KRYLOV AND POWER METHODS FOR EIGENVALUES AND EIGENVECTORS OF A SYMMETRIC MATRIX

Kodirova Malokhat Obidjon kizi
Student of master's degree, "Informatics" Department,
Namangan State University

Abstract: *The article discusses classical methods for finding the eigenvalues and eigenvectors of matrices, such as direct determination using the commands of the mathematical system Mathcad, as well as a two-stage classical method that utilizes the methods of Krylov, and Power methods. For all methods, enlarged algorithms have been built, consisting of a sequence of mathematical formulas and commands of the Mathcad mathematical system. In this case, Mathcad acts as an executor of an enlarged mathematical algorithm for solving the problem. For all methods, the eigenvalues of the matrices are obtained in four ways: by the method of the internal command Mathcad eigenvals(A),by the internal function of Python, with the help of Python code written for the methods and by the most proposed method. Only if the results of these methods coincide, the enlarged algorithm is* considered *correct.*

Keywords: *algorithm, large operation, algorithmic language, algorithm executor, Mathcad-executor of an enlarged algorithm, algorithms and programs in Mathcad and Python for eigenvalues of matrices.*

МЕТОДЫ ЛЕВЕРЬЕ И МИКЕЛАДЗЕ ДЛЯ СОБСТВЕННЫХ ЗНАЧЕНИЙ И СОБСТВЕННЫХ ВЕКТОРОВ СИММЕТРИЧНОЙ МАТРИЦЫ

Кодирова Малохат Обиджон кизи
Магистрант кафедры информатики,
Наманганский государственный университет

Аннотация. *Статья рассматривает классические методы нахождения собственных значений и собственных векторов матриц, такие как прямое определение с использованием команд математической системы Mathcad, а также двухступенчатый классический метод, использующий методы Крылова и методы степени. Для всех методов были построены расширенные алгоритмы, состоящие из последовательности математических формул и команд математической системы Mathcad. В этом случае Mathcad действует как исполнитель расширенного математического алгоритма для решения задачи. Для всех методов собственные значения матриц получаются четырьмя способами: с помощью метода внутренней команды Mathcad eigenvals(A), с помощью внутренней функции Python, с помощью написанного на Python кода для методов и с помощью наиболее предлагаемого* *метода. Только если результаты этих методов совпадают, расширенный алгоритм считается правильным.*

Ключевые слова: *алгоритм, крупная операция, алгоритмический язык, испольнитель алгоритма, Mathcad-исполнитель укрупнённого алгоритма, алгоритмы и программы в Mathcad и Python для собственных значений матриц.*

SIMMETRIK MATRITSANING XOS SON VA XOS VEKTORLARI UCHUN KRILOV, DARAJA VA TESKARI DARAJA USULLARI

Qodirova Malohat Obidjon qizi
Namangan davlat universiteti, Informatika va raqamli texnologiyalar kafedrasi magistranti

Annotatsiya: *Ushbu maqolada matritsalarning xos son va xos vektorlarini aniqlashning Mathcad matematik tizimining buyruqlaridan foydalanib to'g'ridan to'g'ri aniqlash, shuningdek Krilov va daraja usullarini qo'llovchi ikki bosqichli klassik usullari kabilarni muhokama qilinadi. Barcha usullar uchun matematik formulalar ketma – ketligini va Mathcad matematik tizimi buyruqlarini o'z ichiga oluvchi kengaytirilgan algoritmlar qurilgan.Bu o'rinda Mathcad masalani yechish uchun kengaytirilgan matematik algoritmning bajaruvchisi sifatida xizmat qiladi. . Barcha metodlar uchun matritsalarning xos sonlari to'rt xil usulda aniqlanadi: Mathcadning ichki funksiyasi - eigenvals (A), Pythonning ichki funksiyasi, usul uchun yozilgan Python kod va eng ko'p taklif etilgan usul bo'yicha. Usullarning natijalari mos kelgandagina kengaytirilgan algoritm to'g'ri deb qabul qilinadi.*

