ハイパフォーマンス
コンピューティング　93－18
（２００３．３．１２）

計算精度を考慮したGMRES法

神谷和憲[†]　黒田久泰[††]　金田康正[††]

連立一次方程式の反復解法の問題点の一つは行列によっては実用的な反復回数以内に収束に至らない点である．その原因としては行列の特性が挙げられるが，途中の行列ベクトル演算の計算精度も収束に影響が与えている可能性がある．本研究ではGMRES法の収束特性が行列ベクトル演算の計算精度によりどのように影響をうけるか実験的に評価した．計算精度は積和演算における情報落ち誤差を補償する方法，浮動小数点数の精度を設定する方法を用いた．実験からは計算精度の改善によって収束特性が向上する例があることが分かった．

GMRES method considering the calculation accuracy

Kazunori Kamiya,[†] Hisayasu Kuroda[††] and Yasumasa Kanada[††]

One of the problems of the iterative solvers for linear systems is that some matrices hardly converge within practical iteration times. This depends on the characteristics of the matrix but it is probable that it also depends on the calculation accuracy. In this paper, we evaluated the convergence characteristics of GMRES method depending on the calculation accuracy. We defined calculation accuracy as the compensation of sum of products and the precision of floating point. From the experiment, we can find the case that the improvements of calculation accuracy leaded to better convergence characteristics.

1. はじめに

大規模連立一次方程式の解法は流体計算，分子軌道の計算などさまざまな分野で必要で需要が高く，これらは反復解法を用いて解かれることが一般的である．連立一次方程式の解法にはガウスの消去法をもとにした直接解法もあるが，計算過程のメモリ使用量の点で反復解法に利がある．また，反復解法は収束までの反復回数が少ない場合は直接解法に比べ，はるかに少ない計算量で結果を得られる利点もある．

反復解法の大きな問題点は行列によっては反復の過程で実用的な反復回数中に収束に至らないということである．その原因は行列の特性 (固有値の分布，固有ベクトルの向き，初期残差) によるところが大きいが，途中の行列ベクトル演算の計算精度が影響を与えている可能性がある．

本論文は非エルミート行列 (実数の範囲では非対称行列) に対して最も有用な方法の一つであるGMRES法を研究対象とする．GMRES法はクリロフ部分空間法の一つで演算カーネルに直交化操作を含んでいる．直交化操作は積和演算の結果を再び演算に用いるので丸め誤差が蓄積しやすい．

そこで本論文では行列ベクトル演算を浮動小数点数の精度を変化させる方法 (単精度，倍精度，4倍精度，多倍長)，積和演算の誤差を小さくする方法 (誤差補償のアルゴリズム，絶対値の小さい順序で和をとる方法) れについて実験した結果を報告する．

2. GMRES法

GMRES法は1986年にSaad[1)] らによって提案された手法で，非エルミート行列に対する連立一次方程式の解法として有力な手法である．ここではGMRES法の理論収束特性と実装について簡単に説明する．

2.1 GMRES法

連立一次方程式 $Ax = b$ をGMRES法を用いて解くとき，まずアーノルディ原理を用いて初期残差 $||r_0||$ から得られるクリロフ部分空間上に正規直交基底 $V_n = \{v_1, v_2, \ldots, v_n\}$ を生成する．ここで n 回目の反復残差 r_n は以下の最小条件から求まる．ただし x_0 を反復の初期値とする．

$$r_n = \min_{z \in K_n} ||b - A(x_0+z)|| = \min_{z \in K_n} ||r_0 - Az|| \quad (1)$$

このとき，n 回目の反復解は $x_n = x_0 + z = x_0 + V_n y$ とあらわすことができ，これは y についての最小2乗問題とみなすことができる．

