Vol.2019-HPC-169 No.2
2019/5/10

Taylor展開法による常微分方程式の数値解法の高速化

平山 弘[1,a)]

概要：C++言語や Fortran 90 を使えば、Taylor 級数の四則演算や関数計算を容易に定義できる。この定義を使えば四則演算、標準関数および条件文等で記述された関数を Taylor 級数に展開できる。
この Taylor 級数法を使えば、微分方程式 $\frac{d\mathbf{y}}{dx} = \mathbf{f}(x, \mathbf{y})$ の解を Taylor 級数に展開できる。Taylor 級数解は任意の次数まで計算出来るので、Runge-Kutta 法等に代わる任意次数の計算法として利用できる。Taylor 展開式は誤差や与えられた許容誤差内での最良のステップの大きさの計算に容易に使える。
この方法を無駄を排し、高速化すると常微分方程式の級数解法と同様になる。常微分方程式の級数解法は、筆算で行うためか、Taylor 級数の加減乗算の範囲で行われている。この方法は計算機を使えば、除算だけでなく、指数対数関数、三角関数等を含む場合も容易に効率的に解くことができる。
実際の具体例では、6 段 5 次の Fehlberg の公式と比較して、10 倍以上の高速化を図ることができた。

キーワード：常微分方程式，高次数値解法，Taylor 展開法

1. はじめに

常微分方程式の一段法の数値計算法として、現在 Runge-Kutta 法がよく使われている。その他に Taylor 展開式を解法 [3] が知られている。

これらの解法にはそれぞれ次のような特徴がある。Runge-Kutta 法は、いろいろな次数の公式が存在するが、次数毎に計算公式の係数をかなり計算量の大きい計算を行い決定しなければならない。たとえば、25 段 12 次の公式の係数を計算するには 7813 個の非線形方程式 [9] を解かなければならない。各次数の公式は独立性が高く、1 次だけ次数が高いだけでも、共通部分があまりない。一部共通範囲がある公式を埋め込み型と呼ばれている。僅かな追加計算でその計算精度が推定できる公式としてよく使われる。この型の公式として Fehlberg の公式 [2] がよく知られている。この参考文献 [2] には Runge-Kutta の公式の係数の表が多数掲載されている。

Taylor 展開法では、1 個のプログラムで何次でも計算可能である。展開式は途中の次数まで共通であるから、すべての式が誤差評価可能でその計算の誤差推定が容易である。

Taylor 展開法はこのように良い点がいろいろあるが、プログラミングに現れるすべての関数の Taylor 展開計算法だけでなく、四則演算や平方根などの計算法もユーザーがプログラミングしなければならないなどの問題点がある。

このような問題点の解決策として、著者等は Taylor 展開計算のプログラムは東北大の計算センターのライブラリとして登録 [4] している。

2. 関数の Taylor 展開法

関数 $f(x)$ は、$f((x-a)+a)$ を計算することによって、$x=a$ における Taylor 展開式を次のように計算することができる。

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f^{(n)}(a)}{n!}(x-a)^n + \cdots$$

ここでは、この方法を代入法と呼ぶことにする。

Taylor 展開を単純化するため展開位置を原点になるように平行移動して、計算を行う。

$$f(x+a) = f(a) + \frac{f'(a)}{1!}x + \frac{f''(a)}{2!}x^2 + \cdots + \frac{f^{(n)}(a)}{n!}x^n + \cdots$$

関数 $f(x)$ の点 $x=a$ における Taylor 展開式は、x に $x+a$ を代入することによって計算出来る。

例えば、関数 $f(x)=x^3$ を $x=1$ で Taylor 展開すると、次のようになる。

$$f(x+1) = (x+1)^3 = 1 + 3x + 3x^2 + x^3$$

[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@sd.kanagawa-it.ac.jp

すなわち、

$$f(x) = x^3 = 1 + 3(x-1) + 3(x-1)^2 + (x-1)^3$$

この計算法は、汎用性があり、いろいろな関数を Taylor 展開することができる。

例として、$f(x) = e^x$ として $f(x+1)$ の Taylor 展開式を計算する。

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots \quad (1)$$

(1) の 6 次まで式に、x の代わりに $x+1$ を代入すると次の式が得られる。

$$2.71806 + 2.71667x + 1.35417x^2 + 0.444444x^3 + 0.104167x^4 + 0.0166667x^5 + 0.00138889x^6$$