Kalit so'zlar: *algoritm, kengaytirilgan amal, algoritmik til, algoritm bajaruvchisi, kengaytirilgan algoritmning Mathcad-bajaruvchisi, Mathcadda va Pythonda matritsalarning xos sonlari uchun algoritmlar va dasturlar.*

I. INTRODUCTION

The problem of determining eigenvalues and vectors is as follows:

$$Ax = \lambda x, x \neq 0, \lambda, x = ? \qquad (1)$$

where is the order of the matrix is $n \times n$. In this equation, the unknowns are: n-scalar quantities of λ and n eigenvectors $x \neq 0$. Academicians S.L. Sobolev, I.A. Krylov, D.K. Faddeev, G.I. Marchuk, A.A. Samarsky, N.N. Yanenko, S.K. Godunov, alongside prominent figures such as Jacobi, Givens, Wilkinson, Frobenius, and Francis, have contributed significantly to both the theory and practical application of eigenvalues and eigenvectors.

Since problem (1) represents a homogeneous system of linear equations with respect to x, a solution exists if and only if:

$$\det(A-\lambda E)=|A-\lambda E|=(-1)^n(\lambda^n-p_1\lambda^{n-1}-\ldots p_{n-1}\lambda-p_n)=0$$

(2)

This algebraic equation is called the characteristic (secular) equation (in the comments in the algorithms it will be referred to as CHE for short), where

$$p_1=\sum_{i=1}^{n}a_{ii},\ p_2=\sum_{i<j}\begin{vmatrix}a_{ij} & a_{ij}\\ a_{ji} & a_{jj}\end{vmatrix},\ p_3=\sum_{i<k<j}\begin{vmatrix}a_{ii} & a_{ik} & a_{ij}\\ a_{ki} & a_{kk} & a_{kj}\\ a_{ji} & a_{jk} & a_{jj}\end{vmatrix},\ldots,\ p_n=\det(A).$$

(3)

The classic method of searching for eigenvalues consists of two stages:

1) determination of the coefficients of the characteristic equation;

2) solution of the characteristic equation.

To find all eigenvalues of symmetric matrices, iterative methods of Jacobi and Givens rotations have been developed. Iterative methods for searching for extreme eigenvalues and classical universal methods of Leverrier, Faddeev, Krylov, interpolation, Rutishauser LU, Francis QR have been developed for any matrix.

The theoretical foundations of the problem are set out in [1-4]. An algebraic approach to approximate eigenvalue methods is available in [5,6]. The eigenvalue problem for asymmetric matrices is discussed in [5], where the programs that have at that time are analyzed. Programs in the Algol language are available in [6]. In [7] for the first time appeared programs of methods for extreme eigenvalues and the method of interpolation in the three languages Basic, Pascal and Fortran. Programs in BASIC are available for extreme eigenvalues, Jacobi, QR. Programs - enlarged algorithms in the Python language appeared [8]. [9,10] has enlarged algorithms for finding eigenvalues, Mathcad.[11] has a large number of Mathcad programs.

In mathematical systems, the solution to the problem can be found in 4 ways:

1) Command-based-system-large calculator;

2) based on a mathematical algorithm – the system is the executor of the algorithm;

3) based on an internal language-system programming environment;

4) based on the interactive method-system-training program

For solving problems of computational methods, the second method of solving problems plays an important role, since in this method the problem is solved only with the help of a mathematical algorithm for solving the problem. Mathcad acts as the executor of the enlarged algorithm, and the user acts as the creator of the algorithm. By an enlarged algorithm, we mean, a sequence of mathematical formulas and commands of a mathematical system that gives a solution to the problem. It is these instructions of a mathematical system that are a large command, for example, an internal function for calculating the powers of a matrix, a determinant, an inverse matrix, an integral, the solution of a differential equation, etc. We demonstrate our thoughts with examples:

finding the eigenvalues of the matrix by the classical method of direct determination, Leverrier, Faddeev, interpolation and Krylov, QR algorithm. In these methods, the secular equation is found first, and then the eigenvalues. We check the adequate results of our algorithms with the internal function eigenvals(A) and eigenvecs(A) to find the eigenvalues and vectors of the matrix.

