テーマD　線形システムと画像処理への応用 (2016 v1.0)

1　はじめに

1.1　連絡事項

- **プログラム等の事前配布のサイト：**
  `http://www.ide.titech.ac.jp/~yamasita/Exp/index.html`
  を必ず確認しておくこと。
- 上記 Web 上で案内している，eclipse を使って Hello World を表示するまでの基本的な使い方 (youtube.com へのリンク) を，見ておくこと。

1.2　目的

本実験の目的は,「線形システム論 (国際開発工学)」などにおいて学習した，フーリエ級数，フーリエ変換を具体的に実現するための，離散フーリエ変換 (DFT) を学習し,「情報処理実習」で学習した Java 言語を用いて，プログラムで実現する。また，JMF (Java Media Framework) を使って，Web カメラから画像を取得し，DFT を使った畳み込み計算やウィーナーフィルタを，画像処理へ応用する実験を行う。

2　離散フーリエ変換

「線形システム論 (国際開発工学)」において，時間信号を様々な周期をもつ正弦波の和に分解する，フーリエ級数展開とフーリエ変換を勉強した。前者の定義域は有限区間あるいはその繰り返し，後者の定義域は無限区間であるという違いがあった。しかしながら，両者とも連続変数の関数 $f(t)$ を扱うものであった。

一方，コンピュータでデータを処理する場合は，連続関数 $f(t)$ を直接的に扱うことが難しい。そのため，離散フーリエ変換 (Discrete Fourier Transform, DFT) を使う。本節では，まず，フーリエ級数展開とフーリエ変換を復習したのち，この離散フーリエ変換を説明する。さらに，画像を扱うための 2 次元離散フーリエ変換について説明する。

2.1　フーリエ級数展開

0 から T までの関数 $g(t)$ に対して，そのフーリエ級数展開係数 c_n は,

$$c_n = \frac{1}{T}\int_0^T g(\tau)e^{-\frac{2\pi i}{T}n\tau}d\tau$$

によって求まる。この展開係数によって，$g(t)$ をフーリエ級数展開

$$g(t) = \sum_{n=-\infty}^{\infty} c_n e^{\frac{2\pi i}{T}nt}$$

によって表すことができる。オイラーの定理から

$$e^{\frac{2\pi i}{T}nt} = \cos\frac{2\pi}{T}nt + i\ \sin\frac{2\pi}{T}nt$$

が成立する。従って，上式は $g(t)$ を重みをかけた三角関数の和で表していることになる。そして，c_n が $g(t)$ の周波数成分の強さを表している。ここで注意すべきことは，$g(t)$ が実数関数であっても，一般には c_n は複素数になるというこである。

$g(t)$ のフーリエ変換 $G(f)$ は,

$$G(f) = \int_{-\infty}^{\infty} g(\tau) e^{-2\pi i f \tau} d\tau$$

によって求めることができる。また，$G(f)$ のフーリエ逆変換

$$g(t) = \int_{-\infty}^{\infty} G(f) e^{2\pi i f t} df$$

によって，$G(f)$ から $g(t)$ を求めることができる。この式でも，$g(t)$ が $G(f)$ を重みとした三角関数の和 (積分) によって表され，$G(f)$ が $g(t)$ の周波数成分の強さを表している。

数値計算をコンピュータで行う現代では，連続変数関数 $g(t)$ を直接的に扱うことが難しい。そのため，離散フーリエ変換 (Discrete Fourier Transform, DFT) が使われる。DFT では，データは有限個の離散時間上の値である。いま，N 個のデータを g_n $(n = 0, 1, 2, \ldots, N-1)$ で表す。このデータを DFT した結果 G_m $(m = 0, 1, 2, \ldots, N-1)$ は,

$$G_m = \sum_{n=0}^{N-1} g_n e^{-\frac{2\pi i}{N} nm}$$

で与えられる。G_m の逆離散フーリエ変換 (Inverse DFT, IDFT, 逆 DFT) は,

$$g_n = \frac{1}{N} \sum_{m=0}^{N-1} G_m e^{\frac{2\pi i}{N} nm}$$

で与えられる。

この 2 つの式は，g_n と G_m をすべての整数 n, m 上で定義された周期 N の周期関数に拡張しても成立する。**以下では，関数はすべてこのような周期関数であるものとして考えることにする。**

フーリエ級数展開同様に，g_n が実数であっても，一般には G_m は複素数になる。ただし，g_n が実数の場合には，次式が成立する。

$$G_{N-m} = \overline{G_m}$$

従って，g_n が実数の場合，g_m の実数としての自由度は N であるが，上式から G_m の実数としての自由度も N ということになる。

周期関数を考えているので,

$$G_{-m} = G_{N-m} = \overline{G_m}$$

と書くこともできる。これは，実数値関数 $g(t)$ をフーリエ変換したときに,

$$G(-f) = \overline{G(f)}$$

が成立することに相当する。

この関係を証明する。$\overline{g_n} = g_n$，$\overline{e^{i\theta}} = e^{-i\theta}$ であるから，次のようにして証明できる。

$$\begin{aligned} G_{N-m} &= \sum_{n=0}^{N-1} g_n e^{-\frac{2\pi i}{N} n(N-m)} = \sum_{n=0}^{N-1} g_n e^{-2\pi i n + \frac{2\pi i}{N} nm} = \sum_{n=0}^{N-1} g_n e^{\frac{2\pi i}{N} nm} \\ &= \sum_{n=0}^{N-1} \overline{g_n} \overline{e^{-\frac{2\pi i}{N} nm}} = \overline{\sum_{n=0}^{N-1} g_n e^{-\frac{2\pi i}{N} nm}} = \overline{G_m} \end{aligned}$$

