高精度特異値計算ルーチンの開発とその性能評価

高田雅美[†,††] 岩崎雅史[†,††]
木村欣司[†††,☆] 中村佳正[††,†]

高精度かつ高速に行列を特異値分解するために，我々は，dLV（離散ロトカ・ボルテラ：discrete Lotka-Volterra）系による新たな特異値分解ライブラリを開発している．この一環として，本論文では，特異値のみを計算する *mdLVs*（原点シフト付き modified dLV）法の実装ならびに性能評価を行う．既存ルーチンとしては，線形数値計算ライブラリ LAPACK における *DBDSQR* と *DLASQ* がある．*DBDSQR* は，QRs 法に基づいた特異値分解ルーチンであるが，大規模特異値計算において計算速度，精度がともに十分でない．一方，*DLASQ* は，dqds 法に基づいた高速高精度な特異値計算ルーチンであるが，収束性が証明されていない．そのため，現在利用されている計算方法は，QRs 法である．これらに比べ，*mdLVs* 法では，収束性が証明されており，*DLASQ* より速度では劣るものの，理論的には，同等以上の高精度性を持つ．本論文では，*mdLVs* 法を適用するために 4 種類の実装手法を提案する．これらの手法の有効性を調べるために，複数の CPU を用いて実行時間と精度の比較実験を行う．その結果，*mdLVs* 法の理論どおりの高精度な性能を引き出す実装に成功した．

Implementation and Its Evaluations of Routine for Computing Singular Values with High Accuracy

Masami Takata,[†,††] Masashi Iwasaki,[†,††] Kinji Kimura[†††,☆] and Yoshimasa Nakamura[††,†]

To perform singular value decomposition of matrices with high accuracy and high-speed, we develop a library by using the dLV (discrete Lotka-Volterra) system. In this paper, as a part of its development, we implement and evaluate the *mdLVs* (modified dLV with shift) algorithm for computing singular values only. The well-known routines for singular values are *DBDSQR* and *DLASQ* provided in LAPACK (Linear Algebra PACKage). Calculation of singular values using *DBDSQR*, which is based on the QRs (QR with shift) algorithm, is slow in speed and has low accuracy. *DLASQ* is based on the dqds (differential quotient-difference with shift) algorithm having high-speed and high accuracy. However, to the best of our knowledge, convergence to singular values has not been proved. Therefore, the most popular method is the QRs algorithm. In the *mdLVs* algorithm, convergence is guaranteed. Though the computational time of the *mdLVs* algorithm is not less than that of *DLASQ*, we can compute singular values by the *mdLVs* algorithm with numerical accuracy equal to or higher than that of the dqds algorithm. In this paper, we propose four methods for implementing the *mdLVs* algorithm. We apply these methods and experiment for performances both in computational time and errors using 5 kinds of CPUs. As results, we have succeed in realizing highly accurate ability of the *mdLVs* algorithm into full play.

1. はじめに

計算機で数値計算をする際，種々のアルゴリズムを新たに独自実装するのではなく，ライブラリを利用する方法が一般的である．既存のものとして，線形数値計算ライブラリ LAPACK[15] や NAG 数値計算ライブラリ[16] などがある．ライブラリに収められている主要な数値計算ルーチンの 1 つとして，特異値分解用ルーチンがある．特異値分解は，データ検索[14]，画像処理[20]，最小 2 乗問題を扱うアプリケーションなどにおいて広く利用されている．

基本的な特異値分解法として，QR 法[2),3),6)~8),17)] や qd（quotient-difference）法[5),21),22)] がある．これらは，原点シフトの導入によって，高速化される．qd

† 独立行政法人科学技術振興機構さきがけ
PRESTO, Japan Science and Technology Agency
†† 京都大学大学院情報学研究科
Graduate School of Informatics, Kyoto University
††† 九州大学大学院数理学研究院
Faculty of Mathematics, Kyushu University
☆ 現在，独立行政法人科学技術振興機構 CREST/立教大学
Presently with CREST, Japan Science and Technology Agency/Rikkyo University

法を改良した dqds（原点シフト付き differential qd）法は，高速かつ高精度であるが，文献 21) においても収束性についての議論が不十分であるため，信頼性に欠ける．そのためか，MATLAB や Mathematica をはじめとする数学支援ソフトウェアでは，計算量が多い QRs（原点シフト付き QR）法が今もなお採用されている．

一方，可積分系の視点から，岩﨑と中村は，dLV（離散ロトカ・ボルテラ：discrete Lotka-Volterra）系[18] による新たな特異値計算法を定式化した[11]．この新計算法は，シフトの導入[12),19)] により，QRs 法より計算量が少なく，誤差解析において dqds 法と同程度の高精度性を示す[19]．また，収束性が証明されており，dqds 法よりも確実な特異値計算が可能である[12),19)]．

我々は，可積分系による数値計算パッケージ *LAPIS*（Linear Algebra Package by Integrable Systems）の開発の一環として，新たな特異値分解ライブラリ *DBDSLV* を実装し公開する予定である．この *DBDSLV* では，可積分系による特異値分解法 *I-SVD*（Integrable-Singular Value decomposition）によって，特異値と特異ベクトルを得ることができる．本論文では，この新ライブラリ *DBDSLV* で用いられる高性能な特異値計算ルーチンの実装について論じる．また，種々の計算機における提案ルーチンの実性能や有効性についても検討する．

2 章では，特異値分解法としての QRs 法，dqds 法，dLV 法およびその高速版である *mdLVs*（原点シフト付き modified dLV）法[10),12),19)] について説明する．3 章では，実装手法を 4 つ提案する．4 章では，5 種類の CPU における計算の精度と時間を比較・検証する．

2. 特異値分解法

任意の長方形行列 A は，ハウスホルダ変換[2),7)] により直交行列と上 2 重対角正方行列 B の積に分解できる．このとき，B は A の特異値を保持する．QRs 法[2),3),6)~8),17)] により B は，直交行列と対角行列の積に分解でき，A の特異値分解が完了する．QRs 法のほかに，dqds 法[5),21),22)] とツイスト分解による高速特異ベクトル計算法[4] を併用する方法もある．また，我々が開発中の新たな特異値分解法 *I-SVD* を基盤とする特異値分解ライブラリ *DBDSLV* もある．

表 1 は，本論文で用いる用語の一覧表である．

以下，2.1 節において，特異値分解に関する研究の進展状況について述べ，特異値分解法 *I-SVD* の位置づけを明確にする．2.2 節では，dLV 系による特異値計算について説明する．2.3 節では，原点シフト付きの高速版である *mdLVs* 法について概説する．

表 1 用語一覧表
Table 1 The table of the main words.

	計算法	ルーチン
LAPACK	QRs	*DBDSQR*
	dqds	*DLASQ*
LAPIS	*I-SVD*	*DBDSLV*
	mdLVs	*DLV_**

2.1 特異値分解法の進展

QRs 法は，特異値と特異ベクトルの計算を同時進行しながら求める特異値分解法である．この方法は，関連研究が十分なされており，特異ベクトルの直交性も良好であるため，ユーザが多い．しかし，行列サイズが大きくなると，収束が遅くなり，特異値の相対精度も悪くなるという欠点がある．

1990 年代後半，3 重対角対称行列のツイスト分解を活用した高速特異ベクトル計算法が考案された[4]．この方法では，先に計算された近似特異値からツイスト分解によって近似特異ベクトルを計算し，逆反復法を 1 回実行するだけで精度が向上した特異ベクトルを得る．

特異値計算と特異ベクトル計算の分離によって，高速性を獲得する反面，近似特異値に含まれる誤差は，それに対応する特異ベクトルに伝播する．その結果，予想以上の精度および直交性の悪化が生じる可能性がある．特に，特異値の絶対誤差が大きい場合，直交性が崩れやすい．ゆえに，特異値計算と特異ベクトル計算を前後に分離した計算法で高精度かつ高速な特異値分解を実現するためには，特異値計算法の精度を上げることが重要である．

高精度な特異値計算法として，dqds 法と *mdLVs* 法がある．dqds 法は，QRs 法の 2 割ほどの時間でより高精度な特異値を計算する．しかし，特異値への収束性についての厳密な証明はない[21]．また，零に近い値での除算や負の数が現れることがあり，その場合，原点シフトをとり直して再計算しなければならない．このため，線形数値計算ライブラリ LAPACK では，dqds 法とツイスト分解法を組み合わせたルーチン *DSTEGR* があるが，大きな誤差を含んだまま停止する可能性がある．

