Trabajo Práctico Entregable N°1

Métodos directos e iterativos para la resolución de sistemas de ecuaciones lineales

Facultad: Facultad de Ingeniería y Ciencias Hídricas.

Carrera: Ingeniería Informática

Materia: Cálculo Numérico

Alumno: Garcia, Lucas Enedín

Introducción

En este informe se dará a conocer las diferentes maneras de resolver un sistema de ecuaciones algebraicas lineales (SEAL) aplicando diferentes métodos de resolución y haciendo una comparación entre estos. Para ello, deberemos resolver un problema de valores de contorno (PVC), cuyo enunciado se dará luego de una breve explicación de los métodos a utilizar.

Métodos directos y métodos iterativos

- *Método de Eliminación de Gauss:*

  Es un método en el que realiza una serie de transformaciones sobre el sistema de ecuaciones, utilizando operaciones elementales, de modo de obtener un sistema equivalente con matriz triangular.

  En un sistema de n ecuaciones se efectúan n-1 pasos de eliminación. Tanto la matriz A del sistema como el vector de términos independientes b, van siendo transformados en matrices $A^{(k)}$ y vectores $b^{(k)}$ en cada paso k de la eliminación. Para cada paso, la k-esima fila queda inalterada, igual que las filas superiores a ella, solo se modifican las interiores. El elemento a_{kk} se denomina pivote. La fila k se denomina fila pivote y la columna k columna pivote.

  Para este caso, generamos una matriz triangular superior, equivalente a la matriz ampliada, y luego por medio de la sustitución hacia atrás o retro-sustitución obtenemos los valores de las incógnitas.

- *Método Iterativo de Jacobi:*

  En este método, la matriz de coeficientes A se descompone de la forma:

  $$A = L+D+U$$

  Donde:

  - D es matriz diagonal con la diagonal de A ($d_{ii} = a_{ii}$);

- L matriz triangular inferior con los términos de A excluyendo la diagonal ($l_{ij} = a_{ij}\ para\ i > j$);
- U matriz triangular superior con los términos de A excluyendo la diagonal ($u_{ij} = a_{ij}\ para\ i < j$).

Luego reemplazamos esta descomposición en el sistema de ecuaciones

$$(L + D + U)x = b$$

Tomando a M = D y N = -(L+U):

$$(M - N)x = b$$

$$Mx = Nx + b$$

$$x = M^{-1}Nx + M^{-1}b$$

Tomando $T = M^{-1}N \quad y \quad C = M^{-1}b$ obtenemos

$$x^k = Tx^{k-1} + c$$

En la iteración k, la componente i de x se calcula:

$$x_i^{(k)} = \left[b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k-1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k-1)} \right] / a_{ii}$$

- *Método Iterativo de Gauss-Seidel:*

En este método, A se descompone de la misma manera que el método anterior con la diferencia de que ahora se toma a

M = D-L y N = -U

Procediendo de manera similar al método de Jacobi se llega a

$$x^k = Tx^{k-1} + c$$

En la iteración k, la componente i de x se calcula:

$$x_i^{(k)} = \left[b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k-1)} \right] / a_{ii}$$

- *Método Iterativo SOR (Sobre-Relajaciones Sucesivas):*

  Este método utiliza la misma descomposición de A que en los métodos anteriores con la diferencia de que se toma a M y a N de la siguiente manera:

$$M = D - wL$$

$$N = (1 - w)D - wU$$

  Donde w es el parámetro de relajación.

  Luego se procede de forma similar a los métodos anteriores.

  Se tiene que en la iteración k, la componente i de x se puede calcular como:

$$x_i^{(k)} = \omega \left[b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k-1)} \right] / a_{ii} + (1-\omega) x_i^{(k-1)}$$

  Observar que si w=1, se obtienen la misma expresión que el método de Gauss-Seidel.

