MAT 275 Laboratory 1
Introduction to MATLAB

MATLAB is a computer software commonly used in both education and industry to solve a wide range of problems. This Laboratory provides a brief introduction to MATLAB, and the tools and functions that help you to work with MATLAB variables and files.

NOTE: Use the provided Live Script template for your lab write-up.

- Unless otherwise specified, your write-up should contain the MATLAB input commands, the corresponding output, and the answers to the questions that you have written (if any).
- If the exercise asks you to write an M-file, display the file in your lab write-up in the appropriate position (after the problem number and before the output generated by the file).
- Convert your file to pdf format. **Only a single pdf file should be submitted!**

The MATLAB Environment

★ To start MATLAB double-click on the MATLAB shortcut icon. The MATLAB desktop will open.

In the left side you will generally find the Current Folder window and on the right the Workspace. The Command Window is where the MATLAB commands are entered and executed. Note that windows within the MATLAB desktop can be resized by dragging the separator bar(s).

Basics And Help

Commands are entered in the Command Window.
★ Basic operations are `+`, `-`, `*`, and `/`. The sequence

```
>> a=2; b=3; a+b, a*b
ans =
      5
ans =
      6
```

defines variables a and b and assigns values 2 and 3, respectively, then computes the sum $a+b$ and product ab. Each command ends with `,` (output is visible) or `;` (output is suppressed). The last command on a line does not require a `,`.
★ Standard functions can be invoked using their usual mathematical notations. For example

```
>> theta=pi/5;
>> cos(theta)^2+sin(theta)^2
ans =
          1
```

verifies the trigonometric identity $\sin^2\theta+\cos^2\theta=1$ for $\theta=\frac{\pi}{5}$. A list of elementary math functions can be obtained by typing

```
>> help elfun
```

★ To obtain a description of the use of a particular function type help followed by the name of the function. For example

```
>> help cosh
```

gives help on the hyperbolic cosine function.
★ To get a list of other groups of MATLAB programs already available enter `help`:

```
>> help
```

★ Another way to obtain help is through the help button ? in the toolbar.

★ MATLAB is case-sensitive. For example

```
>> theta=1e-3, Theta=2e-5, ratio=theta/Theta
theta =
  1.0000e-003
Theta =
  2.0000e-005
ratio =
    50
```

★ The quantities `Inf` (∞) and `NaN` (Not a Number) also appear frequently. Compare

```
>> c=1/0
c =
    Inf
```

with

```
>> d=0/0
d =
    NaN
```

Vectors in MATLAB

To create a row vector, separate the elements with either a comma (,) or a space

```
>> x=[2,4,6,8]
x =
    2  4  6  8
```

We can perform operations on the vector:

```
>> x+x
ans =
    4  8  12  16
>> 2*x
ans =
    4  8  12  16
>> x.^2
ans =
    4  16  36  64
```

Each entry of the vector x is squared.

```
>> x.^x
ans =
    4  16  36  64
```

Each entry of the vector x is multiplied by itself.

Note that in the above example multiplication (`*`) and exponentitation (`^`) are preceeded by a dot (.) in order for the expression to be evaluated for each component (entry) of x. This is necessary to prevent MATLAB from interpreting these symbols as standard linear algebra symbols operating on arrays. Because the standard `+` and `-` operations on arrays already work componentwise, a dot is not ne cessary for `+` and `-`.

We can apply MATLAB built in functions to the vector

```
>> sqrt(x)
ans =
     1.4142   2.0000  2.4495  2.8284
```

Exercise 1

All points with coordinates $x = r\cos(\theta)$ and $y = r\sin(\theta)$, where r is a constant, lie on a circle with radius r, i.e. satisfy the equation $x^2 + y^2 = r^2$.
Create a row vector for θ with the values $\theta = 0, \frac{\pi}{5}, \frac{\pi}{3}, \frac{\pi}{2}, \frac{3\pi}{2}, \frac{4\pi}{3}$ and $\frac{5\pi}{4}$.
Take $r = 5$ and compute the row vectors x and y.
Now check that x and y indeed satisfy the equation of a circle, by computing the radius $r = \sqrt{x^2 + y^2}$.
Hint: To calculate r you will need the array operator `.^` for squaring x and y. Of course, you could also compute x^2 by `x.*x`.

