```
% ECUACIONES NORMALES

% Minimos cuadrados a los datos transformados de g1 y g2. PMCL resolviendo directamente las ecs normales.
X = [0.5,0.6,0.7,0.8,0.9,1.0,1.1,1.2,1.3,1.4,1.5,1.6,1.7,1.8,1.9,2.0]';
%Y1
Y1 = [3.89,2.75,2.01,1.61,1.21,0.89,0.69,0.63,0.44,0.42,0.70,0.32,0.40,0.26,0.32,0.25]';

%Y2
%Y1 = [15.96,9.45,5.75,3.82,2.89,2.17,1.22,1.05,0.86,0.63,0.69,0.40,0.44,0.29,0.43,0.20]';

%   n cantidad de muestras
n = length(X);

A = zeros(n,2);
Xtransf = log(X);
Ytransf = log(Y1);
C=zeros(2,1);
A = [Xtransf ones(n,1)];

C=(A' * A) \ (A' * Ytransf);

p= - round(C(1))
c= exp(C(2))

% Valor minimizado: ||AX - Y||^2
error_g = sum((c*X.^(-p) - Y1).^2);

%----------------------------------------
% Graficas para correccion del modelo: %
%----------------------------------------
% Funcion g1
% Grafica modelo lineal con transformacion logaritmica de los datos
figure (1)
plot(Xtransf, Ytransf, 'bs')
hold on
vect_Xcv = linspace(Xtransf(1),Xtransf(n),50);
plot(vect_Xcv, C(1)*vect_Xcv + C(2), 'r')
title ("Funcion g - lineal");
xlabel ("x");
ylabel ("y");
legend("datos transformados", "Sol Ecs Normales");

% Grafica sin transformacion logaritmica de los datos
figure (2)
plot(X, Y1, 'bs')
hold on
vect_X = linspace(X(1),X(n),50);
plot(vect_X, c*vect_X.^(-p), 'r')
title ("Funcion g - original");
xlabel ("x");
ylabel ("y");
```

```
legend("datos originales", "Sol Ecs Normales");
%----------------------------

%DESCOMPOSICION QR

% Minimos cuadrados a los datos transformados de g1 y g2. PMCL usando descomposicion QR
clear all;
X = [0.5,0.6,0.7,0.8,0.9,1.0,1.1,1.2,1.3,1.4,1.5,1.6,1.7,1.8,1.9,2.0]';
% Funcion g1
Y1 = [3.89,2.75,2.01,1.61,1.21,0.89,0.69,0.63,0.44,0.42,0.70,0.32,0.40,0.26,0.32,0.25]';
% Funcion g2
Y2 = [15.96,9.45,5.75,3.82,2.89,2.17,1.22,1.05,0.86,0.63,0.69,0.40,0.44,0.29,0.43,0.20]';

m = length(X);
n = 2; % modelo lineal

% Modelo de funcion g
% g = @(ci,p) @(x) ci/x^p;

X_cv = log(X);
Y1_cv = log(Y1);
Y2_cv = log(Y2);

function C = PMCLQR(A,Y)
  % Resuelve el PMCL  min(|| AC - Y ||^2) usando descomp QR
  % Matriz A m x n ; vector Y m x 1
  % Calculo matriz Q reducida (1eras n filas de Q)
  % Q = [Q_r Q2]
  [m n] = size(A);
  Q_r = zeros(m,n);
  Q_r(:,1) = A(:,1)/norm(A(:,1));
  for i = 2:n;
    aux = A(:,i);
    for j = 1:(i-1);
      aux = aux - dot(A(:,i),Q_r(:,j))*Q_r(:,j);
    end
  Q_r(:,i) = aux/norm(aux);
  end
  % Calculo matriz R reducida (1eras n filas de R)
  % R = [   R_r  ]
  %     [0 0 .. 0]
  R_r = Q_r'*A;
  % Los elementos fuera del triangulo quedan en x10^-17 etc, entonces los hago cero.
  for i = 1:n;
    for j = 1:(i-1);
    R_r(i,j) = 0;
    end
  end
  % Resuelvo el sistema por sustitucion hacia atras
  ec_opt.UT = true;
  C = linsolve(R_r,Q_r'*Y,ec_opt);
end % PMCLQR
```

```
A = [X_cv ones(m,1)];

%--------------
% Funcion g1: %
%--------------
% resuelvo el sist lineal R1X = (Q'Y)_1 con R1 matriz triangular superior, resolver con sustitucion hacia atras.
