Fast Fourier Transformation (FFT)

Fuyang Tian

Institute for Applied Physics
School of Mathematics and Physics
University of Science and Technology Beijing

May 21, 2018

Outline

Outline

Outline

Fourier Transformation

一种数学变换，有明确的定义和简明的变换形式，作为一种有意义的数学手段在各个科技领域有着广泛的应用，应用FT综合技术，使数值计算、信息光学、电子显微学、图像处理得到了重要进展。

1. 通讯系统：无线电信号是一个带有信息的时间序列，通过傅立叶变换这一数学方法，可把信号大小(幅度)按时间分布的波形(时间域上的分布)转换成按不同的谐波频率的分布(频率域上的分布)，从而获得其频谱特性。
2. 信息光学：傅里叶光学，将二维空间光场分布中，把信号在空间域上的分布转换成频率域上的分布而获得频谱性能。
3. 物理实验的数据处理：通过傅立叶变换建立的实验系统响应函数与传递函数的联系及由此进而形成的传递函数理论，对实验系统性能进行客观评价。

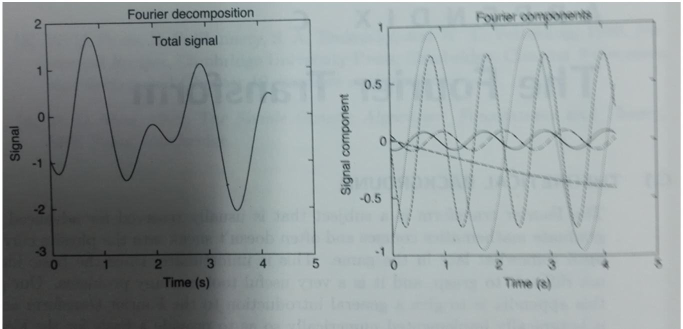


FIGURE C.1: Left: Hypothetical signal; right: individual sine waves whose sum yields the signal at left.

Fourier Transform

Fourier transformation is a very important tool for signal analysis but also helpful to simplify the solution of differential equations or the calculation integrals. We use the symmetric definition of the Fourier transformation:

$$\widetilde{f}(w) = F[f](\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt$$

The inverse Fourier transformation

$$f(t) = F^{-1}[\widetilde{f}](t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \widetilde{f}(\omega) e^{i\omega t} dt$$

decomposes $f(t)$ into a superposition of oscillations. (The definition of Fourier transform pair is not single, the coefficient in the two above equations is random, but their product should be $1/2\pi$.)

The Fourier transform of a convolution integral

$$g(t) = f(t) \otimes h(t) = \int_{-\infty}^{\infty} f(t')h(t-t')dt'$$

becomes a product of Fourier transforms:

$$\begin{aligned}\widetilde{g}(\omega) &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} dt' f(t') e^{-i\omega t} \int_{-\infty}^{\infty} h(t-t') e^{-\omega(t-t')} d(t-t') \\ &= \sqrt{2\pi}\widetilde{f}(\omega)\widetilde{h}(\omega).\end{aligned}$$

A periodic function with $f(t+T) = f(t)$ is transformed into a Fourier series

$$\begin{aligned} f(t) &= \sum_{n=-\infty}^{\infty} e^{iw_n t} \widehat{f}(\omega_n) \quad \text{with} \quad \omega_n = n\frac{2\pi}{T} \\ \widehat{f}(\omega_n) &= \frac{1}{T}\int_0^T f(t) e^{-i\omega_n t} dt.\end{aligned}$$

two-dimension Fourier transform

When u is a two-dimension function, k_x and k_y are the vector of wave in the direction of x and y, respectively.

$$\begin{aligned} \hat{\hat{u}} &= F[u(x,y)] \\ u(x,y) &= F^{-1}[\hat{\hat{u}}(k_x,k_y)] \end{aligned}$$

In matlab, for $[u_1,\cdots,u_j,\cdots,u_N]$, the Fourier transform can be defined as

$$\begin{aligned} \hat{u_k} &= \sum_{j=1}^{N} u_j e^{\frac{-2\pi(j-1)(k-1)i}{N}}, k=1,\cdots,N \\ u_j &= \frac{1}{N}\sum_{k=1}^{N} \hat{u_k} e^{\frac{2\pi(j-1)(k-1)i}{N}}, j=1,\cdots,N \end{aligned}$$

Note that the two above equations suggest the periodic conditions in these equations.

