J. Math. Computer Sci., 31 (2023), 287–304

Online: ISSN 2008-949X

Journal of Mathematics and Computer Science

Journal Homepage: www.isr-publications.com/jmcs

On the computational and numerical approaches for the powers of the doubly Lefkovitch matrix by linear difference equations

Amal Aloui[a,*], Mustapha Rachidi[b]

[a]Département de Mathématiques et Informatique, Faculté des Sciences, Université Ibn Tofail, Kénitra, Morocco.

[b]Institute of Mathematics INMA, Federal University of Mato Grosso do Sul UFMS, Campo Grande, MS 79070-900, Brazil.

Abstract

We explore here a study to derive purely explicit formulas for entries of the $n-th$ powers of doubly Lefkovitch matrices. Our tool is based on some linear properties of difference equations, including recursive, analytical and derivative aspects of solutions to these equations. Three algorithms for computing the entries of the powers of doubly Lefkovitch matrix are built. Moreover, illustrative applications and examples are furnished.

Keywords: Doubly Lefkovitch matrix, entries of the powers of doubly Lefkovitch matrix, linear difference equations, doubly Leslie matrix.

2020 MSC: 15A57, 34K60, 39A05.

1. Introduction

Recently, some studies are devoted to the so-called doubly Lefkovitch matrices defined by,

$$\mathbf{L}=\mathbf{L}([a_j],[s_j],[d_j],[b_j])=\begin{bmatrix} a_1 & a_2 & a_3 & a_4 & \cdots & a_r \\ s_1 & d_1 & 0 & 0 & \cdots & b_{r-2} \\ 0 & s_2 & d_2 & 0 & \cdots & b_{r-3} \\ \vdots & \cdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \cdots & \ddots & \ddots & d_{r-2} & b_1 \\ 0 & \cdots & \cdots & 0 & s_{r-1} & d_{r-1} \end{bmatrix}, \tag{1.1}$$

where b_j $(1\leqslant j\leqslant r-2)$, a_j $(1\leqslant j\leqslant r)$ and d_j $(1\leqslant j\leqslant r-1)$ are in $\mathbb{R}^+$ and $0<s_j\leqslant 1$ $(1\leqslant j\leqslant r-1)$. Some properties of matrices (1.1) have been studied in [2], where the authors consider the problem of inverse eigenvalue. Moreover, the doubly Lefkovitch matrices can be considered as an extension of the Usher matrices $\mathbf{U}$, where $\mathbf{U}=\mathbf{L}([a_i],[s_j],[d_j],[0])$, (see, for example, [8]). On the other side, the doubly Lefkovitch matrices can be viewed as the doubly Leslie matrices $\mathcal{L}$ where $\mathcal{L}=\mathbf{L}([a_i],[s_j],[\widehat{d}_j],[b_j])$, with

*Corresponding author

Email addresses: amal.alaoui@uit.ac.ma (Amal Aloui), mu.rachidi@gmail.com (Mustapha Rachidi)

doi: 10.22436/jmcs.031.03.05

Received: 2022-11-04 *Revised:* 2023-03-28 *Accepted:* 2023-04-05

$[\widehat{d_j}] = (0, \cdots, 0, d_{r-1})$ (see, for example, [1]), which can be used as a fundamental tool to compute the $n-th$ powers of doubly Lefkovitch matrices. Furthermore, they can also be viewed as a generalization of Leslie matrices L, defined by $L = \mathbf{L}([a_i], [s_j], [0], [0])$ (see, for example, [4],[3]). In the current literature, it was shown, in various research papers, that the Lefkovitch matrices are related to the study of population dynamics. Therefore, the computation of the powers of the Leslie and doubly Leslie matrices or the Usher matrices, play a central role, in describing the evolution of the population vector (see for instance [3, 4, 8]).

The main goal of this study is to introduce a new computational method for investigating the entries of the powers of doubly Lefkovitch matrices (1.1). More precisely, our method permits us to give various explicit formulas for the entries of the powers $\mathbf{L}^n$ of the doubly Lefkovitch matrices (1.1), using the properties of the linear difference equation with constant coefficients described by,

$$v_{n+1} = \gamma_1 v_n + \gamma_2 v_{n-1} + \cdots + \gamma_r v_{n-r+1}, \quad \text{for} \quad n \geqslant r, \tag{1.2}$$

where $\gamma_1, \gamma_2, ..., \gamma_r$ are constant coefficients and $v_1, v_2, ..., v_r$ are the initial conditions (see [6, 7]). Our method is based on three approaches for expressing the general terms v_n given by (1.2), namely, the recursive, the analytic, and the derivative approaches. In addition, we outline these three approaches for investigating three algorithms to compute the entries of the powers of matrices (1.1). To highlight the importance of our implementations, we compare the running time of these algorithms with the existing one in the numpy module of Python 3.10.

The outline of this study is as follows. Section 2 is devoted to some explicit formulas for the entries of the powers of the doubly Lefkovitch matrices (1.1). More precisely using the properties of linear difference equations (1.2) we develop three explicit expressions of the entries of the powers of matrices (1.1). In Section 3, some compact formulations for the entries of the vector of the population dynamics are offered as an application to dynamical populations. In Section 4 we develop three algorithms to compute the entries of the powers of matrices (1.1), then we compare the running time of these algorithms with the existing one in the numpy module of Python 3.10. Finally, concluding remarks and perspectives are presented.

2. Entries of the powers of the doubly Lefkovitch matrix

2.1. The linear recursive approach for computing the entries of the powers of the matrix (1.1)

The main goal here is to establish some compact formulas for the entries of the $n-th$ powers of the doubly Lefkovitch matrix (1.1), in terms of a family of sequences defined by the linear recursive relation (1.2) and specific initial conditions. To this aim, the first step consists in considering the similarity between the doubly Lefkovitch matrix and the doubly Leslie matrix. Then, we utilize a tool based on the $n-th$ power of the $r \times r$ doubly Leslie matrix established in [1]. That is, we will prove in this article that every doubly Lefkovitch matrix $\mathbf{L} = \mathbf{L}([a_j], [s_j], [d_j], [b_j])$ is similar to a doubly Leslie matrix, namely, $\mathcal{L} = \mathbf{P}^{-1}\mathbf{L}\mathbf{P} = \mathbf{L}([\phi_j], [s_j], [0], [\psi_j])$, where $\mathcal{L}$ is the doubly Leslie matrix of the form,

$$\mathcal{L} = \begin{bmatrix} \phi_1 & \phi_2 & \phi_3 & \cdots & \phi_r \\ s_1 & 0 & 0 & \cdots & \psi_{r-2} \\ 0 & s_2 & 0 & \cdots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \psi_1 \\ 0 & \cdots & 0 & s_{r-1} & 0 \end{bmatrix}, \tag{2.1}$$

such that the coefficients ϕ_j $(1 \leqslant j \leqslant r)$ are as follows,

$$\begin{cases} \phi_j = -\dfrac{\xi_1^j}{\Lambda_1^{j-1}} + \displaystyle\sum_{k=1}^{j} \dfrac{\xi_k^{j-k}}{\Lambda_k^{j-1}} a_k, & \text{for} \quad 1 \leqslant j \leqslant r-1, \\ \phi_j = -\dfrac{\xi_1^j}{\Lambda_1^{j-1}} + \displaystyle\sum_{k=1}^{j} \dfrac{\xi_k^{j-k}}{\Lambda_k^{j-1}} a_k + \sum_{l=2}^{r-1} b_{r-l} m_{1,l}, & \text{for} \quad j = r, \end{cases} \tag{2.2}$$

with

$$\xi_i^p = \begin{cases} (-1)^p \sum\limits_{i\leqslant i_1<i_2<\cdots<i_p\leqslant r-1} d_{i_1}d_{i_2}\cdots d_{i_p}, & \text{if} \quad 0<p\leqslant r-i, \\ 1, & \text{if} \quad p=0, \\ 0, & \text{if} \quad r-i<p, \end{cases} \tag{2.3}$$

and

$$\Lambda_i^j = \begin{cases} \prod\limits_{k=i}^{j} s_k, & \text{if} \quad i\leqslant j\leqslant r-1, \\ 1, & \text{if} \quad j=i-1, \end{cases} \tag{2.4}$$

and $m_{i,j}$ $(1\leqslant i,j\leqslant r)$ are the entries of the matrix $\mathbf{P}^{-1}$, where the coefficients ψ_j $(1\leqslant j\leqslant r-2)$ are as follows,

$$\begin{cases} \psi_1 &= b_1, \\ \psi_j &= \sum\limits_{k=1}^{j} b_k m_{r-j,r-k}, \quad \text{for} \quad 2\leqslant j\leqslant r-2. \end{cases} \tag{2.5}$$

