Problemas

Ejercicio 1: Resuelva el siguiente sistema de ecuaciones lineales con todos los métodos presentados en esta guía. Utilice una tolerancia en el residuo de 10^{-5} para la convergencia. En cada caso, estime el costo del método en función del tiempo reloj de ejecución (comandos tic y toc de Octave), el número de iteraciones, tasa de convergencia (realizar graficas semilogarıtmicas norma del residuo vs. numero 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:

$$\begin{cases} 2i & Si \;\; j = i \\ 0.5i & Si \;\; j = i + 2 \; o \; j = i - 2 \\ 0.25i & Si \;\; j = i + 4 \; o \; j = i - 4 \\ 0 & En \;\; otra \; posicion \end{cases}$$

Con *i=1,2,…, N y N=250, 500,1000.* Las entradas del vector de términos independientes son $b_i = \pi$. Utilice norma infinito.

Los métodos que se utilizaron para la resolución de este problema fueron el método de Jacobi, Gauss-Seidel, Sobre-relajaciones Sucesivas (Succesive Over Relaxation, SOR) y Gradiente Conjugado, así como también el método directo de Gauss sin pivoteo (más adelante se explica porque esta decisión).

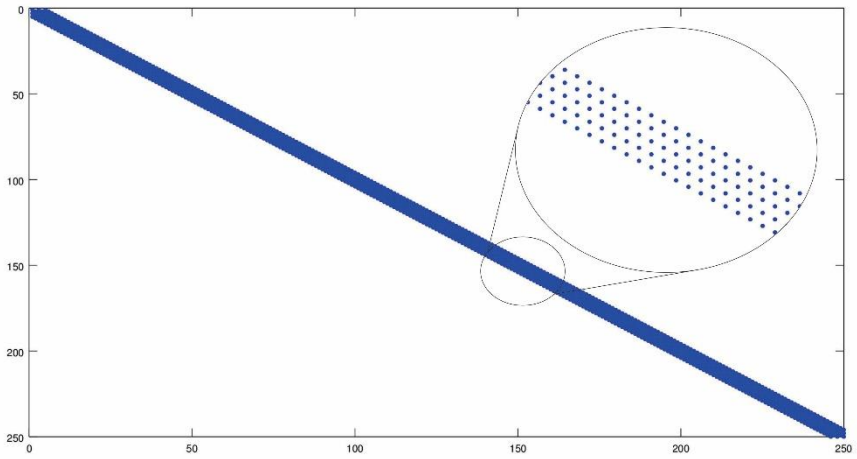
Todos los cálculos y operaciones se realizaron en una computadora con las siguientes características:

Procesador	2x Intel Atom 1.66Ghz
Memoria	1014 Mb
Sistema Operativo	Ubuntu 15.04

Utilizando el software GNUOctave versión 3.8.2.

En primer lugar se analizó la matriz para desarrollar hipótesis previas sobre la eficacia y convergencia de los métodos a utilizar.

A simple vista se trata una matriz donde sus elementos eran, en su mayoría, 0.



La matriz se graficó mediante la función ***spy(A)*** de octave, la cual lo que hace es colocar un punto en la posición de un elemento distinto de 0, donde encuentra un 0 lo deja en blanco.

Luego se procedió a determinar si se trataba de una matriz diagonal dominante.

Dado que para una fila cualquiera *i* se tiene que el elemento de la diagonal principal será ***2i*.** Para el la misma fila ***i*** se obtiene que

$$\sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{i,j}| = 1.5i$$

Es decir que para una misma fila ***i*** Se cumple que

$$\sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{i,j}| < |a_{ii}| \; esto \; es \; |1.5i| < |2i|$$

Por esto se demostró que es una matriz estrictamente diagonal dominante

Lo que garantiza la convergencia de los tres primeros métodos que se utilizaron, para cualquier $x^{(0)}$ y el que no fuera necesaria una estrategia de pivoteo para el método directo de Gauss.

