MATLAB: A Glimpse into the Future of Humanity?

Matthew Gallagher, Ryan Hessler, Brandon Mitchell

17 September 2021

CSCI 305: Concepts of Programming Languages

Fall 2021

Introduction

First appearing in the late 1970s, MATLAB is a programming language and numeric computing environment developed by MathWorks. MATLAB stands for Matrix Laboratory, which is the concept for which the program was originally developed. MATLAB is intended to be used for mathematical computing and was originally geared towards linear algebra problems. It can also work with data plotting, algorithms, UIs, and can interface with other languages allowing for a broad range of useful applications. Some modern packages for MATLAB expand the functionalities to allow model simulations and the ability to interface with embedded systems. Considering the power and versatility of MATLAB, it's no wonder there are now over four million users of the language.

History

Originally, MATLAB was simply a matrix calculator written in FORTRAN. Its inventor was Cleve Moler, and he had designed it for use by his students to aid in linear algebra calculations. When it was first distributed in 1979, it was given out to universities for free and only had seventy-one functions. However, in the early 1980s, MATLAB was re-written in C to allow it to be used on the IBM desktops that were replacing mainframe computers. When it was rewritten, it still retained much of the original FORTRAN syntax. In addition, this marked the creation of the MATLAB programming language. Eventually, it would be released commercially in 1984 and would see success partially due to its vast selection of specialized toolboxes, which could be considered to be similar to libraries in other languages. Nowadays, MATLAB can be considered to be a multi-paradigm, weakly-typed, interpreted, and imperative programming language.

Intended Uses

MATLAB was originally only an interactive matrix calculator, as seen from its full name, Matrix Laboratory. The software quickly became popular among university math departments as a quick way to perform complex mathematics. Over time, the software was not only ported to other operating systems, but it also acquired more features. Rather than being a simple matrix calculator, it could perform all basic math operations and was able to be used by GPUs for rendering graphical models.

Why MATLAB?

To add to its usefulness in working with matrices, present day MATLAB can also interface with embedded systems and create standalone applications, allowing for many more broad uses. Its Simulink toolbox is incredibly helpful in modeling and can generate code based on a model. Additionally, it has parallel computing capabilities which allows multiple processes to be done at

once. MATLAB also has the ability to call functions in other languages like C and FORTRAN, and can even convert several languages into equivalent MATLAB code.

Drawbacks

One of the main criticisms of MATLAB is that it can be slow. This can partially be due to it being an interpreted language, but the software itself is performance heavy and can use a lot of memory. Additionally, first time users may struggle at first due to MATLAB's complicated UI, and experienced programmers may still flounder at first due to the unique syntax. Although MATLAB is great for mathematics and can be used for solving problems in a number of other fields, mathematics is still its chief focus. It can be used for embedded systems, web apps, and desktop apps, but it will likely not be the best choice or only choice for those situations. The final point that may make MATLAB difficult to justify using is the price. A single license for just the base program, for example, can range from $860 for the annual license to $2,150 for a perpetual license. However, special prices are available for education institutions and students, so students will have to pay at most $99 for MATLAB and its most useful toolboxes. Considering freeware like GNU Octave and Julia have expanded to fill the market as well, many prospective users may feel there is not necessarily a need to pay for MATLAB when there are other viable alternatives that accomplish the same purpose.

Syntax and Unique Features

Assignment and Miscellaneous

Variables are assigned similarly to other languages with the `=` operator and must start with an underscore or letter. Numeral characters can appear as the second or subsequent characters. Multiple expressions can be executed in one line by separating the expressions with a semicolon or a comma (e.g., `a = 3; b = 2, c = 4`). If an expression returns a value, then that value is outputted to the command window when the expression is terminated by a comma or newline. Terminating an expression with a semicolon suppresses output. The `who` command can be used to see a list of all assigned variables in the current workspace, but to see the variables' values and types, the `whos` command is needed. This information can also be viewed any time in the workspace panel in the MATLAB or GNU Octave application. One can think of the workspace as being similar to the concept of scope in C.

