2005－MPS－55（7）
2005／6／28

An Evaluation of Singular Value Computation by the Discrete Lotka–Volterra System

Masami Takata * Kinji Kimura † Masashi Iwasaki * and Yoshimasa Nakamura *
takata@amp.i.kyoto-u.ac.jp
* PRESTO JST, and Graduate School of Informatics in Kyoto University
† CREST JST, and Rikkyo University

Abstract

The *mdLVs* (modified discrete Lotka–Volterra with shift) scheme for computing singular values is evaluated. Convergence and stability of the *mdLVs* scheme are guaranteed. It is shown that numerical accuracy of the scheme is equal to or higher than that of the existing routines of LAPACK.

1 Introduction

SVD (singular value decomposition) routines are widely used in, for example, data search systems, image processing, and applications for least square problems.

The QR scheme [1, 2, 5, 6, 7] and the qd (quotient–difference) scheme [4, 14, 15] are known as the efficient SVC (singular value computation) algorithms. By introducing shift the convergence of these schemes is extremely accelerated [1].

On the other hand, two of the authors [10] designs a new SVC algorithm by using the integrable dLV (discrete Lotka–Volterra) system. A convergence to SVs (singular values) of the *mdLVs* (modified dLV with shift) scheme is proved in [11]. It is expected that 1) the computational cost of the *mdLVs* is smaller than that of the QRs (QR with shift) scheme, 2) a numerical accuracy is equal to or higher than the dqds (differential qd with shift) scheme.

In this paper, a performance of the *mdLVs* scheme is evaluated with respect to both computational time and numerical accuracy. To this end the *mdLVs* scheme is implemented to a new routine named the *DLVS*.

2 Bidiagonal SVD

A rectangular matrix A is decomposed to a product of suitable orthogonal matrices and a matrix having an upper bidiagonal block B by a sequence of Householder transformations [1, 6], where the SVs of B are congruent with those of A.

We can calculate SVs and SVECs (singular vectors) simultaneously by the QRs scheme [1, 2, 5, 6, 7]. Namely, B is decomposed to a product of orthogonal matrices and a diagonal matrix by the QRs iterations, where column vectors of the orthogonal matrices and diagonal elements of the diagonal matrix give SVECs and SVs, respectively. Once B is decomposed to such a form, an SVD of A is terminated. The QRs scheme generates sufficiently orthogonal SVECs. In the case of large scaled matrices, the QRs scheme often converges slowly and the relative accuracy of the computed SVs becomes worse.

In late 1990', Dhillon and Parlett [3] introduced a scheme for SVECs using a twisted factorization of tridiagonal matrices. In their scheme, an approximation of SVs is computed by some SVC scheme first.

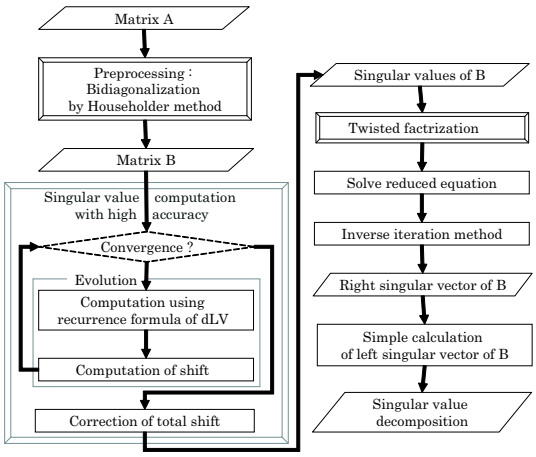

Figure 1: Flowchart of the *I–SVD* scheme.

Then approximationed SVECs are computed by using the SVs through the twisted factorization techniques. Finally, more accurate SVECs are obtained through an inverse iteration one time. However, since errors of SVs spread to all SVECs, it occurs that the orthogonality of SVECs often deteriorates. The computational costs of SVs and SVECs are $O(M^2)$ and $O(M^3)$, respectively [1], where M is the dimension of the bidiagonal matrix B. Hence, for the SVD by the twisted factorization, an SVC with high accuracy should be most important.

As an accurate SVC scheme, the dqds scheme [4, 14, 15] and the *mdLVs* scheme [9, 11] are known.

