Métodos Numéricos

Trabajo Práctico N°1

Profesor a cargo: *Diego Passarella.*

Profesor instructor: *Carlos Caldart.*

Alumnos: *Fernando Amor, Darío Blanco, Federico Rodríguez,*

Fecha: *02/11/12*

Resumen:

En este trabajo práctico se buscó comparar velocidades de resolución de diferentes métodos numéricos bajo ciertas condiciones. Para ello se operó con matrices tridiagonales de distinto tamaño. Los resultados se dispusieron en forma de gráfico para un análisis más sencillo.

Introducción:

La ciencia moderna depende en gran parte de la capacidad de llevar a cabo sustanciales computaciones para explorar las consecuencias de las leyes de la naturaleza. Estas leyes están expresadas como modelos matemáticos, y se utilizan métodos numéricos a través de la computación. El uso de software matemático de alta calidad permite resolver, de forma exacta o aproximada, problemas matemáticos que serían casi imposibles de ser abordados por métodos analíticos.

Los métodos numéricos son, entonces, los algoritmos que permiten resolver lo estipulado anteriormente. En este trabajo práctico se utilizaron para resolver sistemas de ecuaciones lineales y estudiar la estabilidad de varios sistemas.

Los métodos para resolver sistemas de ecuaciones lineales se dividen en dos grandes ramas, de acuerdo a la forma de abordar los problemas: métodos directos y métodos iterativos.

Los métodos directos son utilizados para resolver sistemas relativamente pequeños y con matrices densamente pobladas. Se basan en operar sobre la matriz sola o expandida, para llevarla a una forma que sea fácilmente resoluble. La solución, idealmente exacta, se obtiene en un finito número de operaciones, que puede ser calculado o aproximado de antemano.

En el trabajo práctico se utilizaron diversos métodos directos, a saber: eliminación de Gauss, algoritmos de descenso y remonte, factorización LU, fact. de Cholesky y fact. de Thomas.

La eliminación de Gauss es un método numérico directo que opera con el sistema completo (matriz de coeficientes y vector solución) para hacer cero a los elementos por debajo de la diagonal principal. Esta factorización existe y es única si las submatrices principales de orden 1 hasta n-1 (siendo n x n el tamaño de la matriz) son no singulares. El proceso requiere una cantidad de operaciones del orden de 2(n^3)/3.

Si la matriz no fuera diagonalmente dominante por filas o columnas, o simétrica y definida positiva, es necesario verificar los multiplicadores resultantes y/o pivotar.

Los algoritmos de remonte y descenso son utilizados para llevar a una matriz triangular superior o inferior, expandida, a su sistema equivalente más sencillo. En matrices cuadradas, para sistemas compatibles determinados, se resuelve el sistema.

La factorización LU es un método directo que descompone a una matriz **A** en dos matrices: **L** (*lower*), una triangular inferior; y **U** (*upper*), una triangular superior. Esta factorización no es única, por lo que se impone que la diagonal principal de **L** esté formada por números 1. Este método, si bien plantea resolver dos sistemas con matrices triangulares, resulta útil en el caso de tener que resolver sistemas con una misma matriz A y distintos vectores solución.

El método de factorización de Cholesky requiere que la matriz sea simétrica y definida positiva. Entonces se factoriza a la matriz A de la siguiente manera:

$$\mathbf{A = L\,L^t}$$

Siendo A entonces el producto de dos matrices triangulares donde una es la transpuesta de la otra. Esta factorización requiere de menos operaciones que su contraparte LU, y permite ahorrar memoria, pues solamente se almacenan los coeficientes de **L**.

La factorización de Thomas es la redefinición del algoritmo LU, volviéndolo específico para resolver matrices tridiagonales. Con esto se logra un costo computacional del orden de n operaciones.

La otra rama de métodos numéricos para resolver S.E.L. es la de los métodos iterativos. Estos algoritmos son utilizados prioritariamente en sistemas grandes y estructuradamente ralos. Entonces, generan una sucesión que aproxima a la solución por medio de la repetición de operaciones *matriz x vector* en cada iteración. El método empleado finaliza al haber logrado aproximar la solución bajo una dada tolerancia, o luego de haber empleado un número máximo de iteraciones definido previamente.

Los métodos iterativos utilizados en el trabajo práctico fueron la descomposición de Jacobi y la descomposición de Gauss Seindel, tanto en sus formas eficientes como en sus formulaciones por componentes. Ambos métodos planten una descomposición de la matriz **A** de la siguiente forma:

$$\mathbf{A = M - N}$$

Donde **M** es una matriz fácilmente invertible. De esta manera se genera un punto fijo con $\mathbf{G = M^{-1}\,N}$, $\mathbf{c = M^{-1}b}$. Entonces el sistema queda reescrito de la forma:

$$\mathbf{Ax = b \;\rightarrow\; x = Gx + c}$$

El método de descomposición de Jacobi plantea descomponer la matriz **A** en **D** (diagonal), **L** y **U**, de forma de generar un punto fijo con:

$$\mathbf{A = D - L - U}$$

$$\mathbf{M = D,\;\; N = L + U}$$

$$\mathbf{G = D^{-1}(L + U)}$$

De modo que el método iterativo sea:

$$\mathbf{x = Gx + c, \quad c = D^{-1}b}$$

La descomposición de Gauss Seindel es una reagrupación del método de Jacobi, donde:

$$\mathbf{M = D - L, \quad N = U}$$

De esta forma, los componentes de la k-ésima iteración se multiplican también por la matriz **L.**

La formulación eficiente de ambos métodos conlleva una generalización al aspecto de la matriz, y por consiguiente es versátil pero con cierto grado de desperdicio de recursos. Conociendo la disposición de los elementos en la matriz, pueden formularse los métodos por componentes, obteniéndose métodos más específicos pero sensiblemente más potentes.

