C言語による
計算プログラミング入門
第2版

2002年10月

島根大学総合理工学部
電子制御システム工学科

吉田和信

目 次

第1章　はじめに

C 言語は，1972 年に Dennis M. Ritchie によって DEC 社製の PDP-11 上で動く **UNIX** オペレーティングシステム用に設計され，実用化されたもので，それ以来，UNIX の普及とともに，世界中で広く使われるようになってきた．実際，C のコンパイラは，パーソナルコンピュータ用からスーパーコンピュータ用に至るまで作られている．C がこのように普及した理由として

1. UNIX の基本言語である．
2. 実行速度が速い．
3. メモリ操作の自由度が大きい．
4. 構造化プログラミングができる．
5. 豊富なライブラリを利用できる．

などが挙げられる．また，C は Matlab[1]などの数値計算パッケージとのリンク（結合）が容易であり，A/D・D/A コンバータなどの計測制御周辺機器の制御用言語としてもよく用いられる．

本テキストは，C による計算プログラミングを平易に解説したものである．例題を用いて，具体的にプログラミングを説明するというスタイルをとった．また，演習問題を適量配置した．演習問題の解答はあえて掲載していない．自ら考えて，実力を涵養して頂きたいという配慮からである．

扱った計算テーマは，工科系学部で必要なものの中から特に基礎的かつ本質的なものを選んだ．実用的で興味深いプログラムが揃っていると思う．これらは，今後学習することになる数学，物理などの基礎的科目や計算に関係する専門科目を履修する際に役立つであろう．

テキストの前半（第 1 章から第 6 章まで）では，C の文法を詳述することを避け，基本的なスカラー計算に的を絞って各テーマを展開する．すなわち，C プログラミング入門（変数，数式，入出力，フロー制御），非線形方程式の解法，数値積分，ランダムサーチ，常微分方程式の解法を述べる．後半では，第 7 章で C の文法を系統的に説明し，第 8 章で配列を用いるベクトル計算を中心としたプログラミングを解説する．

学部および大学院を通じて，本テキストが必要に応じて参照されるように，内容を充実させてある．特に，第 6 章，第 8 章の一部は，やや高度な内容となっているので，1・2 年次の講義で使用する場合，指導教員の判断で適宜割愛されたい．

[1]Matlab は，米国 The MathWorks Inc. の登録商標です．

プログラミング上達のコツは，自ら，プログラムを理解し，計算機に打ち込み，コンパイル・実行し，その結果を検討することの繰り返しにある．最初の内は，コンパイル時にプログラムエラーを指摘されることが多いであろうが，根気強く続けてほしい．

本テキストが，計算プログラミング技術習得の一助となることを願っている．

第2章　Cプログラミング入門

2.1　文字の出力

画面に hello，world と表示する C プログラムは次のとおりである．

例 2.1

```
#include <stdio.h>
main()
{
   printf("hello, world\n");
}
```

このプログラムをエディタで作成して**ファイル名**に**拡張子**‘.c’を付けて保存する．ファイル名を例えば hello.c としよう．次に

```
cc hello.c
```

で hello.c を**コンパイル**すると a.out という**実行可能型ファイル**ができる．これを

```
./a.out
```

で実行すればよい．

実行結果

```
hello, world
```

例 2.1 の説明

1. #include <stdio.h>は決り文句である（入出力の仕事に必要な情報を取り込めという命令）．
2. C では main というプログラム単位（**関数**と呼ばれる）の先頭から実行するので，main(){...}の中に主プログラムを書く．
3. printf は**文字列**"..."の内容を画面に表示する関数である．
4. ‘\n’（または‘¥n’）は**改行**を意味する．
5. 文の終わりには‘;’を書く．
6. "..."以外の場所での改行や空白はコンパイル時に無視される．

演習 2.1 例 2.1 のプログラムで \n を省くとどうなるかを確かめよ．

演習 2.2 例 2.1 のプログラムは次のように書くこともできる．

```
#include <stdio.h>
main()
{
   printf("hello, ");
   printf("world");
   printf("\n");
}
```

このことを確認せよ．

演習 2.3 次の文を画面へ出力するプログラムを作れ（同じ箇所で改行せよ）．

```
The best way to learn
programming is to
dive right in and
start writing real programs.
```

演習 2.4 次の表を出力するプログラムを作成せよ．

```
1
2 3
4 5 6
7 8 9 10
```

演習 2.5 特殊文字

```
\   ?   '   "   %
```

は C プログラムにおいてそれぞれ役割をもっている．これらの文字を出力する場合 '\' を前に付ける．ただし，'%' は例外で，これを出力するには%%と書く．上記の特殊文字を出力するプログラムを作成せよ．

2.2　変数の宣言と基本的演算

数値の格納場所として**変数**を宣言する．変数は関数の冒頭で定義され，その関数中で有効となる．次の例を検討しよう．

例 2.2

```
#include <stdio.h>
main()
```

```
{
   int a,b;       /* integers */
   double x,y;    /* floating-point numbers */
   a = 2;
   b = 3*a*a;
   x = 2.0;
   y = 3.0*x*x;
   printf("a = %d  b = %d\n",a,b);
   printf("x = %f  y = %f\n",x,y);
}
```

実行結果

```
a = 2  b = 12
x = 2.000000  y = 12.000000
```

例 2.2 の説明

1. `int a,b;` で**整数型**の変数 `a`，`b` を宣言している．
2. `double x,y;` で**倍精度実数型**の変数 `x`，`y` を宣言している．
3. プログラムを読みやすくするため，適宜**コメント**を‘`/* */`’で囲んで挿入できる．
4. `int` 型の代入文・式では `int` 型の**定数**（小数点を付けない）を用いる（`a = 2; b = 3*a*a;`）．
5. `double` 型の代入文・式では `double` 型の定数（小数点を付ける）を用いる（`x = 2.0; y = 3.0*x*x;`）．
6. `printf` において，**書式**‘`%d`’で整数が，書式‘`%f`’で実数が出力できる．
7. **変数名**で使える文字は，英字，アンダースコア‘_’，数字であり，英字か‘_’を最初に置く．大文字と小文字は区別される．長さは無制限であるが，コンパイラが認識できる文字数は限られている（例 31 文字）．

計算で用いる基本的な**演算子**を表 2.1 に示す．
`printf` 関数において，表 2.2 のような書式を使えば表示桁数を指定できる．

演習 2.6 `int` 型の変数 a, b を用いて

$$a+b,\quad a-b,\quad ab,\quad a/b,\quad a \text{ を } b \text{ で割ったときの剰余} \tag{2.1}$$

を計算して出力するプログラムを作成せよ．a, b の値は適当に設定せよ．（整数同士の割り算は小数点以下切り捨てとなる．）

演習 2.7 `double` 型の変数 x, y を用いて

$$x+y,\quad x-y,\quad xy,\quad x/y \tag{2.2}$$

を計算して出力するプログラムを作成せよ．x, y の値は適当に設定せよ．

表 2.1: 基本的な演算子

演算子	意味
=	代入（右辺を左辺に）
+	和
-	差
*	積
/	商
%	剰余（int 型に適用）

表 2.2: 書式の例

書式の例	意味
%5d	少なくとも 5 文字幅に整数として印字
%6.2f	少なくとも 6 文字幅で，小数点以下 2 桁の浮動小数点数として印字
%.4f	小数点以下 4 桁の浮動小数点数として印字
%.2e	小数点以下 2 桁の浮動小数点数（指数付き表示）として印字

2.3　数値の入力

scanf 関数を使えば変数に数値を入力できる．

例 2.3

```
#include <stdio.h>
main()
{
   int a,b;
   double x,y;
   scanf("%d %d",&a,&b);
   scanf("%lf %lf",&x,&y);
   printf("a = %d  b = %d\n",a,b);
   printf("x = %f  y = %f\n",x,y);
}
```

実行結果

```
1 2
3 4
a = 1  b = 2
x = 3.000000  y = 4.000000
```

例 2.3 の説明

1. scanf 関数の書き方に注意する．int 型の変数への入力には書式 '%d' を用い，double 型の変数への入力には書式 '%lf'（エルエフ）を用いる．また，変数の前には '&' を付ける．

演習 2.8 例 2.3 の `scanf("%d %d",&a,&b);` の前に

```
printf("Enter a b\n");
```

を挿入すれば，入力促進メッセージ（入力プロンプト）が表示される．例 2.3 を変更して，入力プロンプトを表示するプログラムにせよ．

演習 2.9 int 型の変数 a, b, c, d に数値を入力し

$$e = abcd \tag{2.3}$$

を計算して出力するプログラムを作れ．

演習 2.10 double 型の変数 a, b, c, d に数値を入力し

$$e = \frac{1}{abcd} \tag{2.4}$$

を計算して出力するプログラムを作れ．ただし，C では $1/(abcd)$ を

`1.0/(a*b*c*d)` または `1.0/a/b/c/d`

と書く．

2.4 数式の書き方

x を入力して次の関数値を計算するプログラムを次に示す．

$$y_1 = \frac{1}{(x+1)(x+2)}, \quad y_2 = x^5, \quad y_3 = \sqrt{x}, \quad y_4 = \sqrt[3]{x} \tag{2.5}$$

$$y_5 = e^{-2x}, \quad y_6 = \sin x, \quad y_7 = \cos^2 x \tag{2.6}$$

例 2.4

```
#include <stdio.h>
#include <math.h>
main()
{
   double x,y1,y2,y3,y4,y5,y6,y7;
   printf("Enter x\n");
   scanf("%lf",&x);
```

```
    y1 = 1.0/((x+1.0)*(x+2.0));
    y2 = pow(x,5);
    y3 = sqrt(x);          /* or y3 = pow(x,0.5);  */
    y4 = pow(x,1.0/3.0);
    y5 = exp(-2.0*x);
    y6 = sin(x);
    y7 = cos(x)*cos(x);   /* or y7 = pow(cos(x),2); */
    printf("y1 = %f  y2 = %f  y3 = %f  y4 = %f\n", y1,y2,y3,y4);
    printf("y5 = %f  y6 = %f  y7 = %f\n",y5,y6,y7);
}
```

実行結果

```
Enter x
2
y1 = 0.083333  y2 = 32.000000  y3 = 1.414214  y4 = 1.259921
y5 = 0.018316  y6 = 0.909297  y7 = 0.173178
```

例 2.4 の説明

1. sin, cos などの**数学関数**を利用するために`#include <math.h>`が必要である．
2. ‘`( )`’ の入れ子は無制限である．
3. ‘`*`’ は省略できないので注意する．
4. x^a は `pow(x,a)` で計算できる．`a` の型は `double` であるが

   ```
   y2 = pow(x,5);
   ```

   のように，`a` として整数を与えてもよい（自動的に `double` に変換される）．ちなみに，変数の型は代入先の型に自動変換される．
5. $\sqrt{x}$ には `sqrt(x)` という関数が用意されている．

よく利用する数学関数を表 2.3 に示す．

演習 2.11 実数 x を入力して

$$y = \left(1 + \frac{1}{x}\right)^x \tag{2.7}$$

を計算して出力するプログラムを作れ．この関数は $y \to e$ $(x \to \infty)$ となる．x として大きな数を与えて，このことを確かめよ．

演習 2.12 整数 n（`int` 型とする）を入力して，

$$y = \left(1 + \frac{1}{n}\right)^n \tag{2.8}$$

を計算して出力するプログラムを作れ．n は整数として与えるが，y は実数として計算することに注意する．この場合

表 2.3: 数学関数

関数	意味
`sin(x)`	$\sin(x)$
`cos(x)`	$\cos(x)$
`tan(x)`	$\tan(x)$
`asin(x)`	$\arcsin(x)$
`acos(x)`	$\arccos(x)$
`atan(x)`	$\arctan(x)$
`atan2(y,x)`	4 象限逆正接
`sinh(x)`	$\sinh(x)$
`cosh(x)`	$\cosh(x)$
`tanh(x)`	$\tanh(x)$
`exp(x)`	$\exp(x)$
`log(x)`	$\ln(x)$
`log10(x)`	$\log_{10}(x)$
`sqrt(x)`	$\sqrt{x}$
`pow(x,a)`	べき乗 x^a
`abs(i)`	整数の絶対値
`fabs(x)`	実数の絶対値
`cabs(x)`	複素数の絶対値
`floor(x)`	床（x 以下の最大の整数）
`ceil(x)`	天井（x 以上の最小の整数）
`rint(x)`	x に最も近い整数

```
1 + 1/n
```

としないで

```
1.0 + 1.0/n
```

とすれば，`double` 型として計算される（型が異なる演算の場合，上位の型へ自動的に変換される）．また，`1.0+1/n` を試してみよ．

演習 2.13 多項式の計算は入れ子演算にすると掛け算回数が減り計算に有利となる．

$$y = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} = 1 + x\left(1 + x\left(\frac{1}{2} + x\left(\frac{1}{6} + \frac{x}{24}\right)\right)\right) \tag{2.9}$$

この多項式を計算して出力するプログラムを作れ．また，$x = 1, 2, 3$ に対する y の値を求めよ．

（答え）$y(1) = 2.708333,\quad y(2) = 7,\quad y(3) = 16.375$

演習 2.14 実数 x を入力して次の関数値を計算するプログラムを作成せよ.

$$y = \sqrt{\frac{x^2+1}{2x^4+\sqrt{2}|x|+1}} \tag{2.10}$$

また，$x = -2, 1, 2$ に対する y の値を求めよ.
（答え）$y(-2) = 0.373569,\quad y(1) = 0.673114,\quad y(2) = 0.373569$

演習 2.15 二つの整数 n, m（int 型とする）を入力して

$$y = \frac{n}{m} \tag{2.11}$$

を計算するプログラムを作れ．ただし，double 型の計算を行うため

```
y = (double)n/m;
```

とせよ.（型名を ‘( )’ で囲んで変数の前に付ければ型を強制的に変更できる．この構文を**キャスト**という．この場合，n を double 型に変換している.）また

```
y = n/m;
```

```
y = n/(double)m;
```

```
y = (double)n/(double)m;
```

をそれぞれ試してみよ.

2.5　if 文

if 文は，条件により処理の流れを変える構文である.

例題 2.1 整数 a を入力し，$a > 0$ ならば positive，$a \leq 0$ ならば nonpositive を出力するプログラムを作れ.
解答例

```
#include <stdio.h>
main()
{
   int a;
   printf("Enter a\n");
   scanf("%d",&a);

   if (a > 0)
      printf("positive\n");
   else
      printf("nonpositive\n");
}
```

実行結果

```
Enter a
1
positive
```

例題 2.1 の説明

1. if 文は

   ```
   if（条件式）
       文 1；
   else
       文 2;
   ```

   の形で使う．条件式が成立する場合，文 1 を実行し，成立しない場合，文 2 を実行する．

2. 文 2 がない場合，else 以下を省略できる．

3. 文が複数ある場合，‘{ }’ で囲み，一つの**ブロック**とする．

例題 2.2 整数 a を入力し，$a > 0$ ならば positive，$a = 0$ ならば zero，$a < 0$ ならば negative を出力するプログラムを作れ．

解答例

```
#include <stdio.h>
main()
{
   int a;
   printf("Enter a\n");
   scanf("%d",&a);

   if (a > 0)
      printf("positive\n");
   else
      if (a == 0)
         printf("zero\n");
      else
         printf("negative\n");
}
```

例題 2.2 の説明

1. if 文は何重にも入れ子にできる．解答例のように，プログラムをインデント（字下げ）すれば，各 if 文の適用範囲を明示できる．

2. 条件式で等しいことを判定するには ‘==’ を使うことに注意する（‘=’ では単なる代入になってしまう）.

条件式で使われる**関係演算子**を表 2.4 に示す.

表 2.4: 関係演算子

演算子	意味
<	$<$
<=	$\leq$
>	$>$
>=	$\geq$
==	$=$
!=	$\neq$

例題 2.3 整数 a を入力し，$0 \leq a \leq 10$ ならば in，そうでなければ out を出力するプログラムを作れ.

解答例

```
#include <stdio.h>
main()
{
   int a;
   printf("Enter a\n");
   scanf("%d",&a);

   if (a >= 0 && a <= 10)
      printf("in\n");
   else
      printf("out\n");
}
```

実行結果

```
Enter a
1
in
```

例題 2.3 の説明

1. ‘&&’ は AND 演算子（かつ）である.

表 2.5: 論理演算子

演算子	意味
&&	AND
||	OR
!	NOT

論理演算子を表 2.5 に示す．これらを関係演算子と組み合わせて使うと複雑な if 文や条件式をコンパクトに書くことができる．

例題 2.4 実数 x を入力し，x の**符号関数**（図 2.1 参照）

$$y = \mathrm{sgn}\,(x) := \begin{cases} 1 & (x > 0) \\ 0 & (x = 0) \\ -1 & (x < 0) \end{cases} \tag{2.12}$$

の値を出力するプログラムを作成せよ．

解答例

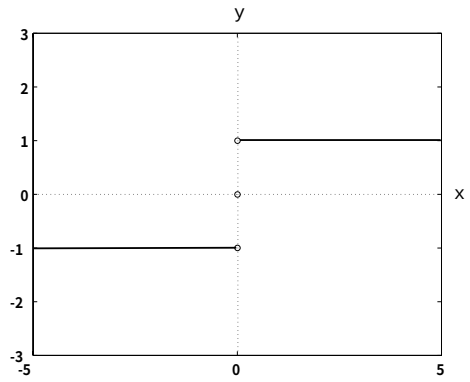

図 2.1: $y = \mathrm{sgn}\,(x)$ のグラフ

```
#include <stdio.h>
main()
{
   double x,y;
   printf("Enter x\n");
   scanf("%lf",&x);

   if (x > 0.0)
```

```
        y = 1.0;
    else if (x == 0.0)
            y = 0.0;
         else
            y = -1.0;

    printf("y = %f\n",y);
}
```

実行結果

```
Enter x
2
y = 1.000000
```

演習 2.16 実数 x を入力し，x の飽和関数（図 2.2 参照）

$$y = \mathrm{sat}\,(x) := \begin{cases} 1 & (x > 1) \\ x & (|x| \leq 1) \\ -1 & (x < -1) \end{cases} \tag{2.13}$$

の値を出力するプログラムを作成せよ．

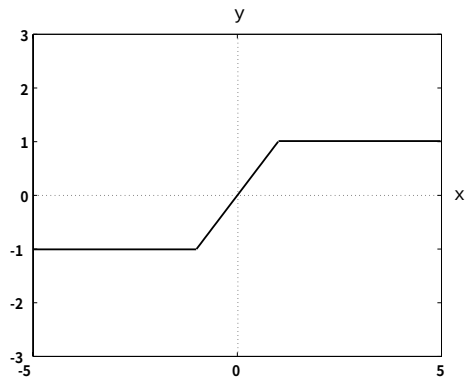

図 2.2: $y = \mathrm{sat}\,(x)$ のグラフ

演習 2.17 実数 x を入力し，次の関数（図 2.3 参照）

$$y = \begin{cases} (\sin x)/x & (x \neq 0) \\ 1 & (x = 0) \end{cases} \tag{2.14}$$

の値を出力するプログラムを作成せよ．グラフから推察できるように，次式が成り立つ．

$$\lim_{x \to 0} \frac{\sin x}{x} = 1 \tag{2.15}$$

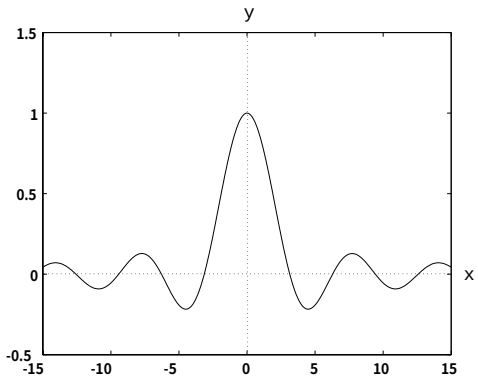

図 2.3: $y = \sin x/x$ のグラフ

演習 2.18 入力された整数 n（$0 \leq n \leq 10$）が素数（1, 2, 3, 5, 7）ならば

```
a prime number
```

素数でなければ

```
not a prime number
```

と表示するプログラムを作成せよ.

演習 2.19 集合 S_1, S_2 を次式で定義する.

$$S_1 := \{x : -3 \leq x \leq 0\} \quad S_2 := \{x : 5 \leq x \leq 15\} \tag{2.16}$$

入力された実数 x について $x \in S_1 \cup S_2$ ならば

```
a member
```

$x \notin S_1 \cup S_2$ ならば

```
not a member
```

と表示するプログラムを作成せよ.

演習 2.20 入力された正整数 a が偶数ならば

```
even
```

奇数ならば

```
odd
```

を表示するプログラムを作成せよ.

演習 2.21 入力された実数 x を四捨五入して整数とするプログラムを作成せよ.

