PROCEEDINGS A

royalsocietypublishing.org/journal/rspa

Research

Cite this article: Adukov VM, Adukova NV, Mishuris G. 2022 An explicit Wiener–Hopf factorization algorithm for matrix polynomials and its exact realizations within ExactMPF package. *Proc. R. Soc. A* **478**: 20210941. https://doi.org/10.1098/rspa.2021.0941

Received: 15 December 2021
Accepted: 13 June 2022

Subject Areas:
analysis

Keywords:
matrix polynomials, Wiener–Hopf factorization, Toeplitz matrices, essential polynomials, exact computation, implementation

Author for correspondence:
V. M. Adukov
e-mail: adukovvm@susu.ru

One contribution to a special feature 'Advances in Wiener-Hopf type techniques: theory and applications' organized by Gennady Mishuris and Anastasia Kisil.

Electronic supplementary material is available online at https://doi.org/10.6084/m9.figshare.c.6060394.

THE ROYAL SOCIETY
PUBLISHING

An explicit Wiener–Hopf factorization algorithm for matrix polynomials and its exact realizations within ExactMPF package

V. M. Adukov[1], N. V. Adukova[1] and G. Mishuris[2]

[1]Institute of Natural Sciences and Mathematics, South Ural State University, Chelyabinsk 454080, Russia
[2]Institute of Mathematics, Physics and Computer Science, Aberystwyth University, Aberystwyth SY23 3BZ, UK

VMA, 0000-0002-3594-3565; NVA, 0000-0002-6758-6101; GM, 0000-0003-2565-1961

We discuss an explicit algorithm for solving the Wiener–Hopf factorization problem for matrix polynomials. By *an exact solution* of the problem, we understand the one constructed by a symbolic computation. Since the problem is, generally speaking, unstable, this requirement is crucial to guarantee that the result following from the explicit algorithm is indeed a solution of the original factorization problem. We prove that a matrix polynomial over the field of Gaussian rational numbers admits the exact Wiener–Hopf factorization if and only if its determinant is exactly factorable. Under such a condition, we adapt the explicit algorithm to the exact calculations and develop the ExactMPF package realized within the Maple Software. The package has been extensively tested. Some examples are presented in the paper, while the listing is provided in the electronic supplementary material. If, however, a matrix polynomial does not admit the exact factorization, we clarify a notion of the numerical (or approximate) factorization that can be constructed by following the explicit factorization algorithm. We highlight possible obstacles on the way and discuss a level of confidence in the final

© 2022 The Author(s) Published by the Royal Society under the terms of the Creative Commons Attribution License http://creativecommons.org/licenses/by/4.0/, which permits unrestricted use, provided the original author and source are credited.

result in the case of an unstable set of partial indices. The full listing of the package ExactMPF is given in the electronic supplementary material.

1. Introduction and motivation

The factorization problem has proven to be an extremely efficient tool in mathematics (for example, in the theory of linear operators [1,–3]) and instrumental in various branches of physics, mechanics and engineering (see, for example, [4,–8]).

Unfortunately, in contrast with the scalar problems, factorization of arbitrary matrix functions remains an unresolved problem [9,10] (especially, when one speaks of its effective applications). We start with definitions, preliminary results and than formulate the problem considered in this paper.

Let Γ be a simple piecewise smooth closed contour in the complex plane $\mathbb{C}$ bounding the domain D_+. We can assume that $0 \in D_+$. The complement of $D_+ \cup \Gamma$ in the closed complex plane, $\overline{\mathbb{C}} = \mathbb{C} \cup \{\infty\}$, is denoted by D_-. Let $a(t)$ be $p \times p$ matrix function on Γ. In what follows we assume that $a(t)$ is continuous and invertible on Γ.

The representation of a matrix function $a(t)$

$$a(t) = r_-(t) d_R(t) r_+(t), \quad t \in \Gamma, \tag{1.1}$$

is called *the right Wiener–Hopf factorization* of the matrix function [1]. Here, $r_\pm(t)$ are continuous and invertible matrix functions on Γ that admit analytic continuation into $D_\pm$ and their continuations $r_\pm(z)$ are invertible into the respective domains; matrix function $d_R(t) = \operatorname{diag}[t^{\rho_1}, \ldots, t^{\rho_p}]$, where integers $\rho_1, \ldots, \rho_p$ are called *the right partial indices* of $a(t)$. It is always possible to assume that $\rho_1 \le \ldots \le \rho_p$. Moreover, $\rho_1 + \cdots + \rho_p = \varkappa$, where $\varkappa = \operatorname{ind} \det a(t)$ is the winding number (or the Cauchy index) of the determinant of the matrix $a(t)$.

Similarly, *the left Wiener–Hopf factorization* of $a(t)$ is the representation [1]

$$a(t) = l_+(t) d_L(t) l_-(t), \quad t \in \Gamma,$$

where $l_\pm(t)$ have the same properties as $r_\pm(t)$, $d_L(t) = \operatorname{diag}[t^{\lambda_1}, \ldots, t^{\lambda_p}]$, and $\lambda_1 \ge \cdots \ge \lambda_p$ are called *the left partial indices* of the matrix function $a(t)$ $(\lambda_1 + \cdots + \lambda_p = \varkappa)$. Note that, if the factorizations exist, their left and right indices are usually different sets of integers and constructions of the right and left factorizations are usually considered as two separate problems. On the contrary, a method proposed in this work requires simultaneous considerations of both factorizations.

In general, a matrix function $a(t)$ with continuous entries does not admit the Wiener–Hopf factorizations with factors in the same class. It is known that if the elements of $a(t)$ satisfy the Hölder condition on Γ or belong to the Wiener algebra $W(\mathbb{T})$, where $\mathbb{T}$ is the unit circle in the complex plane, then the Wiener–Hopf factorizations exist (see [1]).

The scalar factorization problem for a non-singular function $a(t)$ can always be solved explicitly, see, for example, [11] and [12]. Unfortunately, explicit formulae for the factors $r_\pm(t)$, $l_\pm(t)$ and the partial indices of an arbitrary matrix function do not exist at present.

Fortunately, there are classes of matrix functions for which an *explicit (or constructive) solution* to factorize the problem have been found. By this, we understand that there are explicit formulae allowing us to formally construct the partial indices and the factors of the matrix function $a(t)$. For example, the problem for matrix polynomials has been solved explicitly [–1,13,15]. For meromorphic matrix functions, there also exits an explicit solution [13]. A detailed review of constructive methods for the factorization problem is presented in the work [10] (see, also [16]).

Unfortunately, the factorization problem for matrix functions is generally speaking unstable. More precisely, if the difference between the larger and the smaller partial indices is greater than one, then the set of the partial indices can change with an arbitrary small perturbation of the initial matrix [12,17]. Moreover, if left (right) factorization appears to be stable, this does not mean that the right (left) preserves that property [1]. There is also a technical difficulty embedded into the problem as the factors $r_\pm$, $l_\pm$ are not found uniquely, while how to properly normalize the algorithm to guarantee the uniqueness in the chosen class is still open. On the other hand (see, for example, [18], theorem 6.15), if two matrix functions $a(z)$ and $\widetilde{a}(z)$ are close enough, and their partial indices coincide, then it is always possible to find a factorization of $\widetilde{a}(z)$ for which factors $\widetilde{a}_\pm(z)$ are close enough to the factors of $a_\pm(z)$. Recently, a few constructive approximate techniques for factorization of specific matrix functions have been proposed [19,–22], under the condition that their sets of partial indices are known in advance. Accompanied stability analysis has also been performed in the case of unstable partial indices.

Therefore, *the key step* in construction of any (exact or approximate) factorization of a given matrix function (if the factorization exists) is to provide a procedure/formula allowing for calculation of the partial indices. However, even having an explicit route to achieve the goal, it often requires a numerical realization. And here, one may meet with a hidden unexpected obstacle: the procedure can again be unstable. Below we discuss two simple examples highlighting the issue.

Consider two matrix polynomials (the only elements $p_{11}(z)$ are slightly different there):

$$\left.\begin{aligned} \mathcal{P}_1(z) &= \begin{pmatrix} 2z^2 - z + \frac{1}{4} & -4z^2 + 2z - 1 \\ z & z^3 + z^2 + z + 4 \end{pmatrix} \\ \text{and} \qquad \mathcal{P}_2(z) &= \begin{pmatrix} 2z^2 - z + \frac{1}{2} & -4z^2 + 2z - 1 \\ z & z^3 + z^2 + z + 4 \end{pmatrix}. \end{aligned}\right\} \tag{1.2}$$

To factorize explicitly the polynomials by the method developed in [14], we need first to factorize their determinants. For the first matrix function, we have

$$\det \mathcal{P}_1 = 2z^5 + z^4 + \frac{21}{4}z^3 + \frac{21}{4}z^2 - \frac{11}{4}z + 1 \tag{1.3}$$

and the factorization of this polynomial can only be done approximately. Hence, all subsequent steps for the explicit factorization $\mathcal{P}_1(z)$ may be unstable. To apply the effective criterion of stability for partial indices [23], we have to verify that a certain matrix with float point entries, built using the matrix $\mathcal{P}_1(z)$, has the full rank. In turn, this is an ill-condition problem itself if the condition number of the matrix is large enough ([24], ch. 2). Thus, within the framework of this particular and well-developed explicit method, we cannot always even determine for sure whether the partial indices are stable or not. All other methods for matrix polynomials known in the literature [15] have the same obstacle when used to factorize the matrix $\mathcal{P}_1(z)$.

A completely different situation arises with the polynomial $\mathcal{P}_2(z)$. Now the determinant

$$\det \mathcal{P}_2 = 2z^5 + z^4 + \frac{11}{2}z^3 + \frac{11}{2}z^2 - \frac{5}{2}z + 2 \tag{1.4}$$

also has its roots that can be found only numerically (and thus, approximately). However, it can be presented as a product of irreducible polynomials $\det \mathcal{P}_2 = (2z^2 - z + 1/2)(z^3 + z^2 + 3z + 4)$ and all roots of $2z^2 - z + 1/2$ lie in an interior of the unit circle, while all roots of the polynomial $z^3 + z^2 + 3z + 4$ lie in the domain exterior. We can verify these statements without calculating the roots themselves by using Schur's test (see §3). Hence,

$$\det \mathcal{P}_2(z) = \left(2 - \frac{1}{z} + \frac{1}{2z^2}\right) z^2(z^3 + z^2 + 3z + 4) \tag{1.5}$$

is the Wiener–Hopf factorization of the determinant, $\det \mathcal{P}_2$, over the field of the Gaussian rational numbers $\mathbb{Q}(i)$, since the coefficients of the matrix polynomial $\mathcal{P}_2(z)$ belong to this field. In what follows, we show that all steps of the *explicit* factorization algorithm can be performed in the *exact* arithmetic and, as a result, the instability issue does not arise in this case.