El (buen) planteo de los problemas matemáticos puede analizarse de acuerdo a la continuidad de su solución. Se dice que un problema está bien planteado, o es estable, si la solución es única y varía de forma continua de acuerdo a los datos de entrada. Como contrapartida, un problema se halla mal planteado (o es inestable) si su solución varía de forma discontinua.

En el análisis de la calidad del planteo del problema matemático intervienen los llamados números de condición. Estos resultan de cocientes entre diferenciales del valor de la solución con diferenciales del valor de entrada, y cuanto más pequeños sean, mejor planteado estará el problema.

Un método numérico será consistente cuando la aplicación tendiendo a infinito del método no desvíe el valor de la solución. Si además está bien planteado, será también convergente.

Los números de condición empleados en el trabajo práctico fueron $\mathbf{K_2(A)}$, siendo estos obtenidos del cociente del mayor autovalor de la matriz **A** con el menor autovalor de la misma.

Para hallar estos números de condición, fue necesario emplear métodos iterativos para aproximar los autovalores necesarios. Los métodos fueron el de la Potencia Iterada, y el de la Potencia Inversa.

El método de la Potencia Iterada se basa en generar una sucesión tal que tienda al autovector asociado al autovalor de mayor módulo. Para ello emplea al cociente de Rayleigh para, en cada iteración, ir aproximando el autovalor deseado. Para garantizar la convergencia del método, la matriz **A** debe poseer un autovalor de módulo mayor al resto.

El método de la Potencia Inversa es análogo al anterior, pero plantea el problema para la inversa de la matriz **A**, pudiendo aproximar entonces el valor de la inversa del menor autovalor. Invirtiendo este último, recuero el menor autovalor de la matriz **A**. Análogamente al método de la Potencia Iterada, la matriz debe poseer

un autovalor de módulo menor al resto (para que su inverso sea de módulo mayor al resto).

Metodología experimental / Materiales y métodos:

Se programó una función generadora de matrices tridiagonales en base a números *n* fijos que determinaron el tamaño matricial directamente, y con las condiciones impuestas por el trabajo práctico. Esta función también generó vectores solución del sistema conformado por la matriz generada y un vector de incógnitas, donde cada incógnita equivalía al número 1. Se trabajó con 8 *n* distintos, a saber: 10, 30, 100, 300, 1000, 3000, 10000, 20000.

Se programó un script que por cada *n* generó una matriz *n* x *n* con su vector solución *b* mediante la anterior función. Además, se pre alocó espacio en memoria adecuado para variables temporales, para luego llamar a las funciones de los siete métodos numéricos: eliminación de Gauss, factorización LU, factorización de Cholesky, factorización de Thomas, descomposición de Jacobi y descomposición de Gauss Seidel.

Cada método se encargó de resolver el sistema conformado por la matriz generada su vector solución correspondiente.

Todos los llamados a función para los métodos fueron cronometrados con la función *tic… toc* de Matlab, almacenando el valor tiempo en segundos resultante en un vector. A modo de verificación rápida de la validez del método, se imprimieron por pantalla los promedios de la multiplicación de cada vector incógnita hallado traspuesto por el mismo vector.

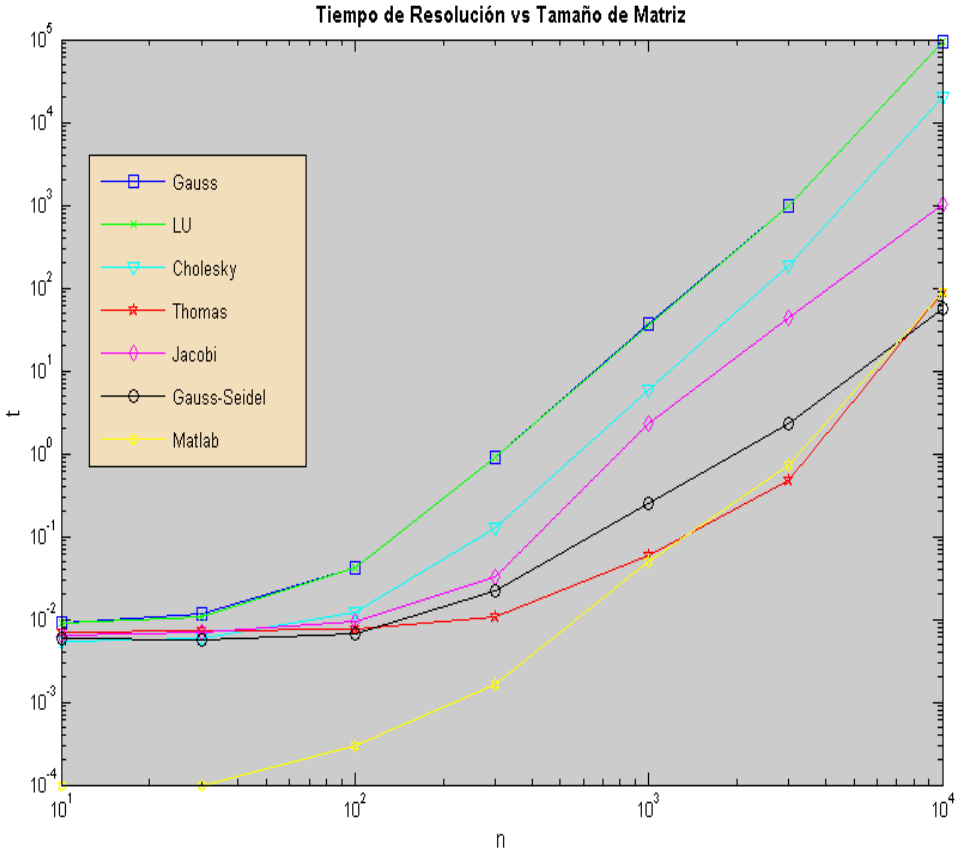
Se resolvieron además todos los sistemas mediante algoritmos de descomposición de Jacobi y Gauss Seidel formulados por componentes, con las mismas premisas que el resto de los métodos.

