Universal algorithms for solving the matrix Bellman equations over semirings

G. L. Litvinov, A. Ya. Rodionov, S. N. Sergeev & A. N. Sobolevski

Soft Computing
A Fusion of Foundations, Methodologies and Applications

ISSN 1432-7643

Soft Comput
DOI 10.1007/s00500-013-1027-5

Springer

Soft Comput
DOI 10.1007/s00500-013-1027-5

FOUNDATIONS

Universal algorithms for solving the matrix Bellman equations over semirings

G. L. Litvinov · A. Ya. Rodionov · S. N. Sergeev · A. N. Sobolevski

Abstract This paper is a survey on universal algorithms for solving the matrix Bellman equations over semirings and especially tropical and idempotent semirings. However, original algorithms are also presented. Some applications and software implementations are discussed.

1 Introduction

Computational algorithms are constructed on the basis of certain primitive operations. These operations manipulate data that describe "numbers." These "numbers" are elements of a "numerical domain," that is, a mathematical object such as the field of real numbers, the ring of integers, different semirings etc.

In practice, elements of the numerical domains are replaced by their computer representations, that is, by elements of certain finite models of these domains. Examples of models that can be conveniently used for computer representation of real numbers are provided by various modifications of floating point arithmetics, approximate arithmetics of rational numbers (Litvinov et al. 2008), interval arithmetics etc. The difference between mathematical objects ("ideal" numbers) and their finite models (computer representations) results in computational (for instance, rounding) errors.

An algorithm is called *universal* if it is independent of a particular numerical domain and/or its computer representation (Litvinov and Maslov 1996, 1998, 2000; Litvinov et al. 2000). A typical example of a universal algorithm is the computation of the scalar product (x, y) of two vectors $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$ by the formula $(x, y) = x_1 y_1 + \cdots + x_n y_n$. This algorithm (formula) is independent of a particular domain and its computer implementation, since the formula is well-defined for any semiring. It is clear that one algorithm can be more universal than another. For example, the simplest Newton–Cotes formula, the rectangular rule, provides the most universal algorithm for numerical integration. In particular, this formula is valid also for idempotent integration [that is, over any idempotent semiring, see (Kolokoltsov and Maslov 1997; Litvinov 2007)]. Other quadrature formulas (for instance, combined trapezoid rule or the Simpson formula) are independent of computer arithmetics and can be used (for instance, in the iterative form) for computations with arbitrary accuracy. In contrast, algorithms based on Gauss–Jacobi formulas are designed for fixed accuracy computations: they include constants (coefficients and nodes of these formulas) defined with fixed accuracy. (Certainly, algorithms of this type can be made more universal by including procedures for computing the constants; however, this results in an unjustified complication of the algorithms.)

Modern achievements in software development and mathematics make us consider numerical algorithms and

Communicated by A. D. Nola.

This work is supported by the RFBR-CRNF grant 11-01-93106 and RFBR grant 12-01-00886-a.

G. L. Litvinov (✉) · A. N. Sobolevski
A.A. Kharkevich Institute for Information Transmission Problems, and National Research University Higher School of Economics, Moscow, Russia
e-mail: glitvinov@gmail.com

A. Ya. Rodionov · S. N. Sergeev
Moscow Center for Continuous Mathematical Education, Moscow, Russia

S. N. Sergeev
University of Birmingham, School of Mathematics, Edgbaston B15 2TT, UK

Published online: 26 March 2013

their classification from a new point of view. Conventional numerical algorithms are oriented to software (or hardware) implementation based on floating point arithmetic and fixed accuracy. However, it is often desirable to perform computations with variable (and arbitrary) accuracy. For this purpose, algorithms are required that are independent of the accuracy of computation and of the specific computer representation of numbers. In fact, many algorithms are independent not only of the computer representation of numbers, but also of concrete mathematical (algebraic) operations on data. In this case, operations themselves may be considered as variables. Such algorithms are implemented in the form of *generic programs* based on abstract data types that are defined by the user in addition to the predefined types provided by the language. The corresponding program tools appeared as early as in Simula-67, but modern object-oriented languages (like C++, see, for instance (Lorenz 1993; Pohl 1997)) are more convenient for generic programming. Computer algebra algorithms used in such systems as Mathematica, Maple, REDUCE, and others are also highly universal.

A different form of universality is featured by iterative algorithms (beginning with the successive approximation method) for solving differential equations (for instance, methods of Euler, Euler–Cauchy, Runge–Kutta, Adams, a number of important versions of the difference approximation method, and the like), methods for calculating elementary and some special functions based on the expansion in Taylor's series and continuous fractions (Padé approximations). These algorithms are independent of the computer representation of numbers.

The concept of a generic program was introduced by many authors; for example, in Lehmann (1977) such programs were called 'program schemes.' In this paper, we discuss universal algorithms implemented in the form of generic programs and their specific features. This paper is closely related to Litvinov (2007), Litvinov and Maslov (1996, 1998), Litvinov and Maslova (2000), Litvinov et al. (2000, 2011), Sergeev (2011), in which the concept of a universal algorithm was defined and software and hardware implementation of such algorithms was discussed in connection with problems of idempotent mathematics, see, for instance, Kolokoltsov and Maslov (1997), Litvinov and Sobolevskiĭ (2001), Mikhalkin (2006), Viro (2001, 2008).

The so-called *idempotent correspondence principle*, see Litvinov and Maslov (1996, 1998), linking this mathematics with the usual mathematics over fields, will be discussed below. In a nutshell, there exists a correspondence between interesting, useful, and important constructions and results concerning the field of real (or complex) numbers and similar constructions dealing with various idempotent semirings. This correspondence can be formulated in the spirit of the well-known N. Bohr's *correspondence principle* in quantum mechanics; in fact, the two principles are closely connected (see Litvinov 2007; Litvinov and Maslov 1996, 1998). In a sense, the traditional mathematics over numerical fields can be treated as a 'quantum' theory, whereas the idempotent mathematics can be treated as a 'classical' shadow (or counterpart) of the traditional one. It is important that the idempotent correspondence principle is valid for algorithms, computer programs and hardware units.

In quantum mechanics the *superposition principle* means that the Schrödinger equation (which is basic for the theory) is linear. Similarly in idempotent mathematics the (idempotent) superposition principle (formulated by V. P. Maslov) means that some important and basic problems and equations that are nonlinear in the usual sense (for instance, the Hamilton-Jacobi equation, which is basic for classical mechanics and appears in many optimization problems, or the Bellman equation and its versions and generalizations) can be treated as linear over appropriate idempotent semirings, see Maslov (1987a, b).

Note that numerical algorithms for infinite dimensional linear problems over idempotent semirings (for instance, idempotent integration, integral operators and transformations, the Hamilton–Jacobi and generalized Bellman equations) deal with the corresponding finite-dimensional approximations. Thus idempotent linear algebra is the basis of the idempotent numerical analysis and, in particular, the *discrete optimization theory*.

Carré (1971, 1979) (see also Gondran 1975; Gondran and Minoux 1979, 2010) used the idempotent linear algebra to show that different optimization problems for finite graphs can be formulated in a unified manner and reduced to solving Bellman equations, that is, systems of linear algebraic equations over idempotent semirings. He also generalized principal algorithms of computational linear algebra to the idempotent case and showed that some of these coincide with algorithms independently developed for solution of optimization problems. For example, Bellman's method of solving the shortest path problem corresponds to a version of Jacobi's method for solving a system of linear equations, whereas Ford's algorithm corresponds to a version of Gauss–Seidel's method. We briefly discuss Bellman equations and the corresponding optimization problems on graphs, and use the ideas of Carré to obtain new universal algorithms. We stress that these well-known results can be interpreted as a manifestation of the idempotent superposition principle.

Note that many algorithms for solving the matrix Bellman equation could be found in Baccelli et al. (1992), Carré (1971, 1979), Cuninghame-Green (1979), Gondran (1975), Gondran and Minoux (2010), Litvinov et al. (2011), Litvinov and Maslova (2000), Rote (1985), Sergeev (2011), Tchourkin and Sergeev (2007). More general problems of linear algebra over the max-plus algebra are examined, for instance in Butkovič (2010).

We also briefly discuss interval analysis over idempotent and positive semirings. Idempotent interval analysis appears in Litvinov and Sobolevskiĭ (2000, 2001), Sobolevskiĭ (1999), where it is applied to the Bellman matrix equation. Many different problems coming from the idempotent linear algebra, have been considered since then, see for instance Cechlárová and Cuninghame-Green (2002), Fiedler et al. (2006), Hardouin et al. (2009), Myšková (2005, 2006). It is important to observe that intervals over an idempotent semiring form a new idempotent semiring. Hence universal algorithms can be applied to elements of this new semiring and generate interval extensions of the initial algorithms.

This paper is about software implementations of universal algorithms for solving the matrix Bellman equations over semirings. In Sect. 2 we present an introduction to mathematics of semirings and especially to the tropical (idempotent) mathematics, that is, the area of mathematics working with *idempotent semirings* (that is, semirings with idempotent addition). In Sect. 3 we present a number of well-known and new universal algorithms of linear algebra over semirings, related to discrete matrix Bellman equation and algebraic path problem. These algorithms are closely related to their linear-algebraic prototypes described, for instance, in the celebrated book of Golub and Van Loan (1989) which serves as the main source of such prototypes. Following the style of Golub and van Loan (1989) we present them in MATLAB code. The perspectives and experience of their implementation are also discussed.

2 Mathematics of semirings

2.1 Basic definitions

A broad class of universal algorithms is related to the concept of a semiring. We recall here the definition (see, for instance, Golan 2000).

A set S is called a *semiring* if it is endowed with two associative operations: *addition* $\oplus$ and *multiplication* $\odot$ such that addition is commutative, multiplication distributes over addition from either side, $\mathbf{0}$ (resp., $\mathbf{1}$) is the neutral element of addition (resp., multiplication), $\mathbf{0} \odot x = x \odot \mathbf{0} = \mathbf{0}$ for all $x \in S$, and $\mathbf{0} \neq \mathbf{1}$.

Let the semiring S be partially ordered by a relation $\preceq$ such that $\mathbf{0}$ is the least element and the inequality $x \preceq y$ implies that $x \oplus z \preceq y \oplus z, x \odot z \preceq y \odot z$, and $z \odot x \preceq z \odot y$ for all $x, y, z \in S$; in this case the semiring S is called *positive* (see, for instance, Golan 2000).

An element $x \in S$ is called *invertible* if there exists an element $x^{-1} \in S$ such that $xx^{-1} = x^{-1}x = \mathbf{1}$. A semiring S is called a *semifield* if every nonzero element is invertible.

A semiring S is called *idempotent* if $x \oplus x = x$ for all $x \in S$. In this case the addition $\oplus$ defines a *canonical partial order* $\preceq$ on the semiring S by the rule: $x \preceq y$ iff $x \oplus y = y$. It is easy to prove that any idempotent semiring is positive with respect to this order. Note also that $x \oplus y = \sup\{x, y\}$ with respect to the canonical order. In the sequel, we shall assume that all idempotent semirings are ordered by the canonical partial order relation.

We shall say that a positive (for instance, idempotent) semiring S is *complete* if for every subset $T \subset S$ there exist elements $\sup T \in S$ and $\inf T \in S$, and if the operations $\oplus$ and $\odot$ distribute over such sups and infs.

The most well-known and important examples of positive semirings are "numerical" semirings consisting of (a subset of) real numbers and ordered by the usual linear order $\leq$ on $\mathbf{R}$: the semiring $\mathbf{R}_+$ with the usual operations $\oplus = +$, $\odot = \cdot$ and neutral elements $\mathbf{0} = 0$, $\mathbf{1} = 1$, the semiring $\mathbf{R}_{\max} = \mathbf{R} \cup \{-\infty\}$ with the operations $\oplus = \max, \odot = +$ and neutral elements $\mathbf{0} = -\infty$, $\mathbf{1} = 0$, the semiring $\widehat{\mathbf{R}}_{\max} = \mathbf{R}_{\max} \cup \{\infty\}$, where $x \preceq \infty, x \oplus \infty = \infty$ for all $x, x \odot \infty = \infty \odot x = \infty$ if $x \neq \mathbf{0}$, and $\mathbf{0} \odot \infty = \infty \odot \mathbf{0}$, and the semiring $S^{[a,b]}_{\max,\min} = [a, b]$, where $-\infty \leq a < b \leq +\infty$, with the operations $\oplus = \max, \odot = \min$ and neutral elements $\mathbf{0} = a$, $\mathbf{1} = b$. The semirings $\mathbf{R}_{\max}, \widehat{\mathbf{R}}_{\max}$, and $S^{[a,b]}_{\max,\min} = [a, b]$ are idempotent. The semirings $\widehat{\mathbf{R}}_{\max}, S^{[a,b]}_{\max,\min}$, $\widehat{\mathbf{R}}_+ = \mathbf{R}_+ \bigcup \{\infty\}$ are complete. Remind that every partially ordered set can be imbedded to its completion (a minimal complete set containing the initial one). The semiring $\mathbf{R}_{\min} = \mathbf{R} \bigcup \{\infty\}$ with operations $\oplus = \min$ and $\odot = +$ and neutral elements $\mathbf{0} = \infty$, $\mathbf{1} = 0$ is isomorphic to $\mathbf{R}_{\max}$.