一方，我々は，**図 1** の dLV 系[18] の特異値分解法 *I-SVD* に基づくライブラリ *DBDSLV* を開発している．このライブラリの特徴として，収束性が保証された特異値計算法である *mdLVs* 法の採用があげられる[10),12),19)]．*mdLVs* 法については，2.3 節で述べる．また，*mdLVs* 法の特異値への収束性[12] と dLV 系に

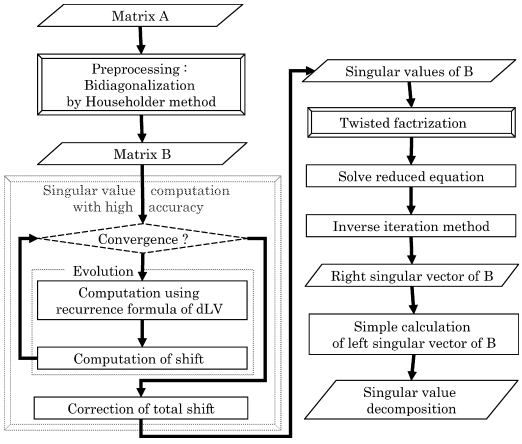


図 1 特異値分解法 *I-SVD* を基盤とするライブラリの流れ
Fig. 1 The flow in the library based on the singular value decomposition *I-SVD*.

よるツイスト分解については，現在投稿準備中である．

各特異値計算法は，変数 n $(n = 0, 1, 2, ...)$ を増加させることによって発展する．特異値計算の核部分において，n を 1 増加させるごとに生じる最大誤差は以下のとおりである．既存手法の QR 法と dqd 法では，それぞれ，$69M^2\varepsilon$，$3(M-1)\varepsilon$ である[5]．ここで，ε はマシンイプシロン程度である．dLV 法では $(2M-1)\varepsilon + (2M-1)\varepsilon'$ (ただし，$\varepsilon \gg \varepsilon'$) となる[19]．この状況は，原点シフト付き版でも大きく変わらない．ゆえに，*mdLVs* 法は，理論的に QRs 法より良く dqds 法と同程度以上の精度で特異値を計算できる．

2.2 離散ロトカ・ボルテラ系による特異値計算法

数理生物学では，複数の生物種の捕食関係のモデルとして LV 系（ロトカ・ボルテラ系）[23] が知られている．可積分な場合の LV 系に対して，可積分系特有の性質を用いた時間変数の離散化を行うと，漸化式

$$u_k^{(n+1)} = \frac{1 + \delta^{(n)} u_{k+1}^{(n)}}{1 + \delta^{(n+1)} u_{k-1}^{(n+1)}} u_k^{(n)} \qquad (1)$$

が導出される[18]．k $(k = 1, 2, ..., 2M-1)$ は種の番号を表す．$u_k^{(n)}$ と $\delta^{(n)}$ は，時刻 n における要素（変数）u_k と差分間隔をそれぞれ表す．$\delta^{(n)}$ は，0 以外の任意の値をとることができる．$\delta^{(n)} > 0$ とすれば，漸化式 (1) では減算，零での除算はない．初期値 $u_k^{(0)}$ が正ならば，つねに正値性が保たれる．この性質は，丸め誤差や数値不安定といった問題が発生しないことを意味し，負の数を積極的に扱う必要がない特異値計算にとって好ましい．

境界条件を式 (2)，初期条件を式 (3) とする．

$$u_0^{(n)} \equiv 0, \quad u_{2M}^{(n)} \equiv 0, \qquad (2)$$

$$u_k^{(0)} = \frac{(b_k)^2}{1 + \delta^{(0)} u_{k-1}^{(0)}}. \qquad (3)$$

ここで，要素 b_{2i-1} $(i : 1 \le i \le M)$ と b_{2i} は，M 次元上 2 重対角行列 B の対角成分および非対角成分を表す．発展回数 n 無限大の極限で，$u_{2i-1}^{(n)}$ は第 i 番目の特異値の 2 乗に，$u_{2i}^{(n)}$ は 0 に収束する．

2.3 離散ロトカ・ボルテラ系による特異値計算法の高速化

QR 法や qd 法では，原点シフトによって実行時間の短縮が可能となる[2]．そこで，前節で説明した dLV 法への原点シフトの導入[19] について概説する．

新たに要素 $w_k^{(n)}$ と $v_k^{(n)}$ を導入し，以下のように定義する．

$$w_k^{(n)} = u_k^{(n)}(1 + \delta^{(n)} u_{k-1}^{(n)}), \qquad (4)$$

$$v_k^{(n)} = u_k^{(n)}(1 + \delta^{(n)} u_{k+1}^{(n)}). \qquad (5)$$

この際，式 (3) より，$w_k^{(n)}$ の初期値は，$w_k^{(0)} = {b_k}^2$ となる．また，漸化式 (1) にシフト量 $S^{(n)}$ を付加することで漸化式

$$\begin{aligned} w_{2i-1}^{(n+1)} &= v_{2i-1}^{(n)} + v_{2i-2}^{(n)} - w_{2i-2}^{(n+1)} - S^{(n)}, \\ w_{2i}^{(n+1)} &= v_{2i-1}^{(n)} v_{2i}^{(n)} / w_{2i-1}^{(n+1)}. \end{aligned} \qquad (6)$$

が導出でき，dLV 法の高速版である *mdLVs* 法が定式化される[10),19]．$S^{(n)}$ の決定には，Gersgorin 境界[9] や Johnson 境界[13] などによる最小特異値見積りが利用できる．一般的に，$S^{(n)}$ を大きくすれば収束は加速されるが，大きすぎると $u_k^{(n)}$ の正値性を壊す原因となり，場合によっては数値不安定になる．そこで，漸化式 (6) において誤った見積りを用いたならば，その誤りを検出し，漸化式 (1) によって発展し直し正値性を保たせる[10),12]．*mdLVs* 法は，dqds 法と比べ，漸化式の数や除算が多く，実行時間が長くなる．しかし，岩﨑・中村の定理[10),12),19] より，$S^{(n)}$ をある範囲に選べば，*mdLVs* 法は，理論的にも計算機上でも必ず収束することが証明されており，信頼性が高い．

3. プログラムの実装手法

数値計算ライブラリの開発には，任意の計算機での動作を考慮した設計，もしくは特定の計算機専用の設計を行う．どちらの場合も計算量とメモリ使用量を最小限に抑えることが望ましい．そこで，イタレーション数や変数・配列への代入回数を極力少なくする工夫が必要となる．キャッシュ・レジスタの積極的な活用も有効である．ただし，計算結果に含まれる丸め誤差

への実装方法の影響に注意する必要がある．

以下，3.1 節において，*mdLVs* 法のプログラムについて簡単に説明する．3.2 節において，ループ展開の適用とその効果について説明する．3.3 節では，メモリ使用量の削減のために，メモリ領域を共用する手法と一時保存用の変数を利用する手法を提案する．3.4 節では，複数の配列を 1 つの大きな配列にまとめ，局所参照性を向上させる手法について述べる．

3.1 ロトカ・ボルテラ系による特異値計算ルーチン

LAPACK にある dqds 法に基づく *DLASQ* では，上 2 重対角行列 B の成分を引数（DOUBLE PRECISION 型）とし，B の非対角成分が対角成分に比べて十分小さくなったとき，行列を 2 つに分割する SPLIT や行列の次元数を下げる減次を行いながら発展計算を繰り返すことで特異値を計算する[21]．

我々が開発する *mdLVs* 法による特異値計算ルーチンでは，発展計算部分が *DLASQ* と異なる．以下，n を 1 増加させることによって $w_k^{(n)}$ を発展させるために必要な手順を示す．

① 式 (4) を $u_k^{(n)}$ について解き，$w_k^{(n)}$ から $u_k^{(n)}$ を k の昇順に計算．
② 式 (5) を用いて，$u_k^{(n)}$ から $v_k^{(n)}$ を計算．
③ シフト量 $S^{(n)}$ を計算．
④ $S^{(n)}$ のチェックと発展計算．
- $S^{(n)}$ が有効な場合
  式 (6) を用いて，$v_k^{(n)}$ から $w_k^{(n+1)}$ を計算．
- その他
  式 (1) より，$w_k^{(n+1)} = v_k^{(n)}$．

