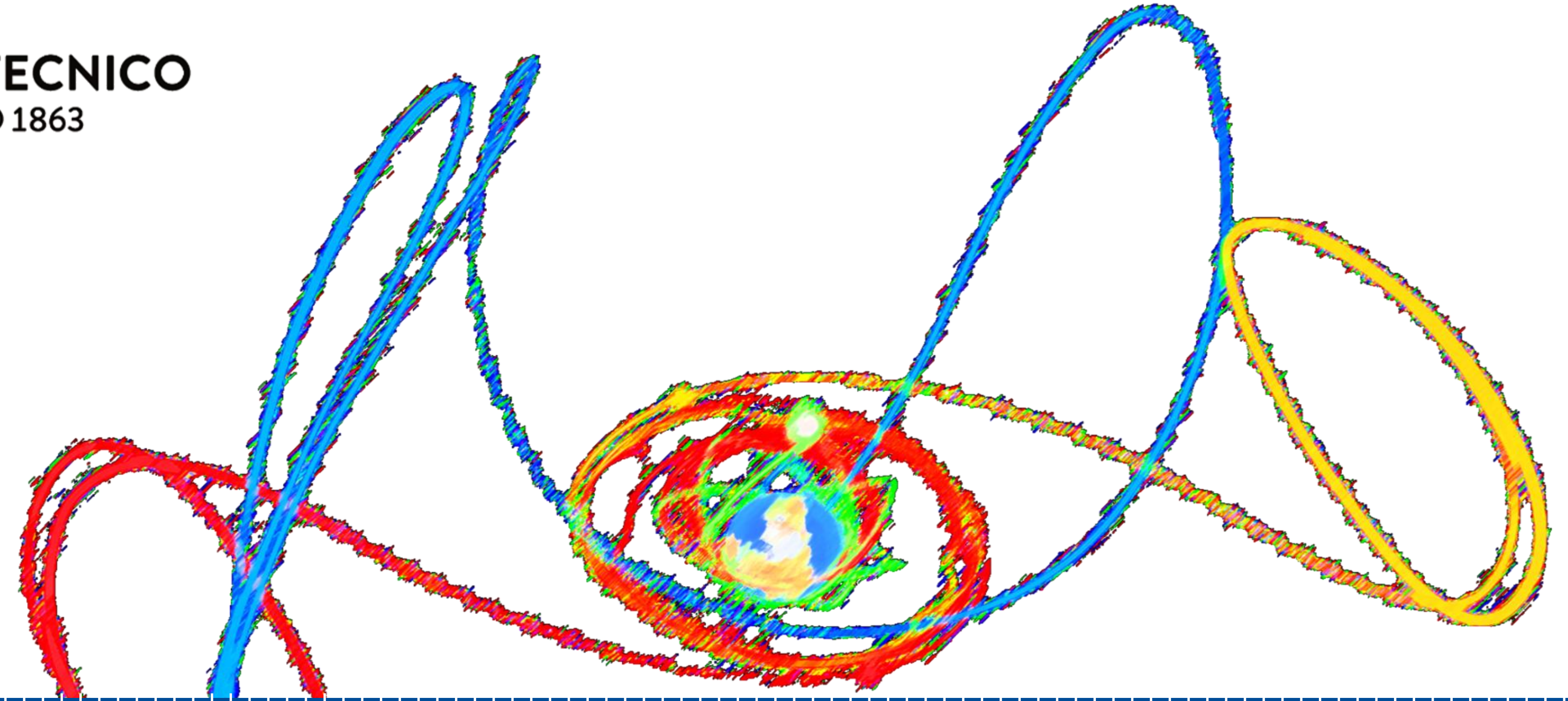


Orbital Mechanics
Lab Chapter 0: Introduction and MATLAB fundamentals

Andrea MUCIACCIA, Mathilda BOLIS, Francesca OTTOBONI, Giacomo BORELLI, Juan Luis GONZALO GOMEZ, Camilla COLOMBO

Department of Aerospace Science and Technology, Politecnico di Milano

Academic year 2025/2026

Chapter Contents

Lab – Introduction and MATLAB fundamentals

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

Teaching staff

Contacts

Very important: For communications about the course do not email directly to the personal address of each member of the teaching staff, use instead the common email:

orbmech-daer@polimi.it

- Camilla **Colombo**
  - Meeting time: please arrange via email
  - Tel: 8352
- Andrea **Muciaccia**
  - Meeting time: please arrange via email
  - Tel: 8401
- Francesca **Ottoboni**
  - Meeting time: please arrange via email
  - Tel: 8401
- Giacomo **Borelli**
  - Meeting time: please arrange via email
  - Tel: 8401
- Mathilda **Bolis**
  - Meeting time: please arrange via email
  - Tel: 8401

Teaching staff

Andrea Muciaccia

2019: BEng, Politecnico di Milano, Aerospace engineering

2021: MEng, Politecnico di Milano, Space engineering

2021-2022: Research Fellow, Politecnico di Milano

- Definition of metric to assess the impact of mission on the space environment

2022-2025: PhD candidate, Politecnico di Milano

- Investigation of the space capacity and analysis of metric to assess the impact of mission on the space environment

2025-present: Post-doctoral research fellow, Politecnico di Milano

Teaching staff

Mathilda Bolis

2020: BEng, Politecnico di Milano, Aerospace Engineering

2022: MEng, Politecnico di Milano, Space engineering

Apr.2023-Dec.2023: Research fellow, Politecnico di Milano

- Mission analysis for the REMEC mission, design of transfer trajectories in the three-body problem of the Earth-Moon and the Sun-Earth system

Dec.2023-present: PhD candidate, Politecnico di Milano

- End-of-life disposal design for missions in cislunar space, including topics such as chaotic dynamical systems, spacecraft disposal, deep-space astrodynamics, and space situational awareness.

Teaching staff

Francesca Ottoboni

2021: BEng, Politecnico di Milano, Aerospace Engineering

2023: MEng, Politecnico di Milano, Space engineering

Dec.2023-present: PhD candidate, Politecnico di Milano

- Short and long-term techniques for in-orbit fragmentation reconstruction and debris cloud evolution analysis.

Lab groups

Rules

- The lab project will be carried out in **teams**, with the **rules** already presented in Introductory lecture by Prof. Camilla Colombo → refer to that lecture and slides for the rules.

- You can **register** your team using the dedicated **activity in WeBeep (section “Project lab”).** Deadline for group registration is **20th October 2025.**
    - To support you in creating team, there is a dedicated Forum in WeBeep (section “Project lab”).
    - The teams could be the same for the course of Spacecraft Attitude Dynamics.

- As explained in the introductory lecture by Prof. Camilla Colombo **submitting the assignment is a prerequisite to attend the final exam**. If you do not intend to do the assignment during this semester, avoid teaming up with people that do.

Lab groups

Teamwork

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

- Members of the group must **cooperate,** but **everyone is responsible for all the work done in the project**.
- You can distribute the work among the team but is better if the code is done by each member and you use the team to validate each other code.
- Make decisions towards design solutions based on numerical/analytical/physical evidence and analyses: you must always be able to **motivate your design choices**.
- During the oral presentation**, any team member can be questioned about any part of the work**.
- The questions during the oral exam are personal, they can be on the assignment lab, the code but also the theory explained in class.

Project

Contents

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

- Project evaluation includes:
  - **Deliverables** (1 submission per group)
    - **Project report**: A single PDF report on both assignments, of maximum **15 pages (total, all page included)**
    - **Simulation codes and results**
    - **Numerical results**, to be submitted via a Form
  - **Oral exam**
    - **All team at the same time of the exam period but personal oral exam**
    - Personal questions about the assignments (project report and codes) and the theory of the course
    - Any student can be questioned about any part of the work.
  - **Peer evaluation**

Project

Submission procedure and deadlines

- The deliverables must be submitted through **WeBeep**
  - Submit a **single ZIP file** with **report, slides, and code**, named "`OrbitalMech_group_nnnn.zip`", where `nnnn` is the group ID (e.g., `OrbitalMech_group_2542.zip`).
  - WeBeep **submission file limit is 250 MB**. Larger submissions sizes are not allowed (nor needed).
  - **Numerical results for specific questions** and **peer review** submitted via forms in WeBeep.
  - Submissions via any other means will not be considered.
  - Changes to the deliverables after the deadline will not be considered.
  - **Wrong deliverables will be penalised with -1 points**
- Deadlines:
  - **Deliverables must be submitted by 7 January 2026**.
    - Delivering the project is a must condition for the oral presentation and attending the written exam.
    - The delivery activity in WeBeep closes automatically on 7 January at 23:59.
  - **Oral presentation**
    - Dates will be available during the Winter, Summer and September exam sessions. They will be notified at the beginning of each session.
    - To be done before or within the same exam session (Winter/Summer/Autumn) when the written is done.

Project

Report

- **Single PDF of maximum 15 pages (all pages included including cover and appendices)**
  - Both assignments in the same report.
- 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 models you use.
  - Include a ‘references’ section with a list of all the sources you consulted and cite them in the text where appropriate including lecture notes.
  - Copying sections of the lecture notes and slides in the report is **plagiarism**.
  - Properly credit all images taken from other sources.

Project

Code

- The codes for both assignments must be included inside a folder named `Code`, with **two separate subfolders for each assignment** as follows:
  - **Assignment 1**: Subfolder `Code\Assignment1\` containing:
    - `InterplanetaryMission_group_N.m`: main script that reproduces your results (`N` is the group ID).
    - `Code\Assignment1\functions\`: subfolder with **all the other functions you developed** for the first assignment.
  - **Assignment 2**: Subfolder `Code\Assignment2\` containing:
    - `PlanetaryMission_group_N.m`: main script that reproduces your results (`N` is the group ID).
    - `Code\Assignment2\functions\`: subfolder with **all the other functions you developed** for the second assignment.
- 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 WeBeep

- 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 (**do not 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 exam, and
  - received a mark for the assignments.

- Past students can check the rules for past year project in the Introductory slide given by Prof. Camilla Colombo.