この展開式は、(1) の 6 次まで採った式の Taylor 展開としては正確な Taylor 展開式であるが、e^x の正確な Taylor 展開式を比較すると定数項は 4 桁、1 次と 2 次の係数は 3 桁、3 次と 4 次の係数は 1 桁しか合っていない。それ以降は 1 桁も合っていない。これが倍精度 (15 桁) 精度計算するには 100 次程度まで計算する必要がある場合がある。

計算は関数の解析接続するため計算と同じで、「関数を解析接続するためには、非常に高次の Taylor 展開式を必要とする。」の事実と一致する。

この代入法は、精度よく計算するには、収束が良い展開式で、非常に次数の高い式を利用して計算しなければならないことがわかる。

Taylor 展開式でなくとも、計算可能な式ならば容易に Taylor 展開式を計算出来る。たとえば、Riemann の zeta 関数は、$\zeta(x) = \sum_{k=1}^{\infty} \frac{1}{k^x}$ と定義される関数で、次のように収束の良い形に変形できる。(https://en.wikipedia.org/wiki/Riemann_zeta_function)

$$\zeta(x) = \frac{1}{1-2^{1-x}} \sum_{n=0}^{\infty} \left(\sum_{k=0}^{n} (-1)^k \binom{n}{k} (1+k)^{-x} \right)$$

この関数の x に $x+2$ を代入して計算すると次のようになる。4 次まで表示すると次の式となる。

$$\zeta(x) = 1.64493 - 0.937548(x-2) + 0.99464(x-2)^2 - 1.00002(x-2)^3 + 1.00006(x-2)^4$$

このように、原理的には容易に Taylor 展開が求められる。定数項はよく知られているように $\frac{\pi^2}{6} = 1.64493406684822643647\cdots$ となる。

3. 常微分方程式の解の Taylor 展開

次の微分方程式の解の Taylor 展開式を計算する。

$$y'(x) = f(x, y(x)) \qquad y_0(x) = y_0 \quad (2)$$

この常微分方程式の解はピカール (Picard) の逐次近似法で求めることができる。

$$y_{n+1}(x) = y_0 + \int_0^x f(t, y_n(t))dt \quad (3)$$

漸化式 (3) の 1 回の計算で最低でも近似精度が 1 次上がる。2 次以上あがる問題もある。

例として、次の微分方程式を Picard の逐次近似法で解く。

$$y'(x) = y(x) + 1 \quad y_0(0) = 1 \quad \text{解: } y(x) = 2e^x - 1 \quad (4)$$

Picard の逐次近似法の漸化式は

$$y_{n+1}(x) = 1 + \int_0^x (y_n(t) + 1)dt \quad (5)$$

$n = 0, 1, \cdots$ として、漸化式 (5) を計算すると

$$\begin{aligned} y_1(x) &= 1 + 2x \\ y_2(x) &= 1 + 2x + x^2 \\ y_3(x) &= 1 + 2x + x^2 + \frac{x^3}{3} \\ y_4(x) &= 1 + 2x + x^2 + \frac{x^3}{3} + \frac{x^4}{12} \end{aligned}$$

このように 1 回の反復毎に 1 次次数が増えることがわかる。上の式では、係数は分数になっているが、実際の計算は倍精度浮動小点数で計算される。

1 回の反復で 2 次次数が増加する例を挙げる。

$$y'(x) = (y(x))^2 + 1 \quad y(0) = 0 \quad \text{解: } y(x) = \tan x \quad (6)$$

この方程式の Picard の逐次近似法の漸化式は次のようになる。

$$\begin{aligned} y_0(x) &= 0 \\ y_{n+1}(x) &= \int_0^x ((y_n(t))^2 + 1)dt \end{aligned} \quad (7)$$

$n = 0, 1, \cdots$ として、漸化式 (7) を計算すると

$$\begin{aligned} y_1(x) &= x \\ y_2(x) &= x + \frac{x^3}{3} \\ y_3(x) &= x + \frac{x^3}{3} + \frac{2x^5}{15} + \frac{x^7}{63} \\ y_4(x) &= x + \frac{x^3}{3} + \frac{2x^5}{15} + \frac{17x^7}{315} + \frac{38x^9}{2835} + \cdots + \frac{x^{15}}{59535} \end{aligned}$$

$y_1(x)$ は 1 次まで、$y_2(x)$ は 3 次まで、$y_3(x)$ は 5 次まで、正しく計算できる。$y_1(x)$ は 1 次まで正しい式であるから、(7) の被積分関数は 1 次まで正しく計算出来る。それを積分した関数は 2 次まで正しく計算出来る。

