Chapter 13

Fourier Analysis

13.1 Introduction

The Fourier transform is an important technique that analyzes information about signals. The Fourier transform breaks down complex waveforms into simple components- that is, as a simple sum of sines and cosines.

In terms of linear algebra, if we let V be the vector space of continuous functions on the interval $[0, 2\pi]$, (or, $C[0, 2\pi]$), then the following forms an orthogonal basis for V:

$$\{1, \sin(kx), \cos(mx) \mid k, m = 1, 2, 3, \ldots\}$$

which means that any function in $C[0, 2\pi]$ can be written as a linear combination:

$$f(x) = a_0 + \sum_{k=1}^{\infty} a_k \sin(kx) + \sum_{k=1}^{\infty} b_k \cos(kx) \tag{13.1}$$

The proof that this is indeed a spanning set can be found in most textbooks on partial differential equations. Let us explore our new vector space with the following remarks:

1. $C[0, 2\pi]$ is an infinite dimensional vector space.
2. The inner product between two functions f, g is defined:

$$< f, g >= \int_0^{2\pi} f(x)g(x)\,dx$$

3. The norm (or size) of f can be defined in terms of the inner product:

$$\|f\|^2 =< f, f >$$

4. The distance between functions f and g is defined as $\|f - g\|$.

5. Two functions are said to be orthogonal (or perpendicular) iff $< f, g >= 0$, as usual.

6. Since $< f, g >= \|f\| \, \|g\| \cos(\theta)$, we can also define the *angle* between two function.

7. Since the basis functions are orthogonal, its "easy" to compute the coefficients a_0, a_k, b_k: We simply project f to each coordinate function:

$$\begin{aligned} a_0 &=< f, 1 >= \frac{1}{2\pi} \int_0^{2\pi} f(x)\, dx \\ a_k &=< f, \sin(kx) >= \frac{1}{\pi} \int_0^{2\pi} f(x) \sin(kx)\, dx \\ b_k &=< f, \cos(kx) >= \frac{1}{\pi} \int_0^{2\pi} f(x) \cos(kx)\, dx \end{aligned}$$

13.1.1 From Real to Complex

In the last section, we saw that we need to compute lots of integrals involving sines and cosines. It's easier to go into the complex plane, where we can compute both simultaneously.

To do this, we will use Euler's Formula:

$$\mathrm{e}^{i\theta} = \cos(\theta) + i \sin(\theta)$$

This formula can be proved using the Taylor series expansion for e^x, but will be omitted here.

Furthermore, differentiation and integration work the same as in the real case (again, the proof will be omitted here):

$$\begin{aligned} \frac{d}{d\theta} \mathrm{e}^{i\theta} &= i\mathrm{e}^{i\theta} = -\sin(\theta) + i\cos(\theta) \\ \int \mathrm{e}^{i\theta}\, d\theta &= \frac{1}{i} \mathrm{e}^{i\theta} = \sin(\theta) - i\cos(\theta) \end{aligned}$$

Example: Compute $< x^2, \sin(2x) >$ and $< x^2, \cos(2x) >$ using the complex exponential.

Because

$$\int_0^{2\pi} x^2 \mathrm{e}^{2ix}\, dx = \int_0^{2\pi} x^2 \cos(2x)\, dx + i \int_0^{2\pi} x^2 \sin(2x)\, dx$$

we will compute $\int x^2 \mathrm{e}^{2ix}\, dx$. The real part of the answer will be $< x^2, \cos(2x) >$, and the imaginary part will be $< x^2, \sin(2x) >$. Use integration by parts[1] to obtain:

$$\int_0^{2\pi} x^2 \mathrm{e}^{2ix}\, dx = \mathrm{e}^{2ix} \left(\frac{1}{2i} x^2 + \frac{1}{2} x - \frac{1}{4i} \right) \Bigg|_0^{2\pi} = \pi - 2\pi^2 i$$

[1] See the Appendix for a shortcut method.

From which we conclude:

$$\int_0^{2\pi} x^2 \cos(2x)\, dx = \pi, \quad \int_0^{2\pi} x^2 \sin(2x)\, dx = -2\pi^2$$

This is an example worth remembering! This will end our discussion of the continuous case. After the exercises, we will focus our attention on Matlab's implementation of the Fourier Transform.

13.1.2 Exercises:

1. Verify that the basis is indeed orthogonal using the inner product. You might want to look up the following integrals in any calculus book:
$$\int \sin(kx)\cos(mx)\, dx, \int \sin(kx)\sin(mx)\, dx, \int \cos(kx)\cos(mx)\, dx$$

2. Compute $< x, \sin(3x) >, < x, \cos(3x) >$ using the complex exponential.

3. What is the distance between x and x^2 using our inner product?

4. What is the angle between x and x^2 using our inner product?

5. Show that the following are identities (e.g., show that the right hand sides simplify to the left hand sides of the equations).
$$\sin(x) = \frac{e^{ix} - e^{-ix}}{2i} \qquad \cos(x) = \frac{e^{ix} + e^{-ix}}{2}$$
[REMARK: By changing x to the complex variable z, this is how $\sin(z)$ and $\cos(z)$ are defined in the complex plane! Also note the similarity to the *hyperbolic* sine and cosine, which can be found in most calculus textbooks]

6. We see that
$$|e^{i\theta}| = \sqrt{\cos^2(\theta) + \sin^2(\theta)} = 1$$
so that $e^{i\theta}$ is on the unit circle. Plot some various values of $e^{i\theta}$ on the unit circle. For each of these, take note of where $e^{-i\theta}$ is.

