UDC 512,519.6
DOI: https://doi.org/10.17721/1812-5409.2024/2.12

Artem FISUNENKO, PhD Student
ORCID ID: 0009-0009-1991-3655
e-mail: artemfisunenko@knu.ua
Taras Shevchenko National University of Kyiv, Kyiv, Ukraine

# FORMULAS FOR DETERMINANT AND CHARACTERISTIC POLYNOMIAL OF SEVEN-LIKE MATRICES

***Finding the determinant and characteristic polynomial is the classic problem of linear algebra that arises in some applications of matrices. There exist formulas and algorithms for calculating the determinant and characteristic polynomial of matrices with general structure, however, in some cases, a matrix has a special structure that lets us calculate its determinant or determinant and characteristic polynomial more efficiently. E. g., the determinant of upper and lower triangular matrices can be calculated as a product of their diagonal entries in time, linear in terms of the matrix's size.***

***In this paper, we derived formulas and algorithms for determinant or determinant and characteristic polynomial of what we called seven-like matrices, which include the matrices introduced in M. B. Usher's renewable resources management model and P. H. Leslie's population model, utilizing the properties of determinants and characteristic polynomials and the seven-like matrices' special structure.***

***We implemented our algorithms in Python 3.10.4 and tested their correctness on different values of matrix size and its elements. In all these experiments, the results matched the reference values, some up to a negligibly small relative error due to the floating-point arithmetic.***

***The proposed algorithms have linear computational time complexity, therefore, for seven-like matrices, they are more asymptotically time-efficient than the most asymptotically time-efficient currently known algorithms for finding the determinant and characteristic polynomial, respectively, of the matrices with general structure, and thus our algorithms can be utilized in applications with seven-like matrices (including those of Usher's and Leslie's models). Runtime measurements may be conducted in our further research to verify these asymptotics in practice.***

***K e y w o r d s :  matrix, determinant, characteristic polynomial, renewable resources management, forest management.***

**AMS 2020 classification: 15A15, 15A18, 68Q25, 68W30, 68W40.**

## Introduction

When there exists an algorithm that applies to a certain class of objects, sometimes we can find a simpler, more efficient algorithm for a special case (or special cases) of these objects. One such example is a recent AI-assisted discovery of a faster matrix multiplication algorithm (Fawzi et al., 2022), where, for the matrices having special structure (certain small sizes), an algorithm more efficient than the best at the time of the research was discovered. Another example is a famous fact that the determinant of an upper or lower diagonal matrix of size $n$ can be calculated as a product of its diagonal elements in $O(n)$ time.

In this paper, we too will consider certain special cases of matrix structure, for which the determinant or determinant and characteristic polynomial can be calculated more efficiently than by using the currently most asymptotically time-efficient algorithms for matrices with general structure.

In (Usher, 1966), M. B. Usher describes a renewable resources management model, including a concrete example for Scots pine forest. Usher's model utilizes matrices of the following form:

$$Q = \begin{pmatrix} a_0 & c_1(\lambda-1) & c_2(\lambda-1) & \cdots & c_{n-2}(\lambda-1) & c_{n-1}(\lambda-1) & c_n(\lambda-a_n) \\ b_0 & a_1 & 0 & \cdots & \cdots & \cdots & 0 \\ 0 & b_1 & a_2 & \ddots & & & \vdots \\ \vdots & \ddots & b_2 & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & \ddots & b_{n-2} & a_{n-1} & 0 \\ 0 & \cdots & \cdots & \cdots & 0 & b_{n-1} & a_n \end{pmatrix}, \quad (1)$$

where:

- $a_i, i = \overline{0,n}$ represent the probability that an organism in the $i^{\text{th}}$ class will remain in that class during a certain period;
- $b_j, j = \overline{0,n-1}$ represent the probability that the organism will recruit up to the $(i+1)^{\text{th}}$ class during that period;
- $c_k(\lambda-1), k = \overline{0,n-1}$ and $c_n(\lambda-a_n)$ represent functions of regeneration from the $i^{\text{th}}$ class, where $\lambda$ is $Q$'s eigenvalue variable.

Also, in Usher's model, the following constraints apply:

- $a_i + b_i = 1$, $0 \le a_i < 1$, $0 < b_i \le 1$, $i = \overline{0,n-1}$;
- $0 \le a_n < 1$. If all the resource of the class $n$ is assumed to be exploited, then $a_n = 0$;
- $c_i \ge 0, i = \overline{1,n}$; when $i$ is sufficiently large, $c_i > 1$ and, in particular, $c_n > 1$.

In this model, we're interested in finding the eigenvalues of the matrix $Q$, namely, the largest real-valued eigenvalue greater than $1$, since, according to Usher, it maximizes the yield from the harvested resource. We can find the eigenvalues by calculating the characteristic polynomial of $Q$ and solving for $\lambda$. Note, however, that we also have $\lambda$ included in the first row of the matrix. To address this when calculating the characteristic polynomial programmatically, we may use a symbolic calculations framework, such as SymPy (Meurer et al., 2017), or introduce support for custom symbolic calculations ourselves. Our $\lambda$ will then become a symbolic variable that we can use to compute the characteristic polynomial.



ISSN 1728-3817

Matrix (1) has a special structure that lets us find its characteristic polynomial more efficiently than state-of-the-art techniques for the general case. In this paper, we will derive a formula that can be applied to find such characteristic polynomial, as well as formulas for determinants or determinants and characteristic polynomials of matrices with similar structure.

