Gradientes Conjugados.

Marco teórico

Se propone resolver el sistema lineal de ecuaciones de la forma $A.x = b$, con $A \in \mathbb{R}^{nxn}$, $b;x \in \mathbb{R}^n$ a partir del método de gradientes conjugados. Para este método se propone una función ϕ: $\mathbb{R}^n \to \mathbb{R}$ de la siguiente forma:

$$\phi(x) = \frac{1}{2}\, x^t.A.x \; - \; b^t.x$$

Luego se propone hallar el mínimo de esta función, que es el punto donde x es la solución al problema inicial. Veamos que esto es así: Sabemos que si ésta función posee un mínimo este se encontrará en el punto donde se anulen todas sus derivadas parciales. tomemos entonces el gradiente de ϕ=0:

$$\nabla \phi = A.x - b \; = 0$$

como se puede ver, hallar la solucion al sistema $A.x = b$ tiene el mismo resultado que hallar el mínimo de la función ϕ.
El método de los gradientes conjugados se puede aplicar sólo cuando A es simétrica y definida positiva.
Una particularidad de este método es que se propone descender sobre la superficie descripta por ϕ de tal manera de acercarse al mínimo, cambiando de dirección cada vez que se llega al mínimo de la dirección antes elegida. Existe una condición en la selección de la dirección de descenso: cada dirección elegida debe ser perpendicular a la anterior. Aplicando esa condición, se llega a que el método tiene un orden de convergencia de n direcciónes de descenso, pues, la dirección n+1 no puede ser perpendicular a todas las demás.

Resultados:

Se midieron los tiempos de resolución de un problema de difusión discretizado para distintos tamaños de n con matrices llenas y mattrices ralas. se utilizó n entre 1 y 3001 con un paso de 40 de esta manera se obtuvieron 75 tiempos distintos para cada tipo de matriz.

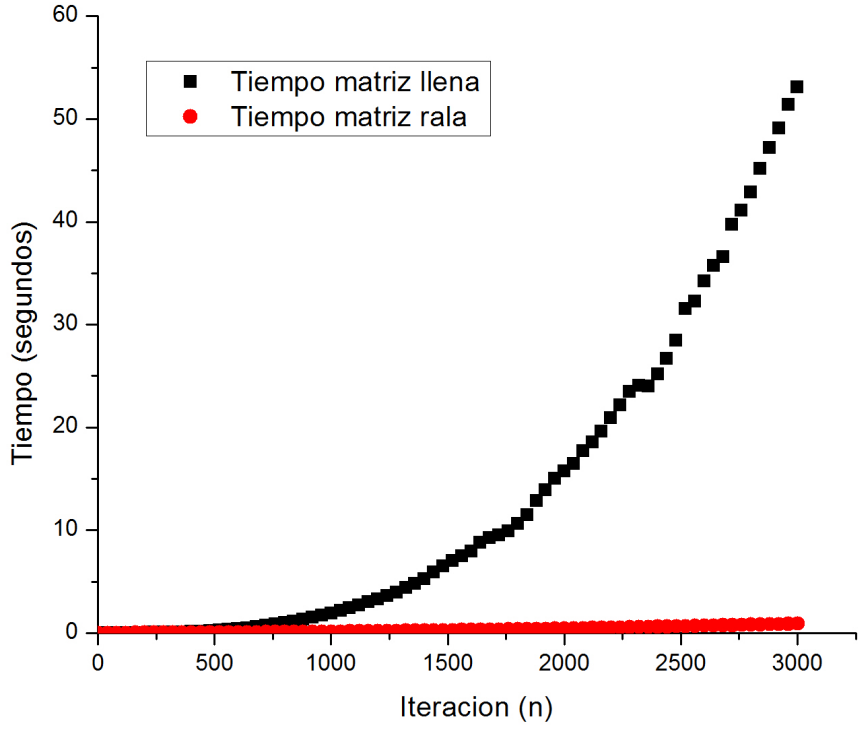

Figura 1: tiempos de resolución para numero de iteraciones entre 1 y 3001 con matrices llenas (los puntos de arriba) y matrices ralas (los puntos de abajo).

Se puede observar en el grafico que los tiempos de resolución para matrices ralas son mucho menores, es decir mayores a un orden de magnitud para n=3000. en cambio para n pequeños la diferencia no es de tal magnitud veamos la **Figura 2:**

