FFT 法による数値ラプラス逆変換

I. ラプラス変換とフーリエ変換の関係

ラプラス変換対及びフーリエ変換対はそれぞれ次式で定義される．

$$\text{ラプラス変換}\quad Y(p)=\int_0^{\infty} y(t)\cdot e^{-pt}dt \;\Leftrightarrow\; y(t)=\frac{1}{2\pi i}\int_{\gamma-i\infty}^{\gamma+i\infty} Y(p)\cdot e^{pt}dp \qquad (1)$$

$$\text{フーリエ変換}\quad Y(\omega)=\int_{-\infty}^{\infty} y(t)\cdot e^{-i2\pi\omega t}dt \;\Leftrightarrow\; y(t)=\int_{-\infty}^{\infty} Y(\omega)\cdot e^{i2\pi\omega t}d\omega \qquad (2)$$

ただし，$i^2=-1$ ，p：**複素数**，γ,ω：**実数**

これらは，実変数(ここでは，時間)の関数 $y(t)$ を複素変数あるいは別の実変数(周波数)の関数 $Y(p),Y(\omega)$ に積分変換したもので，形は異なるがいずれも同じ物理現象を表す．

ラプラス逆変換式を以下のように変数変換する．

$$p=\gamma+i2\pi\omega \;\text{，}\; dp=i2\pi\mathrm{d}\omega \;\text{，}\; p=\gamma-i\infty\rightarrow\gamma+i\infty \;\Rightarrow\; \omega=-\infty\rightarrow\infty$$

$$\therefore\quad \begin{aligned} y(t)&=\int_{-\infty}^{\infty} Y(\gamma;\omega)e^{\gamma t}e^{i2\pi\omega t}d\omega \\ &=e^{\gamma t}\cdot\int_{-\infty}^{\infty} Y(\gamma;\omega)e^{i2\pi\omega t}d\omega \end{aligned} \qquad (3)$$

すなわち，ラプラス変換 $Y(p)$ がわかっていれば，$Y(\gamma;\omega)\equiv Y(\gamma+i2\pi\omega)$ のフーリエ逆変換に $\exp(\gamma t)$ をかければ元の関数 $y(t)$ が得られる．FFT法による数値ラプラス変換法では，このフーリエ逆変換をFFT法を用いて行う．

II. フーリエ変換と離散フーリエ変換DFTの関係

A. 離散フーリエ変換DFT

フーリエ変換対

$$F(\omega)=\int_{-\infty}^{\infty} f(t)e^{-i2\pi\omega t}dt\Leftrightarrow f(t)=\int_{-\infty}^{\infty} F(\omega)e^{i2\pi\omega t}d\omega \qquad (4)$$

に対し，

$$F_k=\sum_{n=0}^{N-1} f_{\mathrm{n}}e^{-i2\pi nk/N}\Leftrightarrow f_k=\frac{1}{N}\sum_{n=0}^{N-1} F_{\mathrm{n}}e^{i2\pi nk/N} \qquad (5)$$

［付1)］

を離散フーリエ変換対(DFT)と呼び，式(5)の積和を高速に実行する計算方法がFFT法である．ここで，フーリエ変換は連続関数の積分変換であるが，DFTは関数とは無関係であり f_k や F_k は単なる数値であり，計算も単純な積和である．

定義式より次のような関係が直ちに導ける．

○ $F_{k\pm m\cdot N}=F_k$ ，$f_{k\pm m\cdot N}=f_k$　　m：整数 (6)

したがって，F_k ，f_k の内，異なる値を持つのは高々 N 個．

○ $F_{N-k} = \sum_{sm=0}^{N-1} f_n e^{-i2\pi(N-k)n/N} = \sum_{n=0}^{N-1} f_n e^{-i2\pi(-k)n/N} = F_{-k}$ (7)

すなわち，F_k は $k=-N/2+1,-N/2+2,\cdots,0,1,2,\cdots,N/2$ での値であると考えられ，このことはフーリエ変換との対応を考えるときに重要となる．

○ 逆変換式において

$$f_{N-k} = \frac{1}{N}\sum_{n=0}^{N-1} F_n e^{i2\pi(N-k)n/N} = \frac{1}{N}\sum_{n=0}^{N-1} F_n e^{-i2\pi kn/N} = f_{-k} \quad (8)$$

すなわち，逆変換式は定数倍を除いて変換式と同じ形である．したがって，計算機プログラムでは変換と逆変換とで同一のアルゴリズムが使用可能である．ただし，k 番目として得られるのは，N-k 番目の値に対応し順序が逆になる．

なお，f_n が実数のとき（通常の物理量）は，

$F_{-k} = \overline{F_k}$ （$\overline{F_k}$ は F_k の共役複素数）

したがって，一般に

$F_{N-k} = \overline{F_k}$ ，$k = 1,2,\cdots N/2-1$ (9)

B. フーリエ変換と離散フーリエ変換の関係

式(5)の f_k を，関数 $f(t)$ の区間 $0 \le t < T$，サンプリング間隔 T/N でのサンプルデータとすれば，次の関係が成立する．

関数 $f(t)$ のフーリエ変換を $F(\omega)$ とするとき，

$f_k = f(kT/N)$ ，$k = 0,1,\cdots,N-1$ (10)

の式(5)による離散フーリエ変換 F_k と $F(\omega)$ の関係は次式となる．

$$F_k = \frac{N}{T}\sum_{n=-\infty}^{\infty} F'(\frac{k}{T} - \frac{nN}{T}) \approx \begin{cases} \frac{N}{T}F(\frac{k}{T}) & , \quad k=0,1,\cdots,N/2 \\ \frac{N}{T}F(\frac{k-N}{T}) = \overline{F(\frac{N-k}{T})} & , \quad k=N/2+1,\cdots,N-1 \end{cases} \quad (11)$$ ［付2)］

