平成 19 年 6 月 10 日

複雑系科学実験１ 渡辺担当分

渡辺 宙志 *

名古屋大学大学院情報科学研究科複雑系科学専攻多自由度システム情報論講座

概要

目 次

*E-mail: hwatanabe@is.nagoya-u.ac.jp

1　はじめに

全体を通して、数値計算の基礎と同時にプログラミングの作法なども学ぶことを目的とする。言語は主に C/C++を用い、補助的に Java や Perl を使用する。数値計算が意外に簡単、単純であることなどを感じてほしい。なお、授業には直接関係のない質問も歓迎する。

2　偏微分方程式の数値解法

物理は、極論すれば現象をあらわす偏微分方程式 (支配方程式と呼ばれる) を考え、それを解く学問である。有名な偏微分方程式として、ニュートンの運動方程式、拡散方程式、波動方程式、ナビエ・ストークス方程式などがある。一般に厳密解を求められればそれに越したことはないが、簡単な方程式でも境界条件が複雑になると解析的に解くのは難しくなる。そこで、数値計算で微分方程式を解く必要が出てくる。

微分方程式の数値解法は、現在いろいろなところで我々の生活の役に立っている。天気予報は、流体の方程式にいくつかの条件を付け加えた連立微分方程式をスーパーコンピュータで力任せに計算することで行われている。車の設計や、建物の安全性解析なども重要な分野である。また、半導体の設計など、ミクロな分野でも数値計算が日常的に使われている。そこで、まず偏微分方程式、特に線形偏微分方程式と呼ばれる微分方程式を数値的に解く手法を学ぶ。

2.1　gnuplot の使い方

2.1.1　gnuplot の起動と操作

まず、本講義で使用するグラフィックソフト、gnuplot の使い方を簡単に学ぶ。

gnuplot は、コマンドライン上で `gnuplot` と打ち込めば起動する。

```
% gnuplot
G N U P L O T
Version 3.7 patchlevel 3
(中略)
gnuplot>
```

すると、コマンド入力待ち状態となる。たとえば、sin 関数をプロットしてみよう。

```
gnuplot> plot sin(x)
```

すると、ウィンドウが現れて sin 関数が表示されるはずである。デフォルトでは $-10 < x < 10$ の範囲が表示されるが、この定義域を変えてみよう。たとえば、$0 < x < \pi$ の範囲を表示するには、

```
gnuplot> plot [0:pi] sin(x)
```

と入力する。さらに y 座標の範囲も変えるには、もう一つ $[y_s : y_e]$ を書く。

```
gnuplot> plot [0:0.5] [0:0.5] sin(x)
```

二つ以上の関数を同時に表示するには、カンマで区切って書く。

```
gnuplot> plot [0:0.5] [0:0.5] sin(x),x
```

これにより、$y = \sin(x)$ が、$x \sim 0$ で $y = x$ に近づくことが分かる。そのほか、gnuplot ではべき乗を** で表す。たとえば、$y = x^2$ をプロットしたければ、

```
gnuplot> plot x**2
```

とする。

2.1.2 データファイルの扱い

gnuplot は、データファイルを表示することも出来る。たとえば、次のようなファイルを用意し、test.dat という名前で保存せよ。

```
1 1
2 4
3 9
4 16
5 25
```

このデータをプロットするには、

```
gnuplot> plot "test.dat"
```

とする。

gnuplot は、対数プロットもできる。先ほどのデータを両対数プロットで表示してみよう。そのためには、`set log` というコマンドを用いる

```
gnuplot> set log xy
gnuplot> plot "test.dat"
```

両対数プロットにより、データが直線状になることが確認できる。なお、対数表示を戻すには、`set nolog` というコマンドを用いる。両方の軸を通常表示に戻したければ、

```
gnuplot> set nolog xy
```

と指定する。

さらに、複数行のデータを扱うこともできる。たとえば、次のようなファイルを用意し、test2.dat という名前で保存せよ。

```
1 1 0.2
2 4 0.8
3 9 1.8
4 16 3.2
5 25 5
```

次のように普通にプロットすると

```
gnuplot> plot "test2.dat"
```

先ほどの結果と同じになる。1 番目のデータを x 軸に、3 番目のデータを y 軸に表示したい場合は、

```
gnuplot> plot "test2.dat" using 1:3
```

と using オプションをつける。また、1 番目のデータを x 軸、2 番目のデータを y 軸として表示し、3 番目のデータをエラーバーとして表示したい場合は、

```
gnuplot> plot "test2.dat" with yerrorbars
```