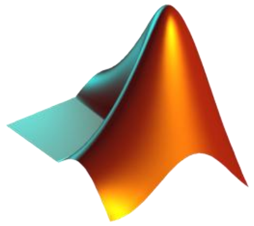
POLITECNICO
MILANO 1863

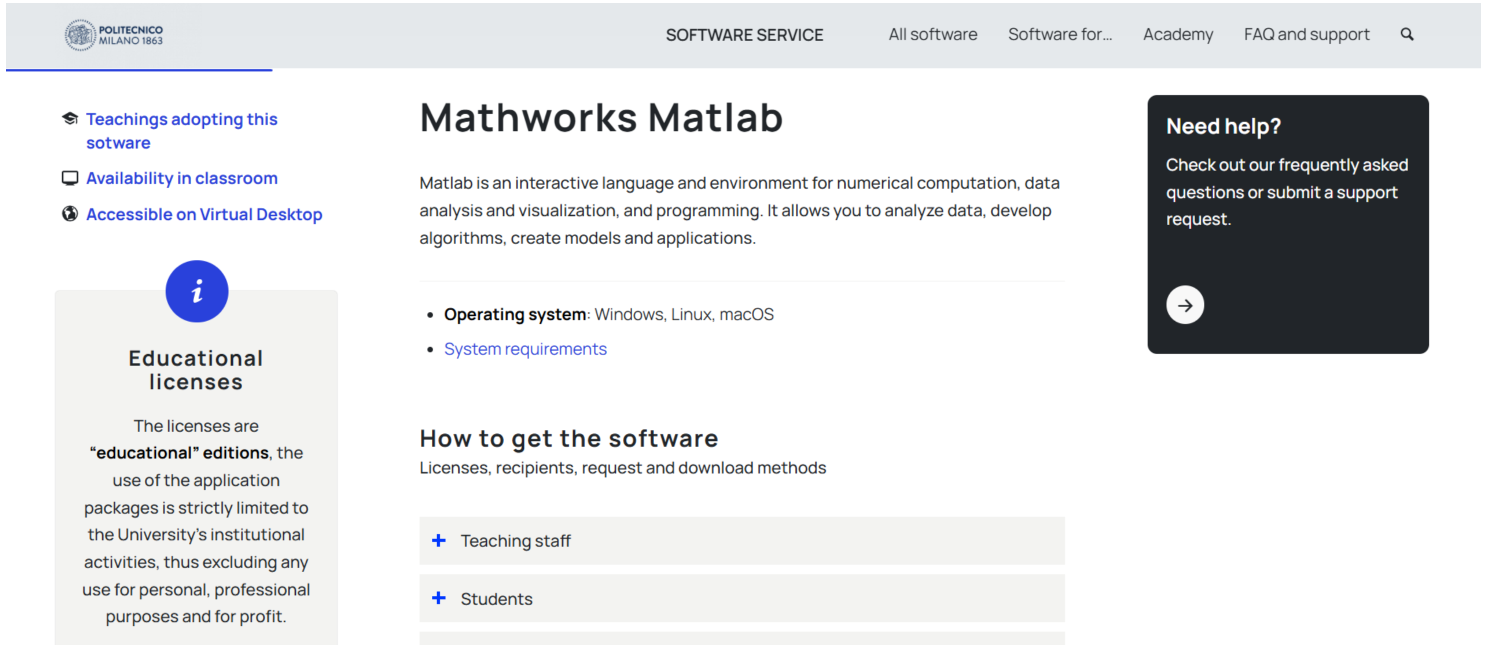
MATLAB FUNDAMENTALS

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. You need to create a Mathworks account following the instructions in:

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





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

Need help?

Check out our frequently asked questions or submit a support request.

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.

How to get the software

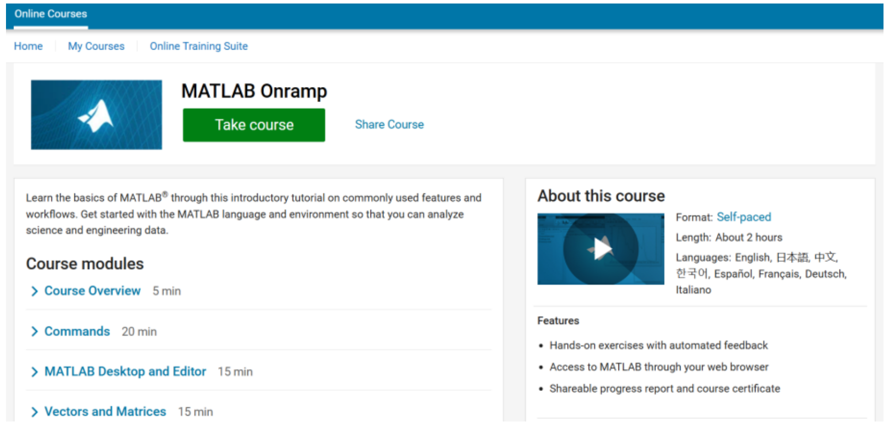
Licenses, recipients, request and download methods

+ Teaching staff

+ Students

MATLAB learning resources

Mathworks online courses


Online Courses

Home | My Courses | Online Training Suite

MATLAB Onramp

Take course | Share Course

Learn the basics of MATLAB® through this introductory tutorial on commonly used features and workflows. Get started with the MATLAB language and environment so that you can analyze science and engineering data.

Course modules

- Course Overview 5 min
- Commands 20 min
- MATLAB Desktop and Editor 15 min
- Vectors and Matrices 15 min

About this course

Format: Self-paced
Length: About 2 hours
Languages: English, 日本語, 中文, 한국어, Español, Français, Deutsch, Italiano

Features

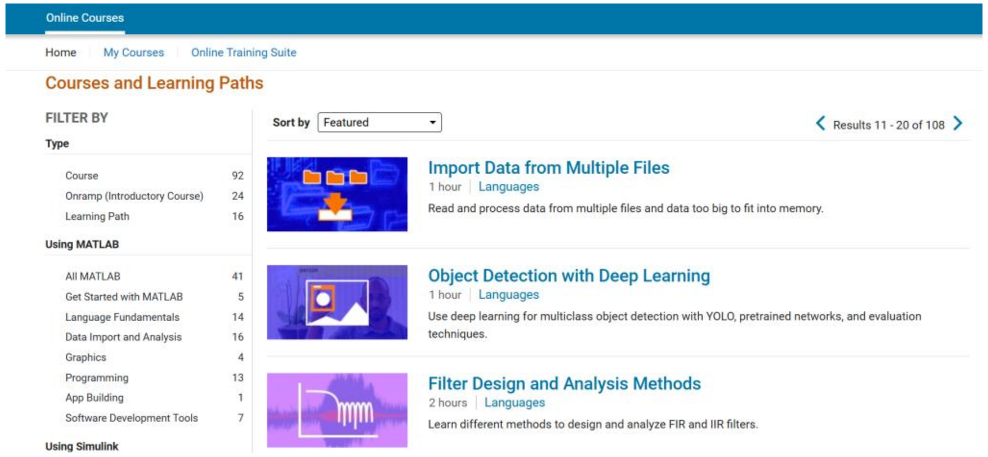
- Hands-on exercises with automated feedback
- Access to MATLAB through your web browser
- Shareable progress report and course certificate

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 (remember to log in with your Mathworks account) to improve your MATLAB competencies

https://matlabacademy.mathworks.com/


Online Courses

Home | My Courses | Online Training Suite

Courses and Learning Paths

FILTER BY

Type

Course	92
Onramp (Introductory Course)	24
Learning Path	16

Using MATLAB

All MATLAB	41
Get Started with MATLAB	5
Language Fundamentals	14
Data Import and Analysis	16
Graphics	4
Programming	13
App Building	1
Software Development Tools	7

Using Simulink

Sort by Featured — Results 11 - 20 of 108

Import Data from Multiple Files — 1 hour | Languages
Read and process data from multiple files and data too big to fit into memory.

Object Detection with Deep Learning — 1 hour | Languages
Use deep learning for multiclass object detection with YOLO, pretrained networks, and evaluation techniques.

Filter Design and Analysis Methods — 2 hours | Languages
Learn different methods to design and analyze FIR and IIR filters.

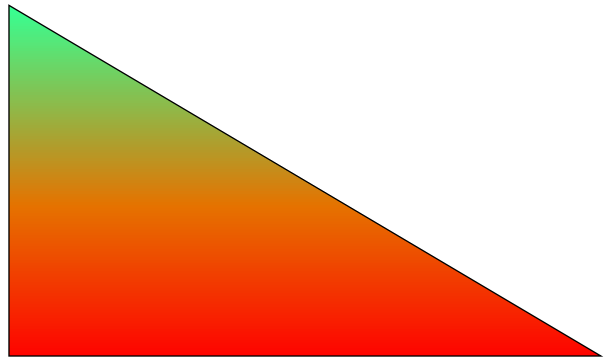
MATLAB learning resources

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

Number of things to remember:



Impossible to remember/know them all!