ただし，$F'(\omega) = F(\omega) * W(\omega) = \int_{-\infty}^{\infty} F(\omega-\tau)W(\tau)d\tau$

$$W(\omega) = \int_0^{\infty} w(t)e^{-i2\pi\omega t}dt = \int_0^{T} e^{-i2\pi\omega t}dt \quad , \quad w(t) = \begin{cases} 1 & 0 \le t \le T \\ 0 & \text{それ以外} \end{cases}$$

式(11)によれば f_k を離散フーリエ変換したものは $F(\omega)$ を $-\Delta\omega N/2 < \omega \le \Delta\omega N/2$ の区間，周波数刻み $\Delta\omega = 1/T$ でサンプリングしたものの近似値であることがわかる．

なお，$w(t)$ は $f(t)$ を $0 < t < T$ に打ち切るための窓関数であり，$f(t) = 0$ ，$t < 0$ or $t \le T$ の場合は

$F'(\omega)=F(\omega)$ となる．さらに，$F(\omega)=0$ ，$|\omega|>N/2T$ の場合は $\sum_{-\infty}^{\infty}F(\mathrm{k}/T-\mathrm{n}N/T)=F(k/T)$ となり，上式の近似は等号になる．

以上時間域から周波数域への変換を基準に考えたが，逆に周波数域での関数 $F(\omega)$ がわかっていれば，式(11)のデータを準備することによって時間域の関数 $f(t)$ を求める事ができる．その際，データを近似式によって準備することは，言い換えれば，変換域で次の矩形ウインドウ関数をかけて逆変換する事に相当する．

$$\hat{W}_R(\omega)=\begin{cases}1 \ , & -N/2T<\omega<N/2T \\ 0 \ , & \omega\le -N/2T \ \text{ or } \ \omega\ge N/2T\end{cases} \qquad (12)$$

そして，時間域ではこれの逆変換を畳み込んだものが得られることになる．N/T を十分大きくとれば，この逆変換はデルタ関数に近づき時間域での値は $f(x)$ に近づく事になる．このことから，逆変換がデルタ関数に近いウインドウ関数を使用すれば，よりよい結果が得られることになる．そのような関数として次のハニングウインドウ関数が知られている．

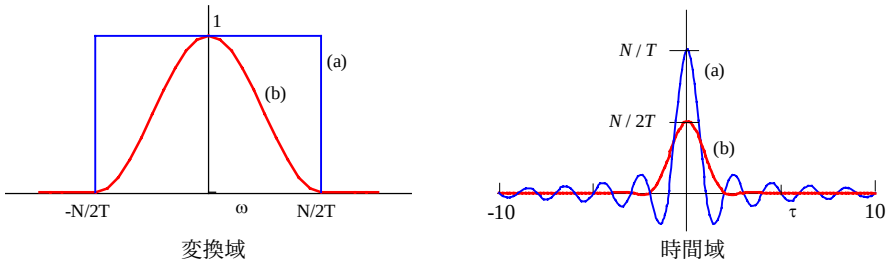
$$\hat{W}_h(\omega)=\begin{cases}\dfrac{1}{2}\left(1+\cos\left(2\pi\omega T/N\right)\right) & , \quad -N/2T\le\omega\le N/2T \\ 0 & , \quad \omega\le -N/2T \ \text{ or } \ \omega\ge N/2T\end{cases} \qquad (13)$$

これらのウインドウ関数の逆変換は，それぞれ次のようになる．

$$w_r(\tau)=\frac{N}{T}\cdot\frac{\sin(\pi\tau)}{\pi\tau}$$

$$w_h(\tau)=-\frac{N}{T}\frac{\sin(\pi\tau)}{\pi\tau}\cdot\frac{1}{2\left(\tau^2-1\right)}=-\frac{w_r(\tau)}{2(\tau^2-1)}$$

ここで，$\tau=tN/T$



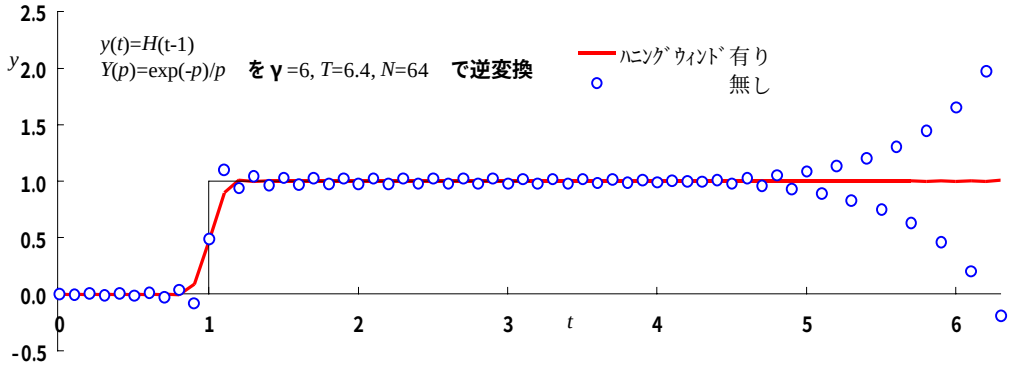
矩形ウインドウとハニングウインドウ

図に二つののウインドウ関数を示す．ハニングウインドウの時間域での形状は矩形ウインドウに較べて滑らかであり畳み込みによるリップルの発生が抑制されることがわかる．ただし，$\tau=0$ での値は矩形ウインドウの半分で横のひろがりも大きいので，極めて急峻な立ち上がりについては誤差がが大きくなる．

ハニングウインドウを使用する場合, 式(11)の代わりに, 変換域において準備すべきデータは次式となる．

$$F_k = \begin{cases} \dfrac{N}{T} F\left(k/T\right) \dfrac{1+\cos\left(2\pi k/N\right)}{2} & , \quad k = 0,1,\cdots,N/2 \\ \dfrac{N}{T} F\left((k-N)/T\right) \dfrac{1+\cos\left(2\pi(k-N)k/N\right)}{2} = \overline{F_{N-k}} & , \quad k = N/2+1,\cdots,N-1 \end{cases} \tag{11'}$$