三种变换

- 傅立叶变换，适用于周期信号，把非周期信号看成周期T趋于无穷的周期信号；原信号必须绝对可积；
- 拉普拉斯变换，傅立叶变换的推广，相当于带有一个指数收敛因子的傅立叶变换，把频域推广到复频域；
- Z变换,离散时间傅立叶变换。

FFT in Matlab

1. fft(u), ifft: one-dimension FFT, u is a vector quantity. fft(u,[],2), fft(u.')'. It is very important that results from FFT are often the frequency, which is not from small to large ones. the first half is not negative frequency, the second half is negative frequency.
2. fft2(u), ifft2: two-dimension FFT.
3. fftshift, ifftshift: order the results of fft, if u is a vector quantity, it orders the left-right (up-down) section, if u is a matrix, it orders the first-third (second-forth) quadrants. fftshift(u,1) and fftshift(u,2) for up-down section and left-right section in a matrix.

$$F[sinc^2(ax) = \frac{1}{\sqrt{2\pi a^2}} tri(\frac{k}{2\pi a})]$$

FFT for differential and integral

For F[u'(x)], via the fourier transformation and the subsection integral, we can get

$$F[u'(x)] = \int_{-\infty}^{+\infty} u'(x)e^{-ikx}dx = u(x)e^{-ikx}|_{-\infty}^{+\infty} - \int_{-\infty}^{+\infty} u(x)(-ik)e^{-ikx}dx$$

when $|x| \to \infty$, $u(x) \to 0$,

$$F[u'(x)] = ik\int_{-\infty}^{+\infty} u'(x)e^{-ikx}dx = ikF[u(x)]$$

Similarly,

$$F[u^{(n)}(x)] = (ik)^n F[u(x)]$$

of which, $u^{(n)}(x)$ is the n-order derivative of u(x).
函数的求导运算在傅立叶变换作用下，可转化为相对简单的代数运算，

$$u^{(n)}(x) = F^{-1}ik^n F[u(x)]$$

由此，利用FT将偏微分程空域或时域上求导运算简化为频域上的代数运算，求解后再通过FT的逆变换得到空域或时域上的结果。在程序方面，可利用fft, ifft的功能。

FT谱方法

偏微分方程的形式为

$$\frac{\partial^n u}{\partial t^n} = Lu + N(u),$$

其中, L代表线性算符，N(u)为非线性项。对于初始条件为$u(x,t_0)$,在周期性边界条件下求某一时刻的u(x,t),可通过变量代换办法将等号左边对t的n阶导数降为1 阶。
在x域上对

$$\frac{\partial^n u}{\partial t^n} = Lu + N(u),$$

进行FT变换得

$$\frac{\partial^n \hat{u}}{\partial t^n} = \alpha(k)\hat{u} + F[N(u)],$$

其中$\hat{u}(k,t)$表示$u(x,t)$在x域上的FT变换。
注意方程中包括着周期性边界条件，被视为$u_{mN+j} = u_j$。

Discrete Fourier Transformation

By introducing a grid of N equidistant points, we divide the time interval $0 \leq t < T$,

$$t_n = n\Delta t = n\frac{T}{N} \text{ with } n = 0, 1, \cdots, N-1.$$

The function values (samples)

$$f_n = f(t_n)$$

are arranged as components of a vector. With respect to the orthonormal basis

$$f = \begin{pmatrix} f_0 \\ \vdots \\ f_{N-1} \end{pmatrix}; e_n = \begin{pmatrix} \delta_{0,n} \\ \vdots \\ \delta_{N-1,n} \end{pmatrix}$$

f is expressed as a linear combination

$$f = \sum_{n=0}^{N-1} f_n e_n$$

The discrete Fourier transformation is the transformation to an orthogonal base in frequency space

$$e_{\omega_j} = \sum_{n=0}^{N-1} f_n e_n = \begin{pmatrix} 1 \\ e^{i\frac{2\pi}{N}j} \\ \vdots \\ e^{\mathrm{i}\frac{2\pi}{\mathrm{N}}\mathrm{j}(\mathrm{N}-1)} \end{pmatrix}$$

with

$$\omega_j = \frac{2\pi}{T} j.$$

These vectors are orthogonal:

$$
\begin{aligned}
e_{\omega_j} e^*_{\omega'_j} &= \sum_{n=0}^{N-1} e^{i(j-j')\frac{2\pi}{N}n} = \begin{cases} \frac{1-e^{i(j-j')2\pi}}{1-e^{i(j-j')2\pi/N}} = 0 & \text{for } j-j' \neq 0 \\ N & \text{for } j-j' = 0 \end{cases} \\
e_{\omega_j} e^*_{\omega'_j} &= N\delta_{j,j'}
\end{aligned}
$$

