微分積分学続論 II ・ プログラミング・ノート （更新 : 2020 年 4 月 18 日）

Karel Švadlenka・京都大学
2020 年 前期

Contents

1　はじめに

Matlab などの計算ソフトウェアは便利だが，計算アルゴリズムがブラックボックスになって見えなかったり，自由に調節できなかったり，難しい問題で失敗したりすることもあるので，自分で数値アルゴリズムを実装して計算過程を透明にすることが望ましい場合がある．

このノートは，数値アルゴリズムをプログラムに写し，計算結果を可視化する方法について基礎的な解説をしている．C 言語をベースにする．マニュアルではないので，これを手がかりにして詳しいことはインターネットなど文献で調べていただきたい．

2　微分方程式の数値解法

コンピュータを用いて，微分方程式を近似的に解く方法を紹介する．コンピュータは有限データしか扱えないため，微分方程式という無限次元の問題（未知関数は無限個の点における値によい決まる）を有限次元の問題に落とす必要がある．これを独立変数（普通は時間）t の離散化により成し遂げる．すなわち，微分方程式を解く時間区間 $(0,T)$ を刻み幅 $\tau = \Delta t = T/N > 0$ をもって有限個（N 個）の部分区間に分割して，数値解はこの区間の節点

$$t_0 = 0, \quad t_1 = \tau, \quad t_2 = 2\tau, \quad \ldots, \quad t_{N-1} = (N-1)\tau, \quad t_N = T$$

のみで求める．この節点の間では数値解は定まっていないので，プロットするときは節点での値の線型補間などを用いる．

数値解法は無数にあるが，以下では，そのなかの代表的なものを 3 つ紹介する．

2.1　Euler 法

Euler 法は最も単純な数値解法である．微分方程式に対する初期値問題

$$\frac{dx}{dt} = f(t,x), \qquad x(0) = x_0 \tag{2.1}$$

を Euler 法で解くとき，時間の刻み幅 τ を決めて，節点 t_n での左辺の微分を

$$\frac{dx}{dt}(t_n) \approx \frac{x(t_{n+1}) - x(t_n)}{\tau} \tag{2.2}$$

のように近似する．微分方程式に代入すると，

$$\frac{x(t_{n+1}) - x(t_n)}{\tau} \approx f(t_n, x(t_n)),$$

つまり，

$$x(t_{n+1}) \approx x(t_n) + \tau f(t_n, x(t_n)).$$

これは，節点 t_n での解の値 $x(t_n)$ を用いて節点 t_{n+1} における解の値 $x(t_{n+1})$ の近似を与える式である．よって，初期条件 $x(0)=x_0$ でわかる t_0 での値よりすべての節点での解の近似値を計算できる．この近似値を $x_n \approx x(t_n)$ と表すと，Euler 法は次のように書ける：

$$x_0 = x(0), \qquad x_{n+1} = x_n + \tau f(t_n, x_n), \quad n = 0, 1, 2, \dots. \tag{2.3}$$

連立方程式の場合，上で x, f をベクトルとして考えて，全く同様に Euler 法を導ける．

例.　連立方程式

$$\begin{aligned} \frac{dx}{dt} &= y \\ \frac{dy}{dt} &= -x \end{aligned} \tag{2.4}$$

では $f(x,y)=(y,-x)^T$ だから，対する Euler 法は

$$\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} x_n \\ y_n \end{pmatrix} + \tau \begin{pmatrix} y_n \\ -x_n \end{pmatrix} = \begin{pmatrix} 1 & \tau \\ -\tau & 1 \end{pmatrix} \begin{pmatrix} x_n \\ y_n \end{pmatrix} \tag{2.5}$$

となる．
解析解が

$$\begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = \begin{pmatrix} \cos t & \sin t \\ -\sin t & \cos t \end{pmatrix} \begin{pmatrix} x(0) \\ y(0) \end{pmatrix}$$

