Chapter 8

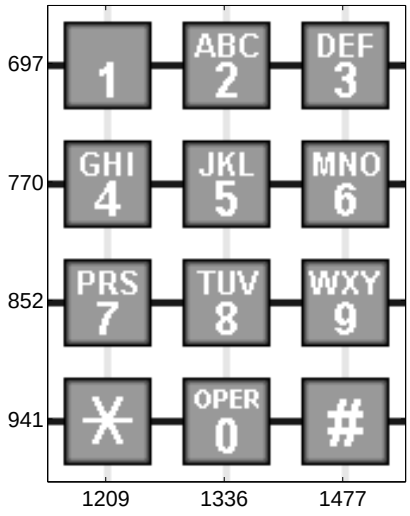
Fourier Transforms

8.1 Touch-Tone Dialing

We all use finite Fourier transforms every day without even knowing it. Cell phones, disc drives, DVDs and JPEGs all involve FFTs.

Touch-tone telephone dialing is another example. The basis for touch-tone dialing is the Dual Tone Multi-Frequency system. The program `touchtone` demonstrates how DTMF tones are generated and decoded. The telephone dialing pad acts as a 4-by-3 matrix. Associated with each row and column is a frequency. These basic frequencies are

```
fr = [697 770 852 941];
fc = [1209 1336 1477];
```



If `s` is a character that labels one of the buttons on the key pad, the corresponding row index `k` and column index `j` can be found with

```
switch s
   case '*', k = 4; j = 1;
   case '0', k = 4; j = 2;
   case '#', k = 4; j = 3;
   otherwise,
      d = s-'0'; j = mod(d-1,3)+1; k = (d-j)/3+1;
end
```

A key parameter in digital sound is the sampling rate.

```
Fs = 32768
```

A vector of points in the time interval $0 \le t \le 0.25$ at this sampling rate is:

```
t = 0:1/Fs:0.25
```

The tone generated by the button in position `(k,j)` is obtained by superimposing the two fundamental tones with frequencies `fr(k)` and `fc(j)`.

```
y1 = sin(2*pi*fr(k)*t);
y2 = sin(2*pi*fc(j)*t);
y = (y1 + y2)/2;
```

If your computer is equipped with a sound card, the MATLAB statement

```
sound(y,Fs)
```

plays the tone.

Figure 8.1 is the display produced by `touchtone` for the `'1'` button. The top subplot depicts the two underlying frequencies and the bottom subplot shows a portion of the signal obtained by averaging the sine waves with those frequencies.

The data file `touchtone.mat` contains a recording of a telephone being dialed. Is it possible to determine the phone number by listening to the signal generated? The statements

```
load touchtone
```

loads both a signal `y` and a scalar `Fs` in the workspace. In order to reduce file size, the vector `y` has been saved with 8-bit integer components in the range $-127 \le y_k \le 127$. The statement

```
y = double(y)/128;
```

rescales the vector and converts it to double precision for later use. The statements

```
n = length(y);
t = (1:n)/Fs
```

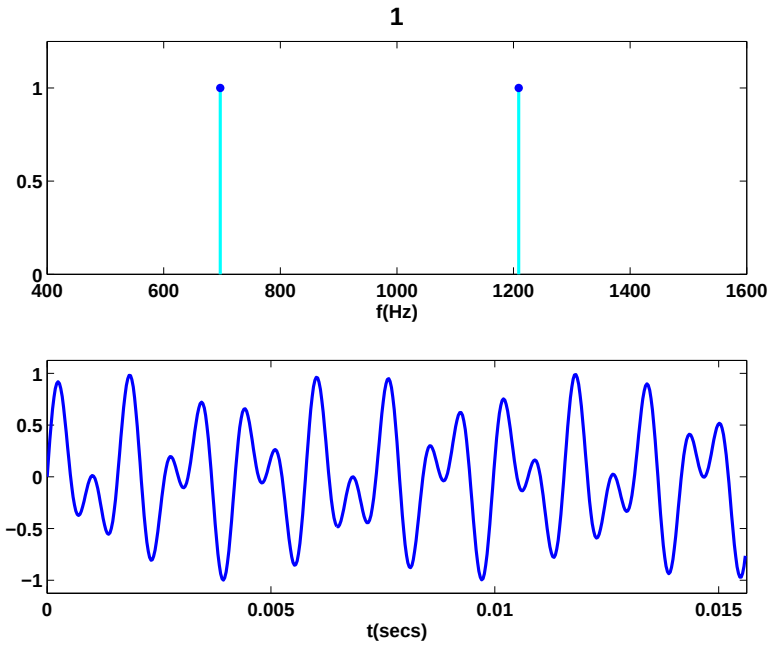


Figure 8.1. *The tone generated by the 1 button*

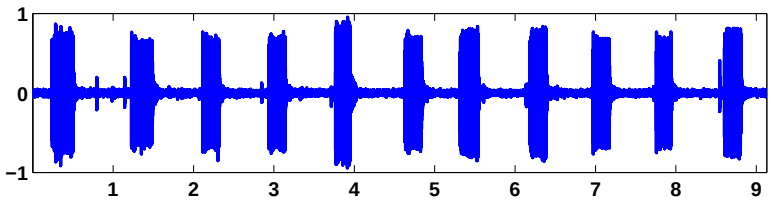


Figure 8.2. *Recording of an 11-digit telephone number*

reproduce the sample times of the recording. The last component of `t` is `9.1309`, indicating that the recording lasts a little over nine seconds. Figure 8.1 is a plot of the entire signal.

