Tarea 8 - Métodos numéricos
Giovanni Gamaliel López Padilla

Índice

1. Introducción

1.1. Métodos iterativos

Un método iterativo es un método que va obteniendo aproximaciones a la solución de un problema. En un método iterativo repite un mismo proceso sobre la solución aproximada. Este método puede ser suspendido por medio de parámetros, ya sea un máximo de iteraciones o un factor de convergencia de la solución.

1.1.1. Método de Jácobi y Gauss-Seidel

Los métodos de Jácobi y Gauss-Seidel son métodos iterativos para encontrar la solución de sistemas de ecuaciones lineales. Los dos consisten en obtener una ecuación de recurrencia y proponr un vector solución inicial.

1.2. Ecuaciones diferenciales parciales

Se denomina a las ecuaciones diferenciales parciales (EDP) a aquellas ecuaciones que involucran derivadas parciales de una función desconocida con dos o más variables independientes. La mayoría de problemas físicos están descritos por EDP de segundo orden. El método análitico para resolver este tipo de ecuaciones es el método de variables separables.

En el caso de el método numérico, uno de los métodos para la solución de EDP es elementos finitos, este considera que el continuo se divide en un número finito de partes que se denominan como elementos. Un parámetro del método es el número de puntos característicos llamados nodos. Estos nodos son los puntos de unión de cada elemento adyacente.[1]

1.2.1. Condición de frontera de Dirichlet

Las condiciones de frontera de Dirichlet son cantidades definidas en los extremos de una ecuación diferencial para obtener la solución particular.

1.2.2. Ecuación de transferencia de calor

La EDP empleada para modelar la transferencia de calor es la siguiente:

$$k\nabla^2 u - \frac{\partial^2 u}{\partial t^2} = 0$$

para el caso estacionario en una dimensión se tiene:

$$k\frac{\partial^2 u}{\partial x^2} + Q = 0 \tag{1}$$

La ecuación 1 es una ecuación diferencial ordinaria con coeficientes constantes.

2. Métodos

2.1. Ecuación de calor

Aplicando el método de diferencias finitas a la ecuación 1 obtenemos la ecuación 2.

$$\frac{k(u_{i+1} - 2u_i + u_{i-1})}{\Delta x^2} + Q = 0 \tag{2}$$

Para el caso de cinco nodos se obtiene el sistema de ecuaciones señalado en la ecuación 3.

$$\begin{cases} u_0 + -2u_1 + u_2 = \frac{-Q\Delta x^2}{K} \\ u_1 + -2u_2 + u_3 = \frac{-Q\Delta x^2}{K} \\ u_2 + -2u_3 + u_4 = \frac{-Q\Delta x^2}{K} \end{cases} \tag{3}$$

El sistema de ecuaciones 3 puede ser descrito la ecuación matricial 4.

$$\begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = \begin{pmatrix} \frac{Q\Delta x^2}{k} + u_0 \\ \frac{Q\Delta x^2}{k} \\ \frac{Q\Delta x^2}{k} + u_4 \end{pmatrix} \tag{4}$$

Donde u_0 y u_4 son las condiciones de frontera del problema. Si expandemos el problmea a n nodos, obtendremos que para describir el sistema como una ecuación matricial debemos seguir los siguientes parámetros. Para obtener la matriz del sistema se tiene que:

$$A = \begin{cases} 2 & \text{para } i = j \\ -1 & \text{para } |i - j| = 1 \\ 0 & \text{en otro lado} \end{cases} \qquad B = \begin{cases} \frac{Q\Delta x^2}{k} + u_0 & \text{para } i = 1 \\ \frac{Q\Delta x^2}{k} + u_n & \text{para } i = n - 1 \\ \frac{Q\Delta x^2}{k} & \text{en otro lado} \end{cases} \tag{5}$$

donde $i, j \in \{1, 2, 3, \ldots, n-1\}$. El sistema de la ecuación 5 describe una matriz tridiagonal, por lo que se obtara por resolver el sistema usando la factorización por Cholesky.

