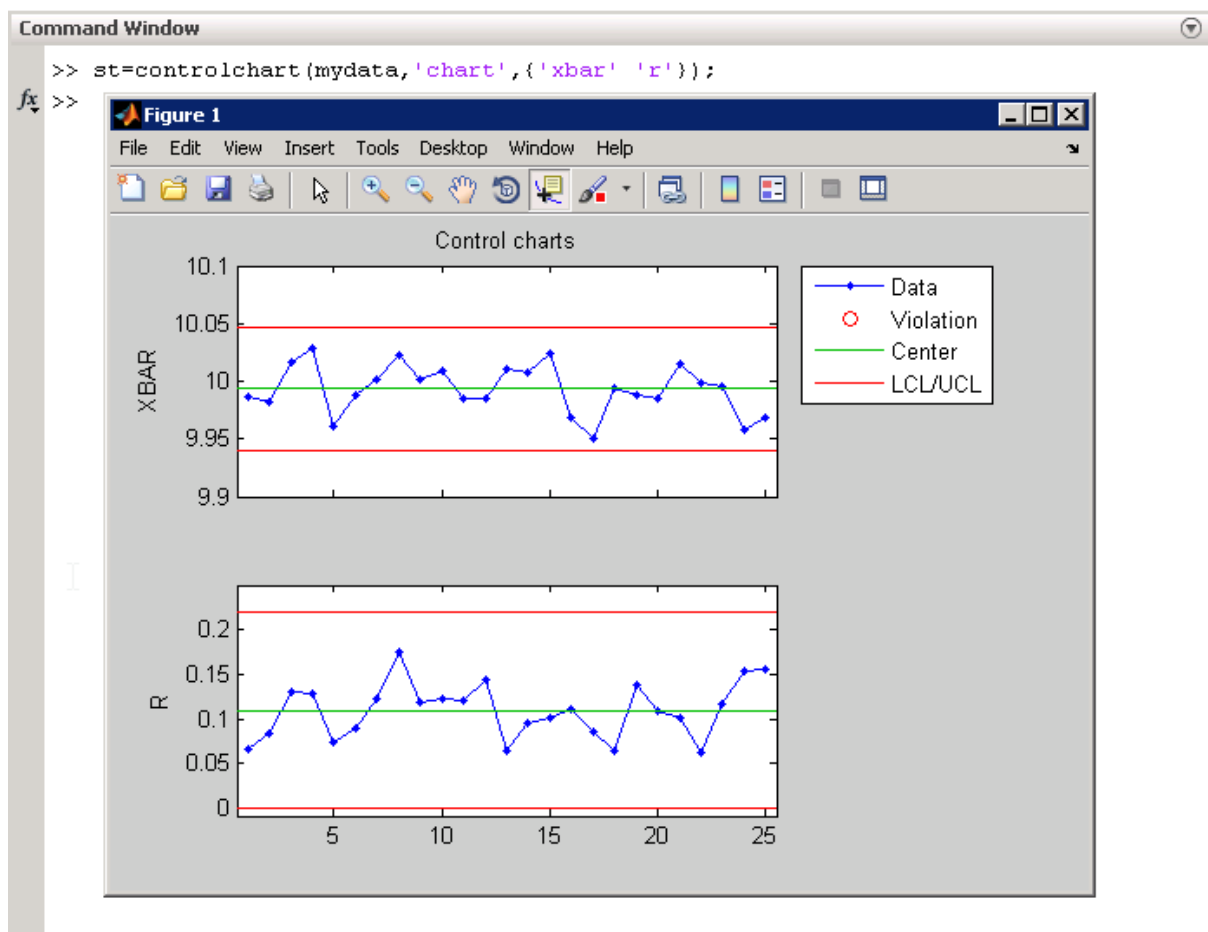
A less common bar chart than the previous one is the the Rbar-R control chart. Here we will also use the same dataset as before to see if we obtain the same result as in the theoretical course.

We always use the **controlchart()** function:



We get exactly the same values as those calculated by hand.

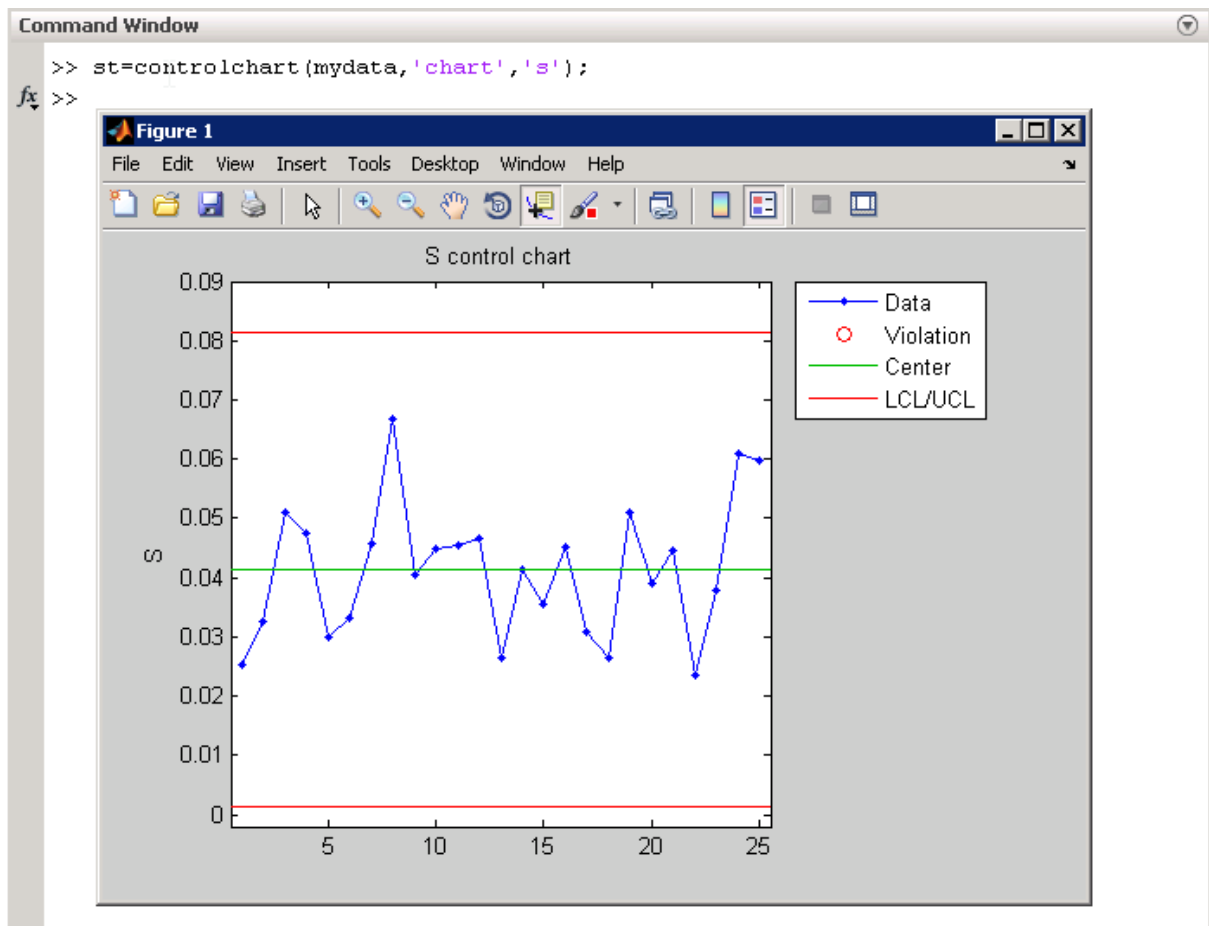
If we want the both chart on the same figure we use the same function again:



Sbar-S continuous CC

Also a less common bar chart than the the Xbar and Rbar one is the the Sbar-S control chart. Here we will also use the same dataset as before to see if we obtain the same result as in the theoretical course.

We always use the **controlchart()** function:



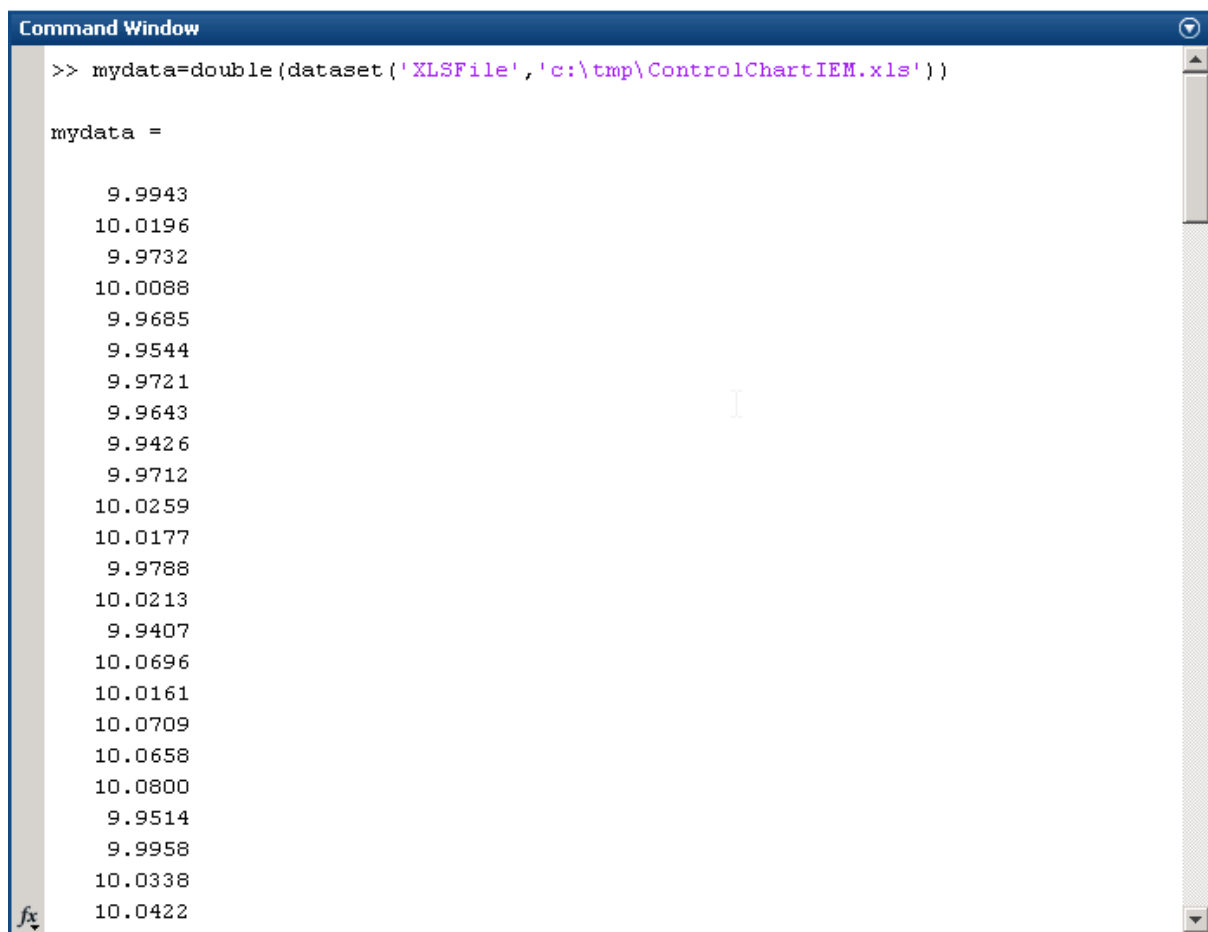
We get exactly the same values as those calculated by hand.

I-EM continuous CC

Individual charts are use a lot in small production process. We will check here if we also get the same individual moving average range control chart as in the theoretical course:

→						
↓	9.9943	10.0196	9.9732	10.0088	9.9685	9.9544
	9.9721	9.9643	9.9426	9.9712	10.0259	10.0177
	9.9788	10.0213	9.9407	10.0696	10.0161	10.0709
	10.0658	10.08	9.9514	9.9958	10.0338	10.0422
	10.0136	9.9791	9.9449	9.9403	9.9424	9.9421
	9.9495	10.0057	9.9715	10.0388	10.0019	9.9616
	9.9144	10.0321	10.0361	10.0253	10.0109	9.9935
	9.9748	9.9962	10.0369	10.15	9.9996	9.9741
	10.0063	10.0749	9.9971	9.978	9.9934	9.9555
	10.084	9.9619	10.0063	10.0364	9.9907	9.9762
	9.9436	10.0638	9.9617	10.0138	9.9672	9.9591
	10.0074	9.9817	9.9753	10.0541	9.9845	9.9107
	10.0069	10.0282	10.0385	9.9747	10.0323	9.986
	9.9836	10.063	9.9824	9.9909	9.9684	10.0575
	10.0317	10.0485	9.9659	10.0675	10.0047	10.0277
	9.9756	9.9174	9.9076	9.9909	10.018	9.9979
	9.9818	9.8962	9.9614	9.9612	9.9694	9.9349
	10.0168	10.0043	10.0029	9.9549	10.0199	9.9691
	9.9857	10.0503	10.0381	9.9811	9.9603	9.9123
	9.9878	9.9841	9.9496	9.9754	9.9532	10.0576
	10.0657	9.9838	9.9636	9.9779	10.0423	10.0547
	9.9564	10.0184	9.989	9.9999	10.0146	10.0145
	10.0611	9.9443	9.9899	9.9833	9.9947	10.0033
	9.9054	9.9049	9.987	10.058	9.9156	9.9782
	9.9768	9.9954	9.9196	9.9341	9.9162	10.0721

We first import the data as usual:

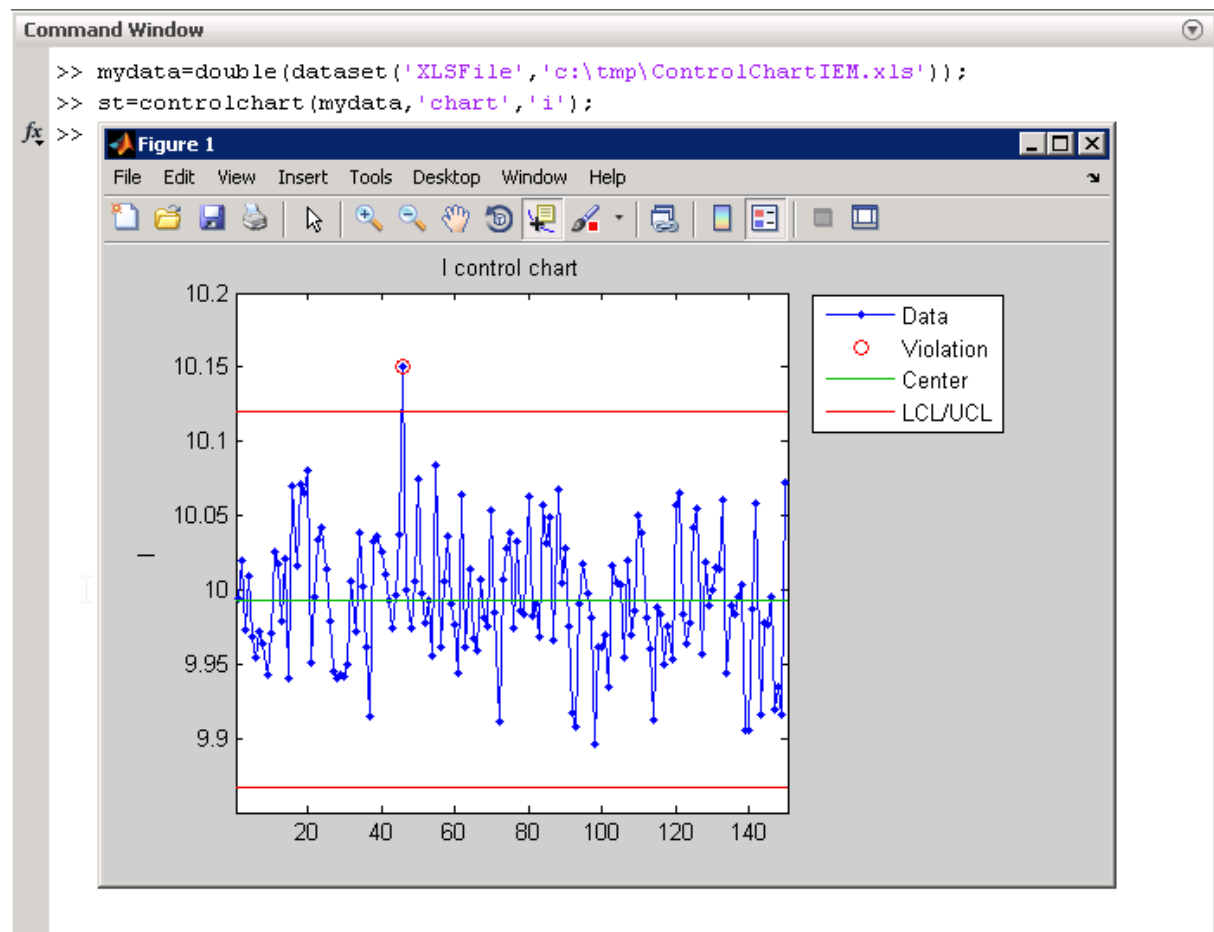
A screenshot of the MATLAB Command Window. The title bar reads "Command Window". The command prompt shows the command: `>> mydata=double(dataset('XLSFile','c:\tmp\ControlChartIEM.xls'))`. Below the command, the output is displayed as `mydata =` followed by a vertical list of 20 numerical values. A large, light gray "SAMPLE" watermark is oriented diagonally across the lower half of the image.

```
>> mydata=double(dataset('XLSFile','c:\tmp\ControlChartIEM.xls'))

mydata =

    9.9943
   10.0196
    9.9732
   10.0088
    9.9685
    9.9544
    9.9721
    9.9643
    9.9426
    9.9712
   10.0259
   10.0177
    9.9788
   10.0213
    9.9407
   10.0696
   10.0161
   10.0709
   10.0658
   10.0800
    9.9514
    9.9958
   10.0338
   10.0422
```

Once the data are imported we use once again the function **controlchart()** but with a special parameter:

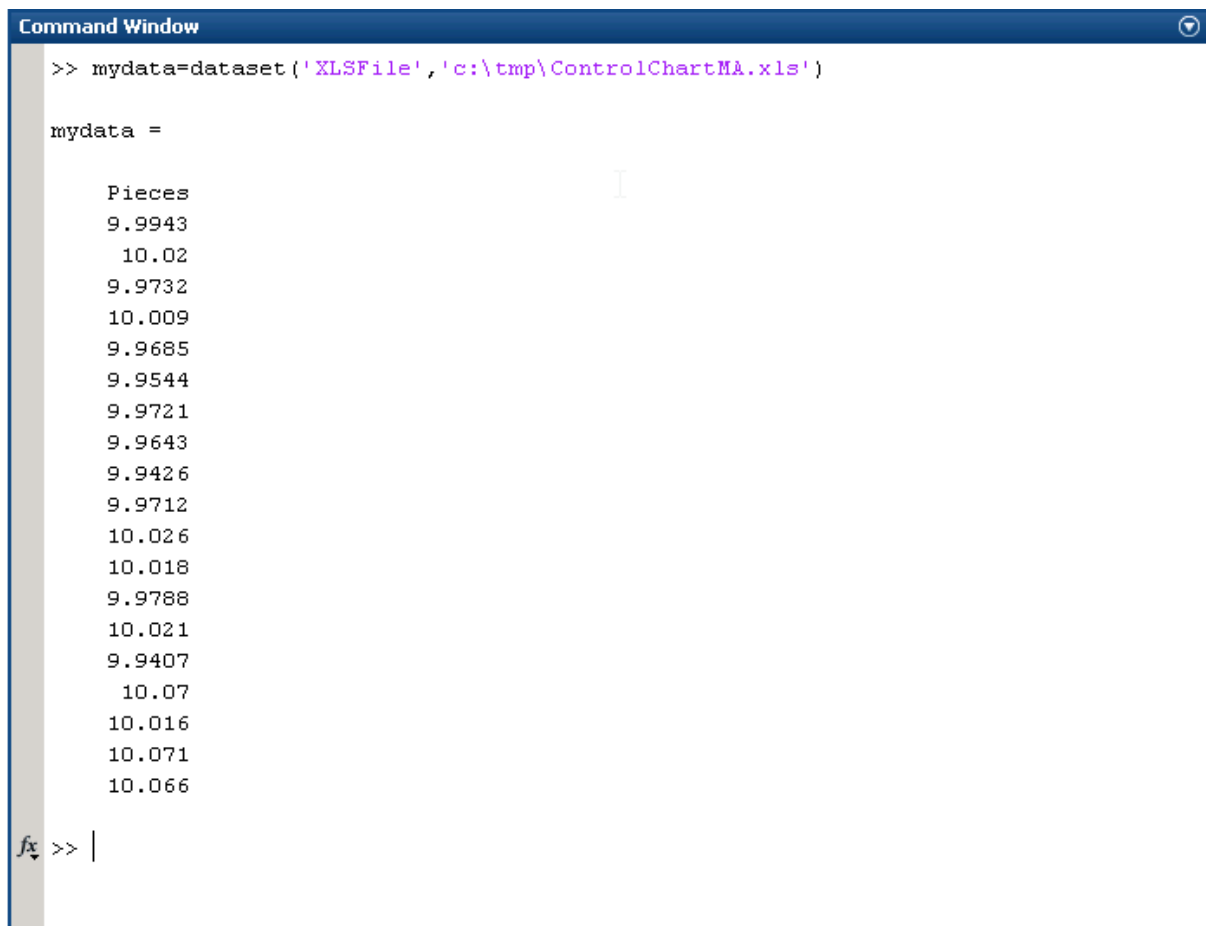


We get exactly the same values as those calculated by hand.

Moving Average continuous CC

We saw in the theoretical course that Moving Average control charts belongs to the family of auto correlated (time series like) control charts.

We will also here use the same data as in the theoretical course:

A screenshot of the MATLAB Command Window. The title bar reads "Command Window". The command prompt shows the command: `>> mydata=dataset('XLSFile','c:\tmp\ControlChartMA.xls')`. Below the command, the output shows the variable `mydata` as a dataset with the following values:
`mydata =`
`Pieces`
`9.9943`
`10.02`
`9.9732`
`10.009`
`9.9685`
`9.9544`
`9.9721`
`9.9643`
`9.9426`
`9.9712`
`10.026`
`10.018`
`9.9788`
`10.021`
`9.9407`
`10.07`
`10.016`
`10.071`
`10.066`
At the bottom, the prompt `fx >> |` is visible.

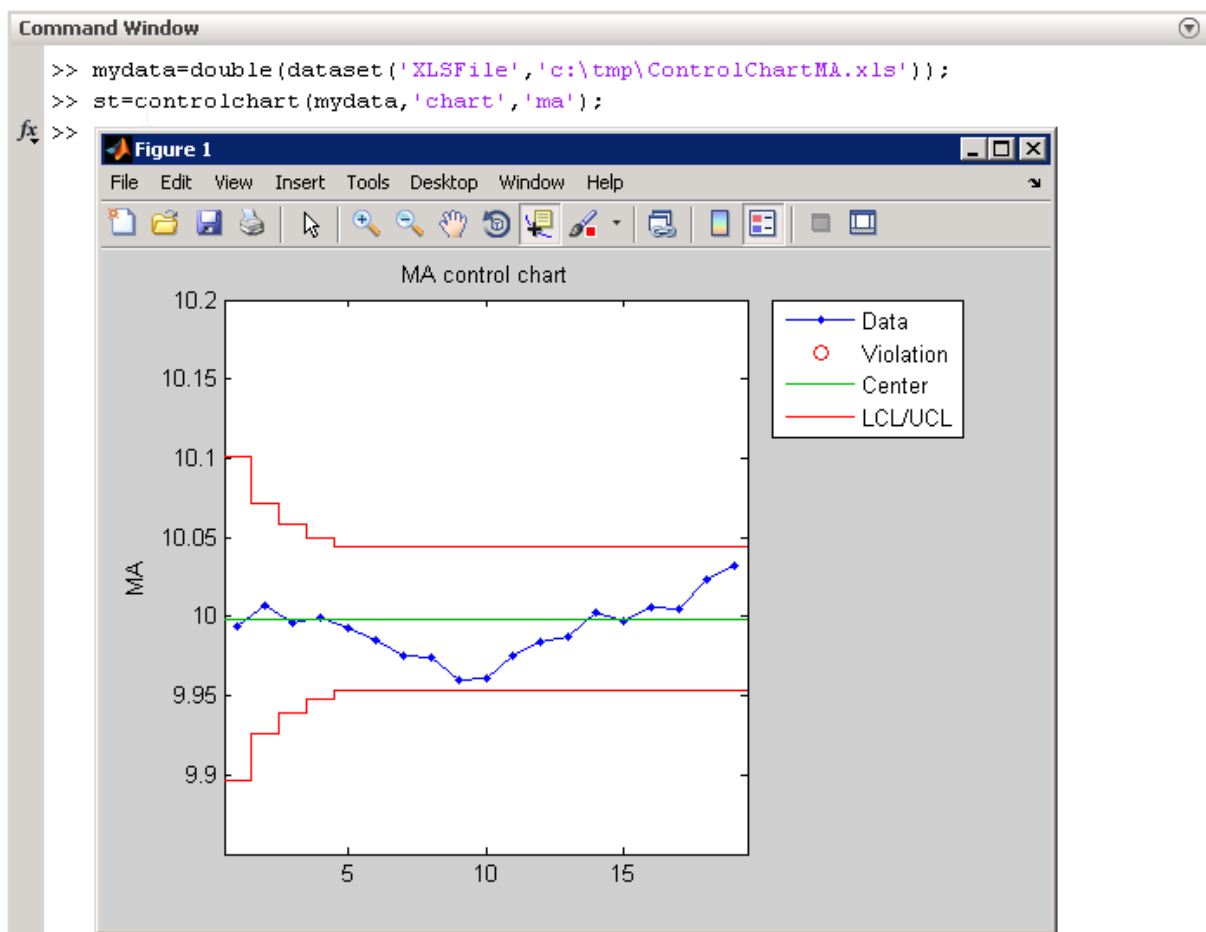
```
Command Window
>> mydata=dataset('XLSFile','c:\tmp\ControlChartMA.xls')

mydata =

    Pieces
    9.9943
    10.02
    9.9732
    10.009
    9.9685
    9.9544
    9.9721
    9.9643
    9.9426
    9.9712
    10.026
    10.018
    9.9788
    10.021
    9.9407
    10.07
    10.016
    10.071
    10.066

fx >> |
```

We use again the function `controlchart()` with a specific argument. We then get:



Here also we get the same values as in the theoretical course except for the first three point.

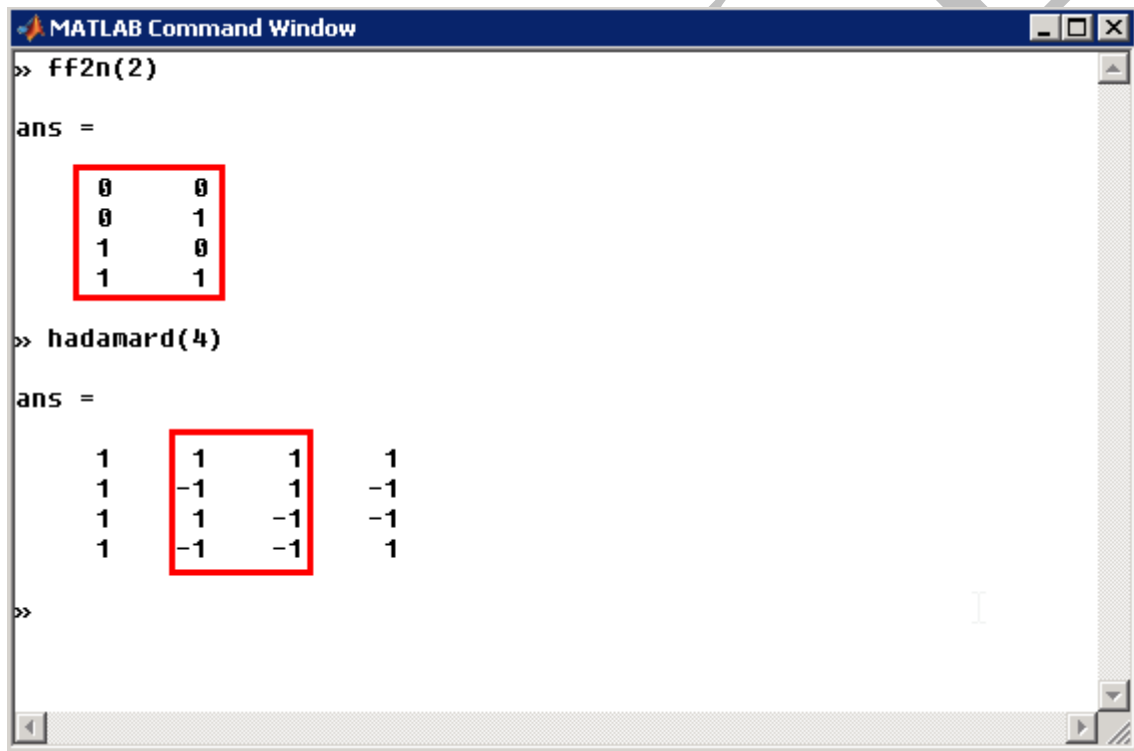
Design of experiments

As we know, designs of experiments are a huuuuge subject! MATLAB is very poor in this area to help to simplify the life of the end user. I strongly recommend to use instead softwares like JMP, Minitab, DesignExpert or R.

In the point of view of the MathWorks dev team (point of view that is fact not totally false), be able to generate a factorial design is enough for processing after the response data with all existing functions of the Statistical Toolbox. Even if it's true it's a pain in the ass for the end user this is why at least 99% of people prefer to use specialized softwares.

Two-level full factorial design

We will check if we get the some simple factorial design that we have calculated by hand in the theoretical course using the function **ff2n()**:



```
MATLAB Command Window
>> ff2n(2)
ans =
 0  0
 0  1
 1  0
 1  1

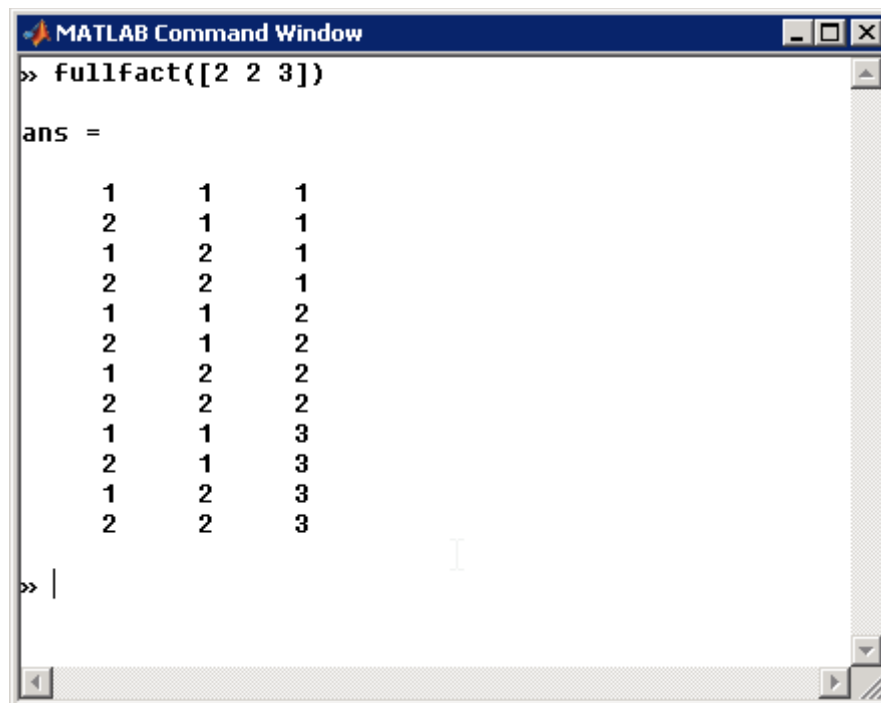
>> hadamard(4)
ans =
 1  1  1  1
 1 -1  1 -1
 1  1 -1 -1
 1 -1 -1  1
```

We have also use the function **hadamard()** as you can see above to compare it with the **ff2n(2)** and see that we can conclude to the same thing as in the theoretical course.

Everything is fine here!

Full factorial design

Now have a look to the full factorial design with 3 factors of respectively 2, 2 and 3 levels. Using the function **fullfact()** we get as expected the 12 rows.



A screenshot of the MATLAB Command Window. The title bar reads "MATLAB Command Window". The command prompt shows the command `>> fullfact([2 2 3])`. The output is displayed as `ans =` followed by a 12x3 matrix of values. The values are arranged in three columns, representing the levels of the three factors. The first column contains values 1 and 2, the second column contains values 1 and 2, and the third column contains values 1, 2, and 3. The rows represent the 12 combinations of the three factors.

```
>> fullfact([2 2 3])

ans =

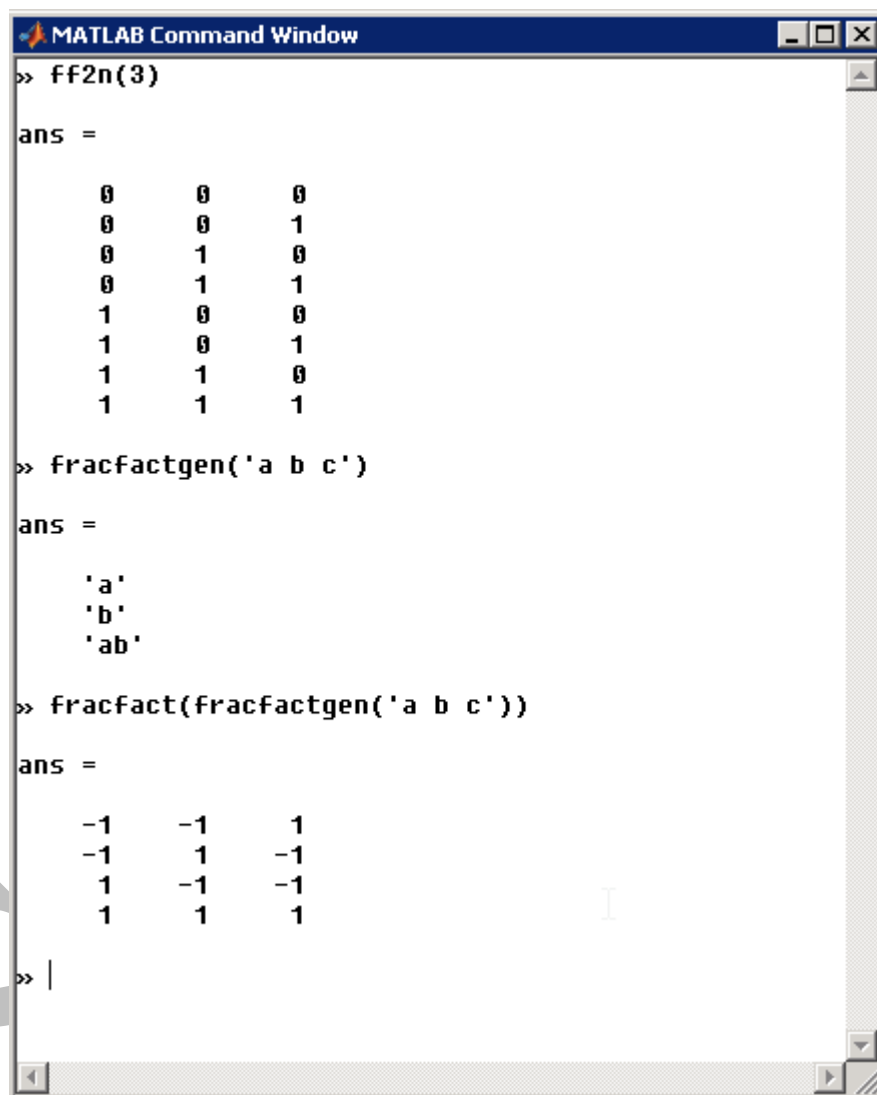
     1     1     1
     2     1     1
     1     2     1
     2     2     1
     1     1     2
     2     1     2
     1     2     2
     2     2     2
     1     1     3
     2     1     3
     1     2     3
     2     2     3

>> |
```

Fractional factorial design

First we use the **ff2n()** function to get the full factorial one as in the theoretical course. Once this is done, we chose the generators as we saw it in the theoretical course using the function **fracfactgen()**.

And after only we use the **fracfact()** function to get the fractional design we calculated by hand in the theoretical course!



```
MATLAB Command Window
>> ff2n(3)

ans =

     0     0     0
     0     0     1
     0     1     0
     0     1     1
     1     0     0
     1     0     1
     1     1     0
     1     1     1

>> fracfactgen('a b c')

ans =

     'a'
     'b'
     'ab'

>> fracfact(fracfactgen('a b c'))

ans =

    -1    -1     1
    -1     1    -1
     1    -1    -1
     1     1     1

>> |
```

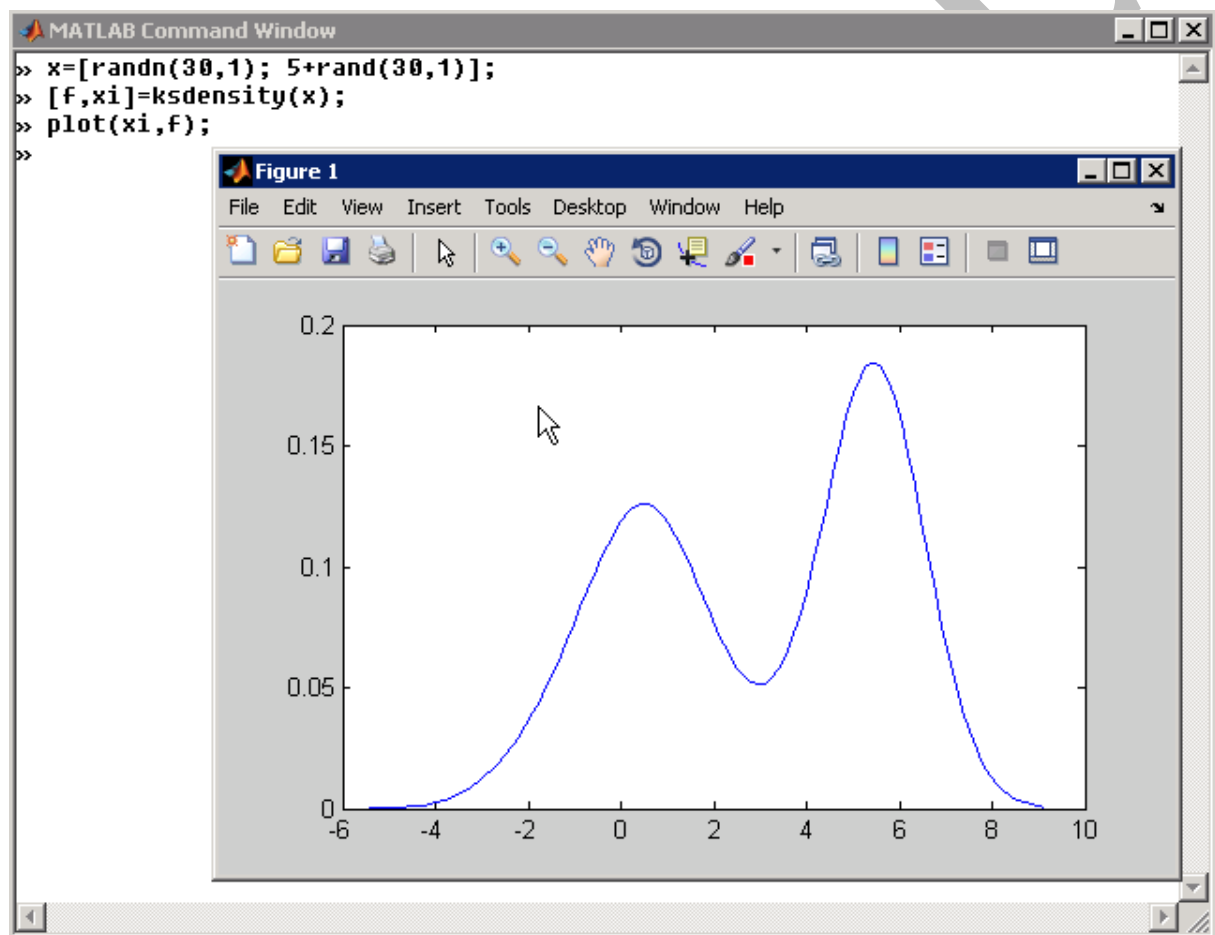
Simulations techniques

Once again we will focus in this section only about stuff that we have study in details during the theoretical course.

Kernel Smoothing

As we know, using a histogram to judge the shape of a distribution is a bias in itself. Using Kernel Smoothing or even better CDF is more accurate.

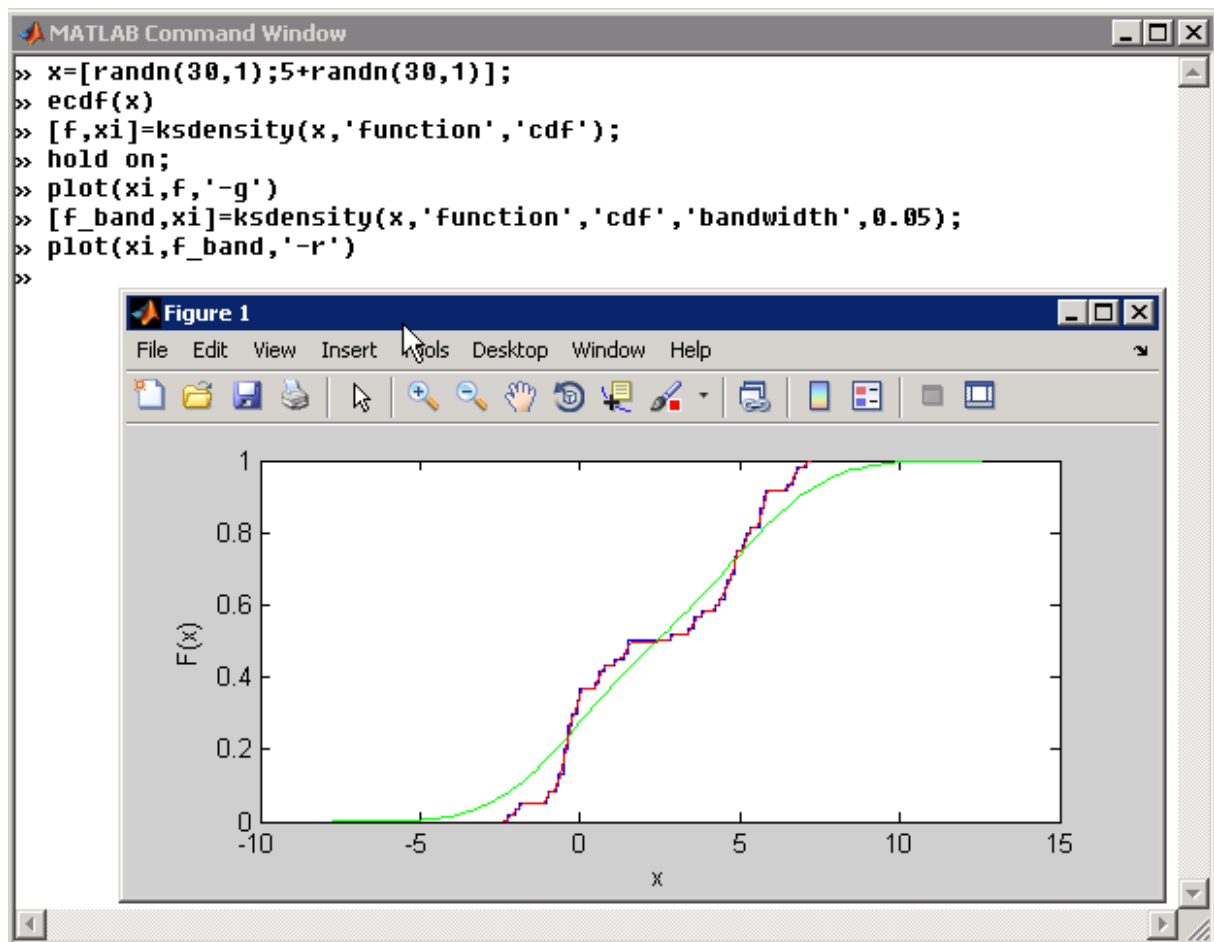
Because the purpose here is to check that Kernel Smoothing exists in MATLAB let us generate a vector of two different Normal random variable and apply the **ksdensity()** function on it with default bandwidth and the default hypothesized distribution (aka the Normal one...):



So fine... seems to do something put in practice I'm not a fan of charts. I want numbers, numbers and numbers!

So can we retrieve any information about this previous kernel smoothing density with MATLAB? Yes we can!!

For example we compare here below the ECDF



Monte Carlo standard technic

The most simple and academic example of Monte-Carlo technics is to play with different random variables and calculates some of their moments.

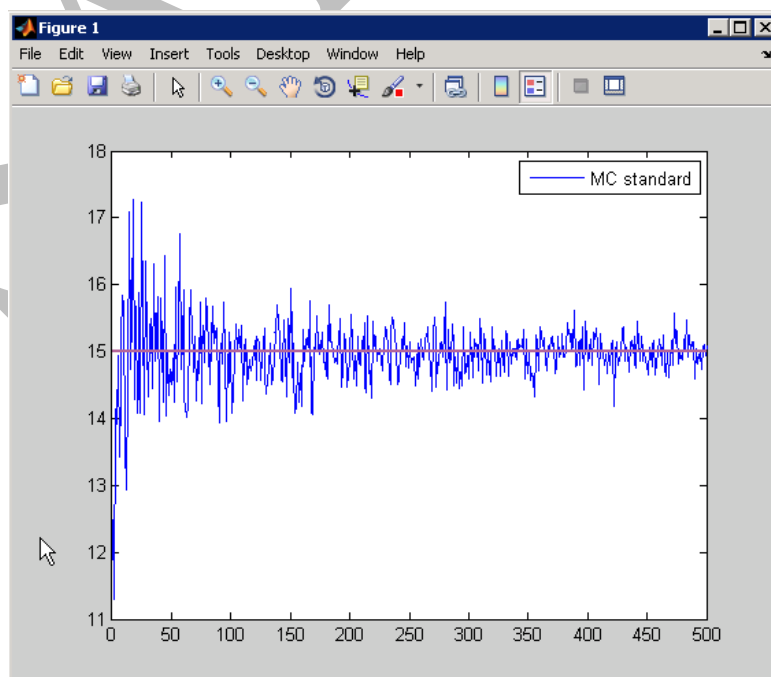
The most trivial and commun case is to compute de mean of the sum of two random Normal variables because we know the Normal distribution is stable by the sum:

```

1      %standard MC method
2      clear all;clc
3      int_Simulations=500;
4      t=1:1:int_Simulations;
5      for k=1:1:int_Simulations
6          x=rand(k,1);
7          y=rand(k,1);
8          mux=5;xsigma=3;
9          muy=10;ysigma=4;
10         Nx=norminv(x,mux,xsigma); %N(5,3)
11         Ny=norminv(y,muy,ysigma); %N(10,4)
12         muNxy(k)=mean(Nx+Ny)
13     end
14     plot(t,muNxy)
15     hy=graph2d.constantline(15,'Color',[0.7 0.4 0.6],'LineWidth',2);
16     changedependvar(hy,'y');
17     legend('MC standard')
18

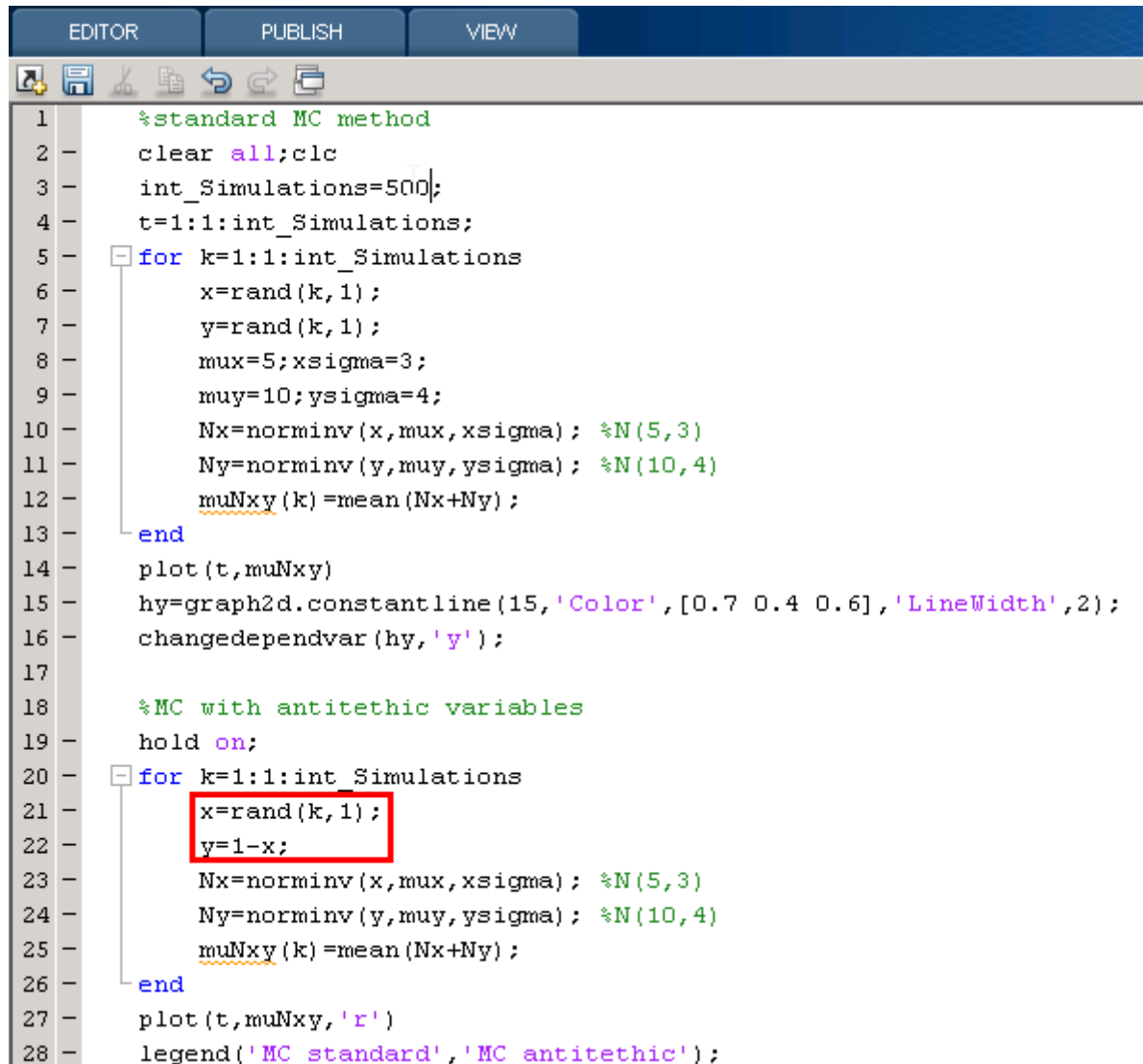
```

We can see qualitatively the convergence with the following chart:



Monte Carlo antithetic variables technique

The purpose is to see the famous and simple antithetic variables technique and compare it to the standard one

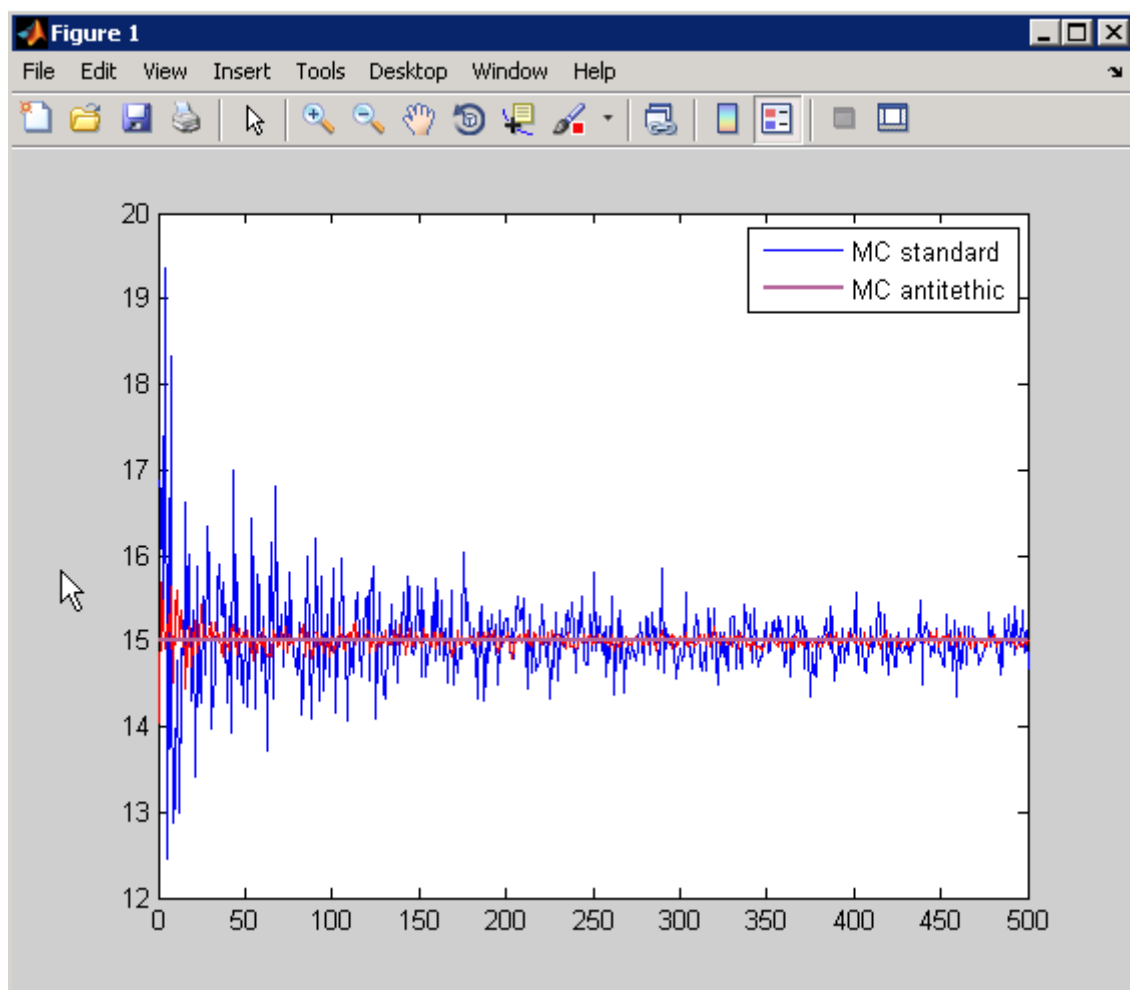


```

1  %standard MC method
2  clear all;clc
3  int_Simulations=500;
4  t=1:1:int_Simulations;
5  for k=1:1:int_Simulations
6      x=rand(k,1);
7      y=rand(k,1);
8      mux=5;xsigma=3;
9      muy=10;ysigma=4;
10     Nx=norminv(x,mux,xsigma); %N(5,3)
11     Ny=norminv(y,muy,ysigma); %N(10,4)
12     muNxy(k)=mean(Nx+Ny);
13 end
14 plot(t,muNxy)
15 hy=graph2d.constantline(15,'Color',[0.7 0.4 0.6],'LineWidth',2);
16 changedependvar(hy,'y');
17
18 %MC with antitethic variables
19 hold on;
20 for k=1:1:int_Simulations
21     x=rand(k,1);
22     y=1-x;
23     Nx=norminv(x,mux,xsigma); %N(5,3)
24     Ny=norminv(y,muy,ysigma); %N(10,4)
25     muNxy(k)=mean(Nx+Ny);
26 end
27 plot(t,muNxy,'r')
28 legend('MC standard','MC antitethic');

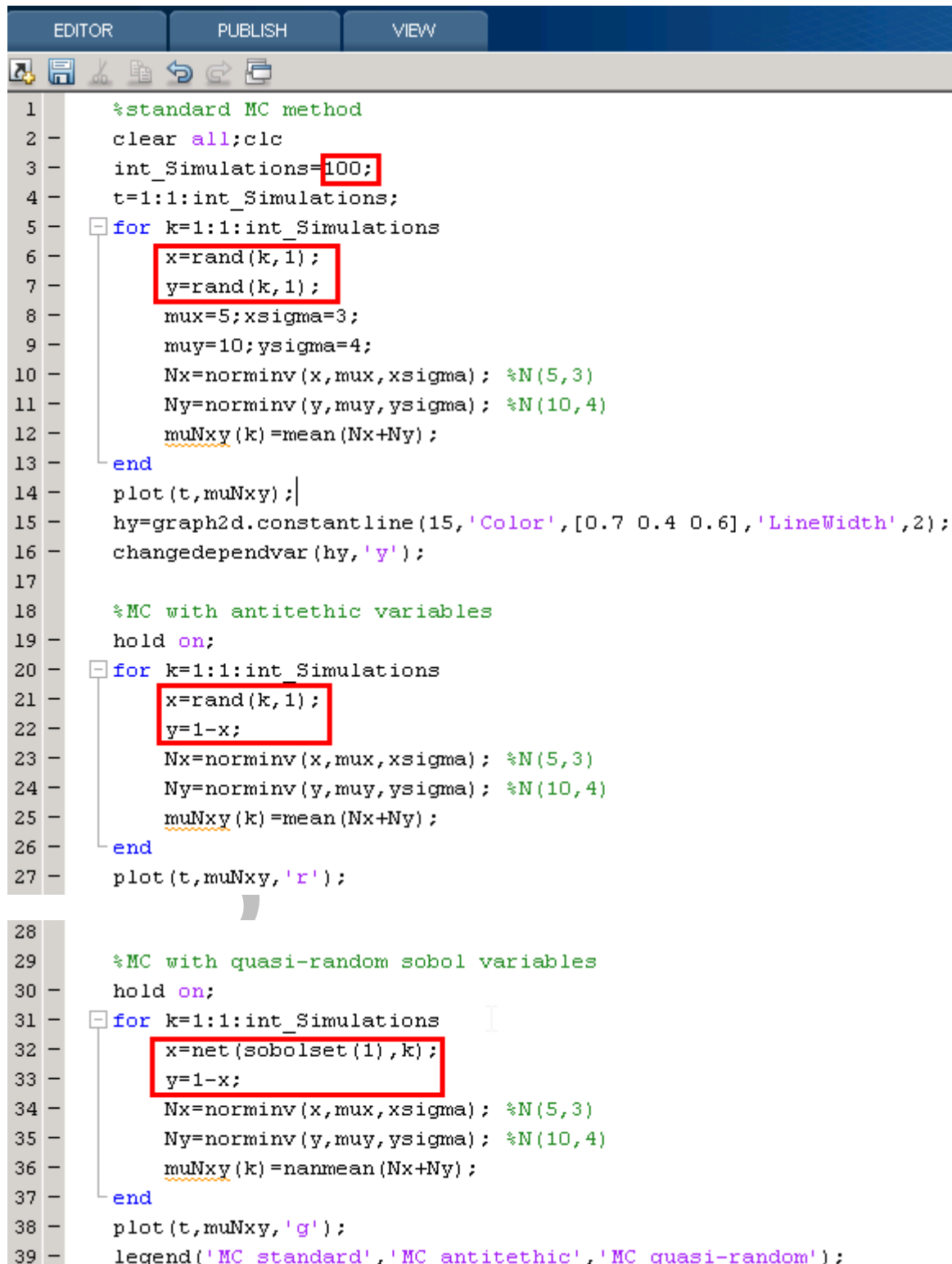
```

And as we can see on the next page, it is logic that in converge faster:



Monte Carlo with antithetic quasi-random variables technique

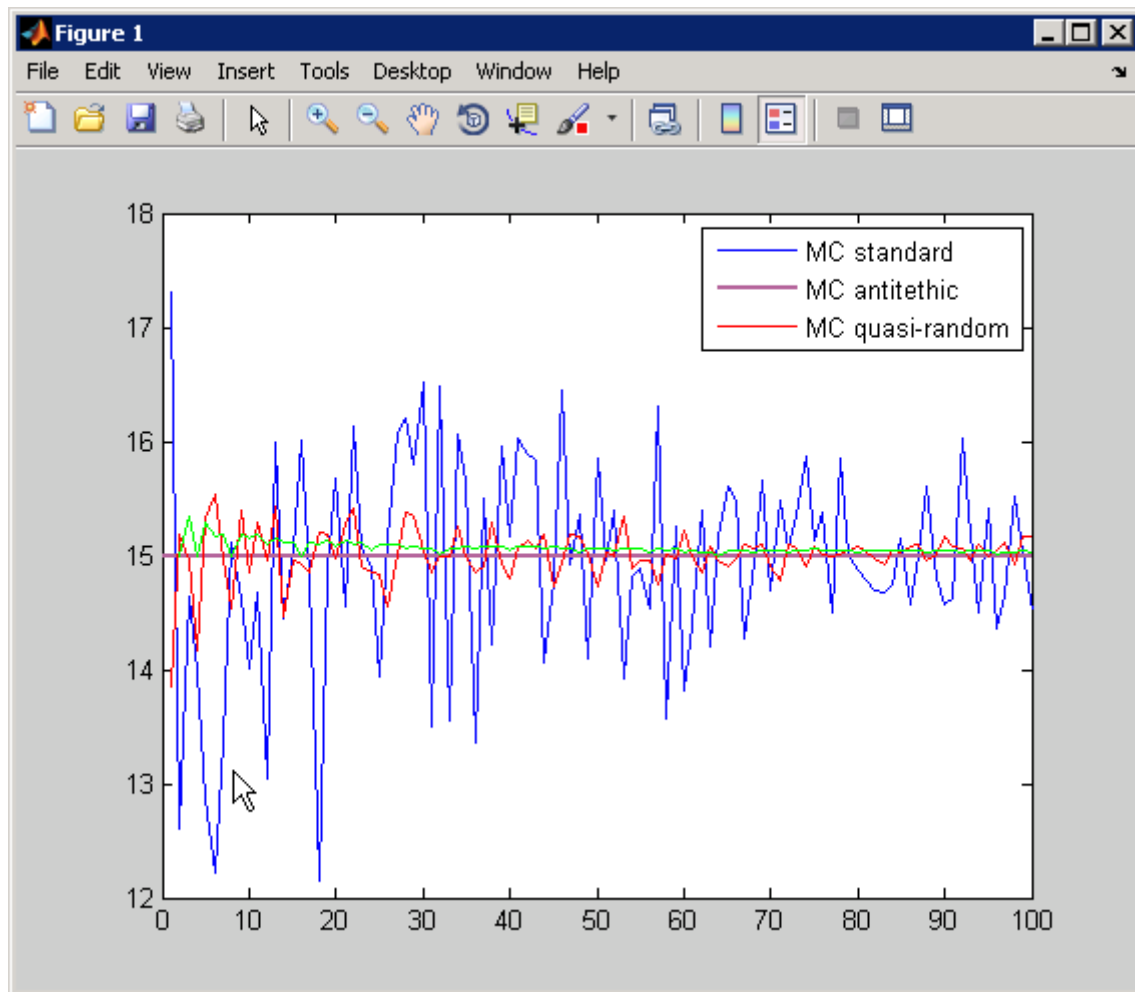
As we saw it in the theoretical course there a lot of quasi-random variables generators. Here we will compare the two previous method with the Sobol quasi-random generator using the function `sobolset()`:



```

1      %standard MC method
2      clear all;clc
3      int_Simulations=100;
4      t=1:1:int_Simulations;
5      for k=1:1:int_Simulations
6          x=rand(k,1);
7          y=rand(k,1);
8          mux=5;xsigma=3;
9          muy=10;ysigma=4;
10         Nx=norminv(x,mux,xsigma); %N(5,3)
11         Ny=norminv(y,muy,ysigma); %N(10,4)
12         muNxy(k)=mean(Nx+Ny);
13     end
14     plot(t,muNxy);|
15     hy=graph2d.constantline(15,'Color',[0.7 0.4 0.6],'LineWidth',2);
16     changedependvar(hy,'y');
17
18     %MC with antitethic variables
19     hold on;
20     for k=1:1:int_Simulations
21         x=rand(k,1);
22         y=1-x;
23         Nx=norminv(x,mux,xsigma); %N(5,3)
24         Ny=norminv(y,muy,ysigma); %N(10,4)
25         muNxy(k)=mean(Nx+Ny);
26     end
27     plot(t,muNxy,'r');
28
29     %MC with quasi-random sobol variables
30     hold on;
31     for k=1:1:int_Simulations
32         x=net(sobolset(1),k);
33         y=1-x;
34         Nx=norminv(x,mux,xsigma); %N(5,3)
35         Ny=norminv(y,muy,ysigma); %N(10,4)
36         muNxy(k)=nanmean(Nx+Ny);
37     end
38     plot(t,muNxy,'g');
39     legend('MC standard','MC antitethic','MC quasi-random');

```



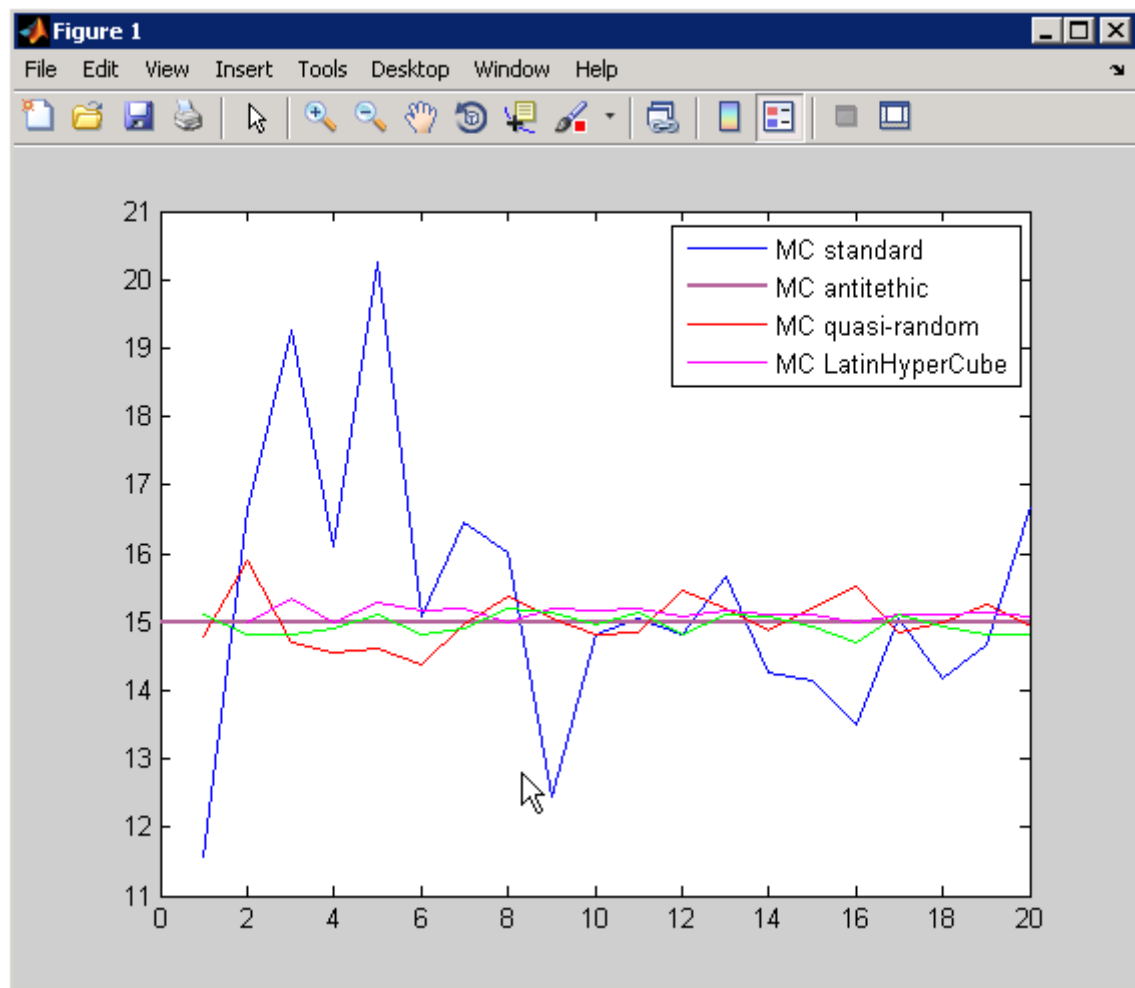
Latin HyperCube

We have also study and applied practically with Microsoft Excel Sheets (it was a nightmare if you remember it...) in the theoretical course a more elaborated method than the previous one called Latin HyperCube. This possibility is implemented in MATLAB trough the function `lhsdesign()`:

```

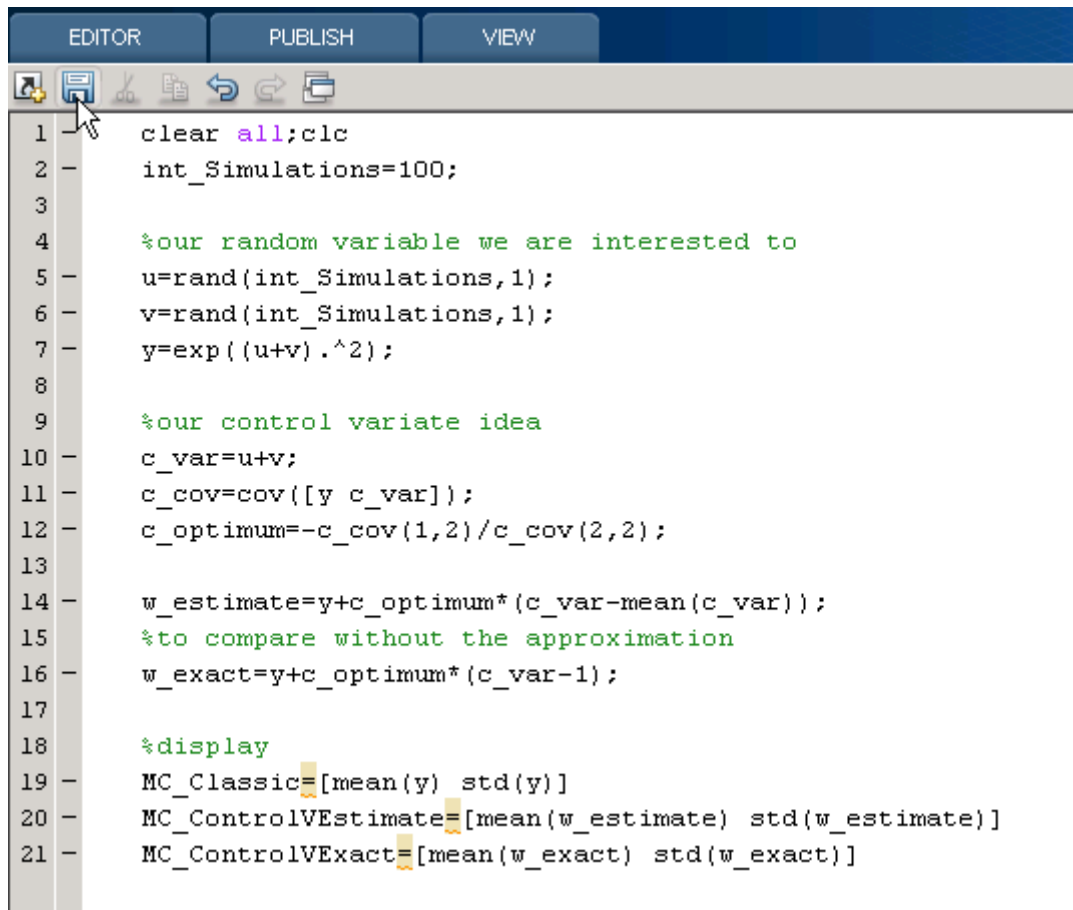
40 %MC with LatinHyperCube
41 hold on;
42 for k=1:1:int_Simulations
43     X=lhsdesign(int_Simulations,2);
44     x=X(:,1);
45     y=X(:,2);
46     Nx=norminv(x,mux,xsigma); %N(5,3)
47     Ny=norminv(y,muy,ysigma); %N(10,4)
48     muNxy(k)=nanmean(Nx+Ny);
49 end
50 plot(t,muNxy,'g');
51 legend('MC standard','MC antitethic','MC quasi-random','MC LatinHyperCube');
```

With 20 simulations we get:



As we can see the difference between the two last methods are qualitatively very small.

Monte Carlo by control variable (variance reduction technique)



```
1 clear all;clc
2 int_Simulations=100;
3
4 %our random variable we are interested to
5 u=rand(int_Simulations,1);
6 v=rand(int_Simulations,1);
7 y=exp((u+v).^2);
8
9 %our control variate idea
10 c_var=u+v;
11 c_cov=cov([y c_var]);
12 c_optimum=-c_cov(1,2)/c_cov(2,2);
13
14 w_estimate=y+c_optimum*(c_var-mean(c_var));
15 %to compare without the approximation
16 w_exact=y+c_optimum*(c_var-1);
17
18 %display
19 MC_Classic=[mean(y) std(y)]
20 MC_ControlVEstimate=[mean(w_estimate) std(w_estimate)]
21 MC_ControlVExact=[mean(w_exact) std(w_exact)]
```

```
Command Window

int_Simulations =

    100

MC_Classic =

    5.1998    7.0634

MC_ControlVEstimate =

    5.1998    4.7898

MC_ControlVExact =

    4.8345    4.7898

fx >>
```

```
Command Window

int_Simulations =

  1000000

MC_Classic =

    4.8945    5.9405

MC_ControlVEstimate =

    4.8945    3.8143

MC_ControlVExact =

    4.8944    3.8143

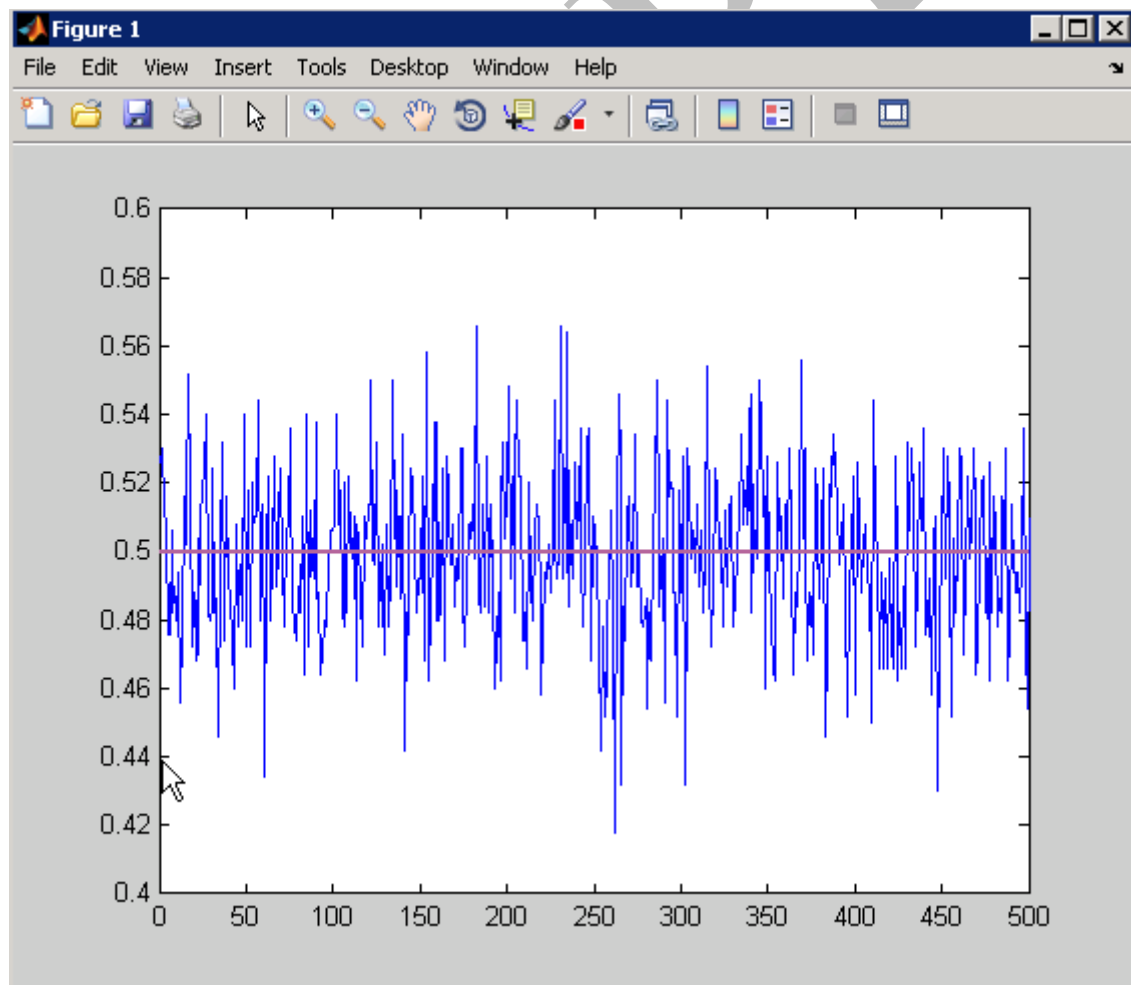
fx >> |
```

Monte Carlo importance sampling

```

EDITOR PUBLISH VIEW
1      %Importance sampling
2      clear all;clc
3      int_Simulations=500;
4      %with significative treshold values no problems for CDF
5      treshold=0;
6      t=1:1:int_Simulations;
7      mu=0;sigma=1;
8      for k=1:1:int_Simulations
9          Nx=normrnd(mu,sigma,1,int_Simulations); %N(3,1)
10         cProb(k)=nnz(Nx(:)<=treshold)/int_Simulations;
11     end
12     plot(t,cProb);
13     cProb(k)
14     hy=graph2d.constantline(0.5,'Color',[0.7 0.4 0.6],'LineWidth',2);
15     changedependvar(hy,'y');

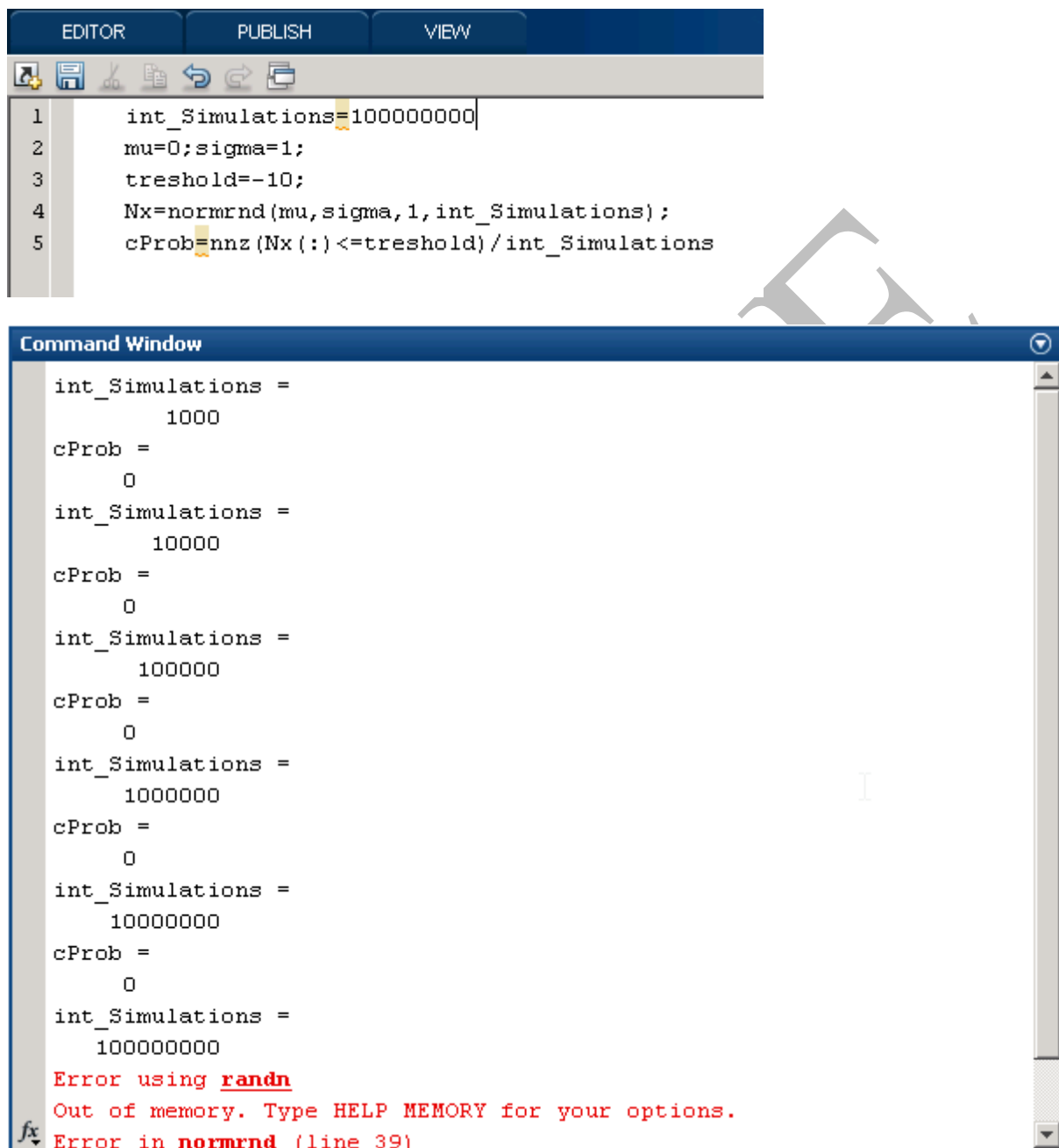
```



Now we want to check with MATLAB that:

$$P(X \leq -10) \cong 7.6199 \cdot 10^{-24}$$

We don't need a chart:



The image shows a MATLAB environment. The top part is the Editor window with a script containing five lines of code. The bottom part is the Command Window showing the execution of these lines, with the number of simulations increasing from 1000 to 10,000,000. The final two iterations result in an 'Out of memory' error.

```
1 int_Simulations=100000000
2 mu=0;sigma=1;
3 treshold=-10;
4 Nx=normrnd(mu,sigma,1,int_Simulations);
5 cProb=nnz(Nx(:)<=treshold)/int_Simulations
```

Command Window

```
int_Simulations =
    1000
cProb =
     0
int_Simulations =
   10000
cProb =
     0
int_Simulations =
  100000
cProb =
     0
int_Simulations =
 1000000
cProb =
     0
int_Simulations =
10000000
cProb =
     0
int_Simulations =
100000000
Error using randn
Out of memory. Type HELP MEMORY for your options.
Error in normrnd (line 39)
```

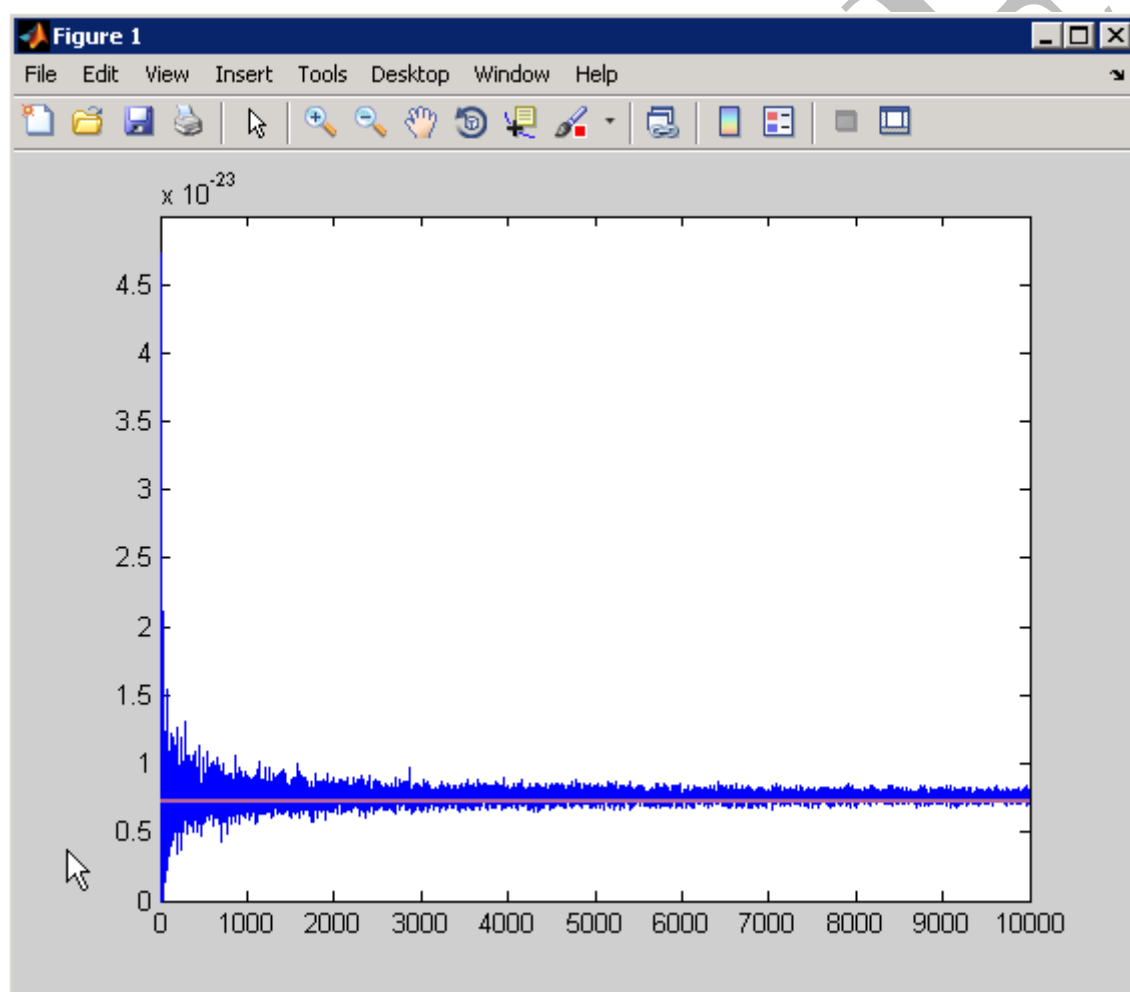
Hummm....

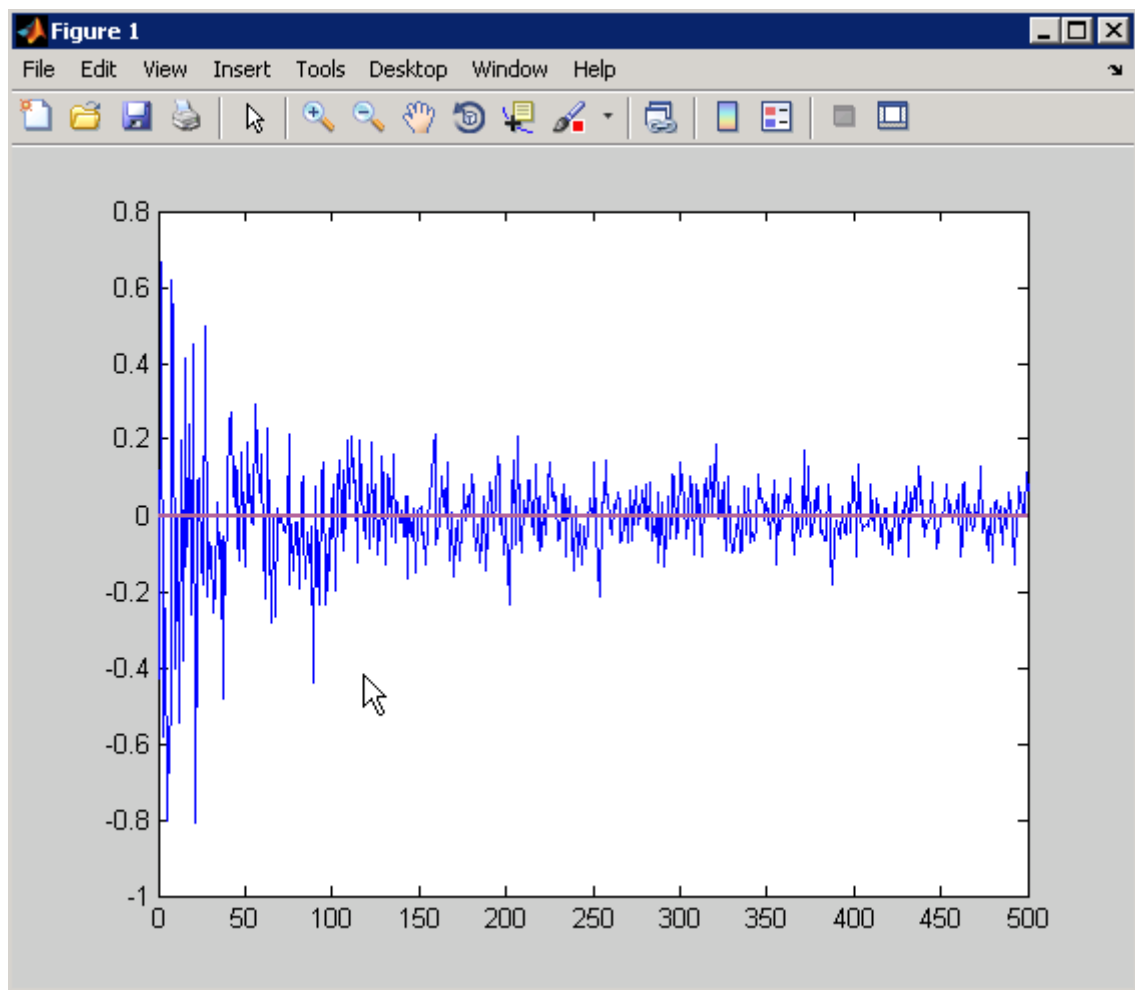
Now we use MC importance sampling technique:

```

EDITOR PUBLISH VIEW
1  int_Simulations=10000;
2  treshold=-10;
3  for k=1:int_Simulations
4      Y=randn(k,1)+treshold;
5      h=(Y<=treshold);
6      m(k)=mean(h.*exp((-treshold)*Y+0.5*(treshold)^2));
7  end
8
9  plot(1:int_Simulations,m)
10 hy=graph2d.constantline(7.1699e-24,'Color',[0.7 0.4 0.6],'LineWidth',2);
11 changedependvar(hy,'y');

```





Optimization (with Optimization Toolbox/Global Optimization Toolbox)

Here we will try to reproduce all examples we made with Microsoft Excel solver excepted those related to finance that are reserved to the finance chapter (se page 875).

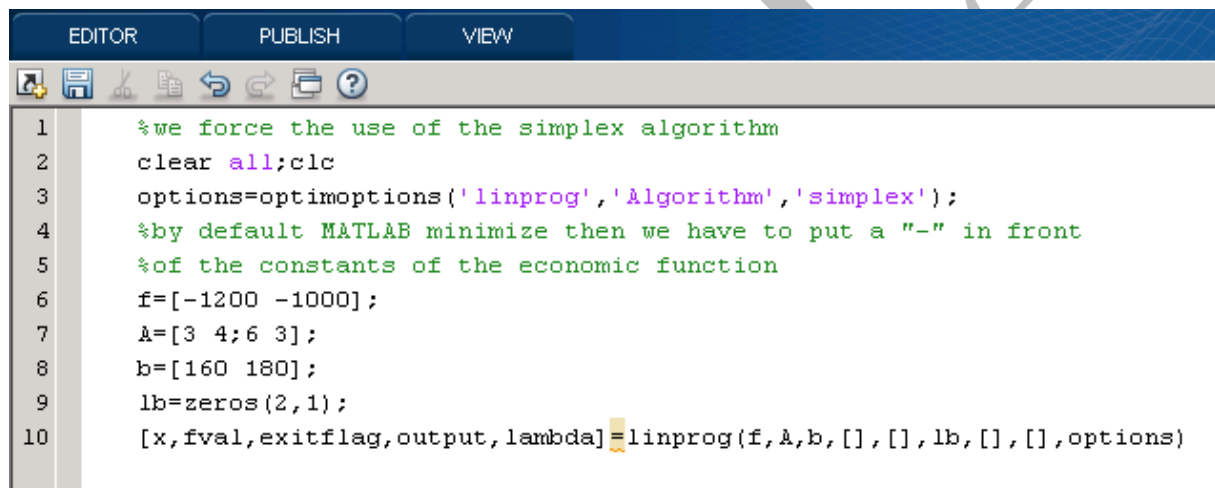
<http://www.mathworks.ch/ch/help/optim/release-notes.html>

<http://www.mathworks.ch/ch/help/gads/release-notes.html>

Linear optimization using LP Simplex (LPS)

All examples are taken from the theoretical training on Numerical Methods. I tried to order the examples in ascending order of complexity:

LP-S Factory optimization problem (**inequality linear system**)



```
EDITOR PUBLISH VIEW
1 %we force the use of the simplex algorithm
2 clear all;clc
3 options=optimoptions('linprog','Algorithm','simplex');
4 %by default MATLAB minimize then we have to put a "-" in front
5 %of the constants of the economic function
6 f=[-1200 -1000];
7 A=[3 4;6 3];
8 b=[160 180];
9 lb=zeros(2,1);
10 [x,fval,exitflag,output,lambda]=linprog(f,A,b,[],[],lb,[],[],options)
```

```
Command Window

Optimization terminated.

x =

    16
    28

fval =

   -47200

exitflag =

     1

output =

    iterations: 2
    algorithm: 'simplex'
    cgiterations: []
    message: 'Optimization terminated.'
    constrviolation: 0
    firstorderopt: 0

lambda =

    ineqlin: [2x1 double]
    eqlin: [0x1 double]
    upper: [2x1 double]
    lower: [2x1 double]
```

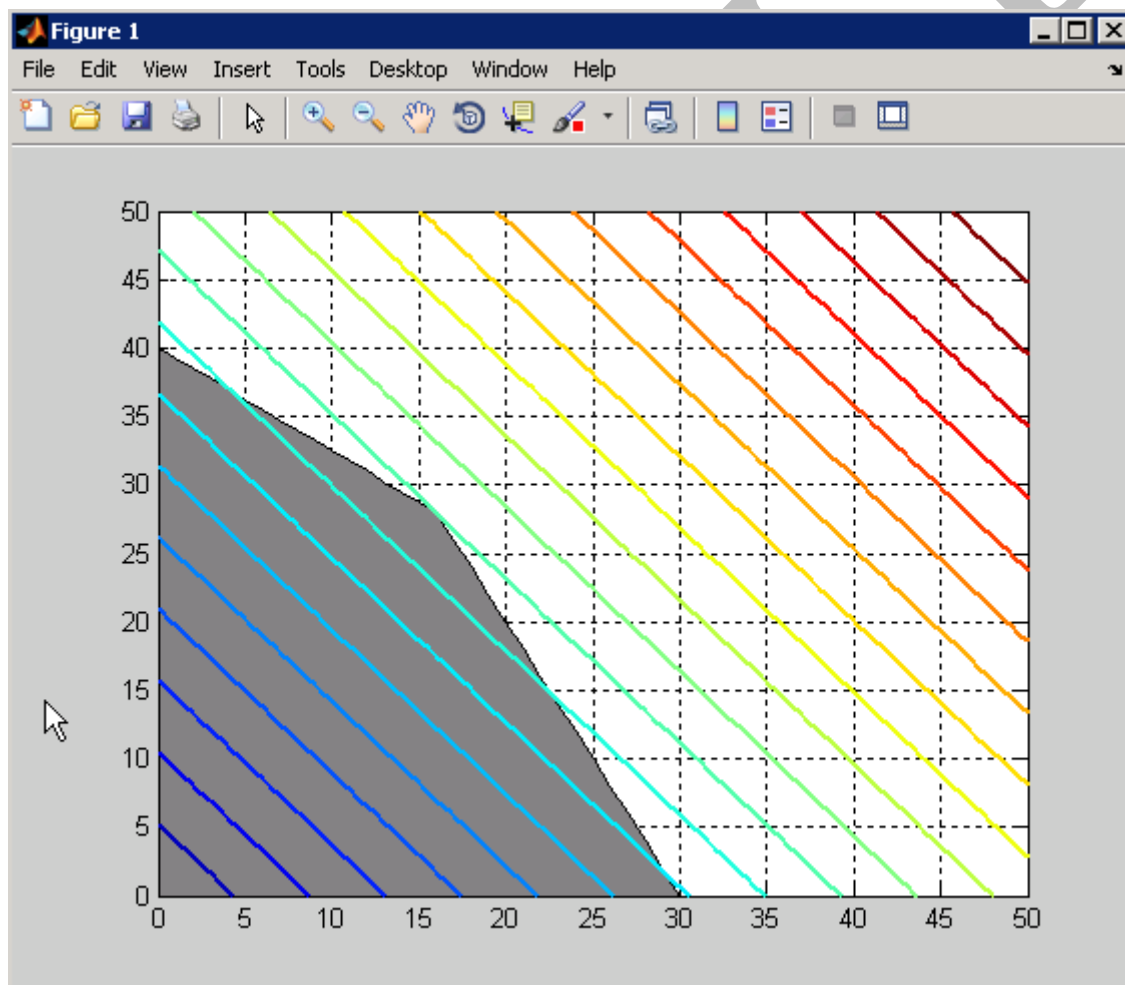
If exitflag is equal to -3 then the solver can not found a feasible solution!

And with the chart in bonus!

```

1      %not intuitive at all the build the polygon chart of constraints...
2      x=0:50; % range for chart
3      y1=max((180-6*x)/3,0);
4      y2=max((160-3*x)/4,0);
5      y3=max((180-6*x)/3,0);
6      ytop=min([y1;y2;y3]);
7      area(x,ytop,'FaceColor',[0.5 0.5 0.5]);
8
9      hold on;
10     [u, v]=meshgrid(0:50,0:50);
11     [c, h]=contour(u,v,1200*u+1000*v,20);
12     set(h,'linewidth',2);
13     grid on;

```



LP-S Naive (without covariance...) Markowitz parametric portfolio optimization problem (**equality linear system without target value**)

```
EDITOR PUBLISH VIEW
1 - clear all;clc
2 - v_Ret=[0.116 0.226 0.252];
3
4 - int_Assets=length(v_Ret);
5 - lb=zeros(1,int_Assets);
6 - ub=ones(1,int_Assets);
7 - Aeq=[ones(1,int_Assets)];
8 - Beq=[1];
9
10 %very naive first example...
11 %without taking the covariance matrix in consideration... O_o
12 - options=optimoptions('linprog','Algorithm','simplex')
13 - [TargetReturnWeights, ~, exitflag]=linprog(-v_Ret,[],[],Aeq,Beq,lb,ub,[],options)
14 - Return=v_Ret*TargetReturnWeights
```

Command Window

Optimization terminated.

TargetReturnWeights =

0
0
1

exitflag =

1

Return =

0.2520

f_x >> |

LP-S Naive (without covariance...) Markowitz parametric portfolio optimization problem (**equality linear system with target value**)

```
EDITOR PUBLISH VIEW
1 - clear all; clc
2 - v_Ret=[0.116 0.226 0.252];
3
4 - int_Assets=length(v_Ret);
5 - lb=zeros(1,int_Assets);
6 - ub=ones(1,int_Assets);
7 - TargetReturn=0.20; %when greater than 0.252 they will be logically no feasible solution
8 - Aeq=[v_Ret;ones(1,int_Assets)];
9 - Beq=[TargetReturn;1];
10
11 %very naive first example...
12 %without taking the covariance matrix in consideration... O_o
13 - options=optimoptions('linprog','Algorithm','simplex')
14 - [TargetReturnWeights, ~, exitflag]=linprog(-v_Ret,[],[],Aeq,Beq,lb,ub,[],options)
15 - Return=v_Ret*TargetReturnWeights
```

```
Command Window

Optimization terminated.

TargetReturnWeights =

    0.3824
         0
    0.6176

exitflag =

         1

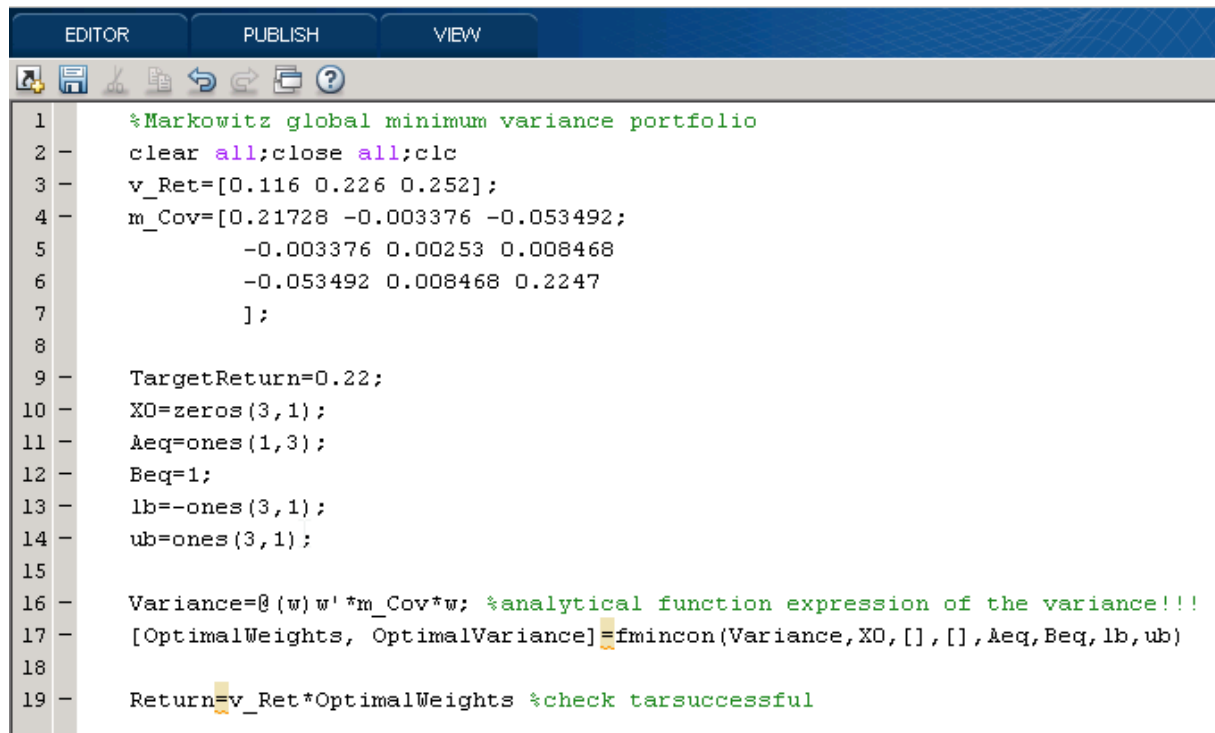
Return =

    0.2000
```

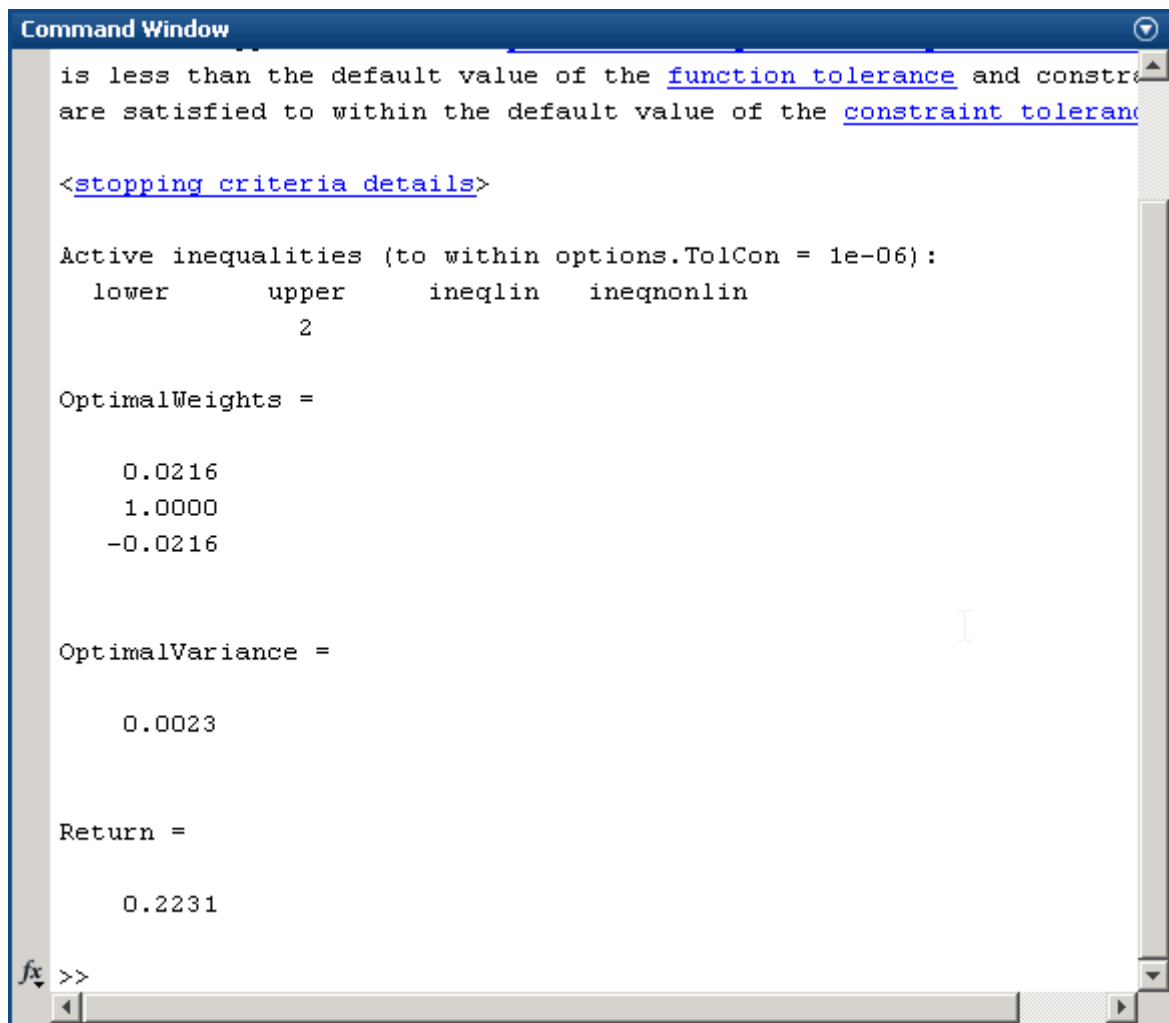
More interesting version including covariance will come soon

Function constrained optimization (FCO)

FCO with constraints and without specific target



```
1 %Markowitz global minimum variance portfolio
2 clear all;close all;clc
3 v_Ret=[0.116 0.226 0.252];
4 m_Cov=[0.21728 -0.003376 -0.053492;
5         -0.003376 0.00253 0.008468
6         -0.053492 0.008468 0.2247
7         ];
8
9 TargetReturn=0.22;
10 X0=zeros(3,1);
11 Aeq=ones(1,3);
12 Beq=1;
13 lb=-ones(3,1);
14 ub=ones(3,1);
15
16 Variance=@(w)w'*m_Cov*w; %analytical function expression of the variance!!!
17 [OptimalWeights, OptimalVariance]=fmincon(Variance,X0,[],[],Aeq,Beq,lb,ub)
18
19 Return=v_Ret*OptimalWeights %check tarsuccessful
```

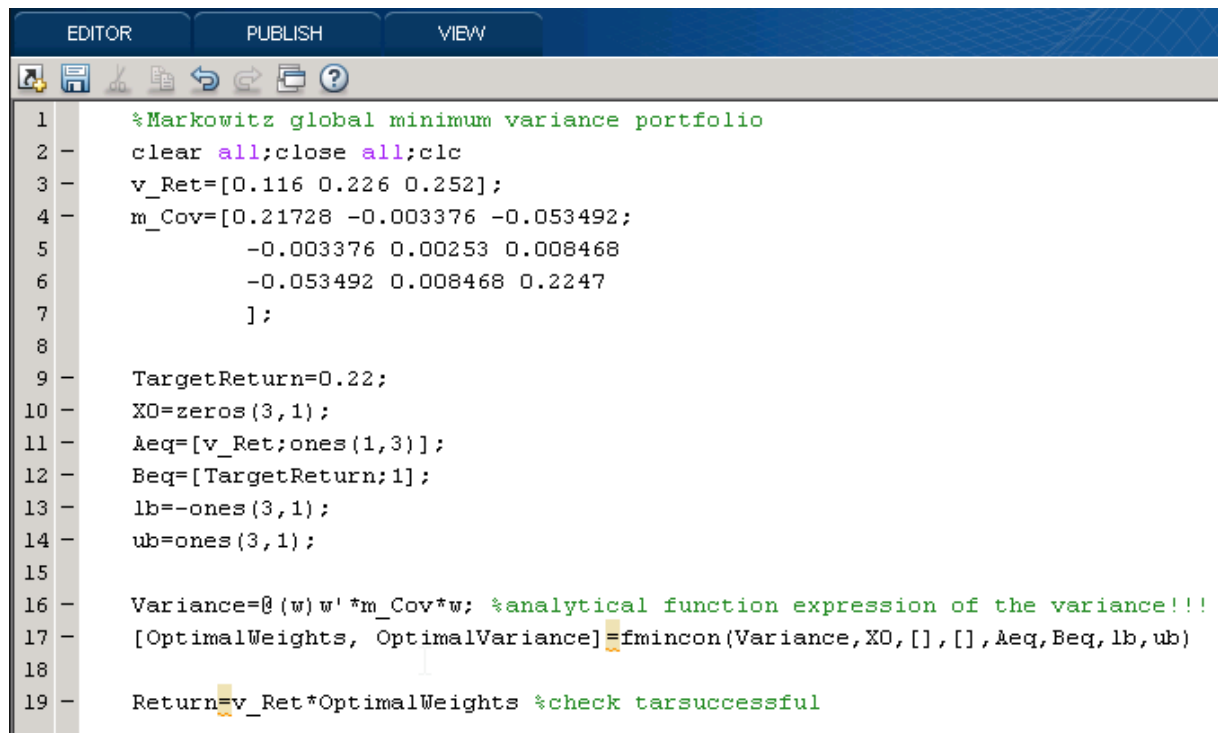


The screenshot shows the MATLAB Command Window with the following text:

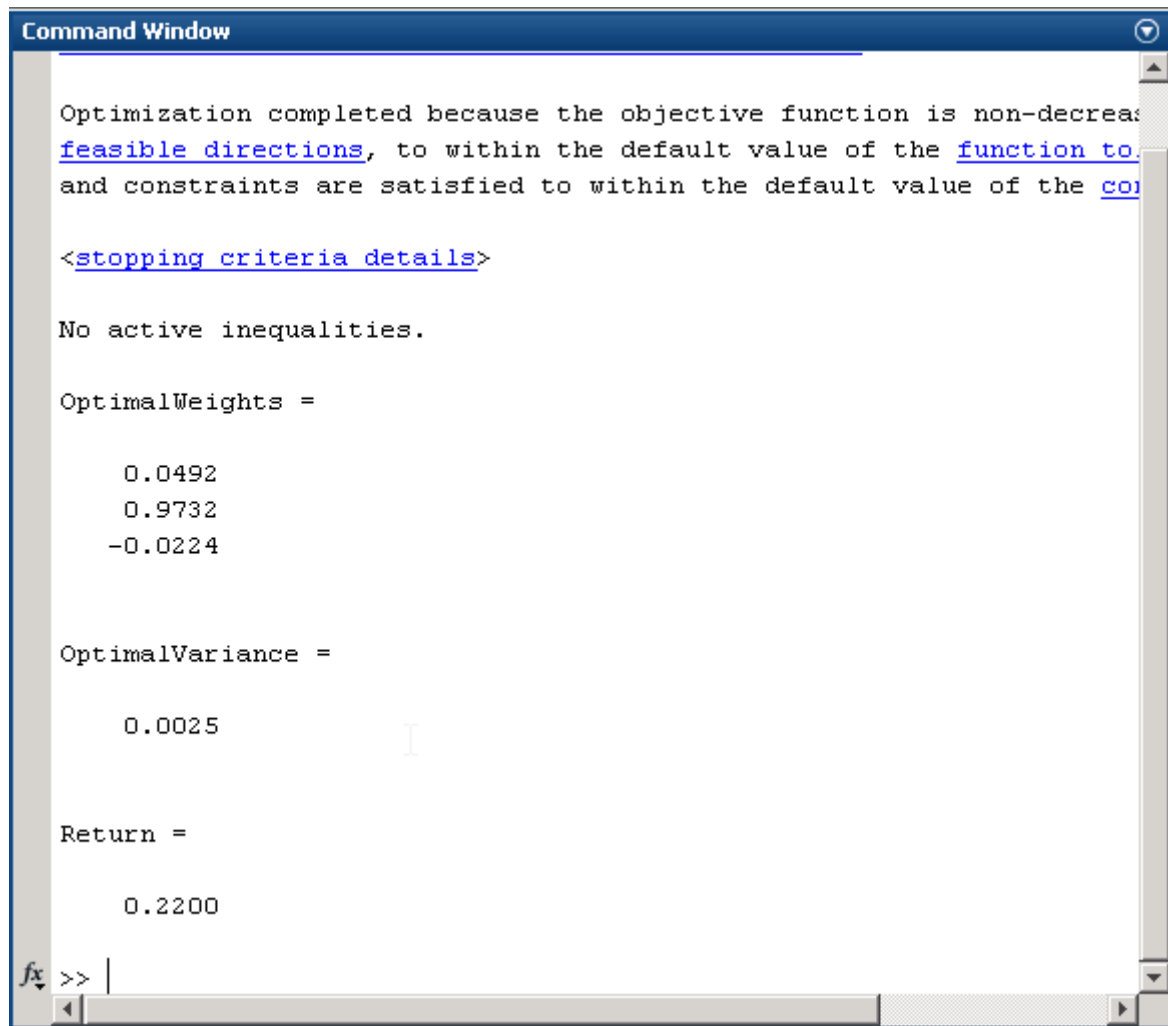
```
is less than the default value of the function tolerance and constraints  
are satisfied to within the default value of the constraint tolerance.  
  
<stopping criteria details>  
  
Active inequalities (to within options.TolCon = 1e-06):  
    lower      upper      ineqlin      ineqnonlin  
         2  
  
OptimalWeights =  
  
    0.0216  
    1.0000  
   -0.0216  
  
OptimalVariance =  
  
    0.0023  
  
Return =  
  
    0.2231  
  
fx >>
```

A large, light gray watermark with the word "SAMPLE" is oriented diagonally across the lower half of the page.

FCO with constraints and specific target



```
1 %Markowitz global minimum variance portfolio
2 clear all;close all;clc
3 v_Ret=[0.116 0.226 0.252];
4 m_Cov=[0.21728 -0.003376 -0.053492;
5         -0.003376 0.00253 0.008468
6         -0.053492 0.008468 0.2247
7         ];
8
9 TargetReturn=0.22;
10 XO=zeros(3,1);
11 Aeq=[v_Ret;ones(1,3)];
12 Beq=[TargetReturn;1];
13 lb=-ones(3,1);
14 ub=ones(3,1);
15
16 Variance=@(w) w'*m_Cov*w; %analytical function expression of the variance!!!
17 [OptimalWeights, OptimalVariance]=fmincon(Variance,XO,[],[],Aeq,Beq,lb,ub)
18
19 Return=v_Ret*OptimalWeights %check tarsuccessful
```



```

Command Window

Optimization completed because the objective function is non-decreasing
feasible directions, to within the default value of the function tolerance,
and constraints are satisfied to within the default value of the constraint
tolerance.

<stopping criteria details>

No active inequalities.

OptimalWeights =

    0.0492
    0.9732
   -0.0224

OptimalVariance =

    0.0025

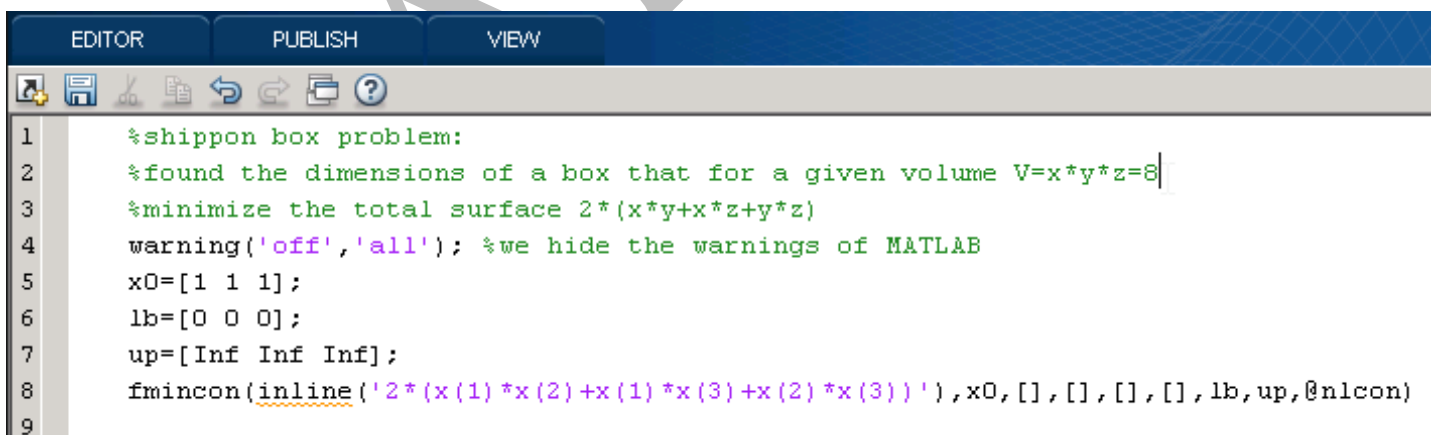
Return =

    0.2200

fx >>

```

FCO with non linear constraints (and specific target)

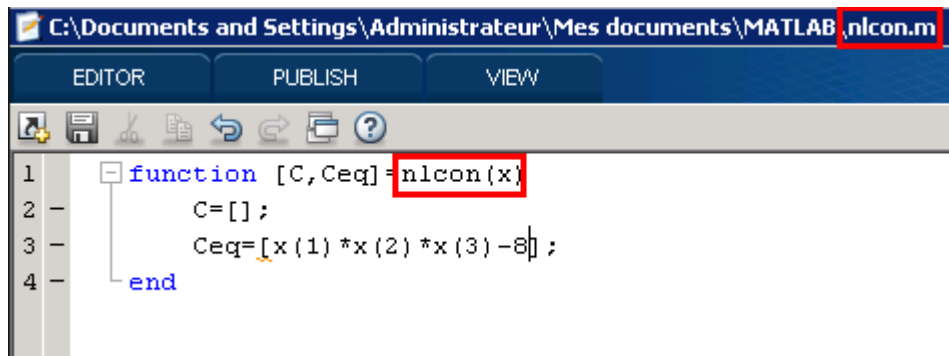


```

EDITOR PUBLISH VIEW

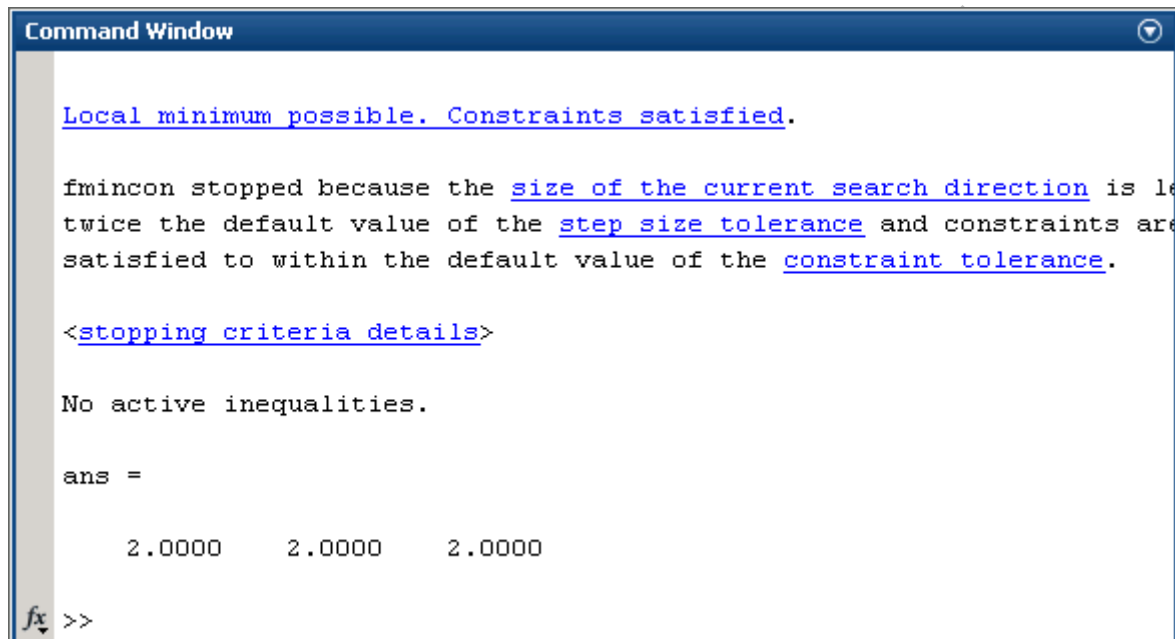
1 %shippon box problem:
2 %found the dimensions of a box that for a given volume V=x*y*z=8
3 %minimize the total surface 2*(x*y+x*z+y*z)
4 warning('off','all'); %we hide the warnings of MATLAB
5 x0=[1 1 1];
6 lb=[0 0 0];
7 up=[Inf Inf Inf];
8 fmincon(inline('2*(x(1)*x(2)+x(1)*x(3)+x(2)*x(3))'),x0,[],[],[],[],lb,up,@nlcon)
9

```



The image shows the MATLAB Editor window with the file path `C:\Documents and Settings\Administrateur\Mes documents\MATLAB\nlcon.m` in the title bar. The editor contains the following code:

```
1 function [C,Ceq]=nlcon(x)
2     C=[];
3     Ceq=[x(1)*x(2)*x(3)-8];
4 end
```



The image shows the MATLAB Command Window with the following output:

```
Local minimum possible. Constraints satisfied.

fmincon stopped because the size of the current search direction is less than
twice the default value of the step size tolerance and constraints are
satisfied to within the default value of the constraint tolerance.

<stopping criteria details>

No active inequalities.

ans =

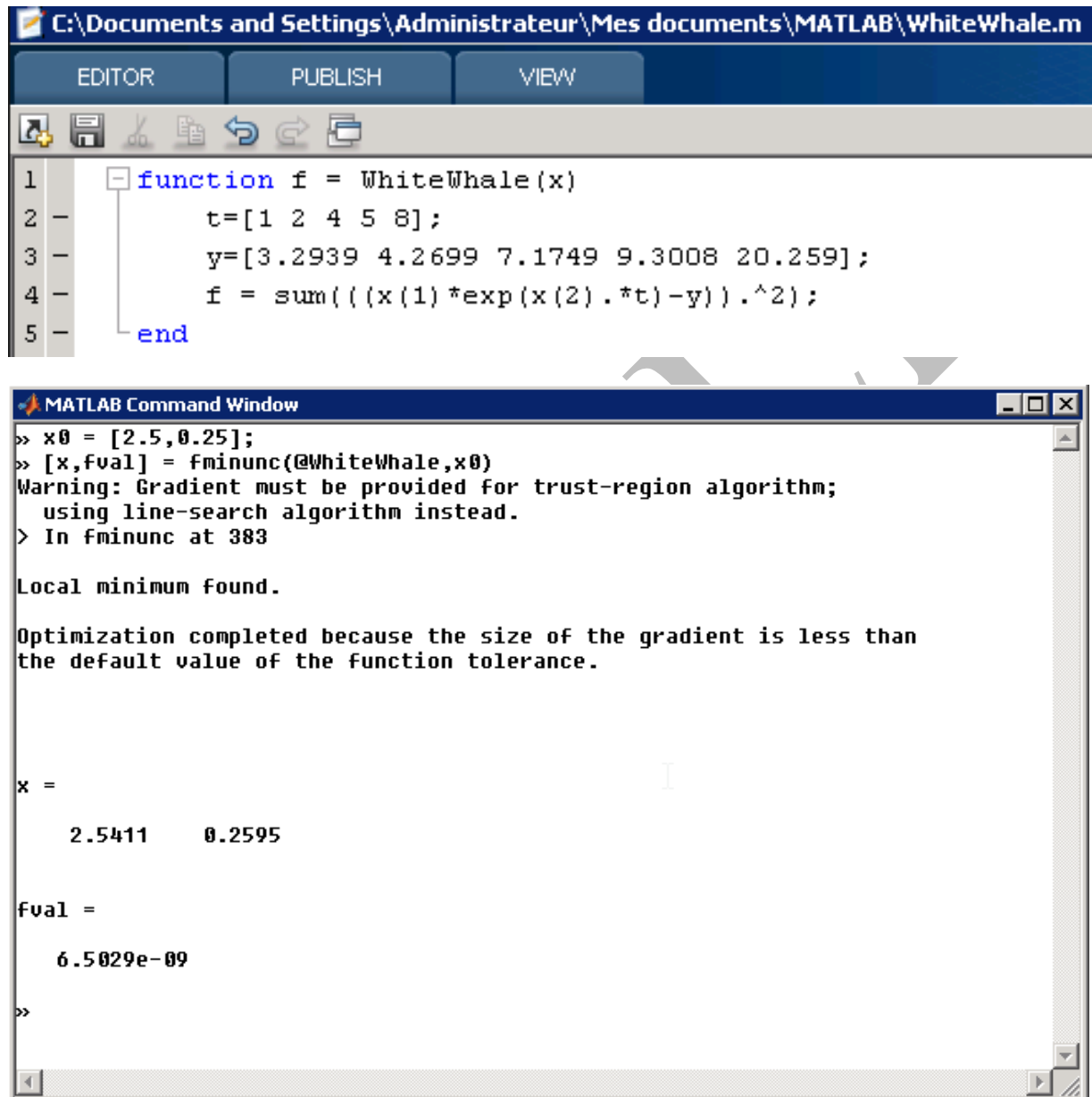
    2.0000    2.0000    2.0000

fx >>
```

Quadratic optimization (QO)

QO using conjugate gradient method (Q-CG)

This is also called "non-constrained" or "unconstrained" optimization problem.



The image shows two MATLAB windows. The top window is the MATLAB Editor, displaying a function named 'WhiteWhale' in a file 'WhiteWhale.m'. The function calculates the sum of squared residuals for a set of data points. The bottom window is the MATLAB Command Window, showing the execution of the 'fminunc' function to find the minimum of the 'WhiteWhale' function. The output shows the optimal parameters 'x' and the function value 'fval'.

```

C:\Documents and Settings\Administrateur\Mes documents\MATLAB\WhiteWhale.m
EDITOR PUBLISH VIEW

1 function f = WhiteWhale(x)
2     t=[1 2 4 5 8];
3     y=[3.2939 4.2699 7.1749 9.3008 20.259];
4     f = sum((x(1)*exp(x(2).*t)-y).^2);
5 end

MATLAB Command Window
>> x0 = [2.5,0.25];
>> [x,fval] = fminunc(@WhiteWhale,x0)
Warning: Gradient must be provided for trust-region algorithm;
        using line-search algorithm instead.
> In fminunc at 383

Local minimum found.

Optimization completed because the size of the gradient is less than
the default value of the function tolerance.

x =
    2.5411    0.2595

fval =
    6.5029e-09

>>
  
```

Where **fminunc()** is for "unc"-constrained.

Or be explicating the Hessian or Gradient:

```

EDITOR PUBLISH VIEW
1 x0=[2.5,0.25]
2 options = optimset('GradObj','on','Hessian','on');
3 [x,fval] = fminunc(@WhiteWhale,x0,options)

```

```

EDITOR PUBLISH VIEW
1 function [f g H] = WhiteWhale(x)
2 - t=[1 2 4 5 8];
3 - y=[3.2939 4.2699 7.1749 9.3008 20.259];
4 - f = sum((x(1)*exp(x(2).*t)-y).^2);
5 - if nargin > 1 % gradient required
6 -     g = 0.5*[sum(2*(x(1)*exp(x(2).*t)-y).*exp(x(2).*t));
7 -         sum(2*(x(1)*exp(x(2).*t)-y).*x(1).*t.*exp(x(2).*t))];
8 - end
9 - if nargin > 2 % Hessian required
10 -     H = [sum(exp(2*x(2).*t)), sum(2*x(1).*t.*exp(x(2).*t)-y.*t.*exp(x(2).*t));
11 -         sum(2*x(1).*t.*exp(x(2).*t)-y.*t.*exp(x(2).*t)),...
12 -         sum(2*x(1)^2.*t.*exp(2*x(2).*t)-x(1)*y.*t.^2.*exp(x(2).*t))];
13 - end
14 - end

```

Command Window

x0 =

2.5000 0.2500

Solver stopped prematurely.

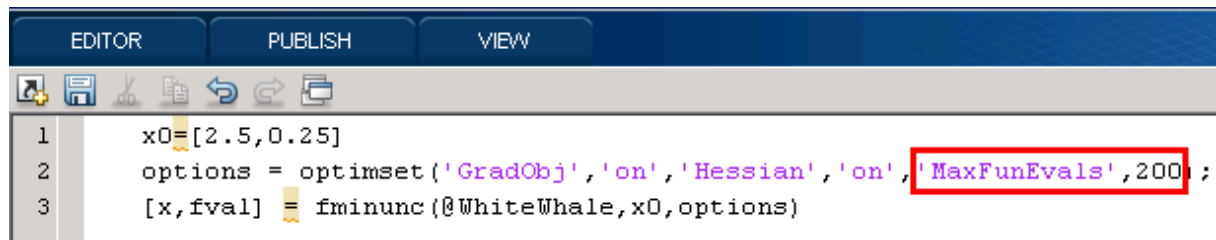
fminunc stopped because it exceeded the function evaluation limit,
options.MaxFunEvals = 200 (the default value).

x =

2.5019 0.2616

fval =

0.0097

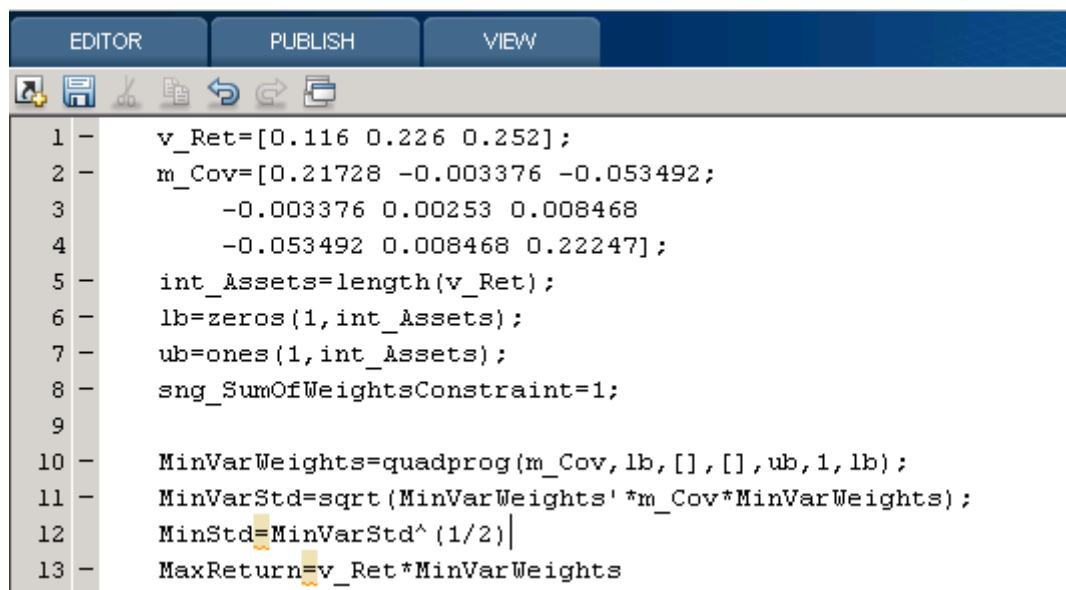


```

1 x0=[2.5,0.25]
2 options = optimset('GradObj','on','Hessian','on','MaxFunEvals',200);
3 [x,fval] = fminunc(@WhiteWhale,x0,options)

```

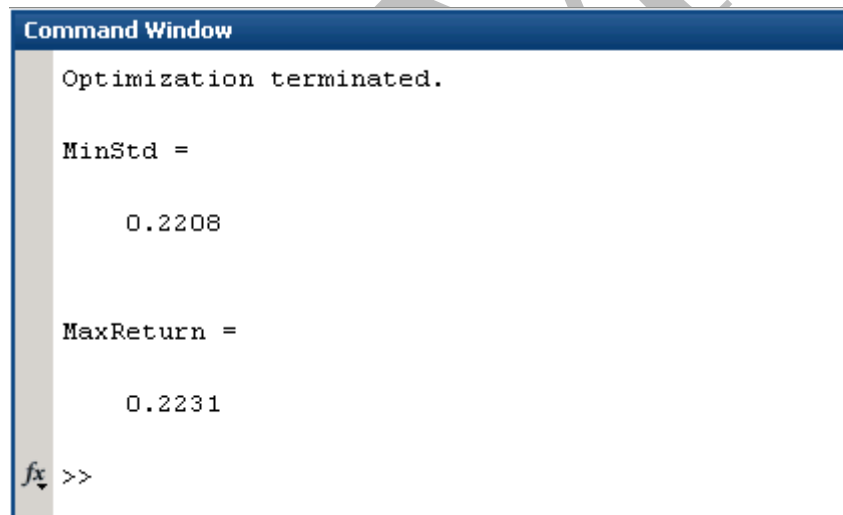
QO using interior point convex method (Q-IPC)



```

1 - v_Ret=[0.116 0.226 0.252];
2 - m_Cov=[0.21728 -0.003376 -0.053492;
3 -       -0.003376 0.00253 0.008468
4 -       -0.053492 0.008468 0.22247];
5 - int_Assets=length(v_Ret);
6 - lb=zeros(1,int_Assets);
7 - ub=ones(1,int_Assets);
8 - sng_SumOfWeightsConstraint=1;
9
10 - MinVarWeights=quadprog(m_Cov,lb,[],[],ub,1,lb);
11 - MinVarStd=sqrt(MinVarWeights'*m_Cov*MinVarWeights);
12 - MinStd=MinVarStd^(1/2);
13 - MaxReturn=v_Ret*MinVarWeights

```



```

Command Window

Optimization terminated.

MinStd =

    0.2208

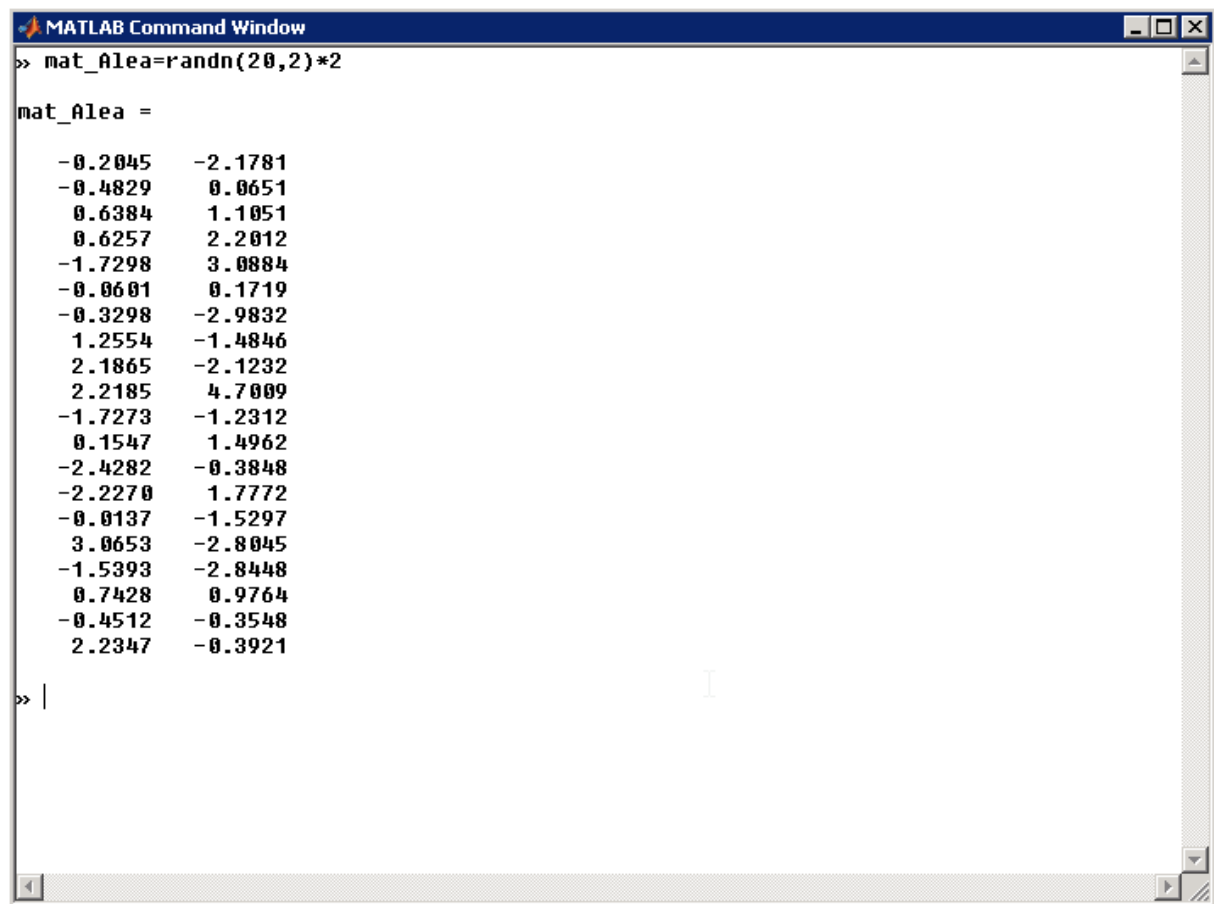
MaxReturn =

    0.2231

fx >>

```


Genetic algorithms



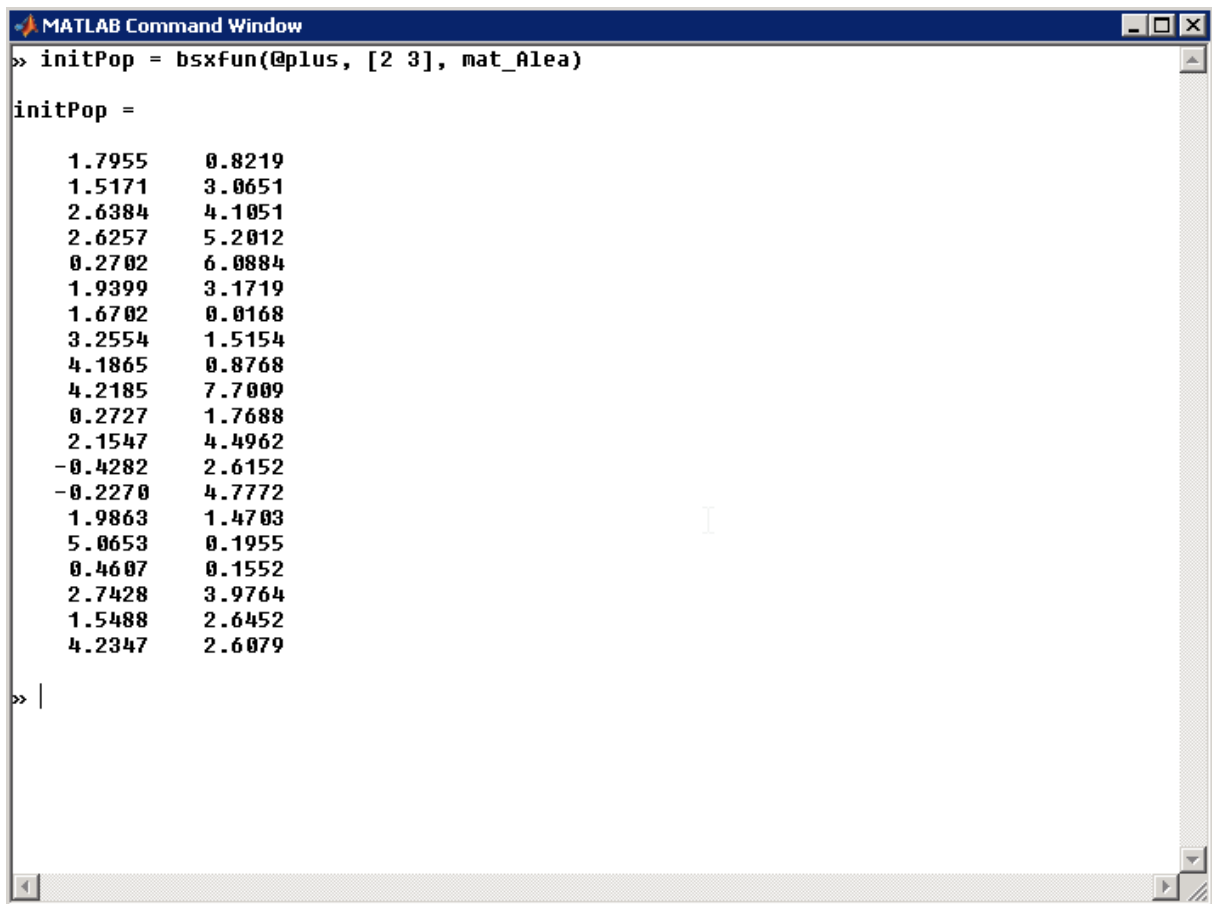
A screenshot of the MATLAB Command Window. The title bar reads "MATLAB Command Window". The command prompt shows the command `>> mat_Alea=randn(20,2)*2`. Below the command, the output is displayed as `mat_Alea =` followed by a 20x2 matrix of random numbers. The numbers are arranged in two columns, with the first column values ranging from approximately -0.2045 to 2.2347 and the second column values ranging from approximately -0.3921 to 4.7009. The window has standard MATLAB window controls (minimize, maximize, close) in the top right corner and a scroll bar on the right side.

```
>> mat_Alea=randn(20,2)*2

mat_Alea =

    -0.2045    -2.1781
    -0.4829     0.0651
     0.6384     1.1051
     0.6257     2.2012
    -1.7298     3.0884
    -0.0601     0.1719
    -0.3298    -2.9832
     1.2554    -1.4846
     2.1865    -2.1232
     2.2185     4.7009
    -1.7273    -1.2312
     0.1547     1.4962
    -2.4282    -0.3848
    -2.2270     1.7772
    -0.0137    -1.5297
     3.0653    -2.8045
    -1.5393    -2.8448
     0.7428     0.9764
    -0.4512    -0.3548
     2.2347    -0.3921

>> |
```



```

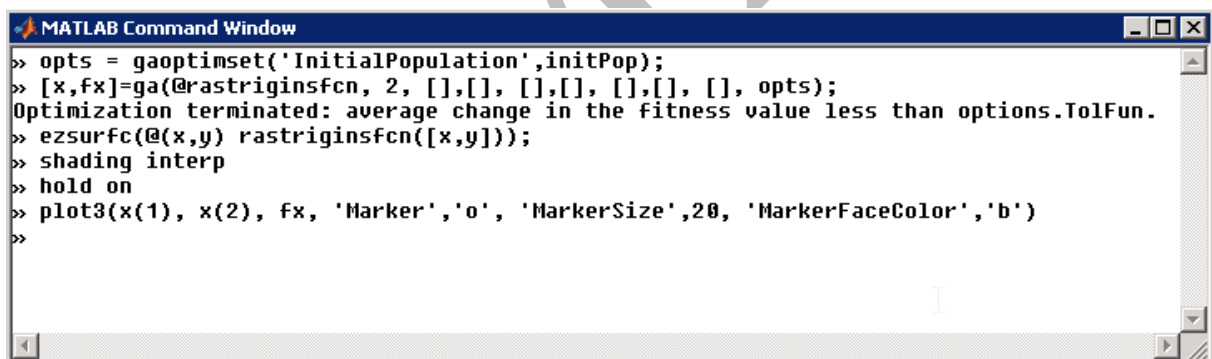
MATLAB Command Window
>> initPop = bsxfun(@plus, [2 3], mat_Alea)

initPop =

    1.7955    0.8219
    1.5171    3.0651
    2.6384    4.1051
    2.6257    5.2012
    0.2702    6.0884
    1.9399    3.1719
    1.6702    0.0168
    3.2554    1.5154
    4.1865    0.8768
    4.2185    7.7009
    0.2727    1.7688
    2.1547    4.4962
   -0.4282    2.6152
   -0.2270    4.7772
    1.9863    1.4703
    5.0653    0.1955
    0.4607    0.1552
    2.7428    3.9764
    1.5488    2.6452
    4.2347    2.6079

>> |

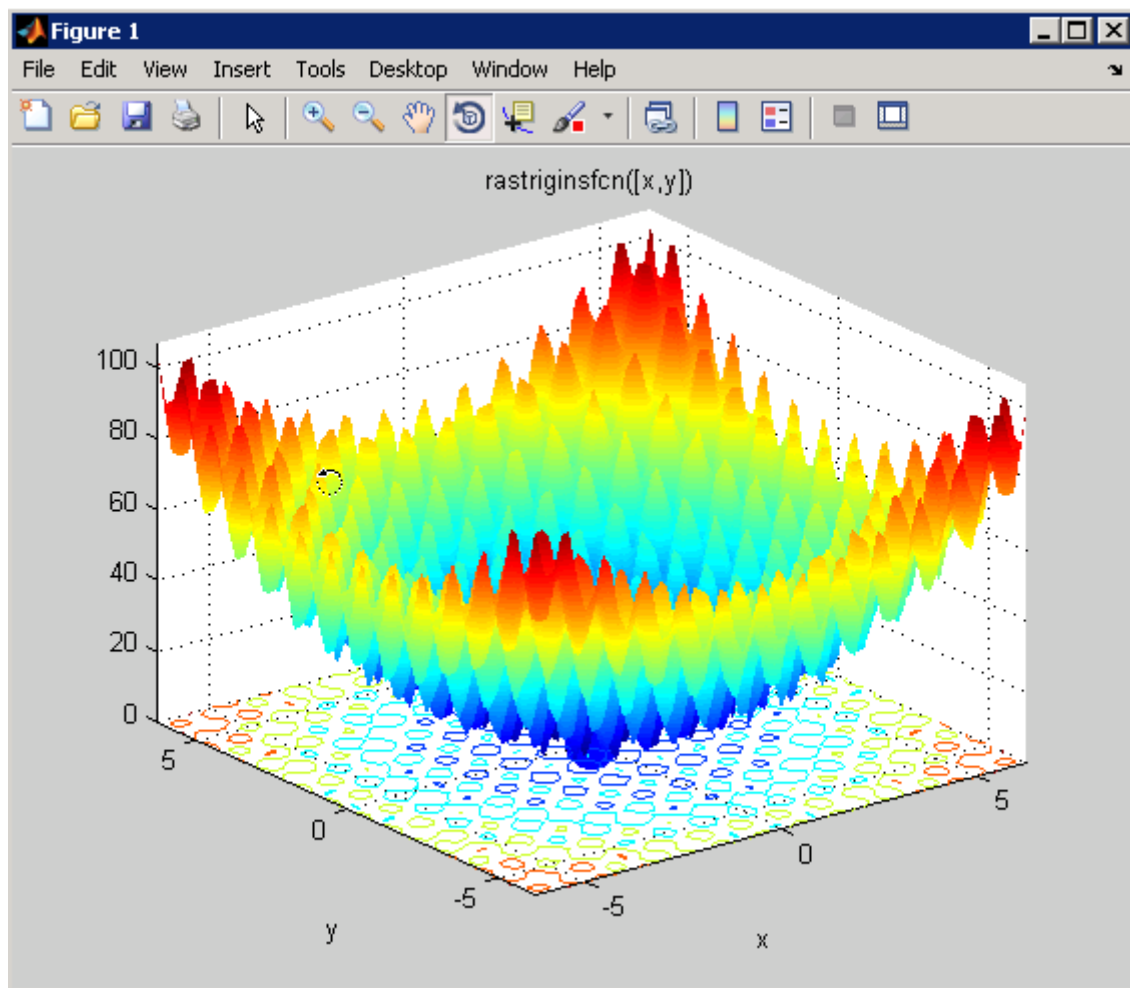
```

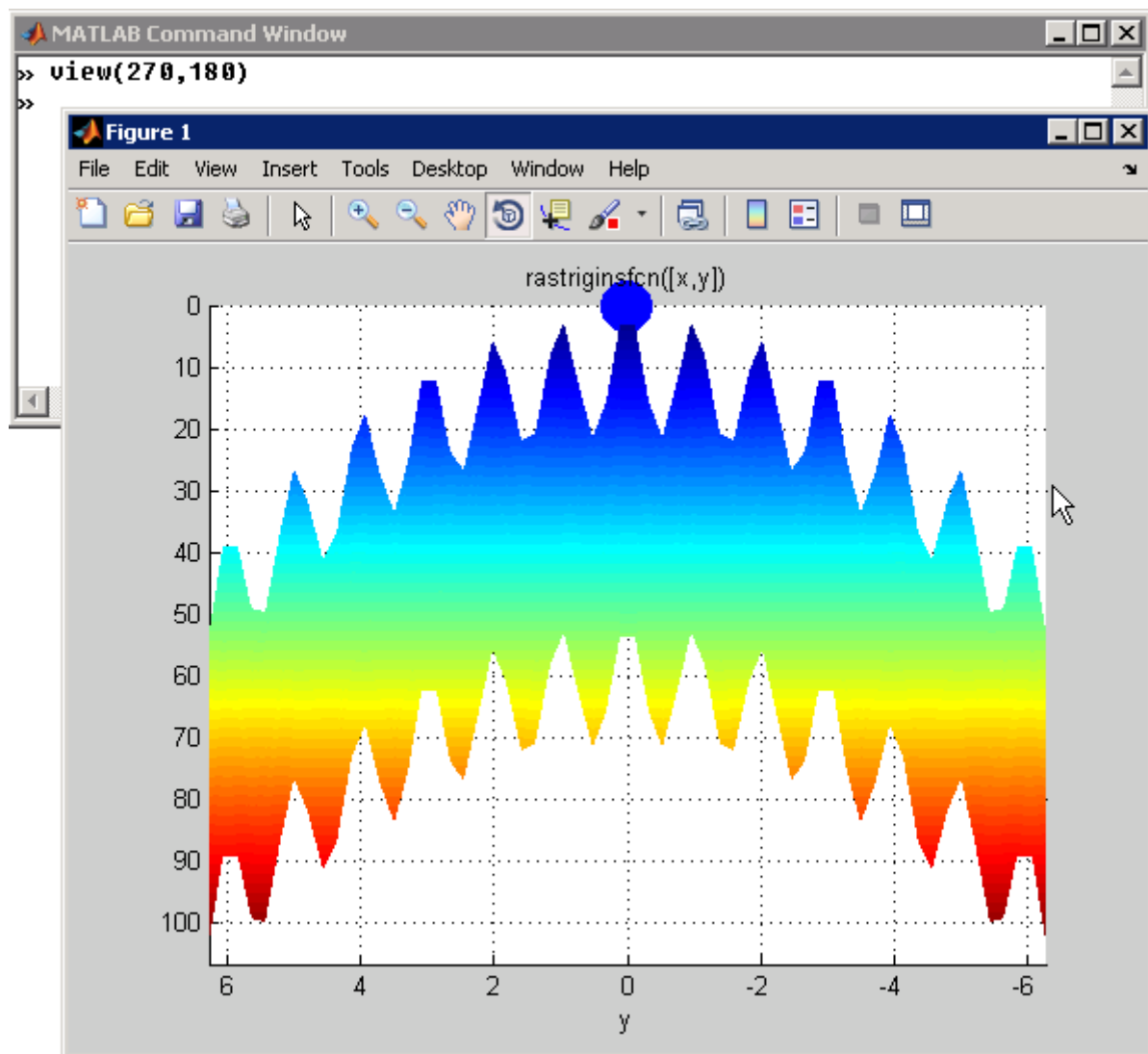


```

MATLAB Command Window
>> opts = gaoptimset('InitialPopulation',initPop);
>> [x,fx]=ga(@rastriginsfcn, 2, [],[], [],[], [],[], [], opts);
Optimization terminated: average change in the fitness value less than options.TolFun.
>> ezsurfc(@(x,y) rastriginsfcn([x,y]));
>> shading interp
>> hold on
>> plot3(x(1), x(2), fx, 'Marker','o', 'MarkerSize',20, 'MarkerFaceColor','b')
>>

```



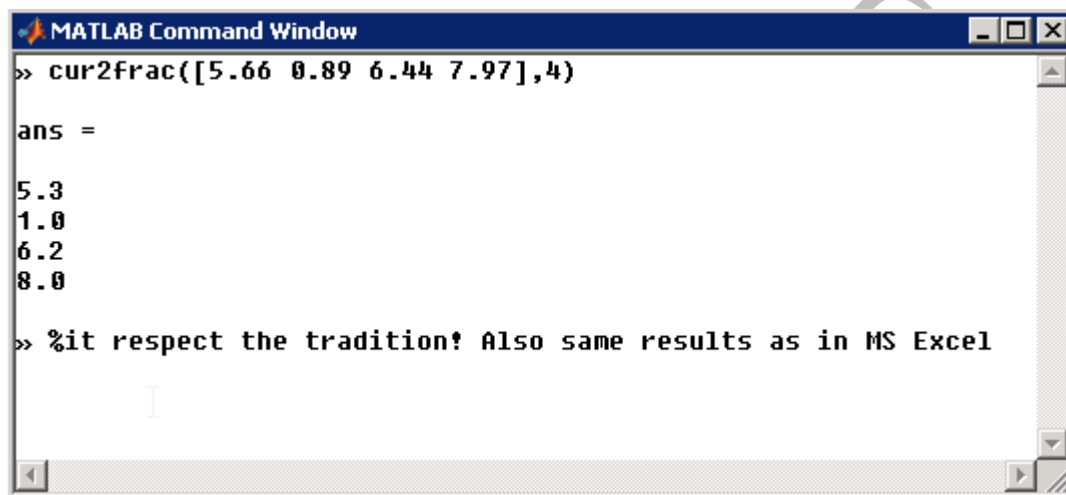


Finance (with Finance/Optimization and Econometrics Toolboxes)

Here we will just focus on the basic of the: Finance Toolbox, Statistics Toolbox, Optimization Toolbox, Trading Toolbox, Financial Instruments and Econometrics Toolboxes.

