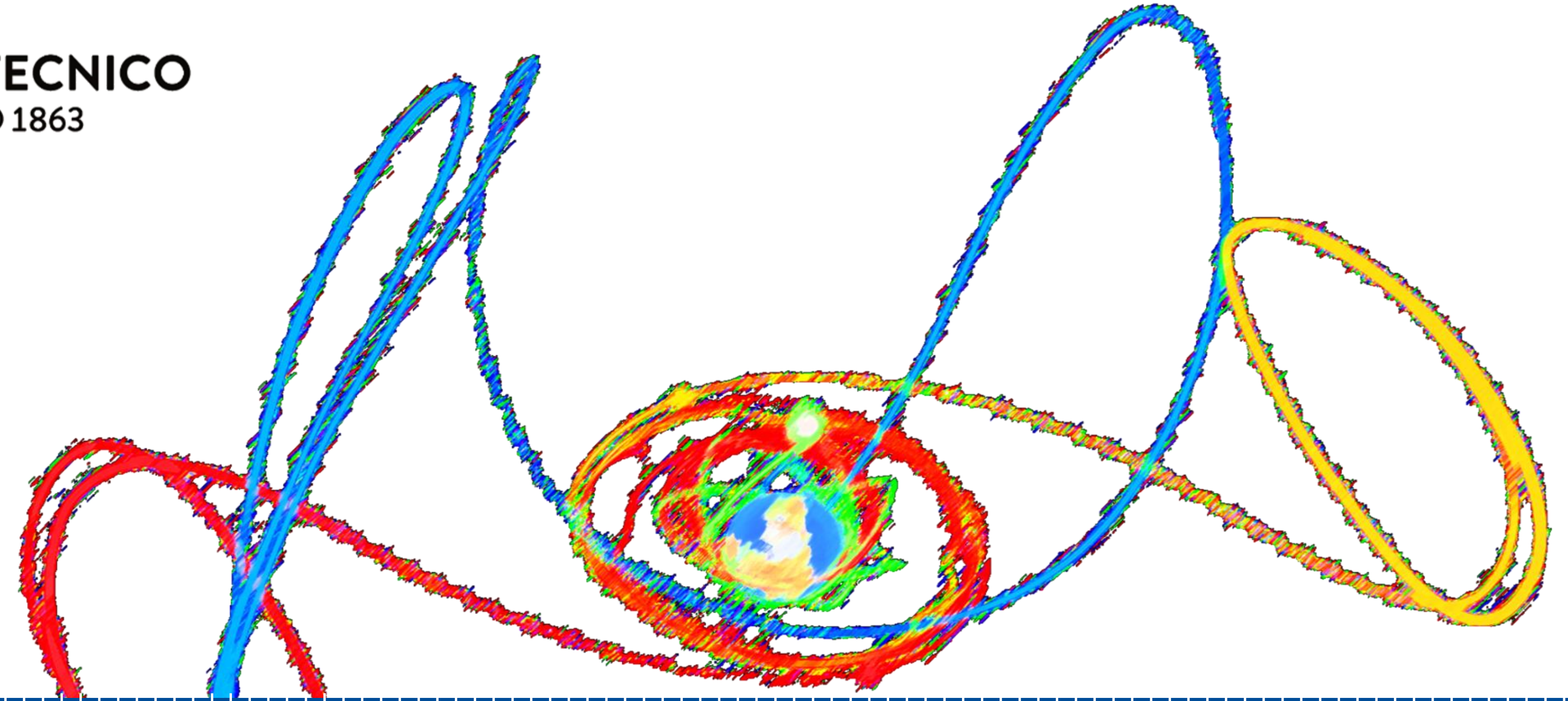


Orbital Mechanics
Module 0: Introduction & MATLAB fundamentals

Juan Luis GONZALO GOMEZ, Lorenzo GIUDICI, Camilla COLOMBO

Department of Aerospace Science and Technology, Politecnico di Milano

Academic year 2021/22

Module Contents

Introduction & MATLAB fundamentals

- **Project lab introduction**
  - Teaching staff
  - Lab groups
  - Project
  - Past year students
- **MATLAB fundamentals**
  - MATLAB learning resources
  - Code structuring
  - Best coding practices
  - Figures

Teaching staff

Contacts

- Camilla **Colombo**
  - Email: camilla.colombo@polimi.it
  - Meeting time: please arrange via email
  - Tel: 8352
- Juan Luis **Gonzalo Gomez**
  - Email: juanluis.gonzalo@polimi.it
  - Meeting time: please arrange via email
  - Tel: 8401
- Lorenzo **Giudici**
  - Email: lorenzo1.giudici@polimi.it
  - Meeting time: please arrange via email
  - Tel: 8401

Teaching staff

Lorenzo Giudici

2018: BEng, Politecnico di Milano, Mechanical Engineering

2020: MEng, Politecnico di Milano, Space engineering

2021-present: PhD Candidate, Politecnico di Milano

- Extension of a debris population model based on the density of orbiting objects in the phase space of Keplerian elements, through an analogy with fluid dynamics.

Teaching staff

Juan Luis Gonzalo

Ingeniero Aeronáutico (equiv. to a laurea quinquennale in Aerospace Engineering), Universidad Politécnica de Madrid

M.Sc. in Aerospace Engineering, Universidad Politécnica de Madrid

Ph.D. in Aerospace Engineering, Universidad Politécnica de Madrid

- Propagation and optimal control of space trajectories using perturbation methods

Visiting Ph.D. student at PoliMi, Department of Aerospace Science and technology

- Indirect optimization methods, applications to end-of-life de-orbiting

Postdoctoral research fellow at Politecnico di Milano, Department of Aerospace Science and Technology

- Space Situational Awareness, asteroid orbit manipulation

Lab groups

Rules

- The lab project will be carried out in **teams**, with the following **rules**:
  - Each team should have **4 members.**
  - To promote diversity, each team **must** include at least one student who did not study the bachelor’s degree at PoliMi.
  - **You can register your team using the activity available in WeBeep (section “Project lab”).**
  - The teams could be the same for the course of Spacecraft Attitude Dynamics.

- Also, please consider the following **recommendations:**
  - **Passing the assignment is a prerequisite to attend the final exam**. If you do not intend to do the assignment now, avoid teaming up with people that do.
  - **This is a team effort**. You can distribute the tasks, but everybody is responsible for the final result. At the oral presentation, **any member can be questioned about any part of the work**.

Lab groups

Teamwork

As in a **Mission Analysis team** at **ESA**, you will also work in a group:

- Members of the group must **cooperate**: you are advised to distribute the work among the team, but **everyone is responsible for all the work done in the project**.
- Make decisions towards design solutions based on numerical/analytical/physical evidence and analyses: you must always be able to **motivate your design choices**. *You are supposed to perform the preliminary mission analysis of a real mission*.
- During the final review (oral presentation), **any team member can be questioned about any part of the work**.

Project

Contents

- The **project** consists of **2 assignments** based on the computer lab
  - Fly-by explorer mission
  - Planetary explorer mission

- Project evaluation includes **2 parts**:
  - **Deliverables** (1 submission per group)
    - **Project report**: A single PDF report on the assignments, of maximum 20 pages
    - **Presentation slides**
    - **Simulation codes and results**
  - **Oral presentation (final review)**
    - 15 minutes followed by questions about the assignments and the theory of the course
    - All team members must participate in the oral presentation
    - Any student can be questioned about any part of the work

Project

Submission and deadlines

- The deliverables must be submitted through **WeBeep**
  - Submit a **single ZIP file** with all the deliverables (report, slides, and code), named "OrbitalMech_aaaaaaaa_group_nn.zip", where aaaaaaaa is the academic year and nn the group number (e.g., OrbitalMech_20212022_group_42.zip).
  - Submission via email will not be considered.
  - **Changes to the deliverables after the deadline will not be considered**.
- Deadlines:
  - **Deliverables must be submitted by 7 January 2022**.
    - Delivering the project is a must condition for the oral presentation and attending the written exam.
  - **Oral presentation**
    - Dates will be available during the Winter, Summer and September exam sessions. They will be notified prior to each exam session.
    - Presentation to be done before or within the same exam session (winter/summer/autumn) when the written is done.

Project

Report

- **Single PDF of maximum 20 pages for both assignments.**
- Include a **front page** with:
  - Title,
  - Group number, academic year,
  - For each member: full name, matriculation number, and person code.
- The report should contain **explanations, data, figures, and tables supporting your design process and final solution.**
  - You may follow the structure from the lab slides.
  - **Properly indicate the units of all numerical data**.
  - **Include labels, legends and titles/captions in all figures**.
  - There is no need to include theory, but properly introduce/reference all the formulas and models you use.
  - Include a ‘references’ section with a list of all the sources you consulted, and cite them in the text where appropriate.
  - Properly credit all images taken from other sources.

Project

Code

- The submission must include **two different folders** with the codes of each assignment:
  - **Assignment 1**: Folder named `Assignment1` containing:
    - `PlanetaryMission_group_N.m`: main script that reproduces your result.
    - `Assignment1\functions\`: subfolder with **all the other functions you developed**.
  - **Assignment 2**: Folder named `Assignment2` containing:
    - `InterplanetaryMission_group_N.m` : main script that reproduces your results.
    - `Assignment2\functions\`: subfolder with **all the other functions you developed.**
- No need to upload the functions we provide to you in WeBeep, unless you modified them.