The colon (:) operator and the linspace command

Rather than manually entering each entry of a vector, we can create equidistant vectors by using the colon : operator:

```
>> x = 0:.2:2
x =
  0  0.2000  0.4000  0.6000  0.8000  1.0000  1.2000  1.4000  1.6000  1.8000 2.0000
```

MATLAB creates a vector with the first entry equal to 0, the last entry equal to 2 and with equidistant entries by increments of 0.2. The most general form of the command is

```
<start> : <step> : <end>
```

where `<start>` is the first entry to be put in the vector, the next entry will be `<start> + <step>`, the next entry `<start> + 2<step>` and so on. The last entry in the vector is the largest value of `<start> + P<step>` less than or equal to `<end>`.

Alternatively, we can use the `linspace` command. This command specifies the number of entries in the vector (rather than the step between each entry). The general form of the command is

```
linspace(<start> : <end> : <number of entries>)
```

For example

```
>> x=linspace(0,2,11)
```

generates the same output vector `x`.

Plotting with MATLAB

★ To plot a function you have to create two arrays (vectors): one containing the abscissae, the other the corresponding function values. Both arrays should have the same length. For example, consider plotting the function

$$y = f(x) = \frac{x^2 - \sin(\pi x) + e^x}{x - 1}$$

for $0 \leq x \leq 2$. First choose a sample of x values in this interval. Here we use the colon operator to create a vector that starts at 0, ends at 2 and has increments of 0.1.

```
>> x= 0:0.1:2
x =
  Columns 1 through 7
        0    0.1000    0.2000    0.3000    0.4000    0.5000    0.6000
  Columns 8 through 14
    0.7000    0.8000    0.9000    1.0000    1.1000    1.2000    1.3000
  Columns 15 through 21
    1.4000    1.5000    1.6000    1.7000    1.8000    1.9000    2.0000
```

★ The output for `x` can be suppressed (by adding `;` at the end of the command). Alternatively, we can also display the output in a more compact form by entering

```
>> format compact
```

This format suppresses unnecessary lines and allows more information to show on the screen.
★ To evaluate the function f simultaneously at all the values contained in x, type

```
>> y=(x.^2-sin(pi.*x)+exp(x))./(x-1)
y =
  Columns 1 through 6
   -1.0000   -0.8957   -0.8420   -0.9012   -1.1679   -1.7974
  Columns 7 through 12
   -3.0777   -5.6491  -11.3888  -29.6059       Inf   45.2318
  Columns 13 through 18
   26.7395   20.5610   17.4156   15.4634   14.1068   13.1042
  Columns 19 through 21
   12.3468   11.7832   11.3891
```

Note that the function becomes infinite at $x = 1$ (vertical asymptote). The array y inherits the dimension of x, namely 1 (row) by 21 (columns). Note also the use of parentheses.

IMPORTANT REMARK
In the above example `*`, `/` and `^` are preceded by a dot `.` in order for the expression to be evaluated for each component (entry) of x.