**1. Formulas derivation**

Consider a matrix

$$R = \begin{pmatrix} \boldsymbol{a_0} & \boldsymbol{d_1} & \boldsymbol{d_2} & \cdots & \boldsymbol{d_{n-2}} & \boldsymbol{d_{n-1}} & \boldsymbol{d_n} \\ \boldsymbol{b_0} & \boldsymbol{a_1} & 0 & \cdots & \cdots & \cdots & 0 \\ 0 & \boldsymbol{b_1} & \boldsymbol{a_2} & \ddots & & & \vdots \\ \vdots & \ddots & \boldsymbol{b_2} & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & \ddots & \boldsymbol{b_{n-2}} & \boldsymbol{a_{n-1}} & 0 \\ 0 & \cdots & \cdots & \cdots & 0 & \boldsymbol{b_{n-1}} & \boldsymbol{a_n} \end{pmatrix} \tag{2}$$

whose entries $a_i, b_j, d_k$ all belong to the same any sensible domain – such that it makes sense to find the determinant and characteristic polynomial of the matrix with these entries, e. g. the domain of complex, real, rational, or integer numbers. We will call matrices defined this way *HM-7 matrices*, or *HMS-matrices*, since their entries $a_i, b_j, d_k$ form a shape that resembles a horizontally mirrored digit 7. Notice that the matrix from Usher's model $Q$ (1) is then an HM-7 matrix; also, we don't assume Usher's model constraints in (2).

To derive our first formula for the determinant of an HM-7 matrix, we're going to utilize the Laplace expansion on the matrix (2), where, for element $R_{1j}$ having minor $M_{1j}$ and cofactor $C_{1j}$,

$$\det R = \sum_{j=1}^{n} (-1)^{1+j} R_{1j} M_{1j} = \sum_{j=1}^{n} R_{1j} C_{1j} \tag{3}$$

First, notice that, when calculating the cofactor of $R_{11} = a_0$, we have a lower triangular matrix

$$\begin{pmatrix} a_1 & 0 & \cdots & \cdots & \cdots & 0 \\ b_1 & a_2 & \ddots & & & \vdots \\ 0 & b_2 & \ddots & \ddots & & \vdots \\ \vdots & \ddots & \ddots & a_{n-2} & \ddots & \vdots \\ \vdots & & \ddots & b_{n-2} & a_{n-1} & 0 \\ 0 & \cdots & \cdots & 0 & b_{n-1} & a_n \end{pmatrix}, \tag{4}$$

whose determinant is the product of its diagonal elements. So, the first term of our formula is a product of $a_0, a_1, \ldots, a_n$, multiplied by $(-1)^{1+1} = 1$:

$$C_{11} = M_{11} = \prod_{i=0}^{n} a_i. \tag{5}$$

Then, notice that when calculating the minors of the element $R_{1j}, j = \overline{2, n+1}$, we obtain them as determinants of the matrices of the following form by crossing out the first row and $j^{\text{th}}$ column in $R$ and merging the split submatrices into one:

$$M_{1j} = \det \begin{pmatrix} b_0 & a_1 & 0 & \cdots & 0 & 0 & \cdots & \cdots & \cdots & 0 \\ 0 & b_1 & a_2 & \ddots & \vdots & \vdots & & & & \vdots \\ \vdots & \ddots & b_2 & \ddots & 0 & \vdots & & & & \vdots \\ \vdots & & \ddots & \ddots & a_{j-2} & \vdots & & & & \vdots \\ 0 & \cdots & \cdots & 0 & b_{j-2} & 0 & \cdots & \cdots & \cdots & 0 \\ 0 & \cdots & \cdots & \cdots & 0 & a_j & 0 & \cdots & \cdots & 0 \\ \vdots & & & & \vdots & b_j & \ddots & \ddots & & \vdots \\ \vdots & & & & \vdots & 0 & \ddots & a_{n-2} & \ddots & \vdots \\ \vdots & & & & \vdots & \vdots & \ddots & b_{n-2} & a_{n-1} & 0 \\ 0 & \cdots & \cdots & \cdots & 0 & 0 & \cdots & 0 & b_{n-1} & a_n \end{pmatrix}. \tag{6}$$

To calculate these determinants, we only need to compute the sum of $b_0$ and $a_1$, each multiplied by its cofactor, since the rest of the elements in the first row are zero and thus we don't need to write down their cofactors.

However, notice that when recursively writing down the cofactor of the element $a_1$, we first cross out the first row and the second column of (6), obtaining its submatrix with the first row being $(0, a_2, 0, \ldots, 0)$. Then, we only write down the cofactor for $a_2$ in a similar way, obtaining the matrix with the first row $(0, 0, a_3, 0, \ldots, 0)$. We continue this procedure until we reach $a_{j-2}$, which yields a cofactor with all zeroes in the first row, making our whole product of cofactors – and thus the initial cofactor of $a_1$ – zero.

Similarly, when computing the cofactors of $b_0$, we get a chain of cofactors for $a_2, \ldots, a_{j-2}$ ending in $a_{j-2}$'s – and thus $a_2$'s – cofactor being zero. For elements $b_{j-2}, a_j, \ldots, a_{n-2}, a_{n-1}, a_n$, all non-crossed-out elements to the right of them are zero, thus, we don't write down their cofactors.

So, we can express the determinant of matrix (6) as the product of its diagonal elements (in expressions for the respective minors, the cofactors on each iteration of recursion are multiplied by $+1$, since each diagonal element in each of the

submatrices is in the first row and first column, yielding $(-1)^{1+1} = 1$), since all other non-crossed-out elements in a horizontal row on each iteration of recursion yield minors equal to zero:

$$M_{1j} = \left(\prod_{i=0}^{j-2} b_i\right)\left(\prod_{k=j}^{n} a_k\right), j = \overline{2, n+1}. \quad (7)$$

We can notice that the formula (5) for the minor $M_{11}$ will actually become a special case of (7) if we make $j = \overline{1, n+1}$ in it, where, for $j = 1$, in the first product, we iterate from $i = 0$ to $j - 2 = 1 - 2 = -1$, which makes the product trivially equal to 1. In our matrix $R$, this corresponds to the fact that we only have $a_k$'s and no $b_i$'s in the main diagonal we compute the determinant along. So, we get an extended formula

$$M_{1j} = \left(\prod_{i=0}^{j-2} b_i\right)\left(\prod_{k=j}^{n} a_k\right), j = \overline{1, n+1}. \quad (8)$$

*Note:* $j = n + 1$ also corresponds to a special case where we only have $b_i$'s and no $a_k$'s on the main diagonal, and the second product that iterates from $k = j = n + 1$ to $n$ is trivially equal to 1.