DFT の例

DFT の具体例を示す。はじめに，ある整数 k $(0 \le k \le N-1)$ に対して,

$$g_n = \frac{1}{N} e^{\frac{2\pi i}{N} kn}$$

を考える (複素数を使った三角関数)。$m \neq k$ のときは,

$$\begin{aligned} G_m &= \sum_{n=0}^{N-1} \frac{1}{N} e^{\frac{2\pi i}{N}kn} e^{-\frac{2\pi i}{N}mn} = \frac{1}{N} \sum_{n=0}^{N-1} e^{\frac{2\pi i}{N}(k-m)n} = \frac{1}{N} \frac{1 - e^{\frac{2\pi i(k-m)N}{N}}}{1 - e^{\frac{2\pi i(k-m)}{N}}} \\ &= \frac{1}{N} \frac{1-1}{1 - e^{\frac{2\pi i(k-m)}{N}}} = 0 \end{aligned}$$

となり，$m = k$ のときは,

$$G_m = \sum_{m=0}^{N-1} \frac{1}{N} e^{\frac{2\pi i}{N}kn} e^{-\frac{2\pi i}{N}mn} = \frac{1}{N} \sum_{m=0}^{N-1} 1 = 1$$

となる。従って,

$$G_m = \begin{cases} 1 & (m = k) \\ 0 & (m \neq k) \end{cases}$$

となる。

次に，通常の三角関数の変換を考える。オイラーの公式を使って，上の結果から簡単に導くことができる。

$$\frac{1}{N} \cos \frac{2\pi}{N} kn = \frac{1}{N} \frac{e^{\frac{2\pi i}{N}kn} + e^{-\frac{2\pi i}{N}kn}}{2} = \frac{1}{N} \frac{e^{\frac{2\pi i}{N}kn} + e^{\frac{2\pi i}{N}(N-k)n}}{2}$$

であるから，$\frac{1}{N} \cos \frac{2\pi}{N} kn$ の DFT は,

$$G_m = \begin{cases} \frac{1}{2} & (m = k) \\ \frac{1}{2} & (m = N-k) \\ 0 & (m \neq k) \end{cases}$$

となる。同様に，$\frac{1}{N} \sin \frac{2\pi}{N} kn$ の DFT は,

$$\frac{1}{N} \sin \frac{2\pi}{N} kn = \frac{1}{N} \frac{e^{\frac{2\pi i}{N}kn} - e^{-\frac{2\pi i}{N}kn}}{2i} = \frac{1}{N} \frac{e^{\frac{2\pi i}{N}kn} - e^{\frac{2\pi i}{N}(N-k)n}}{2i}$$

より,

$$G_m = \begin{cases} \frac{1}{2i} & (m = k) \\ -\frac{1}{2i} & (m = N-k) \\ 0 & (m \neq k) \end{cases}$$

となる。このようにして，DFT によって，原信号の周波数成分を取り出すことができることがわかる。

2.2 DFT の畳み込み

フーリエ級数展開やフーリエ変換で，2 つの関数を変換したものの積は，もとの 2 つの関数の畳み込みをしたものを変換したものになることを，線形システムの講義で示した。DFT でも同様なことが成立する。ここで，g_n と h_n を DFT したものを，G_m と H_m とする。上では g_n は，$0 \leq n \leq N-1$ で定義されていると考えたが，ここでは，h_n や g_n は全ての整数の集合上で定義された周期 N の周期関数と考える。このとき，畳み込みした d_n を次のように定義する。

$$d_n = \sum_{k=0}^{N-1} h_{n-k} g_k = \sum_{k=0}^{N-1} h_k g_{n-k}$$

(周期関数を仮定しているので，2 番目と 3 番目の式は同じものになる。)

この関係を，d_n を DFT した D_m が，$H_m G_m$ になることを示すことによって証明する。

$$D_m = \sum_{n=0}^{N-1} d_n e^{-\frac{2\pi i}{N}nm} = \sum_{n=0}^{N-1} \sum_{k=0}^{N-1} h_{n-k} g_k e^{-\frac{2\pi i}{N}nm}$$

となる。ここで，k と n の和の順番を交換したあと，$l = n - k$ とおき，n に関する和を，l に関する和に書き換える。

$$D_m = \sum_{k=0}^{N-1} \sum_{n=0}^{N-1} h_{n-k} g_k e^{-\frac{2\pi i}{N} nm} = \sum_{k=0}^{N-1} \sum_{l=-k}^{N-1-k} h_l g_k e^{-\frac{2\pi i}{N}(l+k)m}$$

さらに，h_l と $e^{-\frac{2\pi i}{N}(l+k)m}$ は l に関して周期 N の周期関数であるため，$h_l g_k e^{-\frac{2\pi i}{N}(l+k)m}$ も l に関して周期 N の周期関数になる。従って，l に関する $-k$ から $N-1-k$ までの和を，0 から $N-1$ までの和とすることができる。以上より，

$$\begin{aligned} D_m &= \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} h_l g_k e^{-\frac{2\pi i}{N}(l+k)m} \\ &= \left(\sum_{l=0}^{N-1} h_l e^{-\frac{2\pi i}{N} lm} \right) \left(\sum_{k=0}^{N-1} g_k e^{-\frac{2\pi i}{N} km} \right) \\ &= G_m H_m \end{aligned}$$

となり，DFT したものの積になることがわかる。

DFT の場合，周期関数としての畳み込みが得られるが，このような畳み込みを巡回型の畳み込みと呼ぶ。たとえば，g_m をシステムへの入力と考え，d_n をその出力と考えれば，h_n はシステムの単位パルス応答を表している。この場合単位パルス δ_n は，

$$\delta_n = \begin{cases} 1 & (n = 0) \\ 0 & (n \neq 0) \end{cases}$$

である。この単位パルスの定義で，$n = 0$ だけで 1 と書いてあるが，巡回型の畳み込みを考えているため，実際には $n = 0, \pm N, \pm 2N, \ldots$ において 1 になる。

2.3 パワースペクトル

DFT の結果 G_m は周波数成分を表すが，その値は複素数であるため，

$$G_m = |G_m| e^{i\phi}$$

と書くことができる。このとき，絶対値 $|G_m|$ がその周波数成分の大きさを表し，ϕ がその周波数での位相を表すことになる。また，$|G_m|^2$ をパワースペクトルと呼ぶ。

