情報処理 II No.6

常微分方程式の初期値問題の数値解法 (1)

桂田 祐史

1995 年 6 月 1 日

1 はじめに

1.1 常微分方程式って何だったっけ — 復習

常微分方程式というのは大雑把に言うと、「**一つの実独立変数 t の未知関数 $x = x(t)$ を求めるための問題で、x とその導関数 $\dfrac{dx}{dt}$, $\dfrac{d^2x}{dt^2}$, $\cdots$, $\dfrac{d^kx}{dt^k}$ についての方程式になっているもの**」のことです。(以下では $\dfrac{dx}{dt} = x'$, $\dfrac{d^2x}{dt^2} = x''$, $\dfrac{d^3x}{dt^3} = x^{(3)}$, $\cdots$ のような書き方もします。)

例 1: $x'(t) = f(t)$ (f は既知関数)
例 2: $x''(t) = -g$ (g は既知の定数)
例 3: $x'(t) = ax(t)$ (a は既知の定数)
例 4: $x''(t) = -\omega^2 x(t)$ (ω は既知の定数)
例 5: $x''(t) + 2\gamma x(t) + \omega^2 x(t) = 0$ (γ, ω は既知の定数)

ここに例としてあげた方程式はいずれも割とポピュラーなものなのですが、見覚えがあるでしょうか？どの場合もこれらの方程式だけでは解が一つに定まらず、何らかの条件を付け足すことによって初めて解が決定されます。その条件として、ある特定の t の値 t_0 に対する x の値 $x(t_0)$ や導関数の値を指定するというタイプのものがよくありますが、そういうものを**初期条件**と呼びます (これは t が時刻を表す変数で、t_0 を現象が始まる時刻のように解釈するからでしょう)。

例えば、上の例 1, 3 に対して

$$x(0) = x_0 \quad (x_0 \text{ は既知定数}),$$

例 2, 4, 5 に対して

$$x(0) = x_0, \quad x'(0) = v_0 \quad (x_0, v_0 \text{ は既知定数})$$

のように与えられた条件が初期条件です。また、初期条件を添えて解が決定されるようにした問題を、(常微分方程式の) **初期値問題**と言います。

1.2 これからの目標

微分方程式は、他の諸科学への応用のみならず[1]、数学それ自体にとっても非常に重要です[2]。ところが困ったことに、微分方程式は大抵の場合に、良く知られている関数で解を表現することが出来ません。これは解が存在しないということではありません。解はほとんどいつでも存在するけれども、それを簡単な演算 (不定積分を取る、四則演算、逆関数を取る、初等関数に代入するなど) で求める — いわゆる**求積法**で解く — ことは、よほど特殊な問題でない限り出来ない、ということです。

この困った状況をある程度解決するのが、解を数値的に求める方法です。この情報処理 II では、いくつかの基本的な数値解法を学んで、実際に常微分方程式を解いてみます。これは計算機による数値シミュレーション[3]の典型例と呼べるものですし、マスターしておくと役に立ちます。

2 数値解法 (1)

2.1 問題の設定と数値解法の基本原理

常微分方程式としては正規形[4]のもののみを扱います。後で例で見るように、高階の方程式も一階の方程式に帰着されますから、当面一階の方程式のみを考えます。独立変数を t、未知関数を $x = x(t)$ とすれば、一階正規形の常微分方程式とは

$$\frac{dx}{dt} = f(t, x) \quad (t \in [a, b]) \tag{1}$$

の形に表わされる方程式のことです。ここで f は既知の関数です。初期条件としては

$$x(a) = x_0 \quad (x_0 \text{ は既知定数}) \tag{2}$$

の形のものを考えます。x_0 は既知の定数です。(1),(2) を同時に満たす関数 $x(t)$ を求めよ、というのが一階正規形常微分方程式の初期値問題です。この時関数 $x(t)$ を初期値問題 (1),(2) の解と呼びます。