This signal is noisy. You can even see small spikes on the graph at the times the buttons were clicked. It is easy to see that eleven digits were dialed, but on this scale, it is impossible to determine the specific digits.

Figure 8.3 shows the magnitude of the FFT, the *finite Fourier transform*, of the signal, which is the key to determining the individual digits.

The plot was produced with

```
p = abs(fft(y));
```

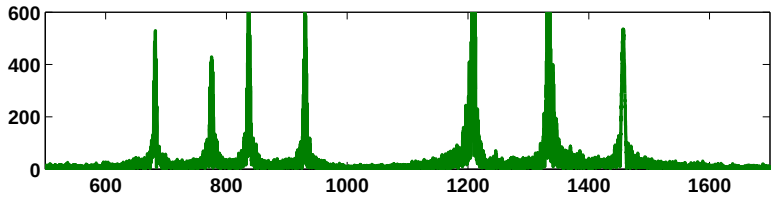


Figure 8.3. *FFT of the recorded signal*

```
f = (Fs/n)*(1:n);
plot(f,p);
axis([500 1700 0 600])
```

The x-axis corresponds to frequency. The `axis` settings limit the display to the range of the DTMF frequencies. There are seven peaks, corresponding to the seven basic frequencies. This overall FFT shows that all seven frequencies are present someplace in the signal, but it does not help determine the individual digits.

The `touchtone` program also lets you break the signal into eleven equal segments and analyze each segment separately. Figure 8.4 is the display from with the first segment.

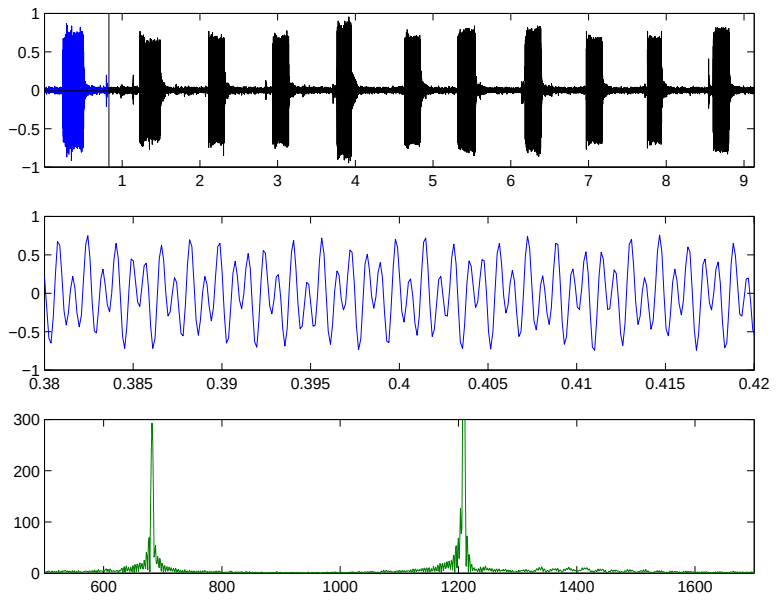


Figure 8.4. *The first segment and its FFT*

For this segment, there are only two peaks, indicating that only two of the

basic frequencies are present in this portion of the signal. These two frequencies come from the `'1'` button. You can also see that the wave form of a short portion of the first segment is similar to the wave form that our synthesizer produces for the `'1'` button. So, we can conclude that the number being dialed in `touchtones` starts with a 1. An exercise asks you to continue the analysis and identify the complete phone number.

8.2 Finite Fourier Transform

The finite, or discrete, Fourier transform of a complex vector y with n elements is another complex vector Y with n elements

$$Y_{k+1} = \sum_{j=0}^{n-1} \omega^{jk} y_{j+1}$$

where ω is a complex nth root of unity,

$$\omega = e^{-2\pi i/n}$$

This notation uses i for the complex unit, $\sqrt{-1}$, and j and k for indices that run from 0 to $n-1$. The subscripts $j+1$ and $k+1$ run from 1 to n, corresponding to the range usually associated with linear algebra and MATLAB vectors.

The Fourier transform can be expressed with matrix-vector notation

$$Y = Fy$$

where the finite Fourier transform matrix F has elements

$$f_{k+1,j+1} = \omega^{jk}$$

It turns out that F is nearly its own inverse. More precisely F^H, the complex conjugate transpose of F, satisfies

$$F^H F = nI$$

so

$$F^{-1} = \frac{1}{n} F^H$$

This allows us to invert the Fourier transform.

$$y = \frac{1}{n} F^H Y$$

Hence

$$y_{j+1} = \frac{1}{n} \sum_{k=0}^{n-1} Y_{k+1} \bar{\omega}^{jk}$$

where $\bar{\omega}$ is the complex conjugate of ω

$$\bar{\omega} = e^{2\pi i/n}$$

We should point out that this is not the only notation for the finite Fourier transform in common use. The minus sign in the definition of ω after the first equation sometimes occurs instead in the definition of $\bar{\omega}$ used in the inverse transform. The $1/n$ scaling factor in the inverse transform is sometimes replaced by $1/\sqrt{n}$ scaling factors in both transforms.

