The Fourier Integral Transform: The Fourier transform (also called the forward transform, usually transforming a time signal into the frequency domain) or Fourier integral is defined as

$$H(f) = \int_{-\infty}^{\infty} h(t) e^{-j2\pi ft} \, dt \tag{1}$$

where $H(f)$ is the frequency domain representation (transform) of $h(t)$, $f$ is frequency, $t$ is time and $j$ is the imaginary quantity ($j = \sqrt{-1}$). The Fourier transform is a complex quantity

$$H(f) = R(f) + jI(f) = |H(f)| e^{j\theta(f)} \tag{2}$$

Where $R(f)$ and $I(f)$ are the real and imaginary parts of $H(f)$, $|H(f)|$ is the amplitude spectrum and $\theta(f)$ is the phase spectrum or phase angle;

$$|H(f)| = \sqrt{R^2(f) + I^2(f)} \quad \theta(f) = \tan^{-1}[I(f) / R(f)] \tag{3}$$
The inverse Fourier transform is defined as

$$h(t) = \int_{-\infty}^{\infty} H(f) e^{+j2\pi ft} df$$  \hspace{1cm} (4)$$

and shows that a time signal can be synthesized from sinusoids of varying amplitudes and frequencies. Note that

$$e^{-j2\pi ft} = \cos(2\pi ft) - j\sin(2\pi ft)$$ \hspace{1cm} and \hspace{1cm} $$e^{+j2\pi ft} = \cos(2\pi ft) + j\sin(2\pi ft)$$  \hspace{1cm} (5)$$

The Fourier transform pairs (forward and inverse transforms) can be represented compactly as

$$h(t) \leftrightarrow H(f)$$  \hspace{1cm} (6)$$

where the arrow symbol means “has Fourier transform”, and going from left to right is the forward transform (equation 1, with the negative sign in the exponent) and going from right to left is the inverse transform (equation 4, with the positive sign in the exponent).

The inverse transform is similar to the Fourier series that is used to approximate a continuous periodic signal or function. The Fourier transform is normally applied to transient signals or truncated periodic signals. Both the Fourier transform and the Fourier series equations can be converted to summations for discrete (sampled or digital) data and therefore can be calculated using computer algorithms.
**Numerical Fourier Transforms:** For discrete data, the Fourier transform can be calculated numerically by the Discrete Fourier Transform (DFT) in which the forward and inverse transforms are converted to summations. Because of the use of sines and cosines (infinitely long periodic signals) as approximating functions, finite length signals, and the inability to sum to infinity (or use infinite frequency which would require an infinitely small sample interval), the DFT is actually closer to the Fourier series than the Fourier transform. Further, the DFT is almost always implemented with the Fast Fourier Transform (FFT) which is a very computationally efficient algorithm. We will examine the details of the DFT and FFT later. For now, we will illustrate the symmetry properties of the DFT and FFT and show some example calculations.

An example of an FFT calculation and the symmetry properties of the FFT is illustrated in Figure 1. These results were calculated using Matlab code **FFTsymmetry.m** (below). Notice the symmetry of the time series and the FFT coefficients. The FFT symmetry (plots on the left side of Figure 1) is the “standard” symmetry assumed by the FFT algorithm (a legacy of computer codes not being able to utilize negative indices in arrays and loops). The origin-centered symmetry (plots on the right side of Figure 1) can easily be produced by rearranging the arrays. This symmetry is sometimes easier to visualize.
Figure 1. A 5-point boxcar function time series (upper plots) and the FFT of the time series (lower plots) using a 16-point time series and a 16-point FFT calculation (Matlab code `FFTsymmetry.m`).
% Test FFT symmetry, FFTsymmetry.m
x = [1 1 1  zeros(1,11) 1 1]; % 5-point boxcar with FFT symmetry
subplot(2,2,1)
plot(x,'.r','linewidth',2,'markersize',25)
title('5-point boxcar, FFT symmetry, length of array = 16','fontsize',14)
axis([1 16 0 1]);
x1 = fftshift(x); x1 = fliplr(x1);
t1 = [-7:1:8]; subplot(2,2,2)
plot(t1,x1,'.r','linewidth',2,'markersize',25)
title('5-point boxcar, origin-centered symmetry','fontsize',14)
axis([-8 8 0 1]); X = fft(x,16);
rX = real(X); subplot(2,2,3)
plot(rX,'.r','linewidth',2,'markersize',25)
title('FFT of 5-point boxcar, FFT symmetry','fontsize',14)
axis([1 16 -2 6]);
rX1 = fftshift(rX);
rX1 = fliplr(rX1);
t1 = [-7:1:8]; subplot(2,2,4)
plot(t1,rX1,'.r','linewidth',2,'markersize',25)
title('FFT of 5-point boxcar, origin-centered symmetry','fontsize',14)
axis([-8 8 -2 6]);
Fourier Transform Applications: The Fourier transform, in both theoretical and numerical calculations such as with the FFT, is commonly used for analysis, synthesis and signal processing. **Analysis** is determining the frequency content (how much amplitude or power of various frequencies) of a signal. **Synthesis** refers to the use of the Fourier transform (or Fourier series) to synthesize a signal with certain properties using a summation of sines and cosines. **Signal Processing** is using the Fourier transform operations to enhance a signal such as filtering to remove some of the noise that might be included in a signal.