配列の概念を導入すると，次のようになる．手順 ① では，配列 $W = (w_1^{(n)}, w_2^{(n)},..., w_{2M-1}^{(n)})$ より配列 $U = (u_1^{(n)}, u_2^{(n)},..., u_{2M-1}^{(n)})$ を求める．ここで，n は *mdLVs* 法全体の発展回数を表すものであり，各 n に対応する要素を保持する必要はない．ゆえに，各配列は下付き添え字に対応する 1 次元配列となる．手順 ② では，U より配列 $V = (v_1^{(n)}, v_2^{(n)},..., v_{2M-1}^{(n)})$ を計算する．手順 ③ で決定した適切なシフト量 $S^{(n)}$ を用いて，手順 ④ では V をもとに W を上書きする．

このプログラムにおいて，手順 ① ② のループでは，U と V の各要素を要素番号 k の昇順に更新する．手順 ① で $u_k^{(n)}$ の更新には $w_k^{(n)}$ と $u_{k-1}^{(n)}$ が，手順 ② で $v_k^{(n)}$ を計算するには $u_k^{(n)}$ と $u_{k+1}^{(n)}$ がそれぞれ必要となる．また，手順 ④ では，式 (6) のように，W の要素番号の偶奇によって異なる演算が行われ，$w_k^{(n+1)}$ を計算するためには $w_{k-1}^{(n+1)}$ が必要である．

3.2 ループ展開による性能向上

一般的なメモリやキャッシュでは，1 変数あたり 64 bit の情報を保持する．一方，Intel の Pentium シリーズなどの CPU では，拡張精度レジスタによって 80 bit 情報が保持される．すなわち，拡張精度レジスタでは，より正確な値を保持できる．FORTRAN コンパイラの 1 種である g77 では，オプション O3 により，ループ演算でレジスタの値を優先的に用いる．

```
do k = 1, 2M − 1, 2
    update the k^th element
    update the (k + 1)^th element
end do
```

図 2 手順 ① ② ④ に関するループ
Fig. 2 The loops in Step ①, ② and ④.

mdLVs 法の各手順で，k 番目の要素の値は更新直後，レジスタに保存される．次に，何の工夫もしなければ，メモリやキャッシュへ 64 bit 情報として格納される．その後，$k+1$ 番目の要素を更新する際，64 bit 情報による演算が行われる．ここで，手法 1 を適用することによって，各イタレーションでのデータ依存を増やしレジスタの有効活用ができれば，拡張精度レジスタによる精度向上が期待できる．

> **手法 1**
> ループ展開による精度向上と
> ループオーバヘッドの削減

ただし，各イタレーションにおけるデータ数が過剰になると，レジスタ数が不足する．そこで，各手順において，手法 1 を適用し，2 要素ずつ更新させる．その結果として，手順 ① ② ④ のループでは，**図 2** のように，レジスタに保存されている k 番目の値を用いて $k+1$ 番目の更新ができる．

拡張精度レジスタの CPU を *mdLVs* 法のプログラムで用いる場合，以下のような現象が起こる点に注意すべきである．dLV 系による特異値計算においては，$k-1$ 番目と $k+1$ 番目の要素が k 番目の要素に影響を与える．仮に $k-1$ 番目か $k+1$ 番目のどちらかの最小桁目の値が丸め誤差によって 64 bit のみで計算した場合と比べ 1 ずれたならば，1 発展後の k 番目の要素は，80 bit のみで計算した場合とも 64 bit のみで計算した場合とも異なる結果となる．たとえば，$u_1^{(n)} = 1/3$，$w_2^{(n)} = 1/15$，$w_3^{(n)} = 1/5$ の場合，理論的には $v_2^{(n)} = 5/84$ となる．仮に，$v_2^{(n)}$ の有効桁数が 3 桁であれば，$v_2^{(n)} \approx 0.0595$ となる．ここで，$w^{(n)}$ を有効桁数 3 または 6 桁で与え，$u^{(n)}$ を有効桁数 3 または 6 桁で保存すると，$w^{(n)}$ が 3 桁で $u^{(n)}$ が 6 桁の場合のみ $v_2^{(n)} = 0.0596$ となる．つまり，$w^{(n)}$ が 3 桁で $u^{(n)}$ が 6 桁の場合のみ，結果が悪くなる．こ

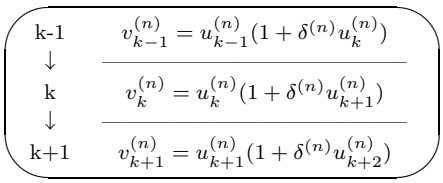


図 3 手順 ② における演算の流れ
Fig. 3 The flow of the computations in Step ②.

のように，計算途中で有効桁数が変更されると，誤差が増加することもある．拡張レジスタを用いた場合でも，レジスタとキャッシュ間で有効桁数の変更が行われる．ゆえに，きわめてまれではあるが，場合によっては，精度が悪化する可能性がある．この影響を避けるためには，64 bit レジスタを用いるか，FPU（浮動小数点演算装置）を操作するか，浮動小数点演算機を選択することによってレジスタ内の演算も 64 bit で実行させる必要がある．なお，乗算と加算の融合ユニットが搭載されている CPU では，他の CPU とは異なる精度の演算結果が得られる可能性がある．なぜならば，乗加算の融合演算ユニット内では，106 bit で保持された乗算の結果に対して加算が行われるからである[1)]．

速度に関して，手法 1 を適用した場合，各イタレーションにおいて，レジスタが有効活用できるため，速度向上が期待できる．また，ループ展開により，図 2 のように $M-1$ 回のループと $2M-1$ 番目の要素の更新のみが必要となるため，各ループを抜けるための判定回数は半減する．

3.3 メモリ使用量の削減

配列は，総メモリ使用量に大きく影響を与える．任意の計算機において *mdLVs* 法を実行するために，配列は本質的に必要なものだけとし，メモリ使用量を最小限にとどめるべきである．

手順 ① で更新される U は，手順 ③ ④ では不要で，手順 ② においてのみ必要である．手順 ② では，**図 3** が示すように，U をもとに V が更新される．ここで，$v_k^{(n)}$ を計算後，$u_k^{(n)}$ が不要となる．また，手順 ② では，V の全要素を上書きする．つまり，$v_k^{(n)}$ を $u_k^{(n)}$ に上書きでき，配列 V のためにメモリを新たに割り当てなくてよい．このような同一配列の複数回利用によるメモリ使用量の削減法を手法 2 とする．

手法 2
複数配列によるメモリ領域の再利用

手順 ② のループは昇順に行われるため，配列 U と V が同じメモリ領域を利用しても，書き込みミスは生じない．ただし，同じメモリ領域を共有するため，計算機アーキテクチャによっては，実行時間が遅くなる．

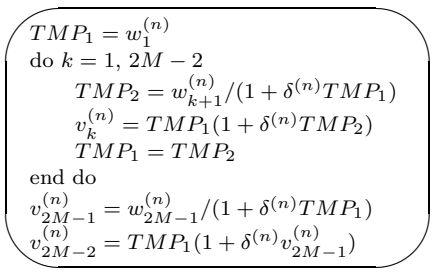


図 4 手順 ① ② に関するループの融合（1 イタレーションにつき 1 要素を更新）
Fig. 4 Loop fusion in Step ① and ② (An element is updated at each iteration).

