1　離散フーリエ変換

1.1　直交性

$\{\exp(ik\alpha 2\pi)\}_{k=0,1,2,\dots}$ は $\alpha \in \mathbb{Z}$ の時 1, そうでない時は等比級数であるから;

$$\sum_{k=0}^{N-1} \exp(ik\alpha 2\pi) = \begin{cases} N & \text{if } \alpha \in \mathbb{Z} \\ \dfrac{1-\exp(iN\alpha 2\pi)}{1-\exp(i\alpha 2\pi)} & \text{otherwize} \end{cases}$$

特に $\alpha = M/N$ $(M \in \mathbb{Z})$ ならば

$$\sum_{k=0}^{N-1} \exp(i\frac{kM}{N}2\pi) = \begin{cases} N & \text{if } M = 0 \pmod N \\ 0 & \text{otherwize} \end{cases}$$

1.2　離散フーリエ変換

$\{x_k\}_{k=0}^{N-1}$, $\{\xi_j\}_{j=0}^{N-1}$ が

$$x_j = \sum_{k=0}^{N-1} \xi_k \exp(i\frac{kj}{N}2\pi)$$

を満たすとき, 辺辺に $\exp(-i\frac{lj}{N}2\pi)$ を乗じて j について和をとれば直交性により

$$\sum_{j=0}^{N-1} x_j \exp(-i\frac{lj}{N}2\pi) = \sum_{k=0}^{N-1} \xi_k \sum_{j=0}^{N-1} \exp(i\frac{(k-l)j}{N}2\pi) = \sum_{k=0}^{N-1} \xi_k N\delta_k^l = N\xi_l$$

よって

$$\xi_l = \frac{1}{N}\sum_{j=0}^{N-1} x_j \exp(-i\frac{lj}{N}2\pi)$$

2　高速フーリエ変換

一つの ξ_l を計算するのに式

$$\xi_l = \frac{1}{N}\sum_{j=0}^{N-1} x_j \exp(-i\frac{lj}{N}2\pi)$$

を用いれば N 回の乗算が必要である. これを $l = 0, 1, \dots, N-1$ まで繰り返せば結局 N^2 回の乗算が必要である.

2.1 $N = N_1 N_2$

$N = N_1 N_2$ とする. 任意の $0 \le l < N$ について ξ_l を求める事は任意の $0 \le l_1 < N_1, 0 \le l_2 < N_2$ について $\xi_{l_1 N_2 + l_2}$ を求める事と同じである.

$$\begin{aligned}\xi_{l_1 N_2 + l_2} &= \frac{1}{N} \sum_{j=0}^{N-1} x_j \exp(-i \frac{(l_1 N_2 + l_2) j}{N} 2\pi) \\ &= \frac{1}{N} \sum_{j_1=0}^{N_1-1} \sum_{j_2=0}^{N_2-1} x_{j_1 + j_2 N_1} \exp(-i \frac{(l_1 N_2 + l_2)(j_1 + j_2 N_1)}{N} 2\pi)\end{aligned}$$

特に $(l_1 N_2 + l_2)(j_1 + j_2 N_1) = (l_1 N_2 + l_2) j_1 + l_2 j_2 N_1 + l_1 j_2 N_2 N_1$ かつ $\exp(-i \frac{l_1 j_2 N_2 N_1}{N} 2\pi) = \exp(-i l_1 j_2 2\pi) = 1$ なる事を使えば

$$\xi_{l_1 N_2 + l_2} = \frac{1}{N} \sum_{j_1=0}^{N_1-1} \exp(-i \frac{(l_1 N_2 + l_2) j_1}{N} 2\pi) \sum_{j_2=0}^{N_2-1} x_{j_1 + j_2 N_1} \exp(-i \frac{l_2 j_2 N_1}{N} 2\pi)$$

特に中の和

$$\eta_{j_1, l_2} := \sum_{j_2=0}^{N_2-1} x_{j_1 + j_2 N_1} \exp(-i \frac{l_2 j_2 N_1}{N} 2\pi)$$

が l_1 に依存しない事に注意せよ.

