Implementación de algoritmos de álgebra lineal numérica

Rodolfo M. Turpo R.

12 de diciembre de 2025

Índice

1. Matriz de Householder

La importancia de las matrices de Householder es que pueden crear ceros en las columnas de una matriz por debajo de la entrada de la diagonal principal de A.

Definición 1.1

Una matriz de la forma

$$H = I - \frac{2uu^\top}{u^\top u}$$

donde u es un vector no nulo, es llamada **matriz de Householder**.

Lema 1.1

Dado un vector no nulo $x \neq e_1$, existe una matriz de Householder H tal que Hx es múltiplo de e_1

Demostración:
Definamos $H = I - \frac{2uu^\top}{u^\top u}$ con $u = x + sign(x_1)\|x\|_2 e_1$. Entonces, multiplicando, se puede ver que Hx es un múltiplo de e_1 ∎

Algorithm 1: Producto de un vector con una matriz de Householder (Hx)

Entrada: Vector $u \in \mathbb{R}^n$ que define la matriz de Householder H, vector $x \in \mathbb{R}^n$.
Salida : Vector x sobrescrito con el producto Hx.

// Paso 1: Calcular el factor de escala
$\beta \leftarrow \frac{2}{u^\top u}$
// Paso 2: Calcular el producto escalar $s = u^\top x$
$s \leftarrow \sum_{i=1}^n u_i x_i$
// Paso 3: Actualizar β con el producto escalar
$\beta \leftarrow \beta \cdot s$
// Paso 4: Actualizar el vector x
for $i = 1$ **to** n **do**
  $x_i \leftarrow x_i - \beta u_i$

A continuación se presenta la implementación del algoritmo:

```python
import numpy as np
import pandas as pd
np.set_printoptions(formatter={'float': lambda x: f"{x:.4e}"})
def HouseHolder(x,n):
  if np.sign(x[[0]])==0:
    u=x+np.linalg.norm(x,2)*np.eye(n)[:,[0]]
    H=np.eye(n)-2*u@u.T/(u.T@u)
  else:
    u=x+np.sign(x[[0]])*np.linalg.norm(x,2)*np.eye(n)[:,[0]]
    H=np.eye(n)-2*u@u.T/(u.T@u)
  return H
if __name__ == "__main__":
    # Un ejemplo pequeno para probar que el script funciona solo
    x=np.array([[-1],[0],[3],[-2]])
    n=4
    H=HouseHolder(x,n)
    print('x\n',x)
    print('------------')
    print('H\n',H)
    print('------------')
    print('H*x\n',H@x)
```

2. Factorización QR

A continuación se presenta la implementación del algoritmo:

```python
import numpy as np
import pandas as pd
np.set_printoptions(formatter={'float': lambda x: f"{x:.4e}"})

def HouseHolder(x,n):
  if np.sign(x[[0]])==0:
    u=x+np.linalg.norm(x,2)*np.eye(n)[:,[0]]
    H=np.eye(n)-2*u@u.T/(u.T@u)
  else:
    u=x+np.sign(x[[0]])*np.linalg.norm(x,2)*np.eye(n)[:,[0]]
    H=np.eye(n)-2*u@u.T/(u.T@u)
  return H
def fact_QR(A):
  n=A.shape[0]
  Q=np.eye(n)
  for i in range(n-1):
    x=A[i:,[i]]
    H0=HouseHolder(x,n-i)
    H=np.eye(n)
    H[i:,i:]=H0
    Q=Q@H
  R=Q.T@A
  return Q,R

if __name__ == "__main__":
    # Un ejemplo pequeno para probar que el script funciona solo
    A=np.array([[-1,4,2,0],[0,3,1,-2],[3,1,5,1],[-2,0,1,4]])
    [Q,R]=fact_QR(A)
    print('Matriz original\n',A)
    print('Q\n',Q)
    print('----------------')
    print('R\n',R)
    print('----------------')
    print('Matriz Q*R\n',Q@R)
```

3. Factorización de cholesky

Para

$$A = HH^\top$$

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} h_{11} & 0 & \cdots & 0 \\ h_{21} & h_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ h_{n1} & h_{n2} & \cdots & h_{nn} \end{bmatrix} \begin{bmatrix} h_{11} & h_{21} & \cdots & h_{n1} \\ 0 & h_{22} & \cdots & h_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & h_{nn} \end{bmatrix}$$

Tenemos:

- $h_{11} = \sqrt{a_{11}}$
- $h_{i1} = \frac{a_{i1}}{h_{11}}, \quad i = 2, 3, \ldots, n$
- $\sum_{k=1}^{i} h_{ik}^2 = a_{ii}$
- $a_{ij} = \sum_{k=1}^{j} h_{ik}h_{kj}, \quad j < i$

