Algorithms for Approximate Subtropical Matrix Factorization

Sanjar Karaev and Pauli Miettinen
Max-Planck-Institut für Informatik
Saarland Informatics Campus
Saarbrücken, Germany
{skaraev,pmiettin}@mpi-inf.mpg.de

Abstract

Matrix factorization methods are important tools in data mining and analysis. They can be used for many tasks, ranging from dimensionality reduction to visualization. In this paper we concentrate on the use of matrix factorizations for finding patterns from the data. Rather than using the standard algebra – and the summation of the rank-1 components to build the approximation of the original matrix – we use the subtropical algebra, which is an algebra over the nonnegative real values with the summation replaced by the maximum operator. Subtropical matrix factorizations allow "winner-takes-it-all" interpretations of the rank-1 components, revealing different structure than the normal (nonnegative) factorizations. We study the complexity and sparsity of the factorizations, and present a framework for finding low-rank subtropical factorizations. We present two specific algorithms, called Capricorn and Cancer, that are part of our framework. They can be used with data that has been corrupted with different types of noise, and with different error metrics, including the sum-of-absolute differences, Frobenius norm, and Jensen–Shannon divergence. Our experiments show that the algorithms perform well on data that has subtropical structure, and that they can find factorizations that are both sparse and easy to interpret.

1 Introduction

Finding simple patterns that can be used to describe the data is one of the main problems in data mining. The data mining literature knows many different techniques for this general task, but one of the most common pattern finding technique rarely gets classified as such. Matrix factorizations (or decompositions, these two terms are used interchangeably in this paper) represent the given input matrix $\boldsymbol{A}$ as a product of two (or more) factor matrices, $\boldsymbol{A} \approx \boldsymbol{BC}$. This standard formulation of matrix factorizations makes their pattern mining nature less obvious, but let us write the matrix product $\boldsymbol{BC}$ as a sum of rank-1 matrices, $\boldsymbol{BC} = \boldsymbol{F}_1 + \boldsymbol{F}_2 + \cdots + \boldsymbol{F}_k$, where $\boldsymbol{F}_i$ is the outer product of the ith column of $\boldsymbol{B}$ and the ith row of $\boldsymbol{C}$. Now it becomes clear that the rank-1 matrices $\boldsymbol{F}_i$ are the "simple patterns" and the matrix factorization is finding k such patterns whose sum is a good approximation of the original data matrix.

This so-called "component interpretation" (Skillicorn, 2007) is more appealing with some factorizations than with others. For example, the classical singular value decomposition (SVD) does not easily admit such an interpretation, as the components are not easy to interpret without knowing the earlier components. On the other hand, the motivation for the nonnegative matrix factorization (NMF) often comes from the component interpretation, as can be seen, for example, in the famous "parts of faces" figures of Lee and Seung (1999). The "parts-of-whole" interpretation is in the hearth of NMF: every rank-1 component adds something to the overall decomposition, and never removes anything. This aids with the interpretation of the components, and is also often claimed to yield sparse factors, although this latter point is more contentious (Hoyer, 2004).

Perhaps the reason why matrix factorization methods are not often considered as pattern mining methods is that the rank-1 matrices are summed together to build the full data. Hence, it is rare for any rank-1 component to explain any part of the input matrix alone. But the use of summation as a way to aggregate the rank-1 components can be considered to be "merely" a consequence of the fact that we are using the standard algebra. If we change the algebra – in particular, if we change how we define the summation – we change the operator used for the aggregation. In this work, we propose to use the *maximum* operator to define the summation over the nonnegative matrices, giving us what is known as the *subtropical algebra*. As the aggregation of the rank-1 factors is now the element-wise maximum, we obtain what we call the "winner-takes-it-all" interpretation: the final value of each element in the approximation is defined only by the largest value in the corresponding element in the rank-1 matrices.

Not only does the subtropical algebra give us the intriguing winner-takes-it-all interpretation, it also provides guarantees about the sparsity of the factors, as we will show in Section 3.2. Furthermore, the different algebra means that we are finding different factorizations compared to NMF (or SVD). The emphasis here is on the word *different*: the factorizations can be better or worse in terms of the reconstruction error – we will discuss this in Section 3.3 – but the patterns they find are usually different to those found by NMF. Unfortunately, the related optimization problems are NP-hard (see Section 3.1). In Section 4, we will develop a general framework, called `Equator`, for finding approximate, low-rank subtropical decompositions, and we will present two instances of this framework, tailored towards different types of data and noise, called `Capricorn` and `Cancer`.[1] `Capricorn` assumes integer data with noise that randomly flips the value to some other integer, whereas `Cancer` assumes continuous-valued data with standard Gaussian noise.

Our experiments (see Section 5) show that both `Capricorn` and `Cancer` work well on datasets that have the kind of noise they are designed for, and they outperform SVD and different NMF methods when data has subtropical structure. On real-world data, `Cancer` is usually the better of the two, although in terms of reconstruction error, neither of the methods can challenge SVD. On the other hand, both `Cancer` and `Capricorn` return interpretable results that show different aspects of the data compared to factorizations made under the standard algebra.

[1]This work is a combined and extended version of our preliminary papers that described these algorithms (Karaev and Miettinen, 2016a,b).

2 Notation and Basic Definitions

Basic notation. Throughout this paper, we will denote a matrix by upper-case boldface letters ($\boldsymbol{A}$), and vectors by lower-case boldface letters ($\boldsymbol{a}$). The ith row of matrix $\boldsymbol{A}$ is denoted by $\boldsymbol{A}_i$ and the jth column by $\boldsymbol{A}^j$. The matrix $\boldsymbol{A}$ with the ith column removed is denoted by $\boldsymbol{A}^{-i}$, and $\boldsymbol{A}_{-i}$ is the respective notation for $\boldsymbol{A}$ with a removed row. Most matrices and vectors in this paper are restricted to the nonnegative real numbers $\mathbb{R}_+ = [0, \infty)$.

We use the shorthand $[n]$ to denote the set $\{1, 2, \ldots, n\}$.

Algebras. In this paper we consider matrix factorization over so called *max-times* (or *subtropical*) *algebra*. It differs from the standard algebra of real numbers in that addition is replaced with the operation of taking the maximum. Also the domain is restricted to the set of nonnegative real numbers.

Definition 1. The *max-times* (or *subtropical*) algebra is a set $\mathbb{R}_+$ of nonnegative real numbers together with operations $a \boxplus b = \max\{a, b\}$ (addition) and $a \boxdot b = ab$ (multiplication) defined for any $a, b \in \mathbb{R}_+$. The identity element for addition is 0 and for multiplication it is 1.

In the future we will use the notation $a \boxplus b$ and $\max\{a, b\}$ and the names *max-times* and *subtropical* interchangeably. It is straightforward to see that the max-times algebra is a *dioid*, that is, a semiring with idempotent addition ($a \boxplus a = a$). It is important to note that subtropical algebra is anti-negative, that is, there is no subtraction operation.

A very closely related algebraic structure is the *max-plus* (*tropical*) algebra (see e.g. Akian et al., 2007).

Definition 2. The *max-plus* (or *tropical*) algebra is defined over the set of extended real numbers $\mathbb{R} \cup \{-\infty\}$ with operations $a \oplus b = \max\{a, b\}$ (addition) and $a \odot b = a + b$ (multiplication). The identity elements for addition and multiplication are $-\infty$ and 0, respectively.

The tropical and subtropical algebras are isomorphic (Blondel et al., 2000), which can be seen by taking the logarithm of the subtropical algebra or the exponent of the tropical algebra (with the conventions that $\log 0 = -\infty$ and $\exp(-\infty) = 0$). Thus, most of the results we prove for subtropical algebra can be extended to their tropical analogues, although caution should be used when dealing with approximate matrix factorizations. The latter is because, as we will see in Theorem 6, the *reconstruction error* of an approximate matrix factorization under the two different algebras does not transfer directly.

Matrix products and ranks. The matrix product over the subtropical algebra is defined in the natural way:

Definition 3. The *max-times matrix product* of two matrices $\boldsymbol{B} \in \mathbb{R}_+^{n \times k}$ and $\boldsymbol{C} \in \mathbb{R}_+^{k \times m}$ is defined as

$$(\boldsymbol{B} \boxtimes \boldsymbol{C})_{ij} = \max_{s=1}^{k} \boldsymbol{B}_{is} \boldsymbol{C}_{sj} \ . \tag{1}$$

We will also need the matrix product over the *tropical* algebra.

Definition 4. For two matrices $\boldsymbol{B} \in (\mathbb{R} \cup \{-\infty\})^{n \times k}$ and $\boldsymbol{C} \in (\mathbb{R} \cup \{-\infty\})^{k \times m}$, their *tropical matrix product* is defined as

$$(\boldsymbol{B} \diamond \boldsymbol{C})_{ij} = \max_{s=1}^{k}\{\boldsymbol{B}_{is} + \boldsymbol{C}_{sj}\} \ . \tag{2}$$

The *matrix rank* over the subtropical algebra can be defined in many ways, depending on which definition of the normal matrix rank is taken as the starting point. We will discuss different subtropical ranks in detail in Section 3.4. Here we give the main definition of the rank we are using throughout this paper, the so-called *Schein* (or *Barvinok*) *rank* of a matrix.

Definition 5. The *max-times (Schein or Barvinok) rank* of a matrix $\boldsymbol{A} \in \mathbb{R}_{+}^{n \times m}$ is the least integer k such that $\boldsymbol{A}$ can be expressed as an element-wise maximum of k rank-1 matrices, $\boldsymbol{A} = \boldsymbol{F}_1 \boxplus \boldsymbol{F}_2 \boxplus \cdots \boxplus \boldsymbol{F}_k$. Matrix $\boldsymbol{F} \in \mathbb{R}_{+}^{n \times m}$ has subtropical (Schein/Barvinok) rank of 1 if there exist column vectors $\boldsymbol{x} \in \mathbb{R}_{+}^{n}$ and $\boldsymbol{y} \in \mathbb{R}_{+}^{m}$ such that $\boldsymbol{F} = \boldsymbol{x}\boldsymbol{y}^T$. Matrices with subtropical Schein (or Barvinok) rank of 1 are called *blocks.*

When it is clear from the context, we will use the term *rank* (or *subtropical rank*) without other qualifiers to denote the subtropical Schein/Barvinok rank.

Special matrices. The final concepts we need in this paper are *pattern matrices* and *dominating matrices.*

Definition 6. A *pattern* of a matrix $\boldsymbol{A} \in \mathbb{R}^{n \times m}$ is an n-by-m binary matrix $\boldsymbol{P}$ such that $\boldsymbol{P}_{ij} = 0$ if and only if $\boldsymbol{A}_{ij} = 0$, and otherwise $\boldsymbol{P}_{ij} = 1$. We denote the pattern of $\boldsymbol{A}$ by $p(\boldsymbol{A})$.

Definition 7. Let $\boldsymbol{A}$ and $\boldsymbol{X}$ be matrices of the same size, and let Γ be a subset of their indices. Then if for all indices $(i,j) \in \Gamma$, $\boldsymbol{X}_{ij} \geq \boldsymbol{A}_{ij}$, we say that $\boldsymbol{X}$ *dominates* $\boldsymbol{A}$ *within* Γ. If Γ spans the entire size of $\boldsymbol{A}$ and $\boldsymbol{X}$, we simply say that $\boldsymbol{X}$ *dominates* $\boldsymbol{A}$. Correspondingly, $\boldsymbol{A}$ is said to be *dominated by* $\boldsymbol{X}$.

Main problem definition. Now that we have sufficient notation, we can formally introduce the main problem considered in the paper.

Problem 1 (Approximate subtropical rank-k matrix factorization)**.** Given a matrix $\boldsymbol{A} \in \mathbb{R}_{+}^{n \times m}$ and an integer $k > 0$, find factor matrices $\boldsymbol{B} \in \mathbb{R}_{+}^{n \times k}$ and $\boldsymbol{C} \in \mathbb{R}_{+}^{k \times m}$ minimizing

$$E(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}) = \lVert \boldsymbol{A} - \boldsymbol{B} \boxtimes \boldsymbol{C} \rVert \ . \tag{3}$$

Here we have deliberately not specified any particular norm. Depending on the circumstanses, different matrix norms can be used, but in this paper we will consider the two most natural choices – the Frobenius and L_1 norms.

3 Theory

Our main contributions in this paper are the algorithms for the subtropical matrix factorization. But before we present them, it is important to understand the theoretical aspects of subtropical factorizations. We will start by studying the computational complexity of Problem 1. After that, we will show that the dominated subtropical factorizations of sparse matrices are sparse. Finally, we compare the subtropical factorizations to factorizations over other algebras, and discuss different ways to define the subtropical rank, and the relationships between these ranks.

3.1 Computational complexity

The computational complexity of different matrix factorization problems varies. For example, SVD can be computed in polynomial time (Golub and Van Loan, 2012), while NMF is NP-hard (Vavasis, 2009). Unfortunately, the subtropical factorization is also NP-hard.

Theorem 1. *Computing the max-times matrix rank is an* NP*-hard problem, even for binary matrices.*

The theorem is a direct consequence of the following theorem by Kim and Roush (2005):

Theorem 2 (Kim and Roush, 2005)**.** *Computing the max-plus (tropical) matrix rank is* NP*-hard, even for matrices that take values only from* $\{-\infty, 0\}$.

While computing the rank deals with exact decompositions, its hardness automatically makes any approximation algorithm with provable multiplicative guarantees unlikely to exist, as the following corollary shows.

Corollary 3. *It is* NP*-hard to approximate Problem 1 to within any polynomially computable factor.*

Proof. Any algorithm that can approximate Problem 1 to within a factor α must find a decomposition of error $\alpha \cdot 0 = 0$ if the input matrix has exact max-times rank-k decomposition. As this implies solving the max-times rank, per Theorem 1 it is only possible if P=NP. □

3.2 Sparsity of the factors

It is often desirable to obtain sparse factor matrices if the original data is sparse, as well, and the sparsity of its factors is frequently mentioned as one of the benefits of using NMF (see, e.g. Hoyer, 2004). In general, however, the factors obtained by NMF might not be sparse, but if we restrict ourselves to *dominated* decompositions, Gillis and Glineur (2010) showed that the sparsity of the factors cannot be less than the sparsity of the original matrix.

The proof of Gillis and Glineur (2010) relies on the anti-negativity, and hence their proof is easy to adapt to max-times setting. Let the *sparsity* of an n-by-m matrix $\boldsymbol{A}$, $s(\boldsymbol{A})$, be defined as

$$s(\boldsymbol{A}) = \frac{nm - \eta(\boldsymbol{A})}{nm}\ , \tag{4}$$

where $\eta(\boldsymbol{A})$ is the number of nonzero elements in $\boldsymbol{A}$. Now we have

Theorem 4. *Let matrices* $\boldsymbol{B} \in \mathbb{R}_+^{n\times k}$ *and* $\boldsymbol{C} \in \mathbb{R}_+^{k\times m}$ *be such that their max-times product is dominated by an* n*-by-*m *matrix* $\boldsymbol{A}$*. Then the following estimate holds*

$$s(\boldsymbol{B}) + s(\boldsymbol{C}) \geq s(\boldsymbol{A})\ . \tag{5}$$

Proof. The proof follows that of Gillis and Glineur (2010). We first prove (5) for $k = 1$. Let $\boldsymbol{b} \in \mathbb{R}^n_+$ and $\boldsymbol{c} \in \mathbb{R}^m_+$ be such that $\boldsymbol{b}_i \boldsymbol{c}_j^T \leq \boldsymbol{A}_{ij}$ for all $1 \leq i \leq n$, $1 \leq j \leq m$. Since $(\boldsymbol{bc}^T)_{ij} > 0$ if and only if $\boldsymbol{b}_i > 0$ and $\boldsymbol{c}_j > 0$, we have

$$\eta(\boldsymbol{bc}^T) = \eta(\boldsymbol{b})\, \eta(\boldsymbol{c}) \ . \tag{6}$$

By (4) we have $\eta(\boldsymbol{bc}^T) = nm(1 - s(\boldsymbol{bc}^T))$, $\eta(\boldsymbol{b}) = n(1 - s(\boldsymbol{b}))$ and $\eta(\boldsymbol{c}) = m(1 - s(\boldsymbol{c}))$. Plugging these expressions into (6) we obtain $(1 - s(\boldsymbol{bc}^T)) = (1 - s(\boldsymbol{b}))(1 - s(\boldsymbol{c}))$. Hence, the sparsity in a rank-1 dominated approximation of $\boldsymbol{A}$ is

$$s(\boldsymbol{b}) + s(\boldsymbol{c}) \geq s(\boldsymbol{bc}^T) \ . \tag{7}$$

From (7) and the fact that the number of nonzero elements in $\boldsymbol{bc}^T$ is no greater than in $\boldsymbol{A}$, it follows that

$$s(\boldsymbol{b}) + s(\boldsymbol{c}) \geq s(\boldsymbol{A}) \ . \tag{8}$$

Now let $\boldsymbol{B} \in \mathbb{R}^{n \times k}_+$ and $\boldsymbol{C} \in \mathbb{R}^{k \times m}_+$ be such that $\boldsymbol{B} \boxtimes \boldsymbol{C}$ is dominated by $\boldsymbol{A}$. Then $\boldsymbol{B}_{il}\boldsymbol{C}_{lj} \leq \boldsymbol{A}_{ij}$ for all $i \in [n]$, $j \in [m]$, and $l \in [k]$, which means that for each $l \in [k]$, $\boldsymbol{B}^l \boldsymbol{C}_l$ is dominated by $\boldsymbol{A}$. To complete the proof observe that $s(\boldsymbol{B}) = k^{-1} \sum_{l=1}^{k} \boldsymbol{B}^l$ and $s(\boldsymbol{C}) = k^{-1} \sum_{l=1}^{k} \boldsymbol{C}_l$ and that for each l estimate (8) holds. □

3.3 Relation to other algebras

Let us now study how the max-times algebra relates to other algebras, especially the standard, the Boolean, and the max-plus algebras. For the first two, we compare the ranks, and for the last, the reconstruction error.