In MATLAB, the Fourier matrix F could be generated for any given `n` by

```
omega = exp(-2*pi*i/n);
j = 0:n-1;
k = j'
F = omega.^(k*j)
```

The quantity `k*j` is an *outer product*, an n-by-n matrix whose elements are the products of the elements of two vectors. However, the built-in function `fft` takes the finite Fourier transform of each column of a matrix argument, so an easier, and quicker, way to generate F is

```
F = fft(eye(n))
```

The function `fft` uses a fast algorithm to compute the finite Fourier transform. The first "`f`" stands for both "fast" and "finite". A more accurate name might be `ffft`, but nobody wants to use that. We discuss the fast aspect of the algorithm in a later section.

8.3 `fftgui`

The GUI `fftgui` allows you to investigate properties of the finite Fourier transform. If `y` is a vector containing a few dozen elements,

```
fftgui(y)
```

produces four plots

```
real(y)          imag(y)
real(fft(y))     imag(fft(y))
```

You can use the mouse to move any of the points in any of the plots, and the points in the other plots respond.

Please run `fftgui` and try the following examples. Each illustrates some property of the Fourier transform. If you start with no arguments,

```
fftgui
```

all four plots are initialized to `zeros(1,32)`. Click your mouse in the upper left hand corner of the upper left hand plot. You are taking the `fft` of the first unit vector, with one in the first component and zeros elsewhere. This should produce figure 8.5.

The real part of the result is constant and the imaginary part is zero. You can also see this from the definition

$$Y_{k+1} = \sum_{j=0}^{n-1} y_{j+1} e^{-2ijk\pi/n}, \ k = 0, \ldots, n-1$$

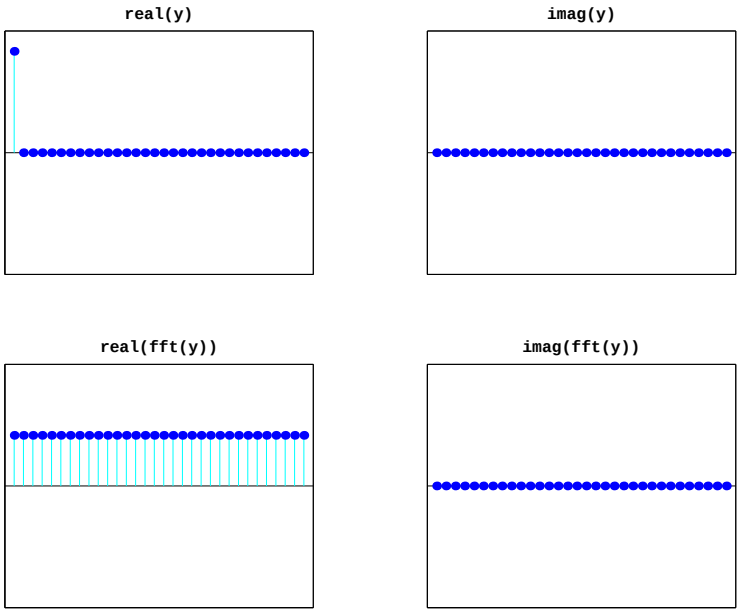


Figure 8.5. *FFT of the first unit vector is constant*

if $y_1 = 1$ and $y_2 = \cdots = y_n = 0$. The result is

$$Y_{k+1} = 1 \cdot e^0 + 0 + \cdots + 0 = 1, \text{ for all } k$$

Click y_1 again, hold the mouse down, and move the mouse vertically. The amplitude of the constant result varies accordingly.

Next, try the second unit vector. Use the mouse to set $y_1 = 0$ and $y_2 = 1$. This should produce figure 8.6.

You are seeing the graph of

$$Y_{k+1} = 0 + 1 \cdot e^{-2ik\pi/n} + 0 + \cdots + 0$$

The nth root of unity can also be written

$$\omega = \cos\delta - i\sin\delta, \text{ where } \delta = 2\pi/n$$

Consequently, for $k = 0, \cdots, n-1$,

$$\text{real}(Y_{k+1}) = \cos k\delta, \ \text{imag}(Y_{k+1}) = -\sin k\delta$$

We have sampled two trig functions at n equally spaced points in the interval $0 \le x < 2\pi$. The first sample point is $x = 0$ and the last sample point is $x = 2\pi - \delta$.

Now set $y_3 = 1$ and vary y_5 with the mouse. One snapshot is figure 8.6.

We have graphs of

$$\cos 2k\delta + \eta\cos 4k\delta \text{ and } -\sin 2k\delta - \eta\sin 4k\delta$$

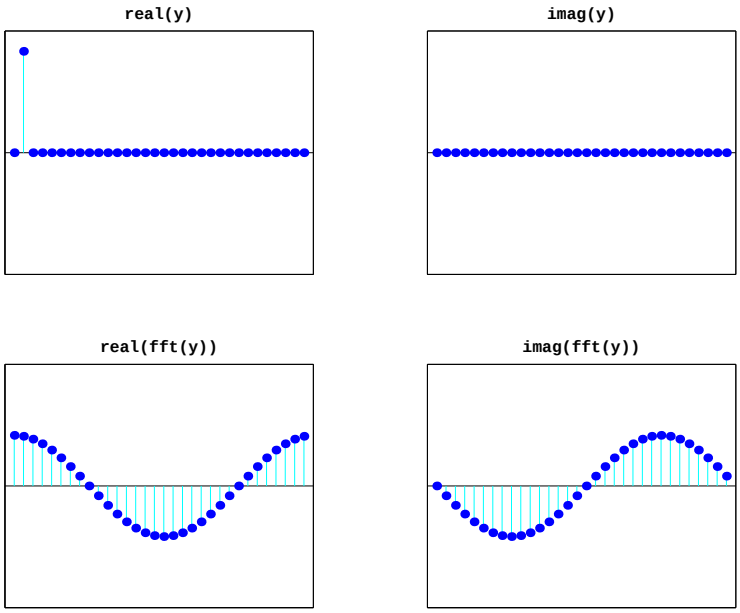


Figure 8.6. *FFT of the second unit vector is a pure sinusoid*

for various values of $\eta = y_5$.