These examples demonstrate that

(i) existence of an *explicit* procedure for a particular matrix factorization does not guarantee that it can be successfully used in practice;
(ii) knowledge of exact values of zeros of the polynomial is not an essential feature (useful though) for execution of the *exact* factorization;
(iii) the notions of *explicit, exact and numerical* factorizations should be clearly distinguished.

By the *explicit* (or *constructive*) solution of the factorization problem, we understand a clearly defined algorithmic procedure that should definitely terminate after a finite number of steps. However, when one implements a specific *explicit* factorization algorithm, each consequent step can be executed *exactly* or *approximately (numerically)*.

To avoid the difficulties associated with possible instability, one can chose the input data belonging to the field $\mathbb{Q}(i)$ and perform all steps of the *explicit algorithm* in the *exact arithmetic*. In such a case, we find that there exists *an exact solution* of the factorization problem. Unfortunately, as has been discussed above, this resolution is not always available even when in possession of a few different *explicit* algorithms.

If a factorization problem does not admit *the exact* solution, the only choice that remains is to construct a *numerical (approximate)* factorization. The latter, in view of possible instability, may represent a real challenge for execution and all computations should be done with special care and supplemented by accurate analysis of the partial indices preservation and a numerical stability.

Thus, the key question arising during the factorization of a matrix function is: how to compute the partial indices with a full confidence?

There are classes of matrix functions for which the partial indices are known in advance. For example, positive definite matrix functions [12] have all their partial indices equal to zero, while, for matrix functions not containing zero in their numerical ranges [25], they are equal (but not necessarily zeros). Such sets of the partial indices are stable with respect to small perturbation of the matrix functions [12], and, as a result, the remaining task in the factorization process (computation of the factors) is less problematic and any appropriately chosen numerical algorithm can be performed and justified by standard techniques.

In this study, we discuss factorization of matrix polynomials on the unit circle $\mathbb{T}$ with particular attention to all these details. The structure of the paper is as follows. In §1, we describe a well-known explicit method for constructing a factorization of a matrix polynomial [14]. We formulated this method algorithmically in order to indicate explicitly all the steps that must be taken to construct the factorization. This will be convenient for further attempts to implement our method numerically. However, in this section, we do not consider computational problems related to our method. Therefore, no analysis of the accuracy (forward/backward error) of the explicit algorithm is carried out in this section. The basic tool for this factorization is a method of essential polynomials [26].

In §3, we show that all steps of the *explicit* algorithm for factorization of a matrix polynomial, $a(z)$, over the Gaussian rational field $\mathbb{Q}(i)$ is implemented in the *exact arithmetic* if and only if its determinant, $\det a(z)$, allows *exact* factorization in this field. The solution of the latter is presented in theorem 3.1. We used representation a scalar polynomial as the product of its irreducible factors and the Schur algorithm to test whether each factor has its roots siting inside (or outside) the unit circle. Further, all steps of the explicit matrix factorization can be performed in the *exact* arithmetic. The algorithm for *exact* evaluation of the essential polynomials is a foundation stone of theorem 3.2 which is the main result of this section. The formal description of this algorithm is rather complicated and will be presented in the separate work.

In §4, we implemented the exact factorization algorithm as the package ExactMPF in the computer algebra system Maple.[1] This package consists of the following basic procedures: SchurTest, SolverExactPF, ExactMLC, seqTk, NumRank, ExactFEP. The main procedure is

[1]https://www.maplesoft.com/.

SolverExactMP. The purposes of these procedures are shortly described in the same section, while the full listing of the package ExactMPF is given in the *electronic supplementary material*. Several examples of the *exact* factorizations performed with use of the package ExactMPF are presented in §5 with the aim to demonstrate the package abilities and to study peculiarities of the factorization for special classes of matrix functions. Other tests are used to analyse stability of the computations (see examples 5.1 and 5.2). All those and other examples are included in the electronic supplementary material with more details.

In the general case, the explicit algorithm can only be realized approximately. This poses a real challenge for users due to the instability of the problem. Our numerical experiments with ExactMPF have demonstrated that even in the case of unstable sets of partial indices there is a good chance to obtain an approximate factorization. Namely, to accelerate the work of ExactMPF, we provide an option 'symnum' into the SolverExactMP. If this option is turn on, the package calculates the ranks of matrices by the singular value decomposition (SVD) method [24,27]. These numerical ranks are used to find partial indices of the initial matrix polynomial. Further calculations are performed in the exact arithmetic. This allows for a verification of the partial indices finding by SVD versus the exact ones: (are the sets of the partial indices the same or not?). It turned out that for many examples, even with unstable sets of the partial indices, such a numerical approach leads to correct results. The partial indices found with the SVD method are called partial τ-indices of the polynomial. However, the question whether or when they do coincide remains open. The opportunity to use SVD for numerical calculations of partial indices in a more general context is discussed in §6. Finally, in the Conclusion, we underline the advances made in this paper and discuss further possible developments and applications.

2. An explicit Wiener–Hopf factorization for matrix polynomials

In this section, following the results of the work [14], we present an *explicit* algorithm for the factorization of matrix polynomials. In accordance with our understanding of the notion of *explicit* solution (see the Introduction), we describe below those finite numbers of steps that, after their execution, allows one to construct a factorization. We focus on the algorithmic structure of *this explicit procedure* that should be implemented within a respective numerical code. We would like to emphasize here that this section is not devoted to any study of possible computational problems that may appear during the algorithm implementation into a specific environment.

We start with **Input data**, that is, a matrix polynomial of a degree n, $a(z)=\sum_{j=0}^{n} a_j z^j$, represented by the set of its consecutive coefficients, $a_j \in \mathbb{C}^{p\times p}$, $j=0,\dots,n$, and $p\in\mathbb{N}$ defines the matrix size. We assume that the matrix polynomial, $a(z)$, is *invertible* on the unit circle $\mathbb{T}$.

Step 1. Construction of the Wiener–Hopf factorization of the scalar polynomial det $a(z)$. We denote $\Delta(z)=\det a(z)$. Since $\Delta(z)\neq 0,\ |z|=1$, the determinant admits the Wiener–Hopf factorization

$$\Delta(z)=\Delta_-(z)z^{\varkappa}\Delta_+(z)\quad\text{and}\quad \Delta_-(\infty)=1, \tag{2.1}$$

relative to $\mathbb{T}$, that is unique with the additional condition at infinity for the polynomial $\Delta_-(z)$. In the sequel, we use, in fact, only one of the factors, namely,

$$\Delta_-(z)=1+\Delta_1^- z^{-1}+\cdots+\Delta_{\varkappa}^- z^{-\varkappa},\qquad \varkappa=\mathrm{ind}_{\mathbb{T}}\det a(z). \tag{2.2}$$

It should be noted that any explicit factorization algorithm will use the calculation of the determinant of the factorizable matrix function. The authors realize that the complexity for computing the determinant of a matrix of order n is $n!$. Fortunately, in applications this happens for matrix function of a small order. Moreover, in §3, we use the matrix determinant in conditions of the exact calculations.

Here, $\varkappa$ is the Cauchy index of $\det a(z)$. There is Gakhov's formula for this, which is convenient for numerical calculations. However, in our case, we can find this index by the exact calculations (see theorem 3.1).

Step 2. Evaluation of a finite set of Laurent coefficients of the matrix function $\Delta_-^{-1}(z)a(z)$.
Laurent coefficients of the auxiliary rational matrix function $\Delta_-^{-1}(z)a(z)$ are computed in terms of matrix coefficients a_j of the original matrix polynomial, $a(z)$, and the coefficients of the scalar polynomial, $\Delta_-(z)$, defined on the previous step. The rational function $\Delta_-^{-1}(z)$ is analytic in a domain $|z| > r$ for some $r < 1$. Let us expand $\Delta_-^{-1}(z)$ and $\Delta_-^{-1}(z)a(z)$ in the Laurent series

$$\Delta_-^{-1}(z) = \sum_{j=0}^{\infty} \frac{\delta_j}{z^j} \quad \text{and} \quad \Delta_-^{-1}(z)a(z) = \sum_{j=-\infty}^{n} c_j z^j \tag{2.3}$$

in this domain. Coefficients $\delta_j \in \mathbb{C}$ and $c_j \in \mathbb{C}^{p\times p}$, in the series above, can be found from the following relations:

$$\left.\begin{aligned} &\delta_0 = 1, \quad \delta_1 = -\Delta_1^- \\ &\delta_j = \begin{cases} -\Delta_j^- - \sum_{k=1}^{j-1} \delta_{j-k}\Delta_k^-, & 2 \le j \le \varkappa, \\ -\sum_{k=1}^{\varkappa} \delta_{j-k}\Delta_k^-, & \varkappa + 1 \le j \le \varkappa + n, \end{cases} \end{aligned}\right\} \tag{2.4}$$

and

$$c_j = \begin{cases} \sum_{k=0}^{n} \delta_{k-j} a_k, & j \le 0, \\ \sum_{k=j}^{n} \delta_{k-j} a_k, & 0n, \end{cases} \tag{2.5}$$

where the factor $\Delta_-(z)$ has been computed on the previous step in (2.2). In what follows, we need only a finite number of the coefficients, c_k, for $k = -\varkappa, \dots, 0, \dots, \varkappa$.
Let $c_{-\varkappa}^{\varkappa} := \{c_{-\varkappa}, \dots, c_0, \dots, c_\varkappa\}$ be the finite sequence of complex $p \times p$ matrices which are the Laurent coefficients of $\Delta_-^{-1}(z)a(z)$. We form a finite family of the block Toeplitz matrices of finite sizes

$$T_k = ||c_{i-j}||_{\substack{i=k,k+1,\dots,\varkappa, \\ j=0,1,\dots,\varkappa+k}} \quad -\varkappa \le k \le \varkappa. \tag{2.6}$$

The next step is a corner stone in implementation of the *explicit* algorithm proposed in [14].

Step 3. Finding the ranks and the null spaces for of block Toeplitz matrices T_k.
Let $\ker_R T_k$ ($\ker_L T_k$) be the right (left) kernels of the matrices T_k. Recall that the right and left kernels of a complex block matrix $A \in \mathbb{C}^{(n+1)p\times(m+1)p}$ consisting of $p \times p$ blocks are defined by the relations $\ker_R A = \{x : Ax = 0\}$, $\ker_L A = \{y : yA = 0\}$.