In addition, the entries $p_{i,j}$ $(1\leqslant i,j\leqslant r)$ of the matrix $\mathbf{P}$ are defined by,

$$\begin{cases} p_{i,j} &= \dfrac{\xi_i^{j-i}}{\Lambda_i^{j-1}}, \quad \text{if} \quad j>i, \\ p_{i,j} &= 0, \quad \text{if} \quad i>j, \\ p_{i,j} &= 1, \quad \text{if} \quad i=j, \end{cases} \tag{2.6}$$

where ξ_i^p $(1\leqslant i\leqslant r,\ 0\leqslant p\leqslant r)$ are given by formulas (2.3) and Λ_i^j $(1\leqslant i\leqslant r,\ 0\leqslant j\leqslant r)$ are given by (2.4).
Since $\mathbf{L}=\mathbf{P}\mathcal{L}\mathbf{P}^{-1}$ then, for every integer $n\geqslant 0$, we have,

$$\mathbf{L}^n = \mathbf{P}\mathcal{L}^n\mathbf{P}^{-1}. \tag{2.7}$$

Therefore, the entries of the powers of the doubly Lefkovitch matrix $\mathbf{L}=\mathbf{L}([a_j],[s_j],[d_j],[b_j])$, will be readily available from the entries of the powers of the doubly Leslie matrix $\mathcal{L}$. It is known, that the computation of the powers $\mathcal{L}^n$ of the doubly Leslie matrix $\mathcal{L}$, in terms of a family of sequences (1.2), has been established in [1]. More precisely, the entries of $\mathcal{L}^n$ are given in terms of the family of sequences $\{\nu_n^{(s)}\}_{n\geqslant 1}$, indexed by s $(1\leqslant s\leqslant r)$, defined as follows,

$$\begin{cases} \nu_{n+1}^{(s)} &= \gamma_1\nu_n^{(s)}+\gamma_2\nu_{n-1}^{(s)}+\cdots+\gamma_r\nu_{n-r+1}^{(s)}, \quad \text{for} \quad n\geqslant r, \\ \nu_n^{(s)} &= \delta_{s,n}, \quad \text{for} \quad 1\leqslant n\leqslant r. \end{cases} \tag{2.8}$$

such that

$$\begin{cases} \gamma_1 &= c_1+f_1, \\ \gamma_j &= -\sum\limits_{i+k=j,i\neq 0,k\neq 0} c_i f_k + c_j + f_j \quad \text{for} \quad 2\leqslant j\leqslant r-1, \\ \gamma_r &= -\sum\limits_{i+k=r,i\neq 0,k\neq 0} c_i f_k + c_r, \end{cases} \tag{2.9}$$

with $c_1=\Phi_1; c_i=\Phi_i\prod\limits_{j=1}^{i-1} s_j$ for $2\leqslant i\leqslant r$ where $\phi_j(1\leqslant j\leqslant r)$ are given by formulas (2.2) and $f_1=0; f_j=\Psi_{j-1}\prod\limits_{k=r-j+1}^{r-1} s_k$ for $2\leqslant j\leqslant r-1$, where ψ_j $(1\leqslant j\leqslant r-2)$ are given by (2.5) .
To compute the entries of the powers of the doubly Lefkovitch matrix, we present some preliminary results that are necessary for the rest of the paper.

Lemma 2.1. *([9]) Let $\mathcal{L}$ be a doubly Leslie matrix of the form (2.1) and $\mathbf{B}$ their associated matrix companion, we have,*

$$\mathcal{L} = \mathbf{Q}\mathbf{B}\mathbf{Q}^{-1}, \tag{2.10}$$

where,

$$\mathbf{B} = \begin{bmatrix} \gamma_1 & \gamma_2 & \gamma_3 & \cdots & \gamma_r \\ 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 & 0 \end{bmatrix}, \tag{2.11}$$

such that the scalars γ_j $(1 \leqslant j \leqslant r)$ are given by formulas (2.9), and the entries of the matrix $\mathbf{Q}$ are denoted by $\Gamma_{i,j}(1 \leqslant i, j \leqslant r)$, where,

$$\begin{cases} \Gamma_{i,j} &= 0, & \text{if} \quad i > j, \\ \Gamma_{i,j} &= \dfrac{-f_{j-i}}{\prod\limits_{k=i}^{r-1} s_k}, & \text{if} \quad j > i, \\ \Gamma_{i,i} &= \dfrac{1}{\prod\limits_{k=i}^{r-1} s_k}, & \text{for} \quad 1 \leqslant i \leqslant r-1, \\ \Gamma_{r,r} &= 1, \end{cases} \tag{2.12}$$

and the entries of the matrix $\mathbf{Q}^{-1}$ are denoted by $\mu_{i,j}$ $(1 \leqslant i, j \leqslant r)$, namely, $\mathbf{Q}^{-1} = (\mu_{i,j})_{1 \leqslant i,j \leqslant r}$.

The following Lemma can be obtained by a standard iterative process of formula (2.10) in Lemma 2.1 and entries of the $n-$th power of the companion matrix $\mathbf{B}$ (2.11) expressed in terms of sequences (2.8).

Lemma 2.2 ([1])**.** *Let $\mathcal{L}$ be a doubly Leslie matrix, then the entries of the matrix power $\mathcal{L}^n$, are expressed in terms of sequences (2.8) as follows,*

$$\mathcal{L}_{ij}^{(n)} = \sum_{p=i}^{r} \Gamma_{i,p} \sum_{k=1}^{j} \mu_{k,j} \nu_{n+r+1-p}^{(r+1-k)}, \tag{2.13}$$

where the sequences $\{\nu_n^{(s)}\}_{n \geqslant 1}$ $(1 \leqslant s \leqslant r)$ are defined by (2.8), the coefficients $\Gamma_{i,j}$ $(1 \leqslant i, j \leqslant r)$ are as in (2.12) and the $\mu_{i,j}$ $(1 \leqslant i, j \leqslant r)$ are the entries of $\mathbf{Q}^{-1}$.

Therefore, combining expressions (2.7) and (2.13) of Lemma 2.2, we get an explicit formula for the entries of the powers of the doubly Lefkovitch matrix.

Theorem 2.3. *(entries of the powers of the doubly Lefkovitch matrix.) Let $\mathbf{L} = (L_{ij})_{1 \leqslant i,j \leqslant r}$ be the doubly Lefkovitch matrix (1.1) and $\mathcal{L}$ the associated doubly Leslie matrix (2.1). Then, the entries $L_{ij}^{(n)}$ of the power $\mathbf{L}^n$, are expressed under the following form,*

$$L_{ij}^{(n)} = \sum_{h=1}^{r} m_{h,j} \sum_{k=i}^{r} p_{i,k} \sum_{p=k}^{r} \Gamma_{k,p} \sum_{s=1}^{h} \mu_{s,h} \nu_{n+r+1-p}^{(r+1-s)}, \tag{2.14}$$

for every $n \geqslant 0$, where the sequences $\{\nu_n^{(s)}\}_{n \geqslant 1}$ $(1 \leqslant s \leqslant r)$ are defined by (2.8), the coefficients $p_{i,j}(1 \leqslant i, j \leqslant r)$ are as in (2.6), $m_{i,j}$ are the entries of the matrix $\mathbf{P}^{-1}$, the coefficients $\Gamma_{i,j}(1 \leqslant i, j \leqslant r)$ are defined by (2.12) and the $\mu_{i,j}$ $(1 \leqslant i, j \leqslant r)$ are the entries of $\mathbf{Q}^{-1}$.

