MATLAB sessions: Laboratory 2

MAT 275 Laboratory 2
Matrix Computations and Programming in MATLAB

In this laboratory session we will learn how to

1. Create and manipulate matrices and vectors.

2. Write simple programs in MATLAB

NOTE: For your lab write-up, follow the instructions of LAB1.

Matrices and Linear Algebra

✿ Matrices can be constructed in MATLAB in different ways. For example the $3 \times 3$ matrix

\[
A = \begin{bmatrix}
8 & 1 & 6 \\
3 & 5 & 7 \\
4 & 9 & 2
\end{bmatrix}
\]

can be entered as

\[
>> A=[8,1,6;3,5,7;4,9,2]
\]

\[
A =
\begin{bmatrix}
8 & 1 & 6 \\
3 & 5 & 7 \\
4 & 9 & 2
\end{bmatrix}
\]

or

\[
>> A=[8,1,6;
3,5,7;
4,9,2]
\]

\[
A =
\begin{bmatrix}
8 & 1 & 6 \\
3 & 5 & 7 \\
4 & 9 & 2
\end{bmatrix}
\]

or defined as the concatenation of 3 rows

\[
>> \text{row1}=[8,1,6]; \text{row2}=[3,5,7]; \text{row3}=[4,9,2]; \ A=[\text{row1};\text{row2};\text{row3}]
\]

\[
A =
\begin{bmatrix}
8 & 1 & 6 \\
3 & 5 & 7 \\
4 & 9 & 2
\end{bmatrix}
\]

or 3 columns

\[
>> \text{col1}=[8;3;4]; \text{col2}=[1;5;9]; \text{col3}=[6;7;2]; \ A=[\text{col1},\text{col2},\text{col3}]
\]

\[
A =
\begin{bmatrix}
8 & 1 & 6 \\
3 & 5 & 7 \\
4 & 9 & 2
\end{bmatrix}
\]

Note the use of , and ;. Concatenated rows/columns must have the same length. Larger matrices can be created from smaller ones in the same way:
The matrix $C$ has dimension $3 \times 6$ (“3 by 6”). On the other hand smaller matrices (submatrices) can be extracted from any given matrix:

$$
\begin{align*}
\text{sin} & \quad 2,3) \quad \% \text{ coefficient of } A \text{ in 2nd row, 3rd column} \\
\text{ans} & = 7 \\
\text{sin} & \quad 1, :) \quad \% \text{ 1st row of } A \\
\text{ans} & = 8 \quad 1 \quad 6 \\
\text{sin} & \quad (:, 3) \quad \% \text{ 3rd column of } A \\
\text{ans} & = 6 \\
& \quad 7 \\
& \quad 2 \\
\text{sin} & \quad ([1, 3],[2, 3]) \quad \% \text{ keep coefficients in rows 1 & 3 and columns 2 & 3} \\
\text{ans} & = 1 \quad 6 \\
& \quad 9 \quad 2
\end{align*}
$$

★ Some matrices are already predefined in MATLAB:

$$
\begin{align*}
\text{sin} & \quad \text{eye}(3) \quad \% \text{ the Identity matrix} \\
I & = 1 \quad 0 \quad 0 \\
& \quad 0 \quad 1 \quad 0 \\
& \quad 0 \quad 0 \quad 1 \\
\text{sin} & \quad \text{magic}(3) \\
\text{ans} & = 8 \quad 1 \quad 6 \\
& \quad 3 \quad 5 \quad 7 \\
& \quad 4 \quad 9 \quad 2
\end{align*}
$$

(what is magic about this matrix?)

★ Matrices can be manipulated very easily in MATLAB (unlike MAPLE). Here are sample commands to exercise with:

$$
\begin{align*}
\text{sin} & \quad \text{magic}(3); \\
\text{sin} & \quad B=A' \quad \% \text{ transpose of } A, \text{i.e, rows of } B \text{ are columns of } A \\
B & = 8 \quad 3 \quad 4 \\
& \quad 1 \quad 5 \quad 9 \\
& \quad 6 \quad 7 \quad 2 \\
\text{sin} & \quad A+B \quad \% \text{ sum of } A \text{ and } B \\
\text{ans} & = 16 \quad 4 \quad 10 \\
& \quad 4 \quad 10 \quad 16 \\
& \quad 10 \quad 16 \quad 4 \\
\text{sin} & \quad A*B \quad \% \text{ standard linear algebra matrix multiplication} \\
\text{ans} & = 101 \quad 71 \quad 53
\end{align*}
$$
MATLAB sessions: Laboratory 2

71 83 71
53 71 101
>> A.*B % coefficient-wise multiplication
ans =
  64  3  24
  3  25  63
 24  63  4