Enunciado

Ejercicio Entregable: Dado el siguiente sistema de ecuaciones lineales:

$$\begin{cases} x_1 = 0, \\ -x_{i-1} + 2x_i - x_{i+1} = 1/N^2, \qquad i = 2, 3, \ldots, N-1, \\ x_N = 0. \end{cases}$$

(a) Realice un script que resuelva el sistema para $N = 100$, utilizando los métodos de Jacobi, Gauss-Seidel, SOR, gradiente conjugado y eliminación de Gauss.

(b) Determine el número de iteraciones necesarias para cada método iterativo, considerando una cota para el **residuo** de $1e-6$. Determine, para el método de SOR, un parámetro de relajación ω óptimo. ¿Todos los métodos convergen? Justifique y grafique el historial del residuo para cada método.

(c) Suponiendo que la solución obtenida corresponde a una función $y = x(t)$ evaluada en N puntos uniformemente distribuidos en el intervalo $[0, 1]$, graficar la solución $y = x(t)$ obtenida con cada método, y saque conclusiones.

Sistema de ecuaciones a resolver

$$Ax = b$$

Donde:

- *A es la matriz de coeficientes de* 100x100.
- *x es el vector columna de* 100x1 (*son las incognitas*).
- *b es el vector columna de* 100x1 (*son constantes*).

$$\begin{bmatrix} 1 & 0 & 0 & & 0 & 0 & 0 \\ -1 & 2 & -1 & \cdots & 0 & 0 & 0 \\ 0 & -1 & 2 & & 0 & 0 & 0 \\ & \vdots & & \ddots & & \vdots & \\ 0 & 0 & 0 & \cdots & 2 & -1 & 0 \\ 0 & 0 & 0 & \ldots & -1 & 2 & -1 \\ 0 & 0 & 0 & \ldots & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{98} \\ x_{99} \\ x_{100} \end{bmatrix} = \begin{bmatrix} 0 \\ 1/100^2 \\ 1/100^2 \\ \vdots \\ 1/100^2 \\ 1/100^2 \\ 0 \end{bmatrix}$$

Script

```
clear all;
clc;
clf;

#Problema de Valor de Contorno

#{      x1 = 0
#{-x(i-1)+2x(i)-x(i+1) = 1/N^2,   i=2,3,...,N-1.
#{      xN = 0

#Condiciones de contorno:
x1 = 0;
xN = 0;

#Número de incognitas
N = 100;

#------------Armado de la matriz-------------------------
#Tomamos el tiempo que lleva el armado de la matriz
tic();
unos = ones(N,1);
MatrizDiagonal = [-1*unos 2*unos -1*unos]; %valores de las diagonales
A = spdiags(MatrizDiagonal, [-1 0 1], N, N); %matriz de NxN con valores solo en las diagonales -1 0 y 1

#Modificamos la matriz por las condiciones de contorno
A(1,[1:2]) = [1 0];      %solo x1
A(N, [N-1:N]) = [0 1];   %solo xN
#Armado del lado derecho
b = [x1;unos(2:N-1).*(1/(N^2));xN];
#Finalizo el tiempo para ver cuanto nos llevó armar la matriz
tArmadoMatriz = toc()
#-------------------------------------------------------------

#Resolucion del sistema por el resolutor de Octave:
tic();
X = A \ b;
tOctave = toc();
#-------------------------------------------------------------
#----------------------Métodos Iterativos----------------------------
#Inicializacion de parámetros:
n = length(A(:,1))
maxit = 15000
tol = 1e-6
Xini = zeros(n,1)

#Solución con Jacobi
[TJ,cJ] = Tc_Jacobi(A,b);
KTJ = cond(TJ);
rhoTJ = max(abs(eig(TJ)));
tic()
[XJ, itJ, rhJ] = Jacobi(A,b,Xini,maxit,tol);
tJ = toc();
#Solución con Gauss-Seidel
[TGS,cGS] = Tc_GaussSeidel(A,b);
KTGS = cond(TGS);
rhoTGS = max(abs(eig(TGS)));
tic()
[XGS, itGS, rhGS] = GaussSeidel(A,b,Xini,maxit,tol);
tGS = toc();

#Solución con SOR
wSOR = 2/(1+sqrt(1-(rhoTJ^2)))
tic()
[XSOR, itSOR, rhSOR] = Sor(A,b,Xini,maxit,tol,wSOR);
tSOR = toc();

#Metodo Directo Eliminacion Gaussiana:
tic()
[xEG,rEG] = Elimin_Gauss_Pivot(A,b);
tEG = toc();
#Solucion con Gradientes Conjugados
#Armado de la matriz simétrica
b1 = b(2:n-1);
length(b1)
A1 = spdiags(MatrizDiagonal, [-1 0 1], N-2, N-2)
XiniGC = zeros((N-2),1);

tic()
[XGC, itGC, rhGC] = GradienteConjugado(A1,b1,XiniGC,maxit,tol);
XGC = [x1;XGC;xN];
tGC = toc();
```