The semiring $\mathbf{R}_{\max}$ is also called the *max-plus algebra*. The semifields $\mathbf{R}_{\max}$ and $\mathbf{R}_{\min}$ are called *tropical algebras*. The term "tropical" initially appeared in Simon (1988) for a discrete version of the max-plus algebra as a suggestion of Choffrut, see also Gunawardena (1998), Mikhalkin (2006), Viro (2008).

Many mathematical constructions, notions, and results over the fields of real and complex numbers have nontrivial analogs over idempotent semirings. Idempotent semirings have become recently the object of investigation of new branches of mathematics, *idempotent mathematics* and *tropical geometry*, see, for instance Baccelli et al. (1992), Cuninghame-Green (1979), Litvinov (2007), Mikhalkin (2006), Viro (2001, 2008).

Denote by $\mathrm{Mat}_{mn}(S)$ a set of all matrices $A = (a_{ij})$ with m rows and n columns whose coefficients belong to a semiring S. The sum $A \oplus B$ of matrices $A, B \in \mathrm{Mat}_{mn}(S)$ and the product AB of matrices $A \in \mathrm{Mat}_{lm}(S)$ and $B \in$

$\mathrm{Mat}_{mn}(S)$ are defined according to the usual rules of linear algebra: $A \oplus B = (a_{ij} \oplus b_{ij}) \in \mathrm{Mat}_{mn}(S)$ and

$$AB = \left(\bigoplus_{k=1}^{m} a_{ij} \odot b_{kj} \right) \in \mathrm{Mat}_{ln}(S),$$

where $A \in \mathrm{Mat}_{lm}(S)$ and $B \in \mathrm{Mat}_{mn}(S)$. Note that we write AB instead of $A \odot B$.

If the semiring S is positive, then the set $\mathrm{Mat}_{mn}(S)$ is ordered by the relation $A = (a_{ij}) \preceq B = (b_{ij})$ iff $a_{ij} \preceq b_{ij}$ in S for all $1 \le i \le m$, $1 \le j \le n$.

The matrix multiplication is consistent with the order $\preceq$ in the following sense: if $A, A' \in \mathrm{Mat}_{lm}(S), B, B' \in \mathrm{Mat}_{mn}(S)$ and $A \preceq A', B \preceq B'$, then $AB \preceq A'B'$ in $\mathrm{Mat}_{ln}(S)$. The set $\mathrm{Mat}_{nn}(S)$ of square $(n \times n)$ matrices over a [positive, idempotent] semiring S forms a [positive, idempotent] semi-ring with a zero element $O = (o_{ij})$, where $o_{ij} = \mathbf{0}, 1 \le i, j \le n$, and a unit element $I = (\delta_{ij})$, where $\delta_{ij} = \mathbf{1}$ if $i = j$ and $\delta_{ij} = \mathbf{0}$ otherwise.

The set Mat_{nn} is an example of a noncommutative semiring if $n > 1$.

2.2 Closure operation

In what follows, we are mostly interested in complete positive semirings, and particularly in idempotent semirings. Regarding examples of the previous section, recall that the semirings $S^{[a,b]}_{\max,\min}$, $\widehat{\mathbf{R}}_{\max} = \mathbf{R}_{\max} \cup \{+\infty\}$, $\widehat{\mathbf{R}}_{\min} = \mathbf{R}_{\min} \cup \{-\infty\}$ and $\widehat{\mathbf{R}}_{+} = \mathbf{R}_{+} \cup \{+\infty\}$ are complete positive, and the semirings $S^{[a,b]}_{\max,\min}$, $\widehat{\mathbf{R}}_{\max}$ and $\widehat{\mathbf{R}}_{\min}$ are idempotent.

$\widehat{\mathbf{R}}_{+}$ is a completion of $\mathbf{R}_{+}$, and $\widehat{\mathbf{R}}_{\max}$ (resp. $\widehat{\mathbf{R}}_{\min}$) are completions of $\mathbf{R}_{\max}$ (resp. $\mathbf{R}_{\min}$). More generally, we note that any positive semifield S can be completed by means of a standard procedure, which uses Dedekind cuts and is described in Golan (2000), Litvinov et al. (2001). The result of this completion is a semiring $\widehat{S}$, which is not a semifield anymore.

The semiring of matrices $\mathrm{Mat}_{mn}(S)$ over a complete positive (resp., idempotent) semiring is again a complete positive (resp., idempotent) semiring. For more background in complete idempotent semirings, the reader is referred to Litvinov et al. (2001).

In any complete positive semiring S we have a unary operation of *closure* $a \mapsto a^*$ defined by

$$a^* := \sup_{k \ge 0} \mathbf{1} \oplus a \oplus \cdots \oplus a^k, \tag{1}$$

Using that the operations $\oplus$ and $\odot$ distribute over such sups, it can be shown that a^* is the **least solution** of $x = ax \oplus \mathbf{1}$ and $x = xa \oplus \mathbf{1}$, and also that a^*b is the the least solution of $x = ax \oplus b$ and $x = xa \oplus b$.

In the case of idempotent addition (1) becomes particularly nice:

$$a^* = \bigoplus_{i \ge 0} a^i = \sup_{i \ge 0} a^i. \tag{2}$$

If a positive semiring S is not complete, then it often happens that the closure operation can still be defined on some "essential" subset of S. Also recall that any positive semifield S can be completed (Golan 2000; Litvinov et al. 2001), and then the closure is defined for every element of the completion.

In numerical semirings the operation * is usually very easy to implement: $x^* = (1 - x)^{-1}$ if $x < 1$ in $\mathbf{R}_{+}$, or $\widehat{\mathbf{R}}_{+}$ and $x^* = \infty$ if $x \ge 1$ in $\widehat{\mathbf{R}}_{+}$; $x^* = \mathbf{1}$ if $x \preceq \mathbf{1}$ in $\mathbf{R}_{\max}$ and $\widehat{\mathbf{R}}_{\max}$, $x^* = \infty$ if $x \succ \mathbf{1}$ in $\widehat{\mathbf{R}}_{\max}$, $x^* = \mathbf{1}$ for all x in $S^{[a,b]}_{\max,\min}$. In all other cases x^* is undefined.

The closure operation in matrix semirings over a complete positive semiring S can be defined as in (1):

$$A^* := \sup_{k \ge 0} I \oplus A \oplus \cdots \oplus A^k, \tag{3}$$

and one can show that it is the least solution X satisfying the matrix equations $X = AX \oplus I$ and $X = XA \oplus I$.

Equivalently, A^* can be defined by induction: let $A^* = (a_{11})^* = (a_{11}^*)$ in $\mathrm{Mat}_{11}(S)$ be defined by (1), and for any integer $n > 1$ and any matrix

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix},$$

where $A_{11} \in \mathrm{Mat}_{kk}(S), A_{12} \in \mathrm{Mat}_{kn-k}(S), A_{21} \in \mathrm{Mat}_{n-kk}(S), A_{22} \in \mathrm{Mat}_{n-kn-k}(S), 1 \le k \le n$, by definition,

$$A^* = \begin{pmatrix} A_{11}^* \oplus A_{11}^* A_{12} D^* A_{21} A_{11}^* & A_{11}^* A_{12} D^* \\ D^* A_{21} A_{11}^* & D^* \end{pmatrix}, \tag{4}$$

where $D = A_{22} \oplus A_{21} A_{11}^* A_{12}$.

Defined here for complete positive semirings, the closure operation is a semiring analogue of the operation $(1 - a)^{-1}$ and, further, $(I - A)^{-1}$ in matrix algebra over the field of real or complex mumbers. This operation can be thought of as **regularized sum** of the series $I + A + A^2 + \cdots$, and the closure operation defined above is another such regularization. Thus we can also define the closure operation $a^* = (1 - a)^{-1}$ and $A^* = (I - A)^{-1}$ in the traditional linear algebra. To this end, note that the recurrence relation above coincides with the formulas of escalator method of matrix inversion in the traditional linear algebra over the field of real or complex numbers, up to the algebraic operations used. Hence this algorithm of matrix closure requires a polynomial number of operations in n, see below for more details.

Let S be a complete positive semiring. The *matrix (or discrete stationary) Bellman equation* has the form

$$X = AX \oplus B, \tag{5}$$

where $A \in \mathrm{Mat}_{nn}(S), X, B \in \mathrm{Mat}_{ns}(S)$, and the matrix X is unknown. As in the scalar case, it can be shown that for complete positive semirings, if A^* is defined as in (3) then A^*B is the least in the set of solutions to equation (5) with respect to the partial order in $\mathrm{Mat}_{ns}(S)$. In the idempotent case

$$A^* = \bigoplus_{i \geq 0} A^i = \sup_{i \geq 0} A^i. \tag{6}$$

Consider also the case when $A = (a_{ij})$ is $n \times n$ *strictly upper-triangular* (such that $a_{ij} = \mathbf{0}$ for $i \geq j$), or $n \times n$ *strictly lower-triangular* (such that $a_{ij} = \mathbf{0}$ for $i \leq j$). In this case $A^n = O$, the all-zeros matrix, and it can be shown by iterating $X = AX \oplus I$ that this equation has a unique solution, namely

$$A^* = I \oplus A \oplus \cdots \oplus A^{n-1}. \tag{7}$$

Curiously enough, formula (7) works more generally in the case of numerical idempotent semirings: in fact, the series (6) converges there if and only if it can be truncated to (7). This is closely related to the principal path interpretation of A^* explained in the next subsection.

In fact, theory of the discrete stationary Bellman equation can be developed using the identity $A^* = AA^* \oplus I$ as an axiom without any explicit formula for the closure (the so-called *closed semirings*, see, for instance, Golan 2000; Lehmann 1977; Rote 1985). Such theory can be based on the following identities, true both for the case of idempotent semirings and the real numbers with conventional arithmetic (assumed that A and B have appropriate sizes):

$$\begin{aligned} (A \oplus B)^* &= (A^*B)^*A^*, \\ (AB)^*A &= A(BA)^*. \end{aligned} \tag{8}$$

This abstract setting unites the case of positive and idempotent semirings with the conventional linear algebra over the field of real and complex numbers.

2.3 Weighted directed graphs and matrices over semirings

Suppose that S is a semiring with zero $\mathbf{0}$ and unity $\mathbf{1}$. It is well-known that any square matrix $A = (a_{ij}) \in \mathrm{Mat}_{nn}(S)$ specifies a *weighted directed graph*. This geometrical construction includes three kinds of objects: the set X of n elements $x_1, \ldots, x_n$ called *nodes*, the set Γ of all ordered pairs (x_i, x_j) such that $a_{ij} \neq \mathbf{0}$ called *arcs*, and the mapping $A\colon \Gamma \to S$ such that $A(x_i, x_j) = a_{ij}$. The elements a_{ij} of the semiring S are called *weights* of the arcs.Conversely, any given weighted directed graph with n nodes specifies a unique matrix $A \in \mathrm{Mat}_{nn}(S)$.

This definition allows for some pairs of nodes to be disconnected if the corresponding element of the matrix A is $\mathbf{0}$ and for some channels to be “loops” with coincident ends if the matrix A has nonzero diagonal elements.

Recall that a sequence of nodes of the form

$$p = (y_0, y_1, \ldots, y_k)$$

with $k \geq 0$ and $(y_i, y_{i+1}) \in \Gamma, i = 0, \ldots, k-1$, is called a *path* of length k connecting y_0 with y_k. Denote the set of all such paths by $P_k(y_0,y_k)$. The weight $A(p)$ of a path $p \in P_k(y_0, y_k)$ is defined to be the product of weights of arcs connecting consecutive nodes of the path:

$$A(p) = A(y_0, y_1) \odot \cdots \odot A(y_{k-1}, y_k).$$

By definition, for a ‘path’ $p \in P_0(x_i, x_j)$ of length $k = 0$ the weight is $\mathbf{1}$ if $i = j$ and $\mathbf{0}$ otherwise (Fig. 1).

