Vol.2009-HPC-120 No.7
2009/6/12

反復改良法を用いた多倍長陰的 Runge-Kutta 法の性能評価

幸谷智紀[†1]

Buttari らは IEEE754 単精度と倍精度計算を組み合わせた混合精度反復改良法を提案し，単精度計算が倍精度計算比べて高性能な計算環境においては高速な計算が可能であることを示している．我々は倍精度計算と多倍長計算を組み合わせることで適用可能な問題の範囲を広げるとともに，より高精度な計算が高速に実行できることを示す．また，陰的 Runge-Kutta 法の内部反復にも本解法を適用し，任意精度計算が高速に実行できることも併せて示す．

Performance Evaluation of Multiple Precision Fully Implicit Runge-Kutta Methods using Mixed Precision Iterative Refinement

Tomonori Kouya[†1]

Buttari et al. have proposed the mixed precision iterative refinement using the IEEE754 single and double precision arithmetic for solving linear systems of equations. We broaden the scope of applications of the mixed precision iterative refinement by using a combination of double precision arithmetic and multiple precision arithmetic, and show that the new method has higher performance and yields more precise solutions than the original method. Finally, throughout our numerical experiments, we demonstrate that the implicit Runge-Kutta methods with the mixed precision iterative refinement can speed up.

†1 静岡理工科大学
Shizuoka Institute of Science and Technology

1. 初　め　に

反復改良法は 1967 年に Moler が提案したものが元になっている．n 次元方程式

$$\mathbf{f}(\mathbf{x}) = 0,\ \mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n \tag{1}$$

を解くため Newton 法を適用すると，その漸化式は

$$\mathbf{x}_{k+1} := \mathbf{x}_k - \left[\frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x}_k)\right]^{-1} \mathbf{f}(\mathbf{x}_k) \tag{2}$$

となる．ここで $\partial\mathbf{f}(\mathbf{x}_k)/\partial\mathbf{x}$ は Jacobi 行列である．

もし (1) が線型方程式

$$\mathbf{f}(\mathbf{x}) = A\mathbf{x} - \mathbf{b}$$

であれば，Jacobi 行列は定係数行列 A となるので，次のようなアルゴリズムで反復を進めることになる．

$$\mathbf{r}_k := \mathbf{b} - A\mathbf{x}_k \tag{3}$$

$$\text{Solve } A\mathbf{z}_k = \mathbf{r}_k \text{ for } \mathbf{z}_k \tag{4}$$

$$\mathbf{x}_{k+1} := \mathbf{x}_k + \mathbf{z}_k \tag{5}$$

これが連立一次方程式向けの反復改良法である．理論的には反復する必要はないが，現実に有限桁の浮動小数点数点演算を使用すると丸め誤差の影響で残差 $\mathbf{r}_k$ がゼロにならないため，これを最小化するように複数回の反復が行われる．そのため，近似解 $\mathbf{x}_k$ の精度を上げるためには，残差の計算は高精度で行う必要がある．

Buttari らは，A の条件数 $\kappa(A) = \|A\|\|A^{-1}\|$ が，使用する浮動小数点数の精度に比してあまり大きくない場合，(3) の残差計算の精度より (4) の計算精度を低くしても，収束するための十分条件が成立する (縮小写像になる) ことを示し，(4) と (5) を IEEE754 倍精度計算 (以下，倍精度と略記)，2 を IEEE754 単精度計算 (以下，単精度) することで全て倍精度計算で実行した時よりも計算効率が上がることをベンチマークテストで示した [2,7]．しかしこの計算効率の向上は，単精度計算が倍精度計算よりも格段に高速に実行できる環境でなければなしえないものである．例えば Cell Broadband Engine のような単精度計算が非常に高速な CPU や，Cache ヒット率の向上や SIMD 命令の使用によって性能向上が図られた ATLAS や GotoBLAS のような線型計算ライブラリを用いた環境がそれにあたる．

本稿では，まず Buttari らの提案する混合精度反復改良法の概要を示し，そのまま多倍長計算でも使用できることを確認する．次に実際の PC 環境で，倍精度計算でも反復改良法の収束が保証される良条件問題と，多倍長計算でなければ保証されない悪条件問題に対してベ

