Vol.2018-HPC-164 No.1
2018/5/7

ピタゴラス三体問題の4倍精度高次Taylor展開法による数値計算

平山 弘[1,a)]

概要：
ピタゴラス三体問題 (Baurrau's problem) は、1913 年にブラウ (C.Burrau) によって研究され、1967 年にエール大学のサブヘイ [10] らによって、Levi-Civita 変換を利用して数値計算によって解決された。
本論文では、高精度（４倍精度）で、高次（24 次）の Taylor 展開法を使えば、特別な変換を使わないで、高精度な計算結果を得ることができることを示す。

キーワード：4 倍精度, 常微分方程式の高次解法，Taylor 展開法, ピタゴラスの三体問題

Numerical calculation by quadruple precision higher order Taylor series method of The Pythagorean problem of three bodies

Hiroshi Hirayama[1,a)]

Abstract:
The Pythagorean problem of three bodies(Baurrau's problem) is studied by Blau (C.Burrau) in 1913. By Szebehely[10], Yale University in 1967, using Levi-Civita transformation, it was solved by numerical computation.
In this paper, it is shown that it's possible to get a highly precise calculation result with higer order Taylor series method of high precision (the quadruple precision) without a special change.

Keywords: quadruple precision, higher order Taylor series method, ordinary differential equation, Pythagorean problem of three bodies

1. はじめに

常微分方程式の数値解法には Euler 法や Runge-Kutta 法などの陽的計算法 [3] を利用することが一般的である。これらの方法で、次のような初期値問題の微分方程式を考える。

$$\mathbf{y}' = \mathbf{f}(x, \mathbf{y}) \qquad \mathbf{y}(x_0) = \mathbf{y}_0 \tag{1}$$

s 段の陽的 Runge-Kutta 法は次のように書ける。

$$\left\{\begin{array}{lcl}
\mathbf{k}_1 &=& \mathbf{f}(x_n, \mathbf{y}_n) \\
\mathbf{k}_2 &=& \mathbf{f}(x_n + c_2 h, \mathbf{y}_n + a_{21} h \mathbf{k}_1) \\
\mathbf{k}_3 &=& \mathbf{f}(x_n + c_3 h, \mathbf{y}_n + a_{31} h \mathbf{k}_1 + a_{32} h \mathbf{k}_2) \\
 & \vdots & \\
\mathbf{k}_s &=& \mathbf{f}(x_n + c_s h, \mathbf{y}_n + a_{s1} h \mathbf{k}_1 + a_{s2} h \mathbf{k}_2 \\
 & + & \cdots + a_{s,s-1} h \mathbf{k}_{s-1}) \\
\mathbf{y}_{n+1} &=& \mathbf{y}_n + \sum_{i=1}^{s} b_i \mathbf{k}_i
\end{array}\right. \tag{2}$$

これらの方法では、高次の公式は段数の 2 乗に比例する定数を含んでいるため使うのが難しいという問題がある。また、これらの定数を求めるのが難しいという問題がある。

このため、よく使われる公式が 4 段 4 次の古典的公式と

1 神奈川工科大学創造工学部自動車システム開発工学科
Department of Vehicle System Engineering, Faculty of Creative Engneering, Kanagawa Institute of Technology, Shimo-Ogino 1030, Atsugi, Kanagawa, 243-0292, Japan
a) hirayama@kanagawa-it.ac.jp

呼ばれる次の公式である。

$$\left\{\begin{array}{lcl}\mathbf{k}_1 & = & \mathbf{f}(x_n, \mathbf{y}_n) \\ \mathbf{k}_2 & = & \mathbf{f}(x_n + \frac{h}{2}, \mathbf{y}_n + \frac{h}{2}\mathbf{k}_1) \\ \mathbf{k}_3 & = & \mathbf{f}(x_n + \frac{h}{2}, \mathbf{y}_n + a_{31}h\mathbf{k}_1 + \frac{h}{2}\mathbf{k}_2) \\ \mathbf{k}_4 & = & \mathbf{f}(x_n + h, \mathbf{y}_n + h\mathbf{k}_3) \\ \mathbf{y}_{n+1} & = & \mathbf{y}_n + \frac{h}{6}(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4)\end{array}\right. \tag{3}$$