演習 2.22 $a_{max} = 10$ とする．整数 a を入力し，$a > a_{max}$ ならば

$a_{max} \leftarrow a$ （a_{max}に a の値を代入する）

とし，そうでなければ a_{max} をそのままとするプログラムを作成せよ．

例題 2.5 関数 $f(x)$ を

$$f(x) := x(x^2 - 25) \tag{2.17}$$

と定義する（図 2.4 参照）．実数 x_1, x_2 を入力し

$$y_1 = f(x_1), \quad y_2 = f(x_2) \tag{2.18}$$

を出力するプログラムを作れ．

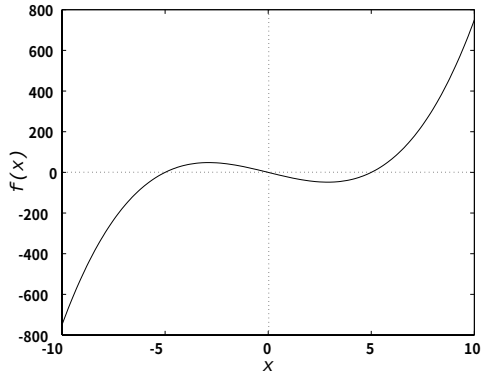

図 2.4: $y = x(x^2 - 25)$ のグラフ

解答例

```
#include <stdio.h>
#define f(x) x*(x*x-25.0)
main()
{
   double x1,x2,y1,y2;
   printf("Enter x1 x2\n");
   scanf("%lf %lf",&x1,&x2);
   y1 = f(x1);
   y2 = f(x2);
   printf("y1 = %f  y2 = %f\n",y1,y2);
}
```

実行例

```
Enter x1 x2
1 6
y1 = -24.000000  y2 = 66.000000
```

例題 2.5 の説明

1. `#define` 文により指定した関数名で関数を定義できる．次のように変数は複数個置くこともできる．

```
#define Kyori(x,y) sqrt(x*x+y*y)
```

演習 2.23 式 (2.17) で定義される $f(x)$ に対して

$$f(x_1)f(x_2) < 0 \tag{2.19}$$

をみたす x_1, x_2 が与えられたとし，これらの中点を x_c とするとき

$$S_1 = [x_1,\, x_c], \quad S_2 = [x_c,\, x_2] \tag{2.20}$$

のどちらで $f(x)$ の符号が変わるかを判定するプログラムを作成せよ．

演習 2.24 図 2.5 において，A 点の $(x,\, y)$ 座標が与えられたとき，x 軸から測った OA の角度 θ は

```
atan2(y,x)
```

で求めることができる．ただし，θ の範囲は

$$-\pi < \theta \leq \pi \tag{2.21}$$

である．`atan2` 関数を利用して θ を

$$0 \leq \theta < 2\pi \tag{2.22}$$

の範囲で表示するプログラムを作成せよ．

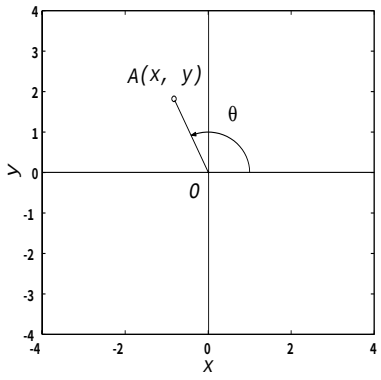

図 2.5: A 点の角度 θ

演習 2.25 実数 x を入力して $x \geq 0$ ならば

$\sqrt{x}$

を，$x < 0$ ならば

$\sqrt{-x}$ j

を出力するプログラムを作成せよ．

演習 2.26 2 次方程式

$$ax^2 + bx + c = 0 \tag{2.23}$$

の解の公式は

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{2.24}$$

である．式 (2.24) を用いて 2 次方程式の解を計算するプログラムを作成せよ．

2.6 for 文

for 文は繰り返し回数がわかっている場合に適した反復構文である．次の例で for 文の使い方を説明する．

例 2.5

```
#include <stdio.h>
main()
{
   int i;
   for (i = 0; i <= 5; ++i)
      printf("i = %d\n",i);
}
```

実行結果

```
i = 0
i = 1
i = 2
i = 3
i = 4
i = 5
```

例 2.5 の説明

1. 上記 for 文の意味は次のとおりである．$i = 0$ と初期化し，$i \leq 5$ が成立するまで

   ```
   printf("i = %d\n",i);
   ```

   を繰り返し実行せよ．そして，一回の実行が終わるたびに i を 1 ずつ増やせ．‘++’ は**インクリメント演算子**で++i は i = i + 1 と書いてもよい．

2. 繰り返しの対象となる文が複数ある場合，それらを ‘{ }’ で囲みブロックとする．

例題を通して，繰り返しの基本的プログラミングを紹介しよう．

例題 2.6 n を入力し，$1 \sim n$ の総和

$$\sum_{i=1}^{n} i = 1 + 2 + 3 + \cdots + n \tag{2.25}$$

を計算するプログラムを作成せよ．

解答例

```
#include <stdio.h>
main()
{
   int i,n,s;
   printf("Enter n\n");
   scanf("%d",&n);
   s = 0;
   for (i = 1; i <= n; ++i)
      s = s+i;
   printf("s = %d\n",s);
}
```

実行例

```
Enter n
10
s = 55
```

例題 2.7 正整数 n を入力し，n の**階乗**

$$n! = 1 \cdot 2 \cdot 3 \cdots n \tag{2.26}$$

を計算するプログラムを作成せよ．

解答例

```
#include <stdio.h>
main()
{
   int i,n,y;
   printf("Enter n\n");
   scanf("%d",&n);
   y = 1;
   for (i = 1; i <= n; ++i)
      y = y*i;
   printf("y = %d\n",y);
}
```

実行例

```
Enter n
5
y = 120
```

$n!$ は n を増やすと急速に大きくなることに注意する．int 型で表現できる範囲は，使用した環境の場合

$$-2147483647 \sim 2147483647 \tag{2.27}$$

であり，$n = 12$ までしか正確に計算できない．

演習 2.27 正整数 n を入力し，$i!$, $i = 1 \sim n$ を出力するプログラムを作成せよ．

例題 2.8 正整数 n と実数 x を入力し，多項式

$$\sum_{i=0}^{n} x^i = 1 + x + x^2 + \cdots + x^n \tag{2.28}$$

を計算するプログラムを作成せよ．

解答例

```
#include <stdio.h>
main()
{
   int i,n;
   double x,y,z;
   printf("Enter n x\n");
   scanf("%d %lf",&n,&x);
   y = 1.0;
   z = 1.0;
```

```
   for (i = 1; i <= n; ++i){
      z = z*x;
      y = y+z;
   }
   printf("y = %f\n",y);
}
```

実行例

```
Enter n x
10 0.5
y = 1.999023
```

例題 2.8 の説明

1. z = 1.0 と初期化し，for 文で z = z*x を反復することで x^i を計算していることに注意する．こちらの方が pow 関数を使う方法に比べて計算効率が良い．

例題 2.9 正整数 n と実数 x を入力し，多項式

$$\sum_{i=0}^{n} \frac{x^i}{i!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!} \tag{2.29}$$

を計算するプログラムを作成せよ．

解答例

```
#include <stdio.h>
main()
{
   int i,n;
   double x,y,z;
   printf("Enter n x\n");
   scanf("%d %lf",&n,&x);
   y = 1.0;
   z = 1.0;
   for (i = 1; i <= n; ++i){
      z = z*x/i;
      y = y+z;
   }
   printf("y = %f\n",y);
}
```

実行例

```
Enter n x
10 1
y = 2.718282
```

例題 2.9 の説明

1. `z = 1.0` と初期化し，`for` 文で `z = z*x/i` を反復することで $x^i/i!$ を計算していることに注意する．$i!$ は急速に大きくなるので直接計算せずに，このような反復により計算すれば実用的なプログラムとなる．

演習 2.28 奇数 n と実数 x を入力し，多項式

$$y = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + - \cdots (-1)^{(n-1)/2} \frac{x^n}{n!} \tag{2.30}$$

を計算するプログラムを作成せよ．

演習 2.29 偶数 n と実数 x を入力し，多項式

$$y = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + - \cdots (-1)^{n/2} \frac{x^n}{n!} \tag{2.31}$$

を計算するプログラムを作成せよ．

演習 2.30 整数

$$a_1, a_2, \ldots, a_n$$

を入力し，$\mathrm{sgn}\,(a_i)$ の総和

$$\sum_{i=1}^{n} \mathrm{sgn}\,(a_i) \tag{2.32}$$

を計算するプログラムを作れ．

例題 2.10 $0 \sim 10$ の範囲の整数

$$a_1, a_2, \ldots, a_n$$

を入力し，$a_i < 10$ ならば a_i を加算し，$a_i = 10$ ならば $a_{i-1} + a_i$ を加算して合計を計算するプログラムを作成せよ．

解答例

```
#include <stdio.h>
main()
{
   int i,n,s,a,a1;
```

```
    s = 0;
    a = 0;
    printf("Enter n\n");
    scanf("%d",&n);
    for (i = 1; i <= n; ++i){
       printf("a%d = ",i);
       a1 = a;
       scanf("%d",&a);
       if (a == 10)
          s = s+a+a1;
       else
          s = s+a;
       }
    printf("s = %d\n",s);
}
```

実行例

```
Enter n
5
a1 = 10
a2 = 10
a3 = 2
a4 = 3
a5 = 10
s = 48
```

例題 2.10 の説明

1. a = 0 と初期化し，scanf で a を更新する前に a1 = a とおき，a1 に a の更新前の値を代入している．

演習 2.31 $0 \sim 10$ の範囲の整数

$$a_1, a_2, \ldots, a_n$$

を入力し，$a_i < 10$ ならば a_i を加算し，$a_i = 10$ ならば $a_{i-2} + a_{i-1} + a_i$ を加算して合計を計算するプログラムを作成せよ．

例題 2.11 関数

$$f(x) = -x^4 + 20x^2 - 20x - 15 \tag{2.33}$$

の値を

$$-5 \leq x \leq 5 \tag{2.34}$$

の範囲で求めるプログラムを作れ．ただし，x の刻み幅は 0.1 とする．また，結果をグラフ表示せよ．

解答例

```
#include <stdio.h>
#include <math.h>
#define f(x) -pow(x,4)+20.0*x*x-20.0*x-15.0
main()
{
   double x,dx;
   dx = 0.1;
   for (x = -5.0; x <= 5.0; x = x+dx)
      printf("%f  %f\n",x,f(x));
}
```

実行結果

```
-5.000000  -40.000000
-4.900000  -13.280100
-4.800000  10.958400
-4.700000  32.831900
-4.600000  52.454400
-4.500000  69.937500
    .....
```

グラフ表示

```
./a.out > data1
gnuplot
set xlabel 'x'
set ylabel 'f(x)'
set nokey
set grid
plot 'data1' with lines
```

例題 2.11 の説明

1. `#define` 文で $f(x)$ を定義し，`for` 文で x を変えながら $f(x)$ を計算する．
2. **リダイレクション機能** '>' を利用して，結果をファイル `data1` に出力し，gnuplot というツールを用いてグラフを作成する．
3. `set xlabel 'x'`：x 軸の**ラベル**を `x` とする．
4. `set ylabel 'f(x)'`：y 軸のラベルを `f(x)` とする．

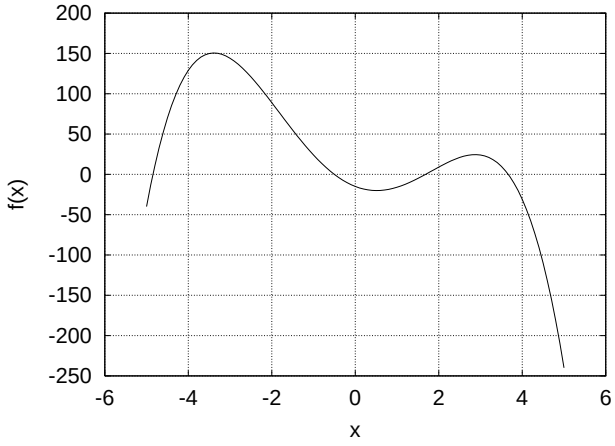

図 2.6: $f(x) = -x^4 + 20x^2 - 20x - 15$ のグラフ

5. `set nokey`：**凡例**を省く．
6. `set grid`：グラフに**格子**を表示する．
7. `plot 'data1' with lines`：ファイル `data1` のデータを**グラフ表示**する．データ点は直線で結ぶ．

演習 2.32 関数

$$f(x) = \frac{\sin x}{x} \tag{2.35}$$

の値を

$$-15 \leq x \leq 15 \tag{2.36}$$

の範囲で求めるプログラムを作れ．ただし，x の刻み幅は 0.1 とする．また，結果をグラフ表示せよ．

例題 2.12 関数

$$f(x) = -x^4 + 20x^2 - 20x - 15 \tag{2.37}$$

の区間

$$-5 \leq x \leq 5 \tag{2.38}$$

における**最大値**を求めるプログラムを作れ．ただし，x の探索幅を 0.01 とせよ．

解答例

```
#include <stdio.h>
#include <math.h>
#define f(x) -pow(x,4)+20.0*x*x-20.0*x-15.0
main()
{
   double x,y,dx,ymax;
   dx = 0.01;
   ymax = f(-5.0);
   for (x = -5.0; x <= 5.0; x = x+dx){
      y = f(x);
      if (ymax < y)
         ymax = y;
    }
    printf("ymax = %f\n",ymax);
}
```

実行結果

```
ymax = 150.573638
```

例題 2.12 の説明

1. `ymax = f(-5.0)` により `ymax` に初期値を与える．
2. `ymax` と `y` を比較して，`y` の方が大きい場合，`ymax` に `y` を代入する．
3. 探索幅 `dx` を小さくすれば，求解精度が上がる．

演習 2.33 関数

$$f(x) = x \sin\left(\frac{1}{x}\right) \tag{2.39}$$

の区間

$$-0.5 \leq x \leq 0.5 \tag{2.40}$$

における最小値を求めるプログラムを作れ．ただし，x の探索幅を 0.001 とせよ．
（答）-0.217225

演習 2.34 関数

$$f(x) = -x^4 + 20x^2 - 20x - 15 \tag{2.41}$$

の区間

$$-5 \leq x \leq 5 \tag{2.42}$$

における実根の数を求めるプログラムを作れ．ただし，x の探索幅を 0.01 とせよ．
ヒント：関数の符号の変化回数を調べよ．
（答）4

2.7 while文

while 文は与えられた条件が成立する間，繰り返しを行うという制御文である．終了基準は示されているが，繰り返し回数が定かでないという場合に適している．

例題 2.13 級数

$$\sum_{n=1}^{\infty} \frac{1}{2^n} = \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots \tag{2.43}$$

の値を計算するプログラムを作れ．ただし，級数の計算は，新たに加える項の絶対値が正数 ϵ より小さくまで行うものとする．$\epsilon = 10^{-6}$ とする．

解答例

```
#include <stdio.h>
#include <math.h>
main()
{
   double x,y,z,eps;
   eps = 1e-6;
   x = 0.5;
   y = 0.0;
   z = 1.0;
   while (fabs(z) > eps){
      z = z*x;
      y = y+z;
   }
   printf("y = %f\n",y);
}
```

実行結果

```
y = 0.999999
```

例題 2.13 の説明

1. while（継続条件式){ ...} の構文で使う．
2. $x = 0.5$ とし，x のべき乗を z で求め，y に加算していくというプログラムである．

演習 2.35 実数 x を入力し，級数

$$\sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots \tag{2.44}$$

の値を計算するプログラムを作れ．ただし，級数の計算は，新たに加える項の絶対値が正数 ϵ より小さくまで行うものとする．$\epsilon = 10^{-6}$ とする．

例題 2.14 実数 $x \geq 0$ が入力されるまで入力を繰り返し

$$\sqrt{x} \tag{2.45}$$

を計算するプログラムを作れ.

解答例

```
#include <stdio.h>
#include <math.h>
main()
{
   double x;
   x = -1.0;
   while (x < 0.0){
      printf("Enter x\n");
      scanf("%lf",&x);
   }
   printf("y = %f\n",sqrt(x));
}
```

実行例

```
Enter x
-1
Enter x
2
y = 1.414214
```

例題 2.14 の説明

1. `x = -1.0` と初期化して，`while` 文に入る.

演習 2.36 $|x| < 1$ をみたす実数 x が入力されるまで入力を繰り返し，級数

$$\sum_{n=0}^{\infty} x^n = 1 + x + x^2 + x^3 + \cdots \tag{2.46}$$

の値を計算するプログラムを作れ．ただし，級数の計算は，新たに加える項の絶対値が正数 ϵ より小さくまで行うものとする．$\epsilon = 10^{-6}$ とする．また，この級数は

$$\frac{1}{1-x} \tag{2.47}$$

に収束することを確認せよ.

2.8 do-while 文

do-while 文は while 文とほぼ同じ働きをする．構文は

```
do
   文
while (継続条件式);
```

であり，文を実行した後で継続条件式が判定される．よって，文は，少なくとも一度は実行されることになる．このようなループが while 文よりも便利なことがある．

例題 2.15 実数 $x \geq 0$ が入力されるまで入力を繰り返し

$$\sqrt{x} \tag{2.48}$$

を計算するプログラムを作れ．

解答例

```
#include <stdio.h>
#include <math.h>
main()
{
   double x;
   do {
      printf("Enter x\n");
      scanf("%lf",&x);
   }
   while (x < 0.0);
   printf("y = %f\n",sqrt(x));
}
```

例題 2.15 の説明

1. 例題 2.14 を do-while 文を用いてプログラムしたものである．x の初期化が不要となる．

第3章　非線形方程式の解法

3.1　ニュートン法

非線形方程式

$$f(x) = 0 \tag{3.1}$$

の解を数値的に求める方法にニュートン法がある．

ニュートン法　$f(x)$ は微分可能とする．x の初期値 x_0 が与えられるとき，次式を満足のいくまで繰り返せ．

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \tag{3.2}$$

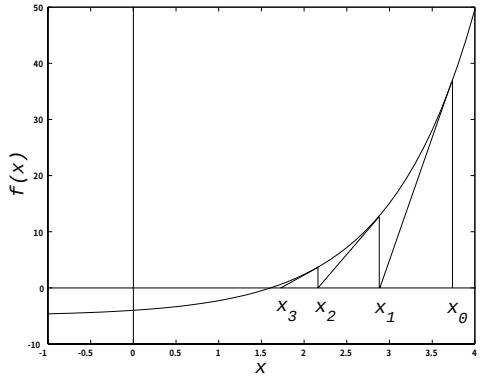

図 3.1: ニュートン法の原理

式 (3.2) は次のように導ける．図 3.1 から，勾配 $f'(x_0)$ は

$$f'(x_0) = \frac{f(x_0)}{x_0 - x_1} \tag{3.3}$$

であるから，これを整理すると

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} \tag{3.4}$$

となり，これを一般化して式 (3.2) を得る．

例題 3.1 方程式

$$f(x) = e^x - 5 = 0 \quad \text{厳密解：} x = \ln 5 = 1.60943791243410 \tag{3.5}$$

の解をニュートン法で求めるプログラムを作成せよ．ただし

$$x_0 = 4 \tag{3.6}$$

とし，終了基準は

$$|f(x)| < \epsilon, \quad \epsilon = 10^{-8} \tag{3.7}$$

とせよ．

解答例

```
#include <stdio.h>
#include <math.h>
#define f(x) (exp(x)-5.0)
#define df(x) exp(x)
main()
{
   int i=0;
   double x,x1=4.0,eps=1e-8;
   do {
      ++i;
      x = x1;
      x1 = x - f(x)/df(x);
   } while (fabs(f(x1)) > eps);
   x = x1;
   printf("i = %d  x = %f  f(x) = %f\n",i,x,f(x));
}
```

実行結果

```
i = 7  x = 1.609438  f(x) = 0.000000
```

例題 3.1 の説明

1. `#define` 文による関数の定義にしばしば‘( )’が必要であることに注意する．コンパイル時に 1 番目の文字列が 2 番目の文字列に置き換えられるだけなので，例えば

   ```
   #define f(x) exp(x)-5.0
   ```

   と定義すると

```
x1 = x - f(x)/df(x);
```

は

```
x1 = x - exp(x)-5.0/exp(x);
```

となり，意図する演算が行われない．

2. 変数の定義時に初期値を設定することができる．

3. `++i` でニュートン法の繰り返し回数をカウントする．

演習 3.1 代数方程式

$$x^3 - x^2 - x + 1.00001 = 0 \tag{3.8}$$

の実根をニュートン法で求めるプログラムを作れ．繰り返し回数をカウントするようにせよ．また，終了基準の ϵ は

$$\epsilon = 10^{-10}$$

とせよ．x_0 は入力できるようにし，種々の x_0 を与えてみよ．
（答）$x = -1.000002$

3.2 ブレント法

ニュートン法は二次収束が保証されている効率的な反復法であるが，次のような欠点がある．

1. 収束判定が適切でないと反復が止まらなくなる．

2. 初期値が適切でないと収束しないことがある．

3. $f'(x)$ の計算が必要である．

4. 重根の場合には収束が遅くなる．

特に，初期値の選定は難しく，根を含むある区間内に x_0 をとらないと，収束が期待できない．したがって，ニュートン法を使う場合，あらかじめ二分法のような大域的に収束する方法を用いて，粗い解を求めておく必要がある．また，$f'(x)$ が簡単に求められない場合もある．$f'(x)$ の計算が困難な場合，次のはさみうち法が利用できる．

はさみうち法　x の初期値 x_0 が与えられるとき，次式を満足のいくまで繰り返せ．

$$x_{i+1} = x_i - \frac{f(x_i)}{\dfrac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}} \tag{3.9}$$

二分法のアルゴリズムは次のようである.

二分法　区間 $[a_0, b_0]$ に対して，$f(a_0)f(b_0) \leq 0$ とするとき，$i = 0, 1, 2, \ldots$ について以下を満足のいくまで繰り返せ.

$$c = \frac{a_i + b_i}{2} \tag{3.10}$$

とし，もし $f(a_i)f(c) \leq 0$ ならば

$$a_{i+1} = a_i, \quad b_{i+1} = c \tag{3.11}$$

そうでなければ

$$a_{i+1} = c, \quad b_{i+1} = b_i \tag{3.12}$$

とせよ．そのとき，$f(x)$ は区間 $[a_{i+1}, b_{i+1}]$ に根を持つ.

二分法の確実さと，はさみうち法の収束の早さとを結びつけた解法を**ブレント法**という.

例題 3.2 方程式

$$f(x) = 2e^{-0.1x}\cos 4x - 3e^{-0.3x} = 0 \tag{3.13}$$

の解は図 3.2 に示すように無数に存在するが，小さい方から n 個解を求めるプログラムを作成せよ.