この問題の場合、解は奇関数であるため、$y_1(x)$ は 1 次式であるが、2 次の項は零なので、2 次まで正しい式となっている。このため、(7) の被積分関数は 2 次まで正しく計算出来る。それを積分すれば 3 次まで正しい式を計算出来る。そのため、$y_2(x)$ は 3 次まで正しい式が求めることができる。

$y_4(x)$ の計算するために、$y_3(x)$ を (7) に代入し 6 次まで正しく計算する。6 次まで正しく計算するためには 7 次以上の項は不要なので零と置いて計算することができる。同様に $y_4(x)$ の 9 次以上の項は零とおいて効率的に計算出来る。

(7) の計算で、$y_n(x)$ から $y_{n+1}(x)$ を計算する場合、2 乗の項の計算は、$(2n-1)$ 次の部分は共通なので、$(2n+1)$ 次の部分だけを計算することによって効率化することができる。このように 2 次づつ解の次数が上がるのは、初期値が $y(0)=0$ の場合で、$y(0)=1$ の場合を計算するとわかるように一般には 1 次づつ増加する。

(6) の方程式の解を $y(x)=a_0+a_1x+a_2x^2+\cdots$ であるとして代入し、x^n に係数を等しいと置くと、定数項 $(n=0)$ および n 次の係数から次の式が得られる。

$$\begin{aligned} a_1 &= 1+a_0 \\ a_{n+1} &= \frac{1}{n+1}\sum_{k=0}^{n} a_k a_{n-k} \end{aligned} \tag{8}$$

この公式を使えば最も効率的に計算出来ることがわかる。(8) の積和の部分は、対称性からさらに約半分の計算量を減らせる。この方法は、常微分方程式の級数展開法そのものであることがわかる。この計算法は、効率的ではあるが通常筆算で行うため加減乗算だけ、しかも簡単な乗算を含む問題だけに使われて来た。

計算機を使えば、この方法で多くの微分方程式でも、解くことができる。筆算で解ける微分方程式の Taylor 級数解の第 n 項の係数は簡単な n の式で表されることが多いが、計算機で解く複雑な微分方程式の級数解の係数は一般には、n の簡単な関数にはならない。ここでは、この方法が常微分方程式の解法の効率的な解法であることを示す。

4. Taylor 級数の計算

平行移動によって、展開位置を原点移すことができるので一般性を失うことなしに、原点で展開した式だけを扱うことができる。この級数を次のように定義する。

$$\begin{aligned} f(x) &= f_0+f_1x+f_2x^2+f_3x^3+\cdots \\ g(x) &= g_0+g_1x+g_2x^2+g_3x^3+\cdots \\ h(x) &= h_0+h_1x+h_2x^2+h_3x^3+\cdots \end{aligned} \tag{9}$$

加減算

$h(x)=f(x)\pm g(x)$ は係数間の加減算を計算することができるので、次のようになる。

$$h_n = f_n \pm g_n \quad (n=0,1,2,\cdots)$$

乗算

乗算 $h(x)=f(x)g(x)$ は簡単にでき、次のようになる。

$$h_n = \sum_{k=0}^{n} f_k g_{n-k} \quad (n=0,1,2,\cdots)$$

除算

乗算 $h(x)=\frac{f(x)}{g(x)}$ は $h(x)g(x)=f(x)$ と変形し、(9) を代入し各次数の係数を比較することによって、次の式が得られる。

$$\begin{aligned} h_0 &= \frac{f_0}{g_0}, \\ h_n &= \frac{1}{g_0}\left(f_n-\sum_{k=1}^{n} g_k h_{n-k}\right) \quad (n=1,2,\cdots) \end{aligned}$$

べき乗

$h(x)=f(x)^a$ (a は定数) を計算する。この計算式を微分すると $h'(x)=af(x)^{a-1}f'(x)$ となる。$h(x)=f(x)^a$ であるから $h'(x)=ah(x)f'(x)/f(x)$ となる。したがって $h'(x)f(x)=ah(x)f'(x)$ となる。この式に (9) を代入し各次数の係数を比較することによって、次の式が得られる。

$$\begin{aligned} h_0 &= f_0^a, \\ h_n &= \frac{1}{nf_0}\left(\sum_{k=1}^{n}((a+1)k-n)f_k h_{n-k}\right) \quad (n=1,2,\cdots) \end{aligned}$$

