Quantitative Macroeconomics I
Introduction to Matlab

Grégoire Sempé
gregoire.sempe@psemail.eu

Paris School of Economics, Université Paris 1 Panthéon-Sorbonne

September 16, 2024

I thank Eustache Elina for his set of slides on which I heavily rely!

Nice to meet you!

What **you** should expect from the tutorials

Format of the tutorials sessions:

Learning Objectives:

What **you** should expect from the tutorials

Format of the tutorials sessions:

- Designed as complementary to Prof Broer's class $\rightarrow$ attend both classes!

Learning Objectives:

What **you** should expect from the tutorials

Format of the tutorials sessions:

- Designed as complementary to Prof Broer's class $\rightarrow$ attend both classes!
- A mix of theory, coding & problem sets $\rightarrow$ expect **a lot** of work

Learning Objectives:

What **you** should expect from the tutorials

Format of the tutorials sessions:

- Designed as complementary to Prof Broer's class $\rightarrow$ attend both classes!
- A mix of theory, coding & problem sets $\rightarrow$ expect **a lot** of work

Learning Objectives:

1. Create your toolbox to solve macroeconomic models with representative agent
   $\hookrightarrow$ Various numerical methods, with advantages and drawbacks
   $\Rightarrow$ **QM2 will build on QM1** and will focus on state-of-the-art heterogenous agent models

What you should expect from the tutorials

Format of the tutorials sessions:

- Designed as complementary to Prof Broer's class $\rightarrow$ attend both classes!
- A mix of theory, coding & problem sets $\rightarrow$ expect a lot of work

Learning Objectives:

1. Create your toolbox to solve macroeconomic models with representative agent
   - $\hookrightarrow$ Various numerical methods, with advantages and drawbacks
   - $\Rightarrow$ **QM2 will build on QM1** and will focus on state-of-the-art heterogenous agent models
2. Become familiar with dynamic programming / recursive methods
   - $\hookrightarrow$ Dominant in macro, widely used in labor, econ theory and structural econometrics …

What **we** expect from you

How to succeed in the class?

Two advice:

What **we** expect from you

How to succeed in the class?

1. Attend classes, and **ask questions** if you don't understand!

Two advice:

What **we** expect from you

How to succeed in the class?

1. Attend classes, and **ask questions** if you don't understand!

2. Study the problem sets, do them by yourself, code regularly

Two advice:

What **we** expect from you

How to succeed in the class?

1. Attend classes, and **ask questions** if you don't understand!

2. Study the problem sets, do them by yourself, code regularly

Two advice:

- Helping each other understand the methods is the best way to learn

What **we** expect from you

How to succeed in the class?

1. Attend classes, and **ask questions** if you don't understand!

2. Study the problem sets, do them by yourself, code regularly

Two advice:

- Helping each other understand the methods is the best way to learn

- Discuss research ideas with each others (thesis, ...)

Grading – *To be confirmed*

- End-of-semester exam — 70% of the final grade
- Around 5-6 problem sets, do be done **by groups of 2** — 30% of the final grade

  ↪ Even if you don't manage to solve the hardest problems, I expect to see some effort

  → Follow the general indications on the formatting of the problem sets

⇒ **First exercises** to be done in two week!

Overall outline

	Model	Computational Methods
PS 0	Solow Growth model	Basic tools + rootfinding algorithms
PS I	Neoclassical Growth model	Shooting, quasi-Newton methods
PS II	Stochastic growth model	Log-linearization, perturbation
PS III	Stochastic growth model	Dyn. Programming, Markov shocks, VFI
PS IV	Real Business Cycle model	VFI and EGM with endog. labor supply
PS V	NK – Representative Agent	Dynare and SMM

Outline for today's bootcamp

PARIS SCHOOL OF ECONOMICS
ECOLE D'ECONOMIE DE PARIS

1. Motivation & general takes on software
2. Matlab basics (arrays, loops, conditional statements, plots, functions)
   - Arrays, matrices, functions
   - Loops, conditional statements
   - Vectorization, plotting
3. Linear Interpolation & Vectorization

→ Problem set 0 to warm-up!

Why learning numerical techniques?