であることを考慮すれば，1 次近似であることが窺える．
これは (2.2) でテーラー展開を用いて

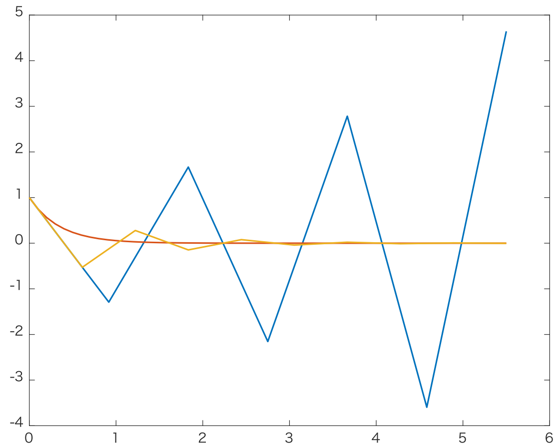
$$\frac{dx}{dt}(t_n) = \frac{x(t_{n+1}) - x(t_n)}{\tau} + O(\tau)$$

であることよりも従う．すなわち，テーラー展開より $x(t_1) - x_1 = O(\tau^2)$ がわかり，この各時間ステップで入る誤差がステップを進めることで積み重なり，累積誤差は $N \cdot O(\tau^2) = O(\tau)$ のように 1 次オーダーとなることが予想できる．正確な証明は，数値解析を扱う本を参考にするとよい（例えば，[2, 3, 1]）．

Euler 法のもう一つの欠点は，安定であるとは限らない，というところである．例えば，

$$x' = -\frac{5}{2}x, \qquad x(0) = 1$$

という初期値問題を三つの異なる時間の刻み幅 τ を用いて Euler 法で解くと，下図のような結果が得られる． $\tau_1 = 0.92$ に対する数値解は青色，$\tau_2 = 0.61$ に対する数値解は黄色，$\tau_3 = 0.1$ に対する数値解はオレンジ色である．これは，刻み幅の取り方によって数値解が発散する場合もあるということを意味する．



Euler 法の精度がよくなく無条件安定ではないため，よりよい性質をもつ解法が開発された．例えば，次節で扱う Runge-Kutta 法やシンプレクティック法である．

以下では Euler 法の細かい解説を行う例を挙げる（上の説明で十分なら，これを無視してもよい）．

例． 簡単な初期値問題

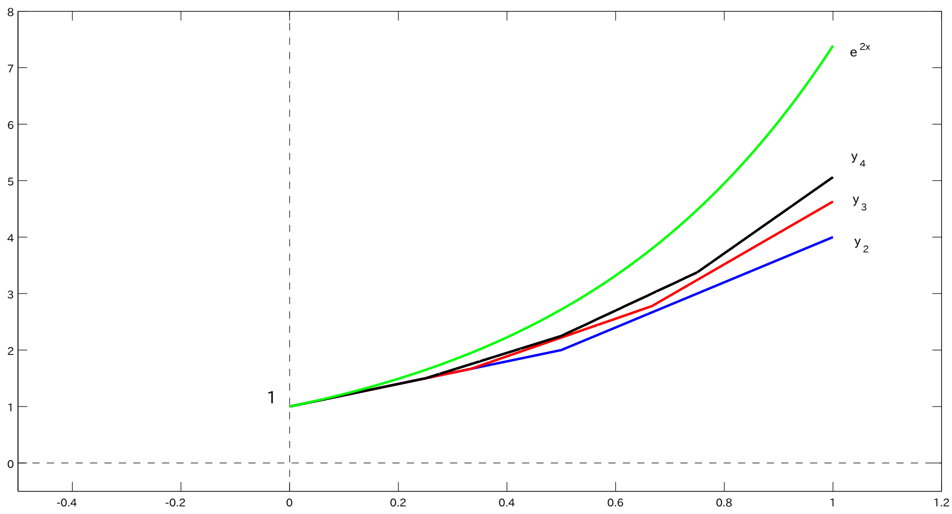
$$x' = 2x, \qquad x(0) = 1 \tag{2.6}$$

について考える．解は明らかに $x(t) = e^{2t}$ である．区間 $t \in (0,1)$ で近似的に解いてみよう．区間 $(0,1)$ を N 等分して，部分区間の長さを $\tau = 1/N$ とする．これにより得られた節点を $t_0 = 0, t_1 = \tau, t_2 = 2\tau, \dots, t_{N-1} = (N-1)\tau, t_N = 1$ とおく．

