An Introduction to Programming in MATLAB
Michael Sockin
February 20, 2015

1 What is MATLAB?

MATLAB is a statistical programming language used in a wide range of applications from simulating flight for aerospace design to signal processing and image analysis. Many basic useful operations that are readily available while more exotic functions, such as optimization or partial differential equation (PDE) solvers, are added through toolboxes. MATLAB stands for matrix laboratory and, as such, it likes to work with data in the form of vectors and matrices. This distinguishes it from other statistical packages like SAS, R, and Stata[1], which work with records (making it easier to associate variable names with data), and programs like Mathematica, which work with records and symbolic logic.

It is important to understand that MATLAB likes to work with vectors and matrices because you can drastically speed up the amount of time it takes for your program to run if you can rewrite "for" loops in terms of vector and matrix operations. MATLAB can do "for" and "while" loops, but they are not as efficient as in languages like "C".

2 Getting Started

The first thing to do when opening MATLAB is to choose your directory. The default folder for MATLAB is "work" but if you want to save your programs or data, it is useful to create a folder for MATLAB to use. Try creating a new folder "test" on your desktop. Then click the button of an open folder with a green arrow pointing out of it and select this "test" folder. What we have just done is given MATLAB a path to the "test" folder. This is important because MATLAB will first look in the path folder for functions that you call before looking in its own directories. Importantly, only functions or programs that you create in the "test" folder will be recognized by MATLAB.[2]

The Workspace on the RHS of the MATLAB screen saves all the variables you define and displays either their values or their dimensionality.

The Command Window is where you enter the commands you want MATLAB to run, which will often be the name of a function or script you have written.

[1] Stata does have a lesser-known matrix module, called MATA, that was introduced to compete with MATLAB.

[2] We could also switch to a directory using the "cd" command, which stands for "change directory". We can also use the "addpath" command to use functions from folders other than the designated directory.

3 Basic Operations

Addition and subtraction in MATLAB are done using the "+" and "-" signs, respectively, while multiplication and division are done using the "*" and "/" or "\" signs. Assignments of expressions to a variable are done with the "=" sign, where the variable is on the left hand side and the expression is on the right.

Example: To assign the value 2 to x, we would write
x = 2

Importantly, you can assign to a variable the results of a mathematical operation.
Example: To assign the value of .5 * (2 - 3/17) to x, we would write
x = .5 * (2 - 3 / 17)

The distinction between "/" and "\" for division comes in the assignment of the numerator and denominator. "\" treats the expression to the left of it as the denominator, and to the right as the numerator, while it is the opposite for "/".

Example:
x = 3/5

assigns to x the value 3/5, while
x = 3\5

assigns to x the value 5/3.

4 Some General Best Practices when Coding

Always make generous use of white space, which the amount of spacing between lines of code and between letters and numbers within lines of code. For instance, consider the same two lines of code:

x=17+.5*(12-8/5)

x = 17 + .5 * (12 - 8 / 5)

The second line is much easier to read and, by coding this way, you are less likely to make mistakes (or "bugs") and more likely to find any you have made.

Comment profusely in your code. Comments tell the reader what the line or section of code is meant to accomplish. This is helpful both for you when reading your own code (trust me, you quickly forget what you were doing after a month) and for others if you are sharing your code. Comments in matlab are created using the "%" sign.

Example: To comment that the assignment x = 5 assigns the value of 5 to the variable x, one would write
x = 5 %assigns the value of 5 to the variable x

We put a semicolon ";" at the end of a line of code to supress its output when you run the program or script. If you write x = 5 in the program "test", then MATLAB will display x = 5 every time you run "test". If instead you write x = 5; in "test", then when "test" runs it will not display x = 5.

The function "display()" displays whatever is in the parentheses when you run a program. Using "display()" can tell you when a certain part of a program has been reached or to display output to the user (such as the result of a series of computations). This is especially useful for diagnostics and debugging. For instance, MATLAB will not tell you where in a "for" loop the program errored out. By inserting a line "display(i)" where i is the index of the loop, MATLAB will display the number that the loop is on while it runs. You can then see on which iteration the loop specifically failed and figure out what went wrong.

The latest versions of Matlab also let you separate your code into models, using the "%%" command. Doing this tells MATLAB to higlight the part of the code you're working on in the editor. This is useful if you want to separate your code into steps.

