

# 数値計算・情報工学演習 II

松原 隆, 鵜飼 孝盛

2024 年度

## 目次

<b>1</b>	<b>はじめに</b>	<b>4</b>
1.1	参考書 . . . . .	4
1.2	演習の準備 . . . . .	4
1.3	Linux コマンドの簡単な復習 . . . . .	7
1.4	授業の操作手順 . . . . .	7
1.5	課題の提出要領 . . . . .	9
<b>2</b>	<b>簡単なプログラムと実行方法</b>	<b>13</b>
<b>3</b>	<b>数値計算の基礎</b>	<b>18</b>
3.1	誤差および誤差の伝搬 . . . . .	18
3.2	有効数字 . . . . .	19
3.3	まるめ誤差 . . . . .	19
3.4	桁落ち . . . . .	20
3.5	積残し . . . . .	27
3.6	計算回数の削減 . . . . .	30
3.7	基本的な計算 . . . . .	31
<b>4</b>	<b>数値積分</b>	<b>33</b>
4.1	概説 . . . . .	33
4.2	中点公式 . . . . .	33
4.3	台形公式 . . . . .	33
4.4	第 1 シンプソン公式 . . . . .	34
4.5	第 2 シンプソン公式 . . . . .	35
4.6	合成積分公式 . . . . .	36
4.7	多重積分 . . . . .	39
<b>5</b>	<b>数列</b>	<b>41</b>
5.1	数列のプログラム . . . . .	41

5.2	数列の収束とグラフ . . . . .	45
<b>6</b>	<b>非線形方程式</b>	<b>56</b>
6.1	2 分法 . . . . .	56
6.2	ニュートン法 . . . . .	57
6.3	ニュートン法（非線形連立方程式の場合） . . . . .	60
<b>7</b>	<b>連立 1 次方程式</b>	<b>63</b>
7.1	概説 . . . . .	63
7.2	クラメールの公式 . . . . .	64
7.3	行列の基本変形 . . . . .	65
7.4	ガウスの消去法 . . . . .	65
7.5	ヤコビの反復法 . . . . .	71
7.6	ガウス・ザイデルの反復法 . . . . .	72
<b>8</b>	<b>常微分方程式</b>	<b>75</b>
8.1	常微分方程式とその数値解 . . . . .	75
8.2	1 階常微分方程式 . . . . .	76
8.3	連立 1 階常微分方程式 . . . . .	81
8.4	線形 2 階常微分方程式 . . . . .	85
<b>9</b>	<b>シミュレーションのための知識</b>	<b>90</b>
9.1	疑似乱数の発生 . . . . .	90
9.2	一様乱数から特定の確率分布の発生 . . . . .	91
9.3	再帰 . . . . .	93
9.4	列挙の方法 . . . . .	94
<b>10</b>	<b>プログラムの例（問題のヒント）</b>	<b>97</b>
10.1	計算と表示 . . . . .	97
10.2	数学関数 math.h の使用 . . . . .	98
10.3	scanf による入力 . . . . .	99
10.4	if 文 . . . . .	101
10.5	for 文と $\sum$ の計算 . . . . .	102
10.6	for 文と $\sum$ の計算（その 2） . . . . .	103
10.7	関数の作成 . . . . .	104
10.8	for 文と break . . . . .	107
10.9	1 次元配列 . . . . .	108
10.10	for 文を作るときの考え方 . . . . .	109
10.11	2 次元配列 . . . . .	110
10.12	ファイルの読み書き . . . . .	111
10.13	sizeof 演算子 . . . . .	112

<b>11 プログラミングについて</b>	<b>113</b>
11.1 C 言語の要点	113
11.2 gcc のオプション	119
11.3 プログラムのデバッグ（修正）の基本	120
11.4 C 言語での注意点	121
11.5 学生舎でのプログラミング：Windows に Linux 環境を構築 (WSL)	125
11.6 学生舎でのプログラミング:MinGW	126
11.7 Mathematica の利用法	127

## 問題一覧

問題 2.1 簡単な計算	16
問題 2.2 分数の計算	16
問題 3.1 $\Sigma$ の計算（まるめ誤差）	19
問題 3.2 2 次方程式の解（桁落ち）	21
問題 3.3 和の順序（積み残し）	27
問題 3.4 相加平均・相乗平均（ $\Sigma$ の計算・ $\Pi$ の計算）	31
問題 3.5 最大・最小	32
問題 4.1 数値積分	35
問題 4.2 合成積分公式	38
問題 5.1 数列	41
問題 5.2 数列の収束	45
問題 6.1 2 分法・ニュートン法	58
問題 7.1 ガウスの消去法（連立 1 次方程式）	68
問題 7.2 ヤコビの反復法・ガウスザイデルの反復法（連立 1 次方程式）	73
問題 8.1 オイラー法・ルンゲクッタ法（微分方程式）	78

## 1 はじめに

自然現象や社会現象などをコンピュータで計算する場合には、多くの場合実数（正確には浮動小数点数）を用いる。また、数学で使われる三角関数・対数関数・指数関数も浮動小数点数の演算を用いている。この数値計算の授業では、さまざまな場面で現れる浮動小数点の計算を扱う。また情報工学科で必要とされることが多い事項については、一部整数の計算についても扱う。

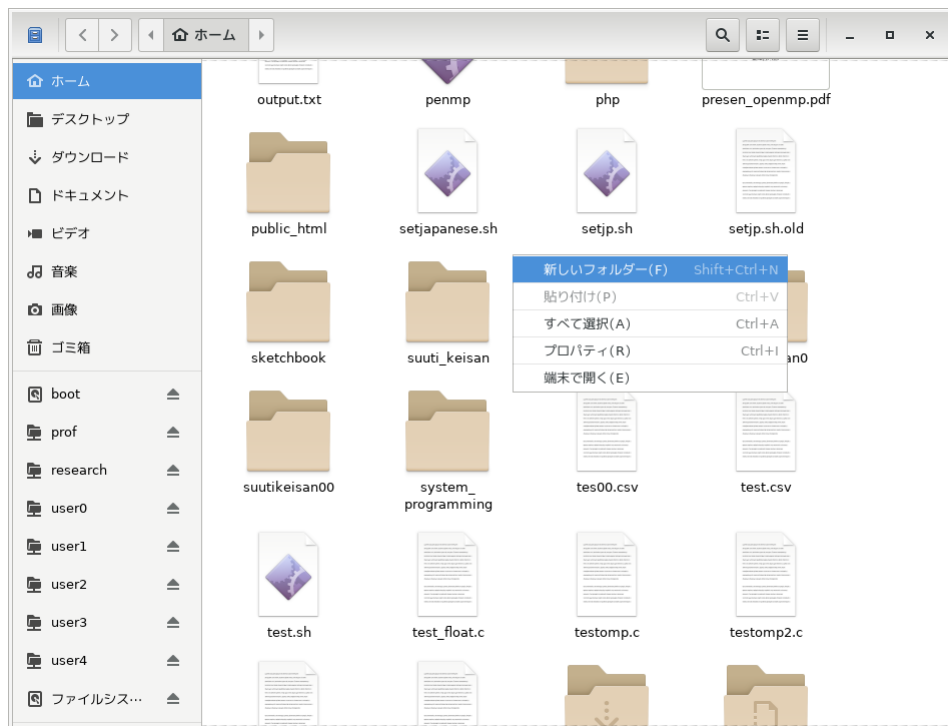
### 1.1 参考書

この講義では理論的な背景についてはあまり取り扱わない。また、解が正しく求まるためにはいくつか条件が必要なものもあるが、これについても詳しく述べていない。これらについて知りたい場合には、以下の本などを参考にせよ。いずれも本校図書館で見ることができる。

- 山本哲朗, 数値解析入門, サイエンス社
- 森 正武, 数値解析, 共立出版

### 1.2 演習の準備

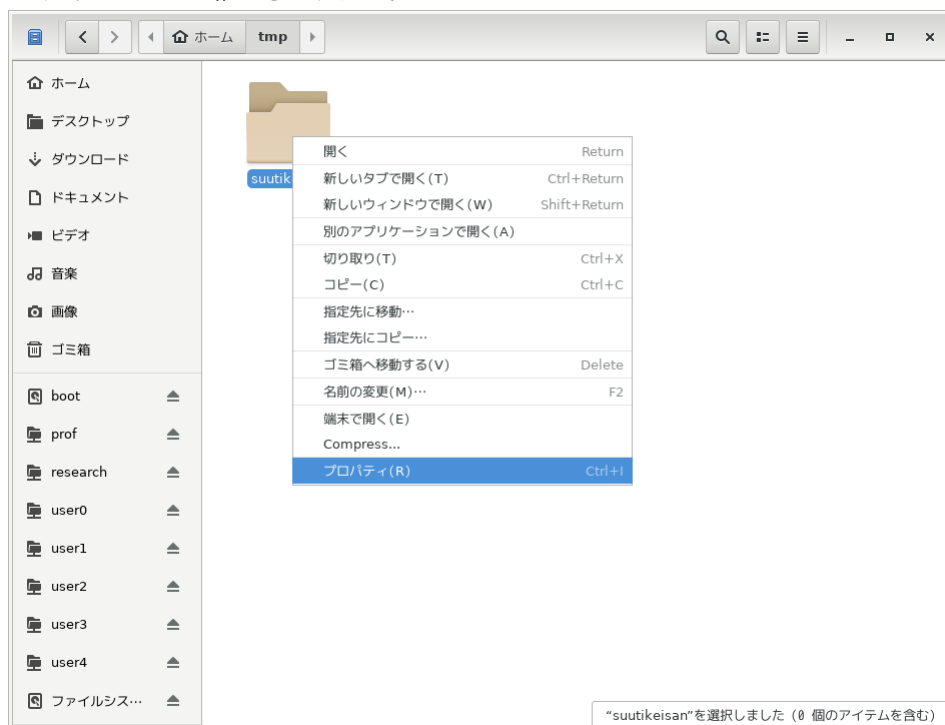
- Linux の開始  
まず、7 ページの「授業の操作手順」に従ってログインする。8 ページの項目 11 まで実行する。
- フォルダの設定（2 回目以降はこの作業は不要）
  - この授業のディレクトリ（フォルダ）を作成する。  
ログインして、コマンドラインから「`nautilus --no-desktop &`」とすると、以下の画面が出るはず。もしエラーが出るようであれば、7 ページの、「TeraTerm で、「設定」→「SSH 転送」」からやりなおす。



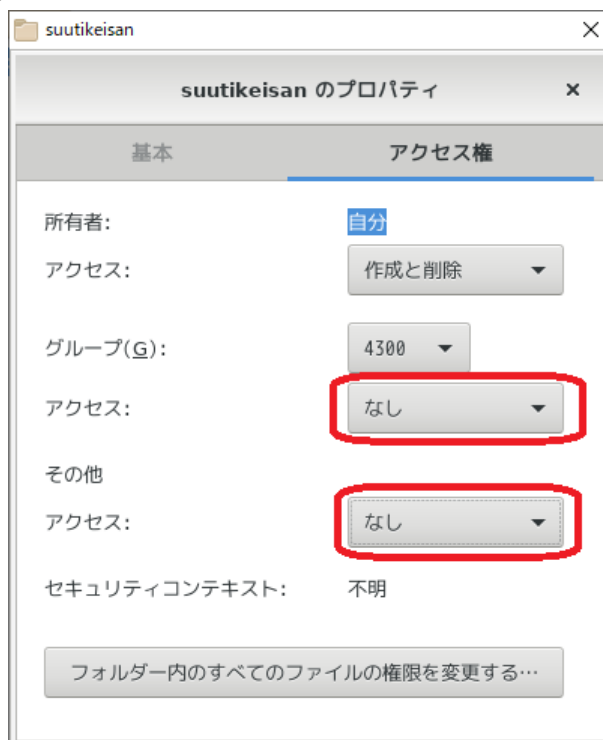
フォルダ名は suutikeisan とする（別の名前でも良いが、英数字のみにすること。漢字やひらがな等の文字は使わないこと）。



- このディレクトリを他人からは読めないようにする．作成した suutikeisan フォルダで右クリック  
→プロパティ→アクセス権 をクリックする．



以下の画面と同じように「グループのアクセス権を『なし』」, 「その他のアクセス権を『なし』」と設定する．所有者は「自分」であることを確認する．設定後にもう一度アクセス権を見て，設定できていることを確認する．



- コマンドラインで `gedit &` と入力して、テキストエディタを起動する。起動しなければ、7 ページの、「TeraTerm で、「設定」→「SSH 転送」からやりなおす。
- 初回の設定（2 回目以降は不要）
  - `gedit` のツールバーで、編集→設定とし、「行番号を表示する」にチェックを入れる。
- ファイルを開くときは、ファイルをダブルクリックするのではなく、まずテキストエディタを起動して、メニューの「ファイル→開く」を用いること。  
他に使いたいエディタがあればそれでも良い。
- Linux の画面下に並んでいるアイコンの中から黒い四角の「ターミナルエミュレータ」を起動する。または、アプリケーション→ターミナルエミュレータ を起動する。  
プロンプトで「`cd suutikeisan`」としてディレクトリを変更する。ディレクトリ名を変えた場合には「`cd` 上で作成したディレクトリ名」とする。

### 1.3 Linux コマンドの簡単な復習

- `cd` : 現在のディレクトリを変更 (change directory の略)
- `ls` : 現在のディレクトリ (フォルダ) のファイルの一覧
- `ls -l` : `ls` の表示に、ファイルの書き込み禁止などの属性を加える。
- `cc` : C 言語のソースファイルのコンパイル

### 1.4 授業の操作手順

1. Windows のメニューまたはデスクトップのアイコンから TeraTerm を起動する。
2. ホストに、
  - 学籍番号 (s\*\*\*\*\*) の最後が...
  - 奇数の学生 : `sedplix01.nda.ac.jp`
  - 偶数の学生 : `sedplix02.nda.ac.jp`
 として、TCP ポートは 22 として (通常は 22 が入っているはず)、サービスは SSH にして、SSH バージョンを SSH2 にして、プロトコルを UNSPEC にして、OK を押す。混んでいてうまく接続できない場合には、もう一つのホスト名にして接続する。  
それでも接続できなければ、上記のホストの IP アドレスを用いる。ホストに、10.0.0.145 または 10.0.0.146 とする。
3. 「セキュリティ警告」が出たら、ホスト名 `sedplix...` または `10.0.0....` が正しいことを確認し、「このホストを known hosts リストに追加する」にチェックを入れて、「続行」を押す。
4. ユーザ名とパスフレーズに、自分のユーザ名とパスワードを入れる。「プレインテキストを使う」にチェックが入っていることを確認して、OK を押す。
5. これでログインできるはず。できなかったら TeraTerm の起動からやり直す。
6. TeraTerm で、「設定」→「SSH 転送」→「リモートの X アプリケーションをローカルの X サーバに表示する」にチェックを入れて OK を押す。
7. TeraTerm で「設定」→「設定の保存」とし、「保存」を押して設定を保存する。ファイル名は

TERATERM.INI から変更しないこと.

8. TeraTerm のプロンプトで, `exit` として, ログアウトして, TeraTerm を終了させる. 上記の「リモートの...」の設定は, もう一度起動したときから有効になるので, 一度 TeraTerm を終了させる必要がある.
9. Astec-X を起動する (Windows のメニューにあるはず). 起動すると, タスクバーに Astec-X のアイコンが現れる.
10. TeraTerm を起動する. 「設定」→「SSH 転送」を開いて, 「リモートの X アプリケーションをローカルの X サーバに表示する」にすでにチェックが入っていることを確認する (ここでチェックを入れてはダメ). もしチェックが入っていなかったら, 上記の 6, 7, 8 をもう一度行った後, TeraTerm を再起動して, 上記のチェックが入っていることを確認する.
11. TeraTerm で, 上記の 2, 4 と同様にして Linux マシンに接続する.
12. `cd` を使って, 目的のディレクトリに移動する.
13. 「`gedit &`」とすると, GNOME エディタが起動する. 最後の `&` を忘れないようにすること.
14. 「`gnome-terminal &`」とすると, ターミナル (端末) 画面が起動する. 最後の `&` を忘れないようにすること.
15. フォルダやファイルを GUI で操作したい場合には, プロンプトで「`nautilus --no-desktop &`」とする. ダブルクリックでファイルを開いたり, 右クリックで別のアプリケーションを指定して開くことができる. ただし Windows と Linux との間でドラッグアンドドロップはできない.



## 1.5 課題の提出要領

課題を提出するときは、以下の要領で行う。

- 授業中には必ず一つ以上提出すること。なにも提出がない場合には欠席とみなす。
- 問題はすべて提出して、教官の合格をもらうこと。追加問題とレポートは、提出しなくても良いが、提出して合格をもらえば成績が良くなる。
- 提出先は、チームスの「情報工学演習 2」の中の、「番号\_問題提出\_学生氏名」のチャンネルとする。番号には各学生の番号が入る。「00\_問題提出サンプル」に提出のサンプルがある。また、「00\_ヒントと注意」のチャンネルにはヒントと注意がある。  
「情報工学演習 II\_2024\_前期...」というチームもあるが、そこではないことに注意する。
- 提出するソースファイルは、問題番号と提出した日付がわかるようにする（作成した日付ではない）。  
例えば、問題 2.1 について、2024 年 4 月 1 日に提出する場合、prob2\_1\_20240401.c などとする。  
もし、同じ日に同じ問題で複数提出する場合（授業中と自習時間など）、2 番目の提出は最後に「\_2」をつけて、prob2\_1\_20240401\_2.c などとする。
- 実行画面のスクリーンショットは、ソースファイルと同じ名前でも拡張子のみ png にする。セーブするときに png 形式にすること。prob2\_1\_20240401.png などとする。
- ファイルを添付したときに、「このチャンネルで同じ名前のファイルが共有されるためコピーをアップロードしています」とでたら、ファイル名が不適切なので、上記に従ってファイル名を変更してから添付しなす。「代わりに古いファイルを再共有する」はクリックしてはいけない。
- 新しい問題を提出するとき
  - 「番号\_問題提出\_学生氏名」のチャンネルで、「投稿を開始する」をクリック
  - 「件名を追加」のところに、問題番号を入れて「問題 2.1」などとする。追加問題の場合は「追加問題 2.1」などとする。
  - メッセージを入力するところに、「完成しました」「途中です」などのコメントを入れて、以下の完成したか未完成かに従う。
- 同じ問題を再提出するとき
  - 採点者から指摘された事項について修正すること。修正を繰り返すことで、指摘なしで完成度の高いプログラムが組めるようになる。
  - すでに提出した問題の「返信」のところに、「途中です」「完成しました」「修正しました」などのコメントを入れて、以下の完成したか未完成かに従う。指摘事項が良くわからない場合には質問すること。
- 完成した（と思われる）場合、または質問がある場合（コンパイルエラーで動かないなど）の提出
  - メンションを用いる。「メッセージを入力（新しい問題）」もしくは「返信（同じ問題）」のところで、「@問題提出\_自分の氏名」とする（提出先に「@」をつける）と、上に候補が出るので、それをクリックすると「番号\_問題提出\_自分の氏名」が青色になる。その後、上記のコメントを記入する。メンションを用いない場合には、教官からの返信が来ない可能性がある。
  - ソースファイルをそのままドラッグアンドドロップ
  - 「コードスニペット </>」をクリックして（返信の場合には左下の「形式 A」をクリックしてから

コードスニペット `</>` をクリック), 左上の「タイトルの追加」に問題番号 (問題 2.1 など), 右上を Text ではなく「C」に設定して, ソースファイルをドラッグアンドドロップする. ソースファイルは 2 つ提出することになる.

- 実行画面のスクリーンショット (上記のファイル名), または実行結果をテキストにして添付する (作成方法は第 2 章を参照).
- 最後に, 「投稿」または「送信ボタン (右端の三角)」を押す. 送信した後に, メンションの記号として, 送信の右端に赤丸のマークが出るので確認する.
- 未完成の場合は,
  - ソースファイルのスクリーンショットまたは, コードスニペット `</>` (返信の場合には左下の「形式 A」を押してから`</>`を押す) で, ソースファイルを提出する.
  - 最後に, 「投稿」または「送信ボタン (右端の三角)」を押す.
- 適宜, 提出された課題に対する質疑応答を教官が行う. そこで適切に答えられない場合, コピーペーストと判断するので, 自分の力で課題に取り組むこと.
- コピーペーストが判明した場合, コピー元の学生もコピー先の学生も「不可」とするので注意すること.
- Windows や Linux といった OS の違い, gcc や clang といったコンパイラの違いに影響されないソースコードを作成するために, 共同利用計算機の Linuxにおいて`cc, g++`のいずれを用いてもコンパイルできることを【必要条件】とする.
  - 関数の内部で宣言される関数 (いわゆる入れ子関数)
  - for ループにおけるカウンタ変数のインライン宣言 (`for (int k=0; k<n; k++)`) のような宣言などは使用してはならない.
- `cc -O -Wall ソースファイル名` または `cc -lm -O -Wall ソースファイル名` などとしてコンパイルして (0 は数字ではなくアルファベットの大文字) 警告が出ないようにすること. 特に「初期化されずに使用されています」という警告が出ないようにすること.

上記が厳密に守られていない場合には, 未提出とする.

メールなどで合格が出たら, 授業中などに教官にプログラムの各行について口頭で説明すること. そのあと教官から質問があるので, それに口頭で正しく答えられたら合格となる.

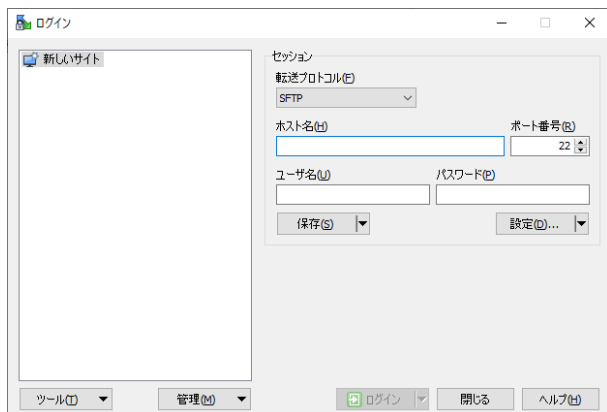
プログラムのソースファイルを Windows に転送するには、Windows のエクスプローラを使う方法と WinSCP を使う方法の 2 種類ある。

#### Windows エクスプローラを使う方法

1. エクスプローラ（インターネットエクスプローラではない）の一番上に `scmvfs00.nda.ac.jp` と入力する。
2. もしパスワードを聞いてきたら、  
ユーザ名： `eds*****`（\* には学籍番号を入れる）  
パスワード：自分のパスワード  
とすればログインできるはず。
3. 学籍番号 (`s*****`) の右端の数を 5 で割った余りが 0 の場合、フォルダ `user0` をクリックする。  
同様に、  
学籍番号の右端の数を 5 で割った余りが 1 の場合、`user1`  
学籍番号の右端の数を 5 で割った余りが 2 の場合、`user2`  
学籍番号の右端の数を 5 で割った余りが 3 の場合、`user3`  
学籍番号の右端の数を 5 で割った余りが 4 の場合、`user4`  
をクリックする。
4. 上記のフォルダの中の、自分の学籍番号のフォルダをクリックして、その中の `suutikeisan` をクリックする。あとは通常のフォルダのようにファイルを扱うことが可能なはず。（直接ダブルクリックすると Excel 等がうまく動かない場合は、Windows にファイルをコピーしてからダブルクリックすると良い）

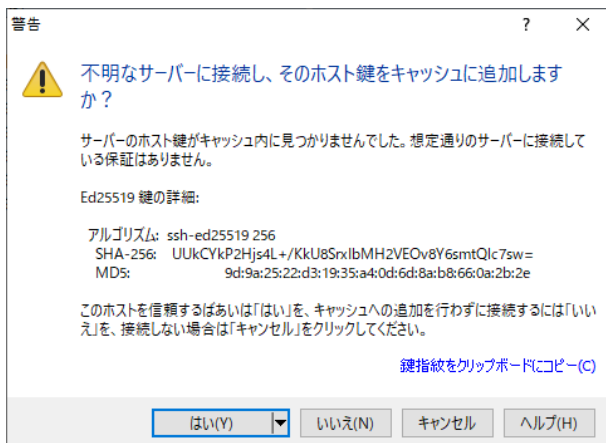
#### WinSCP を使う方法

1. Windows のメニューから WinSCP を起動する。

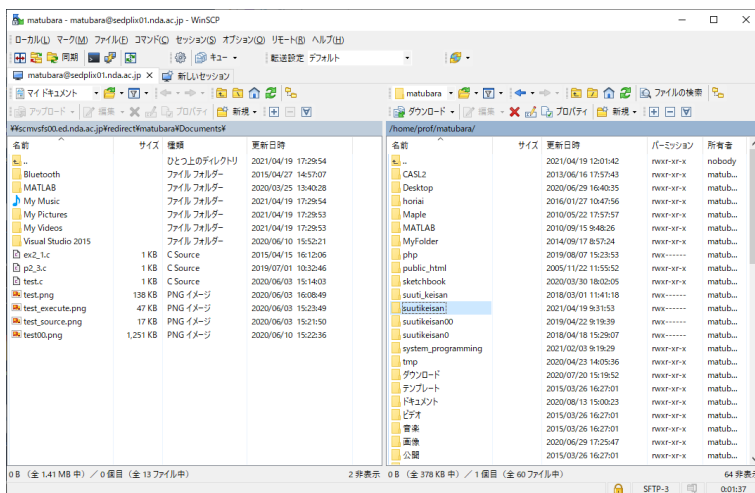


ホスト名に `sedplix01.nda.ac.jp` または `sedplix02.nda.ac.jp` を入れて、自分のユーザ名、パスワードを入れて、「ログイン」をクリックする。

2. 以下の画面が出たら、「はい」を押す。

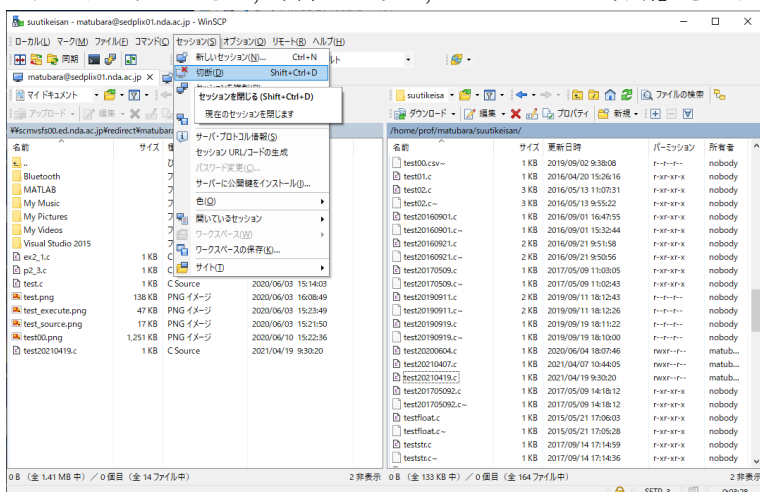


3. 以下のような画面が出る。左側が Windows の画面, 右側が Linux の画面である。右側で, **suutikeisan** をクリックする。



右側から目的のファイルを左側にドラッグ&ドロップすると、ファイルが Windows に転送される。

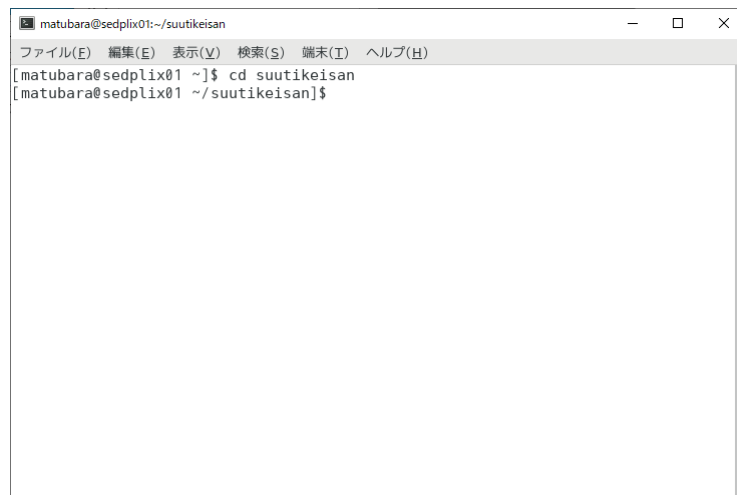
4. ログアウトするときは、以下のように、「セッション→切断」をクリックする。



5. Windows のドキュメントなどに目的のファイルがあるので、それを校内メールに添付する。

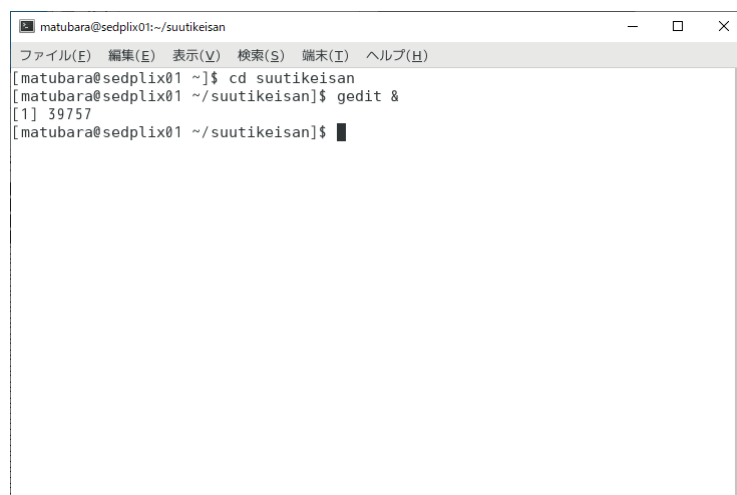
## 2 簡単なプログラムと実行方法

コマンドプロンプトで以下のように `cd suutikeisan` などですべて初めに作成したディレクトリに移動する。(コマンドプロンプトは、環境によっては色が違う場合もある.)

A terminal window titled 'matubara@sedplx01:~/suutikeisan'. The menu bar includes 'ファイル(E)', '編集(E)', '表示(V)', '検索(S)', '端末(T)', and 'ヘルプ(H)'. The command history shows: '[matubara@sedplx01 ~]\$ cd suutikeisan' and '[matubara@sedplx01 ~/suutikeisan]\$'.

```
matubara@sedplx01:~/suutikeisan
ファイル(E) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)
[matubara@sedplx01 ~]$ cd suutikeisan
[matubara@sedplx01 ~/suutikeisan]$
```

コマンドラインで `gedit &` を入力してエディタを起動する。最後の `&` を忘れないようにする。

A terminal window titled 'matubara@sedplx01:~/suutikeisan'. The menu bar is the same as the previous image. The command history shows: '[matubara@sedplx01 ~]\$ cd suutikeisan', '[matubara@sedplx01 ~/suutikeisan]\$ gedit &', and '[1] 39757'. The prompt is now '[matubara@sedplx01 ~/suutikeisan]\$' followed by a cursor.

```
matubara@sedplx01:~/suutikeisan
ファイル(E) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)
[matubara@sedplx01 ~]$ cd suutikeisan
[matubara@sedplx01 ~/suutikeisan]$ gedit &
[1] 39757
[matubara@sedplx01 ~/suutikeisan]$
```

エディタに以下を入力する。

バックスラッシュ記号\は¥キーで入力できる。使用するエディタや設定環境によっては¥が表示される。

左端の空白（インデント）は、タブキー (Tab) を用いて入力すること。

インデントにスペースは使用しない。

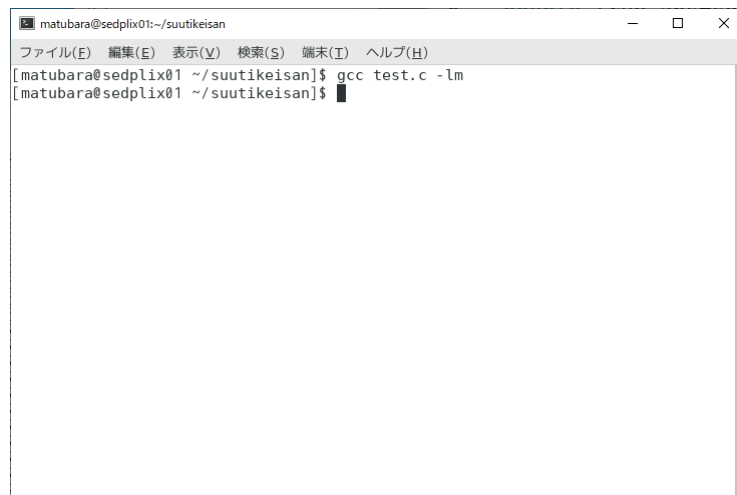
//より右はコメントなので入力する必要は無い。

\は¥キーで入力する。

```
#include <stdio.h>
#include <math.h>
int main(){
    double x=M_PI/6.0;
    double y=cos(x);
    double true_ans=sqrt(3.0)/2.0; // 値はこういふ変数に代入してから表示する
    printf("cos(pi/6)=%f\n",y);
    printf("sqrt(3)/2=%f\n",true_ans);
    printf("error=%f\n",y-true_ans); // error の意味は「誤差」
    return 0; // int main() は正常終了
}
```

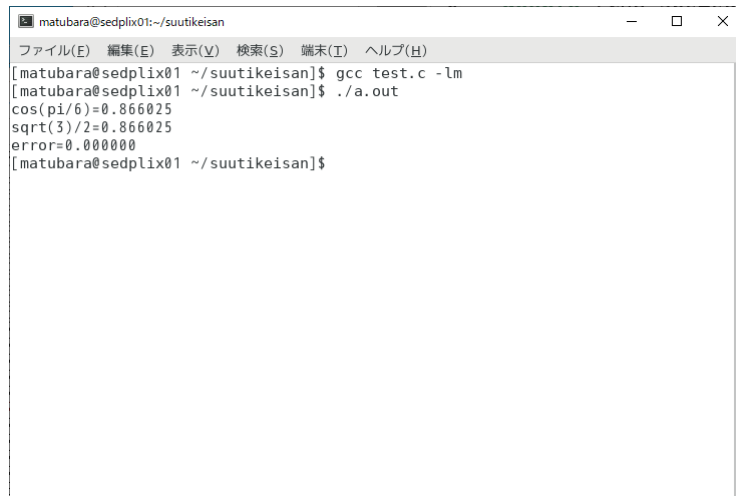
入力したら、test.c などの名前でセーブする（別の名前でも良いが、拡張子は .c にすること）。

以下のようにコンパイルする。コマンドラインに gcc test.c -lm と入力する。math.h をインクルードするときは、-lm オプションが必要なことに注意する。



コンパイルエラーが出たら、メッセージに従ってファイルを修正する。正しくコンパイルできれば、上記のように何もメッセージが表示されない。

コンパイルしたものを実行する。コマンドラインに ./a.out と入力すると以下ようになる。



$\cos\left(\frac{\pi}{6}\right) = \frac{\sqrt{3}}{2}$  となるはずであるが、 $\cos(\pi/6)=0.866025$  と  $\sqrt{3}/2=0.866025$  が同じ値なので、正しく計算できていることがわかる。また、 $\text{error}=0.000000$  ということからも、正しいことがわかる。

この授業では、出力およびコンパイルについて提出してもらうが、その方法を3つ示す。

1. Windows アクセサリの Snipping Tool を起動して、新規作成で、エディタと実行画面を選択する。もし画面の中心に不必要な残像が出る場合には、「遅延」のところを「2 秒」に設定する。  
名前を付けて保存→ファイルの種類を PNG にする。ファイル名は `snip20240415.png` などとする（数字は今日の日付を示す。2024 年 4 月 15 日の場合）。
2. 以下のように入力する。\$ はコマンドプロンプトなので入力しない。数字は上記と同様に今日の日付。

```
$ script scr20240415.txt
$ gcc -lm test.c
$ ./a.out
$ exit
```

このようにすると、`scr20240415.txt` ができるはず。ただし、途中でキーボードの↑や↓などを使うと、ファイルに余計な文字が入ってしまうので、これらは使わないこと。同様に、途中で入力を間違った場合には余計な文字が入るので一度 `exit` して、最初からやり直すこと。

3. 以下のように入力する。\$ はコマンドプロンプトなので入力しない。> と >> は異なるので注意する。

```
$ cc -lm test.c > & result20240415.txt
$ ./a.out >> & result20240415.txt
$ history 3 >> & result20240415.txt
```

このようにすると、`result20240415.txt` ができるはず。

途中で入力を間違ってしまった場合には、上記を最初から入力しなおすこと。ただし、最下行の `history 3` の 3 を変えることで対応することも可能である。

**問題 2.1** 上記の例と同様にして、 $\sin\left(\frac{\pi}{4}\right)$  と  $\frac{\sqrt{2}}{2}$  をプログラムで計算して、 $\sin\left(\frac{\pi}{4}\right) = \frac{\sqrt{2}}{2}$ であることを示せ.

(問題終)

コンパイルオプションについて 119 ページ (第 11.2 節) を見ておくこと.

**問題 2.2**  $\frac{1}{2}$  の値を計算したい. 答えは 0.5 になるはずである.

また、 $c = 2$ ,  $d = 3$ ,  $e = 4$ ,  $f = 6$  に対して  $\frac{cd}{ef}$  を計算したい. 答えは  $\frac{2 \times 3}{4 \times 6} = \frac{1}{4} = 0.25$  になるはずである.

