Introduction to MATLAB

— Macroeconomics —

Vivaldo Mendes

Dep. Economics — Instituto Universitário de Lisboa

September 2017

Summary

I – Introduction

What is Matlab?

1. The name tells all: **Mat**rix**Lab**oratory
2. The package of numerical computation more powerful currently available
3. Also good for symbolic calculus (Symbolic Toolbox), but there are better ones here
4. A wonderful interface with Windows (or Mac): very friendly
5. Very easy to program simple routines
6. There are routines publicly available in the net ... for almost everything ... but

Basic information

1. Matlab – basic bibliographic references:
    1. **An introduction to Matlab**, by David Griffiths (see "Readings")
    2. A Practical Introduction to Matlab, by Mark S. Gockenbach (see "Readings")
    3. **Matlab & Simulink: A Tutorial**, by Tom Nguyen (http://edu.levitas.net/Tutorials/Matlab/index.html)
    4. **Help** of Matlab: very, very useful
2. In this course:
    1. We do not wish students to be able to write down sophisticated routines
    2. Just to use Matlab for simple things
    3. "m-files" are given to students
    4. Or "m-files" found in the net

Windows in MATLAB: 6 windows

1. **Command window**: to run the programs and to obtain the results
2. **Current directory**: to open "m-files"
3. **Workspace**: where results from the simulations are saved
4. **Command history**: keep the registry of the last commands used
5. **"m_files" window** – there are two types of m_files: **function** and **script**
    1. more over this point next slide
6. **Fig_files window:**
    1. where the graphic output is displayed
    2. where we can edit the graphic output
    3. where the graphic output can be exported to another program (Word, Power Point, SWP, etc.)

Functions vs Scripts

"m_files" window: we saw that there are 2 types:

1. **function**: this type of routine can only be run in the **"command window"**
2. **script:** can be run directly from the `run` of the "m_file":
   1. choose: `debug + run`
3. only when an "`m-file`" is open, we know whether is `script` or a `function`
   1. **function**: it comes out with a text in blue indicating`function`, for example, the function: `function f=par(x);f=x.^2`
   2. a **script**: no blue text indicating `function`

Some basic commands

1. >> **clear** % erases all variables from the memory of the program
2. >> **clc** % *clears the command window, but leaves the* variables in its memory
3. >> **Ctrl+C** % ends the run of a routine
4. >> **help** `linspace` % helps about the function `linspace`
5. >> **exit** **%** shuts Matlab

Numbers:types

1. Matlab recognizes various types of numbers
2. Examples:

Type	Examples
Integer	$1362, -217897$
Real	$1.234, -10.76$
Complex	$3.21 - 4.3i \quad (i = \sqrt{-1})$
Inf	Infinity (result of dividing by 0)
NaN	Not a Number, 0/0

Numbers: formatting

1. Matlab can present numbers in various different formats
2. **default format**: assumed as "normal" by default
3. **Other formatting**:
   1. >> format short % number with 5 digits (1.1234)
   2. >> format long % number with *15 digits*
   3. >> format short e % *number in floating format with 5 digits*
   4. >> format long e % *number in floating format with 15 digits*
4. Examples: next slide

Numbers: formatting examples

Command	Example of Output
>>format short	31.4162(4–decimal places)
>>format short e	3.1416e+01
>>format long e	3.141592653589793e+01
>>format short	31.4162(4–decimal places)
>>format bank	31.42(2–decimal places)

Note that for numbers very small or very large, we may use the notation "e", for example

$$\text{-1.3412e+03} = -1.3412 \times 10^{3} = -1341.2$$
$$\text{-1.3412e-01} = -1.3412 \times 10^{-1} = -0.13412$$

II – Functions, operations and vectors

Functions

1. **Trigonometric functions:**
   `sin, cos, asin, acos, tan, atan, cot, acot`
2. **Other functions**:
   1. **exp** (exponential)
   2. **log** (neperian logarithm)
   3. **log10** (base 10)
   4. **sqrt** (square root)
   5. **abs** (absolute value)
   6. **inf** (infinit)
   7. **erf** (err function)