★ One MATLAB command is especially relevant when studying the solution of linear systems of differentials equations: \(x = A\backslash b\) determines the solution \(x = A^{-1}b\) of the linear system \(Ax = b\). Here is an example:

\[
\begin{align*}
A &= \text{magic}(3); \\
z &= [1,2,3]' \quad \text{same as } z = [1;2;3] \\
\text{z =} \\
&1 \\
&2 \\
&3 \\
b &= A*z \\
b = \\
&28 \\
&34 \\
&28 \\
x &= A\backslash b \quad \text{solve the system } Ax = b. \text{ Compare with the exact solution, } z, \text{ defined above.} \\
x = \\
&1 \\
&2 \\
&3 \\
y &= \text{inv(A)}*b \quad \text{solve the system using the inverse: less efficient and accurate} \\
\text{ans =} \\
&1.0000 \\
&2.0000 \\
&3.0000 \\
\end{align*}
\]

Now let’s check for accuracy by evaluating the difference \(z - x\) and \(z - y\). In exact arithmetic they should both be zero since \(x\), \(y\) and \(z\) all represent the solution to the system.

\[
\begin{align*}
\text{z - x} \quad \text{error for backslash command} \\
\text{ans =} \\
&0 \\
&0 \\
&0 \\
\text{z - y} \quad \text{error for inverse} \\
\text{ans =} \\
&1.0e-015 * \\
&-0.4441 \\
&0 \\
&-0.8882 \\
\end{align*}
\]

Note the multiplicative factor \(10^{-15}\) in the last computation. MATLAB performs all operations using standard IEEE double precision.

**Important!**: Because of the finite precision of computer arithmetic and roundoff error, vectors or matrices that are zero (theoretically) may appear in MATLAB in exponential form such as \(1.0e-15 M\) where \(M\) is a vector or matrix with entries between \(-1\) and \(1\). This means that each component of the
EXERCISE 1
Enter the following matrices and vectors in MATLAB
\[ A = \begin{bmatrix} 5 & -1 & 3 \\ 2 & 4 & -7 \\ 6 & 1 & 8 \end{bmatrix}, \quad B = \begin{bmatrix} 12 & 0 & 7 \\ 3 & -2 & 5 \\ -1 & 9 & 10 \end{bmatrix}, \quad b = \begin{bmatrix} 16 \\ 36 \\ 17 \end{bmatrix}, \quad c = [1 \ 2 \ 3], \quad d = \begin{bmatrix} 4 \\ 3 \end{bmatrix} \]

(a) Perform the following operations: \( AB, BA, cA \) and \( Bd \) (use standard linear algebra multiplication).
(b) Construct a \( 6 \times 3 \) matrix \( C = \begin{bmatrix} A \\ B \end{bmatrix} \) and a \( 3 \times 4 \) matrix \( D = [B \ d] \).
(c) Use the “backslash” command to solve the system \( Ax = b \).
(d) Replace \( A(3,2) \) with 0.
(e) Extract the 2nd row of the matrix \( A \) and store it in the vector \( a \).
(f) A row or a column of a matrix can be deleted by assigning the empty vector \([\ ]\) to the row or the column. For instance \( A(2,:) = [] \) deletes the second row of the matrix \( A \).
Delete the third column of the matrix \( B \).

MATLAB Programming
It is often advantageous to be able to execute a segment of a code a number of times. A segment of a code that is executed repeatedly is called a loop.
To understand how loops work, it is important to recognize the difference between an algebraic equality and a MATLAB assignment. Consider the following commands:

```matlab
>> counter = 2
counter =
    2
>> counter = counter +1
```
The last statement does not say that \( counter \) is one more than itself. When MATLAB encounters the second statement, it looks up the present value of \( counter \) (2), evaluates the expression \( counter + 1 \) (3), and stores the result of the computation in the variable on the left, here \( counter \). The effect of the statement is to increment the variable \( counter \) by 1, from 3 to 4.

Similarly, consider the commands:

```matlab
>> v=[1,2,3]
v =
    1    2    3
>> v=[v,4]
v =
    1    2    3    4
```
When MATLAB encounters the second statement, it looks up the present value of \( v \), adds the number 4 as entry of the vector, and stores the result in the variable on the left, here \( v \). The effect of the statement is to augment the vector \( v \) with the entry 4.

There are two types of loops in MATLAB: for loops and while loops.
for loops

When we know exactly how many times to execute the loop, the for loop is often a good implementation choice. One form of the command is as follows:

```matlab
for k=kmin:kmax
    <list of commands>
end
```

The loop index or loop variable is `k`, and `k` takes on integer values from the loop’s initial value, `kmin`, through its terminal value, `kmax`. For each value of `k`, MATLAB executes the body of the loop, which is the list of commands.

Here are a few examples:

- Determine the sum of the squares of integers from 1 to 10: $1^2 + 2^2 + 3^2 + \ldots + 10^2$.

  ```matlab
  S = 0; % initialize running sum
  for k = 1:10
      S = S+k^2;
  end
  S
  ```

  Because we are not printing intermediate values of `S`, we display the final value of `S` after the loop by typing `S` on a line by itself. Try removing the `;` inside the loop to see how `S` is incremented every time we go through the loop.

