Advances in Engineering Software 117 (2018) 8–17

Contents lists available at ScienceDirect

Advances in Engineering Software

journal homepage: www.elsevier.com/locate/advengsoft

Research Paper

Singular Value Decomposition used for compression of results from the Finite Element Method

Štěpán Beneš, Jaroslav Kruis*

Department of Mechanics, Faculty of Civil Engineering, Czech Technical University in Prague Thákurova 7, Prague 166 29, Czech Republic

ARTICLE INFO

Keywords:
Finite Element Method (FEM)
Compression
Singular Value Decomposition (SVD)
Randomized SVD
Data visualization
Data post-processing

ABSTRACT

A complex finite element analysis can produce large amount of data that is problematic to post-process in reasonable time. This paper describes application of Singular Value Decomposition (SVD) to the compression of results from finite element solvers. Although the idea of image compression method is an inspiration for this research work, the SVD compression algorithm used for compression of images cannot be directly used for FEM results. Differences and implementation challenges are discussed in the text. Quality of approximation is more important in scientific field than in computer graphics where the most significant factor is the human perception of the resulting image. Error estimation methods used during compression of finite element results are presented. The focus is also on the algorithm performance. SVD is a very computational intensive method. Therefore, various optimization techniques were investigated, e.g. randomized SVD. The method leads to the lower memory consumption, 10% of the original size or less, with negligible compression error.

1. Introduction

Amount of data produced by a complex finite element analysis can be enormous and typical personal computer is not capable to store, process and visualize the results in reasonable time. Singular Value Decomposition is a well known factorization method that provides rich information about matrix systems. One of its many applications is the image compression where it can significantly reduce the size of the data representing an image while preserving the quality of image appearance.

The effort to reduce size of resulting data from complex finite element analyses to accelerate (or even make possible) post-processing using common personal computer is not new. In [1], there is one possible direction presented for research in this area – the replacement of discrete data produced by finite element solver by continuous functions. These approximation functions can be then described by a small number of parameters. The main goal is to find the areas in domain where the output discrete function has predictable development and can be easily replaced by a simple continuous function, e.g. linear function. Although this approach has remarkable results for some classes of functions it is not applicable to a general problem. For some special cases, such as functions with discontinuities, the method has poor results because approximation error is too high and – what is more important – it cannot be determined in advance.

In this study it was decided to apply purely algebraic approach. Various methods were considered, e.g. discrete wavelet transform [2] and discrete cosine transform [3], for their successful use in image compression. SVD method [4–6] was chosen as the foundation of the compression algorithm. Considering that the results from the FEM analyses can be viewed as a series of arbitrary rectangular matrices, the implementation of compression algorithm based on SVD is straightforward as it can be applied to any rectangular matrix.

Compression methods usually yield approximated data. In the following text, the term *approximation error* denotes an error resulted from compression, i.e. difference between original results of FEM analysis and their compressed form. It should not be confused with the error of the Finite Element Method itself that yields approximate solution to mathematical problems used to model physical reality. To quantify the approximation error, several error metrics were investigated. In [7], there are some of them used in similar area of research. The ability to control the quality of compression was a key requirement for the implementation of the compression algorithm.

The quality of any compression method depends on the nature of input data. Therefore, several different non-trivial finite element analyses should be used as benchmarks for an implementation of the compression algorithm. Also, the size of input data (the output from the FEM analysis) should be large enough to test the algorithmic complexity under heavy load. Details about the analyses that were used as benchmarks can be found in [8–11].

Performance is very important aspect for development of the

* Corresponding author.
E-mail addresses: stepan.benes@fsv.cvut.cz (Š. Beneš), jaroslav.kruis@fsv.cvut.cz (J. Kruis).

https://doi.org/10.1016/j.advengsoft.2017.12.007
Received 28 June 2017; Received in revised form 30 November 2017; Accepted 28 December 2017

compression algorithm. SVD being very computational intensive, randomized algorithms for SVD decomposition were investigated [12–15]. Especially, the implementation described in [16] was used in the compression algorithm for the optimization. Memory consumption and the suitable format of the output data from the compression algorithm also influence overall usability of the resulting work. Efficient data storage is connected with data structure [17,18]. This area will be subject of further research.

Although the idea to use SVD for data compression is not new, it is applied in an entirely new area of post-processing of results from the FEM analysis. In contrast with image compression, where SVD was used in a lossy compression algorithms, the area of post-processing puts great emphasis on the mathematical accuracy of the approximation instead of the human perceptions of visualizations. Also, the storage and dimensions of the input data, and the variance of the values in data, are very different. The main contribution of this paper is therefore to explore this area and to offer a description of an implementation of the algorithm that is successfully used in the post-processing of large data.

The rest of the paper is organized as follows. Section 2 contains mathematical background of the SVD compression (i.e. SVD method itself, its use in low-rank matrix approximation, and description of the randomized SVD method). Implementation of the compression algorithm is presented in Section 3. Section 4 summarizes the results of the benchmarks that were designed to measure quality of the output from the compression algorithm, and also the performance of its implementation. The paper is concluded in Section 5.

2. Mathematical background

Singular value decomposition is based on a theorem from linear algebra which says that a rectangular matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ can be decomposed into the product of three matrices - an orthogonal matrix $\mathbf{U} \in \mathbb{R}^{m\times m}$, a diagonal matrix $\mathbf{S} \in \mathbb{R}^{m\times n}$, and the transpose of an orthogonal matrix $\mathbf{V} \in \mathbb{R}^{n\times n}$:

$$\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^{\mathrm{T}}, \tag{1}$$

where $\mathbf{U}^{\mathrm{T}}\mathbf{U} = \mathbf{I}$, $\mathbf{V}^{\mathrm{T}}\mathbf{V} = \mathbf{I}$. The columns of $\mathbf{U}$ are orthonormal eigenvectors of $\mathbf{A}\mathbf{A}^{\mathrm{T}}$, which are called the left singular vectors. The columns of $\mathbf{V}$ are orthonormal eigenvectors of $\mathbf{A}^{\mathrm{T}}\mathbf{A}$ called the right singular vectors. $\mathbf{S}$ (sometimes referred to as $\mathbf{\Sigma}$) is a diagonal matrix containing singular values in descending order, which are at the same time the nonzero square roots of the eigenvalues of $\mathbf{A}\mathbf{A}^{\mathrm{T}}$ and $\mathbf{A}^{\mathrm{T}}\mathbf{A}$.

SVD can be seen as a method for transforming correlated variables into a set of uncorrelated ones. At the same time, SVD is a method for ordering the dimensions based on variation and identifying the dimension with the largest variation. Once this dimension is identified, it is possible to find the best approximation of the original data points using fewer dimensions. Hence, SVD can be seen as a method for data reduction/compression.

