On the Efficient Update of the Singular Value Decomposition Subject to Rank-One Modifications

Peter Stange

Braunschweig :
Institut Computational Mathematics, AG Numerik,
2011

On the Efficient Update of the Singular Value Decomposition Subject to Rank-One Modifications*

Peter Stange

December 9, 2009

Abstract

In this paper we present an efficient method for updating the *singular value decomposition* (SVD) subject to a *rank-one modification*. The updated SVD can be characterized by two problems involving symmetric matrices. The singular values corresponding to these symmetric problems are computed by solving a *secular equation*. The secular equation can be solved reliably and efficiently with standard software.

The singular vectors can be updated efficiently with a few matrix-matrix products. The computational effort to compute the matrix-matrix products can be considerably decreased by exploiting that some matrices are of *Cauchy-type*. We analyze several methods which exploit this structure. The computational complexity of the proposed approach is $\mathcal{O}(n^2 \log^2 n)$.

1 Introduction

The singular value decomposition is a powerful method which is used in matrix approximations, least squares fitting of data, or determining the rank of a matrix. A lot of applications, e.g., in signal processing, mechanical engineering, or statistics are dealing with the SVD. In some cases it is necessary to compute a sequence of these factorizations. In this paper we are especially interested in the case of rank-one modifications that affect the factorized matrix. Because of the high computational costs we want to avoid a new refactorization for each of these updates.

Problem Definition
Given are the orthogonal matrices $U \in \mathbb{R}^{m\times m}$, $V \in \mathbb{R}^{n\times n}$ and the diagonal matrix $\Sigma \in \mathbb{R}^{m\times n}$ such that

$$A = U\Sigma V^T$$

is the singular value decomposition of $A \in \mathbb{R}^{m\times n}$. The singular values of A are the ordered, diagonal entries σ_{ii} of Σ, $1 \leq i \leq r \leq m$. r denotes the rank of A. We assume that $m \leq n$. Our objective is to determine the matrices $U_+ \in \mathbb{R}^{m\times m}$, $V_+ \in \mathbb{R}^{n\times n}$ and the diagonal matrix $\Sigma_+ \in \mathbb{R}^{m\times n}$ of the singular value decomposition of A_+. Thereby A_+ is defined by a rank-one modification to A

$$\begin{aligned} A_+ &= A + ab^T \\ &= U_+\Sigma_+V_+^T. \end{aligned} \tag{1}$$

The vectors $a \in \mathbb{R}^m$ and $b \in \mathbb{R}^n$ generate the perturbation to A of rank-one. In order to compute the SVD of A_+ we want to avoid a new refactorization which would lead to a computation of cubic complexity. Instead we will update the matrices U, Σ and V directly subject to the rank-one modification of A. This approach reduces the computational costs significantly. We will show an algorithm that allows us to do the updating procedure in $\mathcal{O}(n^2 \log^2 n)$.

*This research has been supported by the DAAD project D/07/13360.

This problem is related to appending a row or a column to the original matrix A. E.g., in [2], an $\mathcal{O}(n^3)$ algorithm for appending or deleting a row to A is described. This update can be improved [9] to $\mathcal{O}((m+n)\min(m,n)\log_2^2\epsilon)$ or a $n \times m$-matrix if single precision is accurate enough or quadruple precision is available. There are more papers which are closely related to our problem, e.g., [6] where a SVD updating algorithm for diagonal matrices is presented, [12] which gives an $\mathcal{O}(n^3)$ algorithm for the rank-one modification of the symmetric eigenproblem or [8] which is dealing with the same problem. We will introduce a new algorithm that is based on the available work. It will give some new perspectives for updating the rectangular SVD by adding a rank-one term in an efficient and stable way. In contrast to the existing algorithms we will use $\mathcal{H}$-matrix approximations based on exponential sums to speed up the evaluation of the singular vectors which is most time consuming part in the updating process. In addition we will use Jacobi's idea to improve the accuracy of our updated matrices. For these improvements we will use some work that was done for solving Trummers problem [3], [14], and for matrix approximation [10], [7], [1].

This paper is organized as follows. In section 2 we will present a symmetric characterization of problem (1). This will lead to an efficient updating algorithm. In section 3 we will briefly show how to compute the new singular values. For computing these values we will use the secular equation for updating Σ. In section 4 we will analyze the structure of the eigenvectors of the symmetric characterization and we will exploit a special Cauchy-matrix structure for evaluating the new singular vectors of (1). In section 5 we will show how to do an accuracy refinement. This becomes necessary because of the squaring in section 2. We will sum up our algorithm and give a complexity overview in section 5. Finally, some numerical examples are given in section 6.

2 Symmetric Characterization

Our first task is to reformulate the general, rectangular updating problem (1). We would like to transform it in such a way that it offers the opportunity to perform efficient computations. Therefore we will use a symmetric characterization that we obtain by multiplying A_+ with its transpose. Of course, due to this squaring we will loose the half of the accuracy for the moment. In section 4 we will describe how to come back to machine precision. In a factorized representation we can write

$$\begin{aligned} A_+A_+^T &= U_+\Sigma_+ \overbrace{V_+^TV_+}^{I}\Sigma_+^TU_+^T && (2)\\ &= (U\Sigma V^T + ab^T)(V\Sigma^TU^T + ba^T) \\ &= U\Sigma\Sigma^TU^T + \tilde{b}a^T + a\tilde{b}^T + \beta aa^T, && (3)\end{aligned}$$

where $\tilde{b} = U\Sigma V^Tb$ and $\beta = b^Tb$. This product leads us to a symmetric problem where V is eliminated. The three rank-one perturbations in (3) can be summarized to two single rank-one updates as follows

$$\begin{aligned} U_+\underbrace{\Sigma_+\Sigma_+^T}_{D_+}U_+^T &= U\underbrace{\Sigma\Sigma^T}_{D}U^T + \begin{bmatrix}a & \tilde{b}\end{bmatrix}\begin{bmatrix}\beta & 1\\ 1 & 0\end{bmatrix}\begin{bmatrix}a^T\\ \tilde{b}^T\end{bmatrix} && (4)\\ &= \begin{bmatrix}a & \tilde{b}\end{bmatrix}Q\begin{bmatrix}\rho_1 & 0\\ 0 & \rho_2\end{bmatrix}Q^T\begin{bmatrix}a^T\\ \tilde{b}^T\end{bmatrix} \\ &= \underbrace{UDU^T + \rho_1a_1a_1^T}_{\tilde{U}\tilde{D}\tilde{U}^T} + \rho_2b_1b_1^T, && (5)\end{aligned}$$

where $Q\begin{bmatrix}\rho_1 & 0\\ 0 & \rho_2\end{bmatrix}Q^T$ is the Schur-decomposition of $\begin{bmatrix}\beta & 1\\ 1 & 0\end{bmatrix}$.