For each matrix $A \in \mathrm{Mat}_{nn}(S)$ define $A^0 = E = (\delta_{ij})$ (where $\delta_{ij} = \mathbf{1}$ if $i = j$ and $\delta_{ij} = \mathbf{0}$ otherwise) and $A^k = AA^{k-1}$, $k \geq 1$. Let $a_{i,j}^{[k]}$ be the (i, j)th element of the matrix A^k. It is easily checked that

$$a_{ij}^{[k]} = \bigoplus_{\substack{i_0 = i, i_k = j \\ 1 \leq i_1, \ldots, i_{k-1} \leq n}} a_{i_0 i_1} \odot \cdots \odot a_{i_{k-1} i_k}.$$

Thus $a_{i,j}^{[k]}$ is the supremum of the set of weights corresponding to all paths of length k connecting the node $x_{i_0} = x_i$ with $x_{i_k} = x_j$.

Let A^* be defined as in (6). Denote the elements of the matrix A^* by $a_{i,j}^*$, $i, j = 1, \ldots, n$; then

$$a_{ij}^* = \bigoplus_{0 \leq k < \infty} \; \bigoplus_{p \in P_k(x_i, x_j)} A(p).$$

The closure matrix A^* solves the well-known *algebraic path problem*, which is formulated as follows: for each pair (x_i, x_j) calculate the supremum of weights of all paths (of arbitrary length) connecting node x_i with node x_j. The closure operation in matrix semirings has been studied extensively (see, for instance, Baccelli et al. 1992; Carré 1971; 1979; Cuninghame-Green 1979; Golan 2000; Gondran and Minoux 1979, 2010; Kolokoltsov and

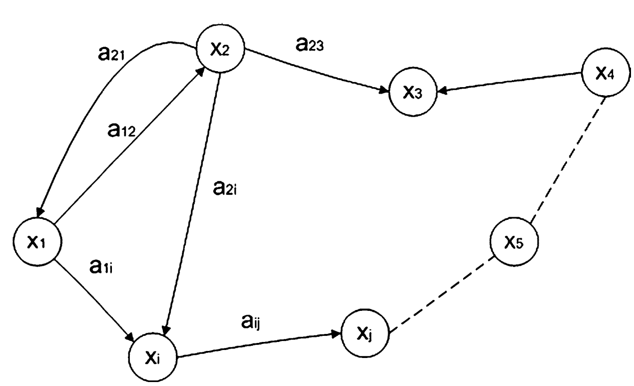


Fig. 1 Weighted directed graph

Maslov 1997; Litvinov and Sobolevskiĭ 2001 and references therein).

Example 1 (*The shortest path problem*) Let $S = \mathbf{R}_{\min}$, so the weights are real numbers. In this case

$$A(p) = A(y_0, y_1) + A(y_1, y_2) + \cdots + A(y_{k-1}, y_k).$$

If the element a_{ij} specifies the length of the arc (x_i, x_j) in some metric, then $a^*_{i,j}$ is the length of the shortest path connecting x_i with x_j.

Example 2 (*The maximal path width problem*) Let $S = \mathbf{R} \cup \{\mathbf{0}, \mathbf{1}\}$ with $\oplus = \max, \odot = \min$. Then

$$a^*_{ij} = \max_{p \in \bigcup_{k \geq 1} P_k(x_i, x_j)} A(p),$$

$$A(p) = \min(A(y_0, y_1), \ldots, A(y_{k-1}, y_k)).$$

If the element a_{ij} specifies the "width" of the arc (x_i, x_j), then the width of a path p is defined as the minimal width of its constituting arcs and the element a^*_{ij} gives the supremum of possible widths of all paths connecting x_i with x_j.

Example 3 (*A simple dynamic programming problem*) Let $S = \mathbf{R}_{\max}$ and suppose a_{ij} gives the *profit* corresponding to the transition from x_i to x_j. Define the vector $B = (b_i) \in \mathrm{Mat}_{n1}(\mathbf{R}_{\max})$ whose element b_i gives the *terminal profit* corresponding to exiting from the graph through the node x_i. Of course, negative profits (or, rather, losses) are allowed. Let m be the total profit corresponding to a path $p \in P_k(x_i, x_j)$, that is

$$m = A(p) + b_j.$$

Then it is easy to check that the supremum of profits that can be achieved on paths of length k beginning at the node x_i is equal to $(A^k B)_i$ and the supremum of profits achievable without a restriction on the length of a path equals $(A^* B)_i$.

Example 4 (*The matrix inversion problem*) Note that in the formulas of this section we are using distributivity of the multiplication $\odot$ with respect to the addition $\oplus$ but do not use the idempotency axiom. Thus the algebraic path problem can be posed for a nonidempotent semiring S as well (see, for instance, Rote 1985). For instance, if $S = \mathbf{R}$, then

$$A^* = I + A + A^2 + = (I - A)^{-1}.$$

If $\|A\| > 1$ but the matrix $I - A$ is invertible, then this expression defines a regularized sum of the divergent matrix power series $\sum_{i \geq 0} A^i$.

We emphasize that this connection between the matrix closure operation and solutions to the Bellman equation gives rise to a number of different algorithms for numerical calculation of the matrix closure. All these algorithms are adaptations of the well-known algorithms of the traditional computational linear algebra, such as the Gauss–Jordan elimination, various iterative and escalator schemes, etc. This is a special case of the idempotent superposition principle (see below).

2.4 Interval analysis over positive semirings

Traditional interval analysis is a nontrivial and popular mathematical area, see, for instance, Alefeld and Herzberger (1983), Fiedler et al. (2006), Kreinovich et al. (1998), Moore (1979), Neumaier (1990). An "idempotent" version of interval analysis (and moreover interval analysis over positive semirings) appeared in Litvinov and Sobolevskiĭ (2000, 2001), Sobolevskiĭ (1999). Rather many publications on the subject appeared later, see, for instance, Cechlárová and Cuninghame-Green (2002), Fiedler et al. (2006), Hardouin et al. (2009), Myšková (2005, 2006). Interval analysis over the positive semiring $\mathbf{R}_+$ was discussed in Barth and Nuding (1974).

Let a set S be partially ordered by a relation $\preceq$. A *closed interval* in S is a subset of the form $\mathbf{x} = [\underline{\mathbf{x}}, \overline{\mathbf{x}}] = \{x \in S \mid \underline{\mathbf{x}} \preceq x \preceq \overline{\mathbf{x}}\}$, where the elements $\underline{\mathbf{x}} \preceq \overline{\mathbf{x}}$ are called *lower* and *upper bounds* of the interval $\mathbf{x}$. The order $\preceq$ induces a partial ordering on the set of all closed intervals in S : $\mathbf{x} \preceq \mathbf{y}$ iff $\underline{\mathbf{x}} \preceq \underline{\mathbf{y}}$ and $\overline{\mathbf{x}} \preceq \overline{\mathbf{y}}$.

A *weak interval extension* $I(S)$ of a positive semiring S is the set of all closed intervals in S endowed with operations $\oplus$ and $\odot$ defined as $\mathbf{x} \oplus \mathbf{y} = [\underline{\mathbf{x}} \oplus \underline{\mathbf{y}}, \overline{\mathbf{x}} \oplus \overline{\mathbf{y}}], \mathbf{x} \odot \mathbf{y} = [\underline{\mathbf{x}} \odot \underline{\mathbf{y}}, \overline{\mathbf{x}} \odot \overline{\mathbf{y}}]$ and a partial order induced by the order in S. The closure operation in $I(S)$ is defined by $\mathbf{x}^* = [\underline{\mathbf{x}}^*, \overline{\mathbf{x}}^*]$. There are some other interval extensions (including the so-called strong interval extension (Litvinov and Sobolevskiĭ 2001)) but the weak extension is more convenient.

The extension $I(S)$ is positive; $I(S)$ is idempotent if S is an idempotent semiring. A universal algorithm over S can be applied to $I(S)$ and we shall get an interval version of the initial algorithm. Usually both versions have the same complexity. For the discrete stationary Bellman equation and the corresponding optimization problems on graphs, interval analysis was examined in Litvinov and Sobolevskiĭ (2000, 2001) in details. Other problems of idempotent linear algebra were examined in Cechlárová and Cuninghame-Green (2002), Fiedler et al. (2006), Hardouin et al. (2009), Myšková (2005, 2006).

Idempotent mathematics appears to be remarkably simpler than its traditional analog. For example, in traditional interval arithmetic, multiplication of intervals is not distributive with respect to addition of intervals, whereas in idempotent interval arithmetic this distributivity is preserved. Moreover, in traditional interval analysis the set of all square interval matrices of a given order does not form even a semigroup with respect to matrix multiplication: this operation is not associative since distributivity is lost in the traditional interval arithmetic. On the contrary, in the idempotent (and positive) case associativity is preserved. Finally, in traditional interval analysis some problems of linear algebra, such as solution of a linear system of interval equations, can be very difficult (more precisely, they are *NP*-hard, see Kreinovich et al. 1998 and references therein). It was noticed in Litvinov and Sobolevskiĭ (2000, 2001) that in the idempotent case solving an interval linear system requires a polynomial number of operations (similarly to the usual Gauss elimination algorithm). Two properties that make the idempotent interval arithmetic so simple are monotonicity of arithmetic operations and positivity of all elements of an idempotent semiring.

Interval estimates in idempotent mathematics are usually exact. In the traditional theory such estimates tend to be overly pessimistic.

2.5 Idempotent correspondence principle

There is a nontrivial analogy between mathematics of semirings and quantum mechanics. For example, the field of real numbers can be treated as a "quantum object" with respect to idempotent semirings. So idempotent semirings can be treated as "classical" or "semi-classical" objects with respect to the field of real numbers.

Let $\mathbf{R}$ be the field of real numbers and $\mathbf{R}_+$ the subset of all non-negative numbers. Consider the following change of variables:

$$u \mapsto w = h \ln u,$$

where $u \in \mathbf{R}_+ \setminus \{0\}, h > 0$; thus $u = e^{w/h}, w \in \mathbf{R}$. Denote by $\mathbf{0}$ the additional element $-\infty$ and by S the extended real line $\mathbf{R} \cup \{\mathbf{0}\}$. The above change of variables has a natural extension D_h to the whole S by $D_h(0) = \mathbf{0}$; also, we denote $D_h(1) = 0 = \mathbf{1}$.

Denote by S_h the set S equipped with the two operations $\oplus_h$ (generalized addition) and $\odot_h$ (generalized multiplication) such that D_h is a homomorphism of $\{\mathbf{R}_+, +, \cdot\}$ to $\{S, \oplus_h, \odot_h\}$. This means that $D_h(u_1 + u_2) = D_h(u_1) \oplus_h D_h(u_2)$ and $D_h(u_1 \cdot u_2) = D_h(u_1) \odot_h D_h(u_2)$, that is, $w_1 \odot_h w_2 = w_1 + w_2$ and $w_1 \oplus_h w_2 = h \ln(e^{w_1/h} + e^{w_2/h})$. It is easy to prove that $w_1 \oplus_h w_2 \to \max\{w_1, w_2\}$ as $h \to 0$.

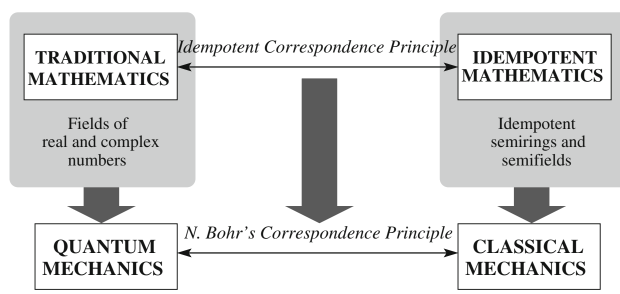
$\mathbf{R}_+$ and S_h are isomorphic semirings; therefore we have obtained $\mathbf{R}_{\max}$ as a result of a deformation of $\mathbf{R}_+$. We stress the obvious analogy with the quantization procedure, where h is the analog of the Planck constant. In these terms, $\mathbf{R}_+$ (or $\mathbf{R}$) plays the part of a "quantum object" while $\mathbf{R}_{\max}$ acts as a "classical" or "semi-classical" object that arises as the result of a *dequantization* of this quantum object. In the case of $\mathbf{R}_{\min}$, the corresponding dequantization procedure is generated by the change of variables $u \mapsto w = -h \ln u$.