Alternatively a real-valued basis can be defined:

$$
\cos(\frac{2\pi}{N}jn) \quad j = 0, 1, \cdots, j_{\max}
$$

$$
\sin(\frac{2\pi}{N}jn) \quad j = 1, 2, \cdots, j_{\max}
$$

$$
j_{\max} = \frac{N}{2}(\text{even } N) \quad j_{\max} = \frac{N-1}{2}(\text{odd } N).
$$

Trigonometric interpolation

The last equation can be interperted as an interpolation of the function $f(t)$ at the sampling points t_n by a linear combination of trigonometric functions:

$$f(t) = \frac{1}{N} \sum_{j=0}^{N-1} \widetilde{f}_{\omega_j} (e^{i\frac{2\pi}{T}t})^j,$$

which is a polynomial of

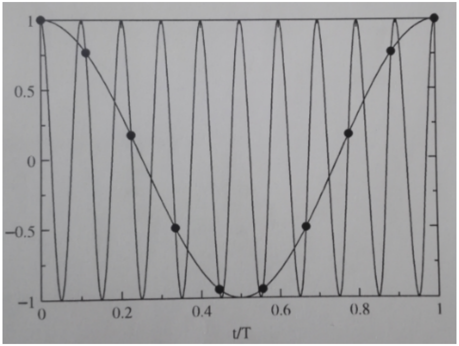
$$q = e^{i\frac{2\pi}{T}t}.$$

Since

$$e^{-i\omega_j t_n} = e^{-i\frac{2\pi}{N}jn} = e^{i\frac{2\pi}{N}(N-j)n} = e^{i\omega_{N-j}t_n},$$

the frequencies ω_j and ω_{N-j} are equivalent

$$\widetilde{f}_{\omega_{N-j}} = \sum_{n=0}^{N-1} f_n e^{-i\frac{2\pi}{N}(N-j)n} = \sum_{n=0}^{N-1} f_n e^{i\frac{2\pi}{N}jn} = \widetilde{f}_{\omega_{-j}}.$$



Equivalence of ω_1 and ω_{N-1}. The two functions $\cos\omega t$ and $\cos(N-1)\omega t$ have the same values at the sample points t_n but are very different in between.

Real-Valued Functions

For a real-valued function

$$f_n = f_n^*$$

and hence

$$\widetilde{f}^*_{\omega_{N-j}} = (\sum_{n=0}^{N-1} f_n e^{-i\omega_j t_n})^* = \sum_{n=0}^{N-1} f_n e^{i\omega_j t_n} = \widetilde{f}_{\omega_{-j}}.$$

Here it is sufficient to calculate the sums for $j = 0, \cdots, N/2$.

Approximate Continuous Fourier Transformation

We continue function $f(t)$ periodically be setting

$$f_N = f_0$$

and write

$$\widetilde{f}_{\omega_j} = \sum_{n=0}^{N-1} f_n e^{-i\omega_j n} = \frac{1}{2} f_0 + e^{-i\omega_j} f_1 + \cdots + e^{-\omega_j (N-1)} f_{N-1} + \frac{1}{2} f_N.$$

Comparing with the trapezoidal rule for the integral

$$\int_0^T e^{-i\omega_j t} f(t) dt = \frac{T}{N} (\frac{1}{2} e^{-i\omega_j^0} f(0) + e^{-i\omega_j \frac{T}{N}} f(\frac{T}{B})$$
$$+ \cdots + e^{-i\omega_j \frac{T}{N}(N-1)} f(\frac{T}{N})(N-1) + \frac{1}{2} f(T))$$

we find

$$\hat{f}(\omega_j) = \frac{1}{T}\int_0^T e^{-i\omega_j t f(t) dt} \approx \frac{1}{N}\tilde{f}_{\omega_j}$$

which shows that the discrete Fourier transformation is an approximation to the Fourier series of a periodic function with period T which coincides with $f(t)$ in the interval $0 < t < T$. The range of the integral can be formally extended to $\pm\infty$ by introducing a windowing function

$$W(t) = \begin{cases} 1 & \text{for } 0 < t < T \\ 0 & \text{else} \end{cases}$$

The discrete Fourier transformation approximates the continuous Fourier transformation but windowing leads to a broadening of the spectrum. For practical purposes smoothers windowing functions are used like a triangular window or one of the following equations.