† 東京大学新領域創成科学研究科
the University of Tokyo, department of frontier informatics
†† 東京大学情報基盤センター
the University of Tokyo, Information Technology Center

$$J_n(y) = ||||r_0||v_1 - AV_n y|||| \quad (2)$$
$$= ||||r_0||e_1 - \bar{H}_n y)|| \quad (3)$$

この問題は行列 $\bar{H}$ を QR 分解することで解くことができる☆．ところで GMRES 法は毎回の反復ごとにメモリ上に保持する直交ベクトルの (正規直交基底のベクトル) 数が増えていき，計算機での実装を考えるとそれは実用的ではない．これはメモリ使用量と計算量の増大を招くので k 回目までの反復の後，リスタートするようにする GMRES(k) 法が一般的である．k として 30 から 100 程度の値を選ぶことが多い．

2.2 GMRES 法の収束について

GMRES 法においては毎回の反復において， 理論的には残差 r_n が必ず減少することが知られている．また N 次元の方程式について解くためには多くとも N 回の反復で収束にいたる，ことも示されている[1)]．ただし大規模な次元になると N 回の反復を待つことなくより速い収束が望まれる．方程式 $Ax = b$ を GMRES 法で解くとき，その反復に至る収束の速さは A の固有値，固有ベクトル，初期残差ベクトル b によっている [2)]．A が正規 (A の固有ベクトルが正規直交系をなす) または正規に近い行列であり，固有値が原点から離れたある点付近の密集している場合は高速に収束することが知られている．逆に固有値が原点を中心に存在すると収束性が悪化することも分かっている[3)]．

3. 計算精度の補償

3.1 条 件 数

行列 A に対して条件数 $\kappa(A)$ は式 (4) で定義される値である．

$$\kappa(A) = ||A|| \cdot ||A^{-1}|| \quad (4)$$

A が n 個の異なる固有値を持つ時，条件数は 2 ノルムで考えると以下のように固有値の最大値と最小値の比で表される値となる．

$$\kappa(A) = \frac{\max_i |\lambda_i|}{\min_i |\lambda_i|} \quad (5)$$

条件数は A を用いた行列ベクトル演算の安定性を示す値である．条件数が大きいと演算過程に小さな摂動が生じたときに非常に大きな値の乱れとなってしまう．このように，わずかな摂動に対して出力結果が大きく異なってしまう場合のことを悪条件である，という．例えば，連立一次方程式 $Ax = b$ において，ある近似解 $\tilde{x}$ の残差 $r = ||A\tilde{x} - b||$ が十分小さな値になっても A が悪条件である (A の条件数が高い) 場合は近似解 $\tilde{x}$ の精度は保証されないことが知られている．行列ベクトル演算を計算機を用いて実装するとき，A が悪条件であると丸め誤差の影響で演算結果が大きく異なってしまう可能性がある．

☆ 実用上は Givens の回転行列を用いて簡単に上三角行列に変形できる．

3.2 浮動小数点の計算精度

計算機上では実数は浮動小数点で表されている．浮動小数点数では実数を限られたビット数で近似してあらわすため，実数との誤差が生じる．

浮動小数点数同士の和は 2 数の絶対値の差が大きい時に下位のビットが丸められてしまい，精度が劣化する．また浮動小数点数同士の積はその演算ごとに丸めの操作が起こるので結果の精度が劣化する．

行列ベクトル演算で用いる浮動小数点の積和演算の計算精度をできるだけ劣化させないために 2 つのアプローチが考えられる．1 つは積和演算における和の情報落ちを防ぐ方法である．具体的には (1) 積和計算の和を 2 つの変数によって補償をする方法，(2) 積和演算における情報落ちを計算順序の変更によって軽減する方法の 2 つを考える．

もう 1 つのアプローチは浮動小数点数の変数に割り当てる bit 数を操作して浮動小数点が表現できる実数の範囲を増やし，積和演算の積の精度を補償する方法である．

以下ではそのそれぞれについて説明する．

3.3 積和演算の精度を補償する方法

3.3.1 積和演算の和を 2 つの変数によって補償する方法[4)]