Operators

MATLAB has all basic mathematical operators, plus a few more specific to working with matrices. These are listed below in Table 1. These operators perform differently depending on the size of the matrices given. For example, when the addition operator is used on two matrices of like size, the first element of each matrix will be summed together, then the second, then the third, and so on. They will return a matrix of the same dimensions but with the individual values

from each matrix summed together. Likewise, subtraction, element-wise division, element-wise left division, element-wise multiplication, and element-wise exponation perform this way. However, if one tries to add a matrix and scalar, a one by one matrix, the scalar value will be applied to each element in the matrix. All other operators perform similarly in this situation. If the matrices are of differing size and neither is a scalar, then they must have the same length (longest dimension) or else an error will be thrown. The non-element-wise operators have different requirements to use and may not return a matrix of the same dimension. For example, matrix multiplication (non-element-wise) takes an `n` by `m` and an `m` by `n` matrix and returns a scalar instead. Most functions in MATLAB are vectorized and will perform element-wise on any matrix passed in.

`*`	Scalar or matrix multiplication
`.*`	Element-wise multiplication
`/`	Scalar or matrix division
`./`	Element-wise division
`\`	Left division (`A/B` may not equal `B\A`)
`.\`	Element-wise left division (`A./B == B.\A`)
`.^`	Element-wise exponation
`'`	Complex conjugate transpose
`.'`	Transpose operator
`:`	Colon operator

Table 1, Operators, (TutorialsPoint, 2021)

Control Flow and Loops

Like most languages, MATLAB features several control flow statements. First off, there is the `if-elseif-else` chain. `if` takes in a logical value or predicate and parentheses do not need to surround the logical or predicate. `elseif` and `else` perform exactly the same as other languages and are optional, but the `if-elseif-else` must finish with the `end` keyword. Since conditionals are not enclosed by curly braces or indented like in Python, the `end` keyword is used to signify the termination of the conditional. Likewise, MATLAB features `switch` statements. They consist of a value to switch on, `case` values, and an optional `otherwise` statement. The `otherwise` performs similarly to `default` in C switch statements, but there is no fallthrough like with C switch statements. Similar to `if-elseif-else`, an `end` statement is needed to signify termination. Both `if` and `switch` statements can be nested.

`while` statements behave similarly to other languages, but like before, an end statement is needed to mark termination. In contrast, `for` loops have vastly different syntax from other languages. A sort of template for a `for` loop would be `for var = matrix`. `var` is the loop variable and `matrix` is the matrix you will be iterating over. The loop variable may be used within the body and its modification on one iteration will not affect its subsequent value on the next iteration. Furthermore, both `break` and `continue` statements may be used in loops.

Matrices and their Creation

In MATLAB, the matrix is king. As such, everything in MATLAB is a matrix, and a one by one matrix is called a scalar. The simplest way to create a one by `m` matrix (also known as an array, row vector, or just vector) is to surround space or comma delimited values with brackets. If I wanted a vector consisting of the values 1, 23, -9, and 444, then I could easily create it with `arr = [1 23 -9 444]`. However, this method may not be desirable for larger matrices or matrices with numeric data following some sort of pattern. Although a for loop could be used, the ideal MATLAB way is to use the colon operator. An example of how to use the colon operator is *`start`*`:`*`increment`*`:`*`stop`*. If the increment is left out, then a default increment of one is used. As such, `1:1:5` and `1:5` are equivalent and will both return `[1 2 3 4 5]`. The colon operator can also be used to create vectors of decreasing values by giving it a negative increment, and floating point values can be used for any value.

A simple way to create a two dimensional matrix is to use a semicolon as a delimiter for each row. Keep in mind, however, that jagged arrays are not supported by normal matrices and instead a cell array will be needed. An example of a simple two by two matrix would be `[1 2; 3 4]`. Two dimensional matrices can also be created by concatenating already existing matrices together as seen in `x = 1:5; y = [x; x]`. The colon operator can be used in the creation of two dimensional matrices such as `[1:5; 6:10]`. Several functions exist to aid in the creation and concatenation of two dimensional matrices, but MATLAB does just stop at two dimensions, however. Matrices of three, four, five, and higher dimensions are possible.