$a=\frac{1}{2}$ と置くと平方根も計算出来る。Taylor 展開式の平方根については、ニュートン法によって効率的に計算する方法がある。

指数対数、三角関数等も同様に計算出来る。

5. 数値例

ここでは、誤差評価を簡単にするため、解析解が知られている二体問題 (Kepler equation)[5] を解くことを考える。方程式は次のようになる。

$$\begin{aligned} \frac{dx_1}{dt} &= x_3, & \frac{dx_2}{dt} &= x_4, \\ \frac{dx_2}{dt} &= -\frac{x_1}{r^3}, & \frac{dx_4}{dt} &= -\frac{x_2}{r^3}, \\ & & r &= \sqrt{x_1^2+x_2^2} \end{aligned} \tag{10}$$

初期値は、e が $0\le e<1$ を満たす実数とする。初期値が以下の式で表されるとする。

$$\begin{aligned} x_1(0) &= 1-e, \quad x_2(0)=x_3(0)=0, \\ x_4(0) &= \sqrt{\frac{1+e}{1-e}} \end{aligned} \tag{11}$$

このとき、解析解は、$E(t)$ が $E(t)-e\sin E(t)=t$ を満たすとき、次のように書ける。

$$\begin{aligned} x_1(t) &= \cos E(t)-e, \\ x_2(t) &= \sqrt{1-e^2}\sin E(t), \\ x_3(t) &= \frac{\sin E(t)}{e\cos E(t)-1}, \\ x_4(t) &= \frac{\sqrt{1-e^2}\cos E(t)}{1-e\cos E(t)} \end{aligned} \tag{12}$$

この問題を $e=0.5$ の場合を解いた。区間 [0,10] の分割数 N を変えて計算した。最大誤差を次のように定義する。

$$\epsilon_N = \max_{1\le n\le N} |x_n - x(t_n)|_\infty \tag{13}$$

このときの Rung-Kutta 法といろいろな次数の Taylor 展開法の $-\log_2 \epsilon_N$ の値を表 1 に示す。最初の Ruge-Kutta 法の結果は三井等 [5] による結果である。このように、高次の Taylor 級数を計算することによって分割数が少ない場合でも高精度で解くことができる。

また、10 次の Taylor 級数が得られるならば、9 次までの計算結果を比較することによって誤差が推定できるなど Taylor 展開法には Runge-Kutta 法にはない特徴がある。

6. 長時間計算

Taylor 展開法を使って長時間の計算を行った。問題は前問と同じ Kepler の問題を扱った。e=0.9 の場合を区間 [0,10],[0,100][0,1000] の間を計算し、最後の点に達したときの計算時間、誤差を求めた。RK 法として、6 段 5 次の Runge-Kutta-Fehlberg 法を使った刻み幅自動調節プログラム [6][10] を使い、許容誤差 1.0e-12 として計算した。Taylor 法でも刻み幅自動調節を行った。Taylor 級数の最後の項が許容誤差以下になるように刻み幅を決めて計算した。すなわち、展開位置が p で最後の項の次数が m の時、次の式が成り立つ。

$$|a_m(x-p)^m| \le \epsilon$$

$h = x - p$ と置くと、この式から刻み幅 h が得られる。

$$h \le \sqrt[m]{\frac{\epsilon}{|a_m|}}$$

Taylor 級数の係数の m 次の係数だけが、特別の大きかったり小さかったりする可能性がある。この場合、$m-1$ の係数に対しても同様に刻み幅を計算し、その中で小さい方を刻み幅 h とするようにする。ある程度次数が高い場合は、この方法が使うことができるが、次数が低い場合、この方法はあまり効果的ではない場合が多い。

この方法を使うと 5 次の Taylor 展開法は 5 次の項が刻み幅の計算に使われ実質 4 次以下の計算となり非常に時間のかかる計算になることがある。

この刻み幅決定方法は、一様に係数が変化すると仮定しているため、途中でその一様性が破られた場合、その刻み幅があまり適切でない場合がある。特に 5 次程度の低次の Taylor 展開法では、この一様性の仮定があまりよく成り立たない場合がある。

表 2 の測定では計算機として、Intel Core i7-8700K 3.7GHz を使用した。

表 1 Kepler 問題の最大誤差

N	160	640	1280	2560	5120
4th Rung	13.99	18.21	22.34	26.40	30.44
4th Tay	9.88	13.84	17.82	21.81	25.81
10th Tay	35.20	44.10	43.06		
15th Tay	44.38	45.05	43.06		
20th Tay	44.38				

