ĐẠI HỌC HOA SEN

Tạp chí Khoa học Trường Đại học Hoa Sen
https://www.hoasen.edu.vn/qlkh/ tapchidhhs.vn

HÀM GIẢ ĐẢO GIẢI HỆ PHƯƠNG TRÌNH TUYẾN TÍNH BẰNG LẬP TRÌNH PYTHON VÀ ỨNG DỤNG

PGS.TS. Nguyễn Văn Lộc*
Khoa Công Nghệ Thông Tin, Trường Đại học Hoa Sen

Thông tin bài báo

Nhận bài: 12/2024
Chấp nhận: 02/2025
Xuất bản online: 03/2025

TÓM TẮT

Sử dụng hàm giả đảo trong lập trình Python có thể giải các dạng hệ phương trình tuyến tính Cramer; Hệ phương trình tuyến tính có số ẩn nhiều hơn số phương trình; Hệ phương trình tuyến tính có số phương trình nhiều hơn số ẩn; Hệ phương trình tuyến tính có số phương trình bằng số ẩn nhưng định thức ma trận hệ số bằng 0 và một số ứng dụng trong thực tế.

Cramer's linear equations can be solved in Python by using pseudo-inverse functions; There are more unknowns than equations in a system of linear equations; There are more equations in linear equation systems than unknowns. Despite having the same number of equations as the unknowns, the linear equation system has certain useful applications and a coefficient matrix determinant of 0.

Keywords: *Hàm giả đảo; hệ phương trình tuyến tính; hệ Cramer; Lập trình Python; Ứng dụng*

1. GIỚI THIỆU

Bài báo trình bày cách sử dụng hàm giả đảo giải các dạng hệ phương trình tuyến tính và ứng dụng của hàm giả đảo trong thực tế.

2. TỔNG QUAN NGHIÊN CỨU

Khi giải hệ phương trình tuyến tính Cramer bằng phương pháp Toán học ngoài phương pháp Gauss, ta có thể dùng phương pháp định thức (phương pháp Cramer) và phương pháp ma trận nghịch đảo. Tuy nhiên, khi giải hệ phương trình tuyến tính tổng quát (khác hệ Cramer), chỉ sử dụng được duy nhất phương pháp Gauss bởi vì, với hệ phương trình tuyến tính tổng quát không tồn tại ma trận nghịch đảo, để giải quyết khó khăn này, các nhà Toán học đã tìm cách tổng quát hóa ma trận nghịch đảo, để được ma trận giả nghịch đảo. Loại ma trận giả nghịch đảo phổ biến nhất là ma giả nghịch đảo Moore–Penrose, còn gọi là ma trận nghịch đảo tổng quát, được tìm ra một cách độc lập bởi E. H. Moore năm 1920, Arne Bjerhammar năm 1951 và Roger Penrose năm 1955. Trong lập trình Python, ma trận giả nghịch đảo (gọi tắt là ma trận giả đảo) xác định bởi hàm pinv ( ) hoặc hàm linalg.pinv ( ). Trong bài báo này, chúng ta trình bày ứng dụng của hàm giả đảo trong giải hệ phương trình tuyến tính và trong thực tế.

* Tác giả liên hệ:
Email: *loc.nguyenvan@hoasen.edu.vn*

3. PHƯƠNG PHÁP NGHIÊN CỨU VÀ DỮ LIỆU

Ma trận giả đảo (Pseudoinverse hay Moore –Penrose Inverse)

Pseudoinverse là tổng quát hóa của ma trận nghịch đảo: thay vì tìm ma trận nghịch đảo của ma trận vuông, ta tìm ma trận giả đảo của ma trận có số lượng hàng và cột tùy ý (có thể bằng nhau hoặc không bằng nhau)

Công thức: $A^+ = V.D^+.U^T$

Trong đó A^+ là pseudoinverse, D^+ là pseudoinverse của ma trận đường chéo Sigma và là U^T hoán vị của U với D = 1/Sigma.

Điều đáng chú ý là có thể thay Inverse bởi Pseudoinverse trong phương trình ma trận vuông thì kết quả bài toán không thay đổi, đối với ma trận chữ nhật thì không tồn tại Inverse nhưng sử dụng Pseudoinverse ta tìm được kết quả của bài toán.