Project

Code headers

- Each code file must include a **header** detailing:
  - Inputs and outputs (specify dimensions and units),
  - Authors,
  - Basic usage information

```
function dy = ode_2bp( t, y, muP )
%ode_2bp ODE system for the two-body problem (Keplerian motion)
%
% PROTOTYPE:
%   dy = ode_2bp( t, y, mu )
%
% INPUT:
%   t[1]        Time (can be omitted, as the system is autonomous)  [T]
%   y[6x1]      Cartesian state of the body ( rx, ry, rz, vx, vy, vz ) [ L, L/T ]
%   muP[1]       Gravitational parameter of the primary [L^3/T^2]
%
% OUTPUT:
%   dy[6x1]     Derivative of the state  [ L/T^2, L/T^3 ]
%
% CONTRIBUTORS:
%   Student 1
%   Student 2
%
% VERSIONS
%   2020-11-19: First version
%
```

Project

Auxiliar functions available in Beep

- For the assignments, you may use the auxiliar MATLAB functions **available in WeBeep**:
  - `astroConstants`: Use it to retrieve common astrodynamic constants (both assignments).
  - `lambertMR`: Use it for solving each Lambert arc (Assignment 1).
  - `uplanet`: Planets' ephemeris (**don't propagate the planets' orbits yourself**).
  - `ephMoon`: Analytical ephemeris of the Moon.
  - `ephNEO:` Ephemerides of several Near-Earth Objects.
  - **timeConversion.zip**: Compressed folder with several time conversion routines

Past year students

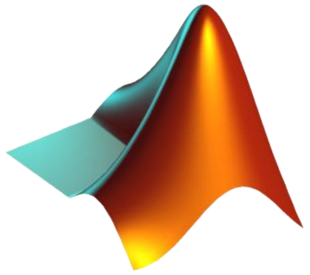
- Project marks of past years are valid if you:
  - presented a report,
  - performed an oral presentation, and
  - received a mark for the assignments.

- Past students who do not fulfil these 3 conditions (incomplete assignments) can either:
  1. Join a group and start the project anew this academic year.
  2. Complete their project from past year and deliver it with the same rules presented before. If several past students of the same project group want to follow this option, they have to maintain the group

MATLAB FUNDAMENTALS

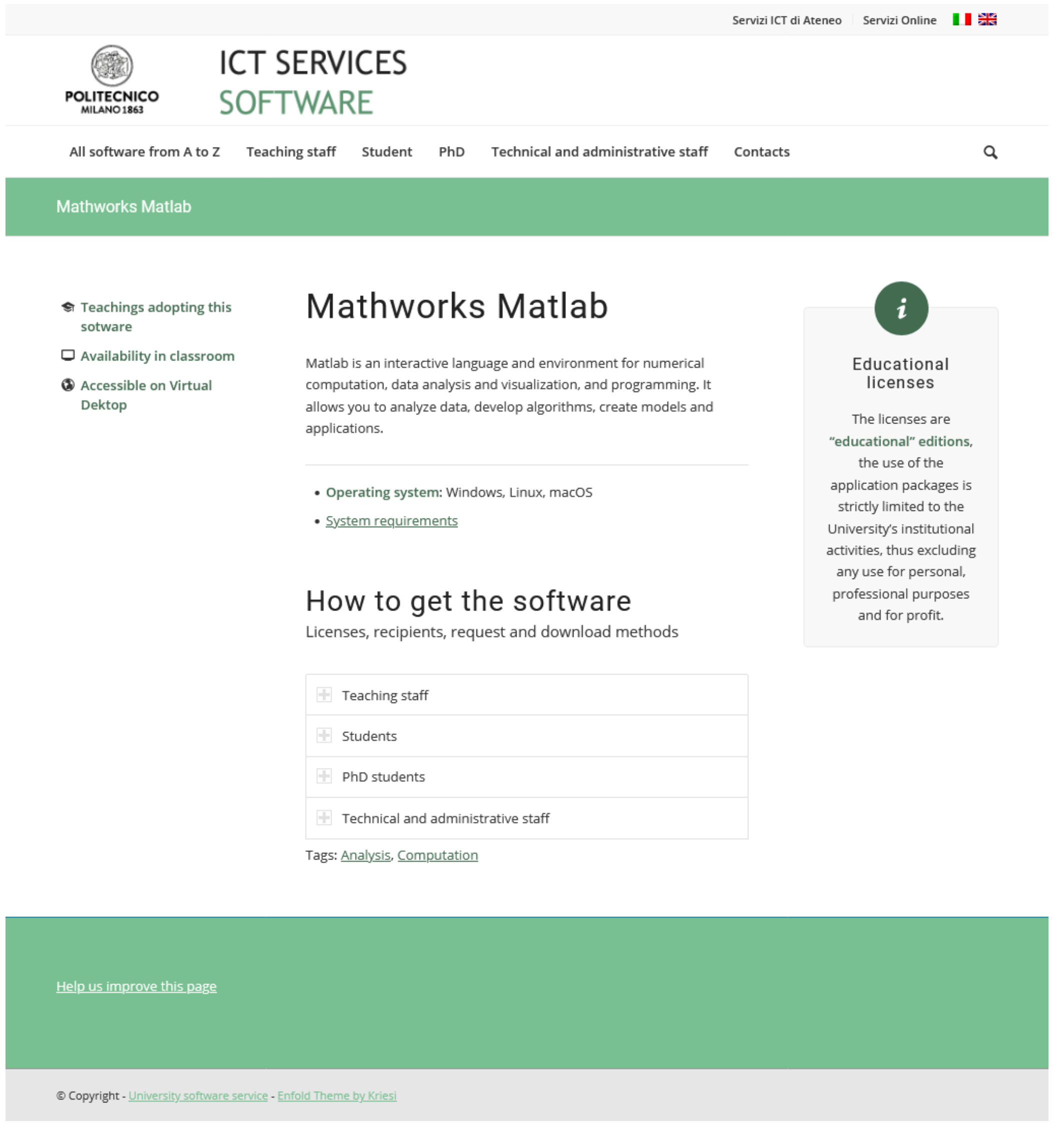
Before we begin…

- We will use **MATLAB** for the lab classes
  - You need to know the basics to follow the lab!



- **PoliMi** has a **Campus License**, giving access to the software and many resources:

  https://www.software.polimi.it/mathworks-matlab/



Servizi ICT di Ateneo Servizi Online

POLITECNICO MILANO 1863

ICT SERVICES
SOFTWARE

All software from A to Z | Teaching staff | Student | PhD | Technical and administrative staff | Contacts

Mathworks Matlab

- Teachings adopting this sotware
- Availability in classroom
- Accessible on Virtual Dektop

Mathworks Matlab

Matlab is an interactive language and environment for numerical computation, data analysis and visualization, and programming. It allows you to analyze data, develop algorithms, create models and applications.

- **Operating system:** Windows, Linux, macOS
- System requirements

How to get the software

Licenses, recipients, request and download methods

- Teaching staff
- Students
- PhD students
- Technical and administrative staff

Tags: Analysis, Computation

Educational licenses

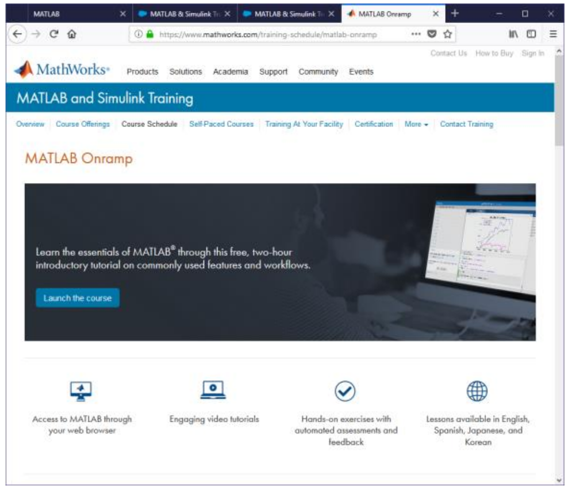
The licenses are **"educational" editions**, the use of the application packages is strictly limited to the University's institutional activities, thus excluding any use for personal, professional purposes and for profit.

Help us improve this page

© Copyright - University software service - Enfold Theme by Kriesi

MATLAB learning resources

Mathworks online courses



MATLAB and Simulink Training

MATLAB Onramp

Learn the essentials of MATLAB® through this free, two-hour introductory tutorial on commonly used features and workflows.

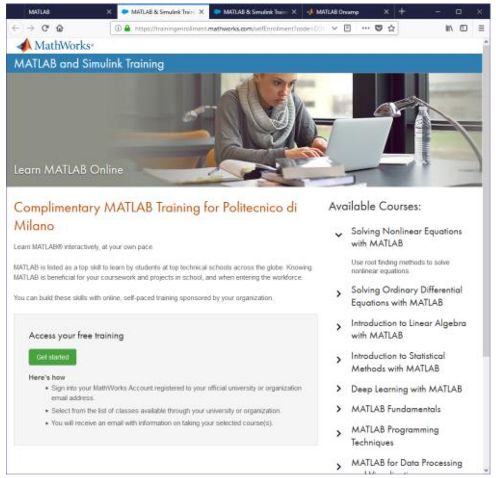
- Access to MATLAB through your web browser
- Engaging video tutorials
- Hands-on exercises with automated assessments and feedback
- Lessons available in English, Spanish, Japanese, and Korean

For those with little or no MATLAB experience, it is **strongly recommended** to follow the **MATLAB Onramp** online course (approx. 2 hours)

https://www.mathworks.com/training-schedule/matlab-onramp

Visit the courses in MATLAB Academy (included in the Campus license) to improve your MATLAB competencies

https://trainingenrollment.mathworks.com/selfEnrollment?code=D3M4TB3KDGIC



MATLAB and Simulink Training

Complimentary MATLAB Training for Politecnico di Milano

Learn MATLAB® interactively, at your own pace.