The command

```
>> plot(x,y)
```

creates a Figure window and shows the function (see Fig. 1). Note how the curve is not smooth. The reason is that the vector `x` does not have enough entries.
We can see more clearly how MATLAB plots the curve, by plotting circles `'o'` at each of the ordered pairs (x, y) and connecting the circles (see Fig. 2):

```
>> plot(x,y,'o-')
```

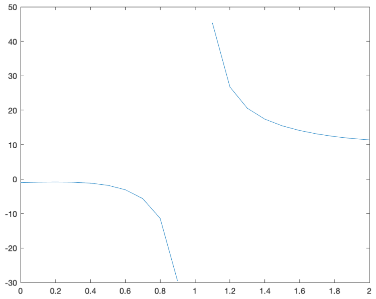

Figure 1

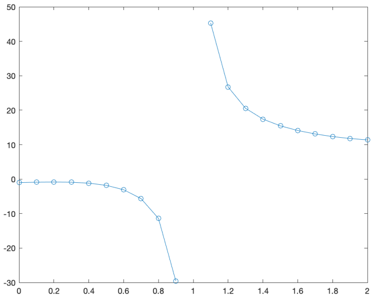

Figure 2

The figure can be edited and manipulated using the Figure window menus and buttons. Alternately, properties of the figure can also be defined directly at the command line:

```
>> x=0:.01:2;     % create a vector x with more entries
>> y=(x.^2-sin(pi.*x)+exp(x))./(x-1);   %recompute y
>> plot(x,y,'r-','LineWidth',2);  %plot in red with a thicker line
>> axis([0,2,-10,20]); % change the window to [0,2]x[-10,20]
>> grid on;   % add a grid
>> title('f(x)=(x^2-sin(\pi x)+e^x)/(x-1)');  % create a title
>> xlabel('x'); ylabel('y');   % create labels
```

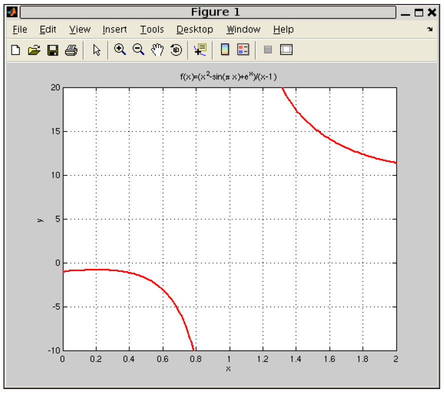
Figure 1
File Edit View Insert Tools Desktop Window Help
f(x)=(x²-sin(π x)+e^x)/(x-1)
20
15
10
5
0
-5
-10
y
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
x

Figure 3

Remarks:

- The number of x-values has been increased for a smoother curve (note that the stepsize is now .01 rather than .1).
- The vector y is recomputed using the new vector x.
- The option `’r-’` plots the curve in red.
- `’LineWidth’,2` sets the width of the line to 2 points (the default is 0.5).
- The range of x and y values has been reset using `axis([0,2,-10,20])` (always a good idea in the presence of vertical asymptotes). The general command is `axis([xmin, xmax, ymin, ymax])`.

- The command `grid on` adds a grid to the plot.
- A title and labels have been added.

The resulting new plot is shown in Fig. 3. For more options type `help plot` in the Command Window.

Exercise 2

Use the `linspace` command or the colon operator `:` to create a vector `t` with entries $6, 6.3, 6.6, \ldots, 30$, and define the function $y = \dfrac{e^{t/15}\cos(2t)}{0.6t^3 + 9}$ (make sure you use ";" to suppress the output for both `t` and y).

(a) Plot the function y in black and include a title with the expression for y.

(b) Make the same plot as in part (a), but rather than displaying the graph as a curve, show the unconnected data points. To display the data points with small circles, use `plot(t,y,’o’)`. Now combine the two plots with the command `plot(t,y,’o-’)` to show the line through the data points as well as the distinct data points.