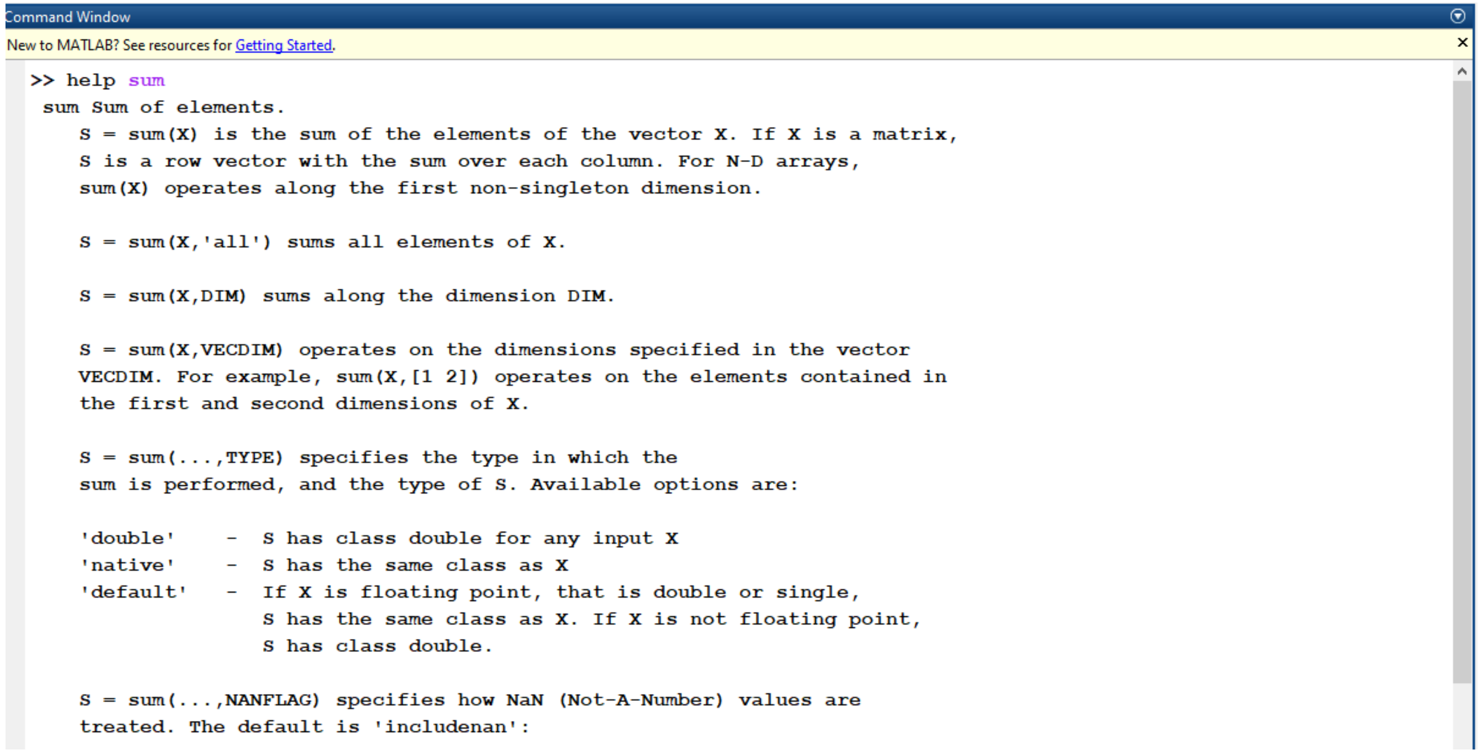
But it is easy remembering to check MATLAB's documentation (even the pros use it)