実行遅延することなくメモリ使用量を削減するために，別の手法を考える．要素 $v_k^{(n)}$ の計算は，漸化式 (4)，(5) より，次のように展開される．

$$\begin{aligned} v_k^{(n)} &= u_k^{(n)}(1+\delta^{(n)}u_{k+1}^{(n)}) \\ &= u_k^{(n)}(1+\delta^{(n)}w_{k+1}^{(n)}/(1+\delta^{(n)}u_k^{(n)})) \quad (7) \end{aligned}$$

この展開式は，手順 ① ② のループの融合を意味する．式 (4)，(5)，(7) より，$u_k^{(n)}$ は，$v_k^{(n)}$ の更新と $v_{k+1}^{(n)}$ の計算に必要な $u_{k+1}^{(n)}$ の更新にのみ用いられる．ゆえに，配列 U に格納しておくことは，メモリの無駄である．そこで，一時保存用変数（TMP_1，TMP_2）によって必要最小限の $u_k^{(n)}$ を保持する手法 3 を提案する．

手法 3
一時保存用の変数を利用したループ融合

手順 ① ② のプログラムソースは，SPLIT[2)] で減次されなかった場合，**図 4** のように表される．手法 3 を適用しなければ，W から V を更新するのに必要な代入回数は，手順 ① では $2M-1$ 回，手順 ② では $2M-2$ 回であり，合計で $4M-3$ 回となる．ただし，ループの繰返し回数の数え上げは除く．一方，図 4 の場合，代入式 $TMP_1 = TMP_2$ が $2M$ 回必要なため，全体の代入回数は $6M-3$ 回に増加する．そのため，ループ融合によって，ループの判定回数が半減し，配列 U が不要になるという利点はあるものの，代入回数の増加によって計算コストが高くなる．

図 5 は，図 4 に対して，ループ展開する手法 1 を適用したものである．この結果，図 4 では必要な代入式 $TMP_1 = TMP_2$ を除くことができ，代入回数は $4M-3$ 回となる．また，ループの判定回数もさらに半減する．ゆえに，手法 3 は，手法 1 と併用すること

$$
\begin{aligned}
&TMP_1 = w_1^{(n)} \\
&\text{do } i = 1,\ M-2 \\
&\qquad TMP_2 = w_{2i}^{(n)}/(1+\delta^{(n)}TMP_1) \\
&\qquad v_{2i-1}^{(n)} = TMP_1(1+\delta^{(n)}TMP_2) \\
&\qquad TMP_1 = w_{2i+1}^{(n)}/(1+\delta^{(n)}TMP_2) \\
&\qquad v_{2i}^{(n)} = TMP_2(1+\delta^{(n)}TMP_1) \\
&\text{end do} \\
&TMP_2 = w_{2(M-1)}^{(n)}/(1+\delta^{(n)}TMP_1) \\
&v_{2(M-1)-1}^{(n)} = TMP_1(1+\delta^{(n)}TMP_2) \\
&v_{2M-1}^{(n)} = w_{2(M-1)+1}^{(n)}/(1+\delta^{(n)}TMP_2) \\
&v_{2(M-1)}^{(n)} = TMP_2(1+\delta^{(n)}v_{2M-1}^{(n)})
\end{aligned}
$$

図 5 手順 ① ② に関するループの融合（1 イタレーションにつき 2 要素を更新）
Fig. 5 Loop fusion in Step ① and ② (Two elements are updated at each iteration).

によって，より速くなる．つまり，配列への代入回数を 1/2 回に，ループの判定回数を 1/4 回に減少させることができ，実行時間が短縮される．

キャッシュ・レジスタの活用に関して，代入回数が等しい場合，図 5 のように手法 3 を適用したほうがよい．なぜならば，一時保存用の変数は，ループ演算中，レジスタもしくは L1 D に保持されるからである．一方，配列 U を用いると，各イタレーションで異なる要素が必要なため，下位キャッシュやメモリへのアクセスが生じる．よって，一時保存用変数を活用するほうが，高速に実行できる．

拡張精度レジスタでは，レジスタを活用することによる高精度化が可能である．手法 3 を適用する場合，一時保存用変数の値としてレジスタに保持されている値を利用することができるため，精度向上が期待される．

3.4 連続メモリアクセス

dqds 法の漸化式は以下のように表される．

$$
\begin{aligned}
d_k^{(n)} &= d_{k-1}^{(n)}(q_k^{(n)}/q_{k-1}^{(n+1)}) - S^{(n)} \\
q_k^{(n+1)} &= d_k^{(n)} + e_k^{(n)} \\
e_k^{(n+1)} &= e_k^{(n)}(q_{k+1}^{(n)}/q_k^{(n+1)})
\end{aligned} \qquad (8)
$$

ただし，$d_k^{(n)}$ は補助変数であり，要素 $q_k^{(n)}$ と $e_k^{(n)}$ はそれぞれ上 2 重対角行列 B の対角および非対角成分に対応する．発展回数 n 無限大で，$q_k^{(n)}$ はシフト移動した分を足しこむと B の特異値の 2 乗に，$e_k^{(n)}$ は 0 になると見なされる．

LAPACK の *DLASQ* では，**図 6** のような作業用配列 $WORK_{QD}$ を利用し各配列の発展回数の奇数番目と偶数番目を交互に保存している．これは，連続メモリアクセスを可能とさせ，キャッシュのヒット率を向上させるための工夫である．

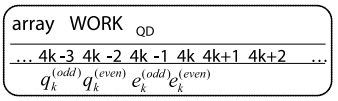


図 6 *DLASQ* で用いられる配列 $WORK_{QD}$
Fig. 6 The array $WORK_{QD}$ of *DLASQ*.

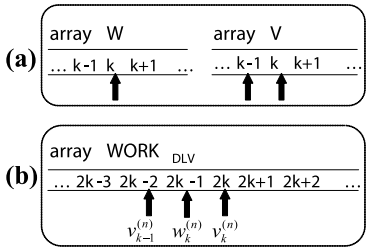


図 7 手順 ① ④ の各イタレーションにおいてアクセスされるデータ．(a) 配列 W と V を用いた場合．(b) 配列 $WORK_{DLV}$ を用いた場合
Fig. 7 The related data in each iteration of Step ① and ④. (a) In the case of the array W and V. (b) In the case of the array $WORK_{DLV}$.

mdLVs 法の手順 ① ④ では，**図 7** (a) のように，$v_k^{(n)}$ の更新には $v_{k-1}^{(n)}$ と $w_k^{(n)}$ へのアクセスが生じる．この際，コンパイラ g77 を用いると，配列はスタックに積まれるため，スタックからのとり出し遅延が生じる．そこで，作業用配列を用いて，2 つの配列を 1 つにまとめる．

手法 4
連続メモリアクセスによる
キャッシュヒット率の向上

mdLVs 法においては，*DLASQ* と異なり，ある発展回数 n に関する要素を作業用配列に格納する．配列 W と V より，サイズ $4M$ の作業用配列 $WORK_{DLV} = (w_1, v_1, w_2, v_2, ..., w_k, v_k, ..., w_{2M-1}, v_{2M-1})$ を定義する．$WORK_{DLV}$ では，図 7 (b) のように，ある要素の更新に必要な要素が同一配列の隣り合う場所に格納される．そのため，連続メモリアクセスが可能となり，図 7 (a) よりも実行時間が短縮される．ただし，キャッシュが十分な大きさであるならば，キャッシュミスの確率が減り，複数の配列を 1 つの大きな配列にまとめなくてもよい．したがって，作業用配列 $WORK$ を用いるべきかは，計算機および問題規模などを十分考慮して判断する必要がある．

4. 各 CPU における実行結果

前章で提案した 4 つの実装手法のうち，ループオーバヘッドを削減する手法 1 は，ループの判定回数を減

表 2 *mdLVs* 法におけるルーチン名と手法の対応表
Table 2 The relation of routines and the methods in *mdLVs*.

	手法 1	手法 2	手法 3	手法 4
DLV_O	○	×	×	×
DLV_M	○	○	×	×
DLV_T	○	×	○	×
DLV_TW	○	×	○	○

表 3 計算機性能
Table 3 The performance of each computer.

80 bit register (default)			
	C_{M3}	C_{P4}	C_X
CPU	Pentium 3	Pentium 4	Xeon
(Intel)	800 MHz	2.6 GHz	3.2 GHz
Memory	500 MB	1 GB	2 GB
L1 D	16 KB	8 KB	8 KB
L1 I	16 KB	12 Kμops	12 Kμops
L2	512 KB	512 KB	512 KB
OS	Vine Linux 3.0	Debian 3.0	Debian 3.0
	Linux 2.4.26	Linux 2.4.24	Linux 2.4.27
Compiler	GNU 3.3.2	GNU 2.95.4	GNU 2.95.4
64 bit register (default)			
	C_{G5}	C_O	
CPU	Power	AMD	
	PC G5	Opteron	
	2.0 GHz	2.4 GHz	
Memory	3.5 GB	2 GB	
L1 D	32 KB	64 KB	
L1 I	64 KB	64 KB	
L2	512 KB	1,024 KB	
OS	Darwin	Fedora Core 2	
	7.5.0	Linux 2.6.5	
Compiler	GNU 3.4	GNU 3.3.3	

少させることができるため，必ず実装する．この際，拡張精度レジスタの CPU において実行した場合，レジスタ内の値を用いた演算と 64 bit 演算との間に差が生じる可能性があることを考慮しなければならない．

本章では，QRs 法と dqds 法を比較対象に，*mdLVs* 法の計算機における実性能について検証する．QRs 法のルーチンとしては，LAPACK の *DBDSQR* のベクトル計算に関する部分をコメントアウトしたものを用いた．dqds 法としては，LAPACK の *DLASQ* を用いた．*mdLVs* 法として，**表 2** の 4 つのルーチンを作成した．この際，差分間隔は固定値 $\delta^{(n)} = 1$，シフト量 $S^{(n)}$ の見積りは Johnson 境界を用いた．また，非対角成分が対角成分に比べて十分小さいかどうかの指標を *DLASQ* と同様の値を用いることによって，SPLIT や収束による減次を行うよう実装した．各実験に用いた計算機性能は**表 3** のとおりである．ただし，C_{M3} はモバイル用の CPU であり，C_O での実行は 64 bit Mode である．GNU コンパイラのオプションには O3 を指定し，各計算機で異なるバイナリを作成する．

表 4 各特異値に対する相対誤差の総和
Table 4 The total number of the relative errors in each singular value.

