計算機実験I (L2) — 常微分方程式の解法

藤堂眞治
wistaria@phys.s.u-tokyo.ac.jp

2019/05/08

準備: 微分方程式の書き換え

- 2 階の常微分方程式の一般形

$$\frac{d^2y}{dx^2} + p(x)\frac{dy}{dx} + q(x)y = r(x)$$

- $y_1 \equiv y$, $y_2 \equiv \frac{dy}{dx}$ とおくと

$$\begin{cases} \frac{dy_1}{dx} &= y_2 \\ \frac{dy_2}{dx} &= r(x) - p(x)y_2 - q(x)y_1 \end{cases}$$

- さらに $\boldsymbol{y} \equiv (y_1, y_2)$, $\boldsymbol{f}(x, \boldsymbol{y}) \equiv (y_2, r(x) - p(x)y_2 - q(x)y_1)$

$$\frac{d\boldsymbol{y}}{dx} = \boldsymbol{f}(x, \boldsymbol{y})$$

- n 階常微分方程式 $\Rightarrow$ n 次元の 1 階常微分方程式

初期値問題と境界値問題

- 初期値問題
  - 微分方程式において、ある 1 点に関する全ての境界条件 (初期値) が与えられているもの
  - 質点の運動など (時系列の問題)
- 境界値問題
  - 複数の点に関する境界条件が与えられているもの
  - 物体のゆがみの計算や静電場の計算など (空間的に解く問題)
- 初期値問題は初期値から逐次的に解くことが可能
- 境界値問題は初期値問題に比べて計算法が複雑

初期値問題の解法 (Euler 法)

- h を微小量として微分を差分で近似する (前進差分)

$$\frac{dy}{dt} \approx \frac{y(t+h) - y(t)}{h} = f(t, y)$$

- $t = 0$ における $y(t)$ の初期値を y_0、$t_n \equiv nh$、y_n を $y(t_n)$ の近似値とおくと、

$$y_{n+1} - y_n = hf(t_n, y_n)$$

- Euler 法
  - y_0 からはじめて、$y_1, y_2, \cdots$ を順次求めていく

Euler 法の精度

- 微分方程式の両辺を t_n から t_{n+1} まで積分 (積分方程式)

$$y(t_{n+1}) - y(t_n) = \int_{t_n}^{t_{n+1}} f(t, y(t)) dt = h \int_0^1 f(t_n + h\tau, y(t_n + h\tau)) d\tau$$

- Euler 法は、被積分関数を定数で近似することに対応

$$f(t_n + h\tau, y(t_n + h\tau)) = f(t_n, y(t_n)) + O(h)$$

- $t = 0$ からある t_f まで積分すると、反復回数 $N = t_f/h$
- $t = t_f$ における誤差 $\sim N \times h \times O(h) = O(h)$

Euler 法の改良

- 積分方程式の被積分関数をもう 1 次高次まで展開

$$f(t_n+h\tau, y(t_n+h\tau)) = f(t_n, y(t_n))+\tau h \left\{ \frac{\partial f}{\partial t} + f\frac{\partial f}{\partial y} \right\}_{t=t_n, y=y_n} +O(h^2)$$

- 積分を実行すると

$$y(t_{n+1}) = y(t_n) + hf(t_n, y_n) + \frac{1}{2}h^2 \left\{ \frac{\partial f}{\partial t} + f\frac{\partial f}{\partial y} \right\}_{t=t_n, y=y_n} + O(h^3)$$

中点法 (2 次 Runge-Kutta 法)

- 2 次公式

$$\begin{aligned} k_1 &= hf(t_n, y_n) \\ k_2 &= hf\left(t_n + \tfrac{1}{2}h, y_n + \tfrac{1}{2}k_1\right) \\ y_{n+1} &= y_n + k_2 \end{aligned}$$

- このとき

$$k_2 = h\left\{f(t_n, y_n) + \frac{1}{2}h\frac{\partial f}{\partial t} + \frac{1}{2}k_1\frac{\partial f}{\partial y} + O(h^2)\right\}$$

- したがって

$$y_{n+1} = y_n + hf(t_n, y_n) + \frac{1}{2}h^2\left\{\frac{\partial f}{\partial t} + f\frac{\partial f}{\partial y}\right\}_{t=t_n, y=y_n} + O(h^3)$$

