Fortran 実習

金子敏明

2009/08/31

目 次

1 はじめに

Fortran は古いプログラミング言語であるが、以下の点に優れ、今でも数値計算のためには他の言語に勝っている。

- 過去の蓄積が豊富であること。例えば

  `http://www.netlib.org/`

  を見よ。

- 互換性がよい。システムやコンパイラの方言を使っていなければ 30 年前のプログラムは今も動く。まともなシステムなら Fortran コンパイラは存在している。

- 浮動小数を使う計算では常に精度の問題を気にする必要がある。それをチェックするには精度を変えて計算してみるのがよい。Fortran の標準では精度は単精度と倍精度があり、コンパイラによってはオプションを指定するだけで、同じプログラムをソースコードの変更なしに 4 倍精度に変換してくれるものもある。

- 言語が C などに比べてある意味で単純であるため、コンパイラの最適化が効きやすい。

- 複素数演算が標準でサポートされている (散乱振幅は複素数!)。これには複素数版の初等関数も含まれている。

○ この実習では、C 言語でプログラムの作成経験があるものとして話を進める。C 以外でも一定のプログラムの経験があれば理解できると期待している。

効率的な学習法として、次の手順が実用的であろう。

1. 他のプログラムを理解する。
2. 他のプログラムを変形してみる。
3. 他のプログラムをまねて自分のプログラムを作る。
4. 文法書等の詳細を調べる。

○ この演習では、文法を精密に説明するわけではない。ある程度使えるようになるための主要な機能を紹介することを目的とし、文法も簡略化してある。Fortran の文法については、以下の URL が参考になるだろう。

```
1  http://www-aos.eps.s.u-tokyo.ac.jp/~takagi/f77-enshu/index.html
2  http://www.star.le.ac.uk/~cgp/prof77.html
3  http://www.ibiblio.org/pub/languages/fortran/
```

○ fortran 77, fortran 90, fortran 95, fortran 2003 などいくつかの規格がある。ここでは保守的だが過去の蓄積の多い fortran 77 を使う。

- コンパイラとして g77 を前提にする。

- ただし信頼性が低いので、最終的な結果に対しては他のコンパイラとの比較が必要。

- 他のコンパイラとの比較は、プログラムのデバッグにも有効である。

1.1 プログラムのチェックについて

Fortran に限らないが、プログラムが出来上がってコンパイラエラーも出ず異常終了しない場合でも、プログラムの答えが正しい保証は何もない。また、簡単な場合や特殊な場合に答えが正しくても、複雑な場合になると答えがおかしくなることも多い。

このため出来上がったプログラムに対して計算結果のチェックは不可欠である。チェックの原則は

- 答が満たすべき条件が成り立っているかをチェックする。
- 正しい答えが分かっている場合について計算する。
- ほかに計算の答えがある場合にはそれと比較する。
- ほかに計算の答えがない場合には、違う方式で計算するプログラムを用意して比較する。

1.2 デバッグについて

- コンパイラがエラーを出す場合

  原則はエラーメッセージをよく読んで、よく考えることである。

  エラーメッセージは間違いを教えてくれるわけではなく、コンパイラが変な部分を見つけた時、どう変だと思ったかを知らせるものである。間違いの原因が文法的に正しい書き方で書いてあれば、コンパイラは間違いとはみなさずに、矛盾が現れたところでメッセージを出す。したがって、エラーメッセージをよく読み、どのような矛盾が発生したかを理解し、そのような矛盾の原因を追求することが必要である。

- 実行時に異常終了する場合

  コンパイラオプションのうち最適化に関するものをすべて外し、かわりに 「`-g`」をつけて再コンパイルし実行する。

  実行はデバッガ (コンパイラにより違う。g77 なら gdb) の下で実行する。デバッガは異常終了するとそこで止まるので、デバッガのコマンドを使ってソースプログラムのどこで異常終了したかを確認する。

  必ずしもデバッガが示す箇所が悪いとは限らないが、多くの場合その付近が怪しい。デバッガが示す箇所が subroutine/function ならばその呼出側に問題のあることも多い。特に、システムライブラリ中で異常終了す場合には、たいがいその呼出側に引数の型や大きさなどが食い違う場合などの問題がある。

  プログラムを読んでも分からないときは、`write` 文を挿入して関連する変数の値を出力しチェックする。

- 実行は完了するが答えがおかしい場合

  原則は、計算の途中経過を表示してみることである。

  このためにも、正しい答えの入力と出力が必要である。また、計算途中の変数間で満たすべき条件があるならば、その条件が満たされているかをチェックし、満たされていないとき関係する変数を表示してプログラムが止まるようにしておくとよい。ただし条件によっては、計算の誤差によりその条件が満たされなくなる場合があるので注意が必要である。

- 代表的なバグの原因
  - 変数のスペルの間違い
    `implicit none` であれば見つかりやすい。
  - 変数の型が間違っている。
    特に型 `real` と `real*8` に注意
    - 定数：`0.3` と `0.3d0` など
    - 変数：暗黙の型に注意 (`implicit`)
    - 関数：組み込み関数 `real` と `dble` など
  - 配列の範囲を越えて代入する
    何でも起こりうる (コンパイル時のエラー、異常終了、正常終了するが答えがおかしい)。
  - subroutine/function へ渡す変数、値の型や配列の大きさが定義と合わない
    何でも起こりうる (コンパイル時のエラー、異常終了、正常終了するが答えがおかしい)。
  - 変数に値を入れずに使っている
    メモリ上のゴミデータを使うことになる。コンパイラオプションによっては、最初に全ての変数を 0 にする場合もある。
  - `common block`
    `common block` の名前がぶつかっている場合
    `common block` 上の変数の型や要素数が合わない場合。

2 簡単な例

数表を作る。以下、Frotran program と、対応する C program の例を示す。

2.1 Fortran program

```
 1  *   Example 1
 2  *---+----1----+----2----+----3----+----4----+----5----+----6----+----7--
 3        implicit none
 4        integer  n, j
 5        real*8   x, dn, pi, gf
 6
 7        pi = 2*asin(1.0d0)
 8        write(*,*) 'pi = ', pi
 9
10        write(*,*) 'enter n'
11        read(*,*) n
12
13        dn = n
14        gf = pi/dn
15
16        do j = 0, n
17            x = j * gf
18            write(*,*) x, sin(x)
19        enddo
20
21        end
```

2.2 C program

```
 1  /*
 2   * Example 1
 3   */
 4  #include <stdio.h>
 5  #include <stdlib.h>
 6  #include <math.h>
 7
 8  int main() {
 9      int n, j;
10      double x, dn, pi, gf;
11
12      pi = 2*asin(1.0);
13      printf("pi = %g\n", pi);
14
15      printf("enter n\n");
16      scanf("%d", &n);
17
18      dn = n;
19      gf = pi/dn;
20
21      for(j = 0; j <= n; j++) {
22           x = j * gf;
23           printf("%g %g\n", x, sin(x));
24      }
25
```

```
26      return 0;
27  }
```

2.3 Makefile

この例では Fortran program は `fex1.f` という file に、C program は `cex1.c` というファイルに作ってあるものとする。

```
 1  # CC      = cc
 2  CFLAGS  = -Wall -g
 3  # FC      = g77
 4  # FFLAGS  = -Wall -g
 5  # FC      = fort77
 6  FFLAGS  = -g
 7
 8  all: fex1 cex1
 9          ./fex1
10
11  fex1: fex1.o
12          $(FC) $(FFLAGS) -o fex1 fex1.o
13
14  cex1: cex1.o
15          $(CC) $(CFLAGS) -o cex1 cex1.o -lm
16
17  clean:
18          rm -f *.o fex1 cex1 a.out
19          rm -f *core *.core t tt t[0-9] *.bak *~
```

8, 12, 15, 16 行目のように “:” のある行の下の行の先頭は空白ではなく Tab である。`Makefile` では空白と Tab とは違うので注意すること。

2.4 解説

- 書き方

C	Fortran
大文字小文字を区別する	大文字小文字は区別しない
空白は単語の区切り	空白は無視する
書く場所は自由	コラムで決まりあり (1:コメント; 2–4:行番号; 5:継続; 6–72:本文)
文は ; で区切る。改行は空白と同等	文は行毎に分かれる。長い文は、2 行目以降 5 コラム目に何か文字を書いて上の行からつながっていることを示す。

- コメント

C		Fortran	
番号	説明	番号	説明
1–3	`/*` と `*/` とでくくる	1–2	1 コラム目に `*` または `c`

- 準備

C		Fortran	
番号	説明	番号	説明
4–6	`#include <stdio.h> ...`	—	不要。ただし file を include する機能はある。
8	`main`：OS からの呼び出し場所	—	`program` 文。なくてよい
—	変数名と型の関係:なし	3	標準ではあり。`implicit none` でなしにできる。

Fortran で `implicit none` をつけないと、宣言のない変数名の先頭の文字で型が決まる。

- `a-h, o-z`：単精度浮動小数
- `i-n`：整数

- 宣言

C		Fortran	
番号	説明	番号	説明
9	`int`：整数	4	`integer`：整数
10	`double`：倍精度浮動小数	5	`real*8`：倍精度浮動小数

`real*8` の 8 は 8 Bypte の意味。Compliler によっては `double precision` と書くことを要求される事がある (“8 Byte” は machine dependent であるため)。