The point just to the right of the midpoint of the x-axis is particularly important. It is known as the *Nyquist point*. With the points numbered from 1 to n for even n, it's the point with index $\frac{n}{2} + 1$. If $n = 32$, it's point number 17. Figure 8.8 shows that the `fft` of a unit vector at the Nyquist point is a sequence of alternating $+1$'s and -1's.

Now let's look at some symmetries in the FFT. Make several random clicks on the `real(y)` plot. Leave the `imag(y)` plot flat zero. Figure 8.9 shows an example. Look carefully at the two `fft` plots. Ignoring the first point in each plot, the real part is symmetric about the Nyquist point and the imaginary part is antisymmetric about the Nyquist point. More precisely, if y is any real vector of length n and $Y = \text{fft}(y)$, then

$$\begin{aligned}
\text{real}(Y_1) &= \sum y_j \\
\text{imag}(Y_1) &= 0 \\
\text{real}(Y_{2+j}) &= \text{real}(Y_{n-j}),\ j = 0, \cdots, n/2 - 1 \\
\text{imag}(Y_{2+j}) &= -\text{imag}(Y_{n-j}),\ j = 0, \cdots, n/2 - 1
\end{aligned}$$

8.4 Sunspots

This section is an expansion of the MATLAB `sunspots` demo.

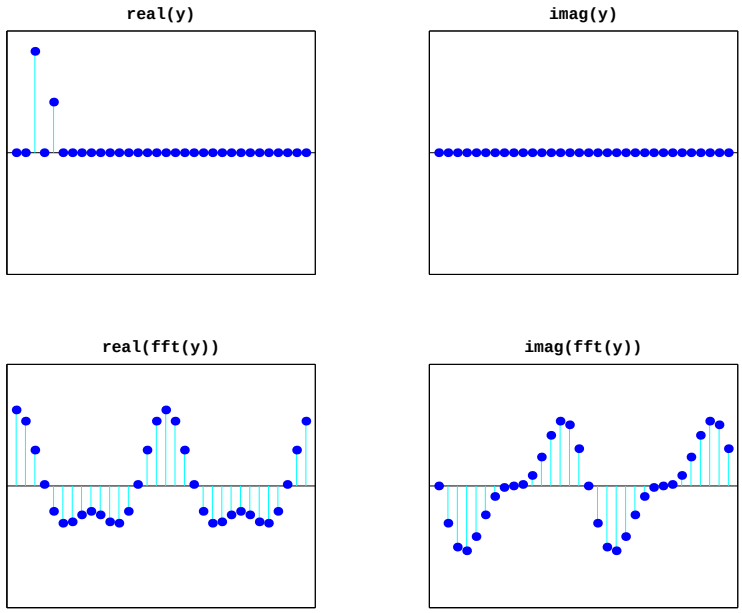


Figure 8.7. *FFT is the sum of two sinusoids*

For centuries people have noted that the face of the sun is not constant or uniform in appearance, but that darker regions appear at random locations on a cyclical basis. This activity is correlated with weather and other economically significant terrestrial phenomena. In 1848, Rudolf Wolfer proposed a rule that combined the number and size of these sunspots into a single index. Using archival records, astronomers have applied Wolfer's rule to determine sunspot activity back to the year 1700. Today the sunspot index is measured by many astronomers and the worldwide distribution of the data is coordinated by the Solar Influences Data Center at the Royal Observatory of Belgium [3].

The text file `sunspot.dat` in the MATLAB `demos` directory has two columns of numbers. The first column is the years from 1700 to 1987 and the second column is the average Wolfer sunspot number for each year.

```
load sunspot.dat
t = sunspot(:,1)';
wolfer = sunspot(:,2)';
n = length(wolfer)
```

There is a slight upward trend to the data. A least squares fit gives the trend line.

```
c = polyfit(t,wolfer,1);
trend = polyval(c,t);
```

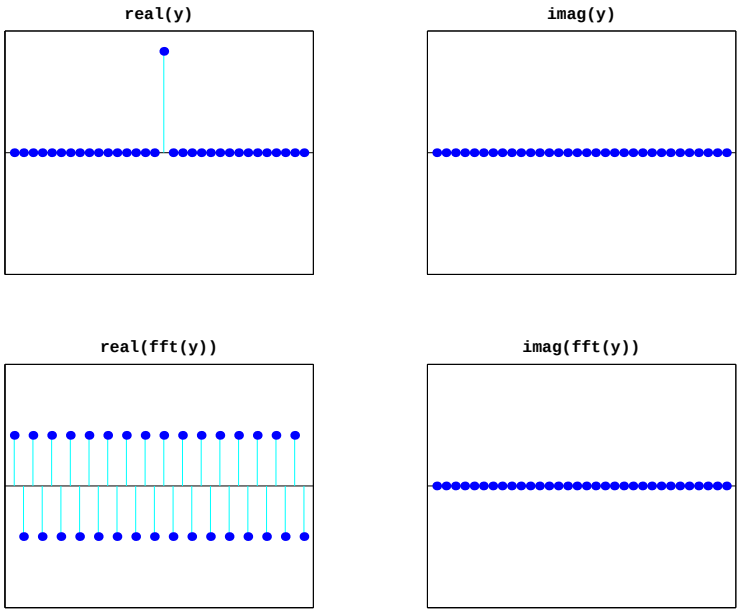


Figure 8.8. *The Nyquist point*

```
plot(t,[wolfer; trend],'-',t,wolfer,'k.')
xlabel('year')
ylabel('Wolfer index')
title('Sunspot index with linear trend')
```

You can definitely see the cyclic nature of the phenomenon. The peaks and valleys are a little more than 10 years apart.

