# Introduction to MATLAB

## — Modern Macroeconomics —

Vivaldo Mendes

Dep. Economics — Instituto Universitário de Lisboa

October 2018

# Summary

# I – Introducing the package

# 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

# Why Matlab?

1. Excel is very slow and very bad for ... numerical computation
2. Excel can not deal with symbolic computation
3. Excel is terrible because it easily allows tremendous mistakes
4. Example: the London Whale case

## The "London Whale" trade

Due to an error in an Excel spreadsheet used to model risk, JP Morgan seriously underestimated the downside of its synthetic credit portfolio, which ultimately led to the bank to declare $6 billion in losses and could lead to another $600 million in fines. As James Kwak explains on Baseline Scenario, the errors stemmed from a combination of copy-paste mistakes and a faulty equation created to crunch the numbers.

# Basic information

1. Mark S. Gockenbach (???). A Practical Introduction to Matlab, Michigan State University.
2. David Griffiths (2015). An introduction to Matlab, The University of Dundee
3. David Houcque (2005). Introduction to Matlab for Engineering Students, Northwestern University.
4. Krister Ahlersten (2012). An Introduction to Matlab, Bookboon.com
5. Craig Warren (2012). An interactive introduction to MATLAB, University of Edinburgh
6. **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, or 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: `run`
3. It is easy to know whether you have a script or a function by inspecting the current folder and also
   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 and symbols

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
6. "," Separates commands on the same line
7. ";" Suppresses output in the command window when commands are executed
8. "%" Tell Matlab that what comes after this symbol is just comments, not code

# II – Basic operations

# Basic arithmetic operators

| Symbol | Operation | Example |
|---|---|---|
| $+$ | Addition | $2+3$ |
| $-$ | Subtraction | $2-3$ |
| $*$ | Multiplication | $2*3$ |
| $/$ | Division | $2/3$ |

# The precedence of arithmetic operations

| Precedence | Mathematical operations |
|---|---|
| First | The contents of all parentheses are evaluated first, starting from the innermost parentheses and working outward. |
| Second | All exponentials are evaluated, working from left to right |
| Third | All multiplications and divisions are evaluated, working from left to right |
| Fourth | All additions and subtractions are evaluated, starting from left to right |

# Relational and logical operators

| Character | Description |
|---|---|
| $<$ | Less than |
| $\leq$ | Less than or equal to |
| $>$ | Greater than |
| $\geq$ | Greater than or equal to |
| $==$ | Equal to |
| $\sim=$ | Not equal to |
| & | Logical or element-wise AND |
| \| | Logical or element-wise OR |
| && | Short-circuit AND |
| \|\| | Short-circuit OR |

# Basic mathematical and logical operations: examples

- Just use Matlab as a simple calculator

```
>> 10/20/10
ans =
0.0500
```

- Give names to variables

```
>>abc=10/20/10
abc =
0.0500
```

- Apply logical statements

```
>> abc<5
ans =
logical
1
```

```
>> z=abc*100
z =
5
>> z==10
ans =
logical
0
```

# Basic mathematical and logical operations: examples

- Consider the case

$$(1+2)\times 3$$

```
>> (1+2)*3
ans =
9
```

A wrong answer if you forget parenthesis

```
>>1+2*3
ans =
7
```

- Consider the case

$$\frac{1}{2+3^2}+\frac{4}{5}\times\frac{6}{7}$$

```
>> 1/(2+3^2)+4/5*6/7
ans =
0.7766
```

if parentheses are missing,

```
>> 1/2+3^2+4/5*6/7
ans =
10.1857
```

# III – Matrices

# Matrices

1. To represent a matrix $A$ of type $(3 \times 3)$. 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 comma **","** or **space**
3. Lines are separated by ";"
4. The command is:

```
>> A=[1,0,-3;2,4,6;2,0,9]
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(A)),` `>> 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 – Matrix vs Arrays arithmetic operations

# Matrix vs Arrays operations

1. MATLAB has two different types of arithmetic operations:
   1. matrix arithmetic operations
   2. array arithmetic operations
2. Matlab follows the rules of algebra ... so no problem, just apply them using the normal symbols of basic operators above
3. Array operations for short, are done element-by-element ... according to your needs

| Operation | Matrix | Array |
|---|---|---|
| Addition | + | + |
| Subtraction | − | − |
| Multiplication | * | .* |
| Division | / | ./ |
| Left division | \ | .\ |
| Exponentiation | ^ | .^ |

# Matrix vs Arrays operations: one example

1. Consider matrices $A$ and $B$

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \quad ; \quad B = \begin{bmatrix} 10 & 20 & 30 \\ 40 & 50 & 60 \\ 70 & 80 & 90 \end{bmatrix}$$

2. Using the rules of algebra we get

$$C = A * B = \begin{bmatrix} 300 & 360 & 420 \\ 660 & 810 & 960 \\ 1020 & 1260 & 1500 \end{bmatrix}$$

3. Using Arrays operations

$$D = A. * B = \begin{bmatrix} 10 & 40 & 90 \\ 160 & 250 & 360 \\ 490 & 640 & 810 \end{bmatrix}$$

# Matrix vs Arrays operations: another example.

- Consider vectors $u$ and $v$

$$u = \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix} \quad ; \qquad v = \begin{bmatrix} 5 & 6 & 7 & 8 \end{bmatrix}$$

```
>> u=[1:4]; v=[5:8];
```

- Try multiply them using simple operators: **incorrect operation**

```
>> u*v
Error using *
Incorrect dimensions for matrix multiplication.  Check that
the number of columns in the first matrix matches the
number of rows in the second matrix.  To perform
elementwise multiplication, use '.*'
```

- Apply the operator for array multiplication ".*"

```
>> u.*v
ans =
5     12     21     32
```

# Linspace: a very useful way of writing down an array or a vector

`linspace(2,9,5)` Linear spacing

This creates a matrix with only one row; a row vector. The first element is 2, the last element is 9, and there are 5 linearly spaced elements (i.e., the distance between each is the same).

- The vector `x= [-1:0.1:1]`, (all the elements between $-1$ and 1 with a step of 0.1) can be written

```
>> x = linspace(-1,1,21);
>> length(x)
ans =
 21
```

- What happens if we do not type ";" at the end of the command line?

