4　常微分方程式の解法

4.1　はじめに

工学系の諸問題では常微分方程式を解くことが多い。解きたい方程式が線形であれば比較的簡単に解くことが出来るが、一般には非線形であることが多く数値計算で解かなければならない。ここでは、常微分方程式の数値解法として Runge(ルンゲ)–Kutta(クッタ)法を学ぶ。

4.2　Euler 法

1 階の常微分方程式は一般に

$$\frac{dy}{dx} = f(x, y) \qquad (1)$$

という形で表すことが出来る。例えば右辺の関数 f が $f(x, y) = -2xy$ で与えられるなら、点 $(x, y) = (0, 1)$ では傾きは $\frac{dy}{dx} = -2xy = 0$、点 $(1/2, 1)$ では傾きは -1、点 $(1, 1)$ での傾きは -2 等々という具合に x–y 面内の各点での局所的な傾きが計算できる。図 1 は、この様にして得た各点での傾きを線分で示したものである。この図を見ると、原点 $(0, 0)$ を通る解はずっと $y = 0$ の x-軸上を進み、y-軸上の原点以外の点から出発した点も次第に x-軸に近付いて行くことが視覚的に理解できる。

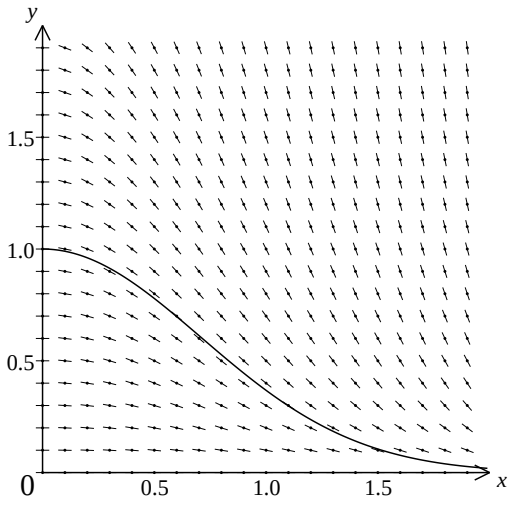

図 1: 傾きの「場」の概念図。

では計算機で微分方程式を解くにはどうしたら良いだろうか。初期条件 (x_0, y_0) が与えられた時に上の微分方程式を解くことを考えよう。初期値から出発して $x_1 = x_0 + h$ における値 y_1 を計算する。求める関数の点 (x_0, y_0) での傾きは $f(x_0, y_0)$ なので、微分の定義から、h が小さければ $(y_1 - y_0)/h \sim f(x_0, y_0)$ が成り立つだろう。この式から $y_1 \sim y_0 + hf(x_0, y_0)$ として x_1 での関数の値が予測できる。

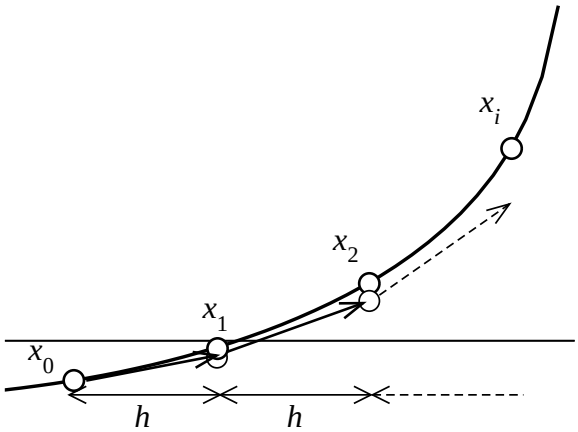

図 2: Euler 法による微分方程式の数値解法。

同様に (x_1, y_1) での傾きが $f(x_1, y_1)$ であることから $x_2 = x_1 + h$ における y の値は $y_2 = y_1 + hf(x_1, y_1)$ と計算できる。この手順を繰り返して多数の点を推定して、ついには点 (x_0, y_0) を通る微分方程式の近似的な解を求めることができる。h は数値解の精度に対する要求により決まる「きざみ値」である。この方法は Euler(オイラー)法と呼ばれる。