常微分方程式の数値解法の基本的な考え方は次のようなものです。「問題となっている区間 $[a, b]$ を

$$a = t_0 < t_1 < t_2 < \cdots < t_N = b$$

と分割して、各 "時刻" t_j での x の値 $x_j = x(t_j)$ $(j = 1, 2, \ldots, N)$ を近似的に求めることを目標とする。そのために微分方程式 (1) から $\{x_j\}_{j=0,\ldots,N}$ を解とする適当な **差分方程式**[5]を作り、それを解く。」

---

[1]微分方程式は物理学の問題を扱うために発明されましたが、現在では自然科学以外でも応用されています。

[2]常微分方程式の簡単なものは高等学校でも学びましたし、1 年次にも微分方程式という授業がありました。数学科の 3 年次にもより詳しいことを学ぶための講義があります。常微分方程式に対する参考書は色々ありますが、例えば、 3 年生向けの講義の教科書になっている、笠原晧司著「微分方程式の基礎」朝倉書店、をあげておきます。

[3]simulation(模擬実験) を「シュミレーション」と読み間違えないでください。「シミュレーション」ですからね。

[4]方程式が最高階の導関数について解かれている、ということですが、よく分からなくても差し支えありません。

[5]漸化式のようなものだと思って構いません。差分とは、高等学校の数列で言う階差のことです。

区間 $[a,b]$ の分割の仕方ですが、以下では簡単のため N 等分することにします。つまり

$$h = (b-a)/N, \quad t_j = a + jh.$$

となります。

2.2 Euler(オイラー) 法の紹介

微分 $x'(t) = \dfrac{dx}{dt}$ は差分商 $\dfrac{x(t+h)-x(t)}{h}$ の $h \to 0$ の極限です。そこで、(1) 式の微分を差分商で置き換えて近似することによって、次の方程式を得ます。

$$\frac{x_{j+1}-x_j}{h} = f(t_j, x_j) \quad (j = 0, 1, \ldots, N-1)$$

変形すると

$$x_{j+1} = x_j + hf(t_j, x_j). \tag{3}$$

これを漸化式として使って、x_0 から順に x_1, x_2, …, x_N が計算出来ます。この方法を **Euler(オイラー) 法**と呼びます。

こうして得られる $N+1$ 個の点 (t_j, x_j) を順に結んで得られる折れ線関数は（f に関する適当な仮定のもとで）$N \to +\infty$ の時（$h = (b-a)/N$ について言えば $h \to 0$）、真の解 $x(t)$ に収束することが証明できます[6]。ここでは簡単な例で収束を確かめてみましょう。

2.3 プログラミングの仕方

初期値 x_0 が与えられたとき、漸化式 (3) によって、数列 $\{x_j\}_{j=1,\cdots,N}$ を計算するプログラムはどう作ったらよいでしょうか？ここでは二つの素朴なやり方を紹介しましょう。

配列を使う方法 数列を配列で表現するのは、Fortran では自然な発想です。例えば

```
integer MAXN
parameter (MAXN = 1000)
real x(0:MAXN)
```

のように配列 “x” を用意しておいて

```
x(0) = x0
t = a
do j=0,N-1
  x(j+1) = x(j) + h * f(t,x(j))
  t = t + h
end do
```

とするわけです。

[6]現在の数学科のカリキュラムでは 3 年次に開講されている常微分方程式の講義で学びます。

配列を使わないですませる方法 漸化式 (3) を解くために、配列は絶対必要というわけではありません[7]。例えば、変数 "x" に各段階の x_j の値を収めておくとして

```
x = x0
t = a
do j = 0,N-1
  x = x + h * f(t,x)
  t = t + h
end do
```

のようなプログラムで計算が出来ます。

重箱の隅をつつく注意: "t = t + h" とすると、誤差が蓄積されがちです。これを避けるには、次のように書き換えると良いでしょう。

```
t = a + (j + 1) * h
```

2.4 例題

例 1: 初期値問題

$$x'(t) = x(t) \quad (t \in [0,1]), \quad x(0) = 1$$

の解は $x(t) = e^t$ であるが、Euler 法を用いて解くと $x_N = \left(1 + \dfrac{1}{N}\right)^N$ となる。したがって、確かに $N \to +\infty$ の時に $x_N \to e = x(1)$ となっている。