まず，$N = 2$ のときのとても粗い近似解を求める．そのために，初期点 $(0,1)$ からスタートして，微分方程式の方向の場が定める方向（つまり，解曲線への接線の方向）に沿って $t = 0.5$ の軸にぶつかるまで直線上に進む．点 $(t_0, x_0) = (0,1)$ における接線方向は $(1, x'(t_0)) = (1, 2x_0) = (1,2)$ であるので，進む直線の方程式は $x = 2t + 1$ となる．節点 t_1 が定める軸 $t = t_1 = 0.5$ にぶつかったら，進む方向を修正する．つまり，ぶつかった点 $(t_1, x_1) = (0.5, 2)$ での方向の場の方向に切り替える．点 $(0.5, 2)$ での方向の場の方向は $(1, x'(0.5)) = (1, 2x_1) = (1, 4)$．よって，次は点 $(0.5, 2)$ からこの方向に進むので，進む直線の方程式は $x = 4t$ となる．最後の節点 $t_2 = 1$ にぶつかるまでこの直線上に進む．この手順で得られた近似解は

$$y_2(t) = \begin{cases} 2t+1, & t \in [0, 0.5] \\ 4t, & t \in [0.5, 1] \end{cases}$$

となる．このグラフは下の図で青で表示してある．



同様に，$N = 3$ の場合に近似解 y_3 を計算する．同じ初期点 $(0,1)$ からスタートするので，$t \in (0, \frac{1}{3})$ の間では y_2 と同じ直線 $x = 2t+1$ の方向に進む．点 $(t_1, x_1) = (\frac{1}{3}, \frac{5}{3})$ での方向は $(1, 2x_1) = (1, \frac{10}{3})$ であるので，区間 $t \in (\frac{1}{3}, \frac{2}{3})$ での直線の方程式は $x = \frac{10}{3}t + \frac{5}{9}$ となる．最後の区間 $(t_2, t_3) = (\frac{2}{3}, 1)$ での近似解の方程式を求め，一つの式にまとめると，

$$y_3(t) = \begin{cases} 2t+1, & t \in [0, \frac{1}{3}] \\ \frac{10}{3}t + \frac{5}{9}, & t \in [\frac{1}{3}, \frac{2}{3}] \\ \frac{50}{9}t - \frac{25}{27}, & t \in [\frac{2}{3}, 1] \end{cases}$$

となる．この近似解を図で赤で示した．

黒で示したのは近似解 y_4 であり，緑の曲線は解析解 $x(t) = e^{2t}$ である．刻み幅を細かくしていくと近似解が解析解に近づくことがわかる．それを確認するために，任意の分割数 N に対して例えば点 $t = 1$ での近似解 y_N の値を求める．上で見たように

$$y_N(x) = 2t + 1, \qquad t \in (0, \tau)$$

である．$t = \tau$ で $y_N(\tau) = 2\tau + 1$ となるので，点 $(t_1, x_1) = (\tau, 2\tau+1)$ での方向の場の方向を計算する：$(1, 2x_1) = (1, 2(2\tau+1))$．区間 $(\tau, 2\tau)$ での近似解はこの方向をもち，点 $(\tau, 2\tau+1)$ を通る直線なので

$$y_N(t) = 2(2\tau+1)t - 2(2\tau+1)\cdot\tau + 2\tau + 1 = 2(2\tau+1)t + (1-2\tau)(1+2\tau), \qquad t \in (\tau, 2\tau)$$