つまり, $\xi_{l_2}, \xi_{l_1 + l_2}, \xi_{2l_1 + l_2}, \ldots$ の計算に η_{j_1, l_2} を再利用する事ができ, その分乗算を節約できる, i.e.,

$$\xi_{l_1 N_2 + l_2} = \frac{1}{N} \sum_{j_1=0}^{N_1-1} \eta_{j_1, l_2} \exp(-i \frac{(l_1 N_2 + l_2) j_1}{N} 2\pi)$$

各 η_{j_1, l_2} の計算に N_2 回の乗算が必要で $0 \le j_1 < N_1$, $0 \le l_2 < N_2$ で $N = N_1 N_2$ 個あるので $N N_2$ 回の乗算が必要. 各 $\xi_{l_1 N_2 + l_2}$ の計算は N_1 回の乗算が必要で $0 \le l_1 < N_1$, $0 \le l_2 < N_2$ で $N = N_1 N_2$ 個あるので $N N_1$ 回の乗算が必要. 結局 $N(N_1 + N_2)$ 回の乗算で計算できる.

2.2 $N = N_0 N_1 \cdots N_{p-1}$

$$\overline{N}_k := \prod_{l>k} N_l, \qquad \underline{N}_k := \prod_{l<k} N_l$$

と置く, $0 \le j_t < N_t$ の時

$$j = \sum_{t=0}^{p-1} j_t \underline{N}_t = j_0 + j_1 N_0 + j_2 N_0 N_1 + \ldots + j_{p-1}(N_0 \cdots N_{p-2})$$

は $0 \leq j < N$ を動く.

$$l = \sum_{t=0}^{p-1} l_t \overline{N}_t = l_{p-1} + l_{p-2}N_{p-1} + l_{p-3}N_{p-1}N_{p-2} + \ldots + l_0(N_{p-1}\cdots N_1)$$

についても同様.

$$jl = (\sum_{t=0}^{p-1} j_t \underline{N}_t)(\sum_{\tau=0}^{p-1} l_\tau \overline{N}_\tau) = \sum_{t=0}^{p-1} j_t \sum_{\tau=0}^{p-1} l_\tau \overline{N}_\tau \underline{N}_t$$

特に定義から $t \geq \tau + 1$ の時 $\underline{N}_t \overline{N}_\tau / N \in \mathbb{Z}$ であるので

$$\begin{aligned} \exp(-i(\sum_{t=0}^{p-1} j_t \underline{N}_t)(\sum_{\tau=0}^{p-1} l_\tau \overline{N}_\tau)\frac{2\pi}{N}) &= \exp(-i\sum_{t=0}^{p-1} j_t \sum_{\tau=t}^{p-1} l_\tau \overline{N}_\tau \underline{N}_t \frac{2\pi}{N}) \\ &= \prod_{t=0}^{p-1} \exp(-ij_t \sum_{\tau=t}^{p-1} l_\tau \overline{N}_\tau \underline{N}_t \frac{2\pi}{N}) \end{aligned}$$

$$\begin{aligned} \xi_{\sum_{\tau=0}^{p-1} l_\tau \overline{N}_\tau} &= \frac{1}{N} \sum_{j_0=0}^{N_0-1} \cdots \sum_{j_{p-1}=0}^{N_{p-1}-1} x_{\sum_{t=0}^{p-1} j_t \underline{N}_t} \exp(-i(\sum_{t=0}^{p-1} j_t \underline{N}_t)(\sum_{\tau=0}^{p-1} l_\tau \overline{N}_\tau)\frac{2\pi}{N}) \\ &= \frac{1}{N} \sum_{j_0=0}^{N_0-1} \cdots \sum_{j_{p-1}=0}^{N_{p-1}-1} x_{\sum_{t=0}^{p-1} j_t \underline{N}_t} \exp(-i\sum_{t=0}^{p-1} j_t \sum_{\tau=t}^{p-1} l_\tau \overline{N}_\tau \underline{N}_t \frac{2\pi}{N}) \\ &= \frac{1}{N} \sum_{j_0=0}^{N_0-1} \exp(-ij_0 \sum_{\tau=0}^{p-1} l_\tau \overline{N}_\tau \underline{N}_0 \frac{2\pi}{N}) \cdots \sum_{j_{p-1}=0}^{N_{p-1}-1} \exp(-ij_{p-1} \sum_{\tau=p-1}^{p-1} l_\tau \overline{N}_\tau \underline{N}_p \frac{2\pi}{N}) x_{\sum_{t=0}^{p-1} j_t \underline{N}_t} \end{aligned}$$