A maximal error per one iteration is estimated as follows. Let B be $M \times M$ bidiagonal matrix. The errors of the QR, the dqd, and the dLV schemes are $69M^2\varepsilon$ [1], $3M\varepsilon$ [4], and $(2M-1)\varepsilon + (2M-1)\varepsilon'$, respectively, where ε indicates a sufficiently small number and $\varepsilon \gg \varepsilon'$. An error analysis for the SVC schemes with shift, namely, the QRs, dqds and *mdLVs*, can be done similarly.

We develop LAPIS (Linear Algebra Package by Integrable Systems), which includes a new SVD package based on the dLV system. Fig.1 gives a flowchart of LAPIS's library named the *DBDSLV* where the *I–SVD* (Integrable–SVD) scheme is implemented. This includes the *DLVS* where the *mdLVs* scheme [9] is implemented.

Tab.1 shows the relationship between the schemes

Table 1: Schemes and routines.

	scheme	routine
LAPACK	QRs	*DBDSQR*
	dqds	*DLASQ*
LAPIS	*I–SVD*	*DBDSLV*
	mdLVs	*DLVS*

and the routines considered in this paper.

3 *mdLVs* scheme

In order to develop LAPIS, we implement the *DLVS* based on the *mdLVs* scheme.

3.1 SVC based on the dLV system

In the mathematical biology, the LV (Lotka–Volterra) system is known as a fundamental prey–predictor model. In some case, the LV system is a completely integrable dynamical system which has explicit solutions and sufficiently many conservation laws. A time discretization

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

of the LV system is known (cf. [10]). This system also has explicit solution and many conservation laws. Therefore, it is called the integrable discrete LV (dLV) system. Here k ($k = 1, 2, ..., 2M-1$) indicates the k–th species and the discrete time n ($n = 0, 1, 2, ...$) corresponds to an iteration number of the scheme, $u_k^{(n)}$ is the value of u_k at n, and the arbitrary nonzero number $\delta^{(n)}$ is a discrete step–size. Let the initial value $u_k^{(0)}$ be positive. In the case where $\delta^{(n)} > 0$, any substraction and division by zero do not occur in Eq.(1) and $u_k^{(n)}$ is always positive. Consequently, cancelling and numerically unstability do not emerge. Let us note here we do not need treat negative numbers in SVCs.

The boundary condition and the initial condition are

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

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

respectively. Here b_{2i-1} ($i : 1 \le i \le M$) and b_{2i} are diagonal and upper–subdiagonal elements of the $M \times M$ bidiagonal matrix B, respectively. When $n \to \infty$, $u_{2i-1}^{(n)}$ and $u_{2i}^{(n)}$ converge to the square of the i–th SV σ_i and 0, respectively. Thus the dLV system gives rise to a stable scheme for computing SVs [10].

3.2 Speed up by means of the shift

The *mdLVs* scheme, the integrable dLV system with shift, can compute SVs in higher speed. The *mdLVs* scheme is formulated as follows [9, 11].

Let us introduce new elements $w_k^{(n)}$ and $v_k^{(n)}$ by

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

By Eq.(3), the initial $w_k^{(0)}$ is just ${b_k}^2$. The shifted integrable dLV system is defined by adding a shift $S^{(n)}$ to Eq.(1). Namely,

$$w_{2i-1}^{(n+1)} = v_{2i-1}^{(n)} + v_{2i-2}^{(n)} - w_{2i-2}^{(n+1)} - S^{(n)},$$
$$w_{2i}^{(n+1)} = v_{2i-1}^{(n)} v_{2i}^{(n)} / w_{2i-1}^{(n+1)}. \quad (5)$$

In general, the convergence is accelerated by enlarging $S^{(n)}$. However, since the positivity of $u_k^{(n)}$ may be destroyed by a larger $S^{(n)}$, it causes a numerical unstablity. It is proved in [11] that $u_k^{(n)} > 0$ if and only if $0 \le S^{(n)} < \sigma_m^2$ where σ_m is the minimal SV of B. Hence the shift $S^{(n)}$ can be determined by using the Gersgorin [8] or the Johnson [12] bound for estimating σ_m.