下図はステップ関数の逆変換についてハニングウインドウの効果を示したものであり，ハニングウインドを使用することで，立ち上がり初期のオーバーシュート及び後半での発散が抑制されていることがわかる．



ステップ関数の逆変換例

III. FFT法による数値ラプラス逆変換手順

以上のことをまとめれば，ラプラス変換域での関数$Y(p)$が与えられたとき，これを逆変換する手順は以下のようである． ［付3)］

[1] 解が必要な時間幅と，間隔からT, Nを決める．

ただし，FFT法の都合により， Nは2のべき乗とする． Nが大きいほど細かい結果が得られ計算精度もよくなるから，計算時間と記憶容量が許す限りなるべく大きい値を選ぶ方がよい．

[2] γの値を決める．

一般にγを大きくすれば精度はよくなると考えられるが，後で$e^{\gamma t}$をかけるのでtが大きいところで解が発散する．場合によっては，無駄な計算時間が増えるが，T, N, γを大きくしてtが小さいところの値のみを採用することも考えられる．理論的に妥当な値を決める事は難しいが，経験上からは，$\gamma = 5/T \sim 7/T$の値が適当である．

[3] $p = \gamma + i2\pi k/T$ と置き，次のデータ列をFFT法計算プログラムにしたがって準備する．

$$Y_k = \frac{N}{T} Y(\gamma + \frac{i2\pi k}{T}) \cdot \frac{1}{2}\left\{1+\cos\frac{2\pi k}{N}\right\} \quad , \quad k = 0,1,\cdots N/2$$

$$Y_k = \overline{Y_{N-k}} \quad , \quad k = N/2+1,\cdots,N-1$$

ただし$\overline{Y_{N-k}}$はY_{N-k}の共役複素数

注）定数N/T倍については使用するFFTの計算ルーチンにもよる．ここでは式(11)の定義とする．

[4] FFT逆変換プログラムを実行する．

普通FFT法の計算プログラムでは，データは配列に格納されるので[3]で計算したY_kを順にデータ配列に格納し，プログラムを実行する．そうすると，同じデータ配列に時間域での値が時刻 $t = kT/N$ ， $k = 0,1,\cdots,N\text{-}1$ の順に得られる．

これを$\tilde{y}_k$とする．

[5] 最後に[4]で得られた結果に$e^{\gamma t}$をかければ， $y(t)$が得られる．

$$y(kT/N) \approx \tilde{y}_k \cdot e^{\gamma T k/N} \quad , \quad k = 0,1,\cdots,N\text{-}1$$

IV. FFT法

式(5)の積和を高速に実行する計算方法がFFT法である．以下にFFT法を説明する．

$$\mathrm{W}[x] = e^{-\mathrm{i}\,2\pi x/N} \tag{14}$$

とおいて，DFT変換式を再記すれば,

$$F_{\mathrm{k}} = \sum_{\mathrm{n}=0}^{N-1} f_{\mathrm{n}}\,\mathrm{W}[\mathrm{k\,n}] \ , \ \mathrm{k} = 0,1,\cdots,N\text{-}1 \tag{15}$$

そして,

$$\begin{aligned} &\mathrm{W}[\mathrm{a}+\mathrm{b}] = \mathrm{W}[\mathrm{a}]\cdot\mathrm{W}[\mathrm{b}] \\ &\mathrm{W}[N+\mathrm{a}] = \mathrm{W}[\mathrm{a}] \\ &\mathrm{W}[N/2+\mathrm{a}] = -\mathrm{W}[\mathrm{a}] \end{aligned} \tag{16}$$

今， $\mathrm{N} = 2^{\mathrm{p}}$ （pは整数）の場合を考えk,nを２進数で表す.

$$\mathrm{n} = 2^{\mathrm{p}-1}\mathrm{n}_{\mathrm{p}-1}+\cdots+2\mathrm{n}_1+\mathrm{n}_0 \ , \ \mathrm{k} = 2^{\mathrm{p}-1}\mathrm{k}_{\mathrm{p}-1}+\cdots+2\mathrm{k}_1+\mathrm{k}_0 \tag{17}$$

ただし　$\mathrm{k}_0,\cdots,\mathrm{k}_{\mathrm{p}-1},\mathrm{n}_0,\cdots,\mathrm{n}_{\mathrm{p}-1} = 0 \text{ or } 1$

そうすれば

$$F(\mathrm{k}_{\mathrm{p}-1},\cdots,\mathrm{k}_0) = \sum_{\mathrm{n}_0=0}^{1}\cdots\sum_{\mathrm{n}_{\mathrm{p}-1}=0}^{1} f(\mathrm{n}_{\mathrm{p}-1},\cdots,\mathrm{n}_0)\,\mathrm{W}[(2^{\mathrm{p}-1}\mathrm{k}_{\mathrm{p}-1}+\cdots+2\mathrm{k}_1+\mathrm{k}_0)(2^{\mathrm{p}-1}\mathrm{n}_{\mathrm{p}-1}+\cdots+2\mathrm{n}_1+\mathrm{n}_0)]$$

$$\begin{aligned} &\mathrm{W}[(2^{\mathrm{p}-1}\mathrm{k}_{\mathrm{p}-1}+\cdots+2\mathrm{k}_1+\mathrm{k}_0)(2^{\mathrm{p}-1}\mathrm{n}_{\mathrm{p}-1}+\cdots+2\mathrm{n}_1+\mathrm{n}_0)] \\ &\quad = \mathrm{W}[\mathrm{k}_0(2^{\mathrm{p}-1}\mathrm{n}_{\mathrm{p}-1}+\cdots+2\mathrm{n}_1+\mathrm{n}_0)+2\mathrm{k}_1(2^{\mathrm{p}-2}\mathrm{n}_{\mathrm{p}-2}+\cdots+2\mathrm{n}_1+\mathrm{n}_0)+\cdots+2^{\mathrm{p}-2}\mathrm{k}_{\mathrm{p}-2}(2\mathrm{n}_1+\mathrm{n}_0)+2^{\mathrm{p}-1}\mathrm{k}_{\mathrm{p}-1}\mathrm{n}_0] \\ &\quad = \mathrm{W}_{\mathrm{p}-1}\cdot\mathrm{W}_{\mathrm{p}-2}\cdots\mathrm{W}_{\mathrm{p}-\mathrm{m}}\cdots\mathrm{W}_0 \end{aligned}$$

