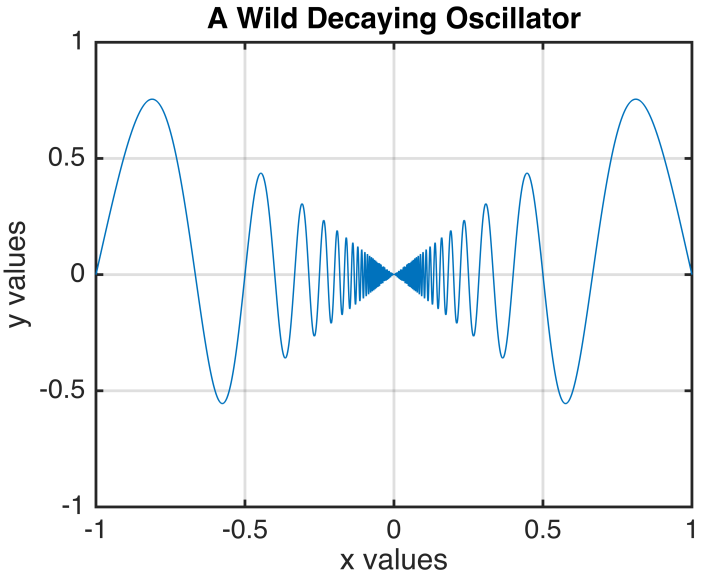
UBC MECH 221: MATLAB Tutorial 1

Numbers, Arrays, Vectorization, Plotting and Functions




```
>> x = [-1:0.0001:-0.001, 0.001:0.0001:1]; % Avoid the value 0
>> y = x .* exp(-0.1 * x.^2) .* sin(2 * pi ./ x);
>> plot(x,y)
>> title('A Wild Decaying Oscillator'), xlabel('x values'), ylabel('y values')
>> ylim([-1,1]), grid('on')
```

Patrick Walls
September 27, 2016

Contents

1 Introduction

MATLAB is the *matrix laboratory* and this tutorial is a tour of the basics: numbers, arrays, vectorization, plotting and functions. This document is designed to guide you through the basic commands by providing example code to execute in MATLAB for yourself. The best way to learn from these notes is to open a new MATLAB workspace, enter the commands presented, observe the output, try variations of each command, refer to the documentation as needed and test your new skills by working on the exercises.

Learning Goals

1. **Numbers**: arithmetic operations, constants π and e
2. **Arrays**: creating and transforming vectors and matrices
3. **Vectorization**: element-by-element calculations applied to arrays
4. **Plotting**: plotting functions, adding labels, titles and legends, adding style to plots
5. **Functions**: writing functions with input and output and creating M-files

Instructions

1. Find a computer with MATLAB or Octave installed (or use Octave online as described below).
2. MATLAB commands are presented in the tutorial in the format:

   ```
   >> x = linspace(-pi,pi,1000); y = sin(x); plot(x,y);
   ```

   where the symbol `>>` represents the MATLAB prompt.
3. Enter each command listed in the sections below. It is important to type these commands for yourself and to not copy and paste the text. You must memorize the basic commands and functions.
4. Understand the output and try variations of each command to test your understanding.

Resources

1. MATLAB is now freely available to download for registered UBC students:

   `students.engineering.ubc.ca/success/software`
2. Visit `mathworks.com/help/matlab` for more documentation.
3. Octave is a free, open-source MATLAB clone and is sufficient for this assignment. Octave is available to download at `gnu.org/software/octave`.
4. You can also use Octave online at `octave-online.net`.