Finally, we derive our formula by multiplying the values of the elements $R_{1j}, j = \overline{1, n+1}$ of matrix $M_{HM7} = R$ by their cofactors:

$$\det M_{HM7} = \sum_{j=1}^{n+1} (-1)^{j+1} d_{j-1} \left(\prod_{i=0}^{j-2} b_i\right)\left(\prod_{k=j}^{n} a_k\right), \quad (9)$$

where $d_0 \coloneqq a_0$ here and in all formulas below where $d_0$ appears.

To obtain the characteristic polynomial formula, we'll notice that, with $\lambda$ being $R$'s eigenvalue variable, a matrix $\lambda E - R$ is itself an HM-7 matrix by definition, thus, its determinant – which is a characteristic polynomial of $R$ – can be obtained via formula (9), yielding

$$P_{M_{HM7}}(\lambda) = \sum_{j=1}^{n+1} (-1)^{j+1} \left(-d_{j-1}\right)\left(\prod_{i=0}^{j-2} (-b_i)\right)\left(\prod_{k=j}^{n} (\lambda - a_k)\right). \quad (10)$$

The minus signs before $d_{j-1}$ and $b_i$ are due to the fact that these elements are subtracted from zeroes of $\lambda E$.

Note that the characteristic polynomial of $R$ can be also calculated as the determinant of $R - \lambda E$, but we use the determinant of $\lambda E - R$ since it yields a monic characteristic polynomial with a coefficient 1 at the leading term (which can be seen from the multiplication $(\lambda - a_{11}) \cdot (\lambda - a_{22}) \cdot \ldots \cdot (\lambda - a_{mm})$ during the calculation, where $m$ is the matrix's size and $a_{ii}$ are its diagonal elements). $R - \lambda E$, on the other hand, yields a characteristic polynomial with the leading term's coefficient $(-1)^m$ for the matrix of size $m$ (which can be seen from the multiplication $(a_{11} - \lambda) \cdot (a_{22} - \lambda) \cdot \ldots \cdot (a_{mm} - \lambda)$ during the calculation, where $a_{ii}$ are the matrix's diagonal elements). In this case, the coefficient is only equal to 1 for the matrix of even size.

We will also define a *D-7 matrix*, or *DS-matrix* (its entries $a_i, b_j, d_k$ form a shape resembling a digit 7) as follows:

$$M_{D7} = \begin{pmatrix} \boldsymbol{d_n} & \boldsymbol{d_{n-1}} & \boldsymbol{d_{n-2}} & \cdots & \boldsymbol{d_2} & \boldsymbol{d_1} & \boldsymbol{a_0} \\ 0 & \cdots & \cdots & \cdots & 0 & \boldsymbol{a_1} & \boldsymbol{b_0} \\ \vdots & & & \iddots & \boldsymbol{a_2} & \boldsymbol{b_1} & 0 \\ \vdots & & \iddots & \iddots & \boldsymbol{b_2} & \iddots & \vdots \\ \vdots & \iddots & \iddots & \iddots & \iddots & & \vdots \\ 0 & \boldsymbol{a_{n-1}} & \boldsymbol{b_{n-2}} & \iddots & & & \vdots \\ \boldsymbol{a_n} & \boldsymbol{b_{n-1}} & 0 & \cdots & \cdots & \cdots & 0 \end{pmatrix}, \quad (11)$$

where entries $a_i, b_j, d_k$ all belong to the same any sensible domain – such that it makes sense to find the determinant and characteristic polynomial of the matrix with these entries.

Notice that, by swapping $i^{\text{th}}$ and $(n - i + 2)^{\text{th}}$ column of (11), $i = \overline{1, \left\lfloor \frac{n+1}{2} \right\rfloor}$ – in other words, by writing its columns in mirrored/reverse order – we obtain an HM-7 matrix of form (2). Thus, we can calculate $M_{D7}$'s determinant using formula (9) and the fact that the determinant of a matrix changes sign with each swap of two columns of this matrix, i.e. is multiplied by $-1$. So, we can express our determinant as follows:

$$\det M_{D7} = (-1)^{\left\lfloor \frac{n+1}{2} \right\rfloor} \sum_{j=1}^{n+1} (-1)^{j+1} d_{j-1} \left(\prod_{i=0}^{j-2} b_i\right)\left(\prod_{k=j}^{n} a_k\right). \quad (12)$$

When $n + 1$ is even, $\frac{n+1}{2}$ column swaps take place, and, since this number is even, as thus is the number of times the determinant is multiplied by $-1$, the determinant remains unchanged and the same as that of the corresponding HM-7 matrix. When $n + 1$ is odd, $\left\lfloor \frac{n+1}{2} \right\rfloor$ swaps take place, and their number can be either odd or even, changing or not changing the determinant sign, respectively.

We'll call a vertical mirroring of a D-7 matrix a *VM-7 matrix*, or *VMS-matrix* (its entries $a_i, b_j, d_k$ form a shape resembling a vertically mirrored digit 7) and define it as follows:

$$M_{VM7} = \begin{pmatrix} \boldsymbol{a_n} & \boldsymbol{b_{n-1}} & 0 & \cdots & \cdots & \cdots & 0 \\ 0 & \boldsymbol{a_{n-1}} & \boldsymbol{b_{n-2}} & \ddots & & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & \ddots & \boldsymbol{a_2} & \boldsymbol{b_1} & 0 \\ 0 & \cdots & \cdots & \cdots & 0 & \boldsymbol{a_1} & \boldsymbol{b_0} \\ \boldsymbol{d_n} & \boldsymbol{d_{n-1}} & \boldsymbol{d_{n-2}} & \cdots & \boldsymbol{d_2} & \boldsymbol{d_1} & \boldsymbol{a_0} \end{pmatrix}, \quad (13)$$

where entries $a_i, b_j, d_k$ all belong to the same any sensible domain – such that it makes sense to find the determinant and characteristic polynomial of the matrix with these entries.

