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

Rafael Alejandro García Ramírez

9 de octubre de 2023

1. Introducción

1.1. Gradiente Conjugado

El método de gradiente conjugado es un método iterativo para resolver un sistema de ecuaciones lineales de la forma $\mathbb{A}\mathbf{x} = \mathbf{b}$ cuando $\mathbb{A}$ es simétrica y definida positiva, es decir que $\mathbb{A} = \mathbb{A}^T$ y $\mathbf{x}^T\mathbb{A}\mathbf{x} > 0$ para cualquier vector no nulo $\mathbf{x}$. Supongamos que nos encontramos en realizando desplazamientos del tipo $\mathbf{x}^* = \mathbf{u} - a\mathbf{v}$, es decir definimos un nuevo vector como la diferencia de dos vectores.

Con esto en mente, supongamos que tenemos un vector $\mathbf{P}^k$ que apunta a algún lugar que no se ha visitado, es decir que contiene información de un estado nuevo a nuestro vector $\mathbf{x}$ y sea una constante α^k que define el paso (o más bien, el peso que este vector previo tiene sobre el nuevo) podemos definir cada nuevo vector como

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{P}^{(k)} \tag{1}$$

Así $\mathbf{x}^{(k)}$ contiene la información de nuestro estado (solución) previo o nuestro estado actual. Dentro de cada paso para un nuevo vector podemos verificar la condición de convergencia de $\mathbf{b} - \mathbb{A}\mathbf{x} = 0$, pero hasta que no se cumpla esta condición definimos un vector de residuo $\mathbf{r}^{(k)}$. Nuestro interés ahora es entonces buscar expresiones que nos permitan actualizar $\alpha^{(k)}$ y $\mathbf{P}^{(k)}$ en la ecuación (1). Si tomamos dos momentos de k

$$\mathbf{r}^{(k)} = \mathbf{b} - \mathbb{A}\mathbf{x}^{(k)}$$
$$\mathbf{r}^{(k+1)} = \mathbf{b} - \mathbb{A}\mathbf{x}^{(k+1)}$$

Uniendo estas dos ecuaciones tenemos

$$\mathbf{r}^{(k+1)} - \mathbf{r}^{(k)} = -\mathbb{A}(\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)})$$

Despejando la ecuación (1) y utilizándola en la ecuación anterior tenemos

$$\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - \alpha^{(k)}\mathbb{A}\mathbf{P}^{(k)}$$

Ahora, supongamos que eventualmente en algún momento el residuo $\mathbf{r}^{(k+1)} = 0$ así tenemos la expresión que el momento anterior, por la expresión anterior, es $\mathbf{r}^{(k)} = \alpha^{(k)}\mathbb{A}\mathbf{P}^{(k)}$, si multiplicamos esta expresión por ${\mathbf{P}^{(k)}}^T$ por la izquierda, podemos encontrar la forma

$$\alpha^{(k)} = \frac{{\mathbf{P}^{(k)}}^T\mathbf{r}^{(k)}}{{\mathbf{P}^{(k)}}^T\mathbb{A}\mathbf{P}^{(k)}} \tag{2}$$

Ahora solo nos falta encontrar una expresión para actualizar $\mathbf{P}^{(k)}$, para ello proponemos una expresión similar para la actualización $\mathbf{x}$ tal que $\mathbf{P}^{(k+1)} = \mathbf{r}^{(k+1)} - \beta^{(k)}\mathbf{P}^{(k)}$, realizando los mismos pasos para determinar $\alpha^{(k)}$ podemos determinar esta constante como

$$\beta^{(k)} = \frac{{\mathbf{P}^{(k)}}^T\mathbf{r}^{(k+1)}}{{\mathbf{P}^{(k)}}^T\mathbf{P}^{(k)}} \tag{3}$$

Para este caso establecemos la condición ${\mathbf{P}^{(k)}}^T\mathbf{P}^{(k+1)} = 0$