3.3 SVC Routine of the dLV system

The *DLASQ* of LAPACK (Linear Algebra PACKage) [13] is an SVC routine (DOUBLE PRECISION) based on the dqds scheme for bidiagonal matrices. When a subdiagonal element is extremely smaller than the diagonal, SPLIT, which divides the matrix to two parts, and a deflation of dimension size are done [14].

The *DLVS* has a dissimilarity to the *DLASQ* routine at each iteration. Each iteration is described as follows.

① $u_k^{(n)}$ is calculated from $w_k^{(n)}$ by Eq.(4).

② $v_k^{(n)}$ is calculated from $u_k^{(n)}$ by Eq.(4).

③ $S^{(n)}$ is calculated.

④ After checking $S^{(n)}$, $w_k^{(n)}$ is calculated.

- In the case of a valid $S^{(n)}$, $w_k^{(n)}$ is calculated from $v_k^{(n)}$ by Eq.(5).
- In other case, $w_k^{(n+1)} = v_k^{(n)}$.

Arrays of the *DLVS* are as follows. In Step①, the array $U = (u_1^{(n)}, u_2^{(n)}, ..., u_{2M-1}^{(n)})$ is calculated from the array $W = (w_1^{(n)}, w_2^{(n)}, ..., w_{2M-1}^{(n)})$, where n represents the iteration number. Since the data at each n does not keep, each array consists of one–dimensional array corresponding to under suffix. In step②, the array $V = (v_1^{(n)}, v_2^{(n)}, ..., v_{2M-1}^{(n)})$ is calculated from U. By using a valid $S^{(n)}$, W is overwritten by V in Step④.

In the loops of Step① and ②, U and V is updated in ascending order of k. For the update of $u_k^{(n)}$, we use $w_k^{(n)}$ and $u_{k-1}^{(n)}$ in Step①, For the update of $v_k^{(n)}$, we need $u_k^{(n)}$ and $u_{k+1}^{(n)}$ in Step②. For the update $w_k^{(n+1)}$ from $w_{k-1}^{(n+1)}$ in Step④, two types of calculations are chosen which is corresponding to whether k is even or odd.

3.4 Worn point for implementation

The number of arrays rather influences the mount of memory. To execute the *DLVS* in computers, the mount of memory should be detained minimum size by an essential array requirement.

In Eq.(4), $u_k^{(n)}$ only uses the update of $v_k^{(n)}$ and $u_{k+1}^{(n)}$ that $v_{k+1}^{(n)}$ needs. Therefore, the necessary $u_k^{(n)}$ is maintained using temporary values (TMP_1, TMP_2).

Fig.2 expresses an adoption of a loop fusion and unrolling in Steps ① and ②, when the SPLIT and the deflation do not occur. The number of substitutions in Steps ① and ② is $2M-1$ and $2M-2$, respectively, where the number of iterations of loop is excepted. Therefore the number of substitutions

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

Figure 2: A loop fusion and unrolling.

Table 2: The performance of each computer.

80bit register (default)		
	C_{P4}	C_I
CPU (Intel)	Pentium4 2.6GHz	Itanium2 1.6GHz
Memory	1GB	8GB
L1 D	8KB	32KB
L1 I	12Kμops	32KB
L2	512KB	256KB
OS	Debian 3.0 (Linux 2.4.24)	RedHut 2.1AS (Linux 2.2.24)
64bit register (default)		
	C_{G5}	C_O
CPU	Power PC G5 2.0GHz	AMD OPTERON 2.4GHz
Memory	3.5GB	2GB
L1 D	32KB	64KB
L1 I	64KB	64KB
L2	512KB	1024KB
OS	Darwin 7.5.0	Fedora Core 2 (Linux 2.6.5)

in Fig.2 is $4M-3$, and the number of judgments of loop becomes to 1/4. Consequently, the computational time decreases.

An access to lower cache and memory frequently occurs, since difference elements need in each iteration for the case of the array U. On the other hand, the temporary value is kept in registers or L1 D during loop calculations. Consequently, in the case where the number of substitutions is equal to each other, the routine of Fig.2 is faster than the others.

