付録 L

独断と偏見に満ちた FORTRAN 入門

L.1 はじめに

一般にプログラムは下の例題のように人が読める形で書かれる。これを計算機が実行するためには，機械が読める形に書き直さなければならない。この作業を「コンパイル」と呼ぶが，これはもっぱら機械（毎に専門家が作ってくれたソフトウェア「コンパイラ」）が自動的にやってくれるので，ここでは「人が読める」プログラムの書き方を示す。このようなプログラムを書く規則（言語）には種々あるが，その中で科学・技術計算に適用し易い FORTRAN (FORmula TRANslation) 言語について説明する。よく使われる BASIC (Beginner's All-purpose Symbolic Instruction Code) 言語に似ているところがあるが，重要な部分で違っているので，注意が必要である。ここでは第 1 著者自身の失敗経験と独断および偏見に基づき，間違いの無いプログラミングができるような鉄則（のようなもの）を示す。もちろん文法的には，必ずしもここに書かれたことを遵守する必要は無いが，計算機が一体何をしているのかを正確に理解できるまでは，ここに書かれた制限を守った方が間違いが少ないと思う。参考書[1]や土木工学関連の例題を含んだ書籍[2]も多数ある。なおここでは FORTRAN77 に限定し， FORTRAN90 については述べない。

さて，そのコンパイラであるが， Cygwin のインストール時にちょっとだけ気を付ければ freeware として使うことができるようになる。デフォルトままだと使えない場合があるので，インストールパッケージを設定する段階で，ソフトウェア開発カテゴリ 'Devel' の一覧を表示させて 'gcc' と 'gcc-g77' の行を探し，そこで図 L.1 に示したように，楕円で囲った辺りに表示される 'Skip' をクリックすることによってそれをインストールするという選択をすると，その楕円で囲った部分にバージョンが表示される。インストール後に環境変数 PATH を適切に設定すると， C と FORTRAN が使えるようになる。

L.2 文法の基本

L.2.1 プログラムの書き方と変数及び算術代入文

まず，実際にプログラムを作って実行してみよう。次の**例題 1** は四角形の縦と横の長さを入力して，その面積を求める簡単なものである。プログラムは，まずエディタでこのような「ソースファイル」を作成しなければならない。どんなエディタが使い易いのか等については回りの人に尋ねること。ここでは簡単のために，名称が edit というエディタがあるものとする。

[1] 例えば，文法と例題がコンパクトになったものとして「森口繁一・伊理正夫編『算法通論（第 2 版）』東京大学出版会， 1985.」を推薦する。文法書を買うよりも，例題が適切なことと，数値解析のことまで書いてあるのでいい。

[2] 例えば，「土木学会『土木情報処理の基礎』土木学会， 1988.」等。この付録の参考文献だけは脚注に記した。

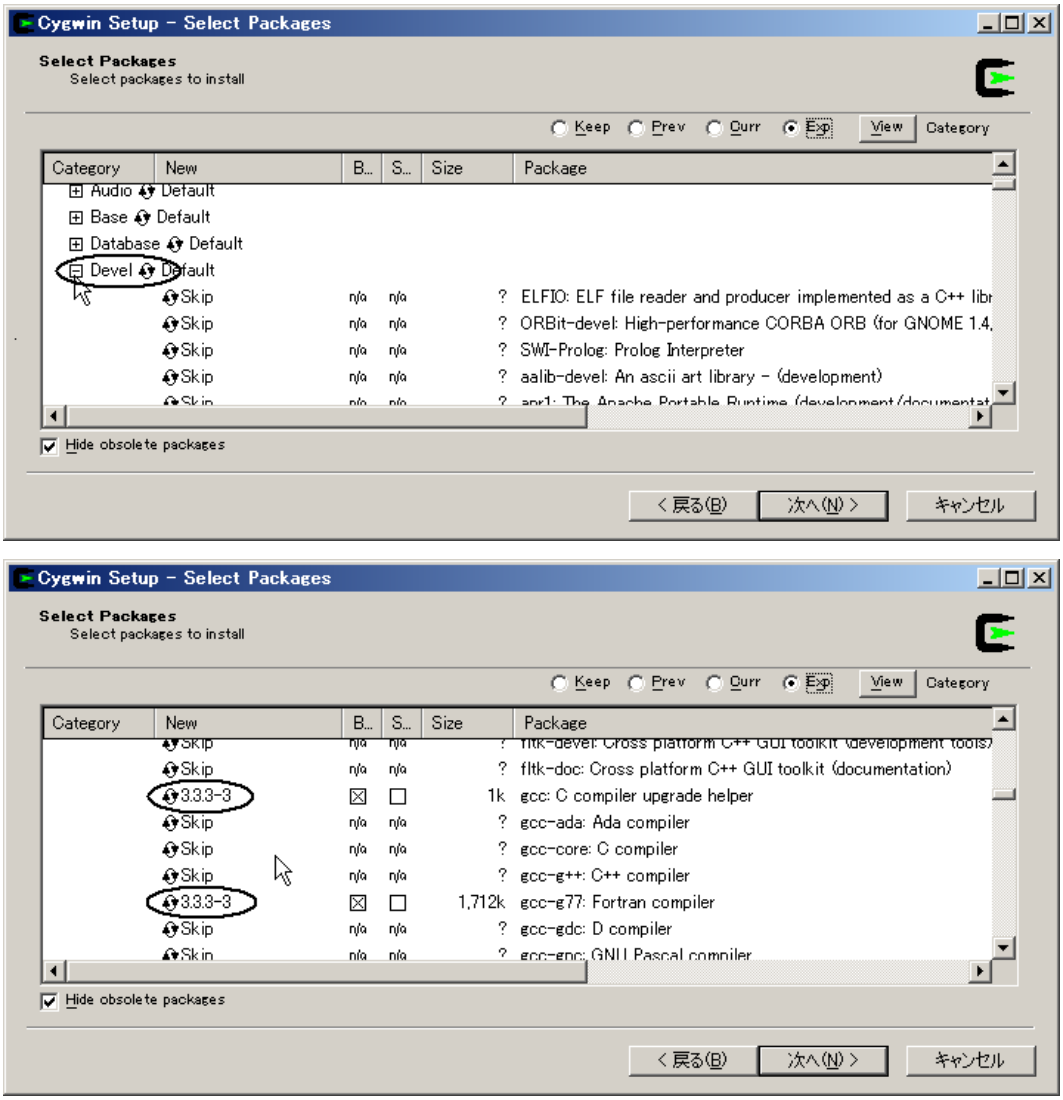

図 L.1 Cygwin で FORTRAN を使えるようにする注意

ファイルの名前は内容を反映したものの方があとでわかり易いので，**例題 1** の 1 行目の program 文の名前に使う『area1』を選んでおこう。まず，このファイルが FORTRAN プログラムのソースであることを明示するために，拡張子を付けて『area1.f』というファイル名[3]にする。以下， Linux あるいは Cygwin におけるコマンドラインでの入力を示すことによって，何をしているかを明らかにしておきたい。 Windows では，「スタート」→「すべてのプログラム」→「アクセサリ」→「コマンドプロンプト」とし，まず，エディタを

```
edit area1.f ↵    (↵ は Enter キーの押し下げ)
```

で立ち上げ，下の例の program 行から end 行までの全部を入力する（コメント行[4]は省いていい）。これを機械語に変換するには，例えば

```
g77 -o area1 area1.f ↵
```

とすればいい。 OS によっては 'f77' かもしれないが，いずれもコンパイラのプログラムである。オプション

[3] 今の世の中，英語を使おう。ローマ字 'menseki.f' とか間違ったスペルの 'eria.f' は見苦しい。漢字を使うのはもってのほか。
[4] 行頭に 'C' あるいは '*' のある行はコメントとみなされプログラムとは関係が無い。この記号は FORTRAN90 では異なる。

等については，他のシステムのコンパイラとほぼ同じであるが， Cygwin のマニュアルも，通常のマニュアルコマンド

```
man g77 ↵   あるいは   g77 --help | less ↵
```

で確認することができる。

【例題 1】

```
      program area1
c                             ex1: 四角形の面積を求める
c                             縦と横の長さを入力し，面積を出力
c ← 1 桁目が‘‘c’’ の行はコメント行（プログラムとは無関係な文）
c    | ← 1 ～ 5 桁が行番号の桁
c     | ←  7 桁目から 72 桁目までがプログラム本体（73 桁以降は無視）      → |
      read(5,100) a,b
      write(6,200) a,b
      a=a*b
      write(6,300) a
      stop
c
  100 format(2f10.0)
  200 format(' ','a =',1pe15.7,'  b =',e15.7)
  300 format(' ','A =',1pe15.7)
c                             5, 6 は標準入出力装置番号
c                             キーボードと画面が 5 と 6，あるいは
c                             area1  <inarea.dat   >outarea.txt
c                             ファイル‘inarea.dat’ の例
c                             ----+----1----+----2
c                                    1.2       2.7
c                             ----+----1----+----2
      end
```

ソースに間違いが無ければ，最終的に『area1』というファイル[5]ができる。これが実行形式のプログラムであり，機械には読めるが人間は読めないプログラムである。実行するには

```
area1 ↵        あるいは        ./area1 ↵
```

と[6]すればよく，縦横の長さを例えば

```
␣␣␣␣␣␣␣3.4␣␣␣␣␣␣␣␣2. ↵
```

と入力すれば (␣ は空白を示す)，答えが画面上に「6.8」と出るはずである。面積を計算している行の “a=a+b” は算術代入文と呼ばれ，右辺を計算した後左辺の “a” に値が代入される。理解できない読者は第 K.3 節の Java プログラミングの例を勉強して欲しい。

> **鉄則 1:** プログラムの内容がわかるような名前（6 文字以内）を “program 文（本質的には不要な文）” には必ず付け，ファイルの名前もわかり易いものにする。

データ入力は read 文で行なっている。この例の read(5,100) は， 5 番の入力装置（標準ではキーボード）から「100」で示した「形式」で入力する，という意味である。そこでこの 100 という数字のついた format 文を捜し，『100 format』を見ると，『2f10.0』となっている。つまり「1 行当り 10 桁で示した実数を二つ入力する」ということなので，入力する時の数字の位置をきちんとしないとエラー[7]が出る。したがって，上の入力例のように適当な数の空白が必要になるわけである。ただし， read(5,*) とすると，二つの値を空白一つ以上離して入力すればよくなる場合もあり，簡単にはなる。