図 2 を見ると分るように、$x = x_i$ と x_{i+1} の間にも傾きの値 $f(x, y)$ は刻々変化するのに、この近似では h の間一定の傾きで y の増分を求めている。そのため単純な Euler 法は精度が高い計算方法ではない。

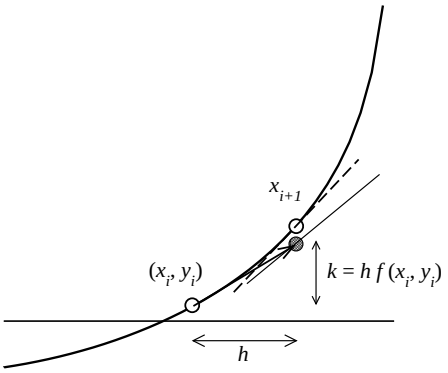

図 3: Euler 法の改良。

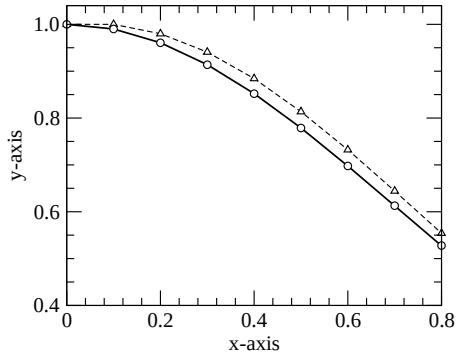

図 4: Euler 法と二次の Runge–Kutta 法の精度の比較。丸印の Runge–Kutta 法による解は三角の Euler 法の解に比べて、解析解 (太線) に近い。

4.3 Runge–Kutta 法

より良い解を得るために、Euler 法を改善する方法を考えよう。Euler 法で用いたのは、$x = x_i$ から $x = x_{i+1}$ に至る道筋の出発点での傾きである。出発点における傾きと終着点 $(x = x_{i+1})$ での傾きの平均値を使えばより真の解に近い値が得られないだろうか。図 3 にこのアイディアを示した。微分方程式を数値的に解くときには正しい終着点の値がどこにあるのかは分らないが、図中の影をつけた丸印で示した Euler 法による解の予想点における傾き (細線) は、正しい終着点での傾き (破線) の近似値になっているであろう。そこで $x = x_{i+1}$ における y として、$y_{i+1} = y_i + h\left(f(x_i, y_i) + f(x_i + h, y_i + k)\right)/2$ としてみるのである。ここで $k = hf(x_i, y_i)$ である。

この近似解法は次のようにして正当化することができる。微分方程式の解を Taylor 展開すると、

$$\begin{aligned} y(x_i + h) &= y(x_i) + h\frac{dy}{dx} + \frac{h^2}{2}\frac{d^2y}{dx^2} + \cdots \\ &= y_i + hf(x_0, y_0) + \frac{h^2}{2}\frac{d}{dx}f(x, y) + \cdots \\ &= y_i + hf(x_0, y_0) + \frac{h^2}{2}\left(\frac{\partial f}{\partial x} + \frac{\partial f}{\partial y}\frac{dy}{dx}\right) + \cdots \end{aligned} \tag{2}$$

ここで y を x で微分したものは、与えられた微分方程式を用いると $\frac{dy}{dx} = f(x, y)$ となることを利用した。f の微分はすぐに計算できるとは限らないので、式 (2) の中の f の偏微分を (x_i, y_i) 近傍の $f(x, y)$ の値の線形結合で置換えることを考える。すなわち、

$$y(x_i + h) = y(x_i) + h\left[a_1 f(x_0, y_0) + a_2 f(x_0 + \theta h, y_0 + \theta hf)\right] \tag{3}$$

が成り立つとしてみる。大括弧の中の f を再び Taylor 展開すると

$$f(x_0 + \theta h, y_0 + \theta hf) = f(x_0, y_0) + \theta h\left(\frac{\partial f}{\partial x} + \frac{\partial f}{\partial y}f(x_0, y_0)\right) + \cdots \tag{4}$$

