Computational Tools for Macroeconomics using MATLAB

Week 1 – Introduction to MATLAB & Computational Thinking

Cristiano Cantore

Sapienza University of Rome

Who am I

- Personal Website: www.cristianocantore.com
- Email: cristiano.cantore@uniroma1.it
- Office hours: by appointment.

Course Info

- Course website: www.ccantore.github.io/computational-macro-matlab
- Classes: Monday 10:00-12:00 - ECODIR LAB
  - I will usually start at 10.05 and finish around 11.35.
  - I will be available for questions after the class until 11.55.
- If you can bring your laptop!
- I have created a google group for the course. Make sure to join it!
  - I will post announcements and important information there.
  - It will also be the place to ask questions and get help.
  - Do not email me directly with questions about the course, use the google group instead!

Course Overview & Objectives

- Introduce students to computational tools for macroeconomic analysis.
- Gain proficiency in MATLAB for solving and simulating economic models.
- Bridge theory and practice: from analytical models to numerical implementation.
- Prepare for empirical work and research projects in macroeconomics.

Why Computational Tools in Macroeconomics?

- Modern macro relies on numerical methods for solving models.
- Many models are too complex for closed-form solutions.
- Simulation is essential for quantitative policy analysis.
- MATLAB is widely used in academia, central banks, and policy institutions.

Course Structure

- Weekly lectures: mix of theoretical introduction and coding practice.
- Readings: won't follow one textbook. most material on coding is outdated one year after! I suggest some readings every week for dig deeper but first make sure you understand the slides.
- Hands-on MATLAB sessions embedded in lectures.
- Homework exercises with provided templates.
- Website with all the material. **Important**: The website will be populated as we move along the course. Make sure you check for updates and
- Assessment: TBD. Probably you can choose between submitting weekly assignments (randomly marked) or a final project.

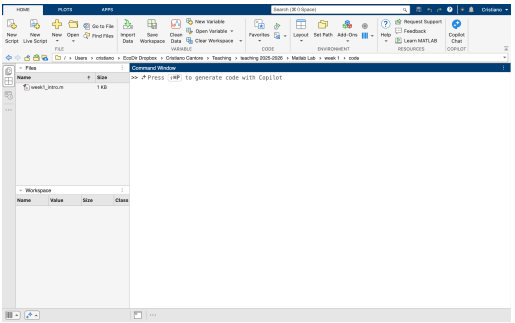
Learning Outcomes

By the end of this week, students will be able to:

1. Understand the MATLAB interface and basic commands.
2. Manage scripts and functions.
3. Use variables, vectors, matrices, and basic operations.
4. Create and interpret simple plots.
5. Perform basic operations in MATLAB.

MATLAB Interface

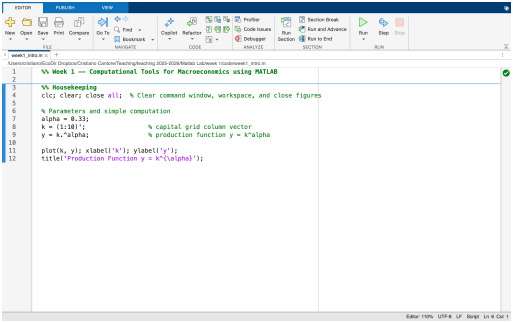
- **Command Window**: run commands, see output.
- **Workspace**: variables currently in memory.
- **Current Folder**: file navigation and path context.
- **Command History**: recall previous commands. (not shown)



Screenshot: MATLAB Desktop (Command Window, Workspace, Current Folder)

MATLAB Interface

- **Editor**: write and run scripts/functions (`.m` files).



Screenshot: MATLAB Desktop (Editor)

MATLAB Syntax Basics

- **Variables**: dynamic typing, case-sensitive (`X` $\neq$ `x`).
- **Operators**: `+ - * / ^` (matrix ops), `.* ./ .^` (elementwise).
- **Vectors/Matrices**: row `[1 2 3]`, column `[1;2;3]`.
- **Indexing**: `A(i,j)`, slices `A(:,2)`, ranges `1:10`.
- **Comments**: `% this is a comment`.
- **Semicolons**: suppress output with `;` for cleaner logs and faster runs.

Scripts vs. Functions

Scripts (`.m`)

- Sequence of commands executed top-to-bottom.
- Share the base workspace.
- Good for quick workflows and reproducing analyses.

Functions (`function ... end`)

- Take inputs, return outputs.
- Own local workspace (avoid polluting the base workspace).
- Reusable, testable components (recommended for larger projects).

Example: Minimal Script

`week1_intro.m`

```
% Parameters and simple computation
alpha = 0.33;
k = (1:10)';                    % capital grid column vector
y = k.^alpha;                   % production function y = k^alpha

plot(k, y); xlabel('k'); ylabel('y');
title('Production Function y = k^{\alpha}');
```

- Save as `week1_intro.m` in the Current Folder.
- Run via `Run` button or type `week1_intro` in Command Window.

Example: Minimal Function

prod_cd.m

```
function y = prod_cd(k, alpha)
%PROD_CD Cobb-Douglas production y = k^alpha
%   y = PROD_CD(k, alpha) computes k.^alpha elementwise
    y = k.^alpha;
end
```

- Save as `prod_cd.m`. Call from a script or Command Window:
- `k = (1:10)'; y = prod_cd(k, 0.33); plot(y);`

Running Commands, Scripts, and Functions

- **Commands**: typed in the Command Window (e.g., `x = 2+2`).
- **Scripts**: run with the green `Run` button or by name.
- **Functions**: called with inputs/outputs (e.g., `y = prod_cd(k,0.33)`).