	Max	Min	Average	σ
C_{M3}				
DBDSQR	1.26E-12	7.06E-13	9.51E-13	1.98E-13
DLASQ	1.09E-12	2.01E-13	5.86E-13	2.11E-13
DLV_O *DLV_M*	1.13E-12	3.09E-13	5.59E-13	2.28E-13
DLV_T *DLV_TW*	1.15E-12	2.41E-13	4.75E-13	2.14E-13
C_{P4}				
DBDSQR	1.26E-12	7.08E-13	9.48E-13	1.97E-13
DLASQ	1.27E-12	1.90E-13	4.56E-13	2.14E-13
DLV_O *DLV_M*	3.14E-13	1.76E-13	2.19E-13	2.65E-14
DLV_T *DLV_TW*	2.51E-13	1.43E-13	1.85E-13	2.27E-14
C_X				
DBDSQR	1.26E-12	7.17E-13	9.69E-13	2.00E-13
DLASQ	1.23E-12	2.16E-13	4.52E-13	1.88E-13
DLV_O *DLV_M*	3.17E-13	1.77E-13	2.22E-13	2.74E-14
DLV_T *DLV_TW*	2.85E-13	1.47E-13	1.87E-13	2.24E-14
C_{G5}				
DBDSQR	1.70E-12	9.51E-13	1.31E-12	2.70E-13
DLASQ	2.59E-13	1.59E-13	1.91E-13	1.67E-14
DLV_O *DLV_M* *DLV_T* *DLV_TW*	9.94E-13	3.26E-13	5.39E-13	1.38E-13
C_O				
DBDSQR	1.83E-12	1.05E-12	1.41E-12	2.85E-13
DLASQ	1.05E-12	2.92E-13	5.64E-13	1.56E-13
DLV_O *DLV_M* *DLV_T* *DLV_TW*	1.05E-12	3.28E-13	5.32E-13	1.32E-13

以下，4.1 節と 4.2 節では，それぞれ，誤差と実行時間に関する実験結果を示し，考察を行う．

4.1 誤差に関する比較実験

計算機内部で表現できる数字に桁数制限があるため，丸め誤差が生じる．ツイスト分解を利用する場合，特異値に含まれる誤差が特異ベクトルの精度・直交性に大きく影響する．精度の良い特異ベクトルを求めるには，可能な限り高精度な特異値計算法を選択する必要がある．本節では，QRs 法，dqds 法，および *mdLVs* 法の精度をそれぞれのルーチンを通じて比較する．

誤差を調べるために，$[1, 500]$ のランダムな特異値を持つ 1,000 次元上 2 重行列を 100 個生成した．表 3 に示す各計算機についての特異値計算を行った実験結果を**表 4** に示す．σ は標準偏差を表す．

平均値に関して，すべての計算機で，*DBDSQR* が最も悪い精度となった．*mdLVs* 法について，C_{G5} や C_O では，実装手法に関係なく，各特異値の相対誤差の総和が同じとなった．これは，64 bit のメモリ・キャッシュ・レジスタを用いたためである．一方，Intel 系列の CPU では，80 bit レジスタであるため，手法 3 を適用する場合としない場合では，異なる誤差となった．

C_{M3}，C_{P4}，C_X では，必要とした発展回数はほぼ同じであったにもかかわらず，*mdLVs* 法のルーチンのほうが，*DLASQ* よりも高精度であった．ただし，C_{M3} のようなモバイル専用 CPU においては，問題行列によって *DLASQ* よりも精度が悪くなることがあった．変数は，レジスタ内とキャッシュ・メモリ内では異なる bit 長で保持される．そのため，レジスタ・キャッシュ間を移動する際に丸め誤差が生じ，精度が悪化するものと考えられる．FPU を操作してレジスタを 64 bit で演算した場合，*mdLVs* の全ルーチンの平均相対誤差は，C_{M3} では $5.24E-13$，C_{P4} では $5.21E-13$ であった．同様に，*DBDSQR* と *DLASQ* の平均相対誤差は，C_{M3} では $1.41E-12$ と $5.33E-13$，C_{P4} では $1.41E-12$ と $5.41E-13$ であった．ゆえに，80 bit レジスタによって，*mdLVs* のルーチンの精度が向上し，*DLASQ* よりもより高精度となったと考えられる．

C_O では，*DLASQ* も *mdLVs* 法の各ルーチンもほぼ同精度であった．よって，理論どおりの相対誤差が得られたと考えられる．C_O において，コンパイラオプション *mfpmath* を用いて 387 命令の 80 bit レジスタによる実験を行った．その結果，*DBDSQR* と *DLASQ* の平均相対誤差は，$1.28E-12$ と $5.65E-13$ であった．一方，*DLV_O* と *DLV_T* では，$5.14E-13$ と $4.98E-13$ であった．Intel 系の CPU ほどの精度向上はみられなかったものの，387 命令の 80 bit レジスタの利用によって，良好な結果が得られた．

C_{G5} においては，*DLASQ* のほうが良好な精度を得た．これは，Power PC G5 の乗加算の融合演算ユニット内において，変数が 4 倍精度で保持されることによって，丸め誤差が少なくなったことが原因と理解される．

DLASQ の場合，$e_k^{(n)}$ は，発展が進むにつれ 0 に近づく．そのため，$q_k^{(n)}$ を高精度に得るためには，変数 $d_k^{(n)}$ の計算精度を上げる必要がある．乗加算の融合演算ユニットを用いた計算において，高精度に $d_k^{(n)}$ を計算できる理由として，次の 2 つが考えられる．1 つ目は，式 (8) より，変数 $d_k^{(n)}$ は，乗加算で表されているため，融合演算ユニットによって高精度に計算できる点である．2 つ目は，$d_k^{(n)}$ の計算において生じる桁落ちを回避する点である．シフト量 $S^{(n)}$ は最小特異値の 2 乗に近いため，$d_k^{(n)}$ は 0 に非常に近い値となる．これは，$d_k^{(n)}$ の計算において大きな桁落ちが生じていることを意味する．ここで，乗加算融合ユニットを使うと，引かれる数 $d_{k-1}^{(n)}(q_k^{(n)}/q_{k-1}^{(n+1)})$ が 4 倍精度で保持されるため，$S^{(n)}$ を引いても下位のビットが正しく残り，$d_k^{(n)}$ を高精度に計算できるものと考えられる．

一方，*mdLVs* 法において，乗加算の融合ユニットを利用しているにもかかわらずその効果が薄い理由として，次の 2 つが考えられる．1 つ目は，手順 ② の $v_k^{(n)}$ の計算である．この計算において必要とされる値はすべて正値であるため，桁落ちによる精度悪化が生じにくい．2 つ目は，式 (6) の $w_{2i-1}^{(n+1)}$ の更新である．非対角成分を表す $v_{2i-2}^{(n)}$ と $w_{2i-2}^{(n+1)}$ は，発展が進むにつれ 0 に近づく．ゆえに，$w_{2i-1}^{(n+1)}$ の更新とは，対角成分からシフト量を減算することを意味し，中間結果が生じない．また，シフト量は，手順 ③ の Johnson 境界によって見積もられるため，平方根計算を必要とし誤差を含む可能性がある．以上より，*mdLVs* 法では，Power PC G5 の CPU 特性を活かしきれないために，現状では，対角成分を高精度に計算することはできていないものと考えられる．