$g_{-n} = g_{N-n}$ を DFT すると，$\overline{G_m}$ が得られる。パワースペクトル $|G_m|^2$ を IDFT したものを h_l とおく。$|G_m|^2 = G_m \overline{G_m}$ であり，DFT したものの積は，畳み込みしたものを DFT したものであることから，h_l は g_l と g_{N-l} を畳み込みしたものになる。従って，

$$h_l = \sum_{k=0}^{N-1} g_k g_{N-(l-k)} = \sum_{k=0}^{N-1} g_k g_{k-l}$$

となる。左辺は g_n を l だけずらしたものと，もとの g_n と乗算して，0 から $N-1$ までたし合わせたものである。このようなものを，自己相関関数呼ぶ。すなわち，自己相関関数を DFT したものが，パワースペクトルとなる。

信号が確率分布することを考える。すわなち，確率的な構造は同じであるが，様々な信号が現れる場合である。このような場合，この確率によるパワースペクトルの平均を考えることができる。これは，自己相関関数の平均を DFT したものに一致する。**実際の信号処理装置はこのような確率的な構造が分かっているものとして，それを使って設計する場合が多い。**例えば，パワースペクトル (自己相関関数) の平均が与えられるものとして，それを使って平均的に誤差が小さい処理装置を設計する。

2.4 高速フーリエ変換

N 点の値からなるデータの DFT を計算するための計算量を考える。N 個の G_m を計算するために，それぞれ，およそ N 回の複素数の乗算と加算が必要であるので，DFT を直接計算する計算量は N^2 のオーダーとなる。

ソートの場合と同様に，この計算量はデータ数が多くなると問題となる。計算量を削減する方法として，高速フーリエ変換 (fast Fourier transform, FFT) が開発されている。考え方としては，選択ソートに対するマージソートと同じであり，問題を半分ずつに分割していく方法である。

データ点数が 2 のべき乗であるときを考える。すなわち，$N = 2^M$ となる場合である。このとき，FFT の計算量は $N \log_2 N$ のオーダーになる。その原理を簡単に説明する。

まず，入力データのインデックスを奇数の場合と偶数の場合に分けることを考える。

$$\begin{aligned}
G_m &= \sum_{n=0}^{N-1} g_m e^{-\frac{-2\pi i}{N}mn} \\
&= \sum_{l=0}^{N/2-1} g_{2l} e^{-\frac{2\pi i}{N}m(2l)} + \sum_{l=0}^{N/2-1} g_{2l+1} e^{-\frac{2\pi i}{N}m(2l+1)} \\
&= \sum_{l=0}^{N/2-1} g_{2l} e^{-\frac{2\pi i}{N}m(2l)} + e^{\frac{-2\pi i}{N}m} \sum_{l=0}^{N/2-1} g_{2l+1} e^{\frac{2\pi i}{N}m(2l)} \\
&= \sum_{l=0}^{N/2-1} g_{2l} e^{-\frac{2\pi i}{N/2}ml} + e^{-\frac{2\pi i}{N}m} \sum_{l=0}^{N/2-1} g_{2l+1} e^{-\frac{2\pi i}{N/2}ml}
\end{aligned}$$

と書くことができる。この式は，DFT の結果 G_m をすべて求めるためには，インデックスが偶数のデータに対する DFT と奇数のデータに対する DFT を計算した後，N 回程度の乗算と加算で計算できることを示している。すなわち，$N/2$ 点のデータの DFT が 2 回と，N のオーダの計算量が必要になる。同様に，$N/2$ 点の DFT の 1 回の計算には，$N/4$ 点のデータの DFT が 2 回と，$N/2$ のオーダの計算量が必要になる。また，データ数が 1 である DFT はデータの値そのものが DFT の結果であるため，計算量は 0 である。従って，この分割を繰り返せば，

$$\begin{aligned}
& (N \text{ 点の DFT の計算量}) \\
= \;& 2 \times (N/2 \text{ 点の DFT の計算量}) + O(N) \\
= \;& 4 \times (N/4 \text{ 点の DFT の計算量}) + 2 \times O(N/2) + O(N) \\
= \;& \cdots \\
= \;& (N/2) \times (2 \text{ 点の DFT の計算量}) + N/4 \times O(4) + \cdots + 2 \times O(N/2) + O(N) \\
= \;& N \times (1 \text{ 点の DFT の計算量}) + N/2 \times O(2) + N/4 \times O(4) + \cdots + 2 \times O(N/2) + O(N) \\
= \;& O(NM) = O(N \log_2 N)
\end{aligned}$$

となり，高速に DFT が計算できることがわかる。

2.5 2 次元フーリエ変換

今まで，フーリエ変換で扱う関数は時間軸上で定義されているものとして扱ってきたが，もっと一般の量を扱うことができる。たとえば，$g(x)$ で，座標 x における温度を表すとすれば，熱の分布を扱うことができる (フーリエ級数展開は，もともとは線状の金属における熱の伝導を解析するために，フーリエ先生が考え出したものである)。

画像を扱うためには，定義域が 2 次元座標の関数を考える。すなわち，点 (x, y) における明るさを $f(x, y)$ で表す。カラー画像を考える場合は，光の 3 原色である赤，緑，青（RGB) の画素値を考えることになるが，ここでは単純に白黒画像を考え，明るさだけを考えることにする。

さらに，コンピュータで処理するため，明るさは離散点で定義されることになる。画像の画素 (m,n) でのこの明るさの値が，$g_{m,n}$ $(0 \leq m \leq M-1, 0 \leq n \leq N-1)$ で表されているものとする。慣例に従い，1 番目のインデックスが横方向，2 番目のインデックスが縦方向を表すものとする。すなわち，横 $M\times$ 縦 N 画素の画像を考えていることになる。

2 次元の DFT は，2 次元データの縦と横をそれぞれに対して変換する形になる。$g_{m,n}$ を DFT したものを $G_{k,l}$ とおけば，

$$G_{k,l} = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} g_{m,n} e^{-\frac{2\pi i}{M}km - \frac{2\pi i}{N}ln}$$

となる。IDFT も同様に，

$$g_{m,n} = \frac{1}{MN}\sum_{k=0}^{M-1}\sum_{l=0}^{N-1} g_{m,n} e^{\frac{2\pi i}{M}km + \frac{2\pi i}{N}ln}$$

