
工学系のための偏微分方程式 数値解法の C 言語による実習

熊本大学理学部 小出眞路

本演習書は小出眞路著「工学系のための偏微分方程式」（森北出版，2006）の付録として公開するものである。

ここでは C 言語の基本的知識を前提とする． C 言語の基本的知識とは次の事柄を言う．

- 変数の取り扱い：型，定義，代入，演算（加減乗除・剰余（余り））
- 数値の出力
- プログラムの制御（反復，選択）
- 配列
- 関数 （関数の引数として配列を用いること）

C 言語でよく用いられるポインタはここでは必要ない．

ここではコンピュータの具体的操作方法として主に UNIX を例に示している．他の OS を使う場合はそれぞれコマンドを置き換えて読んでほしい．

目 次

付 録 A 数値解法の C 言語による実習	3
A.1 課程 1: 拡散方程式の数値解法	3
A.2 課程 2: 1 階線形波動方程式の数値解法	8
A.3 課程 3: 2 階線形波動方程式の数値解法	9
A.4 課程 4: 2 段階ラックス・ヴェンドロフ法	13
A.5 課程 5: 非線形方程式<流体力学方程式>の数値解法	16
付 録 B 2 次元流体力学方程式の数値解法	18

付録 A

数値解法の C 言語による実習

ここでは「工学系のための偏微分方程式」(小出眞路著, 森北出版)の第6章で示した数値計算法の実習を行う。簡単な例題からはじめて最後には非線形偏微分方程式とくに流体力学方程式を解くまでを扱う。第6章の各節の説明を読み実際にプログラムを組みながら実習を行ってほしい。本を読んでいるだけでは分からないことが理解されるはずである。ここで間違いやすいところについては[注意]で注意を促し、プログラムを組む際の細かいコツは[コツ]に書いた。

A.1 課程 1 : 拡散方程式の数値解法

課題 1 では拡散問題を数値的に解いてみよう。まず、ここで扱う数学モデルを示す(第1章例 1.9, 第2章第6節例題 2.3 参照)。

数学モデル :

- (i) 偏微分方程式 $\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} \quad (0 < x < 1, 0 < t < \infty)$
- (ii) 境界条件 $u(0, t) = u(1, t) = 0 \quad (0 < t < \infty)$
- (iii) 初期条件 $u(x, 0) = \sin \pi x \quad (0 \leq x \leq 1)$

この数学モデルを数値計算で解くスキームは次のようになる。記号の意味などは第6章のものをそのまま引き継ぐ。ここでは虚数単位 i を使うことはないので、メッシュの番号として i を用いることにする。また、メッシュ数を I と記す。

差分スキーム :

- (i) 差分方程式 $u_i^{n+1} = u_i^n + \frac{\delta}{h}(u_{i+1}^n - 2u_i^n + u_{i-1}^n) \quad (n = 0, 1, 2, \dots, i = 1, 2, \dots, I-1)$
- (ii) 境界条件 $u_0^n = u_I^n = 0 \quad (n = 0, 1, 2, \dots)$
- (iii) 初期条件 $u_i^0 = \sin \pi x_i \quad (i = 0, 1, 2, \dots, I)$

このスキームに基づいて拡散問題を数値的に解くことになるが、これらの式を見ただけで数値計算のプログラム(「コード」という)を作り上げるのは初心者には難しい。しかし、次から示すような手順を踏んでゆけば比較的簡単にコードが作成できる。

手順1 <変数 x_i の設定>

変数 x_i に対応する変数配列 $x[i]$ を定義・設定し表示する。これは変数の定義、四則演算、反復計算、配列などC言語の基本を使えばすぐにプログラムを作ることができる。ここで I は `imax` と置く。例となるCプログラムを示す。

```
#include <stdio.h>

#define imax 20

main(){
    double x[imax+1],dx;
    int i;

    dx = (double) 1.0/imax;

    for(i=0;i<=imax;i++)
        x[i]=(double) dx*i;
    for(i=0;i<=imax;i++)
        printf("%f \n",x[i]);
}
```

このプログラムに `diffu.c` という名前を付ける。このプログラムをコンパイル・実行すると21行の数値の並びが出力されるはずである。このデータを目で眺めて確認することもできるが、可視化するようにしておくとの後の変化の複雑な数値を扱うときに助かる。ここでは、データをリダイレクトで `data` と名付けたファイルに収め、それをデータ図形処理フリーソフト `gnuplot` を用いてプロットする。ここで `gcc` というCコンパイラを利用する。

```
% gcc diffu.c
% a.out > data
% gnuplot
gnuplot> plot "data" with line
```

これでデータが図形として表示されるはずである。`gnuplot` を終了するには、

```
gnuplot> exit
```

とすればよい。もちろん、他のデータ図形処理ソフトを用いてもよい。

手順2 < u_i^n の初期条件の設定>

次に、 u_i^n の初期条件を設定する。ここでこの変数を `uu[i]` として定義する。まず、`uu[i]` を宣言する。

```
double x[imax+1],uu[imax+1],dx;
```

`uu[i]` の初期値は `x[i]` の値を設定した後に次の文を入れる。

```
for(i=0;i<=imax;i++)
    uu[i] = sin(x[i]*3.14159265);
```

ここで、円周率 π の値を式の中に入れてくなければプログラムのはじめで変数 PI を宣言しておいてもよい。変数 uu[i] の値を出力する。

```
for(i=0;i<=imax;i++)
    printf("%f %f \n",x[i],uu[i]);
```