ンチマークテストを行い，近似解の精度とアルゴリズムの効率性がどの程度得られるのかを示し，性能向上比では倍精度と多倍長精度の組み合わせが最良であることを明らかにする．最後にこの解法のメリットが生かされる問題として，陰的 Runge-Kutta 法の内部反復に適用した場合のベンチマークテストを示す．

2. 混合精度反復改良法の概要

解くべき n 次元連立一次方程式を

$$A\mathbf{x} = \mathbf{b} \qquad A \in \mathbb{R}^{n\times n},\ \mathbf{x} \in \mathbb{R}^{n},\ \mathbf{b} \in \mathbb{R}^{n} \tag{6}$$

とし，係数行列 A は正則行列とする．本稿では，(6) における $A, \mathbf{b}$ の全成分は任意の精度を持つように設定できるものとする．

Buttari らは，混合精度反復改良法は，後述する収束条件を満足すれば，通常の連立一次方程式の解法を全て L 桁で計算した時に得られる近似解の精度と同程度の精度が得られるとしている．この時，計算の効率を上げるために，(4) の計算は $S(< L)$ 桁で実行する必要がある．この部分で使用する解法は，安定しているアルゴリズムが望ましいとしており，具体的には GMRES 法や直接法を挙げている．今回は部分ピボット選択を用いた LU 分解による直接解法を使用する．この時，(4) は，予め A を LU 分解しておくと

$$(PLU)\mathbf{z}_k = \mathbf{r}_k$$

となる．当然反復の前に $A = PLU$ として分解しておき (P は部分ピボット選択による行の入れ替えを表現する行列)，反復過程では前進・後退代入のみ行う．

以上をアルゴリズムの形でまとめると，(3)〜(5) は以下のようになる．ここで，$A^{[S]}, \mathbf{b}^{[L]}$ はそれぞれ S 桁，L 桁の浮動小数点数で表現した行列・ベクトルを意味する．

(1) $A^{[L]} := A, A^{[S]} := A^{[L]}, \mathbf{b}^{[L]} := \mathbf{b}, \mathbf{b}^{[S]} := \mathbf{b}^{[L]}$
(2) $A^{[S]} := P^{[S]}L^{[S]}U^{[S]}$
(3) Solve $(P^{[S]}L^{[S]}U^{[S]})\mathbf{x}_0^{[S]} = \mathbf{b}^{[S]}$ for $\mathbf{x}_0^{[S]}$
(4) $\mathbf{x}_0^{[L]} := \mathbf{x}_0^{[S]}$
(5) For $k = 0, 1, 2, \ldots$
  (a) $\mathbf{r}_k^{[L]} := \mathbf{b}^{[L]} - A\mathbf{x}_k^{[L]}$
  (b) $\mathbf{r}_k^{[S]} := \mathbf{r}_k^{[L]}$
  (c) Solve $(P^{[S]}L^{[S]}U^{[S]})\mathbf{z}_k^{[S]} = \mathbf{r}_k^{[S]}$ for $\mathbf{z}_k^{[S]}$
  (d) $\mathbf{z}_k^{[L]} := \mathbf{z}_k^{[S]}$
  (e) $\mathbf{x}_{k+1}^{[L]} := \mathbf{x}_k^{[L]} + \mathbf{z}_k^{[L]}$
  (f) Exit if $\|\mathbf{r}_k^{[L]}\|_2 \le \sqrt{n}\,\varepsilon_R\,\|A\|_F\|\mathbf{x}_k^{[L]}\|_2 + \varepsilon_A$

この S-L 桁混合精度反復改良法の収束条件 [2] より，S-L 桁混合精度反復改良法は，

$$\kappa(A)\varepsilon_S << 1 \tag{7}$$

でなければならないことが分かる．つまり，条件数 $\kappa(A)$ が大きければ，それに応じて S を大きく取れば良いことになるが，計算速度の向上は見込めなくなる．条件数が小さければ相応に S を小さくすることもできるが，そもそも L 桁も必要な計算なのかという疑問が湧いてくる．従って，S-L 桁混合精度反復改良法が有効なのは

- L 桁の精度が必要で，$\varepsilon_S^{-1} > \kappa(A)$ である時
- S, L が固定されており，S 桁計算が十分に L 桁計算より高速である環境にある時

に限られることが分かる (**図 1**).