2.1. SVD compression

This is the basic idea behind SVD: taking a high dimensional, highly variable set of data points and reducing it to a lower dimensional space that exposes the substructure of the original data more clearly and orders it from the largest variation to the least. What makes SVD practical for data compression applications is that variation below a particular threshold can be simply ignored to massively reduce data with assurance that the main relationships of interest have been preserved.

The objective of a compression algorithm is to reduce amount of data representing FEM results and also the ability to reconstruct original data from its smaller representation. This saves storage capacity and also accelerates the data transfer between computers as the analysis itself and the post-processing of results is sometimes done on different workstations.

A compression method can be lossy or lossless. Lossless methods are able to fully reconstruct original data. Lossy methods, on the other hand, produce only approximations of original data.

SVD is used in this paper as a part of the compression algorithm. The SVD method applied to arbitrary matrix produces decomposition that consists of corresponding singular values and singular vectors. This process is fully reversible (with the assumption that the numerical errors are negligible). The original matrix can be reconstructed by the multiplication of decomposed parts. However, the compression algorithm is based on modification of decomposition to create low-rank approximation matrix. The reconstructed matrix slightly differs from the original matrix and algorithm therefore performs lossy compression.

2.2. Low-rank approximation matrix

From the definition of SVD in (1) and from the properties of SVD, the fact follows that a matrix can be represented in the form of its SVD components as a sum of k rank-1 matrices

$$\mathbf{A} = \sum_{i=1}^{k} s_i \mathbf{u}_i \mathbf{v}_i^{\mathrm{T}}, \tag{2}$$

where s_i is the ith singular value of matrix $\mathbf{A}$, $\mathbf{u}_i$ and $\mathbf{v}_i$ are corresponding singular vectors of matrix $\mathbf{A}$, and $k = \min(m, n)$. Considering the fact that singular values are ordered $s_1 \geq s_2 \geq s_3 \geq \ldots \geq s_k$, the above formula implies that the first term of the sum would have the highest contribution and the last term would have the lowest contribution to matrix $\mathbf{A}$. Therefore, if we take only first r members of the above summation we get an approximation matrix

$$\mathbf{A}' = \sum_{i=1}^{r} s_i \mathbf{u}_i \mathbf{v}_i^{\mathrm{T}}. \tag{3}$$

Quality of approximation depends on the magnitude of the singular values omitted from the approximation formula, namely $s_{r+1} \ldots s_k$. The compression algorithm is based on an assumption that the first singular value is order-of-magnitude higher than singular values at the end of the decomposition sequence. In special cases, when $r = k$, or $s_i = 0$ for all $i > r$, the omitted singular values do not contribute to the sum and the compression is therefore lossless. In other cases, approximation error has to be calculated and taken into account to avoid loss of important details in data.

The main goal of the compression algorithm is to find a compromise between low approximation error and high compression ratio c which is calculated using the formula

$$c = \frac{r(m + n + 1)}{mn}, \tag{4}$$

where m is the number of rows and n is the number of columns of matrix $\mathbf{A}$. Explanation of the compression ratio formula is best done using Fig. 1. Light color represents the part of matrix decomposition that is to be stored in the output file as a low-rank approximation of the input.

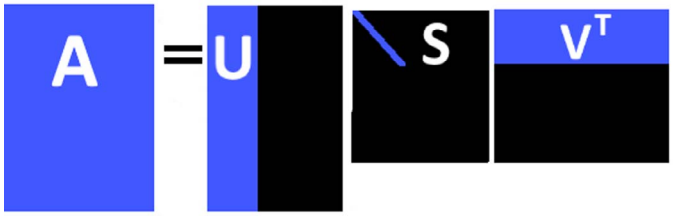

Fig. 1. Decomposition of input matrix $\mathbf{A}$ into diagonal matrix of singular values $\mathbf{S}$ and matrices of left and right singular vectors. Light color illustrates low-rank approximation.

2.3. Error estimation

Low-rank approximation matrix method that is described in this paper is a lossy compression technique. Several error metrics are used to control the quality of results.

- **Mean Square Error**

$$MSE = \frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n}(a_{ij} - a'_{ij})^2, \tag{5}$$

where a_{ij} represents an element of the original matrix and a'_{ij} represents an element of the reconstructed matrix of dimension $m \times n$.

- **Rooted Mean Square Deviation**

$$RMSD = \sqrt{MSE}. \tag{6}$$

- **Normalized Rooted Mean Square Deviation**

$$NRMSD = \frac{RMSD}{X_{max} - X_{min}} = \frac{\sqrt{MSE}}{X_{max} - X_{min}}, \tag{7}$$

where $X_{\min}$ and $X_{\max}$ are elements of input matrix **A** with minimum and maximum value, respectively. This error metric is able to measure and compare errors in datasets with different scales. Therefore, it is the main parameter that is used to control the quality of compression in the algorithm presented in this paper.

- **Peak Signal to Noise Ratio**

PSNR is most commonly used to measure the quality of reconstruction of lossy compression methods (e.g. image compression). The signal in this case is the original data, and the noise is the error introduced by compression. *PSNR* is an approximation to human perception of reconstruction quality. This metric is not so important in area of FEM analyses, where the human perception of visualizations is not as important as the exact mathematical accuracy of approximations. The reason to include *PSNR* in results is in particular to allow comparison with other image-related compression methods. *PSNR* is usually expressed in terms of the logarithmic decibel scale (dB)

$$\begin{aligned} PSNR &= 10\log_{10}\frac{(X_{max} - X_{min})^2}{MSE} = \\ &= 20\log_{10}\frac{X_{max} - X_{min}}{\sqrt{MSE}} = 20\log_{10}\frac{1}{NRMSD} = \\ &= -20\log_{10}NRMSD. \end{aligned} \tag{8}$$

- **Normalized Maximum Error**

$$NME = \frac{\|\mathbf{A}-\mathbf{A}'\|_{\max}}{X_{max} - X_{min}} = \frac{\max_{ij}(a_{ij} - a'_{ij})}{X_{max} - X_{min}}. \tag{9}$$

2.4. Randomized SVD

There are many algorithms with different approaches to compute singular value decompositions. One approach is based on diagonalization of the matrix which essentially yields the whole decomposition at the same time. The other approach is the use of an iterative algorithm that yields one or several singular values at a time and can be stopped after desired number of singular values and vectors has been computed. Although these algorithms have proven to work very well for relatively small matrices, they are not well suited for using with large data sets. The exact SVD of a $m \times n$ matrix has computational complexity O(min (mn^2, m^2n)) using the "big-O" notation. When applied on large data sets it tends to be very time-consuming. Also, the modern hardware architectures use caches to optimize reading of consecutive memory blocks. As these algorithms often need random access to the memory where the input matrix is stored, it can increase communication between different levels in memory hierarchy, which causes higher latency when accessing data. From a numerical linear algebra perspective, an additional problem resulting from increasing matrix sizes is that noise in the data, and propagation of rounding errors, become increasingly problematic.