Ví dụ: Tìm Pseudoinverse của ma trận: $A = \begin{bmatrix} 2 & 4 & -1 & 5 & -2 \\ -4 & -5 & 3 & -8 & 1 \\ 2 & -5 & -4 & 1 & 8 \\ -6 & 0 & 7 & -3 & 1 \end{bmatrix}$ [5, tr 128].

Giải

Sử dụng gói numpy và hàm pinv()

In[1]:	from numpy import array from numpy.linalg import pinv
In[2]:	A = array([[2, 4, -1, 5, -2], [-4, -5, 3, -8, 1], [2, -5,-4,1, 8], [-6, 0, 7, -3, 1]])
In[3]:	B =pinv(A) print(B)
Out[3]:	[[-1.91011236e+00 -1.23595506e+00 -3.23595506e-01 4.49438202e-03] [-5.16853933e+00 -3.43258427e+00 -8.93258427e-01 2.41573034e-01] [5.05617978e-01 2.97752809e-01 7.97752809e-02 7.52808989e-02] [4.00000000e+00 2.50000000e+00 7.00000000e-01 -1.00000000e-01] [-3.00000000e+00 -2.00000000e+00 -4.00000000e-01 2.00000000e-01]]

Ý nghĩa của hàm pinv

- **Pseudo-inverse** là một dạng nghịch đảo tổng quát cho các ma trận không vuông hoặc suy biến (singular), khi mà nghịch đảo thông thường (inv()) không thể tính được.
- Nó giúp giải các hệ phương trình tuyến tính dạng **Ax = b** ngay cả khi A không phải là ma trận vuông hoặc có định thức bằng 0.

Công dụng của hàm pinv

- **Giải hệ phương trình tuyến tính:** Khi A không khả nghịch, có thể sử dụng pinv(A) để tìm nghiệm gần đúng tối ưu.
- **Phân tích dữ liệu & Machine Learning:**
  *Trong hồi quy tuyến tính **OLS (Ordinary Least Squares)**, pseudo-inverse giúp tìm nghiệm bình phương tối thiểu khi hệ phương trình có nhiều hơn một nghiệm hoặc không có nghiệm chính xác.
- Trong mô hình **PCA (Principal Component Analysis)**, nó hỗ trợ trong xử lý ma trận dữ liệu.
- **Xử lý ảnh & Giảm nhiễu:** Giúp khôi phục hình ảnh hoặc xử lý dữ liệu khi thông tin bị mất hoặc suy biến.

4. KẾT QUẢ NGHIÊN CỨU

4.1. Phương pháp hàm giả đảo giải hệ phương trình tuyến tính

Cho hệ phương trình tuyến tính (1): $\begin{cases} a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n = b_1 \\ ..... \\ a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n = b_m \end{cases} ..(1)$

Chuyển hệ phương trình tuyến tính đã cho về dạng ma trận : AX = b (2), với

$$A = \begin{bmatrix} a_{11} & a_{12} & ... & a_{1n} \\ a_{21} & a_{22} & ... & a_{2n} \\ ... & ... & ... & ... \\ a_{m1} & a_{m2} & ... & a_{mn} \end{bmatrix}; b = \begin{bmatrix} b_1 \\ b_2 \\ ... \\ b_m \end{bmatrix}; X = \begin{bmatrix} x_1 \\ x_2 \\ ... \\ x_n \end{bmatrix}$$

Phương pháp (Sử dụng gói numpy và hàm giả đảo pinv( ) hoặc hàm linalg.pinv())

Bước 1: Lập ma trận A với các hệ số a_{ij} và vector b với các hệ số b_j, bằng cách sử dụng np.array([[ ],..,[ ]]) hay array([[ ],..,[ ]])

Bước 2: Tính ma trận giả đảo của A, sử dụng hàm pinv(A) hoặc hàm linalg.pinv()

Bước 3: Tìm vector nghiệm X bằng việc nhân ma trận giả đảo với vector b sử dụng

X = pinv(A).dot(b) hoặc X = np.linalg.pinv(A).dot(b)