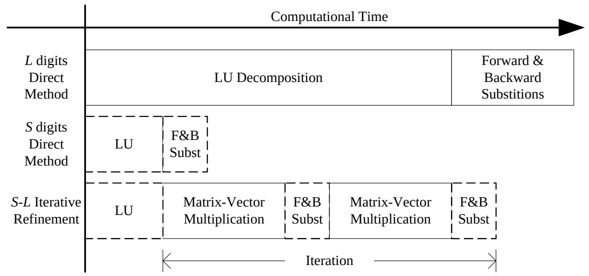

図 1 混合精度反復法の計算時間の構成
Fig. 1 Structure of Computational Time of Mixed Precision Iterative Refinement

一般に，IEEE754 単精度・倍精度計算に比べ，ソフトウェアで実現される四倍精度以上の多倍長計算は多大な計算時間を要する．そのため，ユーザの要求精度桁数 U が 4 倍精度以上であり，U に比して $\log_{10}\kappa(A)$ が小さければ，直接法を $L > U + \log_{10}\kappa(A)$ 桁で計算するよりも，$S > \log_{10}\kappa(A)$ 桁との混合精度反復改良法を用いた方が，より高速に同程度の精度の近似解を得られる可能性が高くなる．従って，条件数に応じて下記のような 3 種の組み合わせ (SP-DP, DP-MP, MP-MP) で混合精度反復改良法を用いることで，精度と計算時間の最適化を図ることが可能となる．

(1) $\kappa(A) < 10^7 \Longrightarrow$ 単精度 ($S = 7$)-倍精度 ($L = 15$): SP-DP 型

(2)　$\kappa(A) < 10^{15} \Longrightarrow$ 倍精度 ($S = 15$)-4 倍精度 or 多倍長精度 ($L > 30$): DP-MP 型

(3)　$\kappa(A) > 10^{15} \Longrightarrow$ 4 倍精度 or 多倍長精度 - 8 倍精度 or 多倍長精度 : MP-MP 型

Buttari らが提案した 1 の SP-DP 型以上に高速化が達成されると期待できるのは 2 の DP-MP 型及び 3 の MP-MP 型のケースである．特に DP-MP 型の場合は高速なハードウェアによる倍精度計算とソフトウェアによる低速な多倍長計算の組み合わせとなるため，この 3 つの中では最大の性能向上比を示すものと期待される．

但し，実際にこれらの組み合わせによってどの程度性能向上が図れるかどうかは，使用する計算機環境に強く依存する．図 1 に示したように，S 桁計算が L 桁計算より高速に実行できる程度に L/S 比が大きくなければ混合精度反復改良法を実行する意味がなくなる．そのため，本稿では $L/S \geq 2$ となるように設定して数値実験を行っている．L/S 比をどこまで小さくすれば性能向上が図れるかは OR 的な考察が求められるが，今後の課題としておく．

3. 性 能 評 価

以下では MP-MP, DP-MP 型反復改良法の性能評価を行う．数値実験を行った計算環境は下記の通りである．

CPU　Intel Core2Quad 6600

RAM　4GB

OS　CentOS 5.2 x86_64

C compiler　GCC 4.1.2(gcc, gfortran を使用)

Multiple Precision Library　MPFR 2.3.2 [9]/GMP 4.2.1 [1] + BNCpack 0.7b [5]

Linear Alegebra Computation Library　LAPACK 3.2 [8], ATLAS 3.8.3 [10]

単精度・倍精度計算は，CPU アーキテクチャを生かした高性能化を全く行っていない BNCpack(の倍精度計算ルーチン) と，オリジナルの LAPACK(gfortran でコンパイルしたもの) 及びそれを高性能化した ATLAS，そして GotoBLAS を使用して性能評価の比較検討を行う．Quad-core CPU ではあるが，マルチスレッド機能は使用していない．

多倍長計算は MPFR/GMP を用いた BNCpack の多倍長計算ルーチンを使用した．MPFR/GMP の多倍長浮動小数点演算は変数ごとに桁数を指定できるため，混合精度計算が自在にできるという特徴がある．