Now, subtract off the linear trend and take the finite Fourier transform.

```
y = wolfer - trend;
Y = fft(y);
```

The first Fourier coefficient, `Y(1)`, can be deleted because subtracting the linear trend ensures that `Y(1) = sum(y)` is zero.

```
Y(1) = [];
```

The complex magnitude squared of Y is called the power and a plot of power versus frequency is a "periodogram." The frequency is the array index scaled by n, the number of data points. The time increment is one year, so the frequency units are cycles per year.

```
pow = abs(Y(1:n/2)).^2;
pmax = 20e6;
```

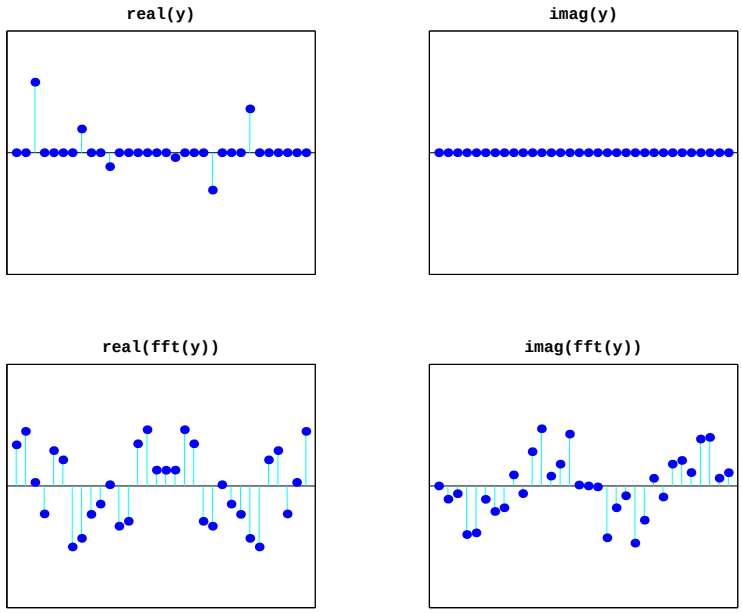


Figure 8.9. *Symmetry about the Nyquist point*

```
f = (1:n/2)/n;
plot([f; f],[0*pow; pow],'c-', f,pow,'b.', ...
   'linewidth',2,'markersize',16)
axis([0 .5 0 pmax])
xlabel('cycles/year')
ylabel('power')
title('Periodogram')
```

The maximum power occurs near frequency = 0.09 cycles/year. We would like to know the corresponding period in years/cycle. Let's zoom in on the plot and use the reciprocal of frequency to label the x-axis.

```
k = 1:36;
pow = pow(k);
ypk = n./k(2:2:end);  % Years per cycle
plot([k; k],[0*pow; pow],'c-',k,pow,'b.', ...
   'linewidth',2,'markersize',16)
axis([0 max(k)+1 0 pmax])
set(gca,'xtick',k(2:2:end))
xticklabels = sprintf('%5.1f|',ypk);
set(gca,'xticklabel',xticklabels)
xlabel('years/cycle')
ylabel('power')
```

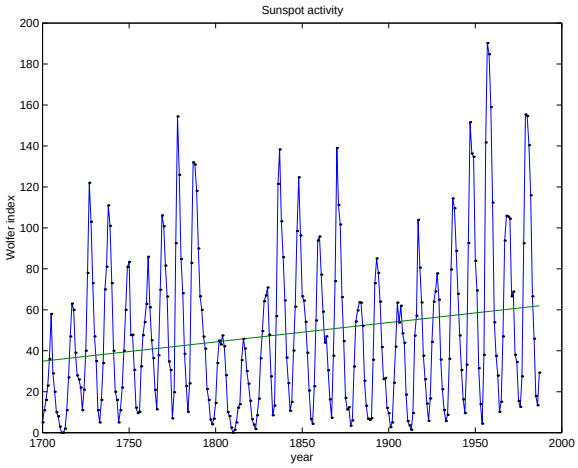


Figure 8.10. *Sunspot index*

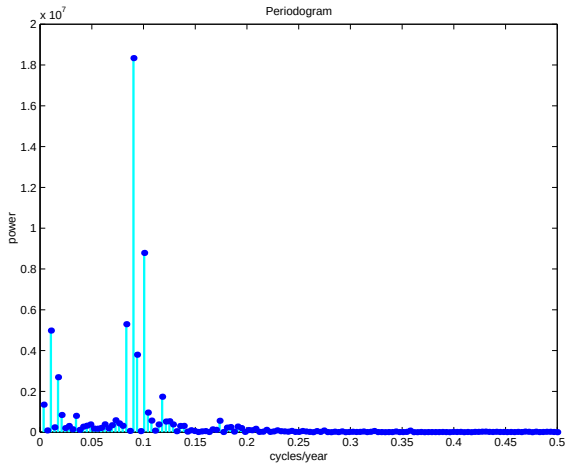


Figure 8.11. *Periodogram of the sunspot index*

```
title('Periodogram')
```

As expected, there is a very prominent cycle with a length of about 11 years. This shows that over the last 300 years, the period of the sunspot cycle has been slightly over 11 years.

