```
% IMPLEMENTACIÓN SISTEMA LINEAL
%-------------------------------------
%  Configuración del problema (1)
%-------------------------------------
L    =  1;       % Longitud del intervalo
K    =  1;       % constante elástica (N/m^2)
Area =  1;       % sección transversal de la cuerda (m^2)
f0   =  1;       % Valor constante de la
% fuerza por unidad de long (N/m)
%-------------------------------------
% Solución analítica del problema
%-------------------------------------
xexact  =  linspace(0,L,100);
uexact  =  f0*xexact.*(L-xexact)/(2*K*Area);
plot(xexact,uexact,'LineWidth',4)
hold on
%-------------------------------------
% Cálculo de la solución para distintos
% tamaños del problema
%-------------------------------------
N     =   [2;5;10];  % Vector de tamaños del problema
for i=1:length(N)
    %-------------------------------------
    %  Configuración del problema (2)
    %-------------------------------------
    n    =  N(i);       % número de nodos intermedios
    h    =  1/(n+1);    % espaciado entre nodos
    x    =  linspace(0,L,n+2);
    %-------------------------------------
    % Creamos la matriz del sistema así
    % como el término independiente (b)
    %-------------------------------------
    A    =  K*Area/h^2*MyToeplitzSparse(n);
    b    =  f0*ones(n,1);
    %-------------------------------------
    % Solver directo
    %-------------------------------------
    udirect  =  A\b;
    udirect  =  [0;udirect;0];
    plot(x,udirect,'LineWidth',4);
end
grid on
legend('Analítica','n=2','n=5','n=10')
xlabel('x')
ylabel('u')
```

```
% COSTE COMPUTACIONAL SOLVER DIRECTO VS INVERSA
%-------------------------------------
%  Configuración del problema (1)
%-------------------------------------
L    =  1;       % Longitud del intervalo
K    =  1;       % constante elástica (N/m^2)
Area =  1;       % sección transversal de la cuerda (m^2)
f0   =  1;       % Valor constante de la
% fuerza por unidad de long (N/m)
%-------------------------------------
% Cálculo de la solución para distintos
% tamaños del problema
%-------------------------------------
N     =   [5;50;500;5000];  % Vector de tamaños del problema
for i=1:length(N)
    %-------------------------------------
    %  Configuración del problema (2)
    %-------------------------------------
    n    =  N(i);       % número de nodos intermedios
    h    =  1/(n+1);    % espaciado entre nodos
    x    =  linspace(0,L,n+2);
    %-------------------------------------
    % Creamos la matriz del sistema así
    % como el término independiente (b)
    %-------------------------------------
    A    =  K*Area/h^2*MyToeplitzSparse(n);
    b    =  f0*ones(n,1);
    fprintf('Tamaño %d\n',n)
    %-------------------------------------
    % Solver directo
    %-------------------------------------
    tic;
    udirect  =  A\b;
    time=toc;
    fprintf('Time for direct solver %f\n',time)
    %-------------------------------------
    % Inversa
    %-------------------------------------
    tic;
    uinverse  =  inv(A)*b;
    time=toc;
    fprintf('Time for inverse %f\n',time)
end
```

```
% FACTORIZACIÓN LU
function [L,U]  =  MyLUFactorisation(A)
%---------------------------------
