Tarea 7 - Método de Jacobi y Gauss-Seidel
Métodos Numéricos

Rafael Alejandro García Ramírez

4 de octubre de 2023

1. Introducción

1.1. Método la potencia

Supongamos una matriz $\mathbf{A}$ de dimensiones $n \times n$ con valores propios $\lambda_1, \lambda_2, \ldots, \lambda_n$ con los vectores independientes $v_1, v_2, \ldots, v_n$, entonces podemos ordenar los valores propios tal que

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

Para lo cual solamente nos importa que los primeros dos sean mayores. Dado que los vectores propios forman una base, para un vector v_0 podemos formar

$$x_0 = c_1 v_1 + c_2 v_2 + \cdots + c_n v_n$$

Tal que, si multiplicamos ambos lados por $\mathbf{A}$ y recordando que $\mathbf{A}x_i = \lambda_i x_i$

$$\mathbf{A}x_0 = c_1\mathbf{A}v_1 + c_2\mathbf{A}v_2 + \cdots + c_n\mathbf{A}v_n = c_1\lambda_1 v_1 + c_2\lambda_2 v_2 + \cdots + c_n\lambda_n v_n$$

Si sacamos como factor común algún, supongamos $c_1\lambda_1$ esto se convierte en

$$\mathbf{A}x_0 = c_1\lambda_1\left[v_1 + \frac{c_2}{c_1}\frac{\lambda_2}{\lambda_1}v_2 + \cdots + \frac{c_n}{c_1}\frac{\lambda_n}{\lambda_1}v_n\right] = c_1\lambda_1 x_1$$

Donde vemos que obtenemos una expresión para x_1 para el lado derecho. Repitiendo esto tenemos que

$$\mathbf{A}x_1 = \lambda_1\left[v_1 + \frac{c_2}{c_1}\frac{\lambda_2^2}{\lambda_1^2}v_2 + \cdots + \frac{c_n}{c_1}\frac{\lambda_n^2}{\lambda_1^2}v_n\right] = c_1\lambda_1 x_2$$

Realizando esto iterativamente se reduce a la siguiente relación recursiva

$$\mathbf{A}x_{(k-1)} = \lambda_1\left[v_1 + \frac{c_2}{c_1}\frac{\lambda_2^{(k)}}{\lambda_1^{(k)}}v_2 + \cdots + \frac{c_n}{c_1}\frac{\lambda_n^{(k)}}{\lambda_1^{(k)}}v_n\right] = c_1\lambda_1 x_{(k)} \tag{1}$$

En general, para cuando k es demasiado grande, la ecuación (1) se termina convirtiendo en $\mathbf{A}x_{(k+1)} \sim \lambda_1 v_1$. Debido que al realizar este método y al estar buscando el valor propio más grande, se debe de realizar una normalización del vector propio lo cuál nos traerá (por la forma de (1)) el valor propio más grande.

Los criterios para detener la iteración es, dada una tolerancia específica: que la diferencia (cualquiera que esta esté definida) entre el vector del estado anterior y el estado actual sea menor que la tolerancia; que el ángulo entre los vectores propios del estado actual y el anterior sea menor que una tolerancia específica; o que la norma del vector residual sea menor que esta tolerancia.

1.2. Método de la potencia inversa

Para el método de la potencia inversa, recordemos que los valores propios de la matriz $\mathbf{A}^{-1}$ son los recíprocos del valores propios de la matriz original $\mathbf{A}$, por lo que si repetimos el proceso del método de la potencia podemos encontrar el mayor $1/\lambda_1$ el cuál será el menor valor propio de nuestra matriz. Más explícitamente podemos verlo

$$\mathbf{A}x = \lambda x \Leftrightarrow \mathbf{A}^{-1}x = \frac{1}{\lambda}x$$