En cuanto al gradiente conjugado no se pudo garantizar convergencia en vista que la matriz no es simétrica, es decir

$$a_{i,j} \neq a_{j,i} \; con \; i \neq j$$

Todos los procedimientos se descriptos se realizaron sin importar el tamaño de la matriz, ya que este no influía en los estudios realizados

Se prosiguió con calcular el número de condición de la matriz, para sus tres tamaños, N=250, 500,1000, por lo que se obtuvieron tres números de condición, uno para cada tamaño posible.

N	250	500	1000
$k(A)$	454.61	921.17	1885.00

Para este cálculo se utilizó el siguiente algoritmo en Octave

```
1 function [k]=NroCondicion(A)
2    k=max(abs(eig(A)))/min(abs(eig(A)));
3 endfunction
```

Como los números de condición obtenidos son mucho mayor a 1 se dice que la matriz está mal condicionada, esto implicaría que el método de gradiente conjugado perdiera precisión, se debería de utilizar un método de condicionamiento de la matriz, pero este proceso excede el alcance del problema planteado, por lo cual se procedió con el método de gradiente conjugado ordinario.

Una vez finalizado el análisis de la matriz y planteadas algunas hipótesis se procedió al análisis de los métodos y aplicación de los mismos.

Todas las resoluciones fueron para sistemas de la forma $Ax = b$.

Las tres primeras metodologías correspondían a una forma de recurrencia dad por:

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

Como se construyen la matriz T y el vector c es particular de cada caso y se detalló más adelante, pero todos parten de la misma descomposición.

$$A = L + D + U$$

$$Con \begin{cases} L\ Matriz\ triangular\ inferior\ de\ elementos\ a_{i,j} \in A\ con\ i = 1,..,n\ y\ j = 1,\dots,i-1 \\ D\ Matriz\ Diagonal\ de\ elementos\ a_{i,i} \in A\ con\ i = 1,..,n \\ U\ Matriz\ triangular\ superior\ de\ elementos\ a_{i,j} \in A\ con\ i = 1,..,n\ y\ j = i+1,\dots,n \end{cases}$$

Remplazando en $Ax = b$ se obtuvo:

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

El método de Gradiente Conjugado es una forma de iteración por subespacio, por tanto su metodología de trabajo se explicó más adelante.

El algoritmo que se utilizó para generar la matriz fue el siguiente

```
function [A b]=CrearMatriz(n,val_b)
 A=diag([2:2:2*n]);
 A=A+diag([0.5:0.5:0.5*(n-2)],2);
 A=A+diag([1.5:0.5:0.5*n],-2);
 A=A+diag([0.25:0.25:0.25*(n-4)],4);
 A=A+diag([1.25:0.25:0.25*n],-4);
 b=zeros(n,1).+val_b;
endfunction
```

Se plantearon los siguientes algoritmos que se utilizaron para métodos directos estacionarios:

```
function [L D U]=DescomponerMatriz(A)
 n=length(A);
 v=diag(A);
 L=tril(A,-1);
 U=triu(A,1);
 D=diag(v);
endfunction
```

I Algoritmo Para Descomponer Matriz

```
function [ro]=RadioEspectral(A,metodo,w)
[L D U]=DescomponerMatriz(A);
if(metodo=='ja')
  [ro pos]=max(abs(eig(-inv(D)*(L+U))));
else
if(metodo=='gs')
  [ro pos]=max(abs(eig(-inv(D+L)*(U))));
else
  [ro pos]=max(abs(eig(-inv(D+w*L)*((w-1)*D+w*U))));
endif
endif
endfunction
```

II Algoritmo Calcular Radio Espectral

- Método de Jacobi:

  Se partió de:

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

  Trabajando la ecuación se llegó a:

$$x = -D^{-1}(L + U)x + D^{-1}b$$

$$si\ T = -D^{-1}(L + U)\ y\ \ c = D^{-1}b$$

$$Escribiendo\ la\ forma\ iterativa\ se\ obtiene\ el\ sistema\ x^{(k)} = Tx^{(k-1)} + c$$

