Lab 1: Using Matlab to solve ODEs

MCHA3400

Semester 1 2024

---

1 Introduction

Modelling and simulation of physical systems are fundamental skills required for designing and analysing mechatronic systems. In this lab, we will work towards developing these proficiencies by familiarising ourselves with the Matlab interface for writing functions and solving ODEs.

This lab is worth 1.5% of your overall grade. Show your results to your tutor by the end of your lab session to be awarded marks. Labs will also be accepted at the beginning of your lab 2 session. After this point in time, no marks will be awarded.

Lab task 0: Python and CoolProp installation

From week 6 onward we will be simulating systems with thermodynamic elements. To enable this, the python library `CoolProp` will be used to evaluate the thermodynamic properties of fluids as they interact with a dynamic system.

This section describe how to install both `Python` and `CoolProp` on your local own machine and verify its correct operation. It will also provide instructions on how to verify the correct installation on university PCs. Note that `CoolProp` is installed on the PCs in ES409, but is not available in other computer labs. It is recommended that you follow these instructions now so that there is time to address any issues prior to week 6.

- **Python installation (Not required on University PCs):** Matlab's Python version compatibility depends on the version of Matlab being used. A list of compatibilities is available via the Mathworks website (`https://au.mathworks.com/support/requirements/python-compatibility.html`). Install a compatible version of Python 3.X, downloaded from the `www.python.org` website. For example, Windows installations are found here with the latest version of Python 3.8 with an executable installer being 3.8.8. Ensure that you **select the optional 'Add Python3.X to PATH'** option when installing so that Matlab can find the installation.

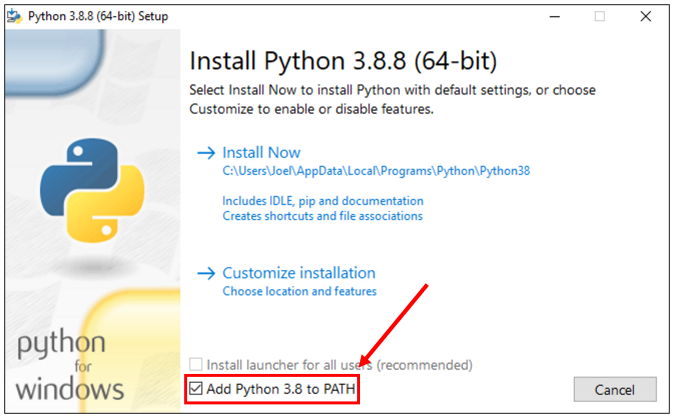

Figure 1: Add the python installation to the system path.

- **Matlab / Python interface:** Open a version of Matlab compatible with your python installation (see above) and test it Matlab can see the Python installation by calling the `pyenv` command. Matlab will return the default Python version based on what is in the system path (see above). Further information on configuring and troubleshooting your installation is available via the Mathworks website here. In particular, you can select which python installation to use within Matlab via the `pyenv` command. For example, to manually select to use the Python 3.8 installation if multiple are available, enter the command `pyenv('Version','3.8')`. Please inform your tutor if you are having difficulties with step, or if the university Matlab installation is unable to detect a Python installation.

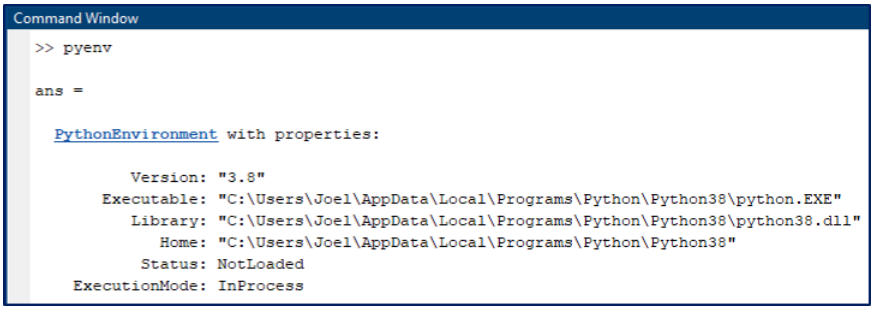

Figure 2: The default Python installation can be viewed from Matlab.

- **CoolProp installation (Not required on University PCs):** Next we will install the CoolProp libraries in Python by following the instructions available here. The installation is done via a pip install which can be completed in a terminal or via the Matlab Python interface. If installing via a terminal, ensure that you are installing to the same Python installation being used by Matlab.