なので、これを式 (3) に代入して式 (2) の h の冪の係数と比べれば、$a_1 + a_2 = 1$、$a_2\theta = 1/2$ が成立すると、この近似は h^2 のオーダーまで正しいことが判る。この条件は例えば $a_1 = a_2 = 1/2$、$\theta = 1$ とすれば満すことができる。これは先程の出発点と終着点の平均を利用する近似そのものである。この近似は二次の Runge–Kutta 法と呼ばれる。

図 4 に微分方程式 $dy/dx = -2xy$ を例に取って、Euler 法と Runge–Kutta 法の精度の違いを示した。丸印の Runge–Kutta 法の解は実線で示した解析解の上に良く乗っているが、三角印の Euler 法は「ずれ」が目立つ。

練習問題 6: 二次の Runge–Kutta 公式を用いて、常微分方程式

$$\frac{dy}{dx} = -xy$$

を初期条件 $(x_0, y_0) = (0, 15)$ のもとで、$x = 5$ まで求めなさい。計算間隔 $h = 0.5$ とした場合と $h = 0.1$ とした場合を計算し、解析解と比較しなさい。

ヒント

微分方程式の右辺を計算する外部関数を定義すると見通しが良くなる。外部関数に `fct` という名前を付けるとすると、関数を定義するプログラムは、

```
function fct(x, y)
implicit none
double precision :: x, y, fct
fct = - x * y
end
```

あとは上に述べた処方箋に従って逐次的に解を求めればよい。すなわち、

1. 出発点での傾きを計算し、終着点の予測をする。
2. 終着点での傾きを求める。
3. 上の二つの平均を取って、改良された変化率を求める。
4. 改良された変化率に刻み幅を掛けて、次の点を求める。

と言う手続きを `do` 文で繰り返せばよい。

```
      implicit none
      double precision :: x, y, h, k1, k2, k, fct
...
      do i = 1, 50
        k1 = h * fct(x, y)
        k2 = h * fct(x + h, y + k1)
        k = (k1 + k2) * 0.5d+0
        x = x + h
        y = y + k
      end do
...
```

こんな感じ。

4.4 四次の Runge–Kutta 法

二次の Runge–Kutta 法では、$f(x, y)$ の線形結合を使って解の Taylor 展開の h^2 の項まで等しくなるように係数を決めたが、この考えを更に進めるとより精度の高い近似手法が得られる。二次の Runge–Kutta 法の導出過程を見直してみると判るように、傾きを計算する点の選び方や線形結合の係数は一通りには決まらない。幾通りもある組合せの中でしばしば用いられるのが四次の Runge–Kutta 公式と呼ばれるもので、次の公式で y の増分を計算する。

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

$$
\begin{aligned}
k_2 &= hf(x_n + \frac{h}{2},\, y_n + \frac{k_1}{2}) \\
k_3 &= hf(x_n + \frac{h}{2},\, y_n + \frac{k_2}{2}) \\
k_4 &= hf(x_n + h,\, y_n + k_3) \quad 。
\end{aligned}
$$

上の公式は多数の可能な組合せの一つに過ぎないのだが、多くの場合上手く解を見付けることが経験的に知られており、普通 Runge–Kutta 法と言えばこの四次の公式を指す。

最後に一点付記しておこう。微分方程式 (1) の右辺が x のみの関数で y に依らないとき、微分方程式の解を求める問題は単純な積分に帰着する。その際、ここで説明した二次の Runge–Kutta 法は数値積分の台形公式に、四次の Runge-Kutta 公式は Simpson 公式に対応している。各自確かめられたい。

練習問題 7: 四次の Runge–Kutta 法を用いて、常微分方程式

$$\frac{dy}{dx} = \sin x + \cos y$$

を初期条件 $(x_0, y_0) = (0, 0)$ のもとで、$x = \pi/2$ まで、15 等分して求めなさい。($x = \pi/2$ で $y = 1.793367$ 位になる。)

ヒント

手続きは複雑になるが、結局前問と同じことである。まず外部関数を定義しておいて、関数の増分 k_1、k_2、k_3、k_4 を順に計算し、重みをつけて平均することで最適な増分を求める。