An example of analysis using the Fourier transform is given below using the code *TestFFT.m* (Matlab code shown below) in which a time signal (a Ricker wavelet) is computed (Figure 2) and the FFT of the time signal (the amplitude spectrum) is shown in Figure 3. Notice that the spectrum is smooth, peaks at 5 Hz and is fairly band-limited (significant amplitudes in the spectrum exist over a range of frequencies, in this case about 0 to 15 Hz; so see this range more accurately, we would use a logarithmic amplitude scale for the graph).
Figure 2. A 5-Hz Ricker wavelet (time signal) sampled at 0.01 s.
Figure 3. Amplitude spectrum of the 5-Hz Ricker wavelet shown in Figure 2.
npts = 200;   % Number of points in wavelet
dt = 0.01;    % sample interval (fnyq = 50 Hz)
freq = 5;     % set ~peak frequency of wavelet
timesh = 0.0; % allow timeshift to center the wavelet
nsw = 20;     % apply cosine bell taper to ends of signal
s = ricker(npts,freq,dt,timesh,nsw); % calculate ricker wavelet
t = [0:dt:(npts*dt - dt)];
figure
plot(t,s,'-r','linewidth',1.5)
set(gca,'fontsize',16,'linewidth',2)
xlabel('Time (s)','fontsize',16)
ylabel('Amplitude','fontsize',16)
title('Ricker Wavelet','fontsize',16)

nf = 1024; % set length of fft
S = fft(s,nf); % calculate FFT of wavelet, added zeros (nf = 1024)
% do not change the spectrum (except providing finer sampling, df)
% because the zeros don't contribute to the sum of the area in the
% Fourier integral
fnyq = 1/(2*dt); % Nyquist frequency
df = fnyq/(nf/2); % calculate frequency sample interval
f = [0:df:fnyq]; % calculate freq variable (is (nf/2) + 1 long
SS = S.*conj(S)/nf;
SS = sqrt(SS); % calculate amplitude spectrum
figure
plot(f,SS(1:(nf/2)+1),'-r','linewidth',1.5)
set(gca,'fontsize',16,'linewidth',2)
xlabel('Frequency (Hz)','fontsize',16)
ylabel('Amplitude','fontsize',16)
title('Amplitude Spectrum of Ricker Wavelet','fontsize',16)

function [s] = ricker(npts,freq,dt,timesh,nsw)
% Calculates Ricker wavelet
% s(npts) = wavelet  %  dt = sampling interval
% freq = predominate frequency of wavelet
% timesh = time shift to the left (s)
% nsw = number of points on each side of wavelet to be tapered
%       by half of a cosine bell window
if timesh<.0001; timesh=npts*dt/2; end
pi2 = sqrt(pi)/2;
b = sqrt(6)/(pi*freq);
const = 2*sqrt(6)/b;
for i = 1:npts/
    tim1 = (i-1)*dt;
    tim2 = tim1 - timesh;
    u = const*tim2;
    amp = ((u*u)/4 - 0.5)*pi2*(exp(-u*u/4));
    s(i) = amp;
end
smax = max(abs(s));
smax = smax*2;
s = s/smax;
if nsw ~= 0
    for i = 1:nsw
        j = nsw-i;
        fac = 0.5*(cos(pi*j/nsw) + 1);
        s(i) = s(i)*fac;
    end
    for i = 1:nsw
        j = i-1;
        fac = 0.5*(cos(pi*j/nsw) + 1);
        k = npts-nsw+i;
        s(k) = s(k)*fac;
    end
end