A value consists of 64*bit* in usual memory and cache. On the other hand, in expand registers such as Intel Pentium Series etc., a value consists of 80*bit*. In the case of 'g77 –O3', a routine prior uses register's value as the needed value in loop calculations. Hence, the routine becomes to have a high accuracy, since the temporary values can be kept in register in the case of the expand ones. However, when the value of the 80*bit* register is copied to cache, a round–off error occurs because of 64*bit* cache.

4 Numerical experiments

In this section, we inspect the *mdLVs* scheme in a comparison with the QRs and dqds schemes. We use the *DLASQ* and the *DBDSQR* without SVEC calculations as the dqds and QRs scheme, respectively. While we prepare the *DLVS* as an implementation of the *mdLVs* scheme. Here the parameter $\delta^{(n)}$ is set constant as $\delta^{(n)} = 1$, and the shift $S^{(n)}$ is given by using the Johnson bound. Tab.2 shows the performance of each computer which we use in numerical experiments. We use GNU compiler as a common free compiler and recommended compilers as in Tab.3.

4.1 Accuracy comparison

To get better SVECs, we should compute SVs as accurate as possible.

In our experiments for accuracy comparison, 100 1,000–dimensional bidiagonal matrices are prepared, where 1,000 SVs of each matrix are randomized on the interval $[1, 500]$. Tab.4 shows the result in the case of GNU compiler. In the average, the *DBDSQR* is the worst. The results for other recommended compilers are similar to Tab.4.

Table 3: Compilers in each computer.

	Compiler	Options
C_{P4}	GNU 2.95.4	-O3
	Intel Fortran 8.1	-O3 -axN -xN -prefetch
C_I	GNU 2.96	-O3
	Intel Fortran 8.1	-O3 -tpp2
C_{G5}	GNU 3.4	-O3
	XL Fortran 8.1	-O5
C_O	GNU 3.3.3	-O3
	PGI Fortran 5.2	-fastsse -Mipa=fast,inline

Table 4: Summation of relative errors. (using GNU compiler)

	Max	Min	Average
C_{P4}			
DBDSQR	1.26E-12	7.08E-13	9.48E-13
DLASQ	1.27E-12	1.90E-13	4.56E-13
DLVS	2.51E-13	1.43E-13	1.85E-13
C_I			
DBDSQR	1.68E-12	9.15E-13	1.27E-12
DLASQ	2.26E-13	1.60E-13	1.87E-13
DLVS	1.05E-12	3.28E-13	5.32E-13
C_{G5}			
DBDSQR	1.70E-12	9.51E-13	1.31E-12
DLASQ	2.59E-13	1.59E-13	1.91E-13
DLVS	9.94E-13	3.26E-13	5.39E-13
C_O			
DBDSQR	1.83E-12	1.05E-12	1.41E-12
DLASQ	1.05E-12	2.92E-13	5.64E-13
DLVS	1.05E-12	3.28E-13	5.32E-13

In C_{P4}, though the number of iterations is almost the same, errors of the *DLVS* is smaller than those of the *DLASQ*. In the case where the bit number in each register is changed to 64*bit* by FPU head–file, the average relative errors of the *DBDSQR*, *DLASQ*, and *DLVS* are $1.41E-12$, $5.41E-13$, and $5.21E-13$, respectively. Hence, the *DLVS* becomes to have a higher relative accuracy because of 80*bit* registers.

In C_O, errors of the *DLASQ* and *DLVS* are almost in the same level. To confirm an influence of 80*bit* registers in C_O, we experiment the case of the compiler option '–mfpmath', which make use the 387 floating point coprocessor. The resulting average errors of the *DBDSQR*, *DLASQ*, and *DLVS* are $1.28E-12$, $5.65E-13$, and $4.98E-13$, respectively. Though the improvement of accuracy is smaller than Intel's CPU case, SVs become higher accurate by using the 387 floating point coprocessor.

In C_I and C_{G5}, the *DLASQ* seems better. It is guessed that round–off errors become smaller, since the intermediate result is stored in the fourfold precision in the fused multiply–add. Since the *DLVS* include more divisions than the *DLASQ*, the *DLVS* does not make use of this property of the CPUs effectively.

