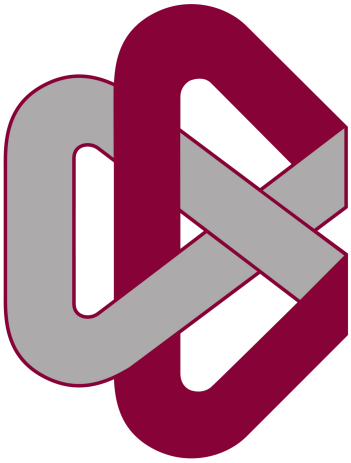
Tarea 8



CIMAT

Nombre:	**Jairo Saul Diaz Soto**
Maestría:	Ciencias Computacionales
Modulo:	Métodos Numéricos
Instructor:	Dr. Salvador Botello Rionda
Fecha de entrega:	2023 - 10 - 08

1. Introducción

Dentro del ámbito del manejo de matrices y realizar cálculos con estas, llámese resolver sistemas de ecuaciones, problemas de eigenvalores, etc, existen diferentes metodologías, más o menos complejas, con más o menos restricciones y que se utilizan para resolver de forma exacta o una aproximación a través de un método iterativo.

A continuación se despliega una lista sumamnete util para trabajr estos problemas.

2. Factorización QR

Bien, este método esta diseñado para descomponer una matriz dada A en dos factores, una matriz Q que se caracteriza por ser una matriz ortonormal y una matriz R, la cual es una matriz triangular superior.

El proceso para realizar esto es lo siguiente se debe proponer una matriz Q para la cual sus columnas sean ortonormales, por ejemplo, la matriz identidad, para la cual la factorización sería tal que

$$A = QR \qquad A = R$$

A partir de aquí, se busca una matriz de rotación la cual anule los elementos fuera debajo de la diagonal de R, esto dado que se propone que esta debería ser una matriz triangular superior, entonces, dadas las propiedades de la matriz Q, se realizarían tantas rotaciones como fuere necesario y la matriz Q permanecería como una matriz ortonormal.

Se muestra un algoritmo:

Algorithm 1: Factorización QR por el Método de Gram-Schmidt

Data: Matriz **A** de tamaño sz × sz
Result: Matrices Q y R que representan la factorización QR de **A**

```
Q ← Matriz de ceros de tamaño sz × sz // Inicializar matriz Q
R ← Matriz de ceros de tamaño sz × sz // Inicializar matriz R
for j ← 1 to sz do
    Q[:, j] ← A[:, j] // Copiar la columna j de A a Q
    for i ← 1 to j − 1 do
        R[i, j] ← (Q[:,i]^T · A[:,j]) / (Q[:,i]^T · Q[:,i]) // Calcular el coeficiente R
        Q[:, j] ← Q[:, j] − R[i, j] · Q[:, i] // Actualizar Q
    R[j, j] ← ‖Q[:, j]‖ // Calcular la norma de la columna j de Q como el elemento
       diagonal de R
    Q[:, j] ← Q[:,j] / R[j,j] // Normalizar la columna j de Q
return Q y R;
```

3. Método del gradiente conjugado

Este método es ampliamente utilizado para resolver sistemas de ecuaciones, dado que es un método iterativo, este es un método aproximado, sin embargo, funciona especialmente bien para matrices muy grandes y las cuales son densas, además se pide que estas matrices sean simétricas y definidas positivas.

Derivado de lo anterior, este método trabaja con un vector inicial, se basa en determinar un residuo que se obtiene de lo siguiente

$$\mathbf{r}_{(k)} = \mathbf{b} - A\mathbf{x}_{(k)}$$

donde claramente, el residuo es $\mathbf{0}$ si $\mathbf{x}_{(k)}$ es el vector que satisface el sistema de ecuaciones a resolver $A\mathbf{x} = \mathbf{b}$.

A continuación se muestra el algoritmo

Algorithm 2: Método del Gradiente Conjugado

Data: Matriz **A**, vector **b**, vector inicial **x**, número máximo de iteraciones **maxIter**, tolerancia **tolerance**

Result: Solución aproximada **x**

```
r = b − Ax // Calcular el residuo inicial
p = r // La dirección inicial es el residuo
alpha = 0 // Inicializar el tamaño de paso
for i = 1 to maxIter do
    alpha = (r^T r) / (p^T Ap) // Calcular el tamaño de paso
    x = x + alphap // Actualizar la solución
    r = r − alphaAp // Actualizar el residuo
    if ||r|| < tolerance then
        break // Criterio de convergencia
    β = (r^T r) / (r^T r_anterior) // Calcular el coeficiente de conjugación
    p = r + βp // Actualizar la dirección de búsqueda
return x
```

3.1. Método del gradiente conjugado: Precondicionador de Jacobi

Para acelerar la convergencia del método, mejorar la estabilidad o ahorrar recursos, se suele utilizar precondicionadores, uno de ellos es el precondiconador de Jacobi, el cual propone una matriz M la cual se denomina como la pseudoinversa de la matriz original. Se propone que si $A = L+D+U$, donde A es la matriz del problema de sistemas de ecuaciones a resolver, es decir, $A\mathbf{x} = \mathbf{b}$, L una matriz triangular inferior, D una matriz diagonal y U una matriz triangular superior, entonces

$$M^{-1} = D^{-1} \tag{1}$$

ahora, este factor, se utilizara para mejorar, pues se emplea en la actualización de los pasos, de la dirección de la búsqueda.