もう一つの区間を計算する．$t = 2\tau$ での近似解の値は $y_N(2\tau) = 2(2\tau+1)2\tau + (1-2\tau)(1+2\tau) = (4\tau+1-2\tau)(2\tau+1) = (2\tau+1)^2$ であるので，$(t_2, x_2) = (2\tau, (2\tau+1)^2)$ での方向は $(1, 2(2\tau+1)^2)$ となる．つまり，

$$y_N(t) = 2(2\tau+1)^2 t - 2(2\tau+1)^2 2\tau + (2\tau+1)^2 = 2(2\tau+1)^2 t + (1-4\tau)(2\tau+1)^2, \qquad t \in (2\tau, 3\tau)$$

次の区間を計算すると，

$$y_N(t) = 2(2\tau+1)^3 t + (1-6\tau)(2\tau+1)^3, \qquad t \in (3\tau, 4\tau)$$

を得るので，近似解の一般的な形が

$$y_N(t) = 2(2\tau+1)^k t + (1-2k\tau)(2\tau+1)^k, \qquad t \in (k\tau, (k+1)\tau)$$

になると推測できる．（正確に示す場合は数学的帰納法を用いる.）

$y_N(1)$ を計算してみよう．

$$y_N(1) = y_N(N\tau) = 2(2\tau+1)^{N-1} + (1-2(N-1)\tau^2)(2\tau+1)^{N-1} = (2\tau+1)^N$$

ここで，分割数 N を無限に大きくする．$\tau = 1/N$ に注意して，

$$\lim_{N\to\infty} y_N(1) = \lim_{N\to\infty} (\tfrac{2}{N}+1)^N = \lim_{N\to\infty} e^{N\log(\frac{2}{N}+1)} = e^{\lim_{N\to\infty} 2\frac{\log(1+\frac{2}{N})}{\frac{2}{N}}} = e^2.$$

やはり，真の解の $t=1$ での値に近づく．
t 方向の分割を細かくすればするほど，近似解が正確な解に近くなることがわかった．

2.2 Runge-Kutta 法

様々な Runge-Kutta 法があるが，ここではよく使われる RK4 を紹介する．

$$\frac{dx}{dt} = f(t,x), \qquad x(0) = x_0 \tag{2.7}$$

の RK4 近似解は次のアルゴリズムで得られる：

$$x_{n+1} = x_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

ただし，

$$\begin{aligned}
k_1 &= \tau f(t_n, x_n) \\
k_2 &= \tau f\left(t_n + \frac{\tau}{2}, x_n + \frac{k_1}{2}\right) \\
k_3 &= \tau f\left(t_n + \frac{\tau}{2}, x_n + \frac{k_2}{2}\right) \\
k_4 &= \tau f(t_n + \tau, x_n + k_3)
\end{aligned}$$

この方法の局所（1 ステップの）誤差は $O(\tau^5)$，累積誤差は $O(\tau^4)$ である．安定性は Runge-Kutta 法のバージョンによるが，陰的なスキームの方が安定である．

例 (2.4) の連立方程式に適用すると，

$$\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} 1-\frac{1}{2}\tau^2+\frac{1}{24}\tau^4 & \tau-\frac{1}{6}\tau^3 \\ -\tau+\frac{1}{6}\tau^3 & 1-\frac{1}{2}\tau^2+\frac{1}{24}\tau^4 \end{pmatrix} \begin{pmatrix} x_n \\ y_n \end{pmatrix} \tag{2.8}$$

2.3　シンプレクティック法

例 (2.4) のハミルトン系の解 $(x(t), y(t))$ はエネルギー

$$H(t) = \frac{1}{2}x^2(t) + \frac{1}{2}y^2(t)$$

を時間とともに保存する．実際，$(x(t), y(t))$ が解ならば，

$$\frac{d}{dt}H(t) = x(t)x'(t) + y(t)y'(t) = x(t)y(t) + y(t)\cdot[-x(t)] = 0.$$

数値解法 (2.5), (2.8) がエネルギーを保存するか，確認する．どのスキームも

$$\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} a & b \\ -b & a \end{pmatrix} \begin{pmatrix} x_n \\ y_n \end{pmatrix}$$