例題 6.1 Euler 法を用いて、例 1 の初期値問題を解くプログラムを作って収束を調べよ。

```
* reidai6-1.f -- 微分方程式の初期値問題を Euler 法で解く
      program rei61
*       開始時刻と終了時刻
      real a,b
      parameter (a=0.0,b=1.0)
*       変数と関数の宣言
      integer N,j
      real t,x,h,x0,f
      external f
*       初期値
      x0 = 1.0
*      区間の分割数 N を入力してもらう
      write(*,*) ' N='
      read(*,*) N
*      小区間の幅
      h=(b-a)/N
*      開始時刻と初期値のセット
      t=a
      x=x0
```

[7]配列はメモリーを消費しますし、(特に Fortran の場合、配列の大きさは実行時に変更できないので) プログラムを書く際に、どれくらいの大きさの配列を用意したらいいのかという問題に悩まなくてはなりません。

```
      write(*,*) t,x
*      Euler 法による計算
      do j=0,N-1
         x=x+h*f(t,x)
         t=t+h
         write(*,*) t,x
      end do
      end
************************************************************************
*       微分方程式 x'=f(t,x) の右辺の関数 f の定義
      real function f(t,x)
      real t,x
      f=x
      end
```

このプログラムをコンパイルして実行すると、分割数 N を尋ねてきますので、色々な値を入力して試してみて下さい。各時刻 t_j における x_j の値 $(j = 0, 1, \cdots, N)$ を画面に出力します。

確認用にいくつかの N の値に対する場合の、$x(1) = x_N$ の値を書いておきます。$N = 10$ の場合 $x_N = 2.59374261$, $N = 100$ の場合 $x_N = 2.70481372$, $N = 1000$ の場合 $x_N = 2.71692038$, $N = 10000$ の場合 $x_N = 2.71814346$, $\cdots$.

分割数 N が大きくなるほど、真の値 $e = 2.7182818284590452\cdots$ に近付いていくはずです。

問題 6.1: 例題 6-1 とは異なる初期値問題を適当に設定して、それを Euler 法で解いてみよ。(結果についてもきちんと吟味すること。)

問題 6.2: "reidai6-1.f" の出力するデータを用いて、解曲線 (関数 $t \mapsto x(t)$ のグラフのこと) を描きなさい。

2.5 Euler 法の収束の速さ

先ほどの実験では Euler 法による解は N が大きくなればなるほど真の解に近くなるはずです。実際 $N \to +\infty$ とすると真の解に収束することが証明できるわけなのですが、それでは、どれくらいの速さで収束するのでしょうか？これを実験的に調べてみましょう。

例題 6.2: 例題 6.1 と同じ初期値問題で、色々な分割数 N に対して問題を時、$t = 1$ での誤差の大きさ $|x(1) - x_N| = |e - x_N|$ について調べよ。

こういう場合は、色々な N に対して一斉に $|e - x_N|$ を計算するプログラムを作るのがいいでしょう。

```
* reidai6-2.f --- 常微分方程式の初期値問題を Euler 法で解く
      program rei62
*       開始時刻と終了時刻
      real a,b
      parameter (a=0.0,b=1.0)
*       初期値
      real x0
*       変数と関数の宣言
      real t,x,h,f,e
      integer N0,N1,N2,N,i
*       自然対数の底 e (=2.7182818284..)
```