Fig.3 and Fig.4 illustrate the relative and absolute errors of computed SVs in terms of the *DLASQ* and *DLVS*. In the figures, the horizontal axis indicates the SV number, namely, the descending order of SVs, and the vertical line represents the size of errors. The absolute errors of large SVs computed by the *DLASQ* are larger than those by the *DLVS*, especially, in C_{P4} and C_O. It gives rise to a large error of the resulting SVECs. On the other hand, the absolute errors by the *DLVS* are within a constant rate overall. Hence, the *DLVS* is better than the *DLASQ* in accuracy.

4.2 Computational time comparison

In the experiments for computational time, 100 10,000–dimensional bidiagonal matrices are prepared whose diagonal and bi–diagonal elements are randomized on the interval $[1, 100]$.

Tab.5 shows the computational time in the case of GNU compiler. The computational time of the *DLVS* is duplication or treble of that of the *DLASQ* in all

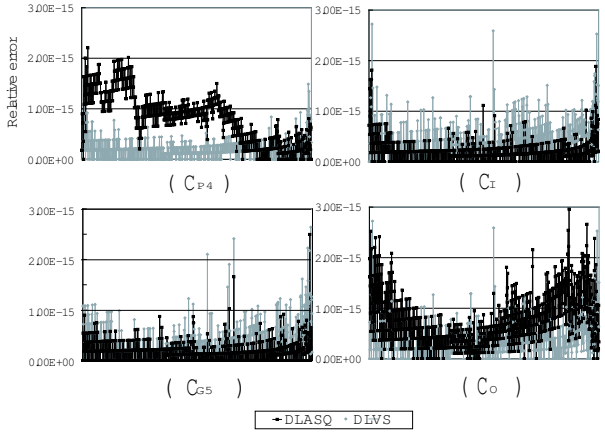

Figure 3: Relative errors of SVs.

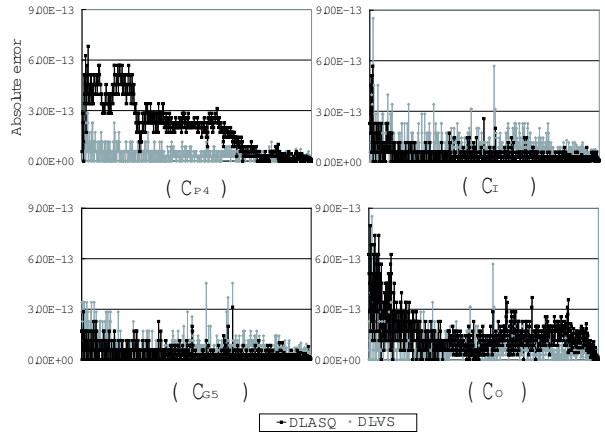

Figure 4: Absolute errors of SVs.

CPUs. This is because the recurrence relations of the *mdLVs* scheme are more complex and have more divisions than those of the dqds scheme. Since all elements of the matrix are needed to compute the Johnson bound, an estimation of shift by this bound takes 40% of the computational time of the *DLVS*. Through the shift $S^{(n)}$ in the *mdLVs* scheme is safely determined by using the bound, this bound causes more computational cost of the scheme. On the other hand, in the *DLASQ*, shift is roughly given by using only few elements of the matrix. This is the second reason why *DLASQ* is faster than the *DLVS*. However, the *DLASQ* needs an exceptional processing to avoid division by zero. There is a big difference between the design principles of the *DLVS* and *DLASQ*.

The *DBDSQR* is faster than the *DLVS* in C_{G5} only. It is caused that the rate of inner product calculation takes a large part in the QRs scheme.

Tab.6 shows the computational time using recommended compilers. In the case of recommended compilers, the computational time of the *DLVS* is 1.5 times of that of the *DLASQ* in most CPUs. Remark that the computational time of the Johnson bound can be reduced under the effect of pipeline. We note that the *DLVS* is the fastest in C_I case. The variance of computational times of the *DBDSQR* and *DLASQ* is larger than that of the *DLVS*. Indeed, the *DBDSQR* is extremely slow in C_I.

5 Conclusion

The QRs scheme has a high reliability, however, it takes a relatively large computational cost, and has

Table 5: Computational time for 10,000–dimensional matrices. (using GNU compiler) [sec.]