陽的5段5次の公式は存在しないので、次によく使われると思われる公式は、6段5次の Runge-Kutta-Fehlberg の公式と呼ばれる次の公式である。

$$\begin{aligned}\mathbf{y}_{n+1} &= \mathbf{y}_n + h_n\left(\frac{16}{135}\mathbf{k}_1 + \frac{6656}{12825}\mathbf{k}_3 + \frac{28561}{56430}\mathbf{k}_4\right. \\ &\quad \left.-\ \frac{9}{50}\mathbf{k}_5 + \frac{2}{55}\mathbf{k}_6\right) \\ \overline{\mathbf{y}}_{n+1} &= \mathbf{y}_n + h_n\left(\frac{25}{216}\mathbf{k}_1 + \frac{1408}{2565}\mathbf{k}_3 + \frac{2197}{4104}\mathbf{k}_4 - \frac{1}{5}\mathbf{k}_5\right) \\ \mathbf{k}_1 &= \mathbf{f}(x_n, \mathbf{y}_n) \\ \mathbf{k}_2 &= \mathbf{f}\left(x_n + \frac{1}{4}h_n, \mathbf{y}_n + \frac{1}{4}h_n\mathbf{k}_1\right) \\ \mathbf{k}_3 &= \mathbf{f}\left(x_n + \frac{3}{8}h_n, \mathbf{y}_n + \frac{1}{32}h_n(3\mathbf{k}_1 + 9\mathbf{k}_2)\right) \\ \mathbf{k}_4 &= \mathbf{f}\left(x_n + \frac{12}{13}h_n,\right. \\ &\quad \left.\mathbf{y}_n + \frac{1}{2197}h_n(1932\mathbf{k}_1 - 7200\mathbf{k}_2 + 7296\mathbf{k}_3)\right) \\ \mathbf{k}_5 &= \mathbf{f}\left(x_n + h_n,\ \mathbf{y}_n\right. \\ &\quad \left.+\ h_n\left(\frac{439}{216}\mathbf{k}_1 - 8\mathbf{k}_2 + \frac{3680}{513}\mathbf{k}_3 - \frac{845}{4104}\mathbf{k}_4\right)\right) \\ \mathbf{k}_6 &= \mathbf{f}\left(x_n + \frac{1}{2}h_n, \mathbf{y}_n\right. \\ &\quad \left.+\ h_n\left(-\frac{8}{27}\mathbf{k}_1 + 2\mathbf{k}_2 - \frac{3544}{2565}\mathbf{k}_3 + \frac{1859}{4104}\mathbf{k}_4 - \frac{11}{40}\mathbf{k}_5\right)\right)\end{aligned}$$

$\mathbf{y}$ は5次の公式で,$\overline{\mathbf{y}}$ は4次の公式である。その差 $e = |\mathbf{y} - \overline{\mathbf{y}}|$ は、誤差と推定され, h^5 に比例する。比例定数は、場所 x_n および関数値 $\mathbf{y}_n$ の関数であるが、局所的には定数で近似できる。その定数を a とすると、次の式になる。

$$error = ah^5$$

この式と許容誤差から、適切な計算刻み幅 h を推定できる。このように、関数を追加計算しないで、低次の公式を計算出来る公式を埋め込み型公式と呼ぶ。文献 [2] には、このような10次程度以下の公式が多数紹介されている。高次の公式（25段12次の公式）の作成には非常に多くの計算が必要 [8] であることが知られている。

常微分方程式で記述された大規模なシステムを長時間に渡り高精度で計算することが求められている。これらの方法では計算次数が限定され長時間高精度の計算は非常に難しくなる。このような問題に対し、任意次数、可変ステップの計算方法である Taylor 展開法が最も適していると思われる。