よって

$$\begin{aligned} \eta^{(p)}_{j_0,\cdots,j_{p-1}} &:= x_{\sum_{t=0}^{p-1} j_t \underline{N}_t} \\ \eta^{(q)}_{j_0,\cdots,j_{q-1},l_q,\cdots,l_{p-1}} &:= \sum_{j_q=0}^{N_q-1} \exp(-ij_q \underline{N}_q \sum_{\tau=q}^{p-1} l_\tau \overline{N}_q \frac{2\pi}{N}) \eta^{(q+1)}_{j_0,\cdots,j_q,l_{q+1},\cdots,l_{p-1}} \end{aligned}$$

とおけば

$$\xi_{\sum_{\tau=0}^{p-1} l_\tau \overline{N}_\tau} = \eta^{(1)}_{l_0,\cdots,l_{p-1}}$$

3 実装に関する考察

ループの”深さ”が実行時に変化する p に依存するのは面倒である為[1], $\eta^{(q)}_{j_0,\cdots,j_{q-1},l_q,\cdots,l_{p-1}}$ を表すのに p 次元配列では無く 1 次元配列を用いる事を考える. i.e.,

$$\eta^{(q)}_{j_0,\cdots,j_{q-1},l_q,\cdots,l_{p-1}} = \widetilde{\eta}^{(q)}[\sum_{\tau=0}^{q-1} j_\tau \overline{N}_\tau + \sum_{\tau=q}^{p-1} l_\tau \overline{N}_\tau]$$

[1]再帰呼び出しにより可能ではある.

とおく事にすれば

$$\eta^{(q)}[\sum_{\tau=0}^{q-1} j_\tau \overline{N}_\tau + \sum_{\tau=q}^{p-1} l_\tau \overline{N}_\tau] \quad = \quad \sum_{j_q=0}^{N_q-1} \exp(-ij_q \underline{N}_q \sum_{\tau=q}^{p-1} l_\tau \overline{N}_q \frac{2\pi}{N}) \eta^{(q+1)}[\sum_{\tau=0}^{q} j_\tau \overline{N}_\tau + \sum_{\tau=q+1}^{p-1} l_\tau \overline{N}_\tau]$$

である. これを「$0 \le j_\tau < N_\tau$ for $\tau < q$ and $0 \le l_\tau < N_\tau$ for $\tau \ge q$ 」について計算する (単純に考えれば, 左辺の各値の計算を繰り返すのに p 個のカウンタ $j_0, \ldots, j_{q-1}, l_q, \ldots, l_{p-1}$ を持つ p 重ループとなる) わけであるが. 式中のカウンタに関連する値が

$$\sum_{\tau=0}^{q-1} j_\tau \overline{N}_\tau, \ \sum_{\tau=q+1}^{p-1} l_\tau \overline{N}_\tau, \ j_q, \ l_q$$

の組み合わせのみである事に注意して $jN_q\overline{N}_q = \sum_{\tau=0}^{q-1} j_\tau \overline{N}_\tau, l = \sum_{\tau=q+1}^{p-1} l_\tau \overline{N}_\tau$ とすれば

$$\eta^{(q)}[jN_q\overline{N}_q + L_q\overline{N}_q + l] \quad = \quad \sum_{j_q=0}^{N_q-1} \exp(-ij_q \underline{N}_q l \frac{2\pi}{N}) \eta^{(q+1)}[jN_q\overline{N}_q + L_q\overline{N}_q + l]$$

を「$0 \le j < uN_q$, $0 \le l_q < N_q$ and $0 \le l < oN_q$」について計算すればよく. 3 重の (p に依存しない) ループで実現できる.

3.1 $N = 2^p$ の時

つまり $N_q = 2$ であるので

$$\begin{aligned} \eta^{(q)}_{j_0,\cdots,j_{q-1},l_q,\cdots,l_{p-1}} &:= \sum_{j_q=0}^{1} \eta^{(q+1)}_{j_0,\cdots,j_q,l_{q+1},\cdots,l_{p-1}} \exp(-ij_q \underline{N}_q \sum_{\tau=q}^{p-1} l_\tau \overline{N}_\tau \frac{2\pi}{N}) \\ &:= \eta^{(q+1)}_{j_0,\cdots,j_q,l_{q+1},\cdots,l_p} + \eta^{(q+1)}_{j_0,\cdots,j_q,l_{q+1},\cdots,l_p} \exp(-i\underline{N}_q \sum_{\tau=q}^{p-1} l_\tau \overline{N}_\tau \frac{2\pi}{N}) \end{aligned}$$

であるので実質的に格段で乗法は 1 回で済むため, $N \log_2 N$ 回の乗法で済む. 更に, ビットマスクにより,

$$m = \sum_{\tau=0}^{q-1} j_\tau \overline{N}_\tau + \sum_{\tau=q}^{p-1} l_\tau \overline{N}_\tau = \sum_{\tau=0}^{q-1} j_\tau 2^{\tau-1} + \sum_{\tau=q}^{p-1} l_\tau 2^{\tau-1}$$

から容易に

$$\sum_{\tau=0}^{q-1} j_\tau 2^{\tau-1}, \ l_q, \ \sum_{\tau=q+1}^{p-1} l_\tau 2^{\tau-1}$$

を取り出す事ができるので効率が良い.

