プログラミング演習1

― 課題4 数値解析法 ―

1. Cによる数値解析のための準備

【資料】

Cによる数値解析のための準備(pdf)

1.1 数学関数

Cには標準ライブラリの中に数学関数が用意されており、ヘッダ <math.h> を インクルードすることで使用可能となる。

> **注：** UNIX用のCコンパイラでは数学関数ライブラリのリンクが必要となる。「undefined reference to `……'」などのリンクエラーが出たときはそれが原因である。

ヘッダ <math.h> 内には使用できる関数が具体的に宣言されている。Cで使用できる主な数学関数を以下に示す。

sin(x)	三角関数 sin(x) を計算する。
cos(x)	三角関数 cos(x) を計算する。
tan(x)	三角関数 tan(x) を計算する。
asin(x)	逆三角関数 $\sin^{-1}(x)$ を計算する。$(-\pi/2 \sim \pi/2)$
acos(x)	逆三角関数 $\cos^{-1}(x)$ を計算する。$(0 \sim \pi)$
atan(x)	逆三角関数 $\tan^{-1}(x)$ を計算する。$(-\pi/2 \sim \pi/2)$
atan2(y,x)	逆三角関数 $\tan^{-1}(y/x)$ を計算する。$(-\pi \sim \pi)$
sinh(x)	双曲線関数 sinh(x) を計算する。
cosh(x)	双曲線関数 cosh(x) を計算する。
tanh(x)	双曲線関数 tanh(x) を計算する。
exp(x)	指数関数 e^x を計算する。
log(x)	自然対数 ln(x) を計算する。$(x > 0)$
log10(x)	10 を底とする対数 $\log_{10}(x)$ を計算する。$(x > 0)$
pow(x,y)	べき乗 x^y を計算する。
sqrt(x)	実数 x の平方根を求める。$(x \geq 0)$
ceil(x)	実数 x の小数部を切り上げた整数を返す。
floor(x)	実数 x の小数部を切り捨てた整数を返す。
fabs(x)	絶対値|x|を求める。
ldexp(x,n)	$x \cdot 2^n$ を計算する。
frexp(x,&exp)	実数 x を仮数部と指数部(2のべき乗) exp に分解する。
modf(x,&ip)	実数 x を整数部 ip と小数部に分ける。
fmod(x,y)	実数の剰余(x % y に相当)を求める。

練習1 pi.c (円周率を求める)

```
 1: #include <stdio.h>
 2: //#include <math.h>
 3:
 4: int main(void)
 5: {
 6:     printf("PI = %.20g¥n", 4.0*atan(1.0));
 7:     return 0;
 8: }
```

■2行目のコメントをはずさないと、どのようなエラーが出るか確認せよ。

ちなみに、通常は

```
#define PI 3.14159265358979323846
  または
const double PI=3.1415926535897932384;
```

をソースの #include の下に貼り付ければ、定数 'PI' に円周率を割り当てたことになる。

ポイント：数学関数使用時は<math.h>をインクルード

1.2 数に関するユーティリティ

ヘッダ<stdlib.h> には、いくつかの有用な数値変換関数や整数乱数発生関数が宣言されている。

atof(s)	文字列 s を double型浮動小数点数に変換する。
atoi(s)	文字列 s を int型整数に変換する。
atol(s)	文字列 s を long型整数に変換する。
rand()	0～RAND_MAXの範囲の整数乱数を発生する。
srand(seed)	rand() の種として seed をセットする。
abs(n)	int型整数 n の絶対値 |n|を返す。
labs(n)	long型整数 n の絶対値 |n|を返す。

練習2 rand10.c (乱数を発生する)

```
 1: #include <stdio.h>
 2: //#include <stdlib.h>
 3:
 4: int main(void)
 5: {
 6:     int i;
 7:
 8:     printf("RAND_MAX = %d¥n", RAND_MAX);
 9:     for (i=0; i<10; i++) printf("%2d: %d¥n", i, rand());
10:     return 0;
11: }
```