<http://www.mathworks.ch/ch/help/finance/release-notes.html>

Fractional prices



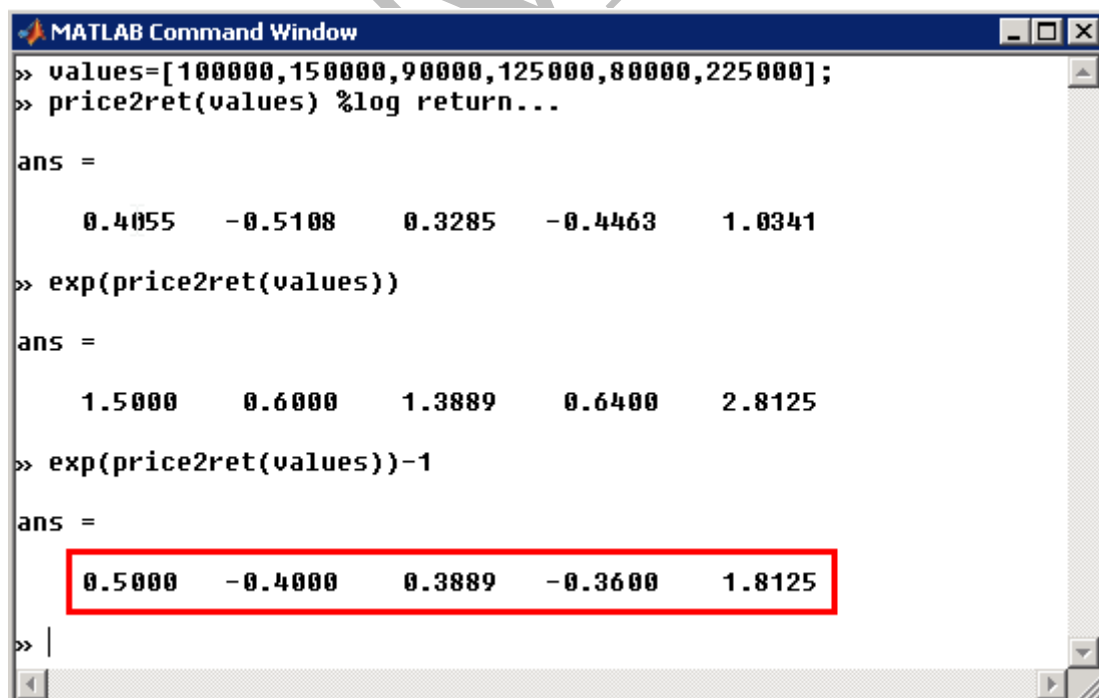
```
MATLAB Command Window
>> cur2frac([5.66 0.89 6.44 7.97],4)

ans =

5.3
1.0
6.2
8.0

>> %it respect the tradition! Also same results as in MS Excel
```

Return



```
MATLAB Command Window
>> values=[100000,150000,90000,125000,80000,225000];
>> price2ret(values) %log return...

ans =

    0.4055    -0.5108     0.3285    -0.4463     1.0341

>> exp(price2ret(values))

ans =

    1.5000     0.6000     1.3889     0.6400     2.8125

>> exp(price2ret(values))-1

ans =

    0.5000    -0.4000     0.3889    -0.3600     1.8125

>> |
```

The last line corresponds to R!

Maximum Draw Down

No comment about this indicator... because there exists multiple definitions...

```
Command Window
>> values=[100000 150000 90000 125000 80000 225000];
>> MDD=maxdrawdown(values)

MDD =

    0.4667

>> 150000 %the value when the first drawdown will appear

ans =

    150000

>> 80000 %the last value befor a drawup appears

ans =

    80000

>> MDD=(150000-80000)/150000

MDD =

    0.4667

>> maxdrawdown(values,'arithmetic')

ans =

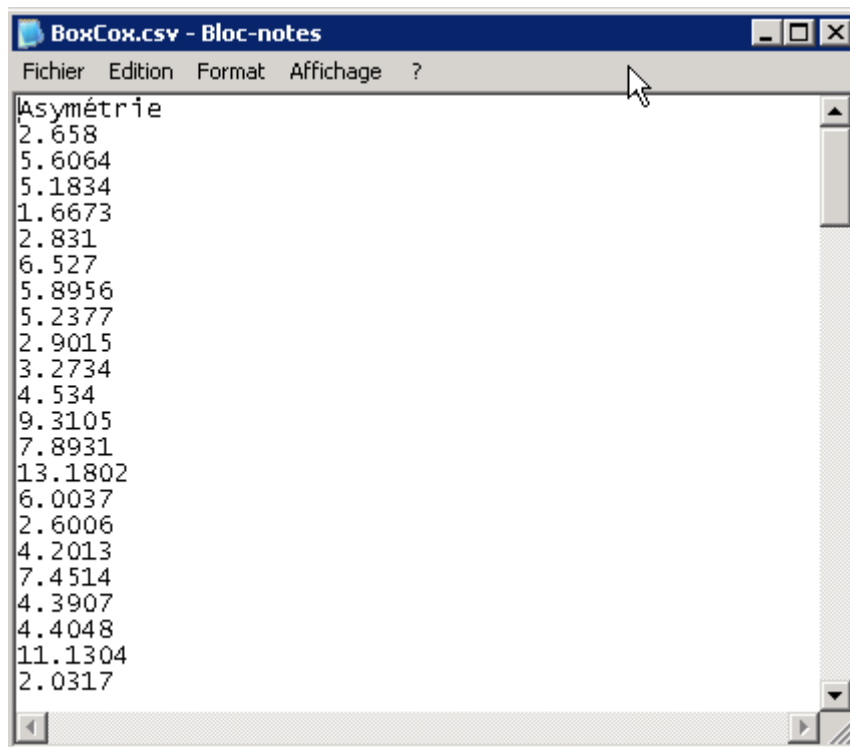
    70000

fx >>
```

In both cases we obtain the same values as in R!

Box-Cox transformation

Caution!! Remember that if you have negative data Box-Cox transformation won't work. You will have to shift your data before and after shift the result back again.

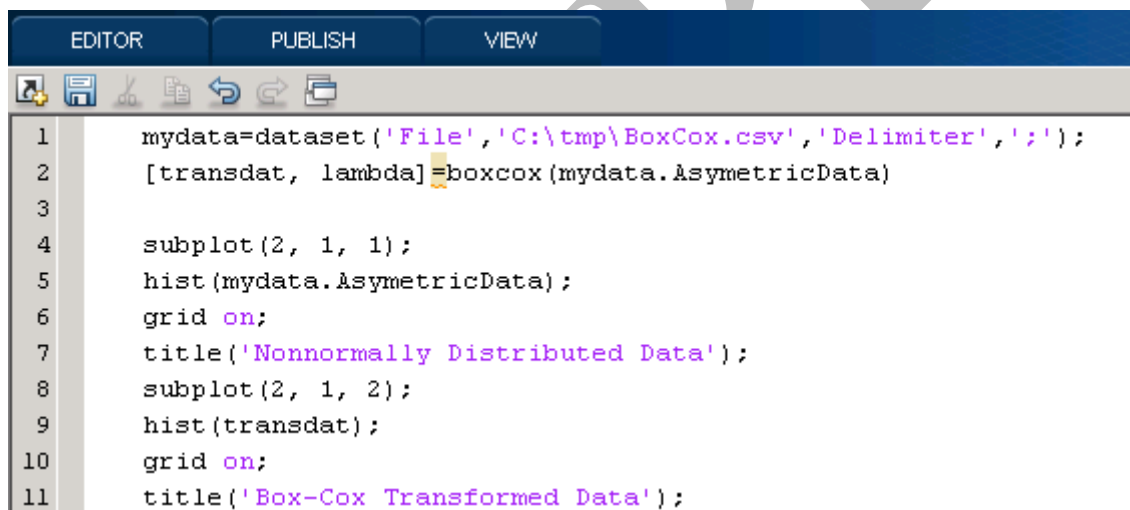


BoxCox.csv - Bloc-notes

Fichier Edition Format Affichage ?

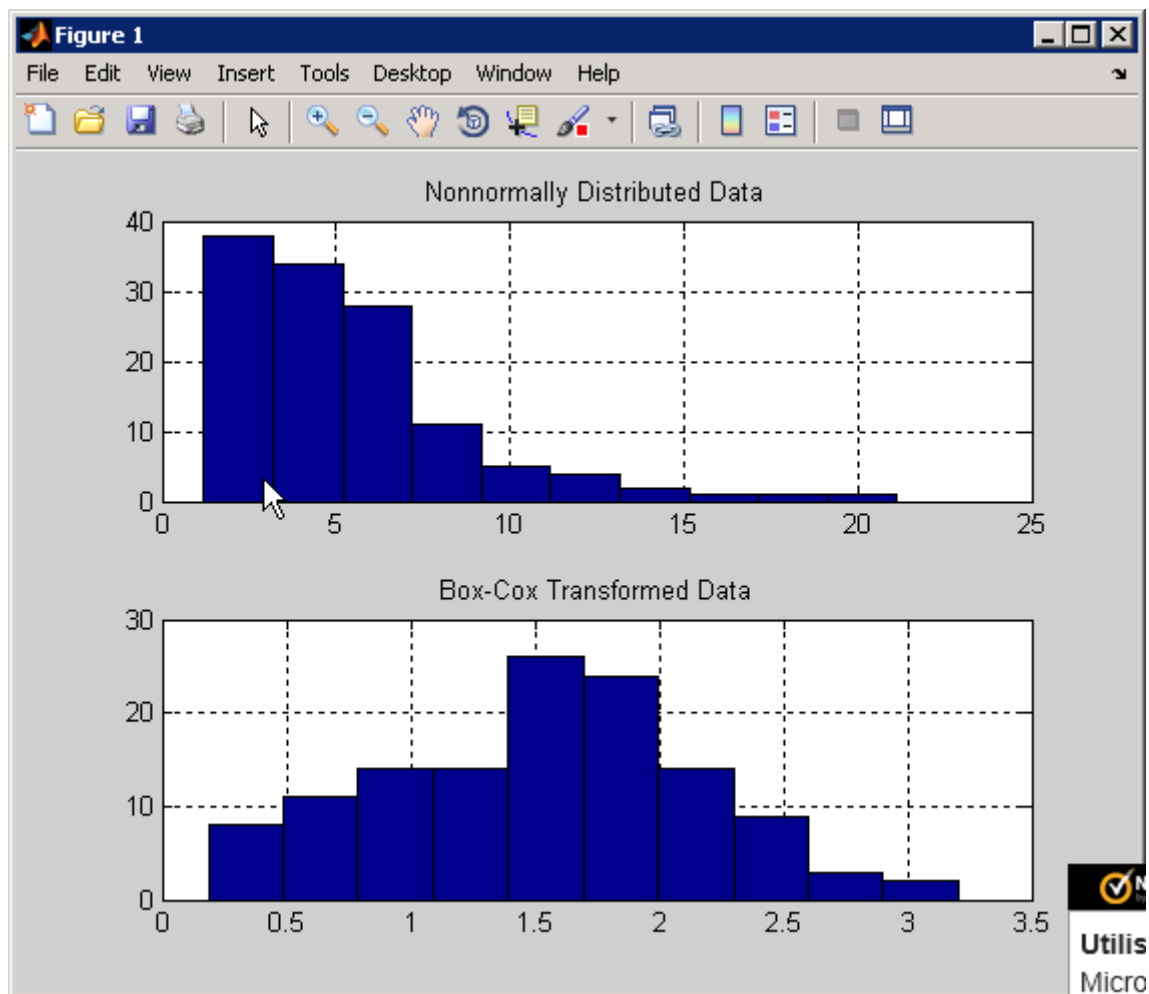
Asymétrie

2.658
5.6064
5.1834
1.6673
2.831
6.527
5.8956
5.2377
2.9015
3.2734
4.534
9.3105
7.8931
13.1802
6.0037
2.6006
4.2013
7.4514
4.3907
4.4048
11.1304
2.0317



EDITOR PUBLISH VIEW

```
1 mydata=dataset('File','C:\tmp\BoxCox.csv','Delimiter',';');
2 [transdat, lambda]=boxcox(mydata.AsymmetricData)
3
4 subplot(2, 1, 1);
5 hist(mydata.AsymmetricData);
6 grid on;
7 title('Nonnormally Distributed Data');
8 subplot(2, 1, 2);
9 hist(transdat);
10 grid on;
11 title('Box-Cox Transformed Data');
```



Simple capital calculations

Composed Interest (fixed future value)

```

MATLAB Command Window
>> fuFix(0.09/12, 12*10, 0, 1500, 0)

ans =

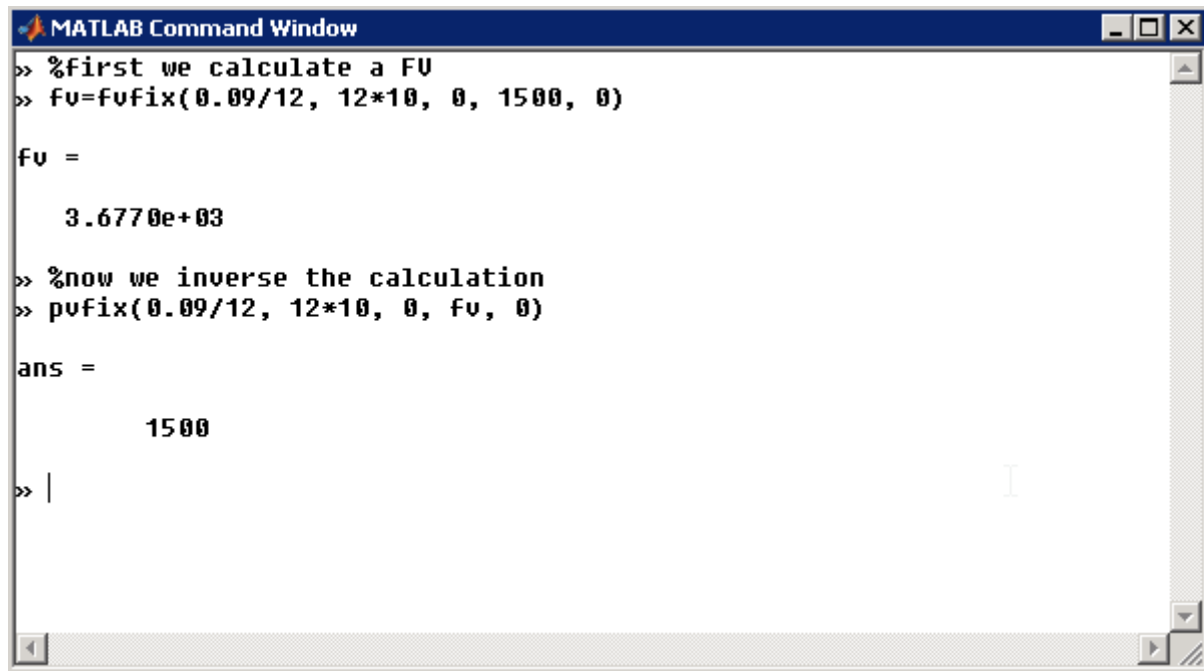
    3.6770e+03
>> 1500*(1+0.09/12)^(12*10)

ans =

    3.6770e+03
>>

```


Composed Interest (fixed present value)



A screenshot of the MATLAB Command Window. The window has a blue title bar with the text "MATLAB Command Window" and standard window controls. The command history shows two main calculations. The first calculation is for the future value (FV) using the function `fxfix` with parameters `(0.09/12, 12*10, 0, 1500, 0)`, resulting in `3.6770e+03`. The second calculation is for the present value (PV) using the function `pvfix` with parameters `(0.09/12, 12*10, 0, fv, 0)`, where `fv` is the variable from the first calculation, resulting in `1500`. The window has a scrollbar on the right and a status bar at the bottom.

```
MATLAB Command Window
>> %first we calculate a FV
>> fv=fxfix(0.09/12, 12*10, 0, 1500, 0)

fv =

    3.6770e+03

>> %now we inverse the calculation
>> pvfix(0.09/12, 12*10, 0, fv, 0)

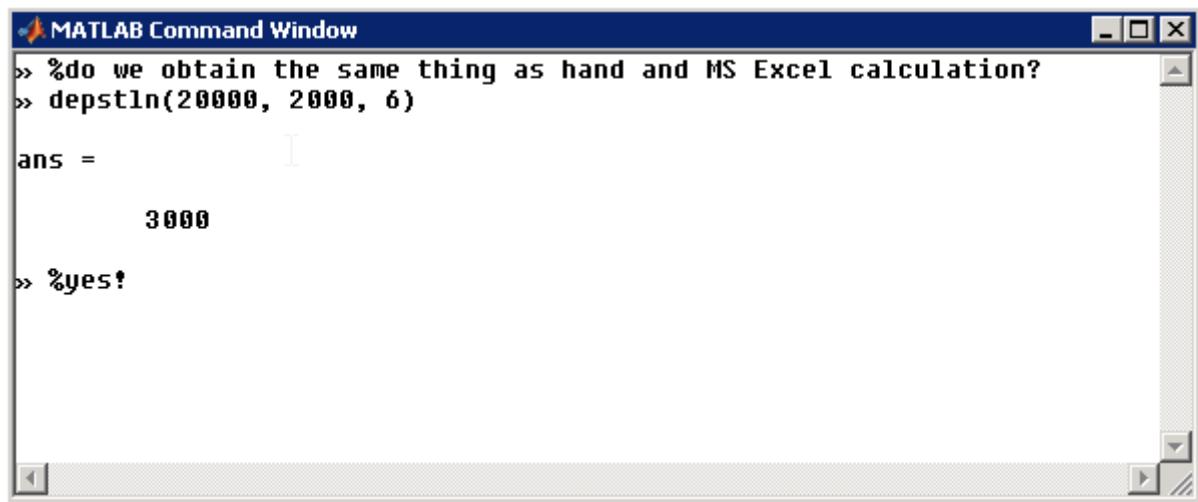
ans =

    1500

>> |
```

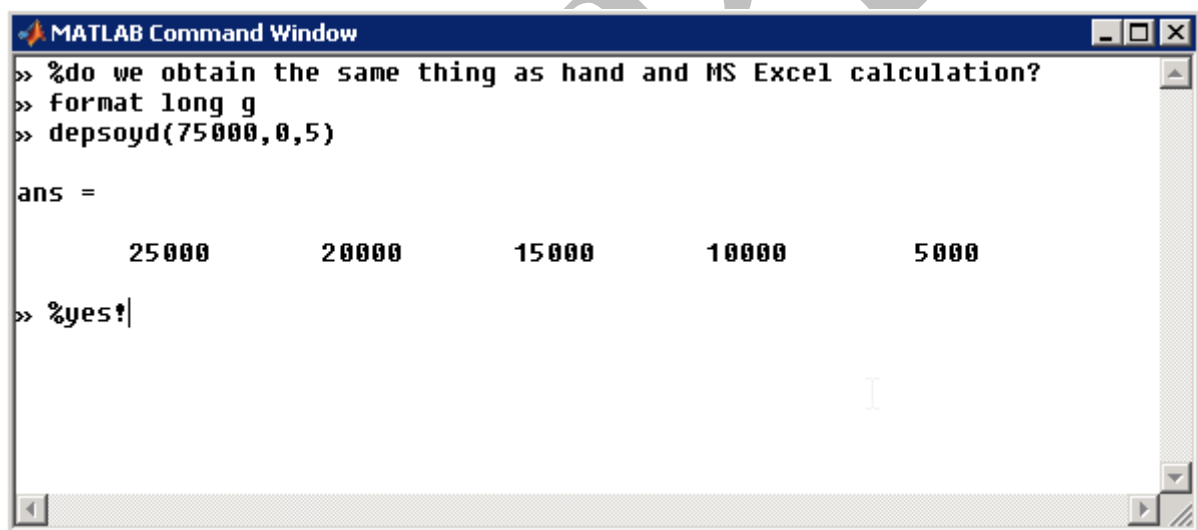
Amortization and depreciation

Straight line depreciation (SLN)



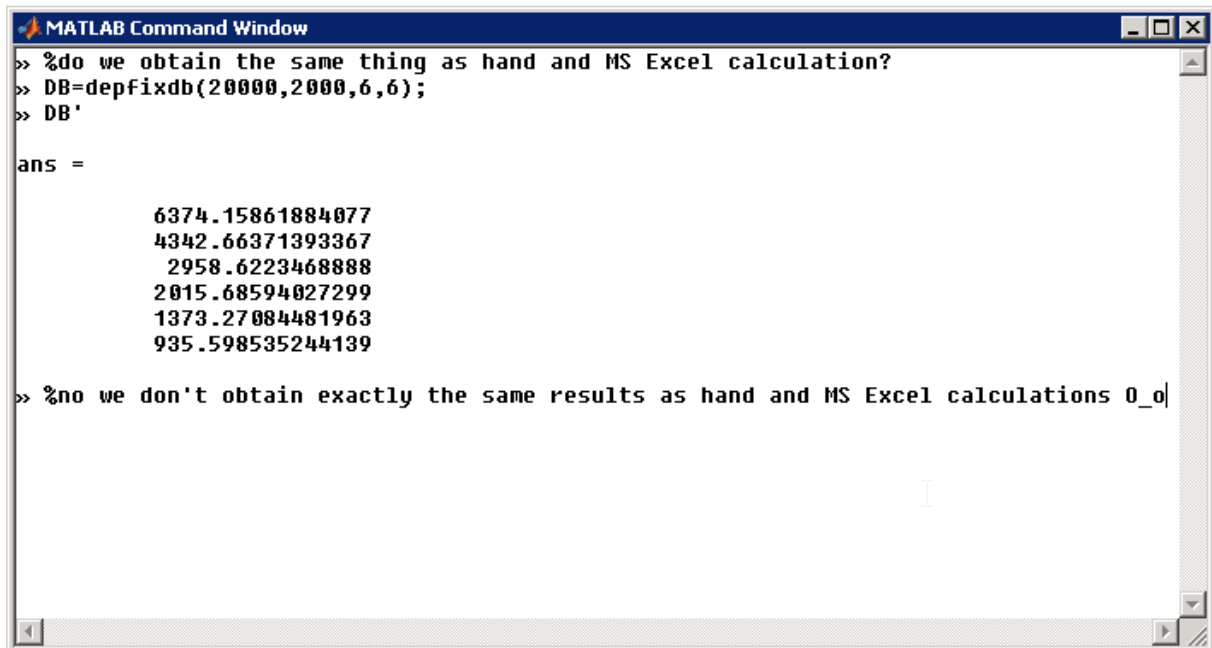
A screenshot of the MATLAB Command Window. The window title is "MATLAB Command Window". The command prompt shows the following text:
» %do we obtain the same thing as hand and MS Excel calculation?
» depstln(20000, 2000, 6)
The output is:
ans =
3000
The command prompt then shows:
» %yes!

Sum of years digit (SYD)



A screenshot of the MATLAB Command Window. The window title is "MATLAB Command Window". The command prompt shows the following text:
» %do we obtain the same thing as hand and MS Excel calculation?
» format long g
» depsoyd(75000,0,5)
The output is:
ans =
25000 20000 15000 10000 5000
The command prompt then shows:
» %yes!

Fixed declining-balance depreciation schedule (DB)



```

MATLAB Command Window
>> %do we obtain the same thing as hand and MS Excel calculation?
>> DB=depfixdb(20000,2000,6,6);
>> DB'

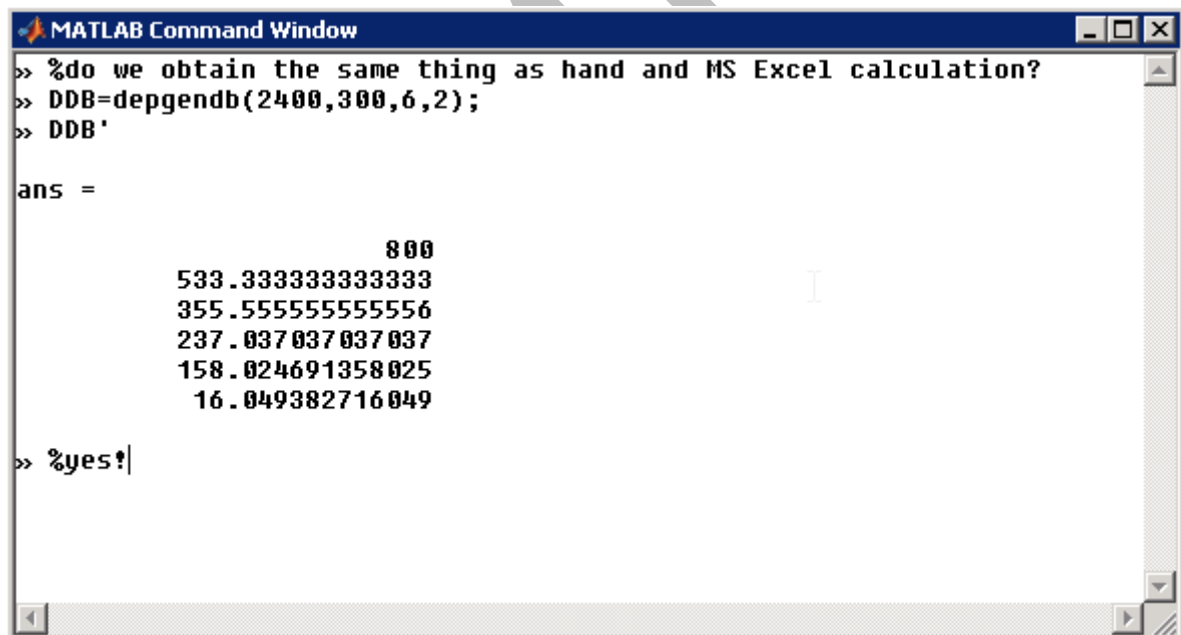
ans =

    6374.15861884077
    4342.66371393367
    2958.6223468888
    2015.68594027299
    1373.27084481963
    935.598535244139

>> %no we don't obtain exactly the same results as hand and MS Excel calculations 0_0|

```

Double declining balance (DDB)



```

MATLAB Command Window
>> %do we obtain the same thing as hand and MS Excel calculation?
>> DDB=depgendb(2400,300,6,2);
>> DDB'

ans =

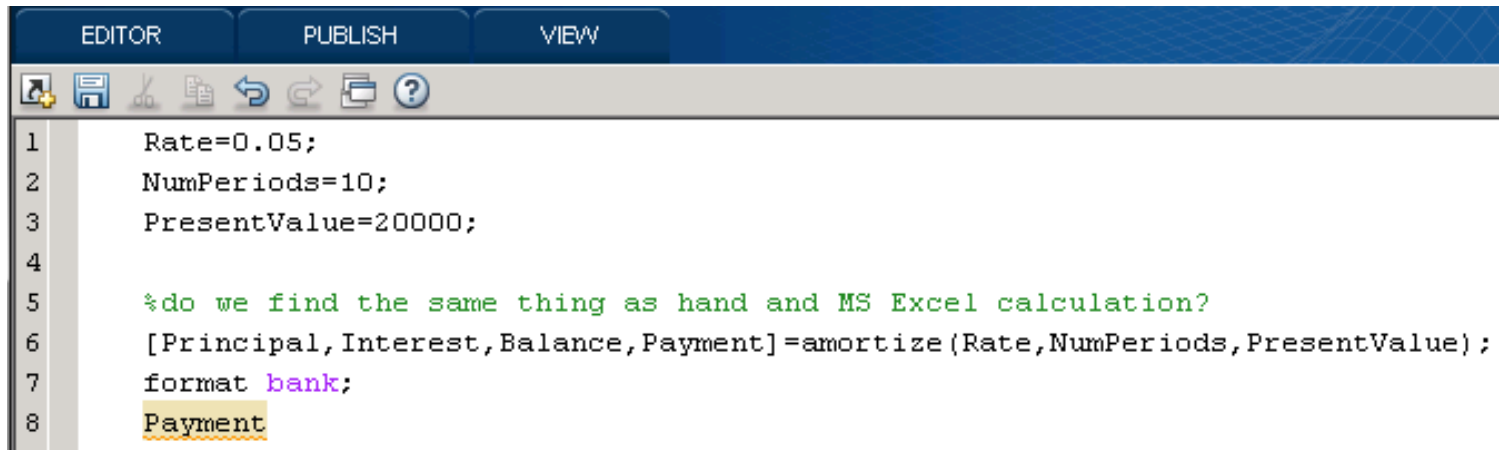
           800
    533.333333333333
    355.555555555556
    237.037037037037
    158.024691358025
    16.049382716049

>> %yes!|

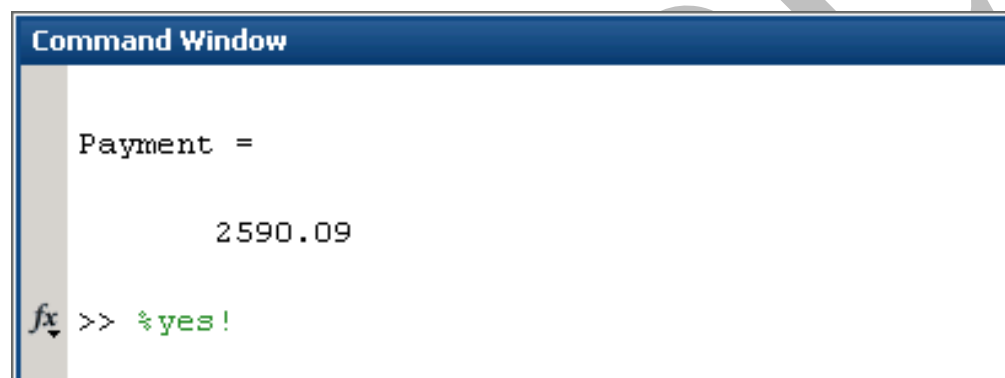
```

Loans and credits

Payment for a loan based on constant payments and a constant interest rate (PMT)



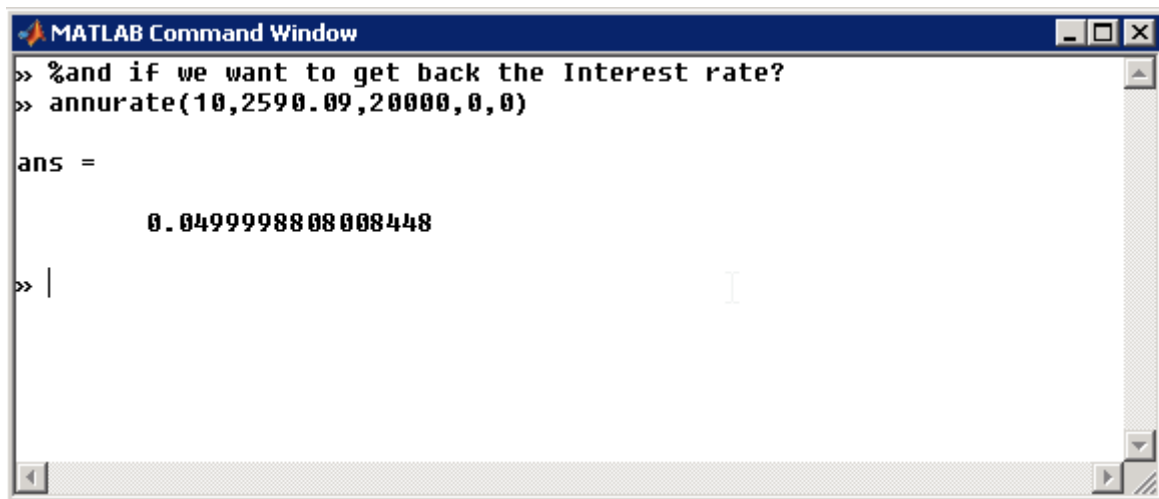
```
1 Rate=0.05;  
2 NumPeriods=10;  
3 PresentValue=20000;  
4  
5 %do we find the same thing as hand and MS Excel calculation?  
6 [Principal,Interest,Balance,Payment]=amortize(Rate,NumPeriods,PresentValue);  
7 format bank;  
8 Payment
```



```
Command Window  
  
Payment =  
  
2590.09  
  
fx >> %yes!
```



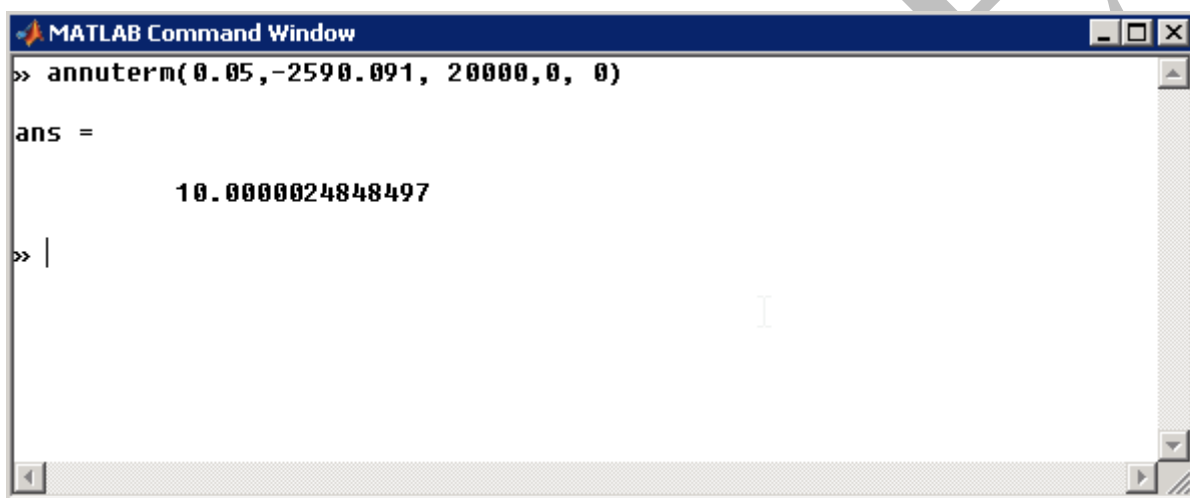
```
MATLAB Command Window  
» %others way...  
» payper(0.05,10,20000,0,0)  
  
ans =  
  
2590.09149930913  
  
» payadv(0.05,10,20000,0,0)  
  
ans =  
  
2590.09149930913  
  
» |
```



```
MATLAB Command Window
>> %and if we want to get back the Interest rate?
>> annurate(10,2590.09,20000,0,0)

ans =

    0.0499998808008448
>> |
```

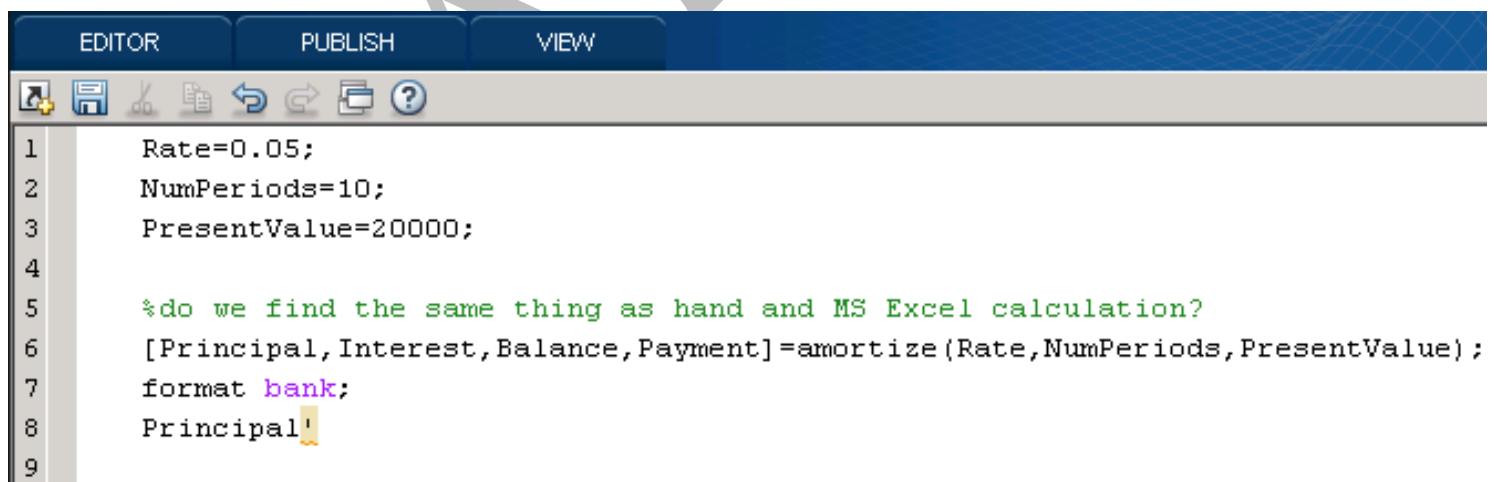


```
MATLAB Command Window
>> annuterm(0.05,-2590.091, 20000,0, 0)

ans =

    10.0000024848497
>> |
```

Principal Payment (PPMT)



```
EDITOR PUBLISH VIEW
1 Rate=0.05;
2 NumPeriods=10;
3 PresentValue=20000;
4
5 %do we find the same thing as hand and MS Excel calculation?
6 [Principal,Interest,Balance,Payment]=amortize(Rate,NumPeriods,PresentValue);
7 format bank;
8 Principal
9
```

```
Command Window

Principal =

    1590.09
    1669.60
    1753.08
    1840.73
    1932.77
    2029.40
    2130.87
    2237.42
    2349.29
    2466.75

fx >> %yes!
```

Interest Payment (IPMT)

```
EDITOR PUBLISH VIEW

1 Rate=0.05;
2 NumPeriods=10;
3 PresentValue=20000;
4
5 %do we find the same thing as hand and MS Excel calculation?
6 [Principal,Interest,Balance,Payment]=amortize(Rate,NumPeriods,PresentValue);
7 format bank;
8 Interest=Interest'
```

```

Command Window

Interest =

    1000.00
     920.50
     837.02
     749.36
     657.33
     560.69
     459.22
     352.67
     240.80
     123.34

fx >> %yes!

```

Payment Present Value (PV)

```

EDITOR PUBLISH VIEW

1 Rate=0.05;
2 NumPeriods=10;
3 PresentValue=20000;
4
5 %do we find the same thing as hand and MS Excel calculation?
6 [Principal, Interest, Balance, Payment]=amortize(Rate,NumPeriods,PresentValue);
7 format bank;
8 PresentValue=Balance'

```

```

Command Window

PresentValue =

    18409.91
    16740.31
    14987.24
    13146.51
    11213.74
     9184.34
     7053.46
     4816.04
     2466.75
      0.00

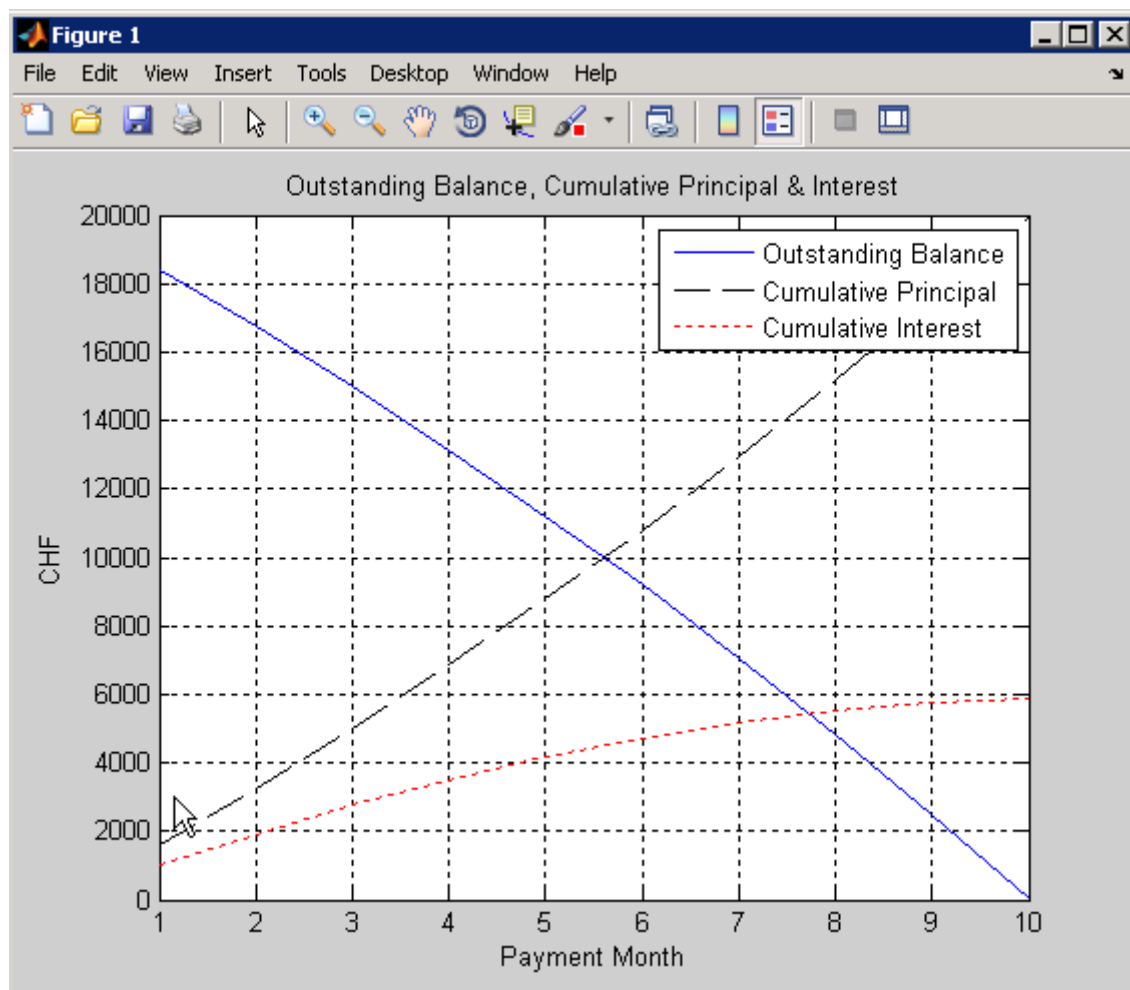
fx >> %yes!

```

Plot resume of loan payment

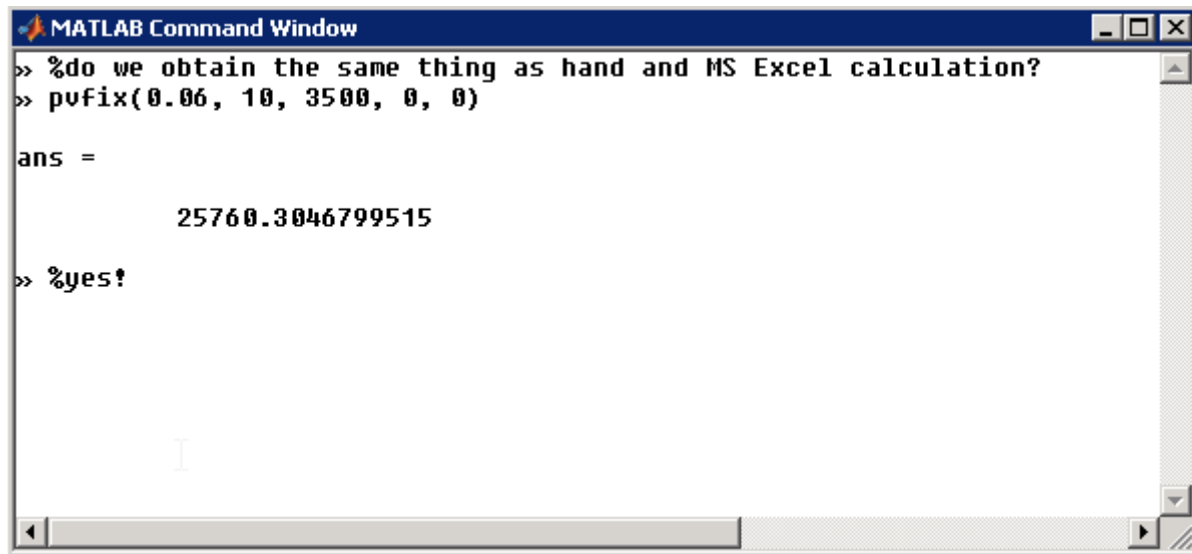


```
1 Rate=0.05;
2 NumPeriods=10;
3 PresentValue=20000;
4
5 %do we find the same thing as hand and MS Excel calculation?
6 [Principal,Interest,Balance,Payment]=amortize(Rate,NumPeriods,PresentValue);
7
8 plot(Balance,'b'), hold('on')
9 plot(cumsum(Principal),'--k')
10 plot(cumsum(Interest),':r')
11
12 xlabel('Payment Month');ylabel('CHF')
13 grid('on');
14 title('Outstanding Balance, Cumulative Principal & Interest')
15 legend('Outstanding Balance','Cumulative Principal','Cumulative Interest')
16 y=0:2000:20000; % define the y values where you want to have a tick
17 set(gca,'YTick',y); % Apply the ticks to the current axes
18 set(gca,'YTickLabel',arrayfun(@(v) sprintf('%d',v),y,'UniformOutput',false));
```

Annuities

Postnumerando actual annuity value (PV)

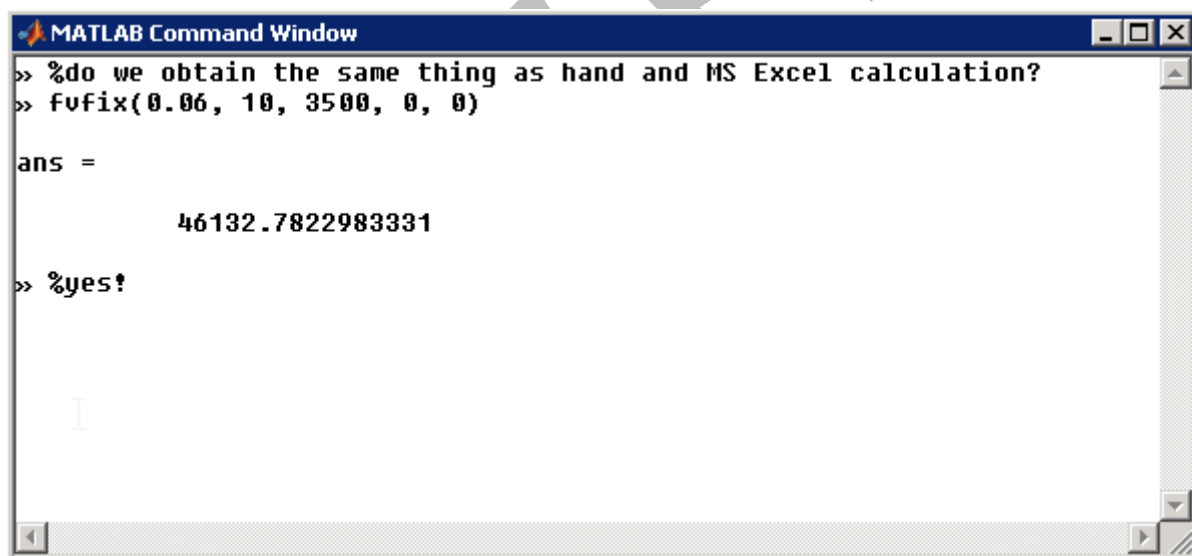


```
MATLAB Command Window
>> %do we obtain the same thing as hand and MS Excel calculation?
>> pvfix(0.06, 10, 3500, 0, 0)

ans =

    25760.3046799515
>> %yes!
```

Postnumerando forward annuity value (PV)

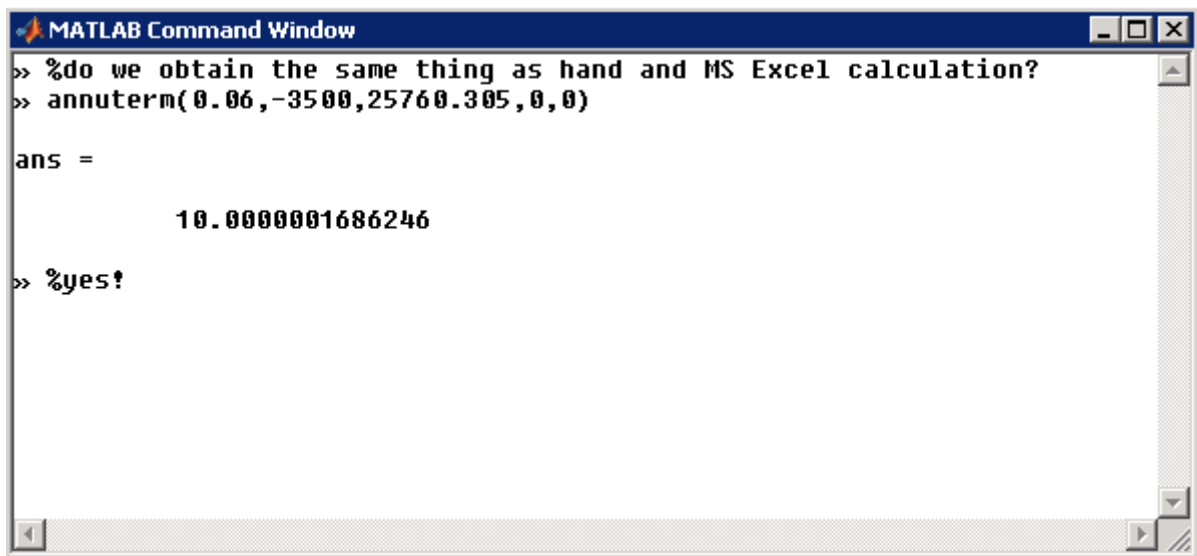


```
MATLAB Command Window
>> %do we obtain the same thing as hand and MS Excel calculation?
>> fvfix(0.06, 10, 3500, 0, 0)

ans =

    46132.7822983331
>> %yes!
```

Postnumerando number of periods (NPER)

A screenshot of the MATLAB Command Window. The title bar reads "MATLAB Command Window". The command prompt shows two lines of input: ">> %do we obtain the same thing as hand and MS Excel calculation?" and ">> annuterm(0.06,-3500,25760.305,0,0)". The output shows "ans =" followed by the value "10.0000001686246". Below the output, the command prompt shows ">> %yes?".

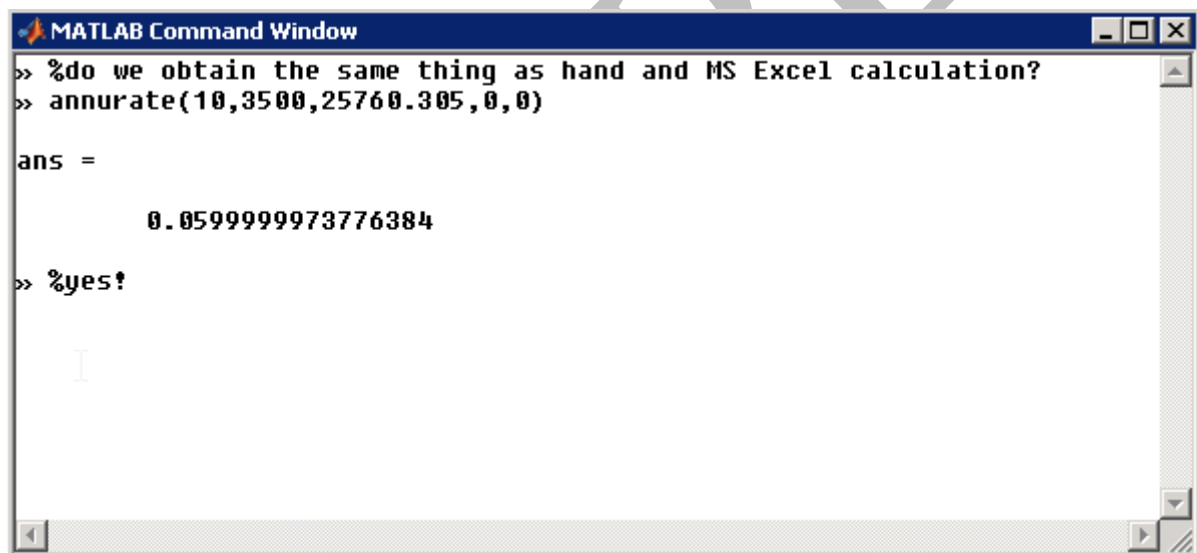
```
MATLAB Command Window
>> %do we obtain the same thing as hand and MS Excel calculation?
>> annuterm(0.06,-3500,25760.305,0,0)

ans =

    10.0000001686246

>> %yes!
```

Postnumerando annuity rate (ANNURATE)

A screenshot of the MATLAB Command Window. The title bar reads "MATLAB Command Window". The command prompt shows two lines of input: ">> %do we obtain the same thing as hand and MS Excel calculation?" and ">> annurate(10,3500,25760.305,0,0)". The output shows "ans =" followed by the value "0.05999999973776384". Below the output, the command prompt shows ">> %yes?".

```
MATLAB Command Window
>> %do we obtain the same thing as hand and MS Excel calculation?
>> annurate(10,3500,25760.305,0,0)

ans =

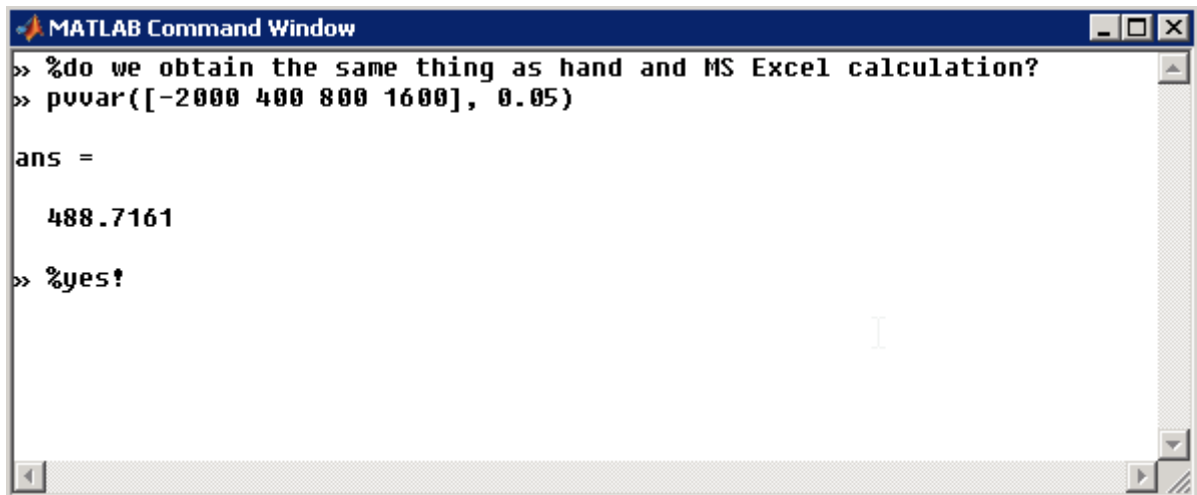
    0.05999999973776384

>> %yes!
```

Corporate finance

Net Present Value (NPV)

Periodic NPV

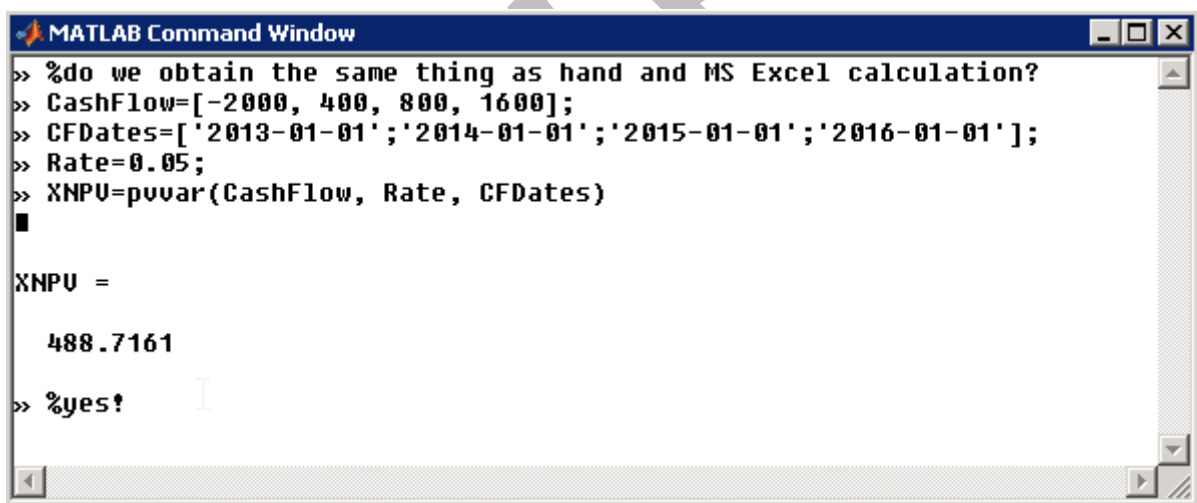
A screenshot of the MATLAB Command Window. The title bar reads "MATLAB Command Window". The command prompt shows the following sequence: a comment line "%do we obtain the same thing as hand and MS Excel calculation?", followed by the command "pvvar([-2000 400 800 1600], 0.05)". The output is "ans = 488.7161". The prompt then shows "%yes!".

```
MATLAB Command Window
>> %do we obtain the same thing as hand and MS Excel calculation?
>> pvvar([-2000 400 800 1600], 0.05)

ans =

    488.7161
>> %yes!
```

non-Periodic NPV

A screenshot of the MATLAB Command Window. The title bar reads "MATLAB Command Window". The command prompt shows the following sequence: a comment line "%do we obtain the same thing as hand and MS Excel calculation?", followed by variable assignments: "CashFlow=[-2000, 400, 800, 1600];", "CFDates=['2013-01-01'; '2014-01-01'; '2015-01-01'; '2016-01-01'];", "Rate=0.05;", and the command "XNPV=pvvar(CashFlow, Rate, CFDates)". The output is "XNPV = 488.7161". The prompt then shows "%yes!".

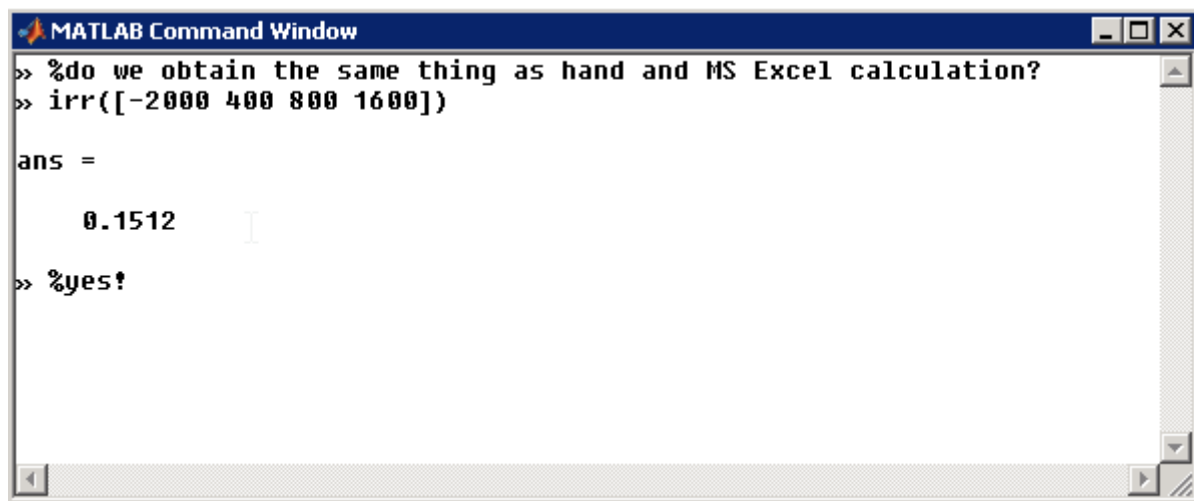
```
MATLAB Command Window
>> %do we obtain the same thing as hand and MS Excel calculation?
>> CashFlow=[-2000, 400, 800, 1600];
>> CFDates=['2013-01-01'; '2014-01-01'; '2015-01-01'; '2016-01-01'];
>> Rate=0.05;
>> XNPV=pvvar(CashFlow, Rate, CFDates)

XNPV =

    488.7161
>> %yes!
```

Internal Rate of Return (IRR)

Periodic IRR

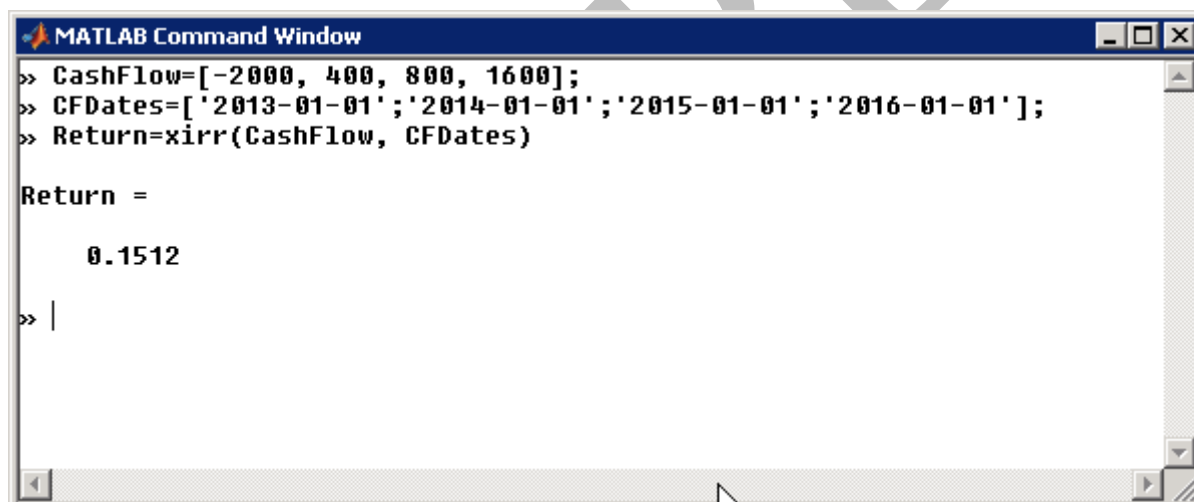
A screenshot of the MATLAB Command Window. The title bar reads "MATLAB Command Window". The command prompt shows the user asking a question and then entering the command `irr([-2000 400 800 1600])`. The output is `ans = 0.1512`. The user then enters `%yes!`.

```
MATLAB Command Window
>> %do we obtain the same thing as hand and MS Excel calculation?
>> irr([-2000 400 800 1600])

ans =

    0.1512
>> %yes!
```

non-Periodic IRR

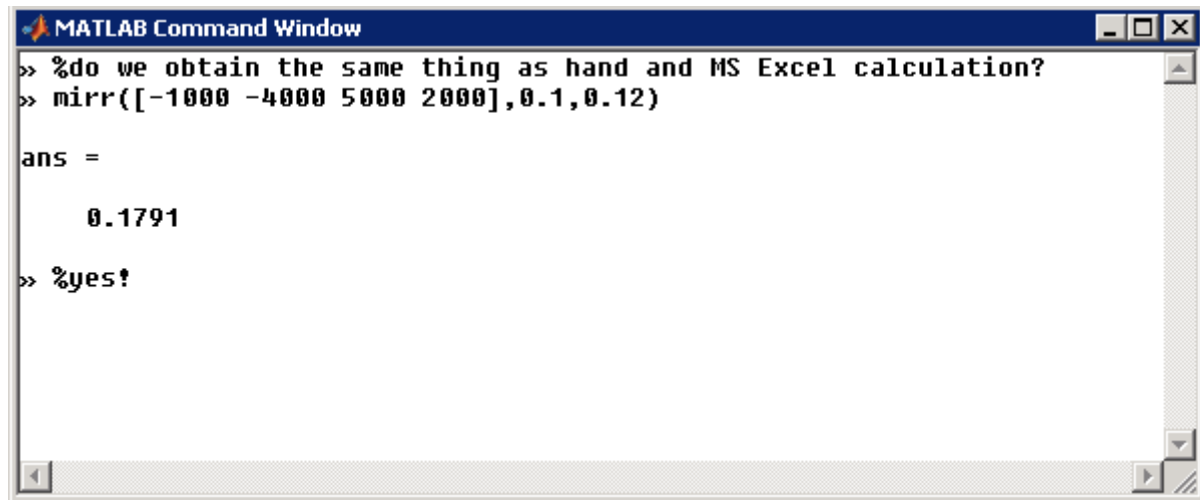
A screenshot of the MATLAB Command Window. The title bar reads "MATLAB Command Window". The command prompt shows the user defining `CashFlow` and `CFDates`, then entering the command `Return=xirr(CashFlow, CFDates)`. The output is `Return = 0.1512`. The user then enters a vertical bar `|`.

```
MATLAB Command Window
>> CashFlow=[-2000, 400, 800, 1600];
>> CFDates=['2013-01-01';'2014-01-01';'2015-01-01';'2016-01-01'];
>> Return=xirr(CashFlow, CFDates)

Return =

    0.1512
>> |
```

Modified Internal Rate of Return (MIRR)

A screenshot of the MATLAB Command Window. The window has a blue title bar with the MATLAB logo and the text "MATLAB Command Window". The command prompt shows a user asking a question and then entering a command. The output shows the result of the MIRR calculation.

```
>> %do we obtain the same thing as hand and MS Excel calculation?
>> mirr([-1000 -4000 5000 2000],0.1,0.12)

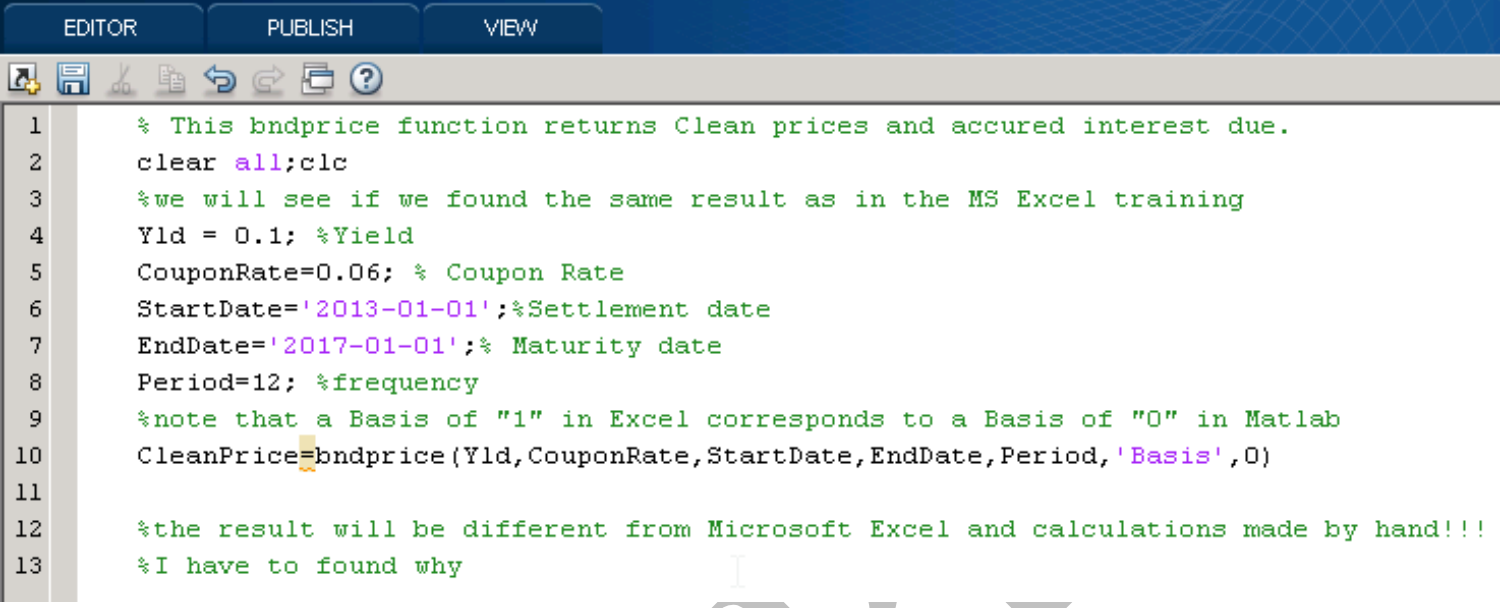
ans =

    0.1791

>> %yes!
```

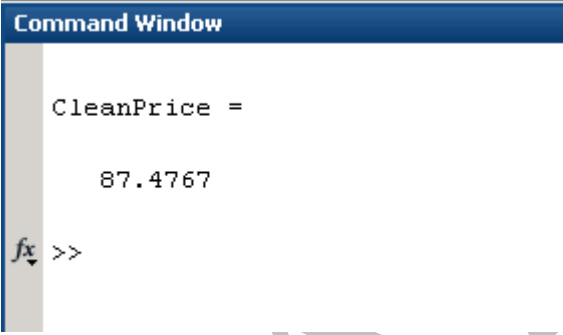
Treasure bonds/securities calculations

Clean price of a security (PRICE)



```

1  % This bndprice function returns Clean prices and accrued interest due.
2  clear all;clc
3  %we will see if we found the same result as in the MS Excel training
4  Yld = 0.1; %Yield
5  CouponRate=0.06; % Coupon Rate
6  StartDate='2013-01-01';%Settlement date
7  EndDate='2017-01-01';% Maturity date
8  Period=12; %frequency
9  %note that a Basis of "1" in Excel corresponds to a Basis of "0" in Matlab
10 CleanPrice=bndprice(Yld,CouponRate,StartDate,EndDate,Period,'Basis',0)
11
12 %the result will be different from Microsoft Excel and calculations made by hand!!!
13 %I have to found why
  
```



```

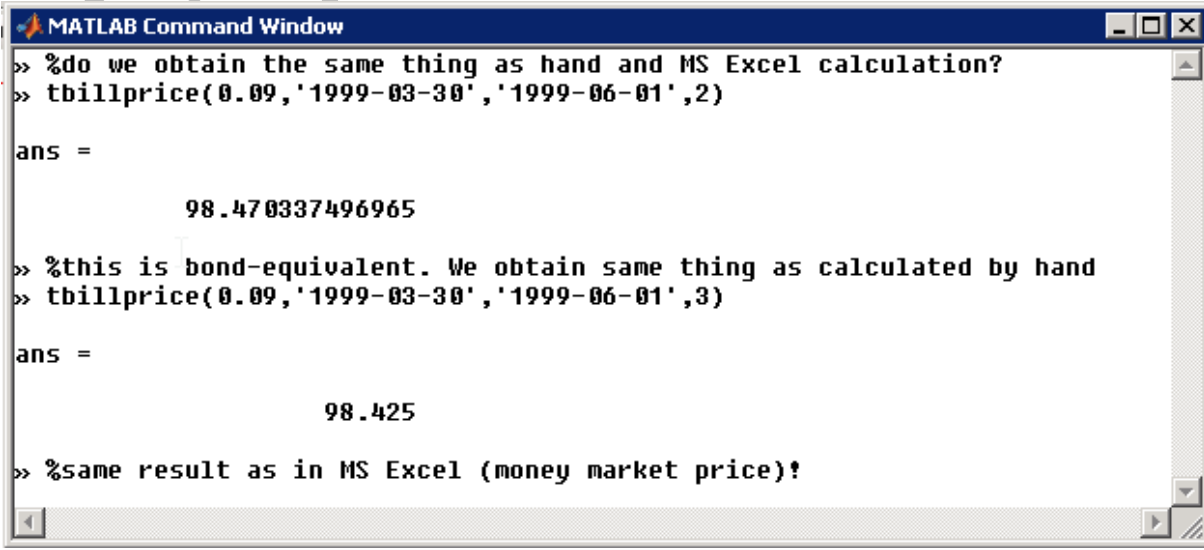
Command Window

CleanPrice =

    87.4767

fx >>
  
```

Price of treasury bond/security (TBILLPRICE)



```

MATLAB Command Window

>> %do we obtain the same thing as hand and MS Excel calculation?
>> tbillprice(0.09,'1999-03-30','1999-06-01',2)

ans =

    98.470337496965

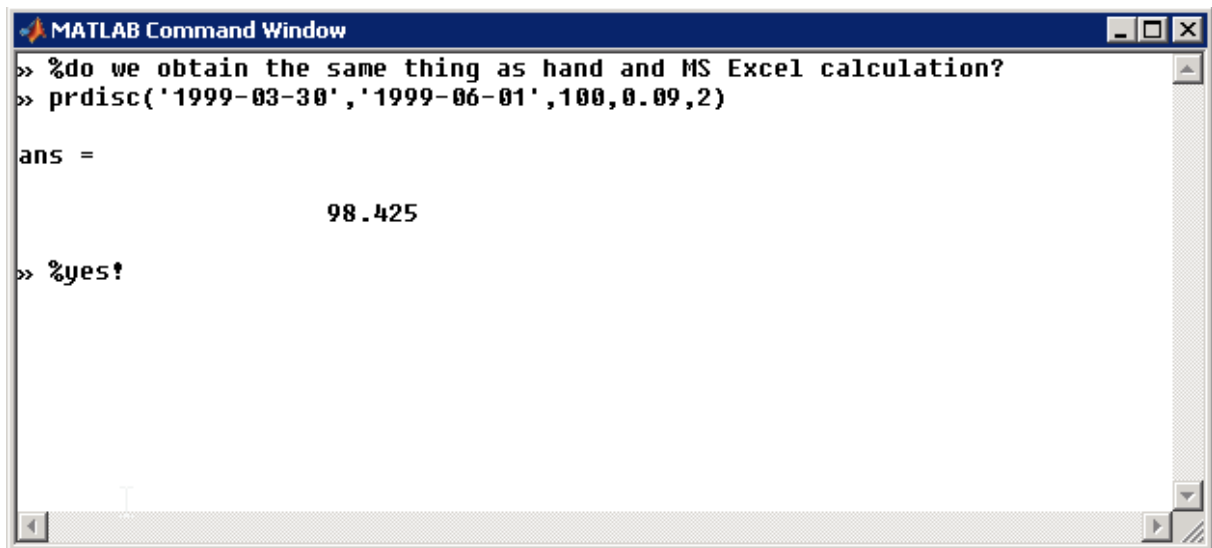
>> %this is bond-equivalent. We obtain same thing as calculated by hand
>> tbillprice(0.09,'1999-03-30','1999-06-01',3)

ans =

    98.425

>> %same result as in MS Excel (money market price)?
  
```

Remember that a treasury bond is by default on 100.-. Pricing a security is based on the same mathematical rules but with a different nominal:



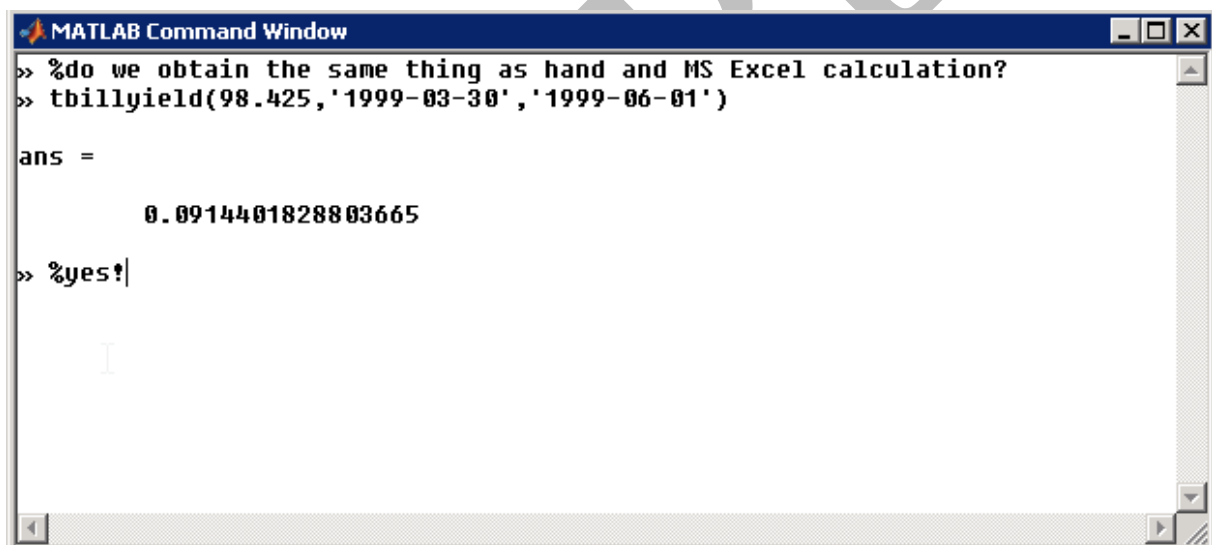
```
MATLAB Command Window
>> %do we obtain the same thing as hand and MS Excel calculation?
>> prdisc('1999-03-30','1999-06-01',100,0.09,2)

ans =

          98.425

>> %yes!
```

Yield of a treasury bond (TBILLYIELD)



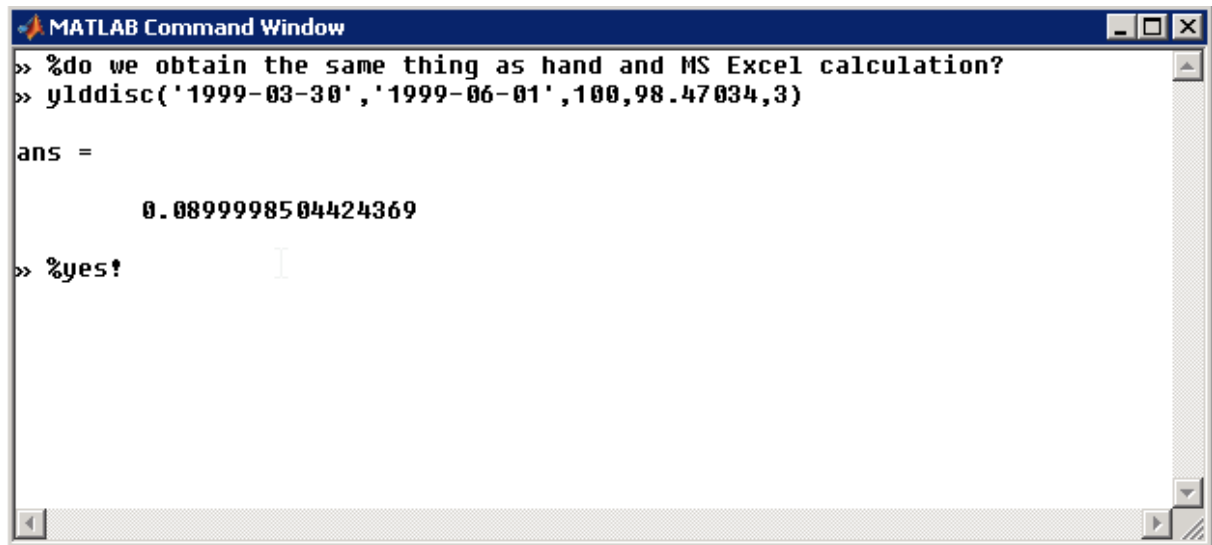
```
MATLAB Command Window
>> %do we obtain the same thing as hand and MS Excel calculation?
>> tbillyield(98.425,'1999-03-30','1999-06-01')

ans =

0.0914401828803665

>> %yes!
```


Discount rate of a security (DISC)



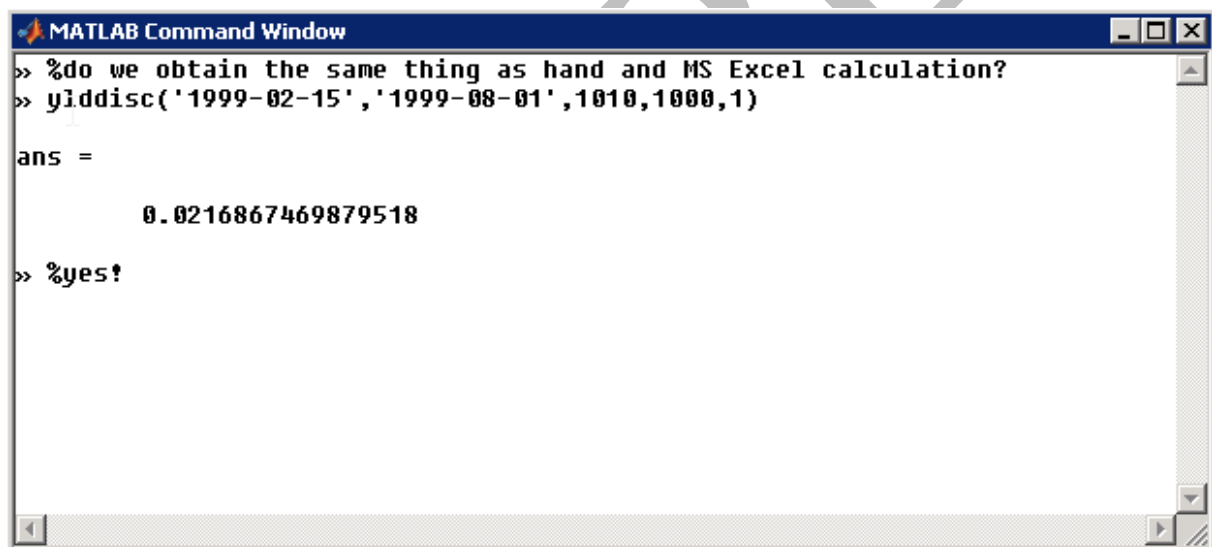
```
MATLAB Command Window
>> %do we obtain the same thing as hand and MS Excel calculation?
>> ylddisc('1999-03-30','1999-06-01',100,98.47034,3)

ans =

    0.0899998504424369

>> %yes!
```

Annual yield for a discounted security (YIELDDISC)



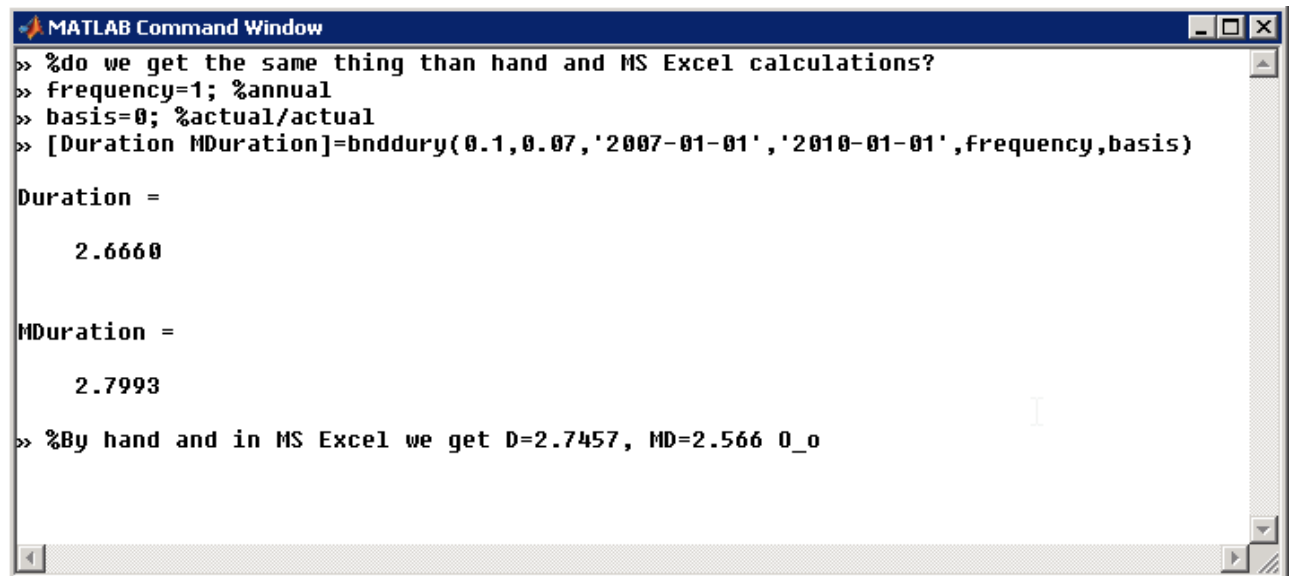
```
MATLAB Command Window
>> %do we obtain the same thing as hand and MS Excel calculation?
>> ylddisc('1999-02-15','1999-08-01',1010,1000,1)

ans =

    0.0216867469879518

>> %yes!
```

Bond duration and Macaulay duration



A screenshot of the MATLAB Command Window. The window has a blue title bar with the MATLAB logo and the text "MATLAB Command Window". The command prompt shows the following commands and outputs:

```
>> %do we get the same thing than hand and MS Excel calculations?
>> frequency=1; %annual
>> basis=0; %actual/actual
>> [Duration MDuration]=bnddury(0.1,0.07,'2007-01-01','2010-01-01',frequency,basis)

Duration =

    2.6660

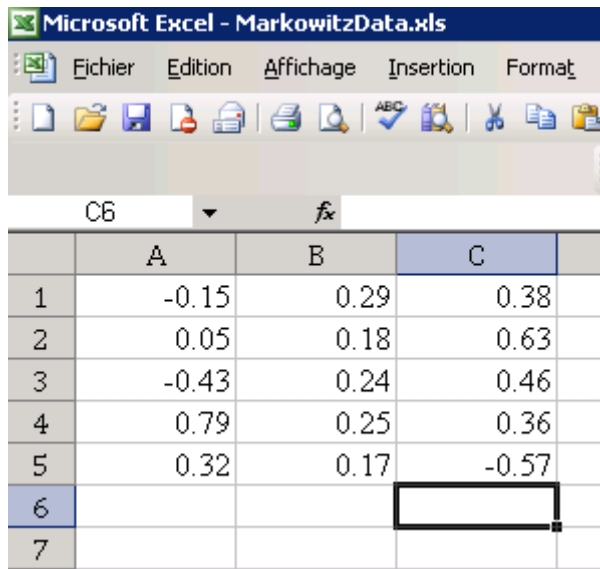
MDuration =

    2.7993

>> %By hand and in MS Excel we get D=2.7457, MD=2.566 0_o
```

A large, light gray "SAMPLE" watermark is diagonally across the page.

Naive Markowitz portfolio models



Microsoft Excel - MarkowitzData.xls

Fichier Edition Affichage Insertion Format

C6

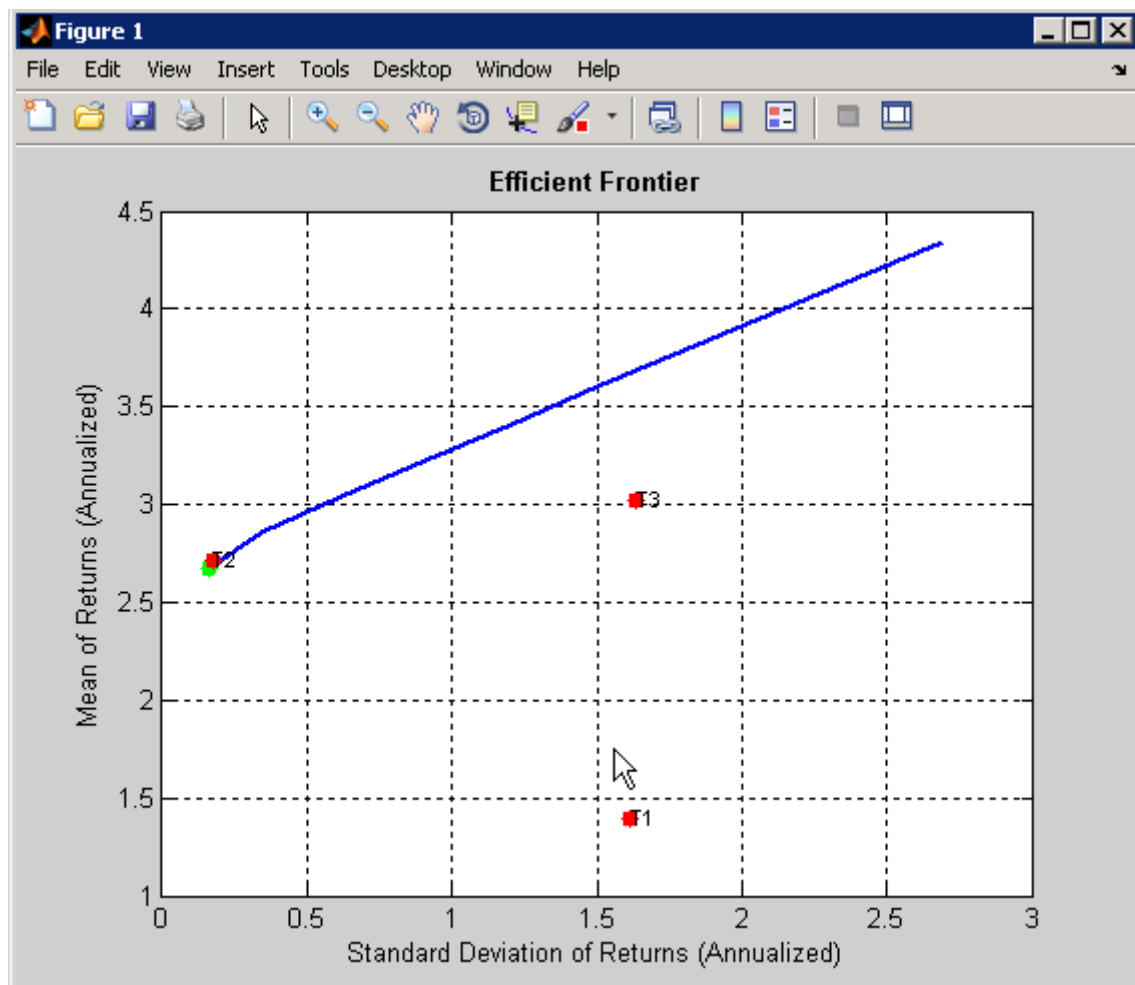
	A	B	C
1	-0.15	0.29	0.38
2	0.05	0.18	0.63
3	-0.43	0.24	0.46
4	0.79	0.25	0.36
5	0.32	0.17	-0.57
6			
7			

Optimal short-long portfolio (without transaction costs)

```

1  %%Global Minimum Variance Markowitz Model (short sales allowed/unconstrained)
2  clear all;close all;clc
3  returns = xlsread('E:\MarkowitzData.xls');
4  ExpReturns = mean(returns);
5  c = cov(returns);
6  StdDev = std(returns);
7
8  AssetBounds = [-1 -1 -1 ;1 1 1];
9  [PortRisk,PortReturn,PortWts]=frontcon(ExpReturns,c,[],[],AssetBounds);
10 names = {'T1' 'T2' 'T3'};
11
12 portfolioexamples_plot('Efficient Frontier', ...
13     {'line', PortRisk, PortReturn}, ...
14     {'scatter',StdDev , ExpReturns, names, 'r'});
15
16 RisklessRate=0.001; %can't be null or even NaN (???)
17 BorrowRate=NaN; %can't be =0 but can be =NaN ... (???)
18 [RiskyRisk, RiskyReturn, RiskyWts, RiskyFraction, OverallRisk, ...
19     OverallReturn] = portalloc(PortRisk, PortReturn, PortWts,RisklessRate,BorrowRate);
20 printmat(RiskyWts,'Optimal Portfolio','weight','T1 T2 T3')
21
22 clf;
23 portfolioexamples_plot('Efficient Frontier', ...
24     {'line', PortRisk, PortReturn}, ...
25     {'scatter',RiskyRisk,RiskyReturn},...
26     {'scatter',StdDev , ExpReturns, names, 'r'});
27
28 RiskyReturn
29 Variance=RiskyRisk^2

```



Command Window

Optimal Portfolio =

	T1	T2	T3
weight	0.02582	1.00000	-0.02582

RiskyReturn =

0.2225

Variance =

0.0021

f_x >> |

Optimal portfolio with implicit constraints no short sales allowed (without transaction costs)

The result is very sensitive to the number of simulations between 0 and 100 simulations.

```

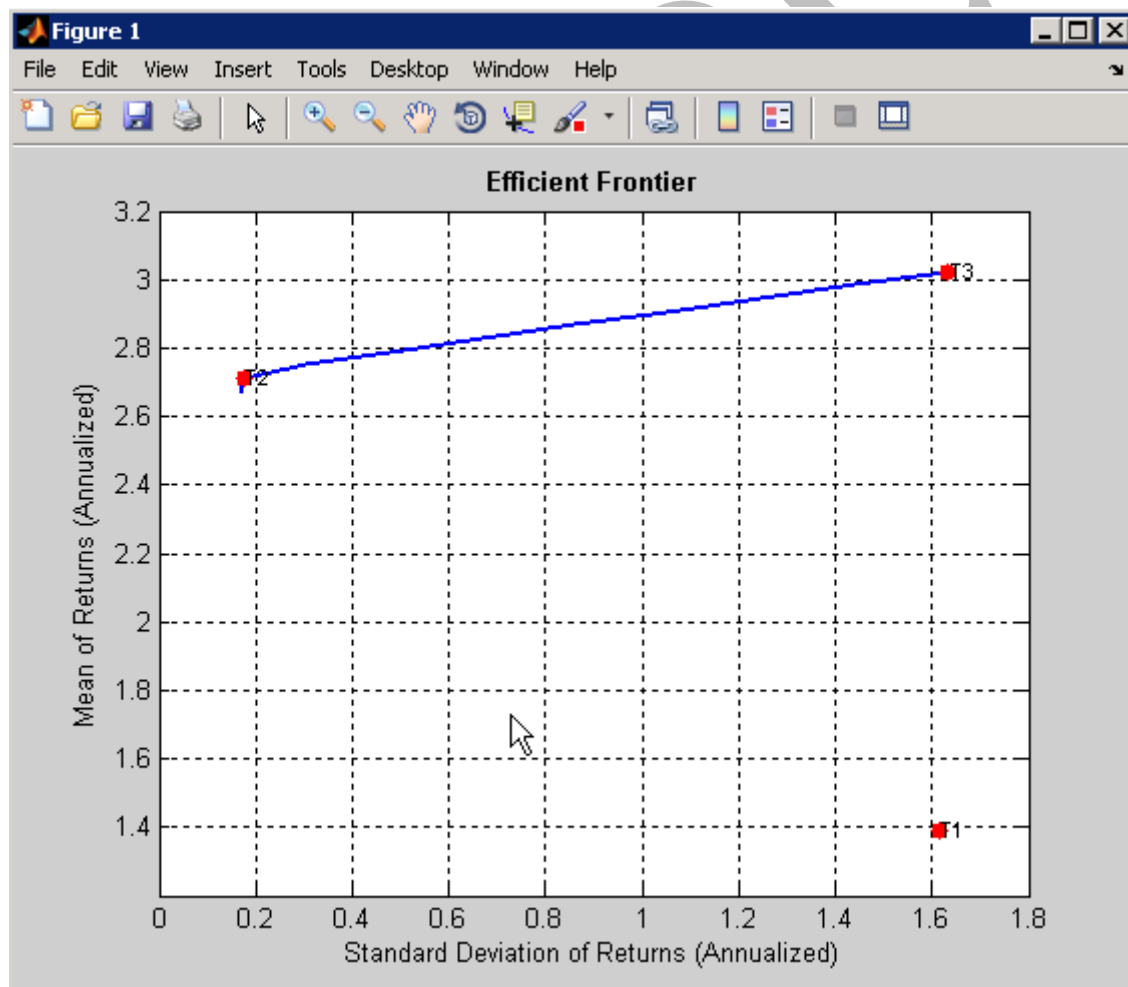
1  %%Global Minimum Variance Markowitz Model (short sales not Allowed)
2  clear all;close all;clc
3  returns = xlsread('E:\MarkowitzData.xls');
4  %If the dataset has missing values,
5  returns(any(isnan(returns),2),:) = [];
6  %we calculate the historical mean, covariance, and standard deviation
7  ExpReturns = mean(returns);
8  M_Cov = cov(returns);
9  StdDev = std(returns);
10 SimulatedPortfolios=10;
11 [PortRisk, PortReturn, PortWts] = portopt(ExpReturns,M_Cov,SimulatedPortfolios)
  
```

PortRisk =	PortReturn =	PortWts =
0.0483	0.2227	0.0296 0.9704 0
0.0503	0.2260	0.0000 1.0000 0
0.0879	0.2292	0 0.8751 0.1249
0.1389	0.2325	0 0.7501 0.2499
0.1929	0.2357	0 0.6251 0.3749
0.2481	0.2390	0 0.5001 0.4999
0.3037	0.2422	0 0.3750 0.6250
0.3595	0.2455	0 0.2500 0.7500
0.4155	0.2487	0 0.1250 0.8750
0.4717	0.2520	0 0 1.0000

```

EDITOR PUBLISH VIEW
1 %%Global Minimum Variance Markowitz Model (short sales not Allowed)
2 clear all;close all;clc
3 returns = xlsread('E:\MarkowitzData.xls');
4 %If the dataset has missing values,
5 returns(any(isnan(returns),2),:) = [];
6 %we calculate the historical mean, covariance, and standard deviation
7 ExpReturns = mean(returns);
8 M_Cov = cov(returns);
9 StdDev = std(returns);
10 SimulatedPortfolios=10;
11 [PortRisk, PortReturn, PortWts] = portopt(ExpReturns,M_Cov,SimulatedPortfolios);
12 names = {'T1' 'T2' 'T3'};
13 portfolioexamples_plot('Efficient Frontier', {'line', PortRisk, PortReturn}, ...
14 {'scatter',StdDev , ExpReturns, names, 'r'});
15

```



```

EDITOR PUBLISH VIEW
1  %%Global Minimum Variance Markowitz Model (short sales not Allowed)
2  clear all;close all;clc
3  returns = xlsread('E:\MarkowitzData.xls');
4  %If the dataset has missing values,
5  returns(any(isnan(returns),2),:) = [];
6  %we calculate the historical mean, covariance, and standard deviation
7  ExpReturns = mean(returns);
8  M_Cov = cov(returns);
9  StdDev = std(returns);
10 SimulatedPortfolios=10;
11 [PortRisk, PortReturn, PortWts] = portopt(ExpReturns,M_Cov,SimulatedPortfolios);
12 names = {'T1' 'T2' 'T3'};
13 portfolioexamples_plot('Efficient Frontier', {'line', PortRisk, PortReturn}, ...
14     {'scatter',StdDev , ExpReturns, names, 'r'});
15
16 RisklessRate=0.001; %can't be null or even NaN (???)
17 BorrowRate=NaN; %can't be =0 but can be =NaN ... (???)
18 [RiskyRisk, RiskyReturn, RiskyWts, ~,~]...
19     = portalloc(PortRisk, PortReturn, PortWts, RisklessRate,BorrowRate);
20 printmat(RiskyWts,'Optimal Portfolio','weight','T1 T2 T3')

```

```

Command Window

Optimal Portfolio =

           T1           T2           T3
weight    0.01627    0.98373         0

fx >>

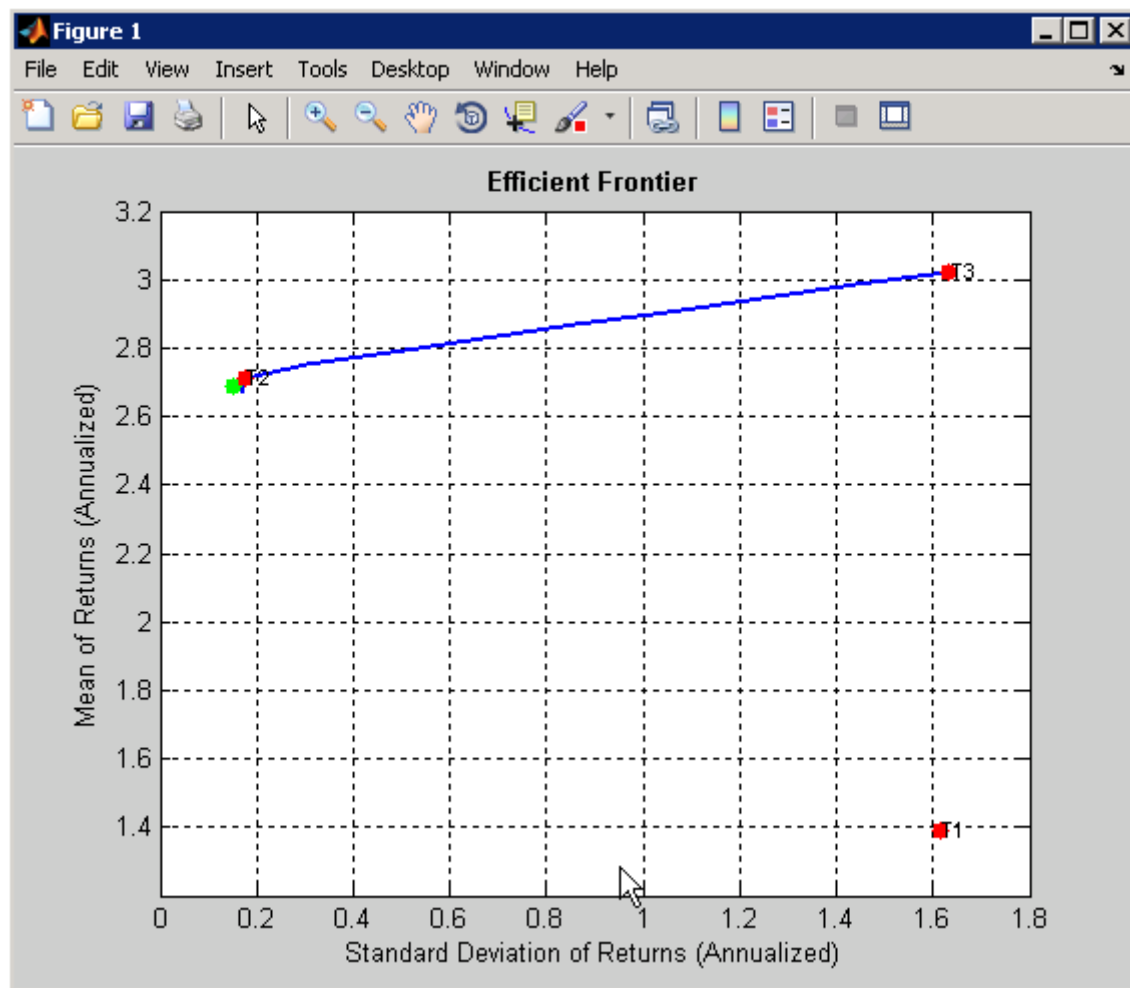
```



```

EDITOR PUBLISH VIEW
1  %%Global Minimum Variance Markowitz Model (short sales not Allowed)
2  clear all;close all;clc
3  returns = xlsread('E:\MarkowitzData.xls');
4  %If the dataset has missing values,
5  returns(any(isnan(returns),2),:) = [];
6  %we calculate the historical mean, covariance, and standard deviation
7  ExpReturns = mean(returns);
8  M_Cov = cov(returns);
9  StdDev = std(returns);
10 SimulatedPortfolios=10;
11 [PortRisk, PortReturn, PortWts] = portopt(ExpReturns,M_Cov,SimulatedPortfolios);
12 names = {'T1' 'T2' 'T3'};
13 portfolioexamples_plot('Efficient Frontier', {'line', PortRisk, PortReturn}, ...
14     {'scatter',StdDev , ExpReturns, names, 'r'});
15
16 RisklessRate=0.001; %can't be null or even NaN (???)
17 BorrowRate=NaN; %can't be =0 but can be =NaN ... (???)
18 [RiskyRisk, RiskyReturn, RiskyWts, ~,~]...
19     = portalloc(PortRisk, PortReturn, PortWts, RisklessRate,BorrowRate);
20 printmat(RiskyWts,'Optimal Portfolio','weight','T1 T2 T3')
21 clf; %clear figure
22 portfolioexamples_plot('Efficient Frontier', ...
23     {'line', PortRisk, PortReturn}, ...
24     {'scatter',RiskyRisk,RiskyReturn},...
25     {'scatter',StdDev , ExpReturns, names, 'r'});

```

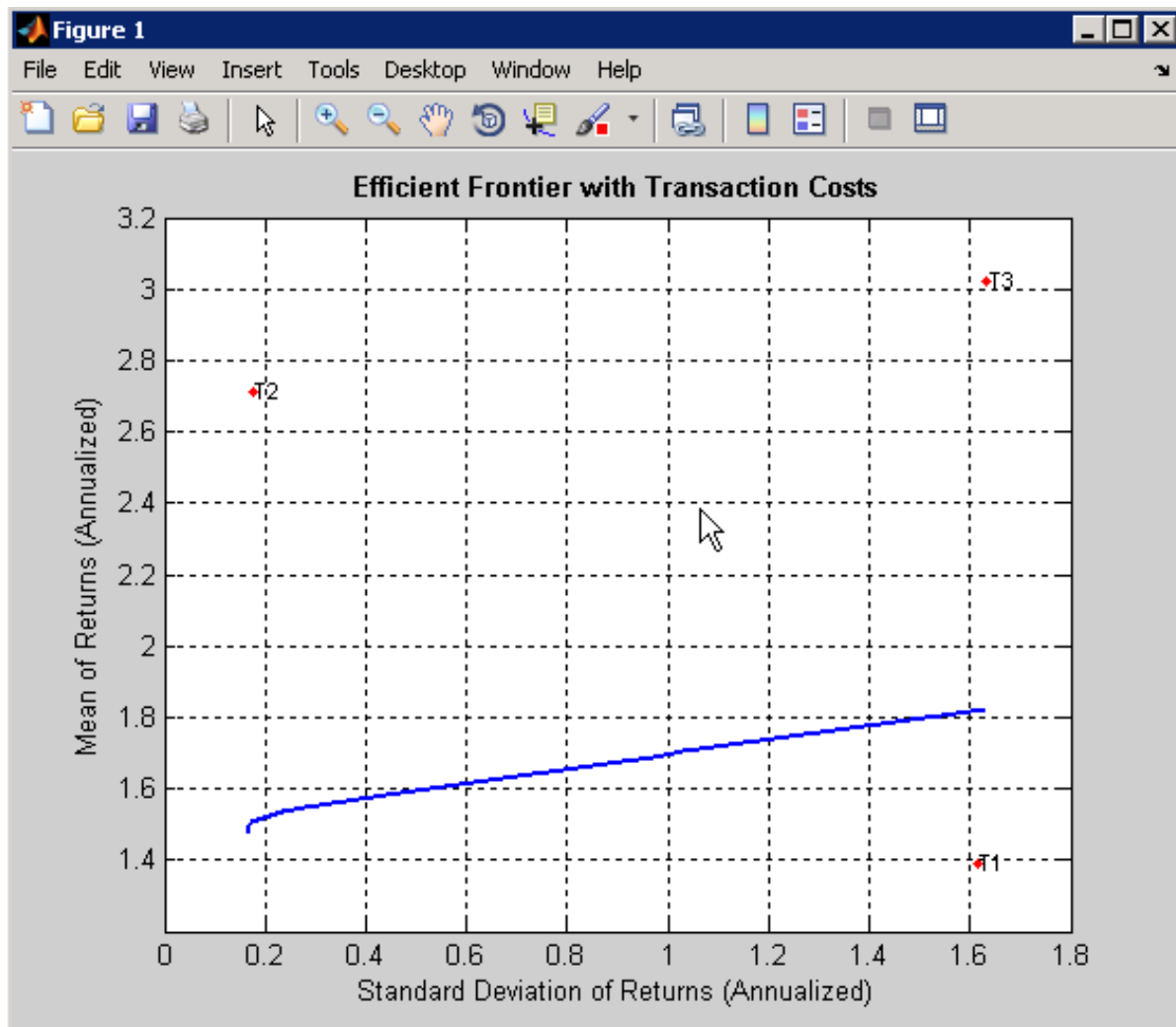


Optimal portfolio with implicit constraints no short sales allowed (**with** transaction costs)

```

1 - clear all;close all;clc
2 - ra=[-0.15,0.05,-0.43,0.79,0.32];
3 - rb=[0.29,0.18,0.24,0.25,0.17];
4 - rc=[0.38,0.63,0.46,0.36,-0.57];
5 - PfM=[0.45,0.61,0.54,0.29,0.12];Rf= repmat(0,1,length(PfM));
6 - data=[ra',rb',rc'];
7
8 - muMarket=mean(PfM); %market index expected return
9 - sigmaMarket=std(PfM); %market index risk
10 - rf=mean(Rf); %get risk free mean
11 - rfsigma=std(Rf); %get risk free sigma
12 - muAsset=mean(data); %expected return for remaining asset
13 - covAsset=cov(data); %covariance
14 - %create portfolio
15 - port=Portfolio('AssetList',{'T1','T2','T3'},'RiskFreeRate',rf);
16
17 - %work without Transactions Costs
18 - port=port.setAssetMoments(muAsset, covAsset);
19 - port=port.setDefaultConstraints; %no short sales!
20
21 - %work with Transactions Costs
22 - %if you put zero you will found the same frontier as before
23 - BuyCost = 0.1; SellCost = 0.3;
24 - q = port.setCosts(BuyCost, SellCost);
25 - qwgt = q.estimateFrontier(30);
26 - [qrsk, qret] = q.estimatePortMoments(qwgt);
27
28 - % Plot efficient frontiers with gross and net returns
29 - portfolioexamples_plot('Efficient Frontier with Transaction Costs', ...
30 -     {'line', qrsk, qret},...
31 -     {'scatter', sqrt(diag(port.AssetCovar)), port.AssetMean,...
32 -     port.AssetList, '.r'});

```



Get portfolios on the efficient frontier with explicit constraints (without transaction costs)

```

EDITOR    PUBLISH    VIEW
1      %Obtaining Efficient Portfolios for Target Returns
2      clear all;close all;clc
3      returns = xlsread('E:\MarkowitzData.xls');
4      %If the dataset has missing values,
5      returns(any(isnan(returns),2),:) = [];
6      %Calculate Markowitz Inputs
7      ExpReturns = mean(returns);
8      m_Cov = cov(returns);
9      %for optimization of Global Minimum Variance Portfolio
10     NumPortf=4;
11     PortReturn = []; %must be empty!!
12     AssetBounds = [-1 -1 -1; 1 1 1];
13     [PortRisk, PortReturn, PortWts] = frontcon(ExpReturns,m_Cov, NumPortf, ...
14         PortReturn,AssetBounds)

```

```
Command Window

PortRisk =

    0.0463
    0.2631
    0.5200
    0.7782

PortReturn =

    0.2225
    0.2690
    0.3155
    0.3620

PortWts =

    0.0258    1.0000   -0.0258
   -0.3161    1.0000    0.3161
   -0.6581    1.0000    0.6581
   -1.0000    1.0000    1.0000

fx >>
```

Equally weighted portfolio (EWP) (without transaction costs)

EDITOR	PUBLISH	VIEW
1	-	<code>clear all;close all;clc</code>
2	-	<code>ra=[-0.15,0.05,-0.43,0.79,0.32];</code>
3	-	<code>rb=[0.29,0.18,0.24,0.25,0.17];</code>
4	-	<code>rc=[0.38,0.63,0.46,0.36,-0.57];</code>
5	-	<code>PfM=[0.45,0.61,0.54,0.29,0.12];</code>
6	-	<code>Rf= repmat(0,1,length(PfM));</code>
7	-	<code>data=[ra' rb' rc'];</code>
8	-	
9	-	<code>muMarket=mean(PfM); %market index expected return</code>
10	-	<code>sigmaMarket=std(PfM); %market index risk</code>
11	-	<code>rf=mean(Rf); %get risk free mean</code>
12	-	<code>rfsigma=std(Rf); %get risk free sigma</code>
13	-	<code>muAsset=mean(data); %expected return for remaining asset</code>
14	-	<code>covAsset=cov(data); %covariance</code>
15	-	<code>%create portfolio</code>
16	-	<code>port=Portfolio('AssetList',{'T1', 'T2', 'T3'},'RiskFreeRate',rf);</code>
17	-	<code>port=port.setAssetMoments(muAsset, covAsset);</code>
18	-	<code>%compute the mean and std of equal-weight portfolio returns</code>
19	-	<code>port = port.setInitPort(1/port.NumAssets);</code>
20	-	<code>[sigmaEwp, muEwp] = port.estimatePortMoments(port.InitPort)</code>
21	-	
22	-	<code>%plot efficient frontier with also Ewp</code>
23	-	<code>port = port.setDefaultConstraints; %long only!</code>
24	-	<code>pwgt = port.estimateFrontier(30);</code>
25	-	<code>[muportfolio, sigmaportfolio] = port.estimatePortMoments(pwgt);</code>
26	-	<code>%plot efficient frontier</code>
27	-	<code>portfolioexamples_plot('Efficient Frontier', ...</code>
28	-	<code> {'line', muportfolio, sigmaportfolio}, ...</code>
29	-	<code> {'scatter',[sigmaMarket,rfsigma,sigmaEwp],[muMarket,rf,muEwp],...}</code>
30	-	<code> {'Market', 'rf', 'Ewp'}), {'scatter', sqrt(diag(port.AssetCovar)),...}</code>
31	-	<code>port.AssetMean, port.AssetList, '.r'});</code>

```
Command Window

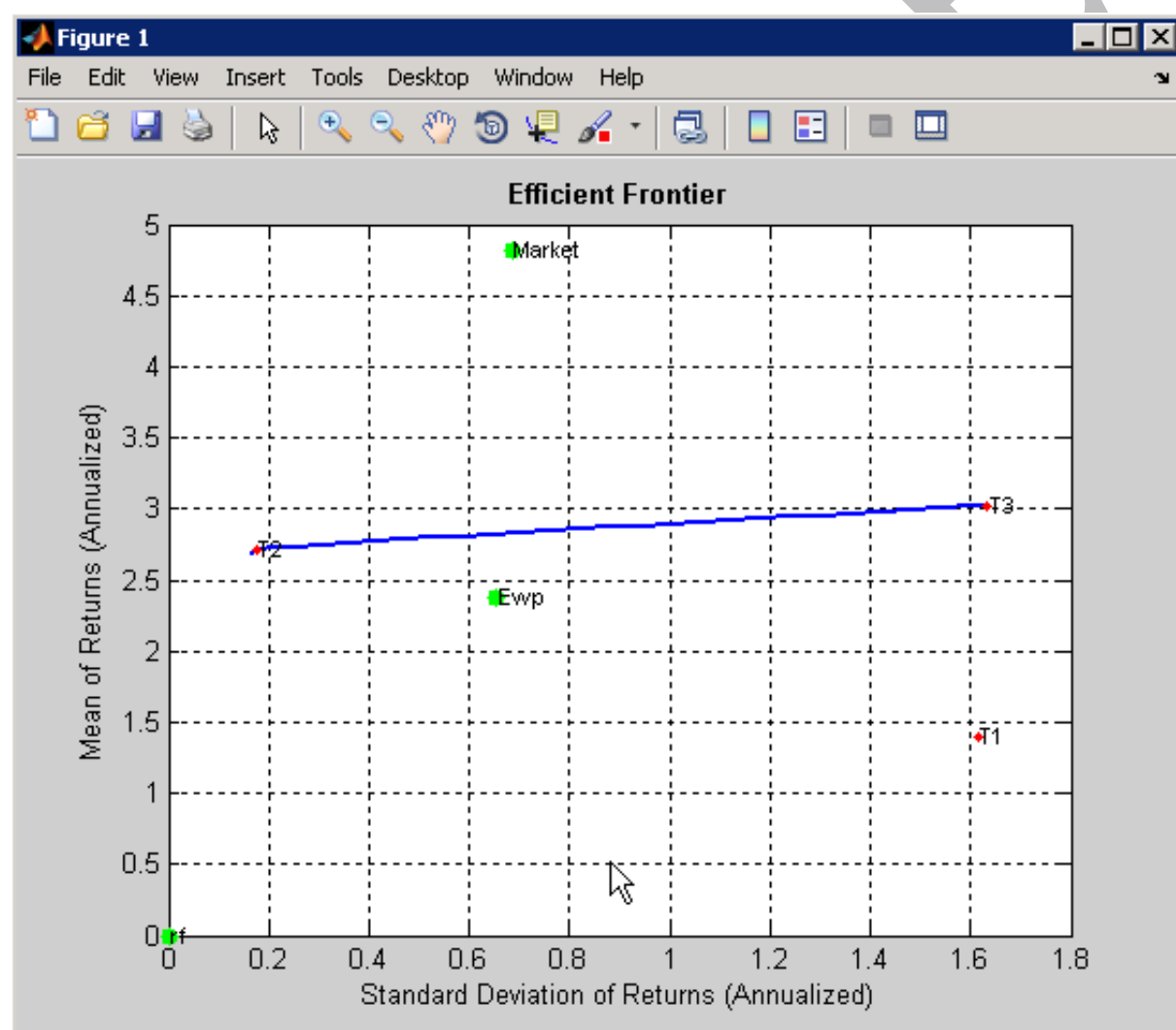
sigmaEwp =

    0.1889

muEwp =

    0.1980

fx >> |
```

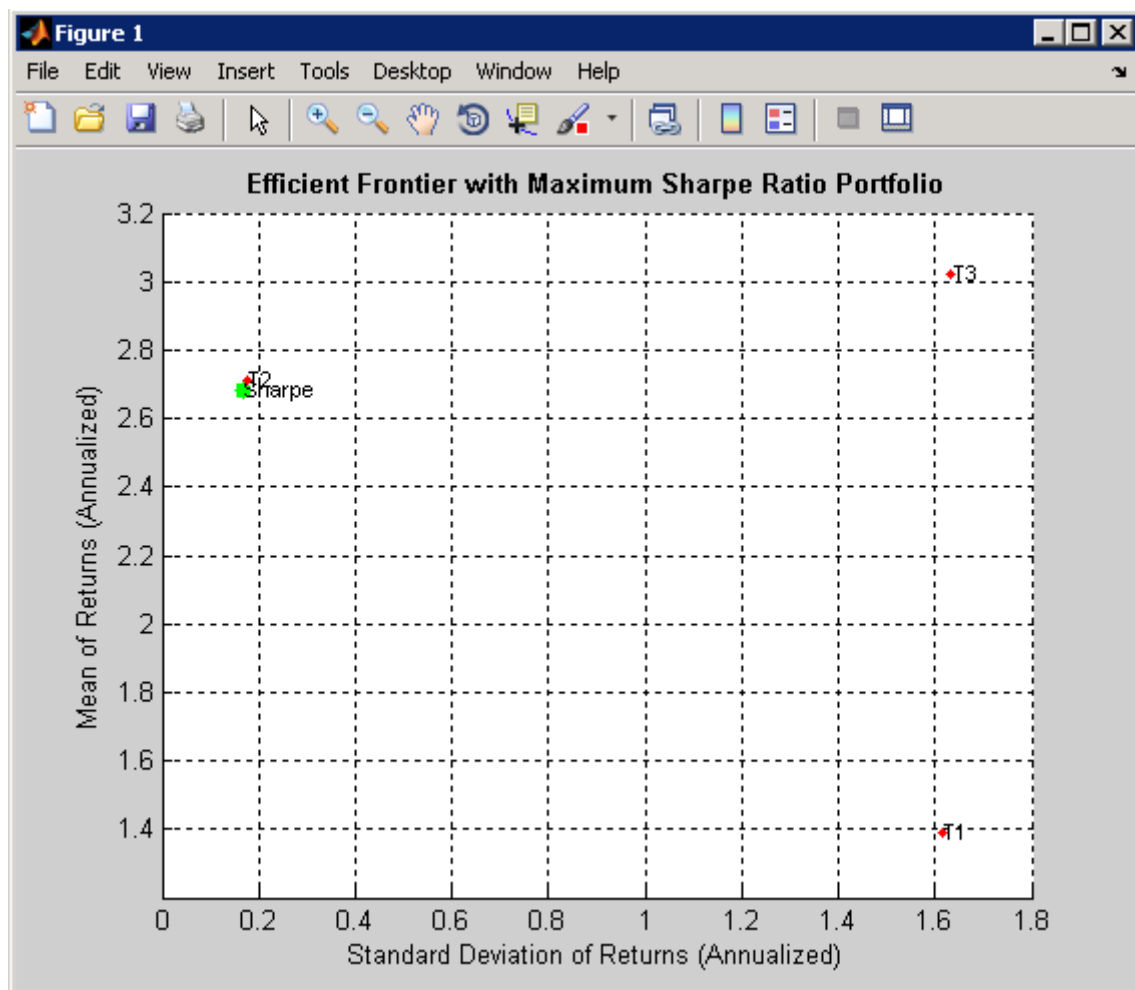


Maximum Sharpe Ratio (max. diversification) portfolio with implicit constraints (without costs)

```

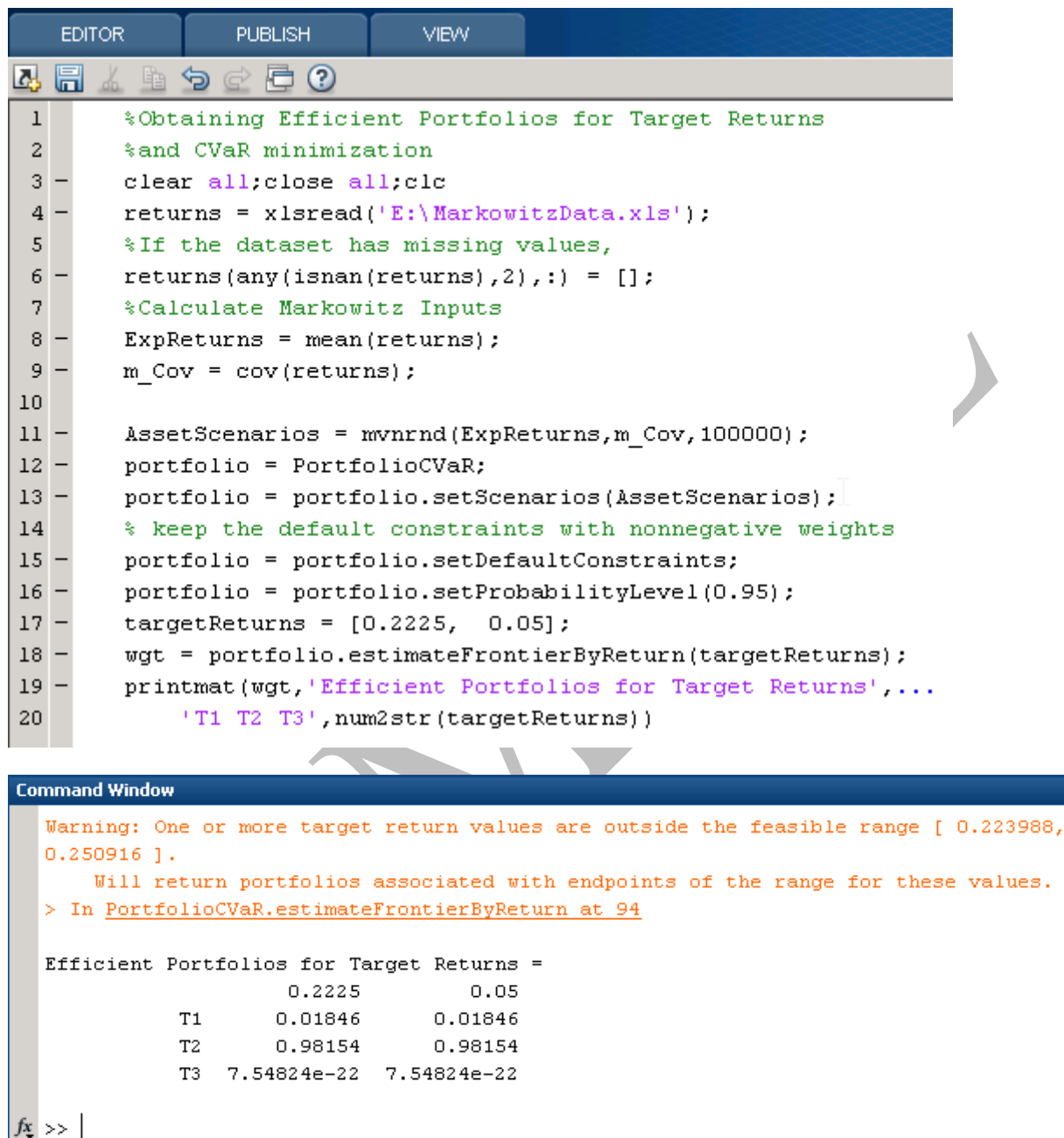
EDITOR PUBLISH VIEW
1 - clear all;close all;clc
2 - ra=[-0.15,0.05,-0.43,0.79,0.32];
3 - rb=[0.29,0.18,0.24,0.25,0.17];
4 - rc=[0.38,0.63,0.46,0.36,-0.57];
5 - PfM=[0.45,0.61,0.54,0.29,0.12];Rf= repmat(0,1,length(PfM));
6 - data=[ra',rb',rc'];
7
8 - muMarket=mean(PfM); %market index expected return
9 - sigmaMarket=std(PfM); %market index risk
10 - rf=mean(Rf); %get risk free mean
11 - rfsigma=std(Rf); %get risk free sigma
12 - muAsset=mean(data); %expected return for remaining asset
13 - covAsset=cov(data); %covariance
14 %create portfolio
15 - port=Portfolio('AssetList',{'T1','T2','T3'},'RiskFreeRate',rf);
16
17 %work without Transactions Costs
18 - port=port.setAssetMoments(muAsset, covAsset);
19 - port=port.setDefaultConstraints; %no short sales!
20
21 %Maximize Sharpe Ratio
22 - swgt = port.estimateMaxSharpeRatio;
23 - [srsk, sret] = port.estimatePortMoments(swgt)
24 % Plot efficient frontier with portfolio that attains maximum Sharpe ratio
25 - portfolioexamples_plot('Efficient Frontier with Maximum Sharpe Ratio Portfolio', ...
26     {'scatter', srsk, sret, {'Sharpe'}}, ...
27     {'scatter', sqrt(diag(port.AssetCovar)),...
28     port.AssetMean, port.AssetList, '.r'});

```


**Command Window**

```
srsk =  
  
    0.0483  
  
sret =  
  
    0.2233
```

Optimal portfolios for target returns and CVaR minimization (without transaction costs)



The image shows a MATLAB script in the Editor window and its execution output in the Command Window. The script calculates efficient portfolios for target returns and CVaR minimization. The Command Window shows a warning about target return values outside the feasible range and the resulting efficient portfolios.

```

1      %Obtaining Efficient Portfolios for Target Returns
2      %and CVaR minimization
3      - clear all;close all;clc
4      - returns = xlsread('E:\MarkowitzData.xls');
5      %If the dataset has missing values,
6      - returns(any(isnan(returns),2),:) = [];
7      %Calculate Markowitz Inputs
8      - ExpReturns = mean(returns);
9      - m_Cov = cov(returns);
10
11     AssetScenarios = mvnrnd(ExpReturns,m_Cov,100000);
12     portfolio = PortfolioCVaR;
13     portfolio = portfolio.setScenarios(AssetScenarios);
14     % keep the default constraints with nonnegative weights
15     portfolio = portfolio.setDefaultConstraints;
16     portfolio = portfolio.setProbabilityLevel(0.95);
17     targetReturns = [0.2225, 0.05];
18     wgt = portfolio.estimateFrontierByReturn(targetReturns);
19     printmat(wgt,'Efficient Portfolios for Target Returns',...
20             'T1 T2 T3',num2str(targetReturns))

```

Command Window

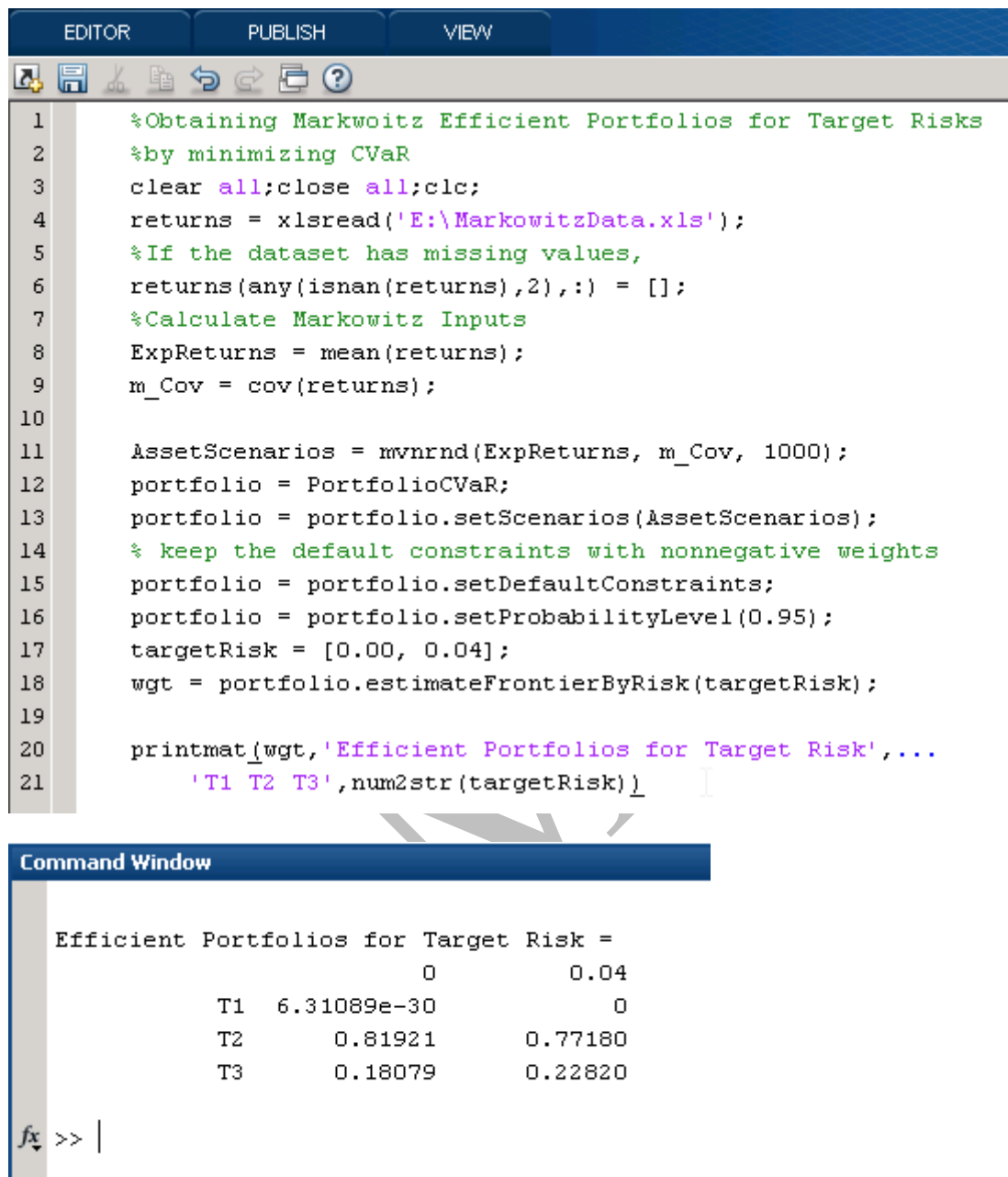
```

Warning: One or more target return values are outside the feasible range [ 0.223988,
0.250916 ].
    Will return portfolios associated with endpoints of the range for these values.
> In PortfolioCVaR.estimateFrontierByReturn at 94

Efficient Portfolios for Target Returns =
           0.2225           0.05
T1         0.01846         0.01846
T2         0.98154         0.98154
T3  7.54824e-22  7.54824e-22
fx >> |

```

Optimal portfolios for target risks and CVaR minimization (without transaction costs)



The image shows a MATLAB Editor window with a script for calculating efficient portfolios. The script reads data from 'E:\MarkowitzData.xls', calculates the mean return and covariance matrix, and uses the PortfolioCVaR class to estimate the frontier for target risks of 0 and 0.04. The Command Window displays the resulting portfolio weights for three target risk levels (T1, T2, T3).

```

1  %Obtaining Markowitz Efficient Portfolios for Target Risks
2  %by minimizing CVaR
3  clear all;close all;clc;
4  returns = xlsread('E:\MarkowitzData.xls');
5  %If the dataset has missing values,
6  returns(any(isnan(returns),2),:) = [];
7  %Calculate Markowitz Inputs
8  ExpReturns = mean(returns);
9  m_Cov = cov(returns);
10
11  AssetScenarios = mvnrnd(ExpReturns, m_Cov, 1000);
12  portfolio = PortfolioCVaR;
13  portfolio = portfolio.setScenarios(AssetScenarios);
14  % keep the default constraints with nonnegative weights
15  portfolio = portfolio.setDefaultConstraints;
16  portfolio = portfolio.setProbabilityLevel(0.95);
17  targetRisk = [0.00, 0.04];
18  wgt = portfolio.estimateFrontierByRisk(targetRisk);
19
20  printmat(wgt,'Efficient Portfolios for Target Risk',...
21          'T1 T2 T3',num2str(targetRisk))

```

Command Window

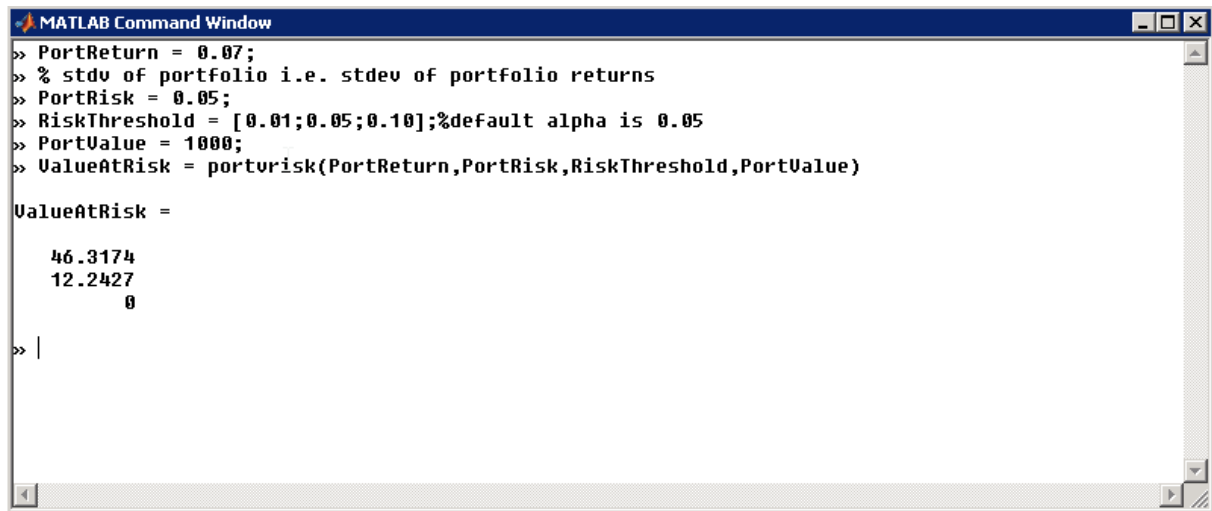
```

Efficient Portfolios for Target Risk =
               0          0.04
T1  6.31089e-30          0
T2   0.81921      0.77180
T3   0.18079      0.22820
fx >> |

```

Values at risk (VaR)

Parametric relative VaR



```
MATLAB Command Window
>> PortReturn = 0.07;
>> % stdv of portfolio i.e. stdev of portfolio returns
>> PortRisk = 0.05;
>> RiskThreshold = [0.01;0.05;0.10];%default alpha is 0.05
>> PortValue = 1000;
>> ValueAtRisk = porturisk(PortReturn,PortRisk,RiskThreshold,PortValue)

ValueAtRisk =

    46.3174
    12.2427
         0

>> |
```

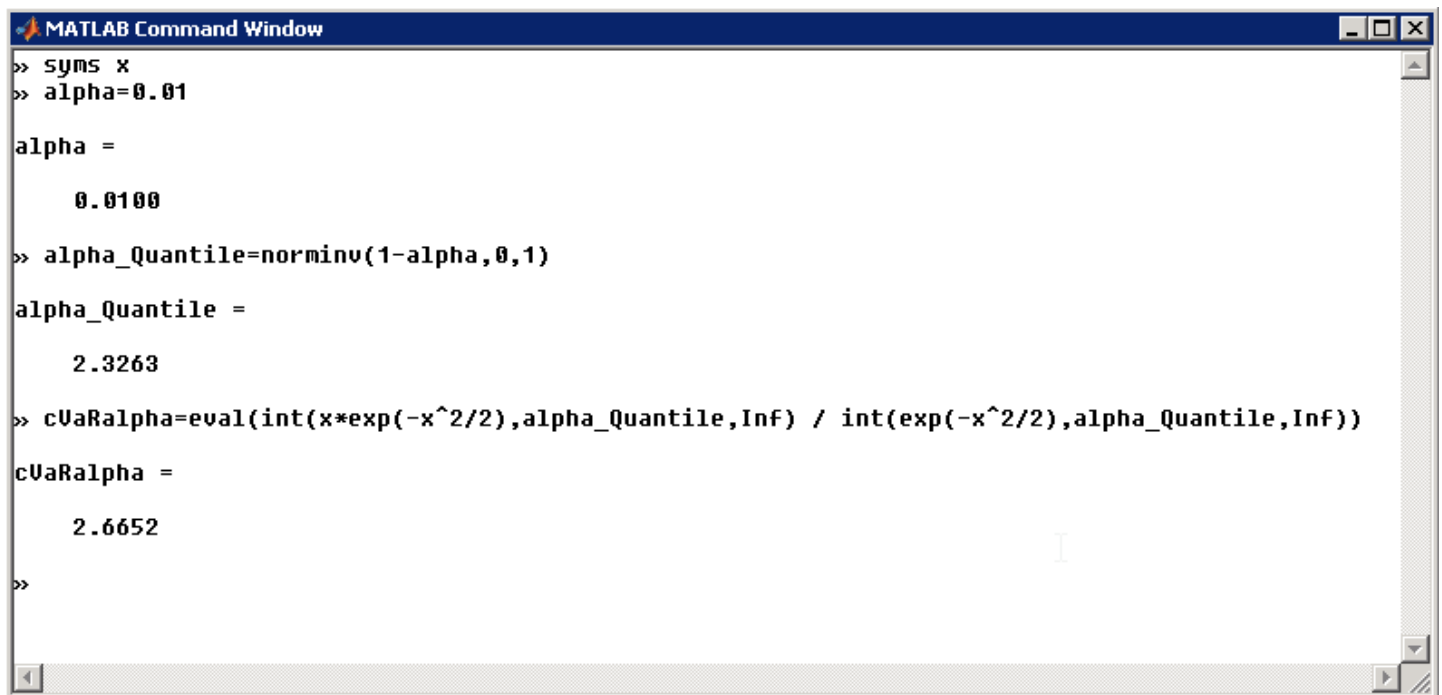
Parametric absolute VaR

non-Parametric (absolute) VaR

Condition (absolute) Normal VaR

We need the symbolic Toolbox for this first example:

Soit une VaR suivant une loi Normale qui a été centrée réduite. L'espérance conditionnelle de perte au-delà de 99% (donc au seuil de $\alpha = 1\%$) est alors donnée par:

A screenshot of the MATLAB Command Window. The window has a blue title bar with the MATLAB logo and the text "MATLAB Command Window". The command history shows the following steps: 1. Defining a symbolic variable x with 'syms x'. 2. Setting alpha to 0.01 with 'alpha=0.01'. 3. Calculating the quantile 'alpha_Quantile' using 'norminv(1-alpha,0,1)', which returns 2.3263. 4. Calculating the conditional value at risk 'cVaRalpha' using the formula 'eval(int(x*exp(-x^2/2),alpha_Quantile,Inf) / int(exp(-x^2/2),alpha_Quantile,Inf))', which returns 2.6652. The cursor is at the end of the last command line.

```
>> syms x
>> alpha=0.01

alpha =

    0.0100

>> alpha_Quantile=norminv(1-alpha,0,1)

alpha_Quantile =

    2.3263

>> cVaRalpha=eval(int(x*exp(-x^2/2),alpha_Quantile,Inf) / int(exp(-x^2/2),alpha_Quantile,Inf))

cVaRalpha =

    2.6652

>>
```

Monte-Carlo (absolute) VaR

Delta-Normal (absolute) VaR

Historical (absolute) VaR

Credit VaR

Operational VaR

Variance-Covariance VaR

Marginal VaR

CVaR

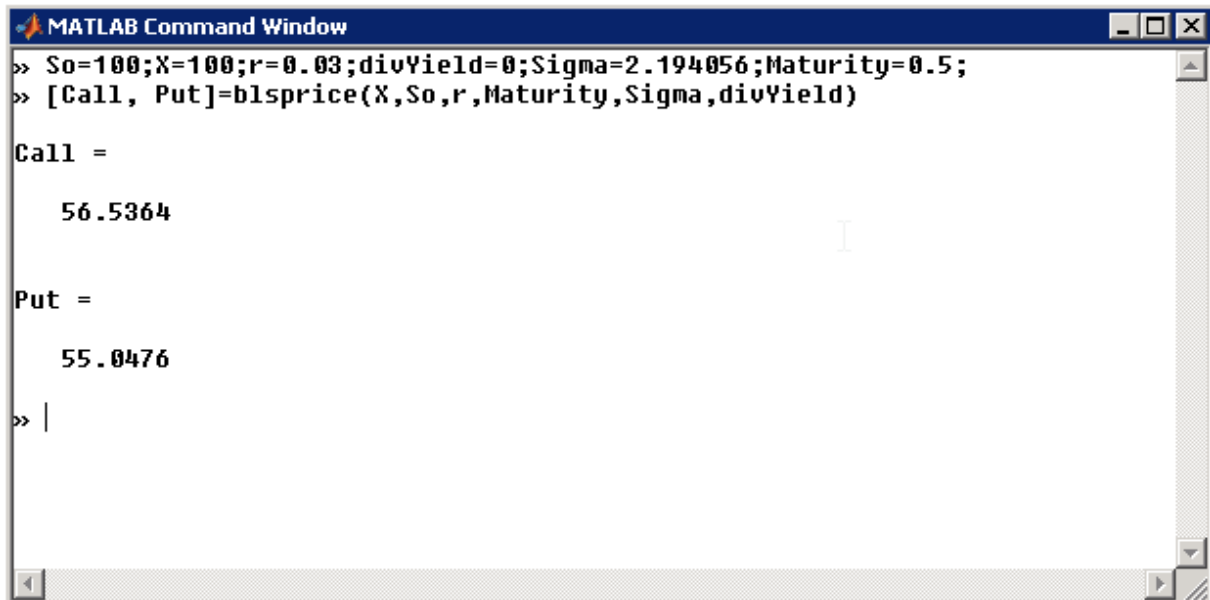
VaR backtesting

SAMPLE

Options Pricing & Monitoring

Crazy thing... MATLAB seems to have function only to price European options...!!!

Pricing European Call & Put



```
MATLAB Command Window
>> So=100;X=100;r=0.03;divYield=0;Sigma=2.194056;Maturity=0.5;
>> [Call, Put]=blsprice(X,So,r,Maturity,Sigma,divYield)

Call =

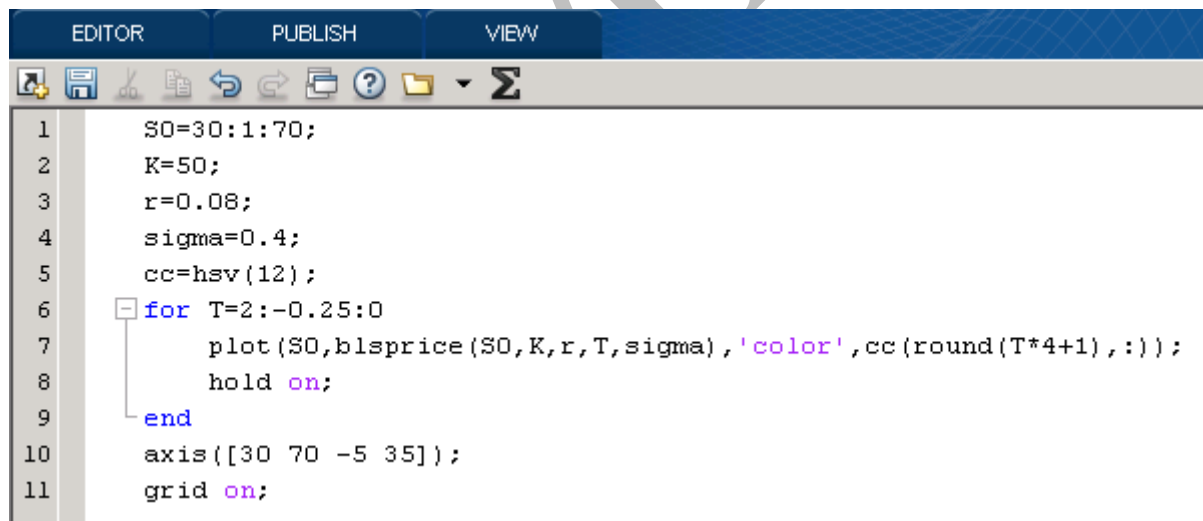
    56.5364

Put =

    55.0476

>> |
```

Or another very useful point of view:



```
EDITOR PUBLISH VIEW
1 S0=30:1:70;
2 K=50;
3 r=0.08;
4 sigma=0.4;
5 cc=hsb(12);
6 for T=2:-0.25:0
7     plot(S0,blsprice(S0,K,r,T,sigma),'color',cc(round(T*4+1),:));
8     hold on;
9 end
10 axis([30 70 -5 35]);
11 grid on;
```

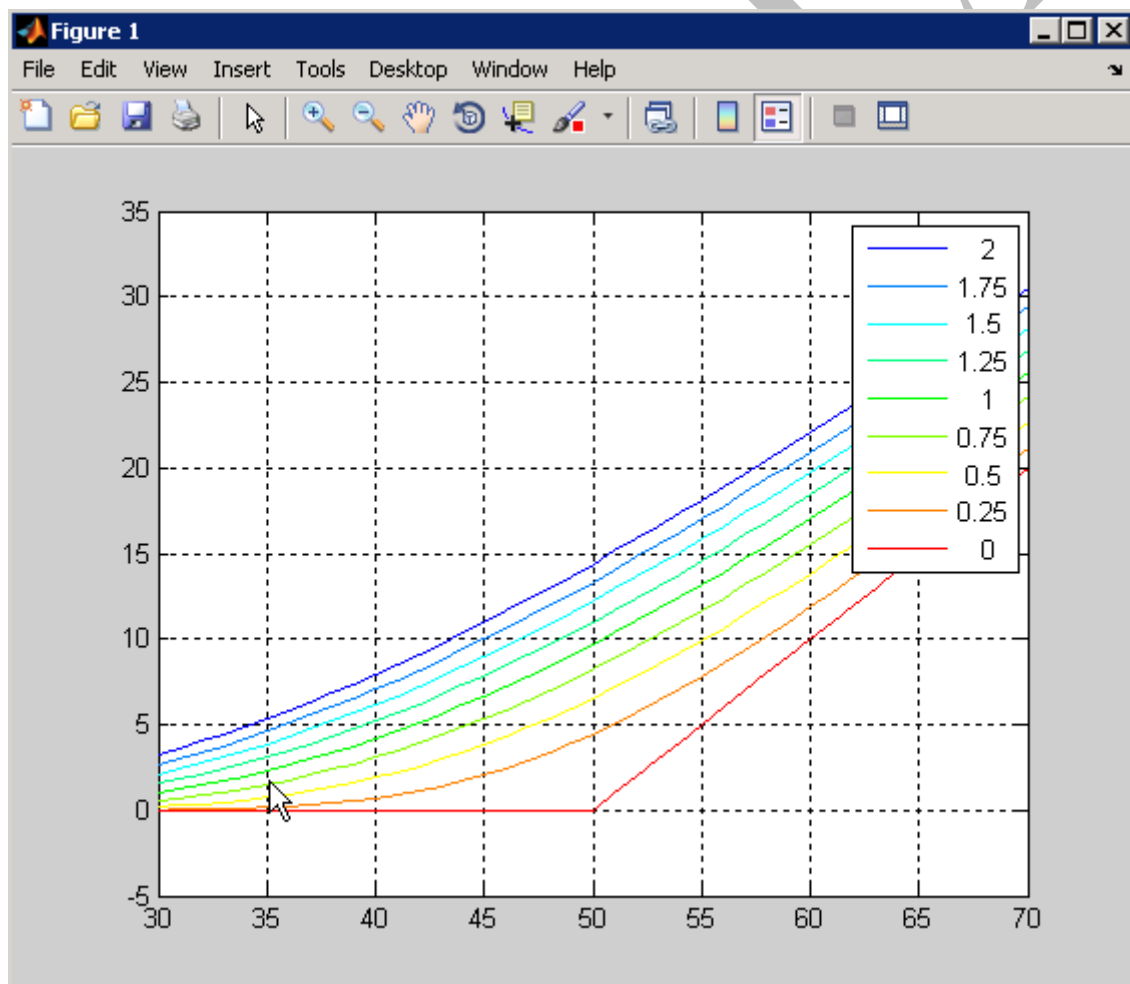
That gives:

```

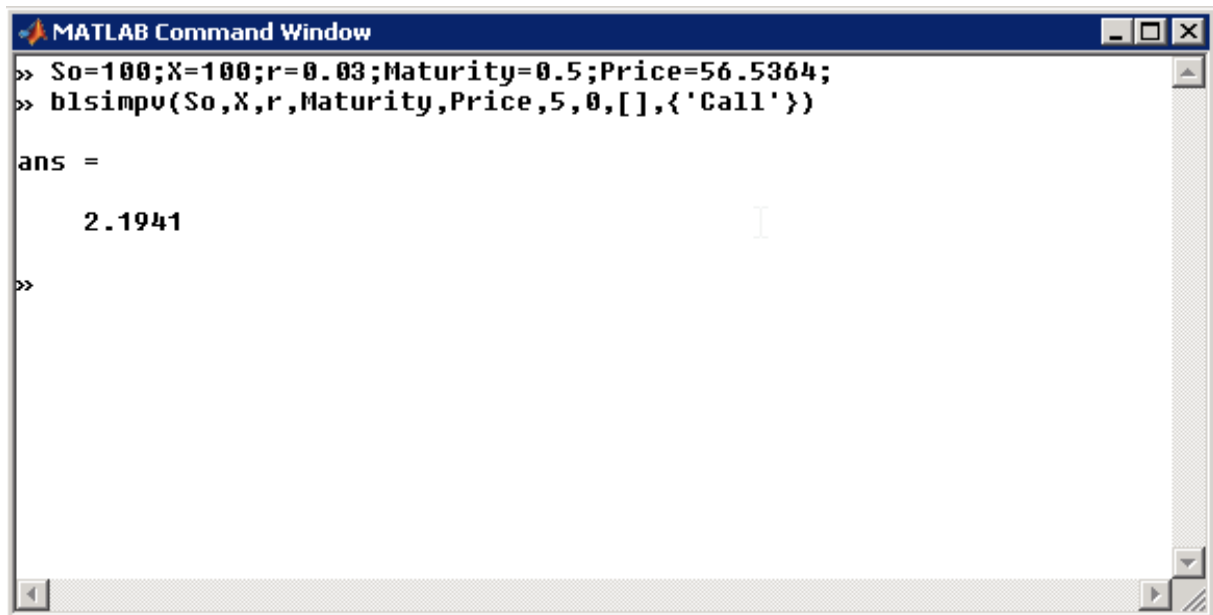
EDITOR PUBLISH VIEW
1 S0=30:1:70;
2 K=50;
3 r=0.08;
4 sigma=0.4;
5 cc=hsv(12);
6 for T=2:-0.25:0
7     plot(S0,bisprice(S0,K,r,T,sigma),'color',cc(round(T*4+1),:));
8     hold on;
9 end
10 axis([30 70 -5 35]);
11 grid on;
12 legend(cellstr(num2str(2:-0.25:0)'))

```

This gives:



European Call & Put implied volatility



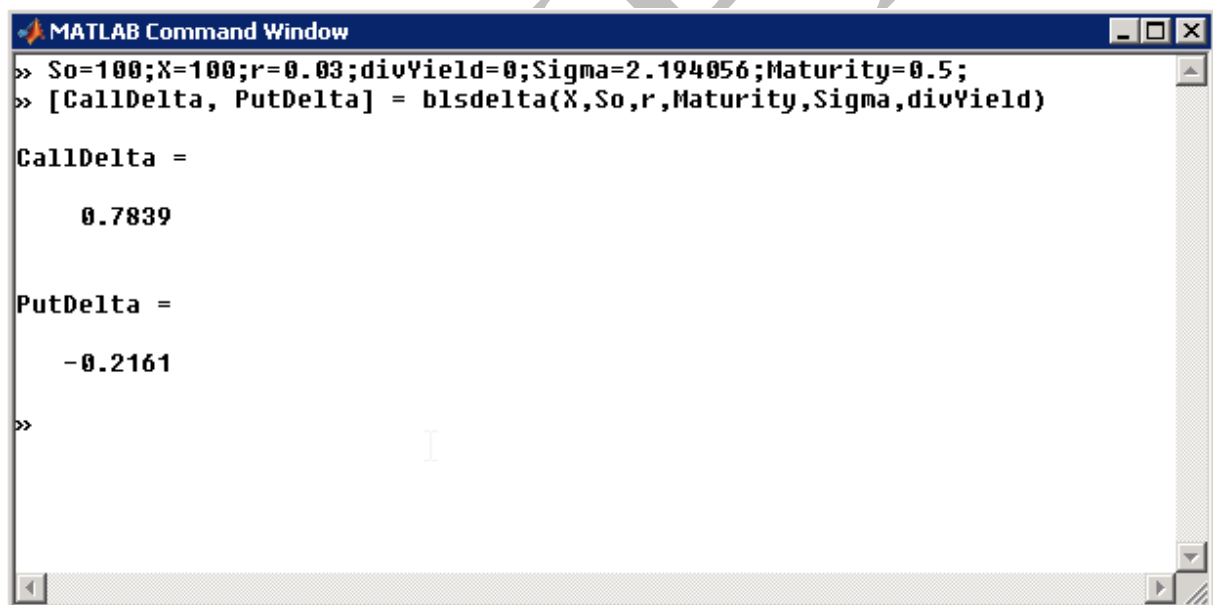
```
MATLAB Command Window
>> So=100;X=100;r=0.03;Maturity=0.5;Price=56.5364;
>> blsimpv(So,X,r,Maturity,Price,5,0,[],{'Call'})

ans =

    2.1941
>>
```

A screenshot of the MATLAB Command Window. The title bar reads "MATLAB Command Window". The command prompt shows two lines of code: `>> So=100;X=100;r=0.03;Maturity=0.5;Price=56.5364;` and `>> blsimpv(So,X,r,Maturity,Price,5,0,[],{'Call'})`. The output is `ans =` followed by `2.1941` on the next line. The prompt `>>` is visible at the bottom.

