Lab 0: Introduction to MATLAB

MATH 5110: Applied Linear Algebra and Matrix Analysis
Northeastern University
Author: Nate Bade

August 29, 2019

1 Introduction to MATLAB

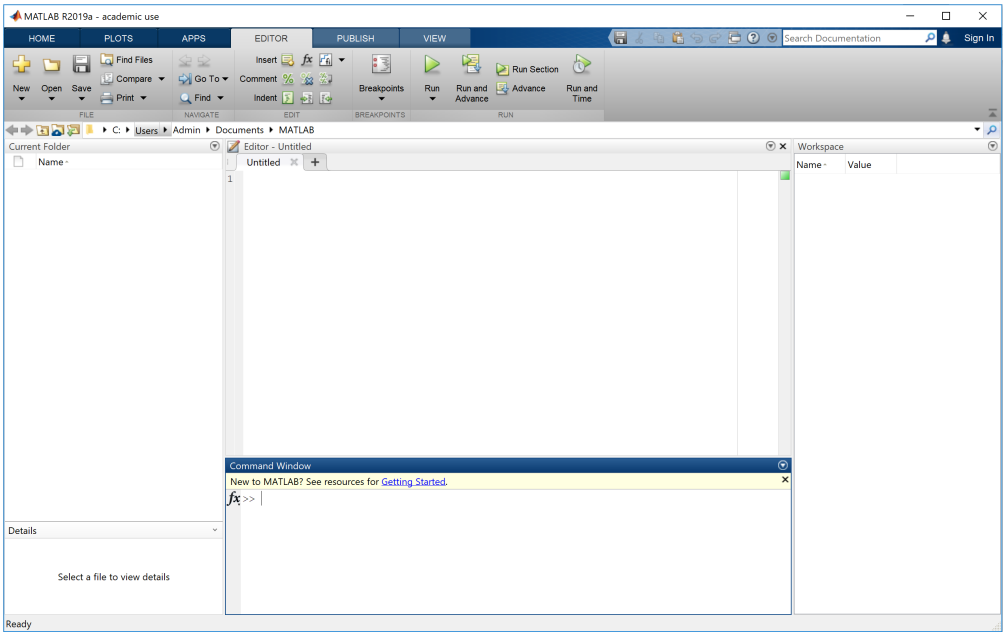
MATLAB, or the MATrix LABoratory, has been built from the ground up to run matrix computations quickly and efficiently. Although most computationally intensive tasks are eventually ported to C, MATLAB serves as a rapid prototyping language; easier than C and able to provide a quick proof of concept. In addition, its many plugins mean that MATLAB is an out of the box solution for many scientific computing and engineering tasks, from solving differential equations to signal processing, to interfacing with ARDUINO boards.

In this first tutorial, we will run through the basics of computing and graphing with MATLAB. We will talk about visualizing data and constructing arrays, vectors and matrices. This lab is an introduction to MATLAB as a tool.

To install and activate MATLAB as a Northeastern student, ~~follow the information found here: https://northeastern.service-now.com/kb_view.do?sysparm_article=KB0012568.~~ You only need to download **MATLAB** and **SIMULINK** for this course, but you may add any other packages your are interested in. (See syllabus for how to download Matlab.)

1.1 Introduction:

Upon opening MATLAB, you'll be greeted by a four part screen as below:



This screen contains

- **Current Folder**: The directory where your scripts and data will be stored.
- **Editor**: For editing scripts and storing code. If this does not appear click "New" and then "Script."
- **Workspace**: A list of the currently initialized variables.
- **Command Window**: A place to enter single commands, like on a calculator.

First steps: MATLAB as a giant calculator At its simplest, MATLAB's command window can be used as a giant calculator. To start, enter

```
>> 1+1
```

in the command window and press **Enter**. Two things will happen: First, MATLAB will return

```
ans =
     2
```

and second a new variable will appear in the workspace called `ans` with the value `2`. The `ans` variable will always contain the result of the last computation performed on the command window.

If you want to store the result of a computation for later use, you can assign a variable to the result using `=` as so:

```
>> myvar = 5 + 7 + 9
```

After you hit Enter, `myvar` will appear in the workspace containing the result of the calculation `5 + 7 + 9`. You may then use `myvar` in any future computation

```
>> myvar*8
```

or assign a new value to it

```
>> myvar=5 + 7 + 9
>> myvar=400 / 20
```

or both

```
>> myvar = myvar/100
```

Question: What are the results of each of the computations above? If you perform them in a different order, do you get the same result?