---

[5] あるいは a.out というファイルかもしれない。 Windows (Cygwin) の場合には a.exe や area1.exe かもしれない。
[6] OS の環境変数 ‘PATH’ にカレントディレクトリ ‘.’ が含まれていないときは後者。
[7] OS によっては，スペースやコンマで二つの数値が離れていれば，エラーにならない場合もある。

一方出力は『1pe15.7』となっている。これは「実数を 15 文字表示とし，小数点以下 7 桁，小数点上 1 桁で指数表示する」ことである。つまり，例えば 123.45678 は， 1.2345678E+02 と表示される。他に， f 形や g 形があるが，慣れれば e 形の方が間違いの無い表示法である。またあとで出てくる整数は，入出力ともに，I 形（i5 や i10 等）である。注意：“100 format” の二つ目の “b” の出力の “e15.7” の頭に “1p” を付けるととんでもないことになる。小数点より上の桁数を指定するのは一箇所のみである。

あるいは予め別のファイルを作成し，そこに入力するデータを書き込んでおいて実行させることもできる。それには，例えば “inarea.dat” というファイルに数値が 1 行で上述の 3.4 と 2. の例の通りに適切に書き込まれていたとすると

```
area1 < inarea.dat ↵
```

で， 5 番の入力はこのファイルからということになり，結果はやはり画面に出る。出力の 6 番を画面ではなくファイルにする場合には

```
area1 < inarea.dat > outarea.txt ↵
```

として実行すると，結果が，つまり 6.8 という文字がファイル “outarea.txt” に出力される。この “<” や “>” は，情報の入出力の向きを指定するのに用いられ，「リダイレクション」と呼ばれる操作である。

鉄則 2: 精度が必要無い実数入力は， f10.0 形にする。整数は i5 か i10。

鉄則 3: 実数の出力は，普通は 1pe15.7 形にする。あるいは 1pe$n$.m ($n = m+8$) とする。感覚的には 1p を付けた方が読み易いだろう。ただし p の有効範囲はその format 文の文末の括弧 ‘)’ までであり， (1pe15.7, ’ b=’, f12.3) のように， 1p が有効な部分で f 変換や別の e 変換を用いると，一桁違うとんでもない答が表示されるので注意。

鉄則 4: 入力（read 文）は 5 番のファイル識別番号から format 付きで行なう。

鉄則 5: 出力（write 文）は 6 番のファイル識別番号へ format 付きで行なう。

入力を format 付きで行なうと，キーボードからの入力は煩雑になり，間違い易い。だから format 無しの，read(5,*) や read(*,*) が好ましい場合がある。しかし，あとで述べるように，通常のプログラムではファイル入出力を標準としており，その場合には format 付きのほうが便利であると同時に，内容の理解が容易となる。したがって，ここでも format 付きを推奨している。

もう一つ，面積と断面 2 次モーメントを計算する**例題 2** を考えてみよう。

【例題 2】

```
      program area11
c                             ex1_1: 四角形の面積と
c                                                断面 2 次モーメントを求める
c                             縦と横の長さを入力
      read(5,100) a,b
      write(6,200) a,b
c                             A= a*b
c                             I= a*a*a*b/12.
      b=a*b
      a=a*a*b/12.
      write(6,300) b,a
      stop
c
  100 format(2f10.0)
  200 format(’ ’,’a =’,1pe15.7,’  b =’,e15.7)
  300 format(’ ’,’A =’,1pe15.7,’  I =’,e15.7)
      end
```

例題 2 で，通常 “I” で表示する断面 2 次モーメントを “a” として計算している。しかし，変数の記号はあとで読んだ時にすぐ内容が解るようにした方がいいので，慣例の記号を用いた方がよさそうだ。そこで，次の**例題 3** のように書き直してみよう。

【例題 3】

```
      program area12
c                             ex1_2: 四角形の面積と
c                                                 断面 2 次モーメントを求める
c                             縦と横の長さを入力
      read(5,100) a,b
      write(6,200) a,b
c                             A= a*b
c                             I= a*a*a*b/12.
      area=a*b
      I=b*a**3/12.
c                          ↓実行するとエラー
      write(6,300) area,I
      stop
c
  100 format(2f10.0)
  200 format(' ','a =',1pe15.7,'  b =',e15.7)
  300 format(' ','A =',1pe15.7,'  I =',e15.7)
c                                        ↑頭文字が‘i’から‘n’までの変数は
c                                          通常は整数として解釈される
c                 一つの解決策: program 文の次に‘real i’の 1 行を入れる
      end
```

ところが，これは実行するとマークのある部分でエラーが起こる。これは，変数名が一般に『暗黙の型宣言』に従っているからである。

変数の頭文字が i から n までの変数以外は暗黙のうちに実数変数として解釈されるが，i から n までの頭文字を持つ変数は整数として解釈されてしまう。したがって，上の**例題 3** の中の “I=b*a**3/12.” の計算では，右辺が例えば「12.345」という値を持っていても整数化されてしまい，「12」という値を変数 “I” は持ってしまう。しかも整数を実数の形式 “e 形” で出力しようとする部分でエラーが発生してしまう。よって上の例のような場合には，変数を “I” の代わりに “aI” 等として使うといいかもしれない。

実際に計算をしている行のような文を算術代入文と呼ぶが，これや format 文は 1 行（72 カラムまで）に書ききれないこともある。この場合は**例題 4** のように継続行を使用する。

【例題 4】

```
      program area13
c                             ex1_3: 長い計算
      read(5,100) a,b
      write(6,200) a,b
c
      x = a * b**3 + b * sin(a)
      y = x * ( a * cos(b) - exp(a) / 2.4 ) / ( 120.9 * tan( a/b ) -
     *    23.03 ) - x
c     ↑
c     6 桁目に‘‘0’’以外の文字を入れると，継続行とみなされる。
      write(6,300) x,y
      stop
c
  100 format(2f10.0)
  200 format(' ','a =',1pe15.7,'  b =',e15.7)
  300 format(' ','x =',1pe15.7,'  y =',e15.7)
      end
```

鉄則 6: i から n までの頭文字を持つ実数を使用しない。

鉄則 7: タイピングに慣れてきたら，意味のある変数名（6 文字以内[8]）を使用する。

[8] これは今は緩和されているかもしれない。

多くのケーススタディーをしようとしているときには，データをプログラム中の代入文として入れてしまうと，データを変更する度にコンパイルし直さなければなければならなくなる。これは不便であり無駄である。したがって，問題や設定によって値の変わる変数の入力は代入文ではなく， read 文で行なうべきである。

鉄則 8: 定数ではないデータは必ず， read 文で入力する。

鉄則 9: read したデータはすぐに write して確認する。

L.2.2 ファイル入出力

上ではキーボードから入力して画面に出力していたが，これだと結果が出るまで端末の前にいなければならない。そこで，ファイルから入力し，結果をファイルに書き込む方法を説明する。こうすれば，キーボードでのキーの押し間違いも避けられるし，出力結果をあとでゆっくり見たり，図化ソフトウェアにデータを渡すことも簡単にできる。また，実行している間は勉強や他の作業ができる。そのためには，**例題 5** のように単に “open 文” を加えるだけでよく，他は書き直す必要が無い。ただし，終了前にファイルを “close” しておく必要がある。

【例題 5】

```
      program area2
c                             ex2: 四角形の面積を求める
c                             縦と横の長さを入力し，面積を出力
      open(5,file='inarea.dat')
      open(6,file='outarea.txt')
c
      read(5,100) a,b
      write(6,200) a,b
      a=a*b
      write(6,300) a
c
      close (5)
      close (6)
      stop
c
  100 format(2f10.0)
  200 format(' ','a =',1pe15.7,'  b =',e15.7)
  300 format(' ','A =',1pe15.7)
c                             5, 6 を明確にファイルに割り当てる
c                             入力: inarea.dat   出力: outarea.txt
c                             ファイル‘inarea.dat’の例
c                             ----+----1----+----2
c                                    1.2       2.7
c                             ----+----1----+----2
      end
```

さて，そういうことになると，いくつもの場合の計算をいっぺんにやらせた方が効率的である。つまり， 1 ケース毎に入力ファイルのデータを変えて実行し直すのではなく，計算したい全部のデータを一つのファイルに書き込んでおいて，一度に実行した方が便利である。これには，入力と出力を大きなループで結んでおけばいいことがすぐわかるが，どこが最後の入力なのかを教えてやらないと，エラーで終了するという醜いプログラムになってしまう。最も簡便に行なう方法は，データの特徴で判断する方法（**例題 6**）である。

【例題 6】

```
      program area3
c                             ex3: 四角形の面積を求める
c                             縦と横の長さを入力し，面積を出力
      open(5,file='inarea.dat')
      open(6,file='outarea.txt')
c ----------------- ここから
   20 continue
      read(5,100) a,b
c                             連続入力を，あり得ない入力値で制御する
      if( a.le.0. ) go to 10
```

```
      write(6,200) a,b
      a=a*b
      write(6,300) a
      go to 20
c ----------------- ここまでがループ
   10 continue
      close (5)
      close (6)
      stop
c
  100 format(2f10.0)
  200 format(' ','a =',1pe15.7,'  b =',e15.7)
  300 format(' ',15x,'A =',1pe15.7)
c                             5, 6 を明確にファイルに割り当てる
c                             入力: inarea.dat   出力: outarea.txt
c                             ファイル'inarea.dat' の例
c                             ----+----1----+----2
c                                    1.2       2.7
c                                    2.3       8.1
c                                    -1.       8.1
c                             ----+----1----+----2
      end
```

また，円の面積の場合も同様 (**例題 7**) に・・・

【例題 7】

```
      program area4
c                             ex4: 円の面積を求める
c                             半径を入力し，面積を出力
      data pi / 3.14159265 /
c
      open(5,file='inarea.dat')
      open(6,file='outarea.txt')
c ----------------- ここから
   20 continue
      read(5,100) r
c                             連続入力を，あり得ない半径の入力で制御する
      if( r.le.0. ) go to 10
      write(6,200) r
      a=pi*r*r
      write(6,300) a
      go to 20
c ----------------- ここまで
   10 continue
      close (5)
      close (6)
      stop
c
  100 format(f10.0)
  200 format(' ','radius =',1pe15.7)
  300 format(' ',15x,'A =',1pe15.7)
c                             ファイル'inarea.dat' の例
c                             ----+----1----+----2
c                                    1.2
c                                    2.3
c                                    -1.
c                             ----+----1----+----2
      end
```