```
x =
```

# V – Real functions

# Functions

| | | | |
|---|---|---|---|
| `cos(x)` | Cosine | `abs(x)` | Absolute value |
| `sin(x)` | Sine | `sign(x)` | Signum function |
| `tan(x)` | Tangent | `max(x)` | Maximum value |
| `acos(x)` | Arc cosine | `min(x)` | Minimum value |
| `asin(x)` | Arc sine | `ceil(x)` | Round towards $+\infty$ |
| `atan(x)` | Arc tangent | `floor(x)` | Round towards $-\infty$ |
| `exp(x)` | Exponential | `round(x)` | Round to nearest integer |
| `sqrt(x)` | Square root | `rem(x)` | Remainder after division |
| `log(x)` | Natural logarithm | `angle(x)` | Phase angle |
| `log10(x)` | Common logarithm | `conj(x)` | Complex conjugate |

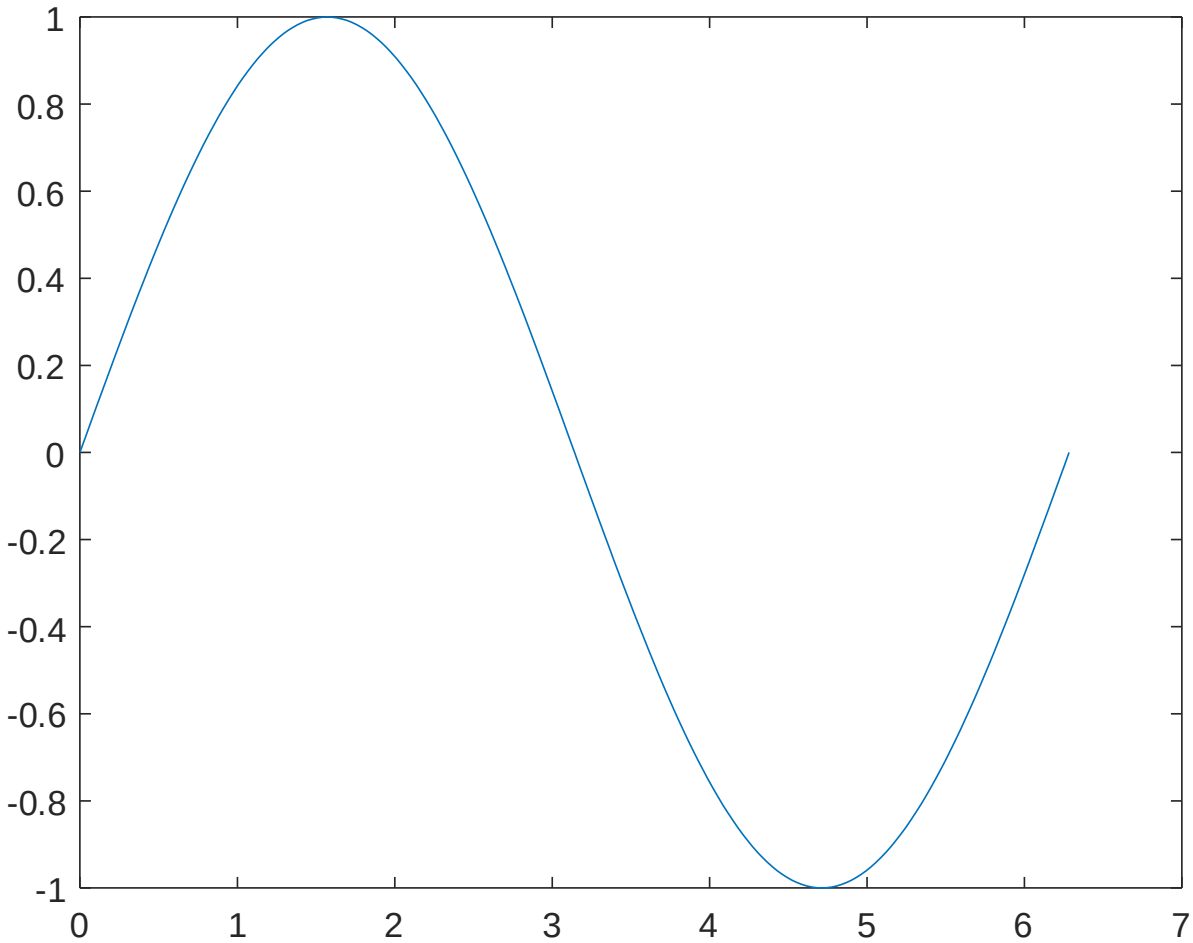
# A simple example

- Plot the function $sin(x)$ on the interval $[0, 2\pi]$,
  - First create a vector of values from $0$ to $2\pi$
  - then compute the sine of these values
  - and finally plot the result
- The code is:

```
>> x = 0:pi/100:2*pi; y = sin(x); plot(x,y)
```

- Note that "`0:pi/100:2*pi`" yields a vector that
  - starts at $0$,
  - takes steps (or increments) of $\pi/100$,
  - stops when $2\pi$ is reached.
- See figure window next slide

# A simple example: our figure



# Specifying line styles and colors

- It is possible to specify line **styles**, **colors**, and **markers** using the plot command

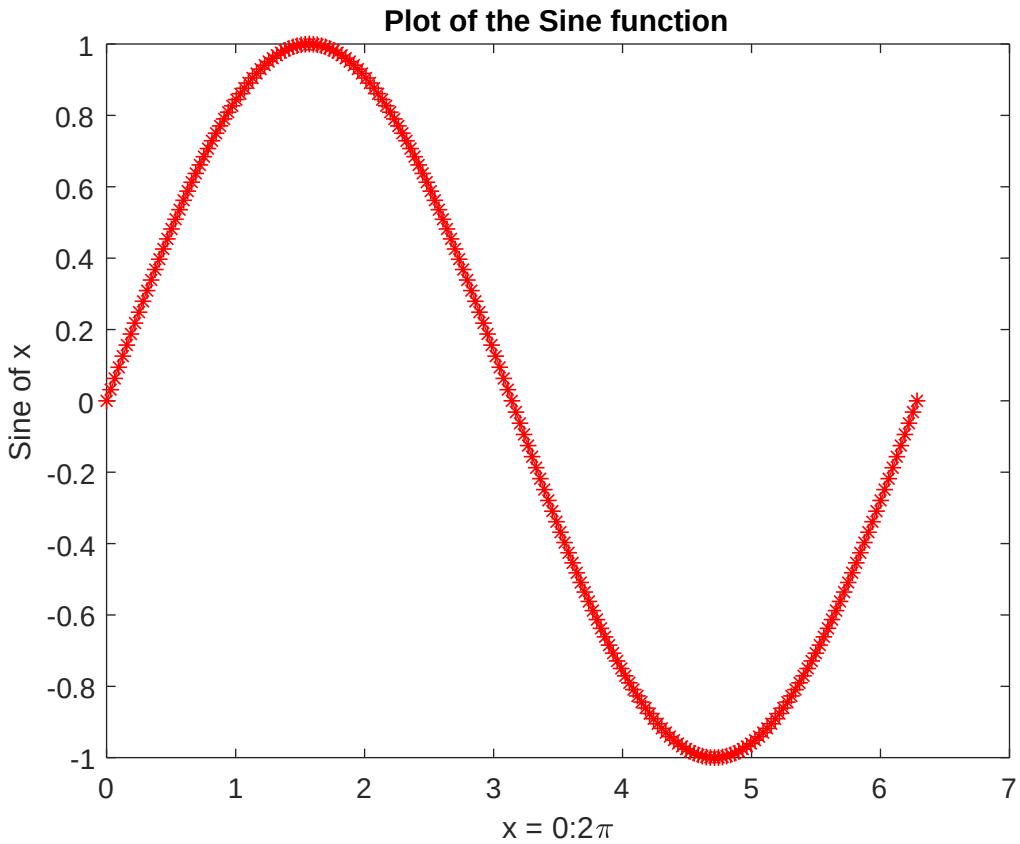
```
plot(x,y,'style_color_marker')
```

| Symbol | Color | Symbol | Line Style | Symbol | Marker |
|---|---|---|---|---|---|
| k | Black | − | Solid | + | Plus sign |
| r | Red | −− | Dashed | o | Circle |
| b | Blue | : | Dotted | * | Asterisk |
| g | Green | −. | Dash-dot | . | Point |
| c | Cyan | none | No line | × | Cross |
| m | Magenta | | | s | Square |
| y | Yellow | | | d | Diamond |

# A simple example (cont)

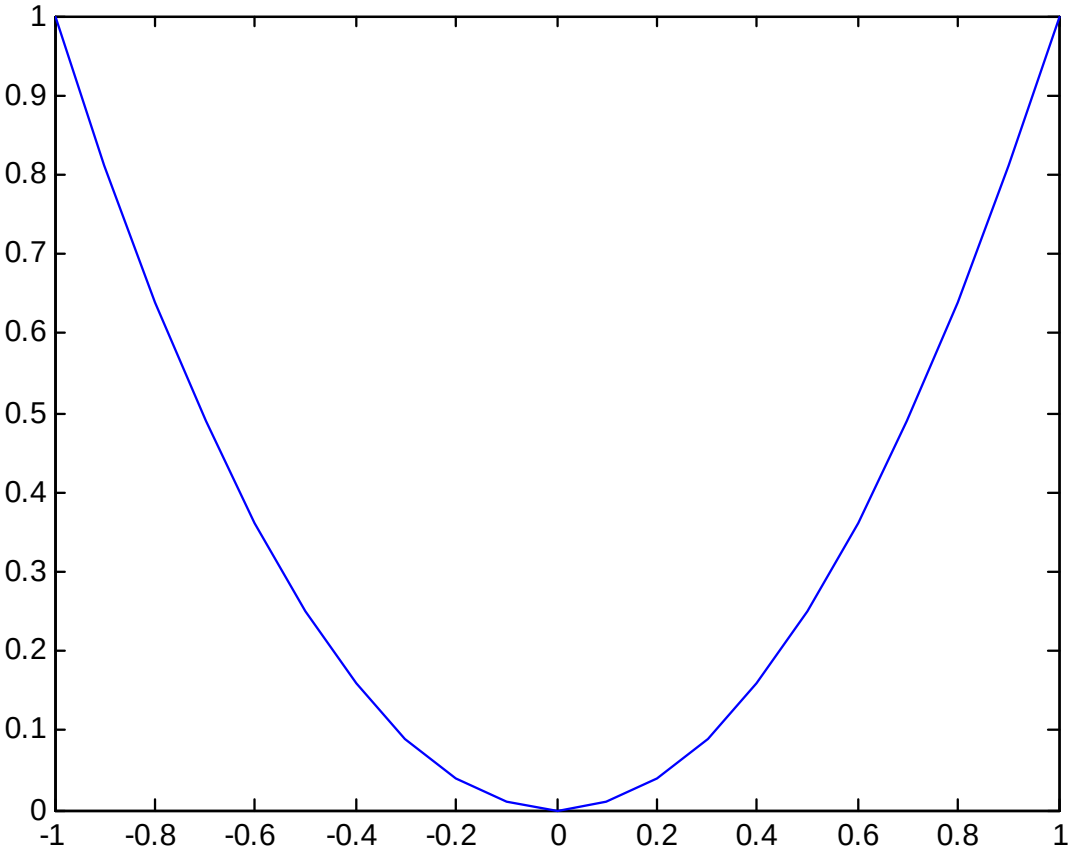
- Now label the axes, add a title, and choose the color of the line and its marker