2 Numbers

Arithmetic operations in MATLAB are quite straightforward. For more information about elementary mathematics in MATLAB, check out the documentation:

`http://www.mathworks.com/help/matlab/elementary-math.html`

2.1 Basic Operations

Add numbers:

```
>> 848 + 1246
```

Subtract numbers:

```
>> 1201 - 398
```

Multiply numbers:

```
>> 34 * 81
```

Divide numbers:

```
>> 7 / 3
```

Compute powers:

```
>> 2^8
```

Compute fractional powers:

```
>> 5^(1/2)
```

Combine these operations to compute $\frac{2(3-2/7)}{\sqrt{1+8^3}}$:

```
>> 2 * (3 - 2/7) / (1 + 8^3)^(1/2)
```

2.2 Constants π and e

The mathematical constants π and e are entered as `pi` and `exp(1)` respectively (the exponential function e^x is entered as `exp` in MATLAB therefore $e = e^1$ is `exp(1)`):

```
>> pi, exp(1)
```

2.3 Exercises

1. Compute $1 + 2\sqrt{5}$.
2. Compute $\pi^2/6$.
3. Compute $\dfrac{e + e^{-1}}{2}$.
4. The Taylor series of e^x centered at 0 is given by

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

Compute the 5th partial sum of the series at $x = 1/2$

$$\sum_{n=0}^{5} \frac{(1/2)^n}{n!} = 1 + (1/2) + \frac{(1/2)^2}{(1)(2)} + \frac{(1/2)^3}{(1)(2)(3)} + \frac{(1/2)^4}{(1)(2)(3)(4)} + \frac{(1/2)^5}{(1)(2)(3)(4)(5)}$$

and compare it to the value $\sqrt{e}$.

3 Arrays

The following is a selection of methods for creating, transforming and applying functions to vectors, matrices and more general n-dimensional arrays. For more array commands and examples, check out the documentation:

`http://www.mathworks.com/help/matlab/matrices-and-arrays.html`

3.1 Creating Arrays

Create a row vector and assign it to the variable `x`:

```
>> x = [1,2,3]
```

Create a column vector and assign it to the variable `x`:

```
>> x = [1;2;3]
```

Create a row vector of values from 0 to 10 by increments of 1 and assign it to the variable `x`:

```
>> x = 0:10
```

Create a row vector of values from 0 by increments of 0.1 up to but not exceeding π and assign it to the variable `x`:

```
>> x = 0:0.1:pi
```

Create a row vector of 11 evenly spaced values from 0 to 1 and assign it to the variable `x`:

```
>> x = linspace(0,1,11)
```

Create a matrix and assign it to the variable `A`:

```
>> A = [1,2,3,4,5; 6,7,8,9,10; 11,12,13,14,15]
```

Equivalently:

```
>> A = [1:5; 6:10; 11:15]
```

Create 2 column vectors and put them into the columns of a matrix `A`:

```
>> c1 = [1;2], c2 = [3;4], A = [c1,c2]
```

Create 2 row vectors and put them into the rows of a matrix `A`:

```
>> r1 = [0,-1], r2 = [5,7], A = [r1;r2]
```

Create a 4 by 4 matrix of zeros:

```
>> zeros(4)
```

Create a 2 by 5 matrix of zeros:

```
>> zeros(2,5)
```

Create the identity matrix of size 6:

```
>> eye(6)
```

3.2 Array Indexing

Create a vector `x` and return the value at index 3:

```
>> x = -5:5, x(3)
```

Create a vector `v` and return the values at indices 2 through 6:

```
>> v = -10:2:10, v(2:6)
```

Create a matrix `A` and return the value in row 3 and column 2:

```
>> A = [-1:2:3; 2:4; -6:3:0; 4:-2:0], A(3,2)
```

Create a matrix `M` and return column 2 of `M`:

```
>> M = [1,0,-2; 7,5,-1; 1,4,-8], M(:,2)
```

Assign a fourth row to `M`:

```
>> M(4,:) = [3,-4,0]
```

Return the values of `M` which are in rows 2 and 4 and columns 1 and 3:

```
>> M([2,4],[1,3])
```