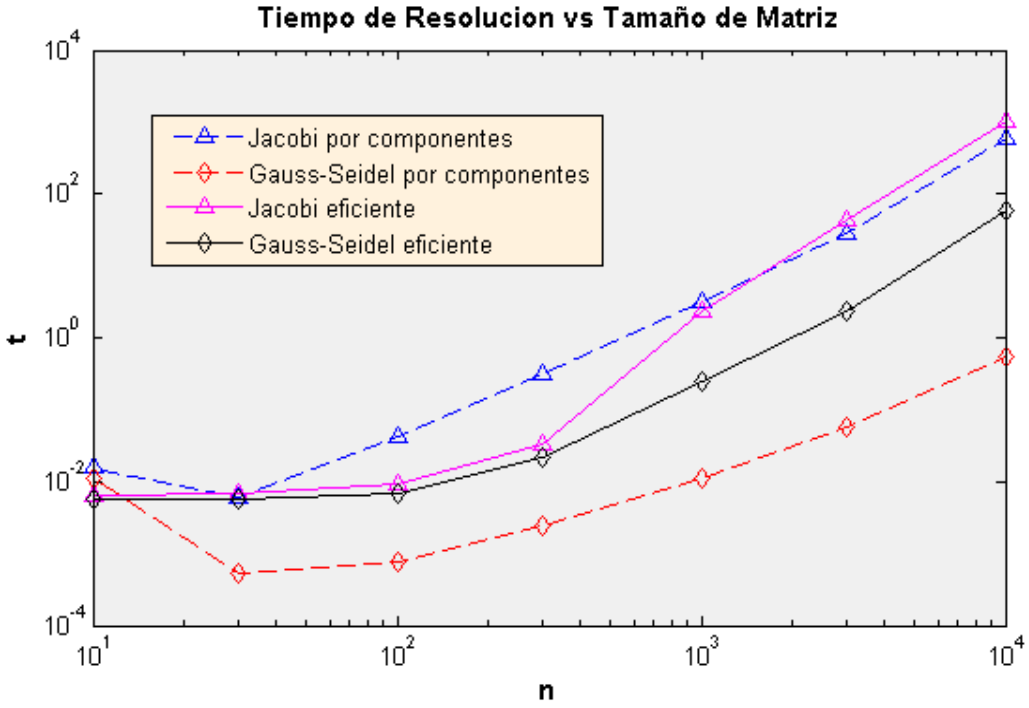
Luego se calcularon los números de condición K_2 de cada matriz *n* x *n* generada. Para ello, se codificó un script con llamados a funciones de métodos de Potencia Iterara y Potencia Inversa, que aproximó los mayores y menores autovalores (por módulo) de cada matriz, y luego realizó el cociente del mayor sobre el menor.

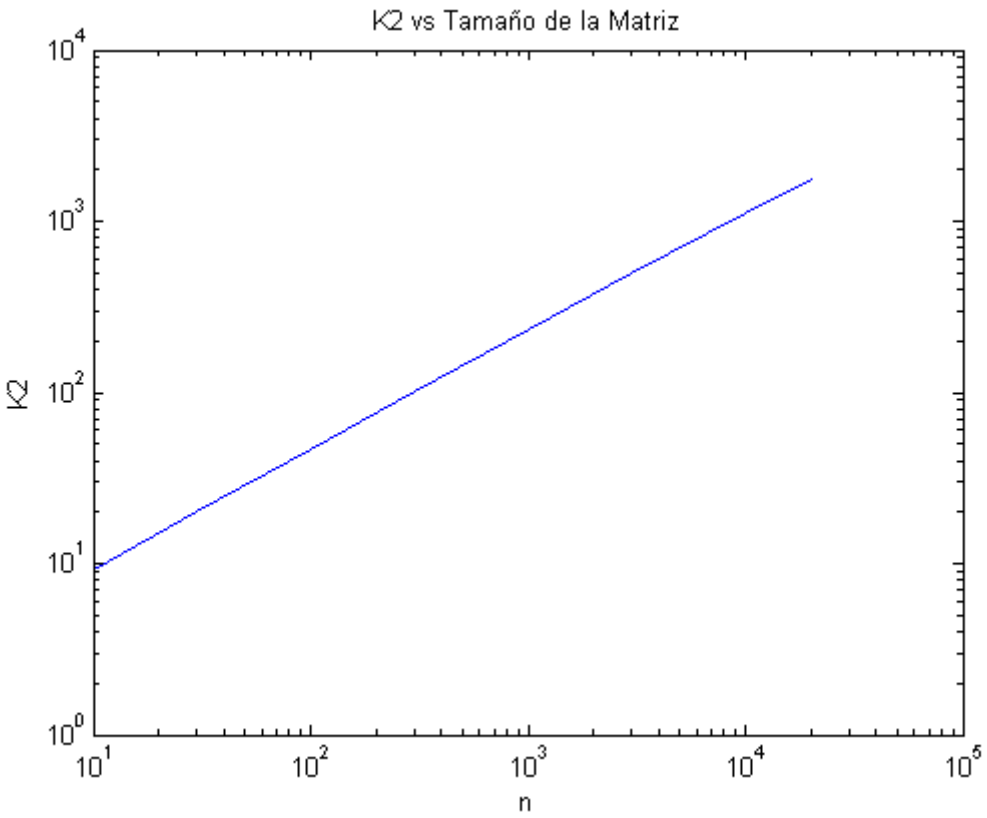
Finalmente, se procedió a graficar los datos obtenidos mediante la función de gráfico en escala logarítmica de Matlab, *loglog().*