Let us represent an element $R \in \ker_R A$ in a block form $R = \begin{pmatrix} r_0 \\ r_1 \\ \vdots \\ r_m \end{pmatrix}$, $r_j \in \mathbb{C}^{p\times 1}$, and define a column-valued generating polynomial in z related to R as $R(z) = r_0 + r_1 z + \cdots + r_m z^m$.
By $\mathcal{N}_k^R$, $-\varkappa \le k \le \varkappa$, we denote the space of generating vector polynomials for vectors in $\ker_R T_k$. Put $\mathcal{N}_{-\varkappa-1}^R := \{0\}$ and let $\mathcal{N}_{\varkappa+1}^R$ be $(2\varkappa + 2)p$–dimensional space of all column-valued polynomials whose degrees are not greater than $2\varkappa + 1$.
Repeating the same line of reasoning, we define the space $\mathcal{N}_k^L$, $-\varkappa \le k \le \varkappa$ of the row-valued generating polynomials in z^{-1} for the rows from $\ker_L T_k$.
As a result of this step, the set of ranks of the matrices T_k defined in (2.6), and bases of the spaces $\mathcal{N}_k^R$, $\mathcal{N}_k^L$, $-\varkappa \le k \le \varkappa$, has been computed. All those elements, values and sets are of finite dimensions.
In general, computing rank and calculation of basis elements of the null-spaces are unstable processes. In our work, we will find the rank and null-space of a matrix with the help of calculation in the exact arithmetic.

Step 4. Computation of indices and factorization essential polynomials.

The main tools for computations of the partial indices and factors in the factorization of matrix functions are the so-called indices and essential polynomials of the sequence $c_{-\varkappa}^{\varkappa}$ (see [14,26]). Below we give necessary definitions and discuss how to compute them.

By d_k^R, we denote a dimension of the right kernel $\mathcal{N}_k^R$ and introduce the following integers: $\Delta_k^R = d_k^R - d_{k-1}^R$ for $-\varkappa \le k \le \varkappa + 1$. A sequence $c_{-\varkappa}^{\varkappa}$ is called *regular* if $\Delta_{-\varkappa}^R = 0$ and $\Delta_{\varkappa+1}^R = 2p$. For a regular sequence, we have (see [14,26])

$$0 = \Delta_{-\varkappa}^R \le \Delta_{-\varkappa+1}^R \le \cdots \le \Delta_{\varkappa}^R \le \Delta_{\varkappa+1}^R = 2p. \tag{2.7}$$

Since a monotone integer sequence is piecewise constant, then there are $2p$ integers $\mu_1 \le \cdots \le \mu_{2p}$ such that

$$\begin{array}{ccccccc} \Delta_{-\varkappa}^R & = & \cdots & = & \Delta_{\mu_1}^R & = & 0, \\ & & \multicolumn{3}{c}{\dots\dots\dots\dots\dots\dots\dots} & & \\ \Delta_{\mu_i+1}^R & = & \cdots & = & \Delta_{\mu_{i+1}}^R & = & i, \\ & & \multicolumn{3}{c}{\dots\dots\dots\dots\dots\dots\dots} & & \\ \Delta_{\mu_{2p}+1}^R & = & \cdots & = & \Delta_{\varkappa+1}^R & = & 2p. \end{array} \tag{2.8}$$

The absence of the jth row here means that $\mu_{j+1} = \mu_j$.

Definition 2.1. The integers $\mu_1, \dots, \mu_{2p}$ defined by the relations (2.8) are the indices of the sequence $c_{-\varkappa}^{\varkappa}$.

It turns out that (see [14,26])

$$\sum_{j=1}^{2p} \mu_j = 0. \tag{2.9}$$

Furthermore, we define the right essential polynomials of the sequence $c_{-\varkappa}^{\varkappa}$. Note that $\mathcal{N}_k^R$ and $z\mathcal{N}_k^R$ are subspaces of $\mathcal{N}_{k+1}^R$ as it follows from the definition of the spaces $\mathcal{N}_k^R$. The dimension, h_{k+1}^R, of the complement $\mathcal{H}_{k+1}^R$ of $\mathcal{N}_k^R + z\mathcal{N}_k^R$ in $\mathcal{N}_{k+1}^R$ is equal to $\Delta_{k+1}^R - \Delta_k^R$. Then, equations (2.8) imply that $h_{k+1}^R \ne 0$ if and only if $k = \mu_j$, $j = 1, \dots, 2p$. Moreover, in this case, h_{k+1}^R is equal to the multiplicity, $\varkappa_j$, of the index μ_j. Therefore, for $k \ne \mu_j$, we have

$$\mathcal{N}_{k+1}^R = \mathcal{N}_k^R + z\mathcal{N}_k^R,$$

and for $k = \mu_j$

$$\mathcal{N}_{k+1}^R = (\mathcal{N}_k^R + z\mathcal{N}_k^R) \oplus \mathcal{H}_{k+1}^R. \tag{2.10}$$

Definition 2.2. Any polynomials $R_j(z), \dots, R_{j+\varkappa_j-1}(z)$ forming a basis for a complement $\mathcal{H}_{\mu_j+1}^R$ are called right essential polynomials of the sequence $c_{-\varkappa}^{\varkappa}$ corresponding to the index μ_j.

As a result, we have defined $2p$ indices $\mu_1, \dots, \mu_{2p}$ and $2p$ right essential polynomials $R_1(z), \dots, R_{2p}(z)$ for any regular sequence $c_{-\varkappa}^{\varkappa}$. Similarly, we can define the left essential polynomials $L_1(z), \dots, L_{2p}(z)$ of the sequence $c_{-\varkappa}^{\varkappa}$.

By theorem 3.1 from [14], the sequence $c_{-\varkappa}^{\varkappa}$ defined in equation (2.5) is regular and there exist respective essential polynomials $R_1(z), \dots, R_{2p}(z); L_1(z), \dots, L_{2p}(z)$ such that

(1) (i) the constant terms of the polynomials $R_1(z), \dots, R_p(z)$ are equal to zero,
(2) (ii) the leading terms of the polynomials $L_{p+1}(z), \dots, L_{2p}(z)$ are equal to zero.

Definition 2.3. The essential polynomials $R_1(z), \dots, R_{2p}(z); L_1(z), \dots, L_{2p}(z)$ satisfying the conditions (i)–(ii) are called the factorization essential polynomials of the sequence.

Step 5. Construction of the factorization of the initial matrix polynomial, $a(z)$.

Now we are in a position to deliver a final result on the explicit Wiener–Hopf factorization of a matrix polynomial $a(z)$ (see theorem 3.2 in [14]). It is crucial to note here that, in contrast to any other methods, if it exists for a particular matrix function, our approach requires simultaneous consideration of both the right and the left factorizations.

Theorem 2.4. *Let $\mu_1, \ldots, \mu_{2p}$ be the indices and $R_1(z), \ldots, R_{2p}(z)$ $(L_1(z), \ldots, L_{2p}(z))$ are the right (left) factorization essential polynomials of the sequence $c^{\varkappa}_{-\varkappa}$. Let us introduce the $p \times p$ matrix functions*

$$\mathcal{R}_1(z) = \begin{pmatrix} R_1(z) & \ldots & R_p(z) \end{pmatrix} \quad \text{and} \quad \mathcal{L}_2(z) = \begin{pmatrix} L_{p+1}(z) \\ \vdots \\ L_{2p}(z) \end{pmatrix} \tag{2.11}$$

and

$$d_L(z) = \begin{pmatrix} z^{-\mu_1} & \ldots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & z^{-\mu_p} \end{pmatrix} \quad \text{and} \quad d_R(z) = \begin{pmatrix} z^{\mu_{p+1}} & \ldots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & z^{\mu_{2p}} \end{pmatrix}.$$

Then the left $(\lambda_1 \geq \cdots \geq \lambda_p)$ and right $(\rho_1 \leq \cdots \leq \rho_p)$ partial indices and the factors $(l_\pm(z), r_\pm(z))$ of the respective factorizations of the matrix polynomial $a(z)$ are defined by the formulae

$$\lambda_1 = -\mu_1, \ldots, \lambda_p = -\mu_p \quad \rho_1 = \mu_{p+1}, \ldots, \rho_p = \mu_{2p}, \tag{2.12}$$

$$l_-(z) = z^{\varkappa+1}\Delta_-(z)d_L^{-1}(z)\mathcal{R}_1^{-1}(z) \quad l_+(t) = z^{-\varkappa-1}\Delta_-^{-1}(z)a(z)\mathcal{R}_1(z) \tag{2.13}$$

and

$$r_-(z) = \Delta_-(z)\mathcal{L}_2^{-1}(z) \quad r_+(z) = \Delta_-^{-1}(z)d_R^{-1}(z)\mathcal{L}_2(z)a(z). \tag{2.14}$$

■

In the statement of this theorem, we have corrected the misprints appearing in the formulae for the factors $l_+(z)$, $r_+(z)$ in theorem 3.2 of [14].

It follows from this theorem that the indices of the sequence $c^{\varkappa}_{-\varkappa}$, in addition to (2.9), satisfy conditions

$$\sum_{j=1}^{p} \mu_j = -\varkappa \quad \text{and} \quad \sum_{j=p+1}^{2p} \mu_j = \varkappa. \tag{2.15}$$

Those five steps above represent an *explicit* algorithm, since their implementation requires only using the linear algebra tools for objects of finite dimensions.

Remark 2.5. The main difficulty in the practical implementation of the aforementioned formulae is that the sequence $c^{\varkappa}_{-\varkappa}$ can be found, in general cases, only numerically (and thus, the result is computed with some accuracy). Therefore, in such cases, the algorithm would produce the factorization of a matrix polynomial $\tilde{a}(z)$ close enough to the original matrix polynomial $a(z)$ with all consequences of such replacement (possible instabilities as discussed above).

Remark 2.6. If, by any means, one believes that the partial indices of a matrix polynomial in question are stable, some computations from Step 4 can be avoided (see example in §6).

3. An *exact* realization of the explicit algorithm

The analysis of the algorithm, carried out in the previous section, shows the reason for the instability of the factorization problem for matrix polynomials. The explanation is in the instability of the problems of finding the rank and null-space of a matrix. Since finding the rank is the key problem for our algorithm (and also for the Gohberg–Lerer–Rodman algorithm [15]), in the general case, an explicit algorithm cannot be implemented numerically.

