Trabajo práctico Nº3: Métodos Iterativos

Darién Julián Ramírez
Universidad Nacional del Litoral. ianshalaga@gmail.com

A continuación se muestran las resoluciones de los ejercicios 7 y 8 de la guía de trabajos prácticos número 3.

I. EJERCICIO 7.

Se procede a resolver el sistema de ecuaciones lineales dado por la siguiente matriz de coeficientes y vector de términos independientes,

$$a_{ij}=\begin{cases}2i \;\; si \;\; j=i \\ 0{,}5i \;\; si \;\; j=i+2 \;\; ó \;\; j=i-2 \\ 0{,}25i \;\; si \;\; j=i+4 \;\; ó \;\; j=i-4 \\ 0 \;\; en otro caso\end{cases} ;i=1,\ldots,N \;\; ; N=250,500,100 \;\; ; b_i=\pi$$

Con los métodos numéricos de solución de sistemas de ecuaciones lineales: eliminación gaussiana con sustitución hacia atrás sin pivoteo, Jacobi, Gauss-Seidel, SOR (con diferentes valores de omega) y Gradiente Conjugado.

La matriz de coeficientes y el vector de términos independientes son creados con el algoritmo *matriz* para los distintos valores de *n* expresados anteriormente. A continuación se muestra una tabla con los tiempos de creación del sistema:

	N = 250	N = 500	N = 1000
Tiempo (t)	0,031002	0,074005	0,14201

Se utiliza el algoritmo *edd* con las tres matrices y se verifica que todas ellas son estrictamente diagonal dominantes, entonces por el *Teorema 6.19* de la fuente bibliográfica se asegura que se puede realizar eliminación gaussiana del sistema presentado para obtener su solución única sin intercambio de renglones ni columnas y que los cálculos son estables respecto al crecimiento de los errores de redondeo. Por esta razón se opta por realizar eliminación gaussiana sin pivoteo.

Se estimará el costo de cada método en función del tiempo de reloj de ejecución y el número de iteraciones realizados. Para los métodos iterativos se utilizará una tolerancia en el residuo de *10e-5*, norma infinito, el parámetro *maxit* será fijado en valores altos para forzar el corte por tolerancia y la aproximación inicial *x0* será un vector columna de ceros cuya dimensión coincida con el *N* de la matriz correspondiente. Para el caso del método *SOR* se utilizarán tres valores de *omega* distintos. El óptimo, calculado mediante el algoritmo *woptimo*, otro valor menor a este y otro mayor.

Matriz N = 250	Tiempo (t)	Número de iteraciones (it)
Eliminación gaussina con sustitución hacia atrás sin pivoteo	0,29302	-
Jacobi	1,2462	37
Gauss-Seidel	0,26402	7
SOR (ω = 0,5) Subrelajación	0,80605	21
SOR (ω = 1,2034) Óptimo	0,48103	12
SOR (ω = 1,5) Sobrerrelajación	0,99706	26
Gradiente conjugado	0,066004	197

