Informe II:
Trabajo Práctico 3

Edgardo Cipolatti
edgardocipolatti@hotmail.com

I. EJERCICIO 7

Resuelva el siguiente sistema de ecuaciones lineales con todos los métodos presentados en esta guía. Utilice una tolerancia en el residuo de 10e-5 para la convergencia. En cada caso, estime el costo del método en función del tiempo reloj de ejecución (comandos tic y toc de Scilab / Octave), el número de iteraciones, tasa de convergencia (realizar gráficas semilogarítmicas norma del residuo vs. número de iteraciones), etc. Compare los resultados con el método directo de Gauss, justificando si es necesario o no utilizar alguna estrategia de pivoteo. Analizar los resultados obtenidos e indicar cual método piensa ud. que es el más conveniente? Cómo justifica que los métodos iterativos logren o no convergencia? Qué expectativa puede tener sobre la precisión de los resultados obtenidos? Las entradas de la matriz de coeficientes del sistema lineal son,

$$a_{ij} = \begin{cases} 2i & si \quad j = i \\ 0.5i & si \quad j = i+2 \text{ ó } j = i-2 \\ 0.25i & si \quad j = i+4 \text{ ó } j = i-4 \\ 0 & en\, otra\, posición \end{cases}$$

Fig. 1

con i = 1; . . . ;N y N = 250; 500; 1000. Las entradas del vector de términos independientes son $b_i = \pi$. Utilice la norma infinito.

Para comenzar de definió una función(script) make_matrix , cuya función es la de crear matrices de tamaño n pasado como parámetro, según la especificación de la figura 1:

A continuación se definió, la función make_T cuyo objetivo es el de encontrar las matrices T respectivas para cada método, el cual es seleccionado en el parámetro con el mismo nombre,(1.Jacobi, 2. Gauss-Seidel, 3.SOR). Y calcular el rho(Radio Espectral) correspondiente con el objetivo final de verificar la convergencia del método previo a su implementación.

Habiendo obtenido los Radios de convergencia respectivos. Se verifica la convergencia de los métodos de Jacobi y de Gauss-Seidel, a partir del Teorema 7.19 [1, p. 444] ”Para cualquier $x^{(0)} \ni R^n$,la sucesión $\{x^{(k)}\}_{k=0}^{\infty}$ definida por $x^{(k)} = Tx^{(k-1)} + c$, para cada $k \geq 1$. Converge a la solución única de x = Tx + c si y solo si $\rho(T) < 1$”.
La convergencia del método SOR se verifica con el Teorema 7.19 [1, p. 444] y del Teorema 7.24 [1, p. 449] ”Si $a_{ii} \neq 0$ para cada i=1,2,...,n entonces $\rho(T_\omega) \geq | \omega - 1 |$. Ello significa que el método SOR puede converger solo si $0 < \omega < 2$”.
Tomando ω = 0.9 se verifica el Teorema 2 .
A continuación se listan los respectivos resultados:

	Jacobi	GS	SOR	GC	Gauss
T[s] 250	0.13	0.039	0.039	0.038	0.446
T[s] 500	0.45	0.107	0.094	0.362	2.92
T[s] 1000	1.48	0.338	0.283	1.884	20.7
ρ 250	0.75	0.37	0.35	-	-
ρ 500	0.75	0.37	0.35	-	-
ρ 1000	0.75	0.37	0.35	-	-
N° It 250	32	8	10	238	-
N° It 500	32	8	10	325	-
N° It 1000	32	8	10	438	-

Para realizar la comparación se utilizo el método de Gauss sin pivoteo, el mismo fue elegido en base al Teorema 6.18 [1, p. 398] ”Se dice que la matriz A de nxn es estrictamente diagonal dominante cuando $| \; a_{ii} \; |> \sum_{j=1 j\neq i}^{n} \; | \; a_{ij} \; |$”. Y al Teorema 6.19 [1, p. 398] Üna matriz A estrictamente diagonal dominante es no singular. Más aún, en este caso podemos realizar la eliminación gaussiana de cualquier sistema lineal de la forma Ax=b para obtener su solución única sin intercambio de renglones ni columnas[...]”. A continuación se listan los algoritmos utilizados para la aplicación de los diferentes métodos, se utilizó error relativo como criterio de convergencia para Jacobi, Gauss-Seidel y SOR, para Gradiente conjugado se uso la norma infinita del error. Los mismos están ubicados en el Apéndice:

