Traballo (16, 17 e 19 de decembro de 2024)

1 Tema 6. Resolución numérica de sistemas lineares.

A resolución numérica de sistemas lineares consiste en atopar o vector solución, x, do sistema linear $Ax = b$, que ten asociada unha matriz invertible A e un vector b. O sistema linear $Ax = b$ tamén se pode escribir como:

$$\left\{\begin{array}{lcl} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n & = & b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n & = & b_2 \\ \vdots & = & \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n & = & b_n \end{array}\right.$$

1.1 Condicionamento da matriz

En Python podemos calcular o condicionamento dunha matriz A usando a función de Numpy:

`linalg.cond(A)`

Vexamos un exemplo de sistema linear mal condicionado.

$$\begin{pmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 32 \\ 23 \\ 33 \\ 31 \end{pmatrix}$$

[1]:
```
import numpy as np
A = np.array([[10,7,8,7],[7,5,6,5],[8,6,10,9],[7,5,9,10]])
b = np.transpose(np.array([32,23,33,31]))
Sol = np.linalg.solve(A, b) # Resolvemos o sistema linear Ax=b
print('Solución do sistema inicial:',Sol,'\n')

# Sistema perturbado 1 (pequenas modificacións nos datos)
db = np.transpose(np.array([0.1,-0.1,0.1,-0.1]))
Sol1 = np.linalg.solve(A, b + db)
print('Solución do sistema inicial:',Sol1,'\n')

# Sistema perturbado 2 (pequenas modificacións na matriz)
dA = np.array([[0,0,0.1,0.2],[0.08,0.04,0,0],[0,-0.02,-0.11,0],[-0.01,-0.01,0,-0.
↪02]])
Sol2 = np.linalg.solve(A+dA, b)
print('Solución do sistema inicial:',Sol2,'\n')

# A inversa de A é:
```

```python
Ainv = np.linalg.inv(A)
display(Ainv)

# Vexamos o condicionamento da matriz A.
condA = np.linalg.cond(A)
print('O condicionamento da matriz A asociada ao sistema linear é: ',condA,'\n')

# Vexamos neste exemplo as cotas do erro relativo dadas nos Teoremas 1 e 2 (moi␣
 ↪boa para o sistema perturbado 1, no sentido de que están próximas ao erro␣
 ↪real).
nA = np.linalg.norm(A,2) # Calculamos a norma 2 pero podemos obter outra norma␣
 ↪cambiando o 2 polo valor indicado na axuda para a norma desexada.
ndA = np.linalg.norm(dA,2)
ndb = np.linalg.norm(db,2)
nb = np.linalg.norm(b,2)
nSol = np.linalg.norm(Sol,2)
nSol2 = np.linalg.norm(Sol2,2)
ndSol1 = np.linalg.norm(Sol1-Sol,2)
ndSol2 = np.linalg.norm(Sol2-Sol,2)

# Sistema perturbado 1 (Teorema 1)
Rerro = ndSol1/nSol
Cerro = condA*ndb/nb
print('O erro do primeiro problema perturbado é: ',Rerro,'\nmentres que a cota␣
 ↪dada usando o condicionamento é:',Cerro,'\n')

# Sistema perturbado 2  (Teorema 2)
Rerro2 = ndSol2/nSol2
Cerro2 = condA*ndA/nA
print('O erro relativo do segundo problema perturbado é: ',Rerro2,'\nmentres que␣
 ↪a cota dada usando o condicionamento é:',Cerro2,'\n')

print('Nótese que o erro relativo (á solución do problema) sería:', ndSol2/nSol)
```

```
Solución do sistema inicial: [1. 1. 1. 1.]

Solución do sistema inicial: [  9.2 -12.6   4.5  -1.1]

Solución do sistema inicial: [-81. 137. -34.  22.]


array([[ 25., -41.,  10.,  -6.],
       [-41.,  68., -17.,  10.],
       [ 10., -17.,   5.,  -3.],
       [ -6.,  10.,  -3.,   2.]])

O condicionamento da matriz A asociada ao sistema linear é:  2984.0927016757
```

```
O erro do primeiro problema perturbado é:  8.198475468037087
mentres que a cota dada usando o condicionamento é: 9.942833687616401

O erro relativo do segundo problema perturbado é:  0.9984414996651639
mentres que a cota dada usando o condicionamento é: 22.741473225797964

Nótese que o erro relativo (á solución do problema) sería: 81.98475468049799
```

