演算精度をかえれば見えてくる線形方程式の世界

SOLVING A SYSTEM OF LINEAR EQUATIONS
WITH MORE ACCURATE ARITHMETIC OPERATIONS

長谷川秀彦[1], 小武守恒[2]
Hidehiko HASEGAWA and Hisashi KOTAKEMORI

[1]博士（工学）筑波大学教授　図書館情報メディア研究科（〒 305-8550 つくば市春日 1-2, hasegawa@slis.tsukuba.ac.jp）
[2]博士（理学）（株）TCAD インターナショナル（〒 101-0051 東京都千代田区神田神保町 1-44-2, hisashi.kotakemori@TCAD-international.com）

In this paper, we show that the SSE2 tuned double-double precision operation consumes only five times of double precision operations for solving a system of linear equations. We also show that iterative methods do not converge for a given problem when its computation is performed with insufficient precision. This means that enough precision make its convergence more smooth. For ease of chice of methods, solvers and preconditioners, and computing precision, we provide "Lis": a library for a system of linear erqations and "Lis-test": its GUI on Windows PC.

Key Words : *double-double precision, quadruple operation, iterative methods, preconditioners*

1. はじめに

現在，数値シミュレーションの世界では IEEE754 に準拠した倍精度演算が標準となっている．いっぽうコンピュータの大規模化・高速化も著しく，単一 CPU での高速化は限界に近づいており，並列化が時代の趨勢となっている．このとき，高速化のネックとなるのはデータ転送であり，いくら CPU が高速に処理できても，メモリとのデータ転送や複数 CPU 間のデータ通信で性能が押さえられてしまう．

これまで，ハードウェア機能として用意されていない 4 倍精度演算や多倍長演算は，実行速度が遅く日常的に使える状況にはなかった．しかし，4 倍精度演算の実現に倍精度演算の 20 倍の演算量が必要であったとしても，データ転送量は 2 倍にしかならず，データ転送が CPU の処理能力に追いつかない計算環境では検討の余地がある．特に，大規模分散並列環境で効果的になる可能性がある．

われわれは分散並列環境を強く意識し，double-double 精度の 4 倍精度演算を線形反復法ライブラリ Lis (a Library of Iterative Solvers for linear systems) に実装した．Lis は前処理付きの Krylov 部分空間法のライブラリで，内部演算の合理化， SSE2 命令を用いた高速化によって，反復法向けの 4 倍精度演算を倍精度演算の 4.5 倍で実行できる．

Krylov 部分空間法は，丸め誤差　(rounding error) に敏感なアルゴリズムであり，高精度演算の導入は反復回数の減少につながる．

本報告では，演算の高精度化が Krylov 部分空間法にどのような影響を与えるのかという実例と， 線形反復法ライブラリ Lis にどのように高速な 4 倍精度演算が実装されているかを示す．

2. Krylov 部分空間法の演算精度依存性

(1) 実験のあらまし

まず，演算精度によって解法がどのようなふるまいを示すかを観察する [1]．実験には BiCG，CGS, BiCGSTAB[2], GPBiCG[3] を取りあげ，問題パラメータと演算精度を変えて収束の変化を調べる．

Omni OpenMP (Fortran) コンパイラ [4] に用意されている MULTIPLE PRECISION 型の変数に対して，コンパイル時に仮数部 (Mantissa) のビット数を指定することで同一のプログラムの演算精度を変えて実行できる．ここでは，一般の倍精度浮動小数演算相当の 53 ビット，4 倍精度相当の 100 ビット， 200 ビット， 300 ビットと 4 種類を扱う．なお MUPLTIPLE PRECISION 型の制限により仮数部は 53 ビット以上となるので，倍精度演算と同一にはならないが，演算精度依存性をみるのにはじゅうぶんである．