However, if Steps (1)–(5) can be carried out in exact arithmetic, i.e. the error-free calculations are used, *problems with the factorization instability, with the accuracy of calculation and rounding errors do not arise.* It should be noted that symbolic calculations have the significant drawback that they often require significant computational resources. In our case, an increase in the size of the matrix function and the degree of the matrix polynomial can lead to a significant slowdown in the algorithm. Fortunately, in most of the applications, matrix functions of the second or third

Table 1. Execution time, t_k, (in seconds) for factorization of the matrix polynomial $C^k(z)$, defined in (5.4). The letter 'F' indicates a failure occurred during the execution, while the symbol '—' highlights that the computations, performed in the symbolic mode, were unable to complete the task within 1 h.

						t_k (symnum)		
k	n_k	p_k	$\varkappa_k$	$\varkappa_2$ (T_0)	t_k (sym)	digits = 5	digits = 10	digits = 15
1	4	6	3	1.10×10	0.28	0.4	0.31	3.44
2	8	12	6	7.95×10	0.81	1.0	0.61	0.62
3	12	18	9	6.04×10^2	2.62	1.4	1.77	1.80
4	16	24	12	5.78×10^3	7.19	F	2.94	2.95
5	20	30	15	5.90×10^4	23.78	F	7.95	8.83
6	24	36	18	5.95×10^5	55.11	F	10.66	13.26
7	28	42	21	5.66×10^6	147.23	F	35.80	38.11
8	32	48	24	5.13×10^7	288.67	F	44.89	42.19
9	36	54	27	4.65×10^8	644.03	F	F	135.55
10	40	60	30	4.44×10^9	1103.06	F	F	150.59
11	44	66	33	4.53×10^{10}	2204.62	F	F	453.78
12	48	72	36	4.66×10^{11}	—	F	F	444.05
13	52	78	39	4.57×10^{12}	—	F	F	F

order mainly appear, and the only reason for a possible slowdown is a large degree of the matrix polynomial. In table 1, we illustrate this feature in detail.

The main result is formulated in theorem 3.2. It gives a necessary and sufficient condition for the *exact* factorization of a matrix polynomial $a(z) \in \mathbb{Q}(i)[z]$, relying on the same property of its determinant, $\det a(z)$. Below, we discuss a practical implementation of those five steps of the *explicit* algorithm in this particular arithmetic.

Input data. A matrix polynomial $a(z) = \sum_{j=0}^{n} a_j z^j$, $a_j \in \mathbb{C}^{p\times p}$, of degree n that is invertible on the unit circle $\mathbb{T}$. The entries of the matrix coefficients a_j belong to the field $\mathbb{Q}(i)$.

Step 1. In order to construct the Wiener–Hopf factorization of the scalar polynomial $\det a(z)$ in *exact* arithmetic, it is sufficient to require that all roots of $\det a(z)$ belong to the field $\mathbb{Q}(i)$. *However, this condition is not necessary.* There exists a weaker necessary and sufficient condition allowing *exact* Wiener–Hopf factorization of $\det a(z)$ over the field $\mathbb{Q}(i)$. The idea here is to factorize the polynomial as a product of irreducible factors, testing (for example, with help of the Schur algorithm) whether the roots of each factor lie in the interior or exterior of the unit disk.

First, we give the criterion in terms of coefficients of the irreducible factors of the polynomial. As has been mentioned above, we assume that no zero lies on the unit circle. Note that, in the opposite case, the Schur test fails during the execution stage. This fact explains why we have not concentrated on how to verify the condition $\det a(z) \neq 0$ on the first step of the algorithm.

Let $Q(z) = q_0 + q_1 z + \cdots + q_N z^N$ be a polynomial with complex coefficients, $q_N \neq 0$. We denote $Q^*(z) = \bar{q}_N + \bar{q}_{N-1} z + \cdots + \bar{q}_0 z^N$, where $\bar{q}_j$ is the complex conjugate of q_j.

For the given $Q(z)$, we define the triangular Toeplitz matrices

$$U_Q := \begin{pmatrix} q_N & \cdots & q_1 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & q_N \end{pmatrix} \quad \text{and} \quad L_Q := \begin{pmatrix} q_0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ q_{N-1} & \cdots & q_0 \end{pmatrix}$$

and form the Hermitian $N \times N$ matrix $C_Q := U_Q^* U_Q - L_Q L_Q^*$. Here, U_Q^* (L_Q^*) is Hermitian conjugate of U_Q (L_Q).

Let $p(z)$ be a polynomial over the field $\mathbb{Q}(i)$. It can be represented as a product of irreducible monic polynomials [28]:

$$p(z) = p_0 Q_1(z) \cdots Q_m(z), \quad 0 \neq p_0 \in \mathbb{Q}(i). \tag{3.1}$$

Theorem 3.1. *The polynomial $p(z)$ admits the exact Wiener–Hopf factorization if and only if, for every polynomial $Q_j(z)$, $j = 1, \dots, m$, one of the matrices C_{Q_j} or $C_{Q_j^*}$ is positive definite.*

Suppose these conditions are fulfilled. Denote by $p_1(z)$ the product of those irreducible factors $Q_j(z)$ in decomposition (3.1) for which the matrices C_{Q_j} are positive definite. Then, the polynomial $p_2(z)$ is the product of the remaining irreducible factors and p_0. Let us define: $\varkappa = \deg p_1(z)$, $p_-(z) = z^{-\varkappa} p_1(z)$ and $p_+(z) = p_2(z)$. Then the exact Wiener–Hopf factorization of the polynomial $p(z)$ is

$$p(z) = p_-(z) z^{\varkappa} p_+(z). \tag{3.2}$$

Proof. It is clear that the polynomial $p(z)$ admits the exact Wiener–Hopf factorization if any of the irreducible factors has all its roots lying inside or outside the unit circle. By the criterion Schur–Cohn [29,30], the polynomial $Q_j(z)$ has all its roots lying inside the unit circle if and only if the matrix C_{Q_j} is positive definite. Similarly, $Q_j(z)$ has all roots lying outside the unit circle if $Q_j^*(z)$ has all its roots lying inside the unit circle, that is $C_{Q_j^*}$ is positive definite. The representation (3.2) is the Wiener–Hopf factorization that is normalized by the condition $p_-(\infty) = 1$. ■

For practical computation, it is more convenient to use the Schur test [31] instead of checking the positive definiteness of matrices. This test states (see, for example, [29]) that

a polynomial $Q(z) = q_0 + q_1 z + \cdots + q_N z^N$ has all its roots lying inside the unit circle if and only if

(i) $|q_0| < |q_N|$, and
(ii) the polynomial $\widetilde{Q}_1(z)$ determined from the equality $\widetilde{Q}_1(z) = z^{-1}(\bar{q}_N Q(z) - q_0 Q^*(z))$ possesses the same property.

Since $\deg \widetilde{Q}_1(z) < \deg Q(z)$, then in a finite number of steps, we arrive at a polynomial of degree 0 or find that the polynomial $Q(z)$ does not have the required property.

In order to verify that all roots of $Q(z)$ lie outside the unit circle, it is necessary to apply the Schur test to the polynomial $Q^*(z)$. Note that the above algorithm automatically verifies the invertibility of the matrix polynomial $a(z)$ on the unit circle $\mathbb{T}$.

Step 2. Since the polynomial $\det a(z)$ admits the *exact* factorization $\det a(z) = \Delta_-(z) z^{\varkappa} \Delta_+(z)$, the coefficients of the polynomial $\Delta_-(z)$ in z^{-1} belong to the field $\mathbb{Q}(i)$. The function $\Delta_-^{-1}(z)$ is analytic in the domain $|z| > r$, for some $r < 1$. Its Laurent coefficients $\delta_j \in \mathbb{Q}(i)$ are obtained by equation (2.4) in *exact* computations. Since only a finite number $(2\varkappa + 1)$ of the coefficients are needed and all operations are arithmetic, Step 2 is therefore performed in the *exact* manner.

Step 3. The matrices $c_{-\varkappa}, \dots, c_{\varkappa}$ defined in equation (2.5) have the entries belonging to $\mathbb{Q}(i)$, i.e. the sequence $c_{-\varkappa}^{\varkappa}$ is *exactly* computed. In Step 3 of the *explicit* algorithm, we find ranks and null spaces of block Toeplitz matrices $T_k(c_{-\varkappa}^{\varkappa})$ for $-\varkappa \leq k \leq \varkappa$. This step can be realized by Gaussian elimination. Since entries of the matrices belong to $\mathbb{Q}(i)$, the Gaussian elimination can be implemented in *exact* arithmetic [28].

Step 4. Obtaining the indices and the factorization essential polynomials of the sequence $c_{-\varkappa}^{\varkappa}$, defined in (2.8) and (2.11), is a foundation stone of the method. Since the ranks of the matrices $T_k(c_{-\varkappa}^{\varkappa})$ have been calculated *exactly*, the differences, Δ_k, and hence the indices, μ_j, are also found *exactly*.

To determine the essential polynomials, we need bases of the complements $\mathcal{H}_{\mu_j+1}^R$, $j = 1, \dots, 2p$, (see equation (2.10) and definition 2.2) and the factorization essential polynomials (see definition 2.3). It is intuitively clear that these polynomials can be found *exactly* using the methods of linear algebra. However, a rigorous proof of this fact is rather complicated and requires a careful study of all steps of the algorithm for determination of the factorization essential polynomials. It will be presented in the separate work.

Step 5. The main result of this section is the following.

Theorem 3.2. *Let $a(z)$ be a matrix polynomial over the field of Gaussian rationales $\mathbb{Q}(i)$. Then $a(z)$ admits exact Wiener–Hopf factorization if the polynomial* $\det a(z)$ *is exactly factorable over $\mathbb{Q}(i)$.*

Proof. If $a(z)$ admits the exact left Wiener–Hopf factorization $a(z) = l_+(z)d_l(z)l_-(z)$, then $\det a(z) = \det l_+(z)z^{\lambda_1+\cdots+\lambda_p} \det l_-(z)$ is the *exact* factorization of $\det a(z)$.