Let us start by considering the Boolean rank of a binary matrix. The *Boolean (Schein or Barvinok) rank* is the following problem:

Problem 2 (Boolean rank)**.** Given a matrix $\boldsymbol{A} \in \{0,1\}^{n \times m}$ and an integer k, are there matrices $\boldsymbol{B} \in \{0,1\}^{n \times k}$ and $\boldsymbol{C} \in \{0,1\}^{k \times m}$ such that $\boldsymbol{A} = \boldsymbol{B} \circ \boldsymbol{C}$, where $\circ$ is the *Boolean matrix product*,

$$(\boldsymbol{B} \circ \boldsymbol{C})_{ij} = \bigvee_{l=1}^{k} \boldsymbol{B}_{il}\boldsymbol{C}_{lj} \ .$$

Lemma 5. *If $\boldsymbol{A}$ is a binary matrix, then its Boolean and subtropical ranks are the same.*

Proof. We will prove the claim by first showing that the Boolean rank of a binary matrix is no less than the subtropical rank, and then showing that it is no larger, either. For the first direction, let the Boolean rank of $\boldsymbol{A}$ be k, and let $\boldsymbol{B}$ and $\boldsymbol{C}$ be binary matrices such that $\boldsymbol{B}$ has k columns and $\boldsymbol{A} = \boldsymbol{B} \circ \boldsymbol{C}$. It is easy to see that $\boldsymbol{B} \circ \boldsymbol{C} = \boldsymbol{B} \boxtimes \boldsymbol{C}$, and hence, the subtropical rank of $\boldsymbol{A}$ is no more than k.

For the second direction, we will actually show a slightly stronger claim: Let $\boldsymbol{A} \in \mathbb{R}^{n \times m}_+$ and let $p(\boldsymbol{A})$ be its pattern. Then the Boolean rank of $p(\boldsymbol{A})$ is never more than the subtropical rank of $\boldsymbol{A}$. As $p(\boldsymbol{A}) = \boldsymbol{A}$ for a binary $\boldsymbol{A}$, the claim follows. To prove the claim, let $\boldsymbol{A} \in \mathbb{R}^{n \times m}_+$ have subtropical rank of k and let

$\boldsymbol{B} \in \mathbb{R}_+^{n\times k}$ and $\boldsymbol{C} \in \mathbb{R}_+^{k\times m}$ be such that $\boldsymbol{A} = \boldsymbol{B} \boxtimes \boldsymbol{C}$. Let (i,j) be such that $\boldsymbol{A}_{ij} = 0$. By definition, $\max_{l=1}^{k} \boldsymbol{B}_{il}\boldsymbol{C}_{lj} = 0$, and hence

$$\max_{l=1}^{k} p(\boldsymbol{B})_{il} p(\boldsymbol{C})_{lj} = \bigvee_{l=1}^{k} p(\boldsymbol{B})_{il} p(\boldsymbol{C})_{lj} = 0 \ . \tag{9}$$

On the other hand, if (i,j) is such that $\boldsymbol{A}_{ij} > 0$, then there exists l such that $\boldsymbol{B}_{il}, \boldsymbol{C}_{lj} > 0$ and consequently,

$$\max_{l=1}^{k} p(\boldsymbol{B})_{il} p(\boldsymbol{C})_{lj} = \bigvee_{l=1}^{k} p(\boldsymbol{B})_{il} p(\boldsymbol{C})_{lj} = 1 \ . \tag{10}$$

Combining (9) and (10) gives us

$$p(\boldsymbol{A}) = p(\boldsymbol{B}) \circ p(\boldsymbol{C}) \ , \tag{11}$$

showing that the Boolean rank of $p(\boldsymbol{A})$ is at most k. $\square$

Notice that Lemma 5 also furnishes us with another proof of Theorem 1, as the computation of the Boolean rank is an NP-complete problem (see, e.g. Miettinen, 2009). Notice also that while the Boolean rank of the pattern is never more than the subtropical rank of the original matrix, it can be much less. This is easy to see by considering a matrix with no zeroes: it can have arbitrarily large subtropical rank, but it's pattern has Boolean rank 1.

Unfortunately, the Boolean rank does not help us with effectively estimating the subtropical rank, as its computation is an NP-hard problem. The standard rank is (relatively) easy to compute, but the standard rank and the max-times rank are incommensurable, that is, there are matrices that have smaller max-times rank than standard rank and others that have higher max-times rank than standard rank. Let us consider an example of the first kind,

$$\begin{pmatrix} 1 & 2 & 0 \\ 2 & 4 & 1 \\ 0 & 4 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 2 & 1 \\ 0 & 2 \end{pmatrix} \boxtimes \begin{pmatrix} 1 & 2 & 0 \\ 0 & 2 & 1 \end{pmatrix} \ .$$

As the decomposition shows, this matrix has max-times rank of 2, while its normal rank is easily verified to be 3. Indeed, it is easy to see that the complement of the n-by-n identity matrix $\bar{\boldsymbol{I}}_n$, that is, the matrix that has 0s at the diagonal and 1s everywhere else, has max-times rank of $O(\log n)$ while its standard rank is n (the result follows from similar results regarding the Boolean rank, see, e.g. Miettinen, 2009).

As we have discussed earlier, max-plus and max-times algebras are isomorphic, and consequently for any matrix $\boldsymbol{A} \in \mathbb{R}_+^{n\times m}$ its max-times rank agrees with the max-plus rank of the matrix $\log(\boldsymbol{A})$. Yet, the errors obtained in approximate decompositions do not have to (and usually will not) agree. In what follows we characterize the relationship between max-plus and max-times errors. We denote by $\overline{\mathbb{R}}$ the extended real line $\mathbb{R} \cup \{-\infty\}$.

Theorem 6. *Let $\boldsymbol{A} \in \overline{\mathbb{R}}^{n\times m}$, $\boldsymbol{B} \in \overline{\mathbb{R}}^{n\times k}$ and $\boldsymbol{C} \in \overline{\mathbb{R}}^{k\times m}$. Let $M = \exp\{N\}$, where*

$$N = \max_{\substack{i\in[n] \\ j\in[m]}} \left\{ \max\{\boldsymbol{A}_{ij}, \max_{1\le d\le k}\{\boldsymbol{B}_{id} + \boldsymbol{C}_{dj}\}\} \right\} \ .$$

If an error can be bounded in max-plus algebra as

$$\|\boldsymbol{A} - \boldsymbol{B} \diamond \boldsymbol{C}\|_F^2 \leq \lambda \;, \tag{12}$$

then the following estimate holds with respect to the max-times algebra:

$$\|\exp\{\boldsymbol{A}\} - \exp\{\boldsymbol{B}\} \boxtimes \exp\{\boldsymbol{C}\}\|_F^2 \leq M^2\lambda \;. \tag{13}$$

Proof. Let $\alpha_{ij} = \max_{d=1}^{k}\{\boldsymbol{B}_{id} + \boldsymbol{C}_{dj}\}$. From (12) it follows that there exists a set of numbers $\{\lambda_{ij} \geq 0 : i \in [n], j \in [m]\}$ such that for any i, j we have $(A_{ij} - \alpha_{ij})^2 \leq \lambda_{ij}$ and $\sum_{ij} \lambda_{ij} = \lambda$. By the mean-value theorem, for every i and j we obtain

$$|\exp\{\boldsymbol{A}_{ij}\} - \exp\{\alpha_{ij}\}| = |\boldsymbol{A}_{ij} - \alpha_{ij}| \exp\{\alpha_{ij}^*\} \leq \sqrt{\lambda_{ij}} \exp\{\alpha_{ij}^*\} \;,$$

for some $\min\{\boldsymbol{A}_{ij}, \alpha_{ij}\} \leq \alpha_{ij}^* \leq \max\{\boldsymbol{A}_{ij}, \alpha_{ij}\}$. Hence,

$$(\exp\{\boldsymbol{A}_{ij}\} - \exp\{\alpha_{ij}\})^2 \leq \lambda_{ij}(\exp\{\max\{\boldsymbol{A}_{ij}, \alpha_{ij}\}\})^2 \;.$$

The estimate for the max-times error now follows from the monotonicity of the exponent:

$$\begin{aligned}\|\exp\{\boldsymbol{A}\} - \exp\{\boldsymbol{B}\} \boxtimes \exp\{\boldsymbol{C}\}\|_F^2 &\leq \sum_{ij} \left(\exp\{\alpha_{ij}^*\}\right)^2 \lambda_{ij} \\ &\leq \sum_{ij} \left(\exp\{\max\{\boldsymbol{A}_{ij}, \alpha_{ij}\}\}\right)^2 \lambda_{ij} \leq M^2\lambda \;,\end{aligned}$$

proving the claim. □

3.4 Different subtropical matrix ranks

The definition of the subtropical rank we use in this work is the so-called Schein (or Barvinok) rank (see Definition 5). Like in the standard linear algebra, this is not the only possible way to define the (subtropical) rank. Here we will review few other forms of subtropical rank that can allow us to bound the Schein/Barvinok rank of a matrix. Following the literature, we will present the definitions in this section over the tropical algebra. Recall that due to isomorphism, these definitions transfer directly to the subtropical case. Unless otherwise mentioned, the definitions are by Guillon et al. (2015); we refer the readers interested in more details to their work.

We begin with the tropical equivalent of the subtropical Schein/Barvinok rank:

Definition 8. The *tropical Schein/Barvinok rank* of a matrix $\boldsymbol{A} \in \overline{\mathbb{R}}^{n\times m}$, denoted $\mathrm{rank}_{\mathrm{S/B}}(\boldsymbol{A})$, is defined to be the least integer k such that there exist matrices $\boldsymbol{B} \in \overline{\mathbb{R}}^{n\times k}$ and $\boldsymbol{C} \in \overline{\mathbb{R}}^{k\times m}$ for which $\boldsymbol{A} = \boldsymbol{B} \diamond \boldsymbol{C}$.

Analogous to the standard case, we can also define the rank as the number of linearly independent rows or columns. The following definition of linear independence of a family of vectors in a tropical space is due to Gondran and Minoux (1984b).

Definition 9. A set of vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k$ from $\overline{\mathbb{R}}^n$ is called *linearly dependent* if there exist disjoint sets $I, J \subset \{1, \ldots, k\}$ and scalars $\{\lambda_i\}_{i \in I \cup J}$, such that $\lambda_i \neq -\infty$ for all i and

$$\max_{i \in I}\{\lambda_i + \boldsymbol{x}_i\} = \max_{j \in J}\{\lambda_j + \boldsymbol{x}_j\} \; . \tag{14}$$

Otherwise the vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k$ are called *linearly independent*.

This gives rise to the so-called *Gondran–Minoux ranks*:

Definition 10. The *Gondran–Minoux* row (column) rank of a matrix $\boldsymbol{A} \in \overline{\mathbb{R}}^{n \times m}$ is defined as the maximal k such that $\boldsymbol{A}$ has k independent rows (columns). They are denoted by $\operatorname{rank}_{\mathrm{G–M;rw}}(\boldsymbol{A})$ and $\operatorname{rank}_{\mathrm{G–M;cl}}(\boldsymbol{A})$ respectively.

Another way to characterize the rank of the matrix is to consider the space its rows or columns can span.

Definition 11. A set $X \subset \overline{\mathbb{R}}^n$ is called *tropically convex* if for any vectors $\boldsymbol{x}, \boldsymbol{y} \in X$ and scalars $\lambda, \mu \in \overline{\mathbb{R}}$, we have $\max\{\lambda + \boldsymbol{x}, \mu + \boldsymbol{y}\} \in X$.

Definition 12. The *convex hull* $H(\boldsymbol{x}_1, \ldots \boldsymbol{x}_k)$ of a finite set of vectors $\{\boldsymbol{x}_i\}_{i=1}^k \in \overline{\mathbb{R}}^n$ is defined as follows

$$H(\boldsymbol{x}_1, \ldots \boldsymbol{x}_k) = \left\{ \max_{i=1}^{k}\{\lambda_i + \boldsymbol{x}_i\} \; : \; \lambda_i \in \overline{\mathbb{R}} \right\} \; .$$

Definition 13. The *weak dimension* of a finitely generated tropically convex subset of $\overline{\mathbb{R}}^n$ is the cardinality of its minimal generating set.

We can define the rank of the matrix by looking at the weak dimension of the (tropically) convex hull its rows or columns span.

Definition 14. The *row rank* and the *column rank* of a matrix $\boldsymbol{A} \in \overline{\mathbb{R}}^{n \times m}$ are defined as the weak dimensions of the convex hulls of the rows and the columns of $\boldsymbol{A}$ respectively. They are denoted by $\operatorname{rank}_{\mathrm{rw}}(\boldsymbol{A})$ and $\operatorname{rank}_{\mathrm{cl}}(\boldsymbol{A})$.

None of the above definitions coincide (see Akian et al., 2009), unlike in the standard algebra. We can, however, have a partial ordering of the ranks:

Theorem 7. *(Guillon et al., 2015; Akian et al., 2009) Let $\boldsymbol{A} \in \overline{\mathbb{R}}^{n \times m}$. Then the the following relations are true for the above definitions of the rank of $\boldsymbol{A}$:*

$$\left.\begin{matrix} \operatorname{rank}_{G–M;rw}(\boldsymbol{A}) \\ \operatorname{rank}_{G–M;cl}(\boldsymbol{A}) \end{matrix}\right\} \leq \operatorname{rank}_{S/B}(\boldsymbol{A}) \leq \left\{\begin{matrix} \operatorname{rank}_{rw}(\boldsymbol{A}) \\ \operatorname{rank}_{cl}(\boldsymbol{A}) \end{matrix}\right. \; . \tag{15}$$

The row and column ranks of an n-by-n tropical matrix can be computed in $O(n^3)$ time (Butkovič, 2010), allowing us to bound the Schein/Barvinok rank from above. Unfortunately, no efficient algorithm for the Gondran–Minoux rank is known. On the other hand, Guillon et al. (2015) presented what they called the *ultimate tropical rank* that lower-bounds the Gondran–Minoux rank and can be computed in time $O(n^3)$. We can also check if a matrix has full Schein/Barvinok rank in time $O(n^3)$ (see Butkovič and Hevery, 1985), even if computing any other value is NP-hard.