- 定数

C		Fortran	
番号	説明	番号	説明
9	`1.0`：`double` 型	4	`1.0d0`：`real*8` 型。1.0 は単精度

`1.0d0/10.0d0` は 2 進法では巡回小数になるので、`0.1d0` (こちらは有限小数) とは違う。

- 型変換：違う型の数が混じる場合の型変換の原則は、整数と浮動小数 ⇒ 浮動小数

C		Fortran	
番号	説明	番号	説明
—	明示：`((double) 5)` (= `5.0`)	—	明示：`dble(5)` (= `5.0d0`)
—	暗黙：`5*0.5` (= `2.5`)	—	暗黙：`5*0.5d0` (= `2.5d0`)
18	暗黙：`int` ⇒`double`	14	暗黙：`integer` ⇒`real*8`

Fortran で `real(5)` は単精度。

- 標準の関数：(sin, cos, tan, log, exp など。`asin` は sin の逆関数 arcsin)

C	Fortran
あらかじめ `#include <math.h>` を書いておく。コンパイル後リンク時に `-lm` をつける。	標準で装備されている

- 四則演算 (`+`,`-`,`*`,`/`), は C と Fortran で同じ。べき乗 x^y は

C	Fortran
`pow(x, y)`	`x**y`

演算子の優先順位は常識的に考えればよいが、ややこしいもの、自信がない場合には「(」と「)」で構造をはっきりさせる。

- 代入文 :

C	Fortran
「a = 式」。「a = b = 式」も可能。	「a = 式」。「a = b = 式」は不可。

- 繰り返し

C		Fortran	
番号	説明	番号	説明
21–24	for(...) { ... }	16–19	do ... enddo

例題では、いずれも変数 j を 0 から n まで 1 づつ増やしながら繰り返す。

- 入力

C		Fortran	
番号	説明	番号	説明
13,23	scanf()	8,18	read(*,*)

scanf での文字列入力は避けるべき。fgets を使う。

- 出力

C		Fortran	
番号	説明	番号	説明
13,23	printf()	8,18	write(*,*)

2.5 実行

手動でコンパイル実行

1. g77 -o 名前 ファイル名
2. ./名前

ファイルがいくつもある時、ライブラリをリンクするときなどは複雑になる。この場合、Makefile により実行するほうが効率が良い。

1. Makefile に手順を書いておく。
2. make

Makefile の書き方で、様々な作業の自動化ができる。

2.6 問題

1. 例題を実行せよ。
2. 例題をわざと間違えて、どうなるかを見よ (終わったらもとに戻すこと)。
   - 変数名を間違える。

- enddo を end do とする (これは問題ない。なぜか)。
- enddo のスペルを間違える。
- enddo を endif と間違える。
- enddo の行をコメントアウトする (その行の先頭を空白から c または * にかえる)。
- sin のスペルを間違える。
- 13 行め dn = n を dn = -1 と間違える。
- 13 行め dn = n を dn = 0 と間違える。

3. 1/10, 1.0/10, 1/10.0, 1/10.0d0, 1.0d0/10.0, 0.1d0, 1/dble(10), 1/real(10) を計算して表示するプログラムを作れ。また、上のそれぞれの計算結果を 10 倍して、それから 1 を引いたものを表示せよ。

4. 対数表を作れ。

5. 質量が 1 GeV の 2 つの粒子の衝突を考える。実験室系のエネルギーを変化させて対応する重心系のエネルギーとの対照表をつくれ。

6. 次の 5 行はそれだけで 1 プログラムになっている。(先頭の行はコメント。続く行のコラム位置に注意。implicit none がないので出てくる変数はあらかじめ宣言されているとみなされる。do 10 i = 1, 5 ... 10 continue は do i = 1, 5 ... enddo と同等)

```
*---+----1----+----2----+----3----+----4
      do 10 i = 1, 5
        write(*,*) i
   10 continue
      end
```

この do 文で 「,」 をわざと 「.」に間違えて

```
      do 10 i = 1. 5
```

のようにして、コンパイル実行せよ (http://ja.wikipedia.org/wiki/FORTRAN 参照)。

3 配列と subroutine

ここでは Lorentz 変換を例に取り、行列の演算を行う (付録参照)。

3.1 Fortran program

```
 1  *   Example of Lorentz transformation
 2  *---+----1----+----2----+----3----+----4----+----5----+----6----+----7--
 3        implicit none
 4
 5        real*8 pi, phi, psi, theta
 6        real*8 v(3)
 7        real*8 al(0:3, 0:3)
 8        real*8 p(0:3), pb(0:3)
 9        real*8 q(0:3), qb(0:3)
10  *---+----1----+----2----+----3----+----4----+----5----+----6----+----7--
11
12  *  -- constant
13        pi = 2*asin(1.0d0)
14
15  *  -- input data
16        phi   =   0.30d0 * pi
17        psi   =   0.40d0 * pi
18        theta =   0.50d0 * pi
19        v(1)  = 0.2d0
20        v(2)  = 0.3d0
21        v(3)  = 0.5d0
22
23  *  -- four momentum for test
24        p(0)  = 1
25        p(1)  = 0
26        p(2)  = 0
27        p(3)  = 0
28
29        q(0)  = 4
30        q(1)  = 1
31        q(2)  = 2
32        q(3)  = 3
33
34  *  -- construct the matrix representation 'al' of Lorentz transformation
35
36        call lorentz(v, psi, theta, phi, al)
37
38  *  -- Lorentz transformation of four vectors
39
40        call prodmv(al, p,  pb)
41
42        call prvect('p', p)
43        call prvect('al*p', pb)
44
45        call prodmv(al, q,  qb)
46
47        call prvect('q', q)
48        call prvect('al*q', qb)
49
50        end
51  *---+----1----+----2----+----3----+----4----+----5----+----6----+----7--
```

```
 52 *   Construct the matrix of Lorentz transformation : al = boost * rotation
 53 *
 54 *     Input :
 55 *       real*8 v(3) : Space components of transformed four vector
 56 *                     of (1,0,0,0).
 57 *       real*8 psi, theta, phi : Eular angle.
 58 *
 59 *     Output :
 60 *       real*8 al(0:3, 0:3) : resulting matrix.
 61 *
 62        subroutine lorentz(v, psi, theta, phi, al)
 63        implicit none
 64
 65 *  -- arguments
 66        real*8 v(3), phi, psi, theta, al(0:3, 0:3)
 67
 68 *  -- for check/debug
 69        logical check
 70        parameter (check = .true.)
 71 c      parameter (check = .false.)
 72
 73 *  -- local variables
 74        real*8 rot1(0:3, 0:3),  rot2(0:3, 0:3), rot3(0:3, 0:3)
 75        real*8 rot(0:3, 0:3),   rot21(0:3, 0:3)
 76        real*8 boost(0:3, 0:3), g(0:3), br1(0:3, 0:3)
 77        real*8 vsq, vd
 78
 79        integer k, k1, k2, j
 80
 81 *  -- clear matrix
 82        do k = 0, 3
 83          do j = 0, 3
 84            rot1(j,k)  = 0
 85            rot2(j,k)  = 0
 86            rot3(j,k)  = 0
 87            boost(j,k) = 0
 88          enddo
 89        enddo
 90        do k = 0, 3
 91          rot1(k,k)  = 1
 92          rot2(k,k)  = 1
 93          rot3(k,k)  = 1
 94          boost(k,k) = 1
 95        enddo
 96
 97        rot1(1,1) =   cos(phi)
 98        rot1(1,2) = - sin(phi)
 99        rot1(2,1) =   sin(phi)
100        rot1(2,2) =   cos(phi)
101
102        rot2(1,1) =   cos(theta)
103        rot2(1,3) = - sin(theta)
104        rot2(3,1) =   sin(theta)
105        rot2(3,3) =   cos(theta)
106
107        rot3(1,1) =   cos(psi)
108        rot3(1,2) = - sin(psi)
109        rot3(2,1) =   sin(psi)
```

```
      rot3(2,2) =   cos(psi)

      vsq = v(1)**2 + v(2)**2 + v(3)**2
      vd = sqrt(1+vsq)
      boost(0, 0) = vd
      do k = 1, 3
        boost(0, k) = v(k)
        boost(k, 0) = v(k)
      enddo
      do k = 1, 3
        do j = 1, 3
          boost(j, k) = (vd-1)*v(j)*v(k)/vsq
        enddo
        boost(k,k) = boost(k,k) + 1
      enddo

      call prodmm(rot2,  rot1, rot21)
      call prodmm(rot21, rot3, rot)
      call prodmm(boost, rot,  al)

*  -- for check
      if(check) then
        g(0) =  1
        g(1) = -1
        g(2) = -1
        g(3) = -1

        do k1 = 0, 3
          do k2 = 0, 3
            br1(k1, k2) = 0
            do j = 0, 3
              br1(k1, k2) = br1(k1, k2) + g(j)*al(j,k1)*al(j,k2)
            enddo
          enddo
        enddo

        call prmat('rot1',  rot1)
        call prmat('rot2',  rot2)
        call prmat('rot3',  rot3)
        call prmat('boost', boost)
        call prmat('rot2*rot1', rot21)
        call prmat('rot3*rot2*rot1', rot)
        call prmat('al = boost*rot3*rot2*rot1', al)
        call prmat('g : check', br1)
      endif

      return
      end
*---+----1----+----2----+----3----+----4----+----5----+----6----+----7--
*   matrix * matrix : r = a*b
*
*   Input:
*     real*8 a(0:3, 0:3), b(0:3, 0:3) : input matrices
*
*   Ouput:
*     real*8 r(0:3, 0:3) : output matrix
*                          memory space of 'r' should not be overlapped with
*                          'a' nor 'b'.
```

```
168  *
169        subroutine prodmm(a, b, r)
170        implicit none
171        real*8 a(0:3, 0:3), b(0:3, 0:3), r(0:3, 0:3)
172        integer k1, k2, j
173
174        do k1 = 0, 3
175          do k2 = 0, 3
176            r(k1,k2) = 0
177            do j = 0, 3
178              r(k1,k2) = r(k1,k2) + a(k1,j)*b(j,k2)
179            enddo
180          enddo
181        enddo
182        return
183        end
184  *---+----1----+----2----+----3----+----4----+----5----+----6----+----7--
185  *   matrix * vector : r = a*v
186  *
187  *   Input:
188  *     real*8 a(0:3, 0:3) : input matrix
189  *     real*8 v(0:3)      : input vector
190  *
191  *   Output:
192  *     real*8 r(0:3)      : output vector
193  *                          memory space of 'r' should not be overlapped with
194  *                          'a' nor 'v'.
195  *
196        subroutine prodmv(a, v, r)
197        implicit none
198        real*8 a(0:3, 0:3), v(0:3), r(0:3)
199        integer k, j
200
201        do k = 0, 3
202          r(k) = 0
203          do j = 0, 3
204            r(k) = r(k) + a(k,j)*v(j)
205          enddo
206        enddo
207        return
208        end
209  *---+----1----+----2----+----3----+----4----+----5----+----6----+----7--
210  *   Print matrix 'ltr' with message 'str'
211  *
212  *   Input
213  *     character*(*) str    : message character string
214  *     real*8 ltr(0:3, 0:3) : matrix to be printed
215  *
216        subroutine prmat(str, ltr)
217        implicit none
218        character*(*) str
219        real*8 ltr(0:3, 0:3)
220
221        integer j, k
222
223        write(*,*) '+++ ', str, ' +++'
224        do j = 0, 3
225          write(*,'(1x,4f10.6)') (ltr(j, k), k=0,3)
```

```
226       enddo
227
228       return
229       end
230  *---+----1----+----2----+----3----+----4----+----5----+----6----+----7--
231  *   Print four vector 'lv' with message 'str'
232  *
233  *   Input
234  *     character*(*) str    : message character string
235  *     real*8 lv(0:3)       : four vector to be printed
236  *
237       subroutine prvect(str, lv)
238       implicit none
239       character*(*) str
240       real*8 lv(0:3)
241
242       real*8 msq
243       integer j
244
245       msq = lv(0)**2 - lv(1)**2 - lv(2)**2 - lv(3)**2
246       write(*,*) '+++ ', str, ' +++'
247       write(*,'(1x,4f10.6, a, f10.6)') (lv(j), j=0,3),' m^2 =',msq
248
249       return
250       end
```

