ルンゲ・クッタ・ギル法

B0SM3003　黒島利沙

2010 年　10 月吉日

1　ルンゲ・クッタ・ギル法

前回のルンゲ・クッタ法では高山氏が素晴らしいトラペを書いてくれた。今回はそのルンゲ・クッタ法のアレンジ版としてルンゲ・クッタ・ギル法 (以下 RKG) を紹介する。これを用いると、常微分方程式を解く際にとても素敵な事があるかもしれない… !?

まずは、RKG 法の表記について。
4 次のルンゲ・クッタ法は一般的に以下のように記される。

$$
\begin{aligned}
k_1 &= h \cdot f(x_n + \alpha_1 h, y_n) && (1)\\
k_2 &= h \cdot f(x_n + \alpha_2 h, y_n + \beta_1 k_1) && (2)\\
k_3 &= h \cdot f(x_n + \alpha_3 h, \underline{y_n + \beta_2 k_1 + \gamma_1 k_2}) && (3)\\
k_4 &= h \cdot f(x_n + \alpha_4 h, \underline{y_n + \beta_3 k_1 + \gamma_2 k_2} + \delta_1 k_3) && (4)
\end{aligned}
$$

$$
y_{n+1} = \underline{y_n + c_1 k_1 + c_2 k_2} + c_3 k_3 + c_4 k_4 \qquad (5)
$$

それぞれの式に対する各係数は、以下の関係性を持ち、c_3 はパラメータとなっ

$$
\begin{array}{llll}
\alpha_1 = 0 & & & \\
\alpha_2 = \frac{1}{2}, & \beta_1 = \frac{1}{2} & & \\
\alpha_3 = \frac{1}{2}, & \beta_2 = \frac{3c_3-1}{6c_3}, & \gamma_1 = \frac{1}{6c_3} & \\
\alpha_4 = 1\ , & \beta_3 = 0, & \gamma_2 = 1 - 3c_3, & \delta_1 = 3c_3 \\
c_1 = \frac{1}{6}, & c_2 = \frac{2}{3} - c_3, & c_3, & c_4 = \frac{1}{6}
\end{array}
$$

ている。ここで $c_3 = \frac{1}{3}$ とすると、前回のゼミで登場した 4 次のルンゲ・クッタ法

に対応する。式 (3),(4),(5) の下線部は y_0,k_1,k_2 の一次結合である。これら 3 式に一次従属性があり、どれか 1 つの式が残りの 2 つの式で書き表せるとすると、計算の手間が 1 つ省けて好都合となる。(記憶領域の削減を目指して発案されたと考えらえる。) その時の C_3 の値として与えられるのが,

$$c_3 = \frac{2 \pm \sqrt{2}}{6}$$

であり、　$c_3 = \frac{2+\sqrt{2}}{6}$　と選ぶと…局所打切誤差の係数の絶対値が小さくなる。この時の各係数を求めた結果が RKG 法の式であり、以下のように表せる。(ゼミでは誤釈していました、すみません…)

$$k_1 = h \cdot f(x_n, y_n) \tag{6}$$

$$k_2 = h \cdot f\left(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right) \tag{7}$$

$$k_3 = h \cdot f\left(x_n + \frac{h}{2}, y_n + \left(\frac{\sqrt{2}-1}{2}\right)k_1 + \left(\frac{2-\sqrt{2}}{2}\right)k_2\right) \tag{8}$$

$$k_4 = h \cdot f\left(x_n + h, y_n + \frac{1}{6}k_1 - \frac{1}{\sqrt{2}}k_2 + \left(\frac{2+\sqrt{2}}{2}\right)k_3\right) \tag{9}$$

$$y_{n+1} = y_n + \frac{1}{6}k_1 + \frac{1}{3}\left(\frac{2-\sqrt{2}}{2}\right)k_2 + \frac{1}{3}\left(\frac{2+\sqrt{2}}{2}\right)k_3 + \frac{1}{6}k_4 \tag{10}$$

局所打切誤差とは…
i ステップでの数値解が真値と完全に等しいと仮定した時、i+1 ステップの数値解と i+1 ステップの真値の間に生じる局所的な誤差のこと。

$$\frac{y_{i+1} - y_i}{h} = \Phi(x_i, y_i)\,,\ \lim_{h \to 0} \frac{y(x_{i+1} - y(x_i))}{h} = f(x_i, y(x_i))$$

