5E　2003.7.17

数値計算用の C 言語コンポーネント集

学習内容 (目次)

秋田工業高等専門学校　電気工学科　山本昌志

１．概要

このような図書を作るのは過保護のような気がします。プログラム言語を習得するためには、いろいろプログラムを書いて苦労するものです。授業だけでプログラム言語の習得は非常に難しいと考えます。何年も学習して英語が使えないのと同じです。皆さんも問題に直面したらプログラムを書いて、解決に努めてください。そうすると上達します。この図書により、すこしでもプログラムを書くハードルが下がれば幸いです。

初心者がプログラムを書き始めると、非常に時間がかかります。フローチャートから実際のプログラムに直せないのです。だれでも初めはプログラムの書き方が分からなくて、苦労します。そしていろいろと考えたり、人のプログラムを見たりして学習します。

皆さんには苦労して学習して欲しいですが、数値計算を学習する場で、プログラムテクニックに時間を取られるのも問題です。そこで、ｃ 言語でプログラムを作成する場合の例を示します。c 言語のプログラミングになれていない人は、この例に倣って、数値計算のプログラムを作成してください。

ここに載せている内容は、中間試験前までに学習したの要約です。以前の学習では、c 言語の文法も含めて非常に細かい説明をしました。c 言語の教科書のつもりで記述しました。一方、この図書は文法の細かい説明は省き、実戦に必要なことのみを書きました。文法の詳細が必要なときは、ｃ 言語の解説書あるいは以前配ったプリントを参照してください。

2．キーボード入力

フローチャートに図1のキーボード入力がある場合の、プログラム方法を示します。

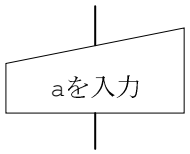

図1 キーボード入力

2.1 書式付入力

フローチャートに図 1 のように書かれていれば、キーボード入力を意味します。この場合、書式付入力の scanf という関数を使う方法が最も一般的です。引数はポインターです。そして、データ入力完了を示す[Enter][1]を読み捨てる必要があります。

ここでは、4 通りの方法を示します。①は temp に読み込む方法(教科書)。②は代入を抑止する方法。③は gets で読み込む方法。この場合、以下の例の dmystr は、char dmystr[80];のように宣言する必要があります。④は rewind 関数[2]でバッファーの内容を捨てる方法です。④がもっとも確実な方法です。

1 文字の場合

```
① scanf("%c%c", &a, &temp);
② scanf("%c%*c", &a);
③ scanf("%c", &a);
   gets(dmystr);
④ scanf("%c", &a);
   rewind(stdin);
```

文字列の場合

a は配列のポインター[3]とします。文字列の中に空白があると scanf は使えません。空白はデータの区切りと解釈されます。そのときは、gets を使います。

```
① scanf("%s%c", a, &temp);
② scanf("%s%*c", a);
③ scanf("%s", a);
   gets(dmystr);
④ scanf("%s", a);
   rewind(stdin);
```

整数(10 進数)の場合

```
① scanf("%d%c", &a, &temp);
② scanf("%d%*c", &a);
③ scanf("%d", &a);
```

---

[1] バッファーの最後に'\n'が入る。

[2] ファイルの位置指定子を先頭に設定する関数。

[3] 例えば、char a[80];のように変数の宣言をします。

```
   gets(dmystr);
④ scanf("%d", &a);
   rewind(stdin);
```

倍精度実数の場合

```
① scanf("%lf%c", &a, &temp);
② scanf("%lf%*c", &a);
③ scanf("%lf", &a);
   gets(dmystr);
④ scanf("%lf", &a);
   rewind(stdin);
```

2 個以上読み込むの場合

入力データの区切りは、空白あるいは tab、改行です。

```
① scanf("%lf%lf%c", &a, &b, &temp);
② scanf("%lf%lf%*c", &a, &b);
③ scanf("%lf%lf%", &a, &b);
   gets(dmystr);
④ scanf("%lf%lf%", &a, &b);
   rewind(stdin);
```

2.2 その他の入力

キーボードに打ち込まれた文字を getchar あるいは gets で文字として読み込んで、必要な形に変換する方法です。変換には、atoi 関数や atof 関数、sscanf 関数を使います。皆さんはとりあえず、scanf 関数を使いましょう。