とする。このデータは、絶対値のおよそ 20%が誤差であるような測定値を表現している。with の後に続くオプションは yerrorbars のほか、点を直線で結ぶ lines というオプションもあるので試してみよ。

2.1.3 課題

課題１

gnuplot で範囲を細かく指定することで、方程式 $x = 2\sin(x)$ の $x = 0$ 以外の解を小数点以下３桁まで求めよ。

ヒント：方程式の解は、$y = x$ と $y = 2\sin(x)$ の交わる点である。

課題２

$\sin(x)$ の $x = 0$ の近傍での Tailor 展開は、

$$\sin(x) = \sum_n \frac{(-1)^{n-1}}{(2n-1)!} x^{2n-1} \tag{1}$$

$$= x - x^3/3! + x^5/5! - \cdots \tag{2}$$

である。この級数を途中で打ち切る近似は、$x = 0$ に十分近い値でのみ正当化される。$y = \sin(x)$ を一次関数 $y = x$ で近似した場合と、3 次関数 $x - x^3/6$ で近似した場合について、真の値からの相対誤差が 1% 以上ずれはじめるのはどのあたりか調べよ。

ヒント：真の値からの相対誤差とは、$\sin(x)$ を近似式で割った値の絶対値から 1 を引いたものである。

課題３

本文中に例示したデータ test2.dat を、1 番目のデータを x 軸とし、2 番目のデータを y 軸の値としたものと 3 番目のデータを y 軸の値としたものを重ねてプロットせよ。また、両対数プロットで表示したとき、二つのデータがどんな関係か考察せよ。

2.2 熱伝導方程式

最初に、熱伝導方程式と呼ばれる微分方程式を扱う。時刻 t、位置 x における温度を $\phi(t, x)$ とすると、温度分布 ϕ の時間発展は

$$\frac{\partial \phi}{\partial t} = \Delta \phi \tag{3}$$

と表すことができる。Δ はラプラシアン (Laplacian) と呼ばれる演算子で、一次元の場合は ∂_x^2 を表す。これは熱流量が温度勾配に比例するということを表している[1]。以下、一次元の場合について考えよう。もし系が十分に落ち着いて、時間変化がなくなったとすると、時間による偏微分がゼロになるので、

$$\Delta \phi = 0 \tag{4}$$

と表される。式 (4) は**ラプラス方程式 (Laplace's equation)** と呼ばれ、応用上非常に重要な方程式である。系が発熱を伴う場合は、時刻 t、位置 x における単位時間当たりの発熱量を $q(x, t)$ として

$$\Delta \phi = -q \tag{5}$$

と表すことができる。この方程式は**ポアソン方程式 (Poisson's Equation)** と呼ばれ、重力場、静電場などを表すこれまた重要な方程式である。時間変化が含まれる場合は

$$\frac{\partial \phi}{\partial t} = \Delta \phi + q \tag{6}$$

と表すことができる。

これらの方程式を、適当な境界条件において解くのが目的である。

[1] これをフーリエ則 (Fourier's Law) と呼ぶ。

2.2.1 解析解

まずは、解析的に解を求めてみよう。長さ L の鉄の棒を考える。左端を $x = 0$、右端を $x = L$ としよう。左端の温度を 0、右端の温度を A に保ったままずっと放っておいたらどうなるかを考える。ずっと放っておいて、系が落ち着いたとすると時間微分はゼロだから、解くべき方程式は

$$\frac{\partial^2 \phi}{\partial x^2} = 0 \tag{7}$$

となる。二階微分がゼロなのだから、$\phi(x)$ の一般解は一次関数

$$\phi(x) = ax + b \tag{8}$$

と書けるだろう。ここで境界条件 $\phi(0) = 0$、$\phi(L) = A$ より、

$$a = A/L \tag{9}$$

$$b = 0 \tag{10}$$

である。結局

$$\phi(x) = \frac{A}{L}x \tag{11}$$

と求めることができる。このように、系の右端と左端を一定温度に保つと、温度変化は直線的になる。

2.2.2 差分化

前節では解析的に問題が解けたが、時間発展も含む場合や境界条件が複雑な場合などは手で解くのは非常に面倒になる。そこで、数値的に解くことにしよう。

数値的にとくためには、コンピュータが理解できる形式でデータを表現してやらないといけない。まず、温度分布 $\phi(x)$ は連続関数だが、これを細かく区切った区間で表現しよう。たとえば 100 区間に区切るなら、

```
const int L = 100;
const int N = L+1;
double phi[N];
```

