Thomas Mach

Computing Inner Eigenvalues of Matrices in Tensor Train Matrix Format

Max Planck Institute Magdeburg Preprints

MPIMD/11-09 **December 5, 2011**

Impressum:

Max Planck Institute for Dynamics of Complex Technical Systems, Magdeburg

Publisher:
Max Planck Institute for Dynamics of Complex Technical Systems

Address:
Max Planck Institute for Dynamics of Complex Technical Systems
Sandtorstr. 1
39106 Magdeburg

www.mpi-magdeburg.mpg.de/preprints

Computing Inner Eigenvalues of Matrices in Tensor Train Matrix Format

Thomas Mach

Abstract The computation of eigenvalues is one of the core topics of numerical mathematics. We will discuss an eigenvalue algorithm for the computation of inner eigenvalues of a large, symmetric, and positive definite matrix M based on the preconditioned inverse iteration

$$x_{i+1} = x_i - B^{-1}\left(Mx_i - \mu(x_i)x_i\right),$$

and the folded spectrum method (replace M by $(M-\sigma I)^2$). We assume that M is given in the tensor train matrix format and use the TT-toolbox from I.V. Oseledets (see http://spring.inm.ras.ru/osel/) for the numerical computations. We will present first numerical results and discuss the numerical difficulties.

1 Introduction

Let $M \in \mathbb{R}^{m\times m}$ be a matrix. If the pair (λ, v) fulfills

$$Mv = \lambda v, \tag{1}$$

then λ is called an eigenvalue and v an eigenvector of M. They are computed in several applications like structural and vibrational analysis or quantum molecular dynamics. If M is small and dense, then this problem is almost solved. There are good algorithms for the computation of all eigenvalues, for instance the implicit, multi-shift QR algorithm with aggressive early deflation [2]. For large sparse matrices the Jacobi-Davidson algorithm [20] or the preconditioned inverse iteration [6], PINVIT for short, can be used to compute some eigenvalues.

Thomas Mach
Max Planck Institute for Dynamics of Complex Technical Systems, Sandtorstr. 1, 39106 Magdeburg, Germany, e-mail: thomas.mach@googlemail.com

For large matrices the dense approach is not feasible, since it requires m^2 storage entries and $\mathcal{O}(m^3)$ flops. Sparse matrices require only $\mathcal{O}(m)$ storage such that large matrices can be handle using sparse matrix arithmetic. But even $\mathcal{O}(m)$ may become too expensive for large m. The tensor-trains, see Section 1.1, are one way out, since the storage complexity is in $\mathcal{O}(\log_2 m)$.

The main feature of PINVIT is the matrix dimension independent convergence. This makes preconditioned inverse iteration a preferable method for the computation of the smallest eigenvalues of a matrix given in a compressed storage scheme like the tensor train matrix format. The combination with the folded spectrum method permits also the computation of inner eigenvalues. In [1], this was investigated for the data-sparse hierarchical matrices by the author.

1.1 Tensor Trains

The concept of tensor trains, TT for short, as described in [17] permits the computation of a data-sparse approximation of tensors. Therefore, the tensor $T \in \mathbb{R}^{n^d}$ is approximated by

$$T = \sum_{\alpha_1,\dots,\alpha_d=1}^{r_1,\dots,r_d} T_1(i_1,\alpha_1)T_2(\alpha_1,i_2,\alpha_2)\cdots T_{d-1}(\alpha_{d-2},i_{d-1},\alpha_{d-1})T_d(\alpha_{d-1},i_d). \quad (2)$$

The ranks r_i of the summations are called the local ranks of the approximation. The smallest r with $r \geq r_i \ \forall i$ is the local rank of the tensor train. The TT decomposition for $d = 2$ is the low rank factorization

$$\mathbb{R}^{n^2} \ni T = \operatorname{vec}\left(AB^T\right), \quad A,B \in \mathbb{R}^{n\times r}.$$