  Por tanto la iteración de Jacobi se basa en las siguientes operaciones

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

```
1  function [x,it,r_h, time]=Jacobi(A,b,x0,MaxIt,Tol)
2  tic();
3  n=length(A);
4  y=zeros(n,1);
5  x=x0;
6  for k=1:MaxIt
7    for i=1:n
8      y(i,1)=(b(i,1)-sum(A(i,1:(i-1))*x(1:(i-1)))-
9          sum(A(i,(i+1):n)*x((i+1):n)))/A(i,i);
10   endfor
11   x((1:n),1)=y((1:n),1);
12   it=k;
13   val=norm(x-x0,'inf')/norm(x,'inf');
14   r_h(k)=norm(A*x-b,'inf');
15   if(val<Tol)
16    break;
17   endif
18   x0=x;
19 endfor
20 r_h=r_h';
21 time=toc();
22 endfunction
```

III Algoritmo Jacobi

Tamaño matriz	250x250	500x500	1000x1000
Nro. Iteraciones	32	32	32
Tiempo en Segundos	4.8770	8.5760	17.132
$\rho(T_j)$	0.7495	0.7499	0.7500

- Método de Gauss-Seidel:

De manera inicial se planteo

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

Trabajando la ecuación se llega a:

$$x = -(D + L)^{-1}Ux + (D + L)^{-1}b$$

$$si\ T = -(D + L)^{-1}U\ y\ \ c = (D + L)^{-1}b$$

$$Escribiendo\ la\ forma\ iterativa\ se\ obtiene\ el\ sistema\ x^{(k)} = Tx^{(k-1)} + c$$

De esta forma se genera una matriz triangular lo que nos permite utilizar un elemento anterior calculado en la misma iteración, y no tener que esperar a la siguiente iteración como Jacobi, logrando así una velocidad mayor.

Por tanto la iteración de Gauss-Seidel se basa en las siguientes operaciones

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

```
function [x,it,r_h,time]=GaussSeidel(A,b,x0,MaxIt,Tol)
tic();
n=length(A);
y=zeros(n);
x=x0;
y=x0;
for k=1:MaxIt
  for i=1:n
    y(i,1)=(b(i,1)-sum(A(i,1:(i-1))*y(1:(i-1)))
        -sum(A(i,(i+1):n)*x((i+1):n)))/A(i,i);
  endfor
  x((1:n),1)=y((1:n),1);
  it=k;
  val=norm(x-x0,'inf')/norm(x,'inf');
  r_h(k)=norm(A*x-b,'inf');
  if(val<Tol) break;
  endif
```

```
18   x0=x;
19 endfor
20 r_h=r_h';
21 time=toc();
22 endfunction
```

IV Algoritmo Gauss-Seidel

Tamaño matriz	250x250	500x500	1000x1000
Nro. Iteraciones	8	8	8
Tiempo en Segundos	1.2460	2.2264	4.3646
$\rho(T_{gs})$	0.2294	0.2297	0.2344

- Método Sobre-relajaciones Sucesivas (Succesive Over Relaxation, SOR):

Como primer detalle cabe mencionar que el SOR para este trabajo se aplicó al algoritmo iterativo de Gauss-Seidel pero que se puede utilizar en el algoritmo de Jacobi, se opta por Gauss-Seidel por tener una velocidad mayor.

En principio la obtención del sistema es el mismo solo que se le agrega un término ω propio del método.

$$1 < \omega < 2 \; Para \; que \; se \; trate \; de \; un \; SOR$$

Agregando este término se obtiene una velocidad de convergencia mayor a al método normal

Se partió de la descomposición de A y se la multiplico por el término omega:

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

Para este caso se demostró cómo se alcanzó la expresión de la matriz T de SOR:

$$\big(D - D + \omega(L + D + U)\big)x = b \Rightarrow (D + \omega L)x + (\omega D - D + \omega U)x = b$$

$$x = -(D + \omega L)^{-1}\big((\omega - 1)D + \omega U\big)x + (D + \omega L)^{-1}b$$

$$si \; T = -(D + \omega L)^{-1}\big((\omega - 1)D + \omega U\big) \; y \;\; c = (D + \omega L)^{-1}b$$

$$Escribiendo \; la \; forma \; iterativa \; se \; obtiene \; el \; sistema \; x^{(k)} = Tx^{(k-1)} + c$$

Por tanto la iteración SOR se basa en las siguientes operaciones

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

Se puede observar que si ω=1 se obtiene Gauss-Seidel.