1. Mathematical sciences always face a trade-off btw. realistic assumptions and solvability

   ↪ Solving your model numerically *partially* solves this issue

2. Makes you able to see the effects of a policy on the **distribution** (HA model)

   a) Effects of macro policies on inequalities (e.g. fiscal policy)

   b) Macroeconomic dynamics are heavily modified! (e.g monetary policy)

3. Build an economic intuition by playing with your model

   a) In partial equilibrium, study the effects of prices on individual decisions

   b) In general equilibrium, study the effects of shocks (e.g. taxes) on prices & aggregates

Why use Matlab?

Pros:

1. Intuitive language
2. Easy to debug: easy to know what you are manipulating
3. Very efficient at handling matrices
4. Widespread use among macroeconomists (e.g central banks)

$\Rightarrow$ Probably not the most efficient language but good enough for simple models

Why use Matlab?

PARIS SCHOOL OF ECONOMICS
ECOLE D'ECONOMIE DE PARIS

Pros:

1. Intuitive language
2. Easy to debug: easy to know what you are manipulating
3. Very efficient at handling matrices
4. Widespread use among macroeconomists (e.g central banks)

$\Rightarrow$ Probably not the most efficient language but good enough for simple models

Cons:

1. Not open source $\rightarrow$ expensive, code can break across versions in the long run
2. Relatively slow compared to low-level languages…
3. Hard to use together with other languages

$\Rightarrow$ Alternatives: **Julia**, Python – Numba, C++, JAX (*see Fernandez Villaverde's list*)

Matlab Interface

Divided in four parts:

1. Command window: where you can type and execute commands directly
2. Editor: where you end up writing your code if you want to keep track of it.
   Note 1: You only use the command window for tests or debugging
   Note 2: Use comments starting with % for your future readers and for yourself!
   Note 3: End a line of code with ; if you don't want to see it printed in the command window
   $\rightarrow$ To run a script : Editor $>$ Run
3. Workspace: all variables, functions, matrices, etc. available to work with
4. Current folder: what scripts you have direct access to
   Note 4: Keep functions you use in your current folder or in the folder that you have included in your *search path* (Home $>$ Environment $>$ Set Path $>$ Add folders)
   Search path : files Matlab have access to

General functions

- Want to clear the workspace?

```
clear
```

- Want to clear the command window?

```
clc
```

- Want to save your workspace into a file named backup?

```
save backup.mat
```

- Want to load your file backup?

```
load backup.mat
```

When in trouble

- You have access to detailed explanations of any function when writing help or doc followed by the name of the function in the command window. Ex with clear function:

```
help clear
```

```
doc clear
```

- LLMs are quite good at explaining how functions work / giving examples…

  ↪ ChatGPT, but also open source alternatives: Mistral Codestral, Llama…

⟹ But always check if the answer provided is right!

Building scalars, vectors and matrices

- Build a scalar:

```
a = 2;
```

- Build a row vector:

```
b = [1 2 3];
```

- Build a column vector:

```
c = [1;2;3];
```

- Build a matrix:

```
d = [1 2; 3 4];
```

Discretization of an interval

- Equally spaced row vector from *a* to *b* with *n* elements:

```
e = linspace(a,b,n);
```

- Equally spaced row vector from *a* to *b* with an increment of *x* (stop before *b* if the increment does not fit):

```
f = a:x:b;
```

- Logarithmic spaced row vector from 10^a to 10^b with *n* elements:

```
g = logspace(a,b,n);
```

Direct command to build matrices

- Construct a matrix of 0 of size $m \times n$

```
zeros(m,n)
```

- Construct a matrix of 1 of size $m \times n$

```
ones(m,n)
```

- Construct a matrix of size $m \times n$ of random draws from an uniform distribution in $[0, 1]$

```
rand(m,n)
```

How to navigate in a matrix: indexing 1/2

PARIS SCHOOL OF ECONOMICS
ECOLE D'ECONOMIE DE PARIS

How to choose specific element(s) in a matrix? Define:

```
h = rand(10,10);
```

- How to pick the element on the 6th row and 7th column:

```
h(6,7)
```

- How to pick all the elements on column 4:

```
h(:,4)
```

Note: In Matlab indexing starts at 1 and not 0! ($\neq$ Python)

How to navigate in a matrix: indexing 2/2

- How to pick the first three rows in column 4

```
h(1:3,4)
```

- How to exclude the first and the last column:

```
h(:,2:end-1)
```

Other object: arrays

Generalization of matrices in more than two dimensions. Ex for an array in 3 dimensions:

```
h = rand(3,5,8);
```

$\rightarrow$ Can be visualised as a book of 8 pages with 3×5 elements of each page

Other object: structure array

A structure array is composed of several fields that can each contain any type of data.