2.3 メッセージのつけ方

データを入力するように促すメッセージをつけると、ユーザーに対して分かりやすいプログラムになります。入力の関数の前に、printf 関数でメッセージを書くことができます。その場合、printf 関数のメッセージの最後に'\n'をつけないようにすれば、スマートなメッセージになります。

```
printf("input intial value? a = ");
scanf("%lf%", &a);
rewind(stdin);
```

3．ディスプレイへの表示

フローチャートに図 2 のディスプレイ出力がある場合の、プログラム方法を示します。

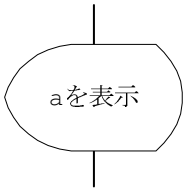

図 2 キーボード入力

この場合は、非常に簡単で、printf 関数を使います。当然、a に対するメッセージも書くことができます。

1 文字の場合

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

文字列の場合

この場合、a は文字列の先頭アドレスを表すポインターです。

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

整数(10 進数)の場合

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

倍精度実数の場合

①は浮動小数点形式、②は指数形式で表示する。数値の値がわからないときは、指数形式で表示しましょう。

① `printf("a = %f\n", a);`
② `printf("a = %e\n", a);`

一度に 2 個以上表示する場合

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

4. 制御文

4.1 分岐

分岐には、if 文と switch 文があります。switch 文はプログラムの構造が分かり難くなりますので、できるだけ if 文を使いましょう。

if 文には典型的な 3 つのパターンがありますので以降説明します。if 文中の制御式は、0(ゼロ)の場合のみ偽であり、その他は全て真です。

if のみ場合

制御式(条件)が真の場合には特定の文を実行しますが、偽の場合には実行する文がありません。

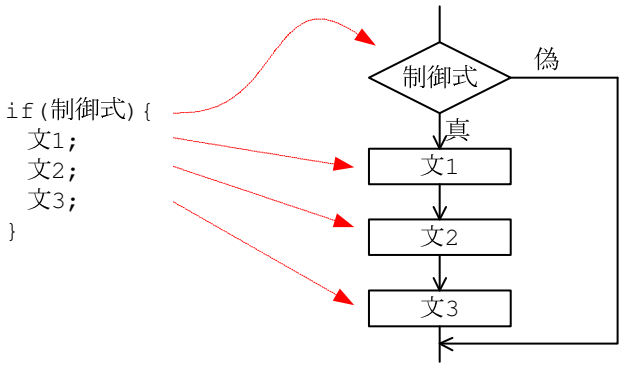

図 3 if のみの場合

if～else の場合

制御式(条件)が真偽に応じて、実行する文が異なります。

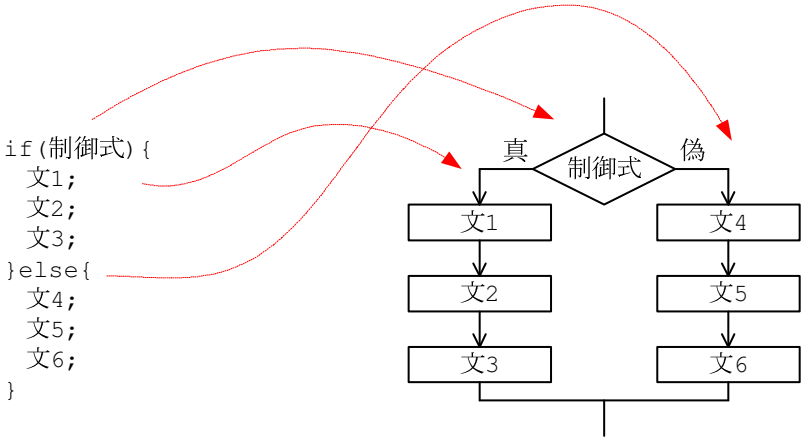

図 4 if～else の場合

if～else if～else の場合

制御式(条件)が多段階になっています。最初に真にマッチしたブロックを実行します。制御式が真にならない場合は、else のブロックを実行します。

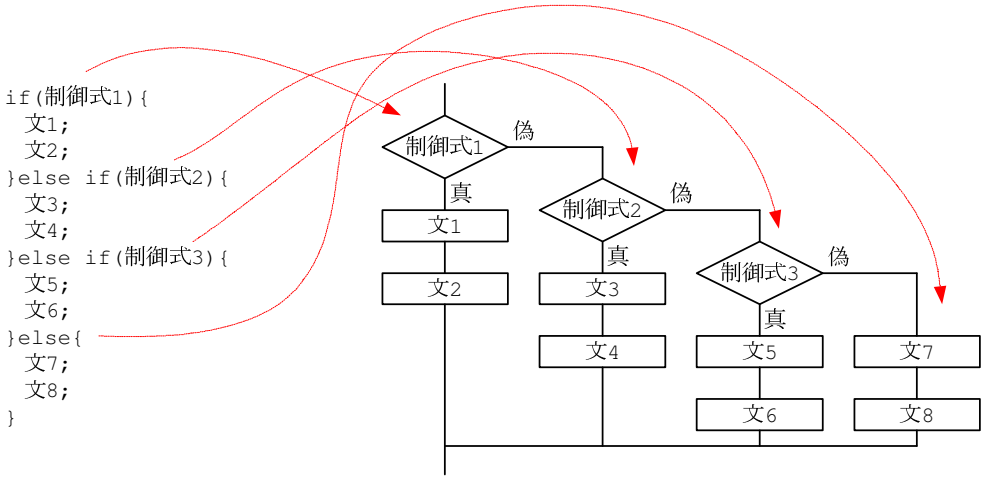

図 5 if～else if～else の場合

4.2 指定回数繰り返し

条件に従い同じ文を繰り返し実行する構造をループと呼びます。繰り返し回数が分かっている場合は、for 文を使います。

for 文を使う方法

ループの回数が予め分かっている場合に使われます。以下の例では、初期値 i=1 から初めて、1 回ループが回るごとに i が 1 ずつ増加して、i≦100 が真の間の間は繰り返します。i=101 になればループを抜け出します。ループ中の分の中で i を使っても良いです。