反復改良法の収束条件は下記のように最良の近似値が得られるよう設定した．

$$\varepsilon_R := \varepsilon_L,\ \varepsilon_A := 0 \tag{8}$$

3.1 MP-MP 型反復改良法の性能評価

条件数を固定して設定できるよう，一様乱数を用いて生成した正則行列 X と逆行列 X^{-1}，対角行列 $D = \mathrm{diag}(n, n-1, ..., 1)$ を用いて

$$A = XDX^{-1} \tag{9}$$

として密行列 A を作成した．これにより，常に $\kappa_2(A) = n$ という，次元数がさほど大きくなければ良条件の行列となる．また解ベクトルは $\mathbf{x} = [1\ 2\ ...\ n]^T$ とし，L 桁に正しく丸められた A 及び $\mathbf{b} = A\mathbf{x}$ を生成してテスト問題を作成している．メモリ容量の制約から，次元数 n は 4096 までにしてある．この問題では，全ての次元数・精度においても反復改良法は収束し，反復回数は 2〜3 に留まっている．

この良条件問題 (9) を用いて MP-MP 型反復改良法の性能を評価する．メモリの制約上，次元数は $n = 128, 256, 512, 1024$，桁数は $S = L/2$ と固定し，$L = 50, 100, 200$ 桁でそれぞれ問題を生成して数値実験を行った．直接法の計算時間は**表 1** に示す通りである．

表 1　多倍長精度直接法の計算時間 (秒)
Table 1　Computational Time of mutiple precision Direct Method (sec)

n	$L = 50$	100	200
128	0.15	0.24	0.46
256	1.81	1.97	5.66
512	13.83	23.59	44.7
1024	93.90	160.51	264.94

良条件問題に対して L 桁直接法と，MP-MP 型 ($L/2$-L 桁) 反復改良法を適用し，相対誤差と性能向上比を**図 2** に示す．

桁が大きいため相対的に目立たないが，相対誤差は直接法に比べて 2〜3 桁程度悪くなっている傾向がある．性能向上比は，多倍長計算の場合，計算桁数に比して計算時間が多く必要になるため，反復改良法の性能が直接法より悪くなることはなく，約 1〜2 倍程度まで計算性能が上がることが分かる．しかしこの問題では S をもっと小さくすることで計算時間をもっと短縮でき，DP-MP 型反復改良法が最も高速に実行できると予想される．

3.2 DP-MP 型反復改良法の性能評価

良条件問題 (9) を用いて DP-MP 型反復改良法の性能を評価する．LU 分解と反復ループ内の前進・後退代入を倍精度計算で，残差と近似解の更新を多倍長精度で行う．そのため，多倍長浮動小数点数の指数部長が倍精度のそれより長い場合は，残差のノルムが倍精度浮動

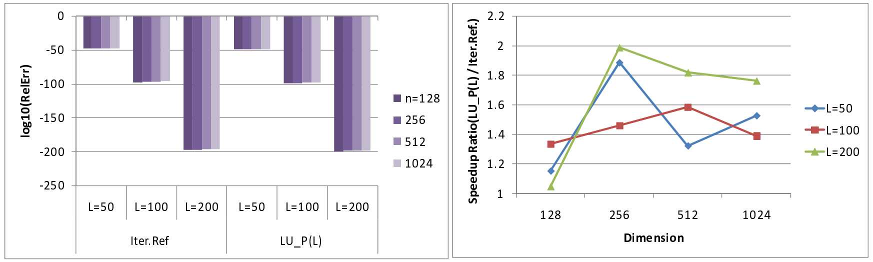

図 2 直接法および MP-MP 型反復改良法の相対誤差 (左) と MP-MP 型反復改良法の速度向上率 (右)

Fig. 2 Relative Error of mutiple precision Direct Method and MP-MP Iterative Refinement (left) and Speedup Ratio of MP-MP Iterative Refinement (right)

小数点数のアンダーフロー制限 (1.0×10^{-308}) 以下になると，反復ループ内の倍精度計算が実質的に無効となる．そのため，桁数は $L = 50, 100, 200$ まで，次元数は $n = 128$〜1024 までとして数値実験を行った．その結果を以下に示す．