次のプログラムを入力して実行する.

```
#include <stdio.h>
int main(){
    double a=1/2;
    double b=1.0/2.0;
    printf("a=%f b=%f\n",a,b);

    double c=2.0, d=3.0, e=4.0, f=6.0;
    double x=c*d/e*f;
    double y=c*d/(e*f);
    double z=c*d/e/f;
    printf("x=%f y=%f z=%f\n",x,y,z);

    return 0;
}
```

上記を入力して実行せよ.  $a, b$  でどちらが正しいだろうか. 間違っているのはどちらか? その理由についてメールに記述すること.  $x, y, z$  でどれが正しいだろうか ( $x, y, z$  のうち, 間違いは一つで正しいものが2つ).

その結果を理解して, 上のプログラムの `return 0` の直前に, 次の式が正しく計算できるように計算式と表示を追加せよ.

$$s = 1/3, t = 4/3$$

$$p = 2 \frac{s^2 + 1}{5t}$$

値は変数  $s, t$  に代入して,  $p$  の計算は  $s, t$  を用いて行う.  $\frac{1}{3}$  や  $\frac{4}{3}$  は  $p = \dots$  の計算式には直接用いない.  $s^2$  の計算は  $s*s$  とする.



$$p = 2 \frac{\left(\frac{1}{3}\right)^2 + 1}{5 \times \frac{4}{3}} = 2 \frac{\frac{1}{9} + 1}{\frac{20}{3}} = \frac{\frac{20}{9}}{\frac{20}{3}} = \frac{1}{3}$$

なので、 $p=0.3333\dots$  となるはずである。

(問題終)

### 3 数値計算の基礎

#### 3.1 誤差および誤差の伝搬

測定値や計算値には、一般に次のような誤差が含まれている。

1. 測定値の数値そのものに含まれている誤差
2. 数式の取り扱いによって発生する誤差
3. 計算機の構造に起因する誤差

このうち、1 について改善するには、測定器の性能を良くしたり、測定法を変えたりすることなどが考えられるが、この授業では取り扱わない。

2, 3 については、計算の方法によって誤差を小さくできる可能性がある。

誤差をなるべく小さく計算するというのが数値計算の重要な課題である。

$a$  を真の値とする。これを「真値」と呼ぶ。 $x$  を  $a$  の近似値とすると、 $\Delta x = x - a$  を近似値  $x$  の「誤差」、 $|\Delta x|$  を「絶対誤差」、 $\Delta x/a$  を「相対誤差」という。 $|\Delta x| \leq \varepsilon$  となる  $\varepsilon$  を誤差の限界と呼ぶ。このとき、

$$x - \varepsilon \leq a \leq x + \varepsilon$$

となり、 $a = x \pm \varepsilon$  と略記する。

第 2 項の誤差の一例として、ある関数  $z = f(x, y)$  の値を  $(x, y) = (a, b)$  で計算することを考える。 $x, y$  に誤差（上記第 1 項による誤差など）が含まれており、

$$x = a \pm \varepsilon_x, y = b \pm \varepsilon_y$$

とし、 $z = f(x, y)$ ,  $z = c \pm \varepsilon_z$ ,  $c = f(a, b)$  とする。 $a, b, c$  は真値である。

- $z = x + y$  のとき、

$$\varepsilon_z \leq \varepsilon_x + \varepsilon_y$$

となる。すなわち、加算の誤差の限界は、それぞれの誤差の限界の和になる。

- $z = x - y$  のとき、

$$\varepsilon_z \leq \varepsilon_x + \varepsilon_y$$

となる。すなわち、減算の誤差の限界も、それぞれの誤差の限界の和になる。

- $z = x \cdot y$  のとき、

$$\frac{\varepsilon_z}{c} \leq \frac{\varepsilon_x}{a} + \frac{\varepsilon_y}{b}$$

となる。すなわち、乗算の相対誤差はそれぞれの相対誤差の和になる。

- $z = x/y$  のとき、

$$\frac{\varepsilon_z}{c} \leq \frac{\varepsilon_x}{a} + \frac{\varepsilon_y}{b}$$

となる。すなわち、除算の相対誤差もそれぞれの相対誤差の和になる。

これらは、微分を用いて証明できるが、ここでは省略する。

演算回数が増えるほど結果の誤差は増えるので、式や計算手順（アルゴリズム）を工夫することにより演算回数を少なくすることが重要である。

## 3.2 有効数字

扱う数値がどこまで正しいかを示すのが「有効数字」である。

次の数字はいずれも有効数字が 5 桁の数値である。

12300, 123.45, 0.0012345.

また、次のような表記も用いる。

$1.2300 \times 10^5$  : 有効数字 5 桁

$1.23 \times 10^5$  : 有効数字 3 桁

## 3.3 まるめ誤差

計算機の中では、実数は一定桁数の 2 進数に変換される。

2 進数の有効数字  $r$  桁の数値は

$$\pm 1.d_1d_2\dots d_r \times 2^e, d_k \in \{0, 1\}, k = 1, 2, \dots, r$$

と表現される。

$1.d_1d_2\dots d_r$  を仮数部,  $e$  を指数部と呼ぶ。2 進数なので仮数部の整数部分は 1 になることに注意せよ。

通常, C 言語の float では, 符号部 (±)1 ビット, 仮数部 23 ビット, 指数部 8 ビットの合計 32 ビット, double は, 符号部 (±)1 ビット, 仮数部 52 ビット, 指数部 11 ビットの合計 64 ビットである。

10 進数で有限桁の小数を 2 進数に変換した場合, 無限桁が必要になる場合がある。例えば,

$$(0.1)_{10} = (0.00011001100110011\dots)_2$$

となる。 $(*)_{10}$  は 10 進数表示,  $(*)_2$  は 2 進数表示である。右辺の 2 進数表示は循環小数となり, 正確に表現するには無限桁が必要になる。コンピュータでは有限桁しか扱えないので, 無限桁をどこかの有限桁で切り捨て (または 0 捨 1 入) することになる。これによる誤差を「まるめ誤差」という。

まるめ誤差を少なくするために, 仮数部の桁数を多くすることがある。float ではなく, double を使う, もしくは long double を使う, または多倍長演算ライブラリ (GMP など) を用いて double や long double より多い桁数の変数を使うなどの方法がある。

三角関数 (sin, cos, tan など), 平方根 ( $\sqrt{x}$ ), べき乗 ( $x^y$ ) などの数学関数は float, double, long double で関数名が異なることがあるので注意せよ。例えば, tan の場合には, Linux のコマンドラインで `man tan` とすると, float の場合は `tanf`, double の場合は `tan`, long double の場合は `tanl` が関数名であることがわかる。

**問題 3.1** 次のプログラムは,  $n = 16$  に対して,  $s = \sum_{i=0}^{n-1} \frac{1}{n}$  を計算するプログラムである。  $\sum_{i=0}^{n-1} \frac{1}{n} = 1$  なので ( $\frac{1}{n}$  を  $n$  個足す),  $s = 1$  になるはずである。

```
#include <stdio.h>

int main(){
    int n=16;
    float s;
    int i;
    s = 0.0; // s の初期化. for 文の直前で行う
    for(i=0; i<n; i++){ // 「i=0 から開始」して, 「i<n を満たす限り」for 文を続ける.
        // 最後の i++ は「各ステップで, i を 1 つ増やす」
        s = s + 1.0/(float)n; // s に 1/n を加えていく
    }
    printf("i:%d s:%.8e\n", i, s); // .8e は, 小数点以下 8 桁で, 指数表示 (e)
    return 0;
}
```

指数表示については,  $1.000\text{e}+00$  は,  $1.000 \times 10^0 = 1.000$  を意味する. 例えば,  $1.23\text{e}-02$  は,  $1.23 \times 10^{-2} = 0.0123$  を意味する.

通常は (今後の問題では) `float d=1.0/(float)n;` を for 文の前において, for 文では `s = s + d;` とする方が, 割り算の回数が少なくなるので望ましい. ただし, ここではそのようなことは行わない.

上記のプログラムを入力して実行し, 正しく動作することを確認せよ.

次に,  $n$  について 10, 1 000, 1 000 000 および 8, 1024, 8192 の場合について, 上記のプログラムを実行し,  $s$  の値を確認せよ.  $s$  が 1.000 となる場合とならない場合の実行結果を一つずつ選び, ソースコードとスクリーンキャプチャを提出し,  $s$  が 1.000 となる  $n$  はどのようなものか説明せよ. (問題終)

### 3.4 桁落ち

**桁落ち**とは, 非常に近い数値の引き算を含む計算において, 有効数字が大幅に減少してしまうことである.

**例 3.1**  $x = 1.00000$ : 有効数字 6 桁

$y = 0.99999$ : 有効数字 5 桁

とする. このとき

$x - y = 0.00001$ : 有効数字 1 桁

となってしまう.

**例 3.2** 2 次方程式の解の公式を用いて解を求める際に, 0 に近い解がある時も桁落ちが生じることがある.

2 次方程式  $ax^2 + bx + c = 0$  の解の公式は,

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \quad (3.4.1)$$

であるが,  $b > 0, b^2 \gg 4ac$  のときには  $-b + \sqrt{b^2 - 4ac}$  は近い値の引き算になる. ( $\gg$  は非常に大きいの意味)

また、 $b < 0, b^2 \gg 4ac$  のときは、 $-b - \sqrt{b^2 - 4ac}$  は近い値の引き算になる。

したがって  $b^2 \gg 4ac$  のときは、解のひとつで桁落ちが生じる可能性がある。

これを防ぐには、桁落ちを生じない一つの解を求めたのち、解と係数の関係を用いてもう一つの解を求める。解と係数の関係にはいくつかあるが、ここで用いるのは、2つの解を  $x_1, x_2$  とするとき、 $x_1 x_2 = c/a$  となる関係である。

すなわち、次のようにすればよい。

$$x_1 = \begin{cases} \frac{-b - \sqrt{b^2 - 4ac}}{2a}, & (b \geq 0) \\ \frac{-b + \sqrt{b^2 - 4ac}}{2a}, & (b < 0) \end{cases} \quad (3.4.2)$$

$$x_2 = \frac{c}{ax_1} \quad (3.4.3)$$

上記のようにすれば、 $x_1$  の分子は同符号の加算になるので桁落ちは生じない。

**問題 3.2** (p.97, p.98, p.101 参照) 例 3.2 についてプログラムを作成し、桁落ちがある場合とない場合の結果を比較せよ。すなわち、解の公式を普通に用いた結果と、例 3.2 のように計算した結果を比較せよ。式 (3.4.2) は if 文を用いて作ること (p.101)。式 (3.4.3) は if 文の外に置く。

変数には `float` を用いること。また、 $a \neq 1$  かつ  $a \neq c$  と設定せよ。  $0.1 \leq a \leq 0.4$ ,  $100 < |b| < 1000$ ,  $0.5 \leq c \leq 0.9$ , を目安に設定せよ。値はソースファイルに直接書き込んで良い。 `scanf` を使用する必要はない (使用しても良い)。これらに関して if 文で判定する必要はない。if 文は  $b$  の正負の判定のみで良い。

C 言語では、 $x_1^2$  については `x1*x1` として作成せよ。

$\sqrt{a}$  は C 言語では `sqrt(a)` とする。ただしこれを使うには、`#include <math.h>` が必要になる (p.98 参照)。

式の数値 2 などを 2.0 と記述しないこと。このように記述すると、`double` となり桁落ちが生じにくくなる。数値は、2 または 2.0f などとすること。これらは `int` または `float` となる。

さらに、Linux でコンパイルする際には、

```
cc ソースファイル名 -lm
```

もしくは

```
gcc ソースファイル名 -lm
```

のように、`-lm` オプションをつける必要があることに注意する。

表示は、まず  $a, b, c$  を表示せよ。次に、解の公式をそのまま用いて計算した解  $x_1, x_2$ 、および例 3.2 のようにして計算した  $x_1, x_2$  を表示し、それらを比較せよ。

また、解を  $x_1, x_2$  とした場合、解と係数の関係より、 $x_1 + x_2 = -\frac{b}{a}$ 、すなわち  $x_1 + x_2 + \frac{b}{a} = 0$  となる。この左辺も表示する。もし同じ解を 2 つ重複して求めるバグがある場合、 $ax^2 + bx + c$  に代入した結果は 0 に近くなるが、解と係数の関係の式は 0 に近くならない (例外はあるが)。

また、求めた解をもとの 2 次方程式に代入した結果 (全部で 4 つ) を表示せよ。例えば、求めた解の一つを  $x_1$  とすると、 $ax_1^2 + bx_1 + c$  を計算せよ。これを求めた解のすべてについて行い、値を表示せよ。

どの解を代入した結果であるかがわかるように表示せよ。解をもとの 2 次方程式に代入すると、ほぼ 0

になるはずである。この値が 0 に近いほうが、解の精度が高いと考えられる。桁落ちがある場合と、ない場合のどちらの精度が高いか？

すなわち、表示は

- $a, b, c$  の値 ( $b^2 \gg 4ac$ ,  $a \neq 1$ ,  $a \neq c$  となっているか各自で確認せよ)
- 解の公式をそのまま用いた 2 つの解  $x_1, x_2$
- 桁落ちを減らした 2 つの解  $x_1, x_2$
- 解の公式の 2 つの解に対して  $x_1 + x_2 + \frac{b}{a}$  の値.
- 桁落ちを減らした 2 つの解に対して  $x_1 + x_2 + \frac{b}{a}$  の値.
- 解の公式をそのまま用いた 2 つの解を、もとの 2 次方程式に代入した値 (2 つ表示)
- 桁落ちを減らした 2 つの解を、もとの 2 次方程式に代入した値 (2 つ表示)

とする (表示サンプルが下にある)。

この実行結果を、 $b > 0$  と  $b < 0$  の場合について (単に  $b$  のプラスマイナスを変更) 提出せよ。

数値は、変数に代入してその変数を `printf` で表示する。printf 内で計算は行わない。すなわち、`printf("x1=%f\n", (sqrt(... などせずに, x1=...; printf("x1=%f\n", x1); などとすること。`

よくわからない場合には、

- 2 次方程式の解の公式、式 (3.4.1) の値 (2 つ) を求める。
- p.101 のプログラムを作成して動作確認。
- その if 文の条件と、if 文の中身を式 (3.4.2) に変更。p.101 では 3 つの条件だが、ここでは 2 つの条件なので注意する。適切に変数を追加 (上記にあるように `float` を用いる)。

式 (3.4.3) はまだ作成しない。

$\sqrt{a}$  は `sqrt(a)` となる。

p.98 を参照して、`#include <math.h>` を入れて上記のコンパイル方法 (オプション `-lm`) を用いる。

数式では積の記号が省略されているところがあるが、プログラムでは\*が必要なので注意すること。

式の分母については p.121 を見る。

- 式 (3.4.3) を if 文が終わったすぐ後に追加する。
- さらにその下に、上記にあるように `printf` で表示を行う (p.97 参照)。

出力は以下のようにすること。(日本語の設定はややこしいので、アルファベットで出力)

出力

```
a=.....
b=.....
c=.....
x1(original)=..... // 変数名は以下の注を参照
x2(original)=..... // original の意味は「元の」
x1(high_accuracy)=..... // accuracy は「精度」、high accuracy は「高精度」
x2(high_accuracy)=.....
Relation_roots_coefficient(original)=.... // relation は「関係」、roots は「根≡解」
Relation_roots_coefficient(high_accuracy)=.... // coefficient は「係数」
substitution with x1(original)= ..... // substitution は「代入」
substitution with x2(original)= ..... // 下の6つの値は0に近いはず。
substitution with x1(high_accuracy)= ..... // そうでなければ間違っている。
substitution with x2(high_accuracy)= .....
```

注：変数名には ( ) や | は使えない。アルファベットと数字とアンダースコア「\_」 くらいが使用できる。変数名は、上記の表示の左辺がわかるようにすること。例えば、x1(original) の変数は、x1\_orig にするなど。  
(問題終)

例 3.3  $x$  が 0 に近い値のとき、

$$y_1 = 1 - \frac{1}{\sqrt{1+x}}$$

を計算する。

$$1 - \frac{1}{\sqrt{1+x}} = \frac{\sqrt{1+x} - 1}{\sqrt{1+x}}$$

であるが、 $x$  が小さいときは右辺、左辺ともに「1」と「1に近い値」の引き算が出てくるので、桁落ちを生じる。右辺をもう少し変形して、分子と分母に  $\sqrt{1+x} + 1$  をかけると、

$$\begin{aligned} \frac{\sqrt{1+x} - 1}{\sqrt{1+x}} &= \frac{(\sqrt{1+x} - 1)(\sqrt{1+x} + 1)}{\sqrt{1+x}(\sqrt{1+x} + 1)} \\ &= \frac{(\sqrt{1+x})^2 - 1^2}{\sqrt{1+x}(\sqrt{1+x} + 1)} \\ &= \frac{x}{\sqrt{1+x}(\sqrt{1+x} + 1)} \end{aligned} \tag{3.4.4}$$

となり、最後の式では  $x$  が小さいときの「1」と「1に近い値」の引き算はなくなっている。

すなわち

$$y_2 = \frac{x}{\sqrt{1+x}(\sqrt{1+x} + 1)}$$

を計算したほうが、 $y_1$  を計算するより桁落ちが少なく精度が高い。

例 3.4  $\theta$  が 0 に近いときに  $1 - \cos \theta$  を計算する.

$1 - \cos \theta$  は「1」 - 「1 に近い値」となっているが、倍角の公式を用いて変形する.

$$1 - \cos \theta = 2 \sin^2(\theta/2)$$

右辺であれば、桁落ちがなく精度の高い計算ができる.

例 3.5  $\Delta x$  が小さい値の時  $\sin(x + \Delta x) - \sin(x)$  を計算する.

三角関数の差の公式

$$\sin A - \sin B = 2 \cos \left( \frac{A+B}{2} \right) \sin \left( \frac{A-B}{2} \right)$$

を用いて変形する.

$$\sin(x + \Delta x) - \sin(x) = 2 \cos \left( x + \frac{\Delta x}{2} \right) \sin \left( \frac{\Delta x}{2} \right)$$

この式の右辺を用いて計算する.

例 3.6 小さな  $x$  について  $e^x - 1$  を計算する. この場合も桁落ちの可能性はある. これに対してテーラー展開などの級数展開を用いる方法がある.

テーラー展開

無限微分可能な関数  $f(x)$  は以下のように書ける.

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!} (x-a)^n$$

$f^{(n)}(a)$  は  $f(x)$  を  $n$  階微分したものに  $x = a$  を代入したものである.

$f(x) = e^x$  をテーラー展開する. 上のテーラー展開の式の右辺で  $a = 0$  とする.  $(e^x)' = e^x$  なので,  $f^{(n)}(x) = e^x$  であり,  $f^{(n)}(a) = e^0 = 1$  となる. また,  $(x-a)^n = x^n$  である.

これより,

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

となるので,

$$e^x - 1 = x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots \quad (3.4.5)$$

$$= x \left( 1 + \frac{x}{2} \left( 1 + \frac{x}{3} (1 + \dots) \right) \right) \quad (3.4.6)$$

となり, この式を用いて計算すると  $x$  が小さいときの桁落ちを防ぐことができる.

**追加問題 3.1** (p.97, p.98 参照) 例 3.3 について, C 言語でプログラムを作成して検証せよ.  $x$  が 0 に近いときに, 誤差がどれくらいになるかを調べよ. 式変形をする前の値  $y_1$  と変形した後の値  $y_2$  を表示して, 比較せよ.



計算の精度は、次のようにして調べよ。  $y_1$  または  $y_2$  は、正しく計算できれば

$$y^2 - 2y + \frac{x}{1+x} = 0$$

を満たす（代入すればすぐに確かめられる）。この式の左辺の  $y$  に、 $y_1$  または  $y_2$  を代入した値を計算し、どちらが 0 に近いかを見る。0 に近いほうが精度が高いはずである。

表示するのは次の 5 つ。  $x$ ,  $y_1$ ,  $y_2$ ,  $y_1$  の精度（上式の左辺の  $y$  に  $y_1$  を代入した値）,  $y_2$  の精度。

C 言語では、 $y^2$  については  $y*y$  として作成せよ。

$x$  が大きいときには、 $y_1$  と  $y_2$  の違いは少ないことを確認せよ。もしそうでなければ、入力した式が間違っている可能性が高い。

変数は誤差が分かりやすくするために `double` は用いず `float` を用いること。また、`printf` で数値を表示するときは、`%f` ではなく `%.20e` を用いること。これは有効桁 20 桁で指数形式 (`1e-5` など) で表示する指定である。

また、Linux (gcc) で作る場合には、以下のようにすること。

```
#include <stdio.h>
#include <math.h>
#include <fpu_control.h> // 計算途中の精度を変えるためのライブラリ

int main(){
    unsigned int flag = _FPU_SINGLE; // 計算途中を float の精度にする
    unsigned short cw;
    _FPU_GETCW(cw);
    cw = (cw & ~0x300) | flag;
    _FPU_SETCW(cw);

    ここに例 2.2 などの計算を行う式を記述。
}
```

これは式の計算途中も `float` の精度で扱うようにするための設定である。Linux (gcc) ではこのような設定がない場合には `double` に変換されて計算されるため、誤差がわかりにくい。

Windows では設定などが少し異なるので、Linux を使用すること。（学生舎で Linux を使う方法は、このテキストの後ろにある。）

`#include <math.h>` は数学の関数を使うためである。 $\sqrt{a}$  は C 言語では `sqrt(a)` とする。

さらに、Linux でコンパイルする際には、

```
cc ソースファイル名 -lm
もしくは
gcc ソースファイル名 -lm
```

のように、`-lm` オプションをつける必要があることに注意する。Windows (Visual C++ 等) でコンパイルするときにはオプションは不要。

$x$  が 0 に近いときを調べた後、 $x$  が 0 に近くない場合も調べよ（ソースを書き直してコンパイル・実行すればよい）。 $x$  が 0 に近くないときは桁落ちは発生しないはずである。桁落ちが発生しているようであれば、計算式が間違っている可能性が高い。(追加問題終)

桁落ちの追加問題に関する一般的注意：

追加問題で、桁落ちが生じない場合には、以下に注意する。以下の一部を用いても良い。十分に桁落ちしている場合には、以下はしなくても良い。

- 上記の `_FPU_SETCW` の設定を行い、計算を `float` にする。
- 2.0 などの記述は `double` になるので、`2.0f` として `float` にするか、`2` として `int` にする。
- `cos` や `sqrt` などの関数は返り値が `double` なので、返り値が `float` の `cosf` や `sqrtf` などを用いる。

**追加問題 3.2** (p.97, p.98 参照) 例 3.4, 例 3.5 について、追加問題 3.1 と同様に、C 言語でプログラムを作成して検証せよ。ただし、追加問題 3.1 のように簡単に誤差を調べることはできないので、変形前と変形後の数値を表示して、違いがあることを確認せよ。その際、入力 of  $\theta, x, \Delta x$  などの数値も表示せよ。

C 言語の変数は全角文字は使えないので、 $\theta, \Delta x$  に対しては `theta, delta_x` などとすること。

$\theta$  や  $\Delta x$  が大きいときには、桁落ちは生じないので、変形前と変形後の値が同じになることを確認せよ。ただし、 $\sin(\theta)$  などの引数  $\theta$  の単位はラジアンで、その範囲は（本質的には） $0 \leq \theta < 2\pi$  であることに注意せよ。また、 $\sin(\theta)$  など、 $\theta \equiv 2\pi$  のときは、 $\theta \equiv 0$  と同じになり、 $\theta$  が小さい場合を考えていることになるので注意せよ。ここで記号  $\equiv$  は、「近似（値が近い）」を意味する。 $\equiv$  は、 $\approx$  や  $\simeq$  と書かれることもある。

`float` で計算した 2 つの結果（式変形前、式変形後）と、`double` で計算した 2 つの結果（式変形前、式変形後）を比較することで、誤差を調べることができるが、もう一度 `_FPU_SETCW` をするなど少し煩雑なので、この問題ではやらなくてよい。興味があれば試してみるとよい。(追加問題終)

**追加問題 3.3** 例 3.6 について、式 (3.4.6) を計算するプログラムを作成し、桁落ちがある場合とない場合の結果を比較せよ。 $e^x$  は `exp(x)` で計算できる。変数には `float` を使い、追加問題 3.1 と同様にして、計算途中も `float` になるようにして計算せよ。

`math.h` には、 $e^x - 1$  を計算する関数 `expm1(x)` が用意されている。この関数を用いると、 $x$  が小さいときの  $e^x - 1$  の値を比較的正確に求めることができる。この値を含めて全部で 3 つの値を比較せよ。

それができたら、桁落ちが生じないような  $x$  の値に設定して、同様に 3 つの値を比較せよ。

(追加問題終)

**追加問題 3.4** 次の計算において、直接的な計算値と桁落ちを防止した計算法の値を比較せよ。変数は `float` を用いよ。また、追加問題 3.1 と同様に、計算途中も `float` の精度になるようにせよ。 $x$  の値は、桁落ちが生じるように設定すること。また、桁落ちが生じないような  $x$  では、式の変形前と変形後の値がほぼ変わらないことを確認せよ。

1.  $x$  が 0 に近い値の時,  $\frac{1 - \cos(x)}{\sin(x)}$
2.  $x$  が非常に大きい値のとき  $(x+1)^{1/3} - x^{1/3}$   
(ヒント:  $x^3 - y^3 = (x-y)(x^2 + xy + y^2)$  を用いる.)  $x^a$  は `pow(x,a)` で計算できる.
3.  $x$  が 4 に非常に近い値のとき,  $\frac{1/\sqrt{x} - 1/2}{x-4}$

(追加問題終)

### 3.5 積残し

大きさが非常に異なる数値の足し算をするとき, 小さい項が無視されることがある. 例えば, 有効桁 5 桁の計算では,

$$12345 + 0.12345 = 12345$$

となってしまう, 0.12345 の項が無視されてしまう. この現象を「積残し」または「情報落ち」と呼ぶ.

上記の例では積み残しは防ぎようがないが, 以下の例では積み残しを減らすことができる.

#### 例 3.7

$$\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6}$$

となる. この左辺を計算する.  $\sum_{n=1}^{\infty} \frac{1}{n^2}$  を数値計算でもとめることはできないので, 大きい  $N$  ( $N = 100000$

など) に対して  $\sum_{n=1}^N \frac{1}{n^2}$  を計算する.

$N$  を大きくすればするほど, 上記の和は  $\frac{\pi^2}{6}$  に近づくはずである.

しかし, この和を計算するとき

$$1 + \frac{1}{2^2} + \frac{1}{3^2} + \dots$$

というように計算すると, ある程度以上大きな  $N$  に対して  $\frac{1}{N^2}$  は非常に小さな値となり, この項は無視されることになり, それ以上  $N$  を大きくしても近似は良くならない.

これを防ぐには項の順序を変えて和を計算すればよい. 小さい項から足していくようにすれば積み残しは防げる. 具体的には

$$\frac{1}{N^2} + \frac{1}{(N-1)^2} + \frac{1}{(N-2)^2} + \dots + \frac{1}{2^2} + 1$$

というように, 小さい項から足していくようにする. これにより積残しを防ぐことができる.

**問題 3.3** (p.97, p.102 参照) 例 3.7 において, 和の順序を変えることにより精度がどのくらい変わるかを確かめよ.

$$1 + \frac{1}{2^2} + \frac{1}{3^2} + \dots + \frac{1}{N^2}$$

の計算と、

$$\frac{1}{N^2} + \frac{1}{(N-1)^2} + \frac{1}{(N-2)^2} + \cdots + \frac{1}{2^2} + 1$$

の計算を行い（両者とも左から順に足す）、 $\frac{\pi^2}{6}$  と比較せよ。

- 問題 3.1 のプログラムをコピーして用いることを推奨。よくわからなければ、p.102 のプログラムを参照および、入力して実行する。
- 変数は `double` は用いず、`float` を用いよ。ただし `for(k=...)` などの `k` は `int` にする（`float`, `double` は使わない）。`N` も `int` とする。もし上記の `k` などに `float` を用いると、丸め誤差や積み残しが発生して、`for` 文の終了条件が正しく動かない可能性がある。これらを正確に考慮すれば使用することも可能だが、「通常は `int` を使用する」と覚えておいたほうが良い。
- 表示するのは全部で 6 つ（下記参照）
- `N` の値の表示は、`printf("N=%d\n",N);` のように変数 `N` を用いること。  
`printf("N=200000\n");` のように直接 `N` の値を書き込んではいけない（正しいことをどうやって確かめる？）。
- $\pi$  の値は、`M_PI` を用いよ。`math.h` を `include` していれば、`M_PI` が使える。（Linux 以外では `M_PI` が使えない場合もある。）
- はじめは `N = 2000000` として計算せよ（0 が 6 個）。エラーが出るときに、もし `N` を小さくしたらエラーが出ない場合には、`n` の値は整数（`int`）の範囲だが、`n2` が整数（`int`）の範囲にならないことが原因の可能性が高い。この場合は、`n*n` のところを `(float)n*n` とすると、浮動小数点数（`float`）`n` と整数 `n` の積となり、整数 `n` は浮動小数点数（`float`）に変換されてから積を計算するのでエラーを回避できる。`(float)n*(float)n` としても良い。
- $\frac{1}{N^2} + \frac{1}{(N-1)^2} + \frac{1}{(N-2)^2} + \cdots + \frac{1}{2^2} + 1$  の計算では、小数点以下 6 ケタ程度正しく求められるはずである。もし正しく求められなければ、 $\frac{1}{n^2}$  のプログラム中での表記法を考えよ。
- さらに、`N` を変化させて誤差がどのように変わるか調べよ。特に、 $100 \leq N \leq 500$  の場合を行うこと。`N` が小さいときには積み残しはどうか？
- この問題では、2 種類の誤差が発生することに注意。  
一つは「積み残し」で、ある値に（相対的に）小さな値を加えるときに、小さな値が無視されてしまうことで生じる。  
もう一つは本来は無限に足さなければならないものを、途中まで打ち切ってしまうことで生じるもの。これを「打ち切り誤差」という。
- `N` の値を変更する際、何か所も変更することがないように、1 か所のみ変更すれば良いようにプログラムを作成せよ。
- 値がおかしい場合には、第 11.3 節、第 11.4 節を読むこと。

わからなければ,

- 問題 3.1 のプログラムをコピー, または p.102 のプログラムを入力して実行 (問題 3.1 よりも, p.102 のプログラムを推奨する)
- double をすべて float に変更
- 各行で何をしているかを理解して, s= の式をこの問題の  $\sum$  の式に変更して値を表示する.
- $\frac{\pi^2}{6}$  を表示して, 値が近いことを確認.
- 次にもう一つ for 文を作り, p.102 の下を見て, 逆順の和を作成
- 表示を下記のように行う.

出力は以下のようにする.

出力

```
N=....  
sum(original)=....  
sum(high_accuracy)=....  
true_value=.... // true value の意味は「真値, 正しい値」  
error(original)=...  
error(high_accuracy)=....
```

error(...) は sum(...)-true\_value で求める.

(問題終)

### 例 3.8

$$\sum_{i=1}^n x_i$$

の計算で,  $x_i$  の大きさがあまり変わらない場合には, 2 分木法と呼ばれる方法がある.

これは, まず 2 項ずつの和を作り, その結果の 2 項ずつを足す, ということを繰り返す方法である.

$$\begin{aligned} & x_1 + x_2 + x_3 + x_4 + \dots \\ &= (x_1 + x_2) + (x_3 + x_4) + \dots (\cdot \text{のついた} + \text{を計算する}) \\ &= ((x_1 + x_2) + (x_3 + x_4)) + \dots (\cdot \text{のついた} + \text{を計算する}) \end{aligned}$$

例 3.9 次のようなアルゴリズム (C 言語に似た形で記述) で積み残しを減らすことができる.

```
// R, S の初期値は 0  
for(i=0; i<=n; i++){  
    xi=      //データ xi を読み込むまたは計算する  
    R = R+xi; // 積み残し R と加算項 xi の和: 積み残しの取り込み  
    T = S;    // S: 部分和 (積み残しあり), T: S の一時記憶  
    S = S+R; // 部分和
```

```

    T = S-T; // 新たに加算した量
    R = R-T; // 積み残し量
}

```

**レポート 3.1** レポートは提出しなくてもよい（追加問題と同じように提出すれば加点される）。

例 3.9 について具体的な和を考えて  $x_i$  の式を与えて、このアルゴリズムを使った場合と使わない場合についての誤差の違いについて調べよ。レポートなので、目的、方法、結果、考察について書くこと（pdf を提出）。

### 3.6 計算回数の削減

一般的には、計算回数を少なくしたほうが誤差は少なくなる。

**例 3.10** 以下の  $n$  次多項式の計算を行う。

$$f(x) = c_n x^n + c_{n-1} x^{n-1} + \cdots + c_0$$

この計算法の代表的なものには以下の 2 つがある。

1. まず  $x^2, x^3, \dots, x^n$  を計算する（計算回数:  $n-1$  回）。次に  $c_n x^n, c_{n-1} x^{n-1}, \dots, c_1 x$  を計算し（計算回数:  $n$  回）、最後にこれらの和を求める。この計算法の計算回数は、乗算  $2n-1$  回、加算  $n$  回である。
2. 式を変形し次の形にする。

$$f(x) = (\dots((c_n x + c_{n-1})x + c_{n-2})x + \dots)x + c_0$$

この式の内側のカッコから順次計算する。この計算の計算回数は乗算  $n$  回、加算  $n$  回である。この方法は **ホーナー (Horner) 法** と呼ばれ、最小の計算回数で計算する手法である。

**追加問題 3.5** 上記の多項式の計算について、ホーナー法を使う場合と使わない場合の精度について比較せよ。変数は `float` を用いよ。また、追加問題 3.1 と同様に、計算途中も `float` の精度になるようにせよ。係数  $c_0, \dots, c_n$  は配列に入れること。

$c_n$  は定数ではないようにすること。  $c_n$  を簡単な  $n$  の式にして、公式等を用いて  $f(x)$  の式（ループを用いない計算）を求めて、その値と比較することにより、精度を比較せよ（3 つの値を比較する）。

2 重の `for` 文は用いない（関数の中に `for` 文があり、その関数を `for` 文の中で用いる場合も 2 重の `for` 文となるので、これも用いない）。

関数 `pow` は用いずに作ること（そうしないと計算回数の削減にならないことがある）。ただし、公式の計算および  $c_n$  の計算では `pow` を用いて良い。公式の例としては以下のようなものがある。他の公式を用いても良い。桁落ちなどに注意すること。

$$\sum_{k=0}^n (a + kd)r^k = \frac{a - (a + nd)r^{n+1}}{1 - r} + \frac{dr(1 - r^n)}{(1 - r)^2}, \quad \sum_{k=0}^n \frac{n!}{k!(n-k)!} a^{n-k} b^k = (a + b)^n$$

(追加問題終)

### 3.7 基本的な計算

ここではこの授業でのプログラムを作成するにあたり重要と思われることを取り扱う。

**問題 3.4**  $r \neq 1$  に対して、数列  $a_k = r^k$  ( $k = 0, 1, \dots, n-1$ ) の相加平均と相乗平均を求めるプログラムを作成せよ。  $n$  の値はプログラム中に記述して、1 か所のみ変更すれば良いようにすること。また、 $1 < r < 2$ ,  $10 \leq n \leq 40$  に設定すること。  $n$  は偶数と奇数の場合を両方行う。変数は `double` を用いる。ただし  $k$  と  $n$  は `int` とする。

$r^k$  は `pow(r,k)` で計算できる。

$a_k$  ( $k = 0, \dots, n-1$ ) の相加平均は、 $\frac{1}{n} \sum_{k=0}^{n-1} a_k$  である。これを for 文により計算する。

$a_k$  ( $k = 0, \dots, n-1$ ) の相乗平均は、 $\sqrt[n]{\prod_{k=0}^{n-1} a_k} = \left( \prod_{k=0}^{n-1} a_k \right)^{\frac{1}{n}}$  である。これを for 文により計算する。

$\prod_{k=0}^{n-1} a_k$  は、 $a_0 \cdot a_1 \cdots a_{n-1}$  のことである。

$a_k = r^k$  ( $r \neq 1$ ) のとき、 $\sum_{k=0}^{n-1} r^k = \frac{r^n - 1}{r - 1}$  なので、相加平均は  $\frac{r^n - 1}{n(r - 1)}$  となる。

$\prod_{k=0}^{n-1} r^k = r^{\sum_{k=0}^{n-1} k} = r^{\frac{(n-1)n}{2}}$  なので、相乗平均は  $r^{\frac{n-1}{2}}$  となる。

これらの公式を用いた結果も表示すること。

表示は、 $r$  の値、 $n$  の値、for 文による相加平均の値、公式による相加平均の値、for 文による相乗平均の値、公式による相乗平均の値の 6 つとする。

出力

```
r=....
n=....
arithmetic mean(for)=.... // arithmetic(算術) mean(平均) = 相加平均
arithmetic mean(formula)=.... // (for)=for 文, (formula)=公式を使用
geometric mean(for)=.... // geometric(幾何) mean(平均) = 相乗平均
geometric mean(formula)=...
```

わからなければ、まずは  $\sum_{k=0}^{n-1}$  を作成する。問題 3.3 のプログラムをコピーして、`s=...` の式と  $k$  の範囲を変更する。

$\prod_{k=0}^{n-1}$  については、 $\sum_{k=0}^{n-1}$  のプログラムをどのように変更すれば良いかを考える。変更は複数ある。

(問題終)

**問題 3.5**  $f(x) = \frac{x}{x^2 + 1} + 1$  ( $-2 \leq x \leq 2$ ) の最大値と最小値および最大となる  $x$ と最小になる  $x$ を求めよ。ただし、 $x$  を実数とした場合の最大・最小ではなく、 $x$  の範囲を  $n$  個に分割して、それらの  $x$  に対して最大・最小を求める。ここでは、 $n = 40$ 、 $x$  の刻み幅  $h = \frac{x \text{ の最大} - \text{最小}}{n} = \frac{2 - (-2)}{n} = \frac{4}{n}$ 、 $x_0 = -2$  に対し、 $x_k = x_0 + h \cdot k$  ( $k = 0, 1, \dots, n$ ) での  $f(x_k)$  を計算して、これらの最大値と最小値を求めよ（配列は用いずに作ること）。変数は `double`、ただし  $k$  と  $n$  は `int` とする。

最小値 (`fmin`) と最大値 (`fmax`) の初期値に関しては、

1. `#include <math.h>` として、無限大を示す `INFINITY` および負の無限大 `-INFINITY` を用いる方法、
2. `#include <float.h>` として `double` の最大値を示す `DBL_MAX` および `-DBL_MAX` を用いる方法、
3.  $f(x_0)$  などの関数の値を初期値に用いて、さらに、最大・最小となる  $x_k$  (`xmax`, `xmin`) の初期値を  $x_0$  とする方法 (`xmax`, `xmin` の初期値の設定は、 $f(x_0)$  が最大や最小の場合、`xmax` や `xmin` が一度も更新されないことに対処)、

の3つのいずれでも良い（環境によっては使えないこともあるので、その場合には別の方法を用いること）。

表示は、最大値 (`fmax`)、最大値となる  $x_k$  (`xmax`)、最小値 (`fmin`)、最小値となる  $x_k$  (`xmin`) の4つとする（カッコ内はそれぞれの変数名）。

$\lim_{x \rightarrow \infty} f(x) = 1$ ,  $\lim_{x \rightarrow -\infty} f(x) = 1$  であり、 $f'(x) = \frac{-(x-1)(x+1)}{(x^2+1)^2}$  なので（計算は省略）、 $x = \pm 1$  で最小または最大となるはずである。

ヒント：最大値の場合、仮の最大値（今まで調べた中での最大値）`fmax` を考える。まず適切に `fmax` の初期値を定め、上記の  $k$  の範囲の `for` 文を作成し、その `for` 文の中で  $x_k$  と  $f(x_k)$  を計算して、`if` 文を用いて  $f(x_k)$  と仮の最大値 `fmax` を比較する。仮の最大値より  $f(x_k)$  が大きければ、`fmax` と `xmax` を更新する。

出力

```
fmax=....
xmax=....
fmin=....
xmin=....
```

(問題終)



## 4 数値積分

### 4.1 概説

関数  $f(x)$  の区間  $[a, b]$  での定積分を数値的に計算する.

$$\int_a^b f(x)dx$$

$x_0 = a < x_1 < x_2 < \cdots < x_n = b$  として,

$$\int_a^b f(x)dx \doteq \sum_{i=0}^n a_i f(x_i)$$

として積分を近似して計算する. 右辺の  $a_i$  を重みという. 重みの決め方にはいくつかの方法がある.