練習問題 8: 常微分方程式

$$\frac{dy}{dx} = x^2 + y$$

を初期条件 $(x_0, y_0) = (1, 1)$ のもとで、計算間隔 $h = 0.1$ として $x = 2$ まで求めなさい。

余談

数学の復習を兼ねて、埋め草に上の微分方程式を解いてみよう。右辺が $x^2 + y$ ではなく y だけだったらこの微分方程式は容易に解くことができる。

$$
\begin{aligned}
\frac{dy}{dx} &= y \\
\int \frac{dy}{y} &= \int dx \qquad \rightarrow \qquad y = Ce^x
\end{aligned}
$$

ここで積分定数の C が定数ではなく x に依存する関数だとしてみる (定数変化法)。
$y(x) = C(x)e^x$ を元の微分方程式に代入して、

$$\frac{dy}{dx} = \frac{dC}{dx}e^x + C\frac{de^x}{dx} = \frac{dC}{dx}e^x + Ce^x = \frac{dC}{dx}e^x + y = x^2 + y\,。$$

よって

$$\frac{dC}{dx} = x^2e^{-x}\,。$$

この微分方程式は部分積分で積分できて、

$$C = -(x^2 + 2x + 2)e^{-x} + c \quad (c \text{ は積分定数})。$$

初期条件 $x = 1$ で $y = 1$ を満すように積分定数を決めると、$y = 6e^{x-1} - (x^2 + 2x + 2)$。

4.5 応用: Kepler 問題

大学入学後に学んだ力学の中から Kepler(ケプラー)問題をトピックとして取り上げよう。 Newton(ニュートン)力学によって太陽のまわりを公転する惑星の軌道を求める問題である。Newton 方程式は

$$m\frac{d^2\vec{x}}{dt^2} = \vec{F}$$

と時間の二階微分を含む微分方程式なので、いままでの手法を適用できないように思ってしまうが、次のような工夫をすれば解くことができる。即ち、時間による位置座標の一階微分 (速度!) を $\frac{d}{dx}x = v$ と置いて、

$$\frac{d}{dt}\begin{pmatrix} x \\ v \end{pmatrix} = \begin{pmatrix} v \\ F/m \end{pmatrix}$$

と書けば、二階微分方程式は一階の連立微分方程式に帰着する。連立微分方程式のそれぞれの従属変数の短時間の間の増分を前節の Runge–Kutta 法等で求めれば、Newton 方程式も解くことができる。

Kepler 運動は平面内での運動なので、x-y 面のみを考える。速度の x 成分を u、y 成分を v とすれば、太陽のまわりを回る地球の運動方程式は、連立一階微分方程式の形では

$$\frac{d}{dt}\begin{pmatrix} x \\ y \\ u \\ v \end{pmatrix} = \begin{pmatrix} u \\ v \\ F_x/m_{\mathrm{e}} \\ F_y/m_{\mathrm{e}} \end{pmatrix}$$

となる。ここで F_x、F_y は地球に働く太陽の重力の x、y 成分、m_{e} は地球の質量である。重力は Newton 定数を G、太陽の質量を M_{S} とすれば、

$$F = \sqrt{F_x^2 + F_y^2} = G\frac{m_{\mathrm{e}}M_{\mathrm{S}}}{r^2}$$

但し $r = \sqrt{x^2+y^2}$ は太陽と地球の間の距離である。Runge–Kutta 法を使えば、時刻が δt だけ進んだ時の x と u の変化分は、

$$\begin{aligned}
x(t+\delta t) &= x(t) + \frac{k_1 + 2k_2 + 2k_3 + k_4}{6} \\
u(t+\delta t) &= u(t) + \frac{K_1 + 2K_2 + 2K_3 + K_4}{6} \\
k_1 &= u(t)\delta t \\
K_1 &= -G\frac{M_{\mathrm{S}}}{x(t)^2 + y(t)^2}\frac{x(t)}{\sqrt{x(t)^2+y(t)^2}}\delta t \\
&\cdots
\end{aligned}$$