These bounds, together with Lemma 5 yield the following corollary regarding the bounding of the *Boolean rank* of a square matrix:

Corollary 8. *Given an n-by-n binary matrix $\boldsymbol{A}$, it's Boolean rank can be bound from below, using the ultimate rank, and from above, using the tropical column and row ranks, in time $O(n^3)$.*

4 Algorithms

The problem of subtropical matrix factorization has some unique challenges that stem from the lack of linearity and smoothness of the max-times algebra. One of such issues is that dominated elements in a decomposition have no impact on the final result. Namely, if we consider the subtropical product of two matrices $\boldsymbol{B} \in \mathbb{R}_+^{n\times k}$ and $\boldsymbol{C} \in \mathbb{R}_+^{k\times m}$, we can see that each entry $(\boldsymbol{B} \boxtimes \boldsymbol{C})_{ij} = \max_{1\le s\le k} \boldsymbol{B}_{is}\boldsymbol{C}_{sj}$ is completely determined by a single element with index $\arg\max_{1\le s\le k} \boldsymbol{B}_{is}\boldsymbol{C}_{sj}$. This means that all entries t with $\boldsymbol{B}_{it}\boldsymbol{C}_{tj} < \max_{1\le s\le k} \boldsymbol{B}_{is}\boldsymbol{C}_{sj}$ do not contribute at all to the final decomposition. To see why this is a problem, observe that many optimization methods used in matrix factorization algorithms rely on local information to choose the direction of the next step (e.g. various forms of gradient descent). In the case of the subtropical algebra, however, the local information is practically absent, and hence we need to look elsewhere for effective optimization techniques.

A common approach to matrix decomposition problems is to update factor matrices alternatingly, which utilizes the fact that the problem $\min_{\boldsymbol{B},\boldsymbol{C}} \|\boldsymbol{A} - \boldsymbol{B}\boldsymbol{C}\|_F$ is biconvex. Unfortunately, the subtropical matrix factorization problem does not have the biconvexity property, which makes alternating updates less useful.

Here we present a different approach that, instead of doing alternating factor updates, constructs the decomposition by adding one rank-1 matrix at a time, following the idea by Kolda and O'Leary (2000). The corresponding algorithm is called `Equator` (Algorithm 1).

First observe that the max-times product can be represented as an elementwise maximum of rank-1 matrices (blocks)

$$\boldsymbol{B} \boxtimes \boldsymbol{C} = \max_{1\le s\le k} \boldsymbol{B}^s\boldsymbol{C}_s \ . \tag{16}$$

Hence, Problem 1 can be split into k subproblems of the following form: given a rank-$(l-1)$ decomposition $\boldsymbol{B} \in \mathbb{R}_+^{n\times(l-1)}$, $\boldsymbol{C} \in \mathbb{R}_+^{(l-1)\times m}$ of a matrix $\boldsymbol{A} \in \mathbb{R}_+^{n\times m}$, find a column vector $\boldsymbol{b} \in \mathbb{R}_+^{n\times 1}$ and a row vector $\boldsymbol{c} \in \mathbb{R}_+^{1\times m}$ such that the error

$$\|\boldsymbol{A} - \max\{\boldsymbol{B} \boxtimes \boldsymbol{C}, \boldsymbol{b}\boldsymbol{c}\}\| \tag{17}$$

is minimized. We assume by definition that the rank-0 decomposition is an all zero matrix of the same size as $\boldsymbol{A}$. The problem of rank-k subtropical matrix factorization is then reduced to solving (17) k times. One should of course remember that this scheme is just a heuristic and finding optimal blocks on each iteration does not guarantee converging to a global minimum.

One prominent issue with the above approach is that an optimal rank-$(k-1)$ decomposition might not be very good when considered as a part of a rank-k decomposition. This is because for smaller ranks we generally have to cover the data more crudely, whereas when the rank increases we can afford to use smaller and more refined blocks. In order to deal with this problem, we find and then update the blocks repeatedly, in a cyclic fashion. That means that after discovering the last block, we go all the way back to block one. The input parameter M defines the number of full cycles we make.

On a high level `Equator` works as folows. First the factor matrices are initialized to all zeros (line 2). Since the algorithm makes iterative changes to the current solutions that might in some cases lead to worsening of the results, it

Algorithm 1 `Equator`

```
Input: $\boldsymbol{A} \in \mathbb{R}_+^{n\times m}$, $k > 0$, $M > 0$
Output: $\boldsymbol{B}^* \in \mathbb{R}_+^{n\times k}$, $\boldsymbol{C}^* \in \mathbb{R}_+^{k\times m}$
 1: function Equator($\boldsymbol{A}, k, M$)
 2:     $\boldsymbol{B} \leftarrow 0^{n\times k}$, $\boldsymbol{C} \leftarrow 0^{k\times m}$
 3:     $\boldsymbol{B}^* \leftarrow \boldsymbol{B}, \boldsymbol{C}^* \leftarrow \boldsymbol{C}$
 4:     $bestError \leftarrow E(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C})$
 5:     for $count \leftarrow 1$ to $k \times M$ do
 6:         $l \leftarrow (count - 1) \pmod{k} + 1$          ▷ Index of the current block
 7:         $[\boldsymbol{B}^l, \boldsymbol{C}_l] \leftarrow$ UpdateBlock($\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, count$)
 8:         if $E(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}) < bestError$ then
 9:             $\boldsymbol{B}^* \leftarrow \boldsymbol{B}, \boldsymbol{C}^* \leftarrow \boldsymbol{C}$
10:             $bestError \leftarrow E(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C})$
11:     return $\boldsymbol{B}^*$, $\boldsymbol{C}^*$
```

also stores the best reconstruction error and the corresponding factors found so far. They are initalized with the starting solution on lines 3–4. The main work is done in the loop on lines 5–10, where on each iteration we update a single rank-1 matrix in the current decomposition using the `UpdateBlock` routine (line 7), and then check if the update improves the best result (lines 8–10).

We will present two versions of the `UpdateBlock` function, one called `Capricorn` and the other one `Cancer`. `Capricorn` is designed to work with discrete (or flipping) noise, when some of the elements in the data are randomly changed to different values. In this setting the level of noise is the proportion of the flipped elements relative to the total number of nonzeros. `Cancer` on the other hand is robust with continuous noise, when many elements are affected (e.g. Gaussian noise). We will discuss both of them in detail in the following subsections. In the rest of the paper, especially when presenting the experiments, we will use names `Capricorn` and `Cancer` not only for a specific variation of the `UpdateBlock` function, but also for the `Equator` algorithm that uses it.

4.1 `Capricorn`

We first describe `Capricorn`, which is designed to solve the subtropical matrix factorization problem in the presence of discrete noise, and minimizes the $L1$ norm of the error matrix. The main idea behind the algorithm is to spot potential blocks by considering ratios of matrix rows. Consider an arbitrary rank-1 block $\boldsymbol{X} = \boldsymbol{b}\boldsymbol{c}$, where $\boldsymbol{b} \in \mathbb{R}_+^{n\times 1}$ and $\boldsymbol{c} \in \mathbb{R}_+^{1\times m}$. For any indices i and j such that $\boldsymbol{b}_i > 0$ and $\boldsymbol{b}_j > 0$, we have $\boldsymbol{X}_j = \frac{\boldsymbol{b}_j}{\boldsymbol{b}_i}\boldsymbol{X}_i$. This is a characteristic property of rank-1 matrices – all rows are multiples of one another. Hence, if a block $\boldsymbol{X}$ dominates some region Γ of a matrix $\boldsymbol{A}$, then rows of $\boldsymbol{A}$ should all be multiples of each other within Γ. These rows might have different lengths due to block overlap, in which case the rule only applies to their common part.

`UpdateBlock` starts by identifying the index of the block that has to be updated at the current iteration (line 2). In order to find the best new block we need to take into account that some parts of the data have already been covered, and we must ignore them. This is accomplished by replacing the original matrix with a residual $\boldsymbol{R}$ that represents what there is left to cover. The building of

Algorithm 2 `UpdateBlock` (`Capricorn`)

```
Input: $\boldsymbol{A} \in \mathbb{R}_+^{n\times m}$, $\boldsymbol{B} \in \mathbb{R}_+^{n\times k}$, $\boldsymbol{C} \in \mathbb{R}_+^{k\times m}$, $count > 0$
Output: $\boldsymbol{b} \in \mathbb{R}_+^{n\times 1}$, $\boldsymbol{c} \in \mathbb{R}_+^{1\times m}$
Parameters: $bucketSize > 0$, $\delta > 0$, $\theta > 0$, $\tau \in [0,1]$
 1: function UpdateBlock($\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, count$)
 2:     $l \leftarrow (count - 1) \pmod{k} + 1$                    ▷ Index of the current block
 3:     $\boldsymbol{R}_{ij} \leftarrow \begin{cases} \boldsymbol{A}_{ij} & (\boldsymbol{B}^{-l} \boxtimes \boldsymbol{C}_{-l})_{ij} < \boldsymbol{A}_{ij} \\ NaN & \text{otherwise} \end{cases}$          ▷ Residual matrix
 4:     $idx \leftarrow \arg\max_i \sum_j r_{ij}$
 5:     $\boldsymbol{H} \leftarrow$ CorrelationsWithRow($\boldsymbol{R}, idx, bucketSize, \delta, \tau$)
 6:     $r \leftarrow \arg\max_i \sum_j h_{ij}$
 7:     $c \leftarrow \arg\max_j \sum_i h_{ij}$
 8:     $b\_idx \leftarrow \{i : \boldsymbol{H}_{ic} = 1\}$
 9:     $c\_idx \leftarrow \{i : \boldsymbol{H}_{ri} = 1\}$
10:     $[\boldsymbol{b}, \boldsymbol{c}] \leftarrow$ RecoverBlock($\boldsymbol{R}, b\_idx, c\_idx$)
11:     $\boldsymbol{b} \leftarrow$ AddRows($\boldsymbol{b}, \boldsymbol{c}, \boldsymbol{A}, \theta, bucketSize, \delta$)
12:     $\boldsymbol{c} \leftarrow$ AddRows($\boldsymbol{c}^T, \boldsymbol{b}^T, \boldsymbol{A}^T, \theta, bucketSize, \delta)^T$
13:     return $\boldsymbol{b}$, $\boldsymbol{c}$
```

the residual (line 3) reflects the winner-takes-it-all property of the max-times algebra: if an element of $\boldsymbol{A}$ is approximated by a smaller value, it appears as such in the residual; if it is approximated by a value that is at least as large, then the corresponding residual element is NaN, indicating that this value is already covered. We then select a seed row (line 4), with an intention of growing a block around it. We choose the row with the largest sum as this increases the chances of finding the most prominent block. In order to find the best block $\boldsymbol{X}$ that the seed row passes through, we first find a binary matrix $\boldsymbol{H}$ that represents the pattern of $\boldsymbol{X}$ (line 5). Next, on lines 6–9 we choose an approximation of the block pattern with index sets b_idx and c_idx, which define what elements of $\boldsymbol{b}$ and $\boldsymbol{c}$ should be nonzero. The next step is to find the actual values of elements within the block with the function `RecoverBlock` (line 10). Finally, we inflate the found core block with `ExpandBlock` (line 11).

The function `CorrelationsWithRow` (Algorithm 3) finds the pattern of a new block. It does so by comparing a given seed row to other rows of the matrix and extracting sets where the ratio of the rows is almost constant. As was mentioned before, if two rows locally represent the same block, then one should be a multiple of the other, and the ratios of their corresponding elements should remain level. `CorrelationsWithRow` processes the input matrix row by row using the function `FindRowSet`, which for every row outputs the most likely set of indices, where it is correlated with the seed row (lines 4–6). Since the seed row is obviously the most correlated with itself, we compensate for this by replacing its pattern with that of the second most correlated row (lines 7–8). Finally, we drop some of the least correlated rows after comparing their correlation value ϕ to that of the second most correlated row (after the seed row). The correlation function ϕ is defined as follows

$$\phi(\boldsymbol{H}, idx, i) = \frac{\langle \boldsymbol{H}_i, \boldsymbol{H}_{idx} \rangle}{\langle \boldsymbol{H}_i, \boldsymbol{H}_i \rangle + 1} \ . \tag{18}$$

The parameter τ is a threshold determining whether a row should be discarded or retained. The auxiliary function `FindRowSet` (Algorithm 4) compares two

Algorithm 3 `CorrelationsWithRow`

```
Input: $\boldsymbol{R} \in \mathbb{R}_+^{n\times m}$, $idx \in [n]$, $bucketSize > 0$, $\delta > 0$, $\tau \in [0,1]$
Output: $\boldsymbol{H} \in \{0, 1\}^{n\times m}$
 1: function CorrelationsWithRow($\boldsymbol{R}$, $idx$, $bucketSize$, $\delta$, $\tau$)
 2:     turn all $NaN$ elements of $\boldsymbol{R}$ to 0
 3:     $\boldsymbol{H} \leftarrow 0^{n\times m}$
 4:     for $i \leftarrow 1$ to $n$ do
 5:         $V_i \leftarrow$ FindRowSet($\boldsymbol{R}_{idx}$, $\boldsymbol{R}_i$, $bucketSize$, $\delta$)
 6:         $\boldsymbol{H}(i, V_i) \leftarrow 1$
 7:     $s \leftarrow \arg\max_{i\,:\,i\neq idx} \sum_j h_{ij}$
 8:     $\boldsymbol{H}_{idx} \leftarrow \boldsymbol{H}_s$
 9:     for $i \leftarrow 1$ to $n$ do
10:         if $\phi(\boldsymbol{H}, idx, i) < \phi(\boldsymbol{H}, idx, s) - \tau$ then
11:             $\boldsymbol{H}_i \leftarrow 0$
12:     return $\boldsymbol{H}$
```

Algorithm 4 `FindRowSet`

```
Input: $\boldsymbol{u} \in \mathbb{R}_+^m, \boldsymbol{v} \in \mathbb{R}_+^m, bucketSize > 0, \delta > 0$
Output: $V \subset [m]$
 1: function FindRowSet($\boldsymbol{u}$, $\boldsymbol{v}$, $bucketSize$, $\delta$)
 2:     $\boldsymbol{r} \leftarrow \log(\boldsymbol{u} \,./\, \boldsymbol{v})$
 3:     $nBuckets \leftarrow \lceil(\max\{r\} - \min\{r\})/\delta\rceil$
 4:     for $i \leftarrow 0$ to $nBuckets$ do
 5:         $V_i \leftarrow \{idx \in [m] \,:\, \min\{\boldsymbol{r}\} + i\delta \leq r_{idx} < \min\{\boldsymbol{r}\} + (i+1)\delta\}$
 6:     $V \leftarrow \arg\max\{|V_i| \,:\, i = 1, \ldots, nBuckets\}$
 7:     if $|V| < bucketSize$ then
 8:         $V \leftarrow \emptyset$
 9:     return $V$
```

vectors and finds the biggest set of indices where their ratio remains almost constant. It does so by sorting the log-ratio of the input vectors into buckets of a fixed size and then choosing the bucket with the most elements. The notation $\boldsymbol{u} \,./\, \boldsymbol{v}$ on line 2 means elementwise ratio of vectors $\boldsymbol{u}$ and $\boldsymbol{v}$.

It accepts two additional parameters: *bucketSize* and δ. If the largest bucket has fewer than *bucketSize* elements, the function will return an empty set – this is done because very small patterns do not reveal much structure and are mostly accidental. The width of the buckets is determined by the parameter δ.

At this point we know the pattern of the new block, that is, the locations of its non-zeros. To fill in the actual values, we consider the submatrix defined by the pattern, and find the best rank-1 approximation of it. We do this using the `RecoverBlock` function (Algorithm 5). It begins by setting all elements outside of the pattern to 0 as they are irrelevant to the block (line 2). Then it chooses one row to represent the block (lines 3–4), which will be used to find a good rank-1 cover.

Finally, we find the optimal column vector for the block by computing the best weights to be used for covering different rows of the block with its representing row (line 5). Here we optimize with respect to the Frobenius norm, rather than L_1 matrix norm, since it allows to solve the optimization problem in closed form.