MATLAB is listed as a top skill to learn by students at top technical schools across the globe. Knowing MATLAB is beneficial for your coursework and projects in school, and when entering the workforce.

You can build these skills with online, self-paced training sponsored by your organization.

Access your free training

Here's how
- Sign into your MathWorks Account registered to your official university or organization email address.
- Select from the list of classes available through your university or organization.
- You will receive an email with information on taking your selected course(s).

Available Courses:
- Solving Nonlinear Equations with MATLAB
  Use root finding methods to solve nonlinear equations
- Solving Ordinary Differential Equations with MATLAB
- Introduction to Linear Algebra with MATLAB
- Introduction to Statistical Methods with MATLAB
- Deep Learning with MATLAB
- MATLAB Fundamentals
- MATLAB Programming Techniques
- MATLAB for Data Processing

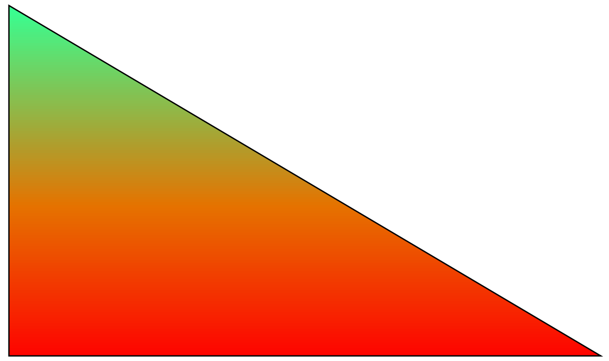
MATLAB learning resources

MATLAB functions: too many things to remember

MATLAB has:

- **Thousands** of included functions and toolsets
  - Most of them with multiple inputs and outputs
    - To be provided in a given way (can affect behaviour)
    - Many are optional, with pre-set default values

Effort to remember it all:



Impossible to remember/know them all!

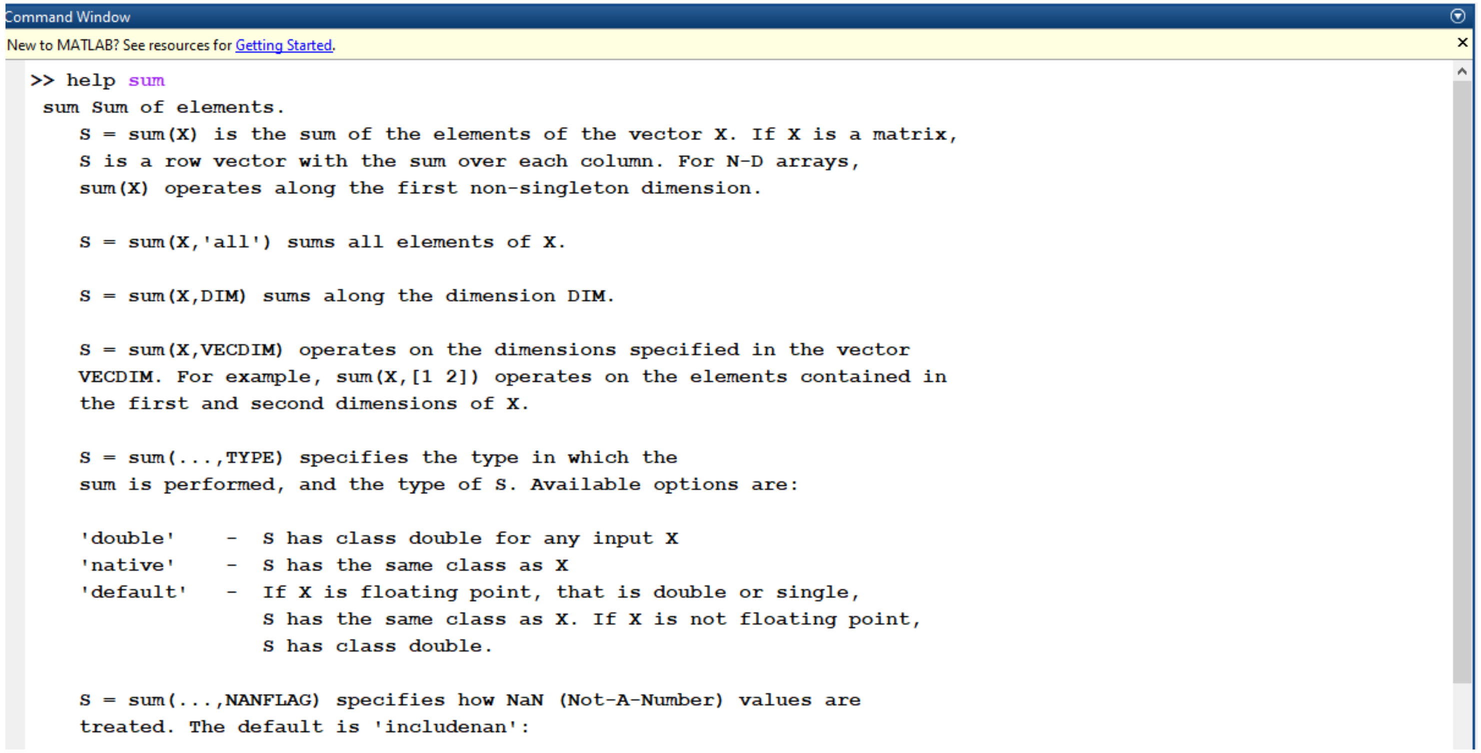
But it is easy remembering to check MATLAB's documentation (even the pros use it)

MATLAB learning resources

MATLAB's help command

MATLAB's `help` command is run from the command window, and provides usage information about a given function:

```
>> help name_of_function
```



Command Window

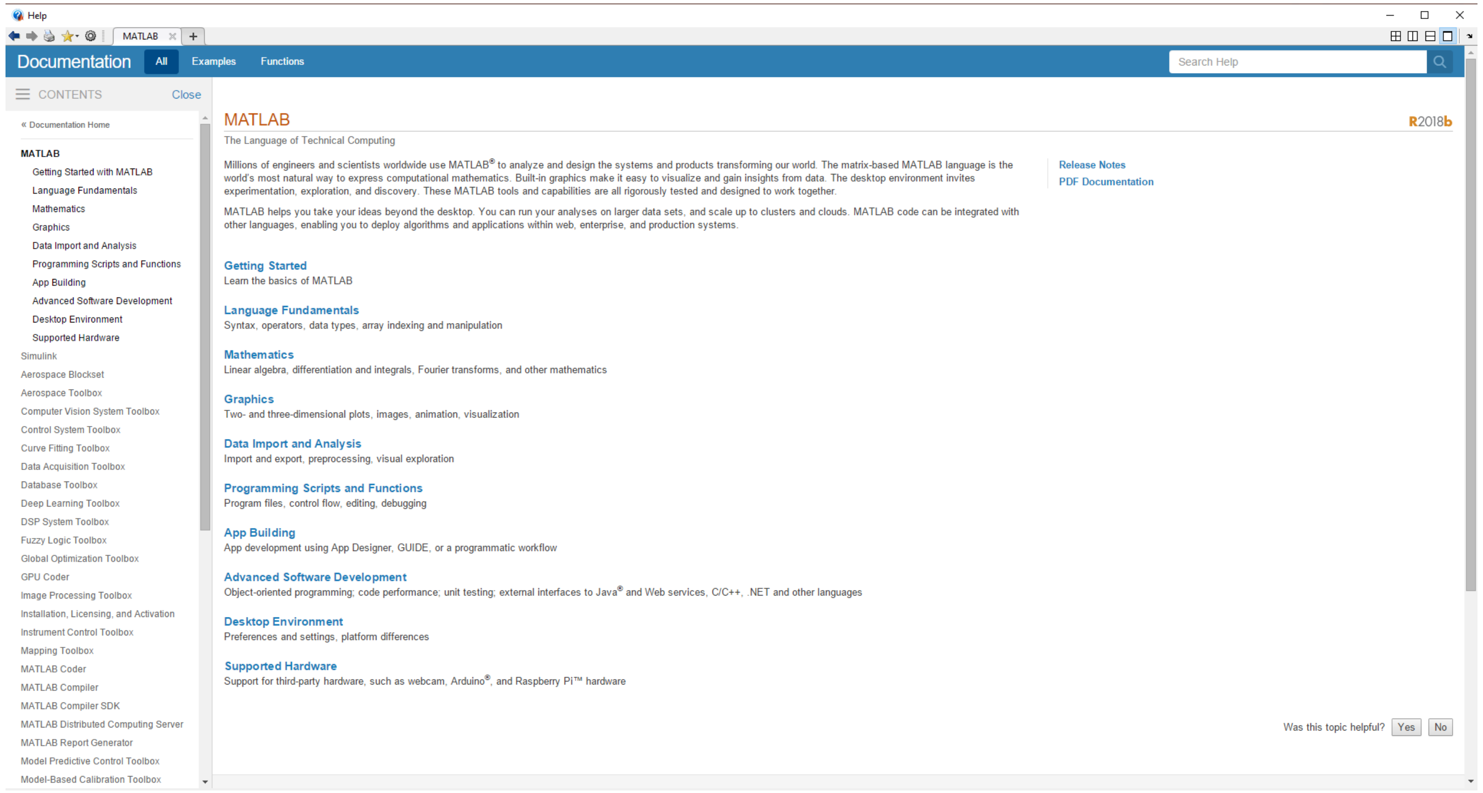
New to MATLAB? See resources for Getting Started.

```
>> help sum
 sum Sum of elements.
    S = sum(X) is the sum of the elements of the vector X. If X is a matrix,
    S is a row vector with the sum over each column. For N-D arrays,
    sum(X) operates along the first non-singleton dimension.

    S = sum(X,'all') sums all elements of X.

    S = sum(X,DIM) sums along the dimension DIM.

    S = sum(X,VECDIM) operates on the dimensions specified in the vector
    VECDIM. For example, sum(X,[1 2]) operates on the elements contained in
    the first and second dimensions of X.

    S = sum(...,TYPE) specifies the type in which the
    sum is performed, and the type of S. Available options are:

    'double'    -  S has class double for any input X
    'native'    -  S has the same class as X
    'default'   -  If X is floating point, that is double or single,
                   S has the same class as X. If X is not floating point,
                   S has class double.

    S = sum(...,NANFLAG) specifies how NaN (Not-A-Number) values are
    treated. The default is 'includenan':
```