本論文では、ひとつの例として、天文学上の三体問題を Taylor 展開法を使って解き、計算精度を調べ、Taylor 展開法の性能を調べた。計算する常微分方程式として、解くのが難しいとされるピタゴラスの三体問題 [5] を選択した。ピタゴラスの三体問題とは、辺長3,4,5の直角三角形の頂点の位置に、それぞれの対辺長に比例する質量3,4,5の質点を図1のように静止状態で配置し、その状態を初期条件として、これらの質点が相互の引力によって、この後どう運動するかを追及する問題である。

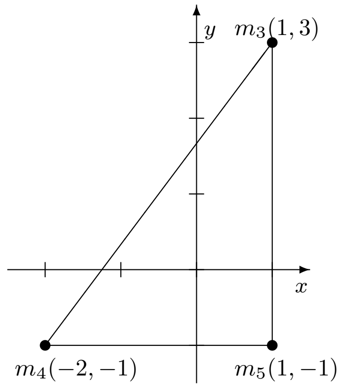

図 1 ピタゴラスの三体問題の初期状態

質点 m_3 、m_4 間の距離を r_{34} というように定義すると

$$\begin{aligned}r_{34} &= \sqrt{(x_3 - x_4)^2 + (y_3 - y_4)^2} \\ r_{35} &= \sqrt{(x_3 - x_5)^2 + (y_3 - y_5)^2} \\ r_{45} &= \sqrt{(x_4 - x_5)^2 + (y_4 - y_5)^2}\end{aligned}$$

これらを用いて運動方程式を書くと、次のようになる。

$$\begin{aligned}\frac{d^2x_3}{dt^2} &= \frac{4(x_4 - x_3)}{r_{34}^3} + \frac{5(x_5 - x_3)}{r_{35}^3} \\ \frac{d^2y_3}{dt^2} &= \frac{4(y_4 - y_3)}{r_{34}^3} + \frac{5(x_5 - x_3)}{r_{35}^3} \\ \frac{d^2x_4}{dt^2} &= \frac{3(x_3 - x_4)}{r_{34}^3} + \frac{5(x_5 - x_4)}{r_{45}^3} \\ \frac{d^2y_4}{dt^2} &= \frac{3(y_3 - y_4)}{r_{34}^3} + \frac{5(x_5 - x_4)}{r_{45}^3} \\ \frac{d^2x_5}{dt^2} &= \frac{3(x_3 - x_5)}{r_{35}^3} + \frac{4(x_4 - x_5)}{r_{45}^3} \\ \frac{d^2y_5}{dt^2} &= \frac{3(y_3 - y_5)}{r_{35}^3} + \frac{4(x_4 - x_5)}{r_{45}^3}\end{aligned} \tag{4}$$

初期条件は

$$\begin{array}{llll} x_3 = 1, & \frac{dx_3}{dt} = 0, & y_3 = 3, & \frac{dy_3}{dt} = 0 \\ x_4 = 1, & \frac{dx_4}{dt} = 0, & y_4 = -1, & \frac{dy_3}{dt} = 0 \\ x_5 = 1, & \frac{dx_5}{dt} = 0, & y_5 = -1, & \frac{dy_3}{dt} = 0 \end{array} \tag{5}$$

以下の計算には、コンパイラとして、Visual Studio 2015 C++、計算機としてショップブランド・コンピュータ (Intel Core i7 7700K 4.2GHz) を利用した。

2. 常微分方程式の解の Taylor 展開

ここでは簡単に、常微分方程式の Taylor 展開法について簡単に説明する。高階の常微分方程式は一般性を失うことなしに 1 階微分方程式に書けるので、次の形を持つものと仮定する。

