

# Análisis Numérico: Taller 3

06 de febrero de 2025

*Universidad Nacional de Colombia*

Jorge Mauricio Ruiz Vera

Andrés David Cadena Simons  
Sandra Natalia Florez Garcia

acadenas@unal.edu.co  
sflorezga@unal.edu.co

## Problema 1:

Sea

$$A = \begin{pmatrix} 0 & -20 & 14 \\ -3 & 27 & 4 \\ -4 & 11 & 2 \end{pmatrix} \quad (1)$$

a) Aplique las transformaciones de Householder, para calcular  $A = QR$ .

### Solución:

Sea  $A$  una matriz de tamaño  $3 \times 3$ . Buscamos descomponerla como  $A = QR$ , donde  $Q$  es una matriz ortogonal y  $R$  una matriz triangular superior, ambas de tamaño  $3 \times 3$ .

Como primer paso, tomemos la primera columna de  $A$ , que llamaremos  $a_1 = (0, -3, -4)^T$ , y definamos el vector de Householder  $v_1$ :

$$\begin{aligned} v_1 &= a_1 + \text{sign}(a_{11})\|a_1\|e_1 \\ &= (0, -3, -4)^T + \|(0, -3, -4)\|(1, 0, 0)^T \\ &= (0, -3, -4)^T + 5(1, 0, 0)^T \\ &= (5, -3, -4)^T \end{aligned}$$

A continuación, calculemos  $v^T v$  y el producto exterior  $vv^T$ :

$$\begin{aligned} v^T v &= (5 \quad -3 \quad -4) \begin{pmatrix} 5 \\ -3 \\ -4 \end{pmatrix} = 50 \\ vv^T &= \begin{pmatrix} 5 \\ -3 \\ -4 \end{pmatrix} (5 \quad -3 \quad -4) = \begin{pmatrix} 25 & -15 & -20 \\ -15 & 9 & 12 \\ -20 & 12 & 16 \end{pmatrix} \end{aligned}$$

Ahora calculamos la matriz de Householder correspondiente:

$$\begin{aligned} F_1 &= I - \frac{2}{v^T v} vv^T \\ &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} - \frac{2}{50} \begin{pmatrix} 25 & -15 & -20 \\ -15 & 9 & 12 \\ -20 & 12 & 16 \end{pmatrix} \\ &= \begin{pmatrix} 0 & \frac{3}{5} & \frac{4}{5} \\ \frac{3}{5} & \frac{16}{25} & -\frac{12}{25} \\ \frac{4}{5} & -\frac{12}{25} & \frac{9}{25} \end{pmatrix} \end{aligned}$$

Aplicamos  $F_1$  a  $A$  para introducir ceros debajo de la primera entrada:

$$F_1 A = \begin{pmatrix} 0 & \frac{3}{5} & \frac{4}{5} \\ \frac{3}{5} & \frac{16}{25} & -\frac{12}{25} \\ \frac{4}{5} & -\frac{12}{25} & \frac{9}{25} \end{pmatrix} \begin{pmatrix} 0 & -20 & 14 \\ -3 & 27 & 4 \\ -4 & 11 & 2 \end{pmatrix} = \begin{pmatrix} -5 & 25 & 4 \\ 0 & 0 & 10 \\ 0 & -25 & 10 \end{pmatrix}$$

Ahora tomamos la submatriz inferior derecha de  $F_1 A$ , denotada por:

$$A^{(2)} = \begin{pmatrix} 0 & 10 \\ -25 & 10 \end{pmatrix}$$

y repetimos el proceso anterior para  $a_2 = (0, -25)^T$ . Definimos entonces  $v_2$ :

$$\begin{aligned} v_2 &= (0, -25)^T + \|(0, -25)^T\| (1, 0)^T \\ &= (25, -25)^T \end{aligned}$$

Calculamos su norma y el producto exterior:

$$\begin{aligned} v_2^T v_2 &= \begin{pmatrix} 25 & -25 \end{pmatrix} \begin{pmatrix} 25 \\ -25 \end{pmatrix} = 1250 \\ v_2 v_2^T &= \begin{pmatrix} 25 \\ -25 \end{pmatrix} \begin{pmatrix} 25 & -25 \end{pmatrix} = \begin{pmatrix} 625 & -625 \\ -625 & 625 \end{pmatrix} \end{aligned}$$

Entonces, la nueva matriz de Householder es:

$$\begin{aligned} F_2 &= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} - \frac{2}{1250} \begin{pmatrix} 625 & -625 \\ -625 & 625 \end{pmatrix} \\ &= \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \end{aligned}$$

Extendemos esta matriz a tamaño  $3 \times 3$  para aplicarla a  $F_1 A$ :

$$F_2' F_1 A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} -5 & 25 & 4 \\ 0 & 0 & 10 \\ 0 & -25 & 10 \end{pmatrix} = \begin{pmatrix} -5 & 25 & 4 \\ 0 & -25 & 10 \\ 0 & 0 & 10 \end{pmatrix} = R$$

Finalmente, obtenemos la descomposición  $A = QR$  donde  $Q = F_1^T (F_2')^T$ :

$$\underbrace{\begin{pmatrix} 0 & -20 & 14 \\ -3 & 27 & 4 \\ -4 & 11 & 2 \end{pmatrix}}_A = \underbrace{\begin{pmatrix} 0 & \frac{4}{5} & \frac{3}{5} \\ \frac{3}{5} & -\frac{12}{25} & \frac{16}{25} \\ \frac{4}{5} & \frac{9}{25} & -\frac{12}{25} \end{pmatrix}}_Q \underbrace{\begin{pmatrix} -5 & 25 & 4 \\ 0 & -25 & 10 \\ 0 & 0 & 10 \end{pmatrix}}_R$$

b) Aplique el método de ortogonalización de Gram-Schmidt, para calcular  $A = QR$ .

### Solución:

Al igual que en el caso anterior, buscamos una matriz ortogonal  $Q$  y una matriz triangular superior  $R$  tal que  $A = QR$ . Para este caso, generamos un conjunto de vectores ortonormales  $v_1, v_2, v_3$  a partir de las columnas de  $A$ .

Denotamos las columnas en orden como  $c_1, c_2$  y  $c_3$ , y aplicamos el método de Gram-Schmidt:

$$v_1 = \frac{c_1}{\|c_1\|} = \frac{1}{5}(0, -3, -4)^T = \left(0, \frac{-3}{5}, \frac{-4}{5}\right)^T.$$

Para calcular  $v_2$ , primero obtenemos un vector ortogonal a  $v_1$ , denotado como  $v'_2$ :

$$\begin{aligned} v'_2 &= c_2 - (c_2^T v_1) v_1 \\ &= (-20, 27, 11)^T + 25 \left(0, \frac{-3}{5}, \frac{-4}{5}\right)^T \\ &= (-20, 12, -9)^T. \end{aligned}$$

Normalizando  $v'_2$  obtenemos  $v_2$ :

$$v_2 = \frac{v'_2}{\|v'_2\|} = \frac{1}{25}(-20, 12, -9)^T = \left(\frac{-20}{25}, \frac{12}{25}, \frac{-9}{25}\right)^T.$$

Repetimos el proceso para  $v_3$ :

$$\begin{aligned} v'_3 &= c_3 - (c_3^T v_1) v_1 - (c_3^T v_2) v_2 \\ &= (14, 4, 2)^T + 4 \left(0, \frac{-3}{5}, \frac{-4}{5}\right)^T + 10 \left(\frac{-20}{25}, \frac{12}{25}, \frac{-9}{25}\right)^T \\ &= \left(6, \frac{32}{5}, \frac{-24}{5}\right)^T. \end{aligned}$$

Finalmente, normalizamos  $v'_3$  para obtener  $v_3$ :

$$v_3 = \frac{v'_3}{\|v'_3\|} = \frac{1}{10} \left(6, \frac{32}{5}, \frac{-24}{5}\right)^T = \left(\frac{3}{5}, \frac{16}{25}, \frac{-12}{25}\right)^T.$$

La factorización  $QR$  surge al reescribir las columnas originales en términos de los nuevos vectores ortonormales:

$$\left( \begin{array}{c|c|c} c_1 & c_2 & c_3 \end{array} \right) = \left( \begin{array}{c|c|c} v_1 & v_2 & v_3 \end{array} \right) \begin{pmatrix} \|c_1\| & c_2^T v_1 & c_3^T v_1 \\ 0 & \|c_2\| & c_3^T v_2 \\ 0 & 0 & \|c_3\| \end{pmatrix}.$$

Por lo tanto, la factorización  $QR$  resultante es:

$$\underbrace{\begin{pmatrix} 0 & -20 & 14 \\ -3 & 27 & 4 \\ -4 & 11 & 2 \end{pmatrix}}_A = \underbrace{\begin{pmatrix} 0 & \frac{-4}{5} & \frac{3}{5} \\ \frac{-3}{5} & \frac{12}{25} & \frac{16}{25} \\ \frac{-4}{5} & \frac{-9}{25} & \frac{-12}{25} \end{pmatrix}}_Q \underbrace{\begin{pmatrix} 5 & -25 & -4 \\ 0 & 25 & -10 \\ 0 & 0 & 10 \end{pmatrix}}_R.$$

- c) Implemente en Matlab los métodos de ortogonalización de Gram-Schmidt y Householder, para calcular  $A = QR$ , compare los resultados numéricos con los encontrados en las partes (a) y (b).

