Facultad de Ingeniería | Universidad de Buenos Aires

95.10 | Modelación numérica

75.12 | Análisis numérico I A
95.13 | Métodos matemáticos y numéricos

Trabajo Práctico 2 – Cuatrimestre 1 2023

Elástica de una viga

Grupo Nº1	**Matias Mitterreiter**	**101789**
	Gaston Matas	**95447**
	Federico Gómez	**109159**

Fecha	Correcciones / Observaciones	Docente

Calificación Final	Docente	Fecha

1 Introducción

En este trabajo se busca calcular la deformación de la viga cuando se aplica una carga y además obtener el diagrama de deflexión en el eje longitudinal.
El modelo matemático de este problema es la ecuación de la curva elástica que es una ecuación diferencial que resuelve el campo de desplazamientos que sufre el eje de la viga desde su forma recta original a la forma deformada final. Para una viga de material elástico lineal sometido a pequeñas deformaciones la ecuación diferencial de la elástica viene dada por esta ecuación:

$$E \cdot I \cdot \frac{d^4 v}{dx^4} = q(x)$$

siendo:

- **E**: módulo de Young,
- **I**: momento de inercia
- **q(x)**: carga aplicada sobre la viga.

La resolución numérica del problema puede realizarse mediante la aplicación de diferencias finitas discretizando el dominio.

2. Metodología

La aplicación del método de diferencias finitas para la resolución de la ecuación diferencial con las condiciones de borde indicadas, implica la resolución de un Sistema de Ecuaciones Lineales con n+1 incógnitas, siendo n la cantidad de nodos. La solución X de este sistema será la deformación en cada nodo de la discretización.

Los métodos utilizados para hallar la solución son:

- **Eliminación Gaussiana:** El método comienza por transformar el sistema de ecuaciones en una matriz ampliada incluyendo así tanto los coeficientes del sistema de ecuaciones como los términos independientes.
  Se aplican operaciones hasta lograr una matriz triangular superior para así aplicar el método de sustitución hacia atrás para obtener las soluciones del sistema:

$$xi = \frac{(bi - \sum_{k=i+1}^{n} x_k * u_{i,k})}{ui,i}, \; i = 1, 2, \ldots, n$$

  Donde $u_{i,j}$ son los coeficientes de la matriz ya triangulada.

  Tanto para triangular la matriz como para aplicar el método de sustitución utilizamos funciones creadas por nosotros en Octave (**Ver Anexo: Códigos**)

- **Jacobi:** El método comienza por reorganizar las ecuaciones del sistema de manera que cada ecuación incluya una sola variable en el lado izquierdo y las restantes variables en el lado derecho. Luego, se descompone la matriz de

coeficientes en una matriz diagonal y dos matrices triangulares: una matriz triangular inferior y una matriz triangular superior.
A continuación, se inicia el proceso iterativo, que consiste en aplicar la siguiente fórmula hasta alcanzar la tolerancia establecida:

$$x_i^{k+1} = \frac{b_i - \sum_{j=1, j\neq i}^{n} a_{i,j} * x_j^k}{a_{i,i}}, \; i = 1, 2, \ldots, n$$

siendo k el número de iteración actual; X^{k+1} el vector x solución en la iteración siguiente; y X^k el vector x solución en la iteración actual.

- **Gauss-Seidel:** El método comienza descomponiendo la matriz de coeficientes en una matriz diagonal y dos matrices triangulares.

  A continuación, se inicia el proceso iterativo, que consiste en aplicar la siguiente fórmula para cada variable en cada iteración hasta conseguir una tolerancia dada:

$$x_i^{k+1} = \frac{(b_i - \sum_{j=0, j\neq i}^{n} a_{i,j} \cdot x_j^{k+1} - \sum_{j=i+1, j\neq i}^{n} a_{i,j} \cdot x_j^k)}{a_{i,i}}, \; i = 1, 2, \ldots, n$$

  siendo k el número de iteración actual; X^{k+1} el vector x solución en la iteración siguiente; y X^k el vector x solución en la iteración actual.

3. Resolución

Para la resolución discretizamos en N = 10 , N = 50 y N = 100. Utilizamos como función de carga la siguiente expresión: $q(x_i) = 2 + 4 * (x_i - x_i^2)$, $L = 1$, $EI = 1$ y una tolerancia relativa de 0,01 para los métodos indirectos.

3.1 Resolución del sistema para cada N

Resolviendo el sistema para los tres valores de N, y utilizando el valor semilla $[0;\ 0,03;\ 0,03;\ \ldots;\ 0,03;\ 0]$ para los métodos indirectos, llegamos a lo siguiente:

Con N = 10, los resultados fueron los mostrados en la figura 1. El costo computacional fue de 0,0028 seg para eliminación de Gauss y de 0,0072 seg para Gauss-Seidel.