あり得ないデータで入力を制御できることになれば，**例題 6**, **例題 7** の二つのプログラムも合体させることができそう（**例題 8**）である。つまり，横の長さが入力されない場合は円の計算をすればいい。同じような内容の計算をするプログラムを種々の対象に対して使えるようにすると非常に便利なことが多い。このようなプログラムを『汎用性を持った』プログラムと呼ぶ。例えば，片持ち梁は解けるが単純支持梁は解けないプログラムよりは，両方共解けるものの方が便利であろう。

【例題 8】

```
      program area5
c                             ex5: 四角形，円の面積を求める
c                             縦横か半径を入力し，面積を出力
      data pi / 3.14159265 /
```

```
c
      open(5,file='inarea.dat')
      open(6,file='outarea.txt')
c
   20 continue
c                              連続入力を，あり得ない縦の入力で制御する
      read(5,100) a,b
      if( a.le.0. ) go to 10
c                              入力の数で円か四角かを判断
      if( b.gt.0. ) go to 30
c                              円
      write(6,200) a
      a=pi*a*a
      write(6,300) a
      go to 20
c                              四角
   30 continue
      write(6,400) a,b
      a=a*b
      write(6,300) a
      go to 20
c
   10 continue
      close (5)
      close (6)
      stop
c
  100 format(2f10.0)
  200 format(' ','radius =',1pe15.7)
  300 format(' ',15x,'A =',1pe15.7)
  400 format(' ','a =',1pe15.7,'  b =',e15.7)
c                              ファイル'inarea.dat' の例
c                              ----+----1----+----2
c                                     1.2        0.
c                                     2.3       4.5
c                                     3.6       .45
c                                      .2        0.
c                                     -1.       .45
c                              ----+----1----+----2
      end
```

さらに，プログラムを見易くするために，異なる場合の計算は別の場所でやらせるようにしたい。これには subroutine と呼ばれる副プログラムを使用すればいい（**例題 9**）。そうすると将来，例えば「台形の計算もしたい」といった場合に，非常に簡単にプログラムの加筆・修正ができる。

【例題 9】

```
      program area6
c                              ex6: 四角形，円の面積を求める
c                              縦横か半径を入力し，面積を出力
      common / const / pi
      pi=4.*atan(1.)
c
      open(5,file='inarea.dat')
      open(6,file='outarea.txt')
c
   20 continue
c                              連続入力を，あり得ない縦の入力で制御する
      read(5,100) a,b
      if( a.le.0. ) go to 10
c                              入力の数で円か四角かを判断
      if( b.gt.0. ) then
                               call rect (a,b)
      else
                               call circle (a)
      end if
      write(6,*) a
c     ^^^^^^^^^^^^  rect を call した後には a の値が変更されている !!!
c
      go to 20
c
   10 continue
      close (5)
```

```
      close (6)
      stop
c
  100 format(2f10.0)
      end
c ---------------------------------------------------------------------
      subroutine circle (r)
c                             円
      common / const / pi
      write(6,100) r
      a=pi*r*r
      write(6,200) a
      return
c
  100 format(' ','radius =',1pe15.7)
  200 format(' ',15x,'A(circle) =',1pe15.7)
      end
c ---------------------------------------------------------------------
      subroutine rect (a,b)
c                             四角
      write(6,100) a,b
      a=a*b
      write(6,200) a
      return
c
  100 format(' ','a =',1pe15.7,'  b =',e15.7)
  200 format(' ',15x,'A(rectangle) =',1pe15.7)
      end
```

subroutine は，それを呼ぶプログラムとの間で変数の受渡しをしなければならないが，それには『引数』と呼ばれるもの（括弧書きの中の並び）と， common 文によるものとがある。上の例でもこの二つを併用しているが，落ち着いてよく見ると， common で渡しているのは「π」，つまり値を変えない変数であり，引数では変化する変数を受渡している。特にこうしなければならないことは無いが，最初のうちはこのような区別をしておくと間違いが少ない。

各 subroutine およびメインプログラムはそれぞれ独立した「プログラム単位」であり，引数と common で受け渡される変数の値以外は，すべてプログラム単位内ローカルであって，個々に独立している。したがって，上の circle という subroutine 中の変数 a はメインの a とは全く関係が無い。しかしながら， rect という subroutine の中の変数 a は，メインの a と引数を通して授受されてしまうことには注意する。そのため，メインプログラムでこの rect を call したあとの write 文で出力される a の値は， subroutine に渡す直前の (a*b) の値に置き換わっていることに注意する。

鉄則 10: 入出力はファイル使用を標準とし，プログラムは多くの場合の計算を連続して行えるようなフローにする。入力を制御する良い方法を考える。

鉄則 11: 必要ならば可能な限り汎用性を持ったプログラムを作る。

鉄則 12: 種々の場合を計算する汎用プログラムでは，場合分けを subroutine で行なっておくと，あとで追加・削除が容易である。

鉄則 13: subroutine へは引数で変数を引き渡す。π 等の値の変わらない変数だけを common 文で引き渡すようにすると最初は間違いが少ない。

鉄則 14: common 文は，できるだけ明確な名前付き（上の例では 'const'）で用いる。また同じ common 文に実数と整数を並べない方がいいが，もし並べたい場合には，実数を先に並べる（倍精度実数を用いる場合の機械依存性の問題点）。

鉄則 15: go to 文や後述の do 文の制御には continue 文を使用した方があとで修正し易い。ただし最近は， do 文の制御には end do を使うようだ。

近年，構造化プログラミングという言葉と共に，見通しの良いプログラムのためには go to 文を使わない方が良いとも言われる。確かにプログラムを作って一ヶ月も放置しておくと，作った本人でも何が何だかわからないものになることはよく経験する。特にこういった go to 文が元凶の一つであることも理解はできるが，システムプログラムと違い，シミュレーションのプログラムをこれ無しでプログラミングするのが面倒な場合があるし，コメントを付けることで見通しをある程度は良くしておくこともできる。余裕が許す限りコメントは入れておくことを推奨する。ただ subroutine をこまめに使って， if then... else... end if が使える人は go to を少なくできる。

L.2.3 文字変数と倍精度実数

通常の実数計算は 4 バイト 32 ビットで計算しているため，機械の中は 7 桁くらいの表現になる。したがって，複雑な演算を行なうと，有効数字でせいぜい 3 桁くらいの精度しか得ることができない。これを 8 バイト 64 ビットでやらせれば，精度が倍になり，約 6 桁程度の有効数字精度を得ることができる。現在，単精度・倍精度計算で実行速度の差は無く， C 言語がそうであるように，科学・技術計算はすべて倍精度で行なうのが普通である。これを簡単に行なうためには，単に各プログラム単位で暗黙の型宣言文 “implicit 文” を挿入するだけでいい。

また，多くの場合を連続して計算する場合の制御に，文字変数を使用したタイトルを利用すると，出力結果の整理にも役に立ち，便利である。**例題 10** では，最大 76 文字のタイトルを使って，入力の制御を行なってみた。“character*4 title(19)” というのは， 4 バイト（4 英数字）ずつの文字変数を 19 個連続して縦に並べて（後述の「配列」）ある，ということで，合計 76 文字に相当する。その最初の 4 文字を “title(1)” という変数として扱っている。この最初の 4 文字で入力を制御しようとしている。プログラムの終了には “end␣” の 4 文字（最後にスペースが一つある）を用いる。

【例題 10】

```
      program area7
c                             ex7: 四角形，円の面積を求める
c                             縦横か半径を入力し，面積を出力
      implicit real*8(a-h,o-z)
c                             倍精度（8 バイト実数）
      character*4 title(19),aend
      common / const / pi
      data aend / 'end ' /
      pi=4.d0*atan(1.d0)
c
      open(5,file='inarea.dat')
      open(6,file='outarea.txt')
c
   20 continue
c                             タイトルで終了制御
      read(5,100) title
      if( title(1).eq.aend ) go to 10
      write(6,200) title
c
      read(5,300) a,b
c                             入力の数で円か四角かを判断
      if( b.gt.0. ) then
                              call rect (a,b)
      else
                              call circle (a)
      end if
c
      go to 20
c
   10 continue
      close (5)
      close (6)
      stop
c
```

```
  100 format(19a4)
  200 format(//' ',19a4)
  300 format(2f10.0)
c                             ファイル'inarea.dat' の例
c                             ----+----1----+----2
c                             Problem No.1
c                                    1.2        0.
c                             Next Problem
c                                    2.3       4.5
c                             Last Problem
c                                     .2        0.
c                             end_                 <-- 4 文字目はスペース！
c                             ----+----1----+----2
      end
c ---------------------------------------------------------------------
      subroutine circle (r)
c                             円
      implicit real*8(a-h,o-z)
      common / const / pi
      write(6,100) r
      a=pi*r*r
      write(6,200) a
      return
c
  100 format(' ','radius =',1pd15.7)
  200 format(' ',15x,'A(circle) =',1pd15.7)
      end
c ---------------------------------------------------------------------
      subroutine rect (a,b)
      implicit real*8(a-h,o-z)
c                             四角
      write(6,100) a,b
      a=a*b
      write(6,200) a
      return
c
  100 format(' ','a =',1pd15.7,'  b =',d15.7)
  200 format(' ',15x,'A(rectangle) =',1pd15.7)
      end
```

鉄則 16: 入力制御はタイトル（文字変数）を利用する。

鉄則 17: 実数計算はすべて倍精度で行なう。“`implicit real*8(a-h,o-z)`” を，必要なメインプログラムと `subroutine` には忘れずに付ける。

鉄則 18: 倍精度変数の出力には，“`1pd15.7`” あるいは“`1pdn.m (n=m+8)`” を使用する。

L.2.4 1 次元配列

上の `title(19)` という変数は実は『配列』と呼ばれるもので，19 個の変数を同じ変数名で定義する代わりに，その何番目の変数かで区別している。これを使うと，ベクトルや行列の計算が非常に簡単にできることがわかる。まず，ベクトルの内積の計算を例に，1 次元の配列の例を考えてみる。