Operations

1. `+, -, *, ^, /` for numbers and matrices
2. `.+, .-, .*, .^, ./,` for arrays (vectors of the same type)
3. $>$ larger, $>=$ larger or equal, $==$ equal, $\sim=$ different
4. `>> nargin % Number of input arguments`
5. `>> nargout % Number of output arguments`
6. `>> varargin % Variable input argument list`
7. `>> varargout % Variable output argument list`

Vectors (arrays)

1. >> `v=1:3` , or >> `v=[1,2,3]` % *defines the vector of integer elements 1,2,3*
   1. **output**: `v = 1 2 3`
2. >> `u=[1:0.1:3]` % *defines the vector starting with 1, with an increment of 0.1, and ending at 3*
3. >> `length(u)` % to obtain the dimension of a vector*:*
   1. **output**: `ans=21`
4. **Note 1**: if we write at the end of command **";"** (in example, u=[1:0.1:3];)
   1. output does not show the values (or the variables), but keeps them in the memory
5. **Note 2**: If nothing is added at the end of the command, keeps them in the memory and shows all the elements of the vector *(21 overall)*
   1. **output**: `Columns 1 through 7`

      1.0000 1.1000 1.2000 1.3000 1.4000 1.5000 1.6000

      `Columns 8 through 14`

Vectors (arrays) – cont.

1. >> `u=[1:4]; v=[3:6];`
2. >> `u*v` % incorrect operation
   1. **output**: "??? Error using ==> mtimes

      Inner matrix dimensions must agree."
1. >> `u .*v` % *correct operation*
   1. **output**: ans = 3 8 15 24
2. >> `v'` % *transpose vector*
3. **linspace:** the vector `x= [-1:0.1:1]`, (all the elements between -1 and 1 with a step of 0.1) can be written
   1. >> `x = linspace(-1,1,21)`

III – Matrices

Matrices

1. To represent a matrix A of type (3×3). For example

$$A = \begin{bmatrix} 1 & 0 & -3 \\ 2 & 4 & 6 \\ 2 & 0 & 9 \end{bmatrix}.$$

2. The elements in one line are separated by a common **","** or **space**
3. Lines are separated by ";"
4. The command is:

```
>> A=[1,0,-3;2,4,6;2,0,9]
```

output: A =

1	0	−3
2	4	6
2	0	9

Matrices (cont.)

1. >> A' % *transpose matrix* A^T *of* A
2. >> A(:,i) % *lists all elements of column i of matrix* A
3. >> A(i,:) % *lists all elements of line i of matrix* A
4. >> A(i,j) % specifies the element *in line i and column j*
5. >> inv(A) % *calculates, if exists, the inverse matrix of* A
6. >> rank(A) % *determines the characteristic of matrix* A
7. >> eye(n) % *lists the identity matrix of order* n
8. >> ones(mxn) % *matrix of type* $(m \times n)$ *, with all unit elements*
9. >> det(A) % *calculates the determinant of* A
10. >> trace(A) % *calculates the trace of matrix* A
11. >> eig(A) % *calculates the eigenvalues of* A
12. >> poly(A) % *characteristic polynomial of* A
13. >> norm(A) % *norm of matrix* A

Matrices (cont.)

Operations with matrices

1. `>> D = A + B;` `>> E = A * C;` `>> F = C * B`
2. `>> F1 = inv(sqrt(F)),` `>> G = B ./ A`
3. `>> sum(A)` % *sum of columns*
4. `>> sum(sum(A))` % sum of all the elements
5. `>> A1 = sort(A)` % *separates the columns*
6. `>> A2 = sort(A, 2)` % *separates the lines of a matrix*

