Tarea 11 Métodos Numéricos

Carlos Giovanny Encinia González

17 de octubre 2021

Resumen

En el presente trabajo se muestran dos métodos para resolver el problema de eigenpares, el primer método que se describe encuentra solo un eigenpar, aunque el método es rápido, este no siempre converge y necesita de una condición inicial muy especifica. El otro método llamado iteración en subespacio, utiliza elementos de otros métodos que se estudiaron en tareas anteriores.

Desarrollo

Método de Rayleigh

Resolveremos el problema de eiigenpares dando un aproximación a los resultados, este método es rápido pero no se puede asegurar su convergencia:

Tenemos el problema de eigenpares, tal que:

$$(A - \lambda I)x = 0 \tag{1}$$

Entonces damos una propuesta del eigenvalor $\lambda = \sigma_0$ el cual tiene un vector asociado ϕ_0

Entonces tendríamos de la ecuación (1):

$$(A - \sigma_0 I)\phi_0 = R \tag{2}$$

Donde R es un vector residual. Y expandiendo el eigenvector, tenemos que:

$$A\phi_0 - \sigma_0\phi_0 = R \tag{3}$$

Ajora podemos suponer que de existir un valor aproximado al eigenvalor original, tal que se cumple lo siguiente:

$$\sigma_1\phi_0 - \sigma_0\phi_0 = R \tag{4}$$

si multiplicamos a la izquierda por un vector traspuesto ϕ tenemos que:

$$\phi_0^T\sigma_1\phi_0 - \phi_0^T\sigma_0\phi_0 = \phi_0^T R \tag{5}$$

Factorizando la aproximación de los eigenvectores podemos escribir que:

$$\phi_0^T(\sigma_1 - \sigma_0)\phi_0 = \phi_0^T R \tag{6}$$

puesto que la diferencia de los eigenvalores sigue siendo un escalar, podemos despejarlo tal que:

$$\sigma_1 - \sigma_0 = \frac{\phi_0^T R}{\phi_0^T \phi_0} \tag{7}$$

Y si despejamos σ_1, nos queda que:

$$\sigma_1 = \sigma_0 + \frac{\phi_0^T R}{\phi_0^T \phi_0} \tag{8}$$

Para la siguiente iteración debemos de aplicar una manera para poder calcular, la nueva aproximación del eigenvector ϕ_i. es posible aplicar la inversa.

En seguida se muestra el pseudocódigo:

Algorithm 1: Método de Rayleigh

```
1  rayleigh_eigen (matrix, m, eigenvalor, eigenvector);
   Input  : matrix: es la matriz de la que se obtendrá el eigenpar ,m: es
            la dimensión de la matriz ,eigenvalor: es la aproximacion al
            eigenvalor, este se ira actualizando ,eigenvector: es la
            aproximacion al eigenvector, se ira actualizando
   Output: eigenvalor, eigenvector
2  while condition and iterations < 1000 do
3  |  eigenvector_n ← dot(matrix, eigenvector, m, m);
4  |  j ← 0 for j: m do
5  |  |  residuo_j ← eigenvector_j − eigenvalor_0 ∗ eigenvector − j;
6  |  end
7  |  valor_t = eigenvalor;
8  |  eigenvalor = valor_t +
   |   dot(eigenvector^T, residuo, m)/dot(eigenvector^T, eigenvector, m);
9  |  if abs(eigenvalor - valor_t) < 1E-12 then
10 |  |  condition ← 0;
11 |  else
12 |  |  j ← 0 for j:m do
13 |  |  |  eigenvector_j = eigenvector_j ∗ eigenvalor;
14 |  |  end
15 |  end
16 |  iterations ← itarations + 1;
17 end
18 return(eigenvector, eigenvalor)
```

Método de iteración en Subespacio

Este algoritmo sirve para encontrar eigenvalores y eigenvectores de sistemas de ecuaciones muy grandes y se quieren encontrar m valores propios, si la matriz es de dimensión n, entonces m debe ser $m << n$.