Performing these computations we obtain a symmetric rank-two problem such that equation (5) is suitable for an efficient computation of the matrices U_+ and Σ_+.

For the evaluation of V_+ the matrix product (2) has to be computed similarly. This means that

$$\begin{aligned} A_+^T A_+ &= V_+ \Sigma_+^T \Sigma_+ V_+^T \\ &= (V\Sigma^T U^T + ba^T)(U\Sigma V^T + ab^T) \\ &= V\Sigma^T \Sigma V^T + \tilde{a}b^T + b\tilde{a}^T + \alpha bb^T, \end{aligned} \tag{6}$$

where $\tilde{a} = V\Sigma^T U^T a$ and $\alpha = a^T a$. Performing similar steps according to (4)-(5) we obtain

$$V_+^T \Sigma_+^T \Sigma_+ V_+ = V^T \Sigma^T \Sigma V + \rho_3 a_2 a_2^T + \rho_4 b_2 b_2^T. \tag{7}$$

Equation (7) has the same properties as equation (5) and will be used for updating the matrix V.

Using these introduced symmetric characterizations (3) and (7) of the initial problem (1), two single symmetric rank-one updates have to be performed for the computation of U_+ and another two ones for the computation of V_+. An efficient way to deal with these low-rank modifications will be illustrated for one of the updates. That means that all the following computations have to be done for each single rank-one perturbation in (5) and (7), respectively.

To simplify the notation we will describe our algorithm for one of these low-rank corrections

$$\tilde{U}\tilde{D}\tilde{U}^T = UDU^T + \rho_1 a_1 a_1^T. \tag{8}$$

Rewriting (8) leads to

$$\tilde{U}\tilde{D}\tilde{U}^T = U(\underbrace{D + \rho_1 \bar{a}\bar{a}^T}_{B})U^T, \tag{9}$$

where $U\bar{a} = a_1$. Matrix B can be written as the following Schur decomposition

$$\begin{aligned} B &:= D + \rho_1 \bar{a}\bar{a}^T \\ &= \tilde{C}\tilde{D}\tilde{C}^T \end{aligned} \tag{10}$$

where $\tilde{C}$ is orthogonal. Including (10) into (9) leads to

$$\tilde{U}\tilde{D}\tilde{U}^T = \underbrace{U(\tilde{C}}_{\tilde{U}}\tilde{D}\underbrace{\tilde{C}^T)U^T}_{\tilde{U}^T}. \tag{11}$$

Obviously updating the eigenvalue matrix D is equivalent to the computation of the eigenvalues of B. Furthermore the updated matrix $\tilde{U}$ equals to the matrix-matrix product $U\tilde{C}$. This implies that solving the updating problems (5) and (7) consists of three major tasks:

(i) Solve the diagonal eigenvalue problem (9) for each symmetric rank-one modification.

(ii) Exploit the structure of $\tilde{C}$ to perform an efficient matrix-matrix multiplication $\tilde{U} = U\tilde{C}$ for each symmetric rank-one update.

(iii) Perform some accuracy refinement to ensure machine precision in the final result.

In the following sections these points will be described in detail.

3 Updating the Singular Values

As it has been shown before, updating the matrix Σ, cf.(1), means updating the singular values of A. Therefore we have to perform two symmetric rank-one updates for computing the singular vectors U_+ and also for V_+ as described in (5) and (7). For solving this task we will use the secular equation. For this purpose a convenient algorithm is given in [12]. Continuing from (9) we will briefly describe this method for updating

$$B := \tilde{C}\tilde{D}\tilde{C}^T = D + \rho_1 \bar{a}\bar{a}^T.$$

Let us suppose that $\text{rank}(D) = n$ and $\|\bar{a}\| = 1$. This is no restriction because we could rescale the factor ρ_1. Further, the old eigenvalues are the diagonal entries of D and we denote them by λ_i where $1 \leq i \leq n$. The i-th eigenvalue of B is given by $\mu_i = \lambda_i + \rho_1 \tilde{\mu}_i$ where $\sum_{i=1}^{n} \tilde{\mu}_i = 1$ and $0 \leq \tilde{\mu}_i \leq 1$. That means that we can get the following ordering for the new and the old eigenvalues [16]

$$\begin{aligned} &\lambda_1 \leq \mu_1 \leq \lambda_2 \leq \cdots \leq \lambda_n \leq \mu_n, \quad \text{if} \quad \rho > 0, \quad \text{and} \\ &\mu_1 \leq \lambda_1 \leq \mu_2 \leq \cdots \leq \mu_n \leq \lambda_n, \quad \text{if} \quad \rho < 0. \end{aligned} \tag{12}$$

If all λ_i are distinct and all entries of $\bar{a}$ are nonzero then the old eigenvalues strictly separate the new ones. As written in [12] the updating algorithm proceeds as follows.

In a first step it could be possible to reduce the initial dimension of the problem. This process is called the deflation of problem (10). It is possible if one of the following cases occur:

(i) If $\bar{a}_i = 0$ for some i. Then the corresponding eigenvalues do not change, i.e., $\mu_i = \lambda_i$.

(ii) If $|\bar{a}_i| = 1$ for some i. Then $\mu_i = \lambda_i + \rho_1$ and all other eigenvalues λ_j, where $j \neq i$, remain unchanged.

(iii) In the case that B has an eigenvalue with the multiplicity $l > 1$. Then the dimension can be reduced by $l - 1$. Details how to proceed in this case are given in [12].

In our numerical computation we have to choose a small threshold γ to validate if the conditions (i) and (ii) are true, e.g., we have to check if $|\bar{a}_i| \leq \gamma$ and $1 - \gamma \leq |\bar{a}_i| \leq 1$ respectively. In these cases the corresponding eigenvalues do not change numerically. In the following we assume for simplicity that deflation is already done and we end up with a problem of dimension $n \times n$. The eigenvalues μ_i of B can be computed as the zeros of the secular equation

$$w(t) = 1 + \rho \sum_{j=1}^{n} \frac{\bar{a}_j^2}{\lambda_j - t}. \tag{13}$$

Therefore a method which is based on a local approximation of partial sums of $w(t)$ via simple rational functions is introduced in [12]. Using this algorithm for the computation of the i-th eigenvalue of B means to do the following, briefly described steps:

(i) Reformulate (13) to

$$w_i(\tilde{\mu}) = 1 + \sum_{j=1}^{n} \frac{\bar{a}_j^2}{\delta_j - \tilde{\mu}}, \tag{14}$$

where $\delta_j = \frac{\lambda_j - \lambda_i}{\rho_1}$.

