CSC 372 report1

Yifan Li (yifanli@Email.arizona.edu), Alex Melnychuck(amelnychuck@email.arizona.edu)

April 21, 2020

1 Abstract

2 Introduction

3 Why Matlab?

4 History

Matlab is a language invited by Cleve Moler, at first it was not a programming language, it was just a interactive matrix calculator. At 1970S, the author, Cleve Moler wants his student can use 'LINPACK' and 'EISPACK' to write program more conveniently, instead of using Fortran. Then he uses Fortran to write the first version of Matlab. And with the development of Matlab until today, it becomes a very important and popular language to scientist or scholars,

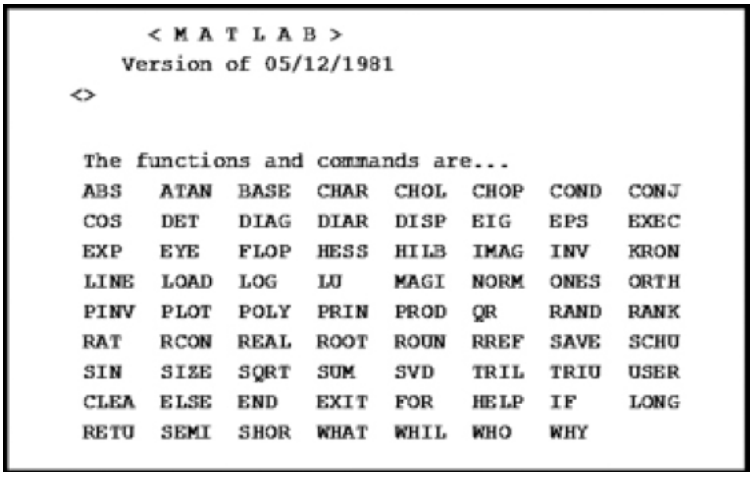

Figure 1: Early Version of Matlab

5 Control Structures

Matlab contains many typical control structures that are found in other programming languages such as C or Java. For a studied programmer first learning to program with Matlab, many control structures will be familiar in concept, even if differences in syntax can take getting used to. To someone who is new to programming, the control structures in Matlab may be viewed as simple yet powerful—allowing one to demonstrate a variety of functionality with a relatively small set of structures.

5.1 If, Else, and Elseif

A concept that is found in most programming languages, if, else, and elseif control statements carry over into Matlab much like they function in traditionally taught languages.

Observe the following code:

```
grade = 90;
if grade >= 90
    disp('You re a good student!');
else
    disp('Try harder!');
end
```

The if statement will execute the block of code when the condition specified after the if keyword is true. In this case, the message "You're a good student!" will be displayed in the console if the grade variable is set to 90 or greater.

An else statement will execute when the initial if condition is not met. In the example above, a grade that is less than 90 will result in the message "Try harder!" displaying in the console.

The elseif statement can be used to add additional cases to basic if else statements.

Using both elseif and else blocks are not required—that is to say that an if statement can be used on it's own. Conditions are checked from top to bottom, and the first condition that evaluates to true will execute it's corresponding statement.

5.2 Switch

For control structures that involve many different cases, or conditions as they were referred to above, a Switch structure may be more appropriate.

Observe the following code:

```
k = 520;
switch k
case 520
area = "Tucson";
case 480
```

```
area = "Mesa"
end
disp(area);
```

The switch keyword takes the value of the variable that follows it—in this case k—and runs a comparison check against each of the subsequent case conditions below. Like the if else block, the first case condition that the variable matches will be the case that executes it's following statement. In the example above, the variable's value is 520 and so the message "Tucson" will be displayed to the user after the switch structure has been run.

5.3 For Loops

For Loops are a repetition control structure that can be used to iterate over a collection or sequence in a fashion similar to other programming languages. Although the syntax may vary from C or Java, the basic concept is the same.