積和計算 $S = a_1b_1 + a_2b_2 + \ldots + a_Nb_N = \Sigma_{i=1}^{N} a_ib_i$ の計算は以下のアルゴリズムによって情報落ちによる誤差を補償できることが知られている．浮動小数点の和を求める時に，2 数の差が大きいと下位のビットが欠落してしまうが，求める和の変数と欠落部分の和を求める変数の 2 変数を用いることで浮動小数点数を連続して和をとるときに精度を補償することができる．この手法は欠落部分を保持するための変数分のメモリと，欠落部分を求める加減算を加えるだけなので実装も容易で，演算量，メモリともにオーバーヘッドは小さいといえる．

補償付積和計算

```
sum = a[0] * b[0]; cmp = 0.0;
for(i = 1; i < N; i++){
    tmp1 = a[i] * b[i] - cmp;
    tmp2 = sum + tmp1;
    cmp = (tmp2 - sum) - tmp1;
    sum = tmp1;
}
```

3.3.2 絶対値の昇順に加算する誤差改善

積和計算において情報落ちの影響を小さくする方法として考えられる別の方法は和の計算を絶対値の小さい順に加算していくという方法である．絶対値の小さい順から足していくと 2 数の差が大きくなるという場合が少なくなるので極力情報落ちを防ぐことができる．難点は積和計算をする度に要素をソートしなくて

はならない，ということである．これは要素数が大きくなると大きなボトルネックとなる可能性がある．

3.4 実数の表現 bit 数を変更する方法

3.4.1 単精度と倍精度

現在の一般的なプロセッサのアーキテクチャは浮動小数点演算は倍精度 (64bit) 計算ができるように設計されている．また C 言語など一般的なプログラミング下のコンパイラでは浮動小数点は単精度 (32bit)， 倍精度の変数設定ができるようになっている．単精度を用いることは計算性能の点では倍精度の場合とあまり変わりがないがメモリ使用量が半分で済む点では意味がある．

3.4.2 4 倍長演算を用いる

倍精度型浮動小数点の積和乗算において計算結果を正しく格納するためには 4 倍精度必要である．そこでこの丸めの影響を軽減するために 4 倍精度の型の浮動小数点を用いる方法が考えられる．C 言語では long double 型として 128bit の浮動小数点数を格納できる型が用意されていることもある．4 倍精度の浮動小数点を使うことは倍精度の 2 倍のメモリ領域が必要になる．また 4 倍精度の浮動小数点演算は内部的には倍精度の演算でエミュレートされていることが多く，計算量の増加も見込まれる．

3.4.3 多倍長演算を用いる

浮動小数点の丸め誤差を 4 倍精度以上で確保するためには別の実装が必要である．32 ビットの整数型配列を用いて実数を 2^{32} 進法で表すことにすると必要ビット数だけの整数型配列を用意することで，必要精度の浮動小数点の表現が可能となる．このような方法を多倍長で浮動小数点を表現する，という．多倍長によって計算精度を補償した場合，メモリ使用量は確保した精度 (整数型の配列長) 分必要になり，積の計算量は整数型配列の配列長が N であるとすると通常は N^2 だけ必要となるが $N \log N \log\log N$ に抑えることのできるアルゴリズムが存在する[5]．C 言語から利用できる多倍長演算用のライブラリ GMP が gnu から提供されている．