の形をしているので，

$$x_{n+1}^2 + y_{n+1}^2 = (ax_n+by_n)^2 + (-bx_n+ay_n)^2 = (a^2+b^2)(x_n^2+y_n^2) = \det\begin{pmatrix} a & b \\ -b & a \end{pmatrix}(x_n^2+y_n^2)$$

Euler 法の場合，

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

なので，数値解のエネルギーは増大する（計算を十分長く続ければ，エネルギーは任意の値を超えてしまう）．
Runge-Kutta 法の場合，

$$\det\begin{pmatrix} 1-\frac{1}{2}\tau^2+\frac{1}{24}\tau^4 & \tau-\frac{1}{6}\tau^3 \\ -\tau+\frac{1}{6}\tau^3 & 1-\frac{1}{2}\tau^2+\frac{1}{24}\tau^4 \end{pmatrix} = 1-\frac{1}{72}\tau^6+\frac{1}{24^2}\tau^8$$

なので，数値解のエネルギーは減少する．

そこで，ハミルトン系では，エネルギーを（小さい誤差を除いて）保存するような数値解法をシンプレクティック数値解法（symplectic）という．一般の微分方程式の場合でも同じ考え方があり，解のなんらかの性質を受け継ぐ数値解法を構造保存解法という．

例えば，ハミルトニアンが $H(x,y) = T(y) + V(x)$ という x, y が分離される形をしているとき，

数値スキーム

$$\begin{aligned} x_{n+1} &= x_n + \tau T'(y_n) \\ y_{n+1} &= y_n - \tau V'(x_{n+1}) \end{aligned}$$

はシンプレクティックである．上の $H(x,y) = \frac{1}{2}x^2 + \frac{1}{2}y^2$ の例に適用すると，

$$\begin{aligned} x_{n+1} &= x_n + \tau y_n \\ y_{n+1} &= y_n - \tau x_{n+1} = y_n - \tau(x_n + \tau y_n) = (1-\tau^2)y_n - \tau x_n \end{aligned}$$

となり，係数行列の行列式は

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

だから，正準変換を表し，エネルギーの計算値と厳密な値の誤差が蓄積することなく, あるオーダーの中に留まることが証明できる．

3　プログラミングについて

数値解法に基づいたプログラムを組み，プログラムを実行した結果をプロットする流れについていくつかコメントをする．様々なプログラミング言語があるが，ここでは C 言語を例として使う．

3.1　準備

C 言語でプログラムを作成するには

- プログラムのコードを入力するための**テキストエディタ**
- プログラムから実行ファイルを生成する **C コンパイラ**
- ボタン一つでコンパイルしたり，様々な便利な機能をもつ**統合開発環境**

が必要になる．これらをすべてまとめたソフトウェアもある．

それぞれのソフトの特徴やインストール方法についてはネットで調べるとわかりやすいだろう．Windows では，Visual Studio が定番 [4] だが，他にもたくさんのオプションがあるので，[5, 6] のようなサイトを見て，自分に合った環境を決めるとよい．Mac OS は Xcode がお勧めである [7]．インストール方法などアドバイスは「Xcode インストール」などとネット検索するとよい．

3.2 プログラム

以下では Euler 法の C 言語のプログラムのサンプルを挙げる．他の数値解法はこれより大幅に高度なプログラミングのスキルを要しないので，このサンプルを修正するだけで実装できるだろう．

このプログラムでは初期値問題

$$x' = -\frac{5}{2}x, \qquad x(0) = 1$$

を時間区間 $(0, 5)$ を 20 等分して解いている．

```
#include<stdio.h>
double t0 = 0.0; // 初期時刻
double x0 = 1.0; // 初期時刻での解の値
double tfin = 5.0; // 方程式を解く最終時刻
int N = 20; // 分割数
FILE *eulerdata; // 出力ファイル
double fun(double t,double x) // 微分方程式の右辺関数 f(t,x)
{
  double f;
  f = -2.5*x;
  return f;
}
main(){
  double t,x,tau;
  int n;
  tau = (tfin-t0)/N; // 時間の刻み幅
  t = t0;
  x = x0;
  eulerdata = fopen ( "result_euler.dat", "w" ); // 出力ファイルを開く
  fprintf(eulerdata,"%.17f\t%.17f\n",t,x); // 初期値をファイルに出力
  for (n=0;n<N;n++){
    x += tau*fun(t,x); // オイラー法の式
    t += tau;
    fprintf(eulerdata,"%.17f\t%.17f\n",t,x);
                              // 各時刻の t,x データをファイルに出力
  }
  fclose(eulerdata); // 出力ファイルを閉じる
}
```

