**fft**
Discrete Fourier transform

**Syntax**

```
Y = fft(X)
Y = fft(X,n)
Y = fft(X,[],dim)
Y = fft(X,n,dim)
```

**Definition**

The functions $Y=\text{fft}(x)$ and $y=\text{ifft}(X)$ implement the transform and inverse transform pair given for vectors of length $N$ by:

$$X(k) = \sum_{j=1}^{N} x(j) \omega_N^{(j-1)k-1}$$

$$x(j) = (1 / N) \sum_{k=1}^{N} X(k) \omega_N^{-(j-1)k-1}$$

where

$$\omega_N = e^{(2\pi i)/N}$$

is an $N$th root of unity.

**Description**

$Y = \text{fft}(X)$ returns the discrete Fourier transform (DFT) of vector $X$, computed with a fast Fourier transform (FFT) algorithm.

If $X$ is a matrix, $\text{fft}$ returns the Fourier transform of each column of the matrix.

If $X$ is a multidimensional array, $\text{fft}$ operates on the first nonsingleton dimension.

$Y = \text{fft}(X, n)$ returns the $n$-point DFT. $\text{fft}(X)$ is equivalent to $\text{fft}(X, n)$ where $n$ is the size of $X$ in the first nonsingleton dimension. If the length of $X$ is less than $n$, $X$ is padded with trailing zeros to length $n$. If the length of $X$ is greater than $n$, the sequence $X$ is truncated. When $X$ is a matrix, the length of the columns are adjusted in the same manner.

$Y = \text{fft}(X, [], \text{dim})$ and $Y = \text{fft}(X, n, \text{dim})$ applies the FFT operation across the dimension $\text{dim}$.

**Examples**

A common use of Fourier transforms is to find the frequency components of a signal buried in a noisy time domain signal. Consider data sampled at 1000 Hz.

Form a signal containing a 50 Hz sinusoid of amplitude 0.7 and 120 Hz sinusoid of
amplitude 1 and corrupt it with some zero−mean random noise:

Fs = 1000;                    % Sampling frequency
T = 1/Fs;                     % Sample time
L = 1000;                     % Length of signal
t = (0:L−1)*T;                % Time vector
% Sum of a 50 Hz sinusoid and a 120 Hz sinusoid
x = 0.7*sin(2*pi*50*t) + sin(2*pi*120*t);
y = x + 2*randn(size(t));     % Sinusoids plus noise
plot(Fs*t(1:50),y(1:50))      % Sinusoids plus noise
title('Signal Corrupted with Zero−Mean Random Noise')
xlabel('time (milliseconds)')

It is difficult to identify the frequency components by looking at the original signal.
Converting to the frequency domain, the discrete Fourier transform of the noisy
signal y is found by taking the fast Fourier transform (FFT):

NFFT = 2^nextpow2(L);       % Next power of 2 from length of y
Y = fft(y,NFFT)/L;
f = Fs/2*linspace(0,1,NFFT/2+1);

% Plot single−sided amplitude spectrum.
plot(f,2*abs(Y(1:NFFT/2+1)))
The main reason the amplitudes are not exactly at 0.7 and 1 is because of the noise. Several executions of this code (including recomputation of \( y \)) will produce different approximations to 0.7 and 1. The other reason is that you have a finite length signal. Increasing \( L \) from 1000 to 10000 in the example above will produce much better approximations on average.

**Algorithm**

The FFT functions (\texttt{fft}, \texttt{fft2}, \texttt{fftn}, \texttt{ifft}, \texttt{ifft2}, \texttt{ifftn}) are based on a library called FFTW\cite{3,4}. To compute an \( N \)-point DFT when \( N \) is composite (that is, when \( N = N_1 N_2 \)), the FFTW library decomposes the problem using the Cooley–Tukey algorithm\cite{1}, which first computes \( N_1 \) transforms of size \( N_2 \), and then computes \( N_2 \) transforms of size \( N_1 \). The decomposition is applied recursively to both the \( N_1 \)- and \( N_2 \)-point DFTs until the problem can be solved using one of several machine–generated fixed–size “codelets.” The codelets in turn use several algorithms in combination, including a variation of Cooley–Tukey\cite{5}, a prime factor algorithm\cite{6}, and a split–radix algorithm\cite{2}. The particular factorization of \( N \) is chosen heuristically.
When \( N \) is a prime number, the FFTW library first decomposes an \( N \)-point problem into three \( (N - 1) \)-point problems using Rader’s algorithm [7]. It then uses the Cooley–Tukey decomposition described above to compute the \( N - 1 \)-point DFTs.

For most \( N \), real-input DFTs require roughly half the computation time of complex-input DFTs. However, when \( N \) has large prime factors, there is little or no speed difference.

The execution time for \texttt{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.

\begin{quote}
\textbf{Note} You might be able to increase the speed of \texttt{fft} using the utility function \texttt{fftw}, which controls the optimization of the algorithm used to compute an FFT of a particular size and dimension.
\end{quote}

\section*{Data Type Support}
\texttt{fft} supports inputs of data types \texttt{double} and \texttt{single}. If you call \texttt{fft} with the syntax \( y = \texttt{fft}(X, \ldots) \), the output \( y \) has the same data type as the input \( X \).

\section*{See Also}
\texttt{fft2, fftn, fftw, fftshift, ifft}
\texttt{dftmtx, filter}, and \texttt{freqz} in the Signal Processing Toolbox

\section*{References}