- Jacobi
- Gauss Seidel
- SOR
- Gradiente conjugado
- Gauss

Para realizar la comparación por tasa de convergencia, se implemento el método log_plot que recibe por parámetro el histórico de residuos y genera las gráficas semi-logarítmicas, de norma del residuo vs. número de iteraciones) correspondientes, a continuación se muestra las correspondientes gráficas:

Se calculó el número de condición para interpretar la veracidad del término de error. Se utilizo la siguiente función, $\kappa(A) = ||A|| * ||A^{-1}||$, tomando la norma infinita. Los valores calculados son:
Siendo estos valores:

- K(250)=510
- K(500)=1028
- K(1000)=2064

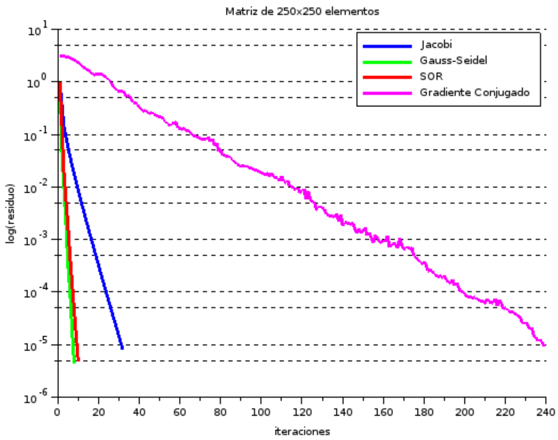

Fig. 2
MATRIZ DE 250X250 ELEMENTOS. TODOS LOS MÉTODOS

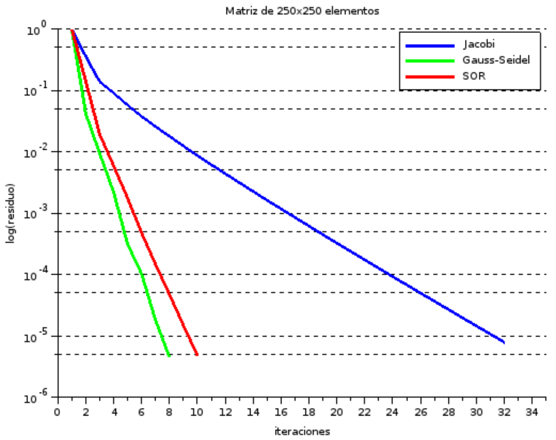

Fig. 3
MATRIZ DE 250X250 ELEMENTOS. JACOBI, GAUSS-SEIDEL Y SOR

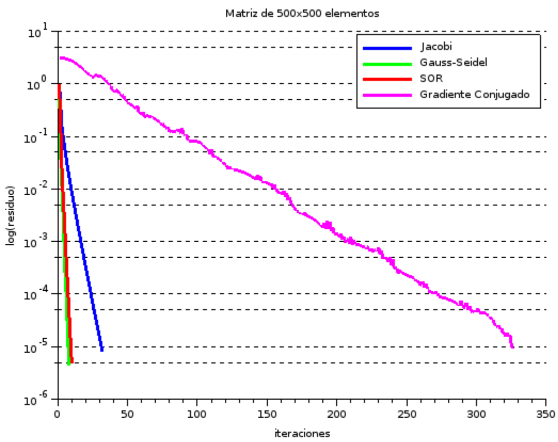

Fig. 4
MATRIZ DE 500X500 ELEMENTOS. TODOS LOS MÉTODOS

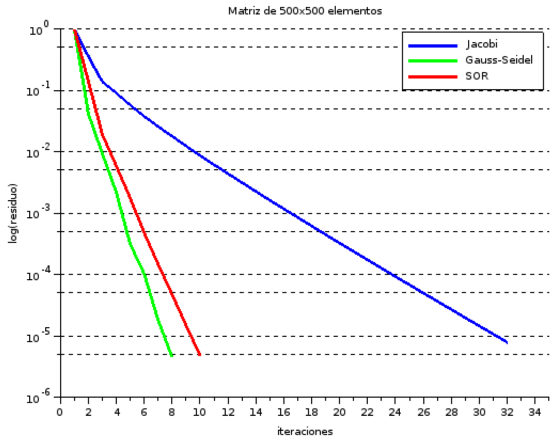

Fig. 5
MATRIZ DE 500X500 ELEMENTOS. JACOBI, GAUSS-SEIDEL Y SOR

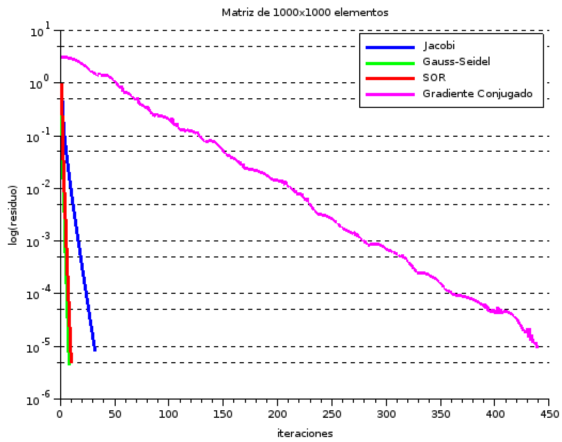

Fig. 6
MATRIZ DE 1000X1000 ELEMENTOS. TODOS LOS MÉTODOS

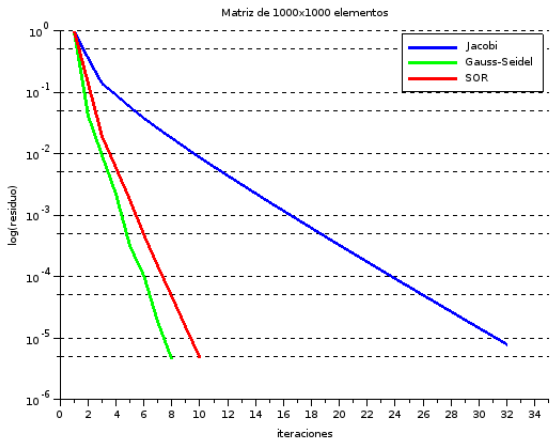

Fig. 7
MATRIZ DE 1000 ELEMENTOS. JACOBI, GAUSS-SEIDEL Y SOR

Conclusión:
Se observa convergencia de todos los métodos. En especial Jacobi, Gauss-Seidel y SOR donde se verifico la convergencia de forma previa a la aplicación de los respectivos procedimientos y cálculos para las 3 matrices.