といった配列で表すことになる。100 区間に区切った場合、点の数は 101 個あることに注意しよう。これは $\phi(x)(0 \leq x \leq L)$ を $\phi_i \quad (i = 0, \cdots, 100)$ で近似したことに対応する。ϕ_i の値が配列 `phi[i]` の値に対応する。

次に、微分演算を数値的に表現する。$\phi(x)$ のある場所 x における微分は、微小距離 Δx を用いて

$$\frac{\partial \phi}{\partial x} \sim \frac{\phi(x + \Delta x) - \phi(x)}{\Delta x} \tag{12}$$

と近似できるだろう。ある場所 x の微分を表すのに、そこより前 $(x + \Delta x)$ の値を使っているので、これを**前進差分 (forward difference)** と呼ぶ。いま、$\Delta x = 1$ とすると、計算機の上では

$$\frac{\partial \phi_i}{\partial x} \sim \phi_{i+1} - \phi_i \tag{13}$$

余談コラム 1

数値計算に限らず、プログラムを組む際には、何よりも**バグを入れない**ということが大事である。良く誤解されているが、バグを入れないというのは バグのないコード を書くことではない。将来仕様を変更した時に バグが入りにくいコード を書くことである。その第一歩が「配列の大きさを定数で定義する」ことだ。配列の大きさはプログラム中で何度も使うことになるので、将来変更した時に変更箇所が一箇所で済むようにする。そうでないと、変更し忘れというバグの危険を生むことになる。デバッグの基本は「バグの危険性を減らすこと」である。

と、単に差の形で表現できる。位置 x を単に添え字 i で表現するようになったことに注意しよう。必要なのは二階微分なので、同様な作業を今度は**後進差分 (backward difference)** を使って繰り返すと

$$\frac{\partial^2 \phi_i}{\partial x^2} \sim \phi_{i+1} - 2\phi_i + \phi_{i-1} \tag{14}$$

を得る。このように前後両方の値を使うことを**中心差分 (central difference)** と呼び、片方の値を使う場合よりも性質が良いことが知られている。次に時間微分について考える。空間微分の場合と同様に微小時間 Δt を用いて

$$\frac{\partial \phi_i(t)}{\partial t} \sim \frac{\phi_i(t+\Delta t) - \phi_i(t)}{\Delta t} \tag{15}$$

と近似しよう。ここで $\phi_i(t)$ が現在時刻での温度の値、$\phi_i(t+\Delta t)$ は次の時間ステップでの値である。これを $\phi_i(t+\Delta t)$ について解けば

$$\phi_i(t+\Delta t) = \phi_i + \frac{\partial}{\partial t}\phi(x,t)\Delta t \tag{16}$$

ここで、式 (3) より $\partial_t \phi = \partial_x^2 \phi$ であったから、

$$\phi_i(t+\Delta t) = \phi_i + \frac{\partial^2 \phi_i}{\partial x^2}\Delta t \tag{17}$$

式 (14) より、

$$\phi_i(t+\Delta t) = \phi_i + (\phi_{i+1} - 2\phi_i + \phi_{i-1})\,\Delta t \tag{18}$$

以上から、次のステップのデータを、現時刻でのデータのみで表現することができた。初期条件を与えてからこのステップを次々と繰り返せば、任意の時間での温度プロファイルを得ることができる。なお、このように求めたい値 (この場合は次の時間での温度) が、現在のデータのみで陽 (explicit) に表現できるような解法を**陽解法 (explicit scheme)** と呼ぶ。逆に、求めたい関数がその関数自身を使って表現されているような解法を**陰解法 (implicit scheme)** と呼ぶ。陰解法では値を求めるには反復法などの手法を使わなくてはならないのに対し、陽解法は少ない計算で求める値が得られる。しかし、一般的に陰解法の方が数値的に安定である[2]。