■2行目のコメントをはずさないと、どのようなエラーが出るか確認せよ。
■実行のみを繰り返したとき、結果が変わるかどうか確認せよ。

ポイント：数値変換、乱数関数使用時は<stdlib.h>をインクルード

1.3 整数に関する制限

ヘッダ <limits.h> には整数型のサイズを表す定数が定義されている。

CHAR_BIT	char型のビット数
CHAR_MAX	char型の最大値
CHAR_MIN	char型の最小値
INT_MAX	int型の最大値
INT_MIN	int型の最小値
LONG_MAX	long型の最大値
LONG_MIN	long型の最小値
SHRT_MAX	short型の最大値
SHRT_MIN	short型の最小値
UCHAR_MAX	unsigned char型の最大値
UINT_MAX	unsigned int型の最大値
ULONG_MAX	unsigned long型の最大値
USHRT_MAX	unsigned short型の最大値

練習3 chkint.c (標準的な整数のビットサイズと制限の確認)

```
 1: #include <stdio.h>
 2: //#include <limits.h>
 3:
 4: int main(void)
 5: {
 6:     printf("int: %u bit, unsigned: %u bit¥n", sizeof(int)*8, sizeof(unsigned)*8);
 7:     printf("INT_MAX = %d¥n", INT_MAX);
 8:     printf("INT_MIN = %d¥n", INT_MIN);
 9:     printf("UINT_MAX = %u¥n", UINT_MAX);
10:     return 0;
11: }
```

■2行目のコメントをはずさないと、どのようなエラーが出るか確認せよ。

1.4 浮動小数点数に関する定数

ヘッダ <float.h> には浮動小数点数に関する様々な定数が定義されている。

単精度	
_FLT_RADIX	指数表現における基数b
_FLT_ROUNDS	丸めのモード
FLT_DIG	精度の10進けた数
FLT_EPSILON	計算機イプシロン
FLT_MANT_DIG	浮動小数点数の有効数字部(_FLT_RADIXを基数とする)のけた数p

FLT_MAX	表現可能な最大の有限浮動小数点数
FLT_MAX_EXP	表現可能な数の指数の最大値
FLT_MIN	最小の正規化された正の浮動小数点数
FLT_MIN_EXP	最小正規化数の指数の値

倍精度	
_DBL_RADIX	指数表現における基数b
_DBL_ROUNDS	丸めのモード
DBL_DIG	精度の10進けた数
DBL_EPSILON	計算機イプシロン
DBL_MANT_DIG	浮動小数点数の有効数字部(_DBL_RADIXを基数とする)のけた数p
DBL_MAX	表現可能な最大の有限浮動小数点数
DBL_MAX_EXP	表現可能な数の指数部の最大値
DBL_MIN	最小の正規化された正の浮動小数点数
DBL_MIN_EXP	最小正規化数の指数部の値

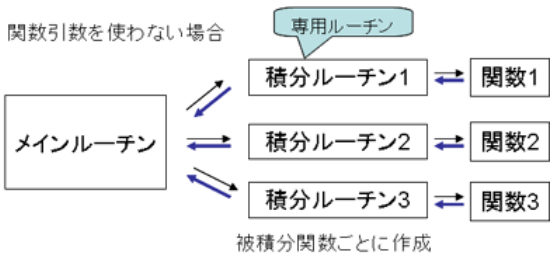
練習4 chkflt.c (浮動小数点数のビットサイズと制限の確認)

```
 1: #include <stdio.h>
 2: //#include <float.h>
 3:
 4: int main(void)
 5: {
 6:     printf("float: %u bit, double: %u bit¥n", sizeof(float)*8, sizeof(double)*8);
 7:     printf("FLT_MAX =     % e¥n", FLT_MAX);
 8:     printf("FLT_MIN =     % e¥n", FLT_MIN);
 9:     printf("FLT_EPSILON = % e¥n", FLT_EPSILON);
10:     printf("DBL_MAX =     % e¥n", DBL_MAX);
11:     printf("DBL_MIN =     % e¥n", DBL_MIN);
12:     printf("DBL_EPSILON = % e¥n", DBL_EPSILON);
13:     return 0;
14: }
```