Todos los tiempos fueron cronometrados en una PC con procesador Intel® Pentium® Dual-Core® E5200 (2.5 GHz), con 3GB de memoria RAM y Matlab® R2013b, sobre Windows® 7 x64.

Resultados:


Tiempo de Resolución vs Tamaño de Matriz
Gauss
LU
Cholesky
Thomas
Jacobi
Gauss-Seidel
Matlab
t
n


Tiempo de Resolucion vs Tamaño de Matriz
Jacobi por componentes
Gauss-Seidel por componentes
Jacobi eficiente
Gauss-Seidel eficiente
t
n



Discusión:

No pudieron calcularse los tiempos de ejecución de los métodos para la matriz de 20000 x 20000 por falta de hardware adecuado y/o tiempo. Hubo una sobrevaloración de la PC sobre la cual se realizó el trabajo práctico. Los tiempos calculados para los sistemas de menor tamaño variaban en demasía al repetir la ejecución del script, aun buscando que se ejecuten bajo similares condiciones. Esto es atribuible a los procesos de funcionamiento internos del sistema operativo, que aportan ruido estadístico.

Salvo para los n más grandes, el método de Jacobi por su formulación eficiente fue más veloz que su contraparte optimizada. Se cree que esto fue por haber utilizado funciones propias de Matlab para la construcción del primero.

Conclusión:

No se logró cumplir en su totalidad las consignas del trabajo práctico. Los resultados fueron los esperados, observándose una mejoría en la velocidad para los procesos optimizados a medida que aumenta el tamaño de la matriz. El condicionamiento del sistema empeoraba exponencialmente conforme aumentaba n, aunque esto fue ilustrativo dada la naturaleza del TP. Si hubiera que repetir el experimento en un futuro, se realizarían pruebas iniciales para verificar que pueden resolverse los sistemas de n más grandes en la computadora que se piensa trabajar.

Bibliografía:

-Rice, John R. *Numerical Methods, Software and Analysis*; McGraw-Hill: NY, USA, 1983

-Filminas de la materia Métodos Numéricos, segundo cuatrimestre 2016, UNQ.