3.2 C program

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3  #include <math.h>
 4
 5  #define True  (1)
 6  #define False (0)
 7
 8  /* for check/debug */
 9  #define CHECK True
10
11  /* prototypes */
12  void lorentz(double v[4],double psi,double theta,double phi,double al[4][4]);
13  void prodmm(double a[4][4], double b[4][4], double r[4][4]);
14  void prodmv(double a[4][4], double v[4], double r[4]);
15  void prmat(char *str, double ltr[4][4]);
16  void prvect(char *str, double lv[4]);
17
18  /*---------------------------------------------------------------*/
19  int main(int argc, char **argv)
20  {
21      double pi, phi, psi, theta;
22      double v[4];
23      double al[4][4];
24      double p[4], pb[4];
25      double q[4], qb[4];
26
27      /* constant */
28      pi = 2*asin(1.0e0);
29
```

```
30      /* input data */
31      phi   =   0.30e0 * pi;
32      psi   =   0.40e0 * pi;
33      theta =   0.50e0 * pi;
34      v[1]  = 0.2e0;
35      v[2]  = 0.3e0;
36      v[3]  = 0.5e0;
37
38      /* four momentum for test */
39      p[0]  = 1;
40      p[1]  = 0;
41      p[2]  = 0;
42      p[3]  = 0;
43
44      q[0]  = 4;
45      q[1]  = 1;
46      q[2]  = 2;
47      q[3]  = 3;
48
49      /* construct the matrix representation "al" of Lorentz transformation */
50      lorentz(v, psi, theta, phi, al);
51
52      /* Lorentz transformation of four vectors */
53
54      prodmv(al, p,  pb);
55
56      prvect("p", p);
57      prvect("al*p", pb);
58
59      prodmv(al, q,  qb);
60
61      prvect("q", q);
62      prvect("al*q", qb);
63
64      return 0;
65  }
66
67  /*-------------------------------------------------------------------
68   *   Construct the matrix of Lorentz transformation : al = boost * rotation
69   *
70   *     Input :
71   *       double v[3) : Space components of transformed four vector
72   *                     of (1,0,0,0).
73   *       double psi, theta, phi : Eular angle.
74   *
75   *     Output :
76   *       double al[4][4] : resulting matrix.
77   */
78  void lorentz(double v[4], double psi, double theta, double phi,
79               double al[4][4])
80  {
81      /* local variables */
82      double rot1[4][4], rot2[4][4], rot3[4][4];
83      double rot[4][4],  rot21[4][4];
84      double boost[4][4], g[4], br1[4][4];
85      double vsq, vd;
86
87      int k, k1, k2, j;
```

```
 88
 89      /* clear matrix */
 90      for(k = 0; k < 4; k++) {
 91          for(j = 0; j < 4; j++) {
 92              rot1[j][k]  = 0;
 93              rot2[j][k]  = 0;
 94              rot3[j][k]  = 0;
 95              boost[j][k] = 0;
 96          }
 97      }
 98      for(k = 0; k < 4; k++) {
 99          rot1[k][k]  = 1;
100          rot2[k][k]  = 1;
101          rot3[k][k]  = 1;
102          boost[k][k] = 1;
103      }
104
105      rot1[1][1] =   cos(phi);
106      rot1[1][2] = - sin(phi);
107      rot1[2][1] =   sin(phi);
108      rot1[2][2] =   cos(phi);
109
110      rot2[1][1] =   cos(theta);
111      rot2[1][3] = - sin(theta);
112      rot2[3][1] =   sin(theta);
113      rot2[3][3] =   cos(theta);
114
115      rot3[1][1] =   cos(psi);
116      rot3[1][2] = - sin(psi);
117      rot3[2][1] =   sin(psi);
118      rot3[2][2] =   cos(psi);
119
120      vsq = v[1]*v[1] + v[2]*v[2] + v[3]*v[3];
121      vd  = sqrt(1+vsq);
122      boost[0][0] = vd;
123      for(k = 1; k < 4; k++) {
124          boost[0][k] = v[k];
125          boost[k][0] = v[k];
126      }
127      for(k = 1; k < 4; k++) {
128          for(j = 1; j < 4; j++) {
129              boost[j][k] = (vd-1)*v[j]*v[k]/vsq;
130          }
131          boost[k][k] += 1;
132      }
133
134      prodmm(rot2,  rot1, rot21);
135      prodmm(rot21, rot3, rot);
136      prodmm(boost, rot,  al);
137
138      /* for check */
139  #if CHECK
140          g[0] =  1;
141          g[1] = -1;
142          g[2] = -1;
143          g[3] = -1;
144
145          for(k1 = 0; k1 < 4; k1++) {
```

```
146             for(k2 = 0; k2 < 4; k2++) {
147                 br1[k1][k2] = 0;
148                 for(j = 0; j < 4; j++) {
149                     br1[k1][k2] += g[j]*al[j][k1]*al[j][k2];
150                 }
151             }
152           }
153 
154         prmat("rot1",  rot1);
155         prmat("rot2",  rot2);
156         prmat("rot3",  rot3);
157         prmat("boost", boost);
158         prmat("rot2*rot1", rot21);
159         prmat("rot3*rot2*rot1", rot);
160         prmat("al = boost*rot3*rot2*rot1", al);
161         prmat("g : check", br1);
162 #endif
163 }
164 
165 /*---------------------------------------------------------------
166  *   matrix * matrix : r = a*b
167  *
168  *   Input:
169  *     double a[4][4], b[4][4] : input matrices
170  *
171  *   Ouput:
172  *     double r[4][4] : output matrix
173  *                          memory space of "r" should not be overlapped with
174  *                          "a" nor "b".
175  */
176 void prodmm(double a[4][4], double b[4][4], double r[4][4])
177 {
178     int k1, k2, j;
179 
180     for(k1 = 0; k1 < 4; k1++) {
181         for(k2 = 0; k2 < 4; k2++) {
182             r[k1][k2] = 0;
183             for(j = 0; j < 4; j++) {
184                 r[k1][k2] += a[k1][j]*b[j][k2];
185             }
186         }
187     }
188 }
189 
190 /*---------------------------------------------------------------
191  *   matrix * vector : r = a*v
192  *
193  *   Input:
194  *     double a[4][4] : input matrix
195  *     double v[4]      : input vector
196  *
197  *   Output:
198  *     double r[4]      : output vector
199  *                          memory space of "r" should not be overlapped with
200  *                          "a" nor "v".
201  */
202 void prodmv(double a[4][4], double v[4], double r[4])
203 {
```