ただし， $\mathrm{W}_{\mathrm{p}-\mathrm{m}} = \mathrm{W}[2^{\mathrm{m}-1}\mathrm{k}_{\mathrm{m}-1}(2^{\mathrm{p}-\mathrm{m}}\mathrm{n}_{\mathrm{p}-\mathrm{m}}+\cdots+\mathrm{n}_0)]$ (18)

$$\therefore \quad F(\mathrm{k}_{\mathrm{p}-1},\cdots,\mathrm{k}_0) = \sum_{\mathrm{n}_0=0}^{1}\cdots\sum_{\mathrm{n}_{\mathrm{p}-1}=0}^{1} f(\mathrm{n}_{\mathrm{p}-1},\cdots,\mathrm{n}_0)\,\mathrm{W}_{\mathrm{p}-1}\cdots\mathrm{W}_0 \tag{19}$$

（注 $f_{\mathrm{n}},F_{\mathrm{k}}$ 等の添字を２進数で表す場合は $f(),F()$ の様に表す事にする.）

上の積和を内側から順に行い,

$$\sum_{n_{p-1}=0}^{1} f(n_{p-1},\cdots,n_0)W_{p-1} = \sum_{n_{p-1}=0}^{1} f(n_{p-1},\cdots,n_0)W[k_0(2^{p-1}n_{p-1}+\cdots+2n_1+n_0)] \Rightarrow {}^1f(k_0,n_{p-2},\ldots,n_0)$$

$$\sum_{n_{p-2}=0}^{1} {}^1f(k_0,n_{p-2},\ldots,n_0)W_{p-2} = \sum_{n_{p-2}=0}^{1} {}^1f(k_0,n_{p-2},\ldots,n_0)W[2k_1(2^{p-2}n_{p-2}+\cdots+2n_1+n_0)] \Rightarrow {}^2f(k_0,k_1,n_{p-3},\ldots,n_0)$$

……

$$\sum_{n_{p-m}=0}^{1} {}^{m-1}f(k_0,\ldots,k_{m-2},n_{p-m},\ldots,n_0)W_{p-m} = \sum_{n_{p-m}=0}^{1} {}^{m-1}f(k_0,\ldots,k_{m-2},n_{p-m},\ldots,n_0)W[2^{m-1}k_{m-1}(2^{p-m}n_{p-m}+\cdots+n_0)] \quad (20)$$

$$\Rightarrow {}^mf(k_0,\ldots,k_{m-1},n_{p-m-1},\ldots,n_0)$$

……

$$\sum_{n_1=0}^{1} {}^{p-2}f(k_0,\ldots,k_{p-3},n_1,n_0)W_1 = \sum_{n_1=0}^{1} {}^{p-2}f(k_0,\ldots,k_{p-3},n_1,n_0)W[2^{p-1}k_{p-2}(2n_1+n_0)] \Rightarrow {}^{p-1}f(k_0,\ldots,k_{p-2},n_0)$$

と順次置換えて行けば，p回目の置換えで

$$\sum_{n_0=0}^{1} {}^{p-1}f(k_0,\ldots,k_{p-2},n_0)W_0 = \sum_{n_0=0}^{1} {}^{p-1}f(k_0,\ldots,k_{p-2},n_0)W[2^{p-1}k_{p-1}n_0] \Rightarrow {}^pf(k_0,\ldots,k_{p-1}) \equiv F(k_{p-1},\ldots,k_0) \quad (21)$$

となり，F_k が全て求まる．

第m段目の積和計算を再記すれば，$f(n_{p-1},\cdots,n_0) \equiv {}^0f(n_{p-1},\cdots,n_0)$ とおいて一般に

$${}^mf(k_0,\ldots,k_{m-1},n_{p-m-1},\ldots,n_0) = \sum_{n_{p-m}=0}^{1} {}^{m-1}f(k_0,\ldots,k_{m-2},n_{p-m},\ldots,n_0)W[2^{m-1}k_{m-1}(2^{p-m}n_{p-m}+\cdots+n_0)] \quad (22)$$

$m = 1,\cdots,p$

で与えられるが，この計算は

$$\begin{aligned} K &= 2^{p-1}k_0+\cdots+2^{p-m+1}k_{m-2} \quad \rightarrow \quad K = 0,2N/2^m,4N/2^m,\cdots,N/2 \\ I &= 2^{p-1-m}n_{p-1-m}+\cdots+n_0 \quad \rightarrow \quad I = 0,1,\cdots,N/2^m-1 \end{aligned} \quad (23)$$

と置いて

$${}^mf(K+\frac{N}{2^m}k_{m-1}+I) = \sum_{n_{p-m}=0}^{1} {}^{m-1}f(K+\frac{N}{2^m}n_{p-m}+I)W[2^{m-1}k_{m-1}(\frac{N}{2^m}n_{p-m}+I)] \ , \ k_{m-1} = 0 \text{ or } 1$$

すなわち次の２組の積和計算の繰返しである．

$${}^mf(K+\frac{N}{2^m}\cdot 0+I) = \sum_{n_{p-m}=0}^{1} {}^{m-1}f(K+\frac{N}{2^m}n_{p-m}+I)W[2^{m-1}\cdot 0\cdot(\frac{N}{2^m}n_{p-m}+I)] \quad \text{より}$$

$$\boxed{{}^mf(K+I) = {}^{m-1}f(K+I)+{}^{m-1}f(K+\frac{N}{2^m}+I) \quad \because W[0]=1} \quad (24)$$