Anexo: *scripts y funciones de Matlab utilizados*

-Funciones de métodos numéricos:

```
function [A2, b2] = elimGaussConPivote(A, b)
%Realiza la eliminación de Gauss de una matriz de nxm, pivotando en caso
de que sea necesario.
    tol = 10^-8;
    A2 = [A b];
    [n,m] = size(A2);
    for k=1:n-1
        if abs(A2(k,k)) < tol

            %Prepara y ejecuta pivote

            [noSeUsa,I] = max(abs(A2),[],1);
            p = I(k);       %Retengo el nro de fila donde se halla el
máximo valor en la columna k
            A2 = pivotEntreFilas(A2, p, k);
            fprintf('Hubo un pivot entre las filas %d', k);
            fprintf(' y %d', p);

            %Fin pivote

        end
        for i=k+1:n
            L(i,k)=A2(i,k)/A2(k,k);
            %Nyanga=A2(i,k)/A2(k,k);
            for j=k:n+1
                A2(i,j)=A2(i,j)-(L(i,k)*A2(k,j));
                %A2(i,j)=A2(i,j)-(Nyanga*A2(k,j));
            end
        end
    end
    b2 = (A2(:,m));
    A2 = (A2(:,1:m-1));
end
```

```
function [ Mat1 ] = pivotEntreFilas( Mat1, fila1, fila2 )
    %Retorna la matriz resultante de pivotar las filas fila1 y fila2 de
Mat1
    [n,noSeUsa2] = size(Mat1);
    P = eye(n,n);
    P(fila2,fila2) = 0;
    P(fila1,fila1) = 0;
    P(fila1,fila2) = 1;
    P(fila2,fila1) = 1;
    Mat1 = (P*Mat1);
end
```

```
function [X] = algoritmoDescenso(L,b)
%Realiza la transformación a la forma equivalente más simple de una matriz
%triangular inferior.
    [n,m] = size(L);
    X = zeros(n,1);
    X(1) = b(1) / L(1,1);
    for i=2:n
        sumatoria = 0;
        for j=1:i-1
            sumatoria = sumatoria + X(j)*L(i,j);
        end
        X(i) = (b(i) - sumatoria) / L(i,i);
    end
end


function [X] = algoritmoRemonte(A,b)
%Retorna el sistema equivalente más sencillo de la matriz A y su vector
%solución b
    A2=[A b];
    [n,m]=size(A2);
    %X=NaN(n,1); %podría hacerse, pero también utilizar directamente a x
    %como vector columna
    X(n,1)=b(n)/A2(n,n);
    for i=n-1:-1:1
        suma = 0;
        for j=i+1:n
            suma=suma+X(j)*A(i,j);
        end
        X(i)=(b(i)-suma)/A2(i,i);
    end
end


function [ L, U ] = algoritmoLU( A )
    %Retorna las matrices L y U correspondientes a la matriz A

    tol = 10^-8;
    [n,m] = size(A);
    L = zeros(n,m);
    Ident = eye(n,m);

    for k=1:n-1
        if abs(A(k,k)) < tol

            %Prepara y ejecuta pivote

            [noSeUsa,I] = max(abs(A),[],1);
            p = I(k);       %Retengo el nro de fila donde se halla el
máximo valor en la columna k
            A = pivotEntreFilas(A, p, k);
            fprintf('Hubo un pivot entre las filas %d', k);
            fprintf(' y %d', p);

            %Fin pivote

        end
        for i=k+1:n
            L(i,k)=A(i,k)/A(k,k);
            for j=k:n
                A(i,j)=A(i,j)-(L(i,k)*A(k,j));
```

```
            end
        end
    end

    U = A;
    L = L + Ident;
end
```

```
function [ L ] = algoritmoCholesky( A )
    %Realiza la factorizacion de Cholesky sobre la matriz A, retornando el
resultado
    %La matriz A debe ser simetrica y sus coeficientes definidos positivos

    [n,m] = size(A);
    L = zeros(n,m);

    if A(1,1) < 0
        error('Argumento de raiz negativo')
    end
    L(1,1) = A(1,1)^(1/2);
    for i=2:n
        L(i,1) = A(i,1)/L(1,1);
    end

    for j=2:n-1
        Sumatoria1 = 0;
        for k=1:j-1
            Sumatoria1 = Sumatoria1 + (L(j,k)^2);
        end
        L(j,j) = A(j,j) - Sumatoria1;
        if L(j,j) < 0
            error('Argumento de raiz negativo')
        end
        L(j,j) = L(j,j)^(1/2);
        for i=j+1:n
            Sumatoria2 = 0;
            for k=1:j-1
                Sumatoria2 = Sumatoria2 + (L(i,k)*L(j,k));
            end
            L(i,j) = (A(i,j) - Sumatoria2)/L(j,j);
        end
    end

    Sumatoria3 = 0;
    for k=1:n-1
        Sumatoria3 = Sumatoria3 + (L(n,k)^2);
    end
    L(n,n) = A(n,n) - Sumatoria3;
    if L(n,n) < 0
        error('Argumento de raiz negativo')
    end
    L(n,n) = L(n,n)^(1/2);
end
```

```
function [ L , U ] = algoritmoThomas( A )
    %Retorna las matrices L y U correspondientes a la fact. de Thomas de
la
    %matriz tridiagonal A
    [n,m] = size(A);
    L = eye(n);
    U = zeros(n);

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

    U(n,n) = A(n,n);

    for i=2:n
        L(i,i-1) = A(i,i-1)/U(i-1,i-1);
        U(i,i) = A(i,i)-(L(i,i-1)*U(i-1,i));
    end
end


function [ x ] = algoritmoJacobi( A , b , x, Tol, itMax)
%Realiza iteraciones de la forma Jacobi sobre la matriz A y su vector
solucion b
%Recibe además los parámetros x: vector inicial, itMax: iteraciones
máximas permitidas
%y Tol: la tolerancia deseada.
    k = 1;
    r = b - A*x;
    Tol2 = Tol*norm(b);
    D = diag(A);

    while (k <= itMax) && (norm(r) > Tol2)
        z = r./D;
        x = x + z;
        r = b - A*x;
        k = k + 1;
        if(k == itMax)
            error('Nro max de iteraciones alcanzados')
        end
    end
    fprintf('Nro de iteraciones: %d', k);
end

function [ x ] = Gauss_Seidel_Param( A , b , x, Tol, itMax)
%Realiza iteraciones de la forma Gauss Seidel sobre la matriz A y su
vector solucion b