Since blocks often heavily overlap, we are susceptible to finding only fragments of patterns in the data – some parts of a block can be dominated by another

Algorithm 5 `RecoverBlock`

```
Input: $\boldsymbol{R} \in \mathbb{R}_+^{n\times m}$, $bIdx \subset [n]$, $cIdx \subset [m]$
Output: $\boldsymbol{b} \in \mathbb{R}_+^{n\times 1}$, $\boldsymbol{c} \in \mathbb{R}_+^{1\times m}$
1: function RecoverBlock($\boldsymbol{R}$, $bIdx$, $cIdx$)
2:     turn $\boldsymbol{R}$ to 0 except elements with indices $(bIdx, cIdx)$
3:     $p \leftarrow$ RowRepresentingBlock($\boldsymbol{R}$, $bIdx$)
4:     $\boldsymbol{c} \leftarrow \boldsymbol{R}_p$
5:     $\boldsymbol{b} \leftarrow \arg\min_{\boldsymbol{t}\in\mathbb{R}_+^{n\times 1}} \|\boldsymbol{R} - \boldsymbol{tc}\|_F$
6:     return $\boldsymbol{b}$, $\boldsymbol{c}$
```

Algorithm 6 `AddRows`

```
Input: $\boldsymbol{b} \in \mathbb{R}_+^{n\times 1}$, $\boldsymbol{c} \in \mathbb{R}_+^{1\times m}$, $\boldsymbol{A} \in \mathbb{R}_+^{n\times m}$, $\theta > 0$, $bucketSize > 0$, $\delta > 0$
Output: $\boldsymbol{b} \in \mathbb{R}_+^{n\times 1}$
1: function AddRows($\boldsymbol{b}, \boldsymbol{c}, \boldsymbol{A}, \theta, bucketSize, \delta$)
2:     $b\_idx \leftarrow \{t : \boldsymbol{b}_t > 0\}$
3:     for $i \in [n] \setminus b\_idx$ do
4:         $V_i \leftarrow$ FindRowSet($\boldsymbol{c}, \boldsymbol{R}_i, bucketSize, \delta$)
5:         if $V_i = \emptyset$ then
6:             continue
7:         $\alpha \leftarrow mean(\boldsymbol{R}_{iV_i}./\boldsymbol{c}_{V_i})$
8:         $impact \leftarrow \frac{\sum_{s\in V_i}\max\{0,\alpha c_s - \boldsymbol{A}_{is}\}}{\sum_{s\in V_i}\boldsymbol{A}_{is} - |\boldsymbol{A}_{is} - \alpha c_s|}$
9:         if $impact \le \theta$ then
10:            $\boldsymbol{b}_i \leftarrow \alpha$
11:    return $\boldsymbol{b}$
```

block and subsequently not recognized. Hence, we need to expand found blocks to make them complete. This is done separately for rows and columns in the method called `AddRows` (Algorithm 6), which, given a starting block $\boldsymbol{X} = \boldsymbol{bc}$ and the original matrix $\boldsymbol{A}$, tries to add new nonzero elements to $\boldsymbol{b}$. It iterates through all rows of $\boldsymbol{A}$ and adds those that would make a positive impact on the objective without unnecessarily overcovering the data. In order to decide whether a given row should be added, it first extracts a set V_i of indices where this row is a multiple of the row vector $\boldsymbol{c}$ of the block (if they are not sufficiently correlated, then the row does not belong to the block) (line 4). A row is added if the evaluation of the following function (line 8)

$$\psi(\alpha) = \frac{\sum_{s\in V_i}\max\{0,\, \alpha c_s - \boldsymbol{A}_{is}\}}{\sum_{s\in V_i}\boldsymbol{A}_{is} - |\boldsymbol{A}_{is} - \alpha c_s|} \tag{19}$$

is below the threshold θ. In (19) the numerator measures by how much the new row would overcover the original matrix, and the denominator reflects the improvement in the objective compared to a zero row.

Parameters. `Capricorn` has four parameters in addition to the common parameters in the Equator framework: $bucketSize > 0$, $\delta > 0$, $\theta > 0$, and $\tau \in [0, 1]$. The first one, $bucketSize$ determines the minimum number of elements in two rows that must have "approximately" the same ratio for them to be considered for building a block. The parameter δ defines the bucket width when computing row correlations. When expanding a block, θ is used to decide whether to add a row (or column) to it – the decision is positive whenever the expression (19) is at most θ. Finally τ is used during the discovery of correlated rows. The value of τ belongs to the closed unit interval, and the higher it is, the more rows will be added.

4.2 Cancer

We now present our second algorithm, `Cancer`, which is a counterpart of `Capricorn` specifically designed to work in the presence of high levels of continuous noise. The reason why `Capricorn` cannot deal with continuous noise is that it expects the rows in a block to have an "almost" constant elementwise ratio, which is not the case when too many entries in the data are disturbed. For example, even low levels of Gaussian noise would make the ratios vary enough to hinder `Capricorn`'s ability to spot blocks. With `Cancer` we take a new approach which is based on polynomial approximation of the objective. We also replace the L_1 matrix norm, which was used as an objective for `Capricorn`, with the Frobenius norm. The reason for that is that when the noise is continuous, its level is defined as the total deviation of the noisy data from the original, rather than a count of the altered elements. This makes the Frobenius norm a good estimator for the amount of noise. `Cancer` conforms to the general framework of `Equator` (Algorithm 1), and differs from `Capricorn` only in how it finds the blocks and in the objective function.

Observe that in order to solve the problem (17) we need to find a column vector $\boldsymbol{b} \in \mathbb{R}_+^{n\times 1}$ and a row vector $\boldsymbol{c} \in \mathbb{R}_+^{1\times m}$ such that they provide the best rank-1 approximation of the input matrix given the current factorization. The objective function is not convex in either $\boldsymbol{b}$ or $\boldsymbol{c}$ and is generally hard to optimize directly, so we have to simplify the problem, which we do in two steps. First, instead of doing full optimization of $\boldsymbol{b}$ and $\boldsymbol{c}$ simultaneously, we update only a single element of one of them at a time. This way the problem is reduced to single variable optimization. Even then the objective is hard to minimize, and we replace it with a polynomial approximation, which is easy to optimize directly.

The `Cancer` version of the `UpdateBlock` function is described in Algorithm 7. It alternatingly updates the vectors $\boldsymbol{b}$ and $\boldsymbol{c}$ using the `AdjustOneElement` routine. Both $\boldsymbol{b}$ and $\boldsymbol{c}$ will be updated $\lfloor f(n+m)/2 \rfloor$ times. `UpdateBlock` starts by finding the index of the block that has to be changed (line 2). Since the purpose of `UpdateBlock` is to find the best rank-1 matrix to replace the current block, we also need to compute the reconstructed matrix without it, which is done on line 3. We then find the number of times `AdjustOneElement` will be called (line 4) and change the degree of polynomials used for objective function approximation (line 5). This is needed because high degree polynomials are better at finalizing a solution that is already reasonably good, but tend to overfit the data and cause the algorithm to get stuck in local minima at the beginning. It is therefore beneficial to start with polynomials of lower degrees and then gradually increase it. The actual changes to $\boldsymbol{b}$ and $\boldsymbol{c}$ happen in the loop (lines 7–9), where we update them using `AdjustOneElement`.

The `AdjustOneElement` function (Algorithm 8) updates a single entry in either a column vector $\boldsymbol{b}$ or a row vector $\boldsymbol{c}$. Let us consider the case when $\boldsymbol{b}$ is fixed and $\boldsymbol{c}$ varies. In order to decide which element of $\boldsymbol{c}$ to change, we need to compare the best changes to all m entries and then choose the one that yields the most improvement to the objective. A single element $\boldsymbol{c}_l$ only has an effect on the error along the column l. Assume that we are currently updating block with

Algorithm 7 `UpdateBlock` (`Cancer`)

```
Input: $\boldsymbol{A} \in \mathbb{R}_+^{n\times m}$, $\boldsymbol{B} \in \mathbb{R}_+^{n\times k}$, $\boldsymbol{C} \in \mathbb{R}_+^{k\times m}$, $count > 0$
Output: $\boldsymbol{b} \in \mathbb{R}_+^{n\times 1}$, $\boldsymbol{c} \in \mathbb{R}_+^{1\times m}$
Parameters: $t > 2$, $0 < f < 1$
 1: function UpdateBlock($\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, count$)
 2:     $l \leftarrow (count - 1) \pmod{k} + 1$                ▷ Index of the current block
 3:     $\boldsymbol{N} \leftarrow \boldsymbol{B}^{-l} \boxtimes \boldsymbol{C}_{-l}$        ▷ Reconstructed matrix without the $i$-th block
 4:     $niters \leftarrow \lfloor f(n+m)/2 \rfloor$
 5:     $deg \leftarrow 2 + \lfloor (count - 1)/k \rfloor \pmod{t}$
 6:     $\boldsymbol{b} \leftarrow \boldsymbol{B}^l$, $\boldsymbol{c} \leftarrow \boldsymbol{C}_l$
 7:     for $iter \leftarrow 1$ to $niters$ do
 8:         $\boldsymbol{c} =$ AdjustOneElement($\boldsymbol{A}, \boldsymbol{N}, \boldsymbol{b}, \boldsymbol{c}, deg$)
 9:         $\boldsymbol{b} =$ AdjustOneElement($\boldsymbol{A}^T, \boldsymbol{N}^T, \boldsymbol{c}^T, \boldsymbol{b}^T, deg)^T$
10:     return $\boldsymbol{b}$, $\boldsymbol{c}$
```

Algorithm 8 `AdjustOneElement`

```
Input: $\boldsymbol{A} \in \mathbb{R}_+^{n\times m}$, $\boldsymbol{N} \in \mathbb{R}_+^{n\times m}$, $\boldsymbol{b} \in \mathbb{R}_+^{n\times 1}$, $\boldsymbol{c} \in \mathbb{R}_+^{1\times m}$, $deg \geq 2$
Output: $\boldsymbol{c} \in \mathbb{R}_+^{1\times m}$
1: function AdjustOneElement($\boldsymbol{A}, \boldsymbol{N}, \boldsymbol{b}, \boldsymbol{c}, deg$)
2:     for $j \leftarrow 1$ to $m$ do
3:         $baseError \leftarrow \sum_{i=1}^{n} (\boldsymbol{A}_{ij} - \max\{\boldsymbol{N}_{ij}, \boldsymbol{b}_i\boldsymbol{c}_j\})^2$
4:         $[err, \boldsymbol{x}_i] \leftarrow$ PolyMin($\boldsymbol{A}^j, \boldsymbol{N}^j, \boldsymbol{b}, deg$)
5:         $\boldsymbol{u}_i \leftarrow baseError - err$
6:     $i \leftarrow$ the index $i$ of largest value of $\boldsymbol{u}$
7:     $\boldsymbol{c}_i \leftarrow \boldsymbol{x}_i$
8:     return $\boldsymbol{c}$
```

index q and let $\boldsymbol{N}$ denote the reconstruction matrix without this block, that is $\boldsymbol{N} = \boldsymbol{B}^{-q} \boxtimes \boldsymbol{C}_{-q}$. Minimizing $E(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C})$ with respect to $\boldsymbol{c}_l$ is then equivalent to minimizing

$$\gamma(\boldsymbol{A}_l, \boldsymbol{N}_l, \boldsymbol{b}, \boldsymbol{c}_l) = \sum_{i=1}^{n} (\boldsymbol{A}_{il} - \max\{\boldsymbol{N}_{il}, \boldsymbol{b}_i\boldsymbol{c}_l\})^2 \ . \tag{20}$$

Instead of minimizing (20) directly, we use polynomial approximation in the `PolyMin` routine (line 4). It returns the (approximate) error *err* and the value x achieving that. Since we are only interested in the improvement of the objective achieved by updating a single entry of $\boldsymbol{c}$, we compute the improvement of the objective after the change (line 5). After trying every column of $\boldsymbol{c}$, we update only the column that yield the largest improvement.

The function γ that we need to minimize in order to find the best change to the vector $\boldsymbol{c}$ in `AdjustOneElement` is hard to work with directly since it is not convex, and also not smooth because of the presence of the maximum operator. To alleviate this, we approximate the error function γ with a polynomial g of degree *deg*. Notice that when updating $\boldsymbol{c}_l$, other variables of γ are fixed and we only need to consider function $\gamma'(x) = \gamma(\boldsymbol{A}_l, \boldsymbol{N}_l, \boldsymbol{b}, x)$. To build g we sample $deg + 1$ points from $(0, 1)$ and fit g to the values of γ' at these points. We then find the $x \in \mathbb{R}_+$ that minimizes $g(x)$ and return $g(x)$ (the approximate error) and x (the optimal value).

Parameters. `Cancer` has two parameters, $t > 2$ and $0 < f < 1$, that control its execution. The first one, t, is the maximum allowed degree of polynomials used for approximation of the objective, which we set to 16 in all our experiments. The second parameter, f, determines the number of single element updates we make to the row and column vectors of a block in `UpdateBlock`.

Generalized Cancer. The `Cancer` algorithm can be adapted to optimize other objective functions. Its general polynomial approximation framework allows for a wide variety of possible objectives, the only constraint being that they have to be additive (we call a function $E(\boldsymbol{A}, \boldsymbol{R})$ *additive* if there exists a mapping $\phi\colon \mathbb{R}_+ \times \mathbb{R}_+ \to \mathbb{R}_+$ such that for all $\boldsymbol{A} \in \mathbb{R}_+^{n\times m}$ and $\boldsymbol{R} \in \mathbb{R}_+^{n\times m}$ we have $E(\boldsymbol{A}, \boldsymbol{R}) = \sum_{ij} \phi(\boldsymbol{A}_{ij}, \boldsymbol{R}_{ij})$). Some examples of such functions are L_1 and Frobenius matrix norms, as well as Kullback–Leibler and Jensen–Shannon divergences. In order to use the generalized form of `Cancer` one simply has to replace the Frobenius norm with another cost function wherever the error is evaluated.

4.3 Time complexity

The main work in `Equator` is performed inside the `UpdateBlock` routine, which is called Mk times. Since M is a constant parameter, the complexity of `Equator` is k times the complexity of `UpdateBlock`. In the following we find the theoretical bounds on the execution time of `UpdateBlock` for both `Capricorn` and `Cancer`.

Capricorn. In the case of `Capricorn` there are three main contributors to `UpdateBlock` (Algorithm 2): `CorrelationsWithRow`, `RecoverBlock`, and `AddRows`. `CorrelationsWithRow` compares every row to the seed row, each time calling `FindRowSet`, which in turn has to process all m elements of both rows. This results in the total complexity of `CorrelationsWithRow` being $O(nm)$. To find the complexity of `RecoverBlock`, first observe that any "pure" block $\boldsymbol{X}$ can be represented as $\boldsymbol{X} = \boldsymbol{bc}$, where $\boldsymbol{b} \in \mathbb{R}_+^{n'\times 1}$ and $\boldsymbol{c} \in \mathbb{R}_+^{1\times m'}$ with $n' \leq n$ and $m' \leq m$. `RecoverBlock` selects $\boldsymbol{c}$ from the rows of $\boldsymbol{X}$ and then finds the corresponding column vector $\boldsymbol{b}$ that minimizes $\|\boldsymbol{X} - \boldsymbol{bc}\|_F$. In order to select the best row, we have to try each of the n' candidates, and since finding the corresponding $\boldsymbol{b}$ for each of them takes time $O(n'm')$, this gives the runtime of `RecoverBlock` as $O(n')O(n'm') = O(n^2m)$. The most computationally expensive parts of `AddRows` are `FindRowSet` (line 4), finding the mean (line 7), and computing the impact (line 8), which all run in $O(m)$ time. All of these operations have to be repeated $O(n)$ times, and hence the runtime of `AddRows` is $O(nm)$. Thus, we can now estimate the complexity of `UpdateBlock` to be $O(nm) + O(n^2m) + O(nm) = O(n^2m)$, which leads to the total runtime of `Capricorn` to be $O(n^2mk)$.

Cancer. Here `UpdateBlock` (Algorithm 7) is a loop that calls `AdjustOneElement` $\lfloor f(n+m) \rfloor$ times. In `AdjustOneElement` the contributors to the complexity are computing the base error (line 3) and a call to `PolyMin` (line 4). Both of them are performed n or m times depending on whether we supplied the column vector $\boldsymbol{b}$ or the row vector $\boldsymbol{c}$ to `AdjustOneElement`. Finding the base error takes time $O(m)$ for $\boldsymbol{b}$ and $O(n)$ for $\boldsymbol{c}$. The complexity of `PolyMin` boils down to that of evaluating the max-times objective at $deg + 1$ points and then minimizing a degree deg polynomial. Hence, `PolyMin` runs in time $O(m)$ or $O(n)$ depending on whether we are optimizing $\boldsymbol{b}$ or $\boldsymbol{c}$, and the complexity of `AdjustOneElement` is $O(nm)$.