まず近似解の相対誤差と反復回数を**表 2** に示す．比較のため，MP-MP 型反復改良法の結果も併せて示す．

相対誤差は MP-MP 型と殆ど差はなく，むしろ 2〜3 桁程度精度が向上していることが多い．反復改良法は収束の十分条件を満足すれば単調に収束し，L/S 比が大きいほど収束のスピードは遅くなっていくため，DP-MP 型は反復回数が MP-MP 型に比べて 2〜7 倍要している．MP-MP 型に比してどの程度計算性能が向上したかを**図 3** に示す．

全体として，計算桁数が大きくなればなるほど性能向上比は下がっていく．表 2 に示す通り，反復回数が増えるため，LU 分解の速度向上分を，反復ループ内の多倍長残差計算が打ち消しているためである．しかし，GotoBLAS を使用すれば 50 桁計算では約 30 倍，200 桁計算でも約 10 倍の性能向上がなされ，最低の性能向上比である BNCpack の倍精度計算でも 1.8〜4 倍の性能向上を達成している．

4. DP-MP 型反復改良法を用いた陰的 Runge-Kutta 法の性能評価

DP-MP 型反復改良法は，倍精度直接法で十分精度が得られる問題に対しては最も高速に実行できることが示されたが，これが有効に働くアルゴリズムとして陰的 Runge-Kutta(IRK) 法が挙げられる．

表 2 DP-MP 型反復改良法の $\log_{10}$(相対誤差) と反復回数 (括弧内)

Table 2 $\log_{10}$(Relative Error) and Iterative Times (in Parethesis) of DP-MP Iterative Refinement

	$L = 50$				
n	MP-MP	BNCpack	LAPACK	ATLAS	GotoBLAS
128	-47.63 (2)	-49.10 (4)	-49.02 (4)	-49.23 (4)	-49.21 (4)
256	-46.71 (2)	-48.84 (4)	-48.80 (4)	-48.74 (4)	-49.12 (4)
512	-47.24 (2)	-48.09 (4)	-48.41 (4)	-48.76 (4)	-48.79 (4)
1024	-46.96 (2)	-48.75 (4)	-48.61 (4)	-48.32 (4)	-48.36 (4)
n	$L = 100$				
128	-97.38 (2)	-98.94 (7)	-98.69 (7)	-98.93 (7)	-98.86 (7)
256	-96.93 (2)	-99.04 (7)	-98.96 (7)	-99.04 (7)	-98.94 (7)
512	-96.18 (2)	-98.00 (7)	-98.43 (7)	-98.62 (7)	-98.60 (7)
1024	-95.56 (2)	-98.66 (7)	-98.71 (7)	-98.60 (7)	-98.64 (7)
n	$L = 200$				
128	-197.39 (2)	-198.50 (14)	-198.59 (14)	-196.97 (13)	-198.64 (13)
256	-196.38 (2)	-198.65 (14)	-198.68 (14)	-198.71 (14)	-198.38 (13)
512	-196.13 (2)	-198.20 (14)	-198.04 (14)	-198.42 (14)	-198.56 (13)
1024	-196.11 (2)	-198.46 (14)	-198.52 (14)	-198.56 (14)	-195.25 (13)

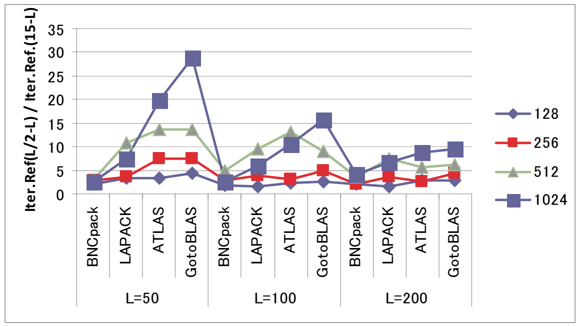

図 3 速度向上率: MP-MP / DP-MP

Fig. 3 Speedup ratio: MP-MP / DP-MP

n 次元常微分方程式の初期値問題

$$\begin{cases} \frac{d\mathbf{y}}{dx} &= \mathbf{f}(x, \mathbf{y}) \\ \mathbf{y}(x_0) &= \mathbf{y}_0 \end{cases} \tag{10}$$