```
>> plot(x,y,'-r*');xlabel('x = 0:2\pi'); ylabel('Sine of x'); title('Plot of the Sine function')
```



# Another example

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: with a function

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

```
>> fplot(@par,[-1 1]) or fplot('par',[-1 1])
```

- We get the graphic of our parable.
- We keep the memory of Matlab completely free: no entry in the Wokspace

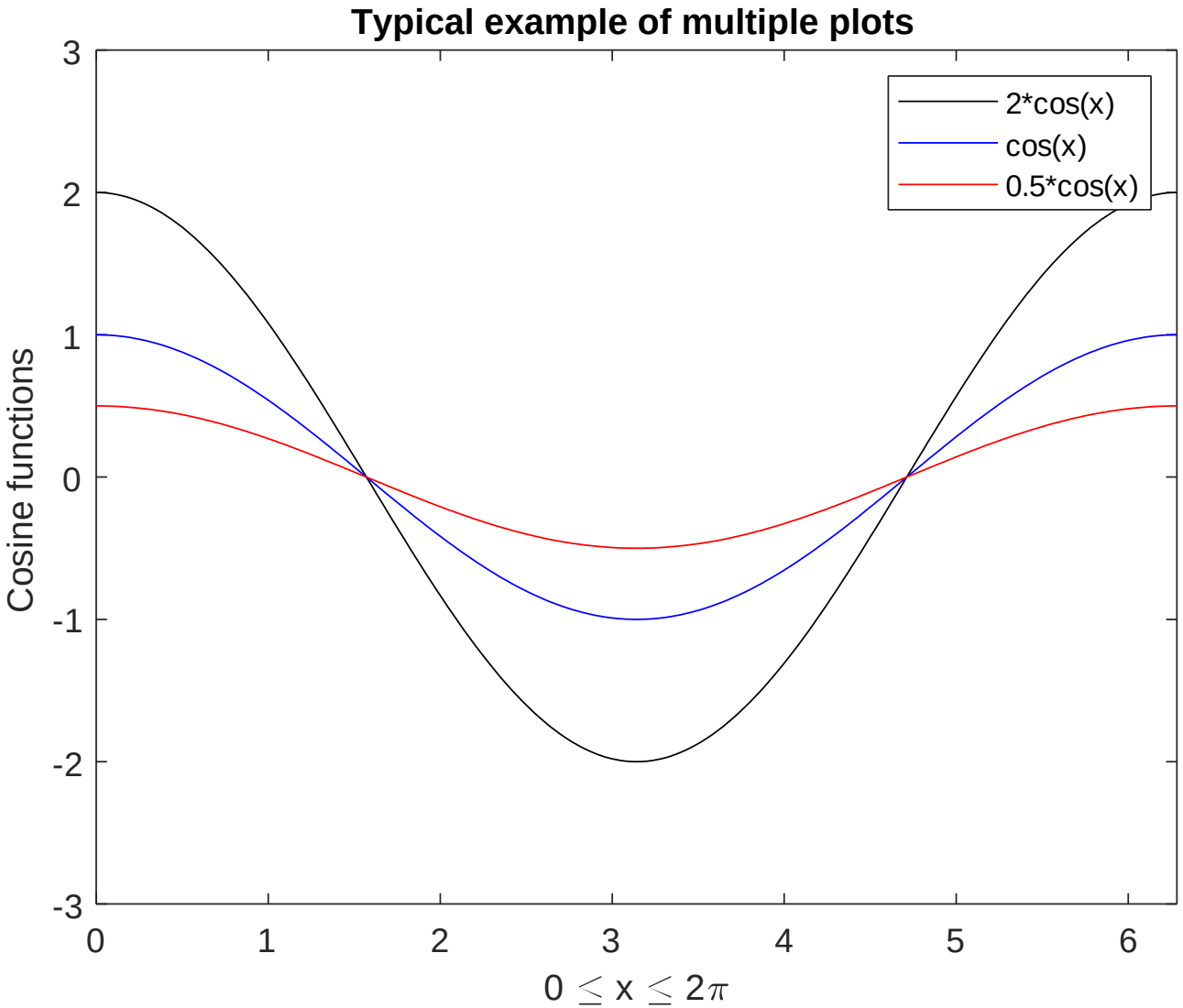
# Various functions in the same coordinates: an example

