Instituto Tecnológico de Costa Rica

Análisis Numérico para Ingeniería

I Semestre 2022

Descripción de la implementación del método de Jacobi en paralelo

Pregunta 1.3

Grupo 5

José Julián Camacho Hernández

Juan Pablo Carrillo Salazar

José Leonardo Guillén Fernández

Fabián Ramírez Arrieta

3 de enero de 2024

Índice

1. Descripción del método de Jacobi

El método de Jacobi se emplea para resolver un sistemas de ecuaciones lineal $Ax = b$, donde $A \in \mathrm{R}^{mxm}$ es una matriz diagonalmente dominante.

Una implementación computacional que no hace uso de la representación matricial es un método iterativo que genera una sucesión que debe converger hacia la solución del sistema.

Si $x^{(k)} = [x_1^{(k)} x_2^{(k)} ... x_m^{(k)}]^T$ es la aproximación generada por el método de Jacobi en la k-ésima iteración, entonces la k+1-ésima iteración $x^{(k+1)} = [x_1^{(k+1)} x_2^{(k+1)} ... x_m^{(k+1)}]^T$ se puede calcular a través de la siguiente fórmula:

$$x_i^{(k+1)} = \frac{1}{A(i,i)} \left(b(i) - \sum_{j=1; j \neq i}^{m} A(i,j) x(j) \right) \quad (1)$$

para cada $i = 1, 2, ..., m$. Cabe apuntar que para calcular cada $x_i^{(k+1)}$ solo se necesitan los valores de la iteración anterior $x^{(k+1)}$. Por lo tanto, se puede realizar una implementación en paralelo, calculando cada $x_i^{(k+1)}$ en un procesador o núcleo.

2. Implementación del método de Jacobi en paralelo

El método de Jacobi anteriormente descrito puede ser implementado de tal manera que se calcule cada uno de los valores del vector solución paralelamente para cada iteración. Este procedimiento se explica a continuación.

En cada iteración el método calcula un vector $x^{(k)} = [x_1^{(k)} x_2^{(k)} ... x_m^{(k)}]^T$ que corresponde a la aproximación de la solución del sistema para la iteración k. Cada uno de los valores de dicho vector para la siguiente iteración se calcularán mediante la fórmula 1.

Como se apuntó, debido a que para cada valor $x_i^{(k+1)}$ del siguiente vector solución solo se necesitan los valores de la iteración anterior, cada uno de estos puede ser calculado en uno de m hilos paralelos, que idealmente se ejecutarán en m procesadores.

De esa manera, se calcularán los $x_i^{(k+1)}$ simultáneamente y será posible obtener el vector solución para la k+1-ésima iteración $x^{(k+1)} = [x_1^{(k+1)} x_2^{(k+1)} ... x_m^{(k+1)}]^T$ de manera paralela.

2.1. Diagrama de flujo y pasos para la implementación

En la figura 1, se presenta el diagrama de flujo de la implementación del método en estudio con el fin de visualizar de mejor manera el funcionamiento del mismo.

En este se puede observar que el paso cero para la ejecución del método es contar con la matriz tridiagonal A, con el vector de constantes b y con un vector inicial x_0.

Con estas entradas, el siguiente paso es realizar un ciclo hasta que se cumpla con determinado número de iteraciones máximas. Si no se ha llegado a ese límite, se procede a calcular el vector solución para la siguiente iteración. Este paso es el que se puede llevar a cabo de manera paralela, al calcular cada uno de los valores de dicho vector en diferentes hilos simultáneos.

En cada uno de estos hilos, para calcular cada $x_i^{(k+1)}$, se realiza un bucle para obtener una sumatoria desde $j = 1$ hasta m con $j \neq i$ y se llevan a cabo las operaciones que se describen en la fórmula 1.

Una vez estos cálculos fueron realizados, se tiene la aproximación para la iteración posterior. El siguiente paso es calcular el error de dicha aproximación mediante la norma euclidiana de $Ax^{(k)} - b$.

Se procede a verificar si este error es menor que una tolerancia establecida. En caso negativo, se vuelve a realizar el ciclo principal. Si lo es, se retorna la aproximación de la solución del sistema $x^{(k)}$ obtenida en la iteración actual.

