2526-MA378 : MATLAB Primer.

This is a primer for labs in Numerical Analysis 2. Read through it to learn some fundamentals of working in MATLAB/Octave.

Please send any comments, corrections and suggestions to Niall Madden at Niall.Madden@UniversityOfGalway.ie. I'm grateful when you point out typos.

Table of Contents

What are MATLAB and Octave?

MATLAB is a language and environment for scientific computing. Originally it was for working with matrices: the name comes from "matrix laboratory". So it excells at working with matrices and vectors.

MATLAB is quite expensive, so some people have written a free-version called Octave. The functionality is similar, though the front-end is not as polished.

A MATLAB/Octave primer

You are using a MATLAB "Live script". This is a note-book type interface for MATLAB (a bit like Jupyter) that let's usa combine text and MATLAB code. You can run pieces of code one section at time (click on "Run Section") or run the entire notebook.

Everything is a vector

In Matlab, just about everything is a vector. In this lab, we are mainly interested in defining vectors the represent interpolation points. If, for example, we wanted to define a vector of points between 0 and 2, with gaps of 0.5, we could do in several ways.

The first is to just list all the points, using `[` and `]` to mark the start and end of the vector:

```
x = [0, .5, 1, 1.5, 2]
```

```
x = 1×5
         0    0.5000    1.0000    1.5000    2.0000
```

The second is the use the colon notation, which is `start:step:end`.

```
x = 0:0.5:2
```

```
x = 1×5
         0    0.5000    1.0000    1.5000    2.0000
```

Finally, there is a function called `linspace(),` which uses the syntax:

`linspace(start, end, number of points)`

```
x = linspace(0,2,5) % this one is more useful later
```

```
x = 1×5
         0    0.5000    1.0000    1.5000    2.0000
```

MATLAB indexes vectors from 1 (and not zero, like Python or C++). So the first, 2rd and 4th elements of the vector we just defined are

```
x(1), x(2), x(4)
```

```
ans =
0
ans =
0.5000
ans =
1.5000
```

This means we have to be careful. In lectures, we've denoted the interpolation points $\{x_0, x_1, \ldots, x_n\}$. But in MATLAB they are `[x(1), x(2), ..., x(n+1)]`.

Vector arithmetic

One of MATLAB's great strengths is that it is easy to compute new vectors from old ones. For example, we can set

```
y = cos(x)
```

```
y = 1×5
    1.0000    0.8776    0.5403    0.0707   -0.4161
```

This has the effect of setting:

- `y(1)=cos(x(1))`
- `y(2)=cos(x(2))`, etc

More generally, we can combine vectors using the standard standard arithmetic operators: +, -, * and /. For any vector, x, the expression

```
y = 2*x +3
```

```
y = 1×5
     3     4     5     6     7
```

sets `y(i) = 2*x(i) + 3`.

However, when we apply operators to pairs of vectors, we need to take a little care. First, of course, to add and subtract a pair of vectors, they need to have the same length.

Multiplication and devision is a little more tricky. For example, mathematically, the expression $z = x * y$ makes no sense, because you can't multiply two (row) vectors. If we want to make the operations "entrywise", we can use the "dot" operators: ".*" and "./" for multiplication and division.

That is `z = x.*y` sets `z(i) = x(i)*y(i)` for each i.

The dot operator also works for exponents:

```
y = x.^2   % sets y(i)=x(i)^2 for each i
```

```
y = 1×5
         0    0.2500    1.0000    2.2500    4.0000
```

You can also apply standard mathematics functions, such as `log()`, `exp()`, `sin()`, `cos()`, `abs()`, etc, to vectors, and they will return vectors.

Getting output (or not)

MATLAB is an interpretive language (like Python). You can run your code one line at a time. If a line ends with a semi-colon, then the result of the computation is not shown. If the vectors are very big, we usually end lines with a colon.

We can also check the value of the variable using the `disp()` function.