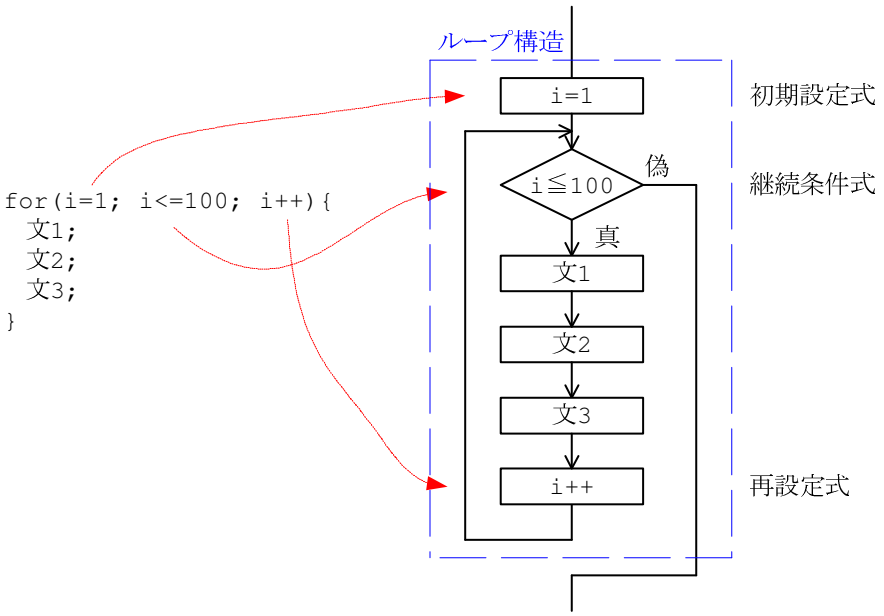

図 6 for 文を用いた指定繰り返し

4.3 不定回数繰り返し

繰り返し回数が予め分かっていない場合は、do while 文か while 文を使います。それぞれの違いは、以下の通りです。

- do while 文の場合、継続条件の判断はループ出口です。
- while 文の場合、継続条件の判断はループ入り口です。

do while 文

出口で継続条件を判断するため、必ず 1 回はループが回ります。ループを抜け出すためには、ループの文中に継続条件に使う値を変更するか、break 文を使います。

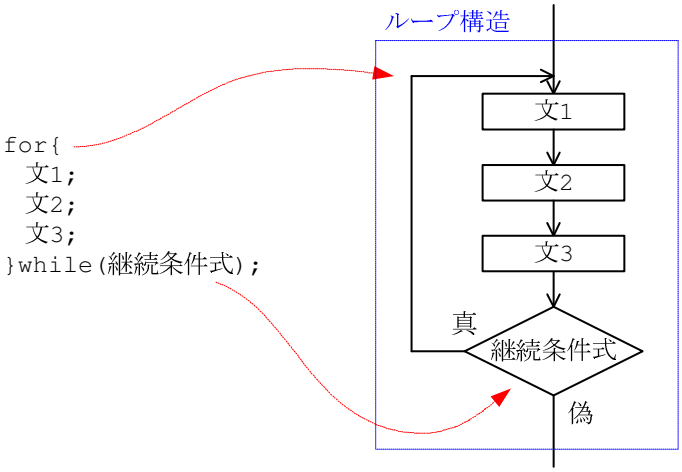

図 7 do while 文を用いた繰り返し

while 文

入口で継続条件を判断するため、最初から偽の場合、ループは一度も回りません。ループを抜け出すためには、ループの文中に継続条件に使う値を変更するか、break 文を使います。

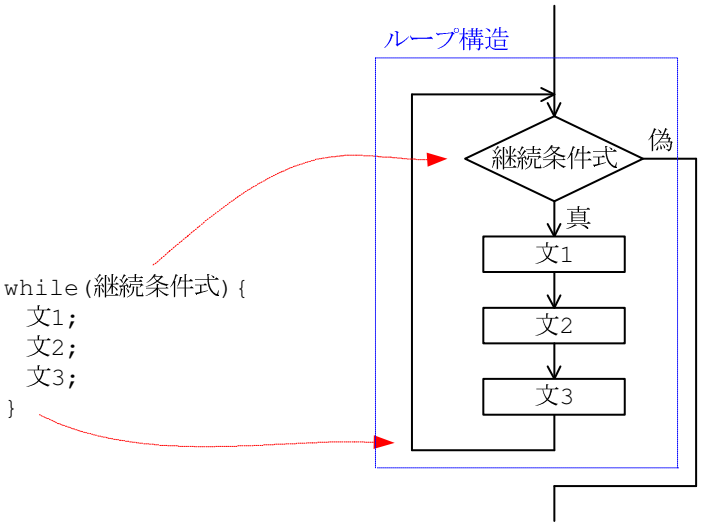

図 8 while 文を用いた繰り返し

4.4 繰り返し(ループ)からの強制脱出

ループから強制的に脱出するために、break 文があります。単独で使われることは希で、以下のように if 文と共に使われます。break 文を用いると、for 文や do while 文、while 文のループから抜け出せます。