Resultados y comparaciones

En la Figura 1 se observa la gráfica de la solución al PVC en el intervalo [0; 1] dibujada con los diferentes métodos que se utilizaron. Se puede apreciar que todos los métodos llegan a una misma solución aproximada. Los pequeños errores en las soluciones, visualmente, no se llegan a apreciar.

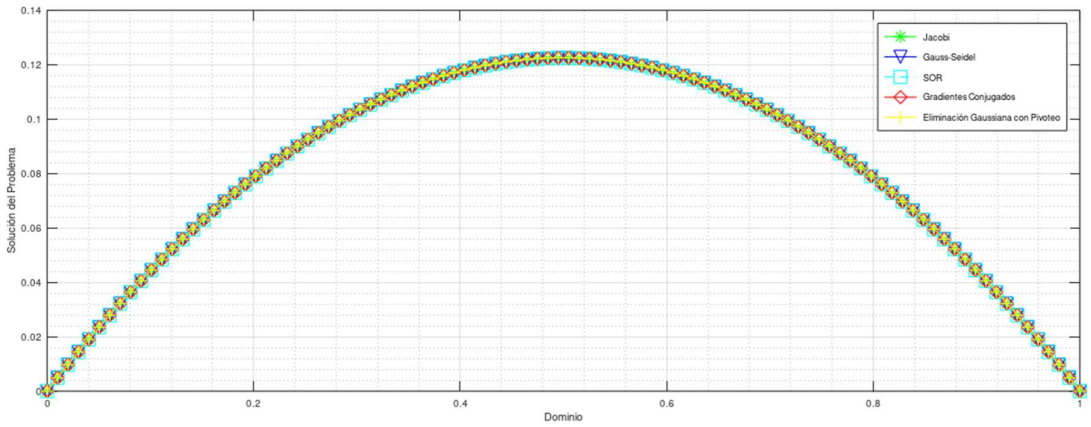

Figura 1: Gráfica de y = x(t) en el intervalo [0; 1]

En la Figura 2 vemos que los 4 métodos iterativos convergen.