```
204     int k, j;
205
206     for(k = 0; k < 4; k++) {
207         r[k] = 0;
208         for(j = 0; j < 4; j++) {
209             r[k] += a[k][j]*v[j];
210         }
211     }
212 }
213
214 /*-----------------------------------------------------------------
215  *   Print matrix "ltr" with message "str"
216  *
217  *   Input
218  *     char   *str    : message character string
219  *     double ltr[4][4] : matrix to be printed
220  */
221 void prmat(char *str, double ltr[4][4])
222 {
223     int j, k;
224
225     printf("+++ %s +++\n ", str);
226     for(j = 0; j < 4; j++) {
227         for(k = 0; k < 4; k++) {
228             printf(" %9.6f", ltr[j][k]);
229         }
230         printf("\n ");
231     }
232 }
233
234 /*-----------------------------------------------------------------
235  *   Print four vector "lv" with message "str"
236  *
237  *   Input
238  *     char   *str    : message character string
239  *     double lv[4]       : four vector to be printed
240  */
241 void prvect(char *str, double lv[4])
242 {
243     double msq;
244     int j;
245
246     msq = lv[0]*lv[0] - lv[1]*lv[1] - lv[2]*lv[2] - lv[3]*lv[3];
247     printf("+++ %s +++\n ", str);
248     for(j = 0; j < 4; j++) {
249         printf(" %9.6f", lv[j]);
250     }
251     printf(" m^2 = %9.6f\n ", msq);
252 }
```

3.3 解説

- 配列の宣言

C		Fortran	
番号	説明	番号	説明
–	型の後ろに `v[4]` のように書く。この 4 は、 index を 0 から始めて 4 個分 memory を確保することを指定。したがって、使えるのは `v[0]`, `v[1]`, `v[2]`, `v[3]` の 4 個。Memory 上にはこの順で配置される。	5	型の後ろに `v(3)` のように書く。この 3 は、index を 1 から始めて 3 まで memory を確保することを指定。したがって、使えるのは `v(1)`, `v(2)`, `v(3)` の 3 個。Memory 上にはこの順で配置される。index を 0 から 3 までにしたければ `v(0:3)` とする。
–	2 次元配列は、`al[3][3]` と書く。Memory 上の配置は、先頭から `al[0][0]`, ..., `al[0][3]`, `al[1][0]`, ... `al[1][3]`, ... の順になる (後ろの index が先に動く)。	5	2 次元配列は、`al(0:3,0:3)` と書く。Memory 上の配置は、先頭から `al(0,0)`, ..., `al(3,0)`, `al(0,1)`, ... `al(3,1)`, ... の順になる (前の index が先に動く)。

- 配列への代入、引用

C		Fortran	
番号	説明	番号	説明
–	`v[1]`, `al[1][2]` などを普通の変数と思えば良い。	–	`v(1)`, `al(1,2)` などを普通の変数と思えば良い。

宣言された配列の範囲を越えて index が指定された場合には、どうなるか分からない。コンパイラがチェックできる場合もあるが、実行時に異常終了する場合や (運が悪ければ) 他の変数の値を変えてしまうこともある。

- 定数

C		Fortran	
番号	説明	番号	説明
–	マクロ「#define 定数名 値」とする。これはコンパイルの前に、プログラム中で「定数名」を「値」で文字列として置き換える。	69 – 70	定数の型の宣言をしておいて、「parameter (定数名 = 値)」とする。代入時には必要な型変換が行われる。

- 論理型

C	Fortran
長短の整数型や文字型を、0 に等しい (false) か等しくないか (true) を判定して使う。	宣言は `logical`、値は `.ture.` と `.false.` (前後にピリオドがつくことに注意。)

- 比較、論理演算

C	==	!=	<	<=	>	>=	!	&&	||
F	.eq.	.ne.	.lt.	.le.	.gt.	.ge.	.not.	.and.	.or.

ややこしい式は 「(」 と 「)」で構造をはっきりさせる。

- サブプログラム

C	Fortran
関数。値を返すかどうかは関数の値の型宣言と `return` 文の書き方で決まる。	値を返すなら `function`。値を返さないなら `subroutine`。

- subroutine (値を返さない関数) の定義

C		Fortran	
番号	説明	番号	説明
–	`void` 名前 (引数リスト) {...} で定義する。引数リストは「型名 変数名」を「,」で区切って並べたもの。配列の場合には、その宣言と同じ形式で書く (古い書き方についての説明は省略)。「...」は文を「;」で区切って並べる。	62	`subroutine` 名前 (引数リスト) ... `end` で定義する。引数リストは変数名を「,」で区切って並べたもの。別途宣言が必要。「...」は改行しながら文を並べる。

- subroutine (値を返さない関数) の呼び出し

C		Fortran	
番号	説明	番号	説明
–	関数 `a, b` が定義されているなら、「`a()`」, 「`b(1)`」などで呼び出す。	–	subroutine `a, b` が定義されているなら、「`call a()`」, 「`call b(1)`」などで呼び出す。

仮引数 (subroutine の定義に書いてある引数) と実引数 (subroutine を呼び出すときに入れる引数) の個数と対応するもの同士のデータの型、配列なら (原則として) 要素の個数を一致させる必要がある。これが一致しない場合、分かりにくいバグの原因となる。

- 引数の渡り方

C	Fortran
変数がスカラーならその値、配列なら先頭のアドレスが呼び出し側から、関数へ渡る。スカラー変数の内容を関数内で変更したい場合には引数の型をポインタ型にしてアドレスを渡す必要がある。	変数のアドレスが渡る。subroutine 中で仮引数の変数を書き換えると、呼び出し側の変数も内容が一緒に変わる。

Fortran では subroutine 中で内容を変更する変数に対応して渡す引数に定数を入れると、呼び出し側の memory を変更できないために実行時エラーが発生する場合がある (コンパイラに依存)。

- 文字列

C		Fortran	
番号	説明	番号	説明
–	型は `char *` (文字の配列、ポインタ: memory 領域は別に確保する場合) など。使い方により宣言の仕方が違う。	–	型は `character *(*)` (不定個の文字の列: 仮引数に使う場合) など。使い方により宣言の仕方が違う。
–	文字列定数は「`"`」(double quote) でくくる。	–	文字列定数は 「`'`」(single quote) でくくる。

他の宣言や文字列の切り貼りなど、例題以上の機能はややこしいので文法書を見ること。

- `write` 文の変数の並び

  Fortran で、`write` 文の後ろの `(ltr(j,k), k=0, 3)` は `ltr(j,0), ltr(j,1), ltr(j,2), ltr(j,4)` と並べたのと同等。

- 出力の書式

C		Fortran	
番号	説明	番号	説明
–	`printf` の第 1 引数で出力の形式を書いて第 2 引数以下で表示する変数や式を書く。	247	`write` の第 2 引数で出力の形式 (format と呼ぶ) を書いて「`)`」の後ろに表示する変数や式を書く。「`'(`」 と 「`)'`」 とではさまれる部分でそれぞれ、`1x` は 1 文字あける、`4f10.6` は普通の小数の形式で小数点以下 6 桁で 4 回表示せよ、`a` は文字列を表示することをあらわす。

Fortran の format は別の行に書いておいて、いくつかの `write` 文で共通に使うことも可能。1 つの format で何行にも渡る表を書くこともありうる。Format についてきちんと説明すると長くなるので、最初は他のプログラムを真似し、いずれは文法書で調べること。

3.4 問題

1. 2 つの運動量の内積が Lorentz 変換で変わらないことを、いくつかの例で計算して確かめよ。
2. 50 行目の `end` の前に `call lorentz(0, 0, 0, 0, 0)` の行を挿入して実行してみよ。(コンパイル時にエラーメッセージが出ても、実行形式が作られたら実行してみよ。)
3. (v, ψ, θ, ϕ) で決まる Lorentz 変換の行列の逆行列を求めるプログラムを作れ。(付録参照、ヒント：v で決まる boost の行列を $B(v)$, 回転の行列を R とすれば $(B(v)R)^{-1} = R^T B(-v)$ となる。)
4. 例題では時間反転と空間反転は入っていない。これらを追加せよ。

5. 例題では簡単な行列を作り、それらを順次かけていっている。行列の積をあらかじめ計算した式を使えば、もっと速くすることができる。これは、何度も Lorentz 変換行列を計算するときに重要となる。そのように書き換えよ。

6. 複素数 $z = x + i\,y$ に対して 2 行 2 列の行列 $\begin{bmatrix} x & y \\ -y & x \end{bmatrix}$ に対応させると、複素数の四則演算は行列の演算として計算しても良い。2 行 2 列の和、差、積、商の subroutine を用意して、このことを具体例で確かめるプログラムを作れ。(ヒント：$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$ の逆行列は $ad - bc \neq 0$ のとき、$\frac{1}{ad-bc}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$ で与えられる。)

4 Function と common block

ここでは台形公式による積分の例を見る。計算方法は付録を見ること。例として

$$\int_0^1 x^m \log^n(x)\, dx \tag{1}$$

を求める。これは $m > 0$ で部分積分により答えが求まる。しかし、$x = 0$ のまわりでは Taylor 展開できない、という意味であまりたちが良くない。

$n = m = 1$ の場合 $f(x) = x \log x$ の積分区間の上下限での値 $f(0), f(1)$ をみておく。$f(1) = 1 \cdot 0 = 0$ であるが、$f(0) = 0 \cdot \infty$ なので l'Hopital の定理から

$$\lim_{x \to +0} x \log x = \lim_{x \to +0} \frac{\log x}{1/x} = \lim_{x \to +0} \frac{1/x}{-1/x^2} = -\lim_{x \to +0} x = 0. \tag{2}$$

よって $f(0) = 0$。しかし、$f'(x) = \log(x) + 1$ なので $x \to +0$ で $f'(x) \to -\infty$ となり $x = 0$ のまわりで $f(x)$ を Taylor 展開することはできない。

4.1 Fortran program