The plot3 command

Space curves can be easily plotted in MATLAB using the `plot3` command. For instance, suppose we want to plot the space curve defined by the parametric equations

$$x = \sin(3t)\cos(t), \quad y = \sin(3t)\sin(t), \quad z = \cos(t)$$

in the interval $0 \le t \le 2\pi$.
We first define the vector t with entries in the given interval (here we will use the colon operator with a stepsize of 0.01) and compute x, y and z as functions of t.

```
>> t = 0:0.01:2*pi;
>> x=sin(3*t).*cos(t);
>> y=sin(3*t).*sin(t);
>> z=cos(t);
```

We then use the `plot3` command to plot the space curve:

```
>> plot3(x,y,z);
>> grid on;
```

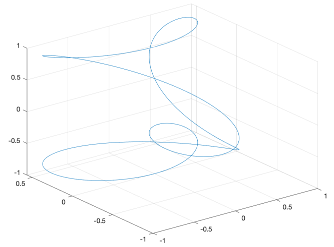

Figure 4

The command `grid on` adds a grid to the plot. The resulting plot is shown in Fig. 4.

Exercise 3

Use the command `plot3(x,y,z)` to plot the circular helix $x = 9\cos(5t)$, $y = 9\sin(5t)$, $z = 6t$, $0 \le t \le 10$.
Add a grid to the plot using the command `grid on`.
NOTE: Use semicolon to suppress the output when you define the vectors t, x, y and z. Make sure you use enough points for your graph so that the resulting curve is nice and smooth.

Combining multiple plots

By default new plots clear existing plots and reset axes properties, such as the title. However there are different ways to combine multiple plots in the same axes.

The `hold on` command

Suppose we want to plot the functions $y = \cos(x)$ and $y = \cos(2x)$ in the interval $-\pi \leq x \leq \pi$. We can use the `hold on` command to concatenate two plots. Then reset the `hold` state to `off`.

```
x=linspace(-pi,pi,100);  %define x-values in the interval [-pi,pi]
y1=cos(x);          %compute cos(x) and store it in the vector y1
plot(x,y1)          %plot y1 vs x
hold on;            %hold the current plot so that subsequent
                    %graphs can be added
y2=cos(2*x);         %compute cos(2x) and store it in the vector y2
plot(x,y2,'r.')     %plot y2 vs x
axis tight;         %axis limits are set to the range of the data
grid on       %add a grid
hold off      %release the plot
```

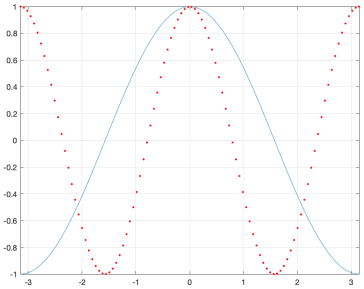

Figure 4

Overlay plots in a single plot command

We can combine multiple plots by creating them using a single plot command. For instance, the commands below give the same plot obtained above (Fig. 4).

```
plot(x,y1,x,y2,'r.')     %plot y1 vs x and y2 vs x
axis tight;
grid on
```

Exercise 4

Plot $y = \sin(4x)$ in red with a solid line and the Taylor polynomial approximation
$z = 4x - \frac{32}{3}x^3$ in blue with a dashed line for $-\frac{\pi}{4} \leq x \leq \frac{\pi}{4}$ on the same plot.
Hint: Use `plot(x,y,'r',x,z,'--')`.
Add a grid to the plot using the command `grid on`.
Type `axis tight` to set the axis limits to the range of the data.
NOTE: Use semicolon to suppress the output when you define the vectors x, y and z. Make sure you use enough points for your graph so that the resulting curves are nice and smooth.

Scripts and Functions

★ Files containing MATLAB commands are called `m`-files and have a `.m` extension. They are two types:

1. A *script* is simply a collection of MATLAB commands gathered in a single file. It is useful for automating series of commands, such as computations that you have to perform repeatedly from the command line. The value of the data created in a script is still available in the Command Window after execution. To create a new script select **New Script** in the upper left corner of the **Home** toolbar. In the MATLAB text editor window enter the commands as you would in the Command window. To save the file click on the save button 💾. Variable defined in a *script* are accessible from the command window.

