Four algorithms to solve symmetric multi-type non-negative matrix tri-factorization problem

Rok Hribar · Timotej Hrga · Gregor Papa · Gašper Petelin · Janez Povh · Nataša Pržulj · Vida Vukašinović

31 August 2021

Abstract In this paper, we consider the symmetric multi-type non-negative matrix tri-factorization problem (SNMTF), which attempts to factorize several symmetric non-negative matrices simultaneously. This can be considered as a generalization of the classical non-negative matrix tri-factorization problem and includes a non-convex objective function which is a multivariate sixth degree polynomial and a has convex feasibility set. It has a special importance in data science, since

R. Hribar
Computer Systems Department, Jožef Stefan Institute, Ljubljana, Slovenia
Jožef Stefan International Postgraduate School, Ljubljana, Slovenia
E-mail: rok.hribar@ijs.si *Corresponding author

T. Hrga
University of Ljubljana, Faculty of Mechanical Engineering, Ljubljana, Slovenia

G. Papa
Computer Systems Department, Jožef Stefan Institute, Ljubljana, Slovenia

G. Petelin
Computer Systems Department, Jožef Stefan Institute, Ljubljana, Slovenia
Jožef Stefan International Postgraduate School, Ljubljana, Slovenia

J. Povh
University of Ljubljana, Faculty of Mechanical Engineering, Ljubljana, Slovenia
Institute of Mathematics, Physics and Mechanics, Ljubljana, Slovenia

N. Pržulj
Institute of Mathematics, Physics and Mechanics, Ljubljana, Slovenia
Department of Computer Science, University College London, London, United Kingdom
Barcelona Supercomputing Center, Barcelona, Spain
ICREA, Barcelona, Spain

V. Vukašinović
Computer Systems Department, Jožef Stefan Institute, Ljubljana, Slovenia

The published version of this AM can be found at: https://link.springer.com/article/10.1007/s10898-021-01074-3
DOI: https://doi.org/10.1007/s10898-021-01074-3
it serves as a mathematical model for the fusion of different data sources in data clustering.

We develop four methods to solve the SNMTF. They are based on four theoretical approaches known from the literature: the fixed point method (FPM), the block-coordinate descent with projected gradient (BCD), the gradient method with exact line search (GM-ELS) and the adaptive moment estimation method (ADAM). For each of these methods we offer a software implementation: for the former two methods we use Matlab and for the latter Python with the TensorFlow library.

We test these methods on three data-sets: the synthetic data-set we generated, while the others represent real-life similarities between different objects.

Extensive numerical results show that with sufficient computing time all four methods perform satisfactorily and ADAM most often yields the best mean square error (MSE). However, if the computation time is limited, FPM gives the best MSE because it shows the fastest convergence at the beginning.

All data-sets and codes are publicly available on our GitLab profile.

**Keywords** Non-negative matrix factorization · Fixed point method · Block coordinate descent · Projected gradient method · ADAM

1 Introduction

1.1 Motivation

With the expand of information technology the amount of data generated within different areas of our life has remarkably increased. As a consequence, the need to extract meaningful information out of the collected data has significantly grown and the knowledge discovery has become widely studied research area. When we have different sources of data at disposal, a deeper insights can only come from a systems-level integration of all the available data. The major challenge is to design integrative and predictive heuristic computational methods that can collectively mine all the available data and can, as a result, enable the extraction of meaningful information.

Medicine and healthcare are exemplary areas where vast amounts of data has been collected and which attracts big attention of scientific community. Diverse molecular and clinical data describe various aspects of the human cells functioning and offer an unprecedented opportunity to revolutionize the biological and healthcare understanding. Hence, methods which can enable the extraction of a discernible biological meaning from various multi-type, systems-level and high-throughput (so called “omic”) networked molecular and clinical data [45] are needed.

Recently, a non-negative matrix factorization approach for network data integration (also called data fusion) was proposed [18,19,20,63]. A new computational framework that is capable of harnessing this unprecedented opportunity provided by the wealth of data is based on a non-negative matrix tri-factorization (NMTF) [57,59]. In a nutshell, NMTF approximates a high-dimensional data matrix $R$, containing associations of $n_1$ data points of data type 1 with $n_2$ data points of data type 2, into a product of three, low-dimensional, non-negative matrix factors.
$R \approx G_1 S G_2^\top$ by solving the following optimization problem:

$$
\min \left\{ \|R - G_1 S G_2^\top\|^2 : G_1, G_2, S \geq 0 \right\}, \tag{NMTF}
$$

where $\| \cdot \|$ is Frobenius norm in $\mathbb{R}^{n_1 \times n_2}$.

The key different between the non-negative matrix factorization and some other matrix factorization methods, like the spectral decomposition of symmetric matrices or singular value decomposition of general matrices is the non-negativity constraint for all factors. It drastically improves the usability and interpretability of the results in the context of data science.

The low-dimensional matrix factors $G_1$ and $G_2$ can be used to assign $n_1$ data points into $k_1 \ll n_1$ clusters and $n_2$ data points into $k_2 \ll n_2$ clusters, respectively. This is known as the clustering property of the NMTF [13]. The matrix $S$ is in this context a $k_1 \times k_2$ compressed representation of $R$ and $G_1$ and $G_2$ imply a co-clustering. The reconstructed data matrix $G_1 S G_2^\top$ is usually more complete then the initial data matrix $R$, featuring new entries, unobserved in the data, that emerged from the latent structure captured by the low-dimensional matrix factors. The observed property is known as the matrix completion property of the NMTF [8].

1.2 Problem formulation

The central problem that we study in this paper is symmetric multi-type non-negative matrix tri-factorization (SNMTF) problem. The input for this problem is an $N$-tuple of symmetric non-negative matrices $(R_1, \ldots, R_N)$, which we want to factorize simultaneously by solving:

$$
\min \left\{ \sum_{i=1}^{N} \|R_i - GS_i G_i^\top\|^2 : G, S \geq 0, \ S_i = S_i^\top \right\}. \tag{SNMTF}
$$

The SNMTF can be considered as a specialisation of the NMTF since we have here $G_1 = G_2$. On the other hand, we consider more than one input matrix simultaneously, so SNMTF is in this sense also a generalization of NMTF Problem SNMTF is also a generalization of the symmetric non-negative matrix factorization problem [24, 31], which can be formulated as:

$$
\min \ \{ \|R - GG^\top\|^2 : G \geq 0 \}. \tag{SNMF}
$$

Generalisation goes into two directions: in [SNMTF] we consider more than one matrix to be (simultaneously) decomposed and we allow a third factor $S$ to be sandwiched between the factors $G$ and $G^\top$. As a consequence the existing methods for solving NMTF and SNMF are not straightforwardly applicable. The work of Ding et al. [13] could be considered as an important step in this direction since they consider [SNMTF] with orthogonality constraint $G^\top G = I$ and with only one input matrix. However, they focus only on the fixed point method (the first one of the four methods that we study in this paper) and their multiplicative rules are slightly different due to considering the orthogonality of columns of $G$ at the same time.
The SNMTF is suitable for modelling the clustering problem with a single set of data points and with different types of associations between them. Each matrix $R_i$ contains one type of associations among the data points. This form is especially beneficial when there are different ways to measure associations, which complement each other. Important example is clustering of genes where several modalities of associations exist: gene co-expression, genetic similarity, protein-protein interaction, etc. Knowledge from each type of associations are combined to generate matrix $G$ that contains information about general properties of all associations, while condensed information about the different types of associations is captured in the matrices $S_i$. For example, the problem SNMTF serves, after omitting the constraint $S_i \geq 0$, as the underlying mathematical model in the data-integrated cell in [34].

In this paper we use two measures for factorization quality of feasible solutions for SNMTF: the square error (SE) and mean square error (MSE), as follows:

$$SE = \sum_i \| R_i - GS_i G^T \|^2, \quad (1)$$

$$MSE = \frac{\sum_i \| R_i - GS_i G^T \|^2}{\sum_i \| R_i \|^2}. \quad (2)$$

The first measure is actually the objective function of SNMTF, while the second is a relative value of the SE compared to the size of the input data. We use SE to define algorithms (gradients, steps sizes), while MSE is used as one of the stopping criteria.

The existing complexity results show that SNMF is NP-hard since the NP-hard problem whether given non-negative matrix is completely positive [12] can be reduced to solving SNMF. The general non-negative matrix factorization problem is also NP-hard [55], but for the case of the SNMTF the complexity is to the best of our knowledge not known. Nevertheless, the SNMTF is a non-convex optimization problem with the objective function being a multivariate polynomial of degree 6, where the matrix variables $G$ and $S_i$ must belong to the cones of non-negative matrices, so we conjecture that it is also in the class of NP-hard problems, which is supported by numerical evidence in this paper.

1.3 Our contribution

The central research question that we address in this paper is how SNMTF can be efficiently solved with state-of-the-art mathematical optimization tools and methods.

The main contributions of this paper are:

– We develop four algorithms to solve SNMTF. They are problem-specific adaptations of four well-known methods: the fixed point method (FPM), block-coordinate descent with projected gradient (BCD), gradient method with exact line search (GM-ELS) and adaptive moment estimation method (ADAM);
– We develop and publish efficient implementations of all four algorithms;
– We provide and make publicly available three benchmark data-sets consisting of (i) synthetic matrices with known optimum solutions for SNMTF and (ii) real data-sets from machine learning and (iii) from biology;
We provide extensive numerical evaluations of our code on these data-sets, which are the bases for our recommendations on which algorithm (and code) to use to solve SNMTF on given data-set.

The layout of the paper is as follows. A literature review on the topic is given in Section 2. Section 3 presents adaptations of FPM and BCD for SNMTF while GM-ELS and ADAM are presented in Section 4. The implementations of the proposed algorithms are described in Section 5 and analysed in Section 6. The implications of this work are discussed in Section 7.

2 Related work

Although SNMTF can be treated as an exciting example of non-linear optimization, the mathematical optimization community did not invest much effort in solving it. Several researchers have studied other formulations of non-negative matrix tri-factorization problem and how to solve them using the fixed point method (FPM) with a different variant of the multiplicative updates.

The literature search shows that non-negative matrix factorization got its visibility and popularity mainly because of data science community, especially because of the clustering-like interpretations of non-negative factors. Most of the results pertain to the classical (2-factor) non-negative matrix factorization problem, which can be formulated as:

$$\min \{\| R - GF^\top \|_2^2 : G, F \geq 0 \}. \quad \text{(NMF)}$$

The dissertation [24] reviews different algorithms for solving NMF and its generalizations. As an alternative, it suggests Rank-one residue iteration, an algorithm of low complexity which is reported to have fast convergence. The book [9] contains comprehensive overview of results up to the year 2009. Another relevant survey with reviews of some standard algorithms for NMF can be found in [10]. Other optimization methods applied to the non-negative matrix-factorization problems can be found in Lee and Seung [28], where a diagonally rescaled gradient method for basic NMF problem is studied. Lin [29] studied how to solve NMF using a (projected) gradient method. An alternating, non-negative, least-squares algorithm with the active-set method is presented in [26].