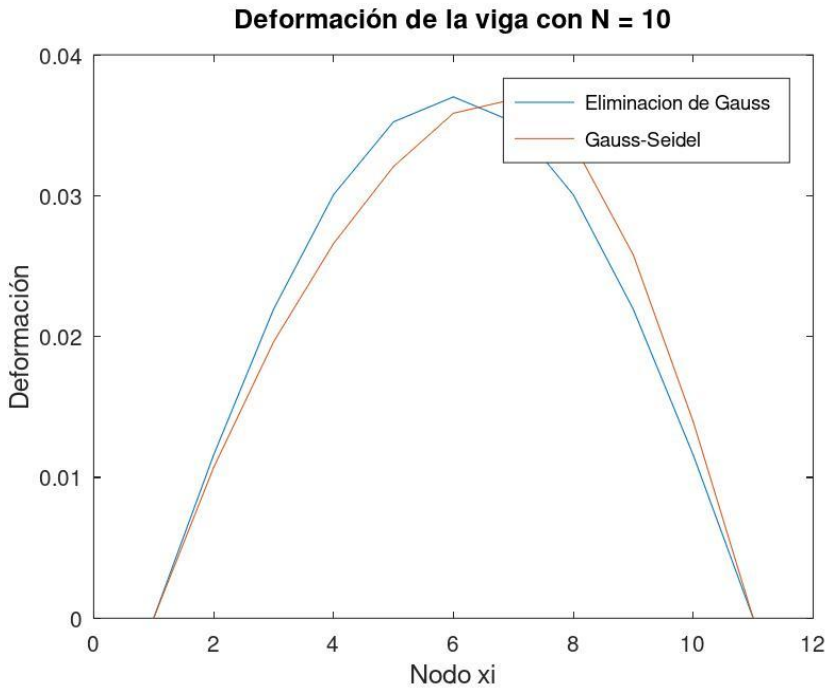

Figura 1. Soluciones de Eliminación de Gauss y Gauss Seidel para N = 10

Con N=50**,** los resultados fueron los mostrados en la figura 2. El costo computacional fue de 0,39 seg para eliminación de Gauss y de 0,31 seg para Gauss-Seidel.

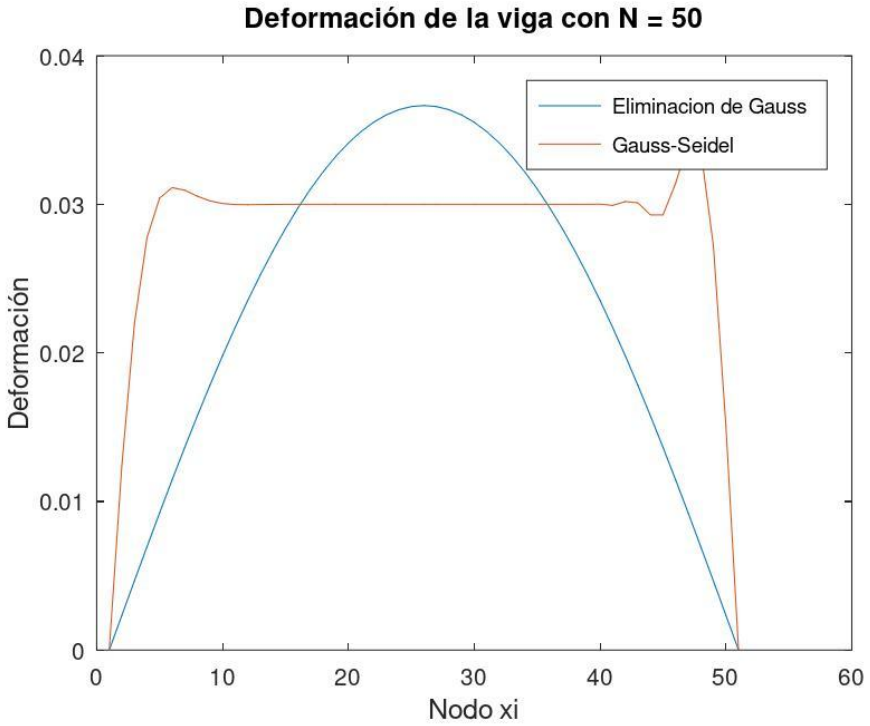

Figura 2. Soluciones de Eliminación de Gauss y Gauss Seidel para N = 50

Con N=100, los resultados fueron los mostrados en la figura 3. El costo computacional fue de 2,76 seg para eliminación de Gauss y de 0,25 seg para Gauss-Seidel.