El algoritmo que resuelve esta EDP es el siguiente:

```
// inputs: k, Q, L, u_0, u_n, n
// output: solutions
matrix, results = create_matrix_system(Q,k,l,n)
L = Cholesky(matrix)
solutions_y = solve_triangular_inferior(L, results)
solutions_x = solve_triangular_superior(LT, solutions_y)
```

La funciones Cholesky, solve_triangular_inferior y solve_triangular_superior fueron creadas en tareas anteriores. La función create_matrix_system aplica la ecuación 5 para obtener la ecuación matricial.

2.2. Método de Jácobi

Sea el sistema de ecuaciones lineales $Ax = b$, donde A es la matriz de coeficientes, x es el vector de incógnitas y b el vector de términos independientes. Se propone que A puede ser escrito como la suma dos matrices, tales que una contiene ceros en un diagonal y una matriz diagonal. Entonces, la ecuación matricial se convierte en:

$$Dx + Rx = b$$

despejando Dx, se obtiene la ecuación 6:

$$Dx = b - Rx \tag{6}$$

Multiplicando la ecuación 6 por D^{-1} de lado izquierdo la ecuación 7:

$$x = D^{-1}(b - Rx) \tag{7}$$

donde

$$D^{-1} = \begin{pmatrix} \frac{1}{a_{11}} & 0 & 0 & \cdots & 0 \\ 0 & \frac{1}{a_{22}} & 0 & \cdots & 0 \\ 0 & 0 & \frac{1}{a_{33}} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \frac{1}{a_{nn}} \end{pmatrix}$$

La cual, escribiendo de forma recursiva resulta en la ecuación 8.

$$x^{(k+1)} = D^{-1}(b - Rx^{(k)}) \qquad k = 0, 1, 2, \ldots, n \tag{8}$$

Desarrollando la ecuación 8 para obtener una ecuación para obetner la solución $x_i^{(k+1)}$, se obtiene la ecuación 9.

$$x_i^{(k+1)} = \frac{b_i - \sum_{j \neq i} a_{ij} x_j^{(k)}}{a_{ii}} \tag{9}$$

La ecuación 9 se implemento en el programa solution.h, en la función solve_jabobi. El algoritmo es el siguiente:

```
// inputs: matrix
// output: x
for (int i = 0; i < n; i++)
    {
        sum = 0;
        for (int j = 0; j < n; j++)
        {
            if (j != i)
            {
                sum += m_ij * x_j;
            }
        }
        x_i = (b_i - sum) / m_ii;
    }
```

La tolerancia de este metodo se definió usando una norma (θ), la cual esta descrita en la ecuación 10.

$$\theta = \sqrt{\sum_{i=1}^{n-1} (x_i^{(k+1)} - x_i^{(k)})^2} \tag{10}$$

El método seguira hasta que $\theta < 10^{-6}$. El método de Jácobi se asegura que converge cuando la matriz A es diagonal dominante. En el caso que A no sea diagonal dominante no se asegura obtener una solución al sistema. Se dice que una matriz diagonal dominante es aquella que los elementos de su diagonal cumplen la condición de la ecuación

$$|a_{ii}| \geq \sum_{i \neq j} |a_{ij}| \tag{11}$$

Al inicio del programa se comprueba si la matriz introducida es diagonal convergente. Si no lo es intentará ordenar los elementos de tal manera que el elemento con valor absoluto mayor se encuentre en la diagonal de la matriz. Por ejemplo, si se tiene la matriz aumentada:

$$A = \left(\begin{array}{ccc|c} -1 & 3 & 1 & 2 \\ 7 & 1 & 4 & 5 \\ 2 & -5 & 5 & -1 \end{array}\right)$$

Se buscará el valor mas grande en la matriz para organizarlo en la posición a_{11}, esto realizando un intercambio de filas y columnas.

