UFSM

CIÊNCIA e NATURA

ISSN 2179-460X
OPEN ACCESS

Ci. e Nat., Santa Maria, v. 47, esp. 1, e90148, 2025 • https://doi.org/10.5902/2179460X90148
Submissão: 09/12/24 • Aprovação: 15/12/24 • Publicação: 03/02/25

IV Jornada de Matematica e Matematica aplicada UFSM

Uma implementação do algoritmo de localização de autovalores de grafos *Threshold* em *Python*

An implementation of the eigenvalue localization algorithm for threshold graphs in Python

Andressa de Oliveira Eckhardt [I], **João Roberto Lazzarin** [I], **Fernando Colman Tura** [I]

[I]Departamento de Matemática, Universidade Federal de Santa Maria, Santa Maria, RS, Brasil

RESUMO

Um grafo *threshold* de ordem n pode ser caracterizado através de uma sequência binária $(b_1, b_2, \ldots, b_n)$. Apresentaremos os detalhes do algoritmo conhecido como *Diagonalize* que fornece a localização dos autovalores da matriz de adjacência de um grafo *threshold* apenas baseando-se na geometria deste, sem recorrer a manipulações diretas com a matriz em si. Veremos que a caracterização binária é uma ferramenta facilitadora para uma implementação linear deste algoritmo em *Python*.

Palavras-chave: Grafo *threshold*; Algoritmo *Diagonalize*; Implementação; *Python*

ABSTRACT

A threshold graph of order n may be characterized by a binary sequence $(b_1, b_2, \ldots, b_n)$.We shall present the particulars of the algorithm known as *Diagonalize*, which provides the location of the eigenvalues of the adjacency matrix of a threshold graph, relying solely on its geometry, without the need for direct manipulations with the matrix itself. We shall observe that the binary characterization is a facilitative tool for a linear implementation of this algorithm in Python.

Keywords: Threshold graph; Diagonalize algorithm; Implementation; Python

1 INTRODUÇÃO

A Teoria de Grafos possui várias aplicações práticas em diversos campos (Mahadev and Peled, 1995), engloba uma subárea conhecida como Teoria Espectral de Grafos. Esta subárea é dedicada à análise das características estruturais de grafos, empregando matrizes associadas a eles e seus respectivos autovalores (Tura, 2013). Nesse contexto, o cálculo dos autovalores de uma matriz associada a um grafo assume uma relevância significativa. Contudo, é importante ressaltar que os autovalores, sendo raízes de um polinômio, geralmente não são determinados de maneira explícita e muitas vezes faz-se necessário recorrer-se a modelos computacionais para a obtenção de valores aproximados dessas raízes.

Neste artigo apresentamos resultados apresentaremos diversos resultados que podem ser encontrados em Jacobs et al. (2013). A essência de nossa contribuição reside na apresentação de uma argumentação meticulosamente elaborada que valida a eficácia do algoritmo proposto pelos autores, ecoando assim suas concepções originais e, apresentar uma implementação na linguagem *Python* do algoritmo que permite-nos localizar autovalores da matriz de adjacência de grafos *threshold*. A escolha da linguagem de programação *Python* se deve à sua sofisticação e natureza de código aberto que facilita o acesso a todos que se interessarem.

2 AUTOVALORES DE MATRIZES SIMÉTRICAS

Um grafo (simples) G consiste de um conjunto $V(G)$ de objetos chamados vértices, juntamente com um conjunto $E(G)$ chamados arestas. Cada aresta representa um par $\{u, v\}$ onde $u, v \in V(G)$ e $u \neq v$. Se $G = (V, E)$ é um grafo, diremos que u e v são vértices adjacentes se $\{u, v\} \in E$; nesse caso, dizemos que a aresta $\{u, v\}$ incide sobre os vértices u e v. A matriz de adjacência de um grafo é definida da seguinte maneira:

$$a_{ij} = \begin{cases} 1, & \text{se } v_i \text{ e } v_j \text{ são adjacentes;} \\ 0, & \text{caso contrário.} \end{cases}$$

Observemos que tal matriz é uma matriz simétrica com entradas reais cujos elementos da sua diagonal principal são nulos. Dada uma matriz quadrada A de ordem n, um número escalar λ é um autovalor de A quando λ é uma raiz do polinômio

característico, $det(A - \lambda I)$, onde I é a matriz identidade de ordem n. A simetria das matrizes adjacentes desempenha um papel importante para o presente trabalho, pois é conhecido que todos os autovalores de uma matriz real simétrica são reais (ver Lipschutz and Lipson (2011) ou Poole (2004)).