```
 1  *   Integration by trapezoidal method
 2  *
 3  *       /1
 4  *       |  fnc(x) dx
 5  *       /0
 6  *
 7  *      fncl(x) = x**npx * log(x)**npl
 8  *      fnce(x) = x**npx * exp(npl * x)
 9  *
10  *    Parameters 'npx' and 'npl' are passed through common block 'idat'.
11  *
12  *---+----1----+----2----+----3----+----4----+----5----+----6----+----7--
13        implicit none
14
15  *  -- constants
16        double precision three, four
17        parameter (three=3, four=4)
18
19  *  -- the maximum iterations
20        integer    maxj
21        parameter (maxj = 25)
22
23  *  -- condition to finish iterations
24  *  -- require at least 'itmin' iterations
25        integer    itmin
26        parameter (itmin = 5)
27  *  -- require estimated error be less than 'ermin'
28        double precision ermin
29        parameter (ermin = 1.0d-8)
30
31  *  -- parameters passed to fncl and fnce for selecting integrand
32        integer         npx, npl
33        common    /idat/npx, npl
34
35  *  -- 'fnc' is a name of function
```

```
36       external fncl, fnce
37       double precision fncl, fnce
38       double precision fncla, fncea
39
40 *  -- local variables
41       integer          j, nc, n, maxn, ncc
42       double precision r0, r1, rp0, rp1, tv
43       double precision e0, e1, ep0, ep1, er0, er1, ans
44 *----------------------------------------------------------------------
45 *  -- power of x
46       npx  = 2
47
48 *  -- sweep parameters
49       maxn = 10
50       do n = -maxn, maxn
51
52 *    -- power of log/exp
53         npl = n
54
55 *    -- analytic result
56         if(npl.lt.0) then
57           tv = fncla()
58         else
59           tv = fncea()
60         endif
61
62 *    -- print messages
63         write(*,*) 'integration from 0 to 1 of'
64         if(npl.ge.0) then
65           write(*,*) '    x**',npx,' * log(x)**',npl
66         else
67           write(*,*) '    x**',npx,' * exp(',npl,'* x)'
68         endif
69         write(*,*) maxj,'iterations.'
70
71         write(*,*) '2**n,  integ,  err, err'
72
73 *    -- initial values for iterations
74         r0 = 0
75         nc = -1
76         e0 = 100
77         e1 = 100
78         ncc = -1
79
80 *     do iterations
81         do j = 0, maxj
82
83 *      -- save the previous results
84           rp0 = r0
85           rp1 = r1
86
87 *      -- estimation for 2**('nc'+1) points using previous result in 'r0'
88           if(n .gt. 0) then
89             call integt(nc, r0, fncl)
90           else
91             call integt(nc, r0, fnce)
92           endif
93
```

```
 94 *      -- for check
 95           if(j.ne.nc) then
 96             write(*,*) 'j = ', j, '!= nc=',nc
 97           endif
 98
 99 *      -- for acceleration
100           r1 = (four*r0 - rp0)/three
101
102 *      -- previous errors
103           ep0 = e0
104           ep1 = e1
105
106 *      -- new errors
107           e0 = abs(rp0-r0)
108           e1 = abs(rp1-r1)
109
110 *      -- ration of new/previous
111           if(ep0.eq.0) then
112             er0 = e0
113           else
114             er0 = e0/ep0
115           endif
116           if(ep1.eq.0) then
117             er1 = e1
118           else
119             er1 = e1/ep1
120           endif
121
122 *      -- output
123           write(*,'(1x,i2,g24.16,2g16.8,2f8.4)')
124      &          nc, r1, e0, e1, er0, er1
125
126 *      -- check convergence
127 *      -- require at least 'itmin' iterations and error < 'ermin'.
128           if(j.gt.itmin .and. abs(e1).lt.ermin) then
129             if(er1.eq.0) then
130               ans = r1
131               ncc = nc
132
133 *           -- skip subsequent iterations
134               goto 900
135
136 *         -- now error is growing ?
137             else if(er1.ge.1) then
138               ans = rp1
139               ncc = nc-1
140
141 *           -- skip subsequent iterations
142               goto 900
143             endif
144           endif
145
146 *      -- the result when iterations are terminated.
147           ans = r1
148
149 *    -- end of iterations
150        enddo
151
```

```
152    900   continue
153  *    -- result
154           write(*,'(1x,i2,g24.16)') ncc, ans
155
156  *    -- comparison with the analytic result.
157           write(*,'(i3, g24.16, a, a, g12.4)')
158       &        npl, tv, ' (analytic)', ' err=', abs((tv-ans)/tv)
159        enddo
160
161        end
162  *=====================================================================
163        subroutine integt(nc, r, fnc)
164  *
165  *     Input
166  *       nc  : the previous value of the iteration number
167  *       r   : previous result when nc >= 0.
168  *       fnc : function to be integrated
169  *     Output
170  *       r   : result of the iteration
171  *       nc  : updated iteration number
172
173        implicit none
174        integer  nc
175        double precision r
176        double precision fnc
177
178        double precision zero, one, two, half
179        parameter (zero = 0, one = 1, two = 2, half = one/two)
180
181        double precision dn, s, x
182        integer          np, j
183  *---------------------------------------------------------------------
184
185  *  -- the first call
186        if(nc.lt.0) then
187          nc = 0
188          r  = half*(fnc(zero) + fnc(one))
189  c       write(*,*) nc, x, r
190          return
191        endif
192
193  *  -- the number of points
194        np  = 2**nc
195
196  *  -- width
197        dn  = half/dble(np)
198
199  *  -- for the next call
200        nc  = nc + 1
201
202  *  -- summation of the values of function 'fnc(x)'
203        s = 0
204        do j = 1, np
205          x = (two*j-one)*dn
206          s = s + fnc(x)
207        enddo
208
209  *  -- the result
```

```
      r = half*r + dn*s

c     write(*,*) nc, x, r
      return
      end
*=====================================================================
      double precision function fncl(x)
*
*     fncl(x) = x**npx * log(x)**npl       for npl >= 0, x >= 0
*
*     Input
*       x : real number for which the function is calculated
*       npx, npl : selector of a function.  This parameter is passed
*                  through common block /idat/
*     Output
*       calculated function value

      implicit none

*  -- argument of the function
      double precision x

*  -- parameters
      integer    npx, npl
      common    /idat/npx, npl
*---------------------------------------------------------------------

*  -- check
      if(x.lt.0) then
        write(*,*) 'log(x), x = ',x,' < 0'
        stop 999
      endif

*    -- special point x=0 ( x*log(x)=0 for x=0 )
      if(npx.gt.0 .and. x.eq.0) then
        fncl = 0
      else
        fncl = x**npx * log(x)**npl
      endif

      return
      end
*=====================================================================
      double precision function fnce(x)
*
*     fnce(x) = x**npx * exp(npl * x)
*
*     Input
*       x : real number for which the function is calculated
*       npx, npl : selector of a function.  This parameter is passed
*                  through common block /idat/
*     Output
*       calculated function value

      implicit none

*  -- argument of the function
      double precision x
```

```
268
269   *  -- parameters
270          integer    npx, npl
271          common    /idat/npx, npl
272   *-----------------------------------------------------------------------
273   *  -- func = Power * exp
274          fnce = x**npx * exp(npl * x)
275
276          return
277          end
278   *=======================================================================
279          double precision function fncla()
280   *
281   *                         /1
282   *     analytic result of | dx fncl(x)
283   *                         /0
284   *
285          implicit none
286
287   *  -- parameters
288          integer    npx, npl
289          common    /idat/npx, npl
290
291   *  -- local vars
292          double precision d1, a1, t1
293          integer          k
294   *-----------------------------------------------------------------------
295          d1 = - 1/dble(npl)
296          a1 = d1
297          t1 = a1
298          do k = 1, npx
299            a1 = a1 * (npx-k+1) * d1
300            t1 = t1 + a1
301          enddo
302          fncla = - t1*exp(dble(npl)) + a1
303
304          return
305          end
306   *=======================================================================
307          double precision function fncea()
308   *
309   *                         /1
310   *     analytic result of | dx fnce(x)
311   *                         /0
312   *
313          implicit none
314
315   *  -- parameters
316          integer    npx, npl
317          common    /idat/npx, npl
318
319   *  -- local vars
320          double precision fpx, tv
321          integer          k
322   *-----------------------------------------------------------------------
323          fpx = 1/dble(npx+1)
324          tv = fpx
325          do k = 1, npl
```

```
326           tv = tv * (- k)*fpx
327         enddo
328         fncea = tv
329
330         return
331         end
```

4.2 解説

この例題の C 版は省略する。

- 変数のスコープ

  C では、変数は宣言の仕方や場所により、その変数名が有効になる範囲が決まっている。その範囲には、ブロック内、関数内、ファイル内、ファイルをまたがる、などのレベルがある。

  Fortran の変数は、基本的に関数・サブルーチン内でのみ有効である。あるメモリの領域を関数・サブルーチンにまたがって使いたい場合には common block を使う。例題の