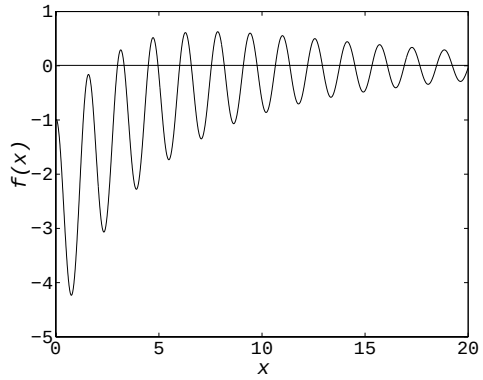

図 3.2: $f(x) = 2e^{-0.1x}\cos 4x - 3e^{-0.3x}$ のグラフ

解答例

```
#include <stdio.h>
#include <math.h>
#define f(x) (2.0*exp(-0.1*x)*cos(4*x)-3.0*exp(-0.3*x))
```

```
main()
{
   int i=0,n;
   double x,dx,x1,zx,eps=1e-10;
   printf("Enter x dx\n");
   scanf("%lf %lf",&x,&dx);
   printf("Enter n\n");
   scanf("%d",&n);

   for (i = 1; i <= n; ++i){
      do {
         zx = x;
         x = x+dx;
      } while (f(x)*f(zx) > 0);

      do {
         x1 = x - f(x)/(f(x)-f(zx))*(x-zx);
         zx = x;
         x = x1;
      } while (fabs(f(x)) > eps);

      printf("i = %d  x = %f  f(x) = %f\n",i,x,f(x));
      x = x+dx;
   }
}
```

実行結果

```
Enter x dx
0 0.2
Enter n
5
i = 1  x = 2.991291  f(x) = 0.000000
i = 2  x = 3.313318  f(x) = 0.000000
i = 3  x = 4.484266  f(x) = -0.000000
i = 4  x = 4.957539  f(x) = 0.000000
i = 5  x = 6.007490  f(x) = -0.000000
```

例題 3.2 の説明

1. x, dx を入力して，初期値 x_0 と探索の刻み幅を与える．

2. 1 番目の `do-while` 文で初期値から `dx` 刻みで関数が符号変化する区間を探索する．`dx` を小さく与えると想定しているので，二分法は使用していない．

3. 2 番目の `do-while` 文で，解が存在すると判明した区間にて，はさみうち法により解を精度良く求める．

演習 3.2 例題 3.2 のプログラムを変更して，各々の解に対するはさみうち法の繰り返し回数も表示するプログラムを作れ．

演習 3.3 方程式

$$f(x) = x \sin\left(\frac{1}{x}\right) + 0.1 = 0 \tag{3.14}$$

の解は図 3.3 に示すように 4 つ存在する．これらをすべて求めよ．$f(x)$ は次のように定義できる．

```
#define f(x) (x ? x*sin(1/x)+0.1 : 0.1)
```

（答）± 0.285791, ± 0.176081

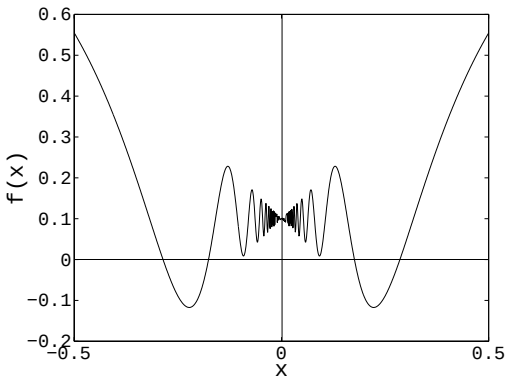

図 3.3: $f(x) = x \sin(1/x) + 0.1$ のグラフ

演習 3.4　演習 3.2 のプログラムを用いて，演習 3.1 の問題を解いてみよ．

第4章　数値積分

4.1　台形則

関数 $f(x)$ の積分

$$S = \int_a^b f(x)dx \tag{4.1}$$

を計算する一つの簡単な方法は，区間 $[x_i, x_{i+1}]$ の台形面積

$$s_i = \frac{1}{2}(f(x_i) + f(x_{i+1}))(x_{i+1} - x_i) \tag{4.2}$$

を加算することである（図 4.1 参照）．この方法は**台形則**といい，アルゴリズムが理解しやすい方法であるが，区間の幅を十分小さくとならければ，高精度な数値積分ができないという欠点を持つ．

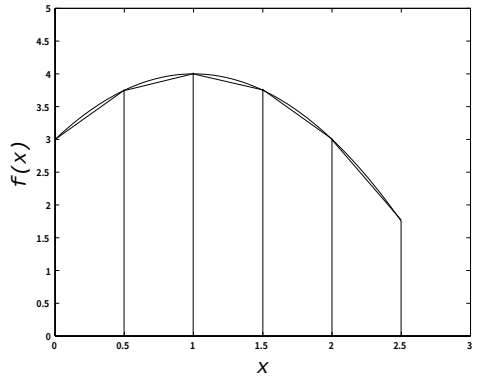

図 4.1: 台形則による積分計算

$[a, b]$ が有界で，$[a, b]$ を n 等分するとき

$$h = \frac{b-a}{n} \tag{4.3}$$

$$x_0 = a, \quad x_{i+1} = x_i + h, \quad i = 1 \sim n-1 \tag{4.4}$$

$$y_i = f(x_i), \quad i = 1 \sim n \tag{4.5}$$

とすると

$$\begin{aligned} S &\simeq \frac{h}{2}(y_0+y_1)+\frac{h}{2}(y_1+y_2)+\cdots+\frac{h}{2}(y_{n-1}+y_n) \\ &= \frac{h}{2}y_0+h\sum_{i=1}^{n-1}y_i+\frac{h}{2}y_n \\ &= h\left\{\frac{y_0+y_n}{2}+\sum_{i=1}^{n-1}y_i\right\} \end{aligned} \tag{4.6}$$

を得る.

例題 4.1

$$S=\int_0^1 e^{-x}dx \qquad \text{厳密解：} S=1-e^{-1}=0.63212055882856 \tag{4.7}$$

を台形則により数値積分せよ（図 4.2）.

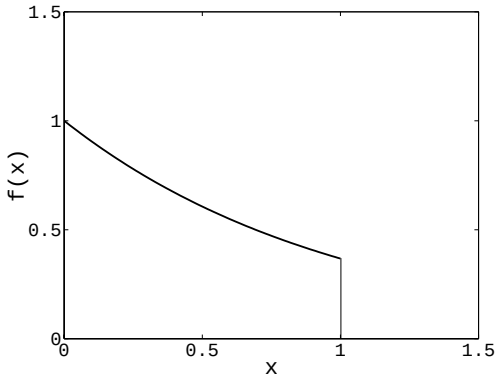

図 4.2: 例題 4.1 の説明

解答例

```
#include <stdio.h>
#include <math.h>
#define f(x) exp(-x)
main()
{
   int i=0,n;
   double x=0.0,h,y1,y2,s=0.0;
   printf("Enter n\n");
   scanf("%d",&n);
   h = 1.0/n;
```

```
    y1 = f(x);
    for (i = 1; i <= n; ++i){
        x = x+h;
        y2 = y1;
        y1 = f(x);
        s = s + 0.5*(y1+y2)*h;
    }
    printf("s = %f\n",s);
}
```

実行結果

```
Enter n
20
s = 0.632252
```

例題 4.1 の説明

1. n は積分区間 $[0, 1]$ の分割数である.
2. 式 (4.2) を用いた.

演習 4.1

$$S = \int_1^2 \frac{1}{x} dx \quad \text{厳密解：} S = \ln 2 = 0.69314718055995 \tag{4.8}$$

を台形則により数値積分せよ（図 4.3）.

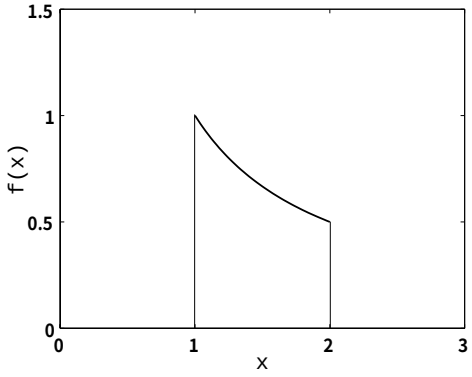

図 4.3: 演習 4.1 の説明

演習 4.2

$$S = \int_0^\infty e^{-x} dx \quad \text{厳密解：} S = 1 \tag{4.9}$$

を台形則により数値積分せよ.

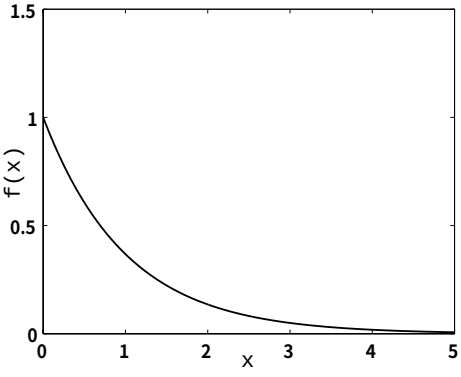

図 4.4: 演習 4.2 の説明

演習 4.3 例題 4.1 で扱った数値積分を式 (4.6) により行うプログラムを作れ.

4.2　シンプソン則

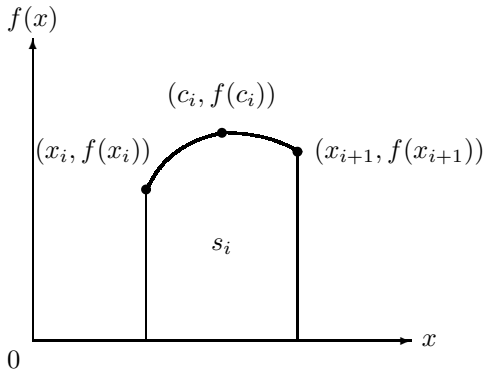

図 4.5: シンプソン則による積分計算

シンプソン則は，関数 $f(x)$ の積分

$$S = \int_a^b f(x)dx \tag{4.10}$$

を計算する際に，区間 $[x_i,\, x_{i+1}]$ の中点を

$$c_i = \frac{x_i + x_{i+1}}{2} \tag{4.11}$$

とするとき，3 点

$$(x_i,\, f(x_i)),\quad (c_i,\, f(c_i)),\quad (x_{i+1},\, f(x_{i+1})) \tag{4.12}$$

を通る 2 次関数 $f_1(x)$ の積分

$$s_i = \int_{x_i}^{x_{i+1}} f_1(x)dx \tag{4.13}$$

を加算することである（図 4.5）．この方法によれは，区間の幅を台形則ほど小さくしないで高精度な結果が得られる．また，アルゴリズムも簡単である．$f_1(x)$ は次式で与えられる．

$$\begin{aligned} f_1(x) &= \frac{(x-c_i)(x-x_{i+1})}{(x_i-c_i)(x_i-x_{i+1})}f(x_i) + \frac{(x-x_i)(x-x_{i+1})}{(c_i-x_i)(c_i-x_{i+1})}f(c_i) \\ &\quad + \frac{(x-x_i)(x-c_i)}{(x_{i+1}-x_i)(x_{i+1}-c_i)}f(x_{i+1}) \end{aligned} \tag{4.14}$$

また，式 (4.13) の s_i を計算すると次式を得る．

$$s_i = \frac{x_{i+1}-x_i}{6}\{f(x_i) + 4f(c_i) + f(x_{i+1})\} \tag{4.15}$$

$[a, b]$ が有界で，$[a, b]$ を n 等分するとき

$$h = \frac{b-a}{n} \tag{4.16}$$

$$x_0 = a, \quad x_{i+1} = x_i + h, \quad i = 1 \sim n-1 \tag{4.17}$$

$$c_i = \frac{x_i + x_{i+1}}{2}, \quad i = 1 \sim n-1 \tag{4.18}$$

とすると

$$\begin{aligned} S &\simeq \frac{h}{6}\{(f(x_0) + 4f(c_0) + f(x_1)) + (f(x_1) + 4f(c_1) + f(x_2)) + \cdots \\ &\quad + (f(x_{n-1}) + 4f(c_{n-1}) + f(x_n))\} \end{aligned} \tag{4.19}$$

$$= \frac{h}{6}\left\{f(x_0) + 2\sum_{i=1}^{n-1} f(x_i) + 4\sum_{i=0}^{n-1} f(c_i) + f(x_n)\right\} \tag{4.20}$$

を得る．

例題 4.2

$$S = \int_0^1 e^{-x}dx \qquad \text{厳密解：} S = 1 - e^{-1} = 0.63212055882856 \tag{4.21}$$

をシンプソン則により数値積分せよ．

解答例

```
#include <stdio.h>
#include <math.h>
#define f(x) exp(-x)
main()
{
```

```
    int i=0,n;
    double x=0.0,c,h,y1,y2,yc,s=0.0;
    printf("Enter n\n");
    scanf("%d",&n);
    h = 1.0/n;
    y1 = f(x);
    for (i=1; i<=n; ++i){
       x = x+h;
       y2 = y1;
       y1 = f(x);
       c = x-0.5*h;
       yc = f(c);
       s = s + (y1+4.0*yc+y2)*h/6.0;
    }
    printf("s = %f\n",s);
}
```

実行結果

```
Enter n
20
s = 0.632121
```

例題 4.2 の説明

1. n は積分区間 [0, 1] の分割数である.
2. 式 (4.15) を用いた.
3. c は中点 c_i である.

種々の n について，台形則のプログラムとシンプソン則のプログラムの実行結果を比較することにより，シンプソン則の高効率性が理解できる.

演習 4.4

$$S = \int_1^2 \frac{1}{x} dx \quad \text{厳密解：} S = \ln 2 = 0.69314718055995 \tag{4.22}$$

をシンプソン則により数値積分せよ.

演習 4.5

$$S = \int_0^\infty e^{-x} dx \quad \text{厳密解：} S = 1 \tag{4.23}$$

をシンプソン則により数値積分せよ.

演習 4.6 例題 4.2 で扱った数値積分を式 (4.20) により行うプログラムを作れ.

演習 4.7 式 (4.15) を導出せよ.

第5章　ランダムサーチ法

5.1　1変数関数に対するランダムサーチ法

乱数を利用した探索法をランダムサーチ法という．1変数関数 $f(x)$ の最小値を求めるランダムサーチ法のアルゴリズムは次のとおりである．

ランダムサーチ法　x の初期探索中心 c_0 および初期探索幅 a_0，各ステップでの探索回数 n が与えられるとき，$i = 0, 1, 2, \ldots$ について以下を満足のいくまで繰り返せ．

1. 中心 c_i，幅 a_i の乱数 h_i を n 個発生させ，その中で $f(x)$ を最小にするものを x_{p_i} とおく．
2. $$c_{i+1} = x_{p_i}, \quad a_{i+1} = \frac{a_i}{2} \tag{5.1}$$
とする．

すなわち，中心 c_0，幅 a_0 の区間を n 回ランダムに探索して，その内で最良のものを次の探索の中心 c_1 とし，区間の幅を2分の1に縮小する．このような繰り返しを区間の幅があらかじめ指定した幅よりも小さくなるまで行い，最後の探索結果を近似解とするのである．

例題 5.1　1変数関数

$$f(x) = x^4 - 20(x - 0.5)^2 \tag{5.2}$$

を最小にする x_p と**最小値** $y_{min} = f(x_p)$ をランダムサーチ法によって求めよ．
（答）

$$x_p = -3.38761905847542,\ y_{min} = -170.5739147305898$$

$f(x)$ のグラフを図5.1に示す．

解答例

```
#include <stdio.h>
#include <stdlib.h>
```

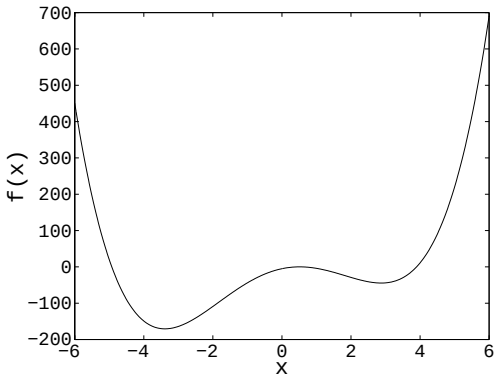

図 5.1: $f(x) = x^4 - 20(x - 0.5)^2$ のグラフ

```
#include <time.h>
#include <math.h>
#define rnd (double)rand()/(double)RAND_MAX
#define f(x) (pow(x,4)-20.0*(x-0.5)*(x-0.5))
main()
{
   int i,n;
   double a,c,x,h,xp;
   double y,ymin,eps=1e-6;
   srand(time(NULL));
   printf("Enter n\n");
   scanf("%d",&n);
   printf("Enter a c\n");
   scanf("%lf %lf",&a,&c);

   ymin = f(c);
   do {
      for (i = 1; i <= n; ++i) {
         h = (rnd-0.5)*2.0*a;
         x = c+h;
         y = f(x);
         if (y < ymin) {
            ymin = y;
            xp = x;
         }
      }
```

```
        printf("a = %f  xp = %f  ymin = %f\n",a,xp,ymin);
        a = a*0.5;
        c = xp;
    } while (a > eps);
}
```

実行結果

```
Enter n
100
Enter a c
5 0
a = 5.000000  xp = -3.379834  ymin = -170.570960
a = 2.500000  xp = -3.380504  ymin = -170.571446
a = 1.250000  xp = -3.380504  ymin = -170.571446
a = 0.625000  xp = -3.390454  ymin = -170.573522
a = 0.312500  xp = -3.390454  ymin = -170.573522
a = 0.156250  xp = -3.389114  ymin = -170.573806
a = 0.078125  xp = -3.389114  ymin = -170.573806
a = 0.039062  xp = -3.387799  ymin = -170.573913
a = 0.019531  xp = -3.387799  ymin = -170.573913
a = 0.009766  xp = -3.387670  ymin = -170.573915
a = 0.004883  xp = -3.387612  ymin = -170.573915
a = 0.002441  xp = -3.387613  ymin = -170.573915
a = 0.001221  xp = -3.387613  ymin = -170.573915
a = 0.000610  xp = -3.387621  ymin = -170.573915
a = 0.000305  xp = -3.387621  ymin = -170.573915
a = 0.000153  xp = -3.387619  ymin = -170.573915
a = 0.000076  xp = -3.387619  ymin = -170.573915
a = 0.000038  xp = -3.387619  ymin = -170.573915
a = 0.000019  xp = -3.387619  ymin = -170.573915
a = 0.000010  xp = -3.387619  ymin = -170.573915
a = 0.000005  xp = -3.387619  ymin = -170.573915
a = 0.000002  xp = -3.387619  ymin = -170.573915
a = 0.000001  xp = -3.387619  ymin = -170.573915
```

例題 5.1 の説明

1. rand は $0 \sim$ RAND_MAX $= 2^{31} - 1$ の範囲の整数をランダムに発生させる関数である．time で時刻（秒）を呼び出し，srand でこの時刻を乱数の種として与えることで，実行する度に乱数の系列を変えている．

演習 5.1

$$f(x) = x \sin\left(\frac{1}{x}\right) + 0.1 \tag{5.3}$$

を最小にする x_p と最小値 $y_{min} = f(x_p)$ をランダムサーチ法によって求めよ（図 3.3 参照）.
（答） $x_p = \pm 0.222548, \quad y_{min} = -0.117234$

5.2　2 変数関数に対するランダムサーチ法

例題 5.2　2 変数関数

$$f(x_1, x_2) = x_1^2 - x_1 x_2 + 2x_2^2 + x_1 - x_2 \tag{5.4}$$

を最小にする (x_{p_1}, x_{p_2}) と最小値 $y_{min} = f(x_{p_1}, x_{p_2})$ をランダムサーチ法によって求めよ.
(答) $x_{p_1} = -0.42857142857143$, $x_{p_2} = 0.14285714285714$, $y_{min} = -0.28571428571429$
参考のため，$f(x_1, x_2)$ の等高線を図 5.2 に示す.

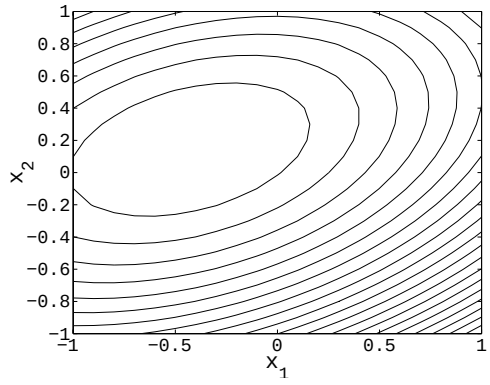

図 5.2: $f(x_1, x_2) = x_1^2 - x_1 x_2 + 2x_2^2 + x_1 - x_2$ の等高線

解答例

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#define rnd (double)rand()/(double)RAND_MAX
#define f(x1,x2) (x1*x1-x1*x2+2.0*x2*x2+x1-x2)
main()
```

```
{
   int i,n;
   double a,c1,c2,x1,x2,h1,h2,xp1,xp2;
   double y,ymin,eps=1e-6;
   srand(time(NULL));
   printf("Enter n\n");
   scanf("%d",&n);
   printf("Enter a c1 c2\n");
   scanf("%lf %lf %lf",&a,&c1,&c2);

   ymin = f(c1,c2);
   do {
      for (i = 1; i <= n; ++i) {
         h1 = (rnd-0.5)*2.0*a;
         h2 = (rnd-0.5)*2.0*a;
         x1 = c1+h1;
         x2 = c2+h2;
         y = f(x1,x2);
         if (y < ymin) {
            ymin = y;
            xp1 = x1;
            xp2 = x2;
         }
      }
      printf("a = %f  xp1 = %f  xp2 = %f  ymin = %f\n",
         a,xp1,xp2,ymin);
      a = a*0.5;
      c1 = xp1;
      c2 = xp2;
   } while (a > eps);
}
```

実行結果

```
Enter n
100
Enter a c1 c2
5 0 0
a = 5.000000  xp1 = -0.739127  xp2 = -0.365276  ymin = 0.169325
a = 2.500000  xp1 = -0.651791  xp2 = -0.142670  ymin = -0.136571
a = 1.250000  xp1 = -0.518994  xp2 = 0.107808  ymin = -0.278250
```

