計算機アーキテクチャ 142−18
ハイパフォーマンス
コンピューティング 85−18
（2001．3．9）

多倍長計算を用いたODE Solverに関する一考察

幸谷智紀* 満田賢一郎† 鈴木千里‡

2001年3月8日(木)–9日(金)

1 多倍長計算について

本講演では，「多倍長計算」を「CPUがハードウェアでサポートする以上の仮数部の桁数を持つ浮動小数点数の演算」という意味で使用する。

現状では，多倍長計算が必要になる場合というのは，丸め誤差を減らして高精度な計算を行う時に限られるだろう。

何故ならば，もし丸め誤差に関して不安定な，しかしシンプルではあるアルゴリズムAと，安定だが複雑なアルゴリズムBを比較する時，多倍長計算を使用してアルゴリズムAの不安定要素を減らしたとしても，そしてメモリ使用量は度外視し計算時間のみ考えたとしても，得ようする精度がCPUサポートの範囲内(倍精度程度)であるならば，では殆どのケースでは安定なアルゴリズムBが有利であると考えられるからである。

勿論，多倍長計算の速度が今以上に向上すれば，逆転する可能性もある。が，現状では既存のアルゴリズムで解決不可能な悪条件問題にのみ適用することを考えることにする。

1.1 多倍長計算ライブラリのベンチマークテスト

本稿で示される数値実験は全て，次の計算機環境下で行ったものである。

1. 多倍長計算ライブラリ: gmp[1] 2.0.2(一部 3.0.1) / BNCpack Ver.0.0.2
2. 本体: (CPU)AMD K6-2 300MHz, (RAM)64MB, (OS)Laser5 Linux 6.0
3. C コンパイラ: gcc egcs-2.91.66 19990314/Linux (egcs-1.1.2 release)
4. Fortran77 コンパイラ: g77 version egcs-2.91.66 19990314/Linux (egcs-1.1.2 release) (from FSF-g77 version 0.5.24-19981002)

多倍長計算ライブラリにgmp[2]を選んだ理由は

- 仮数部の桁数(2進bit数)が変数毎に設定できるため，デフォルトの桁数のスコープに縛られない
- 四則演算の性能が比較的良い

ことにある。なお，gmpでは多倍長の整数型(mpz_t)，有理数型(mpq_t)，浮動小数点型(mpf_t)をサポートしているが，今回は全て浮動小数点型(mpf_t)のみ使用した。

以下にベンチマークテストの結果を示す。なお，gmpは2進の桁数(bit数)を指定する。参考までに，IEEE754浮動小数点数の結果も載せておく。表の数字は全てMFLOPSである。

ベンチマークテストには自作のBASE Benchプログラムを使用した。これは乱数を発生させて二つの配列に格納し四則演算を行い，もう一つの配列に演算結果を代入するという過程を繰り返し，四則演算のMFLOPS値を計算するという極めて単純なプログラムである。メモリキャッシュの効果などについては特段考慮していない。

	IEEE	**gmp(mpf_t)**				
bits	53	64	128	256	512	1024
ADD	5.46	1.60	1.44	1.30	1.00	0.69
SUB	4.82	1.82	1.64	1.32	0.93	0.59
MUL	5.46	0.72	0.46	0.21	0.08	0.02
DIV	2.28	0.36	0.24	0.12	0.05	0.02

反面，

- 数値計算に必要な関数が殆どサポートされていない
- 全てC(C++にあらず)で記述されているので，プログラムが作りにくい

*静岡理工科大学
†システム計画研究所
‡静岡理工科大学
[1]最新情報はhttp://www.swox.com/gmp/を参照。

という欠点があり，数値計算に使用するには機能不足である。従ってgmpをサポートした数値計算用のライブラリが不可欠である。

そこで，筆者のBNCpack[2]を全面的に書き換えて，gmpをサポートした多倍長計算の機能を取り入れることにした。現在は基本的な線型計算 (ベクトル・行列の和・差・内積・ノルム・逆行列・LU分解・CG法) のみ，単精度・倍精度・多倍長のそれぞれ3種類の関数が提供されている。

次にCG法によるテストを行う。このテストはgmp 3.0.1を使用し，CG法のプログラムはBNCpack付属のMPFCG関数を使用した。テスト問題 $Ax = b$ の行列 A と真の解 x は以下の通り。

$$A = \begin{bmatrix} 500 & 499 & \cdots & 1 \\ 499 & 499 & \cdots & 1 \\ \vdots & \vdots & & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix}, x = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \quad (1)$$

残差と初期残差との比が $1.0e-10$ 以下になった時点で反復を停止させるように設定した。結果は以下の通りである。

		gmp(mpf_t)				
prec	**IEEE(53)**	128	192	256	384	512
sec	15	49	46	45	48	53
times	205	107	90	81	74	71

