UNIVERSIDAD FAVALORO

Trabajo Práctico Integrador
Métodos Numéricos

Alumnos: Antraygues Ezequiel, Campo Francisco, Vazquez Constanza

Docentes: Pessana Franco Martin, Perez Gonzalo

Ejercicio 1

Para este ejercicio primero realizamos un archivo con el contenido del sistema matricial para utilizar. Luego, utilizamos el Método Runge-Kutta para resolver este sistema matricial.

```
function [F]= Sistema_Matricial(t,X)
L1 = 0.03;
L2 = 0.17;
K1 = 30250;
K2 = 937;
M = 50;
J = 1;
B = 15;
g = 9.81;
Fo = 490;
fe = Fo;

A = zeros(4);
A(1,2)= K2*L2;
A(1,3)= -K2;
A(2,1)= -L2/J;
A(2,2)= -B/J;
A(2,4) = -L1/J;
A(3,1) = 1/M;
A(4,2) = K1*L1;

B = zeros(4,2);
B(2,1) = L2/J;
B(3,2) = -1;
U = [fe g];

F = A*X'+ B*U';%sistema matricial
F = F';
end
```

Metodo de RUNGE-KUTTA Orden 4
Resolucion de Sistemas Ecuaciones Diferenciales de primer Orden:
function [,X] = Ec_Dif_Runge_Kutta_04_Sistemas(F,t0,tf,X0,M)
Parametros de Entrada:
F = AX+B Nombre del Sistema de EC de Primer Orden (Contenido en forma Matricial en un script MatLab)
t0 = Tiempo inicial de an?lisis de la Ecuacion Diferencial
tf = Tiempo final de analisis de la Ecuacion Diferencial
X0 = [x1(t0) x2(t0) x3(t0)...xn(t0)] Condicion inicial del Sietema de ED de primer orden
Parametros de Salida:
T: Vector Columna de tiempo
X = Matriz resultado. Col#1: vector Solucion x1; Col#2: Vector solucion x2...;Col#n: vector solucion xn

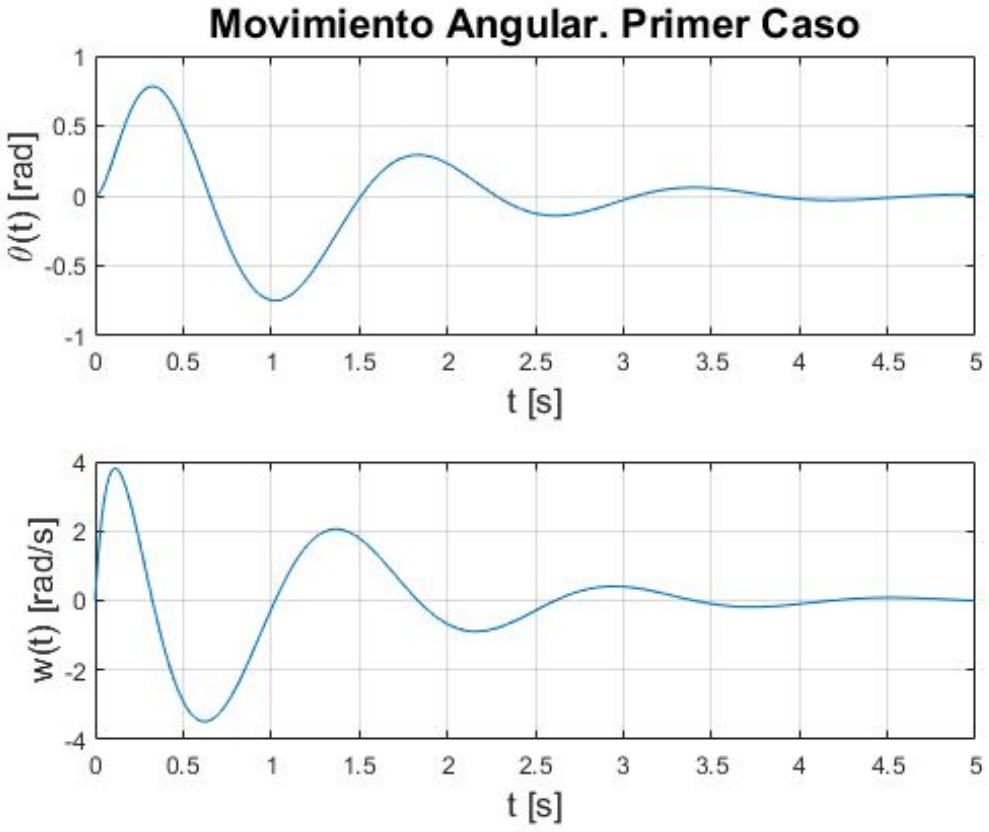
```
function [T,X] = Metodo_Runge_Kutta(F,t0,tf,X0,h)
N=(tf-t0)/h;
T=t0:h:tf;
X=zeros(N+1,length(X0));
X(1,:)=X0;
for k=1:N
    f1=feval(F,T(k),X(k,:));
    f2=feval(F,T(k)+h/2,X(k,:)+(h/2)*f1);
    f3=feval(F,T(k)+h/2,X(k,:)+(h/2)*f2);
    f4=feval(F,T(k)+h,X(k,:)+(h)*f3);
    X(k+1,:)=X(k,:)+h*(f1+2*f2+2*f3+f4)/6;
end
T=T';
```