**Solución:**

A continuación se implementaron los métodos de ortogonalización de Gram-Schmidt y Householder, para calcular  $A = QR$

```
1  % Definir la matriz A
2  A = [0 -20 14;
3       -3 27 4;
4       -4 11 2];
5
6  % Implementación del método de Gram-Schmidt
7  function [Q, R] = gram_schmidt(A)
8      [m, n] = size(A);
9      Q = zeros(m, n);
10     R = zeros(n, n);
11
12     for j = 1:n
13         v = A(:, j);
14         for i = 1:j-1
15             R(i, j) = Q(:, i)' * A(:, j);
16             v = v - R(i, j) * Q(:, i);
17         end
18         R(j, j) = norm(v);
19         Q(:, j) = v / R(j, j);
20     end
21 end
22
23 % Implementación del método de Householder
24 function [Q, R] = householder(A)
25     [m, n] = size(A);
26     Q = eye(m);
27     R = A;
28
29     for k = 1:n
30         x = R(k:m, k);
31         norm_x = norm(x);
32         e1 = zeros(length(x), 1);
33         e1(1) = norm_x;
34
35         % Evitar sign(0) = 0 asegurando una buena dirección de
36         % reflexión
37         sign_x1 = sign(x(1)) + (x(1) == 0);
```

```

37
38     v = x + sign_x1 * norm_x * e1;
39     v = v / norm(v);
40
41     % Construir la matriz de Householder
42     Hk = eye(m);
43     Hk(k:m, k:m) = eye(length(x)) - 2 * (v * v');
44
45     % Aplicar la transformación
46     R = Hk * R;
47     Q = Q * Hk';
48 end
49 end
50
51
52 % Aplicar los métodos a la matriz A
53 [Q_GS, R_GS] = gram_schmidt(A);
54 disp('Factorización QR usando Gram-Schmidt:');
55 disp('Q ='); disp(Q_GS);
56 disp('R ='); disp(R_GS);
57
58 [Q_HH, R_HH] = householder(A);
59 disp('Factorización QR usando Householder:');
60 disp('Q ='); disp(Q_HH);
61 disp('R ='); disp(R_HH);

```

>> QR	
Factorización QR usando Gram-Schmidt:	Factorización QR usando Householder:
Q =	Q =
0      -0.8000      0.6000 -0.6000      0.4800      0.6400 -0.8000      -0.3600      -0.4800	-0.9231      -0.2350      -0.3045 0.2308      -0.9717      0.0502 0.3077      0.0239      -0.9512
R =	R =
5.0000      -25.0000      -4.0000 0      25.0000      -10.0000 0      0      10.0000	-1.9231      28.0769      -11.3846 2.8195      -21.2741      -7.1285 3.6540      -3.0166      -5.9645

Al comparar los métodos de factorización QR mediante Gram-Schmidt y Householder, se observa que el primero produjo una descomposición sin problemas, mientras que el segundo mostró inconsistencias, en particular una matriz  $R$  que no es triangular superior. Esto sugiere que la implementación del método de Householder pudo haber sufrido errores numéricos acumulativos.

Las transformaciones de Householder están diseñadas para introducir ceros en la

parte inferior de la matriz mediante reflexiones ortogonales. Sin embargo, pequeños errores en la normalización de los vectores reflejados pueden amplificarse, afectando la estructura triangular de  $R$ . Además, la sensibilidad a la escala de los valores de la matriz original puede exacerbar estos errores. En este caso, la segunda columna de la matriz tiene valores significativamente mayores en magnitud que la primera, lo que puede haber llevado a problemas de redondeo y pérdida de precisión en los cálculos.

A pesar de que Householder es generalmente más estable que Gram-Schmidt, la implementación computacional puede verse afectada por la acumulación de errores de redondeo, especialmente si las operaciones no se manejan con la precisión adecuada. Esto explicaría por qué Gram-Schmidt, a pesar de ser teóricamente menos estable, produjo mejores resultados en este caso particular. **Nota:** Usando los algoritmos propios de matlab las 2 soluciones (QR y Gram-Schmidt) dan la misma solución teórica.

## Problema 2:

Descargue el archivo Datos.txt de la pagina del curso. En este encontrara un conjunto de 21 datos. Copie estos datos y calcule el polinomio de ajuste de grado 5  $p(x) = c_0 + c_1x + c_2x^2 + c_3x^3 + c_4x^4 + c_5x^5$  utilizando los métodos de ecuaciones normales y factorización  $QR$ . Compare sus resultados con los valores certificados  $c_i = 1$  para  $i = 0, 1, \dots, 5$ . Encuentre el residual  $\|Ac - y\|_2$  en cada caso, así como la diferencia relativa con respecto a los valores certificados. Escriba sus conclusiones.

### Solución:

1. Usando Cholesky.

```
1  format rational
2  function L = descomposicion_de_cholesky(A)
3      % Descomposición de Cholesky de una matriz A sin usar la
      % función chol
4      [n, ~] = size(A);
5      L = zeros(n);
6
7      for i = 1:n
8          for j = 1:i
9              if i == j
10                 L(i, j) = sqrt(A(i, i) - sum(L(i, 1:j-1) .^ 2));
11             else
12                 L(i, j) = (A(i, j) - sum(L(i, 1:j-1) .* L(j,
13                     1:j-1))) / L(j, j);
14             end
15         end
16     end
17
18     n = 5; %Grado del polinomio
19     x = (0:1:20)';
20     y = [75901, -204794, 204863, -204436, 253665, -200894, 214131,
21         -185192, 221249, -138370, 315911, -27644, 455253, 197434,
22         783995, 608816, 1370781, 1303798, 2205519, 2408860, 3444321]';
23
24     % Tomar tiempo de ejecución
25     tic;
26
27     % Construcción de la matriz Vandermonde
28     A = vander(x);
29     A = A(:, end-n:end); % Tomar las n+1 columnas necesarias
30
31     % Construcción de ecuaciones normales
```



```
30 B = A' * A;
31 z = A' * y;
32
33 % Descomposición de Cholesky
34 L = descomposicion_de_cholesky(B); % Matriz triangular inferior
35
36 % Resolver el sistema de ecuaciones
37 solichol = L \ z;
38 solchol = L' \ solichol;
39 tiempo_chol = toc
40
41 residual_cholesky = norm(A * solchol - y, 2)
42
43 % Mostrar solución final
44 disp(solchol);
45
46 xx = linspace(min(x), max(x), 100); % Puntos más finos para la
    curva
47 AA = xx(:).^(0:n); % Matriz de diseño para interpolación
48 yy = AA * solchol; % Evaluación del polinomio
49
50 plot(x, y, 'ro', 'MarkerSize', 8, 'DisplayName', 'Datos
    originales'); % Puntos originales
51 hold on;
52 plot(xx, yy, 'b-', 'LineWidth', 2, 'DisplayName', 'Polinomio
    ajustado'); % Polinomio
53 legend;
54 grid on;
55 title('Ajuste polinomial usando mínimos cuadrados con Cholesky');
56 xlabel('x');
57 ylabel('y');
58 hold off;
```

En dónde en un tiempo de  $\frac{29}{77957} \approx 0,000372$  nos lista los coeficientes  $c_i = 1$  con  $i = 0, 1, \dots, 5$  y la gráfica

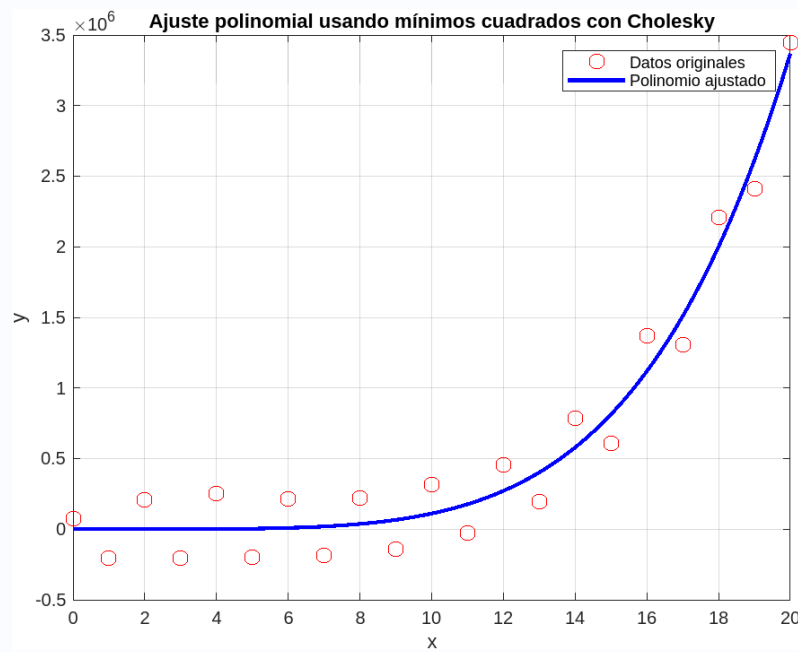


Figura 1: Usando el método de Cholesky.

## 2. Usando la factorización QR.

```
1  format rational
2  % Implementación del método de Householder
3  function [Q, R] = householder(A)
4      [m, n] = size(A);
5      Q = eye(m);
6      R = A;
7
8      for k = 1:n
9          x = R(k:m, k);
10         norm_x = norm(x);
11         e1 = zeros(length(x), 1);
12         e1(1) = norm_x;
13
14         % Evitar sign(0) = 0 asegurando una buena dirección de
           reflexión
15         sign_x1 = sign(x(1)) + (x(1) == 0);
16
17         v = x + sign_x1 * norm_x * e1;
18         v = v / norm(v);
19
20         % Construir la matriz de Householder
21         Hk = eye(m);
22         Hk(k:m, k:m) = eye(length(x)) - 2 * (v * v');
23
24         % Aplicar la transformación
25         R = Hk * R;
26         Q = Q * Hk';
27     end
28 end
29
30 n = 5; %Grado del polinomio
31 x = (0:1:20)';
32 y = [75901, -204794, 204863, -204436, 253665, -200894, 214131,
      -185192, 221249, -138370, 315911, -27644, 455253, 197434,
      783995, 608816, 1370781, 1303798, 2205519, 2408860, 3444321]';
33
34 % Tomar tiempo de ejecución
35 tic;
36
37 % Construcción de la matriz Vandermonde
38 A = vander(x);
39 A = A(:, end-n:end); % Tomar las n+1 columnas necesarias
```

```
40
41 % Factorización QR
42 [Q,R] = householder(A);
43
44 % Ecuaciones QR
45
46 b = Q'*y;
47 solqr = R \ b;
48 tiempo_qr = toc;
49
50 residual_qr = norm(A * solqr - y, 2);
51 disp('Residual QR:')
52 disp(residual_qr)
53
54 % Mostrar solución final
55 disp(solqr);
56
57 xx = linspace(min(x), max(x), 100); % Puntos más finos para la
    curva
58 AA = xx(:).^(0:n); % Matriz de diseño para interpolación
59 yy = AA * solqr; % Evaluación del polinomio
60
61 plot(x, y, 'ro', 'MarkerSize', 8, 'DisplayName', 'Datos
    originales'); % Puntos originales
62 hold on;
63 plot(xx, yy, 'b-', 'LineWidth', 2, 'DisplayName', 'Polinomio
    ajustado'); % Polinomio
64 legend;
65 grid on;
66 title('Ajuste polinomial usando factorización QR');
67 xlabel('x');
68 ylabel('y');
69 hold off;
```

En dónde en un tiempo de  $\frac{47}{125000}s \approx 0,000376s$  nos lista los coeficientes  $c_i = 1$  con  $i = 0, 1, \dots, 5$  y la gráfica

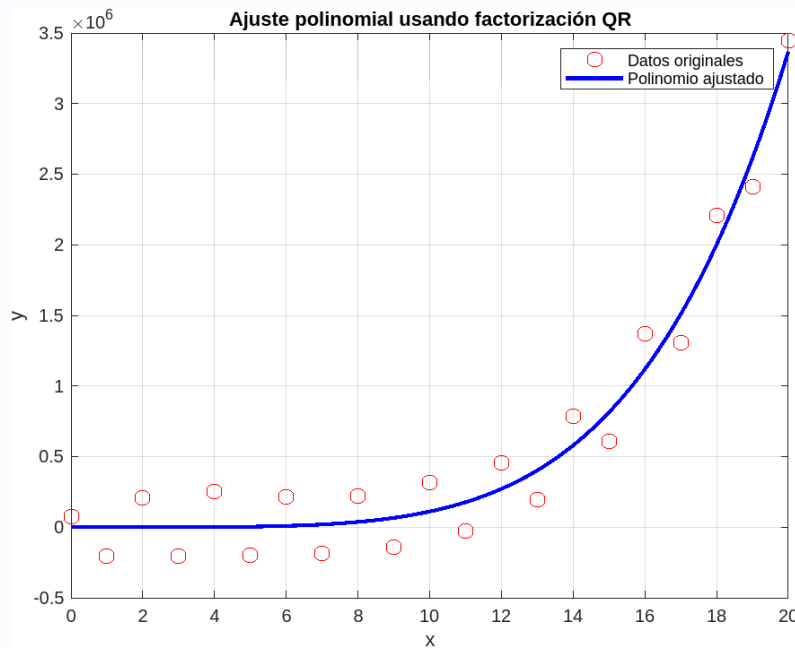


Figura 2: Usando el método de factorización QR.