European Call & Put delta



```
MATLAB Command Window
>> So=100;X=100;r=0.03;divYield=0;Sigma=2.194056;Maturity=0.5;
>> [CallDelta, PutDelta] = blsdelta(X,So,r,Maturity,Sigma,divYield)

CallDelta =

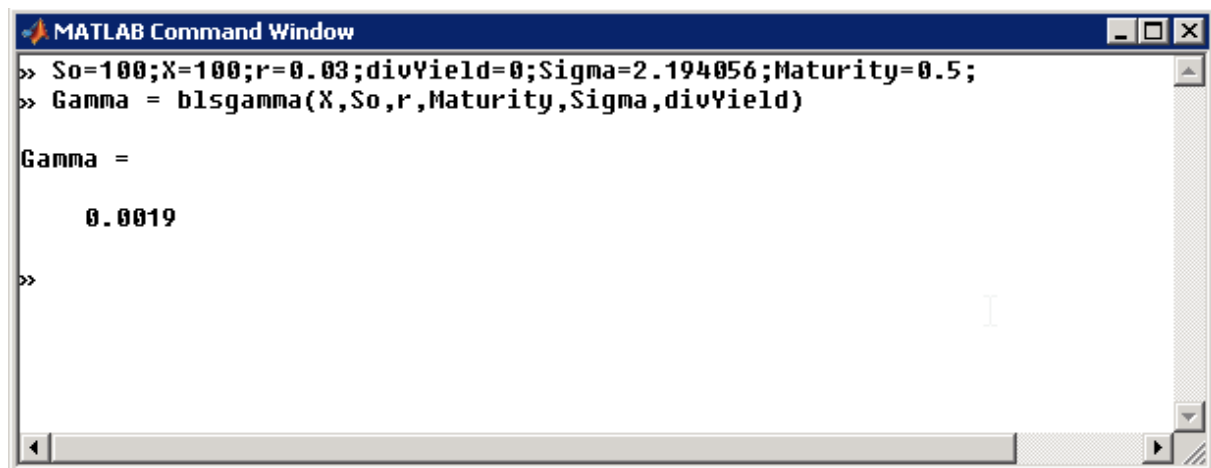
    0.7839

PutDelta =

   -0.2161
>>
```

A screenshot of the MATLAB Command Window. The title bar reads "MATLAB Command Window". The command prompt shows two lines of code: `>> So=100;X=100;r=0.03;divYield=0;Sigma=2.194056;Maturity=0.5;` and `>> [CallDelta, PutDelta] = blsdelta(X,So,r,Maturity,Sigma,divYield)`. The output shows `CallDelta =` followed by `0.7839` on the next line, and `PutDelta =` followed by `-0.2161` on the next line. The prompt `>>` is visible at the bottom.

European Option (call/put) Gamma

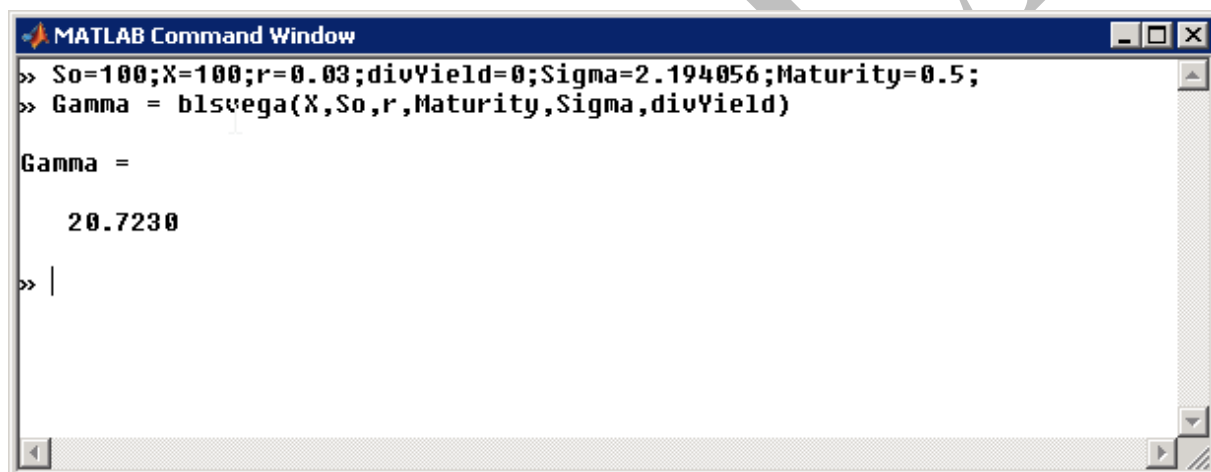


```
MATLAB Command Window
>> So=100;X=100;r=0.03;divYield=0;Sigma=2.194056;Maturity=0.5;
>> Gamma = bslgamma(X,So,r,Maturity,Sigma,divYield)

Gamma =

    0.0019
>>
```

European Option (call/put) Vega

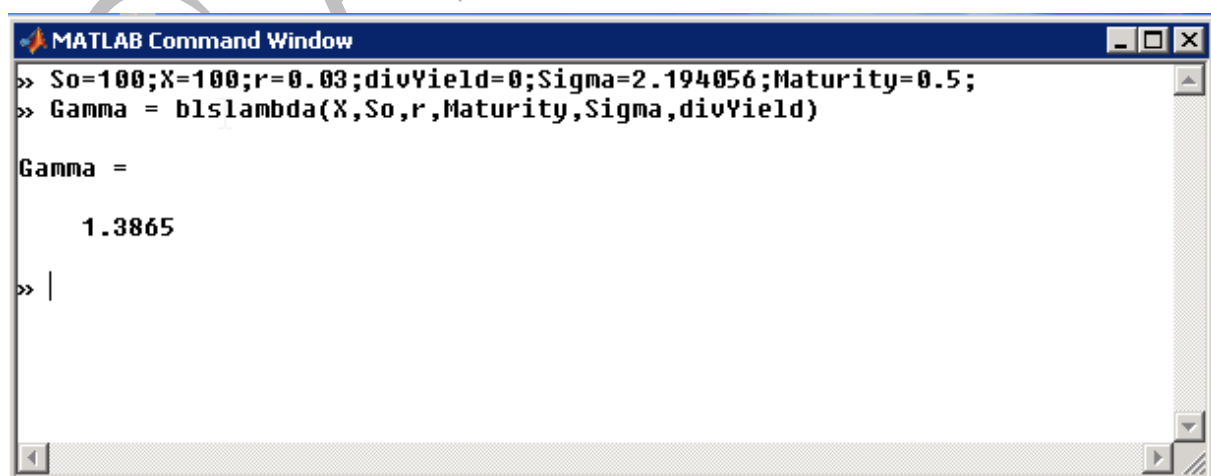


```
MATLAB Command Window
>> So=100;X=100;r=0.03;divYield=0;Sigma=2.194056;Maturity=0.5;
>> Gamma = bslvega(X,So,r,Maturity,Sigma,divYield)

Gamma =

    20.7230
>> |
```

Option european lambda

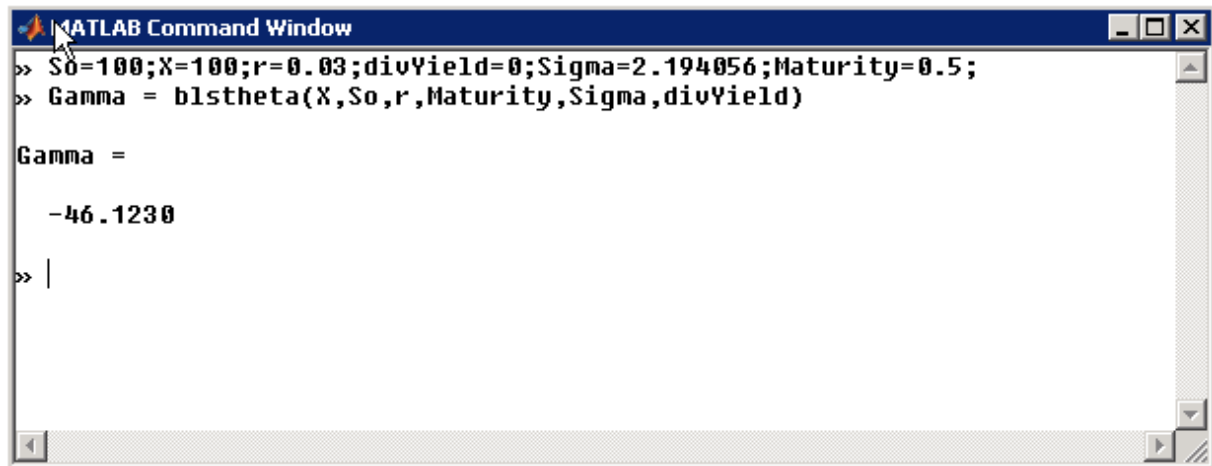


```
MATLAB Command Window
>> So=100;X=100;r=0.03;divYield=0;Sigma=2.194056;Maturity=0.5;
>> Gamma = bslambda(X,So,r,Maturity,Sigma,divYield)

Gamma =

    1.3865
>> |
```

Option european theta



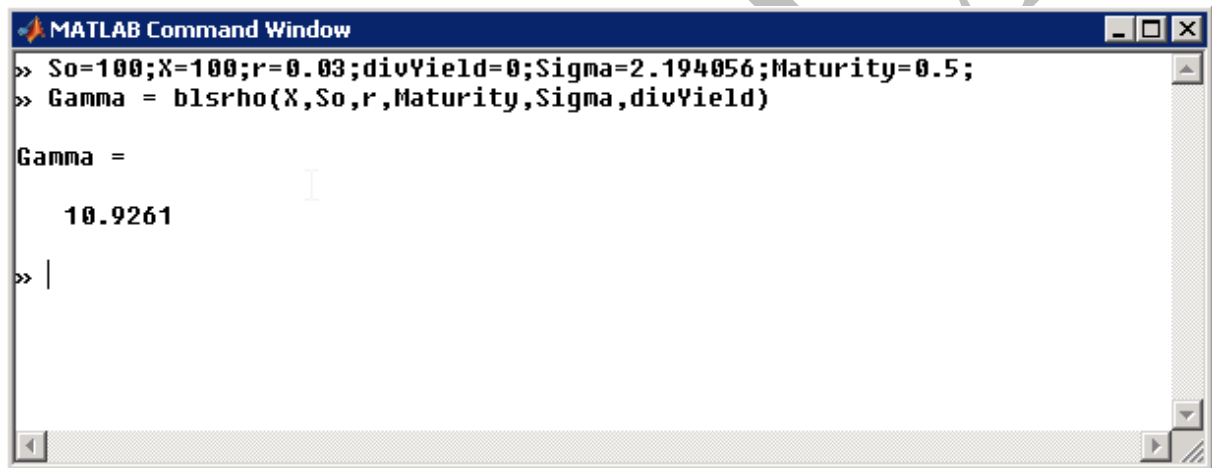
```

MATLAB Command Window
>> So=100;X=100;r=0.03;divYield=0;Sigma=2.194056;Maturity=0.5;
>> Gamma = blstheta(X,So,r,Maturity,Sigma,divYield)

Gamma =

    -46.1230
>> |
  
```

Option european rho



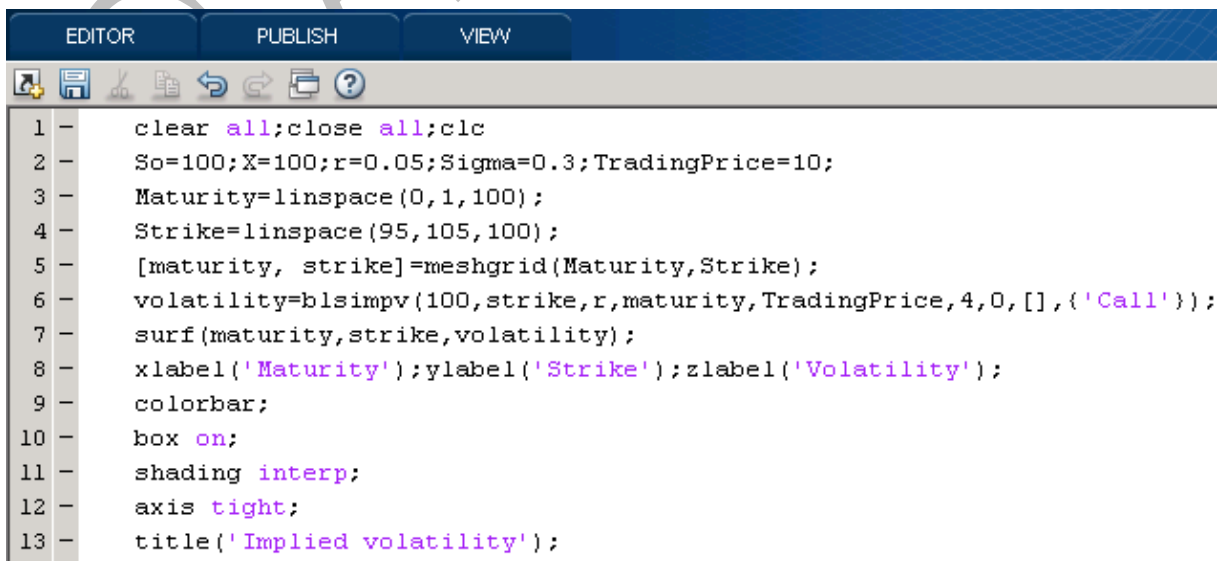
```

MATLAB Command Window
>> So=100;X=100;r=0.03;divYield=0;Sigma=2.194056;Maturity=0.5;
>> Gamma = blsrho(X,So,r,Maturity,Sigma,divYield)

Gamma =

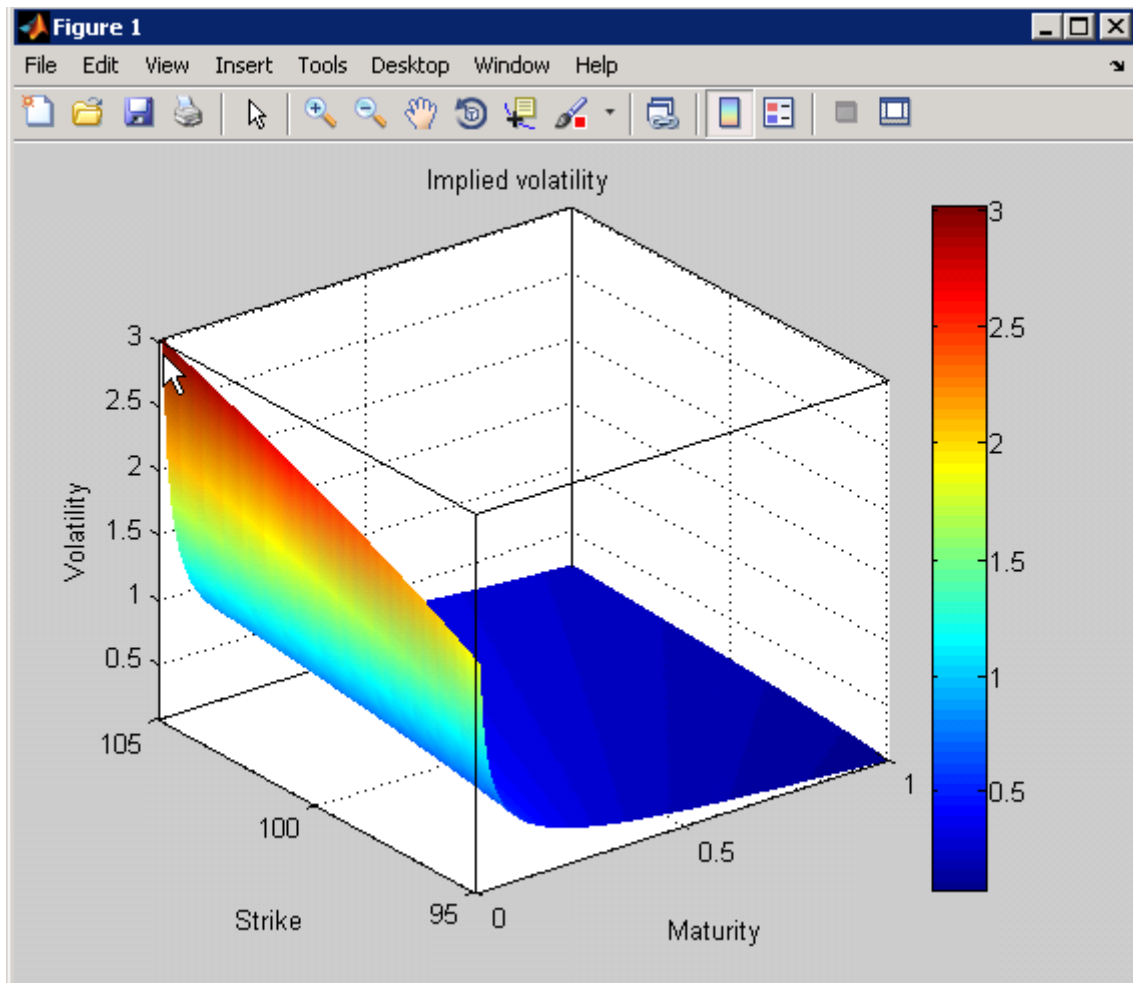
    10.9261
>> |
  
```

Implied volatility surface of an European option



```

EDITOR PUBLISH VIEW
1 - clear all;close all;clc
2 - So=100;X=100;r=0.05;Sigma=0.3;TradingPrice=10;
3 - Maturity=linspace(0,1,100);
4 - Strike=linspace(95,105,100);
5 - [maturity, strike]=meshgrid(Maturity,Strike);
6 - volatility=blsimpv(100,strike,r,maturity,TradingPrice,4,0,[],{'Call'});
7 - surf(maturity,strike,volatility);
8 - xlabel('Maturity');ylabel('Strike');zlabel('Volatility');
9 - colorbar;
10 - box on;
11 - shading interp;
12 - axis tight;
13 - title('Implied volatility');
  
```

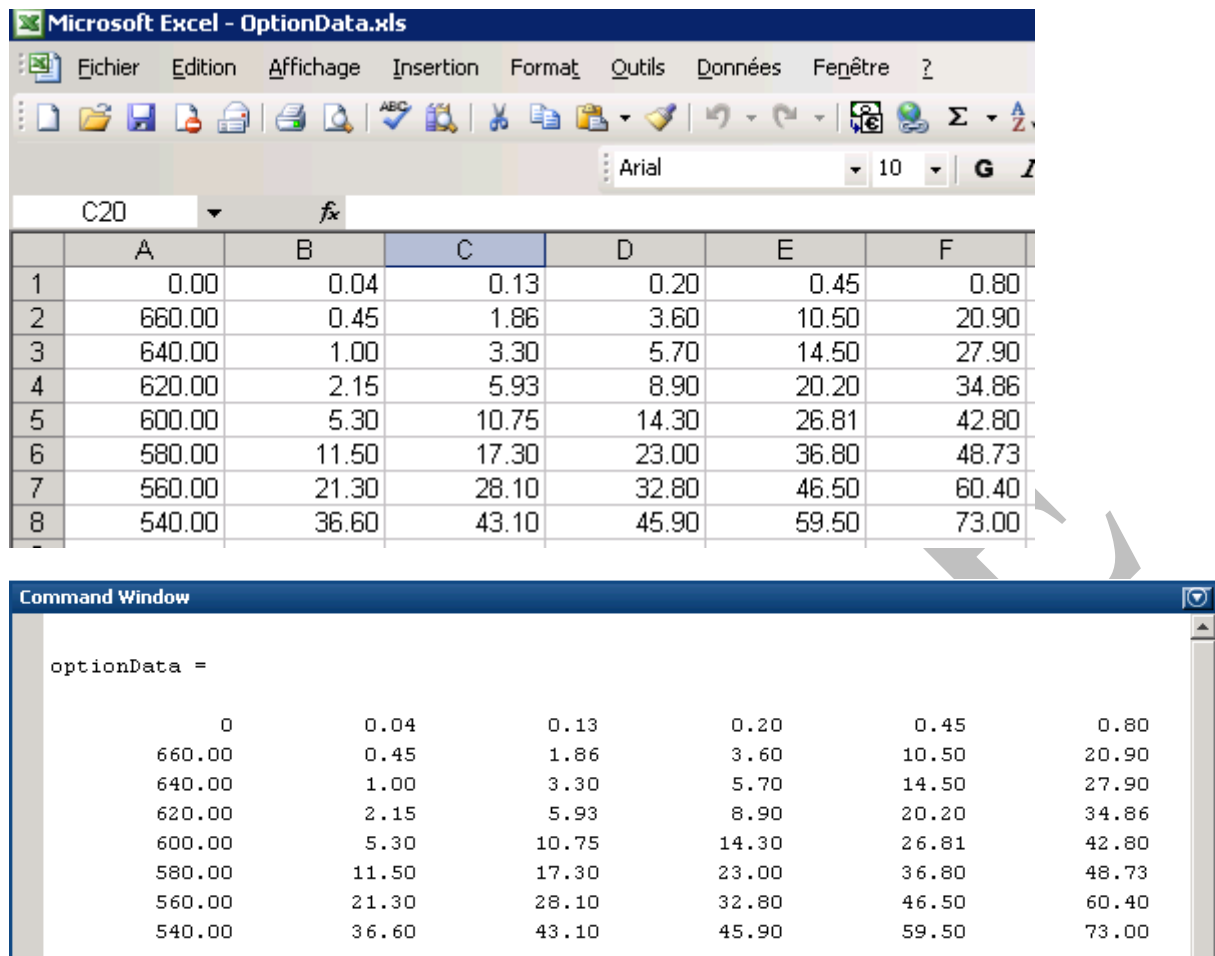


Volatility smile of an European option

```

EDITOR    PUBLISH    VIEW
[Icons]
1 -      clear all;clc;
2 -      optionData=xlsread('OptionData.xls');
3 -      % OptionData ::::
4 -      % the first row represents maturity
5 -      % the first column, strike prices beginning from 2nd row
6 -      % the rest are option prices
7 -      optionData
8 -      % Input parameter
9 -      So=570;
10 -     div=0.0;
11 -     r=0.01;
12 -     PutorCall='Call';
13 -     vec_X=optionData(2:end,1);% get strikes vector
14 -     vec_T=optionData(1,2:end);% get maturity vector
15 -     priceMatrix=optionData(2:end,2:end);
16 -     Volatility_Smile_And_Surface(PutorCall,So,div,r,vec_X,vec_T,priceMatrix)

```



```

EDITOR PUBLISH VIEW
1 function Volatility_Smile_And_Surface(PutOrCall,S0,div,r,vec_X,vec_T,priceMatrix)
2     NumX = length(vec_X);
3     NumM = size(vec_T,2);
4     for i = 1:NumM
5         T = vec_T(1,i);
6         % Loop for the different strike prices
7         for iStrike = 1:NumX
8             price = priceMatrix(iStrike,i);
9             X = vec_X(iStrike);
10            try
11                % Calculation of the implied volatility
12                %ImpliedVolatility(PutOrCall,S0,X,r,T,div,Optionprice)
13                implVolM(iStrike,i) = ImpliedVolatility(PutOrCall,S0,X,r,T,div,price);
14                % Add nan to the value matrix if the function was not able to
15                % solve for the implied volatility.
16            catch MExc
17                implVolM(iStrike,i) = nan;
18            end
19        end
20    end
21    implVolM
22    % Compute vector for moneyness
23    vec_moneyness = log(S0./vec_X);
24
25    % Plot volatility smile
26    figure(1);
27    plot(vec_X, implVolM(:,1),'r-','LineWidth',2);
28    hold on;
29    plot(vec_X, implVolM(:,NumM),'b-','LineWidth',2);
30    title('Volatility Smile');xlabel('Strike');ylabel('Volatility');
31    % Set the range of axis: [x_min x_max y_min y_max]
32    axis([min(vec_X)-10 max(vec_X)+10 min(min(implVolM))-0.1 max(max(implVolM))+0.1]);
33    legend('Implied Volatility for short Maturity',...
34          'Implied Volatility for long Maturity','location','NorthWest');
35    % Plot volatility surface
36    figure(2);
37    surf(vec_moneyness,vec_T,implVolM);
38    title('Volatility Surface');
39    xlabel('Moneyness');ylabel('Maturity (in years)'); zlabel('Volatility');
40    % Set the y-axis to evolve backwards
41    set(gca,'YDir','reverse');
42    colorbar;shading interp;
43 end

```

```

EDITOR PUBLISH VIEW
function [ IV ] = ImpliedVolatility(PutOrCall,S0,X,r,T,div,Optionprice )
1 - options = optimset('fzero');
2 - %TolX, 1e-7: tolerance level
3 - options = optimset(options, 'TolX', 1e-7);
4 - func = @(sigma)diffbls2price(PutOrCall,S0,X,r,sigma,T,div,Optionprice);
5 - % [0.001 5] interval for volatility. To change if necessary!
6 - IV = fzero(func,[0.001 5],options);
7 -
8 - end

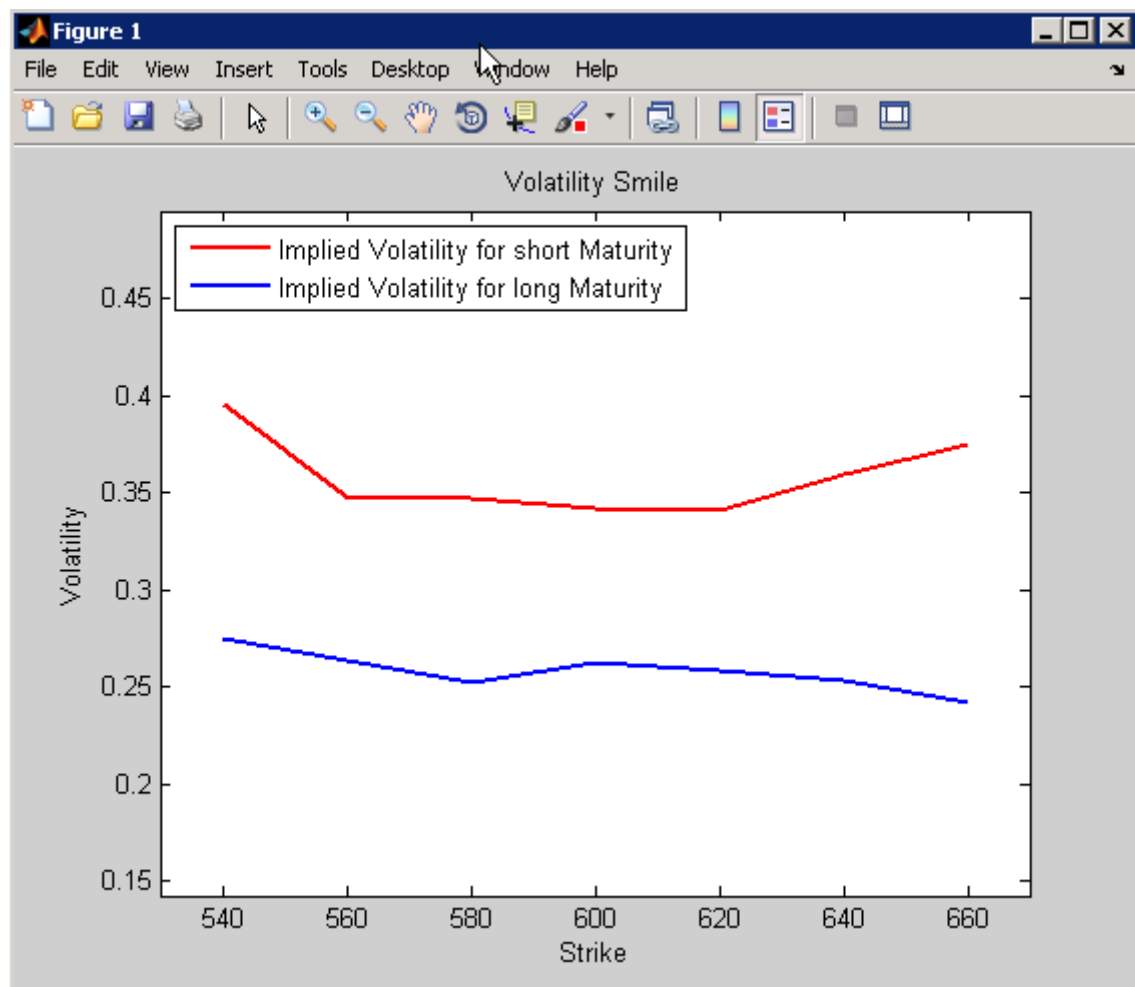
```

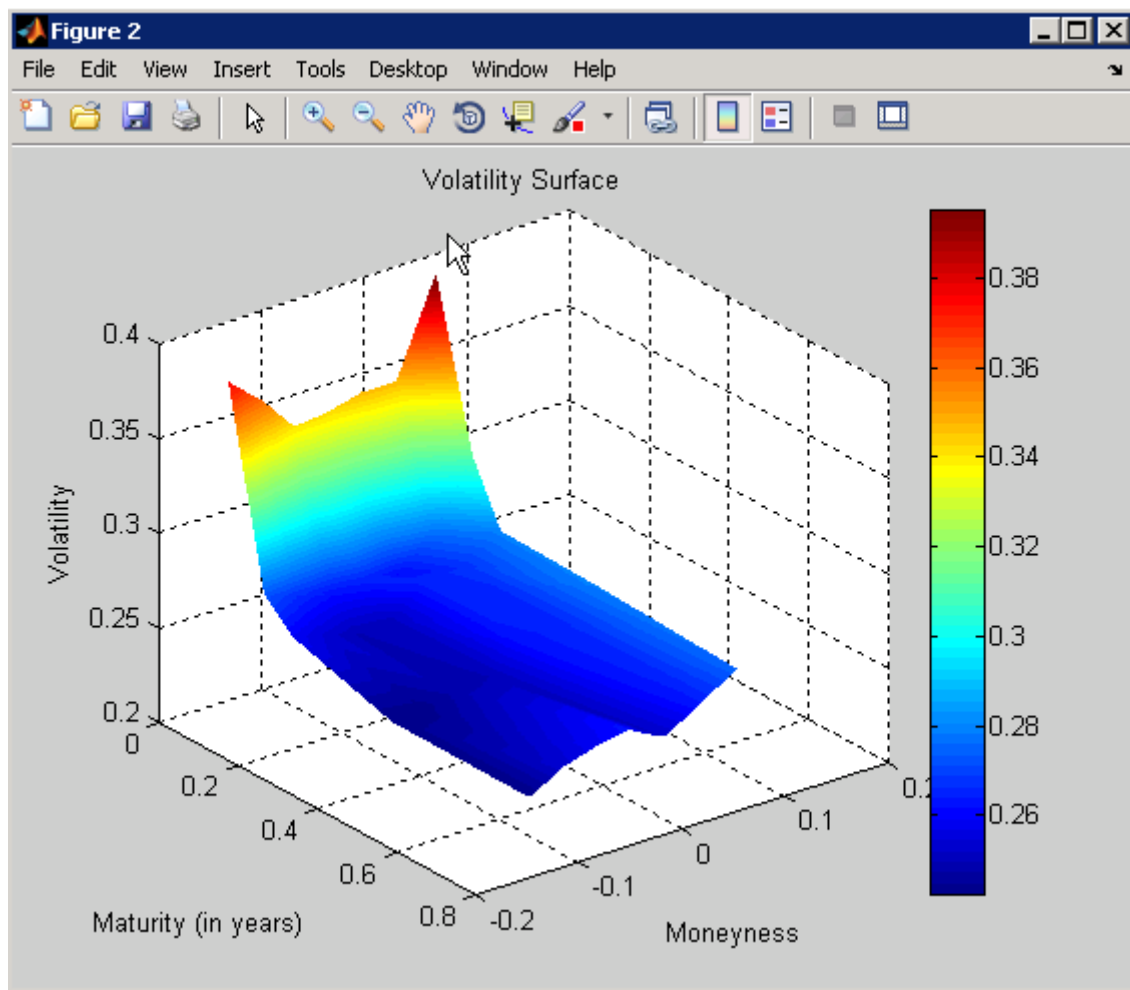
```

EDITOR PUBLISH VIEW
function [value] = diffbls2price(PutOrCall,S0,X,r,sigma,T,div,Optionprice)
1 - %computes the difference between the function
2 - %BlackScholes option value and the given option price.
3 - [call, put] = blsprice(S0,X,r,T,sigma,div);
4 - switch PutOrCall
5 -     case 'Call'
6 -         value = call - Optionprice;
7 -     case 'Put'
8 -         value = put - Optionprice;
9 -     otherwise
10 -         error('Check option type!');
11 -     end
12 - end
13 - end

```

Command Window					
implVolM =					
	0.38	0.27	0.26	0.24	0.24
	0.36	0.27	0.25	0.24	0.25
	0.34	0.26	0.25	0.25	0.26
	0.34	0.27	0.25	0.25	0.26
	0.35	0.26	0.27	0.26	0.25
	0.35	0.28	0.27	0.27	0.26
	0.40	0.31	0.28	0.28	0.28





CRR binomial European option pricing model

The last argument "1" means we want the price for a Call. Otherwise if we want the same for a Put we have to write "0".

```

MATLAB Command Window
>> [Price, Option]=binprice(100,100,0.05,1,1/4,Sigma,1)

Price =
    100.0000    116.1834    134.9859    156.8312    182.2119
         0      86.0708    100.0000    116.1834    134.9859
         0         0      74.0818     86.0708    100.0000
         0         0         0     63.7628     74.0818
         0         0         0         0     54.8812

Option =
    13.5240    22.9039    37.4549    58.0734    82.2119
         0      4.3230     8.6793    17.4256    34.9859
         0         0         0         0         0
         0         0         0         0         0
         0         0         0         0         0

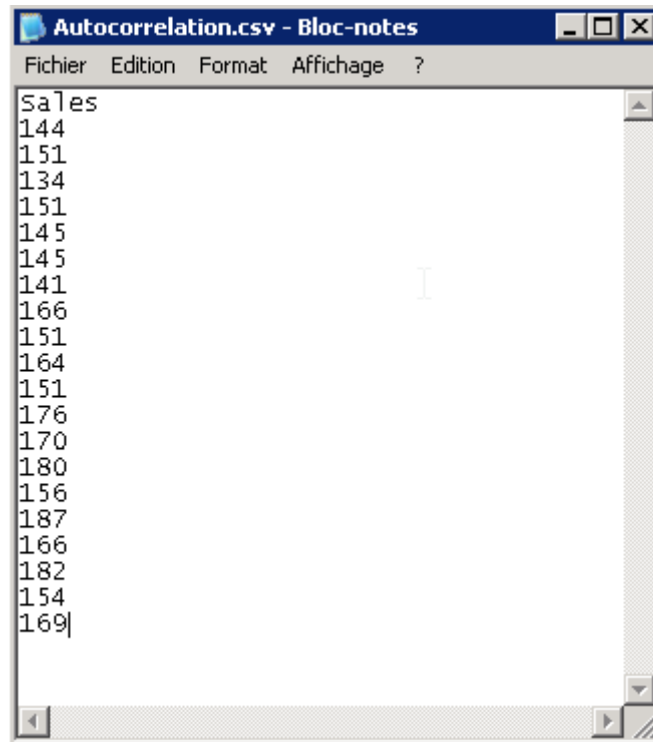
>> |

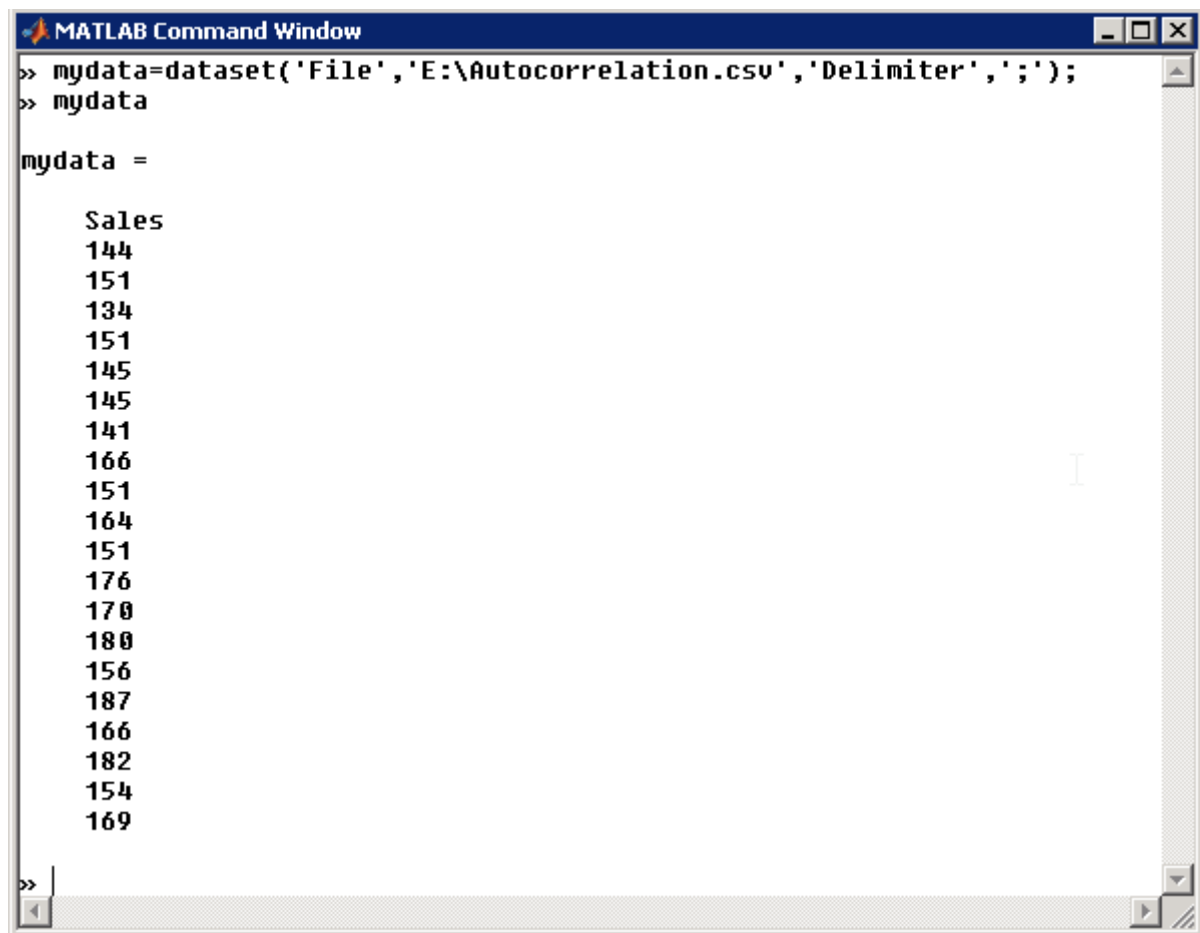
```

Time series

We will here focus only on ARIMA techniques!

Autocorrelation

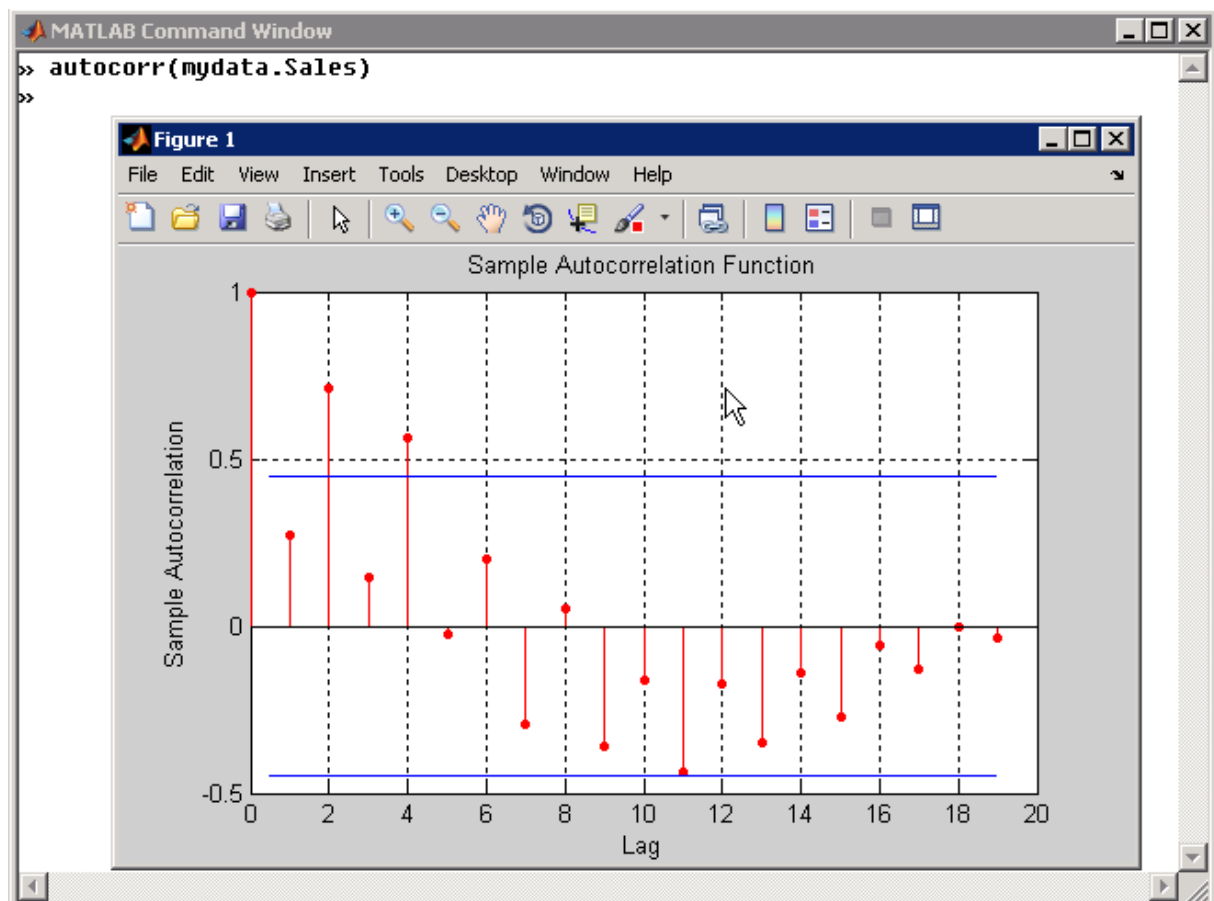


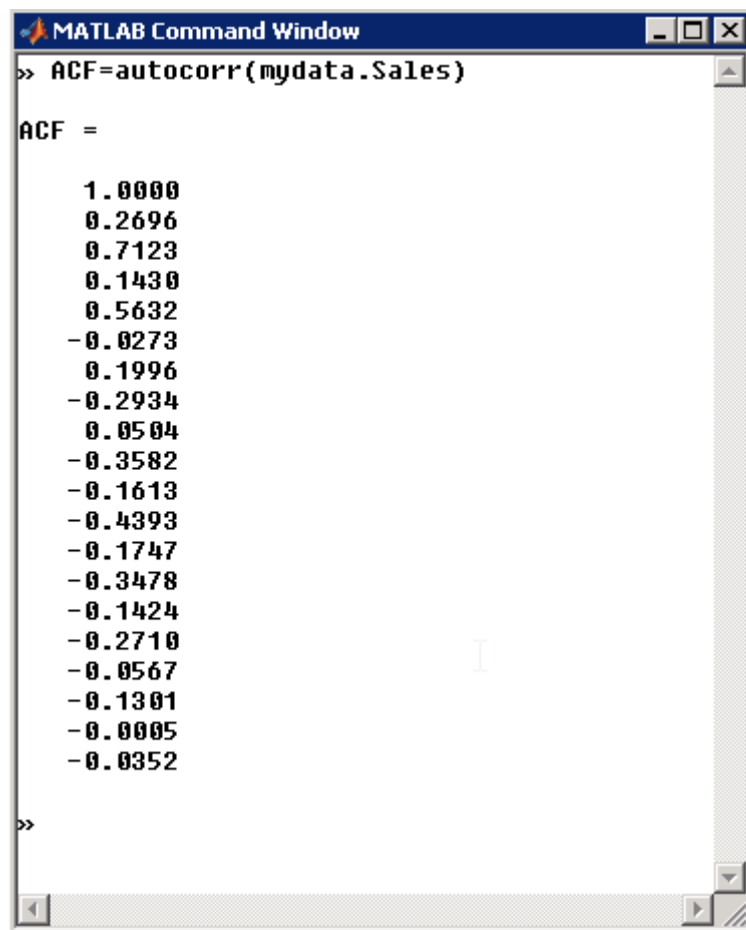


A screenshot of the MATLAB Command Window. The window title is "MATLAB Command Window". The command prompt shows the following commands and output:

```
>> mydata=dataset('File','E:\Autocorrelation.csv','Delimiter',';');  
>> mydata  
  
mydata =  
  
    Sales  
    144  
    151  
    134  
    151  
    145  
    145  
    141  
    166  
    151  
    164  
    151  
    176  
    170  
    180  
    156  
    187  
    166  
    182  
    154  
    169
```

The output displays a column vector of sales data. A large, light gray "SAMPLE" watermark is visible diagonally across the lower half of the image.





The MATLAB Command Window displays the command `ACF=autocorr(mydata.Sales)` and its output. The output is a vector of autocorrelation values for lags from 0 to 20. The values start at 1.0000 for lag 0 and generally decrease in magnitude as the lag increases, with some fluctuations in sign.

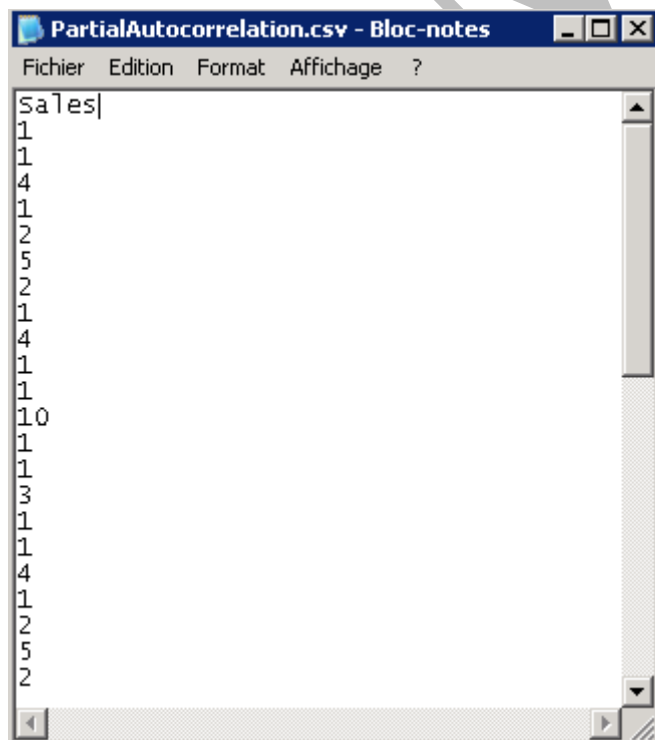
```
>> ACF=autocorr(mydata.Sales)

ACF =

    1.0000
    0.2696
    0.7123
    0.1430
    0.5632
   -0.0273
    0.1996
   -0.2934
    0.0504
   -0.3582
   -0.1613
   -0.4393
   -0.1747
   -0.3478
   -0.1424
   -0.2710
   -0.0567
   -0.1301
   -0.0005
   -0.0352

>>
```

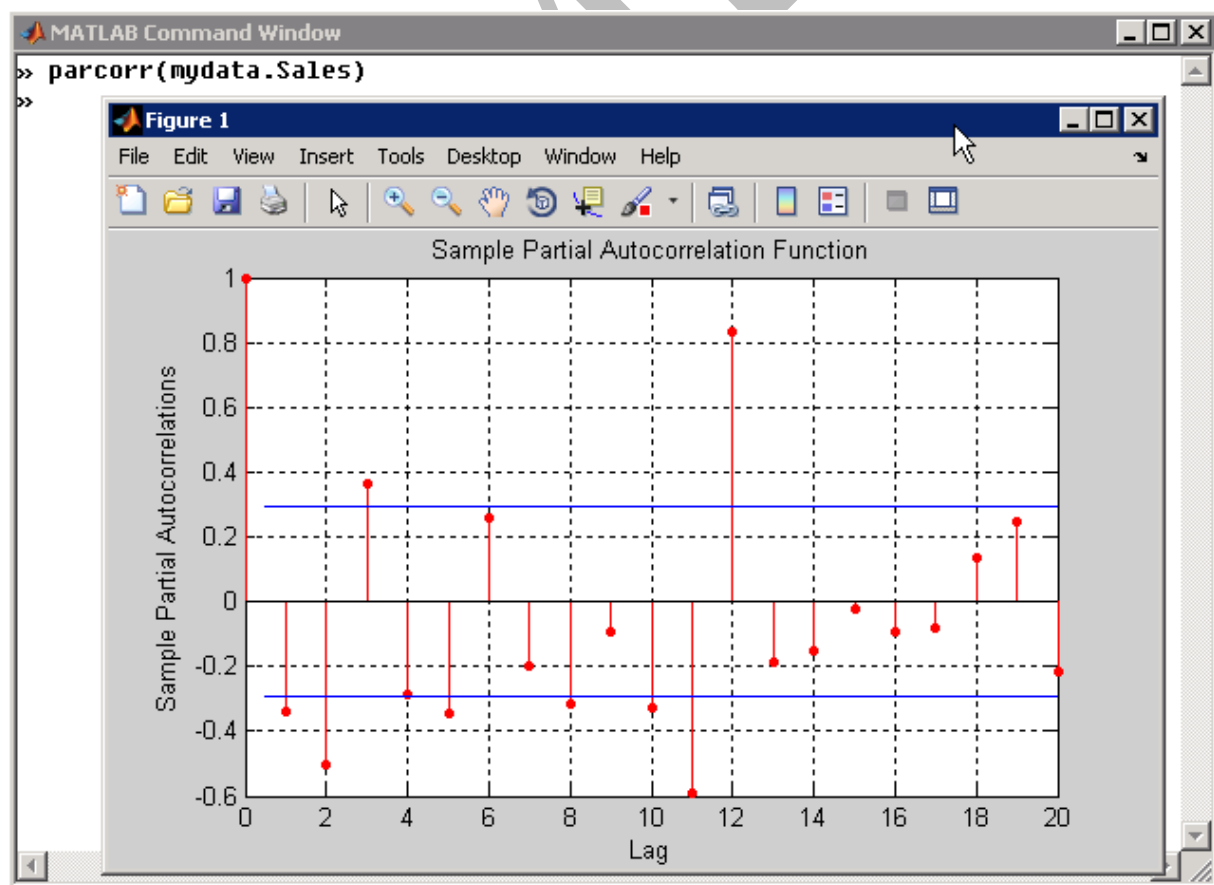
Partial autocorrelation

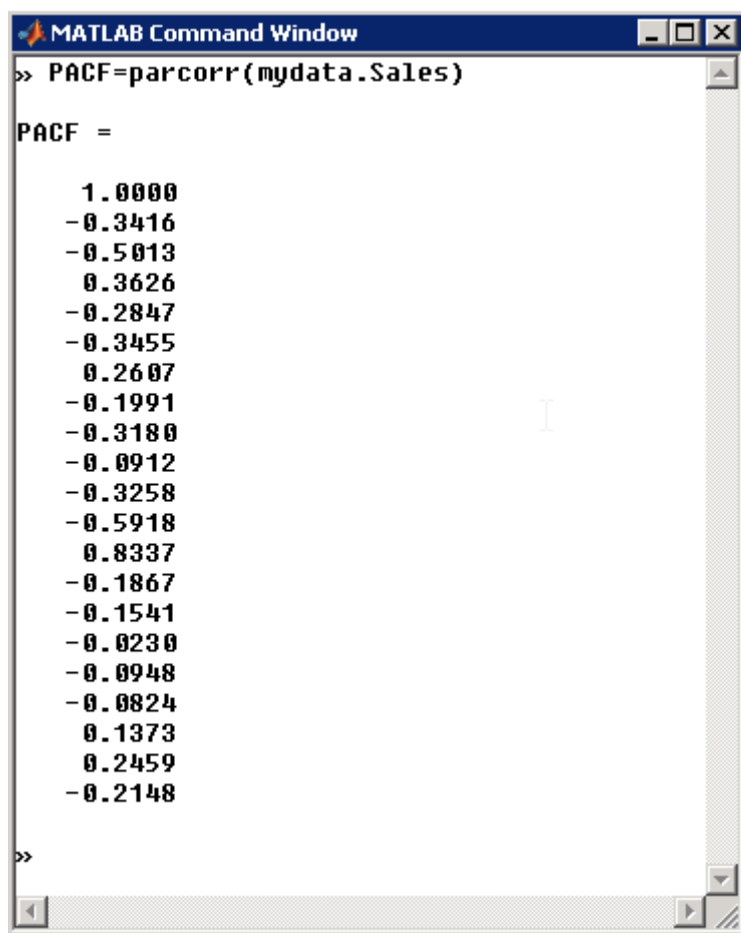


A text editor window titled 'PartialAutocorrelation.csv - Bloc-notes' shows the partial autocorrelation values for lags 1 through 20. The values are mostly small, with a notable peak at lag 10 (0.10) and lag 3 (0.03).

```
PartialAutocorrelation.csv - Bloc-notes
Fichier Edition Format Affichage ?

Sales|
1
1
4
1
2
5
2
1
4
1
1
1
10
1
1
3
1
1
4
1
2
5
2
```





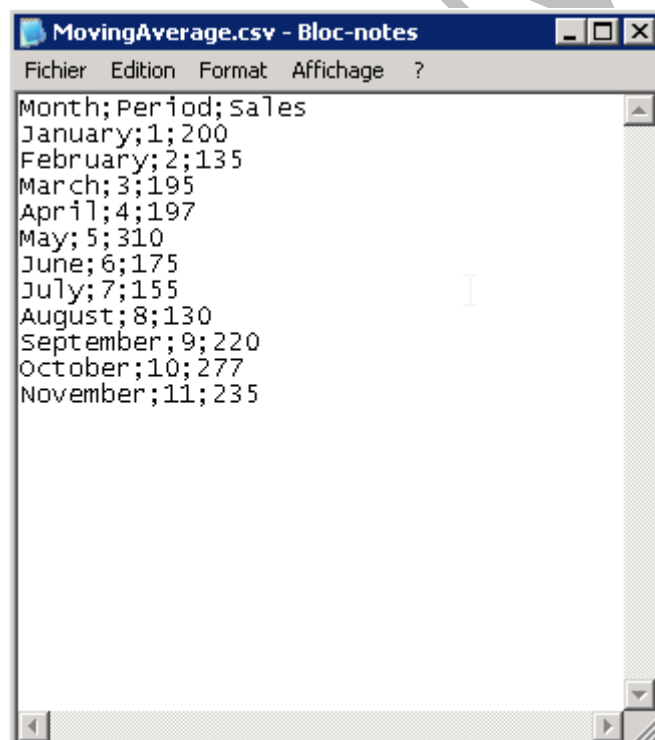
The image shows a MATLAB Command Window with the following content:

```
>> PACF=parcorr(mydata.Sales)

PACF =

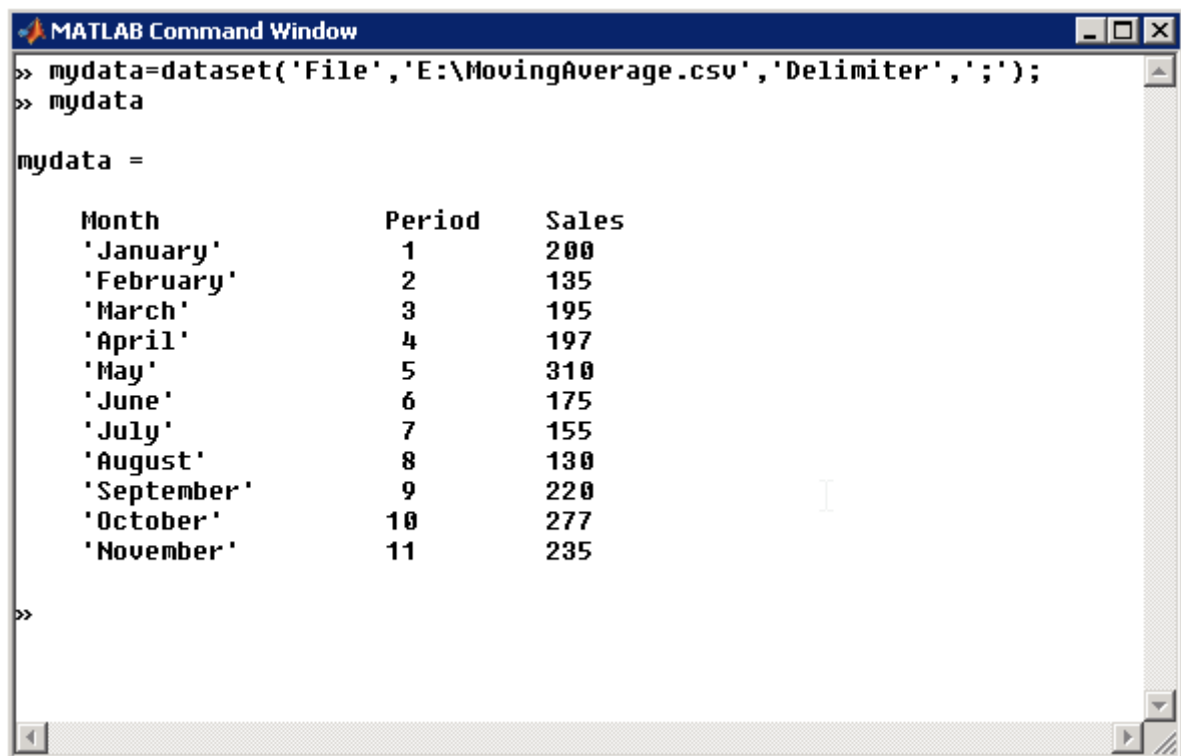
    1.0000
   -0.3416
   -0.5013
    0.3626
   -0.2847
   -0.3455
    0.2607
   -0.1991
   -0.3180
   -0.0912
   -0.3258
   -0.5918
    0.8337
   -0.1867
   -0.1541
   -0.0230
   -0.0948
   -0.0824
    0.1373
    0.2459
   -0.2148
```

Moving average MM3 (without ARIMA)



The image shows a text editor window titled "MovingAverage.csv - Bloc-notes" containing the following data:

Month	Period	Sales
January	1	200
February	2	135
March	3	195
April	4	197
May	5	310
June	6	175
July	7	155
August	8	130
September	9	220
October	10	277
November	11	235



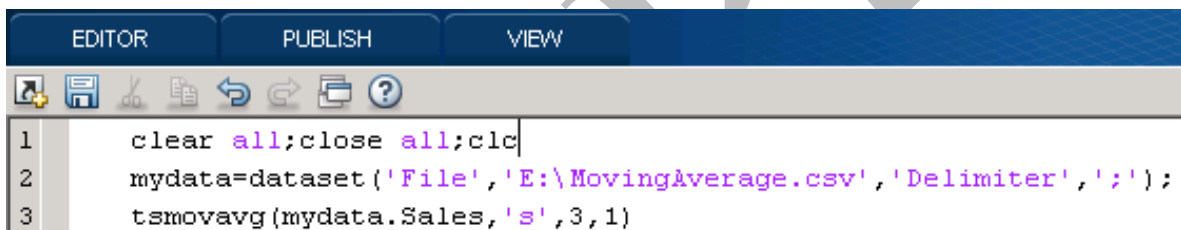
MATLAB Command Window

```
>> mydata=dataset('File','E:\MovingAverage.csv','Delimiter',';');
>> mydata
```

mydata =

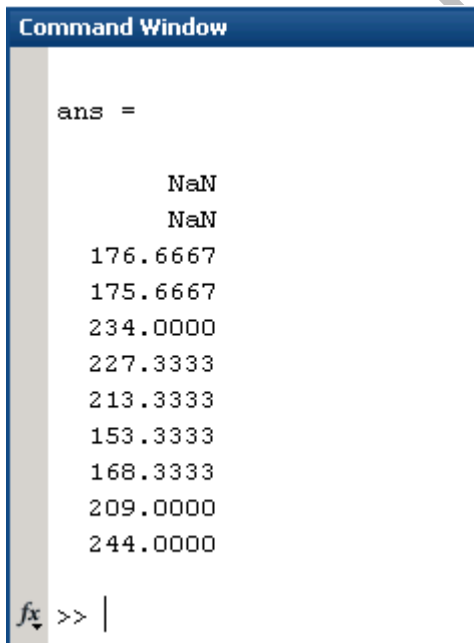
Month	Period	Sales
'January'	1	200
'February'	2	135
'March'	3	195
'April'	4	197
'May'	5	310
'June'	6	175
'July'	7	155
'August'	8	130
'September'	9	220
'October'	10	277
'November'	11	235

```
>>
```



EDITOR **PUBLISH** **VIEW**

```
1 clear all;close all;clc
2 mydata=dataset('File','E:\MovingAverage.csv','Delimiter',';');
3 tsmovavg(mydata.Sales,'s',3,1)
```



Command Window

```
ans =
```

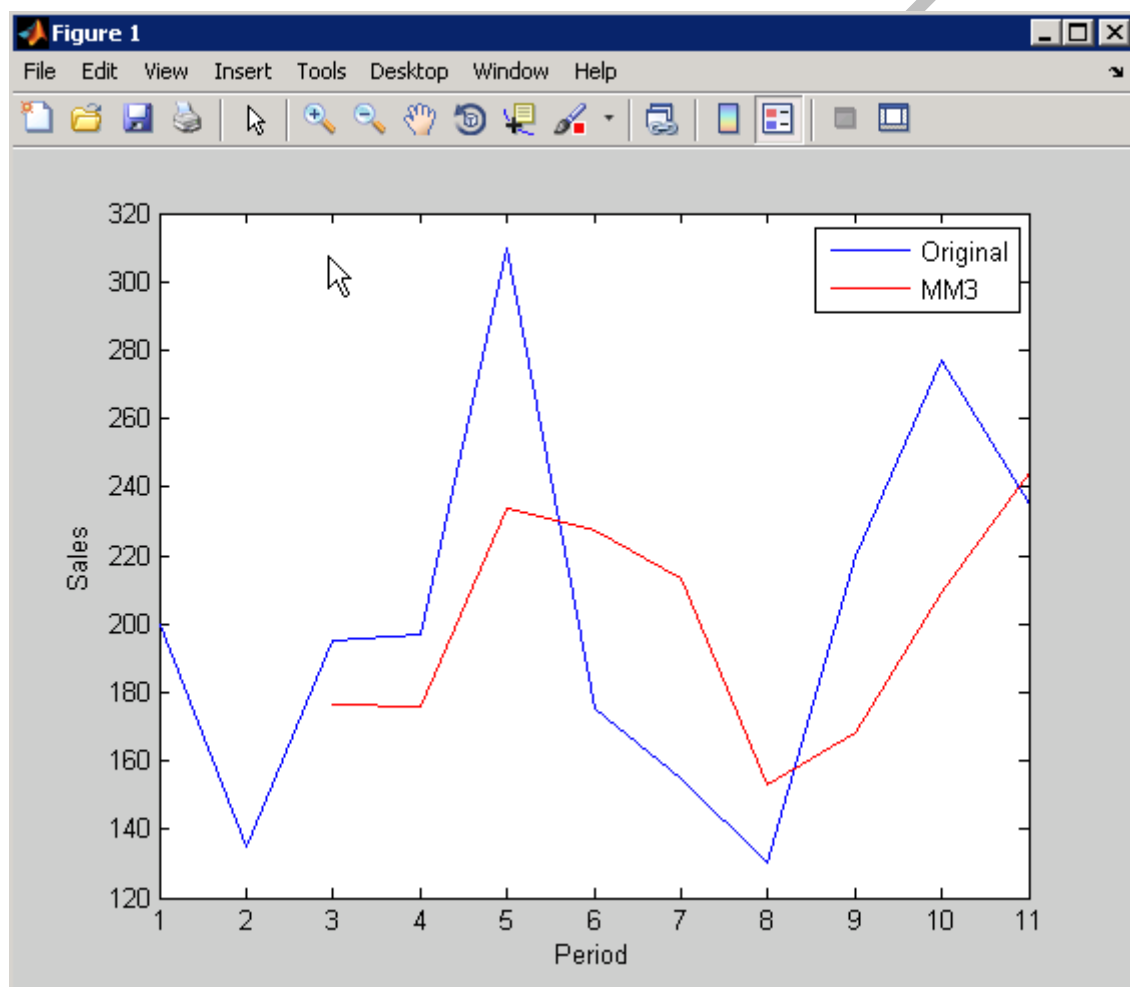
NaN
NaN
176.6667
175.6667
234.0000
227.3333
213.3333
153.3333
168.3333
209.0000
244.0000

```
fx >> |
```

```

EDITOR PUBLISH VIEW
1 clear all;close all;clc
2 mydata=dataset('File','E:\MovingAverage.csv','Delimiter',';');
3 tsmovavg(mydata.Sales,'s',3,1)
4 plot(mydata.Period,mydata.Sales,'b');
5 hold on;
6 plot(mydata.Period,tsmovavg(mydata.Sales,'s',3,1),'r');
7 xlabel('Period');ylabel('Sales');
8 legend('Original','MM3');

```



Simple Exponential Weighted Moving Average (EWMA)

```

EDITOR PUBLISH VIEW
1 clear all;close all;clc
2 Y=[143 152 161 139 137 174 142 141 162 180 164 171 206 193 207 218 229 225 204 227 223 242 239 266];
3 tsmovavg(Y,'e',2)

```

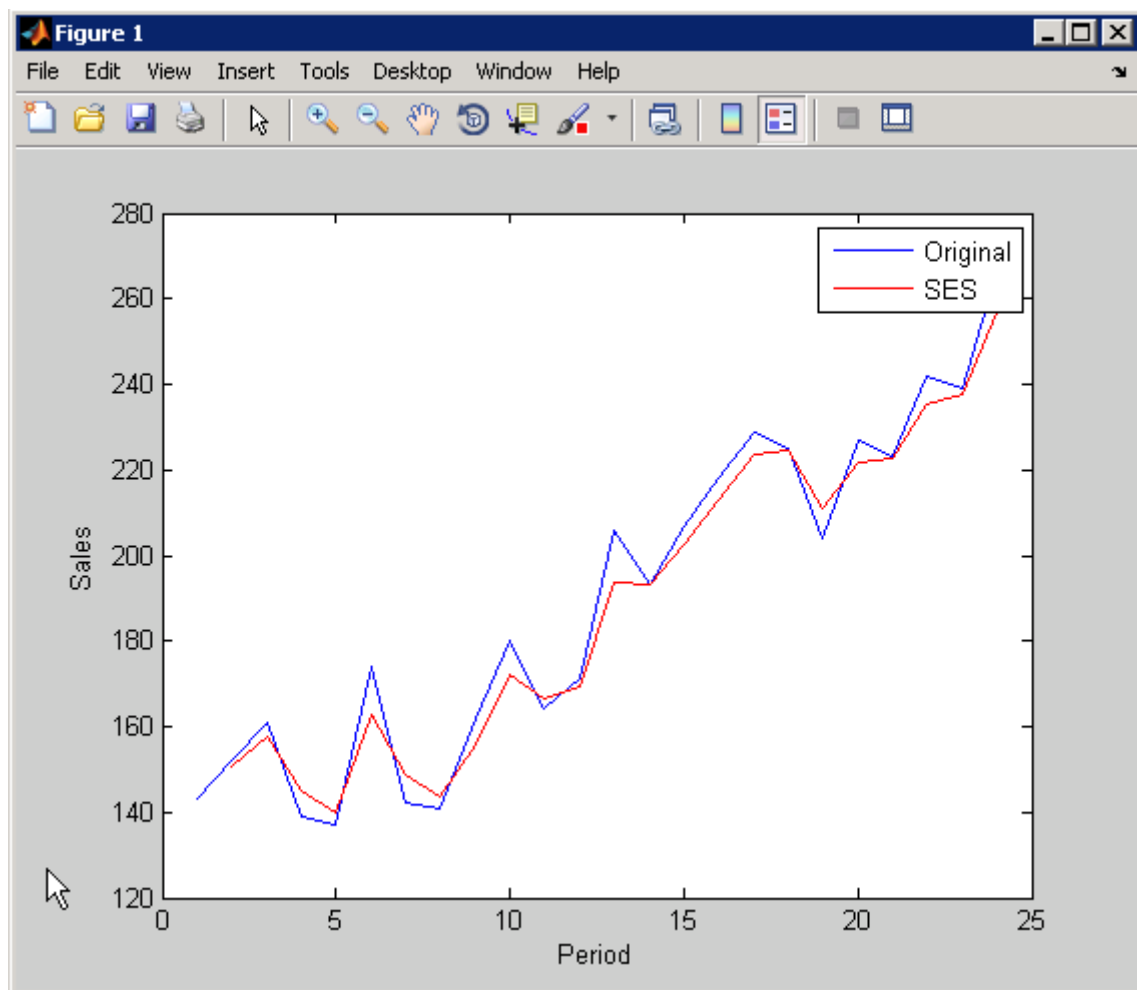
Command Window

```
ans =

      NaN
    150.5000
    157.5000
    145.1667
    139.7222
    162.5741
    148.8580
    143.6193
    155.8731
    171.9577
    166.6526
    169.5509
    193.8503
    193.2834
    202.4278
    212.8093
    223.6031
    224.5344
    210.8448
    221.6149
    222.5383
    235.5128
    237.8376
    256.6125
```

EDITOR PUBLISH VIEW

```
1 clear all;close all;clc
2 Y=[143 152 161 139 137 174 142 141 162 180 164 171 206 193 207 218 229 225 204 227 223 242 239 266];
3 tsmovavg(Y,'e',2)
4 plot(1:length(Y),Y,'b');
5 hold on;
6 plot(1:length(Y),tsmovavg(Y,'e',2),'r');
7 xlabel('Period');ylabel('Sales');
8 legend('Original','SES');
```

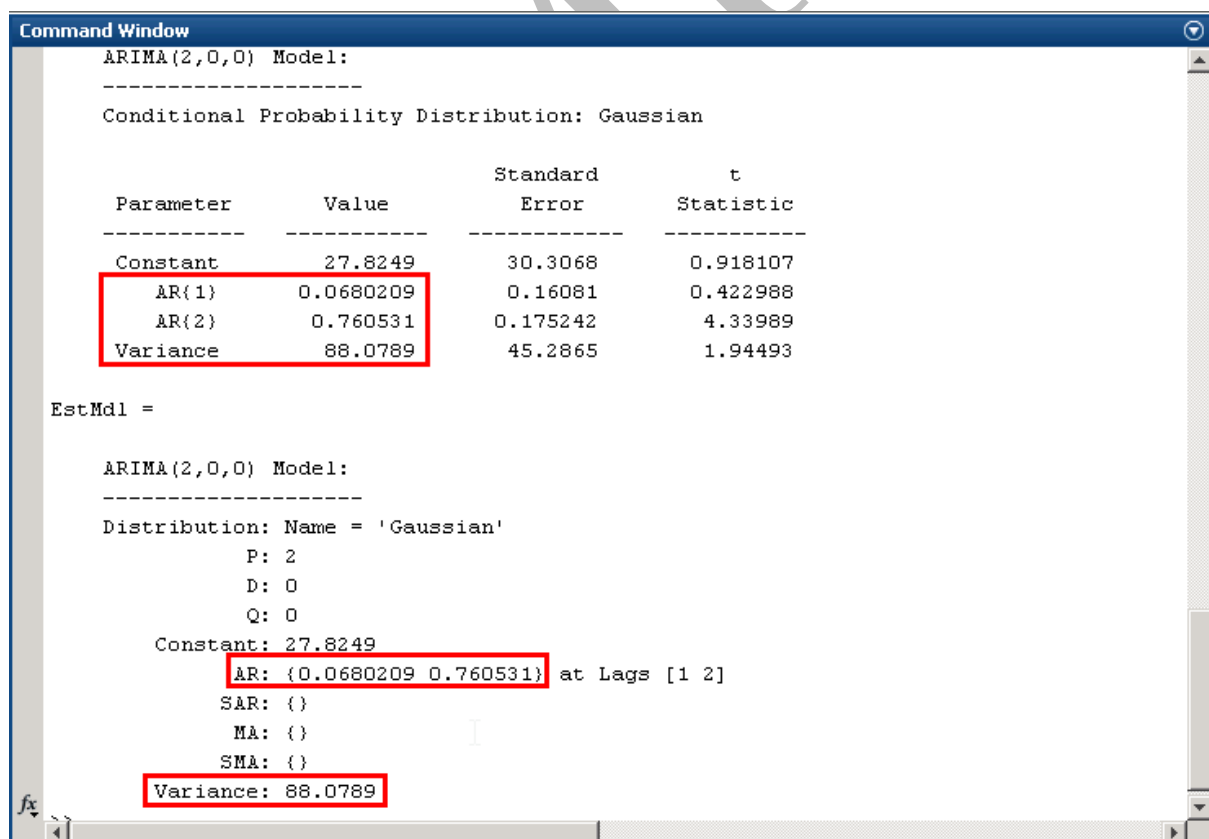


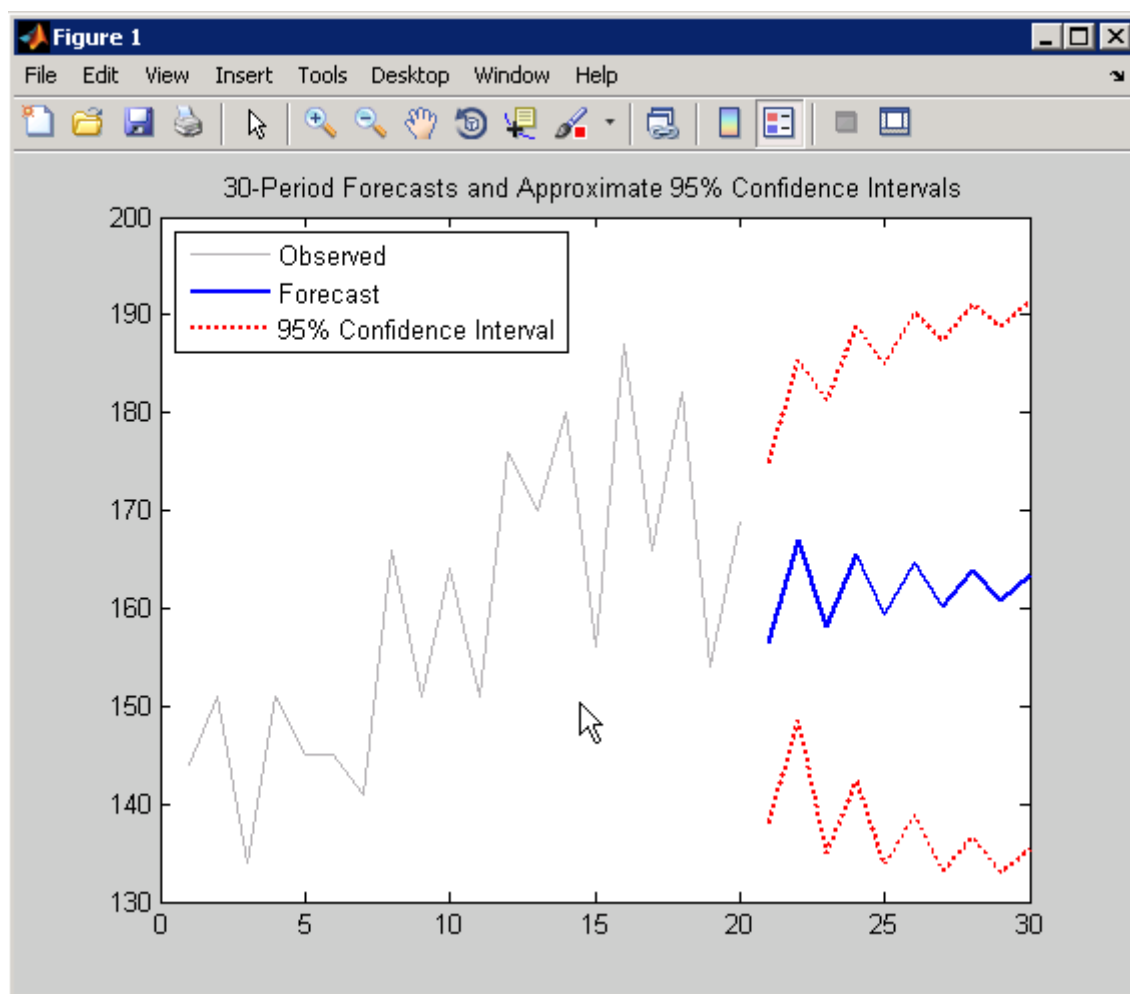
Univariate AR(2)/ARIMA(2,0,0)

```

1  clear all;clc
2  arima(2,0,0)
3  Y=[144 151 134 151 145 145 141 166 151 164 151 176 170 180 156 187 166 182 154 169];
4  Mdl=arima(2,0,0);
5  EstMdl=estimate(Mdl,Y')
6  ForecastPeriods=10;
7  [YF YMSE]=forecast(EstMdl,ForecastPeriods,'YO',Y');
8
9  plot(Y,'Color',[.7,.7,.7]);
10 hold on
11 plot(length(Y)+1:(length(Y)+ForecastPeriods),YF,'b','LineWidth',2);
12 plot(length(Y)+1:(length(Y)+ForecastPeriods),YF + 1.96*sqrt(YMSE),'r','LineWidth',2);
13 plot(length(Y)+1:(length(Y)+ForecastPeriods),YF - 1.96*sqrt(YMSE),'r','LineWidth',2);
14 legend('Observed','Forecast','95% Confidence Interval','Location','NorthWest');
15 title('30-Period Forecasts and Approximate 95% Confidence Intervals');
16
17 %this part is to obtain values for backtesting
18 residuals=infer(EstMdl,Y');
19 BackTesting=Y'-residuals

```





Command Window

```
BackTesting=Y'-residuals
```

```
BackTesting =
```

```
150.5348  
153.5706  
147.6125  
151.7799  
140.0072  
152.5281  
147.9649  
147.6929  
146.3513  
164.3442  
153.8205  
162.8232  
154.6368  
173.2419  
169.3589  
175.3318  
159.1877  
181.3357  
166.4529  
176.7168
```

Univariate ARIMA(0,0,0)/Moving Average

```

EDITOR PUBLISH VIEW
1 clear all;clc
2 arima(2,0,0)
3 Y=[144 151 134 151 145 145 141 166 151 164 151 176 170 180 156 187 166 182 154 169];
4 Mdl=arima(0,0,0);
5 EstMdl=estimate(Mdl,Y')
6 ForecastPeriods=10;
7 [YF YMSE]=forecast(EstMdl,ForecastPeriods,'YO',Y');
8
9 plot(Y,'Color',[.7,.7,.7]);
10 hold on
11 plot(length(Y)+1:(length(Y)+ForecastPeriods),YF,'b','LineWidth',2);
12 plot(length(Y)+1:(length(Y)+ForecastPeriods),YF + 1.96*sqrt(YMSE),'r:','LineWidth',2);
13 plot(length(Y)+1:(length(Y)+ForecastPeriods),YF - 1.96*sqrt(YMSE),'r:','LineWidth',2);
14 legend('Observed','Forecast','95% Confidence Interval','Location','NorthWest');
15 title('30-Period Forecasts and Approximate 95% Confidence Intervals');
16
17 %this part is to obtain values for backtesting
18 residuals=infer(EstMdl,Y');
19 BackTesting=Y'-residuals

```



```
Command Window

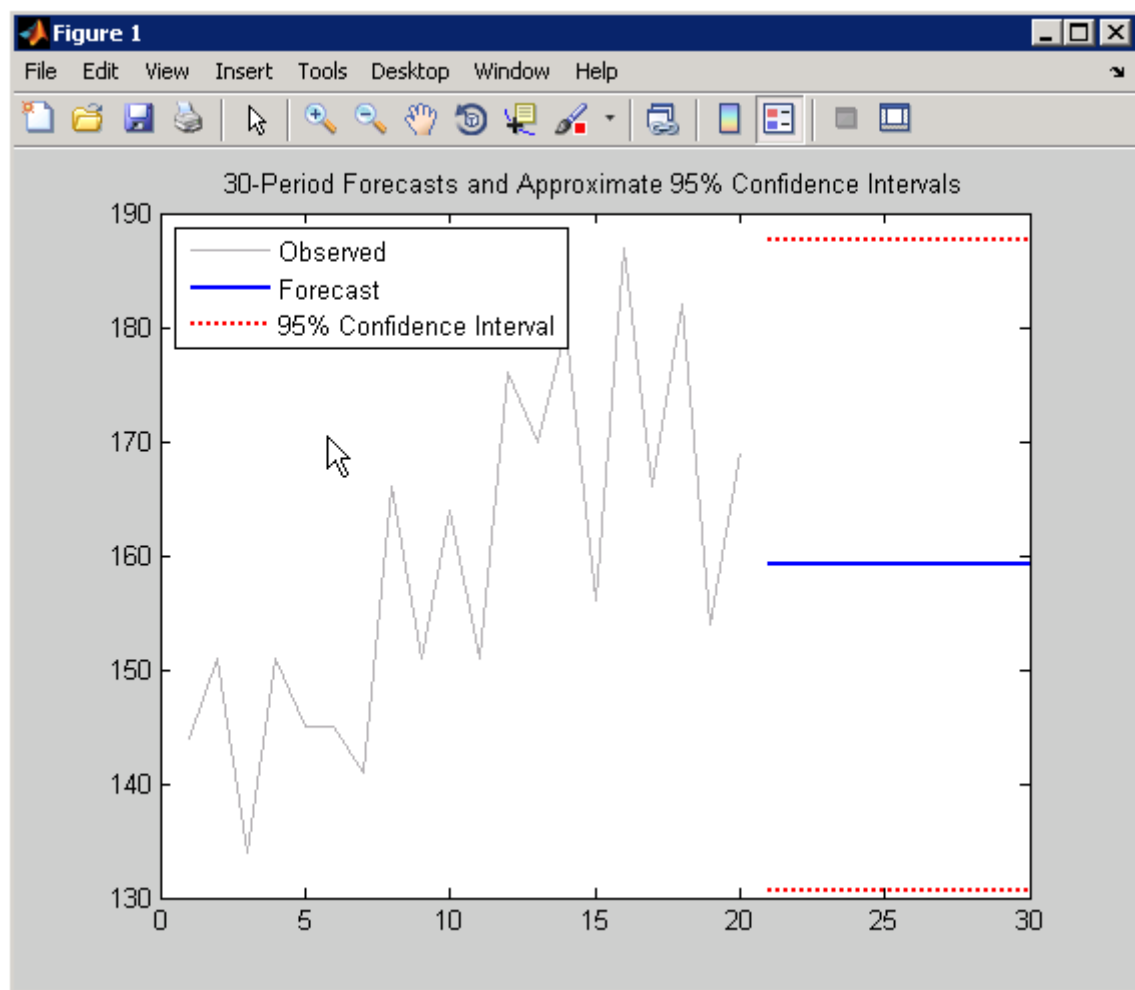
ARIMA(0,0,0) Model:
-----
Conditional Probability Distribution: Gaussian

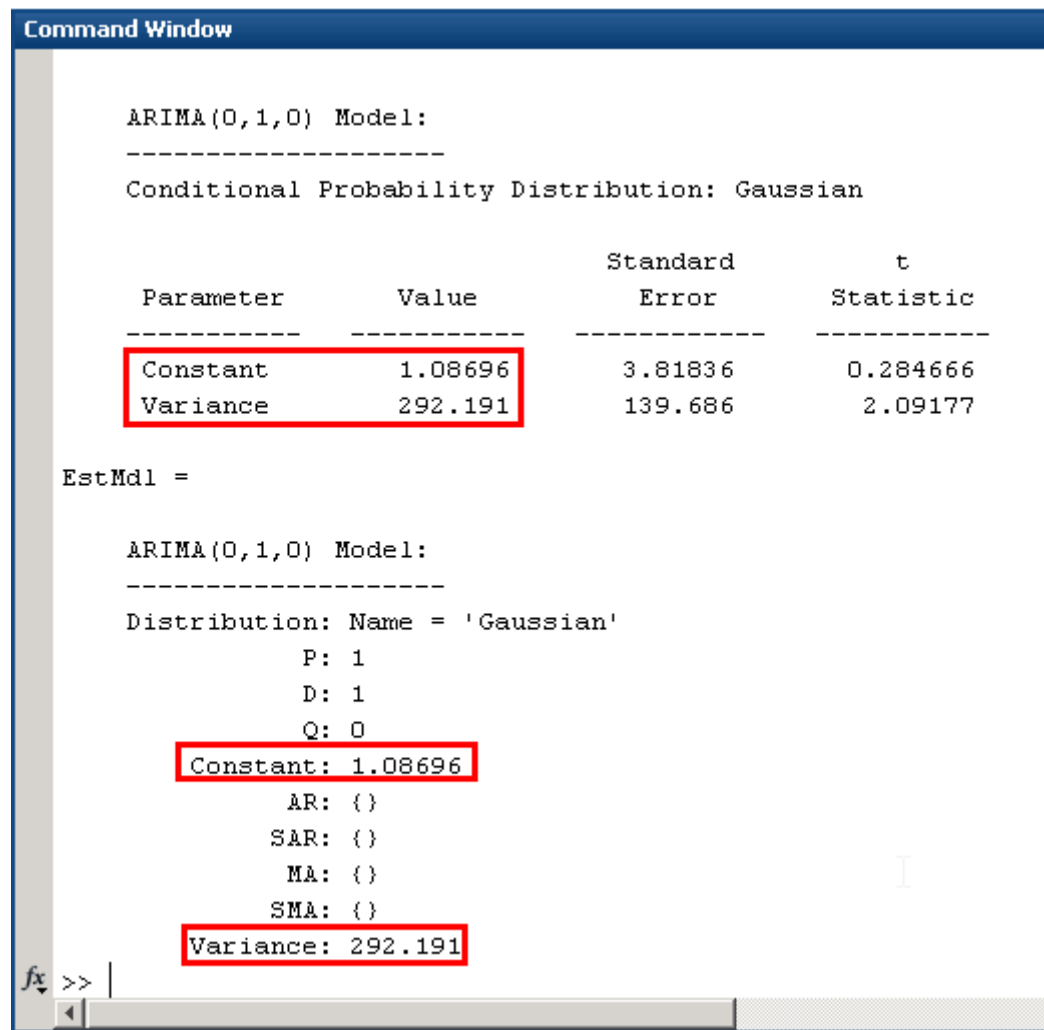
Parameter      Value      Standard      t
              Value      Error      Statistic
-----
Constant       159.15      3.38281      47.0467
Variance       212.127     96.8307      2.1907

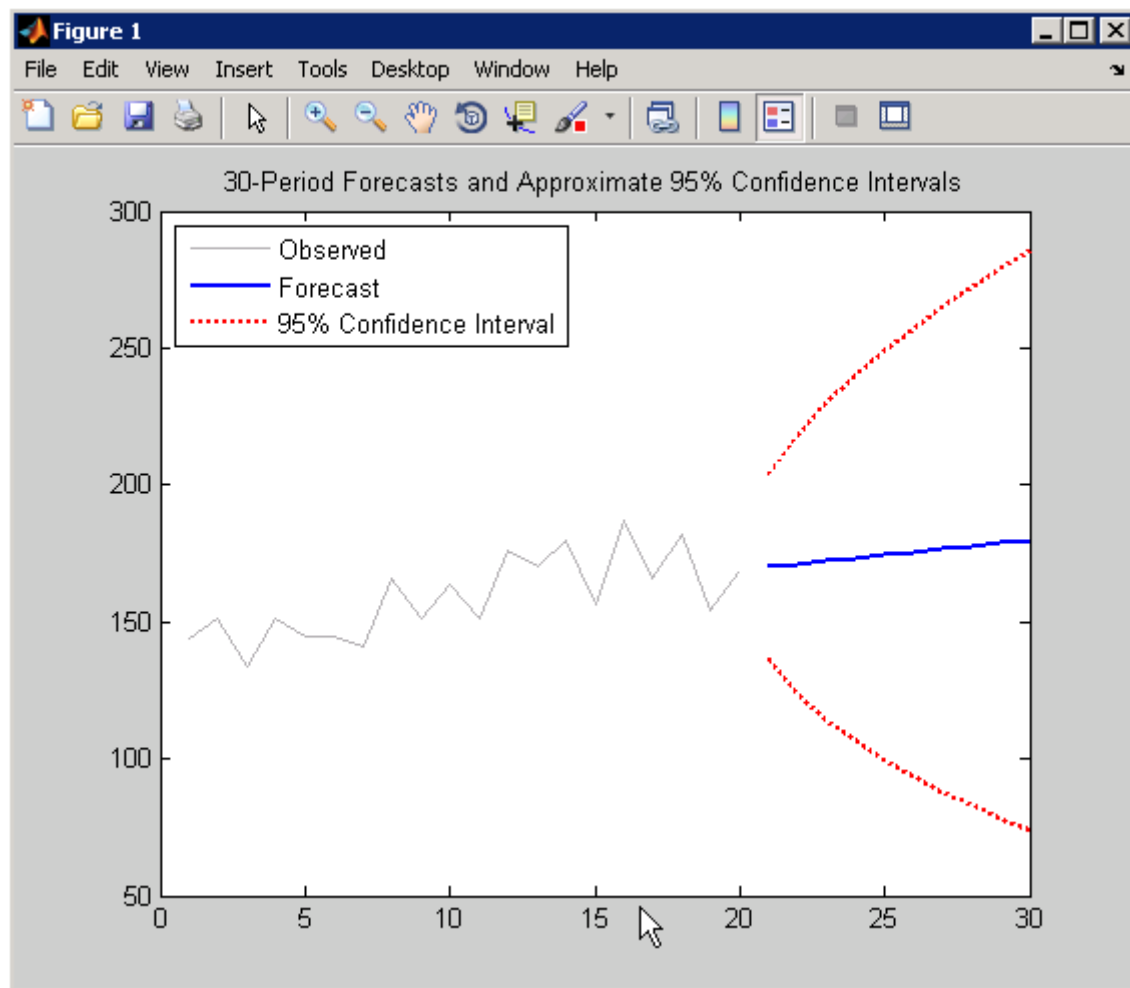
EstMdl =

ARIMA(0,0,0) Model:
-----
Distribution: Name = 'Gaussian'
P: 0
D: 0
Q: 0
Constant: 159.15
AR: {}
SAR: {}
MA: {}
SMA: {}
Variance: 212.127

fx >> |
```







Command Window

BackTesting =

146.1739
145.0870
152.0870
135.0870
152.0870
146.0870
146.0870
142.0870
167.0870
152.0870
165.0870
152.0870
177.0870
171.0870
181.0870
157.0870
188.0870
167.0870
183.0870
155.0870

Univariate ARIMA(0,1,1)/Simple Exponential Smoothing (EWMA)

```

1  clear all;close all;clc
2  arima(2,0,0)
3  Y=[144 151 134 151 145 145 141 166 151 164 151 176 170 180 156 187 166 182 154 169];
4  Mdl=arima(0,1,1);
5  EstMdl=estimate(Mdl,Y')
6  ForecastPeriods=10;
7  [YF YMSE]=forecast(EstMdl,ForecastPeriods,'YO',Y');
8
9  plot(Y,'Color',[.7,.7,.7]);
10 hold on
11 plot(length(Y)+1:(length(Y)+ForecastPeriods),YF,'b','LineWidth',2);
12 plot(length(Y)+1:(length(Y)+ForecastPeriods),YF + 1.96*sqrt(YMSE),'r','LineWidth',2);
13 plot(length(Y)+1:(length(Y)+ForecastPeriods),YF - 1.96*sqrt(YMSE),'r','LineWidth',2);
14 legend('Observed','Forecast','95% Confidence Interval','Location','NorthWest');
15 title('30-Period Forecasts and Approximate 95% Confidence Intervals');
16
17 %this part is to obtain values for backtesting
18 residuals=infer(EstMdl,Y');
19 BackTesting=Y'-residuals

```

Command Window

ARIMA(0,1,1) Model:

Conditional Probability Distribution: Gaussian

Parameter	Value	Standard Error	t Statistic
Constant	1.40876	0.23519	5.98988
MA(1)	-1	0.128317	-7.79322
Variance	113.197	56.493	2.00374

EstMdl =

ARIMA(0,1,1) Model:

Distribution: Name = 'Gaussian'

P: 1

D: 1

Q: 1

Constant: 1.40876

AR: {}

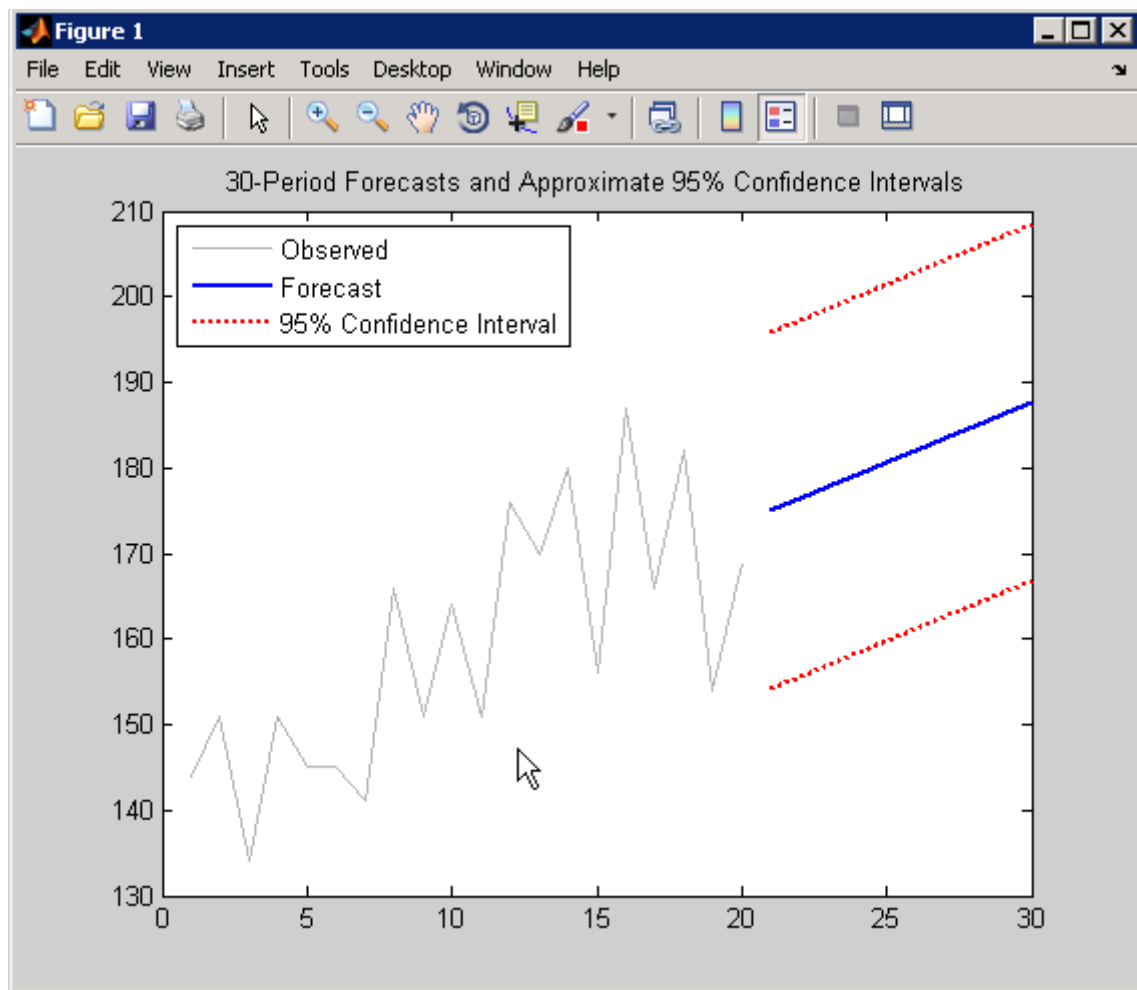
SAR: {}

MA: {-1} at Lags [1]

SMA: {}

Variance: 113.197

f1 >>



Command Window

BackTesting =

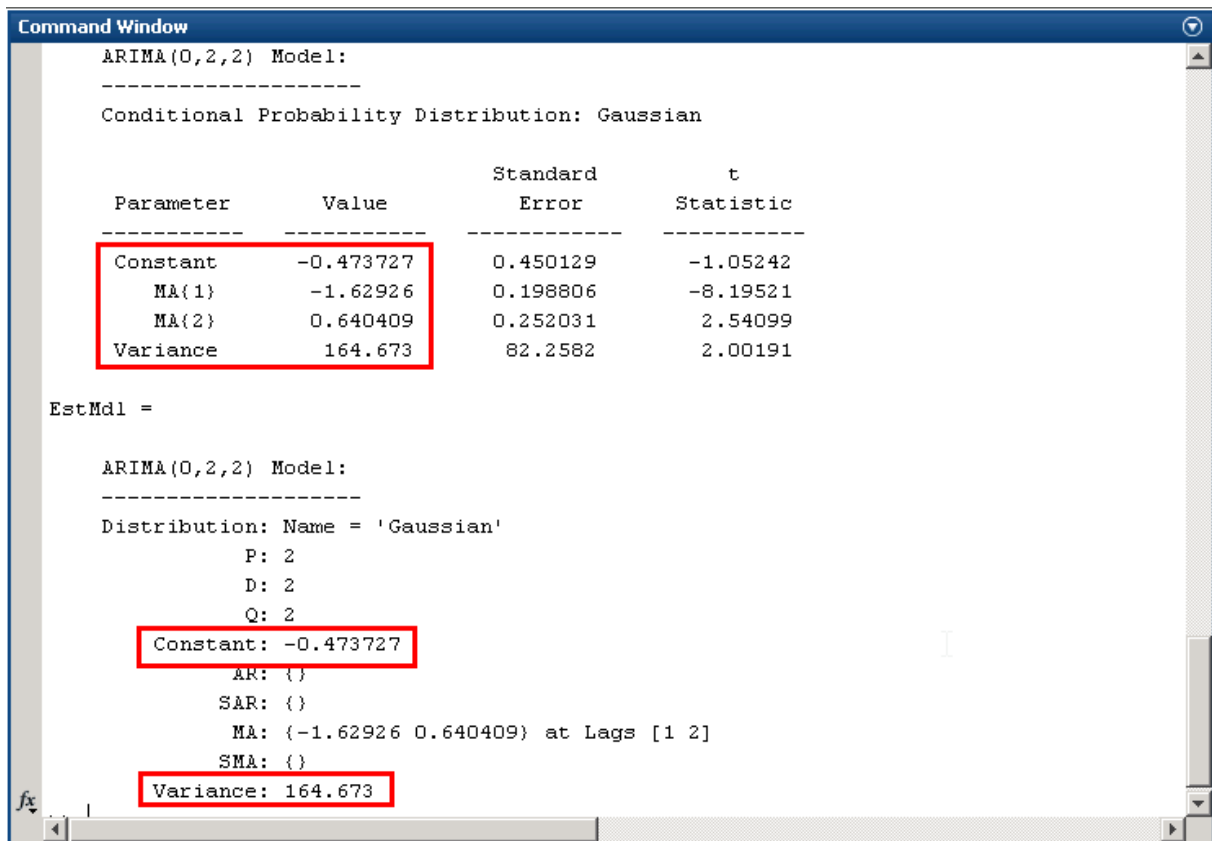
146.8175
 148.2263
 149.6350
 151.0438
 152.4526
 153.8613
 155.2701
 156.6788
 158.0876
 159.4964
 160.9051
 162.3139
 163.7227
 165.1314
 166.5402
 167.9489
 169.3577
 170.7665
 172.1752
 173.5840

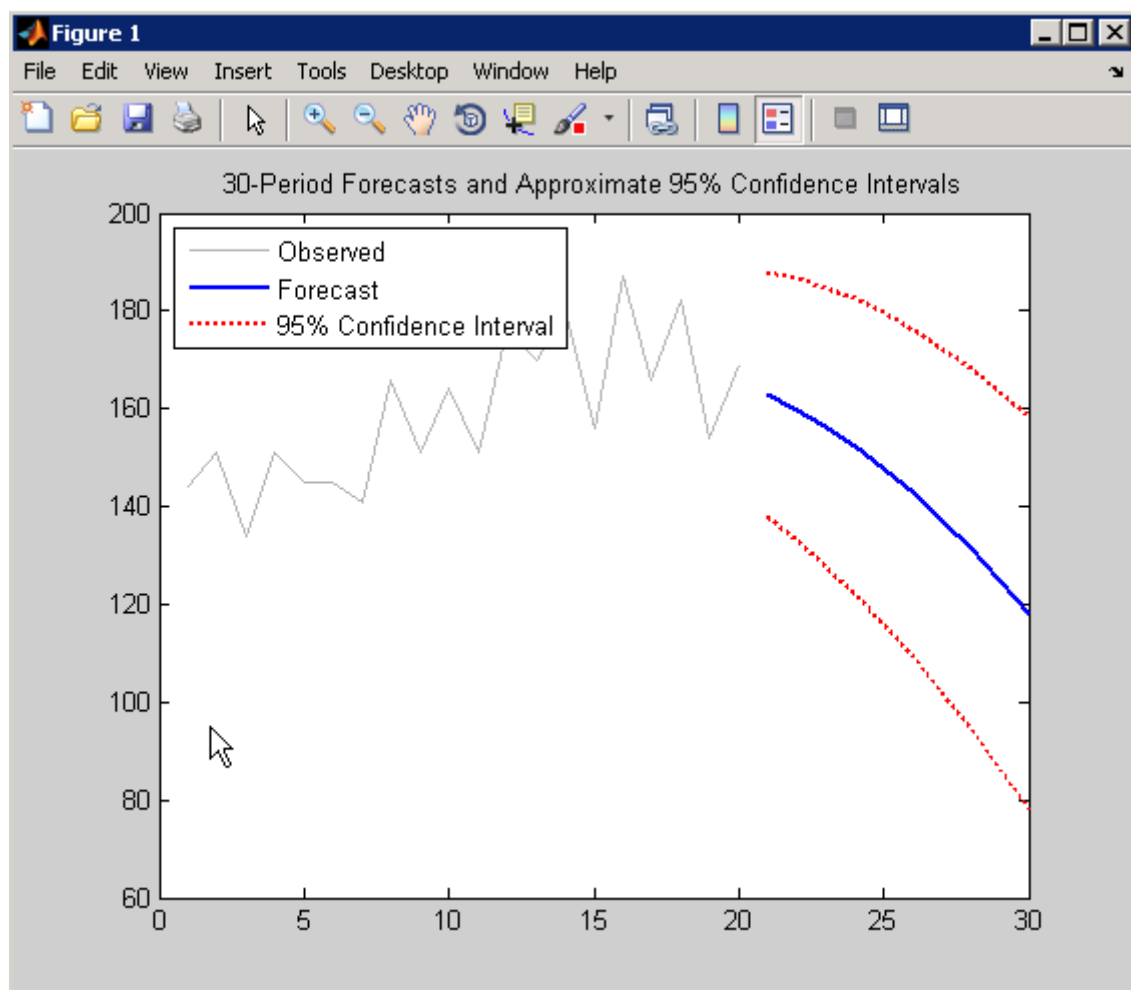
Univariate ARIMA(0,2,2)/Holt's double exponential smoothing

```

EDITOR PUBLISH VIEW
1 clear all;close all;clc
2 arima(2,0,0)
3 Y=[144 151 134 151 145 145 141 166 151 164 151 176 170 180 156 187 166 182 154 169];
4 Mdl=arima(0,2,2);
5 EstMdl=estimate(Mdl,Y')
6 ForecastPeriods=10;
7 [YF YMSE]=forecast(EstMdl,ForecastPeriods,'YO',Y');
8
9 plot(Y,'Color',[.7,.7,.7]);
10 hold on
11 plot(length(Y)+1:(length(Y)+ForecastPeriods),YF,'b','LineWidth',2);
12 plot(length(Y)+1:(length(Y)+ForecastPeriods),YF + 1.96*sqrt(YMSE),'r','LineWidth',2);
13 plot(length(Y)+1:(length(Y)+ForecastPeriods),YF - 1.96*sqrt(YMSE),'r','LineWidth',2);
14 legend('Observed','Forecast','95% Confidence Interval','Location','NorthWest');
15 title('30-Period Forecasts and Approximate 95% Confidence Intervals');
16
17 %this part is to obtain values for backtesting
18 residuals=infer(EstMdl,Y');
19 BackTesting=Y'-residuals

```





Command Window

BackTesting =

144.0000
 151.0000
 157.5263
 154.8566
 158.7433
 158.4478
 157.6348
 155.0166
 161.9784
 160.4468
 163.7065
 160.5040
 167.1419
 168.7935
 173.0983
 166.5606
 173.2753
 169.4692
 172.4512
 163.6129

Univariate ARMA(1,1)/ARIMA(1,0,1)

EDITOR

PUBLISH

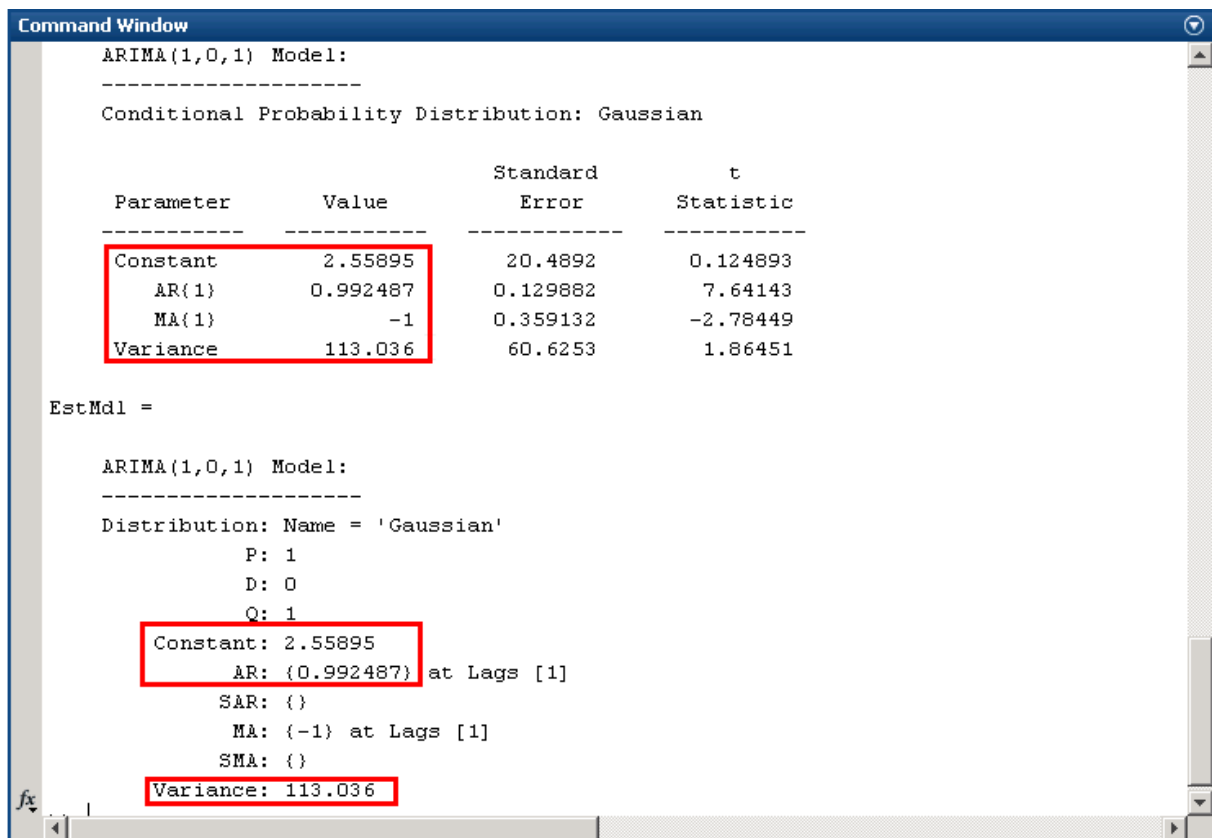
VIEW

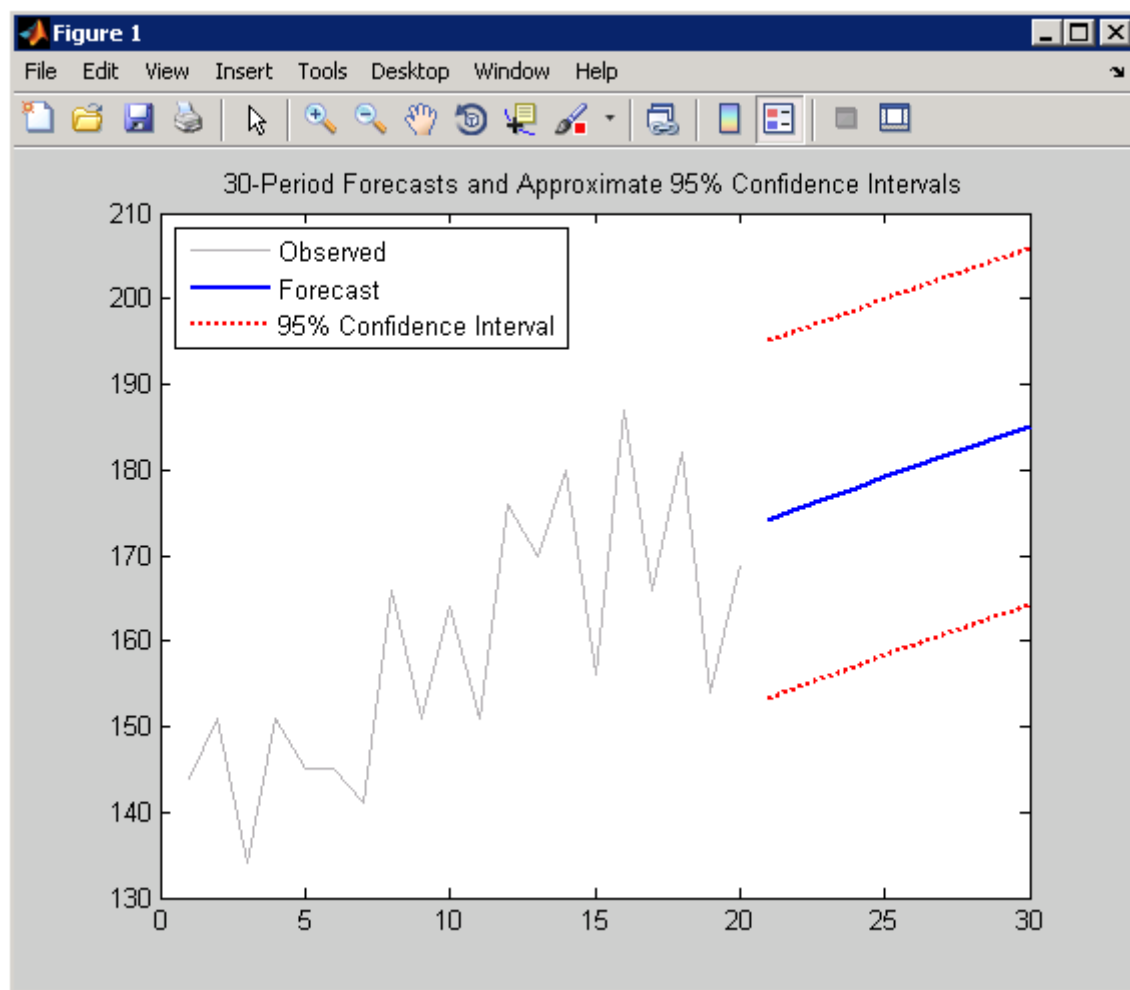


```

1 clear all;close all;clc
2 arima(2,0,0)
3 Y=[144 151 134 151 145 145 141 166 151 164 151 176 170 180 156 187 166 182 154 169];
4 Mdl=arima(1,0,1);
5 EstMdl=estimate(Mdl,Y')
6 ForecastPeriods=10;
7 [YF YMSE]=forecast(EstMdl,ForecastPeriods,'YO',Y');
8
9 plot(Y,'Color',[.7,.7,.7]);
10 hold on
11 plot(length(Y)+1:(length(Y)+ForecastPeriods),YF,'b','LineWidth',2);
12 plot(length(Y)+1:(length(Y)+ForecastPeriods),YF + 1.96*sqrt(YMSE),'r','LineWidth',2);
13 plot(length(Y)+1:(length(Y)+ForecastPeriods),YF - 1.96*sqrt(YMSE),'r','LineWidth',2);
14 legend('Observed','Forecast','95% Confidence Interval','Location','NorthWest');
15 title('30-Period Forecasts and Approximate 95% Confidence Intervals');
16
17 %this part is to obtain values for backtesting
18 residuals=infer(EstMdl,Y');
19 BackTesting=Y'-residuals

```





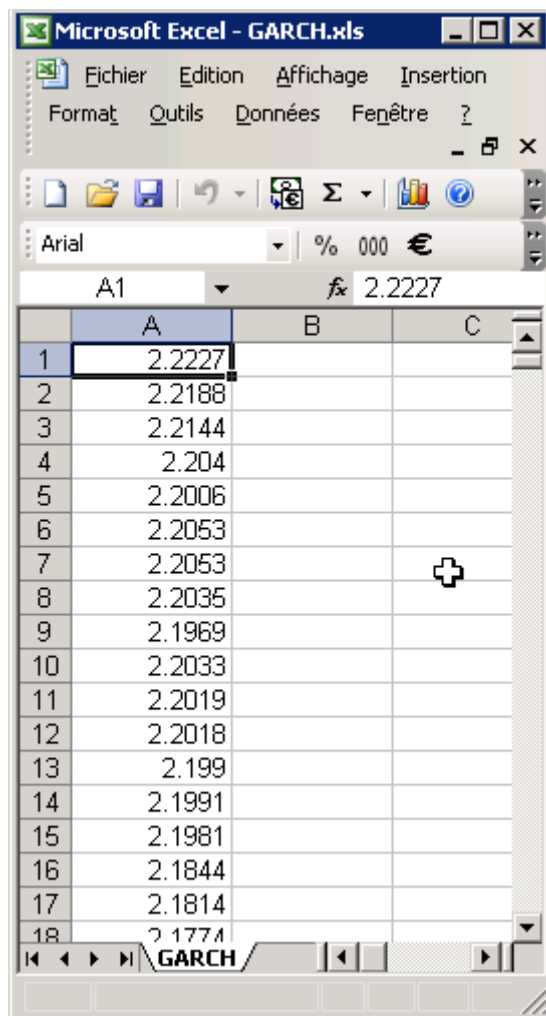
Command Window

BackTesting =

146.9431
148.4201
149.8446
151.3968
152.8213
154.2909
155.7604
157.2600
158.5718
159.9963
161.3231
162.7476
163.9843
165.2660
166.4726
167.8596
169.0136
170.3254
171.5169
172.9189

Univariate GARCH(1,1)

1267 rows



Microsoft Excel - GARCH.xls

Fichier Edition Affichage Insertion
Format Outils Données Fenêtre ?

Arial % 000 €

A1 2.2227

	A	B	C
1	2.2227		
2	2.2188		
3	2.2144		
4	2.204		
5	2.2006		
6	2.2053		
7	2.2053		
8	2.2035		
9	2.1969		
10	2.2033		
11	2.2019		
12	2.2018		
13	2.199		
14	2.1991		
15	2.1981		
16	2.1844		
17	2.1814		
18	2.1774		

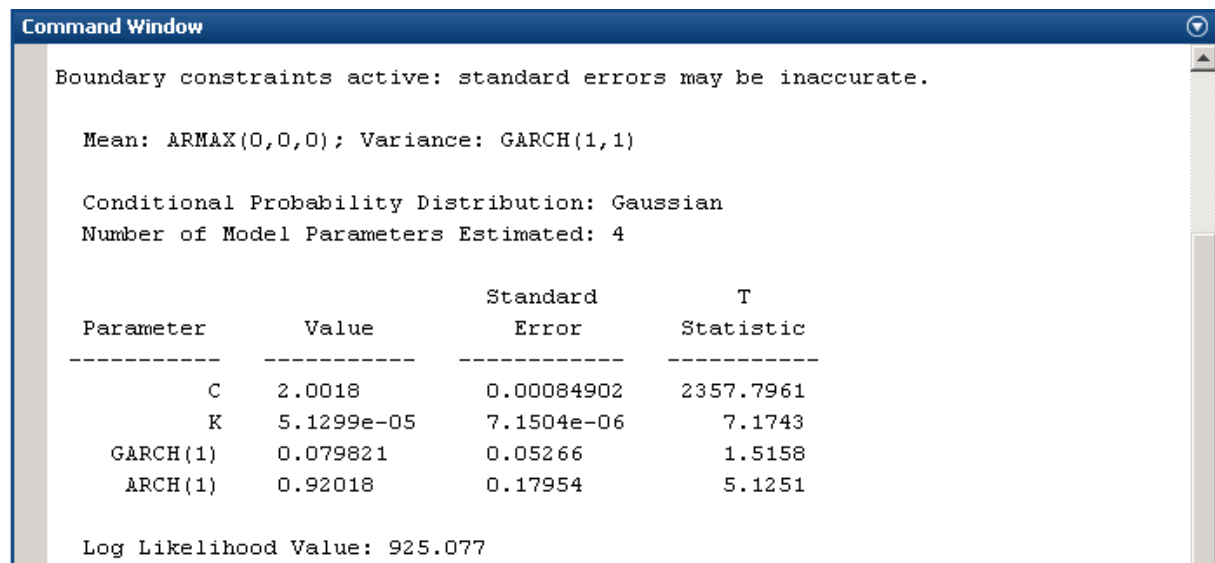
GARCH

EDITOR PUBLISH VIEW

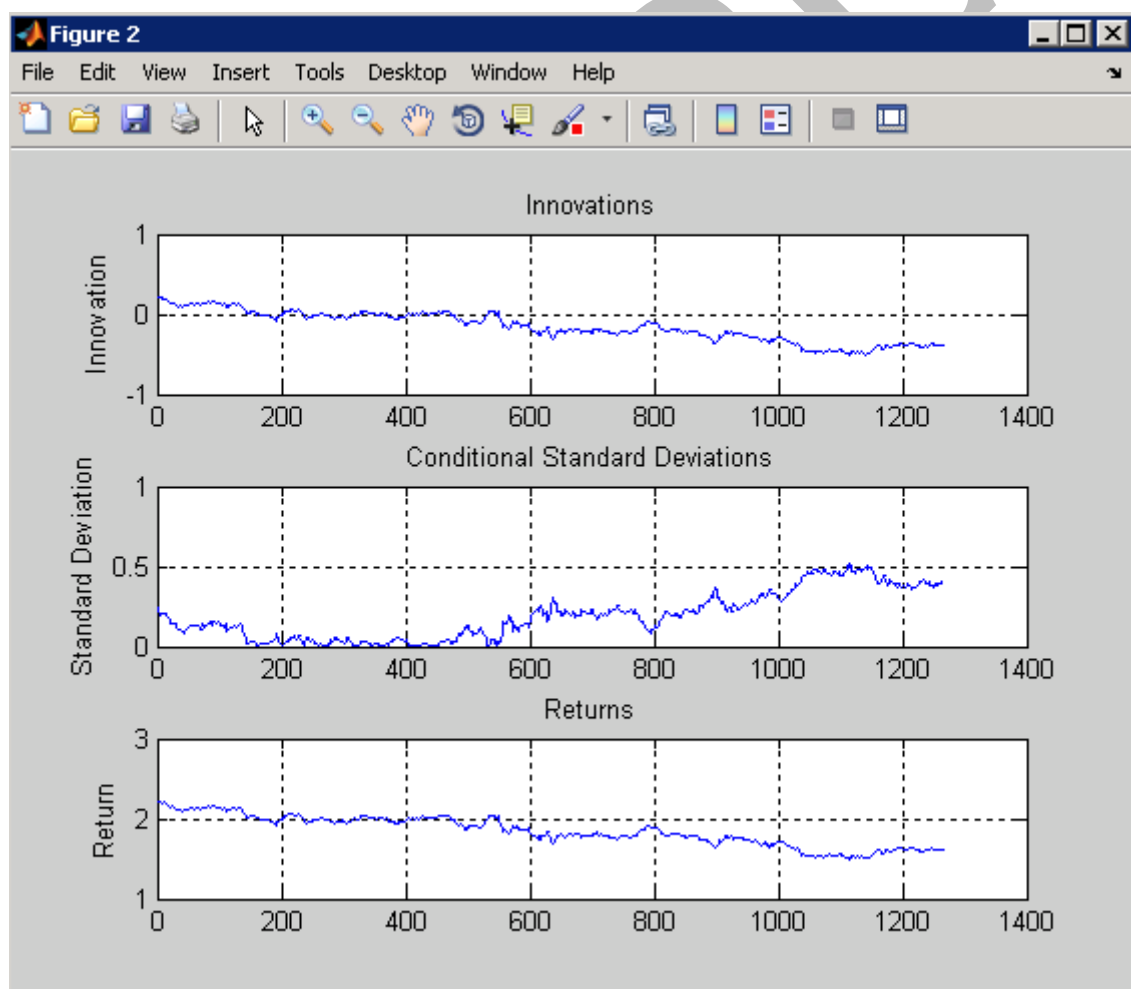
```

1 - Y=xlsread('C:/tmp/GARCH.xls');
2 - spec = garchset('C',0,'K',0.0001,'GARCH',0.9,'ARCH',0.05,'Display','on');
3 - garchfit(spec,Y')

```

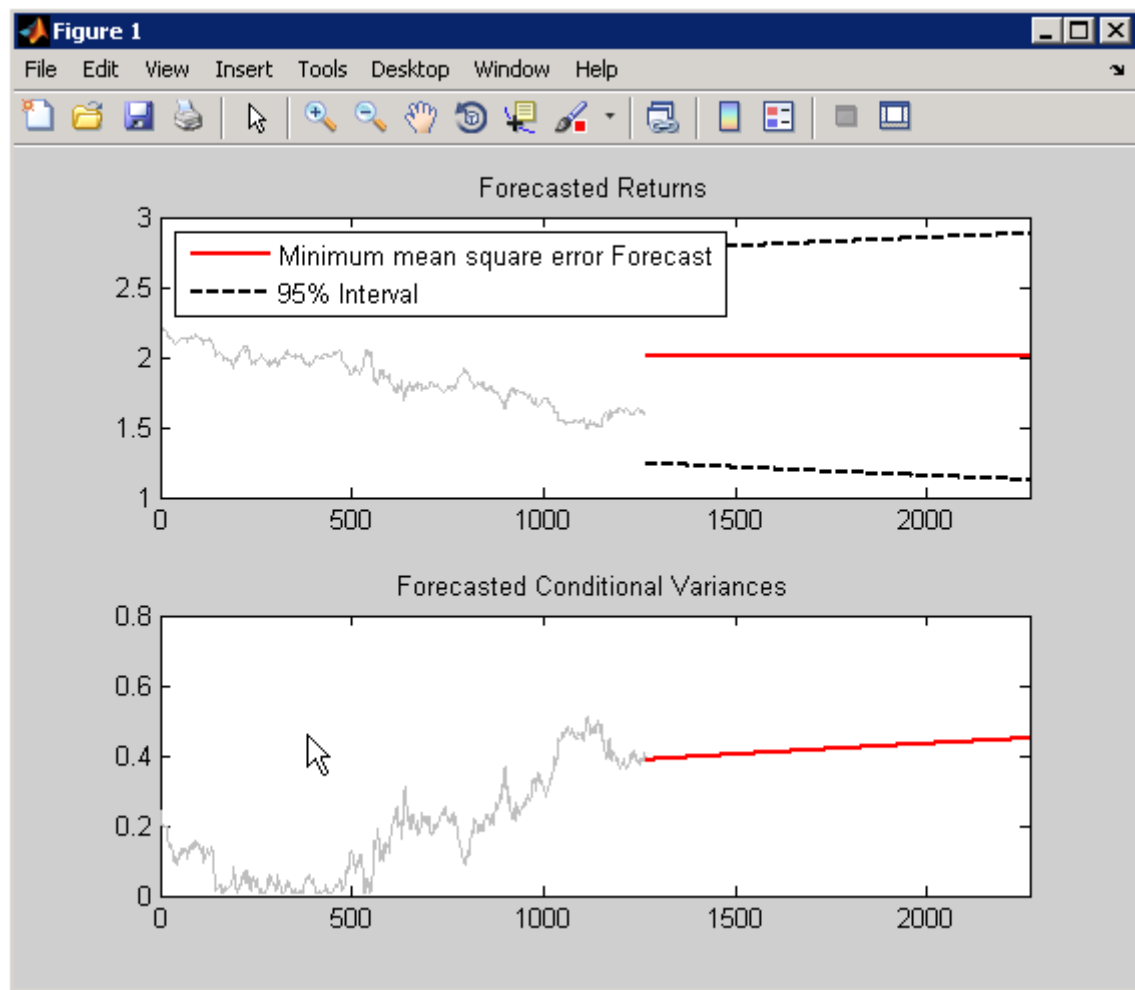
Looking at the results it seems that MATLAB use for ε_t the difference between two consecutive return. The result is thus very different from R.



```

EDITOR PUBLISH VIEW
1 - Y=xlsread('C:/tmp/GARCH.xls');
2 - N=length(Y);
3 - spec = garchset('C',0,'K',0.0001,'GARCH',0.9,'ARCH',0.05,'Display','on');
4 - [coeff,error]=garchfit(spec,Y);
5 - [e0,s0]=garchinfer(coeff,Y);
6 - [sp,yp,~,ymse] = garchpred(coeff,Y',1000);
7
8 - upper = yp + 1.96*ymse;
9 - lower = yp - 1.96*ymse;
10
11 - subplot(2,1,1)
12 - plot(Y,'Color',[.75,.75,.75])
13 - hold on
14 - h1 = plot(N+1:N+1000,yp,'r','LineWidth',2);
15 - h2 = plot(N+1:N+1000,upper,'k--','LineWidth',1.5);
16 - plot(N+1:N+1000,lower,'k--','LineWidth',1.5)
17 - xlim([0,N+1000])
18 - title('Forecasted Returns')
19 - legend([h1,h2],'Minimum mean square error Forecast','95% Interval',...
20 -         'Location','NorthWest')
21 - hold off
22
23 - subplot(2,1,2)
24 - plot(s0,'Color',[.75,.75,.75])
25 - hold on
26 - plot(N+1:N+1000,sp,'r','LineWidth',2)
27 - xlim([0,N+1000])
28 - title('Forecasted Conditional Variances')
29 - hold off

```

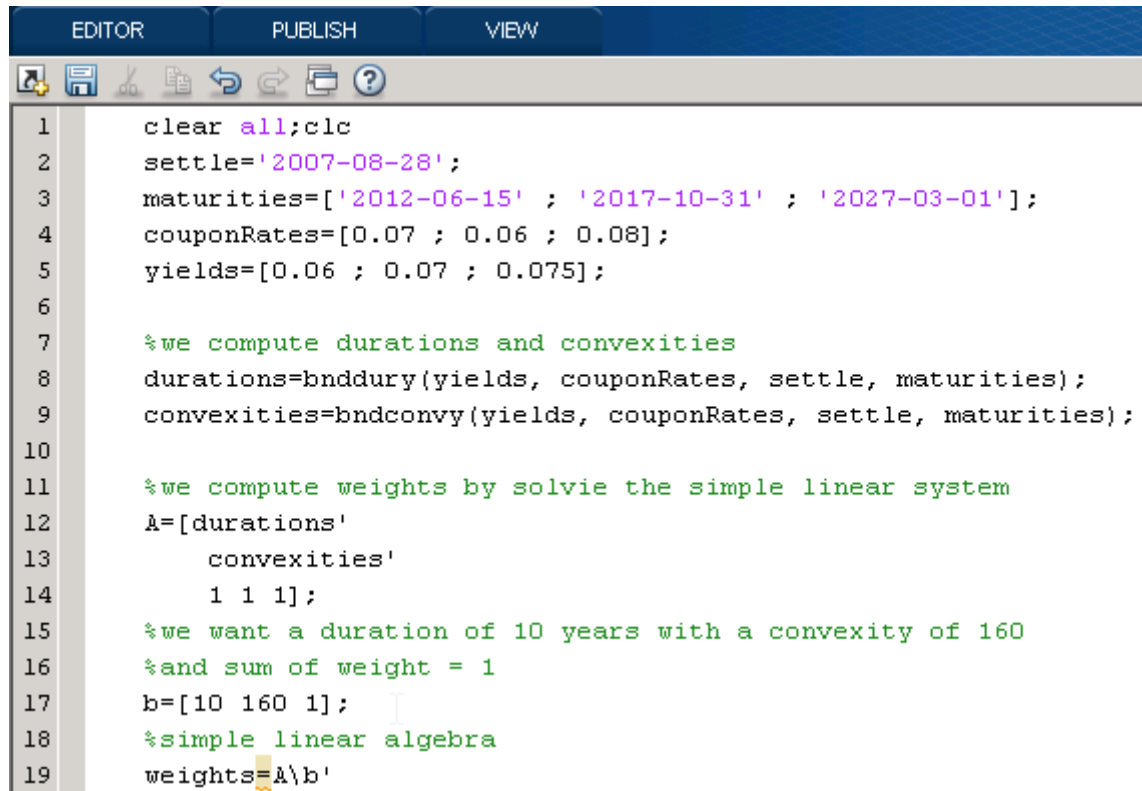


We don't get the same chart as calculated by hand or as obtained with R...

Applications (without Financial Toolbox)

The codes available here do not contain any programmatic essentials such as error checking and they are not optimized for memory efficiency or speed.

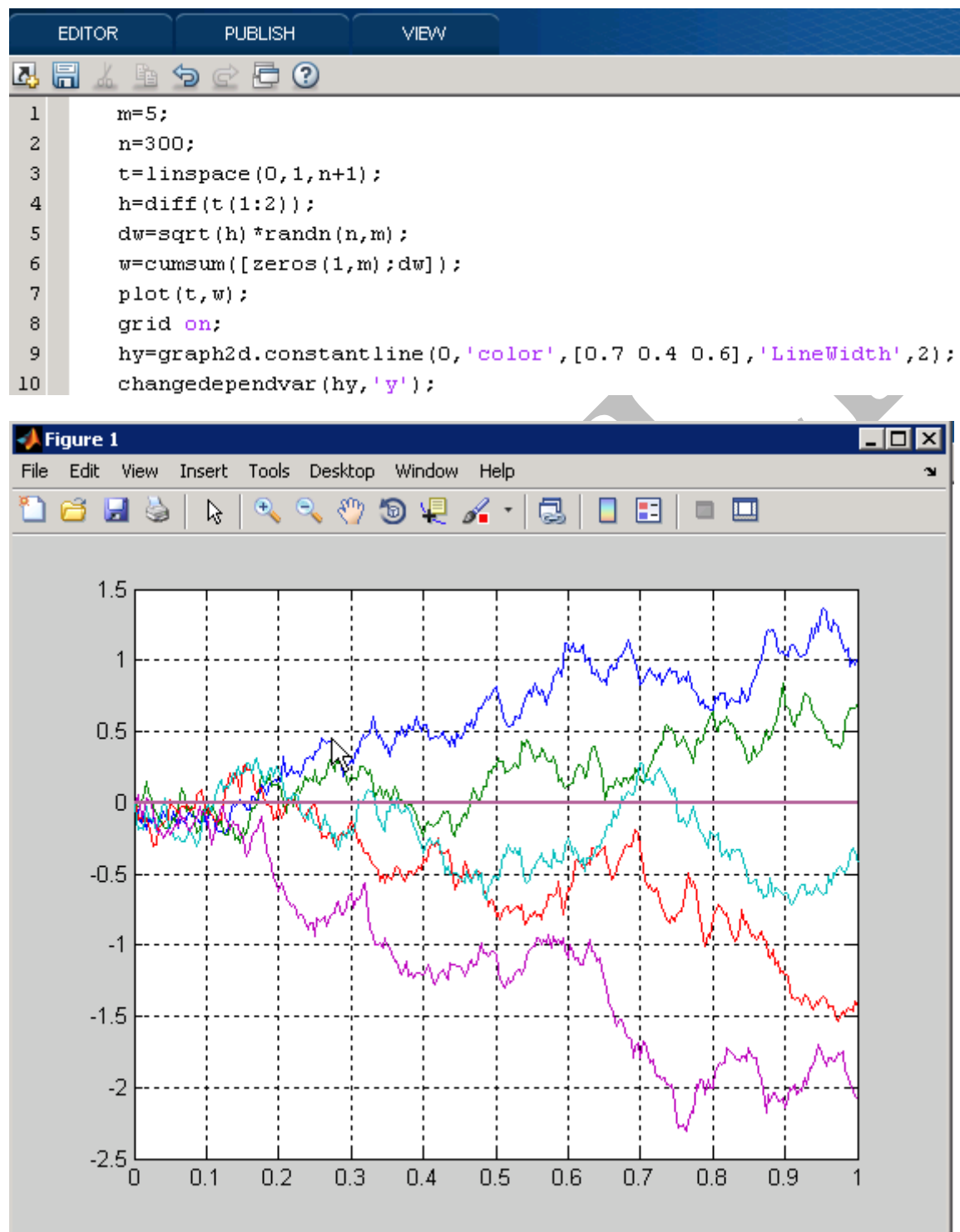
Bond portfolio with duration/convexity targets



```
1 clear all;clc
2 settle='2007-08-28';
3 maturities=['2012-06-15' ; '2017-10-31' ; '2027-03-01'];
4 couponRates=[0.07 ; 0.06 ; 0.08];
5 yields=[0.06 ; 0.07 ; 0.075];
6
7 %we compute durations and convexities
8 durations=bnddury(yields, couponRates, settle, maturities);
9 convexities=bndconvy(yields, couponRates, settle, maturities);
10
11 %we compute weights by solve the simple linear system
12 A=[durations'
13     convexities'
14     1 1 1];
15 %we want a duration of 10 years with a convexity of 160
16 %and sum of weight = 1
17 b=[10 160 1];
18 %simple linear algebra
19 weights=A\b'
```

Brownian motion and Wiener Process

First we plot a few brownian motions:

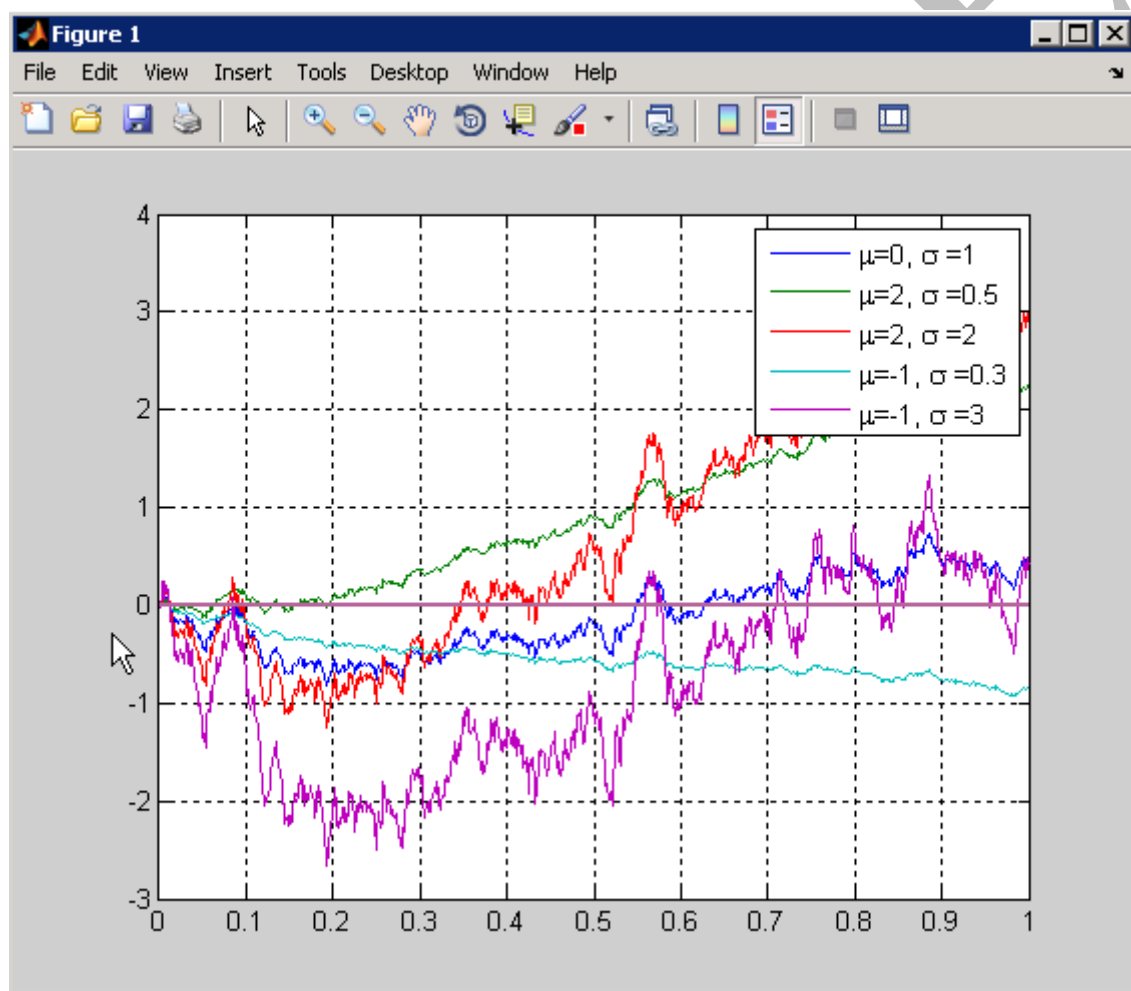


And now a few Wiener processes with different drifts and volatilities:

```

EDITOR PUBLISH VIEW
1 n=1000;
2 t=linspace(0,1,n+1);
3 h=diff(t(1:2));
4 dw=sqrt(h)*randn(n,1);
5 w=cumsum([0;dw]);
6 plot(t,t'*[0 2 2 -1 -1]+w*[1 0.5 2 0.3 3]);
7 legend('\mu=0, \sigma =1', '\mu=2, \sigma =0.5', '\mu=2, \sigma =2',...
8        '\mu=-1, \sigma =0.3', '\mu=-1, \sigma =3');
9 grid on;
10 hy=graph2d.constantline(0,'color',[0.7 0.4 0.6],'LineWidth',2);
11 changedependvar(hy,'y');

```



Risk-return asset pricing models

Capital Asset Pricing Model (CAPM)

```

1 - clear all;close all;clc
2 - ra=[-0.15,0.05,-0.43,0.79,0.32];
3 - rb=[0.29,0.18,0.24,0.25,0.17];
4 - rc=[0.38,0.63,0.46,0.36,-0.57];
5 - PfM=[0.45,0.61,0.54,0.29,0.12];
6 - Rf= repmat(0,1,length(PfM));
7 - Returns=[ra',rb',rc',PfM'];
8 - [m,n]=size(Returns);
9 - %compute the excess return
10 - exreturn = zeros(m,n);
11 - for i = 1:m
12 -     exreturn(i,:)=Returns(i,:)- Rf(i);
13 - end
14 - %regression with each risky asset on the portfolio, note that the
15 - %last columns of the exreturn matrix is market premium data
16 - for i= 1:n
17 -     CovarianceMatrix = cov(exreturn(:,i),exreturn(:,n));% Covariance matrix
18 -     % beta = cov(x,y)/var(x);
19 -     Beta(i)= CovarianceMatrix(1,2)/var(exreturn(:,n));
20 -     % Alpha = return - (rf + beta(E(rm)-rf))
21 -     Alpha(i)= mean(exreturn(:,i))-(mean(Rf)+ Beta(i)*mean(exreturn(:,n)));
22 -     % rt = rf + beta(E(rm)-rf)
23 -     Rt(i) = mean(Rf) + Beta(i)*mean(exreturn(:,n));
24 - end
25 - capm_Alpha = Alpha
26 - capm_Beta = Beta
27 - capm_return = Rt

```

```

Command Window

capm_Alpha =

    0.7186    0.2041   -0.6029         0

capm_Beta =

   -1.4990    0.0545    2.1265    1.0000

capm_return =

   -0.6026    0.0219    0.8549    0.4020

fx >> |

```

Three-factor Fama-French model (FFM)

```

EDITOR PUBLISH VIEW

4 %ER: Excess Return (Assets Return - Risk Free Return)
5 %also known as "Risk Adjusted Return".
6 ER=[4.7 -10.26 26.09 0.06 13.68 16.74 14.65 20.63 15 -15.81 -23.88];
7 %SMB: "Small [market capitalization] Minus Big"
8 %1/3 (Small Value+Small Neutral+Small Growth)-1/3 (Big Value+Big Neutral+Big Growth)
9 SMB=[-9.61 -12.66 4.58 7.81 -9.67 -2.14 -13.36 -15.78 14.86 -6.24 0.53];
10 %HML: "High [book-to-market ratio] Minus Low"
11 %1/2 (Small Value+Big Value)-1/2 (Small Growth+Big Growth)
12 HML=[-6.4 -2.51 23.22 5.4 -5.82 1.96 13.1 -0.52 -22.2 28.17 25.27];
13 %Risk free return
14 Rf=[5.6 3.5 2.9 3.91 5.6 5.2 5.25 4.85 4.69 5.88 3.85];
15
16 format bank
17 FF3=[year', ER', SMB', HML', Rf'];
18 printmat(FF3, 'Raw Data:', num2str(1:size(FF3)), 'Date (Rmt-rf) SMB HML rf')
19
20 AssetReturn1=[0.010 0.022 0.074 0.02 0.009 0.042 0.045 0.069 0.079 0.077 0.087];
21 AssetReturn2=[0.057 0.026 0.040 0.021 0.017 0.058 0.063 0.053 0.084 0.053 0.06];
22 Portfolio=[AssetReturn1' AssetReturn2'];
23 printmat(Portfolio, 'Portfolio', num2str(1:size(Portfolio)), 'Asset(1) Asset(2)')

```

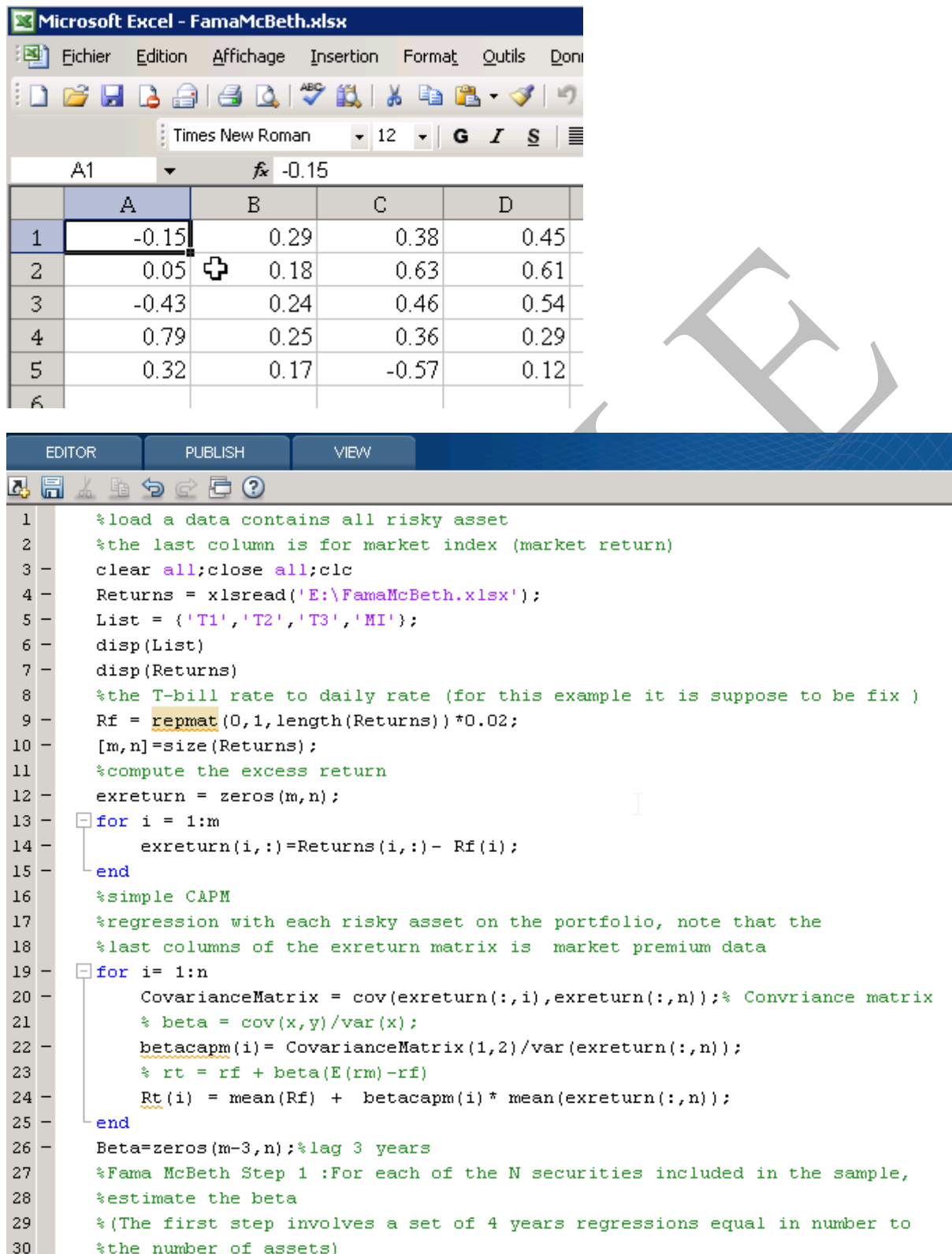

Command Window						
Raw Data: =						
	Date	(Rmt-rf)	SMB	SMH	rf	
1	1991.00000	4.70000	-9.61000	-6.40000	5.60000	
2	1992.00000	-10.26000	-12.66000	-2.51000	3.50000	
3	1993.00000	26.09000	4.58000	23.22000	2.90000	
4	1994.00000	0.06000	7.81000	5.40000	3.91000	
5	1995.00000	13.68000	-9.67000	-5.82000	5.60000	
6	1996.00000	16.74000	-2.14000	1.96000	5.20000	
7	1997.00000	14.65000	-13.36000	13.10000	5.25000	
8	1998.00000	20.63000	-15.78000	-0.52000	4.85000	
9	1999.00000	15.00000	14.86000	-22.20000	4.69000	
10	2000.00000	-15.81000	-6.24000	28.17000	5.88000	
11	2001.00000	-23.88000	0.53000	25.27000	3.85000	

Command Window			
Portfolio =			
	Asset (1)	Asset (2)	
1	0.01000	0.05700	
2	0.02200	0.02600	
3	0.07400	0.04000	
4	0.02000	0.02100	
5	0.00900	0.01700	
6	0.04200	0.05800	
7	0.04500	0.06300	
8	0.06900	0.05300	
9	0.07900	0.08400	
10	0.07700	0.05300	
11	0.08700	0.06000	

```

24
25 - FF3(:,1)=[]; %remove date data
26 - FF3=FF3/100; %format data in %
27 - [~,last_col]=size(FF3);
28 - FF3(:,last_col)=[]; %remove rf
29 - [N,M] = size(Portfolio);
30 - for i = 1:M
31     %compute excess return for all asset in portfolio
32     exreturn(:,i) = Portfolio(:,i) - Rf';
33     %get beta coefficient using multivariate regression function
34     %such as: (rt-rf) = b(Rm-rf) + bs(SMB) + bv(HML) + alpha
35     coef(:,i) = regress(exreturn(:,i),FF3);
36     expReturn(:,i) = mean(Rf') + coef(1,i)*mean(FF3(:,1))...
37         +coef(2,i)*mean(FF3(:,2))+coef(2,i)*mean(FF3(:,3));
38 - end
39
40 - for i = 1:M
41     fprintf('Asset(%d)>> beta(i): %0.2f b(SMB): %0.2f b(HML): %0.2f 3ffExp_Return: %0.2f Alpha: %0.2f\n',...
42         i,coef(1,i),coef(2,i),coef(3,i),expReturn(1,i),mean(Portfolio(:,i))-expReturn(1,i))
43 - end

```

Fama-McBeth


The image shows two windows. The top window is Microsoft Excel, titled 'FamaMcBeth.xlsx', displaying a table of data. The bottom window is MATLAB, showing a script that processes the data from the Excel file.