Observe the following code:

```
for i = 1:5:25
    disp("I Love Csc372!");
end
```

In this example, the message "I Love Csc372!" is printed to the console a total of 5 times. The first number is the starting point for i, and the third number is the stopping point for i. The middle value represents the value by which i should be incremented (or decremented in the case of negative values.)

5.4 While Loops

Similar in nature to the for loop, a while loop can be used to repeat a block of code until a particular condition is no longer met. While loops can be commonly used in recursive functions and in iterative use cases.

Observe the following code:

```
year = 2000;
while year < 2020
    disp("It's not 2020! It's" + year)
    year = year + 1;
end
disp("Wow it's 2020")
```

In the example above, the year variable is initially set to 2000. When the program execution reaches the while block, it checks that the value is less than 2020. Since it is in it's first pass through the block, it continues to display, "It's not 2020! It's 2000". It will continue to increment it's year, and repeat the check to see if the year is still less than 2020. Once the year is actually 2020, it will exit the while block and continue to display "Wow it's 2020.")

5.5 Try Catch

Intro paragraph/sentence

Observe the following code:

```
m = rand(3,4);
n = magic(5);
try
      a = m*n;
      disp(a)
catch
      disp(size(m))
      disp(size(n))
end
disp("Since they have different size so can't do matrix
    multiplication")
```

In the example code above, it defines two matrix in different sizes, and try to multiple them, we know that the matrix multiplication is only legal when the size of matrix fit each others, so since m and n's sizes are not fit with each others, the content in try will not run, produces an exception and then be caught. The content in the catch will run. which display the size of matrices.

6 Data Types

Matlab contains a number of different data types, which may be useful in achieving a wide variety of tasks within the language. These data types are similar to those found in other programming languages, and we will explore them in this section.

6.1 Char

A char in Matlab can be used to store strings, similarly to what can be performed in other languages.

The following code points out some key functionality available in Matlab:

```
a = 'A';
b = ischar(a);
c = char('hellow world');
c = c + c;%char to acsii
num = [77 65 84 76 65 66]; %acsii for 'matlab'
word = char(num); % 'matlab'
```

Chars can be initialized with a single 'A' letter, while also supporting more complex setups, such as instantiation through an array of ascii integers. Like in languages such as C, a char is ultimately backed by a numeric array of numbers that correspond to an encoding scheme. Character arrays can be used to store

strings of text as what are called character vectors, as is shown in line 3 of the example code above.

6.2 Int / Double

As in other languages, Matlab supports a variety of integer classes to be useful across a wide variety of use cases and number sizes. These include the following classes: int8, int16, int32, int64, uint8, uint16, uint32, unint64. With classes for both signed and unsigned integers, it is possible to work with both negative and positive integers. Casting across various types of int, and other numeric classes in Matlab, is fully supported and shown in the example below.

The following code points out some key functionality available in Matlab:

```
1 a = 1.0; %a double
2 a = int32(a); %transfer to int
3 b = a + a;
4 c = a * a;
5 d = (2*a)^(a+a);
```

Interestingly enough, Matlab will store all numbers with floating point precision as a standard. This is an area in which Matlab deviates from many language standard practices—which makes sense considering the heavy emphasis on mathematical modeling and numeric computation that is placed on the language's functionality. This means that if you explicitly want to work with an integer with no decimal precision, it must be converted to an int type explicitly, as shown above.

Additional number-oriented operations on double numeric types can be found below:

```
1 a = pi;
2 b = floor(a); %b = 3, round a to lower.
3 c = fix(a);
4 d = ceil(a); %round to upper
5 e = round(a); %round a.
6 f = a*a; %pi^2
7 g = a/2; %pi/2
```

Detailed Explanation

6.3 Boolean

Boolean values can be used to represent a state being either true or false. Booleans are stored as 1, and 0, indicating true and false respectively.

Below are a handful of useful boolean operations performed in matlab:

```
1 a = true;
2 b = not(a); %1->0
3 if b == false
4     disp('b is false!');
```

```
end
```

Booleans can be initialized or set using the keywords "true" and false". A boolean can be toggled using the not(var) function, and booleans can be used as the basis of logical or conditional operations, such as checking whether or not a particular value is true before executing a block of code or leaving a loop.