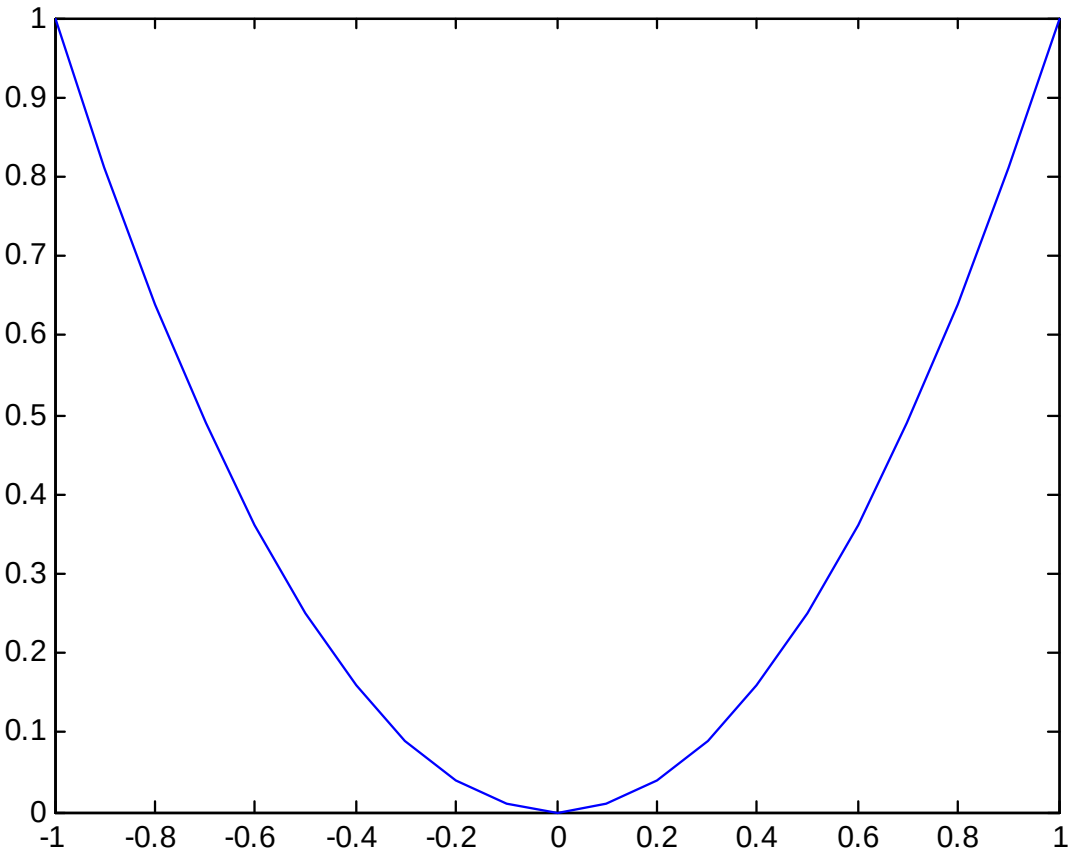
Example: determine the following (don´t type the ";")

1. Determine the inverse and the eigenvalues of A: `inv(A), eig(A)`
2. Represent the second line and the first column of A: `L2=A(2,:),C1=A(:,1)`

IV – Real functions

Real functions with one real variable

1. Representing graphically the function $f(x) = x^2$ in the interval $[-1, 1]$
   1. `>>x = -1:0.1:1; f=x.^2; plot(x,f)`



Another way

1. Suppose we have the following m-file **par.m** saved in your partition
2. This m-file represents the quadratic function above showed