7. Each complex number $a + bi$ can be written in its *polar form*, $re^{i\theta}$. To do this, first recall that the complex number $a + bi$ can be associated with the point (a, b) in the plane. Draw a line segment from the origin to (a, b). Then the length of this line segment is $r = |a + bi| = \sqrt{a^2 + b^2}$, and θ is the angle that the line segment makes with $(1, 0)$ (as usual). Write the following complex numbers in polar form:
   (a) i
   (b) $\frac{1}{2} - \frac{\sqrt{3}}{2}i$
   (c) $-2 + 2i$

13.2 Implementation of the Fourier Transform

To go from the continuous case to the computer implementation, we will need to discretize our function. That is, we will (by necessity) go from an infinite dimensional vector space to a finite dimensional approximation. Before discussing the implications to the Fourier analysis, let us discuss the discretization process, first by means of some Matlab exercises:

13.2.1 Exercises:

1. Consider the functions $y_1 = \cos\left(\frac{\pi}{4}t\right)$, $y_2 = \cos\left(\frac{7\pi}{4}t\right)$ with $0 \le t \le 64$. Will 64 samples be enough to differentiate between these functions? Try the following Matlab script, and write up your observations:

```
k=0:63; t=0:0.1:63;
y1=cos((pi/4)*k); y2=cos((7*pi/4)*k);
Y1=cos((pi/4)*t); Y2=cos((7*pi/4)*t);
plot(t,Y1,'b-',t,Y2,'r-');
hold on
plot(k,y1,'bo',k,y2,'r*');
hold off
%Take a closer look at the first 10 values:
axis([0 10 -1 1]);
```

2. Do the same exercise[2], if $y_1 = \sin\left(\frac{7\pi}{8}\right)$, $y_2 = -\sin\left(\frac{9\pi}{8}\right)$.

3. Recall the last exercise in Section 13.1.2. What is the relationship between $e^{i\frac{7\pi}{8}}$ and $e^{i\frac{9\pi}{8}}$? Between $e^{i\frac{\pi}{4}}$ and $e^{i\frac{7\pi}{4}}$?

4. Using the previous exercise, come up with other pairs of sines and cosines that will discretize to the same function (under the given discretization).

5. Give a conjecture about what we should allow as the smallest admissable period if the distance between two points is Δx. For example, is it adviseable to have a period that is $\frac{1}{2}\Delta x$? Why or why not? Draw a picture to justify your conclusions.

13.2.2 Matlab and the FFT

It is convenient to discretize a function using N points, where N is a power of 2. In this case, there is a fast way to compute the Fourier Transform, and this is built into Matlab as the *Fast Fourier Transform*, or FFT. We won't go into the numerical details here, but if N is a power of 2, your computations will go faster.

In Matlab, the command is easy:

[2]Try the Sampling Rate script file at the Appendix for more!

```
F=fft(f)
```

Let f be length N input vector. Then Matlab makes the following computation:

$$F_k = \sum_{n=1}^{N} f_n \mathrm{e}^{-\frac{2\pi i}{N}\cdot(k-1)(n-1)},\ 1 \le k \le N \tag{13.2}$$

Before investigating this closely, note first that this is an inner product: If x and y are vectors in C^n, then we define the inner product as:

$$< x, y >= x^T \bar{y}$$

where $\bar{y}$ is the complex conjugate of y. Thus, in the definition above, we are using negative values of the exponential.

If we think of the exponent in an $N \times N$ matrix, C, then the element $C_{k,j}$ corresponds to the j^{th} element of the dot product forming F_k, which gives:

$$C_{k,j} = \exp\left(\frac{2\pi i}{N}\cdot(k-1)(j-1)\right)$$

where we define $\exp(x) = \mathrm{e}^x$

To clarify the elements of the matrix, let's write[3] it by factoring out the $\frac{2\pi i}{N}$:

$$C = \exp\left(\frac{2\pi i}{N}\begin{bmatrix} 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 2 & 3 & & N-1 \\ 0 & 2 & 4 & 6 & & 2(N-1) \\ 0 & 3 & 6 & 9 & & 3(N-1) \\ \vdots & & & & & \vdots \\ 0 & N-1 & 2(N-1) & 3(N-1) & \dots & (N-1)(N-1) \end{bmatrix}\right)$$

which in Matlab is written as:

```
C=exp((2*pi*i/N)(0:(N-1))'*(0:(N-1)));
```

In summary if f is a column vector (size $N \times 1$), then we could compute the Fourier transform of f directly (note again the negative exponent):

```
F=exp( (-2*pi*i/N)*(0:(N-1))'*(0:(N-1)) )*f;
```

The inverse transform (`f=ifft(F)`) is computed in a similar manner:

$$f_n = \frac{1}{N}\sum_{k=1}^{N} F_k \mathrm{e}^{\frac{2\pi i}{N}\cdot(k-1)(n-1)},\ 1 \le n \le N$$

or

```
f=(1/N)*exp( (2*pi*i/N)*(0:(N-1))'*(0:(N-1)) )*F;
```

[3]We are taking some liberties with notation- this is not the matrix exponential, e^A, rather, we are exponentiating componentwise.