For the full tensor we have to store n^d entries, but for the approximation in the tensor train format $(d-2)nr^2 + 2nr$ entries are sufficient. The main advantage is that the storage complexity grows only linearly with d in the tensor train format. The Tucker decomposition of tensors

$$T = \sum_{\alpha_1,\dots,\alpha_d=1}^{r_1,\dots,r_d} C(\alpha_1,\dots,\alpha_d)T_1(i_1,\alpha_1)\cdots T_d(i_d,\alpha_d)$$

is another format to handle tensors, but here the core tensor C is a d-dimensional object requiring $\mathcal{O}(r^d)$ storage entries. In [17] it is also described, how to compute a TT approximation to a given tensor. This is an advantage compared to the canonical format, which can only be computed by solving an NP-hard problem, as shown in [4].

There are arithmetic operations with tensor trains available in the tensor train toolbox [16] for MATLAB®. We will use here the TT dot product, the addition of two tensor trains, and the rounding of a tensor train, which are all described in [15].

The addition of two tensor trains $A+B$ can be performed within the TT format by

$$C_k(i_k) = \begin{bmatrix} A_k(i_k) & 0 \\ 0 & B_k(i_k) \end{bmatrix}, \qquad \text{for } k = 2,\dots,d-1$$
$$C_1(i_1) = \begin{bmatrix} A_1(i_1)\ B_1(i_1) \end{bmatrix}, \quad C_d(i_d) = \begin{bmatrix} A_d(i_d) \\ B_d(i_d) \end{bmatrix},$$

for details see [15], where the complexity is given with $\mathcal{O}(dnr^3)$. The local ranks of C are the sum of the local ranks of A and B. After the addition one should do a truncation to get a result of smaller local rank. If one omit the truncation, then a series of addition would lead to a growing local rank, destroying the good complexity of the TT format.

The dot product of two tensor carriages to the same index i_k is

$$\sum_{i_k} A_k(\alpha_{k-1}, i_k, \alpha_k) B_k(\beta_{k-1}, i_k, \beta_k) = v_2(\alpha_{k-1}, \alpha_k, \beta_{k-1}, \beta_k).$$

We start with $k=1$ and get $v(\alpha_1, \beta_1)$. Together with $v_2(\alpha_1, \beta_1, \alpha_2, \beta_2)$ for $k=2$ we compute

$$\sum_{\alpha_1, \beta_1} v(\alpha_1, \beta_1) v_2(\alpha_1, \beta_1, \alpha_2, \beta_2) \Rightarrow v(\alpha_2, \beta_2).$$

We repeat this for all k and get the dot product (A,B). This procedure requires also $\mathcal{O}(dnr^3)$ flop, see [15]. The rounding of tensor trains is also in $\mathcal{O}(dnr^3)$ and is also described in [15].

1.2 Tensor Train Matrix Format

If we have given a vector $v \in \mathbb{R}^{n^2}$ representing the values of a function over a 2D regular grid, then we can regard v as a vector over the product index set (i_1, i_2), with $i_1 \in I_1$ and $i_2 \in I_2$, where I_1 is the index set for the points in the one direction and I_2 for the points in the other direction. The generalization of this concept leads to a tensor.

A matrix over the same grid would be described by $((i_1, i_2), (j_1, j_2))$, with (i_1, i_2) as row index and (j_1, j_2) as column index. The generalization leads to a tensor with the structure

$$T(i_1, \dots, i_d; j_1, \dots, j_d),$$

where the semicolon separates the row and the column indices. Of cause one can use again a tensor train approximation, but now with two indices per tensor carriage:

$$M = \sum_{\alpha_1,\dots,\alpha_d} M_1(i_1, j_1, \alpha_1) M_2(\alpha_1, i_2, j_2, \alpha_2) \cdots M_d(\alpha_{d-1}, i_d, j_d). \tag{3}$$

This data-sparse matrix format was recently invented by Oseledets, see [15]. A matrix in this format is called a *TTM matrix* or a *matrix in tensor train matrix format*.