A continuación el algoritmo

4. Método de Rayleigh

Ahora, para problenmas de eigenvalores, el método de Rayleigh, funciona para determinar el eigenvalor dominante de una matriz simetrica y su eigenvector correspondiente, a traves de un método iterativo, es decir, una aproximación, este se basa en el denominado cociente de Rayleigh, el cual es el siguiente

$$r = \frac{\mathbf{x}^T A\mathbf{x}}{\mathbf{x}^T\mathbf{x}} \tag{2}$$

para el caso especial donde el vector x es un eigenvector de la matriz A, entonces r es el eigenvector correspondiente.

A continuación el algoritmo

Algorithm 3: Método del Gradiente Conjugado Precondicionado

Data: Matriz $\mathbf{A}$, vector $\mathbf{b}$, vector inicial $\mathbf{x}$, número máximo de iteraciones maxIter, tolerancia tolerance, precondicionador $\mathbf{M}$

Result: Solución aproximada $\mathbf{x}$

$\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x}$;
$\mathbf{z} = \mathbf{M}^{-1}\mathbf{r}$;
$\mathbf{p} = \mathbf{z}$;
$\alpha = 0$;
for $i = 1$ **to** *maxIter* **do**
    $\alpha = \frac{\mathbf{z}^T\mathbf{r}}{\mathbf{p}^T\mathbf{A}\mathbf{p}}$;
    $\mathbf{x} = \mathbf{x} + \alpha\mathbf{p}$;
    $\mathbf{r} = \mathbf{r} - \alpha\mathbf{A}\mathbf{p}$;
    $\mathbf{z} = \mathbf{M}^{-1}\mathbf{r}$;
    if $\|\mathbf{r}\| <$ *tolerance* **then**
        break;
    $\beta = \frac{\mathbf{z}^T\mathbf{r}}{\mathbf{z}^T\mathbf{r}_{\text{anterior}}}$;
    $\mathbf{p} = \mathbf{z} + \beta\mathbf{p}$;
return $\mathbf{x}$;

Algorithm 4: Método de Rayleigh para encontrar el eigenvalor principal y el vector propio correspondiente

Data: Matriz simétrica real A, vector inicial v, tolerancia tolerance

Result: Eigenvalor principal aproximado lambda y vector propio correspondiente v

lambda $= 0$; `// Aproximación inicial del eigenvalor`
converged = **false** ; `// Criterio de convergencia`
while *not* *converged* **do**
    $\mathsf{v} = \frac{\mathsf{A}\mathsf{v}}{\|\mathsf{A}\mathsf{v}\|}$; `// Actualizar el vector propio`
    $\text{lambda}_{\text{new}} = \mathsf{v}^T\mathsf{A}\mathsf{v}$; `// Actualizar la aproximación del eigenvalor`
    if $|\text{lambda}_{new} - \text{lambda}| <$ tolerance **then**
        converged = **true** ; `// Criterio de convergencia alcanzado`
    lambda $= \text{lambda}_{\text{new}}$
return lambda, v ; `// Eigenvalor principal aproximado y vector propio correspondiente`

5. Método de iteración de subespacio

Siguiendo en la línea para determinar problemas de eigenvalores, el método de iteración ayuda mejorando ciertos criterios para estabilizar o en su concecuencia mejorar la convergencia a la hora de obtener estos valores.

Este se ayuda de que el método de la potencia se pueden obtener vectores ortogonales entre sí y mejorar la convergencia del método de Jacobi.