6.1 各次数の係数の計算

プログラムは C++言語で記述し、クラス機能を使って Taylor 級数を定義した。無駄な計算を省くため、計算すべき係数の次数をグローバル変数 (n) とし、各関数では、その次数の係数を計算するように作成した。

例えば Taylor 級数 x,y の加算は次のような形になる。

```
taylor operator+( const taylor &x, const taylor &y )
{
    taylor& tmp = stack[stack_pointer++] ;
    tmp[n] = x[n] + y[n] ;
    return tmp ;
}
```

Taylor 級数の加算は、n 次の係数 (x[n],y[n]) を加えるという単純な計算であるが、n 次未満の係数も返す必要があるので、これまで計算した Taylor 級数の係数をスタックにとって置き、そのスタックの n 次の係数を書き換えるようにプログラムを作成する必要がある。

Taylor 級数 x,y の乗算は次のような形になる。

```
taylor operator*( const taylor &x, const taylor &y )
{
    taylor& tmp = stack[stack_pointer++] ;
    tmp[n] = 0 ;
    for( int i=0 ; i<=n ; i++ )
        tmp[n] += x[i]*y[n-i] ;
    return tmp ;
}
```

Taylor 級数の乗算では、2 つの係数 (x[i],y[n-1]) の積を加え、それをスタックの n 次の係数に代入する計算を行う。加算よりも複雑な計算なので、スタック操作は相対的に無視できるが加減算などでは、そのスタック操作は無視できない。単純な負の符号 (-) は、単に符号を変えるだけの関数であるがスタック操作が入るため、相対的にかなり遅い計算になる。

このようなプログラムを作成すると、通常のプログラミング言語で微分方程式の右辺を定義するのとほとんど変わらないように記述できるようになる。

ここで扱っている Kepler の問題は、次のように記述できる。

```
void func( const taylor *y, taylor *f )
{
    static taylor r2, r15 ;
    f[0] = y[2] ;
    f[1] = y[3] ;
    r2 = square(y[0])+square(y[1]) ;
    r15 = pow( r2, -1.5 ) ;
    f[2] =-y[0]*r15 ;
    f[3] = -y[1]*r15 ;
}
```

ここで、関数 square(x) は、Taylor 級数 x を 2 乗する関数である。

中間変数は、それまでの次数の係数を保存している必要があるため、静的変数 (static) と宣言しておく必要がある。

比較のために、Runge-Kuta 法で解くために作成した関数を以下に示す。

```
void func( const double *y, double *dy )
{
    double r2, r15 ;
    dy[0] =y[2] ;
    dy[1] =y[3] ;
    r2 = y[0]*y[0]+y[1]*y[1] ;
    r15 = pow( r2, -1.5 ) ;
    dy[2] =-y[0]*r15 ;
    dy[3] = -y[1]*r15 ;
}
```

このように、Taylor 展開式を使って常微分方程式を解くために定義しなければならない関数は Taylor 展開法と比較すると宣言部分が変わるだけでほぼ同じ関数になる。

6.2 計算実行結果

この計算を行うために、コンパイラーとして、Visual Studio C++ 2017 を使用した。このコンパイラによって 1 コア用のオブジェクトコードを生成した。このプログラムを使って、次数が 5 次から 29 次まで Taylor 展開法を使って計算した。その中の 5 次、10 次、15 次、20 次、25 次の計算結果を表 2 に示した。

この結果から、5 次の低次の場合は、相対的に、スタック操作等のオーバーヘッドのため、RKF 法による計算と比較し、約 3 倍程度時間がかかった。この問題では 15 次以上の Taylor 展開法を使うと長時間に渡って、高精度で、高速に計算が可能であることがわかる。

計算次数が 20 次の時が計算時間が最小であった。区間 [0,10000] のときの計算速度は、6 段 5 次の Runge-Kutta-Fehlberg 法の 16.7 倍であった。このように、高次の Taylor 展開法を使えば、10 倍以上の速度で計算することができる可能性がある。

使い易くするために、低次の計算ではオーバーヘッドの多いプログラムとなった。このオーバーヘッドをなくすには、方程式の Taylor 展開するプログラムを直接書くことによって行うことができる。この方法は、多くの実際に計算されている微分方程式が多くの部分が線形で、比較的単純なので、容易に出来る可能性がある。今回の Kepler の問題では、平方根の計算 (pow(x,a)) の部分は面倒であるが他の部分は比較的容易に行える。