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

初期条件は、次のように与えられているものとする。

$$\mathbf{y}(x_0) = \mathbf{y}_0$$

ここで、$\mathbf{f}$ 、$\mathbf{y}$ は、一般にベクトル関数で、十分なめらかで必要な回数だけ微分可能とする。初期条件の $\mathbf{y}_0$ は定数ベクトルである。このような微分方程式は、次の Picard の逐次近似法 [9] によって解くことができる。

$$\begin{aligned} \mathbf{y}_0(x) &= \mathbf{y}_0 \\ \mathbf{y}_{k+1}(x) &= \mathbf{y}_0 + \int_{x_0}^{x} \mathbf{f}(t, \mathbf{y}_k(t))dt \end{aligned}$$

Taylor 展開式を上の式の被積分関数に代入し、被積分関数を Taylor 展開する。Taylor 級数は、k 回目の反復計算の場合、k 次まで Taylor 展開 [4] する。その k 次の Taylor 展開を積分し、定数項 $\mathbf{y}_0$ を加えて $k+1$ 次の解を計算する。1 回の計算で、最低 1 次次数の高い解が得られる。

例として次の簡単な常微分方程式を解く。

$$\frac{dy}{dx} = 1 + \sqrt{y} \qquad y(0) = 1$$

初期条件から $y_0(x) = 1$ であるから、これを picard の逐次近似式に代入して

$$y_1(x) = 1 + \int_0^x (1 + \sqrt{1})dt = 1 + 2x$$

となる。被積分関数は 0 次の定数となり、最終計算結果は 1 次式になる。さらにこの結果を、Picard の逐次近似式に代入して計算する。被積分関数は展開し 1 次式まで取る。$1 + 2x$ となる。これを積分して、計算すると

$$y_2(x) = 1 + \int_0^x (1 + \sqrt{1+2t})dt = 1 + 2x + 0.5x^2 + O(x^3)$$

となる。このような計算を 2 回繰り返すと、次のように 4 次までの解が得られる。

$$\begin{aligned} y_3(x) &= 1 + \int_0^x (1 + \sqrt{1 + 2t + 0.5t^2})dt \\ &= 1 + 2x + 0.5x^2 - 0.08333333x^3 + O(x^4) \\ y_4(x) &= 1 + \int_0^x (1 + \sqrt{1 + 2t + 0.5t^2 - 0.0833333t^3})dt \\ &= 1 + 2x + 0.5x^2 \\ &- 0.08333333x^3 + 0.0520833x^4 + O(x^5) \end{aligned}$$

この計算を必要な回数行えば、任意の次数の Taylor 展開式が得られる。

この Taylor 展開式を利用して、次のステップにおける関数値を計算する。次のステップの幅を h とすると、$y(h)$ を計算する。この値を、次のステップの初期値として、これまでの方法と同様にしてさらに次のステップの関数値を求める。これを繰り返すことで微分方程式を解く。

ここでは、Picard の逐次近似法を使って計算したが、Taylor 展開式の係数を計算する方法としては、Picard の逐次近似法は、級数展開法と同じ計算になる。

2.1 Taylor 級数の平方根の計算

前の例題で Taylor 展開の平方根の Taylor 展開式を計算している。この計算は、次の微分方程式を級数展開法で解くことによって計算できる。ここでは一般に α 乗することを考える。$\alpha = \frac{1}{2}$ の場合、平方根になる。$f(x)$ と $g(x)$ をそれぞれ次のような Taylor 展開とする。

$$\begin{aligned} f(x) &= f_0 + f_1 x + \cdots + f_n x^n + \cdots \\ g(x) &= g_0 + g_1 x + \cdots + g_n x^n + \cdots \end{aligned} \qquad (6)$$

$f(x)$ の α 乗を $g(x)$ とすると次の関係式が得られる。

$$g(x) = f(x)^\alpha$$

微分することによって、次の式が得られる。

$$g'(x) = \alpha f(x)^{\alpha-1} f'(x)$$

上の式の両辺に $f(x)$ を掛け、$g(x) = f(x)^\alpha$ であることを使うと、次の式が得られる。

$$f(x)g'(x) = \alpha g(x) f'(x) \qquad (7)$$

(6) の Taylor 展開式を (7) に代入すると、次のようになる。

$$\begin{aligned} (f_0 &+ f_1 x + f_2 x^2 + \cdots)(g_1 + 2g_2 x + 3g_3 x^2 + \cdots) \\ &= \alpha(g_0 + g_1 x + g_2 x^2 + \cdots)(f_1 + 2f_2 x + 3f_3 x^2 + \cdots) \end{aligned}$$