An example of synthesis of a signal is illustrated in Figure 4. In this example, computed using the Matlab code **CosineSummation.m** (shown below), no FFT calculation is performed (although we could perform an equivalent calculation with the FFT); the wavelet is synthesized from 32 cosines (similar to a Fourier series summation). An additional example of cosine summation to calculate wavelets is illustrated in Figure 5.
Figure 4. Summation (synthesis) of 32 cosines (1 Hz to 32 Hz; thin lines) to synthesize a wavelet (bold line) centered at time $t = 0$. The wavelet is plotted at one fourth of its true scale. Also shows that wavelet is composed of equal amounts of signals from 1 to 32 Hz (analysis).
% Calculate summation of cosines to illustrate synthesis
% of wavelet (signal) and analysis of signal (frequency
% content).  % CosineSummation.m  L. Braile 03/25/06

dt = 0.002;
t = [-0.5:dt:0.5];
for I = 1:32
  f = I;
  X(I,:) = cos(2*pi*f*t);
end
x = sum(X);
figure
plot(t,x/4,'r-','linewidth',2)
hold on
for I = 1:32
  plot(t,X(I,:)+2*I,'b-')
end
hold off
axis([-0.5 0.5 -5 66]);
set(gca,'fontsize',14,'linewidth',2)
xlabel('Time (s)','fontsize',14)
ylabel('1 Hz to 32 Hz Cosines','fontsize',14)
title('Summation of Cosines to Generate Wavelet','fontsize',14)
text(-0.42,-3,'Wavelet (summation; bold line) scaled by
0.25','fontsize',14)
Figure 5. Wavelets produced by summation of cosines. Five examples are shown with summations of 2, 4, 8, 16, and 32 cosines. The resulting wavelets are plotted to the right of the individual cosines and are plotted at a reduced amplitude scale. Note that the resulting wavelet becomes more compact with more frequencies (broader bandwidth). (from Yilmaz, 1987)
The Fourier Transform -- Numerical Calculations, Gibbs Phenomenon, Smoothing, Truncation and Signal Processing

**Numerical Fourier Transforms:** For discrete (sampled) data and time (or spatial) series which cannot be represented by simple analytical functions, a numerical Fourier Transform, called the **Discrete Fourier Transform** (DFT) is given by:

\[ X(i) = \sum_{k=0}^{N-1} x(k) e^{-j2\pi ik / N} \quad (1) \]

\[ x(k) = \frac{1}{N} \sum_{i=0}^{N-1} X(i) e^{j2\pi ik / N} \quad (2) \]

where \( x(k) \) is the time series, \( X(i) \) is the array of Fourier coefficients, \( j \) is the imaginary quantity (\( j = \sqrt{-1} \)), and \( i \) and \( k \) are integers that range from 0 to \( N-1 \). Equations (1) and (2) are the DFT forward and inverse transform pairs. These equations assume that the sample interval is one, so the frequency domain coefficients must be assigned frequencies from 0 to twice the Nyquist frequency minus one frequency sample (\( df \)). Again, only \( N/2 +1 \) coefficients in the frequency domain are independent.
Due to the inability to sum to infinity, and the calculation of Fourier coefficients at discrete frequencies, equations (1) and (2) are actually closer to the Fourier series than the integral Fourier transform pair. These restrictions mean that the Fourier coefficients are actually representative of a periodically extended version of \( x(k) \). Also, the Fourier coefficients from equations (1) and (2) can be written as real and imaginary parts and amplitude and phase spectra as shown above.

These DFT equations are almost always calculated on a computer using the Fast Fourier Transform (FFT) algorithm. Because \( i \) and \( k \) in equations (1) and (2) occur as a product and both indices vary from 0 to \( N-1 \), there are many duplicate values of the product. For these duplicates, the exponential values (sines and cosines) only have to be calculated once. Therefore, the computer time savings of the FFT versus the DFT is considerable and increases with the array length \( N \). The number of calculations required for the direct DFT is proportional to \( N^2 \) and for the FFT is proportional to \( N \log_2 N \). Due to the \( N \log_2 N \) relationship, most FFT codes are written to utilize arrays of length \( N \), where \( N \) is a power of 2.

An online tool for investigating the FFT is available at: http://www.dspdimension.com/fftlab/. The website provides an interactive FFT calculator. The interactive page shows a time signal ($f(x)$, real and imaginary parts -- for normal, realistic data, the imaginary parts of the time signal should be zero) and its Fourier Transform ($F(k)$, where $k$ is frequency as a fraction of the Nyquist) with real and imaginary parts. For a real input signal ($f(x)$; no imaginary parts), the FFT coefficients (real and imaginary parts) and the amplitude and phase spectra are calculated using the Fast Fourier Transform algorithm.