Proof. For every $n \geqslant 0$, we have $\mathcal{L}^n = \mathbf{P}^{-1}\mathbf{L}^n\mathbf{P}$. Since the entries of $\mathcal{L}^n$ are identified by Expression (2.13) and $\mathbf{L}^n = \mathbf{P}\mathcal{L}^n\mathbf{P}^{-1}$, where the entries of matrix $\mathbf{P}$ are given by formula (2.6) and entries of $\mathbf{P}^{-1}$ are denoted by $m_{i,j}$, thus we can get the entries of the matrix $\mathbf{L}^n$ as follows, $L_{ij}^{(n)} = \sum_{h=1}^{r} m_{h,j} \sum_{k=i}^{r} p_{i,k} \sum_{p=k}^{r} \Gamma_{k,p} \sum_{s=1}^{h} \mu_{s,h} \nu_{n+r+1-p}^{(r+1-s)}$. □

As an application, we exhibit here a numerical example to illustrate how to apply the formula of Theorem 2.3, we detail the following example.

Example 2.4. *Consider the following doubly Lefkovitch matrix* (1.1) *of order* 3×3 *defined by,*

$$\mathbf{L} = \begin{bmatrix} 2 & 24 & 20 \\ 0.6 & 1 & 9 \\ 0 & 0.8 & 3 \end{bmatrix}. \tag{2.15}$$

Then, the associated Leslie matrix $\mathcal{L}$ *and the matrices* $\mathbf{P}$, $\mathbf{P}^{-1}$ *are as follows,*

$$\mathcal{L} = \begin{bmatrix} 6 & 5.6666667 & 2.5 \\ 0.6 & 0 & 9 \\ 0 & 0.8 & 0 \end{bmatrix}, \quad \mathbf{P} = \begin{bmatrix} 1 & -6.66666667 & 6.25 \\ 0 & 1 & -3.75 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{P}^{-1} = \begin{bmatrix} 1 & 6.66666667 & 18.75 \\ 0 & 1 & 3.75 \\ 0 & 0 & 1 \end{bmatrix}.$$

The family of sequences (2.8) *are defined as follows,*

$$\begin{cases} v_{n+1}^{(s)} &= 6v_n^{(s)} + 10.60000002v_{n-1}^{(s)} - 42.7v_{n-2}^{(s)}, \quad \textit{for} \quad n \geqslant 2, \\ v_i^{(s)} &= \delta_{s,i}, \qquad\qquad\qquad\qquad\qquad\quad \textit{for} \quad 0 \leqslant i \leqslant 2. \end{cases}$$

Suppose that the computation of $\mathbf{L}^5$ *is required. First, Expression* (2.13) *shows that we have,*

$$\mathcal{L}^5 = \begin{bmatrix} 11361.84 & 12214.3067 & 20484.0998 \\ 1229.4960012 & 1235.520007 & 2629.440009 \\ 144.5760001 & 193.5680009 & 211.4400007 \end{bmatrix}.$$

Second, since the entries $L_{ij}^{(5)}$ *of* $\mathbf{L}^5$ *are given by Expression* (2.14)*, we derive that,*

$$\mathbf{L}^5 = \begin{bmatrix} 4068.799984896805 & 32312.63993530412 & 100018.39953952564 \\ 687.336001 & 5091.88001 & 16635.24 \\ 144.576 & 1157.408 & 3648.12001 \end{bmatrix}.$$

2.2. Powers of doubly Lefkovitch matrix by the analytic approach

We are concerned in this subsection with the analytic formula of the entries of the doubly Lefkovitch (1.1). Formulas (2.14) shows that these entries are expressed in terms of the family of sequences defined by (2.8). Therefore, to reach our goal, we recall that, the analytic formula of each sequence $\{v_n^{(s)}\}_{n \geqslant 1}$, indexed by s $(1 \leqslant s \leqslant r)$, is given by,

$$v_n^{(s)} = \sum_{i=1}^{l} \left(\sum_{j=0}^{m_i - 1} \beta_{i,j}^{(s)} n^j \right) \lambda_i^n, \quad 1 \leqslant s \leqslant r, \tag{2.16}$$

where, for each fixed $s (1 \leqslant s \leqslant r)$, the $\beta_{i,j}^{(s)}$ are computed by solving the following generalized Vandermonde system of r linear equations,

$$\sum_{i=1}^{l} \left(\sum_{j=0}^{m_i - 1} \beta_{i,j}^{(s)} n^j \right) \lambda_i^n = \delta_{s,n}, \quad 1 \leqslant n \leqslant r,$$

with $\lambda_1, \lambda_2, \cdots, \lambda_l$ are distinct roots of the characteristic polynomial of sequences (2.8) with multiplicities $m_1, m_2, \cdots, m_l$ $(m_1 + \cdots + m_l = r)$, respectively (see, for example, [5]).

Combining Expressions (2.14) and (2.16), we show that the entries of the powers of doubly Lefkovitch matrix can be formulated in terms of the characteristic roots λ_i $(1 \leqslant i \leqslant l)$ and their multiplicities m_i, and the scalars $\beta_{i,j}^{(s)}$. Indeed, we get the following result.

Theorem 2.5. *Let* $\mathbf{L} = (L_{ij})_{1\leqslant i,j\leqslant r}$ *be the doubly Lefkovitch matrix* (1.1) *and* $\mathcal{L}$ *the associated doubly Leslie matrix* (2.1). *Then, the analytic expressions of the entries* $L_{ij}^{(n)}$ *of the power* $\mathbf{L}^n$, *are given by,*

$$L_{ij}^{(n)} = \sum_{h=1}^{r} m_{h,j} \sum_{k=i}^{r} p_{i,k} \sum_{p=k}^{r} \Gamma_{k,p} \sum_{s=1}^{h} \mu_{s,h} \sum_{u=1}^{l} \left(\sum_{t=0}^{m_u-1} \beta_{u,t}^{(r+1-s)} (n+r+1-p)^t \right) \lambda_u^{n+r+1-p}, \tag{2.17}$$

for every $n \geqslant 0$, *where the coefficients* $p_{i,j} (1 \leqslant i,j \leqslant r)$ *are as in* (2.6), $m_{i,j}$ *are the entries of the matrix* $\mathbf{P}^{-1}$, *the coefficients* $\Gamma_{i,j} (1 \leqslant i,j \leqslant r)$ *are defined by* (2.12), *the* $\mu_{i,j}$ $(1 \leqslant i,j \leqslant r)$ *are the entries of* $\mathbf{Q}^{-1}$ *and the* $\beta_{i,j}^{(s)}$ *are computed by solving the linear generalized Vandermonde system of equations* $v_n^{(s)} = \delta_{s,n}$ *for* $1 \leqslant n \leqslant r$.

As an application, we exhibit here the previous doubly Lefkovitch matrix used in Example 2.4 to illustrate how to apply the formula of Theorem 2.5.