	Max	Min	Average
C_{P4}			
DBDSQR	19.59	16.33	18.22
DLASQ	5.05	3.93	4.73
DLVS	10.57	9.49	9.62
C_I			
DBDSQR	31.60	25.75	28.85
DLASQ	9.13	7.10	8.55
DLVS	21.24	17.50	19.96
C_{G5}			
DBDSQR	13.71	11.44	12.53
DLASQ	4.45	3.66	4.21
DLVS	12,38	9.38	11.38
C_O			
DBDSQR	10.18	7.67	8.94
DLASQ	2.84	2.20	2.65
DLVS	6.29	5.07	5.80

Table 6: Computational time for 10,000–dimensional matrices. (using recommended compilers) [sec.]

	Max	Min	Average
C_{P4}			
DBDSQR	33.74	27.67	30.87
DLASQ	5.55	4.31	5.17
DLVS	8.83	7.22	8.21
C_I			
DBDSQR	200.14	164.16	188.57
DLASQ	7.31	6.05	6.82
DLVS	6.72	5.54	6.32
C_{G5}			
DBDSQR	14.44	4.76	11.00
DLASQ	5.31	1.83	4.21
DLVS	7.46	2.53	5.75
C_O			
DBDSQR	9.41	7.6	8.65
DLASQ	2.72	2.12	2.54
DLVS	6.01	4.84	5.54

a low accuracy. The *DBDSQR* of LAPACK has been nevertheless widely adopted, for example, in MATLAB, Mathematica. The dqds scheme is a high–speed scheme for SVC with a good accuracy, however, a safe choice of shift has not been known. While a convergence and stability of the *mdLVs* scheme is theoretically guaranteed.

In this paper, we discuss a practicality of the *mdLVs* scheme for SVC through the *DLVS*. On accuracy, the *DLVS* is more accurate than the *DLASQ* of LAPACK where the dqds scheme is implemented, except for fused multiply–add type CPUs. The *DLVS* is rather accurate than the DBDSQR.

In the case of GNU compiler, the computational time of the *DLVS* is duplication or treble of that of the *DLASQ* and 0.6 times to the *DBDSQR*. In the case of recommended compilers, the computational time of the *DLVS* is 1.5 times of that of the *DLASQ*. In Itanium2, the *DLVS* is faster than two.

For the SVD by the twisted factorization, a higher accuracy of the computed SVs is most important. A rich practicality of the *mdLVs* scheme is then verified in this paper.

References

[1] Demmel, J.: Applied Numerical Linear Algebra, SIAM, Philadelphia (1997).

[2] Demmel, J., and Kahan, W.: *Accurate singular values of bidiagonal matrics*, SIAM J. Sci. Sta. Comput., Vol.67, pp. 191-229 (1994).

[3] 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).

[4] Fernando, K. V., and Parlett, B. N.: *Accurate singular values and differential qd algorithms*, Numer. Math., Vol.67, pp. 191-229 (1994).

[5] Francis, J. G. F.: *The QR transformation a unitary analogue to the LR transformation–part 1*, Computer J., Vol.4, pp. 265-271 (1961).

[6] Golub, G., and Kahan, W.: *Calculating the singular values and pseudo-inverse of a matrix*, J. SIAM, Numer. Anal., Ser. B 2, pp. 205-224 (1965).

[7] Golub, G., and Reinsch, C.: *Singular value decomposition and least squares solutions*, Numer. Math., Vol.14, pp. 403-420 (1970).

[8] Henrici, P.: Applied and Computational Complex Analysis Volume I, Wiley-Interscience Publishing, NewYork (1988).

[9] Iwasaki, M.: *Studies of Singular Value Decomposition in Terms of Integrable Systems*, Doctor Thesis, Kyoto University (2004).

[10] 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).

[11] Iwasaki, M., and Nakamura, Y.: *Accurate computation of singular values in terms of the shifted integrable algorithm*, (2005). (in preparation)

[12] Johnson, C. R.: *A Gersgorin–type lower bound for the smallest singular value*, Lin. Alg. Appl., Vol.112, pp. 1-7 (1989).

[13] LAPACK, *http://www.netlib.org/lapack*

[14] 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).

[15] Rutishauser, H.: *Der Quotienten-Differenzen-Algorithmus*, Z. Angre. Math. Mech., Vol.5, pp. 233-251 (1954). (in Germany)