Excel Data:

	A	B	C	D
1	-0.15	0.29	0.38	0.45
2	0.05	0.18	0.63	0.61
3	-0.43	0.24	0.46	0.54
4	0.79	0.25	0.36	0.29
5	0.32	0.17	-0.57	0.12
6				

MATLAB Script:

```

1 %load a data contains all risky asset
2 %the last column is for market index (market return)
3 clear all;close all;clc
4 Returns = xlsread('E:\FamaMcBeth.xlsx');
5 List = {'T1','T2','T3','MI'};
6 disp(List)
7 disp>Returns)
8 %the T-bill rate to daily rate (for this example it is suppose to be fix )
9 Rf = repmat(0,1,length>Returns))*0.02;
10 [m,n]=size>Returns);
11 %compute the excess return
12 exreturn = zeros(m,n);
13 for i = 1:m
14     exreturn(i,:)=Returns(i,:)- Rf(i);
15 end
16 %simple CAPM
17 %regression with each risky asset on the portfolio, note that the
18 %last columns of the exreturn matrix is market premium data
19 for i= 1:n
20     CovarianceMatrix = cov(exreturn(:,i),exreturn(:,n));% Convriance matrix
21     % beta = cov(x,y)/var(x);
22     betacapm(i)= CovarianceMatrix(1,2)/var(exreturn(:,n));
23     % rt = rf + beta(E(Rm)-rf)
24     Rt(i) = mean(Rf) + betacapm(i) * mean(exreturn(:,n));
25 end
26 Beta=zeros(m-3,n);%lag 3 years
27 %Fama McBeth Step 1 :For each of the N securities included in the sample,
28 %estimate the beta
29 %(The first step involves a set of 4 years regressions equal in number to
30 %the number of assets)
  
```

```

31 - for j = 1 : n
32 -     for i = 1 : (m-3)
33 -         % beta = cov(x,y)/var(x);
34 -         Beta(i,j) = regress(exreturn(i:(i+3),j), exreturn(i:(i+3),n));
35 -     end
36 - end
37 - Beta
38 - %Fama McBeth Step 2 : run a cross sectional regression for every t. That is,
39 - %for period t, estimate risk premium
40 - %(The second stage is a set of regressions equal in number to the number of time periods)
41 - %since Fama-McBeth empirical CAPM is : exreturn = premium*beta + intercept
42 - premium = zeros(m-3,n);
43 - intercept = zeros(m-3,n);
44 - for j = 1:n
45 -     for i = 1:(m-3)
46 -         premium(i,j) = regress(exreturn(i:(i+3),j), (ones(1,4)*Beta(i,j))');
47 -         intercept(i,j) = exreturn(i+3,j) - premium(i,j)*Beta(i,j);
48 -     end
49 - end
50 - premium
51 - %empirical estimated premium
52 - empirical_premium = mean(premium)
53 - %theoretical premium
54 - theoretical_premium = mean(exreturn)
55 - %empirical average CAPM
56 - rt = mean(Rf) + mean(premium) * mean(Beta)' + mean(intercept);
57 - empirical_capm = rt
58 - %theoretical CAPM
59 - theoretical_capm = Rt'

```

Command Window

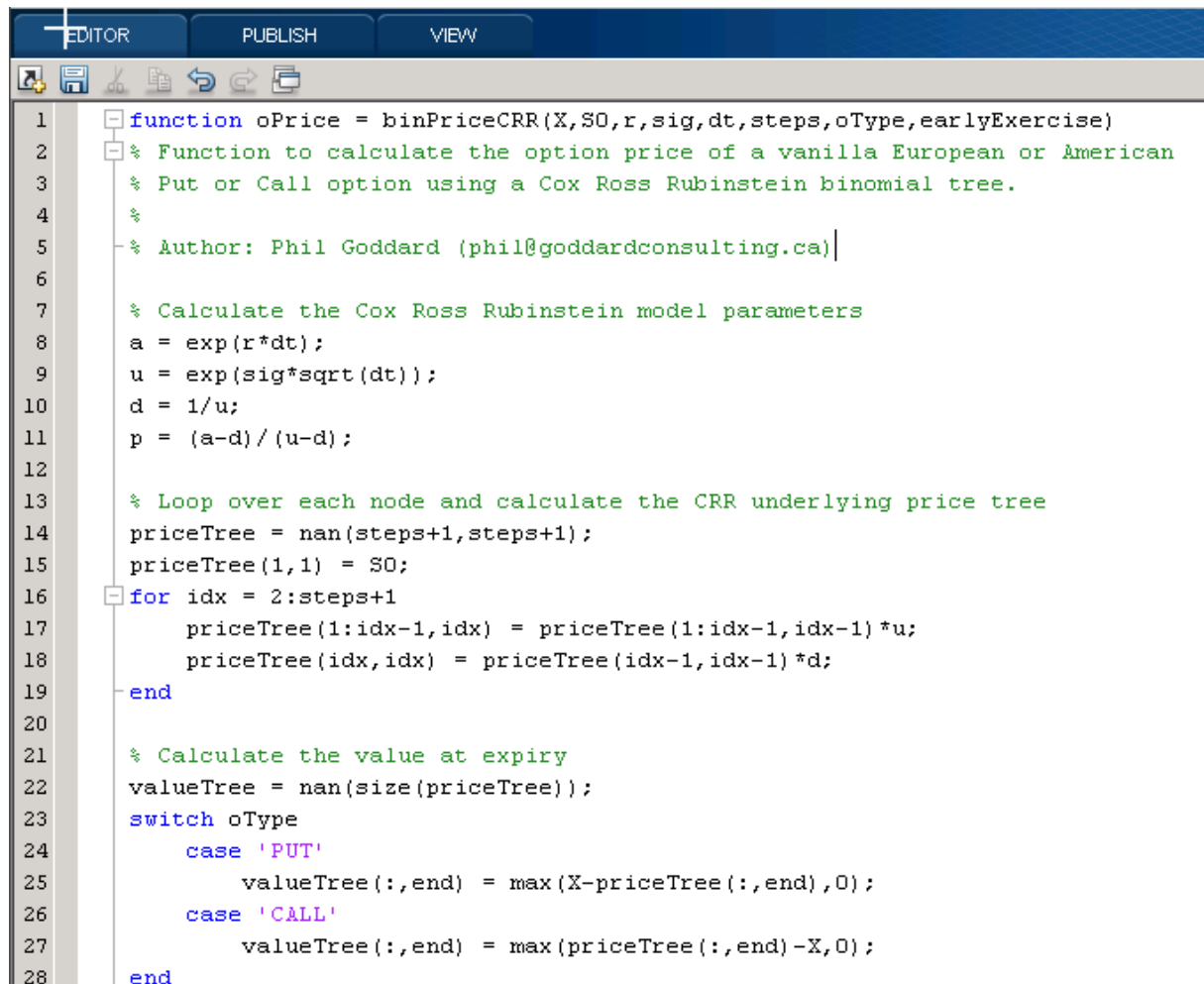
	'T1'	'T2'	'T3'	'MI'
	-0.1500	0.2900	0.3800	0.4500
	0.0500	0.1800	0.6300	0.6100
	-0.4300	0.2400	0.4600	0.5400
	0.7900	0.2500	0.3600	0.2900
	0.3200	0.1700	-0.5700	0.1200
Beta =				
	-0.0422	0.4655	0.9556	1.0000
	0.0863	0.4360	0.8773	1.0000
premium =				
	-1.5404	0.5155	0.4788	0.4725
	2.1140	0.4817	0.2508	0.3900
empirical_premium =				
	0.2868	0.4986	0.3648	0.4312

```
theoretical_premium =  
  
    0.1160    0.2260    0.2520    0.4020  
  
emprirical_capm =  
  
    1.4279    0.9816    0.5529    0.7704  
  
theoretical_capm =  
  
   -0.6026  
    0.0219  
    0.8549  
    0.4020
```

Options Pricing

CRR binomial option pricing model without Financial Toolbox

Inputs: X – strike, S_0 – stock price, r – risk free interest rate, σ – volatility, dt – size of time steps, $steps$ – number of time steps to calculate, $oType$ – must be 'PUT' or 'CALL', $earlyExercise$ – true for American, false for European.



```

1  function oPrice = binPriceCRR(X,S0,r,sig,dt,steps,oType,earlyExercise)
2  % Function to calculate the option price of a vanilla European or American
3  % Put or Call option using a Cox Ross Rubinstein binomial tree.
4  %
5  % Author: Phil Goddard (phil@goddardconsulting.ca)
6
7  % Calculate the Cox Ross Rubinstein model parameters
8  a = exp(r*dt);
9  u = exp(sig*sqrt(dt));
10 d = 1/u;
11 p = (a-d)/(u-d);
12
13 % Loop over each node and calculate the CRR underlying price tree
14 priceTree = nan(steps+1,steps+1);
15 priceTree(1,1) = S0;
16 for idx = 2:steps+1
17     priceTree(1:idx-1,idx) = priceTree(1:idx-1,idx-1)*u;
18     priceTree(idx,idx) = priceTree(idx-1,idx-1)*d;
19 end
20
21 % Calculate the value at expiry
22 valueTree = nan(size(priceTree));
23 switch oType
24     case 'PUT'
25         valueTree(:,end) = max(X-priceTree(:,end),0);
26     case 'CALL'
27         valueTree(:,end) = max(priceTree(:,end)-X,0);
28 end
  
```

```

29
30 % Loop backwards to get values at the earlier times
31 steps = size(priceTree,2)-1;
32 for idx = steps:-1:1
33     valueTree(1:idx,idx) = ...
34         exp(-r*dt)*(p*valueTree(1:idx,idx+1) ...
35         + (1-p)*valueTree(2:idx+1,idx+1));
36     if earlyExercise
37         switch oType
38             case 'PUT'
39                 valueTree(1:idx,idx) = ...
40                     max(X-priceTree(1:idx,idx),valueTree(1:idx,idx));
41             case 'CALL'
42                 valueTree(1:idx,idx) = ...
43                     max(priceTree(1:idx,idx)-X,valueTree(1:idx,idx));
44         end
45     end
46 end
47
48 % Output the option price
49 oPrice = valueTree(1);

```

Command Window

```
>> oPrice = binPriceCRR(60,50,0.05,0.2,0.01,100,'CALL',false)
```

```
oPrice =
```

```
1.6279
```

```
f >> |
```

Drifted CRR binomial option pricing model without Financial Toolbox

Inputs: X – strike, S_0 - stock price, r - risk free interest rate, σ – volatility, dt - size of time steps, $steps$ - number of time steps to calculate, $oType$ - must be 'PUT' or 'CALL', $earlyExercise$ - true for American, false for European, $drift$ – will be the drift...

```

1  function oPrice=binPriceCRRDrift(X,S0,r,sig,dt,steps,oType,earlyExercise,drift)
2      % Author: Phil Goddard (phil@goddardconsulting.ca)
3
4      % Calculate the CRR With Drift model parameters
5      a = exp(r*dt);
6      u = exp(drift*dt + sig*sqrt(dt));
7      d = exp(drift*dt - sig*sqrt(dt));
8      p = (a-d)/(u-d);
9
10     % Loop over each node and calculate the underlying price tree
11     priceTree = nan(steps+1,steps+1);
12     priceTree(1,1) = S0;
13     for idx = 2:steps+1
14         priceTree(1:idx-1,idx) = priceTree(1:idx-1,idx-1)*u;
15         priceTree(idx,idx) = priceTree(idx-1,idx-1)*d;
16     end
17
18     % Calculate the value at expiry
19     valueTree = nan(size(priceTree));
20     switch oType
21         case 'PUT'
22             valueTree(:,end) = max(X-priceTree(:,end),0);
23         case 'CALL'
24             valueTree(:,end) = max(priceTree(:,end)-X,0);
25     end
26
27     % Loop backwards to get values at the earlier times
28     steps = size(priceTree,2)-1;
29     for idx = steps:-1:1
30         valueTree(1:idx,idx) = ...
31             exp(-r*dt)*(p*valueTree(1:idx,idx+1) ...
32                 + (1-p)*valueTree(2:idx+1,idx+1));
33         if earlyExercise
34             switch oType
35                 case 'PUT'
36                     valueTree(1:idx,idx) = ...
37                         max(X-priceTree(1:idx,idx),valueTree(1:idx,idx));
38                 case 'CALL'
39                     valueTree(1:idx,idx) = ...
40                         max(priceTree(1:idx,idx)-X,valueTree(1:idx,idx));
41             end
42         end
43     end
44
45     % Output the option price
46     oPrice = valueTree(1);

```

Jarrow-Rudd binomial option pricing model without Financial Toolbox

Inputs: X – strike, S_0 - stock price, r - risk free interest rate, σ – volatility, dt - size of time steps, $steps$ - number of time steps to calculate, $oType$ - must be 'PUT' or 'CALL', $earlyExercise$ - true for American, false for European

```

1  function oPrice = binPriceJR(X,S0,r,sig,dt,steps,oType,earlyExercise)
2  % Function to calculate the price of a vanilla European or American
3  % Put or Call option using an equal-probability (Jarrow-Rudd) binomial tree.
4
5  % Author: Phil Goddard (phil@goddardconsulting.ca)
6
7  % Calculate the equal probability model parameters
8  p = 0.5;
9  u = exp((r-sig*sig/2)*dt + sig*sqrt(dt));
10 d = exp((r-sig*sig/2)*dt - sig*sqrt(dt));
11
12 % Loop over each node and calculate the JR underlying price tree
13 priceTree = nan(steps+1,steps+1);
14 priceTree(1,1) = S0;
15 for idx = 2:steps+1
16     priceTree(1:idx-1,idx) = priceTree(1:idx-1,idx-1)*u;
17     priceTree(idx,idx) = priceTree(idx-1,idx-1)*d;
18 end
19
20 % Calculate the value at expiry
21 valueTree = nan(size(priceTree));
22 switch oType
23     case 'PUT'
24         valueTree(:,end) = max(X-priceTree(:,end),0);
25     case 'CALL'
26         valueTree(:,end) = max(priceTree(:,end)-X,0);
27 end

```



```

28
29 % Loop backwards to get values at the earlier times
30 steps = size(priceTree,2)-1;
31 for idx = steps:-1:1
32     valueTree(1:idx,idx) = ...
33         exp(-r*dt)*(p*valueTree(1:idx,idx+1) ...
34         + (1-p)*valueTree(2:idx+1,idx+1));
35     if earlyExercise
36         switch oType
37             case 'PUT'
38                 valueTree(1:idx,idx) = ...
39                     max(X-priceTree(1:idx,idx),valueTree(1:idx,idx));
40             case 'CALL'
41                 valueTree(1:idx,idx) = ...
42                     max(priceTree(1:idx,idx)-X,valueTree(1:idx,idx));
43         end
44     end
45 end
46
47 % Output the option price
48 oPrice = valueTree(1);

```

Command Window

```
>> oPrice=binPriceCRRDrift(60,50,0.05,0.2,0.01,100,'CALL',false,log(60)-log(50))
```

```
oPrice =
```

```
1.6115
```

```
fx >> |
```

Jarrow-Rudd binomial risk neutral option pricing model without Financial Toolbox

Inputs: X – strike, S_0 – stock price, r – risk free interest rate, σ – volatility, dt – size of time steps, $steps$ – number of time steps to calculate, $oType$ – must be 'PUT' or 'CALL', $earlyExercise$ – true for American, false for European

```

EDITOR PUBLISH VIEW
1 function oPrice = binPriceJRRN(X,S0,r,sig,dt,steps,oType,earlyExercise)
2 % Function to calculate the price of a vanilla European or American
3 % Put or Call option using a Jarrow-Rudd Risk Neutral binomial tree.
4
5 % Author: Phil Goddard (phil@goddardconsulting.ca)
6
7 % Calculate the JRRN model parameters
8 a = exp(r*dt);
9 u = exp((r-sig*sig/2)*dt + sig*sqrt(dt));
10 d = exp((r-sig*sig/2)*dt - sig*sqrt(dt));
11 p = (a-d)/(u-d);
12
13 % Loop over each node and calculate the JRRN underlying price tree
14 priceTree = nan(steps+1,steps+1);
15 priceTree(1,1) = S0;
16 for idx = 2:steps+1
17     priceTree(1:idx-1,idx) = priceTree(1:idx-1,idx-1)*u;
18     priceTree(idx,idx) = priceTree(idx-1,idx-1)*d;
19 end
20
21 % Calculate the value at expiry
22 valueTree = nan(size(priceTree));
23 switch oType
24     case 'PUT'
25         valueTree(:,end) = max(X-priceTree(:,end),0);
26     case 'CALL'
27         valueTree(:,end) = max(priceTree(:,end)-X,0);
28
29 end
30 % Loop backwards to get values at the earlier times
31 steps = size(priceTree,2)-1;
32 for idx = steps:-1:1
33     valueTree(1:idx,idx) = ...
34         exp(-r*dt)*(p*valueTree(1:idx,idx+1) ...
35         + (1-p)*valueTree(2:idx+1,idx+1));
36     if earlyExercise
37         switch oType
38             case 'PUT'
39                 valueTree(1:idx,idx) = ...
40                     max(X-priceTree(1:idx,idx),valueTree(1:idx,idx));
41             case 'CALL'
42                 valueTree(1:idx,idx) = ...
43                     max(priceTree(1:idx,idx)-X,valueTree(1:idx,idx));
44         end
45     end
46 end
47
48 % Output the option price
49 oPrice = valueTree(1);

```

```

Command Window

>> oPrice=binPriceJRRN(60,50,0.05,0.2,0.01,100,'CALL',false)

oPrice =

    1.6239

fx >> |

```

Leisen-Reimer binomial option pricing model without Financial Toolbox

```

EDITOR PUBLISH VIEW

1 function oPrice = binPriceLR(X,S0,r,sig,dt,steps,oType,earlyExercise)
2 % Function to calculate the price of a vanilla European or American
3 % Put or Call option using a Leisen-Reimer binomial tree.
4
5 % Author: Phil Goddard (phil@goddardconsulting.ca)
6
7 % Create a function handle for the required cumulative distribution
8 % approximation
9 fh = @(z,n) (0.5+sign(z)...
10     .*sqrt(0.25-0.25*exp(-(z/(n+1/3+0.1/(n+1))).^2)*(n+1/6))));
11
12 % Calculate the d1 and d2 parameters of the Black-Scholes model
13 T = dt*steps ; % time to maturity
14 d1 = (log(S0/X)+(r+sig*sig/2)*T)/sig/sqrt(T);
15 d2 = (log(S0/X)+(r-sig*sig/2)*T)/sig/sqrt(T);
16
17 % Calculate the Leisen-Reimer model parameters
18 % The parameter n must be odd
19 if mod(steps,2) > 0
20     n = steps;
21 else
22     n = steps+1;
23 end
24 pbar = fh(d1,n);
25 p = fh(d2,n);
26 u = exp(r*dt)*pbar/p;
27 d = (exp(r*dt)-p*u)/(1-p);

```

```

28
29 % Loop over each node and calculate the CRR underlying price tree
30 priceTree = nan(steps+1,steps+1);
31 priceTree(1,1) = S0;
32 for idx = 2:steps+1
33     priceTree(1:idx-1,idx) = priceTree(1:idx-1,idx-1)*u;
34     priceTree(idx,idx) = priceTree(idx-1,idx-1)*d;
35 end
36
37 % Calculate the value at expiry
38 valueTree = nan(size(priceTree));
39 switch oType
40     case 'PUT'
41         valueTree(:,end) = max(X-priceTree(:,end),0);
42     case 'CALL'
43         valueTree(:,end) = max(priceTree(:,end)-X,0);
44 end
45
46 % Loop backwards to get values at the earlier times
47 steps = size(priceTree,2)-1;
48 for idx = steps:-1:1
49     valueTree(1:idx,idx) = ...
50         exp(-r*dt)*(p*valueTree(1:idx,idx+1) ...
51         + (1-p)*valueTree(2:idx+1,idx+1));
52     if earlyExercise
53         switch oType
54             case 'PUT'
55                 valueTree(1:idx,idx) = ...
56                     max(X-priceTree(1:idx,idx),valueTree(1:idx,idx));
57             case 'CALL'
58                 valueTree(1:idx,idx) = ...
59                     max(priceTree(1:idx,idx)-X,valueTree(1:idx,idx));
60         end
61     end
62 end
63
64 % Output the option price
65 oPrice = valueTree(1);

```

Command Window

```
>> oPrice = binPriceLR(60,50,0.05,0.2,0.01,100,'CALL',false)
```

```
oPrice =
```

```
1.5931
```

```
f_x >> |
```

Tian binomial option pricing model without Financial Toolbox

```

1  function oPrice = binPriceTian(X,S0,r,sig,dt,steps,oType,earlyExercise)
2  % Function to calculate the price of a vanilla European or American
3  % Put or Call option using a Tian binomial tree.
4
5  % Author: Phil Goddard (phil@goddardconsulting.ca)
6
7  % Calculate the Tian model parameters
8  nu = exp(sig*sig*dt);
9  a = exp(r*dt);
10 u = 0.5*a*nu*(nu+1+sqrt(nu*nu+2*nu-3));
11 d = 0.5*a*nu*(nu+1-sqrt(nu*nu+2*nu-3));
12 p = (a-d)/(u-d);
13
14 % Loop over each node and calculate the Tian underlying price tree
15 priceTree = nan(steps+1,steps+1);
16 priceTree(1,1) = S0;
17 for idx = 2:steps+1
18     priceTree(1:idx-1,idx) = priceTree(1:idx-1,idx-1)*u;
19     priceTree(idx,idx) = priceTree(idx-1,idx-1)*d;
20 end
21
22 % Calculate the value at expiry
23 valueTree = nan(size(priceTree));
24 switch oType
25     case 'PUT'
26         valueTree(:,end) = max(X-priceTree(:,end),0);
27     case 'CALL'

```

```

28 -         valueTree(:,end) = max(priceTree(:,end)-X,0);
29 -     end
30 -
31 -     % Loop backwards to get values at the earlier times
32 -     steps = size(priceTree,2)-1;
33 -     for idx = steps:-1:1
34 -         valueTree(1:idx,idx) = ...
35 -             exp(-r*dt)*(p*valueTree(1:idx,idx+1) ...
36 -                 + (1-p)*valueTree(2:idx+1,idx+1));
37 -         if earlyExercise
38 -             switch oType
39 -                 case 'PUT'
40 -                     valueTree(1:idx,idx) = ...
41 -                         max(X-priceTree(1:idx,idx),valueTree(1:idx,idx));
42 -                 case 'CALL'
43 -                     valueTree(1:idx,idx) = ...
44 -                         max(priceTree(1:idx,idx)-X,valueTree(1:idx,idx));
45 -             end
46 -         end
47 -     end
48 -
49 -     % Output the option price
50 -     oPrice = valueTree(1);

```

Command Window

```

>> oPrice=binPriceTian(60,50,0.05,0.2,0.01,100,'CALL',false)

oPrice =

    1.6293

fx >> |

```

Portfolios models

Global Minimum Variance Efficient Portfolios (GMVP) without Toolboxes using langrangien multiplier

This is based on the formal solution see during the theoretical training (Langrang Multiplier):

```

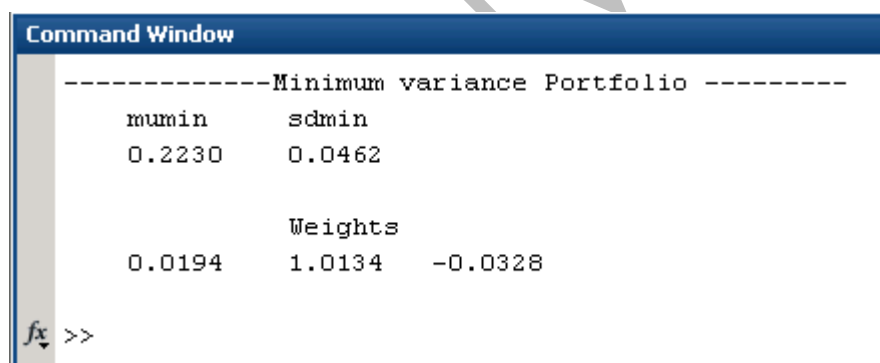
EDITOR PUBLISH VIEW
1 - clear all;close all;clc
2 - returns = xlsread('E:\MarkowitzData.xls');
3
4 - muAsset=mean(returns);
5 - muAsset=muAsset';
6 - sigmaAsset=cov(returns);
7 - N = length(muAsset);
8 - % Calculation of parameters sigma*sigma = A B C D
9 - A = ones(N,1)'*inv(sigmaAsset)*muAsset;
10 - B = muAsset'*inv(sigmaAsset)*muAsset;
11 - C = ones(N,1)'*inv(sigmaAsset)*ones(N,1);
12 - D = B*C - A^2;
13 - H = (B*inv(sigmaAsset)*ones(N,1) - A*inv(sigmaAsset)*muAsset)/D;
14 - G = (C*inv(sigmaAsset)*muAsset - A*inv(sigmaAsset)*ones(N,1))/D;
15 - %Minimum variance portfolio
16 - HH = H'*sigmaAsset*H;
17 - GG = G'*sigmaAsset*G;
18 - gh = H'*sigmaAsset*G;
19
20 - mumin = - gh/GG;
21 - sdmin = sqrt( HH * ( 1 - gh^2/(HH*GG)) );
22 - % Any portfolio is on the efficient frontier can be expressed as W = Ge + H
23 - wmin = (G*mumin + H)';
24 - disp('-----Minimum variance Portfolio -----');
25 - disp('      mumin      sdmin      ');
26 - disp([mumin,sdmin]);
27 - disp('              Weights              ');
28 - disp(wmin);
29
30 - muPortfolio = linspace(min(muAsset),max(muAsset),50); %muP grid

```

```

30 - muPortfolio = linspace(min(muAsset),max(muAsset),50);    %muP grid
31 - sigmaPortfolio = zeros(1,50); %Storage
32 -
33 - for i=1:50 ;
34 -     wPortfolio = H + muPortfolio(i)*G;% weight of ith portfolio
35 -     sigmaPortfolio(i) = sqrt(wPortfolio'*sigmaAsset*wPortfolio) ;%risk
36 - end ;
37 -
38 - fsize = 16 ;
39 - ind = (muPortfolio > mumin);    %Indicates efficient horizon
40 - ind2 = (muPortfolio < mumin);    %Indicates locus below efficient horizon
41 -
42 - %Create plot - efficient horizon is shown as a solid curve
43 - %- the inefficient part of the locus is dashed
44 - p1 = plot(sigmaPortfolio(ind),muPortfolio(ind),'-',...
45 -     sigmaPortfolio(ind2),muPortfolio(ind2),'--' , ...
46 -     sdmin,mumin,'.') ;
47 -
48 - %Change line widths, marker sizes, and colors for better appearance
49 - set(p1(1:2),'linewidth',4) ;
50 - set(p1(1:2),'color','blue') ;
51 - set(p1(3),'markersize',40) ;
52 - set(p1(3),'color','red') ;
53 - xlabel('Risk(standard deviation)','fontsize',fsize) ;
54 - ylabel('Expected return(mean)','fontsize',fsize) ;
55 - grid on;

```



```

Command Window
-----Minimum variance Portfolio -----
      mumin      sdmin
      0.2230      0.0462

      Weights
      0.0194      1.0134      -0.0328

fx >>

```

Global Minimum Variance Efficient Portfolios (GMVP) using only Optimization Toolbox

For this first classical example we will use the following file:

MeanVariancePortfolioReturn.txt - Bloc-notes

Fichier Edition Format Affichage ?

ABC	-7.022472	2.758133	-4.335857	6.978417	5.312709
ANZBY	-9.9270704	5.926388	-6.124853	5.834379	-3.556607
APD	1.621528	6.297742	1.069946	-7.20493	2.55406 .11622
APU	1.148649	-2.004008	-.613497	9.94513 -1.840302	4.3215
ASH	5.134116	6.358381	3.335832	-.3264418	1.564774
ATO	1.279708	-.4693141	-2.067465	-2.592592	7.566544
BDX	-.2640845	5.684025	-2.421914	.1711743	-1.828433
BEN	-2.569993	3.43354 -2.194045	.0436999	5.037856	6.7091
BJS	3.266008	3.953392	3.843075	-6.013878	3.281379
CFN	-.0277778	1.333704	-5.017823	-1.991917	-2.444775
CHFN	-8.554438	-9.47974	-7.792208	-5.543902	10.40609
COL	8.77282 7.342658	3.344191	-3.593192	7.650391	-3.462
CUB	-9.054647	-17.80702	1.067236	-7.127772	.0568527
DMF	-4.7 -.8394544	-5.502645	.3359463	1.897323	1.5334
EMR	-4.079886	-1.368233	-2.08082	-3.480135	6.062545
FICO	-5.806979	-2.170767	1.893491	-4.529617	4.166667
GLAD	1.434599	2.329451	-13.73984	3.675778	3.272724
HP	11.3396 5.646438	-.8491508	-3.173804	7.908428	13.114
IGT	-8.958697	-2.683706	-12.47538	.8627157	4.797324
IMKTA	5.972558	-.9139376	2.382782	-2.927928	-2.242843
ITYBY	-3.710407	1.842105	-1.882614	9.123401	-6.154114
JJSF	-1.590538	-2.154994	-.8047438	4.526046	.8169947
LDR	2.275711	2.931108	-1.184785	-4.606647	7.717748
LEO	2.256532	-.8130081	-4.332553	2.570379	2.386634
LG	-2.728732	3.564356	-6.9471 -6.267123	9.243696	6.2207
MATW	-5.679348	.201671 -5.807936	8.333333	4.226547	5.3257
MMS	-3.341902	12.56649	-1.092735	-8.330845	11.69381
NABZY	2.408564	-.5226481	-3.896672	4.236902	3.933569
NASB	0	5.155155	-5.759162	-1.237374	-.3068226
NFG	-.4940014	.4255319	.9533898	-4.756908	2.827764
NJR	1.176878	1.265823	-1.959459	-3.790489	4.012451
NRGY	12	1.242236	-.7055215	1.173926	-4.366413
PBNY	-3.33586	.8627451	-4.821151	-13.48039	10.57602
PIM	.9146342	.6042296	-4.054054	-.9389671	1.105847
QCOM	-12.16981	-3.195489	1.608877	-4.750205	6.82144 -11.43
QQQQ	-6.312625	-.4812834	-1.746373	-4.347826	8.862212
RDK	-2.720148	11.84834	-1.90678	-2.980561	6.990208
RJF	.62954 -2.26179	-.5416051	-10.9901	-.2224664	4.9609
RKT	-8.509234	2.883922	-6.797477	-23.68421	13.39902

EDITOR PUBLISH VIEW

```

1 %Authors: Ludwig. B. Chincarini / Daewhan Kim
2 clear all;close all;clc
3 %we read the ticker list dand
4 %ticker list is the first column
5 %we skip all the other columns by using (%*[^\\n])
6 ticker=textread('C:\tmp\MeanVariancePortfolioReturn.txt','%s*[^\\n]');
```

Command Window

```
>> ticker

ticker =

    'ABC'
    'ANZBY'
    'APD'
    'APU'
    'ASH'
    'ATO'
    'BDX'
    'BEN'
    'BJS'
    'CFFN'
    'CHFN'
    'COL'
    'CUB'
    'DMF'
    'EMR'
    'ETICO'
```

EDITOR PUBLISH VIEW

```
1 - clear all;close all;clc;
2   % we read the ticker list and
3   % ticker list is the first column
4   % we skip all the other columns by using (%*[^\\n])
5 - ticker=textread('data_Ch9_return.txt','%s*[^\\n]');
6   % we read returns in the file where the delimiter is the Tab (\\t)
7   % we read from the first row (0) and from the second column (1)
8 - ret = dlmread('data_Ch9_return.txt', '\\t', 0, 1);
```

Command Window

```
>> ret

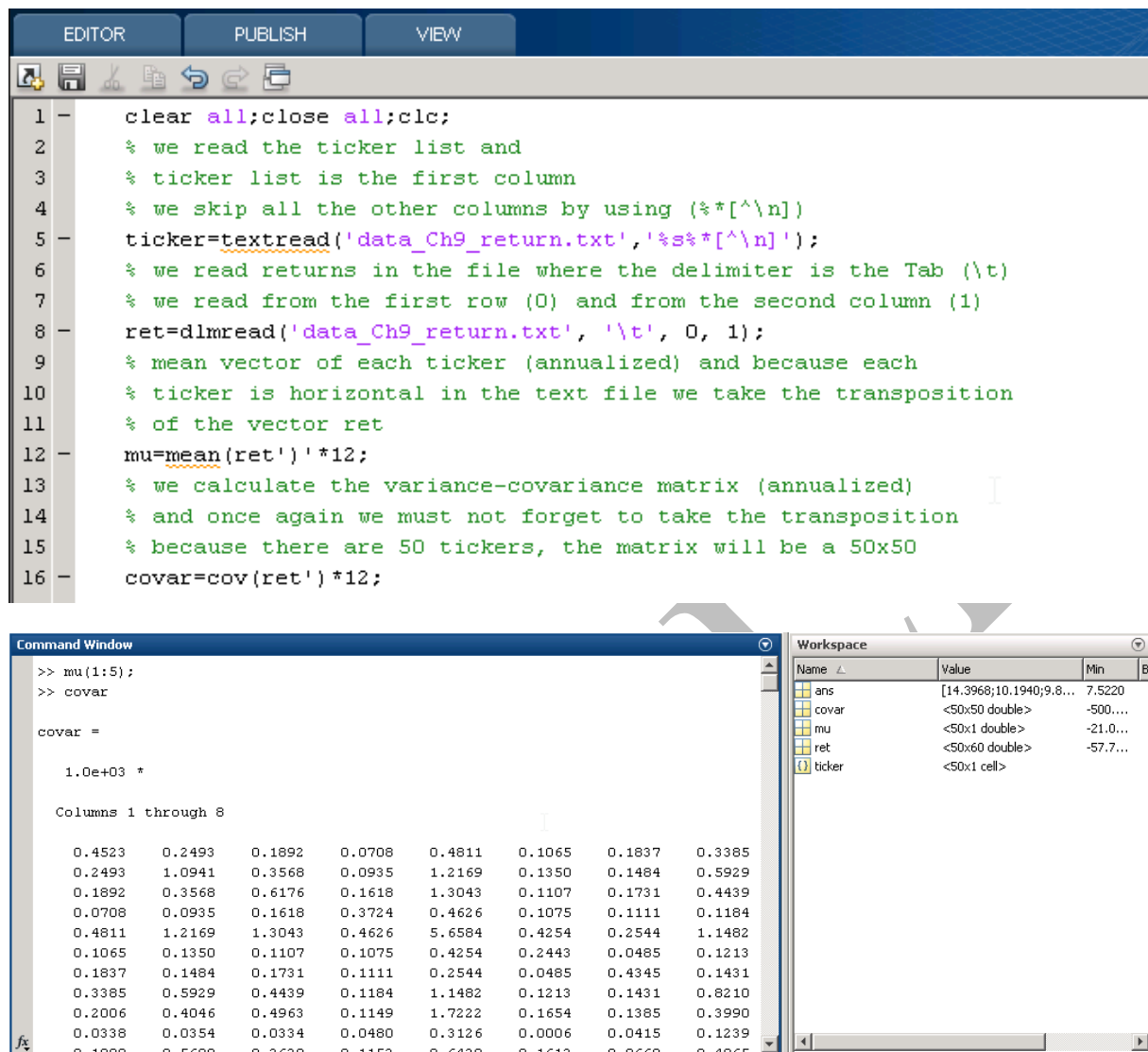
ret =

Columns 1 through 8

    -0.7022    2.7581   -4.3359    6.9784    5.3127    7.0881    3.8760    3.9610
    -0.9271    5.9264   -6.1249    5.8344   -3.5566    1.6595   -1.6324    2.8887
     1.6215    6.2977    1.0699   -7.2049    2.5541    0.1162   -0.8955   -7.2958
     1.1486   -2.0040   -0.6135    9.9451   -1.8403    4.3216    1.8581   -1.4952
     5.1341    6.3584    3.3358   -0.3264    1.5648    5.2311    4.6136   -1.0740
     1.2797   -0.4693   -2.0675   -2.5926    7.5665    1.8028    1.2500    1.4746
    -0.2641    5.6840   -2.4219    0.1712   -1.8284   -8.6684    5.5270   -4.9485
    -2.5700    3.4335   -2.1940    0.0437    5.0379    6.7092    4.9883   -0.4702
     3.2660    3.9534    3.8431   -6.0139    3.2814    4.2097   16.2348    3.4098
    -0.0278    1.3337   -5.0178   -1.9919   -2.4448    4.1063    4.4084   -6.2222
    -8.5544   -9.4797   -7.7922   -5.5439   10.4061    0.4023    0.1717   -1.9429
     8.7728    7.3427    3.3442   -3.5932    7.6504   -3.4622    2.3490   -1.3729
    -9.0546   -17.8070    1.0672   -7.1278    0.0569    0.7955    9.7520   -5.2388
     4.7000     0.8305     5.5026     0.3250     1.8073     1.5234     0.1070     0.1080
```

Workspace

Name	Value	Min	B
ret	<50x60 double>	-57.7...	
ticker	<50x1 cell>		



The image shows a MATLAB environment with an Editor window and a Command Window. The Editor window contains a script that reads data from 'data_Ch9_return.txt' and calculates a covariance matrix. The Command Window shows the execution of the script, displaying the values of 'mu' and 'covar'.

Editor Window Code:

```

1 - clear all;close all;clc;
2   % we read the ticker list and
3   % ticker list is the first column
4   % we skip all the other columns by using (%*[^\\n])
5 - ticker=textread('data_Ch9_return.txt','%s*[^\\n]');
6   % we read returns in the file where the delimiter is the Tab (\\t)
7   % we read from the first row (0) and from the second column (1)
8 - ret=dlmread('data_Ch9_return.txt','\\t',0,1);
9   % mean vector of each ticker (annualized) and because each
10  % ticker is horizontal in the text file we take the transposition
11  % of the vector ret
12 - mu=mean(ret')'*12;
13  % we calculate the variance-covariance matrix (annualized)
14  % and once again we must not forget to take the transposition
15  % because there are 50 tickers, the matrix will be a 50x50
16 - covar=cov(ret')*12;

```

Command Window Output:

```

>> mu(1:5);
>> covar

covar =

    1.0e+03 *

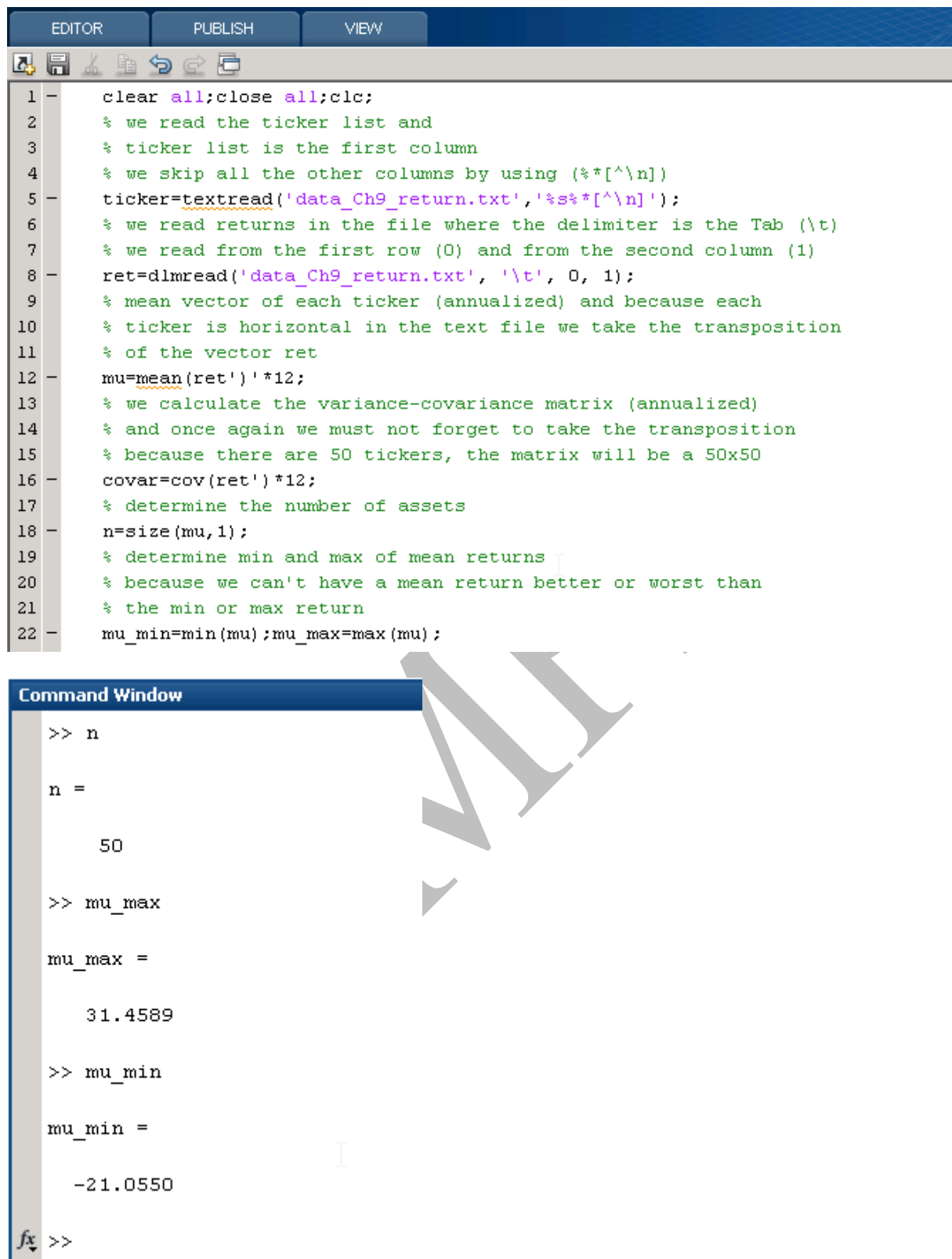
Columns 1 through 8

    0.4523    0.2493    0.1892    0.0708    0.4811    0.1065    0.1837    0.3385
    0.2493    1.0941    0.3568    0.0935    1.2169    0.1350    0.1484    0.5929
    0.1892    0.3568    0.6176    0.1618    1.3043    0.1107    0.1731    0.4439
    0.0708    0.0935    0.1618    0.3724    0.4626    0.1075    0.1111    0.1184
    0.4811    1.2169    1.3043    0.4626    5.6584    0.4254    0.2544    1.1482
    0.1065    0.1350    0.1107    0.1075    0.4254    0.2443    0.0485    0.1213
    0.1837    0.1484    0.1731    0.1111    0.2544    0.0485    0.4345    0.1431
    0.3385    0.5929    0.4439    0.1184    1.1482    0.1213    0.1431    0.8210
    0.2006    0.4046    0.4963    0.1149    1.7222    0.1654    0.1385    0.3990
    0.0338    0.0354    0.0334    0.0480    0.3126    0.0006    0.0415    0.1239
    0.1890    0.5590    0.3628    0.1152    0.6420    0.1612    0.0668    0.4865

```

Workspace Window:

Name	Value	Min	Max
ans	[14.3968;10.1940;9.8...	7.5220	
covar	<50x50 double>	-500....	
mu	<50x1 double>	-21.0...	
ret	<50x60 double>	-57.7...	
ticker	<50x1 cell>		



The image shows a MATLAB environment with two windows: the Editor and the Command Window. The Editor window displays a script with 22 lines of MATLAB code. The Command Window shows the execution of the first three lines of the script, displaying the values of the variables `n`, `mu_max`, and `mu_min`.

Editor Window:

```

1 - clear all;close all;clc;
2   % we read the ticker list and
3   % ticker list is the first column
4   % we skip all the other columns by using (%*[^\\n])
5 - ticker=textread('data_Ch9_return.txt','%s*[^\\n]');
6   % we read returns in the file where the delimiter is the Tab (\\t)
7   % we read from the first row (0) and from the second column (1)
8 - ret=dlmread('data_Ch9_return.txt','\\t',0,1);
9   % mean vector of each ticker (annualized) and because each
10  % ticker is horizontal in the text file we take the transposition
11  % of the vector ret
12 - mu=mean(ret')'*12;
13  % we calculate the variance-covariance matrix (annualized)
14  % and once again we must not forget to take the transposition
15  % because there are 50 tickers, the matrix will be a 50x50
16 - covar=cov(ret')*12;
17  % determine the number of assets
18 - n=size(mu,1);
19  % determine min and max of mean returns
20  % because we can't have a mean return better or worst than
21  % the min or max return
22 - mu_min=min(mu);mu_max=max(mu);

```

Command Window:

```

>> n

n =

    50

>> mu_max

mu_max =

    31.4589

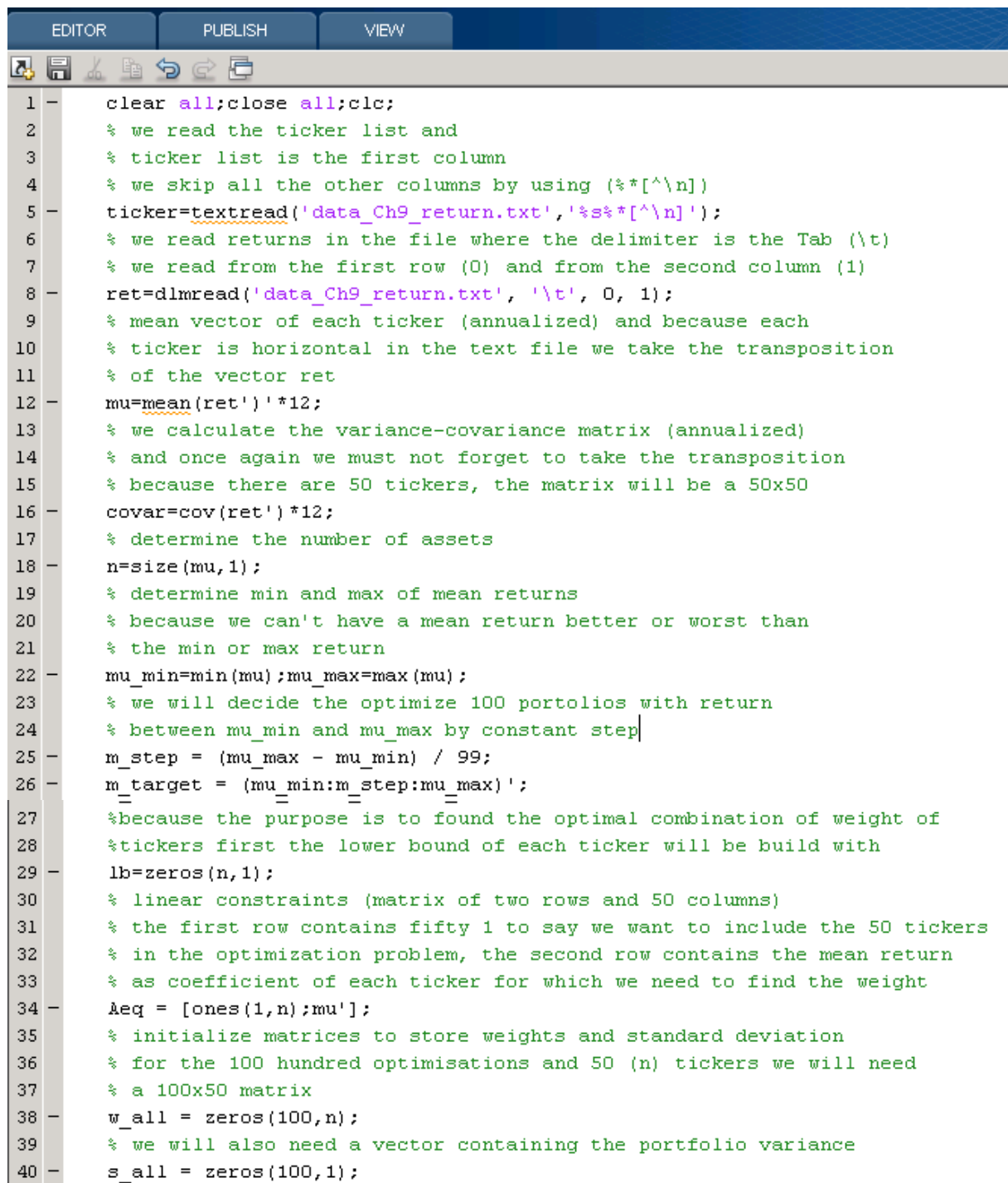
>> mu_min

mu_min =

   -21.0550

fx >>

```



```

1 - clear all;close all;clc;
2 - % we read the ticker list and
3 - % ticker list is the first column
4 - % we skip all the other columns by using (%*[^\\n])
5 - ticker=textread('data_Ch9_return.txt','%s*[^\\n]');
6 - % we read returns in the file where the delimiter is the Tab (\\t)
7 - % we read from the first row (0) and from the second column (1)
8 - ret=dlmread('data_Ch9_return.txt','\\t',0,1);
9 - % mean vector of each ticker (annualized) and because each
10 - % ticker is horizontal in the text file we take the transposition
11 - % of the vector ret
12 - mu=mean(ret')'*12;
13 - % we calculate the variance-covariance matrix (annualized)
14 - % and once again we must not forget to take the transposition
15 - % because there are 50 tickers, the matrix will be a 50x50
16 - covar=cov(ret')*12;
17 - % determine the number of assets
18 - n=size(mu,1);
19 - % determine min and max of mean returns
20 - % because we can't have a mean return better or worst than
21 - % the min or max return
22 - mu_min=min(mu);mu_max=max(mu);
23 - % we will decide the optimize 100 portolios with return
24 - % between mu_min and mu_max by constant step
25 - m_step = (mu_max - mu_min) / 99;
26 - m_target = (mu_min:m_step:mu_max)';
27 - %because the purpose is to found the optimal combination of weight of
28 - %tickers first the lower bound of each ticker will be build with
29 - lb=zeros(n,1);
30 - % linear constraints (matrix of two rows and 50 columns)
31 - % the first row contains fifty 1 to say we want to include the 50 tickers
32 - % in the optimization problem, the second row contains the mean return
33 - % as coefficient of each ticker for which we need to find the weight
34 - Aeq = [ones(1,n);mu'];
35 - % initialize matrices to store weights and standard deviation
36 - % for the 100 hundred optimisations and 50 (n) tickers we will need
37 - % a 100x50 matrix
38 - w_all = zeros(100,n);
39 - % we will also need a vector containing the portfolio variance
40 - s_all = zeros(100,1);

```

```

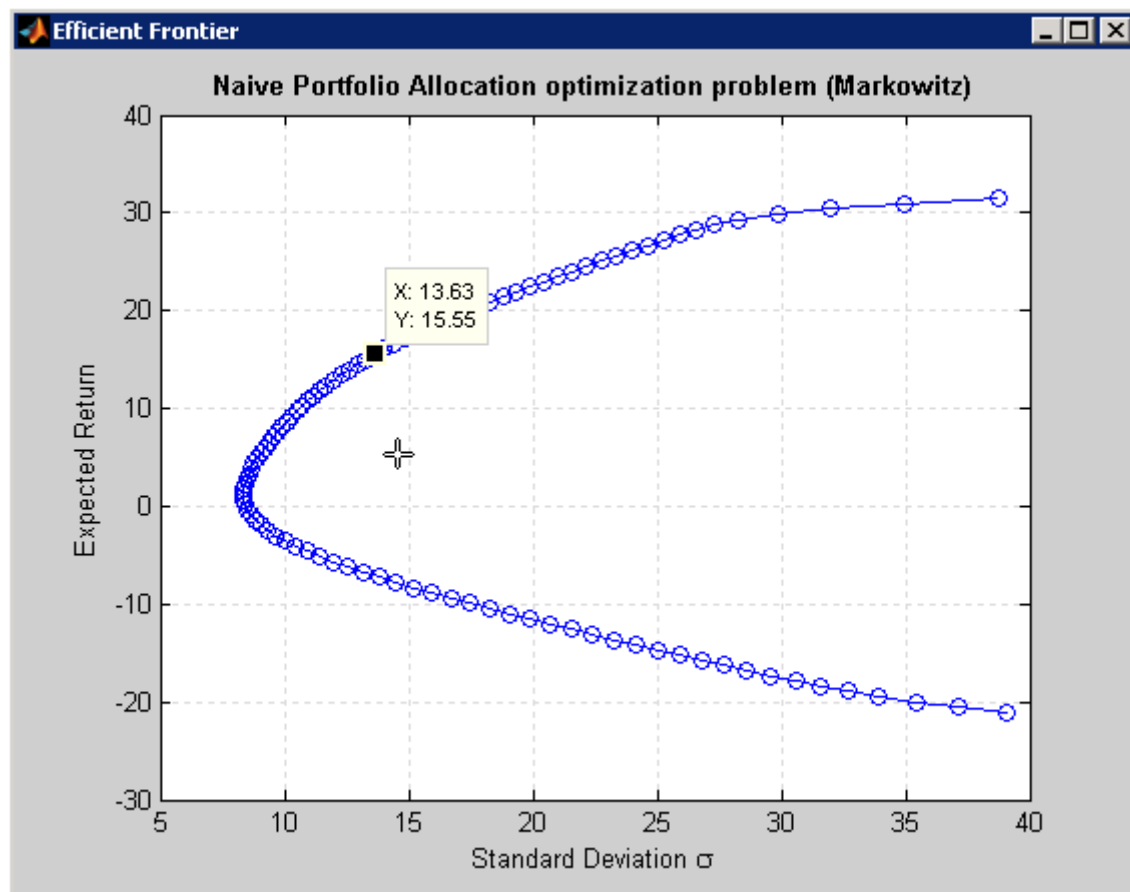
EDITOR PUBLISH VIEW
41 % we define the settings for the optimization function of the
42 % Optimization Toolbox
43 - opts = optimoptions('quadprog','Algorithm','interior-point-convex',...
44     'Display','off');
45
46 % we loop over target means
47 - for i = 1:100
48     % linear constraints
49     % first coefficient is the sum of weights, second is the target return
50 -     Beq = [1;m_target(i)];
51     % quadratic programming (maximization by default)
52 -     w_i = quadprog(covar,[],[],[],&eq,Beq,lb,[],[],opts);
53     % we compute the standard deviation of the optimized portfolio
54     % as we proved it in the Financial Mathematics training
55     % (thus this stdev includes non-independence of tickers)
56 -     s_i = sqrt(w_i'*covar*w_i);
57     % collect weights and standard deviation
58 -     w_all(i,:) = w_i;
59     s_all(i) = s_i;
60 - end
61

```

```

EDITOR PUBLISH VIEW
59 -     s_all(i) = s_i;
60 - end
61
62 % plot the mean-variance efficient frontier
63 - figure
64 - set(gcf,'Menubar','none','Name','Efficient Frontier','NumberTitle','off');
65 - plot(s_all,m_target,'-ob')
66 - xlabel('Standard Deviation \sigma');ylabel('Expected Return');
67 - title('Naive Portfolio Allocation optimization problem (Markowitz)',...
68     'fontweigh','bold');
69 - grid on;
70 - set(gca,'Xcolor',[0.85 0.85 0.85]);
71 - set(gca,'Ycolor',[0.85 0.85 0.85]);
72 - setAxes=copyobj(gca,gcf);
73 - set(setAxes,'color','none','Xcolor','k','xgrid','off','Ycolor','k','ygrid','off');
74 - datacursormode on;
75

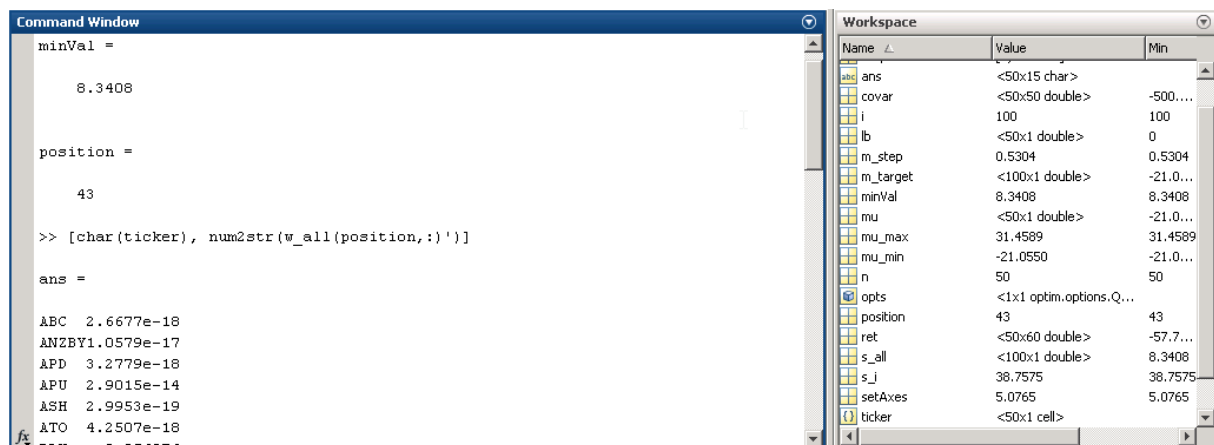
```



```

EDITOR    PUBLISH    VIEW
+  [icon]  [icon]  [icon]  [icon]  [icon]  [icon]
76     % we search the position number of minimum variance portfolio
77 -   [minVal, position]=min(s_all)
78
79     % we display weights of the selected portfolios
80     % we have to convert weights to text because tickers are also texts
81 -   [char(ticker), num2str(w_all(position,:))']

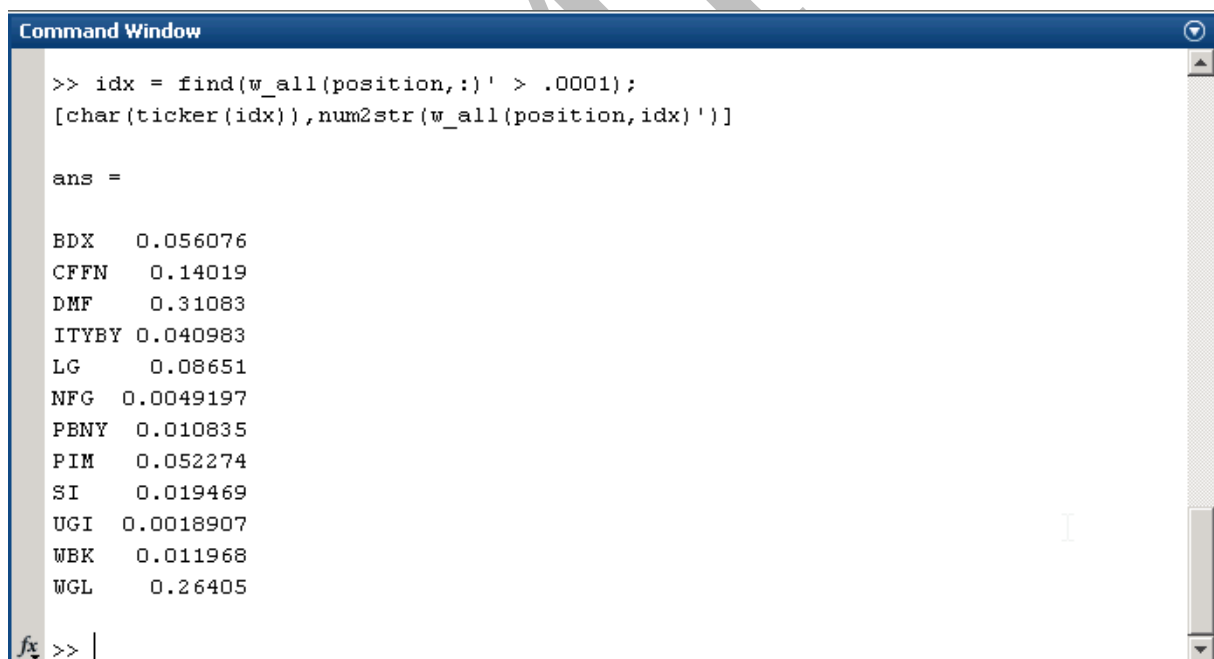
```



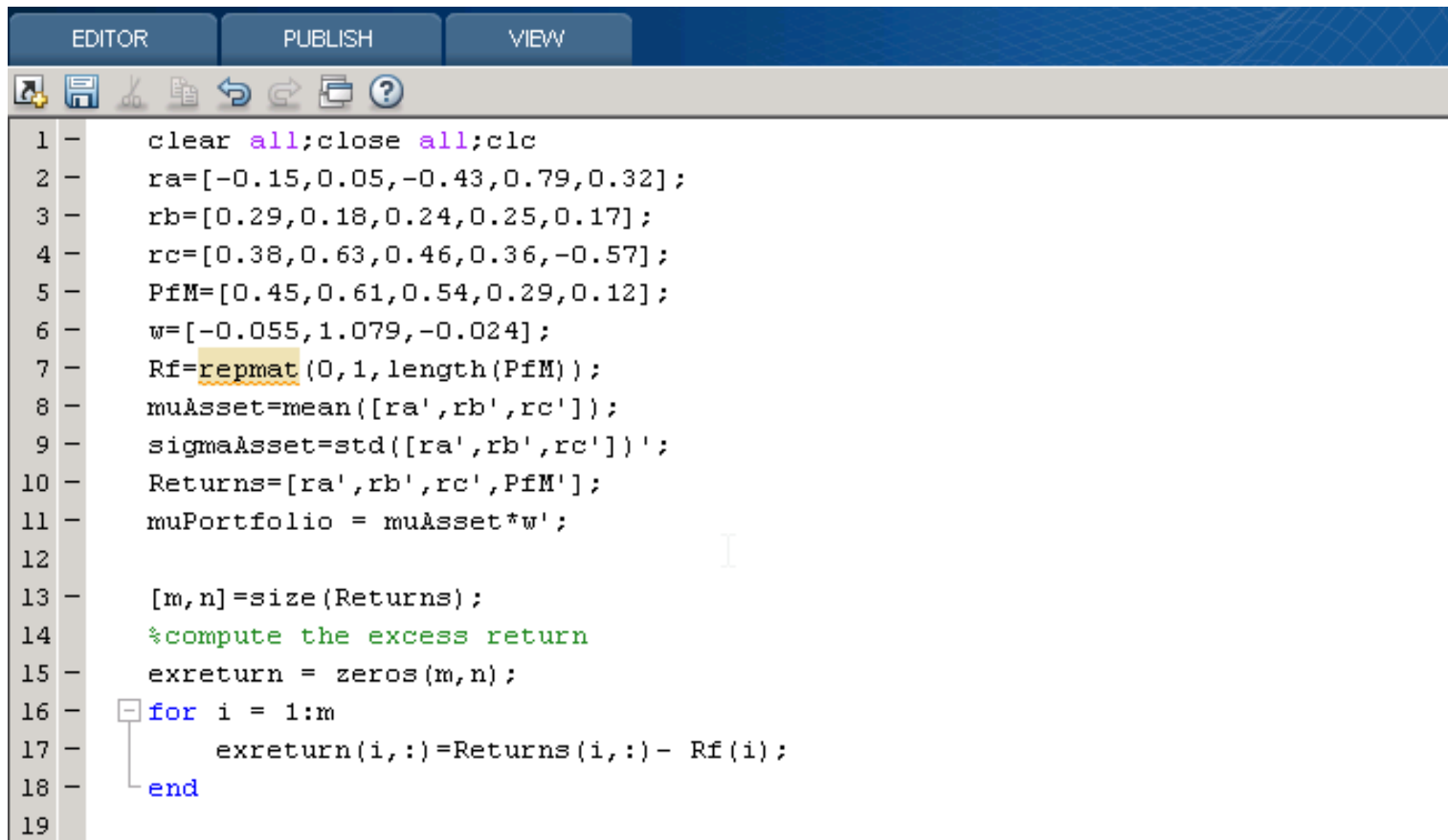
```

79 % we display weights of the selected portfolios
80 % we have to convert weights to text because tickers are also texts
81 - [char(ticker), num2str(w_all(position,:))']
82
83 % we remove too small tickers
84 % display weights of the selected portfolios
85 - idx = find(w_all(position,:) > .0001);
86 - [char(ticker(idx)), num2str(w_all(position,idx))']
87

```



Portfolio Performance



```
1 - clear all;close all;clc
2 - ra=[-0.15,0.05,-0.43,0.79,0.32];
3 - rb=[0.29,0.18,0.24,0.25,0.17];
4 - rc=[0.38,0.63,0.46,0.36,-0.57];
5 - PfM=[0.45,0.61,0.54,0.29,0.12];
6 - w=[-0.055,1.079,-0.024];
7 - Rf= repmat(0,1,length(PfM));
8 - muAsset=mean([ra',rb',rc']);
9 - sigmaAsset=std([ra',rb',rc'])';
10 - Returns=[ra',rb',rc',PfM'];
11 - muPortfolio = muAsset*w';
12
13 - [m,n]=size>Returns);
14 - %compute the excess return
15 - exreturn = zeros(m,n);
16 - for i = 1:m
17 -     exreturn(i,:)=Returns(i,:)- Rf(i);
18 - end
19
```

```

20 %regression with each risky asset on the portfolio, note that the
21 %last columns of the exreturn matrix is market premium data
22 - for i= 1:n-1
23 -     CovarianceMatrix = cov(exreturn(:,i),exreturn(:,n));% Covariance matrix
24 -     % beta = cov(x,y)/var(x);
25 -     Beta(i)= CovarianceMatrix(1,2)/var(exreturn(:,n));
26 -     % Alpha = return - (rf + beta(E(rm)-rf))
27 -     Alpha(i)= mean(exreturn(:,i))-(mean(Rf)+ Beta(i)*mean(exreturn(:,n)));
28 -     % rt = rf + beta(E(rm)-rf)
29 -     Rt(i) = mean(Rf) + Beta(i)*mean(exreturn(:,n));
30 -     %Calculating the Sharpe ratios for individual stocks
31 -     Sharpe(i) = mean(exreturn(:,i))/sigmaAsset(i);
32 -     %Calculating the Treynor ratios for individual stocks
33 -     Treynor(i) = mean(exreturn(:,i))/Beta(i);
34 - end
35
36 %Asset performance
37 - Alpha
38 - Beta
39 - Rt
40 - Treynor
41 - Sharpe
42 %Portfolio performance
43 - pA = muPortfolio - Rt*w'
44 - pB = Beta*w'
45 - pR = Rt*w'
46 - pT = Treynor*w'
47 - pS = Sharpe*w'

```

```

Command Window
Alpha =
    0.7186    0.2041   -0.6029
Beta =
   -1.4990    0.0545    2.1265
Rt =
   -0.6026    0.0219    0.8549
Treydor =
   -0.0774    4.1463    0.1185
Sharpe =
    0.2489    4.4931    0.5343
pA =
    0.1952
pB =
    0.0902
pR =
    0.0363
pT =
    4.4753
pS =
    4.8216
fx >>

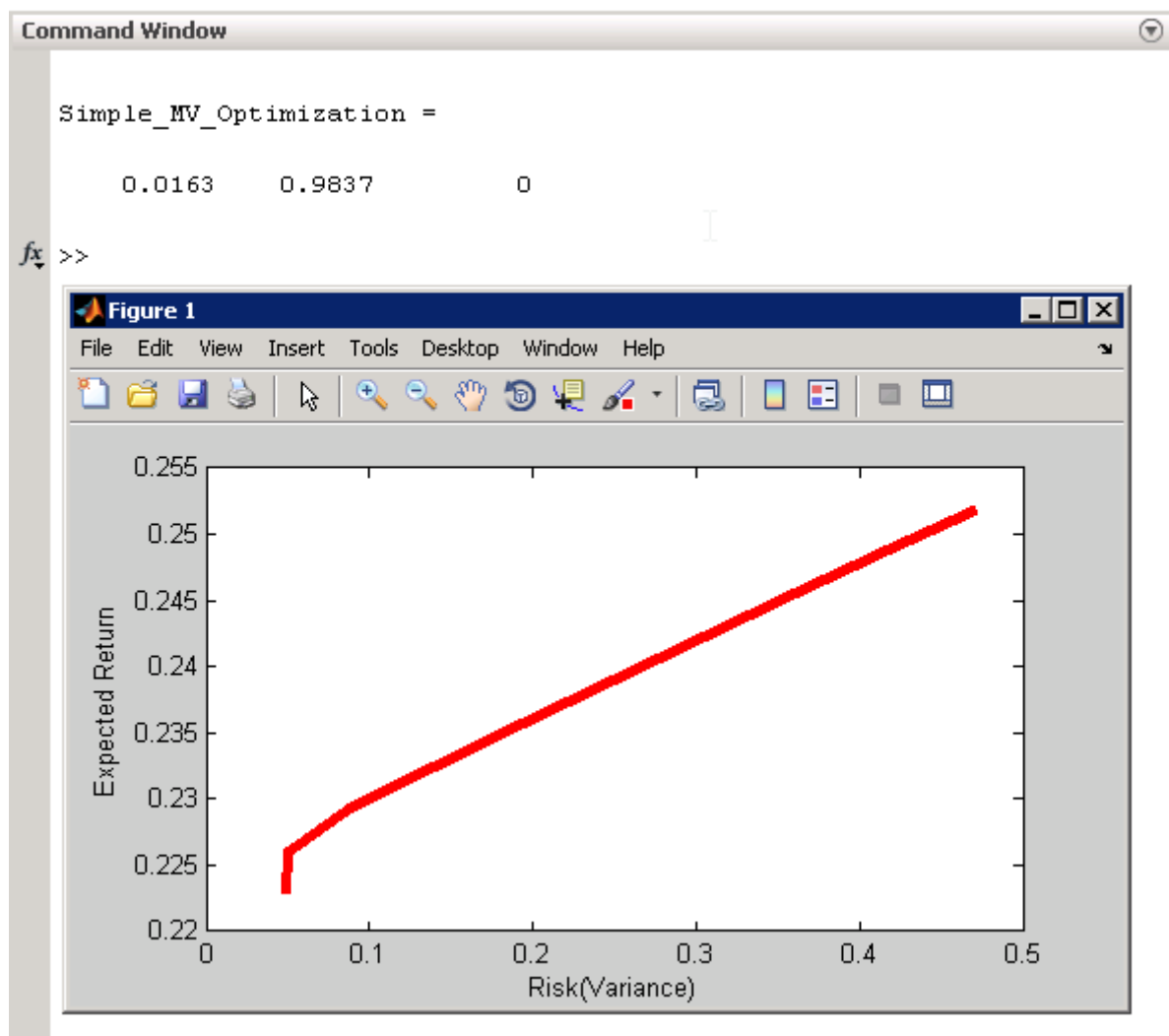
```

Michaud Optimal Portfolio (bootstrap/resample efficiency)

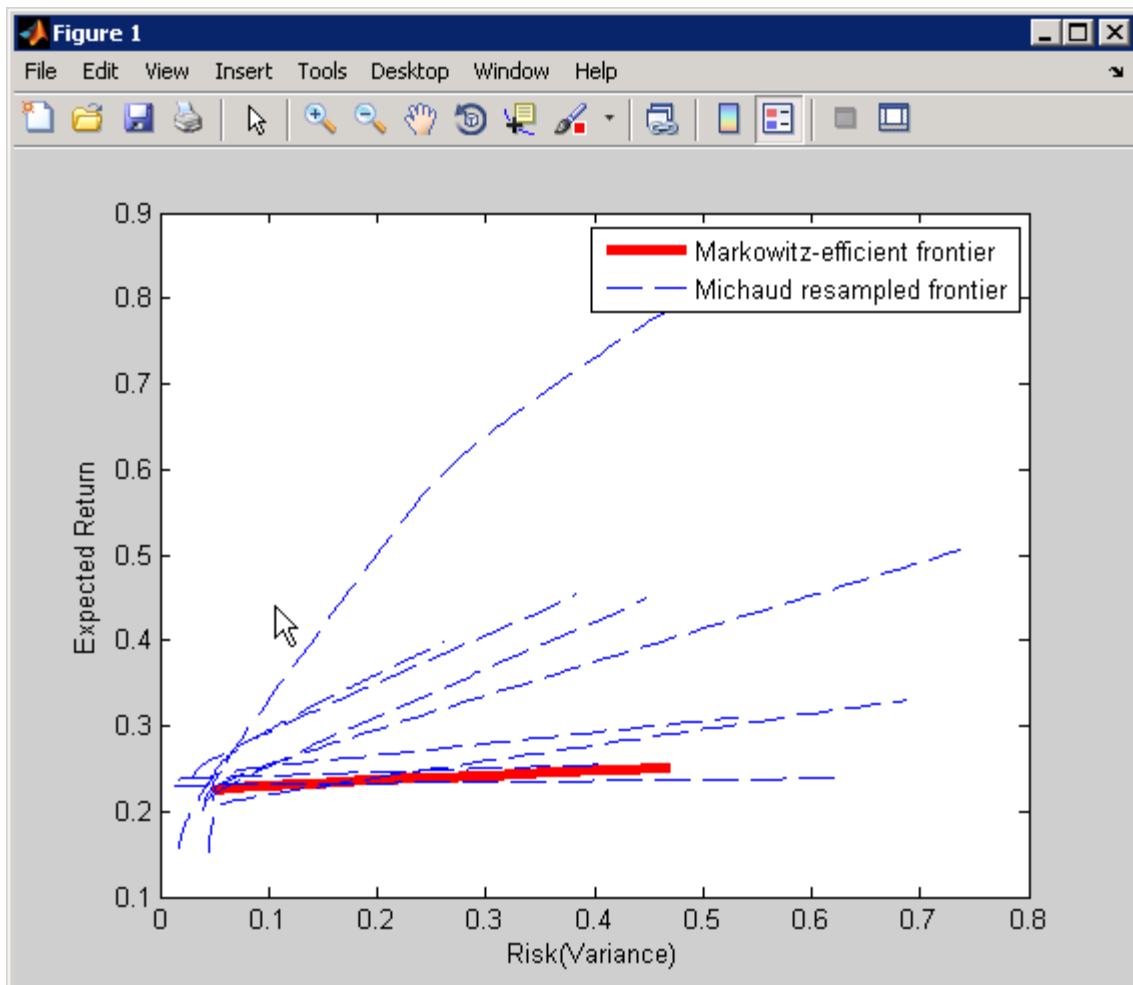
```

EDITOR PUBLISH VIEW
%Michaud first step is standard Markowitz Optimization
%unconstrained in this case
clear all;close all;clc
T1=[-0.15,0.05,-0.43,0.79,0.32];
T2=[0.29,0.18,0.24,0.25,0.17];
T3=[0.38,0.63,0.46,0.36,-0.57];
listed_Asset = {'T1','T2','T3'};
Data=[T1' T2' T3'];
expReturns =mean(Data);
m_Cov = cov(Data);
stdDev = std(Data);
[nrow,NumAssets]= size(Data);
NumSim = 10; % Number of samples
%Markowitz Optimization
[PortRisk, PortReturn, PortWts] = portopt(expReturns,m_Cov,NumSim);
plot(PortRisk,PortReturn,'r','linewidth',4);
ylabel('Expected Return');xlabel('Risk(Variance)')
[RiskyRisk, RiskyReturn, RiskyWts, ~,~]...
    = portalloc(PortRisk, PortReturn, PortWts,0.001,0.001);
Simple_MV_Optimization = RiskyWts;
Simple_MV_Optimization

```



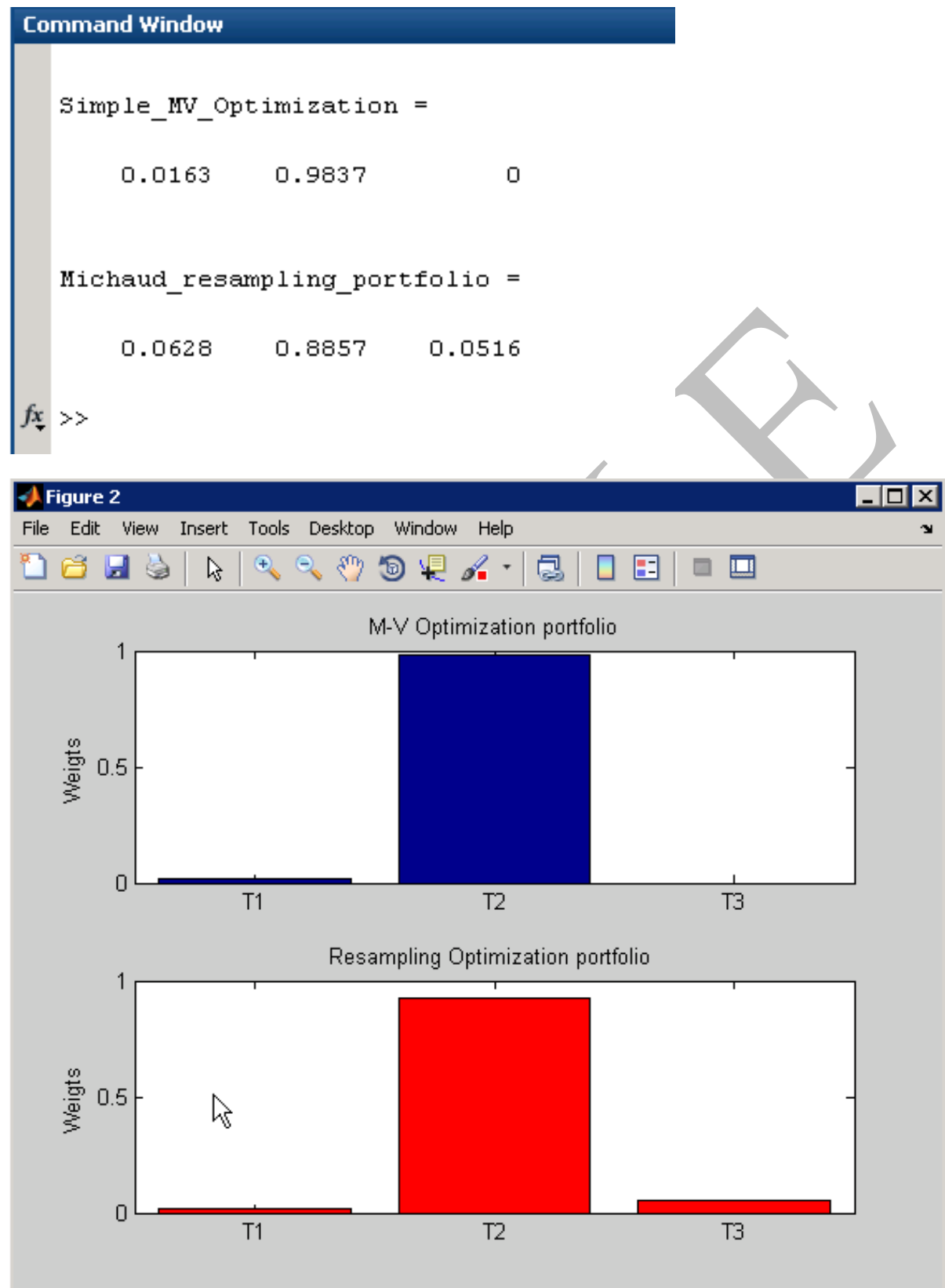
```
22  
23 %Second step is just a resampling routine (the core of Michaud model)  
24 %to manage the estimate of covariance matrix and all other stuff  
25 %if the dataset is too small the script could stop...  
26 i=1;  
27 hold on;  
28 for j=1:NumSim  
29     %Resample from the past returns with normal distribution  
30     returnsim=ones(nrow,1)*expReturns+randn(nrow,NumAssets).*(ones(nrow,1)*stdDev);  
31     meanSim=mean(returnsim);% sample mean  
32     covSim=cov(returnsim);% sample covariance  
33     [PortRisk, PortReturn, PortWts] = portopt(meanSim,covSim,100);  
34     plot(PortRisk,PortReturn,'--');% plot (i)th sample frontier  
35     [RiskyRisk, RiskyReturn, RiskyWts, ~,~]...  
36     = portalloc(PortRisk, PortReturn, PortWts, .02,0.02);  
37     weight(:,i)= RiskyWts;% get (i)th sample optimisation portfolio  
38     i = i + 1;  
39 end  
40 legend('Markowitz-efficient frontier','Michaud resampled frontier')  
41 hold off
```



```

42
43 - weight = weight';
44 - Michaud_resampling_portfolio = mean(weight);
45 - Michaud_resampling_portfolio
46
47 %resume all as bar chart
48 - figure
49 - subplot(2,1,1)
50 - bar(Simple_MV_Optimization)
51 - ylabel('Weights');
52 - set(gca,'XTickLabel',listed_Asset)
53 - title('M-V Optimization portfolio');
54 - subplot(2,1,2)
55 - bar(Michaud_resampling_portfolio,'r')
56 - ylabel('Weights')
57 - set(gca,'XTickLabel',listed_Asset)
58 - title('Resampling Optimization portfolio ')

```



CVaR portfolio

```

EDITOR PUBLISH VIEW
cVaR.m x
1 function result = cVaR(w,data,alpha)
2 % cVaR calculation for scenario returns
3 % w is in (n x 1) vector and r is a (T x n) vector, where
4 % n is the number of assets and T the number of simulations
5 [T,n] = size(data);
6 rp = sort(data*w); % sorted portfolio return
7 cutoff = floor(alpha*T); % cutoff (VaR)
8 result = mean(rp(1:cutoff,1));
9 end

```

```

EDITOR PUBLISH VIEW
cVaR_Optimization.m x
1 close all
2 clear all
3 clc
4 %% Loading Data
5 [data ,symb] = xlsread('Stocks.xlsx');
6 [T ,N] = size(data);
7 covAsset = cov(data);
8 muAsset = mean(data); % This definition is again arbitrary to get some reasonable values
9 % Plotting the distribution of the equal weighted portfolio
10 wgtm = ones(N,1)/N; % equal weighted portfolio weights
11 figure(1)
12 prob1 = 0.01;% Alpha
13 CVaR1 = cVaR(wgtm,data,prob1);
14 prob2 = 0.05; % Alpha
15 CVaR2 = cVaR(wgtm,data,prob2);
16 hist1 = hist(data*wgtm,T/10);
17 hist(data*wgtm,T/10)
18 set(gca,'Box','On','LineWidth',1.5,'FontSize',12)
19 hold on
20 plot([CVaR1 CVaR1],[0 max(hist1)],'r','LineWidth',1.5)
21 plot([CVaR2 CVaR2],[0 max(hist1)],'r--','LineWidth',1.5)
22 legend('returns',['CVar-Level, \alpha = ' num2str(prob1)],['CVar-Level, \alpha = ' num2str(prob2)])
23
24 %% Optimization
25 % first, specify the target returns:
26 Rs = linspace(max(min(muAsset),0), max(muAsset),15);
27 % then the confidence level alpha
28 alpha=0.005;
29 % set the lower und upper bounds
30 UB= 1;
31 LB = -UB;
32 Short=0; %If Short=1, we allow for short positions. If short = 0, we don't.
33 % here comes the optimization loop:
34 clear ES1 R

```

```

35 % we want to minimize -CVaR.
36 - f=@(w) -(cVaR(w,data,alpha));
37 % and the Markowitz problem:
38 - meanvar=@(w) w'*covAsset*w;
39 %Initial guess:
40 - w0 = wgtm;
41 - ESO = cVaR(w0,data,alpha);
42 - for i = 1:length(Rs)
43     R0= Rs(i); % target return i
44     A= - mean(data);
45     if Short==0
46         A=[A;-eye(N)];
47         A=[A; eye(N)];
48         b=[-R0 zeros(1,N) UB*ones(1,N)];
49     elseif Short==1
50         A=[A;-eye(N)];
51         A=[A; eye(N)]; %ok<*AGROW>
52         b=[-R0 -LB*ones(1,N) UB*ones(1,N)];
53     elseif Short~=0||Short~=1
54         error('Set Short=1 or 0')
55     end
56     b=b';
57     Aeq= ones(1,N);
58     beq= 1;
59     options = optimset(optimset('fmincon'),'Display','off','Algorithm','sqp');
60     % CVaR optimization
61     [w,~,~,~]=fmincon(f,w0,A,b,Aeq,beq,[],[],[],options);
62     % Mean-Var optimization
63     [wm,~,~,~]=fmincon(meanvar,w0,A,b,Aeq,beq,[],[],[],options);
64     CVaR(i) = -cVaR(w,data,alpha);
65     CVaRwm(i) = -cVaR(wm,data,alpha);
66     SD(i) = std(data*w);
67     R(i) = mean(data*w);

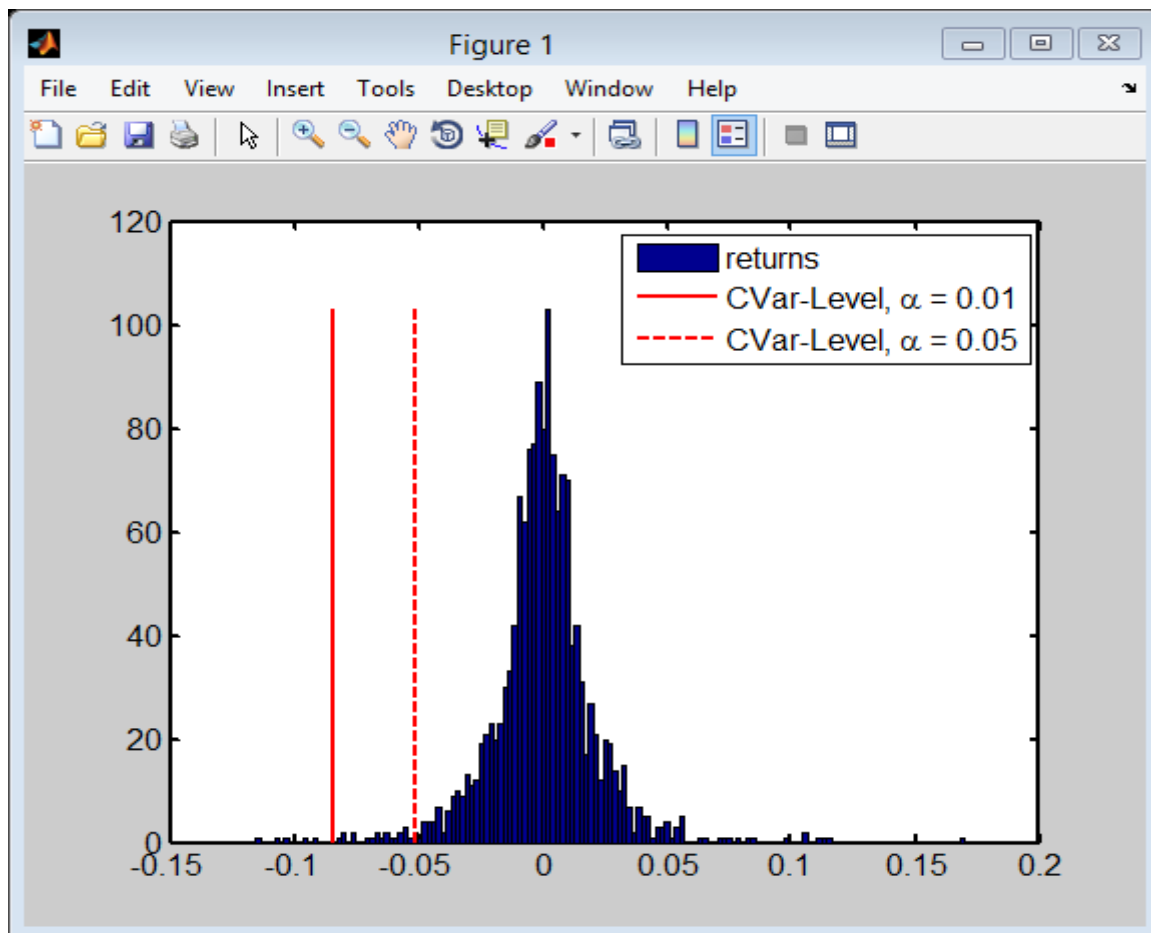
```

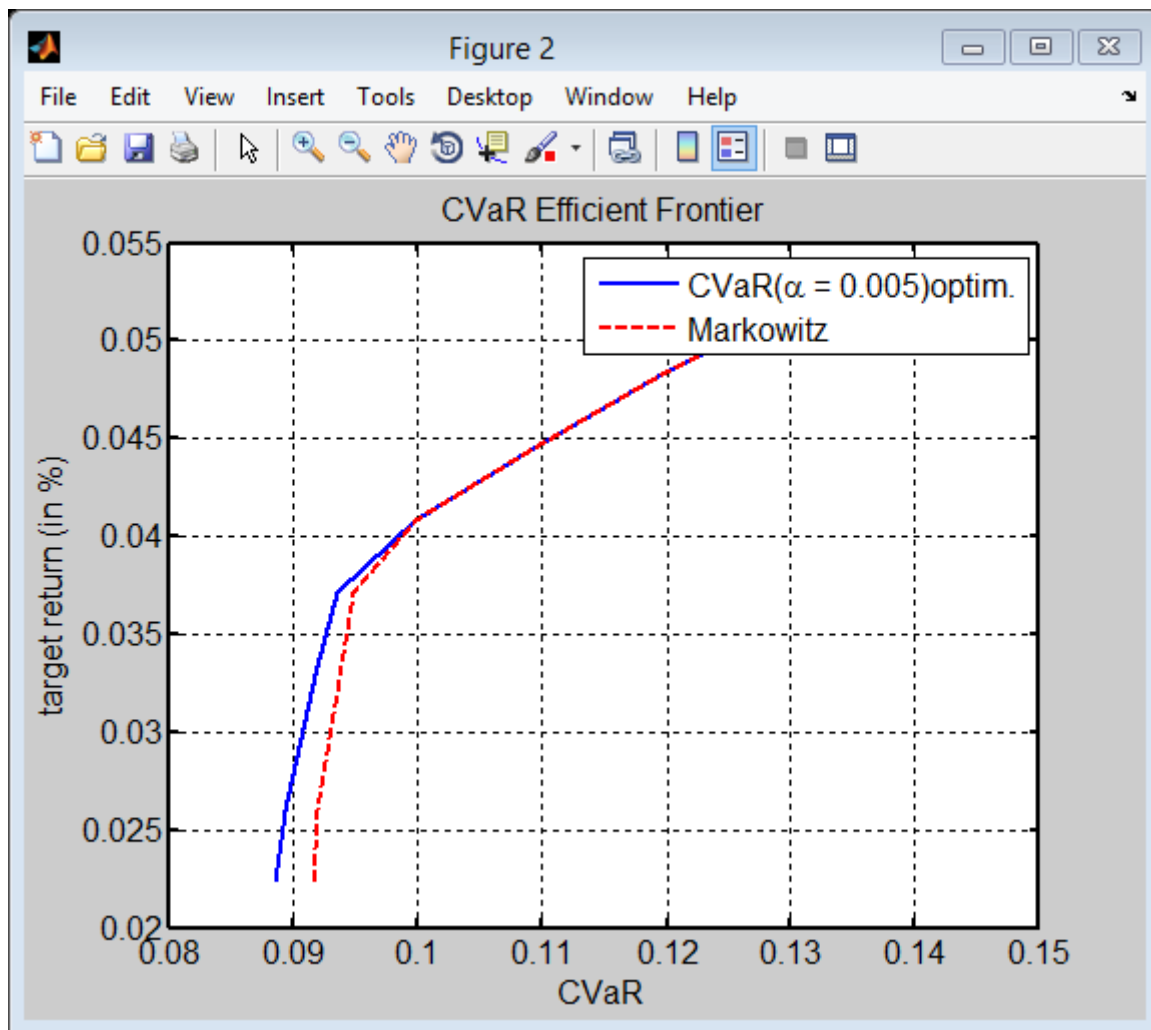


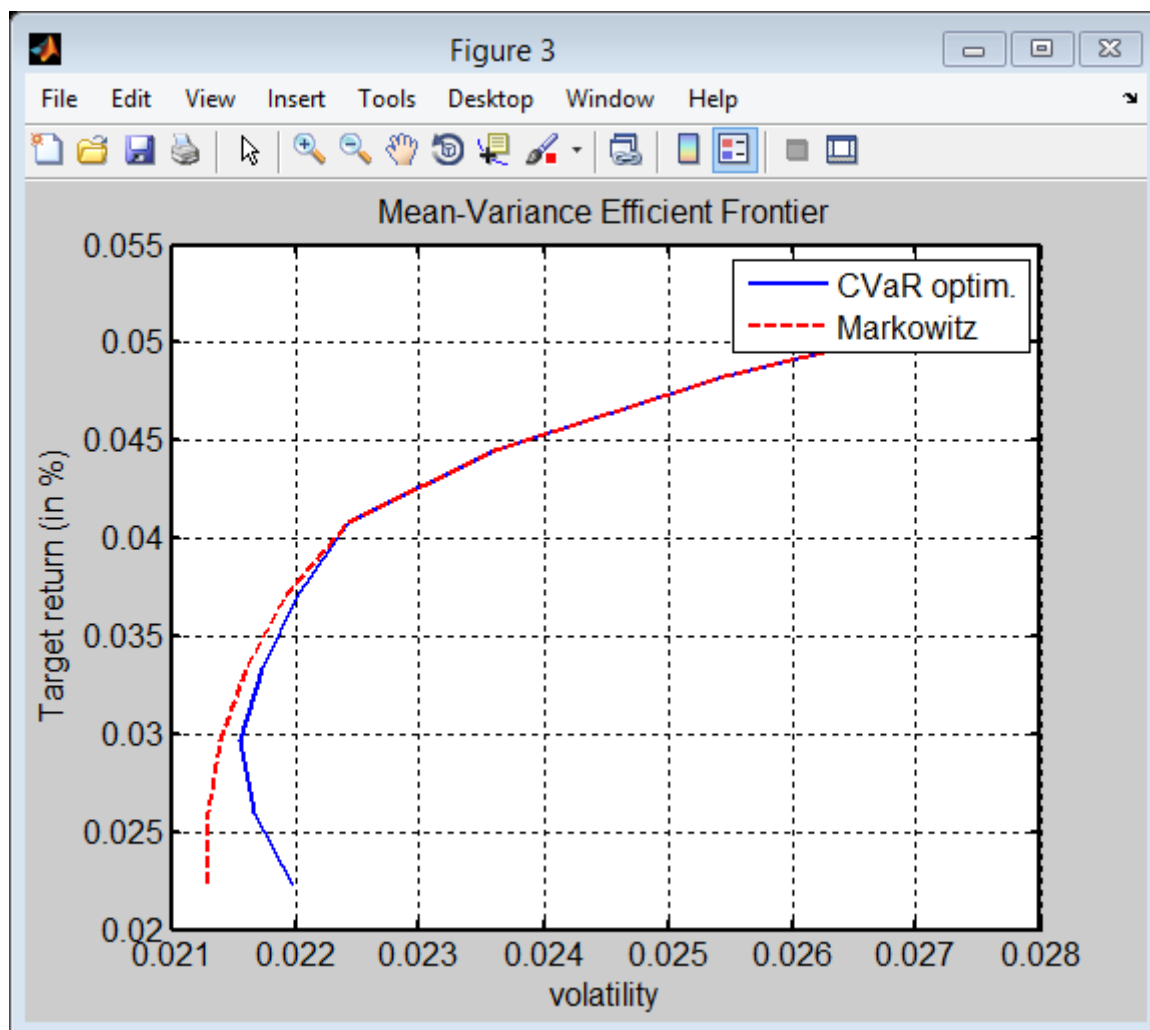
```

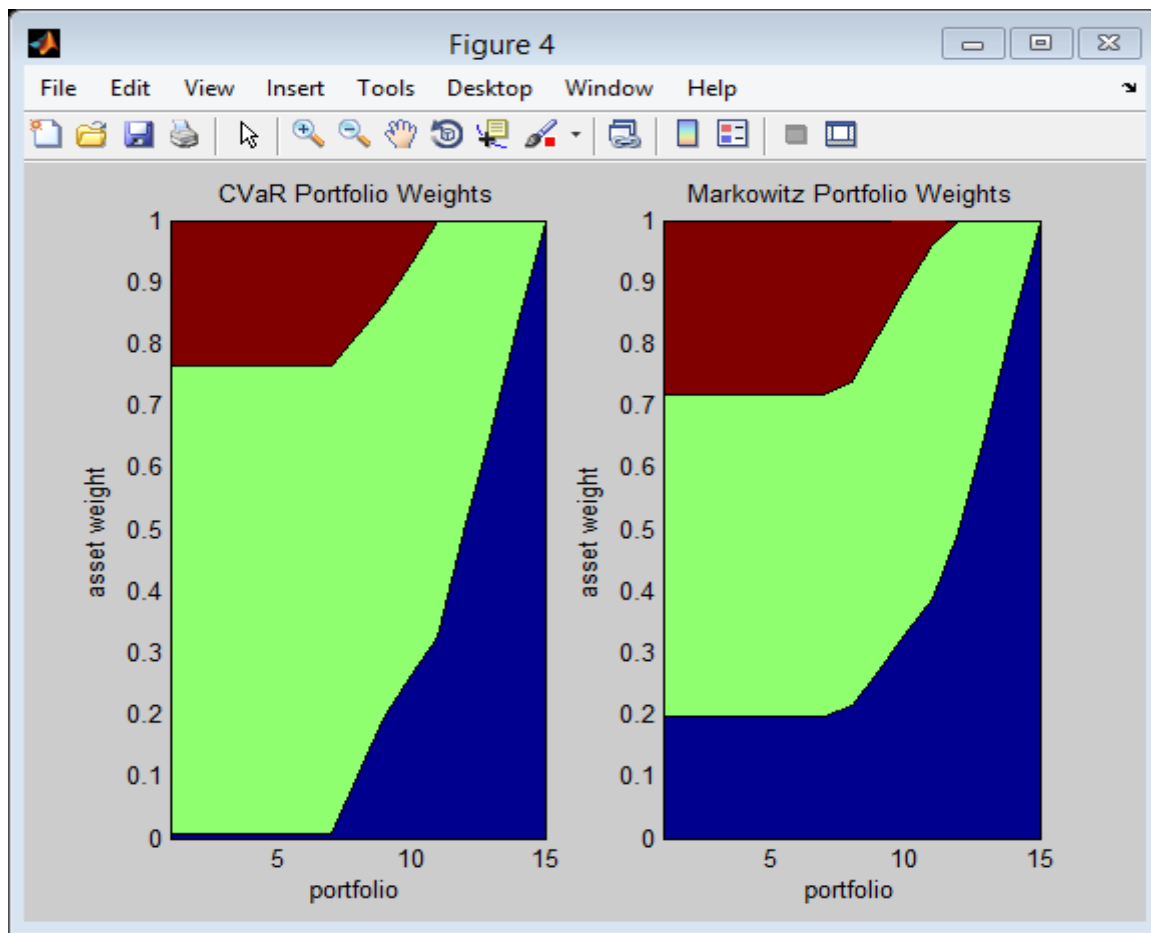
68 -     effSig(i) = std(data*wm);
69 -     effMu(i) = mean(data*w) ;
70 -     Weight(:,i) = w;
71 -     MarkoW(:,i) = wm;
72 - end
73
74 % Plot of the CVaR efficient frontier
75 - figure(2)
76 - hold on
77 - plot(CVaR,R*100,'LineWidth',1.5)
78 - plot(CVaRwm,effMu*100,'r--','LineWidth',1.5)
79 - set(gcf,'Box','on','LineWidth', 1.5, 'FontSize',12')
80 - grid on
81 - ylabel('target return (in %)')
82 - title('CVaR Efficient Frontier')
83 - legend(['CVaR(\alpha = ' num2str(alpha) ')optim.'],'Markowitz')
84 - xlabel('CVaR')
85
86 % Plot of the Mean-Variance efficient frontier
87 - figure(3)
88 - plot(SD,R*100,'LineWidth',1.5)
89 - hold on
90 - plot(effSig,effMu*100,'r--','LineWidth',1.5)
91 - set(gcf,'Box','on','LineWidth', 1.5, 'FontSize',12')
92 - grid on
93 - ylabel('Target return (in %)')
94 - legend('CVaR optim.','Markowitz')
95 - title('Mean-Variance Efficient Frontier')
96 - xlabel('volatility')
97
98 % Plot of the portfolio weights
99 - figure(4)
100 - subplot(1,2,1)
101 - area(Weight);
102 - title('CVaR Portfolio Weights');
103 - set(get(gcf,'Children'),'YLim',[0 1]);
104 - xlabel('portfolio')
105 - ylabel('asset weight')
106
107 - subplot(1,2,2)
108 - area(MarkoW);
109 - title('Markowitz Portfolio Weights');
110 - set(get(gcf,'Children'),'YLim',[0 1]);
111 - xlabel('portfolio')
112 - ylabel('asset weight')
113

```









Black-Littermann portfolio

```

EDITOR PUBLISH VIEW
function [er, ps, w, pw, lambda, theta] = ...
1   Black_litterman(delta, weq, sigma, tau, P, Q, Omega) %#codegen
2   %http://www.blacklitterman.org/Black-Litterman.pdf
3   %Copyright (c) Jay Walters, blacklitterman.org, 2008.
4
5
6   %Input:
7   %delta - Risk tolerance from the equilibrium portfolio
8   %weq - Weights of the assets in the equilibrium portfolio
9   %sigma - Prior covariance matrix
10  %tau - Coefficient of uncertainty in the prior estimate of the mean (pi)
11  %P - Pick matrix for the view(s)
12  %Q - Vector of view returns
13  %Omega - Matrix of variance of the views (diagonal)
14
15  %Outputs
16  %Er - Posterior estimate of the mean returns
17  %w - Unconstrained weights computed given the posterior estimates
18  %of the mean and covariance of returns.
19  %lambda - A measure of the impact of each view on the posterior estimates.
20  %theta - A measure of the share of the prior and sample information
21  %in the posterior precision.
22
23  %Reverse optimize and back out the equilibrium returns
24  pi = weq * sigma * delta;
25  %We use tau * sigma many places so just compute it once
26  ts = tau * sigma;
27  %Compute posterior estimate of the mean
28  er = pi' + ts * P' * inv(P * ts * P' + Omega) * (Q - P * pi');
29  % We can also do it the long way to illustrate that d1 + d2 = I

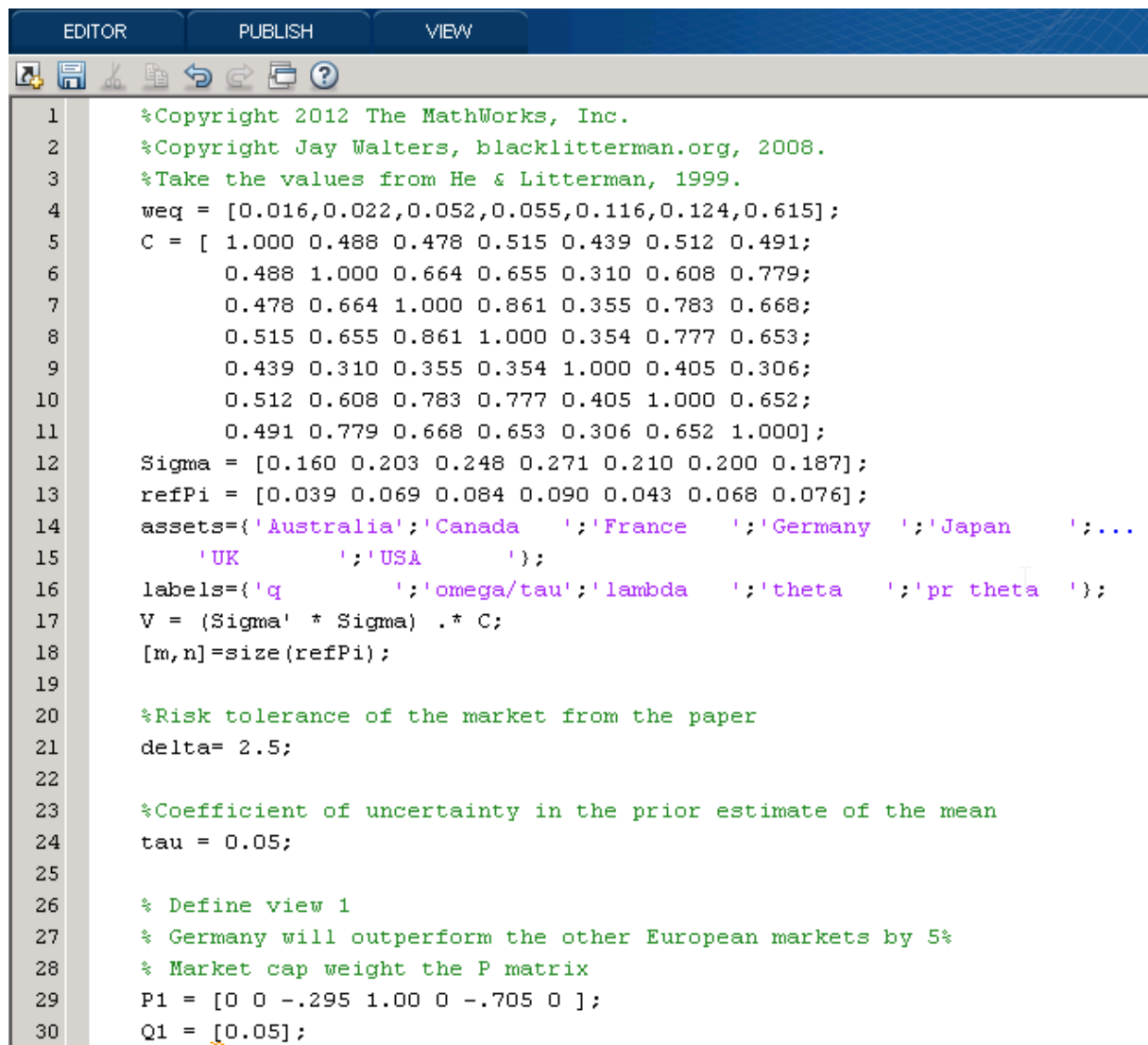
```

```

30 - d = inv(inv(ts) + P' * inv(Omega) * P);
31 - d1 = d * inv(ts);
32 - d2 = d * P' * inv(Omega) * P;
33 - er2 = d1 * pi' + d2 * pinv(P) * Q;
34 - %Compute posterior estimate of the uncertainty in the mean
35 - ps = ts - ts * P' * inv(P * ts * P' + Omega) * P * ts;
36 - posteriorSigma = sigma + ps;
37 - %Compute the share of the posterior precision from prior and views,
38 - %then for each individual view so we can compare it with lambda
39 - theta=zeros(1,2+size(P,1));
40 - theta(1,1) = (trace(inv(ts) * ps) / size(ts,1));
41 - theta(1,2) = (trace(P' * inv(Omega) * P * ps) / size(ts,1));
42 - for i=1:size(P,1)
43 -     theta(1,2+i) = (trace(P(i,:) * inv(Omega(i,i)) * P(i,:) * ps) / size(ts,1));
44 - end
45 - %Compute posterior weights based solely on changed covariance
46 - w = (er' * inv(delta * posteriorSigma))';
47 - % Compute posterior weights based on uncertainty in mean and covariance
48 - pw = (pi * inv(delta * posteriorSigma))';
49 - %Compute lambda value
50 - %We solve for lambda from formula (17) page 7, rather than formula (18)
51 - %just because it is less to type, and we've already computed w*.
52 - lambda = pinv(P)' * (w' * (1+tau) - weq)';
53 - end

```

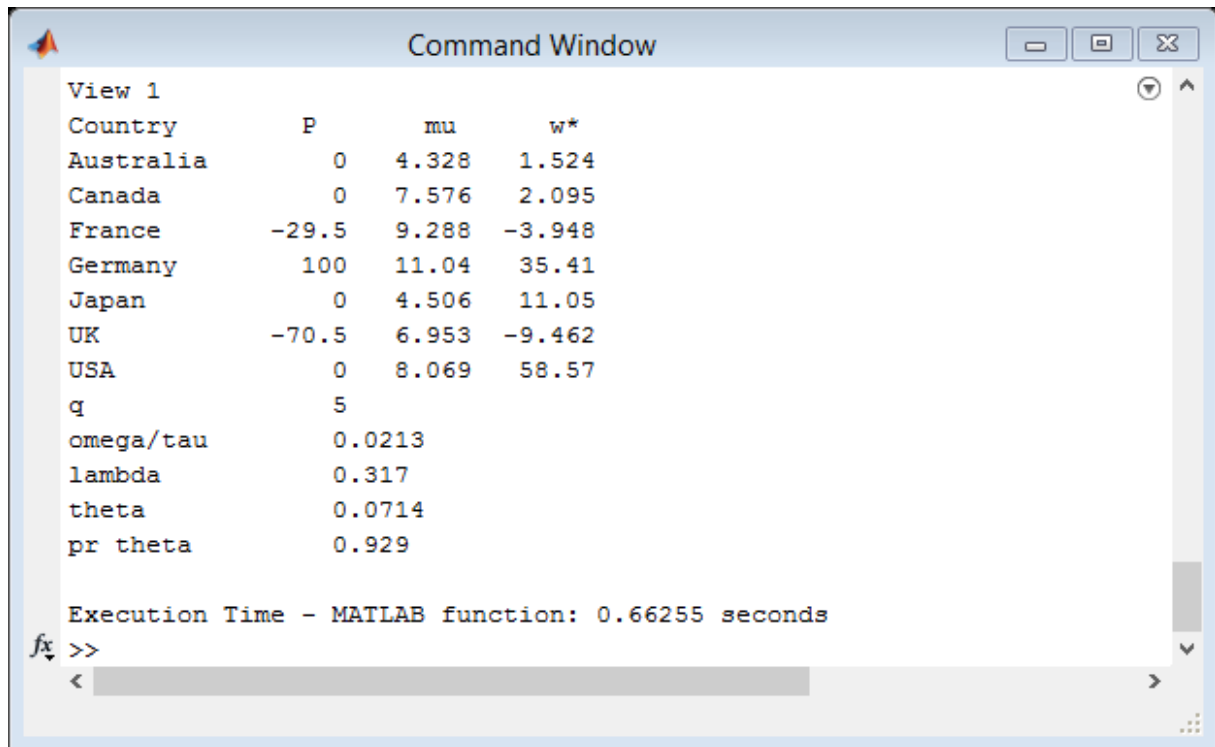
Now we test the script with the values of the article of BL:



```

1  %Copyright 2012 The MathWorks, Inc.
2  %Copyright Jay Walters, blacklitterman.org, 2008.
3  %Take the values from He & Litterman, 1999.
4  weq = [0.016,0.022,0.052,0.055,0.116,0.124,0.615];
5  C = [ 1.000 0.488 0.478 0.515 0.439 0.512 0.491;
6        0.488 1.000 0.664 0.655 0.310 0.608 0.779;
7        0.478 0.664 1.000 0.861 0.355 0.783 0.668;
8        0.515 0.655 0.861 1.000 0.354 0.777 0.653;
9        0.439 0.310 0.355 0.354 1.000 0.405 0.306;
10       0.512 0.608 0.783 0.777 0.405 1.000 0.652;
11       0.491 0.779 0.668 0.653 0.306 0.652 1.000];
12  Sigma = [0.160 0.203 0.248 0.271 0.210 0.200 0.187];
13  refPi = [0.039 0.069 0.084 0.090 0.043 0.068 0.076];
14  assets={'Australia';'Canada';'France';'Germany';'Japan';...
15         'UK';'USA'};
16  labels={'q';'omega/tau';'lambda';'theta';'pr theta'};
17  V = (Sigma' * Sigma) .* C;
18  [m,n]=size(refPi);
19
20  %Risk tolerance of the market from the paper
21  delta= 2.5;
22
23  %Coefficient of uncertainty in the prior estimate of the mean
24  tau = 0.05;
25
26  % Define view 1
27  % Germany will outperform the other European markets by 5%
28  % Market cap weight the P matrix
29  P1 = [0 0 -.295 1.00 0 -.705 0 ];
30  Q1 = [0.05];

```

Command Window

```
View 1
Country      P      mu      w*
Australia    0      4.328    1.524
Canada       0      7.576    2.095
France       -29.5   9.288   -3.948
Germany      100    11.04    35.41
Japan        0      4.506    11.05
UK           -70.5   6.953   -9.462
USA          0      8.069    58.57
q            5
omega/tau    0.0213
lambda       0.317
theta        0.0714
pr theta     0.929

Execution Time - MATLAB function: 0.66255 seconds
fx >>
< >
```

A screenshot of the MATLAB Command Window. The window title is 'Command Window'. It displays a table with four columns: 'Country', 'P', 'mu', and 'w*'. The rows list countries: Australia, Canada, France, Germany, Japan, UK, and USA, along with their corresponding values for P, mu, and w*. Below the table, there are several parameters: q, omega/tau, lambda, theta, and pr theta, each with a numerical value. The execution time is shown as 'Execution Time - MATLAB function: 0.66255 seconds'. At the bottom, there is a prompt 'fx >>' and a horizontal scrollbar.

Country	P	mu	w*
Australia	0	4.328	1.524
Canada	0	7.576	2.095
France	-29.5	9.288	-3.948
Germany	100	11.04	35.41
Japan	0	4.506	11.05
UK	-70.5	6.953	-9.462
USA	0	8.069	58.57

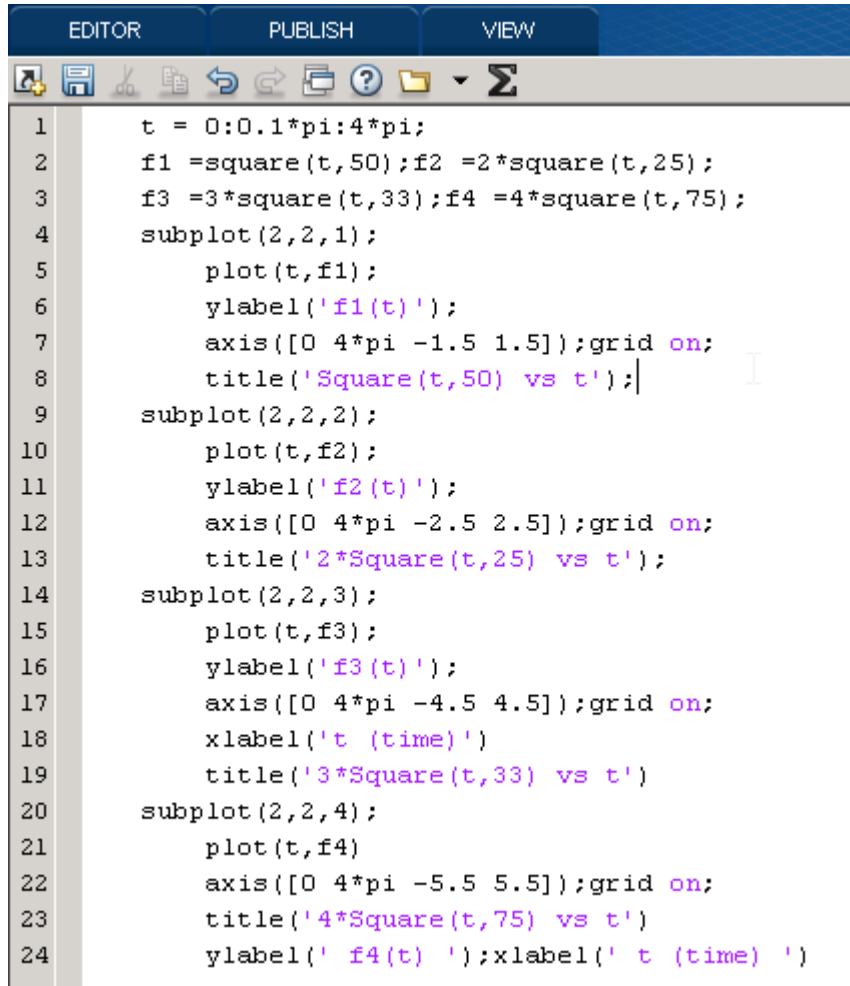
q 5
omega/tau 0.0213
lambda 0.317
theta 0.0714
pr theta 0.929

Execution Time - MATLAB function: 0.66255 seconds

fx >>
< >

Signal processing (Signal processing Toolbox)

We already saw how to make a Dirac (page 296) and a Heaviside (page 659) function without toolbox. Let us see now how to obtain another famous signal using the **square()** function:

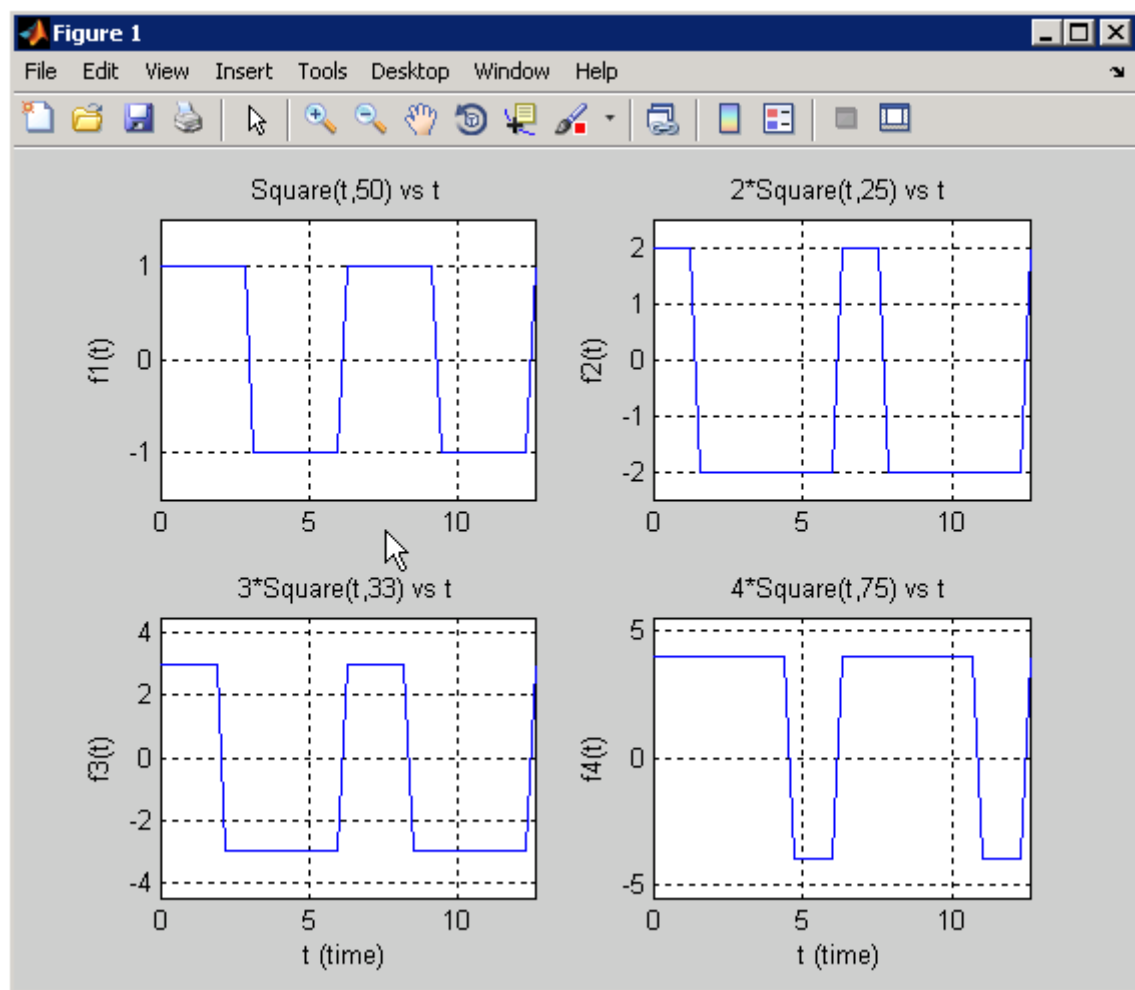


```

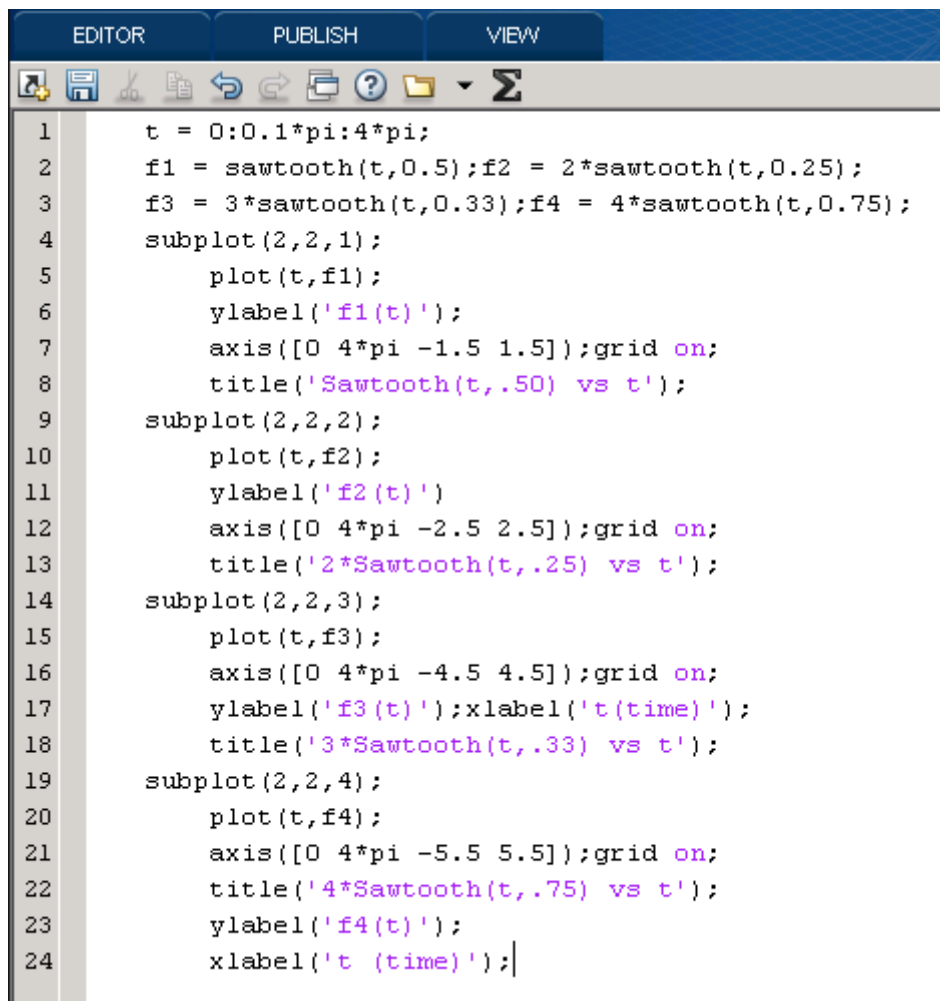
1  t = 0:0.1*pi:4*pi;
2  f1 =square(t,50);f2 =2*square(t,25);
3  f3 =3*square(t,33);f4 =4*square(t,75);
4  subplot(2,2,1);
5      plot(t,f1);
6      ylabel('f1(t)');
7      axis([0 4*pi -1.5 1.5]);grid on;
8      title('Square(t,50) vs t');
9  subplot(2,2,2);
10     plot(t,f2);
11     ylabel('f2(t)');
12     axis([0 4*pi -2.5 2.5]);grid on;
13     title('2*Square(t,25) vs t');
14 subplot(2,2,3);
15     plot(t,f3);
16     ylabel('f3(t)');
17     axis([0 4*pi -4.5 4.5]);grid on;
18     xlabel('t (time)');
19     title('3*Square(t,33) vs t');
20 subplot(2,2,4);
21     plot(t,f4);
22     axis([0 4*pi -5.5 5.5]);grid on;
23     title('4*Square(t,75) vs t');
24     ylabel('f4(t)');xlabel('t (time)')

```

So you get:

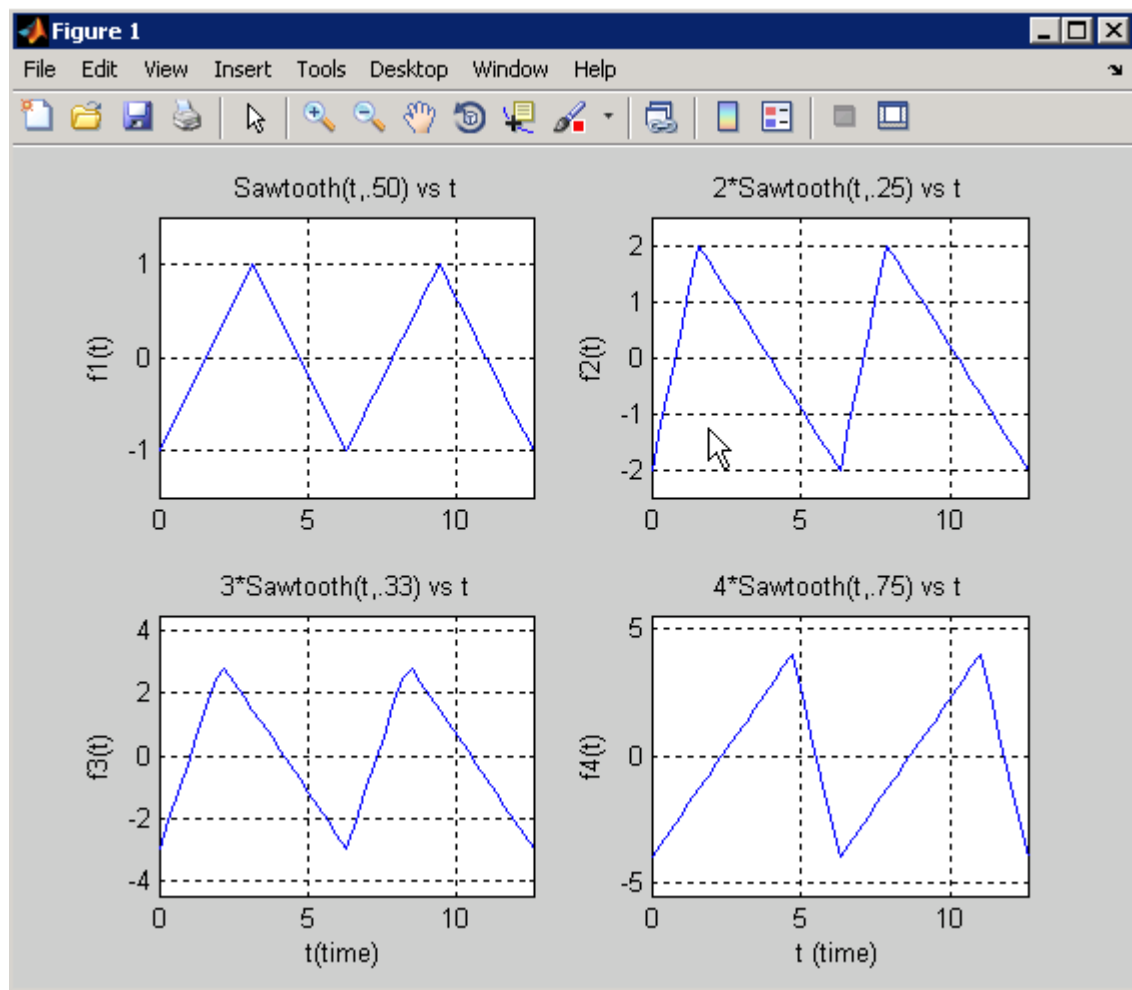


Another typical signal is the sawtooth. Here is a small example of this signal using the **sawtooth()** function:

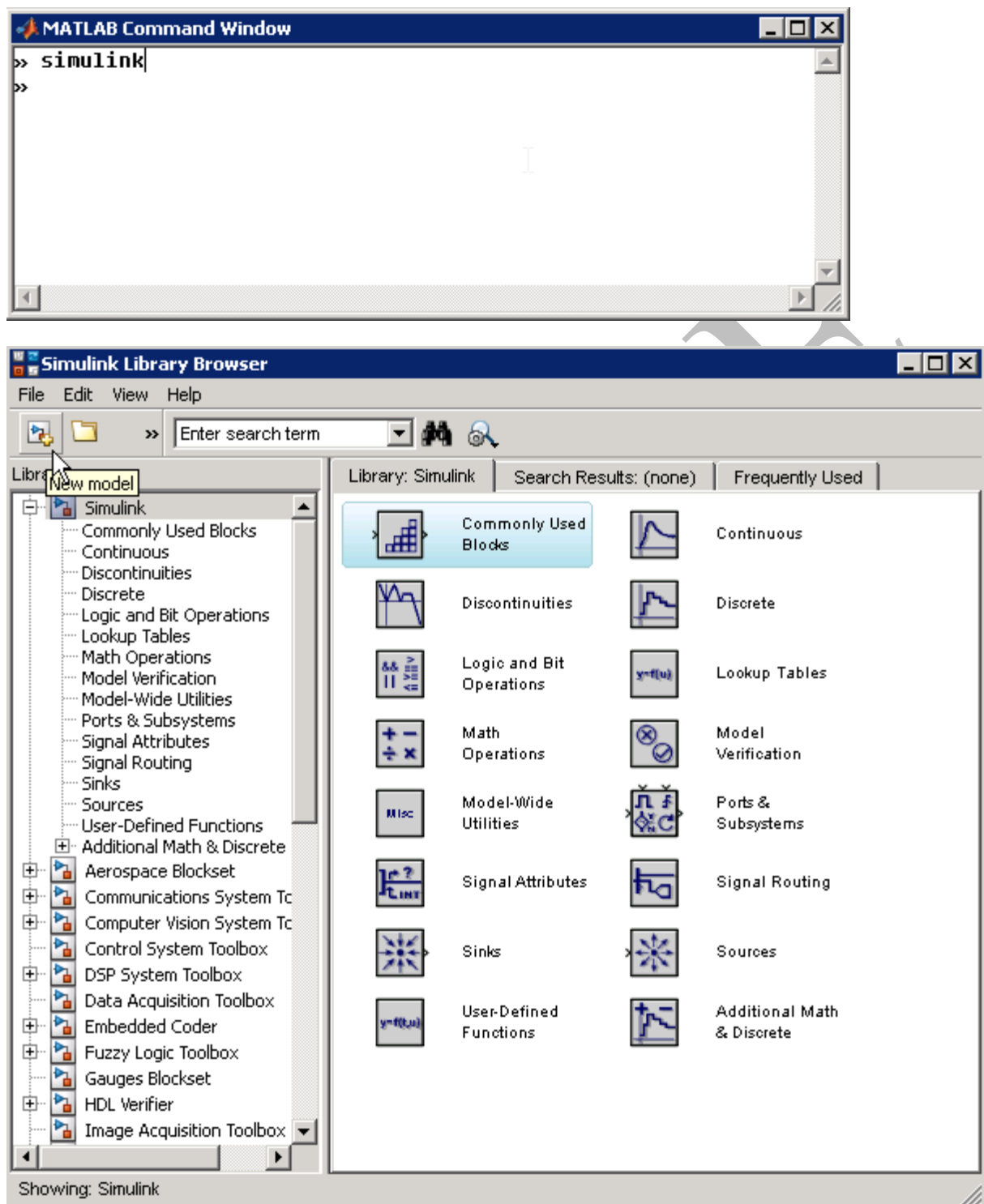


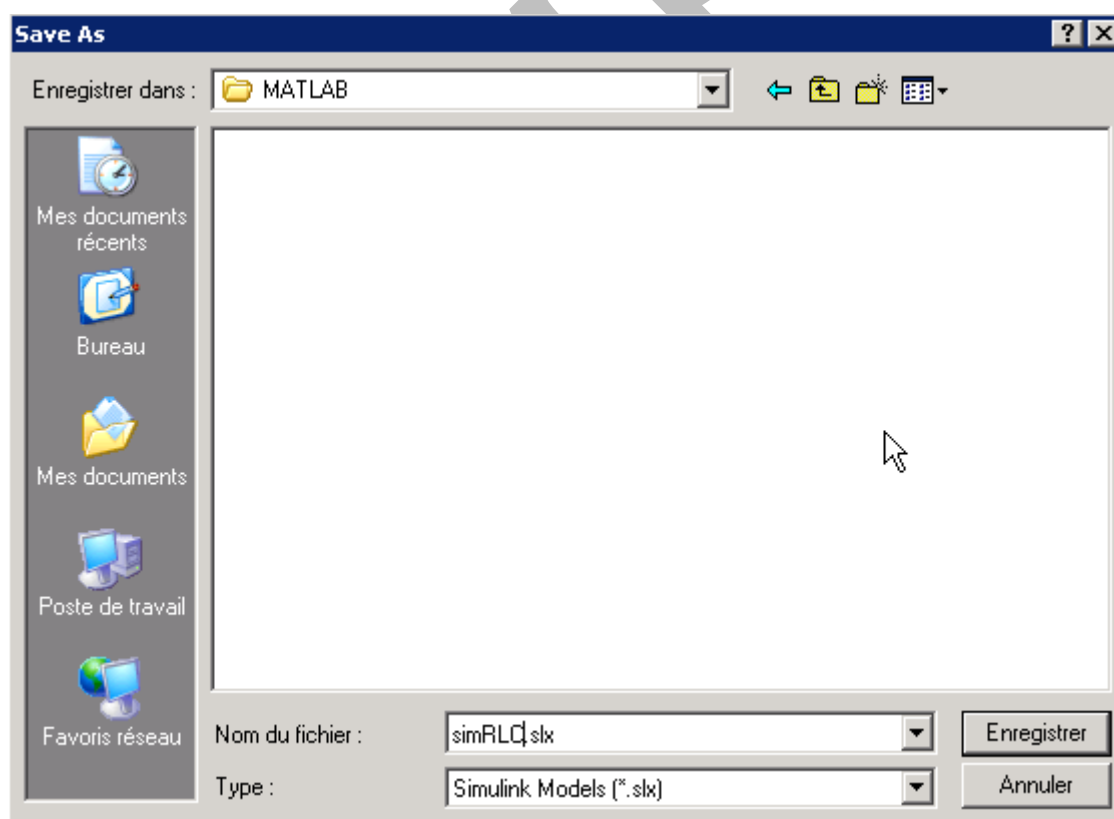
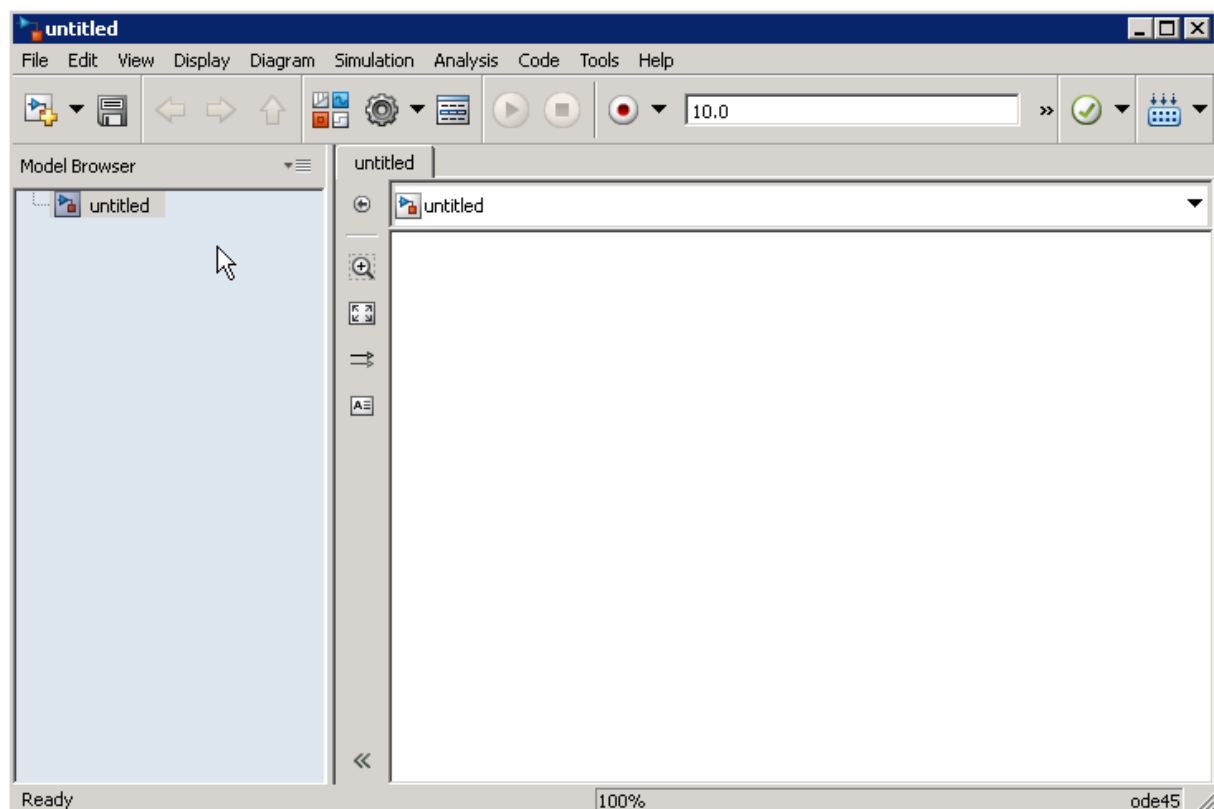
```
1      t = 0:0.1*pi:4*pi;
2      f1 = sawtooth(t,0.5);f2 = 2*sawtooth(t,0.25);
3      f3 = 3*sawtooth(t,0.33);f4 = 4*sawtooth(t,0.75);
4      subplot(2,2,1);
5          plot(t,f1);
6          ylabel('f1(t)');
7          axis([0 4*pi -1.5 1.5]);grid on;
8          title('Sawtooth(t,.50) vs t');
9      subplot(2,2,2);
10         plot(t,f2);
11         ylabel('f2(t)');
12         axis([0 4*pi -2.5 2.5]);grid on;
13         title('2*Sawtooth(t,.25) vs t');
14     subplot(2,2,3);
15         plot(t,f3);
16         axis([0 4*pi -4.5 4.5]);grid on;
17         ylabel('f3(t)');xlabel('t (time)');
18         title('3*Sawtooth(t,.33) vs t');
19     subplot(2,2,4);
20         plot(t,f4);
21         axis([0 4*pi -5.5 5.5]);grid on;
22         title('4*Sawtooth(t,.75) vs t');
23         ylabel('f4(t)');
24         xlabel('t (time)');
```

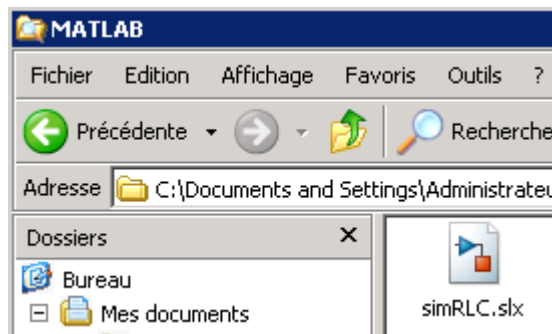
So we get:



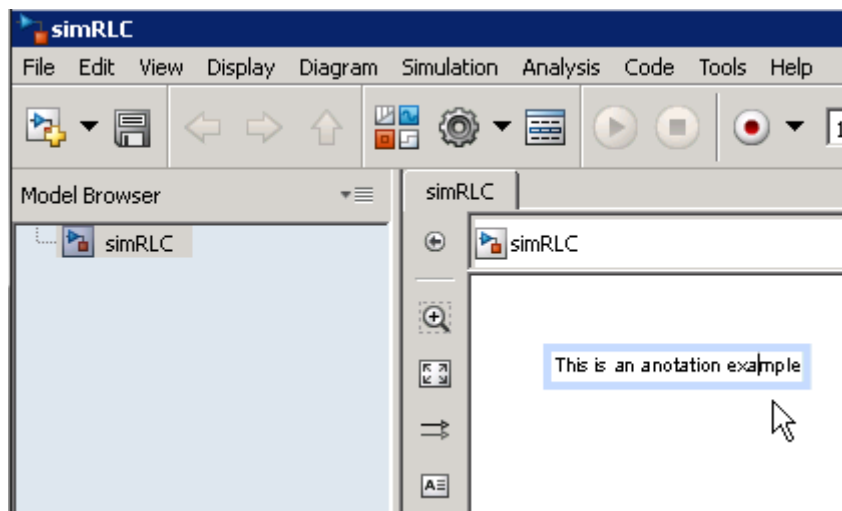
Circuit Analysis (Simulink Toolbox)



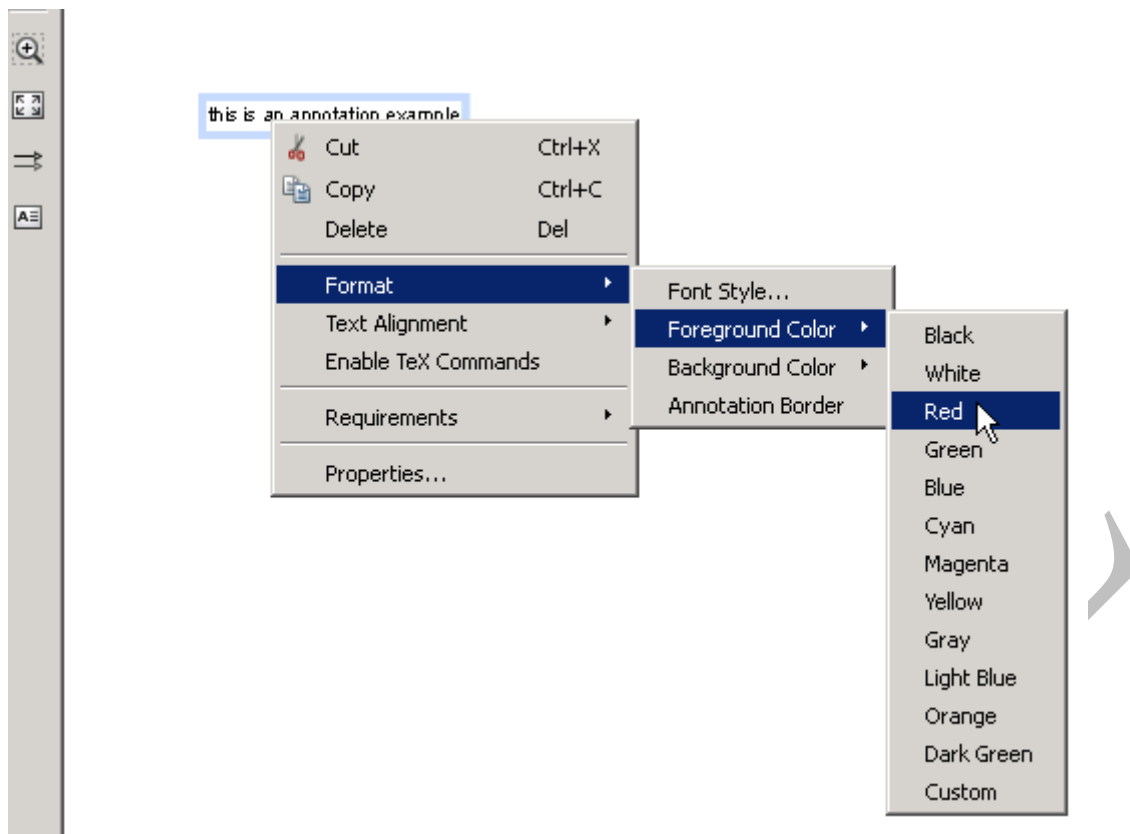




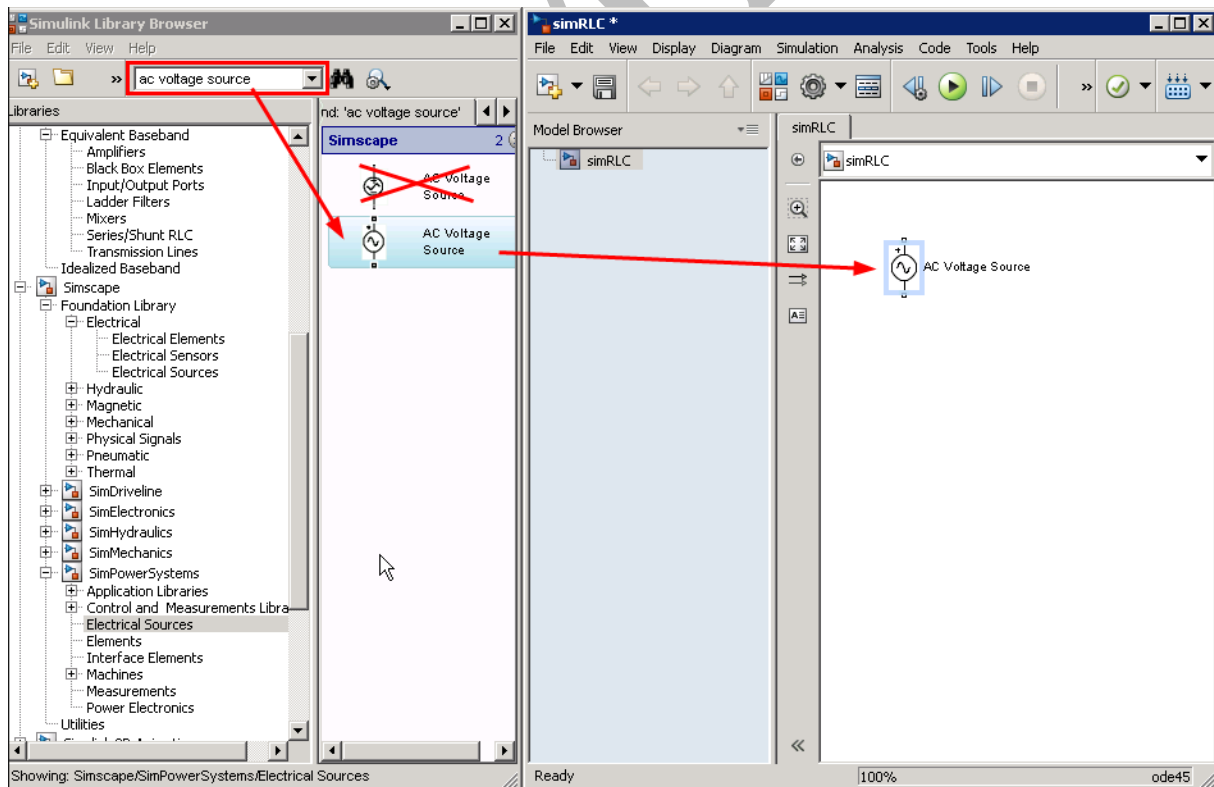
The first thing you have to know is how to annotate. For this purpose just double click anywhere in the background:

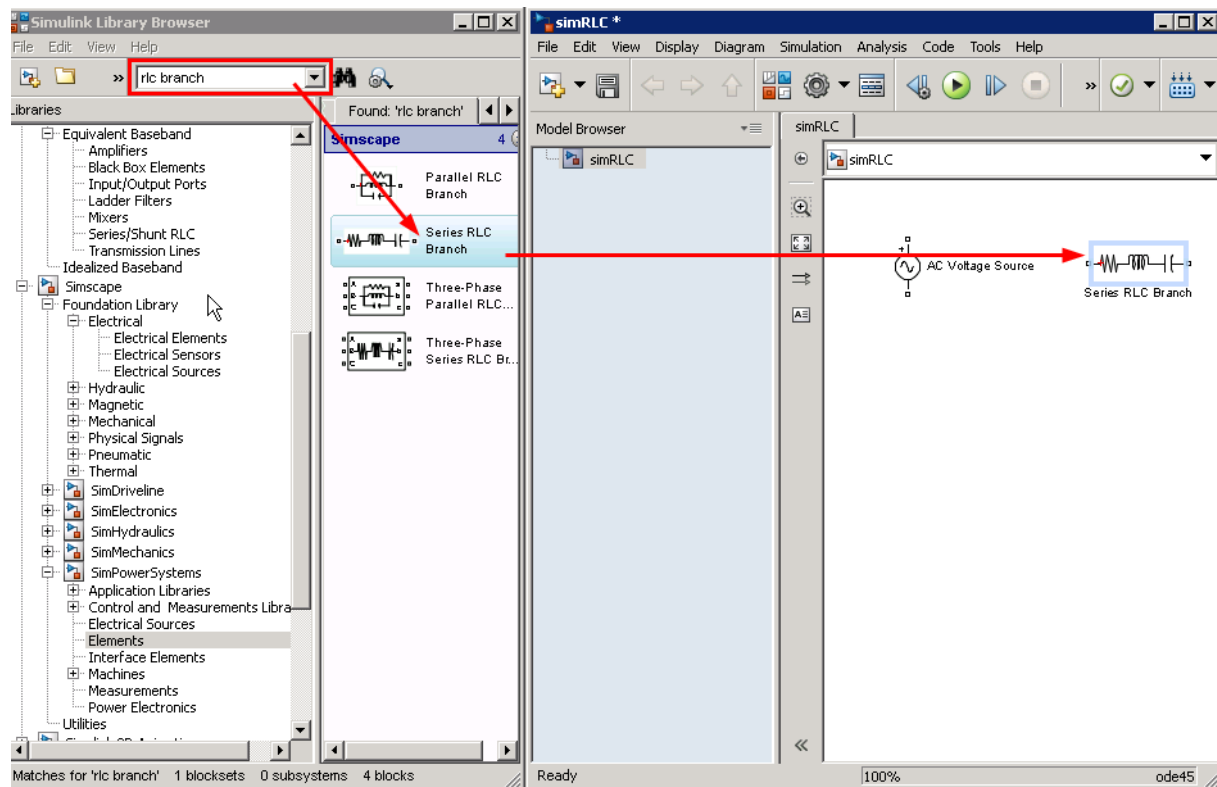


and to change the colors of the text (or of anything else):

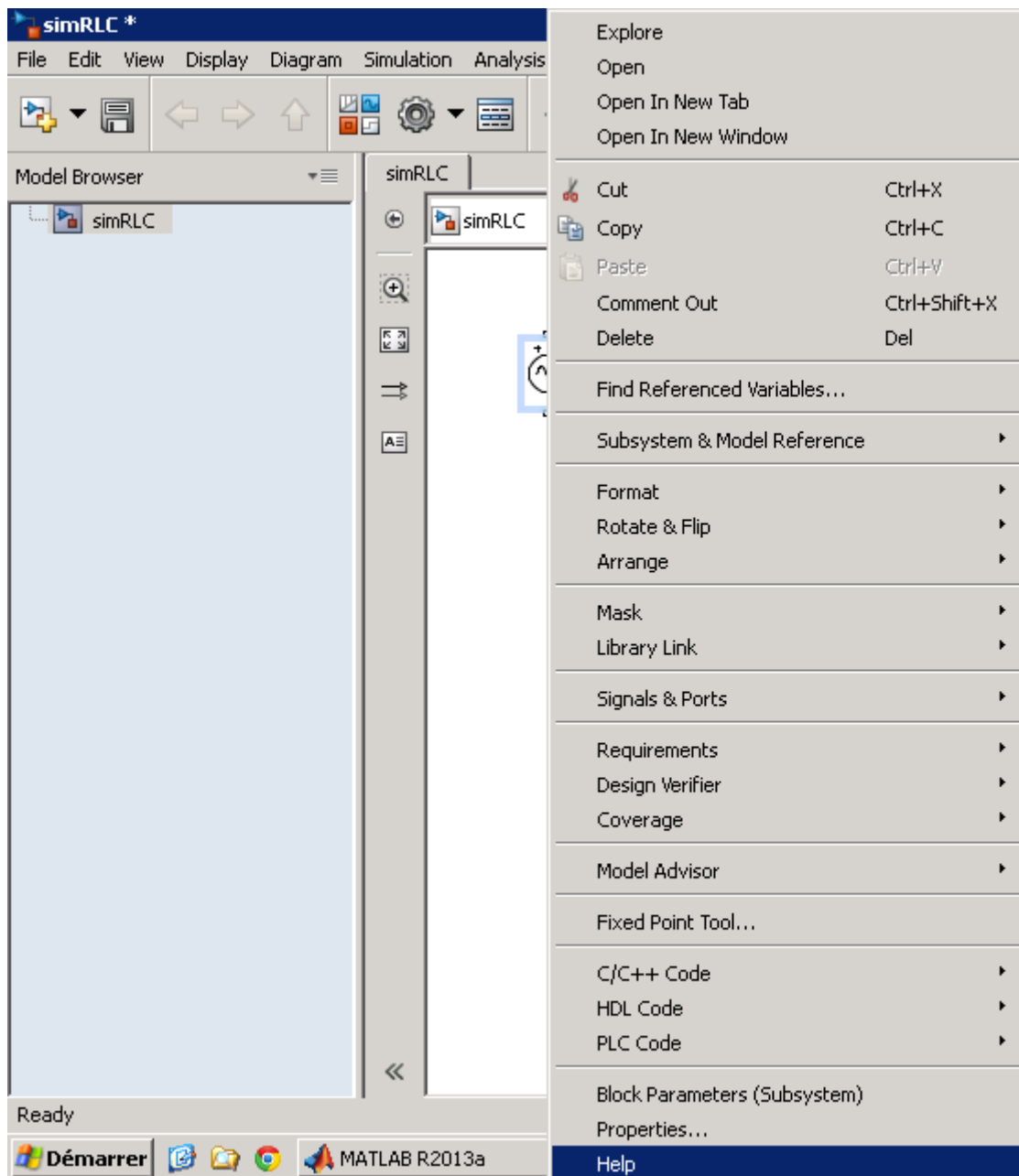


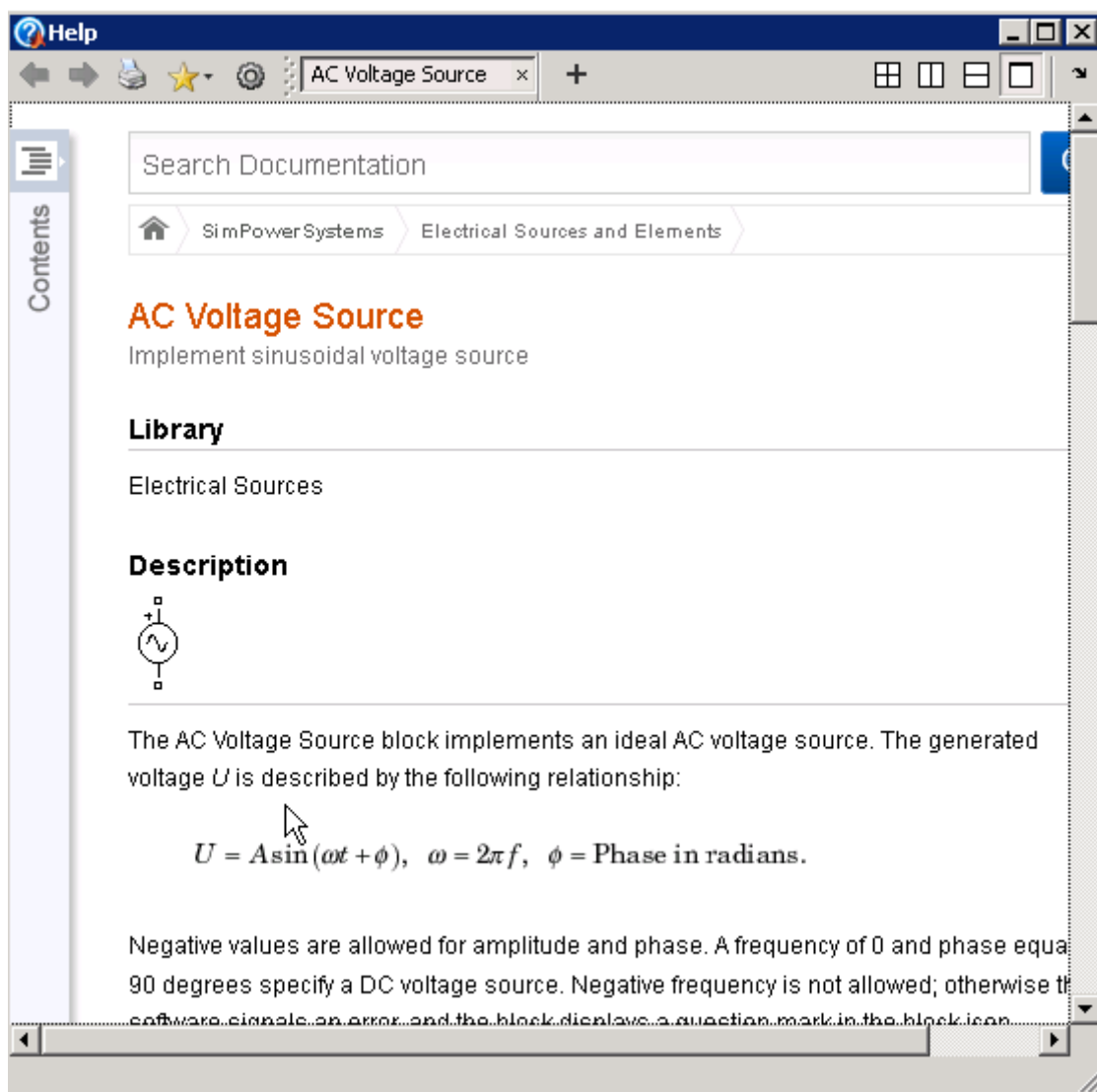
Don't the first one or you won't be able to create wires later!