を，積分区間 $[x_0, \alpha]$ を一定刻み $h = (\alpha - x_0)/(2 \cdot 4^l)$ で離散化して近似する時，$\mathbf{y}_i \approx \mathbf{y}(x_0 + ih)$ から $\mathbf{y}_{i+1}$ を求める際には

$$\begin{cases} \mathbf{k}_1 &= \mathbf{f}(x_i + c_1 h, \mathbf{y}_i + h \sum_{j=1}^{m} a_{1j}\mathbf{k}_j) \\ \mathbf{k}_2 &= \mathbf{f}(x_i + c_2 h, \mathbf{y}_i + h \sum_{j=1}^{m} a_{2j}\mathbf{k}_j) \\ &\vdots \\ \mathbf{k}_m &= \mathbf{f}(x_i + c_m h, \mathbf{y}_i + h \sum_{j=1}^{m} a_{mj}\mathbf{k}_j) \end{cases}$$

という非線型方程式を解き，

$$\mathbf{y}_{i+1} := \mathbf{y}_i + h \sum_{j=1}^{m} w_j \mathbf{k}_j$$

として $\mathbf{y}_{i+1}$ を求める．ここで m は公式の段数，c_p, a_{pq}, w_q は陰的 Runge-Kutta 法を決定する定数である．ここでは m 段 $2m$ 次 Gauss 型公式 [4] を使用する．この場合，係数 c_p は Legendre 直行多項式の零点である．w_q および a_{pq} は，c_p を補間点とする $m-1$ 次 Lagurange 補間多項式 $l_p(x)$ を用いて

$$w_q = \int_0^1 l_q(x)dx,\ a_{pq} = \int_0^{c_p} l_q(x)dx$$

と表現される．我々の実装では $m \geq 4$ の場合は高精度な Gauss-Legendre 積分公式の係数を求めるプログラム [6] を用いて Gauss 型 IRK 公式を導出している．

非線型方程式を Newton 法で解くためには

$$\begin{bmatrix} \mathbf{k}_1^{(l+1)} \\ \mathbf{k}_2^{(l+1)} \\ \vdots \\ \mathbf{k}_m^{(l+1)} \end{bmatrix} := \begin{bmatrix} \mathbf{k}_1^{(l)} \\ \mathbf{k}_2^{(l)} \\ \vdots \\ \mathbf{k}_m^{(l)} \end{bmatrix} - J^{-1}(\mathbf{k}_1^{(l)}, ..., \mathbf{k}_m^{(l)}) \begin{bmatrix} \mathbf{k}_1^{(l)} - \mathbf{f}(x_i + c_1 h, \mathbf{y}_i + h \sum_{j=1}^{m} a_{1j}\mathbf{k}_j^{(l)}) \\ \mathbf{k}_2^{(l)} - \mathbf{f}(x_i + c_2 h, \mathbf{y}_i + h \sum_{j=1}^{m} a_{2j}\mathbf{k}_j^{(l)}) \\ \vdots \\ \mathbf{k}_m^{(l)} - \mathbf{f}(x_i + c_m h, \mathbf{y}_i + h \sum_{j=1}^{m} a_{mj}\mathbf{k}_j^{(l)}) \end{bmatrix}$$

という反復を行う．ここで，$J(\mathbf{k}_1^{(l)}, \mathbf{k}_2^{(l)}, ..., \mathbf{k}_m^{(l)})$ は

$$J(\mathbf{k}_1^{(l}, \mathbf{k}_2^{(l)}, ..., \mathbf{k}_m^{(l)}) = \left[\begin{array}{c|c|c|c} I_n - J_{11} & -J_{12} & \cdots & -J_{1m} \\ \hline -J_{21} & I_n - J_{22} & \cdots & -J_{2m} \\ \hline \vdots & \vdots & & \vdots \\ \hline -J_{m1} & -J_{m2} & \cdots & I_n - J_{mm} \end{array}\right] \tag{11}$$