In what follows, we will denote by $A=[a_{ij}]_{i,j=1}^{n}$-matrix, E – is a unit matrix.

In the theory of eigenvalues, the well-known expansions of matrices according to Jordan and Schur are of great importance [2]. Each matrix is like a jordan and a shur shape: $A=SJS^{*}\,(A=SDS^{*}), A=QRQ^{*}$, where J - is the Jordanian form, D - is the diogonal form, R - is the upper triangular form, which contain all eigenvalues, Q - is the orthogonal matrix. Note also what is the characteristic polynomial for the Frobenius matrix:

$$A=\begin{bmatrix} -p_1 & -p_2 & . & -p_{n-1} & -p_n \\ 1 & 0 & . & 0 & 0 \\ 0 & 1 & . & 0 & 0 \\ . & . & . & . & . \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

I. Methods of Solution. Extended algorithms and programs.

1. Mathcad command method. Let's Compute Your Own Values

and matrix vectors using the Mathcad command and check the correctness of the result using the formula $v=\frac{1}{\lambda}Av$:

ORIGIN: =1 $A:=\begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 1 \\ 3 & 4 & 1 & 2 \\ 4 & 1 & 2 & 3 \end{bmatrix}$ $n:=cols(A)\ E:=identity(n)$ //Specifying A, E matrices

$\lambda:=eigenvals(A)\ v:=eigenvecs(A)\quad \lambda=\begin{bmatrix} -2.828 \\ -2 \\ 2.828 \\ 10 \end{bmatrix}\ v=\begin{bmatrix} -0.6533 & -0.5 & 0.2706 & 0.5 \\ -0.2706 & 0.5 & -0.6533 & 0.5 \\ 0.6533 & -0.5 & -0.2706 & 0.5 \\ 0.2706 & 0.5 & 0.6533 & 0.5 \end{bmatrix}$ // λ ,v

$i:=1..n\ vv^{<i>}:=A\frac{v^{<i>}}{\lambda_i}\ vv=\begin{bmatrix} -0.6533 & -0.5 & 0.2706 & 0.5 \\ -0.2706 & 0.5 & -0.6533 & 0.5 \\ 0.6533 & -0.5 & -0.2706 & 0.5 \\ 0.2706 & 0.5 & 0.6533 & 0.5 \end{bmatrix}$ // Checking the correctness λ ,v

2. **Classic universal method.** Stages of the method:

1) Calculation of the secular equation: $D(\lambda)=|A-\lambda E|=(-1)^n(\lambda^n-\sum_{i=1}^n p_i\lambda^i)=0$.

2) Solution of the secular equation: $D(\lambda_i)=|A-\lambda_i E|=0, \lambda=\lambda_i, i=1..n$.

Previously, this method was used only for small-order matrices: $n\le 3$.

For the two-step process, let's create an extended algorithm:

ORIGIN: =1 $A:=\begin{bmatrix}1&2&3&4\\2&3&4&1\\3&4&1&2\\4&1&2&3\end{bmatrix}$ $n:=cols(A)$ $E:=identity(n)$ //Specifying A, E matrices

$D(x):=(-1)^n|A-E*x| \rightarrow 160+64x-28x^2-8x^3+x^4$ // CHE command Mathcad

$p(x):=D(x)\,coeffs, x \rightarrow [160 \quad 64 \quad -28 \quad -8 \quad 1]^T$ // Allocation of CHE coefficients

$r:=polyroots(p)$ $r=[-2.828 \quad -2 \quad 2.828 \quad 10]^T$ // Output of eigenvalues