Duas matrizes A e B são ditas congruentes (e denota-se $A \cong B$) se, e somente se, existe uma matriz E invertível tal que $E^T AE = B$. Para matrizes congruentes vale o seguinte resultado, cuja demonstração pode ser obtida em Poole (2004).

Teorema 1. *Toda matriz real simétrica é congruente a uma matriz diagonal.*

Desde que os autovalores de uma matriz diagonal são os elementos de sua diagonal principal, seria desejável estabelecer uma relação entre a matriz de adjacência de um grafo e uma matriz diagonal e com isso isso descobrir a relação entre seus respectivos autovalores.

Seja A uma matriz simétrica. Um modo prático de obtermos matrizes congruentes a A é aplicando uma operação elementar sobre as linhas e em seguida repetindo esta mesma operação nas colunas de A. Isto equivale a multiplicar uma matriz elementar E à esquerda de uma matriz real simétrica A e multiplicar a matriz transposta E^t à direita. Desde que E é inversível, tem-se que EAE^t é uma matriz congruente a matriz A. Realizando operações nas linhas e repetindo-as nas colunas quantas vezes for necessário, podemos obter uma matriz diagonal D que é congruente a matriz A.

É válido salientar que apesar de serem congruentes, as matrizes A e D não terão necessariamente os mesmos autovalores. Entretanto, a relação entre seus autovalores é estabelecida pela Lei da Inércia de Sylvester, cuja prova pode ser encontrada em Lipschutz and Lipson (2011).

Teorema 2. *(Lei da Inércia de Sylvester): Duas matrizes reais, simétricas, de ordem $n \times n$ são congruentes se e somente se elas têm o mesmo número de autovalores positivos, o mesmo número de autovalores negativos e o mesmo número de autovalores nulos.*

Sejam A é a matriz de adjacência de um grafo de ordem n, e x um real fixado. Temos que $A + xI$ é uma matriz real simétrica e portanto, pelo Teorema 1, é congruente a alguma matriz diagonal D. O objetivo nas próximas seções será encontrar um procedimento padrão para obtenção da matriz D somente observando

a geometria do grafo e não buscando informações nas linhas e colunas da matriz em si. Um algoritmo que funciona para qualquer grafo ou até para classes muito grandes de grafos ainda é um problema em aberto, mas já existem algoritmos lineares para algumas classes de grafos (Jacobs et al. (2018) e Jacobs and Trevisan (2011)). Nas próximas seções estudaremos o algoritmo *Diagonalize*, apresentado em Jacobs et al. (2013), que foi projetado para a classe de grafos chamada de *Thresholds*.

Tendo em mãos $A + xI$ e sua matriz congruente D, o próximo teorema estabelece uma relação entre os autovalores da matriz A em relação a uma matriz diagonal congruente D.

Teorema 3. *Sejam $x \in \mathbb{R}$ fixado e D uma matriz diagonal congruente à matriz $A - xI$. Então:*

i. O número de autovalores de A maiores do que x é o número de entradas positivas em D;

ii. O número de autovalores de A menores do que x é o número de entradas negativas em D;

iii. A multiplicidade de um autovalor x é o número de entradas nulas na diagonal de D.

De fato, sejam λ_i, μ_i, $i = 1, 2, ..., n$, autovalores de A e $A - xI$, e v_i for um autovetor associado a μ_i, respectivamente, então

$$\mu_i v_i = (A - xI)v_i = Av - xIv_i = Av_i - xv_i = \lambda_i v_i - xv_i = (\lambda_i - x)v_i,$$

portanto, $\mu_i = \lambda_i - x$. Para (i), note que $\lambda_i > x$ se, e somente se, $\mu_i > 0$. Diante disto, sejam d_i, $i = 1, 2, ..., n$ autovalores da matriz diagonal D congruente a $A - xI$. Do Teorema 2 decorre que o número de autovalores positivos de D é o mesmo número de autovalores positivos de $A - xI$, que é o mesmo número de autovalores de A que são maiores que x. A prova de (ii) é análoga. Para (iii), note que se há j autovalores nulos em D, pelo Teorema 2 haverão j autovalores nulos em $A - xI$ e, consequentemente, j autovalores nulos em A.