以上の手順からプログラムの流れを整理すると、

1. 配列宣言など
2. 値の初期化
3. 次のステップの値を式 (18) によって求める
4. 任意の時間ステップだけ 3 を続ける
5. 結果を出力する

という形になる。

2.2.3 実行及び解析

2.5.1 節にあるソースコードを入力し、適当な名前 (たとえば kakusan1d.cc) で保存してから以下の作業をせよ。

[2]数値的に安定、というのは、発散などを起こさずに数値積分を行えること。陽解法では、ある特定の条件下 (たとえば粒子が壁に近づいたり、波の振幅がある程度以上大きくなるなど) で計算の値が発散したりする。数値不安定性は計算物理の中でもやっかいな相手の一つである。

```
% g++ kakusan1d.cc      ←コンパイル
% ./a.out > result.dat ←実行結果を result.dat に保存
% gnuplot               ← gnuplot を起動

G N U P L O T
Version 3.7 patchlevel 3
(中略)
gnuplot> plot "result.dat"  ←結果を表示
```

2.2.4 課題

課題１

ソースのループ数を色々変えて、結果何が起きるか観察せよ。特に、十分緩和したとみなせる時間が経った後に何が起きるか調べよ。

課題２

初期条件の温度をいろいろ変えて、結果がどうなるか調べよ。

課題３

両端の温度を 0 に保ったまま、一様に発熱がある場合を計算するようにソースコードを変更し、十分緩和したらどうなるか調べよ。また、結果を解析解と比較せよ。

2.3 波動方程式

前節でラプラス方程式、ポアソン方程式と呼ばれる微分方程式を解いた。この方程式の時間微分を二階微分にしたような方程式

$$\frac{\partial^2 \psi}{\partial t^2} = c^2 \nabla^2 \psi \tag{19}$$

は**波動方程式 (wave equation)** と呼ばれ、名前の通り波動、振動現象を記述する。式中に現れる c は、この系の速度を表す。一次元における波動方程式は

$$\frac{\partial^2 \psi}{\partial t^2} = c^2 \frac{\partial^2 \psi}{\partial x^2} \tag{20}$$

となる。以下、この方程式を扱うことにしよう。

2.3.1 解析解

まずは解析解を求める。この形の微分方程式は変数分離法で解くことができるが、詳細は別書に譲る。解は $\exp(i(kx - \omega t))$ の重ね合わせで表される。ただし k は波数、ω は角振動数であり、$c^2 = \omega^2/k^2$ を満たす (実際に式に代入して確かめよ)。

まずは初期条件と境界条件を与えよう。長さ L の系を考え、左端を $x = 0$、右端を $x = L$ としよう。境界条件として $\psi(0, t) = 0$ かつ $\psi(L, t) = 0$ を課す。このような条件を**ディクレ条件 (Dirichlet condition)** と言う。すると、解は両端を固定し、ぴんとはった弦の振動の様子を表現する。このとき、$\psi(x, t)$ は、時刻 t における位置 x の弦の位置を与える。初期条件として

$$\psi(x, t = 0) = A \sin(kx) \quad k \equiv \pi/L \tag{21}$$

$$\partial_t \psi(x, t = 0) = 0 \tag{22}$$

を与えよう。ただし A は振幅である。支配方程式の時間微分が 2 階なので、$\partial_t \psi$ の状態も指定しなければならないことに注意しよう[3]。解の形を $A \sin(kx) \cos \omega t$ の形に仮定して、微分方程式に代入すれば $\omega = c^2/k^2$

[3] これはニュートンの運動方程式を解くのに、初期位置と初期速度を指定しなければならないのと同じである。

と求まる。この解は、弦の基本振動を表す。角振動数 ω が波数 k、すなわち長さ L を使って指定されていることに注意してほしい。

2.3.2 差分化

空間を差分化し、位置 $x = i$ における位置データ $\psi(x = i)$ を $q_i(i = 0, \cdots, N)$、速度データ $\partial_t\psi(x = i)$ を $v_i(i = 0, \cdots, N)$ として表現しよう[4]。

空間微分の差分化は熱伝導方程式の場合と同様に

$$\frac{\partial^2 q_i}{\partial x^2} = q_{i+1} - 2q_i - q_{i-1} \tag{23}$$

で表現できる。時間微分も二階であるから同様に計算できないことはないが、せっかくエネルギーが保存する系であるので**シンプレクティック積分 (symplectic integration)** を用いる。シンプレクティック積分とは、シンプレクティック写像の応用で、ハミルトン系を長時間安定に計算することができる。具体的には以下のようにする。

$$q_i \leftarrow q_i + v_i \cdot \frac{\Delta t}{2} \tag{24}$$

$$v_i \leftarrow v_i + f \cdot \Delta t \tag{25}$$

$$q_i \leftarrow q_i + v_i \cdot \frac{\Delta t}{2} \tag{26}$$

ただし $f \equiv \partial_x^2 q_i$ である[5]。

2.3.3 課題

課題１

2.5.2 節にあるソースコードを入力し、適当な名前 (たとえば `hadou1d.cc`) で保存、コンパイルせよ。実行を確認後、弦の運動エネルギーを計算し、その周期を確認せよ。ただし、質量密度を 1 とすると、運動エネルギーは