$\rightarrow$ Use the dot when naming a variable to create a structure.

```
par.alpha = 0.3;
par.beta = 0.95;
par.delta = 0.1;
```

$\Longrightarrow$ Creates a structure *par* with all your parameters.

$\Longrightarrow$ Useful to pass parameters in an *user-written* function (see last section)

Operations

- Standard (matrix or scalar) operators '+', '-', '/', '\','*' '^'
- **Element-by-element operators** by adding a **dot** in front of the operator : '.*', './', '.^'
- Comparison operators
  - equal ==
  - not equal ~=
  - bigger or equal >=
  - smaller or equal <=

$\Rightarrow$ A comparison operation will yield either **1 if the condition is true** and 0 if not

Run Section

PARIS SCHOOL OF ECONOMICS
ECOLE D'ECONOMIE DE PARIS

You can subdivide your code in different sections and run your code only in one specific section

1. Start a line with '%%' to create a section
2. Select a section and click on Editor > Run Section to run it

```
%% 1st section
A = 1;
%% 2nd section
B = rand;
```

Measure the time to run a code: tic toc

You can measure the time a code takes to run using the ’tic toc’ function

1. Write tic and jump a line
2. Include the code you want to measure
3. Jump a line and write toc

```
tic
A = rand(10);
B = inv(A);
toc
```

2D plots

- plot(x,y) plots used to create 2D plot
- Plot all pairs $(x_1, y_1), \ldots, (x_n, y_n)$
- Connect all the dots with a line

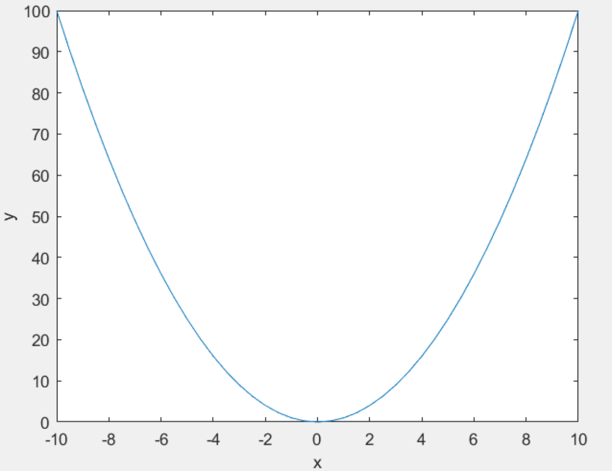
$\Longrightarrow$ x and y must be vectors of the same size

2D plots : example 1

How to plot $f(x) = x^2$ on interval [-10,10]:

```
x = -10:0.5:10;
y = x.^2;
figure(1)
plot(x,y)
xlabel('x')
ylabel('y')
```

2D plots : example 1



Surface in 3D space

Objective: We want to plot z=f(x,y) for all possible (x,y)

- We need a value of z for each pair (x,y)
- x and y are vectors composed of the elements where the function is evaluated
- Z will be a matrix: for each given x, we need to compute z for all possible y; and for each given y we need to compute z for all possible x
- To get our matrix Z we need to transform X and Y into matrices

Surface in 3D space : transform vectors into matrices

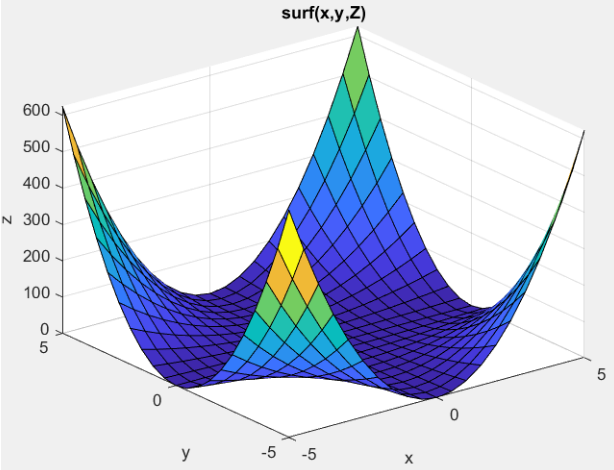
We want to transform x and y into matrices such that applying the transformation f(.,.) to X and Y yields Z