Algorithms

Straightforward evaluation of the sum

$$\widetilde{f}_{\omega_j} = \sum_{n=0}^{N-1} \cos(\frac{2\pi}{N} jn) f_n + i \sin(\frac{2\pi}{N} jn) f_n$$

needs $O(N^2)$ additions, multiplications, and trigonometric functions.

1. $Goertzel's \quad Algorithm$
2. $Fast \quad Fourier \quad Transformation$

FFT

In 1965, J.W. Cooley and John Tukey proposed the Cooley-Tukey FFT method; 为离散傅立叶变换的快速算法，使算法的复杂度由原来的O(N^2)变为O(Nlog$N$).

Forward Discrete Fourier Transform (DFT)

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i2\pi kn/N}$$

Inverse Discrete Fourier Transform (IDFT)

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{i2\pi kn/N}$$

Fast Fourier Transformation

If the number of samples is $N = 2^p$, the Fourier transformation can be performed very efficiently by this method. The phase factor

$$e^{-i\frac{2\pi}{N}jm} = W_N^{jm}$$

can take only N different values. The number of trigonometric functions can be reduced by the sum. Starting from a sum with N samples

$$F_N(f_0 \cdots f_{N-1}) = \sum_{n=0}^{N-1} f_n W_N^{jn},$$

we separate even and odd powers of the unit root

$$\begin{aligned}
F_N(f_0 \cdots f_{N-1}) &= \sum_{m=0}^{\frac{N}{2}-1} f_{2m} W_N^{j(2m)} + \sum_{m=0}^{\frac{N}{2}-1} f_{2m+1} W_N^{j(2m+1)} \\
&= \sum_{m=0}^{\frac{N}{2}-1} f_{2m} e^{-i\frac{2\pi}{N/2}jm} + W_N^j \sum_{m=0}^{\frac{N}{2}-1} f_{2m+1} e^{-i\frac{2\pi}{N/2}jm} \\
&= F_{N/2}(f_0, f_2, \cdots, f_{N-2}) + W_N^j F_{N/2}(f_1, f_3, \cdots, f_{N-1}).
\end{aligned}$$

This division is repeated until only sums with one summand remain:

$$F_1(f_n) = f_n.$$

For example, consider the case $N = 8$:

$$\begin{aligned}
F_8(f_0 \cdots f_7) &= F_4(f_0 f_2 f_4 f_6) + W_8^j F_4(f_1 f_3 f_5 f_7) \\
\cdots\cdots & \\
F_4(f_0 f_2 f_4 f_6) &= F_2(f_0 f_4) + W_4^j F_2(f_2 f_6) \\
F_4(f_1 f_3 f_5 f_7) &= F_2(f_1 f_5) + W_4^j F_2(f_3 f_7) \\
\cdots\cdots &
\end{aligned}$$

$$\cdots\cdots$$

$$\begin{aligned}
F_2(f_0 f_4) &= f_0 + W_2^j f_4 \\
F_2(f_2 f_6) &= f_2 + W_2^j f_6 \\
F_2(f_1 f_5) &= f_1 + W_2^j f_5 \\
F_2(f_3 f_7) &= f_3 + W_2^j f_7
\end{aligned}$$

Expansion gives

$$\begin{aligned}
F_8 &= f_0 + W_2^j f_4 + W_4^j f_2 + W_4^j W_2^j f_6 \\
&\quad + W_8^j f_1 + W_8^j W_2^j f_5 + W_8^j W_4^j f_3 + W_8^j W_4^j W_2^j f_7.
\end{aligned}$$

Generally a summand of the Fourier sum can be written using the binary representation of n

$$n = \sum l_i l_i = 1, 2, 4, 8, \cdots$$

in the following way:

$$f_n e^{-i\frac{2\pi}{N} jn} = f_n e^{-i\frac{2\pi}{N}(l_1+l_2+\cdots)j} = f_n W_{N/l_1}^j W_{N/l_2}^j.$$

The function values are reordered according to the following algorithm:

1. conut from 0 to $N-1$ using binary numbers $m = 000001, 010, \cdots$
2. bit reversal gives the binary numbers $n = 000, 100, 010, \cdots$
3. store f_n at the position m. This will be denoted as $s_m = f_n$