図 8 は，80 bit レジスタを用い *mdLVs* 法の各ルーチンによって計算された特異値の総相対誤差をグラフ化したものである．縦軸は，各行列の要素の相対誤差の総和を表す．横軸には，*DLV_O* または *DLV_M* の総相対誤差の大きい順に 100 個の行列を並べている．表 4 の結果より，手法 3 を適用したものが，概して小さな総相対誤差を持つことが分かる．これは，手法 3 の一時保存用変数の適用によって，レジスタが有効利用できたことが原因であると考えられる．ただし，図 8 より，いくつかの行列に関しては，*DLV_O* および *DLV_M* のほうが誤差が小さい．これは，レジスタが保持していた各要素をキャッシュやメモリに移動する際，64 bit の情報に変換されることによって丸め誤差が生じ，その影響が隣り合う要素に伝播することが原因であると考えられる．しかし，図 8 より，*DLV_T* および *DLV_TW* のほうが全体的に良い精度を得ている．以上より，80 bit レジスタでは，*mdLVs* 法に一時保存用変数を用いてループ融合を行う手法 3 を適用したほうが良い精度を得られる．

各計算機における各特異値の相対誤差は，**図 9** のようになった．横軸は特異値の大きい順に番号を振った

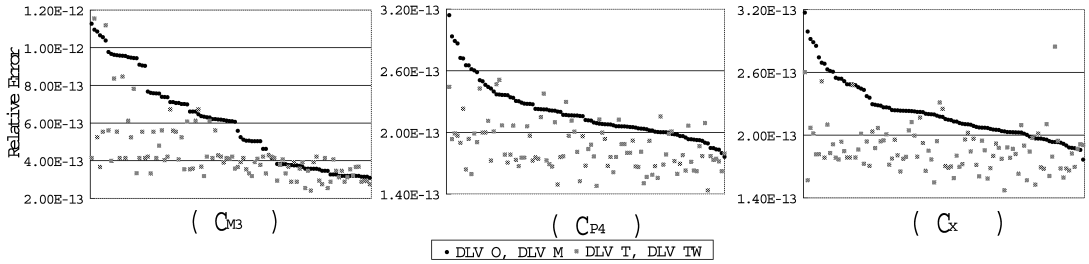


図 8 各行列の総相対誤差（80 bit レジスタを利用）
Fig. 8 The total relative errors in each matrix (using 80 bit register).

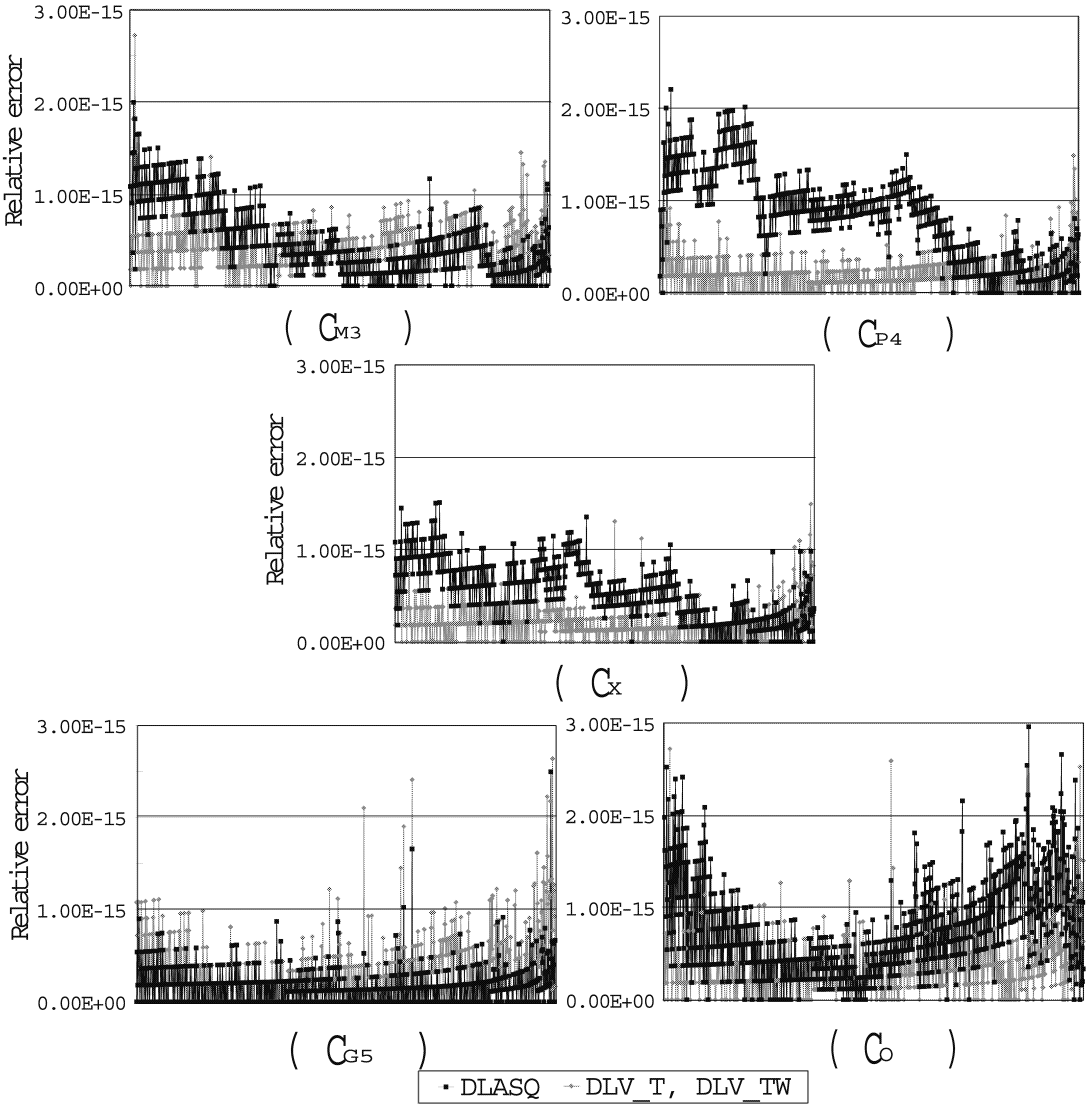


図 9 各特異値に関する相対誤差
Fig. 9 Relative errors of computed singular values.