**Truncation and Gibbs phenomenon:** The discrete nature of the DFT and the inability to sum to infinity causes an effect known as Gibbs Phenomenon in which side lobes (from $\text{sin } x/x$) are associated with truncations and summation. For example, one can show that the continuous and infinitely long $\text{sin } x/x$ function is the Fourier transform of a boxcar. The width of the boxcar and the $\text{sin } x/x$ (measured by the first zero crossing) are inversely related.
To illustrate the Gibbs effect, we can use the Matlab code `GibbsBoxcar.m` (below) to generate a boxcar function by the FFT of a truncated \((\sin x)/x\). The truncation causes the side lobes (Gibbs Phenomenon) in the boxcar function. Figures 6 to 9 illustrate the Gibbs effect for the FFT of truncated sine waves of total length 17, 33, 65, and 127 points. A 2048-point FFT is used for all calculations (the extra zeros between the two “halves” of the \((\sin x)/x\) are not shown in the “arranged for FFT” version of the time series diagrams in Figures 6 to 9).

Note that no matter how long we make the \((\sin x)/x\), there are always side lobes in the boxcar. Further, notice that although the “sharpness” of the boxcar is greater as the length is increased, the amplitude of the side lobes is the same. Therefore, we cannot eliminate the Gibbs effect by increasing the length. The side lobes can be reduced, but only at the cost of degrading the sharpness of the calculated boxcar.
Figure 6. Eight-point (half width) \((\sin x)/x\) and FFT to generate boxcar. Sample interval = 0.01, Nyquist frequency = 50 Hz. Both FFT-ordered and origin-centered plots are shown.
Figure 7. Sixteen-point (half width) \( (\sin x)/x \) and FFT to generate boxcar. Sample interval = 0.01, Nyquist frequency = 50 Hz. Both FFT-ordered and origin-centered plots are shown.
Figure 8. Thirty-two-point (half width) \((\sin x)/x\) and FFT to generate boxcar. Sample interval = 0.01, Nyquist frequency = 50 Hz. Both FFT-ordered and origin-centered plots are shown.
Figure 9. Sixty-four-point (half width) \((\sin x)/x\) and FFT to generate boxcar. Sample interval = 0.01, Nyquist frequency = 50 Hz. Both FFT-ordered and origin-centered plots are shown.
% Illustrate Gibbs Phenomenon with Boxcar function
% GibbsBoxcar.m L Braile 03/24/06
% Calculate boxcar by FFT of truncated (finite length) \((\sin x)/x\)
% \(dt = 0.01\); % sample interval (s)
n = 8; % number of points in \((\sin x)/x\) [length of \((\sin x)/x\)]
% Try \(n = 8, 16, 32, 64, 128\), to see that Gibbs effect is not
% reduced by using a longer and longer \((\sin x)/x\) time series.
nf = 2048; % number of frequencies if FFT
% calculate \(n\) values of time
f = 15; % frequency of \((\sin x)/x\) signal
x = sin(2*pi*t*f)./(2*pi*t*f); % calculate \((\sin x)/x\)
x1 = fliplr(x(1:(n-1))); % make it symmetric
y = [x1 1 x];
t1 = [-(-n)*dt:dt:n*dt-dt];

% Figure 1
subplot(2,2,1); plot(t1,y,'-b','linewidth',1.5)
set(gca,'fontsize',14,'linewidth',2)
xlabel('Time (s)','fontsize',14)
ylabel('Amplitude','fontsize',14)
title('N-Point \((\sin x)/x\)'),'fontsize',14)
y2 = fftshift(y); % arrange order for FFT
y3 = fliplr(y2);
subplot(2,2,3); plot(t1,y3,'-b','linewidth',1.5)  % Figure 2
set(gca,'fontsize',14,'linewidth',2)
xlabel('Time (s)','fontsize',14)
ylabel('Amplitude','fontsize',14)
title('N-Point (sin x)/x), Arranged for FFT','fontsize',14)
y3e = [y3(1:n) 0 zeros(1,nf-2*n) y3(n+2:2*n)]; % Add zeros to
    % middle and make time series symmetric.
Y3 = fft(y3e,nf); % calculate nf-point FFT of truncated (sin x)/x
rY3 = real(Y3);   % Take real part of Y3 (because of symmetry)
    % all imaginary parts are zero.