MATLAB learning resources

MATLAB documentation center

- The **documentation center** provides:
  - **Detailed information on functions** (usage, inputs/outputs, examples, etc.)
  - **General and specific information about how to use MATLAB**
  - **Practical examples and tutorials**



« Documentation Home

MATLAB
- Getting Started with MATLAB
- Language Fundamentals
- Mathematics
- Graphics
- Data Import and Analysis
- Programming Scripts and Functions
- App Building
- Advanced Software Development
- Desktop Environment
- Supported Hardware

Simulink
Aerospace Blockset
Aerospace Toolbox
Computer Vision System Toolbox
Control System Toolbox
Curve Fitting Toolbox
Data Acquisition Toolbox
Database Toolbox
Deep Learning Toolbox
DSP System Toolbox
Fuzzy Logic Toolbox
Global Optimization Toolbox
GPU Coder
Image Processing Toolbox
Installation, Licensing, and Activation
Instrument Control Toolbox
Mapping Toolbox
MATLAB Coder
MATLAB Compiler
MATLAB Compiler SDK
MATLAB Distributed Computing Server
MATLAB Report Generator
Model Predictive Control Toolbox
Model-Based Calibration Toolbox

MATLAB R2018b

The Language of Technical Computing

Millions of engineers and scientists worldwide use MATLAB® to analyze and design the systems and products transforming our world. The matrix-based MATLAB language is the world's most natural way to express computational mathematics. Built-in graphics make it easy to visualize and gain insights from data. The desktop environment invites experimentation, exploration, and discovery. These MATLAB tools and capabilities are all rigorously tested and designed to work together.

MATLAB helps you take your ideas beyond the desktop. You can run your analyses on larger data sets, and scale up to clusters and clouds. MATLAB code can be integrated with other languages, enabling you to deploy algorithms and applications within web, enterprise, and production systems.

Release Notes
PDF Documentation

Getting Started
Learn the basics of MATLAB

Language Fundamentals
Syntax, operators, data types, array indexing and manipulation

Mathematics
Linear algebra, differentiation and integrals, Fourier transforms, and other mathematics

Graphics
Two- and three-dimensional plots, images, animation, visualization

Data Import and Analysis
Import and export, preprocessing, visual exploration

Programming Scripts and Functions
Program files, control flow, editing, debugging

App Building
App development using App Designer, GUIDE, or a programmatic workflow

Advanced Software Development
Object-oriented programming; code performance; unit testing; external interfaces to Java® and Web services, C/C++, .NET and other languages

Desktop Environment
Preferences and settings, platform differences

Supported Hardware
Support for third-party hardware, such as webcam, Arduino®, and Raspberry Pi™ hardware

Was this topic helpful? Yes No

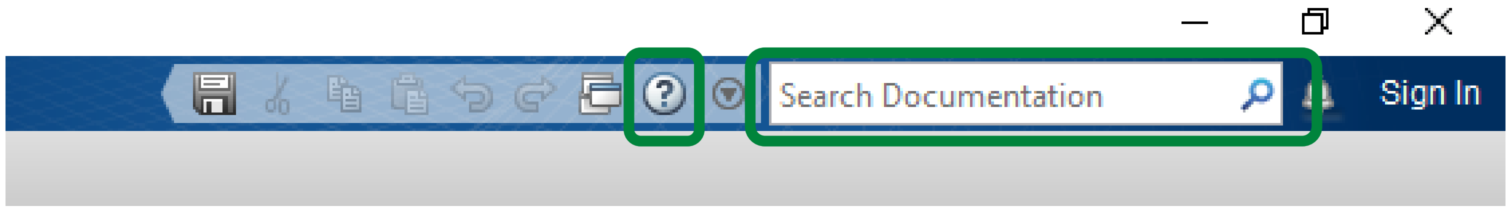
MATLAB learning resources

MATLAB documentation center

- You can access the **documentation center** in several ways:
  - Running command `doc` from the command window
    - If you use command `doc name_of_function`, you will go directly to the documentation page for that function

```
>> doc
>> doc name_of_function
```

  - Using the **help button** or the **Search Documentation** box in the toolbar:

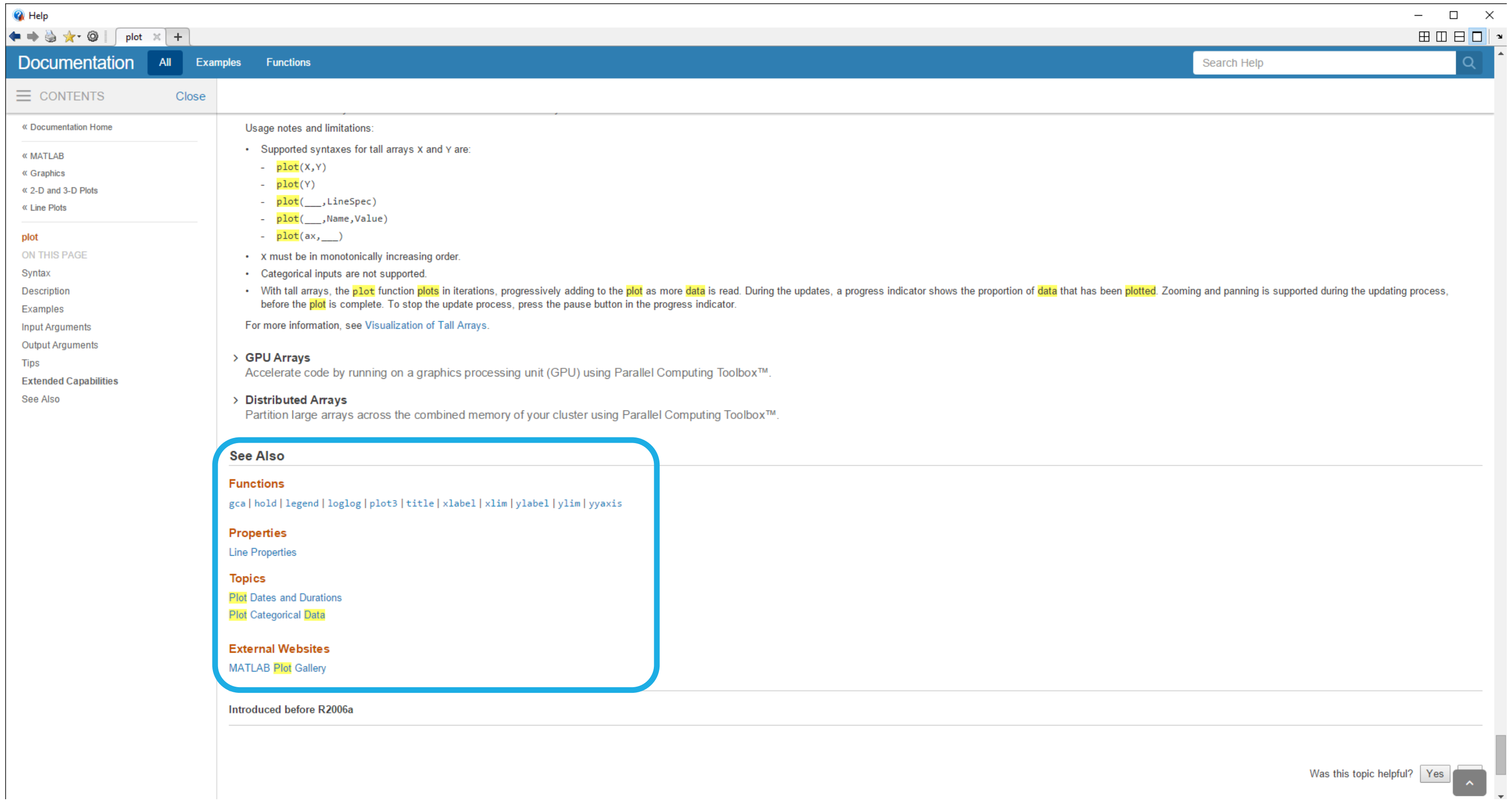


  - Pressing **F1** on your keyboard

MATLAB learning resources

MATLAB documentation center

- Many times MATLAB already has the functionality you need:
  - Search the **documentation center** before implementing new functions
  - At the end of each documentation page, you will find a list of **related functions and topics** that can be very handy



Usage notes and limitations:

- Supported syntaxes for tall arrays `X` and `Y` are:
  - `plot(X,Y)`
  - `plot(Y)`
  - `plot(___,LineSpec)`
  - `plot(___,Name,Value)`
  - `plot(ax,___)`
- `X` must be in monotonically increasing order.
- Categorical inputs are not supported.
- With tall arrays, the `plot` function plots in iterations, progressively adding to the plot as more data is read. During the updates, a progress indicator shows the proportion of data that has been plotted. Zooming and panning is supported during the updating process, before the plot is complete. To stop the update process, press the pause button in the progress indicator.