によって表される。

$g_{m,n}$ と $h_{m,n}$ を DFT したものを，それぞれ，$G_{k,l}$ と $H_{k,l}$ する。このとき，$G_{k,l}H_{k,l}$ は，$g_{m,n}$ と $h_{m,n}$ の畳み込み積分 (横の周期が M，縦の周期が N の周期関数を考えている)

$$d_{m,n} = \sum_{k=0}^{M-1}\sum_{l=0}^{N-1} g_{k,l}h_{m-k,n-l} = \sum_{k=0}^{M-1}\sum_{l=0}^{N-1} g_{m-k,n-l}h_{k,l}$$

を，DFT したものになる。

2.6 ウィーナーフィルタ

最も基本的な画像観測のモデルを考える。まず，原画像 $f_{m,n}$ があり，それがブレたボケなどの変換を受け，雑音が加算されたものが観測される。このブレやボケを画像の劣化と呼ぶことにする。

劣化は，画像の位置によらず一定であるものとする。そのため，劣化を 2 次元単位パルス応答を $a_{m,n}$ で表すことができる。この単位パルス応答を，画像処理の分野では，Point Spread Function (点広がり関数) と呼ぶ。すなわち，原画像が原点だけで値が 1 で，それ以外では 0 の場合は，$a_{m,n}$ が劣化画像となる。原画像が一般の画像 $f_{m,n}$ の場合は，$a_{m,n}$ と $f_{m,n}$ の畳み込みが劣化画像となる。

この劣化画像に雑音 $n_{m,n}$ が加算されたものが，観測画像 $g_{m,n}$ になる。従って，画像の観測モデルは，

$$g_{m,n} = \sum_{k=0}^{M-1}\sum_{l=0}^{N-1} a_{k,l}f_{m-k,n-l} + n_{m,n}$$

で与えられる。ここでは，DFT を使うために，画像，劣化の単位パルス応答，雑音ともに，周期が横 M，縦 N の周期関数であるものとする。

$f_{m,j}$，$a_{m,n}$，$n_{m,n}$，$g_{m,n}$ を DFT したものを，$F_{i,j}$，$A_{i,j}$，$N_{i,j}$，$G_{i,j}$ で表す。このとき，

$$G_{i,j} = A_{i,j}F_{i,j} + N_{i,j}$$

が成立する。

この観測画像から復元画像を推定する問題を考える。ここでは，復元のための単位パルス応答 $b_{m,n}$ を観測画像と畳み込むことによって，復元画像を得ることを考える。従って，復元画像 $\hat{f}_{m,n}$ は，

$$\hat{f}_{m,n} = \sum_{k=0}^{M-1}\sum_{l=0}^{N-1} b_{k,l}g_{m-k,n-l}$$

となる。$b_{m,n}$ と $\hat{f}_{m,n}$ を DFT したものを，$B_{i,j}$，$\hat{F}_{i,j}$ で表せば，

$$\hat{F}_{i,j} = B_{i,j}G_{i,j}$$

となる。

この $b_{m,n}$ を具体的に決める必要がある。本実験で用いるウィーナーフィルタは，平均 2 乗誤差を最小するものとして定義される。いま，原画像と雑音に関して，パワースペクトルの平均が分かっているものとする。それぞれのパワースペクトルの平均を $U_{i,j}$ と $V_{i,j}$ とすれば，ウィーナーフィルタを DFT を使って表現した $B_{i,j}$ は，

$$B_{i,j}=\frac{U_{i,j}\overline{A_{i,j}}}{A_{i,j}U_{i,j}\overline{A_{i,j}}+V_{i,j}}$$

となる。

実際の画像への適用することを考える。ピンボケ画像は，原画像の 1 点が円状に広がることになる。その半径を r とし，原点を中心とした半径 r の円内の格子点の数を L とすれば，ボケの単位パルス応答は $a_{i,j}$ は，

$$a_{i,j}=\begin{cases}\frac{1}{L} & (i^2+j^2\le r^2)\\ 0 & (\text{else})\end{cases} \tag{1}$$

となる。実際には，円周部分を線形補間し，滑らかにした関数を用いる。

画像は，本来は周期関数ではない。ただ，DFT を使うためには周期関数でなければいけない。そのため，画像の領域を拡張して大きな領域で DFT を行う。拡張した部分の画素の値を決めなくてはいけない。そのための最も簡単な方法は，画像の周辺の値を線形補間するものである。

3 実習内容

3.1 前諮問

•注意事項：前諮問では，学科の計算機室で動かしてチェックするため，記述したプログラムを USB メモリーなどで持ってくること。

•まず，手始めに 1 から 1000 まで加算し表示するプログラムを作成せよ。

•a_n，b_n $(n=0,1,\ldots,(N-1))$ を 2 つのベクトルの要素とすれば，実ベクトルの内積は次の式で計算できる。

$$\sum_{n=0}^{N-1} a_n\, b_n$$

•$A_{m,n}$ $(m=0,1,\ldots,M-1,\ n=0,1,\ldots,N-1)$ を (M,N)-行列の要素とすれば，その積の結果となるベクトルの要素 c_m $(m=0,1,\ldots,M-1)$ は，次の式で計算できる。

$$c_m=\sum_{n=0}^{N-1} A_{mn}\, a_n$$

•ここでは，$N=M=5$，$\boldsymbol{a}=(1.0\ \ -3.0\ \ 2.0\ \ 2.0\ \ -1.0)^T$，$\boldsymbol{b}=(3.0\ \ 2.0\ \ -3.0\ \ 1.0\ \ 1.0)^T$，$A_{mn}=m^2-n$ として，上に示した内積および行列とベクトルの積を計算し表示するプログラムを作成せよ。

•次に，a_n と b_n を周期 $N(=5)$ の周期関数と考え，相関関数

$$d_\tau=\sum_{t=0}^{N-1} a_t\, b_{t-\tau}$$

を計算し，表示するプログラムを作成せよ。このとき，$t-\tau$ は負になることがあるので，配列を使う場合には注意すること (周期関数であることを使い，インデックスが 0 から $N-1$ 以内になるようにする)。