x の $n-1$ 次の項を比較すると、次の式が得られる。

$$\sum_{i=0}^{n} \{(n-i) - \alpha i\} f_i g_{n-i} = 0$$

上の式で、$i = 0$ の場合とそれ以外に分けると

$$n f_0 g_n + \sum_{i=1}^{n} \{(n-i) - \alpha i\} f_i g_{n-i} = 0$$

これから

$$g_n = \frac{1}{n f_0} \sum_{i=1}^{n} \{(\alpha+1)i - n\} f_i g_{n-i}$$

$g_0 = f_0^\alpha$ として、上の漸化式から係数を求める。Taylor 展開式の加減乗は簡単に計算でき、上と同様な方法で Taylor 展開式のべき乗、指数対数、三角関数などの計算が可能である。これらの公式を C++言語 [1] を利用すれば、Taylor 展開式を数値のように扱え、非常に簡単に記述できる。

3. 4倍精度計算

本計算に使った Microsoft 社製の C++言語は、4倍精度の数値は扱えない。ここでは、Bailey の double-double アルゴリズム [6] で4倍精度の計算を行う。double-double アルゴリズムでは，4倍精度浮動小数点数 (real16) を二つの倍精度浮動小数点数を使い上位桁を m0、下位を m1 で表し、次のような構造体で表す。

```
class real16 {  double  m0, m1 ; }
```

4精度変数 a を二つの倍精度変数 a.m0(上位データ) および a.m1(下位データ) を用いて次のように表す。

$$\mathrm{a} = \mathrm{a.m0} + \mathrm{a.m0} \quad (\frac{1}{2}ulp(\mathrm{a.m0}) \geq |\mathrm{a.m1}|) \qquad (8)$$

ここで、$ulp(x)$ は x の最小ビット (unit in the last place) を意味する。このとき，a.m0 および a.m1 は通常の倍精度浮動小数点数である。このため仮数部の精度は 53bit であり、2つの倍精度浮動小数点数を利用することで 106bit の精度で表現できる．そのため，double-double アルゴリズムは IEEE754-2008 の4倍精度と比較すると 8bit 分だけ精度が劣る．しかし，IEEE754-2008 の4倍精度はソフトウエアで作成されているため、計算速度はハードウエアの計算をする部分が多い double-double 型4倍精度数が速く計算が出来るので、実用的な方法であると言われている。[7]。

4倍精度加算および乗算を double-double アルゴリズムを利用して計算する方法を説明する。まず、double 型の数値2個 (a,b) の加算は、$|a| > |b|$ ならば、プログラム1の方法で高速に計算できる。厳密に $a + b = s + e$ が成り立ち $\frac{1}{2}ulp(\mathrm{s}) \geq |\mathrm{e}|$ となる。

プログラム１：高速加算

```
void fast_two_sum( const double a,
       const double b, double &s, double &e )
{
    s = a + b ;
    e = b-( s - a ) ;
}
```

この加算プログラムには、$|a| > |b|$ の条件が付くが、次のように書くと計算量は増えるがこの条件なしで加算できる。

プログラム２：加算

```
void two_sum( const double a,
       const double b, double &s, double &e )
{
    double v ;
    s = a + b ;
    v = s - a ;
    e = ( a - ( s - v ) ) + ( b - v ) ;
}
```

double 型の数値2個 (a,b) の乗算は、まず倍精度数を二つに分割することから始める。定数 $\mathrm{con} = 2^{\mathrm{n}} + 1$ とする。この定数を使って、

