Tarea 8 Métodos Numéricos

Carlos Giovanny Encinia González

September 26 2021

Resumen

En el presente trabajo primero mostraré el método de diferencias finitas para poder solucionar la ecuación de calor en una dimensión. Se modelará la función de dicho problema y se le dará solución con alguno de los métodos vistos anteriormente.

Después se mostraran dos métodos iterativos para encontrar soluciones a sistemas de ecuaciones lineales. El primer método que se mostrara es el método de Jacobi, se desarrollaran algunas ideas para poder llegar a su implementación algorítmica. El segundo método es el método de Gauss-Seidel del cuál desarrollaré algunas ideas hasta llegar a su implementación algorítmica.

Desarrollo

Calor

Primero modelaremos la ecuación de calor en una dimensión.

Sea

ϕ la temperatura.

q el flujo de calor Primero definimos el flujo de calor a través de la barra como:

$$q = -k\frac{d\phi}{dx} \tag{1}$$

Donde k es una constante que depende de los materiales de la barra.

Si decimos que al entrar un flujo de calor y aplicar una cantidad de calor Q al salir de la barra debe de de tener la misma cantidad de calor, hacemos

$$q + Q = q + \frac{dq}{dx} \tag{2}$$

Despejando Q tenemos que:

$$Q = \frac{dq}{dx} \tag{3}$$

Utilizando la ecuación (1) y sustituyéndola en la ecuación anterior tenemos que:

$$Q = \frac{d}{dx}\left(-k\frac{d\phi}{dx}\right) \tag{4}$$

Ahora moviendo el termino diferencial al lado izquierdo tenemos que:

$$Q + \frac{d}{dx}\left(k\frac{d\phi}{dx}\right) = 0 \tag{5}$$

Aplicando el operador diferencial tenemos que:

$$Q + k\frac{d^2\phi}{dx^2} = 0 \tag{6}$$

Ahora podemos tomar diferenciales de Q tales que:

$$\frac{d\phi_i}{dx} = \frac{\phi_{i+1} - \phi_i}{\Delta x} \tag{7}$$

Para calcular la segunda derivada podemos hacer

$$\frac{d^2\phi}{dx^2} = \frac{\frac{d\phi_i}{dx} - \frac{d\phi_{i-1}}{dx}}{\Delta x} \tag{8}$$

Ahora sustituyendo la expresión de la primera derivada tenemos que:

$$\frac{d^2\phi}{dx^2} = \frac{\frac{\phi_{i+1}+\phi_i}{\Delta x} - \frac{\phi_i-\phi_{i-1}}{\Delta x}}{\Delta x} \tag{9}$$

Realizando operaciones podemos llegar a:

$$\frac{d^2\phi}{dx^2} = \frac{\phi_{i+1} - 2\phi_i + \phi_{i-1}}{\Delta x^2} \tag{10}$$

Sustituyendo la ecuación (10) en la ecuación (6) tenemos que:

$$Q + k\frac{\phi_{i+1} - 2\phi_i + \phi_{i-1}}{\Delta x^2} = 0 \tag{11}$$

Despejando los ϕ tenemos que:

$$\phi_{i+1} - 2\phi_i + \phi_{i-1} = -\frac{Q\Delta x^2}{k} \tag{12}$$

Ahora sustituyendo algunos valores para i tenemos que

$$\phi_0 - 2\phi_1 + \phi_2 = -\frac{Q\Delta x^2}{k}$$

$$\phi_1 - 2\phi_2 + \phi_3 = -\frac{Q\Delta x^2}{k}$$

$$\vdots$$

$$\phi_{n-2} - 2\phi_{n-1} + \phi_n = -\frac{Q\Delta x^2}{k}$$

Si tenemos condiciones de Dirchlet entonces la ecuación con $i = 0$ e $i = n$, se vuelven constantes, multiplicando por -1 en ambos lados tenemos que:

$$2\phi_1 - \phi_2 = \frac{Q\Delta x^2}{k} + \hat{\phi}_0$$

$$-\phi_1 + 2\phi_2 - \phi_3 = \frac{Q\Delta x^2}{k}$$

$$\vdots$$

$$2\phi_{n-1} - \phi_n = \frac{Q\Delta x^2}{k} + \hat{\phi}_n$$

Y lo anterior sería equivalente a resolver el siguiente sistema de ecuaciones.