$$A = \left(\begin{array}{ccc|c} 7 & 1 & 4 & 5 \\ -1 & 3 & 1 & 2 \\ 2 & 2 & 5 & -1 \end{array}\right)$$

Para elemento a_{22}, se buscara el elemento con valor absoluto mayor que tal que su número de fila y columna es mayor a 2. En este ejemplo, la siguiente organización sería la siguiente:

$$A = \left(\begin{array}{ccc|c} 7 & 4 & 1 & 5 \\ 2 & 5 & 2 & -1 \\ -1 & 1 & 3 & 2 \end{array}\right)$$

Al llegar al ultimo del elemento de la diagonal de la matriz no se realizará ningún cambio. Este procedimiento puede no obtener una matriz diagonal dominante. Sin embargo, se comprobo que algunos sistemas que no convergian, al realizar este proceso se llega a su solución. Este proceso de ordenamiento se encuentra contenido en el archivo dominant_diagonal.h.

El algoritmo que sigue es el siguiente:

```
1     // inputs: matrix
2     // output: matrix_ordenada
3     for (int i = 0; i < n; i++)
4     {
5         // Se supone que el maximo ya se encuentra en la diagonal
6         max = fabs(m_ii);
7         i_max = i;
8         j_max = i;
9         for (int j = i; j < n; j++)
10        {
11            for (int k = i; k < n; k++)
12            {
13                if (fabs(m_ij) > max)
14                {
15                    max = fabs(m_ij);
16                    i_max = j;
17                    j_max = k;
18                }
```

```
            }
        }
        if (i != i_max)
        {
            change_columns(matrix,max_position);
        }
        if (i != j_max)
        {
            change_rows(matrix,results);
        }
    }
    is_diagonal_dominant_matrix(matrix);
```

2.3. Método de Gauss-Seidel

El método de Gauss-Seidel es semejante al método de Jácobi. El método de Jácobi usa el valor de las incógnitas para determinar una nueva aproximación en cada iteración, en cambio, el método de Gauss-Seidel calcula las nuevas aproximaciones con los valores calculados en la misma iteración. Entonces, tomando esto en cuenta, la ecuación recursiva para el método de Gauss-Seidel es la siguiente:

$$x_i^{(k+1)} = \frac{b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} \sum_{j=i+1} a_{ij} x_j^{(k)}}{a_{ii}} \tag{12}$$

La convergencia de la ecuación 12 es definida con la ecuación usada para el método de Jacobi (ecuación 10). El algoritmo implementado para este método se encuentra en el archivo solution.h en la función solve_seidel, el cual es el siguiente:

```
    // inputs: matrix
    // output: x
    for (int i = 0; i < n; i++)
        {
            sum = 0;
            for (int j = 0; j < i; j++)
            {
                sum += m_ij * x_j;
            }
            x_i = (b_i - sum) / m_ii;
            sum = 0;
            for (int j = i + 1; j < n; j++)
            {
                sum += m_ij * x_j;
            }
            x_i = x_i - sum / m_ii;
        }
```

3. Resultados

3.1. Ecuación de calor

Los programas y resultados del problema 1a y 1b se encuentran en la carpeta Problema_1. Cada inciso tiene su propia carpeta llamadas Problema_1a yProblema_1b.

3.1.1. Problema 1a

- **Resolver la ecuación de calor considerando {Q = 3, K = 5, u_0 = 10, u_n = 20, n = 4, L = 1}, L es la longitud de la barra.**

  Para este caso se puede visualizar el sistema de ecuaciones, el cual es el siguiente:

$$\begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = \begin{pmatrix} 10.037500 \\ 0.037500 \\ 20.037500 \end{pmatrix} \tag{13}$$

  La solución del sistema 13 es el siguiente:

$$\begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ u_4 \end{pmatrix} = \begin{pmatrix} 10 \\ 12.556250 \\ 15.075000 \\ 17.556250 \\ 20 \end{pmatrix} \tag{14}$$