Dado que usualmente es costoso realizar la inversión de una matriz, para este caso realizaremos una descomposición $\mathbf{A} = \mathbf{LU}$ para $\mathbf{L}$ una matriz triangular inferior y $\mathbf{U}$ una matriz triangular superior, con esto en mente el método se reduce al proceso iterativo

$$\mathbf{U}x^{(k+1)} = \mathbf{L}^{-1}x^{(k)}$$

1.3. Método de Jacobi para valores propios y vectores propios

Para este método, sea **A** una matriz simétrica. La esencia de este método es realizar rotaciones planares iterativamente determinados para reducir los valores que se encuentran fuera de la diagonal y aumentar aquellos que se encuentran dentro de la diagonal. Esto es para una matriz de la forma, con posiciones p y q

$$R_{ij}(\theta) = \begin{cases} \cos(\theta) & \text{si } i = j, i = p, \text{ o } j = q \\ -\sin(\theta) & \text{si } i = p, j = q \\ \sin(\theta) & \text{si } i = q, j = p \\ \delta_{ij} & \text{si } i \neq j, i \neq p, \text{ y } j \neq q \end{cases}$$

Para δ_{ij} la delta de Kronecker. Esencialmente se realiza una rotación de la forma $\mathbf{R}^T\mathbf{AR}$ para la cuál, descomponiendo cada multiplicación individual llegamos a la relación para el ángulo

$$\theta = \frac{1}{2}\arctan\left(\frac{2a_{ij}}{a_{ii} - a_{jj}}\right)$$

Para $\theta = \pi/4$ cuando $a_{ii} = a_{jj}$. Por lo tanto, este sistema se basa en aplicar para todos los elementos afuera de la diagonal su rotación correspondiente con una condición de parada siendo que, la suma de los cuadrados de todos los elementos afuera de la diagonal sea menor que nuestra tolerancia.

2. Pseudocódigo

Los pseudocódigos que se utilizaron para realizar la implementación en `C` para los algoritmos los diferentes métodos en esta tarea son los siguientes:

Algorithm 1 Método de la potencia para encontrar el valor propio más grande

Require: N, matriz, vector, tolerancia, max_iteraciones
Ensure: El eigenvalor más grande

```
1:  function METODO_POTENCIA(N, matriz, vector, tolerancia, max_iteraciones)
2:
3:      Reservar memoria vector : vector_nuevo
4:      max ← 0
5:
6:      while max_iteraciones > 0 do                          ▷ Comenzamos iterando
7:
8:          vector_nuevo ← matriz × vector                    ▷ Realizamos la multiplicación
9:
10:         max ← vector_nuevo[0]                             ▷ Calculamos el máximo del vector
11:         for i ← 1 hasta N − 1 do
12:             if |vector_nuevo[i]| > max then
13:                 max ← |vector_nuevo[i]|
14:
15:         for i ← 0 hasta N − 1 do                          ▷ Normalizamos el vector
16:             vector_nuevo[i] ← vector_nuevo[i]/max
17:
18:         if Sup_{x_i} ‖vector_nuevo − vector‖ < tolerancia then   ▷ Revisamos condición de convergencia
19:             return max
20:             END
21:
22:         for i ← 0 hasta N − 1 do                          ▷ Actualizamos el vector
23:             vector[i] ← vector_nuevo[i]
24:         max_iteraciones ← max_iteraciones − 1
25:     Imprimir(Error método Potencia: No se ha encontrado el valor propio.)   ▷ Regresamos mensaje de error
26:     return NULL
```

Para efectos prácticos, método de la potencia consiste en los siguiente pasos de manera resumida

1. Multiplicar la matriz por el vector y guardarse en un vector nuevo, es decir $\mathbf{A}x^{(k)} = x^{(k+1)}$
2. Normalizar el vector nuevo $x^{(k+1)}$
3. Si el supremo de las diferencias del vector y el vector nuevo es menor que la tolerancia terminar el proceso, en caso contrario repetir pasos 1 y 2.

El valor con el que se normaliza el vector nuevo en la última iteración será el correspondiente al valor propio mayor y el vector nuevo en su último estado será el vector propio asociado.