There is a natural transition from the field of real numbers or complex numbers to the idempotent semiring $\mathbf{R}_{\max}$ (or $\mathbf{R}_{\min}$). This is a composition of the mapping $x \mapsto |x|$ and the deformation described above.

In general an *idempotent dequantization* is a transition from a basic field to an idempotent semiring in mathematical concepts, constructions and results, see Litvinov (2007), Litvinov and Maslov (1998) for details. Idempotent dequantization suggests the following formulation of the idempotent correspondence principle:

> *There exists a heuristic correspondence between interesting, useful, and important constructions and results over the field of real (or complex) numbers and similar constructions and results over idempotent semirings in the spirit of N. Bohr's correspondence principle in quantum mechanics.*

Thus idempotent mathematics can be treated as a "classical shadow (or counterpart)" of the traditional Mathematics over fields. A systematic application of this correspondence principle leads to a variety of theoretical and applied results, see, for instance, Litvinov (2007), Litvinov and Maslov (1998, 2000), Litvinov and Sobolevskiĭ (2001), Mikhalkin (2006), Viro (2001, 2008). Relations to quantum physics are discussed in detail, for instance, in Litvinov (2007).



In this paper we aim to develop a practical systematic application of the correspondence principle to the algorithms of linear algebra and discrete mathematics. For the remainder of this subsection let us focus on an idea how the idempotent correspondence principle may lead to a unifying approach to hardware design. (See Litvinov et al. 2000, 2011 for more information.)

The most important and standard numerical algorithms have many hardware realizations in the form of technical devices or special processors. These devices often can be used as prototypes for new hardware units resulting from mere substitution of the usual arithmetic operations by their semiring analogs (and additional tools for generating neutral elements $\mathbf{0}$ and $\mathbf{1}$). Of course, the case of numerical semirings consisting of real numbers (maybe except neutral elements) and semirings of numerical intervals is the most simple and natural. Note that for semifields (including $\mathbf{R}_{\max}$ and $\mathbf{R}_{\min}$) the operation of division is also defined.

Good and efficient technical ideas and decisions can be taken from prototypes to new hardware units. Thus the correspondence principle generates a regular heuristic method for hardware design. Note that to get a patent it is necessary to present the so-called 'invention formula', that is to indicate a prototype for the suggested device and the difference between these devices.

Consider (as a typical example) the most popular and important algorithm of computing the scalar product of two vectors:

$$(x, y) = x_1y_1 + x_2y_2 + \cdots + x_ny_n. \tag{9}$$

The universal version of (9) for any semiring A is obvious:

$$(x, y) = (x_1 \odot y_1) \oplus (x_2 \odot y_2) \oplus \cdots \oplus (x_n \odot y_n). \tag{10}$$

In the case $A = \mathbf{R}_{\max}$ this formula turns into the following one:

$$(x, y) = \max\{x_1 + y_1, x_2 + y_2, \ldots, x_n + y_n\}. \tag{11}$$

This calculation is standard for many optimization algorithms, so it is useful to construct a hardware unit for computing (11). There are many different devices (and patents) for computing (9) and every such device can be used as a prototype to construct a new device for computing (11) and even (10). Many processors for matrix multiplication and for other algorithms of linear algebra are based on computing scalar products and on the corresponding "elementary" devices. Using modern technologies it is possible to construct cheap special-purpose multi-processor chips and systolic arrays of elementary processors implementing universal algorithms. See, for instance, Litvinov et al. (2000, 2011), Rote (1985) where the systolic arrays and parallel computing issues are discussed for the algebraic path problem. In particular, there is a systolic array of $n(n + 1)$ elementary processors which performs computations of the Gauss–Jordan elimination algorithm and can solve the algebraic path problem within $5n - 2$ time steps.

3 Some universal algorithms of linear algebra

In this section we discuss universal algorithms computing A^* and A^*B. We start with the basic escalator and Gauss–Jordan elimination techniques in Sect. 3.1 and continue with its specification to the case of Toeplitz systems in Sect. 3.2. The universal LDM decomposition of Bellman equations is explained in Sect. 3.3, followed by its adaptations to symmetric and band matrices in Sect. 3.4. The iteration schemes are discussed in Sect. 3.5. In the final Sect. 3.6 we discuss the implementations of universal algorithms.

Algorithms themselves will be described in a language of Matlab, following the tradition of Golub and van Loan (1989). This is done for two purposes: (1) to simplify the comparison of the algorithms with their prototypes taken mostly from (Golub and van Loan 1989), (2) since the language of Matlab is designed for matrix computations. We will not formally describe the rules of our Matlab-derived language, preferring just to outline the following important features:

1. Our basic arithmetic operations are $a \oplus b, a \odot b$ and a^*.
2. The vectorization of these operations follows the rules of Matlab.
3. We use basic keywords of Matlab like 'for', 'while', 'if' and 'end', similar to other programming languages like *C++* or Java.

Let us give some examples of universal matrix computations in our language:

Example 1 $v(1:j) = \alpha^* \odot a(1:j,k)$ means that the result of (scalar) multiplication of the first j components of the kth column of A by the closure of α is assigned to the first j components of v.

Example 2 $a(i,j) = a(i,j) \oplus a(i,1:n) \odot a(1:n,j)$ means that we add ($\oplus$) to the entry a_{ij} of A the result of the (universal) scalar multiplication of the ith row with the jth column of A (assumed that A is $n \times n$).

Example 3 $a(1:n,k) \odot b(l,1:m)$ means the outer product of the kth column of A with the lth row of B. The entries of resulting matrix $C = (c_{ij})$ equal $c_{ij} = a_{ik} \odot b_{lj}$, for all $i = 1,\ldots,n$ and $j = 1,\ldots,m$.

Example 4 $x(1:n) \odot y(n:-1:1)$ is the scalar product of vector x with vector y whose components are taken in the reverse order: the proper algebraic expression is $\bigoplus_{i=1}^{n} x_i \odot y_{n+1-i}$.

Example 5 The following cycle yields the same result as in the previous example: $s = 0$

for $i = 1{:}n$
 $s = s \oplus x(i) \odot x(n+1-i)$
 end

3.1 Escalator scheme and Gauss–Jordan elimination

We first analyse the basic escalator method, based on the definition of matrix closures (4). Let A be a square matrix. Closures of its main submatrices A_k can be found inductively, starting from $A_1^* = (a_{11})^*$, the closure of the first diagonal entry. Generally we represent A_{k+1} as

$$A_{k+1} = \begin{pmatrix} A_k & g_k \\ h_k^T & a_{k+1} \end{pmatrix},$$

assuming that we have found the closure of A_k. In this representation, g_k and h_k are columns with k entries and a_{k+1} is a scalar. We also represent A^*_{k+1} as

$$A^*_{k+1} = \begin{pmatrix} U_k & v_k \\ w_k^T & u_{k+1} \end{pmatrix}.$$

Using (4) we obtain that

$$\begin{aligned} u_{k+1} &= (h_k^T A_k^* g_k \oplus a_{k+1})^*, \\ v_k &= A_k^* g_k u_{k+1}, \\ w_k^T &= u_{k+1} h_k^T A_k^*, \\ U_k &= A_k^* g_k u_{k+1} h_k^T A_k^* \oplus A_k^*. \end{aligned} \tag{12}$$

An algorithm based on (12) can be written as follows.

Algorithm 1 *Escalator method for computing A^**

Input: an $n \times n$ matrix A with entries $a(i,j)$,
also used to store the final result
and the intermediate results of the computation process.

$a(1,1) = (a(1,1))^*$
for $i = 1 : n-1$
$Ag = a(1:i, 1:i) \odot a(1:i, i+1)$
$hA = a(i+1, 1:i) \odot a(1:i, 1:i)$
$a(i+1,i+1) = a(i+1,i+1) \oplus a(i+1,1:i) \odot Ag(1:i,1)$
$a(i+1,i+1) = (a(i+1,i+1))^*$
$a(1:i,i+1) = a(i+1,i+1) \odot Ag$
$a(i+1,1:i) = a(i+1,i+1) \odot hA$
$a(1:i,1:i) = a(1:i,1:i) \oplus Ag \odot a(i+1,i+1) \odot hA$
end

In full analogy with its linear algebraic prototype, the algorithm requires $n^3 + O(n^2)$ operations of addition $\oplus, n^3 + O(n^2)$ operations of multiplication $\odot$, and n operations of taking algebraic closure. The linear-algebraic prototype of the method written above is also called the *bordering method* in the literature (Carré 1971; Faddeev and Faddeeva 2002).

Alternatively, we can obtain a solution of $X = AX \oplus B$ as a result of elimination process, whose informal explanation is given below. If A^* is defined as $\bigoplus_{i \geq 0} A^i$ (including the scalar case), then A^*B is the least solution of $X = AX \oplus B$ for all A and B of appropriate sizes. In this case, the solution found by the elimination process given below coincides with A^*B.

For matrix $A = (a_{ij})$ and column vectors $x = (x_i)$ and $b = (b_i)$ (restricting without loss of generality to the column vectors), the Bellman equation $x = Ax \oplus b$ can be written as

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \oplus \begin{pmatrix} \mathbf{1} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{1} \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}.$$

After expressing x_1 in terms of $x_2, \dots, x_n$ from the first equation and substituting this expression for x_1 in all other equations from the second to the nth we obtain

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} \mathbf{0} & (a_{11})^* a_{12} & \dots & (a_{11})^* a_{1n} \\ \mathbf{0} & a_{22} \oplus (a_{21}(a_{11})^* a_{12}) & \dots & a_{2n} \oplus (a_{21}(a_{11})^* a_{1n}) \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & a_{n2} \oplus (a_{n1}(a_{11})^* a_{12}) & \dots & a_{nn} \oplus (a_{n1}(a_{11})^* a_{1n}) \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \oplus \begin{pmatrix} (a_{11})^* & \mathbf{0} & \dots & \mathbf{0} \\ a_{21}(a_{11})^* & \mathbf{1} & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1}(a_{11})^* & \mathbf{0} & \dots & \mathbf{1} \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \tag{14}$$

Note that nontrivial entries in both matrices occupy complementary places, so during computations both matrices can be stored in the same square array $C^{(k)}$. Denote its elements by $c_{ij}^{(c)}$ where k is the number of eliminated variables. After $l-1$ eliminations we have

$$\begin{aligned} x_l &= \left(c_{ll}^{(l-1)}\right)^* b_l, \\ c_{il}^{(l)} &= c_{il}^{(l-1)} \left(c_{ll}^{(l-1)}\right)^*, \quad i = 1, \dots, l-1, l+1, \dots, n \\ c_{ij}^{(l)} &= c_{ij}^{(l-1)} \oplus c_{il}^{(l-1)} \left(c_{ll}^{(l-1)}\right)^* c_{lj}^{(l-1)}, \\ i,j &= 1, \dots, l-1, l+1, \dots, n \\ c_{li}^{(l)} &= \left(c_{ll}^{(l-1)}\right)^* c_{li}^{(l-1)}, \quad i = 1, \dots, l-1, l+1, \dots, n \end{aligned} \tag{15}$$

After n eliminations we get $x = C^{(n)} b$. Taking as b any vector with one coordinate equal to $\mathbf{1}$ and the rest equal to $\mathbf{0}$, we obtain $C^{(n)} = A^*$. We write out the following algorithm based on recursion (15).

Algorithm 2 *Gauss-Jordan elimination for computing A^*.*
Input: an $n \times n$ matrix A with entries $a(i,j)$,
also used to store the final result
and intermediate results of the computation process.

```
for i = 1 : n
a(i,i) = (a(i,i))*
for k = 1 : n
if k ≠ i
a(k,i) = a(k,i) ⊙ a(i,i)
end
end
for k = 1 : n
for j = 1 : n
if k ≠ i & j ≠ i
a(k,j) = a(k,j) ⊕ a(k,i) ⊙ a(i,j)
end
end
for j = 1 : n
if j ≠ i
a(i,j) = a(i,i) ⊙ a(i,j)
end
end
end
```

Remark 1 Algorithm 2 can be regarded as a "universal Floyd–Warshall algorithm" generalizing the well-known algorithms of Warshall and Floyd for computing the transitive closure of a graph and all optimal paths on a graph. See, for instance, Sedgewick (2002). for the description of these classical methods of discrete mathematics. In turn, these methods can be regarded as specifications of Algorithm 2 to the cases of max-plus and Boolean semiring.