% Check dimensions of A
%---------------------------------
n  =  size(A,1); % número de filas de A
m  =  size(A,2); % número de columnas de A
if abs(n-m)>0
   error('Chequee las dimensiones de A. La matriz debe ser cuadrada')
end
%-------------------------------------------------------------------------
% Inicializamos las matrices L y U con ceros. Trabajamos en formato full
% Implementación de la factorización en format sparse es difícil
%-------------------------------------------------------------------------
L                      =  zeros(n);
U                      =  zeros(n);
for j=1:n  %  bucle sobre el número de columnas
%  Introducimos la diagonal de L: L(j,j)=1
L(j,j)             =  1;
for i=1:n  %  bucle sobre el número de filas
    %-----------------------------------
    % Situation where i>j
    %-----------------------------------
    if i>j
       L(i,j)      =  A(i,j);
       for k=1:j-1
           L(i,j)  =  L(i,j) - L(i,k)*U(k,j);
       end
       L(i,j)      =  L(i,j)/U(j,j);
    end
    %-----------------------------------
    % Situation where i<=j
    %-----------------------------------
    if i<=j
       U(i,j)      =  A(i,j);
       for k=1:i-1
           U(i,j)  =  U(i,j) - L(i,k)*U(k,j);
       end
   end
end
```

```
% RESOLUCIÓN SISTEMA TRIANGULAR INFERIOR
function x  =  MyLowerSubstitution(A,b)
%--------------------------------
% Check dimensions of A
%--------------------------------
n  =  size(A,1);
m  =  size(A,2);
if abs(n-m)>0
    error('The matrix A must be squared')
end
x    =  zeros(n,1); % inicializamos el vector incógnita
for i=1:n  %  recorremos todas las componentes del vector x
    x(i)  =  b(i);
    for j=1:i-1
        x(i)  =  x(i) - A(i,j)*x(j);
    end
    x(i)    =  x(i)/A(i,i);
end
```

```
% RESOLUCIÓN SISTEMA TRIANGULAR SUPERIOR
function x  =  MyUpperSubstitution(A,b)
%--------------------------------
% Check dimensions of A
%--------------------------------
n  =  size(A,1);
m  =  size(A,2);
if abs(n-m)>0
    error('The matrix A must be squared')
end
x    =  zeros(n,1);
for i=1:n
    alpha  =  n-(i-1);
    x(alpha)  =  b(alpha);
    for j=1:i-1
        beta      =  n - (j-1);
        x(alpha)  =  x(alpha) - A(alpha,beta)*x(beta);
    end
    x(alpha)    =  x(alpha)/A(alpha,alpha);
end
```

```
% SOLVER LU
function x  =  MyLUSolver(A,b)
%----------------------------------------
% Factorización LU (1er paso)
%----------------------------------------
[L,U]         =  MyLUFactorisation(A);
%----------------------------------------
% resolvemos L*z=b (2nd paso)
%----------------------------------------
z             =  MyLowerSubstitution(L,b);
%----------------------------------------
% resolvemos U*x=z (3rd paso)
%----------------------------------------
x             =  MyUpperSubstitution(U,z);
```

```
% FACTORIZACIÓN DE CHOLESKY
function L     =  MyCholeskyFactorisation(A)
%---------------------------------
% Check dimensions of A
%---------------------------------
n  =  size(A,1); % número de filas de A
m  =  size(A,2); % número de columnas de A
if abs(n-m)>0
    error('Chequee las dimensiones de A. La matriz debe ser cuadrada')
end
%-------------------------------------
% Inicializamos las matriz L con ceros.