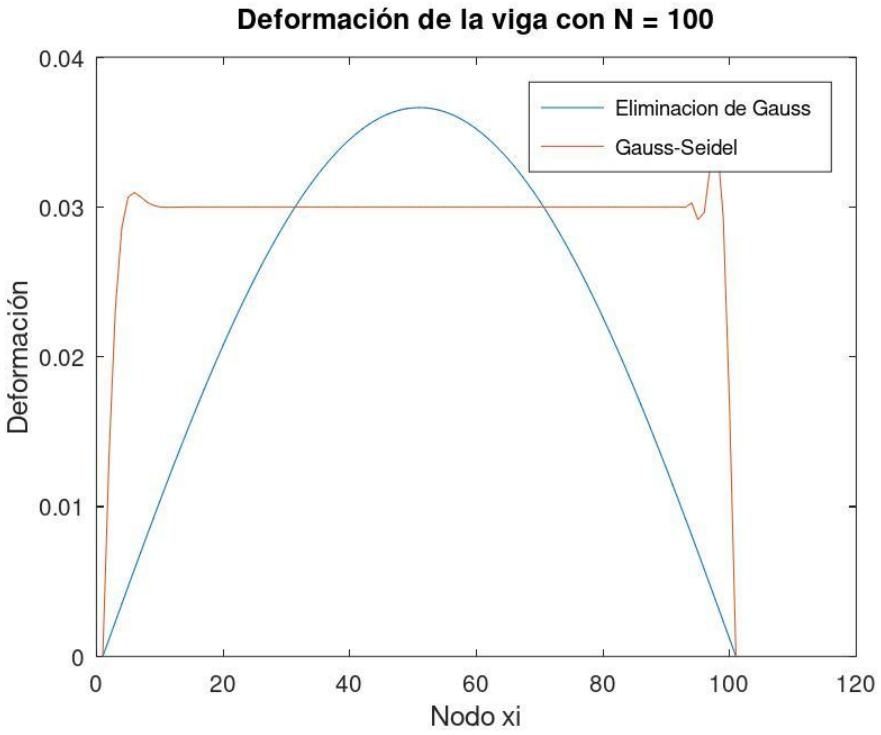

Figura 3. Soluciones de Eliminación de Gauss y Gauss Seidel para N = 100

Al querer resolver los sistemas con el método de Jacobi obtuvimos que no convergen y por ello no se pudo obtener una solución, por lo tanto para corroborar este resultado calculamos el radio espectral de la matriz de iteración, dada por la siguiente expresión:

$$T = -(D^{-1}) * (TI + TS);\ A = D + TI + TS$$

Donde D es una matriz diagonal que tiene los coeficientes de la diagonal de la matriz A; TI es una matriz triangular inferior con los coeficientes por debajo de la diagonal de A; TS es una matriz triangular superior con los coeficientes por encima de la diagonal de A; y A es la matriz del sistema que se quiere resolver.
Al calcularle el radio espectral se obtuvo que su valor es alrededor de 1.6 y al ser mayor a 1 esto indica que el método no converge.

Otra observación importante es que Gauss-Seidel pierde precisión cuanto mayor sea el valor de N para el mismo valor semilla. Pudimos verificar que esto se debe al valor dado de tolerancia, ya que al aumentar la tolerancia a 10^{-2}(figura 4), 10^{-3}(figura 5), y 10^{-4}(figura 6), dejando N fijo en 50, las soluciones estimadas de los métodos indirectos comenzaron a parecerse más a la solución obtenida mediante eliminación gaussiana.