Este teorema pode nos auxiliar na localização de autovalores, como podemos ver no seguinte resultado:

Corolário 1. *O número de autovalores de um grafo com matriz de adjacência A num intervalo $(a, b]$, incluindo multiplicidades, é o número de entradas positivas na*

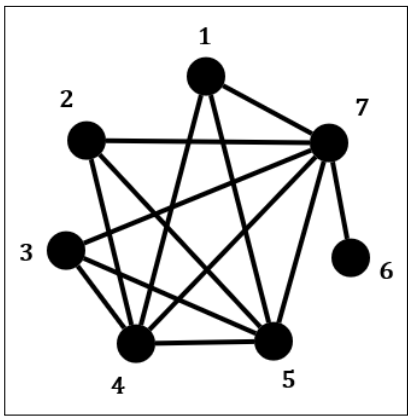
diagonalização de $A - aI$, menos o número de entradas positivas na diagonalização de $A - bI$.

Portanto, a eficácia na determinação do número de autovalores de um grafo dentro do intervalo $(a, b]$ está intrinsecamente relacionada à capacidade de diagonalizar matrizes da forma $A - xI$, para quaisquer valores de x. Nas seções subsequentes, apresentaremos um algoritmo altamente eficiente para grafos *threshold* para essa finalidade.

3 GRAFOS *THRESHOLD*

Existem diversas maneiras de caracterizar um grafo *threshold* (em Tura (2013) podemos encontrar detalhes de algumas dessas caracterizações). A melhor delas para nossos propósitos é por meio de uma sequência binária de modo recursivo. Definimos um grafo *threshold* $G = (V, E)$ de ordem n por meio de uma sequência binária $(b_1, b_2, ..., b_n)$, onde cada $b_i = 0$ representa a adição de um vértice isolado (que não é adjacente a nenhum dos vértices previamente adicionados) e cada $b_i = 1$ representa a adição de um vértice dominante (que é adjacente a todos os vértices que foram previamente adicionados). Para exemplificar, a sequência $(0, 0, 0, 1, 1, 0, 1)$ representa o grafo *threshold* da Figura (1).

Figura 1 – Exemplo de grafo *threshold*



Fonte: Os autores (2023)
Legenda: Grafo *threshold* definido pela sequência binária $(0, 0, 0, 1, 1, 0, 1)$, onde os números que acompanham os nodos indicam a ordem em que foram tomados a partir da sequência binária

A caracterização dos grafos *threshold* através da sequência binária acima definida, estabelece um padrão na matriz de adjacência desses grafos. Se G um grafo *threshold*

de ordem n com sequência binária $(b_1, b_2, ..., b_n)$, e $A = [a_{ij}]$ sua matriz de adjacência, então é fácil ver que se $b_i = 1$ então $a_{ij} = a_{ji} = 1$ para $i < j$, e por outro lado, se $b_i = 0$ então $a_{ij} = a_{ji} = 0$, para $i < j$. Essa relação entre a sequência binária e a matriz de adjacência será essencial para o funcionamento do algoritmo que apresentaremos na próxima seção.

A matriz A a seguir é a matriz de adjacência do grafo *threshold* representado na Figura 1.

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

3.1 Diagonalizando a matriz de adjacência de um grafo *threshold*

O algoritmo de diagonalização da matriz de adjacência de grafos *threshold*, proposto por Jacobs et al. (2013), constrói uma matriz diagonal D congruente a $A + xI$, onde A é a matriz de adjacência do grafo *threshold* G e x é um número real arbitrário.

Tal algoritmo funciona do seguinte modo. Assumindo um grafo $G = (b_1, b_2, \ldots, b_n)$ com matriz de adjacência A de ordem n, o algoritmo executa $n - 1$ passos no total. A cada passo m, $(2 \leq m \leq n)$, conforme os valores de b_m e b_{m-1}, o algoritmo optará por um de três casos possíveis, que ao serem executados, equivalem a fazer operações de escalonamento na matriz (de baixo para cima e da direita para a esquerda), exatamente nas linhas (e também simultaneamente, nas mesmas colunas) m e $m - 1$ da matriz $A - xI$, que ao fim deste passo, todas as entradas da linha e coluna m serão nulas, exceto possivelmente o elemento d_m da diagonal. Executando-se $n-1$ vezes, garantiremos que a matriz D resultante seja diagonal e congruente à matriz $A + xI$.