4.2. Phương pháp hàm giả đảo giải hệ phương trình tuyến tính Cramer

Ví dụ: Giải hệ phương trình

$$\begin{cases} x_1 + x_2 + x_3 + x_4 + x_5 = 15 \\ x_1 + 2x_2 + 3x_3 + 4x_4 + 5x_5 = 35 \\ x_1 + 3x_2 + 6x_3 + 10x_4 + 15x_5 = 70 \\ x_1 + 4x_2 + 10x_3 + 20x_4 + 35x_5 = 126 \\ x_1 + 5x_2 + 15x_3 + 35x_4 + 70x_5 = 210 \end{cases}$$. [4, tr 84]

Giải

Phương pháp toán học (Phương pháp Cramer)

$$D = \begin{vmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 3 & 6 & 10 & 15 \\ 1 & 4 & 10 & 20 & 35 \\ 1 & 5 & 15 & 35 & 70 \end{vmatrix} = 1; D_{x_1} = \begin{vmatrix} 15 & 1 & 1 & 1 & 1 \\ 35 & 2 & 3 & 4 & 5 \\ 70 & 3 & 6 & 10 & 15 \\ 126 & 4 & 10 & 20 & 35 \\ 210 & 5 & 15 & 35 & 70 \end{vmatrix} = 5; D_{x_2} = \begin{vmatrix} 1 & 15 & 1 & 1 & 1 \\ 1 & 35 & 3 & 4 & 5 \\ 1 & 70 & 6 & 10 & 15 \\ 1 & 126 & 10 & 20 & 35 \\ 1 & 210 & 15 & 35 & 70 \end{vmatrix} = 4;$$

$$D_{x_3} = \begin{vmatrix} 1 & 1 & 15 & 1 & 1 \\ 1 & 2 & 35 & 4 & 5 \\ 1 & 3 & 70 & 10 & 15 \\ 1 & 4 & 126 & 20 & 35 \\ 1 & 5 & 210 & 35 & 70 \end{vmatrix} = 3; D_{x_4} = \begin{vmatrix} 1 & 1 & 1 & 15 & 1 \\ 1 & 2 & 3 & 35 & 5 \\ 1 & 3 & 6 & 70 & 15 \\ 1 & 4 & 10 & 126 & 35 \\ 1 & 5 & 15 & 210 & 70 \end{vmatrix} = 2; D_{x_4} = \begin{vmatrix} 1 & 1 & 1 & 1 & 15 \\ 1 & 2 & 3 & 4 & 35 \\ 1 & 3 & 6 & 10 & 70 \\ 1 & 4 & 10 & 20 & 126 \\ 1 & 5 & 15 & 35 & 210 \end{vmatrix} = 1;$$

$$\Rightarrow x_1 = \frac{D_{x_1}}{D} = \frac{5}{1} = 5; x_2 = \frac{D_{x_2}}{D} = \frac{4}{1} = 4; x_3 = \frac{D_{x_3}}{D} = \frac{3}{1} = 3; x_4 = \frac{D_{x_4}}{D} = \frac{2}{1} = 2; x_5 = \frac{D_{x_5}}{D} = \frac{1}{1} = 1$$

$$\Rightarrow \{x_1 = 5; x_2 = 4; x_3 = 3; x_4 = 2; x_5 = 1\}$$

Phương pháp lập trình (Hàm nghịch đảo)

Cách 1: (Sử dụng gói numpy và hàm inv( ))

In[1]:	from numpy import array from scipy import linalg
In2]:	A = array([[1, 1, 1, 1, 1], [1, 2, 3, 4, 5], [1, 3, 6, 10, 15], [1, 4, 10, 20, 35], [1, 5, 15, 35, 70]]) print(A)
Out [2]:	[[1 1 1 1 1] [1 2 3 4 5] [1 3 6 10 15] [1 4 10 20 35] [1 5 15 35 70]]
In[3]:	import numpy as np B= np.array([[15],[35],[70],[126], [210]]) B
Out[3]:	array([[15], [35], [70], [126], [210]])
In[4]:	from numpy.linalg import inv K = inv(A) K
Out[4]:	array([[5., -10., 10., -5., 1.], [-10., 30., -35., 19., -4.], [10., -35., 46., -27., 6.], [-5., 19., -27., 17., -4.], [1., -4., 6., -4., 1.]])
In[5]:	X = K.dot(B) X
Out [5]:	array([[5.], [4.], [3.], [2.], [1.]])

