Taller 2 AN 2130

Andrés Alarcón, Nicolás Barragán, Gabriel de Souza, Pablo Santander

September 22, 2021

Abstract

El taller 2 de análisis numérico busca poner en práctica diferentes métodos como Gauss Sidel, Jacobi, Gradiente, Newton multivariado, entre otros, para poder dar solución a sistemas de ecuaciones. A continuación encontrará la solución a 4 puntos distintos donde se aplican algunos de los métodos mencionados para dar una solución a las preguntas planteadas para cada uno.

Part I
Solución a los ejercicios

1 Punto 2

1. Cálculo del radio espectral de la matriz de transición
   Para este punto el cálculo se realiza utilizando la matriz diagonal y superior, estas 2 matrices se restan y con este valor podemos utilizar las funciones de numpy eigvals y np.dot pára calcular el radio espectral utilizando np.linalg.eigvals. Obtuvimos el siguiente resultado:

   ```
   Radio espectral 0.625
   ```

2. Máximo de iteraciones aproximando la solucion con tolerancia de 1e-16 comparando con Jacobi

   Utilizando el método de Gauss Seidel obtenemos como resultado:

```
Por Gauss Sidel:
99 [ 0.499      -0.58066667  0.59933333]

########################
VERIFICACIÓN POR LIBRERIA NUMPY
[ 0.499      -0.58066667  0.59933333]
```

Este resultado se verificó con la libreria que posee Numpy por defecto para resolver este tipo de sistemas de ecuaciones. Sin embargo por el algoritmo normal obtuvimos que el resultado salia en 37 iteraciones. Comparando con el método Jacobi obtenemos que el resultado es el mismo pero con una sola iteración:

```
jacobi: [ 0.499      -0.58066667  0.59933333]
Con jacobi el resultado es:  [ 0.499      -0.58066667  0.59933333]
```

Podemos ver todos los resultados anteriores en la siguiente imagen:

```
Radio espectral 0.625

Por Gauss Sidel:
99 [ 0.499      -0.58066667  0.59933333]

########################
VERIFICACIÓN POR LIBRERIA NUMPY
[ 0.499      -0.58066667  0.59933333]

jacobi: [ 0.499      -0.58066667  0.59933333]
Con jacobi el resultado es:  [ 0.499      -0.58066667  0.59933333]
```

3. Qué pasa cuando a13 = -2
   Cuando cambiamos el valor a13 que estaba inicialmente en 0 por -1 vemos un aumento en las iteraciones realizadas por Jacobi y un cambio en los valores de él mismo y de Gauss Seidel.
   A continuación tenemos una imagen con los resultados obtenidos:

```
Por Gauss Sidel:
99 [ 1.09833333 -1.06013333  0.47946667]

########################
VERIFICACIÓN POR LIBRERIA NUMPY
[ 1.09833333 -1.06013333  0.47946667]

jacobi: [ 1.09833333 -1.06013333  0.47946667]
jacobi: [ 1.09833333 -1.06013333  0.47946667]
jacobi: [ 1.09833333 -1.06013333  0.47946667]
jacobi: [ 1.09833333 -1.06013333  0.47946667]
Con jacobi el resultado es:  [ 1.09833333 -1.06013333  0.47946667]
```

La impresion de jacobi 4 veces quiere decir que le costó 4 iteraciones obtener el resultado, comparado a cuando el valor era 0 que fue solo una iteración. Así mismo la cantidad de iteraciones de Gauss Seidel se redujo a 30.

2 Punto 2

1. Es la matriz diagonalmente dominante? ¿Se puede reorganizar con operaciones entre filas para que sea diagonalmente dominante?
   No es diagonal dominante debido a que el primer coeficiente de la diagonal el cual es u, tiene como valor 1, pero los demas coeficientes de la ecuacion tienen valor 4 y 0 en valor absoluto, por lo que son mayores que u. De tal manera no se cumple la condicion de que el elemento de la diagonal en valor absoluto sea mayor o igual a los demas coeficientes de la ecuacion.En este caso podriamos realizar una operacion de matriz al cambiar la fila numero 1 por la fila numero 2, de tal manera la matriz de coeficientes se volveria diagonalmente dominante debido a que sus elementos diagonales cumplirian la condicion de que el elemento sea mayor o igual a los demas coeficientes de la ecuacion, y que sea estrictamente mayor en alguno.abs. Esto para la matriz ii

2. Hallar la solucion mediante Jacobi y determinar si converge

```
jacobi: [ 5.99999794 -0.24999948 -3.99999863]
Con jacobi el resultado es:  [ 5.99999794 -0.24999948 -3.99999863]
El metodo de Jacobi converge debido a que la matriz es diagonalmente domina
bido a las operaciones de matriz realizadas para que cumpliera esta condici
```

3. Comparar con Gauss Seidel con tolerancia 10-6

```
Por Gauss Sidel:
99 [ 4  0 -2]
```

3 Punto 6

1. Implementación y tabla del método Jacobi con 10 iteraciones.

$$2x - z = 1$$
$$\beta x + 2y - z = 2$$
$$-x + y + \alpha z = 1$$

Nuestra implementación del método Jacobi para este ejercicio fue la siguiente: El producto entre el valor de x0 y la matriz bidimensional de los elementos diagonales de la matriz original se obtiene y se le resta al valor de los resultados de los ecuaciones dentro del sistema proporcionado. Con este código se hallan las primeras 10 iteraciones a partir del valor dado a x0 como [1, 2, 3], demostrado en la siguiente gráfica:

```
TABLA
Solución  1
[ 2.  -2.5  0. ]
Solución  2
[ 0.5 -3.   5.5]
Solución  3
[ 3.25 -2.75  4.5 ]
Solución  4
[ 2.75 -7.75  7.  ]
Solución  5
[ 4.  -8.  11.5]
Solución  6
[  6.25 -12.75  13.  ]
Solución  7
[  7. -18.  20.]
Solución  8
[ 10.5 -23.   26. ]
Solución  9
[ 13.5 -33.   34.5]
Solución  10
[ 17.75 -43.25  47.5 ]
```

4 Punto 9

Fue realizado una implementación en Python del método gradiente conjugado, para así calcular el tiempo y las interacciones necesarias para llegar a la respuesta. Este resultado fue comparado con los resultados de la función general de solución de sistemas de ecuaciones (no usando gradiente conjugado), para tener una mejor idea de los tiempos que ambos usan.

Lastimosamente 10000 es un numero que genera un gran esfuerzo para lograr la solución. Dado esta razón las pruebas fueron realizadas con n=2000.

La implementacion del code fue hecha basándose en los siguientes códigos:

1. https://sophiamyang.medium.com/descent-method-steepest-descent-and-conjugate-gradient-in-python-85aa4c4aac7b
2. https://gist.github.com/sfujiwara/
3. https://ikuz.eu/machine-learning-and-computer-science/the-concept-of-conjugate-gradient-descent-in-python/

```
In [7]: runfile('C:/Users/cuerv/.spyder-py3/temp
Metodo 1.
Iteraciones: 3698
tiempo:  5.826340436935425

Metodo 2.
tiempo:  0.13937759399414062
```