  ```
  integer          npx, npl
  common     /idat/npx, npl
  ```

  は `idat` という名前の memory の領域を、メインプログラム、関数・サブルーチン間で共通に使うことを宣言している。ここでは、この先頭から `npx` と `npl` という名前をつけた 2 つの整数の領域として使うことにしている。この memory 領域の名前は勝手につけてよく、複数違うものを使っても良い。

  common block は、領域の先頭から順に型に応じた領域を取りながら、変数が割り当てられる。このため、他の関数やサブルーチンで使う場合には、必ずしも同じ名前を並べる必要はない。さらに同じ型の変数を使う必要もない。しかし、整数や実数など違う型の変数が混じった場合、その型の変数が使う領域の大きさは、OS やコンパイラ、およびオプションなどによって違う場合があり、注意する必要がある。また、ここでは説明しないが語境界エラーなどが発生する場合もある。したがって、複数の型のデータを共有したい場合には、次のようにするのが望ましい。

  – 型ごとに違う common block を作る。

  – 名前を統一するために、この部分は別ファイル、たとえば `common.h`、に書いておき、使う関数内で

    ```
    include 'common.h'
    ```

    で取り込む。

  これまでの例題では出てこなかったことであるが、以下補足する。

  – 分割コンパイル

    Fortran ではメインプログラム、サブルーチン、関数がコンパイルの単位になっている。これらは 1 つのファイルに書いてもよいが別々のファイルに書いてコンパイルし、後でリンクして実行形式を作っても良い。

    例えば `a.f` と `b.f` に分けてある場合

```
g77 -g -c a.o a.f
g77 -g -c b.o b.f
g77 -o ab a.o b.o
```

とすると ab に実行形式ができる。

make を使う場合には、Makefile に

```
ab: a.o b.o
      $(FC) $(FFLAGS) -o ab a.o b.o
```

とする (2 番目の行の先頭はタブ)。これは ab のためには a.o と b.o を作り、その後標準の Fortran コンパイラ ($(FC)) で標準のオプション ($(FFLAGS)) をつけて、a.o と b.o をリンクして ab を作れ、という意味である。a.f から a.o を作るルールは普通あらかじめ入っている。

このあと

```
make ab
```

とすれば実行形式 ab ができる。

4.3 問題

1. 例題を実行せよ。
2. 関数・サブルーチン毎にファイルを分割して、コンパイル実行せよ。
3. subroutine integet 中の write 文の書いてある行の先頭の c を、空白で置き換えて見よ。
4. 例題では npl が変わるが、npx も変えるように修正せよ (do を 2 重にし、do の次にくる変数名として新しい名前を作る)。
5. 例題の --- finish ? --- の下の if から endif のブロック全体を if(.false.) then と endif でくるるとどうなるか、実験せよ。

   この例題の様に収束性を自動的に判定するのは、危険な場合がある。ここでは誤差の振る舞いを見ているが、誤差は関数によって振る舞いが大きく異なる場合がある。関数の範囲を限るか、様々な関数で確認するまでは目視確認が必要である。
6. $\sin(\pi x)$ の値を返す新しい関数 fncs(x) を作り、$x = 0, \cdots, 1$ で積分するようにプログラムを書き換えよ。このとき、円周率 π は fncs 中で毎回計算するのではなく、メインプログラムで計算し、新しい common block を作って、それを通して fncs に渡すようにせよ。
7. 例題を 32 bit machine で実行するとき do j = 0, maxj で j を 31 より大きくしても意味がない。なぜか?
8. 新しいプログラムを作って、関数 $f(x)$ の微分を $(f(x+h) - f(x))/h$ で近似して求めるようにせよ。この時 h を $1/2^n$ として n を変化させて見よ。

5 ファイル入出力

ここでは簡単な file への入出力の例を示す。

5.1 Fortran program

```
 1  *   Simple program of output to a file.
 2  *---+----1----+----2----+----3----+----4----+----5----+----6----+----7--
 3        implicit none
 4
 5  *  -- constants
 6        character *(*) fname
 7        parameter (fname = 'ex4.dat')
 8        integer    ndata
 9        parameter (ndata = 20)
10
11  *  -- local vars
12        integer mode
13        real*8  x(0:ndata), y(0:ndata)
14  *---------------------------------------------------------------------
15        write(*,*) 'Read from or write to "', fname, '"'
16        write(*,*) 'read:0 or write:1 ?'
17        read(*,*)  mode
18
19        if(mode.eq.0) then
20          call datrd(fname, ndata, x, y)
21        else if(mode.eq.1) then
22          call gendat(ndata, x, y)
23          call datwrt(fname, ndata, x, y)
24        else
25          write(*,*) '*** illegal mode = ', mode
26          stop 999
27        endif
28
29        call plot(ndata, x, y)
30        end
31  *=====================================================================
32        subroutine datwrt(fname, ndata, x, y)
33        implicit none
34
35  *  -- args
36        character *(*) fname
37        integer  ndata
38        real*8   x(0:ndata), y(0:ndata)
39
40  *  -- constants and variables
41        integer    funit
42        parameter (funit = 1)
43        integer    ios
44  *---------------------------------------------------------------------
45        write(*,*) 'write date to ', fname
46        open(funit, file=fname, status='new',
47       &     form='unformatted', iostat=ios)
48        if(ios.ne.0) then
49          write(*,*) '*** error for writing data to new file: ios=',ios
50          write(*,*) '*** ', fname, ' may be exisiting.'
51          stop 999
```

```
52        endif
53        rewind(funit)
54        write(funit) x, y
55        close(funit)
56        return
57        end
58  *=====================================================================
59        subroutine datrd(fname, ndata, x, y)
60        implicit none
61
62  *  -- args
63        character *(*) fname
64        integer  ndata
65        real*8   x(0:ndata), y(0:ndata)
66
67  *  -- constants and variables
68        integer    funit
69        parameter (funit = 1)
70        integer    ios
71  *---------------------------------------------------------------------
72        write(*,*) 'read date from ', fname
73        open(funit, file=fname, status='old',
74       &     form='unformatted', iostat=ios)
75        if(ios.ne.0) then
76          write(*,*) '*** error for reading data from file: ios=',ios
77          write(*,*) '*** ', fname, ' may not be exisiting.'
78          stop 999
79        endif
80        rewind(funit)
81        read(funit) x, y
82        return
83        end
84  *=====================================================================
85        subroutine gendat(ndata, x, y)
86        implicit none
87
88  *  -- args
89        integer  ndata
90        real*8   x(0:ndata), y(0:ndata)
91
92  *  -- parameters for fncl
93        real*8   fncl
94        integer  npx, npl
95        common  /idat/npx, npl
96
97  *  -- vars
98        real*8   d
99        integer  k
100 *---------------------------------------------------------------------
101       npx = 1
102       npl = 1
103       d = 1/dble(ndata)
104       do k = 0, ndata
105         x(k) = k*d
106         y(k) = fncl(x(k))
107       enddo
108       return
109       end
```

```
110  *====================================================================
111        subroutine plot(ndata, x, y)
112        implicit none
113
114  *  -- args
115        integer  ndata
116        real*8   x(0:ndata), y(0:ndata)
117
118  *  -- vars
119        integer  j, k, ky
120        real*8   x0, x1, y0, y1, a0, a1
121  *--------------------------------------------------------------------
122        y0 = y(0)
123        y1 = y(0)
124        do k = 1, ndata
125          if(y0.gt.y(k)) then
126            y0 = y(k)
127            x0 = x(k)
128          endif
129          if(y1.lt.y(k)) then
130            y1 = y(k)
131            x1 = x(k)
132          endif
133        enddo
134        if(y0.eq.y1) then
135          write(*,*) 'constant y = ', y0
136          return
137        endif
138        write(*,*) 'min: f(',x0,') = ', y0, '  max: f(',x1,') = ', y1
139        a1 = 50/(y1-y0)
140        a0 = - y0*a1
141        do k = 0, ndata
142          ky = a0 + a1*y(k)
143          ky = min(max(0, ky), 50)
144          write(*,'(f8.4,'':'',52a)') x(k), (' ', j=0,ky-1), '*'
145        enddo
146        return
147        end
148  *====================================================================
149        double precision function fncl(x)
150  *
151  *     fncl(x) = x**npx * log(x)**npl       for npl >= 0, x >= 0
152  *
153  *     Input
154  *       x : real number for which the function is calculated
155  *       npx, npl : selector of a function.  This parameter is passed
156  *                  through common block /idat/
157  *     Output
158  *       calculated function value
159
160        implicit none
161
162  *  -- argument of the function
163        double precision x
164
165  *  -- parameters
166        integer    npx, npl
167        common    /idat/npx, npl
```

```
168  *-----------------------------------------------------------------------
169
170  *  -- check
171         if(x.lt.0) then
172           write(*,*) 'log(x), x = ',x,' < 0'
173           stop 999
174         endif
175
176  *    -- special point x=0 ( x*log(x)=0 for x=0 )
177         if(npx.gt.0 .and. x.eq.0) then
178           fncl = 0
179         else
180           fncl = x**npx * log(x)**npl
181         endif
182
183         return
184         end
```

5.2 解説

- File を読み書きするには、まず file を open する。その結果は unit 番号と呼ばれる整数値で示される。この unit 番号を通して実際の入出力が行われる。

  古くは unit=5 はパンチカード入力、unit=6 はラインプリンタ、unit=7 はパンチカード出力であった。現在の普通のシステムでは、unit=5 は標準入力 (C の stdin)、unit=6 は標準出力 (C の stdout) となっており、read(*,*) は read(5,*) と同等、write(*,*) は write(6,*) と同等である。システムやコンパイラによっては unit=7 にくせがあるかもしれない。

  いずれにしても、file に対する入出力を行う場合はこの 3 つの unit 番号を使うのは避けた方が良い。

- File の open には open 文を使うが、その文法にはシステム依存性がある場合がある。詳しくは、文法書とコンパイラのマニュアルなどを参照する。
- 出力用にファイルを新しく作って open するには、