Conversely, if the polynomial $\det a(z)$ admits the *exact* factorization then all preliminary Steps 1–4 can be carried out exactly. It remains to use the formulae (2.12)–(2.14). ■

4. Implementation of the exact algorithm in Maple's code: ExactMPF package

The *exact* factorization algorithm has been realized in the computer algebra system Maple using its linear algebra utilities as the package ExactMPF. The package can work in the version Maple 11 or higher. Clearly, one can do this in any other suitable software (for example, Mathematica[2]). The main procedure of the package is SolverExactMPF. During the execution, it calls a number of subroutines. We list the basic ExactMPF's procedures:

— **SchurTest** verifies whether all polynomial roots lie in the unit circle (see §3, Step 1).
— **SolverExactPF** verifies whether a scalar polynomial admits the exact Wiener–Hopf factorization and builds it (see §3, Step 1).
— **ExactMLC** computes the matrix Laurent coefficients $c_{-\varkappa}, \ldots, c_0, \ldots, c_\varkappa$ for the rational matrix function $\Delta_-^{-1}(z)a(z)$ (see equation (2.5) and §3, Step 2).
— **seqTk** builds the sequence of the matrices $T_k(c_{-\varkappa}^{\varkappa})$, $-\varkappa \le k \le \varkappa$ (§3, Step 3).
— **ExactFEP** finds indices and factorization essential polynomials of the sequence $c_{-\varkappa}^{\varkappa}$.

The main procedure SolverExactMPF finds the exact factorizations of a $p \times p$ Laurent matrix polynomial $a(z) = \sum_{k=-m}^{n} a_k z^k$. It also checks and validates the input entered into the procedure. The following validations are performed:

— are elements of the matrix $a(z)$ polynomials in z, z^{-1}?
— are these polynomials over the field $\mathbb{Q}(i)$?
— does $\det a(z)$ admit the exact Wiener–Hopf factorization?

If the validation fails, the program is interrupted.

To access ExactMPF use the commands.

```
> read("ExactMPF.txt");
> with(ExactMPF);
> with(LinearAlgebra);
```

To obtain the factorizations of $a(z)$, we run the procedure SolverExactMPF with argument $a(z)$.

```
> lplus, dl, lminus, rminus, dr, rplus := SolverExactMPF(a):
```

The procedure SolverExactMPF returns the factors lplus, dl, lminus of the left factorization and the factors rminus, dr, rplus of the right factorization.

This procedure also automatically performs a few tests for the constructed factorization

(i) it finds the sum of the partial indices,
(ii) checks whether the factors $l_+(z), r_+(z)$ ($l_-(z), r_-(z)$) are matrix polynomials in z (in z^{-1}) and
(iii) whether $(\Delta_\pm(z))^{-1} \det l_\pm(z)$, $(\Delta_\pm(z))^{-1} \det r_\pm(z)$ are constants.

[2]https://www.wolfram.com/mathematica/.

A very peculiar option symnum is also built into the procedure SolverExactMPF. It can be evaluated by the command:

```
> lplus, dl, lminus, rminus, dr, rplus := SolverExactMPF(a,"symnum"):
```

In this mode, SolverExactMPF converts the matrices $T_k(c^{\varkappa}_{-\varkappa})$ to matrices with floating point entries to the precision given by the global variable digits. Then, SolverExactMPF uses the procedure NumRank included in the package ExactMPF.

NumRank finds a numerical rank of a $N \times M$ matrix A with float point entries. A good practice is to use a numerical rank, which is defined with the help of the SVD method [24,27]. Recall some necessary definitions.

Let A be a $N \times M$ matrix and $\sigma_1 \geq \cdots \geq \sigma_s \geq 0$, $s = \min\{N, M\}$, its singular numbers. In the case of exact computation, if rank $A = r$, then $\sigma_r \neq 0$ and $\sigma_{r+1} = \cdots = \sigma_s = 0$. The number $k_2(A) = \sigma_1/\sigma_r$ is called *the spectral condition number* of the rectangular matrix A [27,32].

It is known that perturbations of the singular values are of the same order of magnitude as the perturbations of the matrix entries. Thus, computation of the singular values is a stable procedure and this property has led to the definition of a numerical τ-rank (see [24,27]).

A numerical τ-rank of the matrix function $A(z)$ is the number of singular values greater than some tolerance τ, i.e. the singular values that are less than or equal to τ considered as perturbations of the zero singular values.

Thus, the tolerance τ specifies a way of separating the non-zero singular numbers from those assumed to be zero before the perturbation. Hence, the notion of τ-rank is useful when there exists a well-defined gap between non-zero and zero singular values of the matrix. Even for a matrix of the full rank, this requirement may not be fulfilled (if the matrix is ill-conditioned related to the spectral condition number). Choice of the parameter τ may be very sensitive and thus it is the most delicate and difficult task to decide upon in calculation of the numerical τ-rank. In the procedure **NumRank**, a default value of the tolerance is $\tau(A) = \max\{N, M\} \cdot \sigma_1(A) \cdot 10^{-\text{digits}}$.

Further, τ-ranks of the block Toeplitz matrices T_k are used to define τ-indices of the sequence $c^{\varkappa}_{-\varkappa}$ instead of the *exact* indices of this sequence. All remaining calculations are performed in the *exact* arithmetic. If the output is validated successfully, the replacement of the indices by τ-indices may be considered as justified. Using this mode, one can significantly speed up the calculations. Moreover, this mode allows us to perform numerical experiments to test a coincidence of the partial indices and the partial τ-indices that were found by τ-indices of the sequence $c^{\varkappa}_{-\varkappa}$. In the next section, we perform such experiments.

After this short description of the ExactMPF package, we come back to matrix polynomials $\mathcal{P}_1(z)$ and $\mathcal{P}_2(z)$ discussed in the Introduction and try to factorize them *exactly* using the package.

Example 4.1. Consider first the matrix function $a(z) = \mathcal{P}_1(z)$ defined in (1.2) and execute

```
> SolverExactMPF(a):
```

'Failure: The exact polynomial factorization doesn't exist'
'The factorization process is interrupted'

Example 4.2. Consider the second matrix polynomial $a(z) = \mathcal{P}_2(z)$ from (1.2) that allows for the *exact* factorization of its determinant over the field $\mathbb{Q}(i)$. The procedure SolverExactMPF gives in this case the following expressions for the factors of $a(z)$:

```
> lplus; dl; lminus;
```

$$\begin{bmatrix} -4z - \frac{1099}{116} & 2z - \frac{43}{29} \\ z^2 + \frac{93}{58}z + \frac{48}{29} & \frac{2}{29}z + \frac{32}{29} \end{bmatrix}, \quad \begin{bmatrix} z & 0 \\ 0 & z \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} -\frac{2}{29z} & 1 - \frac{35}{58z} \\ 1 + \frac{3}{29z} & \frac{1051}{232z} \end{bmatrix},$$

```
> rminus; dr; rplus;
```

$$\begin{bmatrix} 0 & -4+\frac{2}{z}-\frac{1}{z^2} \\ \frac{35}{22} & \frac{3}{2}+\frac{3}{4z}+\frac{4}{z^2} \end{bmatrix}, \quad \begin{bmatrix} 1 & 0 \\ 0 & z^2 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} \frac{44}{35}+\frac{121}{140}z+\frac{33}{70}z^2 & \frac{11}{70}z-\frac{11}{35}z^2+\frac{22}{35}z^3 \\ -\frac{1}{2} & 1 \end{bmatrix}.$$

The execution time is 0.265 s.

Note that the left and right partial indices do not coincide. Moreover, the set of partial indices in the left factorization is stable, while that for the right factorization is unstable. In the next section, we carry out more extensive numerical experiments with ExactMPF.

5. Examples of matrix factorization using the ExactMPF package

All computations have been performed on a home desktop computer HP with Intel(R) Core(TM)i3-415T CPU, 3.00 GHz, 4G RAM, operating the system Windows 10.

The package ExactMPF can be used for study of the *explicit* factorization of matrix polynomials. In the electronic supplementary material, we include numerous tests performed with use of the ExactMPF package, containing solutions of all examples from the main body of the paper.

(a) Factorization of triangular matrices

First, we consider factorization of triangular matrix functions. In [33], the right factorization of triangular 2×2 matrix functions $A(t)$ is studied in detail and the stability analysis of this problem is carried out. It has been proven that, for a non-trivial case, the factorization of $A(t)$ is reduced to the factorization of the following Laurent matrix polynomial

$$\mathcal{P}(t) = \begin{pmatrix} t^{\nu_1} & 0 \\ \sum_{j=\nu_1+1}^{\nu_2-1} a_j t^j & t^{\nu_2} \end{pmatrix}, \tag{5.1}$$

where $\nu_2 \geq \nu_1 + 3$. If $a_j \in \mathbb{Q}(i)$, factorization of such a matrix can be naturally constructed with the help of our package.

Example 5.1. Consider

$$\mathcal{P}(z) = \begin{pmatrix} \frac{1}{z^3} & 0 \\ -\frac{1}{z^2}-\frac{4}{5z}+1-z^2+z^3 & z^4 \end{pmatrix}.$$

Then, after execution of SolverExactMPF, we get

```
> rminus, dr, rplus;
```

$$\begin{bmatrix} 1+\frac{4}{z}+\frac{21}{5z^2}+\frac{16}{5z^3} & \frac{1}{z^2}+\frac{1}{z^3}+\frac{1}{z^4} \\ -\frac{104}{25}-\frac{169}{25z}-\frac{16}{5z^2} & \frac{1}{5}-\frac{4}{5z}-\frac{9}{5z^2}-\frac{1}{z^3} \end{bmatrix} \quad \begin{bmatrix} 1 & 0 \\ 0 & z \end{bmatrix}$$

and
$$\begin{bmatrix} -5z & -5z^2-5z-5 \\ 5z^2+15z+1 & 5z^3+20z^2+21z+16 \end{bmatrix},$$

```
> lplus, dl, lminus;
```

$$\begin{bmatrix} 0 & 1 \\ 1 & -z-\frac{4}{5}z^2+z^3-z^5+z^6 \end{bmatrix}, \quad \begin{bmatrix} z^4 & 0 \\ 0 & z^{-3} \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

The execution time is 0.547 s. We note that in this case, the left factorization can be easily constructed without the package, but with another order for the left partial indices

$$\begin{bmatrix} 1 & 0 \\ -z-\frac{4}{5}z^2+z^3-z^5+z^6 & 1 \end{bmatrix} \quad \begin{bmatrix} z^{-3} & 0 \\ 0 & z^4 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

As is clear from the presented example, the indices of the diagonal elements in the triangular matrix do not always coincide with its partial indices [33].

The factorization of triangular matrix functions of an arbitrary order requires considerations of a number of special cases, depending on the relation between the indices of the diagonal elements [34]. The procedure SolverExactMPF can be useful for experimenting with such matrices.