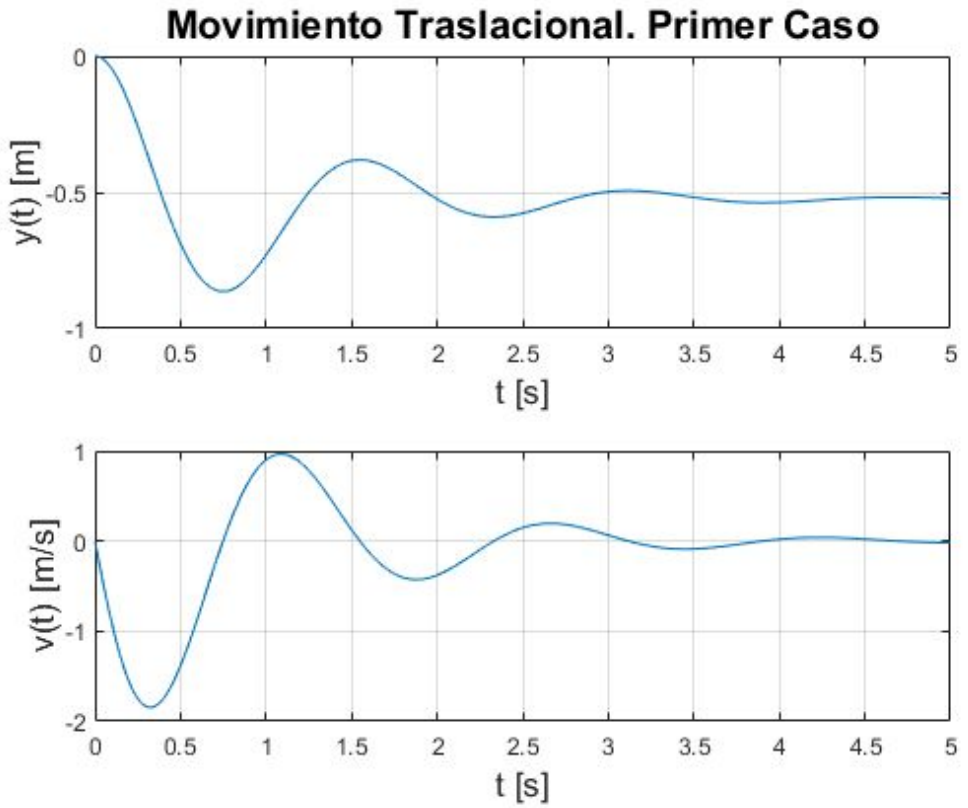
%%

Ejercicio 1 Primer Caso

```
L1 = 0.03;
L2 = 0.17;
K1 = 30250;
K2 = 937;
t0 = 0;
tf = 5;
xo = [0 0 0 0];     %condiciones iniciales
h = 0.01;           %resolución de intervalos
[T,X] = Metodo_Runge_Kutta('Sistema_Matricial',t0,tf,xo,h);
C=[0 0 0 1/(K1*L1); 0 1 0 0; -1/K2 0 0 L2/(K1*L1); 0 0 1 0];
save('Matriz_X.mat','X');       %guardamos los valores de X
Y=C*X';                         %ecuación de salida
save('Matriz_Y.mat','Y');       %guardamos salida del sistema

h1=figure(1);
subplot(2,1,1);
plot(T',Y(1,:));
grid
xlabel('t [s]','fontsize',13);
```

```
ylabel('\theta(t) [rad]','fontsize',13);
title('Movimiento Angular. Primer Caso','fontsize',15);
subplot(2,1,2);
plot(T',Y(2,:));
grid;
xlabel('t [s]','fontsize',13);
ylabel('w(t) [rad/s]','fontsize',13);

h2=figure(2);
subplot(2,1,1);
plot(T',Y(3,:));
grid
xlabel('t [s]','fontsize',13);
ylabel('y(t) [m]','fontsize',13);
title('Movimiento Traslacional. Primer Caso','fontsize',15);
subplot(2,1,2);
plot(T',Y(4,:));
grid;
xlabel('t [s]','fontsize',13);
ylabel('v(t) [m/s]','fontsize',13);
```





Luego realizamos el mismo procedimiento pero cambiando los valores de L2 y K2 y obtuvimos un resultado diferente.

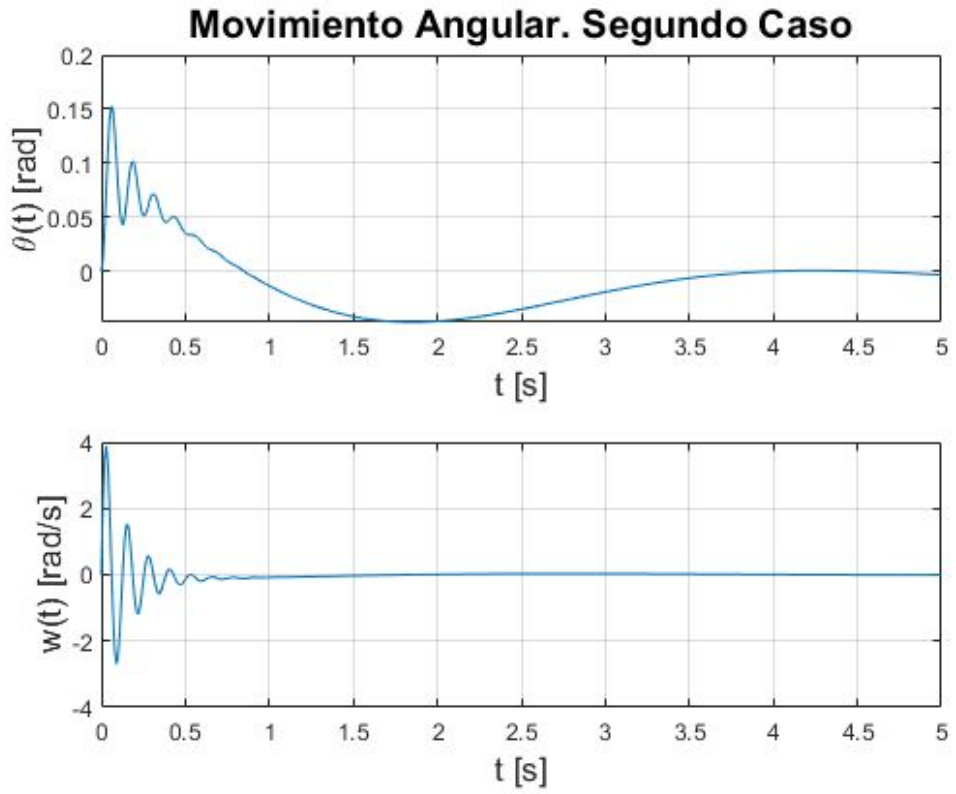
```
function [F]= matriz_mod(t,X)
L1 = 0.03;
L2 = 0.50;
K1 = 30250;
K2 = 9370;
M = 50;
J = 1;
B = 15;
g = 9.81;
Fo = 490;
fe = Fo;  %u(t)=1

A = zeros(4);
A(1,2)= K2*L2;
A(1,3)= -K2;
A(2,1)= -L2/J;
A(2,2)= -B/J;
A(2,4) = -L1/J;
A(3,1) = 1/M;
A(4,2) = K1*L1;

B = zeros(4,2);
B(2,1) = L2/J;
```