1.1.1 Exercicio

Atopa polo menos unha matriz cadrada de orde maior ou igual que 2 que sexa mal condicionada e outra que esté ben condicionada. Comprobao, perturbando un sistema linear que teña asociada esas matrices.

[ ]:
```

```

1.2 Métodos directos

Para resolver o sistema linear $Ax = b$, buscan unha matriz invertible M tal que a matriz MA sexa triangular superior e o novo sistema linear $MAx = Mb$ poida resolverse polo método de remonte.

Imos ver distintos métodos directos para resolver o sistema linear $Ax = b$ con:

$$A = \begin{pmatrix} 9 & 1 & 0 & 0 \\ 2 & 4 & 1 & 0 \\ 0 & 2 & 5 & 1 \\ 0 & 0 & 2 & 3 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}.$$

1.2.1 Método de Gauss (factorización LU)

Para resolver o sistema linear $Ax = b$ buscamos L triangular inferior e U triangular superior tal que $A = LU$.

Teorema (factorización LU dunha matriz) Sexa $A = (a_{ij})$ matriz cadrada de orde n tal que as n submatrices diagonais

$$A = \begin{pmatrix} a_{11} & \dots & a_{1k} \\ \vdots & & \vdots \\ a_{k1} & \dots & a_{kk} \end{pmatrix}, \quad 1 \leq k \leq n,$$

sexan invertibles. Entón existe unha matriz triangular inferior $L = (l_{ij})$ con $l_{ii} = 1$, $1 \leq i \leq n$, e a matriz triangular superior U tal que $A = LU$. Ademais, tal factorización é única.

[2]:
```python
import scipy as sc
from scipy import linalg

A = np.array([[9,1,0,0],[2,4,1,0],[0,2,5,1],[0,0,2,3]])
b = np.transpose(np.array([1,1,1,1]))

# Calculamos a factorización LU da matriz dada
lu, piv = sc.linalg.lu_factor(A)
```

```
L, U = np.tril(lu, k=-1) + np.eye(4), np.triu(lu)
print('A matriz L é: \n', L,'\ne U=\n',U, '\n')
# Resolvemos o sistema linear usando a factorización LU
x = sc.linalg.lu_solve((lu, piv), b)
print('A solución do sistema linear é:', x)
# Resolvemos o sistema linear Ax=b con outro método
print('Solución do sistema linear:    ',np.linalg.solve(A, b),'\n')
```

```
A matriz L é:
 [[1.         0.         0.         0.        ]
 [0.22222222 1.         0.         0.        ]
 [0.         0.52941176 1.         0.        ]
 [0.         0.         0.44736842 1.        ]]
e U=
 [[9.         1.         0.         0.        ]
 [0.         3.77777778 1.         0.        ]
 [0.         0.         4.47058824 1.        ]
 [0.         0.         0.         2.55263158]]

A solución do sistema linear é: [0.09020619 0.18814433 0.06701031 0.28865979]
Solución do sistema linear:     [0.09020619 0.18814433 0.06701031 0.28865979]
```

1.2.2 Método de Cholesky

O método de Cholesky para resolver un sistema linear $Ax = b$ con A simétrica e definida positiva consiste en obter a factorización de Cholesky $A = BB^T$ e despois resolver os sistemas lineais $By = b$ e $B^T x = y$.