  ```
  open(funit, file=fname, status='new', form='unformatted', iostat=ios)
  ```

  入力用に古いファイルを open するには、

  ```
  open(funit, file=fname, status='old', form='unformatted', iostat=ios)
  ```

- それぞれの項目は

  1. funit : unit 番号 (整数値) を指定する。
  2. status='new' : ファイルを新しく作る。
     ファイルが既にあるなら status='old' とする。
     どちらでも良い場合は status='replace' とする (これは fortran 90 以降の仕様なのでコンパイラによる)。
  3. form='unformatted' : データをバイナリで書き出す。
     数を文字列に変換して (例えば '1.5' は文字列) 書き出すには form='formatted' とする。

4. iostat=ios：エラーコードを整数変数 ios に入れる。0 ならエラーなし。

- close(funit)：半端になっている buffer を処理して、このファイルの入出力を終了する。
- rewind(funit)：テープの巻き戻しのイメージで、ファイルの先頭に移動する。
- write(funit)：バイナリでの出力。
  form='formatted' で open したら、write(funit,*) や書式をつけた形を使う。
- read(funit)：バイナリでの入力。
  form='formatted' で open したら、read(funit,*) や書式をつけた形を使う。

5.3 問題

1. 書き出す部分と読み出す部分とを分離して、2 つのプログラムに直せ。
2. open 文で form='unformatted' を form='formatted' に変えて、関連する read 文 write 文に必要な変更を加えよ。この時書き出されたファイルを cat コマンドまたは editor で中身を確認せよ。
3. 例題では、定数 ndata の値はファイルに書かれない。これを変更して、まず ndata の値だけをファイルに書き、続いて x，y の内容をファイルに書き出すようにせよ。

   これに対応して、ファイルを読むときにまず ndata の値を別の変数 ndata1 に読み込み、x，y を読み込む前に x，y の配列の大きさが読み込んだ ndata1 の値を越えないかをチェックせよ。

A Lorentz 変換

◦ 4 vector

時空 3+1 次元空間での 4 vector を考える。ここでは主にエネルギーと運動量を例にとる。

$$p = \{p^\mu\}_{\mu=0,1,2,3} = (p^0, \vec{p}). \tag{3}$$

◦ 内積

2 つの 4 vector p, q の内積

$$(p,q) = p^0 q^0 - (\vec{p},\vec{q}) = p^0 q^0 - \sum_{j=1,2,3} p^j q^j \tag{4}$$

を考える。これを 4×4 の行列 $g_{\mu\nu}$ を使って次のようにも書く。(ただし、重なる index は、index についての和を意味する summation convention を使って、和の記号 $\sum$ は省略する)

$$\begin{aligned}(g_{\mu\nu}) &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}, \\ p_\mu &= g_{\mu\nu} p^\nu, \\ (p,q) &= g_{\mu\nu} p^\mu q^\nu = p^\mu q_\mu = p_\mu q^\mu. \end{aligned} \tag{5}$$

特に $p = q$ なら

$$p^2 = (p,q) = (p^0)^2 - \vec{p}^2 \tag{6}$$

となる。p が粒子のエネルギー・運動量 vector ならば p^2 は質量 (ただし光速 c が 1 になるような単位系とする)。

◦ Lorentz 変換

任意の 4 vector の内積を変えないような、行列による座標変換 Lorentz 変換と呼ぶ。そのような行列を ${\Lambda^\mu}_\nu$ とする。つまり、任意の p,q に対して

$$p' = {\Lambda^\mu}_\nu p^\nu, \qquad q' = {\Lambda^\mu}_\nu q^\nu \tag{7}$$

と座標変換をすると

$$(p,q) = (p',q') \tag{8}$$

となる。p が粒子のエネルギー・運動量 vector ならば粒子の質量を変えないような (線形の) 座標変換を考えることになる。

式 (8) を書き換えると、

$$g_{\rho\sigma} p^\rho q^\sigma = g_{\mu\nu} p'^\mu q'^\nu = g_{\mu\nu} {\Lambda^\mu}_\rho {\Lambda^\nu}_\sigma p^\rho q^\sigma. \tag{9}$$

p, q は任意だったので、元の座標系のそれぞれの単位 vector に置き換えてみれば、

$$g_{\rho\sigma} = g_{\mu\nu} {\Lambda^\mu}_\rho {\Lambda^\nu}_\sigma \tag{10}$$

が得られる。逆に条件を満たす行列は Lorentz 変換になることはすぐにわかる。

この様な 2 つの座標変換を 2 回続けて行うと、その結果は別の 1 つの座標変換と考えることができる。また、1 つの座標変換に対して、それを元に戻す座標変換があることもわかる。この性質を「Lorentz 変換全体は群をなす」といい、全ての Lorentz 変換を集めたものを Lorentz 群と呼んでいる。これを $O(3,1)$ または $O(1,3)$ と書く (時空 $1+3$ に対応する回転群)。

行列 Λ にたいして座標変換を $p' = \Lambda p,\ q' = \Lambda q$ とする。g を $g_{\mu\nu}$ による、先にあげた行列とする。普通のユークリッド空間の内積を $\langle \cdot, \cdot \rangle$ と書くことにすれば内積がからわない条件 (8) は、

$$(p,q) = \langle p, gq \rangle = \langle p', gq' \rangle = \langle \Lambda p, g\Lambda q \rangle = \langle p, \Lambda^T g \Lambda q \rangle \tag{11}$$

となる。$p,\ q$ は勝手に取れるので、行列として

$$g = \Lambda^T g \Lambda \tag{12}$$

が成り立つことになる。ここで Λ^T は Λ の転置行列である。これは ${\Lambda^\mu}_\nu = {(\Lambda^T)_\nu}^\mu$ により先の条件を行列の書き方で書き直したものと等しい。

○ Lorentz 変換の分解

以下、任意の Lorentz 変換は、空間回転、Lorentz boost、空間反転および時間反転を合成すればよい、ということを確認し具体的な行列を求める。

○ 時間反転

時間反転は、時間成分の符号を反転させ、他は変えない。すなわち

$$\Lambda = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{13}$$

となる。

○ 空間反転

空間反転は、空間成分の符号を反転させ、他は変えない。すなわち

$$\Lambda = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \tag{14}$$

となる。

○ 回転

次に時間成分と空間成分とを混ぜないような Loretz 変換を考える。Lorentz 変換で 4 vector p が p' に移ったとする。

$$(p'^0)^2 - \vec{p}'^2 = (p^0)^2 - \vec{p}^2 \tag{15}$$

で、時間成分と空間成分とを混ぜないとしたので

$$(p'^0)^2 = (p^0)^2 \qquad \vec{p}'^2 = \vec{p}^2 \tag{16}$$

時間成分を見ると

$$p'^0 = \pm p^0 \tag{17}$$

$p'^0 = -p^0$ ならば時間反転である。空間成分は

$$\vec{p}'^2 = \vec{p}^2 \tag{18}$$

これは 3 次元空間の、反転と回転であることを意味している。よって R を回転を表す 3×3 行列とすると

$$\Lambda = \begin{bmatrix} \pm 1 & 0 & 0 & 0 \\ 0 & & & \\ 0 & & R & \\ 0 & & & \end{bmatrix} = \begin{bmatrix} \pm 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & & & \\ 0 & & R & \\ 0 & & & \end{bmatrix} \tag{19}$$

と書ける。このうち回転は $\det R = 1$ の場合で $\det R = -1$ ならば $R = -IR'$ (I は 3×3 の単位行列) とすると $-I$ は空間反転、R' は回転となる。

回転の行列 R を成分で書く。次の順序で 3 回続けて回転するとする。

1. z 軸の回りに x 軸を y 軸の方向に角度 ϕ $(0 \le \phi \le 2\pi)$ で回転し、次に
2. 新しい y 軸の回りに z 軸を x 軸の方向に角度 θ $(0 \le \theta \le \pi)$ で回転し、続いて
3. 新しい z 軸の回りに x 軸を y 軸の方向に角度 ψ $(0 \le \phi \le 2\pi)$ で回転した

とすると (Euler 角)

$$\begin{aligned} R &= \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} \cos\psi\cos\theta\cos\phi - \sin\psi\sin\phi & -\cos\psi\cos\theta\sin\phi - \sin\psi\cos\phi & \cos\psi\sin\theta \\ \sin\psi\cos\theta\cos\phi + \cos\psi\sin\phi & -\sin\psi\cos\theta\sin\phi + \cos\psi\cos\phi & \sin\psi\sin\theta \\ -\sin\theta\cos\phi & \sin\theta\sin\phi & \cos\theta \end{bmatrix} \end{aligned} \tag{20}$$

と書ける。この行列の行列式は $\det R = 1$ である。この 3 つの角 (ϕ, θ, ψ) を含む行列と空間反転とで任意の回転が表されることが知られている。

○ Boost

次に Lorentz boost を考える。第 j 成分が 1 で他の成分が 0 の基底 vector を e_j とする。

$$e_0 = (1,0,0,0), \quad e_1 = (0,1,0,0), \quad e_2 = (0,0,1,0), \quad e_3 = (0,0,0,1). \tag{21}$$