For more information, see Visualization of Tall Arrays.

› **GPU Arrays**
Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

› **Distributed Arrays**
Partition large arrays across the combined memory of your cluster using Parallel Computing Toolbox™.

See Also

Functions
`gca` | `hold` | `legend` | `loglog` | `plot3` | `title` | `xlabel` | `xlim` | `ylabel` | `ylim` | `yyaxis`

Properties
Line Properties

Topics
Plot Dates and Durations
Plot Categorical Data

External Websites
MATLAB Plot Gallery

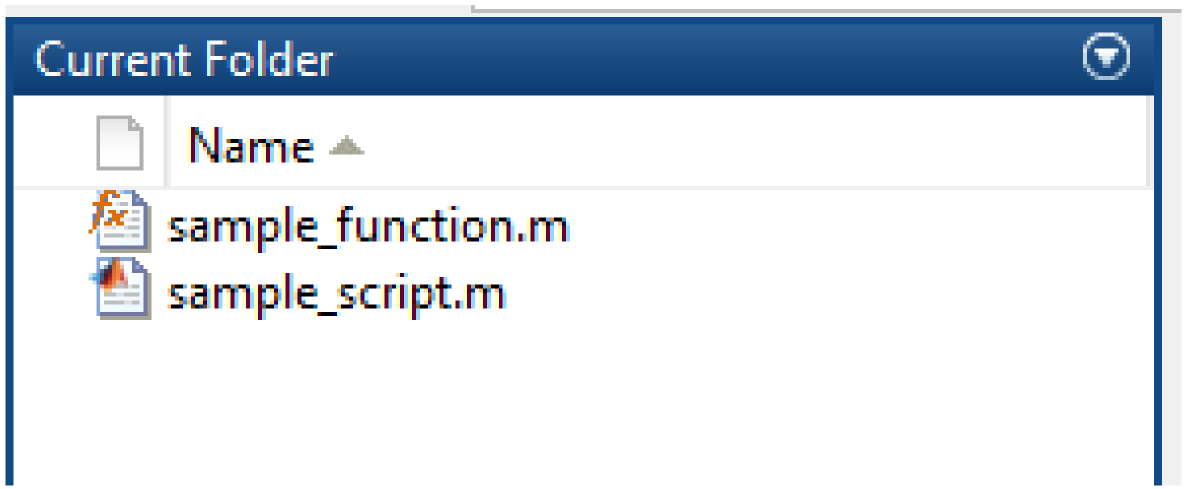
Introduced before R2006a

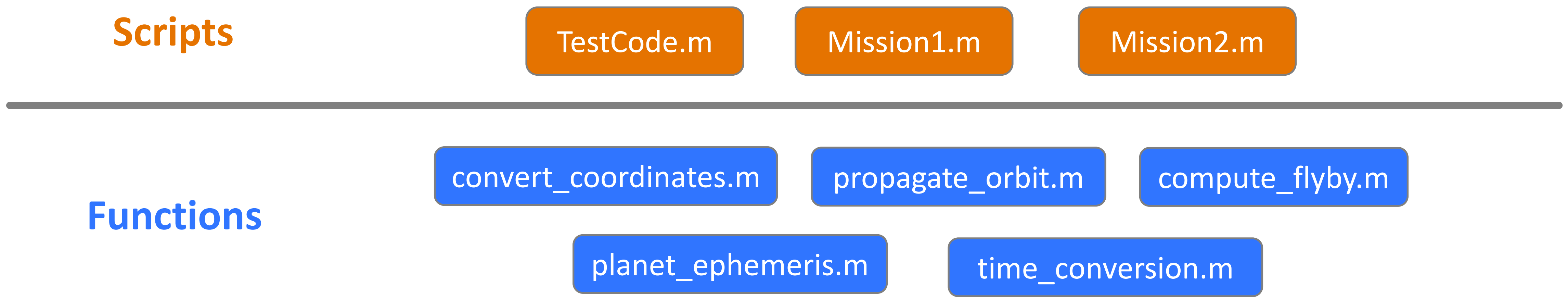
Was this topic helpful? Yes

Code structuring

Scripts and functions

- MATLAB code is organized in **scripts** and **functions**
  - Both have file extension `.m`
  - They get different icons in MATLAB file explorer
  - **Scripts** should be the top-level part of your code, where you organize the whole program.
  - **Functions** should correspond to specific tasks.



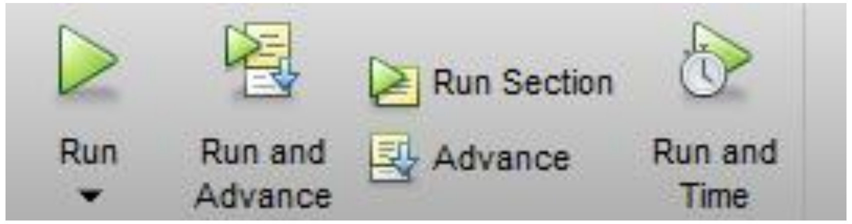


Code structuring

Scripts

Scripts are the top-level files of your code. They can be executed directly:

- From the **editor**, using the toolbar or the keyboard

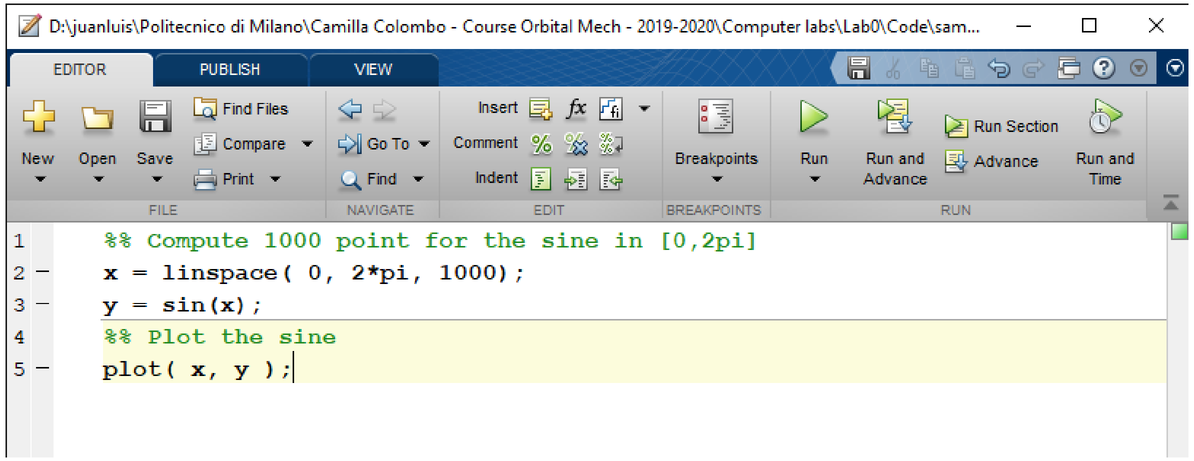


Keep your cursor over a button in the toolbar to see the keyboard shortcut

- From the command window, typing the name of the script (without file extension)

```
>> sample_script
```

- You can also define **sections**, to execute your scripts part by part
  - New sections are created typing **%%**
  - The current section is the one containing the cursor, and is highlighted in yellow