$$\begin{pmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & 2 & -1 & \cdots & 0 \\ 0 & -1 & 2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 2 \end{pmatrix} \begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \vdots \\ \phi_{n-1} \end{pmatrix} = \begin{pmatrix} \frac{Q\Delta x^2}{k} + \hat{\phi}_0 \\ \frac{Q\Delta x^2}{k} \\ \frac{Q\Delta x^2}{k} \\ \vdots \\ \frac{Q\Delta x^2}{k} + \hat{\phi}_n \end{pmatrix} \tag{13}$$

Observamos que tenemos una matriz tri diagonal y que además es simétrica. Con base a lo anterior trataremos de construir un algoritmo para poder hacer una factorización de Cholesky.

$$A = \begin{pmatrix} \beta_1 & \gamma_1 & 0 & \cdots & 0 \\ \gamma_1 & \beta_2 & \gamma_2 & \cdots & 0 \\ 0 & \gamma_2 & \beta_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \beta_{n-1} \end{pmatrix} \tag{14}$$

Si la matriz es definida positiva y sabemos que es simétrica entonces podemos expresarla como:

$$A = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ l_{2,1} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{m,1} & l_{m,2} & \cdots & 1 \end{pmatrix} \begin{pmatrix} d_1 & 0 & \cdots & 0 \\ 0 & d_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & d_m \end{pmatrix} \begin{pmatrix} 1 & l_{2,1} & \cdots & l_{m,1} \\ 0 & 1 & \cdots & l_{m,2} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix} \tag{15}$$

Al multiplicar la primera matriz por la segunda tenemos que:

$$A = \begin{pmatrix} d_1 & 0 & \cdots & 0 \\ l_{2,1}d_1 & d_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{m,1}d_1 & l_{m,2}d_2 & \cdots & d_m \end{pmatrix} \begin{pmatrix} 1 & l_{2,1} & \cdots & l_{m,1} \\ 0 & 1 & \cdots & l_{m,2} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix} \tag{16}$$

Si multiplicamos la primera fila de la matriz a la izquierda por la primera columna de la matriz triangular superior tenemos que:

$$\boxed{d_1 = \beta_1} \tag{17}$$

Al multiplicar la primera fila de la matriz de la izquierda por la segunda columna de la triangular superior.

$$\gamma_1 = l_{2,1}d_1 \tag{18}$$

Al despejar $l_{2,1}$ tenemos que:

$$l_{2,1} = \frac{\gamma_1}{d_1} \tag{19}$$

Al multiplicar la segunda fila de la matriz de la izquierda por la primera columna de la triangular superior.

$$\beta_2 = l_{2,1}^2 + d_2 \tag{20}$$

Despejando d_2 tenemos que:

$$d_2 = \beta_2 - l_{2,1}^2 \tag{21}$$

Podemos observar que los valores para las β y las γ son siempre los mismos, entonces sustituyendo los valores tenemos que:
y observando los resultados anteriores podemos llegar a:

$$\boxed{d_i = 2 - l_{i,i-1}^2 d_{i-1}} \tag{22}$$

$$\boxed{l_{i+1,i} = \frac{-1}{d_i}} \tag{23}$$

Ahora como tenemos tres factores podemos escribir

$$Ax = LDL^T x = y \tag{24}$$

y ahora haciendo

$$L^T x = z \tag{25}$$

$$LDz = y$$

$$Dz = w \tag{26}$$

$$Lw = y \tag{27}$$

Tendríamos entonces primero que resolver un sistema con matriz triangular inferior, después diagonal y finalizar con un diagonal para encontrar x.

Pero al tener solo elementos en forma de dos diagonales, podríamos encontrar una forma sencilla de resolverlo.