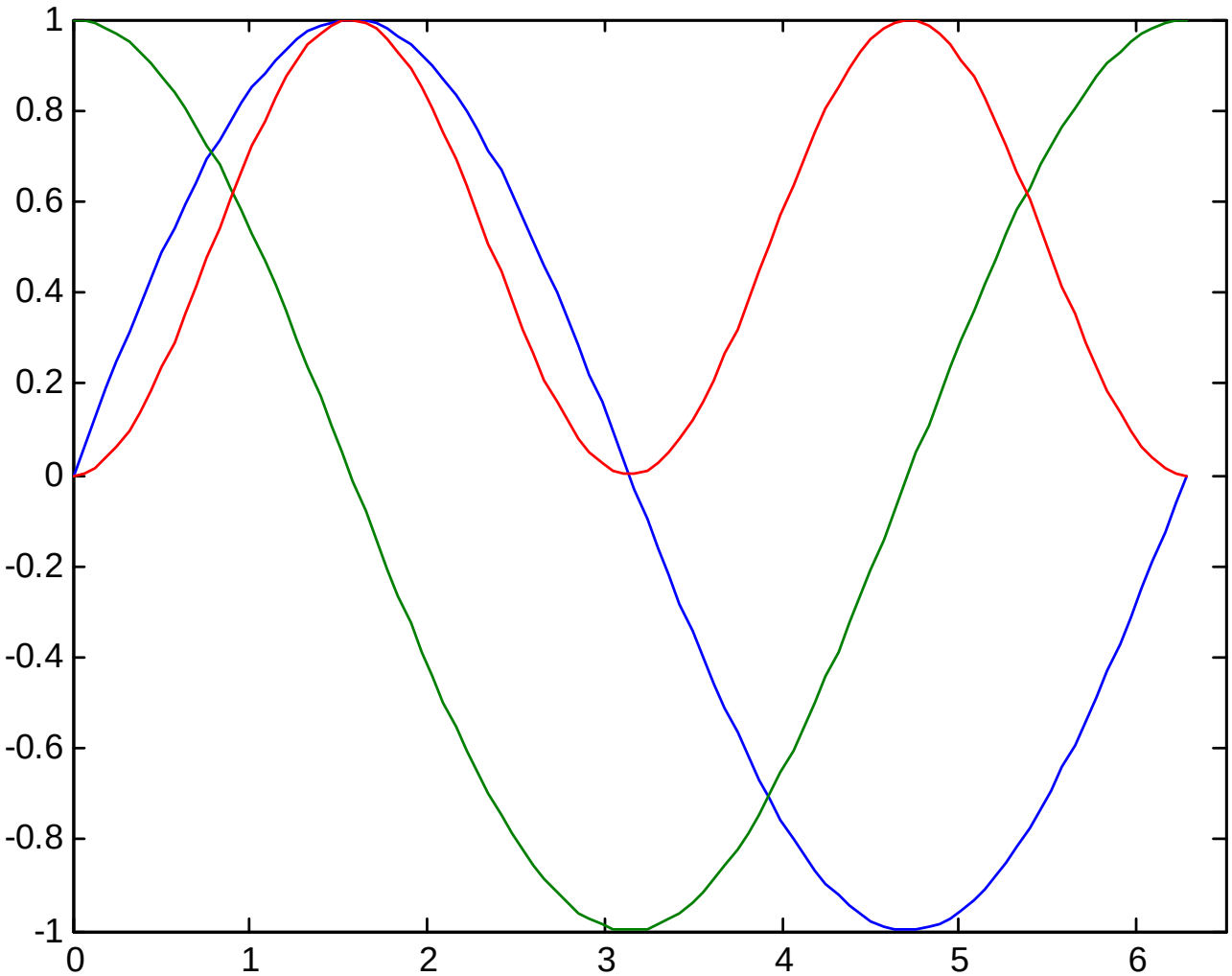
```
function f=par(x)

    f=x.^2;
```

3. Using **fplot** in the Command Window
   1. `>> fplot(@par,[-1 1])` or `fplot('par',[-1 1])`
4. We get the graphic of our parable.
5. **Exercise:** try it.

Various functions in the same coordinates

1. For example:
   1. `>> x = linspace(0,2*pi,100); y1=sin(x); y2=cos(x); y3=sin(x).*sin(x);`
   2. `>> plot(x,y1,x,y2,x,y3)`
2. It creates the graphic with the 3 trigonometric functions just mentioned
3. Note that x is the independent variable, $y1, y2, y3$ are the dependent ones
4. Therefore, the logic is like this:
   1. (independent variable, dependent variable 1, independent variable, dependent variable 1„ ....)