As we can notice, by swapping $i^{\text{th}}$ and $(n-i+2)^{\text{th}}$ row of (13), $i = \overline{1, \left\lfloor \frac{n+1}{2} \right\rfloor}$ (by writing its rows in mirrored/reverse order) we obtain a D-7 matrix of form (11). By similar reasoning as above, this means that the determinant in the form (12) is multiplied by $(-1)^{\left\lfloor \frac{n+1}{2} \right\rfloor}$, which, when multiplied by $(-1)^{\left\lfloor \frac{n+1}{2} \right\rfloor}$, yields $(-1)^{2\left\lfloor \frac{n+1}{2} \right\rfloor} = 1$. Thus, the determinant of (13) can be expressed as follows:

$$\det M_{VM7} = \sum_{j=1}^{n+1} (-1)^{j+1} d_{j-1} \left(\prod_{i=0}^{j-2} b_i\right)\left(\prod_{k=j}^{n} a_k\right) \tag{14}$$

and, as we can see, it is the same as the determinant of the corresponding HM-7 matrix having the same size and the same respective elements $a_i, b_j, d_k$ for the same respective indices $i, j, k$.

We can find the characteristic polynomial of the VM-7 matrix (13) as the determinant of a matrix $\lambda E - M_{VM7}$, which, by definition, is itself a VM-7 matrix, thus, formula (14) can be applied to find its determinant – which is $M_{VM7}$'s characteristic polynomial – and we get

$$P_{M_{VM7}} = \sum_{j=1}^{n+1} (-1)^{j+1} \left(-d_{j-1}\right)\left(\prod_{i=0}^{j-2} (-b_i)\right)\left(\prod_{k=j}^{n} (\lambda - a_k)\right), \tag{15}$$

which is the same as the characteristic polynomial of the corresponding HM-7 matrix having the same size and the same respective elements $a_i, b_j, d_k$ for the same respective indices $i, j, k$.

We'll call a vertical mirroring of an HM-7 matrix a *VHM-7 matrix*, or *VHMS-matrix* (its entries $a_i, b_j, d_k$ form a shape resembling a vertically and horizontally mirrored digit 7) and define it as follows:

$$M_{VHM7} = \begin{pmatrix} 0 & \cdots & \cdots & \cdots & 0 & \boldsymbol{b_{n-1}} & \boldsymbol{a_n} \\ \vdots & & & \iddots & \boldsymbol{b_{n-2}} & \boldsymbol{a_{n-1}} & 0 \\ \vdots & & \iddots & \iddots & \iddots & \iddots & \vdots \\ \vdots & \iddots & \boldsymbol{b_2} & \iddots & \iddots & & \vdots \\ 0 & \boldsymbol{b_1} & \boldsymbol{a_2} & \iddots & & & \vdots \\ \boldsymbol{b_0} & \boldsymbol{a_1} & 0 & \cdots & \cdots & \cdots & 0 \\ \boldsymbol{a_0} & \boldsymbol{d_1} & \boldsymbol{d_2} & \cdots & \boldsymbol{d_{n-2}} & \boldsymbol{d_{n-1}} & \boldsymbol{d_n} \end{pmatrix}, \tag{16}$$

where entries $a_i, b_j, d_k$ all belong to the same any sensible domain – such that it makes sense to find the determinant and characteristic polynomial of the matrix with these entries.

Swapping $i^{\text{th}}$ and $(n-i+2)^{\text{th}}$ row of (16), $i = \overline{1, \left\lfloor \frac{n+1}{2} \right\rfloor}$ (by writing its rows in mirrored/reverse order), we obtain an HM-7 matrix of form (2). By similar reasoning as above, this means that the determinant in form (9) is multiplied by $(-1)^{\left\lfloor \frac{n+1}{2} \right\rfloor}$:

$$\det M_{VHM7} = (-1)^{\left\lfloor \frac{n+1}{2} \right\rfloor} \sum_{j=1}^{n+1} (-1)^{j+1} d_{j-1} \left(\prod_{i=0}^{j-2} b_i\right)\left(\prod_{k=j}^{n} a_k\right) \tag{17}$$

and, as we can see, it is the same as the determinant of the corresponding D-7 matrix having the same size and the same respective elements $a_i, b_j, d_k$ for the same respective indices $i, j, k$.

Notice that this formula can also be derived from the fact that, swapping $i^{\text{th}}$ and $(n-i+2)^{\text{th}}$ column, $i = \overline{1, \left\lfloor \frac{n+1}{2} \right\rfloor}$ (by writing its columns in mirrored/reverse order) we obtain a VM-7 matrix of form (13) and, by similar reasoning as above, this means that the determinant in form (14) is multiplied by $(-1)^{\left\lfloor \frac{n+1}{2} \right\rfloor}$, from which we get (17).

We will call HM-7, D-7, VM-7, and VHM-7 matrices *seven-like matrices*. As we have seen, assuming that these matrices' sizes and respective entries match, besides having structural symmetry, these matrices also have symmetry in the formulas for their determinants and, for HM-7 and V-7, characteristic polynomials.