Operators and Order of Operations: MATLAB has the following common operators:

Symbol	Name
+	Addition
-	Negation and Subtraction
*	Multiplication
/,\	Division
-	Exponentiation

The precedence of the order of operations from highest to lowest is `()`, `^`, `-` negation, `*`,`/`,`\`,, then `+`,`-`. Take a moment to think about how

```
>> -10^-4/2+(10+2^1*2)
```

will evaluate, then check your answer.

Command Window vs Editor While all commands can be typed into the command window, it's often better to use the script editor for longer scripts. Click **New** and then **Script** to bring up a blank script. All commands can be typed into the script editor, and then run all together by pressing **Run**.

1.2 Vectors

A **vector** in MATLAB is just a list of numbers separated by spaces or ,'s for **row vectors**,

```
>> vec1 = [1 2 3 4]
>> vec2 = [7, 10, 30, 50]
```

or by semicolons ; for **column vectors**:

```
>> vec3 = [-1; -4; -9; -16]
```

A column vector can be switched to a row vector (and visa versa) by **transposition** using the apostrophe ' as so:

```
>> vec3'

ans =

    -1    -4    -9   -16
```

You can access the elements of a vector using their position starting at 1, so `vec2(1)` is `7` and `vec3(4)` is `16`.

Vector Operations Vectors of compatible dimensions can be added

```
>> vec1 + vec2
```

multiplied by scalars

```
>> vec1*4
```

have a scalar uniformly added to them

```
>> vec2 + 4
```

Note that trying to add vectors of incompatible dimensions produces something strange:

```
>> vec1 + vec3

ans =

     0     1     2     3
    -3    -2    -1     0
    -8    -7    -6    -5
   -15   -14   -13   -12
```

What has happened here is that MATLAB is constructing a matrix by adding each row entry of `vec1` to the each column of `vec3`. This is rarely used, but can be a convenient way to construct a matrix.

What about vector multiplication? MATLAB is built for matrices, so it will always assume that multiplication like `vec1*vec2` is **matrix multiplication**. Try running

```
>> vec1 * vec2
```

and note the error that you get: `Error using * Incorrect dimensions for matrix multiplication.` It's telling you exactly what has gone wrong, that is that your matrix multiplication has incorrect dimensions. The error messages in MATLAB are quite informative, always remember to read them.

Finally, if we want component-wise multiplication we need to use a `.*`:

```
>> vec1 *. vec3
```

Question: With `vec1`, `vec2` and `vec3` defined as above, which multiplications, for example

```
>> vec1 *. vec2
>> vec1 * vec2
```

will produce valid results? Check you answer with MATLAB.

1.3 Matrices

(See also: https://www.mathworks.com/help/matlab/learn_matlab/matrices-and-arrays.html)

A **matrix** in MATLAB is just a two dimensional vector, and is defined by specifying it's entries row by row. For example, the 2×3 matrix

$$\begin{pmatrix} 1 & 3 & 5 \\ 7 & 9 & 11 \end{pmatrix}$$

can be constructed in MATLAB by

```
>> mat1 = [1,3,5;7,9,11]
```

It is also common to construct a matrix as a vector of vectors:

```
>> mat2 = [[2,4,6];[8,10,12]]
```

As before, we can specify the elements by their position so `mat2(1,1)` is `2`, while `mat2(2,3)` is `12`. Given a matrix, we can get the number of elements using the `numel` function and the dimensions using the `size` function:

```
>> numel(mat1)
>> size(mat1)
```

Note that size itself returns a vector, a 1×2 row vector `[COLUMNS, ROWS]`.

MATLAB allows the basic matrix operations on a matrix `M`:

- `M*` - Matrix multiplication, provided dimensions are compatible.
- `M.*` - Component-wise multiplication, provided dimensions are compatible.
- `M'` - Matrix transpose.
- `inv(M)` - Matrix inversion, provided matrix is square and invertible.

For example, given our matrices above we can compute the inverse of `mat1` times the transpose of `mat2` by

```
>> A = inv(mat1 * mat2')
```

Question: What is the result of multiplying `A` by `mat1`?