Remark 2 Algorithm 2 is also close to Yershov's "refilling" method for inverting matrices and solving systems $Ax = b$ in the classical linear algebra, see Faddeev and Faddeeva (2002) Chapter 2 for details.

3.2 Toeplitz systems

We start by considering the escalator method for finding the solution $x = A^*b$ to $x = Ax \oplus b$, where x and b are column vectors. Firstly, we have $x^{(1)} = A_1^*b_1$. Let $x^{(k)}$ be the vector found after $(k-1)$ steps, and let us write

$$x^{(k+1)} = \begin{pmatrix} z \\ x_{k+1} \end{pmatrix}.$$

Using (12) we obtain that

$$\begin{aligned} x_{k+1} &= u_{k+1}(h_k^T x^{(k)} \oplus b_{k+1}), \\ z &= x^{(k)} \oplus A_k^* g_k x_{k+1}. \end{aligned} \tag{16}$$

We have to compute $A_k^* g_k$. In general, we would have to use Algorithm 1. Next we show that this calculation can be done very efficiently when A is symmetric Toeplitz.

Formally, a matrix $A \in \mathrm{Mat}_{nn}(\mathcal{S})$ is called *Toeplitz* if there exist scalars $r_{-n+1}, \ldots, r_0, \ldots, r_{n-1}$ such that $A_{ij} = r_{j-i}$ for all i and j. Informally, Toeplitz matrices are such that their entries are constant along any line parallel to the main diagonal (and along the main diagonal itself). For example,

$$A = \begin{pmatrix} r_0 & r_1 & r_2 & r_3 \\ r_{-1} & r_0 & r_1 & r_2 \\ r_{-2} & r_{-1} & r_0 & r_1 \\ r_{-3} & r_{-2} & r_{-1} & r_0 \end{pmatrix}$$

is Toeplitz. Such matrices are not necessarily symmetric. However, they are always *persymmetric*, that is, symmetric with respect to the inverse diagonal. This property is algebraically expressed as $A = E_n A^T E_n$, where $E_n = [e_n, \ldots, e_1]$. By e_i we denote the column whose ith entry is $\mathbf{1}$ and other entries are O. The property $E_n^2 = I_n$ (where I_n is the $n \times n$ identity matrix) implies that the product of two persymmetric matrices is persymmetric. Hence any degree of a persymmetric matrix is persymmetric, and so is the closure of a persymmetric matrix. Thus, if A is persymmetric, then

$$E_n A^* = (A^*)^T E_n. \tag{17}$$

Further we deal only with symmetric Toeplitz matrices. Consider the equation $y = T_n y \oplus r^{(n)}$, where $r^{(n)} = (r_1, \ldots, r_n)^T$, and T_n is defined by the scalars $r_0, r_1, \ldots, r_{n-1}$ so that $T_{ij} = r_{|j-i|}$ for all i and j. This is a generalization of the Yule-Walker problem (Golub and van Loan 1989). Assume that we have obtained the least solution $y^{(k)}$ to the system $y = T_k y \oplus r^{(k)}$ for some k such that $1 \le k \le n-1$, where T_k is the main $k \times k$ submatrix of T_n. We write T_{k+1} as

$$T_{k+1} = \begin{pmatrix} T_k & E_k r^{(k)} \\ r^{(k)T} E_k & r_0 \end{pmatrix}.$$

We also write $y^{(k+1)}$ and $r^{(k+1)}$ as

$$y^{(k+1)} = \begin{pmatrix} z \\ \alpha_k \end{pmatrix}, \quad r^{(k+1)} = \begin{pmatrix} r^{(k)} \\ r_{k+1} \end{pmatrix}.$$

Using (16, 17) and the identity $T_k^* r^{(k)} = y^{(k)}$, we obtain that

$$\begin{aligned} \alpha_k &= (r_0 \oplus r^{(k)T} y^{(k)})^* (r^{(k)T} E_k y^{(k)} \oplus r_{k+1}), \\ z &= E_k y^{(k)} \alpha_k \oplus y^{(k)}. \end{aligned}$$

Denote $\beta_k = r_0 \oplus r^{(k)T} y^{(k)}$. The following argument shows that β_k can be found recursively if $(\beta_{k-1}^*)^{-1}$ exists.

$$\beta_k = r_0 \oplus [r^{(k-1)T}\ \ r_k]\begin{pmatrix} E_{k-1}y^{(k-1)}\alpha_{k-1} \oplus y^{(k-1)} \\ \alpha_{k-1} \end{pmatrix}$$
$$= r_0 \oplus r^{(k-1)T}y^{(k-1)} \oplus (r^{(k-1)T}E_{k-1}y^{(k-1)} \oplus r_k)\alpha_{k-1}$$
$$= \beta_{k-1} \oplus (\beta^*_{k-1})^{-1} \odot \alpha^2_{k-1}. \tag{18}$$

Existence of $(\beta^*_{k-1})^{-1}$ is not universal, and this will make us write two versions of our algorithm, the first one involving (18), and the second one not involving it. We will write these two versions in one program and mark the expressions which refer only to the first version or to the second one by the MATLAB-style comments % 1 and % 2, respectively. Collecting the expressions for β_k, α_k and z we obtain the following recursive expression for $y^{(k)}$:

$$\begin{aligned} \beta_k &= r_0 \oplus r^{(k)T}y^{(k)}, \quad \%2 \\ \beta_k &= \beta_{k-1} \oplus (\beta^*_{k-1})^{-1} \odot \alpha^2_{k-1}, \quad \%1 \\ \alpha_k &= (\beta_k)^* \odot (r^{(k)T}E_k y^{(k)} \oplus r_{k+1}), \\ y^{(k+1)} &= \begin{pmatrix} E_k y^{(k)}\alpha_k \oplus y^{(k)} \\ \alpha_k \end{pmatrix}. \end{aligned} \tag{19}$$

Recursive expression (19) is a generalized version of the Durbin method for the Yule-Walker problem, see Golub and van Loan (1989) Algorithm 4.7.1 for a prototype.

Algorithm 3 *The Yule-Walker problem for the Bellman equations with symmetric Toeplitz matrix.*

Input: r_0: scalar,
r: $n-1 \times 1$ vector;

$y(1) = r_0^* \odot r(1)$
$\beta = r_0$ %1
$\alpha = r_0^* \odot r(1)$
for $k = 1 : n-1$
$\beta = r_0 \oplus r(1:k) \odot y(1:k)$ %2
$\beta = \beta \oplus (\beta^*)^{-1} \odot \alpha^2$ %1
$\alpha = \beta^* \odot (r(k:-1:1) \odot y(1:k) \oplus r(k+1))$
$z(1:k) = y(1:k) \oplus y(k:-1:1) \odot \alpha$
$y(1:k) = z(1:k)$
$y(k+1) = \alpha$
end

Output: vector y.

In the general case, the algorithm requires 3/2 $n^2 + O(n)$ operations $\oplus$ and $\odot$ each, and just $n^2 + O(n)$ of $\oplus$ and $\odot$ if inversions of algebraic closures are allowed (as usual, just n such closures are required in both cases).

Now we consider the problem of finding $x^{(n)} = T_n^* b^{(n)}$ where T_n is as above and $b^{(n)} = (b_1, \ldots, b_n)$ is arbitrary. We also introduce the column vectors $y^{(k)}$ which solve the Yule-Walker problem: $y^{(k)} = T_k^* r^{(k)}$. The main idea is to find the expression for $x^{(k+1)} = T^*_{k+1}b^{(k+1)}$ involving $x^{(k)}$ and $y^{(k)}$. We write $x^{(k+1)}$ and $b^{(k+1)}$ as

$$x^{(k+1)} = \begin{pmatrix} v \\ \mu_k \end{pmatrix}, \quad b^{(k+1)} = \begin{pmatrix} b^{(k)} \\ b_{k+1} \end{pmatrix}.$$

Making use of the persymmetry of T_k^* and of the identities $T_k^* b_k = x^{(k)}$ and $T_k^* r_k = y^{(k)}$, we specialize expressions (16) and obtain that

$$\mu_k = (r_0 \oplus r^{(k)T}y^{(k)})^* \odot (r^{(k)T}E_k x^{(k)} \oplus b_{k+1}),$$
$$v = E_k y^{(k)}\mu_k \oplus x^{(k)}.$$

The coefficient $r_0 \oplus r^{(k)T}y^{(k)} = \beta_k$ can be expressed again as $\beta_k = \beta_{k-1} \oplus (\beta^*_{k-1})^{-1} \odot (\alpha_{k-1})^2$, if the closure $(\beta_{k-1})^*$ is invertible. Using this we obtain the following recursive expression:

$$\begin{aligned} \beta_k &= r_0 \oplus r^{(k)T}y^{(k)}, \quad \%2 \\ \beta_k &= \beta_{k-1} \oplus (\beta^*_{k-1})^{-1} \odot \alpha^2_{k-1}, \quad \%1 \\ \mu_k &= \beta_k^* \odot (r^{(k)T}E_k x^{(k)} \oplus b_{k+1}), \\ x^{(k+1)} &= \begin{pmatrix} E_k y^{(k)}\mu_k \oplus x^{(k)} \\ \mu_k \end{pmatrix}. \end{aligned} \tag{20}$$

Algorithm 4 *Bellman system with symmetric Toeplitz matrix*

Input: r_0: scalar,
r: $1 \times n-1$ row vector;
b: $n \times 1$ column vector.

$y(1) = r_0^* \odot r(1);\quad x(1) = r_0^* \odot b(1);$
$\beta = r_0$ %1
$\alpha = r_0^* \odot r(1)$
for $k = 1 : n-1$
$\beta = r_0 \oplus r(1:k) \odot y(1:k)$ %2
$\beta = \beta \oplus (\beta^*)^{-1} \odot \alpha^2$ %1
$\mu = \beta^* \odot (r(k:-1:1) \odot x(1:k) \oplus b(k+1))$
$v(1:k) = x(1:k) \oplus y(k:-1:1) \odot \mu$
$x(1:k) = v(1:k)$
$x(k+1) = \mu$
if $k < n-1$
$\alpha = \beta^* \odot (r(k:-1:1) \odot y(1:k) \oplus r(k+1))$
$z(1:k) = y(1:k) \oplus y(k:-1:1) \odot \alpha$
$y(1:k) = z(1:k)$
$y(k+1) = \alpha$
end
end

Output: vector x.

Expressions (19) and (20) yield the following generalized version of the Levinson algorithm for solving linear symmetric Toeplitz systems, see Golub and van Loan (1989) Algorithm 4.7.2 for a prototype:

In the general case, the algorithm requires 5/2 $n^2 + O(n)$ operations $\oplus$ and $\odot$ each, and just $2n^2 + O(n)$ of $\oplus$ and $\odot$ if inversions of algebraic closures are allowed (as usual, just n such closures are required in both cases).

3.3 LDM decomposition

Factorization of a matrix into the product $A = LDM$, where L and M are lower and upper triangular matrices with a unit diagonal, respectively, and D is a diagonal matrix, is used for solving matrix equations $AX = B$. We construct a similar decomposition for the Bellman equation $X = AX \oplus B$.

For the case $AX = B$, the decomposition $A = LDM$ induces the following decomposition of the initial equation:

$$LZ = B, \quad DY = Z, \quad MX = Y. \tag{21}$$

Hence, we have

$$A^{-1} = M^{-1}D^{-1}L^{-1}, \tag{22}$$

if A is invertible. In essence, it is sufficient to find the matrices L, D and M, since the linear system $AX = B$ is easily solved by a combination of the forward substitution for Z, the trivial inversion of a diagonal matrix for Y, and the back substitution for X.

Using the LDM-factorization of $AX = B$ as a prototype, we can write

$$Z = LZ \oplus B, \quad Y = DY \oplus Z, \quad X = MX \oplus Y. \tag{23}$$

Then

$$A^* = M^*D^*L^*. \tag{24}$$

A triple (L, D, M) consisting of a lower triangular, diagonal, and upper triangular matrices is called an *LDM-factorization* of a matrix A if relations (23) and (24) are satisfied. We note that in this case, the principal diagonals of L and M are zero.

Our universal modification of the *LDM*-factorization used in matrix analysis for the equation $AX = B$ is similar to the *LU*-factorization of Bellman equation suggested by Carré (1971, 1979).

If A is a symmetric matrix over a semiring with a commutative multiplication, the amount of computations can be halved, since M and L are mapped into each other under transposition.