$$\int_0^L \mathrm{d}x \left(\partial_t\psi(x)\right)^2 \tag{27}$$

で表される。

課題２

位置エネルギーを計算し、運動エネルギーとあわせて全エネルギーの保存を確かめよ。ただし、波数が k の場合、位置エネルギーは

$$\int_0^L k^2 \mathrm{d}x \psi^2(x) \tag{28}$$

で表される。

課題３

サンプルコードは初期条件として、節のないものを指定しているが、これを節の数が一つあるもの、二つあるものと変更せよ。その際、運動エネルギーの周期と節の数の関係を確認せよ。また全エネルギーの保存を確かめよ。

---

[4]拡散方程式の場合もそうであったが、実際には空間刻みを Δx として、$\psi(x = i\Delta x)$ に q_i を対応させる、などとするべきであるが、ここでは差分化の精度などはあまり問題にならないので、簡単のために $\Delta x = 1$ とおいてしまっている。

[5]速度の時間微分であるので、力のような気持ちで f と記した。

2.4 シュレーディンガー方程式

これまで陽解法、すなわち求める値が数ステップで計算できる方法を用いてきたが、陰解法も扱うことにしよう。ここではその例として固有値問題を取り上げる。扱う微分方程式は時間非依存のシュレーディンガー方程式 (Schroedinger equation)、

$$-\frac{\partial^2\Psi(x)}{\partial^2 x}+V(x)\Psi(x)=E\Psi(x) \tag{29}$$

である。本来なら質量 m やプランク定数 $\hbar$ が含まれるが、適当に無次元化したと考えることにする。これは与えられたポテンシャル $V(x)$ の中で電子がどのように存在するかを記述する方程式である。また長さ L の一次元系を考えよう。ポテンシャルとして

$$V(x)=\begin{cases} 0 & 0\le x<\frac{L}{2},\frac{3L}{2}\le x<L \\ -V_0 & \frac{L}{2}\le x<\frac{3L}{2} \end{cases} \tag{30}$$

という形を与える。これは箱型ポテンシャルと呼ばれる形状である。求めたいのは、波動関数の形 Ψ と、固有エネルギー E である。

これをどう解くかは後回しにして、とりあえず差分化してみよう。例によって系を N 分割して、$\Psi(x)$ の各分割点での値を $\Psi_i(i=0,\cdots,N)$ としよう。すると、式 (29) は

$$-(\Psi_{i+1}-2\Psi_i+\Psi_{i-1})+V(x=i)\Psi_i=E\Psi_i \tag{31}$$

という形になる。ここで、Ψ_i をならべたベクトルを考えると、式 (31) は行列の形、

$$\begin{pmatrix} -2+V_0 & 1 & 0 & & & 0 \\ 1 & -2+V_1 & 1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & 1 & -2+V_{N-1} & 1 & \\ 0 & & & 1 & -2+V_N & \end{pmatrix}\begin{pmatrix}\Psi_0\\ \Psi_1\\ \Psi_2\\ \vdots\\ \Psi_N\end{pmatrix}=E\begin{pmatrix}\Psi_0\\ \Psi_1\\ \Psi_2\\ \vdots\\ \Psi_N\end{pmatrix} \tag{32}$$

で表すことができる。ただし $V_i\equiv V(\frac{L}{N}i)$ である。左辺の行列を H で表せば、最終的に

$$H\Psi=E\Psi \tag{33}$$

という形の、行列の固有値問題に帰着されることがわかる。

このように、差分化すると、欲しいデータがベクトルに、演算子が行列の形になることがわかる。したがって行列の性質などを論じた線形代数は、数値計算にとって重要な学問となる。中でも、行列の固有値

余談コラム 2

ソースの書き方はプログラマによって異なるが、C/C++や Java など、フリースタイルで書ける言語は特に個人差が大きい。だが、変数名の付け方や関数宣言の仕方、インデントの方法などにはいくつか主流の流儀が存在する。インデントの方法として有名なのは C/C++で良く使われている

```
int main(void)
{
  hogehoge;
}
```