In [12–15], there are described randomized methods for constructing approximate matrix factorizations which offer significant speedups over classical methods. The particular implementation of the randomized decomposition is based on the algorithm described in [16]. The authors proposed an algorithm for efficient computation of low-rank approximation to a given matrix. The method uses random sampling to identify a subspace that captures most of the action of a matrix. The input matrix is compressed to this subspace, and deterministic manipulations are then used to obtain the desired low-rank factorization. For a matrix that is too large to fit in fast memory, the randomized techniques require only a constant number of passes over the data, as opposed to O(k) passes for classical algorithms.

The algorithm can be split into two main computational stages. The first stage is to construct a low-dimensional subspace that captures the action of the matrix. To be more formal, this stage is to compute an approximate basis for the range of the input matrix **A**. This basis matrix **Q** is required to have orthonormal columns and

$$\mathbf{A} \approx \mathbf{Q}\mathbf{Q}^{\mathsf{T}}\mathbf{A}. \tag{10}$$

Matrix **Q** is desired to contain as few columns as possible while producing accurate approximation of matrix **A** at the same time.

The second stage is to use **Q** to obtain approximate SVD factorization of **A**. This can be achieved using simple deterministic steps:

1. Construct $\mathbf{B} = \mathbf{Q}^{\mathsf{T}}\mathbf{A}$.
2. Compute an exact SVD of the small matrix: $\mathbf{B} = \mathbf{W}\widetilde{\mathbf{S}}\widetilde{\mathbf{V}}^{\mathsf{T}}$.
3. Set $\widetilde{\mathbf{U}} = \mathbf{Q}\mathbf{W}$.

The main challenge is therefore to efficiently construct r orthonormal vectors forming the matrix **Q** that (nearly) span the range of **A**; r is the desired rank of approximation and is supposed to be substantially less then both dimensions of **A**. After that an SVD that closely approximates **A** can be constructed (closely in the sense that the spectral norm of the difference between **A** and the approximation to **A** is small relative to the spectral norm of **A**).

In order to estimate the range of matrix **A**, it is applied to a collection of r random vectors. The result of applying **A** to any vector is a vector in the range of **A**, and if the matrix is applied to r random vectors, the results will nearly span the range of **A** with extremely high probability. Mathematical proofs given in [16] and [19] show that the probability of missing a substantial part of the range of **A** is negligible if the vectors to which we apply **A** are sufficiently random (i.e. entries of these vectors are independent and identically distributed).

Therefore, the matrix **A** is applied to a random Gaussian matrix $\boldsymbol{\Omega}$ that contains r columns with random normally distributed entries yielding the matrix $\mathbf{Y} = \mathbf{A}\boldsymbol{\Omega}$. Applying the Gram-Schmidt process (or any other method for constructing QR decomposition) produces the decomposition $\mathbf{Y} = \mathbf{Q}\mathbf{R}$, where columns of **Q** are an orthonormal basis for the range of **Y**, and since columns of **Y** nearly span the range of **A, Q** is an orthonormal basis for the approximate range of **A**.

A is then decomposed as

$$\mathbf{A} \approx \mathbf{Q}\mathbf{Q}^{\mathsf{T}}\mathbf{A} = \mathbf{Q}\mathbf{B} = \mathbf{Q}\mathbf{W}\widetilde{\mathbf{S}}\widetilde{\mathbf{V}}^{\mathsf{T}} = \widetilde{\mathbf{U}}\widetilde{\mathbf{S}}\widetilde{\mathbf{V}}^{\mathsf{T}}. \tag{11}$$

The algorithm produces matrices $\widetilde{\mathbf{U}}$ and $\widetilde{\mathbf{V}}$ with orthonormal columns being approximations of the left and the right singular vectors of matrix **A**, and a nonnegative diagonal matrix $\widetilde{\mathbf{S}}$ that contains approximations of the first r singular values of matrix **A**. For a dense input matrix, randomized SVD algorithm requires O($mn\log r$) floating-point operations, substantially less than classical algorithms.

INPUT: maximum allowed error ($e : e > 0$), array with singular values ($S : S.length > 0$), element count ($c : c > 0$), maximum element value ($x_{\max}$), minimum element value ($x_{\min} : x_{\max} > x_{\min}$)
OUTPUT: rank of resulting matrix

```
procedure CalculateRank(e, S, c, x_max, x_min)
    MSE ← 0
    NRMSD ← 0
    rank ← S.length
    while NRMSD < e do                          ▷ repeat until max error is reached
        MSE ← MSE + S[rank]/c                   ▷ calculate MSE for current rank
        NRMSD ← sqrt(MSE)/(x_max − x_min)       ▷ normalize error
        rank ← rank − 1                         ▷ decrement rank for next loop
    end while
    return rank + 1                             ▷ Add one to not exceed maximum allowed error
end procedure
```

Algorithm 1. Calculation of rank for approximation matrix from maximum allowed error.

3. Implementation

Results from the finite element method are scalar, vector or tensor fields represented by discrete values calculated in nodes of the mesh or in integration points on finite elements. In order to compress data, an auxiliary matrix **A** has to be assembled from the results. The number of rows of the matrix **A** is equal to the number of incremental or time steps while the number of columns is equal to the number of points in which the results are stored. Such auxiliary matrix is assembled for each scalar field and for each component of the vector and tensor fields. It means, three matrices corresponding to the displacement in the x, y, and z directions are assembled for the vector of displacements in three-dimensional problems.

There are two main reasons to store particular results in separate matrices. First, the size of matrices is smaller than the size of a matrix which contains all results and therefore SVD will be performed faster. Second, the magnitudes of particular fields are very different (the stress tensor components are several order of magnitude larger than the components of the displacement vector) and the data compression algorithm would suppress the fields with small magnitudes. Once the matrix **A** is assembled for each field, the compression algorithm can be applied on it. It is purely algebraic procedure and no information about geometry of the mesh is needed.

Let us assume that the matrix is not empty and is full rank. Then it follows from the formula (4) that if r is equal to the rank of matrix **A**, the compression ratio is always higher than one. In other words the memory consumption of stored decomposition is bigger than the size of the original matrix. To make the compression algorithm applicable, the parameter r must satisfy the condition

$$r < \frac{mn}{m + n + 1}. \tag{12}$$

Considering the usual shape of matrix containing FEM results, this inequality is easily satisfiable even for the r being close to the rank of the original matrix as in the typical case the number of nodes or integration points is much higher than the number of analysis steps and therefore $m \ll n$.

3.1. Algorithm description

Once SVD is calculated, the compression algorithm removes a certain number of singular values and corresponding singular vectors. The remaining singular values and vectors represent the compressed data. There are two strategies that influence the way how to preserve the number of singular values – resulting size and quality. Each strategy is assigned a control parameter that determines compression ratio or approximation error.