1.2. Gradiente Conjugado Precondicionado

Para el método de gradiente conjugado precondicionado (también llamado Precondicionador de Jacobi), se busca acelerar la convergencia del método de gradiente conjugado, esto mediante una matriz precondicionada $\mathbb{M}$ simétrica y definida positiva tal que $\mathbb{M}^{-1}\mathbb{A}$ tenga mejor número de condición que $\mathbb{A}$. Para ello hacemos que $\mathbb{A} = \mathbb{L} + \mathbb{D} + \mathbb{U}$ y que $\mathbb{M}^{-1} = \mathbb{D}^{-1}$. Tomamos un estado $y^{(k)}$ tal que $\mathbf{P}^{(0)} = \mathbf{y}^{(0)} = \mathbb{M}^{-1}\mathbf{r}^{(0)}$. Con esto en cuenta podemos seguir exactamente el mismo procedimiento para el gradiente conjugado para determinar las relaciones ahora con la forma $\mathbf{y}^{(\cdot)}$

$$\alpha^{(k)} = \frac{{\mathbf{y}^{(k)}}^T\mathbf{r}^{(k)}}{{\mathbf{P}^{(k)}}^T\mathbb{A}\mathbf{P}^{(k)}} \quad , \quad \mathbf{y}^{(k+1)} = \mathbb{M}^{-1}\mathbf{y}^{(k)} \tag{4}$$

$$\beta^{(k)} = \frac{\mathbf{y}^{(k+1)^T}\mathbf{P}^{(k)}}{\mathbf{y}^{(k)^T}\mathbf{P}^{(k)}} \quad , \quad \mathbf{P}^{(k+1)} = \mathbf{y}^{(k+1)} - \beta^{(k)}\mathbf{P}^{(k)} \tag{5}$$

1.3. Factorización QR

La factorización QR consiste en buscar una factorización de la matriz $\mathbb{A} = \mathbb{Q}\mathbb{R}$ tal que $\mathbb{Q}$ sea una matriz ortogonal, es decir que $\mathbb{Q}\mathbb{Q}^T = \mathbb{I}$, y que $\mathbb{R}$ sea una matriz triangular superior.

Con estas propiedades podemos ver que si e_i es el i-ésimo vector columna de $\mathbb{A}$ podemos obtener mediante Gram-Schmidt una base ortonormal asociada $\{u_1, u_2, \cdots, u_n\}$ dado que $\mathbb{A}$ es invertible; la matriz que constrituyen estos vectores ortonormales será nuestra matriz $\mathbb{Q}$. Dado que la matriz Q es ortogonal, esto es $\mathbb{Q}^{-1} = \mathbb{Q}^T$ por lo tanto si despejamos

$$\mathbb{A} = \mathbb{Q}\mathbb{R} \Rightarrow \mathbb{R} = \mathbb{Q}^{-1}\mathbb{A} = \mathbb{Q}^T\mathbb{A}$$

Por lo tanto el método consiste en obtener una base ortonormal de $\mathbb{A}$ y llamarla $\mathbb{Q}$, y después solamente resolver el sistema $\mathbb{R} = \mathbb{Q}^T\mathbb{A}$.

1.4. Método de Rayleigh

El método de Rayleigh es un método para obtener lo valores propios de una matriz. Para ello definimos el escalar inicial $\sigma^{(0)}$ y una suposición inicial $\phi^{(0)}$ y así tenemos

$$(\mathbb{A} - \sigma^{(0)}\mathbb{I})\phi^{(0)} = r^{(0)}$$

De este modo buscamos un $\sigma^{(k)}$ tal que minimice este residuo $r^{(\cdot)}$. Si tomamos esta expresión y la multiplicamos por la matriz $\phi^{(k)^T}$ por ambos lados podemos llegar a la expresión

$$\sigma^{(k)} = \frac{\phi^{(k)^T}\mathbb{A}}{\phi^{(k)^T}\phi^{(k)^T}}$$

Por lo que uno puede obtener el valor propio más pequeño y su vector propio asociado.

1.5. Iteración Subespacio

El método de iteración de subespacio es un método bastante *"artesanal"*, dado que existen varias formas sobre las cuales abordarse, pero comparten el hecho de resolver a partes sistemas dentro de un subespacio de la matriz. En general se busca resolver el problema de

$$\mathbb{A}\mathbf{x} - \lambda\mathbf{x} = 0$$

Para lo cuál se calcularán los n valores propios con sus vectores propios asociados. Algunas formas de entrar a este subespacio incluyen utilizar le método de factorización de $\mathbb{Q}\mathbb{R}$ y luego realizar el método de Rayleigh[1], pero se utilizará aquí la generalización clásica utilizando el método de la potencia añadiéndole el método de Jacobi para valores y vectores propios. El método de subespacio a pesar de ser más lento, puede ser modificado lo suficiente para que resulte más rápido que otros métodos.