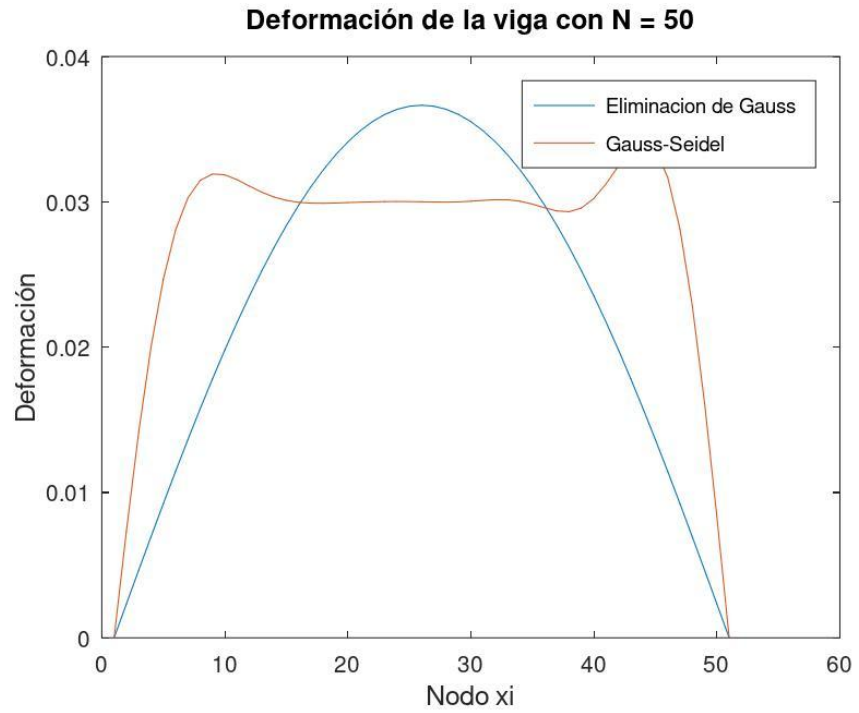

Figura 4. Soluciones estimadas obtenidas para tolerancia 0,001

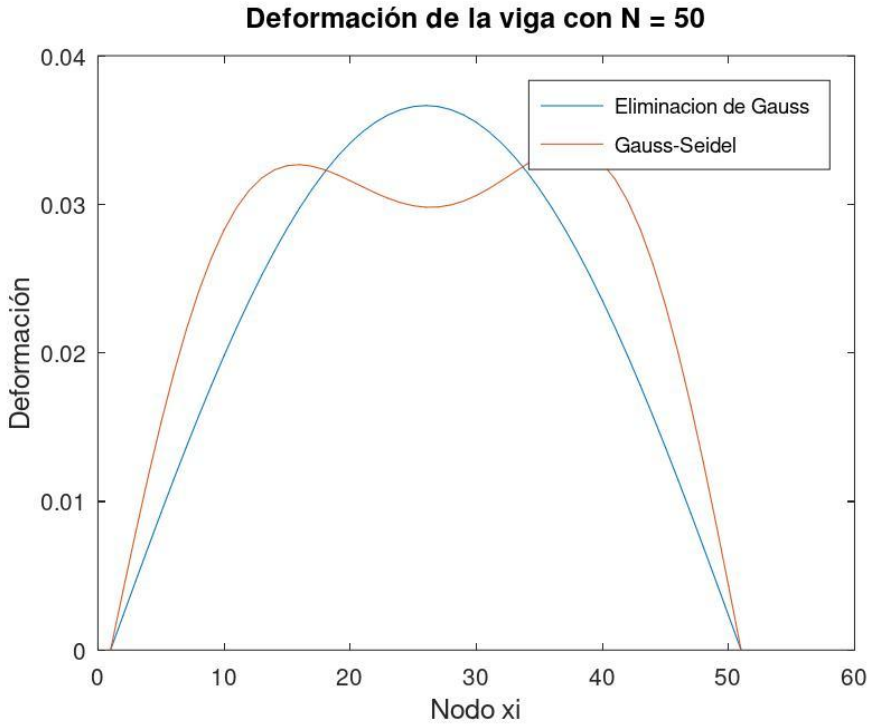

Figura 5. Soluciones estimadas obtenidas para tolerancia 0,0001

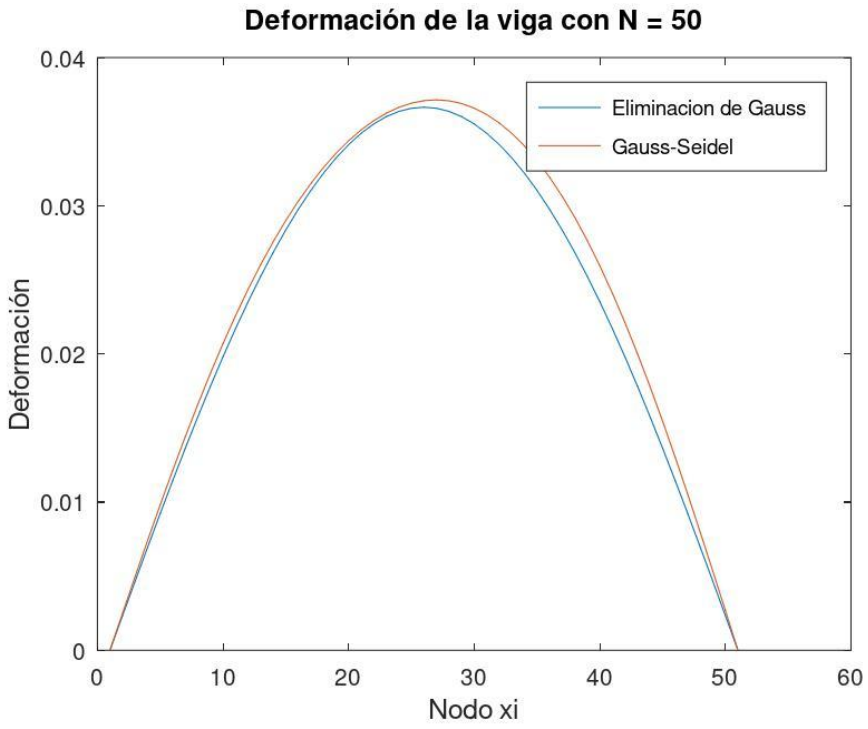

Figura 6. Soluciones estimadas obtenidas para tolerancia 0,00001

Utilizando estas tolerancias observamos los cambios en los tiempos de ejecución del programa, los cuales fueron:

- 3,51 segundos para Gauss-Seidel con tolerancia 0,001
- 38,55 segundos para Gauss-Seidel con tolerancia 0,0001
- 555,63 segundos para Gauss-Seidel con tolerancia 0,00001

Las soluciones de Jacobi no se pueden calcular, y la obtenida por Gauss no cambia. Esto se debe a que Jacobi no converge y por ende no calcula ninguna solución (la ejecución se corta luego de verificar si el radio espectral es mayor a 1) y Gauss no utiliza la tolerancia a la hora de realizar los cálculos.