$\Phi(x_i, y_i)$ は数値計算により求めた傾き、$f(x_i, y(x_i))$ は真の傾きとする。h は有限なので両者は完全に一致しないことは、直ちに理解できる。
i ステップでの数値解と真値が等しい $(y_i = y(x_i))$ と仮定する。i+1 ステップ目の数値解と真値解の誤差は、

$$r_{i+1} = y(x_{i+1}) - \{y_i + h \cdot \Phi(x_i, y_i)\}$$

となる。
RKG 法では r_{i+1} の係数の絶対値が小さくなるようにパラメータ c_3 を決定した。

2　丸め誤差

RKG 法の利点の一つは、記述方法を少し工夫するだけで丸め誤差が考慮できる、と言うものである。まずは丸め誤差について説明する、、前に浮動小数点数を簡単に説明する。計算機中では、整数型以外 (つまり実数型) は、

$$\text{実数値} = \pm(0.f_1 f_2 \cdots f_s)_\beta \times \quad \beta^e$$

という形で表させる。緑の部分を仮数部、β は進数表示を意味していて基数、e を指数部と呼び、この形を浮動小数点数と言う。
尚、仮数部の部分は、

$$(0.f_1 f_2 \cdots f_s)_\beta = f_1 \times \beta^{-1} + f_2 \times \beta^{-2} + \cdots \quad f_s \times \beta^{-s}$$

である。直感的に分かりやすくするため、10 進数で 8 ビットの記憶領域を仮定し[1]、符号で 1 ビット、指数部で 3 ビット、仮数部で 4 ビットを使うとする。図示すると以下のようになる。

±	e	$\cdots$	$\cdots$	f_1	f_2	f_3	f_4

この図から何が言いたいのかというと、計算機が記憶する実数の範囲は制限される。例えば $\sqrt{2}$ の無理数をこの記憶領域に収めようとした場合、仮数部に限りがあるため、$0.141421356\cdots \times 10^1$ は 0.1414×10^1 と記憶され、小数下位の部分は切り捨てられる。またはより近い数に表現するため、四捨五入や切り上げをされて実数は丸められる。丸めの際に発生する誤差を丸め誤差と呼ぶ。
ここで、丸め誤差を補償する方法を考えてみる。そのために、s に h を反復加算していく場合のアルゴリズム

$$s, s+h, (s+h)+h, \cdots$$

と、h の丸め誤差の補償を行った加算 $s = s + r$ を考える。h は s に比べ十分小さい桁と仮定する。

$$\begin{aligned}
r &= h - q \quad &&: q_0 = 0 \text{ とする。} \\
w &= s \quad &&: \text{加算する前の } s \text{ の値を取っておく。} \\
s &= s + r \quad &&: \text{加算の結果, } h \text{ の下位の桁が桁合わせのため切られる。} \\
r &= s - w \quad &&: (\text{加算した } s) - (\text{加算する前の } s) = h \text{ の上位の桁が求まる。} \\
q &= q + r - h \quad &&: \text{切り捨てられた } h \text{ の下位の桁が逆符号で求まる。}
\end{aligned}$$

[1] 本来は計算機は 2 進数表示が一般的である。

…よく分かりにくいと思うので、実際に数字を代入して流れを追い直そう。記憶領域は先ほどの 8 ビットとする。
$s = 0.5000 \times 10^3$, $h = 0.1234 \times 10^0$ とすると、

$$
\begin{aligned}
r &= h - q = 0.1234 \times 10^0 \quad : q_0 = 0 \text{ とする。} \\
w &= s = 0.5000 \times 10^3 \quad : \text{加算する前の } s \text{ の値を取っておく。} \\
s &= s + r = 0.5001 \times 10^3 \quad : h \text{ の下位の桁 } (0.0234 \times 10^0) \text{ が桁合わせで切られる。} \\
r &= s - w \quad : (\text{加算した } s) - (\text{加算する前の } s) = h \text{ の上位の桁 } (0.1000 \times 10^0) \text{ が求まる。} \\
q &= q + r - h \quad : \text{切り捨てられた } h \text{ の下位の桁 } (-0.0234 \times 10^0) \text{ が逆符号で求まる。}
\end{aligned}
$$

上式を見ると、第 2 ループ目に入ったとき、切り捨てられた h の下位の桁が h-q で補償される。$s + h$ の加算では、丸め誤差により真値からのずれが大きくなっていくが、$s + r$ の加算では、丸め誤差の修正がステップ毎にかかるため、丸め誤差を補償できる。数ステップ繰り返すと、

$$
\begin{aligned}
s + h &= 0.5001 \times 10^3,\ s + r = 0.5001 \times 10^3 \\
s + 2h &= 0.5002 \times 10^3,\ s + r = 0.5002 \times 10^3 \\
s + 3h &= 0.5003 \times 10^3,\ s + r = 0.5003 \times 10^3 \\
s + 4h &= 0.5004 \times 10^3,\ s + r = 0.5004 \times 10^3 \\
s + 5h &= 0.5005 \times 10^3, s + r = 0.5006 \times 10^3
\end{aligned}
$$