```
>> x = 0:pi/100:2*pi;
>> y1 = 2*cos(x);
>> y2 = cos(x);
>> y3 = 0.5*cos(x);
>> plot(x,y1,'--',x,y2,'-',x,y3,':')
>> xlabel('0 \leq x \leq 2\pi')
>> ylabel('Cosine functions')
>> legend('2*cos(x)','cos(x)','0.5*cos(x)')
>> title('Typical example of multiple plots')
>> axis([0 2*pi -3 3])
```

1. Note that $x$ is the independent variable, $y1, y2, y3$ are the dependent ones
2. Therefore, the logic is like this:

$$(x, y1, x, y2, x, y3, ....)$$

# Various functions in the same coordinates



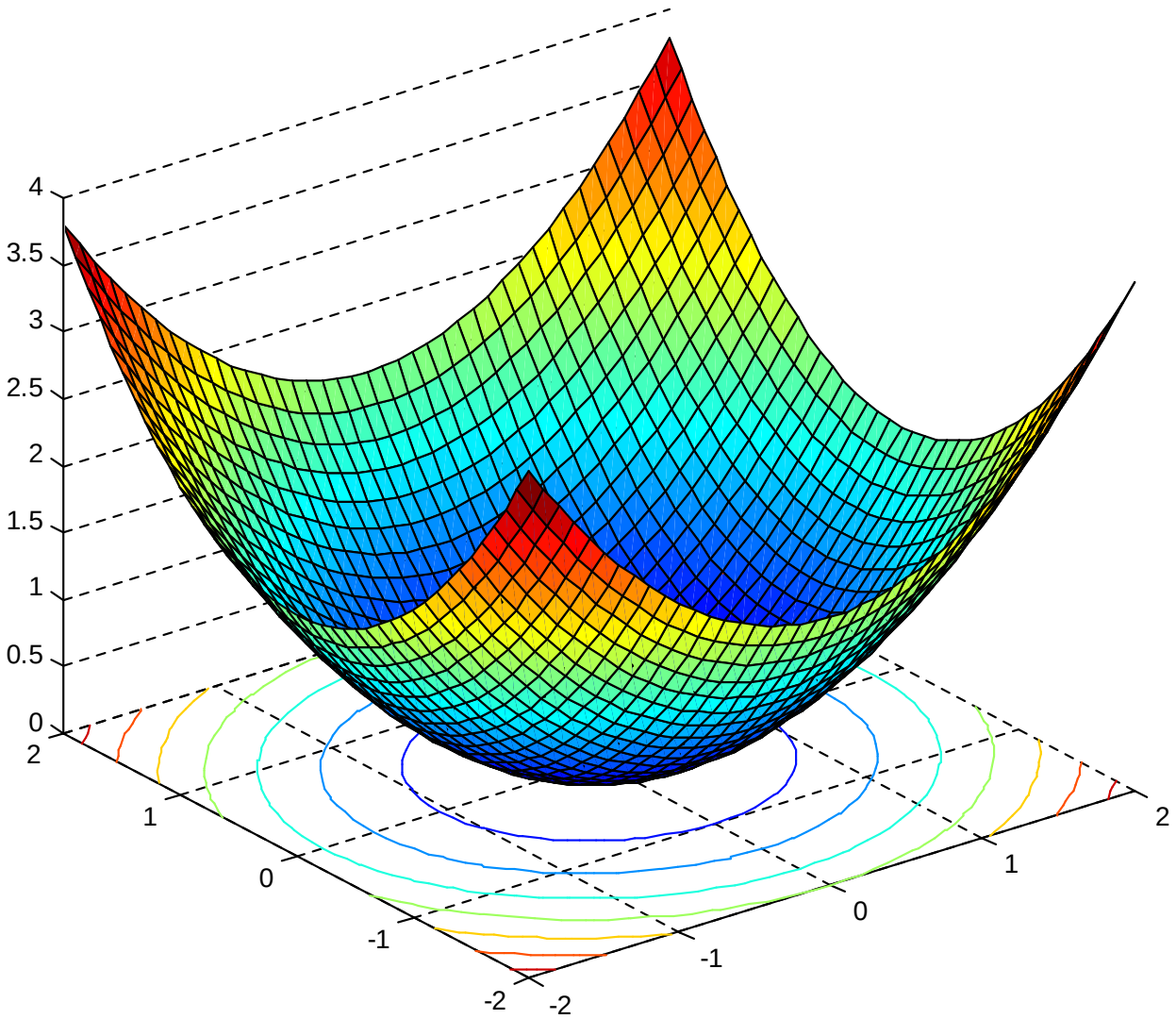
# Real functions of two real variables

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: **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.)



# VI – Loops and conditional statements

# What are loops?

1. Loops provide methods for **repeatedly executing commands**.
2. You might want to repeat the same commands, changing the value of a variable each time, **for a fixed number of iterations**
3. Alternatively, you might want to repeat the same commands, changing the value of a variable each time, **continually until a certain condition is reached**.
4. Two of the most common types of loops, "**for**" and "**while**"
5. We will exemplify the "**for**" loop.
6. They are very useful for economics because of ... dynamics and equilibrium

# The "for...end" loop

- Consider two independent stochastic processes $(h(t)$ and $g(t))$
- A Loop for (i) each regime separately; (ii) a T_AR(1)

```
n=200; % number of iterations
for t=1:n
    h(t+1)=a0+a1*h(t)+wn(t);
    g(t+1)=a2+a3*g(t)+wn(t);
end
```

```
for t=1:n
    if y(t)<c
        y(t+1)=a0+a1*y(t)+wn(t);
    else
      y(t+1)=a2+a3*y(t)+wn(t);
    end