Se A una matriz simétrica de dimensión n y Φ una matriz de dimensión nxm, que contiene eigenvectores ortonormalizados en sus columnas, entonces:

$$A_{nxn}\Phi_{nxm} = \Phi_{nxm}\Lambda_{mxm} \tag{9}$$

Donde Λ es una matriz diagonal con los eigenvalores y la matriz con eigenvectores, tiene sus columnas normalizadas.

Ahora multiplicando a la izquierda por la transpuesta de la matriz con eigenvectores tenemos que:

$$\Phi^T_{mxn} A_{nxn} \Phi_{nxm} = \Phi^T_{mxn} \Phi_{nxm} \Lambda_{mxm} \tag{10}$$

simplificando el termino de la derecha tenemos que:

$$\Phi^T_{mxn} A_{nxn} \Phi_{nxm} = \Lambda_{mxm} \tag{11}$$

Como en nuestro caso partimos de una aproximación, entonces el termino de la derecha, no sera igual a la matriz con eigenvalores, pero si dará una aproximación a esta, entonces denotamos:

$$\left[\Phi^T_{mxn}\right]^0 A_{nxn} \left[\Phi_{nxm}\right]^0 = \Lambda_{mxm} \tag{12}$$

$$\left[\Phi^T_{mxn}\right]^0 A_{nxn} \left[\Phi_{nxm}\right]^0 = B_{mxm} = \Lambda_{mxm} \tag{13}$$

Multiplicando los últimos dos términos por una matriz Q_{mxm} nos queda:

$$B_{mxm} \left[Q_{mxm}\right]^0 = \left[Q_{mxm}\right]^0 \Lambda_{mxm} \tag{14}$$

Multiplicando por la traspuesta tenemos que:

$$\left[Q^T_{mxm}\right]^0 B_{mxm} \left[Q_{mxm}\right]^0 = \left[Q^T_{mxm}\right]^0 \left[Q_{mxm}\right]^0 \Lambda_{mxm} \tag{15}$$

$$\left[Q^T_{mxm}\right]^0 B_{mxm} \left[Q_{mxm}\right]^0 = \Lambda_{mxm} \tag{16}$$

Estos eigenvectores encontrados de la matriz B nos servirán para poder encontrar una nueva matriz que contiene los eigenvectores de la matriz A. entonces tenemos que:

$$\left[\Phi_{nxm}\right]^1 = \left[\Phi_{nxm}\right]^0 \left[Q_{mxm}\right] \tag{17}$$

Ahora la siguiente pregunta es como asegurar la convergencia, y como obtener la matriz Q.

Para la matriz de eigenvectores asociada con la Matriz A, se le aplicara el método de la potencia o potencia inversa, para poder obtener los eigenvalores ya sean mayores o menores, esto nos permitirá hacer que el método converja a una solución.

Para obtener la matriz Q en el subespacio, emplearemos el método de Jacobi. Esto es aplicar las rotaciones como se vio en l atarea 10.

Debido a que los programas como método de la inversa da el conjunto de eigenvectores en una matriz donde las filas son los eigenvectores, es necesario utilizar propiedades de las matrices traspuestas para llegar al resultado. Dicho lo anterior, en seguida muestro el pseudocódigo.