Compression ratio. If the focus is only on the size of compressed data, the rank r of the approximation matrix can be calculated by the formula

$$r = \left\lceil c \times \frac{mn}{m + n + 1} \right\rceil, \tag{13}$$

where c is the compression ratio, $0 \leq c \leq 1$ (0 results in absolute compression while 1 results in no compression); $\lceil . \rceil$ is the ceiling function.

Approximation error. In a usual case, the most important measure to take into account is the approximation error. Algorithm is trying to minimize the compression ratio while at the same time ensuring that predefined approximation error threshold is not exceeded. To quantify the error, the Normalized root-mean-square deviation (*NRMSD*) is used. The normalized error metric enables working with various data sets that have different scales. *NRMSD* is defined in Section 2.3.

To effectively calculate the final rank of the approximation matrix

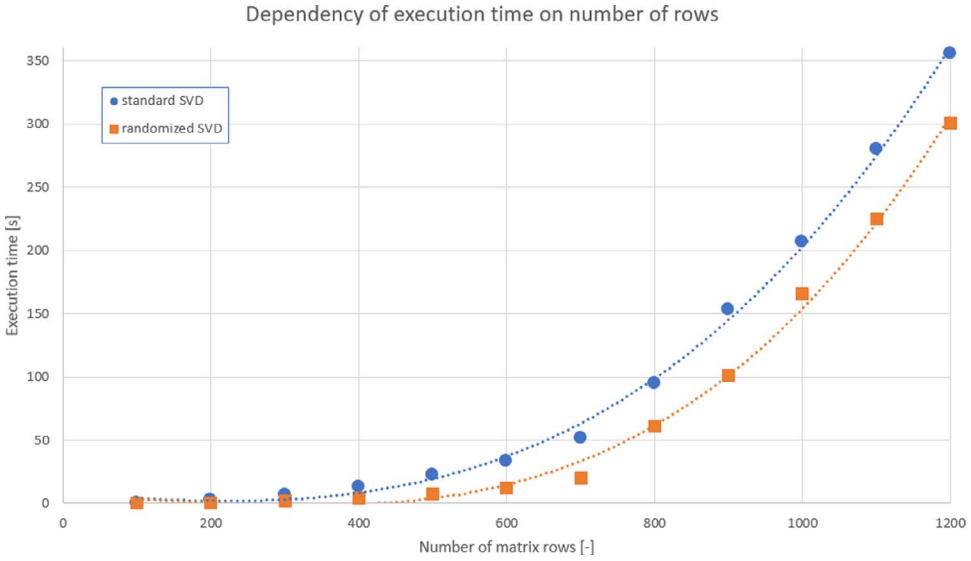

Fig. 2. Dependency of SVD execution time on m (having fixed $n = 10000$ and $r = min(m, n)$).

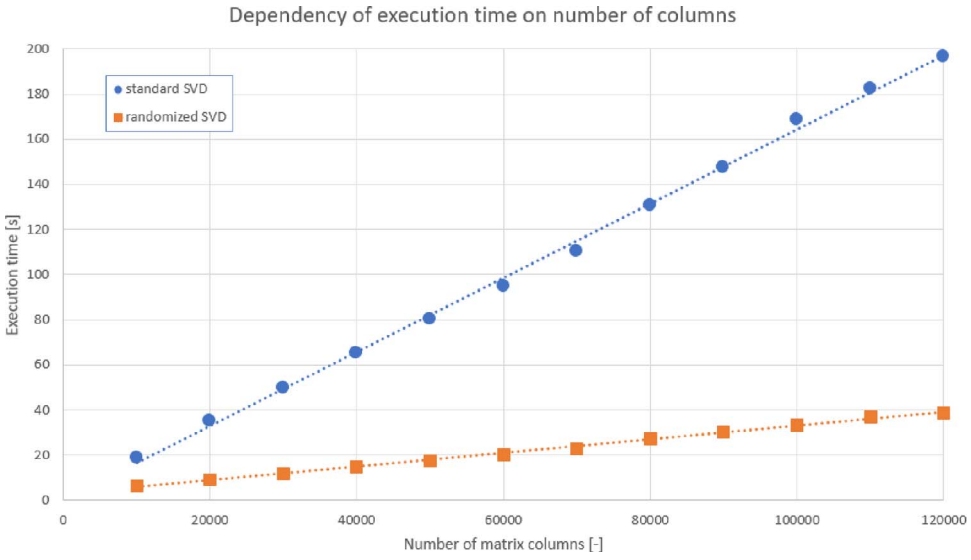

Fig. 3. Dependency of SVD execution time on n (having fixed $m = 100$ and $r = min(m, n)$).

from the desired approximation error, the interesting property of singular values

$$\sum_{i=1}^{m}\sum_{j=1}^{n}(a_{ij})^2 = \sum_{i=1}^{k} s_i^2, \tag{14}$$

where $k = \min(m, n)$, i.e. the smallest of two dimensions of the matrix **A**, is made use of. The above formula states that the sum of squared elements of the matrix **A** equals to the sum of squared singular values s_i of the same matrix **A**.

Using formulas (2) and (3) the equation (14) can be applied to the difference between original matrix **A** and approximation matrix **A′**

$$\sum_{i=1}^{m}\sum_{j=1}^{n}(a_{ij} - a'_{ij})^2 = \sum_{i=r+1}^{k} s_i^2, \tag{15}$$

where the term on the right-hand side is the sum of squares of those singular values of the matrix **A** that are going to be cut away by the compression algorithm. The equation can be rewritten using the definition of *MSE* in (5) to

$$MSE \times mn = \sum_{i=r+1}^{k} s_i^2 \tag{16}$$

and using (7) further to

$$(NRMSD \times (X_{max} - X_{min}))^2 \times mn = \sum_{i=r+1}^{k} s_i^2. \tag{17}$$

Then *NRMSD* can be used as a quality metric for the compression algorithm because normalization makes it usable for different datasets. Calculation of rank of the approximation matrix is depicted as pseudo-code in Algorithm 1. Algorithm uses the inequality

$$e > \frac{\sqrt{\frac{\sum_{i=r+1}^{k} s_i^2}{mn}}}{X_{max} - X_{min}} \tag{18}$$

to test whether the desired rank has been reached; e is *NRMSD* used as an error threshold that can not be exceeded to achieve desired quality of approximation.