Los resultados finales indican que la velocidad de convergencia es según el siguiente Orden: Gauss-Seidel, SOR, Jacobi y Gradiente conjugado.
Los métodos no arriban a buenos resultados, siendo aun que el calculo del residuo de los mismo es muy pequeño. Estos es consecuencia de la estrecha relación que existe entre el residuo y el número de condición(Teorema 7.27 [1, p. 455]), con un numero de condición muy alto, el hecho de que los residuos sean pequeños no son una garantía de que el error sea mínimo.
Sujeto a esto para este caso gradiente conjugado es ineficaz. El método utilizado es el del máximo descenso, que calcula el negativo del gradiente que es la dirección del máximo descenso de la función, calculando el mismo a partir del residuo de la forma $r^{(k)} = b - Ax^{(k)}$. Entonces el número de condición muy alto indica que las aproximaciones no son buenas y en consecuencia que los residuos calculados no lo sean tampoco. Dando como resultado que el método deba realizar un número muy grande de iteraciones hasta cumplir con la condición de corte, donde la norma infinita del error debe ser menor que $10 * e^{-5}$.

Por lo tanto, según los resultados obtenidos, el método mas conveniente es el método de Gauss-Seidel, siendo el que arriba al resultado en el menor tiempo, consecuencia de un menor número de Iteraciones.