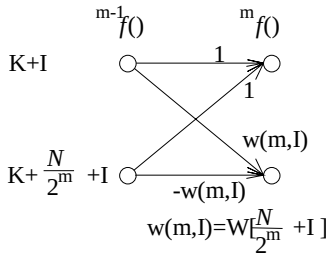
$${}^mf(K+\frac{N}{2^m}\cdot 1+I) = \sum_{n_{p-m}=0}^{1} {}^{m-1}f(K+\frac{N}{2^m}n_{p-m}+I)W[2^{m-1}\cdot(\frac{N}{2^m}n_{p-m}+I)] \quad \text{より}$$

$$
{}^{m}f(K+\frac{N}{2^{m}}+I)={}^{m-1}f(K+I)W[2^{m-1}\cdot I]+{}^{m-1}f(K+\frac{N}{2^{m}}+I)W[\frac{N}{2}+2^{m-1}\cdot I]
$$

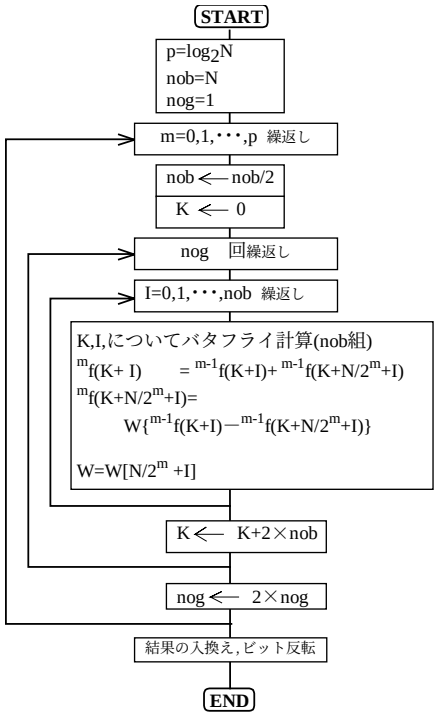
$$
=\left\{{}^{m-1}f(K+I)-{}^{m-1}f(K+\frac{N}{2^{m}}+I)\right\}W[2^{m-1}\cdot I]\quad \because W[\frac{N}{2}+a]=-W[a]
$$

(25)

式(24), (25)の計算を図示すると次の左の図のようになる.



バタフライ計算



FFT法概略フローチャート

すなわち, ${}^{m-1}f()$ から ${}^{m}f()$ を作成する第m段目の積和計算では $K+I$ 番目の要素と $K+N/2^{m}+I$ 番目の要素のみが関係する. 上の図が蝶々に似ているのでこれをバタフライ計算と呼ぶ. 第m段目の計算はこのバタフライ計算を $I=0,1,\cdots,N/2^{m}-1$ の $N/2^{m}$ 回行うグループを $K=0,2N/2^{m},4N/2^{m},\cdots,N/2$ の 2^{m-1} グループ, 合計 $2^{m-1}\times N/2^{m}=N/2$ 回行えば終了する. そして, この様な段階を $p=\log_2 N$ 回行えば全ての計算が終了する. なお, 各段階におけるこのようなグループの数は最初は1で以後2倍ずつ増え, 逆に1グループ毎のバタフライ計算の回数は最初 $N/2$ 回で以後1/2倍づつ減っていく. ［付4)］

全過程で必要な積の回数は $N\cdot\log_2 N$ 回となる. これに対してDFT変換式をそのまま計算する場合に必要な積の回数は N^2 回であるから, 両者の比は $N/\log_2 N$ となり, $N=1024$ の場合は約102倍の短縮効果がある.

付　録

付 1) (5)式が変換対になることの証明は次のようである．

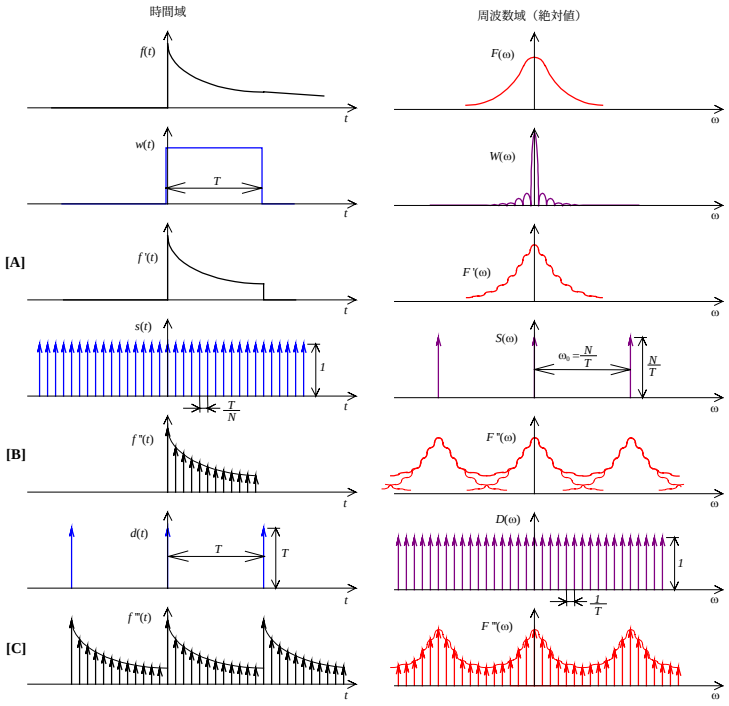
$$g_k = \frac{1}{N}\sum_{n=0}^{N-1} F_n e^{i2\pi nk/N} = \frac{1}{N}\sum_{n=0}^{N-1}\left(\sum_{m=0}^{N-1} f_m e^{-i2\pi mn/T}\right)e^{i2\pi nk/N} = \frac{1}{N}\sum_{m=0}^{N-1} f_m\left(\sum_{n=0}^{N-1} e^{-i2\pi mn/T} e^{i2\pi nk/N}\right)$$

$$= \frac{1}{N}\sum_{m=0}^{N-1} f_m\left(\sum_{n=0}^{N-1} e^{-i2\pi(m-k)n/T}\right) = \frac{1}{N}\sum_{m=0}^{N-1} f_m\left(N\delta_{mk}\right) = f_k$$

