# DAILY ASSESSMENT FORMAT

| | | | |
|---|---|---|---|
| **Date:** | **06-07-2020** | **Name:** | **M V Ramya** |
| **Course:** | **Mat lab Onramp** | **USN:** | **4AL17EC045** |
| **Topic:** | **1. Course Overview**<br>**2. Commands**<br>**3. MATLAB Desktop and Editor**<br>**4. Vectors and Matrices** | **Semester & Section:** | **6th & A** |
| **GitHub Repository:** | **MV-Ramya-045** | | |

**FORENOON SESSION DETAILS**

MAT Lab:

MATLAB is a multi-paradigm numerical computing environment and proprietary programming language developed by MathWorks. MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages.

Although MATLAB is intended primarily for numerical computing, an optional toolbox uses the MuPAD symbolic engine allowing access to symbolic computing abilities. An additional package, Simulink, adds graphical multi-domain simulation and model-based design for dynamic and embedded systems.

As of 2020, MATLAB has more than 4 million users worldwide. MATLAB users come from various backgrounds of engineering, science, and economics.

History

Cleve Moler, the chairman of the computer science department at the University of New Mexico, started developing MATLAB in the late 1970s. He designed it to give his student’s access to LINPACK and EISPACK without them having to learn Fortran. It soon spread to other universities and found a strong audience within the applied mathematics community. Jack Little, an engineer, was exposed to it during a visit Moler made to Stanford University in 1983. Recognizing its commercial potential, he joined with Moler and Steve Bangert. They rewrote MATLAB in C and founded Math Works in 1984 to continue its development. These rewritten libraries were known as JACKPAC. In 2000, MATLAB was rewritten to use a newer set of libraries for matrix manipulation, LAPACK.

MATLAB was first adopted by researchers and practitioners in control engineering, Little's specialty, but quickly spread to many other domains. It is now also used in education, in particular the teaching of linear algebra and numerical analysis, and is popular amongst scientists involved in image processing.

## Syntax

The MATLAB application is built around the MATLAB programming language. Common usage of the MATLAB application involves using the "Command Window" as an interactive mathematical shell or executing text files containing MATLAB code.

### Variables

Variables are defined using the assignment operator, =. MATLAB is a weakly typed programming language because types are implicitly converted. It is an inferred typed language because variables can be assigned without declaring their type, except if they are to be treated as symbolic objects, and that their type can change. Values can come from constants, from computation involving values of other variables, or from the output of a function. For example:

```
>> x = 17
x =
 17

>> x = 'hat'
x =
hat

>> x = [3*4, pi/2]
x =
   12.0000    1.5708

>> y = 3*sin(x)
y =
   -1.6097    3.0000
```

### Vectors and matrices

A simple array is defined using the colon syntax: *initial*:*increment*:*terminator*. For instance:

```
>> array = 1:2:9
array =
 1 3 5 7 9
```

Defines a variable named array (or assigns a new value to an existing variable with the name array) which is an array consisting of the values 1, 3, 5, 7, and 9. That is, the array starts at 1 (the *initial* value), increments with each step from the previous value by 2 (the *increment* value), and stops once it reaches (or to avoid exceeding) 9 (the *terminator* value).

```
>> array = 1:3:9
array =
 1 4 7
```

the *increment* value can actually be left out of this syntax (along with one of the colons), to use a default value of 1.

```
>> ari = 1:5
ari =
 1 2 3 4 5
```

assigns to the variable named ari an array with the values 1, 2, 3, 4, and 5, since the default value of 1 is used as the increment.

Indexing is one-based, which is the usual convention for matrices in mathematics, unlike zero-based indexing commonly used in other programming languages such as C, C++, and Java.

Matrices can be defined by separating the elements of a row with blank space or comma and using a semicolon to terminate each row. The list of elements should be surrounded by square brackets []. Parentheses () are used to access elements and subarrays (they are also used to denote a function argument list).

```
>> A = [16 3 2 13; 5 10 11 8; 9 6 7 12; 4 15 14 1]
A =
 16  3  2 13
  5 10 11  8
  9  6  7 12
  4 15 14  1

>> A(2,3)
ans =
 11
```

Sets of indices can be specified by expressions such as 2:4, which evaluates to [2, 3, 4]. For example, a submatrix taken from rows 2 through 4 and columns 3 through 4 can be written as:

```
>> A(2:4,3:4)
ans =
 11 8
 7 12
 14 1
```

A square identity matrix of size *n* can be generated using the function eye, and matrices of any size with zeros or ones can be generated with the functions zeros and ones, respectively.

```
>> eye(3,3)
ans =
 1 0 0
 0 1 0
 0 0 1

>> zeros(2,3)
ans =
 0 0 0
 0 0 0

>> ones(2,3)
ans =
 1 1 1
 1 1 1
```

Transposing a vector or a matrix is done either by the function transpose or by adding dot-prime after the matrix (without the dot, prime will perform conjugate transpose for complex arrays):

```
>> A = [1 ; 2],  B = A.', C = transpose(A)
A =
    1
    2
B =
    1     2
C =
    1     2

>> D = [0 3 ; 1 5], D.'
D =
    0     3
    1     5
ans =
    0     1
    3     5
```

Most functions accept arrays as input and operate element-wise on each element. For example, mod(2*J,n) will multiply every element in *J* by 2, and then reduce each element modulo *n*. MATLAB does include standard for and while loops, but (as in other similar applications such as R), using the vectorized notation is encouraged and is often faster to execute. The following code, excerpted from the function *magic.m*, creates a magic square *M* for odd values of *n* (MATLAB function meshgrid is used here to generate square matrices *I* and *J* containing *1:n*).

```
[J,I] = meshgrid(1:n);
A = mod(I + J - (n + 3) / 2, n);
B = mod(I + 2 * J - 2, n);
```

```
M = n * A + B + 1;
```

Structures

MATLAB supports structure data types. Since all variables in MATLAB are arrays, a more adequate name is "structure array", where each element of the array has the same field names. In addition, MATLAB supports dynamic field names (field look-ups by name, field manipulations, etc.).

Functions

When creating a MATLAB function, the name of the file should match the name of the first function in the file. Valid function names begin with an alphabetic character, and can contain letters, numbers, or underscores. Variables and functions are case sensitive.

Function handles

MATLAB supports elements of lambda calculus by introducing function handles, or function references, which are implemented either in .m files or anonymous/nested functions.

Classes and object-oriented programming

MATLAB supports object-oriented programming including classes, inheritance, virtual dispatch, packages, pass-by-value semantics, and pass-by-reference semantics. However, the syntax and calling conventions are significantly different from other languages. MATLAB has value classes and reference classes, depending on whether the class has *handle* as a super-class (for reference classes) or not (for value classes).

Method call behavior is different between value and reference classes. For example, a call to a method

```
object.method();
```

can alter any member of *object* only if *object* is an instance of a reference class, otherwise value class methods must return a new instance if it needs to modify the object.

An example of a simple class is provided below.

```
classdef Hello
   methods
      function greet(obj)
         disp('Hello!')
      end
   end
end
```

When put into a file named hello.m, this can be executed with the following commands:

```
>> x = Hello();
>> x.greet();
```

| Hello! |
|---|
| |