(ii) Split (14) into the sums

$$\psi(t) = \sum_{j=1}^{i} \frac{\bar{a}_j^2}{\delta_j - t} \text{ and } \phi(t) = \sum_{j=i+1}^{n} \frac{\bar{a}_j^2}{\delta_j - t}. \tag{15}$$

(iii) Choose an initial approximation t_1 to $\tilde{\mu}_i$ such that $0 < t_1 < \tilde{\mu}_i$.

(iv) Define interpolating functions for (15):

$$\psi(t_1) = \frac{p}{q - t_1}, \quad \psi(t_1) = r + \frac{s}{\delta_{i+1} - t_1},$$

where $p = \frac{\psi^2(t_1)}{\psi'(t_1)}$, $q = t_1 + \frac{\psi(t_1)}{\psi'(t_1)}$, $r = \phi(t_1) - (\delta_{i+1} - t_1)\phi'(t_1)$ and $s = (\delta_{i+1} - t_1)^2 \phi'(t_1)$.

(v) Compute the new approximation t_2 to $\tilde{\mu}_i$ by solving

$$\frac{-p}{q - t_2} = 1 + r + \frac{s}{\delta_{i+1} - t_2}.$$

(vi) Repeat (iv)-(v) until $|\tilde{\mu}_i^{(l+1)} - \tilde{\mu}_i^{(l)}| < \eta|\tilde{\mu}_i^{(l)}|$, where η is the relative machine precision. Alternative stopping criteria are discussed in [12].

(vii) Compute the new eigenvalue by $\mu_i = \lambda_i + \rho_1\tilde{\mu}_i$.

More details to this algorithm are given in [12]. Furthermore it is shown that this method has a quadratic rate of convergence.

Once we have evaluated the new eigenvalues we obtain the diagonal entries of Σ_+ by extracting the square root of μ_i. Here we want to remind the reader that this procedure has to be done for each of the four single rank-one updates in (5) and (7). For each of these eigenvalue updates we have to perform the corresponding eigenvector modification. We will describe in section 3 how to do this in an efficient way.

4 Updating the Singular Vectors

Updating the matrices U and V means to compute the new singular vectors of A_+. As described in (5) and (7), each of the factors U and V underlies a rank-two modification. In the following we will explain how to perform these updates efficiently. As it is shown in (11), for each of these rank-one modifications we have to compute a matrix-matrix product. The question is how to avoid the cubic costs for performing these multiplications. For this we will firstly derive how the matrix $\tilde{C}$ looks like. Then we will exploit the special structure of $\tilde{C}$ to do efficient multiplications.

4.1 Computation of the Cauchy matrix

Problem (11) can be reformulated as the following Sylvester equation

$$D\tilde{C} - \tilde{C}\tilde{D} = -\rho_1\bar{a}\bar{a}^T\tilde{C}. \tag{16}$$

It is known that the solution $\tilde{C}$ of (16) is given by

$$\tilde{C} = \rho_1 \begin{bmatrix} \bar{a}_1 & & \\ & \ddots & \\ & & \bar{a}_n \end{bmatrix} \begin{bmatrix} (\mu_1 - \lambda_1)^{-1} & \cdots & (\mu_n - \lambda_1)^{-1} \\ \vdots & & \vdots \\ (\mu_1 - \lambda_n)^{-1} & \cdots & (\mu_n - \lambda_n)^{-1} \end{bmatrix} \begin{bmatrix} \bar{a}^T c_1 & & \\ & \ddots & \\ & & \bar{a}^T c_n \end{bmatrix} \tag{17}$$

where $\tilde{C} = \begin{bmatrix} c_1 & \cdots & c_n \end{bmatrix}$ and $c_i \in \mathbb{R}^n$ are the columns of $\tilde{C}$. As in section 3, λ_i and μ_i are the eigenvalues of D and $\tilde{D}$. Because of (16) the right hand side of (17) is not independent of $\tilde{C}$. Nevertheless we will use (17) to compute $\tilde{C}$ column-wise. Each column c_i is defined by the following eigenvector problem

$$c_i = \rho_1 \begin{bmatrix} \frac{\bar{a}_1}{\mu_i - \lambda_1} \\ \vdots \\ \frac{\bar{a}_n}{\mu_i - \lambda_n} \end{bmatrix} \bar{a}^T c_i. \tag{18}$$

The solution c_i of (18) is

$$c_i = \alpha \begin{bmatrix} \frac{\bar{a}_1}{\lambda_1 - \mu_i} & \cdots & \frac{\bar{a}_n}{\lambda_n - \mu_i} \end{bmatrix}^T \tag{19}$$

which can be proved by inserting c_i into (18). For all elements $c_i = \begin{bmatrix} c_{i1} & \cdots & c_{ij} & \cdots & c_{in} \end{bmatrix}^T$ the following equations hold

$$\begin{aligned} \alpha \frac{\bar{a}_j}{\lambda_j - \mu_i} &= \alpha \rho_1 \sum_{k=1}^{n} \frac{\bar{a}_k^2 \bar{a}_j}{(\lambda_j - \mu_i)(\mu_i - \lambda_k)} \\ 0 &= \underbrace{\alpha \left(1 + \rho_1 \sum_{k=1}^{n} \frac{\bar{a}_k^2}{(\lambda_k - \mu_i)} \right)}_{=0} \end{aligned} \tag{20}$$

The term inside the bracket defines the secular equation that corresponds to the symmetric updating problem (10). It follows that this term equals to zero for λ_k, μ_i being the eigenvalues of D and $\tilde{D}$. As a consequence of (20) we have a free choice of the scalar α which we will use to scale each column of $\tilde{C}$ to get an orthogonal matrix. Due to the preceding computations the eigenvector matrix of problem (10) is given as follows

$$\tilde{C} = \underbrace{\begin{bmatrix} \bar{a}_1 & & \\ & \ddots & \\ & & \bar{a}_n \end{bmatrix} \overbrace{\begin{bmatrix} \frac{1}{\lambda_1 - \mu_1} & \cdots & \frac{1}{\lambda_1 - \mu_n} \\ \vdots & & \vdots \\ \frac{1}{\lambda_n - \mu_1} & \cdots & \frac{1}{\lambda_n - \mu_n} \end{bmatrix}}^{C}}_{\hat{C} = \begin{bmatrix} c_1 & \cdots & c_n \end{bmatrix}} \begin{bmatrix} |c_1| & & \\ & \ddots & \\ & & |c_n| \end{bmatrix}^{-1}, \tag{21}$$

Definition (21) shows that the structure of $\tilde{C}$ is similar to a Cauchy-matrix. More precisely, $\tilde{C}$ is given by the Cauchy-matrix C which is diagonally scaled from the left and the right.