3.2 Ensayos de Sensibilidad de EI

Para los ensayos de sensibilidad de EI, decidimos dejar N fijo en 100 ya que nos brinda mayor precisión al tener mayor cantidad de nodos así como utilizar una precisión de 0,0001, y modificamos el valor del parámetro EI en 0,1 por encima y por debajo del original (1,0) resultando en 0,9 y 1,1.
Para estos ensayos se obtuvieron los siguientes resultados (figura 7):

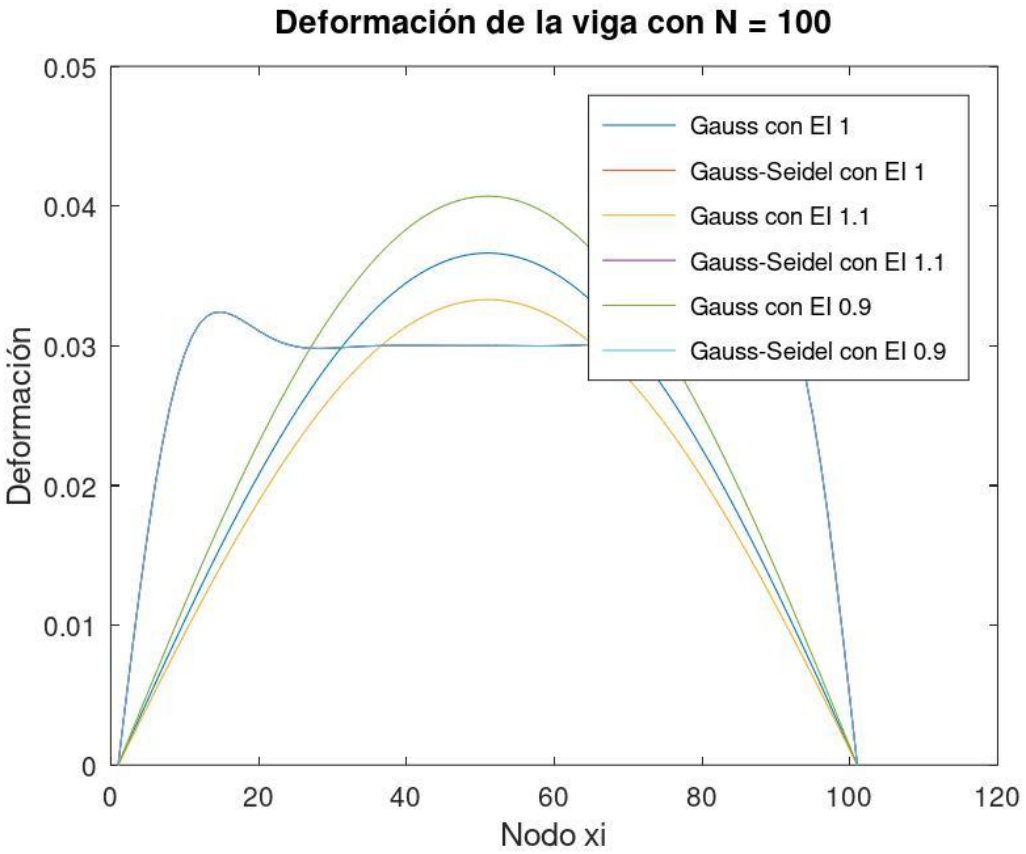

Figura 7. Soluciones estimadas obtenidas para cada valor de EI

Notamos que la viga se deforma más a mayor valor de EI, mientras que si se disminuye este valor, la viga se deforma menos para la misma carga q(x), y que la solución obtenida por Gauss-Seidel.

Algo adicional a esto son los tiempos que se tardaron en ejecutar los ensayos, los cuales al medirlos se obtuvo lo siguiente:

Para un EI de 1,1, se tuvieron los siguientes tiempos de ejecución:
- Gauss: 3,29 segundos
- Gauss-Seidel: 45,96 segundos
- Jacobi (solo calcula radio espectral): 0.0020 segundos

Para un EI de 1, se tuvieron los siguientes tiempos de ejecución:
- Gauss: 3,22 segundos
- Gauss-Seidel: 44,36 segundos
- Jacobi (solo calcula radio espectral): 0.0022 segundos

Para un EI de 0,9, se tuvieron los siguientes tiempos de ejecución:
- Gauss: 3,58 segundos
- Gauss-Seidel: 46,03 segundos
- Jacobi (solo calcula radio espectral): 0.0023 segundos

3.3 Orden de convergencia

Para calcular experimentalmente el orden de convergencia (p) utilizamos los errores absolutos entre las 3 últimas soluciones, es decir, las soluciones de las 3 últimas iteraciones. La fórmula que utilizamos para estimar el orden de convergencia es la siguiente:

$$p = \frac{ln(e^{k+1}) - ln(e^{k})}{ln(e^{k}) - ln(e^{k-1})}$$

Donde e^{k+1} es el error absoluto entre la solución de la iteración k y k+1; e^{k} es el error absoluto entre la solución de la iteración k y k-1; e^{k-1} es el error absoluto entre la solución de la iteración k-1 y k-2;
Todos los errores se calculan como $e^{k+1} = \left|\left|x^{k} - x^{k+1}\right|\right|$. Para estos calculos se utilizó una tolerancia relativa de 0,01, la cual no resulta muy buena, ya que en los valores obtenidos a continuación se puede ver que el orden de convergencia no coincide con el valor real (1) a mayor N.