このプログラムをコンパイルするときには注意すべきことは C 言語本体は三角関数などの繰り返し関数を持っていないことである。そこで数学ライブラリを読み込んでコンパイルする必要がある。具体的には次のような 1 行をプログラム diffu.c の先頭付近 (2 行目) に入れなくてはならない。

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

コンパイルするときは次のようにする。

```
% gcc -lm diffu.c
```

手順 3 < 1 ステップだけ進める >

これで初期条件の設定が終わった。次に 1 ステップだけ計算を進めて (時間発展させて) みよう。新しいステップの変数 u には新しい変数 un[i] を用いることにする。1 ステップ進めるには差分方程式 (i) を用いて計算すればよい。変更する箇所は次のようである。

まず, un[i] を宣言する。また, 時間幅 dt (= s) も定義する。(ついでに κ に相当する変数 kap の利用の宣言もしておく。)

```
double x[imax+1],uu[imax+1],un[imax+1],dx;
double dt=0.001, kap;
```

初期条件を設定した繰り返し文の後に時間発展させる差分式を計算する繰り返し文を入れる。

```
kap = dt/(dx*dx);
for(i=1;i<imax;i++)
    un[i]=uu[i]+kap*(uu[i+1]-2.0*uu[i]+uu[i-1]);
```

また, un[i] の端の値が定義されていないのでこれに続いて境界条件に基づき次のようにする。

```
un[0]=0.0;
un[imax]=0.0;
```

また, 最後に uu[i] を出力するように un を uu で置き換える。そして, それを出力する printf 文を加える。

```
for(i=0;i<=imax;i++)
    uu[i]=un[i];
for(i=0;i<=imax;i++)
    printf("%f %f \n",x[i],uu[i]);
```

これでプログラムをコンパイル・実行させると少し異なった線が増えているはずである。これが第1ステップ目（時刻 $t = s$ ）の u_i^1 ($i = 0, 1, 2, \dots, I$) を表す。

手順4 <時間発展させる>

1ステップ進んだので、次は $nmax=7$ まで進めてみよう。これは時間発展の文からその数値を出力する文までのところを繰り返すようにすればよい。

```
for(n=1;n<=nmax;n++){
    for(i=1;i<imax;i++){
        un[i]=uu[i]+kap*(uu[i+1]-2.0*uu[i]+uu[i-1]);
    }
    for(i=0;i<=imax;i++){
        uu[i]=un[i];
    }
    for(i=0;i<=imax;i++){
        printf("%f %f \n",x[i],uu[i]);
    }
}
```

ここで当然、 n の使用の宣言と $nmax=7$ の定義を忘れてはならない。

```
double x[imax+1],uu[imax+1],un[imax+1],dx;
double dt=0.001, kap;
int i,n;
int nmax=7;
```

これでプログラムをコンパイル・実行しデータをプロットすると8本の線が描かれるはずである。初期の値の三角関数の振幅よりも、小さい振幅の三角関数の線があるとそれが時間が $t=dt*nmax=0.007$ だけ進んだときの u を表す。少し時刻は違っているが図6.8を参照のこと。

試してみよう① 解析解との比較

次に解析解と比較するためにもう少し長く計算してみよう。 $nmax=70$ とすると、時刻が $t = 0.07$ だけ経っていることになるので、解析解によると三角関数の振幅は $e^{-0.07\pi^2} = 1/1.995 \sim 0.5$ になるはずである。ほとんど、そうなっていることを確かめよ。多少違っていることがあれば、それは数値誤差による。しかし、その数値誤差は小さく、十分解析解に近い数値解が得られるはずである。ここで振幅が0.5になっているか分かりにくい場合は、例えば次のような数字をデータファイル (**data**) に直接書き加えて点 $x = 0$, $y = 0.5$ と点 $x = 1$, $y = 0.5$ を結ぶ線を書かせるようにするとよい。

```
1.0  0.5
0.0  0.5
```

試してみよう② 第4章例題4.3に対応する計算（シルクハット型関数の初期条件）

第II部第4章の例題4.3で示した計算を行ってみよう。このときの解析解は例題4.3で求めた解となる。この問題を解くには単に初期条件を変えるだけでよい。具体的には手順2で設定した初期条件のところを次のように書き換えてやるとよい。

```

if(fabs(x[i]-0.5) <= 1.0/6.0)
    uu[i]=1.0;
else uu[i]=0.0;

```

プログラムをコンパイル・実行しプロットしてみると解析解とよく一致していることが確認できるであろう (図 6. 4 参照)。

試してみよう③ 数値不安定性

これで線形拡散方程式を数値的に解くプログラムが出来たことになる。ここで数値不安定性がどんなものかを試してみよう。今までは、 $\kappa = cs/h^2 = 1 \times 0.001/0.05^2 = 0.4 < 0.5$ であり、計算は安定であった。ここで、 $dt=0.002$ ($s=0.002$) として $\kappa = 0.8$ の場合を計算してみよう。nmax=70 のままでは計算できないので、nmax=3 としたほうが良い。すると最後の u はギザギザした分布となるはずである (図 6. 5 参照)。ギザギザはノイマンの方法で予想される最も危険なモードで説明できる。その振幅は初期の振幅を超え、計算そのものを壊しているのが分かる。数値不安定性は数値安定条件 (CFL 条件) が満たされないと十数ステップで計算全体を壊してしまうことが分かるであろう。数値不安定性は数値計算上絶対に避けなくてはならない注意事項のひとつである。dt=0.001, nmax=70 にして計算を元に戻そう。

試してみよう④ 他の初期条件 (魔女の帽子型関数) での計算

次に、初期条件として

$$u(x, 0) = \Lambda(x) = \begin{cases} 1 - 4|x - \frac{1}{2}| & (|x - \frac{1}{2}| \leq \frac{1}{4}) \\ 0 & (\frac{1}{4} < |x - \frac{1}{2}| \leq \frac{1}{2}) \end{cases}$$