- Determine the product of the integers from 1 to 10: $1 \cdot 2 \cdot 3 \ldots \cdot 10$.

  ```matlab
  p = 1; % initialize running product
  for k = 2:10
      p = p*k;
  end
  p
  ```

  ★ Whenever possible all these construct should be avoided and built in MATLAB functions used instead to improve efficiency. In particular lengthy loops introduce a substantial overhead.

The value of `S` in the example above can be evaluated with a single MATLAB statement:

```matlab
>> S = sum((1:10).^2)
```

Type `help sum` to see how the built in `sum` function works.

Similarly the product `p` can be evaluated using

```matlab
>> p = prod(1:10)
```

Type `help prod` to see how the built in `prod` function works.

EXERCISE 2

Recall that a geometric sum is a sum of the form $a + ar + ar^2 + ar^3 + \ldots$.

(a) Write a function file that accepts the values of `r`, `a` and `n` as arguments and uses a for loop to return the sum of the first `n` terms of the geometric series. Test your function for $a = 5$, $r = 1/3$ and $n = 8$.

(b) Write a function file that accepts the values of `r`, `a` and `n` as arguments and uses the built in command `sum` to find the sum of the first `n` terms of the geometric series. Test your function for $a = 5$, $r = 1/3$ and $n = 8$.

   **Hint:** Start by defining the vector `e=0:n-1` and then evaluate the vector `R = r.^e`. It should be easy to figure out how to find the sum from there.
EXERCISE 3
The counter in a for or while loop can be given explicit increment: for i = m:k:n to advance the counter i by k each time. In this problem we will evaluate the product of the first 8 odd numbers 1 \cdot 3 \cdot 5 \ldots \cdot 15 in two ways:

(a) Write a script file that evaluates the product of the first 8 odd numbers using a for loop.

(b) Evaluate the product of the first 8 odd numbers using a single MATLAB command. Use the MATLAB command prod.

while loop
The while loop repeats a sequence of commands as long as some condition is met. The basic structure of a while loop is the following:

```
while <condition>
  <list of commands>
end
```

Here are some examples:

• Determine the sum of the inverses of squares of integers from 1 until the inverse of the integer square is less than $10^{-10}$: $\frac{1}{1^2} + \frac{1}{2^2} + \ldots + \frac{1}{k^2}$ while $\frac{1}{k^2} \geq 10^{-10}$.

```matlab
S = 0; % initialize running sum
k = 1; % initialize current integer
incr = 1; % initialize test value
while incr>=1e-10
  S = S+incr;
  k = k+1;
  incr = 1/k^2;
end
```

What is the value of S returned by this script? Compare to $\sum_{k=1}^{\infty} \frac{1}{k^2} = \frac{\pi^2}{6}$.

• Create a row vector y that contains all the factorials below 2000: $y = [1!, 2!, 3!, \ldots, k!]$ while $k! < 2000$.

```matlab
y = []; % initialize the vector y to the empty vector
k = 1; % initialize the counter
value = 1; % initialize the test value to be added to the vector y
while value < 2000
  y = [y, value]; % augment the vector y
  k = k+1; % update the counter
  value = factorial(k); % evaluate the next test value
end
y
```

EXERCISE 4
Write a script file that creates a row vector v containing all the powers of 3 below 3000. The output vector should have the form: $v = [3, 9, 27, 81, \ldots]$. Use a while loop.
if statement

The basic structure of an if statement is the following:

```matlab
if condition
    <list of commands>
elseif condition
    <list of commands>
else
    <list of commands>
end
```

Here is an example:

- Evaluate

\[
y = \begin{cases} 
  x^3 + 2, & x \leq 1 \\
  \frac{1}{x - 2}, & x > 1 
\end{cases}
\]

for a given (but unknown) scalar \( x \) and, if \( x = 2 \), display “\( y \) is undefined at \( x = 2 \)”.

```matlab
function y=f(x)
    if x==2
        disp('y is undefined at x = 2')
    elseif x <= 1
        y=x^3+2;
    else
        y=1/(x-2);
    end
end
```

We can test the file by evaluating it at different values of \( x \). Below we evaluate the function at \( x = -1, x = 2 \) and \( x = 4 \).

```matlab
>> f(-1)
an = 
1
>> f(2)
y is undefined at x = 2
>> f(4)
an = 
0.5000
```

EXERCISE 5

Write a function file that creates the following piecewise function:

\[
f(x) = \begin{cases} 
  e^{x-1}, & x \leq 2 \\
  x^2 + x, & 2 < x \leq 4 \\
  \frac{x}{x - 7}, & x > 4 
\end{cases}
\]

Assume \( x \) is a scalar. The function file should contain an if statement to distinguish between the different cases. The function should also display “the function is undefined at \( x = 7 \)” if the input is \( x = 7 \). Test your function by evaluating \( f(1), f(2), f(3), f(4), f(7) \) and \( f(10) \).