**Conclusiones:** experimentalmente ambos métodos parecen dar de manera explícita los valores certificados, por lo que sería más objetivo hablar sobre el tiempo de ejecución del método en matlab, lo que vimos anteriormente que le da el punto al método de Cholesky, pues este tuvo un tiempo de ejecución menor al de la factorización QR. Por otro lado, la diferencia relativa con respecto a los valores certificados en ambos casos es de 0 al ser idénticos a los valores dados en el problema, veamos por otro lado el residual:

$$\|A_{chol} - y\|_2 = 914080$$

$$\|A_{QR} - y\|_2 = 914080$$

Esto se explica al ambos métodos llegar a la misma solución.

En fin, si bien en este experimento parece que el método de Cholesky es más eficiente que el método de factorización QR, recordemos que teóricamente el método QR tiene mayor estabilidad que Cholesky por el número de condición que se plantea en ambos casos, por lo que es válido pensar que para polinomios de grado mucho mayor lo sucedido en este experimento no siga sucediendo.

### Problema 3:

$$A = \begin{pmatrix} a & 0 & 0 \\ a\delta & a & 0 \\ 0 & a\delta & a \end{pmatrix}, \quad a < 0, \quad \delta > 0 \quad \text{y} \quad b = \begin{pmatrix} -1 \\ -1,1 \\ 0 \end{pmatrix}.$$

- (a) Obtenga el número de condición de  $A$ .

#### Solución:

Halleemos el número de condición  $\kappa_{\infty}(A) = \|A^{-1}\|_{\infty}\|A\|_{\infty}$ . Para ello, encontremos la inversa de  $A$ :

$$\begin{pmatrix} a & 0 & 0 & | & 1 & 0 & 0 \\ a\delta & a & 0 & | & 0 & 1 & 0 \\ 0 & a\delta & a & | & 0 & 0 & 1 \end{pmatrix} \begin{matrix} R_2 \rightarrow R_2 - \delta R_1, \\ R_3 \rightarrow R_3 - \delta R_2 \end{matrix}$$

$$\begin{pmatrix} a & 0 & 0 & | & 1 & 0 & 0 \\ 0 & a & 0 & | & -\delta & 1 & 0 \\ 0 & 0 & a & | & \delta^2 & -\delta & 1 \end{pmatrix} \begin{matrix} R_1 \rightarrow \frac{R_1}{a}, \\ R_2 \rightarrow \frac{R_2}{a}, \\ R_3 \rightarrow \frac{R_3}{a} \end{matrix}$$

$$\begin{pmatrix} 1 & 0 & 0 & | & \frac{1}{a} & 0 & 0 \\ 0 & 1 & 0 & | & -\frac{\delta}{a} & \frac{1}{a} & 0 \\ 0 & 0 & 1 & | & \frac{\delta^2}{a} & -\frac{\delta}{a} & \frac{1}{a} \end{pmatrix} \implies A^{-1} = \begin{pmatrix} \frac{1}{a} & 0 & 0 \\ -\frac{\delta}{a} & \frac{1}{a} & 0 \\ \frac{\delta^2}{a} & -\frac{\delta}{a} & \frac{1}{a} \end{pmatrix}$$

Ahora calculemos las normas correspondientes,

$$\begin{aligned} \kappa_{\infty}(A) &= \|A^{-1}\|_{\infty}\|A\|_{\infty} = \left( \left| \frac{\delta^2}{a} \right| + \left| \frac{-\delta}{a} \right| + \left| \frac{1}{a} \right| \right) (|a\delta| + |a|) \\ &= (\delta + 1)(\delta^2 + \delta + 1), \quad \delta > 0 \\ &= \delta^3 + 2\delta^2 + 2\delta + 1 \end{aligned}$$

Por consiguiente,  $\kappa_{\infty}(A) = \delta^3 + 2\delta^2 + 2\delta + 1$ , con  $\delta > 0$ .

- (b) Estudiar el condicionamiento del sistema  $Ax = b$  en función de los valores de  $\delta$ . Interprete su resultado.

#### Solución:

Como  $\kappa_{\infty}(A) = \delta^3 + 2\delta^2 + 2\delta + 1$  con  $\delta > 0$ , notemos que:

- Si  $\delta \rightarrow 0$ , entonces  $\kappa_{\infty}(A) \rightarrow 1$ .
- Si  $\delta \rightarrow \infty$ , entonces  $\kappa_{\infty}(A) \rightarrow \infty$ .

Por consiguiente, si  $\delta \ll 1$ , la matriz estará bien condicionada, lo que implica que

los cambios relativos en  $b$  controlan los cambios relativos en  $x$ . En cambio, si  $\delta \gg 1$ , el sistema será mal condicionado, es decir, pequeños cambios en los datos pueden generar grandes errores en la solución.

- (c) Si  $a = -1$ ,  $\delta = 0,1$  y se considera  $x^* = (1, \frac{9}{10}, 1)^T$  como solución aproximada del sistema  $Ax = b$  (sin obtener la solución exacta), determine un intervalo en el que esté comprendido el error relativo. ¿Es coherente con la respuesta dada en el apartado anterior?

### Solución:

Reemplazando  $a = -1$  y  $\delta = -0,1$  en la matriz y tomando  $x^* = (1, \frac{9}{10}, 1)^T$ , obtenemos:

$$\underbrace{\begin{pmatrix} -1 & 0 & 0 \\ -0,1 & -1 & 0 \\ 0 & -0,1 & -1 \end{pmatrix}}_A \underbrace{\begin{pmatrix} 1 \\ \frac{9}{10} \\ 1 \end{pmatrix}}_{x^*} = \underbrace{\begin{pmatrix} -1 \\ -1 \\ -\frac{109}{100} \end{pmatrix}}_{b^*} \approx \underbrace{\begin{pmatrix} -1 \\ -1,1 \\ 0 \end{pmatrix}}_b$$

Usemos la siguiente expresión para encontrar una cota para el error relativo:

$$\frac{1}{\kappa_\infty(A)} \frac{\|b - b^*\|}{\|b\|} \leq \frac{\|x - x^*\|}{\|x\|} \leq \kappa_\infty(A) \frac{\|b - b^*\|}{\|b\|}$$

Antes de reemplazar, calculemos algunos valores:

$$\begin{aligned} \kappa_\infty(A) &= \delta^3 + 2\delta^2 + 2\delta + 1 \\ &= \left(\frac{1}{10}\right)^3 + 2\left(\frac{1}{10}\right)^2 + 2\left(\frac{1}{10}\right) + 1 = \frac{1221}{1000} \\ \|b\|_\infty &= |-1,1| = 1,1 \\ \|b^*\|_\infty &= \left|-\frac{109}{100}\right| = \frac{109}{100} \end{aligned}$$

Reemplazando, la cota del error relativo queda de la siguiente forma:

$$\begin{aligned} \frac{1}{1,221} \frac{|1,1 - 1,09|}{|1,1|} &\leq \frac{\|x - x^*\|}{\|x\|} \leq (1,221) \frac{|1,1 - 1,09|}{|1,1|} \\ 0,00744 &\leq \frac{\|x - x^*\|}{\|x\|} \leq 0,0111 \end{aligned}$$

Es decir, el error relativo está entre 0,74 % y 1,11 %, lo cual considero coherente con los resultados encontrados en el ítem anterior. En este caso, se está tomando un  $\delta \ll 1$ , por lo que el error es pequeño y, en consecuencia, la aproximación es buena.

- (d) Si  $a = -1$  y  $\delta = 0,1$ , ¿es convergente el método de Jacobi aplicado a la resolución del sistema  $Ax = b$ ? Realice tres iteraciones a partir de  $x^* = (0, 0, 0)^T$ .

### Solución:

Sea

$$A = \begin{pmatrix} -1 & 0 & 0 \\ -0,1 & -1 & 0 \\ 0 & -0,1 & -1 \end{pmatrix}$$

Reescribamos la matriz como  $A = D + L + U$ , donde  $D$  es la matriz diagonal de  $A$ ,  $L$  es la matriz con los elementos debajo de la diagonal, y  $U$  es la matriz con los elementos por encima de la diagonal:

$$A = \begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ -0,1 & 0 & 0 \\ 0 & -0,1 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Ahora, calculemos la matriz de iteración  $T_J$  para el método iterativo de Jacobi, dada por:

$$T_J = -D^{-1}(L + U).$$

Dado que en este caso  $U = 0$ , la expresión se reduce a  $T_J = -D^{-1}L$ :

$$T_J = -D^{-1}L = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 \\ -0,1 & 0 & 0 \\ 0 & -0,1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 \\ 0,1 & 0 & 0 \\ 0 & 0,1 & 0 \end{pmatrix}$$

Como  $\|T_J\|_\infty = 0,1 < 1$ , el método de Jacobi converge según el criterio de la norma.

