1 Matlab

1.1 Using matlab as a calculator

We will be introducing MATLAB and demonstrating how it can be used as a powerful calculator. Note that the MATLAB “help” function is very comprehensive. To use it, simply type “help ” and your topic of choice. For example, if interested in MATLAB multiplication, you would type “help *”. If interested in the function cos, simply type “help cos”.

1.2 Vectors and Matrices

In general, vectors are one dimensional arrays of objects (e.g. real numbers) and matrices are two dimensional arrays. Vectors and matrices in MATLAB are usually arrays of real numbers or integers. There are a few methods for setting up vectors and matrices. One way is to specify the values one by one:

\[
\begin{align*}
  r &= [1, 2, 3, 4, 5]; \quad \text{create a } 1 \times 5 \text{ row vector} \\
  c &= [1; 2; 3; 4; 5]; \quad \text{create a } 5 \times 1 \text{ column vector} \\
  A &= [1, 2; 3, 4]; \quad \text{create a } 2 \times 2 \text{ matrix}
\end{align*}
\]

The above will set a row vector \( r = (1, 2, 3, 4, 5) \), a column vector \( c \) and a \( 2 \times 2 \) matrix \( A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \). 

Another common way to set up vectors is the format: start:step:end. For example, the row vector \( r = (1, 2, 3, 4, 5) \) can be set using:

\[
r = 1:0.5:3
\]

Note that inside the brackets, when we separate two numbers (or matrices) by a space or comma, we tell MATLAB to concatenate the two numbers/matrices in a row. When we separate them by a semicolon, we tell matlab to stack them on top of each other in a column. For example, we can set the matrix \( A \) above using either stacking or concatenating:

\[
\begin{align*}
  A &= [1, 2; 3, 4]; \quad \text{stacking (1,2) over (3,4)} \\
  \text{or} \\
  A &= [[1; 3], [2; 4]]; \quad \text{concatenating (1,3)’ next to (2,4)’}
\end{align*}
\]

Many MATLAB functions output objects corresponding to the supplied input. For example: \( x = \cos(t) \) will return a scalar, vector, or a matrix depending on \( v \): a scalar when \( v \) is a scalar, a vector when \( v \) is a vector and a matrix when \( v \) is a matrix. In general, \( x(i, j) = \cos(t(i, j)) \).

1.2.1 Multiplication in MATLAB

Note that the * symbol in MATLAB represents matrix multiplication: if \( A_{n \times k} \), \( B_{k \times m} \) are two matrices then \( C = A \times B \) is defined by matrix multiplication:
\( C(i,j) = \sum_{q=1}^{k} A(i,q)B(q,j) \)

When we need to multiply element by element, we use
\( C = A .* B. \)

### 1.3 Visualization

Let us see how to plot a graph of \( x \) vs. \( t \) using the \texttt{plot(t,x)} command and discuss the MATLAB command \texttt{subplot}.

```matlab
t = 0:0.01:1;
x = \cos(2*\pi*t);
plot(t,x)
```

![Plot of \( x \) vs. \( t \)](image)

**Figure 1: \texttt{plot(t,x)}**

### 1.4 Scripts

A script is a file whose lines represent MATLAB commands. When called, MATLAB reads the file line by line and tries to execute it just as if it was typed inside the command window. We will be using the file TrigDemo.m (handout) to demonstrate a simple script.

### 1.5 Functions

A function should be thought of as a black box that does some computations on the input and provides some output for it. For example, the function \( \exp(x) \) returns the value of \( e^x \). The user is usually interested only in the results and not in the specific implementation used to compute it. Writing a function of our own is very similar to writing a MATLAB script. The main difference is that functions are usually called up with a specific input and return a corresponding output. They also use their own local variables. Function variables are defined in the context of the function alone (local variables). Therefore, they are created within the function and disappear as soon as the function returns its output. This prevents mistakes that may result from using the same variable for different purposes. For example, two different functions may use the variable \( a \). The first would use it for representing a \texttt{MATRIX} and the other, for representing a scalar.

### 1.6 Loops

Sometimes we are interested in repeating a specific computation, each time potentially with a different value. For example, to compute the third Fibonacci number we use the first and the second, and to
compute the fourth number we use the second and the third. There is clearly a pattern in this
computation. If you are interested in the $n - th$ number you need to add the $n - 1$ and the $n - 2$
numbers together. The “for” statement is used in such cases. The syntax for a “for” loop is as follows:

```matlab
for i = A
    statements
end;
```

Which means: for each column of $A$, set the variable $i$ to this column and execute the statements.
For example, if $A = (1, 2, 3)$ then statements are executed three times. The first time with $i = 1$, the
second time with $i = 2$ and the third time with $i = 3$. If $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ then the statements are
executed twice. The first with $i = \begin{pmatrix} 1 \\ 3 \end{pmatrix}$ and the second with $i = \begin{pmatrix} 2 \\ 4 \end{pmatrix}$.

## 1.7 Conditionals

Sometimes we would like MATLAB to do something only when something else is true. This is done
with the “if” statement:

```matlab
if(expression)
    statements 1
else
    statements 2
end;
```

Which means that the first group of statements are executed when expression is true, otherwise the
second group of statements are executed. Here is an example of a function that returns 1 if the input
is positive and -1 if it is negative:

```matlab
function p = ispositive(x)
    if(x < 0)
        p = -1;
    else
        p = +1;
    end
return
```