Cabe salientar que os casos a serem executados em cada passo, foram meticulosamente escolhidos para grafos do tipo *threshold* e dependem diretamente da caracterização binária destes. Outras classes de grafos, mesmo com matrizes de adjacências sendo simétricas, exigiriam outros modos de lidar com esse

escalonamento, e a existência de algoritmo similar a este dependerá muito da caracterização e da geometria destes grafos. Os três casos que podem ocorrer, estão resumidos no algoritmo apresentado na Figura 2 e podem ser vistos com mais detalhes em Jacobs et al. (2013). É importante notar que o Caso 3 não aparece no algoritmo pois caso ocorra, não é necessário nenhuma ação específica, pois estaríamos com a linha (e coluna) m no formato diagonal.

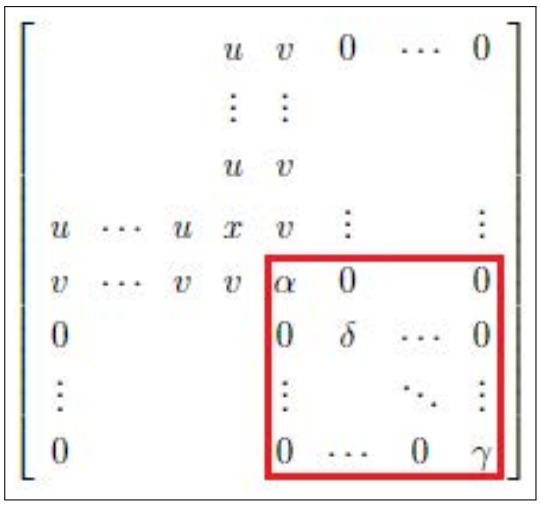
Figura 2 – Algoritmo *Diagonalize*

```
Algorithm Diagonalize(G, x)
    initialize d_i ← x, for all i
    for m = n to 2
        α ← d_m
        if b_{m-1} = 1 and b_m = 1
            if α + x ≠ 2                                  //subcase 1a
                d_{m-1} ← (αx − 1)/(α + x − 2)
                d_m ← α + x − 2
            else if x = 1                                 //subcase 1b
                d_{m-1} ← 1
                d_m ← 0
            else                                          //subcase 1c
                d_{m-1} ← 1
                d_m ← −(1 − x)^2
                b_{m-1} ← 0
        else if b_{m-1} = 0 and b_m = 1
            if x = 0                                      //subcase 2a
                d_{m-1} ← 1
                d_m ← −1
            else                                          //subcase 2b
                d_{m-1} ← α − 1/x
                d_m ← x
                b_{m-1} ← 1
    end loop
```

Fonte: Jacobs et al. (2013)
Legenda: Representação esquemática do algoritmo utilizado para diagonalizar a matriz de adjacência de um grafo *threshold*

A Figura 3 abaixo ilustra a matriz parcialmente diagonalizada após a realização de $n - m$ passos do algoritmo. Observemos que a submatriz em destaque está diagonalizada, então consideramos apenas a submatriz resultante $m \times m$ que continua sendo uma representação matricial de adjacência de um subgrafo *threshold* do grafo *threshold* inicial.

Figura 3 – Diagonalização parcial



Fonte: Jacobs et al. (2013)
Legenda: Diagonalização parcial da matriz de adjacência de um grafo *threshold* após $n - m$ passos do algoritmo

4 IMPLEMENTAÇÃO NA LINGUAGEM *PYTHON*

Inicialmente, é importante observar que a sequência binária que define um grafo *threshold* inicia com zero, já que o primeiro vértice a ser adicionado é sempre isolado. Observe também que, caso o grafo *threshold* seja conexo, a sequência que o representa termina com o número 1, no entanto, se o grafo *threshold* for desconexo, a sequência encerra-se com o número 0.