%Recibe además los parámetros x: vector inicial, itMax: iteraciones
máximas permitidas
%y Tol: la tolerancia deseada.
    k = 1;
    r = b - A*x;
    Tol2 = Tol*norm(b);
    M = tril(A,k);

    while (k <= itMax) && (norm(r) > Tol2)
        z = algoritmoDescenso(M, r);
        x = x + z;
        r = b - A*x;
        k = k + 1;
```

```
        if(k == itMax)
            error('Nro max de iteraciones alcanzados')
        end
    end
    fprintf('Nro de iteraciones: %d', k);
end



function [ x ] = algoritmoJacobiPorComponentes( A , b , x, Tol, itMax)
%Realiza iteraciones de la forma Jacobi por componentes sobre la matriz
tridiagonal A y su vector solucion b
%Recibe además los parámetros x: vector inicial, itMax: iteraciones
máximas permitidas
%y Tol: la tolerancia deseada.
    k = 1;
    [n,m] = size(A);
    %%r = b - A*x;

    r(1) = b(1) - A(1,1)*x(1) - A(1,2)*x(2);
    for i=2 : n-1
        r(i) = b(i) - A(i,i-1)*x(i-1) - A(i,i)*x(i) - A(i,i+1)*x(i+1);
    end
    r(n) = b(n) - A(n,n-1)*x(n-1) - A(n,n)*x(n);


    Tol2 = Tol*norm(b);
    D = diag(A);

    if (prod(D) == 0)
        error('Hay elemento/s en la diagonal nulo/s');
    end

    while (k <= itMax) && (norm(r) > Tol2)
% % %         z = r./D;
% % %         x = x + z;        %Form. eficiente


        x(1) = (b(1) - A(1,2)*x(1)) / A(1,1);
        for i=2 : n-1
            x(i) = (b(i) - A(i,i-1)*x(i) - A(i,i+1)*x(i)) / A(i,i);
        end
        x(n) = (b(n) - A(n,n-1)*x(n)) / A(n,n);


        r(1) = b(1) - A(1,1)*x(1) - A(1,2)*x(2);
        for i=2 : n-1
            r(i) = b(i) - A(i,i-1)*x(i-1) - A(i,i)*x(i) - A(i,i+1)*x(i+1);
        end
        r(n) = b(n) - A(n,n-1)*x(n-1) - A(n,n)*x(n);


        k = k + 1;
        if(k == itMax)
            error('Nro max de iteraciones alcanzados');
        end
    end
    fprintf('Nro de iteraciones: %d', k);
end
```

```
function [ x ] = algoritmoGaussSeidelPorComponentes( A , b , x, Tol,
itMax)
%Realiza iteraciones de la forma Gauss Seidel por componentes sobre la
matriz tridiagonal A y su vector solucion b
%Recibe además los parámetros x: vector inicial, itMax: iteraciones
máximas permitidas
%y Tol: la tolerancia deseada.
    k = 1;
    [n,m] = size(A);
    z = zeros(n,1);
    M = tril(A);

    r(1) = b(1) - A(1,1)*x(1) - A(1,2)*x(2);
    for i=2 : n-1
        r(i) = b(i) - A(i,i-1)*x(i-1) - A(i,i)*x(i) - A(i,i+1)*x(i+1);
    end
    r(n) = b(n) - A(n,n-1)*x(n-1) - A(n,n)*x(n);


    Tol2 = Tol*norm(b);

    while (k <= itMax) && (norm(r) > Tol2)
        %z = algoritmoDescenso(M, r);       %
        %x = x + z;                         %Form. eficiente
        %r = b - A*x;                       %

        z(1) = r(1) / A(1,1);
        for i=2:n
            z(i) = (r(i) - (A(i,i-1)*z(i-1))) / A(i,i);
        end

        x = x + z;

        r(1) = b(1) - A(1,1)*x(1) - A(1,2)*x(2);
        for i=2 : n-1
            r(i) = b(i) - A(i,i-1)*x(i-1) - A(i,i)*x(i) - A(i,i+1)*x(i+1);
        end
        r(n) = b(n) - A(n,n-1)*x(n-1) - A(n,n)*x(n);


        k = k + 1;
        if(k == itMax)
            error('Nro max de iteraciones alcanzados')
        end
    end
    fprintf('Nro de iteraciones: %d', k);
end
```

```
function [ lambda ] = potenciaIteradaTridiag( A, tol, Nmax )
%Retorna el mayor autovalor de la matriz A tridiagonal definida positiva
    [n,noSeUsa] = size(A);
    x = rand(n,1);
    k = 0;

    y = x/norm(x);
    x = A*y;
    lambda = transpose(y)*x;
    err = tol*abs(lambda) + 1;

    while (err > tol*abs(lambda)) && (abs(lambda) > 0) && (k <= Nmax)
        y = x/norm(x);

        %x = A*y

        x(1) = A(1,1)*y(1) + A(1,2)+y(2);
        for i=2:n-1
            x(i) = A(i,i-1)*y(i-1) + A(i,i)*y(i) + A(i,i+1)*y(i+1);
        end
        x(n) = A(n,n-1)*y(n-1) + A(n,n)*y(n);

        lambda2 = transpose(y)*x;
        err = abs(lambda2 - lambda);
        lambda = lambda2;
        k = k+1;
    end
    if k > Nmax
        error('Nro. de iteraciones máximo alcanzado')
    end
    fprintf('Nro. de iteraciones %d \n', k);
end
```

```
function [ lambda ] = potenciaInversaParaTridiag( A, tol, Nmax )
%Retorna el menor autovalor de la matriz A tridiagonal definida positiva
    [n,noSeUsa] = size(A);
    x = rand(n,1);
    k = 0;


    [L,U] = algoritmoThomas(A);
    y = x/norm(x);
    z = algoritmoDescenso(L,y);
    x = algoritmoRemonte(U,z);
    lambda = transpose(y)*x;
    err = tol*abs(lambda) + 1;

    while (err > tol*abs(lambda)) && (abs(lambda) > 0) && (k <= Nmax)
        y = x/norm(x);
        z = algoritmoDescenso(L,y);
        x = algoritmoRemonte(U,z);
        lambda2 = transpose(y)*x;
        err = abs(lambda2 - lambda);
        lambda = lambda2;
        k = k+1;
    end
    if k > Nmax
        error('Nro. de iteraciones máximo alcanzado')
    end
    fprintf('Nro. de iteraciones %d \n', k);
    lambda = 1/lambda;
end
```

```
function [ K2 ] = numeroDeCondicionDe( A, tol, Nmax )
%Retorna el numero de condición K2 de la matriz A definida positiva, para
la
%tolerancia tol y el numero de iteraciones maximo Nmax.
    mayorAv = potenciaIteradaTridiag(A, tol, Nmax);
    menorAv = potenciaInversaTridiag(A, tol, Nmax);

    K2 = mayorAv/menorAv;
End
```