We begin with the case of a triangular matrix $A = L$ (or $A = M$). Then, finding X is reduced to the forward (or back) substitution. Note that in this case, equation $X = AX \oplus B$ has unique solution, which can be found by the obvious algorithms given below. In these algorithms B is a vector (denoted by b), however they could be modified to the case when B is a matrix of any appropriate size. We are interested only in the case of strictly lower-triangular, resp. strictly upper-triangular matrices, when $a_{ij} = 0$ for $i \leq j$, resp. $a_{ij} = 0$ for $i \geq j$.

Algorithm 5 *Forward substitution.*

Input: Strictly lower-triangular $n \times n$ matrix l;
$n \times 1$ vector b.

for $k = 2 : n$
$y(k) = l(k, 1 : k-1) \odot y(1 : k-1)$
end

Output: vector y.

Algorithm 6 *Backward substitution.*

Input: Strictly upper-triangular $n \times n$ matrix m;
$n \times 1$ vector b.

for $k = n-1 : -1 : 1$
$y(k) = m(k, k+1 : n) \odot y(k+1 : n)$
end

Output: vector y.

Both algorithms require $n^2/2 + O(n)$ operations $\oplus$ and $\odot$, and no algebraic closures.

After performing a LDM-decomposition we also need to compute the closure of a diagonal matrix: this is done entrywise.

We now proceed with the algorithm of LDM decomposition itself, that is, computing matrices L, D and M satisfying (23) and (24). First we give an algorithm, and then we proceed with its explanation.

Algorithm 7 *LDM-decomposition (version 1).*

Input: an $n \times n$ matrix A with entries $a(i, j)$,
also used to store the final result
and intermediate results of the computation process.

for $j = 1 : n-1$
$v(j) = (a(j, j))^*$
$a(j+1 : n, j) = a(j+1 : n, j) \odot v(j)$
$a(j+1 : n, j+1 : n) = a(j+1 : n, j+1 : n)$
$\oplus\, a(j+1 : n, j) \odot a(j, j+1 : n)$
$a(j, j+1 : n) = v(j) \odot a(j, j+1 : n)$
end

The algorithm requires $n^3/3 + O(n^2)$ operations $\oplus$ and $\odot$, and $n-1$ operations of algebraic closure.

The strictly triangular matrix L is written in the lower triangle, the strictly upper triangular matrix M in the upper triangle, and the diagonal matrix D on the diagonal of the matrix computed by Algorithm 7. We now show that $A^* = M^*D^*L^*$. Our argument is close to that of Backhouse and Carré (1975).

We begin by representing, in analogy with the escalator method,

$$A = \begin{pmatrix} a_{11} & h^{(1)} \\ g^{(1)} & B^{(1)} \end{pmatrix} \tag{25}$$

It can be verified that

$$A^* = \begin{pmatrix} \mathbf{1} & h^{(1)}a_{11}^* \\ O_{n-1\times 1} & I_{n-1} \end{pmatrix} \odot \begin{pmatrix} a_{11}^* & O_{1\times n-1} \\ O_{n-1\times 1} & (h^{(1)}a_{11}^* g^{(1)} \oplus B^{(1)})^* \end{pmatrix} \begin{pmatrix} \mathbf{1} & O_{1\times n-1} \\ a_{11}^* g^{(1)} & I_{n-1} \end{pmatrix} \tag{26}$$

as the multiplication on the right hand side leads to expressions fully analogous to (12), where $(h^{(1)}a_{11}^* g^{(1)} \oplus B^{(1)})^*$ plays the role of u_{k+1}. Here and in the sequel, $O_{k\times l}$ denotes the $k \times l$ matrix consisting only of zeros, and I_l denotes the identity matrix of size l. This can be also rewritten as

$$A^* = M_1^* D_1^* (A^{(2)})^* L_1^*, \tag{27}$$

where

$$\begin{aligned} M_1 &= \begin{pmatrix} O & h^{(1)}a_{11}^* \\ O_{(n-1)\times 1} & O_{(n-1)\times(n-1)} \end{pmatrix}, \\ D_1 &= \begin{pmatrix} a_{11} & O_{1\times(n-1)} \\ O_{(n-1)\times 1} & O_{(n-1)\times(n-1)} \end{pmatrix}, \\ A^{(2)} &= \begin{pmatrix} O_{1\times 1} & O_{1\times(n-1)} \\ O_{(n-1)\times 1} & R^{(2)} \end{pmatrix}, \\ L_1 &= \begin{pmatrix} O_{1\times 1} & O_{1\times(n-1)} \\ a_{11}^* g^{(1)} & O_{(n-1)\times(n-1)} \end{pmatrix}, \\ R^{(2)} &= h^{(1)}a_{11}^* g^{(1)} \oplus B^{(1)}. \end{aligned} \tag{28}$$

Here we used in particular that $L_1^2 = 0$ and $M_1^2 = 0$ and hence $L_1^* = I \oplus L_1$ and $M_1^* = I \oplus M_1$. The first step of Algorithm 7 ($k = 1$) computes

$$\begin{pmatrix} a_{11} & h^{(1)}a_{11}^* \\ a_{11}^* g^{(1)} & R^{(2)} \end{pmatrix} = A^{(2)} \oplus L_1 \oplus M_1 \oplus D_1, \tag{29}$$

which contains all relevant information.

We can now continue with the submatrix $R^{(2)}$ of $A^{(2)}$ factorizing it as in (26) and (27), and so on. Let us now formally describe the kth step of this construction, corresponding to the kth step of Algorithm 7. On that general step we deal with

$$A^{(k)} = \begin{pmatrix} O_{(k-1)\times(k-1)} & O_{(k-1)\times(n-k+1)} \\ O_{(n-k+1)\times(k-1)} & R^{(k)} \end{pmatrix}, \tag{30}$$

where

$$\begin{aligned} R^{(k)} &= h^{(k-1)}(a_{k-1,k-1}^{(k-1)})^* g^{(k-1)} \oplus B^{(k-1)} \\ &= \begin{pmatrix} a_{kk}^{(k)} & h^{(k)} \\ g^{(k)} & B^{(k)} \end{pmatrix}. \end{aligned} \tag{31}$$

Like on the first step we represent

$$(A^{(k)})^* = M_k^* D_k^* (A^{(k+1)})^* L_k^*, \tag{32}$$

where

$$\begin{aligned} M_k &= \begin{pmatrix} O_{(k-1)\times(k-1)} & O_{(k-1)\times 1} & O_{(k-1)\times(n-k)} \\ O_{1\times(k-1)} & O_{1\times 1} & h^{(k)}(a_{kk}^{(k)})^* \\ O_{(n-k)\times(k-1)} & O_{(n-k)\times 1} & O_{(n-k)\times(n-k)} \end{pmatrix}, \\ D_k &= \begin{pmatrix} O_{(k-1)\times(k-1)} & O_{(k-1)\times 1} & O_{(k-1)\times(n-k)} \\ O_{1\times(k-1)} & a_{kk}^{(k)} & O_{1\times(n-k)} \\ O_{(n-k)\times(k-1)} & O_{(n-k)\times 1} & O_{(n-k)\times(n-k)} \end{pmatrix}, \\ L_k &= \begin{pmatrix} O_{(k-1)\times(k-1)} & O_{(k-1)\times 1} & O_{(k-1)\times(n-k)} \\ O_{1\times(k-1)} & O_{1\times 1} & O_{1\times(n-k)} \\ O_{(n-k)\times(k-1)} & (a_{kk}^{(k)})^* g^{(k)} & O_{(n-k)\times(n-k)} \end{pmatrix}, \\ A^{(k+1)} &= \begin{pmatrix} O_{k\times k} & O_{k\times(n-k)} \\ O_{(n-k)\times k} & R^{(k+1)} \end{pmatrix}, \\ R^{(k+1)} &= h^{(k)}(a_{kk}^{(k)})^* g^{(k)} \oplus B^{(k)}. \end{aligned} \tag{33}$$

Note that we have the following recursion for the entries of $A^{(k)}$:

$$a_{ij}^{(k+1)} = \begin{cases} \mathbf{0}, & \text{if } i \le k \quad \text{or} \quad j \le k, \\ a_{ij}^{(k)} \oplus a_{ik}^{(k)} \left(a_{kk}^{(k)}\right)^* a_{kj}^{(k)}, & \text{otherwise.} \end{cases} \tag{34}$$

This recursion is immediately seen in Algorithm 7. Moreover it can be shown by induction that the matrix computed on the kth step of that algorithm equals

$$A^{(k+1)} \oplus \bigoplus_{i=1}^{k} L_i \oplus \bigoplus_{i=1}^{k} M_i \oplus \bigoplus_{i=1}^{k} D_i. \tag{35}$$

In other words, this matrix is composed from $h^{(1)}a_{11}^*, \ldots, h^{(k)}(a_{kk}^{(k)})^*$ (in the upper triangle), $a_{11}^* g^{(1)}, \ldots, (a_{kk}^{(k)})^* g^{(k)}$ (in the lower triangle), $a_{11}, \ldots, a_{kk}^{(k)}$ (on the diagonal), and $R^{(k+1)}$ (in the south-eastern corner).

After assembling and unfolding all expressions (32) for $A^{(k)}$, where $k = 1, \ldots, n$, we obtain

$$A^* = M_1^* D_1^* \cdots M_n^* D_n^* L_n^* \cdots L_1^*. \tag{36}$$

(actually, $M_n = L_n = 0$ and hence $M_n^* = L_n^* = I$). Noticing that D_i^* and M_j^* commute for $i < j$ we can rewrite

$$A^* = M_1^* \cdots M_n^* D_1^* \cdots D_n^* L_n^* \cdots L_1^*. \tag{37}$$

Consider the identities

$$\begin{aligned} (D_1 \oplus \cdots \oplus D_n)^* &= D_1^* \cdots D_n^*, \\ (L_1 \oplus \cdots \oplus L_n)^* &= L_n^* \cdots L_1^*, \\ (M_1 \oplus \cdots \oplus M_n)^* &= M_1^* \cdots M_n^*. \end{aligned} \tag{38}$$

The first of these identities is evident. For the other two, observe that $M_k^2 = L_k^2 = 0$ for all k, hence $M_k^* = I \oplus M_k$ and $L_k^* = I \oplus L_k$. Further, $L_i L_j = 0$ for $i > j$ and $M_i M_j = 0$ for $i < j$. Using these identities it can be shown that

$$\begin{aligned} (L_1 \oplus \cdots \oplus L_n)^* &= \bigoplus_{i=0}^{n-1} (L_1 \oplus \cdots \oplus L_n)^i \\ &= (I \oplus L_n) \cdots (I \oplus L_1) = L_n^* \cdots L_1^*, \\ (M_1 \oplus \cdots \oplus M_n)^* &= \bigoplus_{i=0}^{n-1} (M_1 \oplus \cdots \oplus M_n)^i \\ &= (I \oplus M_1) \cdots (I \oplus M_n) = M_1^* \cdots M_n^*, \end{aligned} \tag{39}$$

which yields the last two identities of (38). Notice that in (39) we have used the nilpotency of $L_1 \oplus \cdots \oplus L_n$ and $M_1 \oplus \cdots \oplus M_n$, which allows to apply (7).

It can be seen that the matrices $M := M_1 \oplus \cdots \oplus M_n, L := L_1 \oplus \cdots \oplus L_n$ and $D := D_1 \oplus \cdots \oplus D_n$ are contained in the upper triangle, in the lower triangle and, respectively, on the diagonal of the matrix computed by Algorithm 7. These matrices satisfy the LDM decomposition $A^* = M^* D^* L^*$. This concludes the explanation of Algorithm 7.

In terms of matrix computations, Algorithm 7 is a version of LDM decomposition with outer product. This algorithm can be reorganized to make it almost identical with Golub and van Loan (1989), Algorithm 4.1.1:

Algorithm 8 *LDM-decomposition (version 2).*

Input: an $n \times n$ matrix A with entries $a(i,j)$,
also used to store the final result
and intermediate results of the computation process.

for $j = 1 : n$
$v(1 : j) = a(1 : j, j)$
for $k = 1 : j-1$ $v(k+1 : j) = v(k+1 : j) \oplus a(k+1 : j, k) \odot v(k)$
end
for $i = 1 : j-1$
$a(i,j) = (a(i,i))^* \odot v(i)$
end
$a(j,j) = v(j)$
for $k = 1 : j-1$
$a(j+1 : n, j) = a(j+1 : n, j) \oplus a(j+1 : n, k) \odot v(k)$
end
$d = (v(j))^*$
$a(j+1 : n, j) = a(j+1 : n, j) \odot d$
end

This algorithm performs exactly the same operations as Algorithm 7, computing consecutively one column of the result after another. Namely, in the first half of the main loop it computes the entries $a_{ij}^{(i)}$ for $i = 1, \ldots, j$, first under the guise of the entries of v and finally in the assignment "$a(i,j) = (a(i,i))^* \odot v(i)$". In the second half of the main loop it computes $a_{kj}^{(j)}$. The complexity of this algorithm is the same as that of Algorithm 7.