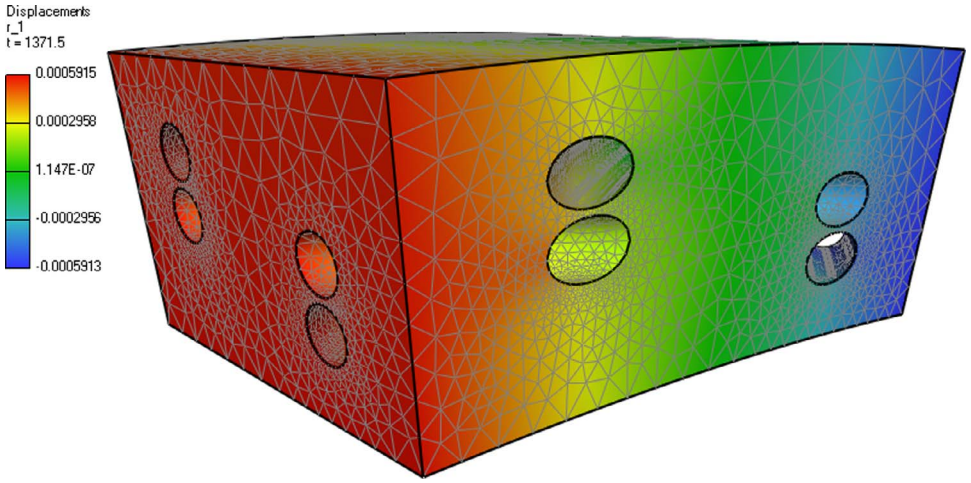

Fig. 4. Segment of reactor containment analyzed. Results visualization (displacement field, x component).

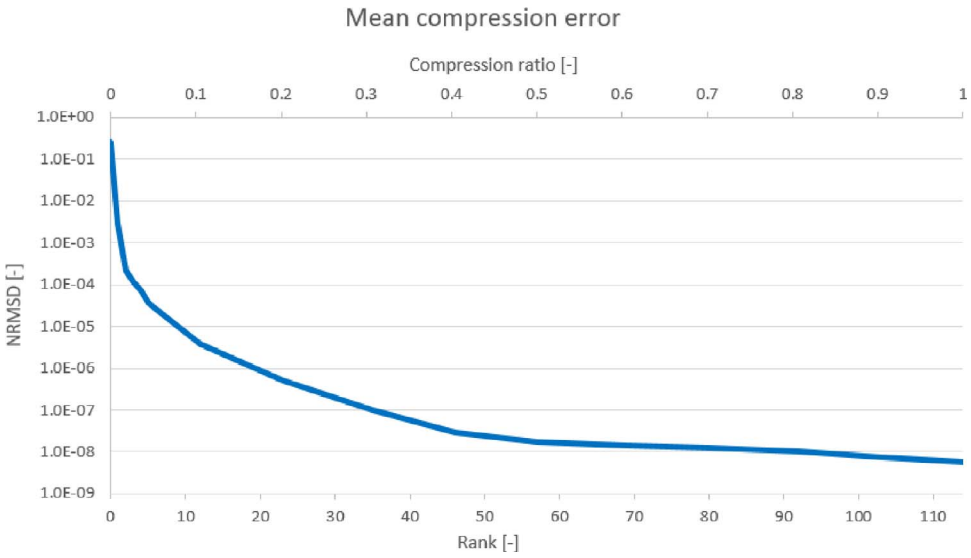

Fig. 5. Dependence of *NRMSD* on *c* and *r* for reactor containment analysis results.

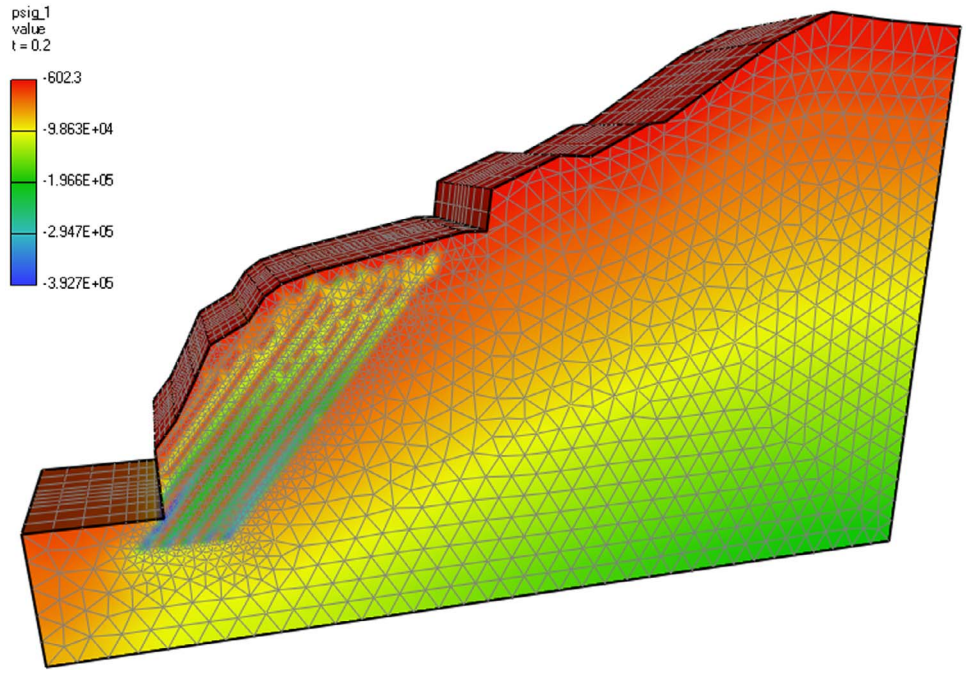

Fig. 6. Analysis of geological layers. Results visualization (stress field, sigma XX component).

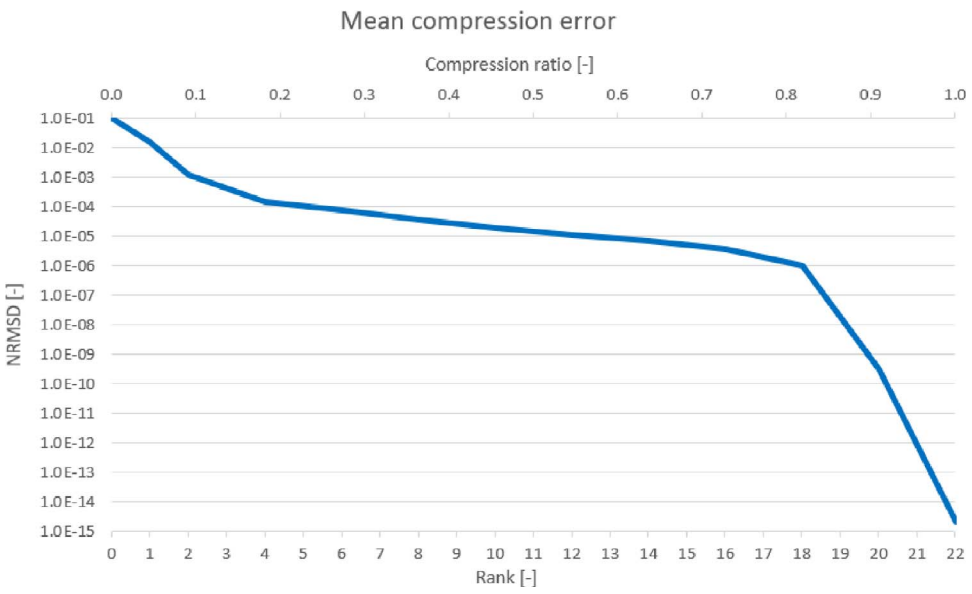

Fig. 7. Dependence of *NRMSD* on c and r for results of geological layers project.