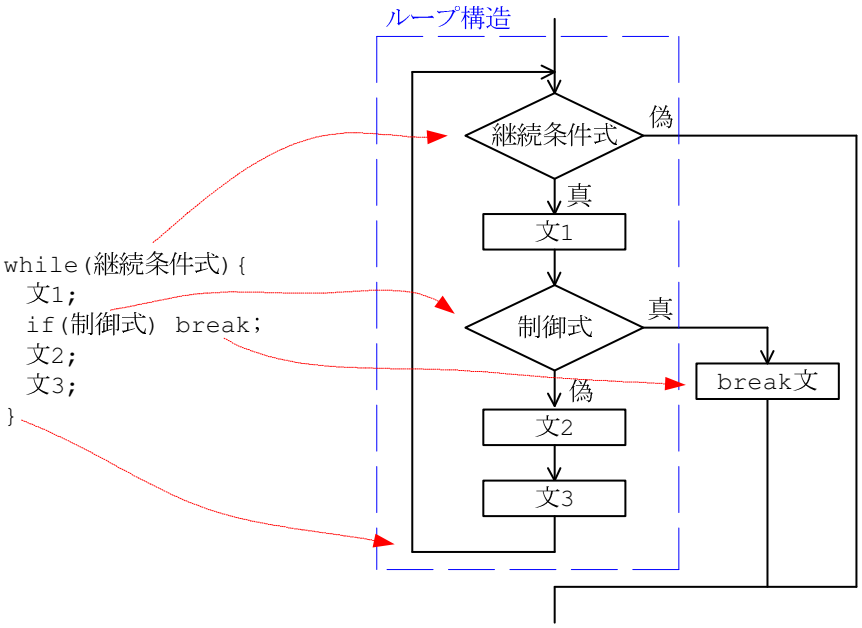

図 9 break 文を用いたループからの脱出

4.5 繰り返しのスキップ

ループ中の繰り返しを 1 回強制的にスキップするために使われます。これも単独で使われることは希で、以下のように if 文と共に使います。

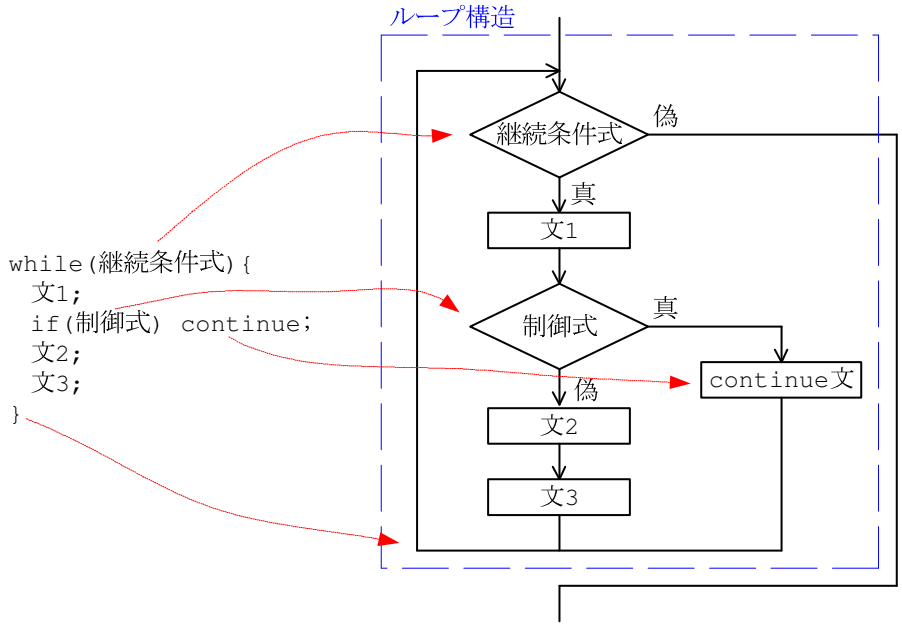

図 10 continue 文を用いたループのスキップ

4.6 強制ジャンプ

強制的にプログラムの制御を移します。goto 文が示すラベルに実行が移ります。if 文と共に用いられることも多いですが、単独で用いられることも多いです。

ただし、goto 文はプログラムの流れがわかりにくくなりますので、使わないほうが良いとされています。行儀の良いプログラムには goto 文は使われていません。ただし、初心者が書くような短いプログラムであれば使っても良いでしょう。簡単ですし、行儀の悪いプログラムでも書くことが重要です。プログラミングに慣れたら、goto 文を書かないようにしましょう。

ラベル名は、文の前に書いてコロンをつけます。セミコロンではないですよ。

```
ラベル1: if(制御式) goto ラベル2
文1;
文2;
goto ラベル1;
```