δ_{ij} はクロネッカーのデルタ．

すなわち，f_k の変換 F_k を逆変換したものは f_k となる

付 2) 式(11)の導出（下図参照）



関数のフーリエ変換とそのサンプリング値の離散フーリエ変換の関係

[A] $f(t)$ を $0 \le t < T$ で打ち切るために，時間域でウインド関数 $w(t)$ をかける．

$$f'(t) = f(t)w(t) \qquad [1]$$

時間域でのかけ算は変換域では畳み込み積分であるから，このフーリエ変換は

$$F'(\omega) = F(\omega) * W(\omega) \qquad [2]$$

$W(\omega)$ は一般にサイドローブを有するので，$F'(\omega)$ は，$F(\omega)$ にリップルが乗ったものとなる．

[B]次に $f(t)$ を T/N 間隔でサンプリングするためにインパルス列

$$s(t) = \sum_{n=-\infty}^{\infty} \delta\left(t - \frac{nT}{N}\right) \qquad [3]$$

を時間域でかける．

$$f''(t) = f(t)w(t)s(t) = \sum_{n=-\infty}^{\infty} f'_n \delta(t - \frac{nT}{N}) \qquad [4]$$

ただし　$f'_n = f'(\frac{nT}{N})$

$s(t)$ のフーリエ変換は

$$S(\omega) = \frac{N}{T} \sum_{n=-\infty}^{\infty} \delta(\omega - \frac{nN}{T}) \qquad [5]$$

で与えられ，時間域でのかけ算は周波数域では畳み込み積分になるから，$f''(t)$ のフーリエ変換は

$$F''(\omega) = \frac{N}{T} \sum_{n=-\infty}^{\infty} F'(\omega - \frac{nN}{T}) \qquad [6]$$

すなわち，$NF'(\omega)/T$ を，N/T 間隔に平行移動させたものを全て足し合わせたものとなる．

このことから，$F''(\omega)$ は $\omega_0 = N/T$ 間隔の周期関数となり，しかも，

$$F''(\omega) = F''(\omega - \omega_0) \quad \frac{\omega_0}{2} < \omega \le \omega_0 \qquad [7]$$

であることがわかる．すなわち，$\omega > \omega_0/2$ の値は $\omega < 0$ の領域の値である．

また，結果には $\omega > \omega_0/2$ の値が順次折り返して加わる，いわゆる折り返しの影響が含まれる事がわかるであろう．そして，$F(\omega) = 0, \omega > \omega_0/2$ ならばこの折り返しの影響は現れないこともわかる．

[C]最後に，周波数域での結果に

$$D(\omega) = \frac{1}{T} \sum_{k=-\infty}^{\infty} \delta(\omega - \frac{k}{T}) \qquad [8]$$

をかけ，$1/T$ 間隔でサンプリングする．すなわち，

$$F'''(\omega) = D(\omega)F''(\omega) = \frac{1}{T} \sum_{k=-\infty}^{\infty} \delta(\omega - \frac{k}{T}) \cdot \frac{N}{T} \sum_{n=-\infty}^{\infty} F'(\omega - \frac{nN}{T})$$

$$\therefore F'''(\omega) = \frac{N}{T^2} \sum_{k=-\infty}^{\infty} \left\{ \sum_{n=-\infty}^{\infty} F'(\frac{k - nN}{T}) \right\} \delta(\omega - \frac{k}{T}) \qquad [9]$$

$D(\omega)$ のフーリエ逆変換は，

$$d(t) = \sum_{k=-\infty}^{\infty} \delta(t - kT) \qquad [10]$$

周波数域でのかけ算は時間域では畳み込み積分であるから，$F'''(\omega)$ は，次式のフーリエ変換である．

$$f'''(t) = d(t) * f''(t) = \sum_{k=-\infty}^{\infty} f''(t - kT)$$
$$= \sum_{k=-\infty}^{\infty} \sum_{n=0}^{N-1} f'_n \cdot \delta(t - \frac{nT}{N} - kT) \qquad [11]$$

以上をまとめれば，$f(t)$ をサンプリングし周期 T の関数に置換えた

$$f'''(t) = \sum_{k=-\infty}^{\infty} \sum_{n=0}^{N-1} f'_n \cdot \delta(t - \frac{nT}{N} - kT) \qquad [12]$$

ただし，　$f'_n = f(\frac{nT}{N}) w(\frac{nT}{N})$

のフーリエ変換は，

$$F'''(\omega)=\frac{N}{T^2}\sum_{k=-\infty}^{\infty}\left\{\sum_{n=-\infty}^{\infty}F'(\frac{k-nN}{T})\right\}\delta(\omega-\frac{k}{T}) \qquad [13]$$

ただし，　$F'(\omega)=F(\omega)*W(\omega)$

となる事がわかった．

[D]　一方，$f'''(t)$ は周期関数であるのでこれのフーリエ変換は以下のように計算できる．

周期Tの周期関数 $g(t)$ のフーリエ変換は次式で与えられる．注）

$$G(\omega)=\sum_{k=-\infty}^{\infty}G_k\,\delta(\omega-\frac{k}{T})\ ,\ G_k=\frac{1}{T}\int_0^T g(t)e^{-i2\pi kt/T}\,dt \qquad [14]$$

そこで，上式第２式の積分を $f'''(t)$ について行えば

$$G_k=\frac{1}{T}\int_0^T f'''(t)e^{-i2\pi kt/T}\,dt=\frac{1}{T}\int_0^T\sum_{m=-\infty}^{\infty}\sum_{n=0}^{N-1}f'_n\cdot\delta(t-mT-\frac{nT}{N})e^{-i2\pi kt/T}\,dt$$

$$=\frac{1}{T}\int_0^T\sum_{n=0}^{N-1}f'_n\cdot\delta(t-\frac{nT}{N})e^{-i2\pi kt/T}\,dt=\frac{1}{T}\sum_{n=0}^{N-1}f'_n\cdot e^{-i2\pi k(nT/N)/T}$$

$$=\frac{1}{T}\sum_{n=0}^{N-1}f'_n\cdot e^{-i2\pi knT/N}$$

となり，$f'''(t)$ のフーリエ変換は次式のようにも表せる．

$$F'''(\omega)=\frac{1}{T}\sum_{k=-\infty}^{\infty}\left\{\sum_{n=0}^{N-1}f'_n\cdot e^{-i2\pi knT/N}\right\}\delta(\omega-\frac{k}{T}) \qquad [15]$$

