Difference Equations and Filtering

MATLAB has functions for working with difference equations and filters. These functions operate primarily on vectors.

Vectors are used to hold sampled-data signals, or sequences, for signal processing and data analysis. For multi-input systems, each row of a matrix corresponds to a sample point with each input appearing as columns of the matrix.

The function y =

```
filter(b,a,x)
```

processes the data in vector x with the filter described by vectors a and b, creating filtered data y.

The filter command can be thought of as an efficient implementation of the difference equation. The filter structure is the general tapped delay-line filter described by the difference equation below, where *n* is the index of the current sample, *na* is the order of the polynomial described by vector a and *nb* is the order of the polynomial described by vector b. The output *y*(*n*), is a linear combination of current and previous inputs, *x*(*n*) *x*(*n*-1) ..., and previous outputs, *y*(*n*-1) *y*(*n*-2) ...

$$a(1)\,y(n) = b(1)\,x(n) + b(2)\,x(n-1) + \ldots + b(nb+1)\,x(n-nb)$$
$$- a(2)\,y(n-1) - \ldots - a(na+1)\,y(n-na)$$

Suppose, for example, we want to smooth our traffic count data with a moving average filter to see the average traffic flow over a 4-hour window. This process is represented by the difference equation

$$y(n) = \frac{1}{4}x(n) + \frac{1}{4}x(n-1) + \frac{1}{4}x(n-2) + \frac{1}{4}x(n-3)$$

The corresponding vectors are

```
a = 1; b = [1/4 1/4 1/4 1/4];
```

Note Enter the format command, format rat, to display and enter data using the rational format.

Executing the command `load count.dat` creates the matrix `count` in the workspace.

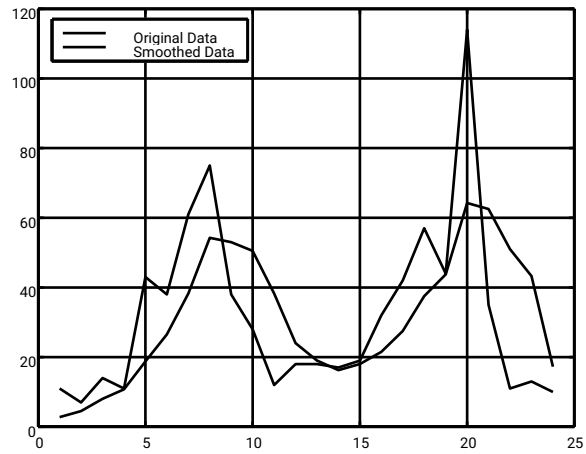
For this example, extract the first column of traffic counts and assign it to the vector `x`.

```
x = count(:,1);
```

The 4-hour moving-average of the data is efficiently calculated with `y = filter(b,a,x);`

Compare the original data and the smoothed data with an overlaid plot of the two curves.

```
t = 1:length(x); plot(t,x,'-.',t,y,'-'), grid on legend('Original Data','Smoothed Data',2)
```

The filtered data represented by the solid line is the 4-hour moving average of the observed traffic count data represented by the dashed line.

For practical filtering applications, the Signal Processing Toolbox includes numerous functions for designing and analyzing filters.

Fourier Analysis and the Fast Fourier Transform (FFT)

Fourier analysis is extremely useful for data analysis, as it breaks down a signal into constituent sinusoids of different frequencies. For sampled vector data, Fourier analysis is performed using the discrete Fourier transform (DFT).

The fast Fourier transform (FFT) is an efficient algorithm for computing the DFT of a sequence; it is not a separate transform. It is particularly useful in areas such as signal and image processing, where its uses range from filtering, convolution, and frequency analysis to power spectrum estimation.

This section:

•Summarizes the Fourier transform functions

•Introduces Fourier transform analysis with an example about sunspot activity

•Calculates magnitude and phase of transformed data

•Discusses the dependence of execution time on length of the transform

Function Summary

MATLAB provides a collection of functions for computing and working with Fourier transforms.

FFT Function Summary

Function	Description
fft	Discrete Fourier transform.
fft2	Two-dimensional discrete Fourier transform.
fftn	N-dimensional discrete Fourier transform.
ifft	Inverse discrete Fourier transform.
ifft2	Two-dimensional inverse discrete Fourier transform.
ifftn	N-dimensional inverse discrete Fourier transform.
abs	Magnitude.

FFT Function Summary (Continued)

Function	Description
angle	Phase angle.
unwrap	Unwrap phase angle in radians.
fftshift	Move zeroth lag to center of spectrum.
cplxpair	Sort numbers into complex conjugate pairs.
nextpow2	Next higher power of two.