Para la matriz triangular inferior, donde n es el numero de nodos tenemos que:

$$Lw = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ l_{2,1} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & l_{n-1,n-2} & 1 \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_{n-1} \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{pmatrix} \tag{28}$$

Al ir multiplicando fila por columna vamos obteniendo que

$$w_1 = y_1$$

$$w_2 = y_2 - l_{2,1} w_1$$

$$w_3 = y_3 - l_{3,2} w_2$$

Como los demás elementos fuera de las diagonales son cero, podemos dejar de utilizar el primer subíndice de l y escribir entonces que:

$$\boxed{w_1 = y_1} \tag{29}$$

$$\boxed{w_i = y_i - l_{i-1} w_{i-1}} \tag{30}$$

Para $i = 2 \rightarrow i = n - 2$

Realizando un análisis similar pero para la matriz triangular superior llegamos a que:

$$\boxed{x_{n-1} = z_{n-1}} \tag{31}$$

$$\boxed{x_i = z_i - l_i x_{i+1}} \tag{32}$$

Para $i = n - 2 \rightarrow i = 1$

Sabiendo esto podemos aprovechar para guardar los valores de la matriz original, y la de los factores en estructuras de n-1 x 2 dimensiones.

$$A = \begin{pmatrix} \beta & \gamma \\ \beta & \gamma \\ \vdots & \vdots \\ \beta & 0 \end{pmatrix} \tag{33}$$

$$Factores = \begin{pmatrix} d_1 & l_1 \\ d_2 & l_2 \\ d_3 & l_3 \\ \vdots & \vdots \\ d_{n-1} & 0 \end{pmatrix} \tag{34}$$

Teniendo en cuenta todo lo anterior y que las dos últimas matrices tienen las mismas dimensiones podemos escribir el siguiente pseudocódigo el cual utiliza la misma matriz para solucionar el problema.

Algorithm 1: Solución ecuación con matriz triangular inferior para calor

```
1 solve_calor_inferior (matrix, y, rows, w);
  Input  : matrix: matriz que contiene coficientes ecuación de calor
           factorizada, y: vector con equivalencias, rows: numero de
           filas, w: vector que guardara las soluciones
  Output: w
2 w_i = y_i;
3 i ← 1;
4 for i:rows do
5 |   w_i = y_i − matrix_{i−1,1} ∗ w_{i−1};
6 end
7 return w;
```

Algorithm 2: Solución ecuación con matriz diagonal para calor

1 solve_calor_diagonal $(matrix, w, rows, z)$;
  Input : matrix: matriz que contiene coficientes ecuación de calor factorizada, w: vector con equivalencias, rows: numero de filas, z: vector que guardara las soluciones
  Output: z
2 $w_i = y_i$;
3 $i \leftarrow 0$;
4 **for** *i:rows* **do**
5 | $z_i = w_i / matrix_{i,0}$;
6 **end**
7 **return** z;

Algorithm 3: Solución ecuación de calor por medio de factorización Cholesky

1 solve_calor_diferencias_finitas (Q, K, d_i, d_n, n, L);
  Input : Q: cantidad de energía (calor), K: difusidad térmica, d_i: condición de Dirchlet inicial, d_n: condición de Dirchlet final, n: el numero de nodos, L: la longitud de la barra
  Output: x
2 $matrix \leftarrow crearMatriz(n-1, 2)$;
3 $matrix_{0,0} \leftarrow 2$;
4 $y \leftarrow crearVector(n-1)$;
5 $value = Q/(n * n * K)$;
6 $i \leftarrow 0$;
7 **for** *i:n - 1* **do**
8 | $y_i \leftarrow value$;
9 **end**
10 $y_0 \leftarrow y + d_i$;
11 $y_{i-1} \leftarrow y_{i-1} + d_n$;
12 $w \leftarrow crearVector(n-1)$;
13 $z \leftarrow crearVector(n-1)$;
14 $matrix_{0,1} = -1/matrix_{0,0}$;
15 $i \leftarrow 1$;
16 **for** *i:rows* **do**
17 | $matrix_{i,0} \leftarrow 2 - matrix^2_{i-1,1} * matrix_{i-1,0}$;
18 | $matrix_{i,1} = -1/matrix_{i,0}$;
19 **end**
20 $matrix_{i-1,1} \leftarrow 0$;
21 solve_calor_inferior(matrix, y, rows, w);
22 solve_calor_diagonal(matrix, w, rows, z);
23 solve_calor_superior(matrix, z, rows, x);
24 **return** x;