Considerando que o grafo *threshold* seja conexo, a sequência binária consistirá de blocos alternados de zeros e uns, onde o primeiro bloco será composto por zeros e o último bloco por uns. Para facilitar a entrada de dados do programa, pode-se reescrever a sequência binária na forma $(0^{r_1}, 1^{r_2}, 0^{r_3}, 1^{r_4}, ..., 1^{r_k})$, onde os índices $r_i \geq 1$, com $i = 1, 2, ..., k$, indicam a quantidade de zeros ou de uns de cada bloco.

A entrada de dados no *Python* se dá através da lista a, que é a lista dos índices $r_1, r_2, ..., r_k$, e por meio da função *def*, em que o usuário digita um número real x. Este x é arbitrário e, através do algoritmo é possível saber se ele é ou não um autovalor, e caso positivo, pelo Item (iii) do Teorema 3, calcularmos sua multiplicidade. Após o usuário entrar com os dados, o programa irá gerar a lista b, que é a sequência binária que representa o grafo *threshold*, a partir da lista a. Em seguida, a variável n é definida como a soma dos elementos de a, o que corresponde à ordem da matriz de adjacência.

Após isso, cria-se a lista d com n elementos, onde todos esses elementos são inicializados com o valor x inserido pelo usuário.

Em seguida, são realizados os cálculos do algoritmo, e os valores da lista d vão sendo atualizados. Ao final do processo, a lista d conterá os elementos da diagonal principal da matriz diagonal D congruente à matriz $A + xI$. Esses números não são os autovalores de A, mas são eles que indicam se o número inserido pelo usuário é autovalor ou não. A interpretação dos valores na saída de dados se dá pelo Teorema 3. Ao final do processamento, é solicitado ao usuário se ele deseja continuar testando outros valores. Caso digite "s", o usuário deverá inserir outro número real e o programa executará novamente todos os cálculos para verificar se tal número é ou não um autovalor, para o mesmo grafo. Caso o usuário digite "n", a operação é finalizada.

Como aplicação podemos encontrar quantos autovalores do grafo *threshold* do exemplo 1 estão no intervalo $(-2, 0]$ aplicando o algoritmo nos extremos desse intervalo, como mostra a Figura 4

Figura 4 – Aplicação do algoritmo

```
Digite um número:-2
[-0.24999999999999978, 2.0, 2.0, 2.0, 1.333333333333333, 1.5, 2.0]
Deseja continuar? (s/n): s
Digite um número:0
[-0.0, -0.0, 1, -1, -2.0, 1, -1]
Deseja continuar? (s/n): n
```

Fonte: Os autores (2023)
Legenda: Saída de dados da aplicação do algoritmo *Diagonalize* para o grafo *threshold* da Figura 1 utilizando os valores -2 e 0

Observemos que, quando aplicamos o algoritmo para $x = -2$, obtemos seis números positivos na saída de dados. Pelo Teorema 3, existem seis autovalores do grafo em questão maiores do que -2. Ainda, quando aplicamos o algoritmo para $x = 0$, vemos que há dois valores na saída de dados positivos, ou seja, o grafo em questão possui dois autovalores maiores que o valor inserido (zero). Além disso, há dois valores na saída de dados iguais a zero, o que significa que o grafo em questão possui dois autovalores iguais ao valor inserido (zero). Assim, segue do Corolário que este grafo possui $6 - 2 = 4$ autovalores no intervalo $(-2, 0]$. Ainda, como tal grafo possui dois autovalores iguais a zero, temos que o intervalo aberto $(-2, 0)$ possui $4 - 2 = 2$

autovalores. A figura 5 mostra a implementação completa deste algoritmo, a qual também está disponível no repositório GitHub, podendo ser acessada clicando aqui.

Figura 5 – Algoritmo em *Python*

```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

from sympy import init_printing, pprint
init_printing(use_latex='mathjax')

from sympy import *
```

```
a = [3,2,1,1]
```

```
def diag(x):
  b = []
  alternar = 0
  for valor in a:
    for i in range(valor):
      b.append(alternar)
    alternar = 1 - alternar
  n = sum(a) #ordem da matriz
  d = [x] * n

  for m in range(1,n):
    y = d[-m]
    if b[-m-1] == 1 and b[-m] == 1:    #caso 1
      if y+x != 2:                     #subcaso 1a
        d[-m-1] = (y*x - 1)/(y+x-2)
        d[-m] = y+x-2
      elif x==1:                       #subcaso 1b
        d[-m-1] = 1
        d[-m] = 0
      else:                            #subcaso 1c
        d[-m-1] = 1
        d[-m] = -(1-x)**2
        b[-m-1] = 0
    elif b[-m-1] == 0 and b[-m] == 1:  #caso 2
      if x == 0:                       #subcaso 2a
        d[-m-1] = 1
        d[-m] = -1
      else:                            #subcaso 2b
        d[-m-1] = y - (1/x)
        d[-m] = x
        b[-m-1] = 1
    else:                              #caso 3
      d[-m] = y
      d[-m-1] = x
  return(d)

while True:
  x = float(input("Digite um número:"))
  diag(-x)
  resultado = diag(-x)
  print(resultado)
  continuar = input("Deseja continuar? (s/n): ")
  if continuar.lower() != "s":
    break
```