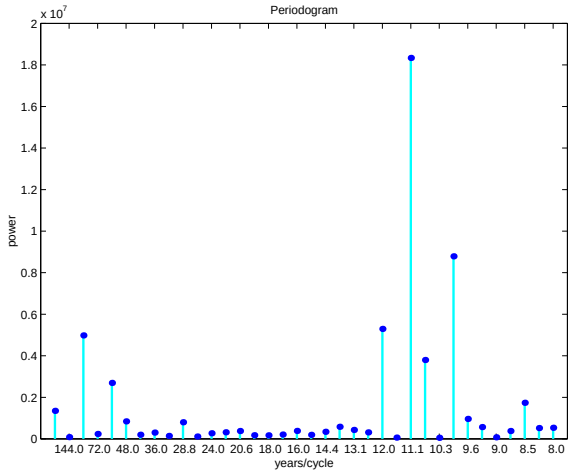


Figure 8.12. *Detail of periodogram shows 11 year cycle*

8.5 Fast Finite Fourier Transform

One-dimensional FFTs with a million points and two-dimensional 1000-by-1000 transforms are common. The key to modern signal and image processing is the ability to do these computations rapidly.

Direct application of the definition

$$Y_{k+1} = \sum_{j=0}^{n-1} \omega^{jk} y_{j+1}, \;\; k = 0, \ldots, n-1$$

requires n multiplications and n additions for each of the n components of Y for a total of $2n^2$ floating-point operations. This does not include the generation of the powers of ω. A computer capable of doing one multiplication and addition every microsecond would require a million seconds, or about 11.5 days, to do a million point FFT.

Several people discovered fast FFT algorithms independently and many people have since contributed to their development, but it was a 1965 paper by John Tukey of Princeton University and John Cooley of IBM Research that is generally credited as the starting point for the modern usage of the FFT.

Modern fast FFT algorithms have computational complexity $O(n \log_2 n)$ instead of $O(n^2)$. If n is a power of 2, a one-dimensional FFT of length n requires less than $3n \log_2 n$ floating-point operations. For $n = 2^{20}$, that's a factor of almost 35,000 faster than $2n^2$. Even if $n = 1024 = 2^{10}$, the factor is about 70.

With MATLAB 6.5 and a 700 MHz Pentium laptop, the time required for `fft(x)` if `length(x)` is $2^{20} = 1048576$ is about one second. The built-in `fft` function is based on FFTW, "The Fastest Fourier Transform in the West," developed at MIT by Matteo Frigo and Steven G. Johnson [1].

The key to the fast FFT algorithms is the double angle formula for trig functions. Using complex notation and

$$\omega = \omega_n = e^{-2\pi i/n} = \cos\delta - i\sin\delta$$

we have

$$\omega_{2n}^2 = \omega_n$$

Written out in terms of separate real and imaginary parts, this is

$$\begin{aligned}\cos 2\delta &= \cos^2\delta - \sin^2\delta \\ \sin 2\delta &= 2\cos\delta\sin\delta\end{aligned}$$

Start with the basic definition.

$$Y_{k+1} = \sum_{j=0}^{n-1} \omega^{jk} y_{j+1},\ k = 0, \ldots, n-1$$

Assume that n is even and that $k \le n/2 - 1$. Divide the sum into terms with even subscripts and terms with odd subscripts.

$$\begin{aligned}Y_{k+1} &= \sum_{even\ j} \omega^{jk} y_{j+1} + \sum_{odd\ j} \omega^{jk} y_{j+1} \\ &= \sum_{j=0}^{n/2-1} \omega^{2jk} y_{2j+1} + \omega^k \sum_{j=0}^{n/2-1} \omega^{2jk} y_{2j+2}\end{aligned}$$

The two sums on the right are components of the FFTs of length $n/2$ of the portions of y with even and odd subscripts. In order to get the entire FFT of length n, we have to do two FFTs of length $n/2$, multiply one of these by powers of ω, and concatenate the results.

The relationship between an FFT of length n and two FFTs of length $n/2$ can be expressed compactly in MATLAB. If `n = length(y)` is even,

```
omega = exp(-2*pi*i/n);
k = (0:n/2-1)';
w = omega .^ k;
u = fft(y(1:2:n-1));
v = w.*fft(y(2:2:n));
```

then

```
fft(y) = [u+v; u-v];
```

Now, if n is not only even, but actually a power of 2, the process can be repeated. The FFT of length n is expressed in terms of two FFTs of length $n/2$, then four FFTs of length $n/4$, then eight FFTs of length $n/8$ and so on until we reach n FFTs of length one. An FFT of length one is just the number itself. If

$n = 2^p$, the number of steps in the recursion is p. There is $O(n)$ work at each step, so the total amount of work is

$$O(np) = O(n \log_2 n)$$

If n is not a power of two, it is still possible to express the FFT of length n in terms of several shorter FFTs. An FFT of length 100 is two FFTs of length 50, or four FFTs of length 25. An FFT of length 25 can be expressed in terms of five FFTs of length five. If n is not a prime number, an FFT of length n can be expressed in terms of FFTs whose lengths divide n. Even if n is prime, it is possible to embed the FFT in another whose length can be factored. We do not go into the details of these algorithms here.

