PHYS 361: Loops

for loops

Sometimes you want to repeat a set of operations over and over again. The next few lines of code give an example.

```
format short

for k=10:-.1:1
    k
end
```

That is a pretty boring loop, but note what it is doing. Every time it runs through the loop, the variable k is replaced with a new value defined by the array 1:10. The for and end (which are blue because they are keywords) tell MATLAB where the looped code begins and ends. Here is another example,

```
format short

for angle=0:pi/2:2*pi
    anscos=cos(angle)
    anssin=sin(angle)
end
```

In the example below, I am summing the first 20 terms of the Taylor series for $sin(35^{\circ})$.

$$sin(x) = \sum_{k=0}^{\infty} \frac{(-1)^k x^{2k+1}}{(2k+1)!}$$

```
clear;              %I'm reusing variable names so clear to be safe

format long;

n=20;               %Specify number of terms

xrad=35.*pi/180;    %Convert degrees to rad

sin(xrad)           %Get real value for comparison

S=0;    %Since I am summing by adding values to S, I have to specify an initial value
for k=0:n-1
    k
    (-1)^k*xrad^(2*k+1)/factorial(2*k+1)
    S=S+(-1)^k*xrad^(2*k+1)/factorial(2*k+1)    %Could have used the shorthand S=+ for S=X+
end
```

Note that after the first few terms, you can't tell that the sum is still an approximate value. That is because MATLAB does not show you very many digits by default. To see more, you can specify the format. The default

format is short, which shows you four decimal digits or values in scientific notation. The long format will show you 15 digits.

```
clear;
doc format     %You should take a look at the documentation
format long    %Tell MATLAB you want to print variables in the long format

S=0;
n=20;

xrad=35.*pi/180;
sin(xrad)

for k=0:n-1
    S=S+(-1)^k*xrad^(2*k+1)/factorial(2*k+1);
end
```

While loops

Sometimes you don't know how many times you want to loop through a few lines of code. You might want to loop until a condition in your code is true or false. For example, run the following code

```
clear all;

n=1;

while n~=10
    n=n+1

end
```

Notice that the loop stopped when the condition became false because n was equal to 10. These types of loops are useful but also dangerous. They are dangerous because if you make a mistake, your condition might never be true, and your loop will continue indefinitely. So be careful!

Hault running code and a protective measure for infinite loops

If your code enters an infinite loop (or a while loop that runs indefinitely) because the exit condition is never met, you can (sometimes) stop MATLAB from running by typing cntrl-c or cntrl-z. In the script editor, you can also (sometimes) hit the "Pause" button under the editor tab and then "Stop." I say "sometimes," because sometimes you need something more powerful like force quitting MATLAB.

Here is another way to safeguard your code from infinite loops:

```
clear all;

n=1;
s=1;

while s >= 1 & n < 100
    s = s*2   %Intended code or the calculation you want to make
```

```
    n=n+1    %Counter to prevent the loop from running more than 100 times
end
```

The code would have run indefinitely, but I limited the number of times the loop can run to 100, which is a value that is larger than the number of times I expect the non-broken loop to run.

Break Statements

Break statements are another way to exit a for or while loop early. Previously, I showed you one way to safeguard a while loop from running indefinitely. Here is another:

```
n=1;
s=1;

while s >= 1 & (n<100000)
    s = s + 1;
    n=n+1;

    if n > 5000
        fprintf('Exiting loop, too many iterations')
        n
        break;
    end
end
```

Here if the condition of the if statement is met, the break command tells MATLAB to exit the loop (go to the next end) and move on. If the break command is in a nested loop (which you will learn about next), the code will move out of the inner loop and into the surrounding outer loop.

Note: I put a message in there if the break condition is met. Putting statements like that in your code is a useful way to debug.

The Continue Statement

The continue statement is similar to the break statement, but instead of exiting the loop the continue statment simply tells the code to move on to the next iteration of the loop without executing the code in the loop below the continue statement. This can be useful for skipping some calculations if a specific condition is met. For example, here the code outputs all the values except for 15 because it was told to skip to the next iteration of the loop in that case:

```
a = 9;
%while loop execution
while a < 20
   a = a + 1;
   if a == 15
      % skip the iteration
      continue;
   end
```

```
fprintf('value of a: %d\n', a);

end
```

Nested Loops

Here is a simple example of a nested loop.

```
n=1;

for k=1:4
    for j=1:3
        for i=1:2
            fprintf('\n i=%i ,j=%i ,k=%i, n=%i',i,j,k,n)
            n=n+1;
        end
    end
end
```

Nested loops are often required to perform calculations on multidimensional data. For example, lets say I wanted to calculate the strength of the electric field from a positive charge placed at the origin in 3-dimensions cube. The magnitude of the field at a radial distance r is:

$$E = \frac{1}{4\pi\epsilon_0}\frac{q}{r^2}$$

If I wanted to calculate the strength of the field in a 2 cm x 2 cm x 2 cm cube surrounding the charge I could code it up like this:

(**Note**: If you don't understand the code below, uncomment the fprinf to observe the output. Notice how the nested loop iterates over the coordinates of a cube centered at the origin. Draw a picture and compare it.)

```
clear all;  %Clear my workspace so I start from scratch

epsilon=8.8e-12;  %Permitivity of free space (units C^2/N/m^2)
q=1e-6;           %Charge (units C)


%Define an empty array for the electric field, units (V/m or N/C)
E=zeros(11,11,11);


%loops over each element of the electric field matrix
for l=1:11
    for m=1:11
        for n=1:11
```

```
            %use the array indeces to find the x, y, z value
            %assuming (0,0,0) is at the center of the cube, in meters
            x=(n-6)*.002;
            y=(m-6)*.002;
            z=(l-6)*.002;



            %Calculate the radial distance
            r=sqrt(x^2+y^2+z^2);

            %Find the electric field and store it in cube
            E(n,m,l)=1/(4*pi*epsilon)*q/r^2;

            %Uncomment out this line to observe the output
            %fprintf('E is %12.3f V/m at (x,y,z)=(%4.3f,%4.3f,%4.3f) m \n',E(n,m,l),x,y,z)


        end
    end
end

%Make a simple plot to illustrate that we have
%build a cube of electric field values
[X,Y,Z] = meshgrid(-1:.2:1);


xslice = [-.8,.1,.8];
yslice = [];
zslice = .1;
slice(X,Y,Z,E,xslice,yslice,zslice)
colorbar;
```