2. A *function* is similar to a script, but can accept and return arguments. Unless otherwise specified any variable inside a function is local to the function and not available in the command window.
   In the upper left corner of the Home toolbar, click on **New** and select **function** from the pulldown menu. A MATLAB text editor window will open with the following predefined commands

   ```
   function [outputArg1,outputArg2] = untitled(inputArg1,inputArg2)
   %UNTITLED Summary of this function goes here
   %   Detailed explanation goes here
   outputArg1 = inputArg1;
   outputArg2 = inputArg2;
   end
   ```

   The “outputArg” are the output arguments, while the “inputArg” are the input arguments. The lines beginning with % are to be replaced with comments describing what the functions does. The command(s) defining the function must be inserted after these comments and before `end`.

   To save the file proceed similarly to the Script M-file.

Use a function when a group of commands needs to be evaluated multiple times.

★ Examples of script/function:

1. **script**

   `myplot1.m`

   ```
   x=0:.01:2;                                    % x-values
   y=(x.^2-sin(pi.*x)+exp(x))./(x-1);            % y-values
   plot(x,y,'r-','LineWidth',2);  % plot in red with wider line
   axis([0,2,-10,20]); grid on;  % set range and add grid
   title('f(x)=(x^2-sin(\pi x)+e^x)/(x-1)');     % add title
   xlabel('x'); ylabel('y');                     % add labels
   ```

2. **script+function** (one single file)

   `myplot2.m` (driver script)

   ```
   x=0:.01:2;                            % x-values
   y=myfunction(x);                  % evaluate myfunction at x
   ```

```
plot(x,y,'r-','LineWidth',2);      % plot in red
axis([0,2,-10,20]); grid on;     % set range and add grid
title('f(x)=(x^2-sin(\pi x)+e^x)/(x-1)');     % add title
xlabel('x'); ylabel('y');              % add labels
%-------------------------------
function y=myfunction(x)                % defines function
y=(x.^2-sin(pi.*x)+exp(x))./(x-1);     % y-values
end
```

3. **script+function** (two separate files)

   `myplot3.m` (driver script)

```
x=0:.01:2;                                          % x-values
y=myfunction(x);                     % evaluate myfunction at x
plot(x,y,'r-','LineWidth',2);                       % plot in red
axis([0,2,-10,20]); grid on;   % set range and add grid
title('f(x)=(x^2-sin(\pi x)+e^x)/(x-1)');     % add title
xlabel('x'); ylabel('y');                           % add labels
```

   `myfunction.m` (function)

```
function y=myfunction(x)            % defines function
y=(x.^2-sin(pi.*x)+exp(x))./(x-1);            % y-values
end
```

In case 3 `myfunction.m` can be used in any other `m`-file (just as other predefined MATLAB functions). In case 2 `myfunction.m` can be used by any other function in the same `m`-file (`myplot2.m`) only. Use 2 when dealing with a single project and 3 when a function is used by several projects.
★ Note that the construct *function+function* in one single file is also allowed.
★ It is convenient to add descriptive comments into the script file. Anything appearing after `%` on any given line is understood as a comment (in green in the MATLAB text editor).
★ To execute a script simply enter its name (without the .m extension) in the Command Window (or click on the SAVE & RUN button ▶).

The function `myfunction` can also be used independently if implemented in a separate file `myfunction.m`:

```
>> x=2; y=myfunction(x)
y =
   11.3891
```

A script can be called from another script or function (in which case it is local to that function).

If any modification is made, the script or function can be re-executed by simply retyping the script or function name in the Command Window (or use the up-arrow on the keyboard to browse through past commands).

IMPORTANT REMARK
By default MATLAB saves files in the Current Folder. To change directory use the Current Directory box on top of the MATLAB desktop.