### 4.2 中点公式

閉区間  $[a, b]$  間の関数  $f(x)$  を  $[a, b]$  の中点  $x_0 = \frac{a+b}{2}$  で近似する.

$$f(x) \doteq f(x_0)$$

これを用いて,

$$\int_a^b f(x)dx \doteq \int_a^b f(x_0)dx = hf(x_0), \quad h = b - a \quad (4.2.1)$$

として計算する. 上式の最後の等号は,  $f(x_0)$  が定数であることから成立する.

### 4.3 台形公式

積分区間  $[a, b]$  間の関数  $f(x)$  を両端点  $(a, f(a))$ ,  $(b, f(b))$  を結ぶ直線で近似する:

$$f(x) \doteq \frac{f(b) - f(a)}{b - a}(x - a) + f(a) = \frac{f(b) - f(a)}{b - a}x + \frac{bf(a) - af(b)}{b - a}$$

これより,

$$\begin{aligned} \int_a^b f(x)dx &\doteq \int_a^b \left( \frac{f(b) - f(a)}{b - a}x + \frac{bf(a) - af(b)}{b - a} \right) dx \\ &= \frac{f(b) - f(a)}{b - a} \int_a^b x dx + \frac{bf(a) - af(b)}{b - a} \int_a^b dx \\ &= \frac{f(b) - f(a)}{b - a} \left[ \frac{1}{2}x^2 \right]_a^b + \frac{bf(a) - af(b)}{b - a} [x]_a^b \\ &= \frac{f(b) - f(a)}{b - a} \left( \frac{1}{2}(b^2 - a^2) \right) + \frac{bf(a) - af(b)}{b - a}(b - a) \\ &= \frac{b - a}{2} (f(b) + f(a)) \end{aligned} \quad (4.3.1)$$

となる. 式 (4.3.1) を台形公式という. もう少し簡単に, 台形の上底  $= f(a)$ , 下底  $= f(b)$ , 高さ  $= b - a$  として, 台形の面積の公式「(上底 + 下底)  $\times$  高さ  $\div 2$ 」に代入しても同じ式が得られる.

#### 4.4 第1 シンプソン公式

閉区間  $[a, b]$  間の関数  $f(x)$  を 2 次式で近似する.

$$f(x) \doteq c_2 x^2 + c_1 x + c_0$$

この式の係数  $c_0, c_1, c_2$  を決めるためには 3 つの点が必要であるが, ここでは両端点と中点  $(a, f(a))$ ,  $(m, f(m))$ ,  $(b, f(b))$  を用いる. ただし  $m = \frac{a+b}{2}$  とする.

$c_0, c_1, c_2$  は以下の連立 1 次方程式で求められる.

$$f(a) = c_2 a^2 + c_1 a + c_0$$

$$f(m) = c_2 m^2 + c_1 m + c_0$$

$$f(b) = c_2 b^2 + c_1 b + c_0$$

この連立 1 次方程式に対しクラメールの公式を用いると,

$$c_0 = \frac{\begin{vmatrix} f(a) & a & a^2 \\ f(m) & m & m^2 \\ f(b) & b & b^2 \end{vmatrix}}{|D|} = \frac{a^2 b f(m) + a m^2 f(b) + b^2 m f(a) - a^2 m f(b) - a b^2 f(m) - b m^2 f(a)}{|D|} \quad (4.4.1)$$

$$c_1 = \frac{\begin{vmatrix} 1 & f(a) & a^2 \\ 1 & f(m) & m^2 \\ 1 & f(b) & b^2 \end{vmatrix}}{|D|} = \frac{a^2 f(b) + m^2 f(a) + b^2 f(m) - a^2 f(m) - m^2 f(b) - b^2 f(a)}{|D|} \quad (4.4.2)$$

$$c_2 = \frac{\begin{vmatrix} 1 & a & f(a) \\ 1 & m & f(m) \\ 1 & b & f(b) \end{vmatrix}}{|D|} = \frac{a f(m) + m f(b) + b f(a) - a f(b) - m f(a) - b f(m)}{|D|} \quad (4.4.3)$$

$$|D| = \begin{vmatrix} 1 & a & a^2 \\ 1 & m & m^2 \\ 1 & b & b^2 \end{vmatrix} = a^2 b + a m^2 + m b^2 - a^2 m - m^2 b - a b^2$$

$$m = \frac{a+b}{2}$$

となり,  $c_0, c_1, c_2$  を求めることができる.

これらの  $c_0, c_1, c_2$  を用いると, 式 (4.4.1), (4.4.2), (4.4.3) は複雑だが, 結局簡単な式が得られ,

$$\begin{aligned} \int_a^b f(x) dx &\doteq \int_a^b (c_2 x^2 + c_1 x + c_0) dx \\ &= \frac{c_2}{3} (b^3 - a^3) + \frac{c_1}{2} (b^2 - a^2) + c_0 (b - a) \\ &= \dots \\ &= \frac{b-a}{6} (f(a) + 4f(m) + f(b)) \\ &= \frac{h}{3} (f(a) + 4f(a+h) + f(b)) \quad \text{ただし } h = \frac{b-a}{2} \end{aligned} \quad (4.4.4)$$

となる. この式 (4.4.4) は第 1 シンプソン公式または  $\frac{1}{3}$ -シンプソン公式と呼ばれる.

## 4.5 第2 シンプソン公式

積分区間  $[a, b]$  を3等分して、端点を加えた4点を用いて  $f(x)$  を3次関数で近似する.

$$f(x) \doteq c_3 x^3 + c_2 x^2 + c_1 x + c_0$$

第1 シンプソン公式の場合と同様にして  $c_0, c_1, c_2, c_3$  を求め、上式の右辺を  $[a, b]$  間で積分して整理すると以下の式が得られる.

$$\int_a^b f(x) dx \doteq \frac{3h}{8} (f(a) + 3f(a+h) + 3f(a+2h) + f(b)) \quad (4.5.1)$$
$$h = \frac{b-a}{3}$$

式 (4.5.1) を第2 シンプソン公式または  $\frac{3}{8}$ -シンプソン公式という.

**問題 4.1** (p.97, p.98, p.104 参照) 次の積分を台形公式 (式 (4.3.1)), 第1 シンプソン公式 (式 (4.4.4)), 第2 シンプソン公式 (式 (4.5.1)) を用いて求めよ. 正しい答え (問題の式の右辺の値) もプログラムで表示せよ. 数値は変数に代入して, その変数を `printf` で表示するようにすること.

$$\int_0^1 \frac{1}{x^2+1} dx = \frac{\pi}{4} \quad (4.5.2)$$

この問題以降 (次の合成積分についても) `float` は用いずに `double` を用いることにする.

$f(x)$  は C 言語の関数を用いて作成すること.

わからなければ, まず p.104 のプログラムを実行し, その関数を  $f(x) = \frac{1}{x^2+1}$  に変更せよ. 関数名は `f` にした方がわかりやすい. 次に, 変数 `a, b, h` などを用意して,  $f(x)$  を用いて台形公式, 第1 シンプソン公式, 第2 シンプソン公式を記述せよ.  $\sum$  の計算は, p.102 や, 問題 3.4 を参照する.

出力

```
trapezoidal=....
first Simpson=....
second Simpson=....
true value=....
```

値は0.7の桁までは正しいはずである. 値がおかしい場合には, 第11.3節 (p.120), 第11.4節 (p.121) を読むこと.

(問題終)

## 4.6 合成積分公式

上述した第 1, 2 シンプソン公式は、積分区間  $[a, b]$  を 2 等分または 3 等分した場合の積分を求める方法である。一般により多くの点で等分した場合には、これらの公式を組み合わせて使用する方法が考えられる。このように基本公式を繰り返し使用して作られる積分公式を合成積分公式という。

後述するように高次の多項式近似を用いるニュートン・コーツの公式もあるが、それよりもシンプソン公式を繰り返し使ったほうが簡単であり、しかも精度も良い場合が多い。

### 4.6.1 合成台形公式

積分区間  $[a, b]$  が  $n$  個の等分点で与えられている場合、 $n$  個の分割領域にそれぞれ式 (4.3.1) の台形公式を用いる。  $h = \frac{b-a}{n}$  とし、  $x_0 = a, x_1 = a + h, \dots, x_i = a + ih, \dots, x_n = a + nh = b$  とする。

$$\begin{aligned}\int_a^b f(x)dx &= \sum_{i=0}^{n-1} \frac{h}{2} (f(x_i) + f(x_{i+1})) \\ &= \frac{h}{2} \left( \sum_{i=0}^{n-1} f(x_i) + \sum_{i=0}^{n-1} f(x_{i+1}) \right) = \frac{h}{2} \left( \sum_{i=0}^{n-1} f(x_i) + \sum_{i=1}^n f(x_i) \right) \\ &= \frac{h}{2} \left( f(a) + 2 \sum_{i=1}^{n-1} f(x_i) + f(b) \right) \quad (4.6.1)\end{aligned}$$

ただし  $x_i = a + ih$

となる。式 (4.6.1) を合成台形公式という。

### 4.6.2 合成第 1 シンプソン公式

式 (4.4.4) の第 1 シンプソン公式は、積分区間  $[a, b]$  を 2 等分して、その間を 2 次式で近似して得られるものである。一般に積分区間を  $2n$  等分し、  $h = \frac{b-a}{2n}$  とし、  $[a, a + 2h], [a + 2h, a + 4h], \dots, [a + 2ih, a + 2(i+1)h], \dots, [a + 2(n-1)h, b]$  の  $n$  個の小区間に対して第 1 シンプソン公式を適用する。

$x_i = a + ih$ , ( $i = 0, 1, \dots, 2n$ ) とすると,  $x_{2n} = b$  であり,

$$\begin{aligned}
\int_a^b f(x)dx &\doteq \sum_{i=0}^{n-1} \frac{h}{3} \left( f(x_{2i}) + 4f(x_{2i+1}) + f(x_{2i+2}) \right) \\
&= \frac{h}{3} \left( \sum_{i=0}^{n-1} f(x_{2i}) + 4 \sum_{i=0}^{n-1} f(x_{2i+1}) + \sum_{i=0}^{n-1} f(x_{2i+2}) \right) \\
&= \frac{h}{3} \left( \sum_{i=0}^{n-1} f(x_{2i}) + 4 \sum_{i=0}^{n-1} f(x_{2i+1}) + \sum_{i=1}^n f(x_{2i}) \right) \\
&= \frac{h}{3} \left( f(x_0) + \sum_{i=1}^{n-1} f(x_{2i}) + 4 \sum_{i=0}^{n-1} f(x_{2i+1}) + \sum_{i=1}^{n-1} f(x_{2i}) + f(x_{2n}) \right) \\
&= \frac{h}{3} \left( f(a) + 2 \sum_{i=1}^{n-1} f(x_{2i}) + 4 \sum_{i=0}^{n-1} f(x_{2i+1}) + f(b) \right) \tag{4.6.2}
\end{aligned}$$

ただし  $x_i = a + ih$ , ( $i = 0, 1, \dots, 2n$ )

$$h = \frac{b-a}{2n}$$

となる. 式 (4.6.2) で  $n = 1$  とすると,  $\sum_{i=1}^0 f(x_{2i}) = 0$ ,  $\sum_{i=0}^0 f(x_{2i+1}) = f(x_1)$  となるので, 式 (4.4.4) が得られる.

#### 4.6.3 合成第 2 シンプソン公式

積分区間  $[a, b]$  を  $3n$  等分し,  $h = \frac{b-a}{3n}$  として,  $[a, a+3h], [a+3h, a+6h], \dots, [a+3ih, a+3(i+1)h], \dots, [a+3(n-1)h, b]$  の  $n$  個の小区間に対して, 第 2 シンプソン公式を適用する.

$x_i = a + ih$ , ( $i = 0, 1, \dots, 3n$ ) とすると,  $x_{3n} = b$  であり,

$$\begin{aligned}
\int_a^b f(x)dx &\doteq \sum_{i=0}^{n-1} \frac{3h}{8} \left( f(x_{3i}) + 3f(x_{3i+1}) + 3f(x_{3i+2}) + f(x_{3i+3}) \right) \\
&= \frac{3h}{8} \left( \sum_{i=0}^{n-1} f(x_{3i}) + 3 \sum_{i=0}^{n-1} f(x_{3i+1}) + 3 \sum_{i=0}^{n-1} f(x_{3i+2}) + \sum_{i=0}^{n-1} f(x_{3i+3}) \right) \\
&= \frac{3h}{8} \left( \sum_{i=0}^{n-1} f(x_{3i}) + 3 \sum_{i=0}^{n-1} f(x_{3i+1}) + 3 \sum_{i=0}^{n-1} f(x_{3i+2}) + \sum_{i=1}^n f(x_{3i}) \right) \\
&= \frac{3h}{8} \left( f(x_0) + \sum_{i=1}^{n-1} f(x_{3i}) + 3 \sum_{i=0}^{n-1} f(x_{3i+1}) + 3 \sum_{i=0}^{n-1} f(x_{3i+2}) + \sum_{i=1}^{n-1} f(x_{3i}) + f(x_{3n}) \right) \\
&= \frac{3h}{8} \left( f(a) + 2 \sum_{i=1}^{n-1} f(x_{3i}) + 3 \sum_{i=0}^{n-1} f(x_{3i+1}) + 3 \sum_{i=0}^{n-1} f(x_{3i+2}) + f(b) \right) \tag{4.6.3}
\end{aligned}$$

ただし  $x_i = a + ih$ , ( $i = 0, 1, \dots, 3n$ )

$$h = \frac{b-a}{3n}$$

となる。式 (4.6.3) で  $n = 1$  とすると、 $\sum_{i=1}^0 f(x_{3i}) = 0$ ,  $\sum_{i=0}^0 f(x_{3i+1}) = f(x_1)$ ,  $\sum_{i=0}^0 f(x_{3i+2}) = f(x_2)$  となるので、式 (4.5.1) が得られる。

**問題 4.2** (p.97, p.98, p.102, p.104 参照) 問題 4.1 の積分について、合成台形公式 (式 (4.6.1)), 合成第 1 シンプソン公式 (式 (4.6.2)), 合成第 2 シンプソン公式 (式 (4.6.3)) で計算するプログラムを作成し、 $n$  を変化させて、正しい答えと比較せよ。問題 4.1 と同様に関数を使用すること。

わからなければ、p.102 と p.103 を参考にして、 $\sum$  の個数だけ for ループを作り、ひとつの for ループでひとつの  $\sum$  を計算し、それを用いて合成積分公式を計算せよ。

- 表示するのは最低でも次の 5 つの値である。 $n$  の値、合成台形公式の値、合成第 1 シンプソンの値、合成第 2 シンプソンの値、正しい値 (問題の式の右辺の値)。
- $n$  の値は、`printf("n=%d\n",n);` などのように変数名を利用すること。`printf("n=500\n");` などのように値を直接書き込んではいけない (表示とプログラムの値が一致していることをどうやって確かめる?)
- 配列は使わずに作ること。(配列はもう少し先で使う。配列を使うと、非常に大きな  $n$  の扱いが難しくなる。)
- $a, b$  の値を変更した場合にも簡単に対応できるようにすること。例えば、 $a = 0$  を代入することで式を簡単にしてからプログラムしてはいけない (もしそうすると、 $a$  を変更することが大変になる)。
- $n$  を変化させることについて、最低でも次の 2 つの場合を行うこと。 $n$  が大きいとき ( $n \geq 300$ ) と  $n$  が小さいとき ( $n = 2$ )。
- $n$  が大きいほうが精度は良いはずである。そうでなければプログラムが間違っている。
- 合成積分公式 ( $n = 2$  の場合) と合成でない公式 (問題 4.1 の式) を比較すると、合成積分公式は、問題 4.1 より精度は良いはずである。精度が悪くなっていたら、プログラムが間違っている。
- プログラムが間違っている場合、 $\sum$  の範囲に注意する。また、p.102 と p.103 の「初期値」の設定がなぜこのようになっているかを理解すること。
- 値がおかしい場合には、第 11.3 節、第 11.4 節を読むこと。

出力

```
n=...
trapezoidal=....
first Simpson=....
second Simpson=....
true value=....
```

(問題終)

**追加問題 4.1** 追加資料の「関数を関数の引数にする」を読み、それを用いて、以下の  $I$  を求める。ただし、合成第 2 シンプソン公式を用いる。

$$I = \int_0^1 \frac{1}{x^2 + 1} dx + \int_2^4 \frac{\sqrt{(x-2)(4-x)}}{2} dx$$

上の式の第 1 項の積分を  $I_1$ 、第 2 項の積分を  $I_2$  として、 $I_1, I_2$  を求めて、 $I = I_1 + I_2$  を求めよ。

$I_1 = \frac{\pi}{4}$ ,  $I_2 = \frac{\pi}{4}$  となるはずなので、 $I = \frac{\pi}{2}$  となるはずである。

$n_1$  を  $I_1$  の分割数、 $n_2$  を  $I_2$  の分割数とする。  $n_1 \neq n_2$  とできるようにプログラムを作成し、 $n_1, n_2$  を変えて、 $I_1, I_2, I$  の精度を確かめる。  $I_1, I_2, I$  が 5 桁以上正しくなるようにするには  $n_1, n_2$  はどれくらいにすれば良いだろうか。なるべく小さい数字で答えよ (100 を単位として)。(追加問題終)

## 4.7 多重積分

多重積分についても、これまでの公式を用いることが可能である。

次の 2 重積分について考える。

$$A = \int_a^b \left( \int_c^d f(x, y) dx \right) dy \quad (4.7.1)$$

ただし  $a, b, c, d$  は定数とする。

$x$  の区間  $[c, d]$  を  $n$  等分、 $y$  の区間  $[a, b]$  を  $m$  等分し、それぞれの等分点を  $x_i, y_j$  とする。まず各  $y_j$  に対し、式 (4.7.1) の内側の  $x$  に関する積分をシンプソン公式などを用いて求める。

$$F(y_j) = \int_c^d f(x, y_j) dx = h_x \sum_{i=0}^n a_i f(x_i, y_j)$$

ただし、 $a_i$  は  $x$  に関する積分のシンプソン公式などの係数、 $h_x = \frac{d-c}{n}$  である。

次に  $F(y_j)$  に対して式 (4.7.1) の外側の  $y$  に関する積分をシンプソン公式などで求める。

$$A \doteq h_y \sum_{j=0}^m b_j F(y_j)$$

ここで  $b_j$  は  $y$  に関する積分のシンプソン公式などの係数、 $h_y = \frac{b-a}{m}$  である。

上記では 2 重積分について扱ったが、一般の多重積分についても同様の手法が適用できる。

**例 4.1** 半径 1 の球の体積を重積分で計算する。

1. 求める積分は

$$I = 8 \int_0^1 \left( \int_0^{\sqrt{1-z^2}} \sqrt{1-z^2-x^2} dx \right) dz$$

である。変数変換することも可能であるが、ここでは上式のままで積分することにする。

2. まずは内側の括弧の中を計算し、以下のように  $I_x(z)$  を定義する.

$$I_x(z) = \int_0^{\sqrt{1-z^2}} \sqrt{1-z^2-x^2} dx$$

$z$  を決めれば,  $I_x(z)$  はシンプソン公式などで計算できる.

3. このとき求める積分は、次式となる.

$$I = 8 \int_0^1 I_x(z) dz$$

**レポート 4.1** このレポートは提出しなくても良い (追加問題などと同じ). レポートなので, 目的, 方法, 結果, 考察について書くこと (pdf にして提出).

例 4.1 の計算を行うプログラムを, 次の手順で作成せよ.

1.  $x$  と  $z$  を与えた時に  $\sqrt{1-z^2-x^2}$  を返す関数を作成せよ. 関数名は `f` とし, 引数は `x` と `z` とする.  $x = \sqrt{1-z^2}$  のとき  $\sqrt{1-z^2-x^2} = 0$  となるはずだが, 計算誤差により,  $1-z^2-x^2$  が小さい負の値になり  $\sqrt{1-z^2-x^2}$  が計算できなくなることがある. そのようなときは返り値として 0 を返すようにせよ.

```
double f(double x, double z){
    ...
}
```

2.  $z$  を与えた時に

$$I_x(z) = \int_0^{\sqrt{1-z^2}} \sqrt{1-z^2-x^2} dx$$

を計算する関数を, 上記の関数 `f` を用いて作成せよ. 合成第 1 シンプソン公式を用いること. 引数は, `z` と `nx` とする. `nx` は積分の際の  $x$  の分割数である. 関数名は `intx` などとせよ. 合成第 1 シンプソン公式に関しては, 追加資料の関数を引数にする部分を用いると, 次の項目が簡単になる (用いなくても良い).

```
double intx(double z, int nx){
    ... 上式の積分の計算
}
```

3. 上記の関数 `intx(z,nx)` を用いて,

$$\int_0^1 \text{intx}(z, nx) dz = \int_0^1 I_x(z) dz$$

を計算するプログラムを作成せよ (関数にしなくて良い). 合成第 1 シンプソン公式を用いること.  $z$  の分割数は `nz` とする. この値を 8 倍すれば目的の積分値  $I$  になる. 計算した積分値  $I$  と正しい値 ( $= \frac{4}{3}\pi$ ) を比較せよ.  $\pi$  は `M_PI` を用いよ.

4. `nx`, `nz` の値を変化させて, 正しい値との違いがどうなるかを調べよ. `nx` と `nz` が等しい場合や異なる場合など, いろいろと行うこと.



## 5 数列

ここでは数列のプログラムについて扱う。数値計算としての内容は特にないが、後の章の基本となるプログラムである。

数列の変数名は、以下の例を参考にする。

- 「\_」はマイナス、何もなければプラスの意味
  - $x_i = \text{xi}$ ,  $x_{i-1} = \text{xi\_1}$ ,  $x_{i-2} = \text{xi\_2}$ ,  $x_{i+1} = \text{xi1}$ ,  $x_{i+2} = \text{xi2}$
  - $x_k = \text{xk}$ ,  $x_{k-1} = \text{xk\_1}$ ,  $x_{k-2} = \text{xk\_2}$ ,  $x_{k+1} = \text{xk1}$ ,  $x_{k+2} = \text{xk2}$ , など
- m はマイナス, p はプラス, 「\_」は添え字 (インデックス) の意味
  - $x_i = \text{x\_i}$ ,  $x_{i-1} = \text{x\_im1}$ ,  $x_{i-2} = \text{x\_im2}$ ,  $x_{i+1} = \text{x\_ip1}$ ,  $x_{i+2} = \text{x\_ip2}$
  - $x_k = \text{x\_k}$ ,  $x_{k-1} = \text{x\_km1}$ ,  $x_{k-2} = \text{x\_km2}$ ,  $x_{k+1} = \text{x\_kp1}$ ,  $x_{k+2} = \text{x\_kp2}$ , など

### 5.1 数列のプログラム

**問題 5.1** 次のプログラムは、等差数列を計算するものである。  $a = 1$ ,  $d = 0.5$ ,  $n = 5$  に対して、等差数列の一般項の式

$$a_i = a + id \quad (i = 0, \dots, n-1)$$

を求める。漸化式で表すと

$$a_0 = a \text{ (初期値)}$$

$$a_i = a_{i-1} + d \quad (i = 1, \dots, n-1)$$

となる。以下のプログラムは、

- 一般項の式 (配列を用いない)
- 一般項の式 (配列を用いる)
- 漸化式 (配列を用いる)
- 漸化式 (配列を用いない)

の 4 つの計算を行うプログラムである (理解しやすいと思われる順番に並べている)。

```
#include <stdio.h>
#define N 5 // n = 5 プログラム中の N を 5 に書き換えてからコンパイルする
int main(){
    double A[N]; // 配列  $a_0, \dots, a_{n-1}$  を定義 (一般項の式用)
    double Ar[N]; // 漸化式用の配列を定義 (r は recurrence relation (漸化式) の意味)
```

```

int i;
double a=1.0, d=0.5;
double ai, ai_1; // ai= $a_i$ , ai_1= $a_{i-1}$ 
// 一般項の式（配列を用いない）
printf("general terms without array\n");
for(i=0;i<N;i++){
    ai=a+i*d; // 一般項の式  $a_i = a + id$ 
    printf("a%d=%f\n",i,ai); //  $a_i$  の表示
}
// 一般項の式（配列を用いる）
printf("-----\n");
printf("general terms with array\n");
for(i=0;i<N;i++){
    A[i]=a+i*d; // 一般項の式  $a_i = a + id$ 
    printf("A[%d]=%f\n",i,A[i]); //  $a_i$  の表示
}
// 漸化式（配列を用いる）
printf("-----\n");
printf("recurrence relation with array\n");
Ar[0]=a; // 初期値  $a_0 = a$ 
printf("Ar[%d]=%f\n",0,Ar[0]); // 初期値の表示
for(i=1;i<N;i++){ // i=0 から開始すると A[i-1] が存在しないので i=1 から開始
    Ar[i]=Ar[i-1]+d; // 漸化式  $a_i = a_{i-1} + d$ 
    printf("Ar[%d]=%f\n",i,Ar[i]); //  $a_i$  の表示
}
// 漸化式（配列を用いない）
printf("-----\n");
printf("recurrence relation without array\n");
ai_1=a; // 初期値  $i = 1$  のとき,  $a_{i-1} = a_0 = a$ 
printf("a%d=%f\n",0,ai_1); // 初期値の表示
for(i=1;i<N;i++){ // 配列の時と同じように i=1 から開始
    ai=ai_1+d; // 漸化式  $a_i = a_{i-1} + d$ 
    printf("a%d=%f\n",i,ai); //  $a_i$  の表示
    ai_1=ai; // (重要)  $a_{i-1} = a_i$  として, 1 時刻前を保存する
}
return 0;
}

```

上記のプログラムを入力して実行せよ．4 つの方法で出力が同じであることを確認せよ．

スクリーンショット（実行画面＋ソースファイル）、およびソースファイルを提出せよ。

次に、このプログラムをもとにして、フィボナッチ数列を同様に4つの方法で求めるプログラムを以下に記述する。

フィボナッチ数列は、

$$\begin{aligned}a_0 &= 0 \\a_1 &= 1 \\a_i &= a_{i-1} + a_{i-2} \quad (i = 2, \dots, n-1)\end{aligned}\tag{5.1.1}$$

で定義される数列である。

一般項の式は以下となる。 $x^i$  の計算は、`pow(x,i)` を用いる。ただし、一番初めに `#include <math.h>` が必要で、コンパイル時には `-lm` オプションが必要。

$$\begin{aligned}a_i &= \frac{1}{\sqrt{5}} (\alpha^i (a_1 - \beta a_0) - \beta^i (a_1 - \alpha a_0)) \\ \text{ただし, } \alpha &= \frac{1 + \sqrt{5}}{2}, \quad \beta = \frac{1 - \sqrt{5}}{2}\end{aligned}\tag{5.1.2}$$

$a_i = a + id$  のプログラムを別名で保存する。 そのプログラムを以下のように変更する。「変更」「追加」の行が変更されているところである。

```
#include <stdio.h>
#include <math.h> // 追加
#define N 5 // n = 5 プログラム中の N を 5 に書き換えてからコンパイルする
double general_term(double a0, double a1, int i){ // 以下4行追加：一般項の関数
    double alpha=(1+sqrt(5))/2, beta=(1-sqrt(5))/2; // 式 (5.1.2)
    double ai=1/sqrt(5)*(pow(alpha,i)*(a1-beta*a0)-pow(beta,i)*(a1-alpha*a0));
    return ai;
}
int main(){
    double A[N]; // 配列 a0,...,a_{n-1} を定義（一般項の式用）
    double Ar[N]; // 漸化式用の配列を定義（r は recurrence relation（漸化式）の意味）
    int i;
    double a0=0, a1=1; // 変更
    double ai, ai_1, ai_2; // 変更：ai=a_i, ai_1=a_{i-1}, ai_2=a_{i-2}
    // 一般項の式（配列を用いない）
    printf("general terms without array\n");
    for(i=0;i<N;i++){
        ai=general_term(a0,a1,i); // 変更：一般項の式
        printf("a%d=%f\n",i,ai); // a_i の表示
    }
```

```

// 一般項の式（配列を用いる）
printf("-----\n");
printf("general terms with array\n");
for(i=0;i<N;i++){
    A[i]=general_term(a0,a1,i); // 変更：一般項の式
    printf("A[%d]=%f\n",i,A[i]); //  $a_i$  の表示
}
// 漸化式（配列を用いる）
printf("-----\n");
printf("recurrence relation with array\n");
Ar[0]=a0; // 変更： $i=0$  の初期値
Ar[1]=a1; // 追加： $i=1$  の初期値
printf("Ar[%d]=%f\n",0,Ar[0]); // 変更：初期値の表示
printf("Ar[%d]=%f\n",1,Ar[1]); // 追加：初期値の表示
for(i=2;i<N;i++){ // 変更： $i=1$  から開始すると  $A[i-2]$  が存在しないので  $i=2$  から開始
    Ar[i]=Ar[i-1]+Ar[i-2]; // 変更：漸化式  $a_i = a_{i-1} + a_{i-2}$ 
    printf("Ar[%d]=%f\n",i,Ar[i]); //  $a_i$  の表示
}
// 漸化式（配列を用いない）
printf("-----\n");
printf("recurrence relation without array\n");
ai_2=a0; // 追加：初期値  $i=2$  のとき,  $a_{i-2} = a_0$ 
ai_1=a1; // 変更：初期値  $i=2$  のとき,  $a_{i-1} = a_1$ 
printf("a%d=%f\n",0,ai_2); // 変更：初期値の表示
printf("a%d=%f\n",1,ai_1); // 変更：初期値の表示
for(i=2;i<N;i++){ // 変更：配列の時と同じように  $i=2$  から開始
    ai=ai_1+ai_2; // 変更：漸化式  $a_i = a_{i-1} + a_{i-2}$ 
    printf("a%d=%f\n",i,ai); //  $a_i$  の表示
    ai_2=ai_1; // 追加（重要）： $a_{i-2} = a_{i-1}$  として, 2 時刻前を保存
    ai_1=ai; // （重要）  $a_{i-1} = a_i$  として, 1 時刻前を保存する
    // （重要）この 2 行の順番が違うと動作しない
}
return 0;
}

```

このプログラムも入力して実行する。

4 つの方法の出力が一致することを確認する。スクリーンショット（実行画面＋ソースファイル）、およびソースファイルを提出せよ。上記のプログラムを別名で保存する。

これまでは、単に入力すれば動作する。これからが本当の問題である。ここまですもとにして、以下の

数列のプログラムを4つの方法で作成する.

これまでのプログラムとの変更点になるべく少なくなるように作成せよ.

$a_0 = 0, a_1 = 1, a_2 = 2$  として, 漸化式  $a_i = 3a_{i-1} + 2a_{i-2} - 4a_{i-3}, \quad (i \geq 3)$  により  $a_i$  の値を求める.  
 $a_i$  の一般項の式は以下となる.

$$a_i = \alpha^i c_1 + \beta^i c_2 + c_3 \quad (5.1.3)$$

$$\text{ただし, } \alpha = 1 - \sqrt{5}, \beta = 1 + \sqrt{5}$$

$$c_1 = \frac{1}{10}(\beta a_0 - (1 + \beta)a_1 + a_2)$$

$$c_2 = \frac{1}{10}(\alpha a_0 - (1 + \alpha)a_1 + a_2)$$

$$c_3 = \frac{1}{5}(4a_0 + 2a_1 - a_2)$$

4つの方法が一致することを確認せよ. スクリーンショットで出力を提出する.

また, 初期値を  $a_0 = 0.5, a_1 = 1.5, a_2 = 2.5$  とした場合も, 4つの方法が一致することを確認する. この出力のスクリーンショットも提出する.

(問題終)

## 5.2 数列の収束とグラフ

このプログラムをよく理解すれば, 以降のプログラム作成はかなり楽になるはずである.

**問題 5.2** 以下の漸化式を考える.

$$a_0 = 0$$

$$a_i = \frac{2}{3}a_{i-1} + 1 \quad (i = 1, 2, \dots)$$

この数列は,  $\lim_{i \rightarrow \infty} a_i = 3$  と収束する.

もし収束することがわかれば,  $\alpha = \lim_{i \rightarrow \infty} a_i$  とすると, 上の漸化式で  $i \rightarrow \infty$  として,  $\alpha = \frac{2}{3}\alpha + 1$  となり, これを解いて  $\alpha = 3$  となる.

$a_i$  が収束するまで  $a_i$  の計算を繰り返すプログラムを作成したい.

収束は, 「小さい正の  $\varepsilon$  (ここでは  $\varepsilon = 10^{-8}$  とする) に対して,  $|a_i - a_{i-1}| < \varepsilon$  となる」ということで判定する.

この方法は確実に収束を判定できるものではないことに注意しておく. 例えば,  $a_i = \sum_{k=1}^i \frac{1}{k}$  とする. 高校数学で扱ったと思われるが,  $a_i$  は無限大に発散する. しかし  $|a_i - a_{i-1}| = \frac{1}{i}$  なので, どんどん小さくなるので, いつかは収束すると判定されてしまう. ただし, この授業ではこのような場合は考慮しない.