高次の Runge-Kutta 法

- 3 次 Runge-Kutta 法

$$\begin{aligned} k_1 &= hf(t_n, y_n) \\ k_2 &= hf(t_n + \tfrac{2}{3}h, y_n + \tfrac{2}{3}k_1) \\ k_3 &= hf(t_n + \tfrac{2}{3}h, y_n + \tfrac{2}{3}k_2) \\ y_{n+1} &= y_n + \tfrac{1}{4}k_1 + \tfrac{3}{8}k_2 + \tfrac{3}{8}k_3 \end{aligned}$$

- 4 次 Runge-Kutta 法

$$\begin{aligned} k_1 &= hf(t_n, y_n) \\ k_2 &= hf(t_n + \tfrac{1}{2}h, y_n + \tfrac{1}{2}k_1) \\ k_3 &= hf(t_n + \tfrac{1}{2}h, y_n + \tfrac{1}{2}k_2) \\ k_4 &= hf(t_n + h, y_n + k_3) \\ y_{n+1} &= y_n + \tfrac{1}{6}k_1 + \tfrac{1}{3}k_2 + \tfrac{1}{3}k_3 + \tfrac{1}{6}k_4 \end{aligned}$$

- 4 次までは次数と f の計算回数が等しい

計算コストと精度

- 実際の計算では $f(t, y)$ の計算にほとんどのコストがかかる
- 計算回数と計算精度の関係

	1 次 (Euler 法)	2 次 (中点法)	3 次	4 次
計算精度	$O(h)$	$O(h^2)$	$O(h^3)$	$O(h^4)$
計算回数	N	$2N$	$3N$	$4N$

- 高次の Runge-Kutta を使う方が効率的
- どれくらい小さな h が必要となるか、前もっては分からない
- 刻み幅を変えて $(h, h/2, h/4, \dots)$ 計算してみることが大事
  - 誤差の評価
  - 公式の間違いの発見

陽解法と陰解法

- 陽解法: 右辺が既知の変数のみで書かれる (例: Euler 法)
  - プログラムがシンプル
- 陰解法: 右辺にも未知変数が含まれる
  - 例: 逆 Euler 法

$$y(t) = y(t+h-h) = y(t+h) - hf(t+h, y(t+h)) + O(h^2)$$
$$y_{n+1} = y_n + hf(t+h, y_{n+1})$$

  - 数値的により安定な場合が多い
  - Newton 法などを使って、非線形方程式を解く必要がある

Numerov 法

- Numerov 法
  - 二階の常微分方程式で一階の項がない場合に使える
  - 4 次の陰解法
  - 方程式が線形の場合は陽解法に書き直せる

- 微分方程式

$$\frac{d^2y}{dx^2} = f(x, y)$$

$y = y(x)$ を $x = x_i$ のまわりでテイラー展開する。
$x_{i\pm1} = x_i \pm h$ での表式は

$$y(x_{i\pm1}) = y(x_i) \pm h y'(x_i) + \frac{h^2}{2} y''(x_i) \pm \frac{h^3}{6} y'''(x_i) + \frac{h^4}{24} y''''(x_i) + O(h^5)$$

Numerov 法

- 二階微分の差分近似 ($y_i \equiv y(x_i)$ 等と書く)

$$\frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} = y_i'' + \frac{h^2}{12} y_i'''' + O(h^4)$$

一方で、微分方程式より

$$y_i'''' = \left.\frac{d^2 f}{dx^2}\right|_{x=x_i} = \frac{f_{i+1} - 2f_i + f_{i-1}}{h^2} + O(h^2)$$

組み合わせると

$$y_{i+1} = 2y_i - y_{i-1} + \frac{h^2}{12}(f_{i+1} + 10f_i + f_{i-1}) + O(h^6)$$

Numerov 法

- 方程式が線形の場合、$f(x, y) = -a(x)y(x)$ を代入すると

$$y_{i+1} = 2y_i - y_{i-1} - \frac{h^2}{12}(a_{i+1}y_{i+1} + 10a_i y_i + a_{i-1}y_{i-1}) + O(h^6)$$

y_{i+1} を左辺に集めると、陽解法となる

$$y_{i+1} = \frac{2(1 - \frac{5h^2}{12}a_i)y_i - (1 + \frac{h^2}{12}a_{i-1})y_{i-1}}{1 + \frac{h^2}{12}a_{i+1}} + O(h^6)$$