Los valores obtenidos fueron:

- Para N = 10, el orden de convergencia resultó 1,082
- Para N = 50, el orden de convergencia resultó 0,78
- Para N = 100, el orden de convergencia resultó 0,74

4. Conclusiones

Pudimos comprobar que Gauss-Seidel genera una solución más precisa cuando se tiene un valor menor de tolerancia, pero a la vez se tarda más tiempo en obtener esa solución, ya que se tienen que hacer más iteraciones. Este tiempo que se tarda en llegar a la solución puede mejorarse utilizando el método SOR (Sobre Relajaciones) con un parámetro w entre 1 y 2, ya que sabemos que Gauss-Seidel converge, por lo que el SOR lo hará también.

Por otro lado, Jacobi no converge sin importar el valor de N y esto se debe a que el valor del radio espectral de la matriz de iteración da un valor mayor a 1.

Otra cosa que pudimos comprobar es que el sistema es sensible a cambios en EI, principalmente a un aumento del mismo, pero la solución a la que se llega mediante Gauss-Seidel no varía tanto como la solución por eliminación de Gauss.

Por último, el orden de convergencia del método de Gauss-Seidel, calculado experimentalmente, se aproxima más a 1 cuanto menor es el valor de la tolerancia que se le pide.

ANEXO: Códigos

Aclaración: Se utilizó el código descrito posteriormente variando los parámetros de: “N”, “EI” y “Tolerancia”