4 実装

$N = 160384$ の時のフーリエ変換を通常の方法, $N = 128 \times 128$ に分解した高速変換, $N = 2^{14}$ に因数分解した高速変換, $N = 2^{14}$ に因数分解した 2 のベキ乗数専門の高速変換の 3 種類の時間を比較した.

結果:

```
#CLOCKS_PER_SEC 1000.000000
#NORMAL time = 80281.00000000000000000000000000000
#I   time = 1266.00000000000000000000000000000 error = 0.000000
#II  time = 187.00000000000000000000000000000 error = 0.000000
#III time = 78.00000000000000000000000000000 error = 0.000000
```

ソース:

```
/************************************************
***  高速フーリエ変換                            ***
************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>

struct cmplx{ double re, im; };


// 通常のフーリエ変換
void FourierTransform(
  struct cmplx x[],  // 変換前 (入力) データ列
  struct cmplx a[],  // 変換後 (出力) データ列
  unsigned int N
){
  unsigned int j, k;
  const double theta = 2.0*M_PI/N;
  double cos_jk, sin_jk;

  for( k=0; k<N; k++ ){

    a[k].re = 0;
    a[k].im = 0;
    for( j=0; j<N; j++ ){
      cos_jk = cos( theta*j*k );
      sin_jk = sin( theta*j*k );
      a[k].re += x[j].re * cos_jk + x[j].im * sin_jk;
      a[k].im += x[j].im * cos_jk - x[j].re * sin_jk;
    }

    a[k].re /= N;
    a[k].im /= N;
  }
}



// 高速フーリエ変換 I(積)
int FFourierTransform_NN(
  struct cmplx x[],  // 変換前 (入力) データ列
  struct cmplx a[],  // 変換後 (出力) データ列
  unsigned int N1,
  unsigned int N2
){
  unsigned int j1, j2, l1, l2;
  const double theta2 = 2.0*M_PI/N2;
  const double theta = 2.0*M_PI/(N1*N2);
  double cos_j2l2, sin_j2l2, cos_l1N2_l2, sin_l1N2_l2;
  struct cmplx *eta;
```

```
  // 作業領域確保
  if( NULL == ( eta = (struct cmplx *)malloc( sizeof(struct cmplx)*N1*N2 ) ) ){
    return 1;
  };

  for( j1=0; j1 < N1; j1++ ){
    for( l2=0; l2 < N2; l2++ ){
      eta[ j1*N2 + l2 ].re = 0;
      eta[ j1*N2 + l2 ].im = 0;
      for( j2=0; j2<N2; j2++ ){
        cos_j2l2 = cos( theta2 *j2*l2 );
        sin_j2l2 = sin( theta2 *j2*l2 );
        eta[ j1*N2 + l2 ].re +=
          x[ j1 + j2*N1 ].re * cos_j2l2 + x[ j1 + j2*N1 ].im * sin_j2l2;
        eta[ j1*N2 + l2 ].im +=
          x[ j1 + j2*N1 ].im * cos_j2l2 - x[ j1 + j2*N1 ].re * sin_j2l2;
      }
    }
  }

  for( l1=0; l1 < N1; l1++ ){
    for( l2=0; l2 < N2; l2++ ){
      a[ l1*N2 + l2 ].re = 0;
      a[ l1*N2 + l2 ].im = 0;
      for( j1=0; j1<N1; j1++ ){
        cos_l1N2_l2 = cos( theta *j1*( l1*N2 + l2 ) );
        sin_l1N2_l2 = sin( theta *j1*( l1*N2 + l2 ) );
        a[ l1*N2 + l2 ].re +=
          eta[ j1*N2 + l2 ].re * cos_l1N2_l2 + eta[ j1*N2 + l2 ].im * sin_l1N2_l2;
        a[ l1*N2 + l2 ].im +=
          eta[ j1*N2 + l2 ].im * cos_l1N2_l2 - eta[ j1*N2 + l2 ].re * sin_l1N2_l2;
      }
      a[ l1*N2 + l2 ].re /= N1*N2;
      a[ l1*N2 + l2 ].im /= N1*N2;
    }
  }

  free( (void *)eta ); // メモリ解放
  return 0;
}


// 高速フーリエ変換 II(因数分解)
int FFourierTransform(
  struct cmplx x[],  // 変換前 (入力) データ列
  struct cmplx a[],  // 変換後 (出力) データ列