II. Ejercicio 8

Resuelva los siguientes sistemas lineales con el método de Gauss-Seidel y analice lo que sucede en cada caso. Luego intente resolverlos mediante el método directo de eliminación de Gauss. Es necesario aplicar alguna estrategia de pivoteo? Si lo fuera, justifique por qué.

$$\left.\begin{aligned} 3x + y + z &= 5 \\ x + 3y - z &= 3 \\ 3x + y - 5z &= -1 \end{aligned}\right\} \qquad (1)$$

$$\left.\begin{aligned} 3x + y + z &= 5 \\ 3x + y - 5z &= -1 \\ x + 3y - z &= 3 \end{aligned}\right\} \qquad (2)$$

Utilizando el algoritmo de Gauss Seidel para sistema de ecuaciones (1) obtengo el siguiente resultado en 11 iteraciones, un tiempo de 0,002 segundos y con una tolerancia de error de 1e-5.

$$\begin{bmatrix} 0{,}9999991 \\ 1{,}0000007 \\ 0{,}9999996 \end{bmatrix}$$

El algoritmo de Gauss-Seidel para el sistema de ecuaciones (2) corta por límite de iteraciones (100) y arroja el siguiente resultado:

$$10^{125} * \begin{bmatrix} 2{,}081159 \\ -29{,}51648 \\ -86{,}46828 \end{bmatrix}$$

sabemos que este resultado es erróneo en base a la comparación con el resultado del método directo de eliminación de Gauss. A demás el radio espectral de la matriz es mayor a 1 entonces el método diverge.

Aplicando el método directo de eliminación por Gauss para la obtención de la solución exacta del sistema de ecuaciones (1), nos encontramos con el resultado:

$$x = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}^t$$

El mismo fue obtenido sin estrategia de pivoteo ya que la matriz es estrictamente diagonal dominante Teorema 6.19 [1, p. 398]. Esto se puede comprobar utilizando el siguiente algoritmo es_EDD .

Para la resolución del sistema (2) se necesita usar Gauss con pivoteo ya que la matriz no es estrictamente diagonal dominante. Por inspección visual en el segundo reglón tenemos que el valor absoluto del elemento de la diagonal es 1 y la suma de los valores absolutos de los demás elementos es 8, por ende no es E.D.D. Teorema 6.18 [1, p. 398] . Además podemos observar que no es simétrica definida positiva. Somos capaces de verificarlo de dos maneras; la primera es que vemos un elemento negativo en la diagonal Teorema 6.21 [1, p. 401] y la segunda manera es verificarlo con el siguiente algoritmo es_SDP .

Utilizando el algoritmo antes mencionado se obtiene el resultado exacto del sistema el cual es:

$$x = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}^t$$

Como conclusión podemos rescatar que el método de Gauss-Seidel obtiene buenos resultado solo si el número de condición es bajo y el radio espectral de la matriz es menor a uno (ya que diverge en caso contrario) Teorema 7.19 [1, p. 444] . Aún así conviene usar el método de Gauss (con estrategia de pivoteo de ser necesario) para matrices chicas, ya que obtenemos la solución exacta del sistema.

Referencias

[1] R. L. Burden, *Análisis Numérico*, 7th ed. Thomson Learning, 2002.
[2] V. Sonzogni, "Cálculo numérico - apuntes de cátedra," 2014.

ANEXO

A. APÉNDICE

A continuación se muestran los códigos utilizados para el desarrollo de este trabajo práctico:

```
function [A,b,x0] = make_matrix(n)
   if n < 5 then
      disp("Dimensión de la matriz inadecuada (ingresar n > 4)");//Limitación agregada por
          la condicion de j=i-4
      return;
   end

   A = zeros(n,n);
   b = %pi * ones(n,1);
   x0 = zeros(n,1);

   for i = 1 : n
      A(i,i) = 2 * i;
      for j = 1 : n
         if ((j == i-2) | (j == i+2)) then
            A(i,j) = 0.5 * i;
         end
         if ((j == i-4) | (j == i+4)) then
            A(i,j) = 0.25 * i;
         end
      end
   end

endfunction
```