4.2 Efficient Multiplications

We will use the previous result to compute the matrix product $\tilde{U} = U\tilde{C}$ with a complexity less than $\mathcal{O}(n^3)$. This computation of the eigenvectors $\tilde{U}$ consists of these steps:

(i) Evaluate the product $U_1 = U * [\mathrm{diag}(a_i)]_{1 \le i \le n}$.

(ii) Compute the product $U_2 = U_1 \tilde{C} = \begin{bmatrix} u_1^T & \cdots & u_n^T \end{bmatrix}^T C$.

(iii) Scale U_2 column wise to obtain $\tilde{U}$.

Performing (i) and (iii) only needs an effort of quadratic complexity. The key problem is the evaluation in step (ii) where the general matrix multiplication would cause a cubic complexity. The problem of the efficient multiplication by Cauchy matrices with vectors is known as Trummer's problem [5]. Step (ii) is equivalent to solving Trummer's problem n times. Several algorithms dealing with this task have been published, e.g., [3], [4], [14]. In the following we will specify two of these methods. Further we will explain how they can be adapted to our problem.

4.3 Method based on Polynomial Interpolation and FFT

One of the first papers dealing with the problem of multiplying a Cauchy matrix with a vector efficiently was [3]. In this work an $\mathcal{O}(n \log^2 n)$ algorithm for this task is presented. This approach is based on the following idea.

The problem of multiplying the Cauchy-matrix C with a vector u, as defined in (21) is equivalent to the evaluation of the function

$$f(x) = \sum_{j=1}^{n} \frac{u_j}{x - \mu_j}$$

at the points λ_i, where $1 \leq i \leq k$. This function can be expressed as a ratio of two polynomials

$$f(x) = \frac{h(x)}{g(x)} = \frac{h(x)}{\prod_{j=1}^{n}(x-\mu_j)},$$

where $h(x)$ is determined by

$$h(x) = g(x)\sum_{j=1}^{n}\frac{u_j}{x-\mu_j}. \tag{22}$$

By inserting $x = \mu_j$ in (22) we derive $h(\mu_i) = u_i g'(\mu_i)$. That means that $h(x)$ is the interpolation polynomial for the points $(\mu_i, u_i g'(\mu_i))$. Using these facts, for the multiplication $v = Cu$ the following algorithm was suggested in [3]:

Algorithm I

(i) Compute the coefficients of $g(x)$ using FFT.

(ii) Compute the coefficients of $g'(x)$.

(iii) Evaluate $g(\lambda_i)$, $g'(\lambda_i)$ and $g'(\mu_i)$.

(iv) Compute $h_j = u_j g'(\lambda_j)$.

(v) Find the interpolation polynomial $h(x)$ for the points (μ_j, h_j).

(vi) Compute $v_i = \frac{h(\lambda_i)}{g(\lambda_i)}$.

v_i are the elements of the resulting vector $v = Cu$. Each of the steps (i)-(vi) can be done with a computational complexity of at most $\mathcal{O}(n \log^2 n)$. The drawback of this algorithm is located in step (v). Due to the interpolation the method is unstable for certain distributions of the eigenvalues λ_i and μ_i.

Cauchy matrix Transformation

To avoid the instabilities of the Algorithm I it is possible to transform the Cauchy matrix as described in [14]:

$$C = D^{-1}(\lambda,\mu)D(\lambda,s)C(\lambda,s)D'^{-1}(s)D(s,\mu)C(s,\mu), \tag{23}$$

where the following notation is used

$$\begin{aligned} C(\lambda,s) &= (c_{ij}) = \frac{1}{\lambda_i - s_j}, \quad 1 \leq i,j \leq n, \\ D(\lambda,s) &= \operatorname{diag}(\prod_{j=1}^{n}(\lambda_i - s_j))_{1\leq i\leq n}, \\ D'(s) &= \operatorname{diag}(\prod_{j=1, j\neq i}^{n}(s_i - s_j))_{i=1}^{n}. \end{aligned}$$

The vector $s \in \mathbb{R}^n$ can be chosen arbitrarily. This allows us to choose it in a way that the new Cauchy matrices have better properties with respect to algorithm I. Using transformation (23) for C in our multiplication $v = Cu$ we have to evaluate four products of the type diagonal matrix times vector and two products of the type Cauchy matrix times vector. Unfortunately, this increases the computational effort. Computing the matrices D and D' may be possible in $\mathcal{O}(n^2)$, but it doubles the more costly $\mathcal{O}(n^2 \log^2 n)$ multiplication by Cauchy matrices. Further we have to avoid the introduction of new instabilities due to the computation of $D(\lambda,\mu)^{-1}$ which may have diagonal elements close to zero. The authors suggest to use fast, numerically stable approximation techniques, e.g., [13] or [15] instead of multipoint polynomial evaluation to avoid the latter problem.

4.4 Method based on Matrix Approximations

To avoid the stability problems of algorithm I we suggest to perform a low rank approximation of the Cauchy matrix C. Thereafter an efficient evaluation of the product $v = Cu$ is possible.

In [7] it is shown that Cauchy matrices are suitable for a low rank approximation. Therefore the idea is to construct a separable representation for the function

$$f(x,y) := \frac{1}{x-y} \approx \sum_{j=1}^{k} g_j(x) h_j(y)$$

such that the elements of the Cauchy matrix (21) are approximated by

$$c_{ij} \approx \sum_{j=1}^{k} g_j(\lambda_j) h_j(\mu_i).$$

To achieve a small rank k the sets of eigenvalues $\lambda = \{\lambda_i | i = 1, ..., n\}$ and $\mu = \{\mu_i | i = 1, ..., n\}$ have to be well separated. That means they have to fulfill the admissibility condition which is defined in the context of hierarchical matrices:

$$\min\{\mathrm{diam}(\lambda), \mathrm{diam}(\mu)\} \leq \mathrm{dist}(\lambda, \mu), \tag{24}$$

where

$$\begin{aligned} \mathrm{diam}(\lambda) &= \max(\lambda) - \min(\lambda) \\ \mathrm{diam}(\mu) &= \max(\mu) - \min(\mu) \\ \mathrm{dist}(\lambda,\mu) &= \begin{cases} 0 & \lambda \cap \mu \neq 0 \\ \min(\mu) - \max(\lambda), & \min(\mu) > \max(\lambda) \\ \min(\lambda) - \max(\mu), & \min(\lambda) \geq \max(\mu). \end{cases} \end{aligned}$$

It is possible to weaken condition (24) by introducing a factor $0 < \eta \leq 1$ on the right hand side. On one hand this would lead to greater subsets which is desirable for the following approximation. On the other hand the rank of the approximation could grow. In our application the old and the new eigenvalues strictly interlace with each other, cf. (12). It follows that the initial sets λ and μ never fulfill condition (24). So it is not possible to approximate C by low rank matrix factors at once. Instead we have to subdivide the sets λ and μ until the new subsets fulfill condition (24). Then we can approximate C piecewise by low rank terms. There are different strategies for separating the sets λ and μ. In our algorithm we use geometric bisection.