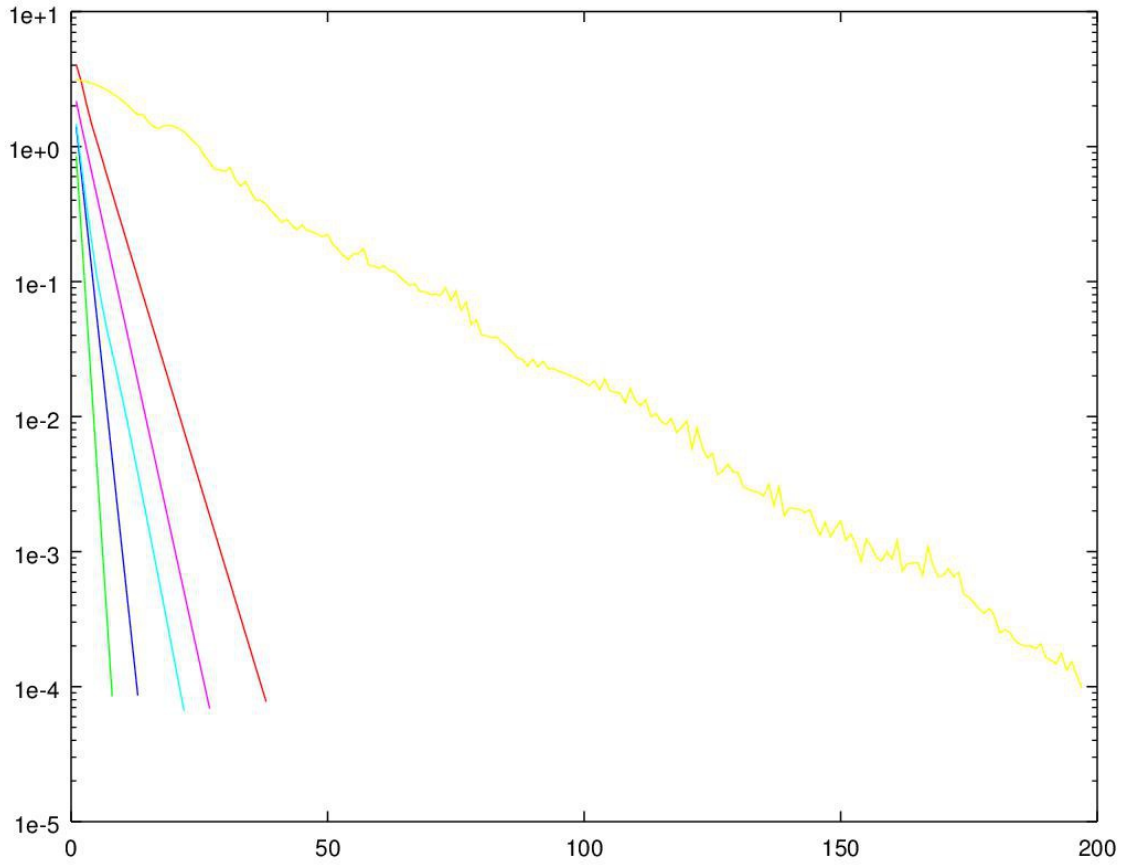

Fig. 1: N=250. Tasa de convergencia (Norma del residuo vs. Número de iteraciones).
Rojo – Jacobi.
Verde – Gauss-Seidel.
Azul – SOR (omega óptimo).
Celeste – SOR (omega = 0,5).
Magenta – SOR (omega = 1,5).
Amarillo – Gradiente conjugado.

Matriz N = 500	Tiempo (t)	Número de iteraciones (it)
Eliminación gaussina con sustitución hacia atrás sin pivoteo	2,5291	-
Jacobi	2,6041	37
Gauss-Seidel	0,54003	7
SOR (ω = 0,5) Subrelajación	1,6541	21
SOR (ω = 1,2037) Óptimo	0,97306	12
SOR (ω = 1,5) Sobrerrelajación	2,0721	26
Gradiente conjugado	0,19501	268