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

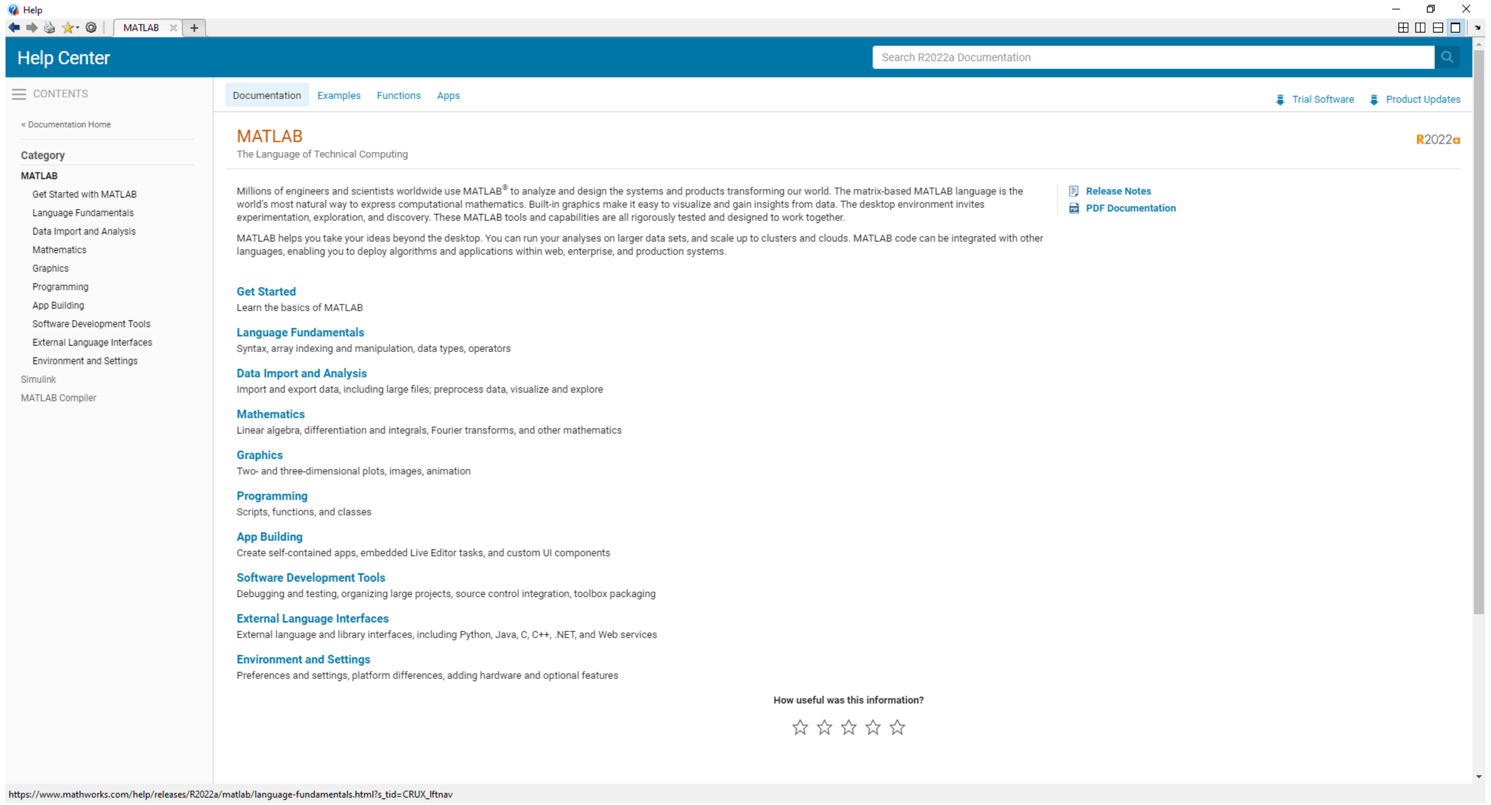
New to MATLAB? See resources for Getting Started.

```
>> 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 centre

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



MATLAB

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.

Get Started
Learn the basics of MATLAB

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

Data Import and Analysis
Import and export data, including large files; preprocess data, visualize and explore

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

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

Programming
Scripts, functions, and classes

App Building
Create self-contained apps, embedded Live Editor tasks, and custom UI components

Software Development Tools
Debugging and testing, organizing large projects, source control integration, toolbox packaging

External Language Interfaces
External language and library interfaces, including Python, Java, C, C++, .NET, and Web services

Environment and Settings
Preferences and settings, platform differences, adding hardware and optional features

How useful was this information?

☆ ☆ ☆ ☆ ☆

https://www.mathworks.com/help/releases/R2022a/matlab/language-fundamentals.html?s_tid=CRUX_lftnav

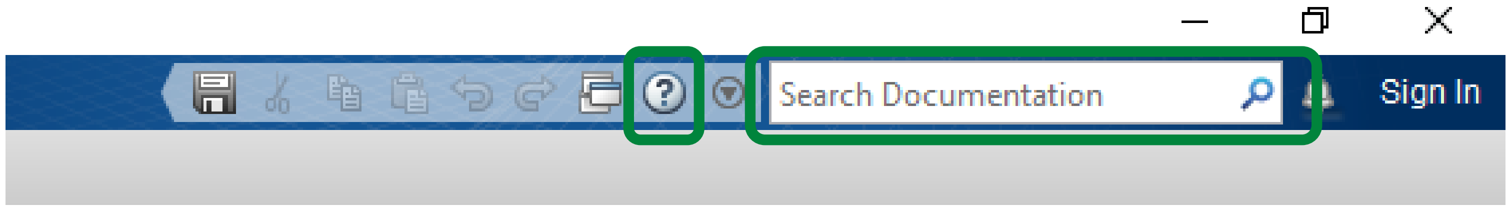
MATLAB learning resources

MATLAB documentation centre

- You can access the **documentation centre** 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:

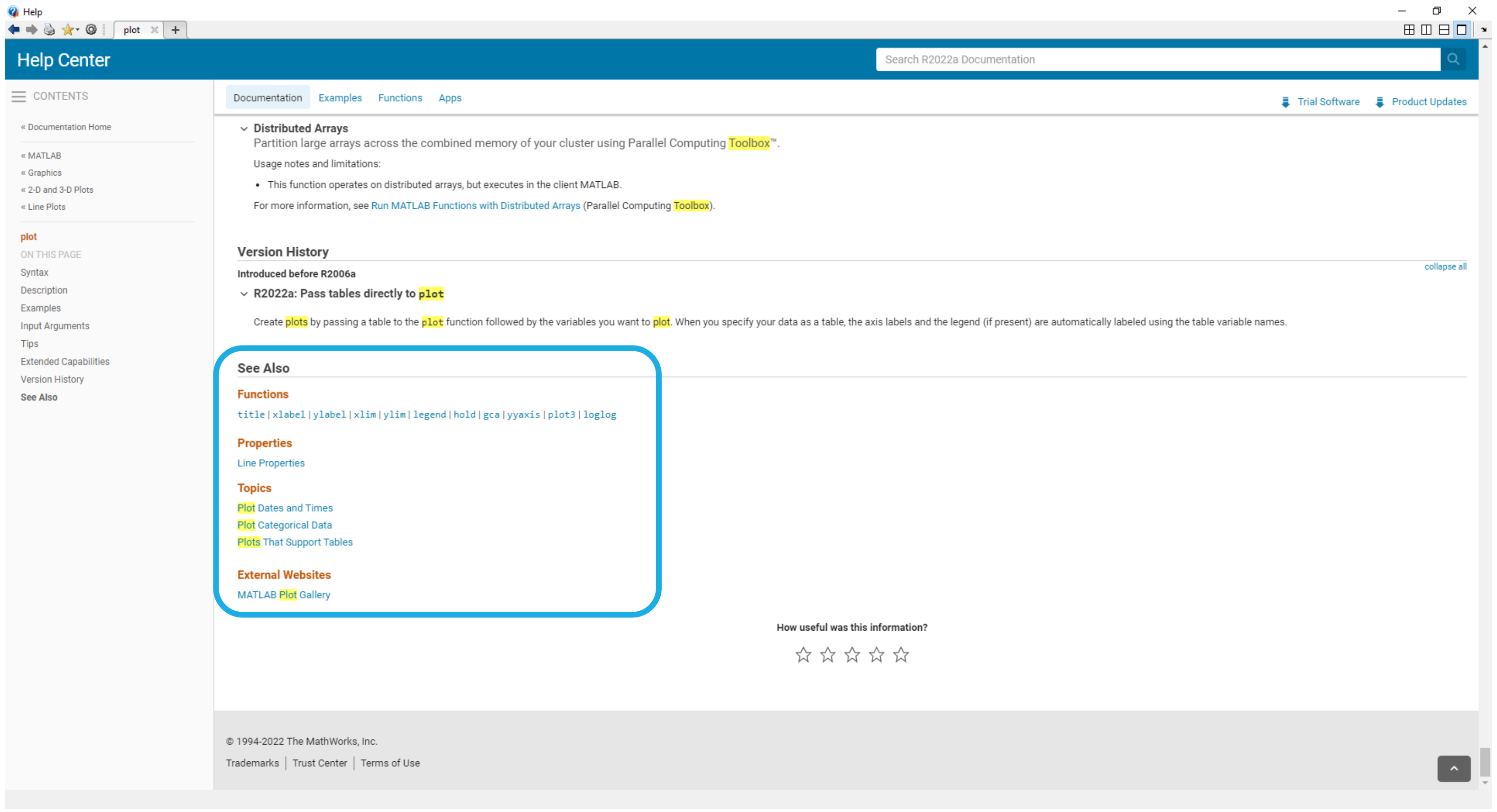


  - Pressing **F1** on your keyboard

MATLAB learning resources

MATLAB documentation centre

- Very often MATLAB already has the functionality you need:
  - Search the **documentation centre** 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



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

Usage notes and limitations:

- This function operates on distributed arrays, but executes in the client MATLAB.

For more information, see Run MATLAB Functions with Distributed Arrays (Parallel Computing Toolbox).

Version History

Introduced before R2006a

R2022a: Pass tables directly to `plot`

Create plots by passing a table to the `plot` function followed by the variables you want to plot. When you specify your data as a table, the axis labels and the legend (if present) are automatically labeled using the table variable names.

See Also

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

Properties
Line Properties

Topics
Plot Dates and Times
Plot Categorical Data
Plots That Support Tables

External Websites
MATLAB Plot Gallery

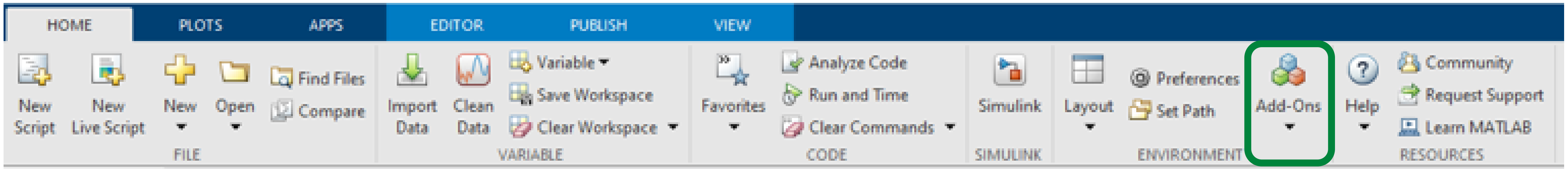
How useful was this information?
☆ ☆ ☆ ☆ ☆

© 1994-2022 The MathWorks, Inc.
Trademarks | Trust Center | Terms of Use

MATLAB Toolboxes

Toolboxes

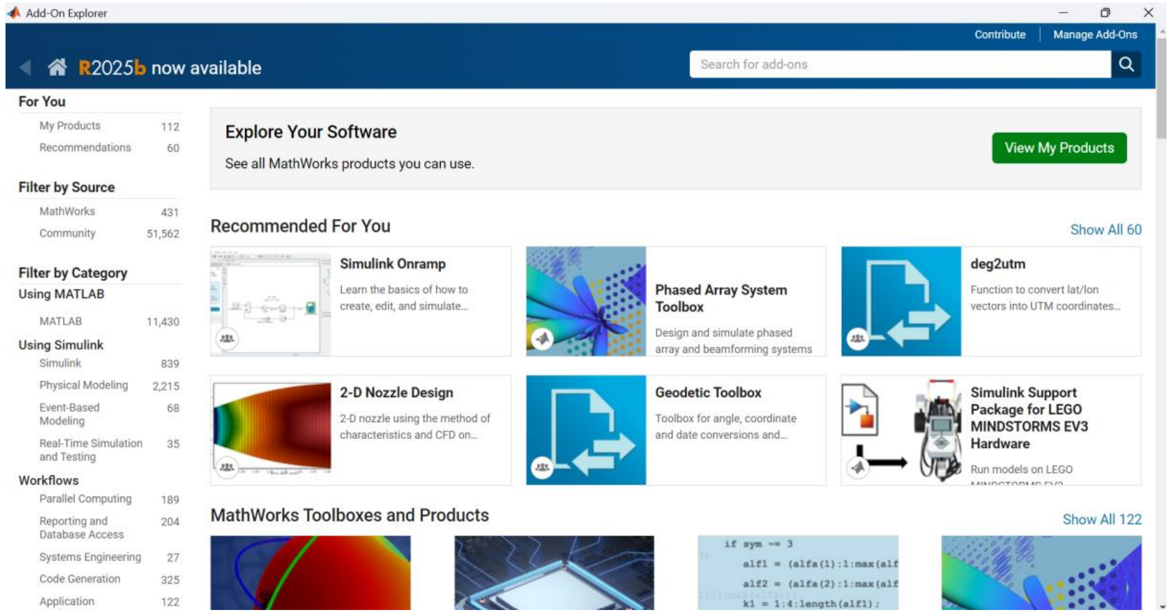
- MATLAB core functionalities are extended through Add-On Toolboxes
  - The available Toolboxes depend on your current license
- Toolboxes can be installed in two different ways
  - During MATLAB installation, you can choose to install some (or all) of the Toolboxes included in your license. Some Toolboxes can take a lot of space, so is better to install only those you need
  - After installation, you can add and remove Toolboxes through the **Add-On Explorer**



MATLAB Toolboxes

Add-On Explorer

- You can search, explore, install, and uninstall Toolboxes and apps from the **Add-on Explorer**
    - For the first lab, you will need the **Optimization Toolbox**, which provides root-finding function `fsolve`. Make sure you have it installed.



For You

My Products	112
Recommendations	60

Filter by Source

MathWorks	431
Community	51,562

Filter by Category

Using MATLAB

MATLAB	11,430

Using Simulink

Simulink	839
Physical Modeling	2,215
Event-Based Modeling	68
Real-Time Simulation and Testing	35

Workflows

Parallel Computing	189
Reporting and Database Access	204
Systems Engineering	27
Code Generation	325
Application	122

Explore Your Software

See all MathWorks products you can use.

Recommended For You

Simulink Onramp — Learn the basics of how to create, edit, and simulate...

Phased Array System Toolbox — Design and simulate phased array and beamforming systems

deg2utm — Function to convert lat/lon vectors into UTM coordinates...

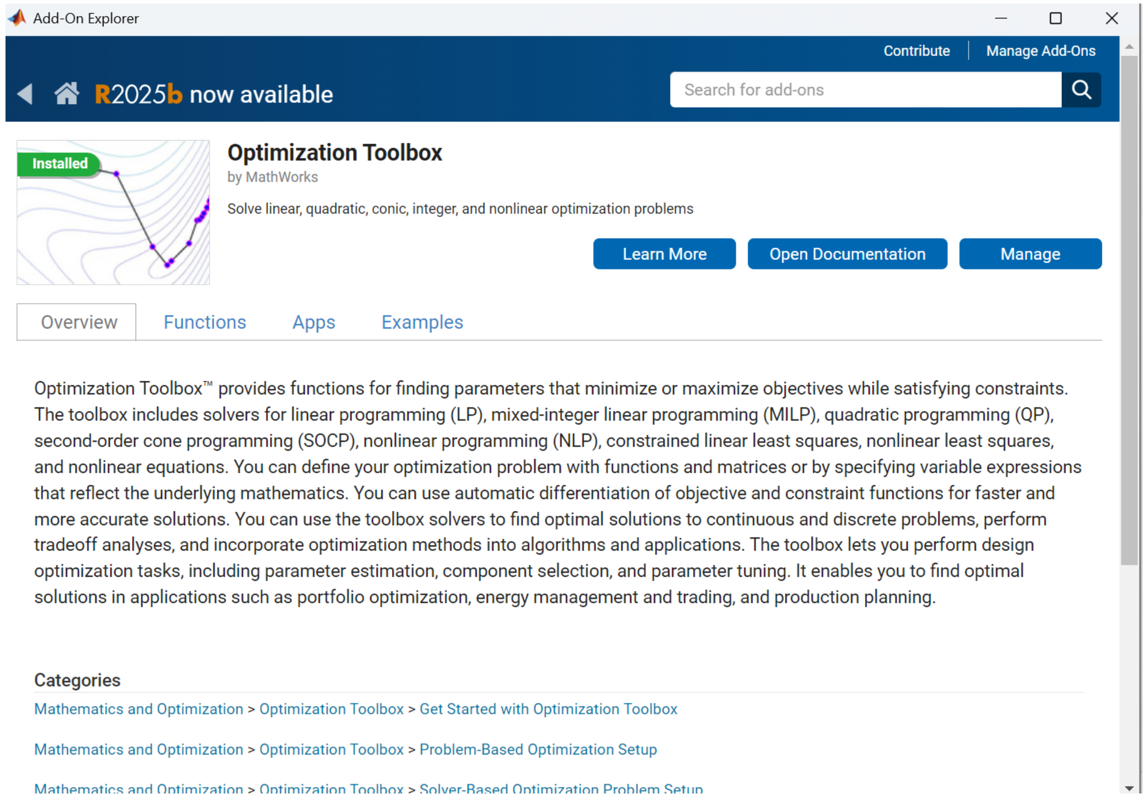
2-D Nozzle Design — 2-D nozzle using the method of characteristics and CFD on...

Geodetic Toolbox — Toolbox for angle, coordinate and date conversions and...

Simulink Support Package for LEGO MINDSTORMS EV3 Hardware — Run models on LEGO

MathWorks Toolboxes and Products