初期時刻 t_0，初期値 x_0，分割数 N，最終時刻 T，微分方程式の右辺 $f(t,x)$ をプログラムの前半で自由に変えられるようになっているが，複数の数値実験のデータを一つの図にプロットするようなときは出力ファイルの設定などに工夫が必要である．

3.3　可視化

前節のプログラムを実行すると，`result_euler.dat` というデータファイルが生成される．このファイルには離散時刻 t_n とその時刻における数値解 x_n というデータが行ごとに分かれて保存されている．このデータをプロットして可視化する方法を二つ述べる．どの方法を使うにしても，上記のデータファイルを working directory にコピーする必要がある．

- Matlab を使う方法：次のコマンドでデータをロードし，プロットできる．

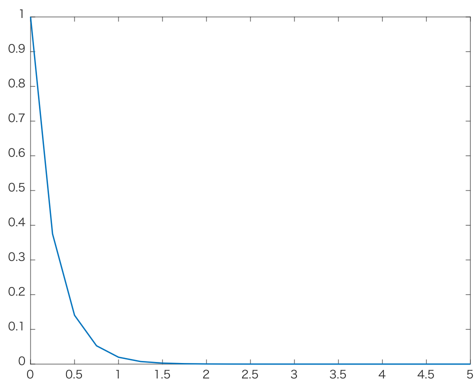
  ```
  data = importdata('result_euler.dat') ;
  plot(data(:,1),data(:,2));
  ```

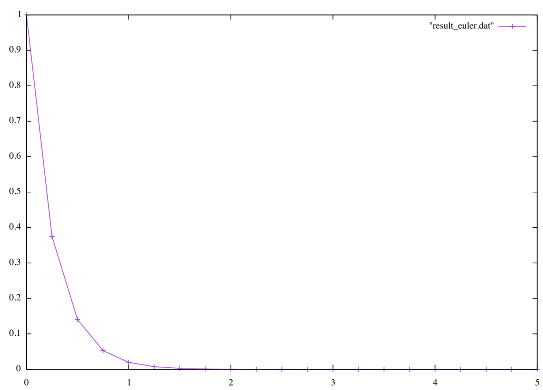
- Gnuplot を使う方法：Gnuplot はどのシステムでも使える無料のソフトで，Matlab などの有料ソフトが利用できない場合，お勧めである．[8] よりダウンロードできる．なお，使い方については [9] などのネットリソースが活用できる．

  インストールしてから，コマンドターミナルで次のコマンドを打てば，グラフが表示される（データファイルが Desktop にあるという設定）：

  ```
  gnuplot
  cd "Desktop"
  plot "result_euler.dat" with linespoints
  ```

それぞれのグラフは以下のようになる（左：Matlab，右：Gnuplot）．





References

[1] A. Quarteroni, R. Sacco, F. Saleri, *Numerical Mathematics*, Springer, 2007.

[2] 菊地 文雄，齊藤 宣一,「数値解析の原理」，岩波書店，2016.

[3] 齊藤 宣一,「数値解析入門」， 東京大学出版会，2012.

[4] https://www.microsoft.com/ja-jp/dev/default.aspx

[5] http://www.ooyashima.net/db/prog.htm

[6] https://gabekore.org/windows-c-dev-tool

[7] https://developer.apple.com/jp/xcode/

[8] http://www.gnuplot.info

[9] http://www.cse.kyoto-su.ac.jp/˜oomoto/lecture/program/gnuplot/gnuplot.html