Example: To create two sections of code, one for assigning variables and one for computations, one would write

```
%%
%Initialize Variables

x = 5;

y = 6;

%%Compute z, the sum of x and y

z = x + y; %sum of x and y
```

To find out how long a part of a program takes to run, the "tic" and "toc" commands are useful. When MATLAB sees "tic", it starts an internal clock and when it sees "toc" it displays the amount of time that has elapsed on this internal clock.

5 For and While Loops and If Statements

These objects are fairly standard in programming across languages. "For" loops increment over an index variable, typically i, and perform some operation with each iteration. The syntax for a "for" loop is to start with for along with the range of the index variable, and then to close the "for" loop with an "end". The general form of a "for" loop is then

```
for index = first:increment:last
statements
end
```

The "for" loop will automatically increase the index by the increment specified after each iteration of the loop.

Example: Suppose we want to add the first ten counting numbers together. Let the sum be S. Then we could do this by initializing S = 0 and running the following "for" loop

```
for i = 1:10

    S = S + i; %add to S the next counting number i

end
```

Importantly, if we have a vector n x 1 vector v, then we can use a for loop to perform some operation with each element of v. Suppose now that the vector v had the first ten counting numbers as its entries, or

```
v = [1,2,3,4,5,6,7,8,9,10]
```

then we could also construct the above sum S by initializing S = 0 and running the following "for" loop

```
for i = 1:10

    S = S + v(i); %add to S the next counting number i

end
```

One could also do this indexing with "for" loops with matrices.

"While" loops are similar to for loops except that they run until some exit condition has been met. The general form of a "while" loop is

```
while condition
statements
end
```

I advise avoiding "while" loops when possible, since you can accidentally program an infinite loop and MATLAB does not automatically terminate when this happens. To iterate on an index as with "for" loops, we would include the command i = i + 1 inside the while loop. For example, we could implement the above for loop as a "while" loop by initializing S = 0 and running the "while" loop

```
i = 0; %initialize index to be 0

while i < 10

S = S + v(i); %add to S the next counting number i

end
```

"If" statements are conditional statements that execute a segment of code only if the condition in the if statement is met. The general form of an "if" statement is

if condition
statements
end

For instance, if we want a program to display the value of z only if the variable z is positive, we would use the command

if z > 0

display (z);

end

We can create multiple conditional statements using the "elseif" and "else" clauses. For instance if we want a program to display the square root of z when z is positive and display "z is imaginary" otherwise, we would use the command

if z > 0

display (sqrt(z));

else

display("z is imaginary");

end

You can nest "if" statements just as you can nest loops in MATLAB.

6 Vectors

The advantage of MATLAB is that it is built for operations that use vectors and matrices.[3] To define a vector, we simply simply assign to it a bracketed object of entries. For instance, to create a vector v of the first ten counting numbers. We would then enter the command

v = [1,2,3,4,5,6,7,8,9,10]; %create a vector v of the first ten counting numbers[4]

The vector v is a 1 x 10 vector. To make it into a 10 x 1 vector, we could take its transpose. The transpose operation in MATLAB is the apostrophe or " ' " symbol. Thus if we want vt, the transpose of v, we would enter the command

vt = v'; %take the transpose of v

[3] An important caveat is that this is only really true in 1 (vector) or 2 (matrix) dimensions. MATLAB is pretty inefficient for handling cubes and higher-dimension objects.

[4] Alternatively, a much easier way of accomplishing this would be to write v = (a:c:b), where a is the starting value, b is the ending value, and c is the increment. In this case, we would write v = (1:1:10).

We can do a variety of operations with vectors. Let v and w be two vectors each of length n. We can add and subtract v and w with the "+" and "-" symbols, or v + w and v - w. We find the inner-product of these two vectors v' * w = $\sum_{i=1}^{n} v_i w_i$ or the outer product, which would be a matrix VW = v * w' which has elements $VW_{ij} = v_i w_j$.[5]

We can also raise each element of a vector v to a power p using the ".^" command. If we want to raise each element of a vector v to the 2nd power, to find v2, we would write the command

v2 = v .^ 2; %raise each element of v to the 2nd power.

We can also multiply and divide two vectors element-by-element using the ".*" and "./" commands.

Two useful functions for vectors is "find" which can be used to find specific elements in a vector, and "sort" which can sort elements in a vector. For instance, we could use the "find" function to find all the zeros in a vector with u = find(v == 0). The output u gives the indices of v whose elements are 0.