```
if sym ~= 3
    alf1 = (alfa(1):1:max(alf
    alf2 = (alfa(2):1:max(alf
    k1 = 1:4:length(alf1);
```



Optimization Toolbox

by MathWorks

Solve linear, quadratic, conic, integer, and nonlinear optimization problems

Optimization Toolbox™ provides functions for finding parameters that minimize or maximize objectives while satisfying constraints. The toolbox includes solvers for linear programming (LP), mixed-integer linear programming (MILP), quadratic programming (QP), second-order cone programming (SOCP), nonlinear programming (NLP), constrained linear least squares, nonlinear least squares, and nonlinear equations. You can define your optimization problem with functions and matrices or by specifying variable expressions that reflect the underlying mathematics. You can use automatic differentiation of objective and constraint functions for faster and more accurate solutions. You can use the toolbox solvers to find optimal solutions to continuous and discrete problems, perform tradeoff analyses, and incorporate optimization methods into algorithms and applications. The toolbox lets you perform design optimization tasks, including parameter estimation, component selection, and parameter tuning. It enables you to find optimal solutions in applications such as portfolio optimization, energy management and trading, and production planning.

Categories

Mathematics and Optimization > Optimization Toolbox > Get Started with Optimization Toolbox

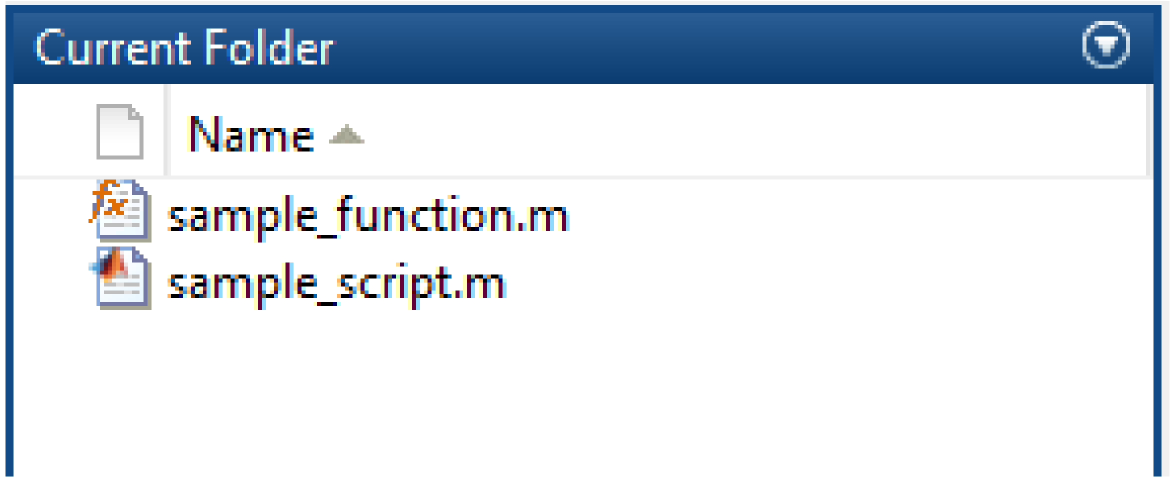
Mathematics and Optimization > Optimization Toolbox > Problem-Based Optimization Setup

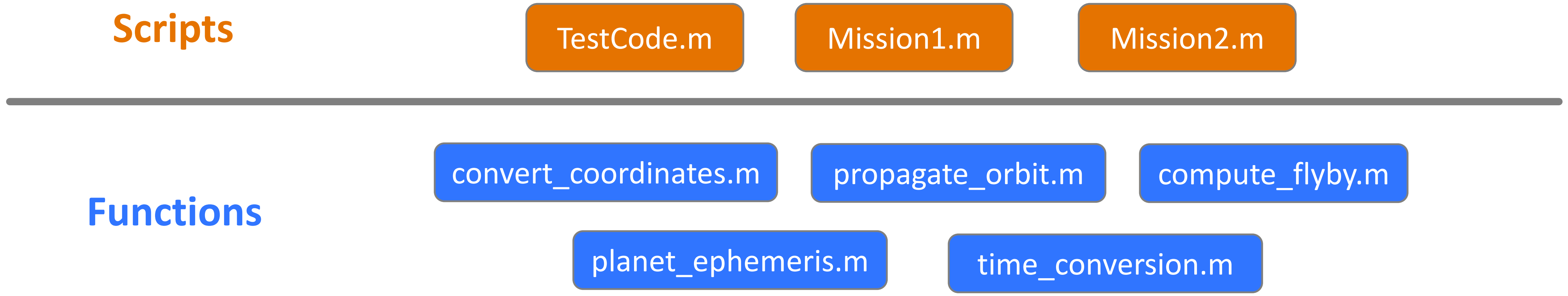
Mathematics and Optimization > Optimization Toolbox > Solver-Based Optimization Problem Setup

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



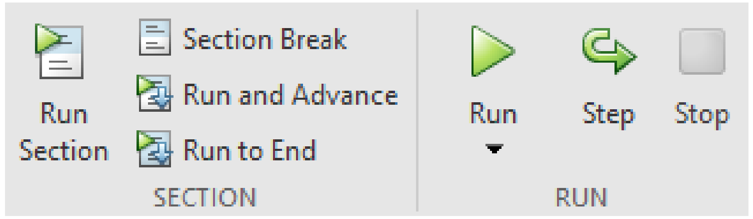


Code structuring

Scripts

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

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

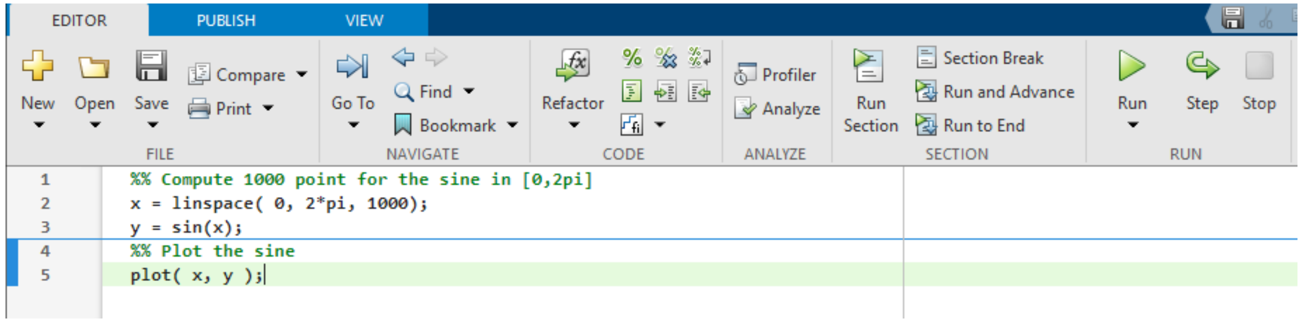


Keep your cursor over a button in the toolbar to see the keyboard shortcut

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

```
>> sample_script
```

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



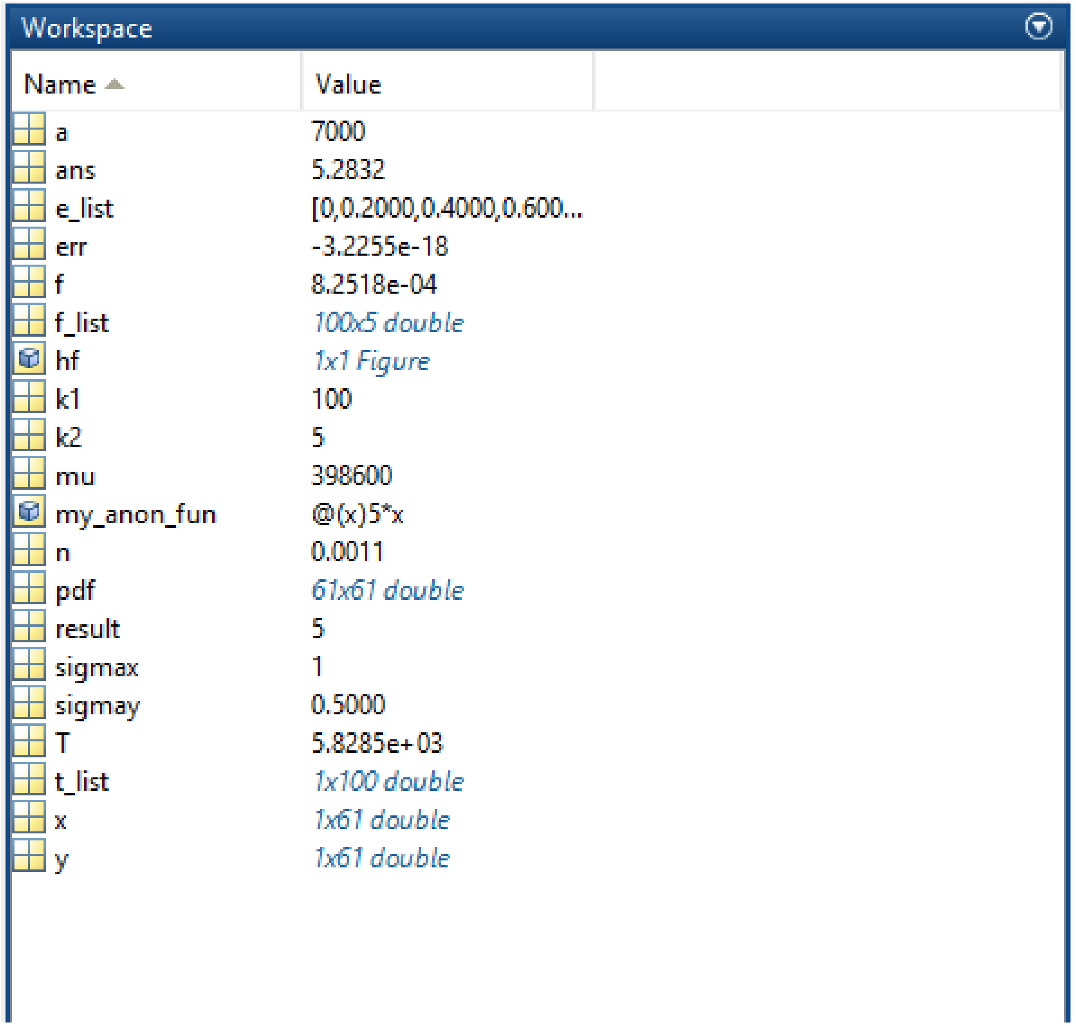
```
%% Compute 1000 point for the sine in [0,2pi]
x = linspace( 0, 2*pi, 1000);
y = sin(x);
%% Plot the sine
plot( x, y );
```

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 the 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