精度の増加が反復回数の減少に繋がっていることは明らかだが，この問題については，計算時間を減少させるにはほど遠い。但し，これは停止則の取り方によって結果も変わってくるので，一概には結論を下せない。条件数と丸め誤差の影響を慎重に見極める必要があろう。

1.2 多倍長計算のODEへの適用

多倍長計算を適用する，という手法は極めて単純明快なものである。しかし，その結果どの程度のアドバンテージを得ることが出来るかは，アルゴリズムや問題，計算機環境によって異なる。

前述の通り，多倍長計算を使う目的は，CPUサポートの精度では不足する問題 (所謂，悪条件問題) を解く，ということに限定する。即ち，多倍長計算を用いる以外，処方箋がない場合に限る。

今回はそのうち，ODEにおける悪条件問題として，Rössler Modelを計算した例を取り上げる。

[2] http://member.nifty.ne.jp/tkouya/bnc/を参照。

2 数値例

Rössler Modelは Chaos 現象が見られる比較的簡単な3次元力学系の一つである [7]。

$$\frac{d}{dt}\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} -(y+z) \\ x + \alpha y \\ \beta + z(x-\mu) \end{bmatrix} \quad (2)$$

ここでは $\alpha = \beta = 1/5$ と固定して考える。μ を 3, 4, 5としていくと，この常微分方程式の解の周期が増加し，その運動はChaosになることが知られている [7]。

ここでは初期値を $[1\ 0\ 0]^T$ とし，積分区間を $[0, 500]$ に設定して，6次の陽的 Runge-Kutta，陰的 Runge-Kutta法を使用して (2) を計算した。このとき，刻み幅 h を $h = 1/8192$ とした時の結果を示す。Fig.1が $\mu = 4$ の時の図，Fig.2が $\mu = 5$ の時の図である。

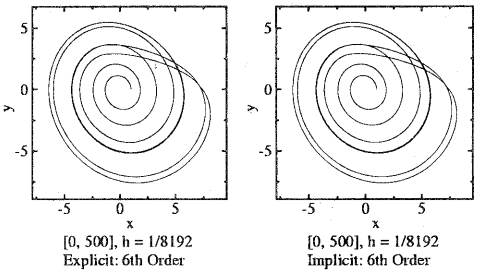

Figure 1: Explicit and Implicit: 6th Order

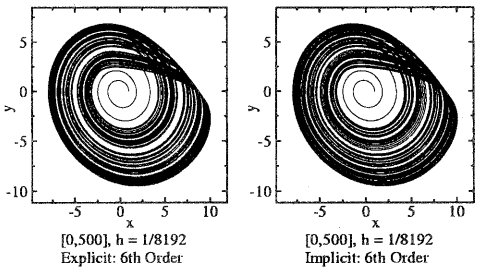

Figure 2: Explicit and Implicit: 6th Order

前述の通り，Rössler Modelは μ が 3, 4, 5,... となるにつれて軌道周期が増加し，Chaosになる。数値解もそれに歩調を合わせて，同じ刻み幅であっても次第に精度が悪くなる。実際，陽的・陰的 Runge-Kutta法のどちらも同じ軌道を再現できていない。

真の解が不明である常微分方程式の初期値問題に対して，どこまでが丸め誤差でどこまでが打切り誤差で

あるかを調べるには，次のようにして考えなければならない。

1. 積分区間を固定し，刻み幅を小さくしていき (例えば，$1/2, 1/2^2, 1/2^3 \ldots$ とする)，それぞれ数値解を求める。
2. 刻み幅が小さくなるにつれて，数値解の上の桁から数値が一致していく時，これを真の解として採用する。
3. 刻み幅に合わせて，一致する桁が増えてきているときには，不一致桁の上位が打切り誤差を現している。
4. ある程度を超えると，刻み幅を小さくしても一致する桁数が増えないか，一致する桁数が減少し出す。このとき，不一致桁の上位が丸め誤差を表わしている。

以上は数値計算の常識である。今回のケースでは一致している桁を得ることすら出来ていないため，このままでは丸め誤差と打ち切り誤差の切り分けが不可能である。よってまず切り分けすることを考える。

2.1 IEEE754 浮動小数点計算 — radau5 の結果

前述の結果は固定刻み，独自のプログラムによるものである。そこで，信頼性の高い，刻み幅制御を行っているプログラムによる検証もしてみる。ここでは，"Test Set for IVP Solvers: Release 2.1"[3]にある radau5.f を使用する。以下，その結果を示す (適宜編集してある)。精度は固定刻みの場合より改善されているように見受けられるが，それでも倍精度では不足していることがわかる。

```
1CWI Test Set for IVP Solvers (release 2.1)
 Solving Roessler Model using RADAU5
 User input:
 give relative error tolerance: 1.0e-15
 give absolute error tolerance: 1.0e-100
 give initial stepsize: 1.0e-3
                                          scd
        solution component          -----------
                                     abs   rel
 --------------------------------- ----- -----
 y(  1) =  0.2315033252997187E+001 -0.17  0.41
 y(  2) =  0.5584074912950650E+001 -0.24  0.34
 y(  3) =  0.1529261691852015E+000 -1.01  0.01
```