3.5 反復解法において計算精度の向上によって期待される点

- 収束回数の改善
  反復法において計算精度の影響で反復が正常に終了しない場合や，反復回数が増加する可能性がある．実際，理論的に N 回で反復する，とされる CG 法も実際には計算精度の影響で終了しないこともある．Bi-CG 法などの積型反復法においては 4 倍長演算を用いることで倍精度の実装で収束しなかった問題が収束することもあった，という報告がある[6]．また積和演算を補償を用いることで GPBiCG 法において倍精度では収束しなかった問題が収束したという報告もある[4]．
- パフォーマンスの改善
  計算精度を向上させても収束に与える影響が小さい場合は計算精度がある程度低くても反復解が収束に至るので計算パフォーマンスの改善が見込まれる．具体的には 4 倍長や多倍長を用いなくても倍精度で十分収束する場合は明らかに倍精度を用いたほうがパフォーマンスがよいと思われる．また，単精度を用いても収束する場合は並列環境においては通信時間が単精度を用いると通信量が半分になるのでパフォーマンスの向上が見込まれる．

4. 実　　験

4.1 実 験 手 法

GMRES 法を表 1 に示すような積和演算の計算精度を補償する手法と表 2 に示すような実数の表現精度を変更する方法で実装した．

表 1 積和演算の計算精度を確保する手法

手法名	(略記名)	内容
normal	(nrm)	特に計算精度対策のない手法
compensation	(cmp)	2 つの変数により和をを補償する手法
sorted	(srt)	絶対値の昇順に加算する手法

表 2 実数の表現手法

表現精度 (ビット数)	(略記名)	実現方法
単精度 (32bit)	(float)	C 言語 float 型
倍精度 (64bit)	(double)	C 言語 double 型
4 倍精度 (128bit)	(quadr.)	C 言語 long double 型
多倍長精度 (256bit, 512bit 1024bit, 2048bit)	(multi.)	GMP ライブラリ version 4.1

4.2 実 験 環 境

実験環境を表 3 に示す．使用マシンとして COMPAQ-GS80E を選択したのは実行性能が高いこと，搭載コンパイラが浮動小数点の 4 倍長演算をサポートしていることによる．またコンパイラオプションとして最適化による浮動小数点演算の順序変更を避けるために-nofp_reorder を指定した．

表 3 実験環境

使用マシン	COMPAQ GS80E
プロセッサ	Alpha21264 731MHz
搭載メモリ	2GB
使用 OS	True64 Unix
コンパイラ	Compaq CC
コンパイルオプション	-fast -nofp_reorder
実装言語	C 言語
実装方式	逐次

4.3 実 験 条 件

すべての数値実験は特に断らない場合は表 4 の条件で行った．行列によっては右辺ベクトル b の値によって意味をなすものがあるが，ここでは真の解を $x = (1,\ldots,1)^T$ と設定して計算することにした．また計算精度の反復解法の収束に与える影響を調べることが目的であるため前処理は使用しなかった．

表 4　実験条件

真の解	$x = (1, 1, \ldots, 1)^T$
初期解	$x_0 = (0, 0, \ldots, 0)^T$
リスタート周期	30
前処理	なし
収束条件	$\frac{\lVert r \rVert}{\lVert b \rVert} < 10^{-12}$
終了条件	$time.$ 指定回数反復後も収束に至らない場合☆ $div.$ 残差が発散した場合 ($r_i \leq r_{i+1}$)

以下の数値例での表中の値は各手法の収束までの反復回数 (回) と実行時間 (秒) を示している．

4.4　数値例 1: Toeplitz 行列

Toeplitz 行列は式 (6) で定義される行列である．一般的に γ の値が大きくなると反復法の収束が悪くなることが知られている．それは γ の値が大きくなるにつれ固有値が複素平面の左半面に位置し，原点を囲うようになっていくからである．

$$A = \begin{pmatrix} 2 & 1 & & & & \\ 0 & 2 & 1 & & & \\ \gamma & 0 & 2 & 1 & & \\ & \gamma & 0 & 2 & 1 & \\ & & \gamma & 0 & 2 & \ddots \\ & & & \ddots & \ddots & \ddots \end{pmatrix} \qquad (6)$$