の場合を計算してみよう。このとき解析解は第 II 部第 4 章例題 4. 2 のところで求めた解になる (図 4. 6 参照)。数値的にこの問題を解くには単に初期条件を変えてやるだけでよい。具体的には手順 2 で設定した初期条件のところを次のように書き直してやるとよい。

```

if( fabs(x[i]-0.5) <= 0.25 )
    uu[i]=1.0 - 4.0*fabs(x[i]-0.5);
else uu[i]=0.0;

```

プログラムをコンパイル・実行しプロットしてみると解析解とよく一致することが確認できるであろう。

以上 4 つの実習で分かるように数値計算による解法は解析的解法と違い扱える問題に基本的に制限がなく適用範囲が非常に広い。一方、誤差や数値不安定性など数値計算の設定や得られた解の解釈を慎重に行う必要がある。

手順 5 <出力データの間引き>

ここで計算のメッシュ数 (imax) が大きくなると dt を小さくしなくてはならず、図形出力の線が混み合ってきて見にくくなる。そこで出力するデータをステップを nskip = nmax/10 毎に出力し、出力を 10 個程度の時間ステップの間引く。これをするには nskip を初期条件の設定の前あたりで定義・設定し、各ステップ毎にデータを出力するところを次のように if 文で囲めばよい。

```

if( n%nskip == 0 )
    printf("%f, %f \n", x[i], uu[i]);

```

これで $n_{\max} = 1,000$ くらいの長い時間を計算してみよう。ほとんど最終状態 ($t = 10$) では拡散し終わって u がゼロになることが見て取れるはずである。

うまく計算できないときはうまく計算できた手順まで戻ってその手順からもう一度はじめると良い。そのためにもうまく計算できた手順のプログラムの小まめに保存すると良い。

A.2 課程 2 : 1 階線形波動方程式の数値解法

次に波動方程式の数値計算に入ってみよう。いきなり 2 階線形波動方程式を扱う前に 1 階線形波動方程式 $\frac{\partial u}{\partial t} = -\frac{\partial u}{\partial x}$ の数値計算の実習を行う。ここでまず扱う数学的モデルを示す。

- (i) 偏微分方程式 $\frac{\partial u}{\partial t} = -\frac{\partial u}{\partial x} \quad (0 < x < 1, 0 < t)$
- (ii) 境界条件 $u(0, t) = u(1, t) = 0 \quad (0 < t)$
- (iii) 初期条件 $u(x, 0) = \Lambda(x) \equiv \begin{cases} 1 - 4|x - \frac{1}{2}| & (\frac{1}{4} \leq x \leq \frac{3}{4}) \\ 0 & (0 \leq x < \frac{1}{4}, \frac{3}{4} < x \leq 1) \end{cases}$

このモデルの解析解は $u(x, t) = \Lambda(x - t)$ である。(ただし、 u がゼロでないところが境界に達するまで、 $t \leq 0.25$) この数学モデルをラックス・ヴェンドロフ法により差分化すると次のように書ける。記号の意味は前の課程のものを引き継ぐ。

差分モデル

- (i) 差分方程式 $u_i^{n+1} = u_i^n - \frac{s}{2h}(u_{i+1}^n - u_{i-1}^n) + \frac{s^2}{2h^2}(u_{i+1}^n - 2u_i^n + u_{i-1}^n)$
 $(i = 1, 2, \dots, I - 1, n = 0, 1, 2, \dots)$
- (ii) 境界条件 $u_i^n = u_I^n = 0 \quad (n = 0, 1, 2, \dots)$
- (iii) 初期条件 $u_i^0 = \begin{cases} 1 - 4|x_i - \frac{1}{2}| & (\frac{1}{4} \leq x_i \leq \frac{3}{4}) \\ 0 & (0 \leq x_i < \frac{1}{4}, \frac{3}{4} < x_i \leq 1) \end{cases}$

手順 1 <差分式の書き換え>

差分モデルの式 (i),(ii),(iii) と前の課程 1 の試してみよう②の対応する条件は境界条件と初期条件は同じなので、違いは差分方程式 (i) のみである。すなわち、前の課程の最後に作ったプログラムの時間発展させる部分を差分モデルの式 (i) に変えるだけでよい。具体的には時間発展させる文を次のように変更すればよい。kap の定義をここでは $s/h = dt/dx$ に変える。時間発展させる差分式は

```

for(i=1; i<imax;i++)
    un[i]=uu[i] - 0.5*kap*(uu[i+1] - uu[i-1])
    + 0.5*kap*kap*(uu[i+1] - 2.0*uu[i] + uu[i-1]);

```

とできる。ここで前のプログラムをコピーして作った新しいプログラムの名前は wave1.c としておく。まず、 $dt = 0.01$, $n_{\max} = 10$, $imax = 50$ として計算してみよう。ここで最終時刻は $t = dt * n_{\max} = 10 \times 0.01 = 0.1$ である。時刻 $t = 0.1$ での解析解は $u(x, t) = \Lambda(x, t - 0.1)$ なので、右に 0.1 だけずれた点が見られるはず