Various functions in the same coordinates



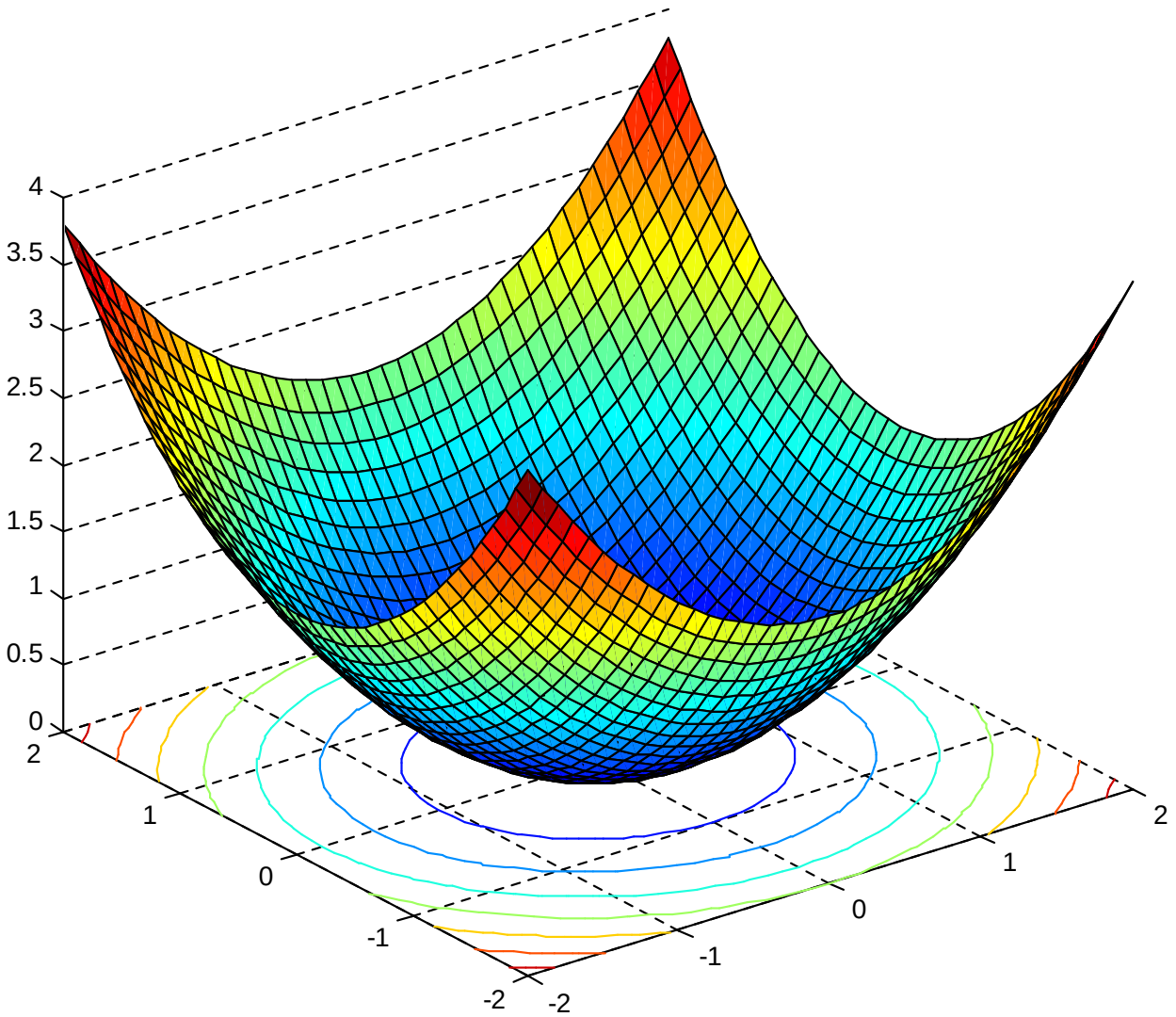
Real functions of two real variables

1. Represent graphically the function $f(x,y) = 0.5(0.9x^2 + y^2)$ in the interval $[-2,2] \times [-2,2]$.
2. `>>x=-2:0.1:2; y=-2:0.1:2; [X,Y]=meshgrid(x,y);`

   `z = 0.5.*(0.9.*X.^2+Y.^2); figure;surf(z)`

1. Another way to get the figure:
   1. Open the m-file: **BC_Objectivo.m**
   2. Choose: **debug+run**
   3. The figure comes out in another window **(vide next slide)**
2. Other commands for graphical representations: `mesh(z)`, `meshc(z)`, `surfc(z)`

Real functions of two real variables (cont.)