200 次元の Toeplitz 行列に対し，$\gamma = 1.0, 1.75, 2.125$ と $\gamma = 2.43$ の場合に対して実験を行った．$\gamma = 2.125$ と $\gamma = 2.43$ の時の各手法の収束の特徴を図 1 に示す．横軸が反復回数，縦軸が反復残差を表す．$\gamma = 2.125$ の時は単精度以外のどの手法を用いても等しく収束に至ることが分かる．単精度を用いた場合は途中までは他の手法と同じ特性を持つが，残差が小さくなると停留してしまうことも分かる．$\gamma = 2.43$ の時はどの手法に用いても収束せず，また残差の特性も等しいことが分かる．200 次元で実験した時の詳しい結果は表 5 にまとめた．ただし $\gamma = 2.43$ の時はいずれも収束に至らなかったので表は省略した．

表 5　200 次元 Toeplitz 行列に対する結果

		$\gamma = 1$	$\gamma = 1.75$	$\gamma = 2.125$
float	nrm	81 0.003	$div.$	$div.$
	cmp	53 0.008	$div.$	$div.$
	srt	54 0.037	$div.$	$div.$
double	nrm	52 0.003	175 0.009	436 0.021
	cmp	52 0.008	175 0.029	436 0.072
	srt	52 0.046	175 0.240	436 0.630
quadr.	nrm	52 0.051	175 0.180	436 0.430
	cmp	52 0.090	175 0.330	436 0.800
	srt	52 0.210	175 1.010	436 2.460
multi.	256	52 0.240	175 0.930	436 2.230
	512	52 0.400	175 1.480	436 3.600
	1024	52 0.910	175 3.300	436 8.130
	2048	52 2.500	175 9.010	436 22.370

次に 262144 次元について $\gamma = 1.0, 1.75, 2.125$ について実験を行った．その結果を表 6 に示す．ここでは多倍長を用いる実験は使用メモリの限界から 256bit の場合のみで実験を行った．

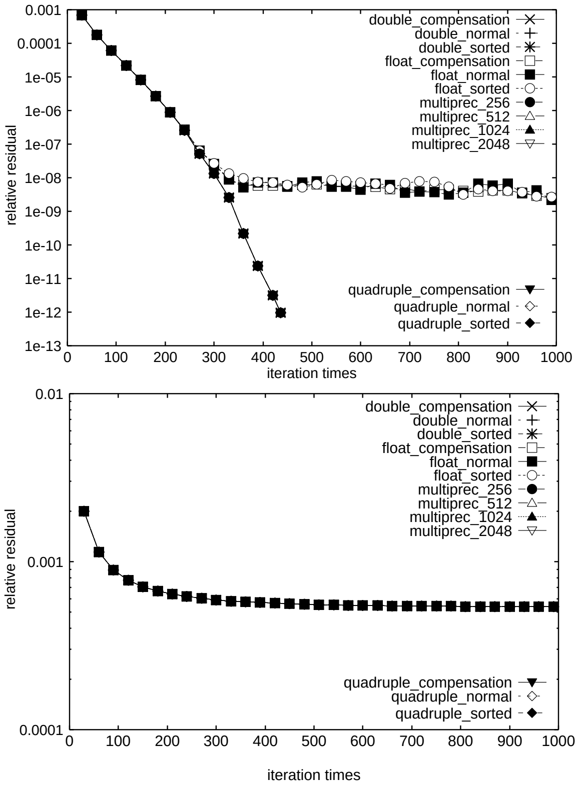

図 1　200 次元 Toeplitz 行列に対する各手法の収束特性
(上図 $\gamma = 2.125$, 下図 $\gamma = 2.43$)

表 6　262144 次元 Toeplitz 行列に対する結果

		r = 1	r = 1.75	r = 2.125
float	nrm	105 41.55	$div.$	$div.$
	cmp	45 21.92	$div.$	$div.$
	srt	$div.$	$div.$	$div.$
double	nrm	44 26.67	153 90.46	539 351.21
	cmp	44 27.28	153 104.53	540 370.02
	srt	44 44.27	153 175.99	539 582.87
quadr.	nrm	44 96.96	153 372.18	539 1359.29
	cmp	44 140.20	153 527.28	539 2713.31
	srt	44 167.04	153 643.72	539 2185.26
multi.	256	44 594.08	153 2340.79	539 11189.91