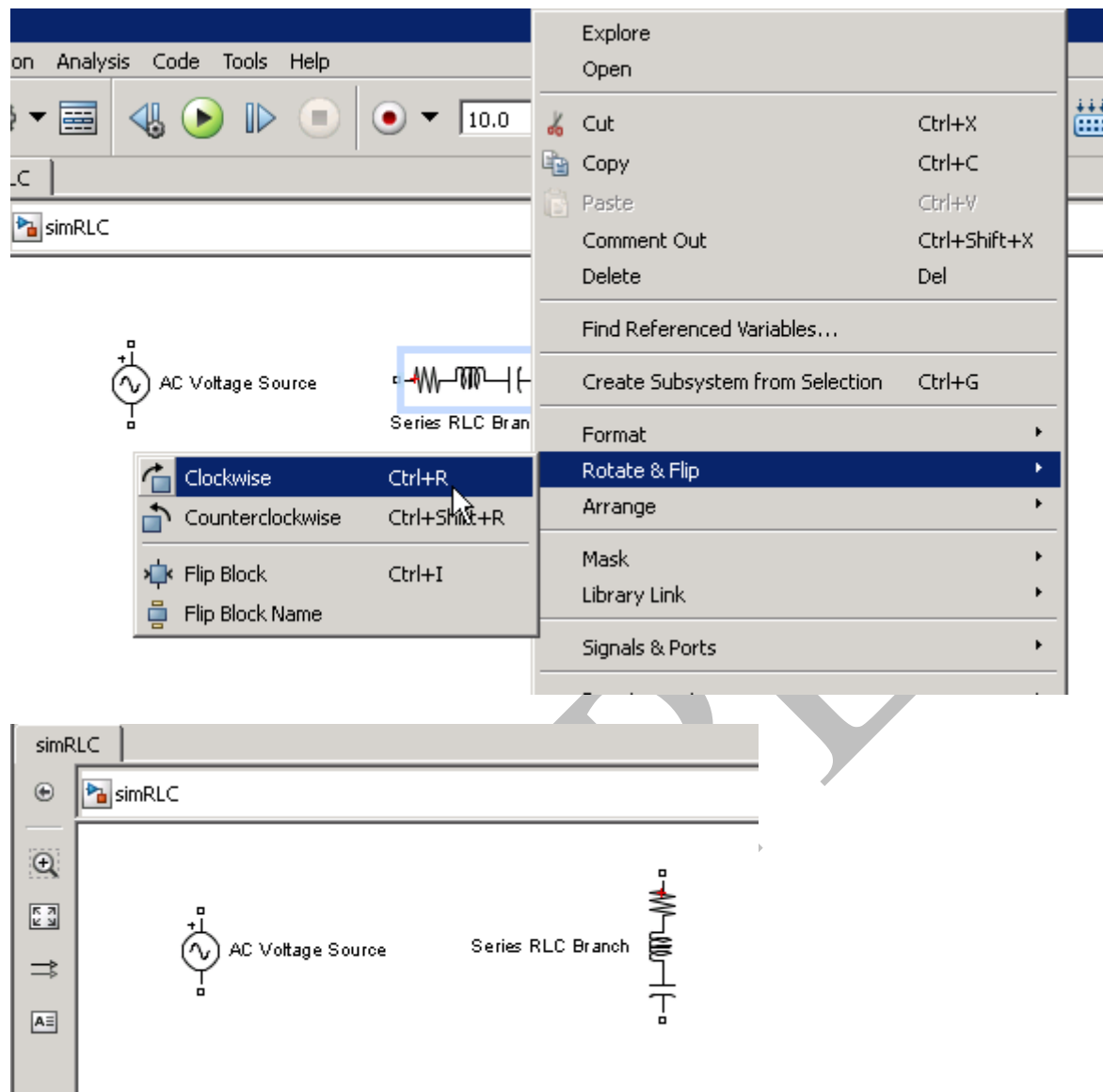




All standard blocks in Simulink have detailed Help. Click the Help button in the Block Parameter window for the specific block in order to get detailed help for that block.

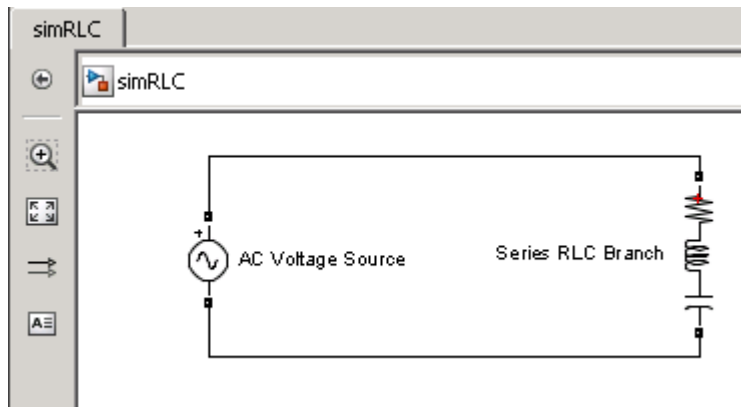




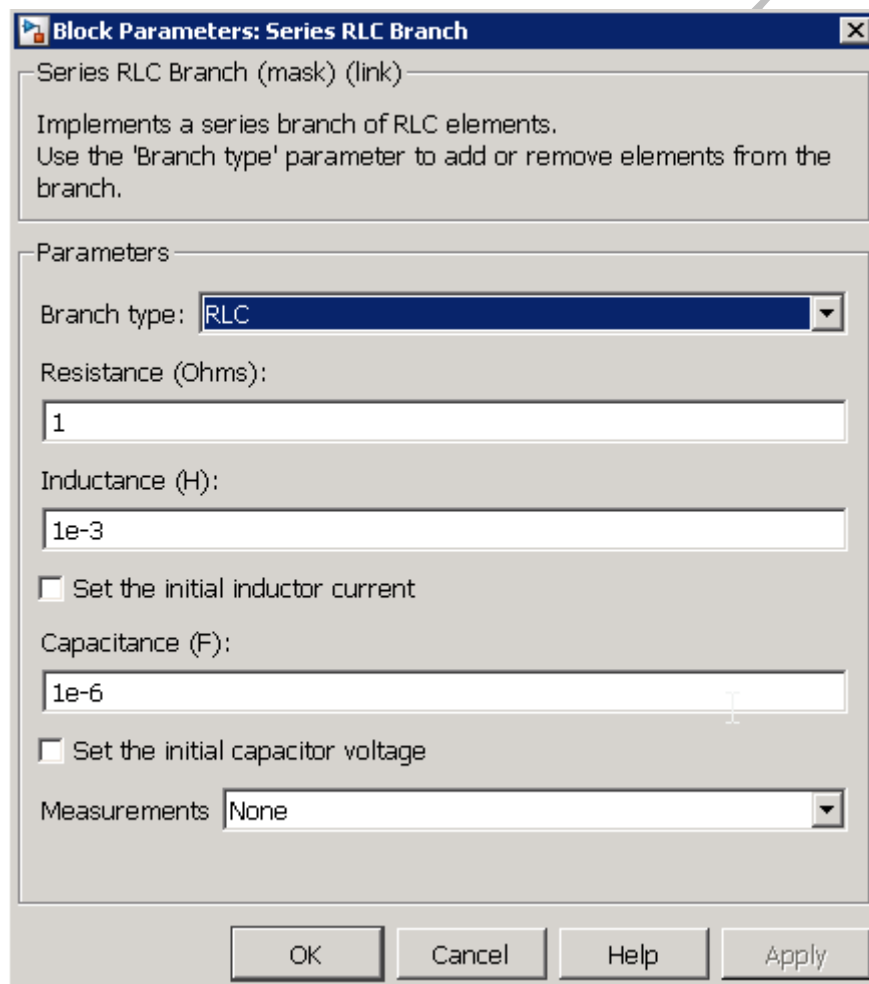


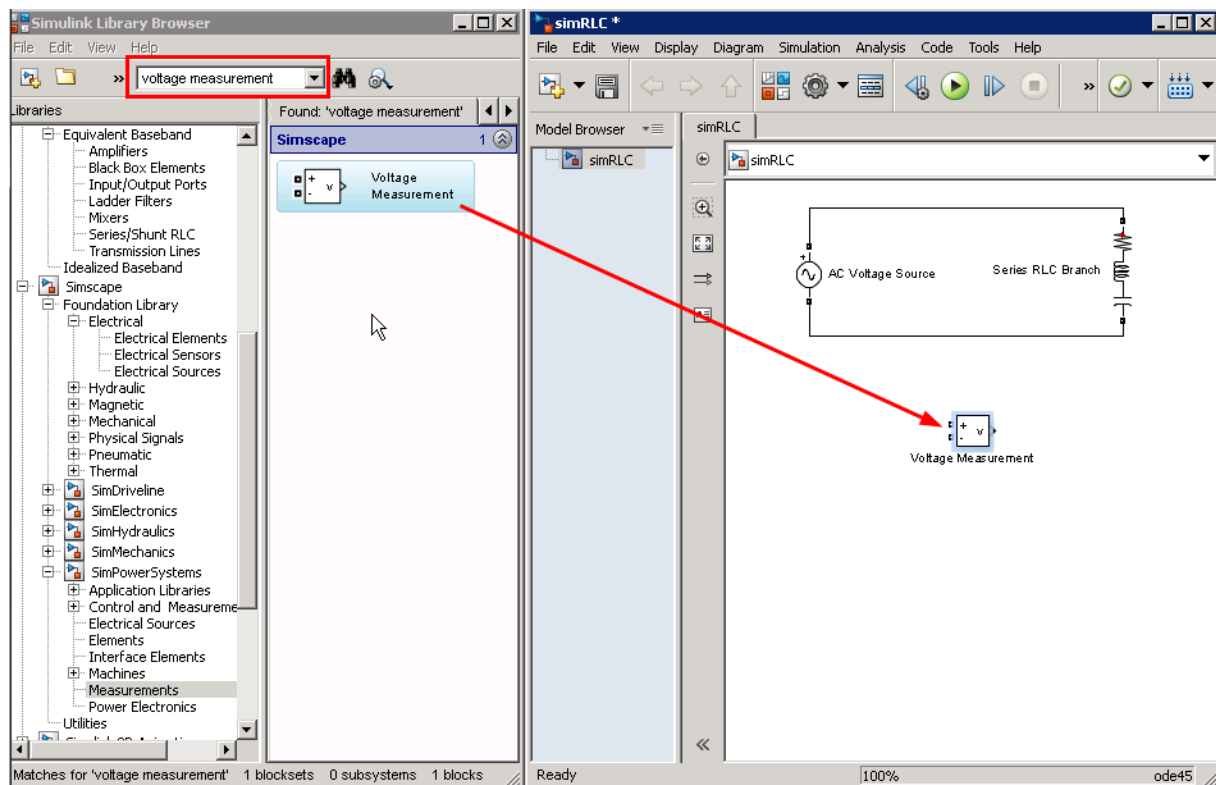
If you want you can create connectors without any problems just by dragging the mouse between connector points.

Create an RLC circuit

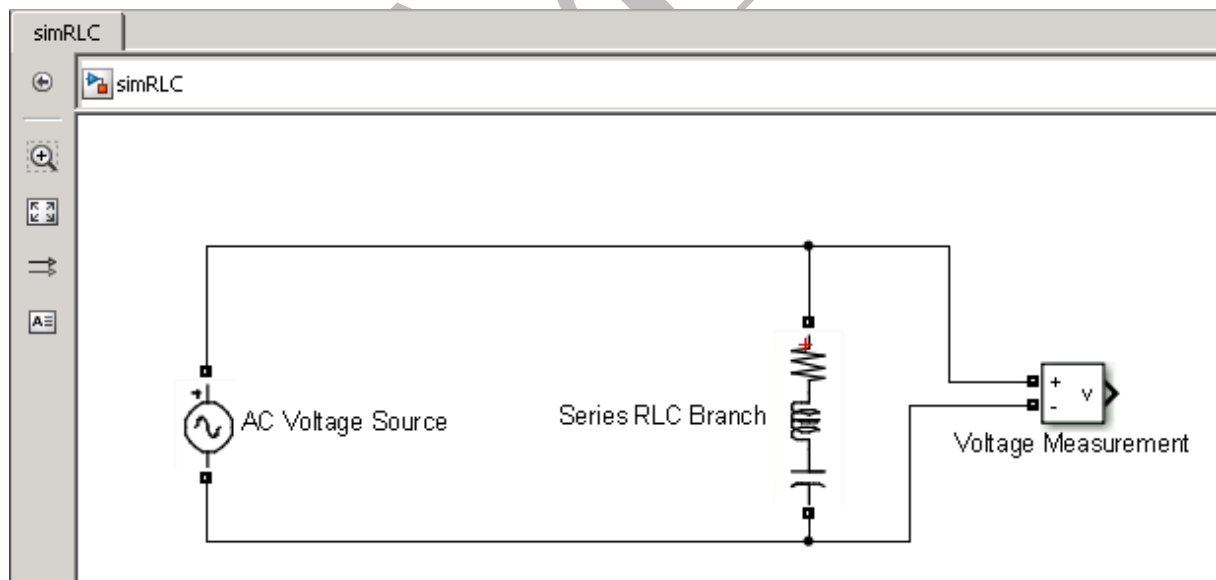


If you double click on the RLC element you get:

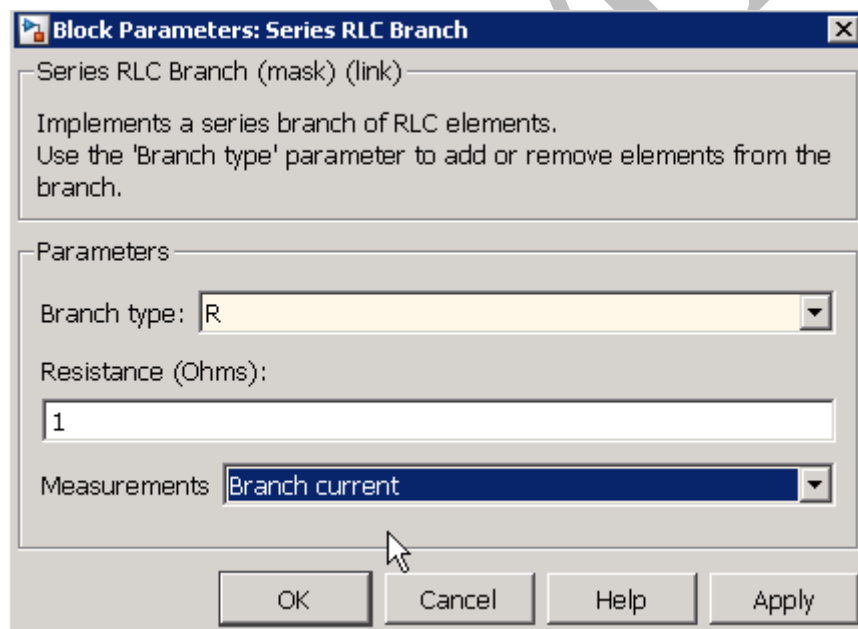
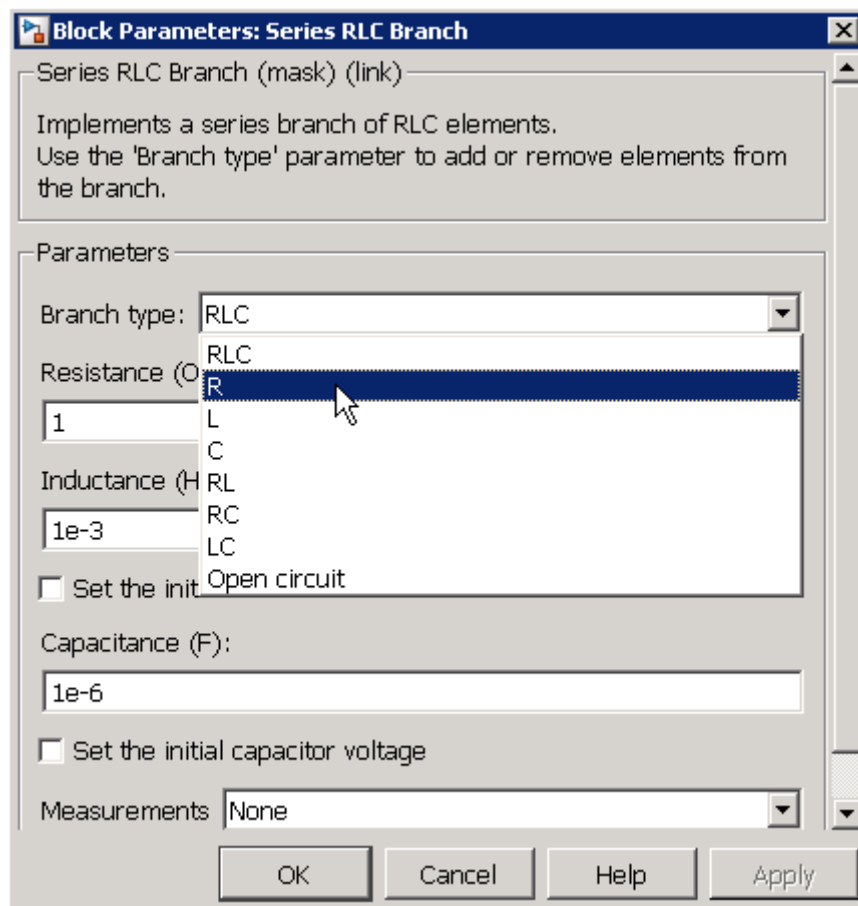




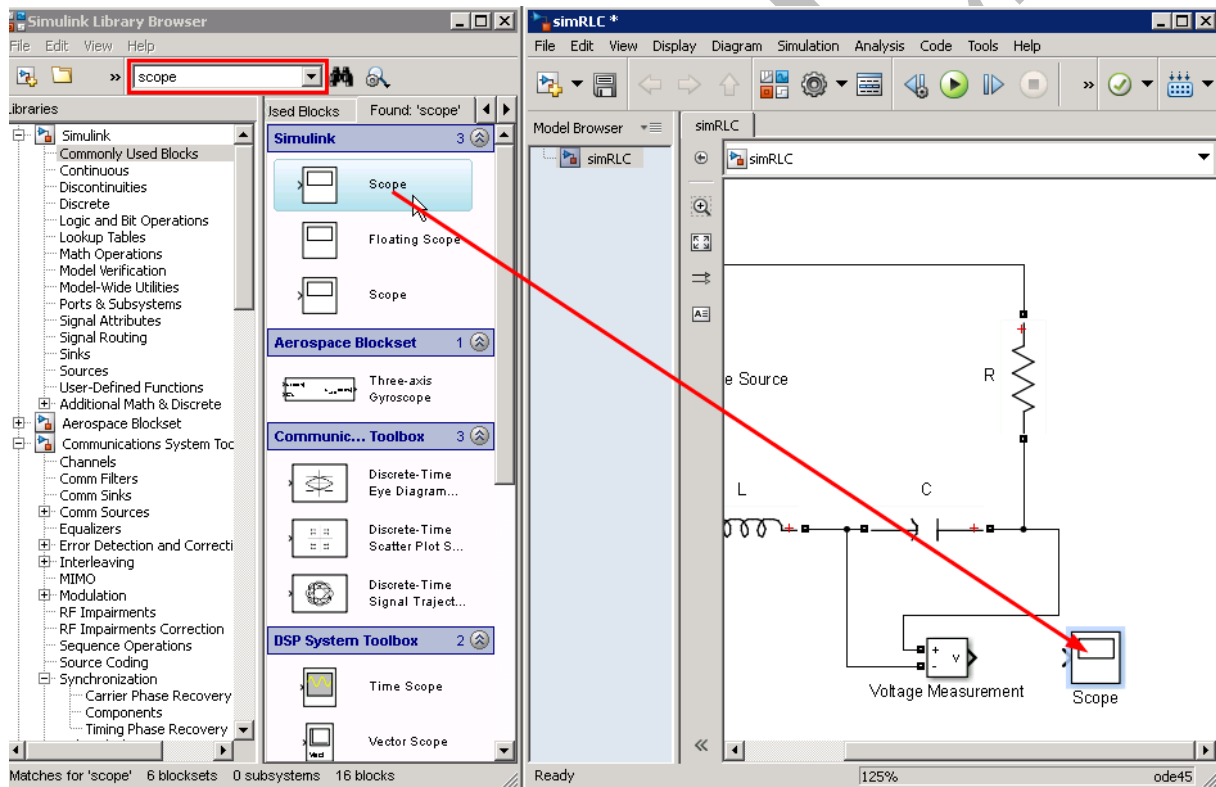
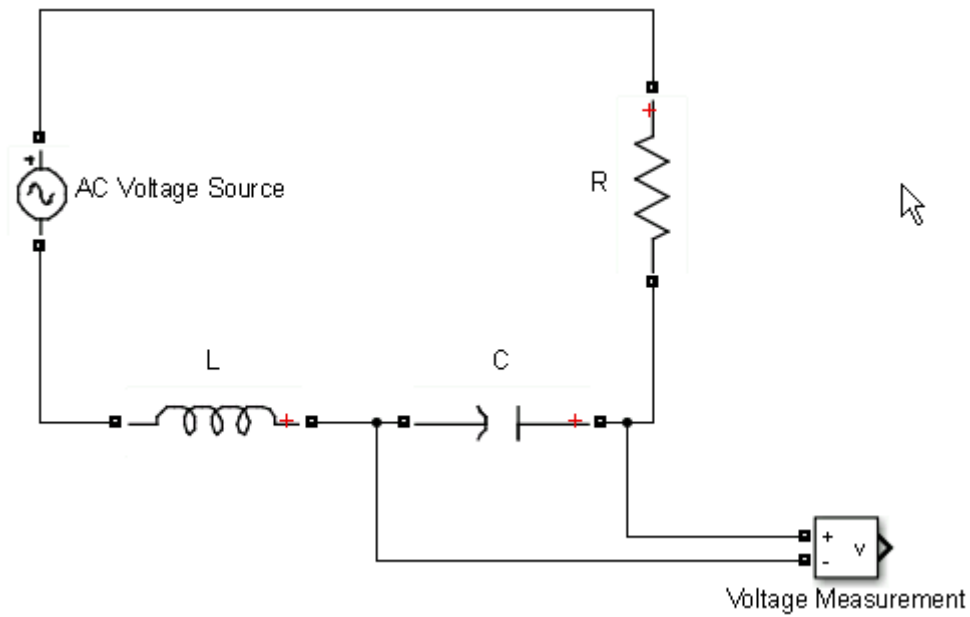
After link the output of the Voltage Measurement and connect them on the existing wire parts (you can't connect the wires between the R only or the L only or the C only with an RLC Branch):

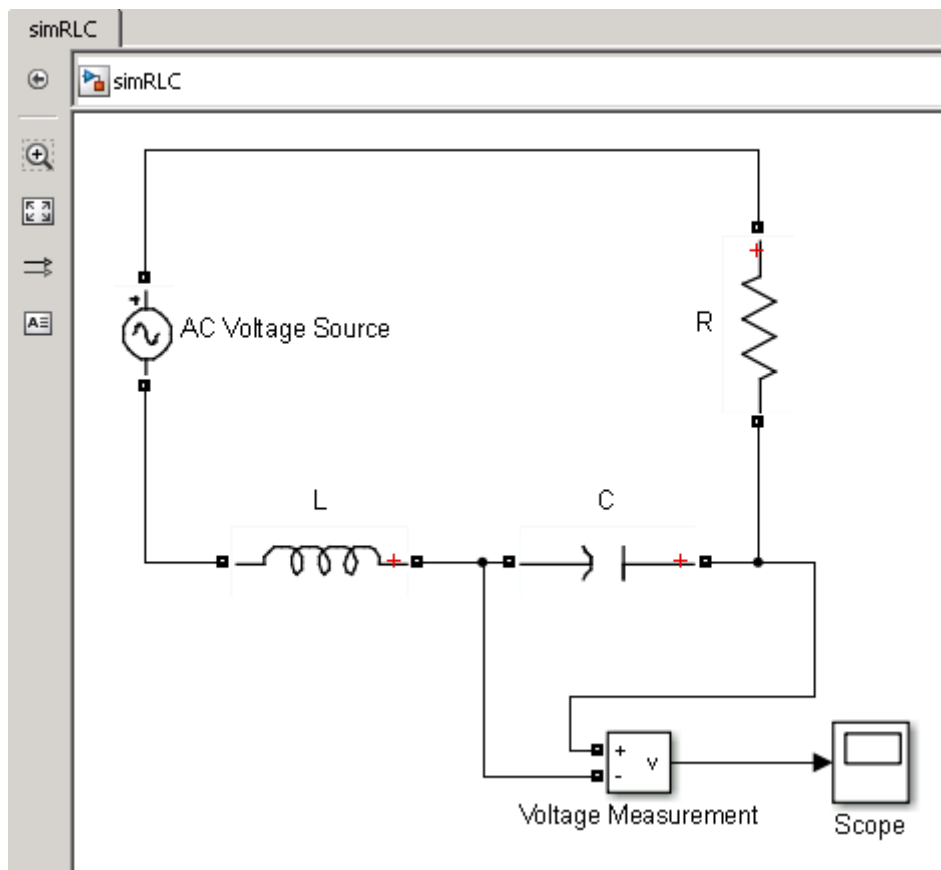


If you want to connect only between one of the items:

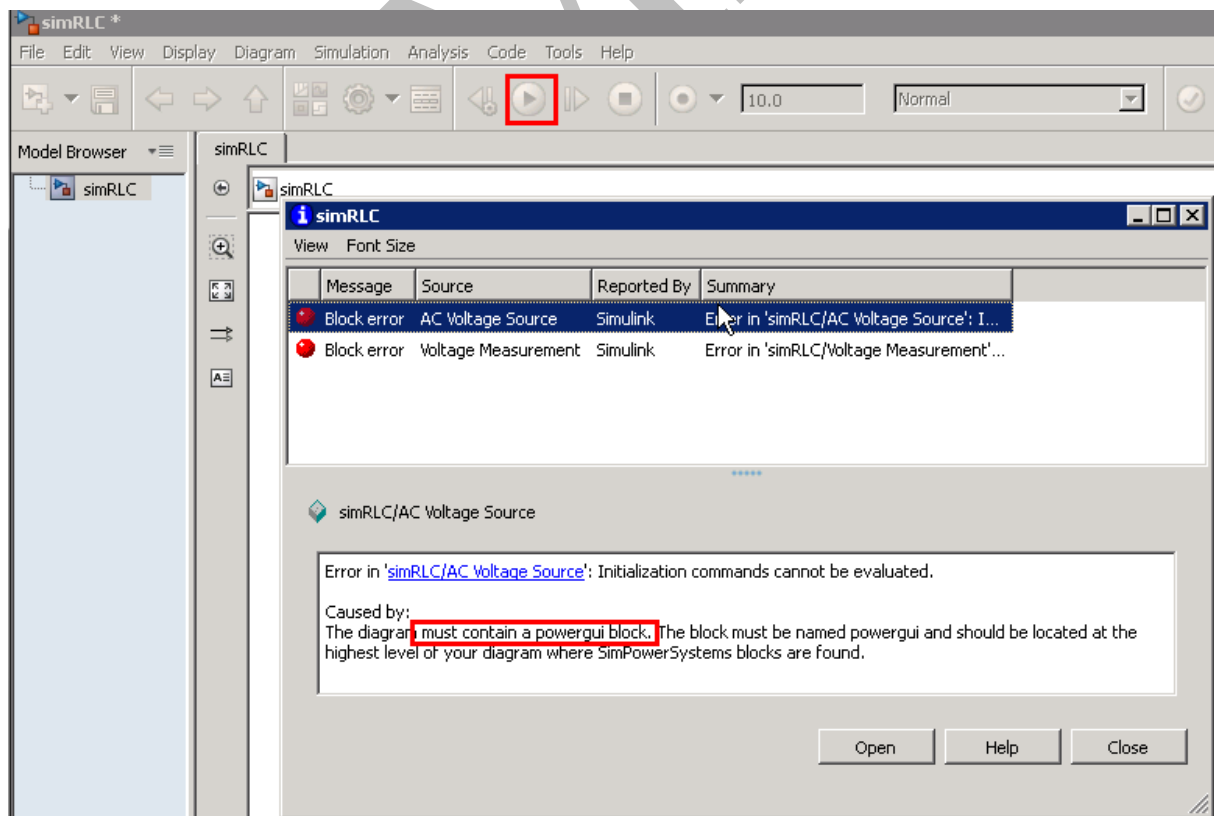


Then we change the circuit into separate components:

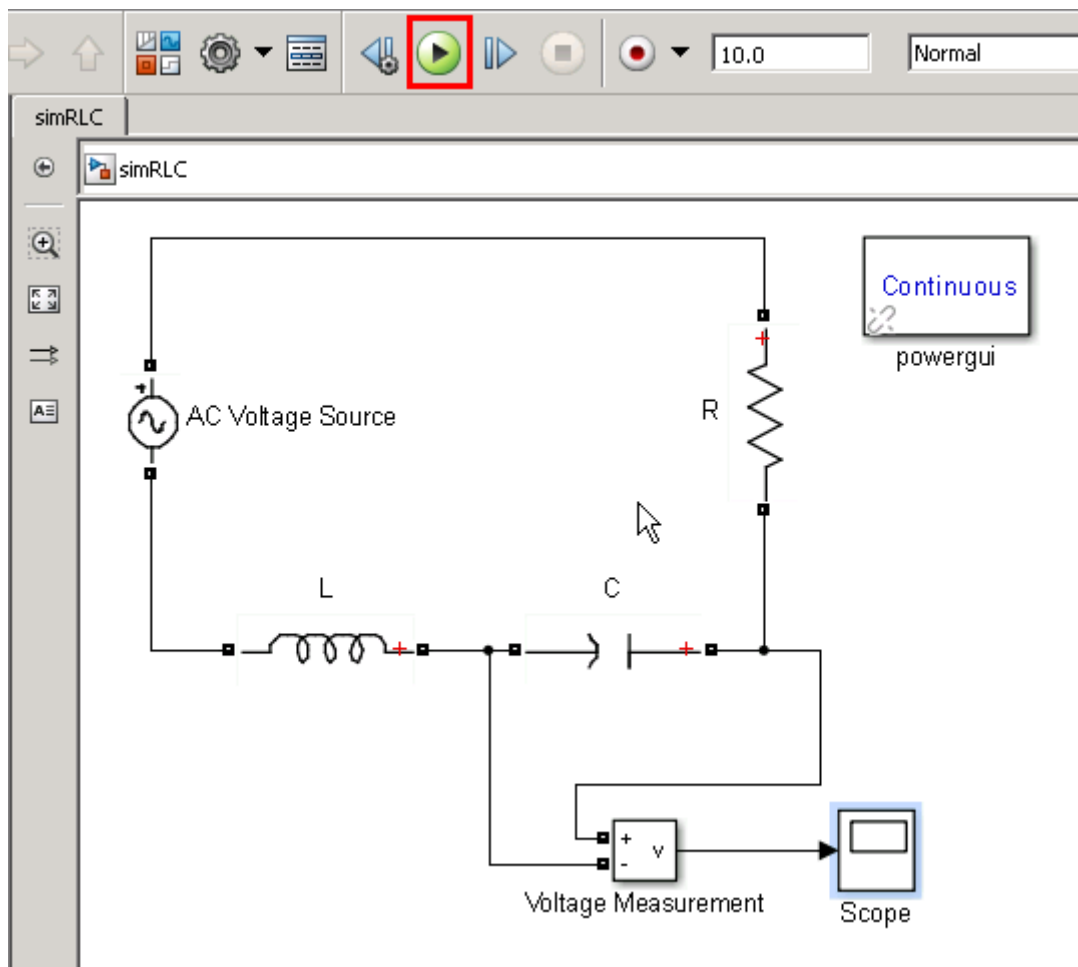




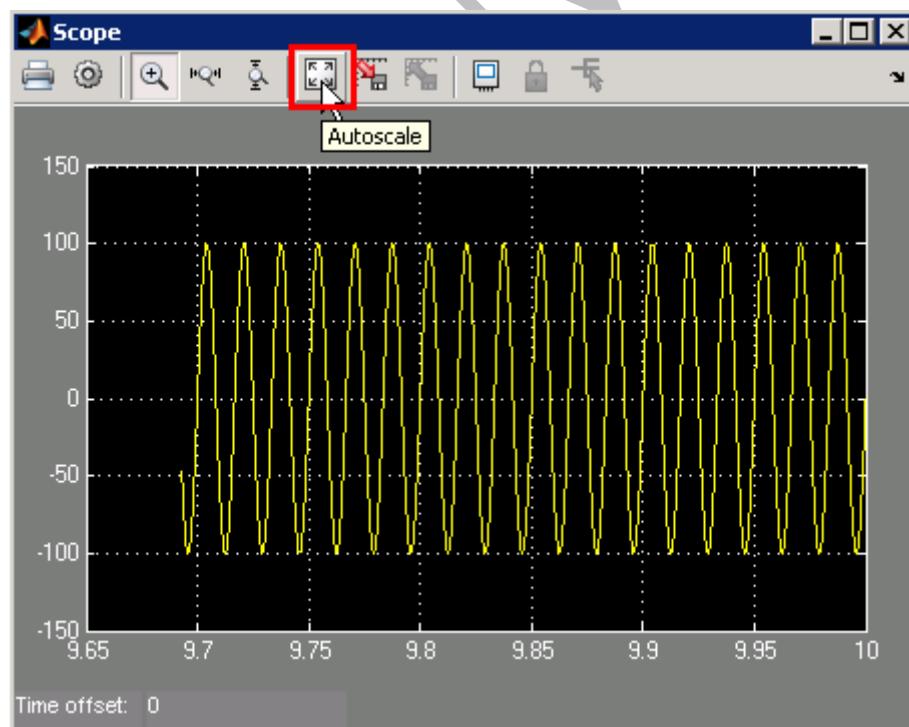
If we run it now it won't run:



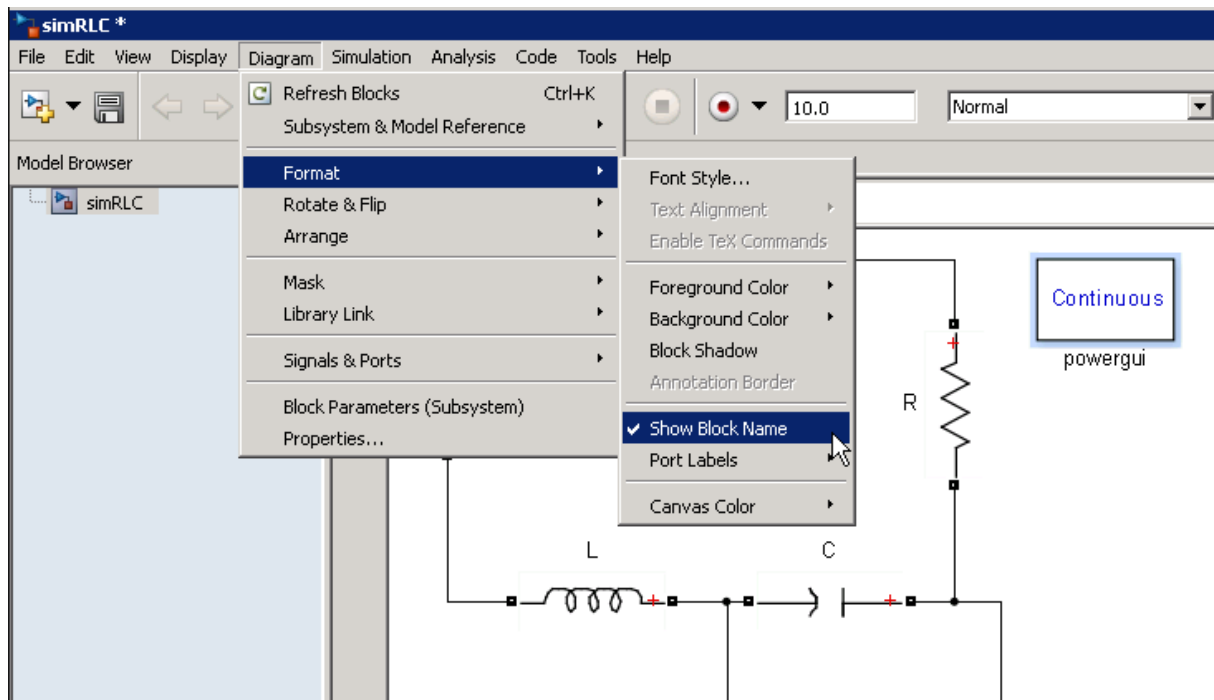
Then:



Once the circuit simulated, double click on the Scope to get:



Now if you wish to hide the names of the blocks you can use (don't hesitate to make multiple selection!):



Neural Networks (Neural Network Toolbox)

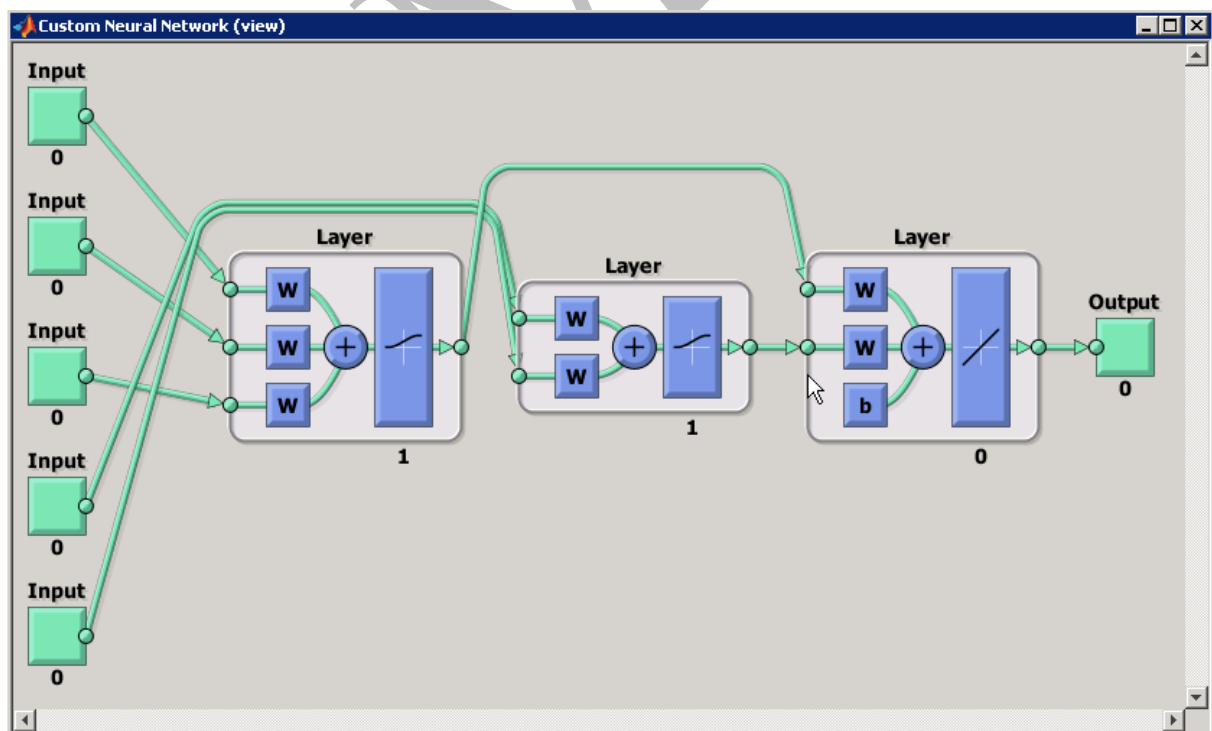
<http://www.mathworks.ch/ch/help/nnet/release-notes.html>

Multiple input/One output Neural Network (MIOO-NN)

```

1 - net=network(...
2     5,... %number of inputs
3     3,... %number of layers
4     [0;0;1],... %bias input for layer i (third layer only in this case)
5     [1 1 1 0 0;0 0 0 1 1;0 0 0 0 0],... %original inputs coming into the layer i
6     [0 0 0;0 0 0;1 1 0],... %layer i output as input of layer j
7     [0 0 1]... %has the layer j an independant output?
8 );
9 %we choose the transfer function for each layer
10 %logsig, purelin, hardlim, logsig
11 - net.layers{1}.transferFcn='logsig';
12 - net.layers{2}.transferFcn='logsig';
13 %number of hidden layers in each layer
14 - net.layers{1}.size=1;
15 - net.layers{2}.size=1;
16 - net.layers{3}.size=0;
17 - view(net)|

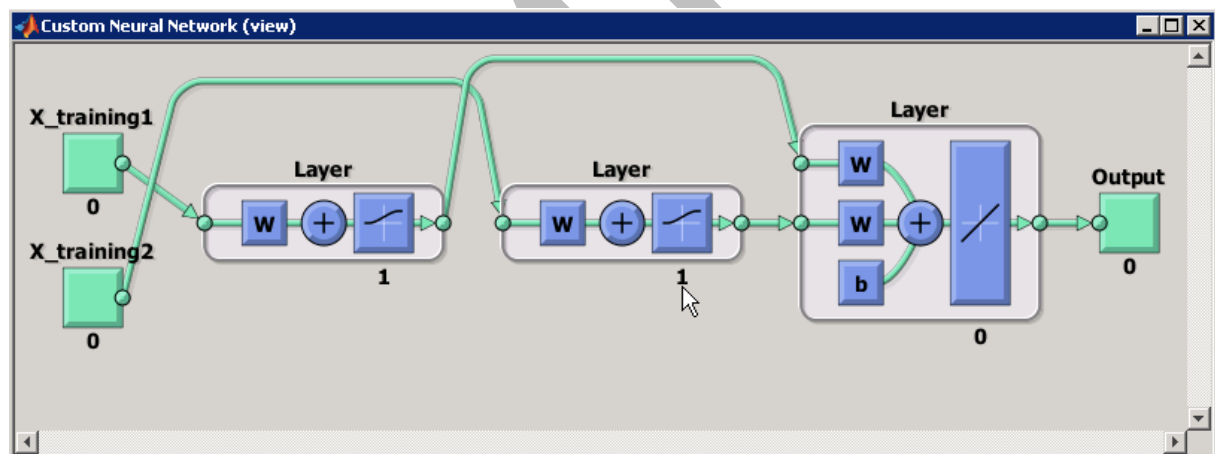
```



```

1 - clear all;close all;clc
2 - net=network(...
3     2,... %number of inputs
4     3,... %number of layers
5     [0;0;1],... %bias input for layer i (third layer only in this case)
6     [1 0;0 1;0 0],... %original inputs coming into the layer i
7     [0 0 0;0 0 0;1 1 0],... %layer i output as input of layer j
8     [0 0 1]... %has the layer j an independant output?
9 );
10 %we choose the transfer function for each layer
11 %logsig, purelin, hardlim, logsig
12 net.layers{1}.transferFcn='logsig';
13 net.layers{2}.transferFcn='logsig';
14 %number of hidden layers in each layer
15 net.layers{1}.size=1;
16 net.layers{2}.size=1;
17 net.layers{3}.size=0;
18 net.inputs{1}.name='X_training1';
19 net.inputs{2}.name='X_training2';
20 net.outputs{1}.name='Y_training';
21 view(net);

```

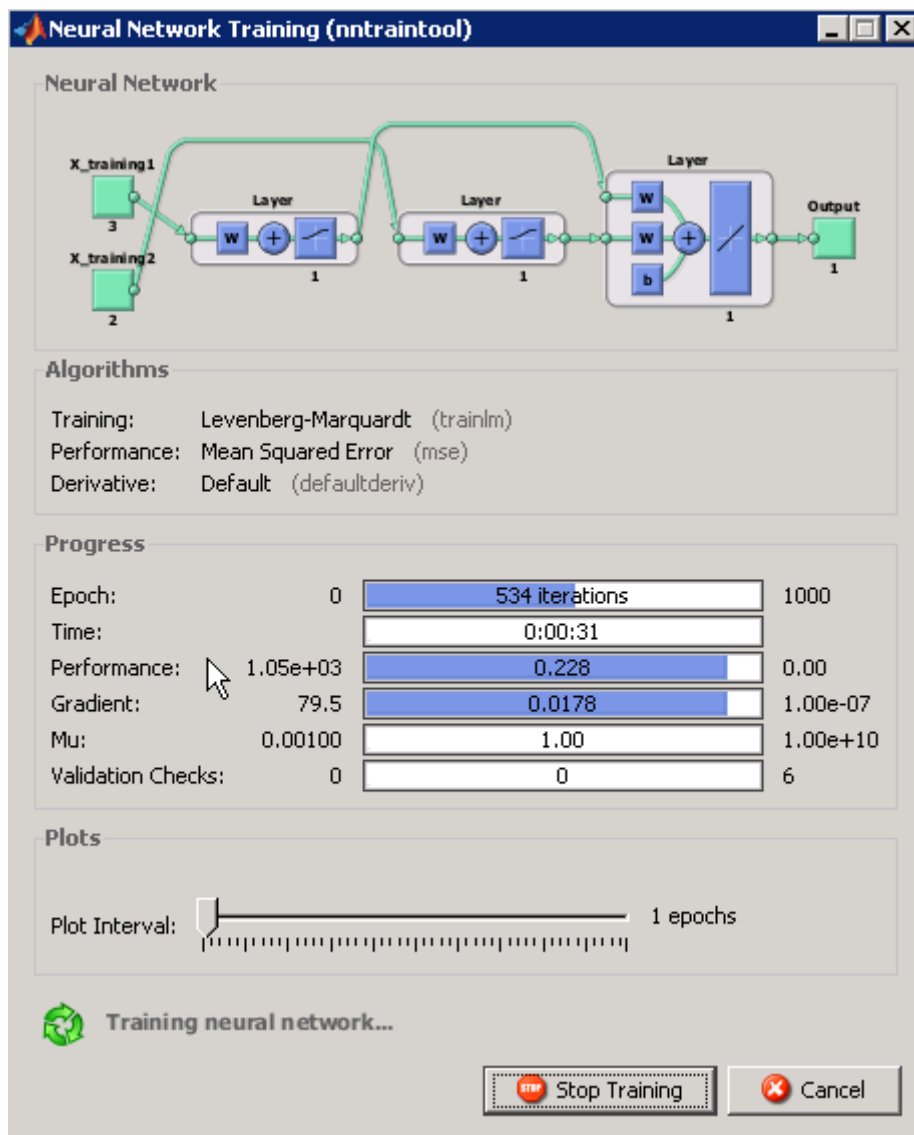


Now let us train the Neural Network:

```

1 - clear all;close all;clc
2 - net=network(...
3     2,... %number of inputs
4     3,... %number of layers
5     [0;0;1],... %bias input for layer i (third layer only in this case)
6     [1 0;0 1;0 0],... %original inputs coming into the layer i
7     [0 0 0;0 0 0;1 1 0],... %layer i output as input of layer j
8     [0 0 1]... %has the layer j an independant output?
9 );
10 %we choose the transfer function for each layer
11 %logsig, purelin, hardlim, logsig
12 - net.layers(1).transferFcn='logsig';
13 - net.layers(2).transferFcn='logsig';
14 %number of hidden layers in each layer
15 - net.layers(1).size=1;
16 - net.layers(2).size=1;
17 - net.layers(3).size=0;
18 - net.inputs(1).name='X_training1';
19 - net.inputs(2).name='X_training2';
20 - net.outputs(1).name='Y_training';
21 - net.numinputs = 2;
22 - view(net);
23
24 - X_training=double(dataset('XLSFile','c:\tmp\nn\NeuralNetworkTrainingSet.xls'));
25 - X_training1=X_training(:,1:3);
26 - X_training2=X_training(:,4:5);
27 - Y_training=double(dataset('XLSFile','c:\tmp\nn\NeuralNetworkTrainingTarget.xls'));
28 - net.trainFcn='trainlm';
29 - net.performFcn='mse';
30 - net=train(net,{X_training1';X_training2'},Y_training')|

```




```
Command Window

net =

Neural Network

    name: 'Custom Neural Network'
 efficiency: .cacheDelayedInputs, .flattenTime,
            .memoryReduction, .flattenedTime
    userdata: (your custom info)

dimensions:

    numInputs: 2
    numLayers: 3
    numOutputs: 1
    numInputDelays: 0
    numLayerDelays: 0
    numFeedbackDelays: 0
    numWeightElements: 8
    sampleTime: 1

connections:

    biasConnect: [0; 0; 1]
    inputConnect: [1 0; 0 1; 0 0]
    layerConnect: [0 0 0; 0 0 0; 1 1 0]
```

```

    outputConnect: [0 0 1]

subobjects:

    inputs: {2x1 cell array of 2 inputs}
    layers: {3x1 cell array of 3 layers}
    outputs: {1x3 cell array of 1 output}
    biases: {3x1 cell array of 1 bias}
    inputWeights: {3x2 cell array of 2 weights}
    layerWeights: {3x3 cell array of 2 weights}

functions:

    adaptFcn: (none)
    adaptParam: (none)
    derivFcn: 'defaultderiv'
    divideFcn: (none)
    divideParam: (none)
    divideMode: 'sample'
    initFcn: 'initlay'
    performFcn: 'mse'
    performParam: .regularization, .normalization
    plotFcns: {}
    plotParams: {1x0 cell array of 0 params}
    trainFcn: 'trainlm'
    trainParam: .showWindow, .showCommandLine, .show, .epochs,
    .time, .goal, .min_grad, .max_fail, .mu, .mu_dec,
    .mu inc, .mu max

weight and bias values:

    IW: {3x2 cell} containing 2 input weight matrices
    LW: {3x3 cell} containing 2 layer weight matrices
    b: {3x1 cell} containing 1 bias vector

methods:

    adapt: Learn while in continuous use
    configure: Configure inputs & outputs
    gensim: Generate Simulink model
    init: Initialize weights & biases
    perform: Calculate performance
    sim: Evaluate network outputs given inputs
    train: Train network with examples
    view: View diagram
    unconfigure: Unconfigure inputs & outputs

evaluate:      outputs = net(inputs)

```

Command Window

```
>> net.IW

ans =

    [1x3 double]    []
           []    [1x2 double]
           []    []
```

```
>> net.IW{1}

ans =

    -0.0480    0.0673    -0.0585
```

```
>> net.IW{2,2}

ans =

    0.0532    -0.0078
```

```
>> net.LW

ans =

           []           []           []
           []           []           []
    [10.4258]    [10.7986]           []
```

```
>> net.b

ans =

    []
    []
    [20.2050]
```

```

Command Window

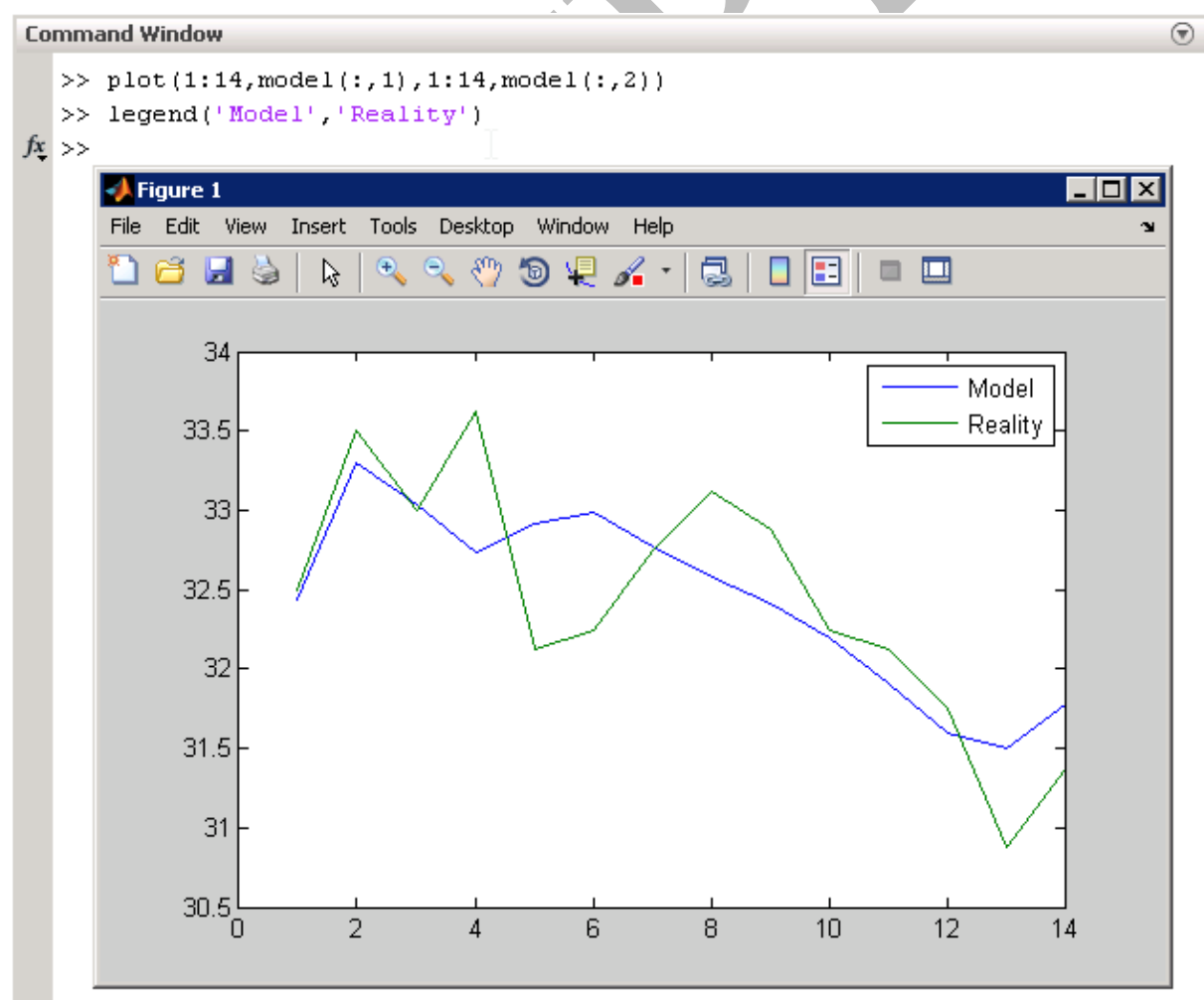
>> model=[cell2mat(net({X_training1';X_training2'}))',Y_training]

model =

    32.4325    32.5000
    33.3047    33.5000
    33.0391    33.0000
    32.7406    33.6250
    32.9124    32.1250
    32.9916    32.2500
    32.7703    32.7500
    32.5749    33.1250
    32.4162    32.8750
    32.1965    32.2500
    31.9064    32.1250
    31.5933    31.7500
    31.5004    30.8750
    31.7761    31.3750

fx >> |

```



```
Command Window
>> SSE=sum((model(:,1)-model(:,2)).^2)

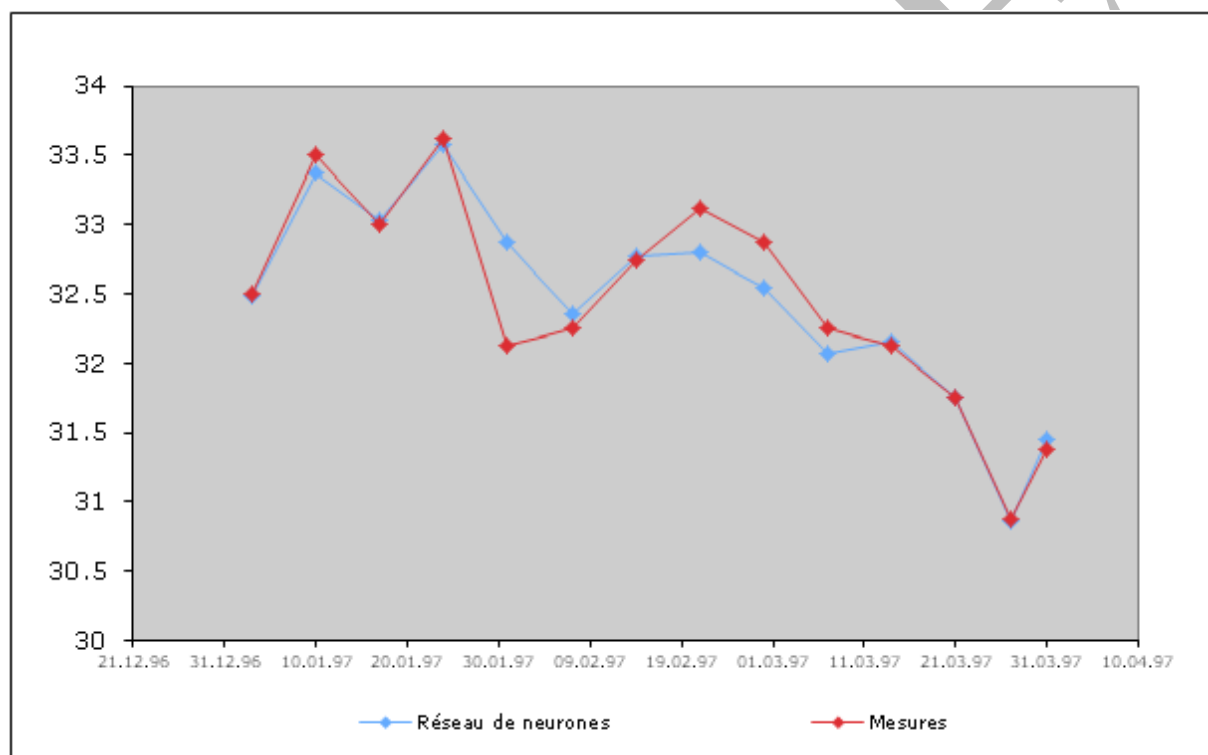
SSE =

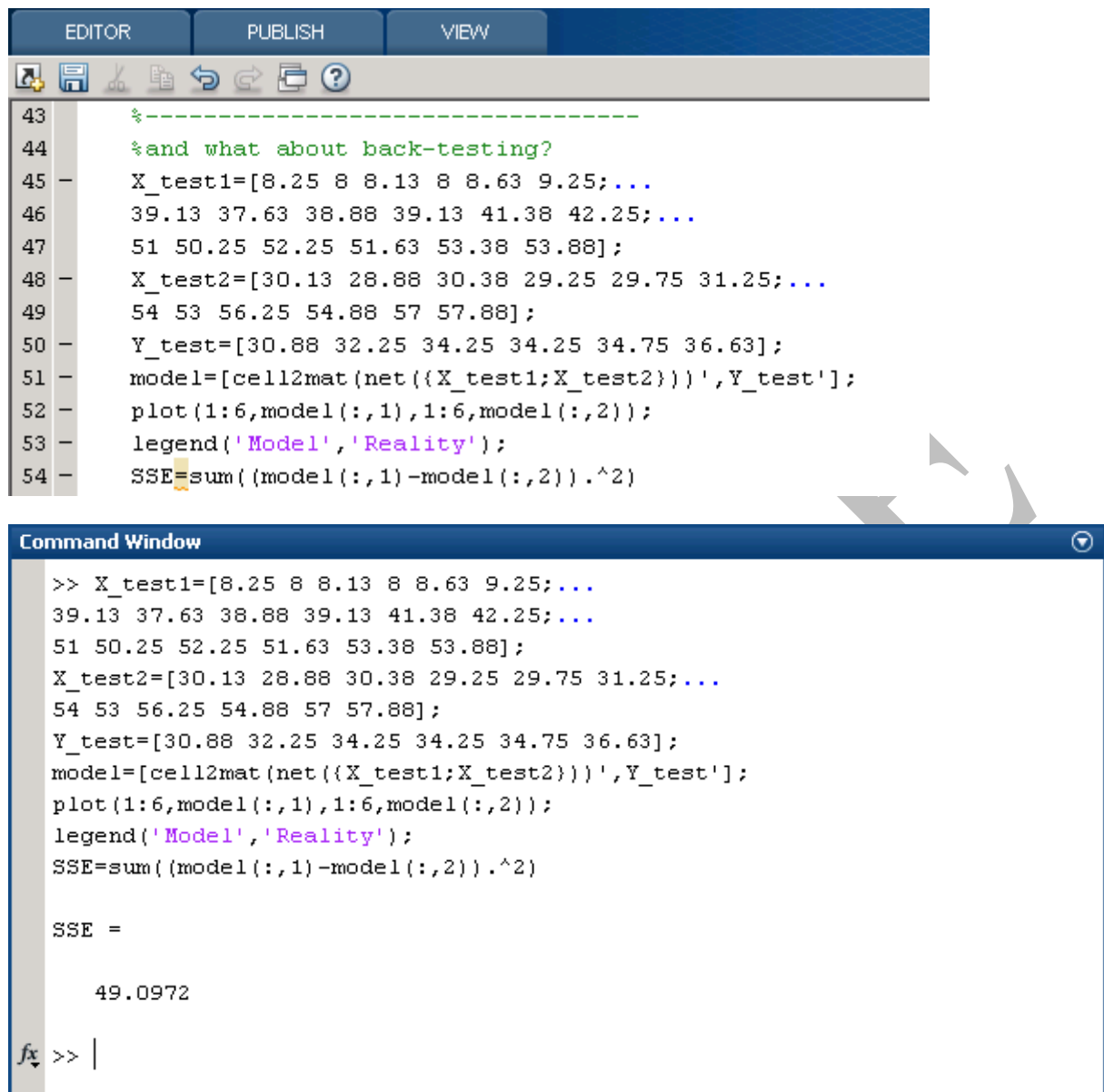
    3.1371

fx >> |
```

Worst than with MS Excel...

To compare with MS Excel:





The image shows the MATLAB software interface. The top bar has tabs for EDITOR, PUBLISH, and VIEW. Below the tabs is a toolbar with icons for file operations and editing. The main window displays a script with the following code:

```

43 %-----
44 %and what about back-testing?
45 - X_test1=[8.25 8 8.13 8 8.63 9.25;...
46 39.13 37.63 38.88 39.13 41.38 42.25;...
47 51 50.25 52.25 51.63 53.38 53.88];
48 - X_test2=[30.13 28.88 30.38 29.25 29.75 31.25;...
49 54 53 56.25 54.88 57 57.88];
50 - Y_test=[30.88 32.25 34.25 34.25 34.75 36.63];
51 - model=[cell2mat(net({X_test1;X_test2}))',Y_test'];
52 - plot(1:6,model(:,1),1:6,model(:,2));
53 - legend('Model','Reality');
54 - SSE=sum((model(:,1)-model(:,2)).^2)

```

Below the script is the Command Window, which shows the execution of the code and the resulting SSE value:

```

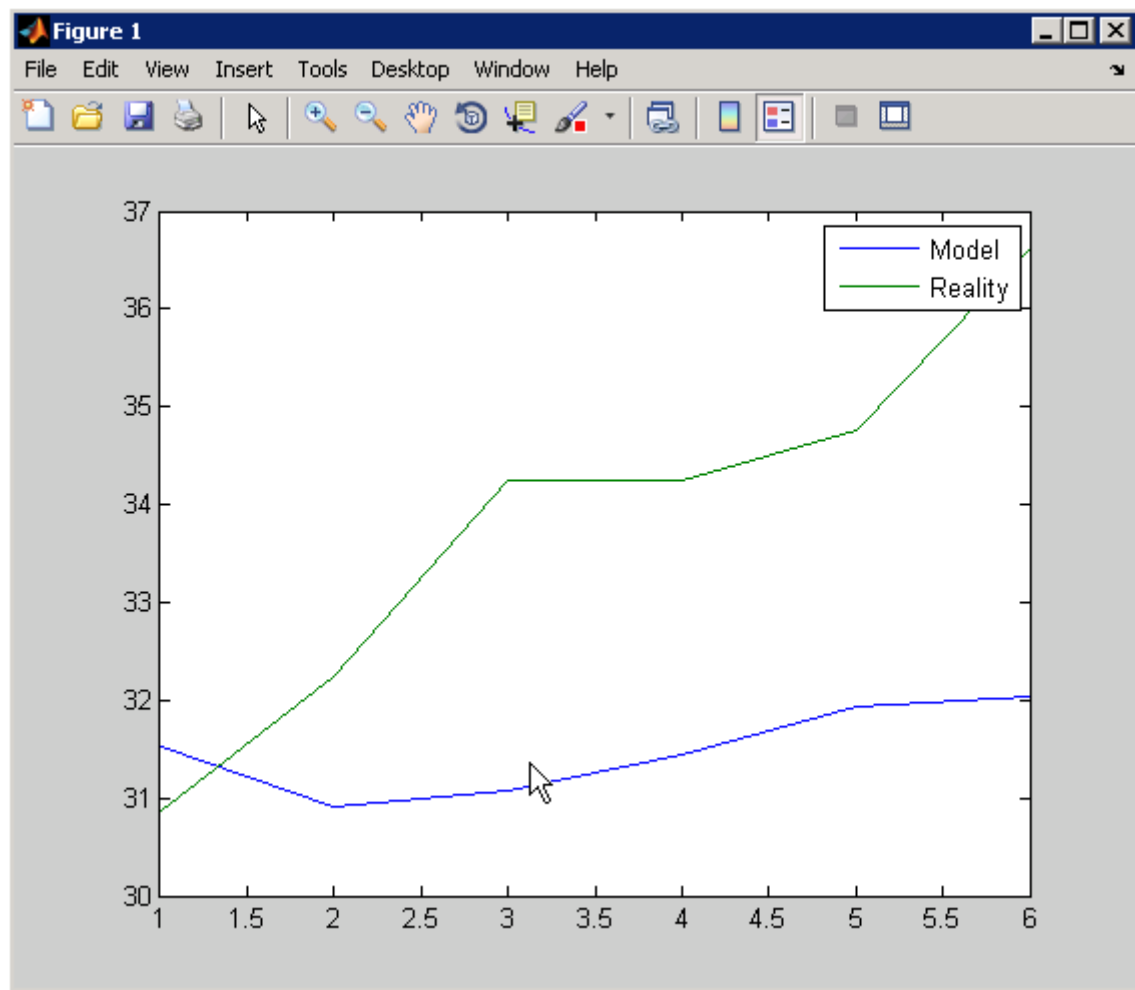
>> X_test1=[8.25 8 8.13 8 8.63 9.25;...
39.13 37.63 38.88 39.13 41.38 42.25;...
51 50.25 52.25 51.63 53.38 53.88];
X_test2=[30.13 28.88 30.38 29.25 29.75 31.25;...
54 53 56.25 54.88 57 57.88];
Y_test=[30.88 32.25 34.25 34.25 34.75 36.63];
model=[cell2mat(net({X_test1;X_test2}))',Y_test'];
plot(1:6,model(:,1),1:6,model(:,2));
legend('Model','Reality');
SSE=sum((model(:,1)-model(:,2)).^2)

SSE =

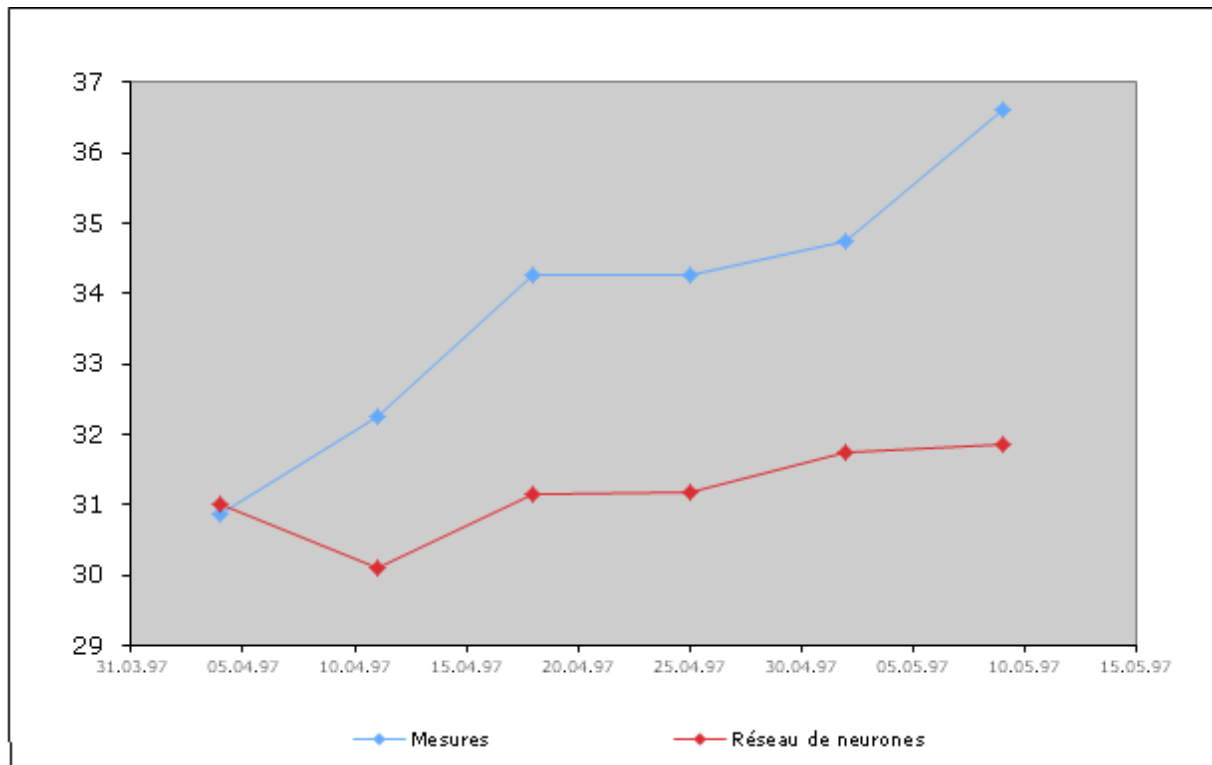
    49.0972

```

The Command Window prompt is `fx >> |`.



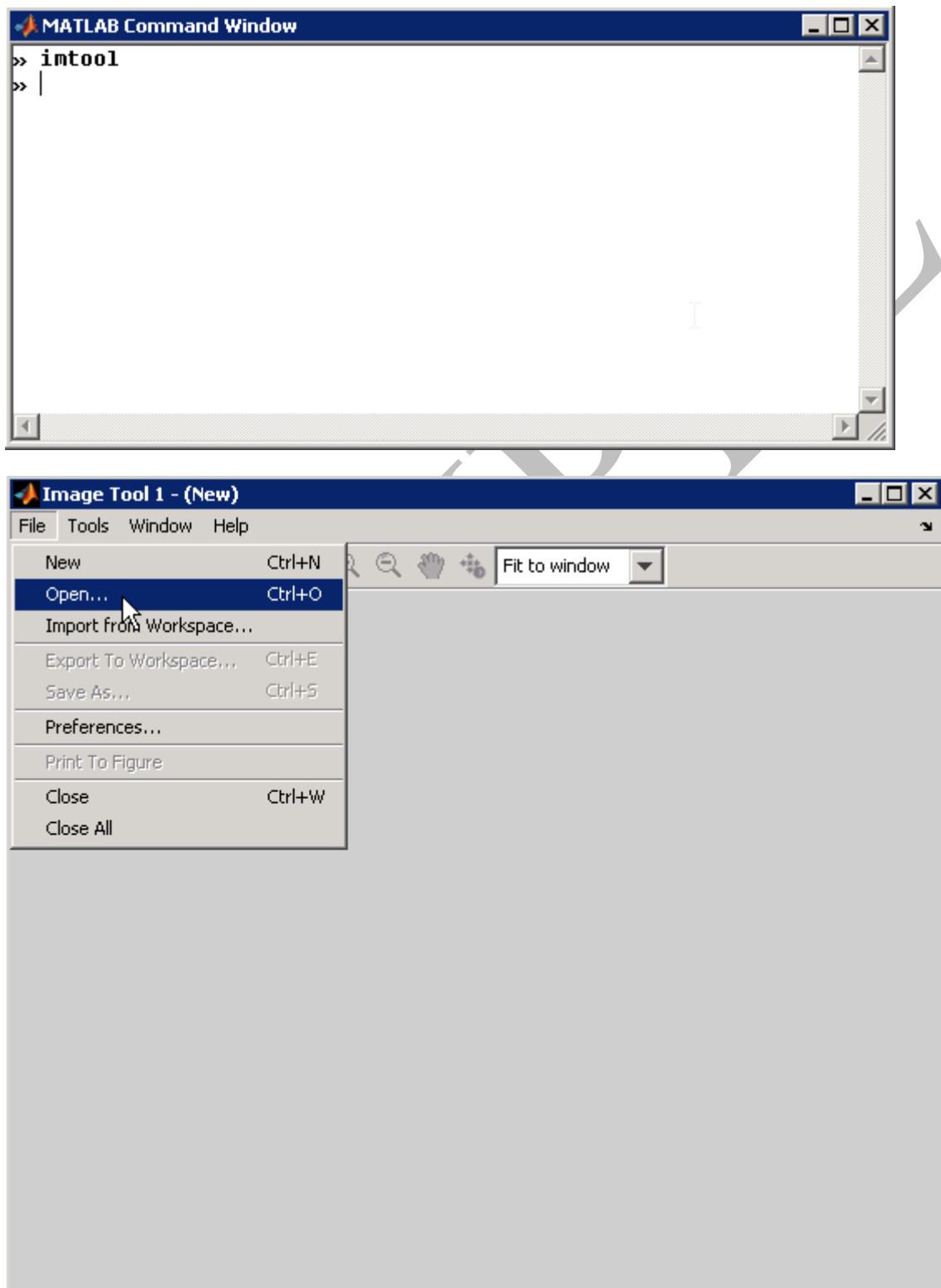
And it MS Excel we get:

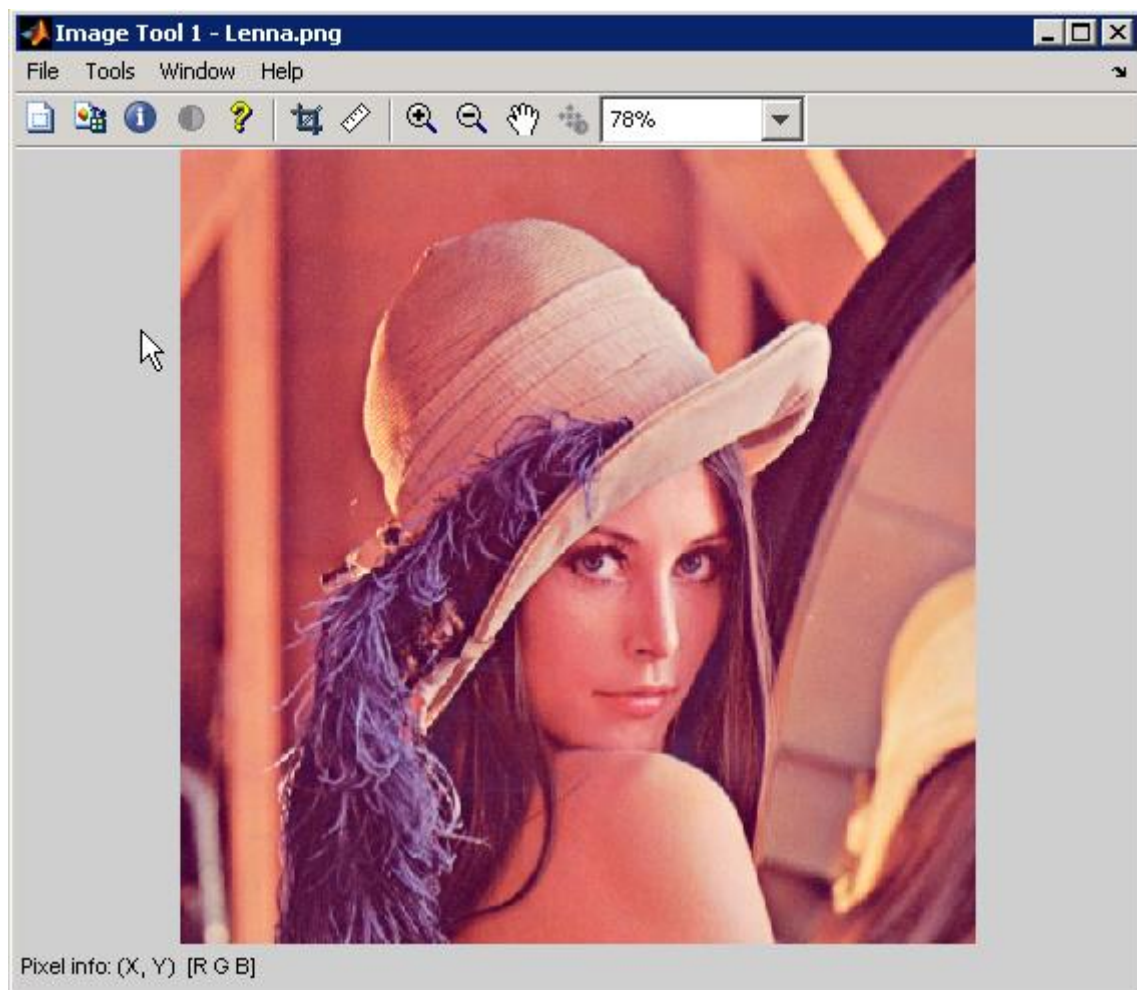


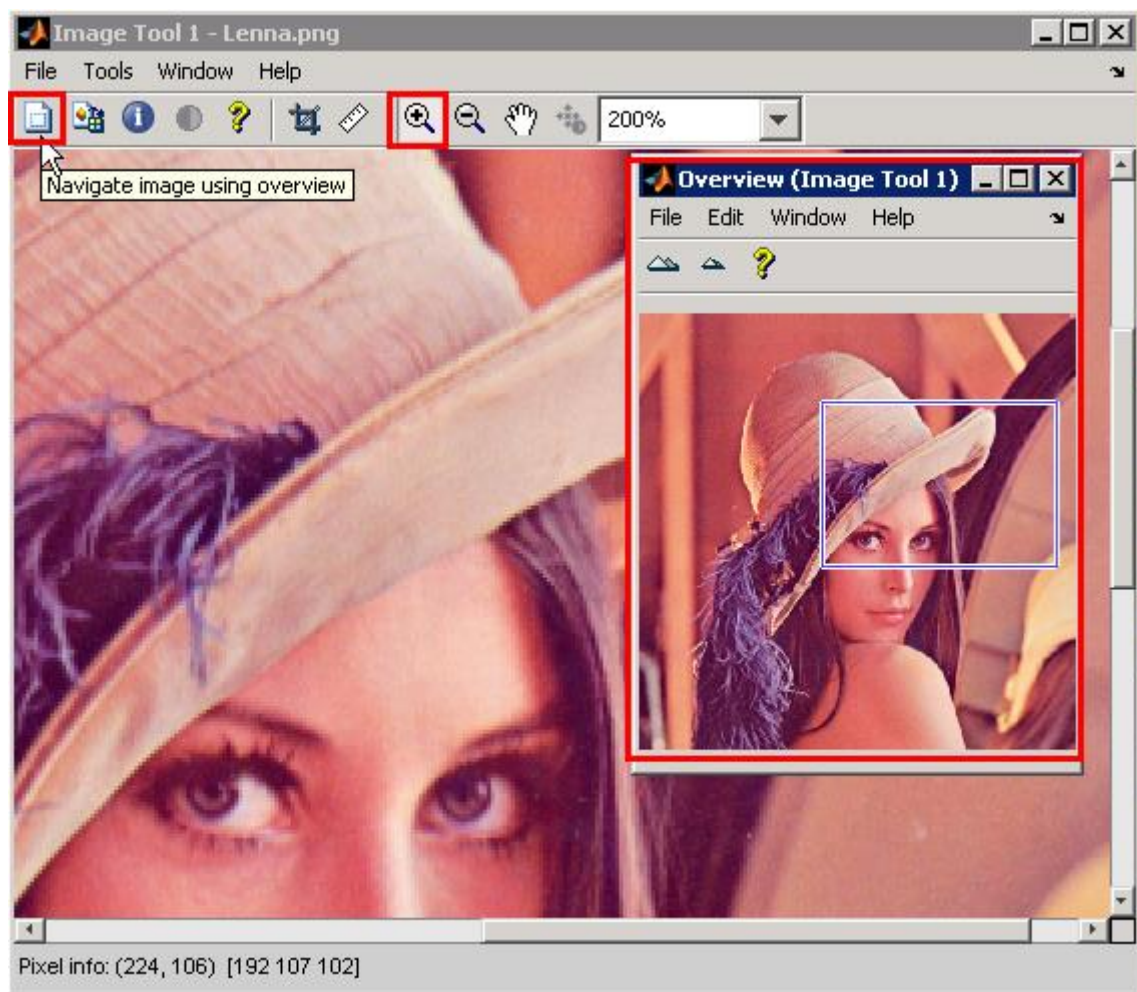
with a SSE of 55. MATLAB makes better for prediction!

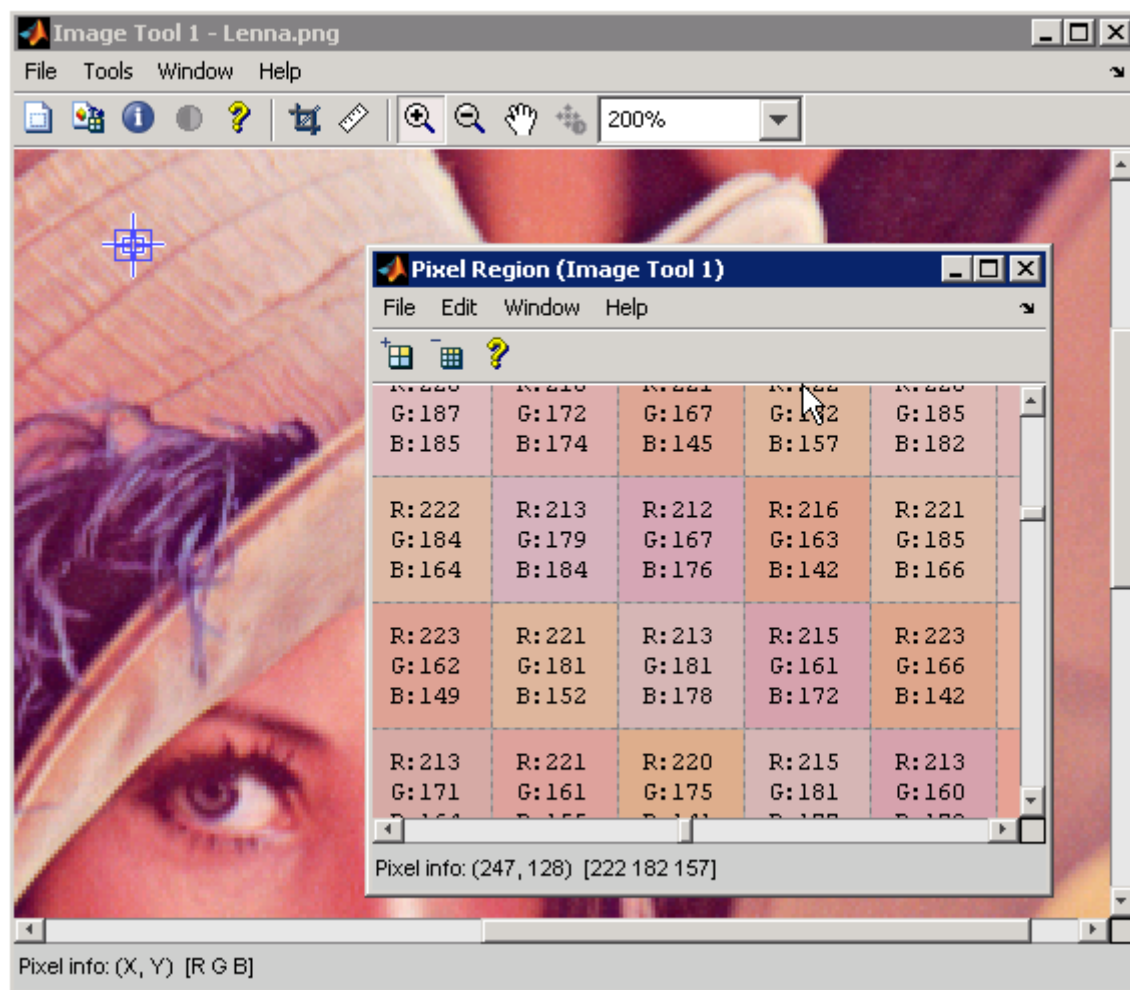
Image Processing (with Image Processing and Acquisition Toolboxes)

Play with the image tool wizard









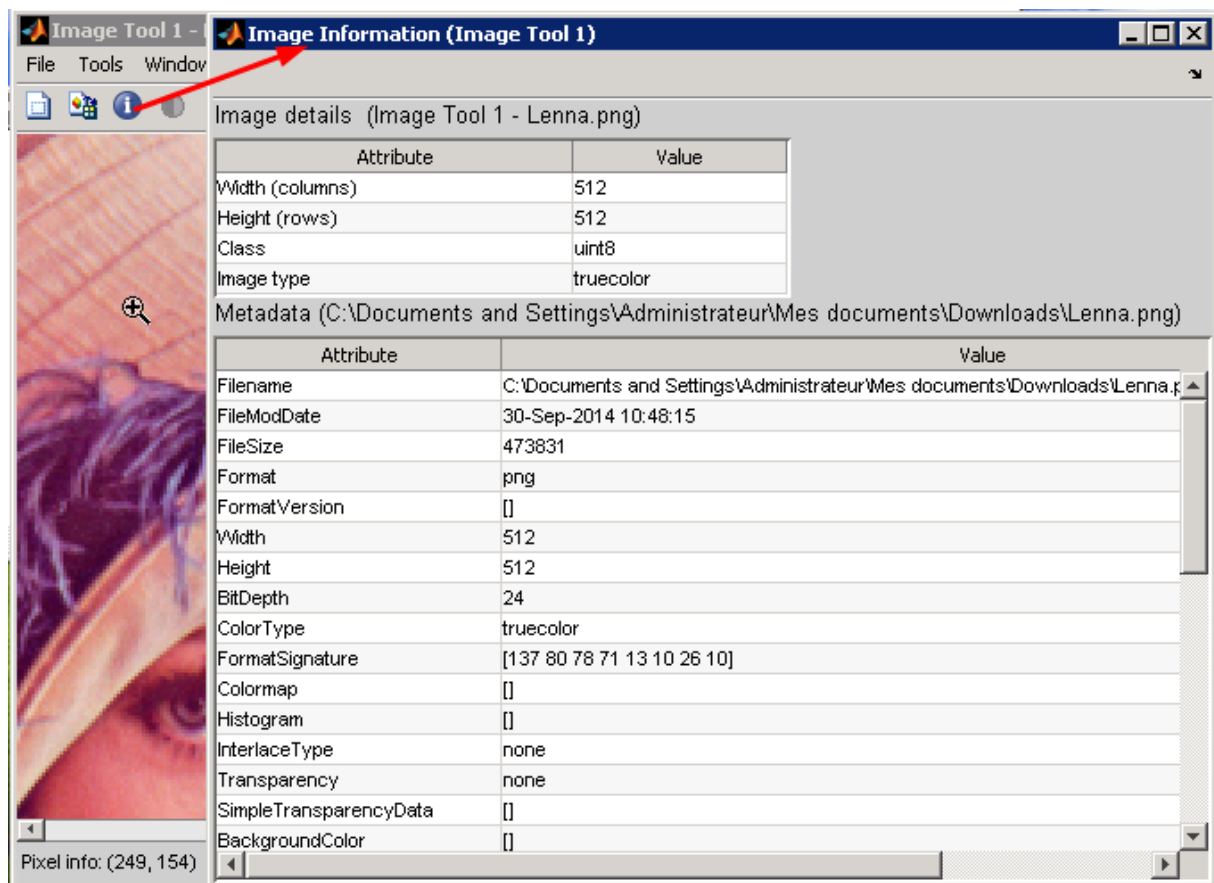
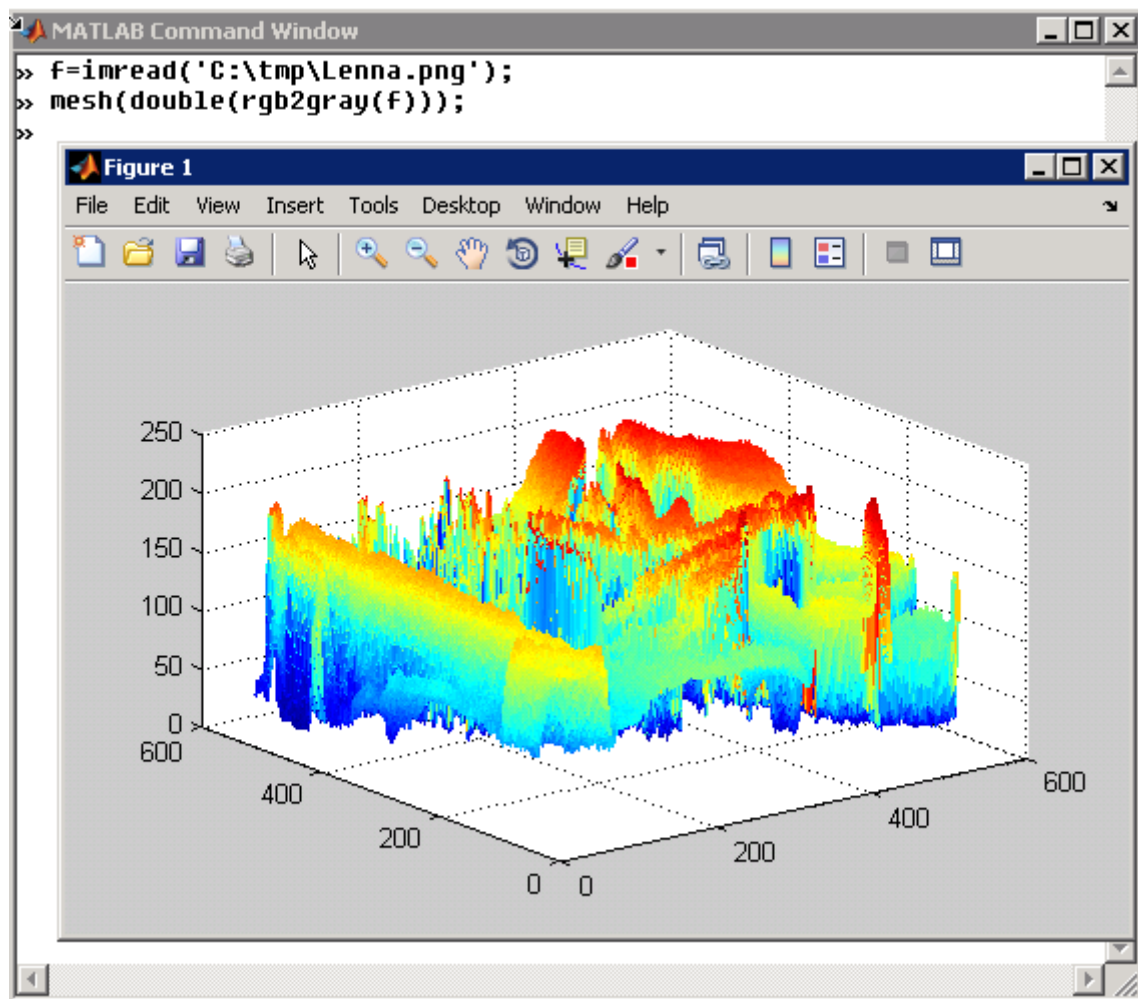


Image as a function

See the image as a function:



If we change the perspective:

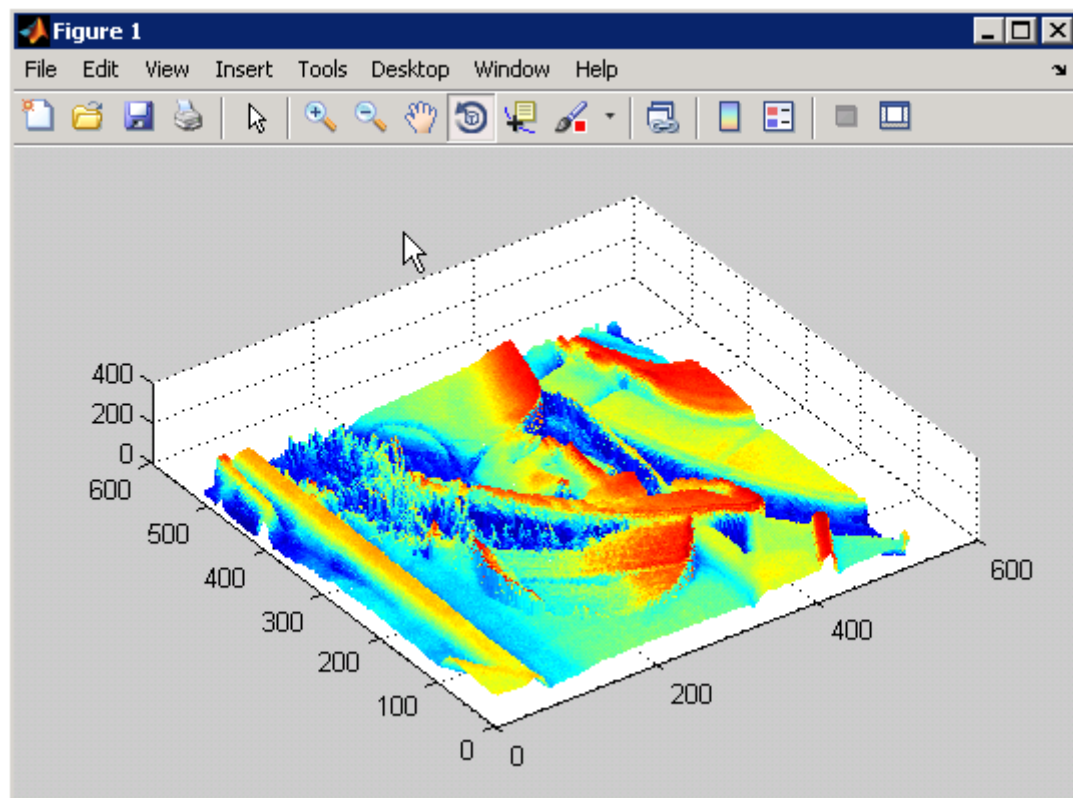
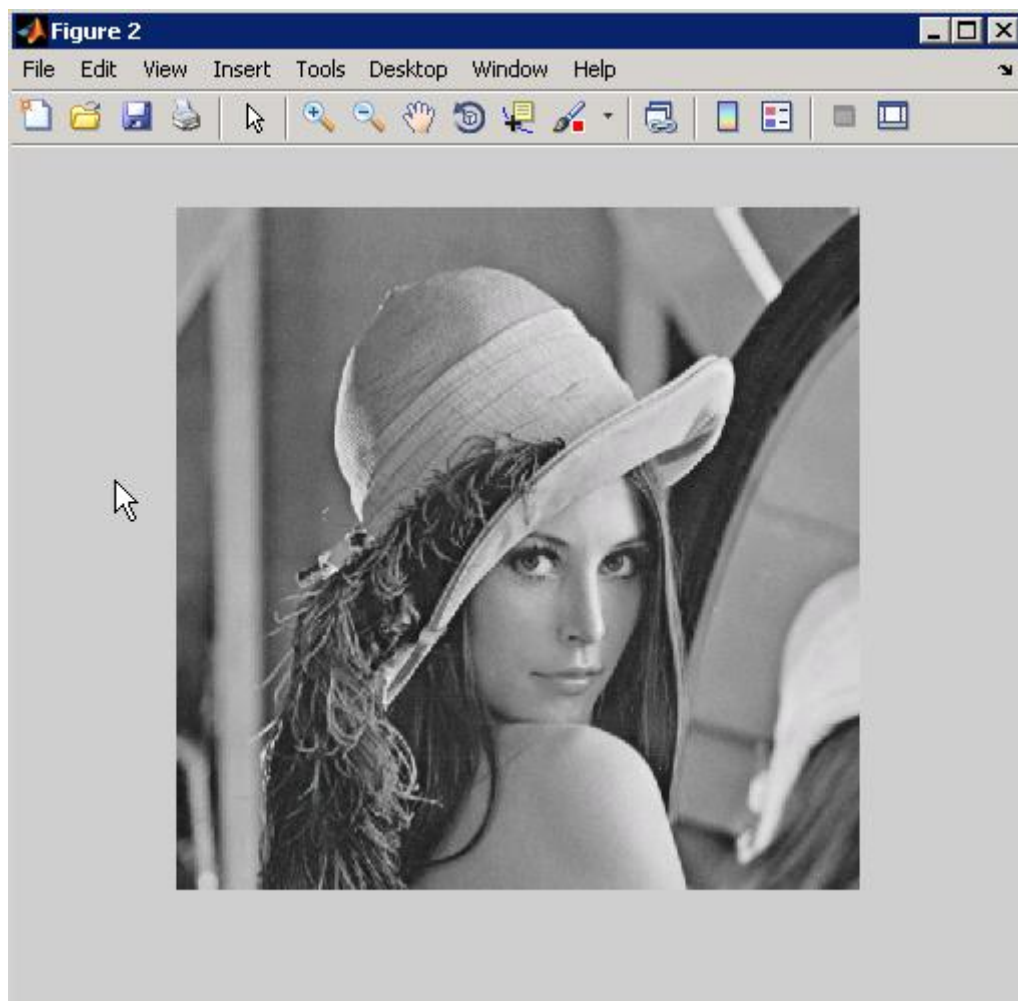
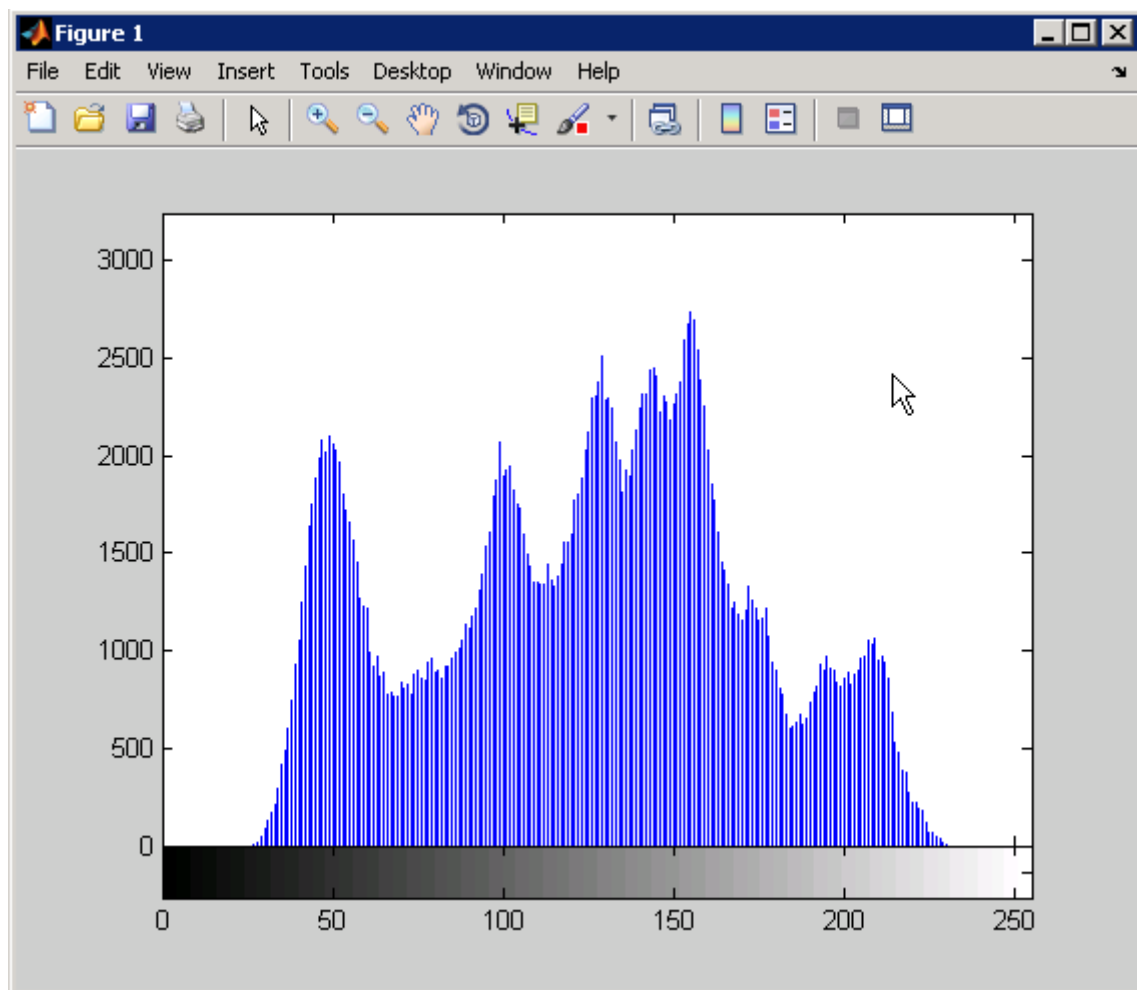


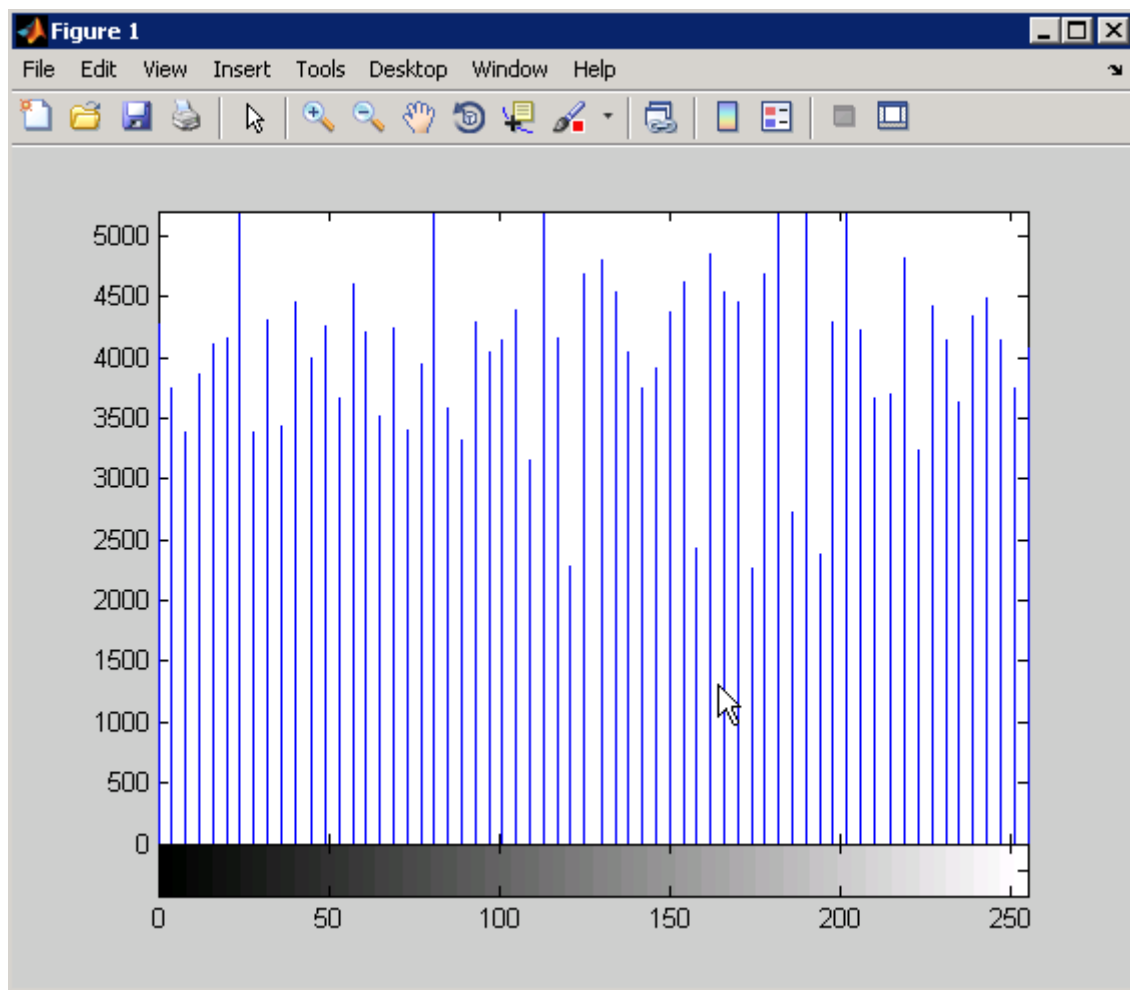
Image histogram

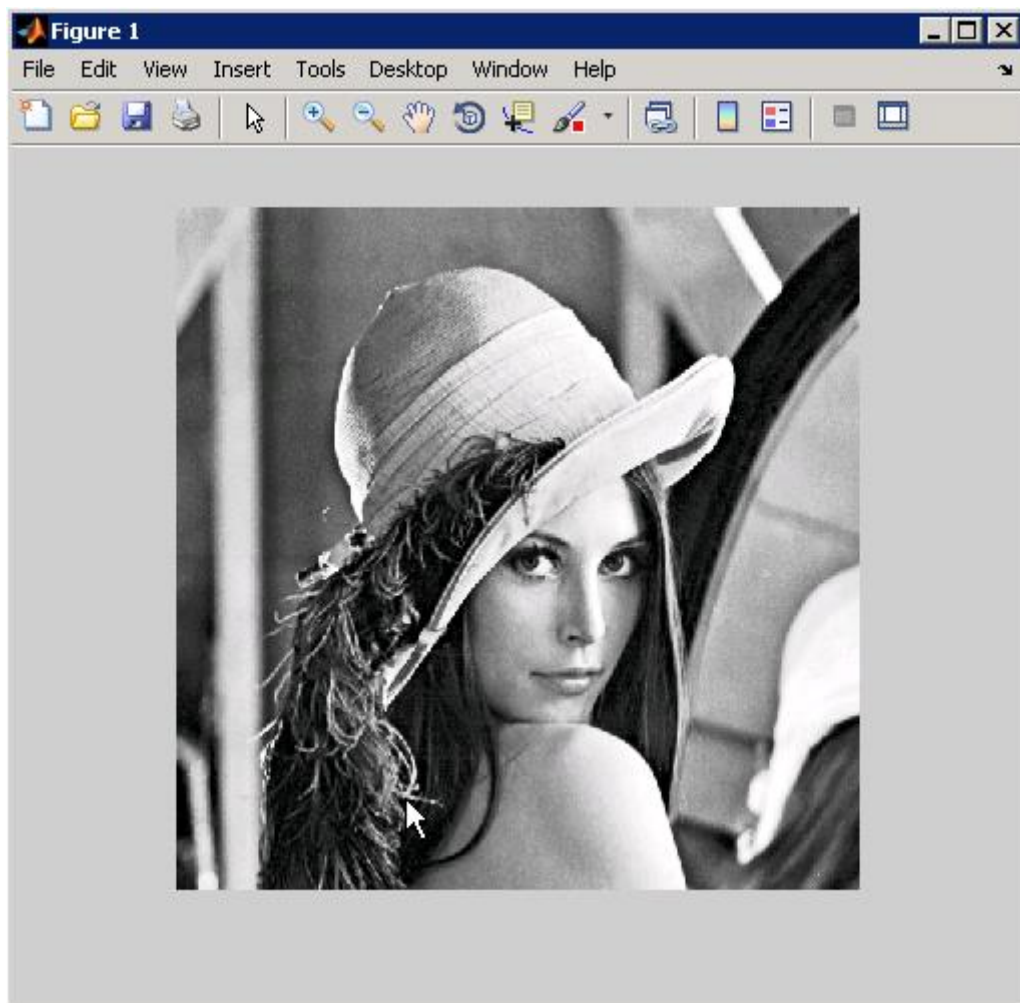
```
MATLAB Command Window
>> img=imread('c:\tmp\Lenna.png');
>> img=rgb2gray(img);
>> imhist(img)
>> figure
>> imshow(img)
Warning: Image is too big to fit on screen; displaying at 67%
> In imuitools\private\initSize at 72
   In imshow at 283
>> |
```



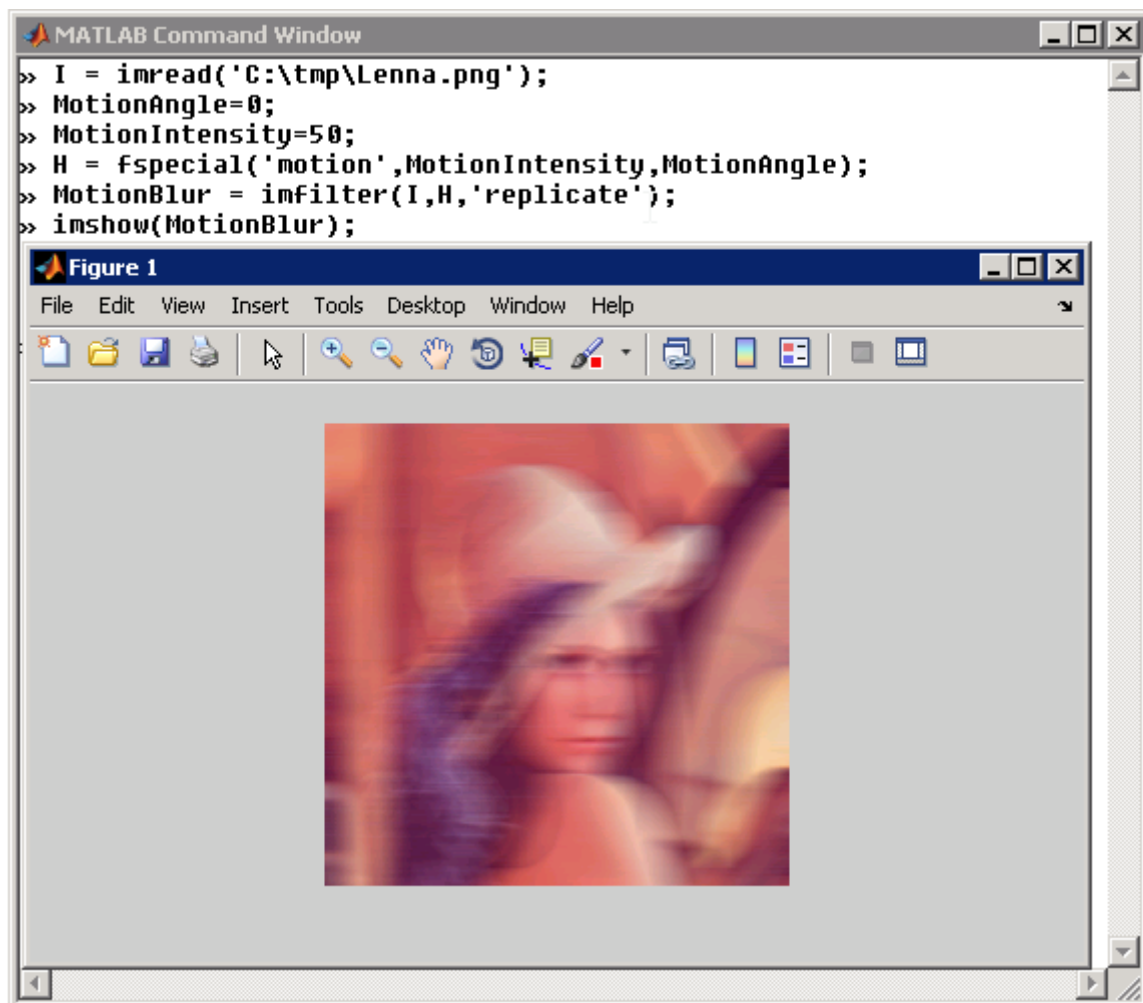
```
>> img=histeq(img);  
>> inhist(img)  
>>
```

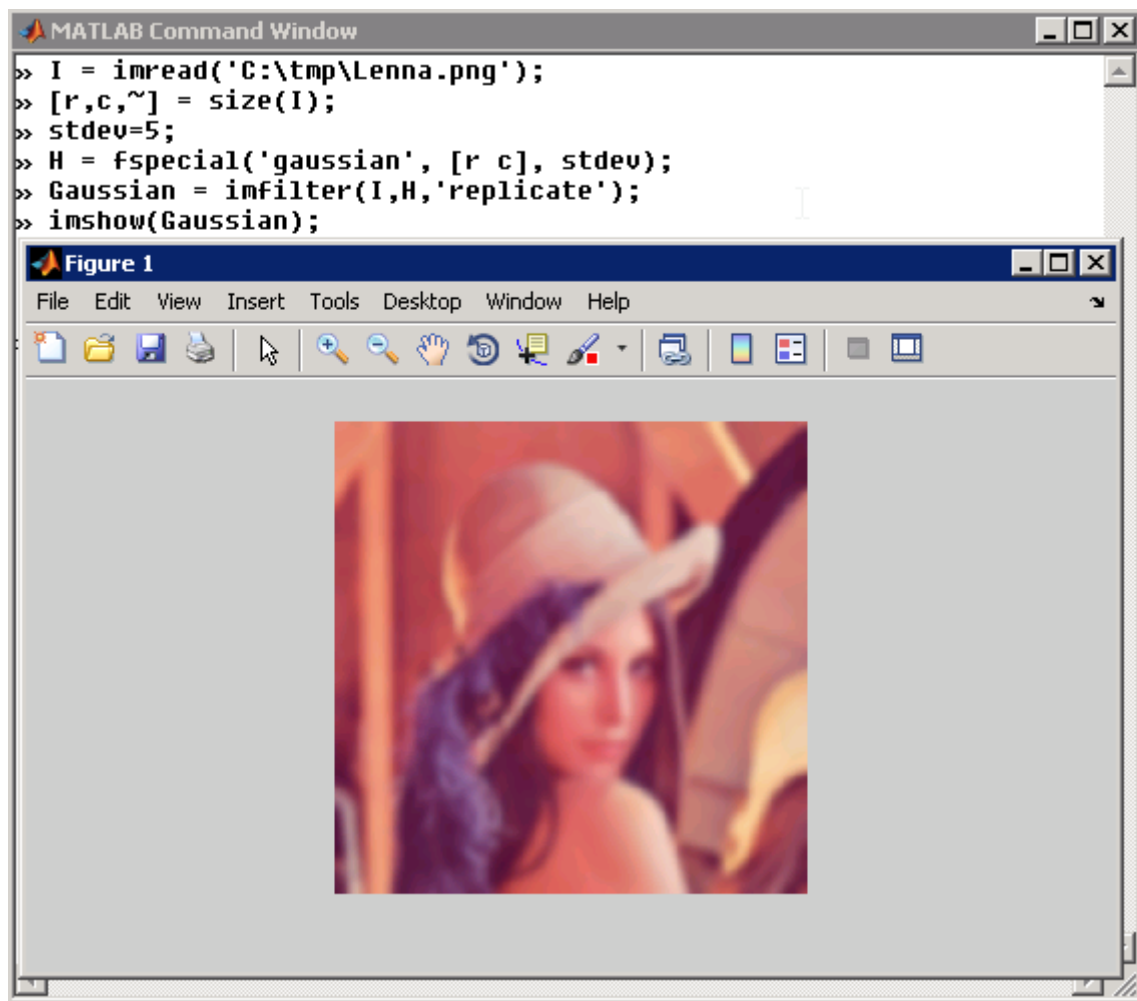


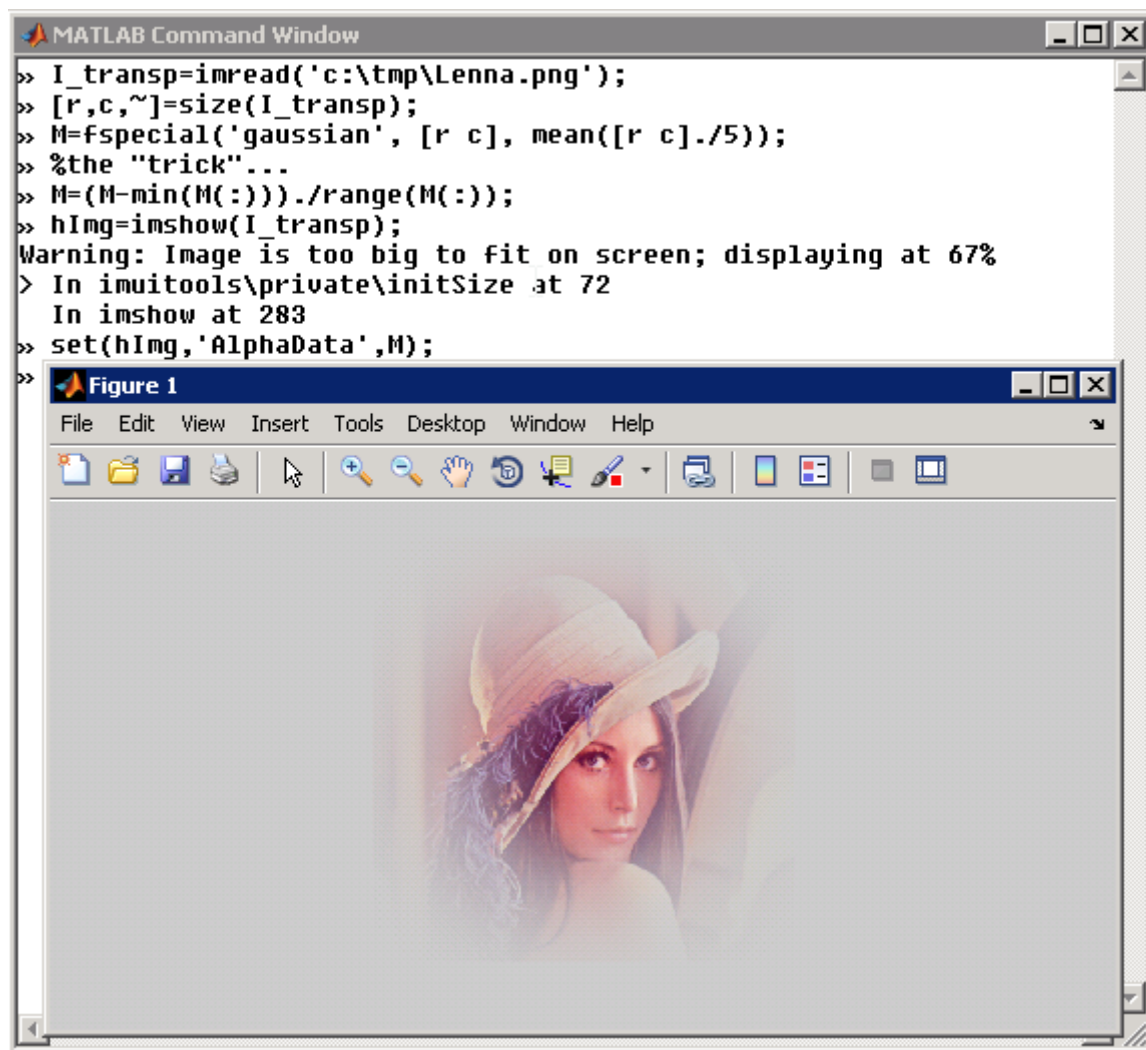


Filters (blur, motion) on an image

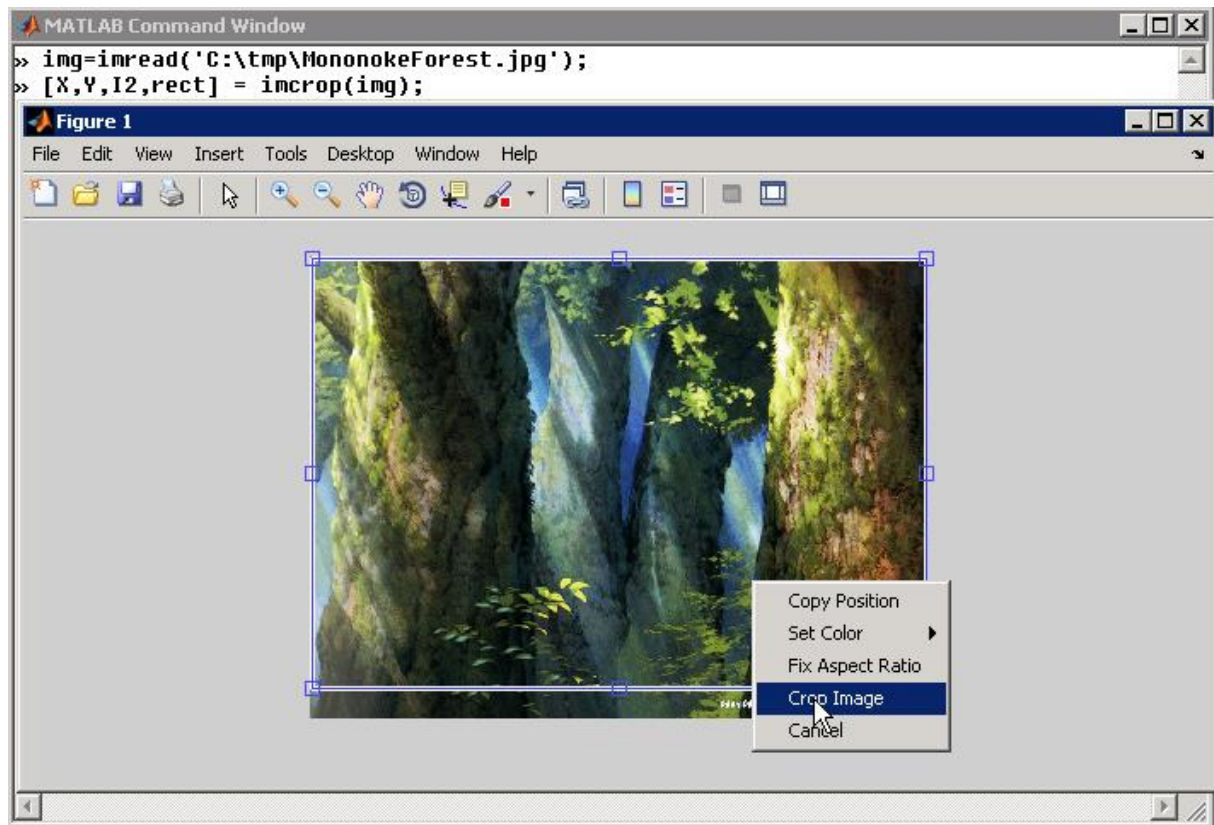
Put a filter on an image:

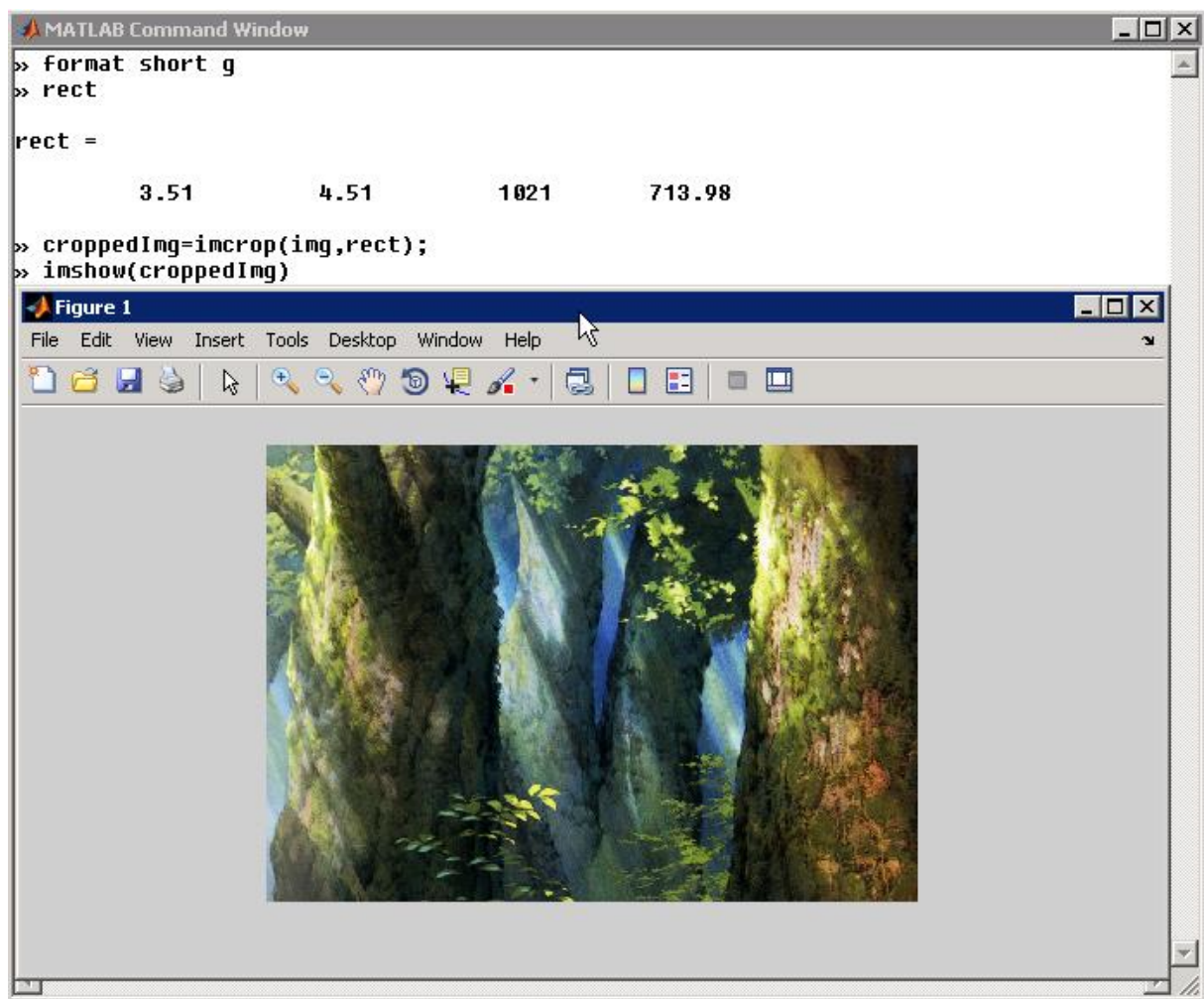






Crop an image





Convert a *.gif file into an *.avi

This is sometimes a request of some MATLAB users that have some animated chart as *.gif without having the original source code or data and have then no other choices than convert the *.gif into *.avi to have video.

Suppose we have the following animated *.gif file for the example:



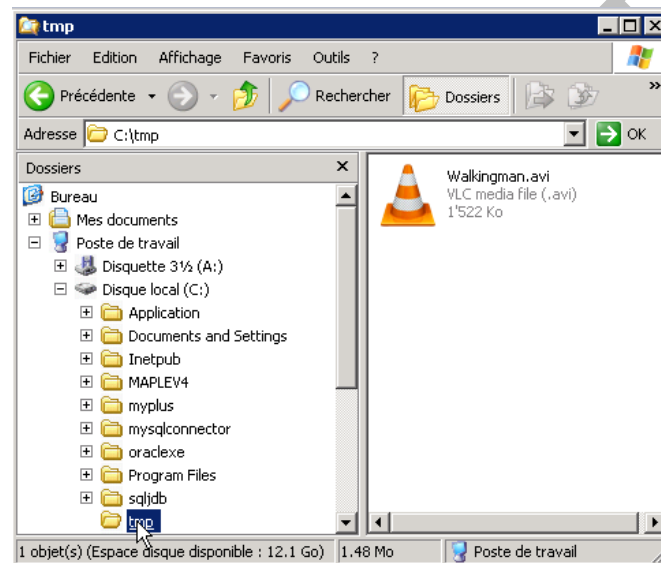
The script will use the function **immovie()**:

```

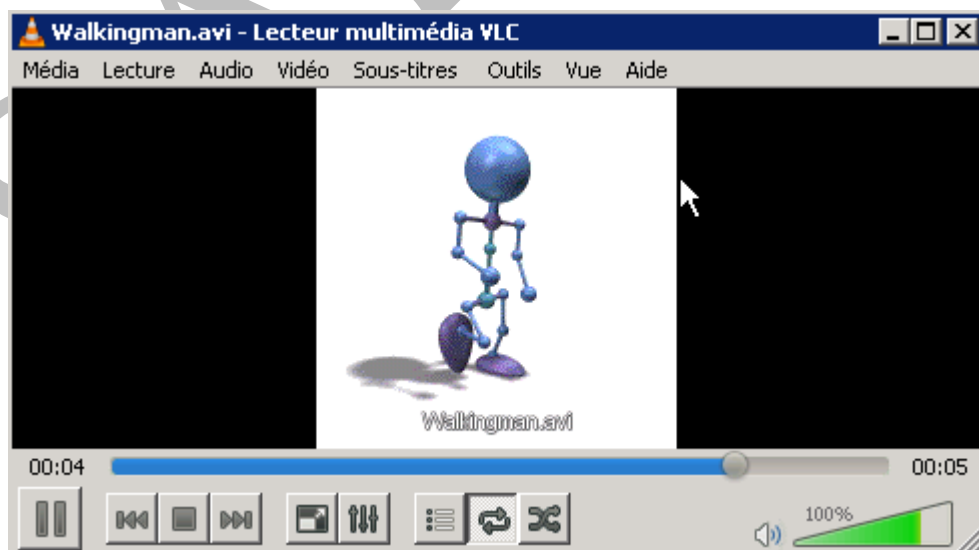
EDITOR PUBLISH VIEW
1 %convert a *.gif to a move file without compression and 3 frames per
2 %second
3 [A,map]=imread('E:\Walkingman.gif','frames','all');
4 mov=immovie(A,map);
5 movie2avi(mov,'C:\tmp\Walkingman.avi','compression','None','fps',3);

```

Then we get a new file:

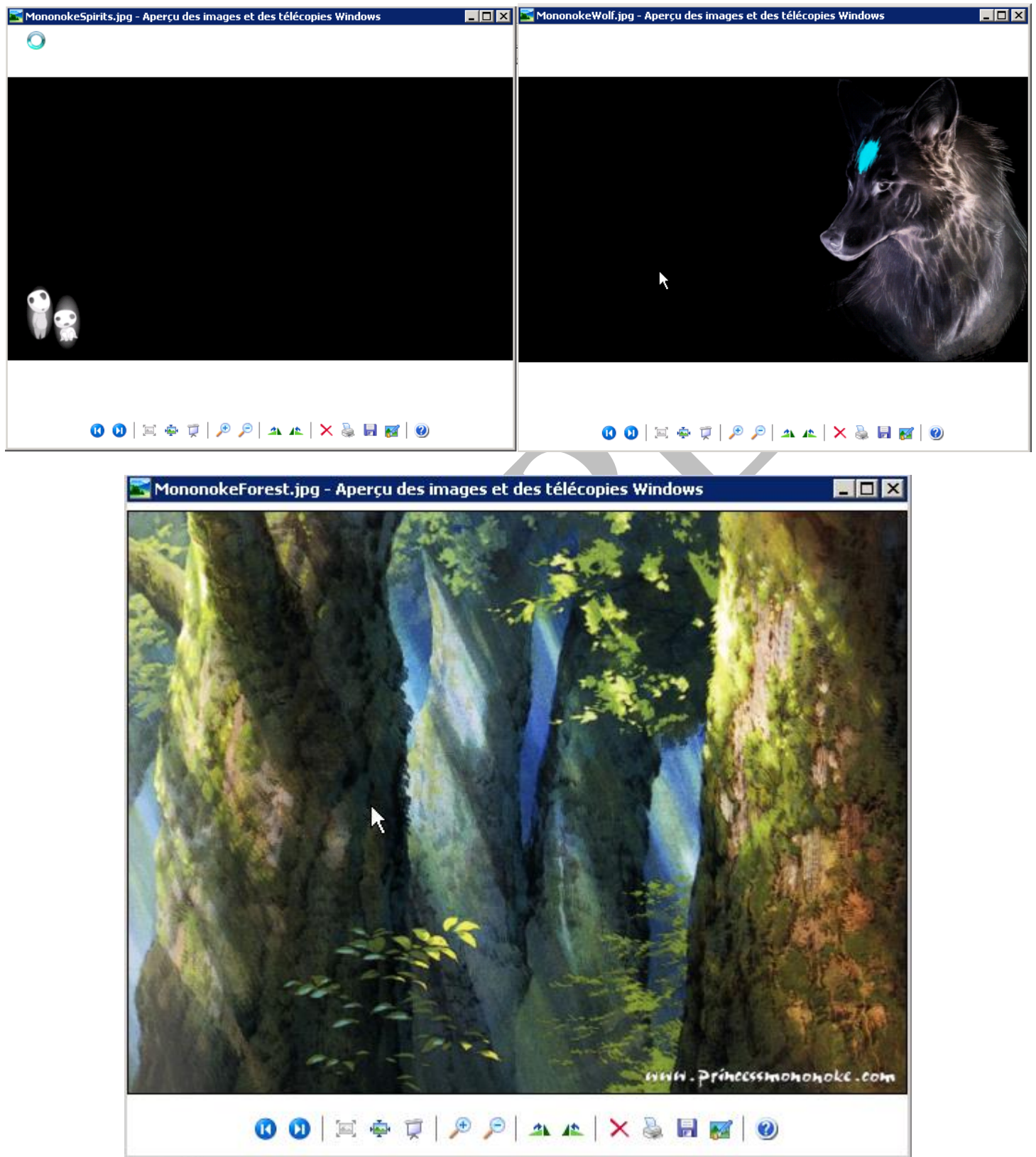


And if we play it:

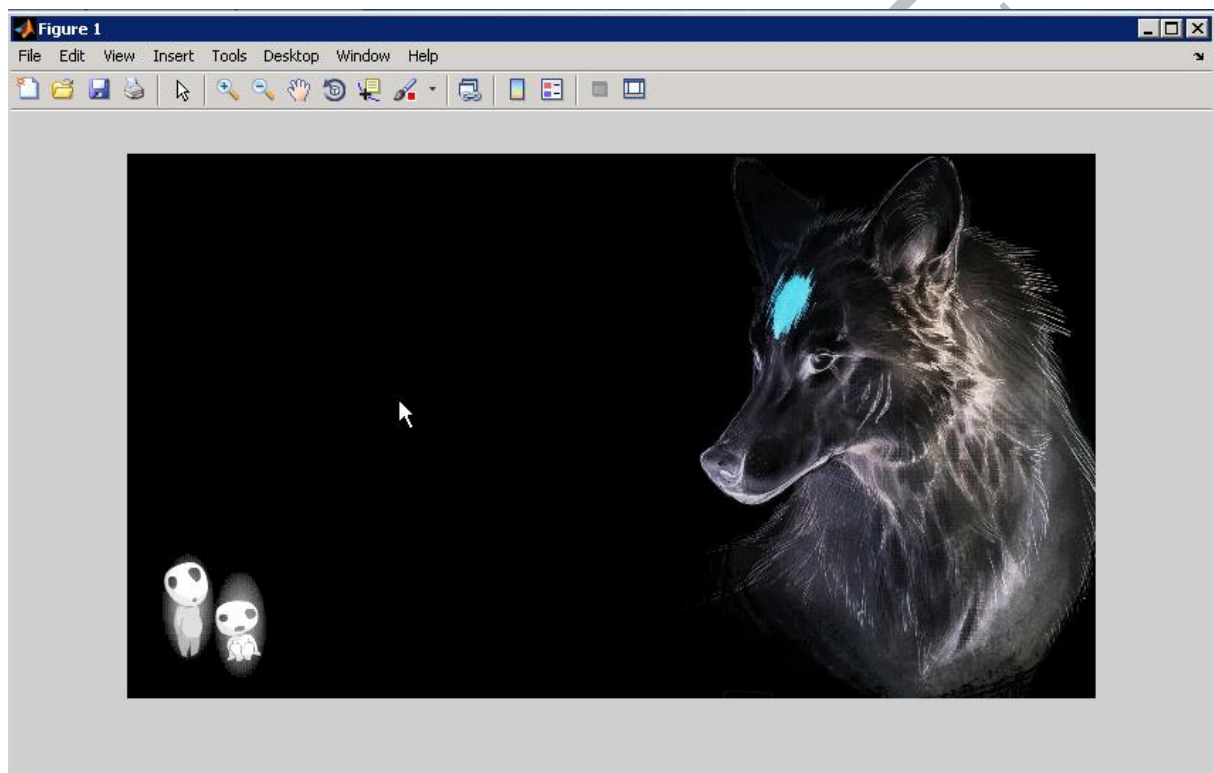


Job done!