The relationships between the determinants and characteristic polynomials of these matrices are visualized in Fig. 1, where all the depicted matrices are of the same size $n+1$, and entries $a_i, b_j, d_k$ in each of these matrices are the same as entries with the same index $a_i, b_j, d_k$, respectively, in any other of these matrices:

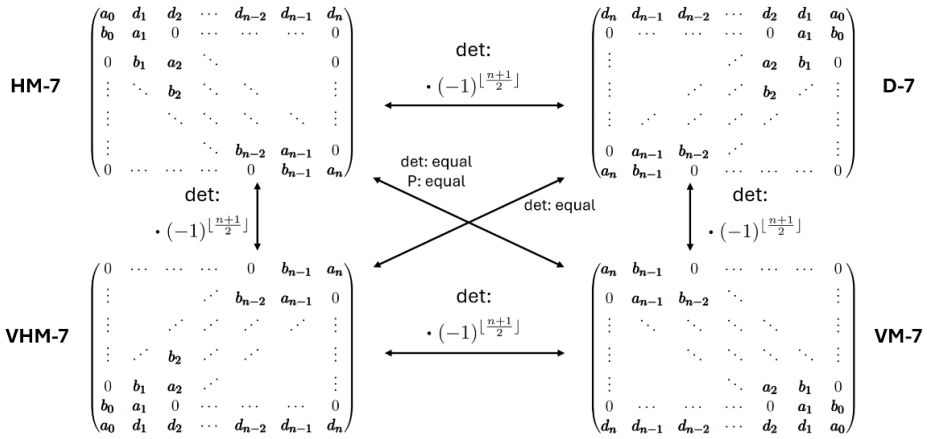


**Fig. 1. Relationships between the determinants and characteristic polynomials of seven-like matrices. Here, "det" stands for determinants, and "P" stands for characteristic polynomials**

**2. Algorithms**

By examining the formulas (9), (10), (12), (14), (15), and (17), we may notice that they have overlapping sum terms: the product of $b_i$'s starting from 1, and growing by a new term $b_i$ on each further iteration, and the product of $a_k$'s or $(\lambda - a_k)$'s, shrinking by the first term of each previous iteration, and ending in 1 on the last iteration (assuming our calculations are done with $j$ *growing* from 1 to $n+1$). To avoid performing the same multiplications multiple times in programmatic implementations of our formulas, we can use the dynamically growing cumulative product of $b_i$'s (growing by a term on each iteration of the sum), and a precomputed (before calculation of the product of $b_i$'s) list of cumulative products of $a_k$'s on each iteration $k$ (with $j$ *growing* from 1 to $n+1$); or vice versa – precomputed products of $b_i$'s on each iteration $i$ and the dynamically growing product of $a_k$'s (with $j$ *shrinking* from $n+1$ to 1).

Note that we can't accumulate both products simultaneously in the same direction of the index $j$ (since one product grows by a term, and the other shrinks by a term), and thus we compute one product dynamically, and the other before the beginning of such dynamic computation. It is also not beneficial to cumulatively grow one product (starting from 1 and multiplying by a new term on each iteration) and simultaneously cumulatively shrink the other (starting from the pre-computed overall product and dividing by a new term on each iteration), since it may cause division by zero and/or rounding errors. Also, each division yields partial products that we can instead obtain from calculating the overall product via term-by-term multiplication, starting from 1.

In this paper, we'll use the first approach (dynamically growing product of $b_i$'s and pre-computed of $a_k$'s), however, the second one can be used similarly with the same asymptotic time complexity, which we will prove the following way.

Consider the formulas (9), (10), (12), (14), (15), (17). A sum in each of them ranges from $j = 1$ to $n+1$. The product of $b_i$'s in this sum ranges from $i = 0$ to $j-2$, thus, with $j = \overline{1, n+1}$, we have a sequence of partial products

$$1, b_0, b_0 b_1, \dots, \prod_{i=0}^{n-2} b_i, \prod_{i=0}^{n-1} b_i. \tag{18}$$

We can pre-compute these products by consecutively multiplying the starting value 1 by the values of the matrix's $b_i$'s elements and storing the partial products, totaling $n$ multiplications, or $n-1$ if we omit multiplying $b_0$ by 1 and just start from $b_0$. Similarly, the dynamic growth of the cumulative product, starting from 1, yields the same products (18) with $n$ multiplications.

Now, consider the product of $a_k$'s or $(\lambda - a_k)$'s. It ranges from $k = j$ to $n$, thus, with $j = \overline{1, n+1}$, we have a sequence of partial products

$$\prod_{i=1}^{n} a_i, \prod_{i=2}^{n} a_i, \dots, a_n, 1 \tag{19}$$

or

$$\prod_{i=1}^{n} (\lambda - a_i), \prod_{i=2}^{n} (\lambda - a_i), \dots, \lambda - a_n, 1. \tag{20}$$

We can pre-compute these products in the direction of $j = \overline{n+1, 1}$ by consecutively multiplying the starting value 1 by the values of the matrix's $a_i$ or $(\lambda - a_i)$ elements and storing the partial products, totaling $n$ multiplications, or $n-1$ if we omit multiplying $a_n$ or $(\lambda - a_n)$ by 1 and just start from $a_n$ or $(\lambda - a_n)$. Similarly, the dynamic growth of the cumulative product, starting from 1, yields the same products (19) or (20) with $n$ multiplications.

As we can see, either approach takes at most $O(n)$ multiplications, which concludes our proof.

■

We can further opt to either use extra space (for example, an array) or to allocate the precomputed $a_k$ cumulative products in the matrix itself (e. g. on the main diagonal, except for the first element). Both strategies have a tradeoff to them: using an auxiliary array and filling it with these cumulative products requires $O(n)$ time and space. Filling in the input matrix, on the other hand, requires $O(n)$ time and no additional space, but it overwrites the main diagonal entries that were there before replacing them with the cumulative products. This may be a problem if one wants to use the original matrix's diagonal elements afterwards, so, in that case, the main diagonal entries should be stored elsewhere and written back to the input matrix after computing the determinant or characteristic polynomial – again, requiring $O(n)$ space and time.

