Computational Science: Computational Methods in Engineering

Applications of the FFT

Outline

• What is a Fourier transform?
• Family of Fourier transforms
• The fast Fourier transform (FFT)
• Calculating Fourier series coefficients
• Calculating frequency plots
• Calculating derivatives using FFT
• Calculating convolutions using FFT
• Generating low frequency noise
• Design of kinoforms
Family of Fourier Transforms

What is a Fourier Transform?

A Fourier transform is essentially a histogram quantifying how much of each frequency makes up a signal.

A Fourier transform is missing time information!!
The Temporal Fourier Transform

The general Fourier transform is defined as

\[ F(s) = \int_{-\infty}^{\infty} f(u) e^{-j2\pi su} du \]

The temporal Fourier transform calculates the frequency content of a time-domain signal.

\[ F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-j\omega t} dt \]

The Spatial Fourier Transform

The general Fourier transform is defined as

\[ F(s) = \int_{-\infty}^{\infty} f(u) e^{-j2\pi su} du \]

The spatial Fourier transform calculates the “spatial waves” comprising a signal.

\[ F(k) = \int_{-\infty}^{\infty} f(x) e^{ikx} dx \]
If a signal is periodic with period $\tau$, the Fourier transform reduces to a Fourier series.

The Fourier coefficients are essentially values of the Fourier transform at discrete frequencies.

$$f(t) = \sum_{k=-\infty}^{\infty} a_k e^{\frac{-j2\pi kt}{\tau}}$$

$$a_k = \frac{1}{\tau} \int_{-\tau/2}^{\tau/2} f(t) e^{-\frac{j2\pi kt}{\tau}} dt$$
The Discrete-Time Fourier Transform (DTFT)

The ordinary Fourier transform of a signal that has been sampled gives what is called a *discrete-time Fourier transform* (DTFT).

\[
F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-j\omega t} dt \\
= \int_{-\infty}^{\infty} \left[ \sum_{n=-\infty}^{\infty} f(n) e^{-j\omega n} \right] dt \\
= \sum_{n=-\infty}^{\infty} \int_{-\infty}^{\infty} f(n) e^{-j\omega n} dt \\
= \sum_{n=-\infty}^{\infty} f(n) e^{-j\omega n}
\]

Alternatively, a DTFT is the reconstruction of a Fourier series done in the frequency domain.

The Discrete Fourier Transform (DFT)

The DTFT for a finite-length sequence \( f(n) \) of length \( N \) is uniquely defined from only \( N \) points in the DTFT. This set of \( N \) points is called the discrete Fourier transform.

\[
F(k) = F(\omega) \bigg|_{\omega = \frac{2\pi k}{N}} = \sum_{n=0}^{N-1} f(n) e^{-\frac{2\pi nk}{N}} \quad k = 0, 1, 2, \ldots, N - 1
\]

A common notation for the DFT is

\[
F(k) = \sum_{n=0}^{N-1} f(n) W_N^{kn} \\
f(n) = \frac{1}{N} \sum_{k=0}^{N-1} F(k) W_N^{-kn} \quad W_N = e^{-j\frac{2\pi}{N}}
\]
Comparison and Visualization of Various Fourier Transforms

The Fast Fourier Transform (FFT)
Fast Fourier Transform (FFT)

For a sequence of 7 points, the DFT is

\[ F(0) = f(0) + f(1) + f(2) + f(3) + f(4) + f(5) + f(6) + f(7) \]
\[ F(1) = f(0) + f(1)W + f(2)W^2 + f(3)W^3 + f(4)W^4 + f(5)W^5 + f(6)W^6 + f(7)W^7 \]
\[ F(2) = f(0) + f(1)W^2 + f(2)W^4 + f(3)W^6 + f(4)W^8 + f(5)W^10 + f(6)W^{12} + f(7)W^{14} \]
\[ F(3) = f(0) + f(1)W^3 + f(2)W^6 + f(3)W^9 + f(4)W^12 + f(5)W^{15} + f(6)W^{18} + f(7)W^{21} \]
\[ F(4) = f(0) + f(1)W^4 + f(2)W^8 + f(3)W^{12} + f(4)W^{16} + f(5)W^{20} + f(6)W^{24} + f(7)W^{28} \]
\[ F(5) = f(0) + f(1)W^5 + f(2)W^10 + f(3)W^{15} + f(4)W^{20} + f(5)W^{25} + f(6)W^{30} + f(7)W^{35} \]
\[ F(6) = f(0) + f(1)W^6 + f(2)W^{12} + f(3)W^{18} + f(4)W^{24} + f(5)W^{30} + f(6)W^{36} + f(7)W^{42} \]

Notice that there are MANY redundant calculations above.