```
a = 0.625000  xp1 = -0.510070  xp2 = 0.117071  ymin = -0.279844
a = 0.312500  xp1 = -0.408254  xp2 = 0.159677  ymin = -0.285077
a = 0.156250  xp1 = -0.445513  xp2 = 0.141226  ymin = -0.285450
a = 0.078125  xp1 = -0.422926  xp2 = 0.139550  ymin = -0.285642
a = 0.039062  xp1 = -0.422926  xp2 = 0.139550  ymin = -0.285642
a = 0.019531  xp1 = -0.428939  xp2 = 0.141972  ymin = -0.285713
a = 0.009766  xp1 = -0.428085  xp2 = 0.142717  ymin = -0.285714
a = 0.004883  xp1 = -0.428085  xp2 = 0.142717  ymin = -0.285714
a = 0.002441  xp1 = -0.428545  xp2 = 0.142964  ymin = -0.285714
a = 0.001221  xp1 = -0.428545  xp2 = 0.142964  ymin = -0.285714
a = 0.000610  xp1 = -0.428525  xp2 = 0.142895  ymin = -0.285714
a = 0.000305  xp1 = -0.428525  xp2 = 0.142895  ymin = -0.285714
a = 0.000153  xp1 = -0.428543  xp2 = 0.142874  ymin = -0.285714
a = 0.000076  xp1 = -0.428575  xp2 = 0.142860  ymin = -0.285714
a = 0.000038  xp1 = -0.428575  xp2 = 0.142859  ymin = -0.285714
a = 0.000019  xp1 = -0.428572  xp2 = 0.142857  ymin = -0.285714
a = 0.000010  xp1 = -0.428572  xp2 = 0.142857  ymin = -0.285714
a = 0.000005  xp1 = -0.428572  xp2 = 0.142857  ymin = -0.285714
a = 0.000002  xp1 = -0.428572  xp2 = 0.142857  ymin = -0.285714
a = 0.000001  xp1 = -0.428571  xp2 = 0.142857  ymin = -0.285714
```

例題 5.2 の説明

1. 1 変数関数の場合の 2 変数関数への拡張である.

演習 5.2 2 変数関数

$$f(x_1, x_2) = (x_1 + x_1 x_2) e^{(-x_1^2 + 0.5 x_1 x_2 - x_2^2)} \tag{5.5}$$

を最小にする (x_{p_1}, x_{p_2}) と最小値 $y_{min} = f(x_{p_1}, x_{p_2})$ をランダムサーチ法によって求めよ.

（答）$x_{p_1} = -0.678316,\ x_{p_2} = 0.235210,\ y_{min} = -0.462037$

演習 5.3 3 変数関数

$$f(x_1, x_2, x_3) = x_1^2 + 2x_2^2 + 3x_3^2 + 2x_1 - 2x_2 + 4x_3 \tag{5.6}$$

を最小にする $(x_{p_1},\, x_{p_2},\, x_{p_3})$ と最小値 $y_{min} = f(x_{p_1},\, x_{p_2},\, x_{p_3})$ をランダムサーチ法によって求めよ.

（答）$(x_{p_1},\, x_{p_2},\, x_{p_3}) = (-1,\, 0.5,\, -2/3), \quad y_{min} = -2.833333$

第6章　常微分方程式の解法

6.1　オイラー法

6.1.1　1次系の解法

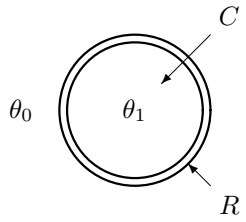

図 6.1: 熱系

図 6.1 の**熱系**において，容器の内部温度を θ_1，外部温度を θ_0 としよう．**ニュートンの冷却則**（物体の温度変化率は物体と周囲との温度差に比例する）から，θ_1 に関する微分方程式が

$$C\dot{\theta}_1 = \frac{1}{R}(\theta_0 - \theta_1) \tag{6.1}$$

と得られる．ただし，C は容器の熱容量，R は壁面の伝熱抵抗を表す．各変数を

$$x = \theta_1,\ u = \theta_0 \tag{6.2}$$

とおけば，次の数学モデルを得る．

$$\dot{x} = \frac{1}{CR}(u - x) \tag{6.3}$$

このような常微分方程式の解を数値計算する簡便な方法としてオイラー法がある．

オイラー法　式 (6.3) の差分近似

$$\frac{\Delta x}{\Delta t} \simeq \frac{1}{CR}(u - x) \tag{6.4}$$

から，x の増分

$$\Delta x \simeq \frac{1}{CR}(u - x)\Delta t \tag{6.5}$$

を得る．これから，Δt のちの変数 $x(t + \Delta t)$ を

$$x(t + \Delta t) \simeq x(t) + \Delta x \tag{6.6}$$

と近似する．

オイラー法による近似は，$x(t + \Delta t)$ を t のまわりでテイラー級数展開して，Δt の 1 次項までを考慮することに相当する．オイラー法では，刻み幅 Δt をある程度小さく与えないと十分な求解精度が得られない．

例題 6.1　式 (6.3) の解をオイラー法で計算せよ．また，計算結果をグラフ表示せよ．ただし，初期条件等は次のとおりとする．

$$x(0) = 0, \quad u = 1, \quad CR = 1, \quad \Delta t = 0.01, \quad t_f = 10$$

t_f は終了時刻である．

解答例

```
#include <stdio.h>
#define CR 1.0
main()
{
   double x,u,t=0.0,dx,dt,tf;
   x = 0.0;
   u = 1.0;
   dt = 0.01;
   tf = 10.0;

   while (t < tf) {
      printf("%f %f\n",t,x);
      dx = (u-x)/CR*dt;
      x = x+dx;
      t = t+dt;
   }
}
```

実行結果

```
0.000000 0.000000
0.010000 0.010000
0.020000 0.019900
0.030000 0.029701
0.040000 0.039404
0.050000 0.049010
0.060000 0.058520
0.070000 0.067935
  ...
```

グラフ表示

```
./a.out > data1
gnuplot
set xlabel 'time'
set ylabel 'x'
set nokey
set grid
plot 'data1' with lines
```

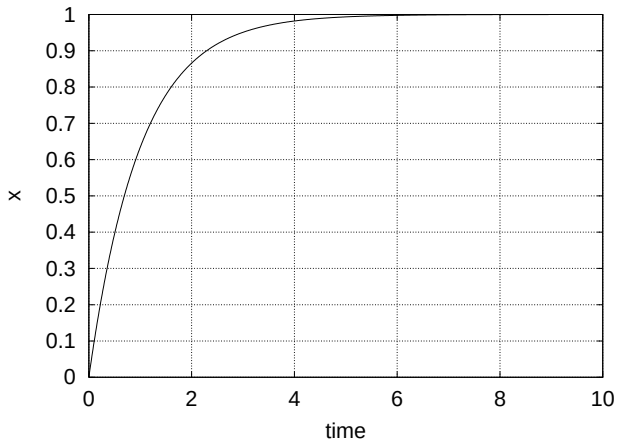

図 6.2: 例題 6.1 のグラフ

例題 6.1 の説明

1. CR は#define 文で定義した．もちろん，CR を double 型の変数として，CR=1.0 としてもよい．

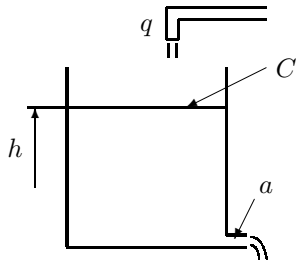

図 6.3: 流出孔のある水槽系

図 6.3 の**水槽系**において，q は流入する水の流量，a は流出孔の断面積とする．$a \ll C$ の場合，**トリチェリの定理**によって，孔から流出する水の速度は次式となる．

$$v = \sqrt{2gh} \tag{6.7}$$

ただし，g は重力加速度である．これを利用して水位 h に関する微分方程式を求めると

$$C\dot{h} = q - fav \tag{6.8}$$

すなわち

$$\dot{h} = \frac{q - fa\sqrt{2gh}}{C} \tag{6.9}$$

となる．f は**流量係数**であり，1 よりも小さな値を持つ．

演習 6.1 以下の条件に対して，式 (6.9) の解をオイラー法により計算せよ．また，結果をグラフ表示せよ．

$$h(0) = 1, \quad C = 1, \quad a = 0.1 \quad g = 9.8, \quad f = 0.9, \quad q = 0$$

$$\Delta t = 0.01, \quad t_f = 10$$

ただし，h が負になったときの安全策として，$\sqrt{2gh}$ の代わりに次式を用いること．

$$\mathrm{sgn}\,(h)\sqrt{2g|h|} = \begin{cases} \sqrt{2gh} & (h \geq 0) \\ -\sqrt{-2gh} & (h < 0) \end{cases} \tag{6.10}$$

6.1.2　2 次系の解法

図 6.4 の直列結合された水槽系に対する数学モデルは次式となる．

$$\left.\begin{aligned} \dot{h}_1 &= -\frac{fa_1}{C_1}\sqrt{2gh_1} + \frac{q}{C_1} \\ \dot{h}_2 &= -\frac{fa_2}{C_2}\sqrt{2gh_2} + \frac{fa_1}{C_2}\sqrt{2gh_1} \end{aligned}\right\} \tag{6.11}$$

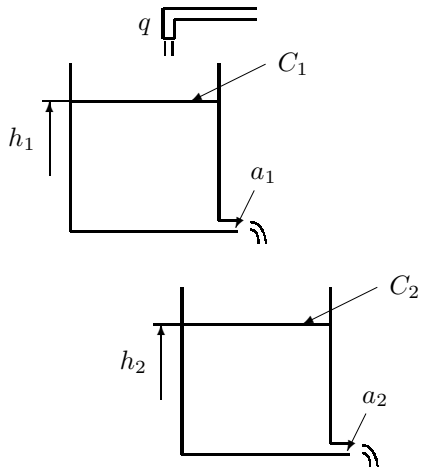

図 6.4: 直列結合された水槽系

例題 6.2 式 (6.11) の解をオイラー法で計算せよ．また，計算結果をグラフ表示せよ．ただし，初期条件等は次のとおりとする．

$$h_1(0) = 1, \quad h_2(0) = 0, \quad C_1 = C_2 = 1, \quad a_1 = a_2 = 0.1, \quad g = 9.8, \quad f = 0.9$$

$$q = 0, \quad \Delta t = 0.01, \quad t_f = 10$$

解答例

```
#include <stdio.h>
#include <math.h>
#define C1 1
#define C2 1
#define a1 0.1
#define a2 0.1
#define g 9.8
#define f 0.9
#define sgn(x)  (x ? x/fabs(x) : x)
main()
{
   double h1,h2,dh1,dh2,t=0.0,dt,tf,q;
   double a,b,c,d;
   h1 = 1.0;
   h2 = 0.0;
   dt = 0.01;
   tf = 10.0;
```

```
    q = 0.0;
    a = f*a1*sqrt(2*g)/C1;
    b = 1.0/C1;
    c = f*a2*sqrt(2*g)/C2;
    d = f*a1*sqrt(2*g)/C2;
    while (t < tf){
       printf("%f  %f  %f\n",t,h1,h2);
       dh1 = (-a*sgn(h1)*sqrt(fabs(h1))+b*q)*dt;
       dh2 = (-c*sgn(h2)*sqrt(fabs(h2))+d*sgn(h1)*sqrt(fabs(h1)))*dt;
       h1 = h1+dh1;
       h2 = h2+dh2;
       t = t+dt;
    }
}
```

実行結果

```
0.000000  1.000000  0.000000
0.010000  0.996016  0.003984
0.020000  0.992039  0.007709
0.030000  0.988070  0.011328
0.040000  0.984110  0.014865
0.050000  0.980157  0.018332
0.060000  0.976212  0.021737
0.070000  0.972276  0.025086
  ...
```

グラフ表示

```
./a.out > data1
gnuplot
set xlabel 'time'
set ylabel 'variables'
set nokey
set grid
plot 'data1' using 1:2 with lines, 'data1' using 1:3 with lines
```

例題 6.2 の説明

1. `#define sgn(x) (x ?  x/fabs(x) :  x)` で符号関数が定義できる.
   構文

   条件式 ? 式 1 : 式 2

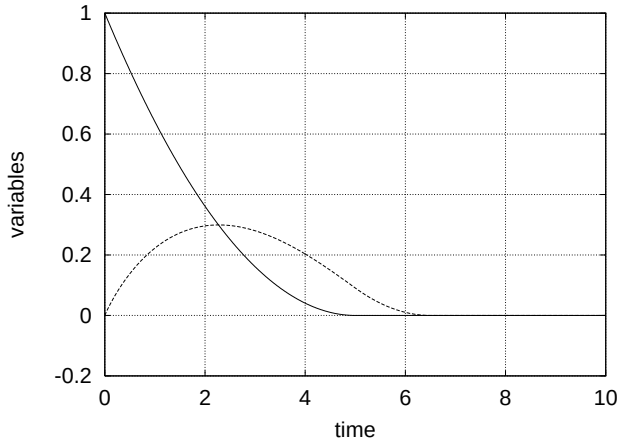

図 6.5: 例題 6.2 のグラフ

は，条件式 $\neq 0$ のとき式 1 を与え，条件式 $= 0$ のとき式 2 を与える．

2. 微分方程式の係数は，あらかじめ計算して `a,b,c,d` に代入した．

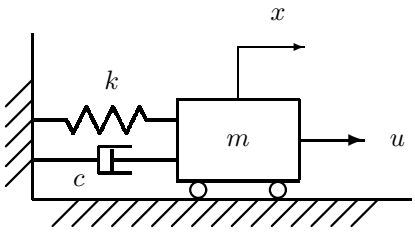

図 6.6: 1 自由度振動系

図 6.6 の **1 自由度振動系**の運動方程式は

$$m\ddot{x} + c\dot{x} + kx = u \tag{6.12}$$

すなわち

$$\ddot{x} = -\frac{k}{m}x - \frac{c}{m}\dot{x} + \frac{1}{m}u \tag{6.13}$$

で表される．ただし，x は物体の平衡点からの振れ，c はダンパーの**粘性減衰係数**，k は**ばね係数**である．ここで

$$x_1 = x, \quad x_2 = \dot{x} \tag{6.14}$$

とおくと，式 (6.13) は

$$\left.\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= -\frac{k}{m}x_1 - \frac{c}{m}x_2 + \frac{1}{m}u \end{aligned}\right\} \tag{6.15}$$

とオイラー法が適用できる形に変換することができる．

演習 6.2 式 (6.15) をオイラー法で解くプログラムを作成せよ．そして，次の条件で解を計算し，それらをグラフ表示せよ．

$$m = 1, \quad c = 0.3, \quad k = 1, \quad x(0) = 1, \quad \dot{x}(0) = 0, \quad u = 0$$

$$\Delta t = 0.05, \quad t_f = 40$$

6.2　ルンゲクッタ法

以下で示す（4 次）**ルンゲクッタ法**は，$x(t + \Delta t)$ を t のまわりでテイラー級数展開して，Δt の 4 次までの項を残したことに相当する近似法を用いる方法である．一般的な一階微分方程式

$$\dot{x}(t) = f(x(t)) \tag{6.16}$$

に対するルンゲクッタ法は次のように与えられる．

$$x(t + \Delta t) \simeq x(t) + \frac{d_1 + 2d_2 + 2d_3 + d_4}{6} \tag{6.17}$$

ただし

$$\left.\begin{aligned} d_1 &= f(x(t))\Delta t \\ d_2 &= f(x(t) + d_1/2)\Delta t \\ d_3 &= f(x(t) + d_2/2)\Delta t \\ d_4 &= f(x(t) + d_3)\Delta t \end{aligned}\right\} \tag{6.18}$$

ルンゲクッタ法はオイラー法に比べ格段に精度が良く，刻み幅をオイラー法よりも大きくとることができる．通常，常微分方程式の数値解法では，ルンゲクッタ法が使われる．

6.2.1　1 次系の解法

例題 6.3 式 (6.3) の解をルンゲクッタ法で計算せよ．また，計算結果をグラフ表示せよ．ただし，初期条件等は次のとおりとする．

$$x(0) = 0, \quad u = 1, \quad CR = 1, \quad \Delta t = 0.05, \quad t_f = 10$$

t_f は終了時刻である．

解答例

```
#include <stdio.h>
#define CR 1.0
#define u 1.0
#define f(x) (u-x)/CR
main()
{
   double x,y,d1,d2,d3,d4,t=0.0,dt,tf;
   x = 0.0;
   dt = 0.05;
   tf = 10.0;

   while (t < tf) {
      printf("%f %f\n",t,x);
      d1 = f(x)*dt;
      y = x+d1*0.5;
      d2 = f(y)*dt;
      y = x+d2*0.5;
      d3 = f(y)*dt;
      y = x+d3;
      d4 = f(y)*dt;
      x = x + (d1+2.0*d2+2.0*d3+d4)/6.0;
      t = t+dt;
   }
}
```

実行結果

```
0.000000 0.000000
0.050000 0.048771
0.100000 0.095163
0.150000 0.139292
0.200000 0.181269
0.250000 0.221199
0.300000 0.259182
0.350000 0.295312
  ...
```

グラフ表示

```
./a.out > data1
gnuplot
```

```
set xlabel 'time'
set ylabel 'x'
set nokey
set grid
plot 'data1' with lines
```

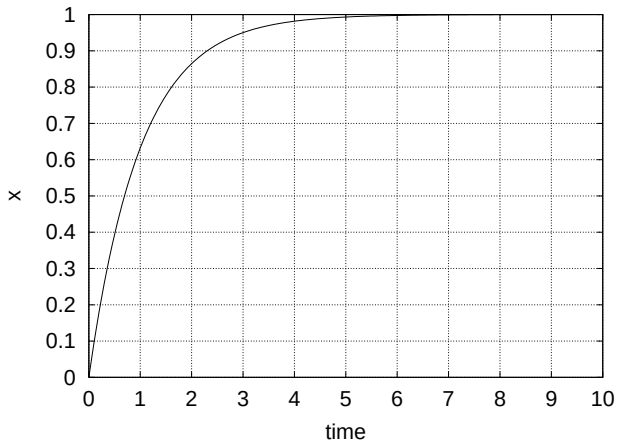

図 6.7: 例題 6.3 のグラフ

例題 6.3 の説明

1. `#define` 文で $f(x)$ を定義すればプログラムが簡潔になる．

演習 6.3 次の条件に対する式 (6.9) の解をルンゲクッタ法により計算せよ．また，結果をグラフ表示せよ．

$$h(0) = 1, \quad C = 1, \quad a = 0.1 \quad g = 9.8, \quad f = 0.9, \quad q = 0$$

$$\Delta t = 0.05, \quad t_f = 10$$

6.2.2　2 次系の解法

例題 6.4 式 (6.11) の解をルンゲクッタ法で計算せよ．また，計算結果をグラフ表示せよ．ただし，初期条件等は次のとおりとする．

$$h_1(0) = 1, \quad h_2(0) = 0, \quad C_1 = C_2 = 1, \quad a_1 = a_2 = 0.1, \quad g = 9.8, \quad f = 0.9$$

$$q = 0, \quad \Delta t = 0.05, \quad t_f = 10$$

解答例

```
#include <stdio.h>
#include <math.h>
#define C1 1.0
#define C2 1.0
#define a1 0.1
#define a2 0.1
#define g 9.8
#define f 0.9
#define q 0.0
#define sgn(x) (x ? x/fabs(x) : x)
#define f1(h1) (q-f*a1*sgn(h1)*sqrt(2.0*g*fabs(h1)))/C1
#define f2(h1,h2) (a1*sgn(h1)*sqrt(2.0*g*fabs(h1))\
-a2*sgn(h2)*sqrt(2.0*g*fabs(h2)))*f/C2
main()
{
   double h1,h2,y1,y2;
   double d11,d12,d13,d14,d21,d22,d23,d24,t=0.0,dt,tf;
   h1 = 1.0;
   h2 = 0.0;
   dt = 0.05;
   tf = 10.0;

   while (t < tf) {
      printf("%f %f %f\n",t,h1,h2);
      d11 = f1(h1)*dt;
      d21 = f2(h1,h2)*dt;
      y1 = h1+d11*0.5;
      y2 = h2+d21*0.5;
      d12 = f1(y1)*dt;
      d22 = f2(y1,y2)*dt;
      y1 = h1+d12*0.5;
      y2 = h2+d22*0.5;
      d13 = f1(y1)*dt;
      d23 = f2(y1,y2)*dt;
      y1 = h1+d13;
      y2 = h2+d23;
      d14 = f1(y1)*dt;
      d24 = f2(y1,y2)*dt;
      h1 = h1+(d11+2.0*d12+2.0*d13+d14)/6.0;
      h2 = h2+(d21+2.0*d22+2.0*d23+d24)/6.0;
```

```
        t = t+dt;
      }
    }
```

実行結果

```
0.000000 1.000000 0.000000
0.050000 0.980177 0.018088
0.100000 0.960552 0.034490
0.150000 0.941126 0.049828
0.200000 0.921898 0.064297
0.250000 0.902869 0.078012
0.300000 0.884038 0.091050
0.350000 0.865406 0.103469
  ...
```

グラフ表示

```
./a.out > data1
gnuplot
set xlabel 'time'
set ylabel 'variables'
set nokey
set grid
plot 'data1' using 1:2 with lines, 'data1' using 1:3 with lines
```

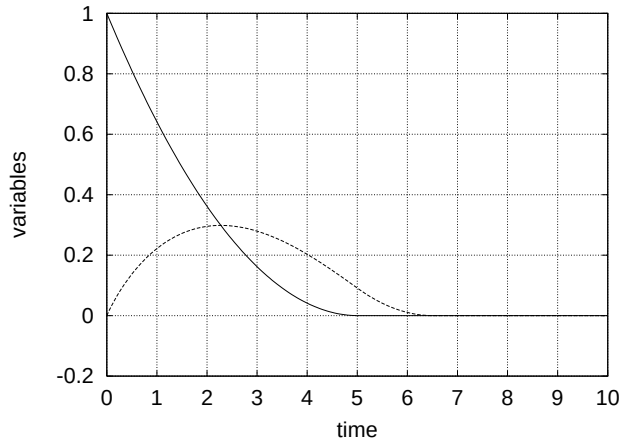

図 6.8: 例題 6.4 のグラフ

例題 6.4 の説明

1. `#define` 文は原則として 1 行に書く．複数行に書きたい場合，

   ```
   \
   ```

   を置き，その直後に改行する（このようにすればコンパイル時に 1 行に併合される）．

演習 6.4 式 (6.15) をルンゲクッタ法で解くプログラムを作成せよ．そして，次の条件で解を計算し，それらをグラフ表示せよ．

$$m = 1, \quad c = 0.3, \quad k = 1, \quad x(0) = 1, \quad \dot{x}(0) = 0, \quad u = 0$$

$$\Delta t = 0.1, \quad t_f = 40$$

演習 6.5 次の条件

$$m = 1, \quad c = 0, \quad k = 1, \quad x(0) = 1, \quad \dot{x}(0) = 0, \quad u = 0$$

$$\Delta t = 0.01, 0.05, 0.1, \quad t_f = 40$$

を与えた場合の式 (6.15) の解（非減衰振動解）をオイラー法とルンゲクッタ法によって求め，グラフ表示することにより両者を比較せよ．

第7章　C言語文法概論

7.1　Cプログラムの例

7.1.1　画面へ文字を表示させる

画面に hello，world と表示する C プログラムは次のようである．

例題 7.1

```
#include <stdio.h>
main()
{
   printf("hello, world\n");
}
```

C プログラムは関数と呼ばれるプログラム単位によって構成される．C プログラムをコンパイルして実行すると，C は main という関数を捜して実行するので，main 関数が唯一つ必要である．{...} はプログラム単位の範囲を示す．

printf 関数は，‘" "’で囲まれる内容を画面に表示する関数である．‘\n’ を置くとそこで改行される．‘;’ は文の終わりを表す．

次の二つの例題によって ‘\n’ の働きが理解できる．

例題 7.2

```
#include <stdio.h>
main()
{
   printf("equation (1)");
   printf("equation (2)");
   printf("equation (3)");
}
```

実行結果

```
equation (1)equation (2)equation (3)
```

例題 7.3

```
#include <stdio.h>
main()
{
   printf("equation (1)\n");
   printf("equation (2)\n");
   printf("equation (3)\n");
}
```

```
equation (1)
equation (2)
equation (3)
```

このプログラムはまた次のように書くこともできる.

例題 7.4

```
#include <stdio.h>
main()
{
   printf("equation (1)");
   printf("\n");
   printf("equation (2)");
   printf("\n");
   printf("equation (3)");
   printf("\n");
}
```

7.1.2　数値を入力して計算を行う

整数 a, b を入力して

$$\left.\begin{array}{rcl} x_1 & = & a+b \\ x_2 & = & a-b \\ x_3 & = & a \times b \\ x_4 & = & a \text{ を } b \text{ で割ったときの剰余} \end{array}\right\} \tag{7.1}$$

を計算して，結果を表示するプログラムは次のように書ける.

例題 7.5

```
#include <stdio.h>
main()
{
```

```
    int a,b,x1,x2,x3,x4;

    printf("Enter a b \n");
    scanf("%d %d",&a,&b);

    x1 = a+b; x2 = a-b; x3 = a*b; x4 = a%b;

    printf("x1 = %d  x2 = %d  x3 = %d  x4 = %d\n",x1,x2,x3,x4);
}
```

使用する変数はすべて**型**の**宣言**しなければならない．上の場合