3.2. Optimization

Computational complexity of the exact SVD algorithm is $O(m^2n)$, where $m < n$. This theoretical algorithm complexity is confirmed by two benchmarks where the dependency of the execution time on the varying matrix dimension is shown. The results of the benchmarks are depicted in Figs. 2 and 3. Several observations were made from the results:

- The algorithm is most efficient in cases where one dimension of the input matrix is very small compared to the other. However, this is almost always the case when compressing results from FEM – number of incremental or time steps seldom exceeds hundreds.
- Moreover, incremental or time steps can be devided into smaller ranges and the algorithm can be applied on each range separately. This will improve performance and can also increase quality of compression if the key time steps on the range boundaries are carefully selected.
- The randomized SVD algorithm has the same order of algorithmic complexity when full decomposition is required, but yet can significantly reduce execution time. However, the benchmarks are not designed to highlight the benefits of randomized SVD algorithms. The main advantage of the randomized SVD is in the ability to choose the rank of the approximation matrix in advance. In that case only limited number of singular values and corresponding singular vectors are calculated and algorithm performs much faster.

Storage size of SVD itself can also be optimized. **S**, being a diagonal matrix, can be stored as single list of singular values s_i, or can be even multiplied with the matrix of left singular vectors **U**.

4. Results

All procedures presented here were tested on a common PC having Intel Core i5-4690K @ 3.5GHz CPU with 16GB RAM, running on Microsoft Windows 10 64-bit operating system.

The first benchmark was designed to measure computational complexity of the SVD algorithm. Series of 100 random matrices with standard distribution were generated and execution times were recorded and averaged. The execution time with respect to the number of the stored incremental or time steps (the number of rows in the matrix **A**) is depicted in Fig. 2 while the execution time with respect to the

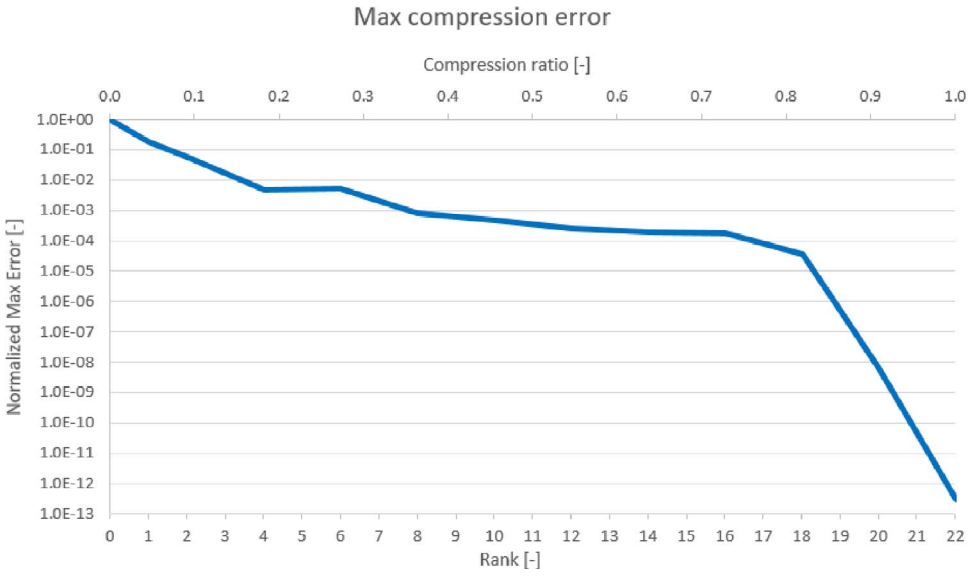

Fig. 8. Dependence of *NME* on c and r for results of geological layers project.

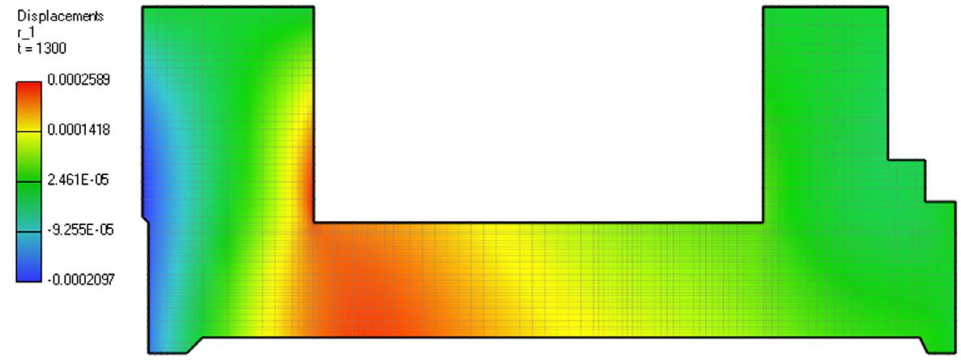

Fig. 9. 2D model of a reactor vessel. Results visualization (displacement field, x component).

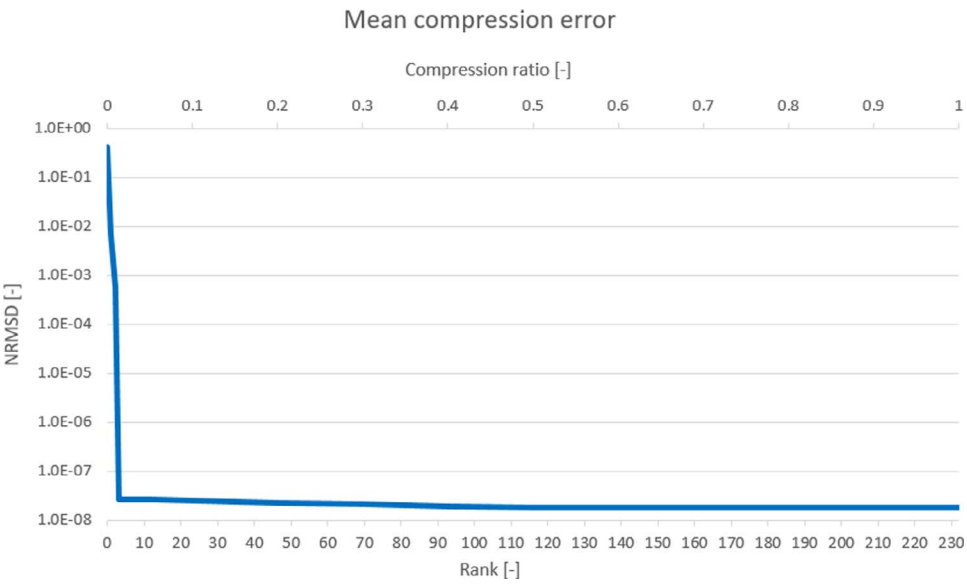

Fig. 10. Dependence of *NRMSD* on *c* and *r* for reactor vessel analysis results.