である。確かめてみよう。ほとんど形は保っているが山の頂上付近がすこし滑らかにされているかもしれない。これが数値拡散と呼ばれるものである。

[コツ] 後のためにここでのプログラムを `wave2.c` として保存する。

```
cp wave1.c wave2.c
```

試してみよう① 数値不安定性

ここで数値不安定性についても体験してみよう。今は、 $\kappa = cs/h = 1 \times 0.01/0.02 = 0.5 < 1$ で数値安定になっている。ここで、 $s = dt = 0.03$ とすると $\kappa = 1.5$ となり数値安定性の条件が満たされなくなる。実際に計算してみると全く解析解とは関係のないようなギザギザの線が現れる。数値不安定性は計算を跡形もなく破壊してしまう絶対に避けなくてはならない事柄のひとつである。dt = 0.02 の場合 ($\kappa = 1$) もやってみよう。ここで初期値が数値拡散なしで幾何学的に右側にずれていくのか分かる。dt = 0.01 に戻して次に進む。

試してみよう② 矩形型初期条件

初期条件で次のような不連続な関数（シルクハット型関数）を設定してみよう。

$$u(x, 0) = \Omega(x) = \begin{cases} 1 & (\frac{1}{3} \leq x \leq \frac{2}{3}) \\ 0 & (\text{その他}) \end{cases}$$

プログラム中の具体的な書き方は課程 1 の手順 4 の試してみよう② の式を用いる。この初期条件で時間発展させると矩形は移動することはするがかなり形が崩れることが分かる。矩形の前方では振動現象が見られ、後方では数値拡散の影響が出ている。この現象はラックス・ヴェンドロフ法で衝撃波や固相と液相などの境界のある場合のように物理量が不連続関数となる数値計算において顕著に現れ、実用上最も厄介な問題である。このような不連続な関数を取り扱う方法はいろいろと開発されているが、差分法とは異なった CIP 法という数値計算法が最近注目されている¹。

A.3 課程 3: 2 階線形波動方程式の数値解法

いよいよ次に 2 階線形波動方程式 $\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2}$ を数値的に解いてみよう。まず、数学モデルを示す（第 6 章第 3 節例題 6.3 参照）。

(i) 偏微分方程式

$$\frac{\partial u}{\partial t} = -\frac{\partial v}{\partial x}, \quad \frac{\partial v}{\partial t} = -\frac{\partial u}{\partial x} \quad (0 < x < 1, \quad 0 < t < \infty)$$

(ii) 境界条件

$$u(0, t) = u(1, t) = 0, \quad v(0, t) = v(1, t) = 0 \quad (0 < t < \infty)$$

(iii) 初期条件

$$u(x, 0) = v(x, 0) = \Lambda(x) \equiv \begin{cases} 1 - 4|x - \frac{1}{2}| & (\frac{1}{4} \leq x \leq \frac{3}{4}) \\ 0 & (0 \leq x < \frac{1}{4}, \frac{3}{4} < x \leq 1) \end{cases}$$

^{*1} 矢部孝・内海隆行・尾形陽一、「CIP 法」（森北出版）

ここで差分方程式にしやすいように 2 元連立 1 階線形波動方程式にする. ベクトル $\mathbf{u} = \begin{pmatrix} u \\ v \end{pmatrix}$, $\mathbf{w} = \begin{pmatrix} v \\ u \end{pmatrix}$

を用いて書き直せば,

$$(i) \quad \text{偏微分方程式} \quad \frac{\partial \mathbf{u}}{\partial t} = -\frac{\partial \mathbf{w}}{\partial x} \quad (0 < x < 1, 0 < t < \infty)$$

$$(ii) \quad \text{境界条件} \quad \begin{pmatrix} u(0, t) \\ \frac{\partial v}{\partial x}(0, t) \end{pmatrix} = \begin{pmatrix} u(1, t) \\ \frac{\partial v}{\partial x}(1, t) \end{pmatrix} = \mathbf{0} \quad (0 < t < \infty)$$

$$(iii) \quad \text{初期条件} \quad \mathbf{u}(x, 0) = \begin{pmatrix} \Lambda(x) \\ \Lambda(x) \end{pmatrix} \quad (0 \leq x \leq 1)$$

と書ける. ただし, $\mathbf{0} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ である. ラックス・ヴェンドロフ法を用いて差分化した条件に書き直すと

$$(i) \quad \text{差分方程式} \quad \mathbf{u}_i^{n+1} = \mathbf{u}_i^n - \frac{\kappa}{2}(\mathbf{w}_{i+1}^n - \mathbf{w}_{i-1}^n) + \frac{\kappa^2}{2}(\mathbf{u}_{i+1}^n - 2\mathbf{u}_i^n + \mathbf{u}_{i-1}^n)$$

$$(n = 0, 1, 2, \dots, i = 1, 2, \dots, I - 1)$$

$$(ii) \quad \text{境界条件} \quad \begin{aligned} u_0^n &= u_I^n = 0 \\ v_0^n &= v_1^n, \quad v_I^n = v_{I-1}^n \end{aligned} \quad (n = 1, 2, \dots)$$

$$(iii) \quad \text{初期条件} \quad \mathbf{u}_i^0 = \begin{pmatrix} \Lambda(x_i) \\ \Lambda(x_i) \end{pmatrix} \quad (i = 0, 1, 2, \dots, I)$$

ここで差分式 (i) を成分で書くと

$$u_i^{n+1} = u_i^n - \frac{\kappa}{2}(v_{i+1}^n - v_{i-1}^n) + \frac{\kappa^2}{2}(u_{i+1}^n - 2u_i^n + u_{i-1}^n)$$

$$v_i^{n+1} = v_i^n - \frac{\kappa}{2}(u_{i+1}^n - u_{i-1}^n) + \frac{\kappa^2}{2}(v_{i+1}^n - 2v_i^n + v_{i-1}^n)$$

である.