$\lambda=eigenvals(A)$ $v=eigenvecs(A)$ $\lambda=\begin{bmatrix}-2.828\\-2\\2.828\\10\end{bmatrix}$ $v=\begin{bmatrix}-0.653&-0.5&0.271&0.5\\-0.271&0.5&-0.653&0.5\\0.653&-0.5&-0.271&0.5\\0.271&0.5&0.653&0.5\end{bmatrix}$ // commands

II. Krylov's Method for Eigenvalues and Eigenvectors [1].

Let $D(\lambda)=|A-\lambda E|=(-1)^n(p_1+p_2\lambda+...+p_{n-1}\lambda+p_n\lambda^{n-1}+\lambda^n)=0$ is the secular equation of matrix A. According to the Hamilton-Kelly theorem, the matrix satisfies its secular equation: $D(A)=0: D(A)=|A-AE|=0$ (brief proof).

$$D(A)=|A-AE|=|0|=(-1)^n(p_1+p_2A+...+p_{n-1}A^{n-1}+p_nA^n)=0 \Rightarrow (p_1+p_2A+...+p_{n-1}A^{n-1}+p_nA^n)=0$$

From here we get $p_1E+p_2A+...+p_nA^{n-1}=-A^n$. Multiply the last equation by an arbitrary non-zero vector $x\in R^n$ and get the vector equation:

$$p_1Ex+p_2Ax+...+p_nA^{n-1}x=-A^nx.$$

By introducing vectors $x^{(k)}:=A^{(k)}*x, k=1..n$, for the unknown coefficients of the secular equation, we obtain a system of linear equations:

$$\begin{bmatrix}x_{11}&x_{12}&.&x_{1n}\\x_{21}&x_{22}&.&x_{2n}\\.&.&.&.\\x_{n1}&x_{n2}&.&x_{nn}\end{bmatrix}*\begin{bmatrix}p_1\\p_2\\.\\p_n\end{bmatrix}=\begin{bmatrix}d_1\\d_2\\.\\d_n\end{bmatrix}, x_{ij}=(A^j*x)_i \Leftrightarrow \sum_{j=1}^n p_jx_{ij}=d_i, i=1..n$$

III. Finding eigenvectors in the Krylov method. In equation (4), we put $x=C_1$ $q=-p$. We express the characteristic equation in the form $q=-p$. According to the

Hamilton-Kelley theorem, we have $q_1E+q_2A+...+q_nA^{n-1}=A^n$. Let's multiply it by the vector C_1.

$$q_1C_1+q_2C_2+...+q_nC_n=A^nC_1=C_{n+1},C_{i+1}=AC_i,i=1..n \Leftrightarrow \sum_{j=1}^{n}q_jC_{ij}=C_{i,n+1}$$

We denote the eigenvector corresponding to the eigenvalue $\lambda=\lambda_i$ as:

$$X_i=\beta_{i1}C_1+\beta_{i2}C_2+...+\beta_{in}C_n=\sum\nolimits_{k=1}^{n}\beta_{i,k}C^{<k>},AX_i=\lambda_iX_i,i=1..n$$

(5)

Taking into account $\lambda_iX_i=AX_i$ equation (4), we can write:

$$\lambda_iX_i=\lambda_i(\beta_{i1}C_1+\beta_{i2}C_2+...+\beta_{in}C_n)=\beta_{i1}C_2+\beta_{i2}C_3+...+\beta_{in}C_{n+1}\Leftrightarrow$$
$$\lambda_i(\beta_{i1}C_1+\beta_{i2}C_2+...+\beta_{in}C_n)=\beta_{i1}C_2+\beta_{i2}C_3+...+\beta_{in}C_{n+1}=$$
$$\beta_{i1}C_2+\beta_{i2}C_3+...+\beta_{in-1}C_n+\beta_{in}(q_nC_1+q_{n-1}C_2+...+q_1C_n)$$

(6)