If one regards (i_k, j_k) as one long index of length n^2, then the same data structure as in the tensor train case can be used. This immediately permits us to perform additions and rounding with in the TTM matrices. Further the dot product can be used to compute the Frobenius norm of a TTM matrix. The complexity of this operations is due to the indices of length n^2 instead of n:

$$\mathcal{O}(dn^2r^3).$$

We will further use the TTM-TT matrix-vector product, which is defined by

$$W(i_1, i_2, \dots, i_d) = \sum_{j_1, j_2, \dots, j_d} M(i_1, j_1, i_2, j_2, \dots, i_d, j_d) T(j_1, j_2, \dots, j_d).$$

This product can be computed carriage-wise by

$$W_k((\alpha_{k-1}, \beta_{k-1}), i_k, (\alpha_k, \beta_k)) = \sum_{j_k} M_k(\alpha_{k-1}, i_k, j_k, \alpha_k) T_k(\beta_{k-1}, j_k, \beta_k).$$

This procedure squares the local rank. So one should truncate afterwards. The n is often chosen to be 2 and the tensor trains are then called QTT.

1.3 Problem Setting

We assume the matrix $M \in \mathbb{R}^{2^d \times 2^d}$ is given in tensor train matrix format. The task is to compute an eigenvector v of M. Since 2^d might be large we are satisfied with an approximation to v in the tensor train format.

We will use preconditioned inverse iteration (PINVIT) to compute the smallest eigenvalue and the corresponding eigenvector of M. A similar approach was already investigated by O. Lebedeva in [10], but not for TTM matrices. Here we will extend this to the computation of inner eigenvalues by using the folded spectrum method.

In [9] Christine Tobler and Daniel Kressner investigated the usage of LOBPCG for the solution of eigenvalues problems with matrices in $\mathscr{H}$-Tucker format. The LOBPCG method is a variant of PINVIT that uses the optimal vector in the space spanned by the two previous iterates and the preconditioned residual as next iterate.

2 Preconditioned Inverse Iteration

PINVIT can be motivated as an inexact Newton-method for the minimization of the Rayleigh quotient. The Rayleigh quotient $\mu(x)$ for a vector x and a symmetric, positive definite matrix M is defined by

$$\mu(x,M) := \mu(x) = \frac{x^T M x}{x^T x}. \tag{4}$$

The global minimum of $\mu(x)$ is reached for $x = v_1$, with $\lambda_1 = \mu(x)$. This means that minimizing the Rayleigh quotient is equal to computing the smallest eigenvalue. Doing this minimization by the following inexact Newton method:

$$x_{i+1} = x_i - B^{-1}\left(Mx_i - \mu(x_i)x_i\right), \tag{5}$$

we get the update equation of preconditioned inverse iteration. The preconditioner B^{-1} for M have to fulfill

$$\left\|I - B^{-1}M\right\|_M \leq c < 1. \tag{6}$$

This method is know for a long time and was, among others, extensively investigated by Knyazev and Neymeyr, see [6, 12, 13, 8]. The main feature of preconditioned inverse iteration is the independence of the convergence from the matrix dimension m. PINVIT requires positive definiteness.

In [7] Knyazev used the optimal vector in the subspace $\{x_{i-1}, B^{-1}e_{i-1}, p_{i-1}\}$ as next iterate. The resulting method is called linear optimal (block) preconditioned conjugate method (LOBPCG):

$$\begin{aligned} e_{i-1} &= Mx_{i-1} - x_{i-1}\mu(x_{i-1}) \\ x_i &= B^{-1}e_{i-1} + \tau_{i-1}x_{i-1} + \gamma_{i-1}p_{i-1} \\ p_i &= B^{-1}e_{i-1} + \gamma_{i-1}p_{i-1}, \quad p_0 = 0. \end{aligned}$$

3 Preconditioned Inverse Iteration for Matrices in Tensor Train Matrix Format

Now we apply the preconditioned inverse iteration briefly discussed in the last section to a matrix in tensor train matrix format. A subspace version of preconditioned inverse iteration is listed in Algorithm 1.