Perhaps there is a more efficient way of calculating a DFT that avoids redundant calculations. This is an FFT?

A fast Fourier transform (FFT) is just an efficient calculation of a DFT.

Scaling of the FFT

Most FFT algorithms scale the data so that you will have to divide by the number of points \( N \) to get physically meaningful data.

\[ H(k) = \frac{1}{N} \text{FFT} \{ h(n) \} \]

\[ H_0 \neq h_{\text{avg}} \]

\[ H_0 = h_{\text{avg}} \]
Frequency Shifting of the FFT

Also, most FFT algorithms output a shifted spectrum. Use fftshift() to correct this.
The Fourier Series

The Fourier series is the Fourier transform of a periodic signal.

**One-Dimensional Standard Fourier Series**

\[ f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \left[ a_n \cos(nt) + b_n \sin(nt) \right] \]

\[ a_n = \frac{1}{\tau} \int_{-\tau/2}^{\tau/2} f(t) \cos(nt) \, dt \quad n \geq 0 \]

\[ b_n = \frac{1}{\tau} \int_{-\tau/2}^{\tau/2} f(t) \sin(nt) \, dt \quad n \geq 1 \]

**One-Dimensional Complex Fourier Series**

\[ f(t) = \sum_{k=-\infty}^{\infty} a_k e^{2 \pi i k t/\tau} \quad a_k = \frac{1}{\tau} \int_{-\tau/2}^{\tau/2} f(t) e^{-2\pi i k t/\tau} \, dt \]

---

Calculating the Coefficients Using FFT

**One-Dimensional Standard Fourier Series**

```matlab
% CALCULATE FOURIER SERIES COEFFICIENTS
F = fft(f)/N;
a0 = F(1);
an = -2*real(F(2:M))
bn = -2*imag(F(2:M))
```

**One-Dimensional Complex Fourier Series**

```matlab
% CALCULATE COMPLEX FOURIER SERIES COEFFICIENTS
F = fftshift(fft(f))/N;
m0 = 1 + floor(N/2);
cn = F(m0-M:m0+M);
```
Calculating Frequency Plots

Nyquist Sampling Theorem

If a signal is periodic with period $\tau$, the maximum possible sampling period and still resolve the signal is

$$\tau_s \leq \frac{\tau}{2}$$

Similarly, if a signal has a maximum frequency of $f_{\text{max}}$, the minimum sampling frequency to resolve the signal is

$$f_s \geq 2f_{\text{max}}$$
Frequency Limits

Given a function $f(n)$ sampled at intervals of $\Delta t$,

(1) the maximum frequency resolved by the FFT is determined by the sampling rate.

$$f_{\text{max}} = \frac{0.5}{\Delta t}$$

(2) the frequency resolution is determined by the number of samples $N$.

$$\Delta f = \frac{2f_{\text{max}}}{N} = \frac{1}{N \cdot \Delta t}$$

Frequency Axis (1 of 2)

The FFT calculates the amplitude of the frequencies over a range of frequencies from $-f_{\text{max}}$ to $+f_{\text{max}}$.

Based on this, calculate the frequency axis according to:

```matlab
fmax = 0.5/dt;
df = 1/(N*dt);
freq = linspace(-fmax,fmax,N);
```

Note: Make $N$ odd!
Frequency Axis (2 of 2)

Padding the FFT to Plot the DTFT
Calculating Derivatives

Property of the Fourier Transform and the DFT

If the derivative of a function is Fourier transformed, the following property applies.

\[ \text{FT} \left\{ \frac{d^a}{dt^a} f(t) \right\} = (j\omega)^a F(\omega) \]

There is a similar property for the DFT

\[ \text{DFT} \left\{ \frac{d^a}{dt^a} f(n) \right\} = (jk)^a F(k) \]

% CALCULATE DERIVATIVE OF ORDER a
a = 2;
k = (2*pi/L)*[-floor(N/2):floor(N/2)];
F = fftshift(fft(f));
F = ((1i*k).^a).*F;
fd = ifft(ifftshift(F));
Visualizing the Calculation ($\alpha = 2$)