13.2.3 Exercises

1. Let f be a random set 16 numbers, `f=rand(16,1)`. Compare the F that is computed via Matlab's `F1=fft(f);` versus using

   ```
   F2=exp( (-2*pi*i/16)*(0:15)'*(0:15) )*f;
   ```

   You might check the comparisons by looking at the maximum size of the differences: `max(abs(F1-F2))`

   Also check the inverses: Compute `f1=ifft(F1)` and compare it against both the original f and

   ```
   f2=(1/N)*exp( (2*pi*i/16)*(0:15)'*(0:15) )*F2;
   ```

2. The previous exercise showed that the FFT can be written as an invertible linear operation,
$$F = Cf$$
   where C is as defined previously. Using the same code, check that C is invertible, and that:

   ```
   exp( (2*pi*i/16)*(0:15)'*(0:15) )* exp( (-2*pi*i/16)*(0:15)'*(0:15) )
   ```

   is 16 times the identity.

3. Since C is a complex matrix, define $C^* = \bar{C^T}$. The previous computation shows that $CC^* = C^*C = (N)I$. Recall that in linear algebra, we had a special name for a (real) matrix U where $UU^T = U^TU = I$. What was it? What could we conclude about it's columns? This is the extension to complex matrices.

13.2.4 Discretization Produces Orthogonal Vectors

Now that we've had some practice, we'll show that the discretization process we previously defined does indeed produce orthogonal vectors in C^n. The easiest way to show this will be to first define **the p roots of unity**:

Definition: The p roots of unity are the p solutions to $z^p = 1$. They are computed by taking $z = \mathrm{e}^{i\theta}$, so that:

$$\mathrm{e}^{i(p\theta)} = 1$$

From this, we get that $p\theta = 2\pi k$ (so that $p\theta$ is some constant multiple of 2π), and:

$$\theta = \frac{2\pi}{p}k, \quad k = 0, 1, 2, \ldots, p-1$$

Graphically, these are the angles equally spaced around the unit circle, always starting at $(1, 0)$. In Figure XX, we show the 3 roots of unity with $\theta = 0, \frac{2\pi}{3}, \frac{4\pi}{3}$, and the 5 roots of unity, with $\theta = 0, \frac{2\pi}{5}, \frac{4\pi}{5}, \frac{6\pi}{5}, \frac{8\pi}{5}$. The corresponding z values are computed by taking $e^{i\theta}$- we're just showing the angles.

Now, recall how we're discretizing the interval from 0 to 2π using N subintervals. The left endpoints of these intervals are:

$$0, \frac{2\pi}{N}, \frac{4\pi}{N}, \ldots, \frac{2\pi}{N} \cdot k, \ldots, \frac{2\pi}{N}(N-1)$$

These are exactly the N roots of unity! Now look back at the matrix C we computed in the previous section. If we define $z = e^{\frac{2\pi i}{N}}$, verify that we can write the $(j+1)^{\text{st}}$ column of C as:

$$\mathbf{v}^{(j)} = (z^0, z^j, z^{2j}, \ldots, z^{(N-1)j})^T$$

where $j = 0, 1, \ldots, N-1$.

We can now show that the vectors $\{\mathbf{v}^{(j)}, j = 0, 1, \ldots, N-1\}$ are orthogonal (this proof follows that found in Kirby). By taking the complex inner product of $\mathbf{v}^{(j)}$ with $\mathbf{v}^{(k)}$, we get:

$$S = < \mathbf{v}^{(j)}, \mathbf{v}^{(k)} >$$

$$S = \sum_{t=1}^{N} z^{tj} \bar{z^{tk}} = \sum_{t=1}^{N} z^{tj} z^{-tk} = \sum_{t=1}^{N} z^{t(j-k)}$$

which can be reformulated (set $p = j - k$, which is a constant with respect to the sum):

$$S = \sum_{t=0}^{N-1} z^{tp} = \sum_{t=0}^{N-1} (z^p)$$

This is a finite geometric sum:

$$\begin{aligned} S &= 1 + z^p + \ldots + z^{(N-1)p} \\ z^p S &= z^p + z^{2p} + \ldots + z^{Np} \end{aligned}$$

so that $S(1 - z^p) = 1 - z^{Np}$. Now, $z^{Np} = e^{\frac{2\pi i}{N} \cdot Np} = 1$, so $S(1 - z^p) = 0$. This leaves $S = 0$ or $1 - z^p = 0$.

When will $1 - z^p = 0$? When $p = 0, \pm N, \pm 2N, \ldots$. But $p = j - k$, where j, k were taken from $0, 1, 2, \ldots, N-1$. The smallest value of $j - k$ is then $-(N-1)$, and the largest value is $(N-1)$. Therefore, $1 - z^p = 0$ if and only if $j = k$, and in that case, we are taking the inner product of $\mathbf{v}^{(j)}$ with itself, and in that case, $S = \sum_{t=0}^{N-1} 1 = N$.

If we assume therefore, that $j \neq k$, then S must be zero; and this is what we wanted to prove:

Theorem: The Fourier vectors form an orthogonal basis for C^n.