Introduction

For length *N* input sequence *x,* the DFT is a length N vector, *X*. fft and ifft implement the relationships

$$X(k) = \sum_{n=1}^{N} x(n)\, e^{-j2\pi(k-1)\left(\frac{n-1}{N}\right)} \qquad 1 \le k \le N$$

$$x(n) = \frac{1}{N} \sum_{k=1}^{N} X(k)\, e^{j2\pi(k-1)\left(\frac{n-1}{N}\right)} \qquad 1 \le n \le N$$

Note Since the first element of a MATLAB vector has an index 1, the summations in the equations above are from 1 to *N*. These produce identical results as traditional Fourier equations with summations from 0 to *N*-1.

If *x(n)* is real, we can rewrite the above equation in terms of a summation of sine and cosine functions with real coefficients

$$x(n) = \frac{1}{N} \sum_{k=1}^{N} a(k)\cos\left(\frac{2\pi(k-1)(n-1)}{N}\right) + b(k)\sin\left(\frac{2\pi(k-1)(n-1)}{N}\right)$$

$k = 1$

where $a\,k(\,) = \text{real}(X\,k(\,))$, $b\,k(\,) = -\text{imag}(X\,k(\,))$, $1 \le n \le$

N

Finding an FFT

The FFT of a column vector x x

= [4 3 7 -9 1 0 0 0]' ; is found with

y = fft(x) which results in

```
y =
6.0000
11.4853 - 2.7574i -2.0000 -
12.0000i
-5.4853 +11.2426i 18.0000
-5.4853 -11.2426i
-2.0000 +12.0000i
11.4853 + 2.7574i
```

Notice that although the sequence x is real, y is complex. The first component of the transformed data is the constant contribution and the fifth element corresponds to the Nyquist frequency. The last three values of y correspond to negative frequencies and, for the real sequence x, they are complex conjugates of three components in the first half of y.

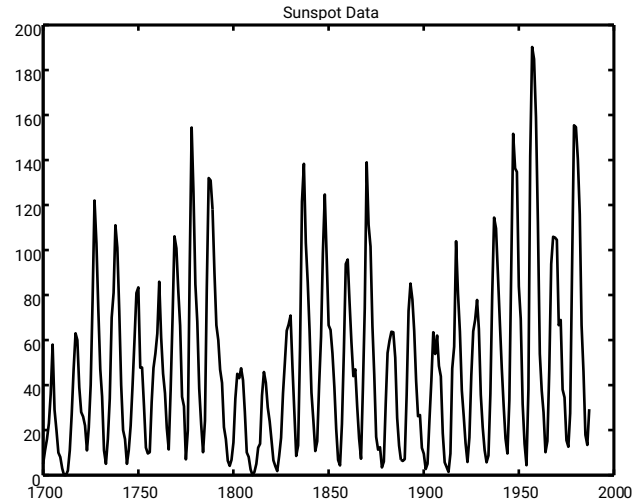
Example: Using FFT to Calculate Sunspot Periodicity

Suppose, we want to analyze the variations in sunspot activity over the last 300 years. You are probably aware that sunspot activity is cyclical, reaching a maximum about every 11 years. Let's confirm that.

Astronomers have tabulated a quantity called the Wolfer number for almost 300 years. This quantity measures both number and size of sunspots.

Load and plot the sunspot data

```
load sunspot.dat year =
sunspot(:,1); wolfer =
sunspot(:,2);
plot(year,wolfer)
title('Sunspot Data')
```

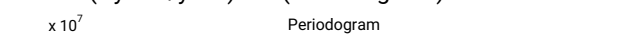


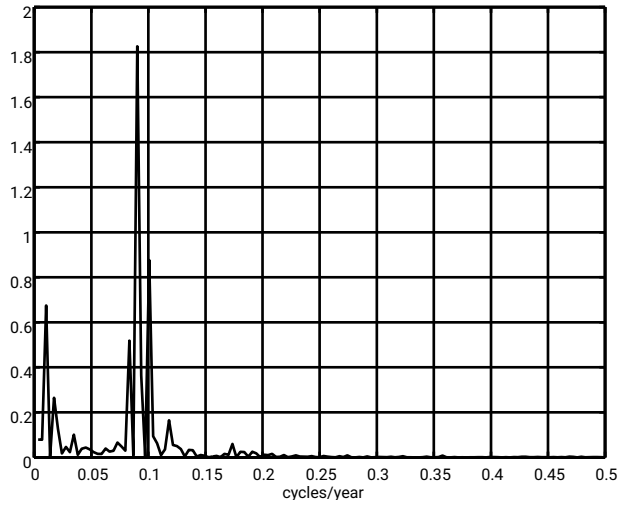
Now take the FFT of the sunspot data.

```
Y = fft(wolfer);
```

The result of this transform is the complex vector, Y. The magnitude of Y squared is called the power and a plot of power versus frequency is a "periodogram." Remove the first component of Y, which is simply the sum of the data, and plot the results.