```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 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

Functions

- It is possible to call a **function** without passing all the inputs
  - You can check the number of inputs passed using `nargin`
  - This way, you can program a **function** to have some inputs as ‘optional’

```
function n = mean_motion( a, mu )
% mean_motion Mean motion of a Keplerian orbit
%
% [Remember to document your code, as shown in the previous slide]
%


% If gravitational parameter mu is not provided, the one of Earth is used
if nargin < 2
    mu = 398600.433; % [km^3/s^2]
end


n = sqrt( mu/a^3 );

end
```

Code structuring

Anonymous functions

- An **anonymous function** is a function that is not created 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 executed as any normal **function**

Code structuring

Anonymous functions

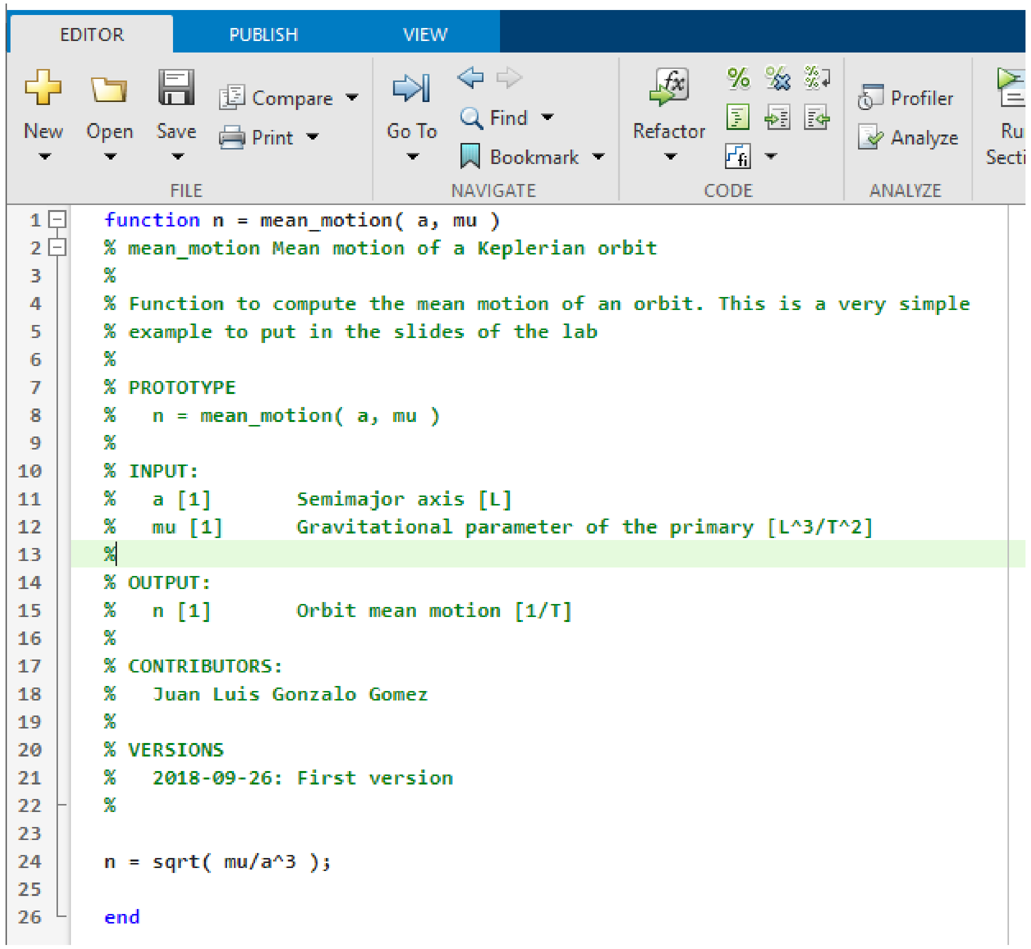
- **Anonymous functions** are normally used to:
  - Define small auxiliary functions within a **function** or **script**, without creating a new file
  - Adapt the **interface** (inputs/outputs) of another function. Example:

```
% 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 );

% I don't even need to store the anonymous function in a variable, I can define
% it directly inside the call to integral:
>> result = integral( @(x) my_fun( x, 5 ), 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



```
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
```

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**:

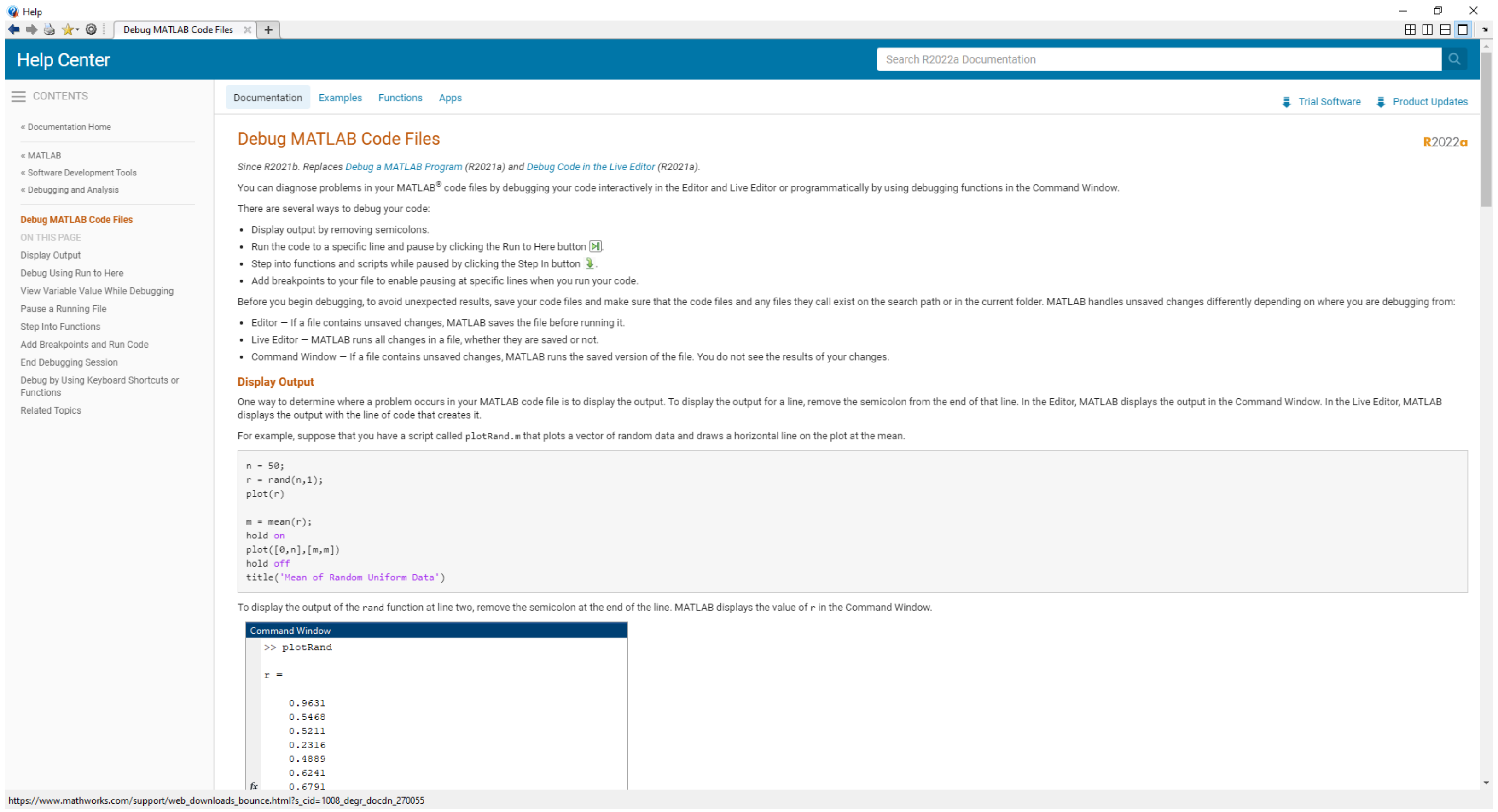
- Pause the execution and advance line by line,
- Check the memory workspace of each function,
- And much more.

Read the documentation page **"Debug MATLAB Code Files"** to learn more!

Best coding practices

Testing

- If you are not familiar with the concept of debugging, read the documentation page **"Debug MATLAB Code Files"** to learn how to pause execution, execute line by line, set breakpoints, and other really useful tools.



« Documentation Home

« MATLAB
« Software Development Tools
« Debugging and Analysis

Debug MATLAB Code Files
ON THIS PAGE
Display Output
Debug Using Run to Here
View Variable Value While Debugging
Pause a Running File
Step Into Functions
Add Breakpoints and Run Code
End Debugging Session
Debug by Using Keyboard Shortcuts or Functions
Related Topics

Debug MATLAB Code Files

Since R2021b. Replaces Debug a MATLAB Program (R2021a) and Debug Code in the Live Editor (R2021a).

You can diagnose problems in your MATLAB® code files by debugging your code interactively in the Editor and Live Editor or programmatically by using debugging functions in the Command Window.

There are several ways to debug your code:

- Display output by removing semicolons.
- Run the code to a specific line and pause by clicking the Run to Here button.
- Step into functions and scripts while paused by clicking the Step In button.
- Add breakpoints to your file to enable pausing at specific lines when you run your code.

Before you begin debugging, to avoid unexpected results, save your code files and make sure that the code files and any files they call exist on the search path or in the current folder. MATLAB handles unsaved changes differently depending on where you are debugging from:

- Editor — If a file contains unsaved changes, MATLAB saves the file before running it.
- Live Editor — MATLAB runs all changes in a file, whether they are saved or not.
- Command Window — If a file contains unsaved changes, MATLAB runs the saved version of the file. You do not see the results of your changes.

Display Output

One way to determine where a problem occurs in your MATLAB code file is to display the output. To display the output for a line, remove the semicolon from the end of that line. In the Editor, MATLAB displays the output in the Command Window. In the Live Editor, MATLAB displays the output with the line of code that creates it.

For example, suppose that you have a script called `plotRand.m` that plots a vector of random data and draws a horizontal line on the plot at the mean.