First we have to compute the preconditioner. We will use here the Newton-Schulz iteration [19], which was first used for TTM matrices in [14], see Algorithm 2 that is based on the modification of the Newton-Schulz method for TTM matrices in [18]. If the iteration does not converge, we reduce the truncation parameter ε and start the process again.

Algorithm 1: Subspace Preconditioned Inverse Iteration

Input: $M \in \mathbb{R}^{m\times m}$, $X_0 \in \mathbb{R}^{m\times s}$ e. g. randomly chosen
Output: $X_p \in \mathbb{R}^{m\times s}$, $\mu \in \mathbb{R}^{s\times s}$, with $\|MX_p - X_p\mu\| \leq \varepsilon$

$T^{-1} \approx (M)^{-1}$;
Orthogonalize X_0;
$\mu := X_0^T M X_0$;
$R := \texttt{round}(MX_0 - X_0\mu, \varepsilon)$;
$i := 1$;
while $\|R\|_F > \varepsilon$ **do**
  $i := i + 1$;
  $X_i := \texttt{round}(X_{i-1} - T^{-1}R, \varepsilon)$;
  Orthogonalize X_i;
  $R := \texttt{round}(MX_i - X_i\mu, \varepsilon)$ with $\mu := X_i^T M X_i$;
end

Algorithm 2: Newton-Schulz Inversion

Input: $M \in \mathbb{R}^{m\times m}$
Output: $M^{-1} \in \mathbb{R}^{m\times m}$

$Y = M/\|M\|_2$;
$X = I/\|M\|_2$;
while $\|Y - I\| > \varepsilon_c$ **do**
  $H = \texttt{round}(2I - Y, \varepsilon)$;
  $Y = \texttt{round}(YH, \varepsilon)$;
  $X = \texttt{round}(HX, \varepsilon)$;
end
$M^{-1} = X$;

In the remainder of the Algorithm 1 only TT dot products, TTM-TT matrix-vector products, and additions are used.

4 Computing Inner Eigenvalues by Folded Spectrum Method

Sometimes one is also interested in the inner eigenvalues M. Unfortunately a simple shifting, $M - \mu I$, is not possible, since the preconditioned inverse iteration requires positive definiteness of M. One way out is the use of the so called folded spectrum method, [21], which was before also mentioned in [11]. The key observation are the following facts: First, the matrix $M_\mu = (M - \mu I)^2$ is positive definite. If M is symmetric, then M_μ is symmetric, too. Second, an eigenvector v of M to the eigenvalue λ is also an eigenvector of M_μ, since

$$\begin{aligned} M_\mu v &= (M - \mu I)^2 v = M^2 v - 2\mu M v + \mu^2 v \\ &= \lambda^2 v - 2\mu\lambda v + \mu^2 v = (\lambda - \mu)^2 v. \end{aligned}$$

The computation of inner eigenvalues consists of the following steps:

- Choose a shift μ.
- Compute $M_\mu = (M - \mu I)^2$.
- Use PINVIT to compute the smallest eigenpair (v, λ) of M_μ.
- The sought eigenpair is $(v, v^T M v / v^T v)$.

As the numerical experiments in the next section show this procedure can be used to compute some inner eigenvalues of a TTM matrix. But this algorithm have two drawbacks. First the condition number of M_μ is approximately the square of the condition number of M. This means that it is more difficult to invert M_μ. The first consequence is that the Newton-Schulz iteration takes longer. Second, the approximate inverse has larger local ranks and so more storage is necessary. Third, the larger local ranks make the application of the preconditioner more expensive, such that the preconditioning in each step takes also longer. The numerical experiments confirm this.

The second drawback is that the squaring "fold" the spectrum. It may happen that two eigenvalues on different sides of μ become (almost) equal after the squaring. In this case the preconditioned inverse iteration will compute an vector in the invariant subspace spanned by the eigenvectors of both eigenvalues. If we compute the Rayleigh quotient for this vector, then we may get an arbitrary value between the two eigenvalues. So one should carefully choose the shift in a way not producing new multiple eigenvalues. Further the computation of the whole invariant subspace lead to good approximations of the sought eigenvalues. In the numerical experiments the shifts and subspace dimension are chosen by hand, such that these problems do not occur.