  In Matlab, enter the command `[v,e] = pyversion; system([e,' -m pip install --user -U CoolProp'])` which should produce results similar to below. **If you get an error in this step, please inform your tutor.**

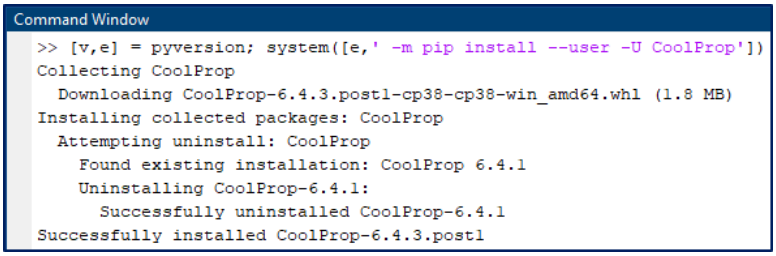

Figure 3: CoolProp can be added to the Python installation in a terminal or via the Matlab interface.

- **Testing CoolProp via the Matlab interface:** The final step is to verify that the CoolProp libraries have been properly installed and are visible from Matlab. In the Matlab console, enter the command `py.CoolProp.CoolProp.PropsSI('T','P',101325,'Q',0,'Water')`, which should produce the response below. You may need to restart Matlab after first installing Python or CoolProp. **If you receive an error in this step, please inform your tutor.**

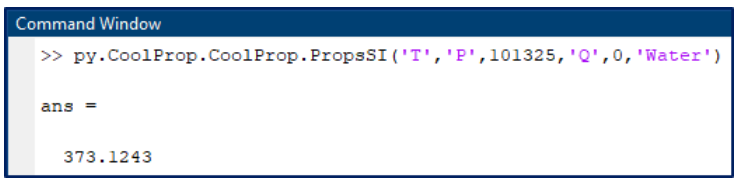

Figure 4: CoolProp can be used to evaluate thermodynamic properties in Matlab via the Python interface.

Lab task 1: Numerical ODE solvers

Matlab offers a variety of ODE solvers that can be used to generate numerical approximations to the solutions of ODEs. An overview of the available ODE solvers can be found in the Matlab documentation (https://au.mathworks.com/help/matlab/math/choose-an-ode-solver.html). In this lab, we will make use of the ode45 and ode23s solvers. The ode45 solver is a good general purpose solver and a good first attempt at finding solutions to ODEs.

Utilising ODE solvers require a Matlab function handle to be passed. This is how the solver is given the information about the ODE to be solved. Further documentation on function handles in Matlab is available at (https://au.mathworks.com/help/matlab/ref/functions.html).

a) Write an anonymous function in Matlab that, given a value for $x \in \mathbb{R}$, evaluates the expression

$$f(x) = -|x|x. \tag{1}$$

Verify your function is correct by substituting in some test values. For example, your function should successfully evaluate $f(-4) = 16$.

> **Tip**
> This should be 1 line of code of the form `f = @(x) ...`

b) Write a function that, given a value for $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in \mathbb{R}^2$, evaluates the expression

$$g(\mathbf{x}) = \begin{bmatrix} -\mathbf{x} \cdot \mathbf{x} + x_1 x_2 \\ -x_2|x_2| \end{bmatrix}, \tag{2}$$

where $\mathbf{x} \cdot \mathbf{x} = x_1^2 + x_2^2$ is the standard dot product. Verify that your function is functioning as expected by substituting in some test values.

> **Tip**
> Given a vector $\mathbf{x}$, the command $\mathbf{x}(1)$ evaluates the first element of $\mathbf{x}$.

c) Consider the ODE

$$\dot{\mathbf{x}} = \begin{bmatrix} -a\mathbf{x} \cdot \mathbf{x} + x_1 x_2 \\ -b x_2|x_2| \end{bmatrix}, \tag{3}$$

where $x(0) = \begin{bmatrix} 2 & -5 \end{bmatrix}^\top$, $a = 0.2$ and $b = 0.8$. Write a scrip in Matlab using the ode45 function to determine the solution of this ODE over the time interval $t \in [0, 30]$. Your solution should agree with Figure 5.

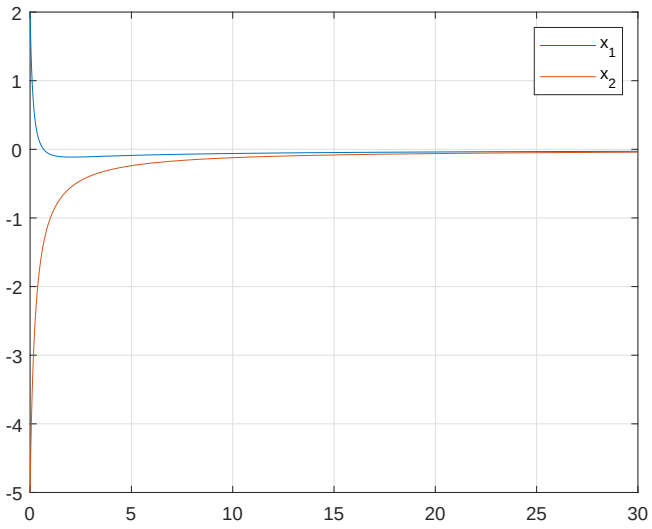

Figure 5: Solution output from task 1c.