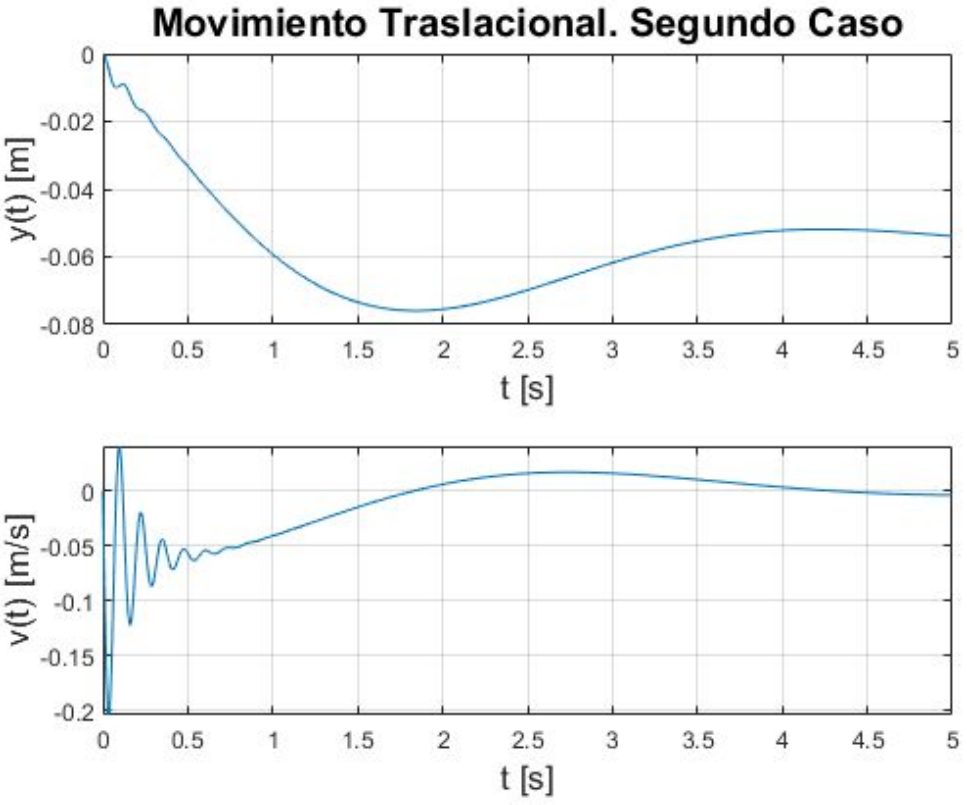
```
B(3,2) = -1;
U = [fe g];
F=A*X'+B*U';
F=F';
end
```

%%

Ejercicio 1 Segundo Caso

```
L1 = 0.03;
L2 = 0.50;
K1 = 30250;
K2 = 9370;
t0 = 0;
tf = 5;
xo = [0 0 0 0];
h = 0.01;
[T,X] = Metodo_Runge_Kutta('Sistema_Matricial_Mod',t0,tf,xo,h);

C=[0 0 0 1/(K1*L1); 0 1 0 0; -1/K2 0 0 L2/(K1*L1); 0 0 1 0];
Y=C*X';     %ecuación de salida

h1=figure(1);
subplot(2,1,1);
plot(T',Y(1,:));
grid
xlabel('t [s]','fontsize',13);
ylabel('\theta(t) [rad]','fontsize',13);
title('Movimiento Angular. Segundo Caso','fontsize',15);
subplot(2,1,2);
plot(T',Y(2,:));
grid;
xlabel('t [s]','fontsize',13);
ylabel('w(t) [rad/s]','fontsize',13);

h2=figure(2);
subplot(2,1,1);
plot(T',Y(3,:));
grid
xlabel('t [s]','fontsize',13);
ylabel('y(t) [m]','fontsize',13);
title('Movimiento Traslacional. Segundo Caso','fontsize',15);
subplot(2,1,2);
plot(T',Y(4,:));
grid;
xlabel('t [s]','fontsize',13);
ylabel('v(t) [m/s]','fontsize',13);
```


Movimiento Angular. Segundo Caso
θ(t) [rad]
t [s]
w(t) [rad/s]
t [s]


Movimiento Traslacional. Segundo Caso
y(t) [m]
t [s]
v(t) [m/s]
t [s]

Conclusión del Ejercicio 1

Comparando los movimientos traslacionales podemos observar que la posición y(t) en el primer caso tiende a estabilizarse en -0.5m mientras que en el segundo caso tiende a -0.05m. Esto es debido al cambio de la constante del resorte k2, al hacerlo más rígido la variación de posición es menor.

Para v(t) podemos observar que se reduce mucho el rango de velocidades y tiende a estabilizarse más rápidamente. Pensamos que esto se debe a que al aumentar la rigidez del resorte, esta limita aún más el movimiento del sistema masa-varilla.

Luego comparando los movimientos angulares podemos observar que la posición $\theta(t)$ en el primer caso se estabiliza en cero, mientras que en el segundo caso al ser la varilla más larga y el resorte más rígido tarda más en alcanzar el equilibrio (desfasado por debajo de cero). También debido a esto podemos observar que la velocidad w(t) en el primer caso tarda más en estabilizarse que en el segundo caso pero en ambas se estabiliza en cero.

Ejercicio 2

Para obtener las derivadas de y(t) y $\theta(t)$ pensamos en tres métodos distintos. El primero consistia en hacer una interpolación mediantes los valores discretos obtenidos en el Ejercicio 1 para así poder obtener otras dos funciones (pensamos en usar interpolación por splines cúbicas ya que de los métodos visto durante la cursada es el único que admite una nube de puntos de la magnitud obtenida en nuestro sistema) pero al intententarlo y notar su complejidad, sumado al hecho de que los últimos dos ejercicios están relacionados con dicho método, optamos por descartarlo.