13.2.5 Exercises:

1. Write out the four roots of unity, and show where they are on the unit circle.

2. If the interval $[0, 2\pi]$ is subdivided into 4 equal subintervals, how long is each one? What are the left endpoints?

3. Write out the 4×4 matrix resulting from

$$\left[\mathbf{v}^{(0)}\ \mathbf{v}^{(1)}\ \mathbf{v}^{(2)}\ \mathbf{v}^{(3)}\right]$$

   Show by direct computation that these are orthogonal vectors. *Hint 1: It might be easier to leave them in complex (versus polar) form. Hint 2: Be sure to use the complex conjugate when taking the inner products!*

4. Show that, if $e^{i\theta}$ is one of the p roots of unity, then so is its conjugate, $e^{-i\theta}$.

13.2.6 Interpreting the Matlab Results

Suppose that we want to perform a discrete Fourier Transform on a function f over the interval $[a, b]$. For now[4], assume that the interval has been shifted to the origin, so that the interval is $[0, B]$. If we cut the interval into N equal subintervals, then:

$$\Delta x = \frac{B}{N}, \text{ or } B = N \cdot \Delta x$$

To do the Fourier analysis, it is assumed this interval is going from $[0, 2\pi]$. To do this, we must take:

$$x_j \in [0, B] \longrightarrow t_j = \frac{2\pi}{B} \cdot x_j \in [0, 2\pi]$$

or to translate back:

$$t_j \in [0, 2\pi] \longrightarrow x_j = \frac{B}{2\pi} t_j \in [0, B]$$

so we see that $f(x)$ over the interval $[0, B]$ is the same as $f\left(\frac{B}{2\pi}t\right)$ over the interval $[0, 2\pi]$.

Let us now hook up the coefficients that Matlab will give us to our original definition of the Fourier series. Putting this into Equation 13.1, we get:

$$f(x) = a_0 + \sum_{k=1}^{\infty} a_k \cos(k\frac{2\pi}{B}x) + \sum_{k=1}^{\infty} b_k \sin(k\frac{2\pi}{B}x)$$

Now, inserting $B = N \cdot \Delta x$, we get the final discretized representation corresponding to Equation 13.2.

[4]We will show later that the shift to the origin leaves the Transform invariant

SUMMARY:

If we have N points with spacing Δx, and these points are labeled

$$\{f_0, f_1, \ldots, f_{N-1}\}$$

with $x \in [0, B]$, then the Matlab coefficients:

```
F=fft(f)
```

correspond to:

$$f_n = a_0 + \sum_{k=1}^{N/2} a_k \cos\left(\frac{2\pi x_n}{N\,\Delta x}k\right) + b_k \sin\left(\frac{2\pi x_n}{N\,\Delta x}k\right) \tag{13.3}$$

via the relations:

$$a_0 = \frac{F(1)}{N}$$
$$a_k = \frac{2}{N}\cdot \mathrm{Real}\,(F(k+1))$$
$$b_k = -\frac{2}{N}\cdot \mathrm{Imag}\,(F(k+1))$$

Remarks about these coefficients:

1. Recall that the period and frequency of $\sin(At)$ or $\cos(At)$ are defined as:

   $$\text{Period } = \frac{2\pi}{A}, \quad \text{Frequency } = \frac{1}{\text{Period}} = \frac{A}{2\pi}$$

   where the *Period* is measured in time per cycle, and *Frequency* is measured in cycles per time.

2. Note the periods of the functions in Equation 13.3, as $k = 1, 2, \ldots, \frac{N}{2}$ are (measured in Time per Cycle):

   $$N \cdot dt, \quad \frac{N}{2}\cdot dt, \quad \frac{N}{3}\cdot dt, \ldots, \frac{N}{N/2}\cdot dt = 2\cdot dt \tag{13.4}$$

   or as frequencies: (measured in Cycles per Time):

   $$\frac{1}{N}\cdot\frac{1}{dt}, \quad \frac{2}{N}\cdot\frac{1}{dt}, \quad \frac{3}{N}\cdot\frac{1}{dt}, \ldots, \frac{N/2}{N}\cdot\frac{1}{dt} = \frac{1}{2}\cdot\frac{1}{dt} \tag{13.5}$$

   EXAMPLE: Given $N = 16$ samples, each at $\Delta x = 1$ time unit. What are the possible periods that the discrete Fourier Transform will pick up?

   There are 8 possible periods: 16, 8, $5\frac{1}{3}$, 4, 3.2, $2\frac{2}{3}$, $2\frac{3}{7}$, 2.

   Note that the shortest period is 2, twice the sampling distance Δx. The longest period is 16 time units long- as long as the entire interval.

   You might have noticed that in this example, there were only 8 periods, but 16 terms of the FFT. See Exercise Set 13.1.2 for a hint on why that was.

3. The term a_0 is the mean (average) of the original time series.

4. The frequency content is given by:

$$\text{freq}(k) = \sqrt{a_k^2 + b_k^2} = \frac{2}{N}|X(k+1)|, \quad k = 1, 2, \ldots, \frac{N}{2}$$

   from which we get the *Power Spectrum*, which is a plot of the sizes of the Fourier transform, $|F|$. This plot reveals the frequency content of the signal, and we will explore this in the next section.