【例題 11】

```
      program inprod
c                             ex8: 2つのベクトルの内積を求める
      character*4 title(19),aend
      dimension a(3),b(3)
      data aend / 'end ' /
c
      open(5,file='invec.dat')
      open(6,file='outvec.txt')
c
   20 continue
c                             タイトルで終了制御
      read(5,100) title
      if( title(1).eq.aend ) go to 30
      write(6,200) title
```

```
c
      read(5,300) (a(i), i=1,3)
      read(5,300) (b(i), i=1,3)
      write(6,400) (a(i), i=1,3)
      write(6,400) (b(i), i=1,3)
c                             内積の計算
      prod=0.
      do 10 i=1,3
      prod=prod + a(i)*b(i)
   10 continue
      write(6,500) prod
c
      go to 20
c
   30 continue
      close (5)
      close (6)
      stop
c
  100 format(19a4)
  200 format(//' ',19a4)
  300 format(3f10.0)
  400 format(' ','1:',1pe15.7,'  2:',e15.7,'  3:',e15.7)
  500 format(' ',15x,'Inner product =',1pe15.7)
c                             ファイル'invec.dat' の例
c                             ----+----1----+----2----+----3
c                             Problem No.1 (3-D)
c                                    1.2       2.2       1.1
c                                    -.3       3.9      -4.5
c                             Next Problem (2-D)
c                                    2.3      -4.5        0.
c                                    8.1       .34        0.
c                             Last Problem (3-D)
c                                     .2       2.1       -.2
c                                   -1.1      -3.2       9.1
c                             end_                 <-- 4 文字目はスペース！
c                             ----+----1----+----2----+----3
      end
```

L.2.5 common 文と parameter 文・ data 文

subroutine 間のデータのやりとりは common 文や引数で行なわれる。この時，やりとりする変数 (?) をどのように定義するかについて少し検討してみる。まず， data 文の特性について追求してみる。次の**例題 12** を実行してみる。

【例題 12】

```
      program param1
c                             ex9: データのやりとりと宣言について
      data n / 5 /
      write(6,100) n
      n = 2
      write(6,100) n
      data n / 8 /
      write(6,100) n
      stop
c
  100 format(' ',i5)
c                             このプログラムの出力結果は
c                                 8
c                                 2
c                                 2
c                             です。なぜでしょう?
      end
```

この場合なぜ出力がそうなるのか? 実は data 文はどこにあってもいい代わりに『プログラムが実行される直前に一度だけしか』実行されない。つまり， data 文は，その変数の『初期値』を与えるに過ぎない。したがって上の例題では， n の初期値はうしろに現われる二つ目の data 文で 8 に初期化され，実行が始まったらすべ

ての data 文は無視されるから，“n=2” の行の上では n=8 のままであり，“n=2” の行より下では常に n=2 のままとなる。どこにあってもいいから，複雑なプログラムの subroutine なんかで定義してしまうと，将来の変更の場合の妨げにもなる。このような場合には block data 文と common とを用いる。これについては参考書参照。

鉄則 19: 変化させない変数か，初期値を適切に設定したい変数だけを data 文で定義する。

鉄則 20: data 文はメインプログラムの最初だけで定義する。

さて FORTRAN では，変数の他に『定数』も定義できる。これは， parameter 文で行なう。この性質について見てみる。

【例題 13】

```
      program param2
c                             ex10: データのやりとりと宣言について
      parameter (n=5)
      write(6,100) n
      n = 2
      write(6,100) n
      stop
c
  100 format(' ',i5)
c                             このプログラムは
c                             5 行目の‘n=2’ が文法違反です。
c                             なぜでしょう?
      end
```

parameter 文で定義される定数（本質的には『文字』に過ぎない!）というのは，数字の代わりにプログラムを書く時点で用いられるだけのものである。だから，プログラムの実行とは全く無関係にコンパイルの段階で処理されてしまう。したがって，上の例題のエラーが生じる行「n = 2」は，本質的には

```
5 = 2
```

という文を書いたことに相当するから，文法違反となるわけである。ということだから，次の**例題 14** もおかしいことになる。 n は変数ではなく『5』だからである。「変数」と「定数」が違うことを理解しておくこと。

【例題 14】

```
      program param3
c                             ex11: データのやりとりと宣言について
      parameter (n=5)
      common n
      write(6,100) n
      call suba
      stop
c
  100 format(' ',i5)
c                             このプログラムは
c                             4 行目の‘common n’ が文法違反です。
c                             なぜでしょう?
      end
c ----------------------------------------------------------------------
      subroutine suba
      common n
      return
      end
```

また

【例題 15】

```
      program param4
c                             ex12: データのやりとりと宣言について
      common n
      data n / 5 /
      write(6,100) n
```

```
      call suba
      stop
c
  100 format(' ',i5)
c                             このプログラムも
c                             3 行目の'common n' が文法違反です。
c                             なぜでしょう?
      end
c ----------------------------------------------------------------------
      subroutine suba
      common k
      write(6,100) k
      return
c
  100 format(' ',i5)
      end
```

これも，先に述べたように data 文は初期化する働きを持った非実行文であるため， common 領域で定義された変数を実行前に初期化しようとするのが困難だからであろう? このようなことをどうしてもしたい場合は，他の手段 (block data) の使用が考えられる。

鉄則 21: parameter 文は（「定数」の意味が理解できない間は）使用しない。百害あって一利無し。

鉄則 22: common 文に含める変数の初期化を data 文で行なわない。

さて， subroutine 間の変数の引渡しには引数と common が使用され，もちろんどちらも同時に用いて構わないが，次のような重複の仕方は間違いとなる。

【例題 16】

```
      program param5
c                             ex13: データのやりとりと宣言について
      data n / 5 /
      write(6,100) n
      call suba(n)
      write(6,100) n
      stop
c
  100 format(' ',i5)
c                             このプログラムも subroutine suba の
c                             2 行目の'common k' が文法違反です。
c                             なぜでしょう?
      end
c ----------------------------------------------------------------------
      subroutine suba(k)
      common k
      k=2*k
      call subb
      return
      end
c ----------------------------------------------------------------------
      subroutine subb
      common j
      j=j/2
      return
      end
```

つまり，引数で受渡しているから common は必要無いわけだが，これは， common 領域の概念と引数の概念とが全く異なることによるエラーである。 common 文というのは，その common （共通）メモリ領域の最初の番地から順番に値が記憶されているものを，参照するだけのものである。だから，上の例の subb のように，異なるプログラム単位毎に違う変数名で参照しても構わない（あとでわからなくなるのでできるだけそろえた方がいいが）。これに対して引数は， call する側の変数の記憶されている番地を， subroutine 側に教えるだけの機能しか持っていない。したがって，これも違う変数名で参照しても構わない。

ただし，上の**例題 16** で

【例題 17】

```
      program param5
c                             ex13_1: データのやりとりと宣言について
      common n
      n=5
      write(6,100) n
      call suba(n)
      write(6,100) n
      stop
c
  100 format(' ',i5)
      end
c ----------------------------------------------------------------------
      subroutine suba(k)
      k=2*k
      call subb
      return
      end
c ----------------------------------------------------------------------
      subroutine subb
      common j
      j=j/2
      write(6,100) j
      return
  100 format(' ',i5)
      end
```

とすることは可能である。つまり call する側の引数が common にあってもいい（**例題 17** のメインプログラム）が，call された側での重複（**例題 16** の suba）は不可となる。またこのプログラムを実行すると，すべての write 文は「5」を出力する。ただし，このように異なった方法で受渡しを行なうのは「不節操」であり，将来の修正が困難になるので避けた方がいい。

鉄則 23: 鉄則 13 を再度強調したいが，変数のうちのどれを common で引き渡すのかの自分なりの思想を持つ。

鉄則 24: 内容を間違わないためにも common 中の変数名はすべての箇所で同じものに統一しておく。つまりエディタの機能を用いてコピー・ペーストするのがベスト。

L.2.6 2 次元配列

次に，2 次元の配列を用いて行列計算をしてみる。簡単のために，2×2 の逆行列を求める。

【例題 18】

```
      program inv22
c                             ex14: 2x2 の逆行列
      character*4 title(19),aend
      dimension a(2,2),b(2,2),c(2,2)
      data aend / 'end ' /
c
      open(5,file='inmat.dat')
      open(6,file='outmat.txt')
c
   20 continue
c                             タイトルで終了制御
      read(5,100) title
      if( title(1).eq.aend ) go to 30
      write(6,200) title
c
      do 10 i=1,2
      read(5,300) (a(i,j), j=1,2)
      write(6,400) (a(i,j), j=1,2)
   10 continue
c
      det=a(1,1)*a(2,2)-a(1,2)*a(2,1)
```

```
      write(6,500) det
c
      b(1,1)=a(2,2)/det
      b(1,2)=-a(1,2)/det
      b(2,1)=-a(2,1)/det
      b(2,2)=a(1,1)/det
c
      do 40 i=1,2
      write(6,400) (b(i,j), j=1,2)
   40 continue
c                             チェック
      do 50 i=1,2
      do 50 j=1,2
      c(i,j)=0.
      do 50 k=1,2
      c(i,j)=c(i,j)+a(i,k)*b(k,j)
   50 continue
c                              きちんと単位行列になるか?
c                              どのくらいおかしいか?   なぜか?
      write(6,600)
      do 60 i=1,2
      write(6,400) (c(i,j), j=1,2)
   60 continue
c
      go to 20
c
   30 continue
      close (5)
      close (6)
      stop
c
  100 format(19a4)
  200 format(//' ',19a4/' ','Given matrix:')
  300 format(2f10.0)
  400 format(' ',' |',1pe15.7,'  |',e15.7,' |')
  500 format(' ',15x,'Determinant =',1pe15.7/' ','Inverse matrix:')
  600 format(' ','Check.....')
c                             ファイル'inmat.dat' の例
c                             ----+----1----+----2
c                             Problem No.1
c                                    1.2       2.2
c                                    -.3       3.9
c                             Next Problem
c                                    2.3      -4.5
c                                    8.1       .34
c                             Last Problem
c                                     .2       2.1
c                                   -1.1      -3.2
c                             end_                  <-- 4 文字目はスペース!
c                             ----+----1----+----2
      end
```

チェックのところでは，元の行列と逆行列を掛けたときに単位行列になるかどうかを検討しているが，実際実行してみると? 倍精度でも実行して比較してみるといい。計算機は前にも述べたように，絶対に (!) 厳密解は出してくれない。限られた精度の中での計算であることを忘れないように。

鉄則 25: 計算機は馬鹿である。言われた通りにしか動かないし，その結果を鵜呑みにしてはならない。

鉄則 26: プログラマーはもっと間が抜けている。計算結果が変な時やエラーが出る時の原因は 120% プログラマーにある。努々計算機やコンパイラを疑ってはならない。

L.2.7 2 次元配列の受渡しと上手なメモリの使い方（整合配列）

次に，行列の積を計算する。次の**例題 19** では，せいぜい 8×8 の行列までしか計算しないものと規定している。行なう計算は， $\boldsymbol{d} = \boldsymbol{A}^{\mathrm{T}}\,\boldsymbol{B}\,\boldsymbol{A}\,\boldsymbol{c}$ である。