In [17,18,19,20] big effort to network data fusion using non-negative matrices has been invested, but only relaxations of the NMTF and SNMTF have been studied and only FPM was used. In [33] the authors used a relaxation of the SNMTF obtained by omitting the constraints \( S_i \geq 0 \) to develop computational model called iCell which integrates different molecular interaction network types and provides new biological and medical insights related to cancer.

In [56] the authors compared different approaches for solving the NMTF problem on six large data-sets. Comparing alternating least squares, projected gradients, and coordinate descent methods they concluded that methods based on projected gradients and coordinate descent converge up to twenty-four times faster than multiplicative update rules and coordinate descent-based NMTF converges up to sixteen times faster compared to well-established methods.
In [11,30] authors studied the NMTF problem with orthogonality constraints. In [11] the authors suggested a process of identifying a clearer correlation structure represented by the block matrix on two different NMF algorithms. The results showed that in most cases, the quality of the obtained clustering increases, especially in terms of average inter-cluster similarity. In [30] the authors propose an L1-norm symmetric non-negative matrix tri-factorization method to solve the high-order co-clustering problem that aims to cluster multiple types of data simultaneously by utilizing the inter- or/intra-type relationships across different data types. Due to orthogonal constraints and symmetric L1-norm formulation they derived the solution algorithm using the alternating direction method of multipliers.

Ding et al. [13] studied several versions of tri-factorization problems including a problem similar to the SNMTF. However, only the problem with one input data matrix is studied, the orthogonality constraint for the factor $G$ is included, and only the FPM is used to obtain solutions that might not even be local optima. This is probably the reason why they obtain the multiplicative update rule for $G$, which is slightly different from our update rules (16).

Wang et al. [60] and Žitnik et al. [63] studied a simplified version of the SNMTF with only one data matrix $R$. Again they used only the most straightforward formulation of FPM to obtain solutions close to a local minimum and to perform network-data fusion to predict the gene function. Another similar study can be found in [59], where a multi-type relational data is mined by solving a simplified version of the SNMTF. These authors again used only FPM with adapted variants of the updating rules.

Non-negative matrix factorization approach can be found also in the evolutionary clustering. In [62] authors proposed a framework of evolutionary clustering based on graph regularised non-negative matrix factorization, to detect dynamic communities and the evolution patterns and to predict the varying structure across the temporal networks. The proposed approach has better performance on community detection in temporal networks compared to some widely used models based on evolutionary clustering and heuristic methods. In [58] evolutionary non-negative matrix factorization was proposed, which incrementally updates the factorized matrices in a computation and space efficient manner with the variation of the data matrix. In [32] they proposed semi-supervised evolutionary non-negative matrix factorization approach for detecting dynamic communities. The main advantage of the proposed approach is to escape the local optimum solution without increasing time complexity. In [48], a time evolving non-negative matrix factorization was used for time-sequential matrix data to track the time-evolution of data clusters.

Bayesian semi-non-negative matrix tri-factorization method to identify associations between cancer phenotypes e.g., molecular sub-types or immunotherapy response, and pathways from the real-valued input matrix, e.g., gene expressions is proposed in [41,42].

### 3 Fixed point method and block-coordinate descent approach to SNMTF

In this section, we describe two out of the four methods which we devise to solve SNMTF, the fixed point method (FPM) and the block coordinate descent with projected gradient (BCD). Both methods try to find a local minimum for the
problem by addressing the Karush-Kuhn-Tucker (KKT) conditions. Note that the fixed point method could be also treated as a variant of block-coordinate descent method since in each iteration we make an update only on one block of variables keeping the other fixed. However, we decided to keep this notation since it is most common in the literature.

3.1 Karush-Kuhn-Tucker conditions of SNMTF

The KKT conditions consist of equations and inequalities that describe necessary conditions for local minima for the SNMTF. The FPM and BCD methods try to find a solution for these conditions and this way a good approximation for a local minimum of SNMTF.

We start with the Lagrangian function for the SNMTF:

\[ L(G, S, \beta, \gamma) = \sum_i \| R_i - GS_i G^T \|^2 - \langle G, \beta \rangle - \sum_i \langle S_i, \gamma_i \rangle. \] (3)

Matrices \( \beta \) and \( \gamma_i \) are the Lagrangian multipliers (dual variables) for the non-negativity constraints \( G \geq 0 \) and \( S_i \geq 0 \), respectively. By the definition of the SNMTF, the matrices \( R_i \) and \( S_i \) are symmetric. Hence, Lagrangian multipliers \( \gamma_i \) for \( S_i \) are also symmetric and the Karush-Kuhn-Tucker conditions for the SNMTF are:

- **Stationarity**
  \[ \nabla_G L = -4 \sum_i R_i GS_i + 4 \sum_i GS_i G^T GS_i - \beta = 0 \] (4)
  \[ \nabla_{S_i} L = -2G^T R_i G - 2G^T GS_i G^T G - \gamma_i = 0 \] (5)

- **Primal feasibility**
  \[ G \geq 0, \quad S_i \geq 0, \quad \forall i \] (6)

- **Dual feasibility**
  \[ \beta, \quad \gamma_i \geq 0, \quad \forall i \] (7)

- **Complementary slackness**
  \[ \langle \beta, G \rangle = 0 \] (8)
  \[ \langle \gamma_i, S_i \rangle = 0, \quad \forall i \] (9)

By \( \nabla_G L \) and \( \nabla_{S_i} L \) we denoted the gradients of \( L \) in variables \( G \) and \( S_i \), respectively.

Since the Slater constraint qualification is trivially satisfied for the constraints of SNMTF it holds from the Lagrangian theory (see, e.g., [6]) that any candidate for a local minimum of the SNMTF must satisfy the necessary conditions which consist of (3), (4), and:
The FPM and BCD methods devised for the SNMTF try to find a solution satisfying (6)–(11) conditions. Both have already been used for standard non-negative matrix factorization problems \cite{3,13,37,38} and in this paper we adapt them to work for the SNMTF.

3.2 Fixed point method (FPM)

The fixed point method is a standard and the most common approximation method to solve different variants of the NMF problem. It is reduced to the so-called multiplicative update rules, see, e.g., \cite{17,57}. In the specific case of SNMTF conditions (6)–(11) imply:

\[
\beta = \sum_i \left( -4R_i GS_i + 4GS_i G^\top GS_i \right) \quad (10)
\]
\[
\gamma_i = -2G^\top R_i G + 2G^\top GS_i G^\top G, \quad \forall i \quad (11)
\]

The FPM and BCD methods devised for the SNMTF try to find a solution satisfying (6)–(11) conditions. Both have already been used for standard non-negative matrix factorization problems \cite{3,13,37,38} and in this paper we adapt them to work for the SNMTF.

In the terms above we use $\circ$ to denote the Hadamard (element-wise) matrix product. Since $G \geq 0$, $S_i \geq 0$, $\forall i$, we can substitute $G \rightarrow G \circ G$, $S_i \rightarrow S_i \circ S_i$ in (12)–(13). Therefore, (12) is equivalent to

\[
G \circ \beta = G \circ \sum_i \left( -4R_i GS_i + 4GS_i G^\top GS_i \right) = 0,
\]
\[
S_i \circ \gamma_i = S_i \circ \left( -2G^\top R_i G + 2G^\top GS_i G^\top G \right) = 0, \quad \forall i.
\]

After division with the term on the left and by applying the square root on both sides we obtain the following multiplicative update rule for $G$:
Four algorithms to solve symmetric multi-type NMTF problem

Fig. 1 Fixed point method to solve SNMTF

We could use also the reverse order - division with the term on the right, but this yields in practice worse convergence.

Note that if we start with non-negative $G$ and $S_i$, then we retain their non-negativity throughout the iterations, and consequently the products of matrices in (14)–(15) are non-negative (recall, $R_i$ are assumed to be non-negative), so all terms $(-)$ are zero and these multiplicative update rules simplify to

$$S_i \leftarrow S_i \odot \frac{(G^\top G S_i G^\top G)^{-} + (G^\top R_i G)^{+}}{(G^\top G S_i G^\top G)^{+} + (G^\top R_i G)^{-}}. \quad (15)$$

3.3 Block-coordinate descent method with projected gradient method (BCD)

In this subsection we show how to apply to the SNMTF a well-known block-coordinate descent method, also known as non-linear Gauss-Seidel method, see e.g. [6,53,61]. We can consider SNMTF as an optimization problem in $(1 + N)$-block variables $(G, S_1, S_2, \ldots, S_N)$ that must belong to the cone $\mathbb{R}_n^{n \times k} \times S_k^N$. We separate the feasible set into direct product of two cones, one consisting of $\mathbb{R}_+^{n \times k}$ and the other consisting of $(S_k)^N$. We optimize $G$, while $\{S_i\}$ are fixed and vice versa. Note, that for fixed $G$, the SNMTF problem decomposes into $N$ convex quadratic subproblems in nonnegative variables $S_i$. Therefore, we propose a 2-block coordinate descent algorithm, as described in Figure 2.

The optimization problem in Step 2.1 in Figure 2 is “easy”, which means that it boils down to solving $N$ convex quadratic optimization problems over the non-negative orthants. We have a large variety of iterative methods to solve them, including interior-point methods, projected gradient methods, active set methods etc., for a brief and a comprehensive overview on different special types of quadratic optimization problems, methods to solve them and state-of-the-art solvers see [12]. We decided to use a simple projected gradient method with exact line search, since

$$G \leftarrow G \odot \left( \sum_i R_i G S_i \right) \odot \left( \sum_i G S_i G^\top G S_i \right), \quad (16)$$

$$S_i \leftarrow S_i \odot \left( G^\top R_i G \right) \odot \left( G^\top G S_i G^\top G \right), \quad (17)$$

where $\odot$ denotes the element-wise division. The fixed-point method for the SNMTF is summarised in Figure 1.
Fig. 2 2-block coordinate descent algorithm

| Input: Non-negative symmetric $R_1, \ldots, R_N$. |
| 1. **Initialisation**: Compute initial values for $G$. |
| 2. While termination test not satisfied |
| 2.1 Compute $(S_1, \ldots, S_N)_{\text{new}} = \text{argmin} \left\{ \sum_{i=1}^{N} \| R_i - G S_i G^\top \|_2^2 \mid S_i \geq 0, S_i = S_i^\top \right\}$. |
| 2.2 $(S_1, \ldots, S_N) = (S_1, \ldots, S_N)_{\text{new}}$. |
| 2.3 $G_{\text{new}} = \text{argmin} \left\{ \sum_{i=1}^{N} \| R_i - G S_i G^\top \|_2^2 \mid G \geq 0 \right\}$. |
| 2.4 $G = G_{\text{new}}$. |

It is very easy to implement and it works very well. We tried also to use existing solvers for such problems, like the `quadprog` function in Matlab [36] or `mskqpopt` from MOSEK, but the time performance was much worse.