5 Numerical Experiments

In this section we will present some first numerical experiments. The numerical computation are performed on an compute node with two Intel®Xeon®Westmere X5650 with 2.66GHz and 48GB DDR3 RAM. We use the preconditioned inverse iteration as described in Algorithm 1. We implement this algorithm based on the tensor train toolbox for MATLAB, [16].

The source code used for the following numerical examples is available from http://www.mpi-magdeburg.mpg.de/preprints/2011/1109/.

As example we use the Laplace-equation over the unit-square in 2, 3 or 4 dimensions. We stop the iteration after 100 steps. In [5] it was shown that there is an explicit QTT representation of the Laplace operator. Tables 1–5 show the results. The computation of the eigenvalues has two main parts, the computation of the preconditioner and the iteration itself. We choose the preconditioner accuracy $c = 0.2$, see Equation 6. We stop the iteration if the Frobenius norm of the residual drops below 10^{-5}. Inside the iteration we truncate each tensor trains to a precision of 10^{-6}. If the convergence stagnates we reduce the truncation tolerance. We observe that the

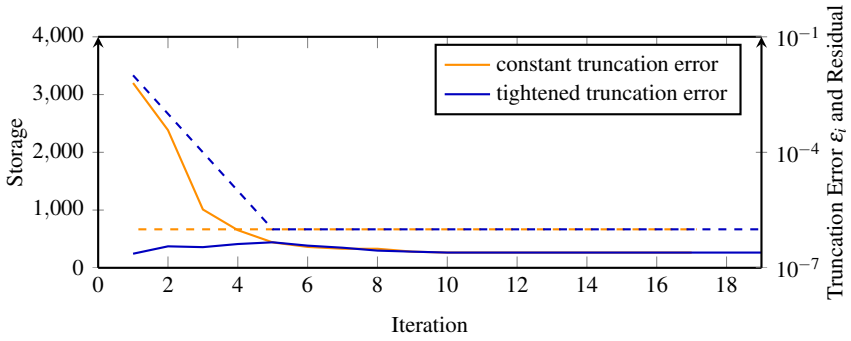

Fig. 1 Memory (left axis, solid line) used for $X^{(i)}$ depending on the truncation error ε_i (right axis, dashed line); 2D-Laplace example, $d = 6$.

local rank of the approximation in the first steps can be very high compared with the local rank in the final approximation. Therefore it makes sense to start with precision 10^{-2} in the first iteration and tighten this by a factor 10 in the first four steps, so that we reach 10^{-6}. The effect is shown in Figure 1 for the 2D Laplace example with $d = 6$. This idea was described in [3] for general truncated iterations.

The tables show the size of the matrix in TTM format in the first column, d in the second. We have 2^d discretization points in each direction. The third column gives the time for the computation of the approximate inverse, where each * indicates a failed of the Newton-Schulz iteration, due to too large round-off errors. The fourth column shows the time for the preconditioned inverse iteration. This is followed by the number of iterations. The error of the computed eigenvalues, which you found in the last column, is measured by

$$\left\| \left(\frac{\lambda_i - \hat{\lambda}_i}{\lambda_i} \right)_{i=1}^{s} \right\|_\infty .$$

One can see that the number of iterations is almost independent from the matrix dimension. The relative error is smaller than 10^{-5}. We further observe that the required CPU time grows slower than the matrix dimension m. This permit the computation of the eigenvalues of large matrices.