Example 2.6. *Let us consider again the doubly Lefkovitch matrix used in Example 2.4. By direct computation, we show that the characteristic roots are* $\lambda_1 = -2.85677971$, $\lambda_2 = 2.21281587$ *and* $\lambda_3 = 6.64396385$. *The analytic formula* (2.16) *show that each sequence* $\left\{v_n^{(s)}\right\}_{n\geqslant 1}$ $(1 \leqslant s \leqslant 3)$ *takes the form,*

$$v_n^{(s)} = (-2.85677971)^n \beta_{1,0}^{(s)} + (2.21281587)^n \beta_{2,0}^{(s)} + (6.64396385)^n \beta_{3,0}^{(s)},$$

where the real numbers $\beta_{k,0}^{(s)}$ $(s = 1,2,3, 1 \leqslant k \leqslant 3)$ *are derived from the initial conditions* $v_n^{(s)} = \delta_{n,s}$, *by solving a generalized Vandermonde system of equations. And a straightforward computation gives,*

$$\begin{cases} \beta_{1,0}^{(1)} = -0.10685, & \beta_{2,0}^{(1)} = 0.38183, & \beta_{3,0}^{(1)} = -0.02260. \\ \beta_{1,0}^{(2)} = 0.064368, & \beta_{2,0}^{(2)} = 0.07619, & \beta_{3,0}^{(2)} = 0.00230. \\ \beta_{1,0}^{(3)} = -0.00727, & \beta_{2,0}^{(3)} = -0.02012, & \beta_{3,0}^{(3)} = 0.00357. \end{cases}$$

Suppose that the calculation of $\mathbf{L}^5$ *is required. Then, using* (2.17), *we can check that,* $L_{11}^{(5)} = 4068.8000089474$, $L_{12}^{(5)} = 32312.640018464$, $L_{13}^{(5)} = 100018.39994186$, $L_{21}^{(5)} = 687.3360017496$, $L_{22}^{(5)} = 5091.880008017$, $L_{23}^{(5)} = 16635.24000260$, $L_{31}^{(5)} = 144.57600032582$, $L_{32}^{(5)} = 1157.3782836101$, $L_{33}^{(5)} = 3648.1199992108$.

2.3. Powers of doubly Lefkovitch matrix by the derivative approach

We recall that the derivative expression of sequences $\{v_n^{(s)}\}_{n\geqslant 1}$, indexed by s $(1 \leqslant s \leqslant r)$ is given by the following formula,

$$v_{n+1}^{(s)} = \sum_{i=1}^{l} \sum_{p=1}^{r} A_p^{(s)} \times \frac{f_{i,n-p+1}^{(m_i-1)}(\lambda_i)}{(m_i-1)!}, \quad n \geqslant r, \tag{2.18}$$

where the function f_i for $1 \leqslant i \leqslant l$ is defined by $f_{i,n}(x) = \dfrac{x^{n-1}}{\prod\limits_{k=1,k\neq i}^{l} (x-\lambda_k)^{m_k}}$, $A_j^{(s)} = \gamma_r v_j^{(s)} + \cdots + \gamma_j v_r^{(s)}$ for $1 \leqslant j \leqslant r$ and $\lambda_1, \lambda_2, \cdots, \lambda_l$ are distinct roots of the characteristic polynomial of sequences (2.8) with multiplicities $m_1, m_2, \cdots, m_l$ $(m_1 + \cdots + m_l = r)$, respectively (see, for example, [5]).

Formula (2.18) illustrates the important rule of the properties of sequences (2.8), notably their derivative formula, to explore more properties of the doubly Lefkovitch matrix. Indeed, we get the result by combining the expressions (2.14) and (2.18).

Theorem 2.7. *Let* $\mathbf{L} = (L_{ij})_{1\leqslant i,j\leqslant r}$ *be the doubly Lefkovitch matrix* (1.1) *and* $\mathcal{L}$ *the associated doubly Leslie matrix* (2.1). *Then, the entries* $L_{ij}^{(n)}$ *of the power* $\mathbf{L}^n$, *are given as follows,*

$$L_{ij}^{(n)} = \sum_{h=1}^{r} m_{h,j} \sum_{k=i}^{r} p_{i,k} \sum_{p=k}^{r} \Gamma_{k,p} \sum_{s=1}^{h} \mu_{s,h} \sum_{u=1}^{l} \sum_{t=1}^{r} A_t^{r+1-s} \times \frac{f_{u,n+r-p-t+1}^{(m_u-1)}(\lambda_u)}{(m_u-1)!}, \tag{2.19}$$

for every $n \geqslant 0$, *where the coefficients* $p_{i,j}(1 \leqslant i,j \leqslant r)$ *are as in* (2.6), $m_{i,j}$ *are the entries of the matrix* $\mathbf{P}^{-1}$, *the coefficients* $\Gamma_{i,j}(1 \leqslant i,j \leqslant r)$ *are defined by* (2.12) *and the* $\mu_{i,j}$ $(1 \leqslant i,j \leqslant r)$ *are the entries of* $\mathbf{Q}^{-1}$.

As an application, we exhibit here a numerical example to illustrate how to apply the formula (2.19) of Theorem 2.7, we detail the following example.

Example 2.8. *Let's consider the doubly Lefkovitch matrix used in Example 2.4 The characteristic polynomial is* $P(X) = X^3 - 6X^2 - 10.6X + 42$ *and the roots of this polynomial are* $\lambda_1 = -2.85677971, \lambda_2 = 2.21281587, \lambda_3 = 6.64396385$. *The derivative formulas of the sequences* $\left\{v_n^{(s)}\right\}_{n\geqslant 1}$ $(1 \leqslant s \leqslant 3)$ *associated with this doubly Lefkovitch matrix* (2.15) *are,*

$$\begin{aligned} v_n^{(s)} &= \frac{1}{48.165}\left[A_1^{(s)}(-2.85678)^{n-1} + A_2^{(s)}(-2.85678)^{n-2} + A_3^{(s)}(-2.85678)^{n-3}\right] \\ &-\frac{1}{22.464}\left[A_1^{(s)}(2.21281)^{n-1} + A_2^{(s)}(2.21281)^{n-2} + A_3^{(s)}(2.21281)^{n-3}\right] \\ &+\frac{1}{42.10}\left[A_1^{(s)}(6.64396)^{n-1} + A_2^{(s)}(6.64396)^{n-2} + A_3^{(s)}(6.64396)^{n-3}\right], \end{aligned}$$

where the numbers $A_p^{(s)}$ *for* $1 \leqslant p,s \leqslant 3$ *are defined by* $A_1^{(s)} = -42v_1^{(s)} + 10.60000002v_2^{(s)} + 6v_3^{(s)}$, $A_2^{(s)} = -42v_2^{(s)} + 10.60000002v_3^{(s)}, A_3(s) = -42v_3^{(s)}$. *Then, we have,* $A_1^{(1)} = -42, A_2^{(1)} = 0, A_3^{(1)} = 0, A_1^{(2)} = 10.60000002, A_2^{(2)} = -42, A_3^{(2)} = 0$ *and* $A_1^{(3)} = 6, A_2^{(3)} = 10.60000002, A_3^{(3)} = -42$. *Suppose that the computation of the power* $\mathbf{L}^5$ *is required. Then, using formula* (2.19), *we get,*

$$L_{ij}^{(5)} = \sum_{h=1}^{3} m_{h,j} \sum_{k=i}^{3} p_{i,k} \sum_{p=k}^{3} \Gamma_{k,p} \sum_{s=1}^{h} \mu_{s,h} \sum_{u=1}^{l} \sum_{t=1}^{3} A_t^{(4-s)} f_{u,9-p-t}(\lambda_u)$$

Therefore, a straightforward computation allows us to obtain,
$L_{11}^{(5)} = 4068.8000093760$, $L_{12}^{(5)} = 32312.640101136$, $L_{13}^{(5)} = 100018.40032292$, $L_{21}^{(5)} = 687.336002020$, $L_{22}^{(5)} = 5091.8800426682, L_{23}^{(5)} = 16635.240067640$, $L_{31}^{(5)} = 144.5760003189, L_{32}^{(5)} = 1157.4080036367, L_{33}^{(5)} = 3648.120011552$.