Example 5.2. Let

$$\mathcal{A}(z) = \begin{pmatrix} z-\frac{1}{2} & 0 & 0 \\ 0 & \frac{z^2+13z+15}{z^2} & 0 \\ z-1 & z^{-1} & 1 \end{pmatrix}.$$

Here, the indices of diagonal elements are $\varkappa_1 = 1$, $\varkappa_2 = -2$, $\varkappa_3 = 0$. SolverExactMPF returns the following data:

```
> lplus, dl, lminus;
```

$$\begin{bmatrix} 2 & 1 & 0 \\ 0 & 0 & z^2+13z+15 \\ 2 & 2 & z \end{bmatrix}, \quad \begin{bmatrix} z & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & z^{-2} \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} \frac{1}{2} & 0 & -\frac{1}{2z} \\ -\frac{1}{2} & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Since the expression for the right factors $r_\pm(z)$ are cumbersome (see the electronic supplementary material for full details), we present only the middle factor

```
> dr
```

$$\begin{bmatrix} z^{-1} & 0 & 0 \\ 0 & z^{-1} & 0 \\ 0 & 0 & z \end{bmatrix}.$$

The executing time is 0.563 s.

Again, both the left and right partial indices of this matrix function do not coincide with the indices of the diagonal elements.

(b) Factorization of a matrix polynomial with complex coefficients

In the following example, we construct the *exact* factorization of a matrix polynomial with complex coefficients.

Example 5.3.

$$B(z) = \begin{pmatrix} 2iz^2 - z + 1/2 - z^{-1} & -1 \\ z^2 - iz + 2 + 4iz^{-1} & z^3 - iz^2 + 2z + 4i \end{pmatrix}.$$

Then, we get

```
> lplus, dl, lminus, dr;
```

$$\begin{bmatrix} 2i & -1 \\ 0 & z^3 - iz^2 + 2z + 4i \end{bmatrix} \begin{bmatrix} z^2 & 0 \\ 0 & 1 \end{bmatrix}, \quad \begin{bmatrix} 1 + (i/2)z^{-1} - (i/4)z^{-2} & 0 \\ z^{-1} & 1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} z & 0 \\ 0 & z \end{bmatrix}.$$

The execution time is 0.438 s.

For the same reason as above, we present the factors only for the left factorization.

(c) Experiments with the τ-factorization of matrix polynomials

As have been already underlined, the main unsolved problem in the approximate factorization of matrix polynomials is a verification whether its partial τ-indices coincide with the partial indices of the original matrix (both left and right ones). Since it has been impossible so far to prove any definite result theoretically, we carry out here numerical experiments using our designed solver SolverExactMPF in its symbolic-numeric mode symnum for the cases when exact factorization can be performed.

In the next example, we introduce a sequence of *exact* factorizable matrix polynomials with increasing degrees. We use the full capability of SolverExactMPF in both its modes (exact and symbolic-numeric) to calculate their partial indices and the partial τ-indices (see §4). Then we carry out all subsequent calculations in the *exact* arithmetic. If the obtained factors $l_\pm(z)$, $r_\pm(z)$ and the partial indices pass all the necessary verifications, then the factorizations are constructed correctly and the replacements of the ranks by their numerical τ-ranks is justified. Moreover, using option symnum mode gives an additional bonus: it significantly speeds up the computations.

The purpose of our experiments is to test applicability of the notion of the τ-rank for the factorization of matrix polynomials. *A priori*, it is clear that the method is ineffective if the task of computation of the τ-ranks for the sequence of block Toeplitz matrices $T_{-\varkappa}, \dots, T_0, \dots, T_\varkappa$ is extremely ill-conditioned. Let us define an analogue of the condition number for this task.

As usual, if A is a rectangular matrix, then $\varkappa_2(A) = \sigma_1/\sigma_r$, $r = \operatorname{rank} A$, is the condition number of A with respect to the spectral matrix norm. We say that $K_2 = \max_{-\varkappa \le j \le \varkappa} \varkappa_2(T_j)$ is the condition number of the matrix sequence $\{T_j\}_{j=-\varkappa}^{\varkappa}$.

Since the τ-rank can be found only numerically, we are able only to estimate the value of K_2. It is easy to verify that rank $T_0 = \varkappa(p-1) + p$. Hence, we can explicitly calculate $\varkappa_2(T_0)$ and can find a lower bound for $K_2 \ge \varkappa_2(T_0)$. As has been mentioned above, a choice of parameter τ, to be used in τ-factorization, is not trivial. We use in our analysis the value $\tau(T_j) = \operatorname{maxsize}(T_j) \cdot \sigma_1(T_j) \cdot 10^{-\text{digits}}$ as it is the same as the default tolerance in MATLAB'S rank command. We can prove that

$$\left.\begin{aligned} &\operatorname{maxsize}(T_j) \le (2\varkappa + 1)p, \quad \sigma_1(T_j) = ||T_j||_F \le ||T_\varkappa||_F = \sqrt{|c_{-\varkappa}|^2 + \cdots + |c_\varkappa|^2}, \\ &-\varkappa \le j \le \varkappa, \end{aligned}\right\} \tag{5.2}$$

where $||\cdot||_F$ is the Frobenius norm of a matrix.

Taking into account (5.2), we see that

$$\tau \le (2\varkappa + 1)p\sqrt{|c_{-\varkappa}|^2 + \cdots + |c_\varkappa|^2} \cdot 10^{-\text{digits}}, \tag{5.3}$$

chosen as the tolerance τ is controlled by the value of parameter digits.

In the next example, we will see how the correctness of such an approach depends on the choice of the parameter digits.

Example 5.4. The matrix polynomial

$$C(z) = \begin{pmatrix} 5iz^2 - 3z + 1 & -10iz^2 + 6z - 2 \\ z & z^4 + z^3(-1-i) + 2z^2 - \frac{13z}{2} \end{pmatrix} \tag{5.4}$$

admits the exact factorization. Let us consider matrix polynomials $C_k(z) = C^k(z)$ and denote $n_k = \deg C_k(z)$, $p_k = \deg \det C_k(z)$, $\varkappa_k = \text{ind} \det C_k(z) = k\varkappa$. We also record the execution time, t_k, required by the package to compute factorization of the matrix polynomial $C_k(z)$. Three different machine precisions: digits $= 5$, 10 or 15 are used for the testing.

The results of the numerical experiment are shown in table 1. Expressions for the factors $l_\pm(z)$, $r_\pm(z)$ are very cumbersome and are omitted.

The test has demonstrated that the symbolic part of the ExactMPF package is able to deliver the results in reasonable time for $1 \le k \le 11$ and the matrix polynomials $C_k(z)$ have the stable sets of their partial left and right indices. Those are

$$\left.\begin{aligned} &\lambda_1 = \lambda_2 = \rho_1 = \rho_2 = \frac{k\varkappa}{2}, \quad k \text{ is even,} \\ &\lambda_1 = \rho_2 = \frac{k\varkappa + 1}{2}, \quad \lambda_2 = \rho_1 = \frac{k\varkappa - 1}{2}, \quad k \text{ is odd.} \end{aligned}\right\} \tag{5.5}$$

From table 1, we can also see that the executions in the sym-mode, when the ranks are calculated in *exact* arithmetic, become very slow for large values of k and the use of symbolic-numeric calculation (symnum-mode) can greatly accelerate the computations.

In the cases indicated by the letter F in table 1, the procedure NumRank has found the ranks of T_k incorrectly (the validation of the sum of the indices for the sequence $c_{-\varkappa}^{\varkappa}$ fails), and the ExactMPF package interrupts computations because of the failure. The reason for this is that the condition number K_2 was too large. In this case, increasing parameter digits (that decreases the threshold for numerical tolerance, τ) makes it possible to continue correct calculations. Note that the partial indices computed for $k = 12$ with use of the τ-factorization follow the same rule as in (5.5). Unfortunately, this fact does not formally allow us to conclude (with 100% confidence) that the *exact* partial indices would be the same.

The next example shows that symnum mode is able to correctly calculate the partial indices even in unstable cases.

Example 5.5. Let us consider a matrix polynomial

$$A = \begin{pmatrix} 36z^2 + 17z - 14 & 0 \\ -z^2 + 3z & z^2 + 13z + 15 \end{pmatrix}.$$

The *exact* computations using SolverExactMPF give $\lambda_1 = \lambda_2 = 1$ and $\rho_1 = 0$, $\rho_2 = 2$. Moreover, in this case, $\varkappa_2(T_0) = 5.294083$.

Next, let us construct another matrix polynomial $B = A^8$. For this one, SolverExactMPF returns the following values of the partial indices: $\lambda_1 = 14$, $\lambda_2 = 2$ and $\rho_1 = 0$, $\rho_2 = 16$. In the symnum mode with the parameter digits=10, the package was unable to find the correct values since the tests on the sums of indicies are not successful and the factorization process is interrupted. In this case, $\varkappa_2(T_0) = 6.465447 \times 10^{10}$ and, as a result, the condition number $K_2 \ge 6.465447 \times 10^{10}$ is too large. If, however, we set digits $= 15$, the package returns, in the symnum mode, the correct values of the partial indices.

Thus, our numerical experiments with several matrix polynomials (see electronic supplementary material) have shown that, for the case of not too large condition numbers, calculations with use of both the symbolic and numeric options of the ExactMPF package give identical partial indices and partial τ-indices, even in the unstable cases. This fact gives hope for a stable construction of an approximate factorization but requires a solid theoretical proof.

6. On numerical realization of the explicit algorithm

In previous sections, we considered the problem of the Wiener–Hopf factorization for matrix polynomial over the field $\mathbb{Q}(i)$. In this case,

(i) the input data are given with infinite precision, and
(ii) there is a possibility to use the *exact* (error-free) calculations.

As a result, we have constructed the ExactMPF package for the *exact* solution of the factorization problem, that returns the solution under condition that the determinate of the matrix polynomial is *exactly* factorizable or, in the opposite case, interrupts the computations. In this case, the stability and accuracy problems do not arise. However, a natural question arises: can we deliver an approximate (numerical) factorization of such a matrix? If the conditions for the existence of the exact factorization are not fulfilled, then it is not yet possible to build any algorithm for the factorization problem that satisfies the entire canon of computational mathematics (with carrying stability analysis, obtaining forward/backward errors and accuracy for approximate factorization).

Here, we highlight possible obstacles when implementing the *explicit* algorithm (Steps 1–5) numerically *without relating the steps to the ExactMPF* package, in the case, when, at least, one of the conditions (i), (ii) is not fulfilled.

Input data. We consider a matrix polynomial $a(z)$ of the degree n and size $p \times p$ that is invertible on the unit circle $\mathbb{T}$. The entries of the matrix coefficients, a_j, belong to the field $\mathbb{C}$. They can be represented by floating point numbers with the accuracy 10^{-q} (finite precision) or the entries belong to the field $\mathbb{Q}(i)$ (infinite precision).