Here, we equate the coefficients of C_i in different parts of the equation:

$$q_1\beta_{in}=\lambda_i\beta_{i1},\beta_{i1}+q_2\beta_{in}=\lambda_i\beta_{i2},\beta_{i2}+q_3\beta_{in}=\lambda_i\beta_{i3},...,\beta_{in-1}+q_n\beta_{in}=\lambda_i\beta_{in}.$$

(7)

From here, starting from the last equation, we find sequentially:

$$\beta_{in-1}=q_1\beta_{in}=(\lambda_i-q_n)\beta_{in},$$
$$\beta_{in-2}=\lambda_i\beta_{in-1}-q_{n-1}\beta_{in}=[\lambda_i(\lambda_i-q_n)-q_{n-1}]\beta_{in}=[\lambda_i^2-q_n\lambda_i-q_{n-1}]\beta_{in},$$
$$\beta_{in-3}=\lambda_i\beta_{in-2}-q_{n-2}\beta_{in}=[=[\lambda_i^3-q_n\lambda_i^2-q_{n-1}\lambda_i-q_{n-2}]\beta_{in},...,$$
$$\beta_{i1}=\lambda_i\beta_{i2}-q_2\beta_{in}=[\lambda_i^{n-1}-q_n\lambda_i^{n-2}-q_{n-1}\lambda_i^{n-3}-...-q_3\lambda_i-q_2]\beta_{in}$$

One of the solutions to these equations is defined as follows:

$$\beta_{in}=1,\beta_{in-1}=\lambda_i-q_n,\beta_{in-2}=[\lambda_i^2-q_n\lambda_i-q_{n-1}],...,$$
$$\beta_{i1}=\lambda_i^{n-1}-q_n\lambda_i^{n-2}-q_{n-1}\lambda_i^{n-3}-...q_2=\lambda_i^{n-j}+\sum\nolimits_{k=1}^{n-j}q_{n+1-k}\lambda_i^{n-j-k},$$
$$\lambda_i^n-q_n\lambda_i^{n-1}-q_{n-1}\lambda_i^{n-2}-...q_2\lambda_i-q_1=D(\lambda_i)=0$$

(8)

After determining the coefficients, we find the eigenvector:

$$X_i=\beta_{i1}C_1+\beta_{i2}C_2+...+\beta_{in}C_n=\sum\nolimits_{k=1}^{n}\beta_{i,k}C^{<k>},AX_i=\lambda_iX_i,i=1..n$$

(9)

Here's a program that finds both eigenvalues and eigenvectors:

ORIGIN: =1 $\quad A:=\begin{bmatrix}1&2&3&4\\2&3&4&1\\3&4&1&2\\4&1&2&3\end{bmatrix}\ n:=cols(A)\ E:=identity(n)\ \beta:=identity(n)\ y:=\begin{bmatrix}1\\0\\0\\0\end{bmatrix}$

$D(x):=(-1)^n\left|A-E*x\right|\rightarrow160+64x-28x^2-8x^3+x^4$ //CHE with the help of Mathcad command

$p(x):=D(x)coeffs,x\rightarrow\begin{bmatrix}160&64&-28&-8&1\end{bmatrix}^T$ // Allocation of CHE coefficients

$r := polyroots(p)\ r = \begin{bmatrix} -2.828 & -2 & 2.828 & 10 \end{bmatrix}^T$ // Eigenvalues on command