Código de cálculo y graficación de soluciones

```
# no puedo arrancar el script con function
clear all;

#Arma la matriz de coeficientes para las ecuaciones de la viga
#segun la cantidad de nodos, n.

function matriz = generar_matriz_viga(n)
  matriz = zeros(n+1);

  for (i = 1:n+1)
    if(i == 1)
      matriz(i,1) = 1;
    elseif(i == 2)
      matriz(i,1) = -4;
      matriz(i,2) = 5;
      matriz(i,3) = -4;
      matriz(i,4) = 1;
    elseif (i > 2 && i < n)
```

```
      matriz(i,i-2) = 1;
      matriz(i,i-1) = -4;
      matriz(i,i) = 6;
      matriz(i,i+1) = -4;
      matriz(i,i+2) = 1;
    elseif (i == n)
      matriz(i,(n+1)-3) = 1;
      matriz(i,(n+1)-2) = -4;
      matriz(i,(n+1)-1) = 5;
      matriz(i,n+1) = -4;
    elseif(i == n+1)
      matriz(i,n+1) = 1;
    endif
  endfor

  matriz = matriz(2:n, 2:n);
endfunction

# Armado del vector b
function b = generar_b(n, L, EI)
  b = zeros(n+1, 1);

  for i=1:n-1
    x = i * (L/n);
    q = 2 + 4 * (x - x^2);
    f = (q/EI) * (L/n)^4;
    b(i+1,1) = f;
  endfor
endfunction

# i fila, j columna, k paso
# mik = aik/akk
# aij(k+1) = aij(k) - mik * akj(k)
# j hasta n
# i hasta n
#  Recibe una matriz extendida con b y la dimension de la matriz sin
extender

function matriz_triangulada = triangular_matriz(matriz_extendida, n)

  for k = 1:n-1
    for i = k+1:n
      m = matriz_extendida(i, k) / matriz_extendida(k, k);

      matriz_extendida(i, k) = 0;

      for j = k+1:n+1
              matriz_extendida(i, j) = matriz_extendida(i, j) - m *
matriz_extendida(k, j);
      endfor

    endfor

  endfor

  matriz_triangulada = matriz_extendida;
endfunction
```

```
# xi = (bi - sumatoria(k=i+1, n) uik * xk) / uii
# A es la matrix extendida con b, A|b
# Recibe la matriz triangulada y extendida con b
function solucion = susticion_inversa(A)
  solucion = zeros(rows(A), 1);

  for i = rows(A):-1:1
    sumatoria = 0;

    for k = i+1:columns(A)-1
      sumatoria = sumatoria + A(i, k) * solucion(k, 1);
    endfor

    solucion(i, 1) = (A(i, columns(A)) - sumatoria) / A(i, i);
  endfor

endfunction

function soulcion = eliminacion_gaussiana(A, b, n)
  matriz_extendida = A;
  matriz_extendida(:, n) = b;

  matriz_triangulada = triangular_matriz(matriz_extendida, n-1);
  soulcion = susticion_inversa(matriz_triangulada);
endfunction

# Los errores se corresponden de la siguiente forma:
# e1 -> ek-1, e2 -> ek, e3 -> ek+1
function orden_convergencia = calcular_orden_convergencia(e1, e2, e3)
  orden_convergencia = (log(e3) - log(e2)) / (log(e2) - log(e1));
endfunction

# La matriz es la matriz reducida, tolerancia 0,01
function x_solucion = jacobi(matriz, tolerancia, x_inicial, b)

  x_actual = x_inicial; #xk
  x_solucion = zeros(rows(matriz), 1); #xk+1
  k = 1;

  #Calculo D (matriz que contiene la diagonal de A)
  D=diag(diag(matriz));
  #Calculo L (matríz diagonal inferior, sin la diagonal)
  L=-tril(matriz, -1); %La opc -1 elimina la diagonal
  #Calculo U (matríz diagonal superior, sin la diagonal)
  U=-triu(matriz, 1);
  #Calculo T (matriz de iteracion de Jacobi)
  T=inv(D)*(L+U);

  radio_espectral = max(abs(eig(T)));

  if(radio_espectral > 1)
    disp('Jacobi Diverge');
    disp(['Radio espectral: ' mat2str(radio_espectral)]);
    return;
  endif

  error_relativo = 0; # error entre soluciones sucesivas
  error_anterior = 0; # error abs de xk-1
```

```
  error_actual = 0; # error abs de xk
  error_siguiente = 0; # error abs de xk+1

  while(error_relativo > tolerancia || k == 1)
    k = k + 1;

    x_actual = x_solucion;

    for i = 1:rows(matriz)
    sumatoria = 0;

    for j = 1:columns(matriz)
      if(j != i)
          sumatoria = sumatoria + matriz(i, j) * x_actual(j, 1); #aij *
xj(k)
      endif
    endfor

    # xi(k+1) = (-sumatoria + bi) / aii
    x_solucion(i, 1) = (b(i, 1) - sumatoria) / matriz(i, i);

    endfor

    error_relativo = norm(x_actual - x_solucion) / norm(x_actual);
    error_anterior = error_actual; # error abs de xk-1
    error_actual = error_siguiente; # error abs de xk
    error_siguiente = norm(x_actual - x_solucion); # error abs de xk+1
  endwhile

endfunction


function x_solucion = gauss_seidel(matriz, tolerancia, x_inicial, b)

  x_actual = x_inicial; #xk
  x_solucion = zeros(rows(matriz), 1); #xk+1
  k = 1;

  error_relativo = 0; # error entre soluciones sucesivas
  error_anterior = 0; # error abs de xk-1
  error_actual = 0; # error abs de xk
  error_siguiente = 0; # error abs de xk+1

  while(error_relativo > tolerancia || k == 1)
    k = k + 1;

    for i = 1:rows(matriz)
      sumatoria = 0;

      for j = 1:i
        if(j != i)
            sumatoria = sumatoria + matriz(i, j) * x_solucion(j, 1); #
aij * xj(k+1)
        endif
      endfor

      for j = i+1:columns(matriz)