Tip

Before using an unfamiliar function, you can access the MATLAB Documentation Center using the `doc` command. This documentation helps you to understand any function as it gives the function's purpose, inputs, outputs and short examples on how to use the function in varying situations.

Open the documentation for `ode45` by typing the following into the command window:

Matlab

```
doc ode45
```

d) Construct an anonymous function for an input signal $\mathbf{u}(\mathbf{x}, t)$ which is defined by

$$\mathbf{u}(\mathbf{x}, t) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}. \tag{4}$$

Using the same parameters from the previous step, simulate the ODE

$$\dot{\mathbf{x}} = \begin{bmatrix} -a\mathbf{x} \cdot \mathbf{x} + x_1 x_2 \\ -b x_2 |x_2| \end{bmatrix} + \mathbf{u}(\mathbf{x}, t). \tag{5}$$

The solution should agree with the previous solution.

e) Now let the input be given by

$$\mathbf{u}(\mathbf{x}, t) = \begin{bmatrix} -x_1 + \sin(t) \\ -\cos(4t) \end{bmatrix}. \tag{6}$$

Simulate the system over the time interval $t \in [0, 30]$ and plot the output. Determine the minimum value attained by $x_1(t)$ in the simulated time interval? Your solution should agree with Figure 6.

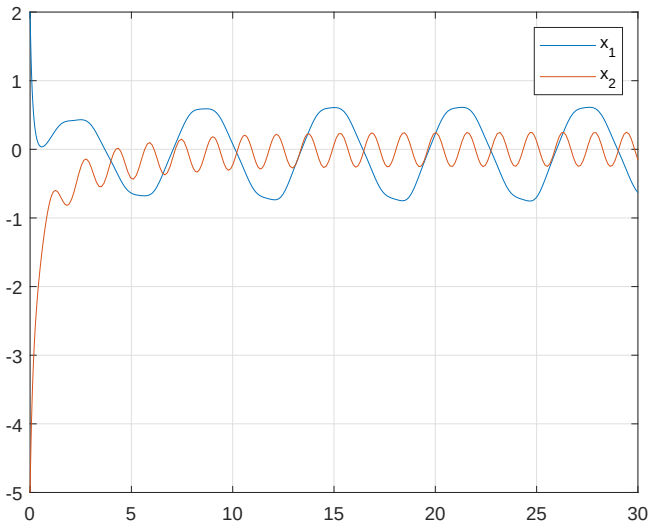

Figure 6: Solution output from task 1e.

Lab task 2: Matlab functions

As seen in lectures, we are able to construct both explicit and anonymous functions in Matlab. Furthermore, functions are able to be declared in separate Matlab files, or within the scripts that they are to be used. In this task, we will practice several different types of function declarations.

Standard deviation is a measure of the amount of variation in a dataset. Standard deviation is evaluated according to the formula

$$s = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}}, \tag{7}$$

where x_i is the i^{th} data-point, $\bar{x}$ is the mean of all data and n is the total number of data-points. In this task, we will create a function that, given a set of scalar values, returns the mean and standard deviation.

a) Create a function called `computeAverage` in the same way that was shown in Lecture 2. Save the function in your working directory and verify that it works by entering some test values.

b) Create a function called `computeStdDev` with the following inputs and outputs:

Inputs	Outputs	Description
`inputMatrix`		$N \times M$ matrix of real values
	`mean`	Mean of all inputs values
	`standardDev`	Standard deviation of input data

Use your `computeAverage` function to compute the value of $\bar{x}$.

c) Test your function by inputting some test data. For example, you could use the inputs:

- `A1 = zeros(5,8)`
- `A2 = ones(5,8)`
- `A3 = normrnd(0,1,[500,800])`

Ensure that you know what return value you are expecting before running your tests!

Tip

You can test your function by inputting data with known return values. For example, you may use the Matlab function `zeros`, `ones` or `normrnd` for this task.

d) Once your `computeStdDev` function is behaving as expected, move the `computeAverage` function to the bottom of the `computeStdDev.m` file and remove `computeAverage.m` from your workspace (but keep a backup!). Once completed, ensure that your `computeStdDev` function continues to work as expected.

Recommendations

Here are some suggestions on what you should work on next:

- The way in which Matlab ODE solvers operate can be modified by using the `odeset` command (`https://au.mathworks.com/help/matlab/ref/odeset.html`). Use this optional input to set the ODE solver in task 1 to a have a maximum relative tolerance of 1×10^{-6}.
- There are many ways in which the Matlab plots can be modified. Modify your output of task 1 so that the axis of your plots are labelled, a legend appears on the plot, the text has font size 11 and the lines have width of 2.