1.4 Constructing Vectors and Matrices

MATLAB has some built-in ways to make matrix construction easier:

Sequences: The code `3:10` will return the vector containing all integers from 3 to 10 as a row vector:

```
>> vec1 = 3:10
```

Similarly, the code `3:4:20` will count starting at 3 and adding 4 each time until it is above 20, returning a vector of all the numbers in the count *less than or equal to* 20. In this case,

```
>> vec2 = 3:4:20
vec2 =

     3     7    11    15    19
```

It does not contain 20, because 20 is not a multiple of 4 more than 3, 20 just serves as an end point. However,

```
>> vec3 = 0:4:20
vec3 =

     0     4     8    12    16    20
```

will contain 20. We can also count down, for example

```
>> vec4 = 20:-2:1
```

returns the numbers we get when, starting from 20, we add -2 until we are less than 1.

Matrix Construction Functions: MATLAB also has some basic matrix construction functions:

- `zeros(N,M)` - Produces an N by M matrix of 0's.
- `ones(N,M)` - Produces an N by M matrix of 1's.
- `rand(N,M)` - Produces an N by M matrix of uniformly distributed random numbers between 0 and 1.

Additionally, you can use sequences to build matrix columns. For example, we can make the matrix

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 4 & 6 & 8 & 10 \\ 4 & 4 & 4 & 4 & 4 \end{pmatrix}$$

by

```
>> mat = [1:5 ; 2:2:10 ; 4*ones(1,5)]
```

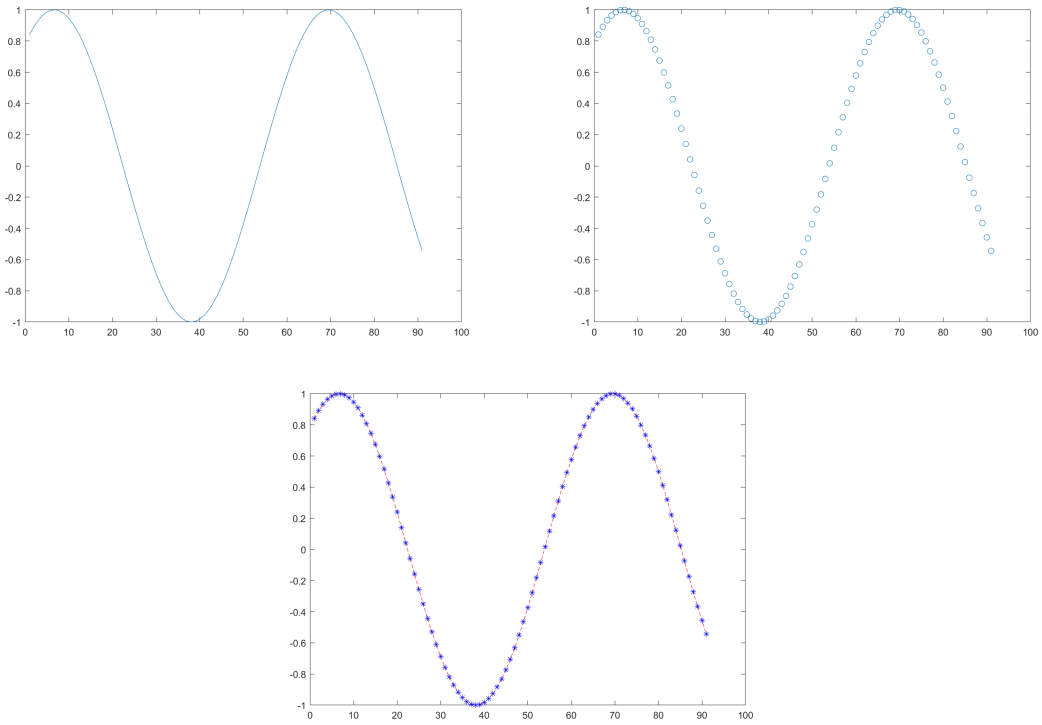
Question: Try using matrix addition, multiplication, and the matrix construction functions above to build

- A 2×5 matrix with -1 in the first row and the odd numbers starting at 5 in the second row.
- A random matrix with all numbers between 20 and 30.
- (Challenge) Multiply two vectors to produce a 100 by 100 matrix

$$\begin{pmatrix} 1 & 2 & 3 & \\ 2 & 4 & 6 & \dots \\ 3 & 6 & 9 & \\ \vdots & & & \ddots \end{pmatrix}$$

1.5 Visualization with MATLAB

MATLAB's `plot` function can be used to display a wide variety of 2D visual information and we will use it frequently in this course to generate graphs and display linear transforms. The `plot` function displays a sequence of points in 2d connected by lines. The markers for each point and the line styles can be edited, or turned off completely.