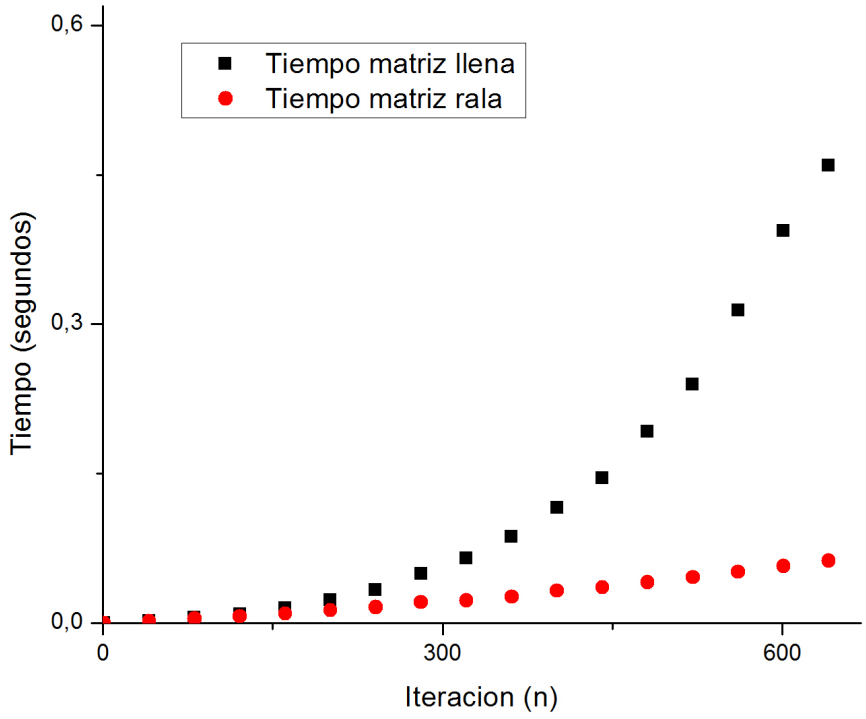

Figura 2: tiempos de resolución para numero de iteraciones entre 1 y 500 con matrices llenas (los puntos de arriba) y matrices ralas (los puntos de abajo).

En la **Figura 2** puede verse que los puntos correspondientes a las matrices ralas son menores que para las matrices llenas pero en este caso la diferencia no es mayor que un orden de magnitud. Esto se debe a que a medida que crece el n crece junto con este el numero de ceros en la matriz llena lo cual alentiza el proceso de resolución.

La forma de la matriz *A* que se utilizó es la siguiente para n=5:

$$A = \begin{pmatrix} 2 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{pmatrix}$$

Es decir una matriz cuya cantidad de elementos no nulos es *n+2.(n-1)* y su cantidad total de elementos es n^2. De esto se puede concluir que la cantidad de elementos nulos en la matriz es $n^2 - 3n + 2$ para n muy grande el término predominante es el cuadrado, esto explica por qué se reduce tanto la cantidad de tiempo utilizado en la resolución.

Codigo del programa

El siguiente es el código utilizado, para realizar la resolución del problema y las mediciónes correspondientes:

```
rand("seed",1.058)
tol=1E-10;
for n=1:40:3001
S1 = sparse([1:n], [1:n], 2*ones(1,n), n, n, 0);
```

```
S2 = sparse([1:n-1], [2:n], -1*ones(1,n-1), n, n, 0);
S3 = sparse([2:n], [1:n-1], -1*ones(1,n-1), n, n, 0);
A = full(S1 + S2 + S3); # Si se quiere la matriz llena
b = ones (1,n)/(n + 1)**2; # Para generar el vector segundo miembro
r=zeros(n,1);
x=rand(n,1);
b=b';
t1=time();

r=b-A*x;
k=0;
d=r;
alfa=(transpose(d)*r)/(transpose(d)*A*d);
x=x+alfa*d;
r1=r;
r=b-A*x;
while(norm(r)>tol && k<n) % lo del k es para que corte si tarda mucho
        k=k+1;
        bet=(norm (r)^2)/(norm (r1)^2);
        d=r+bet*d;
        alfa=(transpose(d)*r)/(transpose(d)*A*d);
        x=x+alfa*d;
        r1=r;
        r=b-A*x;
endwhile
t2=time();
# Tllen((n-1)/40+1)=t2-t1;
it_llen((n-1)/40+1)=k;
endfor #fin del for en n

# Resolucion de las ralas
for n=1:40:3001
S1 = sparse([1:n], [1:n], 2*ones(1,n), n, n, 0);
S2 = sparse([1:n-1], [2:n], -1*ones(1,n-1), n, n, 0);
S3 = sparse([2:n], [1:n-1], -1*ones(1,n-1), n, n, 0);
A = S1 + S2 + S3; # Si se quiere la matriz en formato RALO
b = ones (1,n)/(n + 1)**2; # Para generar el vector segundo miembro
r=zeros(n,1);
x=rand(n,1);
b=b';
t1=time();

r=b-A*x;
k=0;
d=r;
alfa=(transpose(d)*r)/(transpose(d)*A*d);
x=x+alfa*d;
r1=r;
r=b-A*x;
```

```
while(norm(r)>tol && k<n) % lo del k es para que corte si tarda mucho
        k=k+1;
        bet=(norm (r)^2)/(norm (r1)^2);
        d=r+bet*d;
        alfa=(transpose(d)*r)/(transpose(d)*A*d);
        x=x+alfa*d;
        r1=r;
        r=b-A*x;
endwhile

t2=time();
Tral((n-1)/40+1)=t2-t1;
it_ral((n-1)/40+1)=k;
endfor
q(:,1)=Tllen;
q(:,2)=Tral;
save Datos3P4E2.dat q;
printf(“fin del programa”)
```