```
int a,b,x1,x2,x3,x4;
```

によって，int 型の変数を宣言している．変数名で使える記号は，英字，アンダースコア '_'，数字であり，英字か '_' を最初に置く．大文字と小文字は区別される．

scanf は入力を行う関数である．**引数**があり，それらは次を意味する．

```
"%d %d"      int 型の変数が二つある．
&a,&b        a のアドレスと b のアドレス
```

a, b の型宣言をした時点で，a, b に計算機メモリーのアドレスが割り当てられるのであるが，scanf 関数にそのアドレスを渡して変数の値を設定する．

'+', '-', '*', '%' などの演算記号が用意されている．'=' は右辺を左辺に代入するという働きを持つ．

printf 関数の使い方にも注意されたい．int 型書式%d によって，変数の出力を行っている．

C プログラムでは，**空白**，空行を任意に置ける．また，プログラムを読みやすくするため，適宜コメントを '/* */' で囲んで挿入できる．

例題 7.6

```
#include <stdio.h>
main()
{
    int a,b,x1,x2,x3,x4;   /* definition of variables */

    printf("Enter a b \n");
    scanf("%d %d",&a,&b);

/* calculations of the values
                required in the problem */
```

```
    x1 = a+b; x2 = a-b; x3 = a*b; x4 = a%b;

    printf("x1 = %d  x2 = %d  x3 = %d  x4 = %d\n",x1,x2,x3,x4);
}
```

7.2　データ型

表 7.1 の**データ型**がしばしば使われる．

表 7.1: データ型の例

型名	意味	バイト長（一例）
char	文字型	1
int	整数型	4
float	単精度実数型	4
double	倍精度実数型	8

通常，整数には int が，実数には double が使われる．典型的な型宣言の例を以下に示す．

例題 7.7

```
#include <stdio.h>
main()
{
    int i,j,k;
    double x,y,z;
    double value1,value2,value3;
    ...
}
```

宣言は複数行に分割できる．また，宣言の中で，変数の初期値を設定できる．

例題 7.8

```
#include <stdio.h>
main()
{
    int i=0,j=-1,k;
    double x=0.0,y=10.0,z;
    double value1,value2,value3;
```

```
    ...
}
```

初期化されていない変数には，通常どのような数値が入っているかわからないので注意する（BASIC のように，宣言すれば 0 と初期化されることはない）．

プログラムの途中で値を設定するときにもいえるが，int 型の変数には

```
1  2  10  25 1000
```

のように整数形式を用い，double 型の変数には

```
1.0  1.23  1.04e-3  3.14
```

という実数形式を用いる．これは，型を一貫させるということである．事実，計算の際には型に注意する必要がある．次の例を見てみよう．

```
#include <stdio.h>
main()
{
   int i=1;
   double x;
   x = i;
   printf("x = %f\n",x);
}
```

代入によって，右辺の型が左辺の型に変換されるのであるが，このような型の自動変換が常に都合よくいくとは限らない．例えば

```
#include <stdio.h>
main()
{
   int i=1,j=2;
   double x;
   x = i/j;
   printf("x = %f\n",x);
}
```

を実行すると

```
x = 0.000000
```

となる．これは，i/j が int 型として計算され（/は割り算を表す），小数点以下が切り捨てられ，その結果が double 型に変換されたからである．したがって，型の異なる演算はできるだけ避けるようにする．もし，その必要があれば，変数の前に (型名) を置き，型を変換して（キャストという）から演算するようにする．上のプログラムは，次のように修正すれば問題なく実行される．

```
#include <stdio.h>
main()
{
   int i=1,j=2;
   double x;
   x = (double)i/(double)j;
   printf("x = %f\n",x);
}
```

7.3　配列

行列計算を行うときなど，**配列**が使えれば便利である．C では，任意のデータ型に対して配列を宣言できる．配列宣言は次のように書く．

例題 7.9

```
#include <stdio.h>
main()
{
   int i,j,k,s[5];
   double x,y,z,a[20][20],b[2][20][20];
   ...
}
```

s[5] は 5 つの要素を持つ 1 次元配列，a[20][20] は 2 次元配列（20×20），b[2][20][20] は 3 次元配列（$2 \times 20 \times 20$）である．4 次元以上の配列も同様に宣言できる．配列の添字は 0 から始まることに注意されたい．すなわち，s[5] と宣言すれば

```
s[0], s[1], s[2], s[3], s[4]
```

が使用できる．配列の要素数の指定には定数を用いる．BASIC のように変数を置くことはできない．配列を用いた C プログラムの例を以下に示す．

例題 7.10

```
#include <stdio.h>
main()
{
   int s[5],sum;
   printf("Enter s[0] s[1] s[2]\n");
   scanf("%d %d %d",&s[0],&s[1],&s[2]);
   sum = s[0]+s[1]+s[2];
   printf("sum = %d\n",sum);
}
```

7.4 定数を表す記号の定義

次の例に示すように，#define を使ってプログラムの最初で，**定数**を定義することができる．

例題 7.11

```
#include <stdio.h>
#define N 5
#define PI 3.14
main()
{
   double a[N];
   a[0] = PI;
   a[1] = PI*PI;
   a[2] = PI*PI*PI;
   printf("a[0] = %f  a[1] = %f  a[2] = %f\n",a[0],a[1],a[2]);
}
```

#define は，コンパイル時に最初の文字列を後の文字列で置き換えるという働きをする．配列の要素数の指定にも使え便利である．

7.5 文字と文字列

文字は char 型として，**文字列**は char 型の配列として扱う．文字と文字列の使用例を次のプログラムで示す．

例題 7.12

```
#include <stdio.h>
#define N 100
main()
{
   char a1='A',a2;
   char b1[N]="Matrix A",b2[N],b3[N];
   a2 = 'B';
   strcpy(b2,"Equation (1)");
   printf("Enter b3\n");
   gets(b3);
   printf("a1 = %c a2 = %c \nb1 = %s \nb2 = %s \nb3 = %s\n",
   a1,a2,b1,b2,b3);
}
```

文字定数は‘’ ’’で囲む．char の場合，型宣言時，プログラム中で文字定数を代入することができる．文字列定数は‘" "’で囲む．こちらは，型宣言時に設定できるが，プログラム中では代入できない．プログラム中で文字列変数に文字列定数を代入する場合，上のように strcpy 関数（文字列をコピーする関数）を使う．また，%s は文字列の入出力書式であるが，文字列をキーボードから入力する場合

```
scanf("%s",b3);
```

では，空白の入力ができない（確かめてみよ）．空白を含む文字列を入力するには gets 関数を利用する．

7.6　入出力の書式

文字や数値を scanf 関数で入力する場合，よく使う書式を表 7.2 に示す．また，文

表 7.2: 入力書式の例 (scanf 関数)

書式	対象
%c	文字
%d	整数
%f	単精度実数
%lf	倍精度実数

字，文字列，数値を printf 関数で出力する場合の書式の例を表 7.3 に示す．整数，実

表 7.3: 出力書式の例 (printf 関数)

書式	対象
%c	文字
%s	文字列
%d	整数
%f	単精度および倍精度実数

数の出力は，簡単には，表 7.3 の書式でよいが， 整数については

```
printf("a = %5d\n",a);
```

と書けば，5 桁の整数表示となる．また，実数については

```
printf("b = %10.4f\n",b);
```

とすれば，全文字数 10，小数点以下の 4 桁の表示となり

```
printf("c = %10.4e\n",b);
```

で，全文字数 10，小数点以下 4 桁の指数表示となる．

7.7 標準関数と定義ヘッダ

C は，関数をプログラムの単位として構成されることを最初に述べた．C には，**標準関数**が用意されていて，少し複雑な仕事をしようとするとこれらの関数のお世話になることとなる．上述の例では，printf，scanf，strcpy，gets が使われた．

関数にもデータと同じように型があり，関数を使うとき，関数の型を定義する必要がある（定義しなければ，int 型となる）．よって，標準関数を使う場合，正式には，その関数の定義や必要な定数の定義が含まれているファイル（**定義ヘッダ**）をインクルード（挿入）しておく．例えば，printf に対する定義ヘッダは stdio.h というファイルであり，次のように定義ヘッダをインクルードして使う．

例題 7.13

```
#include <stdio.h>
main()
{
   printf("Let's get going.\n");
}
```

printf や scanf を使用する場合，大抵の処理系で，定義ヘッダを省略してもプログラムは問題なく走るが，原則として，対応する定義ヘッダをインクルードしなければならない．上述の標準関数の定義ヘッダを表 7.4 に示す．

表 7.4: 定義ヘッダの例

関数	内容	定義ヘッダ
printf	書式つき出力	stdio.h
scanf	書式つき入力	stdio.h
gets	文字列の入力	stdio.h
strcpy	文字列のコピー	string.h

7.8 ポインタ

C では，データが格納されているメモリのアドレスを利用したプログラムがよく書かれる．メモリアドレスに関する操作は，アセンブラのような低水準言語が得意とするところであるが，C にもこのような機能がある．

各データ型に対して，データが格納されているアドレスを示す**ポインタ**と呼ばれる変数を宣言できる．次の例を検討しよう．

例題 7.14

```
#include <stdio.h>
main()
{
   int p0;
   int *p;
   p = &p0;
   printf("p = %d\n",p);
   *p = 10;
   printf("*p = %d\n",*p);
   p = p+1;
   printf("p = %d\n",p);
}
```

実行結果

```
p = 38076228
*p = 10
p = 38076232
```

`int *p;` で `p` が `int` 型のポインタであることを宣言している．ポインタは変数と同様に初期化して使う．使ってよいアドレスを指定しなければならないが，簡単な方法として，上のプログラムのように，同じ型のダミーの変数を宣言して

```
p = &p0;
```

とする方法がある．‘`&`’ は変数のアドレスを求める演算子である．`p0` には適切なアドレスが割り当てられているので，このアドレスを代入したわけである．次に，アドレス `p` にデータ 10 を代入するには

```
*p = 10;
```

と書く．‘`*`’ はアドレス `p` の内容を示す演算子であり，`*p` を変数として扱うことができる．紛らわしいが，宣言における `int *p;` の`*p` には変数という意味はなく，単に，`p` という `int` 型のポインタという意味である．よって，宣言時に `p` を初期化する次のようなプログラムも書ける．

```
int p0;
int *p = &p0;
```

`p` の値を出力する書式は`%d` であるが，`p` は `int` 型でないことに注意する．試しに

```
p = p+1;
```

という演算を行うと，int 型のメモリ単位（例の場合 4 バイト）だけ，p が増え，次のアドレスが計算されることがわかる（p+1 が使用してよいアドレスかどうかは別として）．

7.9 配列とポインタ

C では，配列とポインタが密接に関係づけられている．次の具体例で説明しよう．

例題 7.15

```
#include <stdio.h>
main()
{
   double a[10];
   a[0] = 1.0; a[1] = 2.0; a[2] = 3.0;
   printf("a = %d  a[0] = %f\n",a,*a);
   printf("a + 1 = %d  a[1] = %f\n",(a+1),*(a+1));
   printf("a + 2 = %d  a[2] = %f\n",(a+2),*(a+2));
}
```

実行結果

```
a = 38076152  a[0] = 1.000000
a + 1 = 38076160  a[1] = 2.000000
a + 2 = 38076168  a[2] = 3.000000
```

まず

```
double a[10];
```

で倍精度実数の配列が宣言された．a は a[0] のアドレスを与えるポインタであり，配列の宣言をした時点で適切なアドレスが割り当てられている（しかし，通常のポインタと違い，a の値を変えることはできない）．また，a[1]，a[2]，... のアドレスが a を基準にして規則的に割り当てられていることもわかる．

2 次元配列

```
double b[10][10];
```

を宣言した場合，第 2 添字を除いた b[0]，b[1]，... が b[0][0]，b[1][0]，... のアドレスを与えるポインタとなる．b も b[0][0] のアドレスを与えるポインタであり，b = b[0] であるが，b に対してポインタの演算はできないことに注意する（*演算子が適用できない）．次の例を参照されたい．

例題 7.16

```
#include <stdio.h>
main()
{
   double b[10][10];
   b[0][0] = 1.0; b[1][0] = 2.0; b[2][0] = 3.0;
   printf("%d  *b = %f\n",b,*b);
   printf("%d  *b[0] = %f\n",b[0],*b[0]);
   printf("%d  *b[1] = %f\n",b[1],*b[1]);
   printf("%d  *b[2] = %f\n",b[2],*b[2]);
}
```

実行結果

```
38075432  *b = 0.000000
38075432  *b[0] = 1.000000
38075512  *b[1] = 2.000000
38075592  *b[2] = 3.000000
```

一般に，配列の最後の添字を除けば，最後の添字を 0 とおいた変数のアドレスを与えるポインタとなる．そして，すべての添字を除いた変数名が，すべての添字を 0 とおいた変数のアドレスを保持する（‘*’ 演算子が使えない）ポインタである．例えば

```
double c[10][10]...[10];
```

と宣言した場合，c[0][0]...[0] のアドレスは c によって参照できる．すなわち

```
c = &c[0][0]...[0]
```

である．

7.10　C における関数の作り方

7.10.1　通常の関数

C で関数と呼ぶプログラムのクラスは広い．プログラム本体の main() でさえ，一つの関数である．すなわち，C の関数とは，BASIC や FORTRAN でいう，メインプログラム，サブプロシージャ，サブルーチン，ユーザ定義関数である．ある単位のプログラムを関数としてまとめるか，あるいは，main() の中で記述するかは，プログラマの意向次第であるが，関数を作ることによって，プログラムの可読性が向上したり，プログラム作成が容易になるような場合には，積極的に関数化を進めればよい．

簡単な例題によって，関数の作り方を説明しよう．

例題 7.17

```
#include <stdio.h>
main()
{
   double a,b,c;
   double wa();
   printf("Enter a b\n");
   scanf("%lf %lf",&a,&b);
   c = wa(a,b);
   printf("c = %f\n",c);
}

double wa(x,y)
double x,y;
{
   double z;
   z = x+y;
   return(z);
}
```

実数 a, b を与えて和 a+b を計算する関数 wa の例である．関数は，main() の外に書き，型の指定をして，引数を列挙する．そして，関数の内容を書く前に，引数の型宣言を行う．引数の名前は，呼ぶ側の変数と違ってもよい．関数中で宣言する変数は，その関数内でのみ有効であり，関数が呼ばれる度に宣言される．したがって，前に呼ばれたときの値は保持しない．関数を呼ぶ側でも，関数の型宣言をする．

引数として，変数を渡して，関数中でそれを変えても，元のプログラムの変数には影響しない．例えば，次のプログラムを実行すればこのことが確認できる．

例題 7.18

```
#include <stdio.h>
main()
{
   int a=1;
   int test();
   printf("a = %d\n",a);
   test(a);
   printf("a = %d\n",a);
}

int test(a)
int a;
```

```
{
   a = a + 1;
   printf("a = %d\n",a);
   return(0);
}
```

実行結果

```
a = 1
a = 2
a = 1
```

test 関数の中で a に 1 を加えている．関数としての値はもたないが，このような場合，関数型を int にして，正常終了を示す 0 を返すようにする．

次のプログラムはデータ入力部を簡素化する関数の例である．

例題 7.19

```
#include <stdio.h>
main()
{
   double x,y,z;
   double input();

   x = input('x');
   y = input('y');
   printf("x = %f  y = %f\n",x,y);
}

double input(a)
char a;
{
   double b;
   printf("Enter %c\n",a);
   scanf("%lf",&b);
   return(b);
}
```

実行結果

```
Enter x
2
Enter y
3
x = 2.000000  y = 3.000000
```

7.10.2 関数に渡す変数を変更する場合

引数を加工して返したい場合がある．関数には，引数をやりとりする機能はないが，ポインタを利用すれば，結果として，同じことが可能になる．次の例を見てみよう．

例題 7.20

```
#include <stdio.h>
main()
{
   int a=1,b=2;
   int swap();
   printf("a = %d  b = %d\n",a,b);
   swap(&a,&b);
   printf("a = %d  b = %d\n",a,b);
}

int swap(x,y)
int *x,*y;
{
   int z;
   z = *x;
   *x = *y;
   *y = z;
   return(0);
}
```

実行結果

```
a = 1  b = 2
a = 2  b = 1
```

swap は，a, b の値を交換する関数である．引数として，a, b のアドレス&a, &b を与え，これらを関数側でポインタ x, y として受けている．このように，対象とする変数のアドレスに対して操作を行うように関数を書けば，変数の値を変えることができる．

配列の場合，すでに述べたように，変数名がポインタなので，関数の引数として，変数名を与えれば，ちょうど，BASIC や FORTRAN のサブプロシージャの感覚で，引数をとおして変数のやりとりをするようなプログラムが作れる．配列を使った関数の例を次に示す．

例題 7.21

```
#include <stdio.h>
```

```
#include <string.h>
#define N 100
main()
{
   char s[N] = "Hello, how are you?";
   int words();
   printf("%s\n",s);
   words(s);
   printf("%s\n",s);
}

int words(x)
char x[N];
{
   strcpy(x,"I'm fine, thank you.");
   return(0);
}
```

実行結果

```
Hello, how are you?
I'm fine, thank you.
```

最初，文字列 s に Hello, how are you?を設定し，関数 words の中で I'm fine, thank you. に変えている．s を引数として渡すことで，関数はこの文字列のアドレスを知り，そのアドレスに対して作業を行っている．もう一つ例を示そう．

例題 7.22

```
#include <stdio.h>
#define N 10
main()
{
   double a[N][N];
   int test();
   a[0][0] = 1.0; a[1][2] = 2.0;
   printf("a[0][0] = %f  a[1][2] = %f\n",a[0][0],a[1][2]);
   test(a);
   printf("a[0][0] = %f  a[1][2] = %f\n",a[0][0],a[1][2]);
}

int test(x)
```

```
double x[N][N];
{
   x[0][0] = 3.0; x[1][2] = 4.0;
   return(0);
}
```

実行結果

```
a[0][0] = 1.000000  a[1][2] = 2.000000
a[0][0] = 3.000000  a[1][2] = 4.000000
```

a[0][0],a[1][2] の内容が関数 test を呼ぶことによって変わることがわかる.

7.11 変数宣言の補足

7.11.1 静的変数

普通の変数（**自動変数**という）の場合，関数中で使われる変数は，呼び出しの度に宣言され，前回の値を保持していない．ところが，型宣言の前に static をつけると，**静的変数**になり，前回呼び出したときの変数値を保持することができる.

例題 7.23

```
#include <stdio.h>
main()
{
   int i;
   int add();

   i = add();
   printf("i = %d\n",i);
   i = add();
   printf("i = %d\n",i);
   i = add();
   printf("i = %d\n",i);
}

int add()
{
   static int j = 0;
   ++j;
   return(j);
}
```

実行結果

```
i = 1
i = 2
i = 3
```

関数 add を呼ぶ度に，i が 1 ずつ増えていくことがわかる．add 関数中の

```
static int j = 0;
```

で最初の呼び出し時に j が初期化され，次回呼び出し時には前回の j が使われる．

7.11.2　大域変数

main() の前で定義された変数は，すべての関数に共通の変数（**大域変数**）となる．比較的小規模なプログラムでは，このような変数は便利に使える．次の例を見てみよう．

```
#include <stdio.h>
double xxx;
main()
{
   int half();

   xxx = 1.0;
   half();
   printf("xxx = %f\n",xxx);
   half();
   printf("xxx = %f\n",xxx);
   half();
   printf("xxx = %f\n",xxx);
}

int half()
{
   xxx = xxx*0.5;
   return(0);
}
```

実行結果

```
xxx = 0.500000
xxx = 0.250000
xxx = 0.125000
```

関数 half で xxx を変えれば，main() でも変わることがわかる．

プログラムの規模がある程度大きくなり，変数の数が増えると，すべての関数で使える大域変数の管理には細心の注意を要する．当然，大域変数を不本意に変更してしまう間違いが生じやすくなるのである．

7.12 条件判定・繰り返し

一般のプログラミングでもいえるが，特に計算プログラミングの本質は，**条件判定**と**繰り返し**である．この機能を持つプログラミングと計算機の高速性とがあいまって，大規模・複雑な計算問題を解くことが可能となる．

7.12.1 if, else, else if

条件判定は if 文で行う．if 文は次の構造を持つ．

```
if (condition)
    statement 1;
else
    statement 2;
```

condition が成立した場合（0 でない場合），*statement 1* を実行し，成立しない場合（0 の場合），*statement 2* を実行する．*statement 2* がない場合，else を省略できる．

```
if (condition)
    statement 1;
```

また，if の中で else if を使える．

```
if (condition)
    statement 1;
else if (condition)
    statement 2;
else
    statement 3;
```

statement 1, 2, 3 が複数の文の場合，それぞれを ‘{ }’ で囲む．

では，if の使い方を例題によって説明しよう．

例題 7.24 a, b を入力して，$b \neq 0$ ならば

$$y = a/b \tag{7.2}$$

を出力するプログラムを作成せよ．

解答例

```
#include <stdio.h>
main()
{
   double a,b,y;
   printf("Enter a b\n");
   scanf("%lf %lf",&a,&b);
   if (b != 0.0)
      y = a/b;
   printf("y = %f\n",y);
}
```

例題 7.25 x を入力して

$$y = \begin{cases} 1 & \text{if} \quad x \geq 0 \\ -1 & \text{if} \quad x < 0 \end{cases} \tag{7.3}$$

を出力するプログラムを作成せよ.

解答例

```
#include <stdio.h>
main()
{
   double x,y;
   printf("Enter x\n");
   scanf("%lf",&x);
   if (x >= 0.0)
      y = 1.0;
   else
      y = -1.0;
   printf("y = %f\n",y);
}
```

例題 7.26 x を入力して

$$y = \begin{cases} 1 & \text{if} \quad x > 0 \\ 0 & \text{if} \quad x = 0 \\ -1 & \text{if} \quad x < 0 \end{cases} \tag{7.4}$$

を出力するプログラムを作成せよ.

解答例

```
#include <stdio.h>
main()
```

```
{
   double x,y;
   printf("Enter x\n");
   scanf("%lf",&x);
   if (x > 0.0)
      y = 1.0;
   else if (x < 0.0)
      y = -1.0;
   else
      y = 0.0;
   printf("y = %f\n",y);
}
```

if ループは何重にもできる．また，if の中で else if を何回でも使える．if や while（後述）で使われる関係演算子を表 7.5 に示す．

表 7.5: 関係演算子

演算子	意味
<	$<$
<=	$\leq$
>	$>$
>=	$\geq$
==	$=$
!=	$\neq$

また，論理演算子を表 7.6 に示す．これらを関係演算子と組み合わせて使うと複雑な if 文や条件式をコンパクトに書くことができる．

表 7.6: 論理演算子

演算子	意味
&&	AND
||	OR
!	NOT

7.12.2　switch

switch 文はある変数の値に従って，処理を切り換える制御文である．構文は次のようである．