等々として順次求めることができる。$k_2, K_2, \ldots$ の式は紙の上に書くと複雑だが、計算機上では、天体の座標を受取ると天体同士に働く力 (の座標成分) を返す関数を用意しておけば、同じような命令の繰り返しでプログラミングすることができる。

> 応用問題 3: 太陽、地球、月の三体系の運動を再現するプログラムを作れ。太陽を動かないものとして原点に置き、地球と月は同一の公転面上を運動すると近似する。太陽、地球、月の質量はそれぞれ $M_{\mathrm{sun}} = 2.0\times10^{30}\mathrm{kg}$、$m_{\mathrm{earth}} = 6.0\times10^{24}\mathrm{kg}$、$m_{\mathrm{moon}} = 7.3\times10^{22}\mathrm{kg}$、地球と月の公転軌道半径は $R_{\mathrm{earth}} = 1.5\times10^{11}\mathrm{m}$、$R_{\mathrm{moon}} = 3.8\times10^{8}\mathrm{m}$、公転速度は $v_{\mathrm{earth}} = 3.0\times10^{4}\mathrm{m/s}$、$v_{\mathrm{moon}} = 1.0\times10^{3}\mathrm{m/s}$ である。重力定数は $G = 6.674\times10^{-11}\mathrm{N{\cdot}m^2{\cdot}kg^{-2}}$ である。

ヒント

どこから手を付けて良いか分らない時は、ややこしそうなことを後回しにすると良い。

Runge–Kutta 法の 1 ステップを実行して、時刻が刻み幅の dt だけ進んだ時点での星々の配置を返す関数が完成できたとすると、プログラムは

```
      integer :: rngktt
      double precision :: x(2), y(2), u(2), v(2), t, dt
c
      ...
      do i = 1, 365
        if (rngktt(x, y, u, v, dt) /= 0) then
          exit
        end if
        t = t + dt
        print *, t, x(1), y(1), x(2), y(2)
      end do
```

こんな感じになるであろう。Runge–Kutta ステップが失敗したら関数 rngktt からは 0 でない値が返って来ることにした。また、(x(1), y(1)) は地球の座標、(x(2), y(2)) は月の座標である。肝腎な部分は関数にしてしまっているので、これなら簡単。

関数 rngktt はどう書けば良いだろう。Runge–Kutta 法では四つの小ステップで予測値を求めながら最後に平均を取るのであった。変数に x、y、u、v 等と別々の名前を付けていると手際の良いプログラムが書けないので、x(1)→r(1)、x(2)→r(2)、y(1)→r(3)、y(2)→r(4) ... などとして一つの配列に変数をまとめて入れてしまおう。配列のサイズは 8 となる。微分を入れるための配列や Runge–Kutta 法の増分を入れる配列 k1、k2... 等も用意すれば、必要になる関数や配列の宣言は

```
      function rngktt(x, y, u, v, dt)
      integer :: rngktt
      double precision :: x(2), y(2), u(2), v(2), dt
      double precision :: k1(8), k2(8), k3(8), k4(8)
      double precision :: r0(8), r(8), f(8)
```

こんな感じになるだろう。四つの小ステップは

```
c 1st step
      if (clcdrv(f, r0, m)          /= 0) then
        return
      end if
      if (rksstp(k1, f, dt)         /= 0) then ...
c 2nd step
      if (rksft (r, r0, k1, 0.5d+0) /= 0) then ...
      if (clcdrv(f, r, m)           /= 0) then ...
      if (rksstp(k2, f, dt)         /= 0) then ...
c 3rd step
      if (rksft (r, r0, k2, 0.5d+0) /= 0) then ...
      if (clcdrv(f, r, m)           /= 0) then ...
      if (rksstp(k3, f, dt)         /= 0) then ...
c 4th step
      if (rksft (r, r0, k3, 1.0d+0) /= 0) then ...
      if (clcdrv(f, r, m)           /= 0) then ...
      if (rksstp(k4, f, dt)         /= 0) then ...
c
      do i = 1, 8
        r(i) = r0(i) + (k1(i) + 2.0d+0 * k2(i)
     $             + 2.0d+0 * k3(i) + k4(i)) / 6.0d+0
      end do
```