3.4 LDM decomposition with symmetry and band structure

When matrix A is symmetric, that is, $a_{ij} = a_{ji}$ for all i, j, it is natural to expect that LDM decomposition must be symmetric too, that is, $M = L^T$. Indeed, going through the reasoning of the previous section, it can be shown by induction that all intermediate matrices $A^{(k)}$ are symmetric, hence $M_k = L_k^T$ for all k and $M = L^T$. We now present two versions of symmetric LDM decomposition, corresponding to the two versions of LDM decomposition given in the previous section. Notice that the amount of computations in these algorithms is nearly halved with respect to their full versions. In both cases they require $n^3/6 + O(n^2)$ operations $\oplus$ and $\odot$(each) and $n-1$ operations of taking algebraic closure.

Algorithm 9 *Symmetric LDM-decomposition (version 1).*

Input: an $n \times n$ symmetric matrix A with entries $a(i,j)$,
also used to store the final result
and intermediate results of the computation process.

for $j = 1 : n-1$
$v(j) = (a(j,j))^*$
for $k = j+1 : n$
for $l = j+1 : k$
$a(k,l) = a(k,l) \oplus a(k,j) \odot v(j) \odot a(l,j)$
end
end
$a(j+1 : n, j) = a(j+1 : n, j) \odot v(j)$
end

The strictly triangular matrix L is contained in the lower triangle of the result, and the matrix D is on the diagonal.

The next version generalizes Golub and van Loan (1989) Algorithm 4.1.2. Like in the prototype, the idea is to use the symmetry of A precomputing the first $j-1$ entries of v inverting the assignment "$a(i,j) = a(i,i)^* \odot v(i)$" for $i = 1,\ldots,j-1$. This is possible since $a(j,i) = a(i,j)$ belong to the first $j-1$ columns of the result that have been computed on the previous stages.

Algorithm 10 *Symmetric LDM-decomposition (version 2).*

A is an $n \times n$ symmetric matrix with entries $a(i,j)$,
also used to store the final result
and intermediate results of the computation process.
for $j = 1 : n$
for $i = 1 : j-1$
$v(i) = (a(i,i)^*)^{-1} a(i,j)$
end
$v(j) = v(j) \oplus a(j, 1 : j-1) \odot v(1 : j-1)$
$a(j,j) = v(j)$
for $k = 1 : j-1$
$a(j+1 : n, j) = a(j+1 : n, j) \oplus a(j+1 : n, k) \odot v(k)$
end
$d = (v(j))^*$
$a(j+1 : n, j) = a(j+1 : n, j) \odot d$ **end**

Note that this version requires invertibility of the closures $a(i,i)^*$ computed by the algorithm.

Remark 3 In the case of idempotent semiring we have $(D^*)^2 = D^*$, hence $A^* = (M^* D^*)(D^* L^*)$. When A is symmetric we can write $A^* = (G^*)^T G^*$ where $G = D^* L$. Evidently, this **idempotent Cholesky factorization** can be computed by minor modifications of Algorithms 9 and 10. See also Golub and van Loan (1989), Algorithm 4.2.2.

$A = (a_{ij})$ is called a **band matrix** with upper bandwidth q and lower bandwidth p if $a_{ij} = 0$ for all $j > i + q$ and all $i > j + p$. A band matrix with $p = q = 1$ is called **tridiagonal**. To generalize a specific LDM decomposition with band matrices, we need to show that the band parameters of the matrices $A^{(2)},\ldots,A^{(n)}$ computed in the process of LDM decomposition are not greater than the parameters of $A^{(1)} = A$. Assume by induction that $A = A^{(1)},\ldots,A^{(k)}$ have the required band parameters, and consider an entry $a_{ij}^{(k+1)}$ for $i > j + p$. If $i \le k$ or $j \le k$ then $a_{ij}^{(k+1)} = \mathbf{0}$, so we can assume $i > k$ and $j > k$. In this case $i > k + p$, hence $a_{ik}^{(k)} = \mathbf{0}$ and

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} \oplus a_{ik}^{(k)} \left(a_{kk}^{(k)}\right)^* a_{kj}^{(k)} = \mathbf{0}.$$

Thus we have shown that the lower bandwidth of $A^{(k)}$ is not greater than p. It can be shown analogously that its upper bandwidth does not exceed q. We use this to construct the following band version of LDM decomposition, see (Golub and van Loan 1989) Algorithm 4.3.1 for a prototype.

Algorithm 11 *LDM decomposition of a band matrix.*

A is an $n \times n$ band matrix with entries $a(i,j)$,
lower bandwidth p and upper bandwidth q
also used to store the final result
and intermediate results of the computation process.
for $j = 1 : n-1$
$v(j) = (a(j,j))^*$
for $i = j+1 : \min(j+p, n)$
$a(i,j) = a(i,j) \odot v(j)$
end
for $k = j+1 : \min(j+q, n)$
for $i = j+1 : \min(j+p, n)$
$a(k,j) = a(k,j) \oplus a(k,i) \odot a(i,j)$
end
end
for $k = j+1 : \min(j+q, n)$
$a(j,k) = v(j) \odot a(j,k)$
end
end

When p and q are fixed and $n > > p, q$ is variable, it can be seen that the algorithm performs approximately npq operations $\odot$ and $\oplus$ each.

Remark 4 There are important special kinds of band matrices, for instance, Hessenberg and tridiagonal matrices. Hessenberg matrices are defined as band matrices with $p = 1$ and $q = n$, while in the case of tridiagonal matrices $p = q = 1$. It is straightforward to write further adaptations of Algorithm 11 to these cases.

3.5 Iteration schemes

We are not aware of any truly universal scheme, since the decision when such schemes work and when they should be stopped depends both on the semiring and on the representation of data.

Our first scheme is derived from the following iteration process:

$$X^{(k+1)} = AX^{(k)} \oplus B \tag{40}$$

trying to solve the Bellman equation $X = AX \oplus B$. Iterating expressions (40) for all k up to m we obtain

$$X^{(m)} = A^m X^{(0)} \oplus \bigoplus_{i=0}^{m-1} A^i B \tag{41}$$

Thus the result crucially depends on the behaviour of $A^m X^{(0)}$. The algorithm can be written as follows (for the case when B is a column vector).

Algorithm 12 *Jacobi iterations*

Input: $n \times n$ matrix A with entries $a(i, j)$;
$n \times 1$ column vectors b and x

```
situation='proceed'
while situation=='proceed'
x = A ⊙ x ⊕ b
situation=newsituation(...)
if situation=='no convergence'
disp('Jacobi iterations did not converge')
exit
end
if situation=='convergence'
disp('Jacobi iterations converged')
exit
end
end
```

Output: situation, x.

Next we briefly discuss the behaviour of Jacobi iteration scheme over the usual arithmetic with nonnegative real numbers, and over semiring $\mathbf{R}_{\max}$. For simplicity, in both cases we restrict to the case of *irreducible* matrix A, that is, when the associated digraph is strongly connected.

Over the usual arithmetic, it is well known that (in the irreducible nonnegative case) the Jacobi iterations converge if and only if the greatest eigenvalue of A, denoted by $r(A)$, is strictly less than 1. This follows from the behaviour of $A^m x^{(0)}$. In general we cannot obtain exact solution of $x = Ax + b$ by means of Jacobi iterations.

In the case of $\mathbf{R}_{\max}$, the situation is determined by the behaviour of $A^m x^{(0)}$ which differs from the case of the usual nonnegative algebra. However, this behaviour can be also analysed in terms of $r(A)$, the greatest eigenvalue in terms of max-plus algebra (that is, with respect to the max-plus eigenproblem $A \odot x = \lambda \odot x$). Namely, $A^m x^{(0)} \to \mathbf{0}$ and hence the iterations converge if $r(A) < \mathbf{1}$. Moreover $A^* = (I \oplus A \oplus \cdots \oplus A^{n-1})$ and hence the iterations yield **exact** solution to Bellman equation after a **finite** number of steps. To the contrary, $A^m x^{(0)} \to +\infty$ and hence the iterations diverge if $r(A) > \mathbf{1}$. See, for instance, Carré (1971) for more details. On the boundary $r(A) = 1$, the powers A^m reach a periodic regime after a finite number of steps. Hence $A^* b \oplus A^m x^{(0)}$ also becomes periodic, in general. If the period of $A^m x^{(0)}$ is one, that is, if this sequence stabilizes, then the method converges to a general solution of $x = Ax \oplus b$ described as a superposition of $A^* b$ and an eigenvector of A (Butkovič et al. 2011; Krivulin 2006). The vector $A^* b$ may dominate, in which case the method converges to $A^* b$ as "expected". However, the period of $A^* b \oplus A^m x^{(0)}$ may be more than one, in which case the Jacobi iterations do not yield any solution of $x = Ax \oplus b$. See Butkovič (2010) for more information on the behaviour of max-plus matrix powers and the max-plus spectral theory.

In a more elaborate scheme of Gauss–Seidel iterations we can also use the previously found coordinates of $X^{(k)}$. In this case matrix A is written as $L \oplus U$ where L is the strictly lower triangular part of A, and U is the upper triangular part with the diagonal. The iterations are written as

$$X^{(k)} = LX^{(k)} \oplus UX^{(k-1)} \oplus B = L^* UX^{(k-1)} \oplus L^* B \tag{42}$$

Note that the transformation on the right hand side is unambiguous since L is strictly lower triangular and L^* is uniquely defined as $I \oplus L \oplus \cdots \oplus L^{n-1}$ (where n is the dimension of A). In other words, we just apply the forward substitution. Iterating expressions (42) for all k up to m we obtain

$$X^{(m)} = (L^* U)^m X^{(0)} \oplus \bigoplus_{i=0}^{m-1} (L^* U)^i L^* B \tag{43}$$

The right hand side reminds of the formula $(L \oplus U)^* = (L^* U)^* L^*$, see (8), so it is natural to expect that these iterations converge to $A^* B$ with a good choice of $X^{(0)}$. The result crucially depends on the behaviour of $(L^* U)^m X^{(0)}$. The algorithm can be written as follows (we assume again that B is a column vector).

Algorithm 13 *Gauss-Seidel iterations*

Input: $n \times n$ matrix A with entries $a(i, j)$;
$n \times 1$ column vectors b and x

```
situation='proceed'
while situation=='proceed'
for i = 1 : n
y(i) = a(i, i : n) ⊙ x(i : n) ⊕ b(i)
end
for i = 2 : n
x(i) = a(i, 1 : i − 1) ⊙ x(1 : i − 1)
end
situation=newsituation(...)
if situation=='no convergence'
disp('Gauss-Seidel iterations did not converge')
exit
end
if situation=='convergence'
disp('Gauss-Seidel iterations converged')
exit
end
end
```

Output: situation, x.

It is plausible to expect that the behaviour of Gauss–Seidel scheme in the case of max-plus algebra and nonnegative linear algebra is analogous to the case of Jacobi iterations.

3.6 Software implementation of universal algorithms

Software implementations for universal semiring algorithms cannot be as efficient as hardware ones (with respect to the computation speed) but they are much more flexible. Program modules can deal with abstract (and variable) operations and data types. Concrete values for these operations and data types can be defined by the corresponding input data. In this case concrete operations and data types are generated by means of additional program modules. For programs written in this manner it is convenient to use special techniques of the so-called object oriented (and functional) design, see, for instance, Lorenz (1993), Pohl (1997), Stepanov and Lee (1994). Fortunately, powerful tools supporting the object-oriented software design have recently appeared including compilers for real and convenient programming languages (for instance, *C++* and Java) and modern computer algebra systems. Recently, this type of programming technique has been dubbed generic programming (see, for instance, Pohl 1997).

C++ implementation Using templates and objective oriented programming, Tchourkin and Sergeev (2007) created a Visual *C++* application demonstrating how the universal algorithms calculate matrix closures A^* and solve Bellman equations $x = Ax \oplus b$ in various semirings. The program can also compute the usual system $Ax = b$ in the usual arithmetic by transforming it to the "Bellman" form. Before pressing "Solve", the user has to choose a semiring, a problem and an algorithm to use. Then the initial data are written into the matrix (for the sake of visualization the dimension of a matrix is no more than 10×10). The result may appear as a matrix or as a vector depending on the problem to solve. The object-oriented approach allows to implement various semirings as objects with various definitions of basic operations, while keeping the algorithm code unique and concise.