```
switch (variable)
{
    case value 1:
       {statements 1}
      break;
    case value 2:
      {statements 2}
      break;
       ⋮
    default:
       {statements}
       break;
}
```

variable が *value 1* に等しいとき，*statements 1* が実行され，break で switch 文を終える．break がないと，次の case を実行してしまう．case はいくらでも書ける．*variable* がいずれの値にも該当しない場合，default 部分を実行する．もし，そのような部分がなければ，default を省略できる．

例題 7.27 n を入力し

$n = -1$	ならば	minus one
$n = 0$	ならば	zero
$n = 1$	ならば	one
その他の場合		other value

を出力するプログラムを作成せよ．

解答例

```
#include <stdio.h>
main()
{
   int n;
   printf("Enter n\n");
   scanf("%d",&n);
   switch (n)
   {
      case -1:
```

```
            printf("minus one\n");
            break;
        case 0:
            printf("zero\n");
            break;
        case 1:
            printf("one\n");
            break;
        default:
            printf("other value\n");
            break;
     }
}
```

switch は次のような使い方もできる.

例題 7.28

```
#include <stdio.h>
main()
{
   int n;
   printf("Enter n\n");
   scanf("%d",&n);
   switch (n)
   {
      case 1:
         printf("1\n");
         break;
      case 2: case 3: case 4:
         printf("2 or 3 or 4\n");
         break;
      case 5:
         printf("5\n");
         break;
      default:
         printf("other value\n");
         break;
    }
}
```

7.12.3　while

繰り返しを実行する文として while 文がある．while は次の構文で使う．

while (*condition*)
{
　statements
}

while は *condition* が成立する（値が 0 でない）限り，*statements* を繰り返し実行する．while ループは break でいつでも中断できる．

例題 7.29 与えられた正数 a を $\epsilon = 0.001$ よりも小さくなるまで 2 で割り続けるプログラムを作成せよ．

解答例

```
#include <stdio.h>
main()
{
   double eps=0.001,a;
   printf("Enter a\n");
   scanf("%lf",&a);
   while (a >= eps)
   {
      a = a*0.5;
      printf("a = %f\n",a);
   }
}
```

実行結果

```
Enter a
1
a = 0.500000
a = 0.250000
a = 0.125000
a = 0.062500
a = 0.031250
a = 0.015625
a = 0.007812
a = 0.003906
a = 0.001953
a = 0.000977
```

7.12.4 do-while

do-while 文は，while 文とほぼ同じ働きをする．ただ，文を実行したあとで，ループ継続条件を判定するところが異なっている．よって，文は少なくとも 1 回は実行されることになる．

do-while は次の構文で使う．

do
{
    statements
} while (*condition*)

do-while は *condition* が成立する（値が 0 でない）限り，*statements* を繰り返し実行する．do-while ループも break で中断できる．

例題 7.30 正の整数 n が入力されるまで，入力を繰り返すプログラムを作成せよ．

解答例

```
#include <stdio.h>
main()
{
   int n;

   do {
      printf("Enter n\n");
      scanf("%d",&n);
   } while (n <= 0);

   printf("n = %d\n",n);
}
```

7.12.5 for

for 文は，繰り返し回数がわかっている場合に適した繰り返しの構文である．for は次のように書かれる．

for (*initialization*; *condition*; *process*)
{
    statements
}

for 文では，*initialization* にて，ループの制御変数を初期化し，*condition* で継続条件を書き，*process* で，制御変数を変更する．*condition* が成立する限り，*statements* を繰り返す．

例題 7.31 n を正整数とする．1 から n までの和を求めるプログラムを作成せよ．

解答例

```
#include <stdio.h>
main()
{
   int n,s=0,i;
   printf("Enter n\n");
   scanf("%d",&n);
   for (i = 1; i <= n ; ++i)
      s += i;
   printf("s = %d\n",s);
}
```

++i は i = i+1 に等しい．また，s += i は s = s+i と同じである．

for も break によっていつでも中断できる．また，while および for 文で，ある条件が成立した場合，残りの処理をとばして，次の繰り返しを行いたいとき，continue 文を使う．

例題 7.32 整数

$$a_1, a_2, \ldots, a_n$$

を入力して，$a_i \geq 0$ の要素の和を計算するプログラムを作成せよ．

解答例

```
#include <stdio.h>
main()
{
   int n,sum=0,i,a;
   printf("Enter n\n");
   scanf("%d",&n);
   for (i = 1; i <= n ; ++i)
   {
      printf("a%1d = ",i); scanf("%d",&a);
      if (a < 0)
         continue;
      sum += a;
    }
   printf("sum = %d\n",sum);
}
```

7.12.6 goto とラベル

C には，BASIC や FORTRAN と同様，ジャンプ命令である goto 文とジャンプ先を示す**ラベル**が用意されている．使用例を一つ示せば goto 文の使い方をすぐ覚えてしまうだろう．次の例は，整数の入力と積算を 0 が入力されるまで繰り返すプログラムである．

例題 7.33

```
#include <stdio.h>
main()
{
   int a,s=0;
   aaa:
   printf("Enter an integer\n");
   scanf("%d",&a);
   s += a;
   printf("s = %d\n",s);
   if (a != 0)
     goto aaa;
}
```

初心者はつい簡単な goto 文を使いがちであるが，ジャンプを多用するとプログラムがスパゲッティ化して読みづらいものになるので，goto 文は控えめに使用する．実際，goto 文を用いないプログラミングは常に可能である．しかしながら，多重ループ内で，ある条件が成立した場合，いっきにそれらのループを抜けるようなときには，goto 文が適している．

7.13 便利な演算子

7.13.1 インクリメント演算子とデクリメント演算子

++i は i = i+1 に等しいことを述べた．++を**インクリメント演算子**という．同様に，--i は i= i-1 に等しい．--を**デクリメント演算子**という．

これらの演算子は，変数の前後どちらかに付けることができ，演算の順序がこれに従う．++i は，i+1 を置くということであり，i++は，i を置き，これに 1 を加えるということである．単に変数を 1 だけ増加または減少させるのであれば，演算子をどちらに付けても同じである．例えば

```
for(i = 0; i <= 10; ++i)  と  for(i = 0; i <= 10; i++)
```

は同じ働きをする．一方

```
i = 0;
x = i++;
```

とした場合，x に 0 が代入され，その後，i が 1 となる．また

```
i = 0;
x = ++i;
```

では，まず，i が 1 となり，それから，x に 1 が代入される．

7.13.2　代入演算子

ある変数に対して行った演算結果を元の変数と置き換えることがよくある．例えば

```
s = s + a;
```

という計算である．すでに述べたように，この式は次のように簡略化できる．

```
s += a;
```

差，積，商の演算についても，同様の**代入演算子**が用意されている（表 7.7 参照）．

表 7.7: 代入演算子

演算例	簡略形
`s = s+a`	`s += a`
`s = s-a`	`s -= a`
`s = s*a`	`s *= a`
`s = s/a`	`s /= a`

第8章　技術計算プログラミングの基礎

8.1　初歩的なプログラミング

これから与えられる一連の例題と演習問題を通じて，基本的なプログラミング技術を習得しよう．まず，もっとも初歩的な例題から導入を行う．以下の二つの例題はすでに説明した範囲であるが，基本的な事項を確認しておこう．

例題 8.1 実数 a, b を入力し

$$x_1 = a + b, \quad x_2 = a - b, \quad x_3 = a \times b, \quad x_4 = \frac{a}{b} \tag{8.1}$$

を計算して出力せよ．

解答例

```
#include <stdio.h>
main()
{
   double a,b,x1,x2,x3,x4;

   printf("Enter a b\n");
   scanf("%lf %lf",&a,&b);
   x1 = a+b;  x2 = a-b;  x3 = a*b;  x4 = a/b;
   printf("x1 = %f  x2 = %f  x3 = %f  x4 = %f\n",x1,x2,x3,x4);
}
```

実行例

```
Enter a b
3.0 4.0
x1 = 7.000000  x2 = -1.000000  x3 = 12.000000  x4 = 0.750000
```

基本演算子 '+', '-', '*', '/' と代入演算子 '=' の使い方の確認である．scanf 関数で，入力変数のアドレスを引数として渡すことに注意する．

例題 8.2 実数 a, b を入力し，$b \neq 0$ ならば

$$x_4 = \frac{a}{b}$$

を計算して出力し，$b = 0$ ならば，a, b の入力に戻るプログラムを作成せよ．

解答例

```
#include <stdio.h>
main()
{
   double a,b,x4;

   label_1:
   printf("Enter a b\n");
   scanf("%lf %lf",&a,&b);
   if (b == 0.0)
      goto label_1;
   x4 = a/b;
   printf("x4 = %f\n",x4);
}
```

（注意）計算機内部で正確に表現できる実数は限られていて，ほとんどの場合，近似的にしか扱うことができない．よって，`if (b == x)` という文は，一般には書けない．そのような場合，`if (b == x)` の代わりに

> `if (fabs(b-x) < eps)`（eps は小さな正数）
> `fabs(b-x)` は $|b-x|$ を求める関数である（`#include <math.h>` が必要）

を用いる．

例題 8.3 与えられる正数 n, p に対して，次式を計算せよ．

$$y = \sum_{i=1}^{n} i^p = 1^p + 2^p + 3^p + \cdots + n^p \tag{8.2}$$

解答例

```
#include <stdio.h>
#include <math.h>
main()
{
   int n,p,i;
   double y=0.0;

   printf("Enter n p\n");
   scanf("%d %d",&n,&p);
   for (i = 1; i <= n; ++i)
      y += pow(i,p);
```

```
    printf("y = %f\n",y);
}
```

実行例

```
Enter n p
5 2
y = 55.000000
```

pow(x,a) は x^a を求める関数（定義ヘッダは math.h）である．引数 x，a の型は double であるが，ここでは，int 型の i，p を渡している（代入の際には，型が自動変換される）．

演習 8.1 例題 8.2 を goto 文を用いないでプログラムせよ．

演習 8.2

$$y_i = \sum_{j=1}^{i} j = 1 + 2 + \cdots + i \tag{8.3}$$

すなわち

$$\begin{aligned} y_1 &= 1 \\ y_2 &= 1 + 2 \\ y_3 &= 1 + 2 + 3 \\ &\vdots \end{aligned}$$

とする．

$$z = \sum_{i=1}^{p} y_i = y_1 + y_2 + \cdots + y_p \tag{8.4}$$

を計算するプログラムを作成せよ．また，$p = 10$ に対する z を計算せよ．
（答）$z = 220$

8.2 ベクトルの計算

例題 8.4 2 次元ベクトル

$$\boldsymbol{a} = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

に対して

$$\text{和} \quad \boldsymbol{x} = \boldsymbol{a} + \boldsymbol{b} = \begin{bmatrix} a_1 + b_1 \\ a_2 + b_2 \end{bmatrix}, \quad \text{差} \quad \boldsymbol{y} = \boldsymbol{a} - \boldsymbol{b} = \begin{bmatrix} a_1 - b_1 \\ a_2 - b_2 \end{bmatrix}$$

内積　$z = \boldsymbol{a} \cdot \boldsymbol{b} = a_1 b_1 + a_2 b_2$

を計算するプログラムを作れ．そして，次の $\boldsymbol{a}, \boldsymbol{b}$ について計算してみよ．

$$\boldsymbol{a} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

解答例

```
#include <stdio.h>
#define N 100
main()
{
   int i;
   double a[N],b[N],x[N],y[N],z;

   printf("Enter vector a\n");
   for (i=1; i<=2; ++i)
      scanf("%lf",&a[i]);
   printf("Enter vector b\n");
   for (i=1; i<=2; ++i)
      scanf("%lf",&b[i]);

   z = 0.0;
   for (i = 1; i <= 2; ++i){
       x[i] = a[i] + b[i];
       y[i] = a[i] - b[i];
       z += a[i]*b[i];
   }
   printf("Vector x\n");
   for (i = 1; i <= 2; ++i)
      printf(" %10.4f ",x[i]);
   printf("\n");
   printf("Vector y\n");
   for (i = 1; i <= 2; ++i)
      printf(" %10.4f ",y[i]);
   printf("\n");
   printf("z = %10.4f\n",z);
}
```

実行結果

```
Enter vector a
```

```
1
2
Enter vector b
3
4
Vector x
     4.0000      6.0000
Vector y
    -2.0000     -2.0000
z =    11.0000
```

n 次元ベクトル $\boldsymbol{a}$ の大きさを 1 としたものを $\boldsymbol{a}$ の**正規化ベクトル**といい，$\hat{\boldsymbol{a}}$ で表す．すなわち

$$\hat{\boldsymbol{a}} = \frac{\boldsymbol{a}}{\| \boldsymbol{a} \|} = \begin{bmatrix} a_1/\| \boldsymbol{a} \| \\ a_2/\| \boldsymbol{a} \| \\ \vdots \\ a_n/\| \boldsymbol{a} \| \end{bmatrix} \tag{8.5}$$

ただし，$\| \boldsymbol{a} \|$ は $\boldsymbol{a}$ の**ノルム**であり，次式で計算される．

$$\| \boldsymbol{a} \| = \sqrt{a_1^2 + a_2^2 + \cdots + a_n^2} \tag{8.6}$$

演習 8.3 n 次元ベクトル

$$\boldsymbol{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

の和，差，内積を求めるプログラムを作成せよ．また，次の $\boldsymbol{a}, \boldsymbol{b}$ について計算せよ．

$$\boldsymbol{a} = \begin{bmatrix} 1 \\ 2 \\ 4 \\ 6 \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} 3 \\ 1 \\ -2 \\ -1 \end{bmatrix}$$

演習 8.4 n 次元ベクトル $\boldsymbol{a}$ のノルムと正規化ベクトルを求めるプログラムを作成せよ．また，次の $\boldsymbol{a}$ について計算せよ．

$$\boldsymbol{a} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \end{bmatrix}^T$$

ヒント：$\sqrt{x}$ は `sqrt(x)`（定義ヘッダは `math.h`）で計算できる．
（答）

$$\| \boldsymbol{a} \| = 7.4162$$

$$\hat{\boldsymbol{a}} = \begin{bmatrix} 0.1348 & 0.2697 & 0.4045 & 0.5394 & 0.6742 \end{bmatrix}^T$$

8.3　行列の和・差・積

行列を扱う第一歩は，行列の入出力法を知ることである．そこで，次の例題を検討しよう．

例題 8.5 $n \times m$ 行列 $\boldsymbol{A}(n \times m)$ を入力して，出力するプログラムを作成せよ．また，プログラム動作を次の行列で確かめよ．

$$\boldsymbol{A} = \begin{bmatrix} 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \\ 3 & 6 & 9 & 12 \end{bmatrix}$$

解答例

```
#include <stdio.h>
#define N 100
main()
{
   int i,j,n,m;
   double a[N][N];

   printf("Enter n m\n");
   scanf("%d %d",&n,&m);
   printf("Enter matrix A\n");
   for (j = 1; j <= m; ++j){
      printf("Column %d\n",j);
      for (i = 1; i <= n; ++i)
         scanf("%lf",&a[i][j]);
   }

   printf("Matrix A\n");
   for (i = 1; i <= n; ++i){
      for (j = 1; j <= m; ++j)
         printf(" %10.4f ",a[i][j]);
      printf("\n");
   }
}
```

実行例

```
Enter n m
3 4
Enter matrix A
```

```
Column 1
1
2
3
Column 2
4
5
6
Column 3
7
8
9
Column 4
10
11
12
Matrix A
     1.0000      4.0000      7.0000     10.0000
     2.0000      5.0000      8.0000     11.0000
     3.0000      6.0000      9.0000     12.0000
```

配列は，宣言時に次のように初期化できる．例えば，上の行列を a に設定する場合以下のように書く．

例題 8.6

```
#include <stdio.h>
#define N 100
main()
{
   int i,j,n=3,m=4;
   double a[N][N]=
      {{0,0,0,0,0},{0,1,4,7,10},{0,2,5,8,11},{0,3,6,9,12}};

   printf("Matrix A\n");
   for (i = 1; i <= n; ++i){
      for (j = 1; j <= m; ++j)
         printf(" %10.4f ",a[i][j]);
      printf("\n");
   }
}
```

配列の添字は 0 から始まるので，実際には，行列

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 4 & 7 & 10 \\ 0 & 2 & 5 & 8 & 11 \\ 0 & 3 & 6 & 9 & 12 \end{bmatrix}$$

を設定していることに注意する．同様に，ベクトル

$$\boldsymbol{b} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \end{bmatrix}^T$$

の要素をそれぞれ

```
b[1], b[2], b[3], b[4], b[5]
```

に設定する場合，次のように書く．

例題 8.7

```
#include <stdio.h>
#define N 100
main()
{
   int i,n=5;
   double b[N]={0,1,2,3,4,5};

   printf("Vector b^T\n");
   for (i = 1; i <= n; ++i)
      printf(" %7.3f ",b[i]);
      printf("\n");
}
```

実行結果

```
Vector b^T
   1.000    2.000    3.000    4.000    5.000
```

例題 8.8 行列 $\boldsymbol{A}(n \times n)$, $\boldsymbol{B}(n \times n)$ を入力して

和　$\boldsymbol{X} = \boldsymbol{A} + \boldsymbol{B}$,　差　$\boldsymbol{Y} = \boldsymbol{A} - \boldsymbol{B}$,　積　$\boldsymbol{Z} = \boldsymbol{AB}$

を計算するプログラムを作成せよ．また，次の行列について計算してみよ．

$$\boldsymbol{A} = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}, \quad \boldsymbol{B} = \begin{bmatrix} 2 & -1 & 1 \\ 1 & -1 & 2 \\ 0 & -2 & 1 \end{bmatrix}$$

解答例

```
#include <stdio.h>
#define N 100
main()
{
   int n,i,j,k;
   double a[N][N],b[N][N],x[N][N],y[N][N],z[N][N];

   printf("Enter n\n");
   scanf("%d",&n);
   printf("Enter matrix A\n");
   for (j = 1; j <= n; ++j){
      printf("Column %d\n",j);
      for (i = 1; i <= n; ++i)
         scanf("%lf",&a[i][j]);
   }

   printf("Enter matrix B\n");
   for (j = 1; j <= n; ++j){
      printf("Column %d\n",j);
      for (i = 1; i <= n; ++i)
         scanf("%lf",&b[i][j]);
   }

   for (i = 1; i <= n; ++i)
      for (j = 1; j <= n; ++j){
        x[i][j] = a[i][j] + b[i][j];
        y[i][j] = a[i][j] - b[i][j];
        z[i][j] = 0.0;
        for (k = 1; k <= n; ++k)
           z[i][j] += a[i][k]*b[k][j];
      }

   printf("Matrix A + B\n");
   for (i = 1; i <= n; ++i){
      for (j = 1; j <= n; ++j)
         printf(" %7.3f ",x[i][j]);
      printf("\n");
   }
   printf("Matrix A - B\n");
```

```
    for (i = 1; i <= n; ++i){
       for (j = 1; j <= n; ++j)
          printf(" %7.3f ",y[i][j]);
       printf("\n");
    }
    printf("Matrix AB\n");
    for (i = 1; i <= n; ++i){
       for (j = 1; j <= n; ++j)
          printf(" %7.3f ",z[i][j]);
       printf("\n");
    }
}
```

実行例

```
Enter n
3
Enter matrix A
Column 1
1
2
3
Column 2
4
5
6
Column 3
7
8
9
Enter matrix B
Column 1
2
1
0
Column 2
-1
-1
-2
Column 3
1
```

```
2
1
Matrix A + B
   3.000    3.000    8.000
   3.000    4.000   10.000
   3.000    4.000   10.000
Matrix A - B
  -1.000    5.000    6.000
   1.000    6.000    6.000
   3.000    8.000    8.000
Matrix AB
   6.000  -19.000   16.000
   9.000  -23.000   20.000
  12.000  -27.000   24.000
```

上のプログラムでは，計算効率を考えて，行列の和，差，積の計算を同時に行っているが，次のように，和・差の計算部と積の計算部を分けて書けば，プログラムがわかりやすくなるだろう．

```
for (i = 1; i <= n; ++i)
   for (j = 1; j <= n; ++j){
     x[i][j] = a[i][j] + b[i][j];
     y[i][j] = a[i][j] - b[i][j];
   }
for (i = 1; i <= n; ++i)
   for (j = 1; j <= n; ++j){
     z[i][j] = 0.0;
     for (k = 1; k <= n; ++k)
        z[i][j] += a[i][k]*b[k][j];
   }
```

このあたりの行列計算でつまずくと，次節以降の説明はほとんど理解不能となるので，上記プログラムをよく理解されたい．わかりにくい場合は，2×2 や 3×3 行列の具体例で，`for` 文の繰り返しを手計算でトレースしてみることをお勧めする．

演習 8.5 行列 $\boldsymbol{A}(n\times p)$, $\boldsymbol{B}(p\times m)$ を入力して，積 $\boldsymbol{AB}$ を計算するプログラムを作成せよ．また，次の具体例について計算してみよ．

$$\boldsymbol{A}=\begin{bmatrix} 1 & 2 & 3 \\ 1 & 1 & -1 \end{bmatrix},\quad \boldsymbol{B}=\begin{bmatrix} 2 & 1 \\ 5 & -1 \\ 4 & 0 \end{bmatrix}$$

（答）

$$\boldsymbol{AB} = \begin{bmatrix} 24 & -1 \\ 3 & 0 \end{bmatrix}$$

8.4　種々の行列の作り方

8.4.1　零行列・単位行列・対角行列・Jordan 標準形

$n \times m$ の**零行列** a[i][j] は，プログラム中で次のように書けば作れる．もちろん，必要な変数の宣言は行われているものとする．

```
for (i = 1; i <= n; ++i)
  for (j = 1; j <= m; ++j)
     a[i][j] = 0.0;
```

$n \times n$ の**単位行列**（n 次の単位行列）は，例えば，次のようにプログラムすれば得られる．

```
for (i = 1; i <= n; ++i){
  for (j = 1; j <= n; ++j)
     a[i][j] = 0.0;
  a[i][i] = 1.0;
}
```

配列の初期化が必要であることを忘れないようにする．

対角行列も同様に作れる．次の例は，**対角行列**

$$\boldsymbol{D} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 9 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{bmatrix}$$

を設定して，出力するプログラムである．

例題 8.9

```
#include <stdio.h>
#define N 100
main()
{
   int n=5,i,j;
   double a[N] = {0,1,9,4,4,2},d[N][N];
```