Para ello, realizaremos en secuencia una iteración del método de la potencia sobre nuestra matriz, para después realizar una iteración sobre con el método de Jacobi, e iteramos sobre estos dos pasos hasta que la matriz de valores propios sea diagonal o la matriz de vectores propios sea la identidad.

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:

2.1. Pseudocódigo Gradiente Conjugado

El método del gradiente conjugado se puede ver de la siguiente forma

[1] En lo general, resulta una alternativa para los métodos basados en Krylov.

Algorithm 1 Método del Gradiente Conjugado

Require: N, tolerancia, matriz, vector, b

function GradienteConjugado(N, tolerancia, matriz, vector, b)

  $iteraciones \leftarrow 2 \cdot N$
  $ak, bk \leftarrow 0$
  · Reservar memoria para $Ax, r, r_n, p, paso_{Apk}$
  · Calcular $Ax = \mathbf{matriz} \cdot \mathbf{vector}$
  · Calcular $r = \mathbf{b} - Ax$
  · Calcular la norma máxima de r

  $max \leftarrow 0.0$
  for $i \leftarrow 0$ **hasta** $N - 1$ **do**
    if $|r[i]| > max$ **then**
      $max \leftarrow |r[i]|$
    end if
  end for
  if $max <$ tolerancia **then** ▷ Revisamos la semilla como solución
    Imprimir("Gradiente Conjugado: El vector inicial es una solución aproximada")
    Liberar memoria
    END
  end if

  · Inicializar $p = r$ ▷ Comenzamos iterando
  for $iter \leftarrow 0$ **hasta** $iteraciones - 1$ **do**
    · Calcular $paso_{Apk} = \mathbf{matriz} \cdot \mathbf{p}$
    · Calcular $ak = producto\_punto(r, r)/producto\_punto(p, paso_{Apk})$
    · Actualizar $vector$ con $\mathbf{vector} := \mathbf{vector} + ak * p$
    · Calcular $r_n = r - ak \cdot paso_{Apk}$

    if $\sqrt{producto\_punto(r_n, r_n)} <$ tolerancia **then** ▷ Revisamos convergencia
      Imprimir("Gradiente Conjugado: el método convergió.")
      Liberar memoria
      END
    end if

    · Calcular $bk = producto\_punto(r_n, r_n)/producto\_punto(r, r)$
    for $i \leftarrow 0$ **hasta** $N - 1$ **do** ▷ Actualizamos el residuo
      $p[i] = r_n[i] + bk * p[i]$
    end for
    · Actualizar $r := r_n$
  end for
  Imprimir("Gradiente Conjugado: El método no convergió después.")
  Liberar memoria

end function

Donde podemos ver que el método consiste en determinar el mejor valor para las constantes ak y bk para poder realizar un desplazamiento en alguna dirección $\mathbf{P}$ y determinar la diferencia entre los residuos que se tenga.

Dado a que en cada iteración se redefinen estas constantes con la información de la posición actual, se tiene una convergencia dada en pasos cada vez menores, es decir, el desplazamiento que se realiza entre iteración e iteración es cada vez menor reduciendo el problema de convergencia al problema de cuánta precisión se desee obtener.

Esta actualización también permite que los vectores $\mathbf{P}^{(k)}$ siempre sean uno distinto y nunca se tenga información repetida, esto es de gran ayuda pues podemos asegurarnos que el cambio en las direcciones de estos vectores nunca será tal que se genere un bucle en los valores o el sistema se quede atascado en un conjunto de valores.

Podemos ver que este proceso esencialmente juega con información obtenida de nuestra matriz $\mathbb{A}$ a forma de residuos, lo que le permite determinar un mejor paso. Sin embargo esto significa necesariamente que se necesitará hacer cálculos con información no fundamental o basura en el caso de la inicialización del vector. Si pudiéramos liberarlos o quitar algo de esta información permitiría calcular los datos mucho más rápidos.