V – Importing data and representing time series

Representation of time series

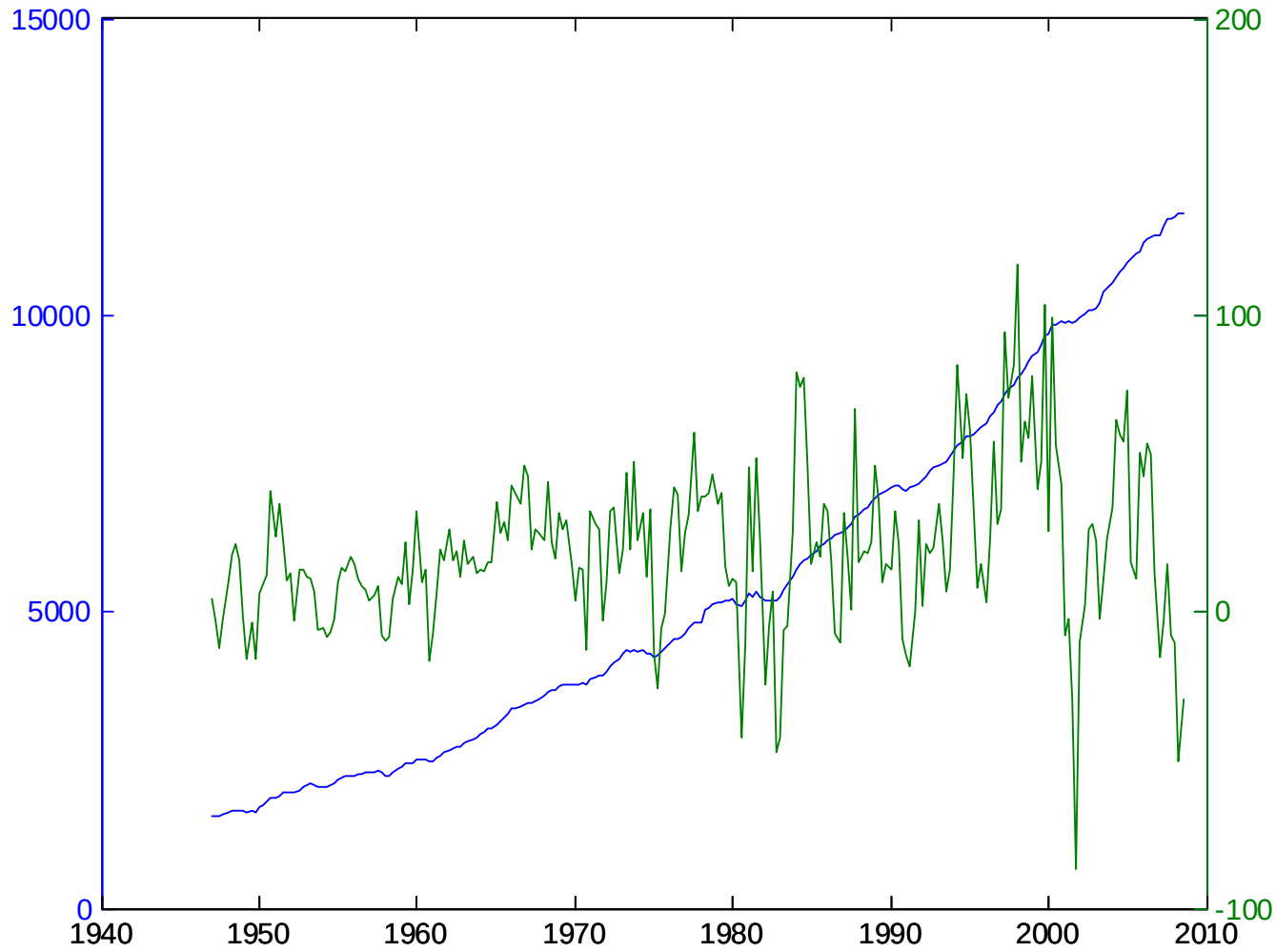
1. Two forms of "importing" data into Matlab
   1. Using the "Command Window"
   2. Using the submenu "Import data" from the main menu
2. An example using the "Command Window"
   1. `>> load temp.dat` % *importing the data from a temp file*
   2. `>> plot(temp)` % *does the figure with the data*
3. **Exemple – "Command Window"**
   1. `>> load portugal_aulas.txt`
   2. `>> figure; plot(portugal_aulas)`
   3. `>> days=linspace(1990,2008,4790);`
   4. `>> figure; plot(days,portugal_aulas)`

Representation of time series

Representing two scales in the same coordinates

1. Sometimes it is very useful to represent in the same graphic two different scales
    1. >> **plotyy(x1, y1, x2, y2)**
    2. represents a function (or time series) $y1$ of variable $x1$ in the yy axis defined on the left hand side of the graphic, and $y2$ of variable $x2$ in the yy axis defined on the right hand side
2. **Example** :
    1. `>> load USdata.txt`
    2. `>> GDP=USdata(:,3);`
    3. `>> INVENT=USdata(:,2);`
    4. `>> time=1947.0:0.25:2008.5;`
    5. `>> figure;`
    6. `>> plotyy(time,GDP,time,INVENT)`

Representing two scales in the same coordinates (cont.)