This can be used to calculate fractional derivatives. Just let $\alpha$ be a fractional number!

```matlab
% FUNCTION
N = 101;
x = linspace(-1,1,N);
f = exp(-(x/0.2).^2); % Calculate Derivative Using FFT
k = (2*pi/L)*[-floor(Nx/2):floor(Nx/2)];
F = fftshift(fft(f));
F = ((1i*k).^a).*F;
fd = real(ifft(ifftshift(F)));
% Show Derivative
subplot(NA,1,na);
plot(x,fd,'-b');
title(['a = ' num2str(a,'%4.2f')]);
axis tight off;
end
```
Calculating Convolutions

Property of the Fourier Transform and the DFT

If a convolution is Fourier transformed, it becomes a point-by-point multiplication in the Fourier domain.

\[ \text{FT} \{ f(t) * g(t) \} = F(\omega)G(\omega) \]

There is a similar property for the DFT

\[ \text{DFT} \{ f(n) * g(n) \} = F(k)G(k) \]

In fact, this is a “circular convolution.” To ensure this looks like an ordinary convolution, pad the function with enough zeros so that it is equal to or greater than the final convolution length output.
Convolutions for Statistics

Point Distribution Probability Distribution Score Expectation

See Homework #2, Problem #3

% ACCURACY
accx = 3.0 * centimeters;
accy = 5.0 * centimeters;

% THROWING PDF
PDF = (0.5*X/accx).^2 ... + (0.5*Y/accy).^2;
PDF = exp(-PDF);
PDF = PDF / sum(PDF(:));

See Homework #2, Problem #3

% PERFORM CONVOLUTION USING FFT
S = fft2(ifftshift(PDF))*fft2(DB);
S = real(ifft2(S));

% SHOW SCORE EXPECTATION
imagesc(xa,ya,S');
axis equal tight off;
colorbar;

Convolutions for Blurring and Smoothing Data

% FUNCTION
N = 1000;
x = linspace(-1,1,N);
f = zeros(1,N);
n1 = round(0.25*N);
n2 = round(0.75*N);
f(n1:n2) = 1;

% BLUR FUNCTION
r = 0.05;
b = exp(-abs(x/r).^2);
b = b / sum(b(:));

% CONVOLVE
f2 = fft(f).*fft(ifftshift(b));
f2 = real(ifft(f2));
Blurring in Two Dimensions

Calculating Low Frequency Noise
Creating Low Frequency Noise for 1D Functions

% CREATE HIGH FREQUENCY NOISE
N = 1000;
n = (0:N-1);
f = rand(1,N) - 0.5;

% CALCULATE SPECTRUM
F = fft(f);

% FILTER SPECTRUM
n1 = round(0.02*N);
F(n1:N-n1) = 0;

% RECONSTRUCT LOW FREQUENCY NOISE
f2 = real(ifft(F));

Creating Low Frequency Noise for 2D Functions

% CREATE HIGH FREQUENCY NOISE
Nx = 128;
Ny = Nx;
f = rand(Nx,Ny) - 0.5;

% CALCULATE SPECTRUM
F = fft2(f);

% FILTER SPECTRUM
nx = round(0.05*Nx);
ny = round(0.05*Ny);
F(nx:Nx-nx,:) = 0;
F(:,ny:Ny-ny) = 0;

% RECONSTRUCT LOW FREQUENCY NOISE
f2 = real(ifft2(F));
Design of Kinoforms

What is a Kinoform?

A kinoform is a diffraction grating that forms a patterned image when a coherent beam of light is shined through it.
Near-Field to Far-Field

After propagating a long distance, the field within a plane tends toward the Fourier transform of the initial field.

Gerchberg-Saxton Algorithm:

**Initialization**

- **Step 1 – Start with desired far-field image.**
- **Step 2 – Calculate near-field**
- **Step 3 – Replace amplitude with calculated near-field.**
- **Step 4 – Calculate far-field from new amplitude.**

Step 1 – Start with desired far-field image.
Gerchberg-Saxton Algorithm:

Iteration

Step 5 – Replace amplitude with desired image.

Step 6 – Calculate near-field

Step 7 – Replace amplitude

Step 8 – Calculate far-field

This is what the final image will look like.

This is the phase function of the diffractive optical element.

After several dozen iterations...
A surface relief pattern is etched into glass to induce the phase function onto the beam of light. This can also be accomplished with an amplitude mask fabricated in a high-resolution laser printer.