The problem in Step 2.3 is more involved. It is a non-convex programming problem since the objective function is multivariate polynomial of order $4$ in matrix variable $G$. We decided to use projected gradient method, where the step-length could be determined either using Armijo rule or by exact line search. After extensive testing we decide for the latter. For example, the exact line search for the update of $G$ in the direction of gradient $\Delta G = \nabla G_{\text{SE}}$ boils down to finding optimum $t$ such that the univariate polynomial of degree $4$

$$p(t) = \sum_i \| R_i - (G + t \Delta G) S_i (G + t \Delta G)^\top \|_2^2$$

is minimum.

We compute coefficients of $p(t)$ by writing them in closed form formulae and computing them exactly, using all available information, like symmetries of $R_i$ and $S_i$. Note that we also tried to compute the coefficients of $p(t)$ approximately by evaluating $p(t)$ on at least $5$ points and using curve fitting methods, like `polyfit` in Matlab. However, this method finds best curve in a least-squares sense, which in practice gives almost exact values for the coefficients of the highest degree terms (for $t^4$ and $t^3$) which were also the largest, while the linear term coefficient usually deviates a lot from the exact value. Since this coefficient plays an important role in computing the minimum of $p(t)$, which is usually close to $0$, we decided not to use this approach. Similar behaviour was observed with the Gradient method with exact line search (GM-ELS) using Python and TensorFlow in Section 4.

Once we know $p(t)$, we compute its minimum numerically by the Matlab function `fminbnd` over the interval $[-1, 0]$. If $t_{\text{min}} = 0$ or if $p(t_{\text{min}}) - p(0) > -10^{-3}$, we take $G_{\text{new}} = G + t_{\text{min}} \Delta G + 10^{-5} \text{rand}(n,k)$, otherwise we take $G_{\text{new}} = G + t_{\text{min}} \Delta G$, where $\text{rand}(n,k)$ is a random matrix with entries uniformly distributed on $(0,1)$. Finally, we project $G_{\text{new}}$ to the cone of non-negative matrices $G_{\text{new}} = \max(G_{\text{new}}, 0)$.

To compute new iterate $(S_1, \ldots, S_N)_{\text{new}}$, we again perform exact line search in the directions defined by gradients $\Delta S_i = \nabla S_i \text{SE}$, but in this case we have for
each $i = 1, \ldots, N$

$$p_i(t) = \| R_i - G(S_i + t \Delta S_i)G^T \|^2,$$

which is a convex quadratic polynomial of degree 2 in $t$ with minimum

$$t_{\min} = \frac{\langle R_i - G S_i G^T, G \Delta S_i G^T \rangle}{\| G \Delta S_i G^T \|^2}.$$

Therefore, new iterate is $(S_i)_{new} = \max(S_i + t_{\min} \Delta S_i, 0)$.

### 4 Approaches based on search space transformation

The third and the fourth method that we developed for solving the NMTF are based on so-called space transformation, where we replace the non-negativity constraints on $G$ and $S_i$ by transforming objective function $SE$ and we then apply the gradient descent method with exact line search (GM-ELS) and ADAM (adaptive moment estimation) from [27]. We can transform the feasible set for the NMTF $G$ (search space) from $\mathbb{R}_+^{n \times k}$ to $\mathbb{R}^{n \times k}$ by a simple variable substitution

$$G = f(\tilde{G})$$

$$S_i = f(\tilde{S}_i),$$

where $G$ is a non-negative matrix, $\tilde{G}$ is a real matrix and $f$ is an appropriate transformation function. Similarly, we can transform the feasible sets for $S_i$ from $(S_k)_+$ to $(\tilde{S}_k)$ using the same transformation.

The transformed optimization problem, with $\tilde{G}, \tilde{S}_i$ as new variables, is unconstrained optimization problem, however, this gain is traded for a more complicated objective function

$$SE = \sum_i \| R_i - f(\tilde{G}) f(\tilde{S}_i) f(\tilde{G})^T \|^2. \quad (20)$$

Advantage of this approach is that common local search algorithms developed for unconstrained optimization problems can be applied without modifications.

This technique has a long history in non-linear optimization and is usually employed in order to transform bounded search space variables to unbounded. Examples of transformations used in literature include smooth transformation of open hyper-rectangle to $\mathbb{R}^m$ [4], continuous transformation of closed hyper-rectangle to $\mathbb{R}^m$ [54] and even continuous transformation of closed hyper-rectangle to search space with toroidal topology [50].

Two transformation functions that we consider in this work are:

- Element-wise absolute value (search space boundary extension by mirroring [54] adapted for non-negativity)

$$f_1(X) = |X| \quad (21)$$

- Element-wise square

$$f_2(X) = X \odot X \quad (22)$$

We can apply $f_1$ and $f_2$ to any matrix, regardless its dimension. The benefit of using absolute value is that this transformation does not change characteristics of the optimization landscape in any way. However, by using absolute value, the objective function ceases to be a polynomial. Element-wise square, on the other hand, changes the optimization landscape, however, objective function remains a polynomial, but with doubled degree.
4.1 Gradient calculation

With this optimization approach we have gradients or subgradients in every feasible point, which can be computed using the chain rule. By introducing

\[ Z_i = R_i - f(\tilde{G})f(\tilde{S}_i)f(\tilde{G})^\top, \]

and by using that \( R_i \) and \( S_i \) are symmetric and that \( f \) does not break the symmetry, the gradient of SE can be expressed as

\[ \Delta G = \nabla_{\tilde{G}} SE = -4 \sum_i f'(\tilde{G}) \odot \left( Z_i f(\tilde{G})f(\tilde{S}_i)^\top \right) \]

\[ \Delta S_i = \nabla_{\tilde{S}_i} SE = -2 \sum_i f'(\tilde{S}_i) \odot \left( f(\tilde{G})^\top Z_i f(\tilde{G}) \right), \]

where \( f' \) is the derivative of function \( f \), applied component-wise, i.e. \( f'(\tilde{G}) = (f'(G_{ij}))_{i,j} \). Formulae (24) and (25) are derived in appendix A.

Function \( f \) may also be only piecewise differentiable. This is the case for \( f_1(X) = |X| \) from above, which is non-differentiable at any component of \( X \) equal to 0. In such a situation, we take in the non-differentiable points the sub-derivatives. In the case of \( f_1 \), the set of all sub-derivatives is \([-1, 1]\). Therefore, we are free to make a choice \( f'_1(0) = 0 \) which is also in accordance with the statement of Fermat’s theorem on stationary points which says that every local extreme is a stationary point (the function derivative is zero at that point).

This is a very common approach in gradient based optimization, especially in deep learning. However, the proof of mathematical validity of using subgradients in stochastic optimization is, despite its wide use, remarkably recent [10].

4.2 Gradient method with exact line search (GM-ELS)

In this subsection we present an algorithm employing the line search strategy to the transformed problem. This is an iterative procedure of making steps in direction of gradient of SE using step size \( t \) which is calculated independently for each specific step made so that the step taken minimizes SE.

\[ \tilde{G} \leftarrow \tilde{G} - t \Delta \tilde{G} \]

\[ \tilde{S}_i \leftarrow \tilde{S}_i - t \Delta \tilde{S}_i \]

\[ t = \arg\min \ SE(\tilde{G} - t \Delta \tilde{G}, \ \tilde{S}_i - t \Delta \tilde{S}_i) \]

Optimisation problem (28) needs to be solved in every iteration of the algorithm. Choice of function \( f \) affects the form of this optimization problem. In general (eg. \( f = f_1 \)), step size calculation (28) is a nonlinear optimization problem which can only be solved approximately using inexact methods. However, by choosing \( f = f_2 \) minimisation (28) becomes a uni-variate minimisation of a polynomial of order 12 which can be solved exactly (up to numerical errors). We call the algorithm employing this technique gradient method with exact line search (GM-ELS).
Four algorithms to solve symmetric multi-type NMTF problem

Fig. 3 Exact line search algorithm

| Input: | Non-negative symmetric $R_1, \ldots, R_N$. |
|--------|------------------------------------------|
| Initialisation: | Compute initial values for $\tilde{G}$ and $\tilde{S}_i$. |
| 1. | While termination test not satisfied |
| 2.1 | Compute gradients $\Delta G = \nabla_{\tilde{G}} SE$ and $\Delta S_i = \nabla_{\tilde{S}_i} SE$ using (24) and (25) with $f = f_2$. |
| 2.2 | Compute coefficients of polynomial $p(t)$ using (36). |
| 2.3 | $t \leftarrow \text{argmin } p(t)$ |
| 2.4 | For $X \in \{\tilde{G}, \tilde{S}_1, \ldots, \tilde{S}_N\}$ |
| 2.4.1 | $X \leftarrow X - t \nabla_X SE$. |
| 3. | Return $\tilde{G} \odot \tilde{G}, \tilde{S}_1 \odot \tilde{S}_1, \ldots, \tilde{S}_N \odot \tilde{S}_N$ |

By choosing $f = f_2$ the objective function becomes

$$SE = \sum_i \| R_i - \tilde{G}^2 \tilde{S}_i T \tilde{G}^2 \|^2,$$

where squares act element-wise. To find out how $SE$ changes when moving in direction of gradient we substitute $\tilde{G}$ and $\tilde{S}_i$ in (29) with right hand sides of (26) and (27). We observe that $SE$ changes with $t$ as a polynomial

$$p(t) = \sum_{i=0}^{12} c_i t^i. \tag{30}$$

By finding $t$ that minimizes $p(t)$ we find what step size to make in direction of gradient so that $SE$ is minimal. Even though minimisation of polynomial (30) is an easy problem, calculation of its coefficients is computationally quite expensive. Initial tests show that the this calculation is approximately 7 times more costly compared to the calculation of the gradient. We also tried to reconstruct coefficients of $p(t)$ by polynomial curve fitting methods using it evaluations on $\geq 13$ points, but the step lengths obtained this way yielded worse performance. See our comment on page 10 and Section B for details on calculation of these coefficients. Full GM-ELS algorithm is stated in Fig. 3.

Due to exact calculation of optimum $t$, $SE$ monotonically decreases during iterations, hence it converges to some non-negative value. This means that at some iteration we have

$$\min p(t) \approx p(0). \tag{31}$$

The equality can (approximately) hold only if the polynomial $p(t)$ has a minimum at point $t = 0$ which indicates that the gradient of $SE$ is zero and proves convergence of GM-ELS toward a point with zero gradient.

4.3 Adaptive moment estimation

The adaptive moment estimation (ADAM) \cite{27} is a gradient based optimization algorithm and is currently state-of-the-art algorithm for training deep neural networks. It was designed for problems with high dimensional search space, noisy objective function and sparse gradients. The ADAM cannot be straightforwardly applied for solving the SNMTF due to non-negativity constraint. However, this
limitation can be circumvented by using transformed formulation of the SNMTF presented in (20) which releases the non-negativity constraint.

In order to apply ADAM to the SNMTF we chose to transform search space by using element-wise absolute value, i.e. $f = f_1$. We decided to use absolute value because this operation is the most computationally efficient way to guarantee non-negativity, since it can be implemented with a conditional clause and no other operations. Also, this transformation leaves optimization landscape unchanged. The use of absolute value imply the use of sub-derivatives as explained in section 4.1.