プログラム３：分割

```
void split( const double a, double &ah, double &al )
{
    double t, v, con ;
    t = con * a ;
    v = t - a ;
    ah = t - v ;
    al = a - ah ;
}
```

al には、a の下位 n ビットが入り、ah に残りの上位ビットが入る。n=26 とすると con=134217729.0 となる。この場合、ah と al はほぼ同じビット数の数値に分割される。それを使って以下のように、二つの倍精度数の乗算を行うことができる。

プログラム４：乗算

```
void two_prod( const double a,
       const double b, double &p, double &e )
{
    double ah, al, bh, bl ;
    p = a * b ;
    split( a, ah, al ) ;
    split( b, bh, bl ) ;
    d = (( ah * bh - p ) + ah * bl + al * bh )
      + al * bl ;
}
```

この計算によって、$\mathrm{a}*\mathrm{b} = \mathrm{p}+\mathrm{e}$ が成り立ち $\frac{1}{2}ulp(\mathrm{p}) \geq |\mathrm{e}|$ となる。もし、C99 言語で定義されている fma(fused multiply add) 関数が使用できれば、この関数は中身は次の2行で書ける。fma(a,b,c)=a*b+c と定義される関数である。この関数では計算は 128 ビットで行い最終的に 64 ビットに丸めた数値を返す関数である。したがって fma(a,b,-a*b) を計算することによって、a*b の下位 64 ビットが得られる。

プログラム４－１：乗算

```
void two_prod( const double a,
       const double b, double &p, double &e )
{
    p = a * b ;
    e = fma( a, b, -p ) ;
}
```

この fma 命令は、最近の Intel 社の CPU では、ハードウエア命令になっているので、それを使えば、高速に乗算の計算できると期待できる。

これらのプログラムを利用して、double-double 型4倍精度数の加算と乗算のプログラムを作成出来る。プログラム5、プログラム6に示す．これから加算、乗算の演算数はそれぞれ 11flops，24flops であることがわかる．この

プログラムは、double-double 型 4 倍精度数 (quad) である a,b の積を計算する C++言語で記述したものである。

プログラム 5：4 倍精度加算

```
quad add( const quad &a, const quad &b )
{
  real16 c ;
  double s1, s2 ;
  two_sum( a.m0, b.m0, s1, s2 ) ;
  s2 = s2 + a.m1 + b.m1 ;
  fast_two_sum( s1, s2, c.m0, c.m1 ) ;
  return c ;
}
```

プログラム 6：4 倍精度乗算

```
quad mul( const quad & a, const quad &b )
{
  real16 c ;
  double z1, z2 ;
  two_prod( a.m0, b.m0, z1, z2 ) ;
  z2 = z2 + a.m0 * b.m1 + a.m1 * b.m0 ;
  fast_two_sum( z1, z2, c.m0, c.m1 ) ;
  return c ;
}
```

計算例として、次の 2 次方程式を解くプログラムを 4 倍精度のプログラムを紹介する。プログラム 7 で示したプログラムは、2 次方程式のプログラムである。

プログラム 7：4 倍精度 2 次方程式の解法

```
#include "r16.h"
int main()
{
  real16 a, b, c, d, x1, x2 ;
  a=2 ; b = 7.5 ; c=real16("-12.2") ;
  d=b*b-4*a*c ; d = sqrt(d) ;
  x1=(-b+d)/(2*a) ; x2=(-b-d)/(2*a) ;
  set_format("%35.32g") ;
  cout << "x1=" << x1 << endl ;
  cout << "x2=" << x2 << endl ;
}
```

計算結果は次のようになる。

```
x1=  1.2259071253425182195488491564024
x2= -4.9759071253425182195488491564024
```