Since `AdjustOneElement` is called $\lfloor f(n+m)/2 \rfloor$ times and f is a fixed parameter, this gives the complexity $O\big((n+m)nm\big)$ for `UpdateBlock` and $O\big((n+m)nmk\big) = O(\max\{n,m\}nmk)$ for `Cancer`.

5 Experiments

We tested both `Capricorn` and `Cancer` on synthetic and real-world data. In addition we also compare against a variation of `Cancer` that optimizes the Jensen–Shannon divergence, which we call `CancerJS`. The purpose of the synthetic experiments is to evaluate the properties of the algorithm in controlled environments where we know the data has the max-times structure. They also demonstrate on what kind of data each algorithm excels and what their limitations are. The purpose of the real-world experiments is to confirm that these observations also hold true in real-world data, and to study what kinds of data sets actually have max-times structure. The source code of `Capricorn` and `Cancer` and the scripts that run the experiments in this paper are freely available for academic use.[2]

Parameters of `Capricorn`. In both synthetic and real-world experiments we used the following default set of parameters: $M = 4$, $\mathit{bucketSize} = 3$, $\delta = 0.01$, $\theta = 0.5$, and $\tau = 0.5$.

Parameters of `Cancer`. Both variations of `Cancer` use the same set of parameters. For the synthetic experiments we used $M = 14$, $t = 16$, and $f = 0.1$. For the real world experiments we set $t = 16$, $f = 0.1$, and $M = 40$ (except for Eigenfaces, where we used $M = 50$).

5.1 Other methods.

We compared our algorithms against `SVD` and six versions of NMF. For `SVD`, we used Matlab's built-in implementation. The first NMF method, called simply `NMF`, by Kim and Park (2008), is based on the block principal pivoting algorithm. The second form of NMF is a sparse NMF algorithm by Hoyer (2004),[3] which we call `SNMF`. It defines the sparsity of a vector $\boldsymbol{x} \in \mathbb{R}_+^n$ as

$$\operatorname{sparsity}(\boldsymbol{x}) = \frac{\sqrt{n} - \left(\sum_i |\boldsymbol{x}_i|\right) / \sqrt{\sum_i \boldsymbol{x}_i^2}}{\sqrt{n} - 1}\ , \tag{21}$$

and returns factorizations where the sparsity of the factor matrices is user-controllable. In all of our experiments, we used the sparsity of `Cancer`'s factors as the sparsity parameter of `SNMF`. We also compare against a standard alternating least squares algorithm called `ALS` (Cichocki et al., 2009). Next we have two versions of NMF that are essentially the same as `ALS`, but they use L_1

[2] `http://people.mpi-inf.mpg.de/~pmiettin/tropical/`

[3] `https://github.com/aludnam/MATLAB/tree/master/nmfpack`, accessed 18 July 2017

regularization for increased sparsity (Cichocki et al., 2009), that is, they aim at minimizing

$$\|\boldsymbol{A} - \boldsymbol{B}\boldsymbol{C}\|_F + \alpha \|\boldsymbol{B}\|_1 + \beta \|\boldsymbol{C}\|_1 \ .$$

The first method is called `ALSR` and uses regularizer coefficient $\alpha = \beta = 1$, and the other, called `ALSR5`, has regularizer coefficient $\alpha = \beta = 5$. The last NMF algorithm, `WNMF` by Li and Ngom (2013), is designed to work with missing values in the data.

5.2 Synthetic experiments.

The purpose of synthetic experiments is to prove the concept, that is that our algorithms are capable of identifying the max-times structure when it is there. In order to test this, we first generate the data with the pure max-times structure, then pollute it with some level of noise, and finally run the methods. The noise-free data is created by first generating random factors of some density with nonzero elements drawn from a uniform distribution on the $[0, 1]$ interval and then multiplying them using the max-times matrix product.

We distinguish two types of noise. The first one is the discrete (or tropical) noise, which is introduced in the following way. Assume that we are given an input matrix $\boldsymbol{A}$ of size n-by-m. We first generate an n-by-m noise matrix $\boldsymbol{N}$ with elements drawn from a uniform distribution on the $[0, 1]$ interval. Given a level of noise l, we then turn $\lfloor (1 - l)nm \rfloor$ random elements of $\boldsymbol{N}$ to 0, so that its resulting density is l. Finally, the noise is applied by taking elementwise maximum between the original data and the noise matrix $\boldsymbol{F} = \max\{\boldsymbol{A}, \boldsymbol{N}\}$. This is the kind of noise that `Capricorn` was designed to handle, so we expect it to be better than `Cancer` and other comparison algorithms.

We also test against continuous noise, as it is arguably more common in the real world. For that we chose Gaussian noise with 0 mean, where the noise level is defined to be its standard deviation. Since adding this noise to the data might result in negative entries, we truncate all values in a resulting matrix that are below zero.

Unless specified otherwise, all matrices in the synthetic experiments are of size 1000-by-800 with true max-times rank 10. All results presented in this section are averaged over 10 instances. For reconstruction error tests, we compared our algorithms `Capricorn`, `Cancer`, and `CancerJS` against `SVD`, `NMF`, `SNMF`, `ALS`, `ALSR`, and `ALSR5`. The error is measured as the relative Frobenius norm $\|\tilde{\boldsymbol{A}} - \boldsymbol{A}\|_F / \|\boldsymbol{A}\|$, where $\boldsymbol{A}$ is the data and $\tilde{\boldsymbol{A}}$ its approximation, as that is the measure both `SVD` and `NMF` aim at minimizing. We also report the sparsity s of factor matrices obtained by algorithms, which is defined as a fraction of zero elements in the factor matrices,

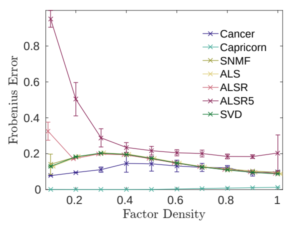
$$s(\boldsymbol{A}) = |\{(i,j) \,:\, \boldsymbol{A}_{ij} = 0\}| / (nm) \ , \tag{22}$$

for an n-by-m matrix $\boldsymbol{A}$. For the experiments with tropical noise, the reconstruction errors are reported in Figure 1 and factor sparsity in Figure 2. For the Gaussian noise experiments, the reconstruction errors and factor sparsity are shown in Figure 3 and Figure 4 respectively.

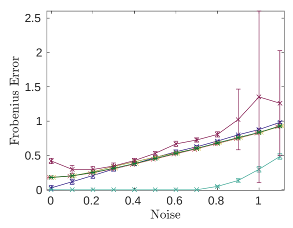
Varying density with tropical noise. In our first experiment we studied the effects of varying the density of the factor matrices in presence of the tropical noise. We changed the density of the factors from 10% to 100% with an increment of 10%, while keeping the noise level at 10%. Figure 1(a) shows the reconstruction error and Figure 2(a) the sparsity of the obtained factors. `Capricorn` is consistently the best method, obtaining almost perfect reconstruction; only when the density approaches 100% does its reconstruction error deviate slightly from 0. This is expected since the data was generated with the tropical (flipping) noise that `Capricorn` is designed to optimize. Compared to `Capricorn` all other methods clearly underperform, with `Cancer` being the second best. With the exception of `ALSR5`, all NMF methods obtain results similar to those of `SVD`, while having a somewhat higher reconstruction error than `Cancer`. That `SVD` and NMF methods (except `ALSR5`) start behaving better at higher levels of density indicates that these matrices can be explained relatively well using standard algebra. `Capricorn` and `Cancer` also have the highest sparsity of factors, with `Capricorn` exhibiting a decrease in sparsity as the density of the input increases. This behaviour is desirable since ideally we would prefer to find factors that are as close to the original ones as possible. For NMF methods there is a trade-off between the reconstruction error and the sparsity of the factors – the algorithms that were worse at reconstruction tend to have sparser factors.

Varying tropical noise. The amount of noise is always with respect to the number of nonzero elements in a matrix, that is, for a matrix $\boldsymbol{A}$ with $\kappa(\boldsymbol{A})$ nonzero elements and noise level α, we flip $\alpha\kappa(\boldsymbol{A})$ elements to random values. There are two versions of this experiment – one with factor density 30% and the other with 60%. In both cases we varied the noise level from 0% to 110% with increments of 10%. Figure 1(b) and Figure 1(c) show the respective reconstruction errors and Figure 2(b) and Figure 2(c) the corresponding sparsities of the obtained factors. In the low-density case, `Capricorn` is consistently the best method with essentially perfect reconstruction for up to 80% of noise. In the high-density case, however, the noise has more severe effects, and in particular after 60% of noise, `Cancer`, `SVD`, and all versions of NMF are better than `Capricorn`. The severity of the noise is, at least partially, explained by the fact that in the denser data we flip more elements than in sparser data: for example when the data matrices are full, at 50% of noise, we have already replaced half of the values in the matrices with random values. Further, the quick increase of the reconstruction error for `Capricorn` hints strongly that the max-times structure of the data is mostly gone at these noise levels. `Capricorn` also produces clearly the sparsest factors for the low density case, and is mostly tied with `Cancer` and `ALSR5` when the density is high. It should be noted however that `ALSR5` generally has the highest reconstruction error among all the methods, which suggests that its sparse factors come at the cost of recovering little structure from the data.

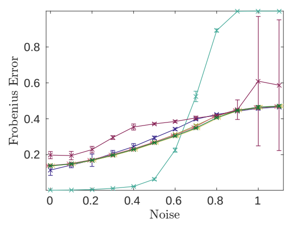
Varying rank with tropical noise. Here we test the effects of the (max-times) rank, with the assumption that higher-rank matrices are harder to reconstruct. The true max-times rank of the data varied from 2 to 20 with increments of 2. There are three variations of this experiment: with 30% factor density and 10% noise (Figure 1(d)), with 30% factor density and 50% noise (Figure 1(e)), and with 60% factor density and 10% noise (Figure 1(f)). The corresponding sparsities are shown on Figures 2(d), 2(e), and 2(f). `Capricorn` has a clear advantage for all settings, obtaining nearly perfect reconstruction. `Cancer` is generally second best, except for the high noise case, where it is mostly tied with



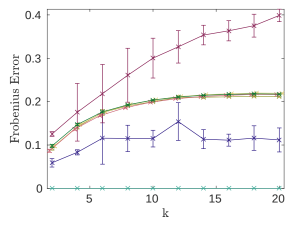
(a) Varying density test.



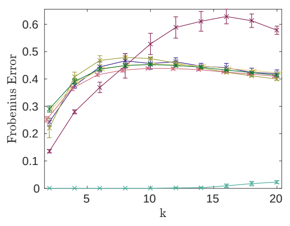
(b) Varying noise test.



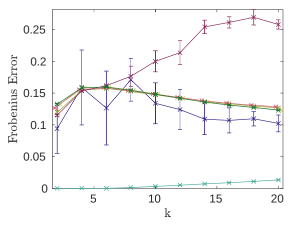
(c) Varying noise with high density.



(d) Varying rank test with 10% noise and 30% factor density.



(e) Varying rank test with 50% noise and 30% factor density.



(f) Varying rank test with 10% noise and 60% factor density.

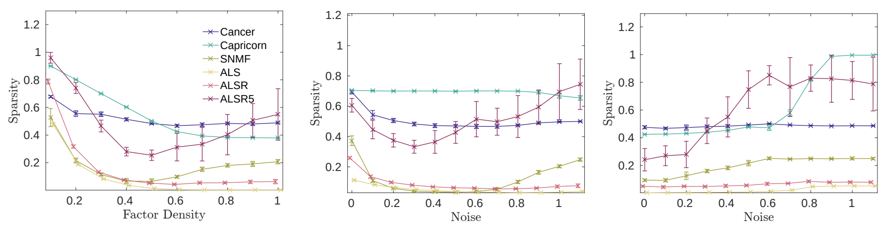
Figure 1: **Reconstruction errors on synthetic data with tropical noise**. x-axis is the parameter varied and y-axis is the relative Frobenius norm. All results are averages over 10 random matrices and the width of the error bars is twice the standard deviation.

a bunch of NMF methods. Interestingly, on the last two plots the reconstruction error actually drops for `Cancer`, `SVD`, and NMF-based methods. This is a strong indication that at this point they no longer can extract meaningful structure in the data, and the improvement of the reconstruction error is largely due to uniformization of the data caused by high density and high noise levels.

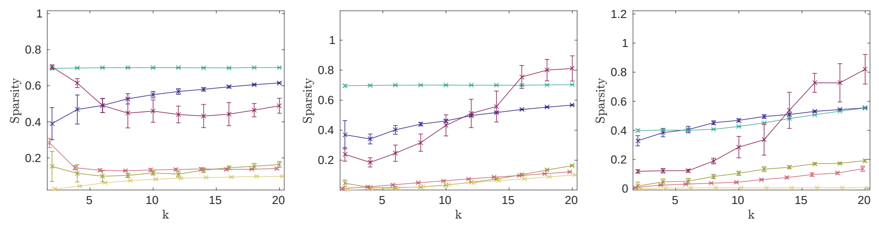
Varying Gaussian noise. Here we investigate how the algorithms respond to different levels of Gaussian noise, which was varied from 0 to 0.14 with increments of 0.01. A level of noise is a standard deviation of the Gaussian noise used to generate the noise matrix as described earlier. The factor density was kept at 50%. The results are given on Figure 3(a) (reconstruction error) and Figure 4(a) (sparsity of factors).

Here `Cancer` is generally the best method in reconstruction error, and second in sparsity only to `Capricorn`. The only time it loses to any method is when there is no noise, and `Capricorn` obtains a perfect decomposition. This is expected since `Capricorn` is by design better at spotting pure subtropical structure.

Varying density with Gaussian noise. In this experiment we studied what effects the density of factor matrices used in data generation has on the algorithms' performance. For this purpose we varied the density from 10% to 100% with increments of 10% while keeping the other parameters fixed. There are two versions of this experiment, one with low noise level of 0.01 (Figures 3(b) and 4(b)), and a more noisy case at 0.08 (Figures 3(c) and 4(c)).



(a) Varying density test. (b) Varying noise test. (c) Varying noise with high density.



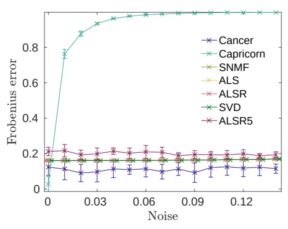
(d) Varying rank test with 10% noise and 30% factor density. (e) Varying rank test with 50% noise and 30% factor density. (f) Varying rank test with 10% noise and 60% factor density.

Figure 2: **Sparsity (fraction of zeroes) of the factor matrices for synthetic data with tropical noise.** x-axis is the parameter varied and y-axis is the sparsity of the factors. The markers are averages of 10 random matrices and the width of the error bars is twice the standard deviation.

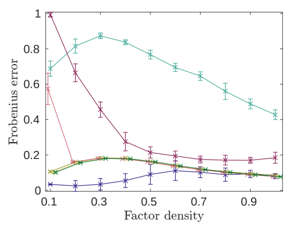
`Cancer` provides the least reconstruction error in this experiment, being clearly the best until the density is 0.7, from which point on it is tied with `SVD` and the NMF-based methods (the only exception being the least-dense high-noise case, where `ALSR` obtains a slightly better reconstruction error). `Capricorn` is the worst by a wide margin, but this is not surprising, as the data does not follow its assumptions. On the other hand, `Capricorn` does produce generally the sparsest factorization, but these are of little use given its bad reconstruction error. `Cancer` produces the sparsest factors from the remaining methods, except in the first few cases where `ALSR5` is sparser (and worse in reconstruction error), meaning that `Cancer` produces factors that are both the most accurate and very sparse.

Varying rank with Gaussian noise. The purpose of this test is to study the performance of algorithms on data of different max-times ranks. We varied the true rank of the data from 2 to 20 with increments of 2. The factor density was fixed at 50% and Gaussian noise at 0.01. The results are shown on Figure 3(d) (reconstruction error) and Figure 4(d) (sparsity of factors). The results are similar to those considered above, with `Cancer` returning the most accurate and second sparsest factorizations.