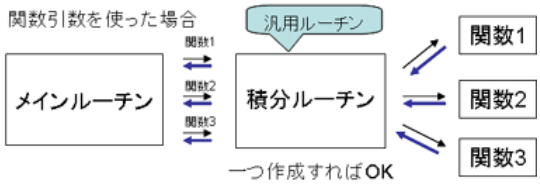
■2行目のコメントをはずさないと、どのようなエラーが出るか確認せよ。

1.5 関数引数

非線形方程式や数値積分、微分方程式のサブルーチンを作成する場合、対象となる関数や方程式が替わるたびにサブルーチン内の関数名を書き換える必要がある。



しかし、サブルーチンの引数の一つに関数名を指定できるようにすれば、サブルーチンを呼び出すメイン側に関数名の決定権を譲ることになり、非常に汎用性の高いサブルーチンとなる。



それを実現するために使われるのが**関数引数**である。(熊谷・玉城・白川「例題で学ぶC言語」の第11章 11.5 関数引数 pp.106-108 を参照のこと)

例えば後述する「二分法」のサブルーチン bisection で方程式 $f(x)=0$ の左辺の関数 $f(x)$ を任意の名前で呼び出せるようにするには、仮の関数名 f の前に*(アスタリスク)を付けてカッコで閉じた 関数名 (*f) (これを**関数ポインタ**という)を用い、これをサブルーチンの引数リスト内に関数の引数(型のみでよい)をつけて double (*f)(double) のように定義しておく。

```
double bisection(double (*f)(double), double x1, double x2)
{
    二分法の本体
}
```

さらに、このサブルーチン内部で関数を呼び出すときは

```
    if (f(x1) * f(x3) < 0) x2 = x3;
```

というように引数リスト内で定義した**仮の関数名**(この場合は f())で呼び出すようにしておく。

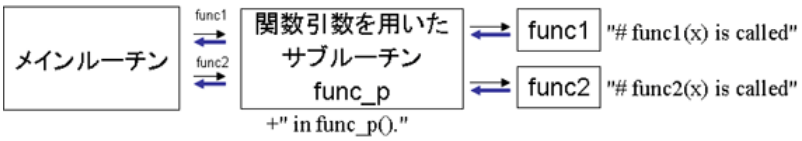
方程式の左辺の関数が実際には func1(x) という名前で定義されているなら、main関数で二分法のサブルーチンを呼び出すときに

```
    x = bisection(func1, x1, x2);
```

と**実際の関数名**を引数なしで指定して呼び出せばよい。

練習5 funcp.c (関数引数の使用)

```
 1: #include <stdio.h>
 2:
 3: double func1(double x) // f(x) = x を返す関数
 4: {
 5:     printf("# func1(% g) is called", x);
 6:     return x;
 7: }
 8:
 9: double func2(double x) // f(x) = x^2 を返す関数
10: {
11:     printf("# func2(% g) is called", x);
12:     return x*x;
13: }
14:
15: double func_p(double (*f)(double), double x) // 関数引数を用いたサブルーチン
16: {
17:     double y;
18:
19:     y = f(x); // ここでは仮の関数名 f() を用いる
20:     printf(" in func_p().¥n");
21:     return y;
22: }
23:
24: int main(void)
25: {
26:     printf("func1(1) = % g¥n", func_p(func1,1)); // func1 を指定して func_p を呼び出す
27:     printf("func2(2) = % g¥n", func_p(func2,2)); // func2 を指定して func_p を呼び出す
28:     return 0;
29: }
```



■上記のプログラムを実行するとどうなるかを事前に予想せよ。
■実際に実行してみて関数引数の使い方を確認せよ。

1.6 for文による等間隔実数列の生成

数値解析では一定間隔に刻んだ実数列を用いて計算することが多い。以下は、柴田望洋「新版 明解C言語 入門編」の第7章にある、0 から1 まで 0.01 間隔の実数列を生成するプログラム List 7-9 を少し手直ししたもの(forflt.c)である。

```
 1: /*
 2:     0 から1 まで 0.01 間隔の実数列を生成する。
 3:  */
 4: #include <stdio.h>
 5:
 6: int main(void)
 7: {
 8:     float x; // double にするとどうなるか?
 9:
10:     for (x=0; x<=1; x+=0.01) printf(" x = % g¥n", x);
11:
12:     return 0;
13: }
```