【例題 19】

```
      program matprd
c                             ex15: 行列の積 {d} = [A(T)] [B] [A] {c}
      implicit real*8 (a-h,o-z)
      character*4 title(19),aend
      dimension a(8,8),b(8,8),c(8),d(8),e(8,8)
      data aend / 'end ' /
c
      open(5,file='inmatp.dat')
      open(6,file='outmatp.txt')
c
   20 continue
c                             タイトルで終了制御
      read(5,100) title
      if( title(1).eq.aend ) go to 30
      write(6,200) title
c
      read(5,600) mat
      if( mat.le.8.and.mat.gt.0 ) go to 40
      write(6,700)
      go to 30
   40 continue
      write(6,500) mat
      do 10 i=1,mat
      read(5,300) (a(i,j), j=1,mat)
      write(6,400) (a(i,j), j=1,mat)
   10 continue
      write(6,350)
      do 50 i=1,mat
      read(5,300) (b(i,j), j=1,mat)
      write(6,400) (b(i,j), j=1,mat)
   50 continue
      write(6,450)
      read(5,300) (c(i), i=1,mat)
      write(6,400) (c(i), i=1,mat)
c
      do 60 i=1,mat
      d(i)=0.d0
      do 60 j=1,mat
      d(i)=d(i)+a(i,j)*c(j)
      e(i,j)=0.d0
      do 60 k=1,mat
      e(i,j)=e(i,j)+a(k,i)*b(k,j)
   60 continue
c
      do 70 i=1,mat
      c(i)=0.d0
      do 70 j=1,mat
      c(i)=c(i)+e(i,j)*d(j)
   70 continue
c
      write(6,550)
      write(6,400) (c(i), i=1,mat)
c
      go to 20
c
   30 continue
      close (5)
      close (6)
      stop
c
  100 format(19a4)
  200 format(//' ',19a4)
  300 format(8f10.0)
  350 format(' ',' [B]')
  400 format(' ',8(' |',1pd15.7),' |')
  450 format(' ',' {c}')
  500 format(' ','Size of matrices =',i5/' ','Given matrices:'/
     *       ' ',' [A]')
  550 format(' ','Obtained vector:'/' ',' {d}')
  600 format(i5)
  700 format(' ','***** Data error : 0<mat<9 *****')
c                             ファイル'inmatp.dat' の例
```

```
c                             ----+----1----+----2----+----3
c                             Problem No.1
c                                 2
c                                    1.2       2.2
c                                    -.3       3.9
c                                    2.3      -4.5
c                                    8.1       .34
c                                    .55       -.2
c                             Next Problem
c                                 3
c                                     .2       2.1       -.4
c                                   -1.1      -3.2       2.7
c                                    1.2       2.2       -.9
c                                    -.3       3.9       5.7
c                                    7.1       8.8       -.8
c                                    3.2      -1.2        1.
c                                     1.       -.5        2.
c                             end_                  <-- 4 文字目はスペース！
c                             ----+----1----+----2----+----3
      end
```

特に必要なわけではないが, subroutine を使ってみると

【例題 20】

```
      program matprd
c                             ex15_1: 行列の積 {d} = [A(T)] [B] [A] {c}
      implicit real*8 (a-h,o-z)
      character*4 title(19),aend
      dimension a(8,8),b(8,8),c(8),d(8),e(8,8)
      data aend / 'end ' /
c
      open(5,file='inmatp.dat')
      open(6,file='outmatp.txt')
c
   20 continue
c                             タイトルで終了制御
      read(5,100) title
      if( title(1).eq.aend ) go to 30
      write(6,200) title
c
      read(5,600) mat
      if( mat.le.8.and.mat.gt.0 ) go to 40
      write(6,700)
      go to 30
   40 continue
      write(6,500) mat
      do 10 i=1,mat
      read(5,300) (a(i,j), j=1,mat)
      write(6,400) (a(i,j), j=1,mat)
   10 continue
      write(6,350)
      do 50 i=1,mat
      read(5,300) (b(i,j), j=1,mat)
      write(6,400) (b(i,j), j=1,mat)
   50 continue
      write(6,450)
      read(5,300) (c(i), i=1,mat)
      write(6,400) (c(i), i=1,mat)
c
      call prod (a,b,c,d,e,mat)
c
      write(6,550)
      write(6,400) (c(i), i=1,mat)
c
      go to 20
c
   30 continue
      close (5)
      close (6)
      stop
c
  100 format(19a4)
  200 format(//' ',19a4)
```

```
  300 format(8f10.0)
  350 format(' ',' [B]')
  400 format(' ',8(' |',1pd15.7),' |')
  450 format(' ',' {c}')
  500 format(' ','Size of matrices =',i5/' ','Given matrices:'/
     *       ' ',' [A]')
  550 format(' ','Obtained vector:'/' ',' {d}')
  600 format(i5)
  700 format(' ','***** Data error : 0<mat<9 *****')
      end
c -----------------------------------------------------------------------
      subroutine prod (a,b,c,d,e,mat)
      implicit real*8 (a-h,o-z)
      dimension a(8,1),b(8,1),c(1),d(1),e(8,1)
c
      do 10 i=1,mat
      d(i)=0.d0
      do 10 j=1,mat
      d(i)=d(i)+a(i,j)*c(j)
      e(i,j)=0.d0
      do 10 k=1,mat
      e(i,j)=e(i,j)+a(k,i)*b(k,j)
   10 continue
      do 20 i=1,mat
      c(i)=0.d0
      do 20 j=1,mat
      c(i)=c(i)+e(i,j)*d(j)
   20 continue
      return
      end
```

とできる。ここで一つ重要なのは，`subroutine` で

```
dimension a(8,1),b(8,1),c(1),d(1),e(8,1)
```

と宣言したことである。これは，当然，

```
dimension a(8,8),b(8,8),c(8),d(8),e(8,8)
```

でもいいが，必要なのは，多次元の配列の最後の添え字以外の宣言である。つまりその値は，呼んでいるルーチン側と同じ数値で宣言しなければならないことである。つまり，この例では，`a(i,j)` の `i` 側の上限値はきちんと呼ぶ側の宣言と同じサイズで宣言しなければならないことである。1 次元の場合と右端のサイズが “1” でいいのは，引数がその変数の番地だけを受け渡すからであり，その先頭の番地が呼ぶ側の変数の先頭の番地に等しいことを示すだけでいいからである。なぜ，右端のサイズ以外をきちんと宣言しなければならないかというと，実は多次元の配列はメモリの中では，例えばこの 2 次元の場合

```
a(1,1),a(2,1),a(3,1),,,,a(8,1),a(1,2),a(2,2),a(3,2),,,,
```

の順番に記憶されているからである。これは C 言語での順番と逆である。3 次元の例えば `c(2,3,4)` の場合は

```
c(1,1,1),c(2,1,1),c(1,2,1),c(2,2,1),c(1,3,1),c(2,3,1),c(1,1,2),,,
```

の順になっている。

さて，上の例題ではサイズ “mat” 分の配列だけを `subroutine` に渡せばいいはずだから

```
dimension a(mat,mat),b(mat,mat),c(mat),d(mat),e(mat,mat)
```

あるいは

```
dimension a(mat,1),b(mat,1),c(1),d(1),e(mat,1)
```

としたいと思うのは，多くの人間が考えることであろうが，実はとんでもないことが起こる。これをコンパイルしても全くエラーは検出されず，実行してもエラーが起こらない。しかし結果が目茶苦茶になる。それに気がつけばいいが，大ポカをすると人身事故にもつながりかねない。つまり，例えば mat=2 だったとすると，メモリ内の変数の配置は呼ぶ側と呼ばれる側とで，次のような対応になってしまうからである。

```
呼 ぶ 側:  a(1,1) a(2,1) a(3,1) a(4,1) a(5,1) a(6,1) .....
              Ok     Ok      ?      ?
呼ばれる側:  a(1,1) a(2,1) a(1,2) a(2,2)
```

ところが，これを次のように書き換えてしまうと，当り前だが正解が求められるから不思議なのである。理由については，じっくり腰を下ろして考えてみるといい。

【例題 21】

```
      program matprd
c                             ex15_2: 行列の積 {d} = [A(T)] [B] [A] {c}
      implicit real*8 (a-h,o-z)
      character*4 title(19),aend
      dimension a(8,8),b(8,8),c(8),d(8),e(8,8)
      data aend / 'end ' /
c
      open(5,file='inmatp.dat')
      open(6,file='outmatp.txt')
c
   20 continue
c                             タイトルで終了制御
      read(5,100) title
      if( title(1).eq.aend ) go to 30
      write(6,200) title
c
      read(5,600) mat
      if( mat.le.8.and.mat.gt.0 ) go to 40
      write(6,700)
      go to 30
   40 continue
      write(6,500) mat
c
      call matin (a,b,c,mat)
c
      call prod (a,b,c,d,e,mat)
c
      call matout (c,mat)
c
      go to 20
c
   30 continue
      close (5)
      close (6)
      stop
c
  100 format(19a4)
  200 format(//' ',19a4)
  500 format(' ','Size of matrices =',i5/' ','Given matrices:'/
     *       ' ',' [A]')
  600 format(i5)
  700 format(' ','***** Data error : 0<mat<9 *****')
      end
c ---------------------------------------------------------------------
      subroutine matin (a,b,c,mat)
      implicit real*8 (a-h,o-z)
      dimension a(mat,1),b(mat,1),c(1)
c
      do 10 i=1,mat
      read(5,300) (a(i,j), j=1,mat)
      write(6,400) (a(i,j), j=1,mat)
   10 continue
      write(6,350)
      do 50 i=1,mat
      read(5,300) (b(i,j), j=1,mat)
      write(6,400) (b(i,j), j=1,mat)
```

```
   50 continue
      write(6,450)
      read(5,300) (c(i), i=1,mat)
      write(6,400) (c(i), i=1,mat)
      return
  300 format(8f10.0)
  350 format(' ,' [B]')
  400 format(' ,8(' |',1pd15.7),' |')
  450 format(' ,' {c}')
      end
c -----------------------------------------------------------------
      subroutine prod (a,b,c,d,e,mat)
      implicit real*8 (a-h,o-z)
      dimension a(mat,mat),b(mat,mat),c(mat),d(mat),e(mat,mat)
c
      do 10 i=1,mat
      d(i)=0.d0
      do 10 j=1,mat
      d(i)=d(i)+a(i,j)*c(j)
      e(i,j)=0.d0
      do 10 k=1,mat
      e(i,j)=e(i,j)+a(k,i)*b(k,j)
   10 continue
      do 20 i=1,mat
      c(i)=0.d0
      do 20 j=1,mat
      c(i)=c(i)+e(i,j)*d(j)
   20 continue
      return
      end
c -----------------------------------------------------------------
      subroutine matout (c,mat)
      implicit real*8 (a-h,o-z)
      dimension c(mat)
      write(6,550)
      write(6,400) (c(i), i=1,mat)
      return
  400 format(' ,8(' |',1pd15.7),' |')
  550 format(' ,'Obtained vector:'/' ,' {d}')
      end
```