Step 1. Since one of the conditions (i), (ii) is not fulfilled, we can solve the factorization problem for the function $\Delta(z) = \det a(z)$ only approximately. If the input data are given with infinite precision we convert them in the floating point format with a desired accuracy 10^{-q}.
A standard way for the factorization of a scalar polynomial, recommended by all explicit methods, suggests using approximate values of the polynomial roots. However, those roots are, in general, not well-conditioned functions of polynomial coefficients (see, e.g. [35]), and vice versa, the coefficients of a polynomial are also not well-conditioned functions of their roots [36]. The latter means that searching numerically for zeros of the polynomial to construct its factorization (2.1) may lead to the algorithm instability.
In order to solve the factorization problem for scalar polynomials over the field $\mathbb{C}$, a dedicated package PolynomialFactorization (in the Maple Software) has been developed by Adukov [37]. It factorizes an input polynomial $p(z)$ based on the indices and essential polynomials technique. Input data for its main procedure SolverPF is a scalar polynomial $p(z)$ and the accuracy 10^{-q} of the polynomial coefficients. SolverPF returns factors $p_-(z)$, $p_+(z)$ of the Wiener–Hopf factorization of $p(z)$ and their *guaranteed* accuracy ε. The accuracy, ε, calculated by the procedure SolverPF depends on a type of polynomial and the accuracy of the input data.
Note, that the PolynomialFactorization package is not included in the ExactMPF package, and is different from the standard 'Factoring a Polynomial' Maple command Factor.[3] Its main feature is: it does not rely on computation of the polynomial's roots.

Step 2. Now we can calculate the Laurent coefficients δ_j of $\Delta_-^{-1}(z)$ using given matrix coefficients a_j of $a(z)$ and taking into account their accuracy.Hence, we can find the sequence $c_{-\varkappa}, \ldots, c_{\varkappa}$ with some accuracy as described in §2. This step does not contain any technical difficulties. Note that, if the input data are given with infinite precision, i.e. belong to the field $\mathbb{Q}(i)$, then we can calculate c_j with any reasonable accuracy. Otherwise, the accuracy is restricted by finite accuracy of input data.

[3]https://www.maplesoft.com/support/help/maple/view.aspx?path=Task/FactoringAPolynomial.

Step 3. This is a crucial point in the factorization problem. In the *exact* arithmetic, the respective calculations do not meet any problem (except of computational time). However, if the entries of the matrices are known only approximately, evaluations of the ranks and null spaces of T_k might be unstable. This is a reason for the instability of the factorization problem itself.

In §4, we have discussed the procedure **NumRank** from ExactMPF in detail. It uses the SVD method for finding a numerical rank of matrices with floating point entries. The procedure can also be used in this case.

For definition of the essential polynomials, we need a method for evaluation of a basis of the null space for matrices with floating point entries. This can also be done with the SVD method [24,27]. Computation practice shows that this method is the most reliable way to numerically find the rank and the basis for the null space of a matrix. Here, however, the main problem is how to select the computational tolerance τ. Unfortunately, in general, there is no rigorous way to do this. As an initial choice, we can recommend using $\tau(A) = \max\{N, M\} \cdot \sigma_1(A) \cdot 10^{-\text{digits}}$, if one works in the Maple environment.

Step 4. Since entries of the matrices T_k $(-\varkappa \le k \le \varkappa)$ are, in general, floating point numbers, we need to compute the numerical τ-rank of the sequence $c_{-\varkappa}^{\varkappa}$ instead of *exact*. The indices of the sequence $c_{-\varkappa}^{\varkappa}$ found by τ-ranks are respectively called τ*-indices*.

Next, to construct an approximate Wiener–Hopf factorization of the matrix polynomial, we need to find the factorization essential polynomial of the sequence $c_{-\varkappa}^{\varkappa}$. In §3, we describe an algorithm for construction of these polynomials and prove that, for input data with infinite precision, the algorithm can be realized in the exact arithmetic. An open problem, however, is how to adapt the algorithm for input data with a finite precision. In such cases, we may use a basis of null space for matrix T_k $(-\varkappa \le k \le \varkappa)$ via the SVD method with the tolerance τ (τ-bases). The factorization essential polynomials, constructed with the help of τ-bases, are called factorization essential τ-polynomials.

Step 5. For the fixed tolerance $\tau > 0$, the integers $\lambda_1^\tau, \ldots, \lambda_p^\tau$, $\rho_1^\tau, \ldots, \rho_p^\tau$, computed by formulae (2.12) with the help of τ-indices of the sequence $c_{-\varkappa}^{\varkappa}$ are called partial τ-indices of the matrix polynomial $a(z)$. The factorization of $a(z)$ constructed by formulae (2.12)–(2.14) with the help the partial τ-indices and the factorization essential τ-polynomials will be called τ-factorization of the matrix polynomial $a(z)$.

The key question here remains open:

Does the τ-factorization of a matrix polynomial $a(z)$ with a chosen τ represent an approximate factorization of $a(z)$ (preserving the same set of the partial indices) with a certain accuracy?

To solve this problem, we must find answers to the following basic questions:

(i) does a tolerance τ exist such that the partial τ-indices of $a(z)$ coincide with the matrix partial indices? How can this tolerance be found?

(ii) can the algorithm for constructing factorization essential polynomials from §3 be adapted to find the factorization essential τ-polynomials?

In order to carry out numerical experiments, we should develop a new appropriate software. At the present time, we have in our hands the package PolynomialFactorization from [37] and the procedure NumRank from this ExactMPF package. Since the procedure for finding the factorization essential τ-polynomials is still unavailable, in the following example, we restrict ourselves to the evaluation of partial τ-indices only.

We consider the matrix polynomial $\mathcal{P}_1(z)$ from the Introduction in (1.2). Note that the input data have infinite precision, while the determinant $\det \mathcal{P}_1(z)$ can be factorized only approximately. Let us first construct a factorization of $p(z) = \det \mathcal{P}_1(z)$ with the help of the package PolynomialFactorization to obtain

$$p_1(z) = 1.00000000z^2 - 0.4572987z + 0.1301582$$

and

$$p_2(z) = 2.0000000z^3 + 1.9145973z^2 + 5.8652264z + 7.6829597,$$

with a precision $\epsilon = 0.0000034$. This precision was returned by the package PolynomialFactorization proceeded under the Maple value of $\mathtt{digits} = 15$ and accuracy $10^{-q} = 10^{-15}$.

Thus, an approximate factorization (2.1) has been constructed with the index $\varkappa = \det \mathcal{P}_1 = \deg p_1 = 2$, and the factors $\Delta_-(z) = z^{-2}\, p_1(z)$ and $\Delta_+(z) = p_2(z)$.

The Laurent coefficients of $\Delta_-^{-1}(z)$ are found directly from the recurrence formulae (2.4)

$$\delta_0 = 1,\ \delta_1 = 0.4572986,\ \delta_2 = 0.0789639, \delta_3 = -0.0234111,$$
$$\delta_4 = -0.0209836 \quad \text{and} \quad \delta_5 = -0.0065486.$$

Calculation of the matrix Laurent coefficients by formulae (2.5) gives the following result:

$$c_{-2} = \begin{pmatrix} 0.001184 & -0.041851 \\ -0.023411 & 0.264908 \end{pmatrix} \quad \text{and} \quad c_{-1} = \begin{pmatrix} -0.011461 & -0.205726 \\ 0.078963 & 1.863760 \end{pmatrix}$$

and

$$c_0 = \begin{pmatrix} -0.049372 & -0.401256 \\ 0.457298 & 4.512850 \end{pmatrix}, \quad c_1 = \begin{pmatrix} -0.085404 & 0.170808 \\ 1.000000 & 1.536260 \end{pmatrix}$$

and $$c_2 = \begin{pmatrix} 2.000000 & -4.000000 \\ 0.000000 & 1.45730 \end{pmatrix}.$$

In order to verify the stability criterion for matrix polynomials (theorem 3.1 in [23]), we need to compute the τ-ranks of matrices

$$T_{-1} = \begin{pmatrix} c_{-1} & c_{-2} \\ c_0 & c_{-1} \\ c_1 & c_0 \\ c_2 & c_1 \end{pmatrix} \quad \text{and} \quad T_1 = \begin{pmatrix} c_1 & c_0 & c_{-1} & c_{-2} \\ c_2 & c_1 & c_0 & c_{-1} \end{pmatrix}.$$

This can be done with use of the function SingularValues(A) of the MAPLESOFTWARE that returns the singular values of a matrix A. For the 8×4 matrix T_{-1}, it gives the following singular values:

$$\sigma_1 = 7.449916, \quad \sigma_2 = 4.432173, \quad \sigma_3 = 1.349779,$$
$$\sigma_4 = 0.737172 \quad \text{and} \quad \sigma_5 = \sigma_6 = \sigma_7 = \sigma_8 = 0.0.$$

and, therefore, $\tau(T_{-1}) \approx 6.0 \times 10^{-14}$ for digits = 15.

Since the gap between σ_4 and σ_5 is much larger than the above tolerance, we can conclude that τ-rank$(T_{-1}) = 4$. Since $k_2(T_{-1}) = \sigma_1/\sigma_4 \approx 10.11$, it appears that T_{-1} is a well-defined matrix of the full τ-rank.

Similarly, the 4×8 matrix T_1 has the following non-zero singular values:

$$\sigma_1 = 7.052308, \quad \sigma_2 = 4.295688, \quad \sigma_3 = 3.050511 \quad \text{and} \quad \sigma_4 = 0.130922,$$

while $\tau(T_1) \approx 5.64 \times 10^{-14}$ and $k_2(T_1) \approx 53.87$. Hence, T_1 also is a well-defined matrix of the full τ-rank.

Let us represent $\varkappa = \operatorname{ind} \det \mathcal{P}_1$ in the form $\varkappa = sp + r$, where s and $0 \le r < p$ are integers. In this case $\varkappa = p = 2$, i.e. $s = 1$, $r = 0$. Then, by the above-mentioned theorem 3.1 from [23], the left (right) partial indices of $\mathcal{P}_1(z)$ are stable if and only if $\operatorname{rank} T_{-s} = (\varkappa + 1)p - \varkappa(\operatorname{rank} T_s = (\varkappa + 1)p - \varkappa)$, that is: $\operatorname{rank} T_{-1} = 4$ $(\operatorname{rank} T_1 = 4)$. Thus, we can conclude that τ-indices are $\lambda_1^\tau = \lambda_2^\tau = 1$ and $\rho_1^\tau = \rho_2^\tau = 1$. Finally, since the matrices T_{-1}, T_1 are well-conditioned, *we can expect* that the partial indices of $\mathcal{P}_1(z)$ are also stable and equal to those computed numerically. But this needs anyway a rigorous proof.