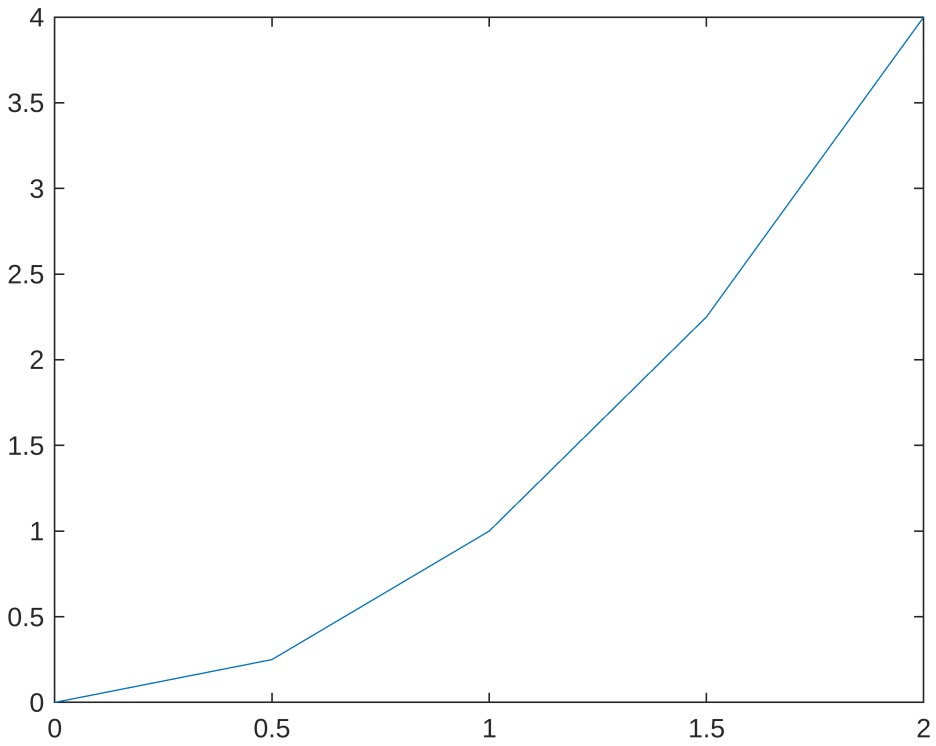
```
disp(x)
```

```
         0    0.5000    1.0000    1.5000    2.0000
```

Plotting vectors

We can plot sets of points as follows. Assmung the vectors x and y are already defined, and are of the same length:

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

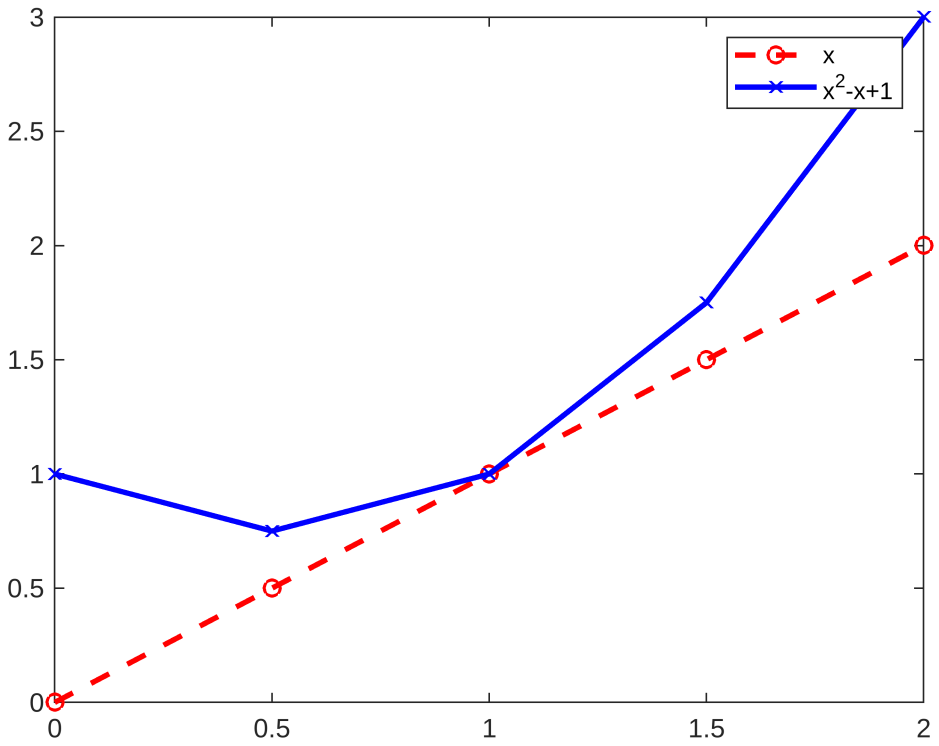


Often we want a little more control over:

- colour of the line
- line style (e.g., continuous, dashed)
- how points are displated
- line width.

See if you can work out why this line of code gives the output it does:

```
plot(x,x, 'r--o', x, x.^2-x+1, 'b-x', 'LineWidth',2)
legend('x', 'x^2-x+1')
```



Defining functions

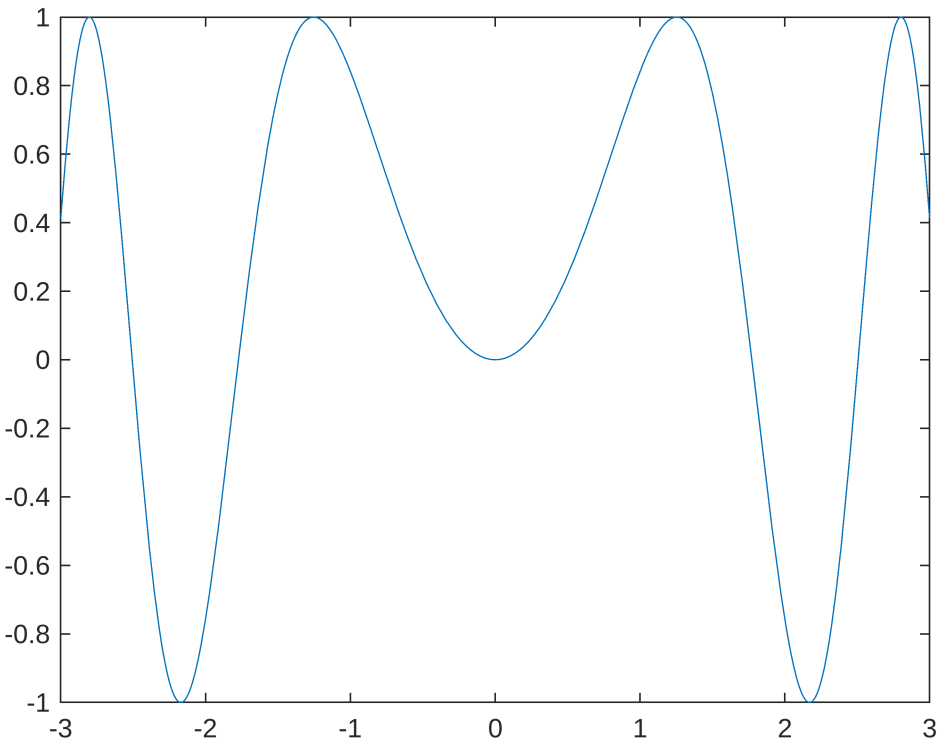
In computational mathematics, we often work with functions. Here is how to define a simple function (as a mapping from one real number to another):

```
f = @(x)sin(x.^2)
```

There are many operations that can then be performed on functions, such as computing definite integrals, or plotting. Here is how to plot.

Example:

```
f = @(x)sin(x.^2);
fplot(f,[-3,3])  % plots f(x) for -3 <= x <= 3
```



What Else?

Is there something that you think you need to be able to do in MATLAB in order to complete these labs, or investigate some aspect of MA378? If so, let Niall.Madden@UniversityOfGalway.ie know.