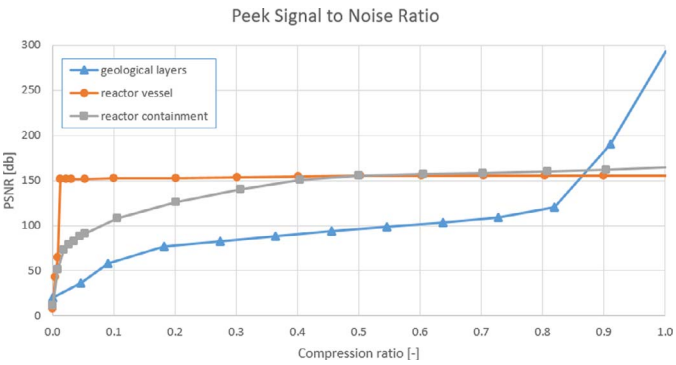

Fig. 11. Dependence of *PSNR* value on *c* and *r* calculated for different SVD decompositions.

number of points, where the results are stored (the number of columns of the matrix **A**), is depicted in Fig. 3. Especially in Fig. 3, it is clearly visible that the randomized SVD algorithm is much faster than the classical one.

These results confirm that the SVD implementation has computational complexity $O(m^2n)$, where m is number of rows, n is number of columns, and $m < n$. In case of $m > n$ the complexity would be $O(mn^2)$ as the algorithm takes advantage of non-squareness in that its complexity is quadratic only in the smaller dimension.

Behavior of the compression strategy introduced is presented on three real world examples. First example is an analysis of aging of nuclear power plant's containment made from prestressed concrete. Finite element mesh used in this analysis is in Fig. 4. More details about the analysis can be found in [8,9]. This analysis includes high number of analysis time steps (thousands) with very little differences between them. There is therefore potential for compression to be very effective (compression ratio to be very low) as proven in Fig. 5 that examines the impact of changes in the compression ratio to the mean error of approximation.

Fig. 6 shows results from an analysis of geological layers which was based on theory of plasticity. More details can be found in [10]. This project was chosen mainly to study behavior of compression algorithm when dealing with high discontinuities in data in spatial dimension (as can be seen in visualization). As summarized in Figs. 7 and 8 this has negligible effect (*NRMSD* and *NME* are bellow 1% even for very small *r* – 3 out of 22) on quality of compression.

Fig. 9 contains visualization of results of two-dimensional analysis, where axisymmetric description was used for analysis of aging of a reactor vessel. Details about the analysis can be found in [11]. There are exactly 232 analysis time steps. The resulting data has linear function character with several discontinuities in temporal dimension. There are few time steps in which resulting discrete functions have very different values compared to neighboring time steps. This was supposed to have negative impact on the quality of compression. However, as can be seen in Fig. 10, the quality is better than expected; e.g., if the rank of approximation matrix is set to 3 (compared to 232 being the rank of the original matrix) the normalized relative error (*NRMSD*) does not exceed 10^{-5}.

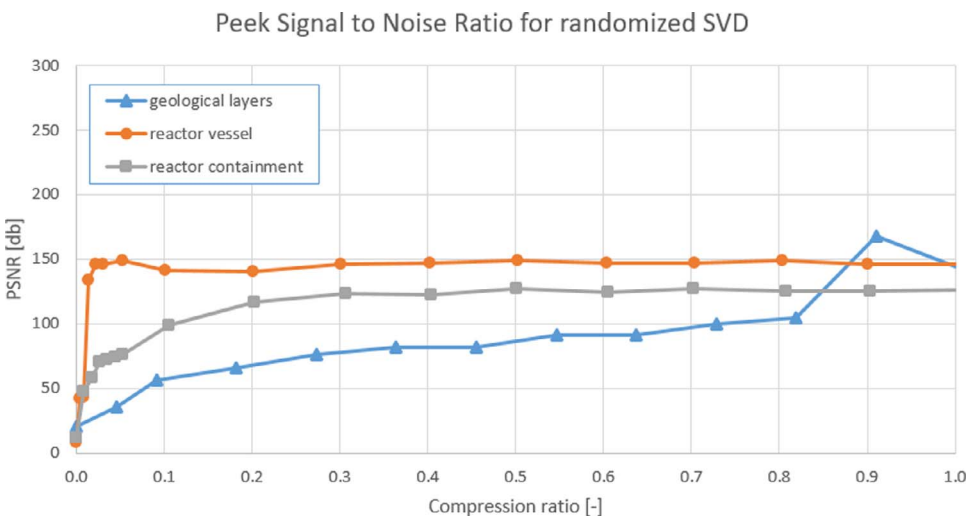

Fig. 12. Dependence of *PSNR* value on c and r calculated for different randomized SVD decompositions.

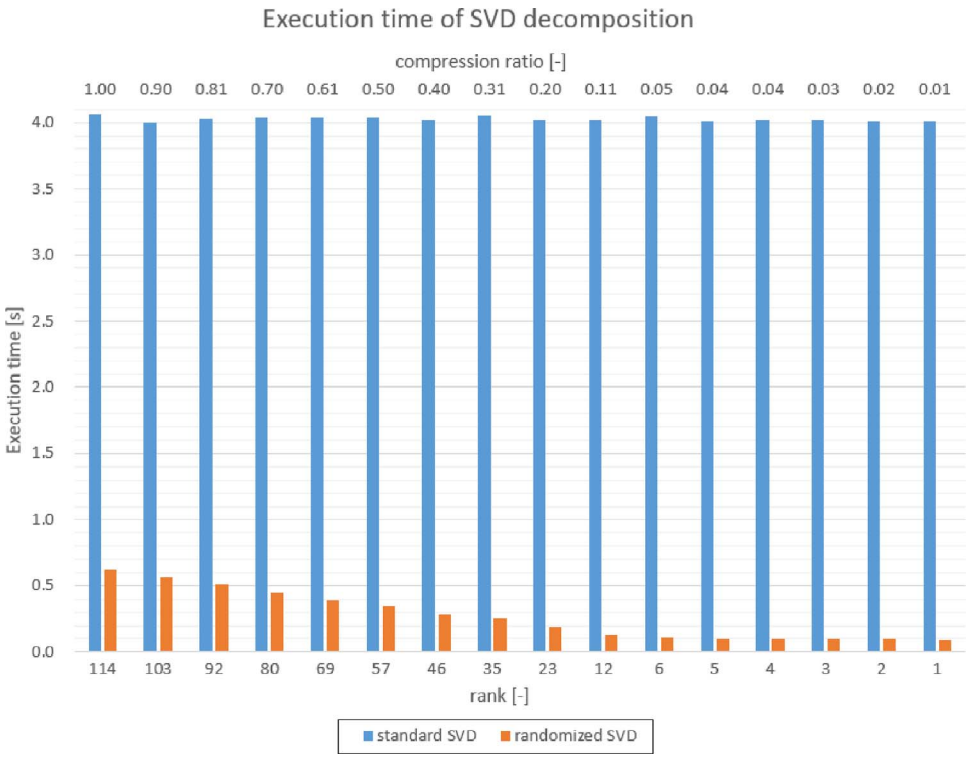

Fig. 13. Variation of execution time of standard and randomized SVD decompositions calculated for reactor containment analysis results.