手順 1 <変数の数 qmax を増やす>

これまで扱う物理量は 1 つだけであったが, 今回は u と v の 2 つになる. 今後, 非線形方程式のところで流体力学方程式を扱うようになると物理量は 3 つになるので, 一般に qmax 個の物理量を扱うようにする.

まずは同じ計算 (現在 1 階線形波動方程式) を qmax 個の変数について行うようにプログラムを変更する. ここで, 扱う変数は 2 次元配列を用いて, `uu[0][i]`, `uu[2][i]`, ..., `uu[qmax-1][i]` である. 具体的な手順は次のようにすればよい. ここで元となるプログラムは前の手順で **試してみよう①** をはじめる前のものを使うことが好ましい. 新しいプログラムの名前を `wave2.c` とでもしておこう. ここでは, `qmax = 2` として u と v はそれぞれ `uu[0][i]`, `uu[1][i]` に対応させる.

(1) 新しく用いる変数を宣言文に加える.

```
#define qmax 2
int q;
```

(2) 変数 `uu`, `un` を 2 次元配列に変える.

(宣言文)

```
double x[imax+1], uu[qmax][imax+1], un[qmax][imax+1], dx;
```

(初期値設定)

```
for(i=0;i<=imax;i++){
    if((0.25<=x[i]) && (x[i]<=0.75))
        uu[0][i] = 1.0 - 4.0*fabs(x[i]-0.5);
    else    uu[0][i]=0.0;
    uu[1][i] = uu[0][i];
}
```

(時間発展)

```
kap=dt/dx;
for(i=1;i<imax;i++){
    un[0][i] = uu[0][i] - 0.5*kap*(uu[0][i+1] - uu[0][i-1])
        + 0.5*kap*kap*(uu[0][i+1] -2.0*uu[0][i] + uu[0][i-1]);
    un[1][i] = uu[1][i] - 0.5*kap*(uu[1][i+1] - uu[1][i-1])
        + 0.5*kap*kap*(uu[1][i+1] -2.0*uu[1][i] + uu[1][i-1]);
}
```

(境界条件)

```
un[0][0]=0.0;
un[1][0]=un[1][1];
un[0][imax]=0.0;
un[1][imax]=un[1][imax-1];
```

(un と uu の置き換え)

```
for(i=0;i<=imax;i++)
    for(q=0;q<qmax;q++)
        uu[q][i] = un[q][i];
```

(データの出力)

```
if( n%nskip == 0 )
    for(i=0;i<=imax;i++)
        printf("%f, %f \n",x[i],uu[0][i]);
```

初期の出力の `uu` の添字も ‘`[0][i]`’ に変更する必要がある。

これは単に同じ計算を 2 (=qmax) 回しただけなので `uu[0][i]` と `uu[1][i]` の値は以前の `uu[i]` と全く変わらないはずである。このことを確かめよう。確かめるためには `uu[0][i]`, `uu[1][i]` をそれぞれ独立に出力し、それぞれの出力した先のファイルを `data1`, `data2` として、それらのファイルが一致していることを確認するのがよい。(図で見ただけでは完全に一致しているか保障はない。) UNIX 上ではファイル 1 とファイル 2 が一致しているかどうかをみるには `diff` というコマンドを使う。

```
unix% diff ファイル1 ファイル2
```

として何の出力 (応答) もなければ 2 つのファイルは一致していることになる。例えば以前の (`wave1.c` の実行結果の) データファイル名を `data1` とし、新しく得られたデータファイル名を `data2` とすれば

```
unix% diff data1 data2
```

として何の応答もなければ 2 つの結果は同じと確認できる。Windows ではコマンドプロンプトで `diff` の代わりに `fc` コマンドを用いる。

手順 2 < 2 階線形波動方程式の差分方程式への変更 >

次に実際に 2 階線形波動方程式の数値解法のプログラムに変更する。ラックス・ヴェンドロフ法の差分方程式を見ると、2 階の線形波動方程式と 1 階の波動方程式の違いは単に右辺第 2 項の u と v が入れ替わっているだけである。よって、プログラムでも対応するところを `uu[0][i]` と `uu[1][i]` を入れ替えてやるだけでよい。時間発展のところが次のようになる。

```
for(i=1;i<imax;i++){
    uu[0][i] = uu[0][i] - 0.5*kap*(uu[1][i+1] - uu[1][i-1])
              + 0.5*kap*kap*(uu[0][i+1] - 2.0*uu[0][i] + uu[0][i-1]);
    uu[1][i] = uu[1][i] - 0.5*kap*(uu[0][i+1] - uu[0][i-1])
              + 0.5*kap*kap*(uu[1][i+1] - 2.0*uu[1][i] + uu[1][i-1]);
}
```

ここでの変更は時間発展させる差分式の右辺の第 2 項で `uu` のはじめの添字 `0,1` について 4 箇所交換すればよいだけである。

このプログラムをコンパイル・実行してデータを見ると `uu[0][i]`, `uu[1][i]` 共に前と同様に右側に移動してゆくのが見られるはずである。数学モデルの解析解が

$$u(x, t) = \Lambda(x - t), \quad v(x, t) = \Lambda(x - t)$$

であるので、うまくこれが再現できていることになる。ここで、計算経過時刻 t と図形の移動距離から伝播速度を求め、その値が 1 になっていることを確かめよ (図 6.9 参照)。

試してみよう① 初期条件と波の伝播方向

ここで少し初期条件を変えてみよう。

(1) まず、`uu[1][i]` に対応する v の初期条件を

$$v(x, 0) = -\Lambda(x)$$

としてみよう. すると `uu[0][i]` のデータを見ると形が左側にずれて行くのが観察できるであろう. これは, この初期条件を満たす解析解が

$$u(x, t) = \Lambda(x + t), \quad v(x, t) = -\Lambda(x + t)$$

であることから理解できる.