The computation for the shifted and squared matrix M_μ is much more expensive than for M. This confirms the expectations from the last section. The large local ranks of the approximation of M_μ^{-1} limit the usage of the method to the computation of inner eigenvalues of comparable small matrices.

n	d	t_{inv} in s	t_{PINVIT} in s	# it	error
16	2	0.253	0.557	20	2.1651 e−07
64	3	0.132	0.729	19	2.0918 e−07
256	4	0.408	1.315	24	2.0509 e−07
1 024	5	1.038	1.933	18	2.5084 e−07
4 096	6	2.637	4.371	17	1.8998 e−07
16 384	7	5.174	9.132	19	7.3614 e−08
65 536	8	9.233	25.326	21	6.8874 e−08
262 144	9	16.240	37.645	19	1.5501 e−08
1 048 576	10	27.387	48.643	21	2.5858 e−10
4 194 304	11	99.911	146.910	25	1.5207 e−09
16 777 216	12	140.043	154.761	23	9.9111 e−10
67 108 864	13	528.491**	348.101	20	2.8968 e−08
268 435 456	14	1 064.433***	767.721	26	1.5802 e−07
1 073 741 824	15	1 919.606***	2 767.084	53	7.4038 e−07
4 294 967 296	16	3 423.903****	2 796.697	28	6.8776 e−07

Table 1 Numerical results 2D Laplace, three smallest eigenvalues.

n	d	t_{inv} in s	t_{PINVIT} in s	# it	error
16	2	0.116	0.486	17	2.4130 e−11
64	3	0.440	0.304	7	2.2533 e−10
256	4	2.832	2.827	47	8.9370 e−09
1 024	5	15.632	3.431	25	5.0051 e−11
4 096	6	47.027	13.979	33	2.5580 e−12
16 384	7	986.699**	39.519	35	4.8601 e−12
65 536	8	248.160	1 354.844	100	1.4737 e−01

Table 2 Numerical results 2D Laplace, shifted ($\mu = 203.3139$), three eigenvalues.

n	d	t_{inv} in s	t_{PINVIT} in s	# it	error
64	2	0.304	1.438	24	2.0448 e−07
512	3	0.327	2.710	26	3.1414 e−07
4 096	4	1.310	7.760	22	3.1719 e−07
32 768	5	4.045	40.870	30	2.5526 e−07
262 144	6	10.305	129.937	25	2.4926 e−07
2 097 152	7	18.168	473.681	23	1.4520 e−07
16 777 216	8	37.216	2 084.095	24	3.0187 e−08
134 217 728	9	97.761	14 432.496	100	1.9257 e−06
1 073 741 824	10	93.512	4 425.800	21	1.6571 e−08

Table 3 Numerical results 3D Laplace, four smallest eigenvalues.

n	d	t_{inv} in s	t_{PINVIT} in s	# it	error
64	2	0.289	11.450	100	5.9918 e−06
512	3	8.174	18.930	30	2.5871 e−07
4 096	4	99.393	117.183	67	2.9559 e−12
32 768	5	2 347.897*	out of memory		

Table 4 Numerical results 3D Laplace, shifted ($\mu = 230.6195$), six eigenvalues.

n	d	t_{inv} in s	t_{PINVIT} in s	# it	error
256	2	0.362	3.555	30	1.8569 e−07
4 096	3	0.480	9.519	29	2.4328 e−07
65 536	4	1.885	58.279	28	2.8761 e−07
1 048 576	5	5.719	260.385	28	2.4963 e−07
16 777 216	6	15.194	1 028.412	25	1.6518 e−07
268 435 456	7	29.104	5 282.001	35	2.9061 e−09

Table 5 Numerical results 4D Laplace, five smallest eigenvalues.

6 Conclusions

We have seen that the preconditioned inverse iteration can be used to compute the smallest eigenvalues of a matrix in tensor train matrix format. Further the folded spectrum method makes PINVIT also applicable for the computation of inner eigenvalues. The numerical experiments showed that the computation of the smallest eigenvalues is much cheaper than the computation of inner eigenvalues.

The folded spectrum method leads to bad conditioned problems. The bad condition makes the inversion more expensive and cause higher local ranks in the approximate inverse, which make the application of the preconditioner expensive. The choose of adequate shifts and adequate subspace dimensions for the computation of inner eigenvalues remains a difficult issue that require further investigations.

A shorted version of this preprint was submitted to the Proceedings of the ENUMATH 2011 (Leicester).

Acknowledgements The author thanks Peter Benner for suggesting to investigate the combination of preconditioned inverse iteration and folded spectrum method also for tensor trains.