Overlay images with masks



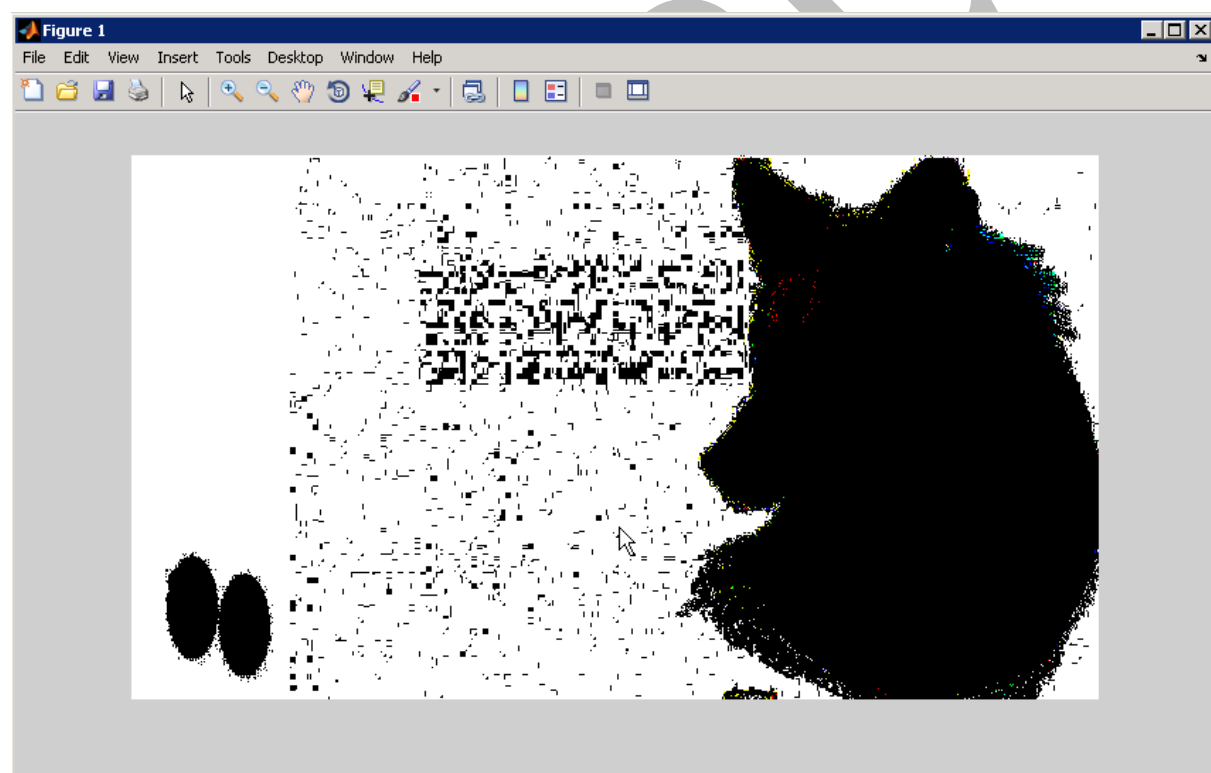
```
EDITOR PUBLISH VIEW
1 - background=imread('C:\tmp\MononokeForest.jpg');
2 - imgSpirits=imread('C:\tmp\MononokeSpirits.jpg');
3 - imgWolf=imread('C:\tmp\MononokeWolf.jpg');
4
5 %Image addition
6 %Both A and B are of same size
7 - imgOver=imgSpirits+imgWolf;
8 - imshow(imgOver)|
```



```

EDITOR PUBLISH VIEW
1 - close all;clear all;clc
2 - background=imread('C:\tmp\MononokeForest.jpg');
3 - imgSpirits=imread('C:\tmp\MononokeSpirits.jpg');
4 - imgWolf=imread('C:\tmp\MononokeWolf.jpg');
5
6 %Image addition
7 %Both A and B are of same size
8 - imgOver=imgSpirits+imgWolf;
9
10 %we create the mask
11 - background=imresize(background,[size(imgOver,1) size(imgOver,2)]);
12 - Im3=uint8(zeros(size(imgOver)));
13 - whiteImg=uint8(ones(size(imgOver)));
14 - mask=whiteImg./imgOver;
15 - im3=uint8(mask&imgOver);%uint8(and(mask,background));
16 - imshow(mask)

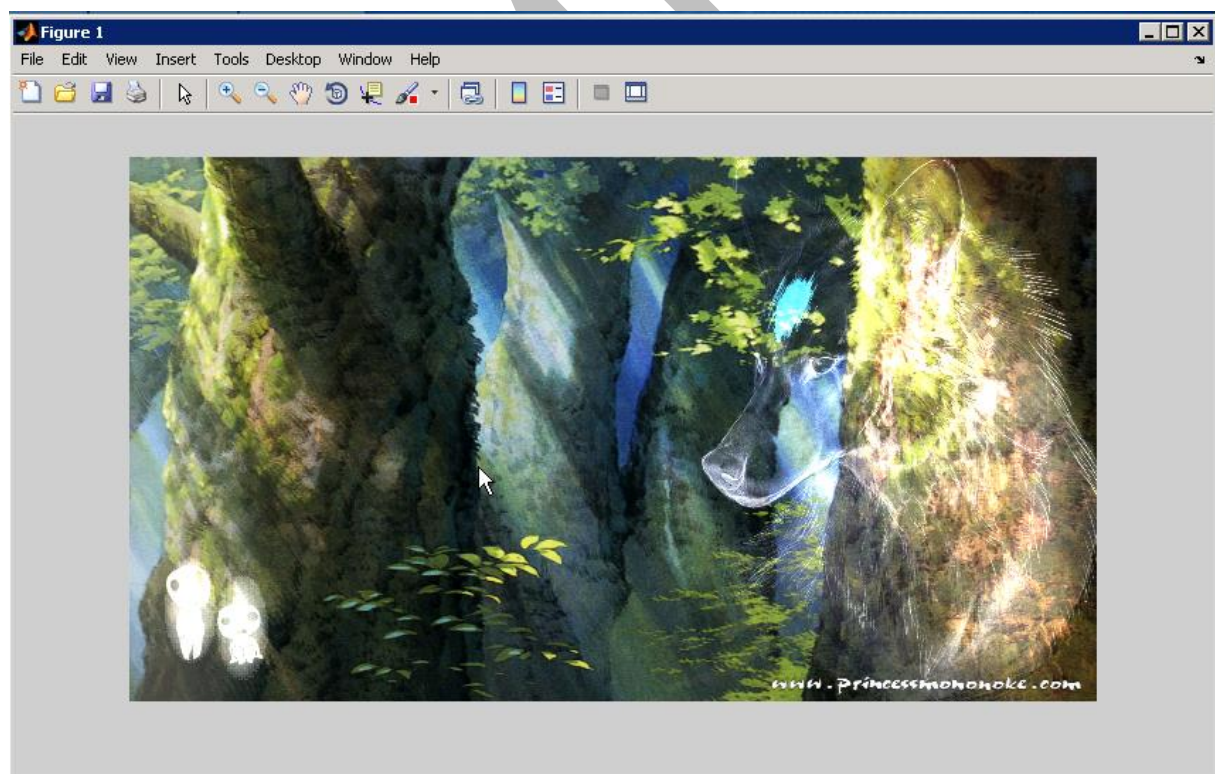
```



```

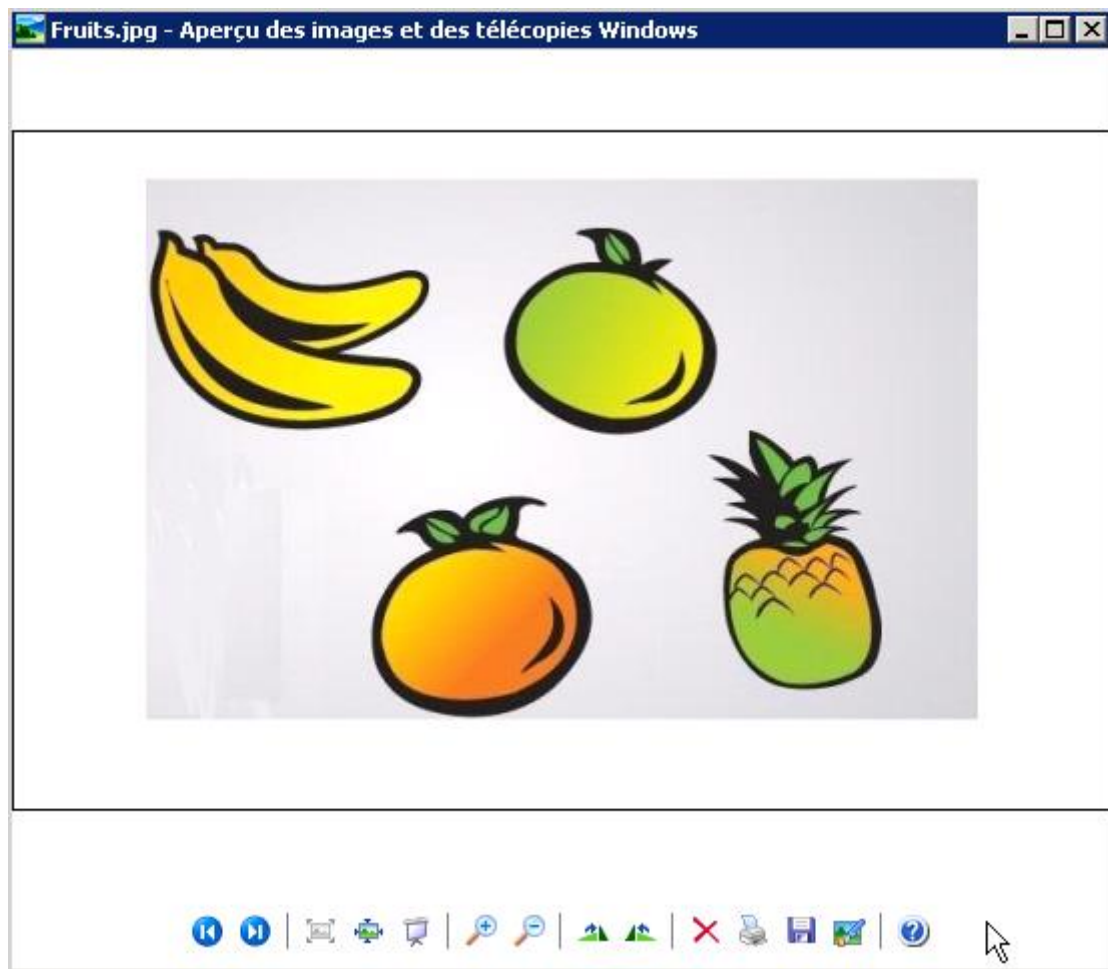
1 - close all;clear all;clc
2 - background=imread('C:\tmp\MononokeForest.jpg');
3 - imgSpirits=imread('C:\tmp\MononokeSpirits.jpg');
4 - imgWolf=imread('C:\tmp\MononokeWolf.jpg');
5
6 %Image addition
7 %Both A and B are of same size
8 - imgOver=imgSpirits+imgWolf;
9
10 %we create the mask
11 - background=imresize(background,[size(imgOver,1) size(imgOver,2)]);
12 - Im3=uint8(zeros(size(imgOver)));
13 - whiteImg=uint8(ones(size(imgOver)));
14 - mask=whiteImg./imgOver;
15 - im3=uint8(mask&imgOver);%uint8(and(mask,background));
16
17 %we add the two images
18 - finalImg=(background)+imgOver;
19 - imshow(finalImg);

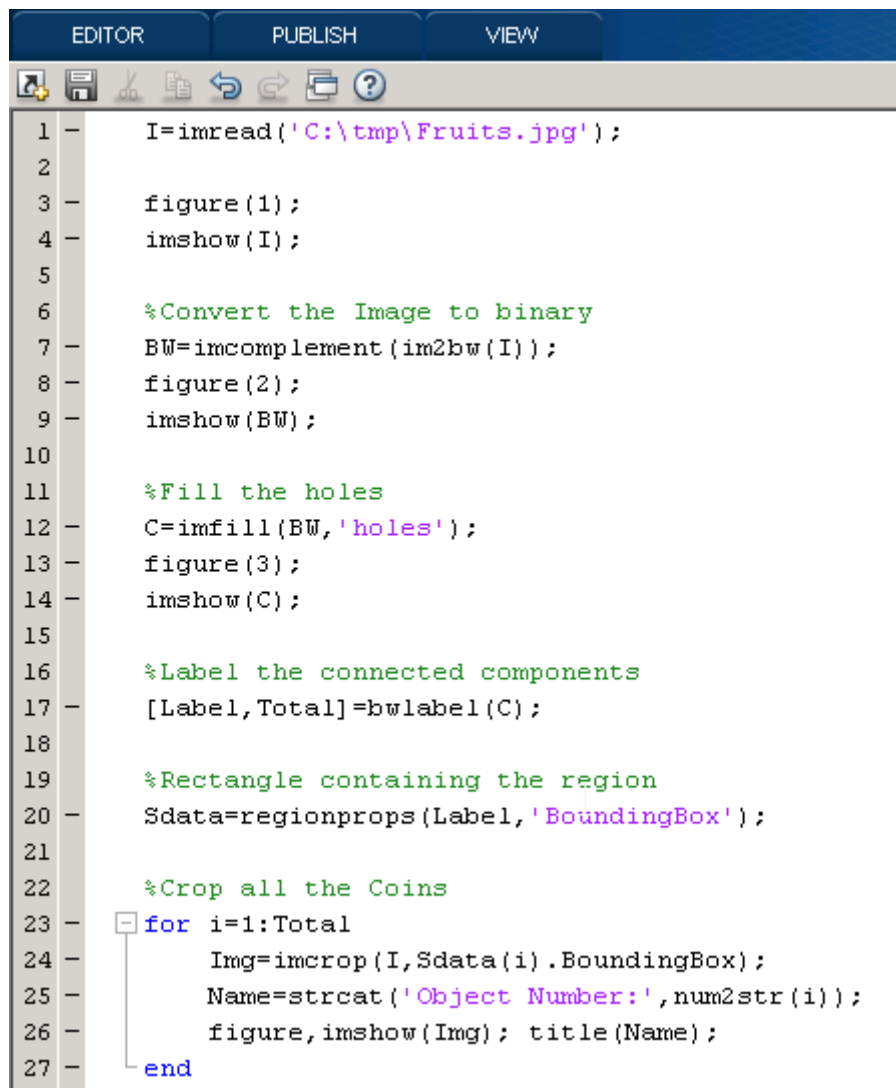
```



Objects recognition and extraction

Without references

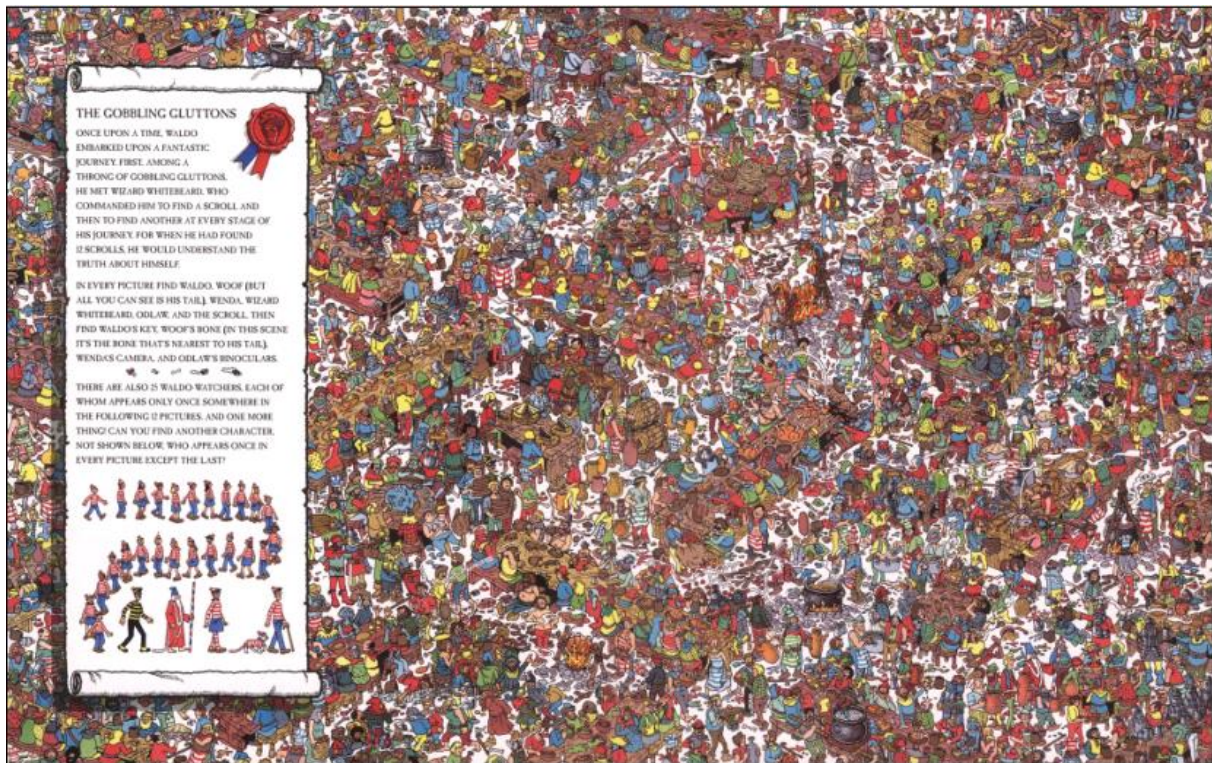




The image shows a screenshot of the MATLAB Editor window. The window has a dark blue header with three tabs: 'EDITOR', 'PUBLISH', and 'VIEW'. Below the header is a toolbar with icons for file operations (new, open, save, print), editing (undo, redo, cut, copy, paste), and help. The main area displays a MATLAB script with line numbers 1 through 27. The script performs the following steps: 1. Read an image 'Fruits.jpg' from the path 'C:\tmp\'. 2. Display the image in a figure window. 3. Convert the image to binary using 'im2bw' and create its complement 'BW'. 4. Display 'BW'. 5. Fill the holes in 'BW' using 'imfill' with the 'holes' option, resulting in 'C'. 6. Display 'C'. 7. Label the connected components of 'C' using 'bwlabel', returning 'Label' and 'Total'. 8. Use 'regionprops' to find the bounding boxes for each region, storing them in 'Sdata'. 9. Loop through each region (from 1 to 'Total') to crop the original image 'I' using the bounding box, resulting in 'Img'. 10. Create a title for each crop: 'Object Number: ' followed by the region index. 11. Display each crop 'Img' in a new figure window with the corresponding title. The script is syntactically correct and follows standard MATLAB conventions for image processing.

```
1 - I=imread('C:\tmp\Fruits.jpg');
2
3 - figure(1);
4 - imshow(I);
5
6 %Convert the Image to binary
7 - BW=imcomplement(im2bw(I));
8 - figure(2);
9 - imshow(BW);
10
11 %Fill the holes
12 - C=imfill(BW,'holes');
13 - figure(3);
14 - imshow(C);
15
16 %Label the connected components
17 - [Label,Total]=bwlabel(C);
18
19 %Rectangle containing the region
20 - Sdata=regionprops(Label,'BoundingBox');
21
22 %Crop all the Coins
23 - for i=1:Total
24 -     Img=imcrop(I,Sdata(i).BoundingBox);
25 -     Name=strcat('Object Number:',num2str(i));
26 -     figure,imshow(Img); title(Name);
27 - end
```

With reference

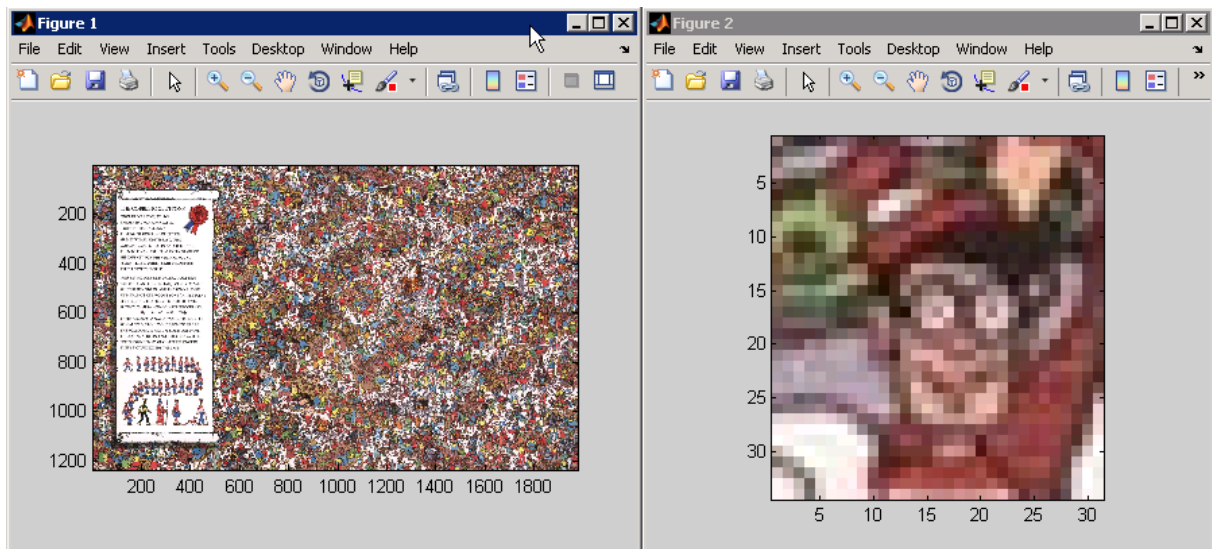


EDITOR PUBLISH VIEW

```

1 - clear all;close all;clc
2 - template = importdata('C:\tmp\Waldo.png');
3 - im = importdata('C:\tmp\FindWaldo.jpg');
4
5 - %we show the images
6 - imagesc(im); axis image;
7 - figure;
8 - imagesc(template); axis image;
9

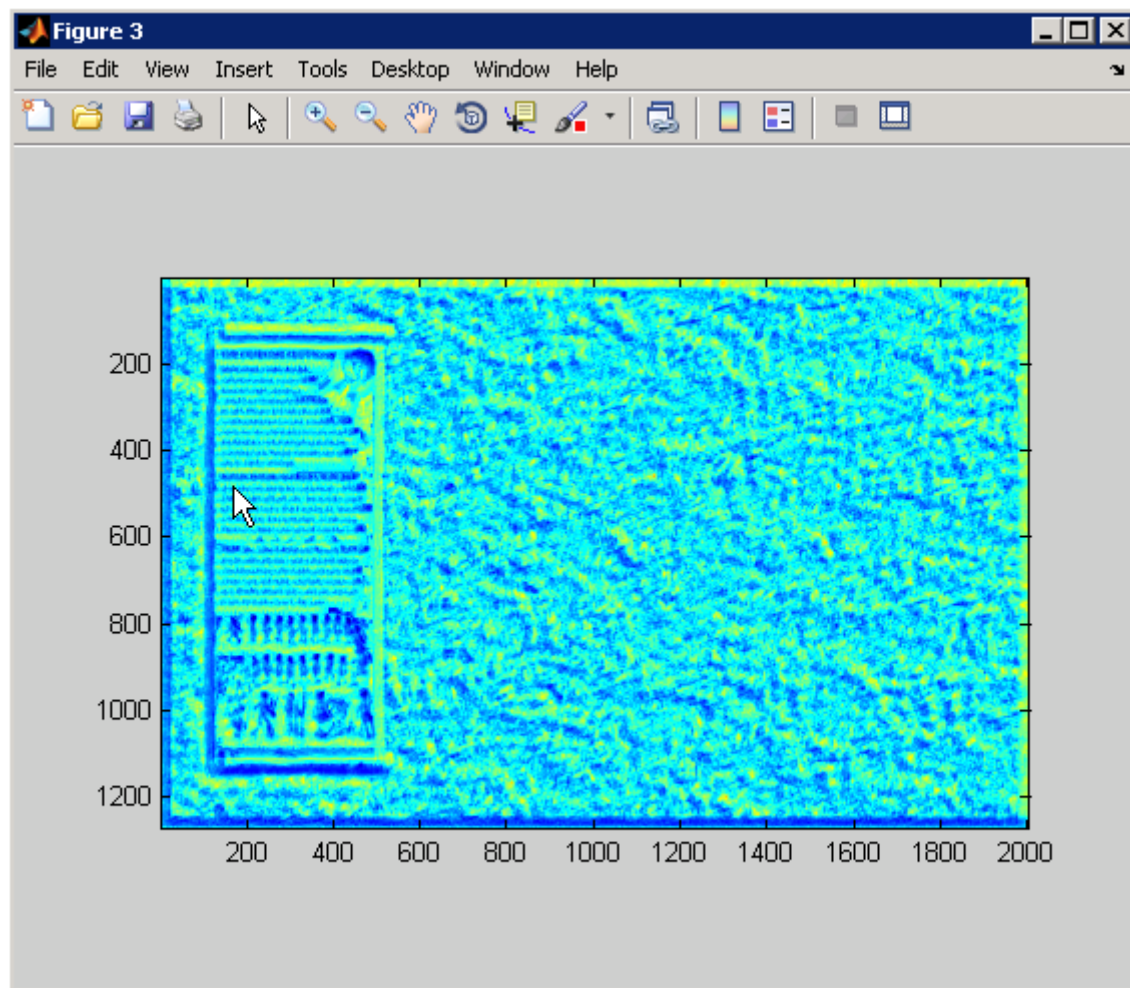
```

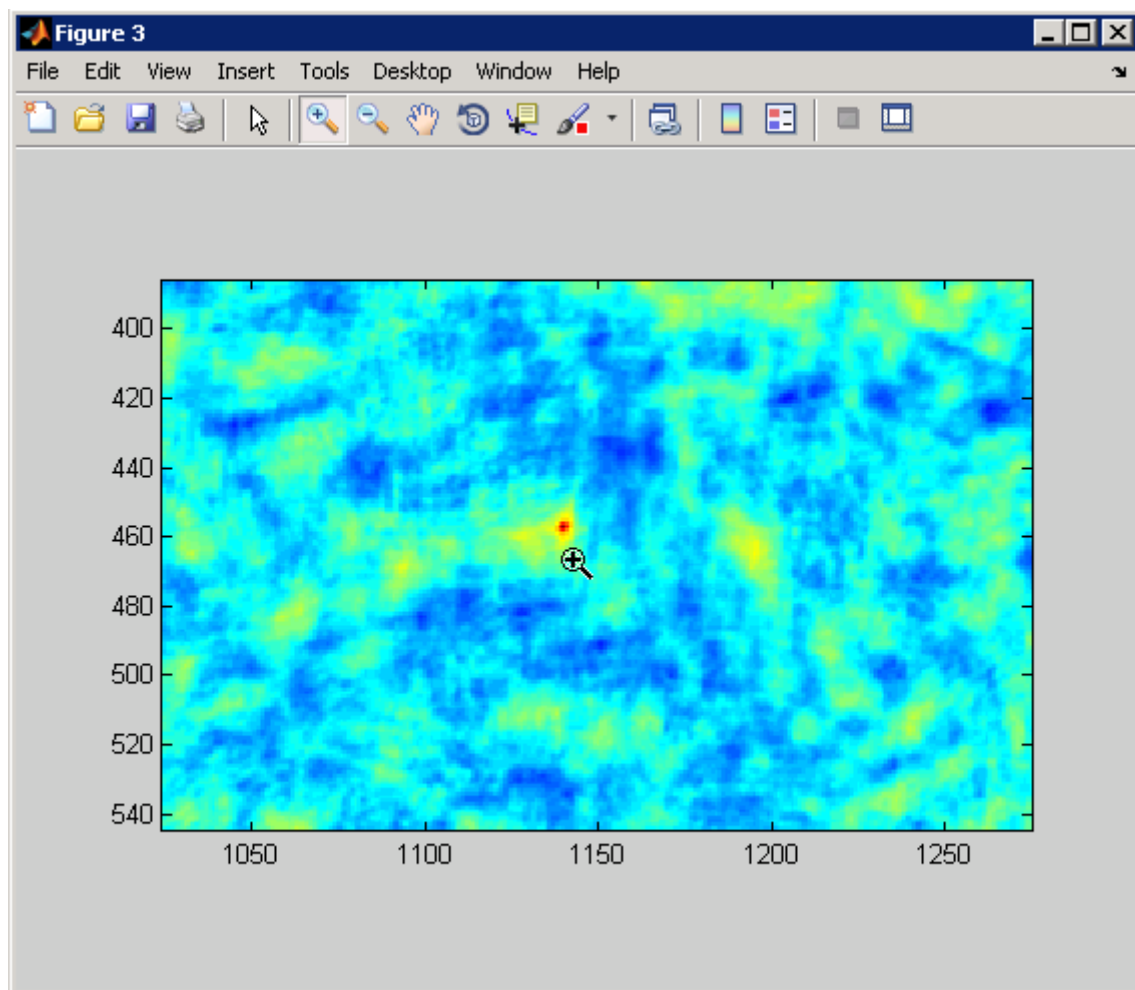
```

9
10 %we convert in grayscale
11 - template = mean(template, 3);
12 - im = mean(im, 3);
13
14 %search 2D correlation using the normxcorr2 function
15 %of the image processing toolbox
16 - figure;
17 - W = normxcorr2(template, im);
18 - imagesc(W); axis image; colormap('jet');
19
20 - W1 = W(:);
21 - size(W1);
22 % we search the max correlation
23 % I gives the number of pixel (we will get the coordinates later with
24 % ind2sub() function
25 - [C, I] = max(W1);

```



If we zoom:



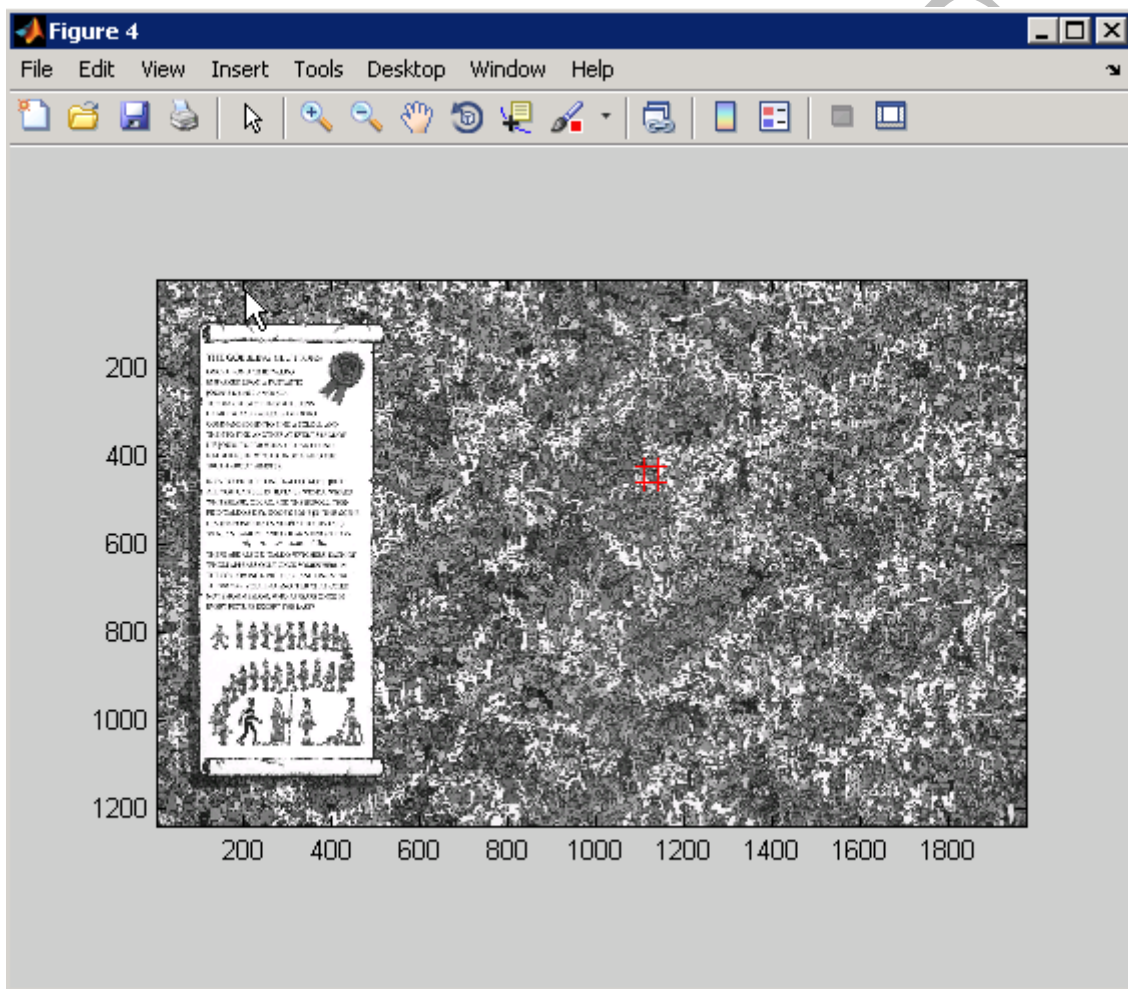
Command Window

```
>> C  
  
C =  
  
1.0000  
  
>> I  
  
I =  
  
1450404  
  
fx >> |
```

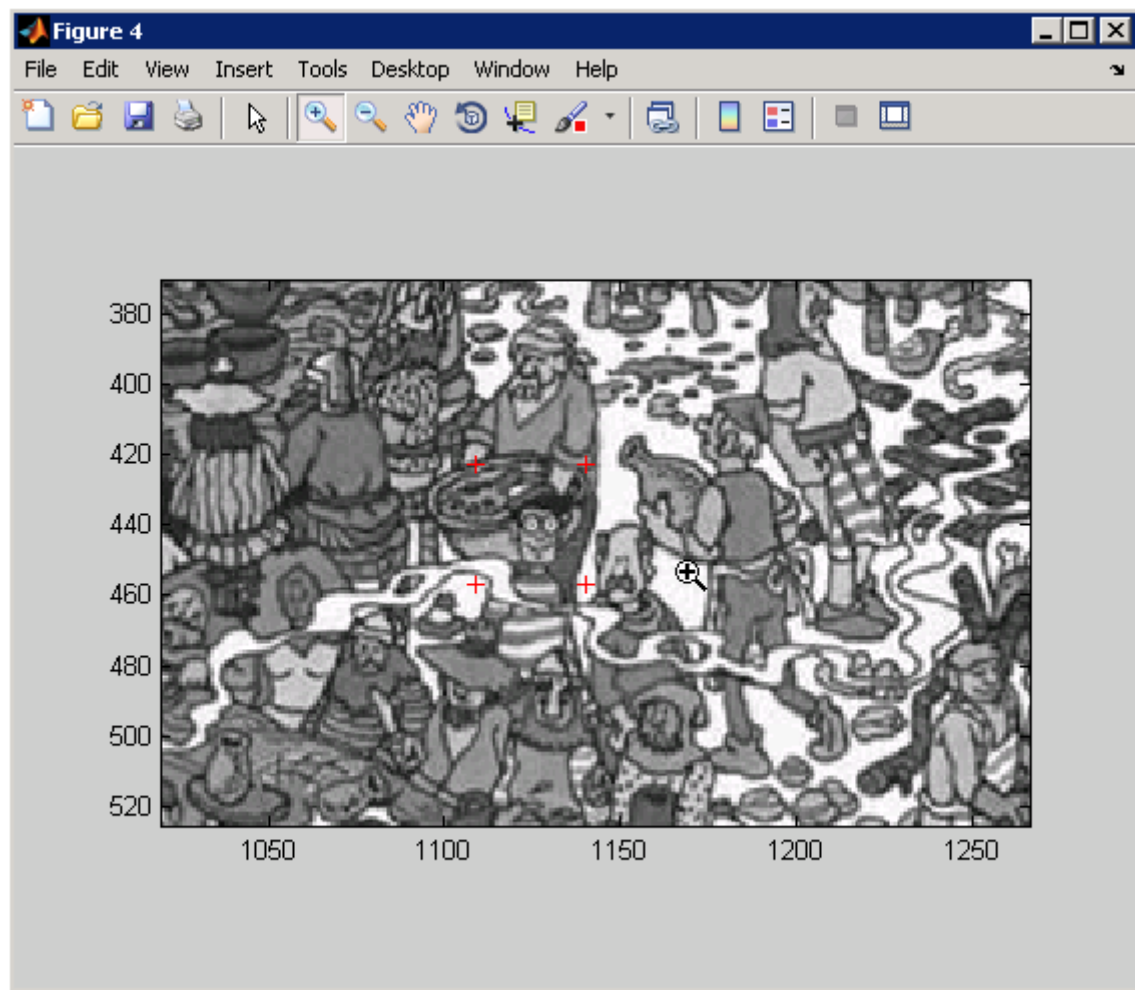
```

26
27 - figure;
28 %we prepare the image to show where is Waldo
29 - imagesc(im); axis image; colormap('gray');
30 - hold on;
31
32 - [y, x] = ind2sub(size(W), I)
33 - scatter(x, y, 'r+');
34 - scatter(x - size(template, 2), y, 'r+');
35 - scatter(x, y - size(template, 1), 'r+');
36 - scatter(x - size(template, 2), y - size(template, 1), 'r+');

```



if we zoom:



```
Command Window

y =
    457

x =
    1140

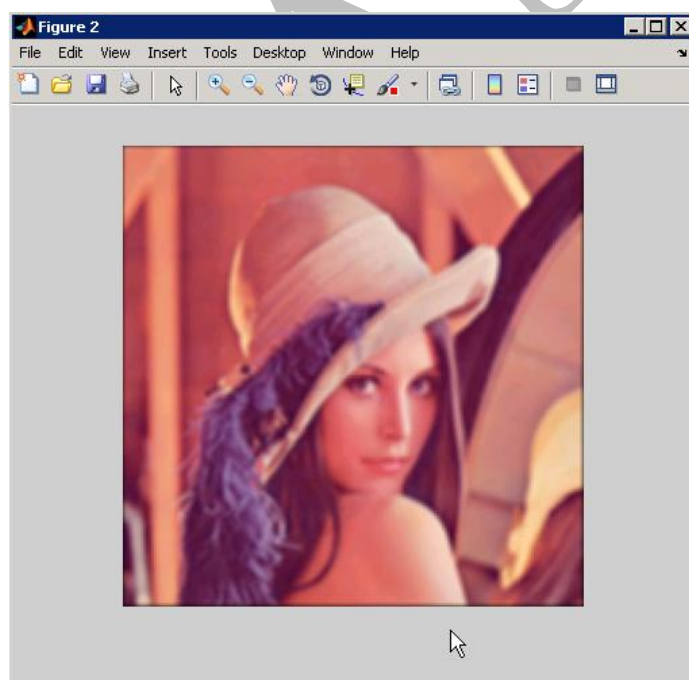
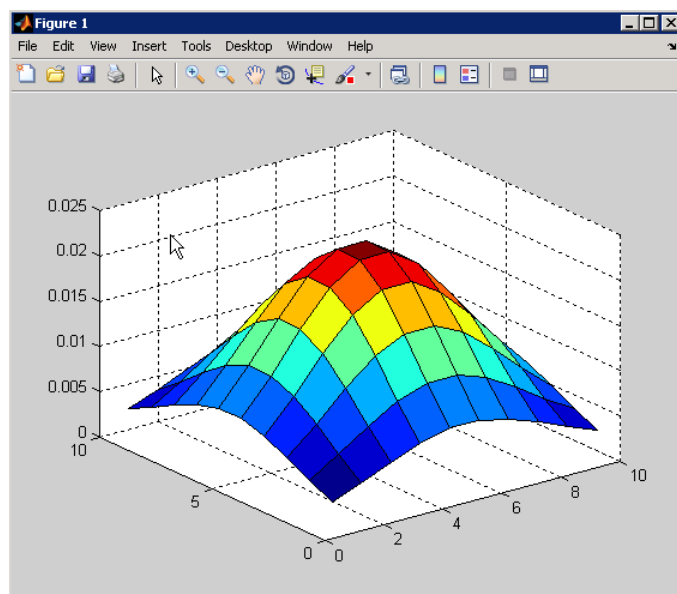
fx >> |
```

Image filters (convolution)

Gaussian (blur) filter

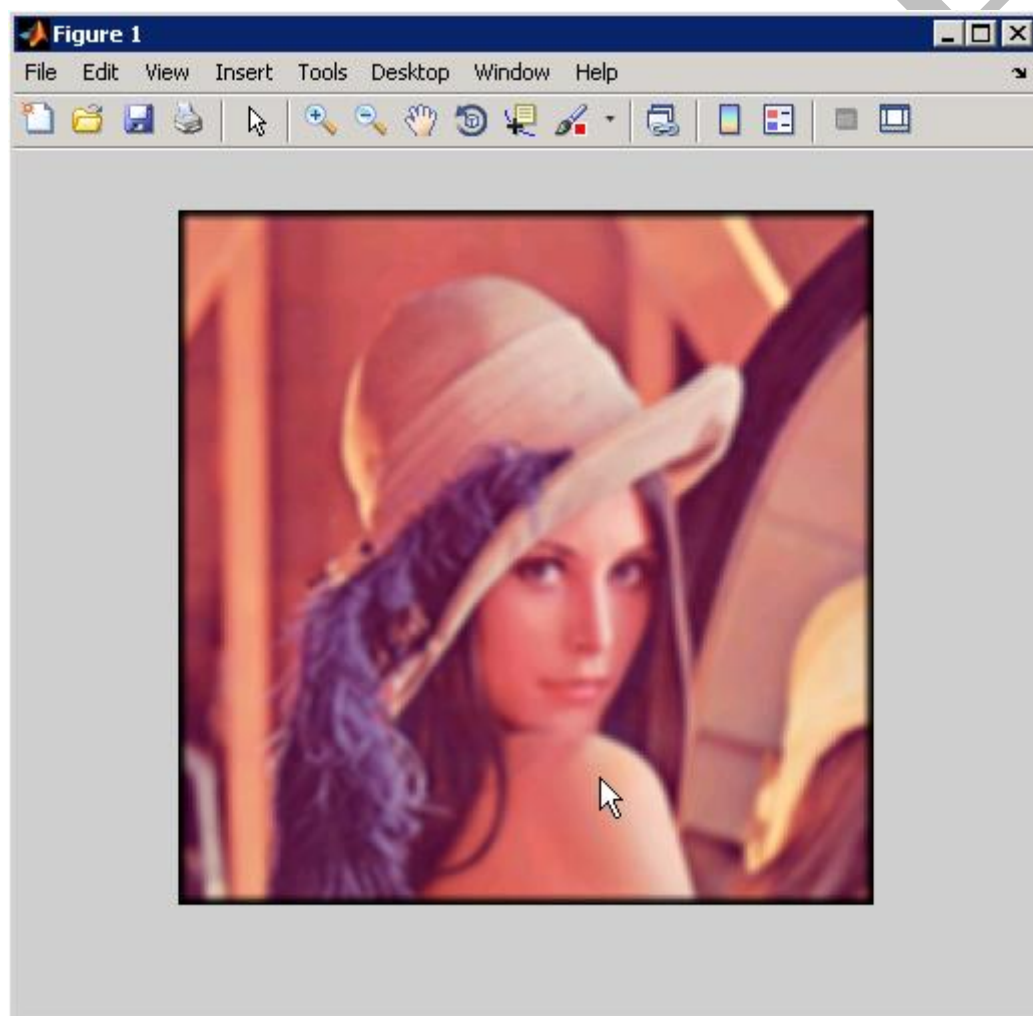
1st method using imfilter()


```
EDITOR PUBLISH VIEW
1 - clear all;close all;clc
2 - img=imread('C:/tmp/Lenna.png');
3 - img=im2double(img);
4 - matrix_conv_size=10;
5 - sigma=3;
6 - filter=fspecial('gaussian',matrix_conv_size,sigma);
7 - surf(filter);
8 - conv=imfilter(img,filter);
9 - figure
10 - imshow(conv)
```



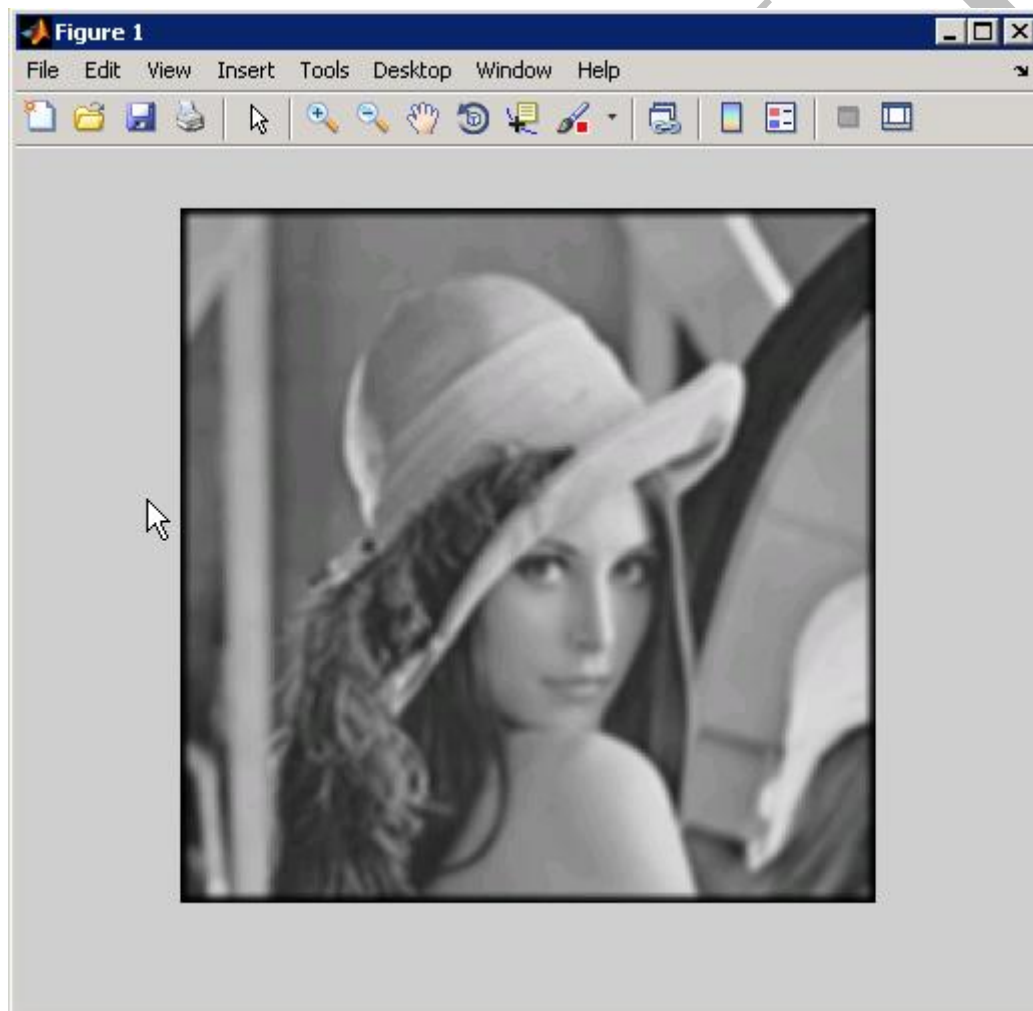
2nd method using convn()

```
EDITOR PUBLISH VIEW
1 %Color images are 3 dimensional arrays (x,y,color).
2 %conv2() is only defined for 2-dimensions, so it won't work directly on a 3D array.
3 %Solution: use convn
4 clear all;close all;clc
5 - img=imread('C:/tmp/Lenna.png');
6 - img=im2double(img);
7 - matrix_conv_size=10;
8 - sigma=3;
9 - filter=fspecial('gaussian',matrix_conv_size,sigma);
10 conv=convn(img,filter);
11 - imshow(conv)
```

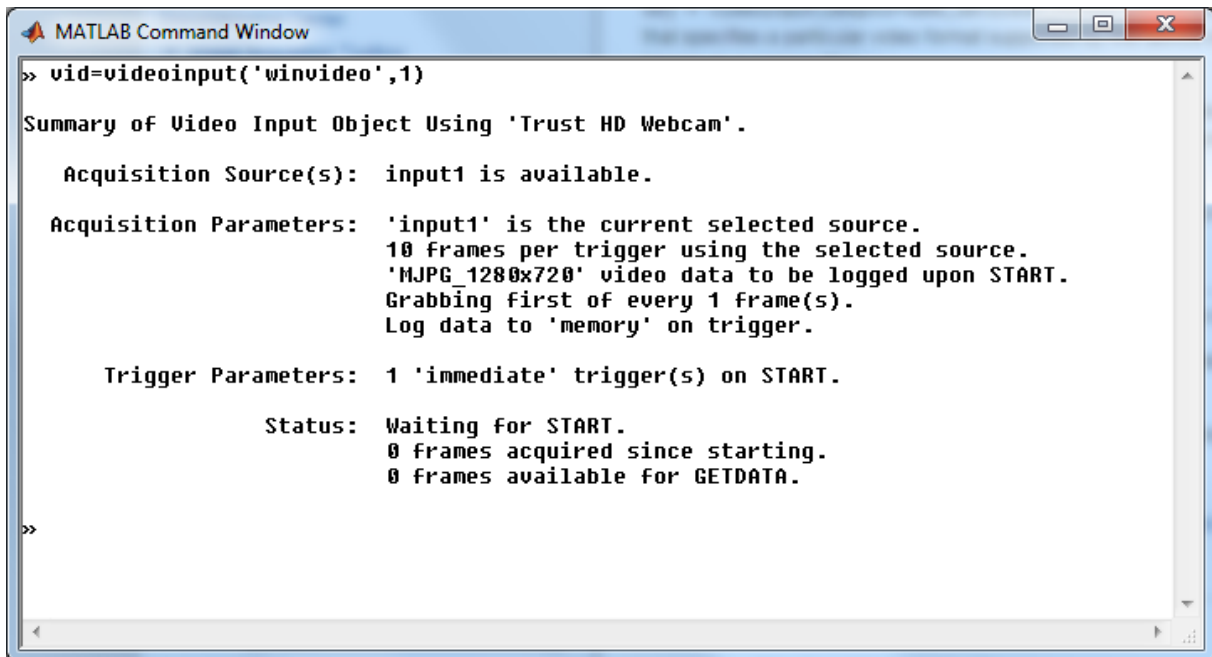


3rd method using conv2()

```
EDITOR PUBLISH VIEW
1 %conv2() is only defined for 2-dimensions so we have to work
2 %on a grayscale image
3 - clear all;close all;clc
4 - img=imread('C:/tmp/Lenna.png');
5 - img=rgb2gray(img); %img=uint8(repmat(mean(img,3),[1,1,3]));
6 - matrix_conv_size=10;
7 - sigma=3;
8 - filter=fspecial('gaussian',matrix_conv_size,sigma);
9 - conv=conv2(img,filter);
10 - imshow(uint8(conv));|
```



External Camera Acquisition



A screenshot of the MATLAB Command Window. The title bar reads 'MATLAB Command Window'. The command entered is `>> vid=videoinput('winvideo',1)`. Below the command, a summary of the video input object is displayed. The summary includes acquisition source information, parameters for frame rate, resolution, and triggering, and the current status of the acquisition.

```
>> vid=videoinput('winvideo',1)

Summary of Video Input Object Using 'Trust HD Webcam'.

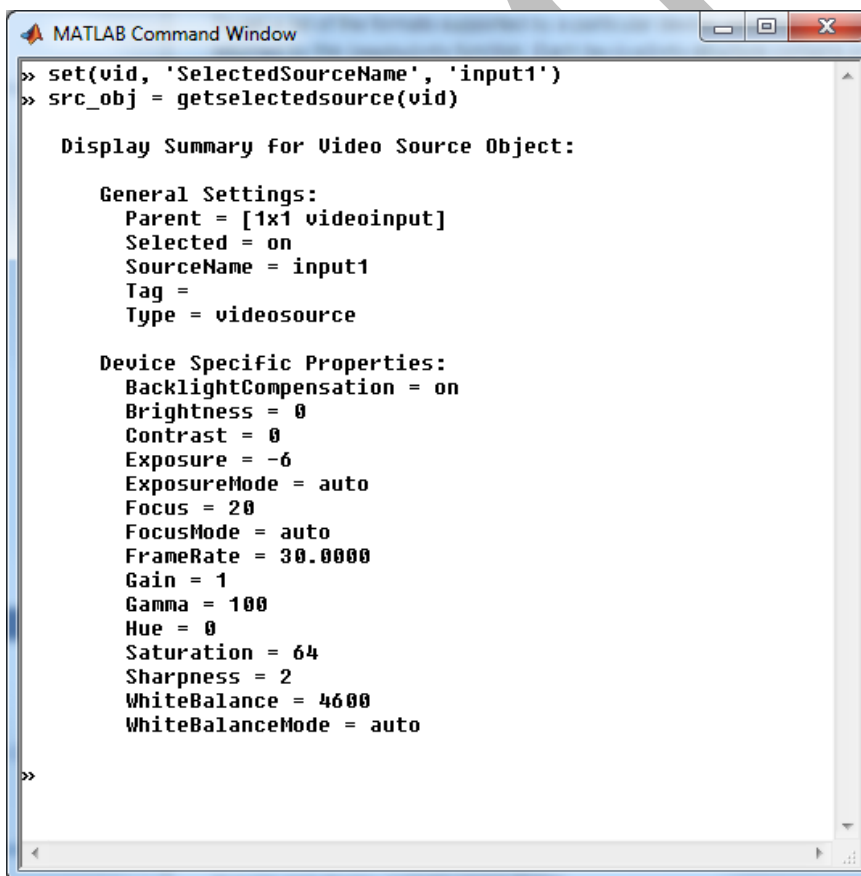
Acquisition Source(s):  input1 is available.

Acquisition Parameters:  'input1' is the current selected source.
                        10 frames per trigger using the selected source.
                        'MJPG_1280x720' video data to be logged upon START.
                        Grabbing first of every 1 frame(s).
                        Log data to 'memory' on trigger.

Trigger Parameters:  1 'immediate' trigger(s) on START.

Status:  Waiting for START.
         0 frames acquired since starting.
         0 frames available for GETDATA.

>>
```



A screenshot of the MATLAB Command Window. The title bar reads 'MATLAB Command Window'. Two commands are entered: `>> set(vid, 'SelectedSourceName', 'input1')` and `>> src_obj = getselectedsource(vid)`. Below the commands, a detailed summary of the video source object's settings is displayed, categorized into general settings and device-specific properties.

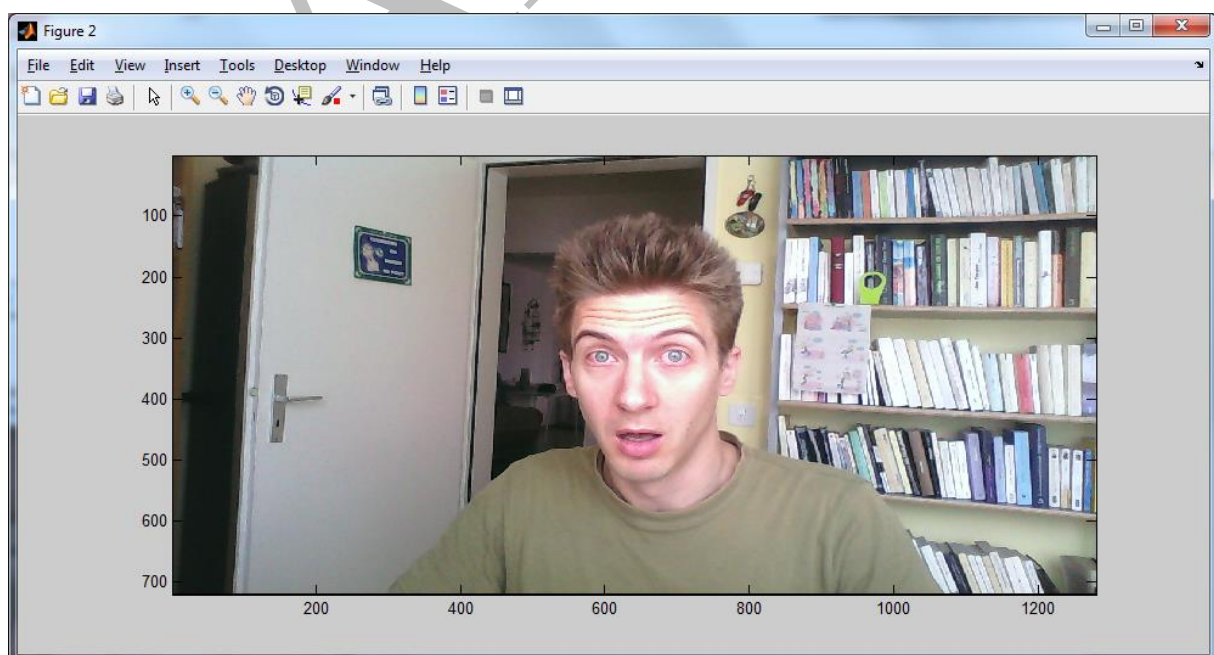
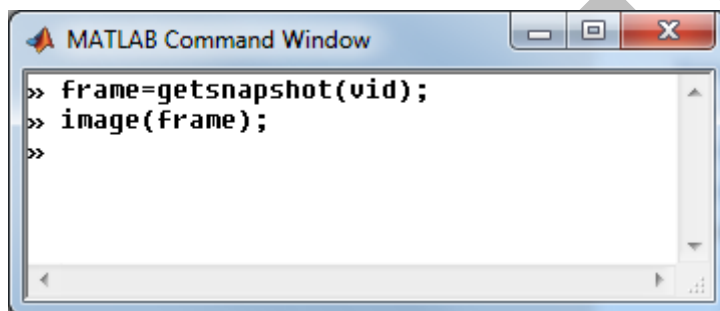
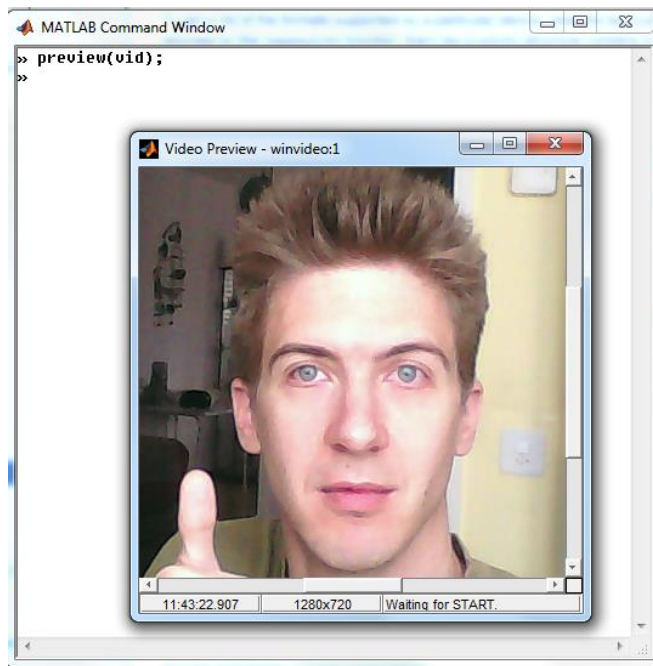
```
>> set(vid, 'SelectedSourceName', 'input1')
>> src_obj = getselectedsource(vid)

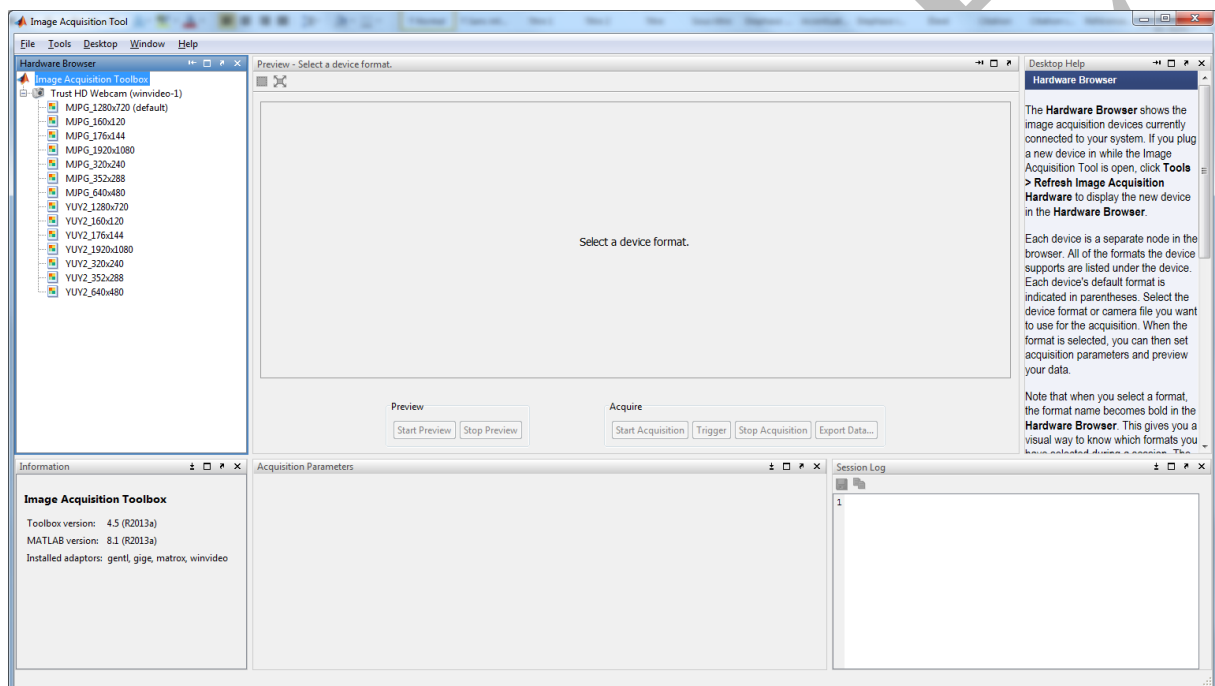
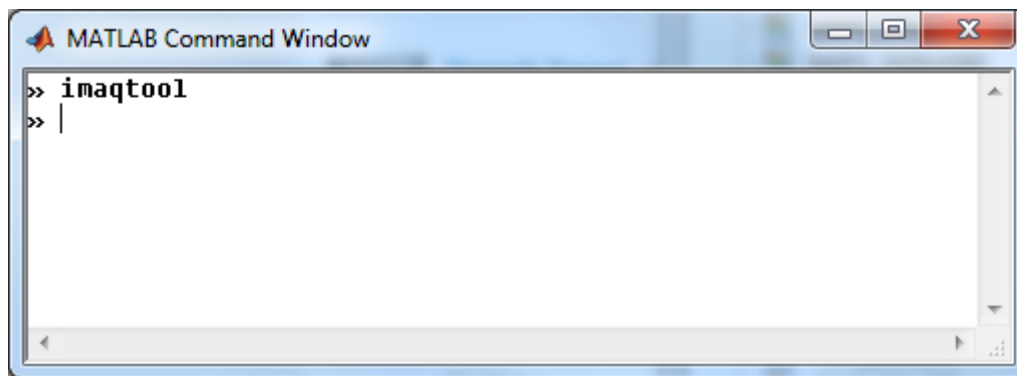
Display Summary for Video Source Object:

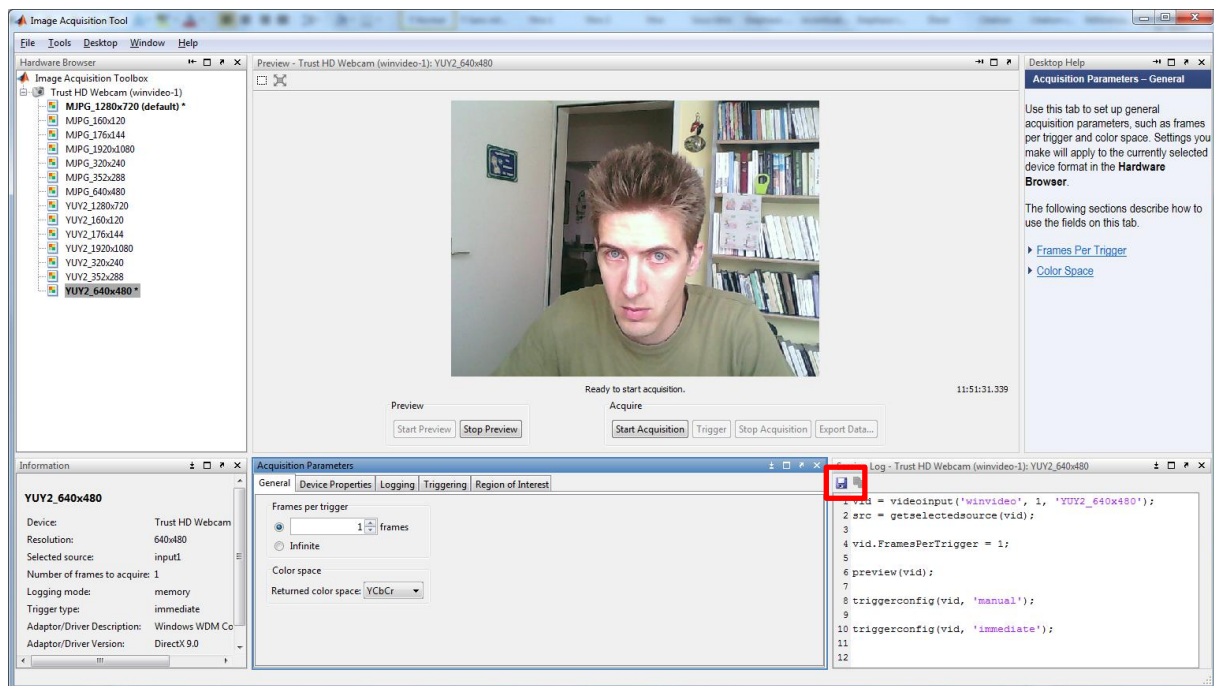
General Settings:
  Parent = [1x1 videoinput]
  Selected = on
  SourceName = input1
  Tag =
  Type = videosource

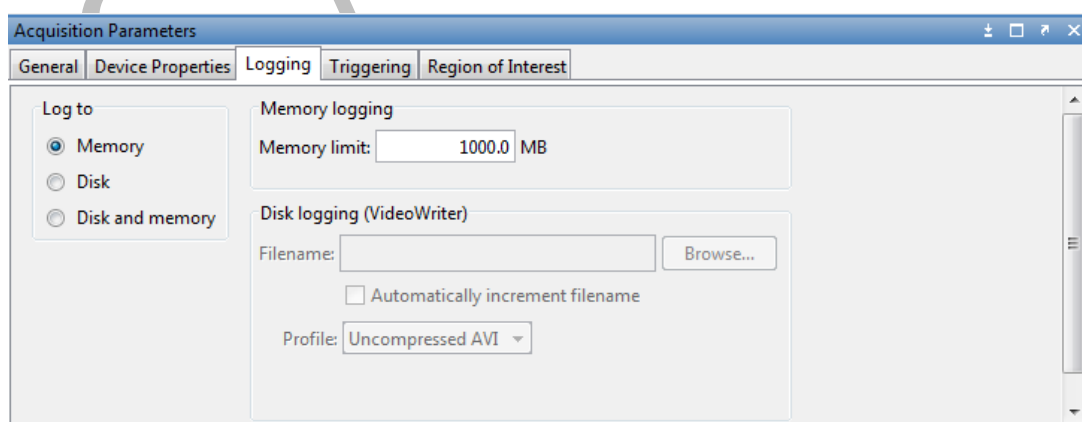
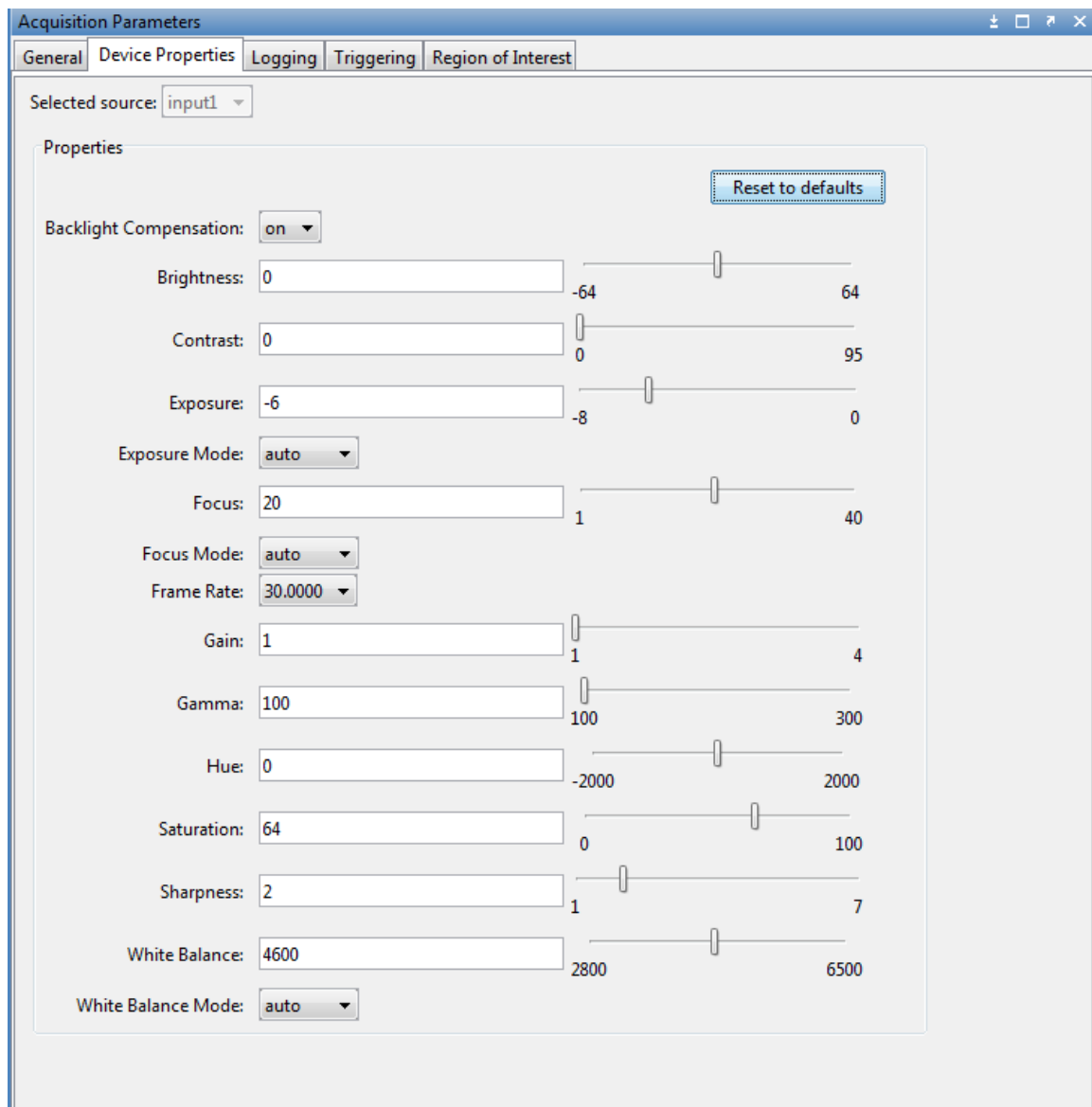
Device Specific Properties:
  BacklightCompensation = on
  Brightness = 0
  Contrast = 0
  Exposure = -6
  ExposureMode = auto
  Focus = 20
  FocusMode = auto
  FrameRate = 30.0000
  Gain = 1
  Gamma = 100
  Hue = 0
  Saturation = 64
  Sharpness = 2
  WhiteBalance = 4600
  WhiteBalanceMode = auto

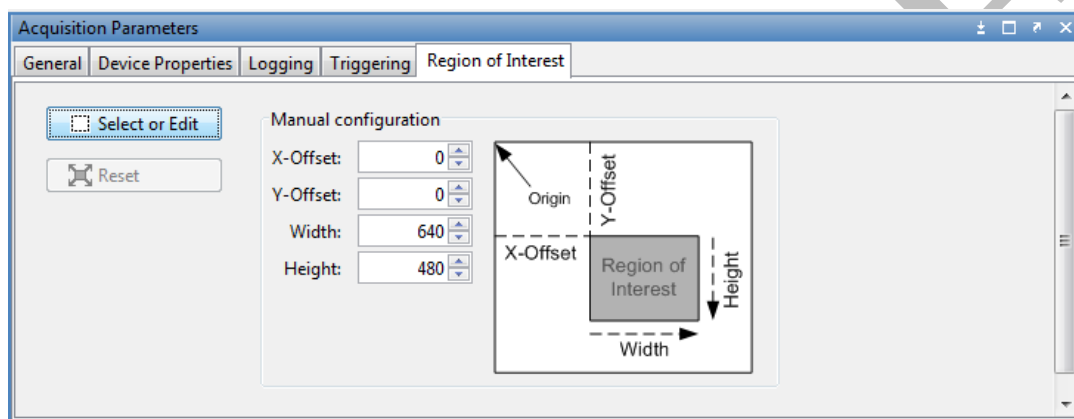
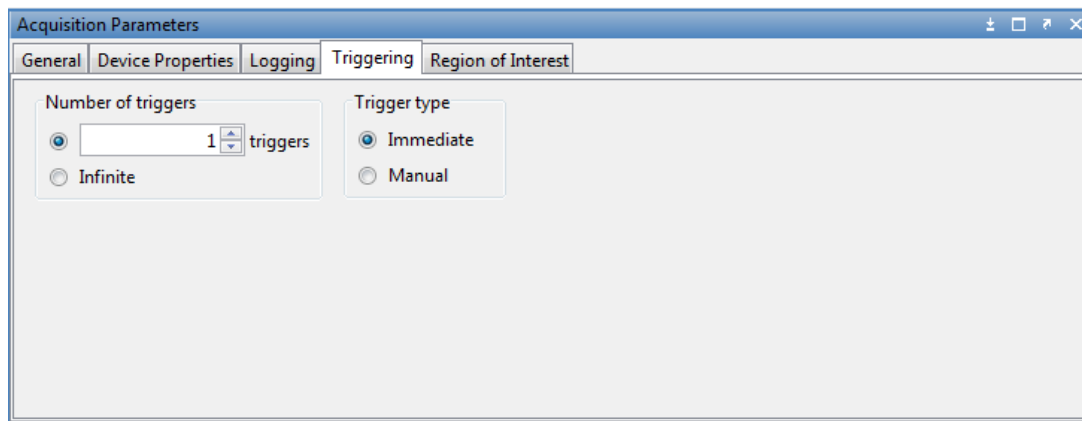
>>
```







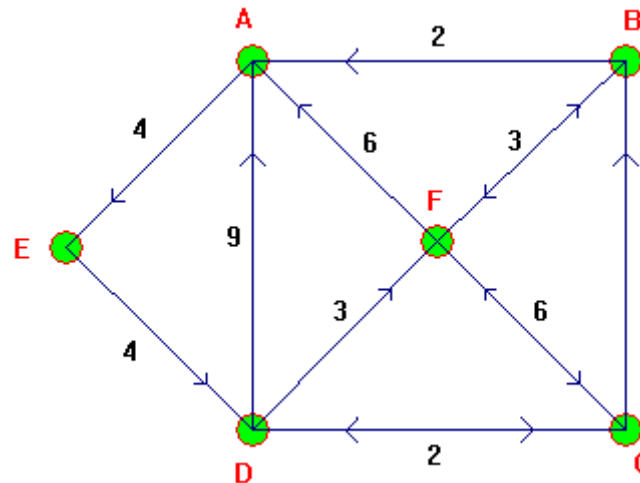




Bioinformatic (with Bioinformatics Toolbox)

Shortest Path

We have study in the theoretical computing course the dijkra algorithm based on the following convex oriented chart:



It seems the bioinformatics toolbox can only manage one directional links or no directions at all. Thus we were obliged to simplify the example calculate by hand as you will see in the figure below.

The corresponding script using the functions **biograph()**, **graphshortestpath()** and **getedgesbynodeid()**:

```

1 - close all;clear all;clc
2 - W = [2 6 3 9 6 3 2 9 4 4];
3 - DG = sparse([2 6 6 3 3 4 3 4 1 5],[1 1 2 2 6 6 4 1 5 4],W);
4 - h = view(biograph(DG,[],'ShowWeights','on'));
5 - [dist,path] = graphshortestpath(DG,3,1)
6 - set(h.Nodes(path),'Color',[1 0.4 0.4])
7 - edges = getedgesbynodeid(h,get(h.Nodes(path),'ID'));
8 - set(edges,'LineColor',[1 0 0])
9 - set(edges,'LineWidth',1.5)

```

And we get in the **Command Window**:

```
Command Window

dist =

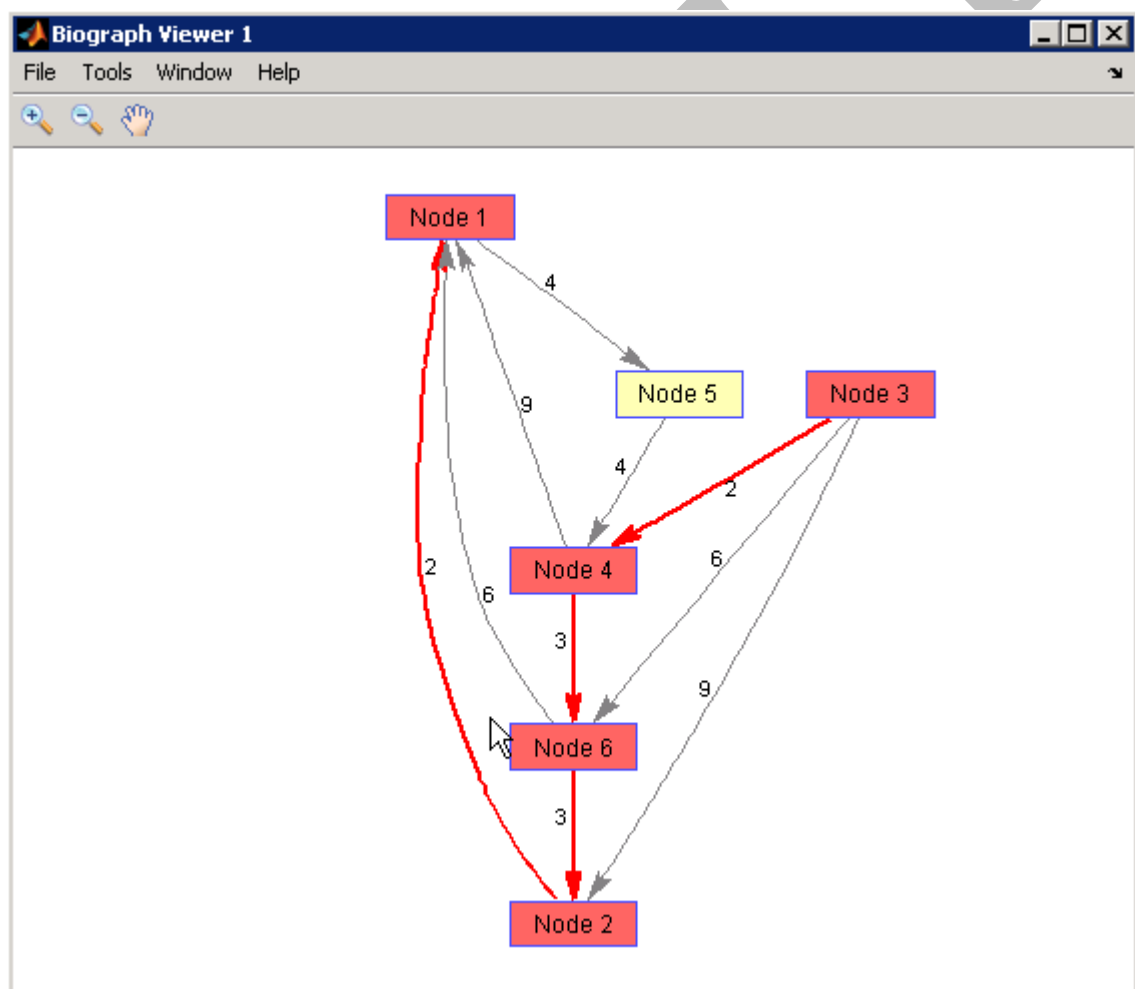
    10

path =

     3     4     6     2     1

fx >>
```

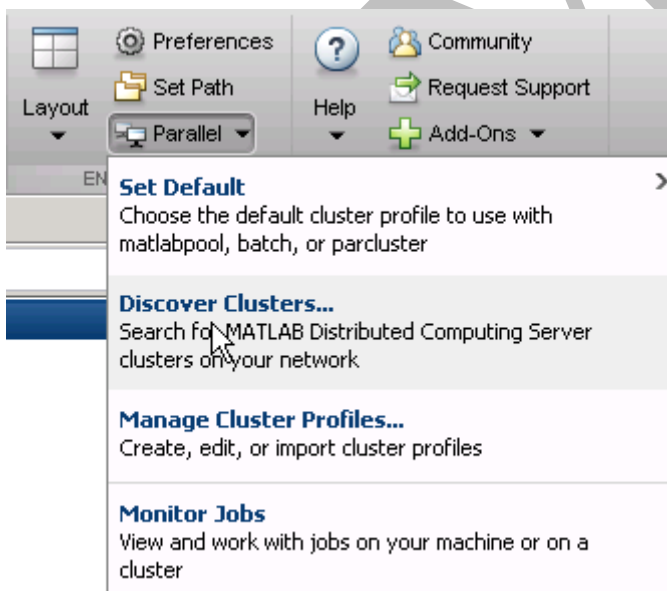
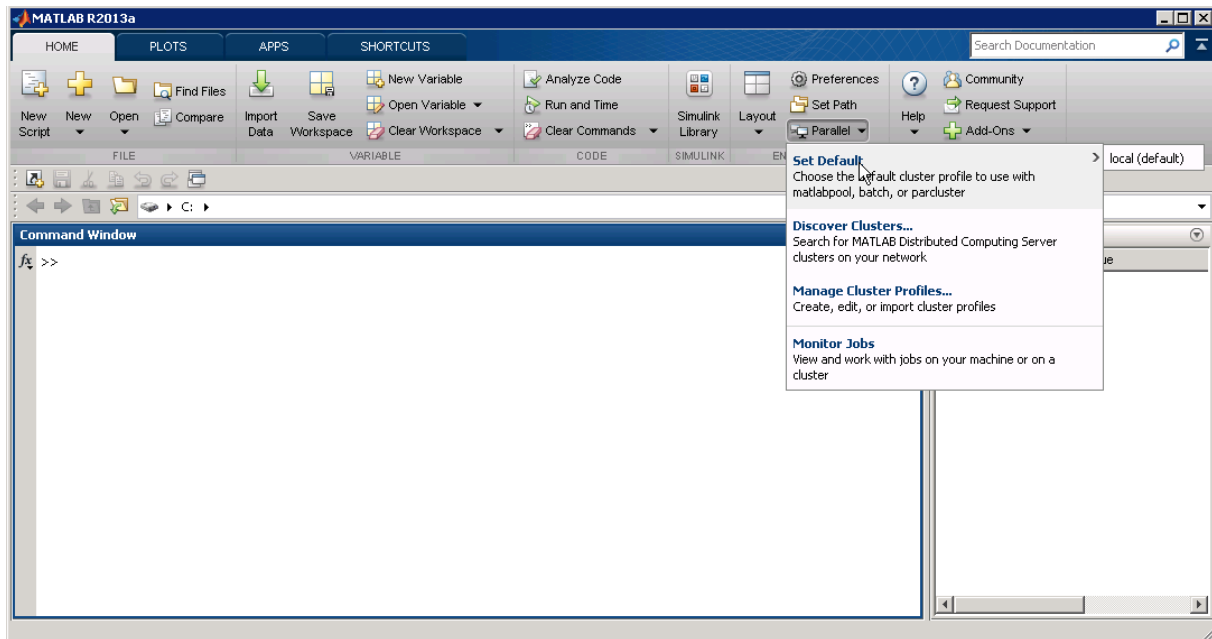
And the corresponding graph:

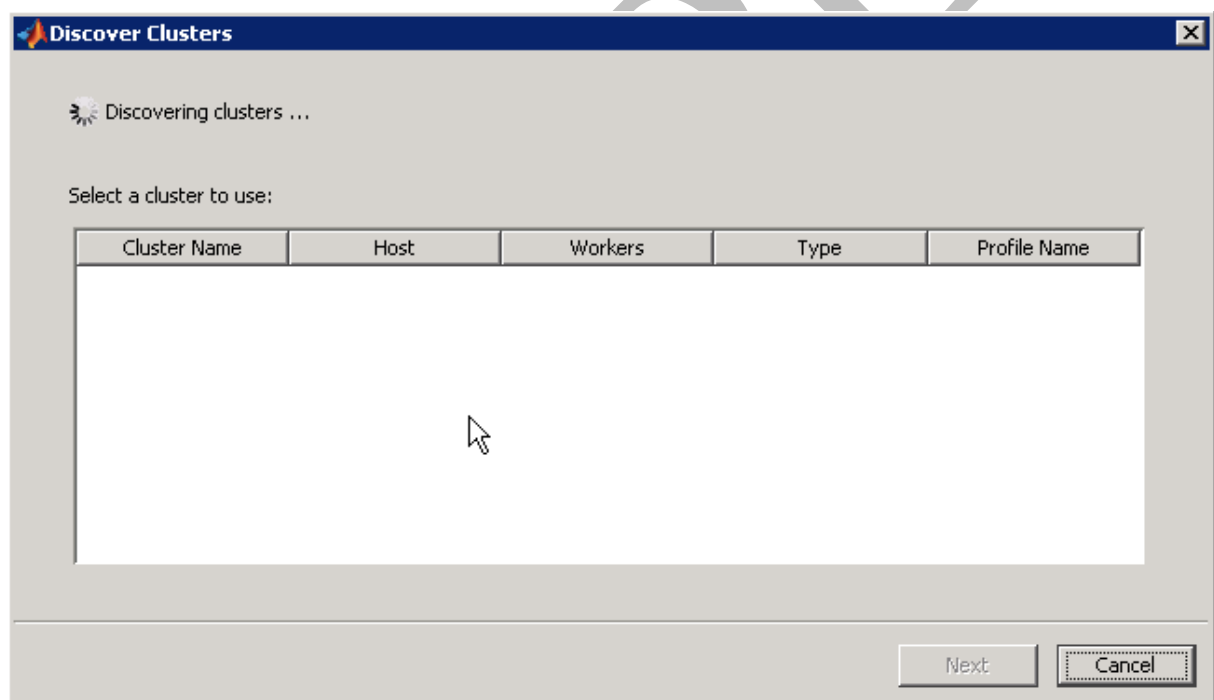
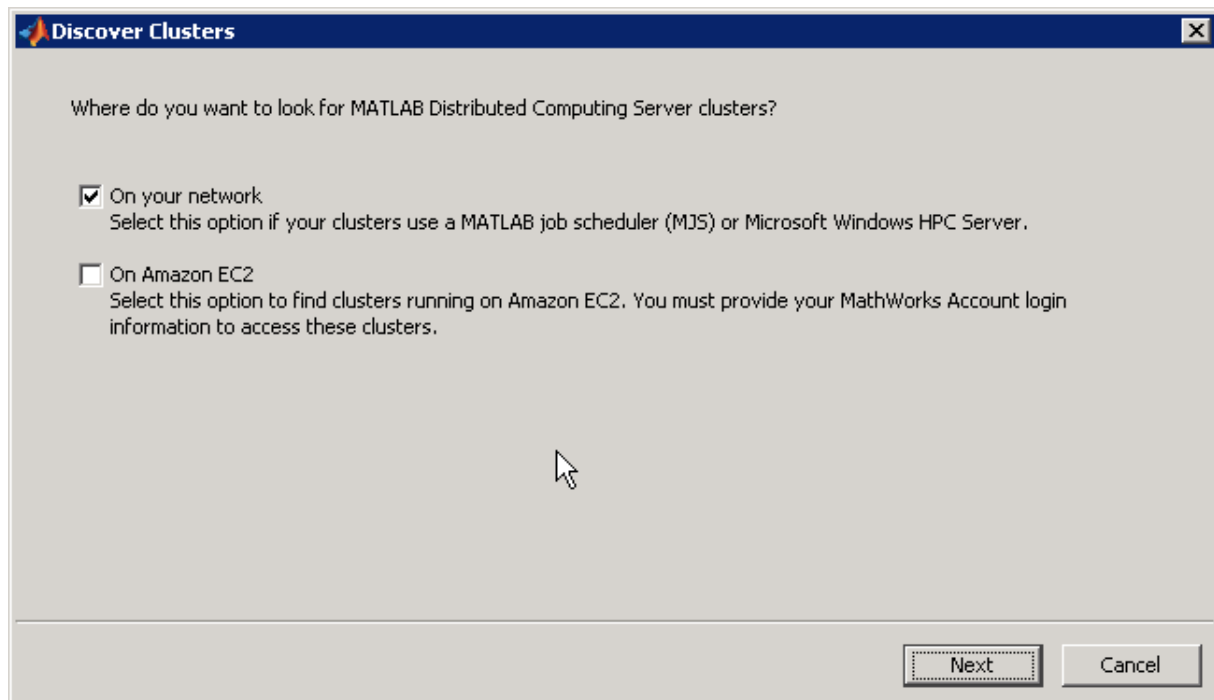


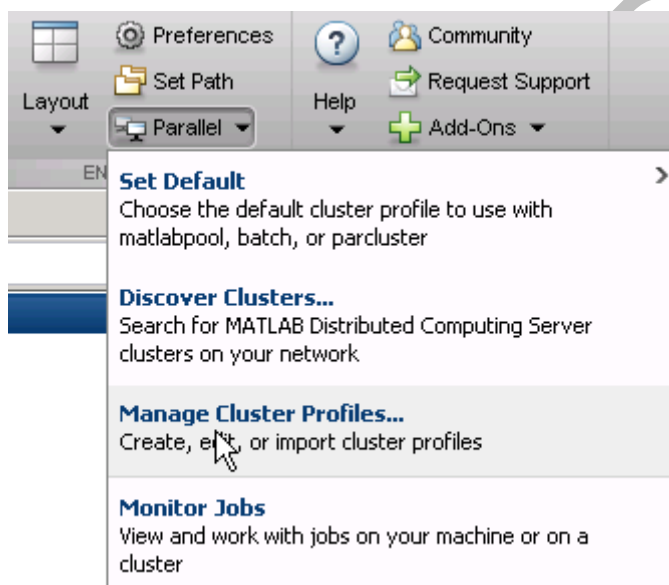
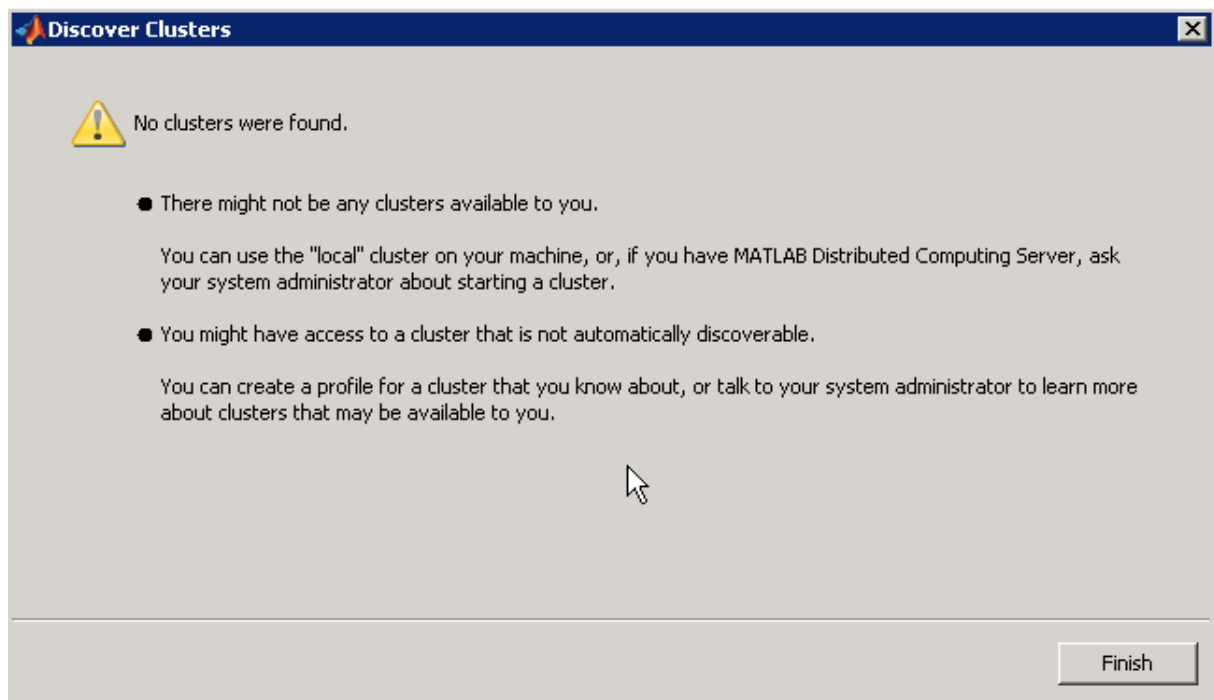
Parallel Computing (with Parallel Computing Toolbox)

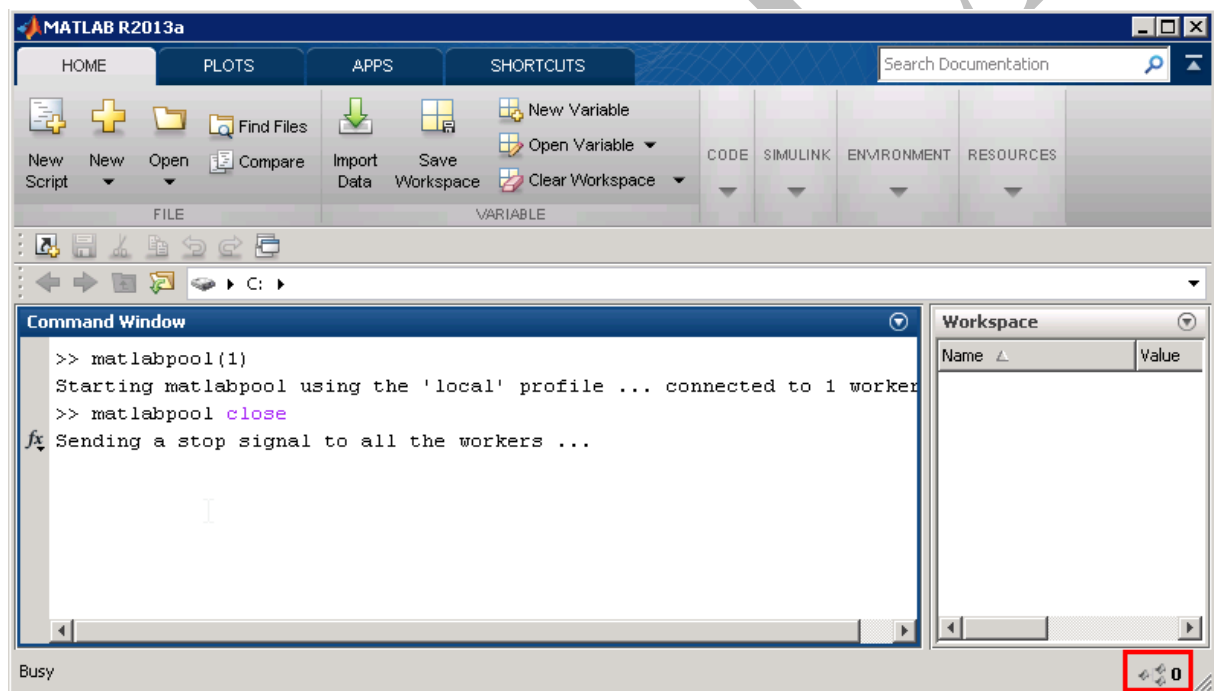
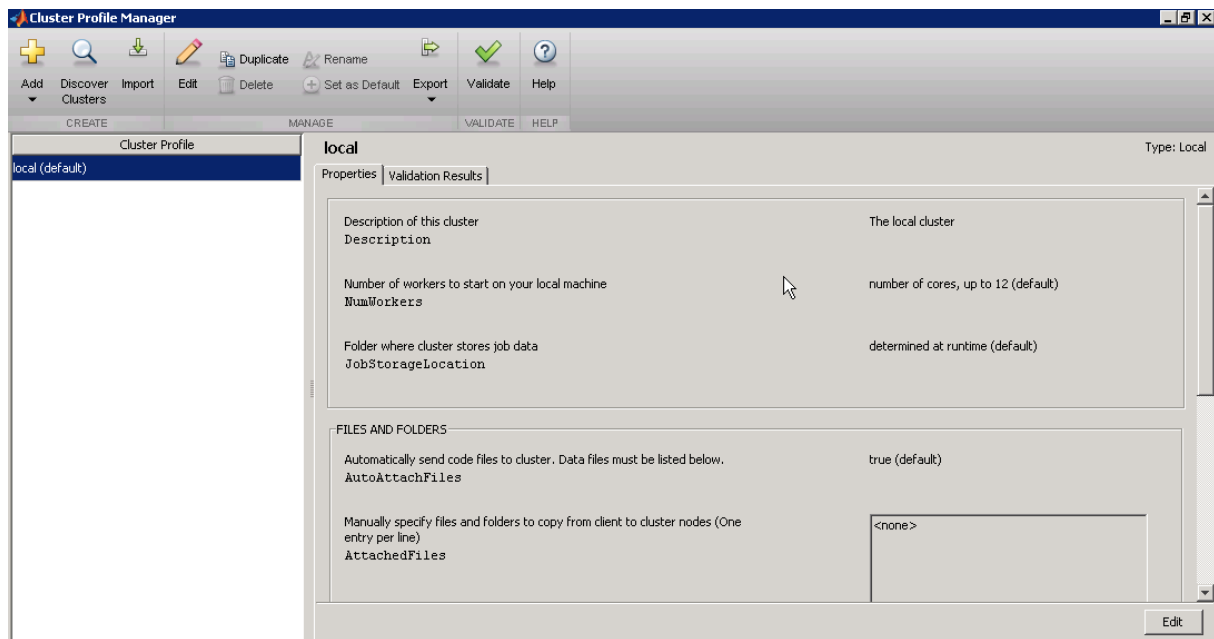
<http://www.mathworks.ch/ch/help/distcomp/release-notes.html>

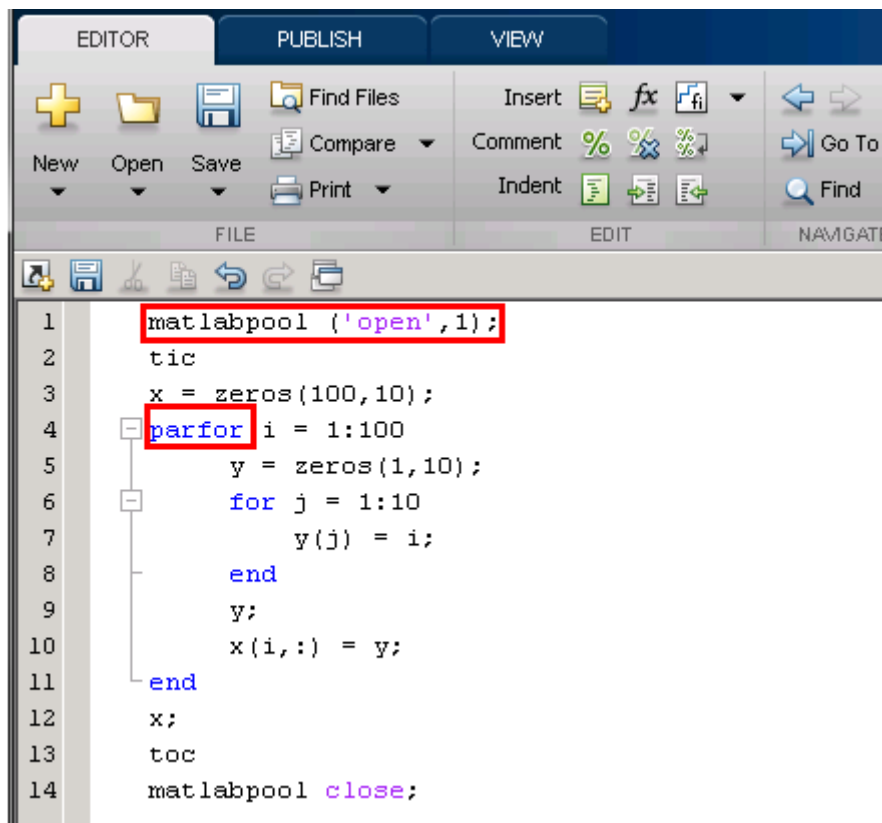
Clusters computing











Here are some tips:

- when using **parfor**, all the executions of the loop need to be independent from each other. This is such that the **parfor** loop doesn't even executes consecutively ($i = 1, 2, 3, 4, \dots$), but quite randomly ($i = 3, 4, 59, 37, \dots$) decided by MATLAB. This way, you cannot have any variable that can be modified by all the different processors at the same time, unless it has the proper dimensional index ($x(i,j,k)$).
- the **parfor** loop can only be done for one level, i.e., you cannot have a **parfor** loop inside another **parfor** loop. Sorry! However, the **parfor** loop doesn't need to be the outermost, it can be inside a regular **for** loop if needed.
- You can have **for** loops inside the **parfor** loop, but you will need to pre-allocate all the variables that grow inside the **for** loops. Just pre-allocating immediately before the corresponding **for** loop will do. (This is usually a recommendation for speed in regular MATLAB runs, but it is a must in parallel mode.)
- You cannot have any instruction that calls to the workspace inside the **parfor** loop, such as delete variables (**clear**) or list variables (**who**).
- Be careful if you have figures inside the **parfor** loop. They should work, and you can even save them. But if you are displaying on screen, make sure you close the figures within the loop, so that they don't flood your desktop!

NVidia GPU computing

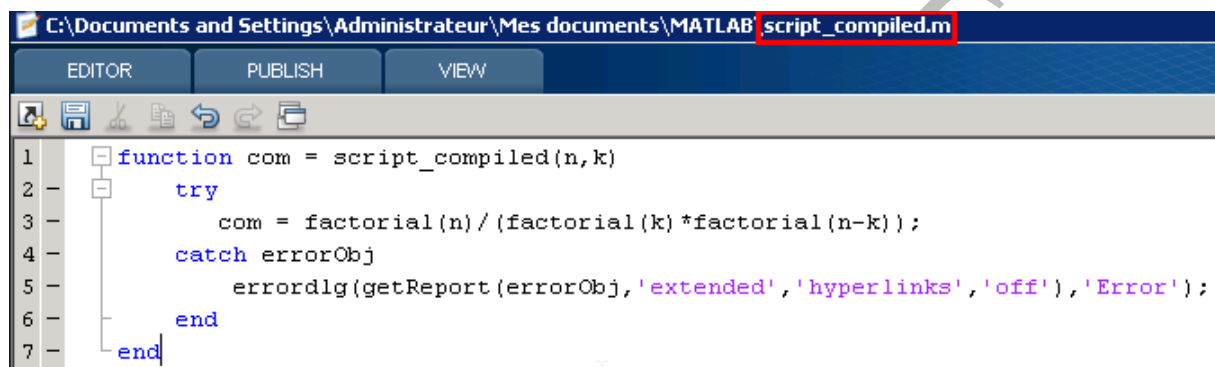
<https://developer.nvidia.com/cuda-gpus>

Advanced scripting (C#, C++, Compiler Toolbox)

*Create a *.p (protected script) file (aka "pcode")*

For this part you do not need any compiler but in my point of view this is the good place to speak about this feature.

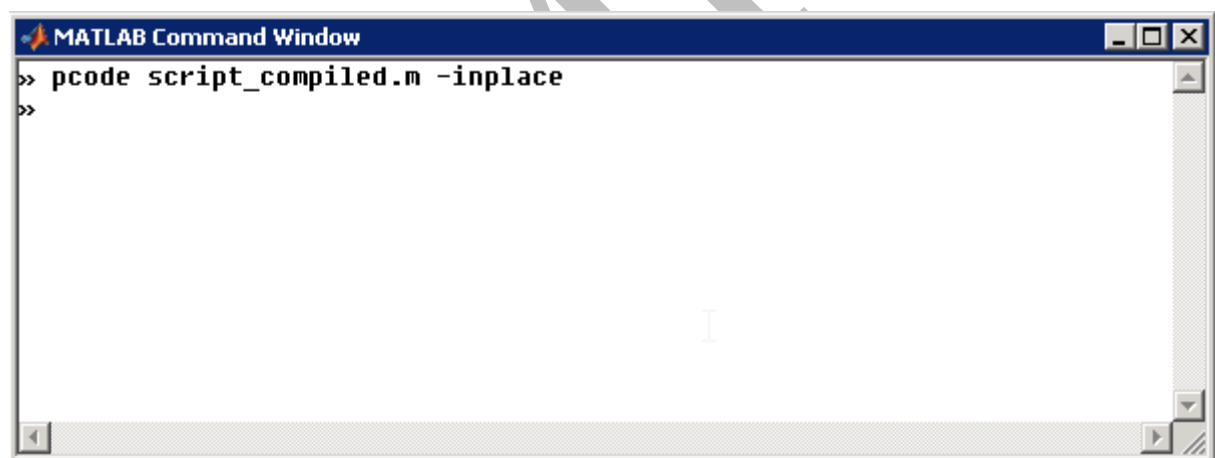
Consider the following script:



The image shows a MATLAB Editor window with the title bar 'C:\Documents and Settings\Administrateur\Mes documents\MATLAB script_compiled.m'. The window has tabs for EDITOR, PUBLISH, and VIEW. The EDITOR tab is active, showing a script with the following code:

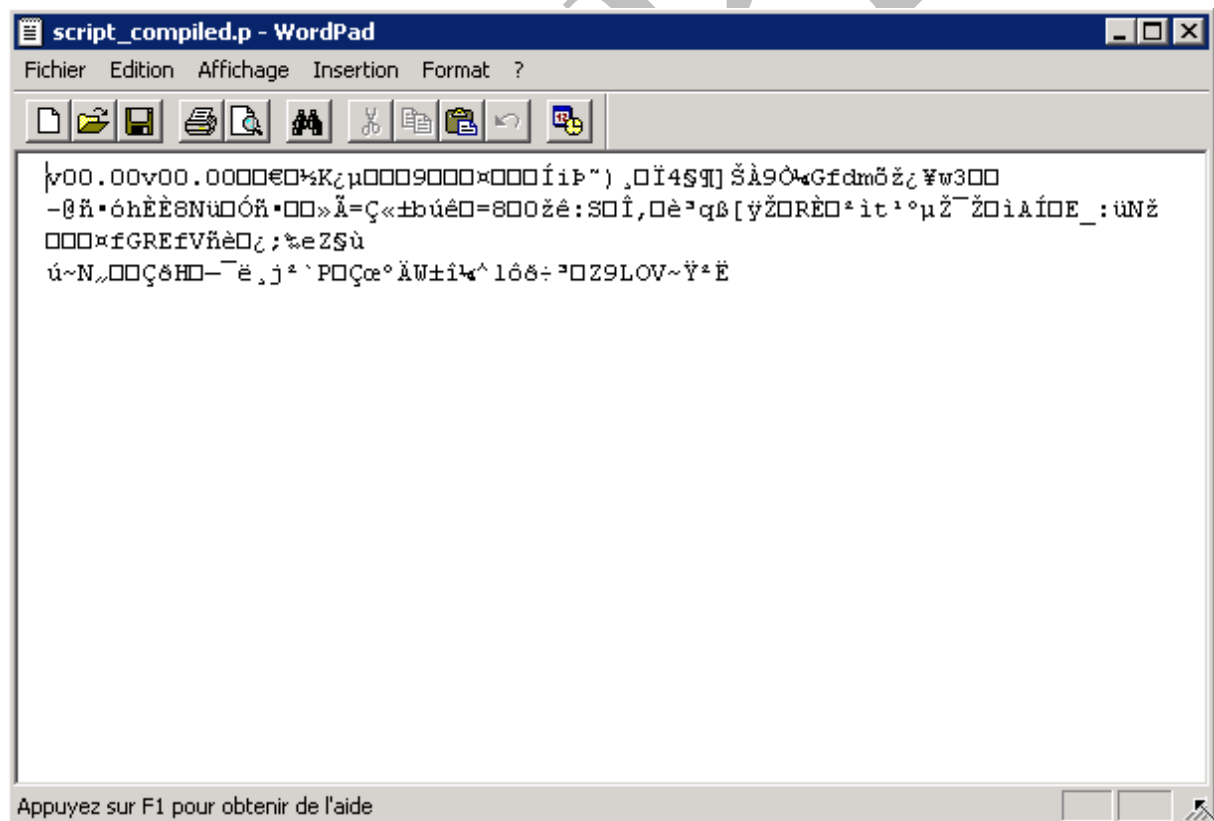
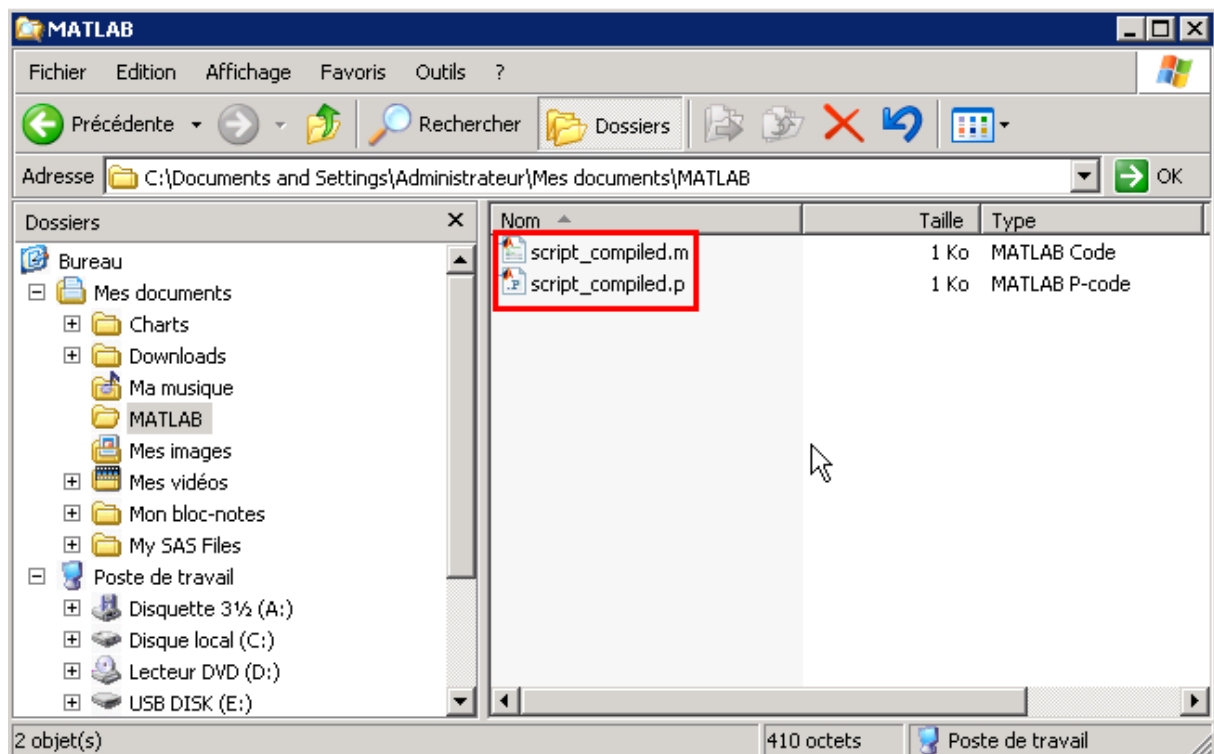
```
1 function com = script_compiled(n,k)
2     try
3         com = factorial(n) / (factorial(k) * factorial(n-k));
4     catch errorObj
5         errordlg(getReport(errorObj, 'extended', 'hyperlinks', 'off'), 'Error');
6     end
7 end
```

We want to make this an unreadable file (or difficult to read...) for other colleagues.



The image shows a MATLAB Command Window with the title bar 'MATLAB Command Window'. The command entered is:

```
>> pcode script_compiled.m -inplace
>>
```

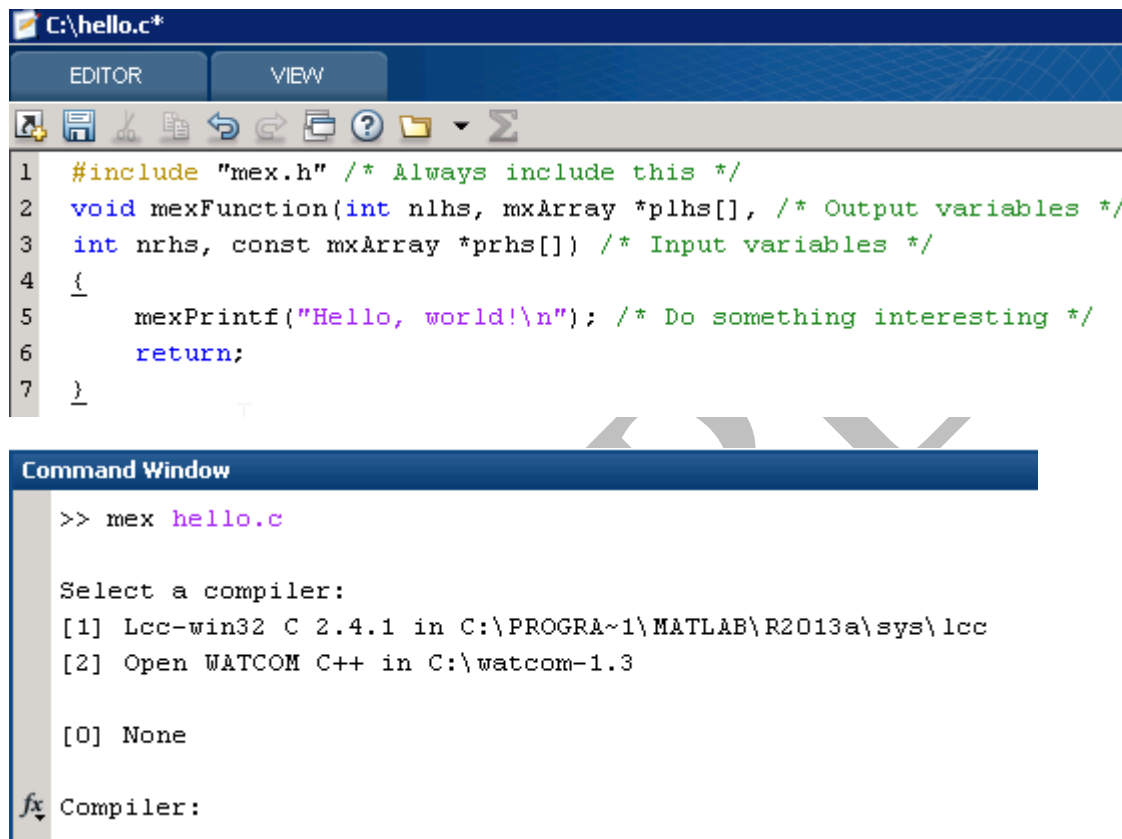


C MEX file

It is possible to compile C, C++, or Fortran code so that it is callable from Matlab. This kind of program is called a Matlab Executable (MEX) external interface function, or more briefly a "MEX-function." MEX enables the high performance of C, C++, and Fortran while working

within the Matlab environment. We will discuss C/MEX functions, which also applies directly to C++/MEX. Fortran/MEX is quite different and we do not discuss it here.

Matlab's JIT interpreter in recent versions runs M-code so efficiently that it is often times difficult to do much better with C. Before turning to MEX in an application, optimize your M-code (see my other article, "Writing Fast Matlab Code"). MEX-functions are best suited to substitute one or two bottleneck M-functions in an application. If you replace all functions in an application with MEX, you might as well port the application entirely to C.



The screenshot shows a MATLAB editor window titled 'C:\hello.c*'. The editor contains the following C code:

```
1 #include "mex.h" /* Always include this */
2 void mexFunction(int nlhs, mxArray *plhs[], /* Output variables */
3 int nrhs, const mxArray *prhs[]) /* Input variables */
4 {
5     mexPrintf("Hello, world!\n"); /* Do something interesting */
6     return;
7 }
```

Below the editor is the Command Window, which shows the command `>> mex hello.c` and the resulting compiler selection dialog:

```
Select a compiler:
[1] Lcc-win32 C 2.4.1 in C:\PROGRA~1\MATLAB\R2013a\sys\lcc
[2] Open WATCOM C++ in C:\watcom-1.3
[0] None

fx Compiler:
```

If successful, this command produces a compiled file called `hello.mexa64` (or similar, depending on platform). Compiling requires that you have a C compiler and that Matlab is configured to use it. Matlab will autodetect most popular compilers, including Microsoft Visual C/C++ and GCC. As a fallback, some distributions of Matlab come with the Lcc C compiler. Run `mex -setup` to change the selected compiler and build settings.

Note that compiled MEX files might not be compatible between different platforms or different versions of Matlab. They should be compiled for each platform/version combination that you need. It is possible to compile a MEX file for a target platform other than the host's using the `-<arch>` option, for example, `mex -win32 hello.c`.


```

Command Window
>> mex hello.c

Select a compiler:
[1] Lcc-win32 C 2.4.1 in C:\PROGRA~1\MATLAB\R2013a\sys\lcc
[2] Open WATCOM C++ in C:\watcom-1.3

[0] None

Compiler: 1

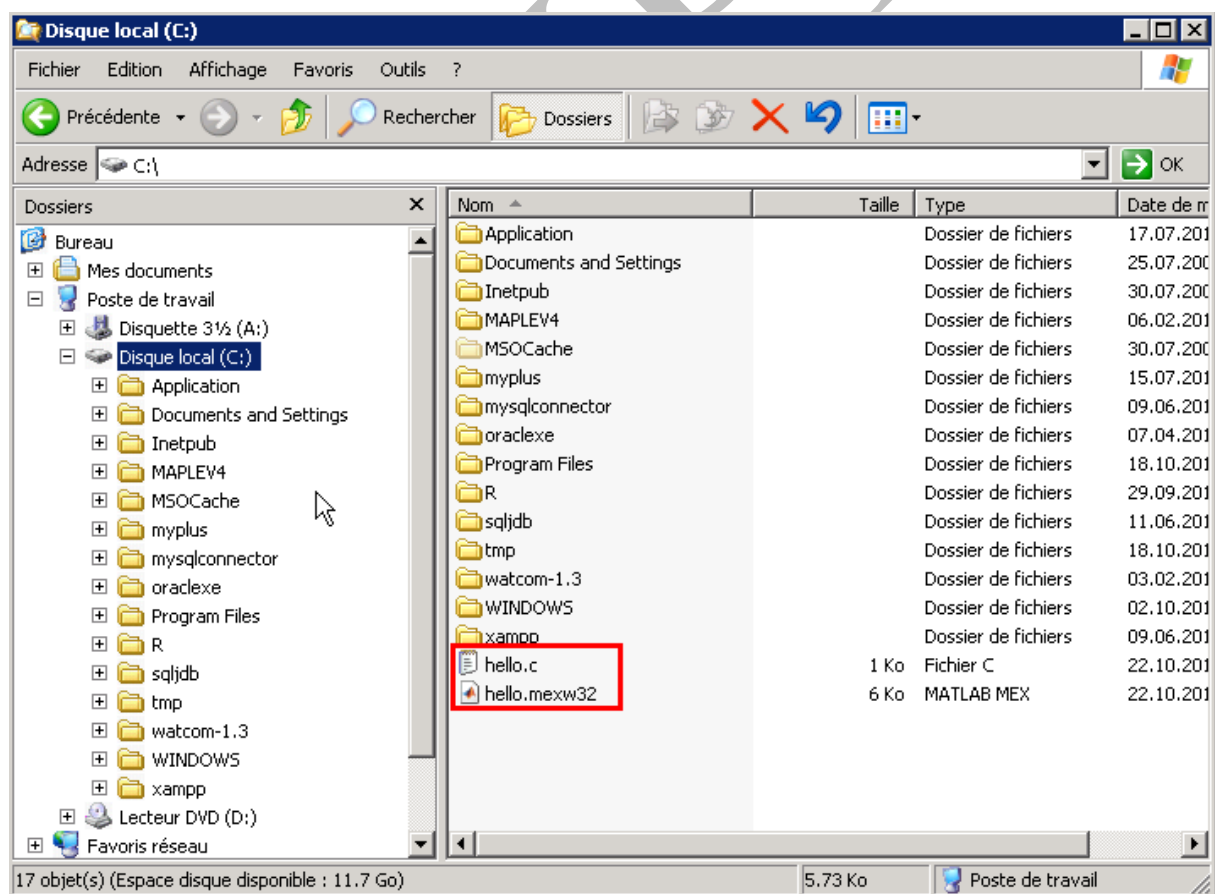
Trying to update options file: C:\Documents and Settings\Administrateur\Application Data
From template: C:\PROGRA~1\MATLAB\R2013a\bin\win32\mexopts\lccopts.bat

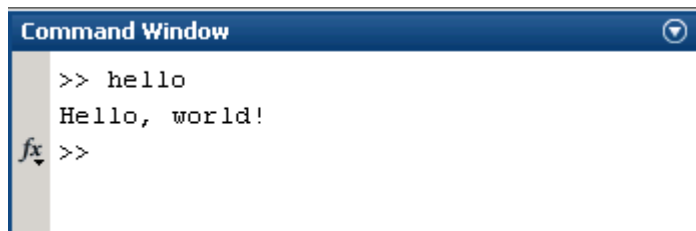
Done . . .

*****
Warning: The MATLAB C and Fortran API has changed to support MATLAB
variables with more than 2^32-1 elements. In the near future
you will be required to update your code to utilize the new
API. You can find more information about this at:
http://www.mathworks.com/help/matlab/matlab\_external/upgrading-mex-files-to-1
Building with the -largeArrayDims option enables the new API.
*****

fx >> |

```

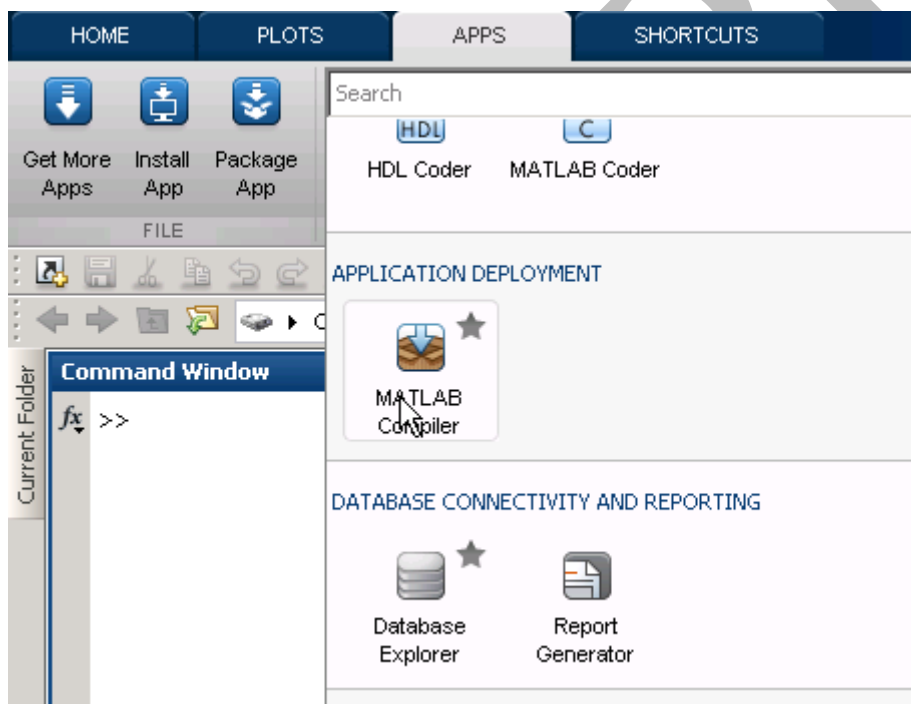
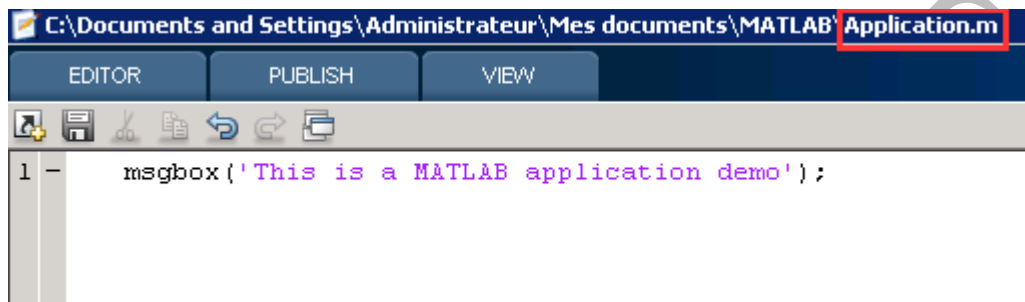


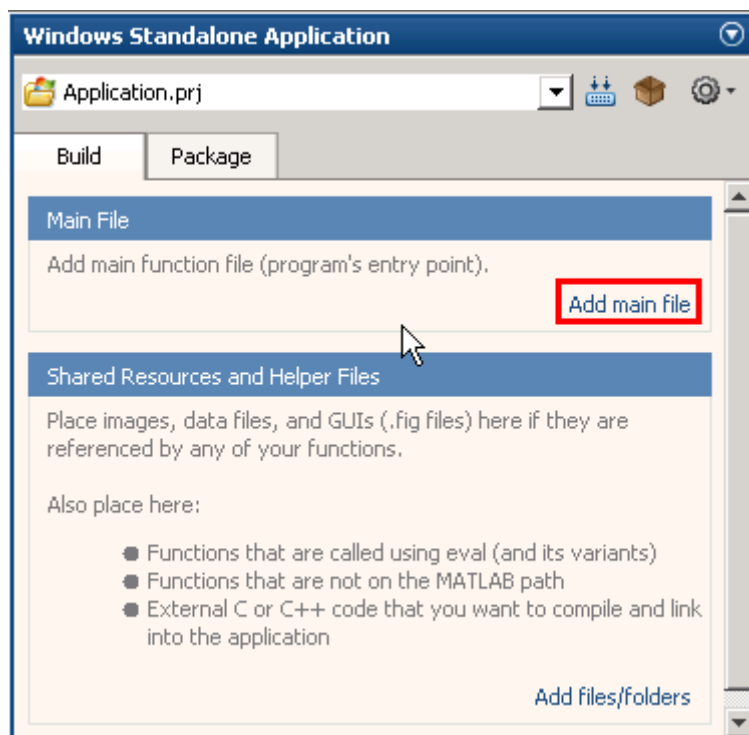
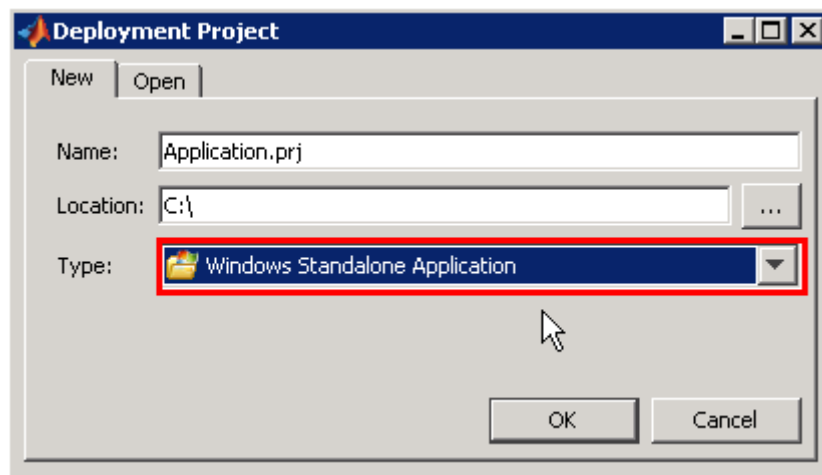


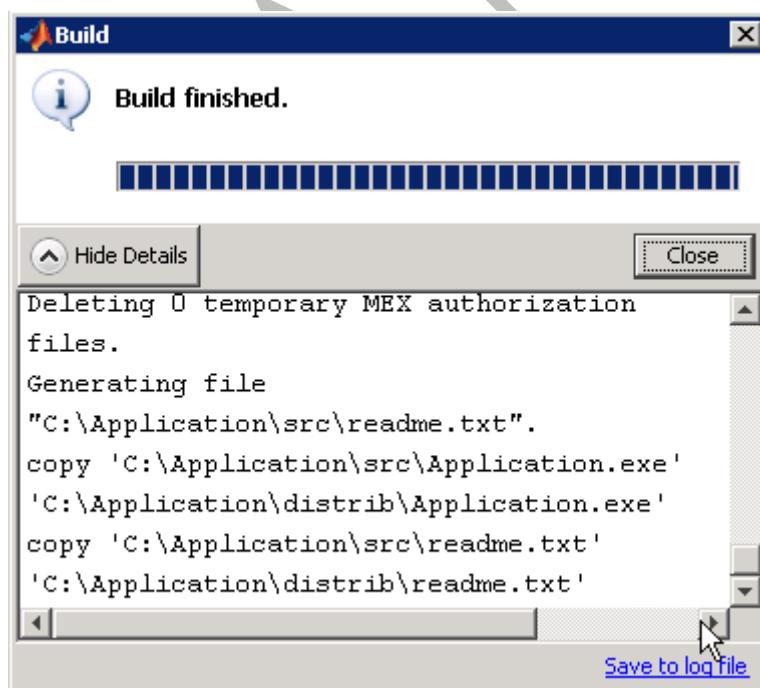
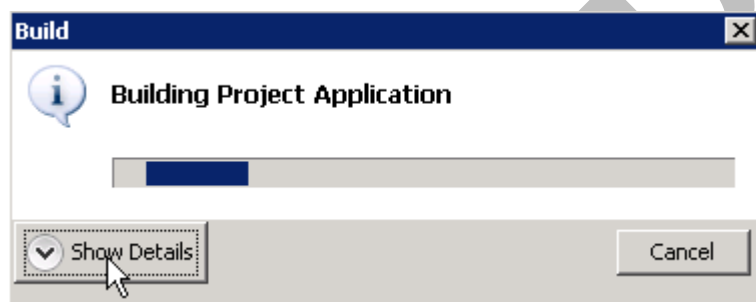
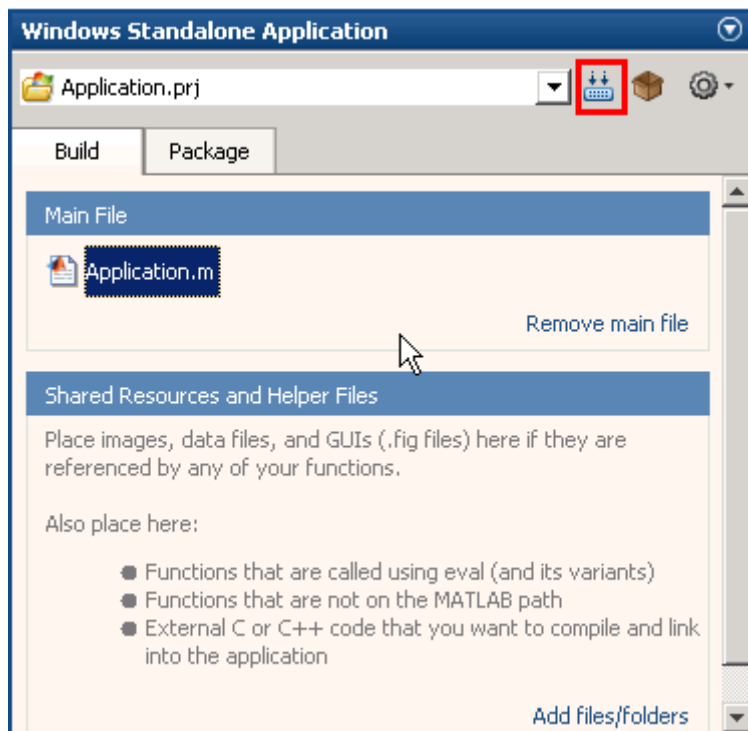
```
Command Window
>> hello
Hello, world!
fx >>
```

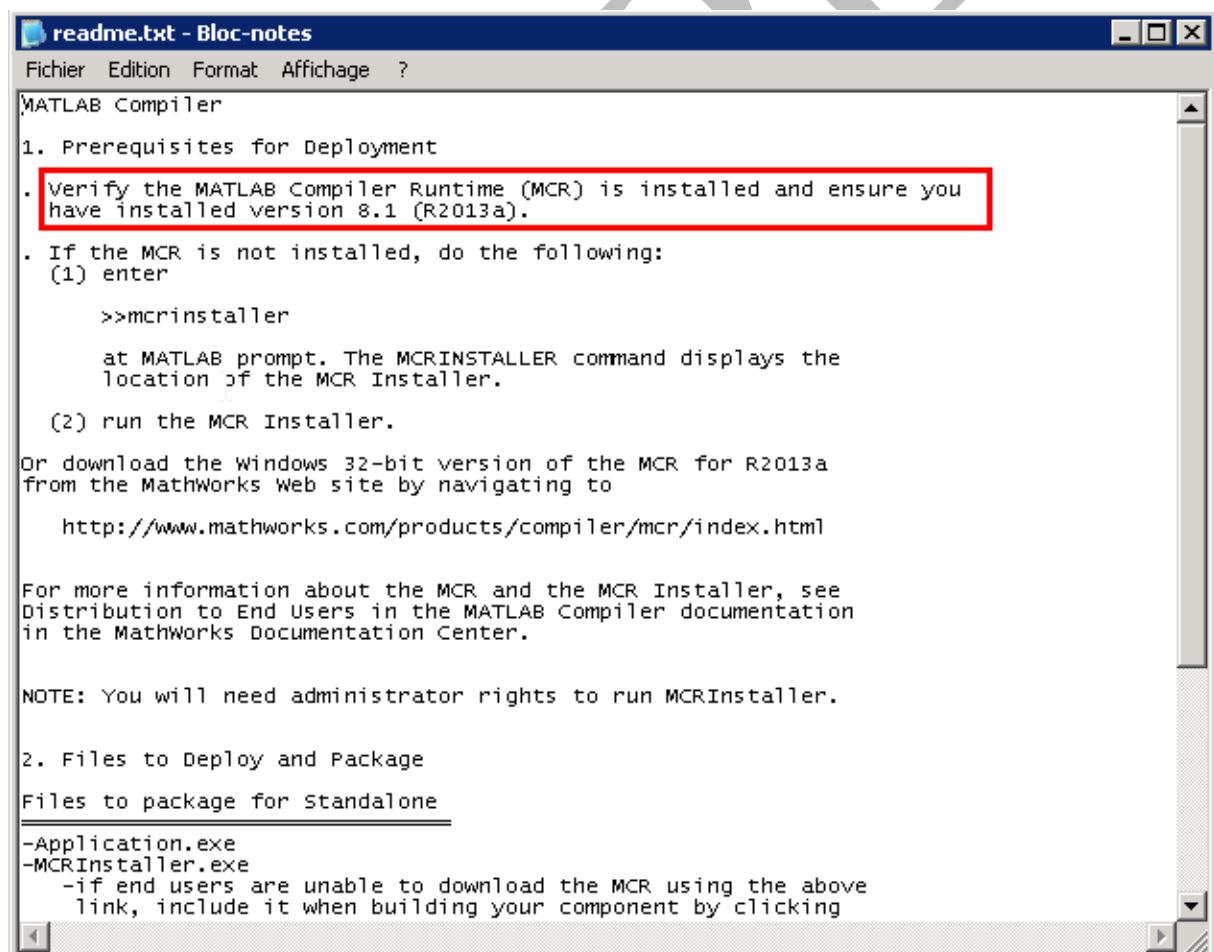
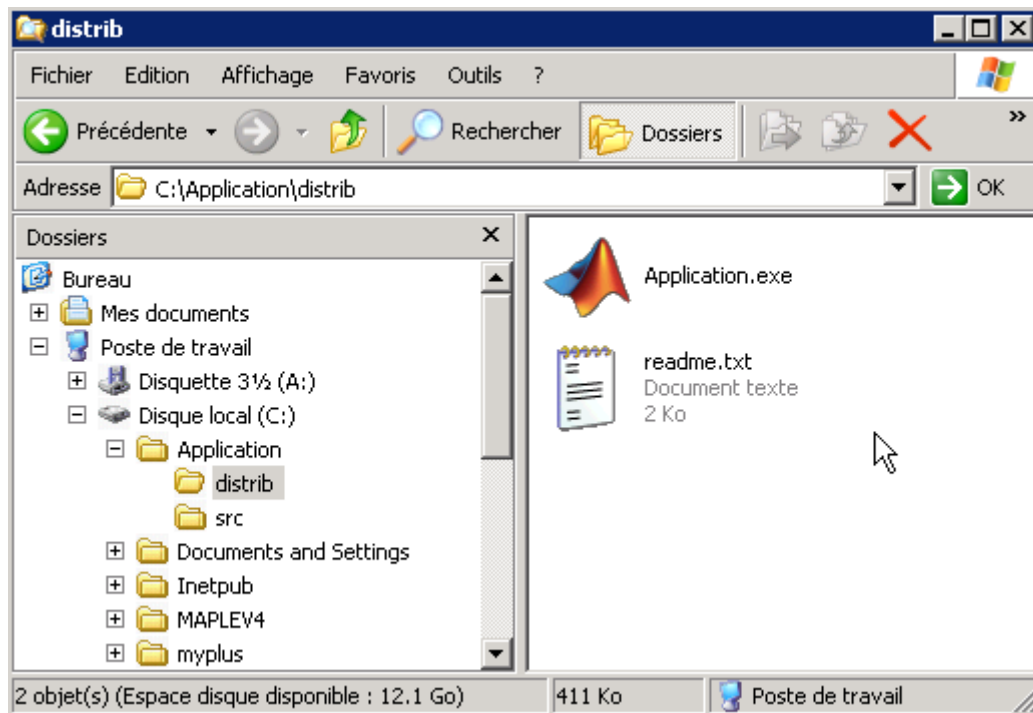
Create executable (*.exe)

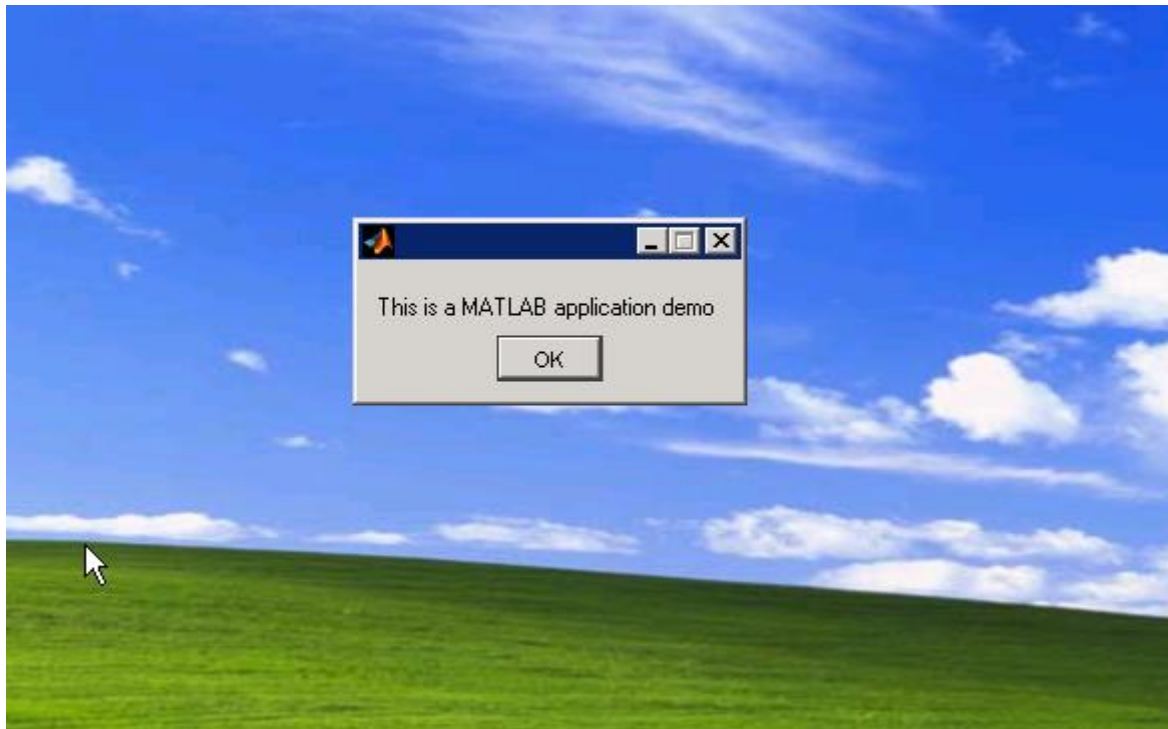
You need MATLAB Runtime for this on the customer computer.





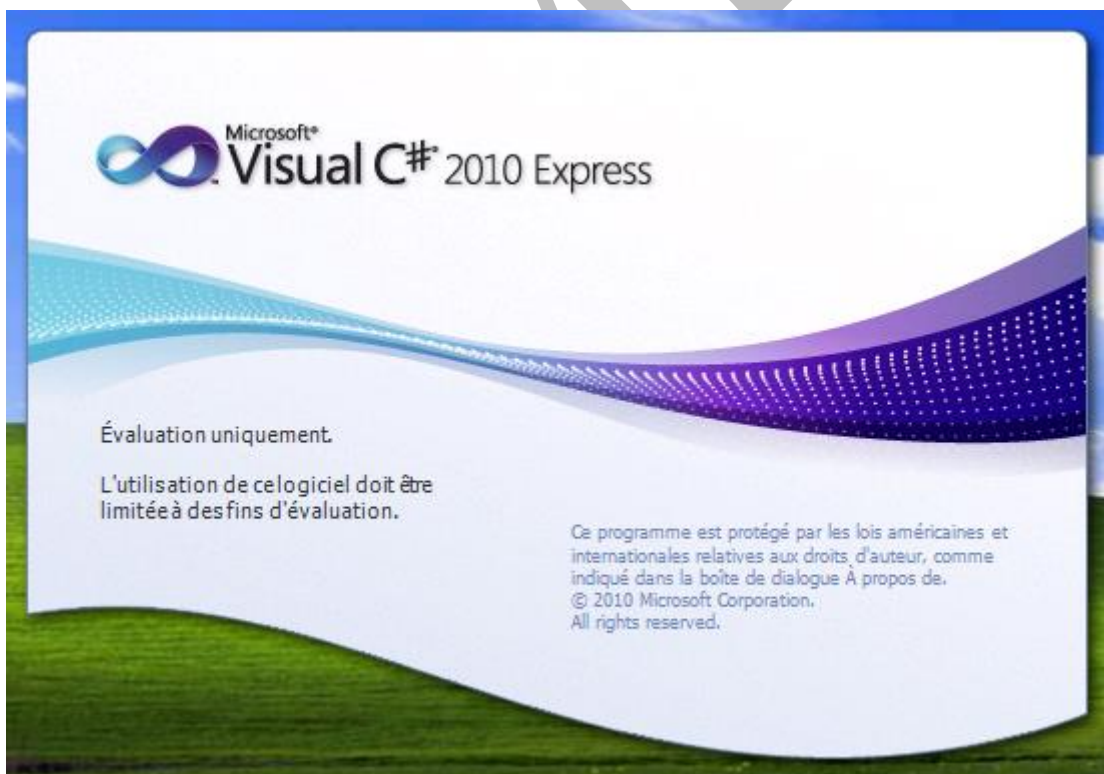


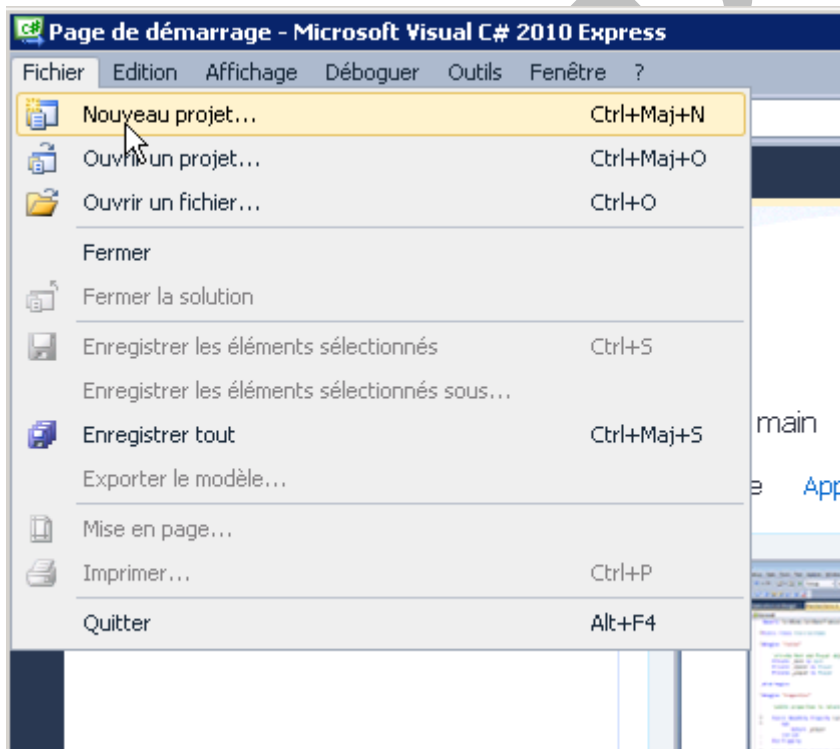
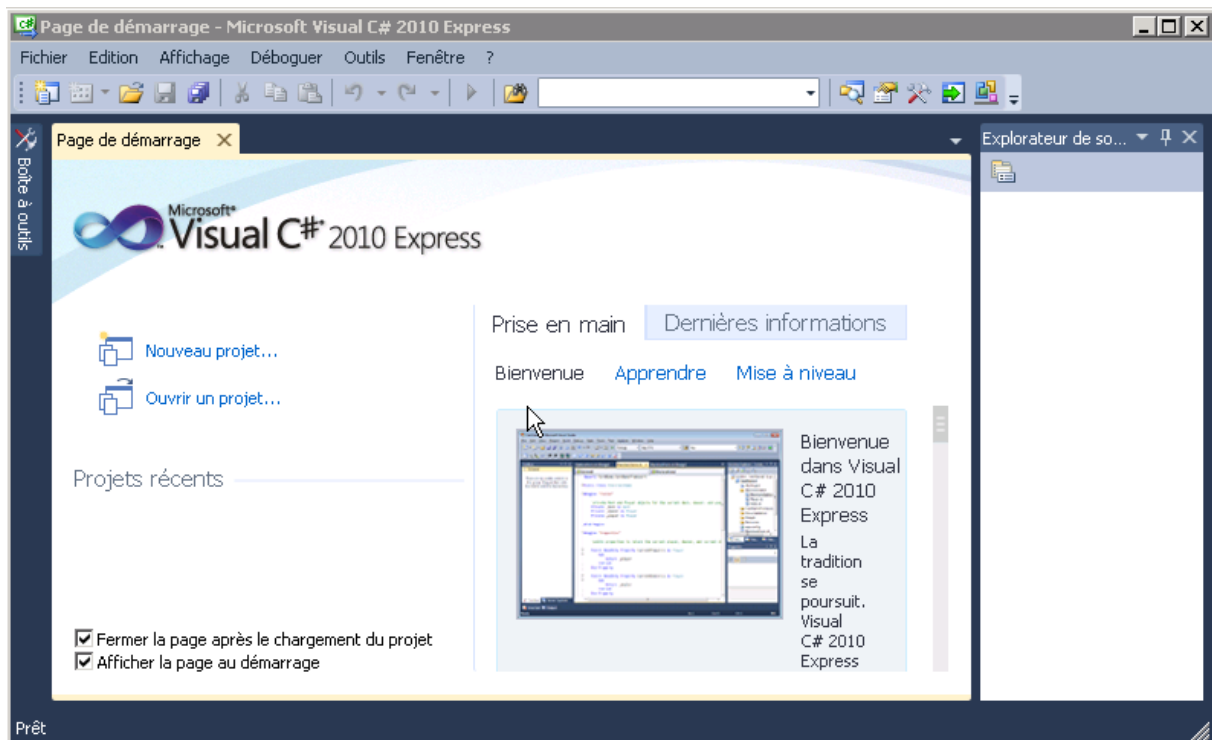


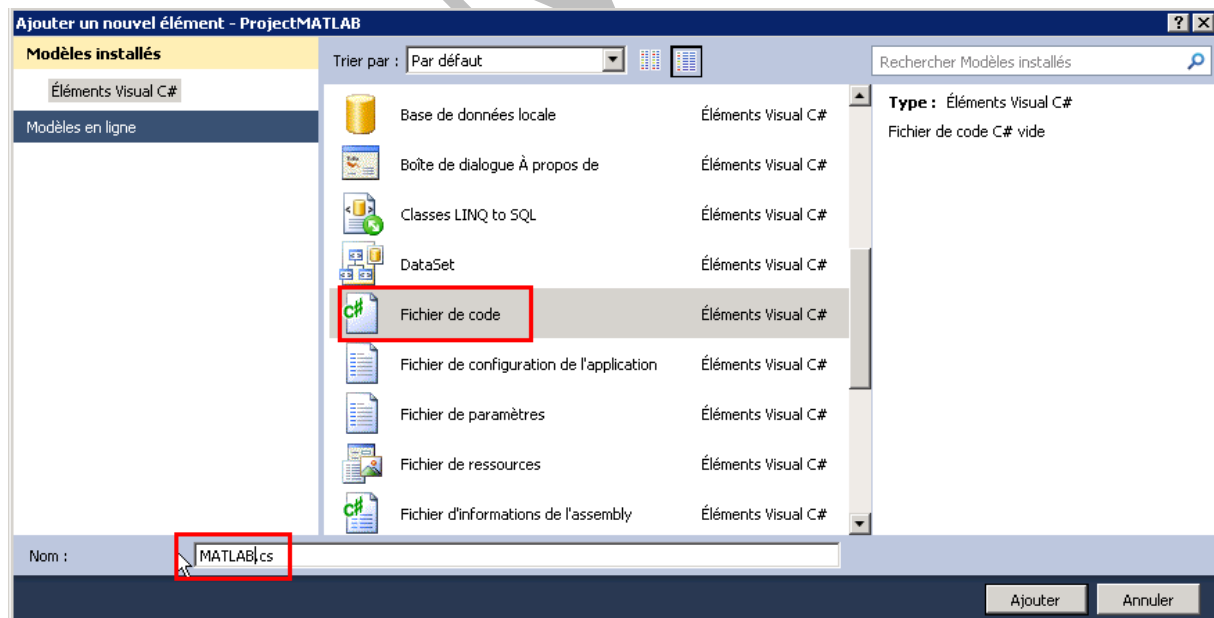
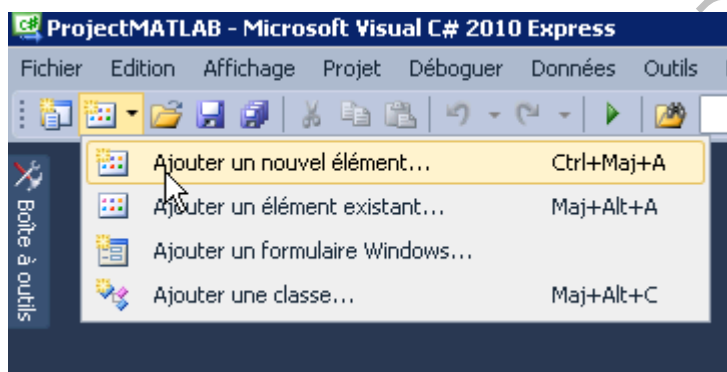
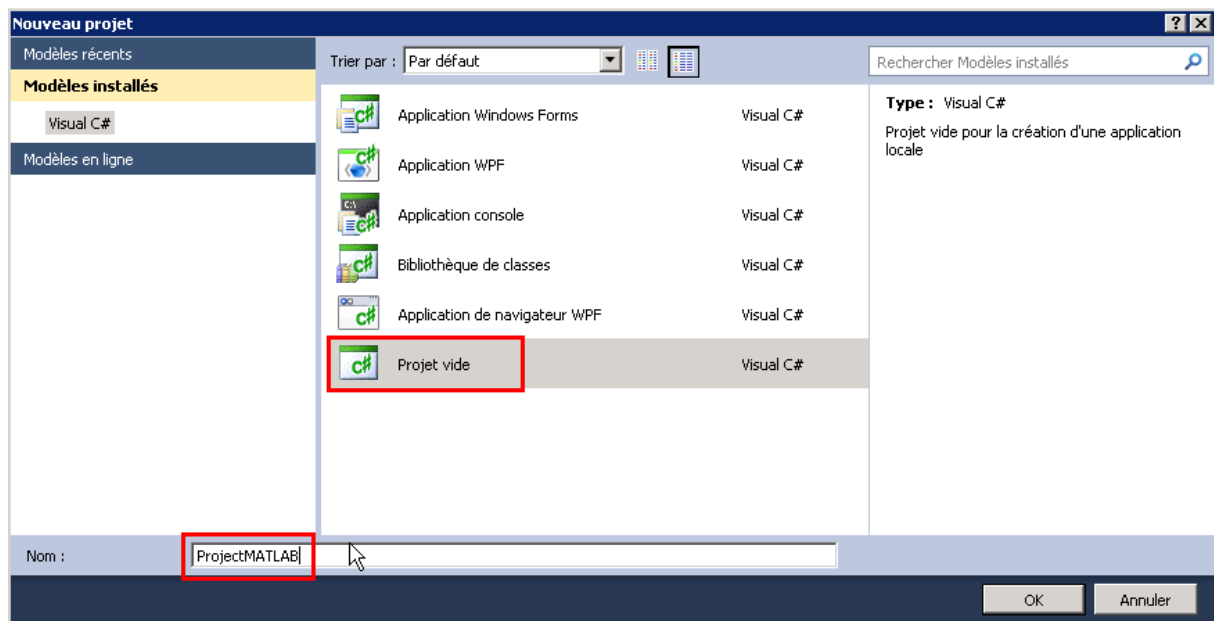


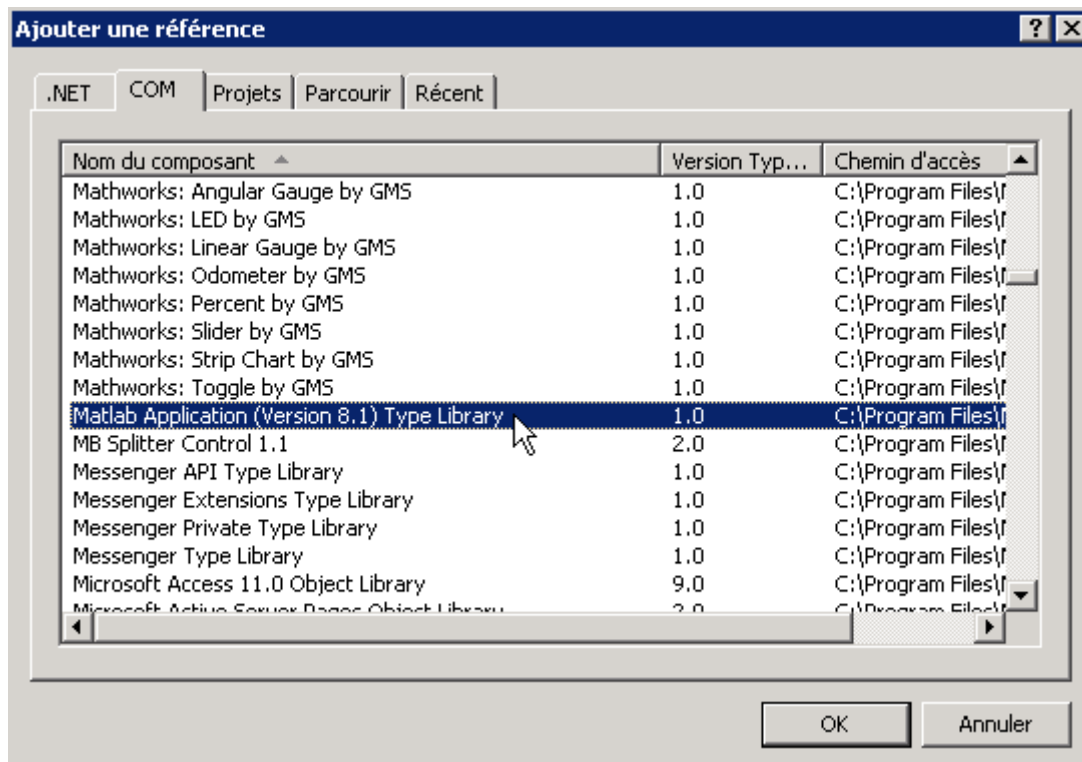
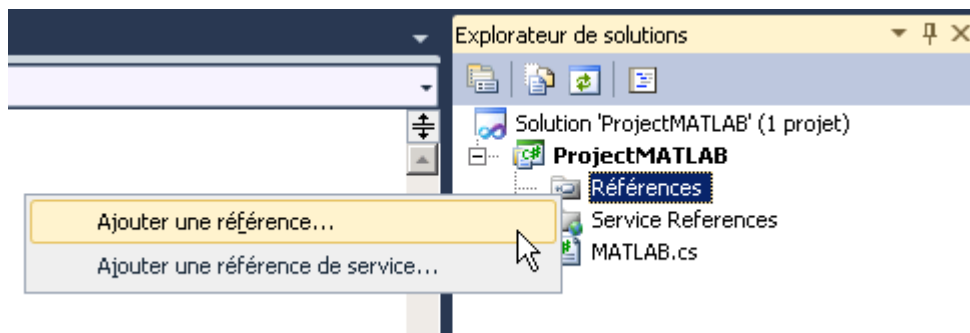
Call MATLAB from C# (VS Express 2010)

This part doesn't need any toolboxes:



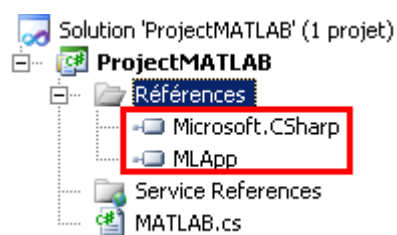
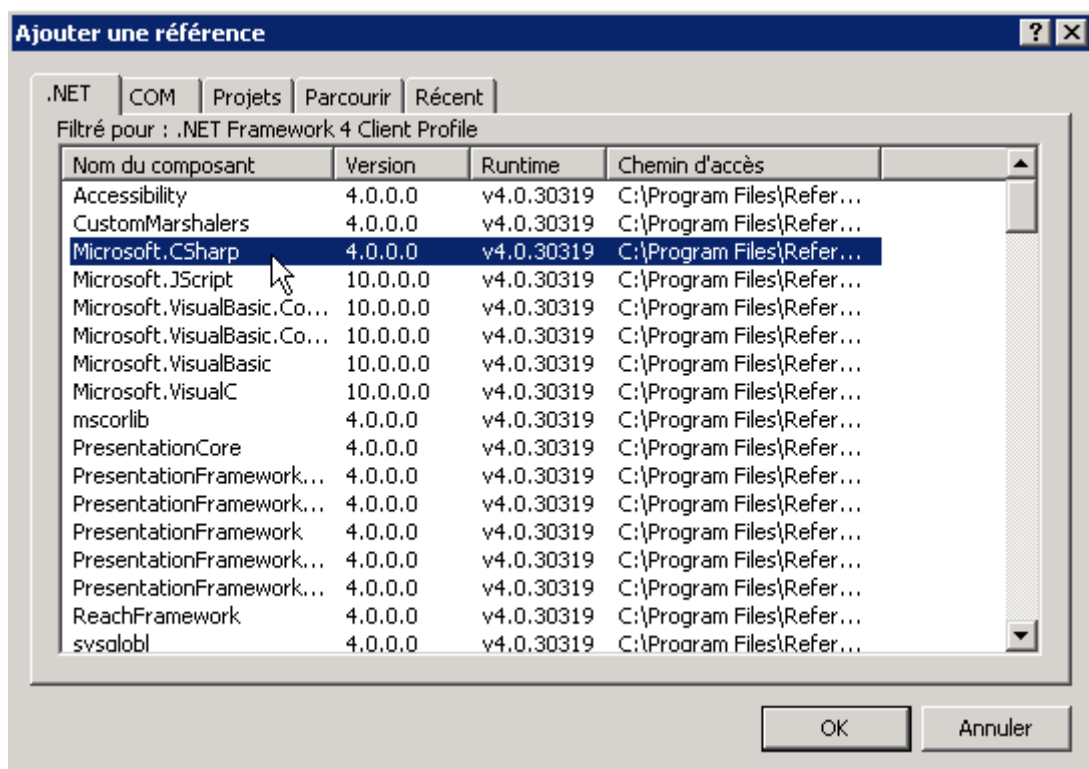


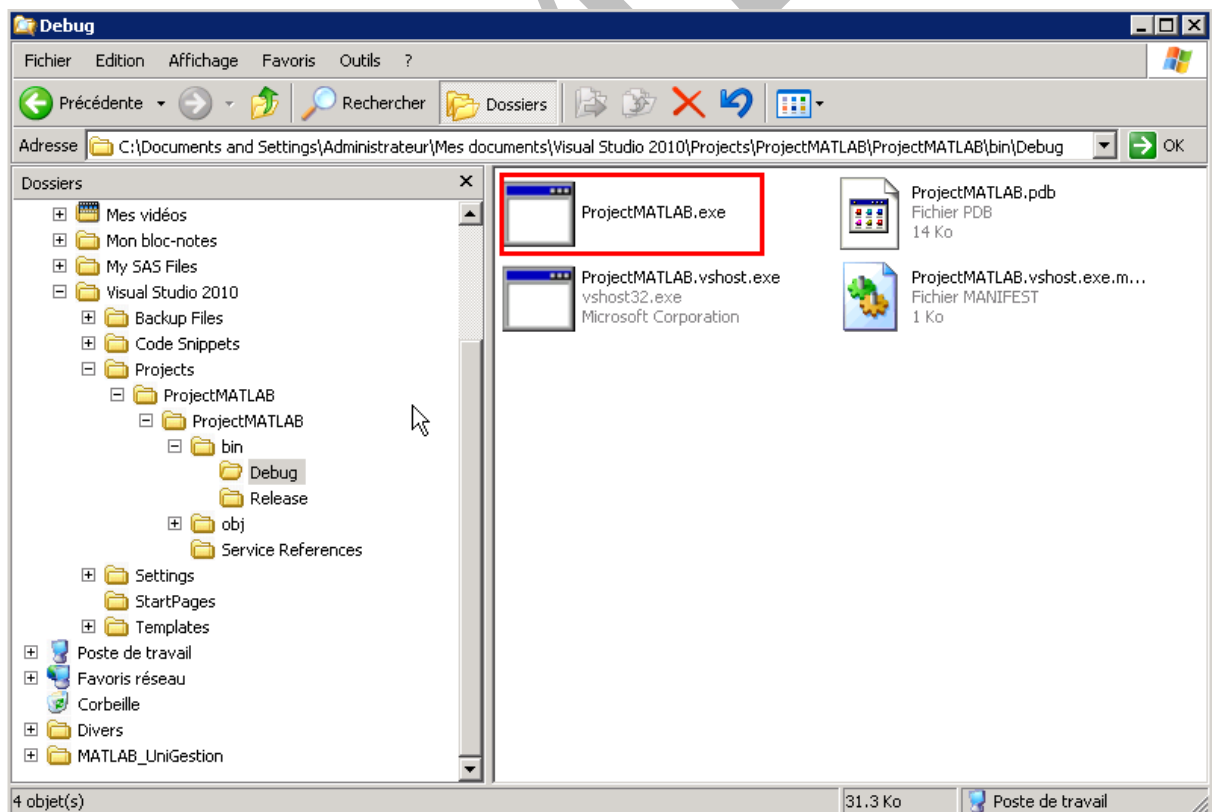
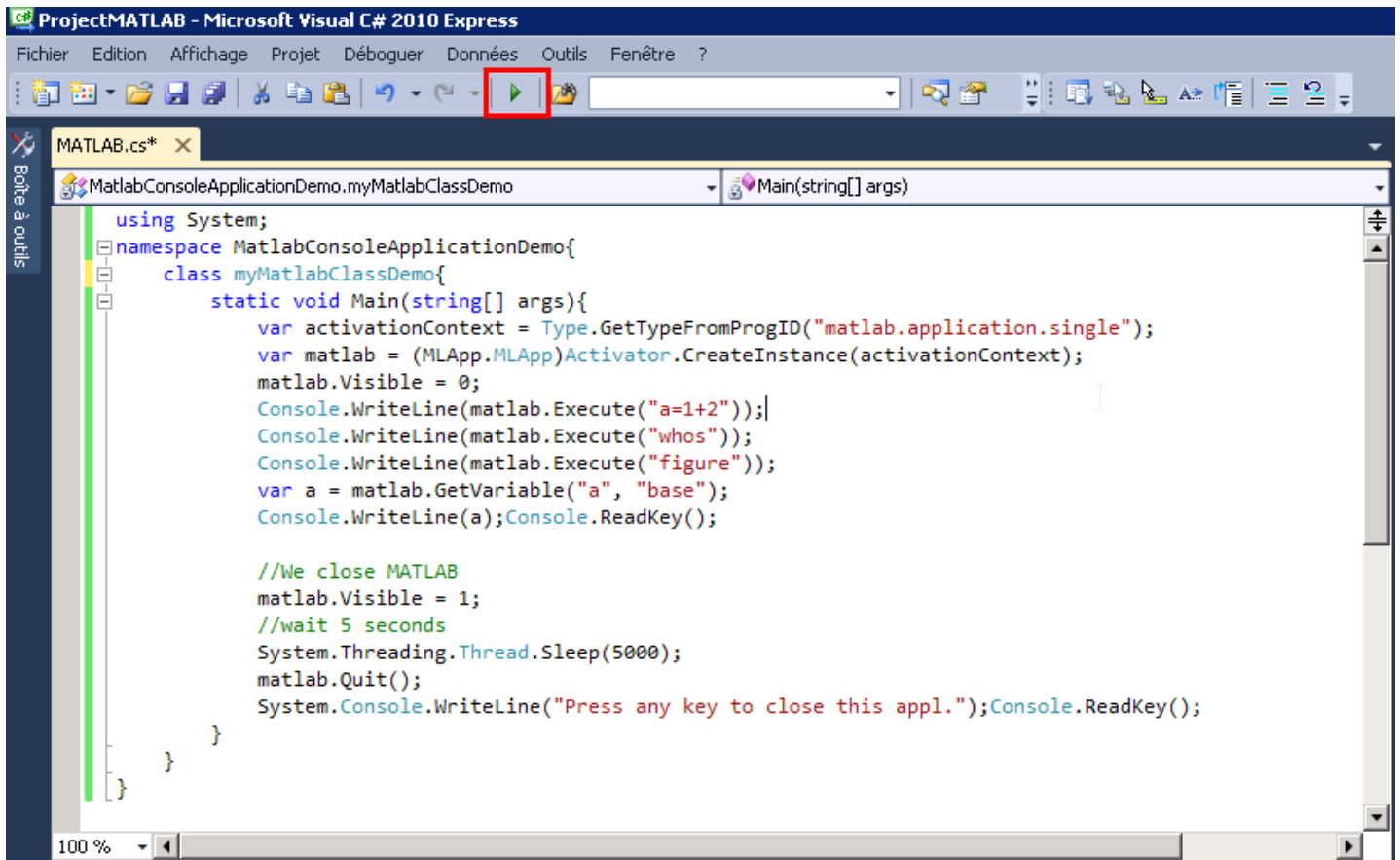


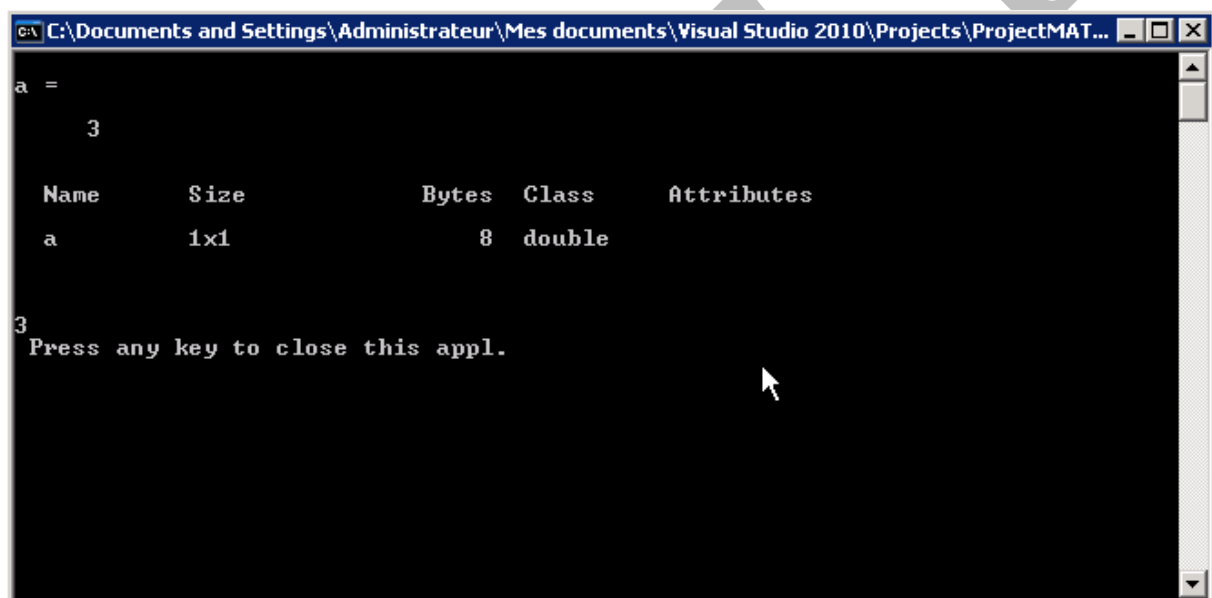
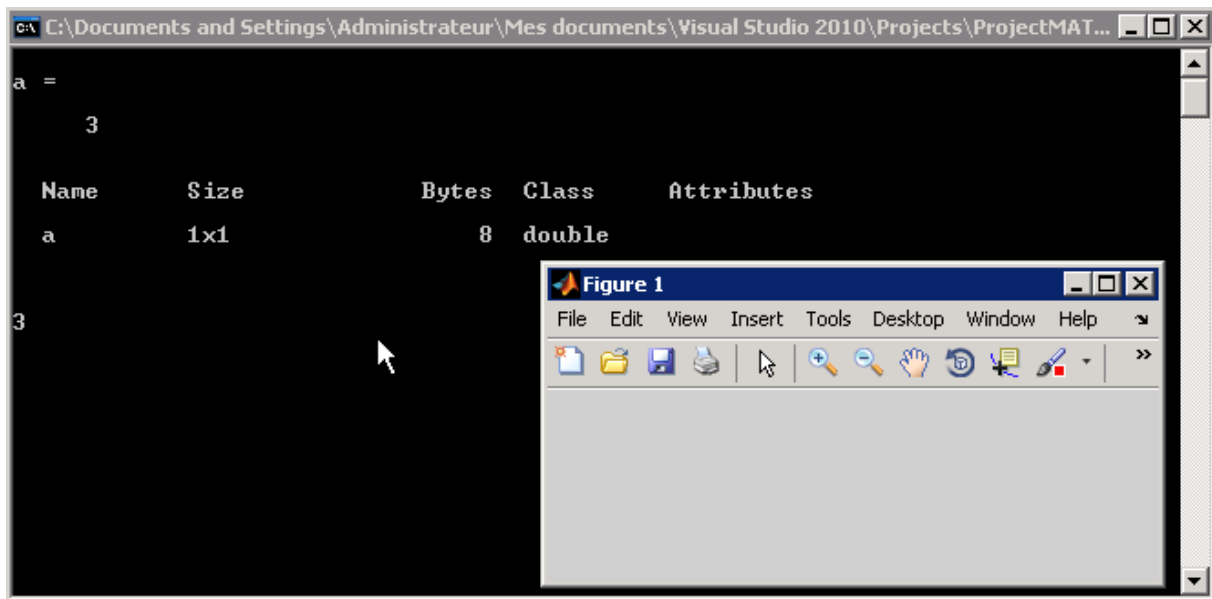


COM technology cons: MATLAB is required to be installed on the target computer.