(2) 次に, `uu[1][i]` に対応する v の初期条件を

$$v(x, 0) = 0$$

としてみよう. すると `uu[0][i]` の高さ 0.5 の山形が左右に分かれて伝播してゆくのが見られるはずである. これは, この初期条件を満たす解析解が

$$u(x, t) = \frac{1}{2}[\Lambda(x + t) + \Lambda(x - t)], \quad v(x, t) = \frac{1}{2}[\Lambda(x + t) - \Lambda(x - t)]$$

であることから理解できる.

試してみよう② 数値不安定性

初期条件を元に戻して, `dt = 0.03` としてみよう. この場合, $\kappa = 1.5$ となるので数値不安定が起こるはずである (図 6. 10 参照). 数値不安定が起こることを確かめたら, また `dt = 0.01` に戻しておこう. (ついでに `dt=0.02` の場合も見てみよう)

A.4 課程 4: 2 段階ラックス・ヴェンドロフ法

これまでの課程で線形波動方程式をラックス・ヴェンドロフ法を用いて計算してきた. 次に非線形の計算をするために 2 段階ラックス・ヴェンドロフ法を用いたプログラムに書き換えてゆく. ただし, この課程では今までの計算を 2 段階ラックス・ヴェンドロフ法を用いて行うだけで, 得られる結果は今までと全く同じはずである. (ここでの手順は難しい. 前のファイルをセーブして進むと良い.)

ここで 2 段階ラックス・ヴェンドロフ法にするに次の差分式にすればよい.

$$\mathbf{u}_i^{\overline{n+1}} = \mathbf{u}_i^n - \frac{s}{h}(\mathbf{w}_{i+1}^n - \mathbf{w}_i^n)$$

$$\mathbf{u}_i^{n+1} = \frac{1}{2} \left[\mathbf{u}_i^n + \mathbf{u}_i^{\overline{n+1}} - \frac{s}{h}(\mathbf{w}_i^{\overline{n+1}} - \mathbf{w}_{i-1}^{\overline{n+1}}) \right]$$

ここで新しい変数 \mathbf{w}_i^n と $\mathbf{u}_i^{\overline{n+1}}$ が必要であることが分かるであろう. 具体的には次のような手順でプログラムを変更していけばよい. ここで, 新しいファイル (`laxwen.c` とでも名付ける) を作って書き直していくようにする.

```
cp wave2.c laxwen.c
```

手順 1 <配列 w の導入>

w_i^n を用いて差分式を書き直す. すると差分式は 1 本の式で書くことができる.

(1) w_i^n に対応する変数 `ww[q][i]` の使用を宣言する.

```
double ww[qmax][imax+1];
```

(2) w_i^n の計算を差分式の計算の直前で行う.

```
for(i=0;i<=imax;i++){
    ww[0][i]=uu[1][i];
    ww[1][i]=uu[0][i];
}
```

[注意] 必ず **ww** の計算は **n** の **for** 文の中で計算すること. もちろん, 時間発展の差分式の直前である.

(3) 差分式を **ww** を用いて書き直す.

```
for(i=1;i<imax;i++)
    for(q=0;q<qmax;q++)
        un[q][i]=uu[q][i]-0.5*kap*(ww[q][i+1]-ww[q][i-1])
            +0.5*kap*kap*(uu[q][i+1] -2.0*uu[q][i] + uu[q][i-1]);
```

この変更によるデータの変化はないはずである. 確認せよ (UNIX では **diff** を用いる).

手順 2 < 2 段階ラックス・ヴェンドロフ法の差分式への変更 >

2 段階ラックス・ヴェンドロフ法の差分式に変更する. 差分式のところ全体を次のように変更する.

(1) 2 段階ラックス・ヴェンドロフ法の間段階の差分式の変数 u_i^{n+1} に対応する変数 **uh** を使用することを宣言する.

```
double ww[qmax][imax+1],uh[qmax][imax+1];
```

(2) 今までの差分式のところを第 1 段階目の差分式の計算に置き換える.

```
for(i=0;i<imax;i++)
    for(q=0;q<qmax;q++)
        uh[q][i]=uu[q][i]-kap*(ww[q][i+1]-ww[q][i]);
for(q=0; q<qmax; q++) {uh[0][imax] = 0.0;
                        uh[1][imax] = uh[1][imax-1];}
```

ここで **i=imax** での値が計算されていないので, 境界条件によりそれを与えている.

(3) 第 2 段階目の計算を加える. その直前 (第 1 段階目の計算の直後) に **ww** を計算し直すことも忘れてはならない.

```
for(i=0;i<=imax;i++){
    ww[0][i]=uh[1][i];
```

```

        ww[1][i]=uh[0][i];
    }
    for(i=1;i<=imax;i++)
        for(q=0;q<qmax;q++)
            un[q][i] = 0.5*(uu[q][i] + uh[q][i]
                -kap*(ww[q][i]-ww[q][i-1]));
    for(q=0;q<qmax;q++) {un[0][0]=0.0;
        un[1][0]=un[1][1];}

```

ここでまた境界条件を整えた。以前の境界条件の設定はここではもう不必要なので、消すと良い。このプログラムの変更によるデータの変化はないはずである。(ただ、丸め誤差程度のずれはあるかもしれない) 確かめよ。