実行例

```
 x = 0
 x = 0.01
 x = 0.02
 ...
 x = 0.979999
 x = 0.989999
 x = 0.999999
```

このように、実数(単精度浮動小数点数)を直接、for文で増減させると、刻み幅(上記の場合は0.01)を2進数にしたときの丸め誤差(表現誤差) が累積されて次第に値を正確に刻んでくれなくなる場合が多い。このようなことを避けるためにも、**for**文は必ず**整数**で制御するべきである。以下のプログラム(forflt2.c)は、この考え方を適用した「明解C言語」の第7章 List 7-10と同じものである。

```
 1: /*
 2:     0 から1 まで 0.01 間隔の実数列を生成する。 (整数で制御)
 3:  */
 4: #include <stdio.h>
 5:
 6: int main(void)
```

```
 7: {
 8:     int i;
 9:     float x;
10:
11:     for (i=0; i<=100; i++) {
12:         x = i / 100.0;       // 100 にするとどうなるか? 100. では?
13:         printf(" x = % g¥n", x);
14:     }
15:     return 0;
16: }
```

また、実数(浮動小数点数)宣言を double にすることで表現誤差が非常に小さくなる。float を用いていて結果がおかしいときには double にするだけで直ることがある。

練習6 (等間隔実数列の生成)

- 上記の2つのプログラム(forflt.cと forflt2.c)の結果を比較せよ。
- 2つめのプログラム(forflt2.c)で12行めの '100.0' を '100' にしてみよ。また、'100.' にしてみよ。
- 1つめのプログラム(forflt.c)で8行目の 'float' を 'double' にしてみよ。

ポイント(3つ)
for文は必ず整数で制御
実数定数は小数点をつけておく
実数は float よりも double で宣言

1.7 Excelによるグラフ描画

出力データを**CSV形式**で保存すれば、**Excel**でただちに読み出すことができる。CSV形式は一組のデータを単にコンマで区切り、組ごとに改行すればよい。ファイル保存で名前をつける際、拡張子を".csv" とする。先頭行にデータの名前を入れておくと後で便利である。

以下の例は後述の2分法の出力ファイルで、名前を"sample.csv"とする。

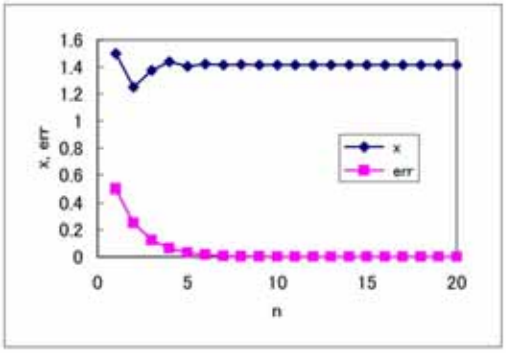
```
n,  x,  err
  1,  1.500000000000000e+000,  5.000000000000000e-001
  2,  1.250000000000000e+000,  2.500000000000000e-001
  3,  1.375000000000000e+000,  1.250000000000000e-001
  4,  1.437500000000000e+000,  6.250000000000000e-002
  5,  1.406250000000000e+000,  3.125000000000000e-002
  6,  1.421875000000000e+000,  1.562500000000000e-002
  7,  1.414062500000000e+000,  7.812500000000000e-003
  8,  1.417968750000000e+000,  3.906250000000000e-003
  9,  1.416015625000000e+000,  1.953125000000000e-003
 10,  1.415039062500000e+000,  9.765625000000000e-004
 11,  1.414550781250000e+000,  4.882812500000000e-004
 12,  1.414306640625000e+000,  2.441406250000000e-004
 13,  1.414184570312500e+000,  1.220703125000000e-004
 14,  1.414245605468750e+000,  6.103515625000000e-005
 15,  1.414215087890625e+000,  3.051757812500000e-005
 16,  1.414199829101563e+000,  1.525878906250000e-005
 17,  1.414207458496094e+000,  7.629394531250000e-006
 18,  1.414211273193359e+000,  3.814697265625000e-006
 19,  1.414213180541992e+000,  1.907348632812500e-006
 20,  1.414214134216309e+000,  9.536743164062500e-007

sqrt(2) = 1.414213657379150e+000
```