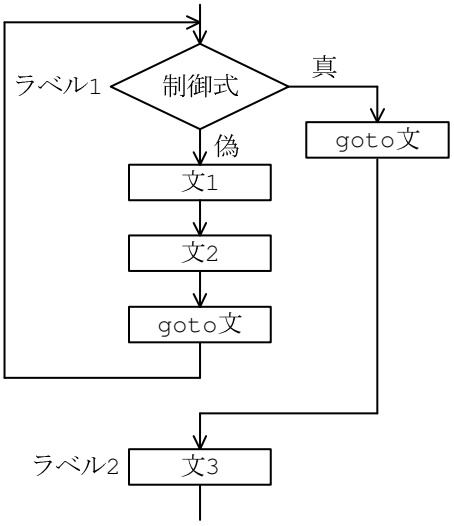

図 11 goto 文を用いたジャンプ

5．数学関数の定義

5.1 C 言語の関数の機能を使う方法

最も一般的な方法です。教科書は#define を使っていますが、間違える可能性がありますので、以下の方法を使うほうが良いでしょう。

例えば、

$$f(x) = x^3 - 3x^2 + 9x - 8 = 0 \qquad (1)$$

を、C 言語の関数の機能を使って定義してみましょう。以前学習した通りで、関数の外にプロトタイプ宣言を書いて、main 関数と同列のところに処理を書きます。

```
#include <stdio.h>
double my_func(double x);          ← プロトタイプ宣言
main(){
        文

     y1 = my_func(x1);             ← 関数をコール

        文
}

double my_func(double x){
  double y;

  y=x*x*x-3*x*x+9*x-8;                関数の処理
  return(y);
}
```

5.2 #define を使う方法

関数を定義するのに#define を使うメリットは、私には分かりません。実行速度が少し速くなるかも・・・。教科書で使っているので、とりあえず説明しておきます。先ほどと同じ関数は、次のように定義します。

```
#include  <stdio.h>
#define F(x)  (x*x*x-3*x*x+9*x-8)
main(){
   文
   文

   y1 = F(x1);

   文

}
```

6．数列、ベクトル、行列の表現

数列やベクトル、行列の成分を表す場合、下添え字がつきます。これら添え字がつく数字を計算するプログラムでは、配列を使うのが常識です。

添え字がつかない量、通常のスカラー量の場合、それが 1 個の場合は通常の変数として扱います。しかし、同じようなスカラー量が多くある場合、それを並べてベクトルのように扱います。

数列の場合

数学の表現	C 言語の表現
a_1	a[1]
a_2	a[2]
a_3	a[3]
：	：
a_i	a[i]
a_{i+1}	a[i+1]
：	：
a_{2i+1}	a[2*i+1]
：	：

ベクトルの場合

ベクトルが 1 個の場合は、数列と同じです。同じようなベクトルが複数ある場合は、行列のような書き方をします。

行列の場合

数学の表現	C 言語の表現
$a_{1\,1}$	a[1][1]
$a_{1\,2}$	a[1][2]
$a_{1\,3}$	a[1][3]
：	：
$a_{3\,3}$	a[3][3]
$a_{3\,4}$	a[3][4]
$a_{i\,j}$	a[i][j]
$a_{i+1\,j+1}$	a[i+1][j+1]
：	：
$a_{2i+1\,3j+2}$	a[2*i+1][3*j+2]
：	：

7．ファイル入出力

計算結果をハードディスクに保存したり、計算処理する対象をハードディスクから読み出したりするファイル入出力は大変重要です。その方法を示します。

7.1 ファイル出力

計算結果などをハードディスクに保存する方法です。ファイル出力のコツは、

- 1 行に複数のデータがある場合は、"\t"の[Tab]区切り[4]とします。
- 作成するデータのイメージは、表です。
- ループ文を用いて、fprintf 関数を繰り返し使います。

です。

```
#include <stdio.h>
#include <math.h>
main(){
  FILE *out_file;                          ← ファイルポインター宣言
  double x, y1, y2, y3;
  double pi, dphi;
  int i, n;

  pi = 4*atan(1);
  n = 360;

  dphi = 2*pi/n;
  printf("dphi = %e\n", dphi);

  out_file = fopen("func.txt","w");        ← ファイルオープン


  for(i=0; i<=n; i++){
    x=i*dphi-pi;
    y1 = sin(x);
    y2 = cos(x);
    y3 = tan(x);                           ← データの書き出し

    fprintf(out_file, "%e\t%e\t%e\t%e\n", x, y1, y2, y3);
  }

  fclose(out_file);                        ← ファイルクローズ
}
```

[4] 空白区切りでもプログラム上問題ないのですが、テキストファイルを直接人間が見る場合、[Tab]のほうが見やすいです。

7.2 ファイル入力

処理の対象となるデータをハードディスクから読み込む方法です。7.1 節で作成したデータを読み込むプログラムは、以下の通りです。ファイルからのデータ入力のコツは、

- データ数が多い場合、読み込んだデータは配列に格納します。
- ループ文を用いて、fscanf 関数を繰り返し使います。
- fscanf 関数の戻り値が EOF の場合、データの読み込みを止める。EOF は end of file の略でファイルの終わりを示します。