The `fft` function in older versions of MATLAB used fast algorithms if the length was a product of small primes. Beginning with MATLAB 6, the `fft` function uses fast algorithms even if the length is prime. (See [1].)

8.6 ffttx

Our textbook function `ffttx` combines the two basic ideas of this chapter. If n is a power of 2, it uses the $O(n \log_2 n)$ fast algorithm. If n has an odd factor, it uses the fast recursion until it reaches an odd length, then sets up the discrete Fourier matrix and uses matrix-vector multiplication.

```
function y = ffttx(x)
%FFTTX Textbook Fast Finite Fourier Transform.
% FFTTX(X) computes the same finite Fourier transform
% as FFT(X).  The code uses a recursive divide and conquer
% algorithm for even order and matrix-vector multiplication
% for odd order.  If length(X) is m*p where m is odd and
% p is a power of 2, the computational complexity of this
% approach is O(m^2)*O(p*log2(p)).

x = x(:);
n = length(x);
omega = exp(-2*pi*i/n);

if rem(n,2) == 0
   % Recursive divide and conquer
   k = (0:n/2-1)';
   w = omega .^ k;
   u = ffttx(x(1:2:n-1));
   v = w.*ffttx(x(2:2:n));
   y = [u+v; u-v];
else
   % The Fourier matrix.
   j = 0:n-1;
   k = j';
```

```
    F = omega .^ (k*j);
    y = F*x;
end
```

8.7 fftmatrix

The n-by-n matrix F generated by the MATLAB statement

```
F = fft(eye(n,n))
```

is a complex matrix whose elements are powers of the nth root of unity,

$$\omega = e^{-2\pi i/n}$$

The statement

```
plot(fft(eye(n,n)))
```

connects the elements of each column of F and thereby generates a subgraph of the graph on n points. If n is prime, connecting the elements of all columns generates the complete graph on n points. If n is not prime, the sparsity of the graph of all columns is related to the speed of the FFT algorithm. The graphs for $n = 8$, 9, 10, and 11 are shown in figure 8.13.

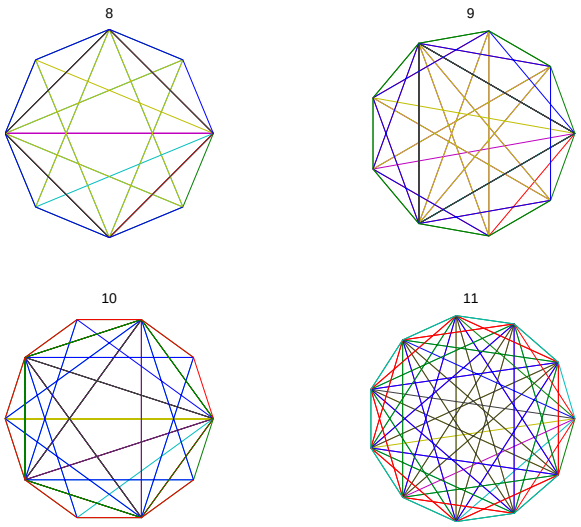


Figure 8.13. *Graphs of FFT matrix*

Because $n = 11$ is prime, the corresponding graph shows all possible connections. But the other three values of n are not prime. Some of the links in their graphs are missing, indicating that the FFT of a vector with that many points can be computed more quickly.

The program `fftmatrix` allows you to investigate these graph.

```
fftmatrix(n)
```

plots all the columns of the FFT matrix of order `n`.

```
fftmatrix(n,j)
```

plots only the `j+1`-st column.

```
fftmatrix
```

defaults to `fftmatrix(10,4)`. In all cases, uicontrols allow `n`, `j` and the choice between one or all columns be changed.

8.8 Other Fourier Transforms and Series

We have been studying the finite Fourier transform, which converts one finite sequence of coefficients into another sequence of the same length, n. The transform is

$$Y_{k+1} = \sum_{j=0}^{n-1} y_{j+1} e^{-2ijk\pi/n}, \ k = 0, \ldots, n-1$$

The inverse transform is

$$y_{j+1} = \frac{1}{n} \sum_{k=0}^{n-1} Y_{k+1} e^{2ijk\pi/n}, \ j = 0, \ldots, n-1$$

The Fourier integral transform converts one complex function into another. The transform is

$$F(\mu) = \int_{-\infty}^{\infty} f(t) e^{-2\pi i \mu t} dt$$

The inverse transform is

$$f(t) = \int_{-\infty}^{\infty} F(\mu) e^{2\pi i \mu t} d\mu$$

The variables t and μ run over the entire real line. If t has units of seconds, then μ has units of radians per second. Both functions $f(t)$ and $F(\mu)$ are complex valued, but in most applications the imaginary part of $f(t)$ is zero.

Alternative units use $\nu = 2\pi\mu$, which has units of cycles or revolutions per second. With this change of variable, there are no factors of 2π in the exponentials, but there are factors of $1/\sqrt{2\pi}$ in front of the integrals, or a single factor of $1/(2\pi)$ in the inverse transform. Maple and the MATLAB Symbolic Toolbox use this alternative notation with the single factor in the inverse transform.

A Fourier series converts a periodic function into an infinite sequence of Fourier coefficients. Let $f(t)$ be the periodic function and let L be its period, so

$$f(t + L) = f(t) \text{ for all } t$$

The Fourier coefficients are given by integrals over the period

$$c_j = \frac{1}{L}\int_{-L/2}^{L/2} f(t)e^{-2\pi ijt}dt,\ j = \ldots, -1, 0, 1, \ldots$$

With these coefficients, the complex form of the Fourier series is

$$f(t) = \sum_{j=-\infty}^{\infty} c_j e^{2\pi ijt/L}$$

A discrete time Fourier transform converts an infinite sequence of data values into a periodic function. Let x_k be the sequence, with the index k taking on all integer values, positive and negative.