---

1. "sample.csv"のアイコンをダブルクリックする。
   - ファイル名 *.csv には Excel 2010 が関連付けられているので、Excel 2010 が起動し、sample.csv が読み込まれる。(起動しない場合は、Excel 2010を立ち上げてから読み込むこと:ファイル(F)から開く(O)とする)
2. グラフを作成するために、データの範囲を選択する。以下の操作により、データ範囲をすばやく選択できる。
   - セル **A1** が選択されていることを確認。
   - [Ctrl] キーと [Shift] キーを同時に押しながら [→] キーを押す。
   - [Ctrl] キーと [Shift] キーを同時に押しながら [↓] キーを押す。
   - 数値の範囲が反転し、選択されていることを確認する。
3. **[挿入]→[グラフ]→「散布図」**を選択。
4. **[散布図(直線とマーカー)]**を選択。
5. **[グラフツール]→[レイアウト]→[軸]**を選択し、**「主横軸」**に**"n"** を、**「主縦軸」**に**"x, err"** を記入する。
6. **[縦軸]** ダブルクリックし、**「軸の書式設定」の[表示形式]**で表示を標準にする。
7. 変更したい枠線はダブルクリックして、**[枠線の色]** などで変更する。当然、枠線なしも可能である。
8. 好みに応じて軸の目盛の範囲や凡例の位置、背景などを変更する。
9. グラフをコピーして Word 2010 など他のアプリケーションに貼り付けることも可能である。

# n	x	err
1	1.5	0.5
2	1.25	0.25
3	1.375	0.125
4	1.4375	0.0625
5	1.40625	0.03125
6	1.421875	0.015625
7	1.4140625	0.0078125
8	1.41796875	0.00390625
9	1.416015625	0.001953125
10	1.415039063	0.000976563
11	1.414550781	0.000488281
12	1.414306641	0.000244141
13	1.41418457	0.00012207
14	1.414245605	6.10352E-05
15	1.414215088	3.05176E-05
16	1.414199829	1.52588E-05
17	1.414207458	7.62939E-06
18	1.414211273	3.8147E-06
19	1.414213181	1.90735E-06
20	1.414214134	9.53674E-07

sqrt(2) = 1.414213657379150e+000



練習7 (Excelによるグラフの描画)

- まず“sample.csv”を右クリックし、「対象をファイルに保存(A)...」で演習1課題4用のフォルダ(例えば“Z:¥proen1¥kadai4”)に保存する。
- 実際に上記の操作を行い、グラフを描いてみよ。

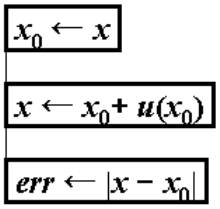
練習7' (Excelによる3Dグラフの描画)

- 上記と同様に、“sample2.csv”を右クリックで保存し、3Dグラフ(等高線)を描いてみよ。

1.8 PADについて

PADとは Problem Analysis Diagram の略である。構造化フローチャートの一種で、処理の方向は「上から下へ」流れる。具体的な処理内容が書かれた「処理箱」と呼ばれる 横長の四角い箱の左端を「縦線で結合(連接)」することによりプログラムのフローをあらわす。Cにおいては、処理箱は文に、連結されたものはブロックに相当する。

処理、連接

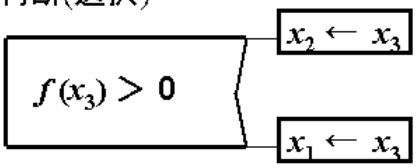



```
{
   x0 = x;
   x  = x0+u(x0);
   err= fabs(x-x0);
}
```

「判断(選択)」、「反復」などの処理の中の文・ブロックはそれらの箱の右側に置き、横線で結ぶ。これらの処理ではフローをいったん右側の文・ブロックに移す。ブロックの場合は上からはじまり、一番下で左側に戻る。