したがって，式[13] 右辺と式[15] 右辺を比較して，

$$\sum_{n=0}^{N-1}f'_n\cdot e^{-i2\pi knT/N}=\frac{N}{T}\sum_{n=-\infty}^{\infty}F'(\frac{k-nN}{T}) \qquad [16]$$

そして，ウィンド関数を

$$w(t)=1\ ,\ 0\le t<T \qquad [17]$$

に選べば，$f'_n=f_n$ となり，次式が成立する．

$$F_k\equiv\sum_{n=0}^{N-1}f_n\cdot e^{-i2\pi knT/N}=\frac{N}{T}\sum_{n=-\infty}^{\infty}F'(\frac{k-nN}{T})$$

注）周期関数 $g(t)$ は一般に

$$g(t)=\gamma(t)*\sum_{k=-\infty}^{\infty}\delta(t-kT)\ \ ,\ \ \gamma(t)=0\ \ \text{at}\ \ t<0\ \ \text{or}\ \ t>T$$

と表せるから

$$G(\omega)=\Phi(\omega)\cdot\Delta(\omega)$$

$$\Phi(\omega)=\int_{-\infty}^{\infty}g(t)e^{-i2\pi t\omega}\,dt=\int_0^T g(t)e^{-i2\pi t\omega}\,dt$$

$$\Delta(\omega)=\int_{-\infty}^{\infty}\sum_{k=-\infty}^{\infty}\delta(t-kT)e^{-i2\pi t\omega}\,dt=\frac{1}{T}\sum_{k=-\infty}^{\infty}\delta(\omega-\frac{k}{T})$$

したがって，$G(\omega)=\left\{\int_0^{\infty}g(t)e^{-i2\pi t\omega}\,dt\right\}\frac{1}{T}\sum_{k=-\infty}^{\infty}\delta(\omega-\frac{k}{T})=\sum_{k=-\infty}^{\infty}\left\{\frac{1}{T}\int_0^T g(t)e^{-i2\pi tk/T}\,dt\right\}\delta(\omega-\frac{k}{T})$

付 3)ビジュアルベーシック言語による逆変換データ作成プログラム例

以下は，γ, T, N を与えて，Laplacefunction(p)で定義される複素関数の逆変換を求めるサブルーチンである．Function プロシージャ~ Laplacefunction(p)で定義される関数の逆変換値が得られる．

このサブルーチン InvLapSub では、

①SetInvLapDataSub　を呼び出し FFT 逆変換用データを作成

②FFT 計算に必要なコサインテーブルを作成

③FFTInvSub　を呼び出し FFT 逆変換

④得られた結果に exp(γt) をかける

○逆ラプラス変換サブルーチン

```
Private Sub InvLapSub(G0 As Double, T As Double, N As Integer, Y() As Double)
'**************************************************************************
' G0=γ×Ｔ，　T=Ｔ，　N=Ｎ，　Y()：逆変換結果を記憶し持帰る配列
'  このサブルーチンで作製使用する配列
'　 FFTDataArray()：ＦＦＴサブルーチンで使用する
'　 cst()　　　：ＦＦＴサブルーチンで使用するコサイン表を記憶する
'**************************************************************************
Dim k As Integer
Dim Cst() As Double
Dim FFTDataArray() As ComplexDbl                    '注 1)
    ReDim FFTDataArray(N - 1)                       '計算時に使用する配列を作製
    ReDim Cst(N / 4)
    SetInvLapDataSub G0, T, N, FFTDataArray()       'ＦＦＴ用の配列に関数値を準備する
    For k = 0 To N / 4                              'コサイン表作製
        Cst(k) = Cos(2 * PAI / N * k)
    Next
    FFTInvSub N, FFTDataArray(), Cst()              'ＦＦＴ逆変換ルーチン呼出し
    For k = 0 To N - 1                              'exp(γt)をかける
        Y(k) = FFTDataArray(k).x * Exp(G0 / N * k)  '結果は実部のみを採用すればよい
    Next
    Erase FFTDataArray, Cst                         '不要になった配列を消去
End Sub
```

注 1)ユーザ定義データ型 `ComplexDbl` は複素数を記憶するデータ型で定義文は以下のとおり

```
Type ComplexDbl
    x as Double     '実数部
    y as Double     '虚数部
End Type
```

以下は、γ, T, N を与えて、Function プロシージャー　Laplacefunction(p) で定義される複素関数を逆ラプラス変換するために必要な逆 FFT 用のデータを作成するサブルーチンである．

○逆変換データ作成サブルーチン

```
Sub SetInvLapDataSub(  G0 As Double,  T As Double,  N As Integer,  FFTDataArray() As ComplexDbl)
'*****************************************************************************************
'G0= γ × T ,  T= T,  N= N,  FFTDataArray( )= ＦＦＴサブルーチンで使用する配列
'*****************************************************************************************
Dim p As ComplexDbl, fvalue As ComplexDbl
Dim Hw As Double
Dim k As Integer
    p.x = G0 / T
    For k = 0 To N / 2
        Hw = 0.5 * (Cos(k * PAI * 2 / N) + 1#)
        p.y = 2 * k * PAI / T
        fvalue = Laplacefunction(p)                          '逆変換したい関数の計算　注 2)
        FFTDataArray(k).x = N / T * Hw * fvalue.x
        FFTDataArray(k).y = N / T * Hw * fvalue.y
        If k <> 0 And k <> N / 2 Then
            FFTDataArray(N – k).x = FFTDataArray(k).x
            FFTDataArray(N – k).y = -FFTDataArray(k).y        '共役複素数
        End If
    Next
End Sub
```