- The function [X,Y] = meshgrid(x,y) yields two matrices with the first having the rows filled of copies of the vector X and the second one having the columns filled of copies of the vector Y
- Now, applying the transformation f(.,.) to our X and Y will yield Z for all possible pairs (x,y)
- The function surf(x,y,Z) plots the values in matrix Z as heights above a grid in the x-y plane defined by X and Y

Surface in 3D space: example

```
x = -5:0.5:5;
y = -5:0.5:5;
[X,Y] = meshgrid(x,y);
Z = (X.*Y).^2;
figure(2)
surf(x,y,Z)
xlabel('x'); ylabel('y'); zlabel('Z');
title('surf(x,y,Z)')
```

Surface in 3D space: example



Loops and Conditional Statements

Conditional statement : if

Syntax example:

```
if x > 10
  % command block 1
elseif x > 5
  % command block 2
else
   %command block 3
```

1. If $x > 10$ then execute command 1
2. If not, then:
   2.1 If $x > 5$ then execute command 2
   2.2 If not then execute command 3

Loop for

- Runs the interior code a pre-specified number of times
- At each iteration the loop control variable is increased by one

Loop for : example 1

Generate 50 random number in uniform distribution over $[0, 1]$ and compute the average:

Loop for : example 1

Generate 50 random number in uniform distribution over $[0, 1]$ and compute the average:

```
a = rand(50,1);
mean_a = 0;
for i=1:size(a,1)
  mean_a = mean_a + a(i);
end
mean_a = mean_a/size(a,1);
```

Loop for : example 2

Compute the following sum:

$$\sum_{k=1}^{100} \sum_{i=1}^{k} i$$

Loop for : example 2

Compute the following sum:

$$\sum_{k=1}^{100} \sum_{i=1}^{k} i$$

```
x=0;
for k=1:100
   for i=1:k
      x=x+i;
   end
end
```

Loop for : example 3

Compute 100!

Loop for : example 3

Compute 100!

```
%Method1
x=1;
for i=1:100
    x=x*i;
end
%Method2
prod(1:100);
```

Loop while

- Runs the interior code as long as a condition is true. Exit the loop when it is false
- Ex-ante the number of iterations is unknown
  $\rightarrow$ Possible that it will keep running if the condition is always true
- Sometimes useful to include a maximum number of iterations

Loop while : example 1

Compute the limit of the following sequence: $u_{n+1} = -\frac{1}{2}u_n + 3$ with $u_0 = 5$

```
u = 5;
dif = 10;
i=0;
while dif > 1e-8
    u_prime = -0.5 * u + 3;
   dif = abs(u_prime - u);
   i = i + 1;
   u = u_prime;
end
```

If you prefer, you can use a maximum number of iterations + break

Exercise: the Solow model

Compute how much period does it take to reach the steady state value of the capital stock (at an approximation error of 10^{-10}) given an initial condition $k_0 = 0.1$, and that $s = 0.4$, $\alpha = 0.3$, and $\delta = 0.1$:

Exercise: the Solow model

Compute how much period does it take to reach the steady state value of the capital stock (at an approximation error of 10^{-10}) given an initial condition $k_0 = 0.1$, and that $s = 0.4$, $\alpha = 0.3$, and $\delta = 0.1$:

```
alpha = 0.3; s = 0.4; delta = 0.1; k = 0.1;
err = 1; t=0;
while err > 10^(-10)
    knew = s * k^alpha + (1 - delta) * k;
    err = knew - k;
    k = knew;
    t=t+1;
end
```

Loops and Conditional Statements

Functions : build-in and user-written

- Build-in functions are already available rand(.), diff(.) etc.
- Two types of user-written functions:
    1. Anonymous functions
    2. Functions (either saved in script or in a separate file)

Some useful build-in functions

- The function max(x) is one of the most useful function
- Extract the highest value in a vector and gives the index associated

```
xx = rand(1,5);
[max˙xx,i]=max(xx);
```

$\rightarrow$ Knowing the index gives the optimal policy function. More on that next class...