To plot points (x_1, y_1), (x_2, y_2), ..., (x_n, y_n), we must supply the plot function with a list of the x-values, `X = [x_1, ..., x_n]` and a list of the y-values `Y = [y_1, ... , y_n]`. For example, to plot a graph of a function like sine or cosine from 1 to 10 we write

```
>> X = 1:10
>> Y = sin(X)
>> plot(X,Y)
```

Question: Looking at `X` and `Y`, we are only plotting 10 data points: $(1, \sin(1))$, $(2, \sin(2))$, etc. How do you make the plot smoother by plotting all the points from 1 to 10 up to two decimal places, for example $(1.01, \sin(1.01))$.

The plot function has many options the are detailed in the documentation (https://www.mathworks.com/help/matlab/ref/plot.html). All attributes can be explicitly defined using property flags like `LineWidth`, but many can also be defined using a character vector, with

Line Style	Description	Line Style	Description
-	Solid line (default)	--	Dashed line
:	Dotted line	-.	Dash-dot line

Marker	Description	Marker	Description
o	Circle	+	Plus sign
.	Point	x	Cross
d	Diamond	^	Upward-pointing triangle
>	Right-pointing triangle	<	Left-pointing triangle
h	Hexagram		

Color	Description	Color	Description
y	yellow	m	magenta
r	red	g	green
w	white	k	black

For example, to plot a red dashed line with crosses for markers we would use

```
>> X = 1:10
>> Y = sin(X)
>> plot(X,Y,'r--x')
```

If you don't specify a line style, MATLAB will not plot the line.

```
>> X = 1:10
>> Y = sin(X)
>> plot(X,Y,'rx')
```

Question: Plot cosine from $-\pi$ to π with blue point markers and no connecting lines. You may use `pi` for the value of π.

Titles and Labels To make our plots readable, we should always include at least a title and if applicable axis labels and a legend.

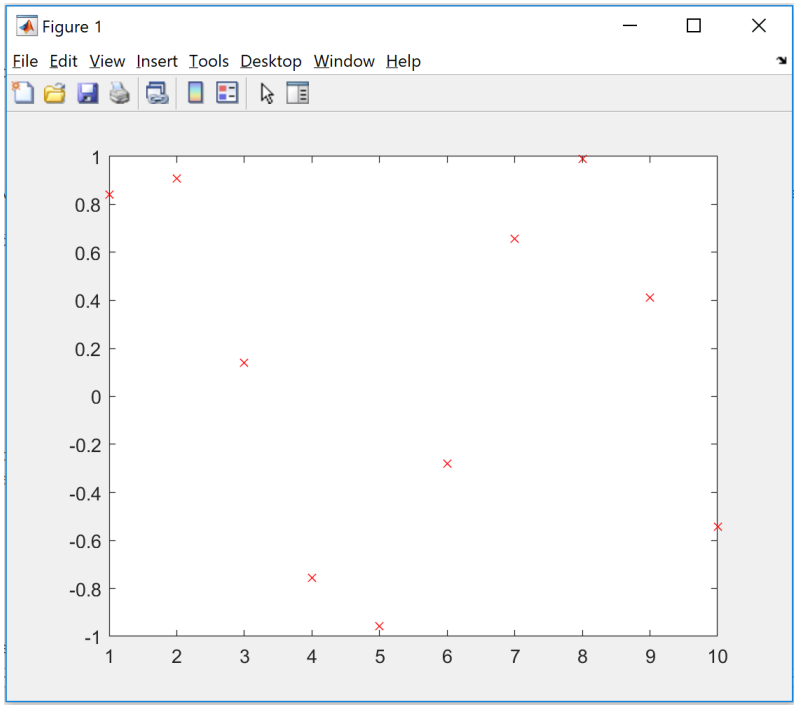
- `title('Plot Title')` - Add a plot title.
- `xlabel('X-Axis Label')` - Add a label to the horizontal axis.
- `ylabel('Y-Axis Label')` - Add a label to the vertical axis.