注 2)ユーザ定義関数　Laplacefunction() は ComplexDbl 型であり，ここに逆変換したい関数の計算手順を記述する．

```
' ○ステップ関数　exp(-p)/p
Function　Laplacefunction( p As ComplexDbl)　As ComplexDbl
Dim b As Double, p2 As ComplexDbl
    b = p.x * p.x + p.Y * p.Y                                '今の場合 b=0 となる事はない
    p2.x = p.x / b
    p2.Y = -p.Y / b
    Laplacefunction = ComplexDiv(p2, ComplexExp(p))
End Function
```

また、ComplexDbl 型の四則演算関数等は次のようである．

```
'○足し算
Function ComplexAdd(a as ComplexDbl, b as ComplexDbl) as ComplexDbl
    ComplexAdd.x = a.x + b.x
    ComplexAdd.y = a.y + b.y
End Function
'○引き算
Function ComplexSub(a as ComplexDbl, b as ComplexDbl) as ComplexDbl
    ComplexSub.x = a.x - b.x
    ComplexSub.y = a.y - b.y
End Function
'○かけ算
Function ComplexMul(a as ComplexDbl, b as ComplexDbl) as ComplexDbl
    ComplexMul.x = a.x* b.x – a.y * b.y
    ComplexMul.y = a.x * b.y + a.y * b.x
End Function
'○わり算
Function ComplexDiv(a as ComplexDbl, b as ComplexDbl) as ComplexDbl
On Error Resume Next                          ' b=0  のとき０による割り算エラーでプログラムが
dim r As Double                               '  停止しないための処置
    r = b.x * b.x + b.y * b.y
    ComplexDiv.x = (a.x* b.x + a.y * b.y) / r
    ComplexDiv.y = (a.y * b.x - a.x * b.y) / r
End Function
'○指数関数  exp(p)
Function ComplexExp( a As ComplexDbl) As ComplexDbl
Dim r As Double
    r = Exp(a.x)
    ComplexExp.x = r * Cos(a.y)
    ComplexExp.y = r * Sin(a.y)
End Function
```

付 4)FFT 法のプログラム例を以下に示す.

```
○ＦＦＴサブルーチン
Sub FFTSub(NW As Integer, x() As ComplexDbl, cst() As Double)
'************************************************
'* x(0~nw-1) を FFT する
'* Cst(i)=cos(2pai/nw*i)[i=0~nw/4]: コサイン表
'************************************************
Dim n1 As Integer, n2 As Integer, n3 As Integer, StageNo As Integer
Dim ButterflyNo As Integer, GroupNo As Integer, Stage As Integer
Dim Group As Integer, Butterfly As Integer
Dim i As Integer, j As Integer, j1 As Integer, k As Integer
Dim DataIndex As Integer, TableIndex As Integer, iw As Integer
Dim wc As ComplexDbl, Wt As ComplexDbl
    n1 = NW ¥ 4
    n2 = n1 * 2: n3 = n1 * 3
    StageNo = Log(NW) / Log(2)
    ButterflyNo = NW
    GroupNo = 1
    For Stage = 1 To StageNo
        ButterflyNo = ButterflyNo / 2
        DataIndex = 0
        For Group = 1 To GroupNo
            TableIndex = 0
            For Butterfly = 0 To ButterflyNo – 1
                i = DataIndex + Butterfly
                j = DataIndex + Butterfly + ButterflyNo
                GoSub CalcuButterfly
                TableIndex = TableIndex + GroupNo
            Next
            DataIndex = DataIndex + 2 * ButterflyNo
        Next
        GroupNo = 2 * GroupNo
    Next
'ﾋﾞｯﾄ ﾊﾝﾃﾝ
    For i = 1 To NW – 1
        j = 0: k = i : j1 = NW / 2
        Do While k > 0
            j = j + (k Mod 2) * j1
            k = k ¥ 2
            j1 = j1 ¥ 2
        Loop
        If j > i Then    wc = x(i): x(i) = x(j): x(j) = wc
    Next
Exit Sub
CalcuButterfly:   'sin,cos はコサイン表（メモリー節約のために [i=0~nw/4]のみ記憶）から選択
    If TableIndex < n1 Then
        Wt.x = cst(TableIndex) : Wt.y = -cst(n1 - TableIndex)
    ElseIf TableIndex < n2 Then
        iw = TableIndex - n1
        Wt.x = -cst(n1 - iw) : Wt.y = -cst(iw)
    ElseIf TableIndex < n3 Then
        iw = TableIndex - n2
        Wt.x = -cst(iw) : Wt.y = -cst(iw)
    Else
        iw = TableIndex - n3
        Wt.x = cst(n1 - iw) : Wt.y = cst(iw)
    End If
    wc = ComplexSub(x(i), x(j))          '複素数 x(i), x(j) の引き算
    x(i) = ComplexAdd(x(i), x(j))        '複素数 x(i), x(j) の足し算
    x(j) = ComplexMul(wc, Wt)            '複素数 wc, Wt のかけ算
    Return
End Sub
```

○逆ＦＦＴサブルーチン

```
Sub FFTInvSub(NW As Integer, x() As ComplexDbl, cst() As Double)
'************************************************
'* x(0~nw-1) を 逆 FFT する
'*
'* cst(i)=cos(2pai/nw*i)[i=0~nw/4]: コサイン表
'************************************************
Dim r As Double
Dim wc As ComplexDbl
Dim k As Integer
    FFTSub NW, x(), cst()       'FFT ルーチン呼び出し
    r = 1# / NW
    For k = 0 To NW – 1         '1/NW をかける
        x(k).x = r * x(k).x
        x(k).y = r * x(k).y
    Next
    For k = 1 To NW / 2 – 1     '順序入れ換え
        wc = x(k)
        x(k) = x(NW - k)
        x(NW - k) = wc
    Next
End Sub
```