CS372 MATLAB Project

By: Evan Lee and Sebastian Tu

# Abstract

# 

# 

# Introduction

# 

# History

MATLAB’s original purpose was to be a strong matrix calculator, and nothing more. Algorithms for solving matrix linear equations and eigenvalue programs were written in a research paper by J. H. Wilkinson and eighteen of his colleagues. MATLAB was born out of Fortran, with matrices as the only data type. In 1983, MATLAB creator Cleve Moler’s associate Jack Little used a Compaq PC clone to write a more extended version of MATLAB. Functions, toolboxes, graphics, and many other additions were made to MATLAB with this new version. What resulted was named PC-MATLAB, and was revealed at the IEEE Conference on Decision and Control in December of 1984. Since then, MATLAB has continued to develop uses and capabilities for technical computing. Computation, visualization, and programming are its key components. Today, it still has a presence as a private company which specializes in mathematical computation. Matlab today is still used due to its graphical displays and “toolboxes”, which greatly expand the language to interact with other languages and programs such as Microsoft Excel and R.

# Control Structures

While MATLAB has plenty of control structures, it does not contain any special or unique control structures as compared to other languages such as Ruby’s Until loop (which is a while not loop). MATLAB’s control structures can be split into two categories: branching statements and loop statements.

3.1 Branch Statements

Branch statements compare logical conditions to decide which program block should be executed. MATLAB has two types of branch statements: if conditions and switch statements.

Branch statements in MATLAB consist of if statements and switch cases. An if statement in MATLAB is written as i f (1 == 1) display (" t h i s will be shown " ) ; else

display (" t h i s will not be shown " ) ;

In the example above, the code uses an ELSE clause, which handles cases that evaluate to false in the if statement. If statements can be used together to create nested-if statements, which allow for more complex conditions. An example of this is :

i f (1 == 1) i f (2 == 2) display (" This will pass through two conditions " ) ; else

display (" This else i s part of the inner i f statement ) ; else

display (" This else i s part of the outer i f statement " ) ;

Switch statements are similar to if statements,though they allow for multiple branches, instead of a single branch in the case of an if statement. They have a variable which is used as the "switch", and will execute a specific code block based on the value of the switch. An example of a switch statement is: t e s t S c o r e = ’A’ switch ( t e s t S c o r e ) case ’A’ display (" Good Job ! " ) ;

case ’B’ display (" Nice ! " ) ;

case ’C’ display ("C’ s get degrees ! " ) ;

case ’D’ display (" Close ! " ) ;

In this case, the code block that will be executed is case ’A’, which will output the string "Good Job!" to the standard output.

3.2 Loop Statements

Loop statements are used to run a block of code a certain number of times. MATLAB has two types of loops statements: while loops and for loops.

While loops run a block of code continuously until their condition is broken. They effectively are repeatedly calling an if statement and running the code inside as long the condition evaluates to true. The MATLAB implementation of this is:

count = 0; while ( count < 5){ display (" This portion will be printed out 5 times ) ;

count = count + 1;

}

In the example above, the while loop checks a variable’s value (count). The code block repeats as many times as the the condition holds true, so in this case, 5 times.

For loops are similar to while loops, with more information built in. For loops have 3 portions: a block that will only be executed once at the beginning of the loop, a conditional block for the loop, and a block that is executed after every pass through the loop. MATLAB implements for loops as such: for ( i =0; i < 5; i = i + 1){ display (" This will be printed out 5 times " ) ;

}

The example above uses the first portion of the loop to create a variable i, the second portion to check the value of i, and the third portion to increment i by one. This effectively does the same as the while loop example above.

# Data Types

4.1 Integer, Unsigned Integer, Floating Point.

Integers, floating point, and unsigned integers are stored by default as double-precision floating point numbers. They can be specified as single-precision numbers, which offer much more memory-efficient storage than double-precision numbers. You can expose and output properties and traits of the type by using the command whos. Integers will be represented by the class double.

X = 10;

whos x;

Name Size Bytes Class Attributes

x 1x1 8 double

4.2 Character Array and String Array

In MatLab, strings are representations of character arrays. Just like any generic array, you can make arrays of strings, which will be discussed alongside all other data structures further on. Using the String function, you can convert an arbitrary data type into a string array using the string function.

str = "Hello, world"

str = ["Mercury","Gemini","Apollo"; "Skylab","Skylab B","ISS"]

str = 2x3 string array

"Mercury" "Gemini" "Apollo"

"Skylab" "Skylab B" "ISS"

str = ["hello", "world"];

x = join (str, " ");

disp(x);

“hello world”

4.3 Numeric Array

Any numeric value can be stored as single or double precision arrays. There are functions (ex: int8) for specifying byte sizes: including 8, 16, 32, 64; along with equivalent functions for unsigned integers (ex: uint8). There are also functions for the testing of numerous array contents, including: integer, float, numeric, real, finite, infinite, and NaN.

w = int8(10);

x = int8(11);

disp(isinteger(w));

1

disp(isfloat(x));

0

Seeing as MatLab stands for ”Matrix Laboratory”, all arrays are two dimensional arrays, or matrices. To create these, you separate horizontal elements by whitespace and rows by semicolons.

a = [1 2 3 4]

a = 1×4

1 2 3 4

# Subprograms

# 

# Summary

# 

# References