A continuación, escribamos explícitamente la forma matricial de las iteraciones, que se define como:

$$x^{(k+1)} = -D^{-1}Lx^{(k)} + D^{-1}b.$$

Sustituyendo los valores:

$$\begin{pmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \\ x_3^{(k+1)} \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 \\ -0,1 & 0 & 0 \\ 0 & -0,1 & 0 \end{pmatrix} \begin{pmatrix} x_1^{(k)} \\ x_2^{(k)} \\ x_3^{(k)} \end{pmatrix} + \begin{pmatrix} 1 \\ 1,1 \\ 0 \end{pmatrix}.$$

Ahora, realicemos tres iteraciones del método de Jacobi:

$$\begin{pmatrix} x_1^{(1)} \\ x_2^{(1)} \\ x_3^{(1)} \end{pmatrix} = \begin{pmatrix} 1 \\ 1,1 \\ 0 \end{pmatrix}.$$

$$\begin{pmatrix} x_1^{(2)} \\ x_2^{(2)} \\ x_3^{(2)} \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ -0,11 \end{pmatrix}.$$



$$\begin{pmatrix} x_1^{(3)} \\ x_2^{(3)} \\ x_3^{(3)} \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ -0,1 \end{pmatrix}.$$

Notemos que:

$$Ax^{(3)} = \begin{pmatrix} -1 & 0 & 0 \\ -0,1 & -1 & 0 \\ 0 & -0,1 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ -0,1 \end{pmatrix} = \begin{pmatrix} -1 \\ -1,1 \\ 0 \end{pmatrix}.$$

Por lo tanto, el método de Jacobi no solo converge, sino que lo hace rápidamente y en solo tres iteraciones alcanza la solución exacta.

**Problema 4:**

Sea  $A \in \mathbb{R}^{n \times n}$ . Probar que  $\lambda = 1$  es un vector propio de la matriz de iteración del método de Jacobi (o Gauss-Seidel) de  $A$  si y solo si  $A$  no es invertible.

**Solución:**

Sea  $A = D - L - U$ , donde  $D$  es la matriz diagonal de  $A$ ,  $L$  es la matriz que contiene los elementos debajo de la diagonal con signo invertido, y  $U$  es la matriz con los elementos por encima de la diagonal también con signo invertido.

Supongamos que  $\lambda = 1$  es un valor propio de la matriz de iteración de Jacobi,  $T_J = -D^{-1}(L + U)$ . Por definición, esto significa que:

$$\begin{aligned} 0 &= \det(D^{-1}(L + U) - I) \\ &= \det(D^{-1}(L + U) - D^{-1}D) \\ &= \det(D^{-1}(L + U - D)) \\ &= \det(D^{-1}) \det(-(D - L - U)) \\ &= \det(D^{-1})(-1)^n \det(A). \end{aligned}$$

Dado que  $D$  es invertible, se sigue que  $\det(A) = 0$ , lo que implica que  $A$  no es invertible. Recíprocamente, si  $A$  no es invertible, entonces  $\det(A) = 0$ . A partir de la ecuación anterior, esto implica que  $\det(D^{-1}(L + U) - I) = 0$ , lo que prueba que  $\lambda = 1$  es un valor propio de la matriz de iteración del método de Jacobi.

**Problema 5:**

Considere el sistema  $Ax = b$  donde

$$A := \begin{pmatrix} 3 & -1 & -1 & 0 & 0 \\ -1 & 4 & 0 & -2 & 0 \\ -1 & 0 & 3 & -1 & 0 \\ 0 & -2 & -1 & 5 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{pmatrix} \quad b := \begin{pmatrix} 2 \\ -26 \\ 3 \\ 47 \\ -10 \end{pmatrix}$$

1. Investigue la convergencia de los métodos de Jacobi, Gauss-Seidel y sobrerelajación.

**Solución:**

- a) Convergencia en Jacobi.

Note que  $A$  es una matriz diagonal dominante, por lo que se puede asegurar que el método converge desde cualquier solución inicial  $\mathbf{x}^0$ .

- b) Convergencia en Gauss-Seidel.

Note que  $T_{GS}$  es:

$$T_{GS} = (D - L)^{-1}U = \begin{pmatrix} 0 & \frac{1}{3} & \frac{1}{3} & 0 & 0 \\ 0 & \frac{1}{12} & \frac{1}{12} & \frac{1}{2} & 0 \\ 0 & \frac{1}{9} & \frac{1}{9} & \frac{1}{3} & 0 \\ 0 & \frac{1}{18} & \frac{1}{18} & \frac{4}{15} & \frac{1}{5} \\ 0 & \frac{1}{36} & \frac{1}{36} & \frac{2}{15} & \frac{1}{10} \end{pmatrix}$$

Luego el radio espectral de  $T_{GS}$  es  $1061/2071 \approx 0,51 < 1$ , por lo que sabemos que la sucesión converge con cualquier  $\mathbf{c}$  y  $\mathbf{x}^0$ .

Estos cálculos se ayudaron de matlab

```

1  % Definir la matriz
2  A = [ 3  -1  -1  0  0;
3        -1  4  0  -2  0;
4        -1  0  3  -1  0;
5         0  -2  -1  5  -1;
6         0  0  0  -1  2];
7  D = [ 3  0  0  0  0;
8        0  4  0  0  0;
9        0  0  3  0  0;
10       0  0  0  5  0;
11       0  0  0  0  2];
12  L = [ 0  0  0  0  0;
13        1  0  0  0  0;
14        1  0  0  0  0;
15        0  2  1  0  0;
16        0  0  0  1  0];
17  U = [ 0  1  1  0  0;
18        0  0  0  2  0;
19        0  0  0  1  0;
20        0  0  0  0  1;
21        0  0  0  0  0];
22
23  TGS = inv(D-L)*U;
24  latex_codigo = latex(sym(TGS));
25
26  % Obtener valores propios
27  valores_propiosGS = eig(TGS);
28
29  % Calcular el radio espectral
30  radio_espectralGS = max(abs(valores_propiosGS));
31
32  % Mostrar resultado
33  disp('Radio espectral en Gauss-Seidel:')
34  disp(radio_espectral)

```

c) Convergencia en SOR.

Note que  $A$  es una matriz simétrica definida positiva, ya que el polinomio característico es  $P_A(x) = 167 - 373x + 295x^2 - 105x^3 + 17x^4 - x^5$ , el cual tiene raíces reales, positivas (esto se puede comprobar utilizando graficadoras o un programa de cómputo de factorización, omitiremos ese paso ya que lo único importante es saber que son reales), por lo que sabemos que si tomamos  $0 < w < 2$  entonces el método SOR converge a la única solución del sistema  $Ax = b$  desde cualquier solución inicial  $\mathbf{x}^0$ .

2. ¿Cuál es el radio espectral de la matriz  $J$  y de la matriz  $S$ ?

### Solución:

a) Jacobi.

La matriz  $T_J$  es:

$$\begin{pmatrix} 0 & \frac{1}{3} & \frac{1}{3} & 0 & 0 \\ \frac{1}{4} & 0 & 0 & \frac{1}{2} & 0 \\ \frac{1}{3} & 0 & 0 & \frac{1}{3} & 0 \\ 0 & \frac{2}{5} & \frac{1}{5} & 0 & \frac{1}{5} \\ 0 & 0 & 0 & \frac{1}{2} & 0 \end{pmatrix}$$

Luego el radio espectral de  $T_J$  es  $\frac{1040}{1453} \approx 0,72 < 1$ .

b) SOR.

La matriz  $T_{SOR}$  que depende de  $w$  es:

$$\begin{pmatrix} 1-w & \frac{w}{3} & \frac{w}{3} & 0 & 0 \\ -\frac{w(3w-3)}{12} & \frac{w^2}{12} - w + 1 & \frac{w^2}{12} & \frac{w}{2} & 0 \\ -\frac{w(3w-3)}{9} & \frac{w^2}{9} & \frac{w^2}{9} - w + 1 & \frac{w}{3} & 0 \\ -\frac{w^2(3w-3)}{18} & \frac{w^3}{18} - \frac{w(4w-4)}{10} & \frac{w^3}{18} - \frac{w(3w-3)}{15} & \frac{4w^2}{15} - w + 1 & \frac{w}{5} \\ -\frac{w^3(3w-3)}{36} & \frac{w^4}{36} - \frac{w^2(4w-4)}{20} & \frac{w^4}{36} - \frac{w^2(3w-3)}{30} & \frac{2w^3}{15} - \frac{w(5w-5)}{10} & \frac{w^2}{10} - w + 1 \end{pmatrix}$$

así, usemos el  $w^*$  encontrado en el siguiente item para calcular el radio espectral, por lo que  $T_{SOR}(w^*)$  sería:

$$\begin{pmatrix} -\frac{17}{100} & \frac{39}{100} & \frac{39}{100} & 0 & 0 \\ -\frac{1989}{40000} & -\frac{2237}{40000} & \frac{4563}{40000} & \frac{117}{200} & 0 \\ -\frac{663}{10000} & \frac{1521}{10000} & -\frac{179}{10000} & \frac{39}{100} & 0 \\ -\frac{698697453389513}{18014398509481984} & \frac{678674449446225}{72057594037927936} & \frac{7090251080549995}{144115188075855872} & \frac{1219}{6250} & \frac{117}{500} \\ -\frac{3269904081862921}{144115188075855872} & \frac{794049105852083}{144115188075855872} & \frac{8295593764243493}{288230376151711744} & \frac{4110828093788559}{36028797018963968} & -\frac{3311}{100000} \end{pmatrix}$$

Luego el radio espectral de  $T_{SOR}(w^*)$  es  $\frac{918}{3797} \approx 0,24 < 1$ .

3. Aproxime con dos cifras decimales el parámetro de sobrerrelajación  $w^*$ .

### Solución:

Utilizemos la fórmula:

$$w^* = \frac{2}{1 + \sqrt{1 - \rho(J)^2}} \approx 1,17$$

4. ¿Qué reducción en el costo operacional ofrece el método de sobrerrelajación con el parámetro

$w^*$ , en comparación con el método de Gauss-Seidel?

### Solución:

Note que como  $w^* \approx 1,17$ , o sea  $1 < w^* < 2$ , es decir el método sería de sobrerelajación que debería de acelerar la convergencia del método original (Gauss-Seidel).

5. ¿Cuántas iteraciones más requiere el método de Gauss-Seidel para lograr una precisión mejorada en una cifra decimal? ¿Cuántas necesita el método de sobrerelajación con  $w^*$ ?

### Solución:

a) Gauss-Seidel.

Aquí para llegar al  $x$  esperado realizó 9 iteraciones, en donde

$$x = \begin{pmatrix} 4306 \\ 1367 \\ 615 \\ 617 \\ 4142 \\ 635 \\ 2205 \\ 164 \\ 3845 \\ 328 \end{pmatrix}$$

b) SOR.

Aquí para llegar al  $x$  esperado realizó 7 iteraciones, en donde

$$x = \begin{pmatrix} 1867 \\ 585 \\ 509 \\ 493 \\ 1219 \\ 186 \\ 5173 \\ 384 \\ 2711 \\ 231 \end{pmatrix}$$

Con lo que podemos concluir que Gauss-Seidel realizó 2 iteraciones más que *SOR* y el método de sobrerelajación con  $w^*$  necesitó 7 iteraciones.