なお，結果は次のようになる。

ソースコード 1: 前諮問

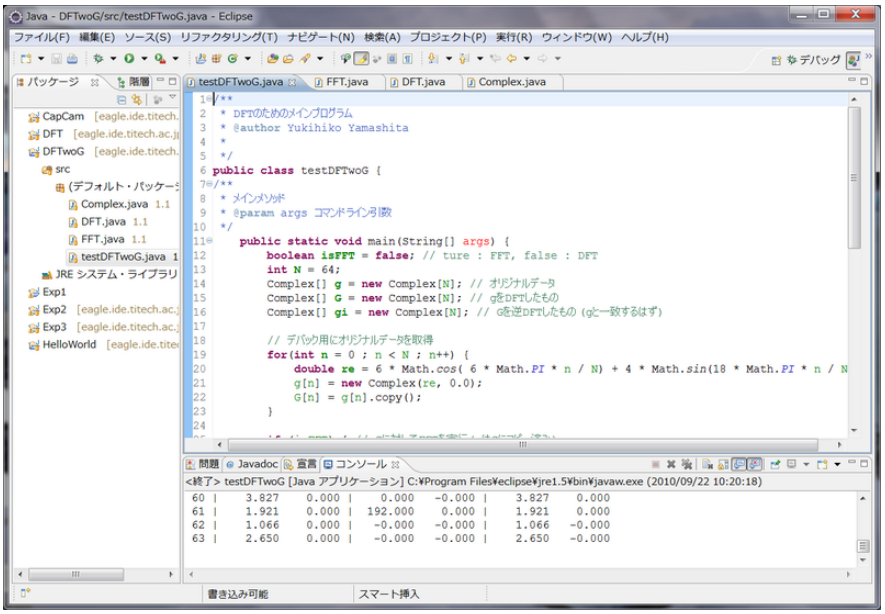
```
public class PrevTest {
  static public void main(String args[]) {
    // 1から1000までを加算して表示
    int sum = 0;
    for (int i = 0 ; i <= 1000 ; ++i) {
      sum += i;
    }
    System.out.println("1 + 2 + ... + 1000 = " + sum);

    // ベクトルの内積
    // 積を計算する配列の宣言・生成・値のセット
    double aV[] = {1.0, -3.0,  2.0, 2.0, -1.0};
    double bV[] = {3.0,  2.0, -3.0, 1.0,  1.0};
    // 内積を格納する変数を宣言
    double innerProduct = 0.0;
    // 次元( 長さ) をNに格納
    int N = aV.length;
    // 内積を計算

    /********* 各自考える *********/

    // 内積を表示
    System.out.println("<aV, bV> = " + innerProduct);

    // 行列とベクトルの積
    // 行列を設定
    double AM[][] = new double[N][N];
    for (int i = 0 ; i < N ; ++i) {
      for (int j = 0 ; j < N ; ++j) {
        AM[i][j] = (double) ((i + 1) * (i + 1) - (j + 1));
      }
    }
    // 答えを格納する配列を宣言・生成
    double cV[] = new double[N];
    // 和を計算するための変数を宣言
    double sumD;
    // 積を計算

    /********* 各自考える *********/

    // 積を表示
    for (int i = 0 ; i < N ; ++i) {
      System.out.println("cV[" + i + "] = " + cV[i]);
    }

    // 周期関数を仮定した相互相関関数
    // 答えを格納する配列を宣言・生成
    double dV[] = new double[N];
    // 和を計算するための変数を宣言
    double cor;
    // 相互相関関数を計算


    /********* 各自考える *********/


    // 相互相関関数を表示
    for (int tau = 0 ; tau < N ; ++tau) {
      System.out.println("correlation[" + tau + "] = " + dV[tau]);
    }
  }
}
```

```
1 + 2 + ... + 1000 = 500500
<aV, bV> = -8.0
cV[0] = -3.0
cV[1] = 0.0
cV[2] = 5.0
cV[3] = 12.0
cV[4] = 21.0
correlation[0] = -8.0
correlation[1] = -11.0
correlation[2] = 11.0
correlation[3] = 0.0
correlation[4] = 12.0
```

3.2 実習内容 (１日目)

3.2.1 Java 開発環境の整備

開発環境として，eclipse を用いる (計算機室での eclipse の起動方法については当日説明する)。



最初の「連絡事項」に書いた通り，eclipse の基本的な使い方を見ておくこと。eclipse の表示として，主に Java パースペクティブを用いる。Java パースペクティブの中で，次の 3 つのウィンドウを表示する。

まず，左側にプロジェクトを管理する「パッケージ・エクスプローラ」を表示する。そして，右 (と中央) 上側にプログラムを編集するためのウィンドウを表示する。表示されてないときは，パッケージ・エクスプローラで，プログラムのファイルをダブルクリックすると，この場所にプログラムを編集するためのウィンドウが開く。右 (と中央) 下側に，エラーの場所などを表示する「問題」，あるいは `System.out.println()` の出力を表示する「コンソール」のウィンドウを表示する (両者は自動的に切り替わるが，そのウィンドウのタブを使って変更可能である)。

それ以外のウィンドウは，ウィンドウ右上の「最小化」のボタンをクリックして，非表示にする (メニューやツールボックスはそのまま)。

eclipse の機能でデバッグを行うと，Debug パースペクティブになる。デバッグの機能の使い方が分からない場合は，変数の値を表示する `System.out.println()` をプログラムの各所に配置し，変数の値の推移を調べてデバッグを行う。

3.3 複素数の計算のプログラミング

- まず，DFT を計算する準備として，複素ベクトルの内積，複素行列と複素ベクトルの積を計算するプログラムを作成する。
- a_n，b_n $(n = 1, 2, \ldots, N)$ を 2 つのベクトルの要素とすれば，内積は次の式で計算できる。

$$\sum_{n=1}^{N} \overline{a_n}\, b_n$$

- 行列とベクトルの積の計算は，実数の場合と同じである。
- 複素数で計算するために，複素数のクラス `Complex` を用意する。
- クラス `Complex` の仕様は以下の通りである。