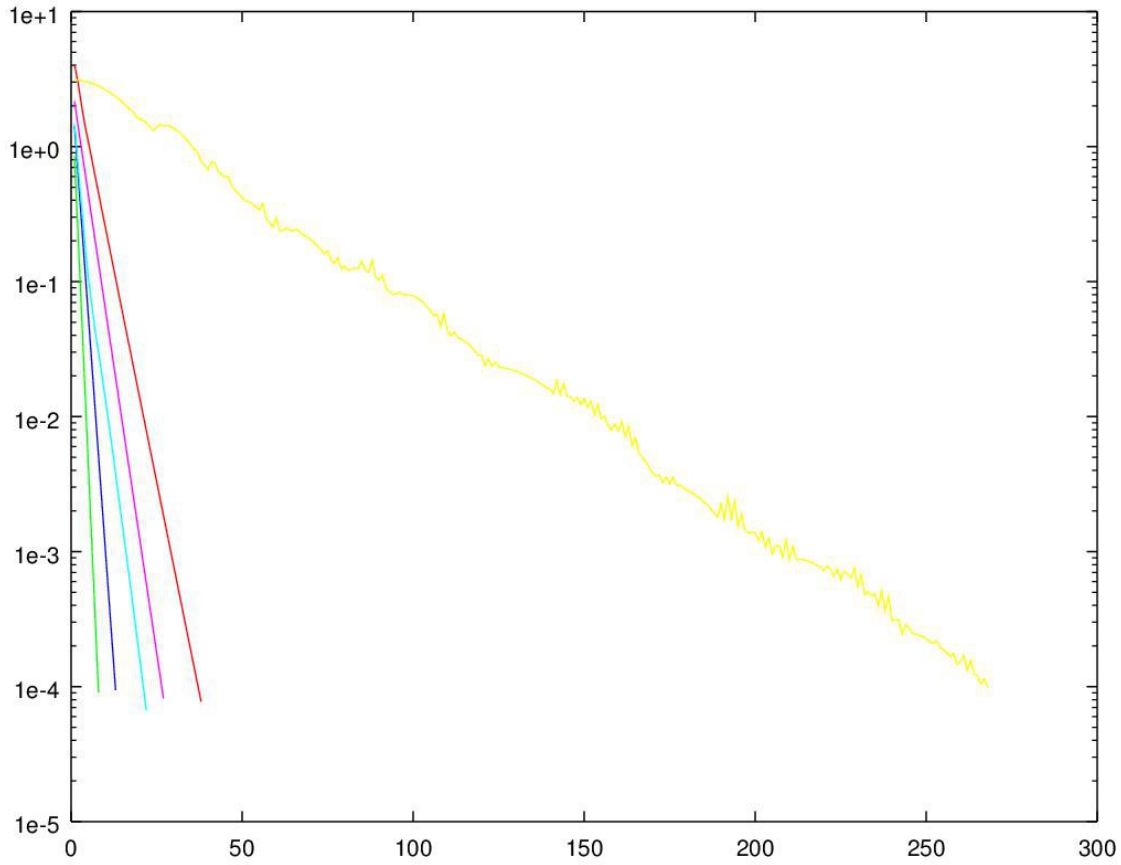

Fig. 2: N=500. Tasa de convergencia (Norma del residuo vs. Número de iteraciones).
Respeta el código de colores de la Fig. 1.

Matriz N = 1000	Tiempo (t)	Número de iteraciones (it)
Eliminación gaussina con sustitución hacia atrás sin pivoteo	16,424	-
Jacobi	5,7683	37
Gauss-Seidel	1,2211	7
SOR (ω = 0,5) Subrelajación	3,9082	21
SOR (ω = 1,2038) Óptimo	2,3011	12
SOR (ω = 1,5) Sobrerrelajación	4,5723	26
Gradiente conjugado	0,97906	360