3. Application to a discrete matrix dynamical system

This section is devoted to the application of the doubly Lefkovitch matrix in a dynamical population. Let's consider the linear matrix difference equation

$$N(n+1) = \mathbf{L}N(n), \quad \text{for } n \geqslant 0, \tag{3.1}$$

where $\mathbf{L}$ is a doubly Lefkovitch matrix (1.1) and $N(n) = {}^t(x_1(n); x_2(n); \cdots; x_r(n))$ is the dynamical vector constructed from the initial condition $N(0) = {}^t(x_1(0), x_2(0), \cdots, x_r(0))$ with $x_j(n) \in \mathbb{R}(1 \leqslant j \leqslant r)$. A standard iterative process shows that the matrix equation (3.1) takes the form,

$$N(n) = \mathbf{L}^n N(0), \quad \text{for } n \geqslant 0, \tag{3.2}$$

Therefore, a simple matrix multiplication using Formula (3.2) and formulas (2.14), (2.17) and (2.19) allows us to deduce the compact expressions for the entries $x_m(n)(1 \leqslant m \leqslant r)$ of the vector $N(n) = {}^t(x_1(n), x_2(n), \cdots, x_r(n))$ describing the population dynamics.

Theorem 3.1. *(entries of the vector of the population dynamics.) Let* $\mathbf{L} = \left(L_{ij}\right)_{1\leqslant i,j\leqslant r}$ *be the doubly Lefkovitch matrix defined in* (1.1). *Suppose that* $\lambda_1, \lambda_2, \cdots, \lambda_l$ *are the distinct roots of the polynomial* $P(X) = X^r - \gamma_1 X^{r-1} - \cdots - \gamma_r$ *of multiplicities* $m_1, m_2, \cdots, m_l$, *respectively. Then, for every* $n \geqslant 0$, *the entries* $x_b(n)(1 \leqslant b \leqslant r)$, *of the vector* $N(n) = {}^t(x_1(n); x_2(n); \cdots; x_r(n))$, *are given by,*

$$x_b(n) = \sum_{d=1}^{r}\sum_{h=1}^{r} m_{h,d} \sum_{k=b}^{r} p_{b,k} \sum_{p=k}^{r} \Gamma_{k,p} \sum_{s=1}^{h} \mu_{s,h} \nu_{n+r+1-p}^{(r+1-s)} x_d(0), \tag{3.3}$$

$$x_b(n) = \sum_{d=1}^{r}\sum_{h=1}^{r} m_{h,d} \sum_{k=b}^{r} p_{b,k} \sum_{p=k}^{r} \Gamma_{k,p} \sum_{s=1}^{h} \mu_{s,h} \sum_{u=1}^{l} \left(\sum_{t=0}^{m_u-1} \beta_{u,t}^{(r+1-s)} (n+r+1-p)^t \right) \lambda_u^{n+r+1-p} x_d(0), \tag{3.4}$$

$$x_b(n) = \sum_{d=1}^{r}\sum_{h=1}^{r} m_{h,d} \sum_{k=b}^{r} p_{b,k} \sum_{p=k}^{r} \Gamma_{k,p} \sum_{s=1}^{h} \mu_{s,h} \sum_{u=1}^{l} \sum_{t=1}^{r} A_t^{r+1-s} \times \frac{f_{u,n+r-p-t+1}^{(m_u-1)}(\lambda_u)}{(m_u-1)!} x_d(0), \tag{3.5}$$

where $N(0) = {}^t(x_1(0); x_2(0); \cdots; x_r(0))$ *is the initial vector,* $\left\{\nu_n^{(s)}\right\}_{n\geqslant 1} (1 \leqslant s \leqslant r)$ *are the sequences* (2.8), *the coefficients* $p_{i,j}(1 \leqslant i,j \leqslant r)$ *are as in* (2.6), $m_{i,j}$ *are the entries of the matrix* $\mathbf{P}^{-1}$ *the coefficients* $\Gamma_{i,j}(1 \leqslant i,j \leqslant r)$ *are defined by* (2.12), *the* $\mu_{i,j}$ $(1 \leqslant i,j \leqslant r)$ *are the entries of* $\mathbf{Q}^{-1}$ *and the* $\beta_{i,j}^{(s)}$ *are computed by solving the linear generalized Vandermonde system of equations* $\nu_n^{(s)} = \delta_{s,n}$ *for* $1 \leqslant n \leqslant r$.

We illustrate through the following example how to apply our results of Theorem 3.1.

Example 3.2. *Suppose that the doubly Lefkovitch matrix and the initial population vector are the following,* $\mathbf{L} = \begin{bmatrix} 3 & 50 & 10 \\ 0.75 & 2 & 150 \\ 0 & 0.25 & 5 \end{bmatrix}$, $\mathbf{N}(0) = \begin{bmatrix} 11 \\ 19 \\ 28 \end{bmatrix}$

Then, the associated doubly Leslie matrix $\mathcal{L}$ and the matrices $\mathbb{P}$ and $\mathbb{P}^{-1}$ are as follows

$$\mathcal{L} = \begin{bmatrix} 10 & 8.67 & 570 \\ 0.75 & 0 & 150 \\ 0 & 0.25 & 0 \end{bmatrix}, \mathbb{P} = \begin{bmatrix} 1 & \frac{-28}{3} & \frac{160}{3} \\ 0 & 1 & -20 \\ 0 & 0 & 1 \end{bmatrix}, \mathbb{P}^{-1} = \begin{bmatrix} 1 & \frac{28}{3} & \frac{400}{3} \\ 0 & 1 & 20 \\ 0 & 0 & 1 \end{bmatrix}$$

The family of sequences $\{\nu_n^{(s)}\}$ are as follows

$$\begin{cases} \nu_{n+1}^{(s)} &= 10\nu_n^{(s)} + 44\nu_{n-1}^{(s)} - 268.125\nu_{n-2}^{(s)}, & \text{for } n \geqslant 3, \\ \nu_n^{(s)} &= \delta_{s,n}, & \text{for } 1 \leqslant n \leqslant 3. \end{cases}$$

Suppose we need to compute N(6), then using formulas (3.3), (3.4) and (3.5) we get

$$N(6) = {}^t(4337761815.176; 744984471.48; 27877959.54175).$$

4. Algorithms

In this subsection, we will develop efficient computational algorithms for entries of the $n-th$ powers of doubly Lefkovitch matrix (1.1).

4.1. Algorithm based on the linear recursive approach for computing the entries of the powers of the matrix (1.1)

The recursive algorithm for finding the entries of the powers of an $r \times r$ doubly Lefkovitch matrix, $\mathbf{L}^n$, can be summarized in Algorithm 1. For reasons of practical convenience, the following notations will be used, in the development of our algorithms:

- p is the matrix $\mathbf{P}$ given by (2.6).
- m is the matrix $\mathbf{P}^{-1}$.
- dl is the doubly Leslie matrix $\mathcal{L}$ expressed by (2.1).
- Γ is the matrix $\mathbf{Q}$.
- μ is the matrix $\mathbf{Q}^{-1}$.
- b is the matrix $\mathbf{B}$.
- v3 is the sequences $\{v_n^{(s)}\}_{n\geqslant 1}$ given by (2.8).

4.2. Algorithm based on the analytic approach for computing the entries of the powers of the matrix (1.1)

The algorithm for finding the entries of the powers of an $r \times r$ doubly Lefkovitch matrix, $\mathbf{L}^n$, using an analytic approach, can be summarized in Algorithm 2. The steps from 1 to 4 of Algorithm 2 are the same as those in Algorithm 1. The next step consists on computing $\beta_{i,j}^{(s)}$ for each fixed s ($1 \leqslant s \leqslant r$). In the last step of Algorithm 2, we use the expression (2.17) to compute the entries of the powers of matrices (1.1).

4.3. Algorithm based on the derivative approach for computing the entries of the powers of the matrix (1.1)

The algorithm for finding the entries of the powers of an $r \times r$ doubly Lefkovitch matrix, $\mathbf{L}^n$, using the derivative approach, can be summarized in Algorithm 3. The steps from 1 to 4 of Algorithm 3 are the same as those in Algorithm 1. The next step consists on computing $f_{i,n}(\lambda)$ and $A_j^{(s)}$ for $1 \leqslant j \leqslant r$. In the last step of Algorithm 3, we use the expression (2.19) to compute the entries of the powers of matrices (1.1).