  unsigned int N[],  // N の因数分解
  unsigned int p     // N の因数分解の項数
){
  unsigned int q, t, j_q, l_q, j, l, NN, m, k, tmpl, qq;
  double theta;
  double cos_jql, sin_jql;
  struct cmplx *eta, *last_eta, *tmpp, *buf;
  unsigned int *uN, *oN;


  NN = 1;
  for( q=0; q<p; q++) NN *= N[q];
  theta = 2.0*M_PI/NN;

  // 作業領域確保
  if(
    NULL == ( buf = (struct cmplx *)malloc( sizeof(struct cmplx)*NN ) )
    || NULL == ( uN = (unsigned int *)malloc( sizeof(unsigned int)*NN ) )
    || NULL == ( oN = (unsigned int *)malloc( sizeof(unsigned int)*NN ) )
  ){
    printf("ERROR\n");
    return 1;
  };
  eta = a; last_eta = buf;

  // uN[t] = uN[0]*...uN[t-1]
  // oN[t] = uN[t+1]*...uN[p-1]
```

```
  uN[0] = 1;
  for( q=0; q<p-1; q++) uN[q+1] = uN[q]*N[q];
  for( q=0; q<p; q++) oN[q] = NN/(uN[q]*N[q]);

//  for( q=0; q<p; q++) printf( "q %d uN[q] %d  oN[q] %d\n" ,q,uN[q], oN[q]);

  // eta^p 並べ替え
  for( l=0; l<NN; l++){
    tmpl = l; j=0 ;
    for( q=0; q<p; q++){
      l_q  = tmpl / oN[q];
      tmpl = tmpl % oN[q];
      j += l_q*uN[q];
    }
    eta[l] = x[j];
//    printf( "%d %d\n" ,l,j);
  }


  for( qq=p; qq>0; qq--){
    q = qq - 1 ;
    tmpp = last_eta; last_eta = eta; eta = tmpp;

    for( j=0; j< uN[q] ; j++ ){
      for( l=0; l< oN[q] ; l++ ){
        for( l_q=0; l_q < N[q] ; l_q++ ){
          m = j*oN[q]*N[q] + l_q*oN[q] + l;

          eta[m].re = 0; eta[m].im = 0;
          for( j_q=0; j_q < N[q] ; j_q++ ){
            k = j*oN[q]*N[q] + j_q*oN[q] + l;
            cos_jql = cos( theta * j_q*(l_q*oN[q] + l)*uN[q] );
            sin_jql = sin( theta * j_q*(l_q*oN[q] + l)*uN[q] );
            eta[ m ].re +=
              cos_jql * last_eta[k].re + sin_jql * last_eta[k].im;
            eta[ m ].im +=
              cos_jql * last_eta[k].im - sin_jql * last_eta[k].re;
          }
        }
      }
    }
  }


  // この時 eta のポイント先は a か malloc で確保した領域のどちらか不定
  for( j=0; j< NN ; j++ ) {
    a[j].re = eta[j].re /NN;
    a[j].im = eta[j].im /NN;
  }

  free( (void *)buf ); // メモリ解放
  return 0;

}


// 高速フーリエ変換 III(2 のべき数)
int FFourierTransform_bi(
  struct cmplx x[],
  struct cmplx a[],
  unsigned int p
){
  int q;
  unsigned int j, l, ll_q, NN, m, k0, k1 ;
  double theta;
  double cos_l, sin_l;
  struct cmplx *eta, *last_eta, *tmpp, *buf;
  unsigned int uN_q, oN_q;
  unsigned int j_msk, l_msk, ll_q_msk;

  NN = 1;
  for( q=0; q<(int)p; q++) NN *= 2;
  theta = 2.0*M_PI/NN;
```

```
  // 作業領域確保
  if(
    NULL == ( buf = (struct cmplx *)malloc( sizeof(struct cmplx)*NN ) )
  ){
    printf("ERROR\n");
    return 1;
  };
  eta = a; last_eta = buf;

  // eta^p 並べ替え
  for( l=0; l<NN; l++){
    j=0 ;
    oN_q = 1; uN_q = NN/2;
    for( q=(int)p-1; q>=0; q-- ){
      if( oN_q & l ) j += uN_q;
      oN_q *= 2; uN_q /= 2;
    }
    eta[l] = x[j];
  }

  j_msk = NN-1; // all 1
  ll_q_msk = 0; // all 0