手順 3 < w の計算の関数化 >

ここで w の計算を 1 ステップ毎に 2 回計算する必要があることに気づくであろう。 w の計算が簡単な今回のような場合はこの同じ計算を 2 回繰り返すのも良いが、非線形方程式などを扱う場合 w の計算が複雑になり 2 箇所ほとんど同じことを書くのはかなり面倒になる。そこで、 w の計算は「関数」の中でまとめて計算するとよい。例えば次のような関数を作っておけば、それを呼び出すだけで w の計算ができることになる。ここでこの関数に `calw` という名前を付ける。

```

void calw(double uu[qmax][imax+1], double ww[qmax][imax+1])
{
    int i;

    for(i=0;i<=imax;i++){
        ww[0][i] = uu[1][i];
        ww[1][i] = uu[0][i];
    }
}

```

計算本体 (main) では w を `for` 文中で計算する代わりに単に `calw(uu,ww)`, `calw(uh,ww)` と書けばよい。ここで、結果が前の計算と同じであることを (UNIX であれば `diff` を用いて) 確認する。

手順 4 < 周期境界条件 >

ここでステップ数 `nmax` を大きくする (`nmax=50`) と山が境界にぶつかり、計算が崩れてしまう。解析的には完全反射が起こるはずだが数値計算ではそうはならないことが分かる。そこで数値的に扱いやすい境界条件である「周期境界条件」に設定しなおす。周期境界条件とは現象が周期的で端がないという条件である。今回の場合、数学的には $\mathbf{u}(0, t) = \mathbf{u}(1, t)$ と書ける。

(1) `uh` の境界条件を与えるところを次のように変更する。

```

    for(q=0;q<qmax;q++)

```

```
uh[q][imax]=uh[q][0];
```

(2) 次に, un の境界条件を与えるところを次のように書き換える.

```
for(q=0;q<qmax;q++)
    un[q][0]=un[q][imax];
```

こうすると波が右側の境界を横切って出てゆくと, 同じような波が左の境界から入ってくるのが分かる. 以上で非線形方程式 (流体力学方程式) をとりあつかう準備が整った.

A.5 課程 5 : 非線形方程式 <流体力学方程式> の数値解法

さて, いよいよ非線形方程式の数値解法の例として流体力学方程式の数値計算に取り掛かろう.

手順 1 <変数 u の数 qmax を 3 に増やす>

まず, 変数の数 qmax を 3 に増やす. 新しい未定義の配列には他の対応する変数と同じになるようにする. 例えば, uu[2][i]=uu[1][i] や ww[2][i]=ww[1][i] とすればよい. これは calw の中と main の初期条件のところのみ変更すればよい. この変更は単に変数の個数を増やすだけで計算結果は変わらない. このことを確認する.

手順 2 <w を流体力学方程式の流束密度にする>

流束密度の行列 \mathbf{w} を計算する関数 calw での計算を

$$\mathbf{w} \equiv \begin{pmatrix} \rho v \\ \rho v^2 + p \\ (e + p)v \end{pmatrix}$$

に対応するよう書き換える. このとき各メッシュにおいて \mathbf{u} の値から p , v を次のように求めてから \mathbf{w} の計算をすると分かりやすい.

$$v = \frac{\rho v}{\rho}, \quad p = (\gamma - 1) \left(e - \frac{\rho}{2} v^2 \right)$$

ここで, ρ , ρv , e は $\mathbf{u} = \begin{pmatrix} \rho \\ \rho v \\ e \end{pmatrix}$ の各成分である. また, γ は比熱比であり, $\gamma = 5/3$ とする.

手順 3 <初期条件の設定>

ここで初期条件を与える.

$$\begin{aligned} \rho &= 1 && (0 \leq x \leq 1), \\ \rho v &= 0 && (0 \leq x \leq 1), \\ e &= \begin{cases} 1.9 - 4|x - \frac{1}{2}| & (\frac{1}{4} \leq x \leq \frac{3}{4}) \\ 0.9 & (0 \leq x < \frac{1}{4}, \frac{3}{4} < x \leq 1) \end{cases} \end{aligned}$$

また, ここでは `uu[2][i]` を出力するようにする. ガスが中央付近 ($x = 0.5$) から両側に向かって広がる様子が見て取れるであろう (図 6. 11 参照). (広がる速度を計算せよ.)

試してみよう① 音波のシミュレーション

音波の伝播のシミュレーションを行うために, 初期条件を次のように変えてみよう. ここで比熱比を $\gamma = 5/3$ とする.

$$u = \rho v = A \cos 2\pi x, \quad \rho = 1 + A \cos 2\pi x, \quad e = 0.9(1 + \gamma A \cos 2\pi x)$$

これは音速 $c = \sqrt{\gamma p / \rho} = 1$ の音の伝播を与える初期条件である. はじめは音波の振幅を $A = 0.01$ (あるいは, $A = 0.0001$) として線形音波の伝わる様子を確認しよう (図 6. 12(a) 参照). このとき音波の伝播速度が 1 になっていることを確認せよ.

[注意] このときファイルを新しくし, そのファイル名を例えば `sound.c` とでもしておく.

うまく線形音波の計算ができていようであれば, 音波の振幅 A を 0.01 から徐々に大きくしてゆき 0.2 ~ 0.3 くらいまでの非線形音波の計算を行ってみよう. このとき自発的に衝撃波が形成されるようすが見られる (図 6. 12(b) 参照). ただ, 解析的にはノコギリ歯状の衝撃波が伝わる解になるので, 衝撃波のような不連続な関数が現れるはずである. しかし, 数値計算の結果では衝撃波の下流に当たるところに振動が現れる. これは数値的なもので, 不連続な関数の現れる現象をラックス・ヴェンドロフ法で計算したときに現れる.

試してみよう② 爆縮

爆縮の数値シミュレーションを行うために, さらに初期条件を第 6 章第 5 節の式 (6.35) に変えてみよう. 結果については図 6. 14 と比較せよ.