収束するまでどれくらい繰り返せばよいかわからないので, 配列を用いることは適切ではない. もし, かならず収束するであろう大きい繰り返し回数を想定して, 非常に大きい配列を用いるとすると, 配列を確

保できなかったり、確保できてもメモリを大量に使うことになり、他のプログラムなどに影響が出る。

以下は、配列を用いずに作成したプログラムである。さらに  $i$  と  $a_i$  を `output.csv` というファイルに書き込んでいる。

以下では、if 文の条件を間違えたりして無限ループになることを避けるために、繰り返し回数の上限 `maxi` を定めている。

もし `maxi` を使いたくなければ、for 文を `for(i=0; ;i++)` のようにすれば `maxi` を使わないプログラムになる。ただし、少し間違えると無限ループになるのでお勧めしない。

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

int main(){
    int i,maxi=10000; // maxi は i の最大値（無限ループを避けるため）
    double a0=0.0;
    double ai, ai_1; // ai= $a_i$ , ai_1= $a_{i-1}$ 
    double eps=1e-8; //  $\varepsilon = 10^{-8}$ 
    FILE *fp; // ファイルポインタ（ファイルに使う変数）

    fp=fopen("output.csv","w"); // output.csv というファイルを
                                // w モード（write 書き込み）で開く
    fprintf(fp,"i,ai\n"); // i,ai という文字列をファイルに保存（グラフの凡例用）
    // "i,ai\n" の中にスペースは入れない（説明は以下）

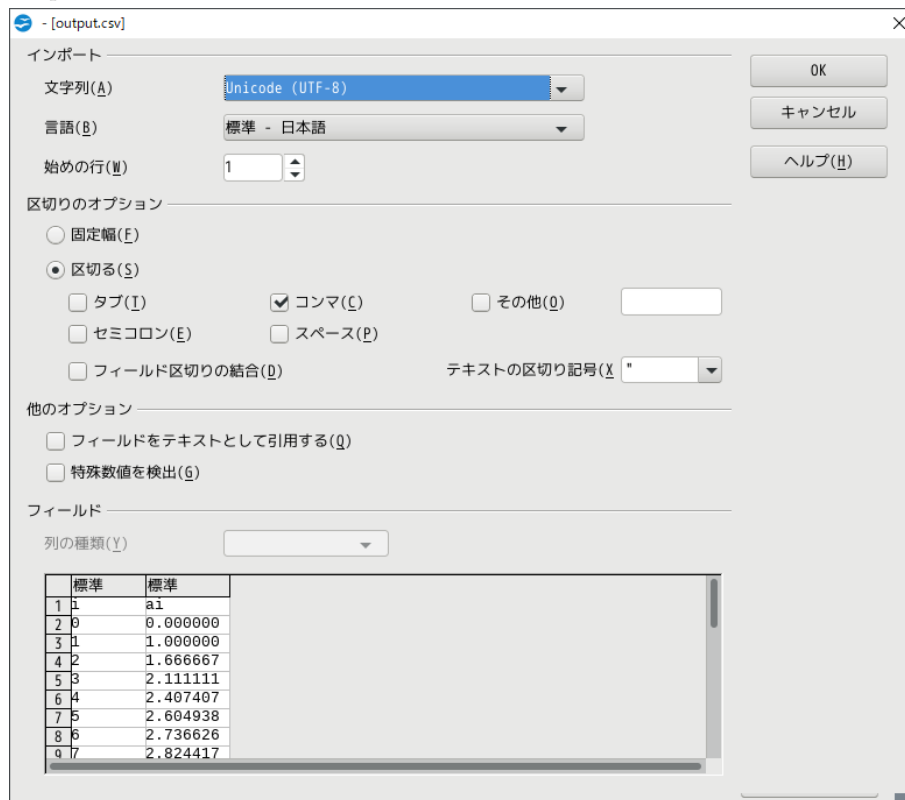
    ai_1=a0; // 初期値  $a_{i-1} = a_0$ 
    printf("a%d=%f\n",0,ai_1); //  $a_0$  の値を表示
    fprintf(fp,"%d,%f\n",0,ai_1); //  $a_0$  の値をファイルに保存
    // fprintf で csv ファイル（データをカンマで区切るファイル）を作るときは
    // "%d,%f\n" の中にスペースは入れない（以下の fprintf も同様）
    for(i=1;i<maxi;i++){
        ai=2.0/3.0*ai_1+1; //  $a_i = \frac{2}{3}a_{i-1} + 1$ 
        printf("i=%d ai=%f\n",i,ai); // i と  $a_i$  を表示
        fprintf(fp,"%d,%f\n",i,ai); // i と  $a_i$  をファイルに保存
        if(fabs(ai-ai_1)<eps){ //  $|a_i - a_{i-1}| < \varepsilon$  のとき（絶対値は fabs で求める）
            break; // for ループを終了（多重ループでは一番内側のループを終了）
        }
        ai_1=ai; // （重要） $a_{i-1} = a_i$  として、1 時刻前を保存
    }
    fclose(fp); // ファイルを閉じる
```

```
return 0; // 正常終了
}
```

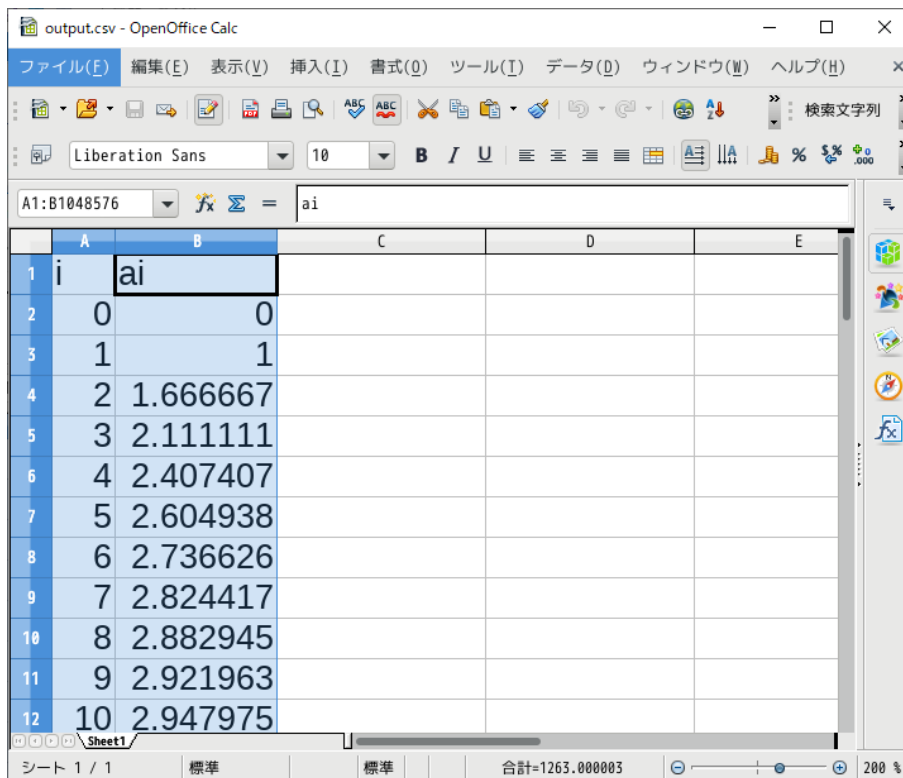
上記のプログラムを入力して実行せよ。実行すると output.csv というファイルができる。

OpenOffice を起動する。TeraTerm で、「openoffice4 -calc &」として起動する。output.csv を OpenOffice で開く。

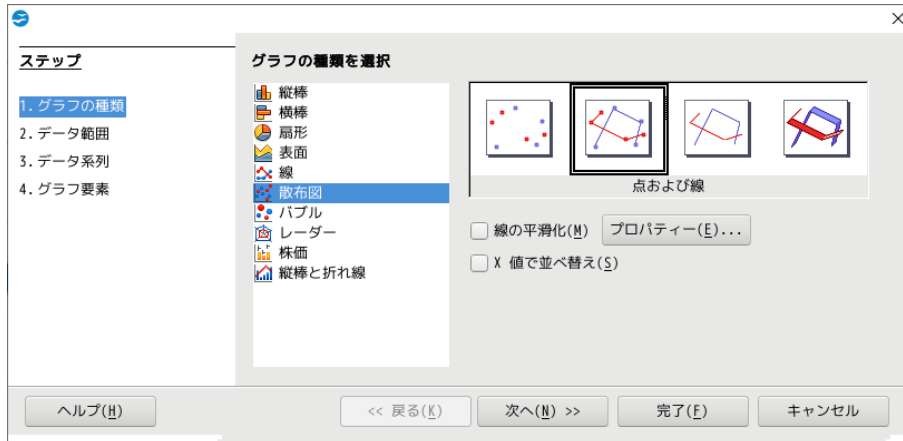
オプション等を以下のように設定（コンマにチェックを入れて、他のチェックを外す）して、OK をクリックする。ファイルを開くことに失敗したり、以下のような画面が出ない場合には、output.csv を削除して、output.csv をもう一度作成する。



データのある 2 列を指定（ドラッグ）する。すなわち、以下の図の A と B のところをドラッグして、左の 2 列を指定する。

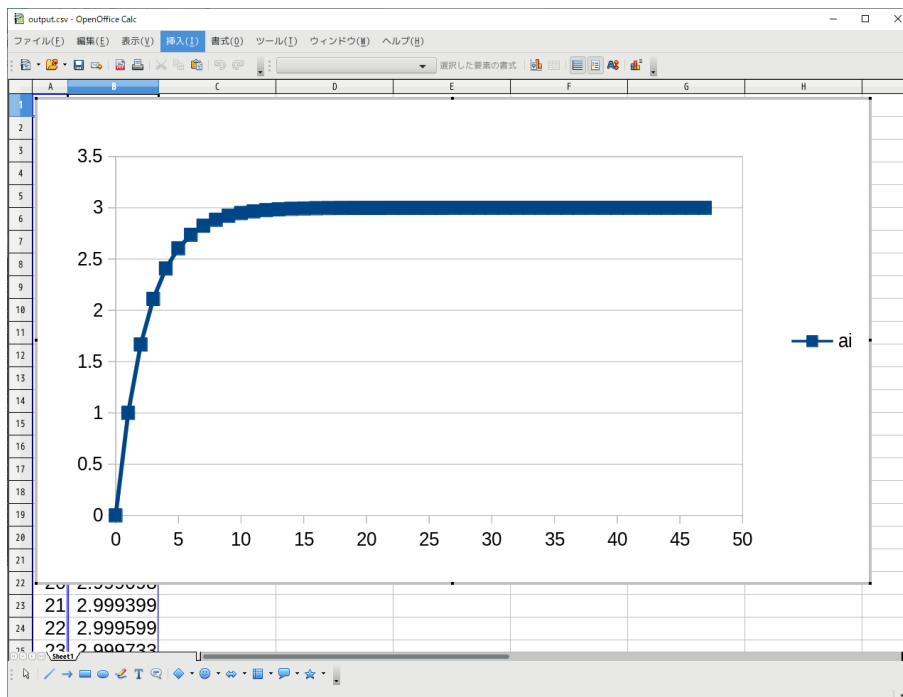


グラフのアイコンを押す。以下のように「散布図」「点および線」を指定して「完了」をクリックする。



すると、以下のようなグラフができる。





グラフと実行画面から 3 に収束していることがわかる。

上記のグラフを作成せよ。グラフを含めてセーブする際は、Microsoft Excel 97/2000/XP (.xls) の形式でセーブする。

まずはここまでを提出する。スクリーンショットは、ソースファイル、グラフ、実行画面の最後が見えるようにする。ソースファイルと output.csv とエクセルファイルも提出する。 .xls ファイルを添付する際は、OpenOffice を終了してから添付する。

次に、そのプログラムをもとにして、フィボナッチ数列  $a_i$  の隣接する 2 項の比  $b_i = a_i/a_{i-1}$  を求め、 $b_i$  が収束するまで繰り返すプログラムを以下に示す。  $\varepsilon = 10^{-8}$  に対して、 $|b_i - b_{i-1}| < \varepsilon$  となるまで繰り返すプログラムである。

$b_i$  は以下の式となる。

$$b_1 = a_1/a_0 \quad (b_i \text{ の初期値})$$

$$b_i = a_i/a_{i-1} \quad i = 2, 3, \dots$$

$a_0 = 0$  のときは、 $b_1$  は計算できないが、上の式のまま入力する  $a_0 = 0$  のとき、授業の環境では  $b_1 = \text{inf}$  (無限大の意味) になる。

$b_1$  の値はファイルに保存しない (無限大になることがあるため)。すなわち、 $b_i (i \geq 2)$  のみファイルに保存する。

フィボナッチ数列  $a_i$  は、漸化式を用いて、配列は用いずに作成している。

$b_i$  の一般項 ( $a_0 = a_1 = 0$  の場合は除く) は, 式 (5.1.2) より, 以下となる.

$$b_i = \frac{\alpha^i(a_1 - \beta a_0) - \beta^i(a_1 - \alpha a_0)}{\alpha^{i-1}(a_1 - \beta a_0) - \beta^{i-1}(a_1 - \alpha a_0)} \quad (5.2.1)$$

$$\text{ただし, } \alpha = \frac{1 + \sqrt{5}}{2} \quad \beta = \frac{1 - \sqrt{5}}{2}$$

$b_i$  は初期値  $a_0, a_1$  に関わらず  $\frac{1+\sqrt{5}}{2}$  に収束する (ただし,  $a_1 - \beta a_0 = 0$  の場合を除く).  
 $\frac{1+\sqrt{5}}{2}$  は黄金比と呼ばれる.

$b_i$  の極限值は,  $|\frac{\beta}{\alpha}| < 1$  より, 上の  $b_i$  の一般項の分子分母を  $\alpha^{i-1}$  で割ると, 以下ようになる.

$$\begin{aligned} \lim_{i \rightarrow \infty} b_i &= \lim_{i \rightarrow \infty} \frac{\alpha(a_1 - \beta a_0) - \beta(\frac{\beta}{\alpha})^{i-1}(a_1 - \alpha a_0)}{(a_1 - \beta a_0) - (\frac{\beta}{\alpha})^{i-1}(a_1 - \alpha a_0)} \\ &= \frac{\alpha(a_1 - \beta a_0)}{a_1 - \beta a_0} = \alpha = \frac{1 + \sqrt{5}}{2} \end{aligned}$$

今までのプログラムを別名で保存する. それを以下のように変更する. 「変更」「追加」の所を記述する.

```
#include <stdio.h>
#include <math.h>
#define N 5 // n = 5 プログラム中の N を 5 に書き換えてからコンパイルする

double general_term(double a0, double a1, int i){ // 以下 4 行追加: 式 (5.2.1)
    double alpha=(1+sqrt(5))/2, beta=(1-sqrt(5))/2;
    double bi=(pow(alpha,i)*(a1-beta*a0)-pow(beta,i)*(a1-alpha*a0))
    / (pow(alpha,i-1)*(a1-beta*a0)-pow(beta,i-1)*(a1-alpha*a0));
    return bi;
}

int main(){
    int i,maxi=10000;
    double a0=0, a1=1; // 変更
    double ai, ai_1, ai_2; // 変更: ai=ai, ai_1=ai-1, ai_2=ai-2
    double bi, bi_1, bi_formula; // 変更: bi=bi, bi_1=bi-1, bi_formula は一般項の式用
    double golden_ratio=(1+sqrt(5))/2; // 黄金比
    double eps=1e-8;
    FILE *fp;

    fp=fopen("output.csv","w");
    fprintf(fp,"i,ai\n");
```

```

// 漸化式（配列を用いない）
ai_2=a0; // 変更：初期値  $i = 2$  のとき,  $a_{i-2} = a_0$ 
ai_1=a1; // 追加：初期値  $i = 2$  のとき,  $a_{i-1} = a_1$ 
bi_1=a1/a0; // 追加：初期値  $i = 2$  のとき,  $b_{i-1} = b_1 = a_1/a_0$ 
printf("b%d=%f\n",1,bi_1); // 初期値の表示
for(i=2;i<maxi;i++){ // 変更：i=2 から開始
    ai=ai_1+ai_2; // 変更：漸化式  $a_i = a_{i-1} + a_{i-2}$ 
    bi=ai/ai_1; // 追加： $b_i = a_i/a_{i-1}$ 
    bi_formula=general_term(a0,a1,i);
    printf("i=%d bi=%f bi(formula)=%f\n",i,bi,bi_formula); //  $b_i$  と一般項の式の値を表示
    fprintf(fp,"%d,%f\n",i,bi);
    if(fabs(bi-bi_1)<eps){
        break;
    }
    ai_2=ai_1; // 追加（重要）： $a_{i-2} = a_{i-1}$  として, 2 時刻前を保存
    ai_1=ai; // （重要）  $a_{i-1} = a_i$  として, 1 時刻前を保存する
    // 上の 2 行の順序に注意.
    bi_1=bi; // 追加（重要）： $b_{i-1} = b_i$  として, 1 時刻前を保存
}
printf("limit(true value)=%f\n",golden_ratio); // 極限値の正しい値を表示
return 0;
}

```

このプログラムを入力して実行し、以下を提出する。

- ソースファイル
- スクリーンショット。ただし、グラフ、ソースファイル、実行画面（ $i = 2, 3, 4, 5$  と収束値が見える）を入れる
- OpenOffice で「Microsoft Excel 97/2000/XP (.xls) の形式」でデータとグラフをセーブした .xls のファイル（.xls ファイルを添付する際は、OpenOffice を終了してから添付）
- output.csv

ここまでは、そのまま入力すれば動作するはずである。これからが本当の問題である。  
 これまでのプログラムとの変更点がなるべく少なくなるように作成せよ。

問題 5.1 の最後の  $a_i$  に対して、 $b_i = \frac{a_i - a_{i-1}}{a_{i-1} - a_{i-2}}$ , ( $i \geq 3$ ) を求めて収束するまで続ける。

すなわち、 $a_0 = 0, a_1 = 1, a_2 = 2$  として、漸化式  $a_i = 3a_{i-1} + 2a_{i-2} - 4a_{i-3}$ , ( $i \geq 3$ ) により  $a_i$  の値を求めて、それを用いて  $b_i$  を求める。 $b_i$  の初期値は  $b_2 = \frac{a_2 - a_1}{a_1 - a_0}$  とする。

$b_i$  の一般項の式は、式 (5.1.3) より以下となる。

$$b_i = \frac{(\alpha^i - \alpha^{i-1})c_1 + (\beta^i - \beta^{i-1})c_2}{(\alpha^{i-1} - \alpha^{i-2})c_1 + (\beta^{i-1} - \beta^{i-2})c_2} \quad (5.2.2)$$

$\alpha, \beta, c_1, c_2$  は問題 5.1 の最後と同じ。

$|\beta| > |\alpha|$ ,  $|\beta| > 1$  なので、 $b_i$  は、この式より  $\beta = 1 + \sqrt{5}$  に収束することがわかる。

漸化式により  $a_i$  を求めて、それを用いて  $b_i$  を計算したものと、一般項の式が一致することを確認せよ。

また、最後に  $\beta$  を表示して、収束した値と一致することを確認せよ。

`printf` による表示は、「 $i$ 」, 「漸化式による  $a_i$  を用いた  $b_i$ 」, 「式 (5.2.2) による一般項  $b_i$ 」の 3 つとする。それより多くても良いが、見やすいようにすること。

`printf` の出力例

```
a0=...
a1=...
a2=...
i=2  bi=...  bi(formula)=...
i=3  bi=...  bi(formula)=...
....
i=... bi=... bi(formula)=...
limit(true value)=...
```

`fprintf` によるファイルへの保存は、「 $i$ 」, 「漸化式による  $a_i$  を用いた  $b_i$ 」の 2 つとする。それより多くても良いが、見やすいようにすること。

`for` 文などのループは、ソースファイル全体で一つしか使わないようにすること。

プログラムの最後に、 $1 + \sqrt{5}$  を表示して、収束した値と一致することを確認せよ。

$a_0 = 0$ ,  $a_1 = 1$ ,  $a_2 = 2$  の場合と、 $a_0 = 0.5$ ,  $a_1 = 1.5$ ,  $a_2 = 2.5$  の場合を実行する。「漸化式による  $a_i$  を用いた  $b_i$ 」と「一般項  $b_i$ 」が一致することを確認する。特に  $i = 2, 3, 4, 5$  あたりで一致することを確認する。スクリーンショットでもそれがわかるようにする。また、 $1 + \sqrt{5}$  に収束していることがスクリーンショットでわかるようにする。

以下を  $a_0 = 0, a_1 = 1, a_2 = 2$  と  $a_0 = 0.5, a_1 = 1.5, a_2 = 2.5$  に対して提出する。

- ソースファイル
- スクリーンショット。ただし、グラフ、ソースファイル、実行画面 ( $i = 2, 3, 4, 5$  と収束値が見える) を入れる
- OpenOffice で「Microsoft Excel 97/2000/XP (.xls) の形式」でデータとグラフをセーブした .xls のファイル (.xls ファイルを添付する際は、OpenOffice を終了してから添付)
- output.csv

(問題終)

**追加問題 5.1** 次の数列  $a_n$  が収束するか発散するかを判定し表示せよ（収束：convergence, 非収束：nonconvergence）. 繰り返し回数も表示せよ. ただし配列は用いずに作成する. 初期値の値も表示する.

収束は, 小さい正の数  $\varepsilon$  (例えば  $\varepsilon = 10^{-9}$ ) に対して  $|a_n - a_{n-1}| < \varepsilon$  のとき収束したと考えて繰り返しを終了する.

非収束は, 大きい正の数  $L$  (例えば  $L = 10^{20}$ ) に対して  $a_n > L$  となったら非収束で  $\infty$  に発散と考えて infinity と表示して繰り返しを終了する.  $a_n < -L$  となったら非収束で  $-\infty$  に発散と考えて -infinity と表示して繰り返しを終了する (振動する場合も判定すべきであるが, ここでは省略する).

収束する場合には極限值として  $a_n$  を表示する. 非収束の場合にも  $a_n$  を表示して収束していないことを確認する.

1. 初期値  $a_0 = 2$ , 漸化式  $a_n = 2a_{n-1} - 1$  ( $n \geq 1$ ) で定義される数列. もし収束する場合,  $\lim_{n \rightarrow \infty} a_n = \alpha$  とすると,  $\alpha = 2\alpha - 1$  となるのでこれを解いて  $\alpha = 1$  となるが, これに収束するだろうか.

$a_0 = 1$  にした場合はどうなるか.  $a_0 = -1$  にした場合はどうなるか.

最後に初期値, および収束値 (または  $\infty, -\infty$ ) を表示する.

もしチェックのために一般項を知りたいければ, Mathematica で

`RSolve[{a[n]==2*a[n-1]-1, a[0]==2}, a[n], n]` と入力して shift + Enter すると,  $a_0 = 2$  の一般項が得られる.

2. 初期値  $a_0 = 0, a_1 = 1$ , 漸化式  $a_n = \frac{1}{3}a_{n-1} + \frac{1}{6}a_{n-2} + 1$  ( $n \geq 2$ ) で定義される数列. もし収束する場合, 同様に  $\alpha$  を計算すると何になるだろうか. 実際にその値に収束するだろうか.

もしチェックのために一般項を知りたいければ, Mathematica で

`RSolve[{a[n]==1/3*a[n-1]+1/6*a[n-2]+1, a[0]==0, a[1]==1}, a[n], n]` と入力して shift + Enter すると, 一般項が得られる. 上記は  $a_0 = 0, a_1 = 1$  の場合だが, 一般の初期値の場合には `a[0]==a0, a[1]==a1` と書き換える.

このようにして得られた  $a_0 = 0, a_1 = 1$  のときの一般項は以下となる.

$$a_n = 2 + \left( \frac{2\sqrt{7}}{7} - 1 \right) \beta^n - \left( \frac{2\sqrt{7}}{7} + 1 \right) \gamma^n, \quad \beta = \frac{1 - \sqrt{7}}{6}, \quad \gamma = \frac{1 + \sqrt{7}}{6}$$

教官から, プログラムが間違っているといわれた場合には, 一般項も表示して比較すること.

(追加問題終)

**追加問題 5.2** 問題 4.2 では, 数値積分の値と正解を比較していたが, 一般には正解と比較することはできない. 正解がわからない場合に,  $n$  の値が適切であるかを判断する必要がある. ここでは数値積分の値が収束するまで  $n$  を大きくすることで適切な  $n$  と積分値を求める.

まず, 合成台形公式に対して,  $n$  を 1 ずつ大きくしていき, 収束したら  $n$  と積分の値を出力するプログラムを作成せよ. 合成台形公式を関数にすることが望ましい ( $n$  を関数の引数に含むようにする).  $n$  に対する合成台形公式の値を  $I_n^{(trap)}$  として,  $\varepsilon = 10^{-8}$  などに対して  $|I_n^{(trap)} - I_{n-1}^{(trap)}| < \varepsilon$  となったときに収束したと判定する.  $I_n^{(trap)}$  の初期値は  $\infty$  とする. プログラム中では  $I_n^{(trap)}$  は `I_trap_n`,  $I_{n-1}^{(trap)}$  は `I_trap_n1` などとする.

同様に、合成第 1 シンプソン公式の値を  $I_n^{(1simp)}$ 、合成第 2 シンプソン公式の値を  $I_n^{(2simp)}$  とし、それらについてもプログラムを作成し、3 つの方法で同じ  $\varepsilon$  に対する  $n$  の値を比較せよ。もし  $n$  の値に違いが無いようであれば、 $\varepsilon$  を小さくするなどの変更をすること。  
(追加問題終)

**追加問題 5.3** 追加問題 5.2 では、 $n$  を一つ増やしたときに計算をすべてやり直すことになり時間がかかる。ここでは無駄な計算を少なくする。

この追加問題では、合成台形公式を扱う。 $n$  は、整数  $m$  に対して  $n = 2^m$  となる場合のみを考える。 $J_m^{(trap)} = I_{2^m}^{(trap)}$  と書くことにする。

このとき、 $h_m = \frac{b-a}{2^m}$  として (すなわち、 $h_1 = \frac{b-a}{2}$ ,  $h_{m+1} = \frac{h_m}{2}$ ),  $x_i^{(m)} = a + ih_m$  とすると、以下となる。

$$J_m^{(trap)} = I_{2^m}^{(trap)} = \frac{h_m}{2} \left( f(a) + 2 \sum_{i=1}^{2^m-1} f(x_i^{(m)}) + f(b) \right)$$

$$S_m^{(trap)} = \sum_{i=1}^{2^m-1} f(x_i^{(m)}) \text{ とする. } J_m^{(trap)} = \frac{h_m}{2} \left( f(a) + 2S_m^{(trap)} + f(b) \right) \text{ である.}$$

$m$  に対する  $S_m^{(trap)}$  がわかっているとして、 $m+1$  に対して  $S_{m+1}^{(trap)}$  を考える。 $h_{m+1} = h_m/2$  であり、 $x_i^{(m+1)} = a + ih_{m+1} = a + \frac{i}{2}h_m$  である。 $i$  が偶数 ( $i = 2k$ ) のとき、 $x_{2k}^{(m+1)} = a + kh_m = x_k^{(m)}$  となるので、これらの計算は省略できるはずである。

すなわち、 $i$  を偶数 ( $i = 2k$ ) と奇数 ( $i = 2k+1$ ) に分けて考える。

$$\begin{aligned} S_{m+1}^{(trap)} &= \sum_{i=1}^{2^{m+1}-1} f(x_i^{(m+1)}) = \sum_{k=1}^{2^m-1} f(x_{2k}^{(m+1)}) + \sum_{k=0}^{2^m-1} f(x_{2k+1}^{(m+1)}) = \sum_{k=1}^{2^m-1} f(x_k^{(m)}) + \sum_{k=0}^{2^m-1} f(x_{2k+1}^{(m+1)}) \\ &= S_m^{(trap)} + \sum_{k=0}^{2^m-1} f(x_{2k+1}^{(m+1)}) \end{aligned}$$

$S_m^{(trap)}$  の初期値は、 $m = 1$  に対して

$$S_1^{(trap)} = \sum_{i=1}^{2-1} f(x_i^{(1)}) = f(x_1^{(1)}) = f\left(a + \frac{b-a}{2}\right) = f\left(\frac{a+b}{2}\right)$$

となる。これにより  $S_m^{(trap)}$  の漸化式ができるので、これを用いて  $S_m^{(trap)}$  および  $J_m^{(trap)}$  を求めて  $J_m^{(trap)}$  の収束値を求める。 $m$  の値と  $J_m^{(trap)}$  を出力する。 $h_m$  の計算や、for 文の範囲の  $2^m - 1$  の計算は、pow を用いずに漸化式により計算する。そのほうが高速である。

このプログラムを作成し、 $n = 2^m$  に対して追加問題 5.2 の結果と比較して正しく求められていることを確認せよ。さらに計算時間を測定せよ。計算時間については補足資料を参照すること (配布していなければ教官に請求する)。追加問題 5.2 の合成台形公式の実行時間と比較せよ。  
(追加問題終)

**追加問題 5.4** 上の追加問題と同様に、合成第 1 シンプソン公式でも  $n = 2^m$  に対して  $\sum$  に関する漸化式を求めてプログラムを作成せよ。

合成第 2 シンプソン公式では,  $n = 3^m$  に対して漸化式を求めてプログラムを作成せよ.

これらの漸化式とその求め方も提出すること (手書きのスキャンでも可).

3つのプログラムが完成したら, 同じ  $\varepsilon$  に対する  $n$  を比較せよ ( $m$  の比較ではない). また, それぞれの方法に関して, 追加問題 5.2 と実行時間を比較せよ. (追加問題終)

## 6 非線形方程式

ここでは非線形方程式  $f(x) = 0$  の解を数値的に求める方法について扱う。

方程式が

$$f(x) = ax + b, (a \neq 0)$$

のとき、方程式は線形または1次方程式と呼ばれ、それ以外るとき非線形方程式と呼ばれる。

非線形方程式の中でも  $f(x)$  が  $n$  次の多項式、すなわち

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

の場合、代数方程式という。

$f(x)$  が4次式以下であれば解の公式が存在するが、それより大きい次数のときは解の公式はない。また一般的な非線形方程式については解の公式は存在しない。

そのため、数値的に解を求めることが必要になる。

### 6.1 2分法

2分法とは微積分学の中間値の定理を基礎としたものである。

**定理 6.1** (中間値の定理) 関数  $g(x)$  が  $a \leq x \leq b$  に対して連続で  $g(a) \neq g(b)$  であれば  $g(a)$  と  $g(b)$  の間の任意の値  $\alpha$  に対して  $g(x_0) = \alpha$  となる  $a \leq x_0 \leq b$  が存在する。

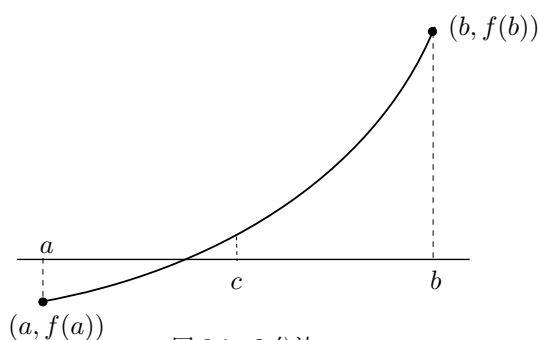


図 6.1 2分法

2分法による非線形方程式の数値解のアルゴリズムは以下のようになる。

$f(x) = 0$  となる  $x$  を求める。

1.  $f(a)$  と  $f(b)$  の符号が反対になる  $a, b$  を見つける (ただし  $a < b$ )。すなわち「 $f(a) < 0$  かつ  $f(b) > 0$ 」または「 $f(a) > 0$  かつ  $f(b) < 0$ 」となることである。このような  $a, b$  はランダムに探すか、または既知とする。 $f(a)$  と  $f(b)$  の符号が反対になることは  $f(a)f(b) < 0$  として確かめることができる。(この授業では、初期値  $a, b$  の設定についてはプログラムでは行わずに、問題文に従って設定する。)
2.  $c_1 = a$  とする。(  $c_1$  は1ステップ前の解の候補)
3.  $c = \frac{a+b}{2}$  とする ( $c$  は  $a$  と  $b$  の中点、 $c$  は解の候補である)。



4.  $f(c) = 0$  であれば、解  $x = c$  として終了.
5.  $f(c)$  と  $f(a)$  の符号が同じであれば  $a = c$  と置く.  $b$  はそのまま. 符号が同じかどうかは  $f(c)f(a) > 0$  で判定する.
6.  $f(c)$  と  $f(a)$  の符号が異なれば、 $b = c$  と置く.  $a$  はそのまま.
7. 終了条件を調べ、満足されていれば、解  $x = c$  として終了. 満足されていなければ、 $c_1 = c$  として 3. に戻る.

終了条件は、以下のものを用いる.

- あらかじめ設定した繰り返し回数の上限  $n$  を超えたら終了する (無限ループを回避).
- 十分小さい正の数  $\varepsilon$  に対して  $|c - c_1| < \varepsilon$  となったら終了 ( $c$  は収束したので終了).
- 十分小さい正の数  $\delta$  に対して  $|f(c)| < \delta$  となったら終了 ( $f(c) = 0$  とみなして解が求まった).

これらの条件は複合で用いられることが多い. 必要な解の精度が与えられている場合には、その値を  $\varepsilon$  として用いる. ただし、何らかの理由でその精度が永久に得られない場合には無限ループになってしまうので、繰り返し回数に関する終了条件も併用する. ここで注意しなければならないのは、繰り返し回数を小さく設定してしまうと、ほとんどの場合で目的の精度が得られなくなることである. したがって繰り返し回数はある程度大きく設定する必要がある.

$f(x)$  のまるめ誤差等の計算誤差を見積もることが可能な場合には、その誤差を  $\delta$  に設定する. 計算誤差以上の精度で解を求めることは無意味だからである.

## 6.2 ニュートン法

ニュートン法は、一般的に 2 分法より速く収束する.

ニュートン法では  $f(x)$  は微分可能であると仮定する.

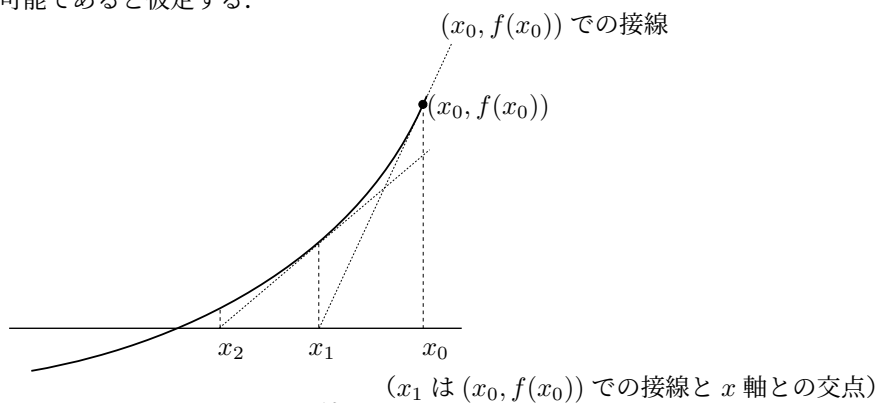


図 6.2 ニュートン法

1. 解の候補の初期値を  $x_0$  とする.
2.  $x_0$  から  $f(x_0)$ ,  $f'(x_0)$  を求め、 $x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$  とする.  $f'(x)$  は  $\frac{df(x)}{dx}$  のことである.  
 $x_1$  は、点  $(x_0, f(x_0))$  における  $f(x)$  の接線と  $x$  軸の交点である.  
 ただし  $f'(x_0) \neq 0$  と仮定する.  $f'(x_0) = 0$  のときは、 $x_0$  の値を少しずらしたりする. もしくは 1. の初期値をランダムに取り直して、はじめから再実行することもある. (この授業では、 $f'(x_0) = 0$  の時

の処理については行わなくてよい)

3.  $|x_1 - x_0| < \varepsilon$  であれば終了 ( $x_1$  は収束したので終了). または  $|f(x_1)| < \delta$  であれば終了 ( $f(x_1) = 0$  とみなして解が求まったので終了). また, 繰り返し回数があらかじめ設定した  $n$  を超えたら終了する (無限ループを回避). そうでなければ  $x_0 = x_1$  として 2. に戻る. 解は  $x = x_1$  であることに注意すること.

ニュートン法は一般には収束が速いという利点があるが, 次のような場合には, ニュートン法は収束しないことがあるので注意が必要である.

- 解の近くに変曲点がある場合
- 解の近くに極小点や極大点がある場合

**問題 6.1** (p.97, p.98, p.104, p.107 参照) 次の方程式  $f(x)$  に対し,  $f(x)$  の解を 2 分法, ニュートン法で求め, 収束の速さを比較せよ. すなわち, 同程度の精度を得るのに必要な繰り返し回数を比較せよ. 求めた解に対し, 解  $x$  と  $f(x)$  と繰り返し回数の 3 つを表示せよ.  $f(x)$  の表示には, %f ではなく %e を用いよ. 変数には double を用いること. 終了条件の絶対値には fabs を用いる ( $|x| = \text{fabs}(x)$ ).  $f(x)$  および  $f'(x)$  は関数を用いて作ること (p.104 参照).  $f'(x)$  については手で微分を計算する (プログラムで微分の計算はしない).  $f'(x)$  の関数名は df などとする (f' という関数名は使えない).

出力を見て, 解  $x$  に対して,  $f(x)$  が 0 に十分近いときは解は正しいと考えられる. そうでないときは, プログラムが間違っているか, もしくは  $\varepsilon$  や  $\delta$  の設定が不適切であることがほとんどである. 後者の場合には  $\varepsilon$  や  $\delta$  をもっと小さく変更せよ. プログラムでは  $\varepsilon$  は eps,  $\delta$  は delta などと変数名がわかるようにする.  $\text{eps} = 0.000000001$  などとしないこと (0 を数えるのが大変).  $\text{eps} = 1\text{e-}9$  などとすること. これは  $\varepsilon = 1 \times 10^{-9} = 0.000000001$  を意味する.

- $f(x) = \tan(x) - 2x$ , ただし  $x \neq 0$  の解を求めよ.

2 分法の初期値  $a, b$  の設定についてのヒント:  $f(\pi/4) = \tan(\pi/4) - 2\pi/4 = 1 - \pi/2 < 0$  なので  $a \equiv \pi/4$  とすれば  $f(a) < 0$  となる.  $\lim_{x \rightarrow \frac{\pi}{2}-0} f(x) = +\infty$  なので,  $b \equiv \pi/2$  かつ  $b < \pi/2$  とすれば  $f(b) > 0$  となる.

$$\left( \text{Newton 法のヒント: } \tan(x)' = \frac{1}{\cos^2(x)} \right)$$

プログラムでは C 言語の関数を用いよ.

よくわからなければ, まずは 2 分法について, p.104 をみて, 上記の  $f(x)$  の関数を作成し, p.102 などを見て for 文を作る. for 文の {...} の中にアルゴリズムの中の繰り返す部分を記入する (繰り返さない部分は for の外にする). if 文を用いて  $f(c)f(a) > 0$  などの判定をして, その時の処理を作成する. p.107 を見て, if 文と break 文を作る. if 文の条件には終了条件を記入する (～であればの部分). 「終了」は break で作成する. if 文と break 文は終了条件に応じて複数入れたほうがわかりやすい. (そうすると「かつ」や「または」およびその「否定」についてあまり知らなくてもできる) ニュートン法も基本的な形はほぼ同じ.

出力 (例)

```
---- bisection method -----  
x=....  
f(x)=.....  
iteration=....  
  