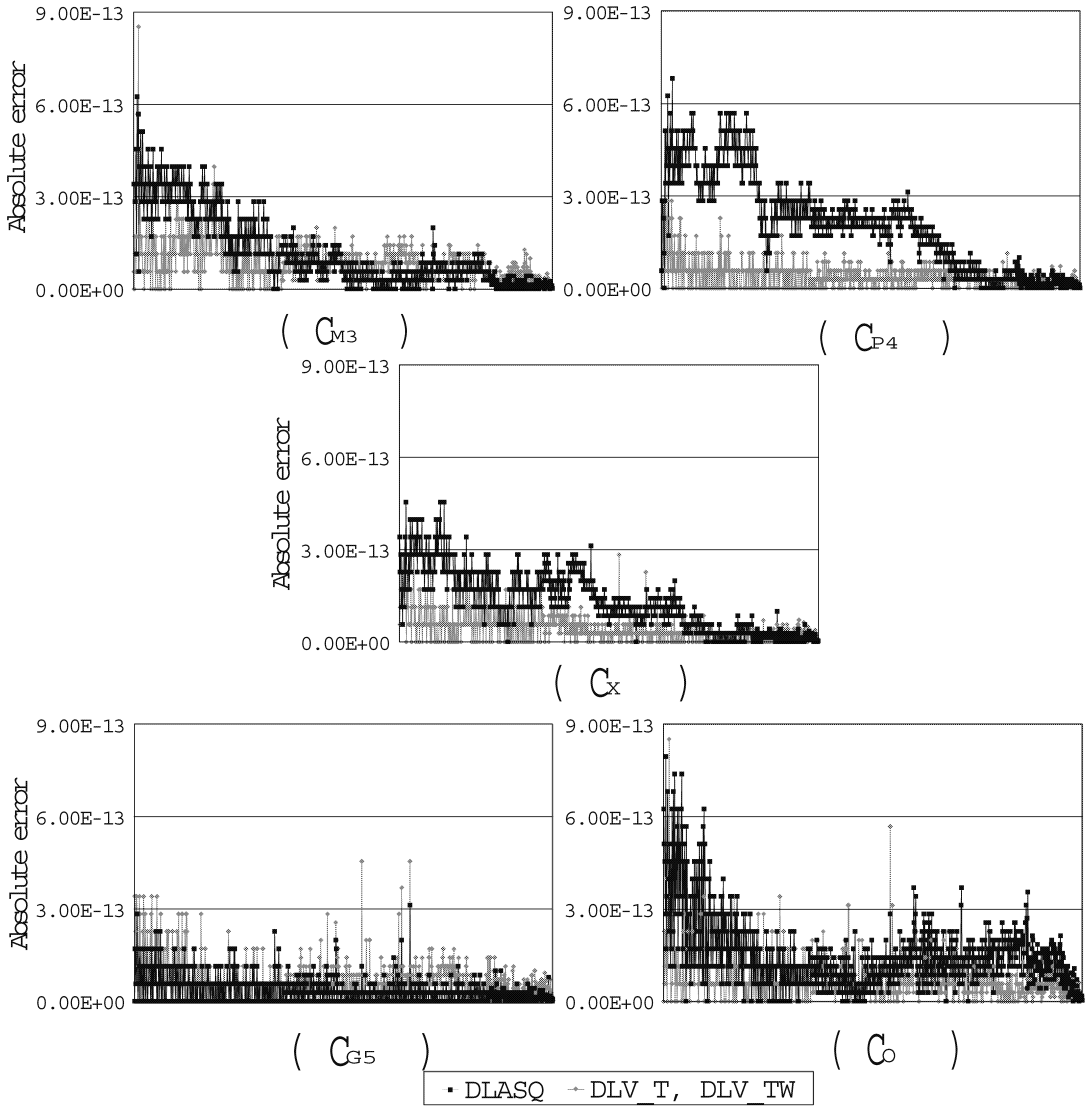


図 10 各特異値に関する絶対誤差
Fig. 10 Absolute errors of computed singular values.

状態を表し，縦軸は特異値の相対誤差を表す．図 9 より，80 bit レジスタでは，*DLASQ* を用いた場合，特異値が大きくなるにつれて相対誤差も大きくなっていることが分かる．一方，*DLV_T* と *DLV_TW* は，特異値が小さい場合，若干 *DLASQ* より相対誤差が大きいものの，全体的に一定であった．

計算された特異値からツイスト分解を利用して特異ベクトルを計算する特異値分解法では，特異値の誤差が特異ベクトルに大きく影響する．特に，大きな特異値に対する誤差は，小さな特異値の誤差に比べて同じ相対精度であっても非常に直交性を悪化させる原因となる．**図 10** は，図 9 の行列における絶対誤差を表す．C_{G5} を除くどの計算機においても，*DLASQ* のほうが，特異値の誤差が大きい．また，すべての計算機において，特異値の値が大きければ大きいほど，*DLASQ* の特異値に対する絶対誤差が大きい．これは，特異ベクトルが大きな誤差を含む原因となる．一方，*DLV_T* および *DLV_TW* は，大きい特異値の誤差が若干増えるものの，全体的に一定の誤差以下に抑えられている．ゆえに，精度面において，一時保存用の変数を用いてループ融合する手法 3 を適用した *DLV_T* もしくは *DLV_TW* が有効であると考えられる．

4.2 実行時間に関する比較実験

実行時間を比較するために，$[1, 100]$ の乱数を割り

表 5 10,000 次元行列の実行時間
Table 5 The computational time in 10,000 dimensional matrices.

	Max(s)	Min(s)	Average(s)	σ
C_{M3}				
DBDSQR	60.91	48.1	54.44	2.73
DLASQ	15.41	12.77	14.41	0.50
DLV_O	52.14	42.32	48.54	1.94
DLV_M	55.4	42.43	48.89	2.26
DLV_T	52.21	40.67	47.08	2.29
C_{P4}				
DBDSQR	19.59	16.33	18.22	0.67
DLASQ	5.05	3.93	4.73	0.17
DLV_O	16.75	13.7	15.58	0.60
DLV_M	16.77	13.72	15.60	0.60
DLV_T	15.00	12.05	13.71	0.53
C_X				
DBDSQR	16.22	13.26	14.91	0.61
DLASQ	4.14	3.23	3.87	0.14
DLV_O	13.74	11.22	12.79	0.50
DLV_M	14.11	11.24	12.79	0.51
DLV_T	12.83	9.84	11.29	0.50
C_{G5}				
DBDSQR	13.71	11.44	12.53	0.41
DLASQ	4.45	3.66	4.21	0.14
DLV_O	17.23	13.4	16.11	0.73
DLV_M	17.20	13.28	16.09	0.73
DLV_T	14.52	11.25	13.59	0.61
C_O				
DBDSQR	10.18	7.67	8.94	0.47
DLASQ	2.84	2.20	2.65	0.10
DLV_O	8.73	7.07	8.05	0.30
DLV_M	8.88	7.2	8.19	0.31
DLV_T	7.87	6.39	7.27	0.27

当てた 10,000 次元上 2 重行列を 100 個用意した．

まず最初に，メモリ使用量の削減を目的とする手法 2 と 3 の比較実験を行う．各計算機の実行時間を**表 5** に示す．どの CPU においても，*mdLVs* 法のルーチンは，*DLASQ* の 3 から 4 倍の実行時間を必要とする．これは，dqds 法と比べ，漸化式がやや複雑かつ除算が多いことに加え，計算時間の 4 割を占める Johnson 境界を用いたシフト見積りが原因である．Johnson 境界には行列の全要素が必要であるが，シフトが導入された *mdLVs* 法の収束性を保証する．一方，*DLASQ* では，行列のいくつかの要素のみを用いて大雑把なシフト見積りをするため，計算量は小さい．しかし，零に近い値での除算を行う恐れがあり，そのため *DLASQ* は例外処理を必要としている．

DBDSQR についてみれば，C_{G5} の場合にのみ，*mdLVs* 法の各ルーチンより速度が速い．この原因としては，QRs 法において内積計算の占める割合が大きいためであると考えられる．

mdLVs 法に関して，*DLV_O* と比べ，*DLV_M* は若干実行が遅い．これは，各配列にメモリ領域を割り当ててある場所からロードしたデータを別の場所にストアするほうが，表 3 のアーキテクチャにおいては速いためである．しかし，大きな速度差はなく，任意の計算機で実行させるためにメモリ使用量の削減を重視すべきであり，その点で *DLV_M* は有効であるといえよう．*DLV_T* は，メモリ使用量の削減だけでなく，配列への代入回数とループの判定回数が減少するため，どの CPU においても実行時間の短縮に成功している．ゆえに，メモリ使用量を削減し，なおかつ，実行時間を短縮するためには，一時保存用の変数を利用したループ融合の手法 3 は，手法 2 よりも有効である．

表 6 10,000 次元行列に対する *DLV_TW* の実行時間
Table 6 The computational time of the *DLV_TW* in 10,000 dimensional matrices.

	Max(s)	Min(s)	Average(s)	σ
C_{M3}	55.43	41.53	48.27	2.60
C_{P4}	13.85	11.11	12.66	0.49
C_X	11.33	9.09	10.36	0.40
C_{G5}	15.02	11.56	14.04	0.64
C_O	8.05	6.53	7.42	0.27

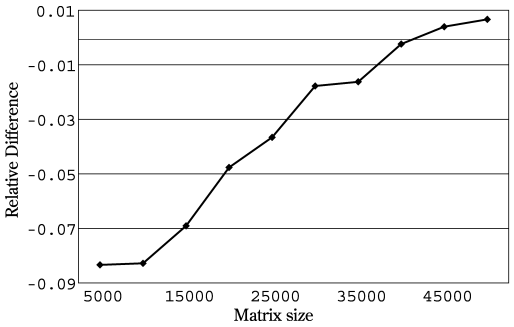


図 11 *DLV_T* に対する *DLV_TW* の平均時間の相対的な差
Fig. 11 The relative differences to *DLV_TW*'s average time and *DLV_T*'s.