Ý nghĩa của SciPy

- Là một thư viện mở rộng của numpy, cung cấp các hàm nâng cao để xử lý toán học, khoa học dữ liệu, và kỹ thuật.
- Hỗ trợ các phép tính tối ưu hóa, tích phân, nội suy, thống kê, xử lý tín hiệu, đại số tuyến tính.

Công dụng chính của SciPy

- Hỗ trợ các phép toán ma trận như nghịch đảo, pseudo-inverse, giá trị riêng, SVD, phân rã Cholesky,.trong đại số tuyến tính.
- Hỗ trợ tính toán phân phối xác suất, kiểm định giả thuyết, hồi quy, mô phỏng Monte Carlo, trong xác suất thống kê.
- Dùng để tìm giá trị nhỏ nhất hoặc lớn nhất của hàm, giải phương trình phi tuyến. trong tối ưu hóa.

- Hỗ trợ tính tích phân xác định và giải phương trình vi phân.
- Hỗ trợ lọc tín hiệu, biến đổi Fourier, phân tích hệ thống động học.
- Dùng để tạo các mô hình nội suy từ dữ liệu rời rạc.

Nghiệm của hệ phương trình là: $\{x_1 = 5; x_2 = 4; x_3 = 3; x_4 = 2; x_5 = 1\}$

Cách 2: Sử dụng gói numpy và hàm np.linalg.inv ()

In[1]:	import numpy as np from scipy import linalg
In2]:	A = np.matrix([[1, 1, 1, 1, 1], [1, 2, 3, 4, 5], [1, 3, 6, 10, 15], [1, 4, 10, 20, 35], [1, 5, 15, 35, 70]]) print(A)
Out [2]:	[[1 1 1 1 1] [1 2 3 4 5] [1 3 6 10 15] [1 4 10 20 35] [1 5 15 35 70]]
In[3]:	A_nghichdao = np.linalg.inv(A) A_nghichdao
Out[3]:	matrix([[5., -10., 10., -5., 1.], [-10., 30., -35., 19., -4.], [10., -35., 46., -27., 6.], [-5., 19., -27., 17., -4.], [1., -4., 6., -4., 1.]])
In[4]:	B = np.matrix([[15],[35],[70],[126], [210]]) print(B)
Out[4]:	[[15 35 70 126 210]]
In[5]:	c = B.T c
Out [5]:	matrix([[15], [35], [70], [126], [210]])
In[6]:	X = A_nghichdao.dot(c) X
Out [6]:	matrix([[5.], [4.], [3.], [2.], [1.]])

Ý nghĩa của hàm linalg.pinv()

- **Nghịch đảo giả Moore-Penrose** là một khái niệm mở rộng của nghịch đảo ma trận thông thường, áp dụng cho cả ma trận không vuông và ma trận suy biến.
- Được sử dụng để giải hệ phương trình tuyến tính dạng Ax=bAx = bAx=b trong trường hợp ma trận A không khả nghịch hoặc không vuông.
- Được tính dựa trên **phân rã giá trị suy biến (SVD - Singular Value Decomposition)**.

Công dụng chính của hàm linalg.pinv()

- **Giải hệ phương trình tuyến tính:** Dùng pinv() để tìm nghiệm gần đúng khi ma trận A không khả nghịch. **Ứng dụng:** Trong Machine Learning, hệ phương trình tuyến tính kiểu này xuất hiện trong bài toán hồi quy tuyến tính OLS. Ứng dụng: Trong Machine Learning và thống kê, đặc biệt khi dữ liệu có đa cộng tuyến.
- **Giải quyết vấn đề suy biến** (Singular Matrix).Nếu sử dụng numpy.linalg.inv() cho ma trận suy biến, nó sẽ báo lỗi. pinv() giúp khắc phục vấn đề này.
- **So sánh scipy.linalg.pinv() và numpy.linalg.pinv().**
  *scipy.linalg.pinv(): Linh hoạt hơn, hiệu suất cao hơn, hỗ trợ nhiều thuật toán tối ưu.
  *numpy.linalg.pinv(): Chỉ dựa trên SVD, phù hợp cho tính toán đơn giản.