と、5 ループ目にわずかに違いが見られる。

3　アルゴリズム

では丸め誤差のアルゴリズムを考慮したRKG法の表記はどんなものか簡単に書き下す。

$$
\begin{aligned}
k &= dx \cdot f(x, y) \\
r &= 0.5k - q
\end{aligned}
$$

$$
\begin{aligned}
Y_1 &= y + r \\
r &= Y_1 - y \\
q &= q + 3r - 0.5k \\
k &= dx \cdot f(x + 0.5dx, Y_1) \\
r &= \left(1 - \frac{1}{\sqrt{2}}\right)(k - q)
\end{aligned}
$$

$$
\begin{aligned}
Y_2 &= Y_1 + r \\
r &= Y_2 - Y_1 \\
q &= q + 3r - \left(1 - \frac{1}{\sqrt{2}}\right)k \\
k &= dx \cdot f(x + 0.5dx, Y_2) \\
r &= \left(1 + \frac{1}{\sqrt{2}}\right)(k - q)
\end{aligned}
$$

$$
\begin{aligned}
Y_3 &= Y_2 + r \\
r &= Y_3 - Y_2 \\
q &= q + 3r - \left(1 + \frac{1}{\sqrt{2}}\right)k \\
k &= dx \cdot f(x + dx, Y_3) \\
r &= \frac{1}{6}(k - 2q)
\end{aligned}
$$

$$
\begin{aligned}
y &= Y_3 + r \\
r &= y - Y_3 \\
q &= q + 3r - 0.5k
\end{aligned}
$$

この一巡をdoループで回していく。赤色の式部分では、iステップでのyから次のステップであるi+1でのyの値を求めている。また、rは丸め誤差の補償値となっていて、随時yに足し上げていく事で、切り捨てられた下位の桁を補っている。

[考察]

昔は記憶装置は数百語程度 (非常に少ない!)、当然記憶装置の必要数には神経を使った。そんな中、gill さんが記憶装置を減らす記述方法 (上記) を提唱した、これが RKG 法である。連立 m 元方程式を解くとき通常の 4 次ルンゲクッタ法の必要記憶装置数が 4m に対し、RKG 法ではこのセクションで紹介したアルゴリズムで記述するだけで 3m となる。ついでにこの記述法で丸め誤差も修正してくれるなんてまぁお得♪

本当に 3 つになるの？と言う疑問。必要最低限の記憶装置を使おうと思うと、まず r は簡単に記述するための文字として置いたので、置き換えることをしなければ r という記憶装置は不必要になる。そして、Y_i(i=1,2,3) だが、必要なのは Y_i と Y_{i+1} の 2 つだけなので Y_3 の記憶装置を Y_1 で代用してしまえば、Y_3 は必要なくなる。
結局必要な文字は…q と Y_1、Y_2 の 3 つだけになります !! (k は 4 次ルンゲクッタ法でも用いるのでノーカウントです。) すごいこっちゃ！

しかし現在の記憶装置の発達や精度の進歩を思えば、4 次ルンゲクッタ法も RKG 法もさほど変わらない、という結論に至りました。詳しくは宿題試してみてください。

4　宿題

$$\frac{dy}{dx} = x^7 \tag{11}$$

という微分方程式を、

$$x_0 = 1\ , y_0 = 0.125 \tag{12}$$

という初期値を用いて以下の５つの方法で $x = 1$ から $x = 2$ まで積分し、その結果を較べなさい。

1. オイラー法
2. ホイン法
3. 2 次ルンゲクッタ法
4. 4 次ルンゲクッタ法
5. ルンゲクッタギル法

また、ステップ幅はそれぞれの方法に対し、$h = 0.1, 0.01, 0.001$ の三通りを試行し、ループ数は $n = \frac{1}{h}$ とする。

★追加問題★　ステップ数 h は細かくすればする程求める解の精度は向上する。が、ある値以上小さくしても精度が上がらない、打ち切りのようなものが存在する。そこで！各人で h の打ち切りのオーダー (精度が飽和してしまう h) がどれくらいなのかを、４次のルンゲクッタ法にて簡単に見積もってみてください。
ex. 高〇氏の計算機では、$h = 0.05$ くらいだそうです。

完