クラス `Complex` の仕様

- フィールド
  - `double re`
    複素数の実部を格納する。
  - `double im`
    複素数の虚部を格納する。
  - `static final double EPS`
    絶対値がこれ以下のとき，0 と判断する値 (以下の行の様に記述する)。

```
private static final double EPS = 1.0e-10;
```

- コンストラクター
  - `Complex(double re, double im)`
    実部を `re`，虚部を `im` とする `Complex` のオブジェクトを作成する。
- メソッド
  - `boolean equals(Complex c)`
    自身と `c` の実部，虚部の差の絶対値が両方とも `EPS` より小さければ，`true` を返し，そうでなければ `false` を返す。
  - `double real()`
    自身の実部 `re` を返す。
  - `double imag()`
    自身の虚部 `im` を返す。
  - `String toString()`
    この複素数を，それを表す文字列に変換して，その文字列を返す。
  - `Complex set(double re, double im)`
    自身の実部に `re` を虚部に `im` をセットして，自身の参照情報を返す。
  - `Complex set(Complex f)`
    入力された Complex のオブジェクトの実部，虚部を，自身の実部，虚部代入し，自身の参照情報を返す。
  - `Complex toConj()`
    自身をその複素共役に変換して，自身の参照情報を返す。

– `boolean isZero()`
　自身の実部と虚部の絶対値が両方とも `EPS` より小さければ，`true` を返し，そうでなければ `false` を返す。

– `Complex add(Complex a, Complex b)`
　入力された `Complex` のオブジェクト `a` と `b` の値を加算したものを，自身にセットして自身の参照情報を返す (複素数と複素数の加算)。

– `Complex substract(Complex a, Complex b)`
　入力された `Complex` のオブジェクト `a` から `b` の値を減算したものを，自身にセットして自身の参照情報を返す (複素数と複素数の減算)。

– `Complex multiply(Complex a, Complex b)`
　入力された `Complex` のオブジェクト `a` と `b` の値を乗算したものを，自身にセットして自身の参照情報を返す (複素数と複素数の乗算)。このとき，`a`，`b` に自身の参照情報が代入されても正確に計算できるようにする。

– `Complex multiply(Complex a, double b)`
　入力された `Complex` のオブジェクト `a` から `double` 型の `b` の値を乗算したものを，自身にセットして自身の参照情報を返す (複素数と実数の乗算)。

– `Complex divide(Complex a, Complex b)`
　入力された `Complex` のオブジェクト `a` と `b` の値を除算したものを，自身にセットして自身の参照情報を返す (複素数と複素数の除算)。このとき，`a`，`b` に自身の参照情報が代入されても正確に計算できるようにする。

– `Complex negate()`
　自身のオブジェクトの符号 (プラス／マイナス) を反転させて，自身の参照情報を返す。

●2 つの複素数ベクトルは，以下のようにする。

$$\boldsymbol{a} = \begin{pmatrix} 1.0 \\ 1.0 + 1.0i \\ 1.0i \end{pmatrix} \qquad \boldsymbol{b} = \begin{pmatrix} -1.0 \\ 1.0 - 2.0i \\ 1.0 - 1.0i \end{pmatrix}$$

●$M = 4$，$N = 3$ とする。(M, N)-行列 $\boldsymbol{A}$ はその (m, n)-要素が，$m - in^2$ であるものとする。$\boldsymbol{a}$ と $\boldsymbol{b}$ の内積，および $\boldsymbol{A}$ と $\boldsymbol{a}$ の積を計算して表示するプログラムを作成せよ。

●このプログラムの計算以外の部分のソースコードはコピーするので，具体的にはそのプログラムを完成させればよい。

eclipse 上での操作

●eclipse 上に，新しい Java プロジェクト「ComplexMatrix」を作成する。「ファイル」→「新規作成」で，Java プロジェクトを選択し，名前を ComplexMatrix として作成する。

●エクスプローラを開き，フォルダ `C:¥Experiment¥src¥ComplexMatrix` に移動する。そして，`MainComplexMatrix.java`，`Complex.java` を選択し，右クリックして現れるメニューでコピーを選択する。次に，eclipse の「パッケージ・エクスプローラ」のプロジェクト `ComplexMatrix` の `src` にペーストすれば，ファイルのコピーが行われる。

●コピーした `MainComplexMatrix.java` には，計算するプログラムを記述していないため，それを追加して記述し `MainComplexMatrix.java` を完成させる。

●完成したら `MainComplexMatrix.java` を実行させ，その結果を記録せよ。

●なお，結果は次のようになる。

```
<cVectA, cVectB> = -3.000 - 4.000i
cVectC[0] =  7.000 - 1.000i
cVectC[1] =  13.000 + 5.000i
cVectC[2] =  23.000 + 15.000i
cVectC[3] =  37.000 + 29.000i
```

3.3.1 DFT のプログラミング

●DFT を実装したクラス DFT のプログラムを作成する。

●クラス DFT の仕様は以下の通りである。

クラス DFT の仕様

●フィールド

– int N
　処理するデータ数

– Complex[] wTbl
　高速化のための三角関数値のテーブル (高速化のために，DFT を実行する前に，必要な三角関数の値を計算し，配列に格納しておく)。三角関数のテーブルの利用が難しければ，利用しなくてよい。

●コンストラクタ

– DFT(int N)
　要素数 N の DFT を計算するためのオブジェクトを作成する。
　フィールド N の設定と，三角関数値のテーブルの作成を行う。

●メソッド

– void dft(Complex[] data)
　data の DFT を計算して data に格納する (メソッド内部に，長さ N の作業用配列を作成し，そこに DFT を計算したものを格納し，それを最後に data に書き戻す)。