です。

```
#include <stdio.h>
main(){
   FILE *in_file;                          ← ファイルポインター宣言
   double x[500], y1[500], y2[500], y3[500];
   int i;

   in_file = fopen("func.txt","r");        ← ファイルオープン

   for(i=0; i<=499; i++){                  ← データの読み出し
      if(EOF ==
         fscanf(in_file, "%lf%lf%lf%lf",
            &x[i], &y1[i], &y2[i], &y3[i])
      ) break;
   }

   fclose(in_file);                        ← ファイルクローズ
}
```

8．グラフィックス

数値計算の結果を可視化することは、非常に重要です。計算結果を他の人に示す場合、説得力が増します。あるいは、計算の間違いを探す場合も有効です。ここでは、世間でよく使われているフリーソフトウェアーの gnuplot というプログラムを紹介します。これは、秋田高専の計算サーバーの namahage および情報教育ルームの windows NT にインストールされています。グラフの見栄えは Excel より良いです。

8.1 gnuplot の説明

2 次元および 3 次元グラフを作成するフリーソフトウェアーです。手軽に使えますが、インターフェースはコマンドを打ち込む Character User Interface (CUI) です。通常、C 言語のプログラムで作図する場合は、グラフィックライブラリーを使いますが､結構めんどくさいので､お手軽な gnuplot を使うことにします｡gnuplot のせつめいは、以下の web を見てください。

web
http://t16web.lanl.gov/Kawano/gnuplot/
http://lagendra.s.kanazawa-u.ac.jp/ogurisu/manuals/gnuplot-intro/

マニュアル
http://dsl4.eee.u-ryukyu.ac.jp/DOCS/gnuplot.pdf

C 言語との連携
http://ayapin.film.s.dendai.ac.jp/~matuda/TeX/PDF/40th.pdf

8.2 C 言語の作図プログラム

数値計算の結果をグラフにする方法は、①計算結果のデータファイルを作成②gnuplot データファイルを開きグラフ作成の 2 段階経ます。詳細は、8.3 プログラム例を見てください。データファイル作成とグラフ作成の注意は以下の通りです。

データファイル

- データは、タブ区切りとします。要するに printf 関数で 1 行に複数のデータを書くときは、その区切りに\t を使います。
- 通常 1 行に、

```
x       y1      y2      y3      y4
```

のように、x とそれに対応する y の値を書きます。

データファイル

- C 言語のプログラムから gnuplot を呼び出して、それにデータを送るためにパイプを使います。パイプを使うために popen 関数を使います。
- パイプの使い方は、ファイルと同じです。ファイル感覚で gnuplot にデータ(コマンド)を送ることができます。
- gunplot が終了しても、グラフが消滅しないように -persist オプションをつけて呼び出します。

8.3 gnuplot の代表的なコマンド

代表的なコマンドは、先に示した web ページを見てください。

8.4 プログラム例

三角関数の sin, cos, tan のグラフ(図 13)を作成しています。main 関数の他に、三角関数の数値データを作成する mk_triangle_data 関数とグラフを作成する mk_graph 関数を使っています。main 以外の 2 つの関数は、私が作成したもので、必要に応じて皆さんが改良して使ってください。

それぞれの関数の引数は、以下の通りです。

```
mk_triangle_data("file 名", x 初期値, x 最終値, 分割数);
```

file 名	データを格納するファイル名
x 初期値	計算する三角関数の変数の初期値
x 最終値	計算する三角関数の変数の最終値
分割数	計算する x の分割数

```
mk_graph("file 名", "x ラベル", x 最小, x 最大, "y ラベル", y 最小, y 最大);
```