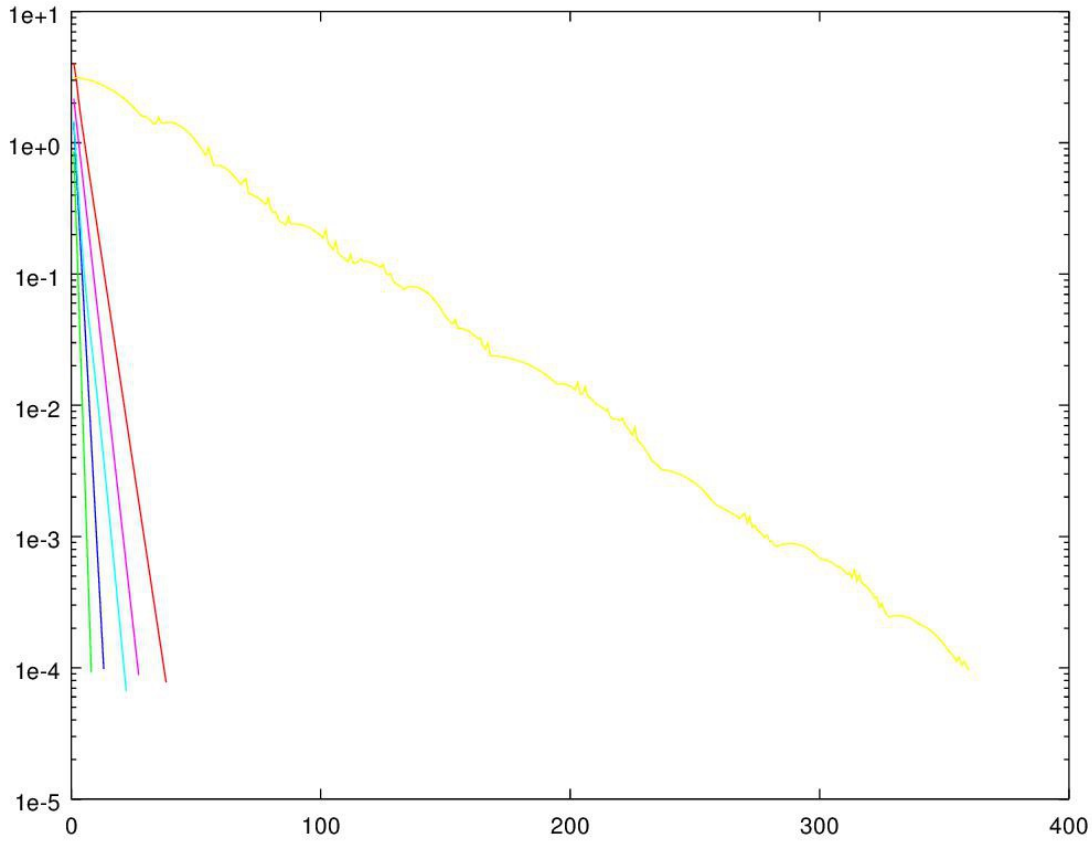

Fig. 3: N=1000. Tasa de convergencia (Norma del residuo vs. Número de iteraciones).
Respeta el código de colores de la Fig. 1.

Tomando al tiempo como criterio principal para determinar cual de los métodos es más conveniente se puede decir que para los tres casos gobierna el método del *gradiente conjugado*. Sin embargo, no puede asegurarse la convergencia de este método y esto está ligado al condicionamiento de la matriz. Los números de condición de las tres matrices son mucho mayores a uno dando un problema mal condicionado que provoca que este método se vuelva altamente susceptible a los errores de redondeo. Por esto, es válido considerar como conveniente al método más próximo en tiempo sabiendo que tiene la convergencia asegurada. Con el algoritmo *matrizt* se determinan las matrices de iteración de Jacobi, Gauss-Seidel y SOR. A calcular sus radios espectrales se muestra que estos son menores a uno para los tres valores de *N*, entonces por el *Teorema 7,19* de la fuente bibliográfica se asegura que se converge a la solución única de la sucesión a la que responden estos algoritmos.

II. EJERCICIO 8.

Se tienen los siguientes sistemas lineales:

$$A_1=\begin{pmatrix}3 & 1 & 1\\ 1 & 3 & -1\\ 3 & 1 & -5\end{pmatrix};\ b_1=\begin{pmatrix}5\\ 3\\ -1\end{pmatrix};\ A_2=\begin{pmatrix}3 & 1 & 1\\ 3 & 1 & -5\\ 1 & 3 & -1\end{pmatrix};\ b_2=\begin{pmatrix}5\\ -1\\ 3\end{pmatrix}$$

Al resolverlos con el método iterativo de Gauss-Seidel se converge a la solución en un total de 8 iteraciones para el primer caso, no obstante, en el segundo caso se corta por tolerancia tras realizar 243 iteraciones y no se logra converger a la solución. Utilizando el algoritmo *matrizt* se obtienen las matrices de iteración de Gauss-Seidel de ambos sistemas, al calcular sus radios espectrales se ve que para el primer sistema dicho radio es 0,25820 < 1 y por lo tanto converge pero por otro lado, el radio espectral para la matriz del segundo sistema es 18,577 > 1 y por ende no es convergente (*Teorema 7,19*).

Utilizando el algoritmo *eed* se obtiene que la primer matriz es estrictamente diagonal dominante mientras que la segunda no lo es, por lo tanto, el primer sistema podrá resolverse sin efectuar pivoteo mientras que en el segundo será necesario realizarlo para disminuir los errores de redondeo (*Teorema 6,19*).

