Tutorial 5: Cells & Structs

Getting Started

1. This tutorial is designed to emphasize some advantages of cells and structs, and, in the spirit of nostalgia, will try to clarify some issues surrounding Fourier analysis.

2. Download the versions of myfft.m and FourierMat.m on the assignments web page. Then download myffSt.m, FourierMatSt.m, and SpectraSt.m from the Tutorial 5 page. Make sure these files are on your Matlab path.

3. Start Matlab.

Storing Data in Cells

4. Our goal is to play around with Fourier series. First, let's specify the variables for a Fourier series and then evaluate it. Let's have 5 frequencies with integer values, so type

   ```
   >>f=0:4;
   ```

   Now, let's create the coefficient vectors a and b. These must be the same size as f (length 5), but they should be column vectors. For simplicity, make a(1) and b(1) =0 and the rest of the coefficients equal to 1. Create a and b—make sure they are column vectors.

5. Now, use FourierMat to evaluate this function for a range of times. Create a vector of 100 evenly spaced times from 0 to 1. The command linspace does this:

   ```
   >>t=linspace(0,10,100)';
   ```

   (Don't forget the " ' " to make t a column vector. The, pass t (and a, b, f, and t0) to FourierMat to evaluate the function at these times:

   ```
   >>x=FourierMat(a, b, f, t, 0);
   ```

6. Plot x as a function of t.

   ```
   >>plot(t,x);
   ```

   You should see an oscillatory function with spikes approximately every 1 units of time.

7. We will now do a little experiment to see how the length of the series input to myfft affects the function we can represent. Our strategy will be to

create 5 input series by taking different sized sections of x. We will then pass these to myfft and look at the spectra and the ability of the new functions to represent x. Ideally, we would save the 5 input series as columns in an array, and then conduct each step by looping over the columns. However, since the input series will each be a different size, they will be difficult to store in an array. Cells to the rescue!

8. Create a vector l holding the lengths of the input series, say l=[100, 50, 24, 12, 6]. Then, write a loop that grabs x and t from 1:l(j) and saves them in a cell array:

```
>> for j=1:5; dat{j}=[t(1:l(j)),x(1:l(j))];end
```

   This will create a cell array dat with 5 cells. Type "dat" and hit return to see a summary of the contents of dat. Do the sizes of each cell agree with l?

9. Now, we will compute the Fourier coefficients for each cell. Using myfft, this will involve creating five arrays corresponding to the a's, five for the b's, etc. We need a way to manage these arrays. Cell arrays could work, but I think structs are more appropriate here. What we want is to define a structure containing all of the information returned by myfft, let's call it an "FFT-struct." This allows us keep the information for each data set together—a feature known as "encapsulation." The function myfftSt.m takes the same inputs as myfft, but returns a struct rather than 3 arrays. Open myfftSt.m in the editor and look it over.

10. Another advantage of structs is that we can create arrays of them. In our case, we'll create a length 5 vector of FFT-structs corresponding to the coefficients for each fit. We can do this in a loop:

```
>>for j=1:5;FFTstr(j)=myfftSt(dat{j}(:,1),dat{j}(:,2));end
```

11. I've also provided a version of FourierMat called FourierMatSt that accepts an FFT-struct and a vector of times, and returns the value of the function evaluated at that time. We will use FourierMatSt to evaluate each of the five approximations at the original times (t). This will allow us to evaluate the information lost as we consider shorter data sets. To save each new series in a cell, run the following loop:

```
>> for j=1:5;newx{j}=FourierMatSt(FFTstr(j),t);end
```

    Note: newx could be a matrix with the data stored in columns.

12. To see the differences between the data, let's plot each on top of the full approximation (newx{1}). The following loop plots newx{1} in blue and plots newx{j} on top in red. After plotting, the command pause is called.

Pause without any arguments, waits for the user (i.e. you) to press any key:

```
>> for j=2:5;clf;plot(t,newx{1},'b',t,newx{j},'r');pause;end
```

How do they look. Notice the two series agree until t(I(j)), and then the agreement is rapidly lost.

13. Finally, let's look at the spectra. I've provided a function spectraSt that plots the spectrum defined by an FFT-struct. This function will accept an optional color argument, and we can use this feature to plot each spectrum with a unique color. First, define a character array with five colors:

```
>>col='rgbcm';
```

Now, loop over each element in FFTstr and call spectraSt.

```
>> for j=1:5;spectraSt(FFTstr(j),col(j));hold on;end
```

How do these spectra compare with the original? To plot the original data as black stars:

```
>>plot(f, sqrt(a.^2+b.^2),'k*');
```

14. You are now set up to learn more about how Fourier series and the FFT work. You can try repeating the experiment starting from a simpler (or more complicated) x. You could also look at how the resolution of the data affects how well you do. Have fun!