        if(j != i)
```

```
            sumatoria = sumatoria + matriz(i, j) * x_actual(j, 1); # aij
* xj(k)
        endif
      endfor

      # xi(k+1) = (-sumatoria + bi) / aii
      x_solucion(i, 1) = (b(i, 1) - sumatoria) / matriz(i, i);
    endfor

    error_relativo = norm(x_actual - x_solucion) / norm(x_actual);
    error_anterior = error_actual; # error abs de xk-1
    error_actual = error_siguiente; # error abs de xk
    error_siguiente = norm(x_actual - x_solucion); # error abs de xk+1
    x_actual = x_solucion;
  endwhile

  disp('-- Método de Gauss-Seidel --');
disp(['Orden                    de                    convergencia:'
mat2str(calcular_orden_convergencia(error_anterior,       error_actual,
error_siguiente))]);
  disp(['Iteraciones: ' mat2str(k)]);

endfunction

# Parametros para la viga
N = 10;
L = 1;
EI = 1;

# Tolerancia de metodos indirectos
tolerancia = 0.01;

matriz_reducida = generar_matriz_viga(N);
b = generar_b(N, L, EI);

solucion_elim_gauss = zeros(N+1, 1);

disp('---- Ejecucion de Eliminacion Gaussiana ----');
tic;
solucion_elim_gauss(2:N,  1)  =  eliminacion_gaussiana(matriz_reducida,
b(2:N, 1), N);
toc;

solucion_gs = zeros(N+1, 1);
solucion_jacobi = zeros(N+1, 1);

for i = 2:N
  solucion_gs(i, 1) = 0.03;
endfor

disp('---- Ejecucion de Jacobi ----');
tic;
solucion_jacobi(2:N,    1)    =    jacobi(matriz_reducida,    tolerancia,
solucion_jacobi(2:N, 1), b(2:N, 1));
toc;

disp('---- Ejecucion de Gauss-Seidel ----');
tic;
```

```
solucion_gs(2:N,   1)   =   gauss_seidel(matriz_reducida,   tolerancia,
solucion_gs(2:N, 1), b(2:N, 1));
toc;

plot(1:N+1, solucion_elim_gauss(1:N+1, 1));
hold on;
plot(1:N+1, solucion_gs(1:N+1, 1));
title('Deformación de la viga con N = 10', 'fontsize', 12);
xlabel('Nodo xi', 'fontsize', 12);
ylabel('Deformación', 'fontsize', 12);
legend('Eliminacion de Gauss', 'Gauss-Seidel');

print -djpeg soluciones_n.jpg;

hold off;

# Ensayos de sensibilidad
tolerancia = 0.0001;

# EI 1.0
EI = 1;
matriz_reducida = generar_matriz_viga(N);
b = generar_b(N, L, EI);

solucion_elim_gauss = zeros(N+1, 1);

disp('---- Elim Gauss EI 1.0 ----');
tic;
solucion_elim_gauss(2:N,  1)  =  eliminacion_gaussiana(matriz_reducida,
b(2:N, 1), N);
toc;

solucion_gs = zeros(N+1, 1);
solucion_jacobi = zeros(N+1, 1);

for i = 2:N
  solucion_gs(i, 1) = 0.03;
endfor

disp('---- Ejecucion de Jacobi 1.0 ----');
tic;
solucion_jacobi(2:N,   1)   =   jacobi(matriz_reducida,   tolerancia,
solucion_jacobi(2:N, 1), b(2:N, 1));
toc;

disp('---- Ejecucion de Gauss-Seidel 1.0 ----');
tic;
solucion_gs(2:N,   1)   =   gauss_seidel(matriz_reducida,   tolerancia,
solucion_gs(2:N, 1), b(2:N, 1));
toc;

plot(1:N+1, solucion_elim_gauss(1:N+1, 1));
hold on;
plot(1:N+1, solucion_gs(1:N+1, 1));
hold on;

#EI 1.1
EI = 1.1;
```

```
matriz_reducida = generar_matriz_viga(N);
b = generar_b(N, L, EI);

solucion_elim_gauss = zeros(N+1, 1);

disp('---- Elim Gauss EI 1.1 ----');
tic;
solucion_elim_gauss(2:N,  1)  =  eliminacion_gaussiana(matriz_reducida,
b(2:N, 1), N);
toc;

solucion_gs = zeros(N+1, 1);
solucion_jacobi = zeros(N+1, 1);

for i = 2:N
  solucion_gs(i, 1) = 0.03;
endfor

disp('---- Ejecucion de Jacobi 1.1 ----');
tic;
solucion_jacobi(2:N,    1)    =    jacobi(matriz_reducida,    tolerancia,
solucion_jacobi(2:N, 1), b(2:N, 1));
toc;

disp('---- Ejecucion de Gauss-Seidel 1.1 ----');
tic;
solucion_gs(2:N,   1)   =   gauss_seidel(matriz_reducida,   tolerancia,
solucion_gs(2:N, 1), b(2:N, 1));
toc;
plot(1:N+1, solucion_elim_gauss(1:N+1, 1));
hold on;
plot(1:N+1, solucion_gs(1:N+1, 1));
hold on;

# EI 0.9
EI = 0.9;
matriz_reducida = generar_matriz_viga(N);
b = generar_b(N, L, EI);

solucion_elim_gauss = zeros(N+1, 1);

disp('---- Elim Gauss EI 0.9 ----');
tic;
solucion_elim_gauss(2:N,  1)  =  eliminacion_gaussiana(matriz_reducida,
b(2:N, 1), N);
toc;

solucion_gs = zeros(N+1, 1);
solucion_jacobi = zeros(N+1, 1);

for i = 2:N
  solucion_gs(i, 1) = 0.03;
endfor

disp('---- Ejecucion de Jacobi 0.9 ----');
tic;
solucion_jacobi(2:N,    1)    =    jacobi(matriz_reducida,    tolerancia,
solucion_jacobi(2:N, 1), b(2:N, 1));
```

```
toc;

disp('---- Ejecucion de Gauss-Seidel 0.9 ----');
tic;
solucion_gs(2:N,   1)   =   gauss_seidel(matriz_reducida,   tolerancia,
solucion_gs(2:N, 1), b(2:N, 1));
toc;
plot(1:N+1, solucion_elim_gauss(1:N+1, 1));
hold on;
plot(1:N+1, solucion_gs(1:N+1, 1));
hold on;
title('Deformación de la viga con N = 100', 'fontsize', 12);
xlabel('Nodo xi', 'fontsize', 12);
ylabel('Deformación', 'fontsize', 12);
legend('Gauss con EI 1', 'Gauss-Seidel con EI 1', 'Gauss con EI 1.1',
'Gauss-Seidel con EI 1.1', 'Gauss con EI 0.9', 'Gauss-Seidel con EI
0.9');

print -djpeg ensayos_sensibilidad_EI.jpg;

hold off;
```