For example, use

```
>> plot(X,Y,'rx')
>> title('Cosine Function')
>> xlabel('Independent Variable')
>> ylabel('Dependent Variable')
```

to plot and label the cosine function.

Figures and Multiple Plots Each plot in MATLAB lives inside a **figure**. A figure is the window containing the plot.



The `figure` function creates a new figure. You can create multiple figures if you want to generate multiple plots, or you can plot multiple things to the same figure. For example, to plot two functions on different axis, we use

```
>> f1 = figure
>> plot(X, sin(X))
>> title("Sine")
>> f2 = figure
>> plot(X, cos(X))
>> title("Cosine")
```

After we create a new figure, all of the actions we perform will be on that figure. If we want to return to the figure with $\sin(x)$ and change something there, we use `figure(f1)` to reactive it:

```
>> f1 = figure
>> plot(X, sin(X))
>> title("Sine")
>> f2 = figure
>> plot(X, cos(X))
>> title("Cosine")
>> figure(f1)
>> xlabel('The Domain')
```

You may have noticed that each time we call `plot` it erases the previous plot. What if we want to plot two charts on the same axis? There are two options: We can specify multiple charts within the plot function by listing them as `plot(x,f_1(x), x, f_2(x),...)`

```
>> f1 = figure
>> plot(X, sin(X), X, cos(X))
```

or use `hold on` to hold the current contents of the axis while we draw to it:

```
>> f1 = figure
>> hold on
>> plot(X, sin(X))
>> plot(X, X.^2)
>> hold off
```

In the above `hold off` tell MATLAB to stop holding the contents of the current axis. Note that `X.^2` computes x^2, the `.^` tells MATLAB to exponentiate each component instead of trying to exponentiate as a matrix. For more examples take a look here (https://www.mathworks.com/help/matlab/creating_plots/combine-multiple-plots.html)

Question: Plot the function $x/(x^2+1)$ and the function $\cos(1/x)$ from 1 to 4 on the same plot.

1.5.1 2D Plotting

In addition to plotting graphs of functions, we can perform 2D plotting by specifying the x and y coordinates of points. For example, we can draw the diamond with points at (1,0), (0,1), (-1,0), (0,-1) by writing the x coordinates in one vector `X` and the y coordinates in another vector `Y` and passing them to the plot function:

```
>> X = [1,0,-1,0]
>> Y = [0,1,0,-1]
>> plot(X, Y)
```

Notice that MATLAB doesn't complete the square because we didn't tell it to connect the last point back to the first. To do so we just add another copy of $(1,0)$ at the end:

```
>> X = [1,0,-1,0,1]
>> Y = [0,1,0,-1,0]
>> plot(X, Y)
```

Question: Plot a 3x3 grid with no connective lines and circle markers. Can you find a clever way to define the vectors `X` and `Y` that makes the code simple?

Axis Boundaries: Often, it's alright to let MATLAB figure out the axis boundaries itself, but (as you may have noticed in the question above) sometime we want to choose the axis boundaries ourselves. To do so, we have use `axis` function:

- `axis( [x_min, x_max, y_min, y_max] )` - Sets the axis so that the horizontal view runs between $[x_{min}, x_{max}]$ and the vertical view runs between $[y_{min}, y_{max}]$.

For example, to display our square on a scale with x from -2 to 2 and y from -1 to 1, we write

```
>> X = [1,0,-1,0,1]
>> Y = [0,1,0,-1,0]
>> plot(X, Y)
>> axis( [-2,2,-1,1] )
```

Labels: With 2D plotting we often want to add labels. The labels support some latex and can be used to annotate points:

- `text(x,y,'My Text','left')` - Add the text `My Text` with the left anchor at the point (x, y). You may specify `'right'` or `'center'` as well for alignments, and if nothing is specified left is assumed. (https://www.mathworks.com/help/matlab/creating_plots/add-text-to-specific-points-on-graph.html)

For example, lets add a label to the point $1, 0$ in the diamond:

```
>> X = [1,0,-1,0,1]
>> Y = [0,1,0,-1,0]
>> plot(X, Y)
>> axis( [-2,2,-1,1] )
>> text(1,0,'(1,0)')
```

1.6 Scripts and Functions

Typing code into the command line is repetitive, and often makes the code hard to keep track of. That's why usually with MATLAB we use **scripts** to keep track of large blocks of code and **functions** to keep track of code that we want to use repeatedly with slight changes. For example, `plot` is a function, that we can call with many different parameters.