同様に，今度は積分区間を [0,100] にしてみる。今度は 11 桁程度の精度を得ることが出来た。

[3]http://www.cwi.nl/cwi/projects/IVPtestset/を参照。

```
1CWI Test Set for IVP Solvers (release 2.1)
 Solving Roessler Model using RADAU5
 User input:
 give relative error tolerance: 1.0e-15
 give absolute error tolerance: 1.0e-100
 give initial stepsize: 1.0e-3
                                          scd
        solution component          -----------
                                     abs   rel
 --------------------------------- ----- -----
 y(  1) =  0.2209934164417044E+001 10.79 11.13
 y(  2) = -0.1078306454586536E+002 11.04 12.08
 y(  3) =  0.3879297489491614E-001 12.98 11.57
```

2.2 gmp を使用した多倍長計算

IEEE 浮動小数点数を用いた計算では，丸め誤差が全ての桁を覆い尽くしてしまい，その刻み幅で打ち切り誤差が十分に小さくなっているかどうかは判然としないことが明らかとなった。よって，多倍長計算で丸め誤差を追いやった後は，打ち切り誤差について考慮しなくてはならないことになる。

$\mu = 5.7$ とし，先程と同じ 7 段 6 次陽的 Runge-Kutta 法を使用して，積分区間を $[0, 500]$ とした時の $[x\ y\ z]^T$ の値を計算した結果を以下の表に示す。下線部は丸め誤差と推定される部分を示す。

Explicit Runge-Kutta(7 steps, 6th order), 1/4096

	bits	
x	64	0.4200216265525560936e1
	128	0.377976408619610571385290835153 61628517e1
	256	0.377976408619610571367320738859 430944589443686130292283078891618 70091831326861e1
	512	0.377976408619610571367320738859 430944589443686130292283078838758 233635115202820139844739379466153 733388171563011098099672045367750 871589300081703851108862096e1

Explicit Runge-Kutta(7 steps, 6th order), 1/8192

	bits	
x	64	0.897092088142246549 2e0
	128	0.377976405520001126976329829788 31388409e1
	256	0.377976405520001126940489649563 257825224467004520406323114378 58062608729623888e1
	512	0.377976405520001126940489649563 257825224467004520406323114273 45529310893012836742216826618512 12136246918694400351208276401155 49074682669909264560793639847 3e1

表から，次のことが分かる。

1. 丸め誤差は 10 進 18〜19 桁程度の大きさである。故に，64bit 計算の結果は全く信用できない。
2. 128bit 以上の計算結果のうち，同じ計算桁数で刻み幅を小さくしていった結果の不一致桁の上位は打ち切り誤差を示している。刻み幅を 1/2 にすると 10 進 2 桁ずつ減少していく。

従って，128bit 以上の計算結果のうち刻み幅が 1/8192 のものも，打ち切り誤差は更に縮小させることができると予想される。しかし，陽的 Runge-Kutta 法で更に次数を上げるためにはより多くの段数を必要とし，段数と次数の差は拡大していく。そのため，多倍長計算においてはあまり効率の良い方法ではない。アルゴリズムが単純で，可変次数の公式が望まれる。

そのために補外法を使用する [5]。以下の計算は 128bit の浮動小数点数を使用して行われたものである。補外表の大きさは 4 段とした。この補外には Romberg 数列を使用しているため，4 段では 8 次程度の公式に相当する [5]。この表では，下線部が一致している桁を示す。

Extrapolation 4 Stages, 128bit

	Stepsize	
x	1/4096	0.377976405470760894482920610 00266069882e1
	1/8192	0.377976405470760905493547015 61087091259e1
y	1/4096	0.384281723713263047040226732 50611715412e1
	1/8192	0.384281723713263055437171714 79686423255e1
z	1/4096	0.104390900799393972686850428 212526679e2
	1/8192	0.104390900799393969437896498 84268313596e2

以上の結果より，一致している桁は上から 16〜17 桁である。全 38 桁のうち，下位桁 18〜19 桁は丸め誤差であることが示されており，桁の一致している上位桁は正確であると考えて良い。従って，ほぼ IEEE754 倍精度程度の精度を得ていると言える。参考までに z の相対丸め誤差と $\mu = 5.7$ の時の z を描いたグラフを Fig.3 に示す。μ の値によって丸め誤差の変動が著しく異なっていることが分かる。$\mu = 5, 5.7$ の時と $\mu = 4$ の時とを比較してみると，単なる丸め誤差の蓄積以上に増大が激しいことが分かる。