---- Newton's method -----  
x=...  
f(x)=.....  
iteration=....
```

1つのプログラムで2つの方法を行う必要はない. 2つのプログラムで2つの方法をそれぞれ行うことでも良い.

(問題終)

**追加問題 6.1** 次の順序で  $\pi$  を計算する. 最終的には `math.h` は用いずに加減乗除のみで  $\pi$  の計算を行う.  $\pi$  の桁数を非常に多く求める際などには, `math.h` を用いることはできないので, 以下のような方法を用いる.

以下の各ステップのプログラムと実行結果を提出する.

1.  $\pi$  の計算 (ガウス＝ルジャンドルのアルゴリズム)

以下のアルゴリズムは収束が非常に速く, ガウスはこのアルゴリズムを用いて  $\pi$  の値を 12 桁まで正しく求めている. 近年では  $\pi$  を 2 兆桁求める際にも用いられている. このアルゴリズムは加減乗除しか用いないので `math.h` が使えないような環境でも実行可能である. (コンピュータは, 基本的には加減乗除しかできないので, `math.h` を使う場合でも最終的には加減乗除の繰り返しによって数値を求めている. ただしユーザからは通常はその加減乗除を見ることはできない)

以下のアルゴリズムを用いて  $\pi$  を計算し, 真値 (M\_PI) と比較せよ. 真値との引き算をすることで, 何桁正しい値かを確認する. また繰り返し回数を表示せよ. ただし配列は用いずに作成すること. ここでは `math.h`, `sqrt`, `fabs` を用いてよい. `pow` は用いない.

(a) 初期値を次のように定める.  $a_0 = 1$ ,  $b_0 = \frac{1}{\sqrt{2}}$ ,  $t_0 = \frac{1}{4}$ ,  $p_0 = 1$ ,  $\varepsilon =$  小さい正の数 (例えば  $\varepsilon = 10^{-9}$ ).

(b) 以下の計算を行う.

$$\begin{aligned}a_{n+1} &= \frac{a_n + b_n}{2} \\b_{n+1} &= \sqrt{a_n b_n} \\t_{n+1} &= t_n - p_n (a_n - a_{n+1})^2 \\p_{n+1} &= 2p_n\end{aligned}$$

$|a_{n+1} - a_n| < \varepsilon$  かつ  $|b_{n+1} - b_n| < \varepsilon$  のとき (c) に進む. そうでなければ (b) に戻る.  
(c) 以下の値が  $\pi$  の近似値となる.

$$\frac{1}{4t_{n+1}}(a_{n+1} + b_{n+1})^2$$

## 2. 平方根および絶対値の計算

$f(x) = x^2 - a$  として,  $f(x) = 0$  となる  $x$  をニュートン法で求めれば  $x = \sqrt{a}$  となる.

ニュートン法を用いて  $a > 0$  に対して平方根  $\sqrt{a}$  を求める関数 `mysqrt(a)` を作成する. また実数  $a$  に対して絶対値を求める関数 `myfabs(a)` を作成する. これらの関数は `math.h` を用いずに動くようにする. 平方根を求めるには, ニュートン法以外に開平法もあるがここでは扱わない.

`sqrt` と `mysqrt` を比較して, 正しく計算できているか確認する. 確認するためだけに `math.h` と `sqrt` を使用する.

次に `if` 文を用いて絶対値を求める関数 `myfabs` を作成し, 正しくできていることを確認する.

- 1 の `sqrt` を `mysqrt` にして, `fabs` を `myfabs` にして, `M_PI` と比較して何桁正しいかを確認せよ. ここでは `M_PI` を使うために `math.h` が必要. `M_PI` 以外には `math.h` は使用しない.
4. `M_PI` との比較部分を削除して, `math.h` を用いずに加減乗除のみで  $\pi$  を計算するプログラムを作成せよ.

(追加問題終)

## 6.3 ニュートン法 (非線形連立方程式の場合)

非線形の連立方程式にもニュートン法を適用することができる.

$n$  変数  $x_1, x_2, \dots, x_n$  に対する  $n$  個の方程式を考える.

$$\begin{aligned} f_1(x_1, x_2, \dots, x_n) &= 0 \\ f_2(x_1, x_2, \dots, x_n) &= 0 \\ &\vdots \\ f_n(x_1, x_2, \dots, x_n) &= 0 \end{aligned}$$

このとき

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad F(X) = \begin{bmatrix} f_1(x_1, x_2, \dots, x_n) \\ f_2(x_1, x_2, \dots, x_n) \\ \vdots \\ f_n(x_1, x_2, \dots, x_n) \end{bmatrix}$$

とし,

$$F: \mathbf{R}^n \longrightarrow \mathbf{R}^n$$

とすると, ニュートン法の反復の式は以下ようになる.

$$X_1 = X_0 - J^{-1}(X_0)F(X_0) \quad (6.3.1)$$

ただし,  $J(X)$  は  $F(X)$  のヤコビ行列で以下のように定義され, 1 変数の場合の  $f'(x)$  に対応する.  $J^{-1}(X_0)$  は  $J(X)$  の逆行列に  $X = X_0$  を代入したものであり, 1 変数の場合の  $\frac{1}{f'(x_0)}$  に対応する.

$$J(X) = \left[ \frac{\partial f_i(X)}{\partial x_j} \right] = \begin{bmatrix} \frac{\partial f_1(X)}{\partial x_1} & \frac{\partial f_1(X)}{\partial x_2} & \cdots & \frac{\partial f_1(X)}{\partial x_n} \\ \frac{\partial f_2(X)}{\partial x_1} & \frac{\partial f_2(X)}{\partial x_2} & \cdots & \frac{\partial f_2(X)}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n(X)}{\partial x_1} & \frac{\partial f_n(X)}{\partial x_2} & \cdots & \frac{\partial f_n(X)}{\partial x_n} \end{bmatrix} \quad (6.3.2)$$

終了条件は,  $|X| = \sum_{i=1}^n |x_i|$  として扱えばよい (この  $|X|$  を  $X$  の  $L^1$  ノルムという). すなわち,

$$X_1 = \begin{bmatrix} x_1^{(1)} \\ \vdots \\ x_n^{(1)} \end{bmatrix}, \quad X_0 = \begin{bmatrix} x_1^{(0)} \\ \vdots \\ x_n^{(0)} \end{bmatrix}$$

とすると,  $|X_1 - X_0| = |x_1^{(1)} - x_1^{(0)}| + |x_2^{(1)} - x_2^{(0)}| + \cdots + |x_n^{(1)} - x_n^{(0)}| < \varepsilon$  のとき終了, または  $|F(X_1)| = |f_1(X_1)| + |f_2(X_1)| + \cdots + |f_n(X_1)| < \delta$  のとき終了, などとする.

**例 6.1** 次の非線形連立方程式のニュートン法を作る.

$$\begin{aligned} x^2 + y^2 &= 3 \\ xy &= 1 \end{aligned}$$

$f_1(x, y) = x^2 + y^2 - 3$ ,  $f_2(x, y) = xy - 1$  とすると,  $f_1(x, y) = 0, f_2(x, y) = 0$  となる  $x, y$  を求めることになる.

$$\begin{aligned} \frac{\partial f_1(x, y)}{\partial x} &= 2x, & \frac{\partial f_1(x, y)}{\partial y} &= 2y \\ \frac{\partial f_2(x, y)}{\partial x} &= y, & \frac{\partial f_2(x, y)}{\partial y} &= x \end{aligned}$$

なので,

$$J(x, y) = \begin{bmatrix} 2x & 2y \\ y & x \end{bmatrix}$$

となり,

$$J^{-1}(x, y) = \frac{1}{2(x^2 - y^2)} \begin{bmatrix} x & -2y \\ -y & 2x \end{bmatrix}$$

となる.

$2 \times 2$  の行列  $J$  の逆行列は以下の式で計算することに注意する.

$$J = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \text{ のとき } J^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix} \quad \text{ただし, } ad - bc \neq 0$$

式 (6.3.1) において,

$$X_0 = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}, \quad X_1 = \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

とする.

$$F(x_0, y_0) = \begin{bmatrix} f_1(x_0, y_0) \\ f_2(x_0, y_0) \end{bmatrix} = \begin{bmatrix} x_0^2 + y_0^2 - 3 \\ x_0 y_0 - 1 \end{bmatrix}$$

となる. これより

$$J^{-1}(x_0, y_0)F(x_0, y_0) = \frac{1}{2(x_0^2 - y_0^2)} \begin{bmatrix} x_0 & -2y_0 \\ -y_0 & 2x_0 \end{bmatrix} \begin{bmatrix} x_0^2 + y_0^2 - 3 \\ x_0 y_0 - 1 \end{bmatrix} = \frac{1}{2(x_0^2 - y_0^2)} \begin{bmatrix} x_0^3 - x_0 y_0^2 - 3x_0 + 2y_0 \\ -y_0^3 + x_0^2 y_0 - 2x_0 + 3y_0 \end{bmatrix}$$

となる. したがって, 式 (6.3.1) は

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} - \frac{1}{2(x_0^2 - y_0^2)} \begin{bmatrix} x_0^3 - x_0 y_0^2 - 3x_0 + 2y_0 \\ -y_0^3 + x_0^2 y_0 - 2x_0 + 3y_0 \end{bmatrix} \quad (6.3.3)$$

となる.

終了条件は,  $|x_1 - x_0| + |y_1 - y_0| < \varepsilon$  または  $|x_1^2 + y_1^2 - 3| + |x_1 y_1 - 1| < \delta$  となる.

**追加問題 6.2** 例 6.1 について式 (6.3.3) を用いてプログラムを作成して非線形連立方程式の解を求めよ. 求めた解に対し, 解  $X = (x, y)$  と  $|F(X)|$  と繰り返し回数を表示せよ.  $|F(X)|$  の表示には, %f ではなく %e を用いよ.

$|F(X)|$  が十分 0 に近い値であれば, 解は正しいと考えられる. そうでない場合は, プログラムが間違っているか,  $\varepsilon$  や  $\delta$  の設定が不適切であることがほとんどである. 後者の場合には  $\varepsilon$  や  $\delta$  の値を小さく変更せよ. (追加問題終)

**追加問題 6.3** 次の連立方程式について例 6.1 および上の問題と同様にして式を作成し, 非線形連立方程式の解を求めよ.

$$\begin{aligned} x^2 + 4y^2 &= 4 \\ xy &= 1 \end{aligned}$$

(追加問題終)

## 7 連立 1 次方程式

### 7.1 概説

$n$  個の未知数  $x_i$ , ( $i = 0, \dots, n-1$ ) について連立 1 次方程式の解を数値的に求める. 通常は添え字は  $i = 1, \dots, n$  とすることが多いが, C 言語の配列の添え字との対応がわかりやすいように  $i = 0, \dots, n-1$  とする. 以下の方程式の添え字も同様である.

$$\begin{aligned} a_{00}x_0 + a_{01}x_1 + \cdots + a_{0,n-1}x_{n-1} &= b_0 \\ a_{10}x_0 + a_{11}x_1 + \cdots + a_{1,n-1}x_{n-1} &= b_1 \\ &\vdots \\ a_{n-1,0}x_0 + a_{n-1,1}x_1 + \cdots + a_{n-1,n-1}x_{n-1} &= b_{n-1} \end{aligned} \quad (7.1.1)$$

行列とベクトルを用いると次のように書ける.

$$A = \begin{bmatrix} a_{00} & a_{01} & \cdots & a_{0,n-1} \\ a_{10} & a_{11} & \cdots & a_{1,n-1} \\ & & \ddots & \\ a_{n-1,0} & a_{n-1,1} & \cdots & a_{n-1,n-1} \end{bmatrix}, \quad X = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{bmatrix}, \quad B = \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n-1} \end{bmatrix}$$

$$AX = B$$

ここで係数の行列が正則のとき, すなわち行列式  $|A|$  が 0 でない ( $|A| \neq 0$ ) とを考える.

ここで  $A$  の 行列式 を  $|A|$  と書き, 以下のようにして定義される.

$A$  が 2 次の行列のとき,  $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$  とすると,  $|A| = ad - bc$ .

$A$  が  $n \times n$  のとき,  $A$  の  $i$  行  $j$  列を取り除いてできる  $(n-1)$  次の行列の行列式を  $D_{ij}$  と書き, 小行列式という. ただし  $i = 0, \dots, n-1$ ,  $j = 0, \dots, n-1$  とする. すなわち一番上の行を 0 行目とし, 一番下の行を  $n-1$  行目とする. 列に関しても同様である.

$A_{ij} = (-1)^{i+j} D_{ij}$  を  $A$  の余因子という.

余因子を用いて  $A$  の行列式は次のように定義される.

$$|A| = \sum_{i=0}^{n-1} a_{ij} A_{ij}$$

ここで  $j$  は  $0, 1, \dots, n-1$  のうちどれでもよい. (答えは同じになるので, 計算しやすい  $j$  を選べばよい)

この式を用いれば,  $n$  次の行列式を求めるには  $(n-1)$  次の行列式を求めればよいことになり, これを繰り返して, 最終的には 2 次の行列式を求めればよいことになり, 上記の  $A$  が 2 次の行列の場合に帰着される.

ほかの定義の方法として, 以下のものがあるが詳細は省略する.

$S_n$  を  $n$  次の置換全体 (ただし  $0, 1, \dots, n-1$  に対する置換) として,  $\text{sgn}(\sigma)$  を置換  $\sigma \in S_n$  の符号 (詳細略) とするとき,

$$|A| = \sum_{\sigma \in S_n} \text{sgn}(\sigma) x_{\sigma(0)0} x_{\sigma(1)1} \cdots x_{\sigma(n-1),n-1}$$

## 7.2 クラメールの公式

行列式を用いて上記の連立1次方程式を解くことができる。上式に対して、

$$x_i = \frac{|A_i|}{|A|}, \quad (i = 0, 1, \dots, n-1)$$

ただし、 $A_i$  は  $A$  の  $i$  列を  $B$  で置き換えたものである。これをクラメル (Cramer) の公式という。

この公式は、理論的には扱いやすく、また  $n$  が小さいときには計算にも用いることがあるが、一般には計算量が多いので数値計算には用いられない。

### 例 7.1

$$3x_0 + 4x_1 = 10$$

$$5x_0 - 2x_1 = 8$$

をクラメールの公式を用いて解く。

$$A = \begin{bmatrix} 3 & 4 \\ 5 & -2 \end{bmatrix}, X = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix}, B = \begin{bmatrix} 10 \\ 8 \end{bmatrix} \text{ とすると } AX = B \text{ である.}$$

$$A_0 = \begin{bmatrix} 10 & 4 \\ 8 & -2 \end{bmatrix}, A_1 = \begin{bmatrix} 3 & 10 \\ 5 & 8 \end{bmatrix} \text{ である. } (A_i \text{ は } A \text{ の } i \text{ 列を } B \text{ で置き換えたもの})$$

$$|A| = 3 \cdot (-2) - 4 \cdot 5 = -26 \text{ である.}$$

クラメールの公式より、

$$x_0 = \frac{|A_0|}{|A|} = \frac{10 \cdot (-2) - 4 \cdot 8}{-26} = \frac{-52}{-26} = 2$$

$$x_1 = \frac{|A_1|}{|A|} = \frac{3 \cdot 8 - 10 \cdot 5}{-26} = \frac{-26}{-26} = 1$$

となる。

一般の  $n$  変数の場合の計算量については以下のようなになる。

- $|A|$  の計算： $(n-1)$  回の掛け算 ( $\times$ ) が  $n!$  通り、 $(n!-1)$  個の項間に  $\pm$  の演算がある。したがって  $(n-1)n! + (n!-1) = n \cdot n! - 1$  回の演算を行う。
- 上記の演算が  $|A_0|, |A_1|, \dots, |A_{n-1}|, |A|$  まで全部で  $(n+1)$  回ある。
- 各  $x_i$  を求めるために  $n$  回の割り算  $x_i = \frac{|A_i|}{|A|}$  を行う。
- 全部で  $(n \cdot n! - 1)(n+1) + n = n(n+1)n! - 1$  回の演算が必要。

$n=4$  ならば 480 回くらいで済むが、 $n=10$  になると  $3.99 \times 10^8$  回の演算で、コンピュータでも困難になる。また計算回数が非常に多いため、桁落ちにより精度が著しく低下することがある。これらのことからクラメールの公式は数値計算に用いられない。



### 7.3 行列の基本変形

連立1次方程式  $AX = B$  に対して,  $\begin{bmatrix} A & B \end{bmatrix}$  を拡大係数行列という.

拡大係数行列  $\begin{bmatrix} A & B \end{bmatrix}$  に対して行列の基本変形を行い, 解を得やすい形に変形する. これが数値計算で解を求める方法の原理である.

1. ある行を定数倍 ( $\neq 0$ ) する.
2. ある行をほかの行に加える.
3. ある行の定数倍をほかの行に加える (上の 1. + 2.).
4. 2つの行を交換する.

これらの操作を加えても, 元の方程式と同じ解が得られる. この性質を利用して2つの方法 (直接法及び反復法) により連立1次方程式の解を求める.

直接法は方程式の加減乗除により変数を消去して直接, 厳密解を求めるものである.

反復法は, 適当な初期値から出発して繰り返し近似計算を行い, 収束値として解を求める方法である.

ここではそれぞれの代表的な計算法としてガウスの消去法とヤコビの反復法を扱う.

### 7.4 ガウスの消去法

直接法にはガウスの消去法, ガウス・ジョルダンの消去法, LU 分解法等があるが, ここでは代表的なものとしてガウスの消去法について述べる.

ガウスの消去法は, 1. 前進消去と 2. 後退代入の2段階からなる. また, ピボットを選択が必要になる.

以下ではそれらについて例を用いながら説明する.

#### 7.4.1 前進消去

例 7.2 次の4元連立1次方程式を解くことを考える.

$$\begin{cases} 4x_0 + 3x_1 + 2x_2 + x_3 = 20 \\ 2x_0 + 5x_1 - 3x_2 - 2x_3 = -5 \\ x_0 - 4x_1 + 8x_2 - x_3 = 13 \\ -3x_0 + 2x_1 - 4x_2 + 5x_3 = 9 \end{cases}$$

(1)  $k = 0$ : 最初の方針として第1行目以下の第0列目の係数を消去することを考える.

以下のように方程式を変形する.

$$\begin{cases} (1 \text{ 行目の方程式}) - (0 \text{ 行目の方程式}) \times 2/4 \\ (2 \text{ 行目の方程式}) - (0 \text{ 行目の方程式}) \times 1/4 \\ (3 \text{ 行目の方程式}) - (0 \text{ 行目の方程式}) \times (-3)/4 \end{cases}$$

この結果, 以下を得る.

$$\begin{cases} 4x_0 + 3x_1 + 2x_2 + x_3 = 20 \\ \frac{7}{2}x_1 - 4x_2 - \frac{5}{2}x_3 = -15 \\ -\frac{19}{4}x_1 + \frac{15}{2}x_2 - \frac{5}{4}x_3 = 8 \\ \frac{17}{4}x_1 - \frac{5}{2}x_2 + \frac{23}{4}x_3 = 24 \end{cases}$$

(2)  $k = 1$  : 次に第 2 行目以下の第 1 列目の係数を消去することを考える.

$$\begin{cases} (2 \text{ 行目の方程式}) - (1 \text{ 行目の方程式}) \times \frac{-19/4}{7/2} \\ (3 \text{ 行目の方程式}) - (1 \text{ 行目の方程式}) \times \frac{17/4}{7/2} \end{cases}$$

このように変形すると, 以下の式になる.

$$\begin{cases} 4x_0 + 3x_1 + 2x_2 + x_3 = 20 \\ \quad \quad \frac{7}{2}x_1 - 4x_2 - \frac{5}{2}x_3 = -15 \\ \quad \quad \quad \frac{29}{14}x_2 - \frac{65}{14}x_3 = -\frac{173}{14} \\ \quad \quad \quad \frac{33}{14}x_2 + \frac{123}{14}x_3 = \frac{591}{14} \end{cases}$$

(3)  $k = 2$  : 最後に第 3 行目以下の第 2 列目の係数を消去することを考える.

$$(3 \text{ 行目の方程式}) - (2 \text{ 行目の方程式}) \times \frac{33/14}{29/14}$$

これにより, 階段状の連立方程式が得られる.

$$\begin{cases} 4x_0 + 3x_1 + 2x_2 + x_3 = 20 \\ \quad \quad \frac{7}{2}x_1 - 4x_2 - \frac{5}{2}x_3 = -15 \\ \quad \quad \quad \frac{29}{14}x_2 - \frac{65}{14}x_3 = -\frac{173}{14} \\ \quad \quad \quad \frac{408}{29}x_3 = \frac{1632}{29} \end{cases} \quad (7.4.1)$$

このような変形を前進消去という.

上の例を一般化すると, 前進消去は以下のように書ける.

以下の連立方程式を解く.

$$\begin{array}{rclcl} a_{00}x_0 & + & a_{01}x_1 + \cdots + a_{0j}x_j + \cdots + a_{0,n-1}x_{n-1} & = & b_0 \\ a_{10}x_0 & + & a_{11}x_1 + \cdots + a_{1j}x_j + \cdots + a_{1,n-1}x_{n-1} & = & b_1 \\ \vdots & & \vdots & & \vdots \\ a_{i0}x_0 & + & a_{i1}x_1 + \cdots + a_{ij}x_j + \cdots + a_{i,n-1}x_{n-1} & = & b_i \\ \vdots & & \vdots & & \vdots \\ a_{n-1,0}x_0 & + & a_{n-1,1}x_1 + \cdots + a_{n-1,j}x_j + \cdots + a_{n-1,n-1}x_{n-1} & = & b_{n-1} \end{array}$$

上の連立方程式の  $a_{00}x_0$  の下をすべて 0 にするように変形する.

$$\begin{array}{rclcl} a_{00}x_0 & + & a_{01}x_1 + \cdots + a_{0j}x_j + \cdots + a_{0,n-1}x_{n-1} & = & b_0 \\ 0 & + & a_{11}x_1 + \cdots + a_{1j}x_j + \cdots + a_{1,n-1}x_{n-1} & = & b_1 \\ \vdots & & \vdots & & \vdots \\ 0 & + & a_{i1}x_1 + \cdots + a_{ij}x_j + \cdots + a_{i,n-1}x_{n-1} & = & b_i \\ \vdots & & \vdots & & \vdots \\ 0 & + & a_{n-1,1}x_1 + \cdots + a_{n-1,j}x_j + \cdots + a_{n-1,n-1}x_{n-1} & = & b_{n-1} \end{array}$$

$i$  行目をこのように変形するためには, 1 行目の両辺を  $a_{00}$  で割り  $a_{i0}$  をかけた式を  $i$  行目から引けばよい. したがって,  $i$  行目の係数  $a_{ij}$ ,  $j = 1, \dots, n-1$  および  $b_i$  は以下のようになる.

$$\begin{aligned} a_{ij} &= a_{ij} - a_{i0}a_{0j}/a_{00}, \quad j = 1, \dots, n-1 \\ b_i &= b_i - a_{i0}b_0/a_{00} \end{aligned}$$

この変形を  $i = 1, \dots, n-1$  に対して実施する.

次に  $a_{11}x_1$  の下をすべて 0 にする変形は,  $i = 2, \dots, n-1$  に対し,

$$\begin{aligned} a_{ij} &= a_{ij} - a_{i1}a_{1j}/a_{11}, \quad j = 2, \dots, n-1 \\ b_i &= b_i - a_{i1}b_1/a_{11} \end{aligned}$$

を行う.

同様に,  $a_{kk}x_k$  の下を 0 にする際には,  $i = k+1, \dots, n-1$  のそれぞれの  $i$  に対して次式を計算する.

$$\begin{aligned} a_{ij} &= a_{ij} - a_{ik}a_{kj}/a_{kk}, \quad j = k+1, \dots, n-1 \\ b_i &= b_i - a_{ik}b_k/a_{kk} \end{aligned} \quad (7.4.2)$$

これを  $k = 0, 1, \dots, n-2$  と変えて計算すれば, 次のような 3 角形の連立方程式が得られる.

$$\begin{array}{cccccc} a_{00}x_0 & + & a_{01}x_1 + \cdots + & a_{0k}x_k + \cdots + & a_{0,n-1}x_{n-1} & = & b_0 \\ & & a_{11}x_1 + \cdots + & a_{1k}x_k + \cdots + & a_{1,n-1}x_{n-1} & = & b_1 \\ & & & \vdots & \vdots & & \vdots \\ & & & a_{kk}x_k + \cdots + & a_{k,n-1}x_{n-1} & = & b_k \\ & & & & \vdots & & \vdots \\ & & & & a_{n-1,n-1}x_{n-1} & = & b_{n-1} \end{array}$$

これが前進消去である.

#### 7.4.2 後退代入

例 7.3 例 7.2 の最後の式は,

$$\left\{ \begin{array}{l} 4x_0 + \frac{3}{2}x_1 - \frac{29}{14}x_2 - \frac{408}{29}x_3 = \frac{1632}{29} \\ \frac{7}{2}x_1 - 4x_2 - \frac{65}{14}x_3 = -\frac{173}{14} \\ 4x_2 - \frac{5}{2}x_3 = -15 \\ x_3 = \frac{1632/29}{408/29} = 4 \end{array} \right.$$

である. これから後退代入は以下のようにする.

最下行の式は  $x_3$  のみの式なので,  $x_3$  を求めることができる. 次にその値を下から 2 行目に代入すると,  $x_2$  のみの式なので,  $x_2$  を求めることができる. これを繰り返して,  $x_3, x_2, x_1, x_0$  を順番に求めることができる.

$$\left\{ \begin{array}{l} x_3 = \frac{1632/29}{408/29} = 4 \\ x_2 = \frac{\frac{14}{29}(-\frac{173}{14} + \frac{65}{14}x_3)}{\frac{29}{14}} = \frac{-173/14 + (65/14) \times 4}{29/14} = \frac{87/14}{29/14} = 3 \\ x_1 = \frac{\frac{2}{7}(-15 + 4x_2 + \frac{5}{2}x_3)}{\frac{7}{2}} = \frac{-15 + 4 \times 3 + (5/2) \times 4}{7/2} = \frac{7}{7/2} = 2 \\ x_0 = \frac{\frac{1}{4}(20 - 3x_1 - 2x_2 - x_3)}{\frac{4}{4}} = \frac{20 - 3 \times 2 - 2 \times 3 - 4}{4} = \frac{4}{4} = 1 \end{array} \right.$$

一般的に記述すると以下ようになる.

$k = n-1, n-2, \dots, 0$  の順に  $x_k$  を求める.

まず,  $x_{n-1} = b_{n-1}/a_{n-1,n-1}$  である.

$$\text{一般に } k \text{ 行目は, } x_k = \left( b_k - \sum_{j=k+1}^{n-1} a_{kj}x_j \right) / a_{kk} \text{ である. } (k = n-1, n-2, \dots, 0) \quad (7.4.3)$$

$k = n - 1, n - 2, \dots, 0$  の順に  $k$  を動かすためには, C 言語のプログラムでは

```
for(k=n-1;k>=0;k--){
    .....
}
```

のように書けばよい.

**問題 7.1** (p.97, p.102, p.108, p.110 参照) 例 7.2 の連立方程式について, ガウスの消去法で解を求めるプログラムを作成せよ. 前進消去は式 (7.4.2) を用いてプログラミングせよ. 後退代入は式 (7.4.3) を用いてプログラミングせよ. プログラムは,

```
...