3.3 Array Functions

Create a vector and compute its length:

```
>> v = 0:3:100, length(v)
```

Create a vector and compute its minimum and maximum values:

```
>> w = [2,-1,2,-5], min(w), max(w)
```

Create a vector and compute the sum of the entries:

```
>> x = 1:10, sum(x)
```

The array functions `max`, `min` and `sum` operate on the columns of matrices and return a row vector of the results:

```
>> A = [1:3;4:6]
>> max(A), min(A), sum(A)
```

The `length` function returns the largest dimension of a matrix:

```
>> A = zeros(4,5)
>> length(A)
```

3.4 Random Numbers and Matrices

There are several MATLAB functions for generating arrays with random entries. For more information about random number generation, check out the documentation:

`http://www.mathworks.com/help/matlab/random-number-generation.html`

Create a 2 by 7 matrix of random numbers which are uniformly distributed in $(0, 1)$:

```
>> A = rand(2,7)
```

Create a 5 by 6 matrix of random numbers which are normally distributed (with mean 0 and standard deviation 1):

```
>> B = randn(5,6)
```

Create a 3 by 4 matrix of random integers sampled from the interval $[0, 9]$:

```
>> M = randint(3,4,[0,9])
```

3.5 `clc` and `clear`

It is good practice to keep a clean workspace. See the documentation to learn more:

`http://www.mathworks.com/help/matlab/matlab_prog/strategies-for-efficient-use-of-memory.html`

Clear the command window:

```
>> clc
```

Remove all variables from the current workspace:

```
>> clear
```

3.6 Exercises

1. Create the vector $\boldsymbol{v} = [-8, 5, 3, 1]$.
2. Create the matrix
$$A = \begin{bmatrix} 4 & 0 & 7 & -6 \\ 5 & 7 & -1 & 2 \\ 9 & -1 & 0 & 5 \end{bmatrix}$$
3. Create a vector of 25 evenly spaced values from -1 to 1.
4. Create a 4 by 5 matrix whose entries are random integers sampled from the interval $[-2, 5]$.
5. Create the 5 by 5 identity matrix and select the last column.

6. Create the vector $\boldsymbol{v} = [-17, -14, -11, -8, -5, -2, 1, 4, 7, 10, 13, 16, 19, 22, 25]$.

7. Compute the sum

$$\sum_{n=0}^{1000} 2n - 1$$

by applying the array function `sum` to the vector $[-1, 1, 3, 5, 7, \ldots, 1999]$.

4 Vector and Matrix Operations

MATLAB is the *matrix laboratory* and is equipped with many tools for solving problems in linear algebra. We only introduce the most basic operations in this tutorial and you can check out the documentation to learn much more:

`http://www.mathworks.com/help/matlab/linear-algebra.html`

4.1 Vector Operations

Add random vectors:

```
>> v = rand(1,3), w = rand(1,3)
>> v + w
```

Multiply a random vector by a scalar:

```
>> v = randint(4,1,[0,10]), s = -5
>> s*v
```

Compute the dot product of two vectors:

```
>> v1 = [1,0,-1,2], v2 = [3,-2,2,1]
>> dot(v1,v2)
```

Compute the cross product of two vectors in 3D:

```
>> w1 = [1,0,1], w2 = [-2,3,1]
>> cross(w1,w2)
```

Compute the norm of a random vector:

```
>> x = randint(1,5,[0,3])
>> norm(x)
```

4.2 Matrix Operations

Add random matrices:

```
>> A = randint(3,2,[-9,9]), B = randint(3,2,[-9,9])
>> A + B
```

Multiply random matrices:

```
>> A = randint(2,2,[-3,3]), B = randint(2,2,[-3,3])
>> A * B
```

Multiply a matrix by a scalar:

```
>> A = [-1,3; 4,2]
>> 3 * A
```

Create a matrix `A` and compute the transpose:

```
>> A = [0,0,0; 1,1,1; 2,2,2]
>> A'
```