4.4. Numerical Example

In this subsection, we give an illustrative example. All tests were performed in Python 3.10.0. Consider the 4×4 matrix L given by

$$>> \mathbf{L} = \begin{bmatrix} 4 & 14 & 10 & 11 \\ 0.6 & 1 & 0 & 5 \\ 0 & 0.8 & 3 & 10 \\ 0 & 0 & 0.2 & 4 \end{bmatrix} \tag{4.1}$$

Suppose we need to compute $\mathbf{L}^{90}$. Therefore, using Algorithms 1,2 and 3, we obtain

$$>> \mathbf{L}^{90} = \begin{bmatrix} 6.08651680 \times 10^{70} & 2.18891740 \times 10^{71} & 3.46110157 \times 10^{71} & 2.42148011 \times 10^{72} \\ 7.31608456 \times 10^{69} & 2.63111157 \times 10^{70} & 4.16029605 \times 10^{70} & 2.91065545 \times 10^{71} \\ 2.62350508 \times 10^{69} & 9.43501201 \times 10^{69} & 1.49185780 \times 10^{70} & 1.04374400 \times 10^{71} \\ 2.43164464 \times 10^{68} & 8.74501693 \times 10^{68} & 1.38275624 \times 10^{69} & 9.67413600 \times 10^{69} \end{bmatrix}$$

We remark that we obtain the same result as the function $matrix\_power()$ of the numpy module existing in Python 3.10.0. To highlight the efficiency of our algorithms, we compare the running time of these algorithms. The running time of these algorithms is summarized in Table 1.

In Table 1, we fix the length of the matrix to $r = 4$ and vary the power n. We observed that the mean elapsed time of the recursive and analytic approaches is much less than the derivative approach. We can also observe that the mean elapsed time of Algorithm 1 and Algorithm 2 are almost the same as the mean elapsed time of the function $matrix\_power()$.

Table 1: Mean elapsed time over 10 runs for the doubly Lefkovitch matrix (4.1)

	Mean elapsed time			
n	recursive approach	analytic approach	derivative approach	matrix_power() function
5	0.0421875	0.06875	7.0046875	0
10	0.0734375	0.0734375	6.978125	0.0046875
15	0.1171875	0.0671875	7.3015625	0
20	0.1515625	0.0703125	7.3015625	0.003125
25	0.1859375	0.0703125	6.978125	0
30	0.23125	0.0703125	7.1	0
35	0.2734375	0.06875	7.1671875	0.003125
40	0.3421875	0.1265625	7.2453125	0.00625
45	0.3625	0.1625	6.821875	0.0078125
50	0.2484375	0.159375	4.134375	0.0015625
55	0.24375	0.14375	3.778125	0
60	0.2703125	0.1390625	3.7765625	0.0015625

5. Concluding Remarks and perspectives

In this paper, we propose three approaches for computing the entries of the powers of the doubly Lefkovitch matrix, namely recursive, analytic and derivative approaches. These approaches are based on the properties of linear difference equations of the Fibonacci type. Furthermore, we developed three algorithms based on these approaches to compute the $n-th$ powers of the doubly Lefkovitch matrix. To the best of our knowledge, our results are not current in the literature on this important subject. Finally, it seems to us that our results may have interesting perspectives in various fields of mathematics and applied sciences, notably on population dynamics.

Appendix A.

Algorithm 1: Computing $\mathbf{L}^n$ denoted as power_l by recursive approach

Step 1 : Compute entries of the matrix **P** using expression (2.6)
Step 2 : Compute entries of the matrix $\mathbf{P}^{-1}$ using the function inv() of numpy.linalg module
Step 3 : Compute entries of the matrix $\mathcal{L}$ defined by (2.1) using the function dot() of numpy module
Step 4 : Compute the entries of companion matrix **C** and entries of matrices **Q** and $\mathbf{Q}^{-1}$, where $\mathcal{L} = \mathbf{QCQ}^{-1}$
Step 5 : Compute $v_n^{(s)}$ defined by (2.8)
Step 6 : Compute entries of the matrix **L** using expression (2.14) wich we will detail bellow
Input: L,n,p,m,dl,gamma,mu,and b
Output: $\mathbf{L}^n$

```
power_l = zeros((n, n))
for i = 1 : len_matrix + 1 do
    for j = 1 : len_matrix + 1 do
        som = 0
        for h = 1 : r + 1 do
            for k = i : r + 1 do
                for p = k : r + 1 do
                    for s = 1 : r + 1 do
                        som = som + m[h − 1][j − 1] ∗ p[i − 1][k − 1] ∗ gamma[k − 1][p − 1] ∗ mu[s − 1][h − 1] ∗ v3(n +
                          len_matrix − u + 1, len_matrix + 1 − s, dl, b)[n + len_matrix − u]
                    end
                end
            end
        end
        power_l[i − 1][j − 1] = som
    end
end
```

Algorithm 2: Computing $\mathbf{L}^n$ denoted as power_l by analytic approach

Steps 1-4 : The same as those in Algorithm 1
Step 5 : Compute $\beta_{i,j}^{(s)}$
Step 6 : Compute entries of the matrix **L** using expression (2.17) wich we will detail bellow
Input: n,m,p,gamma,mu and **L**
Output: $\mathbf{L}^n$

```
power_l = zeros((n, n))
for i = 1 : len_matrix do
    for j = 1 : len_matrix do
        som = 0
        for h = 1 : r do
            for k = i : r do
                for p = k : r do
                    for s = 1 : h do
                        f = beta_vanderm(l, r + 1 − s)
                        for u = 1 : l do
                            for t = 0 : m_u − 1 do
                                som = som + m[h − 1][j − 1] ∗ p[i − 1][k − 1] ∗ gamma[k − 1][c − 1] ∗ mu[s − 1][h − 1] ∗
                                  f[u − 1] ∗ ((n + len_matrix − c + 1) ∗ ∗t) ∗ ((unique_elements[u − 1]) ∗ ∗(n +
                                  len_matrix − c + 1))
                            end
                        end
                    end
                end
            end
        end
        power_l[i − 1][j − 1] = som
    end
end
```