end
```

- See the **T_Var.m** example where we find the values of the

# VII – Importing data and representing time series

# Representation of time series

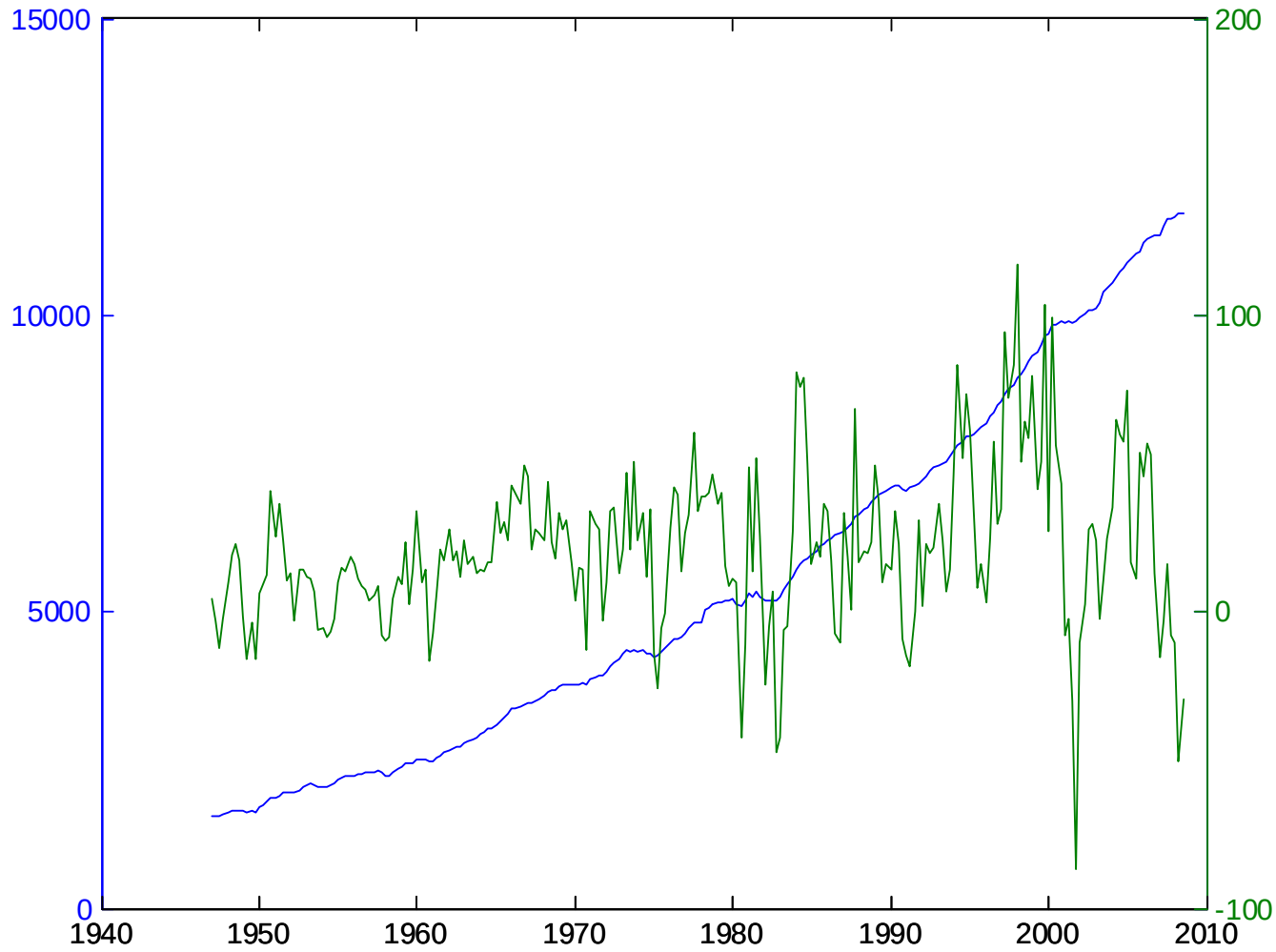
1. Two forms of "importing" data into Matlab
   1. Using the "Command Window"
   2. Using the submenu "Import data" from the main menu
2. An example using the "Command Window"
   1. `>> load temp.dat` % *importing the data from a temp file*
   2. `>> plot(temp)` % *does the figure with the data*
3. **Exemple – "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=1947.0:0.25:2008.05;`
    5. `>> figure;`
    6. `>> plotyy(time,GDP,time,INVENT)`

# Representing two scales in the same coordinates (cont.)



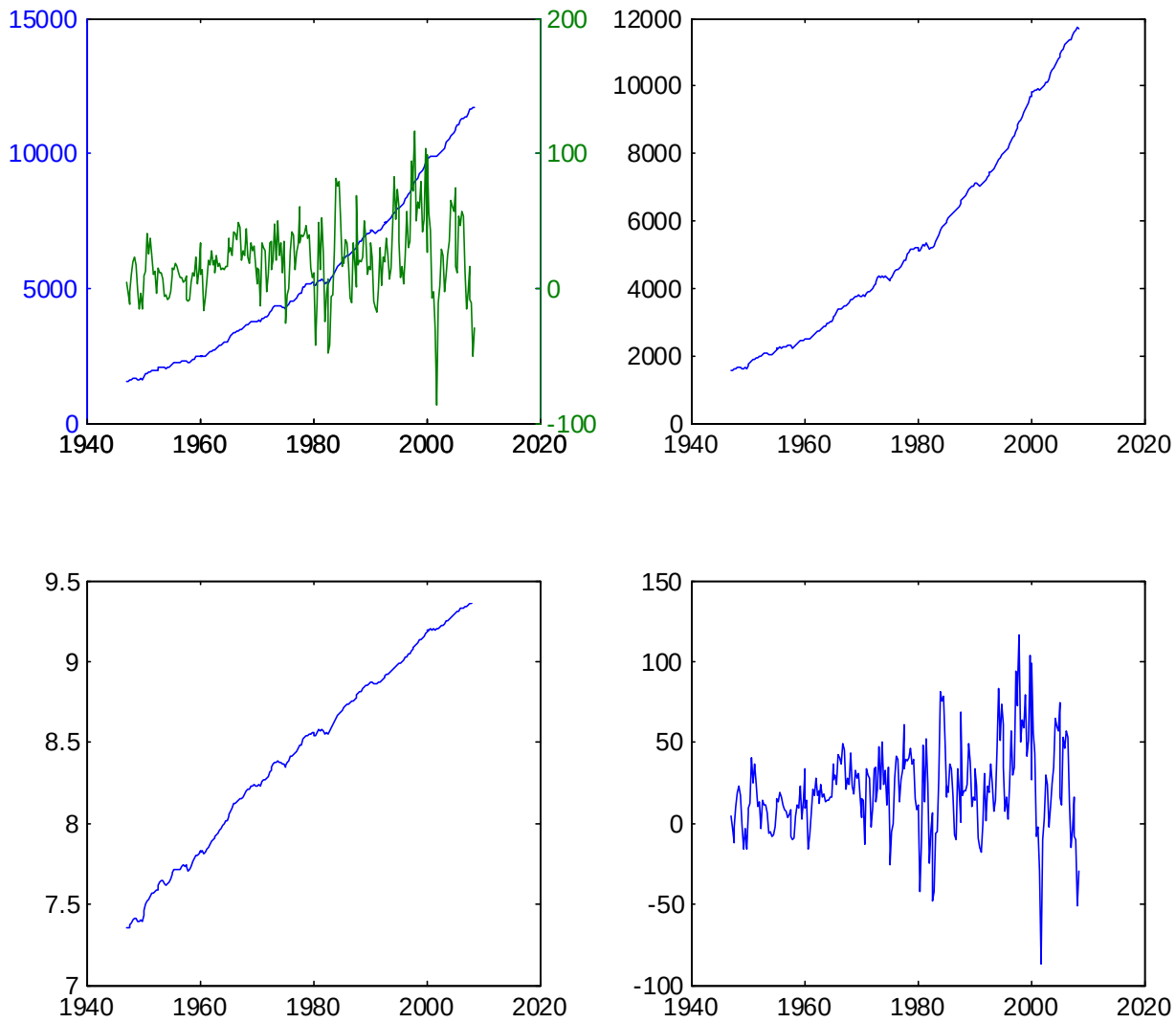
# Representing various graphics in the same figure

- 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=1947.0:0.25:2008.05;
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.)