4. 三体問題の数値計算

方程式 (5) を初期条件 (6) を n 次の Taylor 級数を利用して解いた。刻み幅 h は、最高次数項の係数の絶対値が要求精度 ϵ より小さくなるように決定した。最高次数項の係数がゼロの場合は、その一つ次数の低い項の係数を使う。その項の係数がゼロならば、同様にさらに繰り返す。a_n が n 次の係数とすると、$|a_n|h^n <= \epsilon$ でなければならない。すなわち、各 Taylor 級数に対して、h を次のように計算する。

$$h = \sqrt[n]{\frac{\epsilon}{|a_n|}}$$

その中で最小の値を h とする。

表 1　3 天体の座標

t	x_3 / x_4 / x_5	y_3 / y_4 / y_5
0.0	1.0000000000000	3.0000000000000
	-2.0000000000000	-1.0000000000000
	1.0000000000000	-1.0000000000000
10.0	0.7784804101381	0.1413923002901
	-2.0250924779782	0.0972193841461
	1.1529857362997	-0.1626108874909
20.0	3.0042926366964	0.5119252350247
	-1.3886265375109	-0.4704760502527
	-0.6916743520091	0.0692256991874
30.0	0.8563404988973	2.2870936636963
	-0.8779838790318	-0.8659638277157
	0.1885828038871	-0.6794851360452
40.0	-0.6220036918011	1.8583181578998
	0.1735445568164	-2.3684104432832
	0.2343665696275	0.7797374598867
50.0	-2.7014614092655	-3.7972226827224
	1.5059381924207	0.9608134249384
	0.4161262916228	1.5096828696828
60.0	0.7438075001181	1.9399479510949
	0.2640103346863	-0.7316243948700
	-0.6574927678199	-0.5786692547609
70.0	6.9334635990119	20.2618043059517
	-2.0030060026060	-6.8724634358245
	-2.5576733573223	-6.6591118349114
80.0	12.4474428920319	36.6423087251151
	-3.5558667150022	-12.3547812055635
	-4.6237723632174	-12.1015602706182

今回の計算では、24 次の Taylor 展開式 ($n = 24$) を利用し、要求精度 $\epsilon = 10^{-28}$ として計算した。その結果の小数点以下 13 桁を表 1 に示す。この計算結果は長沢、桧山**?** の Levi-Civita 変換を利用して計算した結果より高精度の結果であるが精度の範囲で完全に一致する。

この計算結果を確かめるために、最終時間 t の時点で速度を逆転 ($\mathbf{v} = -\mathbf{v}$) させて、同じ経路を逆に計算し、初期値にどの程度戻るかを計算した。

初期条件から $t = 80$ まで計算した。最大刻み幅 $h_{max} = 0.1362$、最小刻み幅 $h_{min} = 1.7092 \times 10^{-7}$ で 10633 回 Taylor 展開を計算する必要があった。その時点の速度の符号を変えた値を初期値として $t = 0$ まで計算した。この計算には 10635 回 Taylor 展開する必要があった。Taylor 展開の計算回数はほぼ同じだが、逆計算の方が 2 回多かった。ここで計算した初期値と元の初期値との差は最大で 2.23×10^{-18} で約 17 桁一致した。この結果から計算

は 17 桁程度正しいと思われる。

さらに途中のエネルギーが保存されているかどうかを確かめた。エネルギー $E = -\frac{769}{60} = -12.81666\cdots$ との相対誤差は、最大で 1.2×10^{-26} であった。ほぼ 25 桁の精度でエネルギーが一定であった。他の計算より精度良くエネルギー保存則が成りたっていることがわかる。

物体が最接近する時間と距離を計算する。微分方程式の計算途中で、$t = 15.829920$ 付近で、物体 4 と物体 5 の距離 $r45$ が最小になることがわかるので、この時点で $r45$ を時間の Taylor 展開式を計算する。Taylor 展開式は、次の 24 次の Taylor 展開式となる。

$$\begin{aligned} r45 &= 4.1403728 \times 10^{-4} - 4.7226778t + 2.6220212 \times 10^{7}t^{2} \\ &+ 3.9888548 \times 10^{11}t^{3} + \cdots - 1.6455306 \times 10^{128}t^{24} \end{aligned}$$

定数項は 4.1403728×10^{-4} と小さいが、24 次の係数は $-1.6455306 \times 10^{128}$ と非常に大きな数値なる。このことから、刻み幅 h は非常に小さく採る必要があることがわかる。