III. ALGORITMOS.

```
function [b,t] = edd(A) #Verifica que una matriz sea estrictamente diagonal dominante
  tic();
  n = length(A);
  for i=1:n
    if (abs(A(i,i)) <= (sum(abs(A(i,:))) - abs(A(i,i))))
      b = false;
      return;
    endif
  endfor
  b = true;
  t = toc();
endfunction
```

```
function [M,b,t] = matriz(n) #Crea la matriz de coeficientes
  tic();
  M = zeros(n,n);
  for i=1:n
    M(i,i) = 2*i;
    if (i+2 <= n)
      M(i,i+2) = 0.5*i;
    endif
    if (i-2 >= 1)
      M(i,i-2) = 0.5*i;
    endif
    if (i+4 <= n)
      M(i,i+4) = 0.25*i;
    endif
    if (i-4 >= 1)
      M(i,i-4) = 0.25*i;
    endif
    b(i) = pi;
  endfor
  b = b';
  t = toc();
endfunction
```

```
function [Tj,Tgs,Tsor,t] = matrizt(A,w)
  tic();
  D = diag(diag(A));
  L = tril(A,-1);
  U = triu(A,1);
  Tj = inv(D)*(L+U);
  Tgs = -inv(D+L)*U;
  Tsor = inv(D-w*L)*((1-w)*D+w*U);
  t = toc();
endfunction
```

```
function [w,t] = woptimo(A)
  tic();
  [Tj] = matrizt(A,1);
  w = 2/(1+sqrt(1-(max(eig(Tj)))^2));
  t = toc();
endfunction
```

```
function [x,t] = eliminacion_gauss(A,b)
  tic();
  #b debe ser un vector columna
  n = length(A); #n = length(b);
  for d=1:n #Pivot, recorre la diagonal
```

```
    m = A(d+1:n,d)/A(d,d);
    A(d+1:n,d:n) = A(d+1:n,d:n) - m*A(d,d:n);
    b(d+1:n) = b(d+1:n) - m*b(d);
  endfor
  x = sust_back(A,b,n);
  t = toc();
endfunction
```

```
function [x]= sust_back(A,b,n)
  x(n) = b(n)/A(n,n);
  for i=n-1:-1:1
    x(i) = (b(i) - sum(A(i,i+1:n).*x(i+1:n)))/A(i,i);
  endfor
  x = x';
endfunction
```

```
function [x,r,it,t] = jacobi(A,b,x0,maxit,tol)
  tic()
  n = length(A);
  x = x0; #Debe inicializarse x
  it = 0;
  while(it < maxit)
    for i=1:n
      x(i) = (b(i) - A(i,1:i-1)*x0(1:i-1) - A(i,i+1:n)*x0(i+1:n))/A(i,i);
    endfor
    r(it+1) = norm(A*x-b,inf);
    if (r(it+1) < tol)
      break
    endif
    x0 = x;
    it = it+1;
  endwhile
  t = toc();
endfunction
```

```
function [x,r,it,t] = gs(A,b,x0,maxit,tol)
  tic()
  n = length(A);
  x = x0; #Debe inicializarse x
  it = 0;
  while(it < maxit)
    for i=1:n
      x(i) = (b(i) - A(i,1:i-1)*x(1:i-1) - A(i,i+1:n)*x0(i+1:n))/A(i,i);
    endfor
    r(it+1) = norm(A*x-b,inf);
    if (r(it+1) < tol)
      break
    endif
    x0 = x;
    it = it+1;
  endwhile
  t = toc();
endfunction
```

```
function [x,r,it,t] = sor(A,b,x0,maxit,tol,w)
  tic()
  n = length(A);
  x = x0; #Debe inicializarse x
  it = 0;
  while(it < maxit)
    for i=1:n
      x(i) = (1-w)*x0(i) + w*(b(i) - A(i,1:i-1)*x(1:i-1) - A(i,i+1:n)*x0(i+1:n))/A(i,i);
```

```
    endfor
    r(it+1) = norm(A*x-b,inf);
    if (r(it+1) < tol)
      break
    endif
    x0 = x;
    it = it+1;
  endwhile
  t = toc();
endfunction
```

```
function [x,rh,it,t] = gc(A,b,x,maxit,tol)
  tic();
  r = b-A*x;
  v = r;
  c = r'*r;
  for it=1:maxit
    if (norm(v,inf)<tol)
      break;
    endif
    z = A*v;
    t = c/(v'*z);
    x = x + t*v;
    r = r - t*z;
    d = r'*r;
    rh(it) = norm(r,inf);
    if (rh(it)<tol)
      break;
    endif
    v = r + d/c*v;
    c = d;
  endfor
  t = toc();
endfunction
```

REFERENCIAS

[1] Richard L. Burden & J.Douglas Faires, *Análisis Numérico*.