Con lo anterior en mente, uno puede proponer un precondicionador que brinde información fundamental extra al sis-

tema que le permita converger más rápidamente a esta información, una propuesta en la mayoría de los casos para este precondicionador es que contenga la información de la diagonal de la matriz, aunque realmente puede ser en esencia cualquier matriz. El pseudocódigo de este método precondicionado se expone a continuación.

2.2. Pseudocódigo Gradiente Conjugado Precondicionado

Tomando como base el pseudocódigo creado para el Gradiente Conjugado, podemos contruir el siguiente pseudocódigo:

Algorithm 2 Método del Gradiente Conjugado Precondicionado

Require: N, tolerancia, matriz, vector, b, M

function GradienteConjugadoPrecondicionado(N, tolerancia, matriz, vector, b, M)

    $iteraciones \leftarrow 2 \cdot N$
    $ak, bk \leftarrow 0$

    · Reservar memoria para $Ax, r, r_n, z, p, paso_{Apk}$
    · Calcular $Ax = \mathbf{matriz} \cdot \mathbf{vector}$
    · Calcular $r = \mathbf{b} - Ax$
    · Resolver $z = \mathbb{M}^{-1} r$

    $max \leftarrow 0.0$
    for $i \leftarrow 0$ **hasta** $N - 1$ **do** ▷ Revisamos la semilla como solución
        if $|r[i]| > max$ **then**
            $max \leftarrow |r[i]|$
        end if
    end for
    if $max <$ tolerancia **then**
        Imprimir(El vector inicial es una solución aproximada)
        Liberar memoria
        END
    end if

    · Inicializar $p = z$
    for $iter \leftarrow 0$ **hasta** $iteraciones - 1$ **do** ▷ Comenzamos iterando
        · Calcular $paso_{Apk} = \mathbf{matriz} \cdot \mathbf{p}$
        · Calcular $ak = producto\_punto(r, z) producto\_punto(p, paso_{Apk})$
        · Actualizar $vector$ con $\mathbf{vector} := \mathbf{vector} + ak * p$
        · Calcular $r_n = r - ak \cdot paso_{Apk}$

        if $\sqrt{producto\_punto(r_n, r_n)} <$ tolerancia **then** ▷ Revisamos convergencia
            Imprimir("Gradiente Conjugado Precondicionado: el método convergió")
            Liberar memoria
            END
        end if

        · Resolver $z = \mathbb{M}^1 \cdot r$
        $z_n = producto\_punto(r_n, z)$
        · Calcular $bk = producto\_punto(r_n, z_n) / producto\_punto(r, z)$

        · Actualizar $p := z_n + bk \cdot p$
        · Actualizar $r := r_n$
        · Actualizar $z := z_n$
    end for
    Imprimir("Gradiente Conjugado Precondicionado: El método no convergió.")
    Liberar memoria

end function

Este método se distingue del anterior en, como se puede ver en la introducción, en multiplicar la matriz inicial por un matriz precondicionador que acelera la convergencia, esto se aplica durante la inicialización del método y en cada iteración al momento de actualizar los datos. En este caso se escogió el precondicionador

$$\mathbb{M}_{ij} = \begin{cases} 1/A_{ii} \ , & i = j \\ 0 \ , & i \neq j \end{cases}$$

Es decir tomar el inverso de los valores de la diagonal solamente. En general esta decisión dependerá totalmente de la matriz y el sistema en cuestion, pero para efectos prácticos esta función es una bastante usada.

2.3. Pseudocódgio $\mathbb{QR}$

Algorithm 3 Factorización QR

Require: N, M, matriz, Q, R
Ensure: Matrices Q y R resultantes de la factorización QR

function QR(N, M, matriz, Q, R)
  · Reservar memoria para matriz A
  for $i \leftarrow 0$ **hasta** $N-1$ **do** ▷ Hacemos una copia
    Reservar memoria para A[i]
    for $j \leftarrow 0$ **hasta** $M-1$ **do**
      $A[i][j] \leftarrow$ matriz[i][j]
    end for
  end for