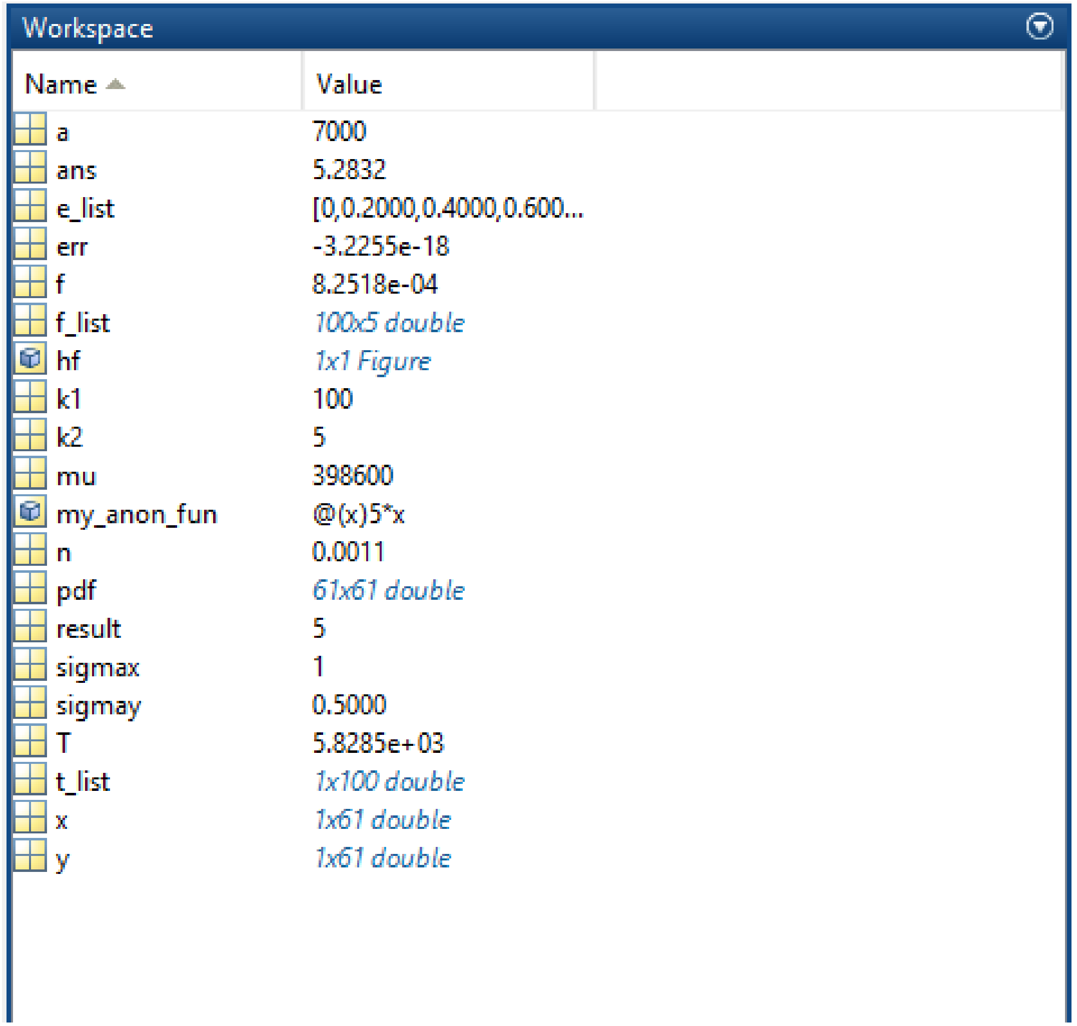
```
%% Compute 1000 point for the sine in [0,2pi]
x = linspace( 0, 2*pi, 1000);
y = sin(x);
%% Plot the sine
plot( x, y );
```

Code structuring

Scripts

- **Scripts** use the **global memory workspace**:
  - They can access any variable previously defined in the workspace. This includes variables created by other scripts or in the command line
  - They can add new variables to the workspace

- This is very dangerous when calling scripts from scripts. Because they share the workspace, if they accidentally use the same name for different variables, they will overwrite each other:
  - Very difficult to debug
  - You have better control calling **functions** instead



Workspace

Name	Value
a	7000
ans	5.2832
e_list	[0,0.2000,0.4000,0.600...
err	-3.2255e-18
f	8.2518e-04
f_list	*100x5 double*
hf	*1x1 Figure*
k1	100
k2	5
mu	398600
my_anon_fun	@(x)5*x
n	0.0011
pdf	*61x61 double*
result	5
sigmax	1
sigmay	0.5000
T	5.8285e+03
t_list	*1x100 double*
x	*1x61 double*
y	*1x61 double*

Code structuring

Functions

- **Functions** can be called from the **command window**, **scripts**, or **other functions**:

```
>> [ouput1,output2] = sample_function(input1,input2)
```

- **Functions** should focus on **performing specific tasks**:
  - Breaking code into small pieces makes it easier to **test**, **develop** and **maintain**
  - Try making them as general as possible, so that they can be **reused**
- **Functions** have their own **memory workspace**:
  - They don't "see" variables created outside of their workspace, except for those defined as **global** (strongly discouraged)
  - Data exchange is controlled through **input/output variables**
  - Their internal variable names will not conflict with other scripts/functions
  - All internal variables are erased when the function ends, unless they are defined as **persistent** (be careful when using them)

Code structuring

Functions

```
function n = mean_motion( a, mu )
% mean_motion Mean motion of a Keplerian orbit
%
% Function to compute the mean motion of an orbit. This is a very simple
% example to put in the slides of the lab
%
% PROTOTYPE
%   n = mean_motion( a, mu )
%
% INPUT:
%   a [1]        Semimajor axis [L]
%   mu [1]       Gravitational parameter of the primary [L^3/T^2]
%
% OUTPUT:
%   n [1]        Orbit mean motion [1/T]
%
% CONTRIBUTORS:
%   Juan Luis Gonzalo Gomez
%
% VERSIONS
%   2018-09-26: First version
%

n = sqrt( mu/a^3 );

end
```

Code structuring

Anonymous functions

- An **anonymous function** is a function that is not written in a separate file, but defined directly as a variable inside another function/script
  - They can have several inputs and outputs, as a normal **function**
  - But they can only contain a single executable statement
  - You can define them inside **scripts**, **functions**, or in the command line

- **Anonymous functions** are created as follows:

```
vector_norm = @(x,y,z) sqrt( x.*x + y.*y + z.*z );
```

Name of the anonymous function

List of inputs (don't forget the `@` at the beginning)

Executable statement

- **Anonymous functions** are called as any normal **function**

Code structuring

Anonymous functions

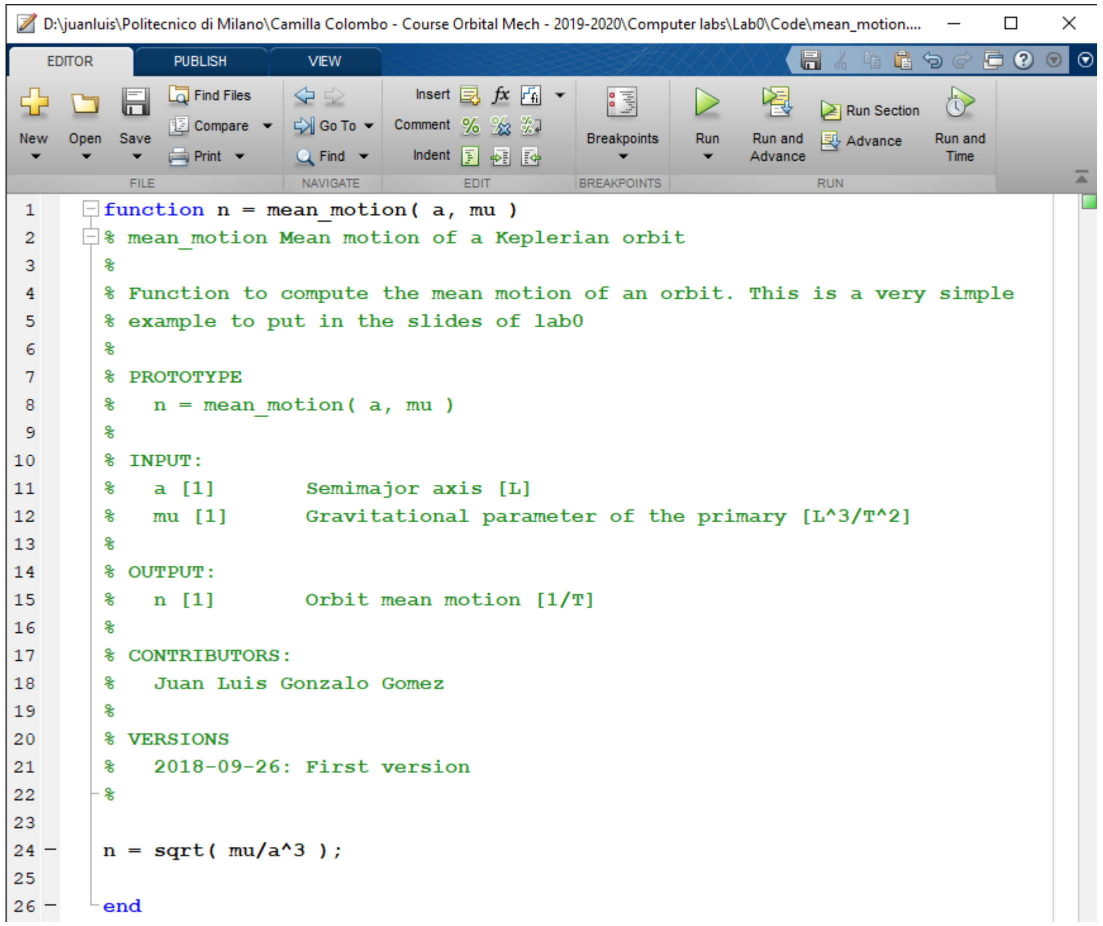
- **Anonymous functions** are normally used to:
  - Define small auxiliary functions within a **function** or **script**, without the need of creating a new file
  - Adapt the **interface** (inputs/outputs) of another function. Example:

```
% MATLAB's 'integral' function can be used to perform the
% numeric integral of a 1D function. 'integral' expects
% a function with a single input variable.
% I want to perform the integral of function my_fun
% between 0 and 1, but it has 2 inputs (the second one
% is a parameter, of value 5 for this example).
% I can solve this using an anonymous function:

my_anon_fun = @(x) my_fun( x, 5 );
result = integral( my_anon_fun, 0, 1 );
```