Teorema (factorización de Cholesky) Se a matriz A é simétrica e definida positiva, existe polo menos unha matriz real triangular inferior B tal que $A = BB^T$, esta igualdade constitúe unha factorización de Cholesky da matriz A. Ademais, podemos impoñer que os elementos da diagonal da matriz B sexan todos maiores que 0, co que a factorización $A = BB^T$ é única.

[3]:
```
# Comprobemos que a matriz A dade é definida positiva
autovalores = sc.linalg.eigvals(A)
print('Son tódolos autovalores de A positivos?', autovalores > 0)
# Como son todos positivos sabemos que A é definida positiva, vexamos cal é a␣
 ↪factorización de Cholesky
B = sc.linalg.cholesky(A, lower=True)
print('A matriz B é: \n', B,'\n')
# Resolvemos o sistema linear usando a factorización de Cholesky en scipy
c, low = sc.linalg.cho_factor(A)
x = sc.linalg.cho_solve((c,low),b)
print('A solución do sistema linear é:', x)
# Resolvemos o sistema linear Ax=b con outro método
print('Solución do sistema linear:    ',np.linalg.solve(A, b),'\n')
print('PARECE QUE NON FUNCIONA CORRECTAMENTE!!! Por que? \n')
```

```
# Consideresmos unha nova matriz A2:
A2 = np.array([[9,1,0,0],[1,4,1,0],[0,1,5,1],[0,0,1,3]])
B = sc.linalg.cholesky(A2, lower=True)
print('A matriz B é: \n', B,'\n')
# Resolvemos o sistema linear usando a factorización de Cholesky en scipy
c, low = sc.linalg.cho_factor(A2)
x = sc.linalg.cho_solve((c,low),b)
print('A solución do sistema linear é:', x)
# Resolvemos o sistema linear Ax=b con outro método
print('Solución do sistema linear:    ',np.linalg.solve(A2, b),'\n')
print('PARECE QUE FUNCIONA CORRECTAMENTE!!! Por que? \n')
```

```
Son tódolos autovalores de A positivos? [ True  True  True  True]
A matriz B é:
 [[3.         0.         0.         0.        ]
 [0.66666667 1.88561808 0.         0.        ]
 [0.         1.06066017 1.96850197 0.        ]
 [0.         0.         1.01600102 1.40276225]]

A solución do sistema linear é: [0.08855292 0.20302376 0.09935205 0.30021598]
Solución do sistema linear:     [0.09020619 0.18814433 0.06701031 0.28865979]

PARECE QUE NON FUNCIONA CORRECTAMENTE!!! Por que?

A matriz B é:
 [[3.         0.         0.         0.        ]
 [0.33333333 1.97202659 0.         0.        ]
 [0.         0.50709255 2.17781017 0.        ]
 [0.         0.         0.45917684 1.67007683]]

A solución do sistema linear é: [0.08855292 0.20302376 0.09935205 0.30021598]
Solución do sistema linear:     [0.08855292 0.20302376 0.09935205 0.30021598]

PARECE QUE FUNCIONA CORRECTAMENTE!!! Por que?
```

1.2.3 Método de Householder, factorización QR

O método de Householder para resolver un sistema linear $Ax = b$ equivale a atopar $(n-1)$ matrices de Householder $H_1, H_2, \ldots, H_{n-1}$ tal que a matriz $H_{n-1} \ldots H_2 H_1 A$ sexa triangular superior. Restaría resolver por remonte o sistema linear:

$$H_{n-1} \ldots H_2 H_1 A = H_{n-1} \ldots H_2 H_1 b.$$

Teorema (factorización QR): Dada unha matriz A de orde n, existe unha matriz ortogonal Q e unha matriz triangular superior R tal que $A = QR$. Ademais, podemos conseguir que os elementos da diagonal de R sexan todos maiores ou iguais que 0. Se a matriz A é invertible, a correspondente factorización $A = QR$ é única.

