Proyecto 2 - Valores propios

Table of Contents

Planteamiento

El propósito de este proyecto es calcular todos los valores propios, y opcionalmente, los vectores propios correspondientes, de una matriz cuadrada real. Se debe implementar una función llamada `qrAlgorithm` que realice esta tarea utilizando una función llamada `DescomposicionQR` que utilice el proceso de ortonormalización de Gram-Schmidt vía eliminación de Gauss.

La función `DescomposicionQR` debe realizar la descomposición QR de la matriz de entrada, es decir, divide la matriz original en dos partes fundamentales: una matriz ortogonal, donde sus columnas representan vectores ortogonales unitarios, y una matriz triangular superior. El proceso de Gram-Schmidt se utilizará como base para la implementación de la función QR, ya que permite obtener vectores ortogonales a partir de un conjunto inicial de vectores linealmente independientes. La eliminación de Gauss se aplicará para llevar la matriz original a una forma triangular superior.

La función `qrAlgorithm` iniciará el proceso de cálculo de los valores propios de la matriz original. Este proceso se lleva a cabo mediante un conjunto de iteraciones utilizando el método QR iterativo, que también es conocido como el algoritmo QR. Además, si es requerido, será posible calcular los vectores propios correspondientes en esta etapa del proceso.

Modelo matemático

Para calcular los valores propios de una matriz mediante la implementación del Algoritmo QR, es esencial entender la fundamentación matemática detrás de la Factorización QR y el Proceso de Gram-Schmidt. A continuación se presenta un modelo matemático que describe estos conceptos:

Factorización QR

Para una matriz A de dimensiones $m \times n$ con columnas linealmente independientes, existe una matriz ortogonal Q de dimensiones $m \times m$ y una matriz R triangular superior de dimensiones $n \times n$, tal que $A=QR$. Hay dos formas de computar esta descomposición QR, a través del proceso de transformaciones de Householder o a través del proceso de Gram-Schmidt. En este proyecto nos enfocaremos en el proceso de Gram-Schmidt.

Proceso de Gram-Schmidt

El Proceso de Gram-Schmidt es un método para ortogonalizar un conjunto de vectores en un espacio vectorial con producto interno, que es un paso esencial para la factorización QR.

Dado un conjunto de vectores linealmente independientes $\{a_1, a_2, ..., a_n\}$, el proceso de Gram-Schmidt genera un conjunto ortogonal $\{q_1, q_2, ..., q_n\}$ de la siguiente manera:

$$q_1 = \frac{a_1}{||a_1||}$$

Para $i = 2, 3, \ldots, n$:

$$u_i = a_i - \sum_{j=1}^{i-1} \langle q_j, a_i \rangle \, q_j$$

$$q_i = \frac{u_i}{\| u_i \|}$$

Donde $\langle q_j, a_i \rangle$ denota el producto interno.

Algoritmo QR para valores propios:

El Algoritmo QR es un método iterativo para encontrar los valores propios de una matriz. En cada iteración, se realiza una factorización QR, seguida de una multiplicación en el orden inverso para obtener una nueva matriz que converge hacia una forma triangular, cuyos elementos diagonales son los valores propios.

Dada una matriz *A*, se ejecutan las siguientes etapas hasta que la matriz *A* converja:

