Anonymous functions and Intro to Matrices

1 Anonymous functions

We've learned how to use subfunctions to pass function handles to `ode45`. These require you to put all of your code inside a Matlab function file, which can sometimes be annoying if you're trying to get an answer quickly. That's why we have Anonymous Functions! Try this from the command line

```
>> f = @(t,y)-2*t.*y;
>> [t,y] = ode45(f,[0 2],1);
>> plot(t,y)
```

Hey, that was pretty easy! Note that you don't need the `@` symbol like you do with sub functions. Anonymous functions can be used in `.m` files containing scripts or functions as well.

Anonymous functions allow you to use variables stored in the workspace when defining a function. For example, if you want to define a function describing the parabola $f(x) = ax^2 + bx + c$, but don't want to pass in the values of a, b, and c each time, you can do something like this:

```
a = 1;
b = 2;
c = 3;
f = @(x) a*x.^2 + b*x + c;
```

Note that if you change the value of a, b, or c you must redefine f.

Subfunctions are better for complicated things that contain multiple lines or require multiple outputs. Anonymous functions are nice if your expression is easy, and only requires one output.

2 `fzero`

`fzero` is a numeric root finder, just like `FindRoot` in Mathematica. It has syntax similar to `ode45`. Matlab's documentation lists the syntax as `X = fzero(FUN,X0)`, where `FUN` is a function handle and `X0` is an initial guess. Like any root finder, `fzero` is not guaranteed to converge to an answer, and the likelihood of convergence depends on your initial guess. Ideally, you would have some intuition on what guess would be close, which can often be obtained by looking at a plot. You can use this function by either defining a separate function that determines the functions values like we initially did for `ode45` or by using an anonymous function.

To use a subfunction, first define the function in a separate file in MATLAB:

```
function [y] = func(x)
    y = 0.65*x - 0.01*x.^2 - 1.2*x.^3./(1+x.^3);
end
```

Then, call `fzero` in the command line as in the following:

```
p = fzero(@func,60);
```

Here's an example using an anonymous function:

```
f = @(y) 0.65*y - 0.01*y.^2 - 1.2*y.^3./(1 + y.^3);
fzero(f,60)
```

Running this code gives a result of about `63.0982`. If you plug that into f the result is very close to zero. Again, note that you don't need the `@` symbol when passing the anonymous function into `fzero`.

3 Vector and Matrix basics

You've already seen how to make vectors. You can define them manually as a list of numbers inside square brackets separated by commas or spaces, like `[1 2 3 4]` or `[0,1,2,3,34534534]`. You learned how to use colons to automatically build vectors with start, stop, and step size values, and using `linspace`. We also talked about using `zeros` to initialize vectors before we stored values in them. Ok, you're all vector experts...on to matrices!

3.1 Defining matrices

Matrix columns are separated by commas or spaces, just like vector elements, while row are separated by semicolons. here's an example: If I type in `A = [1 2 3;4 5 6;7 8 9]`, Matlab spits out:

```
A =
     1     2     3
     4     5     6
     7     8     9
```

You can see how the semicolons separate the rows of the matrix.

3.2 Matrix indexing

Accessing elements of a matrix is similar to a vector, except you have 2 indices. So if you typed in `A(i,j)` you would be asking Matlab for the i^{th} row and the j^{th} column. The convention is $(row, column)$, which is backwards from the (x, y) convention used for Cartesian coordinates. It's a bit confusing so just say "row column. row column. row column" over and over until you remember! Using my example matrix A, if I typed $A(2, 3)$ into the command line, Matlab would return 6, since it's in the 2^{nd} row and 3^{rd} column.

If you wanted to access the first column of A, you would do `A(:,1)`. Here, the colon means "all rows," and the "1" means "first column." You can also use the colon in the column place, so `A(3,:)` means "third row, all columns." You can use `end` here the same way we did with vectors, so `A(3,:)` would be accessible by `A(end,:)`, which in this case means "last row, all columns." These commands are a bit confusing, so you should play around with them until it makes sense.

3.3 Matrix multiplication

Matrix multiplication is a little harder than normal multiplication, and hopefully you learned about it in class! In Matlab, matrix multiplication is performed by using the * (shift 8) symbol. As an example, try defining a matrix of all 1s, like `B = ones(3,3);`. Now if I multiply the matrix A that was defined above by the new matrix B, I get a strange result:

```
>> A*B
ans =

     6     6     6
    15    15    15
    24    24    24
```

Basically, the ij^{th} entry in the answer is the dot product of the i^{th} row of A and the j^{th} column of B. This is why we've been using `.*` or `.^2` to evaluate functions. We didn't want matrix multiplication, we wanted element wise multiplication. If you do A "dot times" B, the answer is quite different:

```
>> A.*B
ans =

     1     2     3
     4     5     6
     7     8     9
```

3.4 Matrix indexing using for loops

Sometimes it's useful to populate a matrix with values using for loops. Let's say you want to build a matrix whose elements were the sum of the row and column index, I could write the following function:

```
function A = buildMatrix(n)

A = zeros(n,n);

for i = 1:n
    for j = 1:n
        A(i,j) = i + j;
    end
end
```

Note how I initialized A by doing `zeros(n,n)`. This is important for speed.

4 Homework

1. Using anonymous functions, write a script that finds all the roots of

$$f(x) = (x-1)^2(x+1)^2 - e^{-x^2} + \frac{1}{4}$$

You should find 4, and it's very easy to check to see if they're actually roots.

2. This is the 5×5 Hilbert matrix.

$$\begin{bmatrix} 1 & 1/2 & 1/3 & 1/4 & 1/5 \\ 1/2 & 1/3 & 1/4 & 1/5 & 1/6 \\ 1/3 & 1/4 & 1/5 & 1/6 & 1/7 \\ 1/4 & 1/5 & 1/6 & 1/7 & 1/8 \\ 1/5 & 1/6 & 1/7 & 1/8 & 1/9 \end{bmatrix}$$

Using the pattern, write a function, `myHilb.m`, to construct an $n \times n$ Hilbert matrix.