-Scripts del TP: Obtención de los números de condición

```
% Obtencion de los numeros de condicion para las matrices del TP1

condicion = zeros(8,1);      %prealocación
tol = 1e-6;                  %%
Nmax = 10000;                %%parámetros de iteración

for i=1:8
    A = generadorMatrizTP1(i);
    condicion(i) = numeroDeCondicionDeTridiag(A, tol, Nmax);
    condicion(i)    %Respuesta "on the fly"
end
condición
```

-Scripts del TP: Obtención de los tiempos de ejecución de los métodos optimizados

```
% Inicializacion

    clear all;
    clc;

    n = [10 30 100 300 1000 3000 10000 20000];

    prealocadaTemp = zeros(n(i),1);
    tiempoJParaN_i = zeros(8,1);
    resultadoJParaN_i = zeros(8,1);     %preAlocaciones de memoria
    tiempoGSParaN_i = zeros(8,1);
    resultadoGSParaN_i = zeros(8,1);

    %Parámetros de los métodos

    itMax = 1e6;              %nro de iteraciones maximas
    Tol = 1e-6;               %tolerancia de corte

%Ejecuta los métodos

    for i=1:8

        [A1, b1] = generadorMatrizTP1(i);      %matriz nxn y b(n,1)
        x1 = zeros(n(i),1);                    %vector inicial

    %Resuelve por el método de Jacobi por componentes

        tic;
        prealocadaTemp = algoritmoJacobiPorComponentes(A1,b1,x1,Tol,
itMax);
        tiempoJParaN_i(i) = toc;

        resultadoJParaN_i(i)=((prealocadaTemp')*prealocadaTemp)/n(i);

        tiempoJParaN_i(i)
        resultadoJParaN_i(i)        %comprobaciones "on the fly"


    %Resuelve por el método de Gauss Seidel por componentes

        tic;
        prealocadaTemp = algoritmoGaussSeidelPorComponentes(A1,b1,x1,
Tol,itMax);
        tiempoGSParaN_i(i) = toc;

        resultadoGSParaN_i(i)=((prealocadaTemp')*prealocadaTemp)/n(i);

        tiempoGSParaN_i(i)
        resultadoGSParaN_i(i)       %comprobaciones "on the fly"

    end

%comprobacion final y tiempos

    tiempoJParaN_i
    resultadoJParaN_i
    tiempoGSParaN_i
    resultadoGSParaN_i
```

-Scripts del TP: Obtención de los tiempos de ejecución de los métodos numéricos para n(i)

```
%   TP 1 - 28/09/16

% Inicializacion

    clear all;
    clc;

    i=4;        %i entre 1 y 8 selecciona el valor de n con el que se va a
trabajar


    n = [10 30 100 300 1000 3000 10000 20000];
    [A1, b1] = generadorMatrizTP1(i);

    tiempoN1 = zeros(7,1);
    prealocadaTemp = zeros(n(i),1);     %prealocaciones de memoria
    resultadoN1 = zeros(7,1);

 %Modelo de toma de tiempo para Gauss

     A2 = A1;         %prealocación de memoria
     b2 = b1;         %prealocación de memoria

 %corre primero la eliminacion de gauss con pivote, y luego realiza el
 %algoritmo de remonte

     tic;
     [A2, b2] = elimGaussConPivote(A1, b1);
     prealocadaTemp = algoritmoRemonte(A2, b2);
     tiempoN1(1) = toc;

     resultadoN1(1) = ((prealocadaTemp')*prealocadaTemp)/n(i);


 %Modelo de toma de tiempo LU

     L = A1;          %prealocación de memoria
     U = A1;          %prealocación de memoria

 %Factoriza la matriz en L y U, y resuelve por descenso de L con b1, y
luego
 %por remonte de U con el resultado de la anterior operacion

     tic;
     [L,U]=algoritmoLU(A1);
     prealocadaTemp = algoritmoDescenso(L, b1);
     prealocadaTemp = algoritmoRemonte(U, prealocadaTemp);
     tiempoN1(2) = toc;

     resultadoN1(2) = ((prealocadaTemp')*prealocadaTemp)/n(i);


 %Modelo de toma de tiempo Cholesky
```

```
    L = A1;          %prealocación de memoria
    Lt = A1;         %prealocación de memoria

 %Factoriza la matriz por el metodo de Cholesky, y resuelve por descenso L
 %con b1, y por remonte L transpuesta con el resultado de la anterior
 %operacion

    tic;
    [L]=algoritmoCholesky(A1);
    prealocadaTemp = algoritmoDescenso(L, b1);
    Lt = L';
    prealocadaTemp = algoritmoRemonte(Lt, prealocadaTemp);
    tiempoN1(3) = toc;

    resultadoN1(3) = ((prealocadaTemp')*prealocadaTemp)/n(i);


 %Modelo de toma de tiempo Thomas

    L = A1;          %prealocación de memoria
    U = A1;          %prealocación de memoria

 %Factoriza la matriz en L y U por el metodo de Thomas, y resuelve por
descenso de L con b1, y luego
 %por remonte de U con el resultado de la anterior operacion

    tic;
    [L,U]=algoritmoThomas(A1);
    prealocadaTemp = algoritmoDescenso(L, b1);
    prealocadaTemp = algoritmoRemonte(U, prealocadaTemp);
    tiempoN1(4) = toc;

    resultadoN1(4) = ((prealocadaTemp')*prealocadaTemp)/n(i);


%Modelo de toma de tiempo jacobi
%Parámetros

    itMax = 1e6;              %nro de iteraciones maximas
    Tol = 1e-6;               %tolerancia de corte
    x1 = zeros(n(i),1);       %vector inicial

%Resuelve por el método de Jacobi

    tic;
    prealocadaTemp = algoritmoJacobi( A1 , b1 ,x1, Tol, itMax);
    tiempoN1(5) = toc;

    resultadoN1(5) = ((prealocadaTemp')*prealocadaTemp)/n(i);


%Modelo de toma de tiempo Gauss Seidel
%Parámetros

    itMax = 1e6;              %nro de iteraciones maximas
    Tol = 1e-6;               %tolerancia de corte
    x1 = zeros(n(i),1);       %vector inicial

%Resuelve por el método de Gauss Seidel

    tic;
```

```
    prealocadaTemp = Gauss_Seidel_Param( A1 , b1 , x1, Tol, itMax);
    tiempoN1(6) = toc;

    resultadoN1(6) = ((prealocadaTemp')*prealocadaTemp)/n(i);


 %Modelo de toma de tiempo MatLab
 %Resuelve por el método de MatLab

     tic;
     prealocadaTemp = A1\b1;
     tiempoN1(7) = toc;

     resultadoN1(7) = ((prealocadaTemp')*prealocadaTemp)/n(i);


 %Impresión por pantalla - chequeo de validez

    tiempoN1
    resultadoN1
    fprintf('lo anterior corresponde a i=%d', i);
```