と、perl、Java などで使われている形式

```
int
main(void){
  hogehoge;
}
```

がある。たかが左括弧の前に改行を入れるかいれないかなんて趣味の問題であるが、統一してあると読みやすいし愛着もわく。ちなみに私はずっと前者だったが、最近になって後者に統一するようにしている。

問題は、数値計算の中でももっとも重要な分野の一つである。身近な例としては、誰もが google の検索を使ったことがあるだろう。google の PageRank という考え方は、基本的には行列の固有値問題である[6]。

2.4.1 累乗法

行列の固有値問題を解く方法にはいくつかあるが、ここではもっとも簡単な累乗法 (power method) を用いる。最初に適当なベクトル $\mathbf{x}$(試行関数と呼ばれる) を用意し、それにひたすら行列 H をかける。すると、試行関数は徐々に絶対値が最大の固有値を持つ固有ベクトル $\mathbf{u}_1$ に収束する。一般的に固有ベクトル、固有値はたくさんあるが、累乗法はそのうち絶対値最大の固有値と固有状態しか得ることができない。しかし、物理では多くの場合絶対値最大固有値を持つ状態 (基底状態) に興味があるのでこれで問題はない。

具体的には次のような手順とする。

1. ノルムが 1 であるように規格化された適当なベクトル $\mathbf{x}$ を用意する。
2. $\mathbf{x}$ に行列 H をかけ、得られた結果を $\mathbf{x}'$ とする。

$$\mathbf{x}' = H\mathbf{x}$$

3. ノルムの比 $e = |\mathbf{x}'|/|\mathbf{x}|$ を計算する。
4. 得られたベクトルを規格化する

$$\mathbf{x} \leftarrow \mathbf{x}'/e$$

5. 以下、収束するまで 1〜4 の手順を繰り返す。

十分な繰り返しのあと、ノルムの比 e が最大固有値を、収束したベクトルが固有ベクトルを与える。ソースコードは 2.5.5 節に示す。

2.4.2 累乗法の仕組み

累乗法の仕組みを簡単に説明しておこう。N 行 N 列の行列 H の固有値を、絶対値の大きいものから $\lambda_1, \lambda_2, \cdots, \lambda_N$、これらの固有値に対応する固有ベクトルをそれぞれ $\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_N$ としよう。最初に与える試行ベクトルを $x^{(0)}$ とすると、そのベクトルは固有ベクトルで一意に展開できるはずである。その展開係数を $c_1, c_2, \cdots, c_N$ とすると、

$$\mathbf{x}^{(0)} = c_1\mathbf{u}_1 + c_2\mathbf{u}_2 + \cdots + c_N\mathbf{u}_N \tag{34}$$

とあらわすことができる。両辺に H をかけると、

$$H\mathbf{x}^{(0)} = c_1H\mathbf{u}_1 + c_2H\mathbf{u}_2 + \cdots + c_NH\mathbf{u}_N \tag{35}$$

$$H\mathbf{x}^{(0)} = c_1\lambda_1\mathbf{u}_1 + c_2\lambda_2\mathbf{u}_2 + \cdots + c_N\lambda_N\mathbf{u}_N \tag{36}$$

となる。ただし固有ベクトルの定義から、$H\mathbf{u}_i = \lambda_i\mathbf{u}_i$ である。$H\mathbf{x}^{(k)} = \mathbf{x}^{(k+1)}$ とすると、累乗法の手続きを n 回繰り返した結果 $\mathbf{x}^{(n)}$ は

$$\mathbf{x}^{(n)} = H^n\mathbf{x}^{(0)} \tag{37}$$

$$= c_1\lambda_1^n\mathbf{u}_1 + c_2\lambda_2^n\mathbf{u}_2 + \cdots + c_N\lambda_N^n\mathbf{u}_N \tag{38}$$

[6] より正確にはウェブページのリンクを状態遷移と考え、ページのつながりを Markov 鎖だとみなす。それに適切な重みを考えると、Markov 鎖の状態遷移を表す巨大な行列を考えることができる。ページの重要性は、この行列の固有値問題に帰着される。これを並列化したクラスターマシンで解いているのである。

両辺を λ_1^n で割ると、

$$\frac{\mathbf{x}^{(n)}}{\lambda_1^n} = c_1\mathbf{u}_1 + c_2\left(\frac{\lambda_2}{\lambda_1}\right)^n \mathbf{u}_2 + \cdots + c_N\left(\frac{\lambda_N}{\lambda_1}\right)^n \mathbf{u}_N \tag{39}$$