「判断(if文)」は、箱の右辺が中折れしており、上下の頂点から横に1本ずつ2本の分岐線を伸ばすことができ、条件によっていずれか1つの分岐のみが実行される。

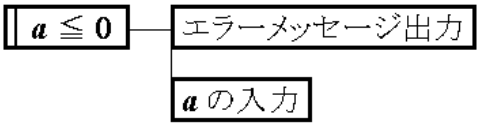
判断(選択)




```
if (f(x3) > 0) x2 = x3;
else           x1 = x3;
```

「前判定反復(while文)」は、箱の左側を二重線とする。

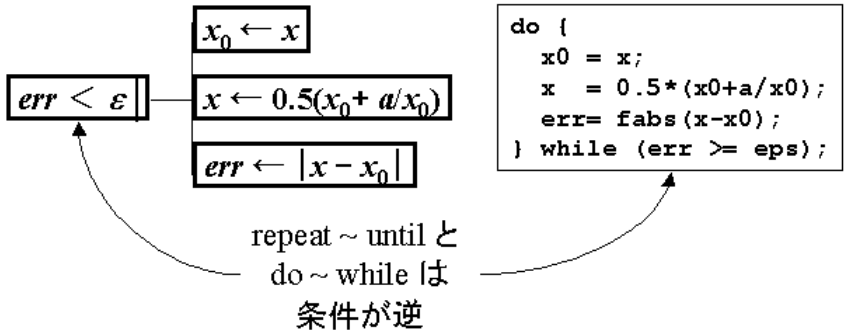
前判定反復




```
while (a <= 0) {
  printf("input error¥n");
  printf("a = ");
  scanf("%lf", &a);
}
```

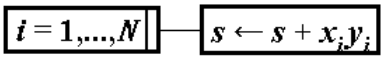
「後判定反復(repeat～until文)」、「問題向き反復(for文)」は箱の右側を二重線とする。Cでの後判定反復の実装にはrepeat～until文の代わりにdo～while文を用いる。このとき、条件文が逆(否定)になることに注意する。

後判定反復 (repeat ~ until)




```
do {
  x0 = x;
  x  = 0.5*(x0+a/x0);
  err= fabs(x-x0);
} while (err >= eps);
```

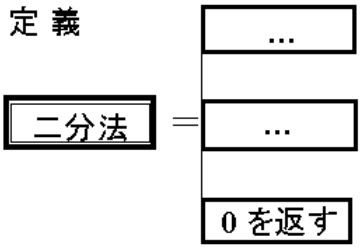
問題向き反復




```
for (i=1;i<=N;i++) {
  s += x[i]*y[i];
}
```

「定義」、「引用(参照)」は箱の内部に手続き名などを書き、両側あるいは境界を二重線とする。定義は、箱の右側から二重線を横に出して処理ブロックに結ぶ。

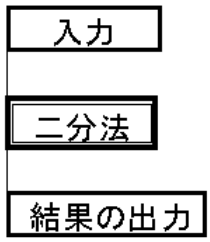
定 義




```
int bisection()
{
     ...  ;
     ...  ;
  return 0;
}
```

引用(参照)は箱の両側または境界が二重線であること以外は処理箱と同様の扱いとなる。

引用(参照)




```
{
  scanf("%lf%lf",&x1,&x2);
  x = bisection(x1,x2);
  printf("x = %g¥n",x);
}
```

PADのテンプレート

1. ここ(pad.ppt)を右クリックし、「対象をファイルに保存」で演習1_課題4用のフォルダに保存する。
2. pad.pptを開き、ファイル内のPADをPowerPointの新規スライドに コピー・貼り付けし、グループ解除し、加工する。
3. 加工し終わったらペイントツールを立ち上げ、新規ページを小さめにしておく。
4. 完成したPADをPowerPoint上でドラッグ選択し、コピーする。
5. ペイントツール(Adobe Photoshopなど)上で新規ページに貼り付け、画像ファイルとして名前をつけて保存する。

Wordなどで挿入できるファイル形式(jpeg, gif, tiffなど)にしておけば、レポート作成に利用できる。

また、Word上で書いたPADのサンプルも用意した。次のように、判断(選択)の箱をフローチャートの「記憶データ」で代用している。(上から処理箱を重ねればもう少し見栄えが良くなるが…。) レポートにはこちらの方法で描いてもよいことにする。