Para el caso planteado en el problema no se pudo encontrar ningún omega tal que mejorara el rendimiento del método Gauss-Seidel.

Mediante un proceso de prueba y error se encontró un omega óptimo para la metodología de SOR.

```
function [x,it,r_h,time]=SOR(A,b,x0,MaxIt,Tol,w)
tic();
n=length(A);
y=zeros(n);
x=x0;
y=x0;
for k=1:MaxIt
  for i=1:n
    y(i,1)=(1-w)*x(i,1)+w*(b(i,1)-sum(A(i,1:(i-1))*y(1:(i-1)))
        -sum(A(i,(i+1):n)*x((i+1):n)))/A(i,i);
  endfor
  x((1:n),1)=y((1:n),1);
  it=k;
  val=norm(x-x0,'inf')/norm(x,'inf');
  r_h(k)=norm(A*x-b,'inf');
  if(val<Tol) break;
  endif
  x0=x;
endfor
r_h=r_h';
time=toc();
endfunction
```

V Algoritmo SOR

ω=1.04844			
Tamaño matriz	250x250	500x500	1000x1000
Nro. Iteraciones	9	9	9
Tiempo en Segundos	1.6386	2.8318	5.3770
$\rho(T_s)$	0.2143	0.2146	0.2244

- <u>Gradiente Conjugado:</u>

Para este método se plantea una forma cuadrática asociada al sistema a resolver

$$\Phi(x) = \frac{1}{2}x^t Ax - b^t x$$

Se puede demostrar que el x que minimiza esta función es solución del sistema $Ax = b$. Esta demostración no es necesaria para entender los procedimientos que se efectuaron en el trabajo, pero es un detalle es importante porque lo que se hizo fue buscar mediante iteraciones el mínimo de la función.

Para hallar este mínimo se planteó:

$$x^{(k)} = x^{(k-1)} + \alpha^{(k-1)} p^{(k-1)}$$

Donde p es la dirección de descenso y alfa es el paso en esa dirección

P se calcula sabiendo que si el gradiente de una función es la dirección de mayor crecimiento de esta, el opuesto del gradiente será la de mayor decrecimiento, por tanto:

$$p^{(k-1)} = \frac{\partial \Phi}{\partial x}\Big|_{x^{(k-1)}}$$

Se busca que la dirección de descenso p en cada iteración se conjugada a las anteriores

Para que dos direcciones son conjugadas si

$$p^{(i)^t} A p^{(j)} = \begin{cases} 0 \; si \; j \neq i \\ > 0 \; si \; j = i \end{cases}$$

De aquí proviene el nombre del método.