Our algorithms for calculating the determinant and characteristic polynomial of HM-7 and VM-7 matrices, and the determinant of D-7 and VHM-7 matrices can be viewed at (Fisunenko, 2024) where we published both their pseudocode and Python implementation under MIT license. We'll call algorithms using the auxiliary array *structure-preserving* (*SP*), as the matrix preserves its original entries at the end of the algorithm; and we'll call algorithms modifying the input matrix *structure-modifying* (*SM*). From now on, we will use the following notation:

- *Algorithm 1* – HM-7 determinant, SP;
- *Algorithm 2* – HM-7 determinant, SM;
- *Algorithm 3* – HM-7 characteristic polynomial, SP;
- *Algorithm 4* – HM-7 characteristic polynomial, SM;
- *Algorithm 5* – D-7 determinant, SP;
- *Algorithm 6* – D-7 determinant, SM;
- *Algorithm 7* – VM-7 determinant, SP;
- *Algorithm 8* – VM-7 determinant, SM;
- *Algorithm 9* – VM-7 characteristic polynomial, SP;
- *Algorithm 10* – VM-7 characteristic polynomial, SM;
- *Algorithm 11* – VHM-7 determinant, SP;
- *Algorithm 12* – VHM-7 determinant, SM.

Below, we will only show Algorithms 1 and 2 for the sake of brevity. Algorithms 3-12 follow a similar pattern and can be found at (Fisunenko, 2024).

*Algorithm 1*. Determinant of an HM-7 matrix (structure-preserving)
**function** DeterminantHM7StructurePreserving($M$):
    $n \coloneqq M$.size $- 1$
    $A \coloneqq$ **new** array[1..$n$]
    $A[i] \coloneqq M[i+1][i+1], i = \overline{1,n}$
    **for** $i \coloneqq n-1$ **downto** $1$ **do**
        $A[i] \coloneqq A[i] \cdot A[i+1]$
    $s \coloneqq 0$
    $b \coloneqq 1$
    **for** $i \coloneqq 1$ **to** $n$ **do**
        $s \coloneqq s + (-1)^{i+1} \cdot b \cdot M[1][i] \cdot A[i]$
        $b \coloneqq b \cdot M[i+1][i]$
    **return** $s + (-1)^n \cdot b \cdot M[1][n]$
**end function**

*Algorithm 2*. Determinant of an HM-7 matrix (structure-modifying)
**function** DeterminantHM7StructureModifying($M$):
    $n \coloneqq M$.size $- 1$
    **for** $i \coloneqq n$ **downto** $2$ **do**
        $M[i][i] \coloneqq M[i][i] \cdot M[i+1][i+1]$
    $s \coloneqq 0$
    $b \coloneqq 1$
    **for** $i \coloneqq 1$ **to** $n$ **do**
        $s \coloneqq s + (-1)^{i+1} \cdot b \cdot M[1][i] \cdot M[i+1][i+1]$
        $b \coloneqq b \cdot M[i+1][i]$
    **return** $s + (-1)^n \cdot b \cdot M[1][n+1]$
**end function**

To compute the determinant of a D-7 matrix, we could reverse the matrix's columns (inside the original matrix, since making a copy of the matrix takes $O(n^2)$ time), compute the determinant of the resulting matrix as that of an HM-7 matrix, and multiply it by $(-1)^{\left\lfloor \frac{n+1}{2} \right\rfloor}$, obtaining the determinant, reverse the columns back, and return the computed determinant – according to the relationships between the determinants and characteristic polynomials of seven-like matrices (see Figure 1). However, assuming that the matrix is represented as an array of its rows, such reversal would take $O(n^2)$ time to swap elements $n+1$ times (for each row) in each of the $\left\lfloor \frac{n+1}{2} \right\rfloor$ pairs of columns. And if the matrix were represented as an array of its columns, swapping the pointers to them would require $O(n)$ time. Similar reasoning applies to the row reversal cases (HM-7 – VHM-7, D-7 – VM-7): if the matrix is represented as an array of its rows, swapping the pointers to them takes $O(n)$ time; if the matrix is represented as an array of columns, swapping elements $n+1$ times (for each column) in each of the $\left\lfloor \frac{n+1}{2} \right\rfloor$ pairs of rows takes $O(n^2)$ time.

Thus, in Algorithms 5–6, 7–8, 9–10, and 11–12 we instead make calculations according to the respective formulas (12), (14), (15), (17) with indices adjusted accordingly.

**3. Complexity analysis**

By examining Algorithms 1–12, we can notice that they have time complexity $O(n)$: loops contain at most $O(n)$ operations, $O(1)$ each; initialization of the auxiliary array in the Algorithms 1, 3, 5, 7, 9, 11 takes $O(n)$ time; the rest of the operations in Algorithms 1-12 all take $O(1)$ time.

The structure-preserving implementations of our formulas (Algorithms 1, 3, 5, 7, 9, and 11) have the additional space complexity $O(n)$ for the auxiliary array and $O(1)$ for other local variables, totaling $O(n)$. The structure-modifying implementations of our formulas (Algorithms 2, 4, 6, 8, 10, and 12) have $O(1)$ additional space complexity for the local variables. Algorithms 1-12 all inherently require $O(n^2)$ space to store the input matrix.

**4. Testing the algorithms**

We implemented Algorithms 1–12 in Python 3.10.4 and tested them in the following three experiments.

*Experiment 1*. Algorithms 1, 2, 5-8, 11-12 (for determinant) on the input matrix were compared with the NumPy's (Harris et al., 2020) determinant-finding method *numpy.linalg.det* on the same input matrix. We used NumPy version 1.22.3. The input matrices were:

- for Algorithms 1 and 2, an HM-7 matrix;
- for Algorithms 5 and 6, a D-7 matrix;
- for Algorithms 7 and 8, a VM-7 matrix;
- for Algorithms 11 and 12, a VHM-7 matrix.