ハミルトン力学系

- 時間をあらわに含まない場合のハミルトン方程式

$$\frac{dq}{dt} = \frac{\partial H}{\partial p}, \quad \frac{dp}{dt} = -\frac{\partial H}{\partial q}$$

  - エネルギー保存則

$$\frac{dH}{dt} = \frac{\partial H}{\partial q}\frac{dq}{dt} + \frac{\partial H}{\partial p}\frac{dp}{dt} = 0$$

  - 位相空間の体積が保存 (Liouville の定理)
    位相空間上の流れの場 $\boldsymbol{v} = (\frac{dq}{dt}, \frac{dp}{dt})$ について

$$\mathrm{div}\boldsymbol{v} = \frac{\partial}{\partial q}\frac{dq}{dt} + \frac{\partial}{\partial p}\frac{dp}{dt} = 0$$

- Euler 法、Runge-Kutta 法などはいずれの性質も満たさない

シンプレクティック数値積分法 (Symplectic Integrator)

- 体積保存を満たす解法
- 例: 調和振動子 $H = \frac{1}{2}(p^2 + q^2)$ の運動方程式

$$\frac{dq}{dt} = p, \quad \frac{dp}{dt} = -q$$

の一方を Euler 法で、他方を逆オイラー法で解く

$$q_{n+1} = q_n + hp_n$$
$$p_{n+1} = p_n - hq_{n+1} = (1 - h^2)p_n - hq_n$$
$$\begin{pmatrix} q_{n+1} \\ p_{n+1} \end{pmatrix} = \begin{pmatrix} 1 & h \\ -h & 1 - h^2 \end{pmatrix} \begin{pmatrix} q_n \\ p_n \end{pmatrix}$$

体積・エネルギーの保存

- 体積保存

$$\det \begin{pmatrix} 1 & h \\ -h & 1-h^2 \end{pmatrix} = 1$$

- エネルギーの保存

$$\frac{1}{2}(p_{n+1}^2 + q_{n+1}^2) + \frac{h}{2} p_{n+1} q_{n+1} = \frac{1}{2}(p_n^2 + q_n^2) + \frac{h}{2} p_n q_n$$

- 位相空間の体積は厳密に保存
- エネルギーは $O(h)$ の範囲で保存し続ける

2 次のシンプレクティック積分法

- ハミルトニアンが $H(p,q) = T(p) + V(q)$ の形で書けるとする
- リープ・フロッグ法

$$p(t+h/2) = p(t) - \frac{h}{2}\frac{\partial V(q)}{\partial q}|_{q=q(t)}$$

$$q(t+h) = q(t) + hp(t+h/2)$$

$$p(t+h) = p(t+h/2) - \frac{h}{2}\frac{\partial V(q)}{\partial q}|_{q=q(t+h)}$$

シンプレクティック積分法

- ハミルトン力学系の満たすべき特性 (位相空間の体積保存) を満たす
- 一般的には陰解法
- ハミルトニアンが $H(p, q) = T(p) + V(q)$ の形で書ける場合は陽的なシンプレクティック積分法が存在する
- エネルギーは近似的に保存する
- n 次のシンプレクティック積分法では、エネルギーは $O(h^n)$ の範囲で振動 (発散しない)

時間依存しないシュレディンガー方程式

- 井戸型ポテンシャル中の一粒子問題

$$\Big[-\frac{\hbar^2}{2m}\frac{d^2}{dx^2}+V(x)\Big]\psi(x)=E\psi(x)$$

$$V(x)=\begin{cases}0 & a\le x\le b\\ \infty & \text{otherwise}\end{cases}$$

- $\hbar^2/2m=1$、$a=0$、$b=1$ となるように変数変換して

$$\Big(\frac{d^2}{dx^2}+E\Big)\psi(x)=0 \qquad 0\le x\le 1$$

を境界条件 $\psi(0)=\psi(1)=0$ のもとで解けば良い

固有値問題の解法

- $x_i = h \times i$ $(h = 1/n)$、$x_0 = 0$、$x_n = 1$ とする
- $\psi(x_0) = 0$、$\psi(x_1) = 1$ を仮定 $(\psi'(x_0) = 1/h$ と与えたことに相当)
- $E = 0$ とおく
- Runge-Kutta 法、Numerov 法などを用いて $x = x_n$ まで積分
- $\psi(x_n)$ の符号がかわるまで、E を少しずつ増やす
- 符号が変わったら、E の区間を半分ずつに狭めていき、$\psi(x_n) = 0$ となる E (固有エネルギー) と $\psi(x)$ (波動関数) を得る