Compute the determinant and inverse of a random matrix:

```
>> A = rand(3,3)
>> det(A), inv(A)
```

Solve the system of equations:

$$\begin{array}{rcrcrcr} 2x & + & 3y & - & z & = & 1 \\ x & + & 5y & + & 4z & = & 0 \\ -x & + & y & - & 2z & = & 3 \end{array}$$

```
>> A = [2,3,-1; 1,5,4; -1,1,-2], b = [1;0;3]
>> x = linsolve(A,b)
```

4.3 `clc` and `clear`

Clear the command window:

```
>> clc
```

Remove all variables from the current workspace:

```
>> clear
```

4.4 Exercises

1. The projection of a vector $\boldsymbol{v}$ onto a vector $\boldsymbol{w}$ is given by the formula

$$\text{proj}_{\boldsymbol{w}}(\boldsymbol{v}) = \frac{\boldsymbol{v} \cdot \boldsymbol{w}}{||\boldsymbol{w}||^2}\boldsymbol{w}$$

   Compute the projection of $\boldsymbol{v}$ onto $\boldsymbol{w}$:

   (a) $\boldsymbol{v} = [1, 0, -1]$, $\boldsymbol{w} = [2, 1, 1]$

   (b) $\boldsymbol{v} = [8, 3, -1, 4]$, $\boldsymbol{w} = [3, -2, 0, 5]$

2. Compute the determinant and inverse of the matrix

$$A = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 1 \\ -1 & 0 & -1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

3. Solve the system of equations:

$$\begin{array}{rcrcrcr} x & + & 2y & + & z & = & -3 \\ -x & + & y & + & 3z & = & 4 \\ x & & & + & 2z & = & 5 \end{array}$$

5 Vectorization

MATLAB is optimized for operations involving vectors and matrices. In some programming languages, loops are required to compute the values of a function applied to elements in a list. Vectorization refers to MATLAB's built-in vector and matrix operations which apply functions to arrays element-by-element all at once. Explore more in the documentation:

`http://www.mathworks.com/help/matlab/matlab_prog/vectorization.html`

5.1 Vectorized Arithmetic Operations

Addition and subtraction are already vectorized:

```
>> v = 1:4, w = 5:8
>> v + w, v - w
```

Multiply the elements of two vectors using the `.*` operator:

```
>> x = 1:3, y = 2:4
>> x .* y
```

Divide the elements of two vectors using the `./` operator:

```
>> a = 1:5, b = ones(1,5)*10
>> a ./ b
```

Create a vector `v` and compute the cube of every element using the `.^` operator:

```
>> v = 0:12
>> v .^ 3
```

Use the `.*` operator and the `sum` function to compute the dot product of vectors (and compare to the built in function `dot`):

```
>> v = [4,3,-1], w = [2,1,5]
>> sum(v .* w)
>> dot(v,w)
```

5.2 Vectorized Mathematical Functions

A vectorized mathematical function takes an array as input and computes the value of the function for each entry in the array. For example, a vectorized version of $\sin(x)$ applied to the vector $\boldsymbol{v} = [0, \pi/2, \pi, 3\pi/2, 2\pi]$ would return

$$\sin([0, \pi/2, \pi, 3\pi/2, 2\pi]) = [0, 1, 0, -1, 0]$$

Find a complete list of all the familiar mathematical functions:

`http://www.mathworks.com/help/matlab/elementary-math.html`

Verify the example above:

```
>> v = [0, pi/2, pi, 3*pi/2, 2*pi], sin(v)
```

Similarly:

```
>> v = [0, pi/2, pi, 3*pi/2, 2*pi], cos(v)
```

Compute the square root of the values from 1 to 10:

```
>> v = 1:10, sqrt(v)
```

Equivalently:

```
>> v = 1:10, v.^(0.5)
```

Compute e^x for 6 evenly spaced values of x from 0 to 1:

```
>> x = linspace(0,1,6), exp(x)
```