-Scripts del TP: scripts de los gráficos

Gráfico de los siete métodos

```
%Script del grafico de la comparación de los 7 metodos

%Vector de n

X=[10 30 100 300 1000 3000 10000];

%Vectores de tiempos segun el metodo numerico empleado para los n

Y1=[0.0091 0.0118 0.0425 0.9047 36.0535 979.7783 1.0e+04 * 9.2381]; %G
Y2=[0.0088 0.0106 0.0418 0.8968 35.7939 971.7350 1.0e+04 * 9.1953]; %LU
Y3=[0.0055 0.0059 0.0119 0.1275 5.7728 184.9242 1.0e+04 * 2.0593];  %Ch
Y4=[0.0071 0.0072 0.0076 0.0108 0.0575 0.4778 90.9256];             %Th
Y5=[0.0065 0.0070 0.0094 0.0329 2.2780 44.2363 1.0e+03 * 1.0242];   %J
Y6=[0.0058 0.0056 0.0067 0.0217 0.2455 2.3139 57.1905];             %GS
Y7=[0.0001 0.0001 0.0003 0.0016 0.0496 0.7399 88.9729];             %\
loglog(X,Y1,'bo--',X,Y2,'go--',X,Y3,'co--',X,Y4,'ro--',X,Y5,'mo--
',X,Y6,'ko--',X,Y7,'yo--', 'LineWidth',1.5,'Marker','+')
title('Tiempo de Resolucion vs Tama  e Matriz')
xlabel('n')
ylabel('t')
```

Gráfico de métodos eficientes versus por componentes

```
%Script del grafico de comparacion de los metodos eficientes vs
optimizados

%vector de ns
X1=[10 30 100 300 1000 3000 10000];

%matriz de tiempos obtenidos para los m鴫dos de Jacobi y
%Gauss Seidel en sus formas eficientes y por componentes

Y1=[0.0156 0.0062 0.0422 0.3111 3.1743 27.5255 605.7931];           %J-C
Y2=[0.0112 5.2129e-04 7.8805e-04 0.0024 0.0113 0.0591 0.5418];      %GS-C
Y3=[0.0065 0.0070 0.0094 0.0329 2.2780 44.2363 1.0e+03 * 1.0242];   %J-E
Y4=[0.0058 0.0056 0.0067 0.0217 0.2455 2.3139 57.1905];             %GS-E

Y = [Y1 ; Y2 ; Y3 ; Y4];

%CREATEFIGURE(X1, Y)
%  Auto-generated by MATLAB on 05-Oct-2016 15:22:22

% Create figure
figure1 = figure('Color',...
    [0.831372559070587 0.815686285495758 0.7843137383461]);

% Create axes
axes1 = axes('Parent',figure1,'YScale','log','YMinorTick','on',...
    'XScale','log',...
    'XMinorTick','on',...
    'Color',[0.941176474094391 0.941176474094391 0.941176474094391]);
box(axes1,'on');
hold(axes1,'all');

% Create multiple lines using matrix input to loglog
loglog1 = loglog(X1,Y,'Parent',axes1);
set(loglog1(1),'Marker','^','LineStyle','--','Color',[0 0 1],...
    'DisplayName','Jacobi por componentes');
set(loglog1(2),'Marker','diamond','LineStyle','--','Color',[1 0 0],...
    'DisplayName','Gauss-Seidel por componentes');
set(loglog1(3),'Marker','^','Color',[1 0 1],...
    'DisplayName','Jacobi eficiente');
set(loglog1(4),'Marker','diamond','DisplayName','Gauss-Seidel
eficiente',...
    'Color',[0 0 0]);

% Create title
title('Tiempo de Resolucion vs Tama  e Matriz','FontWeight','bold',...
    'FontSize',11);

% Create xlabel
xlabel('n','FontWeight','bold','FontSize',11);

% Create ylabel
ylabel('t','FontWeight','bold','FontSize',11);

% Create legend
legend1 = legend(axes1,'show');
set(legend1,...
    'Position',[0.180467091295117 0.633511938823285 0.382165605095541
0.193969030154849],...
    'Color',[1 0.949019610881805 0.866666674613953]);
```

Gráfico de los números de condición

```
%Script del grafico de los numeros de condicion segun n

nrosCond=1.0e+03*[ 0.0092 0.0200 0.0469 0.1013 0.2326 0.4930 1.1155
1.7811];
n=[10 30 100 300 1000 3000 10000 20000];
loglog(n, nrosCond);
title('K2 vs Tama□e  la Matriz');
xlabel('n');
ylabel('K2');
```