```
   for (i = 1; i <= n; ++i){
      for (j = 1; j <= n; ++j)
        d[i][j] = 0.0;
      d[i][i] = a[i];
   }

   printf("Matrix D\n");
   for (i = 1; i <= n; ++i){
      for (j = 1; j <= n; ++j)
         printf(" %7.3f ",d[i][j]);
      printf("\n");
   }
}
```

例題 8.10 i 次の **Jordan 細胞** $\boldsymbol{J}_i(\lambda)$ は，次のように定義される．

$$\boldsymbol{J}_1(\lambda) = \lambda, \quad \boldsymbol{J}_2(\lambda) = \begin{bmatrix} \lambda & 1 \\ 0 & \lambda \end{bmatrix}, \quad \boldsymbol{J}_3(\lambda) = \begin{bmatrix} \lambda & 1 & 0 \\ 0 & \lambda & 1 \\ 0 & 0 & \lambda \end{bmatrix}$$

正整数 n に対して，$\boldsymbol{J}_n(\lambda)$ を作るプログラムを作成せよ．

解答例

```
#define N 100
main()
{
   int n,i,j;
   double J[N][N],lambda;

   printf("Enter n\n");
   scanf("%d",&n);
   printf("Enter lambda\n");
   scanf("%lf",&lambda);

   for (i = 1; i <= n; ++i){
      for (j = 1; j <= n; ++j)
         J[i][j] = 0.0;
      J[i][i] = lambda;
   }
   for (i = 1; i < n; ++i)
      J[i][i+1] = 1.0;
```

```
    printf("Matrix J\n");
    for (i = 1; i <= n; ++i){
       for (j = 1; j <= n; ++j)
          printf(" %7.3f ",J[i][j]);
       printf("\n");
    }
}
```

実行例

```
Enter n
4
Enter lambda
-0.5
Matrix J
  -0.500    1.000    0.000    0.000
   0.000   -0.500    1.000    0.000
   0.000    0.000   -0.500    1.000
   0.000    0.000    0.000   -0.500
```

`J[i][i+1] = 1.0;` のように配列の中に式を書くことができる.

8.4.2　計算で要素が決定できる行列

例題 8.11 正整数 n に対して，次の $1 \times n$ 行列（すなわちベクトル）を作るプログラムを作成せよ.

$$\boldsymbol{x} = \left[\begin{array}{cccc} 1 & 2 & \ldots & n \end{array} \right]$$

解答例

例題 8.12

```
#include <stdio.h>
#define N 100
main()
{
   int n,i;
   double x[N];

   printf("Enter n\n");
   scanf("%d",&n);
```

```
   for (i=1; i<=n; ++i)
      x[i] = i;

   printf("Vector x\n");
   for (i = 1; i <= n; ++i)
         printf(" %7.3f ",x[i]);
   printf("\n");
}
```

実行例

```
Enter n
6
Vector x
   1.000    2.000    3.000    4.000    5.000    6.000
```

例題 8.13 次の規則で定義される行列 $\boldsymbol{A}_i$, $i = 1, 2, \ldots$ がある.

$$\boldsymbol{A}_1 = 1, \quad \boldsymbol{A}_2 = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad \boldsymbol{A}_3 = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

正整数 n が与えられたとき, $\boldsymbol{A}_n$ を求めるプログラムを作成せよ.

解答例

例題 8.14

```
#include <stdio.h>
#define N 100
main()
{
   int n,i,j,k;
   double a[N][N];

   printf("Enter n\n");
   scanf("%d",&n);

   k = 0;
   for (i = 1; i <= n; ++i)
      for (j = 1; j <= n; ++j){
         ++k;
         a[i][j] = k;
```

```
        }

    printf("Matrix A\n");
    for (i = 1; i <= n; ++i){
        for (j = 1; j <= n; ++j)
            printf(" %7.3f ",a[i][j]);
        printf("\n");
    }
}
```

`#define N 100` としているので，$n < 100$ である．もし，これ以上の n を入力するのであれば，`N` を大きくする．

演習 8.6 行列 $\boldsymbol{A}_i$ が次の規則で定義されている．

$$\boldsymbol{A}_1 = 1, \quad \boldsymbol{A}_2 = \begin{bmatrix} 1 & 2 \\ 0 & 3 \end{bmatrix}, \quad \boldsymbol{A}_3 = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 0 & 0 & 6 \end{bmatrix}$$

正整数 n が与えられたとき，$\boldsymbol{A}_n$ を求めるプログラムを作成せよ．

8.5　数学的関数の計算

8.5.1　多項式

例題 8.15 正整数 n と実数 x が与えられたとき

$$f(x) = 1 + x + x^2 + \cdots + x^n \tag{8.7}$$

を計算するプログラムを作成せよ．そして，$n = 4, x = 2.0$ に対する $f(x)$ を計算せよ．

解答例

```
#include <stdio.h>
#define N 100
main()
{
    int n,i;
    double f,x,z;

    printf("Enter n x\n");
    scanf("%d %lf",&n,&x);
    f = 1.0; z = 1.0;
    for (i = 1; i <= n; ++i){
```

```
        z *= x;
        f += z;
    }
    printf("f = %10.4f\n",f);
}
```

実行結果

```
Enter n x
4 2.0
f =    31.0000
```

例題 8.16 係数 $a_0, a_1, \ldots, a_n$ が与えられたとき，次の多項式

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n \tag{8.8}$$

を計算するプログラムを作成せよ．そして

$$f(x) = 5 - 20x + 20x^2 - x^4, \quad x = 2.0$$

の値を計算せよ．

解答例

```
#include <stdio.h>
#define N 100
main()
{
    int n,i;
    double a[N],f,x,z;

    printf("Enter n\n");
    scanf("%d",&n);
    for (i = 0; i <= n; ++i){
        printf("Enter a[%1d]\n",i);
        scanf("%lf",&a[i]);
    }
    printf("Enter x\n");
    scanf("%lf",&x);
    f = a[0]; z = 1.0;
    for (i = 1; i <= n; ++i){
        z *= x;
        f += a[i]*z;
    }
```

```
   printf("f = %10.4f\n",f);
}
```

実行結果

```
Enter n
4
Enter a[0]
5
Enter a[1]
-20
Enter a[2]
20
Enter a[3]
0
Enter a[4]
-1
Enter x
2
f =    29.0000
```

8.5.2　級数

例題 8.17 実数 x に対して，次の級数を計算するプログラムを作成せよ．ただし，ϵ を与えた正数とするとき，新たに加える項の絶対値が ϵ より小さくなるまで計算するものとする．

$y = 1 + x + x^2/2! + x^3/3! + \cdots$　（$x = 2$ のとき $y \to 7.3891$）

解答例

```
#include <stdio.h>
#include <math.h>
main()
{
   int i;
   double y,x,z,eps=1e-7;

   printf("Enter x\n");
   scanf("%lf",&x);
   y = 1.0; z = 1.0; i=0;
   while (fabs(z) > eps){
```

```
        ++i;
        z *= x/(double)i;
        y += z;
    }
    printf("y = %10.4f\n",y);
}
```

階乗はすぐに大きな数となり，オーバーフローしてしまうので，上記のように逐次 i で割ることによって，項の計算を行うようにする．

8.5.3 数学関数

すでに見たように，C には，pow，sqrt，fabs などの数学関数が用意されている．

```
y = sqrt(x);
```

というように使う．これらの関数の定義ヘッダは math.h である．関数および引数の型は大抵 double である．詳しくは，C のマニュアルを参照されたい．代表的な数学関数を表 2.3 に示す．

8.5.4 乱数

シミュレーションや探索において，乱数を利用したいことがしばしばある．基本となるのは，$[0,1]$ の実数乱数である．これから，種々の乱数を得るのは容易である．

例題 8.18 $[0,1]$ の範囲の実数乱数を発生させるプログラムを作成せよ．

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
main()
{
    double x;
    srand(time(NULL));
    x = (double)rand()/(double)RAND_MAX;
    printf("x = %f\n",x);
}
```

実行例

```
x = 0.681902
```

rand は $0 \sim$ RAND_MAX $= 2^{31} - 1$ の範囲の整数をランダムに発生させる関数である．time で時刻（秒）を呼び出し，srand でこの時刻を乱数の種として与えることで，実行する度に乱数の系列を変えている．これらの関数と定義ヘッダを表 8.1 に示す．

表 8.1: 乱数発生で用いた関数と定義ヘッダ

関数	定義ヘッダ
`time`	`time.h`
`rand`	`stdlib.h`
`srand`	`stdlib.h`

演習 8.7 正整数 n が与えられたとき，n の階乗

$$n! = 1 \cdot 2 \cdot 3 \cdots n$$

を計算するプログラムを作成せよ．
（ヒント）次のプログラムが使える．

```
y = 1;
for (i = 1; i <= n; ++i)
   y *= i;
```

演習 8.8 実数 x に対して，次の級数を計算するプログラムを作成せよ．ただし，ϵ を与えた正数とするとき，新たに加える項の絶対値が ϵ より小さくなるまで計算するものとする．

(1) $y = 1 - x^2/2! + x^4/4! - + \cdots$ （$x = 2$ のとき $y \to -0.4161$）

(2) $y = x - x^3/3! + x^5/5! - + \cdots$ （$x = 2$ のとき $y \to 0.9093$）

演習 8.9 n を正整数とする．**フィボナッチ数列**

$$1, 1, 2, 3, 5, 8, \ldots$$

（第 1 項 = 第 2 項 = 1 とし，第 i 項（$i \geq 3$）を第 $i-2, i-1$ 項の和により求める）
を第 n 項まで作るプログラムを作成せよ．

演習 8.10 実数 x と正整数 n が与えられたとき，ベクトル

$$\boldsymbol{y} = \left[\begin{array}{ccccc} 1 & x & x^2 & \ldots & x^n \end{array} \right]$$

を作るプログラムを作成せよ．

8.6　最大・最小・平均・ソーティング

例題 8.19 $[0, 1]$ の範囲の実数乱数を n 個発生させ，その中で最大の数を求めよ．

解答例

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define rnd (double)rand()/(double)RAND_MAX
#define N 10000
main()
{
   int i,n;
   double a[N],amax=0.0;
   srand(time(NULL));

   printf("Enter n\n");
   scanf("%d",&n);
   for (i = 1; i <= n; ++i){
      a[i] = rnd;  printf("a[%1d] = %f\n",i,a[i]);
   }

   for (i = 1; i <= n; ++i)
      if (a[i] > amax)
         amax = a[i];

   printf("amax = %f\n",amax);
}
```

実行例

```
Enter n
5
a[1] = 0.084650
a[2] = 0.828730
a[3] = 0.413215
a[4] = 0.113235
a[5] = 0.721209
amax = 0.828730
```

$[0,1]$ の実数乱数の生成法は例題 8.18 の方法を使っている．ただ，記述を簡単にするため，`#define` 文で

```
#define rnd (double)rand()/(double)RAND_MAX
```

と定義した．

例題 8.20 $[0,1]$ の実数乱数を n 個発生させ，最大値，最小値，平均値を求めよ．

解答例

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define rnd (double)rand()/(double)RAND_MAX
#define N 10000
main()
{
   int i,n;
   double a[N],amax=0.0,amin=1.0,s=0.0,average;
   srand(time(NULL));

   printf("Enter n\n");
   scanf("%d",&n);
   for (i = 1; i <= n; ++i){
      a[i] = rnd;  printf("a[%1d] = %f\n",i,a[i]);
   }

   for (i = 1; i <= n; ++i){
      s += a[i];
      if (a[i] > amax)
         amax = a[i];
      if (a[i] < amin)
         amin = a[i];
   }
   average = s/(double)n;
   printf("amax = %f  amin = %f  average = %f\n",amax,amin,average);
}
```

実行例

```
Enter n
10
a[1] = 0.135405
a[2] = 0.995631
a[3] = 0.333319
a[4] = 0.248480
a[5] = 0.615244
a[6] = 0.751862
a[7] = 0.873466
a[8] = 0.102956
```

```
a[9] = 0.192717
a[10] = 0.397001
amax = 0.995631  amin = 0.102956  average = 0.464608
```

例題 8.21 $[0,1]$ の実数乱数を n 個発生させ，大きい順に並べ替えよ．

解答例

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define rnd (double)rand()/(double)RAND_MAX
#define N 10000
main()
{
   int i,j,n,jmax;
   double a[N],amax,temp;
   srand(time(NULL));

   printf("Enter n\n");
   scanf("%d",&n);
   for (i = 1; i <= n; ++i){
      a[i] = rnd;  printf("a[%1d] = %f\n",i,a[i]);
   }

   for (i = 1; i <= n-1; ++i){
      amax = 0.0;
      for (j = i; j <= n; ++j)
         if (a[j]>amax){
            amax = a[j]; jmax = j;
         }
      temp = a[i];
      a[i] = a[jmax];  a[jmax] = temp;
   }

   for (i=1; i<=n; ++i)
      printf(" %f ",a[i]);
   printf("\n");

}
```

実行例

```
Enter n
5
a[1] = 0.746640
a[2] = 0.176880
a[3] = 0.653998
a[4] = 0.127963
a[5] = 0.422975
 0.746640  0.653998  0.422975  0.176880  0.127963
```

並べ替えの考え方は次のとおりである．まず

$$a_1,\, a_2,\, \ldots,\, a_n$$

の中で最大のものを求め，それを a_1 と入れ替える．つぎに

$$a_2,\, a_3,\, \ldots,\, a_n$$

の中で最大のものを求め，それを a_2 と入れ替える．以下同様である．ここで使用したアルゴリズムは**クイックソートアルゴリズム**と呼ばれる．

演習 8.11 $[0,1]$ の実数乱数を n 個発生させ，小さい順に並べ替えよ．

8.7　代数方程式の根

実係数の**代数方程式**を解く手法は数多く提案されているが，ここでは，条件の悪い問題に対しても信頼度が高いとされる解法の一つである **DKA（Durand-Kerner-Aberth）法**を説明する．この方法は，与えられた代数方程式を，2 次式に分解し，逐次次数を下げていく各種手法と違い，すべての解を同時に求める連立法で，ほとんどすべての初期値に対して解に 2 次収束することが知られている．

代数方程式は次式で表される．

$$f(x) = a_0 x^n + a_1 x^{n-1} + a_2 x^{n-2} + \cdots + a_{n-1} x + a_n = 0 \tag{8.9}$$

一般性を失うことなく

$$a_0 = 1 \tag{8.10}$$

とする．

DKA 法 初期値 x_i^0, $i = 1 \sim n$ が与えられたとき，$k = 0, 1, 2, \ldots$ に対して次の反復計算をせよ.

$$x_i^{k+1} = x_i^k - \frac{f(x_i^k)}{\prod_{j=1(j\neq i)}^{n} (x_i^k - x_j^k)}, \quad i = 1, 2, \ldots, n \tag{8.11}$$

ここで，もし，ある根 x_j^k が $x_i^k = x_j^k$ となれば，x_i^k は収束したものとみなし，以後

$$x_i^{k+1} = x_i^k$$

とおく．反復の終了基準は

$$\left|x_i^{k+1} - x_i^k\right| < \epsilon, \quad i = 1, 2, \ldots, n \tag{8.12}$$

とする．ただし，$\epsilon > 0$ は許容誤差を表す.

式 (8.11) の x_i^k, x_j^k は複素数であることに注意する．ここで，複素数の演算をまとめておく.

複素数演算 複素数 f, x

$$f = f_1 + jf_2, \quad x = x_1 + jx_2, \quad j = \sqrt{-1} \tag{8.13}$$

の和，差，積，商は次式で計算される.

$$f + x = f_1 + x_1 + j(f_2 + x_2) \tag{8.14}$$

$$f - x = f_1 - x_1 + j(f_2 - x_2) \tag{8.15}$$

$$fx = f_1x_1 - f_2x_2 + j(f_2x_1 + f_1x_2) \tag{8.16}$$

$$\frac{f}{x} = \frac{f_1x_1 + f_2x_2}{x_1^2 + x_2^2} + j\frac{f_2x_1 - f_1x_2}{x_1^2 + x_2^2} \tag{8.17}$$

例題 8.22 複素数 x_i に対して $f(x_i)$ を計算するプログラムを作成せよ.

解答例

```
f1 = 1.0;
f2 = 0.0;
for (j = 1;j <= n; ++j){
   w1 = f1*x1[i]-f2*x2[i];
   w2 = f2*x1[i]+f1*x2[i];
   f1 = w1+a[j];
```

```
    f2 = w2;
}
```

x1[i] は x_i の実部，x2[i] は虚部である．f1，f2 も同様．上のプログラムでは，$f(x)$ を

$$f(x) = x(\cdots x(x(x + a_1) + a_2) + \cdots) + a_n \tag{8.18}$$

と計算している．

例題 8.23 複素数 x_i, x_j に対して

$$p = \prod_{j=1(j\neq i)}^{n} (x_i^k - x_j^k) \tag{8.19}$$

を計算するプログラムを作成せよ．

解答例

```
p1 = 1.0;
p2 = 0.0;
for (j = 1; j <= n; ++j){
   if (j == i)
      continue;
   w1 = p1*(x1[i]-x1[j])-p2*(x2[i]-x2[j]);
   w2 = p1*(x2[i]-x2[j])+p2*(x1[i]-x1[j]);
   p1 = w1;
   p2 = w2;
}
```

例題 8.24 DKA 法を用いて n 次代数方程式

$$f(x) = a_0x^n + a_1x^{n-1} + a_2x^{n-2} + \cdots + a_{n-1}x + a_n = 0$$

の根を求めるプログラムを作成せよ．そして，次の代数方程式の根を求めよ．

$$x^4 + 3x^3 + 4x^2 + 2x = 0 \tag{8.20}$$

（答）$-1 \pm j,\ -1,\ 0$

解答例

```
/* DKA method */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
```

```
#include <math.h>
#define N 100
#define rnd (double)rand()/(double)RAND_MAX
main()
{
   int i,j,n;
   double a[N],x1[N],x2[N],w0,w1,w2,f1,f2;
   double p1,p2,e,ae,a1,a2,eps=1e-8;

   printf("Enter n\n");
   scanf("%d",&n);
   for (i = 0 ;i <= n; ++i){
      printf("Enter a[%d] = ",i);
      scanf("%lf",&a[i]);
   }
   /* initial value of x */
   for (i = 1; i <= n; ++i){
      x1[i] = rnd;
      x2[i] = rnd;
   }

   w0 = a[0];
   for (i = 0; i <= n; ++i)
      a[i] /= w0;
   do {
      e = 0.0;
      for (i = 1; i <= n; ++i){
         /* bunsi */
         f1 = 1.0;
         f2 = 0.0;
         for (j = 1; j <= n; ++j){
            w1 = f1*x1[i]-f2*x2[i];
            w2 = f2*x1[i]+f1*x2[i];
            f1 = w1+a[j];
            f2 = w2;
         }
         /* bunbo */
         p1 = 1.0;
         p2 = 0.0;
         for (j = 1; j <= n; ++j){
```

```
            if (j == i)
               continue;
            w1 = p1*(x1[i]-x1[j])-p2*(x2[i]-x2[j]);
            w2 = p1*(x2[i]-x2[j])+p2*(x1[i]-x1[j]);
            p1 = w1;
            p2 = w2;
         }
         a1 = (f1*p1+f2*p2)/(p1*p1+p2*p2);
         a2 = (f2*p1-f1*p2)/(p1*p1+p2*p2);
         ae = sqrt(a1*a1+a2*a2);
         if (e < ae)
            e = ae;
         x1[i] -= a1;
         x2[i] -= a2;
      }
   } while (ae > eps);
   for (i = 1; i <= n; ++i)
   printf("x[%d] = %f + j%f\n",i,x1[i],x2[i]);
}
```

実行例

```
Enter n
4
Enter a[0] = 1
Enter a[1] = 3
Enter a[2] = 4
Enter a[3] = 2
Enter a[4] = 0
x[1] = -1.000000 + j-1.000000
x[2] = -1.000000 + j1.000000
x[3] = -1.000000 + j-0.000000
x[4] = -0.000000 + j-0.000000
```

初期値 x_i, $i = 1, 2, \ldots, n$ は乱数を利用して与えた．eps は許容誤差（$\epsilon = 10^{-8}$ とした）である．

演習 8.12 例題 8.24 のプログラムに DKA 法の繰り返し回数をカウントするプログラムを追加せよ．

演習 8.13 次の代数方程式の根を DKA 法によって求めよ．

$$(1)\quad x^5 + x^4 + x^3 + x^2 + x + 1 = 0 \tag{8.21}$$

（答）$-1,\ 0.5 \pm j0.86602540378444,\ -0.5 \pm j0.86602540378444$

(2) $x^4 + 6x^3 + 13x^2 + 12x + 4 = 0$ (8.22)

（答） $-1,\ -1,\ -2,\ -2$

(3) $x^4 + 4x^3 + 6x^2 + 4x + 1 = 0$ (8.23)

（答） $-1,\ -1,\ -1,\ -1$

8.8 行列のべき乗

$\boldsymbol{A}(n \times n)$ が与えられたとする．また，$\boldsymbol{I}$ を単位行列とする．次のアルゴリズムを考えよう．

1. $\boldsymbol{X} \leftarrow \boldsymbol{I}$
2. $\boldsymbol{Y} \leftarrow \boldsymbol{AX}$
3. $\boldsymbol{X} \leftarrow \boldsymbol{Y}$

これにより，$\boldsymbol{X} = \boldsymbol{A}$ が得られる．続けて，ステップ 2 と 3 を実行すれば，$\boldsymbol{X} = \boldsymbol{AA} = \boldsymbol{A}^2$ となる．すなわち，ステップ 2 と 3 を繰り返すことによって，$\boldsymbol{A}$ のべきが 1 ずつ上がることがわかる．このアルゴリズムを利用して，$\boldsymbol{A}^p$ を求めるプログラムを次のように作ることができる．

例題 8.25 行列 $\boldsymbol{A}(n \times n)$ と正整数 p を入力して $\boldsymbol{A}^p$ を計算するプログラムを作成せよ．また，次の具体例について計算してみよ．

$$\boldsymbol{A} = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix},\ p = 2$$

解答例

```
#include <stdio.h>
#include <stdlib.h>
#define N 100
main()
{
   int n,i,j,k,r,p;
   double a[N][N],x[N][N],y[N][N];

   printf("Enter n p\n");
   scanf("%d %d",&n,&p);
   printf("Enter matrix A\n");
   for (j = 1; j <= n; ++j){
      printf("Column %d\n",j);
```