Como segunda opción pensamos en modificar el ejercicio 1 quitando el intervalo t (dejando t como variable) para obtener funciones "$\theta(t)$" e "y" dependientes de una variable t, pero decidimos descartalo porque no nos pareció viable por el modo en que trabaja matlab.

Finalmente decidimos modificar la función de Extrapolación de Richardson para que funcione utilizando valores discretos (como los obtenidos en el Ejercicio 1) en lugar de una función.

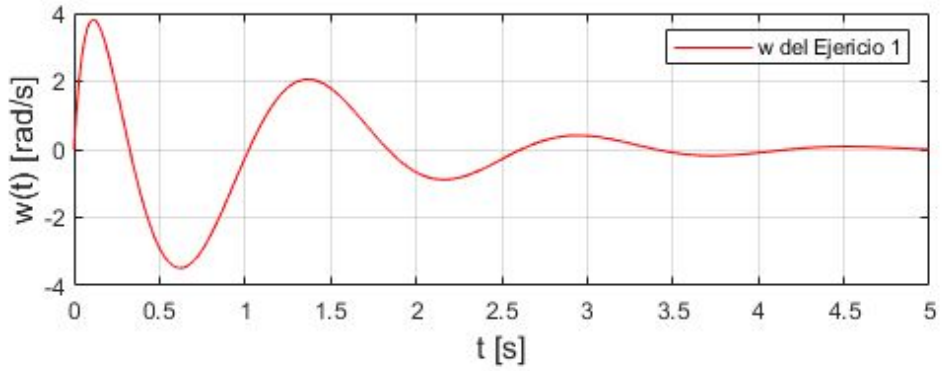
```
Extrapolación de Richardson
Parámetros de Entrada:
f = vector de valores discretos
x = abscisa donde se evaluará la derivada
Delta = Tolerancia del Error para las aproximaciones k-ésimas de la derivada
Tol = Tolerancia del Error Relativo para las aproximaciones k-ésimas de la derivada
Parámetros de Salida:
D =  matriz que contiene las aproximaciones de orden 2k en la diagonal principal
err = error absoluto final
relerr = error relativo final
n = el valor de k en donde se produce la mejor aproximación de la derivada
```

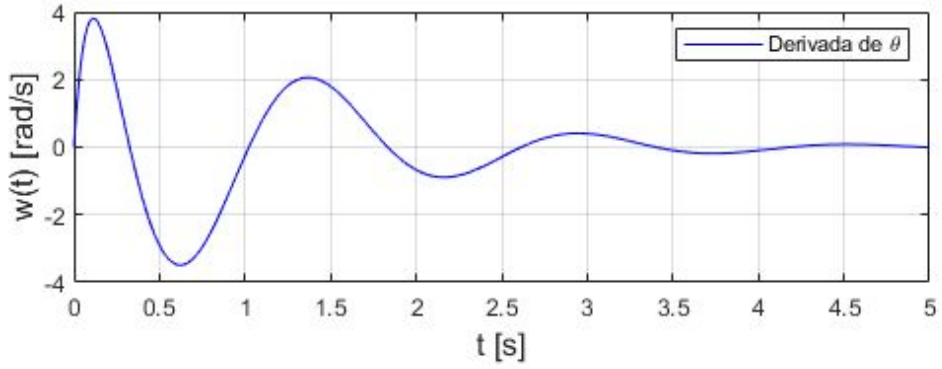
```
function [D,err,relerr,n] = Extrapolacion_Richardson(f,x,Delta,Tol)
Max = 15;
h = 1;
err = 1;
relerr = 1;
j = 1;
D(1,1) = (f(x+h)-f(x-h))/(2*h);   % este es el D0(h), cambiamos feval por los valores en
los puntos.
while (err > Delta) && (relerr > Tol) && (j <= Max)
    h = h/1.35; % obtuvimos este valor en el denominador mediante prueba y error
%comparando con los gráficos de ejercicio 1 hasta que nos quedaran iguales.
    D(j+1,1) = (f(x+1)-f(x-1))/(2*h);
    for k=1:j
       D(j+1,k+1) = D(j+1,k)+(D(j+1,k)-D(j,k))/((4^k)-1);
    end
    err = abs(D(j+1,j+1)-D(j,j));
    relerr = 2 * err / (abs(D(j+1,k+1))+abs(D(j,k))+eps);
    j = j+1;
end
n = size(D,1)-1;
```