Vectorization in MATLAB is important because certain computations that can be written as "for" loops can be done a lot quicker using vectors. For instance, instead of writing a "for" loop that finds the sum of squares e2 of the elements of a vector v, one can use the command

e2 = v' * v; %find the sum of squares of the elements of v

7 Matrices

To define a n x n matrix M, we must specify components indexed by i and now also j. We first enter the first row of entries and then use the semicolon ";" symbol to tell MATLAB we now wish to define a new row. We use brackets to close the first (1,1) entry and the last (n,n) entry.

Example: Let us define the 3 x 3 matrix L which contains the first three natural numbers in row 1, the second three natural numbers in row 2, and the third 3 in row 3. The command for this would be

L = [1, 2, 3; 4, 5, 6; 7, 8, 9]; %define a 3x3 matrix of the first nine natural numbers

You do not have to use commas to separate entries in the same row, though it is best practice to do so. The identity matrix (a matrix with 1's along the diagonal and 0's otherwise), can be created with the function "eye". To create the 3 x 3 identity matrix I3, one enters the command I3 = eye(3).

[5] We can also divide two vectors w \ v if w and v are the same length to give us the projection of v onto w. This is just the coefficient in a linear regression.

To see this, notice that we can rewrite b = w\v as wb = v and then premultiply by w' to arrive at b = (w'*w) \ (w'*v) since the inner products are scalars.

To invert a matrix, one can use the "inv" function, though this can sometimes have bad properties. It is preferable to define matrix inverses using the "/" and "\" symbols. For instance, to invert the 3 x 3 matrix A to find its inverse iA, it is preferable to use the command iA = eye(3) / A.

Matrices can be multiplied together as long as they are conformable (i.e. matrices M and N can be multplied together to form M * N provied that M has the same number of columns as N has rows).

The transpose of a matrix can be found using the apostrophe "'" symbol, the same as for vectors.

To find the determinant of a matrix, one can use the "det" function and to find its rank the "rank" function.

To pull out a row or a column of a matrix into a separate vector, one can use the ":" symbol to select all the rows or columns in a matrix. For instance, to pull the second column of a matrix M into a separate vector m, one would use the command

m = M(:,2); %take the second column of matrix M

Similarly, to extract the first row

m = M(1,:); %take the first row of matrix M

The ":" symbol is also really useful when you want to assign the output a function to a row or column of a matrix if you are collecting the output from multiple trials / runs. If the ith iteration of running a function regress outputs the coefficient vector b, you can populate the ith row of a matrix B that collects the results with the assignment function

B(i,:) = b; %save the ith output of function regress in the ith row of the matrix B

This is extremely useful for simulations.

8 Strings

Strings are essentially text that is saved in an object, whether it be a variable or a vector. To define a string, use single quotation marks ' '. For instance, to define the string "MATLAB string" as a variable x, we would use the command

u = 'MATLAB string'.

Notice that the length of u is 13, which suggests that MATLAB stores the string by its 13 characters. When importing data, however, it is possible that the entire string is encoded as 1 character to MATLAB. We use the function "char" to convert this string into its component characters.

Useful functions associated with strings are "num2str" and "str2num" which convert between the two data types and "strmatch" if you are doing textual analysis and want to compare strings.

9 Calling Functions

MATLAB has a huge library of functions that do a variety of different things at its disposal. We can call these functions by typing their name followed by their inputs in parentheses. We can assign the output of these functions by putting them on the LHS of an equality expression with the function call on the right.

Suppose we wanted to call the function "regress" which performs a linear regression of a vector y onto a vector x. Notice here that the input to these functions are vectors (or a matrix for x). It is important to read the help file for a fuction to understand how to properly call and use it. Type "help" + the name of the function into the command line to display its help file. To see the To use "regress", which outputs the coefficient on x, b, we would time the following into the command line

b = regress(y,x); %regress y on x to find b

Notice in the help display for regress that you can output additional objects, such as the confidence interval for b and the R^2 of the regression. We accomplish this by adding additional LHS output variables in the above expression

[b, bint, R] = regress(y,x); %regress y on x to find b, confidence interval for b, and the R^2 of the regression

Notice also from the help display that we can enter additional inputs, such as the alpha for the (1 - alpha)% confidence interval. Suppose we want an alpha of .05, then the command becomes

[b, bint, R] = regress(y,x,.05); %regress y on x to find b, 95% confidence interval for b, and the R^2 of the regression

If we did not enter a value for alpha, we would have to read the help display to find the default value.