次に，2 つの配列を 1 つの配列に格納する手法 4 の性能を調べる．**表 6** は，各計算機における *DLV_TW* の実行時間を表す．*DLV_T* と比べ，C_{M3}，C_{G5}，C_O では実行時間が長くなった．

C_{P4} を用いて，$[1, 100]$ の乱数を持つ 5,000 から 50,000 次元上 2 重行列 100 個の実行時間を調べた．**図 11** の横軸は行列サイズを表し，縦軸は *DLV_T* の実行時間に対する *DLV_TW* の相対的な実行時間差を表す．図 11 に表されるように，*DLV_TW* のほうが *DLV_T* よりも長くなる．

以上より，どのような問題に対しても *mdLVs* 法を有効に利用するためには，*DLV_T* の実装を行えばよい．また，比較的小規模な問題に対しては，*DLV_TW* を利用することができる．

5. ま と め

本論文では，特異値計算の *mdLVs* 法のルーチン開発とその実性能について論じた．代表的な特異値分解法である QRs 法は，信頼性が高いが，計算量が多く特異値の精度も悪い．また，特異値のみを高速計算する dqds 法は，収束性と信頼性に問題があるため，線形数値計算ライブラリ LAPACK に *DLASQ* があるが，ソフトウェアに利用されている実例はあまり見当たらない．さらに，LAPACK の *DSTEGR* は dqds 法を用いて特異ベクトルの計算までできるが，QRs 法に比べ，格段に悪い精度となることがある．

mdLVs 法を用いた特異値計算ルーチンの開発にあたって，精度と速度の向上を目的とする手法を 4 つ提案した．手法 1 は，ループを展開し各イタレーションで 2 要素ずつ更新させることによって，ループオーバヘッドを削減する手法である．手法 2 は，配列 U と V に同じメモリ領域を割り当てることによって，メモリ使用量を抑える手法である．手法 3 は，一時保存用変数によるループ融合であり，計算量の減少とレジスタの有効活用を可能とする．手法 4 は，作業用配列を用いて連続メモリアクセスを可能にする手法である．

実行時間に関して，小規模問題の場合，*DLV_TW* が最も速い．しかし，大規模計算では，手法 4 を用いない *DLV_T* のほうが速い．つまり，中小問題に対しては *DLV_TW* を用い，それ以外に対しては *DLV_T* を用いることによって，どの CPU においても実行時間の改善に成功した．

精度については，Power PC G5 を除いて，*mdLVs* 法のルーチンは，*DLASQ* より精度が良いことが分かった．とりわけ，80 bit レジスタを持つ CPU においては，より高精度である．

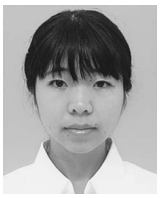
本論文では，*DLASQ* の 3 から 4 倍程度の実行時間を要するが，高精度かつ確実に特異値を計算する *mdLVs* 法のルーチンを開発した．さらに，*DBDSQR*, *DLASQ* と *mdLVs* 法の各ルーチンについて，各特異値の相対誤差と絶対誤差を比較し，特異値分解を行ううえでの *mdLVs* 法の精度上の優位性を示した．その結果，*mdLVs* 法のルーチンは，収束が保証されているルーチンの中で，最も高速かつ高精度に特異値計算をすることが分かった．このルーチンは，パイプラインの有効利用によるシフト量の見積り時間の大幅な削減を経て，特異値分解ライブラリ *DBDSLV* に組み込まれる予定である．特異値分解全体の実行時間に対して，特異値計算部の計算時間の占める割合は非常に小さい．この意味で，*mdLVs* 法を実装した *DLV_T* は有効な特異値計算ルーチンといえよう．

参 考 文 献

1) Agarwal, R.C., Enenkel, R.F., Gustavson, F.G., Kothari, A. and Zubair, M.: Fast pseudorandom-number generators with modulus 2^k or 2^{k-1} using fused multiply-add, *IBM J. RES. & DEV.*, Vol.46, No.1, pp.97–116 (2002).
2) Demmel, J.: *Applied Numerical Linear Algebra*, SIAM, Philadelphia (1997).
3) Demmel, J. and Kahan, W.: Accurate singular values of bidiagonal matrics, *SIAM J. Sci. Sta. Comput.*, Vol.67, pp.191–229 (1994).
4) Dhillon, I.S. and Parlett, B.N.: Orthogonal eigenvectors and relative gaps, *SIAM J. Matrix Anal. Appl.*, Vol.25, No.3, pp.858–899 (2004).
5) Fernando, K.V. and Parlett, B.N.: Accurate singular values and differential qd algorithms, *Numer. Math.*, Vol.67, pp.191–229 (1994).
6) Francis, J.G.F.: The QR transformation a unitary analogue to the LR transformation-part 1, *Computer J.*, Vol.4, pp.265–271 (1961).
7) Golub, G. and Kahan, W.: Calculating the singular values and pseudo-inverse of a matrix, *SIAM J. Numeri. Anal.*, Vol.2, pp.205–224 (1965).
8) Golub, G. and Reinsch, C.: Singular value decomposition and least squares solutions, *Numer. Math.*, Vol.14, pp.403–420 (1970).
9) Henrici, P.: *Applied and Computational Complex Analysis Volume I*, Wiley-Interscience Publishing, NewYork (1988).
10) Iwasaki, M.: Studies of Singular Value Decomposition in Terms of Integrable Systems, Doctor Thesis, Kyoto University (2004).
11) Iwasaki, M. and Nakamura, Y.: On the convergence of a solution of the discrete Lotka-Volterra system, *Inverse Problems*, Vol.18, pp.1569–1578 (2002).
12) Iwasaki, M. and Nakamura, Y.: Accurate computation of singular values in terms of the shifted integrable scheme, (2005). (preprint)
13) Johnson, C.R.: A Gersgorin-type lower bound for the smallest singular value, *Lin. Alg. Appl.*, Vol.112, pp.1–7 (1989).
14) 北　研二，津田和彦，獅々堀正幹：情報検索アルゴリズム，共立出版 (2002).
15) LAPACK. http://www.netlib.org/lapack
16) NAG 数値計算ライブラリ. http://www.nag-j.co.jp
17) 中川　徹，小柳義夫：UP 応用数学選書 7 最小二乗法による実験データ解析，東京大学出版 (1999).
18) 中村佳正（編）：可積分系の応用数理，裳華房 (2000).

19) 中村佳正，岩﨑雅史：シフトつき可積分特異値分解アルゴリズムについて，日本応用数理学会 2004 年度年会講演予稿集，pp.400–401 (2004).
20) 大津展之，栗田多喜夫，関田 巌：行動計量学シリーズ 12 パターン認識—理論と応用，朝倉書店 (1996).
21) Parlett, B.N. and Marques, O.A.: An Implementation of the dqds Algorithm (Positive Case), *Proc. of the International Workshop on Accurate Solution of Eigenvalue Problems* (University Park, PA, 1998), *Lin. Alg. Appl.*, 309, No.1-3, pp.217–259 (2000).
22) Rutishauser, H.: Der Quotienten-Differenzen-Algorithmus, *Z. Angre. Math. Mech.*, Vol.5, pp.233–251 (1954).（ドイツ語）
23) 寺本 英：数理生態学，朝倉書店 (1997).

(平成 17 年 1 月 25 日受付)
(平成 17 年 4 月 25 日採録)



高田 雅美（正会員）

昭和 52 年生．平成 16 年奈良女子大学大学院人間文化研究科複合領域科学専攻修了．博士（理学）を同大学より取得．平成 16 年独立行政法人科学技術振興機構戦略的創造研究推進事業の委嘱研究員として，京都大学大学院情報学研究科数理工学専攻数理解析分野にて従事．数値計算ライブラリの開発，分散メモリ環境を対象とする並列プログラムの開発に関する研究に従事．

岩﨑 雅史（正会員）

昭和 49 年生．平成 16 年京都大学大学院情報学研究科博士後期課程修了．博士（情報学）を同大学より取得．平成 16 年独立行政法人科学技術振興機構戦略的創造研究推進事業の委嘱研究員として，京都大学大学院情報学研究科数理工学専攻数理解析分野にて従事．線形数値計算，微分方程式の漸近解析に関する研究に従事．日本数学会，日本応用数理学会各会員．

木村 欣司

昭和 51 年生．平成 16 年神戸大学大学院自然科学研究科博士課程修了．博士（理学）を同大学より取得．平成 17 年独立行政法人科学技術振興機構戦略的創造研究推進事業の委託研究員として，立教大学理学部数学科にて従事．計算機代数のアルゴリズム開発ならびに実装に従事．日本応用数理学会，日本物理学会，日本数式処理学会各会員．

中村 佳正（正会員）

昭和 30 年生．昭和 58 年京都大学大学院工学研究科博士課程修了．工学博士を同大学より取得．平成 6 年同志社大学工学部教授．平成 8 年大阪大学大学院基礎工学研究科教授．平成 13 年より京都大学大学院情報学研究科教授．応用可積分系，計算数学の研究に従事．日本応用数理学会，日本数学会，AMS，SIAM 各会員．