MATLAB .NET builder compiles your MATLAB code to the .NET assembly and you can use it directly.

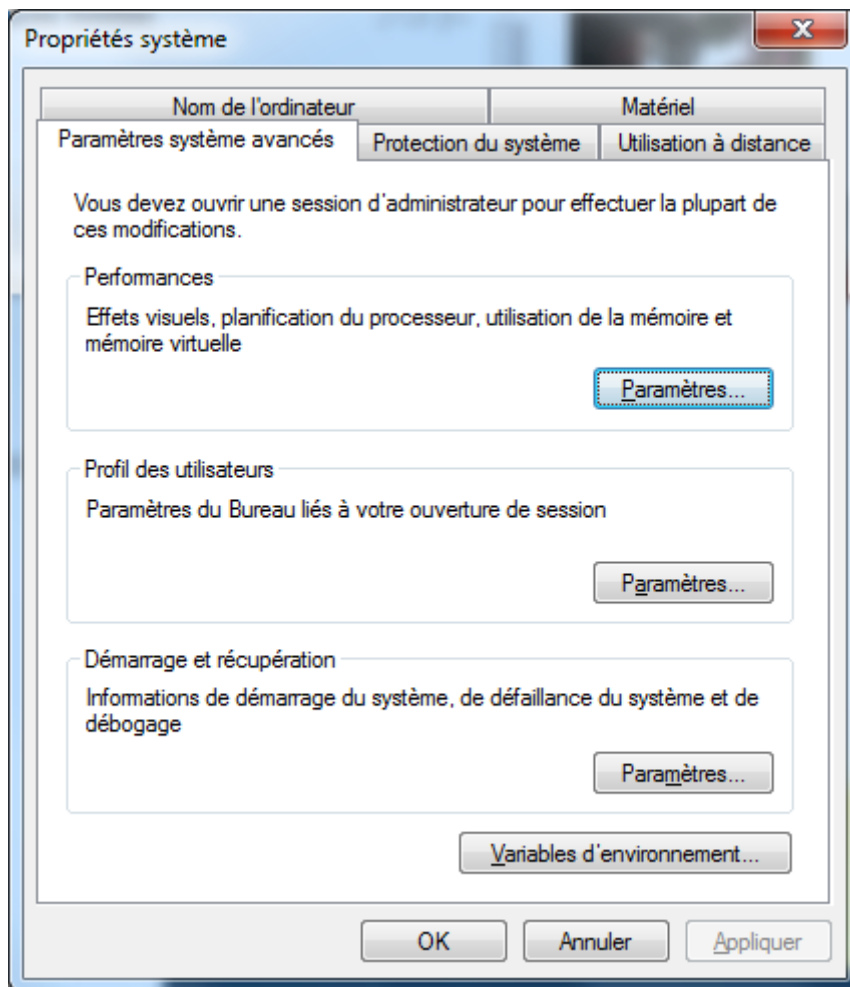


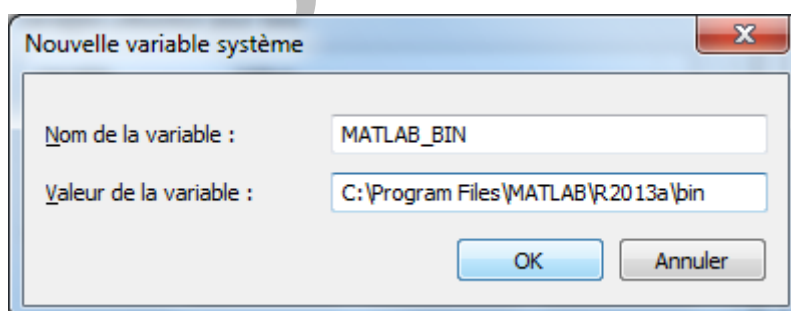
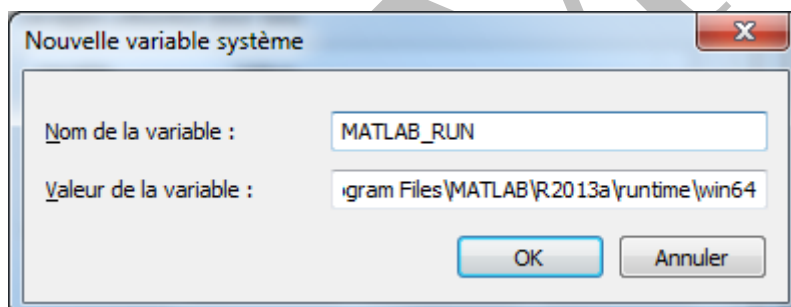
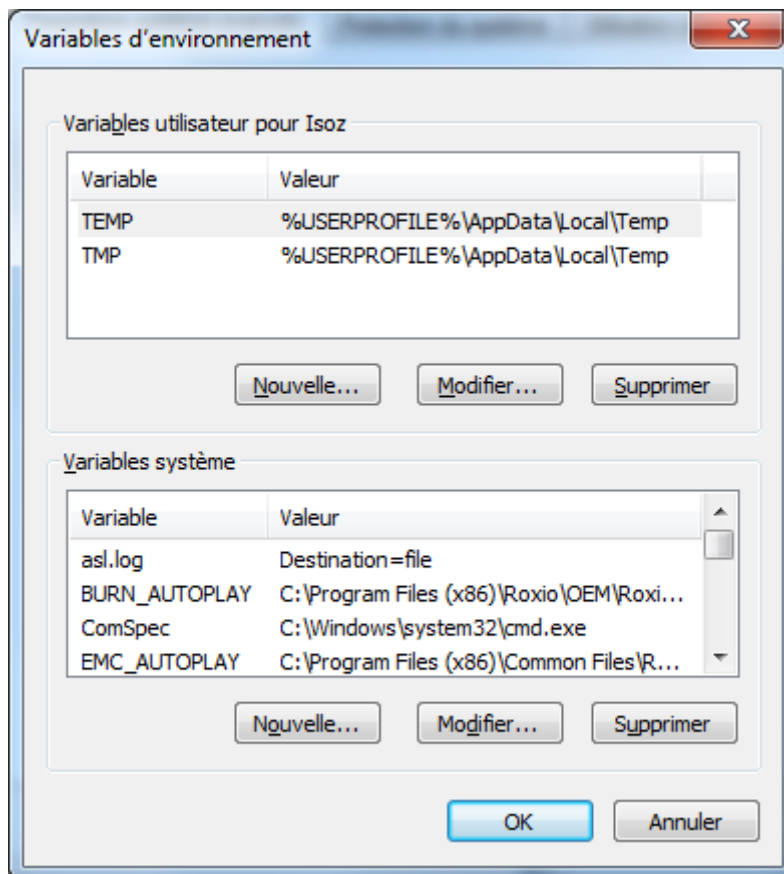


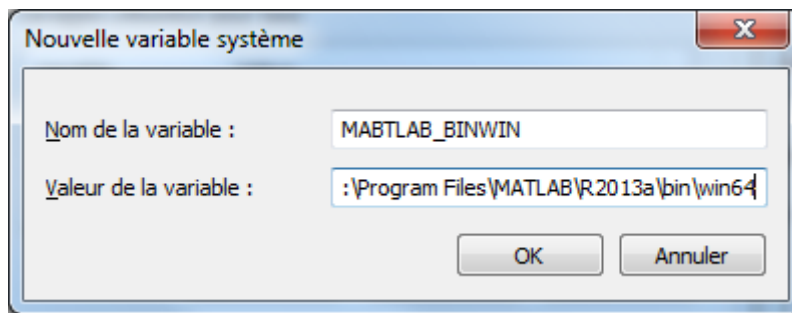


Call MATLAB from C++ (VS Express 2010 x64)

Some steps will seem to be duplicates when you will read what follows but in the facts it is just a cumulated return on experience troubleshooting manipulations to avoid any problems :

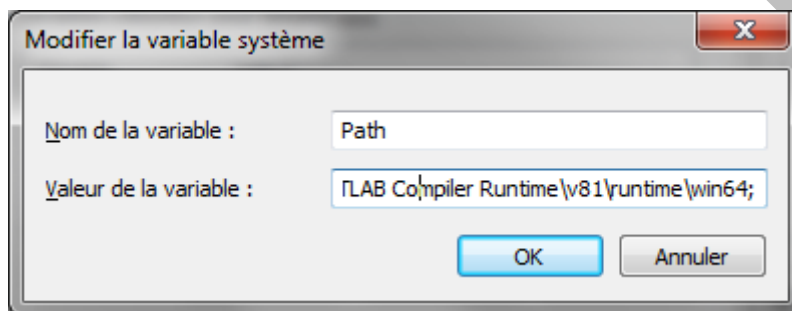




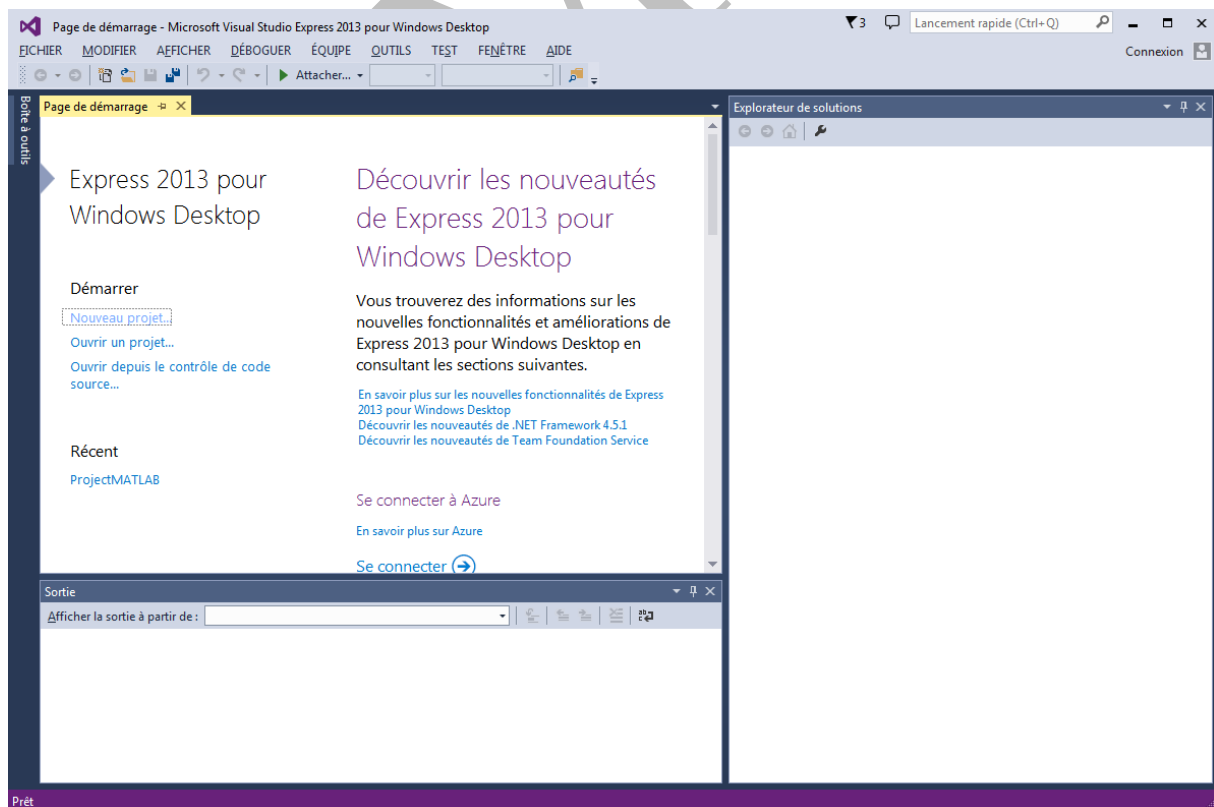
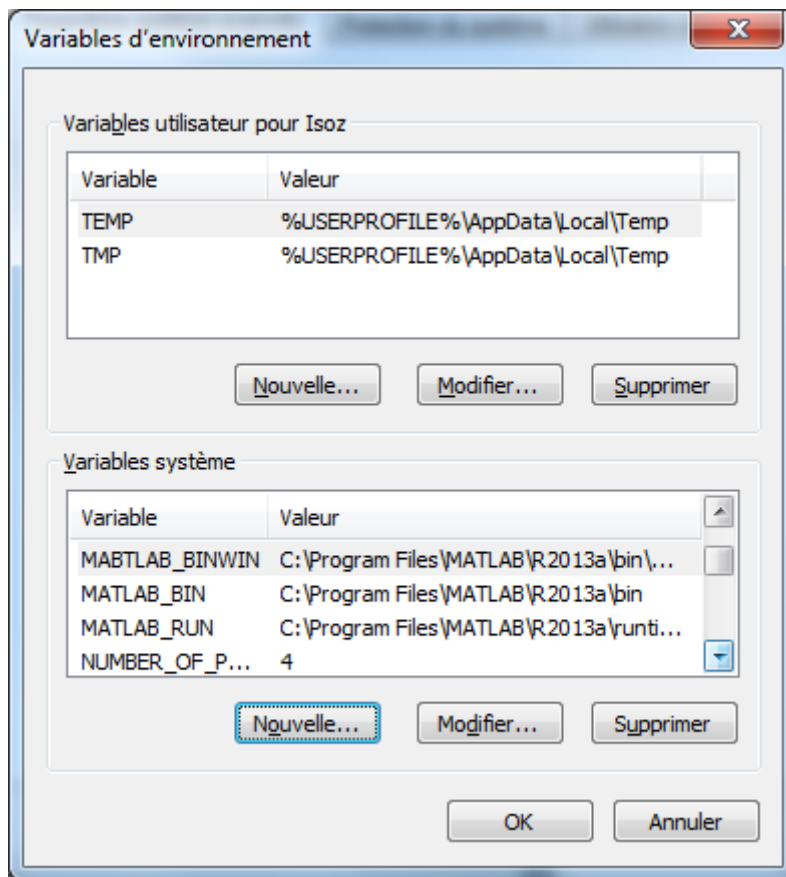


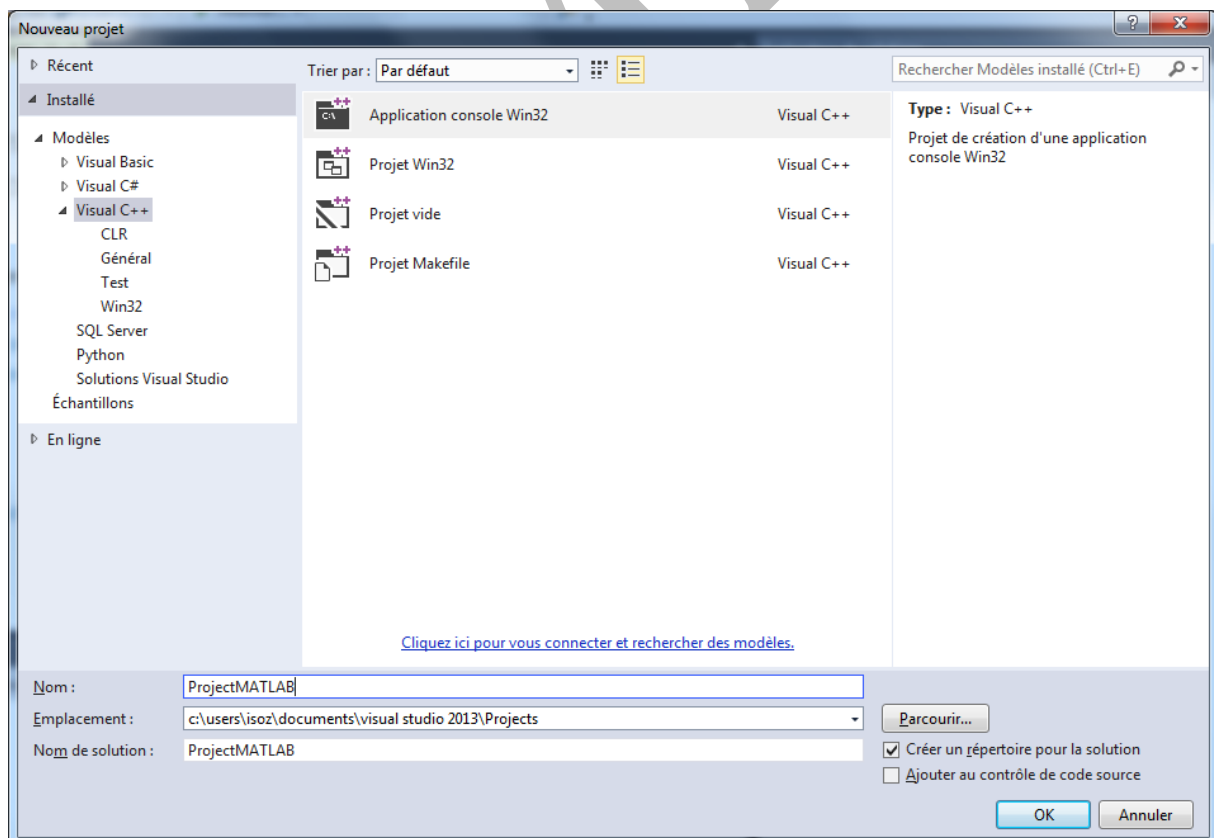
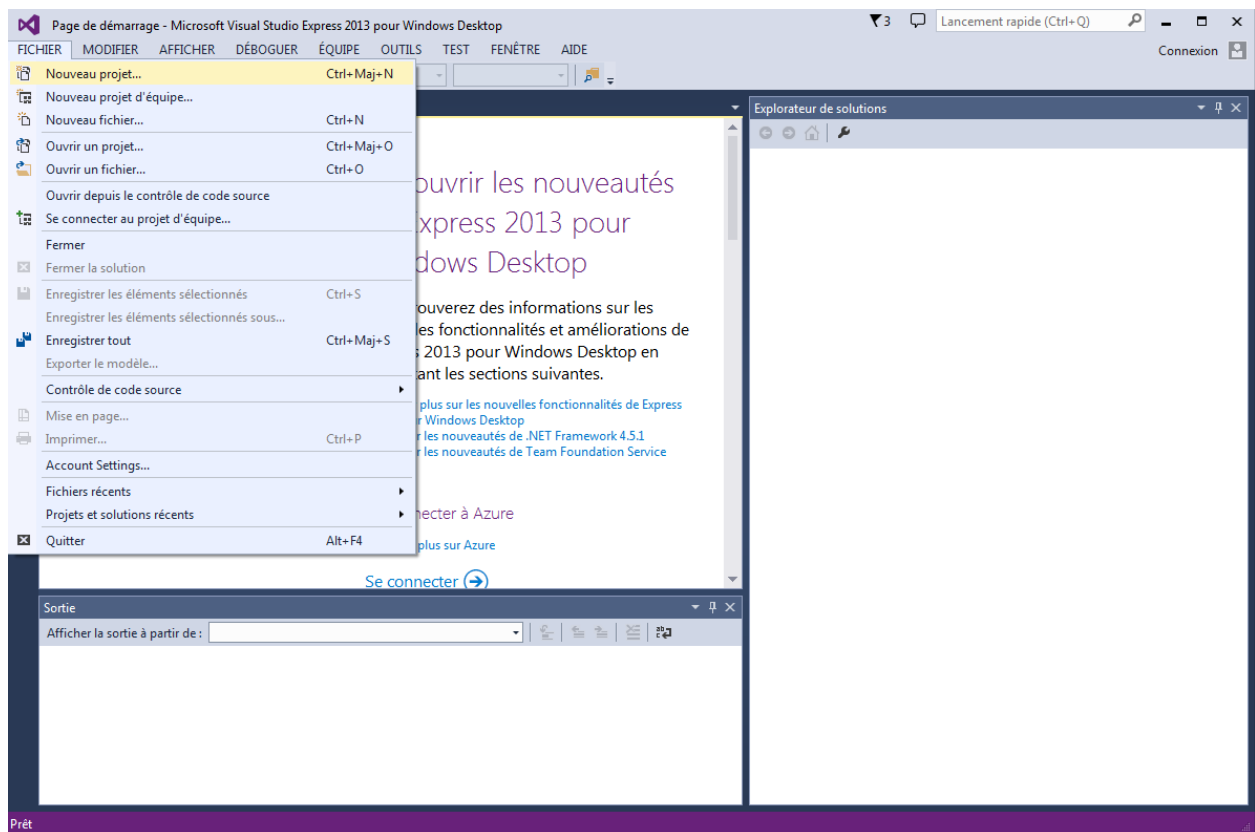
And check that in PATH you have anywhere :

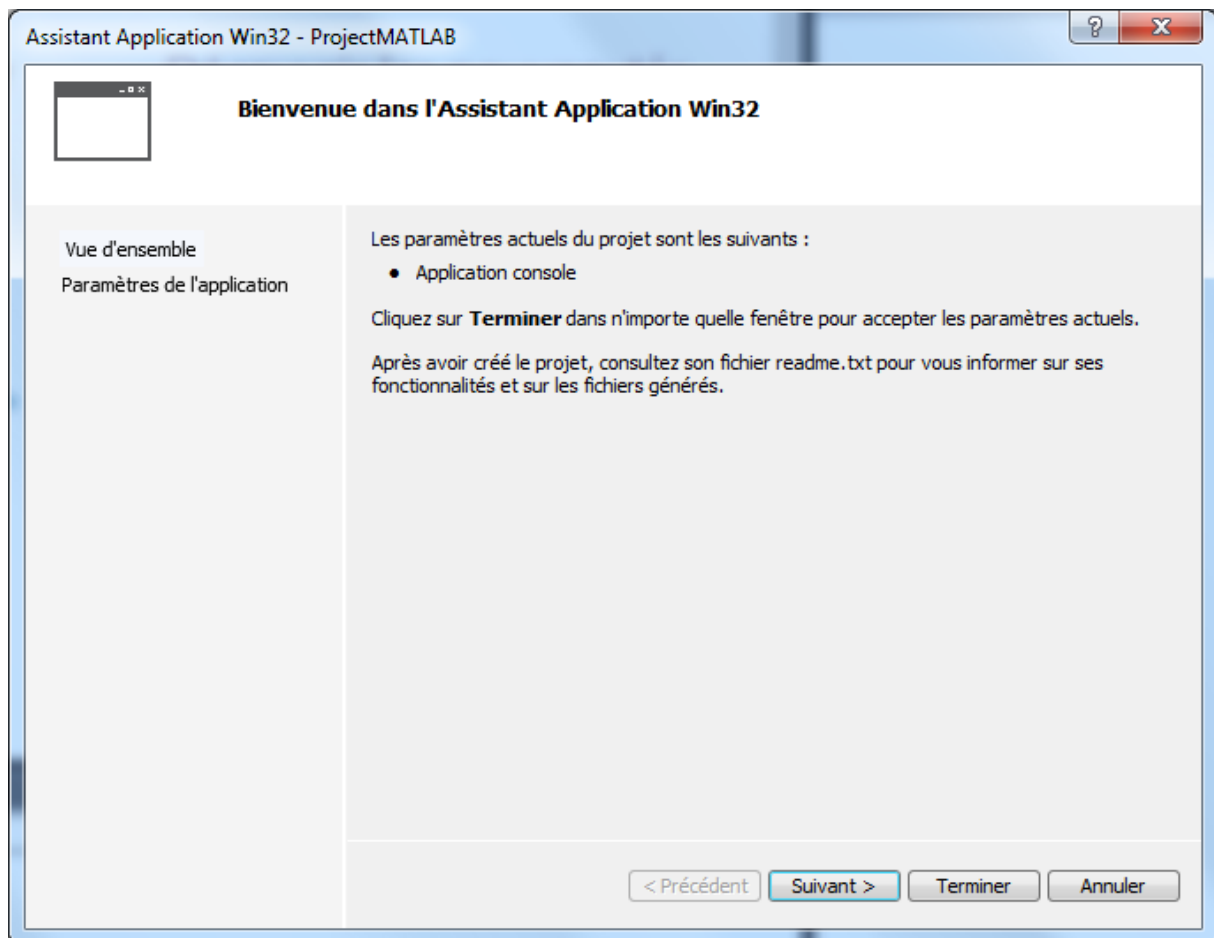
C:\\Program Files\\MATLAB\\MATLAB Compiler Runtime\\v81\\bin\\win64

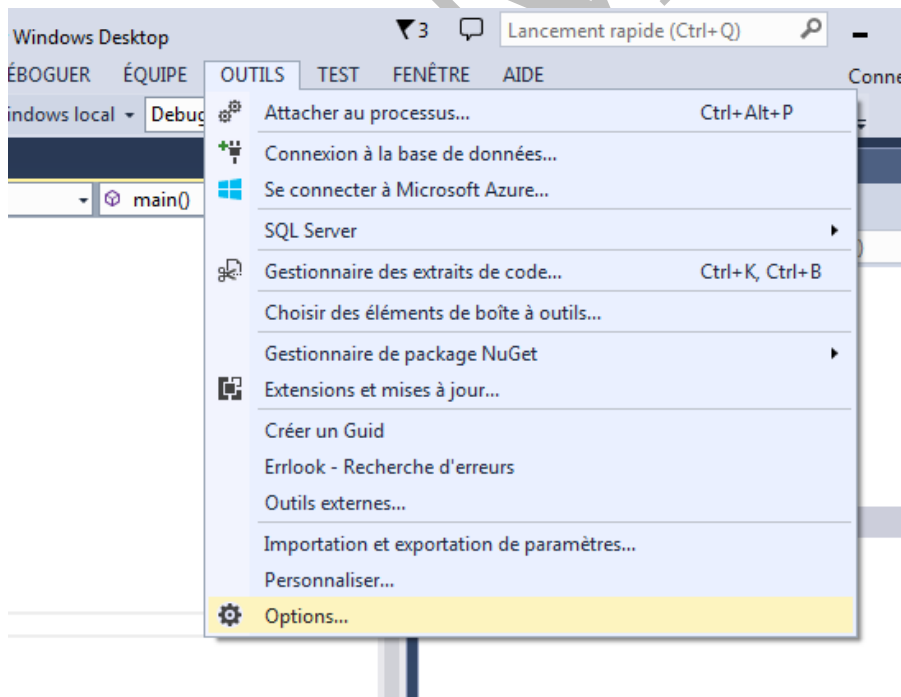
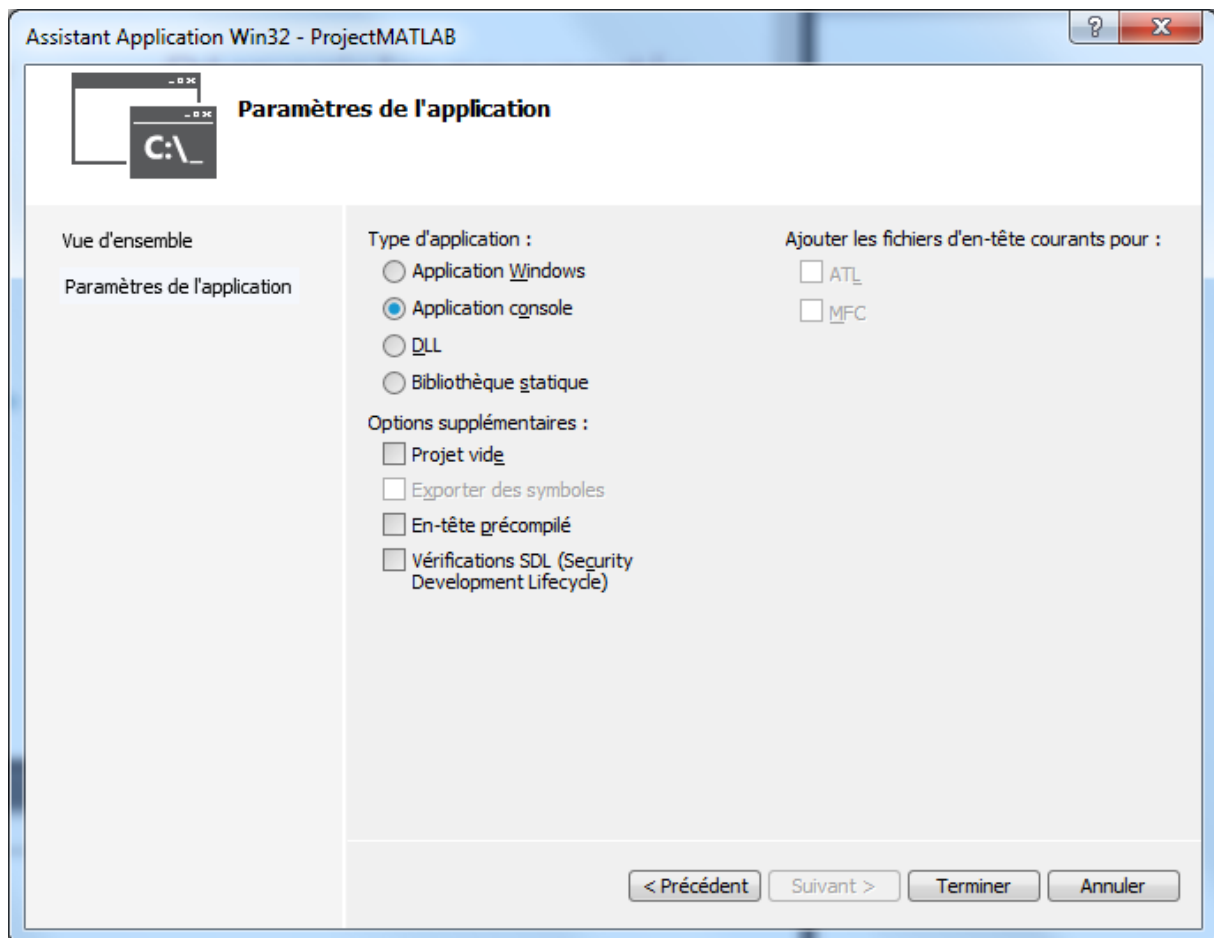


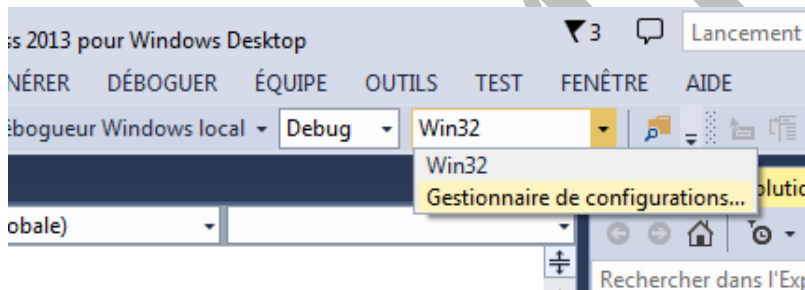
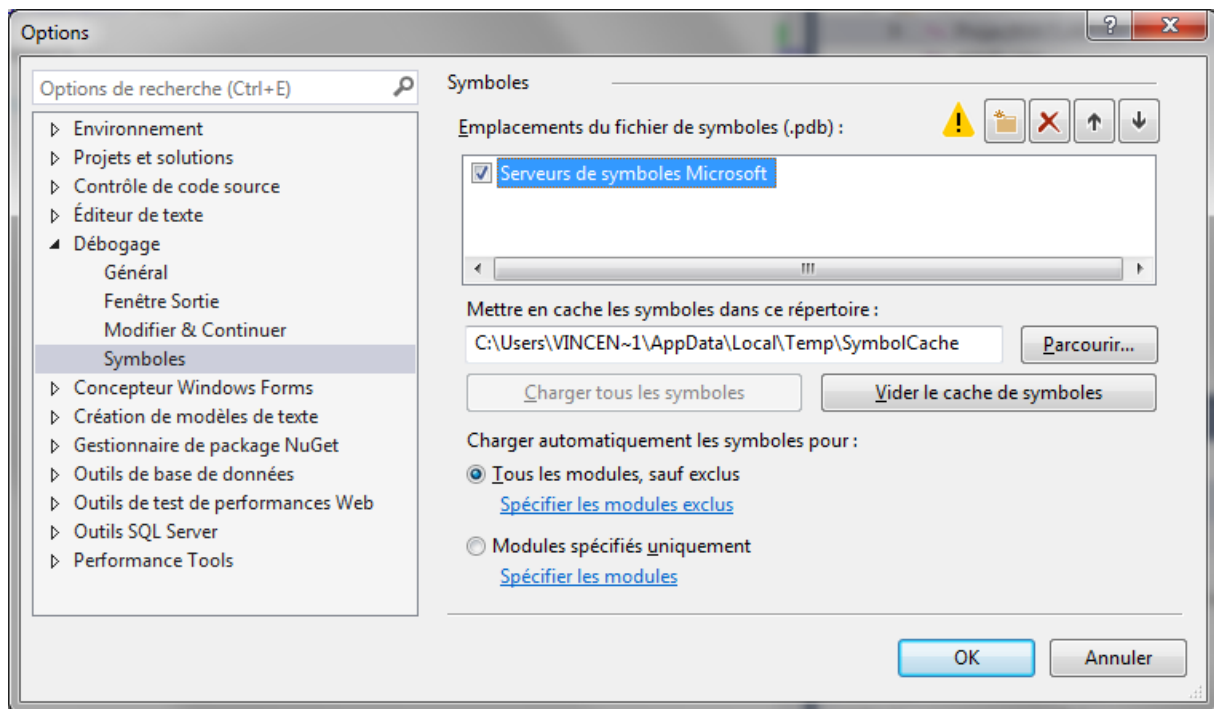
Restart Computer !

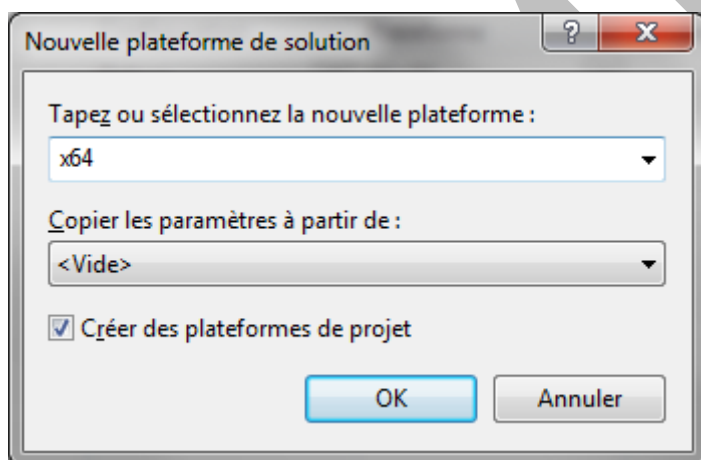
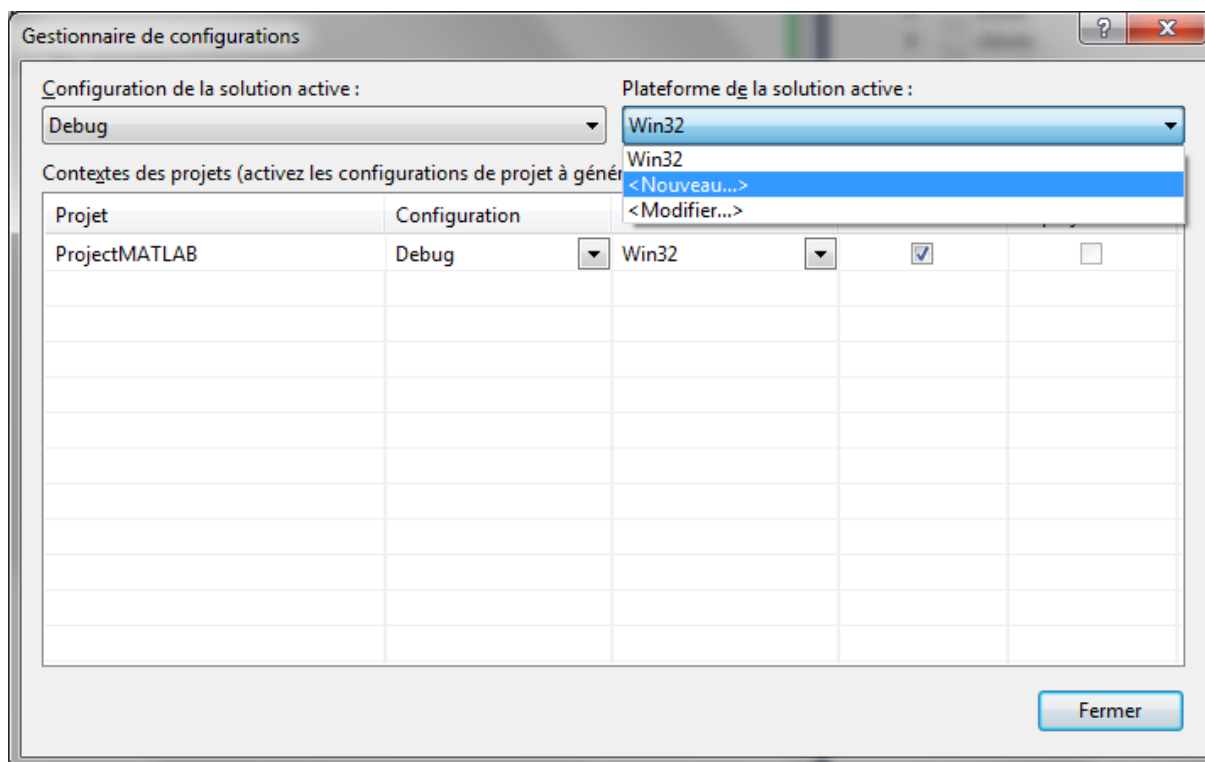


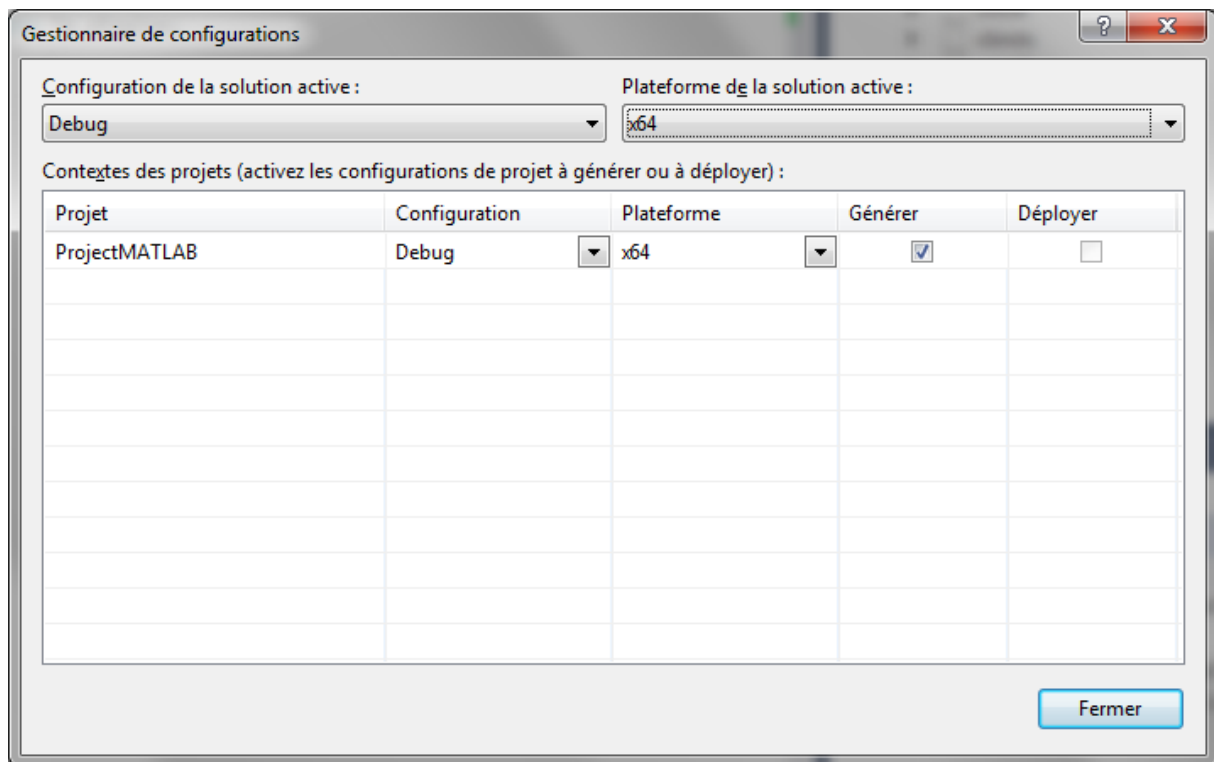


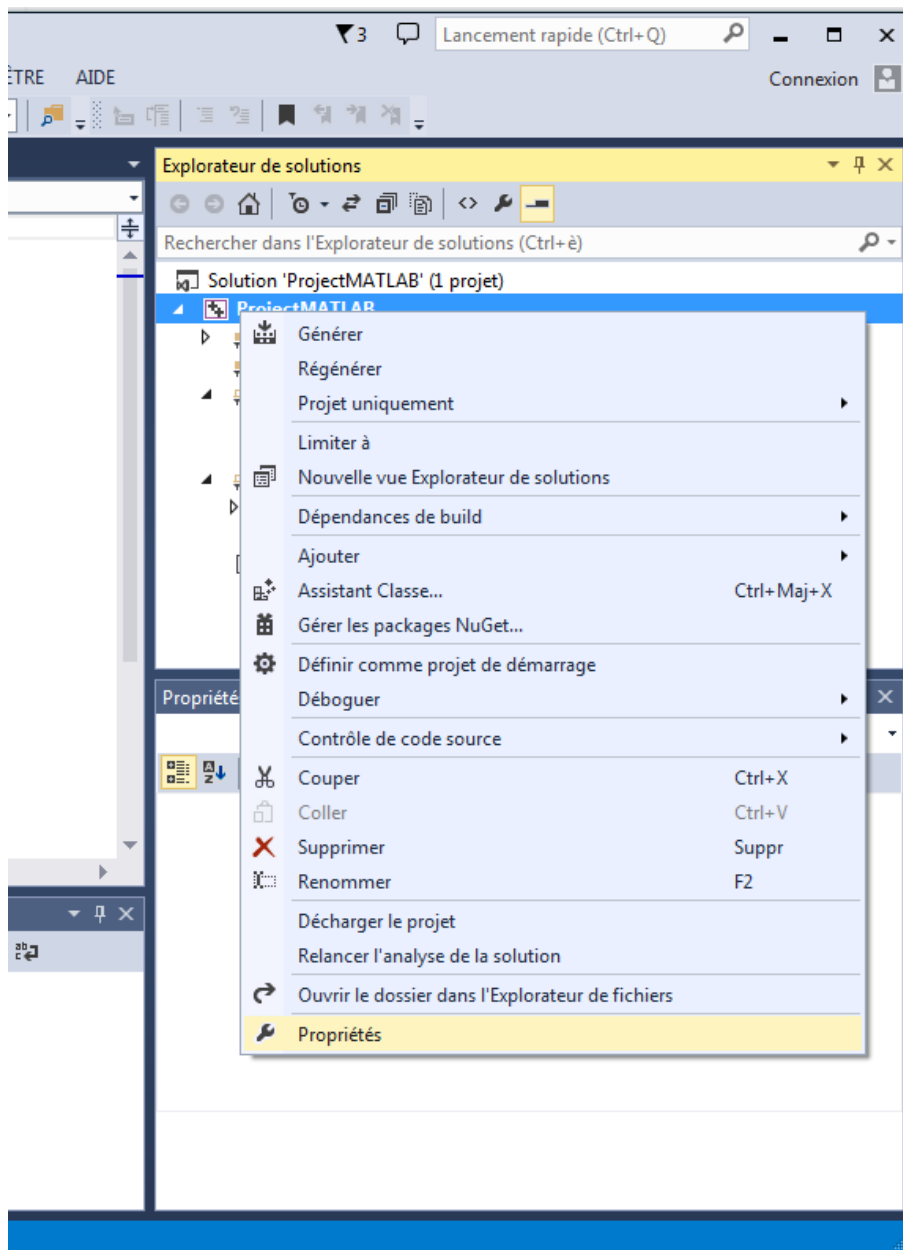


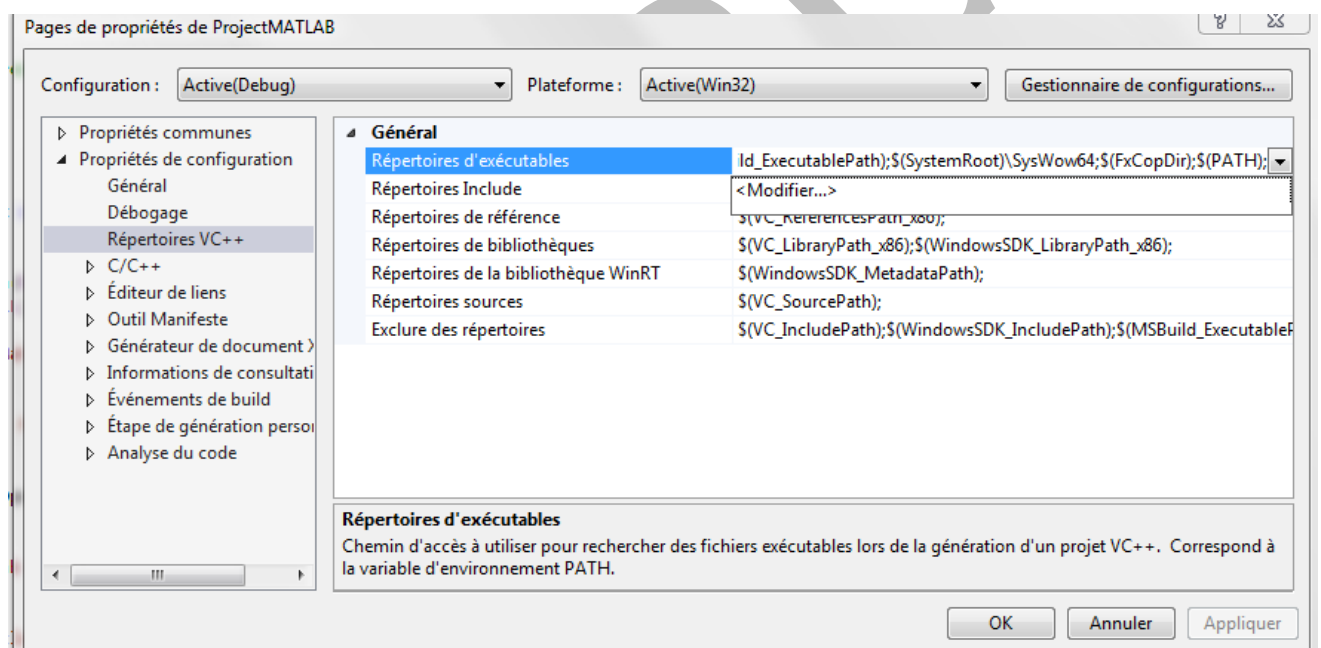
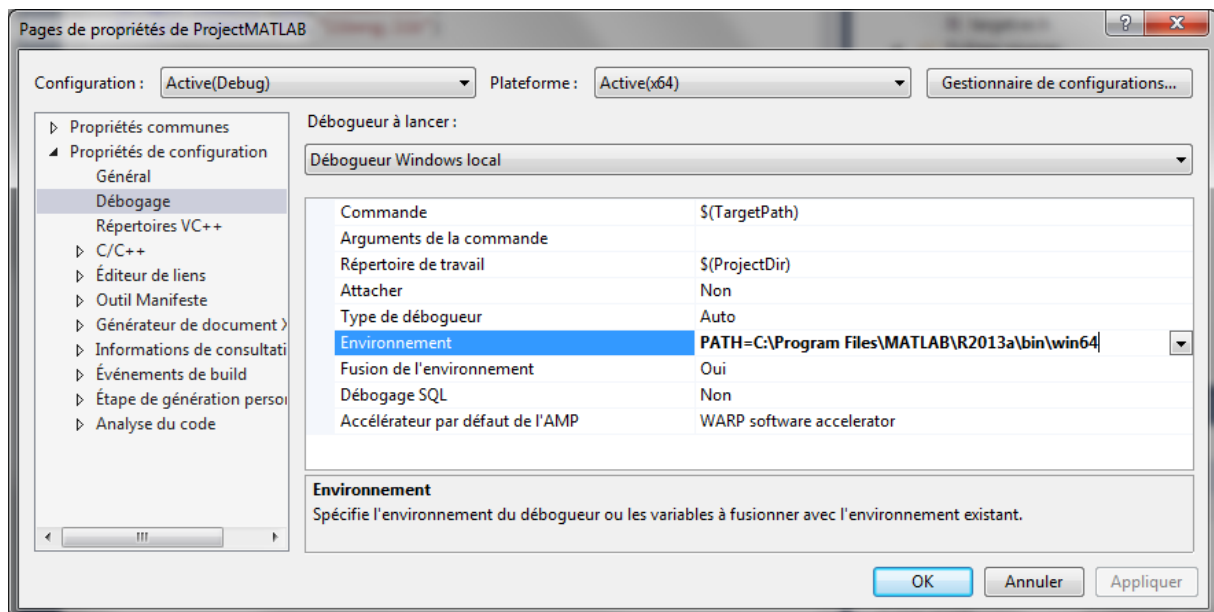


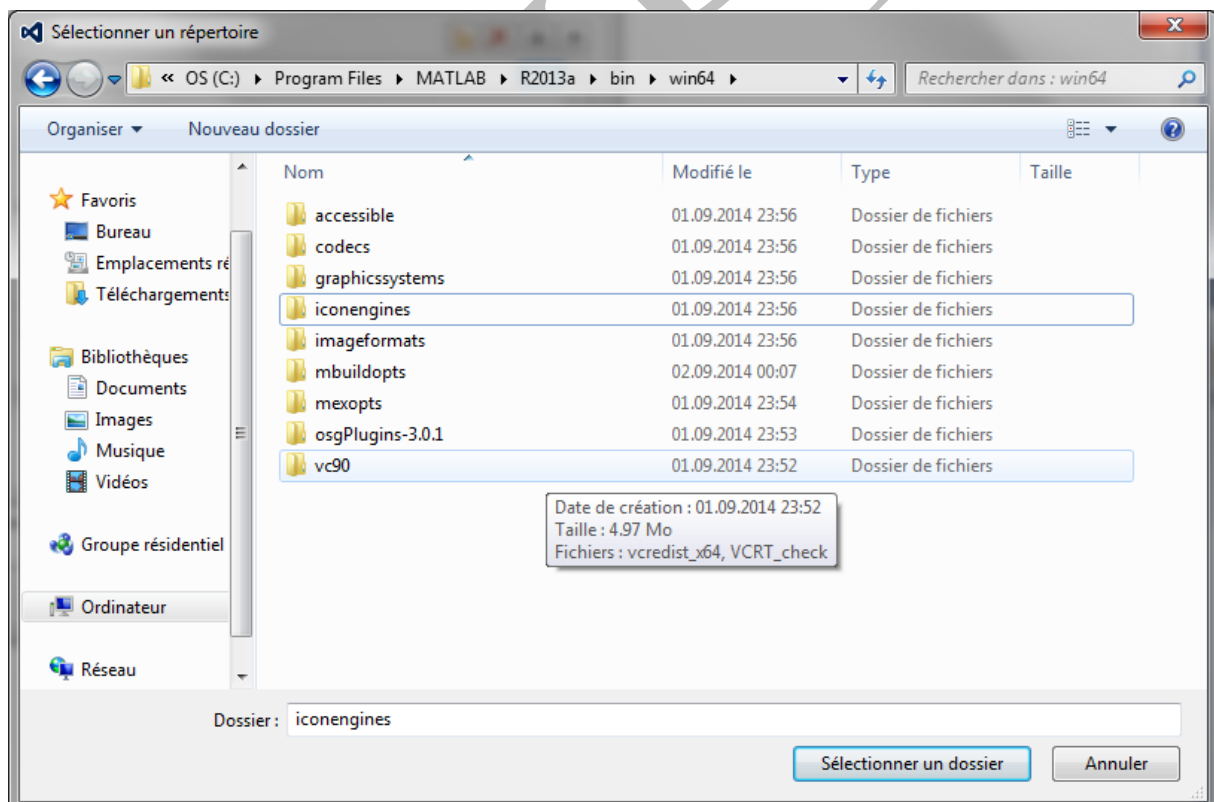
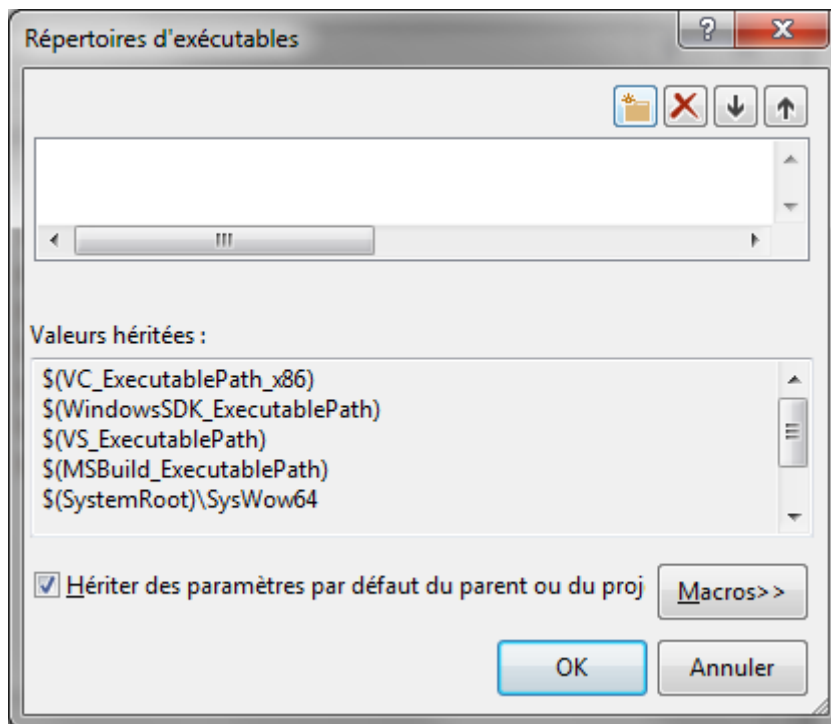


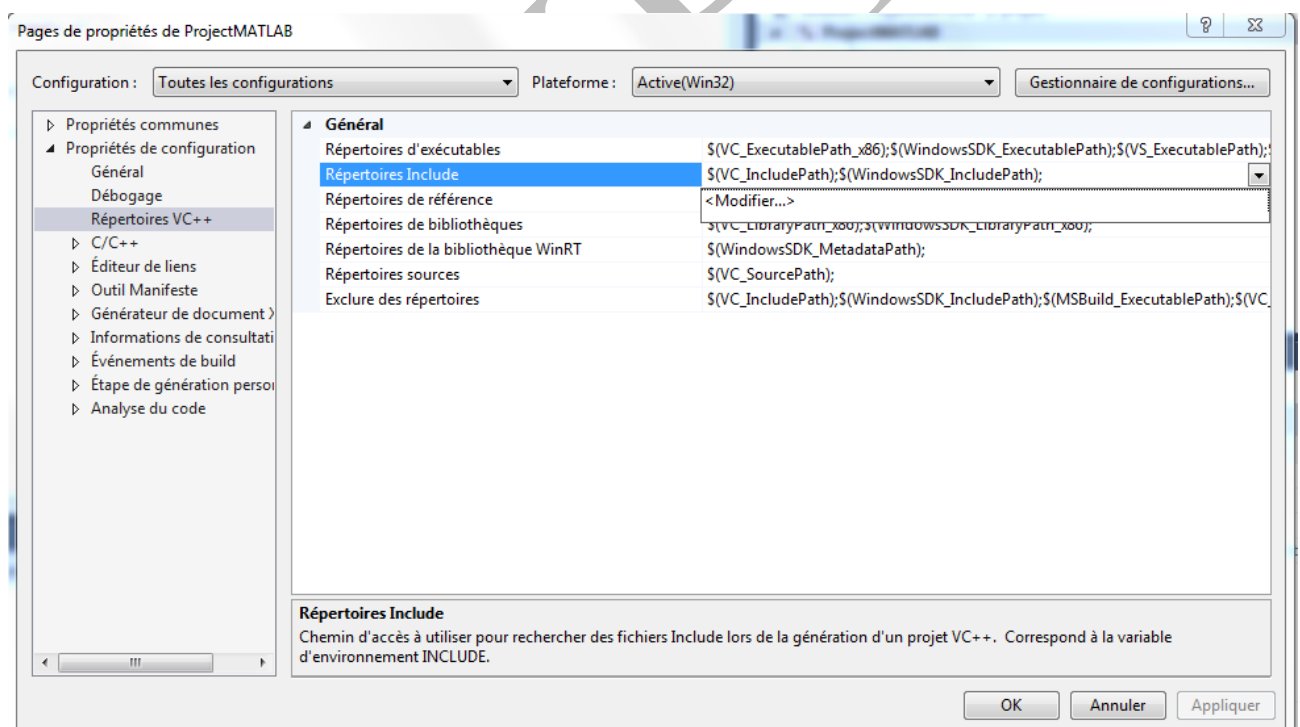
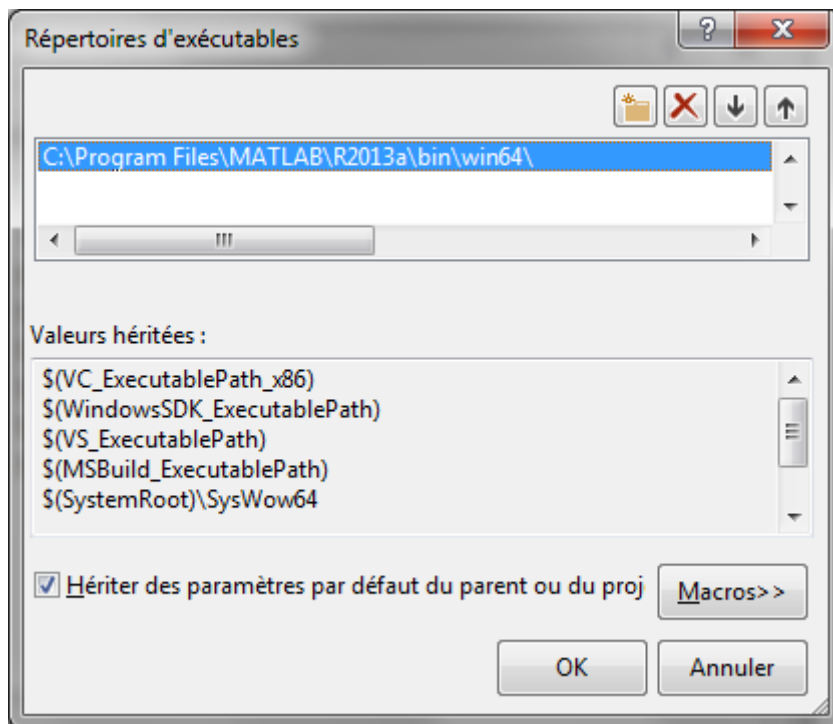


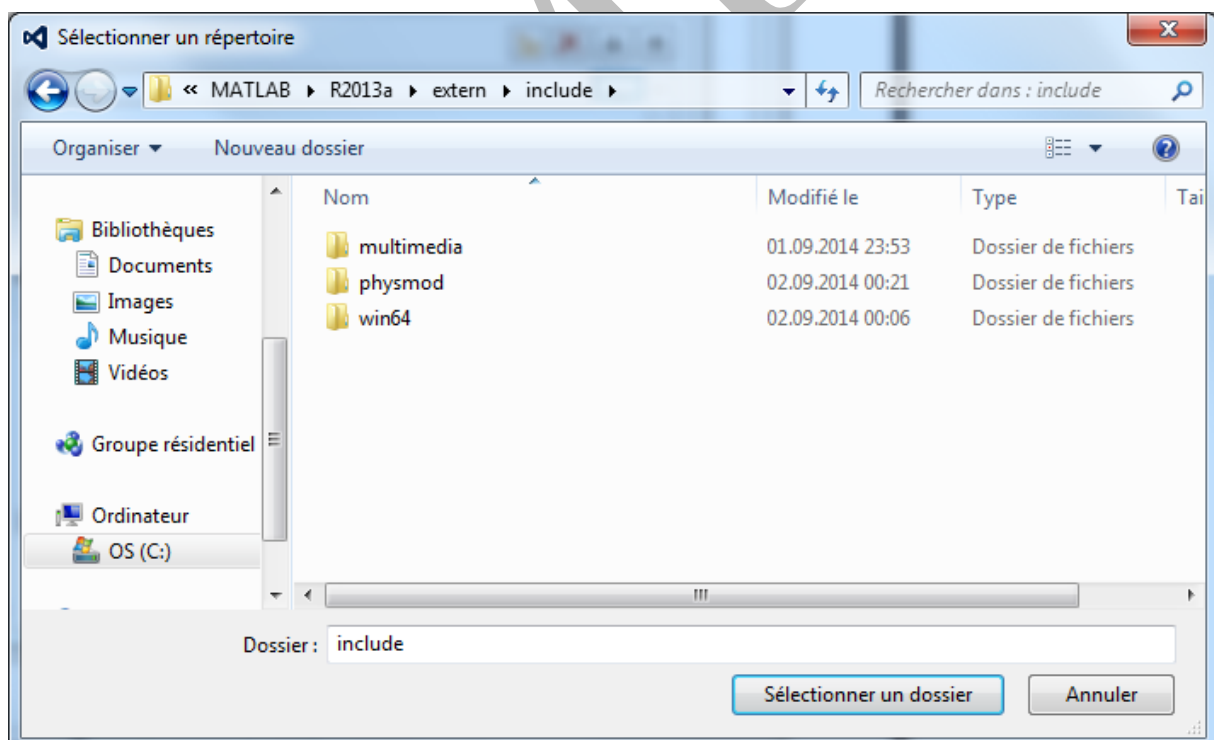
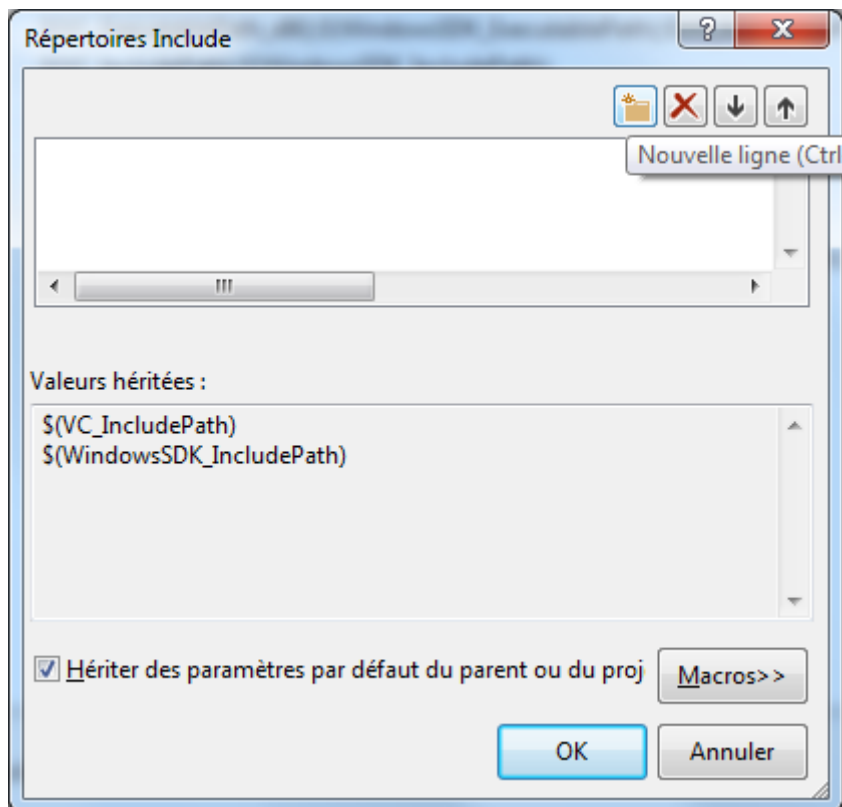


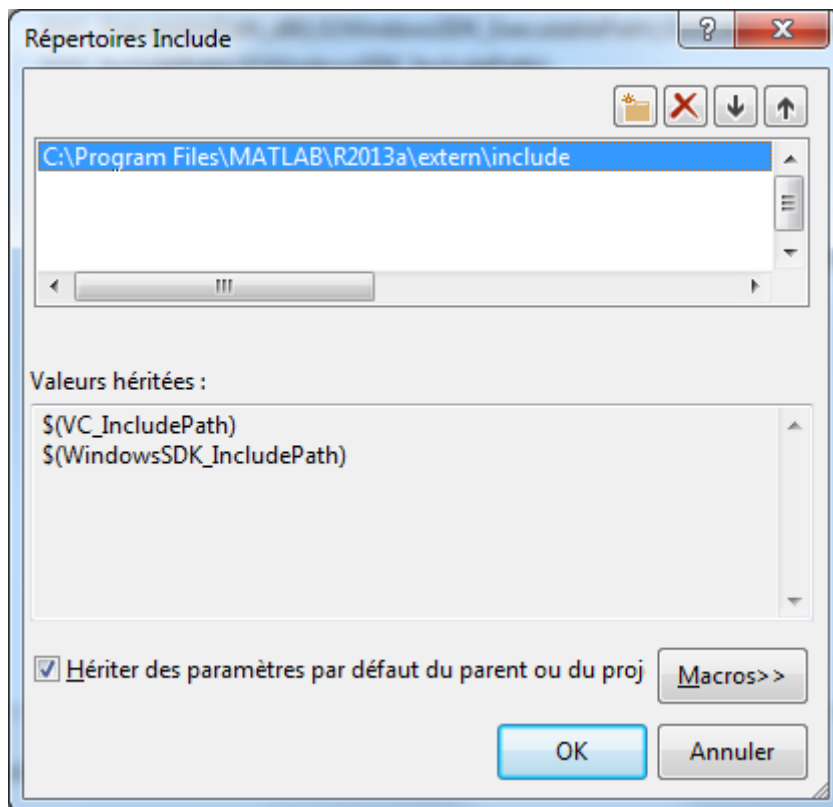




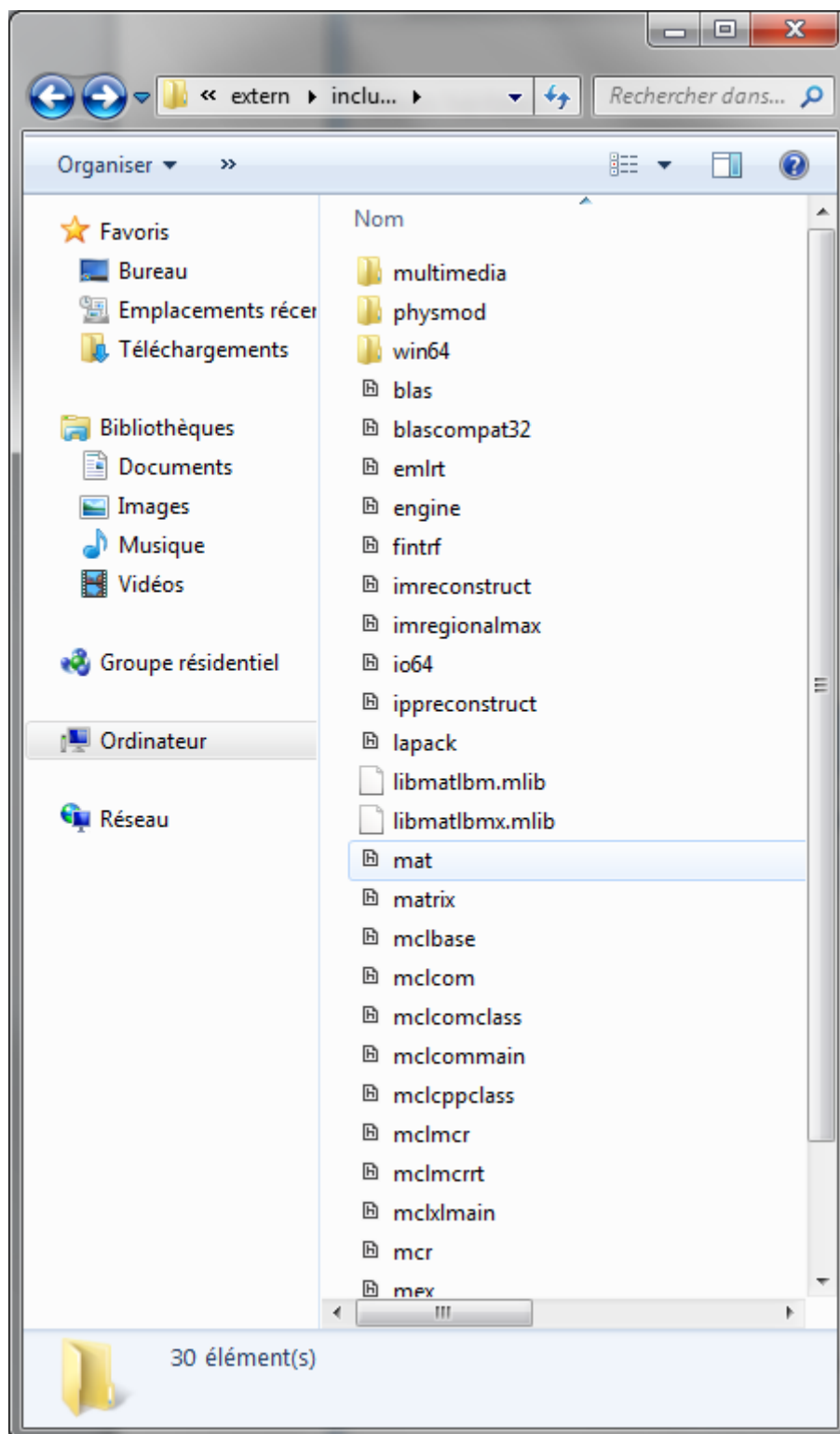


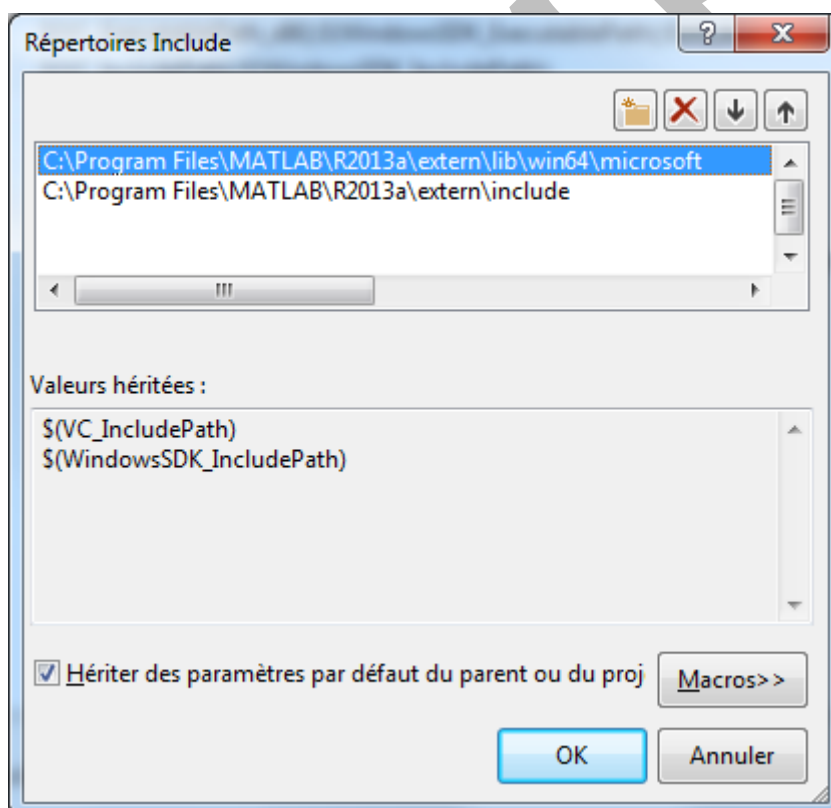
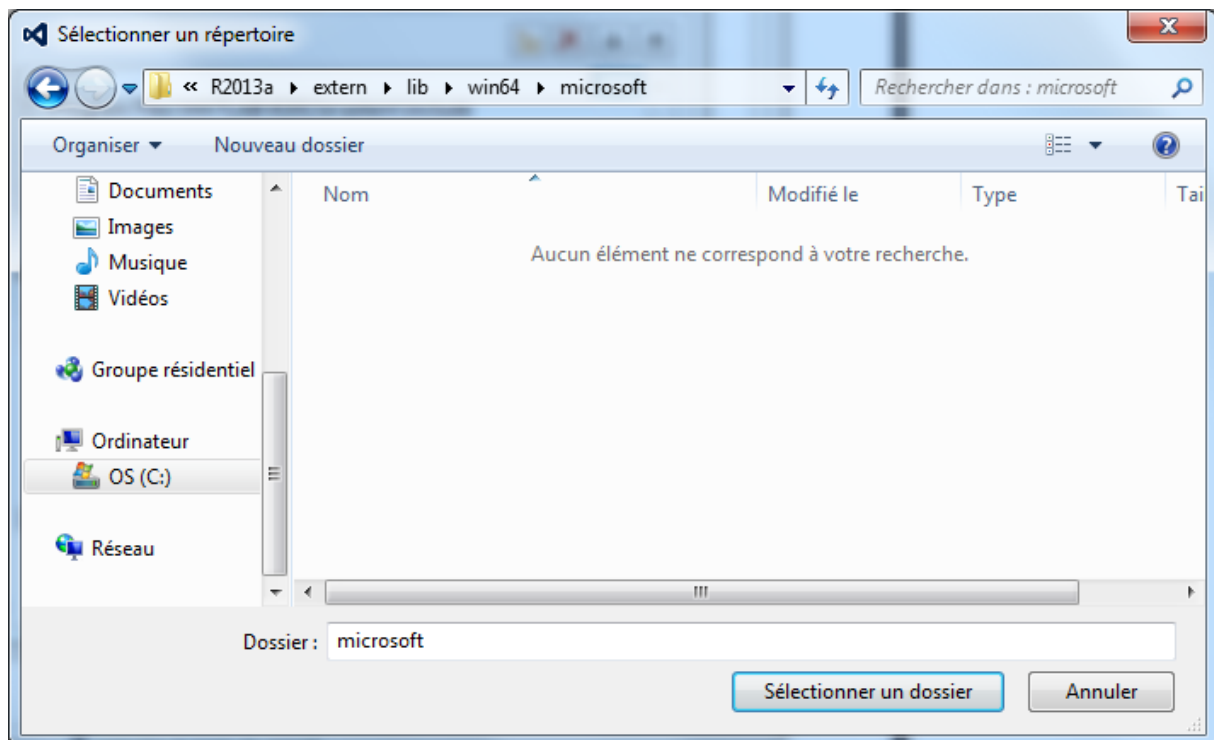




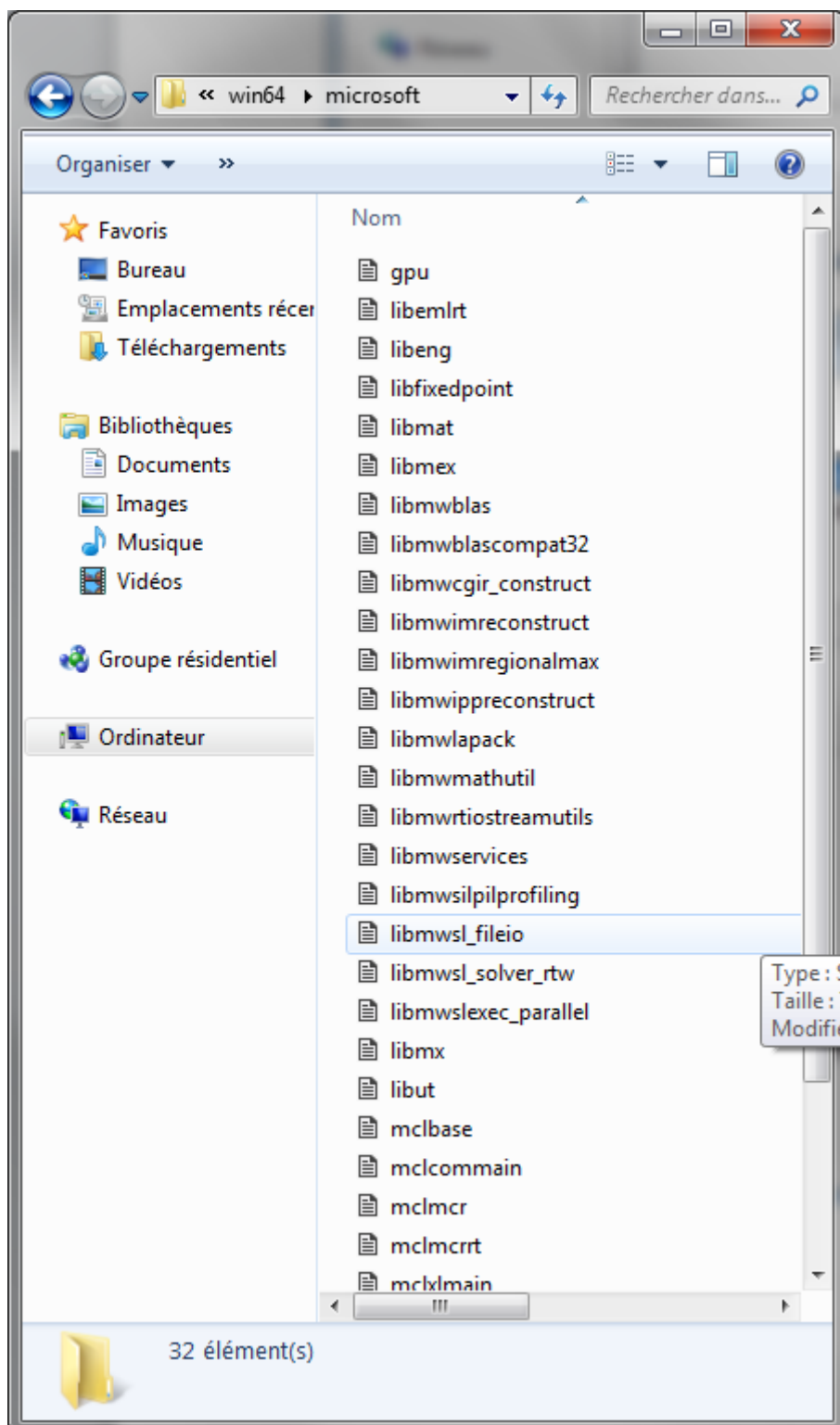


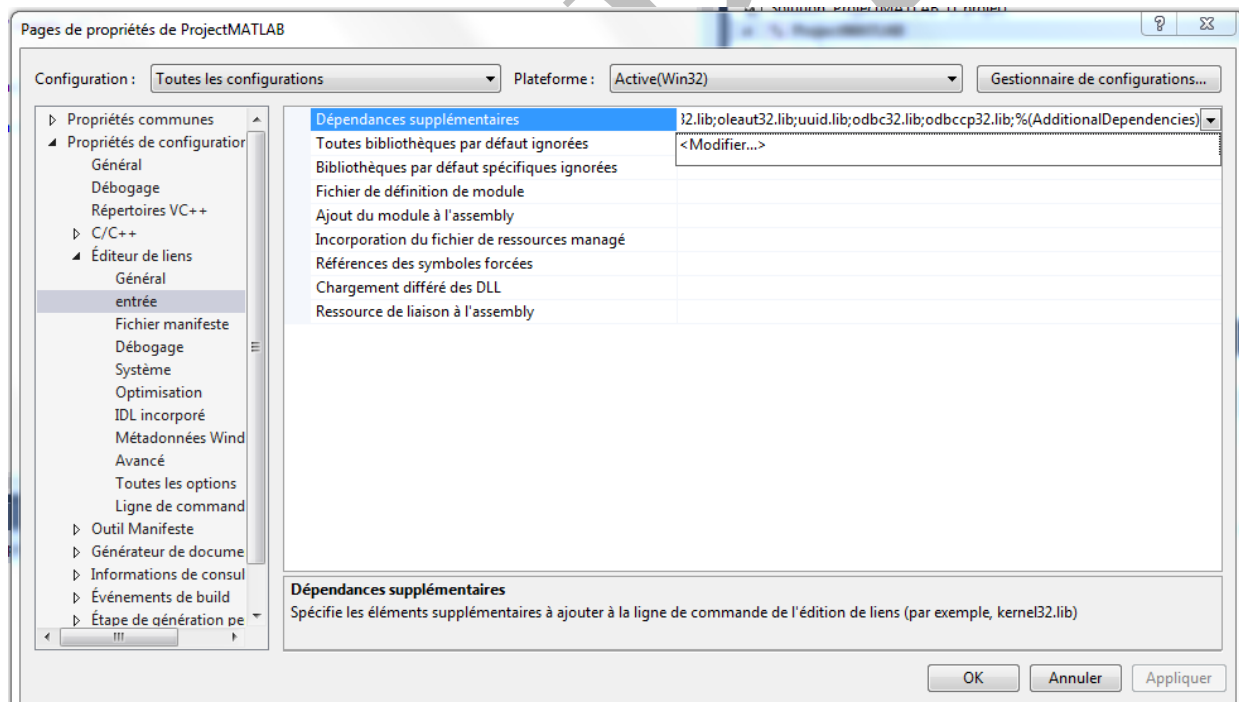
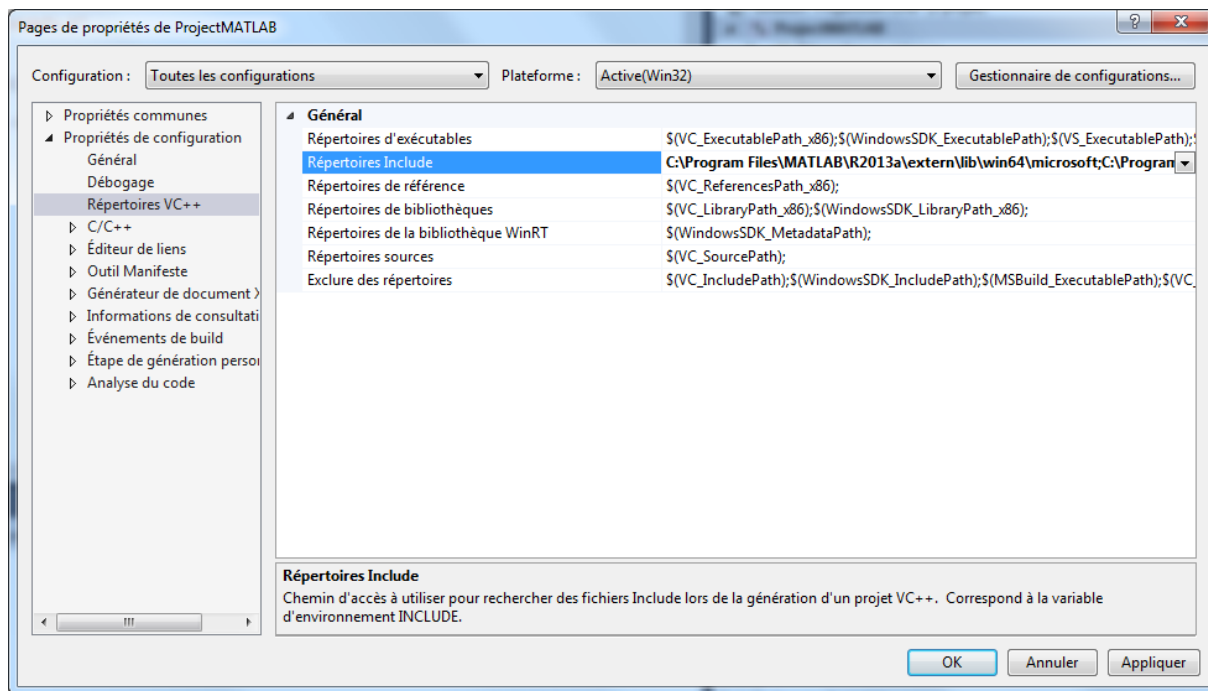
For info that's the content :

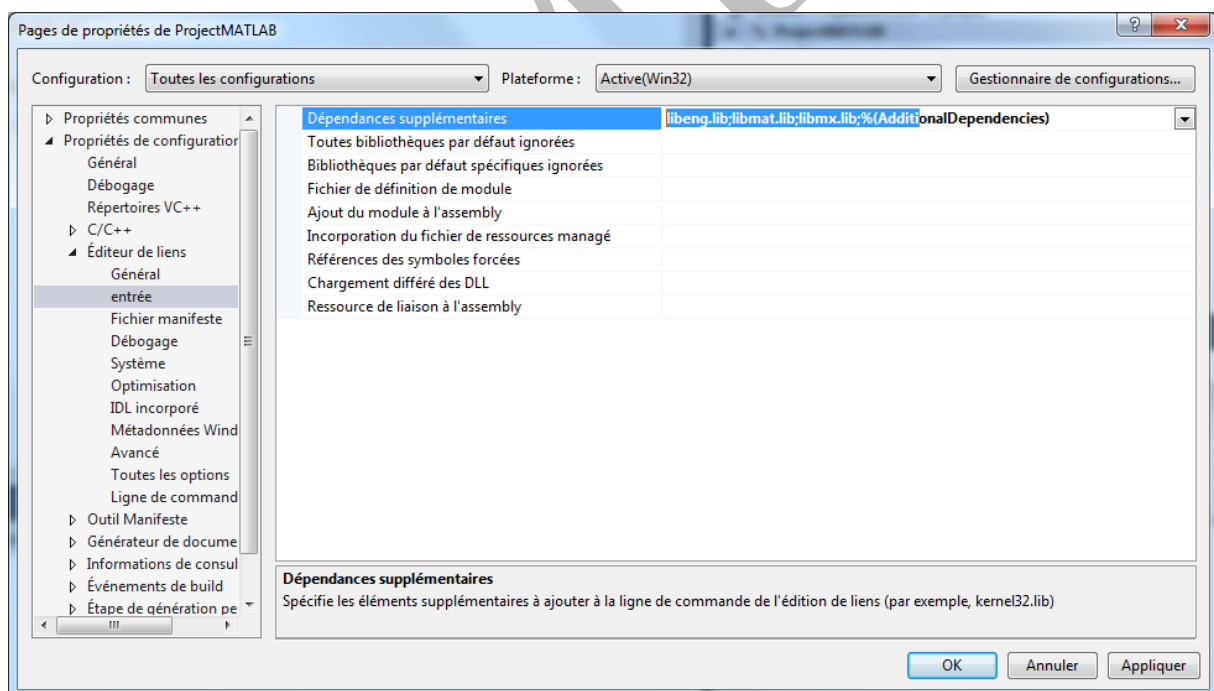
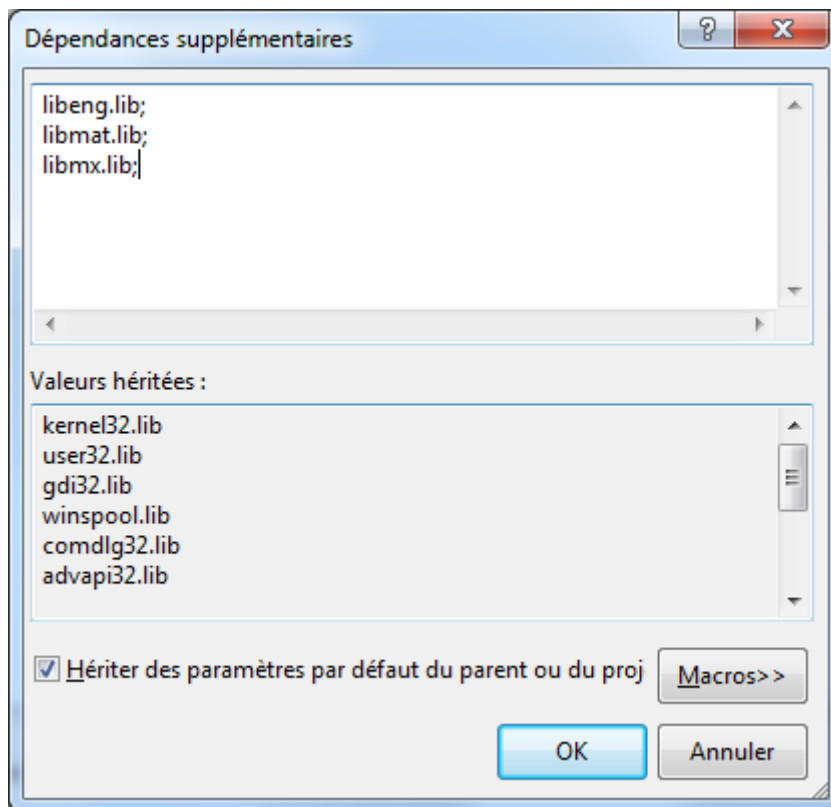


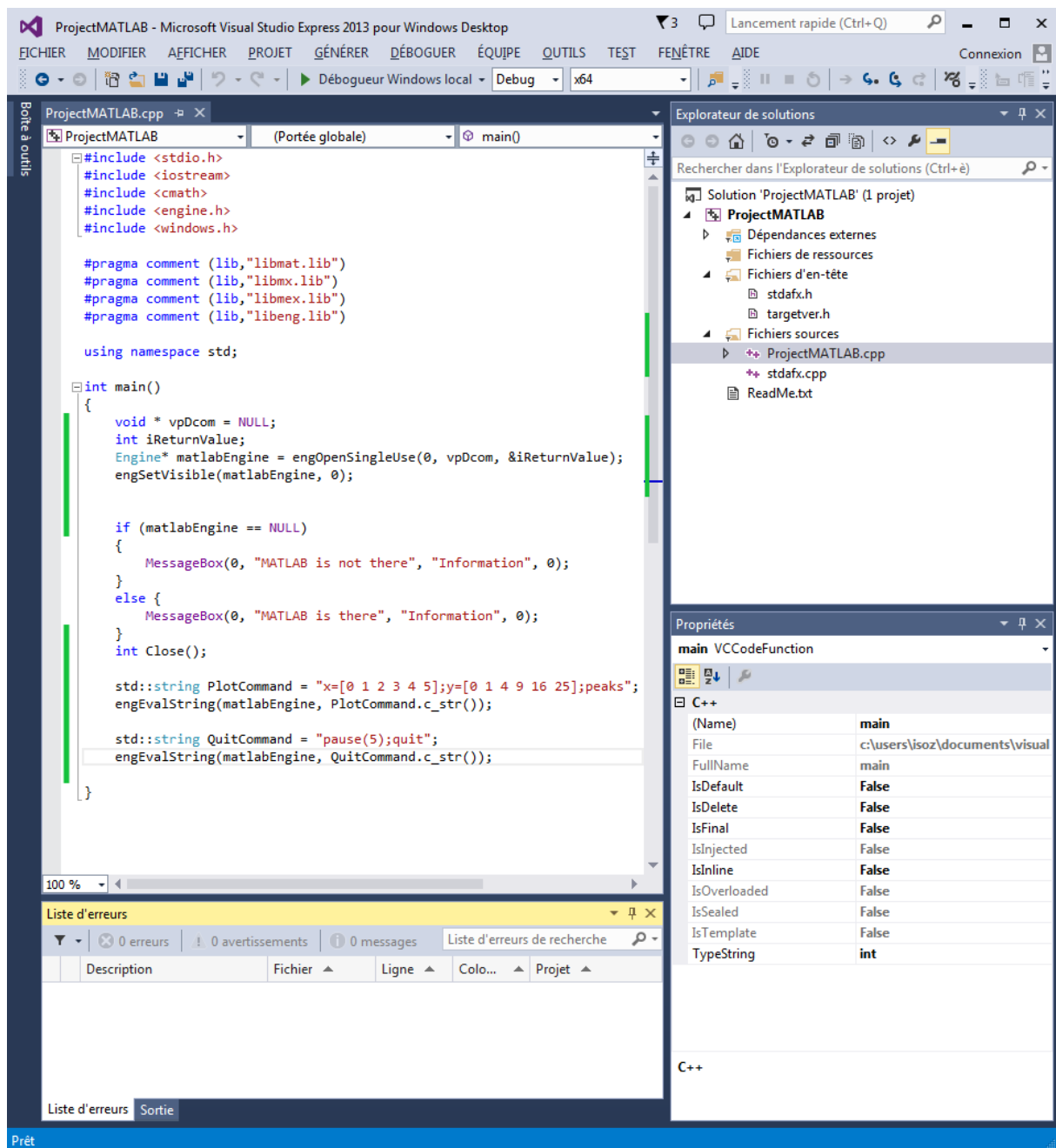


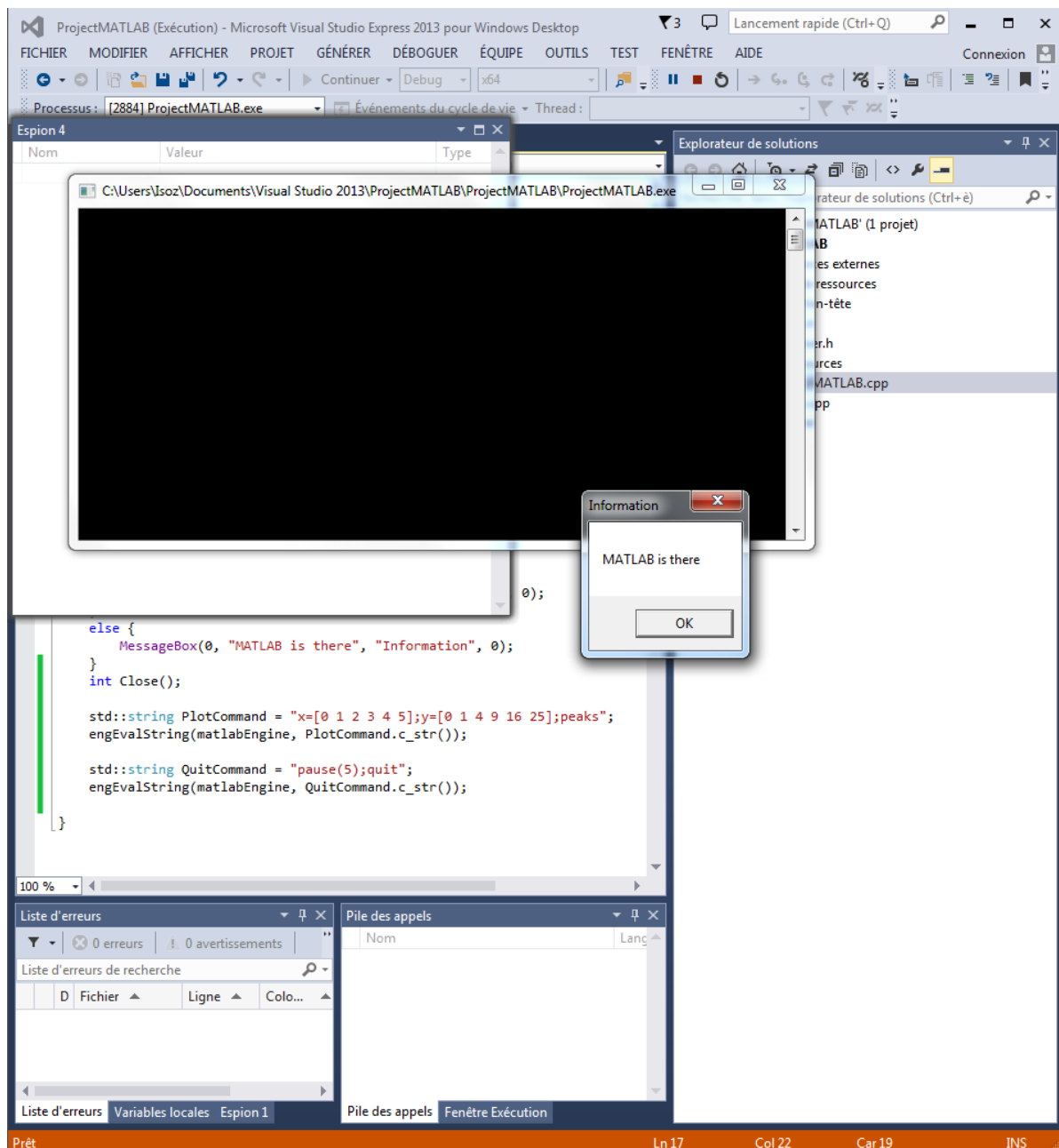
For info that's the content :

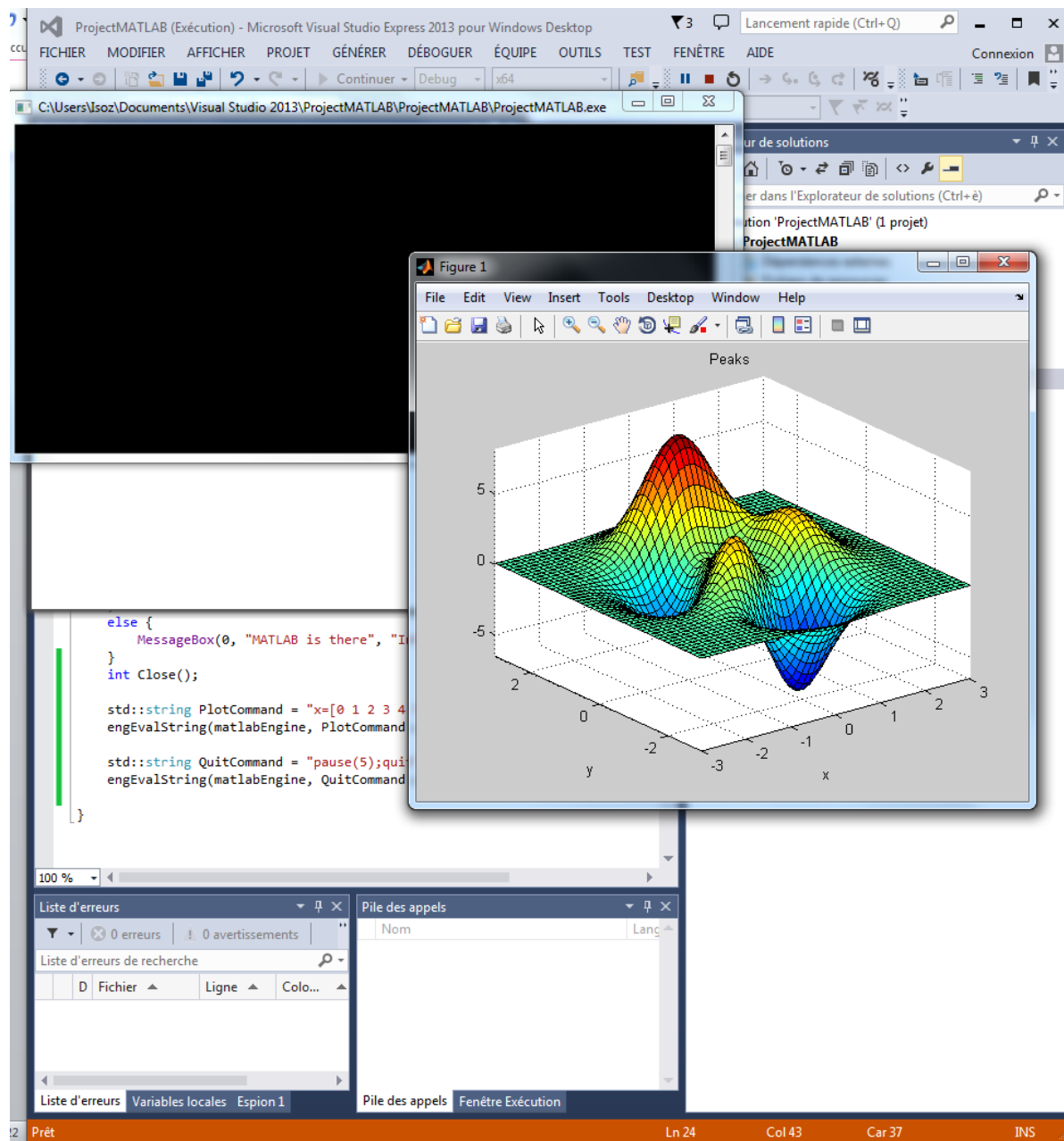


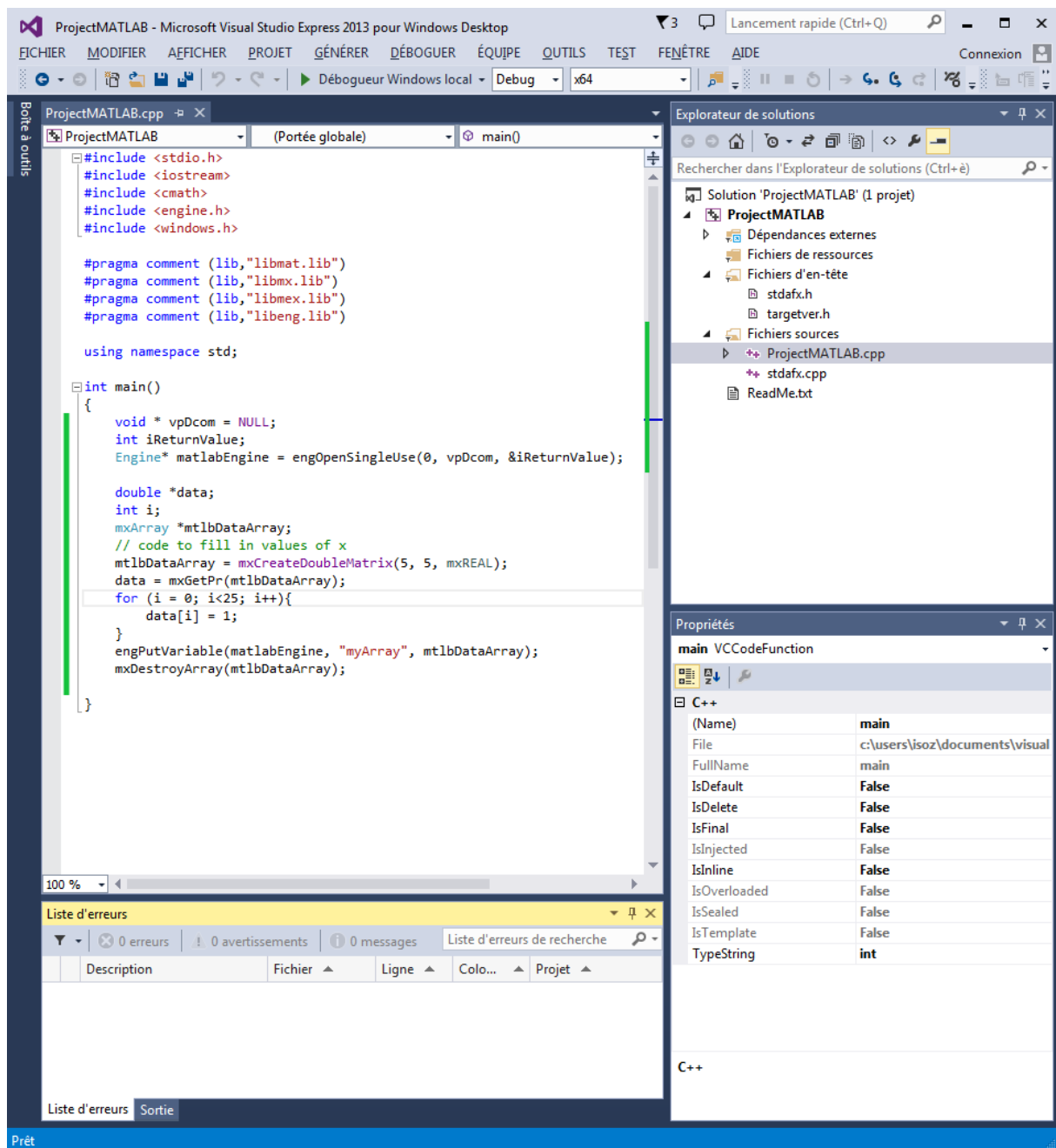


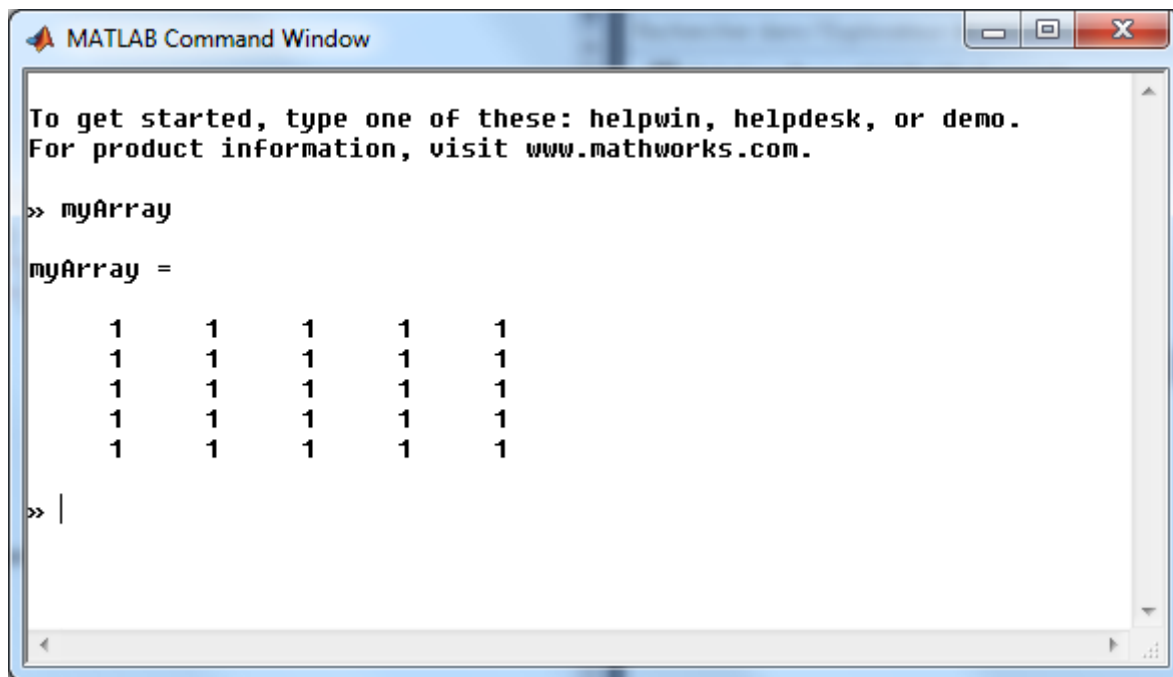






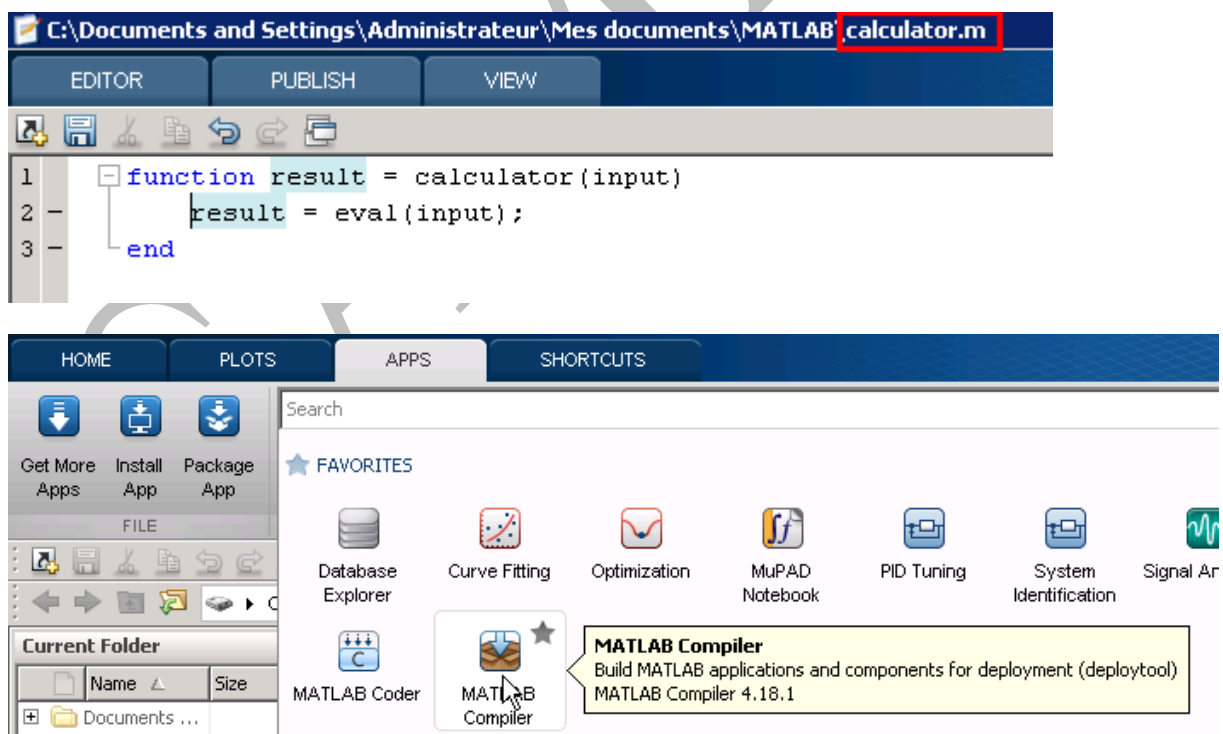


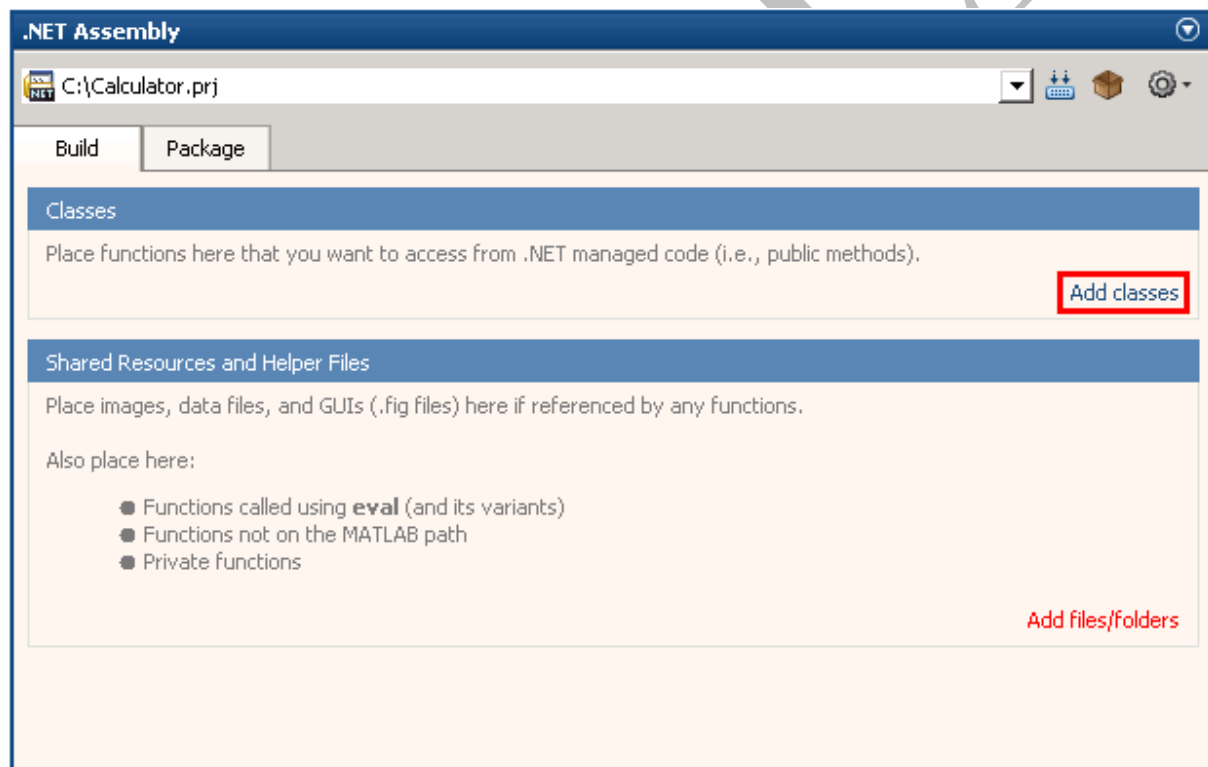
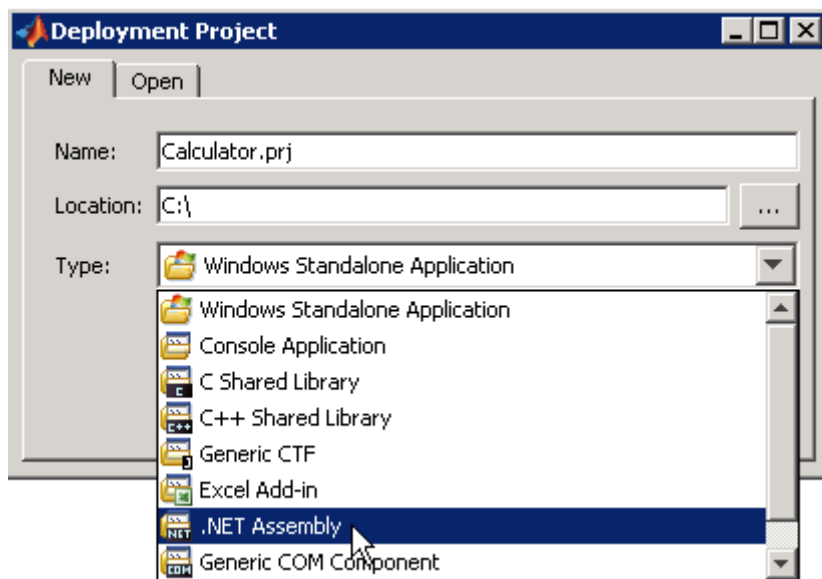


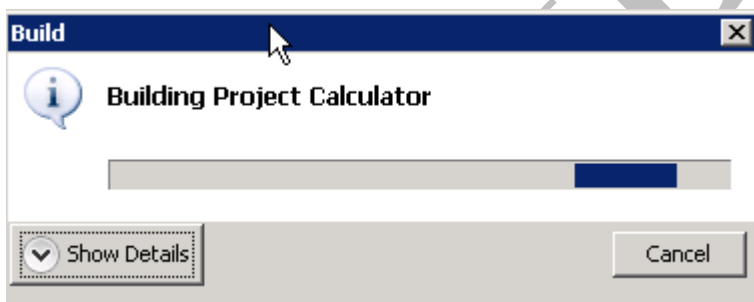
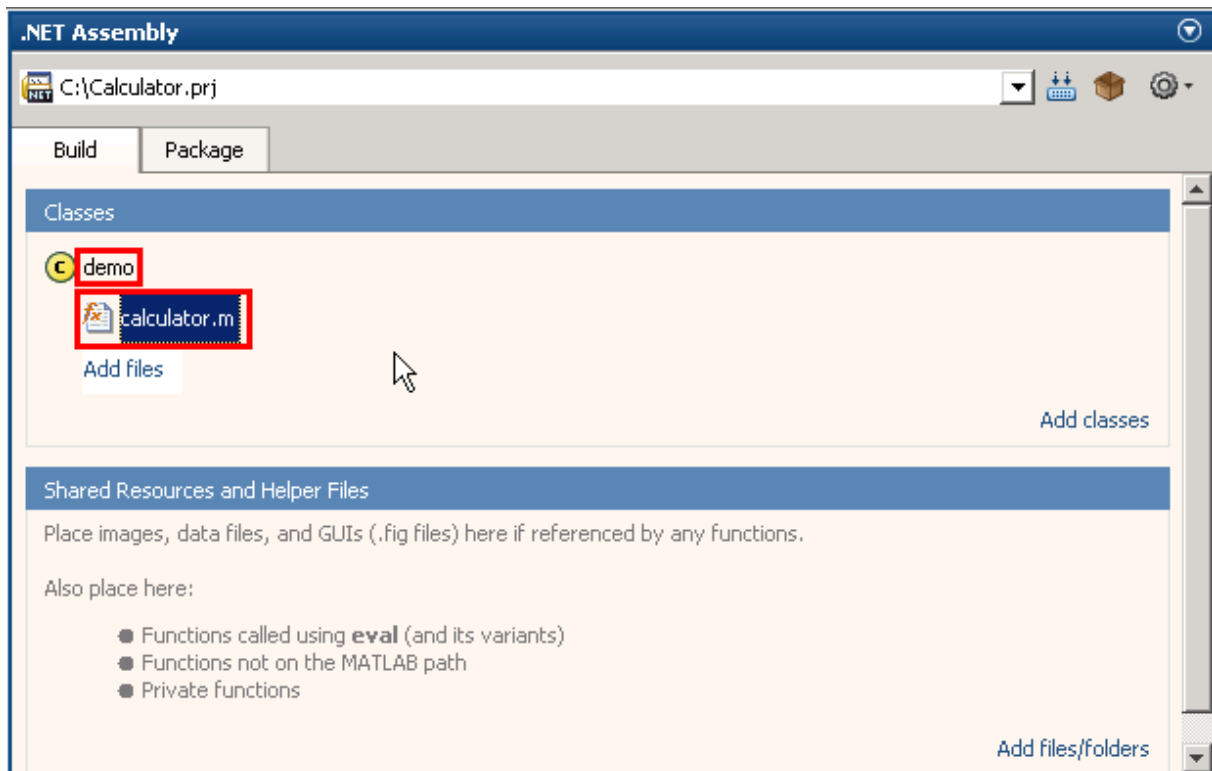


Compile DLL and call from C#

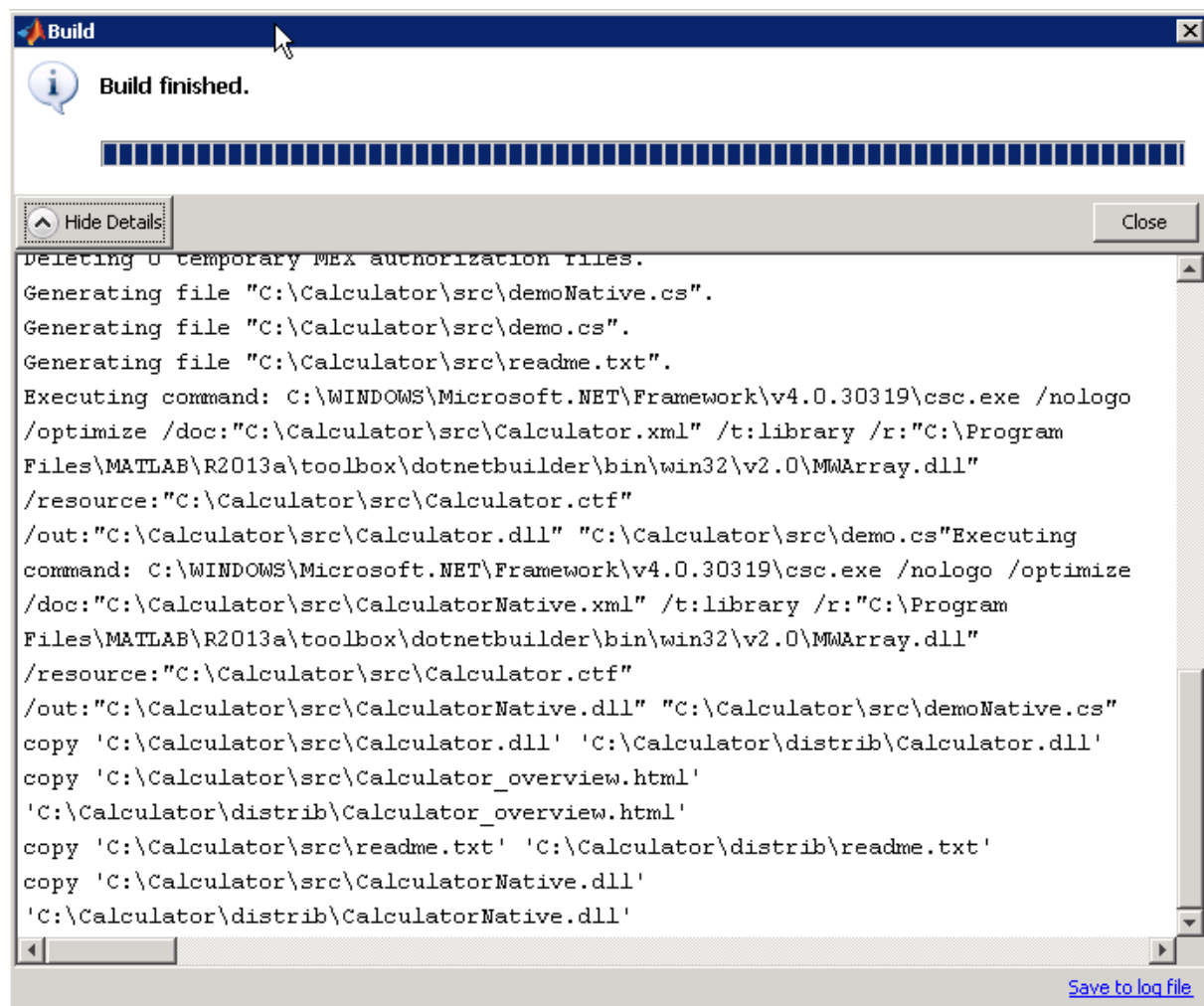
Passing a scalar



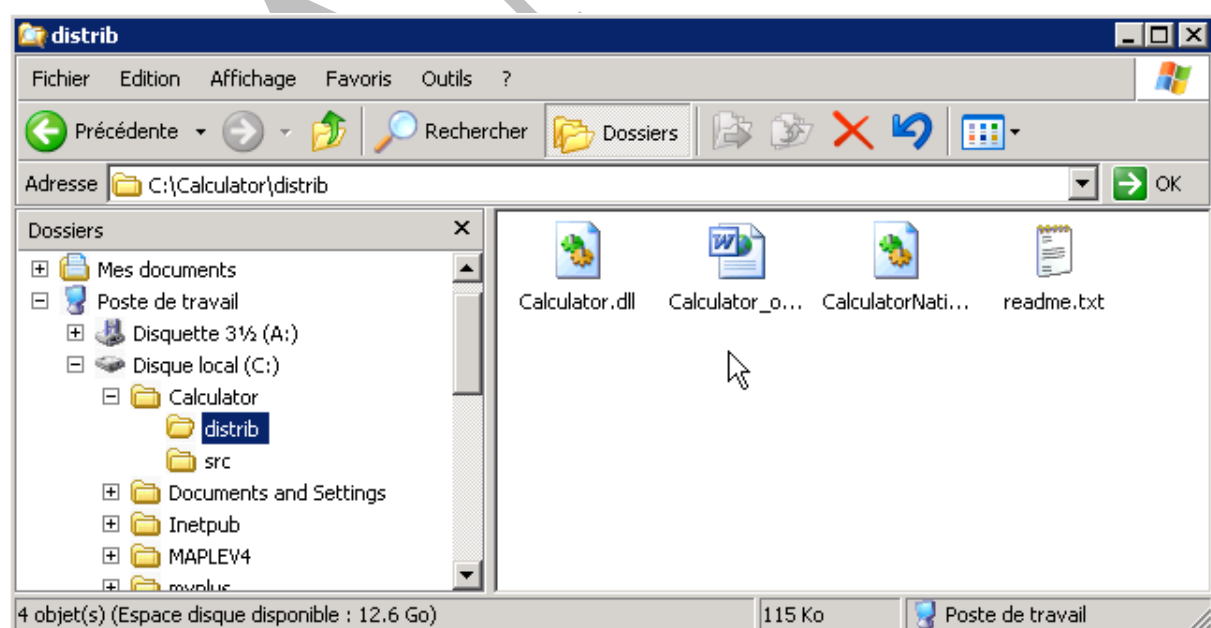


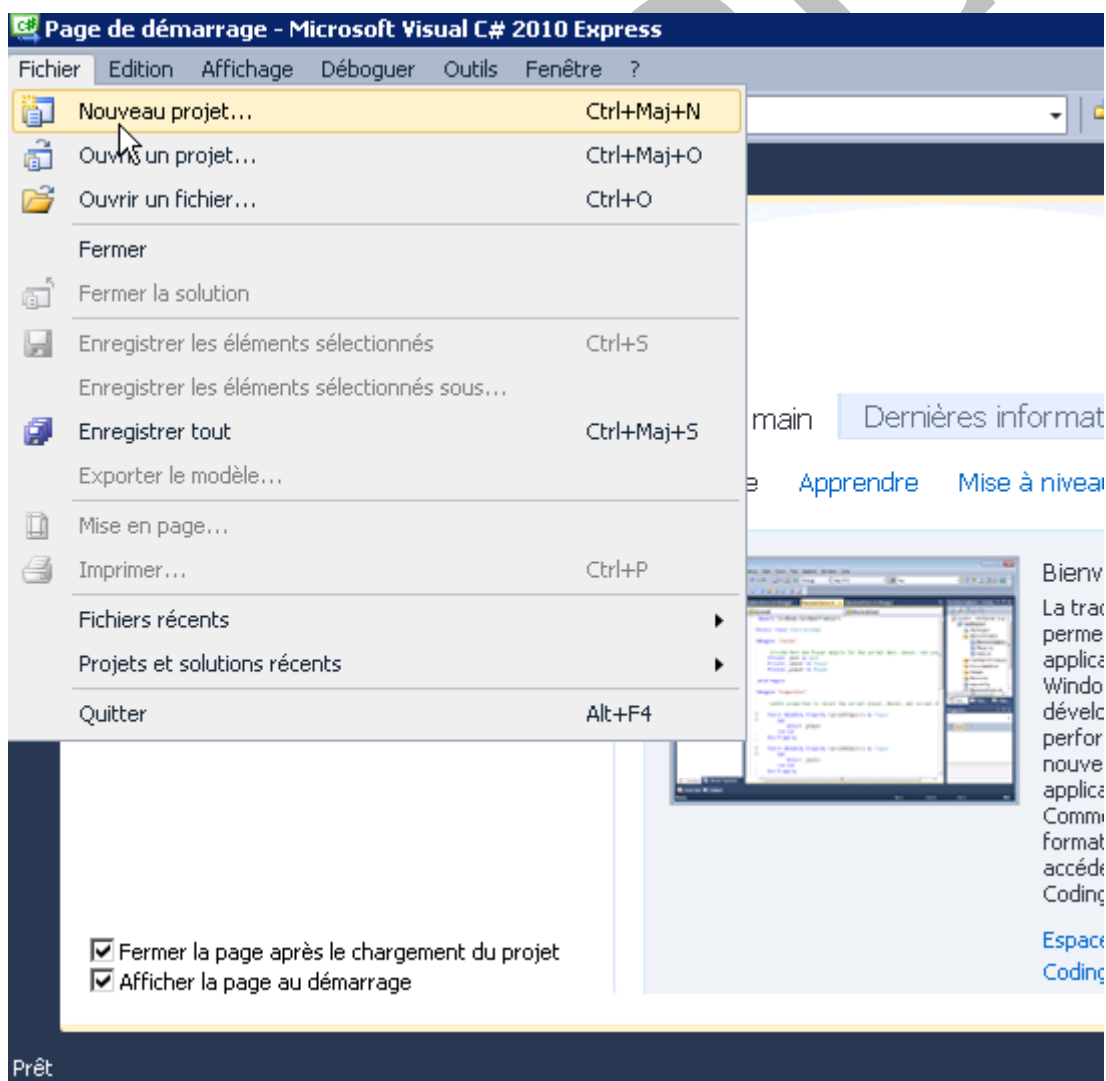
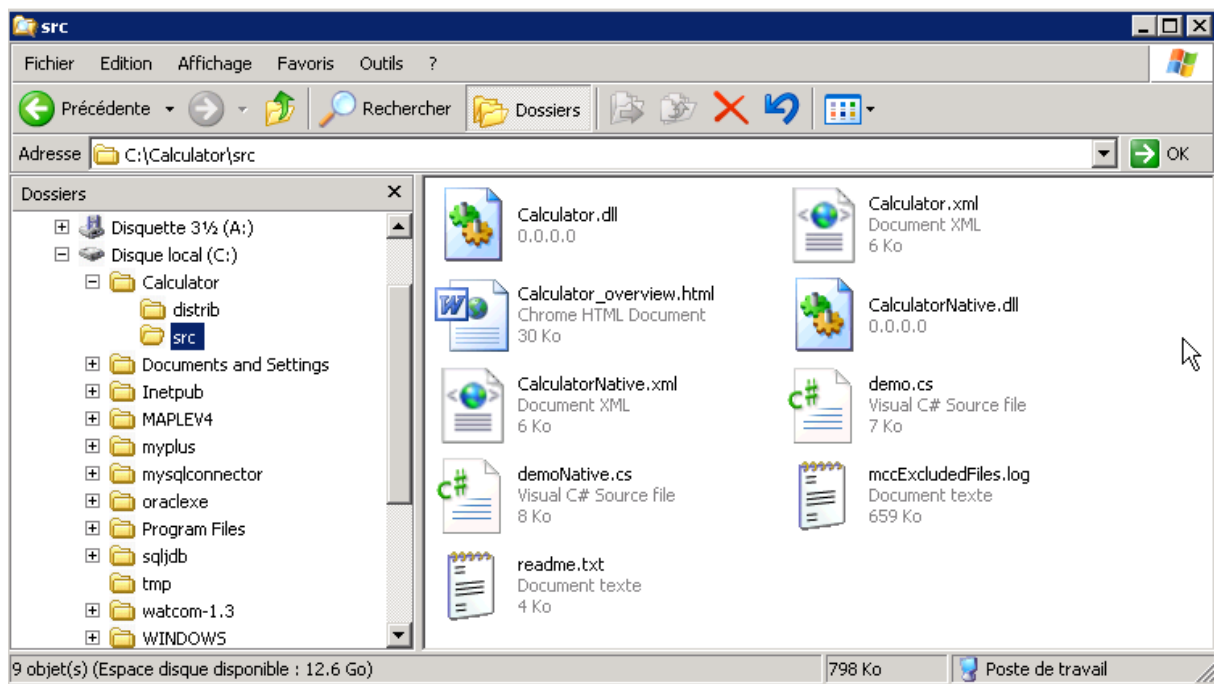


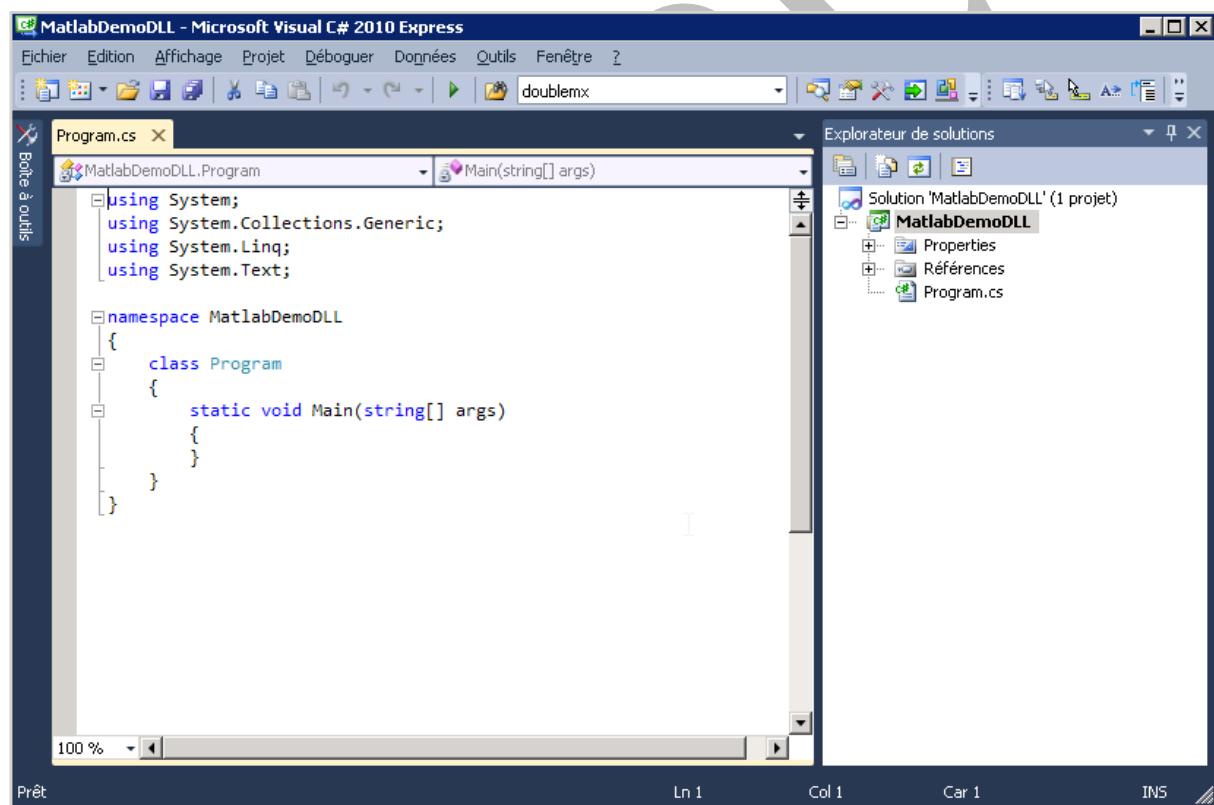
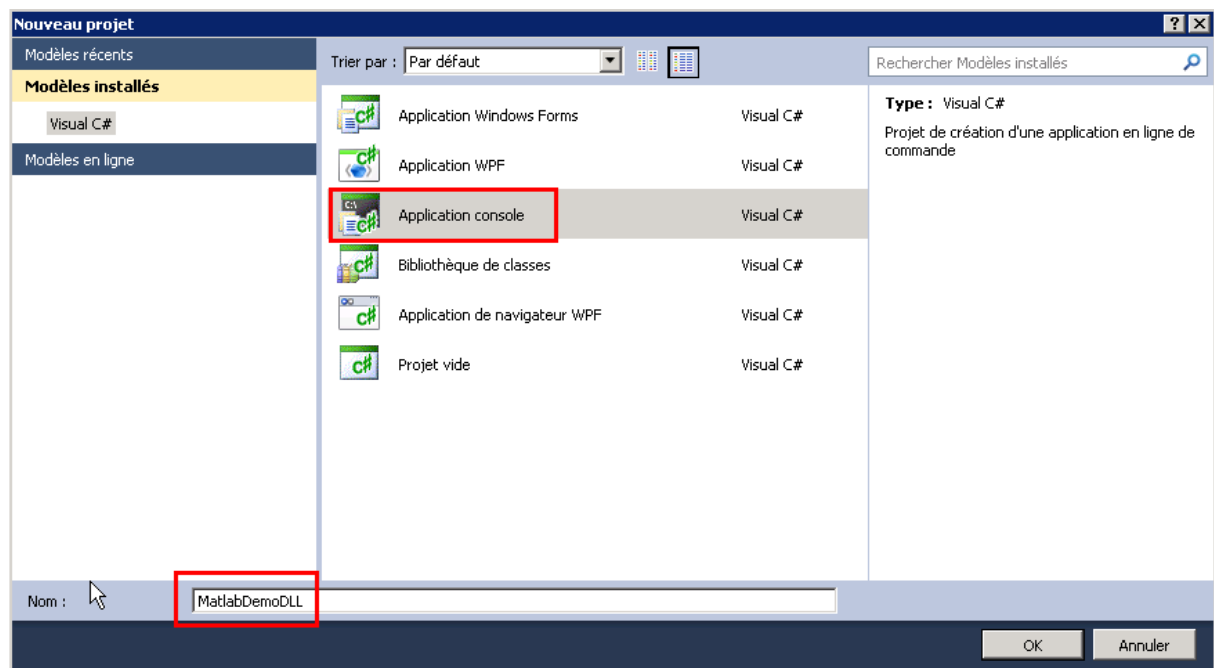
After a long time...

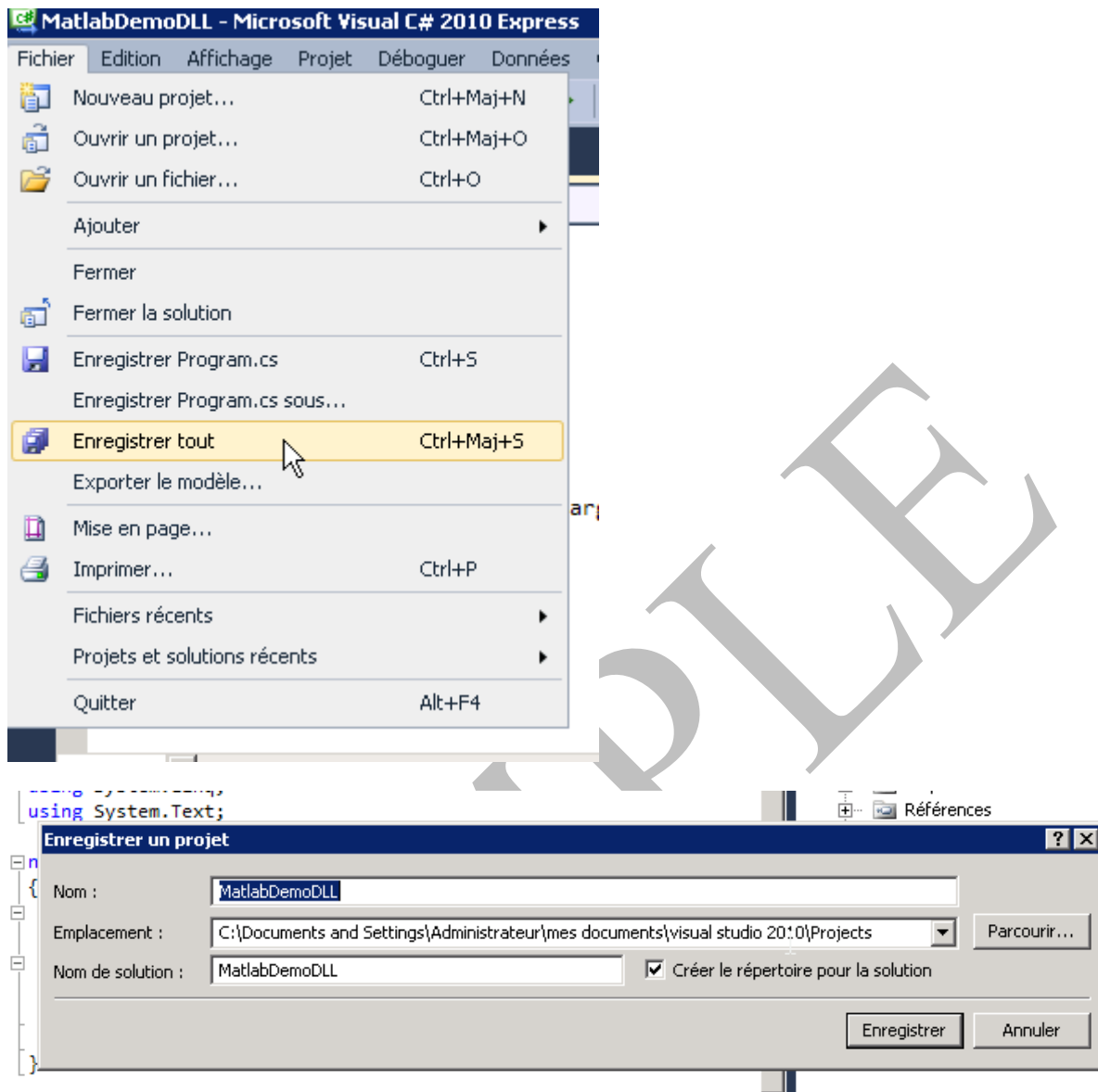


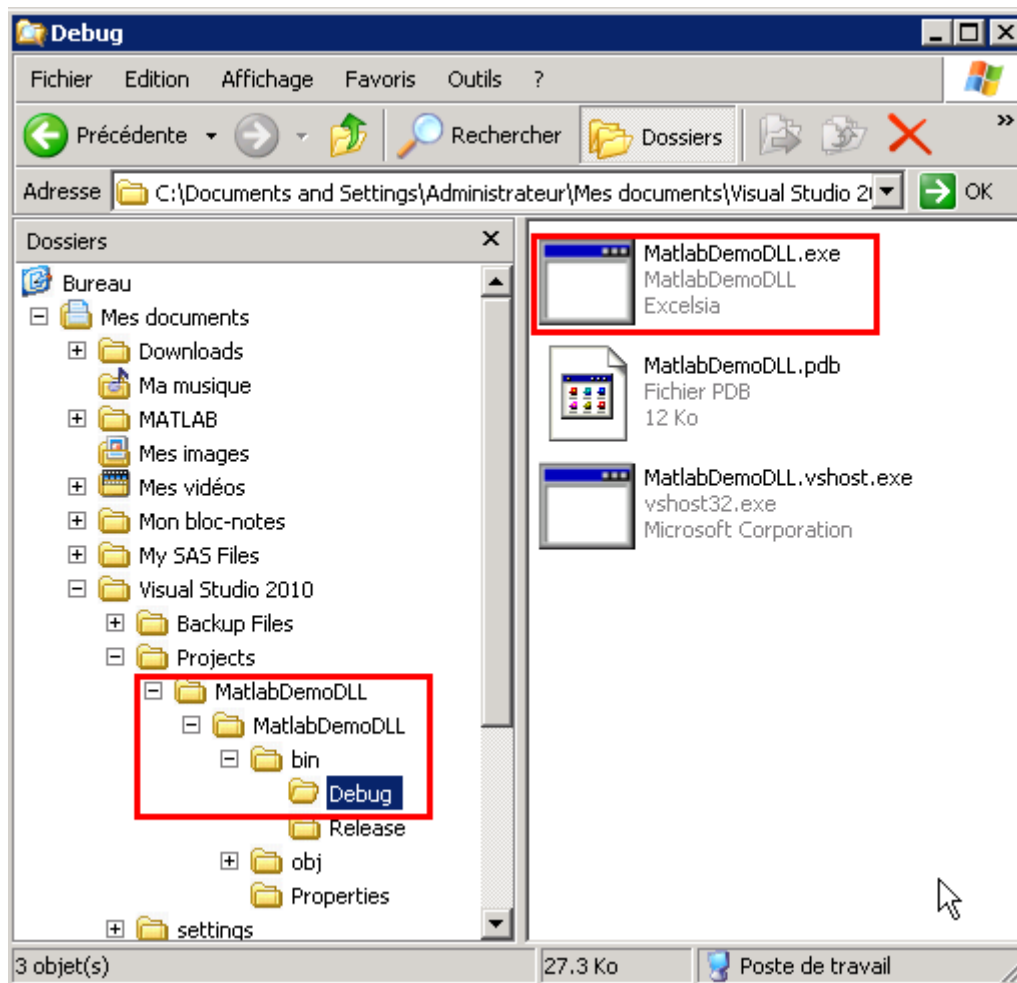
You get:



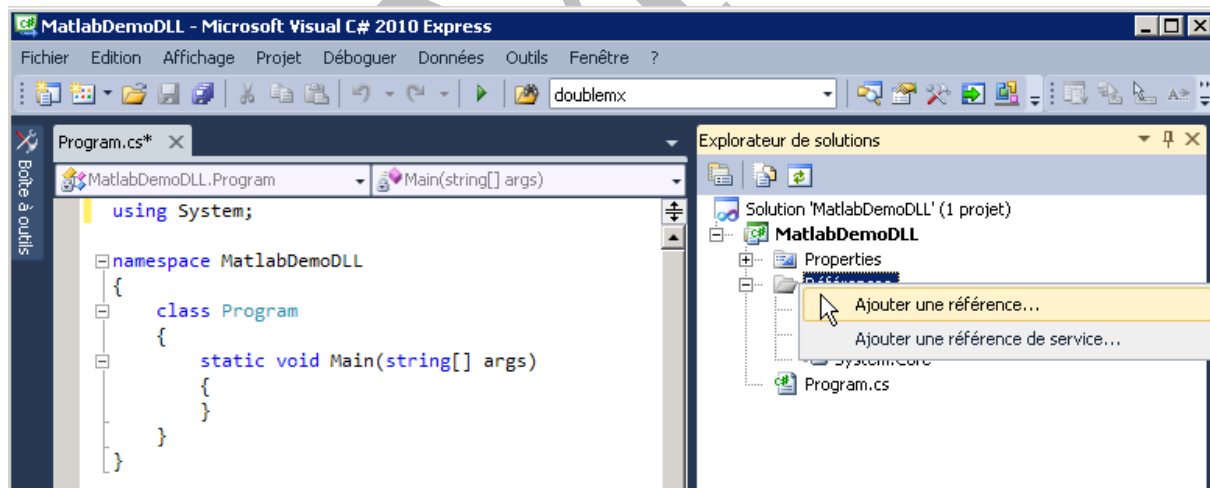
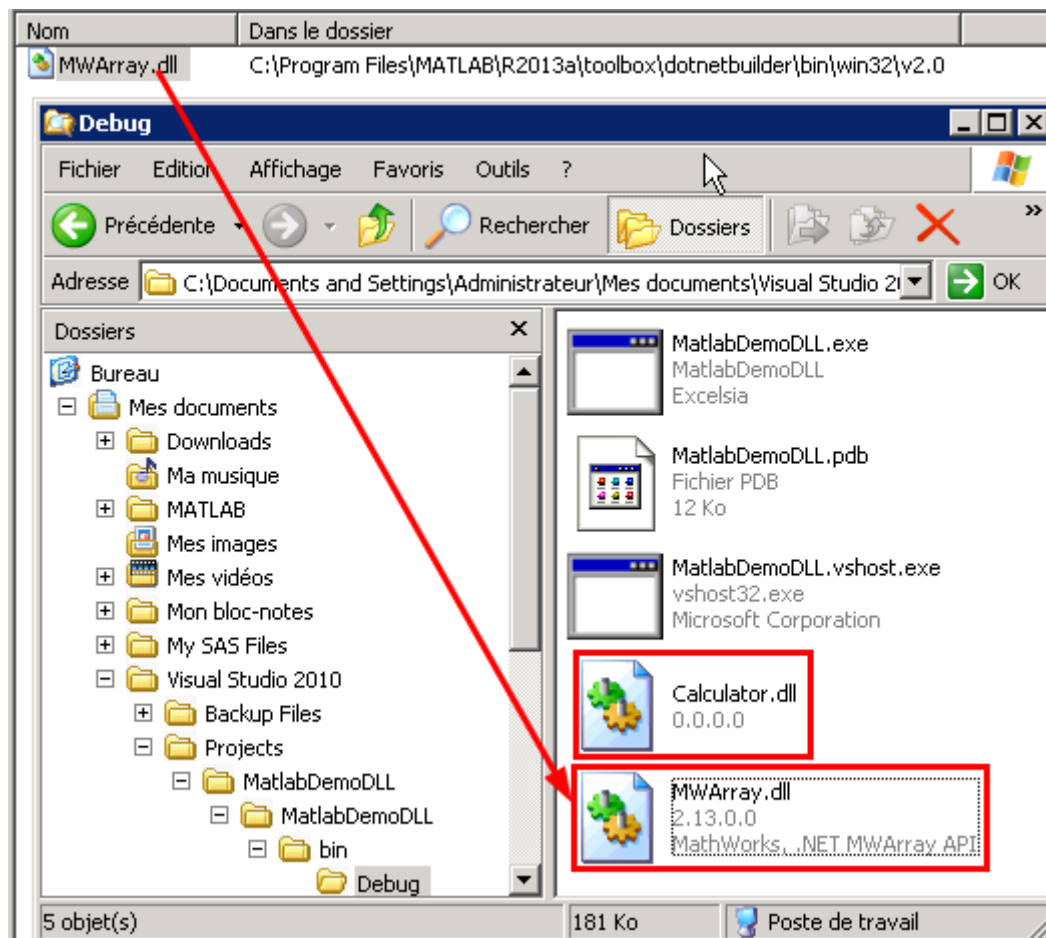


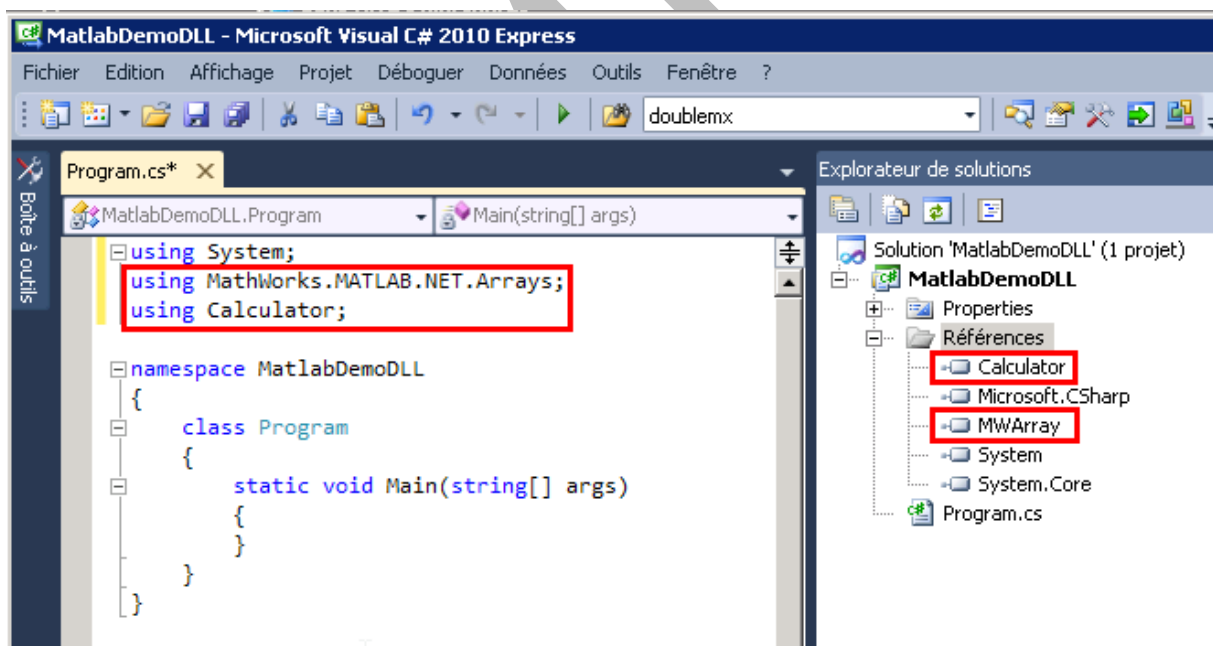
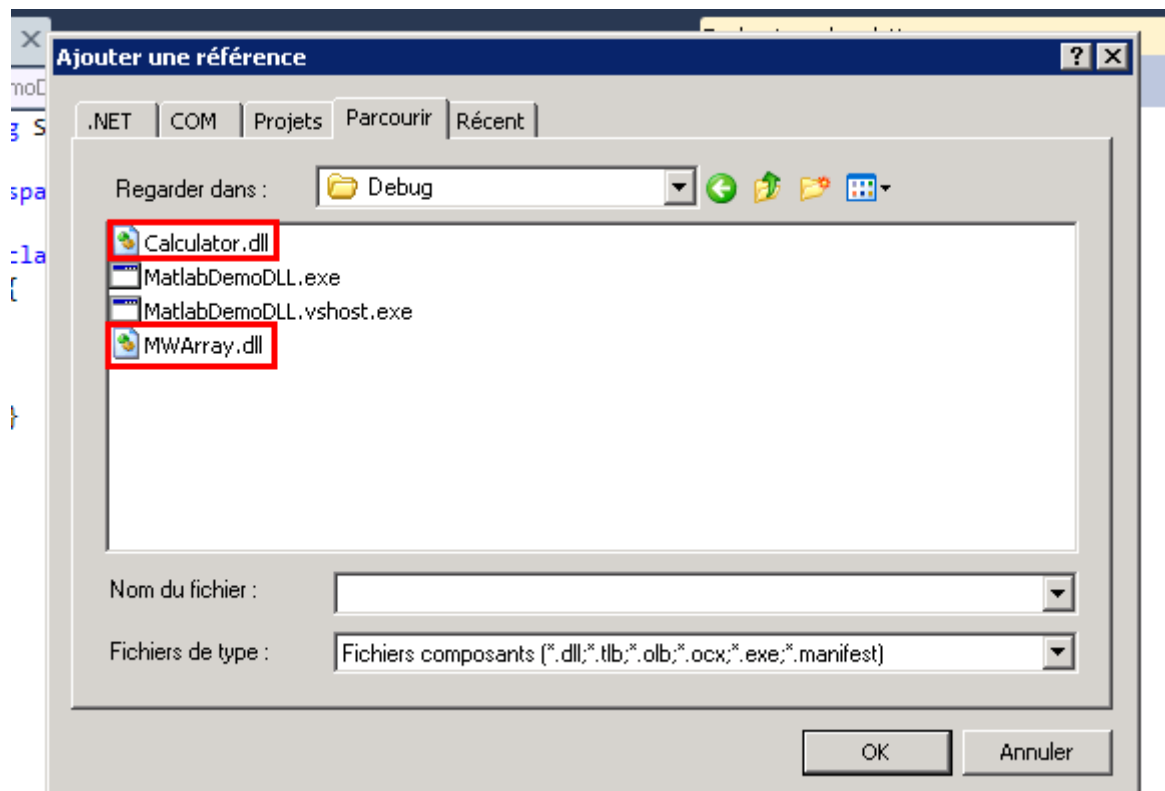


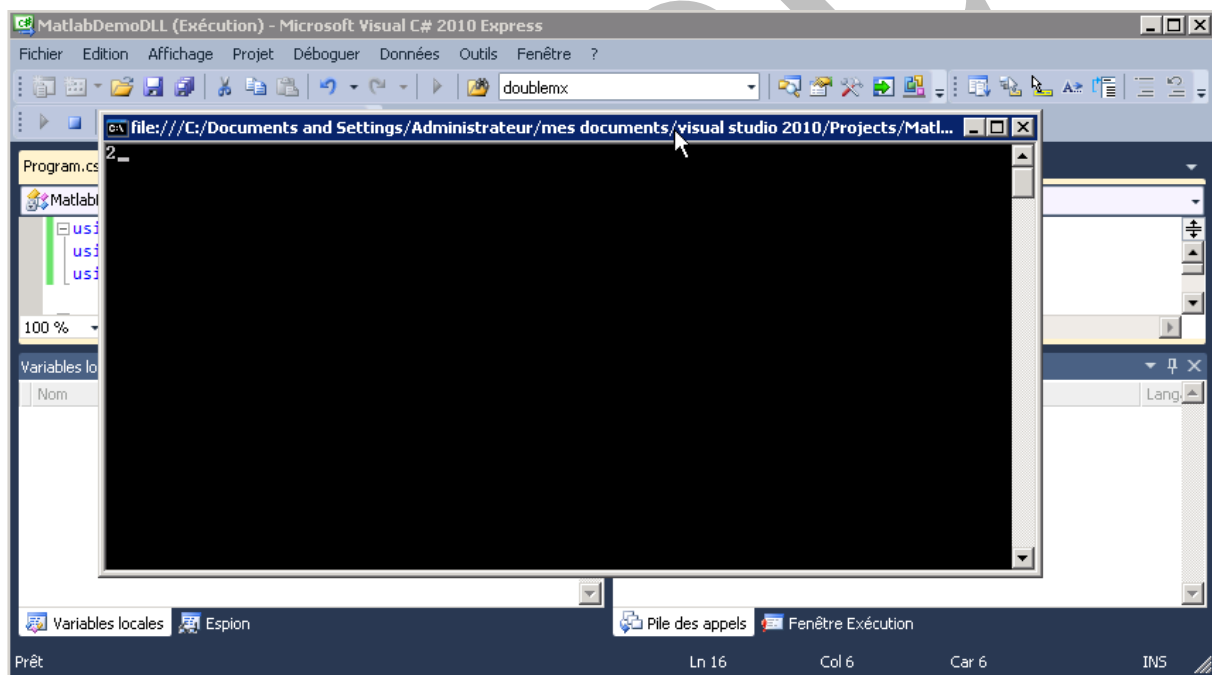
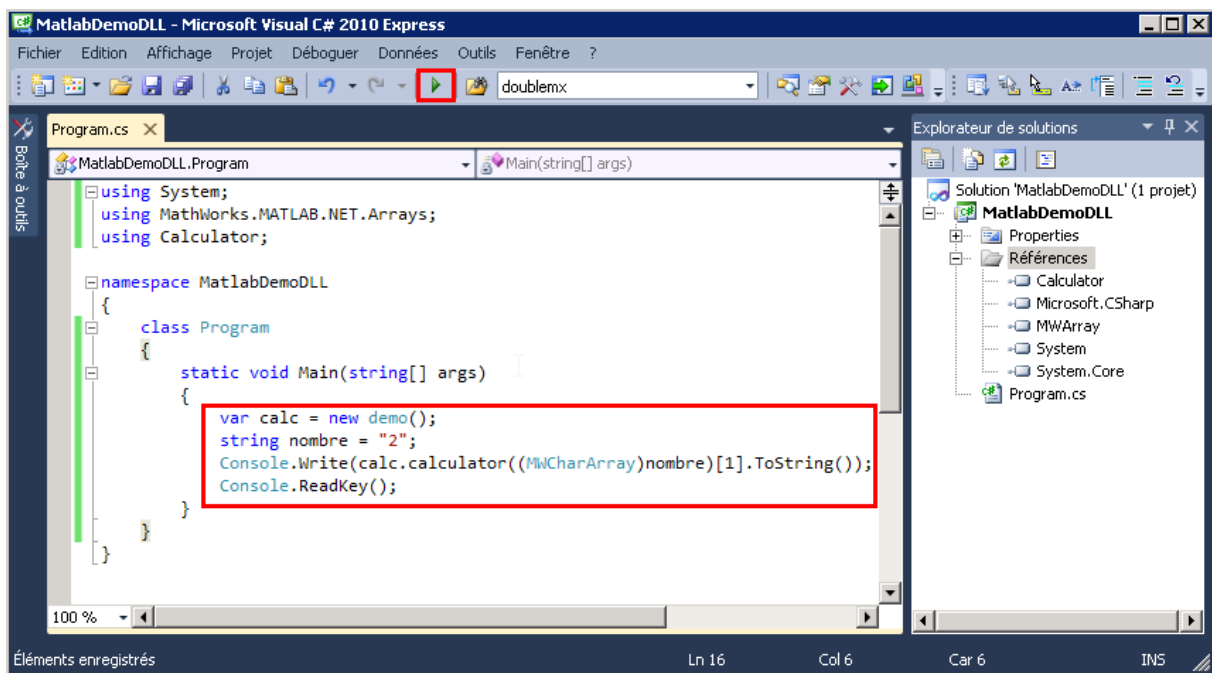




Note that, in order to call the generated DLL, the DLL must reside either in the same directory as your executable or in a directory on your system path through the environment variables of the system!