#define N 4
int main(){
    double A[N][N]={4,3,2,1},{2, 5, -3, -2},{1, -4, 8, -1},{-3, 2, -4, 5};
    double B[N]={20, -5, 13, 9};
    ...
}
```

のように行列のサイズ  $N$  と, 行列  $A$ , ベクトル  $B$  を定義せよ. さらに  $N$  と  $A, B$  の内容を変更すれば別のサイズの行列の計算ができるようにすること. (for 文を用いて作成すればそのようにできるはず)

解が例 7.3 で求めた解と一致することを確認せよ.

**注意:** 式 (7.4.2) において,  $a_{ij}$  ( $j = 0, 1, \dots, k$ ) は計算していない. これらの値は後退代入に不要なので計算は省略している. そのため, 前進消去のあと行列を表示すると, 下三角 (左下の三角の部分) は 0 になるとは限らないが, 気にしなくてよい. 上三角 (行列の右上の三角の部分) の値が式 (7.4.1) と同じであれば前進消去は正しくできていることがわかる. ただし, プログラムでは前進消去の結果を表示する必要はない.

よくわからなければ, p.108 と p.110 を見て, 配列の使い方を理解する.

前進消去において, for 文で動かす変数 ( $\sum$  で動かす変数と, 本文で「変数 = ?, ..., ?」のように値が変化する変数など) を把握して, 多重の for 文 (for 文の中に for 文がある) を作成する.

for 文の順番は, 例えば  $i$  の値を決めるために  $k$  が必要であれば,  $k$  の for 文は  $i$  の for 文より外側になる. 詳しくは p.109 を参照する.

$\sum$  の計算では, p.110 をよく見ること.

ここまでで動かして行列  $A$  の値を表示して, 後退代入の初めの値と一致することを確認する. (行列の左下三角の部分は 0 になるとは限らない)

そのあとで後退代入を作成する. 後退代入は式 (7.4.3) において, まずは  $\sum$  の計算を行う ( $\sum$  の作成は今までと同様). そのあとで式 (7.4.3) を計算する.

出力

X[0]=...

X[1]=...

X[2]=...

X[3]=...

(問題終)

### 7.4.3 ピボットの選択

前進消去において、 $k-1$  列目の消去が終了した状態を示したのが、以下の方程式である。

$$\begin{array}{cccccc}
a_{00}x_0 & + & a_{01}x_1 + \cdots + & a_{0k}x_k + \cdots + & a_{0,n-1}x_{n-1} & = b_0 \\
& & a_{11}x_1 + \cdots + & a_{1k}x_k + \cdots + & a_{1,n-1}x_{n-1} & = b_1 \\
& & & \vdots & \vdots & \vdots \\
& & & a_{kk}x_k + \cdots + & a_{k,n-1}x_{n-1} & = b_k \\
& & a_{k+1,k}x_k + \cdots + & a_{k+1,n-1}x_{n-1} & = b_{k+1} \\
& & & \vdots & \vdots & \\
& & a_{n-1,k}x_k + \cdots + & a_{n-1,n-1}x_{n-1} & = b_{n-1}
\end{array}$$

次に  $a_{kk}x_k$  の下を消去する際の式 (7.4.2) において、 $a_{kk}$  が非常に小さいと式の第 2 項が大きくなり桁落ちが生じる可能性がある。また  $a_{kk} = 0$  だと計算できない。そこで、 $k$  列目の係数  $a_{ik}$  の中で絶対値が一番大きな行  $p$  と、今ある  $k$  行目を交換して、計算をする工夫が必要となる。すなわち

$$|a_{pk}| = \max_{i=k,k+1,\dots,n-1} |a_{ik}| \quad (7.4.4)$$

となる  $p$  を探して、係数  $\{a_{kj} : j = k, k+1, \dots, n-1\}$  と  $\{a_{pj} : j = k, k+1, \dots, n-1\}$  を交換し、 $b_k$  と  $b_p$  を交換した後、前進消去の計算を続行すればよい。この  $p$  のことをピボットという。(正確にはこれは部分ピボットという。完全ピボットというものもあるが、ここでは省略する)

**追加問題 7.1** 次のプログラムを作成せよ。

1. ガウスの消去法のプログラムに、ピボットの部分を組み込んだプログラムを作成せよ。

前進消去の式 (7.4.2) を行う直前に、ピボット  $p$  の選択を式 (7.4.4) で行い、 $\{a_{kj} : j = k, k+1, \dots, n-1\}$  と  $\{a_{pj} : j = k, k+1, \dots, n-1\}$  を交換し、 $b_k$  と  $b_p$  を交換し、そのあとで式 (7.4.2) を行えば良い。

2. 次の連立方程式をピボットの部分を組み込んだ場合と組み込まない場合について解け。

$$\begin{cases} 2x_0 + 4x_1 - 2x_2 + x_3 = 8 \\ x_0 + 2x_1 + x_2 + x_3 = 12 \\ x_0 + 3x_1 + 2x_2 + x_3 = 17 \\ x_0 + x_1 + x_2 + x_3 = 10 \end{cases}$$

解が正しいかを手計算などでチェックせよ。ピボットの部分を組み込まない場合にうまく計算できない理由を調べよ。

3. 上の 2. のプログラムの変数を `float` にせよ. ただし, `fpu_control.h` は使わない. そのうえで次の連立方程式を解き, ピボットの部分がある場合とない場合の解の精度を比較せよ.

$$\begin{cases} 2x_0 + 4x_1 - 2x_2 + x_3 = 8 \\ x_0 + 2.001x_1 + 1.002x_2 + 1.001x_3 = 12.012 \\ x_0 + 3x_1 + 2x_2 + x_3 = 17 \\ x_0 + x_1 + x_2 + x_3 = 10 \end{cases}$$

(追加問題終)

#### 7.4.4 ガウスの消去法での計算量

前進消去での計算量は以下ようになる.

1. 以下で考えるインデックスについて  $k$ : 注目行,  $i (= k+1, \dots, n-1)$ : パラメータ行とする.

$$a_{ij}^{\ell+1} = a_{ij}^{\ell} - (a_{ik}^{\ell}/a_{kk}^{\ell}) \times a_{kj}^{\ell} \quad (\ell \text{ は反復回数})$$

2. 係数比  $a_{ik}^{\ell}/a_{kk}^{\ell}$  を求めるのに 1 回割り算を行っている.  
 3. 列方向に上の操作を  $j = k+1, \dots, n$  まで  $n-k$  回繰り返す ( $n$  列目は, 右辺の定数  $b_i$  の列とする). このそれぞれで掛け算と引き算を行う. ここまでで  $2(n-k)+1$  回の演算が必要である.  
 4. 以上を行方向に  $i = k+1, \dots, n-1$  まで  $(n-k-1)$  行に対し行う. ここまでで  $(2(n-k)+1)(n-k-1)$  回の演算となる.  
 5. さらに注目行  $k$  は,  $k = 0, \dots, n-2$  まで変化するので, 前進消去での全計算量は次のようになる.

$$\sum_{k=0}^{n-2} (2(n-k)+1)(n-k-1) = \frac{n(n-1)(4n+7)}{6}$$

6.  $n = 10$  のときは, ここまでで 705 回の演算になる.

後退代入での計算量は以下ようになる.

1. 一番下の  $n-1$  行目から上に向かって考える.  
 2.  $a_{n-1,n-1}x_{n-1} = b_{n-1} \Rightarrow x_{n-1} = b_{n-1}/a_{n-1,n-1}$  と計算するのでこれには 1 回の割り算  
 3.  $a_{n-2,n-2}x_{n-2} + a_{n-2,n-1}x_{n-1} = b_{n-2} \Rightarrow x_{n-2} = (b_{n-2} - a_{n-2,n-1}x_{n-1})/a_{n-2,n-2}$  という計算で  $\times, -, \div$  各 1 回ずつで計 3 回の演算となる.  
 4. 同様に  $x_{n-3} = (b_{n-3} - a_{n-3,n-1}x_{n-1} - a_{n-3,n-2}x_{n-2})/a_{n-3,n-3}$  という計算で  $\times, -$  各 2 回ずつと 1 回の  $\div$  で, 計  $2 \times 2 + 1 = 5$  回の演算を行う.  
 5. 一般に  $i$  行について  $(n-i-1)$  回の  $\times, -$  と 1 回の割り算を行う. 以上より

$$\sum_{i=0}^{n-1} (2(n-i-1)+1) = n^2 \text{ 回の演算が必要である.}$$

6.  $n = 10$  の場合はここだけで 100 回の演算となる.

ゆえに, 前進消去・後退代入合計で,  $n = 10$  の場合 805 回の演算が必要がある. クラメールの計算方法に比べてはるかに計算量が少ない.

## 7.5 ヤコビの反復法

反復法は、解の近似値を初期値として、反復計算を必要な精度まで繰り返す方法である。反復法にはヤコビ法、ガウス・ザイデル法がある。

次の連立 1 次方程式を解くことを考える。

$$\begin{cases} 9x_0 + 2x_1 + x_2 + x_3 = 20 \\ 2x_0 + 8x_1 - 2x_2 + x_3 = 16 \\ -x_0 - 2x_1 + 7x_2 - 2x_3 = 8 \\ x_0 - x_1 - 2x_2 + 6x_3 = 17 \end{cases}$$

各式を仮に  $x_i$  について解くと、以下のようになる。

$$\begin{cases} x_0 = (20 - 2x_1 - x_2 - x_3)/9 \\ x_1 = (16 - 2x_0 + 2x_2 - x_3)/8 \\ x_2 = (8 + x_0 + 2x_1 + 2x_3)/7 \\ x_3 = (17 - x_0 + x_1 + 2x_2)/6 \end{cases}$$

この操作を反復する。 $(k)$  は  $k$  回目の繰り返しを示す。

$$\begin{cases} x_0^{(k+1)} = (20 - 2x_1^{(k)} - x_2^{(k)} - x_3^{(k)})/9 \\ x_1^{(k+1)} = (16 - 2x_0^{(k)} + 2x_2^{(k)} - x_3^{(k)})/8 \\ x_2^{(k+1)} = (8 + x_0^{(k)} + 2x_1^{(k)} + 2x_3^{(k)})/7 \\ x_3^{(k+1)} = (17 - x_0^{(k)} + x_1^{(k)} + 2x_2^{(k)})/6 \end{cases}$$

これを繰り返すと、 $x_i^{(k+1)}$  と  $x_i^{(k)}$  の差がほとんどなくなり収束する。これがヤコビの反復法である。（場合によっては収束しないこともあるが、それについては省略する。）

反復法により連立方程式を解く方法は、消去法に比べて考え方が単純であり、プログラムも簡単に書ける。しかし、解の精度や収束までの反復回数は、初期値の与え方や方程式の並び方に大きく依存する。

ヤコビの反復法を一般的に記述すると、次のようになる。

式 (7.1.1) の各式を上から順に  $x_0, \dots, x_{n-1}$  について解き、左辺の  $x_i$  には上付き添え字  $(k+1)$  を、また右辺の  $x_j$  ( $j \neq i$ ) には  $(k)$  を付すことにすれば、次式が得られる。 $k = 0, 1, \dots$ , とする。

$$\begin{aligned} x_0^{(k+1)} &= \frac{1}{a_{00}} \left( b_0 - a_{01}x_1^{(k)} - \dots - a_{0,n-1}x_{n-1}^{(k)} \right) \\ x_1^{(k+1)} &= \frac{1}{a_{11}} \left( b_1 - a_{10}x_0^{(k)} - a_{12}x_2^{(k)} - \dots - a_{1,n-1}x_{n-1}^{(k)} \right) \\ &\vdots \\ x_i^{(k+1)} &= \frac{1}{a_{ii}} \left( b_i - \sum_{j=0}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^{n-1} a_{ij}x_j^{(k)} \right) \\ x_{n-1}^{(k+1)} &= \frac{1}{a_{n-1,n-1}} \left( b_{n-1} - a_{n-1,0}x_0^{(k)} - \dots - a_{n-1,n-2}x_{n-2}^{(k)} \right) \end{aligned} \tag{7.5.1}$$

ただし  $a_{ii} \neq 0$  とする。上式を用いてヤコビの反復法の手順は以下のようになる。

1. 計算精度  $\varepsilon$ , および初期値  $\{x_i^{(0)}, i = 0, 1, \dots, n-1\}$  を定め, 繰り返し回数  $k = 0$  とする. 通常, 初期値はすべて 0 とする.
2. 式 (7.5.1) より  $i = 0, \dots, n-1$  に対して,  $x_i^{(k+1)}$  を計算する.
3. もし,  $\sum_{i=0}^{n-1} |x_i^{(k+1)} - x_i^{(k)}| < \varepsilon$  であれば, 解を  $x_i = x_i^{(k+1)}$  として計算終了する.  
そうでなければ,  $k = k + 1$  として 2. に戻る.

ただし, このままでは  $k = 0, 1, 2, 3, \dots$  と  $k$  が増えるにしたがって  $x_i^{(0)}, x_i^{(1)}, x_i^{(2)}, x_i^{(3)}, \dots$  が必要になり, 使うメモリが増えていくので,  $x_i^{(0)}$  と  $x_i^{(1)}$  のみを用いるようにする.

式 (7.5.1) も,  $x_i^{(0)}$  と  $x_i^{(1)}$  を用いるように書き換えると次式となる.

$$x_i^{(1)} = \frac{1}{a_{ii}} \left( b_i - \sum_{j=0}^{i-1} a_{ij} x_j^{(0)} - \sum_{j=i+1}^{n-1} a_{ij} x_j^{(0)} \right) \quad (7.5.2)$$

すると, アルゴリズムは以下ようになる.

ヤコビ法のアルゴリズム

1. 計算精度  $\varepsilon > 0$ , および初期値  $\{x_i^{(0)}, i = 0, 1, \dots, n-1\}$  を定め, 繰り返し回数  $k = 0$  とする. 通常, 初期値はすべて 0 とする.
2. 式 (7.5.2) より  $i = 0, \dots, n-1$  に対して,  $x_i^{(1)}$  を計算する.
3. もし,  $\sum_{i=0}^{n-1} |x_i^{(1)} - x_i^{(0)}| < \varepsilon$  であれば, 解を  $x_i = x_i^{(1)}$  ( $i = 0, 1, \dots, n-1$ ) として計算終了する.  
そうでなければ,  $k = k + 1$  として,  $i = 0, \dots, n-1$  に対して  $x_i^{(0)} = x_i^{(1)}$  として, 2. に戻る.

このアルゴリズムが収束するためにはある条件が必要であるが, 詳細は省略する. 興味があれば, はじめに紹介した参考書などを見よ.

## 7.6 ガウス・ザイデルの反復法

ガウス・ザイデルの反復法は, ヤコビの反復法の改良版であり,  $k+1$  回目で  $m$  行より上ですでに求まっている  $x_0^{(k+1)}, x_1^{(k+1)}, \dots, x_{m-1}^{(k+1)}$  を用いて計算する方法であり, ヤコビの方法より 2 倍程度の速さで収束することが知られている.

$$\begin{cases} x_0^{(k+1)} &= (20 & & - & 2x_1^{(k)} & - & x_2^{(k)} & - & x_3^{(k)})/9 \\ x_1^{(k+1)} &= (16 & - & 2x_0^{(k+1)} & & + & 2x_2^{(k)} & - & x_3^{(k)})/8 \\ x_2^{(k+1)} &= (8 & + & x_0^{(k+1)} & + & 2x_1^{(k+1)} & & + & 2x_3^{(k)})/7 \\ x_3^{(k+1)} &= (17 & - & x_0^{(k+1)} & + & x_1^{(k+1)} & + & 2x_2^{(k+1)} & )/6 \end{cases}$$

ガウス・ザイデルの反復法を一般的に記述すると, 以下のようになる. ただし  $i = 0, 1, \dots, n-1$ .

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left( b_i - \sum_{j=0}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n-1} a_{ij} x_j^{(k)} \right) \quad (7.6.1)$$



アルゴリズムは、第 7.5 節の枠で囲んだヤコビ法のアルゴリズムの式 (7.5.1) を式 (7.6.1) に変更すればよい。また、式 (7.6.1) の計算の際には  $x_i^{(1)}$  を用いるので、この初期化（通常はすべて 0 に設定）も必要となることに注意すること。

式 (7.6.1) を  $x_i^{(0)}$  と  $x_i^{(1)}$  のみにすると以下ようになる。

$$x_i^{(1)} = \frac{1}{a_{ii}} \left( b_i - \sum_{j=0}^{i-1} a_{ij}x_j^{(1)} - \sum_{j=i+1}^{n-1} a_{ij}x_j^{(0)} \right) \quad (7.6.2)$$

ガウス・ザイデル法が収束するための条件については省略する。興味があれば数値計算の代表的な教科書を参照せよ。

ガウス・ザイデル法のアルゴリズム

1. 計算精度  $\varepsilon$ 、および  $\{x_i^{(0)}, i = 0, 1, \dots, n-1\}$  と  $\{x_i^{(1)}, i = 0, 1, \dots, n-1\}$  の初期値を定め、繰り返し回数  $k = 0$  とする。通常、初期値はすべて 0 とする。
2. 式 (7.6.2) より  $i = 0, \dots, n-1$  に対して、 $x_i^{(1)}$  を計算する。
3. もし、 $\sum_{i=0}^{n-1} |x_i^{(1)} - x_i^{(0)}| < \varepsilon$  であれば、解を  $x_i = x_i^{(1)}$  ( $i = 0, 1, \dots, n-1$ ) として計算終了する。そうでなければ、 $k = k + 1$  として、 $i = 0, \dots, n-1$  に対して  $x_i^{(0)} = x_i^{(1)}$  として、2. に戻る。

**問題 7.2** この問題はニュートン法などのプログラムをもとにしている部分があるので、ニュートン法などが合格してから行うこと。

(p.97, p.102, p.108, p.110 参照) 例 7.2 の連立方程式について

1. ヤコビの反復法により解を求めよ。解を求めるのに要した反復回数を表示せよ。式 (7.5.2) を用いよ。求めた解を例 7.3 の解と比較せよ。第 7.5 節の一番最後の枠で囲ったアルゴリズムを用いよ。

```
...
#define N 4
...
int main(){
    double X[N], X1[N];
    ....
```

と変数を定義し、配列  $X[N]$  には  $x_i^{(0)}$  ( $i = 0, \dots, n-1$ ) を格納し、配列  $X1[N]$  には  $x_i^{(1)}$  ( $i = 0, \dots, n-1$ ) を格納せよ。

作るのが難しいようであれば、次の順序で作成せよ。

まずはヤコビ法について、ガウスの消去法のヒントと同様にして、多重の for 文を作成して、 $\sum$  の計算を行う。p.109 を参照する。

初期化の位置に注意すること。 $\sum$  を行う for 文の直前で初期化することを推奨する。

終了条件については、まずはじめに、アルゴリズムの「もし、 $\sum_{i=0}^{n-1} |x_i^{(1)} - x_i^{(0)}| < \varepsilon$  であれば、

解を  $x_i = x_i^{(1)}$  として計算終了する。」の部分で「もし、 $k$  がある数以上（十分大きい数）になったら終了する」と変更して作成し、正しく解が求まることを確認せよ。それから「もし、

$\sum_{i=0}^{n-1} |x_i^{(1)} - x_i^{(0)}| < \varepsilon$  であれば、解を  $x_i = x_i^{(1)}$  として計算終了する。」の部分を作成せよ。

アルゴリズムの 2 が終わってからアルゴリズムの 3 を行うこと。それらを混ぜてはいけない。

$\sum |x_i^{(1)} - x_i^{(0)}|$  の計算が終了してから  $\sum |x_i^{(1)} - x_i^{(0)}| < \varepsilon$  の判定を行う。 $\sum$  の計算をしている途中で判定をしてはならない。

ヤコビ法ができれば、ガウス・ザイデル法は一か所変更すればできるはず。

出力

```
Jacobi's method
X[0]=...
X[1]=...
X[2]=...
X[3]=...
iteration=...
```

2. ガウス・ザイデルの反復法で解を求めよ。解を求めるのに要した反復回数を表示し、ヤコビの反復法と比較せよ。式 (7.6.2) を用いよ。求めた解を例 7.3 の解と比較せよ。

出力