13.3 Applying the FFT

In this section, we will apply the FFT to various signals and look at ways of manipulating the output.

13.3.1 Matlab Example: Fourier Interpolation

In interpolation problems, we build a model function from data points, from which we can evaluate the function at new points of the domain.

For example, suppose I have 16 data points from an unknown function. Then using the Fourier transform, I can approximate the function using a sum of sines and cosines. In this case, we will use the following function, where we will need to input the new values of x, the interval length used in the FFT, Δx, and the FFT. The output will be the function evaluated at the new domain values.

```
function y=fftapprox(x,dx,F)
%FUNCTION y=fftapprox(x,dx,F)
%  Returns new function outputs for data in vector x
%  using the FFT coefficients in F.

N=length(F);  %N should be divisible by 2!
m=N/2;
M=(2*pi)/(N*dx);
x=M*x(:);  %Make sure x is a column vector
F=F(:);

a0=F(1)/N;
a=2*real(F(2:m+1))/N;
b=-2*imag(F(2:m+1))/N;

z=cos(x*(1:m))*a+sin(x*(1:m))*b;
y=a0*ones(size(x))+z;
```

To use this function, try the following example using pure harmonic data:

```
k=0:63;
f=cos((7*pi/8)*k);
F=fft(f);
t=0:63;
x=linspace(0,63,1000);
m=1;
y=fftapprox(x,m,F);

plot(t,f,'r*',x,y,'b-'); %Blue curve is the FFT
```

13.3.2 Exercises:

1. Change the function in the last script file to be random values:

   ```
   f=2*rand(size(k))-1;
   ```

   and report what happens.

2. Without loss of generality, we can always assume that the sampling times are integer valued, $0, 1, 2, \ldots, N-1$ with $\Delta x = 1$, and N a power of 2. The Fourier approximation will assume that $F(N) = F(0)$, so that this is really the interval from $[0, N]$. The only difficulty is in translating the frequencies of the spectrum into frequencies on $[0, B]$, which is what was desired.

   For example, if the interval were really $[0, 2]$, but we input `0:63` as sampling times, the real sampling times are:

   $$0, 2/64, 4/64, \ldots, 2 \cdot 63/64$$

   with $\Delta x = \frac{2}{64}$. In this case, try the following script, noting the differences:

   ```
   k=0:63;
   f=sin((pi/8)*k)+2*cos((7*pi/8)*k);

   %Assume a domain of [0,2] with 64 samples
   F=fft(f);
   t=(2/64)*(0:63);
   x=linspace(0,2,1000);
   m=2/64;
   y=fftapprox(x,m,F);

   plot(t,f,'r*',x,y);
   ```

3. How would that last script file change if we assumed the domain was $[0, 3\pi]$ (with everything else the same)?

13.3.3 Matlab Example: Analyzing a Known Signal

Let $y = \sin(\frac{\pi}{8}k) + 2\cos(\frac{7\pi}{8}k)$, with $k = 0, 1, \ldots, 63$.

Before going into Matlab, let's check the periods involved in this function:

$$\frac{2\pi}{\frac{\pi}{8}} = 16, \quad \frac{2\pi}{\frac{7\pi}{8}} = \frac{16}{7}$$

which correspond to the following frequencies:

$$\frac{1}{16} = \frac{4}{64}, \quad \frac{7}{16} = \frac{28}{64}$$

1. Set up the domain and range information:

   ```
   k=0:63;
   f=sin((pi/8)*k)+2*cos((7*pi/8)*k);
   ```

2. Take the Fourier Transform:

   ```
   F=fft(f);
   ```

3. Look at the Spectrum:

   ```
   plot(abs(F),'k-*');
   ```

4. We should see 4 spikes in the spectrum, and the spectrum is symmetric. The symmetry is due to the fact that the function is real. Note that we'll only be using the first half of the spectrum in Equation 13.3. The spikes of interest are at positions 5 and 29. How do we translate these back to the original period? Using Equation 13.3, we get that position 5 refers to the 4th coefficient, and position 29 corresponds to the 28th coefficient- exactly what was predicted.

5. Now see what the effect of noise is:

   ```
   g=f+0.2*randn(size(f));
   F=fft(g);
   plot(abs(F),'k-*');
   ```

   The effect of adding noise was the addition of some low powered frequencies. Let's try to **filter** them out by setting all the coefficients less than 10 to zero:

   ```
   [vals,idx]=find(abs(F)<10);
   Fnew=F;
   Fnew(gidx)=zeros(size(gidx));
   g=ifft(Fnew);
   ```

   Now compare the values of g to the original values of f.

13.3.4 Matlab Exercises: Filtering, Part I

In these exercises, we'll use the normalized spectrum to zero out small powers (by percent).

The basic idea is to normalize the spectrum before deciding on which frequencies to filter. We'll do this by setting:

$$S = \sum_{k=1}^{N/2} |F(k)|$$

(Sum over all coefficients if your signal is complex). Where we assume that F has already been computed via `F=fft(f)`. Then obtain the normalized powers (in Matlab notation):

```
Fn=abs(F)/S;
```

We'll perform the filtering by creating a temporary vector in which to hold the new Fourier coefficients (don't change the originals unless necessary- we may need them later!). For example, if we want to delete all coefficients contributing less than 5% of the total, we would do the following:

```
[vals,idx]=find(Fn<0.05);
Ffilt=F;
Ffilt(idx)=zeros(size(F(idx));
fnew=ifft(Ffilt);
```

The last command gives the filtered signal (if real, go ahead and truncate the imaginary parts).