All the specified matrices were constructed as two-dimensional NumPy arrays of their rows, with size $n$ ranging from $1$ to $20$. Each matrix was initialized with all elements equal to zero, except elements $a_i, b_j, d_k$, all initialized with random integers in the range $\overline{1,10}$ via Python's built-in *random.randint* function in one series of experiments; and all initialized with random real floating-point numbers in the range $[0,10)$ via Python's built-in *random.random* function in the other.

*Experiment 2*. Algorithms 3, 4, 9, and 10 (for characteristic polynomial) on the input matrix were compared with finding the characteristic polynomial via SymPy (using the *charpoly* method of *sympy.Matrix* instance). We used SymPy version 1.12. The input matrices were:

- for Algorithms 3 and 4, an HM-7 matrix;
- for Algorithms 9 and 10, a VM-7 matrix.

All the specified matrices were constructed as *sympy.Matrix* instances as two-dimensional arrays of their rows, with sizes ranging from $1$ to $10$. Each matrix was initialized with all elements equal to zero, except elements $a_i, b_j, d_k$, all initialized with random integers in the range $\overline{1,10}$ via Python's built-in *random.randint* function in one series of experiments; and all initialized with random real floating-point numbers in the range $[0,10)$ via Python's built-in *random.random* function in the other.

*Experiment 3*. We also tested Algorithms 3 and 4 on the HM-7 matrix $Q$ (1) from (Usher, 1966), representing it as *sympy.Matrix* instance with a symbolic variable $\lambda$ in the form of a *sympy.Symbol* instance. We compared the output of these algorithms (characteristic polynomials $P^{(SP)}$ and $P^{(SM)}$, respectively) with the characteristic polynomial $P^*$ of $Q$ found using the *charpoly*

method of *sympy.Matrix* instance. We also calculated the maximum eigenvalue $\lambda_{\max}$ using the characteristic polynomials found by our algorithms and $P^*(\lambda)$.

For all of the measurements in Experiment 1, the values of computed determinants for structure-preserving and structure-modifying versions match. Compared to the reference values, they have an absolute error that starts extremely low – with order of magnitude around $-16$ to $-15$ – for small matrix sizes and tends to grow as the size grows (the peak value of absolute error being $19$ for $20 \times 20$ D-7 matrix with integer entries). That said, the relative error starts and remains extremely low even for the larger sizes, and its overall growth trend with the increase in size on our testing data is rather slow (the order of magnitude for small matrix sizes is around $-16$ to $-15$, and around $-16$ to $-13$ for larger sizes). For some measurements with various matrix sizes, the calculated determinant values in their floating-point representations match precisely, thus, for them, both absolute and relative errors are equal to zero in these cases.

We should also highlight that NumPy calculates determinants as floating-point numbers even for integer entries, which may inherently lead to precision errors in resulting values that get more and more prominent as the size of the matrix grows. This effect is the most noticeable for integer entries, where, assuming the correct implementation of our algorithms, the calculated value is inherently precise and doesn't have any rounding errors. For real entries, both NumPy and our implementations are prone to floating-point rounding errors.

In Experiment 2, for HM-7 and VM-7 matrices with integer entries, the calculated characteristic polynomials matched precisely for the respective structure-preserving version, structure-modifying version, and the SymPy reference value. For VM-7 matrices with real entries, the calculated characteristic polynomials also matched precisely for all three techniques. For HM-7 matrices with real entries, the calculated characteristic polynomials for all three techniques matched precisely for sizes $1$–$4$ and $6$, however, for sizes $5$,$7$,$8$,$9$,$10$ the polynomials calculated using structure-preserving and structure-modifying versions matched, but with a certain difference in some of their coefficients and the respective coefficients in polynomials calculated using SymPy: the highest order of magnitude being $-6$ for size $10$, and the lowest being $-11$ for size $5$. Again, the relative error starts and remains extremely low and grows rather slowly as the matrix's size increases (its order of magnitude for sizes $5$–$10$ is around $-15$ to $-13$, peaking at $-13$ for sizes $8$–$10$). For sizes $1$–$4$ and $6$, all the calculated coefficients in their floating-point representations matched precisely, so, both absolute and relative errors are equal to zero in these cases.

In Experiment 3, the calculated characteristic polynomials matched precisely for structure-preserving and structure-modifying versions, which matched the polynomial calculated using SymPy with the only difference being the extra term $-2.77555756156289 \cdot 10^{-17}$, added by SymPy to that polynomial. The maximum eigenvalue calculated with all the approaches was the same and equal to $1.204266169$, which closely corresponds to the value $1.204$ stated by Usher in (Usher, 1966).

**Discussion and conclusions**

As of our current knowledge, the most asymptotically time-efficient determinant and characteristic polynomial algorithms for matrices of the general structure both have not less than quadratic time complexity – more precisely, finding both determinant and characteristic polynomial is as asymptotically time-complex as multiplying two square matrices, for which the current best is $O(n^{\omega+\varepsilon})$ for all $\varepsilon > 0$ and $\omega < 2.371339$ (Alman et al., 2024; Chiantini et al., 2018; Dumas, Pernet, & Wan, 2005; Keller-Gehrig, 1985).

In comparison, our determinant and characteristic polynomial finding algorithms for the case of seven-like matrices have $O(n)$ time complexity, which is better than the time complexity of the respective algorithms for the general case. Thus, our algorithms can be used in applications of seven-like matrices for more efficient computation of their determinants or determinants and characteristic polynomials. Such applications include Usher's renewable resources management model, where Algorithms 3 and 4 (or the respective formulas) can be applied to find the characteristic polynomial of an HM-7 matrix, and, from it, the matrix's maximum eigenvalue that is greater than $1$; and Leslie's population model (Leslie, 1945) where HM-7 and VM-7 matrices occur with all $a_i$ coefficients are strictly zero, and the goal is to find the stable age distribution via computing the characteristic polynomial of an HM-7 matrix, so, Algorithms 3 and 4 (or the respective formulas) can be used for this.