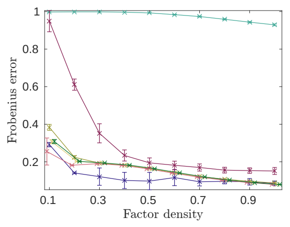
Optimizing the Jensen–Shannon divergence. By default `Cancer` optimizes the Frobenius reconstruction error, but it can be replaced by an arbitrary additive cost function. We performed experiments with Jensen–Shannon diver-



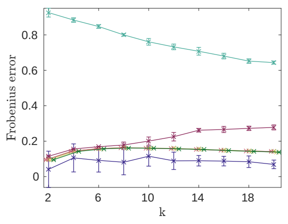
(a) Varying noise with density 50%



(b) Varying density with low Gaussian noise



(c) Varying density with high Gaussian noise



(d) Varying rank; 50% density and low Gaussian noise

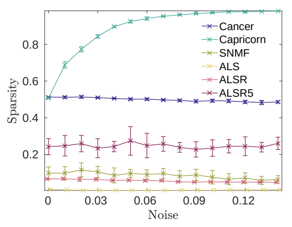
Figure 3: **Reconstruction error (Frobenius norm) for synthetic data with Gaussian noise noise.** The markers are averages of 10 random matrices and the width of the error bars is twice the standard deviation.

gence, which is given by the formula

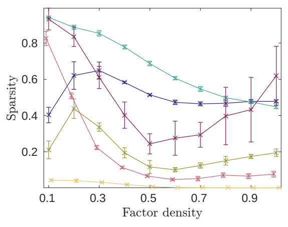
$$J(\boldsymbol{A}, \boldsymbol{B}) = \sum_{ij} \boldsymbol{A}_{ij} \log\left(\frac{2\boldsymbol{A}_{ij}}{\boldsymbol{A}_{ij} + \boldsymbol{B}_{ij}}\right) + \boldsymbol{B}_{ij} \log\left(\frac{2\boldsymbol{B}_{ij}}{\boldsymbol{A}_{ij} + \boldsymbol{B}_{ij}}\right) . \qquad (23)$$

It is easy to see that (23) is an additive function, and hence can be plugged into `Cancer`. Figure 5, shows how this version of `Cancer` compares to other methods. The setup is the same as in the corresponding experiments on Figure 3, except that we have removed `ALSR5` because of its overall bad performance. In all these experiments it is apparent that this version of `Cancer` is inferior to that optimizing the Frobenius error, but is generally on par with `SVD` and NMF-based methods. Also for the varying density test (Figure 5(b)) it produces better reconstruction errors than `SVD` and all the NMF methods, until the density reaches 50%, after which they become tied.

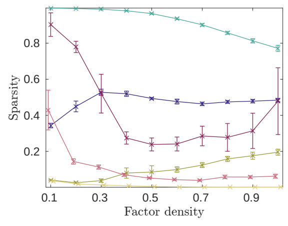
Prediction. In this experiment we choose a random holdout set and remove it from the data (elements of this set are marked as missing values). We then try to learn the structure of the data from its remaining part using `Capricorn` and `WNMF`, and finally test how well they predict the values inside the holdout set. All input matrices are integer-valued and since the recovered data produced by the algorithms can be continuous-valued, we round it to the nearest integer. The quality of the prediction is measured as the fraction of correct values in the hold-out set, and the results are reported in Figure 6. It is easy to see that as the fraction of held-out data increases, `Capricorn`'s results get worse, as expected, but it still is consistently better than `WNMF` that does not seem to be able to recover any specific structure.



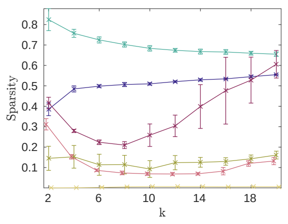
(a) Varying noise with density 50%



(b) Varying density with low Gaussian noise

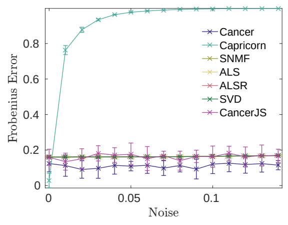


(c) Varying density with high Gaussian noise

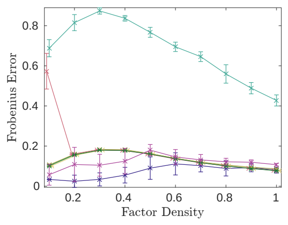


(d) Varying rank; 50% density and low Gaussian noise

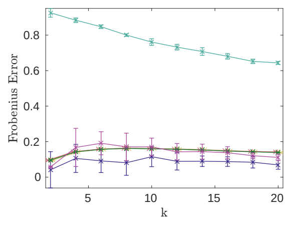
Figure 4: **Sparsity (fraction of zeroes) of the factor matrices for synthetic data with Gaussian noise.** The markers are averages of 10 random matrices and the width of the error bars is twice the standard deviation.



(a) Varying noise test.



(b) Varying density test.



(c) Varying rank test with 10% noise and 30% factor density.

Figure 5: **Comparison of `Cancer` with Jensen–Shannon objective and other methods on synthetic data with Gaussian noise.** x-axis is the parameter varied and y-axis is the relative Frobenius error. All results are averages over 10 random matrices and the width of the error bars is twice the standard deviation.

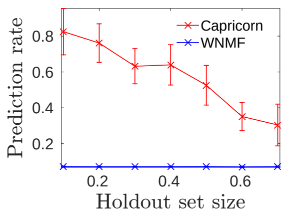


Figure 6: **Prediction rate on synthetic data**. x-axis represents the size of the holdout set and y-axis is the correct prediction rate (higher is better). All results are averages over 10 random matrices and the width of the error bars is twice the standard deviation.

Discussion. The synthetic experiments confirm that both `Capricorn` and `Cancer` are able to recover matrices with max-times structure. The main practical difference between then is that `Capricorn` is designed to handle the tropical (flipping) noise, while `Cancer` is meant for the data that is perturbed with white (Gaussian) noise. While `Capricorn` is clearly the best method when the data has only the flipping noise – and is capable of tolerating very high noise levels – its results deteriorate when we apply Gaussian noise. Hence, when the exact type of noise is not known a priori, it is advisable to try both methods. It is also important to note that `Cancer` is actually a framework of algorithms as it can optimize various objective. In order to demonstrate that, we performed experiments with Jensen–Shannon divergence as objective and obtained results that are, while inferior to `Cancer` that optimizes the Frobenius error, still slightly better than the rest of the algorithms. Overall we can conclude that `SVD` and the NMF-based methods generally cannot recover the structure from subtropical data, that is, we cannot use existing methods as a substitute to find the max-times structure neither for the reconstruction nor for the prediction tasks.

5.3 Real-world experiments.

The main purpose of the real-world experiments is to study to which extend `Capricorn` and `Cancer` can find max-times structure from various real-world data sets. Having established with the synthetic experiments that both algorithms are capable of finding the structure when it is present, here we look at what kind of results they obtain in the real-world data.

It is probably unrealistic to expect real-world data sets to have "pure" max-times structure, as in the synthetic experiments. Rather, we expect `SVD` to be the best method (in reconstruction error's sense), and our algorithms to obtain reconstruction error comparable to the NMF-based methods. We will also verify that the results from the real-world data sets are intuitive.

The datasets

Bas1LP represents a linear program.[4] It is available from the University of Florida Sparse Matrix Collection[5] (Davis and Hu, 2011).

[4]Submitted to the matrix repository by Csaba Meszaros.

[5]`http://www.cise.ufl.edu/research/sparse/matrices/`, accessed 18 July 2017

Trec12 is a brute force disjoint product matrix in tree algebra on n nodes.[6] It can be obtained from the same repository as Bas1LP.

Worldclim was obtained from the global climate data repository.[7] It describes historical climate data across different geographical locations in Europe. Columns represent minimum, maximum, and average temperatures and precipitation, and rows are 50-by-50 kilometer squares of land where measurements were made. We preprocessed every column of the data by first subtracting its mean, dividing by the standard deviation, and then subtracting its minimum value, so that the smallest value becomes 0.

NPAS is a nerdiness personality test that uses different attributes to determine the level of nerdiness of a person.[8] It contains answers by 1418 respondents to a set of 36 questions that asked them to self-assess various statements about themselves on a scale of 1 to 7. We preprocessed NPAS analogously to Worldclim.

Eigenfaces is a subset of the Extended Yale Face collection of face images (Georghiades et al., 2000). It consists of 32-by-32 pixel images under different lighting conditions. We used a preprocessed data by Xiaofei He et al.[9] We selected a subset of pictures with lighting from the left and then preprocessed the input matrix by first subtracting from every column its smallest element and then dividing it by its standard deviation.

4News is a subset of the 20Newsgroups dataset,[10] containing the usage of 800 words over 400 posts for 4 newsgroups.[11] Before running the algorithms we represented the dataset as a TF-IDF matrix, and then scaled it by dividing each entry by the greatest entry in the matrix.

HPI is a land registry house price index.[12] Rows represent months, columns are locations, and entries are residential property price indices. We preprocessed the data by first dividing each column by its standard deviation and then subtracting its minimum, so that each column has minimum 0.

Movielense is a collection of user ratings for a set of movies. The original dataset[13] consists of 100000 ratings from 1000 users on 1700 movies, with ratings ranging from 1 to 5. In order to be able to perform cross-validation on it, we had to preprocess Movielense by removing users that rated fewer than 10 movies and movies that were rated less than 5 times. After that we were left with 943 users, 1349 movies and 99287 ratings.

The basic properties of these data sets are listed in Table 1.

Quantitative results: reconstruction error, sparsity, and convergence

The following experiments are meant to test `Cancer` and `Capricorn`, and how they compare versus other methods, such as `SVD` and NMF. Table 2 provides the relative Frobenius reconstruction errors for various real-world data sets. We omitted `ALSR5` from these experiments due to its bad performance with the

[6]Submitted by Nicolas Thiery.

[7]The raw data is available at `http://www.worldclim.org/`, accessed 18 July 2017.

[8]Tha dataset can be obtained on the online personality website `http://personality-testing.info/_rawdata/NPAS-data.zip`, accessed 18 July 2017.

[9]`http://www.cad.zju.edu.cn/home/dengcai/Data/FaceData.html`, accessed 18 July 2017

[10]`http://qwone.com/~jason/20Newsgroups/`, accessed 18 July 2017

[11]The authors are grateful to Ata Kabán for pre-processing the data, see Miettinen (2009).

[12]Available at `https://data.gov.uk/dataset/land-registry-house-price-index-background-tables/`, accessed 18 July 2017

[13]Available at `http://grouplens.org/datasets/movielens/100k/`, accessed 18 July 2017

Table 1: Real world datasets properties.

Algorithm	Rows	Columns	Density
Bas1LP	9825	5411	1.1%
Trec12	2726	551	10.0%
Worldclim	2575	48	99.9%
NPAS	1418	36	99.6%
Eigenfaces	1024	222	97.0%
4News	400	800	3.5%
HPI	253	177	99.5%
Movielense	943	1349	7.8%

Table 2: Reconstruction error for various real-world datasets.

	Worldclim	NPAS	Eigenfaces	4News	HPI
$k =$	10	10	40	20	15
`Cancer`	0.071	0.240	0.204	0.556	0.027
`Capricorn`	0.392	0.395	0.972	0.987	0.217
`SNMF`	0.046	0.225	0.178	0.546	0.023
`ALS`	0.087	0.227	0.313	0.538	0.074
`ALSR`	0.122	0.226	0.294	1.000	0.045
`SVD`	0.025	0.209	0.140	0.533	0.015

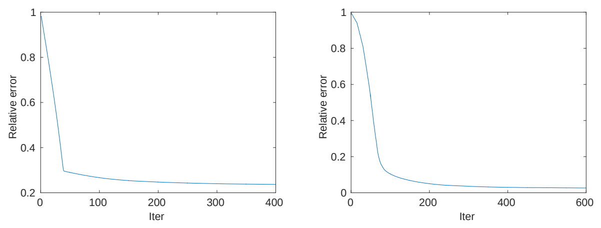
synthetic data. `SVD` is, as expected, consistently the best method. Somewhat surprisingly, Hoyer's `SNMF` is usually the second-best method, even though it did not show any advantage over other methods in the synthetic experiments. `Cancer` is usually the third-best method (with the exception of 4News and NPAS), and often very close to `SNMF` in reconstruction error. Overall, it seems `Cancer` is capable of finding max-times structure that is comparable to what NMF-based methods provide. Consequently, we can study the max-times structure found by `Cancer`, knowing that it is (relatively) accurate. On the other hand `Capricorn` has a high reconstruction error. The discrepancy between `Cancer`'s and `Capricorn`'s results indicates that the datasets used cannot be represented using "pure" subtropical structure. Rather they are either a mix of NMF and subtropical patterns or have relatively high levels of continuous noise.

The sparsity of the factors for real-world data is presented in Table 3, except for `SVD`. Here, `Cancer` often returns the second-sparsest factors (being second only to `Capricorn`), but with 4News and HPI, `ALSR` obtains sparser decompositions.

We also studied the convergence behavior of `Cancer` using some of the real-world data sets. The results can be seen in Figure 7, where we plot the relative error with respect to the iterations over the main for-loop in `Cancer`. As we can see, in both cases `Cancer` has obtained a good reconstruction error already after few full cycles, with the remaining runs only providing minor improvements. We can deduce that `Cancer` reaches quickly an acceptable solution.

Table 3: Factor sparsity for various real-world datasets.

	Worldclim	NPAS	Eigenfaces	4News	HPI
$k =$	10	10	40	20	15
`Cancer`	0.645	0.528	0.571	0.812	0.422
`Capricorn`	0.795	0.733	0.949	0.991	0.685
`SNMF`	0.383	0.330	0.403	0.499	0.226
`ALS`	0.226	0.120	0.434	0.513	0.331
`ALSR`	0.275	0.117	0.480	1.000	0.729



(a) NPAS (b) HPI

Figure 7: Convergence rate of `Cancer` for two real-world datasets. Each iteration is a single run of `UpdateBlock`, that is if a factorization has rank k, then one full cycle would correspond to k iterations.

Prediction

Here we investigate how well both `Capricorn` and `Cancer` can predict missing values in the data.

In order to test `Capricorn`, we ran missing value prediction tests on Bas1LP and Trec12 datasets, and compare it against `NMF`, `WNMF`, and `SVD`. The setup is as follows. A random holdout set is chosen that comprises 10% of the nonzero elements and then removed from the data. Since the input matrices are integer valued, we round the output of the algorithms to the nearest integer and report the fraction of correctly predicted values. There are two versions of this experiment – one where all elements in the data are taken into account and one where zero entries are ignored, that is, they do not contribute to the error. The motivation for this test is that `Capricorn` always aims to extract subtropical patterns, sometimes even at the expense of covering zeros with nonzero values. We therefore want to see how well it performs when only the "significant" part of the data is counted. It is worth noting though that while `Capricorn` and `WNMF` have an option to ignore certain entries in an input matrix, `NMF` does not. Hence the `NMF` algorithm is at a disadvantage here, though we still show its result for completeness. The results for both prediction experiments where zeros "count" and "don't count" are shown in Table 4, left and right, respectively. In both cases `WNMF` is the best method, whereas `Capricorn` is normally the second-best. As expected, `Capricorn`'s results improve greatly when zero elements are ignored.

Next we conduct prediction experiments with `Cancer`. We tested it on the Movielense dataset and compared against `WNMF`. The choice of `WNMF` is motivated by its ability to ignore elements in the input data and its generally good per-

Table 4: Prediction accuracy on Bas1LP and Trec12 datasets. Left: accuracy is computed over all entries. Right: accuracy is computed over the non-zero entries.