```
Gauss-Seidel method
X[0]=...
X[1]=...
X[2]=...
X[3]=...
iteration=...
```

(問題終)

## 8 常微分方程式

理工学の分野の現象の多くは、微分方程式であらわされる。微分方程式には常微分方程式と偏微分方程式があるが、ここでは常微分方程式を扱う。

微分方程式を解析的に（すなわち厳密な数式として）解くことができる場合はそれほど多くないので、多くの場合には数値的に解くことになる。ここでは数値的に解く方法を扱う。

### 8.1 常微分方程式とその数値解

2変数関数  $f(t, x)$  に関する次の形の方程式を1階常微分方程式という。

$$\frac{dx(t)}{dt} = f(t, x(t)) \quad (8.1.1)$$

この方程式の解を求めるためには  $t = t_0$  における  $x$  の値  $x(t_0) = x_0$  が必要である。

例えば、 $f(t, x)$  が  $x$  によらず  $f(t)$  と書けるとき、

$$\frac{dx(t)}{dt} = f(t) \quad (8.1.2)$$

をみたす  $x(t)$  に対し、 $x^*(t) = c + x(t)$  ( $c$  は定数) とすると、 $x^*(t)$  も式 (8.1.2) を満たし、解が一意に決定できない。

このため条件として  $t = t_0$  における  $x$  の値  $x(t_0) = x_0$  が必要となる。これを初期条件といい、与えられた初期条件のもとで微分方程式の解を求める問題を初期値問題という。

$$\frac{dx(t)}{dt} = f(t, x(t)), \quad x(t_0) = x_0 \quad (8.1.3)$$

ここで式 (8.1.3) の右辺  $f(t, x(t))$  が  $x(t)$  を含まない場合には、

$$\frac{dx(t)}{dt} = f(t), \quad x(t_0) = x_0$$

となり、上式を積分することにより  $x(t)$  が以下のように得られる。

$$x(t) = x(t_0) + \int_{t_0}^t f(t) dt$$

この場合には前章の数値積分法を用いることで微分方程式の解が得られる。しかし、式 (8.1.3) の  $f(t, x(t))$  が  $x(t)$  を含む場合には、単なる数値積分の問題として扱うことはできない。

微分方程式を数値的に解く場合には、有限個の  $t_1, t_2, \dots, t_N$  に対する  $x(t_1), x(t_2), \dots, x(t_N)$  を求める。ここでは  $t_1, t_2, \dots, t_N$  は等間隔とし、その間隔を刻み幅という。

すなわち、刻み幅を  $h$  とすると、時刻は以下ようになる。

$$t_n = t_0 + nh, \quad n = 1, \dots, N$$

## 8.2 1 階常微分方程式

**例 8.1** ある生物の集団を考える．時刻  $t$  の単位時間における集団内の個体数の増加は，その時点の個体数  $x(t)$  に比例し，増加率は時間  $t$  と  $x(t)$  の関数  $g(t, x(t))$  で与えられるものとする．ここでは  $g(t, x) = 1 - x$  とし，初期値  $x(0) = x_0$  とする．

$[t, t + \Delta t]$  間の個体の増加数  $\Delta x(t) = x(t + \Delta t) - x(t)$  は  $g(t, x(t))x(t)\Delta t + \text{微小項}$  であらわされる．（微小項は  $o(\Delta t)$  とかける． $o(\Delta t)$  とは  $\lim_{\Delta t \rightarrow 0} \frac{\text{微小項}}{\Delta t} = 0$  となることを意味する．）

$\Delta t \rightarrow 0$  の極限を考えると，以下の微分方程式が得られる．

$$\begin{aligned}\frac{dx(t)}{dt} &= g(t, x(t))x(t) \\ &= (1 - x(t))x(t) \\ \text{初期条件: } t = 0 \text{ のとき } x &= x_0\end{aligned}\tag{8.2.1}$$

式 (8.1.3) において  $f(t, x(t)) = (1 - x(t))x(t)$  とすれば，式 (8.2.1) は，式 (8.1.3) の形の微分方程式になっている．すなわち，

$$\frac{dx(t)}{dt} = (1 - x(t))x(t), \quad x(0) = x_0\tag{8.2.2}$$

という微分方程式が得られる．

この微分方程式の解は解析的に解くことができ，

$$x(t) = \frac{1}{1 + \left(\frac{1}{x_0} - 1\right)e^{-t}}\tag{8.2.3}$$

となる．この曲線はロジスティック曲線と呼ばれる．

この例の場合には，数値計算の解と解析的に求めた解を比較するためにあえて解析的に解を求めることができる微分方程式を扱う．しかし一般には解析的に解を求められないことが多く，その場合には，以下に述べるような数値計算により解を求めることになる．

### 8.2.1 オイラー (Euler) 法

微分方程式を数値的に解く一番簡単な方法は，以下に述べるオイラー法である．オイラー・コーシー (Euler-Cauchy) 法ともいう．この方法は解の精度や安定性に問題があるので，このままで用いられることは少ないが，様々な手法の基本となるので重要である．

式 (8.1.3) の解  $x(t)$  が微分可能な関数である場合には，テーラー展開により次式が成り立つ．

$$x(t + h) = x(t) + h \frac{dx(t)}{dt} + \frac{h^2}{2} \frac{d^2x(t)}{dt^2} + \dots$$

$h$  が小さいと仮定して，この式の右辺の第 3 項以下を無視すると，以下の近似式が得られる．

$$x(t + h) \doteq x(t) + h \frac{dx(t)}{dt}$$

ここに式 (8.1.3) を代入すると以下になる.

$$x(t+h) \doteq x(t) + hf(t, x(t))$$

$t = t_n$  とすると  $t_n + h = t_{n+1}$  なので, 以下が得られる.

$$x(t_{n+1}) \doteq x(t_n) + hf(t_n, x(t_n))$$

これより  $x(t_n)$  の近似値  $x_n$  を以下のように求めるのがオイラー法である.

$$x_{n+1} = x_n + hf(t_n, x_n), \quad n = 0, 1, \dots, N \quad (8.2.4)$$

オイラー法は簡単であるが, 解が不安定 ( $n$  が小さいときの誤差が,  $n$  の増加とともに拡大されていくこと) という問題があり, 改良法が提案されている.

### 8.2.2 ルンゲ・クッタ (Runge-Kutta) 法

さらにオイラー法を改良したものにルンゲ・クッタ法がある. この方法は有名でよく用いられている. ルンゲ・クッタ法は以下のような式である.

$$\begin{aligned} x_{n+1} &= x_n + \frac{k_1 + 2k_2 + 2k_3 + k_4}{6} \\ \text{ただし, } t_n &= t_0 + nh, \quad n = 0, 1, \dots, N \\ k_1 &= hf(t_n, x_n) \\ k_2 &= hf\left(t_n + \frac{h}{2}, x_n + \frac{k_1}{2}\right) \\ k_3 &= hf\left(t_n + \frac{h}{2}, x_n + \frac{k_2}{2}\right) \\ k_4 &= hf(t_n + h, x_n + k_3) \end{aligned} \quad (8.2.5)$$

ルンゲ・クッタ法について説明する. 図 8.1 において, 現在時刻が  $t_n$  として,  $x_n$  が求められているとする.  $t_{n+1}$  における  $x_{n+1}$  を求める.

1.  $(t_n, x_n)$  (点 A) における傾きは  $f(t_n, x_n)$  であり, オイラー法によって  $x_{n+1}$  を求めると点 B になり, この際の増分が  $k_1$  になる.
2. 点 A と点 B の中間点 (点 C) の座標は  $(t_n + h/2, x_n + k_1/2)$  となる. 点 C での傾きは,  $f(t_n + h/2, x_n + k_1/2)$  となる (点 C での矢印). このように傾きを 1. から補正する.  
この傾きで点 A から直線を引き,  $x_{n+1}$  を求めると点 D になる. すなわち直線 AD は点 C での矢印と平行である. この際の増分が  $k_2$  である.
3. 点 A と点 D の中間点 (点 E) の座標は  $(t_n + h/2, x_n + k_2/2)$  となる. 点 E での傾きは,  $f(t_n + h/2, x_n + k_2/2)$  となる (点 E での矢印). このように傾きを 2. から補正する.  
この傾きで点 A から直線を引き,  $x_{n+1}$  を求めると点 F になる. すなわち直線 AF は点 E での矢印と平行である. この際の増分が  $k_3$  である.

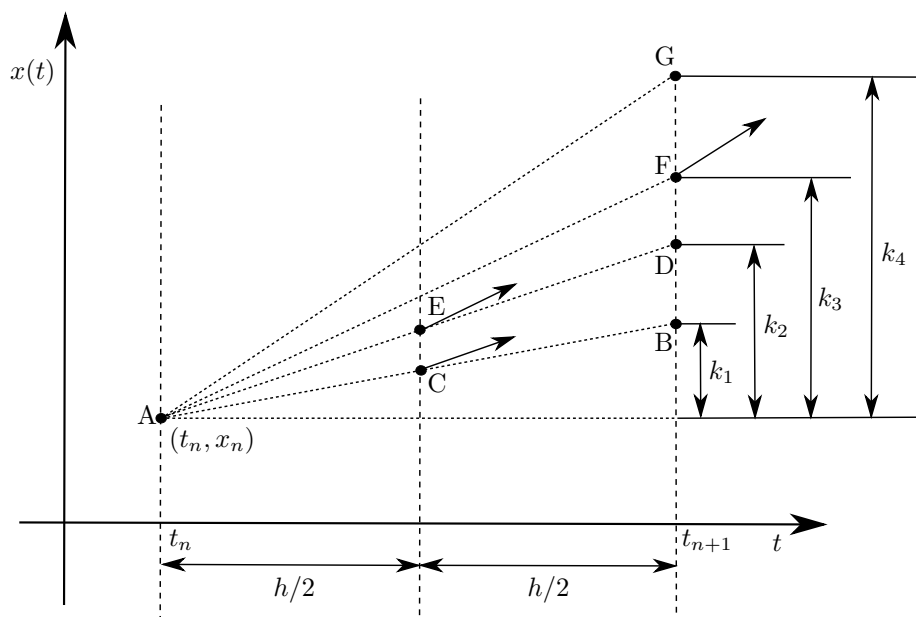


図 8.1 ルンゲ・クッタ法

4. 点 F の座標は  $(t_n + h, x_n + k_3)$  であるので、点 F での傾きは  $f(t_n + h, x_n + k_3)$  である（点 F での矢印）。このように傾きを 3. から補正する。

この傾きで点 A から直線を引き、 $x_{n+1}$  を求めると点 G になる。すなわち直線 AG は点 F での矢印と平行である。この際の増分が  $k_4$  である。

5.  $x_{n+1}$  の候補として点 B, 点 D, 点 F, 点 G が得られたが、これらを 1 : 2 : 2 : 1 の重みで平均する。すなわち  $\frac{k_1 + 2k_2 + 2k_3 + k_4}{6}$  を求める。分母の 6 は  $1 + 2 + 2 + 1$  により求められる。このように重みを決める理由は、誤差を計算したときに  $h^0, h^1, h^2, h^3, h^4$  の項が消えて  $h^5$  以上の高次の微小項にするためであるが、詳細は省略する。

このようにしてルンゲ・クッタ法が得られる。

**問題 8.1** (p.97, p.98, p.104, p.111 参照) 例 8.1 の微分方程式 (8.2.2) に対し、 $x_0$  を  $0.01 \leq x_0 \leq 0.1$  とするように設定し、オイラー法、ルンゲ・クッタ法で計算し、解析的解（ロジスティック曲線：式 (8.2.3)）と比較し、それぞれの方法の精度を検証せよ。式 (8.2.3) には  $e^{-t}$  があるが、C 言語では、 $e^x$  は `exp(x)` となる。この関数を使うには、`#include <math.h>` が必要である。後述するように OpenOffice calc などでは  $e^x$  を計算することも可能である。

配列は使わずに作ること。（配列を使うと、メモリの使用量が増え、非常に小さな  $h$  の扱いが難しくなる。）

表示では、1 行に「時刻」、「その時刻のオイラー法の  $x$ 」、「その時刻のルンゲ・クッタ法の  $x$ 」の 3 つを表示せよ。もしそのような表示が難しければ、各計算法で 1 行に「時刻」、「その時刻のその手法の  $x$ 」を表示して、それを 2 つ（計算法は 2 つ）作成し、コンソールを並べて表示せよ。

よくわからない場合については一番最後を見ること。

時刻は  $t = 0$  から開始し、 $t = 10$  くらいまで表示せよ。時間の刻み幅  $h$  は 0.5 程度とせよ。（実際にシミュレーションするときにはもう少し小さい  $h$  を使うことが多いが、ここでは各手法の差をわかりやすくするために、このように設定する。）プログラムでは  $f(x)$  を計算する関数 `f(double x)` を用いよ。一般には 2 変数関数  $f(t, x)$  を計算する関数が必要だが、この問題の場合には変数は  $x$  のみで良い。関数については、問題 6.1, 4.1 を参照せよ。

計算結果はファイルに記録せよ。データは 1 行に一時刻のデータをカンマ (,) で区切って記入せよ。例えば、

時刻 1 (数値のみ), 時刻 1 のオイラー法の  $x$  の値 (数値のみ)  
時刻 2 (数値のみ), 時刻 2 のオイラー法の  $x$  の値 (数値のみ)  
時刻 3 (数値のみ), 時刻 3 のオイラー法の  $x$  の値 (数値のみ)  
...

というファイル、

時刻 1 (数値のみ), 時刻 1 のルンゲ・クッタ法の  $x$  の値 (数値のみ)  
時刻 2 (数値のみ), 時刻 2 のルンゲ・クッタ法の  $x$  の値 (数値のみ)  
時刻 3 (数値のみ), 時刻 3 のルンゲ・クッタ法の  $x$  の値 (数値のみ)  
...

というファイルの 2 つを作成せよ。ファイル名の拡張子は csv とせよ。

もしくは、

時刻 1, オイラー法の  $x$  の値, ルンゲ・クッタ法の  $x$  の値  
時刻 2, オイラー法の  $x$  の値, ルンゲ・クッタ法の  $x$  の値  
時刻 3, オイラー法の  $x$  の値, ルンゲ・クッタ法の  $x$  の値  
...

というような形式の 1 つのファイルを作成せよ。ファイル名の拡張子は csv とせよ。

csv ファイルのアイコンをダブルクリックして、ファイルを OpenOffice calc(Linux) で開く。開くときに「区切りのオプション」で、「カンマ」のみを指定する。OpenOffice calc(Linux) ではなく、Excel(Windows) を用いても良い。

さらに、横軸を時間、縦軸を  $x$  としたグラフを作成せよ。

グラフは、散布図 (Scatter Plot, XY(Scatter) 等) で、各点を線で結ぶ設定にせよ。

グラフが作れない場合、数値が文字列として認識されている場合がある。標準では右詰めになっていれば数値で、左詰めになっていれば文字列なので、それにより確認できる。文字列になっている場合には、C 言語で csv ファイルを作る際に、以下の 2 点に注意する。ファイルには `%f` を用いて書き込み、数値の間は、カンマのみで、スペースは入れないこと。

また、csv ファイルを作り直さずに、OpenOffice calc で文字列を数値に変換することもできる。

OpenOffice calc の関数として VALUE があるので、空いたセルに「=VALUE(B1)」などとする（B1 はもとのデータのあるセル番号。適宜変更すること）と、B1 のセルを数値に変換したものが得られる。そのセルの右下をドラッグすれば、B1 の列を数値に変換したものが得られる。これを繰り返して、すべて数値に変換する。

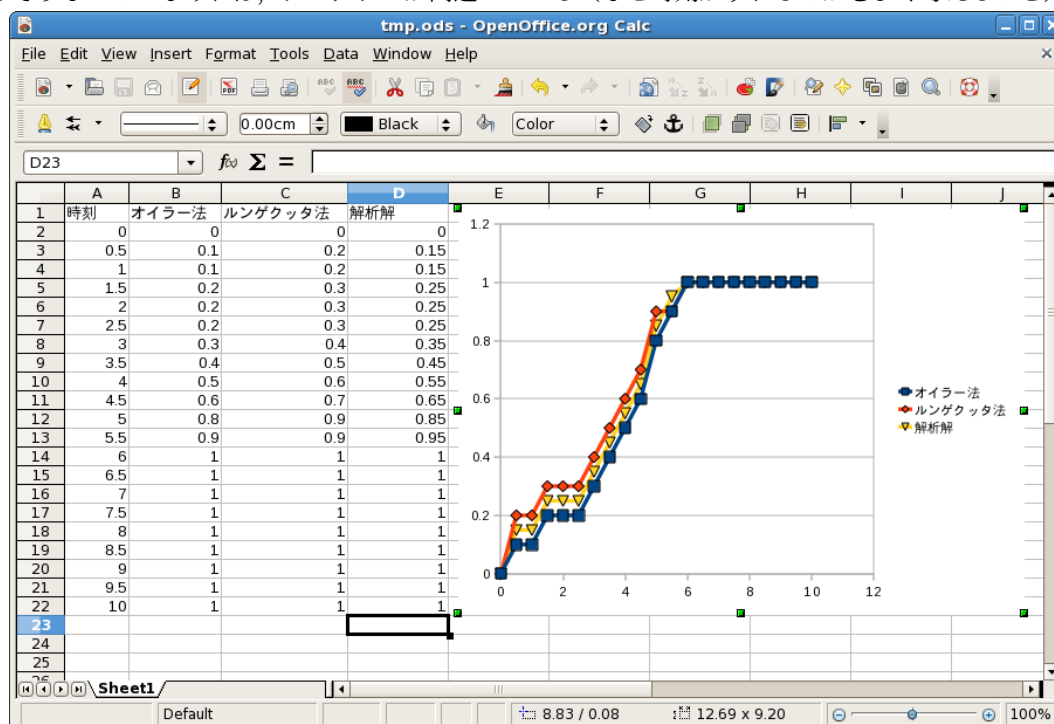
複数の csv ファイルを作成した場合には、それをすべて読み込み 1 つのワークシートにデータをまとめてからグラフを作成すること。

さらに、解析解を C 言語で同様に求めて、csv ファイルを作成して、グラフを作成して、オイラー法、ルンゲ・クッタ法のグラフと比較せよ。比較しやすいようなグラフを作成せよ。

解析解は、Excel や OpenOffice で作るのでも良い。エクセル、OpenOffice calc のどちらの場合でも、EXP(x) で  $e^x$  が求められるので、この関数を用いて解析解のグラフを作成せよ。

OpenOffice calc を使用した場合、左端の数字のところを右クリックして、Insert Rows として、1 行挿入して、「オイラー法」などの文字を入れること。それらの文字も含めてグラフを作成すると、以下のようなものができる。ただし、数字は適当なので参考にしないこと。

時刻が 0 のときのオイラー法などの値は  $x_0$  になっているはずである（手動で入力したのではだめ）。  
もしそうになっていなければ、プログラムが間違っている（なぜ時刻がずれるのかをよく考えること）



ここまでうまくできたら、h の値を小さくして、3 つの方法の答えがほぼ一致することを確認めて、グラフを作成せよ。

提出ファイルは以下の通り。

- グラフとデータ（ $t=0$  付近が見えるもの）とソースファイルの画面キャプチャ
- ソースファイル
- csv ファイル（C 言語で作成したもの。OpenOffice で上書きしないこと）
- excel ファイル（OpenOffice で「名前をつけて保存」→ファイルの種類を「Microsoft Excel



97/2000/XP (.xls) で保存。「Excel ブック (.xlsx)」「Excel 97/2003 ブック (.xls)」などでも良い。いずれにせよ拡張子を .xls または .xlsx にすること

さらに  $h$  を小さくして (どのように小さくすればよいかは結果を見て考えること), 上記の 4 つを添付すること。

すなわち全部で 8 つのファイルを添付すること。

ただし, オイラー法とルンゲクッタ法とでソースファイルを分けたり, 複数の csv ファイルを作る場合には, それに応じて添付するファイルを増やすこと。

よくわからない場合には, まずはオイラー法について行う。関数としては  $f$  と, 解析解 (関数名 kaisaikikai など) の 2 つを作成する。ニュートン法を参考にしてオイラー法を計算するプログラムを作る。初めは for 文の中で printf で, オイラー法と解析解の途中経過を表示して, 値が近いことを確認する (一致はしないが近い値になるはず)。次に p.111 を見て fprintf で結果をファイルにセーブする。あとは問題文をよく読んでグラフにする。

(問題終)

### 8.3 連立 1 階常微分方程式

例 8.2 前節での例 8.1 では, 単一種の生物の増殖プロセスのモデルを考えたが, 捕食関係にある 2 種類の生物の増殖問題は, 連立微分方程式で表される。

クジラ  $x$  はイワシ  $y$  を捕食して増殖する。クジラの出生率は餌であるイワシの数に影響され,  $(\lambda y - a)$  で表されると仮定する。これより 1 年間のクジラの増加数は  $(\lambda y - a)x$  で表される。

一方, イワシは単位の母体あたり出生率  $b$ , 捕食率  $-\mu x$  で変化するので, 増加率は  $(b - \mu x)y$  で表される。

これより次の連立微分方程式が得られる。このモデルはロトカ・ヴォルテラ系と呼ばれる。モデルの意味より,  $x \geq 0, y \geq 0, a > 0, b > 0, \lambda > 0, \mu > 0$  となる。

$$\begin{aligned}\frac{dx(t)}{dt} &= (\lambda y(t) - a)x(t) \\ \frac{dy(t)}{dt} &= (b - \mu x(t))y(t) \\ \text{初期条件: } x(t_0) &= x_0, \quad y(t_0) = y_0\end{aligned}$$

このような連立微分方程式を扱う。もう少し一般的に, 次の連立 1 階常微分方程式を考える。

$$\frac{dx_i(t)}{dt} = f_i(t, x_1(t), x_2(t), \dots, x_m(t)) \quad (8.3.1)$$

$$\text{初期条件: } t = t_0 \text{ のとき } x_i(t_0) = x_i^{(0)}, \quad i = 1, 2, \dots, m \quad (8.3.2)$$

$$X(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \\ \vdots \\ x_m(t) \end{bmatrix}, \quad X_0 = \begin{bmatrix} x_1^{(0)} \\ x_2^{(0)} \\ \vdots \\ x_m^{(0)} \end{bmatrix}, \quad F(t, X(t)) = \begin{bmatrix} f_1(t, x_1(t), \dots, x_m(t)) \\ \vdots \\ f_m(t, x_1(t), \dots, x_m(t)) \end{bmatrix}$$

これを解くには、初期条件  $(t_0, x_1^{(0)}, \dots, x_m^{(0)})$  から始めて、 $t_n = t_0 + nh, (n = 1, 2, \dots, N)$  における  $X(t_n)$  の値  $x_1^{(n)}, x_2^{(n)}, \dots, x_m^{(n)}$  を順次計算する。これには前節までの手法を用いる。例えば、ルンゲ・クッタ法を用いると、以下ようになる。

1. 初期条件  $(t_0, x_1^{(0)}, x_2^{(0)}, \dots, x_m^{(0)})$  を定め、 $n = 0$  とする。
2.  $t_n$  時点の  $X(t_n) = (x_1^{(n)}, x_2^{(n)}, \dots, x_m^{(n)})$  を用いて  $k_{1i}, k_{2i}, k_{3i}, k_{4i}$  を以下の式で計算する。

$$\begin{aligned} k_{1i} &= hf_i(t_n, x_1^{(n)}, x_2^{(n)}, \dots, x_m^{(n)}), \quad i = 1, \dots, m \\ k_{2i} &= hf_i\left(t_n + \frac{h}{2}, x_1^{(n)} + \frac{k_{11}}{2}, x_2^{(n)} + \frac{k_{12}}{2}, \dots, x_m^{(n)} + \frac{k_{1m}}{2}\right) \\ k_{3i} &= hf_i\left(t_n + \frac{h}{2}, x_1^{(n)} + \frac{k_{21}}{2}, x_2^{(n)} + \frac{k_{22}}{2}, \dots, x_m^{(n)} + \frac{k_{2m}}{2}\right) \\ k_{4i} &= hf_i(t_n + h, x_1^{(n)} + k_{31}, x_2^{(n)} + k_{32}, \dots, x_m^{(n)} + k_{3m}) \end{aligned}$$

3.  $t_{n+1}$  時点の  $X(t_{n+1}) = (x_1^{(n+1)}, x_2^{(n+1)}, \dots, x_m^{(n+1)})$  を以下の式で求める。

$$x_i^{(n+1)} = x_i^{(n)} + \frac{k_{1i} + 2k_{2i} + 2k_{3i} + k_{4i}}{6}, \quad i = 1, \dots, m$$

4.  $n = n + 1$  とする。 $n < N$  ならば 2. に戻る。 $n = N$  ならば終了。

特に、変数の数が  $m = 2$  の場合に、 $x_n = x_1^{(n)}, y_n = x_2^{(n)}$  と表し、微分方程式を

$$\begin{aligned} \frac{dx(t)}{dt} &= f_x(t, x(t), y(t)) \\ \frac{dy(t)}{dt} &= f_y(t, x(t), y(t)) \end{aligned}$$

と表すと以下ようになる。

1. 初期条件  $(t_0, x_0, y_0)$  を定め,  $n = 0$  とする.
2.  $t_n$  時点の  $X(t_n) = (x_n, y_n)$  を用いて  $k_{xi}, k_{yi}$  を以下の式により以下の順序で計算する.

$$k_{x1} = hf_x(t_n, x_n, y_n) \quad (8.3.3)$$

$$k_{y1} = hf_y(t_n, x_n, y_n)$$

$$k_{x2} = hf_x\left(t_n + \frac{h}{2}, x_n + \frac{k_{x1}}{2}, y_n + \frac{k_{y1}}{2}\right)$$

$$k_{y2} = hf_y\left(t_n + \frac{h}{2}, x_n + \frac{k_{x1}}{2}, y_n + \frac{k_{y1}}{2}\right)$$

$$k_{x3} = hf_x\left(t_n + \frac{h}{2}, x_n + \frac{k_{x2}}{2}, y_n + \frac{k_{y2}}{2}\right)$$

$$k_{y3} = hf_y\left(t_n + \frac{h}{2}, x_n + \frac{k_{x2}}{2}, y_n + \frac{k_{y2}}{2}\right)$$

$$k_{x4} = hf_x(t_n + h, x_n + k_{x3}, y_n + k_{y3})$$

$$k_{y4} = hf_y(t_n + h, x_n + k_{x3}, y_n + k_{y3})$$

3.  $t_{n+1}$  時点の  $X(t_{n+1}) = (x_{n+1}, y_{n+1})$  を以下の式で求める.

$$x_{n+1} = x_n + \frac{k_{x1} + 2k_{x2} + 2k_{x3} + k_{x4}}{6}, \quad y_{n+1} = y_n + \frac{k_{y1} + 2k_{y2} + 2k_{y3} + k_{y4}}{6}$$

4.  $n = n + 1$  とする.  $n < N$  ならば 2. に戻る.  $n = N$  ならば終了.

以上ではルンゲ・クッタ法を用いたが, 前節の他の手法も同様に適用することができる. 例えば, オイラー法の場合には以下になる.

1. 初期条件  $(t_0, x_0, y_0)$  を定め,  $n = 0$  とする.
2.  $t_n$  時点の  $X(t_n) = (x_n, y_n)$  を用いて,  $t_{n+1}$  時点の  $X(t_{n+1}) = (x_{n+1}, y_{n+1})$  を以下の式で求める.

$$x_{n+1} = x_n + hf_x(t_n, x_n, y_n)$$

$$y_{n+1} = y_n + hf_y(t_n, x_n, y_n)$$

右辺が  $t_n, x_n, y_n$  で, 左辺が  $x_{n+1}, y_{n+1}$  であることに注意せよ.

3.  $n = n + 1$  とする.  $n < N$  ならば 2. に戻る.  $n = N$  ならば終了.

**追加問題 8.1** 次の連立 1 階常微分方程式を考える.

$$\frac{dx(t)}{dt} = -y(t) + \frac{3}{5}x(t)$$

$$\frac{dy(t)}{dt} = 4x(t) - y(t)$$

$$x(0) = 1$$

$$y(0) = 0$$

この連立微分方程式は解析的に解くことができ、

$$x(t) = \frac{1}{42}e^{-\frac{1}{5}t} \left( 42 \cos\left(\frac{2\sqrt{21}}{5}t\right) + 4\sqrt{21} \sin\left(\frac{2\sqrt{21}}{5}t\right) \right)$$

$$y(t) = \frac{10}{\sqrt{21}}e^{-\frac{1}{5}t} \sin\left(\frac{2\sqrt{21}}{5}t\right)$$

となる。この連立微分方程式をオイラー法、ルンゲ・クッタ法で解き、上記の解析的な解と比較せよ。  $h = 0.2$  とし、  $0 \leq t \leq 10$  の範囲を計算せよ。

問題 8.1 と同様にファイルを作成し（ただし  $t, x(t), y(t)$  をファイルに記録）、グラフ 1（横軸を  $t$ 、縦軸を  $x(t)$ ）、およびグラフ 2（横軸を  $t$ 、縦軸を  $y(t)$ ）、グラフ 3（横軸を  $x(t)$ 、縦軸を  $y(t)$ ）の 3 つにより各手法の精度を比較せよ。解析解は、C 言語で求めても良いし、表計算ソフト (OpenOffice, Excel) で求めても良い。

また、 $h$  を小さくして ( $h = 0.01$  など) 同様に各手法の精度を比較せよ。 (追加問題終)

**追加問題 8.2** 例 8.2 のパラメータを適宜設定して、クジラとイワシの個体数の変化を計算せよ。上記の変数の数が  $m = 2$  の場合のルンゲ・クッタ法を用いて解くこと。  $h = 0.01$  とし、  $0 \leq t \leq 30$  の範囲を計算せよ。また、計算結果をファイルに記録せよ。データは 1 行に一時刻のデータをカンマ (,) で区切って記入せよ。例えば、

時刻 1, 時刻 1 の  $x$  の値, 時刻 1 の  $y$  の値

時刻 2, 時刻 2 の  $x$  の値, 時刻 2 の  $y$  の値

時刻 3, 時刻 3 の  $x$  の値, 時刻 3 の  $y$  の値

...

というような形式のファイルを作成せよ。ファイル名の拡張子は csv とせよ。

さらに、エクセル (Windows), OpenOffice calc (Linux) などのツールで作成したファイルを読み込み、横軸を時間、縦軸を  $x$  および  $y$  としたグラフを作成せよ。ここで見られる現象を振動的共存というが、どういう現象かグラフで確認せよ。 (追加問題終)

**追加問題 8.3** 地表から斜めに放出された物体の運動（弾道など）は、空気抵抗がない場合には解析的に解くことができるが、空気抵抗がある場合には解析的に解くことはできない。空気抵抗がある場合の運動方程式は以下の微分方程式となる。

$$\frac{dx}{dt} = v_x \quad (8.3.4)$$

$$\frac{dy}{dt} = v_y \quad (8.3.5)$$

$$m \frac{dv_x}{dt} = -b_1 v_x - b_2 |v| v_x \quad (8.3.6)$$

$$m \frac{dv_y}{dt} = -mg - b_1 v_y - b_2 |v| v_y \quad (8.3.7)$$

ここで  $(x, y)$  は物体の座標（水平方向  $x$ ，垂直方向  $y$ ）， $\mathbf{v} = (v_x, v_y)$  は物体の速度， $g$  は重力加速度， $b_1$  は速度に比例する空気抵抗の係数， $b_2$  は速度の 2 乗に比例する空気抵抗の係数である．実測によると，断面積がある程度大きな物体に対して  $b_1|\mathbf{v}| \ll b_2|\mathbf{v}|^2$  となるので， $b_1$  の項を無視すると以下ようになる．

$$\frac{dx}{dt} = v_x \quad (8.3.8)$$

$$\frac{dy}{dt} = v_y \quad (8.3.9)$$

$$\frac{dv_x}{dt} = -\frac{b_2}{m} v_x \sqrt{v_x^2 + v_y^2} \quad (8.3.10)$$

$$\frac{dv_y}{dt} = -g - \frac{b_2}{m} v_y \sqrt{v_x^2 + v_y^2} \quad (8.3.11)$$

$$(8.3.12)$$

初期値は  $t = 0$  において  $(x, y) = (0, 0)$ ， $(v_x, v_y) = (v_0 \cos \theta, v_0 \sin \theta)$  とする． $v_0$  は初速， $\theta$  は放出する角度（水平方向を 0 度）である．

$g = 9.8[m/s^2]$ ， $m = 1[kg]$ ， $b_2 = 0.15$ ，初速度  $v_0 = 10[m/s]$ ，放出する角度を  $\theta = 30^\circ$  として，放出する物体の位置  $(x, y)$  のグラフを作成せよ．ルンゲ・クッタ法を用いること．変数は  $x, y, v_x, v_y$  の 4 つであり，8.3.3 は  $k_{x1} = hf_x(t_n, x_n, y_n, v_{xn}, v_{yn})$ ， $k_{y1} = hf_y(t_n, x_n, y_n, v_{xn}, v_{yn})$ ， $k_{v_x1} = hf_{v_x}(t_n, x_n, y_n, v_{xn}, v_{yn})$ ， $k_{v_y1} = hf_{v_y}(t_n, x_n, y_n, v_{xn}, v_{yn})$  などとなることに注意すること．

次に，放出する角度を  $15^\circ, 30^\circ, 45^\circ, 60^\circ, 75^\circ$  とした場合の物体の位置のグラフを作成せよ．1 枚にまとめたグラフ（問題 8.1 のグラフ参照）にすること．

これらに対し，飛距離が一番大きくなる角度の近くを 1 度刻み（ $30^\circ, 31^\circ, 32^\circ, \dots, 45^\circ$  など）で変化させて，飛距離が一番大きくなる角度を求めよ．これも同様に 1 枚にまとめたグラフを作成せよ．

(追加問題終)

## 8.4 線形 2 階常微分方程式

ここでは線形 2 階常微分方程式を扱う．

**例 8.3** 質量  $m$  の物体をバネ定数  $k$  のバネにつるし，速度に比例する抵抗係数  $\beta$  の制動装置のもとで運動させる．静止時の平衡点を原点とし，下向きに  $x$  をとる． $t = 0$  で  $x = x_0$  までバネを引き伸ばして運動を開始したとき，物体の運動を求めよ．

### 8.4.1 理論解

質点の運動方程式は次式となる．詳細は省略する．

$$m \frac{d^2 x}{dt^2} + \beta \frac{dx}{dt} + kx = 0 \quad (8.4.1)$$

$$\text{初期条件: } t = 0 \text{ において } x(0) = x_0, \quad \left. \frac{dx}{dt} \right|_{t=0} = 0$$

式 (8.4.1) の特性方程式は

$$\lambda^2 + \frac{\beta}{m} \lambda + \frac{k}{m} = 0$$

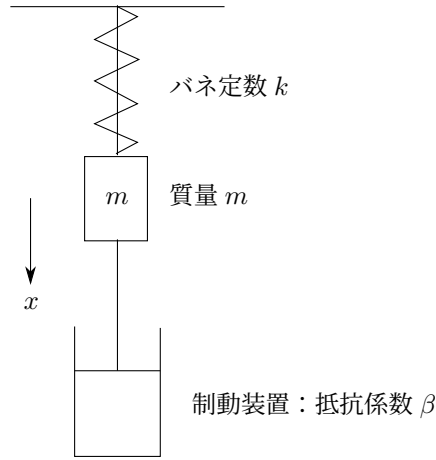


図 8.2 バネと制動装置

となり，その根は

$$\lambda = -\frac{\beta}{2m} \pm \frac{\sqrt{\beta^2 - 4km}}{2m}$$

となる．これより式 (8.4.1) の解は次の 3 つの場合に分けられる．

1.  $\beta^2 < 4km$ ：過小制動（抵抗が小さい）の場合： $\lambda = -b \pm i\omega$

$$x = e^{-bt} \left( A \sin(\omega t) + B \cos(\omega t) \right)$$

$$D = \beta^2 - 4km, \quad b = \frac{\beta}{2m}, \quad \omega = \frac{\sqrt{-D}}{2m}, \quad A = \frac{\beta x_0}{\sqrt{-D}}, \quad B = x_0$$

この場合には上式は以下のようにかけるので，減衰振動となることがわかる．

$$x = C e^{-bt} \sin(\omega t + a)$$

$$C = \sqrt{A^2 + B^2}, \quad a = \tan^{-1} \left( \frac{B}{A} \right)$$

2.  $\beta^2 = 4km$ ：臨界制動の場合： $\lambda = -b$

$$x = e^{-bt} (A + Bt)$$

$$b = \frac{\beta}{2m}, \quad A = x_0, \quad B = bx_0$$

3.  $\beta^2 > 4km$ ：過大制動（抵抗が大きい）の場合： $\lambda = -b_1, -b_2$

$$x = A e^{-b_1 t} + B e^{-b_2 t}$$

$$D = \beta^2 - 4km, \quad b_1 = \frac{\beta - \sqrt{D}}{2m}, \quad b_2 = \frac{\beta + \sqrt{D}}{2m}$$

$$A = \frac{(D + \beta\sqrt{D}) x_0}{2D}, \quad B = \frac{(D - \beta\sqrt{D}) x_0}{2D}$$

この場合は無周期の強制減衰振動となる．

上述の微分方程式 (8.4.1) は一般的には次式でかける．

$$\begin{aligned}\frac{d^2x}{dt^2} &= f(t, x, x') \\ x' &= \frac{dx}{dt} \\ x(t_0) &= x_0 \\ x'(t_0) &= x'_0\end{aligned}\tag{8.4.2}$$

以下では，式 (8.4.2) に対して数値解法を考える．

#### 8.4.2 数値解法 1：連立 1 階常微分方程式に変換する方法

式 (8.4.2) において， $y = \frac{dx(t)}{dt}$  とおけば， $\frac{dy(t)}{dt} = \frac{d^2x(t)}{dt^2} = f(t, x, x') = f(t, x, y)$  なので，次式が成立する．

$$\begin{aligned}\frac{dx(t)}{dt} &= y(t) \\ \frac{dy(t)}{dt} &= f(t, x, y) \\ \text{初期条件：} x(t_0) &= x_0, \quad y(t_0) = x'_0\end{aligned}$$

これは， $(x, y)$  の 1 階連立常微分方程式であり，前節の方法で数値計算できる．さらに一般に  $n$  階の微分方程式は，同様の方法で  $n$  変数の連立 1 階常微分方程式に変換できる．

例 8.3 を連立 1 階常微分方程式に変換してみる．

$$m \frac{d^2x}{dt^2} + \beta \frac{dx}{dt} + kx = 0$$

より，

$$\frac{d^2x}{dt^2} = -\frac{\beta}{m} \frac{dx}{dt} - \frac{k}{m} x\tag{8.4.3}$$

となる． $y = \frac{dx}{dt}$  とすると， $\frac{d^2x}{dt^2} = \frac{dy}{dt}$  なので，

$$\frac{dy}{dt} = -\frac{\beta}{m} y - \frac{k}{m} x$$

となる．また  $y(0) = \left. \frac{dx}{dt} \right|_{t=0} = 0$  となる．これより，求める連立 1 階常微分方程式は，

$$\begin{aligned}\frac{dx(t)}{dt} &= y(t) \\ \frac{dy(t)}{dt} &= -\frac{\beta}{m} y(t) - \frac{k}{m} x(t) \\ x(0) &= x_0, \quad y(0) = 0\end{aligned}\tag{8.4.4}$$

となる．

#### 8.4.3 数値解法 2：陽的離散変数法

オイラー法を式 (8.4.2) に適用すると以下ようになる.

$x(t+h)$  および  $x'(t+h)$  を  $t$  のまわりでテーラー展開して  $x''(t)$  までの項をとると次式が得られる.

$$\begin{aligned} x(t+h) &\doteq x(t) + hx'(t) + \frac{h^2}{2}x''(t) \\ &= x(t) + hx'(t) + \frac{h^2}{2}f(t, x, x') \\ x'(t+h) &\doteq x'(t) + hx''(t) \\ &= x'(t) + hf(t, x, x') \end{aligned}$$

これより,  $t_n$  時点の値  $(t_n, x_n, x'_n)$  が求まっているとき,  $t_{n+1}$  時点の値は次式で求められる.

$$\begin{aligned} x_{n+1} &= x_n + hx'_n + \frac{h^2}{2}f(t_n, x_n, x'_n) \\ x'_{n+1} &= x'_n + hf(t_n, x_n, x'_n) \end{aligned} \tag{8.4.5}$$

#### 8.4.4 数値解法 3：陰的離散変数法

上述した方法は  $t_n$  の値を用いて  $t_{n+1}$  の値を求める方法であるが,  $t_{n+1}$  の値と  $t_{n-1}$  の値を平均して  $t_n$  の値を求めるという連立 1 次方程式として扱う方法がある. この方法によると以下の式が成り立つ.

$$\begin{aligned} \left. \frac{d^2x(t)}{dt^2} \right|_{t=t_n} &\doteq \frac{(\Delta x_n/h) - (\Delta x_{n-1}/h)}{h} \\ &= \frac{x_{n+1} - 2x_n + x_{n-1}}{h^2} \\ \left. \frac{dx(t)}{dt} \right|_{t=t_n} &\doteq \frac{\Delta x_n}{h} \\ &= \frac{(x_{n+1} - x_{n-1})/2}{h} \end{aligned}$$

これを  $t_n$  時点の計算において式 (8.4.2) に代入すると以下が得られる.

$$x_{n+1} - 2x_n + x_{n-1} = h^2 f\left(t_n, x_n, \frac{x_{n+1} - x_{n-1}}{2h}\right) \tag{8.4.6}$$

したがって式 (8.4.2) の常微分方程式は, 式 (8.4.6) の(非線形)連立方程式となる. 上式は  $n = 1, 2, \dots, N-1$  で成立し, その中には  $(x_1, \dots, x_N)$  の  $N$  個の未知数があるから ( $x_0$  は初期条件で与えられている), 方程式が 1 つ足りないが, これには  $x'(t_0)$  を用いる.

$$x'(t_0) = \left. \frac{dx}{dt} \right|_{t=t_0} \doteq \frac{x_1 - x_0}{h} \tag{8.4.7}$$



式 (8.4.6), (8.4.7) を連立させて  $(x_1, x_2, \dots, x_N)$  について解けばよい. この方法は微分を離散的な差分で近似することによって, 微分方程式を連立の代数方程式に置き換える方法であり, 式 (8.4.2) の微分方程式に限らず一般的に偏微分方程式の解法にも用いられる.

**追加問題 8.4** 例 8.3 の過小制動, 臨界制動, 過大制動の各場合についてパラメータ  $m, \beta, k, x_0$  を適宜設定し, 数値解法 1 でルンゲ・クッタ法を用いた場合と数値解法 2 の 2 種類を計算し, 理論解と比較して精度を比較せよ. まず式 (8.4.3) と式 (8.4.2) を比較し, 関数  $f$  を求め, 数値解法 1 では, 式 (8.4.4) を式 (8.3.3) 等に適用する (追加問題 8.1 参照). 数値解法 2 では, 求めた関数  $f$  を式 (8.4.5) に適用する. グラフを作成して比較せよ. また, パラメータは減衰振動などがグラフでわかるように設定せよ. (追加問題終)

**追加問題 8.5** 2 階常微分方程式

$$\frac{d^2x(t)}{dt^2} + t \frac{dx(t)}{dt} + 20x = 0, \quad x(0) = 1, \quad \left. \frac{dx(t)}{dt} \right|_{t=0} = 0$$

を 1 階連立微分方程式に書き換え, ルンゲ・クッタ法により  $x(t)$  を求めよ.  $h$  などのパラメータは適宜設定せよ. グラフも作成せよ.  $n = 0$  のときの式 (8.3.3) の右辺に注意して作成せよ. (追加問題終)

## 9 シミュレーションのための知識

### 9.1 疑似乱数の発生

疑似乱数の作り方 (C 言語)

- `#include <stdlib.h>` とする.
- `rand()` という関数で, 0 から `RAND_MAX` までの整数の一樣乱数を発生する. 通常 `RAND_MAX=32767` であるが, 既に定義されているのでユーザーがこの値を定義する必要はない.
- $[0, 1)$  の一樣乱数の生成法 ( $[0, 1)$  とは 0 以上 1 未満という意味)
  - `drand48()` は  $[0, 1)$  の一樣乱数 (実数:float) を発生する.
  - `(double)rand()/(RAND_MAX+1.0)` とする.  
注意: `(double)rand()/(RAND_MAX+1)` としてはだめ. (1.0 は良い. 1 はダメ)  
理由は, `RAND_MAX+1` が `int` の範囲を超えるから.
- $(0, 1]$  の一樣乱数の生成法 ( $(0, 1]$  とは 0 より大きく, 1 以下という意味)
  - `(double)(rand()+1.0)/(RAND_MAX+1.0)` とする.  
注意: `(double)(rand()+1)/(RAND_MAX+1)` としてはだめ. (1.0 は良い. 1 はダメ)  
理由は, `RAND_MAX+1` が `int` の範囲を超えるから.
- $[0, 1]$  の一樣乱数の生成法 (0 以上, 1 以下)  
`(double)(rand())/RAND_MAX` とする.
- $0, 1, \dots, n-1$  の整数の乱数の生成法
  - `(int)(n*drand48())` とする.
  - `(int)((double)n*rand()/(RAND_MAX+1.0))` とする.
  - `rand()%n` とするのは良くない. 各整数の発生確率が異なることがある. ただし, 各整数の発生確率が等しくなくても良い場合には用いても良い.
- 乱数の初期化について. 乱数を初期化しない場合にはプログラムを実行するたびに同じ乱数列が生成される. プログラムを実行するたびに異なる乱数列を生成したい場合には, 以下を行う.
  - `drand48()` で乱数列を発生させる場合: `#include <time.h>` として, `main()` の中の最初の行で `srand48(time(NULL));` とする. これにより, プログラムを起動する時間が異なれば, 異なる乱数列が生成される.
  - `rand()` で乱数列を発生させる場合: `#include <time.h>` として, `main()` の中の最初の行で `srand(time(NULL));` とする.

**追加問題 9.1**  $[0, 5)$  間の一樣乱数を発生させ,  $[1, 2)$  間になる乱数の数の割合を表示させよ. 理論的には 0.2 である. 発生させる乱数の総数が  $N = 10, 100, 1000, 10000, 100000$  個の 5 つの場合について示せ. (追加問題終)

**追加問題 9.2** 次の積分を、以下に示す乱数を用いる方法で推定せよ。

$$\int_0^1 x^2 dx$$

この積分は、 $0 \leq x \leq 1$  間で曲線  $y = x^2$  の下で  $x$  軸より上の領域  $A$  の面積を求めることに等しいが、正方形  $0 \leq x \leq 1, 0 \leq y \leq 1$  (または正方形  $0 \leq x < 1, 0 \leq y < 1$ ) の 2 次元平面上に任意の乱数を発生させ、領域  $A$  に入った乱数の割合で積分の値を推定せよ。すなわち、

$$\text{積分の推定値} = \frac{\text{領域 } A \text{ に入った乱数の個数}}{\text{乱数全体の個数}}$$

である。乱数全体の個数が  $N = 10, 100, 1000, 10000, 100000$  の 5 つの場合について上の積分の値を推定し、正しい値と比較せよ。(追加問題終)

## 9.2 一様乱数から特定の確率分布の発生

$(0, 1]$  間で一様分布する確率変数 (乱数)  $U$  から、分布関数  $F(x)$  (確率密度関数  $f(x)$ ) に従う確率変数  $X$  を作る。

### 9.2.1 逆関数法

一様乱数  $U$  から  $X = F^{-1}(U)$  により計算した値  $X$  は、確率分布  $F(x)$  を持つ。

ただし逆関数法は、分布関数  $F(x)$  の逆関数  $F^{-1}(x)$  が簡単に書ける場合にしか適用できない。正規分布では  $F^{-1}$  は簡単に書けないのでこの方法は適用できない。

**【証明】**  $Pr(X \leq x) = Pr(F^{-1}(U) \leq x) = Pr(U \leq F(x)) = F(x)$  (証明終)

### 例 9.1 指数分布の発生.

指数分布の確率密度関数は  $f(x) = \lambda e^{-\lambda x}$ , 分布関数は  $F(x) = 1 - e^{-\lambda x}$  である。よって

$$F^{-1}(y) = -\frac{\log(1-y)}{\lambda}$$

したがって、 $(0, 1]$  間の一様乱数  $U$  を発生させ、

$$X = -\frac{\log(1-U)}{\lambda}$$

により計算した  $X$  は、パラメータ  $\lambda$  の指数分布を持つ。

### 9.2.2 キンダーマン・モナハン法

確率密度関数  $f(x)$  をとる非負の確率変数  $X$  を以下のようにして作成する。一様乱数  $U$  と  $V$  による平面上的点  $(U, V)$  が領域  $R = \{(u, v) | 0 \leq u \leq \sqrt{f(u/v)}\}$  内にある場合にのみ  $X = V/U$  により計算した値を採用する。

【証明】 領域  $R$  に入った点が  $X \leq x$  を満たす確率は、点  $(U, V)$  が一様乱数から作成されることを考えると、領域  $A = R \cap \{(u, v) | v/u \leq x\}$  の面積を領域  $R$  の面積で割ったものに等しい。ゆえに

$$Pr(X \leq x) = \frac{\int_A du dv}{\int_R du dv}$$

である。

ここで  $t = v/u$  により変数  $(u, v)$  を  $(u, t)$  に変数変換する。  $dv = u dt$  であり、

$$\begin{aligned} \int_R du dv &= \int_0^\infty \int_0^{\sqrt{f(t)}} u du dt = \int_0^\infty f(t)/2 dt = 1/2 \\ \int_A du dt &= \int_0^x \int_0^{\sqrt{f(t)}} u du dt = \int_0^x f(t)/2 dt = F(x)/2 \end{aligned}$$

であるから、  $Pr(X \leq x) = F(x)$  となる。

また、この計算は  $R = \{(u, v) | 0 \leq u \leq \sqrt{K f(u/v)}\}$  と確率密度関数を定数  $K$  倍しても同じである。(証明終)

例 9.2 標準正規分布をキンダーマン・モナハン法で生成する。

正規分布確率変数  $X$  に対し、その絶対値  $|X|$  の確率密度関数は  $f(x) = \sqrt{2/\pi} \exp(-x^2/2)$  である。まずこの  $|X|$  の作成をキンダーマン・モナハン法により行う。

$K = \sqrt{\pi/2}$  として、  $u \leq \sqrt{K f(v/u)}$  を変形すると、  $(v/u)^2 \leq -4 \log u$  となる。

これより、標準正規分布の作成は以下ようになる。

1.  $(0, 1]$  間の一様乱数  $U, V$  を生成し、  $(V/U)^2 \leq -4 \log U$  ならば  $V/U$  を計算する。そうでなければ乱数  $U, V$  を生成しなおす。
2.  $1/2$  の確率で  $V/U$  の値を、  $1/2$  の確率で  $-V/U$  の値を採用する。すなわち、  $0, 1$  の整数の乱数を発生させ（発生方法は上述）、その乱数が  $0$  なら  $V/U$  の値を、  $1$  なら  $-V/U$  の値を出力する。

### 9.2.3 標準正規分布の発生（中心極限定理による）

中心極限定理とは、以下の定理である。

**定理 9.1** 確率変数  $X_1, X_2, \dots, X_n$  が平均  $\mu$ 、分散  $\sigma^2$  の同じ分布を持つ独立な確率変数であるとする。このとき  $(X_1 + X_2 + \dots + X_n - n\mu)/(\sqrt{n}\sigma)$  は  $n \rightarrow \infty$  のとき、平均  $0$ 、分散  $1$  の標準正規分布になる。

$[0, 1]$  の一様分布の平均は  $\mu = 1/2$ 、分散は  $\sigma^2 = 1/12$  である。したがって、  $[0, 1]$  の一様分布  $U_1, U_2, \dots, U_n$  を作成し、  $X = (U_1 + U_2 + \dots + U_n - n/2)/\sqrt{n/12}$  により計算した  $X$  は標準正規分布の近似となる。

平均と分散の値について補足すると、今の場合の確率密度関数は

$$f(x) = \begin{cases} 1, & (0 \leq x < 1) \\ 0, & (\text{それ以外の } x) \end{cases}$$

なので、平均は、

$$\mu = \int_{-\infty}^{\infty} xf(x)dx = \int_0^1 xdx = \left[\frac{1}{2}x^2\right]_0^1 = \frac{1}{2}$$

となり、 $\mu = 1/2$  となる。

分散は、

$$\sigma^2 = \int_{-\infty}^{\infty} (x - \mu)^2 f(x)dx = \int_0^1 (x^2 - 2\mu x + \mu^2)dx = \left[\frac{1}{3}x^3 - \mu x^2 + \mu^2 x\right]_0^1 = \frac{1}{3} - \frac{1}{2} + \left(\frac{1}{2}\right)^2 = \frac{1}{12}$$

となり、 $\sigma^2 = 1/12$  となる。

標準正規分布を生成する方法は、ほかにもボックス・ミュラー (Box-Muller) 法などがあるが、ここでは省略する。

**追加問題 9.3** キンダーマン・モナハン法 (例 9.2)、中心極限定理 ( $n = 6$  の場合、第 9.2.3 節) により、標準正規分布を 10000 個作成し、分布の形を確認せよ。分布の形は以下のようにして表示せよ。

-3 から 3 の間を 0.2 刻みの区間に分け、そのそれぞれの区間に 10000 個の乱数のうちいくつが入るかを数えよ。すなわち、乱数  $r$  が  $-3 + 0.2k \leq r < -3 + 0.2(k+1)$  のときは、配列の  $k$  番目の要素を 1 増やす。ただし  $k = 0, 1, \dots, 29$  である。 $k = 0$  のとき「 $-3 \leq r < -2.8$ 」で、 $k = 29$  のとき「 $2.8 \leq r < 3$ 」である。

画面表示では、以下の 2 つのうち 1 つを選択せよ。いずれの場合でも、キンダーマン・モナハン法、中心極限定理による方法の 2 つを表示せよ。

- 各区間に入っている乱数の数をファイルに出力し、そのファイルを OpenOffice calc などを読み込んでグラフにする。
- コンソール画面表示で、1 つの行が 1 つの区間に対応し、20 個の乱数につき 1 つの \* を表示することで各行に各区間の乱数の個数を図示する。これにより分布が (縦横を入れ替えた形で) 表示される。

(追加問題終)

## 9.3 再帰

ある処理の中で、単にパラメータのみが違うだけで、自らの処理を利用できる場合に、自分自身を再度呼び出すことがある。これを再帰処理という。

**例 9.3**

$$\sum_{k=1}^n k = 1 + 2 + \dots + n = \sum_{k=1}^{n-1} k + n$$

である。

$$\text{wa}(n) = \sum_{k=1}^n k$$

とすると、 $\text{wa}(n) = \text{wa}(n-1) + n$  とかける。

また、初期条件として  $wa(1) = 1$  である。

これをまとめると、再帰を用いて上記の和を計算するプログラムは以下のようになる。

```
#include <stdio.h>
int wa(int n); // 関数のプロトタイプ宣言
int main(){
    int n=10;
    printf("n=%d wa=%d\n",n,wa(n));
}

int wa(int n){
    if(n==1){
        return 1; // n=1 のときの初期値
    }
    int x;
    x=wa(n-1)+n; // 再帰
    return x;
}
```

**追加問題 9.4** 再帰を用いて  $n!$  を計算せよ。(ヒント： $n! = (n-1)! \times n$ )

(追加問題終)

**追加問題 9.5** 再帰を用いて次の2つを計算せよ。

$$1 + 3 + 5 + \cdots + (2n + 1)$$

$$1 \times 3 \times 5 \times \cdots \times (2n + 1)$$

(追加問題終)

## 9.4 列挙の方法

**追加問題 9.6** A 町から B 町へは 3 本の道 1,2,3 がある。B 町から C 町へは 6 本の道 a,b,c,...,f がある。A 町から C 町へ行く道の取り方を列挙せよ。

画面への表示例：(1,a)

全部で何通りあるかを計算して、それと表示の数が一致していることを確認せよ。

(追加問題終)

**追加問題 9.7** 袋の中に赤、青、白の球がそれぞれ 4, 3, 2 個入っている。1 つの球を取り出してはまた袋に戻しながら（復元抽出）、合計 4 回の取り出しを行う。（赤、青、白）の出る回数の組み合わせをすべて出力せよ。

画面への表示例：（赤、青、白）= (2,2,0)

全部で何通りあるかを計算して、それと表示の数が一致していることを確認せよ。

ヒント：何通りあるかの計算には、次の公式を用いよ。

異なる  $n$  個のもののうちから重複を許して  $r$  個取ってくる組み合わせの総数を  ${}_nH_r$  で表す。  ${}_nH_r = {}_{n+r-1}C_r$

である。  ${}_nC_r = \binom{n}{r} = \frac{n!}{r!(n-r)!}$

また、取り出した球を袋に戻さない場合（非復元抽出）の（赤、青、白）の出る組み合わせを列挙せよ。

全部で何通りあるかを計算して、それと表示の数が一致していることを確認せよ。

ヒント：計算には次のような公式がある（共立出版 数学公式）。

$n$  個のもののうち  $A$  が  $p$  個、 $B$  が  $q$  個、 $C$  が  $r$  個 あるとき、この  $n$  個から  $k$  個とった組み合わせの数は、 $(1+x+\cdots+x^p)(1+x+\cdots+x^q)(1+x+\cdots+x^r)$  の展開式における  $x^k$  の係数に等しい。

その係数を  $c_k$  とし、式の展開を Mathematica 等で行うと、以下の式が得られる（詳細は省略）。

$$S(n, m, k) = \frac{1}{2}(k+m) \min(0, k+m+1) - \frac{1}{2}(k+n-1) \min(0, k+n)$$

として、 $m_0 = \min(r, k)$ 、 $M_0 = \max(0, k - (p+q))$  とすると以下になる。

$$c_k = S(-m_0, -M_0, k-q) + S(M_0, m_0, p-k) + (q+1)(m_0 - M_0 + 1)$$

(追加問題終)

**追加問題 9.8** 上のプログラムが完成した後、合計取り出し回数と赤、青、白の球の数が任意に設定できるようにプログラムを修正せよ。もちろん、非復元抽出の場合には、球の全部の数は取り出し回数以上であるように設定する。また、次の仕様でプログラムを作成せよ。

1. 赤、青、白の球の数、および取り出し回数を `scanf()` 等を用いてキーボードから入力せよ。
2. すべての組み合わせを `printf()` で出力せよ。
3. すべての組み合わせの数を変数 `cnt` に数え上げるように、上の `printf()` の直後に `cnt++;` を置け。
4. 組み合わせの数を、前の問題のように計算して比較せよ。

(追加問題終)

数え上げのもっとも原始的なやり方

r: 赤の個数, b: 青の個数, w: 白の個数

```
for(r=0;r<=4;r++){
    for(b=0;b<=3;b++){
        for(w=0;w<=2;w++){
```

```
if(w+b+w==4) printf("(r,b,w)=(%d,%d,%d)\n",r,b,w);  
.....
```

しかし、これだと  $5 \times 4 \times 3$  回の数え上げを行っている。

$r$  と  $b$  が決まっていれば  $w = 4 - r - b$  となるので、

```
for(r=0;r<=4;r++){  
    for(b=0;b<=3;b++){  
        w=4-r-b;  
        printf("(r,b,w)=(%d,%d,%d)\n",r,b,w);  
        .....
```

とすれば、 $5 \times 4$  回の数え上げですむ。ただし、 $0 \leq w \leq 2$  でなければならないので、そのチェックは必要である。



## 10 プログラムの例（問題のヒント）

この章では、問題を解くためにヒントとなるプログラム例を示す。問題がわからない場合には、まずこの章のプログラムを入力して実行することから始めること。これらの例をすべて理解することができれば、問題は解けるはずである。

### 10.1 計算と表示

これはすべてのプログラムの基礎となる。

次のプログラムは、 $x = 2$  として、 $y = \frac{1}{x^2}$  を計算し、 $x$  の値と、 $y$  の値を表示するものである。

gedit などのエディタに以下を記入する。

左端の空白（インデント）は、タブキー (Tab) を用いて入力すること。

インデントにスペースは使用しない。

//より右はコメントなので入力する必要は無い。

\ は ¥ キーで入力する。

保存するファイル名は `sample_printf.c` などとする。他の名前でも良いが拡張子は `.c` にすること。また、ファイル名にはスペースを入れてはいけない。

```
#include <stdio.h>

int main(){
    int x;
    double y;

    x=2;
    y=1.0/(x*x); // 1.0 として浮動小数点にすることと、分母の括弧に注意

    printf("x=%d\n",x); // %d は int の表示
    printf("y=%f\n",y); // %f は float, double の表示
    return 0; // int main() が正常終了
}
```

コンパイルは端末のプロンプトで以下のように入力する。

```
cc sample_printf.c
```

実行は端末のプロンプトに以下のように入力する。

```
./a.out
```

$x = 2$ ,  $y = 0.25$  と表示されれば正しい。

## 10.2 数学関数 math.h の使用

第 6 章などで用いる。

次のプログラムは、 $x = 2$  として、 $y = \sqrt{x}$ 、 $z = \sin(x)$  を計算して表示するプログラムである。 $\sqrt{x}$ 、 $\sin(x)$  などを計算する場合には `math.h` のインクルードが必要である。保存するファイル名は `sample_math.c` などとする。

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

int main(){
    double x, y, z;
    x=2;
    y=sqrt(x);
    z=sin(x);
    printf("x=%f\n",x);
    printf("sqrt(x)=%f\n",y);
    printf("sin(x)=%f\n",z);
    return 0; // int main() が正常終了
}
```

`math.h` をインクルードするときは、コンパイルの際に `-lm` オプションが必要、

```
cc sample_math.c -lm
```

実行すると（前節参照）、 $x = 2$ 、 $y = 1.414214$ 、 $\sin(x) = 0.909297$  と表示されれば正しい。

### 10.3 scanf による入力

キーボードから数値などを入力したい場合は `scanf` を用いる。

```
#include <stdio.h>
int main(){
    float x, h;
    double xd, hd;
    int i, n;
    printf("input integer n, float h, double hd\n");
    scanf("%d %f %lf", &n, &h, &hd);
    x=0.0;
    xd=0.0;
    for (i=0; i<n; i++){
        x=x+h;
        xd=xd+hd;
    }
    printf("n=%d\n", n);
    printf("h=%f, n*h=%f\n", h, x);
    printf("hd=%f, n*hd=%f\n", hd, xd);
    return 0;
}
```