In ADAM algorithm steps are not made in direction of gradient but in direction of moving average of the gradient. Another trait is step size scaling which is adapted for each matrix element independently. Update rule for both $G$ and $S_i$ are the same, therefore we only state the one for $G$, which is updated using the following formulae:

$$M_{\tilde{G}} \leftarrow \beta_1 M_{\tilde{G}} + (1 - \beta_1) \Delta \tilde{G}$$  \hspace{1cm} (32)

$$V_{\tilde{G}} \leftarrow \beta_2 V_{\tilde{G}} + (1 - \beta_2) \Delta \tilde{G} \odot \Delta \tilde{G}$$  \hspace{1cm} (33)

$$\tilde{G} \leftarrow \tilde{G} - \eta \frac{M_{\tilde{G}} \odot (\sqrt{V_{\tilde{G}}} + \varepsilon)}{p V_{\tilde{G}} + \varepsilon},$$  \hspace{1cm} (34)

where $\odot$ denotes element-wise division of two matrices and $\sqrt{}$ and adding $\varepsilon$ are applied element-wise. The Greek symbols are scalars and $M_{\tilde{G}}$ and $V_{\tilde{G}}$ are the matrices of the same size as $\tilde{G}$ and contain the information about the mean and the variance of gradient $\Delta G$, respectively. Parameter $\eta$ changes with a predetermined schedule as stated in Fig. 4.

As seen in (32), ADAM uses momentum $M$ which is an exponential moving average of the gradient with $\beta_1$ being the decay constant. Using momentum instead of gradient for descent direction makes optimization more stable by preventing oscillations and by averaging possibly noisy or sparse gradients [47]. The ADAM also uses second moment $V$ (33) which is an exponential moving average of element-wise square of the gradients. This quantity is an approximation of diagonal of Fisher information matrix and is used to adaptively scale the step size for each search variable independently as seen in (34).

The complete algorithm for solving the SNMTF is shown in Fig. 4 and is controlled by four parameters ($\alpha$, $\beta_1$, $\beta_2$ and $\varepsilon$) which influence its efficiency. The most important one is the so called learning rate $\alpha$. The usual magnitude of the learning rate is $\alpha \sim 10^{-3}$; however, due to descent direction scaling by using second moment $V$, algorithm efficiency is intended to be robust with respect to this choice. Another important parameter is $\beta_1$ which should be chosen in accordance to the noise level and gradient sparsity level. However, there are no theoretically founded ways to assess appropriate parameter values based on the optimization problem at hand. Therefore some degree of parameter tuning is normally used.

Empirically, ADAM has proven to be very efficient and with a superior convergence rate compared to other gradient based optimization algorithms [24]. However, contrary to these findings there are no guaranties that the algorithm converges. In [38] explicit example of a simple convex optimization problem was provided where ADAM does not converge to the optimal solution. In practice, however, such convergence problems are seldomly encountered and can be mended.
Fig. 4 ADAM algorithm

| Input: Non-negative symmetric $R_1, \ldots, R_N$ and parameters $\alpha, \beta_1, \beta_2, \varepsilon$. |
|---|
| 1. **Initialisation:** Compute initial values for $\tilde{G}$ and $\tilde{S}_i$. |
| 2. **For** $X \in \{\tilde{G}, \tilde{S}_1, \ldots, \tilde{S}_N\}$ |
| 2.1 $M_X \leftarrow X \cdot 0$ |
| 2.2 $V_X \leftarrow X \cdot 0$ |
| 3. $i \leftarrow 0$ |
| 4. **While** termination test not satisfied |
| 4.1 $i \leftarrow i + 1$ |
| 4.2 $\eta \leftarrow \alpha \frac{\sqrt{1-(1-\beta_2)^i}}{1-(1-\beta_1)^i}$ |
| 4.3 Compute derivatives $\Delta \tilde{G}, \nabla \tilde{S}_1 \SE, \ldots, \nabla \tilde{S}_N \SE$ using (24) and (25) with $f = f_1$. |
| 4.4 **For** $X \in \{\tilde{G}, \tilde{S}_1, \ldots, \tilde{S}_N\}$ |
| 4.4.1 $M_X \leftarrow \beta_1 M_X + (1 - \beta_1) \nabla X \SE$ |
| 4.4.2 $V_X \leftarrow \beta_2 V_X + (1 - \beta_2) \nabla X \SE \odot \nabla X \SE$ |
| 4.4.3 $X \leftarrow X - \eta M_X \odot (\sqrt{V_X} + \varepsilon)$ |
| 5. **Return** $|\tilde{G}|, |\tilde{S}_1|, \ldots, |\tilde{S}_N|$ |

by changing initial point and/or algorithm parameters which influence algorithm behaviour.

5 Implementation details

In this section we explain the details about implementation of all four algorithms described in the previous sections. The FPM and BCD were implemented in Matlab [36]. The reason to choose this framework was that the development of this code was part of another project [34] which imposed Matlab as working environment. The ADAM and GM-ELS were implemented later with ambition to be high-performance open-source code. This is why we used Python and TensorFlow library [1]. The Matlab and Python codes and all the data-sets used in this paper are available on GitLab [23].

5.1 Starting points

During our study, we have observed a well-known fact that good starting points are often very important for overall convergence. By default, we start the methods using random starting matrices. More precisely, for FPM we generate $G$ in Matlab by a simple call `rand(n,k)`, while each matrix $S_i$ is generated by `rand(k)` and symmetrisation afterwards. Likewise, we generate starting $G$ for BCD, while the first tuple of $S_i$ is computed from the starting $G$. For GM-ELS and ADAM, we generate starting points for both $G$ and $S_i$ by using NumPy’s function `random.rand`.

Following the literature [7], we also tested deterministic starting points $G$, extracted from eigenvalue decomposition of (symmetric) matrix $R = \sum_i R_i$. We computed for the given inner dimension $k$ the $k$ largest magnitude eigenvalues $|\lambda_1| \geq |\lambda_2| \geq \cdots \geq |\lambda_k|$ and the corresponding eigenvectors $\mathbf{x}_i$. For each eigenvector $\mathbf{x}_i$ we considered its positive and negative parts, $\mathbf{x}_i^+$ and $\mathbf{x}_i^-$, respectively, and took
the one with larger euclidean norm (we denote it by $\tilde{x}_i$). By concatenating the $k$ vectors $\tilde{x}_i$ column-wise we finally obtain the $n \times k$ starting point $G$. Spectral decomposition was performed using function eigs available in Matlab or Scipy’s module sparse.linalg. Note that we could also compute eigenvalues using function svds but our tests showed that eigs was always faster, which is not surprising.

By extensive tests where we compared for each instance the best MSE and the arithmetic mean value of the MSEs over 30 runs with different random starting matrices with the MSE obtained by deterministic starting point we noticed that arithmetic mean values were almost always worse than the deterministic MSE. By using $t$-tests we observed that the differences were usually statistically significant, but in practice these differences were small (below 0.01 – the significance came from the very small variances of the MSEs). On the other hand, the best (minimum) value of MSE out of 30 values obtained with different random starts was always very close to the MSE obtained by deterministic start, several times slightly better. This suggests that if the dimension of the problem is small then making several rounds with different random starting points and taking the best value is reasonable, but for higher dimensions starting with the deterministic $G$ is the best strategy.

All our computations were done with deterministic starting points only, since all three data sets included also large scale matrices.

5.2 The main parts of the algorithms

After the initialisation of FPM and BCD, we iterate through the main parts of the algorithms. For FPM this means that we iteratively compute $(S_1, \ldots, S_N)_{\text{new}}$ using (17) and $G_{\text{new}}$ using (16) after adding Matlab’s defined constant $\varepsilon = 2.2204 \cdot 10^{-16}$ to all of the denominators.

For BCD, the updates $(S_1, \ldots, S_N)_{\text{new}}$ and $G_{\text{new}}$ (Steps 2.1 and 2.3 from Figure 2) are computed using 10 iterations of projected gradient method with exact line search. In both cases, in every iteration, we compute gradient at the current point and the new iterate is obtained by exact line search as described in Section 3.3. Finally, we project each new iterate to the cone of non-negative matrices. We decided to do only 10 inner iterations based on our previous experiences. The underlying idea is that we do not want to invest too much time in computing very good solutions of the block subproblems, when we know that these solutions are needed only as good starting points for the next steps. This is especially relevant at the beginning of BCD algorithm. Later, when we are closer to an optimum it would be reasonable to increase the number of inner iterations but we did not observe any practical improvement, so we decided to keep the number of inner iterations constant throughout the method. For example, in the Boundary point method from [33, 44] we make in practice only one step of block coordinate optimization and the overall method achieves great efficiency.

The GM-ELS and ADAM algorithms were implemented using TensorFlow library. The major advantages of this framework are support for automatic differentiation and highly efficient compiler which results in optimized computational graph for gradient calculation with built-in parallelisation.

A single iteration of GM-ELS starts with gradient calculation. This step is implemented using TensorFlow functions for reverse automatic differentiation. In
other words, computational graph for SE calculation is recorded and differentiated using chain rule and dynamic programming methods at compile time, yielding an optimized computational graph for calculation of gradient of SE with respect to both $\tilde{G}$ and $\tilde{S}_i$. The resulting gradient is equal to the one stated in (24) and (25). We chose this method because it is computationally efficient and requires less programming effort and is therefore less error prone. Knowing the gradient, a polynomial is calculated which tells how SE changes with step size in direction of gradient. This step is computationally intensive and is covered in detail in appendix [3]. In our case these computations are left to TensorFlow library which manages to do them efficiently due to its compiler optimizations such as elimination of common subexpressions and simplification of arithmetic statements. However, the time needed for this step was still non-neglectable (approximately 7 times longer compared to gradient calculation). In the next step polynomial is minimized numerically to find the step size that minimizes SE. Minimization was performed by finding all roots of derivative of this polynomial. We used NumPy’s function `roots` that relies on computing the eigenvalues of the companion matrix. Step that minimizes SE is finally employed to update the matrices $G$ and $S_i$.

The ADAM algorithm makes use of the same method of gradient calculation as GM-ELS. Because of search space transformation there is no non-negativity constraint, therefore, standard implementation of ADAM was used as implemented in TensorFlows module `keras.optimizers.Adam`. Unlike other three algorithms, ADAM also requires optimization parameters to be set by the user. We used parameter tuning to find appropriate parameter values as explained in Section 5.4.

5.3 Termination criterion

The stopping criteria for all methods were maximum number of iterations or if the MSE falls below $10^{-2}$ or if absolute difference between the MSE in given iteration and the MSE in the previous iteration is below $10^{-10}$. We did an extensive testing on small, medium and large scale matrices, see Section 6.2 and finally experimentally set the maximum number of iterations to 4000 for FPM, 300 for BCD, 1000 for GM-ELS and 3000 for ADAM. With these values the computation times (wall time) of all four methods were of the same order. A special section (Section 6.3) is devoted to the analysis how MSE decreases with time.