但し，I_n は n 次単位行列，J_{pq} は $\mathbf{f}$ の Jacobi 行列 $\partial\mathbf{f}/\partial\mathbf{y}$ を用いて

$$J_{pq} = ha_{pq}\frac{\partial}{\partial \mathbf{y}}\mathbf{f}(x_i + c_p h, \mathbf{y}_i + h\sum_{j=1}^{m} a_{pj}\mathbf{k}_j^{(l)}) \in \mathbb{R}^{n \times n}$$

と表現される．

Newton 法の反復過程における連立一次方程式は，$h \to 0$ であれば，$J(\mathbf{k}_1^{(l}, \mathbf{k}_2^{(l)}, ..., \mathbf{k}_m^{(l)}) \to I_{mn}$ となることが期待される．従って，ある程度小さな h に対して IRK 法を実行する場合は，良条件の連立一次方程式を大量に解くことになり，多倍長精度が必要な常微分方程式であっても，この部分に DP-MP 型反復改良法を使用し，高速化が図れるのではないかと思われる．

そこで，良条件問題 (9) として生成した 128 次行列 $A(\|A\|_1 \approx 10^3)$ を用いた

$$\mathbf{f}(x, \mathbf{y}) = -A\mathbf{y},\ \mathbf{y}(0) = [1\ ...\ 1]^T$$

という定係数常微分方程式に対して 3 段 6 次 Gauss 型陰的 Runge-Kutta 法を適用し，$L = 50$ 桁計算した時の性能評価を行うことにする．**図 4** に，この時の，h に対する近似解の相対誤差と $J(\mathbf{k}_1^{(l)}, \mathbf{k}_2^{(l)}, ..., \mathbf{k}_m^{(l)})$ の Frobenius ノルム及び条件数の変化を示す．

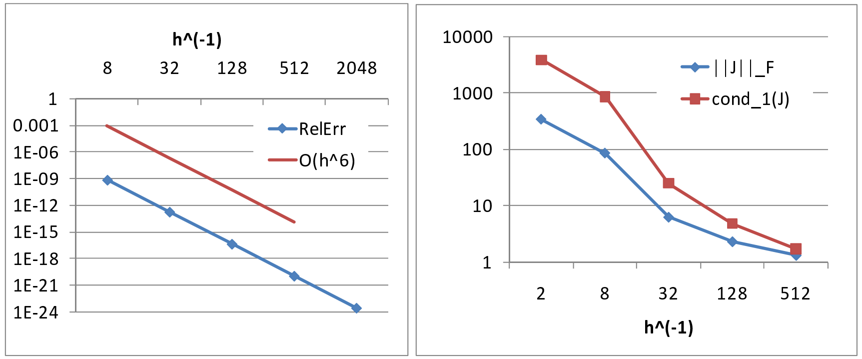

図 4 最大相対誤差の変化 (左), J の Frobenius ノルムと条件数の変化 (右)
Fig. 4 History of Maximum Relative Errors (Left), Frobenius Norms and Condition Numbers of J (Right)

$h = 512$ の時，約 24 桁の近似解が得られることと，条件数は 3.8×10^3 から 1.7 へと減少することが分かる．従って，DP-MP 型反復改良法は十分適用可能である．DP-MP 型反復改良法を使用した IRK 法の性能向上比 (対 直接法) を**図 5** に示す．

直接法を用いた時に比べ，GotoBLAS を使用した場合で約 7.5〜8.9 倍，BNCpack を使用した場合でも約 3.8〜4.8 倍の性能向上が得られた．

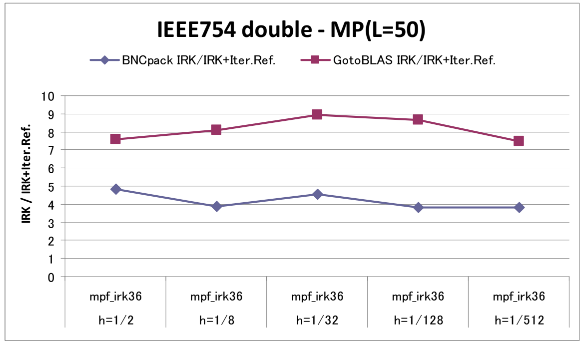

図 5 陰的 Runge-Kutta 法の速度向上率 (10 進 50 桁): DP-MP 型反復改良法 対 直接法
Fig. 5 Speedup ratio of Imlicit Runge-Kutta Method (50 Decimal Precision): DP-MP vs Direct Method