Algoritmo 1: Make Matrix

```
function [x,nit,rh] = jacobi(A,b,x0,maxit,tol)
  nit = 0;
  [m,n] = size(A);
  r = 1.0;
  rh = [];
  x = x0;

  while (r >= tol & nit < maxit)
     xold = x;

     for i = 1 : n
        x(i) = (b(i) - A(i,1:i-1)*xold(1:i-1) - A(i,i+1:n)*xold(i+1:n)) / A(i,i);
     end

     r = norm(x-xold, "inf") / norm(x, "inf");
     rh = [rh,r];
     nit = nit + 1;

     if r <= tol then
        disp("Proceso finalizado por tolerancia de error");
        return;
     end

     if nit >= maxit then
        disp("Proceso finalizado por límite de iteraciones");
        return;
     end
  end
endfunction
```

Algoritmo 2: Método de Jacobi

```
function [x,nit,rh] = gauss_seidel(A,b,x0,maxit,tol)
  nit = 0;
  [m,n] = size(A);
  r = 1.0;
  rh = [];
  x = x0;

  while (r > tol & nit < maxit)
     xold = x;

     for i = 1 : n
        // La segunda sumatoria corresponde x de la iteración anterior,
        // pero dado xold = x, es indistinto
        x(i) = (b(i) - A(i,1:i-1)*x(1:i-1) - A(i,i+1:n)*x(i+1:n)) / A(i,i);
     end

     r = norm(x-xold, "inf") / norm(x, "inf");
     rh = [rh,r];
     nit = nit + 1;

     if r <= tol then
        disp("Proceso finalizado por tolerancia de error");
        return;
     end

     if nit >= maxit then
        disp("Proceso finalizado por límite de iteraciones");
        return;
     end
  end
endfunction
```

Algoritmo 3: Método de Gauss Seidel

```
function [x,nit,rh] = sor(A,b,x0,maxit,tol,w)
  nit = 0;
  [m,n] = size(A);
  r = 1.0;
  rh = [];
  x = x0;
  while (r > tol & nit < maxit)
     xold = x;

     for i = 1 : n
        // La segunda sumatoria corresponde x de la iteración anterior,
        // pero dado xold = x, es indistinto
        xgs(i) = (b(i) - A(i,1:i-1)*x(1:i-1) - A(i,i+1:n)*x(i+1:n)) / A(i,i);
        x(i) = (1-w) * xold(i) + w * xgs(i);
     end

     r = norm(x-xold, "inf") / norm(x, "inf");
     rh = [rh,r];
     nit = nit + 1;

     if r <= tol then
        disp("Proceso finalizado por tolerancia de error");
        return;
     end

     if nit >= maxit then
        disp("Proceso finalizado por límite de iteraciones");
        return;
     end
  end
endfunction
```

Algoritmo 4: Método de Sobre Relajación

```
function[x_k,k,rh] = gc(A,b,x0,maxiter,tol)
  k = 0;
  x_k = x0;
  r_k = b - A * x_k;
  // Criterio de convergencia: norma infinito de r_k
  ninf = max(abs(r_k)); // o lo que es lo mismo ninf = norm(r_k, "inf");
  rh = ninf;
  v_kp1 = r_k; // 1era. dirección de búsqueda (descenso más rápido)

  while (k <= maxiter & ninf > tol)
     dot1 = sum(r_k.^2);
     vaux = A * v_kp1; // kp1 --> k plus one
     dot2 = v_kp1' * vaux;
     t_kp1 = dot1 / dot2;
     x_kp1 = x_k + t_kp1 * v_kp1;
     // comienza el cálculo de la próxima dirección de búsqueda por
     // gradientes conjugados
     r_kp1 = r_k - t_kp1 * vaux;
     s_kp1 = sum(r_kp1.^2) / dot1; // escalar que permite obtener la nueva dir. de búsqueda
     // sale de hacer el planteo que las direcciones de búsqueda son A-ortogonales
         (A-conjugadas)
     v_kp1 = r_kp1 + s_kp1 * v_kp1; // nueva dirección de búsqueda
     // fin del cálculo de la próx. dir. de búsqueda
     r_k = r_kp1;
     ninf = max(abs(r_k));
     rh = [rh, ninf];
     x_k = x_kp1;
     k = k + 1;
  end // end while
endfunction
```