この表から，単精度以外を用いる場合はどの手法を用いても収束の特性は等しいことが分かる．単精度は発散して収束に至らないことが多いがそれは単精度で表現できる桁数が 10 進数で約 7 桁強であることによると考えられる．そこで単精度だけを用いて，収束条件を $\frac{\lVert r \rVert}{\lVert b \rVert} \leq 10^{-8}$ として 262144 次元で $\gamma = 2.125$ として実験を行った．その結果図 2 のような結果が得られた．　この実験から，単精度浮動小数点数を用いる

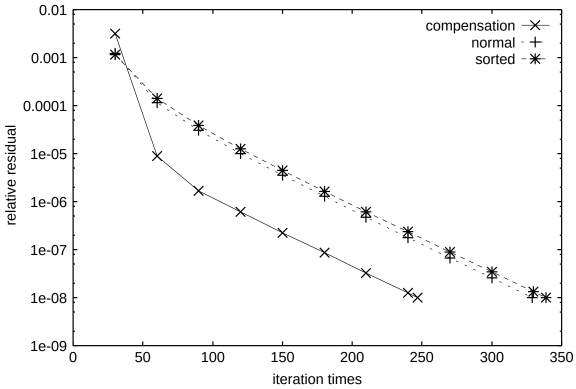

図 2　$\gamma = 2.125$ の Toeplitz 行列に対する収束特性 (262144 次元)

場合は積和演算の補償をすると収束までの反復回数が減少し，倍精度を用いた場合とほぼ同じ特性を示すという結果が得られた．また積和演算の補償をする方法は反復回数は減少させるが，実行時間は必ずしも減少するわけではない，という結果も得られた．そして絶対値の小さい順に加算をとる方法はソートのコストが大きくなる上に収束性の安定にあまり寄与しないことも分かった．

4.5　数値例 2: 偏微分方程式の 5 点差分から得られる差分方程式

偏微分方程式 (7) を 2 次元領域 $[0,1] \times [0,1]$ で 2 次精度の 5 点差分して得られる連立一次方程式の行列を考える．

$$\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} + D\frac{\partial f}{\partial x} = g(x,y) \qquad (7)$$

この行列は対称性が高いので正規性も高い．また固有値はすべて複素平面で右半面に密集し，反復解法による収束性がよい．128×128 の格子点によって 16384 次元の行列を得た．ここで $D = 64, 128, 256$ について実験を行った．その結果表 7 の結果が得られた．単精度はいずれの手法でも発散した．また倍精度以上のどの手法を用いても等しい反復回数で収束することが分かった．多倍長演算を用いた時の結果はここでは省略した．

表 7　5 点差分から得られる行列に対する結果

		D=64	D=128	D=256
float	nrm	*div.*	*div.*	*div.*
	cmp	*div.*	*div.*	*div.*
	srt	*div.*	*div.*	*div.*
double	nrm	827 9.30	762 8.64	804 9.05
	cmp	826 18.12	762 16.72	803 17.59
	srt	854 189.02	762 166.50	803 173.66
quadr.	nrm	807 88.36	757 82.88	802 86.91
	cmp	807 148.23	757 138.99	802 145.47
	srt	807 741.15	757 678.38	802 721.53

4.6　数値例 3: Frank 行列

N 次の Frank 行列 $F(N)$ は以下のように定義されるヘッセンベルグ行列である．

$$F(N) \equiv \begin{pmatrix} N & N-1 & N-2 & \ldots & 1 \\ N-1 & N-1 & N-2 & \ldots & 1 \\ & N-2 & N-2 & \ldots & 1 \\ & & \ddots & \ddots & \vdots \\ & & & 1 & 1 \end{pmatrix} \qquad (8)$$