% Trabajamos en formato full
% Implementación de la factorización
% en format sparse es muy difícil
%-------------------------------------
L                     =  zeros(n);
for j=1:n  %  bucle sobre el número de columnas
    %Introducimos la diagonal de L: L(j,j)=1
    L(j,j)            =  A(j,j);
    for k=1:j-1
        L(j,j)        =  L(j,j) - L(j,k)^2;
    end
    L(j,j)            =  sqrt(L(j,j));
    for i=1:n  %  bucle sobre el número de filas
        if abs(i-j)>0
           L(i,j)          =  A(i,j);
           for k=1:j-1
               L(i,j)      =  L(i,j) - L(i,k)*L(j,k);
           end
           L(i,j)          =  L(i,j)/L(j,j);
        end
    end
end
```

```
% SOLVER CHOLESKY
function x  =  MyCholeskySolver(K,f)
%---------------------------------------
% Factorización LU (1er paso)
%---------------------------------------
L             =  MyCholeskyFactorisation(K);
%---------------------------------------
% resolvemos L*y=f (2nd paso)
%---------------------------------------
y             =  MyLowerSubstitution(L,f);
%---------------------------------------
% resolvemos U*x=y (3rd paso)
%---------------------------------------
x             =  MyUpperSubstitution(transpose(L),y);
end
```

```
% NÚMERO DE CONDICIONAMIENTO Y PRECISIÓN DE LA SOLUCIÓN
clc;
clear all;
L  =  1;   % Longitud de la barra
N          =  [1e1;1e2;1e3;1e4;1e5;1e6];  %  Number of intermediate points
%  excluding boundary
rel_error  =  zeros(size(N,1),1);  % Vector de errores relativos
F          =  -2;
KA         =  1;
for i=1:size(N,1)
    h  =  L/(N(i)+1);   % Spaciado entre nodos
    %----------------------------------------
    % Creamos el vector independiente (b)
    % y la matriz del sistema (K)
    %----------------------------------------
    b  =  F*ones(N(i),1);
    K  =  KA/h^2*MyToeplitzSparse(N(i));
    %----------------------------------------
    % Calculamos solución numérica con solver
    %  directo de Octave
    %----------------------------------------
    unumerical  =  K\b;
    %----------------------------------------
    % Comparamos con solución analítica
    %----------------------------------------
    x           =  (linspace(0,L,N(i)+2))';
    x(1)        =  [];
    x(end)      =  [];
    uexact      =  F*x.*(L-x)/(2*KA);
    rel_error(i)   =  norm(uexact - unumerical)/norm(uexact);
end
```

```
% MÉTODO ITERATIVO DE JACOBI
function [x,iter]  =  MyJacobiIterativeSolver(K,f,tol,maxiter,x0)
%-----------------------------------
% Extraemos la diagonal de K como vector
% e invertimos cada componente del vector
%-----------------------------------
v       =  diag(K);  %  Extraemos vector con la diagonal
D       =  diag(v);  %  Crear la matriz diagonal
d       =  1./v;     %  invertimos los elementos del vector
iter    =  0;        %  iteraciones inicializadas a 0
xk      =  x0;       %  Solución xk vamos a hacerla coincidir con
                     %  solución inicial
diff    =  tol*6;
while and(diff>tol,iter<=maxiter)
    iter  =  iter + 1;  % incrementamos las iteraciones
    %---------------------------------
    % Calculamos (D-K)*xold + b
    %---------------------------------
    ftilde    =  (D-K)*xk + f;
    %---------------------------------
    % Resolvemos
    %---------------------------------
    xk_1     =  d.*ftilde;  %  El vector que contiene la inversa de la diagonal
    % multiplica a ftilde y nos da el nuevo valor de xk+1
    %---------------------------------
    % Criterio de tolerancia
    %---------------------------------
    if iter>1
        diff  =  norm(xk_1-xk)/norm(xk);
    end
    xk       =  xk_1;   %  xk coincide con xk+1
end
x   =  xk_1; % la salida de la función x la hacemos coincidir
% con xk_1
if iter>=maxiter
    warning('Jacobi solver used the maximum number of
    prescribed iterations without reaching the desired tolerance')
end
```

```
% MÉTODO ITERATIVO DE GAUSS-SEIDEL
function [x,iter]  =  MyGaussSeidelIterativeSolver(K,f,tol,maxiter,x0)
%-----------------------------------
% Extraemos la a partir de K su triangular