13.3.5 Matlab Example: Sunspot Analysis

This example has a built-in script. You can type: `sunspots` in the command window to see the slideshow. We'll do the same example here:

The problem: We're given data corresponding to the number of sunspots counted on the sun every year between 1700 and 1987. We'd like to analyze it to see if there is some periodicity in the number of sunspots. We assume the domain to be 0 to 288 years, and the data below corresponds to one sample per year (therefore, $B = N = 288$, and $\Delta x = 1$).

```
load sunspot.dat
year=sunspot(:,1); sun=sunspot(:,2);
Y=fft(sun);
Y(1)=[];  %Remove the mean
plot(abs(Y(1:143));  %We'll only plot the relevant half.
[a,b]=max(abs(Y(1:143));
```

We see that the 26th position gave the max- how do we translate that into years? From the formulas for the "Time per Cycle", (Equation 13.4) we get:

$$\frac{N \cdot \Delta x}{k} = \frac{288 \cdot 1}{26} \approx 11.0769 \text{ years}$$

from which comes the adage that we get a maximum number of sunspots approximately every 11 years.

We could use these time values for our plot:

```
T=288./(1:143);
plot(T,abs(Y(1:143)));
m=max(abs(Y(1:143));
axis([1 20 0 m]);
```

Finally, we could use only the largest powers of the spectrum to approximate the sunspot data. For example, let's compare the original to the approximate curve, if we only use the largest 10 values of the spectrum:

```
Y=fft(sun);
[vals1,idx1]=sort(abs(Y));
idx1=idx1(288:-1:1);  %Reverse so that largest is first
Ynew=zeros(size(Y));
Ynew(idx1(1:10))=Y(idx1(1:10));
sun2=ifft(Ynew);
sun2=real(sun2);  %Numerically, the imaginary parts are very small
plot(1:288,sun,1:288,sun2);
```

13.3.6 Matlab Exercise: Filtering, Part II

In the previous filtering example, we took only those frequencies that contributed a significant amount to the whole (by disregarding frequencies whose contribution was less than some percent). There are other methods to filter a signal, and this section will discuss one of them: Band-pass filtering.

In band-pass filtering, we will retain only those frequencies that are most prevalent in the signal. To do this will require us to examine the *distribution* of frequencies in the signal, and we will use a histogram to plot this. Before continuing, let's do some examples.

In Matlab, there are several ways of producing pseudo-random numbers- either the `rand` command or the `randn` command[5]. What is the difference? Take a look at the following two script files, and consider the output plot in Figure XX:

```
f=rand(10000,1);
subplot(2,2,1)
hist(f);
subplot(2,2,2)
hist(f,100);
f=randn(10000,1);
subplot(2,2,3)
hist(f);
```

[5]With the statistics toolbox, there are lots more, but these two functions come with all distributions of Matlab

```
subplot(2,2,4)
hist(f,100);
```

We see that in the upper left and upper right subplots, `rand` produces a uniform distribution of values; that is, for any two intervals of the same size in $[0, 1]$, the probability of choosing a number in one interval is the same as the probability of choosing a number in any interval of the same length. We also see the effect of the second argument: `hist(f,100)` produces 100 bins along the horizontal axis.

In the lower left and lower right subplots, `randn` produces a *normal* distribution (also known as a Gaussian distribution). That is, we are choosing values close to zero much more often than values away from zero. And we also see what happens if we choose 100 bins along the horizontal axis.

Let's try this technique with frequency content instead of random values (the line numbers are added only for readability- don't type them in!)

```
1  load sunspot.dat
2  f=sunspot(:,2);
3  F=fft(f);
4  hist(abs(F(2:end)),100);
5  %Retain frequencies within 1 standard deviation of the mean
6  mF=mean(abs(F(2:end)));
7  stdF=std(abs(F(2:end)));
8  idx1=find(abs(F)<(mF+stdF) & abs(F)>(mF-stdF)); % & means AND
9  F1=zeros(288,1);
10 F1(1)=F(1);
11 F1(idx1)=F(idx1);
12 y1=ifft(F1);
13 plot(1:288,f,1:288,real(y1))
14 %Retain frequencies outside one deviation from the mean
15 idx2=find(abs(F)>(mF+stdF) | abs(F)<(mF-stdF)); % | means OR
16 F2=zeros(288,1);
17 F2(1)=F(1);
18 F2(idx2)=F(idx2);
19 y2=ifft(F2);
20 plot(1:288,f,1:288,real(y2));
21 %Now compare the spectra:
22 t=2:140;
23 plot(t,abs(F(t)),t,abs(F1(t)),t,abs(F2(t));
24 legend('Original','Within 1 STD','Outside 1 STD');
```

The output of this sample script is given in Figures XX, corresponding to lines 4, 13, 20 and 23. From line 4, we see that most of the frequency content is "small". From lines 6 and 7, we get a mean of 366 with a standard deviation of 561. In line 8, we are getting all frequencies whose powers are less than 927, and in line 15, we are retaining all frequencies whose powers are greater than 927. In Figure XX, we see the histogram of the powers, and in Figure XX, we see the effect of retaining the powers within 1 standard deviation of the mean. In Figure XX, we see the effect of retaining frequencies whose powers are greater than 1 standard deviation from the mean. Finally, the last figure shows which parts of the spectrum are being retained in each case.