  l_msk = 0;    // all 0

  oN_q = 1; uN_q = NN/2;
  for( q = (int)p-1; q >= 0; q--){
    tmpp = last_eta; last_eta = eta; eta = tmpp;

    j_msk    = j_msk*2;       // 0 埋めシフト
    ll_q_msk = ll_q_msk*2 + 1;// 1 埋めシフト

    for( m=0; m<NN ; m++ ){

      j    = m & j_msk;
      l    = m & l_msk;
      ll_q = m & ll_q_msk;

      k0 = j        + l;
      k1 = j + oN_q + l;

      cos_l = cos( theta * ll_q * uN_q );
      sin_l = sin( theta * ll_q * uN_q );

      eta[m].re = last_eta[k0].re
        + cos_l * last_eta[k1].re + sin_l * last_eta[k1].im;
      eta[m].im = last_eta[k0].im
        + cos_l * last_eta[k1].im - sin_l * last_eta[k1].re;
    }

    oN_q *= 2;
    uN_q /= 2;
    l_msk   = l_msk*2 + 1; // 1 埋めシフト

  }

  // この時 eta のポイント先は a か malloc で確保した領域のどちらか不定
  for( j=0; j< NN ; j++ ) {
    a[j].re = eta[j].re /NN;
    a[j].im = eta[j].im /NN;
  }

  free( (void *)buf ); // メモリ解放
  return 0;

}



int main(void){
  const double NN = 4096*4;
  unsigned int N1 = 128, N2 = 128, p=14;
  unsigned int N[] = { 2,2,2,2,2,2,2,  2,2,2,2,2,2,2 };
  int j;
  double err;
```

```
  struct cmplx *a, *x, *b;
  clock_t begin_t, end_t;

  // 作業領域確保
  if(
    NULL == ( a = (struct cmplx *)malloc( sizeof(struct cmplx)*NN ) )
    || NULL == ( x = (struct cmplx *)malloc( sizeof(struct cmplx)*NN ) )
    || NULL == ( b = (struct cmplx *)malloc( sizeof(struct cmplx)*NN ) )
  ){
    fprintf(stderr, "could not get memory!");
    return 1;
  };


  // サンプルデータの生成
  for( j=0; j<NN/2; j++ ){  x[j].re = 1; x[j].im = 0; }
  for( j=NN/2; j<NN; j++ ){ x[j].re = 0; x[j].im = 0; }
  x[NN/2].re = 0.5; x[NN/2].im = 0 ;


  // 実験及び出力
  begin_t = clock();
  FourierTransform(  x, a, NN );
  end_t = clock();
  printf("#CLOCKS_PER_SEC %f\n", CLOCKS_PER_SEC );
  printf("#NORMAL time = %10.30f \n", (double)end_t - begin_t );

  begin_t = clock();
  FFourierTransform_NN(  x, b, N1, N2 );
  end_t = clock();
  err = 0;
  for( j=0; j<NN; j++ ){
    err += ( a[j].re - b[j].re ) * ( a[j].re - b[j].re )
        +( a[j].im - b[j].im ) * ( a[j].im - b[j].im );
  }
  printf("#I   time = %10.30f error = %f\n", (double)end_t - begin_t, err );

  begin_t = clock();
  FFourierTransform(  x, b, N, p );
  end_t = clock();
  err = 0;
  for( j=0; j<NN; j++ ){
    err += ( a[j].re - b[j].re ) * ( a[j].re - b[j].re )
        +( a[j].im - b[j].im ) * ( a[j].im - b[j].im );
  }
  printf("#II  time = %10.30f error = %f\n", (double)end_t - begin_t, err );

  begin_t = clock();
  FFourierTransform_bi(  x, b, p );
  end_t = clock();
  err = 0;
  for( j=0; j<NN; j++ ){
    err += ( a[j].re - b[j].re ) * ( a[j].re - b[j].re )
        +( a[j].im - b[j].im ) * ( a[j].im - b[j].im );
  }
  printf("#III time = %10.30f error = %f\n", (double)end_t - begin_t, err );

  free( (void *)a ); free( (void *)x ); free( (void *)b );
  return 0;
}
```