Nghiệm của hệ phương trình là: $\{x_1 = 5; x_2 = 4; x_3 = 3; x_4 = 2; x_5 = 1\}$

Luyện tập

Giải hệ phương trình :

$$\begin{cases} x_1 + 2x_2 + 3x_3 - 2x_4 = 6 \\ 2x_1 - x_2 - 2x_3 - 3x_4 = 8 \\ 3x_1 + 2x_2 - x_3 + 2x_4 = 4 \\ 2x_1 - 3x_2 + 2x_3 = x_4 = -8 \end{cases}$$ [3 , tr 86]

Đáp số: Nghiệm của hệ phương trình là: $\{x_1 = 1; x_2 = 2; x_3 = -1; x_4 = -2\}$

4.3. Phương pháp hàm giả đảo giải hệ phương trình tuyến tính có số ẩn nhiều hơn số phương trình.

Ví dụ: Giải hệ phương trình

$$\begin{cases} 24x_1 + 14x_2 + 30x_3 + 40x_4 + 41x_5 = 28 \\ 36x_1 + 21x_2 + 45x_3 + 61x_4 + 62x_5 = 43 \\ 48x_1 + 28x_2 + 60x_3 + 82x_4 + 83x_5 = 58 \\ 60x_1 + 35x_2 + 75x_3 + 99x_4 + 102x_5 = 69 \end{cases}$$. [2, tr 102]

Giải

Cách 1: Sử dụng gói numpy và hàm pinv( )

In[1]:	from numpy import array
In2]:	A = array([[24, 14, 30, 40, 41], [36, 21, 45, 61, 62], [48, 28, 60, 82, 83], [60, 35, 75, 99,102]])
In[3]:	import numpy as np b = np.array([[28],[43],[58],[69]])

In[4]:	from numpy.linalg import pinv K = pinv(A)
In[5]:	X = K.dot(b) X
Out [5]:	array([[-0.24180166], [-0.14105097], [-0.30225207], [0.88463058], [0.23073884]])

Cách 2: Sử dụng gói numpy và hàm linalg.pinv( )

In[1]:	from numpy import array from scipy import linalg
In2]:	A = np.array([[24, 14, 30, 40, 41], [36, 21, 45, 61, 62], [48, 28, 60, 82, 83], [60, 35, 75, 99,102]])
In[3]:	A_giadao = np.linalg.pinv(A)
In[4]:	import numpy as np b = np.array([[28],[43],[58],[69]])
In[5]:	X = A_giadao.dot(b) X
Out [5]:	array([[-0.24180166], [-0.14105097], [-0.30225207], [0.88463058], [0.23073884]])

Nghiệm của hệ phương trình là :

$$\{x_1 = -0.24180166, x_2 = -0.14105097, x_3 = -0.30225207, x_4 = 0.88463058, x_5 = 0.23073884\}$$

Luyện tập

Giải hệ phương trình:

$$\begin{cases} 12x_1 + 14x_2 - 15x_3 + 23x_4 + 27x_5 = 5 \\ 16x_1 + 18x_2 - 22x_3 + 29x_4 + 37x_5 = 8 \\ 18x_1 + 20x_2 - 21x_3 + 32x_4 + 41x_5 = 9 \\ 10x_1 + 12x_2 - 16x_3 + 20x_4 + 23x_5 = 4 \end{cases}$$. [1, tr 102]

Đáp số: Nghiệm tổng quát của phương trình:

$$x_1 = x_4 - \frac{53}{18}x_5 + \frac{20}{9}, x_2 = -\frac{5}{2}x_4 + \frac{5}{6}x_5 - \frac{5}{3}, x_3 = \frac{2}{9}x_5 - \frac{1}{9};$$

Nghiệm riêng của hệ phương trình là:

$$\{x_1 = 0.15688271, x_2 = -0.11224087, x_3 = 0.01174584, x_4 = -0.43748489, x_5 = 0.55285629\}$$