In general, we will probably expect that noise in the data will produce a spike in the small powers of the spectrum.

This is only one way to do a band-pass filtering. The exercises below let you try some other methods.

13.3.7 Matlab Exercise: Leakage

Try the following script file and write up the results of what you see:

```
t=0:63;
y=cos((7*pi/8)*t);
y1=zeros(1,64);
y1(1:52)=y(1:52);
F1=fft(y);
F2=fft(y1);
figure(1)
plot(abs(F1))
figure(2)
plot(abs(F2))
```

This effect is known as "leakage": while the frequency content of the signal did not change, the power spectrum did. Can you give a heuristic explanation of what happened?

13.3.8 Matlab Exercise: The Gibb's Phenomena

What happens when you try to fit a round peg into a square hole? This is analogous to what is known as the Gibb's phenomena. In out particular case, this refers to trying to use smooth functions to approximate functions that are either discontinuous, or has very steep parts.

A template example is a square wave. This can be defined by:

$$f(x) = \begin{cases} 1, & \text{if } 0 \leq x \leq \pi \\ -1, & \text{if } \pi < x \leq 2\pi \\ 1 & \text{if } 2\pi < x \leq 3\pi \\ \vdots & \vdots \end{cases}$$

which can be modeled by a sum of odd harmonics (we'll be off by a scaling factor, but you'll see what happens momentarily).

$$f(x) \quad = \sin(x) + \tfrac{\sin(3x)}{3} + \tfrac{\sin(5x)}{5} + \dots$$

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

Explain what happens as we use more and more terms of the Fourier approximation to f. That is, type in the following script and explain what you see:

```
t=linspace(0,3*pi,5000);
x=zeros(1,length(t));
k=1:2:71;
m=length(k);
Y=zeros(m,length(t));
for j=1:m
  x=x+(1/k(j))*sin(k(j)*t);
  Y(j,:)=x;
end
plot(Y(1:6:end,:)');
```

Conversely, lets go ahead and compute the Fourier transforms of a portion of the square wave. Perform the following experiment and report on what you observe:

```
j=6:10;
k=2.^j;
for n=1:4
  m1=floor(k(n)/3);
  m2=ceil(2*k(n)/3);
  y=zeros(k(n),1);
  y(m1:m2)=ones(size(y(m1:m2)));
  F=fft(y);
  plot(abs(F(2:end)));
  pause
end
```

13.3.9 Properties of the Transform and Spectrum

In this section, we discuss several important properties of the Fourier Transform and its spectrum.

1. The transform is defined for complex-valued vectors as well as real-valued, although we have only worked with real-valued data. We might work with complex-valued data when looking at closed curves in the plane (closed will imply periodic, and means that the two ends of the curve are together). Let's try an example. For my data, I'll construct the letter "E"[6]. After this, I'll construct two more copies. Now we have three data sets in the complex plane, each is a translated, scaled or rotated version of the base. See Figure 13.1 for the results.

[6]This example is from M. Kirby's Geometric Data Analysis

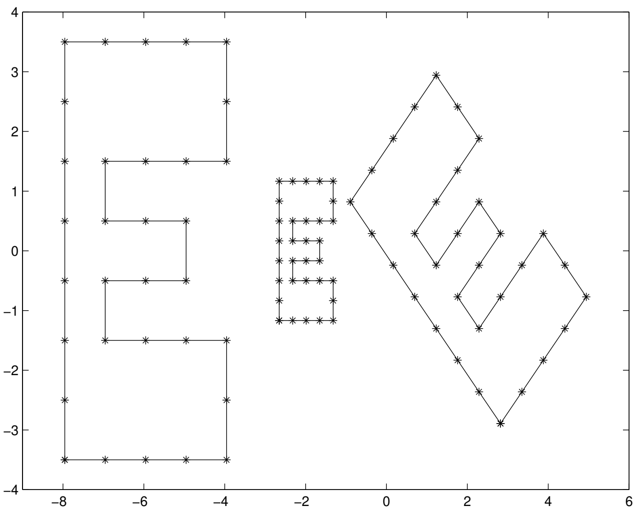

Figure 13.1: The "Letter E" data. Some are translated, rotated versions.

Next, we'll take the transform of each data set, and compare the spectra. See Figure 13.2 for the results. What can we conclude from this example?

2. What will happen if the sequence is shifted in time? Try the following experiment: Let $y = \sin(\frac{\pi}{8}k) + 2\cos(\frac{7\pi}{8}k)$, as before. Now we'll shift the series in time by some constant value, say by 13. Then:

```
k=0:63;
y1=sin((pi/8)*k)+2*cos((7*pi/8)*k);
y2=sin((pi/8)*(k+13))+2*cos((7*pi/8)*(k+13));
Y1=fft(y1);
Y2=fft(y2);
plot(abs(Y1),'r-o');
hold on
plot(abs(Y1),'g-*');
hold off
```

3. From the previous examples, we see that:

   > The spectrum is symmetric for real data, but not for complex data.