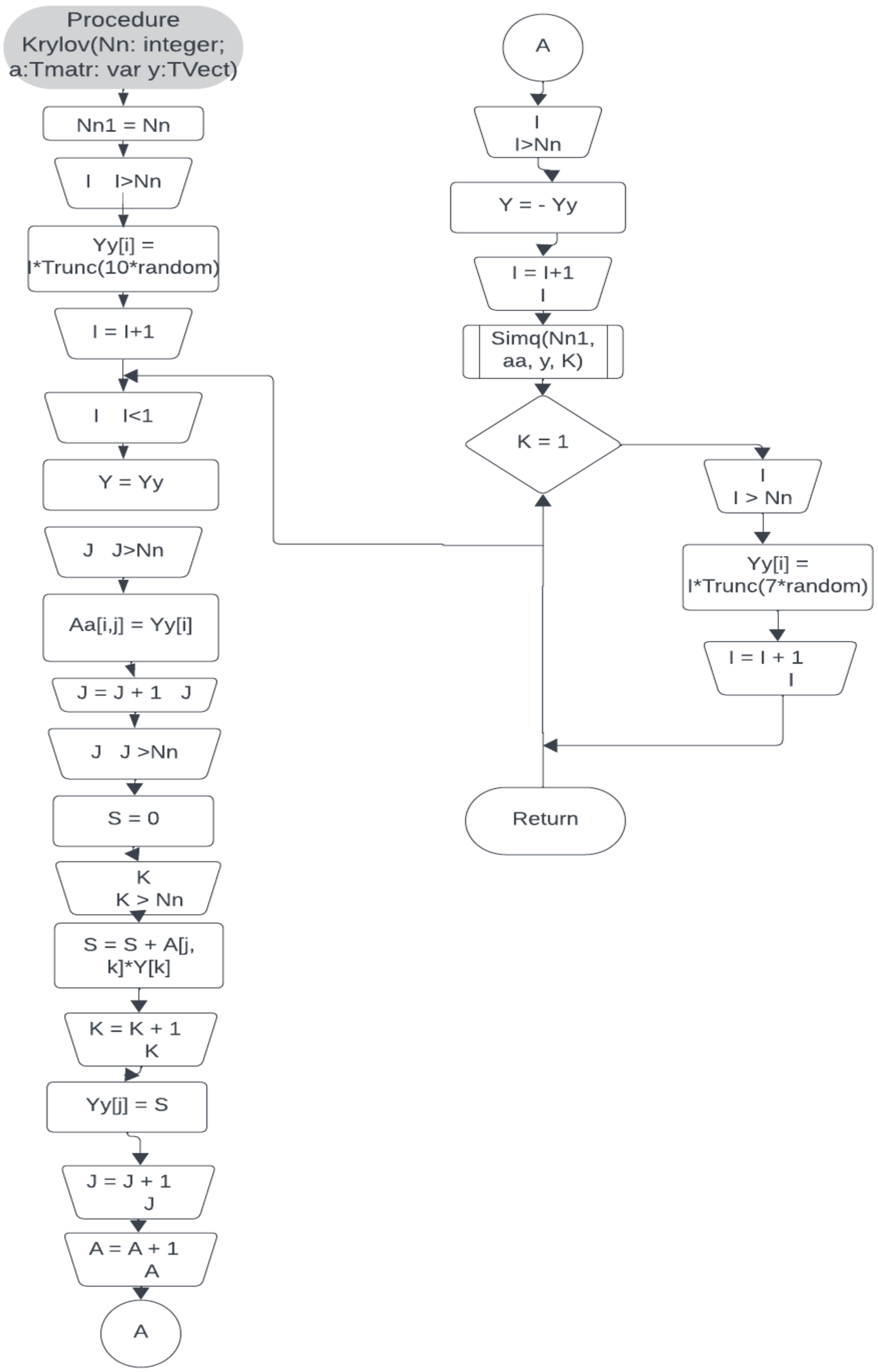
$$\lambda = eigenvals(A)\ v = eigenvecs(A)\ \lambda = \begin{bmatrix} -2.828 \\ -2 \\ 2.828 \\ 10 \end{bmatrix}\ v = \begin{bmatrix} -0.653 & -0.5 & 0.271 & 0.5 \\ -0.271 & 0.5 & -0.653 & 0.5 \\ 0.653 & -0.5 & -0.271 & 0.5 \\ 0.271 & 0.5 & 0.653 & 0.5 \end{bmatrix}$$ //checking

$$k := 1..n\ \ C^{<k>} := A^{k-1} * y\ \ C = \begin{bmatrix} 1 & 1 & 30 & 240 \\ 0 & 2 & 24 & 244 \\ 0 & 3 & 22 & 256 \\ 0 & 4 & 24 & 260 \end{bmatrix}\ d := A^n * y\ \ p := (-1) * C^{-1} * d\ \ p = \begin{bmatrix} 160 \\ 64 \\ -28 \\ -8 \end{bmatrix}$$