Matrix Indexing

Indexing into vectors and matrices can be confusing if one is familiar with other languages. Instead of square brackets to index into vectors (e.g., `arr[5]`), MATLAB uses parentheses. In addition to this difference, MATLAB uses one-based indexing. Although this might seem strange, other languages like FORTRAN (which MATLAB shares some syntax with), COBOL, Lua, Julia, and Mathematica also use one-based indexing. Given that matrices can have one, two, three, or more dimensions, MATLAB's syntax has a very elegant solution. In C, if you had a two dimensional matrix, you could get the second row's third item with `arr[1][2]`. As the dimensions increase, so do the assemblage of brackets. The same code in MATLAB would look like `arr(2, 3)`. Unfortunately, the one-based indexing can be confusing for people who are

not used to it. Furthermore, the way indexing is done looks very much like a function call, both in MATLAB and other languages.

An interesting feature of indexing in MATLAB is that only one value is ever needed to index into a matrix of any dimension. If `arr` is a five by five matrix and someone wants the second row's third element, they can use the aforementioned way or just `arr(12)`. For 2D matrices, this value can be calculated by `(b-1)*n+a` where `a` is the row, `b` the column, and `n` the total number of rows. This method may seem confusing, but it is much quicker due to the layout of the matrix in memory. The matrix is laid out in memory as a one dimensional vector, and this method of indexing treats it as such. As a result of the matrix's representation in memory, it is faster to walk through a matrix going down each column instead of across each row.

In addition to being able to index into multidimensional matrices with a single value, MATLAB has another trick up its sleeve for indexing. Known as logical indexing, you can index into a matrix using another matrix of equal size consisting of logical, or boolean, values. This can be quite useful for cleaning data. For example, if I had a vector `arr = [-2 -1 0 1 2]` and I wanted to remove all negative values, I could find those values using the `>` operator. Since this operator returns a logical value, and it is vectorized, I get a vector of equal size consisting of logical values. `arr > 0` would return `[0 0 1 1 1]` with the zeroes being false and the ones being true. From here, you can use this vector for indexing. `arr(arr > 0)` would return `[0 1 2]`, effectively removing the values that failed the predicate. Any function or operator that returns a logical value and is vectorized can be used like this. Logical indexing can be very useful for non-numeric matrices as well, like structs of data.

In order to select multiple values from a matrix in a single statement (also known as a submatrix), MATLAB allows vectors to be used in indexing. For example, if `arr = [-2 -1 0 1 2]` and I wanted the first, fourth, and fifth elements, I could index into it with a vector containing the specific indices desired. `arr([1 4 5])` would give `[-2 1 2]`. A more complicated version could be done with two dimensional matrices. In that case, the first vector given for indexing would be the rows desired, and the second vector will contain the columns desired. If the matrix is five by five and I want the three by three matrix inside, I could index into it with `mat(2:4, 2:4)`. This will return all the values in rows two through four that are also in columns two through four. The keyword `end` can also be used while indexing and will be equal to the size of the specific dimension being indexed into. In our five by five matrix example, if `end` was used (e.g., `mat(2:end, end-1:end)`) it would return five when used for the rows and also five when used for the columns. Since it is equal to the size of the dimension, calculations can be performed on it. Furthermore, if a user wanted all rows or all columns, they could use the vector `1:end` or just `:`. `:` means to take everything in that specific dimension. So, `mat(:, end-2:end)` will return all values that are in the last three columns.

Finally, MATLAB can handle writing to out-of-range indices, but not reading from out-of-range indices. If we still had our `arr = [-2 -1 0 1 2]` vector from above, then an expression like `x = arr(7)` would throw an error as we are wanting the seventh element in a five element vector. On the flipside, an expression like `arr(7) = 4` would not error and instead return `[-2 -1 0 1 2 0 4]`. The element we are writing to will be set, and any elements in between that do not exist will have their values set to zero. In C, writing to an invalid index (and reading, for that matter) is allowed, but should be avoided as you risk overwriting other values in memory, which is how buffer overflow attacks operate.. However, since MATLAB matrices are not raw pointers, you don't risk overwriting memory contents by accident.

Input / Output Procedures

Standard In / Out

Reading from standard input can be done by using the function `input`. Similar to Python's input statement, it can take a prompt to display to the user. This prompt is required and should be a char array. Any input the user gives will be evaluated and then returned. As such the user could give a numeral literal for input or an expression like `pi / 3` or `rand(4)` and even use variables in the workspace in the expression. Since the input is evaluated, a parse error could be triggered by improper user input. In order to read in strings (i.e., not evaluate input), one can add the optional `'s'` argument after the prompt. Writing to standard output can be done with the `disp` and `fprintf` functions. `fprintf` performs similar to C's `printf` function and is generally more powerful than `disp`.