6.4 Array / Matrix

Arrays and Matrices allow for the one dimensional, or two dimensional representation of data in Matlab. An array can be created from a new representation and initialization, or can be concatenated with other arrays, or copied similarly. Array functionality and scope is fairly similar to what is present in other languages, although some more math oriented functionality is included for developer convenience.

The following code points out some key functionality available in Matlab:

```
a  = [1  2  3];
b = a'
c = a*2;
d = b * c;
e = [1;2;3];
```

Matlab's focus on mathematical computation leads it as a language to include helper fuctions for common tasks with arrays and matrices. For example, calling the function zeros(n) will create an n b n matrix of 0 values.

Indexing values can be performed using helper functions such as end, as well as with colon based indexing which can be seen exemplified above.

7 Subprograms

7.1 Functions, Subroutines, and Classes

Functions, subroutines, are all language features that allow a programmer to break up and organize code. Functions allow for a "subprogram", which may be anything from a helper or utility function to a feature component, to run within the program as a whole. For example, one may break different parts of a console based I/O game up into functions, such that each turn type is performed via a different function.

A function can be passed various parameters, and can have a return value, placed in —pipes—, which can be seen within the example code below:

```
% this function can calculate the area of a triangle with
    input length and width.
% any script in the same folder can call it, no need to
   import.
% note it should be called with
% the file name, which is the
```

```
% p4_sub_program.

function [area] = get_area_tri(length,width)
area = length * width/4;
```

Functions can be used to abstract out functionality, or aid in making code reusable across multiple sections of a project. Functions can also avoid the duplication of code in multiple places, which can make code written in a language harder to maintain.

Subroutines, just like functions, can be used to break down programs into smaller components. Subroutines can also be imported via a script file, which can be run alongside other functions that are defined within the program. This is a great feature which aligns with Matlab's intended mathematics focus.

Additional examples of functions and subroutines can be seen below:

```
%this script define a function get_area_cir which get the
    area of a circile
%with input radius. When we define a varibale which is an
    int, call the
%function with it, it will give the area of a circle with
    the radius of the
%variable.

r = 2
area_A = get_area_cir(2)
%area_B = p4_sub_program(2,4)

function [area] = get_area_cir(radius)
area = radius*radius * pi;
end
```

The Matlab language also has a concept of classes, which are a common feature among object oriented programming languages. Classes can be used to organize code according to an architecture pattern. Classes can be initialized and are mutable over the lifecycle of the instance. Classes can contain functions which can act upon instance variables contained within the class. In the Person class we will explore shortly, one can see that name, age, height, and weight, all have getter methods which provide access to the variables contained in the object's instance.

An example Person class is provided below:

```
% this class defines a person, which have the properties
    of
% name, age, height, and weight. It contains function to
    set value of each
% properties and return the value of each properties.

classdef P4_class_Person
```

```
    properties
        name, age, height, weight;
    end
    methods
        function set_person(name, age, height, weight)
            obj.name = name;
            obj.age = age;
            obj.height = height;
            obj.weight = weight;
        end

        function name = get_name(obj)
            name = obj.name;
        end

        function age = get_age(obj)
            age = obj.age;
        end

        function height = get_height(obj)
            height = obj.height;
        end

        function weight = get_weight(obj)
            weight = obj.weight;
        end

    end
end
```

8 Summary

9 References

10 Fun Fact

These are example citations such as [1] and [2] and [3] and [4] so that they show up in the References section below.

References

[1] "Matlab," Apr 2020. [Online]. Available: https://en.wikipedia.org/wiki/MATLAB

[2] “Matlab.” [Online]. Available: https://www.mathworks.com/help/matlab/

[3] [Online]. Available: https://www.mathworks.com/help/releases/R2014b/pdf_doc/matlab/getstart.pdf

[4] “Newest ’matlab’ questions.” [Online]. Available: https://stackoverflow.com/questions/tagged/matlab