Algorithm 2: Algoritmo de Factorización de Cholesky

Entrada: Una matriz $A \in \mathbb{R}^{n\times n}$ simétrica definida positiva.
Salida : Factor de Cholesky H (matriz triangular superior).

```
// El algoritmo construye H fila por fila y sobrescribe la parte superior de A
```

for $k = 1, 2, \ldots, n$ **do**
  for $i = 1, 2, \ldots, k-1$ **do**
    $a_{ki} \equiv h_{ki} = \frac{1}{h_{ii}}\left(a_{ki} - \sum_{j=1}^{i-1} h_{ij}h_{kj}\right)$

  `// Cálculo del elemento de la diagonal`
  $a_{kk} \equiv h_{kk} = \sqrt{a_{kk} - \sum_{j=1}^{k-1} h_{kj}^2}$

A continuación se presenta la implementación del algoritmo:

```python
import numpy as np
import pandas as pd
np.set_printoptions(formatter={'float': lambda x: f"{x:.4e}"})
def cholesky(A):
    n = A.shape[0]

    # Verificar simetria
    if not np.allclose(A, A.T):
        raise ValueError('La matriz no es simetrica')

    # Verificar definida positiva
    if np.min(np.linalg.eigvals(A)) <= 0:
        raise ValueError('La matriz no es positiva definida')

    H = np.zeros((n,n))

    for k in range(n):
        for i in range(k):
            suma1 = sum(H[i,j]*H[k,j] for j in range(i))
            H[k,i] = (A[k,i] - suma1) / H[i,i]

        suma2 = sum(H[k,j]**2 for j in range(k))
        H[k,k] = np.sqrt(A[k,k] - suma2)

    return H
if __name__ == "__main__":
    # Un ejemplo pequeno para probar que el script funciona solo
    A = np.array([[4,1,1],
                  [1,3,0],
                  [1,0,2]], float)
    H=cholesky(A)
    print('A\n',A)
    print('matriz H\n',H)
    print('Matriz H.T\n',H.T)
    print('A=H*H.T\n',H@H.T)
```

4. minimos cuadrados

5. Método de las potencias

El método de las Potencias se usa frecuentemente para hallar el autovalor dominante de una matriz. Se llama método de las potencias, porque implícitamente se construye a través de las potencias de A.
Sean los autovalores $\lambda_1, \lambda_2, \ldots, \lambda_n$ de A tal que

$$|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_n|$$

esto es, λ_1 es el autovalor dominante de A. Sea v_1 el correspondiente autovector asociado a λ_1. Si $max(g)$ denota el elemento de módulo máximo del vector g. Asumamos que A es diagonalizable.

Algorithm 3: Método de las Potencias

Paso 1. Elegir x_0

Paso 2. **for** $k = 1, 2, 3, \ldots$ ***hasta*** *convergencia* **do**

$$\hat{x}_k = Ax_{k-1}$$

$$x_k = \frac{\hat{x}_k}{\text{máx}\,\hat{x}_k}$$

A continuación se presenta la implementación del algoritmo:

```python
import numpy as np
import pandas as pd
np.set_printoptions(formatter={'float': lambda x: f"{x:.4e}"})

#en tp se indica si se quiere el eige-value maximo o minimo con 'max' o 'min'
def metodo_potencias(A,x0,tol,tp='max'):
  err=np.inf
  if tp=='min':
    if np.min(np.linalg.eig(A)[0])<=0:
      raise ValueError("La matriz no es positiva definida")
    A=np.linalg.inv(A)
    while err>tol:
      xnp=A@x0
      xn=xnp/np.max(xnp)
      err= np.linalg.norm(xn-x0,2)
      x0=xn
  else:
    while err>tol:
      xnp=A@x0
      xn=xnp/np.max(xnp)
      err= np.linalg.norm(xn-x0,2)
      x0=xn
  return np.max(xnp)
if __name__ == "__main__":
    # Un ejemplo pequeno para probar que el script funciona solo
    A = np.array([
        [2, 1, 0],
        [1, 3, 1],
        [0, 1, 4]
    ], dtype=float)
    x0=np.array([[1],[1],[1]])
    tol=1e-4
    eig1=metodo_potencias(A,x0,tol)
    eig2=metodo_potencias(A,x0,tol,'min')
    print('eigen-value maximo:\n',eig1)
    print('eigen-value minimo:\n',1/eig2)
    print('--------------------')
    print('eigen-values con numpy:\n',np.linalg.eig(A)[0])
```