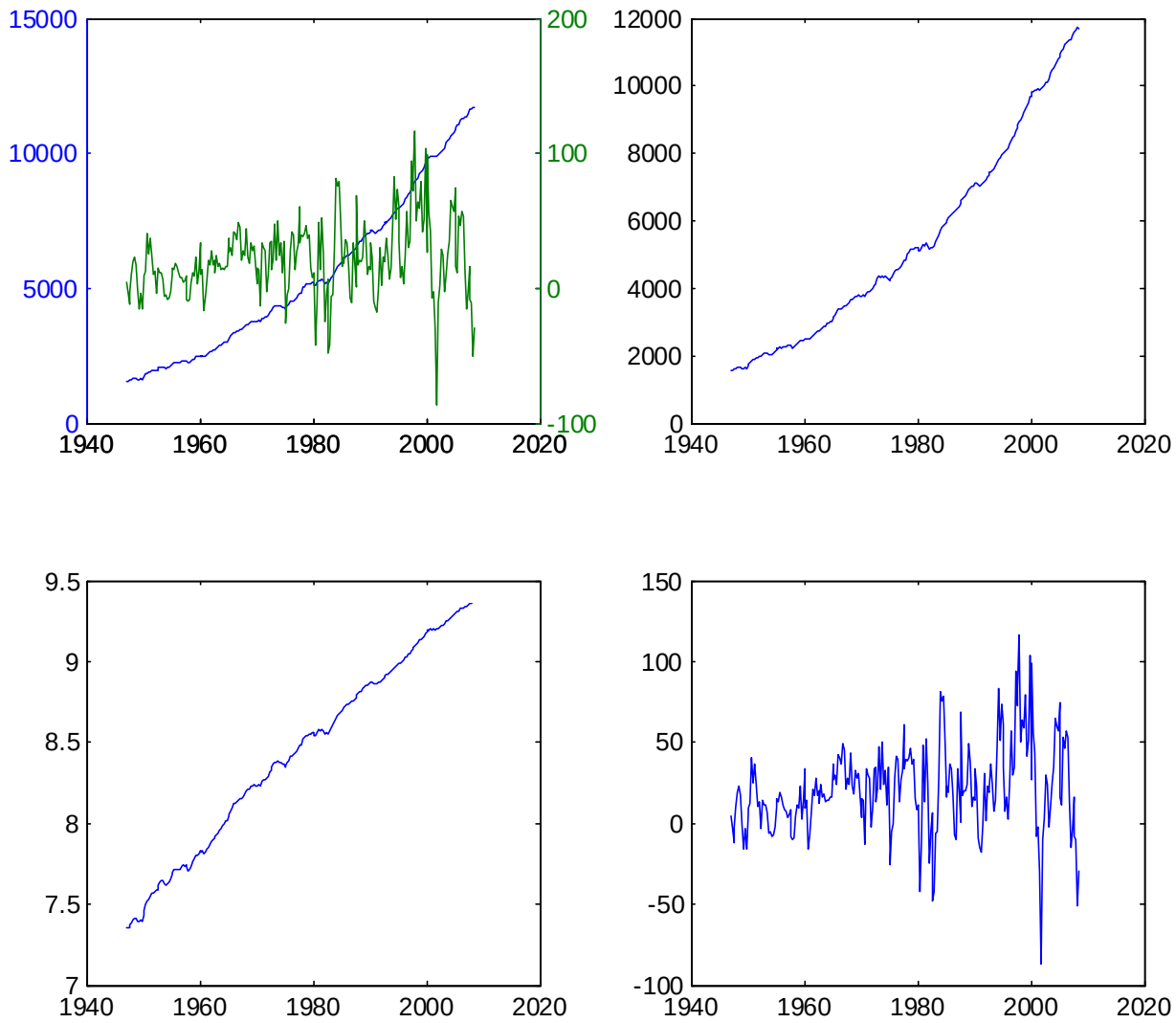
Representing various graphics in the same figure

- Another very useful thing is to put various graphics in the same figure
- A routine that puts 4 panels (graphics) inside 1 figure is:

```
>>load USdata.txt;
GDP=USdata(:,3);INVENT=USdata(:,2);time=1947.0:0.25:2008.5;
figure
subplot(221);plotyy(time,GDP,time,INVENT);
subplot(222);plot(time,GDP);
subplot(223);plot(time,log(GDP));
subplot(224);plot(time,INVENT);
```

- See result in the next slide

Representing various graphics in one figure (cont.)