解きにくさを簡単にコントロールできる問題として Toepliz 行列

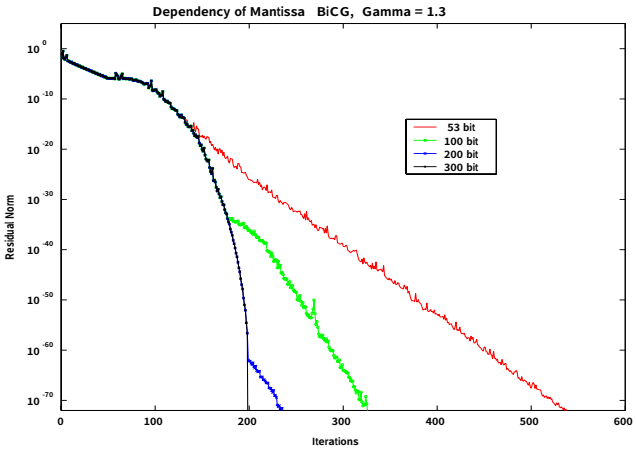
$$A = \begin{pmatrix} 2 & 1 & & & & \\ 0 & 2 & 1 & & & \\ \gamma & 0 & 2 & 1 & & \\ & \ddots & \ddots & \ddots & \ddots & \\ & & \gamma & 0 & 2 & 1 \\ & & & \gamma & 0 & 2 \end{pmatrix}$$

を用いた．行列 A の次数は $n = 200$ で，右辺ベクトルは $b_i = 1$, 収束判定は相対残差ノルムで 10^{-12} とした．γ は 1.3, 1.7, 2.1, 2.5 の 4 通りである．条件数は 6.4 ($\gamma = 1.3$) から 700 程度で ($\gamma = 2.5$) ，大きくはないが虚数軸方向に固有値が分布するため，これらの解法にとっては収束しにくい問題と言われている．

(2) 実験結果

いずれの解法でも，数学的には 200 回以内の反復で近似解に到達するはずである．

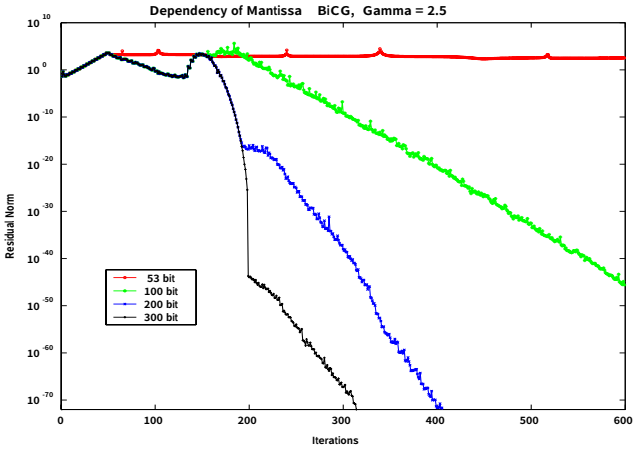
BiCG 法 $\gamma = 1.3$ （図－１）では，途中まで４本のグラフが重なって，どの精度でも正しい結果が得られていることがわかる．いっぽう，仮数部 53 ビットの相対残差ノルム 10^{-15} 以下は見かけ上の収束で，正しい収束ではない．この問題では，100 ビットで 10^{-35} まで， 200 ビットで 10^{-60} まで正しく収束する．BiCG 法の反復過程に現れるスカラー量 α, β は 10^{-5} から 10^{5} の範囲に分布した．



図–1 BiCG 法 $\gamma = 1.3$

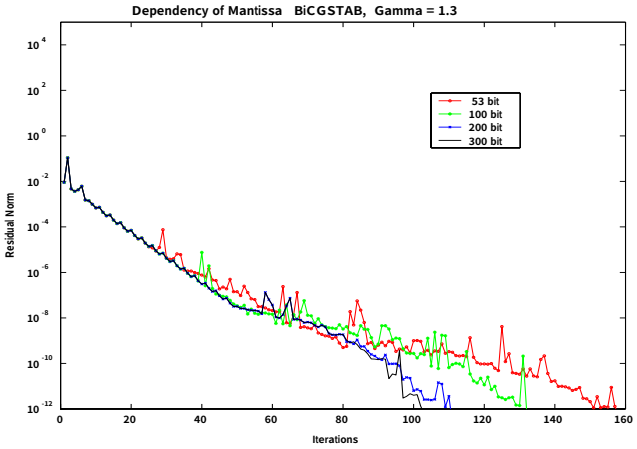
$\gamma = 2.5$（図－２）では，仮数部 53 ビット， 100 ビットは収束せず， 200 ビットで 10^{-15} まで， 300 ビットで 10^{-40} まで収束する．このときにも，少数の例外はあるが，BiCG 法の反復過程に現れる α, β は $\gamma = 1.3$ の場合と大差なかった．したがって，一部の計算結果が表現できる桁数をオーバーして精度を失ったことが主要な原因ではないといえる．