Frank 行列の固有値はすべて異なる正の実数でそれぞれが逆数の関係になっている．つまり $\kappa(F) = \frac{\max_i |\lambda_i|}{\min_i |\lambda_i|} = \max_i \lambda_i^2$ となり条件数も大きくなり非常に悪条件の行列となる．

50, 100, 200 次元の Frank 行列に対してリスタート周期を 30 に対して実験した結果を表 8 に示す．またそのうち代表的な例であった 100 次元 Frank 行列に対する結果を図 3 に示す．この図からは単精度，倍精度，4 倍精度はそれぞれある点において残差が停留していることが分かる．これは行列が悪条件であるため，各反復操作で行う行列ベクトル積の結果の信頼性が損なわれたために起こる結果だと思われる．一方，多倍長演算を用いた場合は Frank 行列でも収束に至ることが分かった．これは多倍長演算によって行列ベクトル積の精度を補償した結果だと考えられる．

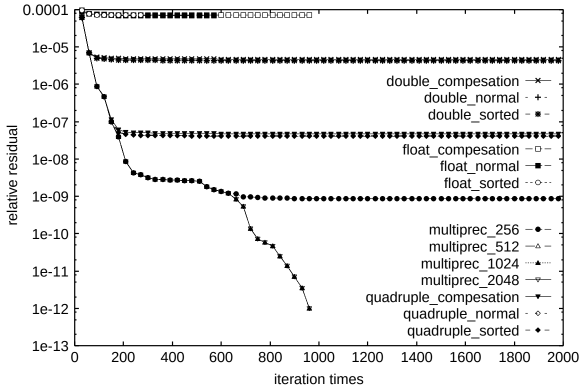

図 3　100 次元 Frank 行列に対する残差の特性 (リスタート周期 30)

表 8　Frank 行列に対する実験結果 (リスタート周期 30)

		50 次元	100 次元	200 次元
float	nrm	*div.*	*div.*	*div.*
	cmp	*div.*	*div.*	*div.*
	srt	*div.*	*div.*	*div.*
double	nrm	*div.*	*time.*	*time.*
	cmp	*div.*	*time.*	*time.*
	srt	*time.*	*time.*	*time.*
quadr.	nrm	28 0.014	*time.*	*time.*
	cmp	28 0.026	*time.*	*time.*
	srt	28 0.075	*time.*	*time.*
multi.	256	28 0.063	*time.*	*time.*
	512	28 0.102	960 9.53	*time.*
	1024	28 0.235	960 22.14	*time.*
	2048	28 0.669	960 64.03	*time.*

4.7 数値例 4: Glued Wilkinson W_g^+ 行列

Glued Wilkinson W_g^+ 行列は以下のように定義される[7)].

$$W_g^+ \equiv \begin{pmatrix} W_{21}^+ & \delta & & & \\ \delta & W_{21}^+ & \delta & & \\ & \delta & \ddots & \ddots & \\ & & \ddots & W_{21}^+ & \delta \\ & & & \delta & W_{21}^+ \end{pmatrix} \quad (9)$$

ただし $\delta = 1$ とする．ここで行列 W_{21}^+ は

$$W_{21}^+ \equiv \begin{pmatrix} 10 & 1 & & & & & \\ 1 & 9 & 1 & & & & \\ & 1 & \ddots & \ddots & & & \\ & & \ddots & 0 & \ddots & & \\ & & & \ddots & \ddots & 1 & \\ & & & & 1 & 9 & 1 \\ & & & & & 1 & 10 \end{pmatrix} \quad (10)$$