[コツ] 爆発のシミュレーションで e の初期条件, それも $\frac{1}{4} < x < \frac{3}{4}$ の場合をちょっと変えるだけでよい.

試してみよう③ 高密度ガスの衝突

高密度ガスの衝突の数値シミュレーションを行うために, さらに初期条件を第 6 章第 5 節の式 (6.34) に変えてみよう. 結果については図 6. 13 と比較せよ.

これで 1 次元の流体力学方程式を解くコードができたことになる. 初期条件を変えてさまざまな計算を試みることをすすめる. 実際は 2 次元・3 次元の数値計算がより用いられる. 付録 B にそのための簡単な方針を示した. これらの計算についても挑戦して頂きたい.

付録 B

2次元流体力学方程式の数値解法

ここで2次元流体力学方程式の数値計算のためのプログラム（流体力学コード）の組み方について簡単に述べる。

2次元流体力学方程式は次のように書ける（第1.4.8節参照）。

$$\frac{\partial \mathbf{u}}{\partial t} = -\frac{\partial \mathbf{w}_1}{\partial x} - \frac{\partial \mathbf{w}_2}{\partial y} \quad (\text{B.1})$$

$$\mathbf{u} \equiv \begin{pmatrix} \rho \\ \rho v_x \\ \rho v_y \\ e \end{pmatrix}, \quad \mathbf{w}_1 \equiv \begin{pmatrix} \rho v_x \\ \rho v_x^2 + p \\ \rho v_x v_y \\ (e+p)v_x \end{pmatrix}, \quad \mathbf{w}_2 \equiv \begin{pmatrix} \rho v_y \\ \rho v_x v_y \\ \rho v_y^2 + p \\ (e+p)v_y \end{pmatrix} \quad (\text{B.2})$$

ここで $e = \frac{\rho}{2}v^2 + \frac{p}{\gamma-1}$ は全エネルギー密度である。

次のような手順で1次元流体力学コードから2次元流体力学コードに変えることができる。

1. 1次元流体力学コード（`sound.c`を用いるとよい）において， \mathbf{u} ， \mathbf{w} の個数（`qmax`）を4に増やす。
このとき，`q=3`の変数は例えば`q=2`と同じにする。計算結果は以前と全く変わらないはずである。
2. 新しく加わる独立変数 y に対応する \mathbf{u} ， \mathbf{w} の次元を増やす。
このとき，独立変数 y については一様として計算する。結果は以前と変わらないことを確認する。

```
double uu[qmax][imax+1][jmax+1];
```

```
double ww[qmax][imax+1][jmax+1];
```

[コツ] このとき，出力は $j=0$ ， $j=jmax/2$ ， $j=jmax$ の3箇所で出力してみる。どれも同じ結果になることを確認する。

3. 新しく \mathbf{w}_2 に対応する配列 `w2` を加える。このとき，`ww` を `w1` と書きかえる。
`ww` を `w1` と書き換え，`w2` を加える。ここで，`w2` の値は全てゼロと設定する。
4. 時間発展の差分を2次元の2段階ラックス・ヴェンドロフ法の式にする。
(第1段階，予測子)

$$\begin{aligned} \text{uh}[q][i][j] = & \text{uu}[q][i][j] - \text{kap1}*(\text{w1}[q][i+1][j]-\text{w1}[q][i][j]) \\ & - \text{kap2}*(\text{w2}[q][i][j+1]-\text{w2}[q][i][j]); \end{aligned}$$

(第2段階, 修正子)

$$\begin{aligned} \text{un}[q][i][j] = & 0.5*(\text{uu}[q][i][j] + \text{uh}[q][i][j] \\ & - \text{kap1}*(\text{w1}[q][i][j]-\text{w1}[q][i-1][j]) \\ & - \text{kap2}*(\text{w2}[q][i][j]-\text{w2}[q][i][j-1])); \end{aligned}$$

ここで, 初期条件を設定する前に $\text{kap1}=1.0/\text{imax}$, $\text{kap2}=1.0/\text{jmax}$ とする.

5. `calw` 内で $w1$, $w2$ を計算するようにする.

このとき, 配列 $w2$ を加えるだけで, $w1$, $w2$ の計算はまだ変更しなくてもよい. 前の計算を再現することを確認する.

6. `calw` 内で $w1$, $w2$ の計算を 2次元流体力学方程式の流束密度の式 (B.2) に変更する.

しかし, このときもまだ前の計算を再現するはずである. 確認しながら進む.

7. 境界条件 (周期境界条件) を整える.

8. y 方向に非一様な計算がうまく計算できるか試すために, y 方向への音波の計算を行う.

このとき, 初期条件を変えるだけでよい.

[コツ] ここで, 1番目~7番目のプログラムの変更では出力結果は変更前の結果と変わらないはずである. もとのプログラム (`wave2.c`) の出力を 'data0' というファイルにしておいて, 常にそれと変更したプログラムの出力結果のファイルが同じであることを確かめながら進むとよい. (ファイルの比較の際は UNIX の場合は `diff`, Windows の場合は `fc` を用いる.) また, 最後の8番目の変更でも出力を $j=0,1,2,\dots,jmax$, $i=0$ とすれば結果は変わらないはずである. 出力ファイルが同じであることを確認しよう.

以上の手順で2次元の流体力学コードができる. このコードの初期条件を変えれば第1章で示したケルビン・ヘルムホルツ安定性 (例1.4) などさまざまな流体計算ができる. 機会があれば, 2次元・3次元数値計算の豊潤な世界を紹介したい.