CGS 法はより多くのビット数が必要だったが，BiCG 法と同様である．



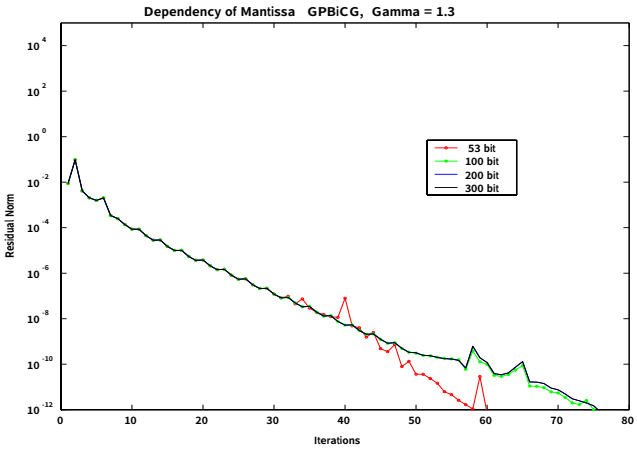
図–2 BiCG 法 $\gamma = 2.5$

BiCGSTAB 法 $\gamma = 1.3$（図－３）のとき，仮数部 53 ビットは約 160 回で収束する．仮数部を増やすと 10^{-6} あたりからグラフは分離してより少ない反復回数で収束する．この結果は BiCGSTAB 法が演算精度に敏感であることを意味する．$\gamma = 2.5$ のときは 1500 ビットで 210 反復となり，事実上，収束しなかった．



図–3 BiCGSTAB 法 $\gamma = 1.3$

GPBiCG 法 [3] $\gamma = 1.3$ （図－４）のとき，仮数部が 100 ビット以上よりも 53 ビットが良い収束を示す．53 ビットのグラフだけが分離していることから，誤差によって収束が加速されたことを表している．$\gamma = 2.5$ のときは 300 ビットで 320 反復だった．



図–4 GPBiCG 法 $\gamma = 1.3$

これらから，(1) 演算精度が不十分だと収束しないこと，(2) 必要な演算精度は問題によって異なること，(3) 必要な演算精度は解法に強く依存することなどがわかる．丸め誤差の影響は収束を加速する方向に作用することもあり，単純ではない．

3. 線形反復法ライブラリ：Lis

(1) Lis の概要

Lis (a Library of Iterative Solvers for linear systems) [5] は大規模疎行列を係数とする線型方程式

$$Ax = b$$

に対する反復法ライブラリで，C 言語と Fortran 90 で記述されたオープンソースのライブラリである．逐次版，OpenMP 共有メモリ並列版，MPI 単独，あるいは OpenMP ＋ MPI のハイブリッド分散メモリ並列版がある．

Lis の特徴は， 20 通りの反復解法， 10 通りの前処理， 11 通りの疎行列格納形式，2 種類の演算精度を簡単に組み合わせて使えることにある．実際，逐次版と並列版は共通のインタフェースであり，逐次環境から並列環境へはプログラム変更なし，あるいはわずかの手間で移行できる．

反復解法は表 1 に示す通り，定常（SOR 等），非定常（GPBiCG 等）など，一般の実行列に対する 20 種類である．前処理は，よく知られているスケーリング，不完全 LU 分解などに加えて，定常反復解法に有効な $I+S$ 型，smoothed aggregation に基づく代数的マルチグリッド SA-AMG，SOR などの反復法を用いる Hybrid 法，A-直交化に基づき逆行列 A^{-1} そのものを近似する近似逆行列 SAINV，従来の ILU よりも安定な分解ができる Crout 版 ILU 前処理など，表 2 の 10 通りである．行列データの格納形式は CRS (Compressed Row Storage), CCS (Compressed Column Sstorage), BSR (Block Sparse Row) など，表 3 に示した 11 種類である．行列データの格納形式は，解法の数学的性質とは関係ないが，実行時間には強く影響する．

表–1 反復解法

CG	CR
BiCG	BiCR
CGS	CRS
BiCGSTAB	BiCRSTAB
GPBiCG	GPBiCR
BiCGSafe	BiCRSafe
BiCGSTAB(l)	TFQMR
Jacobi	Orthomin(m)
Gauss-Seidel	GMRES(m)
SOR	FGMRES(m)