Polynomial sums

Once the separation is done we are working on two subsets that fulfill condition (24). Then we can approximate the matrix entries c_{ij} of our Cauchy matrix by the following sums

$$c_{ij} \approx \begin{cases} \sum_{l=0}^{k} (\mu_0 - \lambda_j)^{-l-1} (\mu_0 - \mu_i)^l, & \text{if } \mathrm{diam}(\mu) < \mathrm{diam}(\lambda) \\ \sum_{l=0}^{k} (\mu_i - \lambda_0)^{-l-1} (\lambda_j - \lambda_0)^l, & \text{otherwise} \end{cases} \tag{25}$$

where $\lambda_0 = \frac{1}{2}(\min(\lambda) + \max(\lambda))$, $\mu_0 = \frac{1}{2}(\min(\mu) + \max(\mu))$. To ensure a relative approximation accuracy ϵ

$$|\tilde{c}_{ij} - c_{ij}| < \epsilon |c_{ij}|$$

it is necessary that the approximating low rank factors have at least rank

$$k = \lceil \log_3(1/\epsilon) \rceil + 1.$$

Using (25), $C \approx (c_{ij})_{1\leq i\leq n^*, 1\leq j\leq m^*}$ can be written as the product of its low rank factors $C \approx WZ^T$, where n^*, m^* are the dimension of the sub-matrix C and

$$\begin{aligned} W_{il} &= \begin{cases} (\mu_0 - \mu_i)^l, & \text{if } \operatorname{diam}(\mu) < \operatorname{diam}(\lambda) \\ (\mu_i - \lambda_0)^{-l-1}, & \text{otherwise} \end{cases} \\ Z_{jl} &= \begin{cases} (\mu_0 - \lambda_j)^{-l-1}, & \text{if } \operatorname{diam}(\mu) < \operatorname{diam}(\lambda) \\ (\lambda_j - \lambda_0)^l, & \text{otherwise} \end{cases} . \end{aligned}$$

It follows that the problem of multiplying the Cauchy-matrix C with the vector u reduces to the multiplication of u with the low rank factors W and Z for each single block of C. If the matrix C is large then these products can be evaluated much faster than computing $v = Cu$ at once.

Approximation by Exponential Sums

Having a low rank k is the key property for an efficient computation of $v = Cu$ by the previously described approximation technique. There are further possibilities to approximate C beside the sums that are given in (25). An alterative approach is to use exponential sums as follows.

If all elements λ_i of subset λ are larger compared to the elements μ_j of subset μ then the inequalities

$$\begin{aligned} \min(\lambda) - \max(\mu) &\leq \lambda_i - \mu_j \leq \max(\lambda) - \min(\mu) \\ \operatorname{dist}(\lambda, \mu) &\leq \lambda_i - \mu_j \leq \operatorname{diam}(\lambda) + \operatorname{diam}(\mu) + \operatorname{dist}(\lambda, \mu) \\ 1 &\leq \lambda_i - \mu_j \leq 1 + \frac{\operatorname{diam}(\lambda) + \operatorname{diam}(\mu)}{\operatorname{dist}(\lambda, \mu)} \end{aligned} \tag{26}$$

hold, which shows that the differences between the old and the new eigenvalues are located in a relatively small interval. This allows to approximate all the elements $\frac{1}{\lambda_i - \mu_j}$ of the corresponding block in the Cauchy matrix by sums of exponentials.

$$\frac{1}{\lambda_i - \mu_j} = \sum_{l=1}^{k} \alpha_l \mathrm{e}^{-a_l \lambda_i} \mathrm{e}^{-a_l(-\mu_j)} \tag{27}$$

where α_l and a_l are positive real values. For numerical reasons we have to avoid the positive exponent in (27) which occur due to the minus in the denominator. That means instead of $\frac{1}{\lambda_i - \mu_j}$ we approximate $\frac{1}{\underbrace{(\lambda_i - \omega)}_{=\hat{\lambda}>0} + \underbrace{(\omega - \mu_j)}_{=\hat{\mu}>0}}$, where $\omega = \frac{\min(\lambda)+\max(\mu)}{2}$. By adding this zero term, both elements in the fraction are positive. Further we divide the denominator by $\operatorname{dist}(\lambda, \mu)$ to scale down the differences. After these transformations we can use (27) to approximate all the admissible blocks of C by the product of two low rank matrices

$$\begin{aligned} C &= \left(\frac{1}{\lambda_i - \mu_j} \right)_{1\leq i\leq n^*, 1\leq j\leq m^*} \\ &= \frac{1}{\operatorname{dist}(\lambda, \mu)} \begin{bmatrix} \alpha_1 \mathrm{e}^{\frac{-a_1 \hat{\lambda}_1}{\operatorname{dist}(\lambda,\mu)}} & \cdots & \alpha_k \mathrm{e}^{\frac{-a_k \hat{\lambda}_1}{\operatorname{dist}(\lambda,\mu)}} \\ \vdots & & \vdots \\ \alpha_1 \mathrm{e}^{\frac{-a_1 \hat{\lambda}_{n^*}}{\operatorname{dist}(\lambda,\mu)}} & \cdots & \alpha_k \mathrm{e}^{\frac{-a_k \hat{\lambda}_{n^*}}{\operatorname{dist}(\lambda,\mu)}} \end{bmatrix} \begin{bmatrix} \mathrm{e}^{\frac{-a_1 \hat{\mu}_1}{\operatorname{dist}(\lambda,\mu)}} & \cdots & \mathrm{e}^{\frac{-a_1 \hat{\mu}_{m^*}}{\operatorname{dist}(\lambda,\mu)}} \\ \vdots & & \vdots \\ \mathrm{e}^{\frac{-a_k \hat{\mu}_1}{\operatorname{dist}(\lambda,\mu)}} & \cdots & \mathrm{e}^{\frac{-a_k \hat{\mu}_{m^*}}{\operatorname{dist}(\lambda,\mu)}} \end{bmatrix} \\ &= WZ^T. \end{aligned}$$

For a small rank k this approximation offers the much better accuracy than the previous approach. The error decays exponentially like $\epsilon = c_1 \mathrm{e}^{-c_2 k}$, where k is the rank of the approximation and c_1, c_2 are constant. From (**??**) and (24) it follows that the denomiantor is in the intervall $\begin{bmatrix} 1 & 3 \end{bmatrix}$. That

means that we need an exponential sum of rank $k = 9$ to get an approximation error $\epsilon \leq 1.55 \cdot 10^{-15}$, cf. [11]. Using a weakening factor $\eta = 0.5$ on the right hand side of (24) leads to an accuracy of $\epsilon \leq 9 \cdot 10^{-16}$ when using rank $k = 11$. To achieve the same accuracy by polynomial sums we need a rank of $k = 32$ for $\eta = 1$. We have to remark that the computation of the values α_l and a_l is too expensive to do it when performing the updating algorithm. Instead we have to use constants, cf. [11], which already have been computed for different intervals of the denominator (27).