VI – Exercises

The sustainability of public debt

1. The evolution of nominal public debt (D_t) is given by

$$P_t G_t + i_t \cdot D_{t-1} = P_t T_t + \underbrace{NewDebt}_{D_t - D_{t-1}}$$

2. P = general price level, T = real taxes, G = real government expenditures, i = nominal interest rate on public debt
3. Designating $P_t (G_t - T_t) = F_t$ as the **primary deficit,** we get

$$D_t - D_{t-1} = F_t + i_t \cdot D_{t-1}$$

4. After a lengthy derivation, the public debt as a percentage of GDP (d_t) is given by

$$d_t = f_t + \left(\frac{1 + r_t}{1 + g_t} \right) d_{t-1}$$

5. where f_t = primary deficit as a % of GDP, r_t = real interest rate, g_t = growth rate of real GDP

The sustainability of public debt (cont.)

1. Using (see *PubliDebt_Simulations.m*) Matlab and Excel, analyze the sustainability of Public Debt in two scenarios:

$$\begin{aligned} \textit{Scenario A}: \quad & f = 0.02 \quad r = 0.02 \quad g = 0.03 \\ \textit{Scenario B}: \quad & f = 0.01 \quad r = 0.02 \quad g = 0.03 \end{aligned}$$

2. Try to see what happens in different snaps of time:
   1. 25 years period
   2. 50 years period
   3. 150 years
   4. 600 years
3. What happens if, between $t = 600, t = 610$, there is a shock to primary deficits in both scenarios
   1. $f = 0.08$ at $t = [600, 610]$, and goes back again to normality after $t = 610$
4. What do you conclude?

Various cases of dynamic processes

1. Linear difference equation (see *Stable_Deterministic_Equilibrum.m*)

$$y_t = a_0 + a_1 y_{t-1}$$

2. The logistic function (*see Logistic.m*)

$$y_t = a_1 (1 - y_{t-1}) y_{t-1}$$

3. A funny equation (strong nonlinearity: *Self_fulfilling_Prophecies.m*)

$$y_t = \frac{a_1 y_{t-1}^3 - a_2 y_{t-1}^2}{2}$$

4. An AR(1) process (see *Stochastic_Equilibrum.m*)

$$y_t = a_0 + a_1 y_{t-1} + \varepsilon_t \quad , \quad 0 < a_1 \leqslant 1$$

5. An AR(1) process with a deterministic trend (*Deterministic-trend.m*)

$$y_t = a_0 + a_1 y_{t-1} + a_2 t + \varepsilon_t \quad , \quad 0 < a_1 \leqslant 1$$

a_0, a_1, a_2 as parameters, $\varepsilon_t \sim iid(0, \sigma^2)$, t is the long term trend of y (measured in logarithmic values)

Bibliography

Bibliography

- David F. Griffiths (2005). "An Introduction to Matlab (Version 2.3)", Department of Mathematics, The University Dundee. (36 pages, to be just consulted, not to be studied)
- Mark Gockenbach (1999). "A Practical Introduction to Matlab", Department of Mathematical Science, Michigan Technological University. (33 pages, to be just consulted, not to be studied)
- William Palm III (2008). A Concise Introduction to MATLAB, *McGraw-Hill*, N.Y. As a book, you have to dive into several chapters to cover most material presented in these slides. But the two firts chapters will be enough.

Bibliography (cont.)

Cleve Moler (2004). Numerical Computing with MATLAB, *MathWorks*, the entire book available at http://www.mathworks.com/moler/chapters.html. A good introduction to Matlab (in fact, more like an appetizer, rather than true introduction to the computing package) can be found in Chapter 1 "Introduction to Matlab" (55 pages) of this book written by the founder and CEO of Matlab.