これを微分すると次の 23 次式が得られる。

$$\begin{aligned} r45' &= -4.7226778 + 5.2440424 \times 10^{7}t + 1.1966564^{12}t^{2} \\ &- 4.4087319 \times 10^{18}t^{3} + \cdots - 3.9492734 \times 10^{129}t^{23} \end{aligned}$$

この Taylor 展開式の零点を求めるために、逆関数の Taylor 展開式 $i45$ を計算する。Taylor 級数の逆関数は比較的簡単に求められる。逆関数を計算すると、展開位置が -4.7226778 になるので、簡単化のために、以下では $a = -4.7226778$ としてある。

$$\begin{aligned} i45 &= 1.9069259 \times 10^{-8}(t-a) - 8.2979517 \times 10^{-12}(t-a)^{2} \\ &+ 5.9019532 \times 10^{-13}(t-a)^{3} + \cdots \\ &- 2.3526647 \times 10^{-56}(t-a)^{24} \end{aligned}$$

逆関数に $t = 0$ を代入して零点の相対位置 z を求めると以下のようになる。

$$z = 8.99347711832073120 \times 10^{-8}$$

したがって、零点の位置 t_{zero} は、次のようになる。

$$t_{zero} = 15.8299202715809$$

そのときの、物体間の距離 $r45$ を $r45$ の Taylor 展開式に代入して、次のように求まる。

$$r45 = 4.13824836258701 \times 10^{-4}$$

このように、Szebehely の結果 $(t_{zero} = 15.8230, r45 = 4 \times 10^{-4})$ を高精度することができた。

5. まとめ

ピラゴラスの 3 体問題を Levi-Civita 変換などの座標変換を行わないで高次公式と 4 倍精度数を使うことによって、十分な精度で計算出来た。

今回はピラゴラスの 3 体問題についての計算であるが、この方法は、精度良く解き難い多くの常微分方程式を高精度で十分な精度で解くことができると思われる。

Taylor 展開法は、容易に高次の計算が可能で、Runge-Kutta にはない特徴でこのような問題には最適と思われる。

現在主流の CPU は高速にはなっているが残念ながら 4 倍精度の計算をハードウエアとして実装されていない。多くの範囲の計算を容易に行うためには必須の機能であると思われる。

25 桁程度の結果を期待して、要求精度 10^{-28} として計算したが、得られる結果は約 17 桁程度であった。約 8 桁の精度が失われている。この理由を調べるのが今後の課題である。

参考文献

[1] Ellis M.A. and Stroustrup B ： The Annotated C++ Reference Manual, Addison-Wesley, 1990
[2] Gisela Engeln-Mullges Frank Uhlig ：Numerical Algorithms with Fortran, Springer(1996)
[3] Hairer E., Wanner G., Solving Ordinary Differential Equations II, Springer-Verlag, 1991
[4] 平山弘, 小宮聖司、佐藤創太郎、Taylor 級数法による常微分方程式の解法, 日本応用数理学会、Vol 12. No.1, 1–8,(2002)
[5] 長沢工、桧山 澄子：パソコンで見る天体の動き、地人書館 (1992)
[6] 小武守, 長谷川, 藤井, 西田, 反復法ライブラリ向け 4 倍精度演算の実装と SSE2 を用いた高速化, 情報処理学会誌 コンピューティングシステム,1(2008),73–84
[7] 山田進, 佐々成正, 今村俊幸, 町田昌彦, 4 倍精度基本線形代数ルーチン群 QPBLAS の紹介とアプリケーションへの応用, 情報処理学会研究報告,vol.2012-HPC-137,No.23(2012)
[8] 大野博, 25 段 12 次陽的ルンゲ・クッタ法構成の試み, 日本応用数理学会論文誌, 16(2006), 177–186
[9] 佐野理, キーポイント微分方程式, 岩波書店, 東京, (1993)
[10] Szebehely, V., Burrau's Problem of Three Bodies, Proceedings of the National Academy of Sciences of the United States of America, vol. 58, Issue 1, 60-65(1967)