5.4 Parameter tuning

Behaviour of ADAM algorithm is controlled by four parameters $(\alpha, \beta_1, \beta_2, \varepsilon)$. To find suitable values we performed parameter tuning on $\alpha, \beta_1, \beta_2$ while leaving $\varepsilon$ fixed since it is the least important parameter which only prevents division by zero [27]. Tuning was performed using random search which is more efficient compared to commonly used grid search [3]. We randomly generated 100 points in parameter space on a hyper-rectangle defined by $\alpha \in [10^{-4}, 10^{-1}]$, $\beta_1 \in [0.2, 0.999]$ and $\beta_2 \in [0.1, 0.999]$. Each of the 100 parameter combinations was evaluated by performing three runs of ADAM algorithm on each problem included in a synthetic data-set (the first data-set described in Section 6.1) with $k = K$, starting from a
Fig. 5 Impact of parameters $\alpha$, $\beta_1$, and $\beta_2$ on efficiency of ADAM algorithm for synthetic problems with $K = k$. Heatmaps show maximum (over problems) mean (over runs) MSE with respect to $\alpha$ and $\beta_1$ for different values of $\beta_2$. Green contours envelop areas where MSE $\leq 0.01$ for all problems in synthetic data-set. Visualisation was made using support vector regression model trained on a set of 100 randomly chosen triples of parameters $(\alpha, \beta_1, \beta_2)$.

random initial point. By construction we know that for each 5-tuple of matrices $(R_1, \ldots, R_5)$ from this data-set the optimum MSE is 0.

We observe that matrix size $n$ and latent dimension $k$ has little influence on the position of optimal parameter values. In fact the MSE is quite low on a large proportion of inspected parameter space. Fig. 5 shows a region of parameter space where the MSE $\leq 0.01$ for all problems from the synthetic data-set and represents a guideline for choosing appropriate parameters of ADAM algorithm for this class of optimization problems. Even though there is no guaranty that this region of parameter space will be promising for other, real-world, factorization problems, we conclude from these results that small changes in the parameters do not considerably change the algorithm efficiency. This means that good results can be acquired even with minimal tuning.

Based on these findings we fixed ADAM parameters to $\alpha = 0.002$, $\beta_1 = 0.95$, $\beta_2 = 0.995$ and $\varepsilon = 10^{-8}$. All reported results for ADAM in the following sections were acquired using these parameter values.
6 Results

6.1 Data

We benchmark the four methods presented in the previous sections on three data-sets: on a set of synthetic random matrices, real-world similarity matrices gathered by [49] and biological network data gathered during our previous studies. All three data-sets are included in the afore mentioned GitLab repository [23].

The set of synthetic random matrices was constructed by us. It consists of 5-tuples of non-negative symmetric matrices of order \( n = 100, 200, 500, 1000, 2000 \) and 5000. For each \( n \) we first select \( K \in \{10, 20, 30, 40, 50\} \) and construct random non-negative \( G \) of order \( n \times K \) with orthogonal columns and 5 non-negative matrices \( S_i \) of order \( K \times K \) with densities approximately 65% and computed matrices \( R_i \) as products \( R_i = GS_iG^\top \). For each possible \( n, K \) we store matrices \( R_i, S_i \) and \( G \) into a file and make them available, e.g., directory test_data_n=1000_k=50_N=5 contains matrices \( R_1, R_2, \ldots, R_5, G, S_1, \ldots, S_5 \) corresponding to \( n = 1000, K = 50 \).

The second data-set was gathered by [49] and represents real-world similarity or proximity matrices from different fields of data science. Out of 13 available data-sets, only 5 of them have only non-negative values:

- **Aural** data-set is taken from [13], investigating the human ability to distinguish different types of sonar signals by ear. The signals were returns from a broadband active sonar system, with 50 target-of-interest signals and 50 clutter signals. Every pair of signals was assigned a similarity score from 1 to 5 by two randomly chosen human subjects unaware of the true labels, and these scores were added to produce a \( 100 \times 100 \) similarity matrix with integer values from 2 to 10.

- **Protein** data-set has sequence-alignment similarities for 213 proteins from 4 classes, where classes one through four contain 72, 72, 39, and 30 points, respectively [22].

- **Voting** data-set comes from the UCI Repository. It is a two-class classification problem with 435 points, where each sample is a categorical feature vector with 16 components and three possibilities for each component. Value difference metric was computed [51] from the categorical data, which is a dissimilarity that uses the training class labels to weight different components differently so as to achieve maximum probability of class separation.

- **FaceRec** data-set consists of 945 sample faces of 139 people from the NIST Face Recognition Grand Challenge data-set. There are 139 classes, one for each person. Similarities for pairs of the original three-dimensional face data were computed as the cosine similarity between integral invariant signatures based on surface curves of the face [14].

- **Zongker** data-set is a digit dissimilarity data (2000 points in 10 classes) and is based on deformable template matching. The dissimilarity measure was computed among 2000 handwritten NIST digits in 10 classes, with 200 entries each, as a result of an iterative optimization of the non-linear deformation of the grid [25].

The third data-set is biological molecular network data. It consists of 4 molecular interaction networks (Protein-Protein Interaction network - PPI, Gene Co-expression COEX, Genetic Interaction - GI and sequence similarity networks -
Table 1  Species considered in the biological network data-set.

| common name       | taxonomic name               | tag   | n    |
|-------------------|------------------------------|-------|------|
| fission yeast     | Schizosaccharomyces pombe    | Yeast | 4979 |
| baker’s yeast     | Saccharomyces cerevisiae     | Yeast | 5890 |
| fruit fly         | Drosophila melanogaster      | Fly   | 13190|
| nematode worm     | Caenorhabditis elegans       | Worm  | 17519|
| human             | Homo sapiens                 | Human | 20967|

SeqSim) for 5 different species listed in Table 1. These networks represent real-world use cases where several matrices need to be factored simultaneously in order to integrate multiple data sources and produce more consistent, accurate and useful information. The data was collected in June 2016 during the preliminary phase of research published in [35] using BioGRID [40], COXPRESdb [39], STRING [52] and BLAST [2], respectively, and is available on our GitLab profile [23].

6.2 Results for MSE on the test data-sets

We run all four methods on all three data-sets using high-performance cluster at University of Ljubljana, Faculty of mechanical engineering, which is a E5-2680 V3 (1008 hyper-cores) DP cluster, with IB QDR interconnection, 164 TB of LUSTRE storage, 4.6 TB RAM, supplemented by GPU accelerators.

Results obtained by all four methods on the data-set of synthetic random orthogonal matrices are reported in a condensed format in Table 2 and in the first 6 plots of Fig. 6. For each $n \in \{100, 200, 500, 1000, 2000, 5000\}$ and $K \in \{10, 20, 30, 40, 50\}$ we run all four methods with deterministic starting solution $G$ and with inner dimension equal to $k = 0.2K, 0.4K, 0.6K, 0.8K, 1.0K, 1.2K$. Procedure of choosing various $k$ values simulates real-world case where inner dimension $K$ is usually not known. However, in this case we know that for $k = 1.0K, 1.2K$, by construction, the optimum MSE$_{opt} = 0$ and our results demonstrate the capability of the methods we used to reach a global optimum.

Each row of Table 2 shows the mean values of the MSE, for each $n$ and each possible value of $k/K$. For example, the last row of this table shows the mean values of MSE for $n = 5000$ and for $k/K = 120 \%$. This means that we computed and reported for each algorithm the mean values of five MSEs: for $k = 12, K = 10$; $k = 24, K = 20$; $k = 36, K = 30$; $k = 48, K = 40$ and $k = 60, K = 50$.

Numerical results from Table 2 reveal that FPM, GM-ELS and ADAM are very competitive regarding the MSE, while BCD yields worse MSE for larger values of $n$ and $k/K$. A deeper analysis shows that worse performance of BCD is due to small number of outer iterations (it was limited to 300). When we allow 1000 iterations, the resulting MSE was of the same order of magnitude – i.e., below $10^{-2}$ as for the other three methods.

Therefore, based on the MSE and this first data-set we cannot decide which method is better. However, when we take into account also the time, situation changes – see Section 6.3.

Table 3 contains results on the second data-set (real-world similarity matrices). The four methods were run on each data-matrix, using deterministic starting $G$. The inner dimensions were chosen such that they increase evenly towards the number of the clusters in the underlying data points. We can see that the mean
values of MSE for Voting, FaceRec, and Zonker are falling rapidly towards 0 which demonstrate that all four methods are capable to find the optimum decompositions when \( k \) approached the estimated real value of \( K \). However, for Aural and Protein the MSE does not come very close to 0. This can mean that those two matrices cannot be factored in the proposed way due to the noise or the inner dimension was not correctly identified or some other hidden reason might be present.

For this data-set we observe no clear pattern that would declare which algorithms are systematically better than the others. For large \( k \) all algorithms obtain similar MSE values (with exception of Protein where BCD is clearly not as good as the others). In the case of smaller \( k \) the ranking of the algorithms changes with respect to problem instance and inner dimension \( k \).

Table 4 contains results of all four methods evaluated on the third data-set, which consists of 5 4-tuples of symmetric non-negative matrices with sizes varying from 4979 (Yeast_SPO) to 20967 (Human_HSA). All methods started in the deterministic starting point, as described in Section 5.1.

Solving (SNMTF) for this data was very time and memory consuming. We can see that GM-ELS runs out of memory (OOM) for the largest three tuples of matrices, while the other three methods always terminate regularly, according to stopping criteria described in Section 5.3. We decided to evaluate methods with 5 inner dimensions: \( \lceil \sqrt{n}/5 \rceil, 2 \lceil \sqrt{n}/5 \rceil, \ldots, 5 \lceil \sqrt{n}/5 \rceil \), where \( \lceil \cdot \rceil \) denotes rounding to the nearest integer.

Regarding the MSE we observe that FPM, BCD, and ADAM perform similarly. For the largest instance (Human_HSA) ADAM returns slightly worse MSE, but the difference is quite small. Similar situation can be noticed in case of Yeast_SPO and Yeast_SCE, where MSEs for FPM algorithm are systematically slightly higher. GM-ELS, on the other hand, is consistently worse than others or, for larger data-sets, even runs out of memory. This indicates that GM-ELS is not an appropriate choice when factoring large data-sets. This is a result of considerable resources needed for calculation of 12 order polynomial used for determining ideal step size for GM-ELS.

We summarise the numerical results from Tables 2–4 in Figures 6–7. The former figure shows how MSE changes (decreases) with increasing inner dimension \( k \). For example, the first 6 plots from the top of the figure depict the results from Table 2, where on x axis we put \( 100k/K \), which has values 20, 40, 60, 100, 120.

We can clearly see that the MSE decreases with increasing the inner dimension \( k \), the only exception is Zonker data-set, where MSE are very small at the very beginning (below \( 10^{-2} \)) and due to the stopping criteria MSE does not decrease any further, actually it even slightly increases for the case of FPM, but remains below \( 10^{-2} \).