表–2 前処理

Jacobi
SSOR
ILU(k)
ILUT
Crout ILU
I+S
SA-AMG
Hybrid
SAINV
additive schwarz
ユーザ定義

表–3 格納形式

Compressed Row Storage	(CRS)
Compressed Column Storage	(CCS)
Modified Compressed Sparse Row	(MSR)
Diagonal	(DIA)
Ellpack-Itpack generalized diagonal	(ELL)
Jagged Diagonal	(JDS)
Block Sparse Row	(BSR)
Block Sparse Column	(BSC)
Variable Block Row	(VBR)
Dense	(DNS)
Coordinate	(COO)

(2) Double-Double 精度の 4 倍精度演算

Lis では，倍精度浮動小数点数を 2 個用いた”double-double”精度 [6],[7] を使用した 4 倍精度演算を実装した．double-double 精度浮動小数 a を $a = a.hi + a.lo$, $\frac{1}{2}\mathrm{ulp}(a.hi) \geq |a.lo|$ （上位 $a.hi$ と下位 $a.lo$ は倍精度浮動小数）として表現し， Dekker[8] と Knuth[9] のアルゴリズムに基づいて倍精度の四則演算の組み合わせで実現している．double-double 精度は FORTRAN の `REAL*16` よりも高速である [10]．しかし，FORTRAN の`REAL*16` の表現形式では仮数部が 112 ビットあるのに対して，倍精度浮動小数を 2 個利用しているため仮数部が 104 ビットと 8 ビット少なくなっている．また，指数部は倍精度浮動小数と同じ 11 ビットである．

Lis では，ライブラリの入力として与えられる係数行列 A，右辺 b，初期値 x_0，解の出力は倍精度になっている．そのため，反復解法内の変数を double-double 精度を用いて 4 倍精度化し， 4 倍精度変数同士， 4 倍精度と倍精度変数の混合演算を実装した． 4 倍精度変数はライブラリ内部で使われるだけなので，ユーザインタフェースに変更はなく，利用者は 4 倍精度演算のオプションを指定するだけである．倍精度の桁数の約 2 倍であること，内部のみで使われ，インターフェースに影響を及ぼさないことから，仮数部のビット数が標準仕様と異なっていても問題はない．また，Intel CPU に対しては SSE2 命令を用いた高速化をおこない， double-double 精度を用いた Lis の 4 倍精度演算は倍精度の約 4.5 倍の計算時間と，非常に高速に実行できる [5].

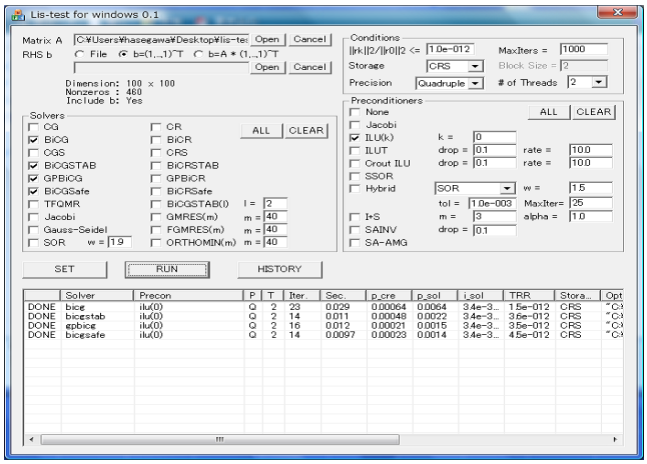
(3) 解法選択と実行時間予測

ソルバの数学的な性能は，問題（係数行列，右辺，初期値），（解法，前処理），演算精度に依存する．反復回数は計算環境には依存しないが，実行時間は疎行列格納形式，実装方法，並列化方法，最適化法，ハードウェアやソフトウェアなどの計算環境に依存する．反復回数で評価するか，実行時間で評価するかによって，最適な解法は異なってくる．