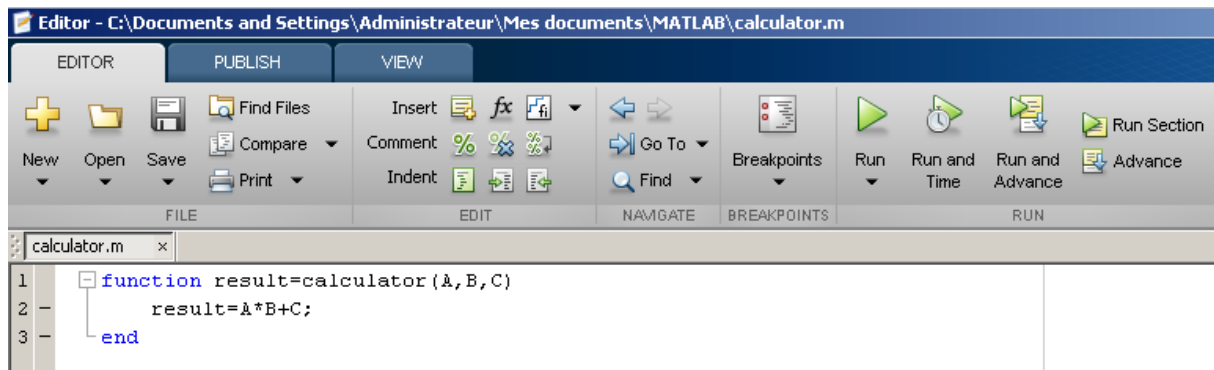




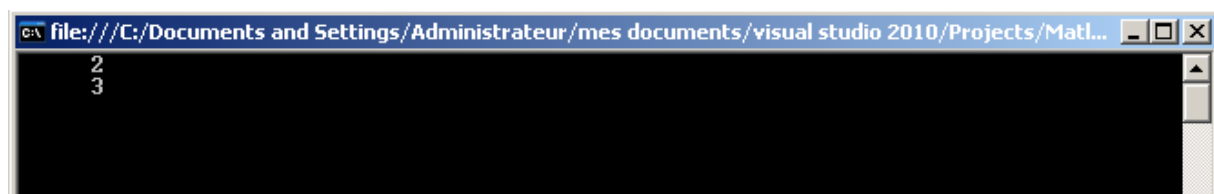
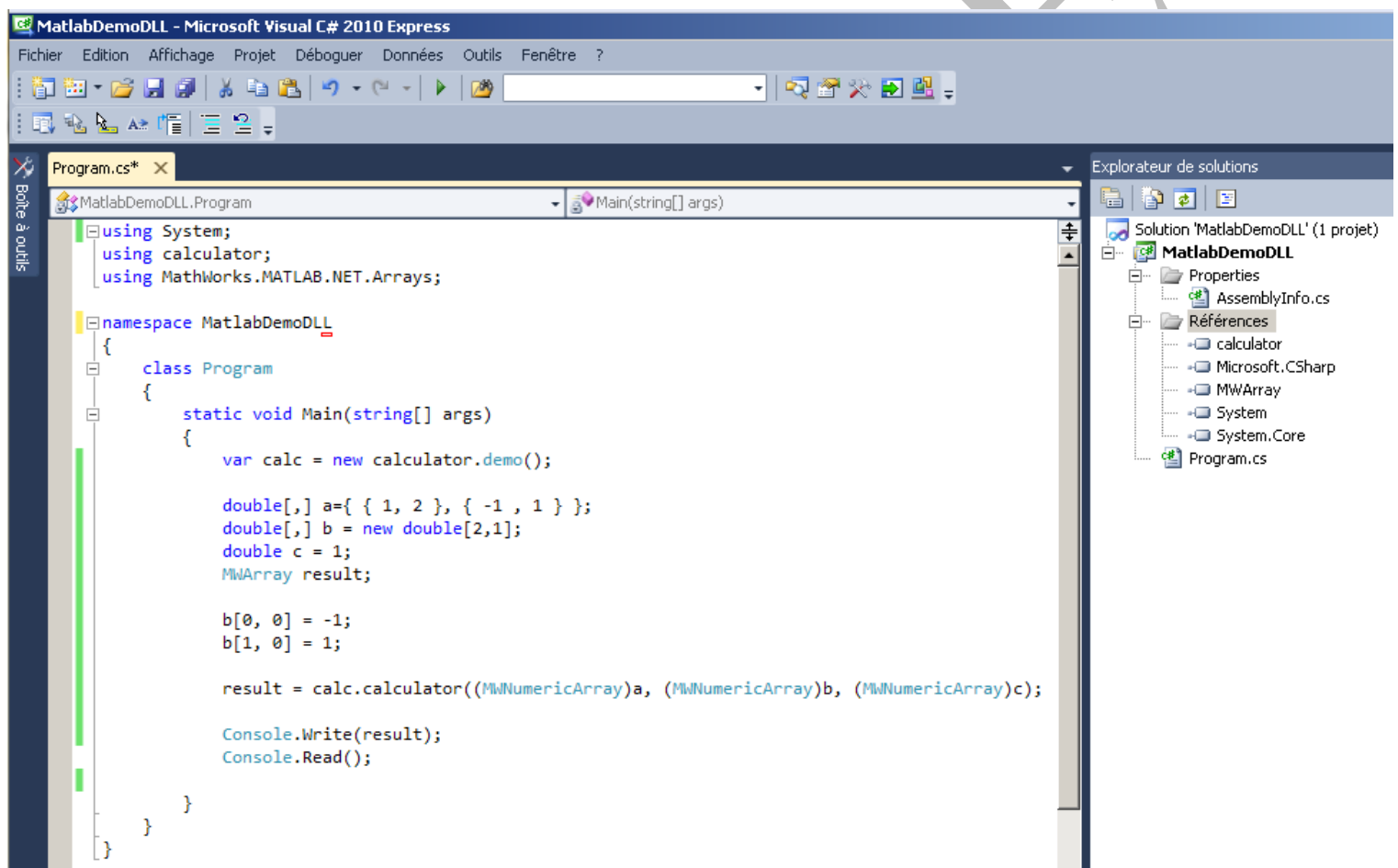


Passing a constant, vector and matrix

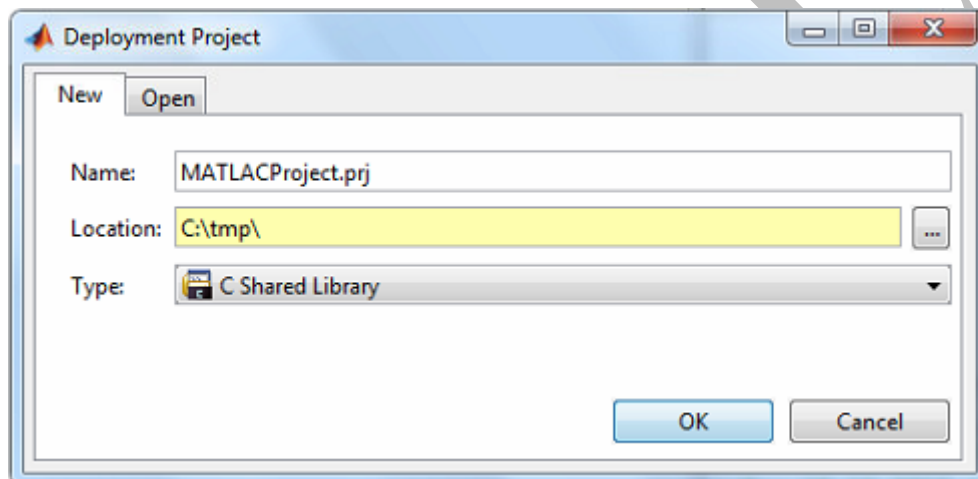
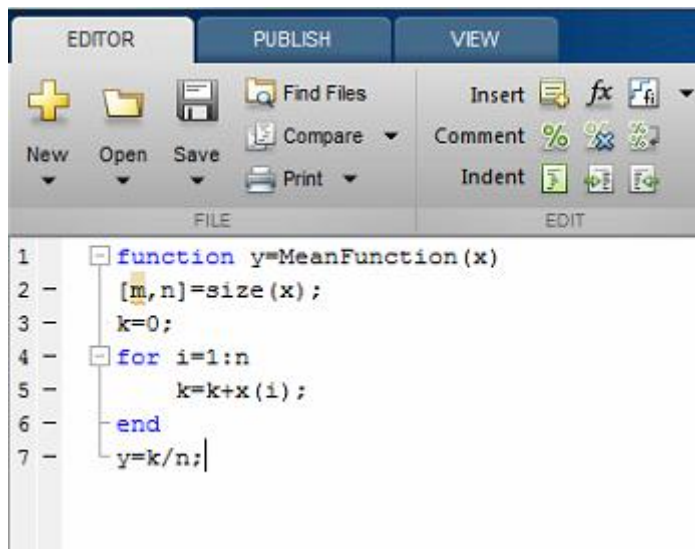
Because C# vectors are column vectors and MATLAB vectors are line vectors, there is a trap for passing vector variables.

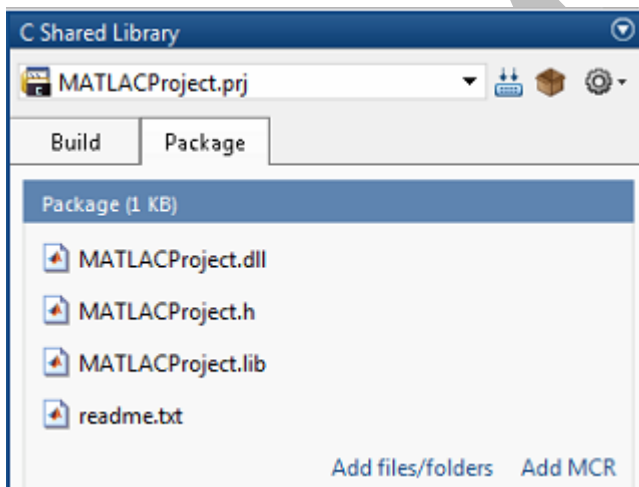
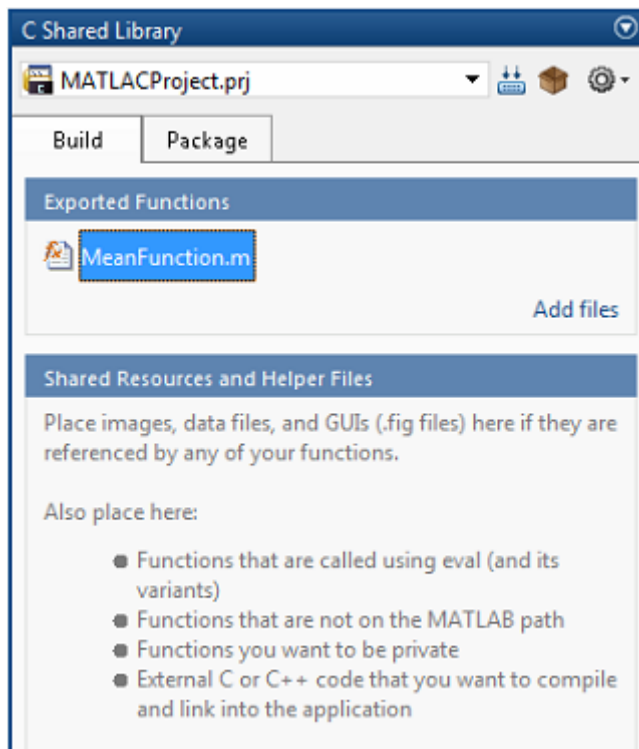


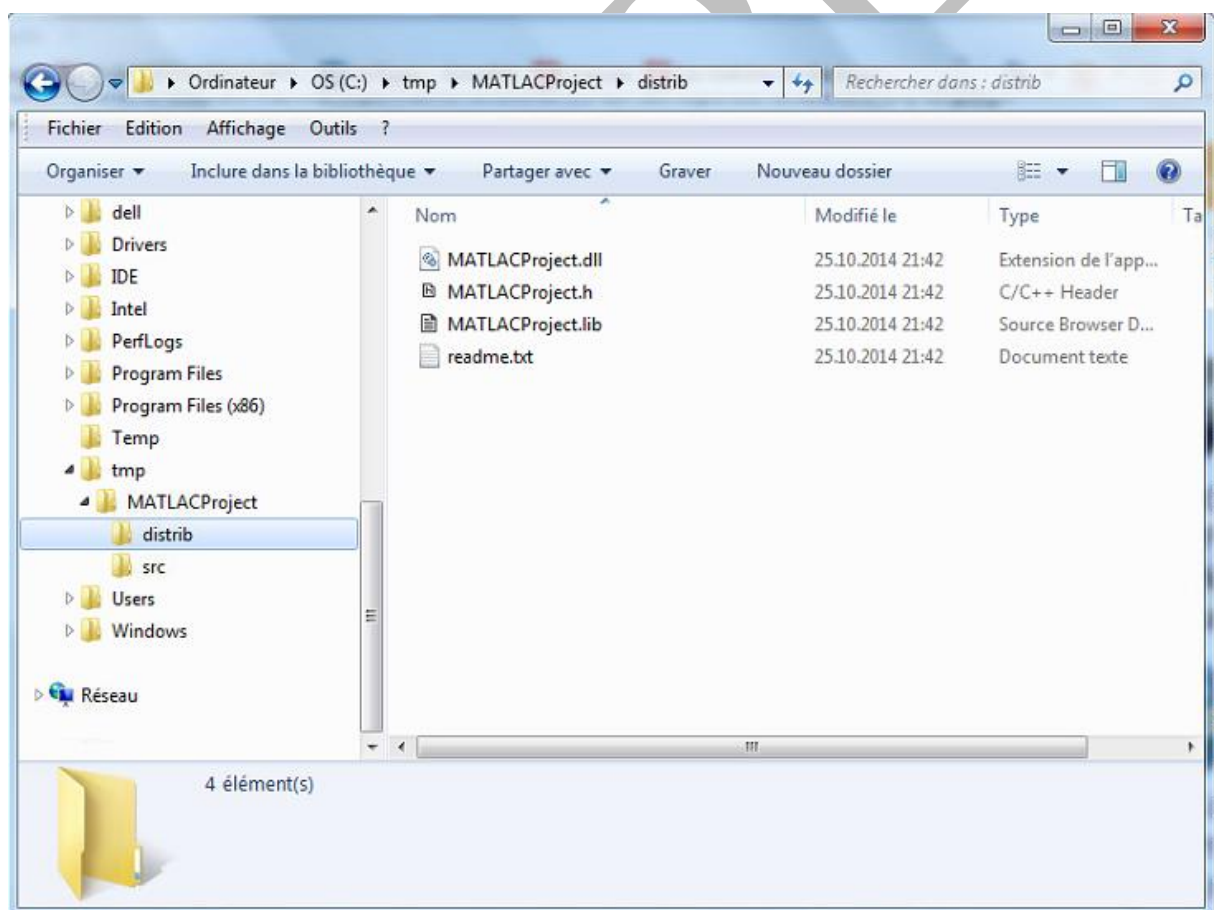
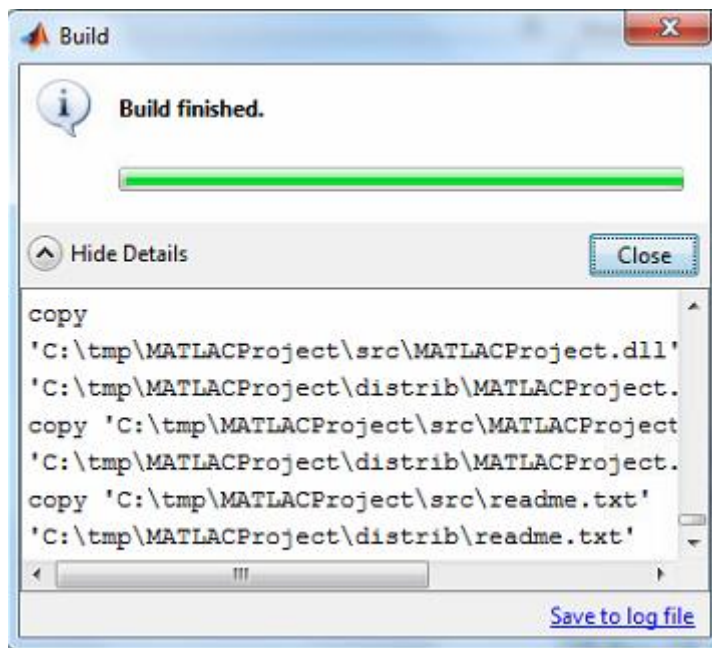
$$A = \begin{pmatrix} 1 & 2 \\ -1 & 1 \end{pmatrix}, B = \begin{pmatrix} -1 \\ 1 \end{pmatrix}, C = 1$$

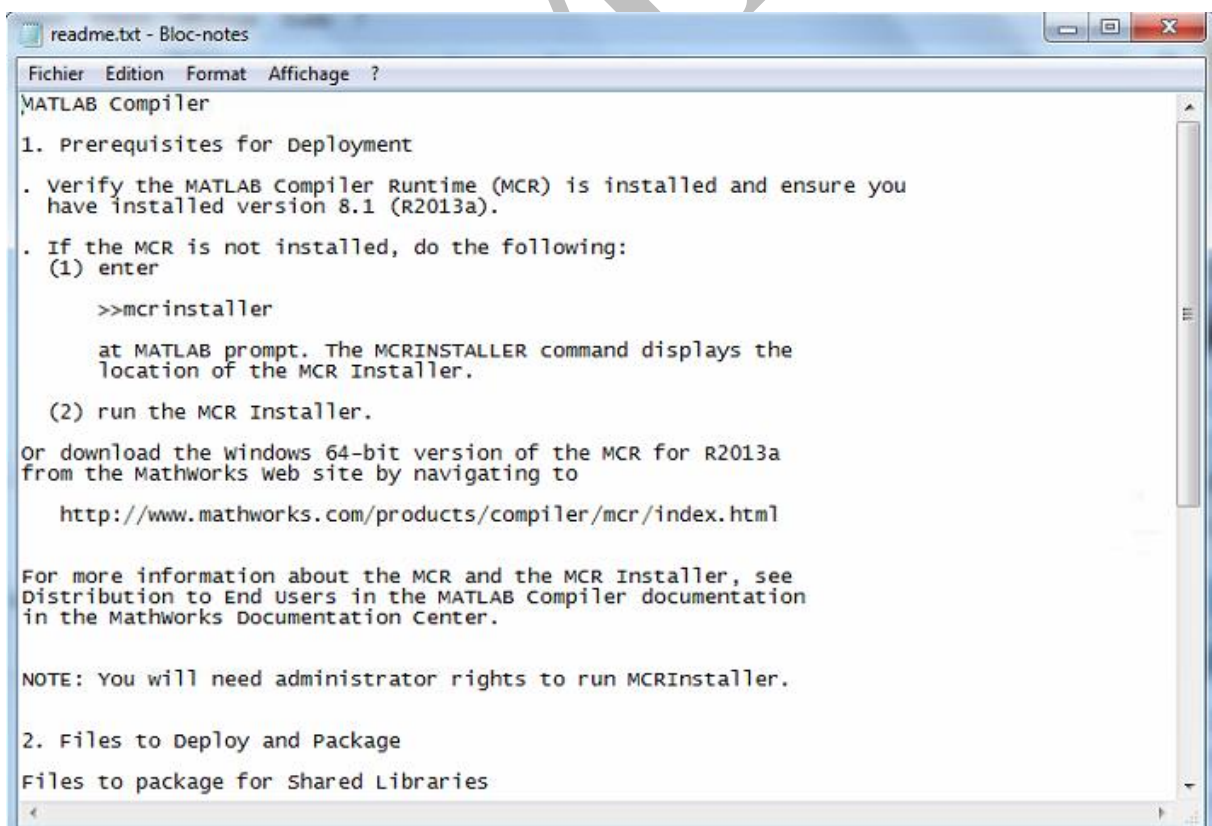
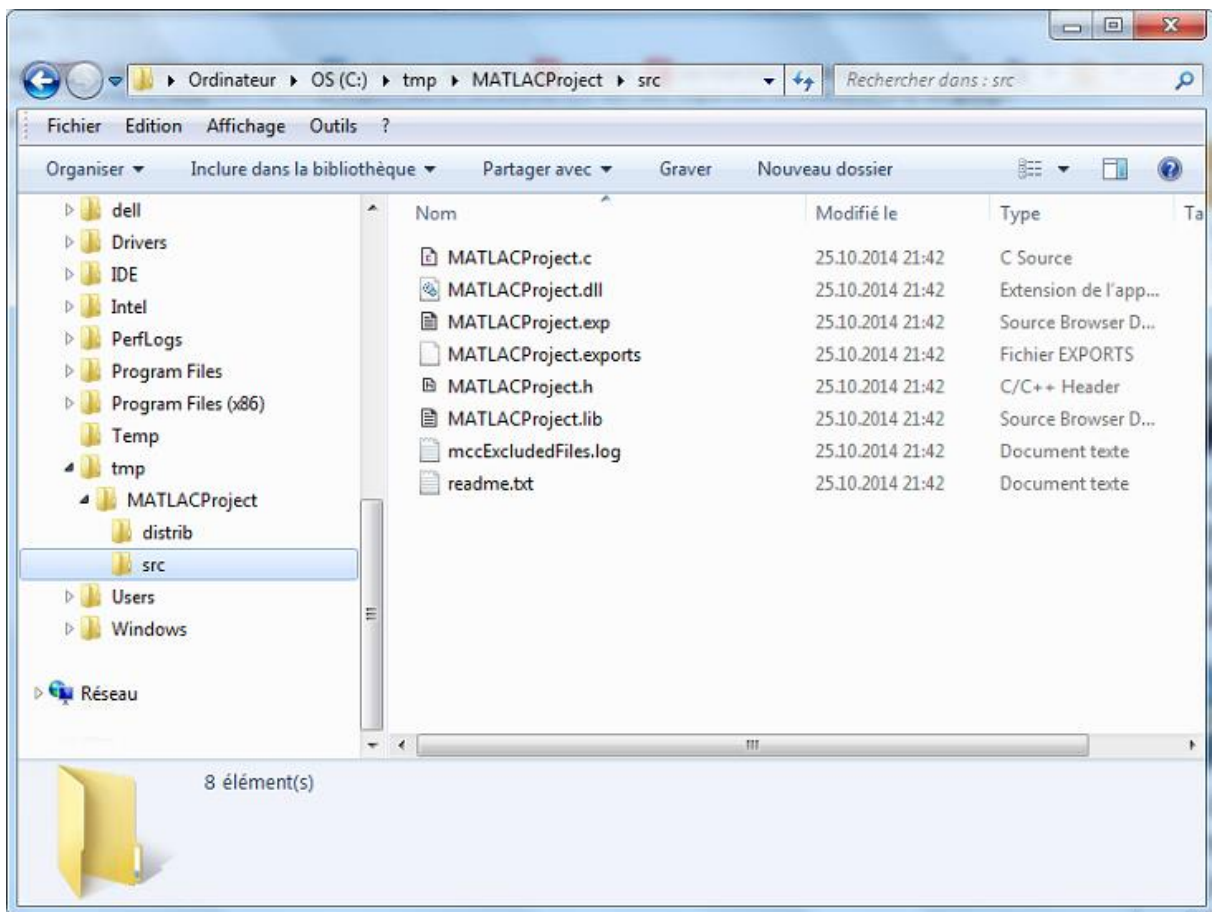


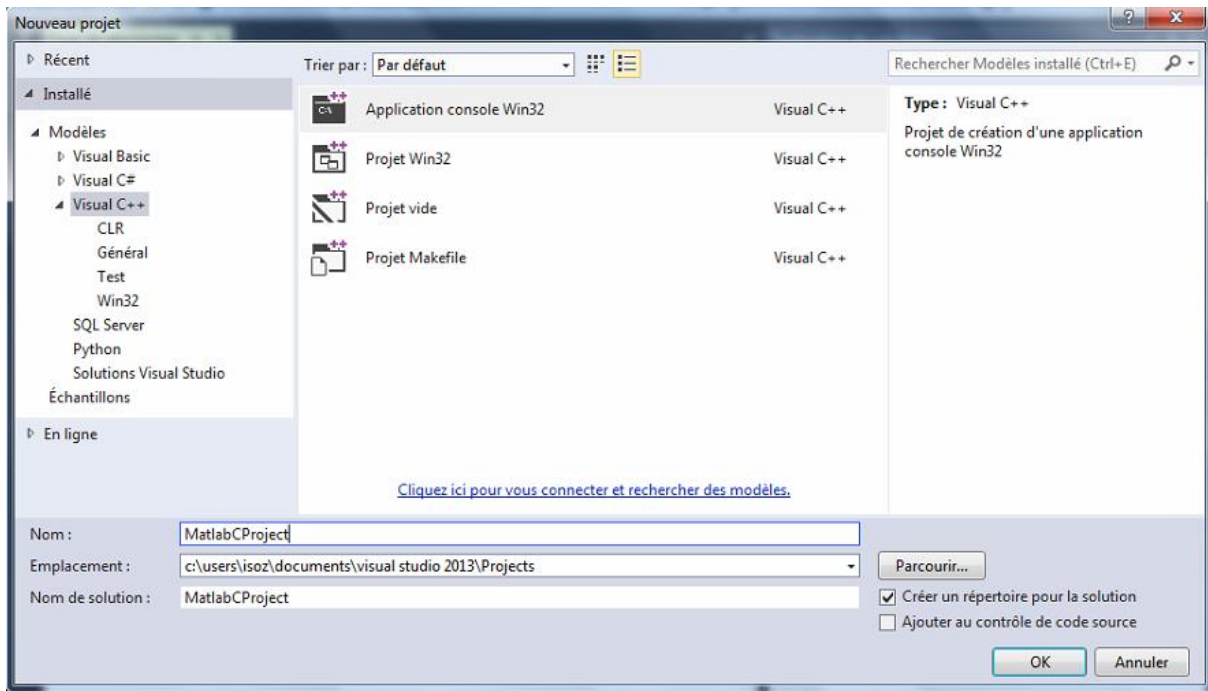
Compile DLL and call from C



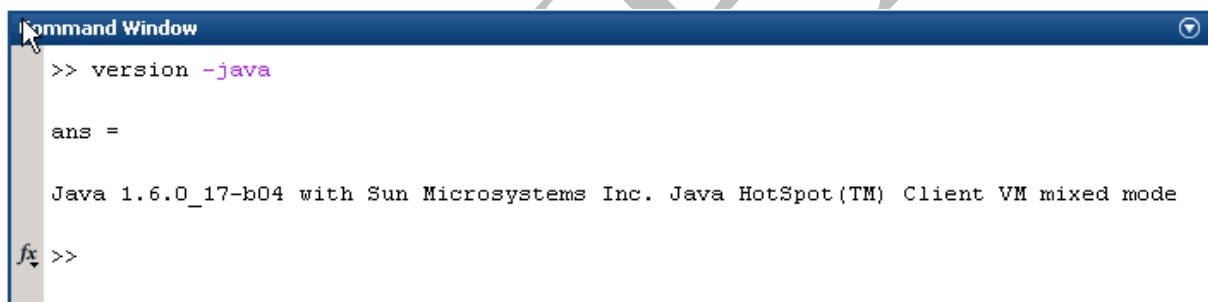




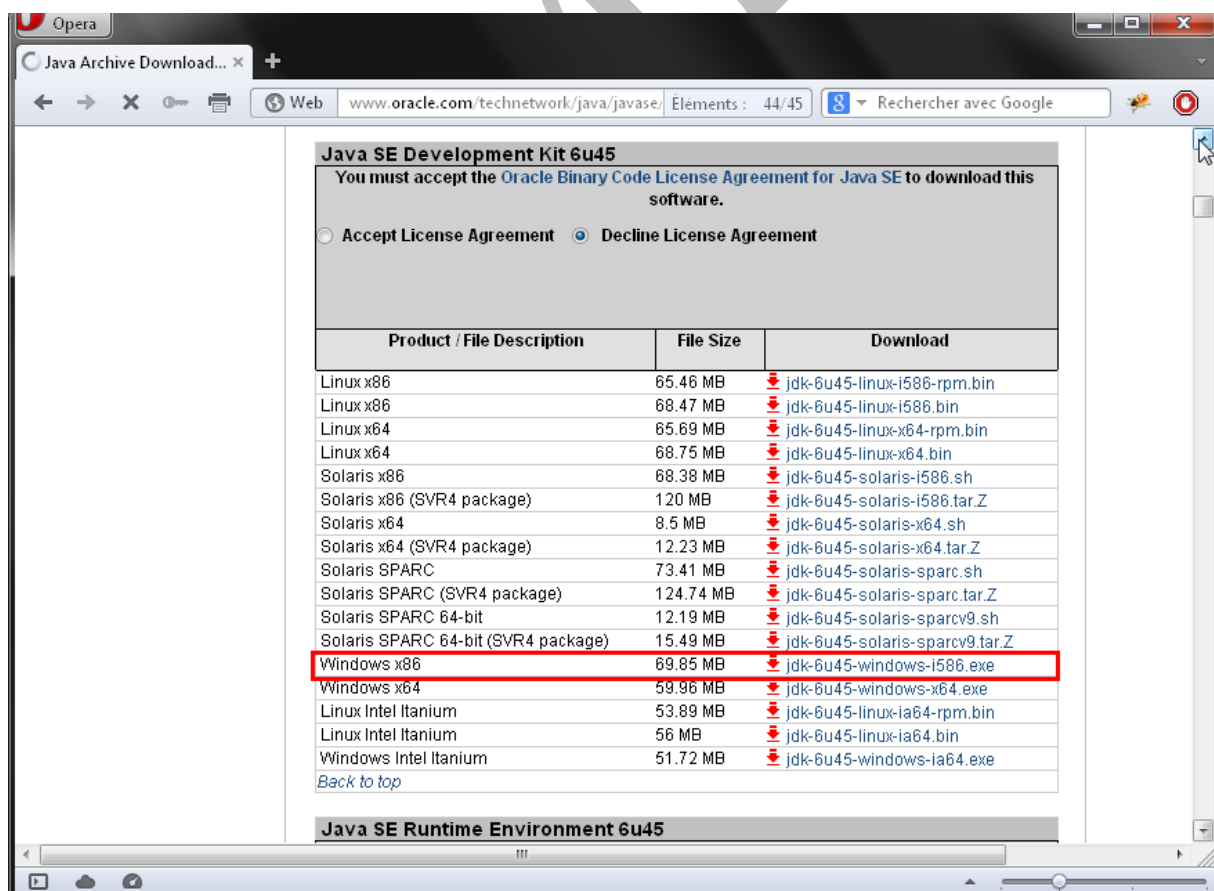
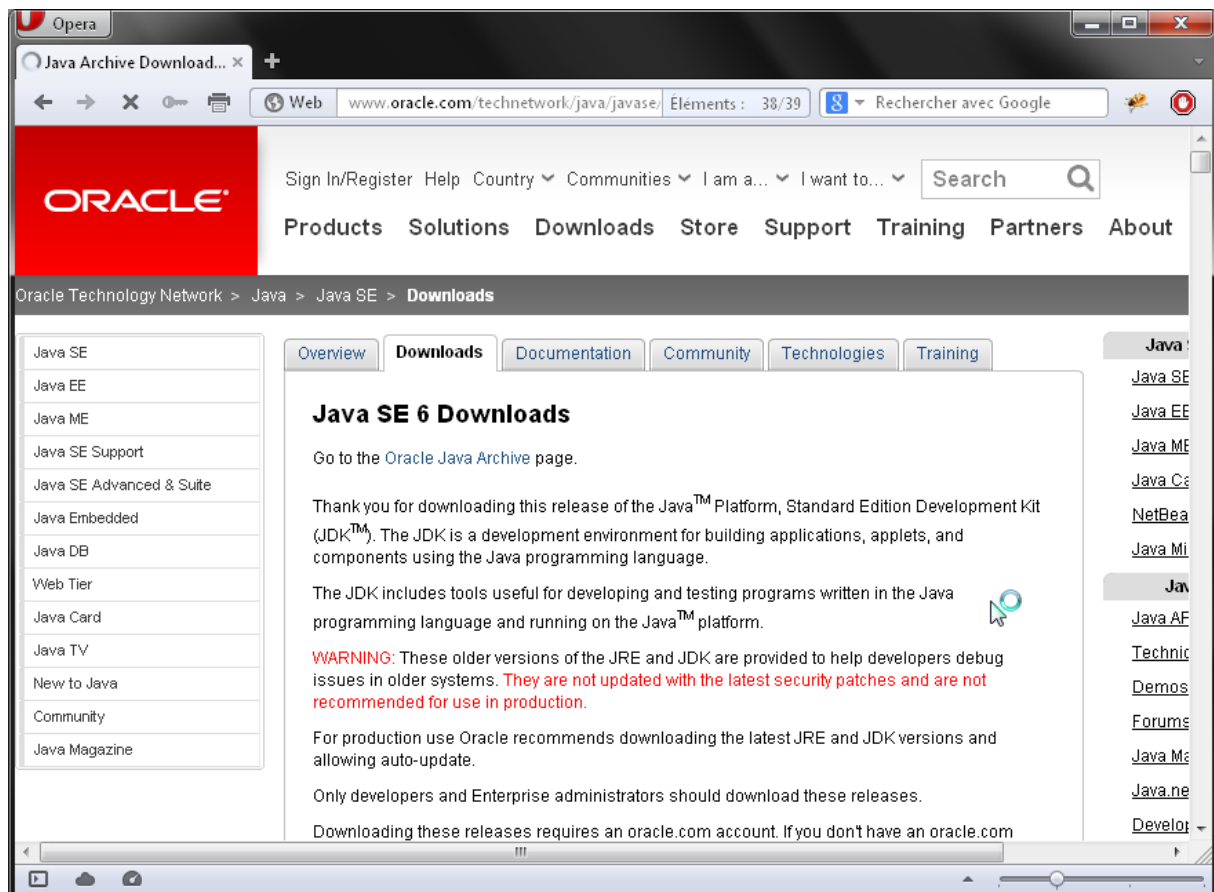


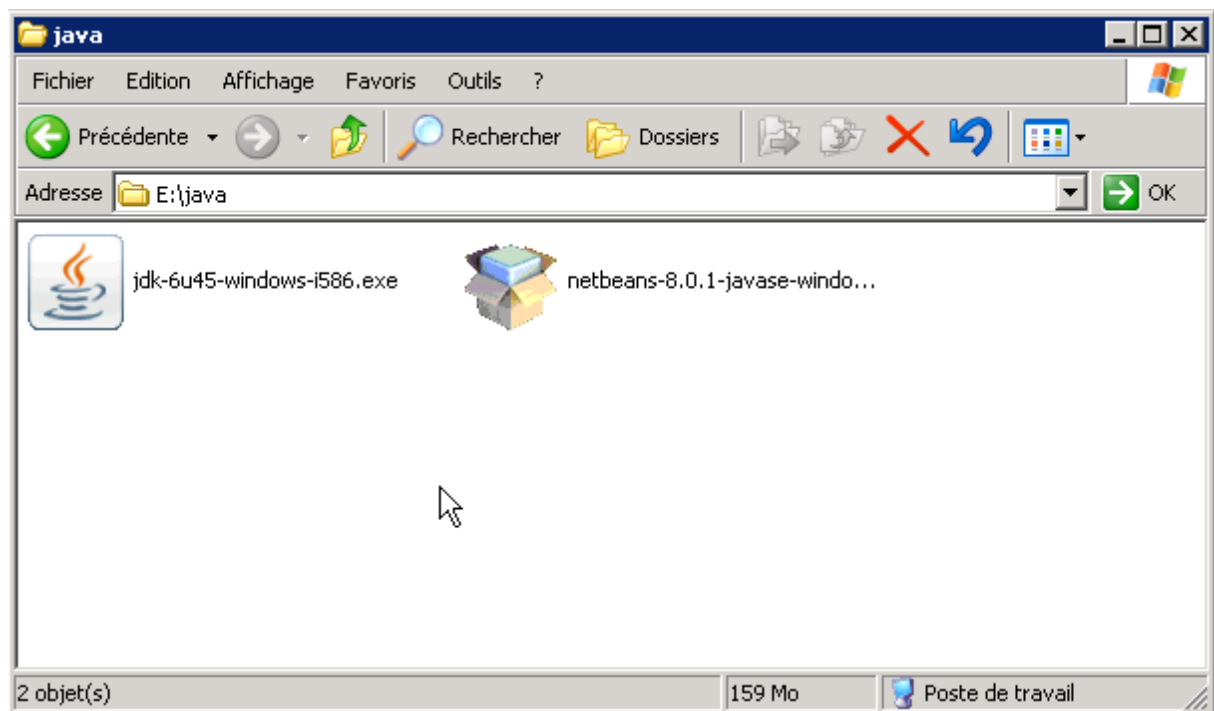


Compile JAR and call from JAVA

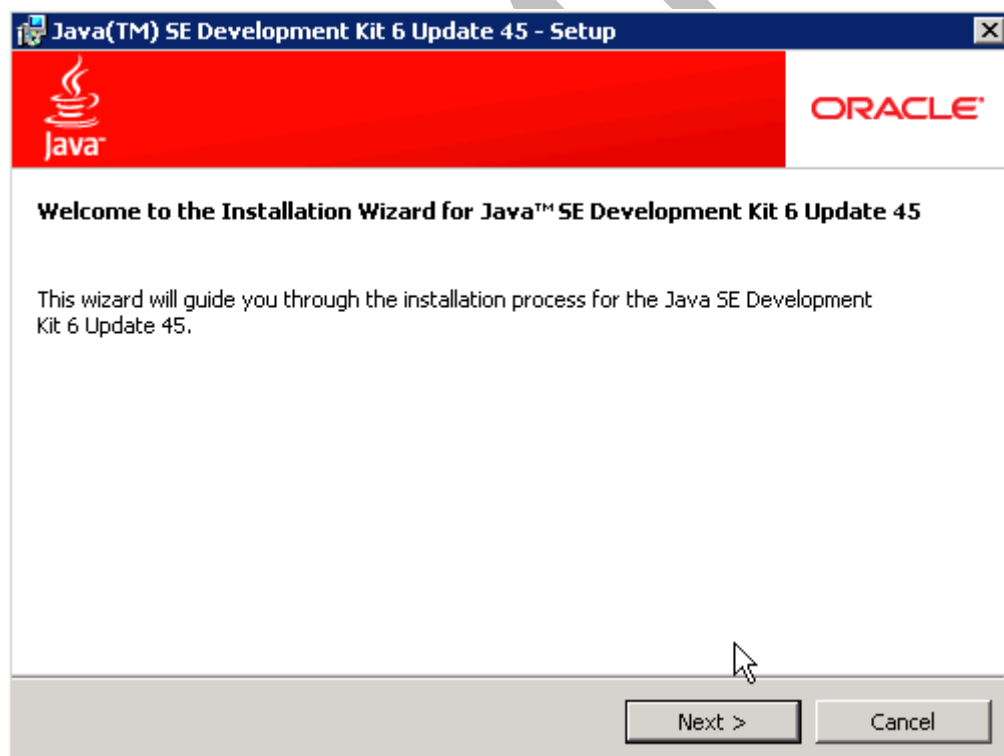


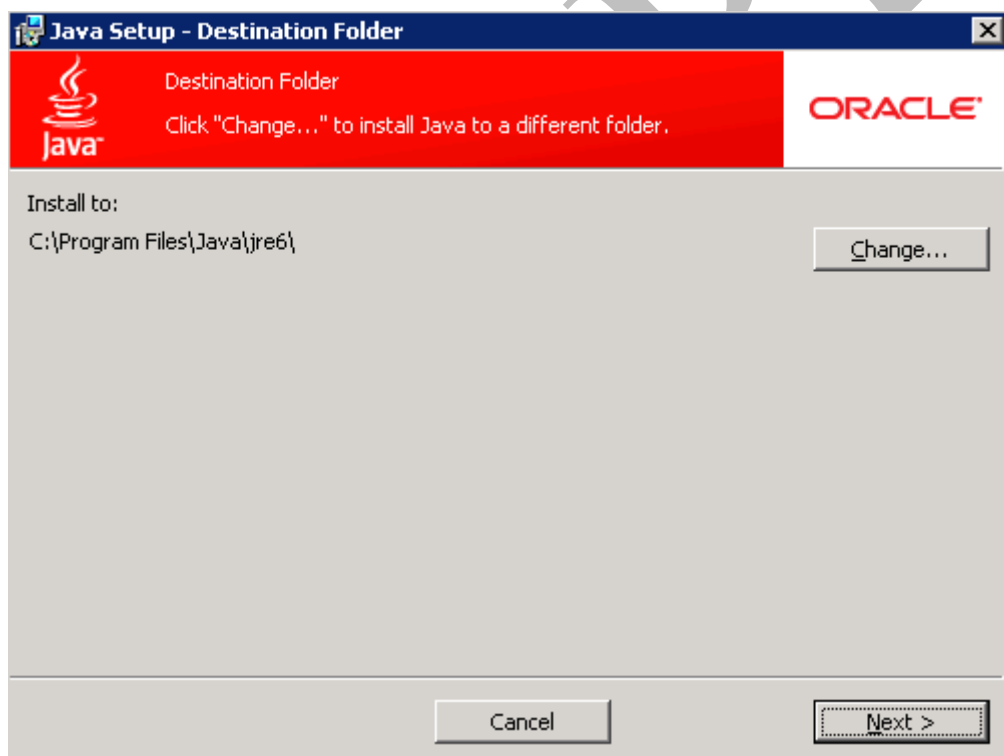
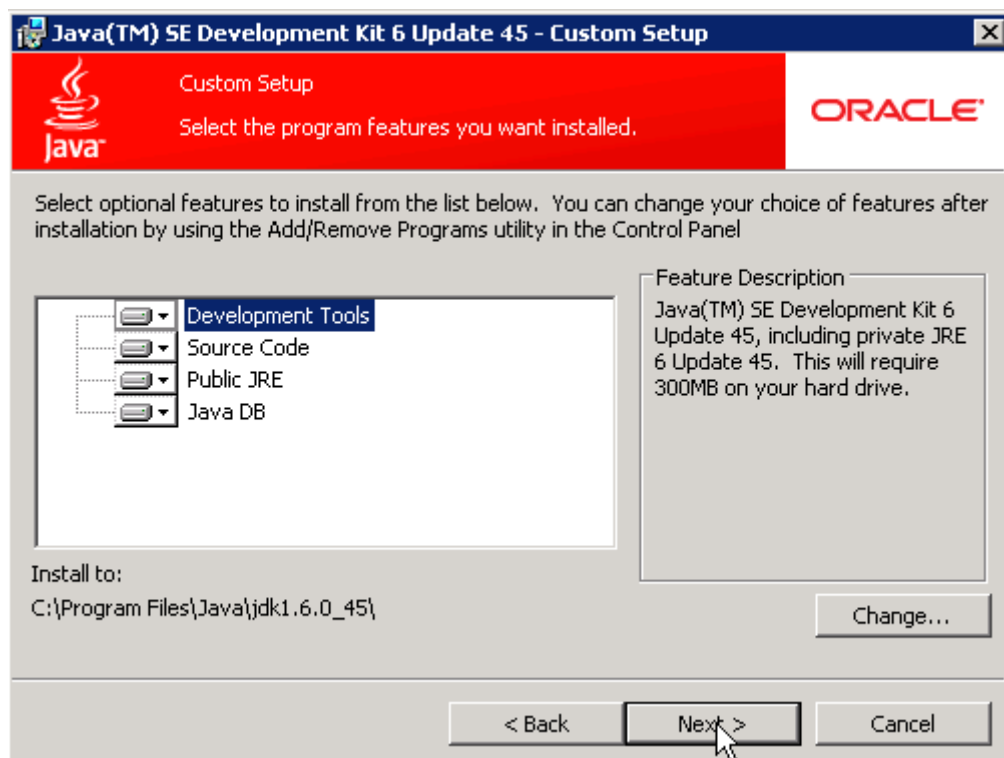
<http://www.oracle.com/technetwork/java/javase/downloads/java-archive-downloads-javase6-419409.html#jdk-6u45-oth-JPR>

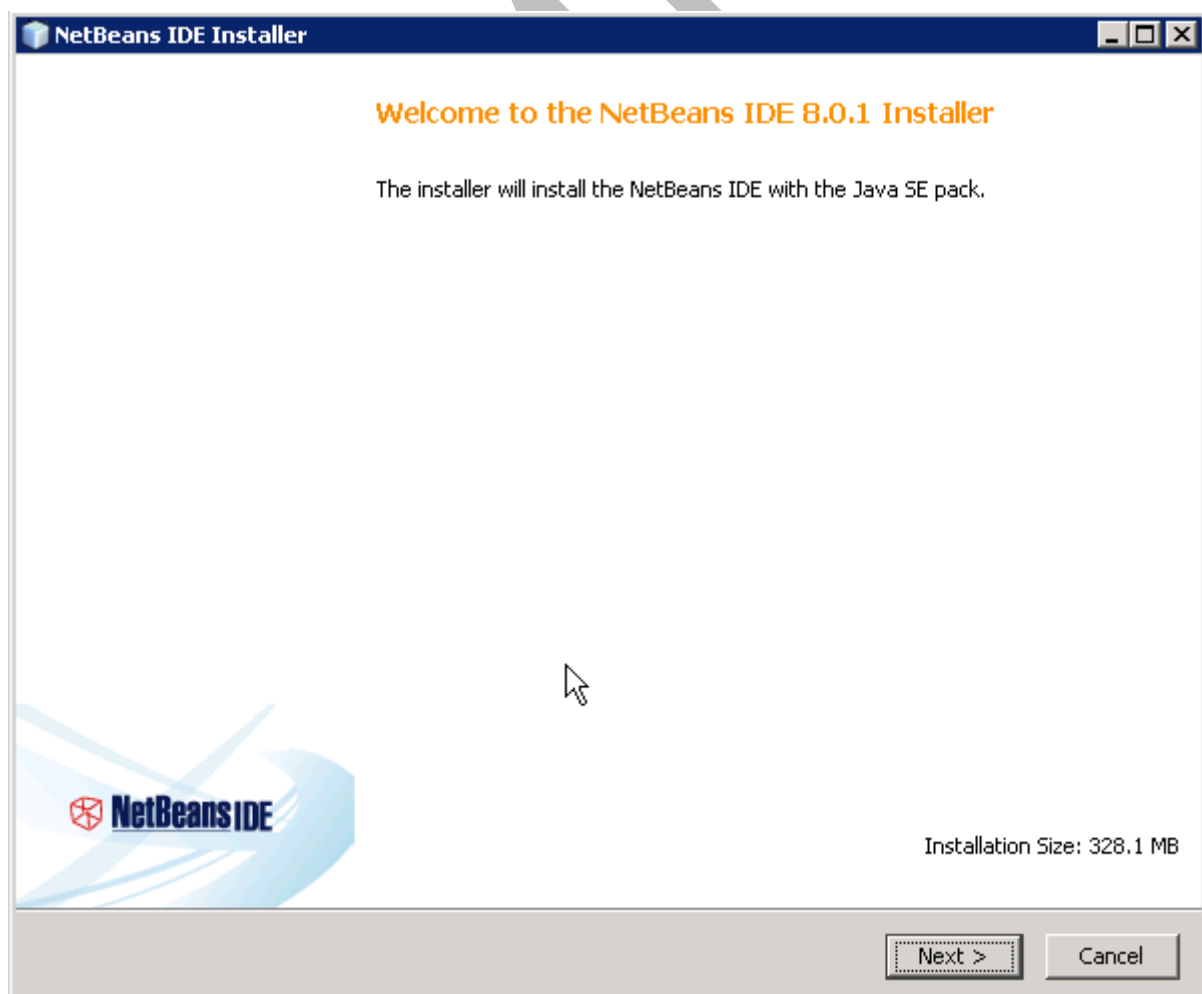
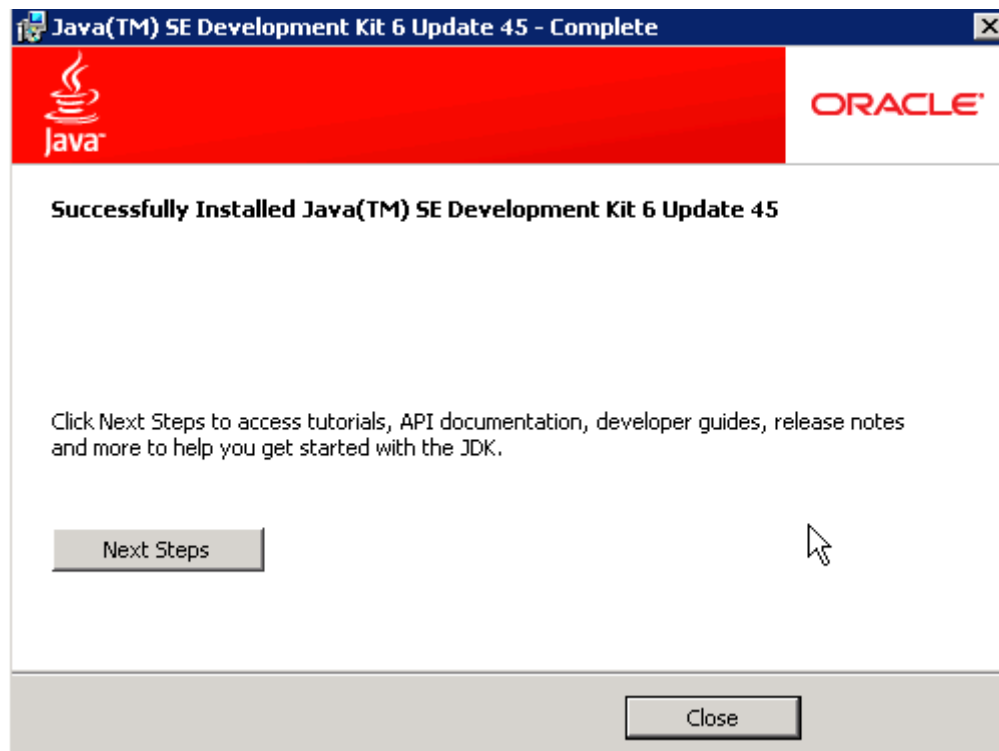


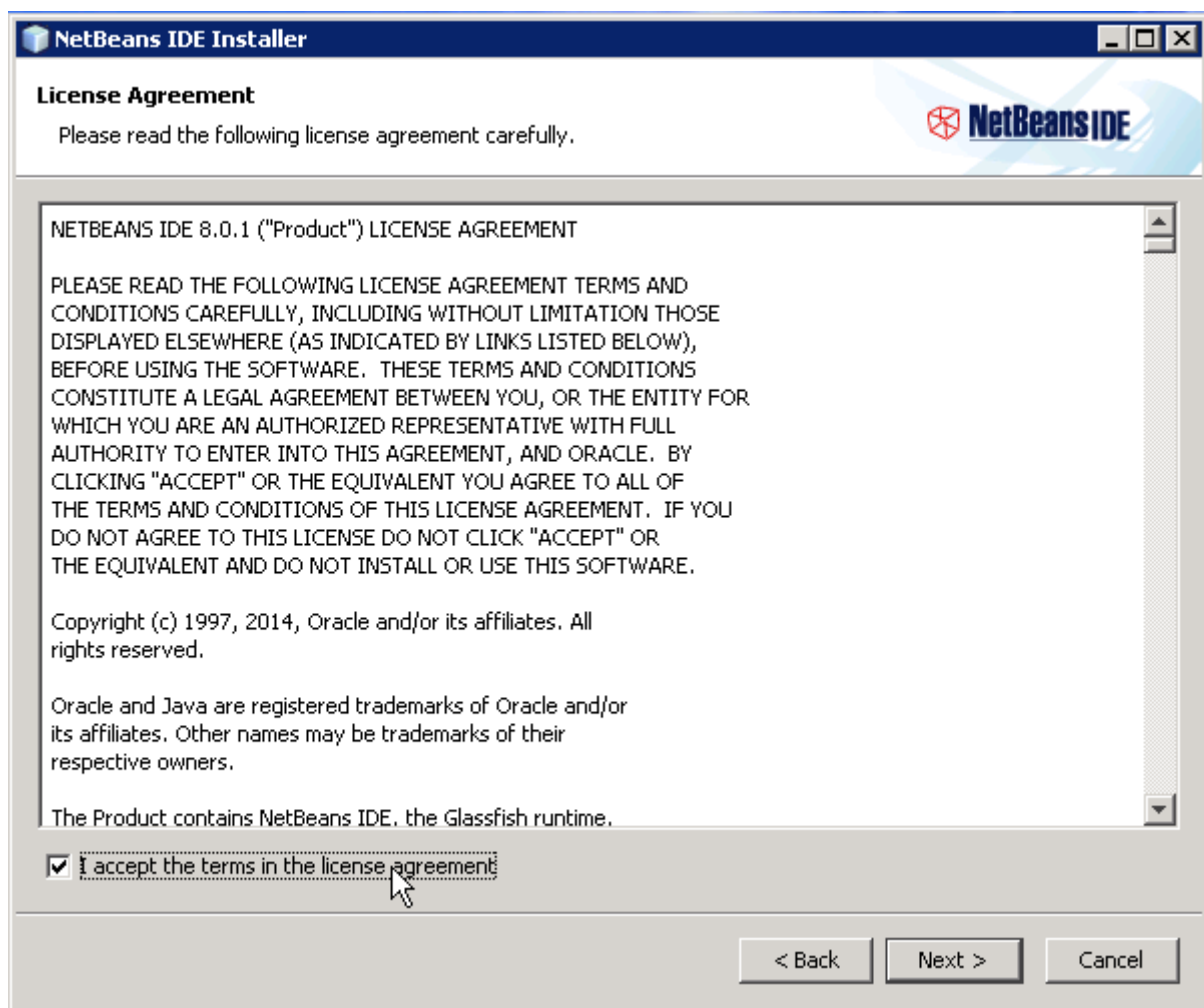


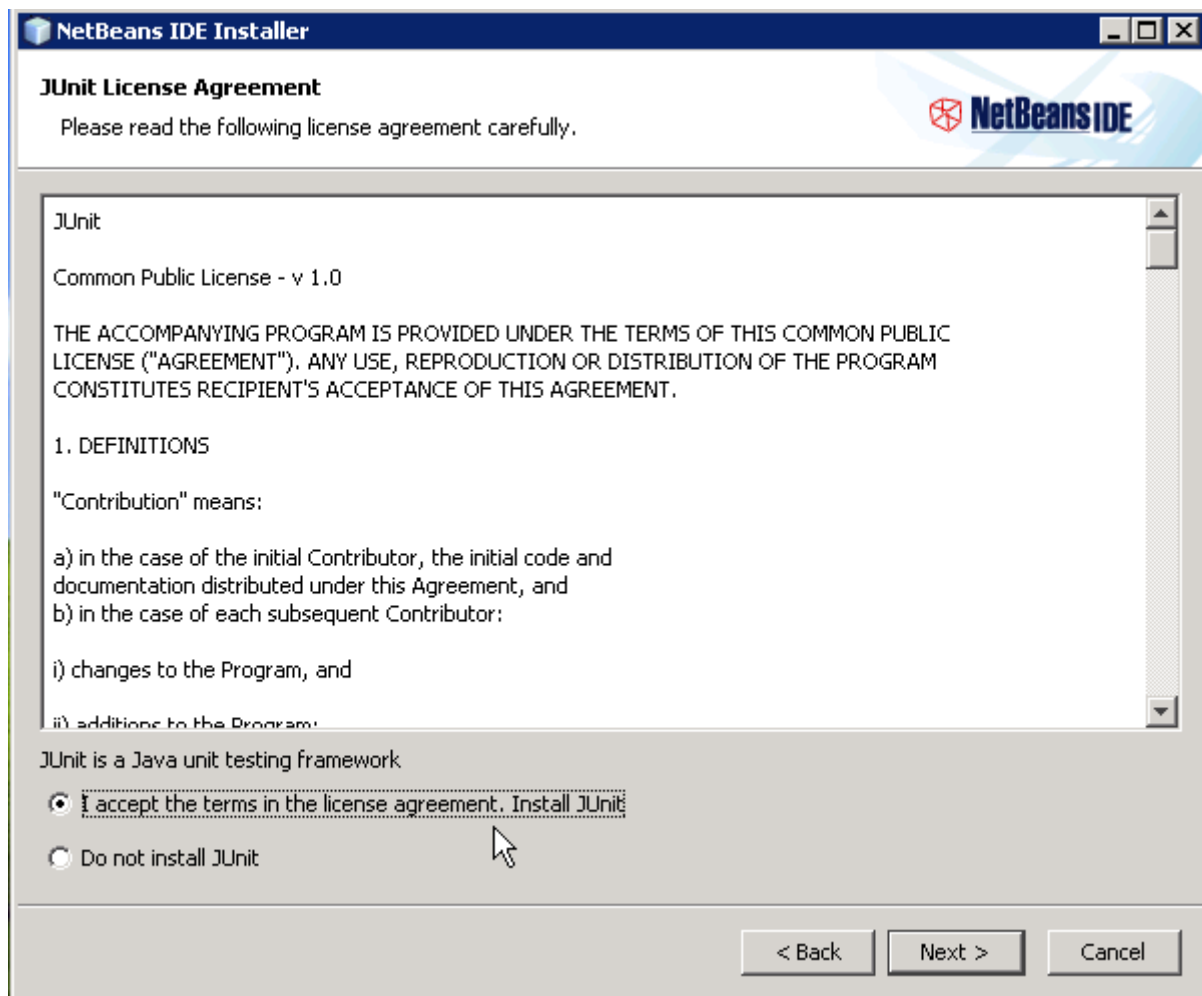
For MATLAB 2013a install JDK 6.45 but for NetBeans install at least the JDK 7.0 (it's not a problem to have more than a version install on the same computer):

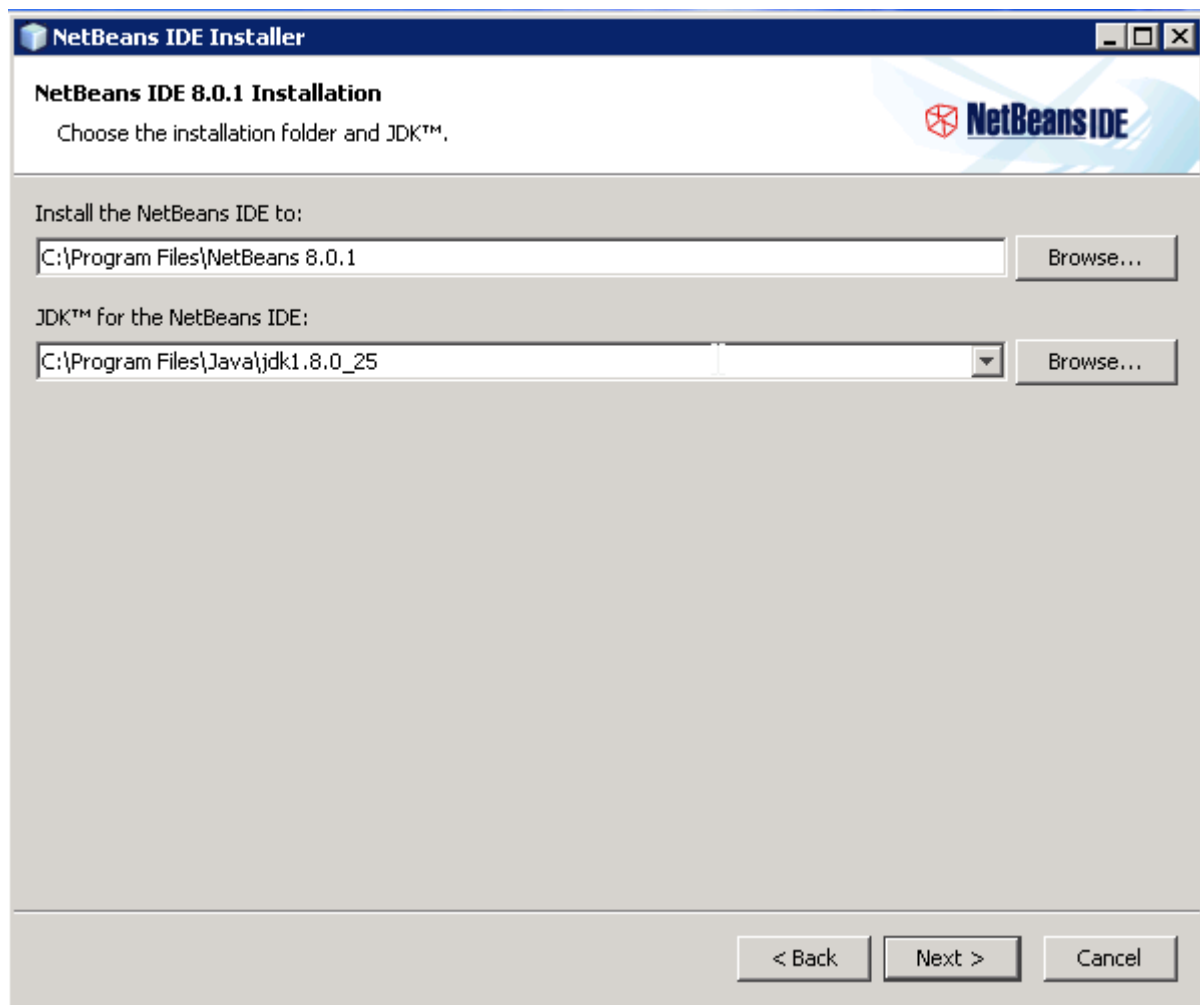


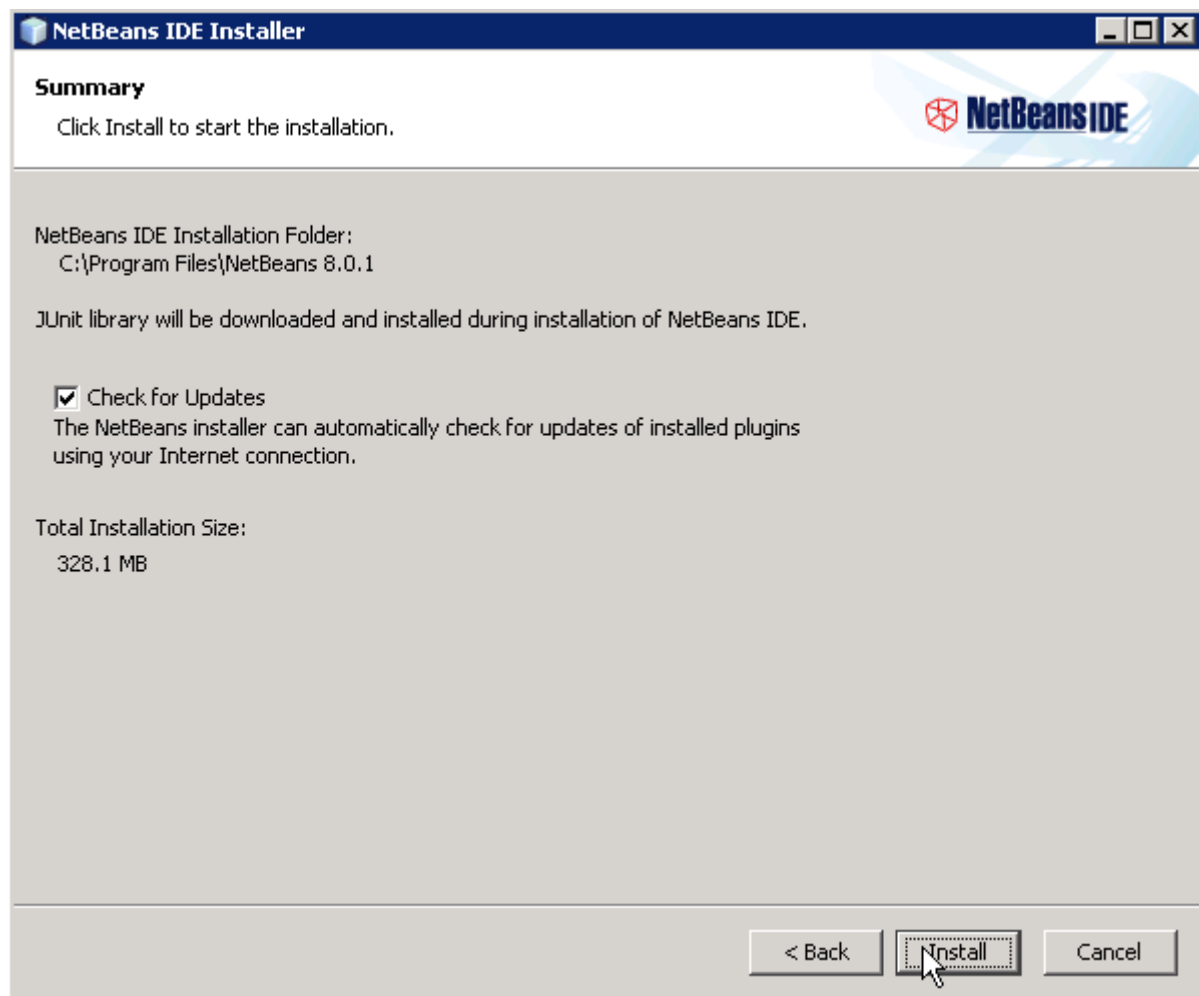


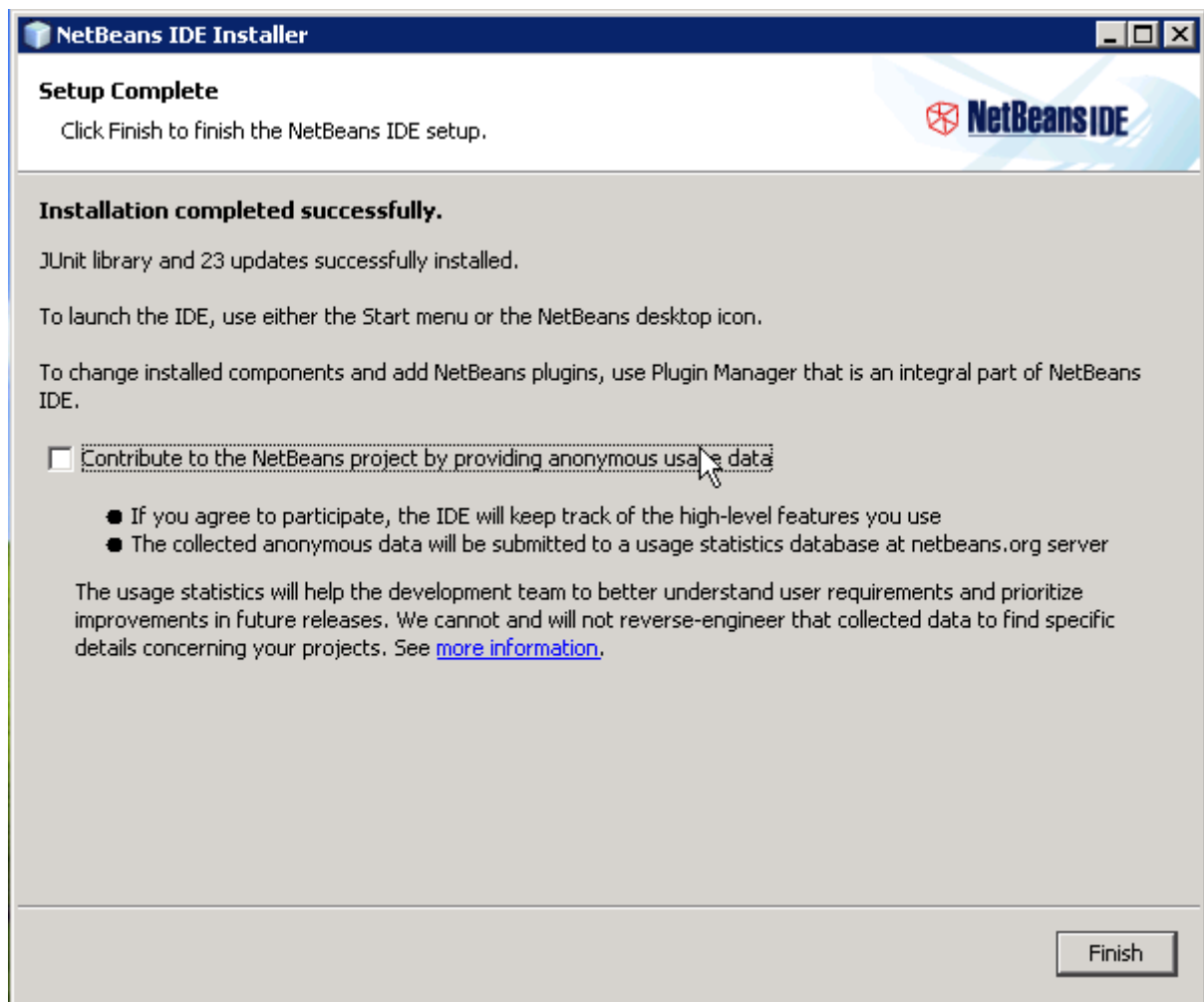


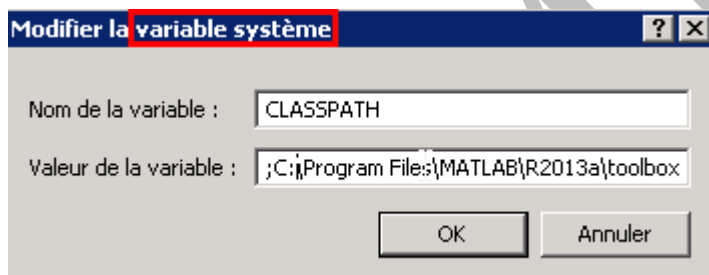
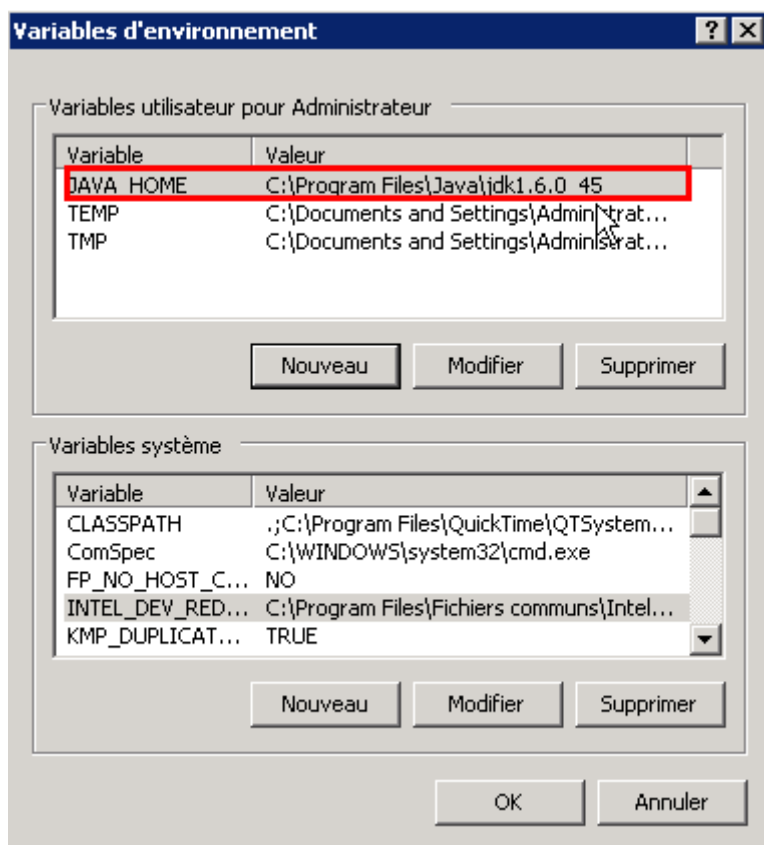




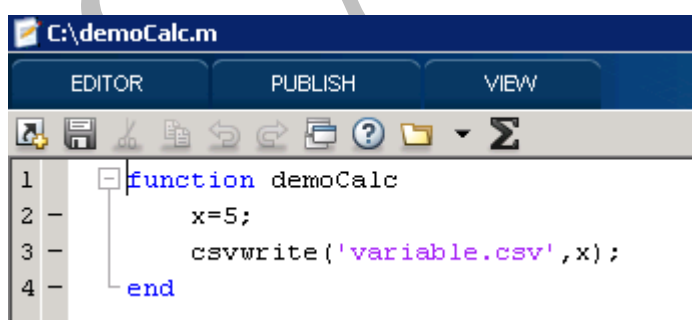


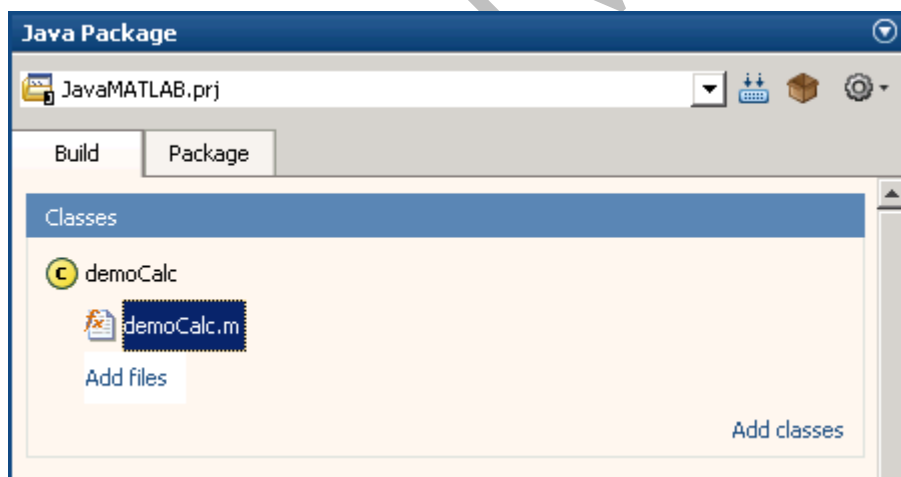
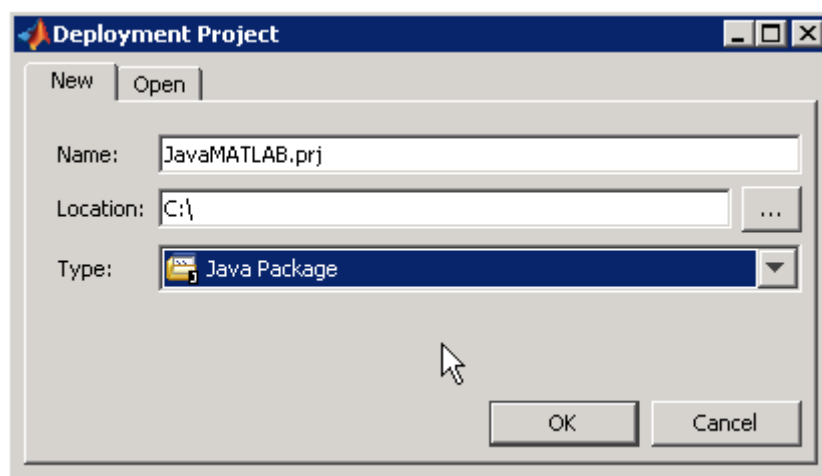
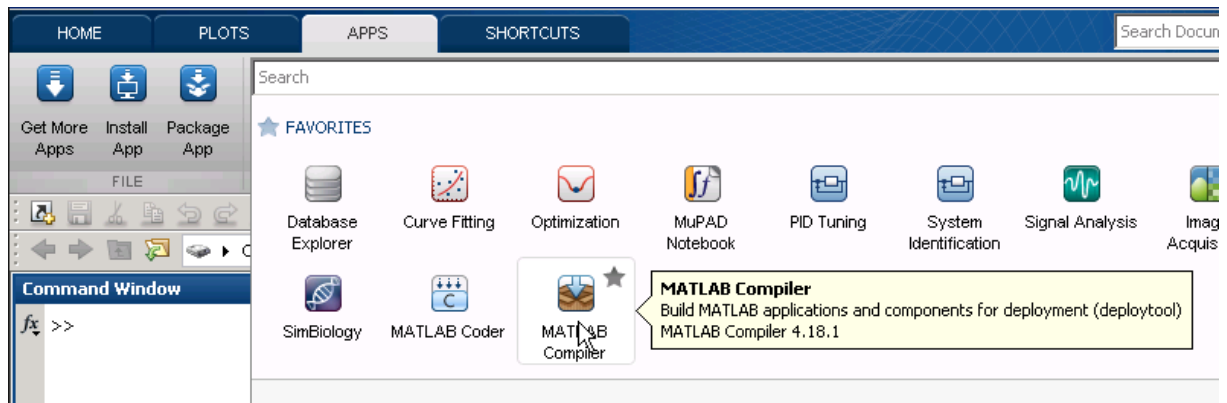


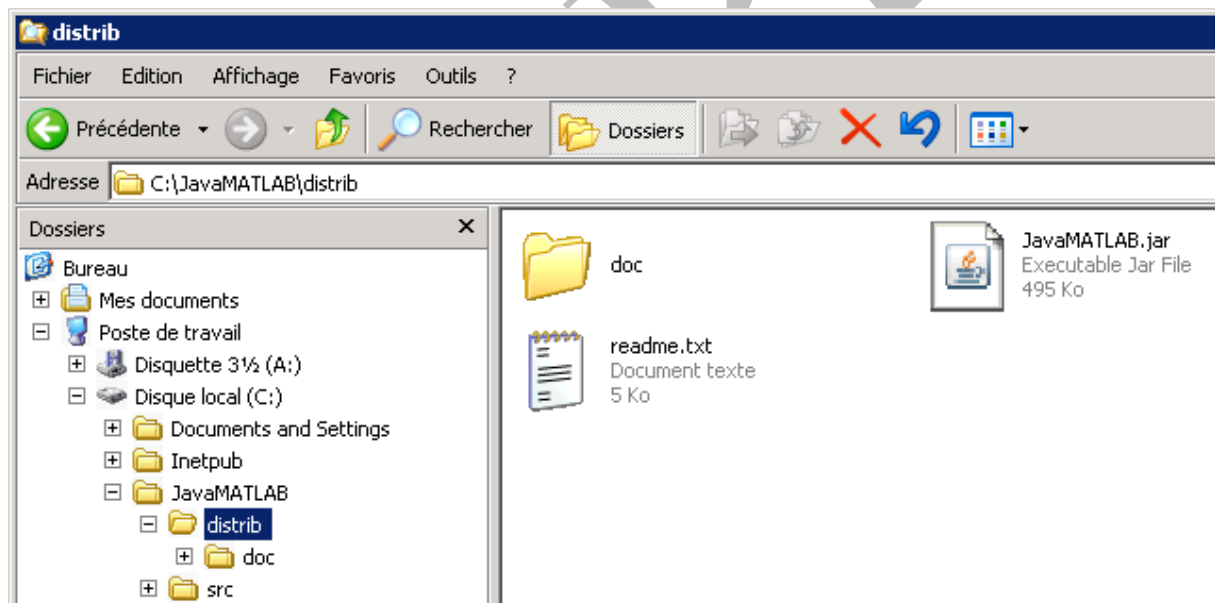
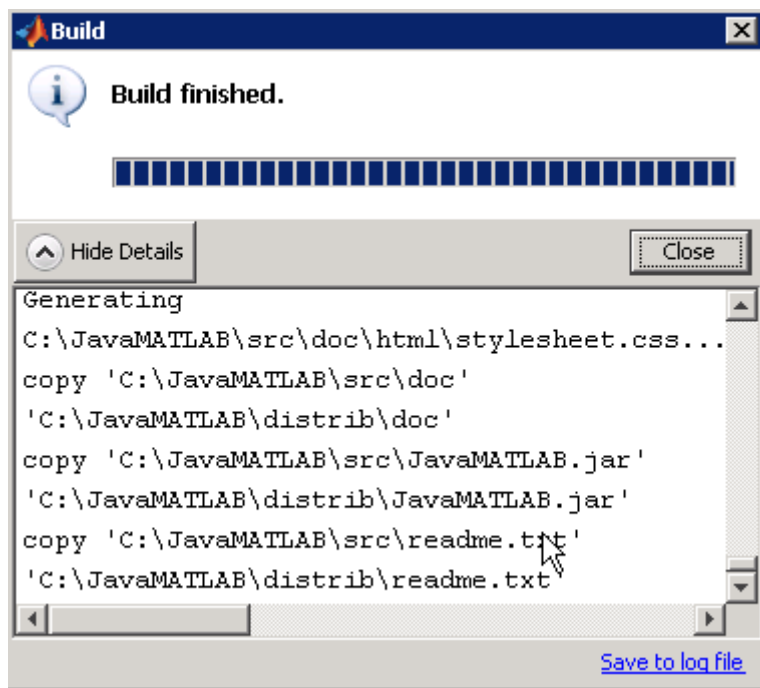


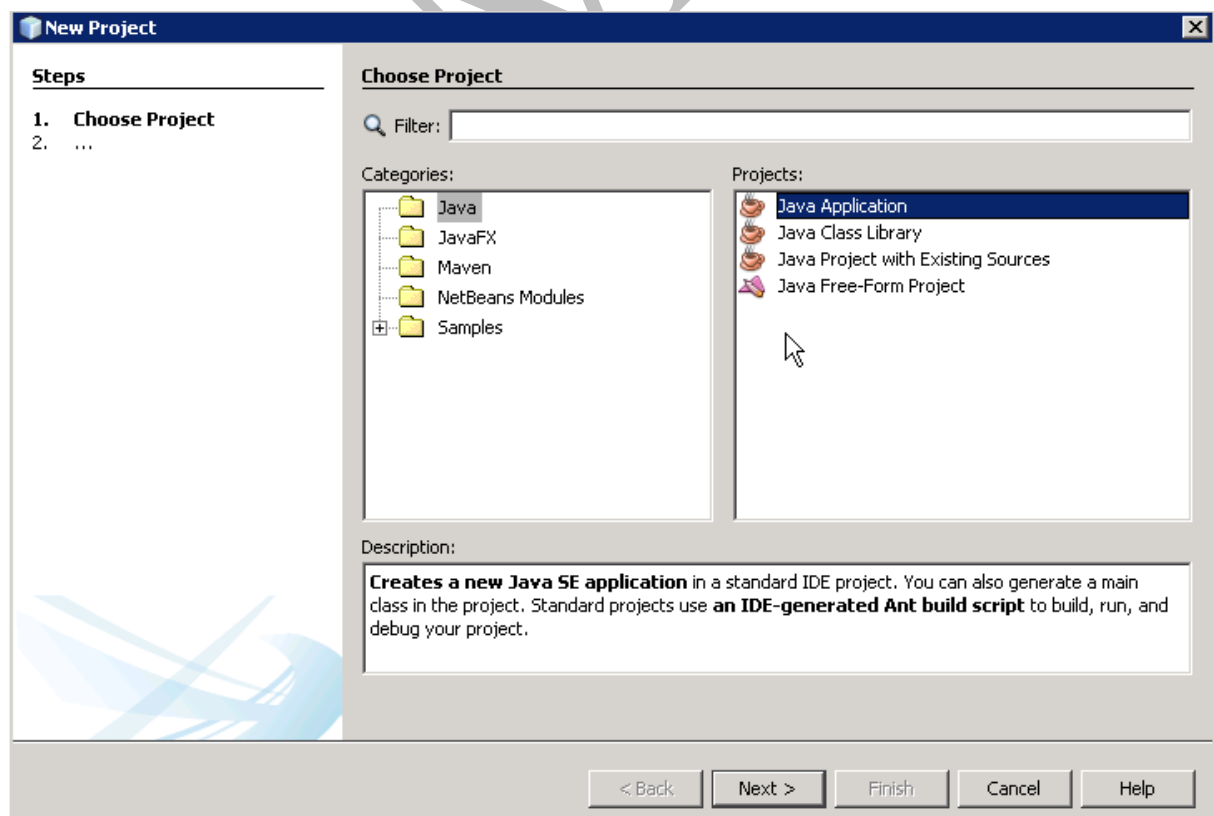
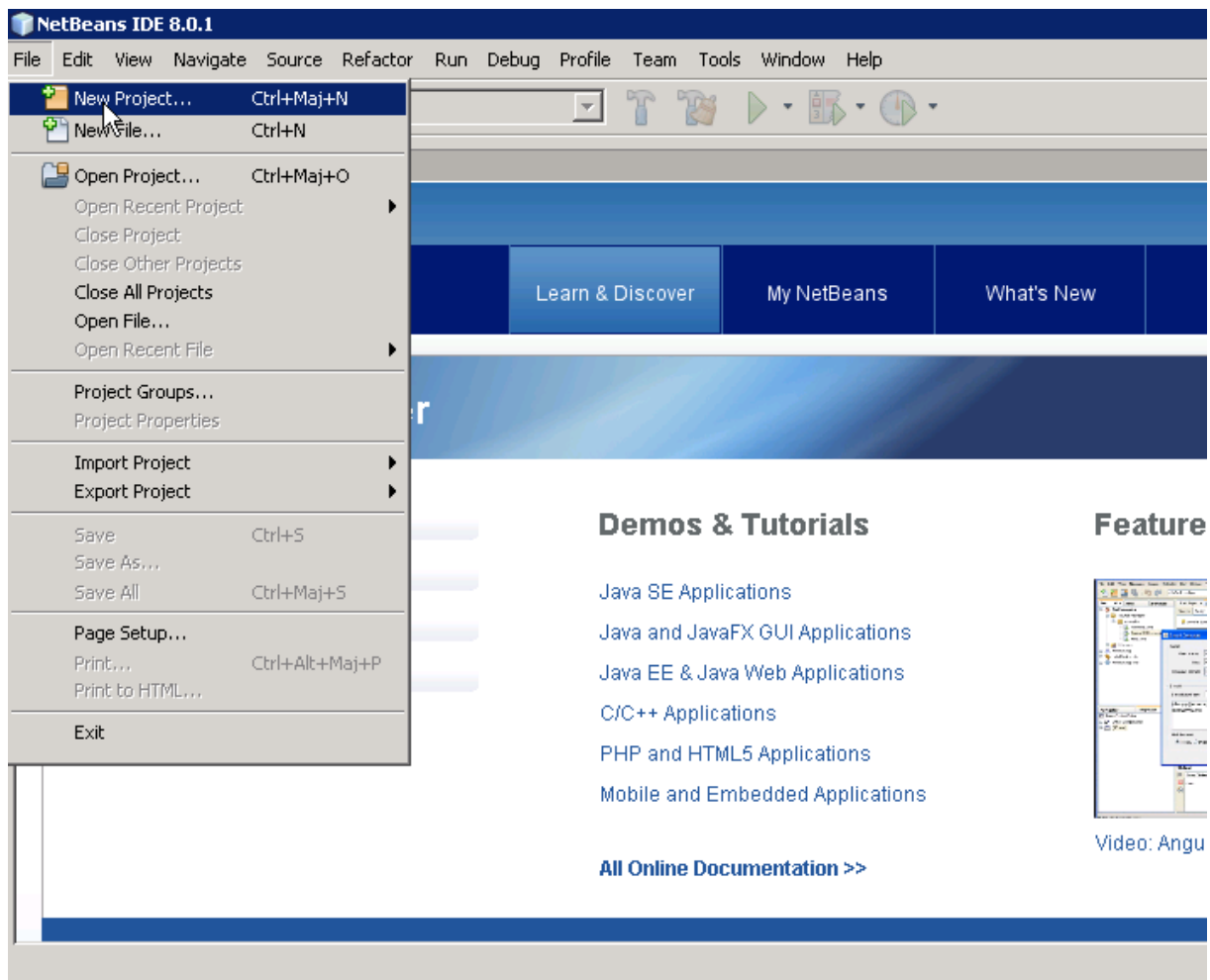


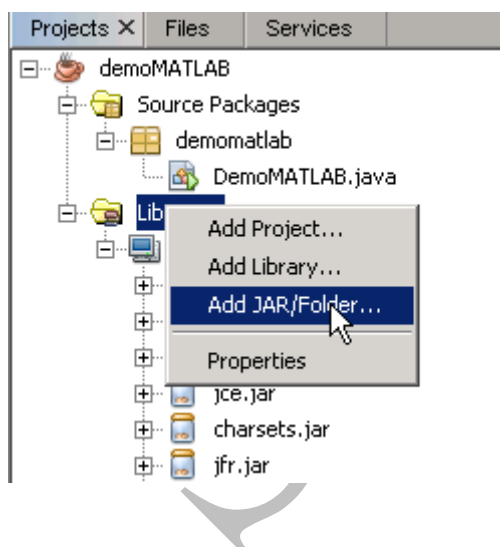
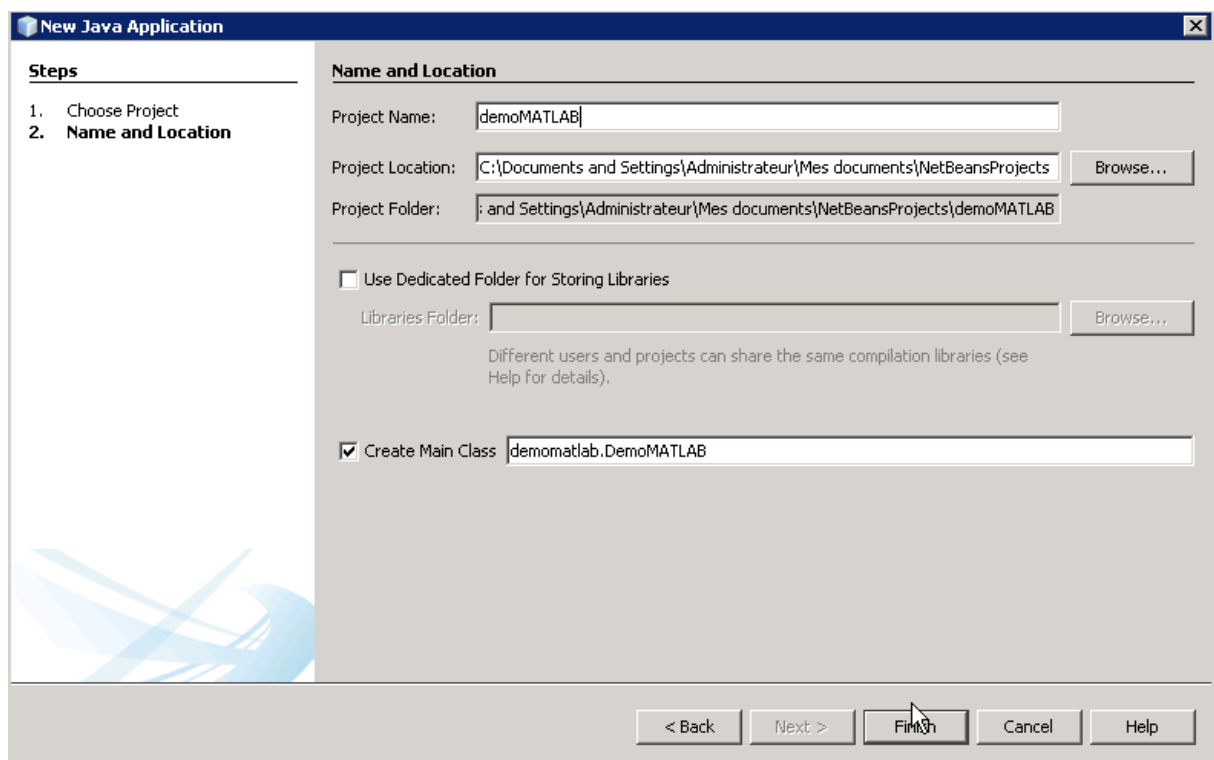
Once this is done don't forget to **restart you computer** (sometimes it helps to avoid issues during compilation).

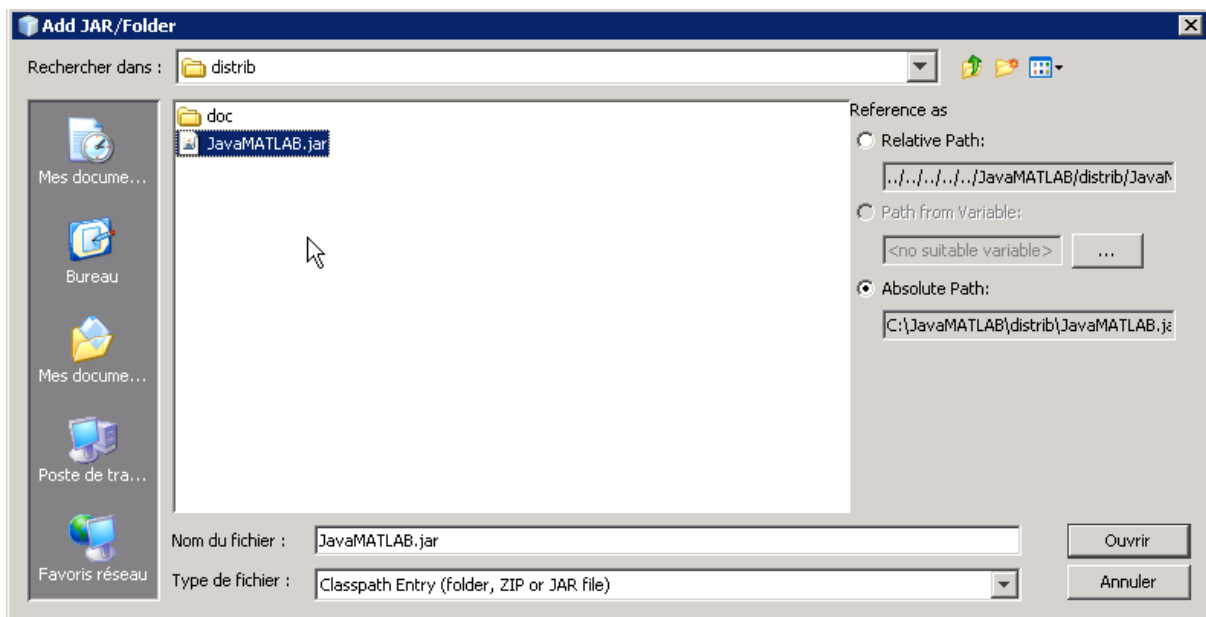




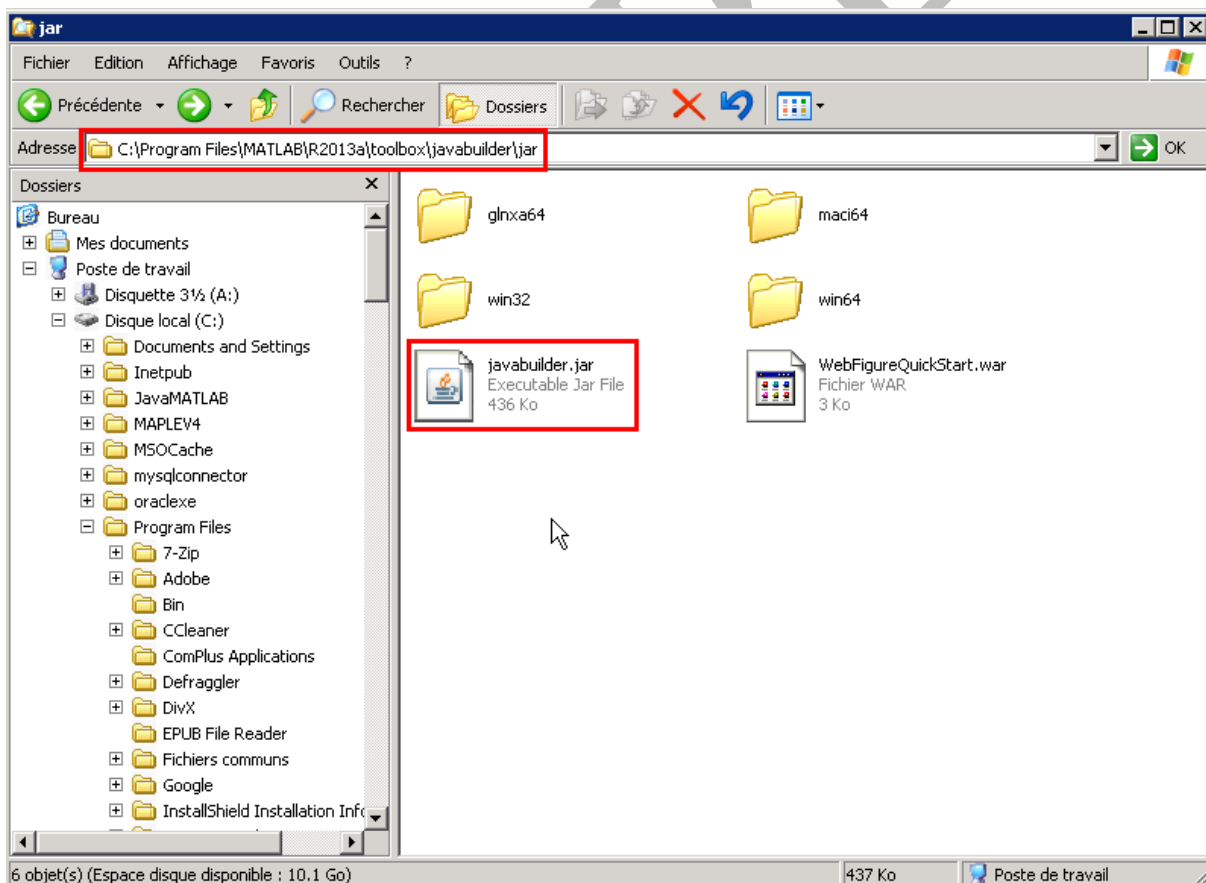




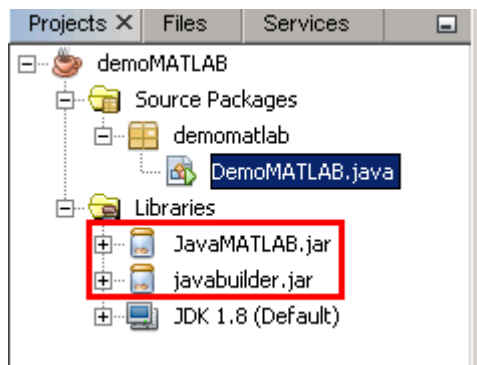




To avoid some problems on some computers also add as reference the MATLAB javabuilder.jar for the toolbox folder:



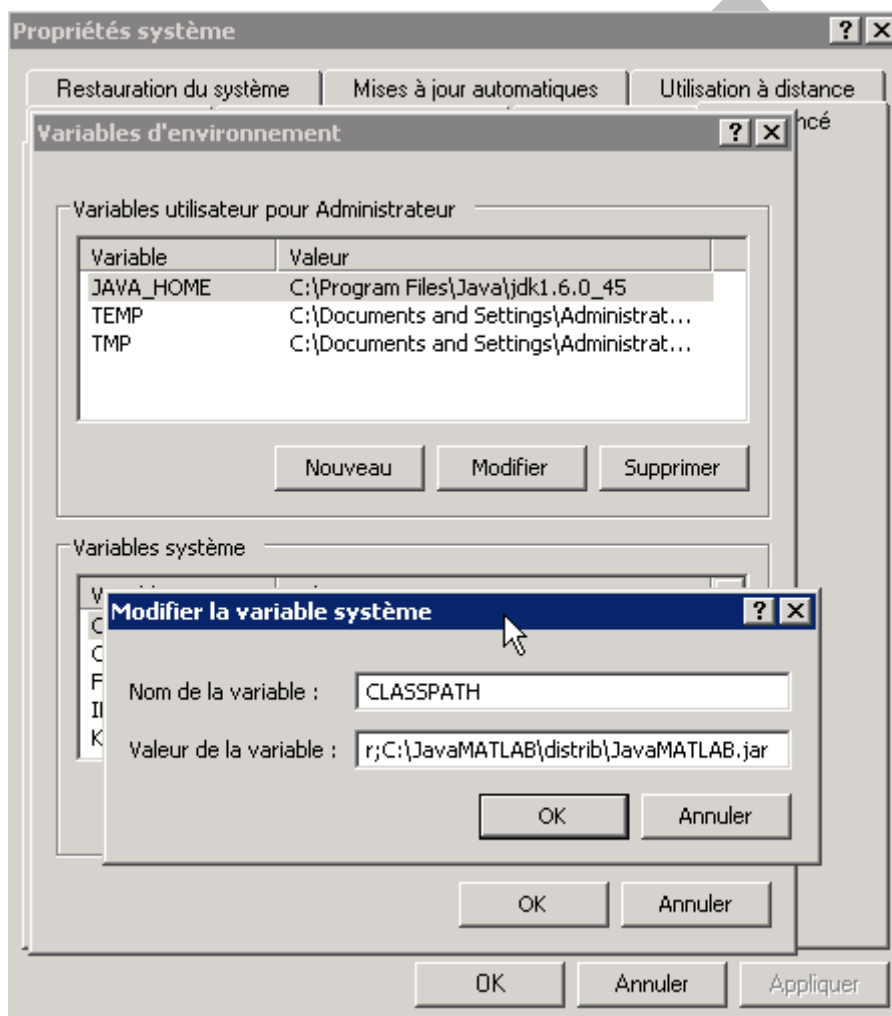
This will give you:



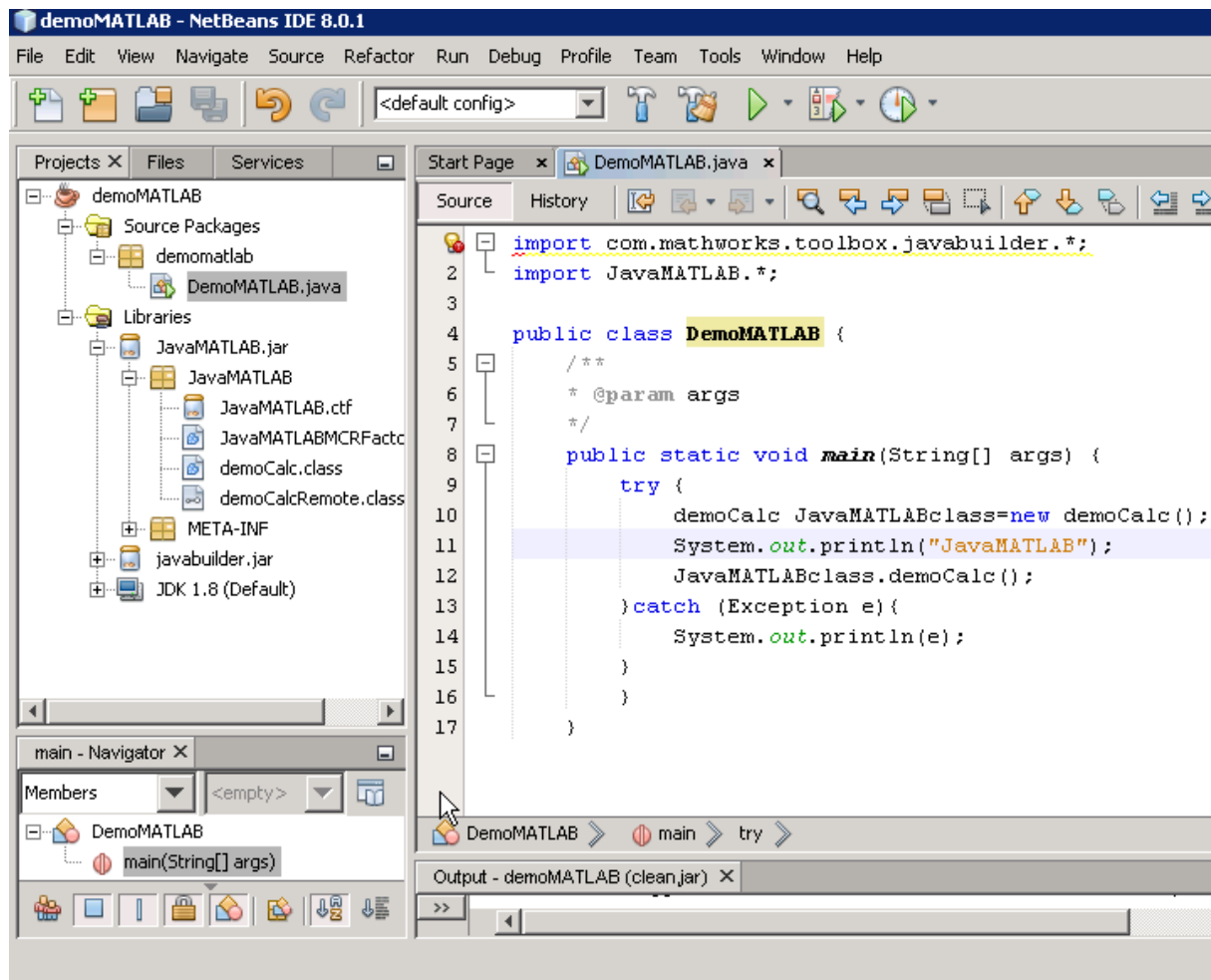
On some computers it is also necessary to add the following files and path to system variables:

C:\Program Files\MATLAB\R2013a\toolbox\javabuilder\jar\javabuilder.jar

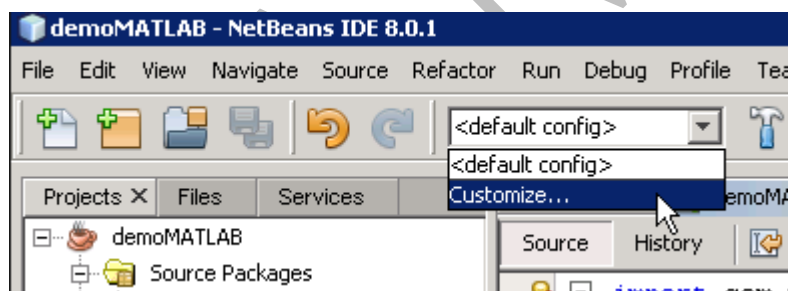
C:\JavaMATLAB\distrib\JavaMATLAB.jar

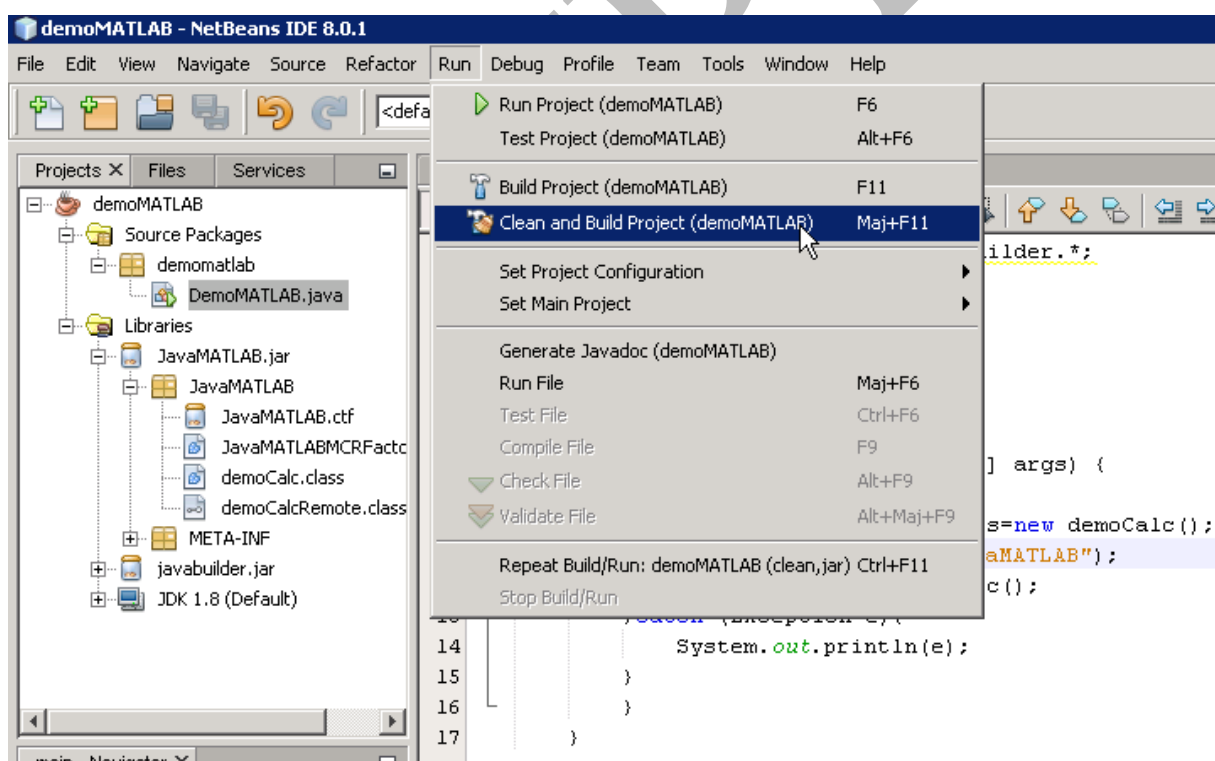
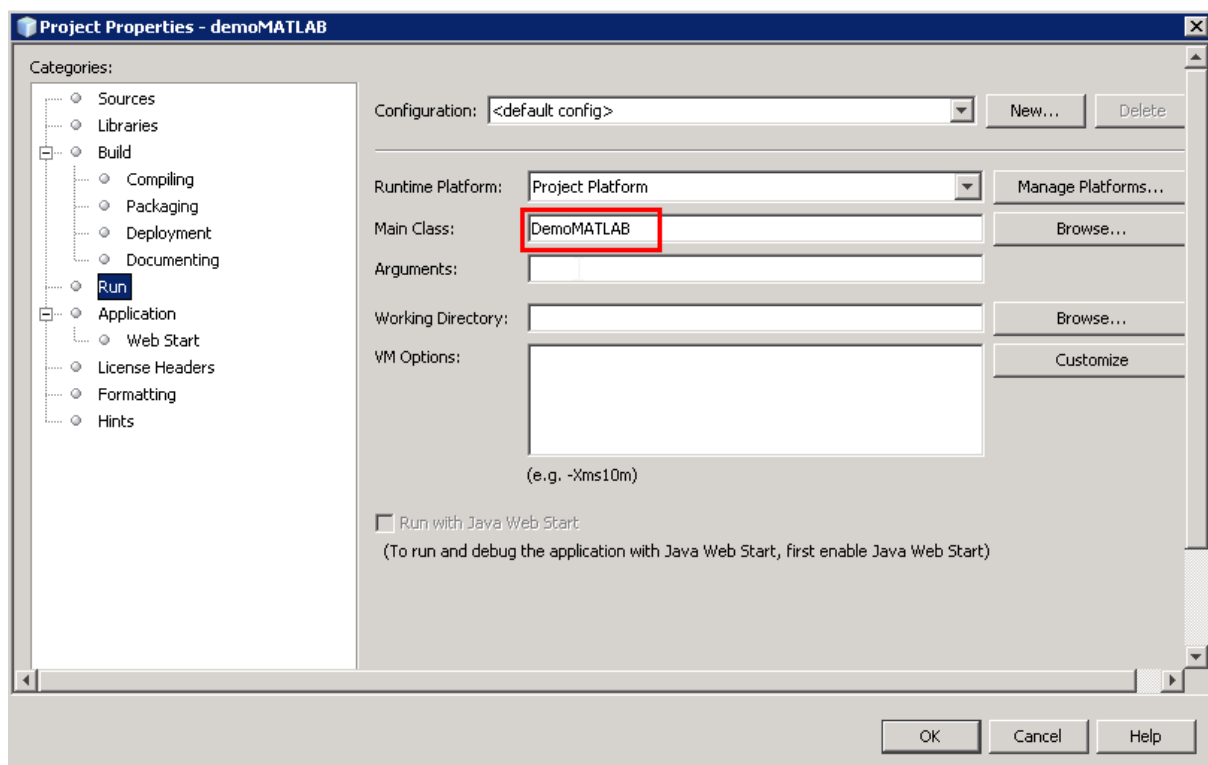


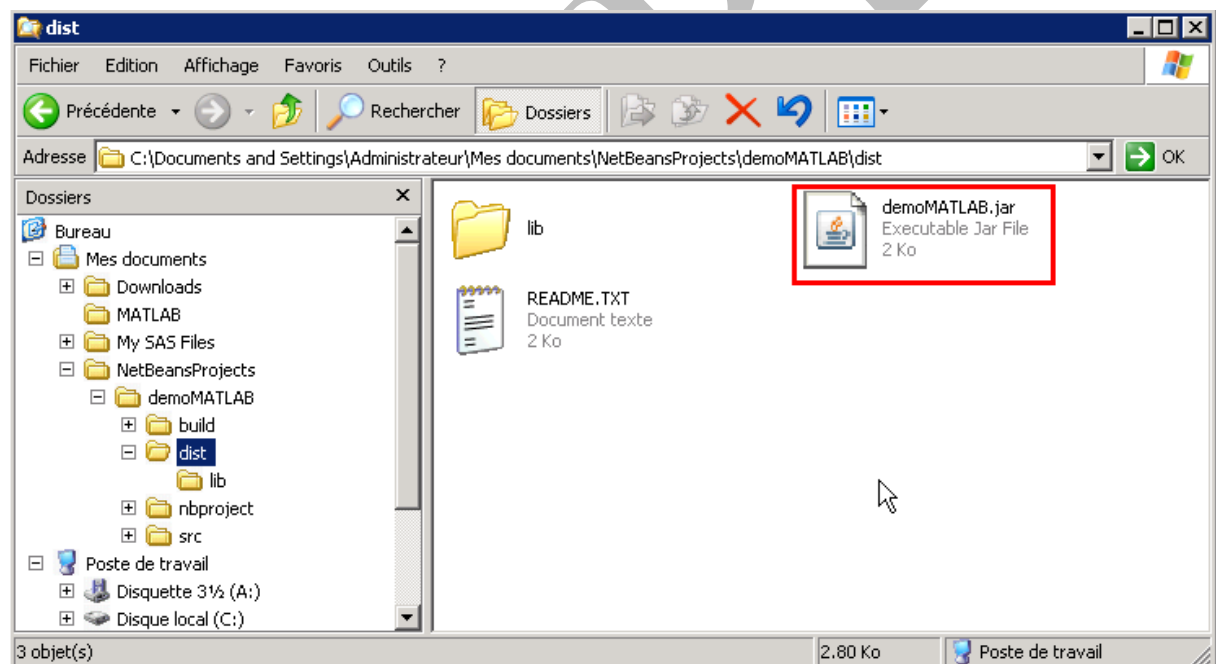
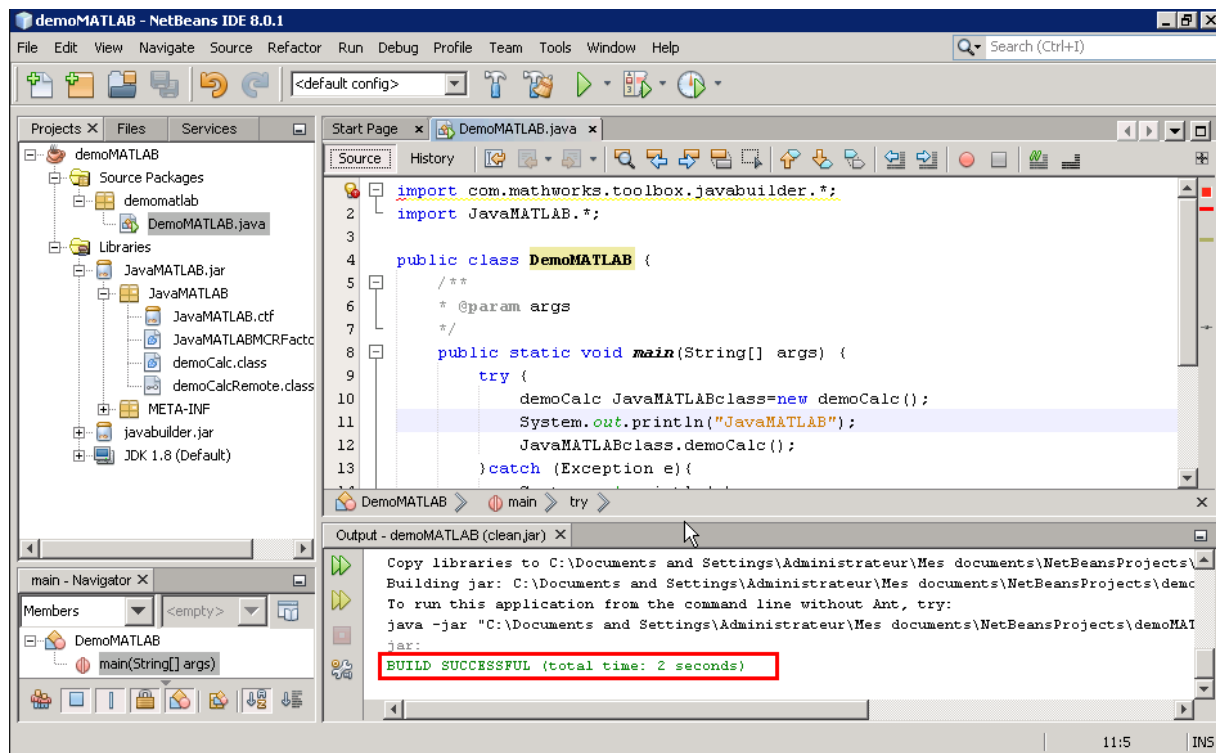
Once this is done don't forget to **restart you computer** (sometimes it helps to avoid issues during compilation).



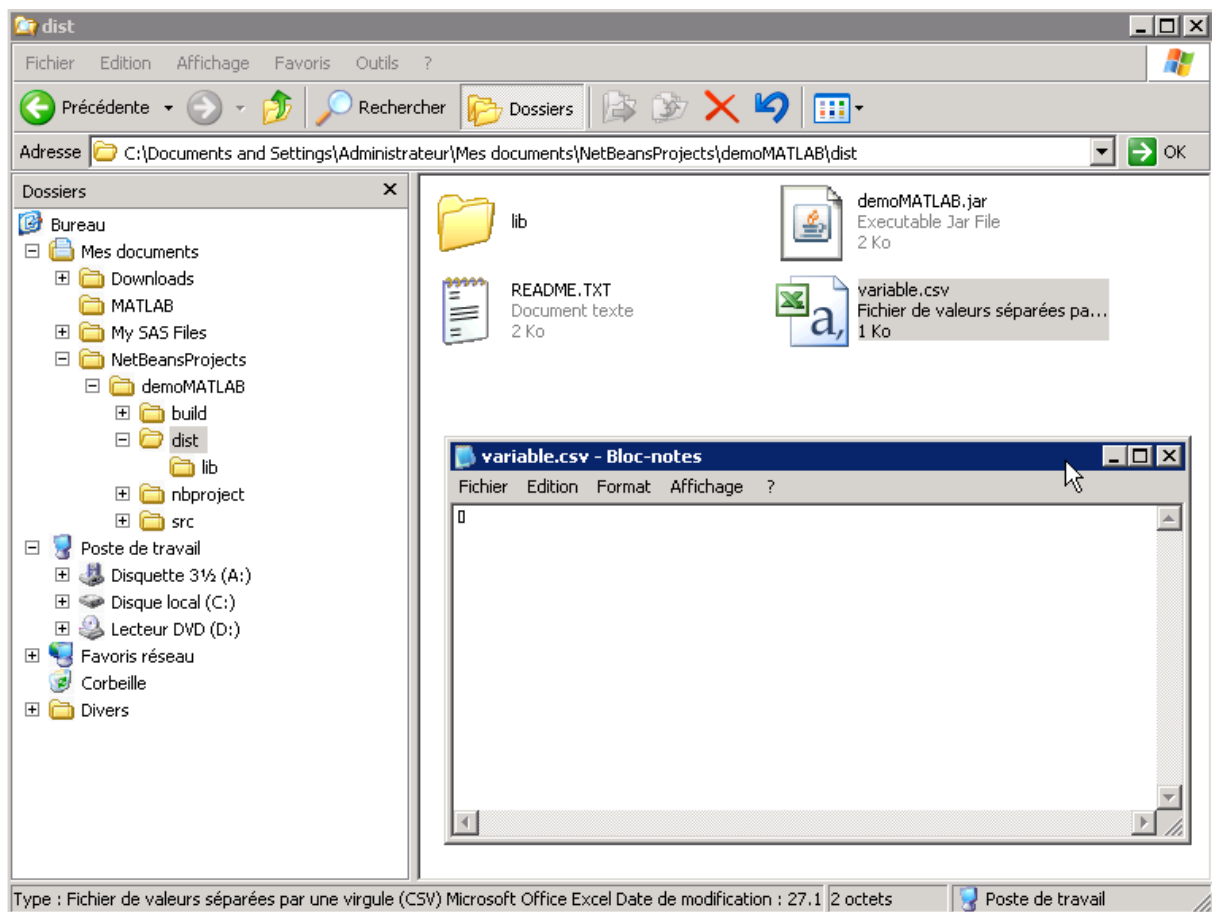
Before building the project check that the class name is really correct:



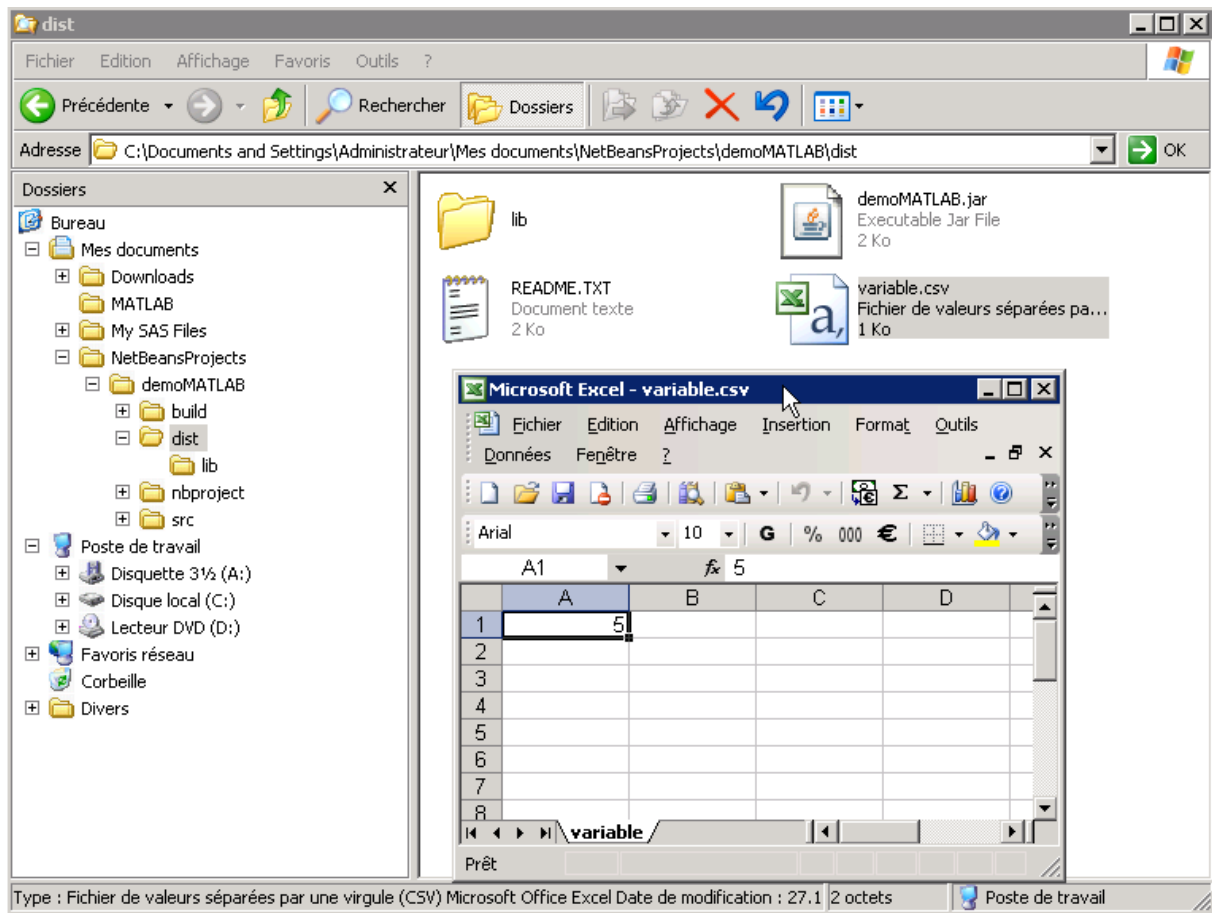




We get the expected result but when we open the file in the NotePad:

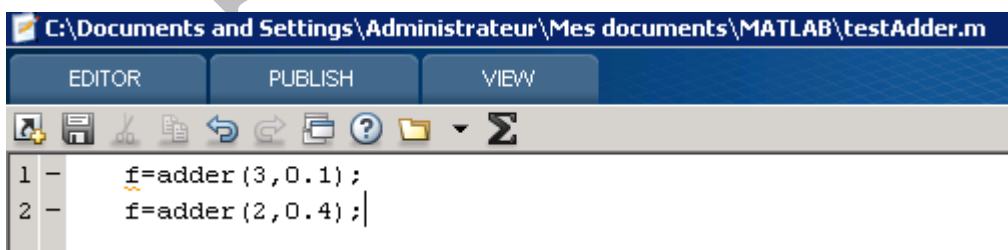
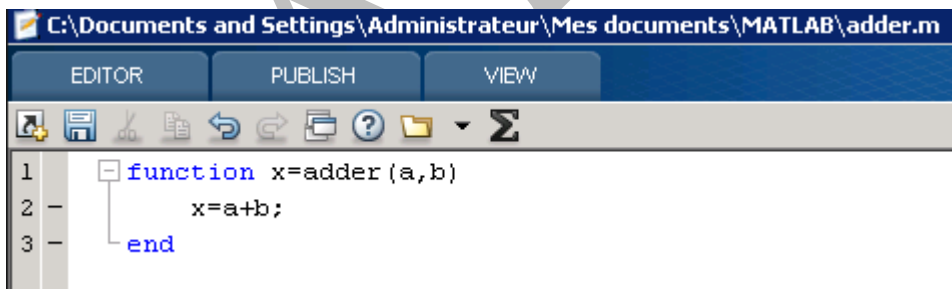


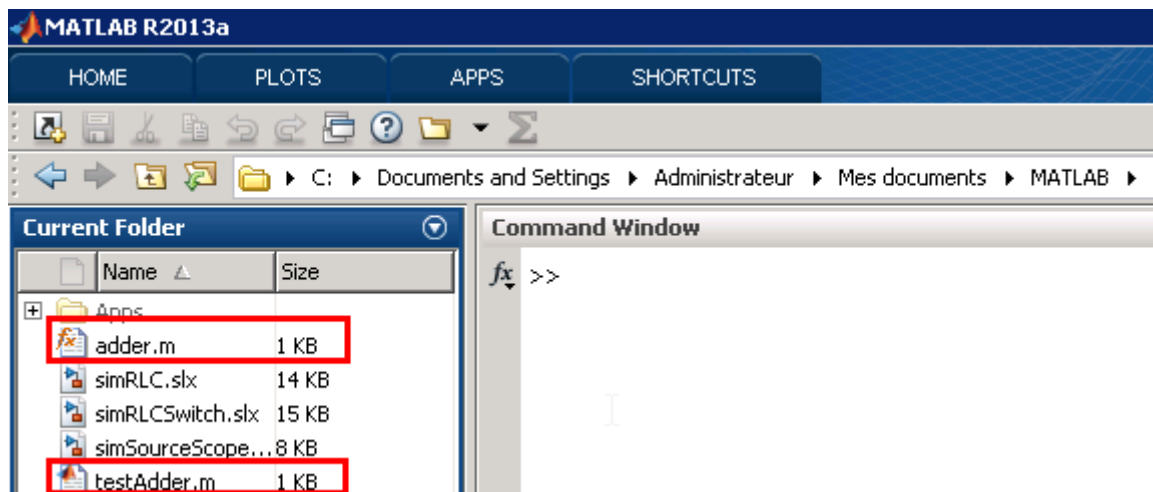
... And if we open it in Microsoft Excel:



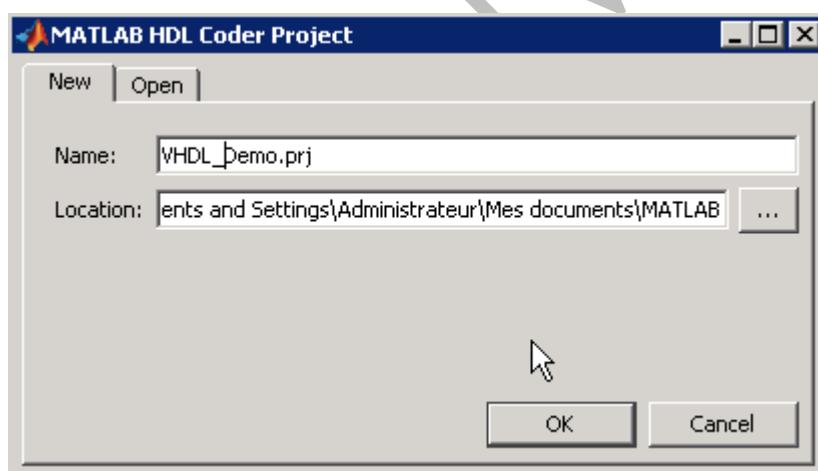
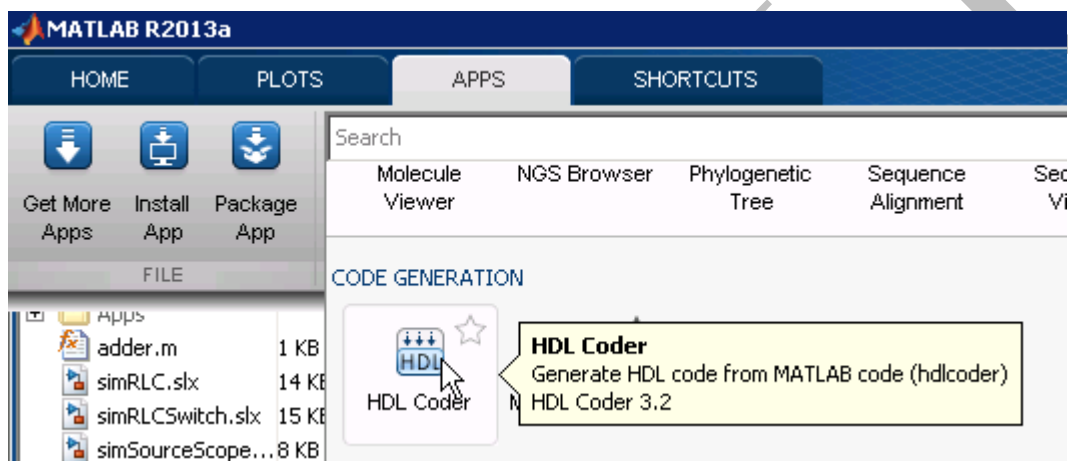
VHDL

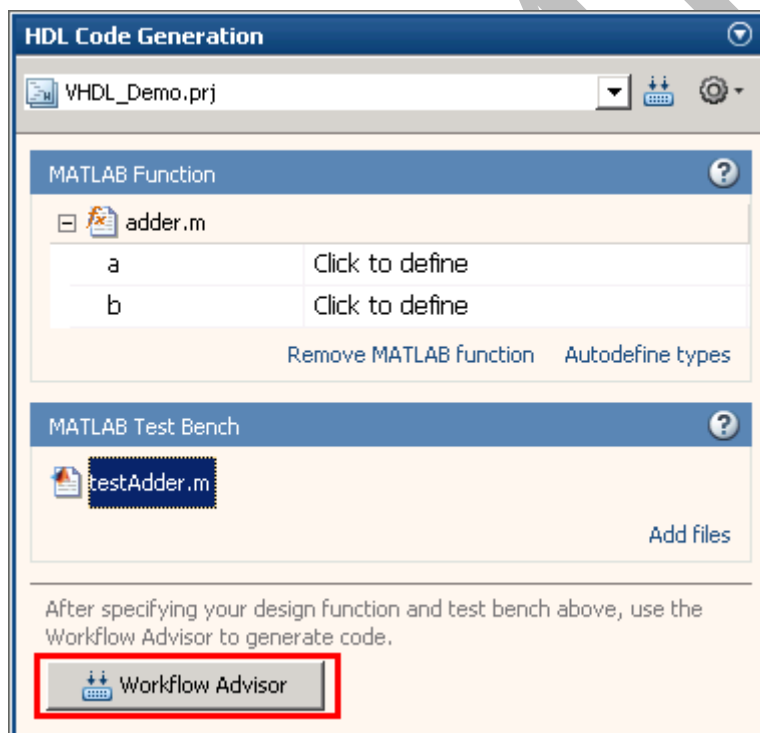
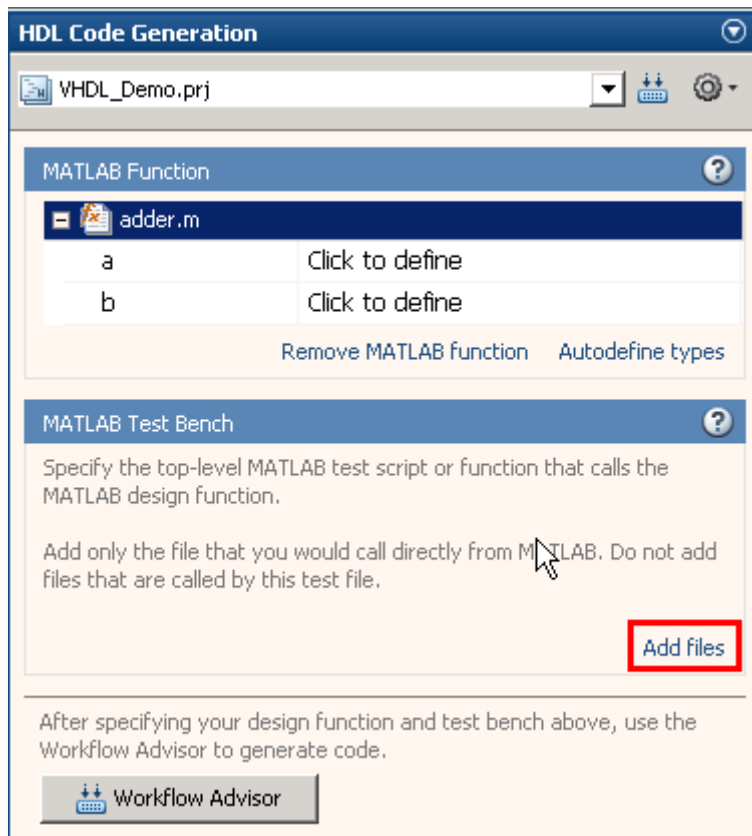
We will use two scripts for the topic. One containing the script and one to call it for test purposes.

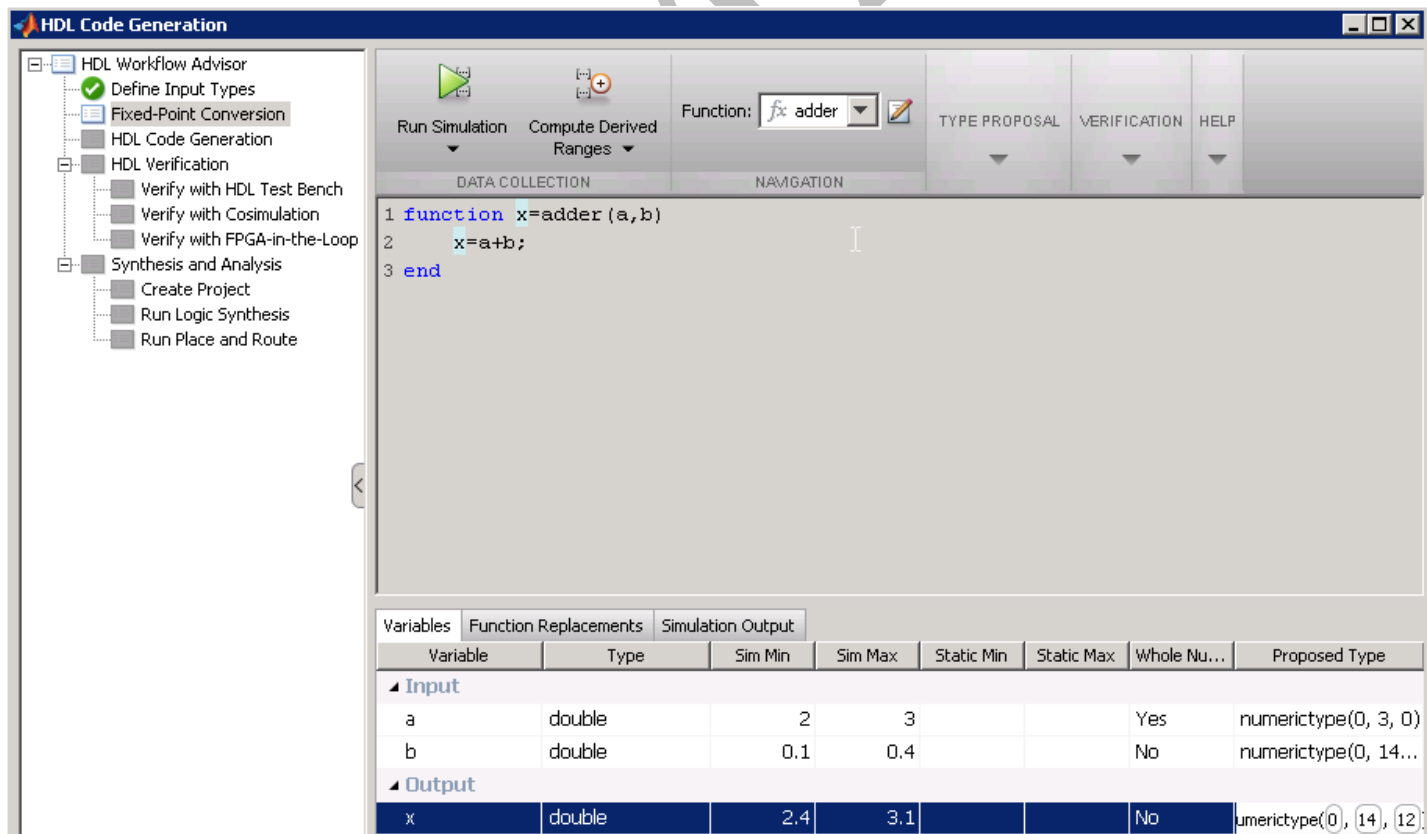
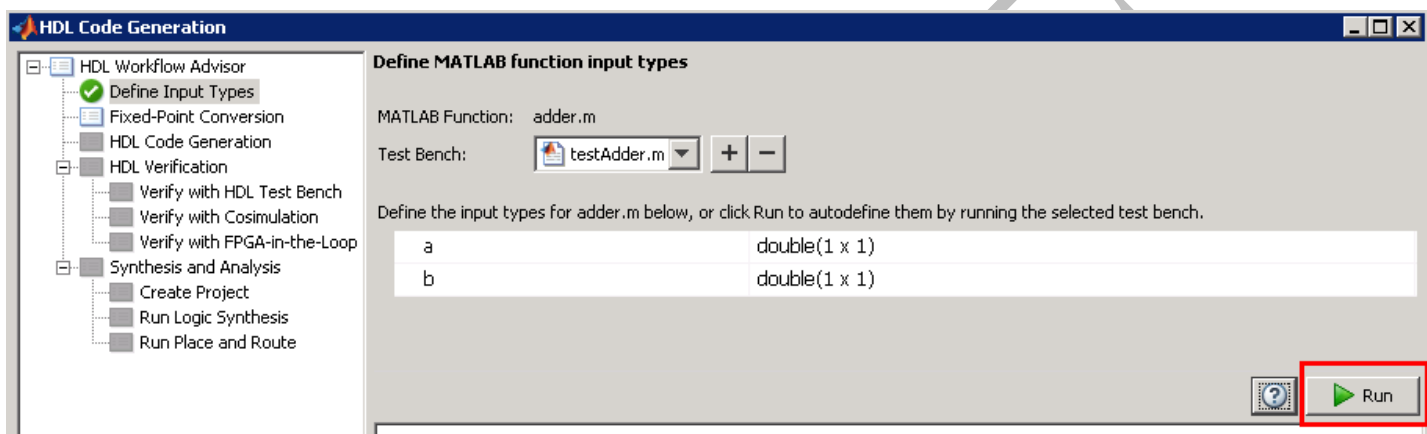
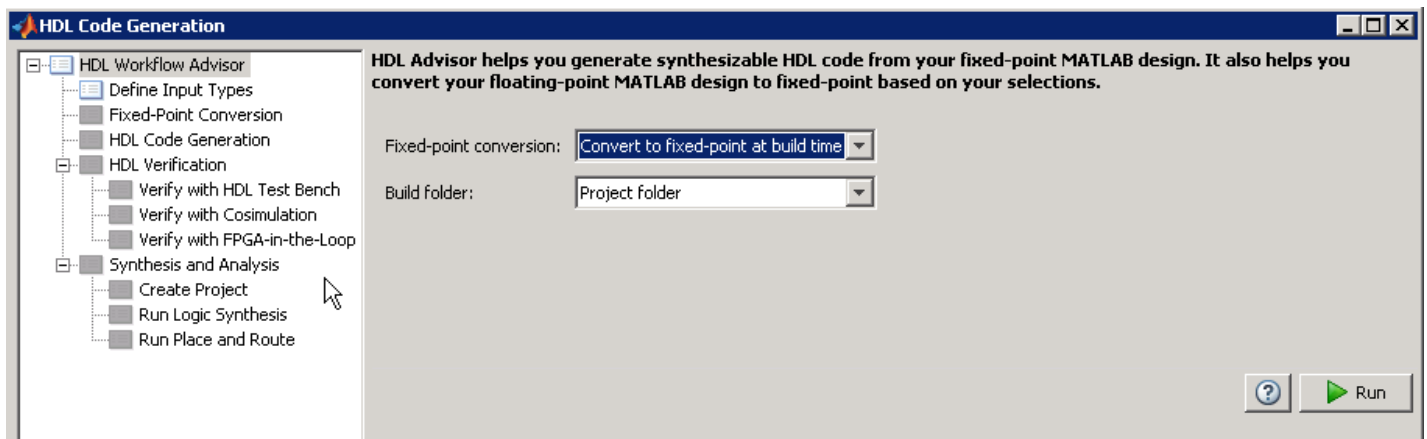




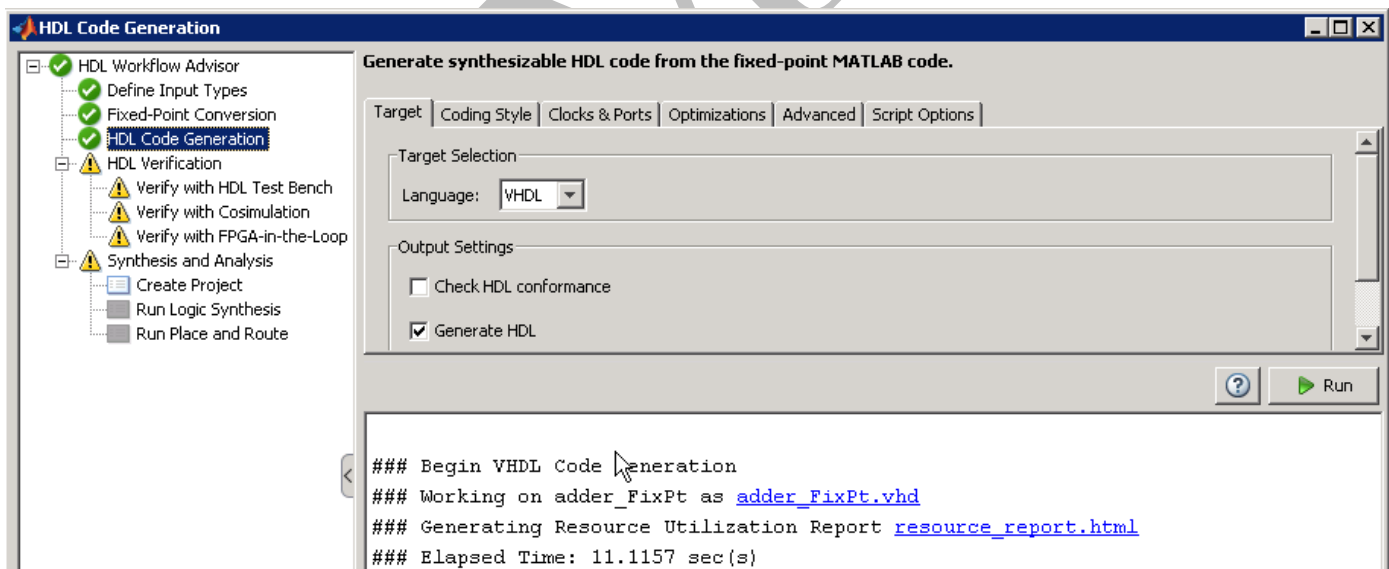
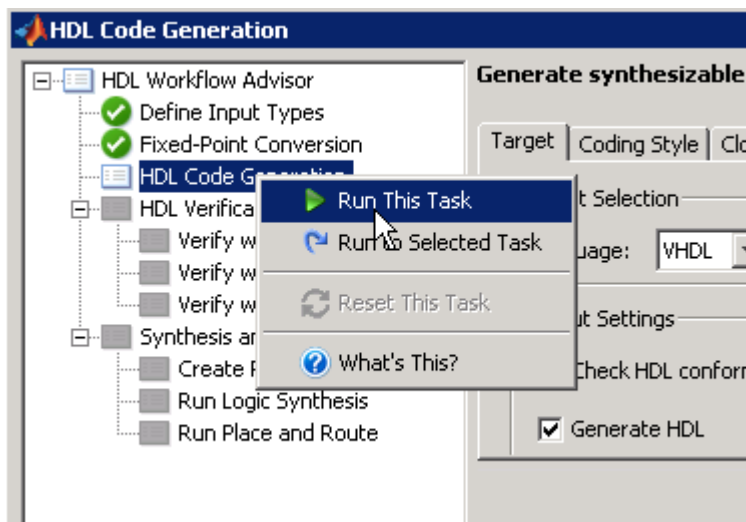
A very important thing: the active directory must have no spaces!!!







The best thing to do is to get the  at each workflow step:



Have a look to the final code by clicking on the link [adder_FixPt.vhd](#):

```

C:\Documents and Settings\Administrateur\Mes documents\MATLAB\codegen\adder\hdlsrc\adder_FixPt.vhd
EDITOR VIEW
36 -----
37 LIBRARY IEEE;
38 USE IEEE.std_logic_1164.ALL;
39 USE IEEE.numeric_std.ALL;
40
41 ENTITY adder_FixPt IS
42     PORT( clk
43           : IN    std_logic;
44           reset
45           : IN    std_logic;
46           clk_enable
47           : IN    std_logic;
48           a
49           : IN    std_logic_vector(2 DOWNTO 0); -- ufix3
50           b
51           : IN    std_logic_vector(13 DOWNTO 0); -- ufix14_En15
52           ce_out
53           : OUT   std_logic;
54           x
55           : OUT   std_logic_vector(13 DOWNTO 0) -- ufix14_En12
56     );
57 END adder_FixPt;
58
59 ARCHITECTURE rtl OF adder_FixPt IS
60
61     -- Signals
62     SIGNAL enb
63         : std_logic;
64     SIGNAL a_unsigned
65         : unsigned(2 DOWNTO 0); -- ufix3
66     SIGNAL b_unsigned
67         : unsigned(13 DOWNTO 0); -- ufix14_En15
68     SIGNAL x_tmp
69         : unsigned(13 DOWNTO 0); -- ufix14_En12
70     SIGNAL adder_FixPt_add_cast
71         : unsigned(18 DOWNTO 0); -- ufix19_En15
72     SIGNAL adder_FixPt_add_cast_1
73         : unsigned(18 DOWNTO 0); -- ufix19_En15
74     SIGNAL adder_FixPt_add_cast_2
75         : unsigned(18 DOWNTO 0); -- ufix19_En15

```