Compute the natural log, $\ln(x)$, for 5 evenly spaced values of x from 1 to e:

```
>> w = linspace(1,exp(1),5), log(w)
```

Combine vectorized operations to compute $xe^{-x^2}\cos(x)$ for values $x \in [0, 1]$:

```
>> x = linspace(0,1,11)
>> x .* exp(-x.^2) .* cos(x)
```

5.3 `clc` and `clear`

Clear the command window:

```
>> clc
```

Remove all variables from the current workspace:

```
>> clear
```

5.4 Exercises

1. Create an array of values $\sin(x^2)$ for $x = 0, 0.2, 0.4, 0.6, 0.8, 1$.
2. Create an array of values $\ln(1 + x)$ for $x = 0, 0.1, 0.2, \ldots, 1$.
3. Create an array of values $x\cos(x)$ for 21 evenly spaced values of x in the interval $[-2, 2]$.
4. In 1734, the Swiss mathematician Leonard Euler proved the beautiful formula

$$\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6}$$

Compute the 1000th partial sum

$$\sum_{n=1}^{1000} \frac{1}{n^2}$$

and then compute the value $\pi^2/6$ and compare. (Hint: Create the vector $[1, 2, \ldots, 1000]$ and use the operator `.^` and the array function `sum`.)

6 Plotting

MATLAB has a very rich plotting library. In this section, we'll cover the basic 2D plotting commands and you can learn more in the documentation:

`http://www.mathworks.com/help/matlab/graphics.html`

6.1 Basic Plotting Commands

The procedure to create a 2D plot is quite simple:

1. Create an array `x` of x values
2. Create an array `y` of y values (the same length as `x`)
3. Use the command `plot(x,y)`

This means that we'll be using the vectorized array functions and operations from the last section to build the array `y` from the array `x` and then apply `plot(x,y)`.

Plot the straight line connecting the points $(0, 0)$, $(1, 2)$ and $(2, 4)$:

```
>> x = [0,1,2], y = [0,2,4], plot(x,y)
```

Create a vector `x` of 50 points from 0 to 2π, compute the array of y values `y = sin(x)`, and plot (and notice that we use the semicolon `;` to suppress the output `x` and `y`):

```
>> x = linspace(0,2*pi,50); y = sin(x); plot(x,y)
```

Notice that MATLAB is simply connecting the dots with coordinates defined by the arrays `x` and `y`. This means that more points create a smoother curve and fewer points create a jagged curve. For example:

```
>> x = linspace(0,2*pi,100); y = sin(x); plot(x,y)
>> x = linspace(0,2*pi,10); y = sin(x); plot(x,y)
```

Plot the function $y = x \cos x$ for $x \in [-5, 5]$:

```
>> x = -5:0.1:5; y = x .* cos(x); plot(x,y)
```

6.2 Adding Styles, Titles, Labels and Legends

There are MATLAB commands to specify line color, line style, marker type, title, labels, etc. Check out the documentation:

`http://www.mathworks.com/help/matlab/ref/plot.html`

Plot the curve $y = 1 - x^2$ for $x \in [-2, 2]$ as a red line with a circle at each data point defined by the arrays `x` and `y`:

```
>> x = -2:0.1:2; y = 1 - x.^2; plot(x,y,'r-o')
```

Plot $y = e^{-x^2}$ for $x \in [-5, 5]$ as a yellow dashed line with a cross at each data point:

```
>> x = linspace(-5,5,200); y = exp(-x.^2); plot(x,y,'y--x')
```

Plot multiple lines simultaneously with different styles:

```
>> x = 0:0.01:2; y = x .* exp(-7*x);
>> plot(x,y,x,0.5*y,'c--',x,2*y,'r.',x,3*y,'k*')
```

Add a title, axis labels, a legend and set display limits:

```
>> x = linspace(-3*pi,3*pi); y1 = sin(x); y2 = cos(x); plot(x,y1,x,y2)
>> title('Waves'), xlabel('x values'), ylabel('y values')
>> legend('Sine', 'Cosine'), xlim([-3*pi,3*pi]), ylim([-2,2])
```

Note that you must keep the figure open while entering the commands `title`, `xlabel`, etc. to add these styles to the plot.