Para el método de la potencia inversa, se realiza un proceso similar, para una descomposición previa

Algorithm 2 Método de la potencia inversa para encontrar el valor propio más pequeño

Require: N, matriz, vector, tolerancia, max_iteraciones
Ensure: El eigenvalor más pequeño

```
1:  function POTENCIA_INVERSA(N, matriz, vector, tolerancia, max_iteraciones)
2:
3:      Reservar memoria matriz : L, U          ▷ Reservamos memoria para las matrices triangulares
4:      Reservar memoria vector : dummy, vector_nuevo
5:
6:      A = LU                                  ▷ Descomponemos la matriz inicial
7:      L ← L^T                                 ▷ Invertimos la matriz inferior
8:
9:      max ← 0
10:     while max_iteraciones > 0 do            ▷ Comenzamos iterando
11:
12:         dummy ← L × vector                  ▷ Multiplicamos la parte derecha
13:         vector_nuevo ← Resolver sustitucion hacia adelante(U, dummy)   ▷ Resolvemos la parte izquierda
14:
15:         max ← vector_nuevo[0]               ▷ Buscamos el máximo
16:         for i ← 1 hasta N − 1 do
17:             if |vector_nuevo[i]| > max then
18:                 max ← |vector_nuevo[i]|
19:
20:         for i ← 0 hasta N − 1 do            ▷ Normalizamos el vector
21:             vector_nuevo[i] ← vector_nuevo[i]/max
22:
23:         if Sup_{x_i} ‖vector_nuevo − vector‖ < tolerancia then   ▷ Revisamos condición de convergencia
24:             return max
25:             END
26:
27:         for i ← 0 hasta N − 1 do            ▷ Actualizamos el vector
28:             vector[i] ← vector_nuevo[i]
29:
30:         max_iteraciones ← max_iteraciones − 1
31:     Imprimir(Error método PotenciaInversa: No se ha encontrado el valor propio.)
32:     return NULL
```

Esencialmente consiste en los mismos pasos que el método de la potencia pero con el paso agredado de la descomposición

1. Descomponer $\mathbf{A} = \mathbf{LU}$
2. Encontrar $\mathbf{L}^T$
3. Calcular $\mathbf{L}^T x^{(k)} = w$
4. Resolver el sistema $\mathbf{U}x^{(k+1)} = w$
5. Si el supremo de las diferencias de los vectores en sus dos momentos es menor que la tolerancia detener el proceso, en caso contrario reescribir todos los valores del momento anterior al actual y repetir los pasos 3 y 4.

Una variación ligera a los anteriores pseudocódigos es aquella que se realizó para determinar el conjunto k de valores propios mayores y menores

Algorithm 3 Método para encontrar los k eigenvalores más grandes

Require: N, k, matriz, vectores, eigenvalores, tolerancia, iteraciones
Ensure: Los k eigenvalores más grandes