  norma $\leftarrow 0.0$
  for $i \leftarrow 0$ **hasta** $N-1$ **do** ▷ Calcular la norma de la columna i
    norma $\leftarrow 0.0$
    for $j \leftarrow 0$ **hasta** $M-1$ **do**
      norma $\leftarrow$ norma $+ A[j][i] \cdot A[j][i]$
    end for
    norma $\leftarrow \sqrt{\text{norma}}$
    $R[i][i] \leftarrow$ norma

    for $j \leftarrow 0$ **hasta** $M-1$ **do** ▷ Actualizar la columna i de R y la columna i de Q
      $Q[j][i] \leftarrow A[j][i]/R[i][i]$
    end for

    for $j \leftarrow i+1$ **hasta** $N-1$ **do** ▷ Actualizar el resto de R y Q
      $R[i][j] \leftarrow 0.0$
      for $k \leftarrow 0$ **hasta** $M-1$ **do**
        $R[i][j] \leftarrow R[i][j] + Q[k][i] \cdot A[k][j]$
      end for
      for $k \leftarrow 0$ **hasta** $M-1$ **do**
        $A[k][j] \leftarrow A[k][j] - R[i][j] \cdot Q[k][i]$
      end for
    end for
  end for
  Liberar memoria A
end function

2.4. Pseudocódigo Rayleigh

Algorithm 4 Método de Rayleigh para encontrar autovalores

Require: N, sigma, matriz, v0, tolerancia, iteraciones
Ensure: El autovalor dominante

function MetodoRayleigh(N, sigma, matriz, v0, tolerancia, iteraciones)
  · Reservar memoria para $\mathbb{B}, Ax, v1, aux$
  $norma \leftarrow 0$
  for $i \leftarrow 1$ **hasta** iteraciones **do** ▷ Comenzamos iterando
    · Calcular $\mathbb{B} = \mathbb{A} - \sigma\mathbb{I}$
    · Resolver $\mathbb{B}v1 = v0$
    $norma \leftarrow \sqrt{producto\_punto(v1, v1)}$
    for $i \leftarrow 0$ **hasta** $N-1$ **do** ▷ Normalizamos y guardamos las diferencias
      $v1[i] \leftarrow v1[i]/norma$
      $aux[i] \leftarrow v1[i] - v0[i]$
    end for
    $norma \leftarrow \sqrt{producto\_punto(aux, aux)}$ ▷ Revisamos convergencia
    if $norma <$ tolerancia **then**
      Imprimir(El método de Rayleigh convergió)
      for $i \leftarrow 0$ **hasta** $N-1$ **do** ▷ Actualizamos el resultado
        $v0[i] \leftarrow v1[i]$
      end for
      Liberar memoria
      return sigma
    end if
    for $i \leftarrow 0$ **hasta** $N-1$ **do** ▷ Actualizamos vectores y σ
      $v0[i] \leftarrow v1[i]$
    end for
    Calcular $Ax =$ **matriz** $\cdot\, v0$
    sigma $\leftarrow producto\_punto(v0, Ax)$
  end for
  Imprimir(El método de Rayleigh no convergió)
  Liberar memoria
  return NULL
end function

2.5. Pseudocódigo Iteración Subespacio

Algorithm 5 Método de Iteración de Subespacio para encontrar eigenvalores y eigenvectores

Require: N, matriz, eigenvalores, eigenvectores, tolerancia, iteraciones
Ensure: Eigenvalores y eigenvectores aproximados

function IteracionSubespacio(N, matriz, eigenvalores, eigenvectores, tolerancia, iteraciones)
  Reservar memoria para matrices p, pt, L, U, aux y vectores $v0, v1, y$
  for $i \leftarrow 0$ **hasta** $N-1$ **do**
    Inicializar $p[i][i] \leftarrow 1.0$
  end for
  for iter $\leftarrow 0$ **hasta** iteraciones **do**
    for $i \leftarrow 0$ **hasta** $N-1$ **do**
      Realizar potencia inversa y normalizar $v1$
      Actualizar p con $v1$
    end for
    Calcular *eigenvalores* mediante $matriz \times p \times p^T$
    if *eigenvalores* es diagonal con tolerancia tolerancia **then**
      Imprimir(El método convergió.)
      Liberar memoria
      return *eigenvalores*
    end if
    Aplicar el Método de Jacobi para diagonalizar *eigenvalores* y actualizar *eigenvectores*
    if *eigenvectores* es una matriz ortogonal con tolerancia tolerancia **then**
      Imprimir("La matriz de eigenvectores es ortogonal.")
      Liberar memoria
      return *eigenvalores*, *eigenvectores*
    end if
    Actualizar p mediante $matriz \times p$
  end for
  Imprimir(El método no convergió.)
  Liberar memoria
  return *eigenvalores*, *eigenvectores*
end function