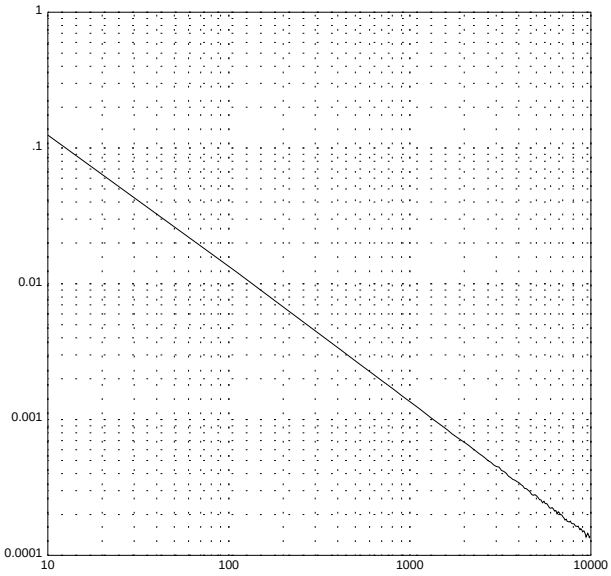
```
      e=exp(1.0)
*       初期値の設定
      x0=1.0
*       どういう N について計算するか？
*       N0 から N1 まで、N2 刻みで増える N に
      write(*,*) ' FIRSTN,LASTN,STEPN='
      read(*,*) N0,N1,N2
*       計算の開始
      do N=N0,N1,N2
         h=(b-a)/N
*       開始時刻と初期値のセット
         t=a
         x=x0
*       Euler 法による計算
         do i=0,N-1
            x=x+h*f(t,x)
            t=t+h
         end do
         write(*,*) N,abs(e-x)
      end do
      end
************************************************************************
*       微分方程式 x'=f(t,x) の右辺の関数 f の定義 -- 前と同じだから略
```

コンパイルして実行すると “ FIRSTN,LASTN,STEPN=” と尋ねてきます。例えば “100 1000 50” と答えると 100 から 1000 まで 50 刻みで増やして行った値 100, 150, 200, $\cdots$, 900, 950, 1000 を N として計算して、最終的に得られた誤差を出力します。実行結果を見てみましょう。

```
waltz11% reidai6-2
 FIRSTN,LASTN,STEPN=
100 1000 50
  100    1.34680271e-02
  150    9.00602341e-03
  200    6.76608086e-03
  .... (途中略) ...
  900    1.50966644e-03
  950    1.42812729e-03
  1000    1.36137009e-03
waltz11%
```

“mgraph” コマンドには、**対数目盛**によるグラフを描く機能 (起動時に -lx, -ly を指定します。ここで “l” は “logarithmic”(=「対数の」) の頭文字です）があります。この場合は、例えば次のようにしたりすると良いでしょう。

```
waltz11% reidai6-2 | mgraph -lx -ly | xplot
10,10000,100
```



このグラフから、誤差 $= O(N^{-1})$ $(N \to +\infty)$ であることが読みとれます。実はこれは Euler 法の持つ一般的な性質です。

実は Euler 法は収束があまり速くないので、実際には特殊な場合を除いて使われていません。そこで、、、、

3 Runge–Kutta (ルンゲ–クッタ) 法

前節で解説した Euler 法は簡単で、これですべてが片付けば喜ばしいのですが、残念ながらあまり効率が良くありません。高精度の解を計算するためには、分割数 N をかなり大きく取る (=大量の計算をする) 必要があります。特別な場合[8]を除けば、実際に使われることは滅多にないでしょう。率直にいって実用性は低いです。

より高精度の公式は、現在まで様々なものが開発されていますが、比較的簡単で、精度がまあまあ高いものに Runge–Kutta 法と呼ばれるものがあります。それは x_j から x_{j+1} を求める漸化式として次のものを用います。

$$\begin{aligned} k_1 &= hf(t_j, x_j) \\ k_2 &= hf(t_j + h/2, x_j + k_1/2) \\ k_3 &= hf(t_j + h/2, x_j + k_2/2) \\ k_4 &= hf(t_j + h, x_j + k_3) \\ x_{j+1} &= x_j + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \end{aligned}$$

これがどうやって導かれたものかは解説しません。まずは使ってみましょう。

```
* reidai6-3.f -- 微分方程式の初期値問題を Runge-Kutta 法で解く
      program rei63
```