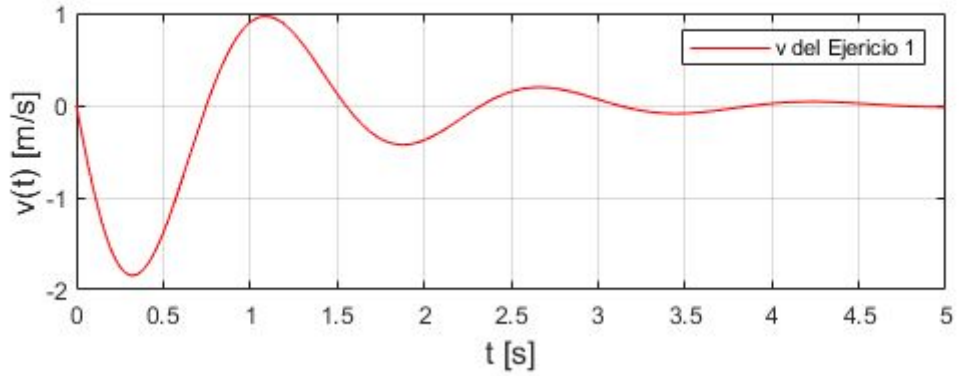
Ejercicio 2

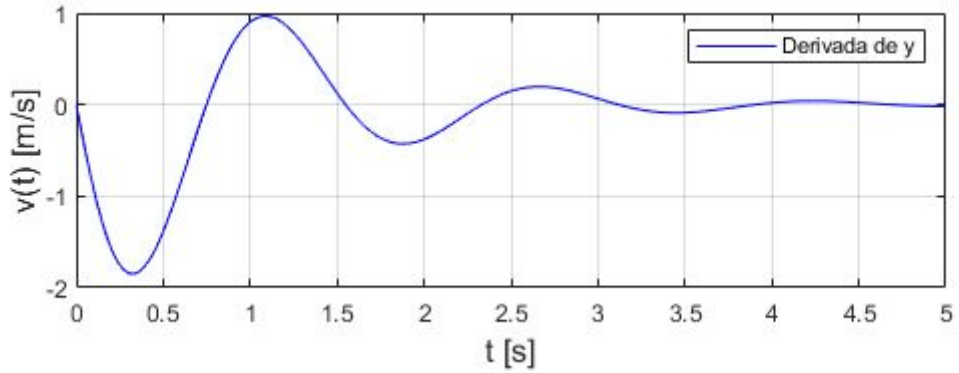
```
Delta=1e-6;
Tol=1e-7;
Y = load('Matriz_Y');       %para graficar ‘v’, ‘w’ y comparar (ejercicio 1)
h=0.01;
x=0:h:5;                    %hay que derivar tita e y
t_der=zeros(size(x));
y_der=zeros(size(x));
N=length(x);
for v=2:N-1
    [D,err,relerr,n] = Extrapolacion_Richardson(Y.Y(1,:),v,Delta,Tol);
    Di=diag(D);
    t_der(v)=Di(length(Di));
end
save('Derivada_t.mat','t_der'); %guardamos los valores de la derivada de tita
H1=figure(1);
subplot(2,1,1);
plot(x,Y.Y(2,:),'r');
legend('w del Ejericio 1')
grid
xlabel('t [s]','fontsize',12)
ylabel('w(t) [rad/s]','fontsize',12)
subplot(2,1,2);
plot(x,t_der,'b');
legend('Derivada de \theta')
xlabel('t [s]','fontsize',12)
ylabel('w(t) [rad/s]','fontsize',12)
grid
for v=2:N-1
    [D,err,relerr,n] = Extrapolacion_Richardson(Y.Y(3,:),v,Delta,Tol);
    Di=diag(D);
    y_der(v)=Di(length(Di));
end
save('Derivada_y.mat','y_der'); %guardamos los valores de la derivada de y
H2=figure(2);
subplot(2,1,1);
plot(x,Y.Y(4,:),'r');
legend('v del Ejericio 1')
grid
xlabel('t [s]','fontsize',12)
ylabel('v(t) [m/s]','fontsize',12)
subplot(2,1,2);
plot(x,y_der,'b');
legend('Derivada de y')
xlabel('t [s]','fontsize',12)
```

```
ylabel('v(t) [m/s]','fontsize',12)
grid
```









Conclusión del Ejercicio 2

Al finalizar este ejercicio podemos concluir que al modificar el método “Extrapolación de Richardson”, hecho por el profesor Franco Pessana, para que trabaje con valores discretos pudimos realizar una buena derivación de las posiciones angulares y traslacionales al comparar ambos gráficos.

Ejercicio 3

Para este ejercicio utilizamos un ajuste lineal por mínimos cuadrados para poder comparar los resultados del Ejercicio 1 con los resultados del Ejercicio 2. Obtuvimos la pendiente, la ordenada al origen, el coeficiente de correlación y calculamos el error cuadrático medio de cada ajuste.

```
Aproximación Lineal por Mínimos Cuadrado:
y = Ax + B
function [A,B,CC] = Ajuste_Lineal_MC(X,Y)
Parámetros de Entrada:
X = vector Nx1 con las abscisas de los pares de puntos
Y = vector Nx1 con las ordenadas de los pares de puntos
Parámetros de Salida:
A = pendiente del Ajuste Lineal
B = Ordenada al origen
CC = Coeficiente de Correlación
```

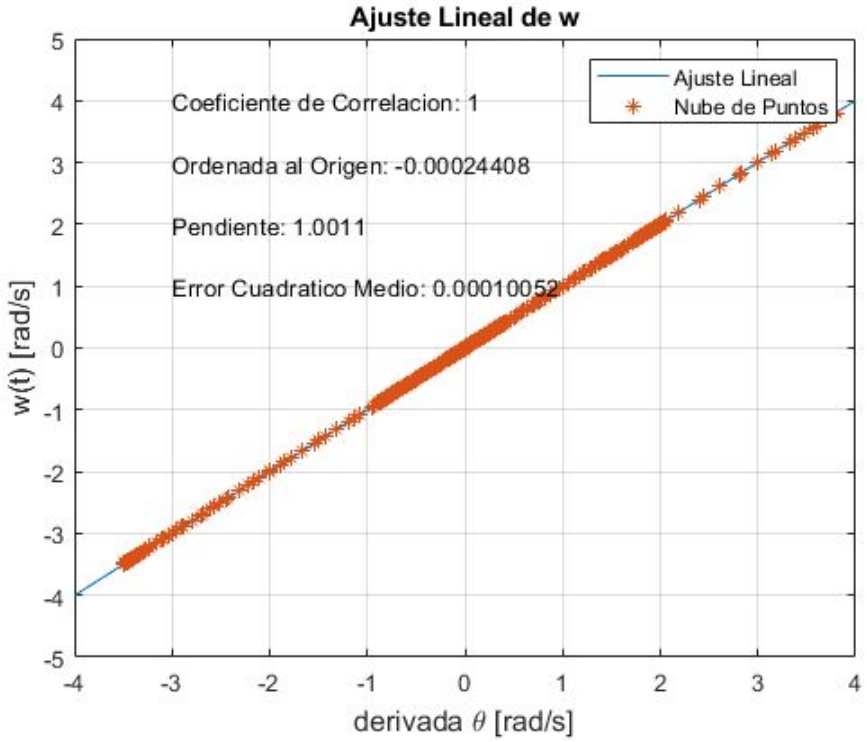
```
function [A,B,CC] = Aproximacion_Lineal(X,Y)

X_M=mean(X); % Promedio de abscisas
Y_M=mean(Y); % promedio de ordenadas

SX2=(X-X_M)'*(X-X_M); % suma de cuadrados de X
SY2=(Y-Y_M)'*(Y-Y_M); % suma de cuadrados de Y
SXY=(X-X_M)'*(Y-Y_M); % suma de productos cruzados de X e Y

A = SXY/SX2; % Pendiente del ajuste lineal
B = Y_M - A*X_M; % Ordenada del ajuste lineal
CC= SXY/sqrt(SX2*SY2); % Coeficiente de correlación de la recta
```