Table 1
Memory consumption of compressed results. 3D reactor containment analysis.

Compression ratio (c)	Memory consumption [MB]	Size factor
1.00	2002.1	1
0.50	1006.9	0.5002
0.10	211.9	0.1053
0.01	35.3	0.0176

Fig. 11 summarizes the compression error for all three benchmarks using *PSNR* metric. *PSNR* is defined using logarithm (see Eq. (8) for definition), and is included here mainly as a comparison to other image-related compression methods whose quality is often expressed by *PSNR*. Fig. 12 contains the same information for the randomized SVD algorithm.

Besides the error also the execution speed of compression algorithm was measured. In Fig. 13, there is a comparison of execution times for standard versus randomized SVD compression algorithms. Interestingly, execution time of standard SVD is independent of target rank whereas execution time of randomized SVD decreases linearly with decreasing target rank. If the rank is known ahead, the fact can be taken advantage of.

The memory consumption of compressed results for reactor containment analysis is summarized in Table 1. For different values of compression ratio it shows memory size in megabytes. In this benchmark, the compression ratio c is an input parameter to the compression algorithm. As follows from the Eq. (13), the value of the compression ratio directly affects the amount of singular values ought to be removed from SVD. Size factor describes the final outcome of compression when compared to the original size.

5. Conclusion

The goal of this paper is to present the use of SVD in compression of results from the Finite Element Method. Implementation details of the compression algorithm have been provided, and the results of its application on real data have been presented. The implemented algorithm is able to compress arbitrary data using low-rank approximation matrices. When the maximum allowed error was set to 10^{-5}, the compression ratio was at most 10% for all tested results. In many cases compression ratio can be even better – bellow 1% of the original size. The important property of the compression algorithm is the fact that the approximation error can be set in advance and there is a guarantee that it will not be exceeded.

The main disadvantage is the computational complexity of the compression algorithm. SVD is a very time-consuming operation. However, this operation is performed only once after FEM analysis is finished and before the post-processing is started. Also, the randomized version of the decomposition algorithm is much faster and can be used if a slight increase of approximation error is tolerated.

Another complication is the necessity to use a special format to store compressed data in the form of matrix decompositions, and the post-processor must be updated to use this new format. Post-processor must perform matrix multiplication to get the original results. However, in a usual case the data for one analysis step are needed at a time, i.e. multiplication of a single row is enough. Detailed description of the post-processor implementation is beyond the scope of this paper.

Acknowledgment

Financial support for this work was provided by project number P105/12/G059 of Czech Science Foundation. The financial support is gratefully acknowledged.

References

[1] Beneš Š, Kruis J. Approximation of large data from the finite element analysis allowing fast post-processing. Adv Eng Softw 2016;97:17–28. https://doi.org/10.1016/j.advengsoft.2016.02.008.

[2] Lui C. A study of the jpeg-200 image compression standard. Master thesis. Queen's University, Ontario, Canada; 2001.

[3] Watson A. Image compression using the discrete cosine transform. Math J 1994;4(1):81.

[4] Baker K. Singular value decomposition tutorial. 24. The Ohio State University; 2005.

[5] Kalman D. A singularly valuable decomposition: the SVD of a matrix. Coll Math J 1996;27(1):2–23. https://doi.org/10.2307/2687269.

[6] Golub G, Van Loan C. Matrix computations. Baltimore, London: The Johns Hopkins University Press; 3 ed.0-8018-5414-8; 1996.

[7] SairaBanu J, Babu R, Pandey R. Parallel implementation of singular value decomposition (SVD) in image compression using OpenMP and sparse matrix representation. Indian J Sci Technol 2015.

[8] Kruis J, Koudelka T, Krejčí T. Hygro-thermo-mechanical analysis of a reactor vessel. Acta Polytech J Adv Eng 2012;52(6/2012):67–73. ISSN 1210-2709, e-ISSN 1805-2363.

[9] Koudelka T, Krejčí T, Šejnoha J. Analysis of a nuclear power plant containment. In: R. B.B.H.V. Topping, L.F. Costa Neves, editors. Proceedings of the twelfth international conference on civil, structural and environmental engineering computing. Stirlingshire, UK: Civil-Comp Press Ltd; 2009. Paper 132. doi:10.4203/ccp.91.132.

[10] Koudelka P, Koudelka T. Risk assessment of a heterogeneous stratified rock cliff under an elevated road. XIIIth Danube European conference on geotechnical engineering. 2. Ljubljana: Slovenian Geotechnical Society; 2006. p. 56–61.

[11] Kruis J, Koudelka T, Bittnar Z, Petkovski M. Hygro-thermo-mechanical analysis of a nuclear power plant prestressed concrete reactor vessel. In: Topping BHV, editor. Proceedings of the tenth international conference on civil, structural and environmental engineering computing. Stirling, Scotland, UK: Civil-Comp Press Ltd1-905088-01-9; 2005.

[12] Candès E, Li X, Ma Y, Wright J. Robust principal component analysis? J ACM (JACM) 2011;58(3):11.

[13] Woolfe F, Liberty E, Rokhlin V, Tygert M. A fast randomized algorithm for the approximation of matrices. Appl Comput Harmon Anal 2008;25(3):335–66. http://dx.doi.org/10.1016/j.acha.2007.12.002.

[14] Martinsson P, Rokhlin V, Tygert M. A randomized algorithm for the decomposition of matrices. Appl Comput Harmon Anal 2011;30(1):47–68. https://doi.org/10.1016/j.acha.2010.02.003.

[15] Szlam A, Kluger Y, Tygert M. An implementation of a randomized algorithm for principal component analysis. J ACM (JACM) 2014;1(1).

[16] Halko N, Martinsson P, Tropp J. Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions. SIAM Rev 2011;53(2):217–88.

[17] Ivanyi P. Finite element mesh conversion based on regular expressions. Adv Eng Softw 2012;51:20–39. http://dx.doi.org/10.1016/j.advengsoft.2012.05.002.

[18] Ivanyi P. Parallel conversion of finite element meshes. Pollack Periodica 2014;9(3):89–102. http://dx.doi.org/10.1556/Pollack.9.2014.3.10.

[19] Witten R, Candes E. Randomized algorithms for low-rank matrix factorizations: sharp performance bounds. Algorithmica 2015;72(1):264–81.