変更したのは，入力から計算・出力までをすべて subroutine でやるようにしたところ。この理由が明確に把握できるまでは・・・

鉄則 27: 呼ぶ側と呼ばれる側とで配列の宣言は完全に一致させる。

さてそうなると，よく BASIC でやるように配列の宣言を変数でできなくなるので，以前に書いた汎用性を持たせたプログラムを作成するのが難しくなる。また，上の例のプログラムで 8×8 の行列以上のものを処理できるようにするには，すべての dimension 文を書き改めなければならなくなり，煩雑な上に間違いを犯す頻度も増える。そこで，ある程度，以上のことが理解できるようになったら，次のような『整合配列』を利用すると，かなり汎用性を持たせたわかり易いプログラムを書くことができる。

次の**例題 22** のメインプログラムでは，実際の計算を一切行なっておらず，単に大きくとった配列 "a(8000)"（これを各機械に合わせて最大限とっておけばいい）をどのように切り刻んで，本当のメインプログラムに相当する matcal で使用する各種配列のメモリ上の格納場所を決めているだけである。あとはこの『実質的なメインプログラム』である subroutine を本当のメインプログラムと考えてプログラミングすればいい。しかもこの『メインプログラム』では， dimension のサイズを変数で宣言できる。このように配列名とサイズの両方が引数に含まれていれば，変数サイズの配列宣言ができ，これを『整合配列』と呼んでいる。

【例題 22】

```
      program matprd
c                          ex16: 行列の積 {d} = [A(T)] [B] [A] {c}
      implicit real*8 (a-h,o-z)
      character*4 title(19),aend
```

```
      common / dtarea / a(8000)
      data aend / 'end ' /
c                             データ用メモリの最大 = 8000 とした
      lasta=8000
      open(5,file='inmatp.dat')
      open(6,file='outmatp.txt')
c
   10 continue
c                             タイトルで終了制御
      read(5,100) title
      if( title(1).eq.aend ) go to 30
      write(6,200) title
c
      read(5,300) mat
      mmat=mat*mat
c                             各配列の番地を設定する
      addrs1=1
      addrs2=addrs1+mmat
      addrs3=addrs2+mmat
      addrs4=addrs3+mat
      addrs5=addrs4+mat
      lastad=addrs5+mmat-1
c                                       最後の番地が許容できるか?
      if( lastad.gt.lasta ) go to 20
c                                                   本当のメインへ
      call matcal (a(addrs1),a(addrs2),a(addrs3),
     *                                  a(addrs4),a(addrs5),mat)
c
      go to 10
c
   20 continue
      write(6,400)
   30 continue
      close (5)
      close (6)
      stop
c
  100 format(19a4)
  200 format(//' ',19a4)
  300 format(i5)
  400 format(' ','***** Data error : mat is too large! *****')
c
      end
c ----------------------------------------------------------------------
      subroutine matcal (a,b,c,d,e,mat)
      implicit real*8 (a-h,o-z)
      dimension a(mat,mat),b(mat,mat),c(mat),d(mat),e(mat,mat)
c
      write(6,100) mat
      do 10 i=1,mat
      read(5,200) (a(i,j), j=1,mat)
      write(6,300) (a(i,j), j=1,mat)
   10 continue
      write(6,400)
      do 20 i=1,mat
      read(5,200) (b(i,j), j=1,mat)
      write(6,300) (b(i,j), j=1,mat)
   20 continue
      write(6,500)
      read(5,200) (c(i), i=1,mat)
      write(6,300) (c(i), i=1,mat)
c
      do 30 i=1,mat
      d(i)=0.d0
      do 30 j=1,mat
      d(i)=d(i)+a(i,j)*c(j)
      e(i,j)=0.d0
      do 30 k=1,mat
      e(i,j)=e(i,j)+a(k,i)*b(k,j)
   30 continue
c
      do 40 i=1,mat
      c(i)=0.d0
```

```
      do 40 j=1,mat
      c(i)=c(i)+e(i,j)*d(j)
   40 continue
c
      write(6,600)
      write(6,300) (c(i), i=1,mat)
c
      return
c
  100 format(' ','Size of matrices =',i5/' ','Given matrices:'/
     *       ' ',' [A]')
  200 format(8f10.0)
  300 format(' ',8(' |',1pd15.7),' |')
  400 format(' ',' [B]')
  500 format(' ',' {c}')
  600 format(' ','Obtained vector:'/' ',' {d}')
      end
```

鉄則 28: 大人になったら整合配列。

この整合配列はさらに，効率的にメモリを利用するのにも便利[9]である。例えばあるプログラムが，それぞれ大きな三つのステップで構成されているとする。それぞれのステップを subroutine としたとき，そこで必要な配列が

ステップ 1: sub1
要保存: a(n,n), b(n), c(m,n)
作業用: e(n), f(n,m)
ステップ 2: sub2
要保存: a(n,n), c(m,n), d(m)
作業用: e(m,m), f(m)
ステップ 3: sub3
要保存: b(n), d(m)
作業用: e(m,m,n)

だったとする。要保存の配列は

ステップ 3 まで: b(n), d(m)
ステップ 2 まで: a(n,n), c(m,n)

であり，残りは作業用だから各ステップで捨ててしまっても構わない。この場合は次のようなプログラムを書くことができる。ただし，作業用のエリアは各ルーチンできちんと初期化しないといけない。

【例題 23】

```
      program c_arry
c                          ex16_1: 整合配列
      implicit real*8 (a-h,o-z)
      parameter (dtmaxa=8000)
      common / dtarea / a(dtmaxa)
c
      read(5,100) n,m
c
      n1=1
      n2=n1+n
      n3=n2+m
      n4=n3+n*n
      n5=n4+m*n
      n6=n5+n
      nlast=n6+n*m-1
      if( nlast.gt.dtmaxa ) stop
c
      call sub1 (a(n3),a(n1),a(n4),a(n5),a(n6),n,m)
```

[9] open source の汎用構造解析プログラム 'SAP' および 'NonSAP' で使われていた方法である。

```
c
      n6=n5+m*m
      nlast=n6+m-1
      if( nlast.gt.dtmaxa ) stop
c
      call sub2 (a(n3),a(n4),a(n2),a(n5),a(n6),n,m)
c
      nlast=n3+m*m*n-1
      if( nlast.gt.dgmaxa ) stop
c
      call sub3 (a(n1),a(n2),a(n3),n,m)
c
      stop
  100 format(2i5)
      end
c ---------------------------------------------------------------------
      subroutine sub1 (a,b,c,e,f,n,m)
      implicit real*8 (a-h,o-z)
      dimension a(n,n),b(n),c(m,n),e(n),f(n,m)
c
      return
      end
c ---------------------------------------------------------------------
      subroutine sub2 (a,c,d,e,f,n,m)
      implicit real*8 (a-h,o-z)
      dimension a(n,n),c(m,n),d(m),e(m,m),f(m)
c
      return
      end
c ---------------------------------------------------------------------
      subroutine sub3 (b,d,e,n,m)
      implicit real*8 (a-h,o-z)
      dimension b(n),d(m),e(m,m,n)
c
      return
      end
```

メインプログラムでは番地の処理以外のいかなる作業もしてはならないことに注意すべきである。

L.3 いくつかの応用例

L.3.1 2 分法による求解

片端固定・片端ローラー単純支持柱の弾性圧縮座屈荷重 $P_{\rm cr}$ は

$$f(x) \equiv \tan(x) - x = 0 \tag{L.1}$$

で求められる。ここに， $x^2 \equiv \dfrac{\ell^2 P_{\rm cr}}{EI}$ であり，最小の値は $x = 4.493$ である。この特性方程式の解を 2 分法で求める。

2 分法は解をはさむ領域を探し，最も情報量の大きい中間点を領域の修正点として用い，その領域を小さくしていく方法（図 L.2 参照）である。簡単な計算である上に，会話型で行なった方が便利であるため，入出力は標準入出力のキーボードと画面とする。この場合，ほとんどの計算機ではファイル番号を指定せず，“*” とするだけでいい。つまり， `read(*,*)`, `write(*,*)` 等である。

【例題 24】

```
      program bisect
c                          ex17: f(x)=tan(x)-x=0 の解は?
      implicit real*8 (a-h,o-z)
c                                  キーボード入力・画面出力
   10 continue
      write(*,510)
      read(*,*) eps
      if( eps.le.0.d0.or.eps.ge.1.d0 ) go to 20
      write(*,520) eps
```