Para realizar el ejercicio se usó el siguiente script de matlab

```

1 % Definir la matriz
2 A = [ 3  -1  -1  0  0;
3       -1  4   0  -2  0;
4       -1  0   3  -1  0;
5         0  -2  -1  5  -1;
6         0  0   0  -1  2];
7 b = [ 2;
8      -26;
9        3;
10       47;
11       10;
```

```
12 ];
13 D = [ 3  0  0  0  0;
14       0  4  0  0  0;
15       0  0  3  0  0;
16       0  0  0  5  0;
17       0  0  0  0  2];
18 L = [ 0  0  0  0  0;
19       1  0  0  0  0;
20       1  0  0  0  0;
21       0  2  1  0  0;
22       0  0  0  1  0];
23 U = [ 0  1  1  0  0;
24       0  0  0  2  0;
25       0  0  0  1  0;
26       0  0  0  0  1;
27       0  0  0  0  0];
28 % Jacobi y Gauss-Seidel
29
30 TGS = inv(D-L)*U;
31 TJ = inv(D)*(U+L);
32
33 latex_codigo = latex(sym(TGS));
34 latex_codigo = latex(sym(TJ));
35
36 % Obtener valores propios
37 valores_propiosGS = eig(TGS);
38 valores_propiosJ = eig(TJ);
39
40 % Calcular el radio espectral
41 radio_espectralGS = max(abs(valores_propiosGS));
42 radio_espectralJ = max(abs(valores_propiosJ));
43
44 % Mostrar resultado
45 disp('Radio espectral en Gauss-Seidel:')
46 disp(radio_espectralGS)
47 disp('Radio espectral en Jacobi:')
48 disp(radio_espectralJ)
49
50 % SOR
51 w = 2 / (1+sqrt(1-(radio_espectralJ^2)));
52 woptimo = 1.17;
53 TSOR = inv(D-woptimo*L)*((1-woptimo)*D+woptimo*U);
54 latex_codigo = latex(sym(TSOR));
55 valores_propiosSOR = eig(TSOR);
56 radio_espectralSOR = max(abs(valores_propiosSOR));
57 disp('Radio espectral en SOR:')
```

```
58 disp(radio_espectralSOR)
59
60 % Encontremos el valor de x en ambos métodos, Gauss-Seidel y SOR con
    woptimo
61
62 cGS = inv(D-L)*b;
63 cSOR = woptimo*inv(D-woptimo*L)*b
64
65 % Gauss-Seidel
66
67 x = zeros(5,1); % x0 inicial
68 x_anterior = x+1; % Para comparar iteraciones
69 tolerancia = 0.05; % Para una cifra decimal
70 iteraciones = 0;
71
72 % Iterar hasta que cada componente de x tenga precisión de una cifra
    decimal
73 while any(abs(x - x_anterior) > tolerancia)
74     x_anterior = x; % Guardar el x anterior
75     x = TGS * x + cGS; % Nueva iteración
76     iteraciones = iteraciones + 1;
77 end
78
79 iteraciones
80 x
81
82 % SOR
83
84 x = zeros(5,1); % x0 inicial
85 x_anterior = x+1; % Para comparar iteraciones
86 tolerancia = 0.05; % Para una cifra decimal
87 iteraciones = 0;
88
89 % Iterar hasta que cada componente de x tenga precisión de una cifra
    decimal
90 while any(abs(x - x_anterior) > tolerancia)
91     x_anterior = x; % Guardar el x anterior
92     x = TSOR * x + cSOR; % Nueva iteración
93     iteraciones = iteraciones + 1;
94 end
95
96 iteraciones
97 x
```



## Problema 6:

Reconstrucción de imágenes a partir de sus bordes.

El operador laplaciano en  $2D$  se define como:

$$\nabla^2 u(x, y) = \frac{\partial u}{\partial x^2} + \frac{\partial u}{\partial y^2}$$

donde  $u(x, y)$  es una función que representa la intensidad de los píxeles de la imagen en el dominio espacial. El operador Laplaciano se utiliza para detectar bordes porque responde a las variaciones de la intensidad de los píxeles en la vecindad de cada punto. En áreas de la imagen donde la intensidad varía rápidamente (bordes), el Laplaciano tiene un valor alto, mientras que en áreas homogéneas (sin bordes) el Laplaciano es cercano a cero. El proceso de detección de bordes implica calcular el Laplaciano de la imagen  $u(x, y)$ , lo que da como resultado un mapa de bordes.

En este ejercicio estamos interesados en reconstruir la imagen original  $u(x, y)$  a partir de los bordes detectados.

Para esto debemos resolver la ecuación de Poisson en  $2D$

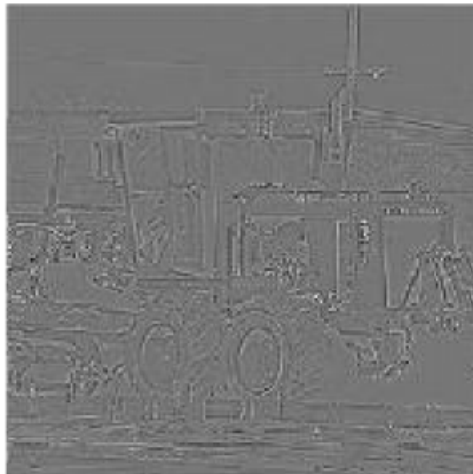
$$\nabla^2 u(x, y) = f(x, y) \quad (2)$$

donde  $f(x, y)$  es el mapa de bordes.

El problema 2 puede ser discretizado en un sistema de ecuaciones lineales de la forma:

$$Au = f$$

donde  $A$  es una matriz que representa el operador Laplaciano discreto,  $u$  es el vector que contiene los valores de la imagen original en cada píxel (a reconstruir), y  $f$  es el vector que contiene los valores de los bordes detectados.



**resolución 240 x 240**



**resolución 276 x 276**

**Instrucciones:**

Su tarea consiste en reconstruir las imágenes originales. En la pagina web del curso encuentran los archivos en Matlab `bordes1.mat` y `bordes2.mat` que contienen los datos de cada fotografía. Siga los pasos descritos a continuación:

**Solución:**

Usando la aproximación de diferencias finitas centrales para la segunda derivada:

$$\begin{aligned}\frac{\partial u}{\partial x^2} &\approx \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2}, \\ \frac{\partial u}{\partial y^2} &\approx \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h^2}.\end{aligned}$$

Sumando ambas expresiones obtenemos:

$$\frac{u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j}}{h^2} = f_{i,j}$$

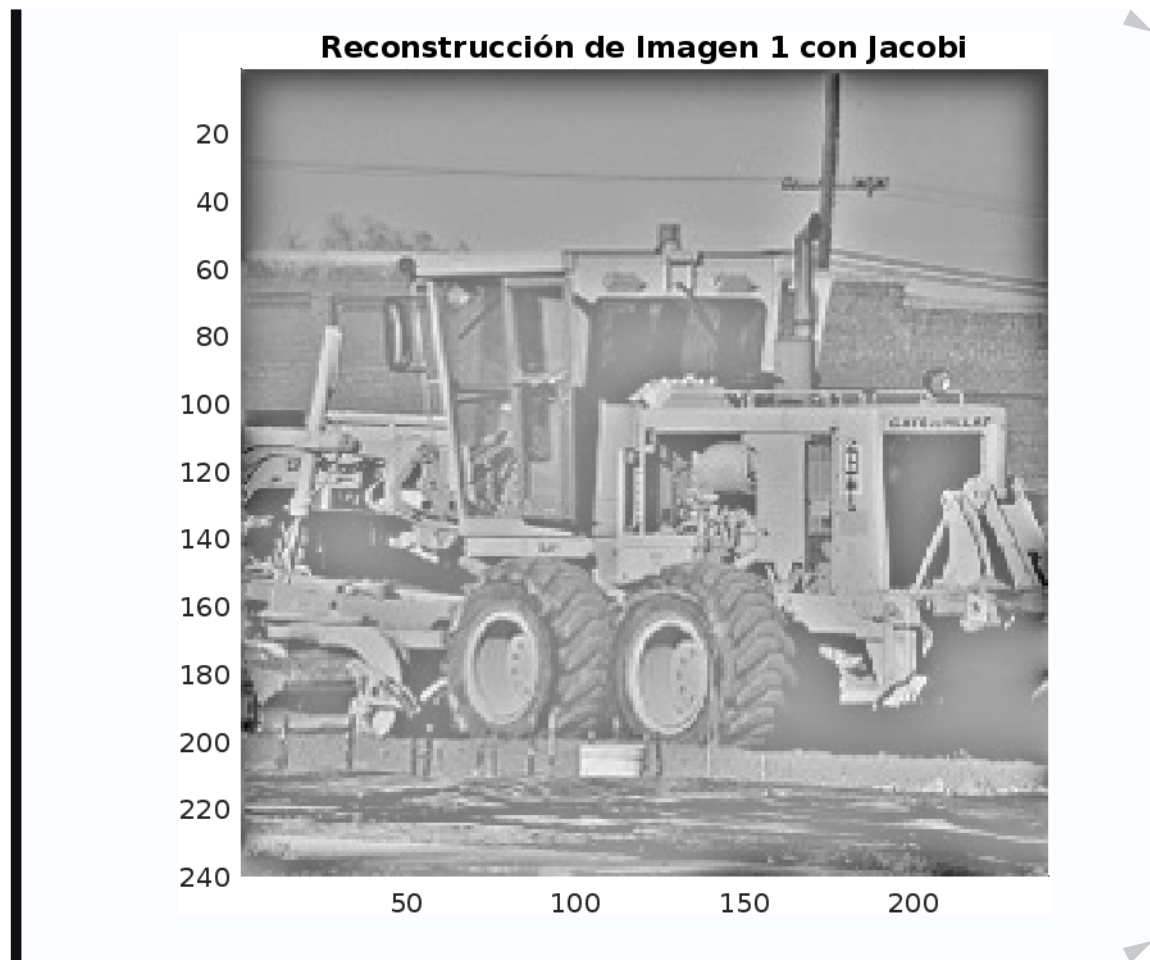
lo que nos ayudará a construir la matriz  $A$  en el ejercicio.

**Nota:** Profesor Mauricio, el tamaño de la matriz parece ser un gran problema para Matlab a la hora de calcular inversas en los métodos de Gauss-Seidel y SOR, por lo que nos fue necesaria buscar una forma de optimizar el proceso, a lo que decidimos investigar y encontrar una fuente en la que dan las fórmulas exactas de manera que "de alguna manera. optimiza el tiempo de o al menos facilita a matlab en el cálculo de los valores, dejamos la cita de la información a la que nos referimos Solving-2D-Poissons-Equation-Using-Finite-Difference-Method-and-Iterative-Solvers, esperamos entienda la situación, de lo contrario simplemente usábamos el código que usamos en Jacobi modificando las  $T$  y las  $c$ . Ahora veamos esto en la practica, daremos 100 iteraciones a cada método y daremos el tiempo tomado para ejecutarse. Siendo así, a continuación dejaremos los códigos y las gráficas generadas.