10 Random Number Generation

MATLAB has many built-in functions for generating random variables. To generate uniform random variables, we use the function "rand()". For instance, to generate a 3 x 1 vector of 3 independent uniform random variables r, we would use the command

r = rand(3,1); %generate a vector of 3 uniform random variables

One can also do this with standard normal random variables, using the "randn" command. For instance, if we wanted to generate a 3 x 4 matrix of standard normal random variables M, we would use the command

M = randn(3,4); %generate a 3 x 4 matrix of standard normal random variables

If instead we want the normal random variables of M to have a mean m and a variance v, we would use the "normrnd" function. The command for this would then be

M = normrnd(m,v,3,4); %generate a standard normal random variable

If instead we want to know the value of the cdf of a normal random variable p with mean m and variance v at a point Z, we would use the "normcdf" function. The command would be

P = normcdf(Z,m,v); %find the cdf of a normal random variable with mean m and variance v at the point Z

One can also do this to find the pdf at the point z using the function "normpdf". We can find the point P that corresponds to the a value for the cdf of a random variable using the "norminv" function.

Similar operations for generating random variables and finding their pdfs and cdfs at a point can be done for other distributions. For instance, the function for generating random variables from a beta distribution is "betarnd", and from a gamma distribution is "gamrnd".

11 Plotting

Matlab has many commands for displaying data graphically. The basic plot function is "plot" in which you specific your x data and y data in vector form using the command

```
plot(x,y);
```

One can plot multiple series (x1,y1) and (x2,y2) on the same set of axes by adding more inputs to plot

```
plot(x1,y1,x2,y2);
```

One can plot two different y variables on different axes with the plotyy command

```
plot(x,y1,y2);
```

One can add labels to the x and y axes, as well as a title and legend (for multiple series), with the "xlabel", "ylabel", "title", and "legend" commands, respectively. Thus, for the two series example, one can add to the original plot with the command

```
plot(x1,y1,x2,y2);
xlabel('Time');
ylabel('Position');
title('Plot of Position vs Time for Two Trials');
legend('Trial 1', 'Trial 2');
```

MATLAB assigns default colors to multiple series in plots, typically the first being blue, the second red, and the third green. One can control the color of a line in a plot using the 'color' option. For instance, to make a plot of one series (x,y) with a red instead of a blue line, one can use the command

plot(x,y,'color','r');

MATLAB has symbols for different colors for lines. In this case "r" is the symbol for red. One can also specificy the RGB triple if one was so inclined. Another option is to change a plot of a solid line to a dash or dotted line. One can use the options "–", ".", in the plot command to create a dashed or a dotted line. For instance, to make the plot of one series a red dashed line, one would use the command

plot(x,y,'.','color','r');

To plot multiple graphs on the same plot, one can use the "subplot" command.

An extremely useful set of functions for plotting data with dates for time-series plots is to use the "datestr" and "datenum" commands to convert strings of dates from data into date numbers that Matlab recognizes. For instance, the date 11/13/2013 is equivalent to the Matlab number 735551. This can be found by using the command

datenum('11/13/2013'); %find the Matlab date associated with 11/13/2013

With dates as Matlab numbers, one can then plot a time-series using the Matlab numbers as the x axis and specifying the function 'datetick' to format the x axis labels.

There are other types of plots one can do. For instance, the command to make a histogram plot is "hist", which by default groups the data into 10 buckets.

12 Functions and Scripts

Programs in MATLAB can be either functions or scripts and are saved as m-files (with the ".m" extension). Commands are entered in the command line while programs are written in the matlab editor (though in priciple, one could write the entire program in the command line).

If you just save lines of code that you want to execute all at once, then it is called a script. For instance, if you save a program as "hello.m" and type "hello" into the command line, then the program "hello" will run. Scripts are useful if you do not have to enter any information as the user to run the program.

Functions are ".m" files that allow the user to enter information, such as parameter values or data, into the program. A function is created using the syntax

function void = functionname()

where void is a placeholder for the output of the function. Inside the parentheses, we could put variables that the function will later use. Importantly, a function must always have the keyword "end" at the end of the program.

Example: Suppose we want to write a function FindSum that adds two numbers entered by the user together. Let's call these two numbers a and b and their sum c. Then we could write this function as

```
function c = FindSum(a,b)

c = a + b; %add a and b together

end
```

It is important to save a ".m" file of a function with the same name as the function. In this case, we could label the ".m" file "FindSum.m". MATLAB is case-sensitive.