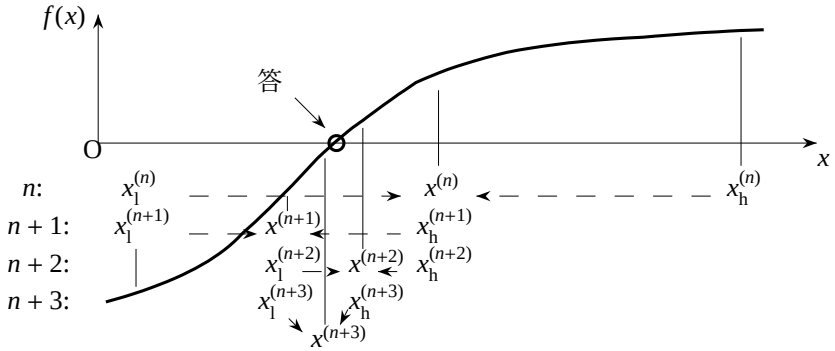

図 L.2 2 分法の考え方 — 各ステップ毎に大きい予想 $x_{\rm h}^{(i)}$ と小さい予想 $x_{\rm l}^{(i)}$ の中央値 $x^{(i)}$ を新予想とし，次のステップの予想の上下限値のどちらかを改善していく方法で，非効率的ではあるが精度を容易に上げることができる方法

```
c
   30 continue
      write(*,530)
      read(*,*) xl
      write(*,540) xl
      write(*,550)
      read(*,*) xh
      write(*,560) xh
      dx=(xh-xl)/1.d1
c
      do 70 i=0,10
      x=xl+float(i)*dx
      write(*,600) x,f(x)
   70 continue
c
      fl=f(xl)
      fh=f(xh)
      if( fl*fh.le.0.d0 ) go to 40
      write(*,570)
      go to 30
c
   40 continue
      x=(xl+xh)/2.d0
      fc=f(x)
      if( abs((xh-xl)/x).le.eps ) go to 50
      if( fc*fl.le.0.d0) then
          xh=x
      else
          xl=x
          fl=fc
      end if
      go to 40
c
   50 continue
      if( abs(fc).le.1.d0 ) go to 60
      write(*,580)
      go to 30
   60 continue
      write(*,590) x,fc
      go to 10
c
   20 continue
      stop
c
  510 format(//' ','*** Bisection method to solve f(x)=0 ***'/
     *         ' ',' Tolerance (Enter 0.0 to stop) ?')
  520 format(' ',5x,'Tolerance =',1pd15.7)
  530 format(' ',' Lower  trial value ?')
  540 format(' ',5x,'Lower  trial value =',1pd15.7)
  550 format(' ',' Higher trial value ?')
  560 format(' ',5x,'Higher trial value =',1pd15.7)
```

```
  570 format(/' ','Both of f(xh) and f(xl) have the same sign.',
     *            'Try again !!!'/)
  580 format(' ','No solution there !!! Try again !!!'/)
  590 format(' ',' >>> Possible solution =',1pd15.7,' <<<',
     *       5x,'(f(x)=',d15.7,')')
  600 format(' ',10x,'x=',1pd15.7,':  f(x)=',d15.7)
      end
c -----------------------------------------------------------------------
      real*8 function f(x)
      implicit real*8 (a-h,o-z)
      f=tan(x)-x
      return
      end
```

はさみこむ値によっては全然答えが求められないことがあることに注意する。

“real*8 function f(x)” が『関数副プログラム』と呼ばれるものである。この問題のように簡単な関数の場合には，『文関数』を利用してもいい。つまり，メインプログラムの先頭の暗黙の型宣言のすぐ次の行で関数を定義する方法で，この例の場合は

【例題 25】

```
      program bisect
c                             ex17: f(x)=tan(x)-x=0 の解は?
      implicit real*8 (a-h,o-z)
      f(z)=tan(z)-z
c                                   キーボード入力・画面出力
   10 continue
      .....
```

となる。実行してみよ。

L.3.2 Newton-Raphson 法

同じ問題を， Newton-Raphson 法で解く。これは，答えの推定値を今 x_n としたとき，それは正解ではないから $f(x_n) \neq 0$ である。これを Δx_n だけ修正して答えを求めたい場合に， $x = x_n$ の近傍で Taylor 展開して， 1 次項までだけをとり

$$f(x_n + \Delta x_n) = f(x_n) + \left\{ \frac{\mathrm{d}f}{\mathrm{d}x}(x_n) \right\} \Delta x_n = 0 \tag{L.2}$$

によって求めようというものである。したがって，次の漸化式

$$x_{n+1} = x_n + \Delta x_n = x_n - \frac{f(x_n)}{\dfrac{\mathrm{d}f}{\mathrm{d}x}(x_n)} \tag{L.3}$$

を繰り返すことによって，最終的に Δx_n が必要な精度（プログラム中では eps の大きさで規定した値）以内になったところで，答えを求める方法である。

【例題 26】

```
      program nwrphs
c                             ex18: f(x)=tan(x)-x=0 の解は?
c                                   by Newton-Raphson method
      implicit real*8 (a-h,o-z)
c                                   キーボード入力・画面出力
      nmax=20
c
   10 continue
      write(*,510)
      read(*,*) eps
      if( eps.le.0.d0.or.eps.ge.1.d0 ) go to 20
      write(*,520) eps
c
   30 continue
      iter=0
      write(*,530)
      read(*,*) x
```

```
      write(*,540) x
c
   40 continue
      if( iter.eq.nmax ) go to 50
      iter=iter+1
      dx=-f(x)/df(x)
      x =x+dx
      error=abs(dx/x)
      write(*,570) iter,x,f(x),error
      if( error.gt.eps ) go to 40
c
      write(*,550) x,f(x)
      go to 10
c
   50 continue
      write(*,560) nmax
      go to 30
c
   20 continue
      stop
c
  510 format(//' ','*** Newton-Raphson method to solve f(x)=0 ***'/
     *       ' ',' Tolerance (Enter 0.0 to stop) ?')
  520 format(' ','Tolerance =',1pd15.7)
  530 format(' ',' First trial value for x ?')
  540 format(' ','First trial value =',1pd15.7)
  550 format(' ',' >>> Possible solution =',1pd15.7,' <<<',
     *       5x,'(f(x)=',d15.7,')')
  560 format(' ','No convergence within iteration ',i5,
     *          ' !!! Try again !!!'/)
  570 format(' ',' iter=',i5,':  x=',1pd15.7,':  f(x)=',d15.7,
     *          ':  error=',d15.7)
      end
c ----------------------------------------------------------------------
      real*8 function f(x)
      implicit real*8 (a-h,o-z)
      f=tan(x)-x
      return
      end
c ----------------------------------------------------------------------
      real*8 function df(x)
      implicit real*8 (a-h,o-z)
      df=1.d0/cos(x)/cos(x)-1.d0
      return
      end
```

これも初期値の与え方によっては全然答が求められない場合があるのは前と同じである。特に接線$\left(\dfrac{\mathrm{d}f}{\mathrm{d}x}\right)$が零に近づくと非常に困ったことが起こる。

L.3.3 連立方程式の Newton-Raphson 法

次に連立方程式になった場合を考える。ここでの問題は

$$f_1(x,y) = x^2 - 3\,x\,y + 8.1 = 0 \tag{L.4a}$$

$$f_2(x,y) = y^3 - 2\,x^2\,y - 5\,x - 15.8 = 0 \tag{L.4b}$$

とした。これも同様に Taylor 展開して

$$\begin{aligned} f_k(x_{n+1},y_{n+1}) = {} & f_k(x_n,y_n) \\ & + \frac{\partial f_k}{\partial x}(x_n,y_n)\left(x_{n+1}-x_n\right) + \frac{\partial f_k}{\partial y}(x_n,y_n)\left(y_{n+1}-y_n\right) = 0, \quad k = 1,2 \end{aligned} \tag{L.5}$$

を解けばいいので

$$\begin{Bmatrix} x_{n+1} \\ y_{n+1} \end{Bmatrix} = \begin{Bmatrix} x_n \\ y_n \end{Bmatrix} - \begin{pmatrix} \dfrac{\partial f_1}{\partial x}(x_n, y_n) & \dfrac{\partial f_1}{\partial y}(x_n, y_n) \\ \dfrac{\partial f_2}{\partial x}(x_n, y_n) & \dfrac{\partial f_2}{\partial y}(x_n, y_n) \end{pmatrix}^{-1} \begin{Bmatrix} f_1(x_n, y_n) \\ f_2(x_n, y_n) \end{Bmatrix} \tag{L.6}$$

を繰り返せばいい。

一般に Newton-Raphson 法は，繰り返す度に次の捜索のための接線係数，つまり上式の関数の微係数でできた行列を更新する。そのため，いわゆる 2 次の収束となって比較的速く正解に近づいていく。これに対し，何らかの理由で接線係数を更新したくない場合には，例えば，繰り返しの 1 回目の接線係数だけを用いて（プログラム中の `call tangnt` を 1 回だけしか計算しないで）漸化式を近似的に解くこともできる。この場合は一度だけ逆行列を計算すれば済むというメリットもあるが，収束は遅くなる上に，収束半径が小さいことから収束するような初期値を選ぶのが比較的難しくなる。図 L.3 は連立方程式の場合の一例で，繰り返す度に誤差（前回の近似解との差）がどのように変化するかが示されている。図中の ‘NR’ で示した線が正しい Newton-Raphson 法の場合の収束の様子で，精度（プログラム中の `eps`）を 10^{-7} としたときに 5 回で近似解が得られていることを示している。これに対し，接線係数を更新しない場合の結果が ‘qNR’ で示した線であるが，かなり収束は遅くなり， 13 回でようやく所定の精度が得られている。

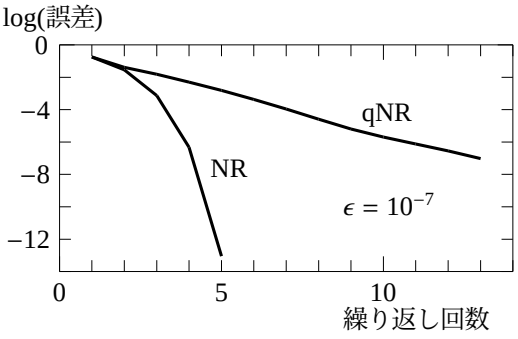

図 L.3 Newton-Raphson 法の収束

【例題 27】

```
      program nwrph2
c                              ex19: 連立 f(x)=0 の解は?
c                                    by Newton-Raphson method
      implicit real*8 (a-h,o-z)
      dimension a(2,2),x(2),f(2),dx(2)
c
      nmax=20
c
   10 continue
      write(*,510)
      read(*,*) eps
      if( eps.le.0.d0.or.eps.ge.1.d0 ) go to 20
      write(*,520) eps
c
   30 continue
      iter=0
      write(*,530)
      read(*,*) x(1),x(2)
      write(*,540) x(1),x(2)
c
   40 continue
      if( iter.eq.nmax ) go to 50
      iter=iter+1
      f(1)=f1(x)
      f(2)=f2(x)
      call tangnt (a,x)
      do 60 i=1,2
      dx(i)=0.d0
      do 60 j=1,2
      dx(i)=dx(i)-a(i,j)*f(j)
   60 continue
      do 70 i=1,2
      x(i)=x(i)+dx(i)
   70 continue
      error=sqrt((dx(1)**2+dx(2)**2)/(x(1)**2+x(2)**2))
```

```
      write(*,570) iter,x(1),x(2),error
      if( error.gt.eps ) go to 40
c
      write(*,550) x(1),x(2)
      go to 10
c
   50 continue
      write(*,560) nmax
      go to 30
c
   20 continue
      stop
c
  510 format(//' ','*** Newton-Raphson method to solve f(x,y)=0 ***'/
     *       ' ',' Tolerance (Enter 0.0 to stop) ?')
  520 format(' ','Tolerance =',1pd15.7)
  530 format(' ',' First trial value for x & y ?')
  540 format(' ','First trial value (x,y) =',1p2d15.7)
  550 format(' ',' >>> Possible solution (x,y) =',1p2d15.7,' <<<')
  560 format(' ','No convergence within iteration ',i5,
     *          ' !!! Try again !!!'/)
  570 format(' ',' iter=',i5,':  x=',1pd15.7,':  y=',d15.7,
     *          ':  error=',d15.7)
      end
c ---------------------------------------------------------------------
      subroutine tangnt (a,x)
      implicit real*8 (a-h,o-z)
      dimension a(2,2),x(2)
      a(1,1)=2.d0*x(1)-3.d0*x(2)
      a(1,2)=-3.d0*x(1)
      a(2,1)=-4.d0*x(1)*x(2)-5.d0
      a(2,2)=3.d0*x(2)*x(2)-2.d0*x(1)*x(1)
c
      det=a(1,1)*a(2,2)-a(1,2)*a(2,1)
      b=a(2,2)/det
      a(2,2)=a(1,1)/det
      a(1,1)=b
      a(1,2)=-a(1,2)/det
      a(2,1)=-a(2,1)/det
      return
      end
c ---------------------------------------------------------------------
      real*8 function f1(x)
      implicit real*8 (a-h,o-z)
      dimension x(2)
      f1=x(1)*x(1)-3.d0*x(1)*x(2)+8.1d0
      return
      end
c ---------------------------------------------------------------------
      real*8 function f2(x)
      implicit real*8 (a-h,o-z)
      dimension x(2)
      f2=x(2)*x(2)*x(2)-2.d0*x(1)*x(1)*x(2)-5.d0*x(1)-1.58d1
      return
      end
```