コンパイルして実行して

```
input integer n, float h, double hd (この行が表示される)
10000000 0.01 0.01 (この3つの数字を自分で入力して Enter を押す。0 は7つ)
```

$n = 10^7$ ,  $h = hd = 0.01$  なので,  $n \cdot h = n \cdot hd = 10^5$  となるはずだが, 結果はこれより大きい小さいか? 再び実行して,

```
input integer n, float h, double hd (この行が表示される)
10000000 0.03 0.03 (この3つの数字を自分で入力して Enter を押す。0 は7つ)
```

$n = 10^7$ ,  $h = hd = 0.03$  なので,  $n \cdot h = n \cdot hd = 3 \cdot 10^5$  となるはずだが, 結果はこれより大きい小さいか? (これらの結果の理由がわからなければ教官に聞きましょう)

```

#include <stdio.h>
int main(){
    int n, k, x;
    printf("input n\n");
    scanf("%d", &n);
    for (k=1; k<=n; k++){
        x = k*k;
        if (x<0){
            break;
        }
    }
    printf("n=%d\n", n);
    if (x<0){
        printf("when k=%d, k*k=%d<0\n", k, x);
        return -1;
    } else {
        printf("for k=1...%d, k*k>0\n", n);
    }
    return 0;
}

```

コンパイルして実行して

input n (この行が表示される)  
 10000 (この数字を自分で入力して Enter を押す. 0 は 4 つ)

$k$  に対し,  $k^2$  は正になるはず.  
 もう一度実行.

input n  
 1000000 (この数字を自分で入力して Enter を押す. 0 は 6 つ)

$k$  に対し,  $k^2$  は正になるはず?? (結果の理由がわからなければ教官に聞きましょう)

## 10.4 if 文

ほぼすべての章で用いる。以下のような符号関数 (sgn 関数) を計算する。

$$\text{sgn}(x) = \begin{cases} 1 & (x > 0) \\ 0 & (x = 0) \\ -1 & (x < 0) \end{cases}$$

```
#include <stdio.h>

int main(){
    double x,sgn;

    x=5; // この値を変更し、正の値, 0, 負の値の3種類行うこと.
    if(x>0){
        sgn=1;
    }else if(x==0){
        sgn=0;
    }else{
        sgn= -1;
    }

    printf("x=%f, sgn(x)=%f\n",x,sgn);
    return 0; // int main() が正常終了
}
```

$x$  の値を変更し、正の値, 0, 負の値の3種類行うこと。

## 10.5 for 文と $\sum$ の計算

第3章、第4章、第7章のプログラムの基礎となる。

$$\sum_{k=1}^n \frac{1}{k(k+1)} = \frac{n}{n+1}$$

の左辺と右辺をそれぞれプログラムで計算する。上記の等式が成立することは次のようにしてわかる。

$$\begin{aligned} \sum_{k=1}^n \frac{1}{k(k+1)} &= \sum_{k=1}^n \left( \frac{1}{k} - \frac{1}{k+1} \right) = \sum_{k=1}^n \frac{1}{k} - \sum_{k=1}^n \frac{1}{k+1} = \left( \frac{1}{1} + \sum_{k=2}^n \frac{1}{k} \right) - \left( \sum_{k=1}^{n-1} \frac{1}{k+1} + \frac{1}{n+1} \right) \\ &= \left( 1 + \sum_{k=2}^n \frac{1}{k} \right) - \left( \sum_{k=2}^n \frac{1}{k} + \frac{1}{n+1} \right) = 1 - \frac{1}{n+1} = \frac{n}{n+1} \end{aligned}$$

以下をエディタに入力する。  $n = 100000$  のときに上式が成立しているかを確かめる。(問題の参考にするため、大きい  $n$  で実施する。)

```
#include <stdio.h>
int main(){
    int k,n;
    n=100000;
    double s=0; // s の初期値を 0 とする。初期化は for の直前で行う (推奨)
    for(k=1;k<=n;k++){ // この for ループで左辺の  $\Sigma$  を計算
        s=s+1.0/((double)k*(k+1)); // この式を繰り返し実行すると  $\Sigma$  が計算できる。
    }
    printf("sum=%f\n",s); // 左辺の表示
    printf("n/(n+1)=%f\n", n/(n+1.0)); // 右辺の計算と表示
    return 0; // int main() が正常終了
}
```

表示された2つの値が同じなら正しい。

$s=s+\dots$  の  $(\text{double})k$  は、 $\text{int}$  型の整数  $k$  を、 $\text{double}$  型の浮動小数点数に変換することを意味している。

$(\text{double})k*(k+1)$  は、 $\text{double}$  型に変換した  $k$  に  $\text{int}$  型の  $(k+1)$  をかけることになるが、この場合には  $\text{int}$  型の  $(k+1)$  は  $\text{double}$  型に自動的に変換されてから掛け算される。

上記のプログラムでは、 $k = 1, k = 2, \dots, k = n$  と動かすが、もし for ループの  $k$  の順番を逆にして  $k = n, k = n-1, \dots, k = 1$  のように動かすときは、以下のようにする。

```
for(k=n;k>=1;k--){ // この for ループで左辺の  $\Sigma$  を計算
    s=s+1.0/((double)k*(k+1)); // この式を繰り返し実行すると  $\Sigma$  が計算できる。
}
```

## 10.6 for 文と $\sum$ の計算 (その2)

第3章, 第4章, 第7章のプログラムの基礎となる.

$$\sum_{k=1}^n k = \frac{n(n+1)}{2}$$

$$\sum_{k=1}^n k^2 = \frac{n(n+1)(2n+1)}{6}$$

であるが (高校で学んだはず),

$$\sum_{k=1}^n k + \sum_{k=1}^{2n} k^2 = \frac{n(n+1)}{2} + \frac{2n(2n+1)(4n+1)}{6}$$

の左辺と右辺をプログラムで計算して比較する.

```
#include <stdio.h>
int main(){
    int k,n;
    n=10;
    double s1, s2, s_left, s_right;
    s1=0; // s1 の初期化を忘れないこと. for の直前で初期化 (推奨)
    for(k=1;k<=n;k++){ // 左辺の一つ目の  $\Sigma$  の計算
        s1=s1+k; // + を忘れないこと
    }
    s2=0; // s2 の初期化をする. for の直前で初期化 (推奨)
    for(k=1;k<=2*n;k++){ // 左辺の2つ目の  $\Sigma$  の計算
        s2=s2+k*k; // + を忘れないこと
    }
    s_left=s1+s2; // 左辺
    s_right=n*(n+1)/2.0 + 2*n*(2*n+1)*(4*n+1)/6.0; // 右辺
    printf("s_left=%f s_right=%f\n",s_left, s_right);
    return 0; // int main() が正常終了
}
```

## 10.7 関数の作成

第 6 章，第 4 章などで用いる．

$$f(x) = \sin(x) + \frac{1}{x^3}$$

として， $x_1 = 1, x_2 = 2, x_3 = 3$  に対して， $y = f(x_1) + f(x_2) + f(x_3)$  を計算する．コンパイルの際は，第 10.2 節を参照．

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

double f(double x){ // 関数の宣言
    double y;
    y=sin(x) + 1.0/(x*x*x);
    return y; // return 文で返り値を指定することが必要
}

int main(){
    double y;
    double x1=1, x2=2, x3=3;
    y=f(x1)+f(x2)+f(x3);
    printf("y=%f\n",y);
    return 0; // int main() が正常終了
}
```

$y=3.053925$  と表示されれば正解．

関数は使う前に定義する必要がある．しかし，プログラムを見やすくするために後ろに記述したい場合もある．その場合，以下のように関数の返り値の型と引数の型のみを初めに記述する．これをプロトタイプ宣言という．

注意：入れ子関数（nested function，関数の中に関数を作る）は C 言語では特定のコンパイラでしか使うことができない．そのためこの授業では入れ子関数は使用しないこと．特に，関数は `main` の外に作ること．`main` の中に関数を作らないこと（`main` も関数の一種である）．



```

#include <stdio.h>
#include <math.h>

double f(double x); // プロトタイプ宣言．出力と引数の型のみを記述．

int main(){
    double y;
    double x1=1, x2=2, x3=3;
    y=f(x1)+f(x2)+f(x3);
    printf("y=%f\n",y);
    return 0; // int main() が正常終了
}

double f(double x){ // 関数の記述
    double y;
    y=sin(x) + 1.0/(x*x*x);
    return y;
}

```

関数を複数作る．引数が複数の関数を作る．

$f(x) = x^2$ ,  $g(x, y) = f(x + y) + y$  とする． $x = 1$ ,  $y = 2$  に対して  $g(x, y)$  を計算する．

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

double f(double x); // プロトタイプ宣言．出力と引数の型のみを記述．
double g(double x, double y); // プロトタイプ宣言．出力と引数の型のみを記述．

int main(){
    double x=1;
    double y=2;
    double z;
    z=g(x,y);
    printf("x=%f\n",x);
    printf("y=%f\n",y);
    printf("g(x,y)=%f\n",z);
    return 0; // int main() が正常終了
}

double f(double x){ // 関数の記述
    double y;
    y=x*x;
    return y;
}

double g(double x, double y){ // 関数の記述
    double z;
    z=f(x+y) + y;
    return z;
}
```

## 10.8 for 文と break

第 6 章などで用いる. 整数  $n$  に対し,  $e^n > 100$  を満たす最小の  $n$  を求める.

プログラムでは,  $n = 0, 1, 2, \dots$  に対して  $e^n$  を計算し,  $e^n > 100$  になったら計算を終了し,  $n$  と  $e^n$  の値を表示する.  $e^n$  はプログラムでは `exp(n)` と記述する.

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

int main(){
    int n;
    int maxn=100; // n の最大値を指定
    double y;
    for(n=0;n<=maxn;n++){
        y=exp(n);
        //printf("n=%d, exp(n)=%f\n",n,y); // 途中結果の表示
        if(y>100){
            break; // これで一番内側のループを終了する
        }
    }
    printf("n=%d\n",n); // 結果の表示
    printf("exp(n)=%f\n",y); // 結果の表示
    return 0; // int main() が正常終了
}
```

`break` は一番内側のループ (`for`, `while`, `do` など) のみを終了する. 2 重の `for` ループなどを終了するときにはいくつか方法があるが, この授業では扱わないので省略する.

途中結果を表示したいときは, 「途中結果の表示」と書かれた行の左端の `//` を消す. デバッグの際は, このように途中結果の表示が有効である.

## 10.9 1次元配列

これは第7章で用いる.

ベクトル  $A = [2, 5, 3]$  と定義し, ベクトル  $A$  の成分を  $a_k (0 \leq k < 3)$  と表す.

C 言語の配列の添え字 (インデックス. 上式の  $k$ ) は 0 から始まるので, 数式もそれにあわせていることに注意.

$$\sum_{k=0}^2 a_k$$

をプログラムで計算する.

以下のプログラムを実行し, 10 が表示されれば正しい.

```
#include <stdio.h>
#define N 3

int main(){
    double A[N]={2.0,5.0,3.0}; // 1次元配列の定義
    int k;
    double s=0; // 初期値として0を設定 (これが必要. for 文の直前を推奨)
    for(k=0;k<N;k++){
        s=s+A[k];
    }
    printf("sum of A=%f\n",s);
    return 0; // int main() が正常終了
}
```

## 10.10 for 文を作るとき考え方

- for の{...}の中に何をに入れるか？アルゴリズムでどこからどこまでを繰り返すのかを把握して、それが for の{...}の中に入っているかを確認する。繰り返さないところは、for の{...}の中に存在してはいけない。
- 問題文に、 $i = k + 1, \dots, n - 1$ ,  $k = 0, \dots, n - 1$  などとあったら、それぞれの for 文が必要であることがわかるはず。また、 $i$  の初期値は、 $k$  が決まらないとわからないので、

```
for(i=k+1;i<n;i++){ // i=k+1 が計算できない (k の値が下の行で決まる)
    for(k=0;k<n;k++){
        ...
```

は、間違いであることがわかるはず。

```
for(k=0;k<n;k++){
    for(i=k+1;i<n;i++){ // k の値は上の for 文で決まるので、i=k+1 が計算できる
        ...
```

が正しい。

- $A[i][j]$  などを用いる場合に、 $i$  や  $j$  の値が決まっているかチェックする。 $i$  や  $j$  を for 文で変化させる場合、その for の{...}の中に  $A[i][j]$  が存在する必要がある。(一つ上の項目とも関係している)

```
for(i=0;i<n;i++){
    A[i][j]=... // j の値は次の行で決まるので、間違い
    for(j=0;j<n;j++){
        ...
```

```
for(i=0;i<n;i++){
    for(j=0;j<n;j++){
        A[i][j]=... // i,j の値は決まっているので正しい
        ...
```

## 10.11 2次元配列

これは第7章で用いる.

$n = 2$ ,  $m = 3$  とし,  $n \times m$  のサイズの行列  $A$  を以下で定義する.

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

行列  $A$  の成分を  $a_{ij}$  ( $0 \leq i < n, 0 \leq j < m$ ) とする. C 言語の配列の添え字 (インデックス. 上式の  $i, j$ ) は 0 から始まることに注意.

$$b_i = \sum_{j=0}^{m-1} a_{ij} \quad (i = 0, 1, \dots, n-1)$$

を計算し,  $b_i (i = 0, 1, \dots, n-1)$  を表示する.  $B[0]=6, B[1]=15$  であれば正しい.

```
#include <stdio.h>
#define N 2
#define M 3
int main(){
    double A[N][M]={1.0, 2.0, 3.0},{4.0,5.0,6.0}};
    double B[N];
    int i,j;
    double s;

    for(i=0;i<N;i++){
        s=0; // Σの for 文の直前に置く (推奨)
        for(j=0;j<M;j++){
            s=s+A[i][j];
        }
        B[i]=s;
    }

    for(i=0;i<N;i++){
        printf("B[%d]=%f\n",i,B[i]);
    }

    return 0; // int main() が正常終了
}
```

## 10.12 ファイルの読み書き

第8章で用いる.  $k = 1, 2, \dots, 100$  に対して,  $x = \frac{1}{k^2}$  を計算し,  $k$  の値と  $x$  の値を `datafile.txt` というファイルに書き込む

```
#include <stdio.h>

int main(){
    FILE *fp;
    fp = fopen("datafile.txt","w"); // w は書き込みモード
    int k,n;
    double x;
    n=100;

    for(k=1;k<=n;k++){
        x=1.0/(k*k);
        fprintf(fp,"%d %e\n",k,x);
    }
    fclose(fp);
    return 0; // int main() が正常終了
}
```

コンパイルして実行した後, `more datafile.txt` とすると, 書き込まれたデータを見ることができる. スペースキーを押すと続きを見ることができる. 一番下の行の値が, `100 1.000000e-04` であれば正しい.

作成したファイル `datafile.txt` を読んで表示するプログラムは以下になる. ただし, 第8章ではデータは OpenOffice で読むので, このプログラムは不要.

```

#include <stdio.h>
int main(){
    FILE *fp;
    fp = fopen("datafile.txt","r"); // r は読み込みモード
    int k;
    double x;

    while(fscanf(fp,"%d %lf",&k,&x) != EOF){ // データを読む. %lf に注意
        printf("k=%d x=%e\n",k,x); // 読んだデータを表示
    }
    fclose(fp);
    return 0; // int main() が正常終了
}

```

実行すると一番下に k=100 x=1.000000e-04 と表示されれば正しい.

### 10.13 sizeof 演算子

sizeof を使うと, float, double, int などのメモリサイズ (バイト数) がわかる.

```

#include <stdio.h>
int main(){
    int s_int=sizeof(int);
    int s_float=sizeof(float);
    int s_double=sizeof(double);
    printf("int = %d bytes\n", s_int);
    printf("float = %d bytes\n", s_float);
    printf("double = %d bytes\n", s_double);
    return 0;
}

```

上のプログラムを実行すると各変数のメモリサイズがわかる. このメモリサイズはシステムによって異なることがある. 授業で使用している Linux サーバでは, int は 4 バイト=32 ビット, float は 4 バイト=32 ビット, double は 8 バイト=64 ビット (=float の倍) であることがわかる.



## 11 プログラミングについて

### 11.1 C 言語の要点

- プログラムの基本形

```
#include <stdio.h>
#include ..... // math.h, stdlib.h などをインクルード

double func1(double x, int n); // 関数 func1 のプロトタイプ宣言（本体は下）
int func2(float y); // 関数 func2 のプロトタイプ宣言（本体は下）

int main(){ // メイン
    ....
    return 0; // int main() が正常終了
}

double func1(double x, int n){ // 関数 func1 の本体
    double z=.....
    return z;
}

int func2(float y){ // 関数 func2 の本体
    int z=.....
    return z;
}
```

- コンパイル

プログラムを書いたら、コンパイルする。Linux でのコンパイルは次のようにする。

ソースファイル名は例として file1.c とするが、適宜変更せよ。

- コンパイル：gcc file1.c
- 実行：./a.out

gcc はオプションがなければ実行ファイルは a.out となる。または、-o オプションを使って実行ファイル名を指定することができる。ここでは実行ファイル名は file1 とする。実行ファイル名とソースファイル名とは違う名前にすること。

- コンパイル（実行ファイル名指定あり）：gcc -o file1 file1.c
- 実行：./file1

- プログラムを他の人から見られないようにする (Linux).

```
>> chmod g-r プログラムファイル名
>> chmod o-r プログラムファイル名
>> ls -l プログラムファイル名
```

最後の結果が「-rw----- プログラムファイル名」となっていれば OK. もし, 上記の作業をせずにプログラムを他の人に見られてコピーされた場合, プログラムを他の人に見せたと判断されて減点になるので注意せよ.

- printf 文

```
int n=10;
double x=0.1;

printf("n=%d, x=%lf\n", n,x); // 整数と浮動小数点の表示. \n は改行
printf("x=%e\n", x); // 浮動小数点の表示 (指数形式)
printf("x=%.20e\n", x); // 小数点以下 20 桁で浮動小数点表示
```

- if 文

```
double x=0, y=1;
...
if(x==y){ // x と y が等しいとき
    ...
}else if(x<y){ // x<y のとき
    ...
}else{
    ...
}
```

- かつ: and(&&), または: or(||), 否定: not(!)

```
double x=...
if(0 <= x && x <= 1){ // 0 ≤ x ≤ 1 のとき
    ....
}else if(x != -1){ // x ≠ -1 のとき
    ....
}else if(x< -2 || x>2){ // x<-2 または x>2 のとき
    ...
}
```

- for 文

```
int k, n=100;
for(k=0;k<n;k++){ // k=0,1,...,n-1
    ....
}
for(k=n-1;k>=0;k--){ // k=n-1,n-2,...,0
    ....
}
```

- for 文 (2 重ループ)

```
int i,j;
for(i=0;i<n;i++){ // i=0,1,...,n-1
    for(j=i;j<n;j++){ // j=i,i+1,...,n-1
        ...
    }
}
```

- while 文

```
int k=0, n=100;
while(k<n){ // k<n ならループを続ける
    ...
    k++;
}
```

- break

```
int k=0, n=100;
while(1){ // 無限ループ
    ....
    k++;
    if(k>=n){
        break; // k>=n になったらループ（一番内側のループのみ）から抜ける
    }
}
```

- 関数

```
#include .....  
double func(double x, int n); // プロトタイプ宣言  
                                // 引数は実数 1 つ (x) 整数 1 つ (n), 戻り値は実数  
  
int main(){  
    ....  
    double x=...  
    int n=...  
    double z=func(x,n);  
    ...  
}  
  
double func(double x, int n){ // 戻り値は実数  
    double z=.... x .... n.... // 戻り値の計算  
    return z; // 戻り値を返す  
}
```

- 配列 (1 次元)

```
#define N 4 // 配列の要素数  
int main(){  
    double A[N]={1.0, 3.0, 5.0, 7.0}; // 配列の定義  
    double z=0;  
    int k;  
    for(k=0;k<N;k++){  
        z=z+A[k]; // 配列の要素の値を用いる  
        A[k]=z; // 配列の要素に値を代入する  
    }  
    ...  
}
```

- 配列 (2 次元)

```
#define N 2
#define M 3
int main(){
    double A[N][M]={1.0, 2.0, 3.0}, {4.0, 5.0, 6.0}};
    double z=0.0;
    int i,j;
    for(i=0;i<N;i++){
        for(j=0;j<M;j++){
            z=z+A[i][j]; // 配列の要素の値を用いる
        }
    }
    ....
}
```

- インデント (字下げ) プログラムを読むのに重要. 中括弧 { } の対応がすぐわかるようにする. インデントができない人は, プログラムが理解できていません.  
以下のサンプルで, 各行の ... の開始位置に注目すること.

```
for(...){
    ..... // インデントによって for 文の中であることがすぐわかる
    .....
    if(...){
        ..... // インデントによって if 文の中であることがすぐわかる
        .....
    } // if 文の終わり (インデントによって始まりと終わりがすぐわかる)
    ..... // インデントによって if 文の外で, for 文の中であることがすぐわかる
} // for 文の終わり (インデントによって始まりと終わりがすぐわかる)
```

- ファイルの読み書き

```
FILE *fp_read=fopen("input.txt","r"); // input.txt を読むモードで開く
FILE *fp_write=fopen("output.txt","w"); // output.txt を書くモードで開く

double a;
while(fscanf(fp_read,"%lf",&a) != EOF){ // input.txt を最後まで読むループ
    double b=2*a; // 読んだデータを2倍する
    fprintf(fp_write,"%lf\n",b); // 計算したbをoutput.txtに書き込む
}

fclose(fp_read);
fclose(fp_write);
```

- 基本的数学関数 (math.h) の使い方

```
#include <math.h>
int main(){
    double x=2.0;
    double y1=sin(x);
    double y2=tan(x);
    double y3=log(x);
    double y4=exp(x); // y4=e^x
    double y5=fabs(x); // y5=|x| y5はxの絶対値
    double y6=atan(x); // atanはtanの逆関数
    double y7=sqrt(x); // y7=√x
}
```

コンパイル (Linux) : コンパイル時に以下のように `-lm` オプションをつける.

```
cc ソースファイル名 -lm
```

## 11.2 gcc のオプション

gcc のオプションでよく使用するものを説明する.

- `-lm`: `math.h` を使うときはこのオプションが必要 (Linux)
- `-o`: 実行ファイルを `a.out` 以外にする場合  
例: `cc -o ex1 ex1.c`  
`ex1.c` をコンパイルして, `ex1` というファイルを作成. 実行時は `./ex1` とする.
- `-Wall`: いろいろな注意が出る. 例えば「使用されていない変数」と出てきた場合には, その変数を削除する.
- `-O -Wall`: いろいろな注意が出る (`0` は数字ではなく大文字). 上の注意に加えて, 初期化されていない変数がある場合に「初期化されずに使用」と出てくるので, 初期化するようにする. `-O` は最適化のオプションであるが, 初期化について表示するにはこのオプションが必要.

`-Wall` または `-O -Wall` を使用して修正することを推奨する.

## 11.3 プログラムのデバッグ（修正）の基本

プログラムを書いても、それだけでは正常に動かないのが普通です。以下を参考にして、プログラムが動くようになるまで根気よく修正を続けましょう。しばらく修正を繰り返していると慣れてくるので、修正にかかる時間はどんどん短くなっていきます。

### 11.3.1 エラーが出てコンパイルできない

コンパイルできないときは以下に注意すること。

- エラーメッセージをよく読むこと。意味が分からなくてもエラーの行数やキーワードはわかるはず。それをもとにエラーを探す。また、はじめはエラーメッセージの意味が分からなくても、毎回読んでいると「以前と同じメッセージ」であることがわかるので、以前と同じ対処法を使えばよいことになる。
- 行末の ; が抜けていないか
- スペルミスはないか。関数名は正しいか。% と & を間違えていないかなど。
- 変数名などで重複はないか。

### 11.3.2 何か値は出力されたが、変な値が出力される。正しいかわからない

- 出てきた答えが正しいかを考える。どうすれば正しいことが確かめられるか？  
例：  $f(x) = 0$  の解  $x_0$  を求めた。これが正しいことはどうすればわかる？
- 変数の初期値は設定されているか。初期値を設定せずに計算すると変な値になる。
- 0 での割り算はないか。
- if 文の条件で = と == を間違えていないか。  
if(a==2){...} は正しい。if(a=2){...} は間違っている。a=2 は a に 2 を代入する、ということ。
- カッコの付け方は正しいか。  $\frac{1}{2a}$  を計算する場合。  
 $1/(2*a)$  は正しい。  $1/2*a$  は間違っている。
- 実数の計算と、整数の計算を理解する。  
 $1/3$  は、整数の計算なので、結果は 0 になる。  
 $1.0/3.0$  は実数の計算となる。
- printf で %d, %e など間違えていると正しく表示されないので注意せよ。
- printf を使っているいろいろな値を表示せよ。上記のバグもこれでかなりわかる。
  - ループの中で計算途中の値を表示せよ。
  - if 文の直前で、if 文の条件 if(...) の括弧の中の値を表示せよ。すなわち括弧の中の変数や式を表示せよ。
  - while 文の中括弧 {...} の中の一番最後で、while 文の条件 while(...) の小括弧の中の値を表示せよ。すなわち小括弧の中の変数や式を表示せよ。
- 文字化けする。以下のようにして修正。  
GNOME エディタでファイルを開き、別名で保存→エンコーディング→現在のロケール (UTF-8) にチェックを入れて保存。



## 11.4 C 言語での注意点

ここでは C 言語を使う際に、よく間違える点について記述する。

- int, float, double の割り算に注意せよ。C 言語で

```
double a=1/2;
double b=1.0/2.0;
```

とすると、a, b の値はそれぞれ何になるか？

- 式  $\frac{a \cdot b}{c \cdot d}$  を計算したい。

```
double a=1.0, b=2.0, c=3.0, d=4.0;
double x=a*b/c*d;
double y=a*b/(c*d);
double z=a*b/c/d;
```

x, y, z のうち正しいものはどれか？（正しいものは複数あるかもしれない。）また、x, y, z のうち正しくないものを数式で表すとどうなるか？

わからなければ、以下を入力して実行する。

```
#include <stdio.h>
int main(){
    double a=1/2;
    double b=1.0/2.0;
    printf("a=%f b=%f\n",a,b);

    a=1.0; b=2.0;
    double c=3.0, d=4.0;
    double x=a*b/c*d;
    double y=a*b/(c*d);
    double z=a*b/c/d;
    printf("x=%f y=%f z=%f\n",x,y,z);
}
```

1/2 は 0 となることがわかる。1/2 は整数としての演算なので、割り算の商は 0 になる。1.0/2.0 は実数（浮動小数）としての演算なので、0.5 となる。

$a*b/c*d$  は  $\frac{ab}{c}d = \frac{abd}{c}$  を計算することになり、目的の式とは異なる。

- 計算式などの右辺の値が確定しているか調べる．以下のプログラムでは， $k=0$  のとき， $s=s+k$  の右辺の  $s$  の値が確定していないので間違い．

```
#include <stdio.h>
int main(){
    double s;
    int k;
    for (k=0;k<10;k++){
        s=s+k; // k=0 のとき，右辺の s の値は何？
    }
    return 0;
}
```

以下のように  $s$  の初期化が必要である．

```
#include <stdio.h>
int main(){
    double s;
    int k;
    s=0; // s の初期化が必要
    for (k=0;k<10;k++){
        s=s+k; // k=0 のとき，右辺の s の値は 0
    }
    return 0;
}
```

- for 文で変化する変数は，for 文の中で計算する． $i = 0, \dots, n-1$  に対して， $x_i = \frac{i}{10}$  として  $y_i = \sin(x_i)$  を求めたいとする．以下は  $x_i$  が  $i$  に応じて変化していないので間違いである．

```
#include <stdio.h>
#include <math.h>
int main(){
    int i=0, n=30;
    double xi,yi;
    xi=i/10.0;
    for (i=0;i<n;i++){
        yi=sin(xi); // xi は変化していない
        printf("%f\n",yi);
    }
    return 0;
}
```

次が正しい.

```
#include <stdio.h>
#include <math.h>
int main(){
    int i, n=30; // i の初期化は不要
    double xi,yi;
    for (i=0;i<n;i++){
        xi=i/10.0; // xi は i に応じて変化している
        yi=sin(xi);
        printf("%f\n",yi);
    }
    return 0;
}
```

- 次のプログラムでも,  $i=k+1$  の右辺の値が  $k$  によって正しく変化していないので間違っている.

```
#include <stdio.h>
int main(){
    int i,k,n=10;
    for (i=k+1;i<n;i++){ // k の値は何?どこで決まる?
        for (k=0;k<n;k++){ // k の値はここで決まる
            ...
        }
    }
    return 0;
}
```

通常は, 以下のように for 文の順序を逆にすると正しくなる.

```
#include <stdio.h>
int main(){
    int i,k,n=10;
    for (k=0;k<n;k++){ // k の値は決まる
        for (i=k+1;i<n;i++){ // k を用いた i の値も決まる
            ...
        }
    }
    return 0;
}
```

- scanf の使い方

```
float a;
scanf("%f", &a); // 1
scanf("%lf",&a); // 2
scanf("%d", &a); // 3

double b;
scanf("%f", &b); // 4
scanf("%lf",&b); // 5
scanf("%d", &b); // 6
```

1,2,3のうち正しいのはどれか。また、4,5,6のうち正しいのはどれか。

- 実数  $x$  に対し、 $\sin(x) + x$  を返す関数を作りたい。

```
double f(x){
    return sin(x)+x;
}

double g(double x){
    return sin(x)+x;
}
```

正しいのは f, g のどちらか？

上記の f の引数 x は int, float, double のいずれとも異なるものに解釈される（何に解釈されるか不明）。引数の型は省略してはならない。

- if 文で、a と b を比較して、処理をしたい。

```
a=... // aの計算式
b=... // bの計算式
if(a=b){ // 1
    ....
}

a=... // aの計算式
b=... // bの計算式
if(a==b){ // 2
    .....
}
```

正しいのは 1,2 のどちらか。

## 11.5 学生舎でのプログラミング：Windows に Linux 環境を構築 (WSL)

Windows に Linux 環境を構築する。Windows11 であれば、WSL が比較的簡単である。

1. インストール。MicrosoftStore で Ubuntu をインストール（ログインは不要のはず）  
下記を参照してインストール

<https://www.atmarkit.co.jp/ait/articles/1903/18/news031.html>

<https://www.atmarkit.co.jp/ait/articles/1904/19/news033.html>

2. プロキシの設定（アップデートに必要）

- (a) ホームディレクトリの .bashrc の最後に以下を追加

（たとえば nano .bashrc としてエディタ起動）

```
export http_proxy=http://cmproxy2.nda.ac.jp:9090
```

```
export https_proxy=http://cmproxy2.nda.ac.jp:9090
```

プロキシサーバーとポート番号は、現在の環境に合わせて変更すること。

Windows からファイルを編集しないこと。Ubuntu でエディタ (nano, vi など) を起動して編集すること。

- (b) ubuntu を再起動

- (c) コマンドラインで、printenv として、proxy の環境変数の設定が正しいことを確認

- (d) proxy を使うときは、sudo -E apt update などと -E オプションを使う。

- (e) proxy を使わないときは、sudo apt update とする（以下も同様）。

3. ファイルサーバーを日本にする。

```
cd /etc/apt
```

```
sudo cp sources.list sources.list_backup
```

```
sudo sed -i -e 's%http://.*ubuntu.com%http://ftp.jaist.ac.jp/pub/Linux%g' /etc/apt/sources.list
```

4. アップデート

```
sudo -E apt update
```

```
sudo -E apt upgrade
```

5. gcc (C コンパイラ) のインストール

```
sudo -E apt-get install gcc
```

6. エディタ nano でソースファイルを編集する。

7. 自分の PC に Linux 環境を構築する場合には (WSL 以外の、VMware, VirtualBox などと同様)、電算機室の C コンパイラとほぼ同様にするために、コンパイルの際に以下のオプションを使用すること。  
(互換性は完全ではないが、これでコンパイルして正しく動作すれば、電算機室のマシンでも正しく動作する確率が非常に高い)

```
gcc -std=gnu89 ソースファイル名
```

または

```
gcc -std=gnu90 ソースファイル名
```

このオプションを使うと

```
for(int i=...)
```

は使えなくなるので,

```
int i;  
for(i=...)
```

などとすること.

もし `math.h` をインクルードするときは, さらに `-lm` オプションが必要.

8. OpenOffice を使うときには, Windows 用の OpenOffice をインストールして, WSL から csv ファイルを Windows に転送して, OpenOffice で開く.
9. WSL のインストールが難しい場合には, MinGW もしくは CygWin をインストールする.

## 11.6 学生舎でのプログラミング:MinGW

前節で記述した WSL のインストールがどうしてもうまくできない場合には, 以下の URL を参考にして MinGW をインストールする. ただし, 授業の環境との互換性は低くなる.

<https://www.javadrive.jp/cstart/install/index6.html>

1. 授業で用いるコンパイラとは若干異なる点があるので, 提出時には MinGW を用いていることを明記する.
2. 電算機室の C コンパイラとなるべく近くするために, コンパイルの際に以下のオプションを使用すること. (互換性は完全ではないが, これでコンパイルして正しく動作すれば, 電算機室のマシンでも正しく動作する確率が非常に高い)

```
gcc -std=gnu89 ソースファイル名
```

または

```
gcc -std=gnu90 ソースファイル名
```

このオプションを使うと

```
for(int i=...)
```

は使えなくなるので,

```
int i;  
for(i=...)
```

などとすること.

もし `math.h` をインクルードするときは, さらに `-lm` オプションが必要.

3. OpenOffice を使うときには, Windows 用の OpenOffice をインストールして, csv ファイルを OpenOffice で開く.

## 11.7 Mathematica の利用法

式の計算などで Mathematica が便利なので、簡単な使い方を記述する。ただし授業では基本的には使用しない。

### 1. 基本的な事項

- (a) 実行は Shift を押しながら Enter を押す。
- (b) 大文字・小文字の区別がある。(例：`a=Sin[Pi/2]`)
- (c) スペースで隔てられた変数は掛け算である。(例：`a=b c`)
- (d) 基本的演算：`a+b`, `a-b`, `a*b` (または `a b`), `a/b`, `a^b`, `a!`, `Mod[a,b]`, `Quotient[a,b]`
- (e) 関数：`Sqrt[x]`, `Cos[x]`, `Sin[x]`, `Tan[x]`, `Log[x]`, `Exp[x]`, `Sum[f[k],{k,a,b}]` ( $\sum_{k=a}^b f(k)$  を求める), `Cos[3]`, `N[Cos[3]]`
- (f) 使用している変数の値は、ずっと保存されている。値をリセットする場合には `Clear[変数名]` とする。

### 2. 高度な関数が入っている。

- (a) 1234567 は素数か? `PrimeQ[1234567]`
- (b) 48 を素因数分解する。 `FactorInteger[48]` とする。答えは  $2^4 \times 3$
- (c) 複素数  $(1+i)/(1-i)$  の絶対値と偏角

```
z=(1+I)/(1-I)
Abs[z]
Arg[z]
```

### 3. ベクトル, 行列

- (a) ベクトル：`a={0,1,1}`; `b={1,2,3}`; `c=a.b` とすると  $a, b$  の内積  $c$  が求まる。
- (b) 行列とベクトル：`a={{1,1},{2,1}}`; `b={2,3}` とする。積：`a.b`, 逆行列：`Inverse[a]`, 行列式：`Det[a]` で求められる。
- (c) `Clear[a,b,c,d,x]`; `x={{a,b},{c,d}}`; `Det[x]` とすると、文字式で行列式を求めることができる。

### 4. 式を解く

- (a) 方程式  $x^2 - x - 1 = 0$  を解く。 `Solve[x^2-x-1==0,x]`
- (b) 連立方程式を解く。  $3x-2y=10$ ,  $5x-4y=2$  を解く。 `Solve[{3*x-2*y==10, 5*x-4*y==2},{x,y}]`
- (c) その他：`Solve[Sin[2*x]==1,x]`  
`Clear[a,b,x,y,w]`; `a={{3,-2},{5,-4}}`; `b={10,2}`; `w={x,y}`; `Solve[a.w==b,w]`

### 5. 関数の定義：`f[x_]:=x^2 + Log[x]` とする。実行は `f[2]` や `f[2.]` などとする。

### 6. 微分, 積分

- (a) 微分：`D[Sin[x+Tan[x]],x]`
- (b) 偏微分：`f[x_,y_]:=x*Log[x]`; `D[f[x,y],x]`
- (c) 全微分：`Dt[a*x + b, x]` や `Dt[x*y,x,y]` など。
- (d) 積分： $\int \frac{1}{(x+1)(x-2)} dx$  の計算。 `Integrate[1/((x+1)*(x-2)),x]`

(e) 数値積分:  $\int_0^1 x^x dx$  の計算. `NIntegrate[x^x,{x,0,1}]`

7. 図の描画

(a) 2次元グラフ:  $\sin(x)$  の描画. `Plot[Sin[x],{x,-Pi,Pi}]`

(b) 3次元グラフ:  $\exp(-x^2 - y^2)$  の描画. `Plot3D[Exp[-x^2-y^2],{x,-2,2},{y,-2,2}]`