Usando o método de Householder $R = H_{n-1} \dots H_2 H_1 A$ e $Q = (H_{n-1} \dots H_2 H_1)^{-1} = H_1 H_2 \dots H_{n-1}$.

[4]:
```
# Obtemos a factorización QR
Q, R = sc.linalg.qr(A)
print('A matriz Q é: \n', Q,'\ne R=\n',R, '\n')
# Resolvemos o sistema usando a factorización QR
x = np.linalg.solve(R, np.matmul(np.transpose(Q),b))
print('A solución do sistema linear é:', x)
# Resolvemos o sistema linear Ax=b con outro método
print('Solución do sistema linear:    ',np.linalg.solve(A, b),'\n')
```

```
A matriz Q é:
 [[-0.97618706  0.19069252 -0.09216815 -0.0469065 ]
 [-0.21693046 -0.85811633  0.41475666  0.21107926]
 [-0.         -0.47673129 -0.78342924 -0.39870528]
 [-0.         -0.         -0.4535643   0.89122356]]
e R=
 [[-9.21954446 -1.84390889 -0.21693046  0.        ]
 [ 0.         -4.19523539 -3.2417728  -0.47673129]
 [ 0.          0.         -4.40951814 -2.14412213]
 [ 0.          0.          0.          2.27496539]]

A solución do sistema linear é: [0.09020619 0.18814433 0.06701031 0.28865979]
Solución do sistema linear:     [0.09020619 0.18814433 0.06701031 0.28865979]
```

1.2.4 Exemplo

Vexamos como construír matrices que teñen moitos ceros, e só algúnhas diagonais distintas de cero. Así como computar tempos de cáculo dos métodos anteriores encapsulando o código nunha función.

[5]:
```
# Construímos unha matriz A de orde n=10 pentadiagonal
n = 10
data = np.array([4*np.ones(n), -2*np.ones(n), np.ones(n),-3*np.ones(n),2*np.
 ↪ones(n)])
diags = np.array([0, -1, -2,1,2])
AA = sc.sparse.spdiags(data, diags, n, n).toarray()
print(AA)
# construimos b
bb = np.ones(n)

def MetodoGauss(A,b):
    lu, piv = sc.linalg.lu_factor(A)
    x = sc.linalg.lu_solve((lu, piv), b)
    return x
```

```
# Chamamos ao método e escribimos por pantalla o tempo
%time x = MetodoGauss(AA,bb)
print(x)
```

```
[[ 4. -3.  2.  0.  0.  0.  0.  0.  0.  0.]
 [-2.  4. -3.  2.  0.  0.  0.  0.  0.  0.]
 [ 1. -2.  4. -3.  2.  0.  0.  0.  0.  0.]
 [ 0.  1. -2.  4. -3.  2.  0.  0.  0.  0.]
 [ 0.  0.  1. -2.  4. -3.  2.  0.  0.  0.]
 [ 0.  0.  0.  1. -2.  4. -3.  2.  0.  0.]
 [ 0.  0.  0.  0.  1. -2.  4. -3.  2.  0.]
 [ 0.  0.  0.  0.  0.  1. -2.  4. -3.  2.]
 [ 0.  0.  0.  0.  0.  0.  1. -2.  4. -3.]
 [ 0.  0.  0.  0.  0.  0.  0.  1. -2.  4.]]
Wall time: 0 ns
[ 5.00000000e-01  1.00000000e+00  1.00000000e+00  5.00000000e-01
 -4.43273036e-16  1.48921482e-16  5.00000000e-01  1.00000000e+00
  1.00000000e+00  5.00000000e-01]
```

1.2.5 Exercicio

Completa o exemplo anterior contruíndo unha función para cada método visto. Utilízaas para comparar os tempos de execución dos anteriores métodos ao resolver o sistema $Ax = b$, tal que A é unha matriz de orde $n = 1000$ tridiagonal con 1 na diagonal inferior e superior e 4 na diagonal principal, mentres que b é un vector de 1.

```
[ ]:
```

1.3 Métodos iterativos.

A solución do problema e o límite dunha sucesión $\{x_k\}_{k\geq 0}$ de soluciówns aproximadas (os cálculos deténense para algún k_0).