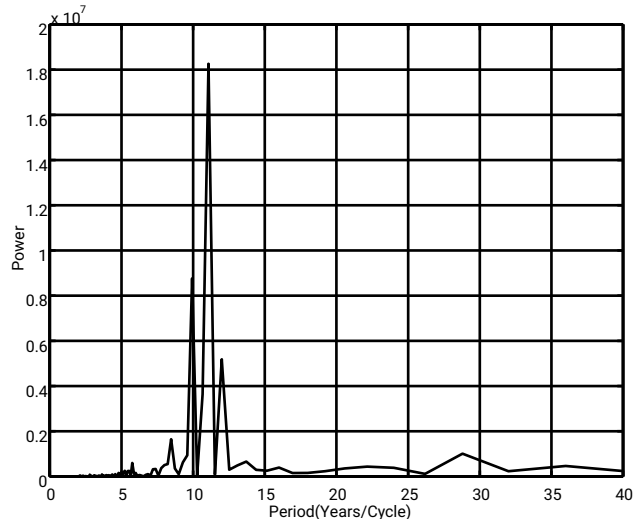
```
N = length(Y); Y(1) = []; power =
abs(Y(1:N/2)).^2; nyquist = 1/2; freq =
(1:N/2)/(N/2)*nyquist;
plot(freq,power), grid on
xlabel('cycles/year') title('Periodogram')
```





The scale in cycles/year is somewhat inconvenient. Let's plot in years/cycle and estimate what one cycle is. For convenience, plot the power versus period (where period = 1./freq) from 0 to 40 years/cycle.

```
period = 1./freq; plot(period,power), axis([0 40 0 2e7]), grid on
ylabel('Power') xlabel('Period(Years/Cycle)')
```



In order to determine the cycle more precisely,

```
[mp,index] = max(power); period(index)

ans =
    11.0769
```

Magnitude and Phase of Transformed Data

Important information about a transformed sequence includes its magnitude and phase. The MATLAB functions abs and angle calculate this information.

To try this, create a time vector t, and use this vector to create a sequence x consisting of two sinusoids at different frequencies.

```
t = 0:1/100:10-1/100; x = sin(2*pi*15*t) +
sin(2*pi*40*t);
```

Now use the fft function to compute the DFT of the sequence. The code below calculates the magnitude and phase of the transformed sequence. It uses the abs function to obtain the magnitude of the data, the angle function to obtain the phase information, and unwrap to remove phase jumps greater than pi to their

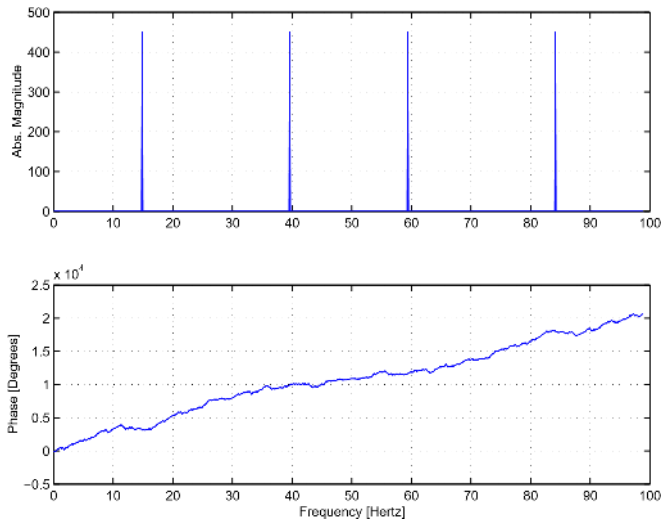
2*pi complement.

```
y = fft(x); m = abs(y); p =
unwrap(angle(y));
```

Now create a frequency vector for the *x*-axis and plot the magnitude and phase.

```
f = (0:length(y)-1)'*100/length(y); subplot(2,1,1),
plot(f,m), ylabel('Abs. Magnitude'), grid on
subplot(2,1,2), plot(f,p*180/pi) ylabel('Phase
[Degrees]'), grid on xlabel('Frequency [Hertz]')
```

The magnitude plot is perfectly symmetrical about the Nyquist frequency of 50 hertz. The useful information in the signal is found in the range 0 to 50 hertz.



FFT Length Versus Speed

You can add a second argument to `fft` to specify a number of points `n` for the transform `y = fft(x,n)`

With this syntax, `fft` pads `x` with zeros if it is shorter than `n`, or truncates it if it is longer than `n`. If you do not specify `n`, `fft` defaults to the length of the input sequence.

The execution time for fft depends on the length of the transform. It is fastest for powers of two. It is almost as fast for lengths that have only small prime factors. It is typically several times slower for lengths that are prime or which have large prime factors.

The inverse FFT function ifft also accepts a transform length argument.

For practical application of the FFT, the Signal Processing Toolbox includes numerous functions for spectral analysi