Param_1 = PMCLQR(A,Y1_cv);
% solucion es y = X(1).t + X(2)
% y = c.x^(-p) ; log(y) = -p.log(x) + log(c) ; y_cv = A.t + B
p = round(-Param_1(1));
c = exp(Param_1(2));

% Valor minimizado: ||AX - Y||^2
error_g1 = sum((c*X.^(-p) - Y1).^2);

%--------------
% Funcion g2: %
%--------------
% resuelvo el sist lineal R1X = (Q'Y)_1 con R1 matriz triangular superior, resolver con sustitucion hacia atras.
Param_2 = PMCLQR(A,Y2_cv);
% solucion es y = X(1).t + X(2)
% y = d.x^(-q) ; log(y) = -q.log(x) + log(d) ; y_cv = A.t + B
q = round(-Param_2(1));
d = exp(Param_2(2));

% Valor minimizado: ||AX - Y||^2
error_g2 = sum((d*X.^(-q) - Y2).^2);

%--------------------------------------
% Graficas para correccion del modelo: %
%--------------------------------------
% Funcion g1
% Grafica modelo lineal con transformacion logaritmica de los datos
figure (1)
plot(X_cv, Y1_cv, 'bs')
hold on
vect_Xcv = linspace(X_cv(1),X_cv(m),50);
plot(vect_Xcv, Param_1(1)*vect_Xcv + Param_1(2), 'r')
title ("Funcion g1 - lineal");
xlabel ("x");
ylabel ("y");
legend("datos transformados", "ajuste QR");

% Grafica sin transformacion logaritmica de los datos
figure (2)
plot(X, Y1, 'bs')
hold on
vect_X = linspace(X(1),X(m),50);
plot(vect_X, c*vect_X.^(-p), 'r')
```

```
title ("Funcion g1 - original");
xlabel ("x");
ylabel ("y");
legend("datos originales", "ajuste QR");
%----------------------------
% Funcion g2
% Grafica modelo lineal con transformacion logaritmica de los datos
figure (3)
plot(X_cv, Y2_cv, 'bs')
hold on
% vect_Xcv = linspace(X_cv(1),X_cv(m),50);
plot(vect_Xcv, Param_2(1)*vect_Xcv + Param_2(2), 'r')
title ("Funcion g2 - lineal");
xlabel ("x");
ylabel ("y");
legend("datos transformados", "ajuste QR");

% Grafica sin transformacion logaritmica de los datos
figure (4)
plot(X, Y2, 'bs')
hold on
% vect_X = linspace(X(1),X(m),50);
plot(vect_X, d*vect_X.^(-q), 'r')
title ("Funcion g2 - original");
xlabel ("x");
ylabel ("y");
legend("datos originales", "ajuste QR");

% dado x, y = 3*x.^2 + 2*x + 1; => polyfit(x,y,2) = [3 2 1]



%GAUSSNEWTON

X = [0.5,0.6,0.7,0.8,0.9,1.0,1.1,1.2,1.3,1.4,1.5,1.6,1.7,1.8,1.9,2.0]';
Y1 = [3.89,2.75,2.01,1.61,1.21,0.89,0.69,0.63,0.44,0.42,0.70,0.32,0.40,0.26,0.32,0.25]';
Y2 = [15.96,9.45,5.75,3.82,2.89,2.17,1.22,1.05,0.86,0.63,0.69,0.40,0.44,0.29,0.43,0.20]';

%   n cantidad de muestras
n = length(X);

function C = PMCLQR(A,Y)
  % Resuelve el PMCL  min(|| AC - Y ||^2) usando descomp QR
  % Matriz A m x n ; vector Y m x 1
  % Calculo matriz Q reducida (1eras n filas de Q)
  % Q = [Q_r Q2]
  [m n] = size(A);
  Q_r = zeros(m,n);
  Q_r(:,1) = A(:,1)/norm(A(:,1));
  for i = 2:n;
    aux = A(:,i);
    for j = 1:(i-1);
```

```
      aux = aux - dot(A(:,i),Q_r(:,j))*Q_r(:,j);
    end
  Q_r(:,i) = aux/norm(aux);
  end
  % Calculo matriz R reducida (1eras n filas de R)
  % R = [   R_r  ]
  %     [0 0 .. 0]
  R_r = Q_r'*A;
  % Los elementos fuera del triangulo quedan en x10^-17 etc, entonces los hago cero.