固有値のうち、λ_1 が絶対値最大であったから、$n \to \infty$ で $\mathbf{x}^{(n)}$ は 対応する固有ベクトル $\mathbf{u}_1$ に収束していくことがわかるだろう。

2.4.3 課題

課題１

2.5.5 節にあるソースコードを入力し、`eigen.cc` という名前で保存した後、コンパイル、実行せよ。ループの中で固有値を出力し、固有値の収束の様子を確かめよ。

課題２

ポテンシャルの底のエネルギー V_0 を色々変えて収束の様子がどのようにかわるか観察せよ。

課題３

ポテンシャルの底が浅すぎると、結果がおかしくなる。これはなぜか考察せよ。またどのようにすればよいかを考えよ。(この問題はやや難しいのでヒント。累乗法は絶対値最大の固有ベクトルを与える方法であって、基底状態を与える方法ではない。ヒントを見て考えても分からない場合は遠慮なく質問せよ。)

2.5 ソースコード

2.5.1 一次元拡散方程式

```
/*
 一次元熱拡散方程式を差分化して解くプログラム
 */
#include <stdio.h>
//定数
const int L = 100; // 系の長さ
const int N = L+1; // 点の数（端があるので長さ＋１）
const int LOOP = 100; //繰り返しの数
const double dt = 0.5;
const double A = 10;//端の温度
//変数
double phi[N]; //温度のデータ
double phi2[N]; //温度のデータのテンポラリ変数

int main(void){
  // 値の初期化
  for (int i=0;i<N;i++){
    phi[i] = 0;
  }
  // 境界条件
  phi[0] = 0;
  phi[N-1] = A;
  //メインループ
  for(int j=0;j<LOOP;j++){
    // 次のステップの温度を計算
    for (int i=1;i<N-1;i++){
      phi2[i] = phi[i] + (phi[i+1] - 2*phi[i] + phi[i-1])*dt;
    }
    // 次のステップの値をコピー
    for (int i=1;i<N-1;i++){
      phi[i] = phi2[i];
    }
  }
  //結果を出力
  for (int i=0;i<N;i++){
    printf("%d␣%f\n",i,phi[i]);
  }
}
```

2.5.2 一次元波動方程式

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

const int L = 100; //系の長さ
const int N = L+1; //点の数
const double A = 30; //振幅
const double k = 3.1415926535/(double)L;
const int LOOP = 1000;

//変数宣言
double q[N],v[N]; //位置と速度データ
double dt = 0.5; //時間刻み

void integrate(void);
double calc_energy(void);

//------------------------------------------------------------------------
/**
 * メイン関数
 */
int
main(void){
  //初期条件を与える
  for(int i=0;i<N;i++){
    q[i] = A*sin((double)i*k);
    v[i] = 0;
  }
  //数値積分
  for(int i=0;i<LOOP;i++){
    integrate();
    printf("%f␣%f\n",i*dt,calc_energy());
  }
}
//------------------------------------------------------------------------
/**
 * 数値積分
 */
void
integrate(void){
  for(int i=0;i<N;i++){
    q[i] += v[i] * dt * 0.5;
  }