などと表現することができる。ここで、clcdrv は微分を計算して配列 f に入れる関数、rksft は初期値に変化分を加えて配列 r に入れる関数、rksstp は各ステップでの変化分を

計算して配列 k に入れる関数である。どの関数も計算がうまくできなかったら 0 でない値を返すものとする。似た名前の関数が何度も呼出されているが、最初の変化分 k_1 を計算したら、二番目のステップでは $r = r_0 + k_1/2$ での傾きを計算する等々の Runge–Kutta 法の手続きが、忠実に実行されているのを確認して欲しい。

これらそれぞれの手続きは、比較的単純にコーディングできると思う。rksft は初期値に変化分を加えて配列 r に入れる関数なので

```
      function rksft(r, r0, k, scl)
...
      do i = 1, 8
        r(i) = r0(i) + k(i) * scl
      end do
```

こんな感じに。rksstp は各ステップでの変化分を計算して配列 k に入れる関数ということなので、

```
      function rksstp(k, f, dt)
...
      do i = 1, 8
        k(i) = f(i) * dt
      end do
```

これで良い。微分を計算する関数 clcdrv は少々面倒くさい。必要な手続きを幾段階かに分けて考えて行こう。まず位置座標の微分は比較的に楽で、対応する速度の成分を代入するだけで済む。

```
      function clcdrv(f, r, m)
...
c derivative of position is velocity
      do i = 1, 2
        f(i + 0) = r(i + 4)
        f(i + 2) = r(i + 6)
      end do
```

「do i = 1, 2」は地球と月についてのループである。地球の速度の x-成分 (u(1)) が x-座標の微分 (f(1)) に代入されていることを、配列 r の定義を思い出して確かめて欲しい。他の変数もきちんと対応しているであろうか。

速度の微分については、天体間の距離を求め、Newton の法則で星に働く力を求めて、質量で割って加速度にした上で、x-成分と y-成分に分解する必要がある。まず太陽と地球、太陽と月の間の重力からの寄与は

```
c gravity of SUN
      do i = 1, 2
        dx = r(i + 0)
        dy = r(i + 2)
        rd2 = dx * dx + dy * dy
        frc = G * M * m(i) / rd2
        f(i + 4) = - frc / m(i) * dx / sqrt(rd2)
        f(i + 6) = - frc / m(i) * dy / sqrt(rd2)
      end do
```

最後に月と地球の間の重力の寄与は

```
c gravity between stars
      n = 2
      do i = 1, n - 1
      do j = i + 1, n
        dx = r(j + 0 * n) - r(i + 0)
        dy = r(j + 1 * n) - r(i + 1 * n)
        rd2 = dx * dx + dy * dy
        frc = G * m(i) * m(j) / rd2
```

```
      f(i + 2 * n) =   frc / m(i) * dx / sqrt(rd2)
      f(i + 3 * n) =   frc / m(i) * dy / sqrt(rd2)
      f(j + 2 * n) = - frc / m(j) * dx / sqrt(rd2)
      f(j + 3 * n) = - frc / m(j) * dy / sqrt(rd2)
    end do
    end do
```

プログラムが完成すると、地球と月以外の衛星や宇宙船等の物体も計算に取り入れたくなりそうなので、天体の数を `n` として一般性を持たせたコーディングにしてみた。太陽以外に地球と月しかない系ならばループは不要である。

Runge–Kutta 法の各ステップを計算する部分で `rksft` や `rksstp` などの関数を呼ぶところは、メインプログラムで関数 `rngktt` を呼んだときのように関数の戻り値を `if` 文を用いて評価したり、何かの変数に代入するなどの処理が必要である。配列にデータをセットする関数のように、関数の「副作用」が重要で戻り値が必要ないときのために、Fortran 言語にはサブルーチンというものも用意されている。今回の例のようなプログラムにはサブルーチンを用いる方が一般的であろう。興味があれば教科書などを見て欲しい。しかしながら計算の流れは上に示したもので問題ない。また座標や速度を一つの配列にまとめて代入する手続きや、結果を配列から座標や速度に戻す手続きも書かねばプログラムは動かない。自分で考えてみて欲しい。