Importantly, functions can be nested within a ".m" file. This is useful when building longer programs because MATLAB essentially uses different memory management when it calls a function (this has to do with "global" vs "local" fields). Essentially, MATLAB will use separate memory when it calls a function within a function, and then will delete all of this memory except for the output when it passes the output back to the main function. This is also an essential part of doing optimization with MATLAB.

Example: suppose calling FindSum was a step in a larger program FindProd that multiplied the sums of the two pairs of numbers, a and b, and d and e. Then we could write this program in one ".m" file named "FindProd.m" as

```
function g = FindProd(a,b,d,e)

c = FindSum(a,b); %add a and b together

f = FindSum(d,e); %add d and e together

g = c * f; %multiply c and f together

end

function c = FindSum(a,b)

c = a + b; %add a and b together

end
```

Notice that I defined the subfunction "FindSum" after I finished defining "FindProd" and also that the "m" file was named after the outermost function "FindProd". In principle, we could define more subfunctions and just define then after "FindSum" in the ".m" file.

You can also assign multiple outputs from a function. For instance, suppose we want FindProd to give us the product of the two sums (a + b) and (d + e), but now we also want it to tell us whether the product is larger than 10. Let the binary variable that indicates whether the product is larger than 10 be h (h will be 1 if it is larger than 10 and 0 otherwise). This function "FindProd2" would now be defined as

function [g, h] = FindProd2(a,b,d,e)

c = FindSum(a,b); %add a and b together

f = FindSum(d,e); %add d and e together

g = c * f; %multiply c and f together

h = logical(g > 10); %assign to h 1 if g > 10 and 0 otherwise

end

Importantly, the outputs of the function do not have to have the same structure. In this case, g is a number and h is a boolean variable. More generally, you can have a function output a number, a vector, a matrix, and strings all at once. In the above "logical" is a function that evaluates the expression and returns a 1 if the statement is true and 0 otherwise.

Functions have local domains so when you call a function only the output of the function will be saved in your Workspace.

13 Importing and Exporting Data

MATLAB has several functions for importing data depending on its format. To import data formatted in comma-separated-values, or csv data one would use the function "csvread". If, instead, the data is an excel file, one would use the function "xlsread". For ASCII data, or data written in text files, one can use the function "textscan" or "fopen" in conjunction with "fscanf" and "fclose". You must specify the extension of the file being read if it is not in the Current Directory.

With functions like "csvread" and "xlsread" you can specify the row and column at which to start reading in data. If you read in an xls file with column names and strings, the column names and strings will be put in a separate matrix from the numerical data. Therefore, if you are trying to import both numbers and strings from an xls file, you should specify that it output two variables containing the data.

To write data to a file, one would use the functions "csvwrite", "xlswrite", and "fwrite" along with "fopen". These functions are used to output objects such as vectors and matrices to data files.

You can also save objects using the "save" function to save it as a ".mat" object that can be called into MATLAB's workspace using the "load" function.

14 Optimization

MATLAB has many functions for optimization an objective function. MATLAB optimizers minimize functions, so if you wish to maximize an objective, just minimize negative of this objective. Many of MATLAB's optimization algorithms are gradient-based linear search methods, which are highly dependent on having a good starting value. This also means that these optimizers find local maxima/minima, so it is a good idea to try different starting values and compare results.

The most robust, but also the most time-consuming, is "fminsearch". This is a triangular search algorithm that does not rely on gradients, so it will not get stuck as easily if the objective is not very well-behaved.

The basic gradient-based search algorithm is "fminunc", where the "unc" stands for "unconstrained". There is also "fmincon", which performs optimization subject to constraints, but in practice this algorithm is not very well-behaved.

MATLAB also has "lsqnonlin" which is ideal for minimizing objectives that can be expressed as a sum of squares, such as minimizing the sum of squared residuals of a model's fit of data.

Optimization algorithms all have the same syntax for the first two arguments, which is the objective function followed by a starting value or initial guess. You can use user-defined functions in opitimization algorithms, provided the function is either defined in the ".m" file or in the Current Directory.

To understand how to properly code the objective function for the optimization algorithm, we must first understand handles and anonymous functions, which are very useful in practice. A handle is a tool for being able to assign a function to a variable. For instance, suppose we have the function "FindProd" from above and we want to assign it to a handle h. Then we would use the command

h = @FindProd; %assign function FindProd to handle h

Then we can treat h as if it were the function FindProd. Handles are useful as a shorthand, since typically functions will have multiple arguments and we could define the handle to be the function we want with certain parameters fixed.