- **Resolver la ecuación de calor considerando {Q = 3, K = 5, u_0 = 10, u_n = 20, n = 100, L = 1}.**

  El sistema de ecuaciones para el problema es el siguiente:

$$\begin{pmatrix} 2 & -1 & 0 & 0 & \cdots & 0 \\ -1 & 2 & -1 & 0 & \cdots & 0 \\ 0 & -1 & 2 & -1 & \cdots & 0 \\ 0 & 0 & -1 & 2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{98} \\ u_{99} \end{pmatrix} = \begin{pmatrix} 10.000060 \\ 0.000060 \\ 0.000060 \\ 0.000060 \\ 0.000060 \\ 0.000060 \\ 20.000060 \end{pmatrix} \tag{15}$$

  La solución del sistema de ecuaciones 15 se encuentra en el archivo Solution_100.txt.

3.1.2. Problema 1b

Graficar la variación de temperatura u del nodo central contra el número de elementos n ∈ {10, 30, 50, 70, 100} que equivale a n + 1 nodos.

Para este problema se creo un ciclo que repetirá el proceso del problema 1a. En este ciclo, el valor de n cambiará para obtener la solución del problema. Los archivos que contienen el resultado del nodo central y anterior al central de cada iteración son results_1.csv y results_2.csv respectivamente. Se creo un programa en python, el cual lee el archivo de resultados y realiza las gráficas 1 y 2.

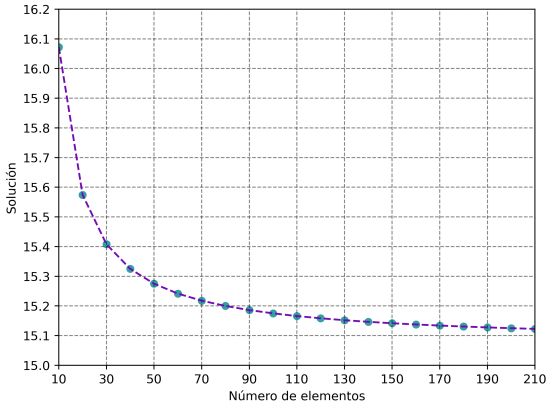

Figura 1: Resultado del nodo central ($\frac{n}{2}$) para cada valor de n.

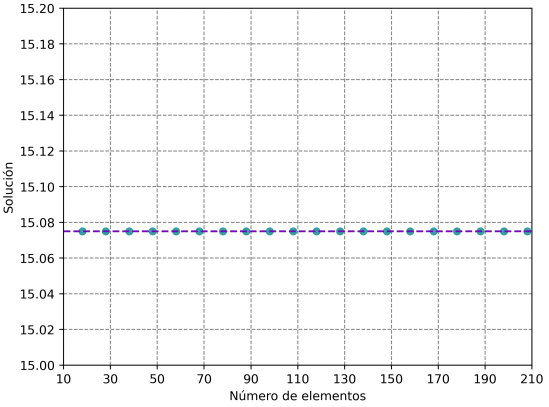

Figura 2: Resultado del nodo anterior al central ($\frac{n}{2} - 1$) para cada valor de n.

3.2. Método de Jácobi

Los programas escritos que implementan el método de Jácobi se encuentran en la carpeta Problema_2a.

3.2.1. Matriz 3x3

La matriz introducida para este caso es la contenida en los archivos M_sys_3x3.txt y V_sys_3x1.txt. El programa arroja la siguiente solución con siete iteraciones:

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 3 \\ -2.5 \\ 7 \end{pmatrix}$$

3.2.2. Matriz 125x125

La matriz introducida para este caso es la contenida en los archivos M_sys_125x125.txt y V_sys_125x1.txt. El programa obtiene una solución con 1874 iteraciones. El output del programa se encuentra en el archivo Solution_125.txt.