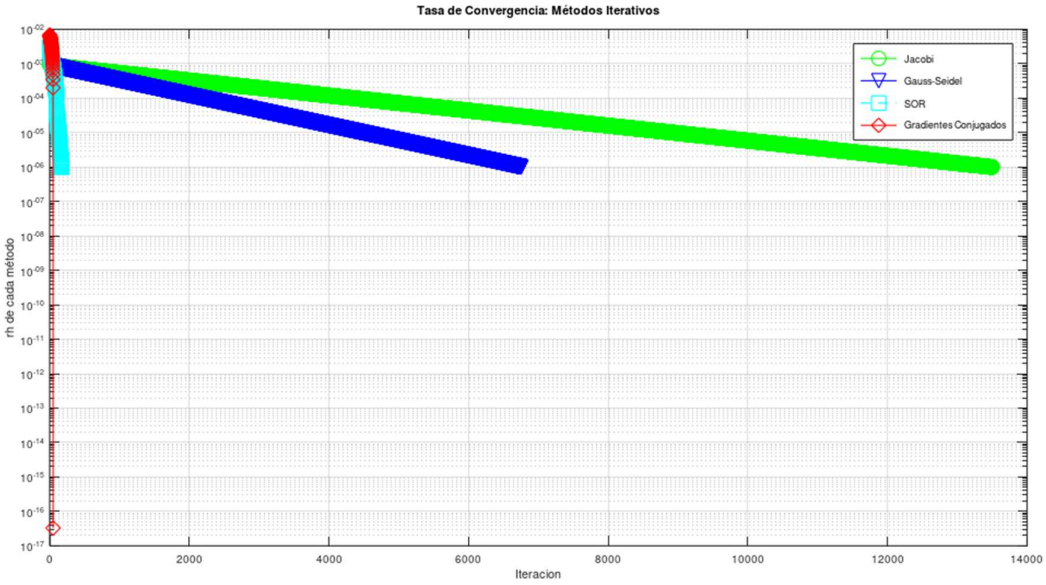

Figura 2: Convergencia de los métodos iterativos

Comparación de entre los diferentes métodos aplicados:

Teniendo en cuenta el tamaño de la matriz de coeficientes (100x100) se obtuvieron los siguientes valores:

	Gauss	Jacobi	Gauss-Seidel	SOR	Grad. Conjugado
Tiempo[seg]	0.019048	45.569	22.846	0.6530	5.5699e-03
Iteraciones	–	13498	6750	172	49
ρ(T)	–	0.9995	0.9990	0.9385	–

- ρ(T) es el Radio Espectral de la matriz de iteración T (correspondiente a los métodos Jacobi, Gauss-Seidel y SOR).

Algoritmos utilizados para la resolución de este problema

- **Método de Eliminación Gaussiana con Pivoteo:**

```
function [x,r] = Elimin_Gauss_Pivot (A, b)
  n=length(b);
  A=[A b];
  r=1:n;
  epsilon=1e-9;
  for k=1:n-1
    % la funcion max devuelve
    % pmax: el pivote de mayor valor absoluto
    % p: posicion donde se encuentra pmax (local --> desde k hasta n. No incluye los de arriba que ya usó)
    [pmax,p] = max(abs(A(r(k:n),k)));
    if pmax<epsilon
      disp('Los posibles pivots son CERO')
      break
    endif
    p = p+k-1; %actualizamos pos. a numeracion global
    if p~=k
      r([p k])= r([k p]); %actualiza el pivote
    endif
    A(r(k+1:n),k) = A(r(k+1:n),k)/A(r(k),k);  %Los 'm'
    A(r(k+1:n),k+1:n+1) = A(r(k+1:n),k+1:n+1)- A(r(k+1:n),k)*A(r(k),k+1:n+1);
  endfor

  x=Sustitucion_Atras(A(r,:));
endfunction
```

Algoritmo auxiliar para realizar la retro-sustitución o sustitución hacia atrás:

```
function [x] = Sustitucion_Atras (A)

  n = length(A(:,1));
  x = zeros(n,1);

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

endfunction
```

- **Métodos iterativos:**

  Para los siguientes métodos se utilizaron las siguientes variables:

  - maxit: es máximo número de iteraciones que le damos al algoritmo. En caso de que las iteraciones lleguen a este valor el algoritmo finaliza. Para el Problema del enunciado maxit = 15000.
  - Tol: es una tolerancia aplicada para finalizar el algoritmo en caso de que el error llegue a ser menor a este valor. Para el Problema del enunciado tol = 1e-6.