Best coding practices

Documentation

- **Document your code!**
  - Software is normally developed by **teams**; others must be able to understand/use your code
  - Not just for others, also **for the future you**
  - Use easily identifiable names for functions and variables
  - Include a header with:
    - Usage of the function
    - List of inputs and outputs (with physical units)
    - Authors and dates
    - Version log
    - External dependencies



```matlab
function n = mean_motion( a, mu )
% mean_motion Mean motion of a Keplerian orbit
%
% Function to compute the mean motion of an orbit. This is a very simple
% example to put in the slides of lab0
%
% PROTOTYPE
%   n = mean_motion( a, mu )
%
% INPUT:
%   a [1]       Semimajor axis [L]
%   mu [1]      Gravitational parameter of the primary [L^3/T^2]
%
% OUTPUT:
%   n [1]       Orbit mean motion [1/T]
%
% CONTRIBUTORS:
%   Juan Luis Gonzalo Gomez
%
% VERSIONS
%   2018-09-26: First version
%

n = sqrt( mu/a^3 );

end
```

Comments in MATLAB begin with symbol %

They can begin at any point of the line (e.g., after a command)

Best coding practices

Testing

- **Test your code as you go!**
  - No large code is bug-free (empirically demonstrated)
  - **Unit testing**: test each function independently as you program them
    - If you only debug the whole program at the end, it is very difficult to identify the source of each problem
  - Breaking up your code into small **functions** eases validation
  - One great way of validation is to check if the code reproduces **known results** (analytic solutions for particular cases, conservation principles, etc.)

MATLAB includes very useful **debugging tools**:

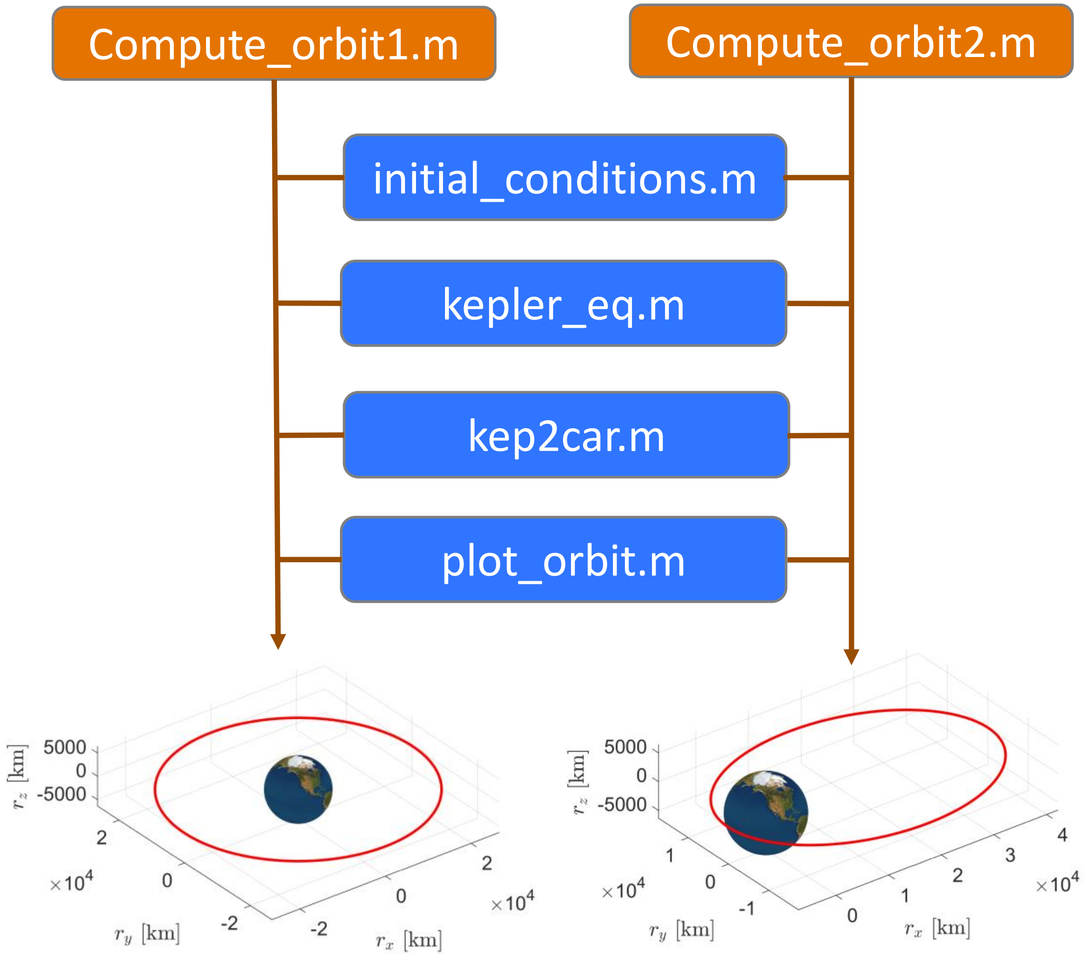
- Pause the execution and advance line by line,
- Check the memory workspace of each function,
- And much more.

Read the documentation page **"Debug a MATLAB Program"** to learn more!

Best coding practices

Reusability

- **Reuse as much as possible!**
  - Break your code into small, single-task **functions** with general inputs, so that they can be reused
  - This is not the same as copy-pasting blocks of code
  - This eases development and testing



By writing small, task-specific, reusable **functions** you will have less code to write, test and maintain

Figures

Are they really important?

- **Figures** (plots, graphs, charts, etc.) are a key component of scientific and technical communication
  - Good figures convey a lot of information in a clear and concise way
- A good figure should:
  - Have its axes clearly labelled, including physical units
  - Include a caption, placed below the figure, presenting its contents
  - If more than one data set is represented, include a legend and use different colours and markers/line styles
  - Use a font size clearly readable
  - Be self-contained (i.e. should be understandable even without reading the whole document)

⚠ Badly presented figures in the reports will be harshly punished

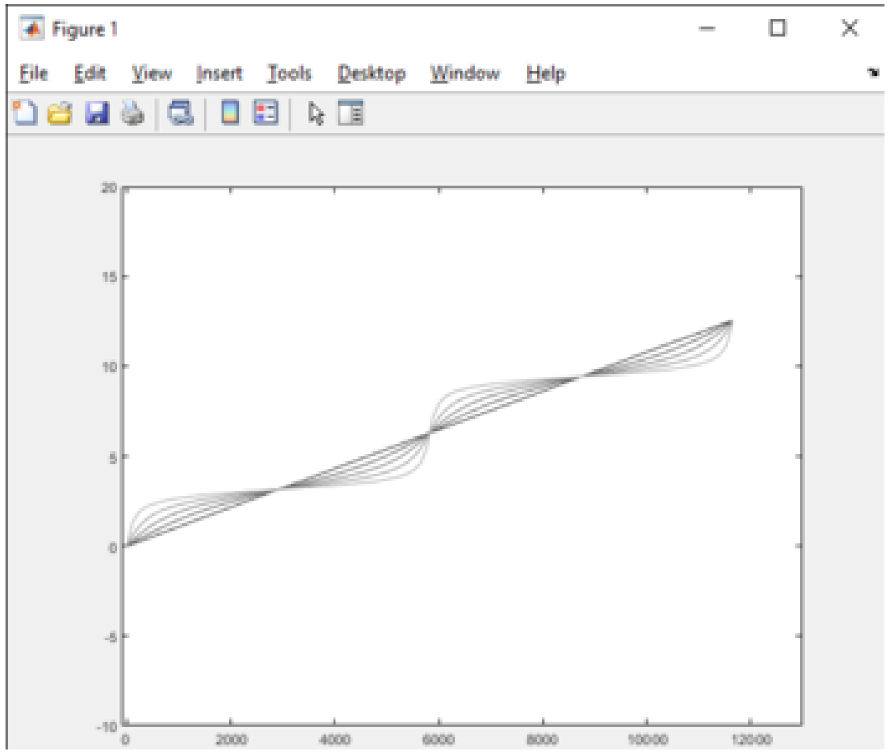
Figures

This is a BAD figure

It is just a low-resolution screenshot

Font is too small to read

I don't know what the axes represents, or their physical units



Very difficult to distinguish the lines

I don't know the meaning of each line

Plenty of wasted blank space

Without a title or a caption, I cannot know what is being represented in the plot

Figures

This is a GOOD figure

Good-quality image

Font size is similar to the text of the slide

The variable represented in each axis is indicated, along with its units

The unit of time (orbital period) is chosen to get numerical values easy to understand