6.3 Parametric Equations

Plot a pentagon, set the display limits wide enough to show the complete figure and set the x and y units equal to achieve a 1:1 aspect ratio:

```
>> t = linspace(0,2*pi,7); x = cos(t); y = sin(t);
>> plot(x,y), xlim([-2,2]), ylim([-2,2]), axis('equal')
```

Plot an octagon with filled blue dots at each vertex:

```
>> t = linspace(0,2*pi,9); x = cos(t); y = sin(t);
>> plot(x,y,'o','MarkerFaceColor','b')
>> xlim([-2,2]), ylim([-2,2]), axis('equal')
```

Plot a circle:

```
>> t = linspace(0,2*pi,1000); x = cos(t); y = sin(t);
>> plot(x,y), xlim([-2,2]), ylim([-2,2]), axis('equal')
```

Plot the parametric curve defined by $x(t) = \cos 3t$ and $y(t) = \sin 2t$ for $t \in [0, 2\pi]$:

```
>> t = linspace(0,2*pi,1000); x = cos(3*t); y = sin(2*t);
>> plot(x,y), xlim([-2,2]), ylim([-2,2]), axis('equal')
```

6.4 `clc` and `clear`

Clear the command window:

```
>> clc
```

Remove all variables from the current workspace:

```
>> clear
```

6.5 Exercises

1. Plot $y = \dfrac{1}{1+x^2}$ for $x \in [-5, 5]$.
2. The square wave is defined by an infinite sum

$$f(t) = \frac{4}{\pi} \sum_{k=1}^{\infty} \frac{\sin(2\pi(2k-1)t)}{2k-1}$$

   Plot the function consisting of the first 3 terms of the sum

$$y = \frac{4}{\pi}\left(\sin(2\pi t) + \frac{1}{3}\sin(6\pi t) + \frac{1}{5}\sin(10\pi t)\right)$$

   for $t \in [0, 3]$. Include the title "First 3 terms of the square wave".
3. Plot the parametric curve defined by $x = \cos(t)$ and $y = \cos(t)\sin(t)$ for $t \in [0, 2\pi]$.
4. Plot $y = \dfrac{1}{1+e^{-kt}}$ for $t \in [-5, 5]$ for $k = 1, 2, 3$ in the same figure and include a legend.

7 Functions

MATLAB functions allow you to write code and reuse it with different input parameters. For more information about functions, check out the documentation:

`http://www.mathworks.com/help/matlab/functions.html`

7.1 Creating a new function

Create a new function by clicking New > Function in the Home toolbar or just click the + symbol in the Editor window. The format of a function follows a specific template. For example, here is a simple function which computes the area of a triangle of a given base and height:

```
function area = triangle(base, height)
% Compute the area of the triangle
% Input:
%   base: positive number for the base of the triangle
%   height: positive number for the height of the triangle
% Output:
```

```
%   area: base * height / 2 computes the area of the triangle

area = base * height / 2;

end
```

where:

- `function` is a keyword that marks the start of a function definition
- `area` is the output of the function
- `triangle` is the name of the function (and **must** be the same as the name of the M-file)
- `base` and `height` are input parameters
- lines beginning with `%` are ignored by MATLAB and we use these lines to describe the functions purpose, its inputs and its outputs
- `end` is a keyword that marks the end of the function definition

Save M-files to your current working directory. You can see the current working directory in the "Current Folder" window in your MATLAB workspace or you can enter the "Print Working Directory" command:

```
>> pwd
```

Always name your function the same as the M-file it's contained in. For the function called `triangle`, written to a M-file called `triangle.m` which is saved to the current directory, the function can be called at the command line:

```
>> triangle(2,3)
```