4.4. Phương pháp hàm giả đảo giải hệ phương trình tuyến tính có số ẩn ít hơn số phương trình.

Ví dụ: Giải hệ phương trình

$$\begin{cases} 3x_1 - x_2 + x_3 = 6 \\ x_1 - 5x_2 + x_3 = 12 \\ 2x_1 + 4x_2 = -6 \\ 2x_1 + x_2 + 3x_3 = 3 \\ 5x_1 + 4x_3 = 9 \end{cases}$$. [6 , tr 83]

Giải

Cách 1: Sử dụng gói numpy và hàm pinv()

In[1]:	from numpy import array
In2]:	A = array([[3, -1, 1], [1,-5, 1], [2, 4, 0], [2, 1, 3], [5, 0, 4]])
In[3]:	from numpy.linalg import pinv K = pinv(A)
In[4]:	import numpy as np b = np.array([6,12,-6,3,9])
In[5]:	X = K.dot(b) X
Out [5]:	array([1., -2., 1.])

Cách 2: Sử dụng gói numpy và hàm linalg.pinv()

In[1]:	from numpy import array from scipy import linalg
In2]:	A = np.array([[3, -1, 1], [1,-5, 1], [2, 4, 0], [2, 1, 3], [5, 0, 4]])
In[3]:	A_giadao = np.linalg.pinv(A)
In[4]:	import numpy as np b = np.array([[6],[12],[-6],[3], [9]])
In[5]:	X = A_giadao.dot(b) X
Out [5]:	array([[1.], [-2.], [1.]])

Nghiệm của hệ phương trình là: $\{x_1 = 1, x_2 = -2, x_3 = 1\}$

Luyện tập

Giải hệ phương trình (Số phương trình lớn hơn số ẩn)

$$\begin{cases} 8x+6y+5z+2t=21 \\ 3x+3y+2z+t=10 \\ 4x+2y+3z+t=8 \\ 3x+5y+z+t=15 \\ 7x+4y+5z+2t=18 \end{cases}$$. [7, tr 100]

Đáp số: Nghiệm của hệ phương trình là: $\{x=3; y=0; z=-5; t=11\}$

4.5. Phương pháp hàm giả đảo giải hệ phương trình tuyến tính có số phương trình bằng số ẩn, nhưng định thức ma trận hệ số bằng 0.

Ví dụ: Giải hệ phương trình

$$\begin{cases} 2x_1-x_2+3x_3+4x_4=5 \\ 4x_1-2x_2+5x_3+6x_4=7 \\ 6x_1-3x_2+7x_3+8x_4=9 \\ x_1-4x_2+9x_3+10x_4=11 \end{cases}$$. [8 , tr 103]

Giải

Cách 1: Sử dụng gói numpy và hàm pinv()

In[1]:	from numpy import array import numpy as np from scipy import linalg
In2]:	A = array([[2,-1, 3, 4], [4, -2, 5, 6], [6, -3, 7, 8], [1, -4, 9, 10]])
In[3]:	detA = linalg.det(A) detA
Out[3]:	0.0
In[4]:	b = np.array([[5], [7], [9], [11]])
In[5]:	from numpy.linalg import pinv K = pinv(A)
In[6]:	X = K.dot(b) X
Out[6]:	array([[-0.55172414], [0.27586207], [0.37931034], [1.31034483]])

Cách 2: Sử dụng gói numpy và hàm linalg. pinv()

In[1]:	from numpy import array import numpy as np from scipy import linalg
In2]:	A = np.array([[2,-1, 3, 4], [4, -2, 5, 6], [6, -3, 7, 8], [8, -4, 9, 10]])
In[3]:	A_giadao = np.linalg.pinv(A)
In[4]:	b = np.array([[5],[7],[9],[11]])
In[5]:	X = A_giadao.dot(b) X
Out5]:	array([[-0.55172414], [0.27586207], [0.37931034], [1.31034483]])

Đáp số: Nghiệm của hệ phương trình là :{-0.55172414, 0.27586207, 0.37931034, 1.31034483}

Luyện tập

Giải hệ phương trình:

$$\begin{cases} 2x_1+3x_2+x_3+2x_4=3 \\ 4x_1+6x_2+3x_3+4x_4=5 \\ 6x_1+9x_2+5x_3+6x_4=7 \\ 8x_1+12x_2+7x_3+2x_4=9 \end{cases}$$ [4, tr 103]