Algorithm	Bas1LP	Trec12
Capricorn	74.0	19.8
NMF	23.4	18.3
WNMF	85.2	39.9
SVD	28.2	20.5

Algorithm	Bas1LP	Trec12
Capricorn	85.2	39.3
NMF	29.1	19.6
WNMF	93.1	49.8
SVD	29.1	22.5

formance on the previous tests. To get a more complete view on how good the predictions are, we report various measures of quality: Frobenius error, root mean square error (RMSE), reciprocal rank, Spearman's ρ, mean absolute error (MAE), Jensen–Shannon divergence (JS), optimistic reciprocal rank, and Kendall's τ. The tests can be divided into two categories. The first one, which comprises Frobenius error, root mean square error, mean absolute error, and Jensen–Shannon divergence, aims to quantify the distance between the original data and the reconstructed matrix. The second group of tests finds the correlation between rankings of movies for each user. It includes Spearman's ρ, Kendall's τ, reciprocal rank, and optimistic reciprocal rank. All these measures are well known, with perhaps only the reciprocal rank requiring some explanation. Let us first denote by U the set of all users. In the following, for each user $u \in U$ we only consider the set of movies $M(u)$ that this user has rated that belong to the holdout set. The ratings by user u induce a natural ranking on $M(u)$. On the other hand both Cancer and WNMF produce approximations $r'(u,m)$ to the true ratings $r(u,m)$, which also induce a corresponding ranking of the movies. The reciprocal rank is a convenient way of comparing the rankings obtained by the algorithms to the original one. For any user $u \in U$, denote by $H(u)$ a set of movies that this user ranked the highest (that is $H(u) = \{m \in M(u) \mid r(u,m) = \max_{m' \in M(u)} r(u,m')\}$). The reciprocal rank for user u is now defined as

$$RR(u) = \frac{1}{\min\limits_{m \in H} R(u,m)} , \tag{24}$$

where $R(u,m)$ is the rank of the movie m within $M(u)$ according to the rating approximations given by the algorithm in question. Now the mean reciprocal rank is defined as the average of the reciprocal ranks for each individual user $MRR = \frac{1}{|U|} \sum_{u \in U} RR(u)$. When computing the ranks $R(u,m)$, all tied elements receive the same rank, which is computed by averaging. That means that if, say, movies m_1 and m_2 have tied ranks of 2 and 3, then they both receive the rank of 2.5. An alternative way is to always assign the smallest possible rank. In the above example both m_1 and m_2 will receive rank 2. When ranks $R(u,m)$ are computed like this, the equation (24) defines the optimistic reciprocal rank.

We perform standard cross-validation tests where a random selection of elements is chosen as a holdout set and removed from the data. The data has 943 users, each having rated from 19 to 648 movies. A holdout set is chosen by sampling uniformly at random 5 ratings from each user. We run the algorithms, while treating the elements from the holdout set as missing values, and then

Table 5: Comparison between the predictive power of `Cancer` and `WNMF` on the Movielense data. The arrow after the value indicates whether higher or lower values are preferable. The p-values are computed using the Wilcoxon signed-rank test.

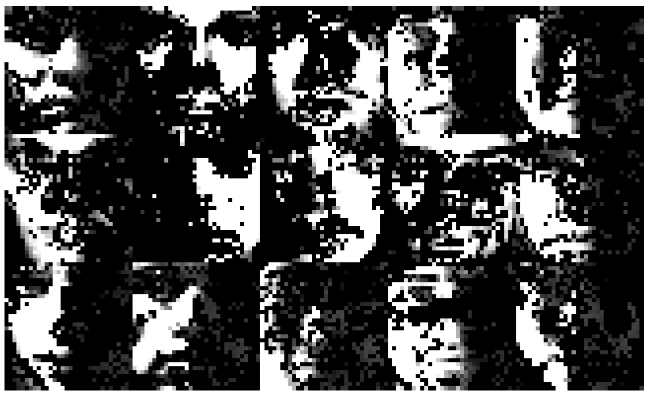
	Frobenius		RMSE	
	value(↓)	p-value	value(↓)	p-value
`Cancer`	**0.2851** $\pm$ 0.003	< 0.0001	**1.0724** $\pm$ 0.013	< 0.0001
`WNMF`	0.2969 $\pm$ 0.003		1.1169 $\pm$ 0.011	
	Recip. rank		Spearman's ρ	
	value(↑)	p-value	value(↑)	p-value
`Cancer`	**0.7472** $\pm$ 0.011	0.0994	0.3097 $\pm$ 0.016	0.2124
`WNMF`	0.7423 $\pm$ 0.009		**0.3133** $\pm$ 0.015	
	MAE		JS	
	value(↓)	p-value	value(↓)	p-value
`Cancer`	**0.8158** $\pm$ 0.008	< 0.0001	**0.0198** $\pm$ 0.001	< 0.0001
`WNMF`	0.8503 $\pm$ 0.007		0.0206 $\pm$ 0.000	
	Recip. rank opt.		Kendall's τ	
	value(↑)	p-value	value(↑)	p-value
`Cancer`	**0.7472** $\pm$ 0.011	0.0994	0.2685 $\pm$ 0.014	0.2204
`WNMF`	0.7423 $\pm$ 0.0093		**0.2712** $\pm$ 0.013	

compare the reconstructed matrices to the original data. This procedure is repeated 10 times.

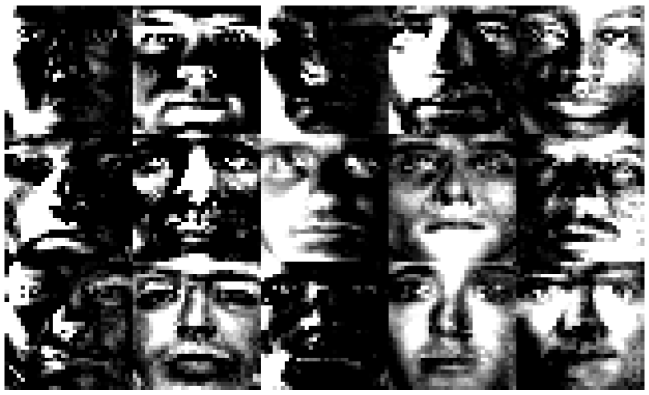
For each test, Table 5 shows the mean and the standard deviation of the results of each algorithm. In addition we report the p-value based on the Wilcoxon signed-rank test. It shows if an advantage of one method over the other is statistically significant. We say that a method A is significantly better than method B if the p-value is < 0.05. `Cancer` is significantly better for the Frobenius error, root mean square error, mean absolute error, and Jensen–Shannon divergence. For the remaining tests the results are less clear, with `Cancer` winning on both version of the reciprocal rank, and `WNMF` being better on Spearman's ρ and Kendall's τ tests. None of these results are statistically significant as the p-values are quite high. In summary, our experiments show that `Cancer` is significantly better in tests that measure the direct distance between the original and the reconstructed matrices, whereas for the ranking experiments it is difficult to give any of the algorithms an edge.

Interpretability of the results

The crux of using max-times factorizations instead of standard (nonnegative) ones is that the factors (are supposed to) exhibit the "winner-takes-it-all" structure instead of the "parts-of-whole" structure. To demonstrate this, we plotted the left factor matrices for the Eigenfaces data for `Cancer` and `ALS` in Figure 8. At first, it might look like `ALS` provides more interpretable results, as most factors



(a) Cancer

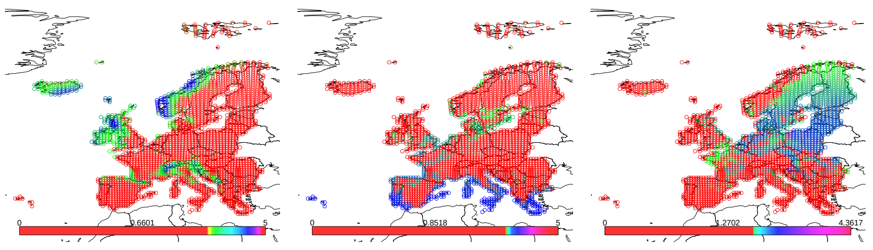


(b) ALS

Figure 8: Cancer finds the dominant patterns from the Eigenfaces data. Pictured are the left factor matrices for the Eigenfaces data.

are easily identifiable as faces. This, however, is not very interesting result: we already knew that the data has faces, and many factors in the ALS's result are simply some kind of 'prototypical' faces. The results of Cancer are harder to identify on the first sight. Upon closer inspection, though, one can see that they identify areas that are lighter in the different images, that is, have higher grayscale values. These factors tell us the variances in the lightning in the different photos, and can reveal information we did not know a priori. Further, as seen in Table 4, Cancer obtains better reconstruction error than ALS with this data, confirming that these factors are indeed useful to recreate the data.

In Figure 9, we show some factors from Cancer when applied to the Worldclim data. These factors clearly identify different bioclimatic areas from Europe: In Figure 9(a) we can identify the mountainous areas in Europe, including the Alps, the Pyrenees, the Scandes, and Scottish Highlands. In Figure 9(b) we can identify the mediterranean coastal regions, while in Figure 9(c) we see the temperate climate zone in blue, with the green color extending to the boreal zone. In all pictures, red corresponds to (near) zero values. As we can see, Cancer identifies these areas crisply, making it easy for the analyst to know which areas



(a) (b) (c)

Figure 9: `Cancer` can find interpretable factors from the Worldclim data. Shown are the values for three columns in the left-hand factor matrix $\boldsymbol{B}$ on a map. Red is zero.

Table 6: Top three attributes for the first two factors of NPAS.

Factor 1	Factor 2
I am more comfortable with my hobbies than I am with other people	I have played a lot of video games
I gravitate towards introspection	I collect books
I sometimes prefer fictional people to real ones	I care about super heroes

to look at.

In order to interpret NPAS we first observe that each column represents a single personality attribute. Denote by $\boldsymbol{A}$ the obtained approximation of the original matrix. For each rank-1 factor $\boldsymbol{X}$ and each column $\boldsymbol{A}_i$ we define the score $\sigma(i)$ as the number of elements in $\boldsymbol{A}_i$ that are determined by $\boldsymbol{X}$. By sorting attributes in descending order of $\sigma(i)$ we obtain relative rankings of the attributes for a given factor. The results are shown in Table 6. The first factor clearly shows introverted tendencies, while the second one can be summarized as having interests in fiction and games.

6 Related Work

Here we present earlier research that is related to the subtropical matrix factorization. We start by discussing classic methods, such as SVD and NMF, that have long been used for various data analysis tasks, and then continue with approaches that use idempotent structures. Since the tropical algebra is very closely related to the subtropical algebra, and since there has been a lot of research on it, we dedicate the last subsection to discuss it in more detail.

6.1 Matrix factorization in data analysis.

Matrix factorization methods play a crucial role in data analysis as they help to find low-dimensional representations of the data and uncover the underlying latent structure. A classic example of a real-valued matrix factorization is the singular value decomposition (SVD) (Golub and Van Loan, 2012, see e.g.), which is very well known and finds extensive applications in various disciplines, such as for example signal processing and natural language processing. The SVD of a real n-by-m matrix $\boldsymbol{A}$ is a factorization of the form $\boldsymbol{A} = \boldsymbol{U\Sigma V}^T$, where $\boldsymbol{U} \in \mathbb{R}^{n\times n}$ and $\boldsymbol{V} \in \mathbb{R}^{m\times m}$ are orthogonal matrices, and $\boldsymbol{\Sigma} \in \mathbb{R}^{n\times m}$ is a rectangular diagonal matrix with nonnegative entries. An important property of SVD is that it provides the best low-rank approximation of a given matrix with respect to the Frobenius norm (Golub and Van Loan, 2012), giving rise to the so called truncated SVD. This property is frequently used to separate important parts of data from the noise. For example, it was used by Jha and Yadava (2011) to remove the noise from sensor data in electronic nose systems. Another prominent usage of the truncated SVD is in dimensionality reduction (see for example Sarwar et al., 2000; Deerwester et al., 1990).

Despite SVD being so ubiquitous, there are some restrictions to its usage in data mining due to possible presence of negative elements in the factors. In many applications negative values are hard to interpret, and thus other methods have to be used. Nonnegative matrix factorization (NMF) is a way to tackle this problem. For a given nonnegative real matrix $\boldsymbol{A}$, the NMF problem is to find a decomposition of $\boldsymbol{A}$ into two matrices $\boldsymbol{A} \approx \boldsymbol{BC}$ such that $\boldsymbol{B}$ and $\boldsymbol{C}$ are also nonnegative. Its applications are extensive and include text mining (Pauca et al., 2004), document clustering (Xu et al., 2003), pattern discovery (Brunet et al., 2004), and many other. This area drew considerable attention after a publication by Lee and Seung (1999), where they provided an efficient algorithm for solving the NMF problem. It is worth mentioning that even though the paper by Lee and Seung is perhaps the most famous in NMF literature, it was not the first one to consider this problem. Earlier works include Paatero and Tapper (1994) (see also Paatero, 1997), Paatero (1999), and Cohen and Rothblum (1993). Berry et al. (2007) provide an overview of NMF algorithms and their applications. There exist various flavours of NMF that impose different constraints on the factors; for example Hoyer (2004) used sparsity constraints. Though both NMF and SVD perform approximations of a fixed rank, there are also other ways to enforce compact representation of data. For example, in maximum-margin matrix factorization constraints are imposed on the norms of factors. This approach was exploited by Srebro et al. (2004), who showed it to be a good method for predicting unobserved values in a matrix. The authors also indicate that posing constraints on the factor norms, rather than on the rank, yields a convex optimization problem, which is easier to solve.

6.2 Idempotent semirings.

The concept of the subtropical algebra is relatively new, and as far as we know, its applications in data mining are not yet well studied. Indeed, its only usage for data analysis that we are aware of was by Weston et al. (2013), where it was used as a part of a model for collaborative filtering. The authors modeled users as a set of vectors, where each vector represents a single aspect about the user (e.g. a particular area of interest). The ratings are then reconstructed by selecting the highest scoring prediction using the max operator. Since their model uses max as well as the standard plus operation, it stands on the border between the standard and the subtropical worlds.

Boolean algebra, despite being limited to the binary set $\{0, 1\}$, is related to the subtropical algebra by virtue of having the same operations, and is thus a restriction of the latter to $\{0, 1\}$. By the same token, when both factor matrices are binary, their subtropical product coincides with the Boolean product, and hence the Boolean matrix factorization can be seen as a degenerate case of the subtropical matrix factorization problem. The dioid properties of the Boolean algebra can be checked trivially. The motivation for the Boolean matrix factorization comes from the fact that in many applications data is naturally represented as a binary matrix (e.g. transaction databases), which makes it reasonable to seek decompositions that preserve the binary character of the data. The conceptual and algorithmic analysis of the problem was done by Miettinen (2009), with the focus mainly on the data mining perspective of the problem. For a linear algebra perspective see Kim (1982), where the emphasis is put on the existence of exact decompositions. A number of algorithms have been proposed for solving the BMF problem (Miettinen et al., 2008; Lu et al., 2008; Lucchese et al., 2014; Karaev et al., 2015).

6.3 Tropical algebra.

Another close cousin of the max-times algebra is the max-plus, or so called tropical algebra, which uses plus in place of multiplication. It is also a dioid due to the idempotent nature of the max operation. As was mentioned earlier, the two algebras are isomorphic, and hence many of the properties are identical (see Sections 2 and 3 for more details).

Despite the theory of the tropical algebra being relatively young, it has been thoroughly studied in recent years. The reason for this is that it finds extensive applications in various areas of mathematics and other disciplines. An example of such a field is the discrete event systems (DES) (Cassandras and Lafortune, 2008), where the tropical algebra is ubiquitously used for modeling (see e.g. Baccelli et al., 1992; Cohen et al., 1999). Other mathematical disciplines where the tropical algebra plays a crucial role are optimal control (Gaubert, 1997), asymptotic analysis (Dembo and Zeitouni, 2010; Maslov, 1992; Akian, 1999), and decidability (Simon, 1978, 1994).

Research on tropical matrix factorization is of interest for us because of the above mentioned isomorphism between the two algebras. However as was explained in Section 3, the approximate matrix factorizations are not directly transferable as the errors can differ dramatically. It should be mentioned that in the general case the problem of the tropical matrix factorization is NP-complete (see e.g. Shitov, 2014). De Schutter and De Moor (2002) demonstrated that if the max-plus algebra is extended in such a way that there is an additive inverse for each element, then it is possible to solve many of the standard matrix decomposition problems. Among other results the authors obtained max-plus analogues of QR and SVD. They also claimed that the techniques they propose can be readily extended to other types of classic factorizations (e.g. Hessenberg and LU decomposition). Despite the apparent successes in the realm of tropical matrix factorization, its subtropical counterpart has not received much attention, and to the best of our knowledge the first work on the subject was done by Karaev and Miettinen (2016b).