  for(int i=1;i<N-1;i++){
    v[i] += (q[i+1] - 2*q[i] + q[i-1]) * dt;
  }

  for(int i=0;i<N;i++){
    q[i] += v[i] * dt * 0.5;
  }
}
//------------------------------------------------------------------------
/**
 * エネルギーの計算
 */
double
calc_energy(void){
  double e = 0;
  for(int i=0;i<N;i++){
    //ここを埋めよ
  }
return e;
}
```

2.5.3 一次元波動方程式 (Java 版)

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class hadou1d extends JFrame{

  static final int WIDTH = 200;
  static final int HEIGHT = 200;
  Image offImage; //Background Surface

  static final int L = 100; //系の長さ
  static final int N = L+1; //点の数
  static final double A = 30; //振幅
  static final double k = 3.1415926535/(double)L*2;

  static final int LOOP = 1000; //ループ数

  double q[] = new double[N]; //位置
  double v[] = new double[N]; //速度
  static final double dt = 1; //時間刻み

  public hadou1d(String s){
    super(s);
  }

  //描画ルーチン
  public void paint(Graphics g){
    if(offImage==null){
      offImage = createImage(WIDTH,HEIGHT);
    }
    Graphics offg = offImage.getGraphics();
    offg.setColor(Color.black);
    offg.fillRect(0,0,WIDTH,HEIGHT);
    int m = 2;
    int SX = 0;
    int SY = 100;
    offg.setColor(Color.white);
    for(int i=0;i<N-1;i++){
      int x1 = i*m+SX;
      int y1 = -(int)(q[i]*m)+SY;
      int x2 = (i+1)*m+SX;
      int y2 = -(int)(q[i+1]*m)+SY;
      offg.drawLine(x1,y1,x2,y2);
    }
    g.drawImage(offImage,0,0,this);
  }
//-----------------------------------------------------------------
  public static void main(String arg[]){
    hadou1d f = new hadou1d("One-dimensional␣Wave␣Equation");
    f.setSize(WIDTH,HEIGHT);
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    f.initialize();
    f.show();
    //メインループ
    for(int j=0;j<LOOP;j++){
      f.integrate();
      f.repaint();
      try{
        Thread.sleep(10);
      }catch(InterruptedException e){
      }
    }
  }
```

```
64 //-------------------------------------------------------------
65   public void initialize(){
66       for(int i=0;i<N;i++){
67         q[i] = A*Math.sin(k*i);
68         v[i] = 0;
69       }
70   }
71 //-------------------------------------------------------------
72   public void integrate(){
73     double dt2 = dt*0.5;
74     for(int i=0;i<N;i++){
75       q[i] = q[i] + v[i]*dt2;
76     }
77     for(int i=1;i<N-1;i++){
78       v[i] = v[i] + (q[i+1]-2*q[i]+q[i-1])*dt;
79     }
80     for(int i=0;i<N;i++){
81       q[i] = q[i] + v[i]*dt2;
82     }
83   }
84 }//End of class
```

2.5.4 一次元波動方程式 (Java 版) のコンパイル、実行方法

上記のソースを、必ず ”hadou1d.java”というファイル名で保存する[7]。保存したら、ファイルのあるディレクトリで

```
% javac hadou1d.java
```

を実行すればコンパイルできる。エラーが出たらよくソースを見なおすこと。エラーなくコンパイルできたら

```
% java hadou1d
```

で、プログラムを実行することができる。

[7]Java は、ファイル中の public なクラス名とファイル名を一致させなくてはいけない、という規則がある。

2.5.5 一次元シュレーディンガー方程式

```
/*
 一次元シュレーディンガー方程式を差分化して解くプログラム
 */
#include <stdio.h>
#include <math.h>

const int N = 100;
const double V0 = -4.0;//井戸型ポテンシャルの底のエネルギー
const int LOOP = 100;

double v[N]; //ポテンシャル
double phi[N]; //波動関数
double phi2[N]; //波動関数のテンポラリ変数

int
main(void){
  //初期化
  for(int i=0;i<N;i++){
    phi[i] = 1/sqrt((double)N);
    v[i] = 0;
  }
  //境界条件
  phi[0] = 0;
  phi[N-1] = 0;
  //箱型ポテンシャルを与える
  for(int i=N/4;i<N/4*3;i++){
    v[i] = V0;
  }

  //累乗法(Power Method)による反復計算
  double e_value = 0;
  for(int j=0;j<LOOP;j++){
    for(int i=1;i<N-1;i++){
      phi2[i] = - (phi[i+1]-2*phi[i]+phi[i-1])+v[i]*phi[i];
    }

    //固有値を求める
    double e1 = 0;
    for(int i=0;i<N;i++){
      e1 += phi[i]*phi[i];
    }
    double e2 = 0;
    for(int i=0;i<N;i++){
      e2 += phi2[i]*phi2[i];
    }
    e_value = - sqrt(e2/e1);

// printf("%d %f\n",j,e_value-V0);

    //規格化する
    for(int i=0;i<N;i++){
      phi[i] = phi2[i]/e_value;
    }
  }
  //結果の表示
  for(int i=0;i<N;i++){
    printf("%d␣%f\n",i,phi[i]);
  }
  //固有値の表示
  fprintf(stderr,"Eigen␣Value␣=␣%f␣\n",e_value-V0);
}
```

2.6 出席及び課題の提出について

授業終了時に、学生番号、名前、課題の回答、感想などをまとめ、hwatanabe@is.nagoya-u.ac.jp までメールにて提出すること。その際、メールの題名 (subject) は、「実験レポート (日付) 学籍番号」とせよ。たとえば、2007 年 5 月 28 日、学籍番号が 050500000 なら、「実験レポート (070528) 050500000」とせよ。感想や改善の指摘等を歓迎する。