A continuación el algoritmo

Algorithm 5: Aproximación de eigenvalores por Iteración de Subespacio

Data: Matriz real matriz, req, sz, TOL, NMax
Result: Estructura Eigenv con eigenvalores y eigenvectores

```
matriz = matriz req = req sz = sz TOL = TOL NMax = NMax
λ = 0 converged = False count = 0 x0 = (req, sz)
for i = 0 to req − 1 do
    for j = 0 to sz − 1 do
        x0[i][j] = 1,0
tphi = (x0, req, sz) eigenp = (matriz, x0, req, sz, TOL, 1)
for i = 0 to req − 1 do
    tphi[i] = eigenp[i].vector
while count < NMax do
    eigenp = (matriz, tphi, req, sz, TOL, 1)
    for i = 0 to req − 1 do
        tphi[i] = eigenp[i].vector
    Phi = (tphi, req, sz) prod = (tphi, matriz, req, sz, sz) Q = (prod, Phi, req, sz, req)
    for i = 0 to req − 1 do
        dq1[i] = Q[i][i]
    max = 0
    for i = 0 to req − 1 do
        for j = i + 1 to req − 1 do
            if |Q[i][j]| > |max| then
                max = Q[i][j]
    if |max| < TOL then
        converged = True break
    eigen_jacobi(Q, R, req, TOL, 1)
    for i = 0 to req − 1 do
        dq2[i] = Q[i][i]
    res = (dq2v, dq2, req)
    if (dq2, dq2v, req) < TOL then
        dq1 = dq2 break
    a1 = (Phi, R, sz, req, req)
    for i = 0 to sz − 1 do
        for j = 0 to req − 1 do
            Phi[i][j] = a1[i][j]
    free(tphi[0]) free(tphi) tphi = (Phi, sz, req)
    for i = 0 to req − 1 do
        dq2v[i] = dq2[i]
    count = count + 1
printf(Ïteraciones:
for i = 0 to req − 1 do
    sol[i].λ = dq1[i] sol[i].vector = Phi[i]
return sol
```

6. Resultados

A continuación, se presentan resultados de lo antes expuesto

```
 % ./qr Eigen_3x3.txt
La matriz de entrada es :
3       -0.1    -0.2
-0.1    7       -0.3
-0.2    -0.3    10
La matriz Q es la siguiente:
0.997234        0.0302282       0.0679052
-0.0332411      0.998492        0.0436873
-0.0664822      -0.0458237      0.996735
La matriz R es la siguiente:
3.00832 -0.312467       -0.854297
-1.9082e-17     7.00017 -0.76383
-5.55112e-17    5.55112e-17     9.94066
```

Figura 1: Obtención de la factorización QR.

```
 % ./grad Eigen_3x3.txt V_sys_3x1.txt
La matriz de entrada es:
3       -0.1    -0.2
-0.1    7       -0.3
-0.2    -0.3    10
El vector de entrada es:
7.850000        -19.300000      71.400000
El metodo convergio tras 13 iteraciones.
La solucion es:
3.010866        -2.408673       7.128135
```

Figura 2: Resolución de un sistema de ecuaciones por el método del gradiente conjugado.

7. Conclusiones

Para el método de QR, existe poco que concluir, puesto que aun no existe finalidad para realizar dicha factorización, aunque parece ser un método estable para matrices simétricas.

Para el método del gradiente conjugado y el precondicionado, se aprecia una mejora sustancial a la hora de la aplicación, sin embargo, el problema del precondicionado es que no asegura la convergencia, mientras que el primero sí.

```
 % ./jacgrad Eigen_3x3.txt V_sys_3x1.txt
La matriz de entrada es:
3       -0.1    -0.2
-0.1    7       -0.3
-0.2    -0.3    10
El vector de entrada es:
7.850000        -19.300000      71.400000
El metodo convergio tras 4 iteraciones.
La solucion es:
3.011577        -2.408636       7.127972
```

Figura 3: Resolución de un sistema de ecuaciones por el método del gradiente conjugado con el precondicionado de Jacobi.

```
 % ./ray Eigen_3x3.txt
La matriz de entrada es:
3       -0.1    -0.2
-0.1    7       -0.3
-0.2    -0.3    10
El metodo convergio tras 6 iteraciones.
La solucion es: 6.998597
```

Figura 4: Obtención de un eigenvalor y su respectivo eigenvector con el método de Rayleigh.

```
 % ./sub Eigen_50x50.txt 7
(02) Se encontro la solucion tras 7 iteraciones.
La solucion es:
El eigenvalor 1 es: 9.999341
El eigenvalor 2 es: 480.448590
El eigenvalor 3 es: 459.996301
El eigenvalor 4 es: 410.000321
El eigenvalor 5 es: 350.000240
El eigenvalor 6 es: 279.999734
El eigenvalor 7 es: 200.000384
```

Figura 5: Obtención de un 7 pares eigenvalor - eigenvector con el método de iteración de subespacio.

Para el método de Rayleigh, unicamente se debe tener cuidado en la inicialización, puesto que dependiendo de esta se obtienen distintos resultados.

En el método de subespacios, aparece una cuestión extraña, puesto que para valores menores

a 7, es decir, cuando se solicitan 6 o menos eigenvalores, el método no termina por converger, sin embargo, para valores más grandes tiene buenas aproximaciones.

8. Referencias

[1] Burden, R.L., Faires, J.D. and Burden, A.M. 2015. *Numerical analysis.* Cengage learning.
[2] Quarteroni, A., Sacco, R. and Saleri, F. 1999. *Numerical Mathematics.* Springer.