5 The SVD Updating Algorithm

With the techniques of the previous sections we can summarize our updating algorithm as follows:

- Initial problem:
  - $U_+\Sigma_+V_+^T = U\Sigma V^T + ab^T$
- Go to symmetric characterizations:
  - $U_+\Sigma_+\Sigma_+^TU_+^T = U\Sigma\Sigma^TU^T + \rho_1 a_1 a_1^T + \rho_2 b_1 b_1^T$
  - $V_+^T\Sigma_+^T\Sigma_+V_+ = V^T\Sigma^T\Sigma V + \rho_3 a_2 a_2^T + \rho_4 b_2 b_2^T$
- And solve two symmetric updating problems of rank two:
  - Updating the singular values Σ_+ as the square root of the eigenvalues of the symmetric problem. Deflate the problem. (section 3)
  - Updating the singular vectors U_+ and V_+. (section 4)

There are two remaining tasks. As a consequence of the separate computation of U_+ and V_+ we have to ensure that the signs of the singular vectors fit together. Otherwise the product $U_+\Sigma_+V_+^T$ is not equal to A_+. Furthermore we have to improve the accuracy of the computed singular values because we lost half of the precision due to the squaring in section 2. Both of these problems are discussed in the following section.

5.1 Accuracy Improvement

In exact arithmetic the following equations hold

$$U\Sigma V^T + ab^T = U \underbrace{(\Sigma + \tilde{a}\tilde{b}^T)}_{C_u\Sigma_+C_v^T} V^T,$$

and

$$C_u(\Sigma + \tilde{a}\tilde{b}^T)C_v^T = \Sigma_+, \tag{28}$$

where $U\tilde{a} = a$ and $V\tilde{b} = b$. The matrix C_u is the product of both Cauchy like matrices that arise during the computation process of U_+. C_v is defined analogously.

Because of squaring the initial singular values and because of round-off errors we lose some accuracy in the matrix factors. Instead of (28) in practice we get a perturbed right hand side

$$C_u(\Sigma + \tilde{a}\tilde{b}^T)C_v^T = \underbrace{\tilde{\Sigma}_+ + \varepsilon}_{E}, \tag{29}$$

where the matrix $\varepsilon = e_{ij}$ is an off-diagonal error that was introduced during the updating process and $\tilde{\Sigma}_+ = \mathrm{diag}(\tilde{\sigma}_i)$ is close to the exact matrix Σ_+. An efficient explicit computation of (29) is possible because all occurring matrix products include either a diagonal or a Cauchy like matrix.

At first we can state that for computational reasons all entries with a magnitude less than $n \cdot eps \cdot \tilde{\sigma}_1$, where *eps* is the machine precision can be assumed to be zero. We observed that only a few off diagonal entries are larger than this magnitude. To refine the precision of the

computed singular values we suggest to follow Jacobi's idea which means to reduce the norm of the off-diagonal part from E.

If an element e_{ij}, where $j \leq n$, is larger than our intended order of accuracy we compute the SVD of the 2×2 sub-matrix $U_2 \Sigma_2 V_2^T = \begin{bmatrix} e_{ii} & e_{ij} \\ e_{ji} & e_{jj} \end{bmatrix}$. By multiplying the corresponding rows of E by U_2 and the corresponding columns by V_2 we eliminate the errors e_{ij} and e_{ji}. On the diagonal we get the more accurate singular values U_2.

If the element e_{ij} is situated in the rear part of E, i.e., $n < j \leq m$ we do an economy size QR factorization $Q_2 R_2 = \begin{bmatrix} e_{ii} \\ e_{ij} \end{bmatrix}$. After multiplying the corresponding rows of E by Q_2, element e_{ij} is eliminated and e_{ii} is more accurate.

We have to notice that all of the eliminating matrices have to be multiplied by U or V respectively. In practice it was observed that compared to the dimension of the matrix just a few number of rotations had to be performed. This fact ensures to maintain a complexity of $\mathcal{O}(n^2 \log^2 n)$.

Finally, we can use the refined Σ_+ to determine the signs of the singular vectors such that they fit together. Therefore we have to check if there are negative diagonal entries in Σ_+. For all of these elements we have to change the sign, and we have to multiply either the corresponding right or the corresponding left singular vector by -1. This sign correction is a special case of a general obstacle, cf. example 6.3, which can occur due to the transferring of the original problem into the symmetric ones.

5.2 Complexity Overview

Solving the initial problem (1) by the described algorithm causes the following computational effort. To obtain the symmetric characterization of our problem we have to compute some matrix-vector products. This can be done in $\mathcal{O}(n^2)$. The computation of the new singular values using the secular equation needs an effort of $\mathcal{O}(n^2)$ computations. One of the most expensive part is updating the singular vectors. Summarized, we have to evaluate four matrix-products. Due to the Cauchy-like structure of the occurring matrices we can solve this task in $\mathcal{O}(n^2 \log^2 n)$. Mostly the dimension n could be reduced previously because of deflation. In the next step we have to compute the product (29) for which it is necessary to evaluate three matrix products including Cauchy-like matrices. To maintain the intended order of accuracy we finally have to perform a number of Jacobi rotations. As long as the number of these rotations is small compared to the dimension of the problem, which is mostly the case in practice, this refinement can be done in $\mathcal{O}(n^2)$. Summarized the total computational complexity of the introduced algorithm is $\mathcal{O}(n^2 \log^2 n)$.

6 Numerical Examples

In this section we will present how the algorithm performs for different numerical examples. We did all the computations with a MATLAB implementation of the described algorithm. We have used a Cauchy matrix approximation by sums of exponentials. This choice significantly speeds up the algorithm compared to the other optinons. We set the deflation parameter to $\gamma = 1 \cdot 10^{-10}$,

dimension	$n^*, m^* > 20$	$n^*, m^* > 40$	time poly. sums	time exp. sums
750×500	228	55	3.467	1.701
1500×1000	571	221	17.643	10.162

Figure 1: matrix multiplication

the admissibility parameter to $\eta = 0.5$, and we declared two singular values as equal if their difference is less than $1 \cdot 10^{-10}$. We did all the computations on an AMD Athlon 64 X2 Dual Core 4400+ machine. Figure 1 shows the runtime comparison between the matrix multiplication with polynomial and exponential sums. When using exponential sums, we did the matrix approximation for each admissible block with a dimension larger than 20. For polynomial sums the dimension

had to be larger than 40. This difference is the result of demanding an approximation accuracy $\epsilon \leq 1 \cdot 10^{-15}$ which leads to a rank $k = 11$ for exponential and a rank $k = 32$ for polynomial sums.