In this paper, we defined matrices with a special structure, which we called seven-like matrices, and showed that their determinants or determinants and characteristic polynomials, for which we derived formulas and algorithms, can be found in a more asymptotically time-efficient way than that of the currently most efficient state-of-the-art techniques, and showed the example of possible applications of some of our algorithms, namely Usher's and Leslie's models.

We implemented our algorithms in Python 3.10.4 and conducted experiments that verified their correctness on certain data. The code for the implemented algorithms was published on GitHub under MIT license (Fisunenko, 2024). The practical verification of linear asymptotic time complexity may be conducted in our further research.

**Acknowledgments, sources of funding.** I would like to express my deepest gratitude to my family for their support and specifically to my father, Andriy Fisunenko, for suggesting improvements to this paper; to my academic supervisor, Vasyl Tereschenko, for his guidance throughout this research; and to assistant lecturer Tetiana Kolianova for giving me an inspiration for this paper through her academic assignment to find an optimal way of calculating the goal values of Usher's forest management model, and for providing me with literature from which I discovered Usher's paper (Usher, 1966).

**References**

Alman, J., Duan, R., Williams, V. V., Xu, Y., Xu, Z., & Zhou, R. (2024). More Asymmetry Yields Faster Matrix Multiplication. *arXiv preprint* arXiv, 2404, 16349.

Chiantini, L., Hauenstein, J. D., Ikenmeyer, C., Landsberg, J. M., & Ottaviani, G. (2018). Polynomials and the exponent of matrix multiplication. *Bulletin of the London Mathematical Society*, 50(3), 369–389.

Dumas, J. G., Pernet, C., & Wan, Z. (2005, July). Efficient computation of the characteristic polynomial. *In Proceedings of the 2005 international symposium on Symbolic and algebraic computation*, 140–147.

Fawzi, A., Balog, M., Huang, A. et al. (2022). Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature,* 610, 47–53. https://doi.org/10.1038/s41586-022-05172-4

Fisunenko, A. (2024, November 8). The experiments conducted to verify the correctness of algorithms for seven-like matrices on certain data. *GitHub*. https://github.com/artandfi/seven-like-matrices

Harris, C. R., Millman, K. J., van der Walt, S. J. et al. (2020). Array programming with NumPy. *Nature,* 585, 357–362. https://doi.org/10.1038/s41586-020-2649-2.
Keller-Gehrig, W. (1985). Fast algorithms for the characteristics polynomial. *Theoretical computer science*, 36, 309–317.
Leslie, P. H. (1945). On the use of matrices in certain population mathematics. *Biometrika*, *33*(3), 183–212.
Meurer, A., Smith, C. P., Paprocki, M. et al. (2017). SymPy: symbolic computing in Python. *PeerJ Computer Science*, 3, e103. https://doi.org/10.7717/peerj-cs.103
Usher, M. B. (1966). A matrix approach to the management of renewable resources, with special reference to selection forests. *Journal of Applied Ecology*, 355–367.

**Отримано редакцією журналу / Received: 09.05.24**
**Прорецензовано / Revised: 05.11.24**
**Схвалено до друку / Accepted: 26.11.24**

**Артем ФІСУНЕНКО, асп.**
**ORCID ID: 0009-0009-1991-3655**
**e-mail: artemfisunenko@knu.ua**
**Київський національний університет імені Тараса Шевченка, Київ, Україна**

## ФОРМУЛИ ДЛЯ ВИЗНАЧНИКА ТА ХАРАКТЕРИСТИЧНОГО ПОЛІНОМА СІМКОПОДІБНИХ МАТРИЦЬ

***Знаходження визначника та характеристичного полінома є класичною задачею лінійної алгебри, яка постає в деяких застосуваннях, що використовують матриці. Існують формули й алгоритми для обчислення визначника та характеристичного полінома матриць загального вигляду, проте в деяких випадках матриця має особливу структуру, що дає змогу обчислювати визначник та/чи характеристичний поліном ефективніше. Прикладом слугують верхньо- та нижньотрикутні матриці, визначник яких обраховують як добуток їхніх діагональних елементів за лінійний відносно розміру матриці час.***

***У пропонованій роботі виведено формули nf алгоритми знаходження визначника або характеристичного полінома для матриць, які ми назвали сімкоподібними, що містять матриці, запроваджені в моделі М. Б. Ашера керування відновлювальними ресурсами та популяційної моделі Леслі шляхом використання властивостей визначників і характеристичних поліномів та особливої структури сімкоподібних матриць.***

***Наші алгоритми реалізовано мовою Python 3.10.4 та протестовано їхню коректність на різних значеннях розміру матриці та її елементів. У всіх цих експериментах результати збіглися з референсним значенням, деякі з точністю до малої відносної похибки, спричиненої обчисленнями з рухомою комою, якою можна знехтувати.***

***Запропоновані алгоритми мають лінійну часову обчислювальну складність. Зважаючи на це, для випадку сімкоподібних матриць вони є більш асимптотично ефективними за часом, ніж найбільш асимптотично ефективні за часом із нині відомих алгоритмів знаходження визначника та характеристичного полінома, відповідно, для матриць загальної структури. Отже, наші алгоритми можна використовувати в застосуваннях із сімкоподібними матрицями (включно із застосуваннями моделей Ашера та Леслі). У подальшому дослідженні можуть бути проведені заміри часу виконання для практичного підтвердження зазначених асимптотик.***

**Ключові слова:** ***матриця, визначник, характеристичний поліном, керування відновлювальними ресурсами, лісокористування.***

Автор заявляє про відсутність конфлікту інтересів. Спонсори не брали участі в розробленні дослідження; у зборі, аналізі чи інтерпретації даних; у написанні рукопису; в рішенні про публікацію результатів.
The author declares no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses or interpretation of data; in the writing of the manuscript; in the decision to publish the results.