C 言語におけるポインタ

- 変数はメモリ上のどこかに格納されている
  - 変数の値: メモリに格納されている数値
  - アドレス: 変数の値が格納されているメモリ上の番地
- ポインタ変数
  - 値としてアドレスを格納する変数のこと
  - ポインタ変数の値 (アドレス) とポインタ変数のアドレスは異なるものであることに注意
- ポインタ変数の宣言、代入、実体へのアクセス
  - 整数型ポインタ変数の宣言: `int *p;`
  - 整数型変数の宣言: `int q;`
  - 変数 `q` のアドレスをポインタ変数 `p` に代入: `p = &q;`
  - ポインタ変数 `p` に格納されているアドレスに格納されている値の参照 (間接参照): `*p`

ポインタの例 (1)

- 例 3.5.1 (ハンドブック 3.5 節)

```
#include <stdio.h>
int main() {
  int *p;
  int q;
  q = 200;
  p = &q;
  printf("q␣is␣%d␣and␣*p␣is␣%d.\n", q, *p);
  return 0;
}
```

  - q のアドレスを p に代入
  - q と*p の値を出力 → 両者とも 200

ポインタの例 (2)

- 例 3.5.2 (ハンドブック 3.5 節)

```
#include <stdio.h>
int main() {
  int *p;
  int q;
  p = &q;
  *p = 300;
  printf("q␣is␣%d␣and␣*p␣is␣%d.\n", q, *p);
  return 0;
}
```

  - q のアドレスを p に代入
  - *p に 300 を代入 (ここで q=300; と書いても等価)
  - q と*p の値を出力 → 両者とも 300

関数呼び出し (ポインタ渡し)

- 例 3.6.4 (ハンドブック 3.6 節)

```
#include <stdio.h>
void division(int divident, int divisor, int *quotient,
              int *residual) {
  *quotient = divident / divisor;
  *residual = divident % divisor;
}
int main() {
  int josuu = 3;
  int hi_josuu = 13;
  int shou, amari;
  division(hi_josuu, josuu, &shou, &amari);
  printf("%d␣/␣%d␣=␣%d␣...␣%d\n", hi_josuu, josuu,
         shou, amari);
}
```

間違った例 (値渡し)

- 例 3.6.5 (ハンドブック 3.6 節)

```
#include <stdio.h>
void division(int divident, int divisor, int quotient,
  int residual) {
  quotient = divident / divisor;
  residual = divident % divisor;
}
int main() {
  int josuu = 3;
  int hi_josuu = 13;
  int shou, amari;
  division(hi_josuu, josuu, shou, amari);
  printf("%d / %d = %d ... %d\n", hi_josuu, josuu,
         shou, amari);
}
```

  - 誤った答えが出力される。なぜ？

ポインタと一次元配列

- 一次元配列を表す変数は、(実は) 最初の要素を指すポインタ (ハンドブック 3.5.3 節)
  - `v` と `&v[0]` は等価
  - `(v+2)` と `&v[2]` は等価
  - `*v` と `v[0]` は等価
  - `*(v+2)` と `v[2]` は等価
  - `(v+2)[3]` は?
- C 言語では配列の添字は 0 から始まることに注意
- `double v[10];` と宣言した場合、`v[0]` 〜 `v[9]` の 10 個の要素を持つ配列が作られる。`v[10]` は存在しない。値を代入したり参照しようとするとエラーとなる
- ポインタ確認プログラム: pointer.c

一次元配列

- (静的) 一次元配列 (ハンドブック 3.3.1 節)

```
double v[10];
v[0] = 1.0;
v[1] = 2.0;
...
```

要素数はコンパイル時にすでに決まっている定数でなければならない

- (動的) 一次元配列 (ハンドブック 3.11 節)

```
double *v; /* ポインタ */
v = (double*)malloc((size_t)(10 * sizeof(double));
...
free(v); /* 確保した領域を開放 */
```

実行時に要素数を指定可能