The discrete time Fourier transform is the complex valued periodic function

$$X(e^{i\omega}) = \sum_{k=-\infty}^{\infty} x_k e^{ik\omega}$$

The sequence can then be represented

$$x_k = \frac{1}{2\pi}\int_{-\pi}^{\pi} X(e^{i\omega})e^{-ik\omega}d\omega,\ k = \ldots, -1, 0, 1, \ldots$$

The Fourier integral transform involves only integrals. The finite Fourier transform involves only finite sums of coefficients. Fourier series and the discrete time Fourier transform involve both integrals and sequences. It is possible to "morph" any of the transforms into any of the others by taking limits or restricting domains.

Start with a Fourier series. Let L, the length of the period, become infinite and let j/L, the coefficient index scaled by the period length, become a continuous variable, μ. Then the Fourier coefficients c_j become the Fourier transform $F(\mu)$.

Again, start with a Fourier series. Interchanging the roles of the periodic function and the infinite sequence of coefficients leads to the discrete time Fourier transform.

Start with a Fourier series a third time. Now restrict t to a finite number of integral values, k, and restrict j to the same finite number of values. Then the Fourier coefficients become the finite Fourier transform.

In the Fourier integral transform context, Parseval's theorem says

$$\int_{-\infty}^{+\infty} |f(t)|^2 dt = \int_{-\infty}^{+\infty} |F(\mu)|^2 d\mu$$

This quantity is known as the *total power* in a signal.

8.9 Further Reading

VanLoan [4] describes the computational framework for the fast transforms. A page of links at the FFTW Web site [2] provides useful information.

Exercises

8.1. What is the telephone number recorded in `touchtone.mat` and analyzed by `touchtone.m`?

8.2. Modify `touchtone.m` so that it can dial a telephone number specified by an input argument, such as `touchtone('1-800-555-1212')`

8.3. Our version of `touchtone.m` breaks the recording into a fixed number of equally spaced segments, each corresponding to a single digit. Modify `touchtone` so that it automatically determines the number and the possibly disparate lengths of the segments.

8.4. Investigate the use of the MATLAB functions `audiorecorder` and `audioplayer`, or some other system for making digital recordings. Make a recording of a phone number and analyze it with your modified version of `touchtone.m`.

8.5. What relationship between `n` and `j` causes `fftmatrix(n,j)` to produce a five-point star? What relationship produces a regular pentagon?

8.6. *el Niño*. The climatological phenomenon *el Niño* results from changes in atmospheric pressure in the southern Pacific ocean. The "Southern Oscillation Index" is the difference in atmospheric pressure between Easter Island and Darwin, Australia, measured at sea level at the same moment. The text file `elnino.dat` contains values of this index measured on a monthly basis over the 14 year period 1962 through 1975.
Your assignment is to carry out an analysis similar to the sunspot example on the *el Niño* data. The unit of time is one month instead of one year. You should find there is a prominent cycle with a period of 12 months, and a second, less prominent, cycle with a longer period. This second cycle shows up in about three of the Fourier coefficients, so it is hard to measure its length, but see if you can make an estimate.

8.7. Train signal. The MATLAB `demos` directory contains several sound samples. One of them is a train whistle. The statement

```
load train
```

gives you a long vector `y` and a scalar `Fs` whose value is the number of samples per second. The time increment is `1/Fs` seconds.
If your computer has sound capabilities, the statement

```
sound(y,Fs)
```

plays the signal, but you don't need that for this problem.
The data does not have a significant linear trend. There are two pulses of the whistle, but the harmonic content of both pulses is the same.
(a) Plot the data with time in seconds as the independent variable.
(b) Produce a periodogram with frequency in cycles/second as the independent variable.
(c) Identify the frequencies of the six peaks in the periodogram. You should find that ratios between these six frequencies are close to ratios between

small integers. For example, one of the frequencies is 5/3 times another. The frequencies that are integer multiples of other frequencies are *overtones*. How many of the peaks are fundamental frequencies and how many are overtones?

8.8. Bird chirps. Analyze the `chirp` sound sample from the MATLAB `demos` directory. By ignoring a short portion at the end, it is possible to segment the signal into eight pieces of equal length, each containing one chirp. Plot the magnitude of the FFT of each segment. Use `subplot(4,2,k)` for `k = 1:8` and the same axis scaling for all subplots. Frequencies in the range from roughly 400 Hz to 800 Hz are appropriate. You should notice that one or two of the chirps have distinctive plots. If you listen carefully, you should be able to hear the different sounds.

Bibliography

[1] M. Frigo and S. G. Johnson, *FFTW: An adaptive software architecture for the FFT*, Proc. 1998 IEEE Intl. Conf. Acoustics Speech and Signal Processing, 3 (1998), pp. 1381–1384.
`http://www.fftw.org`

[2] M. Frigo and S. G. Johnson, *Links to FFT-related resources.*
`http://www.fftw.org/links.html`

[3] Solar Influences Data Center.
`http://sidc.oma.be`

[4] C. Van Loan, *Computational Frameworks for the Fast Fourier Transform*, SIAM Publications, Philadelphia, PA., 1992, 273 pages.