Algorithm 2: Método de iteración en subespacio

```
1  subespacio (matrix, m, number_eigen);
   Input  : matrix: es la matriz de la que se obtendrá el eigenpar ,m: es
            la dimension de la matriz number_eigen: es el numero de
            eigenpares que se quieren encontrar
   Output: eigen_valores, eigenvectores
2  condition ← 1 eigenvalues ← empty();
3  while condition and iteration < 2000 do
4      LU ← copy_matrix(matrix, m, m);
5      eigen_traspuesta ←
         eigen_menores(LU, m, m, number, evalues, number_eigen, 1, eigen_traspuesta);
6      eigen_traspuesta ←
         ortononormalizar(eigen_traspuesta, number_eigen, m);
7      eigen_temp ←
         dot(eigen_traspuesta, matrix, number_eigen, m, m, m);
8      eigenvectores ← traspuesta(eigen_traspuesta, number_eigen, m);
9      eigen_valores ←
         dot(eigen_temp, eigenvectores, number_eigen, m, m, number_eigen);
10     Q ← eigen_jacobi(eigen_valores, number_eigen);
11     i ← 0;
12     for i: number_eigen do
13         j ← 0;
14         for j:number_eigen do
15             if i ≠ j then
16                 if abs(eigen_valores_{i,j}) < 1E − 12 then
17                     condition ← 0
18                 else
19                     condition ← condition + 1
20                 end
21             end
22         end
23     end
24     if not condition or iteration ≥ 1000 then
25         break;
26     end
27     Q_t ← traspuesta(Q, number_eigen, number_eigen);
28     Qtemp ←
         dot_matrix(Q_t, eigen_traspuesta, number_eigen, number_eigen, number_eigen, m);
29     i ← 0;
30     for i:number_eigen do
31         j ← 0;
32         for j :m do
33             eigen_traspuesta_{i,j} ← Qtemp_{i,j}
34         end
35     end
36     iteration ← iteration + 1
37 end
38 return(eigenvectores, eigenvalores)
```

Resultados

En seguida se muestran los resultados obtenidos mediante el método de Rayleigh:

Figure 1: Matriz de 3x3

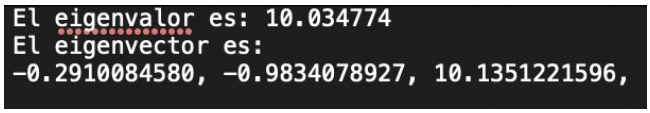
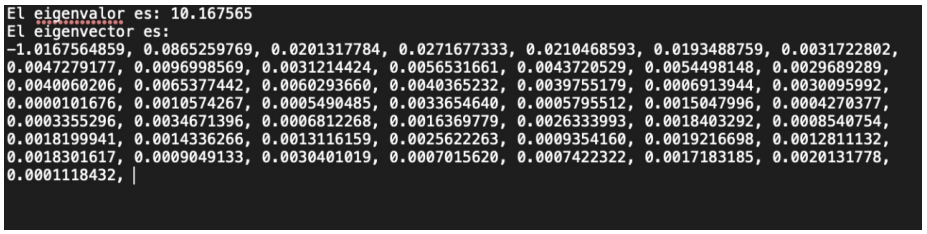


Figure 2: Matriz de 50x50



En seguida se muestran los resultados obtenidos por el método de iteración en el subespacio. Solo se mostraran los primeros elementos de cada eigenvector, ya que son muy grandes, los resultados se pueden obtener en la carpeta /Resultados.