An example for N=8 the function values are in the order $(s_0 \quad s_1 \quad s_2 \quad s_3 \quad s_4 \quad s_5 \quad s_6 \quad s_7)' = (f_0 \quad f_4 \quad f_2 \quad f_6 \quad f_1 \quad f_5 \quad f_3 \quad f_7)'$
Now we can calculate sums with two summands. Since W_2^j can take only two different values

$$W_2^j = \begin{cases} 1 & \text{for } j = 0, 2, 4, 6 \\ -1 & \text{for } j = 1, 3, 5, 7 \end{cases}$$

a total of eight sums have to be calculated which can be stored again in the same workspace:

$$\begin{pmatrix} f_0 + f_4 \\ f_0 - f_4 \\ f_2 + f_6 \\ f_2 - f_6 \\ f_1 + f_5 \\ f_1 - f_5 \\ f_3 + f_7 \\ f_3 - f_7 \end{pmatrix} = \begin{pmatrix} s_0 + W_2^0 s_1 \\ s_0 + W_2^1 s_1 \\ s_2 + W_2^2 s_3 \\ s_2 + W_2^3 s_3 \\ s_4 + W_2^4 s_5 \\ s_4 + W_2^5 s_5 \\ s_6 + W_2^6 s_7 \\ s_6 + W_2^7 s_7 \end{pmatrix}$$

Next calculate sums with four summands. W_4^j can take one of four values

$$W_4^j = \begin{cases} 1 & \text{for } j = 0, 4 \\ -1 & \text{for } j = 2, 6 \\ W_4 & \text{for } j = 1, 5 \\ -W_4 & \text{for } j = 3, 7 \end{cases}$$

The following combinations are needed:

$$\begin{pmatrix} f_0+f_4+(f_2+f_6) \\ f_0+f_4-(f_2+f_6) \\ (f_0-f_4)+W_4(f_2-f_6) \\ (f_0-f_4)-W_4(f_2-f_6) \\ f_1+f_5+(f_3+f_7) \\ f_1+f_5-(f_3+f_7) \\ (f_1-f_5)+W_4(f_3-f_7) \\ (f_1-f_5)-W_4(f_3-f_7) \end{pmatrix} = \begin{pmatrix} s_0+W_4^0 s_2 \\ s_1+W_4^1 s_3 \\ s_0+W_4^2 s_2 \\ s_1+W_4^3 s_3 \\ s_4+W_4^4 s_6 \\ s_5+W_4^5 s_7 \\ s_4+W_4^6 s_6 \\ s_5+W_4^7 s_7 \end{pmatrix}$$

The next step gives the sums with eight summand. With

$$W_8^j = \begin{cases} 1 & j=0 \\ W_8 & j=1 \\ W_8^2 & j=2 \\ W_8^3 & j=3 \\ -1 & j=4 \\ -W_8 & j=5 \\ -W_8^2 & j=6 \end{cases}$$

we calculate

$$\begin{pmatrix} f_0+f_4+(f_2+f_6)+(f_1+f_5+(f_3+f_7)) \\ f_0+f_4-(f_2+f_6)+w_8(f_1+f_5-(f_3+f_7)) \\ (f_0-f_4)+w_4(f_2-f_6)+w_8^2(f_1-f_5)\pm w_4(f_3-f_7)) \\ (f_0-f_4)+w_4(f_2-f_6)+w_8^3(f_1-f_5)\pm w_4(f_3-f_7)) \\ f_0+f_4+(f_2+f_6)-(f_1+f_5+(f_3+f_7)) \\ f_0+f_4+(f_2+f_6)-w_8(f_1+f_5-(f_3+f_7)) \\ (f_0-f_4)+w_4(f_2-f_6)+w_8^2(f_1-f_5)\pm w_4(f_3-f_7)) \\ (f_0-f_4)+w_4(f_2-f_6)+w_8^3(f_1-f_5)\pm w_4(f_3-f_7)) \end{pmatrix} = \begin{pmatrix} s_0+w_8^0 s_4 \\ s_1+w_8^1 s_5 \\ s_0+w_8^2 s_6 \\ s_1+w_8^3 s_7 \\ s_4+w_8^4 s_4 \\ s_5+w_8^5 s_5 \\ s_4+w_8^6 s_6 \\ s_5+w_8^7 s_7 \end{pmatrix}$$

which is the final result.

Brief proceed

The following shows a simple fast Fourier transformation algorithm. The number of trigonometric function evaluations can be reduced but this reduces the readability. At the beginning Data[k] are the input data in bit-reversed order.

```
size:=2
first:0
while first<Number-of-Samples do begin
for n:=0 to size/2-1 do begin
j:=first+n;
k:=j+size/2-1
T:=exp($-2 * Pi * i * n/$Number-of-Samples)$* Data[k]$
Data[j]:=Data[j]+T
Data[k]:Data[k]-T    end
first:=first$*2$
size:=size$*2$
end
```