1. Imágen 1 (Jacobi)  $\frac{137}{4514}s \approx 0,0304s$ .

```
1  format rational
2  %% Construcción de la matriz A usando diferencias finitas
3  function A = construir_matriz_A(N, M)
4      num_pixeles = N * M;
5      A = sparse(num_pixeles, num_pixeles); % Matriz dispersa
6
7      for i = 1:N
8          for j = 1:M
9              k = (i-1) * M + j; % Índice en el vector
10
11              A(k, k) = -4; % Píxel central
12
13              if j < M % Vecino derecho
14                  A(k, k + 1) = 1;
15              end
16              if j > 1 % Vecino izquierdo
17                  A(k, k - 1) = 1;
18              end
19              if i < N % Vecino inferior
20                  A(k, k + M) = 1;
21              end
22              if i > 1 % Vecino superior
23                  A(k, k - M) = 1;
24              end
25          end
26      end
27  end
28
29  %% Visualizar la solución
30  function visualizar(u, N, M, titulo)
31      imagen = reshape(u, [N, M]); % Volver a forma de matriz
32      figure;
33      surf(imagen), shading flat, axis ij, axis equal, view(2);
34      colormap gray;
35      title(titulo);
36  end
37
38  %% Cargar datos de los bordes
39  data1 = load('bordes1.mat'); % Cargar archivo
40
41  f = data1.bordes1; % Extraer matriz de bordes
42
43  [N, M] = size(f); % Dimensiones de la imagen 1
```

```
44
45 % Convertir la matriz en un vector columna
46 f_vec = f(:);
47
48 A = construir_matriz_A(N, M);
49
50 %% Métodos Iterativos
51
52 D = diag(diag(A)); % Matriz diagonal
53 L = -tril(A,-1); % Parte triangular inferior estricta con signo
    negativo
54 U = -triu(A,1); % Parte triangular superior estricta con signo
    negativo
55
56 % Jacobi
57
58 TJ = inv(D)*(U+L);
59 cJ = inv(D)*f_vec;
60 [f_size, basura] = size(f_vec);
61
62 u = zeros(f_size,1); % u0 inicial
63 u_anterior = u+1; % Para comparar iteraciones
64 iteraciones = 0;
65
66 % Iterar hasta que haga 100 iteraciones del método
67
68 tic;
69 while (100 >= iteraciones)
70     u_anterior = u; % Guardar el u anterior
71     u = TJ * u + cJ; % Nueva iteración
72     iteraciones = iteraciones + 1;
73 end
74
75 tiempo = toc;
76
77 disp('Tiempo tomado en el método Jacobi:')
78 disp(tiempo);
79 visualizar(u, N, M, 'Reconstrucción de Imagen 1 con Jacobi');
```



2. Imágen 1 (Gauss-Seidel)  $\frac{1990}{49931}s = 0,03985s$ .

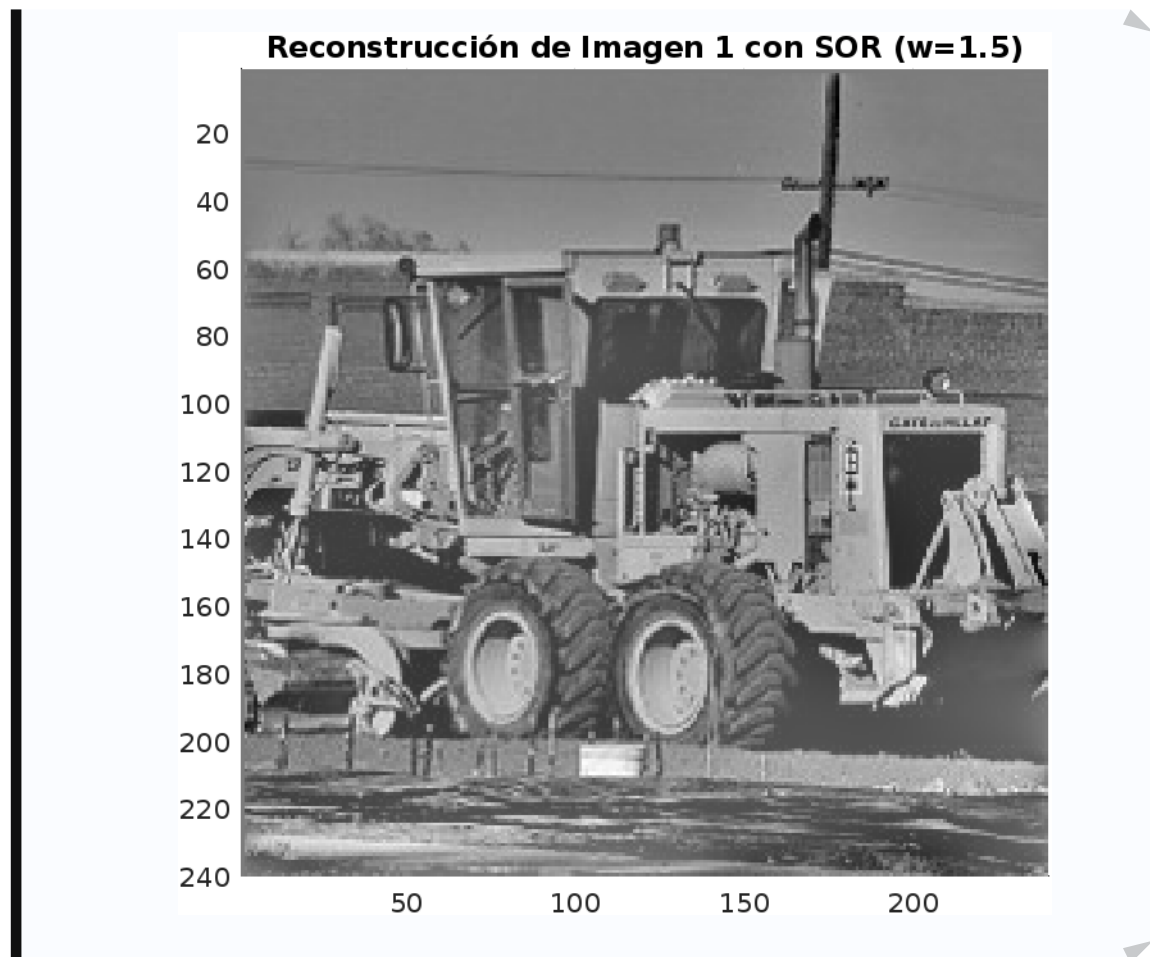
```
1  format rational
2  %% Visualizar la solución
3  function visualizar(u, N, M, titulo)
4      imagen = reshape(u, [N, M]); % Volver a forma de matriz
5      figure;
6      surf(imagen), shading flat, axis ij, axis equal, view(2);
7      colormap gray;
8      title(titulo);
9  end
10
11 %% Cargar datos de los bordes
12 data1 = load('bordes1.mat'); % Cargar archivo
13
14 f = data1.bordes1; % Extraer matriz de bordes
15
16 [N, M] = size(f); % Dimensiones de la imagen 1
17
18
19 %% Métodos Iterativos
20
21 % Inicialización de la solución
22 u = zeros(M,N); % x0 inicial
23 iteraciones = 0;
24
25 tic;
26 while (100 >= iteraciones)
27     u_anterior = u;
28     for i=2:M-1
29         for j=2:N-1
30             u(i, j) = (1/4) * (-f(i, j) + u(i-1, j) +
31                             u_anterior(i+1, j) + u(i, j-1) + u_anterior(i,
32                             j+1));
33         end
34     end
35     iteraciones = iteraciones + 1;
36 end
37
38 tiempo = toc;
39 disp("Tiempo tomado en el método de Gauss-Seidel");
40 disp(tiempo);
41 visualizar(u, N, M, 'Reconstrucción de Imagen 1 con
42     Gauss-Seidel');
```

**Reconstrucción de Imagen 1 con Gauss-Seidel**

3. Imágen 1 (SOR)  $\frac{396}{5729}s \approx 0,06912s$ .

```
1  format rational
2  %% Visualizar la solución
3  function visualizar(u, N, M, titulo)
4      imagen = reshape(u, [N, M]); % Volver a forma de matriz
5      figure;
6      surf(imagen), shading flat, axis ij, axis equal, view(2);
7      colormap gray;
8      title(titulo);
9  end
10
11 %% Cargar datos de los bordes
12 data1 = load('bordes1.mat'); % Cargar archivo
13
14 f = data1.bordes1; % Extraer matriz de bordes
15 w = 1.5;
16 [N, M] = size(f); % Dimensiones de la imagen 1
17
18 %% Métodos Iterativos
19
20 % Inicialización de la solución
21 u = zeros(M,N); % x0 inicial
22 ugs = zeros(M,N);
23 iteraciones = 0;
24
25
26 tic;
27 while (100 >= iteraciones)
28     u_anterior = u;
29     for i=2:M-1
30         for j=2:N-1
31             ugs(i, j) = (1/4) * (-f(i, j) + u(i-1, j) +
32                                 u_anterior(i+1, j) + u(i, j-1) + u_anterior(i,
33                                     j+1));
34             u(i, j) = (1-w)*u(i,j)+w*ugs(i,j);
35         end
36     end
37     iteraciones = iteraciones + 1;
38 end
39
40 tiempo = toc;
41 disp("Tiempo tomado en el método de SOR con w=1.5");
42 disp(tiempo);
43 visualizar(u, N, M, 'Reconstrucción de Imagen 1 con SOR (w=1.5)');
```