% fnyq = 1/(2*dt);
df = fnyq/(nf/2);
f1 = [0:df:2*fnyq-df];
subplot(2,2,2); plot(f1,rY3,'r-','linewidth',1.5)  % Figure 3
set(gca,'fontsize',14,'linewidth',2)
xlabel('Frequency (Hz)','fontsize',14)
ylabel('Relative Amplitude','fontsize',14)
title('FFT of (sin x)/x)','fontsize',14)
rY3n = [rY3((nf/2)+2:nf) rY3(1:(nf/2)+1)];
f2 = [-fnyq+df:df:fnyq];
The Gibbs’ phenomenon and the synthesis of a square wave can also be illustrated with the following code:

```matlab
% Gibbs' square wave code from Matlab Help
% GibbsSquareWave.m
% Building a square wave from the sum of sine waves
t = 0:.02:3.14;
y = zeros(10,length(t));
x = zeros(size(t));
for k=1:2:19
    x = x + sin(k*t)/k;
    y((k+1)/2,:) = x;
end
plot(y(1:1:9,:))
title('The building of a square wave: Gibbs' effect')
```
The building of a square wave: Gibbs' effect

Figure 10. Plot from GibbsSquareWave.m code.
The (Sin x)/x Function: This example shows the (sin x)/x function which is the Fourier transform of a boxcar function (rectangular wave centered at t=0 (Figures 11 and 12).

Figure 11. Fourier transform pairs – the boxcar and “(sin x)/x”, also called the sinc function (from Brigham, 1974).
Figure 12. Fourier transform pairs of boxcars of various widths. Note that there is an inverse relationship between the width in one domain (time or frequency) and width in the other domain (from Brigham, 1974).
Frequency Filtering -- A Signal Processing Operation: In our previous discussion, application and illustrations of Fourier transform operations, we have emphasized analysis and synthesis. We can also use Fourier transform operations to perform signal processing. In these applications, the convolution theorem is often employed to design the process and efficiently implement the calculations using the FFT. An example is frequency filtering in which we filter a signal to remove (actually reduce rather than remove) signal energy in certain frequency ranges in order to enhance the signal in the remaining frequency range. An application of frequency filtering is illustrated schematically in Figure 13. The six-figure diagram showing the time (vertically on the left) and frequency (vertically on the right) domain operations (convolution in the time domain and multiplication in the frequency domain) is a powerful illustration of the relationships between the steps in the signal processing operation. Also, one can see that design of the filter (in this case an “ideal filter”) very simple in the frequency domain. Further, one can see that the filtering operation can be completed by following the bold arrows (FFT the original signal $f(t)$ to get $F(f)$, multiply $F(f)$ by the filter $H(f)$ in the frequency domain to get $G(f)$, then inverse Fourier transform $G(f)$ to get the filtered output in the time domain, $g(t)$). This series of steps requires no direct convolutions and only two FFT operations and one $N$-length array multiplication, where $N$ is the length of the signal.
Figure 13. Time and frequency domain illustrations of frequency filtering.
In practice, smooth filters (no vertical discontinuities or “corners” in the frequency domain representation of the filter) are usually employed. The smoothing can be performed by adding an additional step to the process illustrated in Figure 13 to taper or window the time domain filter ($h(t)$) and then FFT back to the frequency domain to obtain a smooth filter (without a sharp cut-off frequency). Alternatively, many smooth filter operators have been developed and can be calculated in the time or frequency domain. For computational efficiency, the filters are most often applied in the frequency domain and FFT calculations used instead of convolutions or recursive filter operations. Matlab has many efficient and effective filter design and application tools.

In the example codes shown below PltWilberZonlyLow.m and PltWilberZonlyHigh.m, Butterworth band pass filters are applied to enhance the low frequency (Figure 14) and high frequency (Figure 15) components of the original data (a vertical component seismogram).
Magnitude 6.5 earthquake, near coast of central Chile, 29.2934° S, 71.5471° W

Origin time = 17:37:59.0 GMT 1998/09/03, Depth = 27 km
Station = NNA (Nana, Peru, 11.9875° S, 76.8422° W)
Distance = 17.93° (1993 km), Azimuth = 343°

Figure 14. Results of applying a low pass filter using Matlab code PltWilberZonlyLow.m to a seismogram. The original, raw seismogram is shown by the lower trace. The low pass filtered seismogram (actually band pass filtered from 0.01 to 0.1 Hz, equivalent to 10 to 100 s periods) is shown in the upper trace.
Figure 15. Results of applying a high pass filter using Matlab code `PltWilberZonlyHigh.m` to a seismogram. The original, raw seismogram is shown by the lower trace. The high pass filtered seismogram (actually band pass filtered from 0.2 to 5 Hz) is shown in the upper trace. Because the high frequency components of the seismogram are relatively small, the amplitudes of the filtered trace have been multiplied by 10 for plotting.