[8]例えば微分方程式の右辺に現れる関数 f が、解析的な式で定義された滑らかなものではなく、実験により計測されたデータにより定義されている場合等は、高精度の公式を用いるよりも、Euler 法の方が良いことがあります。

```
*       開始時刻と終了時刻
      real a,b
      parameter (a=0.0,b=1.0)
*       変数と関数の宣言
      integer N,j
      real t,x,h,x0,f,k1,k2,k3,k4
      external f
*       初期値
      x0 = 1.0
*      区間の分割数 N を入力してもらう
      write(*,*) ' N='
      read(*,*) N
*      小区間の幅
      h=(b-a)/N
*      開始時刻と初期値のセット
      t=a
      x=x0
      write(*,*) t,x
*      Runge-Kutta 法による計算
      do j=0,N-1
         k1 = h * f(t, x)
         k2 = h * f(t + h / 2, x + k1 / 2)
         k3 = h * f(t + h / 2, x + k2 / 2)
         k4 = h * f(t + h, x + k3)
         x = x + (k1 + 2 * k2 + 2 * k3 + k4) / 6
         t = t + h
         write(*,*) t,x
      end do
      end
************************************************************************
*       微分方程式 x'=f(t,x) の右辺の関数 f の定義 --- これも前と同じだから略
```

コンパイル・実行の結果は次のようになるはずです。

```
waltz11% f77o reidai6-3.f
Fortran 90 diagnostic messages: program name(f)
  jwd2008i-i  ``reidai6-3.f'', line 30: この仮引数は,副プログラム中で使用されてい
ま
せん.(名前:t)
waltz11% reidai6-3
  N=
10
 0.000000000e+00 1.00000000
 0.100000001 1.10517085
 0.200000003 1.22140265
 0.300000012 1.34985852
 0.400000006 1.49182427
 0.500000000 1.64872062
 0.600000024 1.82211792
 0.700000048 2.01375151
 0.800000072 2.22553945
 0.900000095 2.45960140
```

```
 1.00000012 2.71827984
waltz11%
```

たった 10 等分なのに相対誤差が 10^{-6} 以下になっています ($\because x(1) = e^1 = e = 2.7182818284\cdots$ であるので、相対誤差$=|e-2.71827984|/e = 7.31\cdots\times10^{-7}$)。Runge–Kutta 法の公式は Euler 法よりは大部面倒ですが、それに見合うだけの価値があることが納得できるでしょう。

問題 6-3: "`reidai6-3.f`" を普通の "`real`"(単精度実数型) の代わりに "`real*8`" (倍精度実数型) を使うように書き換えて、大きな N に対してどうなるか、実験しなさい。倍精度化のための書き換えは、具体的には、(i) プログラムの中に何箇所かある "`real`" を "`real*8`" に書き換える、(ii) 実数の定数を "`0.0`" → "`0.0d0`" のように、末尾に "`d0`" を付けたりして[9]、倍精度実数定数に書き換える、 (iii) "`real`" や "`complex`" への型変換を指定しているとことがあれば、それぞれ "`real*8`", "`complex*16`" への型変換に書き換える、というようなことをやればいいが、今の場合は (i) だけでも十分動作するプログラムになる。

問題 6-4: 区間の分割数 N を変えながら

$$x'(t) = x \quad (t \in [0,1]), \quad x(0) = 1$$

を Runge–Kutta 法を用いて解くプログラムを作り、$t = 1$ での誤差 $|x_N - x(1)|$ を調べよ (前節 "`reidai6-2.f`" の真似をする)。Euler 法と比べてどうなるか？

4 対数グラフに関するメモ

中学・高校で、データをグラフにプロットして、正比例の関係にあることを確かめ、さらには比例定数を求める、ということをした経験があるでしょう。ここでは $y = cx^\alpha$ のような関数について、同じことをするにはどうしたら良いか、説明します。これには両辺の対数を取って、

$$\log y = \log(cx^\alpha) = \log c + \log x^\alpha = \log c + \alpha \log x.$$

これから、$\log y = Y$, $\log x = X$, $\log c = C$ とおくと

$$Y = \alpha X + C.$$

そこで、データ (x_j, y_j) に対し、$(\log x_j, \log y_j)$ を座標に持つ点をプロットすると、点は直線上に並び、傾きは α になるはずです。—— これが対数グラフの原理です。対数グラフは簡単ですが、便利で役立つものです。

問題 6-5: $y = a^x$ のようなものは、どう調べたらいいか考え、実験して確かめよ。

[9] "`d` 整数" は "`e` 整数" のような指数形式の実数表現の指数部を表します。よって "`1.0e-7`" のような定数は "`1.0d-7`" のように書き換えます。