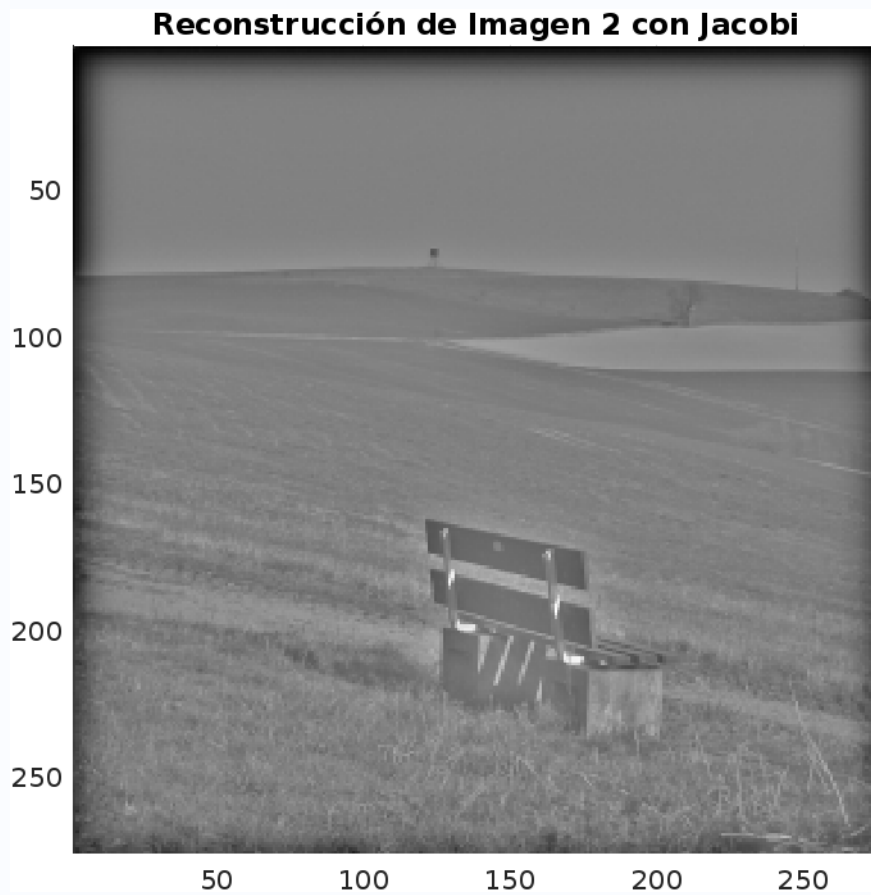




4. Imágen 2 (Jacobi)  $\frac{101}{2697}s \approx 0,03745s$ .

```
1  format rational
2  %% Construcción de la matriz A usando diferencias finitas
3  function A = construir_matriz_A(N, M)
4      num_pixeles = N * M;
5      A = sparse(num_pixeles, num_pixeles); % Matriz dispersa
6
7      for i = 1:N
8          for j = 1:M
9              k = (i-1) * M + j; % Índice en el vector
10
11              A(k, k) = -4; % Píxel central
12
13              if j < M % Vecino derecho
14                  A(k, k + 1) = 1;
15              end
16              if j > 1 % Vecino izquierdo
17                  A(k, k - 1) = 1;
18              end
19              if i < N % Vecino inferior
20                  A(k, k + M) = 1;
21              end
22              if i > 1 % Vecino superior
23                  A(k, k - M) = 1;
24              end
25          end
26      end
27  end
28
29  %% Visualizar la solución
30  function visualizar(u, N, M, titulo)
31      imagen = reshape(u, [N, M]); % Volver a forma de matriz
32      figure;
33      surf(imagen), shading flat, axis ij, axis equal, view(2);
34      colormap gray;
35      title(titulo);
36  end
37
38  %% Cargar datos de los bordes
39  data2 = load('bordes2.mat'); % Cargar archivo
40
41  f = data2.bordes2; % Extraer matriz de bordes
42
43  [N, M] = size(f); % Dimensiones de la imagen 2
```

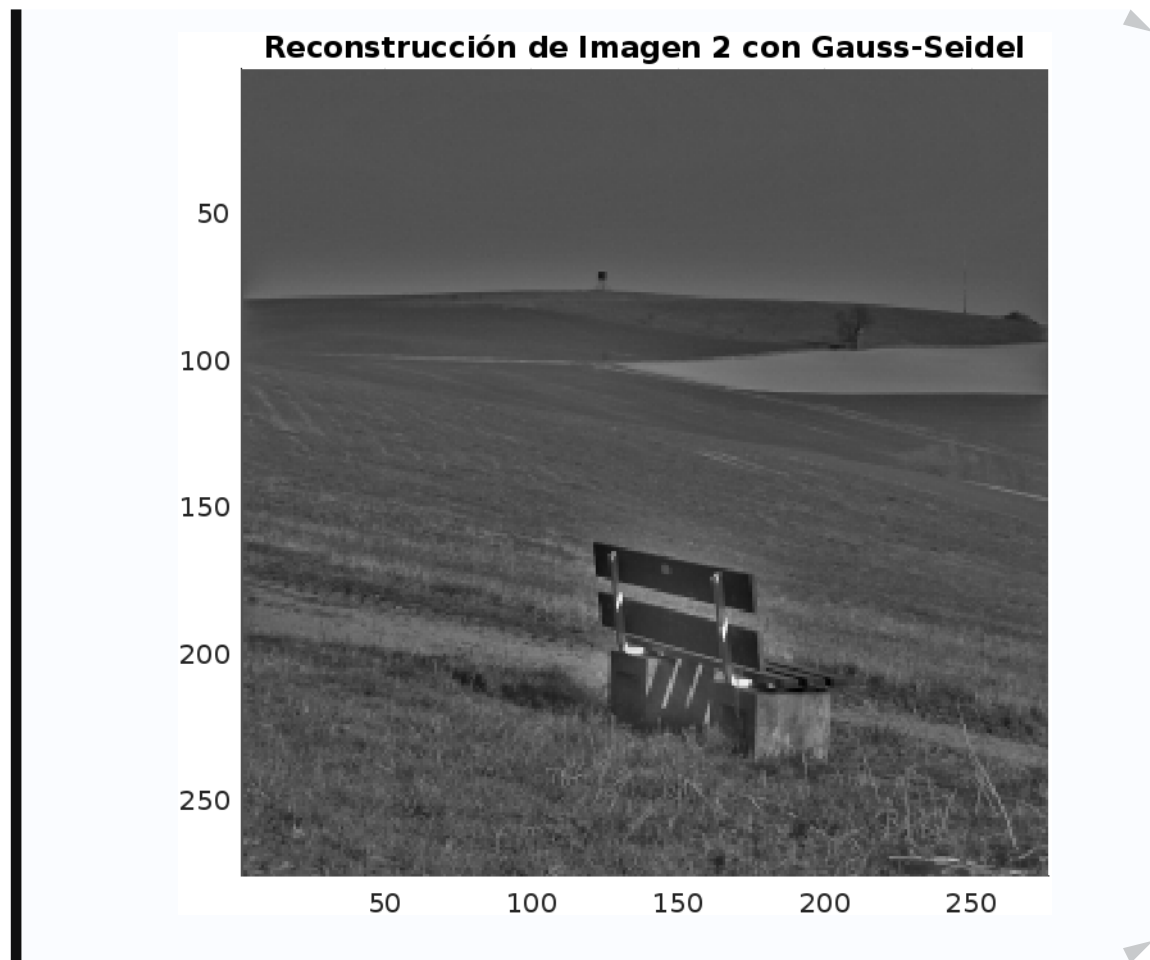
```
44
45 % Convertir la matriz en un vector columna
46 f_vec = f(:);
47
48 A = construir_matriz_A(N, M);
49
50 %% Métodos Iterativos
51
52 D = diag(diag(A)); % Matriz diagonal
53 L = -tril(A,-1); % Parte triangular inferior estricta con signo
    negativo
54 U = -triu(A,1); % Parte triangular superior estricta con signo
    negativo
55
56 % Jacobi
57
58 TJ = inv(D)*(U+L);
59 cJ = inv(D)*f_vec;
60 [f_size, basura] = size(f_vec);
61
62 u = zeros(f_size,1); % u0 inicial
63 u_anterior = u+1; % Para comparar iteraciones
64 iteraciones = 0;
65
66 % Iterar hasta que haga 100 iteraciones del método
67
68 tic;
69 while (100 >= iteraciones)
70     u_anterior = u; % Guardar el u anterior
71     u = TJ * u + cJ; % Nueva iteración
72     iteraciones = iteraciones + 1;
73 end
74
75 tiempo = toc;
76
77 disp('Tiempo tomado en el método Jacobi:')
78 disp(tiempo);
79 visualizar(u, N, M, 'Reconstrucción de Imagen 2 con Jacobi');
```



5. Imágen 2 (Gauss-Seidel)  $\frac{159}{2323}s = 0,06845s$ .

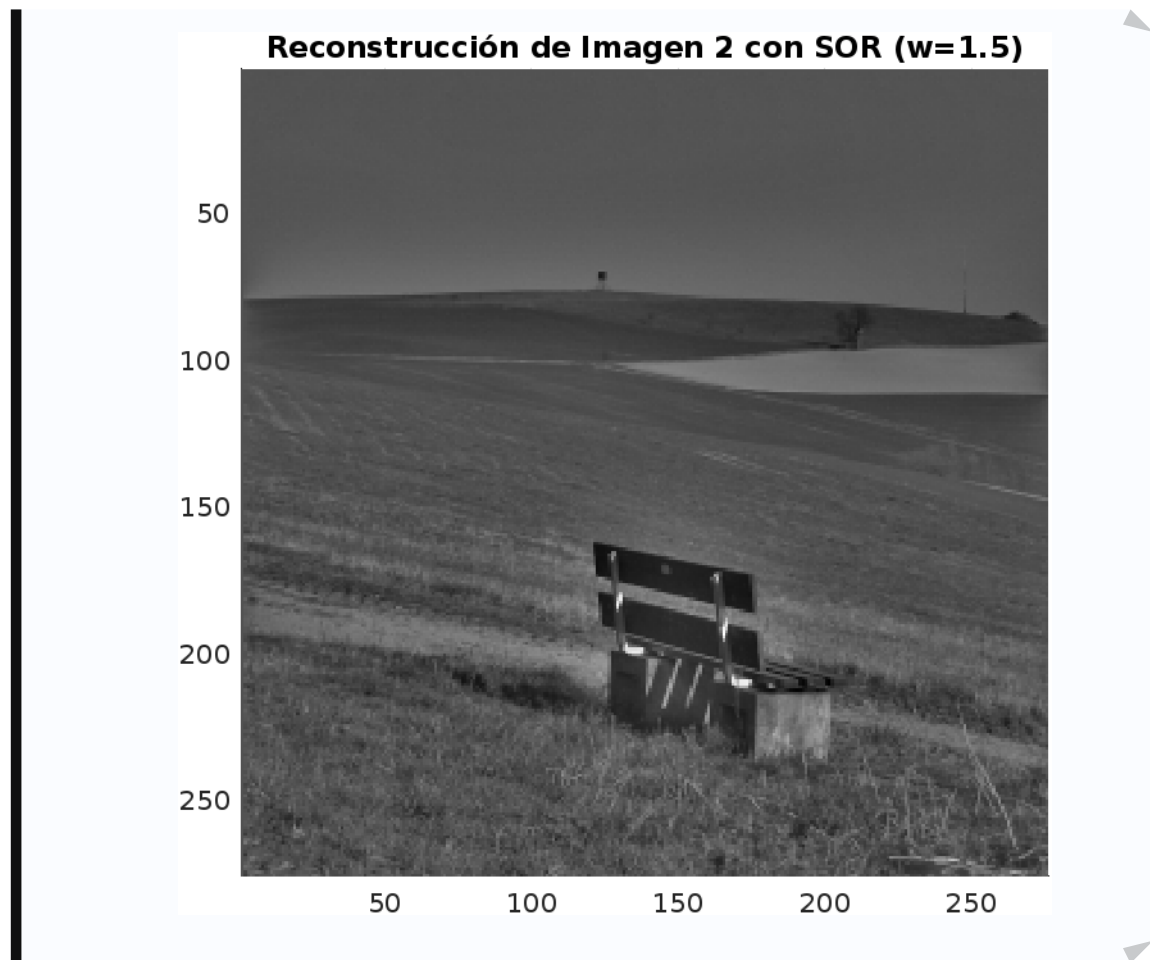
```
1  format rational
2  %% Visualizar la solución
3  function visualizar(u, N, M, titulo)
4      imagen = reshape(u, [N, M]); % Volver a forma de matriz
5      figure;
6      surf(imagen), shading flat, axis ij, axis equal, view(2);
7      colormap gray;
8      title(titulo);
9  end
10
11 %% Cargar datos de los bordes
12 data2 = load('bordes2.mat'); % Cargar archivo
13
14 f = data2.bordes2; % Extraer matriz de bordes
```

```
15
16 [N, M] = size(f); % Dimensiones de la imagen 2
17
18
19 %% Métodos Iterativos
20
21 % Inicialización de la solución
22 u = zeros(M,N); % x0 inicial
23 iteraciones = 0;
24
25 tic;
26 while (100 >= iteraciones)
27     u_anterior = u;
28     for i=2:M-1
29         for j=2:N-1
30             u(i, j) = (1/4) * (-f(i, j) + u(i-1, j) +
31                             u_anterior(i+1, j) + u(i, j-1) + u_anterior(i,
32                             j+1));
31         end
32     end
33     iteraciones = iteraciones + 1;
34 end
35
36 tiempo = toc;
37 disp('Tiempo tomado en el método de Gauss-Seidel');
38 disp(tiempo);
39 visualizar(u, N, M, 'Reconstrucción de Imagen 2 con
    Gauss-Seidel');
```