7.2 Examples

Write a function called `projection` (saved to a M-file named `projection.m`) which computes the projection of a vector onto another:

```
function proj = projection(v,w)
% Compute the projection of v onto w
% Input:
%   v, w: any vectors of the same size
% Output:
%   proj: the projection of v onto w

proj = dot(v,w) / dot(w,w) * w;

end
```

We can call the function in the command line:

```
>> projection([1,1,2],[-1,0,1])
```

Write a function called `polygon` (saved to a M-file named `polygon.m`) which plots a polygon with N sides:

```
function polygon(N)
% Plot a polygon with N sides

t = linspace(0,2*pi,N + 1);
x = cos(t);
y = sin(t);
plot(x,y)
xlim([-2,2]), ylim([-2,2]), axis('equal')

end
```

Notice that this function is so simple that we can don't need a lot of documentation to explain the input. The function simply produces a plot and does not have an output value. We can call the function in the command line:

```
>> polygon(3)
>> polygon(8)
>> polygon(50)
```

Write a function called `parametric_plot` which plots a the parametric curve defined by

$$x(t) = \cos(at) \ , \quad y(t) = \sin(bt) \ , \quad t \in [0, 2\pi k]$$

```
function parametric_curve(a,b,k)
% Plot the parametric curve defined by
% x(t)=cos(at), y(t)=sin(bt), t in [0,2*pi*k]

% The number of points depends on k (to ensure a smooth curve for any k)
t = linspace(0,2*pi*k,1000 * k);
x = cos(a*t);
y = sin(b*t);
plot(x,y), xlim([-2,2]), ylim([-2,2]), axis('equal')

end
```

We can call the function in the command line:

```
>> parametric_curve(2,3,1)
>> parametric_curve(10,3,7)
```

Write a function called `max_min_eigenvalues` which computes the maximum and minimum eigenvalues of AA^T of a given matrix A:

```
function [max_eig_val,min_eig_val] = max_min_eigenvalues(A)
% Compute the maximum and minimum eigenvalues of the matrix A * transpose(A)
% Input:
%   A: any matrix
% Output:
%   max_eig_val: the largest eigenvalue of the matrix A * transpose(A)
%   min_eig_val: the least eigenvalue of the matrix A * transpose(A)

[eigenvectors, eigenvalues] = eig(A * transpose(A));
eigenvalues = diag(eigenvalues);
max_eig_val = max(eigenvalues);
min_eig_val = min(eigenvalues);

end
```

We can call the function in the command line:

```
>> [max_ev, min_ev] = max_min_eigenvalues([1,1;2,3;2,2])
```

Notice that we need to assign the output the variable names `max_ev` and `min_ev` to capture the output of the function. In particular, compare to the following function call without variables to capture the output:

```
>> max_min_eigenvalues([1,1;2,3;2,2])
```

7.3 Exercises

1. Write a function which takes an input `a` and returns the solution of the system of equations:
$$\begin{array}{rcrcrcl} x & + & 2y & + & z & = & 1 \\ x & + & y & - & z & = & 2 \\ x & - & y & + & 2z & = & a \end{array}$$

2. Write a function which takes two inputs `n` and `tlimit` and plots the parametric curve defined by:
$$x = \cos^3(2\pi t) + \cos(2\pi n t)$$
$$y = \sin^3(2\pi t) + \sin(2\pi n t)$$

for t in the interval `[0,tlimit]`. Use the commands `xlim`, `ylim` and `axis('equal')` to properly display the figure.

3. Write a function which takes 3 vectors $\boldsymbol{u}$, $\boldsymbol{v}$ and $\boldsymbol{w}$ in 3D and returns the volume of the parallelpiped spanned by these vectors:

$$V = |\boldsymbol{u} \cdot (\boldsymbol{v} \times \boldsymbol{w})|$$