To provide better insight which algorithms performs best on given instance (regarding the MSE) we decided to depict the winners in Figure 7. More precisely, for each synthetic random instance with \( K = 50 \), each real-world similarity instance and each biological network instance we plot at position \(( \log(n), \log(k) )\) the symbol denoting which out of the four methods gives lowest MSE for given instance within this numerical evaluation. The more signs appear in this plot the more often corresponding method returns the lowest MSE. We omit the synthetic instances with \( K < 50 \) for the sake of more transparent figure. We can observe that ADAM the most often (in approximately 50% of the cases) gives the best MSE, followed by BCD and FPM, each winning in approximately 24% of the instances.
| n   | k/K (%) | FPM   | BCD   | GM-ELS | ADAM   |
|-----|---------|-------|-------|--------|--------|
| 100 | 20      | 0.5112| 0.5119| 0.5196 | 0.5138 |
| 100 | 40      | 0.3924| 0.3883| 0.4008 | 0.3858 |
| 100 | 60      | 0.2740| 0.2726| 0.2841 | 0.2685 |
| 100 | 80      | 0.1402| 0.1519| 0.1506 | 0.1409 |
| 100 | 100     | 0.0085| 0.0283| 0.0430 | 0.0000 |
| 100 | 120     | 0.0079| 0.0127| 0.0001 | 0.0000 |
| 200 | 20      | 0.5136| 0.5135| 0.5197 | 0.5236 |
| 200 | 40      | 0.3916| 0.3852| 0.4151 | 0.3856 |
| 200 | 60      | 0.2759| 0.2763| 0.2873 | 0.2676 |
| 200 | 80      | 0.1389| 0.1619| 0.1473 | 0.1375 |
| 200 | 100     | 0.0092| 0.0246| 0.0286 | 0.0000 |
| 200 | 120     | 0.0085| 0.0335| 0.0070 | 0.0000 |
| 500 | 20      | 0.5109| 0.5104| 0.5189 | 0.5151 |
| 500 | 40      | 0.3986| 0.4015| 0.4042 | 0.3919 |
| 500 | 60      | 0.2789| 0.2768| 0.2873 | 0.2733 |
| 500 | 80      | 0.1420| 0.1561| 0.1479 | 0.1360 |
| 500 | 100     | 0.0084| 0.0199| 0.0189 | 0.0000 |
| 500 | 120     | 0.0090| 0.0096| 0.0001 | 0.0000 |
| 1000| 20      | 0.5325| 0.5159| 0.5154 | 0.5183 |
| 1000| 40      | 0.4197| 0.3895| 0.4059 | 0.3928 |
| 1000| 60      | 0.2881| 0.2777| 0.2810 | 0.2694 |
| 1000| 80      | 0.1528| 0.1444| 0.1489 | 0.1393 |
| 1000| 100     | 0.0107| 0.0183| 0.0210 | 0.0000 |
| 1000| 120     | 0.0084| 0.0180| 0.0002 | 0.0000 |
| 2000| 20      | 0.5192| 0.5036| 0.5072 | 0.5071 |
| 2000| 40      | 0.3992| 0.3830| 0.3864 | 0.3770 |
| 2000| 60      | 0.2714| 0.2614| 0.2655 | 0.2578 |
| 2000| 80      | 0.1401| 0.1447| 0.1391 | 0.1308 |
| 2000| 100     | 0.0103| 0.0865| 0.0231 | 0.0001 |
| 2000| 120     | 0.0090| 0.0885| 0.0001 | 0.0000 |
| 5000| 20      | 0.5042| 0.5021| 0.5042 | 0.5114 |
| 5000| 40      | 0.3847| 0.3823| 0.3892 | 0.3790 |
| 5000| 60      | 0.2644| 0.2704| 0.2685 | 0.2627 |
| 5000| 80      | 0.1383| 0.1446| 0.1497 | 0.1329 |
| 5000| 100     | 0.0112| 0.0415| 0.0387 | 0.0001 |
| 5000| 120     | 0.0130| 0.0739| 0.0004 | 0.0001 |

Table 2 In this table we report MSE values obtained by all four algorithms on the synthetic data-set. We report mean values of MSE taken over all pairs $(k, K)$ with ratio $100k/K = 20, 40, \ldots, 120$. The entries with the best values of MSE for each row are bold-faced.

The GM-ELS results with best MSE only in two cases. Additionally, ADAM outperforms the other algorithms for larger values of the inner dimension $k$, while for the smaller values of $k$ the winners are usually FPM and BCD.
Four algorithms to solve symmetric multi-type NMTF problem

Table 3 MSE values obtained by all four algorithms on the real-world data-sets gathered by [49]. The entries with the best values of MSE for each row are bold-faced.

| instance | n  | k  | FPM | BCD | GM-ELS | ADAM |
|----------|----|----|-----|-----|--------|------|
| Aural    | 100| 2  | 0.1649 | 0.1649 | 0.1637 | 0.1649 |
| Aural    | 100| 4  | 0.1391 | 0.1222 | 0.1242 | 0.1303 |
| Aural    | 100| 6  | 0.1213 | 0.1088 | 0.1217 | 0.1088 |
| Aural    | 100| 8  | 0.1029 | 0.1048 | 0.1099 | 0.0990 |
| Aural    | 100| 10 | 0.0986 | 0.0981 | 0.1072 | 0.0911 |
| Protein  | 213| 3  | 0.1199 | 0.1187 | 0.1196 | 0.1280 |
| Protein  | 213| 6  | 0.0778 | 0.0795 | 0.0987 | 0.0868 |
| Protein  | 213| 9  | 0.0596 | 0.0716 | 0.0710 | 0.0625 |
| Protein  | 213| 12 | 0.0509 | 0.0698 | 0.0574 | 0.0523 |
| Protein  | 213| 15 | 0.0446 | 0.0694 | 0.0528 | 0.0468 |
| Voting   | 435| 1  | 0.2129 | 0.2129 | 0.2129 | 0.2129 |
| Voting   | 435| 2  | 0.0766 | 0.0869 | 0.0881 | 0.0669 |
| Voting   | 435| 3  | 0.0083 | 0.0059 | 0.0073 | 0.0049 |
| Voting   | 435| 4  | 0.0094 | 0.0052 | 0.0062 | 0.0039 |
| Voting   | 435| 5  | 0.0093 | 0.0053 | 0.0063 | 0.0031 |
| FaceRec  | 945| 1  | 0.0051 | 0.0051 | 0.0055 | 0.0051 |
| FaceRec  | 945| 2  | 0.0380 | 0.0054 | 0.0051 | 0.0353 |
| FaceRec  | 945| 3  | 0.0371 | 0.0050 | 0.0064 | 0.0029 |
| FaceRec  | 945| 4  | 0.0095 | 0.0030 | 0.0018 | 0.0010 |
| FaceRec  | 945| 5  | 0.0085 | 0.0029 | 0.0036 | 0.0006 |
| Zongker  | 2000| 1 | 0.0025 | 0.0025 | 0.0679 | 0.0025 |
| Zongker  | 2000| 2 | 0.0068 | 0.0025 | 0.0025 | 0.0025 |
| Zongker  | 2000| 3 | 0.0077 | 0.0025 | 0.0024 | 0.0020 |
| Zongker  | 2000| 4 | 0.0072 | 0.0025 | 0.0023 | 0.0020 |
| Zongker  | 2000| 5 | 0.0094 | 0.0025 | 0.0024 | 0.0020 |

Table 4 MSE values obtained by all four algorithms on biological network data-sets. OOM indicates that the algorithm ran out of memory and did not complete even a single iteration. The entries with the best values of MSE for each row are bold-faced.

| instance | n  | k  | FPM | BCD | GM-ELS | ADAM |
|----------|----|----|-----|-----|--------|------|
| YeastSPO| 4979| 14 | 0.8506 | 0.8525 | 0.8687 | 0.8535 |
| YeastSPO| 4979| 28 | 0.8312 | 0.8263 | 0.8432 | 0.8221 |
| YeastSPO| 4979| 42 | 0.8131 | 0.8012 | 0.8436 | 0.7987 |
| YeastSPO| 4979| 56 | 0.8031 | 0.7845 | 0.8380 | 0.7845 |
| YeastSPO| 4979| 70 | 0.7871 | 0.7688 | 0.8433 | 0.7659 |
| YeastSCE| 5890| 15 | 0.8389 | 0.8367 | 0.8544 | 0.8509 |
| YeastSCE| 5890| 30 | 0.8156 | 0.8076 | 0.8297 | 0.8120 |
| YeastSCE| 5890| 45 | 0.7996 | 0.7899 | 0.8207 | 0.7832 |
| YeastSCE| 5890| 60 | 0.7865 | 0.7692 | 0.8165 | 0.7638 |
| YeastSCE| 5890| 75 | 0.7778 | 0.7562 | 0.8181 | 0.7469 |
| FlyDME  | 13190| 23 | 0.7703 | 0.7708 | 0.7798 | OOM |
| FlyDME  | 13190| 46 | 0.7035 | 0.7107 | 0.7047 | OOM |
| FlyDME  | 13190| 69 | 0.6693 | 0.6722 | 0.6717 | OOM |
| FlyDME  | 13190| 92 | 0.6414 | 0.6432 | 0.6483 | OOM |
| FlyDME  | 13190| 115| 0.6362 | 0.6241 | 0.6299 | OOM |
| WormCEL | 17519| 26 | 0.7535 | 0.7612 | 0.7620 | OOM |
| WormCEL | 17519| 52 | 0.6900 | 0.6977 | 0.6934 | OOM |
| WormCEL | 17519| 78 | 0.6521 | 0.6571 | 0.6592 | OOM |
| WormCEL | 17519| 104| 0.6271 | 0.6246 | 0.6320 | OOM |
| WormCEL | 17519| 130| 0.6094 | 0.6042 | 0.6153 | OOM |
| HumanHSA| 20967| 29 | 0.6728 | 0.6710 | 0.7100 | OOM |
| HumanHSA| 20967| 58 | 0.6140 | 0.6181 | 0.6293 | OOM |
| HumanHSA| 20967| 87 | 0.5858 | 0.5874 | 0.6007 | OOM |
| HumanHSA| 20967| 116| 0.5680 | 0.5678 | 0.5810 | OOM |
| HumanHSA| 20967| 145| 0.5513 | 0.5518 | 0.5681 | OOM |
Fig. 6 MSE with respect to inner dimension $k$ for different algorithms and problems. It is a visual depiction of Tables 2, 3, and 4. In case of the synthetic problems, $k$ is measured in % of the actual latent dimension $K$ (as in Table 2).
Four algorithms to solve symmetric multi-type NMTF problem

6.3 MSE vs. time efficiency

In this subsection we report numerical results where we compare how the MSE is decreasing with time. We decide to take 3 problem instances, one from each of the three data-sets introduced in Section 6.1. We took only 3 instances to have a more transparent figure, but based on our overall observations we claim that these results are representative for all the three data-sets and all four algorithms and show the preceding results in a different perspective.