Algorithm 4: Solución ecuación con matriz triangular superior para calor

solve_calor_superior $(matrix, w, rows, x)$;
Input : matrix: matriz que contiene coeficientes ecuación de calor factorizada, w: vector con equivalencias, rows: numero de filas, x: vector que guardara las soluciones
Output: w
$x_{rows-1} = z_{rows-1}$;
$i \leftarrow rows - 2$;
for $i; i >= 0; i \leftarrow i - 1$ **do**
  $x_i = z_i - matrix_{i,1} * x_{i+1}$;
end
return w;

Método de Jacobi

Para el método de Jacobi primero partiremos de la siguiente ecuación:

$$Ax = b \tag{35}$$

Ahora vamos a descomponer la matriz A en dos matrices, una que contenga solo los valores de la diagonal, y otra que contenga los valores de fuera de la diagonal.

$$(T + D)x = b \tag{36}$$

Pasando a la derecha el termino con la matriz T tenemos que:

$$Dx = b - Tx \tag{37}$$

Ahora multiplicando por la inversa de la matriz D tenemos que:

$$D^{-1}Dx = D^{-1}(b - Tx) \tag{38}$$

$$x = D^{-1}(b - Tx) \tag{39}$$

Si esta ultima ecuación le aplicamos recursividad podemos hacer:

$$x^{n+1} = D^{-1}(b - Tx^n) \tag{40}$$

Pasándola a una forma para implementar en un algoritmo tenemos que:

$$x_i^{n+1} = (b_i - \sum_{j \neq i}^{n} a_{i,j}x_j^n)/a_{i,i} \tag{41}$$

En seguida se muestra el pseudocódigo:

Algorithm 5: Método de Jacobi

```
1  solve_jacobi (matrix, rows, y, x);
   Input  : matrix: matriz a resolver, x: vector inicial, y: es la igualdad,
            rows: numero de filas
   Output: x
2  xk ← crearVector(rows);
3  xk ← x;
4  condition ← 1;
5  k ← 0;
6  while condition do
7  |  i ← 0;
8  |  for i:rows do
9  |  |  sum ← 0;
10 |  |  j ← 0;
11 |  |  for j:rows do
12 |  |  |  if j ≠ i then
13 |  |  |  |  sum ← sum + matrix_{i,j} x_j
14 |  |  |  end
15 |  |  end
16 |  |  xk_i = (y_i − sum)/matrix_{i,i}
17 |  end
18 |  condition ← convergencia(matrix, rows, xk, y);
19 |  x ← xk;
20 end
21 return x;
```

Algorithm 6: Convergencia

```
1  solve_jacobi (matrix, rows, x, y);
   Input  : matrix: matriz a resolver, x: vector inicial, y: es la igualdad,
            rows: numero de filas
   Output: condition
2  i ← 0;
3  intervalo ← 0.00001;
4  condition ← 0 bc ← crearVector(rows);
5  for i:rows do
6  |  j ← 0;
7  |  sum ← 0;
8  |  for j:rows do
9  |  |  sum ← sum + matrix_{i,j} * x_j
10 |  end
11 |  number ← abs(sum − y_i)
12 |  if number > intervalo then
13 |  |  return 1;
14 |  end
15 end
16 return condition;
```

Método Gauss-Seidel

Primero definamos la siguiente ecuación

$$Ax - b = 0 \tag{42}$$

Ahora descompongamos la matriz A en la suma de dos matrices triangulares y una diagonal

$$A = (L + D + U) \tag{43}$$

Entonces nos queda que:

$$(L + D + U)x - b = 0 \tag{44}$$

Ahora propongamos una inversa la cual sera $(L + D)^{-1}$
multiplicando por la inversa tenemos que

$$(L + D)^{-1}(L + D + U)x - b = 0 \tag{45}$$

$$(L + D)^{-1}(L + D)x + (L + D)^{-1}Ux - (L + D)^{-1}b = 0 \tag{46}$$

Multiplicando la inversa y agrupando términos semejantes, nos queda que:

$$x + (L + D)^{-1}(Ux - b) = 0 \tag{47}$$

Despejando x nos queda que:

$$x = (L + D)^{-1}(b - Ux) \tag{48}$$

Multiplicando por $(L + D)$ en ambos lados de la igualdad.