```
      for (i = 1; i <= n; ++i)
         scanf("%lf",&a[i][j]);
   }

   for (i = 1; i <= n; ++i){
      for (j = 1; j <= n; ++j)
         x[i][j] = 0.0;
      x[i][i] = 1.0;
   }

   for (r = 1; r <= p; ++r){
      for (i = 1; i <= n; ++i)
         for (j = 1; j <= n; ++j){
            y[i][j] = 0.0;
            for (k = 1; k <= n; ++k)
               y[i][j] += a[i][k]*x[k][j];
         }
         for (i = 1; i <= n; ++i)
            for (j = 1; j <= n; ++j)
               x[i][j] = y[i][j];
   }

   printf("Matrix A^p\n");
   for (i = 1; i <= n; ++i){
      for (j = 1; j <= n; ++j)
         printf(" %10.4f ",x[i][j]);
      printf("\n");
   }
}
```

実行例

```
Enter n p
2 2
Enter matrix A
Column 1
1
2
Column 2
3
4
```

```
Matrix A^p
     7.0000     15.0000
    10.0000     22.0000
```

演習 8.14 $p = 0$ の場合，$\boldsymbol{A}^p = \boldsymbol{I}$ である．例 8.25 のプログラムを $p = 0, 1, 2, \ldots$ に対して適用できるように変更せよ．

演習 8.15 $p = 0, 1, 2, \ldots$ に対して

$$\boldsymbol{X} = \frac{\boldsymbol{A}^p}{p!}$$

を計算するプログラムを作成せよ．また，次の例について計算せよ．

$$\boldsymbol{A} = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}, \quad p = 5$$

（答）

$$\boldsymbol{X} = \begin{bmatrix} 8.9083 & 19.4750 \\ 12.9833 & 28.3833 \end{bmatrix}$$

演習 8.16 $\boldsymbol{A}(n \times n)$ の行列指数関数は次式で定義される．

$$\boldsymbol{e}^A := \boldsymbol{I} + \boldsymbol{A} + \frac{\boldsymbol{A}^2}{2!} + \frac{\boldsymbol{A}^3}{3!} + \cdots$$

$\boldsymbol{A}$ が与えられたとき，$\boldsymbol{e}^A$ を計算するプログラムを作成せよ．ただし，項の打ち切り基準を

$$\sum_{i=1}^{n}\sum_{j=1}^{n} |a_{ij}^{(k)}| < \epsilon = \texttt{1e-8}$$

とせよ．$a_{ij}^{(k)}$ は $\boldsymbol{A}^k/k!$ の (i, j) 要素である．また，次の行列について計算せよ．

$$\boldsymbol{A} = \begin{bmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & -1 \end{bmatrix}$$

（答）

$$\boldsymbol{e}^A = \begin{bmatrix} 0.3679 & 0.3679 & 0.1839 \\ 0 & 0.3679 & 0.3679 \\ 0 & 0 & 0.3679 \end{bmatrix}$$

8.9　連立一次方程式

8.9.1　Gauss の消去法を用いた求解関数

x_1, x_2 に関する方程式

$$\left.\begin{aligned} x_1 + a_{12}x_2 &= b_1 \\ x_2 &= b_2 \end{aligned}\right\} \tag{8.24}$$

は下の式から解けば容易に解が得られる．一般に，$\boldsymbol{A}(n \times n)$, $\boldsymbol{b}(n \times 1)$ が与えられたとき，n 元連立一次方程式

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \quad \boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \tag{8.25}$$

は，$\boldsymbol{A}$ が

$$\boldsymbol{A} = \begin{bmatrix} 1 & a_{12} & \cdots & a_{1n} \\ & 1 & \cdots & a_{2n} \\ & \mathbf{0} & \ddots & \vdots \\ & & & 1 \end{bmatrix} \tag{8.26}$$

のとき，下方の式から $x_n, x_{n-1}, \cdots, x_1$ を順に求めることができる．通常，$\boldsymbol{A}$ は対角要素が 1 の上三角行列とは限らないので，$\boldsymbol{A}$ をこの形に変換する．この変換は **Gauss の消去法**で行うことができる．

例題 8.26 $\boldsymbol{A}(n \times n)$, $\boldsymbol{b}(n \times 1)$ が与えられたとき，Gauss の消去法を用いて，n 元連立一次方程式

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \tag{8.27}$$

の解を求めるプログラムを作成せよ．また，次の具体例について計算せよ．

$$\boldsymbol{A} = \begin{bmatrix} 1 & 2 & 1 \\ -1 & 1 & -1 \\ 2 & 4 & 6 \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

解答例

```
#include <stdio.h>
#include <math.h>
#define N 100
main()
{
```

```
   int n,i,j;
   double a[N][N],h[N][N],x[N],b[N],y[N];
   int gauss();

   printf("Enter n\n");
   scanf("%d",&n);
   printf("Enter matrix A\n");
   for (j = 1; j <= n; ++j){
      printf("Column %d\n",j);
      for (i = 1; i <= n; ++i)
         scanf("%lf",&a[i][j]);
   }
   printf("Enter b\n");
   for (i = 1; i <= n; ++i)
      scanf("%lf",&b[i]);

   for (i = 1; i <= n; ++i)
      for (j = 1; j <= n; ++j)
         h[i][j] = a[i][j];

   for (i = 1; i <= n; ++i)
      h[i][n+1] = b[i];

   gauss(n,h,x);

   printf("Solution\n");
   for (i = 1; i <= n; ++i)
      printf("%10.4f\n",x[i]);
}

int gauss(n,h,xt)
int n;
double h[N][N],xt[N];
{
   int iv[N],i,j,k,l,ip,jp,temp;
   double x[N],hmax,dtemp,pivot,eps=1e-6;
   for (i = 1; i <= n; ++i)
      iv[i] = i;
   for (i = 1; i <= n; ++i){
      hmax = fabs(h[i][i]);  ip = i;  jp = i;
```

```
      for (j = i+1; j <= n; ++j)
         if (hmax<fabs(h[i][j])){
            hmax = fabs(h[i][j]);  jp = j;
         }
      pivot = h[ip][jp];
      if (fabs(pivot)<eps){
         printf("The matrix is singular.\n"); exit(0);
      }
      if (jp != ip){
         temp = iv[ip];  iv[ip] = iv[jp];  iv[jp] = temp;
         for (k=1; k<=n; ++k){
            dtemp = h[k][ip];  h[k][ip] = h[k][jp];
            h[k][jp] = dtemp;
         }
      }
      for (l = ip+1; l <= n+1; ++l)
         h[ip][l] /= pivot;
      for (k = ip+1; k <= n; ++k)
         for (l = ip+1; l <= n+1; ++l)
            h[k][l] -= h[k][ip]*h[ip][l];
   }

   h[n][n] = 0.0;
   for (i = 1; i <= n; ++i){
      x[n-i+1] = h[n-i+1][n+1];
      for (j = 1; j <= i-1; ++j)
         x[n-i+1] = x[n-i+1] - h[n-i+1][n-j+1]*x[n-j+1];
   }
   for (i = 1; i <= n; ++i)
      xt[iv[i]] = x[i];
   return(0);
}
```

実行結果

```
Enter n
3
Enter matrix A
Column 1
1
-1
```

```
2
Column 2
2
1
4
Column 3
1
-1
6
Enter b
1
1
1
Solution
   -0.0833
    0.6667
   -0.2500
```

$\boldsymbol{Ax} = \boldsymbol{b}$ の解を求める部分は，関数 gauss としてまとめた．gauss の中では，

$$\boldsymbol{H} = \left[\begin{array}{cc} \boldsymbol{A} & \boldsymbol{b} \end{array} \right] \tag{8.28}$$

という行列に対して，掃き出し法を適用した．**枢軸**には，$\boldsymbol{A}$ の各行で，最も絶対値が大きな要素を選んでいる．これは，掃き出し計算によって生じる誤差をできるだけ小さくするためである．枢軸変更による $\boldsymbol{A}$ の列の入れ替えに伴う $\boldsymbol{x}$ 要素の入れ替えを iv[i] で記憶している．最後に，iv[i] を用いて，入れ替えた $\boldsymbol{x}$ を元に戻すためである．また，対角要素は 1 で，下三角要素は 0 となることがわかっているので，それらに対する掃き出し計算を省略し，計算効率を上げている．

8.9.2 最小二乗法による多項式曲線のあてはめ

データ対 (x, y) が p 組与えられたとしよう．

$$(x_i, y_i), \quad i = 1, 2, \ldots, p \tag{8.29}$$

これらのデータを基にして，y を x の n 次多項式で近似することを考える（図 8.1 参照）．

データを近似する多項式を次のように表す．

$$\hat{y}_i = a_0 + a_1 x_i + a_2 x_i^2 + \cdots + a_n x_i^n \tag{8.30}$$

$$= \boldsymbol{X}_i^T \boldsymbol{a}, \quad i = 1, 2, \ldots, p \tag{8.31}$$

ただし

$$\boldsymbol{X}_i^T = \left[\begin{array}{ccccc} 1 & x_i & x_i^2 & \ldots & x_i^n \end{array} \right]$$

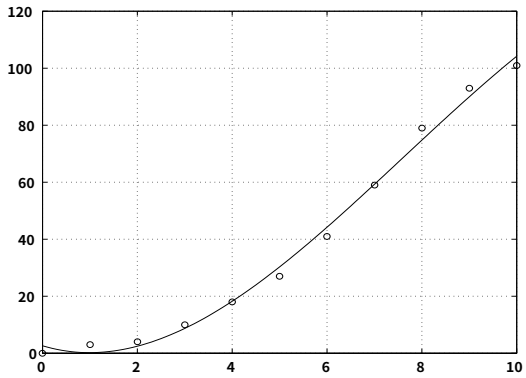

図 8.1: データ点に対する多項式曲線のあてはめ

$$\boldsymbol{a} = \left[\begin{array}{ccccc} a_0 & a_1 & a_2 & \ldots & a_n \end{array} \right]^T$$

とおいた．また

$$p \geq n + 1 \tag{8.32}$$

とする．多項式の係数 $\boldsymbol{a}$ は，真値と推定値との二乗誤差が最小となるように決定される．

$$\text{Minimize} \quad e = \sum_{i=1}^{p} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{p} (y_i - \boldsymbol{X}_i^T \boldsymbol{a})^2 \tag{8.33}$$

いま

$$\boldsymbol{X} := \left[\begin{array}{c} \boldsymbol{X}_1^T \\ \boldsymbol{X}_2^T \\ \vdots \\ \boldsymbol{X}_p^T \end{array} \right], \quad \boldsymbol{Y} := \left[\begin{array}{cccc} y_1 & y_2 & \ldots & y_p \end{array} \right]^T \tag{8.34}$$

を定義すると，式 (8.33) は次のように書ける．

$$e = (\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{a})^T (\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{a}) \tag{8.35}$$

式 (8.35) を最小にする $\boldsymbol{a}$ は，式 (8.35) を $\boldsymbol{a}$ で偏微分し $\boldsymbol{0}$ とおくことによって，次のように求められる．

$$\frac{\partial e}{\partial \boldsymbol{a}} = -2\boldsymbol{X}^T (\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{a}) = \boldsymbol{0} \tag{8.36}$$

式 (8.36) を解いて

$$\boldsymbol{a} = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{Y} \tag{8.37}$$

を得る．式 (8.37) が求める係数である．実際には，計算効率の点から，$\boldsymbol{a}$ は，線形方程式

$$(\boldsymbol{X}^T\boldsymbol{X})\boldsymbol{a} = \boldsymbol{X}^T\boldsymbol{Y} \tag{8.38}$$

の解として計算した方が有利である．

例題 8.27 上述の理論に従い，与えられた (x, y) データ

$$(x_i, y_i), \quad i = 1, 2, \ldots, p$$

を最小二乗近似する n 次多項式の係数を求めるプログラムを作成せよ．そして，次のデータについて，$n = 3$ とした場合を計算せよ．

$$\{x_i\} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$$

$$\{y_i\} = \{0, 3, 4, 10, 18, 27, 41, 59, 79, 93, 101\}$$

解答例

```
#include <stdio.h>
#include <math.h>
#define N 100
main()
{
   int p,n,n1,i,j,k;
   double a[N][N],h[N][N],x[N],b[N],X[N][N],Y[N];
   double XX[N][N],XY[N],z,XT[N][N];
   double xdata[N]
      = {0., 0., 1., 2., 3., 4., 5., 6., 7., 8., 9., 10.};
   double ydata[N]
      = {0., 0., 3., 4., 10., 18., 27., 41., 59., 79., 93., 101.};
   int gauss();
   p = 11;  /*  The number of data  */
   printf("Enter n\n");
   scanf("%d",&n);

   n1 = n + 1;
   for (i = 1; i <= p; ++i){
      z = 1.0;
      for (j = 1; j <= n1; ++j){
         X[i][j] = z;  z *= xdata[i];
      }
   }
```

```
   for (i = 1; i <= p; ++i)
      Y[i] = ydata[i];

   for (i = 1; i <= n1; ++i)
      for (j = 1; j <= p; ++j)
         XT[i][j] = X[j][i];

   for (i = 1; i <= n1; ++i){
      for (j = 1; j <= n1; ++j){
         XX[i][j] = 0.0;
         for (k = 1; k <= p; ++k)
            XX[i][j] += XT[i][k]*X[k][j];
      }
   }

   for (i = 1; i <= n1; ++i){
      XY[i] = 0.0;
      for (j = 1; j <= p; ++j)
         XY[i] += XT[i][j]*Y[j];
   }

   for (i = 1; i <= n1; ++i){
      h[i][n1+1] = XY[i];
      for (j = 1; j <= n1; ++j)
         h[i][j] = XX[i][j];
   }

   gauss(n1,h,x);

   printf("Solution a\n");
   for (i = 1; i <= n1; ++i)
      printf("a%1d = %8.4f\n",i-1,x[i]);
}

int gauss(n,h,xt)
```

上述のものと同じなので省略

実行結果

```
Enter n
```

```
3
Solution a
a0 =   2.6224
a1 =  -5.0000
a2 =   2.6987
a3 =  -0.1183
```

データ点と曲線を図 8.1 に示す.

演習 8.17 次のデータを最小二乗近似する $1 \sim 3$ 次多項式を求めよ．また，データと多項式曲線をグラフ表示せよ.

$$\{x_i\} = \{1, 2, 3, 4, 5, 6\}$$

$$\{y_i\} = \{0.3, 0.55, 1.02, 2.55, 5.67, 7.88\}$$

演習 8.18 行列 $\boldsymbol{A}(n \times n)$ が上三角行列のとき，すなわち

$$\boldsymbol{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ & a_{22} & \cdots & a_{2n} \\ & \mathbf{0} & \ddots & \vdots \\ & & & a_{nn} \end{bmatrix}$$

のとき，$\boldsymbol{A}$ の行列式は次式で計算される.

$$\det \boldsymbol{A} = a_{11}a_{22}\cdots a_{nn}$$

関数 gauss() を参考にして，$\det \boldsymbol{A}$ を計算するプログラムを作成せよ．ただし，列の入れ替えを行うと行列式の符号が逆になることに注意する.

演習 8.19 gauss() を利用して，正則行列 $\boldsymbol{A}(n \times n)$ の逆行列 $\boldsymbol{A}^{-1}$ を計算するプログラムを作成せよ．また，次の行列の逆行列を求めよ.

$$\boldsymbol{A} = \begin{bmatrix} -1 & 0 & 1 \\ 1 & -1 & -1 \\ -1 & 1 & 0 \end{bmatrix} \quad (\text{答})\boldsymbol{A}^{-1} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & -1 & 0 \\ 0 & -1 & -1 \end{bmatrix}$$

（ヒント）$\boldsymbol{Ax} = \boldsymbol{b}$ の解は，$\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}$ であることに注意する.

$$\boldsymbol{b} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

とすると，$\boldsymbol{x}$ として，$\boldsymbol{A}^{-1}$ の第 1 列が計算される．同様に

$$\boldsymbol{b} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

として方程式を解けば，$\boldsymbol{A}^{-1}$ の第 2 列が得られる．

8.10　ファイルに対する入出力

データをファイルに保存する方法として，これまで，Linux のリダイレクション機能（>）を利用したが，ここでは，C プログラムによる方法を説明する．

例題 8.28 次の (x_i, y_i) データをファイルに出力するプログラムを作成せよ．

$$\{x_i\} = \{1, 2, 3, 4, 5, 6\}$$

$$\{y_i\} = \{0.3, 0.55, 1.02, 2.55, 5.67, 7.88\}$$

解答例

```
#include <stdio.h>
#define N 100
main()
{
   char file1[N] = "data1.dat";
   int i,p=7;
   double xdata[N] = {0.,1.,2.,3.,4.,5.,6.};
   double ydata[N] = {0.,0.3,0.55,1.02,2.55,5.67,7.88};
   FILE *fw;

   fw = fopen(file1, "w");
   if (fw == NULL){
      printf("can't open %s\n",file1);
      exit(1);
   }
   for (i = 1; i <= p; ++i)
      fprintf(fw,"%f  %f\n",xdata[i],ydata[i]);
  fclose(fw);
}
```

これを実行すると，現在の作業ディレクトリに data1.dat という次の内容のファイルが作成される（すでに同じ名前のファイルが存在する場合，上書きされる）.

```
1.000000  0.300000
2.000000  0.550000
3.000000  1.020000
4.000000  2.550000
5.000000  5.670000
6.000000  7.880000
```

例題 8.28 の説明

1. 文字列 file1 にファイル名 data1.dat を代入する．p はデータ数である.
2. FILE *fw によりファイルポインタ fw を定義する.
3. fw = fopen(file1, "w") により file1 を書出しファイルとしてオープンする．ファイルを開けない場合，fopen の関数値が NULL = 0 となる.
4. fprintf 関数を用いてデータを出力する.
5. fclose(fw) により，オープンしているファイルを閉じる.

次にファイルからデータを読み込む例題を示そう.

例題 8.29 例題 8.28 で出力したファイルのデータを読み込むプログラムを作成せよ.

解答例

```
#include <stdio.h>
#define N 100
main()
{
   char file1[N] = "data1.dat";
   int i,p,s;
   double xdata[N],ydata[N];
   FILE *fr;

   fr = fopen(file1, "r");
   if (fr == NULL){
      printf("can't open %s\n",file1);
      exit(1);
   }
   i=0;
   do {
      ++i;
```

```
      s=fscanf(fr,"%lf  %lf\n",&xdata[i],&ydata[i]);
   } while (s != EOF);
   p = i-1;
   fclose(fr);

   for (i = 1; i <= p; ++i)
      printf("%f  %f\n",xdata[i],ydata[i]);
}
```

例題 8.29 の説明

1. `fr = fopen(file1, "r")` で `file1` を読込みファイルとしてオープンする．
2. データ数は未知として，`do while` 文の継続基準を `s != EOF` とした．`EOF = -1` である．これによりファイルの終わりまで読込むと同時にデータ数を `i` でカウントする．データ数より一つ多く `i` をカウントして `do` ループが終わるので，`p = i-1` としてデータ数 `p` を計算する．
3. 最後の表示は確認のためのものである．

8.11　ルンゲクッタ法（ベクトル版）

6.2 節で，ルンゲクッタ法のプログラムを学習したことを思い出そう．ルンゲクッタ法も配列を利用すれば，連立一階微分方程式の次数に依存せず，一般的な形でプログラムを記述することができる．

例題 8.30 n 変数を持つ線形微分方程式

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}u \tag{8.39}$$

$$\boldsymbol{A}(n \times n),\ \boldsymbol{b}(n \times 1),\ \boldsymbol{x}(n \times 1),\ u(1 \times 1)$$

をルンゲクッタ法できざみ幅 Δt だけ解く関数を作成せよ．ただし，$\boldsymbol{x}$ の初期値と u は与えられるとする．

解答例

```
int runge(n,a,b,x,u,dt)
int n;
double a[N][N],b[N],x[N],u,dt;
{
   int i,j,l;
   double xt[N],f[N],y[5][N];

   for (i = 1; i <= n; ++i)
```

```
        xt[i] = x[i];

    for (l = 1; l <= 4; ++l){
        for (i = 1; i <= n; ++i){
            f[i] = b[i]*u;
            for (j = 1; j <= n; ++j)
                f[i] += a[i][j]*x[j];
        }
        for (i = 1; i <= n; ++i){
            y[l][i] = f[i]*dt;
            x[i] = xt[i] + y[l][i]*0.5;
            if (l == 3)
                x[i] = xt[i] + y[l][i];
        }
    }

    for (i = 1; i <= n; ++i)
        x[i] = xt[i] + (y[1][i] + 2.*y[2][i] + 2.*y[3][i] + y[4][i])/6.;
    return(0);
}
```

Δt は dt とした.

この関数の使用例を以下に示す.

```
  while (t <= tf){
      printf("%f ",t);
      for (i = 1; i <= n; ++i)
          printf("%f ",x[i]);
      printf("\n");
   u = 1.;
   runge(n,a,b,x,u,dt);
   t += dt;
   }
```

main() 関数の中に上のようなプログラムを書く．dt の時間間隔で時刻と状態が出力される．tf は終了時刻である．

ここでは，線形微分方程式の場合を示したが，非線形微分方程式も同様にベクトル形式プログラムで記述できる．

演習 8.20 図 6.6 の 1 自由度振動系を上で示したルンゲクッタ法の関数を用いてシミュ

レーションするプログラムを作成せよ．シミュレーションの条件は

$$m = 1, \quad c = 0.3, \quad k = 1, \quad x(0) = 0, \quad \dot{x}(0) = 0, \quad u = 1$$

$$\Delta t = 0.05, \quad t_f = 40$$

とする．また，結果をグラフ表示せよ．

関連図書

[1] 玄　光男，井田憲一：パソコン数値計算ライブラリ，CBS 出版，1986.

[2] 伊理正夫，藤野和建：数値計算の常識，共立出版，1985.

[3] B.W. カーニハン，D.M. リッチー（石田訳）：プログラミング言語 C 2 版 ANSI 規格準拠，共立出版，1989.

[4] 平林雅英：新 ANSI C 言語辞典，技術評論社，1997.

[5] 戸川隼人： NS ライブラリ 4 ザ・C 第 2 版–ANSI C 準拠–, サイエンス社，1987.

[6] 玉井 浩：新・演習と応用 プログラミング言語 1 演習と応用 C，サイエンス社，1999.

[7] Steve Summit: C Programming, 1996-9
http://www.eskimo.com/~scs/cclass/

[8] Martin Leslie: C programming Reference. Release 1.09,
http://www.phim.unibe.ch/comp_doc/c_manual/C/cref.html

[9] Thomas Williams and Colin Kelley（田丸 博晴，升谷 保博訳）： GNUPLOT 3.5 マニュアル,
http://www.linux.or.jp/JF/JFdocs/gnuplot.html

索 引

C 言語による計算プログラミング入門

2000 年 10 月 1 日初版
2001 年 10 月 1 日第 2 版第 1 刷
2002 年 10 月 1 日第 2 版第 2 刷

著者・発行者　**吉田和信**
印刷・製本　　**黒潮社**