A continuación se presenta el algoritmo usado en octave

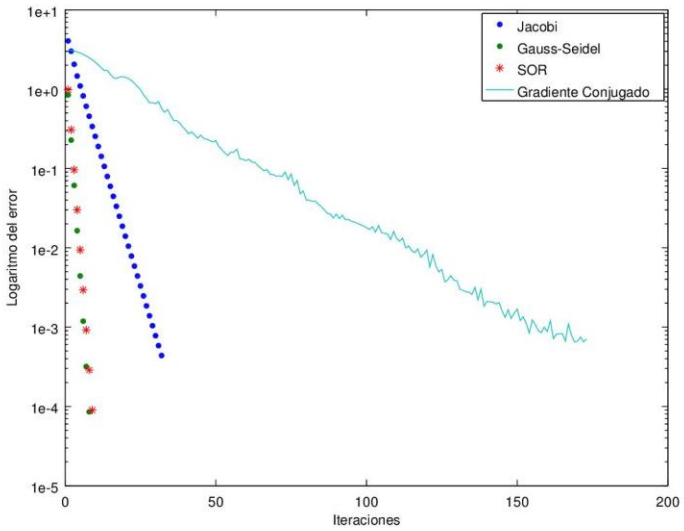
```
function [x,it,r_h,time] = Gradiente_Conjugado(A,b,x0,MaxIt,Tol)
tic();
r=b-A*x0;
p=r;
ro=r'*p;
ro_old=ro;
ro0=ro;
k=1;
x=x0;
for k=1:MaxIt
 it=k;
  %r_h=ro_old;
  a=A*p;
  m=p'*a;
  alfa=ro/m;
  x0=x;
  x=x+alfa*p;
  val=norm(x-x0,'inf')/norm(x,'inf');
  r_h(k)=norm(A*x-b,'inf');
```

```
20    if(val<Tol)break;
21   endif
22   r=r-alfa*a;
23   ro_old=ro;
24   ro=r'*r;
25   lamdba=ro/ro_old;
26   p=r+lamdba*p;
27 endfor
28 time=toc();
29 endfunction
```

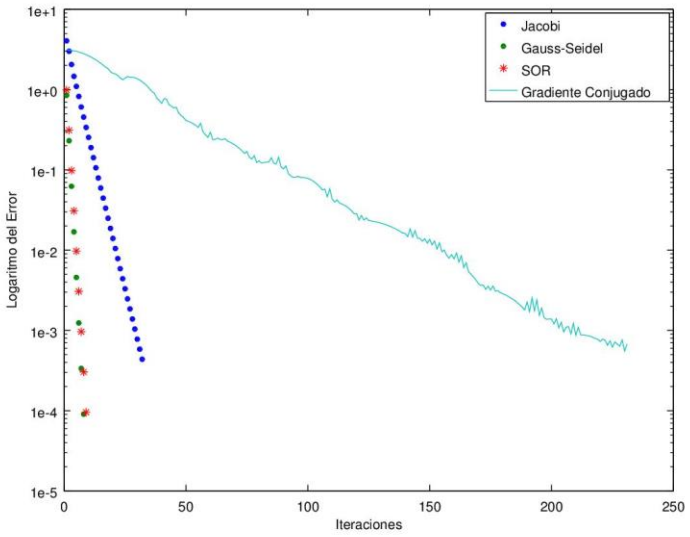
VI Algoritmo Gradiente Conjugado

Tamaño matriz	250x250	500x500	1000x1000
Nro. Iteraciones	173	231	282
Tiempo en Segundos	0.4109	1.1716	4.3180

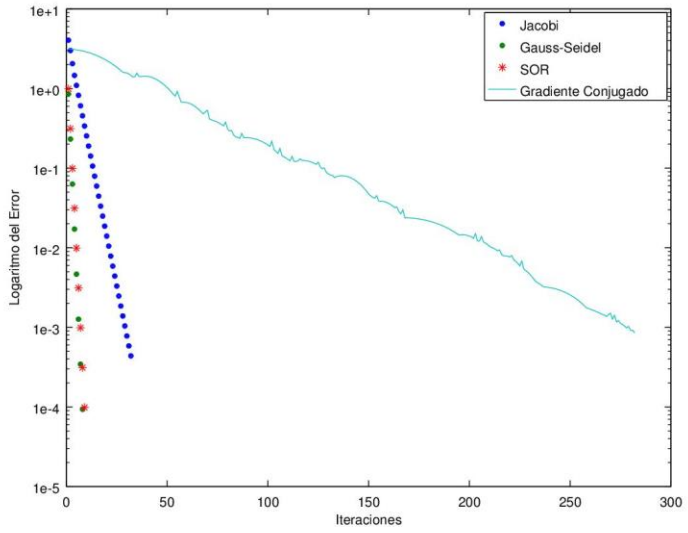
Los siguientes gráficos sirven tener una mejor comparación entre los diferentes métodos iterativos mencionados.