$$(L + D)x = b - Ux \tag{49}$$

Ahora aquí aplicaremos recursividad, siendo que la x del lado derecho sera la nueva variable actualizada.

$$(L + D)x^{n+1} = b - Ux^n \tag{50}$$

Expandiendo el lado derecho y pasando Lx a la derechas nos queda que;

$$Dx^{n+1} = b - Ux^n - Lx^{n+1} \tag{51}$$

Ahora multiplicando por la inversa de la diagonal tenemos que:

$$x^{n+1} = D^{-1}(b - Ux^n - Lx^{n+1}) \tag{52}$$

Ahora reescribiendo de tal manera que podamos usarla para un algoritmo:

$$x_i^{n+1} = (b_i - \sum_{j=1, j\neq i}^{n} U_{i,j}x_j^n - \sum_{j=1, j\neq i}^{n} L_{i,j}x_j^{n+1})/D_{i,i} \tag{53}$$

Si queremos utilizar la matriz original para optimizar espacio, podemos hacer

$$x_i^{n+1} = (b_i - \sum_{j=i+1}^{n} a_{i,j}x_j^n - \sum_{j=1}^{i-1} a_{i,j}x_j^{n+1})/a_{i,i} \tag{54}$$

Ahora para usar un solo vector, podemos partir de que el termino con x^{n+1} es el que se llena primero ya que la sumatoria parte desde j=1, por otro lado x^n se llena después además que esta x^n solo necesita los términos mayores o iguales a $i+1$. Entonces podemos escribir el siguiente pseudocódigo:

Algorithm 7: Método de Gauss Seidel

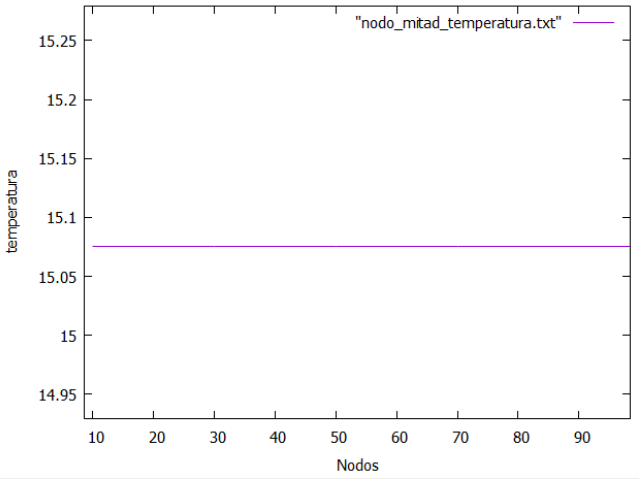
```
1  solve_jacobi (matrix, rows, y, x);
   Input  : matrix: matriz a resolver, x: vector inicial, y: es la igualdad,
            rows: numero de filas
   Output: x
2  condition ← 1;
3  k ← 0;
4  while condition do
5      i ← 0;
6      for i:rows do
7          sum ← 0;
8          j ← 0;
9          for j:rows do
10             if j ≠ i then
11                 sum ← sum + matrix_{i,j} x_j
12             end
13         end
14         x_i = (y_i − sum)/matrix_{i,i}
15     end
16     condition ← convergencia(matrix, rows, x, y);
17 end
18 return x;
```

Resultados

Ecuación de calor

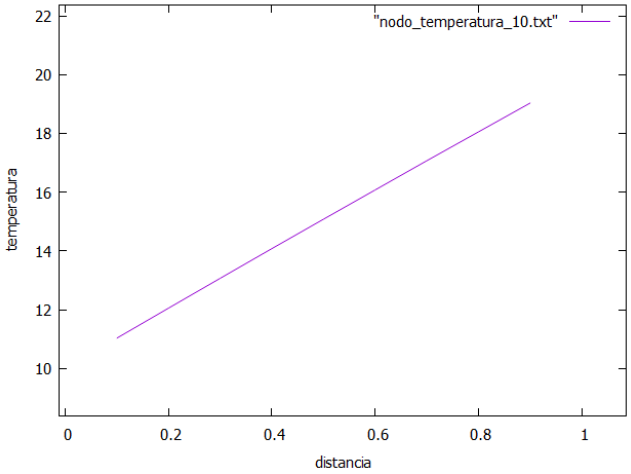
La siguiente imagen muestra lo que se pide en la tarea, el cambio que existe en ϕ para el nodo que esta en medio. Este resulta ser que siempre es el mismo con un valor de 15.075 para las condiciones que se nos da en el problema. $Q = 3$, $K = 5$, $\hat{\phi}_i = 10$, $\hat{\phi}_n = 20$, $L = 1$.