Scripts and functions can be written using the **Editor** part of the window, and a script may be run using the **Run** button. Each line of the script is exactly like a line type into the command window, and they are run sequentially. For example, typing

```
X = [1,0,-1,0,1]
Y = [0,1,0,-1,0]
plot(X, Y)
axis( [-2,2,-1,1] )
text(1,0,'(1,0)')
```

into the editor window and clicking Run (you'll be prompted to save the script, do so) will run each of the lines sequentially. This makes it much easier to store code and reuse it later.

Lets say the we want to trace out a diamond with a different size. We could just the `X` and `Y` vectors above, but if we want to do this many time we might want to use a function. For functions, we need to use a new .m file so go to "New" and then "Script" to open a new editor window. A function takes the form

```
function output = function_name(var1, var2,...)
    Do Something
    Do Something Else
    Etc

    output = The Output
end
```

For example, for our differently sized diamonds we could write

```
function output = diamonds(scale)
    X = scale*[1,0,-1,0,1]
    Y = scale*[0,1,0,-1,0]
    plot(X, Y)
    axis( [-2,2,-1,1] )
    text(1,0,'(1,0)')
end
```

Saving this file as **diamonds.m**, we can run the function for different sizes from the command line, or another script. For example,

```
>> diamonds(5)
```

would create a diamond with corners at (5,0), (0,5), etc.

1.7 Some Basic Looping

Looping allow us to go through a list one element at a time and perform an action for each element in the list. The basic syntax is

```
for var = list
    Do Something
    Do Something Else
    Etc
end
```

Notice that we have dropped the >>. While loops work in the command window they are much easier to write up as scripts. From here on out we will assume script notation. For example, we could square all of the numbers from 1 to 10:

```
for i = 1:10
    i^2
end
```

For another example, we could plot $\sin(x+n)$ where $n = 0, 1, 2, \ldots, 20$ on the same plot using `hold on`:

```
X = 0:.01:30
hold on
for n = 0:20
    plot(X, sin(X + n))
end
```

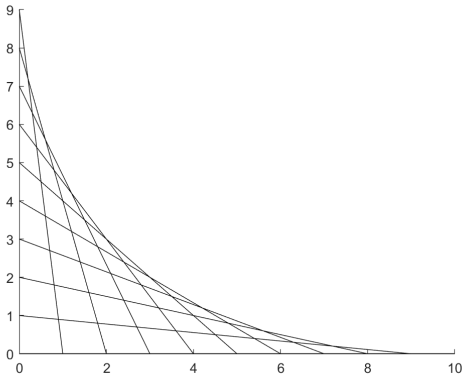
Finally, for our diamond above we can label the axes by accessing each of the four coordinates. MATLAB allows us to concatenate stings together using `+`, so `"Help" + "Me"` would yield `"HelpMe"`. For each of the coordinates `X(i), Y(i)`, we can construct the label by `"(" + X(i) + "," + Y(i) + ")"`. For example, if `X(i)` is 4 and `Y(i)` is 3, then this becomes `"(4,3)"`. The final code looks like

```
X = [1,0,-1,0,1]
Y = [0,1,0,-1,0]
plot(X, Y)
axis( [-2,2,-1,1] )

for i = 1:4
    label = "(" + X(i) + "," + Y(i) + ")"
    text(X(i),Y(i),label)
end
```

Question Modify your grid code to add a coordinate label to each point of the grid.

Question Use a for loop and `hold on` to draw the lines connecting $(0, n)$ to $(10 - n, 0)$ for $n = 0, 1, \ldots, 10$ as below.



1.8 Further Reading:

There are many other resources online if you would like to continue learning before Lab 1. A good companion guide can be found at http://cda.psych.uiuc.edu/matlab_pdf/math.pdf

The best way to learn MATLAB is to use it to solve interesting problems. Project Euler is the best source of mathematical computing exercises on the web and a great place to start honing your programming skills https://projecteuler.net/archives