Paths and Current Folder

- Ensure your **Current Folder** contains your code or add folders with `addpath()`.
- Use `which functionName` to check what MATLAB is calling.

Common Pitfalls in MATLAB

- **Case sensitivity:** `Var` $\neq$ `var`.
- **Overwriting built-in functions:** Avoid using names like `sum`, `mean`, `plot`.
- **Current folder confusion:** Make sure your script is in the active directory.
- **Semicolons:** Missing semicolons prints output for every line.
- **Vector vs. matrix dimensions:** Watch for errors in element-wise (`.*`, `./`) vs. matrix operations (`*`, `/`).
- **Clear misuse:** Excessive use of `clear all` removes useful variables and slows work.

Data Types: Scalars, Vectors, Matrices

- **Scalar:** Single number.
- **Vector:** One-dimensional array (row or column).
- **Matrix:** Two-dimensional array.

Examples

```
a = 5;                  % Scalar
v = [1, 2, 3];          % Row vector
u = [1; 2; 3];          % Column vector
M = [1 2 3; 4 5 6];     % 2x3 matrix
```

Indexing and Slicing

- MATLAB uses **1-based indexing**.
- Access elements with `A(i,j)`.
- Colon operator (`:`) for slices and ranges.

Examples

```
v = [10, 20, 30, 40, 50];
v(1)         % first element -> 10
v(end)       % last element  -> 50
v(2:4)       % slice -> [20 30 40]

M = [1 2 3; 4 5 6; 7 8 9];
M(2,3)       % element at row 2, col 3 -> 6
M(:,2)       % entire 2nd column -> [2; 5; 8]
M(1:2,:)     % first 2 rows, all columns
```

Preallocation

- Efficient coding requires preallocating arrays.
- Avoids MATLAB dynamically resizing in loops.
- Use `zeros`, `ones`, `nan`.

Example

```
% Bad practice (slow)
for i = 1:1000
    x(i) = i^2;
end

% Good practice (preallocate)
x = zeros(1,1000);
for i = 1:1000
    x(i) = i^2;
end
```

Basic Operations

- Standard arithmetic: +, −, ⋆, /, ^.
- **Matrix vs. elementwise** operations.

Examples

```
A = [1 2; 3 4];
B = [5 6; 7 8];

A * B      % matrix multiplication (2x2 * 2x2)
A .* B     % elementwise multiplication
A.^2       % elementwise square
```

Built-in Functions

- MATLAB includes many functions for numerical work.
- Functions often work on vectors/matrices automatically.

Examples

```
v = [1, 2, 3, 4, 5];

mean(v)      % average -> 3
sum(v)       % sum -> 15
max(v)       % maximum -> 5
std(v)       % standard deviation
plot(v)      % quick plot
```

Importing and Exporting Data

- Load data from files with built-in functions.
- MATLAB supports CSV, Excel, MAT-files.

Examples

```
% Load CSV
data = readmatrix('data.csv');
% Save data
writematrix(data, 'output.csv');

% Load Excel
data = readtable('data.xlsx');
% Save data
writetable(data, 'output.csv');
```

Tip

Use the **Import Tool** (GUI) to preview data before importing.

Saving and Loading the Workspace

- **Save current workspace:**

```
save('myWorkspace.mat')
```

- **Reload saved workspace:**

```
load('myWorkspace.mat')
```

- Useful to:
  - Resume work later without re-running scripts.
  - Share variables with colleagues.
- MATLAB saves variables in binary `.mat` format.

Getting Help

- MATLAB provides extensive documentation.
- Use `help` and `doc` commands.

Examples

```
help mean         % short description
doc mean          % open full documentation
lookfor average   % search functions related to 'average'
```

Tip

Use **tab-completion** in the editor to explore function options.

Plotting Basics

- `plot()` creates simple 2D line plots.
- Add labels and titles with `xlabel()`, `ylabel()`, `title()`.

Example

```
x = 0:0.1:10;
y = sin(x);

plot(x, y)
xlabel('x-axis')
ylabel('sin(x)')
title('Sine function')
```

Tip

Always label axes and add a title for clarity!

Line Styles and Economic Data

- Customize plots with line styles (`'-'`, `':'`) and markers (`'o'`, `'*'`).
- Useful for comparing series (e.g., GDP vs. consumption).

Examples

```
t = 2000:2010;
GDP = [1.5 1.7 2.0 2.3 2.5 2.7 2.9 3.1 3.4 3.6 3.8];
C   = [1.2 1.3 1.5 1.6 1.8 1.9 2.0 2.2 2.3 2.5 2.6];

plot(t, GDP, '-o', t, C, '--*')
xlabel('Year')
ylabel('Trillion USD')
title('GDP and Consumption')
legend('GDP','Consumption')
```

Challenge

Given: a vector of annual GDP data (levels).
Tasks:

1. Compute annual **growth rates** (in %).
2. Compute **average GDP**.
3. **Plot** GDP over time with labels and title.
4. **Save** the workspace and **export** the figure.

Deliverables:

- Script: `week1_challenge.m`
- Files saved: `myWorkspace.mat`, `gdp_plot.png`

Homework / Practice

- Create a MATLAB script to load a CSV with nominal and real GDP data.
- Compute GDP deflator and inflation.
- Plot Real GDP, Nominal GDP and Inflation over time with labels and title.
- Compute growth rates for Real GDP and plot it.
- Save plots as PNG and MATLAB figures.

Next Week

Next week: **Week 2 – Programming Basics: Loops, Conditionals, Functions**.