Results clearly indicate that the methods which end with the lowest MSE on Figure 6 also spend much more time to reach these results. If we limit the time, which is a usual situation, then FPM method outperforms the others since it demonstrates the best convergence at the beginning and computes the lowest MSE in a given (short) time-frame.

7 Discussion and Conclusion

In this paper we developed four algorithms to solve a special variant of non-negative matrix factorization, called symmetric multi-type non-negative matrix
Fig. 8 Convergence of all four algorithms considered for three problems with different dimension $n$. 
Four algorithms to solve symmetric multi-type NMTF problem

The first algorithm was an adaptation of the fixed point method which is a classical approach in solving non-negative matrix factorization problems. The second algorithm was an adaptation of the block coordinate decent method combined with projected gradient method and exact line search, which has already been used in non-negative matrix factorization, but not to solve the SNMTF.

The third and the fourth method were based on so-called search space transformation. We substituted the factorization variables in such a way that the non-negative constraint vanished at the expense of the cost function. The gradient method with exact line search (GM-ELS) was based on a substitution of variables by their component-wise squares, while adaptive moment estimation method (ADAM) substituted the variables by their component-wise absolute values. In the former case, the objective function SE became a multivariate polynomial of order 12 while in the latter case, SE involved absolute values.

We implemented all the four algorithms into efficient code. The first two algorithms were implemented in Matlab, while the last two in Python with TensorFlow library. They are available on our GitLab profile [23].

All the four codes were extensively tested on three data-sets: the first was created by us and consists of synthetic data for which we know the optimum solution and wanted to reconstruct it back. We ran all four algorithms for inner dimensions equal to $k = 0.2K, 0.4K, \ldots , 1.2K$, where $K$ was real inner dimension. We observed that for $k < 1.0K$ the resulting SE is decreasing almost linearly with $k$, for all four algorithms, while for $k = K$ and $k = 1.2K$ only FPM and ADAM manage to decrease MSE below $10^{-2}$ in all cases.

The second set consists of real-world data-sets from machine learning and contains 5 real-world similarity or proximity matrices from different fields of data science. We used the background description of each data matrix to detect its inner dimension $K$ and then ran all four algorithms with inner dimension $k = 0.2K, 0.4K, \ldots , 1.0K$. We observed that ADAM returned the smallest SE for all cases while the other algorithms were also performing well, but not on all 5 data matrices.

The third data-set consists of 4 molecular interaction networks for 5 species: fission yeast (Yeast_SPO), baker’s yeast (Yeast_SCE), fruit fly (Fly_DME), nematode worm (Worm CEL) and human (Human_HSA). We ran again all the four algorithms with the inner dimensions equal to $k = [\sqrt{\pi}/5], 2[\sqrt{\pi}/5], \ldots , 5[\sqrt{\pi}/5]$. We observed that FPM, BCD, and ADAM perform similarly, regarding the MSE, while GM-ELS, on the other hand, is consistently worse than the others or, for larger data-sets, even runs out of memory.

We also analysed how MSE was decreasing with time. We took 3 problem instances, one from each of the three data-sets. The results we obtained clearly indicate that if we reasonably limit the time, then FPM method performs best since it demonstrates the best convergence at the beginning and computes the lowest MSE in a given (short) time-frame. However, if we can afford longer computation times, BCD and ADAM become also very competitive or even better regarding the MSE.

Our findings additionally explain why FPM is so popular in solving non-negative matrix factorization problem. It is indeed very good compromise between convergence quality, time complexity and coding complexity. With very little cod-
ing effort and small computing power we can obtain very good feasible solutions. On the other hand, ADAM as a very advanced algorithm, combined with very advanced computing library TensorFlow, demands much more programming skills and stronger hardware, but often produces better feasible solutions (regarding the MSE), if we provide enough computation time.

We continue our research in solving SNMTF by considering (i) other methods, like alternating direction method of multipliers (ADMM), (ii) by extensions of SNMTF which include e.g. orthogonality constraints, and (iii) by considering theoretical guaranties for the convergence for the methods we are using.

Acknowledgements The authors acknowledge the financial support from the Slovenian Research Agency (research core funding No. P2-0098, and projects No. J1-8155, No. PR-07606, No. N1-0071), from the European Research Council (ERC) Consolidator Grant (grant number 770827) and from the Spanish State Research Agency AEI 10.13039/501100011033 (grant number PID2019-105500GB-I00).

Conflict of interest

The authors declare that they have no conflict of interest.

Data availability

All the data-sets used in this paper and the Matlab and Python codes for the algorithms presented in this paper are available on our GitLab portal [23].

References

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015). URL http://tensorflow.org/. Software available from tensorflow.org

2. Altschul, S.F., Gish, W., Miller, W., Myers, E.W., Lipman, D.J.: Basic local alignment search tool. Journal of molecular biology 215(3), 403–410 (1990)

3. Asadi, S., Povh, J.: A block coordinate descent-based projected gradient algorithm for orthogonal non-negative matrix factorization. Mathematics 9(5), 540 (2021)

4. Atwood, G.R., Foster, W.W.: Transformation of bounded variables in simplex optimization techniques. Industrial & Engineering Chemistry Process Design and Development 12(4), 485–486 (1973)

5. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. Journal of machine learning research 13(Feb), 281–305 (2012)

6. Bertsekas, D.: Nonlinear Programming. Athena scientific optimization and computation series. Athena Scientific (2016). URL https://books.google.si/books?id=TwUjgEACAAJ

7. Boutsidis, C., Gallopoulos, E.: Svd based initialization: A head start for nonnegative matrix factorization. Pattern recognition 41(4), 1350–1362 (2008)

8. Cichocki, A., Zdunek, R., Phan, A.H., Amari, S.i.: Nonnegative matrix and tensor factorizations: applications to exploratory multi-way data analysis and blind source separation. John Wiley & Sons (2009)
9. Cichocki, A., Zdunek, R., Phan, A.H., Amari, S.i.: Nonnegative matrix and tensor factorizations: applications to exploratory multi-way data analysis and blind source separation. John Wiley & Sons (2009)

10. Davis, D., Drusvyatskiy, D., Kakade, S., Lee, J.D.: Stochastic subgradient method converges on tame functions. Foundations of computational mathematics 20(1), 119–154 (2020)

11. Del Buono, N., Pio, G.: Non-negative matrix tri-factorization for co-clustering: An analysis of the block matrix. Information Sciences 301, 13–26 (2015). DOI 10.1016/j.ins.2014.12.058

12. Dickinson, P.J., Gijben, L.: On the computational complexity of membership problems for the completely positive cone and its dual. Computational optimization and applications 57(2), 403–415 (2014)

13. Ding, C., Li, T., Peng, W., Park, H.: Orthogonal nonnegative matrix t-factorizations for clustering. In: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 126–135. ACM (2006)

14. Feng, S., Krim, H., Kogan, I.: 3d face recognition using euclidean integral invariants signature. In: Statistical Signal Processing, 2007. SSP '07. IEEE/SP 14th Workshop on, pp. 156–160 (2007)

15. Furini, F., Traversi, E., Belotti, P., Frangioni, A., Gleixner, A., Gould, N., Liberti, L., Lodi, A., Misener, R., Mittelmann, H., et al.: Qplib: a library of quadratic programming instances. Mathematical Programming Computation 11(2), 237–265 (2019)

16. Gillis, N.: The why and how of nonnegative matrix factorization. In: J. Suykens, M. Signoretto, A. Argyriou (eds.) Regularization, optimization, kernels, and support vector machines, pp. 257–291. Chapman & Hall/CRC, New York (2015). DOI https://doi.org/10.1201/b17558

17. Gligorijević, V., Janjić, V., Pržulj, N.: Integration of molecular network data reconstructs gene ontology. Bioinformatics 30(17), i594–i600 (2014)

18. Gligorijević, V., Malod-Dognin, N., Pržulj, N.: Fuse: multiple network alignment via data fusion. Bioinformatics 32(8), 1195–1203 (2015)

19. Gligorijević, V., Malod-Dognin, N., Pržulj, N.: Integrative methods for analyzing big data in precision medicine. Proteomics 16(5), 741–758 (2016)

20. Gligorijević, V., Malod-Dognin, N., Pržulj, N.: Patient-specific data fusion for cancer stratification and personalized treatment. In: Biocomputing 2016: Proceedings of the Pacific Symposium, pp. 321–332. World Scientific (2016)

21. Ho, N.D.: Nonnegative matrix factorization algorithms and applications. Ph.D. thesis, PhD thesis, Université catholique de Louvain (2008)

22. Höfmann, T., Buhmann, J.M.: Pairwise data clustering by deterministic annealing. IEEE Trans. Pattern Anal. Mach. Intell. 19(1), 1–14 (1997)

23. Hrga, T., Hribar, R., Povh, J.: symmetric NMTF (2020). https://repo.ijs.si/hribarr/symmetric-nmtf

24. Huang, K., Sidiroopoulos, N.D., Swami, A.: Non-negative matrix factorization revisited: Uniqueness and algorithm for symmetric decomposition. IEEE Transactions on Signal Processing 62(1), 211–224 (2013)

25. Jain, A.K., Zongker, D.: Representation and recognition of handwritten digits using deformable templates. IEEE Trans. Pattern Anal. Mach. Intell. 19(12), 1386–1391 (1997)

26. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)

27. Lee, D.D., Seung, H.S.: Algorithms for non-negative matrix factorization. In: Advances in neural information processing systems, pp. 556–562 (2001)

28. Liu, C.J.: Projected gradient methods for nonnegative matrix factorization. Neural computation 19(10), 2756–2779 (2007)