L.3.4 ソーティングと最小 2 乗法

最後に，実験データ等の整理で (x, y) の組を x の小さい順に並べ変えたり， x と y の関係を最小 2 乗法で求める場合を想定する。ソーティングは最も簡単な，一つ一つ比較して並べ代える最も時間のかかる方法を示した。最小 2 乗法にも種々あるが，ここでは最も簡単な， x を真値とした線形回帰を行なう。例えば， N 組のデータ (x, y) に対して， $y = a\,x + b$ で回帰するためには

$$\text{誤差} \equiv y_n - (a\,x_n + b), \qquad n = 1 \sim N \tag{L.7}$$

だから，

$$\sum(\text{誤差})^2 = \sum \{y_n - (a\,x_n + b)\}^2 \quad \rightarrow \quad \text{最小} \tag{L.8}$$

とすればいい。

したがって

$$\frac{\partial}{\partial a}\sum\{y_n-(a\,x_n+b)\}^2=0,$$
$$\frac{\partial}{\partial b}\sum\{y_n-(a\,x_n+b)\}^2=0 \qquad \text{(L.9)}$$

よって，この 2 式から a と b を求めようとすると

$$\begin{Bmatrix} a \\ b \end{Bmatrix}=\begin{pmatrix} \sum x_n^2 & \sum x_n \\ \sum x_n & \sum 1 \end{pmatrix}^{-1}\begin{Bmatrix} \sum x_n y_n \\ \sum y_n \end{Bmatrix} \qquad \text{(L.10)}$$

となる。

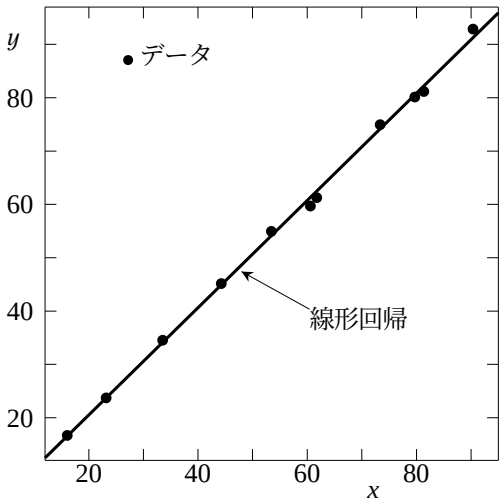

図 L.4 データの並べ替えと線形回帰

【例題 28】

```
      program l_regr
c           ex20: ソートと最小 2 乗法
      implicit real*8 (a-h,o-z)
      character*4 title(19),aend
      common / dtarea / a(8000)
      data aend / 'end ' /
c
      lasta=8000
      open(5,file='regrin.dat')
      open(6,file='regrout.txt')
c                                   グラフィックス用ファイル
      open(1,file='regrph.dat')
c
   10 continue
      read(5,100) title
      if( title(1).eq.aend ) go to 30
      write(6,200) title
c                                   タイトルもグラフィックス用ファイルに
      write(1,500) title
      read(5,300) nn
      addrs1=1
      addrs2=addrs1+nn
      lastad=addrs2+nn-1
      if( lastad.gt.lasta ) go to 20
c
      call regr (a(addrs1),a(addrs2),nn)
      go to 10
c
   20 continue
      write(6,400)
   30 continue
      close (1)
      close (5)
      close (6)
      stop
c
  100 format(19a4)
  200 format(//' ',19a4)
  300 format(i5)
  400 format(' ','***** Data error : nn is too large! *****')
  500 format(' ',19a4)
c
      end
c
      subroutine regr (x,y,nn)
      implicit real*8 (a-h,o-z)
      dimension x(nn),y(nn)
c
      write(6,100) nn
      do 10 i=1,nn
      read(5,200) x(i),y(i)
   10 continue
c                             小さい順に並べる
```

```
      do 20 i=1,nn-1
      do 20 j=i+1,nn
      if( x(i).le.x(j) ) go to 20
      xy=x(i)
      x(i)=x(j)
      x(j)=xy
      xy=y(i)
      y(i)=y(j)
      y(j)=xy
   20 continue
c                             グラフィックスデータ用出力も
      do 30 i=1,nn
      write(6,300) x(i),y(i)
      write(1,600) x(i),y(i)
   30 continue
c                             最小 2 乗法の係数計算
      sumxx=0.d0
      sumxy=0.d0
      sumx =0.d0
      sumy =0.d0
      do 40 i=1,nn
      sumxx=sumxx+x(i)*x(i)
      sumxy=sumxy+x(i)*y(i)
      sumx =sumx +x(i)
      sumy =sumy +y(i)
   40 continue
c
      a=sumxy/sumxx
      write(6,400) a
c
      det=sumxx*float(nn)-sumx*sumx
      a=(float(nn)*sumxy-sumx*sumy)/det
      b=(-sumx*sumxy+sumxx*sumy)/det
      write(6,500) a,b
c
      return
c
  100 format(' ','Number of data =',i5/' ','Sorted data (x,y):')
  200 format(2f10.0)
  300 format(' ',5x,'x=',1pd15.7,': y=',d15.7)
  400 format(' ',2x,'Linear regression : y = (',1pd15.7,' ) * x')
  500 format(' ',2x,'Linear regression : y = (',1pd15.7,' ) * x + (',
     *           d15.7,' )')
c                             グラフィックス用は例えば E 形式出力
  600 format(' ',1p2e15.7)
c
c     data example
c     ----+----1----+----2
c     Data No.1
c        11
c         23.186    23.754
c         81.354    81.192
c         73.363    74.974
c         44.265    45.186
c         61.761    61.290
c         33.535    34.550
c         16.067    16.713
c         60.598    59.715
c         79.709    80.166
c         90.358    92.894
c         53.447    54.974
c     end_                  <-- 4 文字目はスペース！
c     ----+----1----+----2
      end
```

L.3.5 計算を途中で止めたり継続したりするには

構造物の非線形挙動を追跡するような，長時間かかるシミュレーションの場合，せっかくある荷重レベルまで計算したのに，計算がなんらかの原因（停電等）で止まってしまった場合，また最初からやり直しになる。このような場合には，各荷重ステップのデータを一時ファイルに保存しておけば，止まったところからまた計

算を始めることができ，効率的[10]である。例えば，2 番のファイルをそのように使う例を書くと

【例題 29】

```
      dimension what(1000),is(1000),neces(1000),sarry(1000)
c
      read(5,100) icont
      open(2,file='temp.dat',form='unformatted')
c                                  icont=0: 初めて      =1: 中断継続
      if( icont.eq.0 ) go to 10
c                                  継続の場合，読んで巻戻し
      read(2) what,is,neces,sarry
      rewind 2
c                                  実際の計算はここから開始
   10 continue
c
c       ここで各荷重ステップでの長い計算をする
c
c       そして，結果が出たとする     必要なデータだけ記憶しておく
c
      write(2) what,is,neces,sarry
      rewind 2
c       まだ続けるなら・・・
      if( inextstep.eq.0 ) go to 10
c
      close(2)
      stop
      end
```

というようにできる。最後のステップの結果だけを覚えておけばいいから，書き込んむ毎に “rewind” で巻戻してある。人間が読めなくてもいいので，“unformatted” 形式にしてファイルの大きさを小さくしてある。

L.4 その他の一般的なこと

プログラム単位中の文の順序:

program 文や subroutine 文・ function 文
implicit 文
dimension 文や common 文
data 文
文関数定義文
このうしろに算術代入文等の実行文
end 文

となる。注釈行は end 文の前ならどこでもいい。 format 文も program 文等のうしろで end 文までの間ならどこでもいいが， end 文のすぐ前か，各宣言文の直後にまとめた方が見易い。

間違いの無いプログラムを書く順序:

1. プログラムする式をすべて紙にきれいに書く。紙に書けない式はプログラムもできない！
2. 紙に書いた式で，フローチャートをおおまかに書く。短いプログラムならば頭の中に描いてもいい。
3. 書いた式に現われるすべての定数・変数を表にする。そして，それに対応するプログラム中の変数名を書く（変数表の作成）。
4. 各変数の初期化をどうするかを変数表に書き込む。どれが data 文でどれが代入文，どれが read 文による入力か。
5. read 文による入力変数の順番と format を決める（input フォーマットの作成）。これはそのまま他人へのユーザーズマニュアルに流用できるくらい詳細でなければならない。

[10] これも open source の汎用構造解析プログラム ‘SAP’ および ‘NonSAP’ で使われていた方法だったと思う。

6. どの変数を出力するのかを変数表に書き込む。これで，ユーザーズマニュアルの出力書式説明の資料ができる。
7. プログラムを作成開始する。
8. たいていの場合は，幸運を祈ってがんばる。
9. 一度もエラーが無く答が出たときは，まず答を疑う。たいていの場合，間違っている。
10. 一週間以上前に作ったプログラムを変更する場合は，絶対に元のファイルを変更しないで，別のファイル（プログラム名も違うものにする）にして作業する。

以上，第 1 著者が常々やるべきだなぁと思いながら，やっていない手順である。