The problem of solving tropical linear systems of equations arises naturally in numerous applications, and is also closely related to matrix factorization. In order to illustrate this connection, assume that we are given a tropical matrix $\boldsymbol{A} \in \overline{\mathbb{R}}^{n \times m}$ and one of the factors $\boldsymbol{B} \in \overline{\mathbb{R}}^{n \times k}$. Then the other factor $\boldsymbol{C} \in \overline{\mathbb{R}}^{k \times m}$

can be found by solving the following set of problems

$$\boldsymbol{C}_j = \arg\min_{\boldsymbol{c} \in \overline{\mathbb{R}}^k} \|\boldsymbol{B} \diamond \boldsymbol{c} - \boldsymbol{A}_j\|_F, \; j = 1, \ldots, m \; . \tag{25}$$

Each problem in (25) requires "approximately" solving a system of tropical linear equations. The minus operation in (25) does not belong to the tropical semiring, so the approximation here should be understood in terms of minimizing the classical distance. The general form of tropical linear equations

$$\boldsymbol{A}\boldsymbol{x} \oplus \boldsymbol{b} = \boldsymbol{C}\boldsymbol{x} \oplus \boldsymbol{d} \tag{26}$$

is not always solvable (see e.g. Gaubert, 1997); however various techniques exist for checking the existence of the solution for particular cases of (26).

For equations of the form $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ the feasibility can be established for example through the so called *matrix residuation.* There is a general result that for an n-by-m matrix $\boldsymbol{A}$ over a complete idempotent semiring, the existence of the solution can be checked in $O(nm)$ time (see Gaubert, 1997). Although the tropical algebra is not complete, there is an efficient way of finding if the solution exists (Cuninghame-Green, 1979; Zimmermann, 2011). It was shown by Butkovič (2003) that this type of tropical equations is equivalent to the set cover problem, which is known to be NP-hard. This directly affects the max-times algebra through the above-mentioned isomorphism and makes the problem of precisely solving max-times linear systems of the form $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ infeasible for high dimensions.

Homogeneous equations $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{B}\boldsymbol{x}$ can be solved using the *elimination* method, which is based on the fact that the set of solutions of a homogeneous system is a finitely generated semimodule (Butkovič and Hegedüs, 1984) (independently rediscovered by Gaubert, 1992). If only a single solution is required, then according to Gaubert (1997), a method by Walkup and Borriello (1998) is usually the fastest in practice.

Now let $\boldsymbol{A}$ be a tropical square matrix of size $n \times n$. For complete idempotent semirings a solution to the equation $\boldsymbol{x} = \boldsymbol{A}\boldsymbol{x} \oplus \boldsymbol{b}$ is given by $\boldsymbol{x} = \boldsymbol{A}^*\boldsymbol{b}$ (see e.g. Salomaa and Soittola, 2012), where the operator $\boldsymbol{A}^*$ is defined as

$$\boldsymbol{A}^* = \oplus_{k=1}^{\infty} \boldsymbol{A}^k \; .$$

Since the tropical semiring is not complete (it is missing the ∞ element), $\boldsymbol{A}^*$ can not always be computed. However, when there are no positive weight circuits in the graph defined by $\boldsymbol{A}$, then we have $\boldsymbol{A}^* = \boldsymbol{A}^0 \oplus \ldots \oplus \boldsymbol{A}^{n-1}$, and all entries of $\boldsymbol{A}^*$ belong to the tropical semiring (Baccelli et al., 1992). Computing the operator $\boldsymbol{A}^*$ takes time $O(n^3)$ (see e.g. Gondran and Minoux, 1984a; Gaubert, 1997).

Another important direction of research is the eigenvalue problem $\boldsymbol{A}\boldsymbol{x} = \lambda\boldsymbol{x}$. Tropical analogues of the Perron–Frobenius theorem (see e.g. Vorobyev, 1967; Maslov, 1992), and Collatz–Wielandt formula (Bapat et al., 1995; Gaubert, 1992) were developed. For a general overview of the results in the $(\max, +)$ spectral theory, see for example Gaubert (1997).

Tropical algebra and tropical geometry were used by Gärtner and Jaggi (2008) to construct a tropical analogue of an SVM. Unlike in the classical case, tropical SVMs are localized, in the sense that the kernel at any given point is not influenced by all the support vectors. Their work also utilizes the fact that tropical hyperplanes are somewhat more complex than their counterparts in the classical geometry, which makes it possible to do multiple category classification with a single hyperplane.

7 Conclusions

Subtropical low-rank factorizations are a novel approach for finding latent structure from nonnegative data. The factorizations can be interpreted using the winner-takes-it-all interpretation: the value of the element in the final reconstruction depends only on the largest of values in the corresponding elements of the rank-1 components (cf. NMF, where the value in the reconstruction is the *sum* of the corresponding elements). That the factorizations are different does not necessarily mean that they are better in the terms of reconstruction error, although they can yield lower reconstruction error than even SVD. It does mean, however, that they find different structure from the data. This is an important advantage, as it allows the data analyst to use both the classical factorizations and the subtropical factorizations to get a broader understanding of the kinds of patterns that are present in the data.

Working in the subtropical algebra is harder than in the normal algebra, though. The various definitions for the rank, for example, do not agree, and computing many of them – including the subtropical Schein rank, which is arguably the most useful one for data analysis – is computationally hard. That said, our proposed algorithms, `Capricorn` and `Cancer`, can find the subtropical structure when it is present in the data. Not every data have subtropical structure, though, and due to the complexity of finding the optimal subtropical factorization we cannot distinguish between the cases where our algorithms fail to find the latent subtropical structure, and where it does not exist. Based on our experiments with synthetic data, our hypothesis is that the failure of finding a good factorization indicates the lack of the subtropical structure rather than the algorithms' failure.

That said, the presented algorithms are heuristics. Developing algorithms that achieve better reconstruction error is naturally an important direction of future work. In our `Equator` framework, this hinges on the task of finding the rank-1 components. In addition, the scalability of the algorithms could be improved. A potential direction could be to take into account the sparsity of the factor matrices in dominated decompositions. This could allow one to concentrate only on the non-zero entries in the factor matrices.

The connection between Boolean and (sub-)tropical factorizations raises potential directions for future work. The continuous framework could allow for easier optimization in the Boolean algebra. Also, the connection allows us to model combinatorial structures (e.g. cliques in a graph) using subtropical matrices. This could allow for novel approaches on finding such structures using continuous subtropical factorizations.

References

M. Akian. Densities of idempotent measures and large deviations. *Trans. Amer. Math. Soc.*, 351(11):4515–4543, 1999.

M. Akian, R. Bapat, and S. Gaubert. Max-Plus Algebra. In L. Hogben, editor, *Handbook of Linear Algebra.* Chapman & Hall/CRC, Boca Raton, 2007.

M. Akian, S. Gaubert, and A. Guterman. Linear independence over tropical semirings and beyond. *Contemp. Math.*, 495:1–38, 2009.

F. Baccelli, G. Cohen, G. J. Olsder, and J.-P. Quadrat. *Synchronization and Linearity: An Algebra for Discrete Event Systems.* John Wiley & Sons, Hoboken, New Jersey, 1992.

R. Bapat, D. P. Stanford, and P. Van den Driessche. Pattern properties and spectral inequalities in max algebra. *SIAM J. Matrix Anal. Appl.*, 16(3): 964–976, 1995.

M. W. Berry, M. Browne, A. N. Langville, V. P. Pauca, and R. J. Plemmons. Algorithms and applications for approximate nonnegative matrix factorization. *Comput. Stat. Data Anal.*, 52(1):155–173, 2007.

V. D. Blondel, S. Gaubert, and J. N. Tsitsiklis. Approximating the spectral radius of sets of matrices in the max-algebra is NP-hard. *IEEE Trans. Autom. Control*, 45(9):1762–1765, 2000.

J.-P. Brunet, P. Tamayo, T. R. Golub, and J. P. Mesirov. Metagenes and molecular pattern discovery using matrix factorization. *Proc. Natl. Acad. Sci. U.S.A.*, 101(12):4164–4169, 2004.

P. Butkovič. Max-algebra: The linear algebra of combinatorics? *Linear Algebra Appl.*, 367:313–335, 2003.

P. Butkovič. *Max-linear systems: Theory and algorithms.* Springer Science & Business Media, New York, 2010.

P. Butkovič and G. Hegedüs. An elimination method for finding all solutions of the system of linear equations over an extremal algebra. *Ekon.-Mat. Obzor*, 20(2):203–215, 1984.

P. Butkovič and F. Hevery. A condition for the strong regularity of matrices in the minimax algebra. *Discrete Appl. Math.*, 11(3):209–222, 1985.

C. G. Cassandras and S. Lafortune. *Introduction to discrete event systems.* Springer, Berlin, second edition, 2008.

A. Cichocki, R. Zdunek, A. H. Phan, and S. Amari. *Nonnegative matrix and tensor factorizations: Applications to exploratory multi-way data analysis and blind source separation.* John Wiley & Sons, Chichester, 2009.

G. Cohen, S. Gaubert, and J.-P. Quadrat. Max-plus algebra and system theory: Where we are and where to go now. *Annu Rev Control*, 23:207–219, Jan. 1999.

J. E. Cohen and U. G. Rothblum. Nonnegative ranks, decompositions, and factorizations of nonnegative matrices. *Linear Algebra Appl.*, 190:149–168, 1993.

R. A. Cuninghame-Green. *Minimax Algebra.* Springer, Berlin, 1979.

T. A. Davis and Y. Hu. The university of Florida sparse matrix collection. *ACM Trans Math Soft*, 38(1):1–25, 2011.

B. De Schutter and B. De Moor. The QR decomposition and the singular value decomposition in the symmetrized max-plus algebra revisited. *SIAM Rev.*, 44 (3):417–454, 2002.

S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *J. Am. Soc. Inf. Sci.*, 41:391–407, 1990.

A. Dembo and O. Zeitouni. *Large deviations techniques and applications.* Springer, Berlin, 2010.

B. Gärtner and M. Jaggi. Tropical support vector machines. Technical Report ACS-TR-362502-01, 2008.

S. Gaubert. *Théorie des systèmes linéaires dans les dioïdes.* PhD thesis, Ecole nationale supérieure des mines de Paris, 1992.

S. Gaubert. Methods and applications of (max,+) linear algebra. In *14th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 261–282. Springer, 1997.

A. S. Georghiades, P. N. Belhumeur, and D. J. Kriegman. From few to many: Generative models for recognition under variable pose and illumination. In *4th IEEE International Conference on Automatic Face and Gesture Recognition (FG)*, pages 277–284, 2000.

N. Gillis and F. Glineur. Using underapproximations for sparse nonnegative matrix factorization. *Pattern Recogn.*, 43(4):1676–1687, 2010.

G. H. Golub and C. F. Van Loan. *Matrix computations.* Johns Hopkins University Press, Baltimore, 3 edition, 2012.

M. Gondran and M. Minoux. *Graphs and algorithms.* John Wiley & Sons, New York, 1984a.

M. Gondran and M. Minoux. Linear algebra in dioids: A survey of recent results. *North-Holland Math. Stud.*, 95:147–163, 1984b.

P. Guillon, Z. Izhakian, J. Mairesse, and G. Merlet. The ultimate rank of tropical matrices. *J. Algebra*, 437:222–248, 2015.

P. O. Hoyer. Non-negative Matrix Factorization with Sparseness Constraints. *J. Mach. Learn. Res.*, 5:1457–1469, 2004.

S. K. Jha and R. Yadava. Denoising by singular value decomposition and its application to electronic nose data processing. *IEEE Sens. J.*, 11(1):35–44, 2011.

S. Karaev and P. Miettinen. Cancer: Another algorithm for subtropical matrix factorization. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*, pages 576–592, 2016a.

S. Karaev and P. Miettinen. Capricorn: An algorithm for subtropical matrix factorization. In *16th SIAM International Conference on Data Mining (SDM)*, pages 702–710, 2016b.

S. Karaev, P. Miettinen, and J. Vreeken. Getting to know the unknown unknowns: Destructive-noise resistant boolean matrix factorization. In *15th SIAM International Conference on Data Mining (SDM)*, pages 325–333, 2015.

J. Kim and H. Park. Toward faster nonnegative matrix factorization: A new algorithm and comparisons. In *8th IEEE International Conference on Data Mining (ICDM)*, pages 353–362, 2008.

K. H. Kim. *Boolean matrix theory and applications.* Marcel Dekker, New York, 1982.

K. H. Kim and F. W. Roush. Factorization of polynomials in one variable over the tropical semiring. Technical Report math/0501167, arXiv, 2005.

T. Kolda and D. O'Leary. Algorithm 805: Computation and uses of the semidiscrete matrix decomposition. *ACM Trans. Math. Softw.*, 26(3):415–435, 2000.

D. D. Lee and H. S. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791, 1999.

Y. Li and A. Ngom. The non-negative matrix factorization toolbox for biological data mining. *Source Code Biol. Med.*, 8(1):1–15, 2013.

H. Lu, J. Vaidya, and V. Atluri. Optimal boolean matrix decomposition: Application to role engineering. In *24th IEEE International Conference on Data Engineering (ICDE)*, pages 297–306, 2008.

C. Lucchese, S. Orlando, and R. Perego. A unifying framework for mining approximate top-k binary patterns. *IEEE Trans. Knowl. Data Eng*, 26(12): 2900–2913, 2014.

V. Maslov. *Idempotent analysis.* American Mathematical Society, Providence, 1992.

P. Miettinen. *Matrix decomposition methods for data mining: Computational complexity and algorithms.* PhD thesis, University of Helsinki, 2009.

P. Miettinen, T. Mielikäinen, A. Gionis, G. Das, and H. Mannila. The discrete basis problem. *IEEE Trans. Knowl. Data Eng*, 20(10):1348–1362, 2008.

P. Paatero. Least squares formulation of robust non-negative factor analysis. *Chemometr. Intell. Lab.*, 37(1):23–35, 1997.

P. Paatero. The multilinear enginea table-driven, least squares program for solving multilinear problems, including the n-way parallel factor analysis model. *J. Comp. Graph. Stat.*, 8(4):854–888, 1999.

P. Paatero and U. Tapper. Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values. *Environmetrics*, 5(2):111–126, 1994.

V. P. Pauca, F. Shahnaz, M. W. Berry, and R. J. Plemmons. Text mining using nonnegative matrix factorizations. In *4th SIAM International Conference on Data Mining (SDM)*, pages 22–24, 2004.

A. Salomaa and M. Soittola. *Automata-theoretic aspects of formal power series.* Springer Science & Business Media, New York, 2012.

B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Application of dimensionality reduction in recommender system – a case study. Technical report, GroupLens Research Group, 2000.

Y. Shitov. The complexity of tropical matrix factorization. *Adv. Math.*, 254: 138–156, 2014.

I. Simon. Limited subsets of a free monoid. In *19th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 143–150, 1978.

I. Simon. On semigroups of matrices over the tropical semiring. *Inform. Theor. Appl.*, 28(3-4):277–294, 1994.

D. Skillicorn. *Understanding complex datasets: Data mining with matrix decompositions.* Data Mining and Knowledge Discovery. Chapman & Hall/CRC, Boca Raton, 2007.

N. Srebro, J. Rennie, and T. S. Jaakkola. Maximum-margin matrix factorization. In *17th Advances in Neural Information Processing Systems (NIPS)*, pages 1329–1336, 2004.

S. A. Vavasis. On the complexity of nonnegative matrix factorization. *SIAM J. Optim.*, 20(3):1364–1377, 2009.

N. Vorobyev. Extremal algebra of positive matrices. *Elektron. Informationsverarbeitung und Kybernetik*, 3:39–71, 1967.

E. A. Walkup and G. Borriello. A general linear max-plus solution technique. In J. Gunawardena, editor, *Idempotency*, pages 406–415. Cambridge University Press, Cambridge, 1998.

J. Weston, R. J. Weiss, and H. Yee. Nonlinear latent factorization by embedding multiple user interests. In *7th ACM Conference on Recommender Systems (RecSys)*, pages 65–68, 2013.

W. Xu, X. Liu, and Y. Gong. Document clustering based on non-negative matrix factorization. In *26th Annual International ACM SIGIR Conference (SIGIR)*, pages 267–273, 2003.

U. Zimmermann. *Linear and combinatorial optimization in ordered algebraic structures.* Elsevier, Amsterdam, 2011.