29. Lu, S., Hong, M., Wang, Z.: A nonconvex splitting method for symmetric nonnegative matrix factorization: Convergence analysis and optimality. IEEE Transactions on Signal Processing 65(12), 3120–3135 (2017)
32. Ma, X., Dong, D.: Evolutionary nonnegative matrix factorization algorithms for community detection in dynamic networks. IEEE Transactions on Knowledge & Data Engineering 29(05), 1045–1058 (2017). DOI 10.1109/TKDE.2017.2657752
33. Malod-Dognin, N., Petschnigg, J., Windels, S.F., Povh, J., Hemingway, H., Ketteler, R., Pržulj, N.: Towards a data-integrated cell. Nature communications 10(1), 1–13 (2019)
34. Malick, J., Povh, J., Rendl, F., Wiegele, A.: Regularization methods for semidefinite programming. SIAM Journal on Optimization 20(1), 336–356 (2009)
35. Malod-Dognin, N., Petschnigg, J., Windels, S.F., Povh, J., Hemingway, H., Ketteler, R., Pržulj, N.: Towards a data-integrated cell. Nature communications 10(1), 1–13 (2019)
36. MATLAB: 9.6.0.1072779 (R2019a). The MathWorks Inc., Natick, Massachusetts (2019)
37. Mirzal, A.: A convergent algorithm for orthogonal nonnegative matrix factorization. Journal of Computational and Applied Mathematics 260, 149–166 (2014)
38. Mirzal, A.: A convergent algorithm for bi-orthogonal nonnegative matrix tri-factorization. arXiv preprint arXiv:1710.11478 (2017)
39. Obayashi, T., Kagaya, Y., Aoki, Y., Tadaka, S., Kinoshita, K.: Coexpresdb v7: a gene coexpression database for 11 animal species supported by 23 coexpression platforms for technical evaluation and evolutionary inference. Nucleic acids research 47(D1), D55–D62 (2019)
40. Oughtred, R., Stark, C., Breitkreutz, B.J., Rust, J., Boucher, L., Chang, C., Kolas, N., O'Donnell, L., Leung, G., McAdam, R., et al.: The biogrid interaction database: 2019 update. Nucleic acids research 47(D1), D529–D541 (2019)
41. Oughtred, R., Stark, C., Breitkreutz, B.J., Rust, J., Boucher, L., Chang, C., Kolas, N., O'Donnell, L., Leung, G., McAdam, R., et al.: The biogrid interaction database: 2019 update. Nucleic acids research 47(D1), D529–D541 (2019)
42. Philips, S., Pitton, J., Atlas, L.: Perceptual feature identification for active sonar echoes. In: OCEANS 2006, pp. 1–6 (2006)
43. Phillips, S., Pitton, J., Atlas, L.: Perceptual feature identification for active sonar echoes. In: OCEANS 2006, pp. 1–6 (2006)
44. Povh, J., Rendl, F., Wiegele, A.: A boundary point method to solve semidefinite programs. Computing 78(3), 277–286 (2006)
45. Pržulj, N., Malod-Dognin, N.: Network analytics in the age of big data. Science 353(6295), 123–124 (2016)
46. Reddi, S.J., Kale, S., Kumar, S.: On the convergence of Adam and beyond. arXiv preprint arXiv:1904.09237 (2019)
47. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by backpropagating errors. nature 323(6088), 533–536 (1986)
48. Saito, S., Hirata, Y., Sasahara, K., Suzuki, H.: Tracking time evolution of collective attention clusters in twitter: Time evolving nonnegative matrix factorization. PLOS ONE 10(9), 1–17 (2015). DOI 10.1371/journal.pone.0139085. URL https://doi.org/10.1371/journal.pone.0139085
49. Schleif, F., Gisbrecht, A.: A data analysis of (non-)metric proximities at linear costs. In: E.R. Hancock, M. Pelillo (eds.) Similarity-Based Pattern Recognition - Second International Workshop, SIMBAD 2013, York, UK, July 3-5, 2013. Proceedings, Lecture Notes in Computer Science, vol. 7953, pp. 59–74. Springer (2013). DOI 10.1007/978-3-642-39140-8_4. URL https://doi.org/10.1007/978-3-642-39140-8_4
50. Someya, H., Yamamura, M.: A robust real-coded evolutionary algorithm with toroidal search space conversion. Soft Computing 9(4), 254–269 (2005)
51. Stanfill, C., Waltz, D.: Toward memory-based reasoning. Commun. ACM 29(12), 1213–1228 (1986)
52. Szklarczyk, D., Gable, A.L., Lyon, D., Junge, A., Wyder, S., Huerta-Cepas, J., Simonovic, M., Doncheva, N.T., Morris, J.H., Bork, P., et al.: String v11: protein–protein association networks with increased coverage, supporting functional discovery in genome-wide experimental datasets. Nucleic acids research 47(D1), D607–D613 (2019)
53. Tseng, P.: Convergence of a block coordinate descent method for nondifferentiable minimization. Journal of optimization theory and applications 109(3), 475–494 (2001)
54. Tsutsui, S.: Multi-parent recombination in genetic algorithms with search space boundary extension by mirroring. In: International Conference on Parallel Problem Solving from Nature, pp. 428–437. Springer (1998)
55. Vavasis, S.A.: On the complexity of nonnegative matrix factorization. SIAM Journal on Optimization 20(3), 1364–1377 (2009)
Appendix

A Calculation of gradient

Here we offer a derivation of \(^{(24)}\) and \(^{(25)}\) presented in Section 4.1. We use notation \(X_{\mu\nu}\) for component of matrix \(X\) in row \(\mu\) and column \(\nu\) and \(\delta_{ij}\) for the Kronecker delta. Objective function being differentiated is

\[
SE = \sum_{i=1}^{N} \| R_i - f(\tilde{G})f(\tilde{S}_i)f(\tilde{G})^\top \|^2 = \sum_{i=1}^{N} \| Z_i \|^2 = \sum_{i=1}^{N} \sum_{\mu,\nu} |Z_i|_{\mu\nu}^2,
\]

where \(Z_i\) is defined in \(^{(23)}\). Let us differentiate \(SE\) with respect to a single component of matrix \(\tilde{S}_i\).

\[
\frac{\partial SE}{\partial [\tilde{S}_i]_{\rho\sigma}} = -2 \sum_{i=1}^{N} \sum_{\mu,\nu} Z_i |_{\mu\nu} \frac{\partial \left[f(\tilde{G})f(\tilde{S}_i)f(\tilde{G})^\top\right]_{\mu\nu}}{\partial [\tilde{S}_i]_{\rho\sigma}}
\]

\[
= -2 \sum_{i=1}^{N} \sum_{\mu,\nu} Z_i |_{\mu\nu} \frac{\partial f(\tilde{G})f(\tilde{S}_i)f(\tilde{G})^\top}{\partial [\tilde{S}_i]_{\rho\sigma}} Z_i |_{\mu\nu} \delta_{\rho\sigma}
\]

\[
= -2 \sum_{i=1}^{N} \sum_{\mu,\nu} Z_i |_{\mu\nu} f'(\tilde{G}|_{\mu\nu}) f'(\tilde{S}_i|_{\rho\sigma})
\]

\[
= -2 \sum_{i=1}^{N} f'(\tilde{S}_i|_{\rho\sigma}) \left[f(\tilde{G})^\top Z_i f(\tilde{G})\right]_{\rho\sigma}
\]
This result can be written more compactly as
\[ \Delta S_i = \nabla_{\tilde{S}_i} SE = -2 \sum_{i=1}^{N} f'((\tilde{S}_i) \odot (f(G)^T Z_i f(G))), \]
which proves (24).

Using the same steps for \( G \), we obtain (25).

B Calculation of polynomial used in the exact line search

Here we show exactly how the polynomial (30) used by GM-ELS is calculated. This polynomial tells how \( SE \) changes with respect to step size \( t \) when making a move in direction of gradient. For sake of brevity we use notation \( X^2 = \mathbb{X} \odot \mathbb{X} \).

\[ p(t) = \sum_{i=1}^{N} ||R_i - (G + t \Delta G)^2(\tilde{S}_i + t \Delta \tilde{S}_i)^2(G + t \Delta \tilde{G})^2||^2 = \sum_{j=0}^{12} c_j t^j \]

To calculate coefficients \( c_j \) we first express how matrix \( Z_i \) changes with respect to \( t \).

\[ Z_i(t) = R_i - (G + t \Delta G)^2(\tilde{S}_i + t \Delta \tilde{S}_i)^2(G + t \Delta \tilde{G})^2 = \sum_{j=0}^{6} A_{ij} t^j, \]

where matrices \( A_{ij} \) can be calculated using simple expansion steps and are equal to

\[
\begin{align*}
A_{i0} &= R_i - G^2 S_i^2 G^2^T \\
A_{i1} &= -2\tilde{G}^2 S_i^2 (G \odot \Delta \tilde{G})^T - 2\tilde{G}^2 (\tilde{S}_i \odot \Delta \tilde{S}_i) G^2 G^T - 2(G \odot \Delta G) S_i^2 \tilde{G}^2 G^T \\
A_{i2} &= -G^2 S_i^2 \Delta G^2^T - 4G^2 (\tilde{S}_i \odot \Delta \tilde{S}_i) (G \odot \Delta G)^T - G^2 \Delta S_i^2 G^2^T - \Delta G^2 S_i^2 \tilde{G}^2 G^T \\
&- 4(G \odot \Delta G) (\tilde{S}_i \odot \Delta \tilde{S}_i) G^2 G^T - 4(G \odot \Delta G) S_i^2 \tilde{G}^2 G^T \\
A_{i3} &= -2\tilde{G}^2 (\tilde{S}_i \odot \Delta \tilde{S}_i) \Delta \tilde{G} G^T - 2\tilde{G}^2 \Delta \tilde{S}_i^2 (G \odot \Delta G)^T - 2(G \odot \Delta G) \tilde{S}_i \Delta \tilde{G}^2 G^T \\
&- 2\Delta G \tilde{S}_i^2 (G \odot \Delta G)^T - 2\Delta G^2 (\tilde{S}_i \odot \Delta \tilde{S}_i) G^2 G^T \\
A_{i4} &= -G^2 \Delta \tilde{S}_i^2 \Delta \tilde{G} G^T - 4(G \odot \Delta G) (\tilde{S}_i \odot \Delta \tilde{S}_i) \Delta \tilde{G} G^T - 4(G \odot \Delta \tilde{G}) S_i \Delta \tilde{G}^2 G^T \\
&- 4\Delta G^2 S_i \Delta \tilde{G}^2 G^T - 4\Delta G \tilde{S}_i \Delta \tilde{G}^2 G^T - \Delta \tilde{G}^2 \Delta \tilde{S}_i^2 G^2 G^T \\
A_{i5} &= -2(G \odot \Delta \tilde{G}) \Delta \tilde{S}_i^2 \Delta \tilde{G} G^T - 2\Delta G^2 (\tilde{S}_i \odot \Delta \tilde{S}_i) \Delta \tilde{G} G^T - 2\Delta \tilde{G}^2 S_i \Delta \tilde{G}^2 G^T \\
A_{i6} &= -\Delta \tilde{G}^2 \Delta \tilde{S}_i^2 \Delta \tilde{G} G^T.
\end{align*}
\]

Knowing matrices \( A_{ij} \), the element-wise square of \( Z_i(t) \) is

\[ Z_i^2(t) = \sum_{j=0}^{12} B_{ij} t^j \]

with

\[ B_{ij} = \sum_{r} A_{ir} \odot A_{i,j-r}. \]  

(35)
equation (35) is an application of equivalence of polynomial multiplication and discrete convolution. Therefore, the sum is over all \( r \) values that lead to legal indices for both \( A_{ir} \) and \( A_{i,j-r} \) which are all \( r \) between \( \max(0,j-6) \) and \( \min(j,6) \). Remembering

\[ p(t) = \sum_{i=1}^{N} \sum_{\mu, \nu} [Z_i^2(t)]_{\mu, \nu} = \sum_{j=0}^{12} c_j t^j \]
we can now express

\[ c_j = \sum_{i=1}^{N} \sum_{\mu,\nu} [B_{ij}]_{\mu\nu}. \quad (36) \]

Calculation of coefficients \( c_j \) requires far higher number of matrix multiplications compared to the number needed for gradient calculation. When implemented in TensorFlow calculation of \( c_j \) is approximately 7 times more expensive than gradient calculation.