   And

   > For real data, the spectrum is invariant under shifts in time. For complex data, the spectrum is invariant to translations, scaling and rotations.

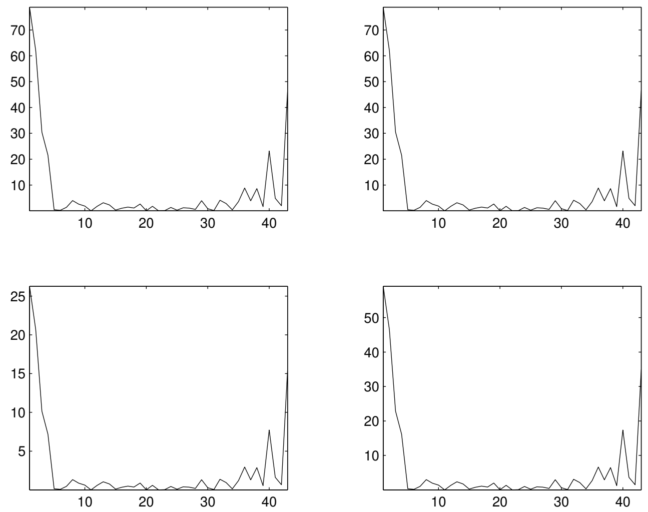

Figure 13.2: The spectra for the letter E data. All are the same!

4. The transform is a *linear operator*. From linear algebra, we will recall that a linear operator, $\mathcal{F} : X \to Y$,

$$\mathcal{F}(a\mathbf{x} + b\mathbf{y}) = a\mathcal{F}(\mathbf{x}) + b\mathcal{F}(\mathbf{y})$$

Show this. (NOTE: You can use the matrix version of the transform if you'd like).

13.4 Short Term Fourier and Windowing

If you've got a large data file, you don't want to put it all into Matlab at once. Instead, you may want to analyze only part of the time series, then shift the transform over to the next part, and so on.

In this case, you will be doing multiple transforms. Before getting into the details, lets look at some Matlab commands that will parse a large time series into a matrix. Let's begin with a simple time series consisting of the integers from 1 to 2400. The first idea will be to make this into a matrix:

$$X = \begin{bmatrix} 1 & 2 & \dots & 2385 \\ 2 & 3 & \dots & 2386 \\ 3 & 4 & \dots & 2387 \\ \vdots & & & \vdots \\ 16 & 17 & \dots & 2400 \end{bmatrix}$$

which in Matlab is done by:

```
for j=1:16
  X(j,:)=j:(2385+(j-1));
end
```

In general, with the integers from 1 to N with cuts every k integers:

```
k=16;
N=2400;
m=N-(k-1);  %This was how we got the 2385 last time
for j=1:k
  X(j,:)=j:(m+j-1);
end
```

Its also easy to skip values. For example, suppose we wanted the array:

$$X = \begin{bmatrix} 1 & 5 & 9 & \ldots & 2385 \\ 2 & 6 & 10 & \ldots & 2386 \\ 3 & 7 & 11 & \ldots & 2387 \\ \vdots & & & \vdots & \\ 16 & 20 & 24 & \ldots & 2400 \end{bmatrix}$$

The changes would be:

```
k=16;
N=2400;
skip=7;
m=N-(k-1);  %This was how we got the 2385 last time
for j=1:k
  X(j,:)=j:skip:(m+j-1);
end
```

Perhaps it would be useful to have this as a function. Write the following function out and save it for later:

```
function X=lagmatrix(x,numrows,skip)
%function X=lagmatrix(x,numrows,skip)
%This function creates a lag matrix- the output matrix
%  has numrows rows, and the number of columns will vary
%  depending on how many full columns we can get from size
%  of x.
%  Example:  Try X=lagmatrix(1:65,7,3)
k=numrows;
N=length(x);
m=N-(k-1);
for j=1:k
  X(j,:)=x(j:skip:(m+j-1));
end
```

Try out the example.

13.4.1 Windowed Fourier Transform

What will the following commands produce?

```
load sunspot.dat
X=lagmatrix(sunspot(:,2),32,1);
Z=fft(X);
imagesc(abs(Z(2:end,:)));
colorbar
```

This is a look at how the spectrum of the Fourier Transform changes in time using data that is cut into lengths of 32 time units. Try the same example, but skip 5. Is anything noticeably different?

```
t=linspace(0,15,2048);
idx1=find(t<=7);
idx2=find(t>7&t<=7.2);
idx3=find(t>7.2);
y=zeros(1,2048);
y(idx1)=sin(5*t(idx1));
y(idx2)=sin(5*t(idx2))+1;
y(idx3)=sin(5*t(idx3))+sin(12*(t(idx3)));
plot(t,y)
Y=lagmatrix2(y,128,128);
Z=fft(Y');
imagesc(abs(Z));
```

13.5 Fourier and Biological Mechanisms

Here we look at two particular applications of the Fourier Transform to brain processing.

13.5.1 The Visual Cortex

13.5.2 The Cochlea and Cochlear Implants

13.6 Chapter Summary