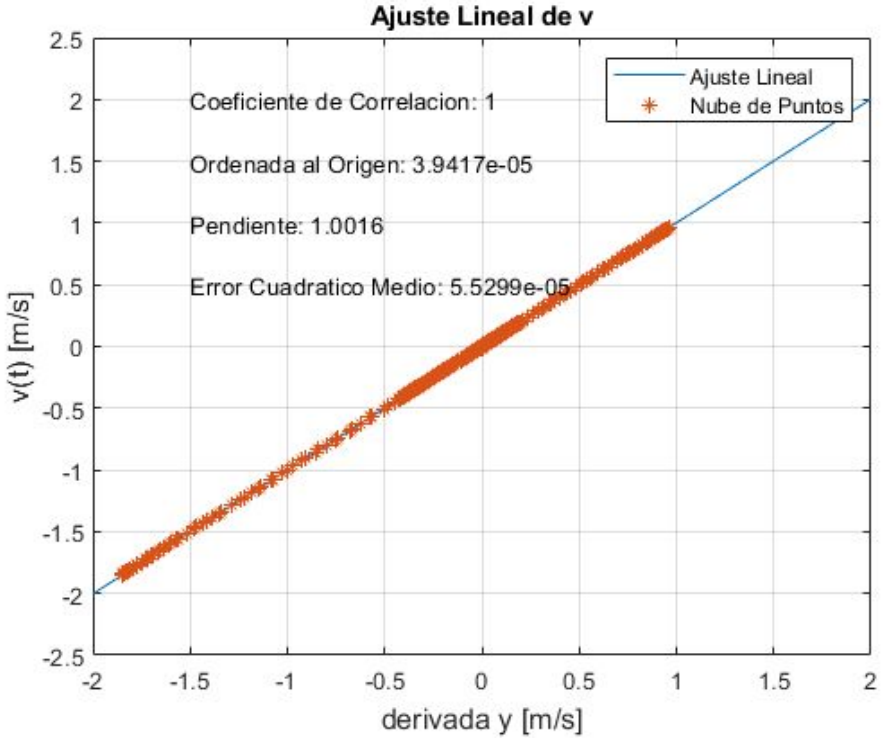
```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
Ejercicio 3
```

```
X = load('Matriz_Y');
Xw = X.Y(2,:);              %valores del ejercicio 1 de w

Y = load('Derivada_t');     %valores del ejercicio 2 de tita
Yw = Y.t_der;
[A,B,CC] = Aproximacion_Lineal(Xw',Yw');
x1=-4:0.01:4;
Ajuste_w = A*x1 + B;                        %recta de ajuste lineal
E1 = sqrt(sum((Xw-Yw).^2))/length(Xw);      %error cuadrático medio
H1=figure(1);
plot(x1,Ajuste_w,Yw,Xw,'*');
title('Ajuste Lineal de w')
xlabel('derivada \theta [rad/s]','fontsize',12)
ylabel('w(t) [rad/s]','fontsize',12)
legend('Ajuste Lineal','Nube de Puntos')
txt = ['Pendiente: ' num2str(A)];
```

```
text(-3,2,txt)
txt = ['Ordenada al Origen: ' num2str(B)];
text(-3,3,txt)
txt = ['Coeficiente de Correlacion: ' num2str(CC)];
text(-3,4,txt)
txt = ['Error Cuadratico Medio: ' num2str(E1)];
text(-3,1,txt)
grid

Xv = X.Y(4,:);           %valores del ejercicio 1 de v
Y = load('Derivada_y'); %valores del ejercicio 2 de y
Yv = Y.y_der;
[A,B,CC] = Aproximacion_Lineal(Xv',Yv');
x2=-2:0.01:2;
Ajuste_v = A*x2 + B;
E2 = sqrt(sum((Xv-Yv).^2))/length(Xv);  %error cuadrático medio
H2=figure(2);
plot(x2,Ajuste_v,Yv,Xv,'*');
title('Ajuste Lineal de v')
xlabel('derivada y [m/s]','fontsize',12)
ylabel('v(t) [m/s]','fontsize',12)
legend('Ajuste Lineal','Nube de Puntos')
txt = ['Pendiente: ' num2str(A)];
text(-1.5,1,txt)
txt = ['Ordenada al Origen: ' num2str(B)];
text(-1.5,1.5,txt)
txt = ['Coeficiente de Correlacion: ' num2str(CC)];
text(-1.5,2,txt)
txt = ['Error Cuadratico Medio: ' num2str(E2)];
text(-1.5,0.5,txt)
grid
```





Conclusión del Ejercicio 3

Observando que en ambos ajustes el coeficiente de correlación nos da uno, la pendiente nos da muy cercano a uno, también el error cuadrático medio y la ordenada nos dan muy cercano a cero podemos concluir que el ajuste es bueno y por lo tanto podemos corroborar que los resultados del Ejercicio 2 están bien.