7. おわりに

Taylor 展開法は、高次の計算が容易なので、高精度で高速な計算が可能である。Kepler の問題については、10 倍以上の性能が発揮することが示すことができた。常微分方程式の高速高精度な数値計算には Taylor 展開法を使うべきである。

常微分方程式以外の数値計算で高次計算が容易な分野では 10 次や 20 次の公式を使うのは当たり前であるが、常微分方程式では、よく使われる Runge-Kutta の公式が 4 次の公式であるためか、4 次程度の公式を使うにのが当たり前になってなっている。これでは、長時間にわたる、高精度計算が非常に難しくなる。

Taylor 展開法は筆算で計算する方法としては昔から知られている方法であるが、計算機を使って Taylor 展開する法は現時点では、あまり参考資料がないためか、あまり使われているとは言えない。

Taylor 展開法は、微分代数方程式にも容易に適用出来できるため、それを強調する論文 [11][7][8] が多い。微分代数方程式は、Runge-Kutta 系列の解法では一部の問題を除いてほとんど不可能であるため、Taylor 展開法で解くことが

表 2 kepler 問題の計算時間と誤差

t	誤差	step 数	計算時間 (msec)	最小刻幅
RKF 法				
10.0	5.7e-12	25356	1.29	1.7e-05
100.0	1.8e0-9	260574	13.31	4.7e-07
1000.0	1.2e-07	2663118	136.75	9.8e-08
10000.0	4.3e-06	26582346	1181.50	2.9e-08
5th Taylor				
10.0	1.1e-13	27302	3.45	3.0e-04
100.0	4.3e-12	281592	34.39	3.0e-04
1000.0	5.9e-10	2872230	345.04	3.0e-04
10000.0	2.7e-08	28674280	3463.91	3.0e-04
10th Taylor				
10.0	2.6e-13	622	0.17	1.4e-02
100.0	9.3e-12	6356	1.43	1.4e-02
1000.0	1.2e-09	64708	14.23	1.4e-02
10000.0	6.5e-08	646014	140.78	1.4e-02
15th Taylor				
10.0	1.6e-13	216	0.076	4.3e-02
100.0	6.6e-12	2182	0.78	4.3e-02
1000.0	6.6e-10	22188	8.80	4.3e-02
10000.0	3.6e-08	221492	78.79	4.3e-02
20th Taylor				
10.0	8.8e-13	134	0.070	7.2e-02
100.0	1.0e-10	1318	0.72	7.2e-02
1000.0	8.2e-09	13384	7.20	7.2e-02
10000.0	4.2e-07	133602	70.58	7.2e-02
25th Taylor				
10.0	4.5e-12	100	0.075	9.8e-02
100.0	3.1e-10	982	0.76	9.8e-02
1000.0	2.0e-07	9956	7.60	9.8e-02
10000.0	1.2e-05	99362	75.49	9.8e-02

できることが強調され、比較対象もないためか計算速度はほとんど述べられていない。

参考文献

[1] Chang Y. F. and Corliss G. F., ATOMFT:Solving ODEs and DAEs using Taylor series., Comp. Math. Appl., 28(1994)209–233
[2] Gisela Engeln-Müllges, Frank Uhlig, Numerical Algorithms with Fortran, Springer, 1996
[3] 平山, 小宮, 佐藤, Taylor 級数法による常微分方程式の解法, 日本応用数理学会論文誌, 12(2002), 1–8.
[4] 平山, 舘野, 浅野, 川口, Taylor 級数演算ライブラリの使用法, 東北大学情報シナジーセンター大規模科学計算システム広報 SENAC, 40(2007) 29–68
[5] 三井、小藤、齋藤, 微分方程式による計算科学入門, 共立出版, (2005)
[6] 森正武, FORTRAN77 数値計算プログラミング, 岩波書店, (1987)
[7] Nedialkov N. S. and Pryce J. D., Solving differential-algebraic equations by Taylor series (I): Computing Taylor coefficients. BIT, 45(2005)561–591.
[8] Nedialkof N. S. and Pryce J. D., Solving Differential-Algebraic Equation by Taylor Series (III) : the DAETS Code, J. Numerical Analysis, Industrial and Appl. Math. 1(2007) 1–30
[9] 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
[10] 渡部, 名取, 小国, Fortran 77 による数値計算ソフトウエア, 丸善, (1989)
[11] Chang Y. F. and Corliss G. F., ATOMFT:Solving ODEs and DAEs using Taylor series., Comp. Math. Appl., 28(1994)209–233