Fonte: Os autores
Legenda: Algoritmo *Diagonalize* em *Python*

5 CONCLUSÃO

As novas linguagens computacionais proporcionaram um avanço e até um renascimento de algumas áreas de pesquisa em Teoria de Grafos. Nosso trabalho vem nesta direção com a implementação de um algoritmo que localiza autovalores de grafos pertencentes a uma classe bastante utilizada em resoluções de problemas de diversas áreas do conhecimento. Com esta localização é possível, por exemplo, detectar em que setores da reta não devem esperar respostas e quais haverão concentração de autovalores.

É fundamental ressaltar que existe um algoritmo para uma classe mais ampla, chamada classe dos cografos, cuja implementação exige uma entrada de dados altamente artificial, através de coárvores isomorfas que são uma espécie de união e junção de grafos *threshold*, o que dificulta a compreensão e manuseio de dados gerados. Em contraste, o *Diagonalize* se destaca pela simplicidade da entrada de dados e por ser de ordem linear, fornecendo rapidamente os resultados esperados. O algoritmo se baseia nas operações elementares utilizadas em escalonamento de matrizes e na geometria do grafo *threshold* de entrada.

REFERÊNCIAS

Jacobs, D. P. & Trevisan, V. (2011). Locating the eigenvalues of trees. *Linear Algebra and its Applications*, 434(1):81–88. doi: 10.1016/j.laa.2010.08.006.

Jacobs, D. P., Trevisan, V., & Tura, F. (2013). Eigenvalue location in threshold graphs. *Linear Algebra and its applications*, 439(10):2762–2773. Recovered from: https://www.sciencedirect.com/science/article/pii/S002437951300493X.

Jacobs, D. P., Trevisan, V., & Tura, F. C. (2018). Eigenvalue location in cographs. *Discrete Applied Mathematics*, 245:220–235. doi: https://doi.org/10.1016/j.dam.2017.02.007.

Lipschutz, S. & Lipson, M. L. (2011). *Álgebra linear*. (2nd ed.). Bookman.

Mahadev, N. V. & Peled, U. N. (1995). *Threshold graphs and related topics*. Elsevier.

Poole, D. (2004). *Álgebra linear*. (56th ed.). Cengage Learning.

Tura, F. C. (2013). *O espectro de grafos threshold e aplicações* (Tese submetida ao Programa de Pós-Graduação em Matemática Aplicada). Instituto de Matemática, Universidade Federal do Rio Grande do Sul, Porto Alegre, RS, Brasil.

Contribuições dos autores

1 – Andressa de Oliveira Eckhardt
Bacharela em Matemática e mestranda em Matemática
https://orcid.org/0009-0002-0597-6179 • andressa.o.eckhardt@gmail.com
Contribuição: Redação, edição & implementação do algoritmo no Python

2 – João Roberto Lazzarin
Doutor em Matemática e professor
https://orcid.org/0000-0001-9527-0430 • joao.lazzarin@ufsm.br
Contribuição: Implementação do algoritmo no Python, revisão

3 – Fernando Colman Tura
Doutor em Matemática e professor
https://orcid.org/0000-0001-5423-7191 • fernando.tura@ufsm.br
Contribuição: Revisão

Como citar este artigo

Eckhardt, A. de O., Lazzarin, J. R., & Tura, F. C. (2025). Uma implementação do algoritmo de localização de autovalores de grafos *Threshold* em *Python*. *Ciência e Natura*, Santa Maria, v. 47, esp. 1, e90148. DOI: https://doi.org/10.5902/2179460X90148.