Ejercicio 4

Para este ejercicio tuvimos que modificar la Regla de Simpson para que trabaje con valores discretos en vez de una función.

```
Aproximación central de derivada de orden cuadrático
Parámetros de Entrada:
f = vector de valores discretos
a y b limites de integracion
M = cantidad de sub intervalos
Parámetros de Salida:
S = la solucion de la integracion definida, suma discreta
```

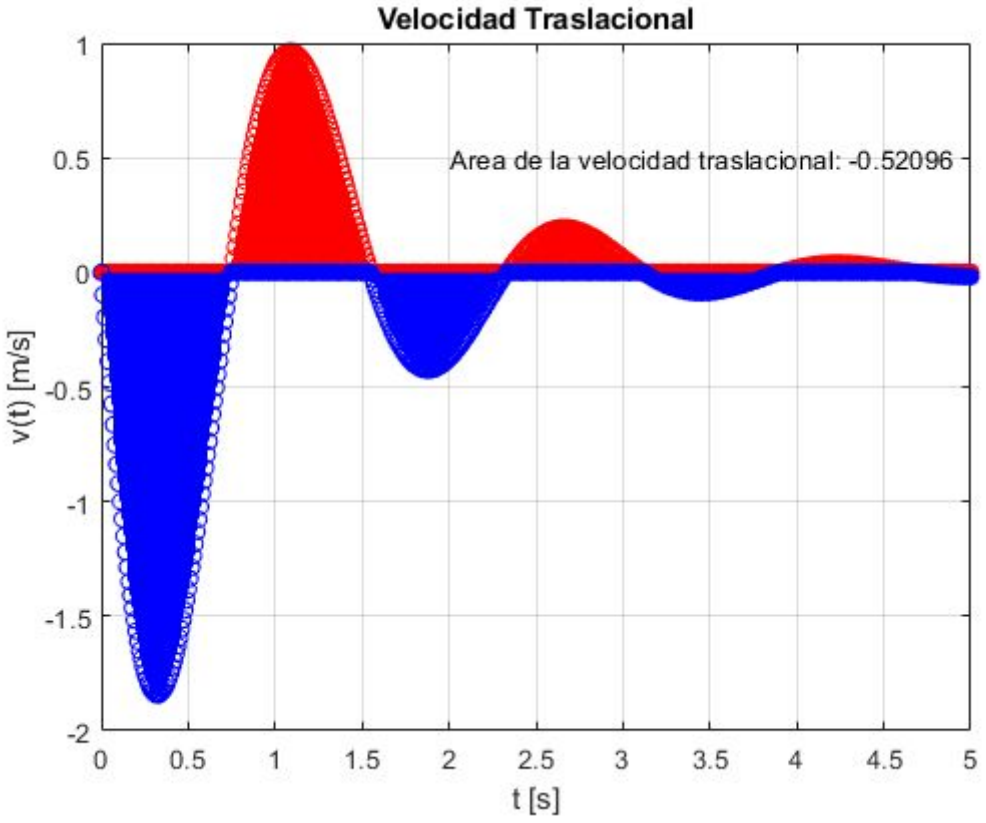
```
function S = Regla_Simpson_Compuesta(f,a,b,M)
h=(b-a)/(2*M);
S=0;
for k=3:M
    S = S +f(k-2)+4*f(k-1)+f(k);  %cambiamos feval por los valores en los puntos
end
S=S*h/3;
```