Algoritmo 5: Método Gradiente Conjugado

```
function [A] = gauss(A,b)
   [m,n] = size(A);

   for k = 1 : n-1 // desde k = 1 hasta n-1 con incremento de 1
      // calcula los multiplicadores
      A(k+1:n,k) = A(k+1:n,k) / A(k,k);

      // eliminación
      for j = k+1 : n // desde j = k+1 hasta n con incremento de 1
         A(k+1:n,j) = A(k+1:n,j) - A(k+1:n,k) * A(k,j);
      end

     //b(k+1:n) = b(k+1:n) - A(k+1:n,k)*b(k);
   end

   //x = backsub(A,b);
endfunction

function [x] = backsub(U,b)
   [m,n] = size(U);
   tol = 1e-9;
   x = zeros(n,1); // vector columna de nx1 ceros

   if(abs(U(n,n)) < tol) // si Unn es muy chico
      disp("No se puede continuar con el algoritmo");
      return;
   end

   x(n) = b(n) / U(n,n);

   for i = n-1: -1 : 1 // para i = n-1 hasta 1 decrementando de a 1
      x(i) = (b(i) - sum(U(i,i+1:n) * x(i+1:n))) / U(i,i);
   end
endfunction
```

Algoritmo 6: Método de Gauss

```
function [resp]=es_EDD(A)
   [m,n]=size(A);

   for i=1:m
      if(abs(A(i,i))<(sum(abs(A(i,j=1:n)))-abs(A(i,i)))
      disp ("NO ES EDD");
      resp=0;//0 = No es EDD.
      return;
      end
   end
   disp ("SI ES EDD");
   resp=1;
endfunction
```

Algoritmo 7: Verificación de EDD

```
function [T,D,L,U,rho] = make_t(A,method,w)
   [m,n] = size(A);
   T = zeros(n,n);
   D = zeros(n,n);
   L = zeros(n,n);
   U = zeros(n,n);

   for i = 1 : n
      D(i,i) = A(i,i);
      for j = i + 1 : n
         U(i,j) = -A(i,j);
         L(j,i) = -A(j,i);
      end
   end

   select method
      case 1 then T = inv(D) * (L + U),       // jacobi
      case 2 then T = inv(D - L) * U,         // gauss-seidel
      case 3 then T = inv(D - w*L) * ((1-w) * D + w*U), // SOR
      else T = inv(D) * (L + U),
   end
   rho = norm(T);

endfunction
```

Algoritmo 8: Make T

```
function [] = log_plot3(rh1,rh2,rh3,tit)
   [m1,n1] = size(rh1);
   iter1 = [1:n1];
   [m2,n2] = size(rh2);
   iter2 = [1:n2];
   [m3,n3] = size(rh3);
   iter3 = [1:n3];

   scf(5);
   clf(5);
   plot2d("nl",iter1,rh1,style=2); // "nl" --> normal x, logarítmica y; style: color de la
       linea
   p1 = get("hdl");
   p1.children.thickness = 3; // grosor
   plot2d("nl",iter2,rh2,style=3);
   p2 = get("hdl");
   p2.children.thickness = 3;
   plot2d("nl",iter3,rh3,style=5);
   p3 = get("hdl");
   p3.children.thickness = 3;
   legend(["Jacobi";"Gauss-Seidel";"SOR"]);
//   plot2d("nl",iter,rh3,style=2);
//   plot2d("nl",iter,rh4,style=2);
//   p.children.mark_mode = "off";
//   p.children.mark_style = 9; // tipo de punto
//   p.children.mark_foreground = 2; // color del punto


   xtitle(tit, "iteraciones", "log(residuo)"); // titulo, eje x, eje y
   set(gca(), "grid", [-1,1]);
endfunction
endfunction
```

Algoritmo 9: log_plot

```
function [resp]=es_SDP (A)
  [m,n]=size(A);

    if(A '= A')//Verifico que la matriz sea simétrica
      disp("No es simétrica definida positiva");
      resp=0;
      return;
    end
    for i = 1 : n
        if(det(A(1:i)) <=0;
          disp("No es simétrica definida positiva");
          resp=0;
          end
    end;
    disp("Es simétrica definida positiva");
    resp=1;
endfunction
```

Algoritmo 10: es_SDP