Reading and Writing to Files

Since one of MATLAB's uses for the analysis of data, MATLAB's file I/O is very powerful and is capable of reading in structured and unstructured data. To read from text files, the function `fopen` can be used. `fopen` returns a file identifier and this identifier must be passed to other functions like `fgetl` (file get line), which reads a single line in as a char array, or `textscan` in order to actually read in the data. `textscan` has the ability to parse data out of formatted text. For example, `textscan(fid, '(%f, %f)')` will parse the input and create a cell array in which each row contains two floats. `fopen` can also be used to open or create a file for writing to as long as the correct permission is given, and to write the file, `fprintf` can actually be used. If a file identifier is given as the first argument, `fprintf` will write to that file instead of standard out.

For structured data like Excel spreadsheets or comma delimited files (*.csv), there are several functions that can be used depending on how you want the data to be stored. The function `xlsread` will read in a numeric matrix from the filename passed and will remove any textual

data. However, this function is no longer recommended. Instead, functions like `readmatrix`, `readcell`, and `readtable` can be used, and they return a matrix, a cell array, and a table respectively. In addition, these functions can take several optional values to change how they parse the data or even to tell them to only read in specific ranges. To write to Excel spreadsheets or comma delimited files, the functions `writematrix`, `writecell`, and `writetable` can be used. As one might expect, they take a matrix, a cell array, and a table, respectively, and the type of file they write to depends on the file extension of the file name given.

Importing data of any form can be done using the `importdata` function. This function can be used to read in comma delimited files, textual files, and even image data. The function takes a file name and several optional parameters. Like before, these optional parameters change how it reads in the data and parses it. Since there are other functions for more specific file types, this function should only be used as a last resort. However, it does have the interesting capability of loading in data from the system clipboard (where data is saved when control-c, or copy, is hit). To do so, call `importdata('-pastespecial')`.

Plot and Graphing Capabilities

MATLAB's ability to graph data is by far one of its strongest features. It is able to do both two dimensional graphs and even full three dimensional graphs with camera controls and other such customizable features. The basic `plot` will be the one that likely gets the most use, but other two dimensional graphs include, `bar`, `scatter`, `pie`, and more. At its simplest, plot takes two vectors of equal length, the first value being the x values and the second being the y values. From there, a line is constructed. Multiple lines can be graphed in a single `plot` call by simply including two or more sets of x and y values in the call (e.g., `plot(x1, y1, x2, y2)`). `plot(x1, [y1 y2])` also works well if `y1` and `y2` are of the same length and you wish to use the same x value for both. Although `plot` can auto set the bounds and the axes to best fit the line, the user is free to override these. Other such features of the graph can be changed by the user such as line color and type, title, labels on axes and tick marks, and even the color of the background. Some of these can be changed with specific functions or by passing `plot` optional values, but others have to be changed by accessing the line, axes, or figure object handles.

Creating Functions

In MATLAB, the syntax for functions can be somewhat compared to the syntax of functions in algebra. An example of a function in MATLAB would be `function y = f(x)` where `y` is the output variable, `f` is the function name, and `x` is the input variable. Multiple values can be returned by a function. To do so, the output variables should be wrapped in brackets (e.g., `function [x, y] = mystery`). When assigning variables to a function with multiple return values, the variables should be written in brackets. However, if only the first value is

desired, `var = mystery()`, `[var] = mystery()`, and `[var, ~] = mystery()` will work. `[var1, var2] = mystery` will give the first return value to the first variable listed, and second to second, and so on. The `~` (tilde) can be used to ignore a specific return value. So, if I only wanted the third return value of a different function named `secret`, I could do `[~, ~, var] = secret(x, y, z)`. A variable amount of return values is possible with the keyword `varargout` (variable argument output), which is how the built-in function `size` operates. If no output variables are desired, then the return values can be left off of the declaration like with `function doSomething(x)`. MATLAB lacks a `return` statement. Instead, whatever the current value of the output variable is when the function ends is the value that is returned. In addition, each function has its own workspace, or scope, and will only have access to the variables it created or was passed when called.