file 名	データが格納格納されているファイル名
x ラベル	x 軸のラベル名
x 最小	x 軸の範囲の最小値
x 最大	x 軸の範囲の最大値

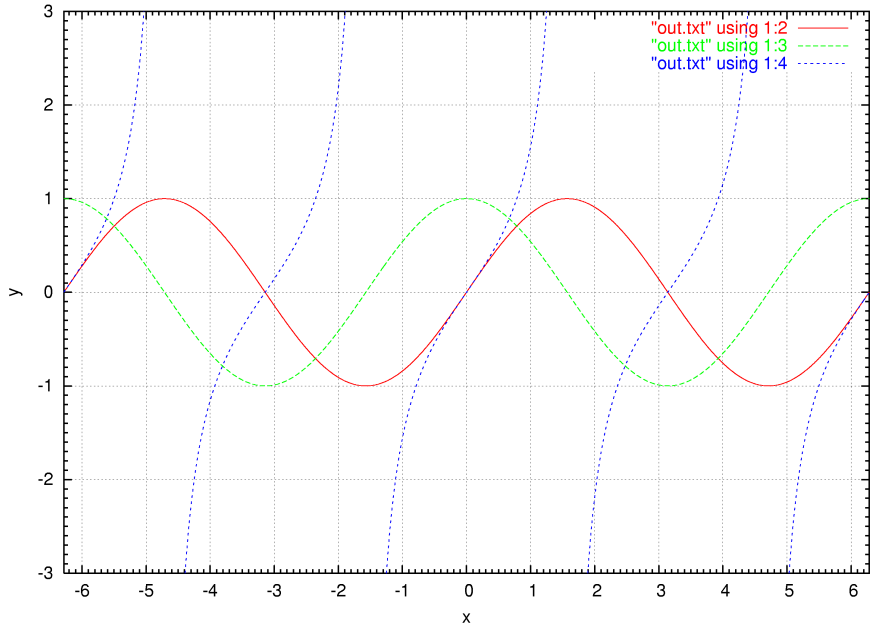

図 12 三角関数のグラフ

```
#include <stdio.h>
#include <math.h>
void mk_triangle_data(char *a, double x1, double x2, int n);
void mk_graph(char *f, char *xlb, double x1, double x2,
                       char *ylb, double y1, double y2);

/*=========================================================*/
/*   main function                                         */
/*=========================================================*/
main(){

   double pi = 4*atan(1);

   mk_triangle_data("out.txt", -2*pi, 2*pi, 1000);
   mk_graph("out.txt", "x", -2*pi, 2*pi,  "y", -3, 3);

}

/*=========================================================*/
/*   make a data file                                      */
/*=========================================================*/
void mk_triangle_data(char *a, double x1, double x2, int n){
   double x, dx;
   double y1, y2, y3;
   int i;
   FILE *out;

   dx = (x2-x1)/n;

   out = fopen(a, "w");

   for(i=0; i<=n; i++){
      x = x1+dx*i;
      y1 = sin(x);
      y2 = cos(x);
      y3 = tan(x);

      fprintf(out, "%e\t%e\t%e\t%e\n", x, y1, y2, y3);
   }

      fclose(out);
}

/*=========================================================*/
/*   make a graph                                          */
/*=========================================================*/
void mk_graph(char *f, char *xlb, double x1, double x2,
                       char *ylb, double y1, double y2)
{

  FILE *gp;
  char close;
  int i;
  double x;

   gp = popen("gnuplot -persist","w");

   fprintf(gp, "reset\n");
```

```
    fprintf(gp, "set terminal postscript eps color\n");
    fprintf(gp, "set output \"graph.eps\"\n");
    fprintf(gp, "set grid\n");

       /* -------  set x axis ---------*/

    fprintf(gp, "set xtics 1\n");
    fprintf(gp, "set mxtics 10\n");
    fprintf(gp, "set xlabel \"%s\"\n", xlb);
    fprintf(gp, "set nologscale x\n");
    fprintf(gp, "set xrange[%e:%e]\n", x1, x2);

              /* -------  set y axis ---------*/

    fprintf(gp, "set ytics 1\n");
    fprintf(gp, "set mytics 10\n");
    fprintf(gp, "set ylabel \"%s\"\n", ylb);
    fprintf(gp, "set nologscale y\n");
    fprintf(gp, "set yrange[%e:%e]\n", y1, y2);

       /* -------  plat graphs ---------*/

    fprintf(gp, "plot \"%s\" using 1:2 with line,\
                      \"%s\" using 1:3 with line,\
                      \"%s\" using 1:4 with line\n", f, f, f);

    fprintf(gp, "set terminal x11\n");
    fprintf(gp, "replot\n");

   pclose(gp);
}
```