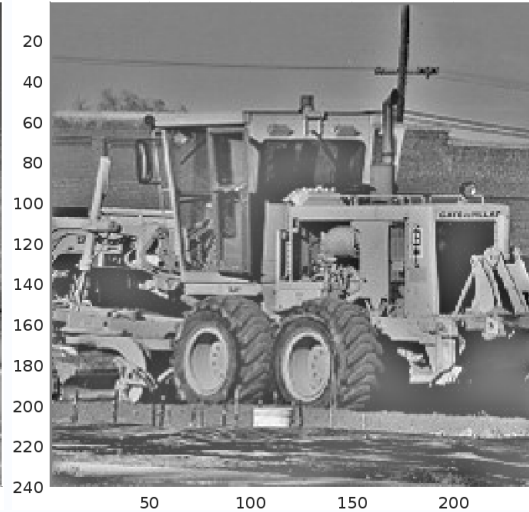
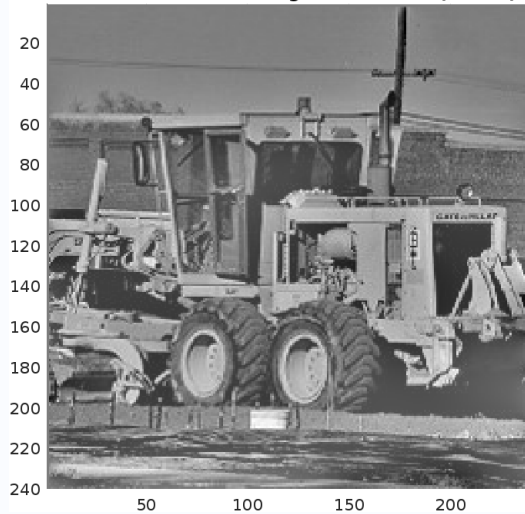
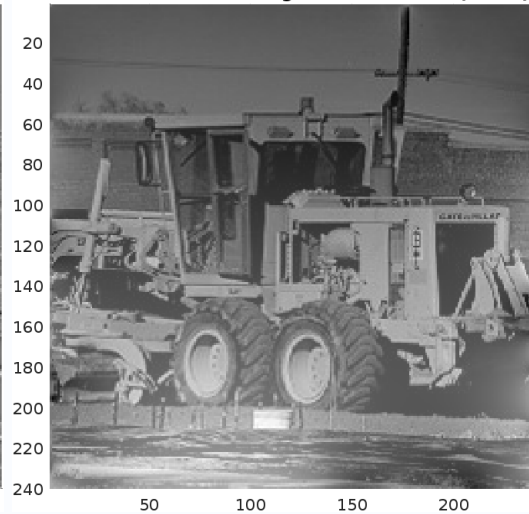


6. Imágen 2 (SOR)  $\frac{163}{2617}s \approx 0,06229s$ .

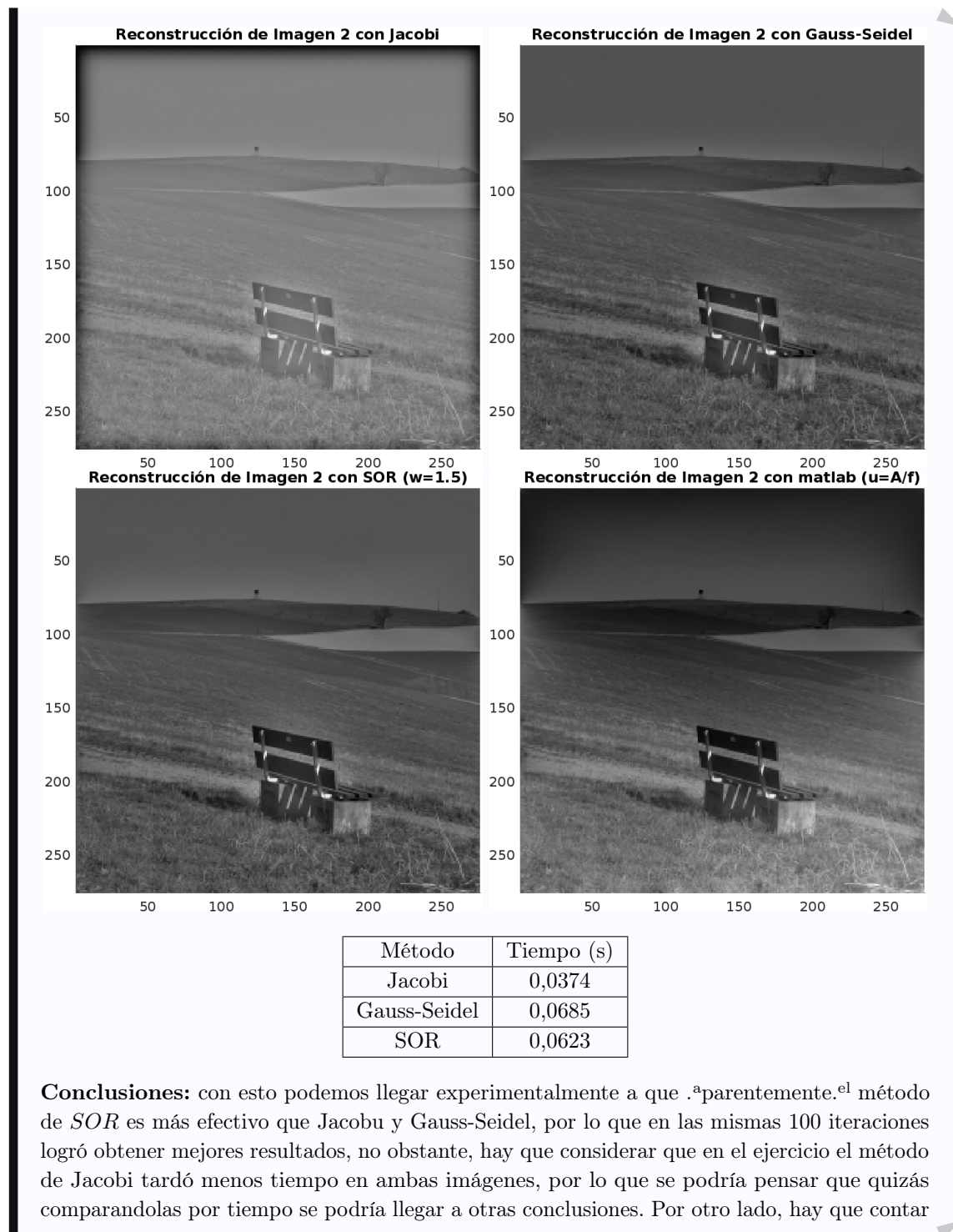
```
1  format rational
2  %% Visualizar la solución
3  function visualizar(u, N, M, titulo)
4      imagen = reshape(u, [N, M]); % Volver a forma de matriz
5      figure;
6      surf(imagen), shading flat, axis ij, axis equal, view(2);
7      colormap gray;
8      title(titulo);
9  end
10
11 %% Cargar datos de los bordes
12 data2 = load('bordes2.mat'); % Cargar archivo
13
14 f = data2.bordes2; % Extraer matriz de bordes
15 w = 1.5;
16 [N, M] = size(f); % Dimensiones de la imagen 2
17
18
19 %% Métodos Iterativos
20
21 % Inicialización de la solución
22 u = zeros(M,N); % x0 inicial
23 ugs = zeros(M,N);
24 iteraciones = 0;
25
26 tic;
27 while (100 >= iteraciones)
28     u_anterior = u;
29     for i=2:M-1
30         for j=2:N-1
31             ugs(i, j) = (1/4) * (-f(i, j) + u(i-1, j) +
32                                 u_anterior(i+1, j) + u(i, j-1) + u_anterior(i,
33                                     j+1));
34             u(i, j) = (1-w)*u(i,j)+w*ugs(i,j);
35         end
36     end
37     iteraciones = iteraciones + 1;
38 end
39 tiempo = toc;
40 disp("Tiempo tomado en el método de SOR con w=1.5");
41 disp(tiempo);
42 visualizar(u, N, M, 'Reconstrucción de Imagen 2 con SOR (w=1.5)');
```





**Comparación:****Reconstrucción de Imagen 1 con Jacobi****Reconstrucción de Imagen 1 con Gauss-Seidel****Reconstrucción de Imagen 1 con SOR ( $w=1.5$ )****Reconstrucción de Imagen 1 con matlab ( $u=A/f$ )**

Método	Tiempo (s)
Jacobi	0,0304
Gauss-Seidel	0,0398
SOR	0,0691



con las modificaciones que se tuvieron que hacer para poder lograr hacer óptima la aplicación de los métodos de Gauss-Seidel y SOR en matlab (situación que comentamos en la nota), por lo que quizás si realizáramos el mismo trato en Jacobi podrían cambiar las situaciones. No obstante, es claro que SOR logró un mejor resultado y que en las primeras 100 iteraciones ese tiempo entre él y los demás es prácticamente despreciable en pro de que *SOR* realizó la tarea de una forma más efectiva.