Examples of the semirings. The choice of semiring determines the object used by the algorithm, that is, the concrete realization of that algorithm. The following semirings have been realized:

(1) $\oplus = +$ and $\otimes = \times$: the usual arithmetic over reals;
(2) $\oplus = \max$ and $\otimes = +$: max-plus arithmetic over $\mathbb{R} \cup \{-\infty\}$;
(3) $\oplus = \min$ and $\otimes = +$: min-plus arithmetic over $\mathbb{R} \cup \{+\infty\}$;
(4) $\oplus = \max$ and $\otimes = \times$: max-times arithmetic over nonnegative numbers;
(5) $\oplus = \max$ and $\otimes = \min$: max-min arithmetic over a real interval $[a, b]$ (the ends a and b can be chosen by the user);
(6) $\oplus =$OR and $\otimes =$ AND: Boolean logic over the two-element set $\{0,1\}$.

Algorithms. The user can select the following basic methods:

(1) **Gaussian elimination scheme**, including the universal realizations of escalator method (Algorithm 1), Floyd-Warshall (Algorithm 2, Yershov's algorithm (based on a prototype from 11. Ch. 2), and the universal algorithm of Rote (Rote 1985);
(2) **Methods for Toeplitz systems** including the universal realizations of Durbin's and Levinson's schemes (Algorithms 3 and 4);
(3) **LDM decomposition** (Algorithm 7) and its adaptations to the symmetric case (Algorithm 9), band matrices (Algorithm 11), Hessenberg and tridiagonal matrices.
(4) **Iteration schemes** of Jacobi and Gauss–Seidel. As mentioned above, these schemes are not truly universal since the stopping criterion is different for the usual arithmetics and idempotent semirings.

Types of matrices. The user may choose to work with general matrices, or with a matrix of special structure, for instance, symmetric, symmetric Toeplitz, band, Hessenberg or tridiagonal.

Visualization. In the case of idempotent semiring, the matrix can be visualized as a weighted digraph. After performing the calculations, the user may wish to find an optimal path between a given pair of nodes, or to display an optimal paths tree. These problems can be solved using parental links like in the case of the classical Floyd-Warshall method computing all optimal paths, see, for instance, Sedgewick (2002). In our case, the mechanism of parental links can be implemented directly in the class describing an idempotent arithmetic.

Other arithmetics and interval extensions. It is also possible to realize various types of arithmetics as data types and combine this with the semiring selection. Moreover, all implemented semirings can be extended to their interval versions. Such possibilities were not realized in the program of Churkin and Sergeev (Tchourkin and Sergeev 2007), being postponed to the next version. The list of such arithmetics includes integers, and fractional arithmetics with the use of chain fractions and controlled precision.

MATLAB realization. The whole work (except for visualization tools) has been duplicated in MATLAB (Tchourkin and Sergeev 2007), which also allows for a kind of object-oriented programming. Obviously, the universal algorithms written in MATLAB are very close to those described in the present paper.

Future prospects. High-level tools, such as STL (Pohl 1997; Stepanov and Lee 1994), possess both obvious advantages and some disadvantages and must be used with caution. It seems that it is natural to obtain an implementation of the correspondence principle approach to scientific calculations in the form of a powerful software system based on a collection of universal algorithms. This approach should ensure a working time reduction for programmers and users because of the software unification. The arbitrary necessary accuracy and safety of numeric calculations can be ensured as well.

The system has to contain several levels (including programmer and user levels) and many modules.

Roughly speaking, it must be divided into three parts. The first part contains modules that implement domain modules (finite representations of basic mathematical objects). The second part implements universal (invariant) calculation methods. The third part contains modules implementing model dependent algorithms. These modules may be used in user programs written in C^{++}, Java, Maple, Matlab etc.

The system has to contain the following modules:

- Domain modules:
  - infinite precision integers;
  - rational numbers;
  - finite precision rational numbers (see Sergeev 2011);
  - finite precision complex rational numbers;
  - fixed- and floating-slash rational numbers;
  - complex rational numbers;
  - arbitrary precision floating-point real numbers;
  - arbitrary precision complex numbers;
  - p-adic numbers;
  - interval numbers;
  - ring of polynomials over different rings;
  - idempotent semirings;
  - interval idempotent semirings;
  - and others.
- Algorithms:
  - linear algebra;
  - numerical integration;
  - roots of polynomials;
  - spline interpolations and approximations;
  - rational and polynomial interpolations and approximations;
  - special functions calculation;
  - differential equations;
  - optimization and optimal control;
  - idempotent functional analysis;
  - and others.

This software system may be especially useful for designers of algorithms, software engineers, students and mathematicians.

Acknowledgments The authors are grateful to the anonymous referees for a number of important corrections in the paper.

References

Alefeld G, Herzberger J (1983) Introduction to interval computations. Academic Press, New York
Baccelli FL, Cohen G, Olsder GJ, Quadrat JP (1992) Synchronization and linearity: an algebra for discrete event systems. Wiley, Hoboken
Backhouse RC, Carré BA (1975) Regular algebra applied to path-finding problems. J Inst Math Appl 15:161–186
Barth W, Nuding E (1974) Optimale Lösung von Intervalgleichung-systemen. Comput Lett 12:117–125
Butkovič P (2010) Max-linear systems: theory and algorithms. Springer, Berlin
Butkovič P, Schneider H, Sergeev S (2011) Z-matrix equations in max algebra, nonnegative linear algebra and other semirings. http://www.arxiv.org/abs/1110.4564
Carré BA (1971) An algebra for network routing problems. J Inst Math Appl 7:273–294
Carré BA (1979) Graphs and networks. Oxford University Press, Oxford
Cechlárová K, Cuninghame-Green RA (2002) Interval systems of max-separable linear equations. Linear Alg Appl 340(1–3): 215–224
Cuninghame-Green RA (1979) Minimax algebra, volume 166 of lecture notes in economics and mathematical systems. Springer, Berlin
Faddeev DK, Faddeeva VN (2002) Computational methods of linear algebra. Lan', St. Petersburg. 3rd ed., in Russian
Fiedler M, Nedoma J, Ramík J, Rohn J, Zimmermann K (2006) Linear optimization problems with inexact data. Springer, New York
Golan J (2000) Semirings and their applications. Kluwer, Dordrecht
Golub GH, van Loan C (1989) Matrix computations. John Hopkins University Press, Baltimore and London
Gondran M (1975) Path algebra and algorithms. In: Roy B (ed), Combinatorial programming: methods and applications, Reidel, Dordrecht, pp 137–148
Gondran M, Minoux M (1979) Graphes et algorithmes. Éditions Eylrolles, Paris
Gondran M, Minoux M (2010) Graphs, dioids and semirings. Springer, New York a.o.
Gunawardena J (eds) (1998) Idempotency. Cambridge University Press, Cambridge
Hardouin L, Cottenceau B, Lhommeau M, Le Corronc E (2009) Interval systems over idempotent semiring. Linear Alg Appl 431:855–862
Kolokoltsov VN, Maslov VP (1997) Idempotent analysis and its applications. Kluwer, Dordrecht
Kreinovich V, Lakeev A, Rohn J, Kahl P (1998) Computational complexity and feasibility of data processing and interval computations. Kluwer, Dordrecht
Krivulin NK (2006) Solution of generalized linear vector equations in idempotent linear algebra. Vestnik St.Petersburg Univ Math 39(1):23–36
Lehmann DJ (1977) Algebraic structures for transitive closure. Theor Comp Sci 4:59–76
Litvinov GL (2007) The Maslov dequantization, idempotent and tropical mathematics: a brief introduction. J Math Sci 141(4):1417–1428. http://www.arxiv.org/abs/math.GM/0507014
Litvinov GL, Maslov VP (1996) Idempotent mathematics: correspondence principle and applications. Russ Math Surv 51:1210–1211

Litvinov GL, Maslov VP (1998) The correspondence principle for idempotent calculus and some computer applications. In: Gunawardena J (ed) Idempotency, Cambridge University Press, Cambridge, pp 420–443. http://www.arxiv.org/abs/math.GM/0101021

Litvinov GL, Maslova EV (2000) Universal numerical algorithms and their software implementation. Progr Comput Softw 26(5):275–280. http://www.arxiv.org/abs/math.SC/0102114Z

Litvinov GL, Sobolevskiĭ AN (2000) Exact interval solutions of the discrete bellman equation and polynomial complexity in interval idempotent linear algebra. Dokl Math 62(2):199–201. http://www.arxiv.org/abs/math.LA/0101041

Litvinov GL, Sobolevskiĭ AN (2001) Idempotent interval analysis and optimization problems. Reliab Comput 7(5):353–377. http://www.arxiv.org/abs/math.SC/0101080

Litvinov GL, Maslov VP, Rodionov AYa (2000) A unifying approach to software and hardware design for scientific calculations and idempotent mathematics. International Sophus Lie Centre, Moscow. http://www.arxiv.org/abs/math.SC/0101069

Litvinov GL, Maslov VP, Shpiz GB (2001) Idempotent functional analysis. An algebraic approach. Math Notes 69(5):696–729. http://www.arxiv.org/abs/math.FA/0009128

Litvinov GL, Rodionov AYa, Tchourkin AV (2008) Approximate rational arithmetics and arbitrary precision computations for universal algorithms. Int J Pure Appl Math 45(2):193–204. http://www.arxiv.org/abs/math.NA/0101152

Litvinov GL, Maslov VP, Rodionov AYa, Sobolevskiĭ AN (2011) Universal algorithms, mathematics of semirings and parallel computations. Lect Notes Comput Sci Eng 75:63–89. http://www.arxiv.org/abs/1005.1252

Lorenz M (1993) Object oriented software: a practical guide. Prentice Hall Books, Englewood Cliffs, N.J.

Maslov VP (1987a) A new approach to generalized solutions of nonlinear systems. Soviet Math Dokl 42(1):29–33

Maslov VP (1987b) On a new superposition principle for optimization process. Uspekhi Math Nauk [Russian Math Surveys] 42(3): 39–48

Mikhalkin G (2006) Tropical geometry and its applications. In: Proc ICM 2:827–852. http://www.arxiv.org/abs/math.AG/0601041

Moore RE (1979) Methods and applications of interval analysis. SIAM Studies in Applied Mathematics. SIAM, Philadelphia

Myšková H (2005) Interval systems of max-separable linear equations. Linear Alg Appl 403:263–272

Myšková H (2006) Control solvability of interval systems of max-separable linear equations. Linear Alg Appl 416:215–223

Neumaier A (1990) Interval methods for systems of equations. Cambridge University Press, Cambridge

Pohl I. (1997) Object-Oriented Programming Using *C++* , 2nd ed. Addison-Wesley, Reading

Rote G (1985) A systolic array algorithm for the algebraic path problem. Comput Lett 34:191–219

Sedgewick R (2002) Algorithms in *C++* . Part 5: graph algorithms, 3rd ed. Addison-Wesley, Reading

Sergeev S (2011) Universal algorithms for generalized discrete matrix Bellman equations with symmetric Toeplitz matrix. Tambov University Reports, ser. Natural and Technical Sciences 16(6):1751–1758. http://www.arxiv.org/abs/math/0612309

Simon I (1988) Recognizable sets with multiplicities in the tropical semiring. Lect Notes Comput Sci 324:107–120

Sobolevskiĭ AN (1999) Interval arithmetic and linear algebra over idempotent semirings. Dokl Math 60:431–433

Stepanov A, Lee M (1994) The standard template library. Hewlett-Packard, Palo Alto

Tchourkin AV, Sergeev SN (2007) Program demonstrating how universal algorithms solve discrete Bellman equation over various semirings. In: Litvinov G, Maslov V, Sergeev S (eds) Idempotent and tropical mathematics and problems of mathematical physics (Volume II), Moscow. French-Russian Laboratory J.V. Poncelet. http://www.arxiv.org/abs/0709.4119

Viro O (2001) Dequantization of real algebraic geometry on logarithmic paper. In: 3rd European Congress of Mathematics: Barcelona, July 10–14, 2000. Birkhäuser, Basel, pp 135. http://www.arxiv.org/abs/math/0005163

Viro O (2008) From the sixteenth hilbert problem to tropical geometry. Jpn J Math 3:1–30