最後に，計算時間と精度の関係を荒っぽく眺めてみる．図 4 から分かるように，3 段 6 次 ($m = 3$) において $h = 512^{-1}$ における成分ごとの最大相対誤差は約 10^{-20} であったが，これを基準として各段数とステップ幅，計算時間をまとめてみると**表 3** のようになる． 明らかに，段数 (次数) が高い方が計算時間が短くなっていることが分かる．従って，要求精度に応じて最短の計算時間になるような段数の公式を選ぶような最適化技法の開発が今後の重要な課題となろう．

表 3 最大相対誤差 対 計算時間 ($L = 50$, GotoBLAS 使用時)
Table 3 Maximum Relative Error vs. Computational Time(sec) ($L = 50$ with GotoBLAS)

m	h^{-1}	max RelErr	Comp.Time (sec)
3	512	1.1×10^{-20}	6583
4	128	1.1×10^{-23}	2211
5	32	1.8×10^{-24}	849
10	2	2.1×10^{-30}	210

5. 結論と今後の課題

以上の数値実験により，混合精度反復改良法は，DP-MP,MP-MP 型のいずれにおいても，収束条件を満足するときにはほぼ L 桁直接法と同程度の精度の近似解が得られ，GotoBLAS のような高性能な IEEE754 単精度・倍精度線型計算ライブラリを用いることで DP-MP 型においても直接法に比べて高速な計算が可能なこと，陰的 Runge-Kutta 法の内部計算においても有用に働く可能性があることが示された．

今後の課題は，一般の常微分方程式の初期値問題に対して混合精度反復改良法がどの程度適用可能かどうかを調査し，適用可能な問題に対してはどの程度の性能向上が図れるのかを数値実験によって調べることである．応用上重要な，時間発展の偏微分方程式を線の方法によって離散化して得られた常微分方程式にも適用可能なようにするためには，疎行列問題にも適用可能なように，直接法だけでなく Krylov 部分空間法などの反復解法を用いた混合精度反復改良法が望ましい．既に Buttari らは SP-DP 型の反復改良法が Krylov 部分空間法に対して有効に働くことが多いことを示しており [3]，DP-MP, MP-MP 型に対しても同様の有効性があるものと期待される．陰的 Runge-Kutta 法の Newton 法の内部反復を高速化する手法は幾つか提案されており，それとの併用によって更なる高速化が可能になる問題も存在すると思われる．

参 考 文 献

1) Swox AB. GNU MP. `http://gmplib.org/`.
2) A.Buttari, J.Dogarra, Julie Langou, Julien Langou, P.Luszczek, and J.Karzak. Mixed precision iterative refinement techniques for the solution of dense linear system. *The International Journal of High Performance Computing Applications*, Vol.21, No.4, pp. 457–466, 2007.
3) A.Buttari, J.Dongarra, J.Kurzakand P.Luszczek, and S.Tomov. Using mixed precision for sparse matrix computations to enhance the performance while achieving 64-bit accuracy. *ACM Trans. Math. Softw.*, Vol.34, No.4, pp. 1–22, 2008.
4) S.P.Nørsett E.Hairer and G.Wanner. *Solving Ordinary Differential Equations*. Springer-Verlarg, 1996.
5) Tomonori Kouya. BNCpack. `http://na-inet.jp/na/bnc/`.
6) 幸谷智紀. 実用的な古典的誤差評価法の提案と gauss 型積分公式の分点計算への応用について. 情報処理学会論文誌, Vol.48, No. SIG18(ACS20), pp. 1 – 11, 2007.
7) Julie Langou, Julien Langou, Piotr Luszczek, Jakub Kurzak, Alfredo Buttari, and Jack J. Dongarra. Exploiting the performance of 32 bit floating point arithmetic in obtaining 64 bit accuracy (revisiting iterative refinement for linear systems). Technical Report 175, LAPACK Working Note, June 2006.
8) LAPACK. `http://www.netlib.org/lapack/`.
9) MPFR Project. The MPFR library. `http://www.mpfr.org/`.
10) ATLAS: Automatically Tuned Linear Algebra Software. `http://math-atlas.sourceforge.net/`.