3. Resultados

Para una tolerancia $\texttt{TOL} = 1e-10$, 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 3 valores propios más altos de cada matriz:

	Método				
Tamaño Datos	Iteracion subespacio	Rayleigh	$\mathbb{QR}$	Gradiente Conjugado	Precondicionador de Jacobi
3×3	$\lambda = [2.991343, 6.973860, 10.034796]$	2.991343	$\mathbb{QR} = \mathbb{A}$	$\mathbf{x} = [3.0115, -2.4086, 7.1279]$	$\mathbf{x} = [3.011577, -2.408636, 7.127972]$
5×5	$\lambda = [0.057075, 3.745331, 8.338447, \dots]$	3.745331	$\mathbb{QR} = \mathbb{A}$	-	-
50×50	*	19.998794	$\mathbb{QR} = \mathbb{A}$	-	-
125×125	*	`Error Doolitle 2: L[124][124]*U[124][124] = 0`	$\mathbb{QR} = \mathbb{A}$	$\mathbf{x} = [0.604901, 0.567971, 0.006738, \dots]$	$\mathbf{x} = [0.604901, 0.567971, 0.006738, \dots]$

Donde en el caso del método de la iteración subespacio los lugares con * corresponden a valores propios calculados para 50×50 de ~ 0.9999 con sus tres primeros valores propios muy cercanos, y para el caso de 125×125 de ~ 0.000004; en estos casos lo valores propios no cambian demasiado entre ellos como para reportar un resultado notable.

Notamos que en general se tienen resultados bastantes buenos, siendo que en el caso del método $\mathbb{QR}$ (donde se implementó una función que revisara que $\mathbb{A} - \mathbb{QR} = 0$) donde se obtuvieron todos los resultados favorables. En el caso del método de Rayleigh, se realizó para resolver la factorización $\mathbb{B} = \mathbb{LU}$ el método de Doolittle, para el cual se tuvieron

resultados buenos a excepción de la última matriz donde alguno de los últimos elementos de la diagonal o su producto se hace cero e impide seguir continuando con el método.

4. Conclusiones

Notamos una velocidad mucho mayor al momento de realizar los cálculos de los vectores con el método de la iteración subespacio a comparación de los métodos de la tarea pasada (método de la potencia con Gram-Schmidt y método de Jacobi), lo cuál es confirma que la modificación funciona. Asimismo notamos que el método de gradiente conjugado funciona también bastante mejor que los métodos ya vistos para resolución de sistemas de ecuaciones al ser un método con iteraciones acotadas, y el método del gradiente conjugado precondicionado funciona todavía mejor.

Sin embargo en el último caso, estos dos métodos requieren mayor espacio en la memoria pues se requieren de varios momentos del vector y la matriz en el cálculo, sin mencionar que en el caso del precondicionado que se requiere un espacio en la memoria de N^2 extra en el peor de los casos para el precondicionador (en el menor de los casos y para nuestra elección, de N mayor) sin contar que se requiere hacer su cálculo; todavía más si se necesita o se propone un precondicionador más complicado que el propuesto en este trabajo.

Finalmente vemos que si bien el método de Rayleigh es bastante más rápido que los métodos ya vistos, suponen riesgos al momento de manipular su información lo que nos llevaría a posibles errores. Por otro lado el método de $\mathbb{QR}$ supone un método seguro de descomposición si se cumplen los requisitos para su proceso, esto eso que $\mathbb{A}$ sea invertible; el contra en este caso sería determinar si la matriz cumple con esta condición pues requeriría de un calculo externo.

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. Stewart, G. W. (2001). Matrix Algorithms: Volume II: Eigensystems. Society for Industrial and Applied Mathematics.
5. Teukolsky, S. A., Flannery, B. P., Press, W. H., & Vetterling, W. T. (1992). Numerical recipes in C. SMR, 693(1), 59-70.
6. 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