%  inferior. Ese es el precondicionador
%-----------------------------------
P       =  tril(K);
iter    =  0;
diff    =  tol*6;
x       =  x0;
while and(diff>tol,iter<=maxiter)
    iter  =  iter + 1;
    xold  =  x;
    %--------------------------------
    % Calculamos (P-K)*xold + b
    %--------------------------------
    ftilde    =  (P-K)*xold + f;
    %--------------------------------
    % Resolvemos
    %--------------------------------
    x     =  MyLowerSubstitution(P,ftilde);
    %--------------------------------
    % Criterio de tolerancia
    %--------------------------------
    if iter>1
        diff  =  norm(x-xold)/norm(xold);
    end
end
if iter>=maxiter
    warning('Gauss-Seidel solver used the maximum number
    of prescribed iterations without reaching the desired tolerance')
end
```

```
% INFLUENCIA NÚMERO DE CONDICIONAMIENTO/CONVERGENCIA MÉTODO ITERATIVO
sizes  =  [5;10;50;100;200];

for isize=1:length(sizes)
    n  =  sizes(isize);        %  número de nodos interiores
    h  =  1/(n+1);  %  Divisiones entre nodos
    K  =  (1/h^2)*MyToeplitz(n); %  Matriz del problema
    f  =  -2*ones(n,1);  %  vector de fuerzas
    [~,niterJacobi(isize)]=MyJacobiIterativeSolver(K,f,1e-5,1e5,zeros(n,1));
    v     =  diag(K);  %  Extraemos vector con la diagonal
    D     =  diag(v);  %  Crear la matriz diagonal
    d     =  1./v;
    % Invertimos vector v
    invD  =  diag(d);
    %  Creamos inversa diagonal
    M     =  eye(n) - invD*K;
    % Creamos matriz M
    rhoJacobi(isize)   =  max(eig(M));  % radio espectral matriz M
    conKJacobi(isize)  =  cond(K);  % número de condicionamiento de K
end
```

```
% CÁLCULO AUTOVALORES/AUTOVECTORES FACTORIZACIÓN QR
function [V,Lambda]  =  MyQRIterationAlgorithm(K)

condition  =  1;
Iter       =  0;
tol        =  1e-5;
maxIter    =  1e3;  % Número máximo de iteraciones
Qtot       =  eye(size(K));
while condition==1
   %-----------------------------------
   % Algoritmo
   %-----------------------------------
   Iter   =  Iter + 1;
   [Q,R]  =  MyQRFactorisation(K);
   K      =  R*Q;
   Qtot   =  Qtot*Q;  %  Qtot=Q0*Q1*Q2*...*Qn-1
   %-----------------------------------
   % Criterio de parada
   %-----------------------------------
   I1     =  sum(abs(K(:)));  %  sumamos el valor absoluto de
   %todos los elementos de K
   I2     =  sum(abs(diag(K))); % sumamos el valor absoluto de
   %  todos los elementos de la diagonal de K
   diff   =  abs(I1 - I2)/I1;
   if diff<tol
      condition  =  0;
   end
   if Iter>maxIter
      break;
   end
end
%-----------------------------------------------------
% Obtengo los autovectores
%-----------------------------------------------------
VV    =   Qtot;
for i=1:size(K,1)
    VV(:,i)  =  VV(:,i)/norm(VV(:,i));
end
%-----------------------------------------------------
% Ordeno los autovalores y autovectores
%-----------------------------------------------------
[Lambda,Index]   =  sort(diag(K));
V                =  VV(:,Index);
end
```

```
% FACTORIZACIÓN QR
function [q,r]  =  MyQRFactorisation(A)

n  =  size(A,1);
q  =  zeros(n);
q(:,1)  =  A(:,1)/norm(A(:,1));

for i=2:n
   %--------------------------------
   % Inicializamos matriz B de tamaño (n-1)x(n-1)
   % y vector f de tamaño (n-1)x1
   %--------------------------------
   B  =  zeros(i-1);
   f  =  zeros(i-1,1);
   %--------------------------------
   % Creamos matriz B y vector f
   %--------------------------------
   for k=1:i-1
       for j=1:i-1
           B(j,k)  =  q(:,j)'*q(:,k);
       end
       f(k)  =  -A(:,i)'*q(:,k);
   end
   %--------------------------------
   % Resolvemos alpha
   %--------------------------------
   alpha     =  B\f;
   %--------------------------------
   % Obtenemos fila i de matriz q
   %--------------------------------
   q(:,i)    =  A(:,i);
   for j=1:i-1
       q(:,i)  =  q(:,i) + alpha(j)*q(:,j);
   end
   q(:,i)      =  q(:,i)/norm(q(:,i));
end
%--------------------------------
% Obtenemos matriz r
%--------------------------------
r    =  q'*A;
```