  for i = 1:n;
    for j = 1:(i-1);
    R_r(i,j) = 0;
    end
  end
  % Resuelvo el sistema por sustitucion hacia atras
  ec_opt.UT = true;
  C = linsolve(R_r,Q_r'*Y,ec_opt);
end % PMCLQR

function PCk = PMCNL(PCinit,X,Y,iter=10)
  % Calcula la solucion al PMCNL con datos X e Y usando Gauss-Newton
  fun = @(PC) PC(2)*X.^(-PC(1)); % funcion f
  diffun = @(PC) [(-PC(2)*X.^(-PC(1)).*log(X)) (X.^(-PC(1)))]; % jacobiano de f
  PCk = PCinit;
  for i = 1:iter
    Ak = diffun(PCk);
    Yk = Y - fun(PCk);
    % PCk = PCk + ols(Yk,Ak);
    PCk = PCk + PMCLQR(Ak,Yk);
  endfor
endfunction

% Calculo del error minimizado:
PC = PMCNL([1;1],X,Y1,10);
p = round(PC(1));
c = PC(2);
error_g1 = sum((c*X.^(-p) - Y1).^2);

QD = PMCNL([1;1],X,Y2,10);
q = round(QD(1));
d = QD(2);
error_g2 = sum((d*X.^(-q) - Y2).^2);

%--------------------------------------
% Graficas para correccion del modelo: %
%--------------------------------------
% Funcion g1
figure (2)
plot(X, Y1, 'bs')
hold on
vect_X = linspace(X(1),X(n),50);
plot(vect_X, c*vect_X.^(-p), 'r')
```

```
title ("Funcion g1");
xlabel ("x");
ylabel ("y");
legend("datos originales", "Sol Gauss-Newton");
%----------------------------
% Funcion g2
figure (4)
plot(X, Y2, 'bs')
hold on
% vect_X = linspace(X(1),X(n),50);
plot(vect_X, d*vect_X.^(-q), 'r')
title ("Funcion g2");
xlabel ("x");
ylabel ("y");
legend("datos originales", "Sol Gauss-Newton");



%ECUACIONES DIFERENCIALES

% Funcion g1(x) = c/x^p
c = 0.96750; % con outlier
p = 2; % con outlier
% [c p] = [2 0.94613] sin outlier

% Funcion g2(x) = d/x^q
d = 1.9949;
q = 3;

% EDO:
% y' = g1(x)*y = g2(x)
% y(0.5) = 0
% x in [0.5 .. 2]

a = 0.5;
b = 2;
y0 = 0;
% paso h
h = 0.01;
N = ceil((b-a)/h);
% Recalculo h
h = (b-a)/N;

x_exacta = linspace(a,b,100)';
% sol analitica con p = 2 y q = 3:
k = (-d/c)*(1/c + 2)*exp(-2*c); % obtenido con la cond inicial.