Algorithm 3: Computing $\mathbf{L}^n$ denoted as power_l by derivative approach

Steps 1-4 : The same as those in Algorithm 1
Step 5 : Compute $f_{i,n}(\lambda)$ **Step 6** : Compute$A_j^{(s)}$
Step 7 : Compute entries of the matrix **L** using expression (2.19) wich we will detail bellow
Input: n,m,p,gamma,mu and **L**
Output: $\mathbf{L}^n$

```
power_l = zeros((n, n))
for i = 1 : len_matrix + 1 do
    for j = 1 : len_matrix + 1 do
        additionner = 0
        for h = 1 : len_matrix + 1 do
            for k = i : len_matrix + 1 do
                for c = k : len_matrix + 1 do
                    for s = 1 : h + 1 do
                        for u = 1 : l do
                            for t = 0 : m_u − 1 do
                                additionner = additionner + m[h − 1][j − 1] ∗ p[i − 1][k − 1] ∗ gamma[k − 1][c − 1] ∗
                                  mu[s − 1][h − 1] ∗ A_j(t, len_matrix + 1 − s, gam) ∗ sym.diff(fun_f(u, n +
                                  len_matrix − c − t + 1), x, multiplicity[u − 1] − 1)
                                  power_l[i − 1][j − 1] = additionner
                            end
                        end
                    end
                end
            end
        end
        power_l[i − 1][j − 1] = additionner
    end
end
```

```
from __future__ import division
import numpy as np
import numpy . linalg as alg
from numpy . linalg import inv
from scipy . linalg import circulant
from scipy . linalg import companion
from numpy . linalg import matrix_power
import time
from itertools import combinations
import sympy as sym
from time import process_time

# function witch generate doubly_lefkovitch matrix ( expression 1.1 in the article )
def doubly_lefkovitch (a , s , b, d) :
    a = np. atleast_1d(a)
    s = np.atleast_1d(s)
    b = np.atleast_1d(b)
    d = np.atleast_1d(d)
    if a. ndim != 1:
        raise ValueError (" Incorrect shape for a . a must be one - dimensional ")
    if s. ndim != 1:
        raise ValueError (" Incorrect shape for s . s must be one - dimensional ")
    if b. ndim != 1:
        raise ValueError (" Incorrect shape for b . b must be one - dimensional ")
    if a. size != s . size + 1:
        raise ValueError (" Incorrect lengths for f and s . The length "
                          " of s must be one less than the length of f .")
    if a. size != b . size + 2:
        raise ValueError (" Incorrect lengths for f and b . The length "
                          " of b must be one less than the length of f .")
    if a. size != d . size + 1:
        raise ValueError (" Incorrect lengths for f and b . The length "
                          " of b must be one less than the length of f .")
    if s. size == 0:
        raise ValueError (" The length of s must be at least 1.")
    if d. size == 0:
        raise ValueError (" The length of d must be at least 1.")
    tmp = a [0] + s [0] + b [0]+d[0]
    n = a . size
    l = np . zeros (( n , n ) , dtype = tmp . dtype )
    l [0] = a
    l [ list ( range (1 , n) ) , list ( range (0 , n - 1) ) ] = s
    l [ list ( range (1 , n-1) ) ,n -1]= b
    l [ list ( range (1 , n) ) , list (range (1,n))]= d
    return l

#formule 2.4

#formule 2.5
def xi(i,p,l):
    xi=0
    r=len(l)
    if r-i<p:
        xi=0
    if p==0:
        xi=1
    if 0<p & p<=r-i:
        L=[]
        for k in range(i,r):
            L.append(l[k][k])
        Liste=[]
        for i in combinations(L,p):
```

```
            Liste.append(i)
        result=[]
        for j in Liste:
            result.append(np.product(j))
        xi=((-1)**p)*np.sum(result)
    return xi

#formule 2.6
def nabla(i,j,l):
    r=len(l)
    s=[]
    prod=1
    nabla=1
    for m in range(1,r):
        s.append(l[m][m-1])
    if j==i-1:
        nabla=1
    if i!=0 & i<=j & j<=r-1:
        for k in range(i,j+1):
            prod *=s[k-1]
    nabla=prod
    return nabla

#formula 2.8
def matrices_p(l):
    n=len(l)
    p=np.zeros((n,n))
    for i in range(1,n+1):
        for j in range(1,n+1):
            if j>i:
                p[i-1][j-1]=xi(i,j-i,l)/nabla(i,j-1,l)
            if i>j:
                p[i-1][j-1]=0
            if i==j:
                p[i-1][j-1]=1
    return p

#inverse of the matrix P
def inv_mat_p(p) :
    m = inv(p)
    return m

#formula 2.4
def doubly_leslie(l,p,m):
    r=len(l)
    c=np.dot(m,l)
    dl=np.dot(c,p)
    return dl

def matrix_compan (dl , gamma , mu ) :
    e = np . dot (mu , dl)
    b = np . dot (e , gamma )
    return b

def delta(i , s) :
    if i == s :
        return 1
    else :
        return 0

def v3(n ,s ,dl , b) :
    r = len(dl) # matrix length
```

```
    if(n <= r): return delta(s,n)   #case where 1 <= n <=r
    else:
        len_matrix = len(dl)
        gamma = np.zeros(r)
        for i in range (0 , r) : # define gamma[0] to gamma[r-1]
            gamma[i] = b[0][i]
            V = np.zeros(n)
            for i in range (0,r): # define V[0] to V[r-1]
                V [i] = delta (i+1,s)
            for j in range (r,n) : # define V[r] to V[n-1]
                for k in range (0,r) :
                    V [j] += gamma [k]* V[j-k-1]
    return V

def mat_p(dl):
    len_matrix = len(dl)
    gamma = np. zeros (( len_matrix , len_matrix ) )
    for i in range (1,len_matrix+1):
        for j in range (1,len_matrix+1) :
            prod =1
            if i == j & i != len_matrix:
                for k in range (i,len_matrix) :
                    prod *= dl[k][k-1]
                    gamma [i-1][i-1]=1/prod
            if i>j :
                gamma[i-1][j-1]=0
            if j>i:
                if j-i==1:
                    gamma[i-1][j-1]=0
                else :
                    produit =1
                    for s in range ( len_matrix - j+i+1, len_matrix) :
                        fois =1
                        produit *= dl[s][s-1]
                        for h in range (i,len_matrix) :
                            fois *= dl[h][h-1]
                            gamma[i-1][j-1]= -(dl[len_matrix-j+i][len_matrix-1]*produit)/fois
            else :
                gamma[len_matrix-1][len_matrix-1]=1
    return gamma


def inv_mat_p(gamma):
    mu = inv ( gamma )
    return mu



#entries of the powers of doubly Lefkovitch matrix using recursive approach
def power_matrix_doublyLefkovitch_recursive(l,n,p,m,dl,gamma,mu,b):
    len_matrix = len(l)
    power_l = np.zeros((len_matrix,len_matrix))
    for i in range (1,len_matrix+1 ) :
        for j in range (1,len_matrix+1) :
            som =0
            for h in range(1,len_matrix+1):
                for k in range(i,len_matrix+1):
                    for u in range(k , len_matrix+1) :
                        for s in range(1,h+1) :
                            som+=m[h-1][j-1]*p[i-1][k-1]*gamma[k-1][u-1]*mu[s-1][h-1]*
                            ↪  v3(n+len_matrix-u+1,len_matrix+1-s,dl,b)[n+len_matrix-u]
                            power_l[i-1][j-1]=som
```

```
    return power_l


#
def beta_vanderm (dl, s ) :
    len_matrix = len(dl)
    beta = np.zeros(len_matrix)
    mat = np.zeros((len_matrix,len_matrix) , dtype = complex )
    vect = np.zeros(len_matrix)
    evals , evecs = np.linalg.eig(dl)
    unique_elements , counts_elements = np.unique(evals,return_counts=True)
    for n in range (1,len_matrix+1) :
        for f in range (1,len_matrix+1) :
            vect[f-1]= delta (s , f)
            #print(vect)
        result = []
        for k in range (1,len(counts_elements)+1) :
            for j in range (0 , counts_elements[k-1]) :
                result.append (( (n) ** j)*unique_elements[k-1])
    #print(result)
    mat = np.vander(result,n+1)
    mat2=np.delete(mat,[len_matrix], 1)
    #print('mat2',mat2)
    mat3 = np.rot90(mat2)
    #print('mat3',mat3)
    #print('vect',vect)
    beta = np.linalg.solve(mat3,vect)
    return beta

#computing entries of the power of the doubly Lefkovitch matrix using analytic approach
def power_matrix_analytic_doublyL(l,n,p,m,dl,b,gamma,mu):
    len_matrix = len(l)
    power_l=np.zeros((len_matrix,len_matrix),dtype=np.complex_)
    evals , evecs = np.linalg.eig(dl)
    unique_elements , counts_elements = np.unique(evals,return_counts=True)
    for i in range(1,len_matrix+1):
        for j in range(1,len_matrix+1) :
            som =0
            for h in range(1,len_matrix+1):
                for k in range(i,len_matrix+1):
                    for c in range (k,len_matrix+1):
                        for s in range(1,h+1) :
                            f= beta_vanderm(dl,len_matrix+1-s)
                            for u in range(1,len(unique_elements)+1):
                                for t in range(0,counts_elements[u-1]):
                                    som += m[h-1][j-1]*p[i-1][k-1]*gamma[k-1][c-1]*mu[s-1][h-1]*
                                    ↪  f[u-1]*((n+len_matrix-c+1)**t)*
                                    ↪  ((unique_elements[u-1])**(n+len_matrix-c+1))
            power_l[i-1][j-1]=som
    return power_l

def mul_valpropre ( dl ) :
    eigenvalues = np . linalg . eigvals ( dl)
    test , multiplicity = np . unique ( eigenvalues , return_counts = True )
    return test , multiplicity

def fun_f (i ,n ) :
    x = sym . symbols ('x ')
    y =1
    c =1
    for k in range (0 , len ( test ) ) :
        if k != i -1:
```

```
            y *= pow (( x - test [ k ]) , multiplicity [ k ])
            c = pow (x ,n -1) /y
    return c

# fonction calcul ajs verifiee
def fun_a (j ,s ,dl ,b ) :
    sommer =0
    len_matrix = len(dl)
    gamma2 = np.zeros(len_matrix )
    for i in range (0,len_matrix ):
        gamma2 [i] = b[0][len_matrix-i-1]
        # print gamma2
        for o in range (j ,len(dl)) :
            print(len(v3(o,s,dl,b)))
            sommer += gamma2 [o-j ]* v3(o,s,dl,b)[len(v3(o,s,dl,b))-1]
    return sommer

def fact ( n ) :
    """ fact (n ) : calcule la factorielle de n ( entier >= 0) """
    if n <2:
        return 1
    else :
        return n* fact (n -1)


def gamma_function(b):
    r = len(b) # matrix lenght
    len_matrix = len(b)
    gam = np.zeros(r)
    for i in range (0 , r):
        gam[i] = b[0][i] # define gamma[0] to gamma[r-1]
    return gam

def A_j(j,s,gam):
    Aj = 0
    r = len(gam)
    if(j > r or j<0): return Aj
    for i in range(j,r+1):
        #print("i = ",i, "g = ",r-i+j )
        Aj += gam[r-i+j-1]*delta(i,s)
    return Aj


def power_matrix_derivative_method(l,n,m,p,gamma,mu,dl,b):
    len_matrix = len (l)
    power_l = np.zeros((len_matrix,len_matrix) , dtype = np . complex_ )
    x = sym.symbols ('x ')
    for i in range (1 , len_matrix+1 ) :
        for j in range (1 , len_matrix +1) :
            additionner =0
            for h in range(1,len_matrix+1):
                for k in range(i,len_matrix+1):
                    for c in range (k , len_matrix+1 ) :
                        for s in range (1 , h +1) :
                            for u in range (1 , len ( test ) +1) :
                                for t in range (1, len_matrix +1) :
                                    additionner += m[h-1][j-1]*p[i-1][k-1]*gamma[k-1,c-1]*mu[s-1,h-1]*
                                    ↪  A_j(t,len_matrix+1-s,gam)* sym.diff(fun_f(u,n+len_matrix-c-t+1),x,
                                    ↪  multiplicity[u-1]-1).subs({x:test[u-1]})/fact(multiplicity[u-1]-1)
                                    power_l[i-1][j-1]= additionner
    return power_l
```

```
#taille 3*3
#l=doubly_lefkovitch([2,24,20],[0.6,0.8],[9],[1,3])
l=doubly_lefkovitch([3,50,10],[0.75,0.25],[150],[2,5])
#taille 4*4
#l=doubly_lefkovitch([4,14,10,11],[0.6,0.8,0.2],[5,10],[1,3,4])
#taille 5*5
#l=doubly_lefkovitch([2,24,20,4,5],[0.6,0.8,0.2,0.1],[9,5,3],[1,3,4,6])
#taille 6*6
#l=doubly_lefkovitch([2,24,20,4,5,7],[0.6,0.8,0.2,0.1,0.11],[9,5,3,4],[1,3,4,6,8])
#taille 7*7
#l=doubly_lefkovitch([2,24,20,30,50,75,88],[0.6,0.2,0.8,0.75,0.25,0.11],[9,7,11,5,7],[1,3,5,12,44,89])
print('doubly lefkovitch=',l)
p=matrices_p(l)
print('p=',p)
m=inv_mat_p(p)
print('m=',m)
dl=doubly_leslie(l,p,m)
print('doubly leslie=',dl)
gamma=mat_p(dl)
print('gamma=',gamma)
mu=inv_mat_p(gamma)
b=matrix_compan(dl,gamma,mu)

gam = gamma_function(b)
test , multiplicity=mul_valpropre ( dl )

t1_start = process_time()
power_lr=power_matrix_doublyLefkovitch_recursive(l,6,p,m,dl,gamma,mu,b)
t1_stop = process_time()
print('power_lr=',power_lr)

t2_start = process_time()
print('matrix_power=',matrix_power(l,6))
t2_stop = process_time()

t3_start = process_time()
power_ld=power_matrix_derivative_method(l,6,m,p,gamma,mu,dl,b)
t3_stop = process_time()
print('power_ld=',power_ld)

t4_start= process_time()
power_la=power_matrix_analytic_doublyL(l,6,p,m,dl,b,gamma,mu)
t4_stop=process_time()
print('power_la',power_la)

print("Elapsed time 1:", t1_stop, t1_start)

print("Elapsed time during the whole program in seconds:", t1_stop-t1_start)

print("Elapsed time 2:", t2_stop, t2_start)

print("Elapsed time during the whole program in seconds:", t2_stop-t2_start)
print("Elapsed time 3:", t3_stop, t3_start)

print("Elapsed time during the whole program in seconds:", t3_stop-t3_start)
print("Elapsed time 4:", t4_stop, t4_start)

print("Elapsed time during the whole program in seconds:", t4_stop-t4_start)
```