3.3. Métodos Gauss-Seidel

3.3.1. Matriz 3x3

La matriz introducida para este caso es la contenida en los archivos M_sys_3x3.txt y V_sys_3x1.txt. El programa arroja la siguiente solución con cinco iteraciones:

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 3 \\ -2.5 \\ 7 \end{pmatrix}$$

3.3.2. Matriz 125x125

La matriz introducida para este caso es la contenida en los archivos M_sys_125x125.txt y V_sys_125x1.txt. El programa obtiene una solución con 991 iteraciones. El output del programa se encuentra en el archivo Solution_125.txt.

4. Conclusiones

4.1. Ecuación de transferencia de calor

Observando la gráfica 1 se concluye que el número de nodos influye en el valor de la solución de la ecuación diferencial. Esto es debido a que al ser un nodo que varia su distancia este obtiene una diferente solución y al añadir más nodos, la distancia entre elementos es menor. En el caso de la gráfica 2 se concluye que al tomar un nodo que tenga la misma distancia independientemente del número de nodos su solución será la misma.

4.2. Método Jácobi y Gauss-Seidel

El método de Jácobi y Gauss-Seidel obtienen la solución a un sistema de ecuaciones. Su método para obtener la solución es muy semejante en los dos métodos, la ventaja que tiene Gauss-Seidel es que este converge a una solución con un menor número de iteraciones. Esto puede reflejarse con las matrices que se probo. Jácobi necesita siete iteraciones para llegar a una solución de la matriz de 3x3 dada, en cambio, Gauss-Seidel realizo cinco. En este caso la diferencia no es muy grande, pero esto puede deberse a que es un sistema de ecuaciones pequeño.

Con la matriz de 125x125, el método de Jácobi realizo 1874 iteraciones y Gauss-Seidel le yomo 991 iteraciones para llegar a la misma solución. Aqui se puede probar que el método de Gauss-Seidel es más optimo debido a que en la misma iteración toma en cuenta las aproximaciones que calcula.

5. Compilación y ejecucción de los programas

5.1. Ecuación de calor

La compilación de los programas se realizo con el siguiente comando:

```
1    gcc -Wall -Wextra -Werror -pedantic -ansi -o main.out main.c -std=c11 -lm
```

5.1.1. Problema 1a

Para ejecutar el programa se realiza la siguiente linea de código

```
1    ./main.out
```

5.1.2. Problema 1b

Para ejecutar el programa se realiza la siguiente linea de código

```
1    ./main.out
```

5.2. Método de Jacobi

La Compilación de los programas se realizo con el siguiente comando:

```
gcc -Wall -Wextra -Werror -pedantic -ansi -o main.out main.c -std=c11 -lm
```

La ejecución del programa se realiza con el siguiente comando:

```
./main.out matrix vector
```

donde matrix es el nombre del archivo de la matriz y vector es el nombre del archivo del vector de términos independientes. Por ejemplo, para ejecutar el programa con los archivos M_sys_3x3.txt y V_sys_3x1.txt el comando tendria que ser el siguiente:

```
./main.out M_sys_3x3.txt V_sys_3x1.txt
```

5.3. Métodos Gauss-Seidel

La Compilación de los programas se realizo con el siguiente comando:

```
gcc -Wall -Wextra -Werror -pedantic -ansi -o main.out main.c -std=c11 -lm
```

La ejecución del programa se realiza con el siguiente comando:

```
./main.out matrix vector
```

donde matrix es el nombre del archivo de la matriz y vector es el nombre del archivo del vector de términos independientes. Por ejemplo, para ejecutar el programa con los archivos M_sys_3x3.txt y V_sys_3x1.txt el comando tendria que ser el siguiente:

```
./main.out M_sys_3x3.txt V_sys_3x1.txt
```

6. Referencias

[1] M. Acosta, C. R.de Coss. Solución numérica de ecuaciones diferenciales unidimensionales por el método de diferencias finitas. *Ingeniería*, 2016.

[2] Curtis Gerald. *Applied numerical analysis*. Pearson/Addison-Wesley, Boston, 2004.