The name of the function needs to be a valid identifier, so it can not start with a number. In addition, functions in MATLAB are typically written in a separate file in the same directory as the main program. Due to this, functions should have the same name as the file they reside in, or vice versa. Otherwise, MATLAB will issue a warning and the function will have to be called by the name of the file it is in, not its actual name. By placing them in a separate file, you can call the function from any other scripts or function files within the same directory. When a function is placed in a file with other code, it can not be called from outside that file and it must be terminated with an `end` statement. Ideally, all functions should be terminated by an `end` statement, but functions in their own separate file may be terminated by end-of-file instead.

Similar to output variables, MATLAB can have a variable amount of input variables. This is how the built-in function `plot` can take so many different values at once or even graph several lines at once. It is called `varargin` and can be combined with other input variables or just used by itself. `varargout` also has this ability. A couple examples would be `function [x, varagout] = func(r, s, varargin)` and `function func2(varagin)`. In addition, if no input variables are needed, empty parentheses can be used or even no parentheses at all. `function y = f()` and `function y = f` are equivalent.

Example
A short example of a function and its body is below in Table 2. Users are encouraged to make their functions vectorized. In other words, try to write your function in such a way that both scalars and matrices (of any dimension) can be passed in. Usually, this can be done by using the element-wise operators. This function converts fahrenheit temperatures to celsius and returns the converted value or values.

```
function newTemps = fahren2celsius(temps)
     newTemps = (temps - 32) .* (5 / 9);
end
```

Table 2, Example Function

Data Types and Conversions

MATLAB has eight basic data types. A logical value is a true or false value and can be considered to be similar to boolean types in other languages. Strings consist of characters surrounded by double quotes (quotation marks) while char arrays consist of characters surrounded by single quotes (apostrophes). Strings and char arrays are not the same thing and even though modern versions of MATLAB can use them interchangeably, older versions can not. As such, to maintain compatibility with older versions, char arrays should be preferred. Numeric values can be integers, floating points, or even complex numbers. Tables are used to store mixed type data and metadata and allows access with numeric or named indices. Cell arrays allow each individual element to be a different type and are often used in the creation of jagged arrays. Structs are very similar to structs in C, but struct arrays have some special properties. If I had a struct array `students` and I wanted all the students' grades, I could do `students.grades`. However, if I only needed the first three students' grades, I could do `students(1:3).grades`. Finally, there are function handles. Unlike all other data types, function handles can only be scalars, though there can be multiple function handles within a cell array or struct array. MATLAB has several built in functions for converting between data types, and most have the form *`currentType2desiredType`*. Some examples are `num2str`, `cell2table`, and `table2array`.

Libraries

In MATLAB, what are known as libraries in most programming languages are called "toolboxes". They serve the same function as in other languages; to expand the base functionality of MATLAB for a specific purpose. One example of a MATLAB toolbox is the Parallel Computing Toolbox. This toolbox gives MATLAB code the ability to take advantage of multi-core processors to complete complex tasks faster and more efficiently. As with any programming language that supports parallel computing/threading, the ability to take advantage of multiple cores allows the user to solve many more classes of problems.

One of the largest and most useful toolboxes for MATLAB is Simulink. In short, Simulink adds a graphical editor to MATLAB for modeling dynamic systems. For example: modeling electronic systems like microcontrollers and circuits, analyzing signals, robotics, and more. As the name suggests, Simulink allows the user to simulate a wide variety of systems, all within

MATLAB. Being within MATLAB, the user has direct control over the backend of the simulation being run.

MathWorks has a massive library of toolboxes available to download from their website. There are even toolboxes for computer vision and image processing, which, when combined with MATLAB's AI and machine learning toolboxes, creates a suite of features that enables MATLAB to work with some of the world's most complex traffic/image recognition problems. To put it simply, MATLAB has some extremely advanced toolboxes that allow it to accomplish virtually any task you would like. Hence, we have in our paper's title, "A glimpse into the future of humanity?". Companies can find a use for MATLAB in any major job field.

Debugging

When it comes to debugging, the MATLAB environment has several solutions. Users can place breakpoints in their script, and like other IDEs, this will allow the user to view the values of variables or step through the program line by line. Besides that, the workspace panel will show all current variables, their values, and their types at all times. After a program has run to completion, a user can view these variables and use them in other calculations in the command window. Even variables that are never printed to the screen are displayed in the workspace, so it is useful for confirming the values of in between calculations and temporary variables. To interact and modify the workspace while the program is running, the `keyboard` command can be used. This function returns control to the user. From there, they are free to read and write to variables in the workspace or make any other calculations they desire. Once finished, control is returned to the program with the command `dbcont`.