```
n = 50;
r = rand(n,1);
plot(r)

m = mean(r);
hold on
plot([0,n],[m,m])
hold off
title('Mean of Random Uniform Data')
```

To display the output of the `rand` function at line two, remove the semicolon at the end of the line. MATLAB displays the value of `r` in the Command Window.

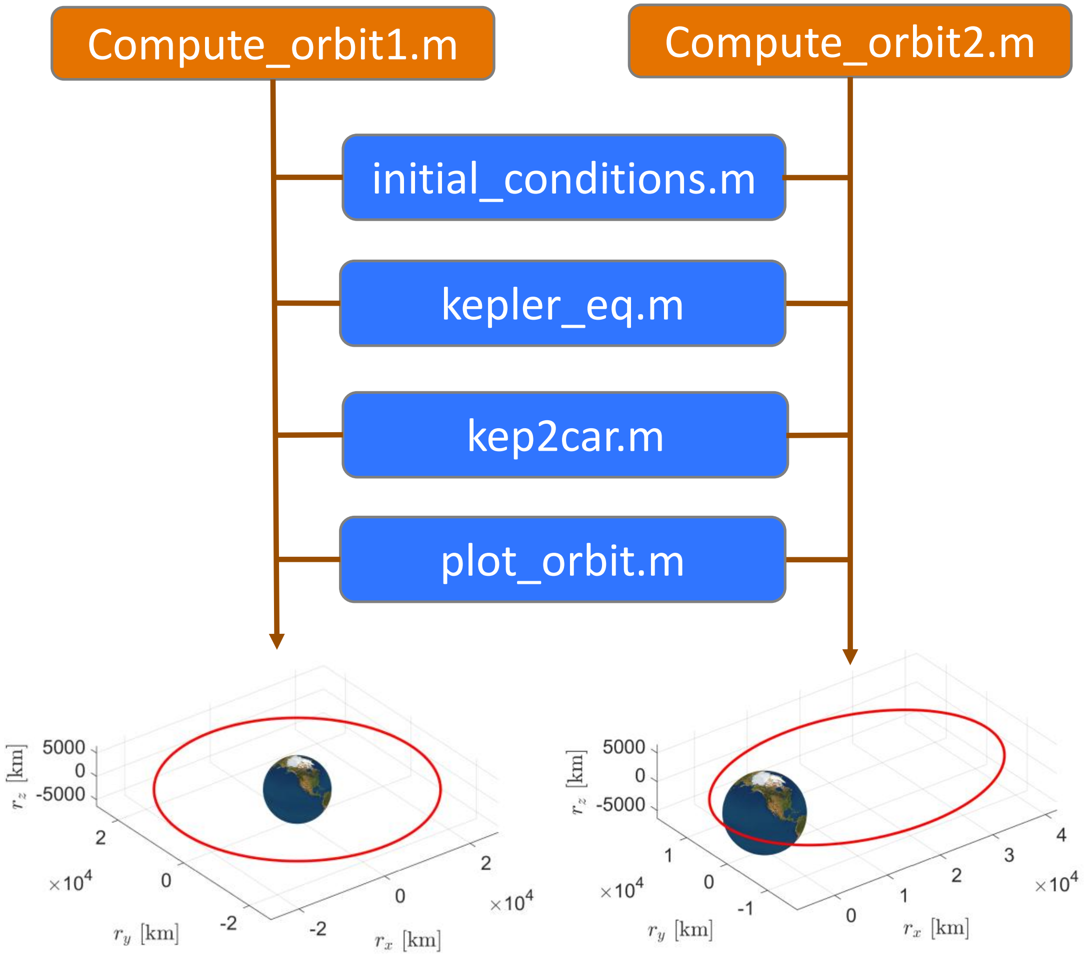
Command Window
```
>> plotRand