References

1. Benner, P., Mach, T.: The preconditioned inverse iteration for hierarchical matrices. Tech. Rep. MPIMD/11-01, Max Planck Institute Magdeburg Preprints (2011)
2. Braman, K., Byers, R., Mathias, R.: The multi-shift QR-algorithm: Aggressive early deflation. SIAM J. Matrix Anal. Appl. **23**(4), 948–989 (2002)
3. Hackbusch, W., Khoromskij, B., Tyrtyshnikov, E.E.: Approximate iterations for structured matrices. Numer. Math. **109**, 365–383 (2008). DOI 10.1007/s00211-008-0143-0
4. Hastad, J.: Tensor rank is NP-complete. Journal of Algorithms **11**(4), 644–654 (1990)
5. Kazeev, V.A., Khoromskij, B.N.: On explicit qtt representation of laplace operator and its inverse. Tech. Rep. 75, Max-Planck-Institute for Mathematics in the Sciences, Leipzig (2010)
6. Knyazev, A.V.: Preconditioned eigensolvers — an oxymoron? Electr. Trans. Num. Anal. **7**, 104–123 (1998)
7. Knyazev, A.V.: Toward the optimal preconditioned eigensolver: locally optimal block preconditioned conjugate gradient method. SIAM J. Sci. Comput. **23**, 517–541 (2001)
8. Knyazev, A.V., Neymeyr, K.: Gradient flow approach to geometric convergence analysis of preconditioned eigensolvers. SIAM J. Matrix Anal. Appl. **31**(2), 621–628 (2009)

9. Kressner, D., Tobler, C.: Preconditioned low-rank methods for high-dimensional elliptic PDE eigenvalue problems. Comput. Methods Appl. Math. **11**(3), 363–381 (2011)
10. Lebedeva, O.S.: Block tensor conjugate gradient-type method for Rayleigh quotient minimization in two-dimensional case. Comput. Math. Math. Phys. **50**, 749–765 (2010). URL http://dx.doi.org/10.1134/S0965542510050015. 10.1134/S0965542510050015
11. Morgan, R.B.: Computing interior eigenvalues of large matrices. Linear Algebra Appl. **154–156**, 289–309 (1991)
12. Neymeyr, K.: A geometric theory for preconditioned inverse iteration I: Extrema of the Rayleigh quotient. Linear Algebra Appl. **322**(1–3), 61–85 (2001)
13. Neymeyr, K.: A hierarchy of preconditioned eigensolvers for elliptic differential operators. Habilitationsschrift, Mathematische Fakultät der Universität Tübingen (2001)
14. Oseledets, I.V.: Approximation of $2^d \times 2^d$ matrices using tensor decomposition. SIAM J. Matrix Anal. Appl. **31**(4), 2130–2145 (2010)
15. Oseledets, I.V.: Tensor-train decomposition. SIAM J. Sci. Comput. **33**(5), 2295–2317 (2011)
16. Oseledets, I.V.: TT-toolbox 2.1. http://spring.inm.ras.ru/osel/ (2011)
17. Oseledets, I.V., Savostyanov, D.V., Tyrtyshnikov, E.E.: Linear algebra for tensor problems. Computing **85**, 169–188 (2009)
18. Oseledets, I.V., Tyrtyshnikov, E.E.: Breaking the curse of dimensionality or how to use SVD in many dimensions. SIAM J. Sci. Comput. **31**(5), 3744–3759 (2009)
19. Schulz, G.: Iterative Berechung der reziproken Matrix. ZAMM Z. Angew. Math. Mech. **13**, 57–59 (1933)
20. Sleijpen, G.L.G., Van der Vorst, H.A.: A Jacobi-Davidson iteration method for linear eigenvalue problems. SIAM Rev. **42**(2), 267–293 (2000)
21. Wang, L.W., Zunger, A.: Electronic structure pseudopotential calculations of large ($\sim$1000 atoms) Si quantum dots. J. Phys. Chem. **98**(98), 2158–2165 (1994)