Figure 3: Matriz de 3x3

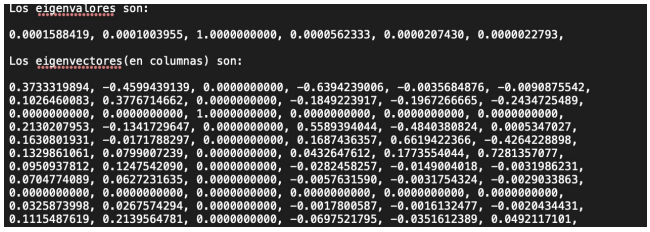
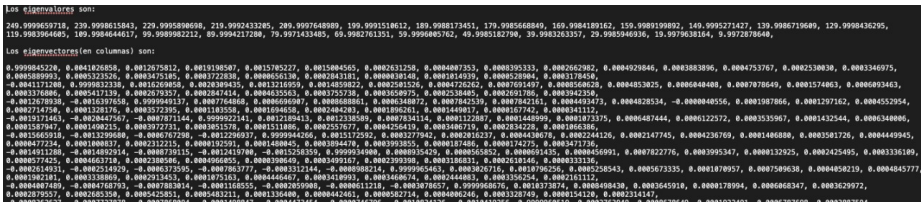
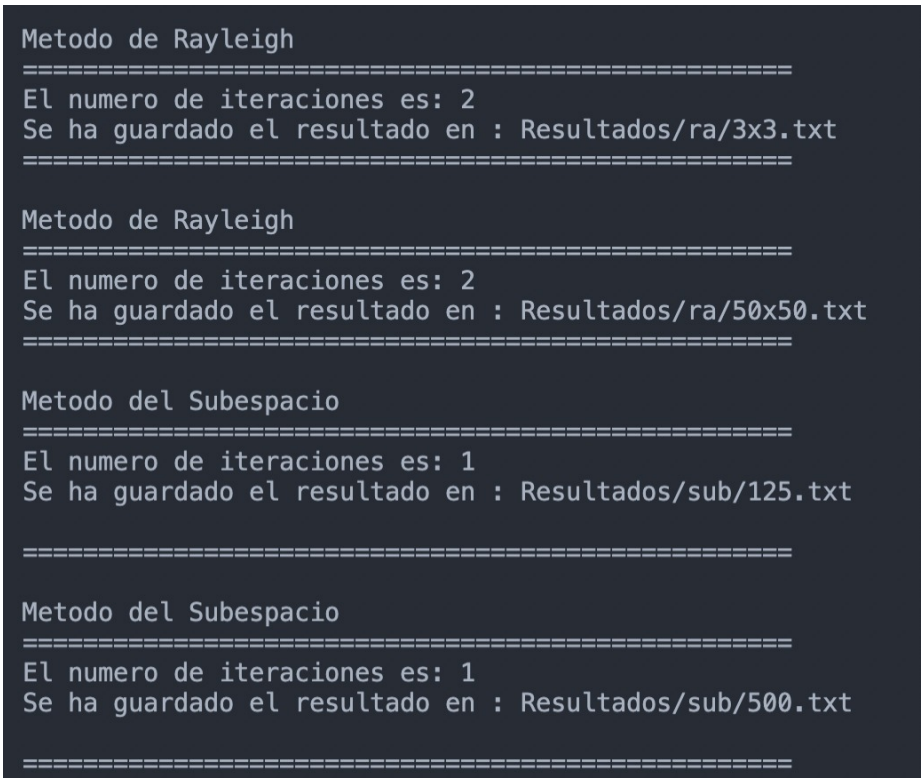


Figure 4: Matriz de 50x50



En seguida se muestra lo que aparece en la terminal, podemos observar que menciona donde se a generado el resultado, y el numero de operaciones realizadas.

Figure 5: Resultado en terminal



Conclusión

El método de Rayleigh a mi parecer es un método que nos podría servir solamente para poder afinar resultados de otros métodos, es rápida la convergencia si se le dan vectores iniciales adecuados. En cambio el método de iteración en el subespacio, da resultados mas precisos, y converge de manera rápida, es de

los mejores métodos para encontrar eigenpares que he programado hasta el momento, claro que no se podrá aplicar siempre, además que existe la restricción de que las matrices tienen que ser simétricas. Los algoritmos tienen un error absoluto del 1E-12, con respecto a los criterios que se toman para detener el algoritmo, por ejemplo, para el método de la iteración en el subespacio, se acaba el algoritmo cuando los elementos fuera de la diagonal son menores a 1E-12. El numero de iteraciones aparece en terminal cada vez que se llama a un método, pero en general ambos métodos necesitan de pocas iteraciones para poder encontrar el resultado. Es importante mencionar que el numero de iteraciones en el segundo método, en realidad engloba mas cálculos debido al método de Jacobi que se aplica para encontrar la solución.