Đáp số: Nghiệm của hệ phương trình là: $\{x_1=0.61538462, x_2=0.92307692, x_3=-1, x_4=0\}$

4.6. Một số ứng dụng thực tế của hàm giả đảo.

Hàm giả nghịch đảo (inverse function) trong thực tế được áp dụng nhiều trong các bài toán thống kê, tối ưu hóa, mô phỏng Monte Carlo và giải phương trình phi tuyến. Dưới đây là một số ứng dụng thực tế và cách sử dụng Numpy để giải chúng.

Ứng dụng tối ưu hóa: Giải các bài toán tối ưu với các ràng buộc phi tuyến sử dụng hàm giả nghịch đảo.

Bài toán: Tìm giá trị tối ưu của $f(x)=x^2+2e^{-x}, x>0$

Giải pháp sử dụng Gradient hoặc phương pháp xấp xỉ nghiệm

Code Python

	import numpy as np
	from scipy.optimize import minimize
	# Hàm cần tối ưu
	def objective(x):
	  return x**2 + 2 * np.exp(-x)
	# Ràng buộc x > 0
	constraints = {'type': 'ineq', 'fun': lambda x: x}
	# Khởi tạo giá trị ban đầu
	x0 = np.array([1.0])
	# Tối ưu hóa
	result = minimize(objective, x0, constraints=constraints)
	print("Giá trị tối ưu của x là:", result.x[0])
	print("Giá trị tối ưu của hàm là:", result.fun)

	Giá trị tối ưu của x là: 0.5671481834272308 Giá trị tối ưu của hàm là: 1.455938092713924

Ứng dụng trong xử lý tín hiệu Dùng hàm giả đảo để chuẩn hóa hoặc điều chỉnh cường độ tín hiệu.

Bài toán Chuyển đổi tín hiệu đầu vào $y = x^2$ về tín hiệu gốc x

Giải pháp Dùng hàm giả đảo: $x = \sqrt{y}$

	import numpy as np
	# Dữ liệu tín hiệu
	y_signal = np.array([1, 4, 9, 16, 25])
	# Sử dụng hàm giả nghịch đảo
	x_signal = np.sqrt(y_signal)
	print("Tín hiệu gốc là:", x_signal)

	Tín hiệu gốc là: [1. 2. 3. 4. 5.]

5. KẾT LUẬN

Việc đưa hàm giả đảo vào giảng dạy trong chương trình Đại số tuyến tính ở các trường đại học không chỉ giúp cho sinh viên biết các ứng dụng đa dạng của hàm giả đảo trong giải các dạng hệ phương trình tuyến tính tổng quát và vận dụng giải các mô hình tuyến tính trong phân tích kinh tế mà còn giúp cho sự phát triển ở sinh viên các thao tác tư duy như: khái quát hóa, đặc biệt hóa tạo cơ sở cho sự hình thành và phát triển tư duy sáng tạo, một loại hình tư duy quan trọng trong hoạt động thực tiễn của người lao động.

TÀI LIỆU THAM KHẢO

[1] Lê Sĩ Đồng (2010)Toán cao cấp. Đại số tuyến tính. NXB Giáo dục Việt Nam.

[2] Nguyễn Tiến Quang (2014). Cơ sở Đại số tuyến tính.NXB Giáo dục Việt Nam.

[3] Nguyễn Đình Trí (2014). Bài tập Toán cao cấp tập một. NXB. Giáo dục Việt Nam.

[4] I.B. Prockuriakob (1978). Tuyển tập các bài toán Đại số tuyến tính .NXB. " Khoa học" (Tiếng Nga).

[5] David C.Lay-Steven R.Lay-Judi J.McDonald (2016). Linear Algebra and its Applications. Pearson.

[6] Howard Anton (2010). Elementary Linear Algebra. Application Version.

[7] Kevin Sheppard (2019). Introduction To Python For Econometrics, Statistics And Data Analysis. University Of Oxford .

[8] AStephen Boyd (2018). Introduction to Applied Linear Algebra. Cambridge University Press.