6.1 Single Updates

In our first example we will show some results of some single rank-one updates for matrices with different dimensions in Figure 2. The type of matrix, either random or from matrix-market, and their dimension $m \times n$ is given in the first two columns. In the third and fourth column we show the 2-norm of U_+ and V_+ subtracted by 1. We use these values as a measurement how close the matrices are to be orthogonal. Column 'err. σ' shows the maximal relative error, $\max\left(\frac{|\sigma_+ - \sigma_M|}{\max(\sigma_M)}\right)$, between the updated singular values σ_+ and the exact singular values σ_M computed directly from A_+ by the standard MATLAB routine 'svd'. The relative error $\max\left|\frac{A_+ - U_+\Sigma_+V_+^T}{\max \sigma_M}\right|$ between the exact matrix A_+ and the product $U_+\Sigma_+V_+^T$ is given in the column 'err. A_+'. In the last one we show the number of refinement steps. Figure 3 shows the runtimes for the same examples as they

dimension	type	$\|U_+\| - 1$	$\|V_+\| - 1$	err. σ	err. A_+	ref. steps
1000×1250	random	$1.5 \cdot 10^{-11}$	$7.9 \cdot 10^{-11}$	$2.2 \cdot 10^{-15}$	$8.8 \cdot 10^{-14}$	7
500×625	random	$1.8 \cdot 10^{-11}$	$2.0 \cdot 10^{-11}$	$2.9 \cdot 10^{-16}$	$4.3 \cdot 10^{-14}$	6
250×320	random	$3.7 \cdot 10^{-11}$	$1.8 \cdot 10^{-11}$	$5.8 \cdot 10^{-16}$	$9 \cdot 10^{-14}$	32
1500×1500	random	$2.1 \cdot 10^{-11}$	$3.5 \cdot 10^{-10}$	$1.6 \cdot 10^{-15}$	$1.1 \cdot 10^{-13}$	12
200×1500	random	$7.6 \cdot 10^{-12}$	$3.0 \cdot 10^{-12}$	$2.3 \cdot 10^{-15}$	$4.8 \cdot 10^{-14}$	2
497×506	beacxc	$3.4 \cdot 10^{-12}$	$1.4 \cdot 10^{-12}$	$4.4 \cdot 10^{-13}$	$4.3 \cdot 10^{-12}$	803
66×66	bcsstk02	$4.4 \cdot 10^{-14}$	$1.9 \cdot 10^{-10}$	$3.5 \cdot 10^{-15}$	$1.6 \cdot 10^{-11}$	82

Figure 2: Single Singular Value Updates, Accuracy

are in Figure 2 in seconds. For the direct computation we have used the MATLAB function 'svd'.

dimension	1000×1250	500×625	250×320	1500×1500	200×1500	497×506	66×66
direct	39.502	5.642	0.659	95.941	2.432	1.884	0.018
update	30.728	5.237	1.354	62.335	35.989	6.590	0.429

Figure 3: Single Singular Value Updates, Runtime

At first we have to note that we compare our MATLAB code with MATLAB binaries which is disadvantageously for our algorithm. Nevertheless, compared to the direct computation the updating algorithm is faster for problems with a high dimension. In these cases it is possilble to approximate a lot of large blocks of the Cauchy matrix by low rank terms which makes the algorithm more efficient. For lower dimensions we can't exploit this approximation that much. Further, for small matrices A we lose a lot of time because it is necassary to compute four symmetric rank one updates in contrast to the direct computation of just one.

6.2 Update Sequence

In a second example we will show how the algorithm performs for a sequence of updates. We will demonstrate this for the following rank decomposition. We start with the initial matrices $A_0 = 0$ and B_0 which is randomly generated. In each step k we are searching for the largest element b_{ij}^k of B_k. Then we compute the schur-complement $B_{k+1} = B_k - \frac{1}{b_{ij}^k} B_k e_i e_j^T B_k$ and add the latter low rank term to matrix A_k, i.e., $A_{k+1} = A_k + \frac{1}{b_{ij}^k} B_k e_i e_j^T B_k$. We show the results of this decomposition for two problems of different dimension. For each of these problems Figure 4 shows for a several number of steps k the order of orthogonality of the matrices U and V, and the maximal relative error between the direct computed singular values compared to the updated ones. Further, in the

50×60				500×750			
upd	$\|U\| - 1$	$\|V\| - 1$	err. σ	upd	$\|U\| - 1$	$\|V\| - 1$	err. σ
10	$2.0 \cdot 10^{-15}$	$6.1 \cdot 10^{-15}$	$2.1 \cdot 10^{-15}$	100	$5.5 \cdot 10^{-13}$	$5.9 \cdot 10^{-13}$	$1.2 \cdot 10^{-15}$
20	$3.5 \cdot 10^{-14}$	$6.4 \cdot 10^{-13}$	$3.0 \cdot 10^{-15}$	200	$3.0 \cdot 10^{-12}$	$5.5 \cdot 10^{-12}$	$1.3 \cdot 10^{-14}$
30	$1.8 \cdot 10^{-13}$	$4.6 \cdot 10^{-13}$	$3.0 \cdot 10^{-15}$	300	$7.6 \cdot 10^{-12}$	$6.3 \cdot 10^{-12}$	$9.6 \cdot 10^{-14}$
40	$3.4 \cdot 10^{-13}$	$4.0 \cdot 10^{-13}$	$1.9 \cdot 10^{-13}$	400	$4.6 \cdot 10^{-11}$	$8.9 \cdot 10^{-11}$	$3.8 \cdot 10^{-13}$
50	$4.0 \cdot 10^{-13}$	$3.3 \cdot 10^{-13}$	$4.9 \cdot 10^{-13}$	500	$8.7 \cdot 10^{-11}$	$9.1 \cdot 10^{-11}$	$5.6 \cdot 10^{-13}$
$\max \left\lvert \frac{B_0 - U_n \Sigma_n V_n^T}{\max \sigma_M} \right\rvert = 4.1 \cdot 10^{-13}$				$\max \left\lvert \frac{B_0 - U_n \Sigma_n V_n^T}{\max \sigma_M} \right\rvert = 6.3 \cdot 10^{-11}$			

Figure 4: Single Singular Value Updates

last row the relative error (which should be zero in exact arithmetic and after n steps) between the original matrix B_0 and the matrix A_n is given.