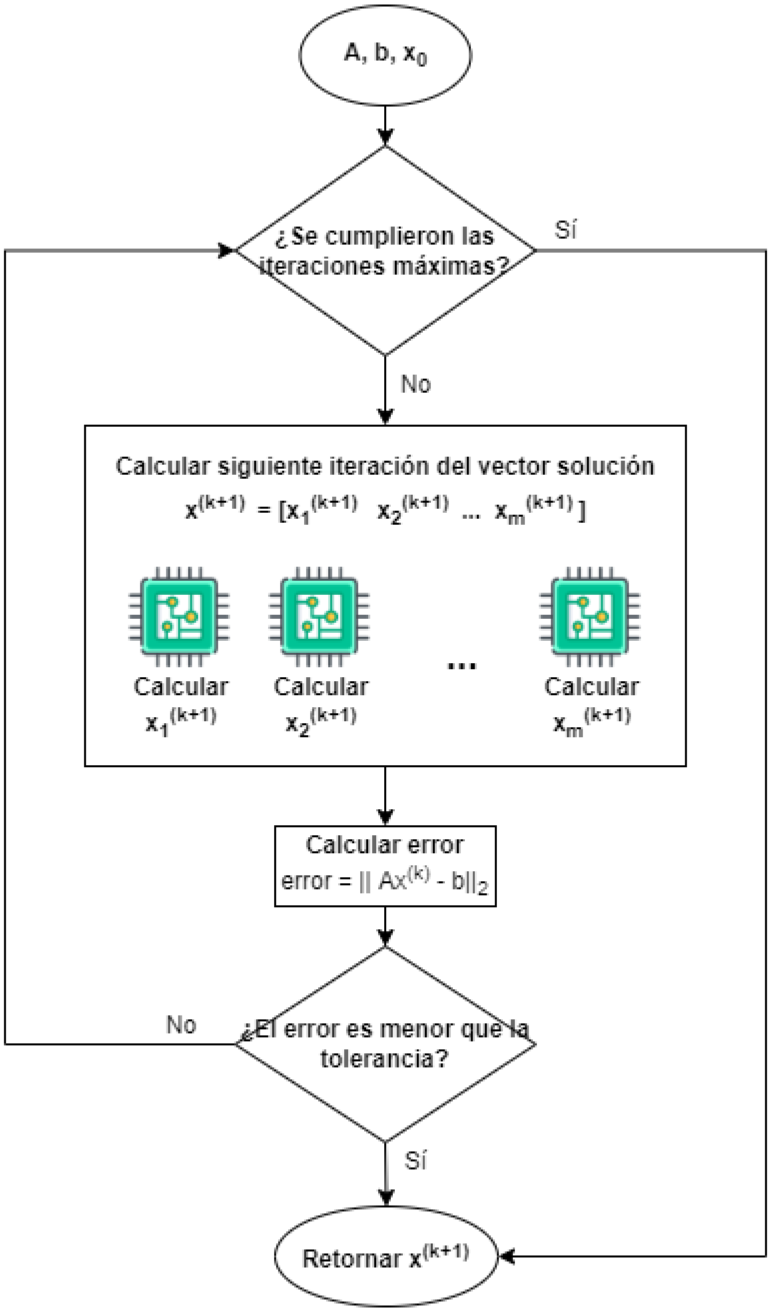

Figura 1: Diagrama de flujo para el método de Jacobi en paralelo

2.2. Pseudocódigos del método

Una vez se tiene el diagrama de flujo de la implementación, se presentan los pseudocódigos de cada una de las partes. El primero que se muestra corresponde a la función principal. Esta se encarga de llevar el control de las iteraciones máximas, del error y de las diferentes aproximaciones del vector solución.

Para generar las aproximaciones, llama a la función que se presenta en el segundo pseudocódigo, que es la implementación computacional de la fórmula 1, que se encarga de calcular cada uno de los valores del vector solución de la siguiente iteración.

Algorithm 1 : Método de Jacobi en paralelo
Entradas:
A: matriz tridiagonal de coeficientes
b: vector de constantes
x0: vector inicial
tol: tolerancia máxima de error del resultado
iterMax: cantidad máxima de iteraciones
Salidas:
x: vector solución del sistema

1: **function** JACOBI_PARALELO(A,b,x_0,tol,iterMax)
2: $\quad m \leftarrow length(A)$
3: $\quad x^{(k)} \leftarrow x_0$
4: $\quad valores\_de\_i \leftarrow [1, 2, 3, ..., m]$
5: $\quad$ **for** $k = 1 : iterMax$ **do**
▷ % Calcular paralelamente cada $x_i^{(k+1)}$ de esta iteración de la solución para cada uno de los valores de i
6: $\quad\quad x_k^{(k+1)}$ = calcular_cada_xi(A,b,x_k,valores_de_i)
7: $\quad\quad error \leftarrow \|Ax^{(k)} - b\|_2$
8: $\quad\quad$ **if** $error < tol$ **then**
9: $\quad\quad\quad$ *break*
10: $\quad\quad$ **end if**
11: $\quad\quad x^{(k)} \leftarrow x_i^{(k+1)}$
12: **return** $x^{(k)}$

En este pseudocódigo, cabe mencionar que en la línea 6, $x_k^{(k+1)}$ será el vector solución para la siguiente iteración, y este será construido al llamar la función `calcular_cada_xi(A,b,xk,valores_de_i)` para cada uno de los valores de i.

