Elementos de Cálculo Numérico / Cálculo Numérico

Primer Cuatrimestre 2021

Sexto ejercicio computacional
17-05-21 al 24-05-21
Recuerde subir el archivo en formato ejercicioX_NOMBREAPELLIDO.py
Recuerde enviar su código al hacer consultas

El objetivo de este ejercicio es implementar los métodos iterativos de Jacobi (J) y Gauss-Seidel (GS), y realizar una pequeña comparación de desempeño entre ambos métodos.

Para esto, consideremos en primer paso las matrices

$$A_1 = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \quad A_2 = \begin{bmatrix} 1 & 1 \\ 0 & 2 \end{bmatrix}$$

y el vector de soluciones

$$b = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Partiendo de estas matrices:

A Encuentre la solución a la ecuación $Ax = b$ para las matrices A_1 y A_2, de forma de poder emplear esa solución para corroborar los resultados de los siguientes pasos (ya sea manualmente o mediante la función que calcula la inversa de una matriz de numpy.linalg).

B Construya una función `jacobi(A,b,x0,max_iter,tol)`, que reciba la matriz de interés `A`, el vector de soluciones `b`, una semilla `x0`, un número tope de iteraciones a realizar `max_iter` y una tolerancia para la diferencia entre el resultado obtenido y el resultado correcto `tol`. Para esto, básese en el siguiente modelo:

```
def jacobi(A,b,x0,max_iter,tol):
    D = ## COMPLETAR ##
    L = ## COMPLETAR ##
    U = ## COMPLETAR ##
    M = D
    N = U + L
    B = ## COMPLETAR ##
    C = ## COMPLETAR ##
    xN = x0
    resto = ## COMPLETAR ##
    iter = 0
    while ## COMPLETAR ##:
        xN = ## COMPLETAR ##
        iter = iter + 1
        resto = ## COMPLETAR ##
```

```
if iter==max_iter:
    print('max_iter alcanzado)
return([xN,iter])
```

Ayuda: Partiendo de que ya realizamos `import numpy as np` y `import numpy.linalg as npl`

- La funcion `np.diag` permite obtener la diagonal de una matriz, y construir una matriz diagonal a partir de ella. Por ejemplo, `np.diag(A2)` dará por resultado `array[1,2]`, y `np.diag(np.diag(A2))` dará por resultado `array([[1,0],[0,2]])`.
- La funcion `np.triu` permite obtener la componente triangular superior de una matriz. Por default esta incluye la diagonal. Para evitar esto, agregamos el argumento `k`. Por ejemplo, `np.triu(A2,k=1)` dará por resultado `array([[0,1],[0,0]])`. La función `np.tril` tiene un funcionamiento análogo, pero con el triangulo inferior. En este caso, para sacarnos de encima la diagonal, usamos `k=-1`.
- La función para invertir una matriz la tenemos en el paquete de algebra lineal: la inversa de `A` la obtenemos con `npl.inv(A)`.
- La función para calcular la norma de un vector (util para calcular el resto) también está en el paquete de álgebra lineal: `npl.norm`. Por default, calcula la norma 2.

C Probar la función `jacobi` usando las matrices A_1 y A_2, y comprobar que el resultado sea el esperado.

D Construir una función `gauss_seidel(A,b,x0,max_iter,tol)` análoga a la anterior, pero que implemente el método de GS. El modelo de función es análogo al anterior.

F Testee la función anterior usando las matrices A_1 y A_2.

G Construya una función que genere aleatoriamente una A matriz de 3x3, y un vector de soluciones b de 3x1, de forma tal que la matriz sea inversible (garantizando que se pueda resolver el sistema de ecuaciones). Para esto, básese en el siguiente modelo:

```
def matriz_y_vector(n):
    A = np.random.rand(n, n)
    while ## COMPLETAR ##:
        A = np.random.rand(n, n)
    b = ## COMPLETAR ##
    return([A,b])
```

Ayuda:

- La función `np.random.rand(n,m)` construye una matriz aleatoria, con coeficientes entre 0 y 1.
- La función `npl.det` calcula el determinante de una matriz, util para saber fácilmente si la matriz generada es inversible o no.

H Empleando estos ingredientes, construya un programa que aplique J y GS a matrices y vectores generados aleatoriamente, y tome nota de la cantidad de pasos que tuvo que realizar el programa para llegar al nivel de tolerancia especificado. Para esta prueba, considere un límite de pasos suficientemente grande como para que siempre se alcance el nivel de tolerancia especificado. Básese en el siguiente modelo:

```
trials_J = []
trials_GS = []
Ntrials = 10**4
trials = 0
x0 = np.transpose(np.array([[0,0,0]]))
max_iter = 10**6
tol = 10**-5
while trials<Ntrials:
    Ab = matriz_y_vector(3)
    A = ## COMPLETAR ##
    b = ## COMPLETAR ##
    rGS= gauss_seidel(A,b,x0,max_iter,tol)
    rJ = jacobi(A,b,x0,max_iter,tol)
    trials = trials + 1
    print(trials)

import matplotlib.pyplot as plt
plt.scatter(trials_J,trials_GS)
plt.ylabel('Gauss-Seidel')
plt.xlabel('Jacobi')
plt.yscale('log')
plt.xscale('log')
```

Basandose en esto, ¿que puede decir sobre el desempeño de ambos métodos? ¿Funciona uno mejor que el otro?

Sugerencia: Mientras prueba el programa, emplee un valor de `Ntrials` menor, como por ejemplo 10 o 100, para acelerar el testeo. Luego utilize un valor más alto (como el de 10^3 propuesto), de forma de obtener una cantidad razonable de puntos.