VII Iteraciones Vs. Log Error N=250



VIII Iteraciones Vs. Log Error N=500



IX Iteraciones Vs. Log Error N=1000

Todas los gráficos se realizaron usando la función “semilogy” de octave, la cual permite graficar el logaritmo de un argumento en el eje y, en este caso los errores pertenecientes a cada método en cada iteración, en el eje de las x, se ubica el total de iteraciones para cada método.

Luego se utilizó el método directo de Gauss para resolver el mismo sistema. Como ya se mencionó previamente al ser una matriz estrictamente diagonal dominante, no fue necesario recurrir a una metodología de pivoteo.

```
1  function [U x b time]=GaussExtend(A,b)
```

```
2  tic();
3  n=length(A);
4  for k=1:n-1
5  s=A(k+1:n,k)/A(k,k);
6  A(k+1:n,k)=0;
7  A(k+1:n,k+1:n)=A(k+1:n,k+1:n)-s*A(k,k+1:n);
8  b(k+1:n)=b(k+1:n)-s*b(k);
9  endfor
10 x=Rest_atras(A,b);
11 U=A;
12 time=toc();
13 endfunction
```

X Algoritmo Gauss

```
1  function [x] = Rest_atras (A,b);
2  v=[size(b)];
3  x=zeros(v(1),1);
4  n=v(1);
5  x(n)=b(n)/A(n,n);
6  y=0;
7  for i=(n-1):-1:1
8  y=y+A(i,(i+1):n)*x((i+1):n,1);
9  x(i,1)=(b(i,1)-y)/A(i,i);
10 y=0;
11 endfor
12 endfunction
```

XI Algoritmo Restitucion hacia Atras

Mediante Gauss se obtuvieron los siguientes datos:

Tamaño Matriz (N)	Tiempo en segundos
250	0.8751
500	3.4530
1000	26.385

Para una mayor comparación entre los métodos iterativos y el directo se confeccionaron las siguientes tablas, tomando el error (residuo) de la última iteración para realizar la comparación con el error del método directo.

- Para N=250

Método	Tiempo en segundos
Jacobi	4.8770
Gauss-Seidel	1.2460

SOR	1.6386
Gradiente Conjugado	0.4109
Gauss (directo)	0.8751

- Para N=500

Método	Tiempo en segundos
Jacobi	8.5760
Gauss-Seidel	2.2264
SOR	2.8318
Gradiente Conjugado	1.1716
Gauss (directo)	3.4530

- Para N=1000

Método	Tiempo en segundos
Jacobi	17.132
Gauss-Seidel	4.3646
SOR	5.3770
Gradiente Conjugado	4.3180
Gauss (directo)	26.385

A partir de todos los datos encontrados se determinaron las siguientes conclusiones. Los métodos directos son más eficientes y exactos para matrices de pocos elementos, pero a medida que la matriz aumenta la eficiencia se va perdiendo, y pasan a ser una mejor alternativa los métodos iterativos, aunque si bien no otorgan una solución exacta, logran una aproximación muy cercana a la real.

El método del gradiente conjugado tiene una mayor cantidad de iteraciones, pero realiza las operaciones en una velocidad mucho menor que el resto de los algoritmos.

Para este trabajo se puede ver que los métodos que obtuvieron un mejor rendimiento en general fueron el SOR y el de Gauss-Seidel, esto se debe a que el radio espectral de sus respectivas matrices era menor que el de Jacobi. El método del gradiente conjugado perdió exactitud por el mal condicionamiento que posee la matriz, como se había supuesto en un principio, si se quisiera emplear este método de manera más exacta y eficiente se debería de recurrir a una estrategia de acondicionamiento de la matriz.

Ejercicio 2: Resuelva los siguientes sistemas lineales con el método de Gauss-Seidel y analice que sucede en cada caso. ¿Es necesario utilizar alguna estrategia de pivoteo? Si lo fuera justifique por qué.