r =

    0.9631
    0.5468
    0.5211
    0.2316
    0.4889
    0.6241
fx  0.6791
```

https://www.mathworks.com/support/web_downloads_bounce.html?s_cid=1008_degr_docdn_270055

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 consider in the final mark

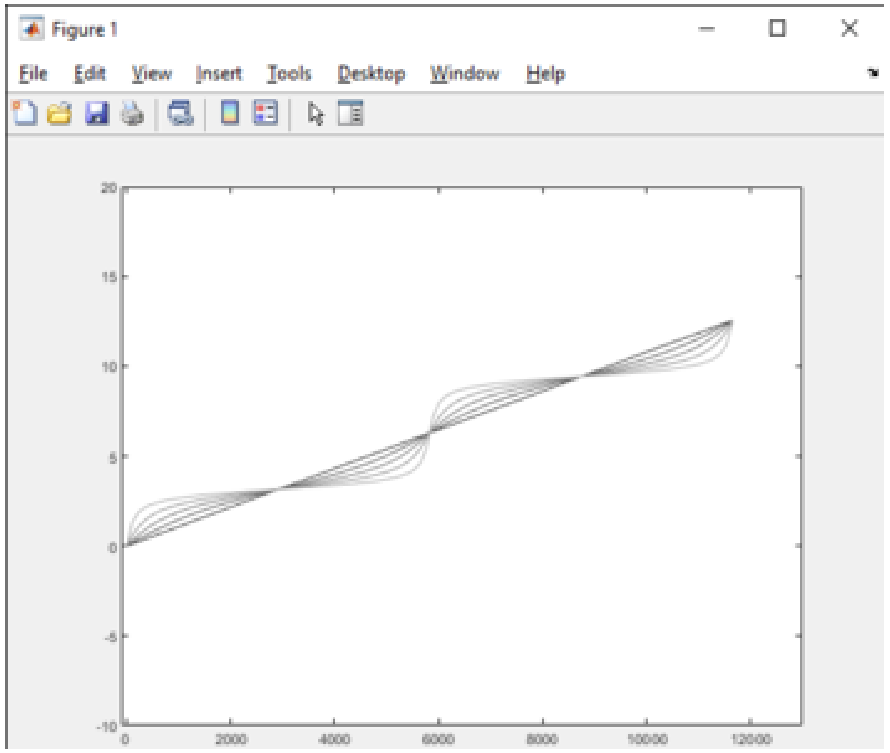
Figures

This is a BAD figure

It is just a low-resolution screenshot

Font is too small to read

Not clear what the axes represents, or their physical units



Very difficult to distinguish the lines

Not explained the meaning of each line

Plenty of wasted blank space

Without a title or a caption, it is impossible to 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

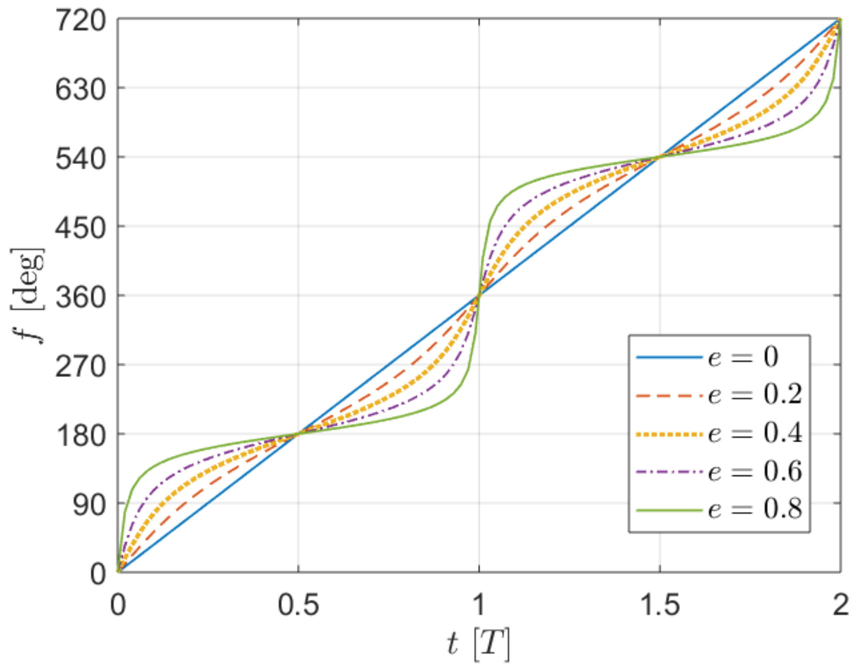


Figure 1. Evolution of true anomaly with time for several orbits with $a = 7000$ km, $\mu = 398600\ \mathrm{km}^3/\mathrm{s}^2$, and different e

Lines are set apart using different colors and line styles

A legend is included

A grid is included to improve readability

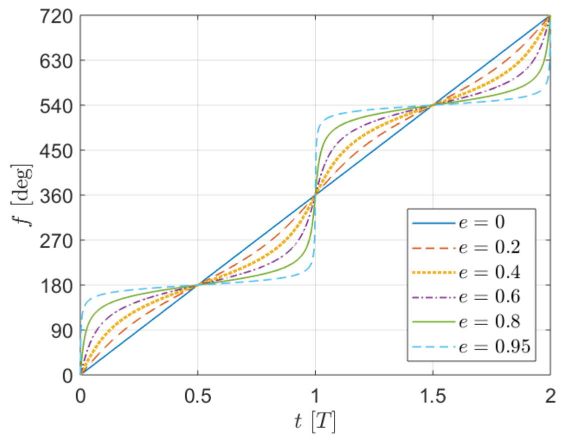
Ranges are set to make use of all available space

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

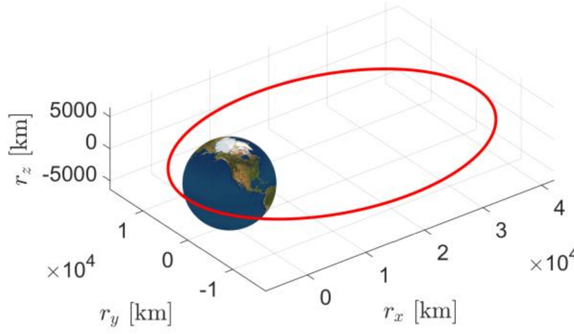
Figures

Plotting in MATLAB

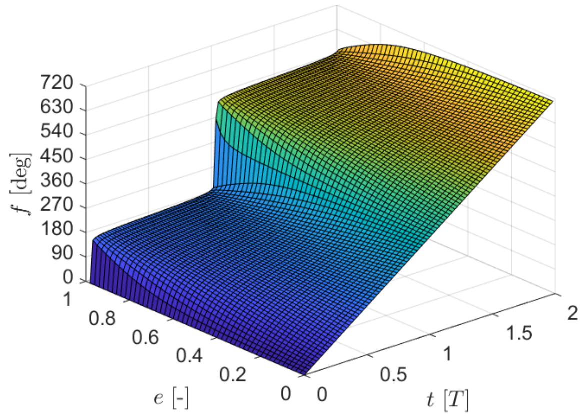
- MATLAB can represent many types of figures (check the **documentation centre** to learn how to use them)



`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

Customising plots in MATLAB

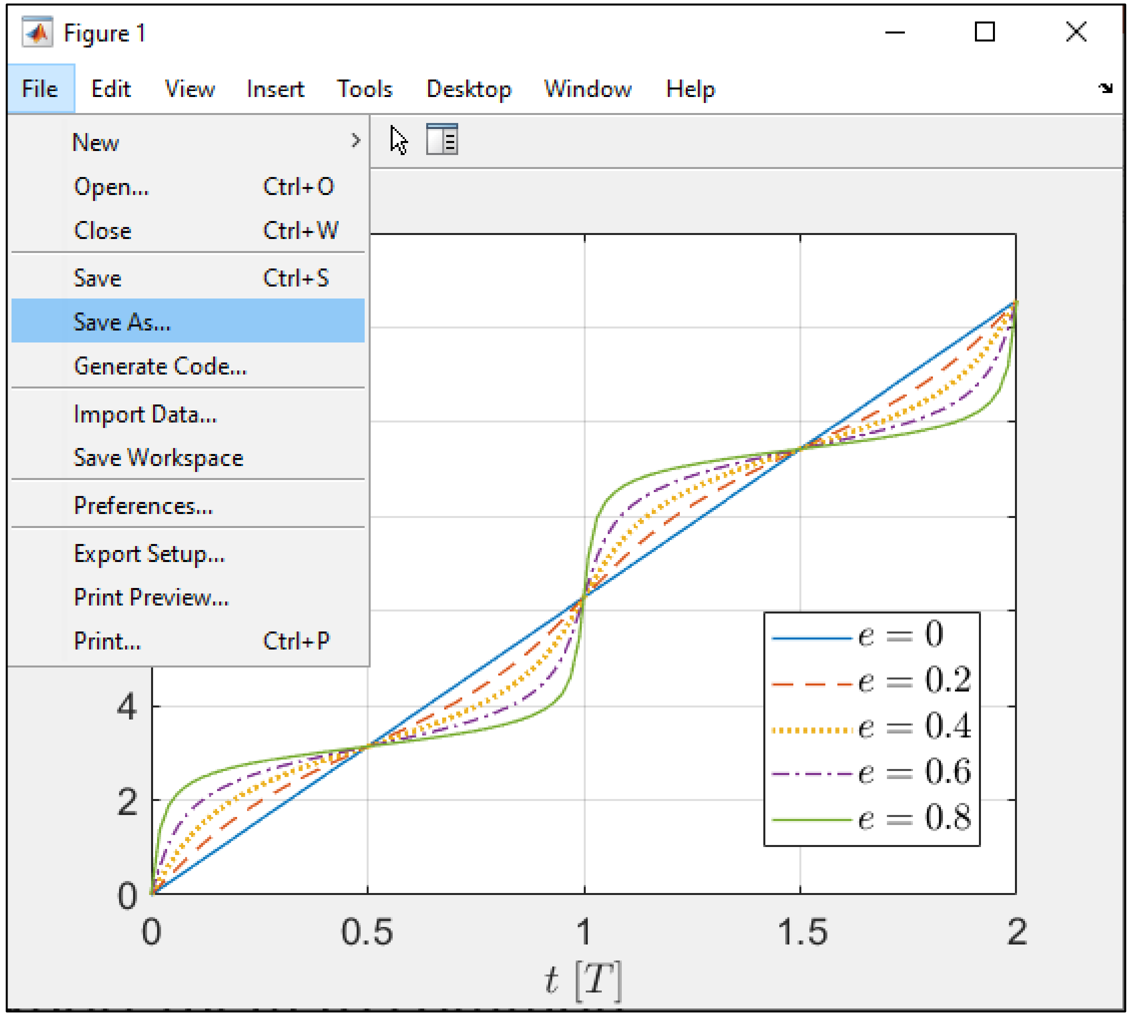
- Properties of plots can be adjusted from the **graphical interface** or using **commands** inside function/scripts and the command line:
    - `xlabel`, `ylabel`, and `zlabel`: Set labels for x, y, and z axes
    - `xlim`, `ylim`, and `zlim`: Set the plotting range for each axis
    - `title`: Set 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…”
  - You can configure the export options in menu “File > Export Setup…”
  - MATLAB cannot edit these formats after exporting
- You can also save figures from code using commands such as `saveas` and `print`

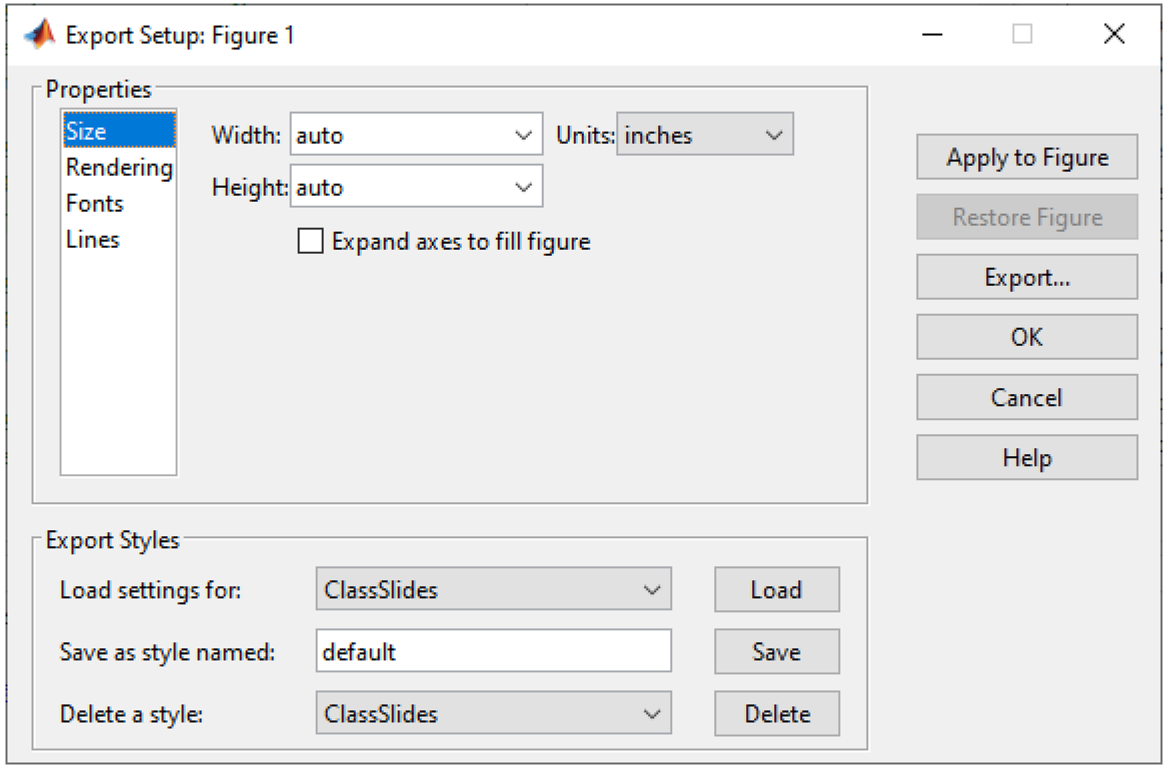


Screenshots are not a good way to include figures in the report or presentation

Figures

Export Setup

- The **Export Setup** menu allows you to configure several properties for your exported figure
  - Size of the figure (in inches, cm, points)
  - Rendering resolution for raster formats (increase it if your images look blurry)
  - Font size and family
  - Line width
  - Remember to click on "Apply to Figure" when you are done setting the properties
- You can save and load settings, to reuse them for future images
  - This saves time and ensures a uniform style in your documents
  - You may want to use different styles for reports and presentations


Export Setup: Figure 1

Properties

Width: auto
Units: inches
Height: auto
Expand axes to fill figure

Export Styles

Load settings for: ClassSlides
Save as style named: default
Delete a style: ClassSlides