7. Conclusion

We have used the *explicit* algorithm for the *exact* Wiener–Hopf factorization of a matrix polynomial $a(z)$ with rational coefficients and with exact factorizable $\det a(z)$ and described the main steps in its realization within the ExactMPF package. Its listing is included into the electronic supplementary material.

A message delivered by this paper is twofold: (a) the existence of an *explicit* algorithm does not necessarily guarantee that the factorization can be effectively performed; and, on the other hand, (b) in contrast to the general opinion, it is indeed possible to find sometimes *exact* values of the partial indices and then to perform *exact* factorization, even though the indices are not stable.

One of the main results of this paper is the necessary and sufficient condition for existence of the *exact* factorization for matrix polynomials whose coefficients are matrices with entries from the Gaussian rational field $\mathbb{Q}(i)$. It turns out that a matrix polynomial $a(z)$ admits the *exact* Wiener–Hopf factorization if and only if the polynomial $\det a(z)$ is *exactly* factorable.

The second (and extremely useful) result of the paper is the development of the package ExactMPF in the Maple software that realizes factorization algorithm *exactly* for a matrix polynomial with the coefficients from the Gaussian rational field $\mathbb{Q}(i)$ and the exact factorizable determinant.

The use of the ExactMPF package is manyfold. The authors confidently declare that

— it can be used for symbolic or symbolic-numeral factorization of matrix polynomials.
— to analyse stability of the partial indices of matrix polynomials.
— to make tests and experiments for verification of proved statements and for a motivation to formulate hypothesis or conjectures for further theoretical development of the theory.

They also believe that it may be useful with additional efforts

— in some cases to build an approximate factorization of a given matrix polynomial with arbitrary complex coefficients and
— to solve real life problems as a computational tool. We shall demonstrate this in the accompanying paper dealing with solving of a discrete analogue of nonlinear Schrödinger equation by the inverse scattering transform method.

Data accessibility. The data are provided in electronic supplementary material [38].

Authors' contributions. V.M.A.: conceptualization, investigation, methodology, project administration, software, supervision, validation, writing—original draft; N.V.A.: conceptualization, investigation, methodology, software, validation, writing—original draft; G.M.: conceptualization, funding acquisition, investigation, methodology, project administration, software, supervision, validation, writing—original draft.

All authors gave final approval for publication and agreed to be held accountable for the work performed therein.

Conflict of interest declaration. We declare we have no competing interests.

Funding. V.M.A. and N.V.A. were supported by RFBR grant no. 20-41-740024. G.M. has been supported by funding from the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement EffectFact no. 101008140. This programme was supported by EPSRC grant no. EP/R014604/1. G.M. is also thankful to the Royal Society for the Wolfson Research Merit Award and to the Welsh Government for the Future Generation Industrial Fellowship.

Acknowledgements. The authors would like to thank the Isaac Newton Institute for Mathematical Sciences for the Simon Fellowship and for all support and hospitality during the WHT programme 'Bringing pure and applied analysis together via the Wiener–Hopf technique, its generalizations and applications', where work on this paper was initiated.

References

1. Clancey KF, Gohberg I. 1981 *Factorization of matrix functions and singular integral operators*. Operator Theory, Advances and Applications, vol. 3. Basel: Springer.

2. Prössdorf S. 1978 *Some classes of singular equations*. Series North-Holland Mathematical Library, vol. 17 Amsterdam, the Netherlands; New York, NY: North-Holland Pub. Co.
3. Speck F-O, 1985. *General Wiener-Hopf factorization methods*. Research Notes in Mathematics, vol. 119 AMS. Boston, London, Mulbourn: Pitman Advanced Publishing Program.
4. Abrahams ID. 2002 On the application of the Wiener–Hopf technique to problems in dynamic elasticity. *Wave Motion* **36**, 311–333. (doi:10.1016/S0165-2125(02)00027-6)
5. Daniele VG, Lombardi G. 2020 *Scattering and diffraction by wedges 1: the Wiener-Hopf solution theory*, vol. 1. London, UK: Wiley-ISTE.
6. Daniele VG, Zich RS. 2014 *The Wiener–Hopf method in electromagnetics*. ISMB Series. Edison, New York, NY: SciTech Publishing.
7. Kisil AV, Abrahams ID, Mishuris G, Rogosin SV. 2021 The Wiener–Hopf technique, its generalizations and applications: constructive and approximate methods. *Proc. R. Soc.* **477**, 20210533. (doi:10.1098/rspa.2021.0533)
8. Lawrie JB, Abrahams ID. 2007 A brief historical perspective of the Wiener–Hopf technique. *J. Eng. Math.* **59**, 351–358. (doi:10.1007/s10665-007-9195-x)
9. Abrahams ID, Huang X, Kisil A, Mishuris G, Nieves M, Rogosin S, Spitkovsky I. 2021 Reinvigorating the Wiener-Hopf technique in the pursuit of understanding processes and materials. *Natl. Sci. Rev.* **8**, nwaa225. (doi:10.1093/nsr/nwaa225)
10. Rogosin SV, Mishuris G. 2016 Constructive methods for factorization of matrix-functions. *IMA J. Appl. Math.* **81**, 365–391. (doi:10.1093/imamat/hxv038)
11. Gakhov FD. 1990 *Boundary value problems*. New York, NY: Dover Publ. Inc.
12. Gohberg IC, Krein MG. 1958 Systems of integral equations on a half-line with kernels depending on the difference of arguments. *Uspekhi Mat. Nauk.* **13**, 3–72.
13. Adukov VM. 1993 Wiener–Hopf factorization of meromorphic matrix-valued functions. *St. Petersbg. Math. J.* **4**, 51–69.
14. Adukov VM. 1999 Factorization of analytic matrix-valued functions. *Theor. Math. Phys.* **118**, 255–263. (doi:10.4213/tmf704)
15. Gohberg IC, Lerer L, Rodman L. 1978 Factorization indices for matrix polynomials. *Bull. Am. Math. Soc.* **84**, 275–277. (doi:10.1090/S0002-9904-1978-14473-5)
16. Giorgadze G, Manjavidze N. 2013 On some constructive methods for the matrix Riemann–Hilbert boundary value problem. *J. Math. Sci.* **195**, 146–174. (doi:10.1007/s10958-013-1571-7)
17. Boyarskij BV. 1958 On stability of the Hilbert problem for a holomorphic vector. *Soobshch. Akad. Nauk. Gruzin. SSR* **21**, 391–398.
18. Litvinchuk GS, Spitkovsky IM, 1987. *Factorization of measurable matrix functions*. Basel-Boston, MA: Birkhäuser.
19. Kisil AV. 2015 Stability analysis of matrix Wiener–Hopf factorization of Daniele–Khrapkov class and reliable approximate factorization. *Proc. R. Soc. A* **471**, 20150146. (doi:10.1098/rspa.2015.0146)
20. Mishuris G, Rogosin SV. 2014 An asymptotic method of factorization of a class of matrix functions. *Proc. R. Soc. A* **470**, 20140109. (doi:10.1098/rspa.2014.0109)
21. Mishuris G, Rogosin SV. 2016 Factorization of a class of matrix-functions with stable partial indices. *Math. Methods Appl. Sci.* **39**, 3791–3807. (doi:10.1002/mma.3825)
22. Mishuris G, Rogosin SV. 2018 Regular approximate factorization of a class of matrix-function with an unstable set of partial indices. *Proc. R. Soc. A* **474**, 20170279. (doi:10.1098/rspa.2017.0279)
23. Adukova NV, Adukov VM. 2020 On effective criterion of stability of partial indices for matrix polynomials. *Proc. R. Soc. A* **476**, 20200012. (doi:10.1098/rspa.2020.0012)
24. Golub GH, van Loan CF, 1996 *Matrix computations*. London, UK: The John Hopkins Press Ltd.
25. Spitkovsky IM. 1974 Stability of partial indices of the Riemann boudary problem with a strictly nonsingular matrix. *Dokl. Akad. Nauk SSSR* **218**, 46–49. (doi:In Russian)
26. Adukov VM. 1998 Generalized inversion of block Toeplitz matrices. *Linear Algebr. Appl.* **274**, 85–124. (doi:10.1016/S0024-3795(97)00304-2)
27. Horn RA, Johnson CR, 1986. *Matrix analysis*. Cambridge, UK: Cambridge University Press.
28. Geddes KO, Czapor SR, Labahn G. 1992 *Algorithm for computer algebra*. Boston, Dordrecht, London: Kluwer Academic Publishers.
29. Krein MG, Naimark MA. 1981 The method of symmetric and Hermitian forms in the theory of the separation of the roots of algebraic equations. *Linear Multilinear Algebra* **10**, 265–308. (doi:10.1080/03081088108817420)

30. Vieira A, Kailath T. 1977 An another approach to the Schur–Cohn criterion. *IEEE Trans. Circuits Syst.* **24**, 218–220. (doi:10.1109/TCS.1977.1084319)
31. Schur I. 1918 Ueber Potenzreihen, die im Innern des Einheitskreises beschraenkt sind. *J. Reine Angew. Math.* **148**, 122–145.
32. Trefethen LN, Bau D. 1997 *Numerical linear algebra*. Philadelphia, PA: SIAM.
33. Adukov VM, Mishuris G, Rogosin SV. 2020 Exact conditions for preservation of the partial indices of a perturbed triangular 2×2 matrix function. *Proc. R. Soc. A* **476**, 20200099. (doi:10.1098/rspa.2020.0099)
34. Primachuk L, Rogosin S. 2018 Factorization of triangular matrix-functions of an arbitrary order. *Lobachevskii J. Math.*, **39**, 809–817. (doi:10.1134/s1995080218060148)
35. Gautschi W. 1973 On the condition of algebraic equations. *Numer. Math.* **21**, 405–424. (doi:10.1007/BF01436491)
36. Uhlig F. 1992 Are the coefficients of a polynomial well-condition function of its roots? *Numer. Math.* **61**, 383–393. (doi:10.1007/BF01385516)
37. Adukov VM. 2018 Algorithm of polynomial factorization and its implementation in Maple. *Bull. SUSU MMCS* **11**, 110–122. (doi:10.14529/mmp180408)
38. Adukov VM, Adukova NV, Mishuris G. 2022 An explicit Wiener–Hopf factorization algorithm for matrix polynomials and its exact realizations within ExactMPF package. Figshare. (doi:10.6084/m9.figshare.c.6060394)