– void idft(Complex[] data)
　data の逆 DFT を計算して，false ならば，data の DFT を data に格納する

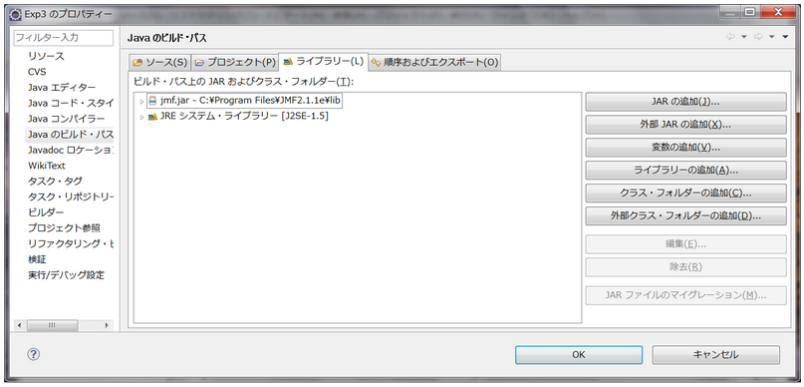
eclipse 上での操作

●eclipse 上に，新しい Java プロジェクト「DFT」を作成する。「ファイル」→「新規作成」で，Java プロジェクトを選択し，名前を DFT として作成する。

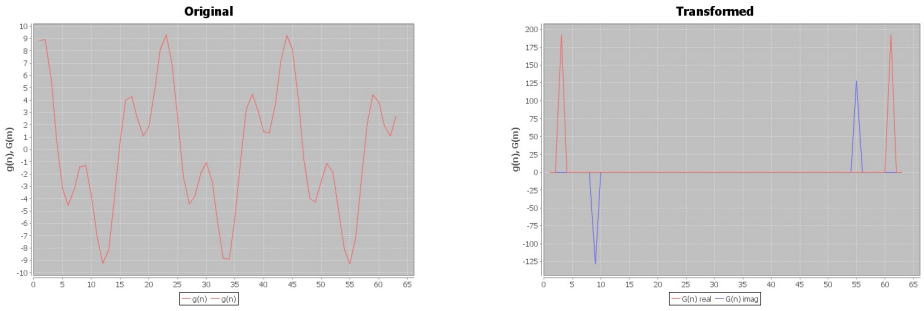
●グラフを書くためのライブラリーを追加する。「パッケージ・エクスプローラ」のプロジェクト DFT の上で右クリックし,「プロパティ」をクリックする。現れたウィンドウの「Java のビルド・パス」をクリック,「ライブラリー」のタブをクリックして,「外部 JAR の追加」をクリックすれば，追加することができる。追加するライブラリーは，C:¥Experiment¥jars の下の，jcommon-1.0.16.jar と jfreechart-1.0.13.jar の 2 つである。

ソースコード 2: 複素数の計算

```
64 /**
65  * 複素数のベクトルの内積と複素数行列と複素数ベクトルの積を実行する。
66  * ベクトルや行列は特にクラスを作らずに，配列で計算する。
67  * @author yamasita
68  *
69  */
70 public class MainComplexMatrix {
71  public static void main(String[] args) {
72   int M = 4, N = 3;
73   Complex cVectA[] = new Complex[N];
74   Complex cVectB[] = new Complex[N];
75   Complex cVectC[] = new Complex[M];
76   Complex cMat[][] = new Complex[M][N];
77 
78   Complex cConjugate = new Complex(0.0, 0.0); // cVectA[n] の複素共役を計算
79                                               // するための変数
80   Complex cInProduct = new Complex(0.0, 0.0); // 最終的に内積の値が入る変数
81   Complex cProduct   = new Complex(0.0, 0.0); // 複素数の積を計算するための
82                                               // 変数
83   Complex cSum       = new Complex(0.0, 0.0); // 行列とベクトルの積を計算す
84                                               // る時に，和を計算するための
85                                               // 変数
86 
87   // ベクトルcVectAとcVectBの要素に値を代入したオブジェクトをセット
88   cVectA[0] = new Complex(1.0, 0.0);
89   cVectA[1] = new Complex(1.0, 1.0);
90   cVectA[2] = new Complex(0.0, 1.0);
91 
92   cVectB[0] = new Complex(-1.0,  0.0);
93   cVectB[1] = new Complex( 1.0, -2.0);
94   cVectB[2] = new Complex( 1.0, -1.0);
95 
96   // 内積を計算
97 
98   /***************  各自考える  *********************/
99 
100  }
101  // 内積を表示
102  System.out.println("<cVectA, cVectB> = " + cInProduct.toString());
103 
104  // cVectCの要素のコンストラクトと行列cMatの要素に値を代入した
105  // オブジェクトをセット
106  for (int m = 0 ; m < M ; ++m) {
107   cVectC[m] = new Complex(0.0, 0.0);
108   for (int n = 0 ; n < N ; ++n) {
109    cMat[m][n] = new Complex((double) (m + 1), (double) (n + 1) * (n + 1));
110   }
111  }
112 
113  // 行列  x ベクトルを計算
114 
115  /***************  各自考える  *********************/
116 
117  // ベクトルを表示
118  for (int m = 0 ; m < M ; ++m)
119   System.out.println("cVectC[" + m + "] = " + cVectC[m].toString());
120 }
121 }
```



- エクスプローラを開き，フォルダ `C:¥Experiment¥src¥DFT` に移動する。そして，`MainDFT.java`，`Complex.java`，`DFT.java`，`FFT.java`，`Plot.java` を選択し，右クリックして現れるメニューでコピーを選択する。次に，eclipse の「パッケージ・エクスプローラ」のプロジェクト `DFT` の `src` にペーストすれば，ファイルのコピーが行われる。
- コピーした `DFT.java` には，宣言程度しか記述していないため，`DFT.java` を完成させる。
- クラス `MainDFT` における，メソッド `main()` の変数 `saveFolder` の初期値を，各自のグラフを保存するフォルダに変更する。
- 完成したら `MainDFT.java` を実行させ，その結果を記録せよ。



結果の例

- **データ数を変化させ**，`DFT` と `FFT` のプログラムの速度を比較せよ。
  - DFT の計算に数十秒かかる程度まで，データ数を増やしてみる。
  - 時間を計測するためには，`MainDFT.java` の `measureTime` を `true` にする。その設定によって表示などの DFT の計算に関係ない部分を実行しないようにする。計算時間は，コンソールに表示される。
  - 計算時間が短すぎて 0 と表示される場合は，DFT の計算を 100 回繰り返し，得られた時間を 100 で割ると良い。
  - データ数を変化させるためには，`MainDFT.java` の `N` を変化させる。