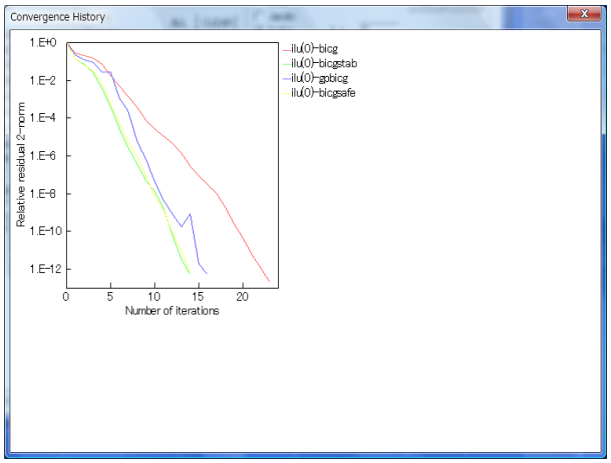
実際にシミュレーションでは，完全に異なるタイプの問題を解くことは稀で，ある程度の共通性を持ったタイプの問題を解くことが多い．反復法（解法，前処理）の収束に問題のどのような要素が有効かは明らかになっておらず，現時点では解法選択が適切だったか

どうかは問題を解いてみてはじめてわかる．

そこで， Lis の機能をわかりやすく提示することと，ユーザのプログラム作成に役立てることを目的として Windows 上で動作する Lis のテストプログラム Lis-test を提供している [11]．問題データを MatrixMarket[12] 形式に準じたテキストファイルで与えれば，Windows パソコン上で反復解法，前処理，演算精度を変化させて収束を調べることができる．OpenMP による並列化もおこなわおこなわれているため，パソコン上で並列化の効果をみることもできる．Lis-test を用いて代表的な問題を解けば，適切な反復解法，前処理がわかる．その問題が倍精度演算で十分なのか，それとも４倍精度演算が必要かどうかもわかる．ソルバの必要条件がわかったなら，その手法を実装すればいいし，インターネットから入手してもよい．もちろん Lis をダウンロードして使ってもよい．なお， Lis-test の実行に必要なのは二つのファイルのみで，インストールも不要なため，USB メモリに入れておいて実行することもできる．



図–5　Lis-test の選択メニュー



図–6　Lis-test が表示する収束履歴

4.　まとめ

倍精度演算では収束しなかった問題が高精度演算では容易に収束することがある．コンピュータの大規模化・高速化によって，演算性能やメモリ容量に余裕が生まれ，倍精度演算の高々５倍で double-double 精度を用いた４倍精度演算が実行できる．４倍精度演算が現実的になったことをふまえると，解法選択は倍精度の世界に限定せずに考える必要がある．

今後は，複数の演算精度を使い分ける解法，解法選択や精度選択の自動化なども視野にいれたうえで，より多くの現実的な実験データを集積し、より現実的な比較・評価・検討をおこなう必要がある．このような解析の手軽なツールとして Lis と Lis-test が役立つことであろう．

謝辞:　Lis と Lis-test は，科学技術振興機構 (JST) 戦略的創造研究推進事業 (CREST) 「大規模シミュレーション向け基盤ソフトウェアの開発」プロジェクト (http://ssi.is.s.u-tokyo.ac.jp/) の成果物である．

参考文献

1) 長谷川秀彦 :クリロフ部分空間法の計算精度依存性, 日本応用数理学会 2003 年度年会講演予稿集, pp. 316-317, 2003.
2) R. Barrett, et al. :Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, SIAM, 1994.
3) S.-L. Zhang :GPBi-CG: Generalized product-type methods based on Bi-CG for solving nonsymmetric linear systems, *SIAM Journal on Scientific Computing*, Vol. 18, No. 2, pp. 537-551, 1997.
4) Omni OpenMP Compiler Project, http://phase.hpcc.jp/Omni/
5) 小武守恒, ほか :SSE2 を用いた反復解法ライブラリ Lis 4 倍精度版の高速化, 情報処理学会研究報告, 2006-HPC-108, pp. 7-12, 2006.
6) D. H. Bailey :A fortran-90 double-double library, http://www.nersc.gov/~dhbailey/mpdist/mpdist.html
7) Y. Hida, X. S. Li, and D. H. Bailey :Algorithms for quad-double precision floating point arithmetic, *Proceedings of the 15th Symposium on Computer Arithmetic*, pp. 155-162, 2001.
8) T. Dekker :A floating-point technique for extending the available precision, *Numerische Mathematik*, vol. 18 pp. 224-242, 1971.
9) D. E. Knuth :The Art of Computer Programming: Seminumerical Algorithms, vol. 2, Addison-Wesley, 1969.
10) D. H. Bailey :High-precision arithmetic in scientific computation, *SC06 Technical Paper*, 2006.
11) Lis. http://ssi.is.s.u-tokyo.ac.jp/lis/.
12) Matrix Market. http://math.nist.gov/MatrixMarket.