Exercise 5

The general solution to the differential equation $\dfrac{dy}{dx} = -9x - 17x^2 - 14\cos(x)$ is

$$y(x) = -\frac{9}{2}x^2 - \frac{17}{3}x^3 - 14\sin(x) + C \quad \text{with } y(0) = C.$$

The goal of this exercise is to write a `script` file to plot the solutions to the differential equation in the interval $0 \leq x \leq 8$, with initial conditions $y(0) = 640, 1070, 1600$.
The `script` file should have the structure `script+function` (similarly to the M-file `myplot2.m` Example 2). Call the file `ex5.m`. The function that defines $y(x)$ must be included in the same file (note that the function defining $y(x)$ will have two input arguments: x and C).
Your M-file should have the following structure (fill in all the `??` with the appropriate commands):

```
x =  ?? ;   % define the vector x in the interval [0,8]
y1 = f(??); % compute the solution with C = 640
y2 = f(??); % compute the solution with C = 1070
y3 = f(??); % compute the solution with C = 1600
plot(??) % plot the three solutions with different line-styles
title(??)  % add a title
legend(??) % add a legend

function y = f(x,C)
y = ?? % fill-in with the expression for the general solution
end
```

Plot the graphs in the same window and use different color and/or line-styles for each graph. To plot the graphs in the same window you can use the command `hold on` or use the plot command similarly to Exercise 4.
Add the title 'Solutions to $dy/dx = -9x - 17x^2 - 14\cos(x)$'.
Add a legend with the list of C values used for each graph.
(Type `help plot` for a list of the different line-styles, and `help legend` for help on how to add a legend.) Include both the M-file and the plot in your report.
NOTE: the only output of the script file should be the graph of the three curves. Make sure you use enough points so that the curves are nice and smooth.

Anomymous function

A function file can contain a lot more than a simple evaluation of a function $f(x)$ or $f(t, y)$. But in simple cases $f(x)$ or $f(t, y)$ can simply be defined as **Anonymous Function**.

For instance, if we want to define the function $f(t, y) = t^2 - y$, we can write the function file `f.m` containing

```
function dydt = f(t,y)
 dydt = t^2-y;
end
```

and, in the command window, we can evaluate the function at different values:

```
>> f(2,1)     % evaluate the function f at t = 2 and y = 1
ans =
      3
```

or we can define the function directly on the command line as an *anonymous* function. An anonymous function is a function that is *not* stored in a program file, but is associated with a variable whose data type is `function_handle`. Anonymous functions can accept inputs and return outputs, just as standard functions do. However, they can contain only a single executable statement. For example:

```
>> f = @(t,y)(t^2-y) % define f(t,y)=t^2-y as anonymous function
```

The parentheses ( ) immediately after the `@` operator include the function input arguments. This anonymous function accepts two inputs t and y and implicitly returns a single output, the value of $t^2 - y$:

```
>> f(2,1)     % evaluate the function f at t = 2 and y = 1
ans =
      3
```

★ CAUTION!

- The names of script or function M-files must begin with a letter. The rest of the characters may include digits and the underscore character. You may not use periods in the name other than the last one in '.m' and the name cannot contain blank spaces.
- Avoid name clashes with built-in functions. It is a good idea to first check if a function or a script file of the proposed name already exists. You can do this with the command `exist('name')`, which returns zero if nothing with name `name` exists.
- *NEVER name a script file or function file the same as the name of the variable it computes.* When MATLAB looks for a name, it first searches the list of variables in the workspace. If a variable of the same name as the script file exists, MATLAB will never be able to access the script file.

Exercise 6

(a) Enter the function $g(x,y) = \dfrac{x^3}{y^7} + \dfrac{\cos(4xe^{8y})}{x^2+2}$ as an *anonymous* function. Evaluate the function at $x = 5$ and $y = -4$ by entering `g(5, -4)` in the command window.

(b) Type `clear g` to clear the value of the function from part (a). Now write a *function* M-file for the function $g(x,y) = \dfrac{x^3}{y^7} + \dfrac{\cos(4xe^{8y})}{x^2+2}$. Save the file as `g.m` (include the M-file in your report).
Evaluate the function at $x = 5$ and $y = -4$ by entering `g(5,-4)` in the command window.