y_exacta = d/c^2 + d./(c*x_exacta) + k*exp(c./x_exacta);

% Euler hacia atras:
x = linspace(a,b,N)';
y_EA = zeros(length(x),1);
y_EA(1) = y0;
```

```
for i = 2:length(x);
   y_EA(i) = (y_EA(i-1) + d*h/x(i)^3)/(1 + c*h/x(i)^2);
   %y_EA(i) = ( x(i)^2*y_EA(i-1) + d*h/x(i) )/(x(i)^2 + c*h); % otra forma
end

% Euler hacia adelante:
% usa el mismo x que euler hacia atras
y_E = zeros(length(x),1);
y_E(1) = y0;
for i = 2:length(x);
   y_E(i) = y_E(i-1) + h*(-c*y_E(i-1)/x(i-1)^2 + d/x(i-1)^3);
end

% Runge-Kuta con ode solver ode45()
function ans = ecdif(x,y)
   c = 0.96750;
   d = 1.9949;
   ans = -c*y/x^2 + d/x^3;
end

[x_RK, y_RK] = ode45(@ecdif, [a b], y0);

% Graficas comparativas de Euler hacia atras, Euler hacia adelante, R-K y sol analitica:
figure(1)
%Sol analitica
plot(x_exacta, y_exacta, "b")
hold on
% Euler hacia adelante
plot(x, y_E, "r")
% Euler hacia atras
plot(x, y_EA, "m")
% Runge-Kuta
plot(x_RK, y_RK, "k")

axis ([a b])
title ("Sol de la EDO");
xlabel ("x");
ylabel ("y");
legend_text = legend ("Sol Analitica", "Euler hacia adelante", "Euler hacia atras", "Runge-Kuta");
legend (legend_text, "location", "southeast");
hold off
% ---------------------------------------------------------------------------------

% Evolucion de los errores de euler %
figure(2)
error_E = (d/c^2 + d./(c*x) + k*exp(c./x) - y_E).^2;
error_EA = (d/c^2 + d./(c*x) + k*exp(c./x) - y_EA).^2;
% Error de Euler hacia adelante
plot(x,error_E,"r")
hold on
% Error de Euler hacia atras
plot(x,error_EA,"b")
```

```
%title ("Evolucion del error");
xlabel ("x");
ylabel ("Error^2");
legend_text = legend ("Euler hacia adelante", "Euler hacia atras");
legend (legend_text, "location", "southeast");
hold off


% INTERPOLACION

% Funcion g1(x) = c/x^p
c = 0.96750; % con outlier
p = 2; % con outlier
% [c p] = [2 0.94613] sin outlier

% Funcion g2(x) = d/x^q
d = 1.9949;
q = 3;

a = 0.5;
b = 2;
y0 = 0;

x_exacta = linspace(a,b,100);
% sol analitica con p = 2 y q = 3:
k = (-d/c)*(1/c + 2)*exp(-2*c); % obtenido con la cond inicial.
y_exacta = d/c^2 + d./(c*x_exacta) + k*exp(c./x_exacta);

% Runge-Kuta con ode solver ode45()
function ans = ecdif(x,y)
   c = 0.96750;
   d = 1.9949;
   ans = -c*y/x^2 + d/x^3;
end
[x_RK, y_RK] = ode45(@ecdif, [a b], y0);

% interpolacion por splines cubicas
sp = linspace(a,b,100);
spp = spline(x_RK,y_RK,sp);


% Graficas comparativas de Euler hacia atras, Euler hacia adelante, R-K y sol analitica:
figure(1)
%Sol analitica
grosor = 1.5;
plot(x_exacta, y_exacta, "b",'LineWidth',grosor)
hold on
% Runge-Kuta
plot(x_RK, y_RK, "k")
% interpolacion por splines cubicas
plot(sp,spp,"--r",'LineWidth',grosor);
```

```
axis ([a b])
title ("Sol de la EDO");
xlabel ("x");
ylabel ("y");
hold off
legend_text = legend ("Sol Analitica", "Runge-Kuta", "Spline Cubica");
legend (legend_text, "location", "southeast");
```