3.4 実習内容 (２日目)

3.4.1 畳み込み演算のプログラミング

- eclipse で，プロジェクト `DFT` をコピーして，畳み込み演算のためのプロジェクト `Conv` を作成する。
  プロジェクト `DFT` を選択して，メニューの「編集」→「コピー」をクリックする。そして，メニューの「編集」→「貼り付け」をクリックする (右クリックを使っても操作可能である)。
- パッケージエクスプローラのプロジェクト `Conv` の下の `src` の下のデフォルトパッケージの下にあるファイルを削除する (選択して Delete キーを押せば削除できる)。

●エクスプローラを開き，フォルダ C:¥Experiment¥src¥Conv に移動する。そして，MainConv.java，Complex.java，Conv.java，FFT.java，Plot.java を選択し，右クリックして現れるメニューでコピーを選択する。次に，eclipse の「パッケージ・エクスプローラ」のプロジェクト Conv の src のディフォルトパッケージにペーストすれば，ファイルのコピーが行われる。

●クラス MainConv における，メソッド main() の変数 saveFolder の初期値を，各自のグラフを保存するフォルダに変更する。

●クラス MainConv のメソッド main() は，畳み込む 2 つのデータを作成し (ある一定間隔で 1 になり他では 0 である信号と，指数関数 e^{-an})，その畳み込み演算をメソッド directConv() または fftConv() を使って計算し，それぞれをグラフに表示し，グラフをファイルに保存する。畳み込みのプログラムの動作を確認後，正弦波と指数関数など，各自データを作成するプログラムを書き，畳み込み演算をしてみること。また，データ量と実行時間の関係などを比べてみること。

クラス Conv の仕様

●フィールド
- int N : 処理するデータ数
- Complex[] hF, gF
  受け取ったデータ df1 と df2 を FFT したものを格納する。

●コンストラクタ
- Conv(int N)
  要素数 N の畳み込みを計算するためのオブジェクトを作成する。
  フィールド N の設定と，配列 hF, gF のための領域確保を行う。

●メソッド
- void directConv(double[] h, double[] g, double[] d)
  h と g の (周期関数を仮定した) 畳み込みを定義式から直接計算し，d に格納する。
- void fftConv(double[] h, double[] g, double[] d)
  h と g の (周期関数を仮定した) 畳み込みを計算し，d に格納する。計算は，h と g を hF と gF に格納し，DFT 係数を FFT を使って求め，その積を逆 FFT して実数部を d に格納する。

3.4.2 画像処理の実験

●Dell のウェブカメラアプリケーション (Dell Webcam Central) が動いている場合には，それを止める (具体的方法は実習中に指示する)。

●C:¥Program Files¥JMF2.1.1e¥bin¥の下の jmfinit.exe を実行する。どのカメラを使うか指定するウィンドウが現れるので，内部カメラ (Integrated Webcam) を指定する。

●eclipse を立ち上げ，新規プロジェクト Wiener を作成し，外部 JAR として C:¥Program Files¥JMF2.1.1e¥lib¥の下の jmf.jar を追加する。また，ソースプログラムとして，C:¥Experiment¥src¥Wiener の下の，Complex.java，FFT.java，FFT2.java (2 次元 FFT)，ImagePanel.java (画像の表示と動作制御を行うプログラム，ImagegProc.java (ウィーナフィルタの計算などを行うプログラム)，MainWiener.java をコピーする。

●クラス MainWiener における，メソッド main() の変数 saveFolder の初期値を，各自の画像を保存するフォルダに変更する。

●MainWiener.java を開いて，eclipse の「実行」をクリックすると，カメラから画像取得し，白黒画像を生成し，ウィーナーフィルタで処理するプログラムが立ち上がる。どのカメラを使うか指定するウィンドウが現れるので，内部カメラ (Integrated Webcam) を指定する (現れない時もある)。

- ここでこのプログラムを一度終了する。
- ヒープ領域を増やす。「パッケージ・エクスプローラ」のプロジェクト `Wiener` の上で右クリックし，「プロパティ」をクリックする。現れたウィンドウの「実行/デバック設定」をクリックし,「MainWiener」選択して「編集」をクリックする。現れたウィンドウの「引数」タブをクリックして，VM 引数の欄に，

  `-Xms512m`

  と記入し,「OK」をクリックして，プロパティウィンドウの「適用」と「OK」をクリックする。
- 再度「実行」をクリックして，プログラムを開始する。このプログラムにおいて,「画像取得」をクリックすると，画像を取得し，白黒画像に変換する。「画像処理」をクリックすると，ウィーナフィルタによる処理を実行する。「画像保存」をクリックすると，現在表示されている画像を JPEG ファイルとして保存する。また,「表示：」の右側の 3 つのボタン「カメラ画像」,「取得画像」,「処理済画像」のボタンをクリックすると，それぞれ，現在のカメラの画像，取得した白黒画像，ウィーナーフィルタで処理した画像を表示する。「ボケ半径：」右横のテキストボックスに数値を入力すると，ウィーナフィルタで使われるボケを表す式 (1) における r の値 (ボケの半径) を設定できる。
- 画像を取得した後，ボケ半径の値を変え，ウィーナーフィルタを使って処理を行い，画像が程度復元可能か調べよ。また，物体とカメラの距離を調整して画像のボケの度合いを変えて，同様に調べよ。
- 時間があれば，プロジェクト `Wiener` のソースプログラムを読み，画像や雑音のパワースペクトルを変更したり，画像に対して，2 次元畳み込みを行い表示するプログラムなどを作成せよ。

3.5 後試問

- 画像処理の実験が終了後，課題を出すので，プログラムを作成し，結果を教員または TA に見せる。

3.6 レポートに書く内容

- DFT (FFT) の実験結果と，それに関して考察したことを述べよ。
- 畳み込み演算の実験結果と，それに関して考察したことを述べよ。
- ウィーナーフィルタの実験結果と，それに関して考察したことを述べよ。