Para llevar a cabo dichas llamadas a la función m veces, se utiliza el paquete *parallel* de Octave. Este se encarga de ejecutar la función de manera paralela en la cantidad de procesadores del computador que se indiquen o sea posible.

Algorithm 2 : Método para calcular cada x_i

Entradas:

A: matriz tridiagonal de coeficientes

b: vector de constantes

xk: vector solución para la iteración k

i: numero de la solución

Salidas:

*xi_km*1: vector solucion para la iteración k+1

function CALCULAR_CADA_XI(A,b,x_k,i)

  $m \leftarrow length(A)$

  $sumatoria \leftarrow 0$

  for $j = 1 : m$ **do**

    if $j \neq i$ **then**

      $sumatoria \leftarrow sumatoria + A(i,i) * x_k(j)$

    end if

    $x_i^{(k+1)} \leftarrow \frac{1}{A(i,i)} * (b(i) - sumatoria)$

return $x_i^{(k+1)}$

2.3. Implementación computacional en Octave

La implementación computacional del procedimiento descrito en la sección anterior se presenta en las dos figuras siguientes.

En la primera se destaca el uso del paquete *parallel* para realizar las llamadas a la función en hilos paralelos. El comando que se encarga de ejecutar esto es `pararrayfun(nproc,f,val_i)`. Esta tiene como parámetros la cantidad de procesadores en los que se ejecutará la función *f*, que es la que se evalúa para cada uno de los valores de *val_i*, que en este caso será desde 1 hasta *m*.

La función *f* que se evalúa debe estar en un archivo aparte del mismo nombre.

```
% Implementacion metodo de Jacobi en paralelo
% Entradas:
  % A: matriz tridiagonal de coeficientes
  % b: vector de constantes
  % x0: vector inicial
  % tol: tolerancia maxima de error del resultado
  % iterMax: cantidad maxima de iteraciones
% Salidas: x: vector solucion del sistema
function xk = jacobi_paralelo(A,b,x0,nproc,tol,iterMax)
  m = size(A,1);
  xk = x0;
  val_i = 1:m; %Vector de valores de i para los que se va a evaluar la funcion

  pkg load parallel
  for k=1:iterMax
    f = @(i) calcular_xi(A,b,xk,i); %Funcion que se va a evaluar para cada i
    xkm1 = pararrayfun(nproc,f,val_i).'; %Ejecuciones en paralelo
    error = norm(A*xk-b); %Error para la iteracion
    if error < tol
      break
    endif
    xk = xkm1;
  endfor
endfunction
```

```
% Funcion para solucionar cada componente de la solucion
% de manera paralela
% Entradas:
  % A: matriz tridiagonal de coeficientes
  % b: vector de constantes
  % xk: vector solucion para la iteracion k
  % i: numero de la solucion
% Salidas: xi_km1: vector solucion para la iteracion k+1
function xi_km1 = calcular_xi(A,b,xk,i)
  m = size(A,1);
  sumatoria = 0;
  for j=1:m %Sumatoria que es parte de xi_km1
    if j != i
      sumatoria += A(i,j)*xk(j);
    endif
  endfor
  xi_km1 = (1/A(i,i))*(b(i)- sumatoria); %Calculo final de xi_km1
endfunction
```

3. Anexos

3.1. Instalación paquete parallel

Para la instalación del paquete parallel se debe utilizar el siguiente comando en la terminal de Octave:

pkg install -forge parallel

De no ser posible la instalación por medio de este método, otra opción es descargar el paquete desde el siguiente enlace:

parallel-4.0.1.tar.gz

Una vez descargado, desde la terminal de Octave se dirige a la carpeta donde se guardó el archivo y se utiliza el siguiente comando:

pkg install parallel-4.0.1.tar.gz

En los siguientes enlaces se puede consultar el uso de este paquete.

`https://octave.sourceforge.io/parallel/overview.html`

`https://wiki.octave.org/Parallel_package`