1. Factorización QR: *A=QR*
2. Actualización de la matriz: *A=RQ*

Resultados

```
A = [4, -2, 1; -2, 3, 1; 1, 1, 2];
maxIter = 100;
rTol = 1e-6;
opcionVector = true;

[valP, vectP] = qrAlgorithm(A, maxIter, rTol, opcionVector);

if opcionVector
    disp('Vectores Propios:');
    disp(vectP);
end
```

```
Vectores Propios:
    0.7985    0.3472   -0.4918
   -0.5994    0.5341   -0.5962
    0.0557    0.7708    0.6346
```

```
disp('Valores Propios:');
```

```
Valores Propios:
```

```
disp(valP);
```

```
    5.5712
    3.1433
    0.2855
```

```
[a,b]=eig(A)
```

```
a = 3×3
    0.4918    0.3472   -0.7985
    0.5962    0.5341    0.5994
   -0.6346    0.7708   -0.0557
b = 3×3
    0.2855         0         0
         0    3.1433         0
         0         0    5.5712
```

```
%------------------------------------------------------------------------------
-------
D = diag([1, 2, 3, 4, 5]);

[valP, vectP] = qrAlgorithm(D, maxIter, rTol, opcionVector);

if opcionVector
    disp('Vectores Propios:');
    disp(vectP);
end
```

```
Vectores Propios:
     1     0     0     0     0
     0     1     0     0     0
     0     0     1     0     0
     0     0     0     1     0
     0     0     0     0     1
```

```
disp('Valores Propios:');
```

```
Valores Propios:
```

```
disp(valP);
```

```
     1
     2
     3
     4
     5
```

```
[a,b]=eig(D)
```

```
a = 5×5
     1     0     0     0     0
     0     1     0     0     0
```

```
     0     0     1     0     0
     0     0     0     1     0
     0     0     0     0     1
b = 5×5
     1     0     0     0     0
     0     2     0     0     0
     0     0     3     0     0
     0     0     0     4     0
     0     0     0     0     5
```

```
%-----------------------------------------------------------------------------
-------
E = rand(4);
E = E + E';

[valP, vectP] = qrAlgorithm(E, maxIter, rTol, opcionVector);
if opcionVector
    disp('Vectores Propios:');
    disp(vectP);
end
```

```
Vectores Propios:
    0.4875    0.7175    0.2288   -0.4417
    0.4820   -0.6664    0.4833   -0.3002
    0.4985   -0.1719   -0.8342   -0.1611
    0.5305    0.1075    0.1346    0.8300
```

```
disp('Valores Propios:');
```

```
Valores Propios:
```

```
disp(valP);
```

```
    4.2709
   -1.6885
   -0.9351
    0.4990
```

```
[a,b]=eig(E)
```

```
a = 4×4
    0.7175   -0.2288    0.4417    0.4875
   -0.6664   -0.4833    0.3002    0.4820
   -0.1719    0.8342    0.1611    0.4985
    0.1075   -0.1346   -0.8300    0.5305
b = 4×4
   -1.6885         0         0         0
         0   -0.9351         0         0
         0         0    0.4990         0
         0         0         0    4.2709
```

```
%-----------------------------------------------------------------------------
-------
F = complex(rand(3), rand(3));
```

```
[valP, vectP] = qrAlgorithm(F, maxIter, rTol, opcionVector);
if opcionVector
    disp('Vectores Propios:');
    disp(vectP);
end
```

```
Vectores Propios:
  -0.5347 - 0.1940i  -0.7155 + 0.3146i   0.2556 - 0.0070i
  -0.5068 - 0.3336i   0.4894 + 0.1147i  -0.0677 + 0.6121i
  -0.5426 - 0.1179i   0.1394 - 0.3419i  -0.4303 - 0.6085i
```

```
disp('Valores Propios:');
```

```
Valores Propios:
```

```
disp(valP);
```

```
   1.3732 + 1.5916i
  -0.5588 - 0.0605i
   0.1659 - 0.4696i
```

```
[a,b]=eig(F)
```

```
a = 3×3 complex
   0.5534 - 0.1319i   0.7136 + 0.0000i   0.6947 + 0.0000i
   0.6067 + 0.0000i  -0.4987 - 0.1897i  -0.2764 + 0.5533i
   0.5180 - 0.1998i  -0.2764 + 0.3600i  -0.0955 - 0.3545i
b = 3×3 complex
   1.3732 + 1.5916i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i  -0.5588 - 0.0605i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.1659 - 0.4696i
```

Conclusiones

Este proyecto presenta una manera eficaz y precisa para determinar los valores propios y los vectores propios de una matriz cuadrada real. Este método se basa en la utilización de la descomposición QR, que incorpora tanto el proceso de Gram-Schmidt como la eliminación de Gauss. El método de descomposición QR es apreciado por su estabilidad numérica, versatilidad, eficiencia, y su capacidad para manejar una amplia gama de matrices.

Implementación

```
function [Q,R] = DescomposicionQR(A)
    n = size(A,1);
    Q = zeros(n);
    R = zeros(n);
    for j = 1:n
        v = A(:,j);
        for i = 1:j-1
            R(i, j) = Q(:, i)' * A(:, j);
```

```
            v = v - R(i, j) * Q(:, i);
        end
        R(j, j) = norm(v);
        Q(:, j) = v / R(j, j);
    end
end

function [valP,vectP] = qrAlgorithm(A, maxIter, rTol, opcionVector)
    if nargin < 4
        opcionVector = false;
    end
    n = size(A,1);
    if opcionVector
        vectP = eye(n);
    else
        vectP = [];
    end
    condicion = true;
    i = 0;
    while condicion
        [Q, R] = DescomposicionQR(A);
        A = R * Q;
        if opcionVector
            vectP = vectP * Q;
        end
        condicion = norm(diag(A, -1)) > rTol && i < maxIter;
    end
    valP = diag(A);
end
```

Referencias

- Golub, G. H., & Van Loan, C. F. (2013). Matrix computations (Vol. 3). JHU Press.
- Trefethen, L. N., & Bau III, D. (1997). Numerical linear algebra (Vol. 50). SIAM.
- Ascher, U. M., & Greif, C. (2011). A first course in numerical methods. The Universityof British Columbia.