- Método de Jacobi:

```
function [x,it,r,t] = Jacobi(A,b,x0,maxit,tol)
  #Inicializamos las variables:
  n = length(A);
  x = x0;
  it = 0;

  #Ejecutar las siguientes intrucciones teniendo en cuenta un valor
  # máximo de iteraciones (maxit):
  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

    #Calculamos el residuo y lo guardamos en el vector de residuos(r)
    r(it+1) = norm(A*x - b); %norm(x-x0)

    #Comparamos el valor del residuo con la tolerancia dada;
    if r(it+1) < tol
      break;
    endif
    #Actualizamos el x0 y el numero de iteracion (it)
    x0 = x;
    it += 1;
  endwhile
endfunction
```

- Método iterativo de Gauss-Seidel:

```
function [x,it,r] = GaussSeidel(A,b,x0,maxit,tol)

  #Inicializamos las variables:
  n = length(A);
  x = x0;
  it = 0;

  #Ejecutar las siguientes intrucciones teniendo en cuenta un valor
  # máximo de iteraciones (maxit):
  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

    #Calculamos el residuo y lo guardamos en el vector de residuos(r)
    r(it+1) = norm(A*x - b); %norm(x-x0)

    #Comparamos el valor del residuo con la tolerancia dada;
    if r(it+1) < tol
      break;
    endif
    #Actualizamos el x0 y el numero de iteracion (it)
    x0 = x;
    it += 1;
  endwhile
endfunction
```

- Método iterativo de SOR:

Para este método además de los parámetros anteriores, se agrega un parámetro de relajación ‘w’ el cual, para que el valor sea óptimo en el método, lo calculamos a partir de la fórmula

$$w = \frac{2}{1 + \sqrt{1 - [\rho(T_J)]^2}}$$

Donde $\rho(T_J)$ es el radio espectral de la matriz de iteración T del método de Jacobi.

```
function [x,it,r] = Sor(A,b,x0,maxit,tol,w)
  #Inicializamos las variables:
  n = length(A);
  x = x0;
  it = 0;

  #Ejecutar las siguientes intrucciones teniendo en cuenta un valor
  # máximo de iteraciones (maxit):
  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

    #Calculamos el residuo y lo guardamos en el vector de residuos(r)
    r(it+1) = norm(A*x - b); %norm(x-x0)

    #Comparamos el valor del residuo con la tolerancia dada;
    if r(it+1) < tol
      break;
    endif
    #Actualizamos el x0 y el numero de iteracion (it)
    x0 = x;
    it += 1;
  endwhile
endfunction
```

- Método iterativo de Gradientes Conjugados:

```
function [x,it,rh] = GradienteConjugado(A,b,x,maxit,tol)
  r = b - A*x;
  v = r;
  c = r'*r;
  for it=1:maxit
    if norm(v) < 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,2);
    if rh(it)<tol
      break;
    endif
    v = r + d/c*v;
    c = d;
  endfor
endfunction
```

Conclusión

Luego del análisis de los diferentes métodos puede decirse que, en cuanto a tiempo de ejecución, cantidad de iteraciones y rapidez de convergencia, el método de Gradientes Conjugados es el más eficiente ya que la cantidad de iteraciones que realiza es menor o a lo sumo igual al tamaño (N) del sistema y llega a la solución en un corto tiempo. Sin embargo, tiene ciertas desventajas como, por ejemplo, la matriz de coeficientes (A) debe ser simétrica y definida positiva. Por ello, para utilizar el algoritmo de Gradientes Conjugados se debió hacer unos cálculos previos para trabajar con una matriz distinta a la del sistema planteado.

En cuanto a los otros métodos iterativos, Jacobi, Gauss-Seidel y SOR tienen la ventaja de que, si la matriz de coeficientes es definida positiva, entonces tienen convergencia asegurada. Además, sus matrices de iteración T son relativamente sencillas de calcular. Claramente la desventaja es que tienen una convergencia muy lenta, principalmente Jacobi. El método SOR es bastante eficiente para un parámetro de relajación ‘w’ óptimo, pero tiene la desventaja de que deben realizarse varios cálculos previos para obtener ese valor, o realizar varias pruebas hasta hallar un valor aproximado al óptimo.

Para el método de eliminación de Gauss vimos se obtuvo la solución en un tiempo relativamente corto en comparación a otros métodos, por lo que podemos decir que resultó ser bastante eficiente para el tipo de matriz de este ejercicio, ya que en la diagonal principal se encontraban los elementos de mayor valor absoluto, por lo que se evitó tener que hacer intercambios de filas y mayores operaciones.