である．

Glued Wilkinson 行列は対称行列なので固有ベクトルは正規直交系を成す．しかし，固有値は正と負の実数からなるため収束性は良くないと考えられる．Glued Wilkinson 行列自体の条件数はそれほど大きい値ではないがこの行列を用いて直交化操作を行う時に生成される行列の条件数が非常に大きくなることが知られている．

420 次元の Glued Wilkinson 行列についてリスタート周期を $restart = 30, 50$ について実験を行った結果を表 9 に示す．またそのうち，特徴的な例を図 4 に示す．この結果から Glued Wilkinson 行列においては計算精度を補償することは各手法ごとに効果があることが分かる．注目すべきは計算順序を変更した場合に反復回数が大幅に削減されていることである．また浮動小数点の精度を上げるほど反復回数は少なくなっていくことも分かる．

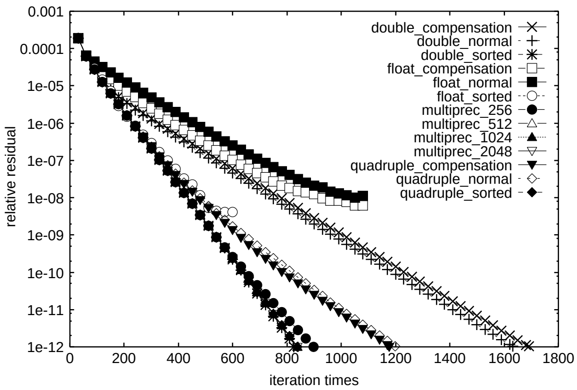

図 4 420 次元 Glued Wilkinson 行列に対する残差の特性 (リスタート周期 30)

表 9 Glued Wilkinson 行列に対する結果 (420 次元)

		restart=30	restart=50
float	nrm	*div.*	*div.*
	cmp.	*div.*	*div.*
	srt	*div.*	*div.*
double	nrm	1633 0.17	673 0.11
	cmp.	1691 0.65	656 0.38
	srt	825 2.58	217 1.11
quadr.	nrm	1199 2.68	244 0.80
	cmp.	1176 4.91	249 1.57
	srt	838 11.31	222 5.25
multi	256bit	897 9.58	218 3.29
	512bit	838 13.48	218 4.90
	1024bit	838 28.29	218 10.34
	2048bit	838 76.09	218 28.48

5. ま と め

GMRES 法の収束性に関して，行列の性質以外に，計算精度がどのような影響を与えるかを実験した．その結果，(1) 単精度を用いる時は積和演算の補償が有効なこと，(2) 悪条件な行列 (Frank 行列)，計算過程が悪条件な行列 (Glued Wilkinson 行列) には多倍長演算が有効なこと，(3) 計算順序を変更して積和演算を計算することが場合によっては有効なこと，(4) 行列の特性がよいときは計算精度を補償することの利点は少ないことなどが分かった．今後はより多くの行列に対して実験し，本研究で得た知見が一般的に成り立つことかどうかを確かめていきたい. 特に精度を補償する方法が条件数が高い行列に対してどの程度有効か実験していきたい.

参 考 文 献

1) Youcef Saad, M. H. S.: GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems, *SIAM J. Sci. Stat. Comput.*, pp. 856–869 (1986).
2) J.Dongarra, J.: *Numerical Linear Algebra for High-Performance Computers*, SIAM (1998).
3) Lloyd N. Trefethen, D. B. I.: *Numerical Linear Algebra*, SIAM (1997).
4) 藤野清次, 渡辺善隆, 南里豪志: 基本数値演算:内積の評価とその応用, 情報基盤センター年報第 2 号, 九州情報基盤センター (2002).
5) Knuth, D.: *The Art of Computer Programming*, Vol. 2: Seminumerical Algorithms.
6) 長谷川秀彦: 4 倍長演算における積型反復法の比較. 第 15 回 LA 研究会.
7) 片桐孝洋: 並列固有ベクトル計算における強制対角ブロック化の効果, *Swopp 2002* (2002).