Dado o sistema linear $Ax = b$ trátase de buscar unha matriz B e un vector c tal que a matriz $(I - B)$ sexa invertible e que a solución do sistema linear $x = Bx + c$ tamén o sexa do inicial.

Presentan a forma: $x_{k+1} = Bx_k + c$, $k \geq 0$, con x_0 sendo un vector arbitrario. Á matriz B chámaselle matriz do método iterativo.

Teorema Sexa B a matriz dun método iterativo, entón as seguintes proposicións son equivalentes:

O método iterativo é converxente.

O raio espectral de B é menor que 1, $\rho(B) < 1$.

$||B|| < 1$ para polo menos, unha norma matricial $|| \cdot ||$.

Veremos os metodos de Jacobi e Gauss-Seidel para os que a matriz A se descompón como $A = M - N$, onde M é unha matriz invertible. Entón temos:

$$Ax = b \Leftrightarrow Mx = Nx + b \Leftrightarrow x = M^{-1}Nx + M^{-1}b,$$

que ten a forma $x = Bx + c$ e $B = M^{-1}N = I - M^{-1}A$. Máis concretamente imos considerar a descomposición $A = D - L - U$ onde: D é a diagonal principal de A, $d_{ij} = a_{ij}$ se $i = j$ e 0 noutro caso; L é a parte triangular inferior de $-A$, $l_{ij} = -a_{ij}$ se $i > j$ e 0 noutro caso; e U é a parte triangular superior de $-A$, $u_{ij} = -a_{ij}$ se $i < j$ e 0 noutro caso.

1.3.1 Método de Jacobi

Consideramos que $M = D$ polo que $N = L + U$ e a matriz do método de Jacobi é $B = D^{-1}(L + U)$. Para que D sexa invertible precisamos que $a_{ii} \neq 0$, $1 \leq i \leq n$. Vexamos unha posible implementación do método de Jacobi.

[6]:
```
def Jacobi(A, b, k_stop, tol=1e-9):
    x = np.zeros(np.size(b))  # x0 fixo, aínda que podería ser unha entrada da␣
 ↪función
    D = np.diag(np.diagonal(A))
    N = -A + D
    for k in range(k_stop):
        x_old  = x
        x = np.matmul(np.linalg.inv(D),(b + np.matmul(N, x)))
        if np.linalg.norm(x-x_old,2)/np.linalg.norm(x,2) < tol:
            break
    return x, k

# Resolvemos o exemplo co método de Jacobi
k_stop = 100
x_jac, k_end = Jacobi(A,b,k_stop)
print('A solución usando Jacobi en',k_end,' iteracións é:', x_jac)
# Resolvemos o sistema linear Ax=b con outro método
print('Solución exacta  do sistema linear é          :',np.linalg.solve(A,␣
 ↪b),'\n')
```

```
A solución usando Jacobi en 32  iteracións é: [0.09020619 0.18814433 0.06701031
0.28865979]
Solución exacta  do sistema linear é           : [0.09020619 0.18814433
0.06701031 0.28865979]
```

1.3.2 Método de Gauss-Seidel

Consideramos que $M = D - L$ polo que $N = U$ e a matriz do método de Gauss-Seidel é $B = (D - L)^{-1}U$. Para que D sexa invertible precisamos que $a_{ii} \neq 0$, $1 \leq i \leq n$.

1.3.3 Exercicio

Implementa unha función para resolver sistemas lineais usando o método de Gauss-Seidel.

Resolve o mesmo sistema que resolvemos cos anteriores métodos.

Razoa porque son converxentes estes métodos iterativos e compróbao numéricamente.

Utiliza estes dous métodos para resolver o sistema do anterior exercicio onde a matriz A é de orde 1000.

```
[ ]:
```