Figure 1: Temperatura nodo medio



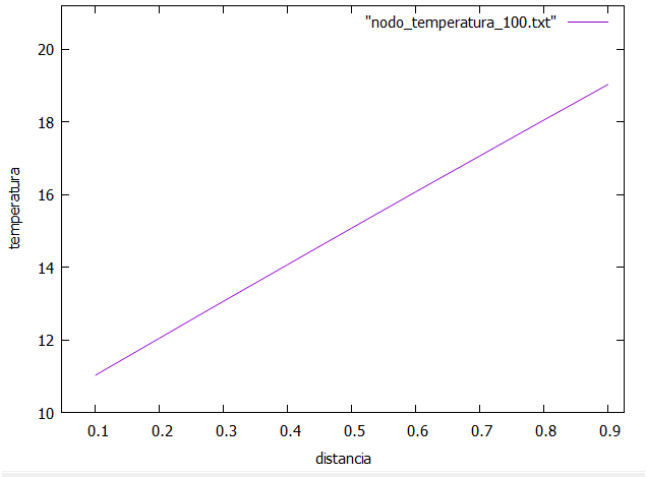
Decidí graficar la solución del problema, es decir los valores de ϕ contra los valores de la distancia en donde se encuentran, y esto es lo que obtuve.

Figure 2: Distancia vs. Temperatura con 10 nodos



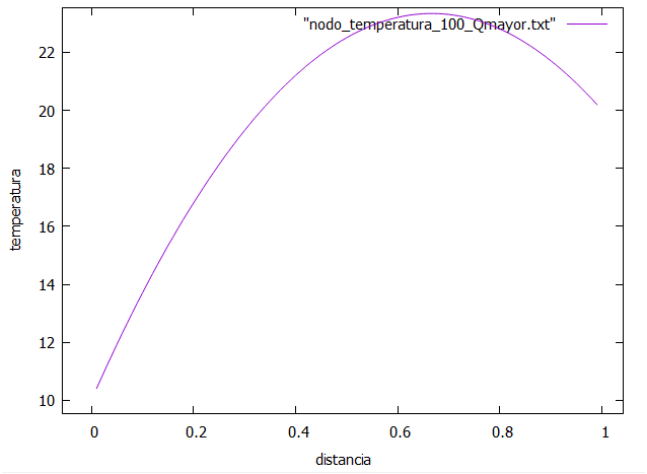
si aumentamos la cantidad de nodos, la recta sigue siendo la misma.

Figure 3: Distancia vs. Temperatura con 100 nodos



Si aumentamos Q, por ejemplo $Q = 300$, observamos que hay un cambio en la gráfica.

Figure 4: 100 nodos con aumento de Q



Método Jacobi, Método Gauss-Seidel

En seguida se muestran los resultados obtenidos al aplicar los métodos de Jacobi y Gauss-Seidel, la primera imagen es para una matriz 3x3, y las siguientes dos son para una matriz de 125x125.