1: **function** POTENCIA_K_VALORES_MAYORES(N, k, matriz, vectores, eigenvalores, tolerancia, iteraciones)
2: Reservar memoria para dummy1, dummy2, temp
3:
4: **for** $i \leftarrow 0$ **hasta** $k-1$ **do** ▷ Inicializar eigenvalores
5: $eigenvalores[i] \leftarrow 0.0$
6:
7: $eigenvalor\_anterior, contador \leftarrow 0, 0$
8: producto, norma $\leftarrow 0.0, 0.0$
9:
10: **for** $i \leftarrow 0$ **hasta** $k-1$ **do** ▷ Comenzamos iterando
11:
12: **for** $d \leftarrow 0$ **hasta** $N-1$ **do** ▷ Inicializamos el vector de ayuda 1
13: $dummy1[d] \leftarrow 1.0$
14:
15: $eigenvalor\_anterior, contador \leftarrow 0, 0$
16:
17: **while** $contador <$ iteraciones **do**
18:
19: **for** $d \leftarrow 0$ **hasta** $N-1$ **do** ▷ Inicializamos el vector de ayuda 2
20: $dummy2[d] \leftarrow 0.0$
21:
22: **for** $j \leftarrow 0$ **hasta** $i-1$ **do** ▷ Calculamos la proyección
23: producto $\leftarrow$ producto_punto(N, dummy1, vectores$[j]$)
24: **for** $jj \leftarrow 0$ **hasta** $N-1$ **do**
25: dummy1$[jj]$ $\leftarrow$ dummy1$[jj]$ $-$ producto $\cdot$ vectores$[j][jj]$
26:
27: multiplicar_matriz_vector(matriz, dummy1, temp) ▷ Realizamos método de la potencia
28: $eigenvalores[i] \leftarrow$ producto_punto(dummy1, temp)
29:
30: norma $\leftarrow \sqrt{\text{producto\_punto(temp, temp)}}$ ▷ Normalizamos
31: **for** $jj \leftarrow 0$ **hasta** $N-1$ **do**
32: dummy1$[jj]$ $\leftarrow$ temp$[jj]$/norma
33:
34: **if** $|eigenvalores[i] - eigenvalor\_anterior| <$ tolerancia **then** ▷ Condición de convergencia
35: **BREAK**
36:
37: $eigenvalor\_anterior \leftarrow eigenvalores[i]$
38: $contador \leftarrow contador + 1$
39:
40: **for** $jj \leftarrow 0$ **hasta** $N-1$ **do** ▷ Actualizamos los vectores propios
41: vectores$[i][jj]$ $\leftarrow$ dummy1$[jj]$

Como podemos ver, el método es bastante similar al método de la potencia con un agregado de que le restamos la proyección. Esto genera que le restemos la contribución de nuestro vector propio anterior y poder buscar el siguiente. Para el caso de los k vectores propios más pequeños utilizaremos como base el método de la potencia inversa, el cuál consistirá en casi el mismo pseudocódigo, las modificaciones se resumen las siguientes dos cosas:

1. En la línea 21 agregar dos instrucciones para resolver el sistema $\mathbf{LU}x^{(k)} = dummy1$ con sus dos sistemas de ecuaciones, mediante sustitución hacia adelante y sustitución hacia atrás,
2. Modificar el cómo se guarda el i-ésimo vector propio en la línea 28 y que en su lugar se guarde el inverso del producto punto.

Con estas modificaciones se tendrá el mismo comportamiento y funcionamiento que el método anterior para k valores propios mayores. Para el método de Jacobi el pseudocódigo que tenemos es el siguiente

Algorithm 4 Método de Jacobi para encontrar eigenvalores y eigenvectores

Require: N, matriz, eigenvectores, tolerancia, max_iteraciones
Ensure: Eigenvalores y eigenvectores

```
function JACOBI_EIGEN(N, matriz, eigenvectores, tolerancia, max_iteraciones)
    θ, suma ← 0.0                                          ▷ Inicializamos las variables

    for iter ← 0 hasta max_iteraciones − 1 do              ▷ Comenzamos iterando

        for n ← 0 hasta N − 1 do
            for m ← 0 hasta n − 1 do
                θ ← (1/2) arctan(2 · matriz[m][n]/matriz[m][m] − matriz[n][n])   ▷ Calculamos el ángulo a rotar
                rotar(matriz, θ)                           ▷ Rotamos el ángulo sobre la matriz
                rotar(eigenvectores, θ)

        suma ← 0.0
        for i ← 0 hasta N − 1 do
            for j ← 0 hasta N − 1 do
                if i ≠ j then
                    suma ← suma + matriz[i][j] · matriz[i][j]
        if suma < tolerancia then
            Imprimir(Jacobi: Se han encontrado los eigenvalores y eigenvectores.)
            END
    Imprimir(Error Jacobi: Noe han encontrado los eigenvalores y eigenvectores.)
    return NULL
```

El psedocódigo de Jacobi consiste en explotar la propiedad de matrices de rotación operando sobre una matriz, tratando de reducir los elementos afuera de la diagonal hasta reducirlos a cero. Dentro de esta implementación, es importante destacar que se evita la búsqueda del elemento mayor fuera de la diagonal pues realmente no hace falta dado que solo hace tardado el cálculo. Esta variación se llama *Cyclic-by-Row Algorithm*.