Finally, MATLAB will highlight errors in the script file with a red, squiggly underscore sort of like Microsoft Word or other word processing applications. These swiggles will exist even if the program has not been run yet, so it is a good idea to hover over them and see what the problem is and fix it before running the program. Unfortunately, MATLAB is not able to show all errors this way; some can only be caught by executing the script. In addition, warnings are issued in the same way but with yellow squiggles. Like before, hovering over these squiggles will explain the warning and may even offer a solution to fix it. When a script is free of errors and warnings, a green square will be visible above the scrollbar in the editor window.

Conclusion

MATLAB is a modern command line environment that also includes its own programming language, specializing in mathematical and linear algebra applications. Originally developed only for matrix operations for a handful of universities, MATLAB is now used by millions of people, and the program itself has expanded and evolved to be an extremely efficient tool for not

only mathematics, but also computer science applications like AI and machine learning, as well as engineering applications like dynamic modeling of electronic and mechanical systems.

Like other command line driven programs (such as R or Octave), MATLAB has syntax that can differ significantly from standard programming languages. However, this is rarely a drawback; the main user base of MATLAB does not have an extensive background in programming, and therefore will not need to undo any bad habits from traditional programming syntax. Additionally, the syntax that may be considered strange from a computer science perspective (such as one based indexing) is instead very useful and intuitive for making the program accessible for people unaccustomed to programming.

While MATLAB is already an exceptional tool in its base form, it really shines when you expand its capabilities with libraries, called toolboxes. These toolboxes add advanced functionalities like 3D modeling, machine learning, image recognition, and much more. The applications of MATLAB when equipped with the right toolboxes are endless.

References

MATLAB compiler. MATLAB. (n.d.). Retrieved September 17, 2021, from https://www.mathworks.com/products/compiler.html.

MATLAB runtime - MATLAB Compiler. MATLAB Compiler - MATLAB. (n.d.). Retrieved September 17, 2021, from https://www.mathworks.com/products/compiler/matlab-runtime.html.

Matlab app designer. MATLAB & Simulink. (n.d.). Retrieved September 17, 2021, from https://www.mathworks.com/products/matlab/app-designer.html.

Matlab. MathWorks. (n.d.). Retrieved September 17, 2021, from https://www.mathworks.com/products/matlab.html.

Fundamental MATLAB Classes. Fundamental MATLAB Classes - MATLAB & Simulink. (n.d.). Retrieved September 17, 2021, from https://www.mathworks.com/help/matlab/matlab_prog/fundamental-matlab-classes.html.

Simulink - simulation and model-based design. Simulation and Model-Based Design - MATLAB &. (n.d.). Retrieved September 17, 2021, from https://www.mathworks.com/products/simulink.html.

Tables. Tables - MATLAB & Simulink. (n.d.). Retrieved September 17, 2021, from https://www.mathworks.com/help/matlab/tables.html.

TutorialsPoint. (n.d.). MATLAB - Operators. MATLAB - operators. Retrieved September 12, 2021, from https://www.tutorialspoint.com/matlab/matlab_operators.htm.

What is MATLAB? (n.d.). Retrieved September 17, 2021, from https://cimss.ssec.wisc.edu/wxwise/class/aos340/spr00/whatismatlab.htm.

What is matlab? MATLAB & Simulink. (n.d.). Retrieved September 17, 2021, from https://www.mathworks.com/discovery/what-is-matlab.html.

Wikimedia Foundation. (2021, September 4). Matlab. Wikipedia. Retrieved September 12, 2021, from https://en.wikipedia.org/wiki/MATLAB.

Write Data to Excel Spreadsheets. Write Data to Excel Spreadsheets - MATLAB & Simulink. (n.d.). Retrieved September 17, 2021, from https://www.mathworks.com/help/matlab/import_export/exporting-to-excel-spreadsheets.html.

keyboard. Give control to keyboard - MATLAB. (n.d.). Retrieved September 17, 2021, from https://www.mathworks.com/help/matlab/ref/keyboard.html.

varargout. Variable-length output argument list - MATLAB. (n.d.). Retrieved September 17, 2021, from https://www.mathworks.com/help/matlab/ref/varargout.html.