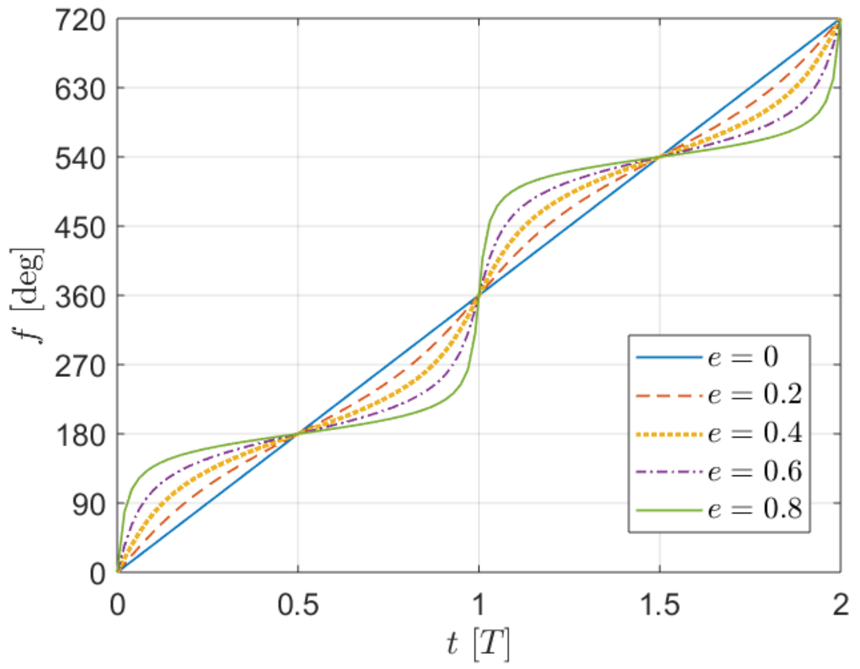


Figure 1. Evolution of true anomaly with time for several orbits with $a = 7000$ km, $\mu = 398600\ \mathrm{km}^3/\mathrm{s}^2$, and different e

Lines are set apart using different colors and line styles

A legend is included

A grid is included to improve readability

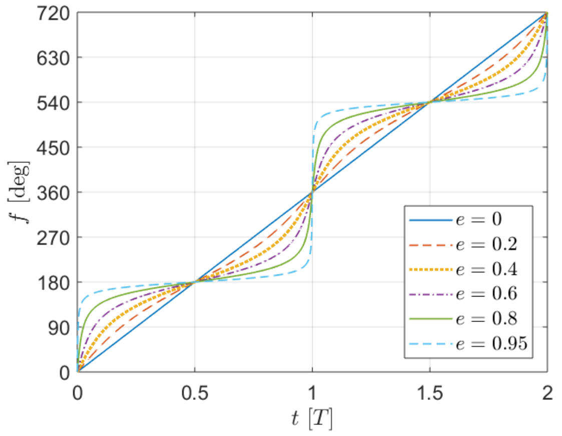
Ranges are set to make use of all available space

Figure contents are clearly introduced in a caption, together with a number to reference the figure

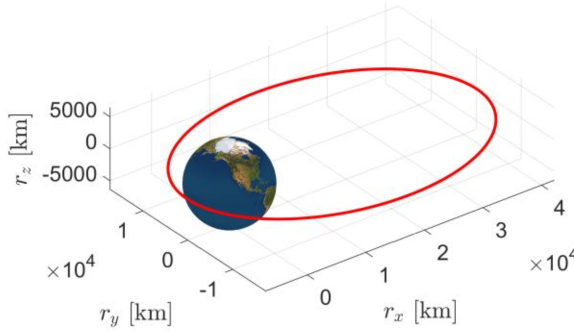
Figures

Plotting in MATLAB

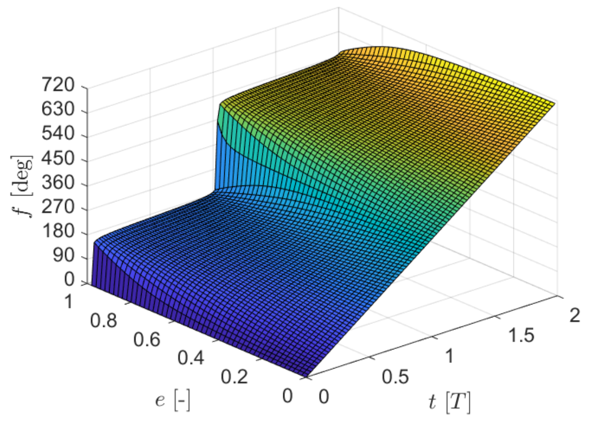
- MATLAB can represent many types of figures (check the **documentation center** to learn how to use them)



`plot`



`plot3`



`surface`

- Other types of MATLAB plots:
  - `quiver` and `quiver3`: 2D and 3D vector field plots
  - `comet` and `comet3`: animations in 2D and 3D plots
  - `sphere`: Generates a sphere
  - `mesh`: 3D surface plots

Figures

Customizing plots in MATLAB

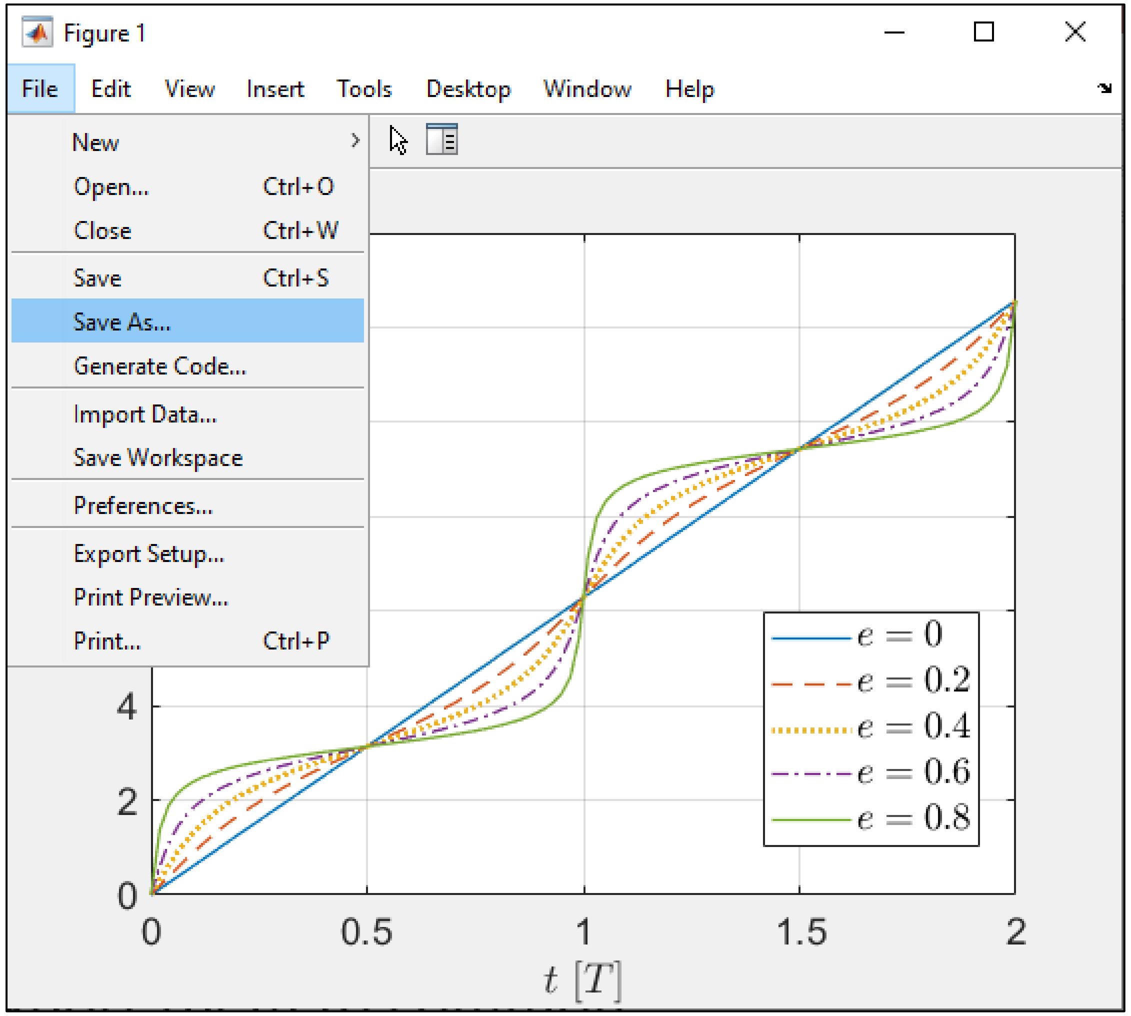
- Properties of plots can be adjusted from the graphical interface or using commands:
  - `xlabel`, `ylabel`, and `zlabel`: Set labels for x, y, and z axes
  - `xlim`, `ylim`, and `zlim`: Set the plotting range for each axis
  - `title`: Set the title of the plot
  - `legend`: Add a legend to the plot
  - `grid on`: Add a grid to the plot
  - `hold on`: Hold the plot, so new lines can be added
  - `axis equal`: Use equal unit length along all axes
  - and many more

Check the documentation page for each kind of plot to discover all the available options and related commands

Figures

Saving figures in MATLAB

- MATLAB uses its own format (`.fig`) to save figures
  - You can open fig files in MATLAB to edit them
- You can export figures to common formats (png, jpg, eps, etc), to use them in the report and presentation
  - Menu “File > Save as…”
  - MATLAB cannot edit these formats after exporting
- You can also save figures from code using commands such as `saveas` and `print`



Screenshots are not a good way to include figures in the report or presentation