Figure 5: Solución 3x3

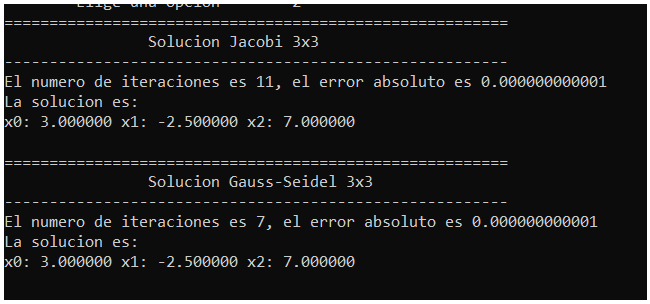


Figure 6: Solución 125x125 Jacobi

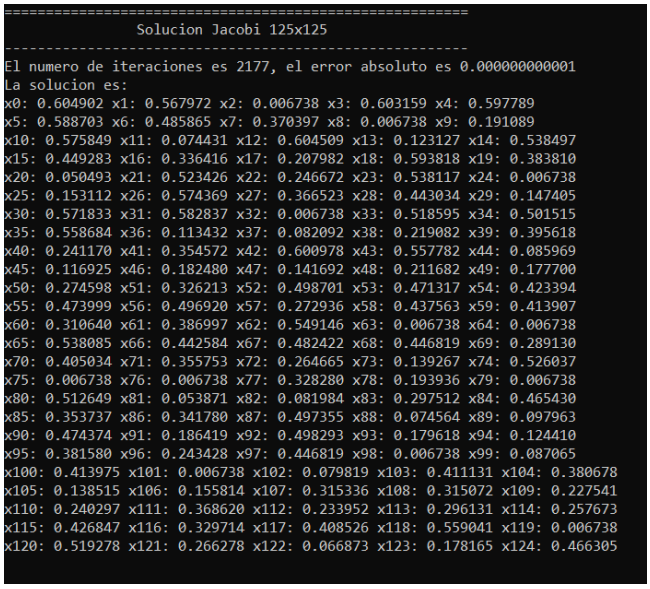
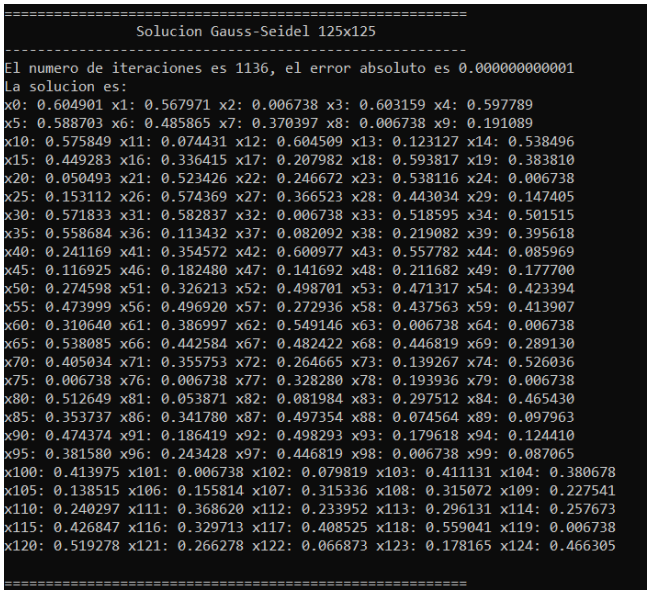


Figure 7: Solución 125x125 Gauss-Seidel



Conclusión

Para el problema de calor pude observar que la temperatura en el punto medio no es dependiente de la cantidad de nodos que le demos al sistema.Si lo pensamos detenidamente, al ser un sistema estacionario, la temperatura debería de distribuirse de manera "uniforme" y con esto me refiero a que si le damos una fuente de calor que no sea en los extremos, la temperatura será invariante a la cantidad de nodos, pero claro esta que entre mas nodos tengamos más información tendremos para cada punto de la barra. También pude observa que un aumento de energía Q hace que la temperatura en el centro aumente esto tiene sentido ya que en los extremos la temperatura tendría a estabilizarse hasta llegar a las condiciones iniciales. Entonces si se tiene poca energía el sistema simplemente debería poder describirse con una ecuación lineal, al aumentar la energía los valores cercanos al centro siempre aumentarán.

Para la solución a las matrices pude observar que el método de Gauss-Seidel siempre es mucho más rápido. En el programa pido que el error para cada elemento de la solución tenga un error absoluto de $1E-12$, y podemos observar que al método de Gauss-Seidel, le toma la mitad de iteraciones que al método de Jacobi. Además el método de Gauss-Seidel, requiere de menos espacio en memoria, ya que podemos utilizar la misma estructura del vector antiguo, para guardar la solución.