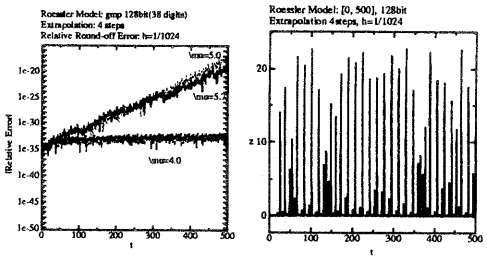

Figure 3: Relative round-off errors of z and z at $\mu = 5.7$

前のグラフのうち，積分区間 [0, 100] の部分を拡大したグラフを示す。

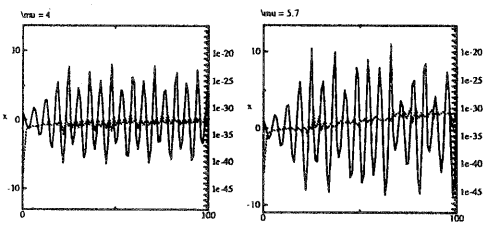

Figure 4: Relative round-off error of x and x

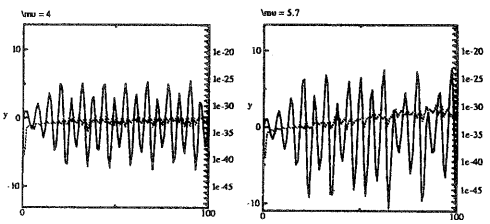

Figure 5: Relative round-off error of y and y

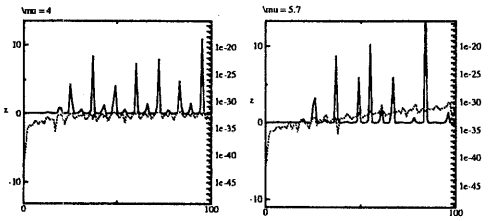

Figure 6: Relative round-off error of z and z

2.3 計算時間について

最後に計算時間を示す。同次数での比較のため，3段の補外法の計算時間も載せておく。

Computation Time (sec)

	Runge-Kutta	Extrapolation	
Stepsize: 1/1024			
	7-6	3 stage	4 stage
gmp 64 bits	146.69	167.08	364.3
gmp 128 bits	194.53	204.82	444.4
gmp 256 bits	328.71	308.55	664.8
Stepsize: 1/2048			
gmp 64 bits	293.32	334.13	727.1
gmp 128 bits	388.35	409.62	888.8
gmp 256 bits	657.34	617.09	1330.0

Rössler Model の場合，4段の補外法は7段6次の陽的 Runge-Kutta 法の倍近い時間が必要であるが，同次数の3段の場合はほぼ同時間で済む。

補外のように可変次数のアルゴリズムは，多倍長計算のように丸め誤差の影響を少なくすることが出来る環境では有用である。なおかつこのように計算時間においても固定次数のアルゴリズムと差があまりないのであれば，更に有用であろう。

3 今後の課題

今後の課題としては以下のものが挙げられる。

- 丸め誤差の影響の原因を，より初等的な観点で追及する。恐らくは，Lorenz Model についても同様のことが言えるのではないかと予想されるので，そこでも同様の考察を行う。区間の一部ではなく，全体での数値解の挙動を考慮に入れた丸め誤差解析 [6] が必要と思われる。
- 現状では，多倍長計算はソフトウェアによって行わざるを得ず，必要な桁数が多大になると計算時間もそれに応じて増加する。分散処理を活用した多倍長計算の環境が必要であり，それに応じた ODE Solver の実装を考える必要があろう。補外のような可変次数のアルゴリズムが有用であると予想される。

参考文献

[1] 幸谷智紀, 常微分方程式の初期値問題における多倍長計算の必要性についての一考察, 研究集会「工学における微分方程式の数値解析」口頭発表, 2000.

[2] T.Grandlund, TMG Datakonsult, GNU MP, The GNU Multiple Precision Arithmetic Library, Manuscript, 1996.

[3] 幸谷智紀・永坂秀子, IEEE754 規格を用いた丸め誤差測定法について, 日本応用数理学会論文誌, Vol.7, No.1, 1997.

[4] 幸谷智紀, 常微分方程式および固有値問題の高精度計算法の研究, 博士論文 (日本大学), 1997.

[5] 室伏誠, 有限桁計算における Richardson の補外法による丸め誤差評価の研究, 博士論文 (日本大学), 1998.

[6] 永坂秀子, 常微分方程式の数値計算における誤差伝播の解析, 博士論文 (日本大学), 1985.

[7] 下條隆嗣, カオス力学入門, 近代科学社, 1992.