これらの vector が Lorentz 変換 Λ により e'_j に変換されたとする。これらの vector の内積は Lorentz 変換により変わらない。$(e'_i, e'_j) = (e_i, e_j) = \pm\delta_{i,j}$ ただし、符号は $i = j = 0$ ならプラス, そうでなければマイナスである。

Lorentz 変換を反転や回転を含まない z-軸方向の boost とする。e'_0 および e'_3 の空間成分は、z 成分以外 0 である。また回転を含まないとしたので x, y 成分を混ぜることはなく、e_1 の y 成分と e_2 の x 成分は 0 である。

$e'_0 = (\lambda, 0, 0, \rho)$ とおくと $(e'_0, e'_0) = 1$ から $\lambda = \pm\sqrt{1+\rho^2}$ が得られる。この符号はエネルギーの成分の符号に対応し、マイナスならば時間反転したことになる。いまプラスの解をとる (もし

くは $\rho \to 0$ で $\lambda \to 1$)。特に e'_0 は静止した質量 1 の粒子のエネルギー運動量を boost したものに等しい。その速度は運動量とエネルギーの比 $v = \rho/\lambda < 1$ で与えられるので、逆にとけば $\rho = v/\sqrt{1-v^2}$, $\lambda = 1/\sqrt{1-v^2}$ と表される。$e'_3 = (a, 0, 0, b)$ とおいて、$(e'_0, e'_3) = 0$, $(e'_3, e'_3) = -1$ を解けば $e'_3 = \pm(\rho, 0, 0, \lambda)$ が得られる。複号でマイナスのとき空間反転をかければ $e'_3 = (\rho, 0, 0, \lambda)$ となる。同様に e'_1, e'_2 について $(e'_i, e'_j) = (e_i, e_j)$ を解く。これから e'_1, e'_2 の時間成分と z-軸成分は 0 であることがわかる。また x, y 成分は混じらないとしたので、$e'_1 = e_1$, $e'_2 = e_2$ と変化しないことがわかる。以上の e'_j が得られるような Λ の行列表示は次のようになることがわかる。

$$\Lambda == \begin{bmatrix} \frac{1}{\sqrt{1-v^2}} & 0 & 0 & \frac{v}{\sqrt{1-v^2}} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \frac{v}{\sqrt{1-v^2}} & 0 & 0 & \frac{1}{\sqrt{1-v^2}} \end{bmatrix} \tag{22}$$

boost する空間の方向を変えるには空間回転を組み合わせればよい。

実際に、3 次元 vector v を任意の方向を向いている vector として boost したければ、z 軸の正の方向を v の正の方向に向ける回転を表す 3×3 行列を R とすれば、まず $R^{-1} = R^T$ により v が z 軸上にくるように回転させ、そこで z 軸にそって $|v|$ の大きさで boost し、そのあと R によって z 軸を v 方向にくるように R で回転させれば良い。

$$\Lambda = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & & & \\ 0 & & R & \\ 0 & & & \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{1-v^2}} & 0 & 0 & \frac{|v|}{\sqrt{1-v^2}} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \frac{|v|}{\sqrt{1-v^2}} & 0 & 0 & \frac{1}{\sqrt{1-v^2}} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & & & \\ 0 & & R^T & \\ 0 & & & \end{bmatrix} \tag{23}$$

この結果は次のようにかける。

$$\Lambda = \begin{bmatrix} \lambda & \lambda v^T \\ \lambda v & 1 + (\lambda - 1)S \end{bmatrix} \tag{24}$$

ただし、$\lambda = 1/\sqrt{1-v^2}$ とし、S は次の、v の方向だけに依存する 3×3 行列とした。

$$S = \frac{1}{v^2} \begin{bmatrix} v_x^2 & v_y v_x & v_z v_x \\ v_x v_y & v_y^2 & v_z v_y \\ v_x v_z & v_y v_z & v_z^2 \end{bmatrix}, \quad S^2 = S, \quad Sv = v \tag{25}$$

これは、回転行列を実際に求めて計算してもよいが、Λ が式 (10) をみたし、e_1 を変換した vector の空間成分が v に等しいこと、v 以外の vector に依存しないことを確認すれば良い。上の式で、Λ の逆変換 Λ^{-1} は v を $-v$ で置き換えれば良いことも確かめられる。

以上で一般の向きの boost に対する Lorentz 変換の行列表示が分かった。上の表式では, $v^2 < 1$ を仮定しているが、v が光速に近い場合には $1 - v^2$ は精度が悪くなる。$e_0 = (1, 0, 0, 0)$ が boost により (E, p_x, p_y, p_z) $(E^2 = p^2 + 1, p$ は 3 次元 vector) となることがわかっていれば $\lambda = E, v = p/E$ により

$$\Lambda = \begin{bmatrix} E & p^T \\ p & 1 + (E-1)S \end{bmatrix}, \qquad S = \frac{1}{p^2} \begin{bmatrix} p_x^2 & p_y p_x & p_z p_x \\ p_x p_y & p_y^2 & p_z p_y \\ p_x p_z & p_y p_z & p_z^2 \end{bmatrix} \tag{26}$$

から計算した方が良い。

◦ boost と回転の合成

以下簡単のために反転は考えない。Λ_B と Λ_R とをそれぞれ boost と回転を表す 4×4 行列とする。これらの行列は Lorentz 変換であるが、これらの積もまた Lorentz 変換である。$e_0=(1,0,0,0)$ をこれらに作用させて $e_0'=(E,p_x,p_y,p_z)$ $(E^2=1+p^2)$ に変換されたとする。

$$e_0'=\Lambda_B\Lambda_R e_0 \tag{27}$$

e_0 には空間成分がないので回転で不変であり、 $\Lambda_R e_0=e_0$ となる。これから $e_0'=\Lambda_B e_0$ が得られ、$e0'$ の 3 次元 vector p から式 (26) により Λ_B が決まる。

逆に、任意に Lorentz 変換 Λ が与えられたとする。e_0 を変換して $e_0'=\Lambda e_0=(E,p)$ が求まったとする。(もし $E<0$ ならば時間反転をかけて $E>0$ としておく。) この p から式 (26) により boost の行列 Λ_B を作ることができる。

$$\Lambda_B^{-1}\Lambda e_0=\Lambda_B^{-1}e_0'=e_0 \tag{28}$$

となり $\Lambda_B^{-1}\Lambda e_0$ は e_0 を不変に保つので回転であることが分かる。$\Lambda_R=\Lambda_B^{-1}\Lambda$ とする。(もし $\det\Lambda_R<0$ ならば空間反転をかける。) 任意の Lorentz 変換 Λ は

$$\Lambda=\Lambda_B\Lambda_R \tag{29}$$

と回転と boost の積で表せることがわかった。($\det\Lambda_R<0$ なら空間反転も必要。) 回転と boost が持つ実数のパラメターはどちらも 3 個なので Lorentz 変換の持つ自由度は 6 である。

結局 Lorentz 変換は、時間反転と、Lorentz boost、空間反転、空間回転の合成でかけ、それぞれの行列表現がわかった。

B　台形公式

以下、伊理, 藤野「数値計算の常識」(共立出版,1985) pp.40 による。

$[0,1]$ で定義された実関数 $f(x)$ を積分したい ($f(0), f(1)$ は有限とする)。

$$I = \int_0^1 f(x)\,dx \tag{30}$$

積分区間を等分に分けて台形公式で求めた値で I を近似する。

$$I_1 = \frac{f(0)+f(1)}{2} \tag{31}$$

近似をあげるために積分区間を 2 つに割ってそれぞれに台形公式を使う。それぞれの積分区間の幅は $1/2$ なので

$$I_2 = \frac{1}{2}\left[\frac{f(0)+f(1/2)}{2} + \frac{f(1/2)+f(1)}{2}\right] = \frac{f(0)}{4} + \frac{f(1/2)}{2} + \frac{f(1)}{4} \tag{32}$$

同様にして区間を 2^n 個に分割していく (つまり区間を半分に切っていく) ことにする。

$$I_n = \sum_{j=0}^{2^n} w_j^{(n)} f(x_j^{(n)}) \tag{33}$$

ただし、

$$x_j^{(n)} = \frac{j}{2^n}, \quad w_j^{(n)} = \begin{cases} 2^{-n-1} & j = 0, m \\ 2^{-n} & 1 \le j \le m-1 \end{cases} \tag{34}$$

つまり $x_j^{(n)}$ は $[0,1]$ を 2^n 個に分けたうちの j 番目の座標。$(w_0^{(n)}, w_1^{(n)}, \cdots, w_{2^n-1}^{(n)}, w_{2^n}^{(n)})$ は $(1, 2, \cdots, 2, 1)/2^{n+1}$ で、当然 $\sum_{j=0}^{2^n} w_j^{(n)} = 1$ を満たしている。

これから容易に次がわかる。

$$I_0 = \frac{f(0)+f(1)}{2}, \tag{35}$$

$$I_{n+1} = \frac{I_n}{2} + \frac{1}{2^n}\sum_{k=1}^{2^{n+1}} f\left(\frac{2k+1}{2^{n+1}}\right). \tag{36}$$

この漸化式により数列 I_n を求めていく。この際、異なる x の点について $f(x)$ が 2 度以上計算されることはない。n を増やしていけば、順次精度が上がると期待される。

さらに、 $f(x)$ が素直な (すなわち “解析的な”、Taylor 展開可能な) 関数の時には I_n は n を 1 増やす毎に誤差は $1/4$ にへり、

$$I_n^{(1)} = \frac{3I_n - I_{n-1}}{3} \tag{37}$$

を作ると、n を増やす毎に誤差は $1/16$ になる。さらに一般化もできる (Richardson 補外)。しかし、関数が解析的でなければこのような誤差の振る舞いからはずれてくる。また n をどんどん増やしていくと、解析的な関数でも数値計算の精度の問題が現れるので誤差は頭打ちになり、さらに n を増やすと誤差が逆に増えることになる。