We can also define an anonymous function this way. An anonymous function is a function that is not explicitly defined using the "function" command. For instance, suppose we wanted to have an anonymous function q that square numbers. The we could define this function with the command

q = @(x) x^2; %create anonymous function q that suqares numbers.

Then if we wanted to square a number x, we could just enter the command q(x).

Handles and anonymous functions allow us to enter objective functions into optimization algorithms. Suppose we wanted to minimize x^2. Then we can make use of "fminunc", by defining an anonymous function with an initial guess of 1 and the following command

```
x0 = fminunc(@(x) x^2,1);
```

The first output of fminunc is the value that minimizes the objective function.

15 Eigenvalues / Eigenvectors

To find the eigenvalues and eigenvectors of a square matrix, one can use the "eig" function. For instance, if A is the 2x2 matrix [1 2; 2 1], the command to find its eigenvalues L is

```
[V,D] = eig(A); %find diagonal matrix of eigenvalues D of A and eigenvectors V
```

16 Structures

Another useful object in MATLAB is a structure. This is MATLAB's incorporation of object-oriented programming. A structure is basically a giant filing cabinet where you can store almost anything, regardless of data type. This is done by defining subfields. Subfields are extensions of the structure object, added after a period "." following the structure name.

Example: Suppose I want to define a structure S that holds the number n, the string s, and the vector v = [1,2,3,4,5]. This can be built with the following commands

```
S.number = n; %assign the number n to a subfield "number" of structure "S"
S.string = s; %assign the string s to a subfield "string" of structure "S"
S.vector = v; %assign the vector v to a subfield "vectoe" of structure "S"
```

Importantly, we can call each of these objects using their subfield extension in programs. This is very useful when defining inputs and outputs of functions. For instance, we can take the number in S and multiply each element of the vector by it to arrive at the vector V. This is accomplished with the following command, after defining the structure,

```
V = S.number * S.vector; %multiply each element of vector v by the number n
```

17 What is a NaN?

NaN stands for "Not a Number". One encounters a NaN when trying to import files with both numbers and text or when there are missing values. NaNs also occur when you encounter undefined objects, such as 0/0 or inf-inf (inf is the MATLAB symbol for infinity).

You can check if your variables have NaNs using the isnan() command. Importantly, NaNs can ruin computations because adding numbers to a NaN returns a NaN. It is important to check how functions deal with NaNs. For instance, "regress" treats NaNs as missing values rather than returning the not so useful result that the coefficient is NaN.

18 Debug Mode

MATLAB has a "debug" mode. Simply click on the black dash "–" to the left of a line of MATLAB code in the MATLAB editor to have the program run up until that line. A red circle will appear over the black dash. Importantly, MATLAB will save all variables up to that line of code since functions usually erase all of the variables they generate to create their output. This is useful because you can then check that variables are initializing correctly and that loops and other operations are doing what you intended them to do. One can debug a program by having the program run a little further each time before it stops in debug mode until the entire program has been run.

A good practice when trying to solve a problem in statistical software like MATLAB is to break the problem down into smaller steps and then to write your code sequentially one step at a time. This will make it much easier to write your program because debugging a module that does one specific step is much easier than debugging an entire program that performs all thes teps at once. The Matlab Cell structure for coding (using the %% command) helps you do this.

WHEN IN DOUBT, GOOGLE IS YOUR BEST FRIENDS!

19 Exercises

19.1 Exercise 1

Please download the file "Exercise1.xls" into your current directory. For the first exercise, please write a matlab program that first imports the data in the csv file into MATLAB and then saves the "time" data to a variable t and the "position" data to a variable y.

Then please save these variables to the current directory using the "save" command. Save the t variable as "time" and the y variable as "position".

Lastly, please plot the "position" versus the "time" variable, with a title and x and y labels, and then plot velocity versus the "time" variable, where the velocity is defined as change in position divided by change in time. For the velocity plot, please make the velocity curve a dashed red line.

19.2 Exercise 2

Please generate 3 vectors of standard normal random variables of length 100. Then regress the 2nd two vectors on the first and output the residuals to a csvfile labeled "residuals.csv". If you need help, type "help regress" into MATLAB's command window.

19.3 Exercise 3

For the final exercise, please minimize the objective function $\frac{1}{2}$x'Ax - C'x, where

$$A = \begin{bmatrix} 1 & \frac{1}{2} \\ \frac{1}{2} & 1 \end{bmatrix}, \; C = \begin{bmatrix} 2 \\ 1 \end{bmatrix},$$

with the "fminunc" optimization routine.

The answer is x = [-2; 0].