//coef.

$$x^{<i>} := \sum_{k=1}^{n} \beta_{i,k} * C^{<k>}\ \ x^{<i>} := \frac{x^{<i>}}{\max(|x^{<i>}|)}$$ // Normalization of eigenvectors

$$x = \begin{bmatrix} -0.6533 & 0.5 & -0.2706 & 0.5 \\ -0.2706 & -0.5 & 0.6533 & 0.5 \\ 0.6533 & 0.5 & 0.2706 & 0.5 \\ 0.2706 & -0.5 & -0.6533 & 0.5 \end{bmatrix}\ v = \begin{bmatrix} -0.6533 & 0.5 & -0.2706 & 0.5 \\ -0.2706 & -0.5 & 0.6533 & 0.5 \\ 0.6533 & 0.5 & 0.2706 & 0.5 \\ 0.2706 & -0.5 & -0.6533 & 0.5 \end{bmatrix}$$ //eigenvectors



Block - scheme of algorithm of Krylov method

Input: n – natural number, order of the given matrix

Output – q – array, n – dimentional real vector, eigenvectors

Here is the python code for Krylov's method:

```
#Krylov's Method
import numpy as np
import math
A = np.array([[1, 2, 3, 4], [2, 3, 4, 1], [3, 4, 1, 2], [4, 1, 2, 3]])
x = np.array([1.0, 0.0, 0.0, 0.0])
n = A.ndim #n = 4
E = np.identity(n)
M = np.zeros(n)
D = np.array([1.0, 2.0, 3.0, 4.0])
d = np.array([1.0, 2.0, 3.0, 4.0])
#matrix M
for i in range (1, n + 1):
  M[i, 1] = x[i]
  B = E
for j in range (2, n + 1):
  for i in range (1, n + 1):
    A = np.dot(A, B)
    B = A
    Y = np.dot(A, x)
    M[i, j] = y[i]
#right hand of Mp = d
A = np.dot(A, B)
d = -np.dot(A, x)
#solving of Mp = d
p = np.linalg.solve(M, d) # p = [p[1], p[2], .. p[n]]
p = np.insert(p, 0, 1) #p = [1, p[1], p[2], .. p[n]]
print('koef.pol p: \n', p)
r = np.roots(p)
print('lambda of A:\n', r)
```

The results are as follows:

```
koef pol:
1 -8 -28 64 160
roots:
10 2.828427712 -2.828427712 -2
```

IV. POWER METHOD

The power method is an iterative algorithm used to find the eigenvector associated with the largest eigenvalue (also known as the dominant eigenvector) of a square matrix. It works by repeatedly multiplying a randomly chosen initial vector by the matrix and then normalizing the resulting vector. Over time, the vector will converge to the dominant eigenvector.

Algorithm:

1. Initialize:

○ Choose a random vector x0 (non-zero) of the same dimension as the matrix.

○ Set a tolerance tol for convergence and an iteration counter k to 0.

2. Iteration:

○ Repeat the following steps until convergence or maximum iterations are reached:

▪ Multiply the current vector by the matrix: x_new = Ax_k

▪ Normalize the resulting vector: x_new = x_new / ||x_new||

▪ Calculate the Rayleigh quotient: lambda_k = (x_new^T * Ax_k) / (x_new^T * x_new)

▪ Check for convergence:

▪ If the difference between successive eigenvalues is less than the tolerance, stop the iteration.

▪ Otherwise, increment the iteration counter k.