Some useful build-in functions

From matrix to vector to matrix:

```
% Define a matrix
A=[1,2,3;4,5,6]
% Vectorize it (column vector)
A˙vec = A(:);
% Get back your original matrix
A˙new = reshape(A˙vec,2,3);
```

$\rightarrow$ Useful to speed up codes to do operations on vectors than going for one cell at a time

Some useful build-in functions

- In simulations, it can be useful to always get the same sequence of random numbers
- In that case, you have to set a seed with any integer to the random number generator

```
rng(2);
x = rand(1,5)
```

Running the code above will always print the following vector:

```
x = [0.4360 0.0259 0.5497 0.4353 0.4204]
```

Some useful build-in functions

- The size(.) function returns a row vector whose elements are the lengths of the corresponding dimensions of A

```
A = [1,2,3;4,5,6];
size(A)
```

$\rightarrow$ Returns a row vector $[2, 3]$

- size(.,x) returns a scalar of the length of the dimension x of our matrix

```
size(A,2)
```

$\rightarrow$ Returns 3

User-written functions

- You can write your own function as a script saved in a .m-file
- Your function must be saved in your current folder or in a folder that you have added to your search path if you want to use it
- The syntax must be the following:

```
function [y1,...,yN] = myfun(x1,...,xM)
  % interior command block
end
```

(x1,...,xM) are the inputs to the function and (y1,...,yN) are the outputs that come out of it

User-written functions

Build a function that takes a number and returns the square, the square root, and the factorial

```
function [a,b,c] = fun1(x)
   a = x^2;
   b = x^(1/2);
   c = prod(1:x);
end
```

To use it, write in a script or the command window:

```
fun1(x)
```

with x, any positive integer

Script vs function: differences

- What they use as **inputs**:
  - Functions only use the received inputs
  - Scripts have access to the whole workspace
- What they have as **output**:
  - Functions only give the demanded output and erase the rest
  - Scripts return all variables used in it

Anonymous functions

- Anonymous functions are functions defined within a script (have a name but not their own .m file)
- Anonymous because they don't have their own .m-file but they do have a name

Example:

```
sqrt = @(x) x.^(1/2); sqrt(144)
```

Anonymous functions

Once a function is saved in the workspace, it can be easily plotted:

```
fun = @(x) 0.1*x.^2 + sin(x);
fplot(fun,[-5,5])
```

Optimize

PARIS SCHOOL OF ECONOMICS
ECOLE D'ECONOMIE DE PARIS

Tricks to write a high-performance code

1. Vectorization
   $\rightarrow$ Matlab prefers vector/matrix operations than codes using loops

2. Write your own functions
   $\rightarrow$ Example: maximization problem with a solver vs golden algorithm
   Wait for QM2...

Optimize with vectorization

Example 1: Evaluate a function over a discrete interval:

```
a = linspace(0,10,1000);
f˙a = zeros(1,10000);
tic
for i=1:size(a,1)
   f˙a = exp(-a(i));
end
toc
tic
f_a_vec = exp(-a);
toc
```

Optimize with vectorization

Example 2: Element-by-element matrix operation:

```
A = rand(1000,1000);
B = zeros(1000,1000);
for i=1:size(A,1)
  for j=1:size(A,2)
      B(i,j) = A(i,j)*A(i,j);
  end
end
B˙alt = A.*A;
```

Which method works best?

Problem set 0 - Warm-up

Ex 1: Solving the usual growth model

1. Define your parameters $\delta = 0.1$, $\alpha = 0.3$, and $k_0 = 0.1$ using a structure
2. Write a function to solve the Solow model, given the saving rate s (input).
3. From now on, set $s = 0.7$
   3.1 Compute the number of period to reach the steady state
   3.2 Solve for k^* in $sf(k) - \delta k = 0$ using the Newton method (code the function and provide the analytical derivative).
   3.3 Compare the steady state capital stock with the analytical solution
4. Create two equi-spaced grids for the saving rate (N = 50 and N = 500)
   ↪ Compute the steady-state consumption for each saving rate. Find the golden rule saving rate using the *max* function.
5. Compare with the analytical solution. Which on-grid maximization gives the best approximation?

Ex 2: An introduction to non-linear equation system

1. Assume you have a reduced form model of supply and demand with $Q_d = \frac{b}{1+2P^2} + 5$ and $Q_s = P^2 + eP - T$. Take the following parameters: $b = 10$, $e = 3$, $T = 2$.
2. Write the equation system and solve it using the Broyden method (write a function for Broyden, and one for the system given guesses on prices and quantities)