6.3 Special Case

As a last example we want to deal with a special case of the updating problem. After performing steps (2)-(5) it might be possible that the updating vectors a_1 and b_1 are linear dependent. The positive effect in this case is that we can sum up the two rank one modifications to one term. That means we can skip one of the four symmetric rank one updates. The negative effect is that the separately computed subspaces of some of the multiple eigenvalues of $\Sigma_+\Sigma_+^T$ and $\Sigma_+^T\Sigma_+$ do not 'fit together'. As a consequence $A_+ \neq U_+\Sigma_+V_+^T$. Fortunately, this problem can be fixed by a small number of the refinement steps which are described in section 5.1.

This special case should be demonstrated by the following small example

$$U_+\Sigma_+V_+^T = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{U} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}}_{\Sigma} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}}_{V^T} + \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

After performing steps (2)-(5) we obtain the following symmetric rank two problem

$$U_+\Sigma_+\Sigma_+^TU_+^T = U\Sigma\Sigma^TU^T - 0.19258 \begin{bmatrix} -0.79284 \\ -0.79284 \\ -0.79284 \\ -0.79284 \end{bmatrix} \begin{bmatrix} -0.79284 \\ -0.79284 \\ -0.79284 \\ -0.79284 \end{bmatrix}^T + 5.19258 \begin{bmatrix} -1.17106 \\ -1.17106 \\ -1.17106 \\ -1.17106 \end{bmatrix} \begin{bmatrix} -1.17106 \\ -1.17106 \\ -1.17106 \\ -1.17106 \end{bmatrix}^T.$$

Obviously, both rank one terms are linearly dependent which allows us to sum them up and to do the update with just one term. To check if the finally computed product $U_+\Sigma_+V_+^T$ is equal to A_+ we use equation (28) which right hand side should be diagonal. But in our problem we get the disturbed right hand side (29)

$$\tilde{\Sigma}_+ + \epsilon = \begin{bmatrix} 5.3851 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -0.8944 & 0.4472 & 0 \\ 0 & 0 & -0.4472 & -0.8944 & 0 \end{bmatrix}.$$

After performing one refinement step as described in section 5.1 we finally end up with the exact solution. This examples shows that the last step of our method, the accuracy improvement, is essential to guarantee the computation of the right SVD factors.

7 Conclusions

In this paper we have shown that the rank-one updating problem of the singular value decomposition can be performed for almost all cases with a computational complexity of $\mathcal{O}(n^2 \log^2 n)$. The method that we presented consists of four major steps.

In the first step we are going from the initial problem to a symmetric characterization.

In the second step we have to compute the new eigenvalues of four symmetric subproblems. To solve these problems we have used the method which was introduced in [12].

In the third step we have to compute the new singular vectors. For this we have shown different approximation techniques to do an efficient multipliction of Cauchy matrices with matrices. It clearly turned out that using exponential sums is the method of choice. It offers the possibility to approximate large blocks of the Cauchy matrix C by low rank terms. Compared to this, using polynomial sums leads to much higher ranks for this approximation. The method which was introduced in [3] is not useable for large problems because of their stability problems.

In the last step we have to do some refinement steps to the computed matrices. Due to the squaring in step one we have lost some accuracy in our matrix factors. To recover a high order of accuracy the described transformations, based on Jacobi's idea are essential. Furthermore we need this step to determine the correct sign of the singular vectors and we need it to handle some special cases as described in section 6.3. Summarized, our algorithm offers the possibility to perform efficient rank-one updates of the singular value decomposition for high dimensions. For problems with lower dimension we can not exploit the low rank approximation of the Cauchy matrix which is one of the key points for efficiency in our algorithm. Furthermore it is still an open problem if it would be possible to do the squaring which is necessary to get the symmetric characterization implicitly. This would lead to a higher accuracy of the updated singular values and vectors.

References

[1] D. Braess and W. Hackbusch, *Approximation of* $1/x$ *by exponential sums in* $[1, \infty)$, IMA Journal of Numerical Analysis, (2005), pp. 685–697.

[2] J. Bunch and C. Nielsen, *Updating the singular value decomposition*, Numerische Mathematik, 31 (1978), pp. 111–129.

[3] A. Gerasoulis, *A fast algorithm for the multiplication of generalized hilbert matrices with vectors*, Mathematics of Computation, 50 (1988), pp. 179–188.

[4] I. Gohberg and V. Olshevsky, *Fast algorithms with preprocessing for matrix-vector multiplication problems*, Journal of Complexity, 10 (1994), pp. 411–427.

[5] G. Golub, *Trummer problem*, SIGACT News, 17 (1985), pp. 17.2–12.

[6] G. H. Golub, *Some modified matrix eigenvalue problems*, SIAM Rev., 15 (1973), pp. 318–334.

[7] L. Grasedyck and W. Hackbusch, *A multigrid method to solve large scale sylvester equations*, SIAM Journal on Matrix Analysis and Applications, 29 (2007), pp. 870–894.

[8] M. Gu and S. C. Eisenstat, *A stable and efficient algorithm for the rank-one modification of the symmetric eigenproblem*, SIAM Journal on Matrix Analysis and Applications, 15 (1994), pp. 1266–1276.

[9] ——, *Downdating the singular value decomposition*, SIAM Journal on Matrix Analysis and Applications, 16 (1995), pp. 793–810.

[10] W. Hackbusch, *A sparse matrix arithmetic based on* $\mathcal{H}$*-matrices. part i: Introduction to* $\mathcal{H}$*-matrices.*, Computing, 62 (1999), pp. 89–108.

[11] ——, *Entwicklungen nach exponentialsummen*, technical report, Max-Planck-Institut für Mathematik in den Naturwissenschaften Leipzig, 2009.

[12] D. S. J.R. Bunch, C.P. Nielsen, *Rank-one modification of the symmetric eigenproblem*, Numerische Mathematik, 31 (1978), pp. 31–48.

[13] V. Pan, *An algebraic approach to approximate evaluation of a polynomial on a set of real points*, Advances in Computational Mathematics, 3 (1995), pp. 41–58.

[14] V. Pan, *New transformations of cauchy matrices and trummer's problem*, Computers and Math. (with Applics.), 35 (1998), pp. 1–5.

[15] V. Rohklin, *A fast algorithm for the discrete laplace transformation*, Journal of Complexity, 4 (1988), pp. 12–32.

[16] J. Wilkinson, *The algebraic eigenvalue problem*, Clarendon Press, (1965).