# VIII – Exercise 1

# 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?

# VIII – Exercise 2

# Various cases of dynamic processes

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

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

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

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

3. 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, $\varepsilon_t \sim iid(0, \sigma^2)$, $t$ for the long term trend of $y$

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

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

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

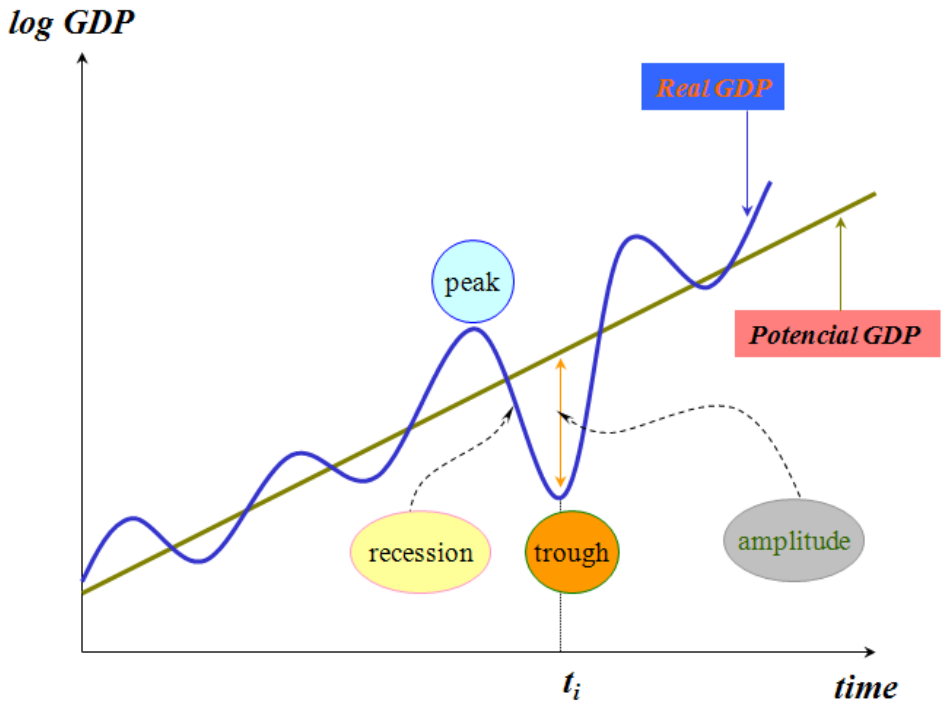
$$y_t = \frac{a_1 y_{t-1}^3 - a_2 y_{t-1}^2}{2}$$

# VIII – Exercise 3 : The Hodrick-Prescott Filter

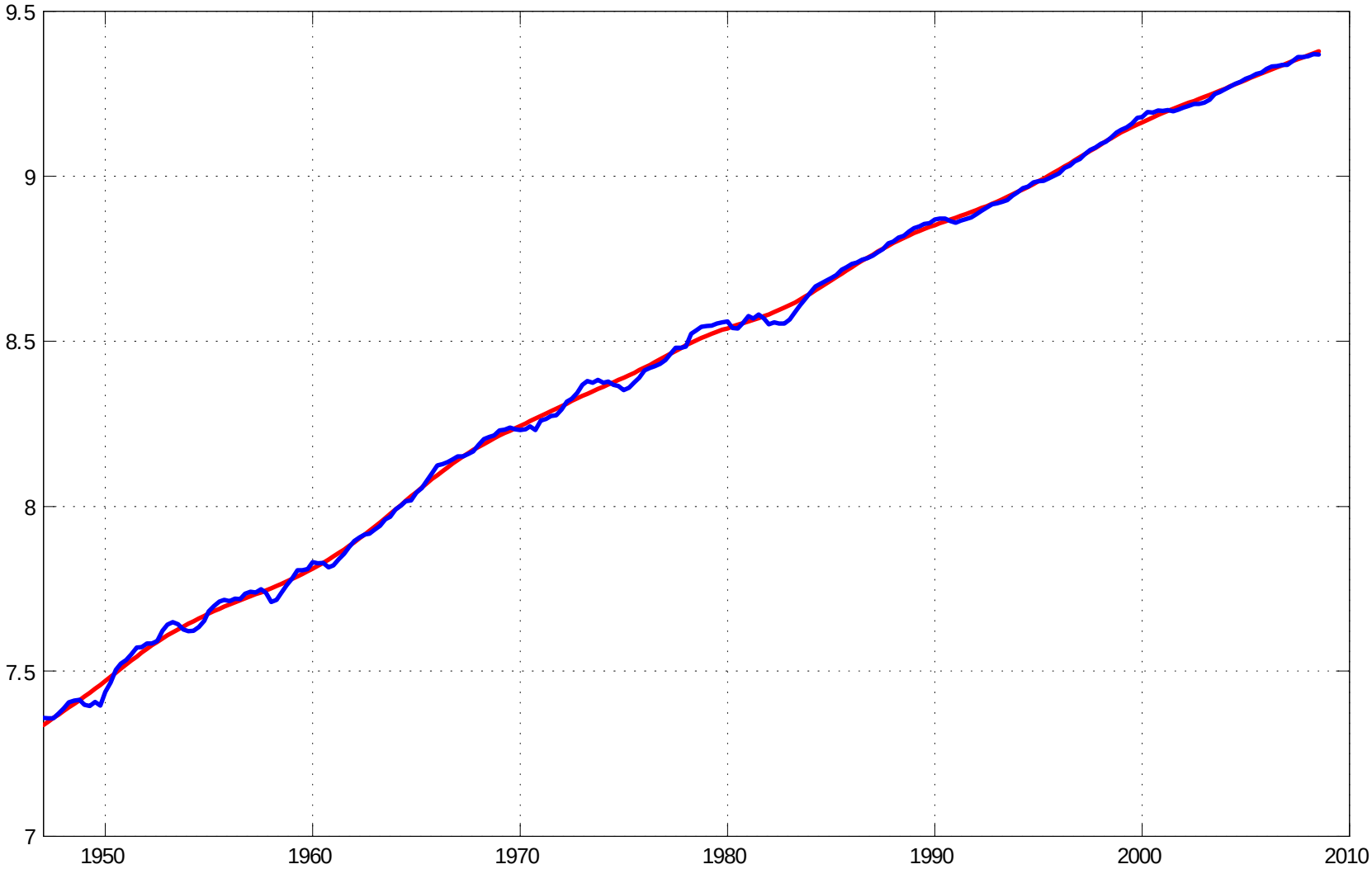
# Business cycles: a figure

## Definition

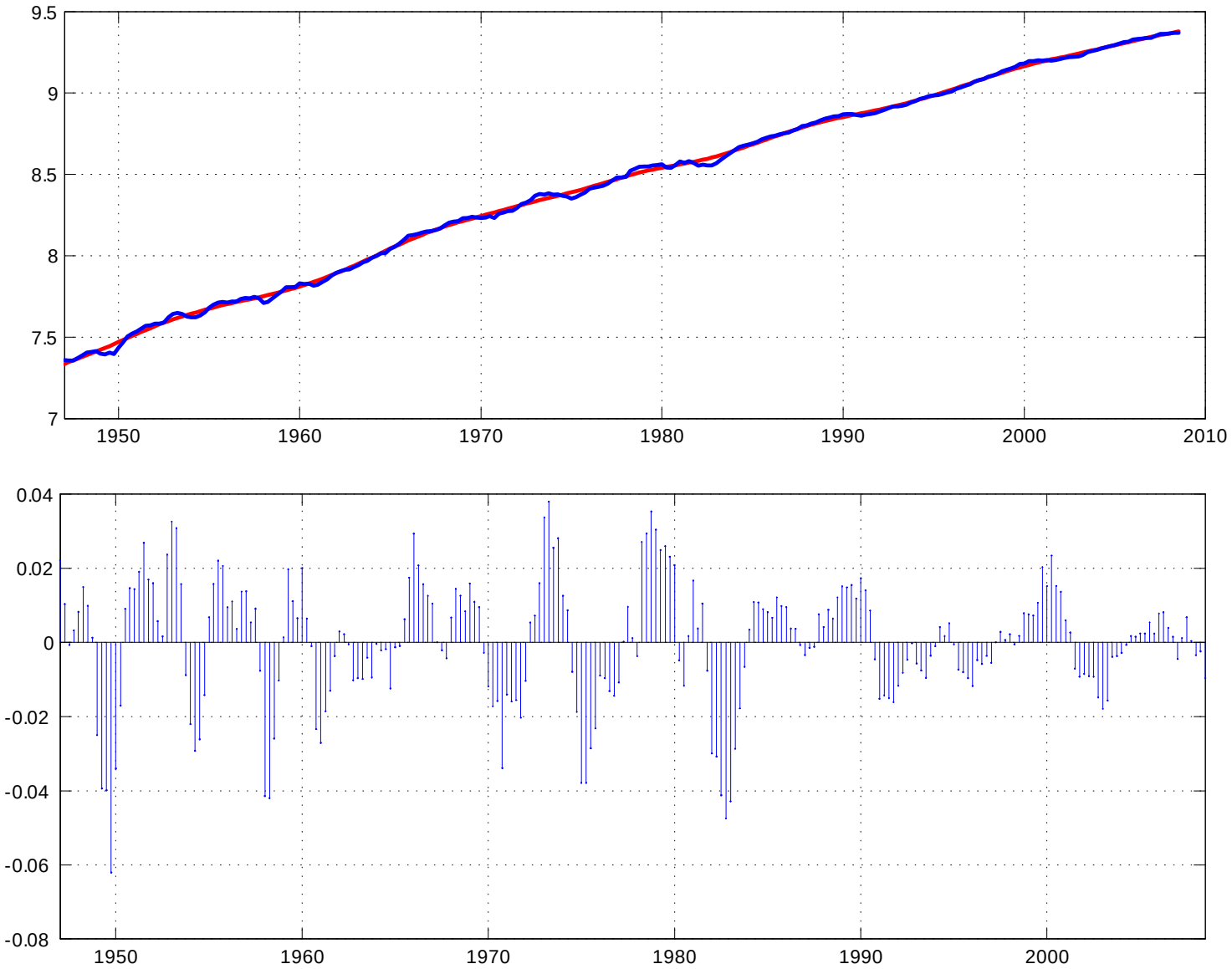
Business cycles are the recurrent short-run movements around a smooth long-run trend in endogenous economic variables, like production, investment, consumption, employment, price level, real wage, average labor productivity, money supply, among others.



# Business cycles: GDP and its trend (USA)



# Business cycles: GDP and its trend (USA)



# Business cycles: GDP and its trend (USA)

1. Using the file USdata.txt, calculate the HP Filter for GDP and for Consumption
2. Are the two variables above correlated or not?
3. Is Consumption a procyclical variable?

# IX – Bibliography

David Houcque (2005). INTRODUCTION TO MATLAB FOR ENGINEERING STUDENTS, Northwestern University.
Krister Ahlersten (2012). An Introduction to Matlab, Bookboon.com

# 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)*
- David Houcque (2005). Introduction to Matlab for Engineering Students, Northwestern University., Northwestern University.
- Krister Ahlersten (2012). An Introduction to Matlab, Bookboon.com
- Craig Warren (2012). An interactive introduction to MATLAB, University of Edinburgh