Mathematics 174 – Applied Mathematics 2
Lab Day 5 – The Fast Fourier Transform (FFT)
April 18, 2000

Background and Goals

Last time we introduced the *N-point discrete Fourier transform* (DFT). For discrete time-domain signals (sequences) f_j, $j = 0, \ldots, N-1$, the DFT produces a frequency-domain sequence F_k defined by

$$F_k = \frac{1}{2\pi} \sum_{j=0}^{N-1} f_j e^{2\pi i (jk)/N}. \tag{1}$$

If f_j are sampled values of an analog signal $f(x)$ defined for $x \in [0, \Omega]$: $f_j = f\left(\frac{j\Omega}{N}\right)$, then we can view the term F_k in the DFT sequence as yielding an approximation to a value of the Fourier transform of f according to the following formula:

$$F_k \doteq \frac{N}{\Omega} \cdot \mathcal{F}[f]\left(\frac{2\pi k}{\Omega}\right).$$

(This comes from (1) by rearranging that sum to look like an appropriate Riemann sum approximation to the FT integral.)

In 1965, two mathematicians named J.W. Cooley and J.W. Tukey published an extremely clever way and fast way to set up the computations for the DFT for the case where N is a power of 2: $N = 2^m$. This allows a *recursive computation of the N-point DFT using $N/2$-point DFTs*, in a scheme reminiscent of the binary search or the merge sort algorithms. The advantage is that this method takes $O(N \log_2(N))$ operations to compute the DFT, while the direct method using (1) uses $O(N^2)$ operations. Try comparing $N \log_2(N)$ and N^2 if $N = 10^6$ or larger!! It is no exaggeration to say that Cooley and Tukey's "Fast Fourier Transform" is the engine that has driven the development of digital signal processing for the past 30 years. Today, we will use Maple's implementation of the FFT to get a first taste of some of the ideas.

Maple's FFT Package – A Worked Example

Maple has a decent (but not "industrial-strength") implementation of the FFT algorithm in a library package. *Important Note:* Maple's FFT algorithm actually computes:

$$F_k = \sum_{j=0}^{N-1} f_j e^{2\pi i (jk)/N} \doteq \frac{2\pi N}{\Omega} \cdot \mathcal{F}[f]\left(\frac{2\pi k}{\Omega}\right) \tag{2}$$

Our $\frac{1}{2\pi}$ factor in (1) *is not included in Maple's value.* To reconcile Maple's results with what you expect from our formulas, you will need to adjust by a factor of 2π as in (2).

To use it, you will need to start by entering

```
readlib(FFT);
```

This package contains two main functions called `FFT` (the Fast Fourier Transform), and `iFFT` (the Fast Inverse Transform). You use them like this:

- Each of these commands takes *three arguments*. The first is the *exponent* m in the equation $N = 2^m$. The second and third are two 1-dimensional arrays giving the *real and imaginary parts* of the input (time-domain) sequence f_j.
- The format of the `FFT` command is:

```
FFT(m,Rarray,Iarray);
```

- When you execute this, the output will be the integer $N = 2^m$, but *the real and imaginary parts of the transform sequence F_k will be put into the two array variables you supplied in the call to the FFT function.* Thus, you access the transform by referring to `Rarray, Iarray` again.
- In the process, the values of the real and imaginary parts of the input sequence *are lost.* If they are not easy to generate and you want to use them again (and you don't want to have to keep using the inverse transform to recover the input), be sure to store them separately!
- The inverse transform `iFFT` works exactly the same way.

For example, suppose we take $\Omega = 1024 = 2^{10}$, and $N = 1024$, so the sampling interval is $\Delta x = 1$. Consider the signal with sampled values

$$f_j = \begin{cases} 1 & \text{if } j \in [0, 50] \\ 0 & \text{if } j \in [51, 1024], \end{cases}$$

To define the corresponding lists for the input to `FFT`, the easiest way is to use Maple's `seq` function as follows (see the online help pages for `seq`, `array` for more information):

```
recR := array([seq(1,j=0..50),seq(0,j=51..1023)]):
          recI := array([seq(0,j=0..1023)]):
```

(The array of imaginary parts contains all zeros, since f_j is real for all j. Also, you *do not* want to see the output – see why?!) Now we are ready to compute the DFT:

```
FFT(10,recR,recI);
```