El resultado de los pseudocódigos se enlistan a continuación.

3. Resultados

Para una tolerancia $\texttt{TOL} = 1e-16$, se realizaron pruebas en cuatro conjuntos de datos, de dimensiones 3×3, 5×5, 50×50 y uno de 125×125. A continuación se presentan los 5 valores propios más altos de cada matriz

	Método		
Tamaño Datos	Potencia	Potencia Inversa	Jacobi
3×3	11.5305	4.247786	11.538405, 8.213809, 4.247786
5×5	700.030781	0.057075	700.030781, 60.028366,8.338447, 3.745331, 0.057075
50×50	500.001543	9.998080	500.001543, 490.001431, 480.00052, 470.0001058, 460.001269
125×125	1.0000	0.000002	1.00000.000598, 0.000581, 0.000561, 0.000556

Debido a la limitaciones dentro de este documento, solo se presentan los primeros 5 valores [1], así como que se omite el los campos para los k valores propios más grandes/pequeños debido a que su información dependerá de la k sobre la cuál se tenga interés. El número de iteraciones sobre la cuál se les permitió fue en general diferente para cada uno dado que por la naturaleza de los métodos, entre mayor sea las dimensiones de la matriz más iteraciones se requerirán.

4. Conclusiones

Dentro de la variedad de métodos anteriores para determinar los valores y vectores propios, existe una diversidad de formas de cálculo, unas más costosas computacionalmente que otras aunque, en general, todas representan un costo considerable, siendo quizás, las más rápidas, los métodos de las potencias. Esto nos plantea la idea de buscar alternativas al cálculo de estas cantidades para otras aplicaciones, pues el proceso comienza a ser muy costoso en matrices de apenas 50×50.

Se deja en claro que como primer paso al momento de resolver un problema que involucre el cálculo de vectores y valores propios debe ser *buscar no hacerlo*, sobre todo cuando se tienen problemas que involucren realizar repetidamente

[1] En el caso del método de Jacobi para 125×125 se presentan los primeros 5 diferentes.

el cálculo sobre diferentes matrices. Una alternativa a este problema podría ser buscar alternativas de paralelización en diferentes puntos del código.

5. Referencias

1. Kong, Q., Siauw, T., & Bayen, A. (2020). Python programming and numerical methods: A guide for engineers and scientists. Academic Press.
2. Richard. L. Burden y J. Douglas Faires, Análisis Numérico, 7a Edición, Edito-rial Thomson Learning, 2002.
3. Samuel S M Wong, Computational Methods in Physics and Enginering, Ed. World Scientific, 3rd Edition, 1997.
4. Teukolsky, S. A., Flannery, B. P., Press, W. H., & Vetterling, W. T. (1992). Numerical recipes in C. SMR, 693(1), 59-70.
5. William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery, Numerical Recipes: The Art of Scientific Computing, 3rd Edition, Cambridge University Press, 2007

Anexo: otros pseudocódigos

A continuación se presentan los pseudocógios que complementan la sección 2 de este documento que brindan ayuda para la resolución del problema completo.

Algorithm 5 Función para rotar una matriz

Require: N, matriz, eigenvectores, θ, p, q

function ROTAR(N, matriz, eigenvectores, θ, p, q)
    $coseno \leftarrow \cos(\theta)$
    $seno \leftarrow \sin(\theta)$
    Inicializar matriz de rotación R como la identidad de tamaño $N \times N$
    Actualizar elementos de R según ángulo de rotación:
    R[p][p] $\leftarrow coseno$
    R[p][q] $\leftarrow -seno$
    R[q][p] $\leftarrow seno$
    R[q][q] $\leftarrow coseno$
    Calcular matriz transpuesta de R, denotada como R^T
    Crear matriz auxiliar $A' = R^T \cdot \text{matriz} \cdot R$
    Actualizar matriz matriz $\leftarrow A'$
    Actualizar la matriz de eigenvectores mediante la función actualizar_rotacion con R