3. Output:

○ The final normalized vector x_k is the dominant eigenvector.

○ The Rayleigh quotient lambda_k is the associated eigenvalue.

Additional Notes:

• The power method is particularly useful for large matrices where other methods might be computationally expensive.

• The convergence of the power method depends on the initial vector chosen. Choosing a vector that is close to the dominant eigenvector can accelerate convergence.

• The power method can also be used to find other eigenvectors by deflating the matrix after each iteration.

Here is python code for the algorithm above:

```
import numpy as np

def power_method(A, tol=1e-6, max_iter=100):
  """
  Finds the dominant eigenvector and eigenvalue of a square matrix using the power method.

  Args:
    A: The square matrix.
    tol: The tolerance for convergence.
    max_iter: The maximum number of iterations.

  Returns:
    eigenvector: The dominant eigenvector.
    eigenvalue: The associated eigenvalue.
  """

  # Initialize
  n = A.shape[0]
  x0 = np.random.rand(n)
  x0 = x0 / np.linalg.norm(x0)
  k = 0

  while True:
    # Iteration
    x_new = np.dot(A, x0)
    x_new = x_new / np.linalg.norm(x_new)
    lambda_k = np.dot(x_new.T, np.dot(A, x_new)) / np.dot(x_new.T, x_new)

    # Check for convergence
    if k > 0 and abs(lambda_k - lambda_k_prev) < tol:
      break

    # Update
    x0 = x_new
    lambda_k_prev = lambda_k
    k += 1

    # Check maximum iterations
    if k >= max_iter:
      print("Maximum iterations reached.")
      break

  return x_new, lambda_k

# Example usage
A = np.array([[1, 2, 3, 4],
              [2, 3, 4, 1],
              [3, 4, 1, 2],
              [4, 1, 2, 3]])

eigenvector, eigenvalue = power_method(A)

print("Eigenvector:", eigenvector)
print("Eigenvalue:", eigenvalue)
```

The results are as follows:

```
Eigenvector: [0.50001551 0.50001216 0.49998417 0.49998816]
Eigenvalue: 9.999999990296292
```

The code provided only finds the dominant eigenvector and eigenvalue of the matrix which can be considered correct. Iterative methods usually give approximate values, not exact quantities.

V. DISCUSSION OF RESULTS

The article considers eigenvalues of the choosen matrix through these methods: Kylov and power iterative. Python scripts have been written for them and checked through the assistance of internal function of Python and Mathcad mathematical system commands. The results are the same.

VI. Acknowledgement.

The author would like to thank the professors of the Faculty of Mathematics of NSU A.Imomov, N. Otakhonov, E.Abduqodirov, M. Dadakhonov, Sh. Boltiboev, M. Eshnazarova, M. Ikramova for useful discussions.

VII. REFERENCES:

1. Shary S.P. Course of Computational Methods. ICT SORAN, N.:2018.-607 p.
2. Shup T.E. Applied Numerical Methods in Physics and Technology.Moscow: VSH, 1990.-256 p.
3. Bellman.Introduction to the Theory of Matrices.Moscow: Nauka, 1976.-352 p.
4. Burden R.L. Numerical Analysis. Books Cole. Boston, USA. - 2010.-895 p.
5. Ikramov Kh.D. Unsymmetrical problem of proper values. Moscow: Nauka, 1991, 240 p.
6. Wilkinson J., Rainsch K. Handbook of algorithms in the Algol language. Moscow: Mashinostroenie Publ., 1976.-390 p.
7. Mudrov E.M. Programs for PCs in the language of Basic, Pascal, Fortrane. Tomsk, MP Rasko, 1992.-272 p.
8. Okhorzin V.A. Applied Mathematics in the Mathcad System. St. Petersburg, 2008.-350 p.
9. Kiusalaas J. Numerical Methods in Ingineering with Phyton 3. NY. - 2013.-438p.
10. https://github.com/AnduezaGarcia/PathPlanning