Acknowledgment

The authors express their sincere gratitude to the reviewers. The second author is supported by PPGEdumat and the Profmat programs of the INMA-UFMS. He expresses his sincere thanks to the INMA and the UFMS for their valuable support and encouragements.

References

[1] A. Aloui, M. Rachidi, B. El Wahbi , *On a numerical approach for the powers of the doubly Leslie and doubly companion matrices with applications*, Int. J. Math. Comput. Sci., **16** (2021), 613–638. 1, 2.1, 2.1, 2.2

[2] S. Arela-Pérez, H. Nina, J. Pantáz, H. Pickmann-Soto, E. Valero, *Construction of Lefkovitch and doubly Lefkovitch matrices with maximal eigenvalues and some diagonal elements prescribed*, Linear Algebra Appl., **626** (2021), 152–170. 1

[3] R. Ben Taher, N. Naassi, Y. Elkettani, M. Rachidi, *Another approach for Leslie model via linear difference equations*, J. Interdiscip. Math., **24** (2021), 1–25. 1

[4] R. Ben Taher, N. Naassi, M. Rachidi, *On the Leslie matrices, Fibonacci sequences and population dynamics*, J. Discrete Math. Sci. Cryptogr., **20** (2017), 565–594. 1

[5] R. Ben Taher, M. Rachidi, *Solving some generalized Vandermonde systems and inverse of their associate matrices via new approaches for the Binet formula*, Appl. Math. Comput., **290** (2016), 267–280. 2.2, 2.3

[6] F. Dubeau, W. Motta, M. Rachidi, O. Saeki, *On weighted* r*-generalized Fibonacci sequences*, Fibonacci Quart., **35** (1997), 102–110. 1

[7] M. Mouline, M. Rachidi, *Application of Markov chains properties to* r*-generalized Fibonacci sequences*, Fibonacci Quart., **37** (1999), 34–38. 1

[8] M. B. Usher, *A matrix model for forest management*, Biometrics, **25** (1969), 309–315. 1

[9] W. Wanicharpichat, *Explicit minimum polynomial, eingenvector and inverse formula of doubly Leslie matrix*, J. Appl. Math. Inform., **33** (2015), 247–260. 2.1