As explained above, after executing this command, the two arrays will contain the real and imaginary parts of the DFT sequence F_k. To visualize these, we can construct plots as follows. Recall by (2) above that if we think of f_j as sampled values of an analog signal $f(x)$ for $x \in [0, \Omega]$, then F_k is an approximation to $\frac{N}{\Omega} \cdot \mathcal{F}[f]\left(\frac{2\pi k}{\Omega}\right)$, for k from 0 to $N - 1$. So we can construct the *lists of points*

$$\left(\frac{2\pi k}{\Omega}, \mathrm{Re}(F_k)\right), \qquad \left(\frac{2\pi k}{\Omega}, \mathrm{Im}(F_k)\right)$$

and have Maple plot them by "connecting the dots":

```
REpoints:=[seq([2*Pi*k/1024,recR[k]],k=1..1024)]:
IMpoints:=[seq([2*Pi*k/1024,recI[k]],k=1..1024)]:
```

(N.B. the indices in the arrays are 1..1024, not 0..1023.) Then try the following commands:

```
plot(REpoints);
plot(IMpoints);
```

Even more informative than these plots is often the plot of what is called the *frequency power spectrum* of the original signal. By definition, this is the absolute value of the Fourier Transform. You can plot the discrete version by:

```
FPSpoints:=[seq([2*Pi*k/1024,sqrt(recR[k]^2+recI[k]^2)],k=1..1024)]:
plot(FPSpoints);
```

Finally, apply the inverse transform to these sequences, and plot to see that the original sequence is recovered.

Lab Questions

A) Explain the shape of the plot of the frequency power spectrum in the example above by computing the absolute value of the "analog" Fourier transform of the function

$$f(x) = \begin{cases} 1 & \text{if } x \in [0, 50] \\ 0 & \text{otherwise} \end{cases},$$

plotting that on $[-\pi, \pi]$, and comparing what you get with the results from FFT. Don't forget, $\Omega = N = 1024$ here, and the DFT is always periodic: $F_{k+N} = F_k$ for all k.

B) Construct discrete signals by sampling

$$f(x) = \cos(30x) \text{ and } 25\cos(194x)$$

for $x \in [0, \Omega] = [0, 2\pi]$. Use $N = 1024$. Compute the DFT of each and explain the results. In particular:

- Where are the centers of the "peaks"?
- How high are they?
- And most importantly, why do the DFT's look that way? What should the analog FT of each of these functions look like? Be careful: We are *not really* considering signals $f(x) = a\cos(bx)$ here, but rather "windowed signals" $f(x) = a\cos(bx)W(x)$ where the window function is

$$W(x) = \begin{cases} 1 & \text{if } x \in [0, \Omega] = [0, 2\pi] \\ 0 & \text{otherwise} \end{cases}$$

Note: These computations will take longer than those in the example above if you don't optimize for the way Maple treats input data. Put all calls to the cosine *function inside* `evalf`*'s when you compute the real part of the original signal so that will be stored as single floating point numbers. If you don't do that, Maple will treat each value of the transform as an expression – a big hairy sum with 1024 terms. That will slow things down* **a lot***!*

C) One of the most basic and important signal processing tasks is the *analysis* of signals – in particular, decomposing signals into their pure-frequency components to understand how they are made up. For instance, many of the instruments used by chemists and physicists (like FT-IR spectroscopes, etc.) measure some signal, then take the Fourier transform to display output showing the different frequency components present. In my directory for our class is a Maple output file containing a "mystery signal." You can read it into your Maple session using the following command:

```
read "/home/fac/little/public_html/Applied9900/mystery.m";
```

That file contains definitions for two array variables called `MystR, MystI` containing the real and imaginary parts for the mystery signal for $x \in [0, 2\pi]$, sampled with $N = 1024$. Using the FFT, determine which frequencies are especially strongly present, and give a possible formula for an analog signal from which `MystR, MystI` might have been sampled. Note though: there is some "noise" in this signal so peaks are not as sharp (or as high) as you might expect. Also, be sure you look at the real and imaginary parts of the DFT sequence separately, not just the frequency power spectrum – there is important info there.

D) Design and use Maple to implement a discrete (ideal, brick-wall) "band-pass" filter for input signals with $x \in [0, 2\pi]$, using the 1024-point FFT. The pass-band should be the frequency range $60 \leq |\omega| \leq 120$, and all other frequencies should get filtered out. Test it on the mystery signal from question C. (Your result should be close to a pure cosine waveform, with some "noise".)

E) The FFT can also be used to produce a numerical method for approximating the convolution of two functions that is far superior to what you get by approximating the convolution integral directly by a numerical integration method. Use the 1024-point DFT on the interval $[0, 10]$ to approximate the convolution of the functions $f(x) = \exp(-3(x-2)^2)$ and $g(x) = \exp(-2(x-3)^2)$. Compare with the result you get applying the formula from question II on Problem Set 5.

Assignment

Individual write-ups due Thursday, April 27.