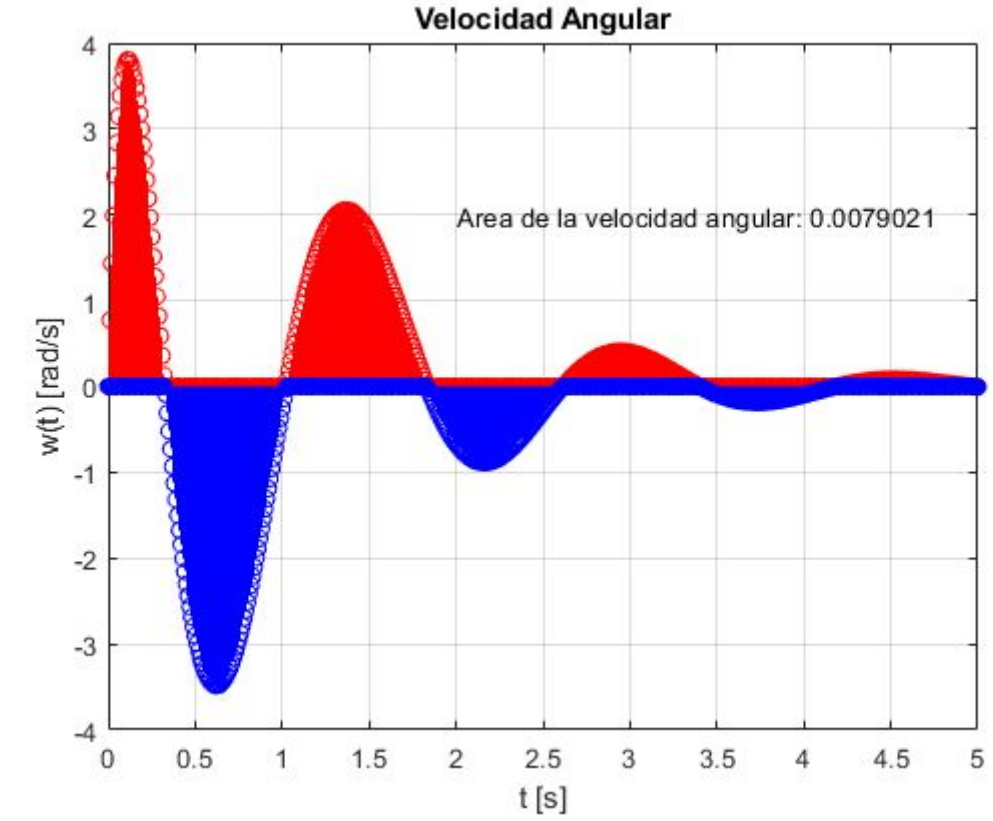
```
Ejercicio 4
```

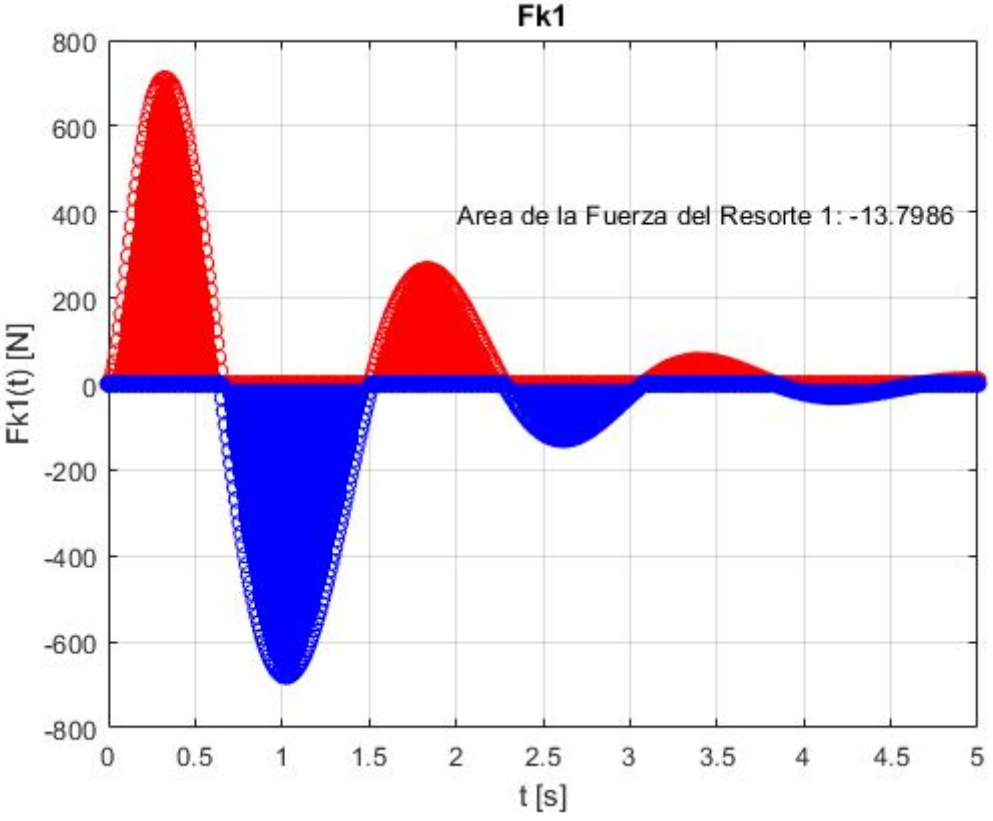
```
a = 0;      %limite inferior
b = 5;      %limite superior
M = 500;    %cantidad de sub intervalos
Y = load('Matriz_Y');   %datos del ejercicio 1
t=0:0.01:5;

Yv = Y.Y(4,:);          %datos de v
Sv = Regla_Simpson_Compuesta(Yv,a,b,M);

for k=1:1:length(Yv)      %para graficar de distinto color
    if(Yv(k)>0)
        Yv_pos(k) = Yv(k);
        Yv_neg(k) = 0;
    end
    if(Yv(k)<0)
        Yv_pos(k) = 0;
        Yv_neg(k) = Yv(k);
    end
end
h1=figure(1);
stem(t,Yv_pos,'r')
hold on
stem(t,Yv_neg,'b')
grid
txt = ['Area de la velocidad traslacional: ' num2str(Sv)];
text(2,0.5,txt)
```

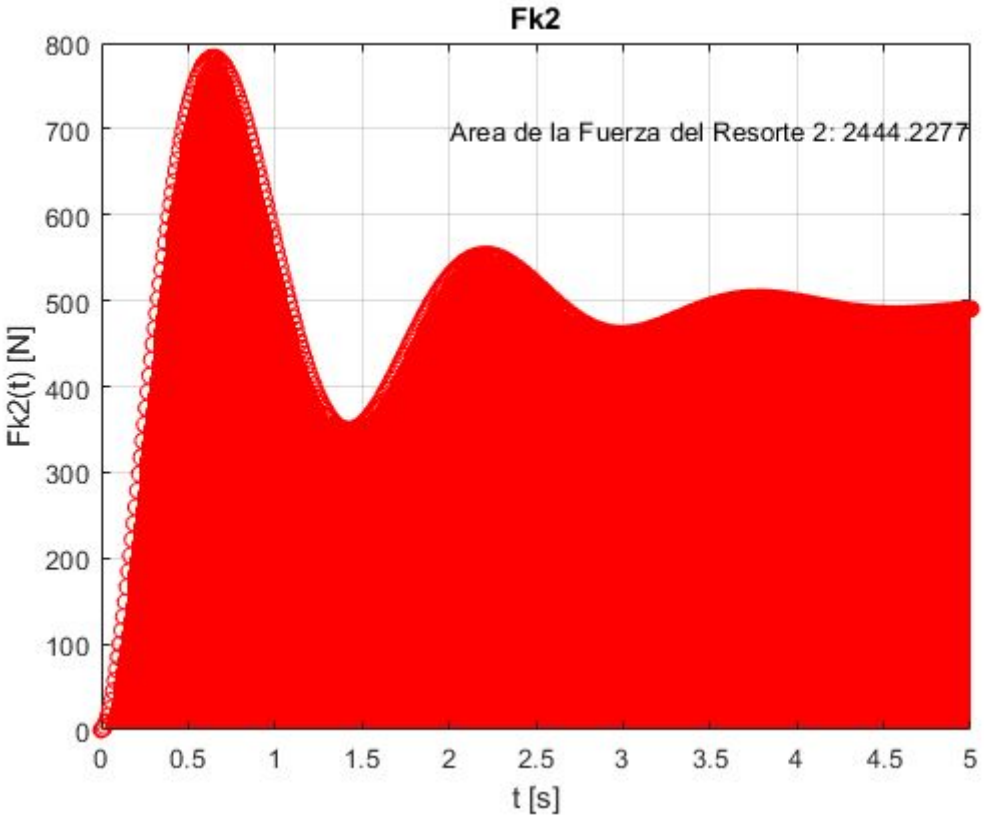
```
title('Velocidad Traslacional')
xlabel('t [s]')
ylabel('v(t) [m/s]')

Yw = Y.Y(2,:);          %datos de w
Sw = Regla_Simpson_Compuesta(Yw,a,b,M);

h2=figure(2);
for k=1:1:length(Yw)      %para graficar de distinto color
    if(Yw(k)>0)
        Yw_pos(k) = Yw(k);
        Yw_neg(k) = 0;
    end
    if(Yw(k)<0)
        Yw_pos(k) = 0;
        Yw_neg(k) = Yw(k);
    end
end
stem(t,Yw_pos,'r')
hold on
stem(t,Yw_neg,'b')
grid
txt = ['Area de la velocidad angular: ' num2str(Sw)];
text(2,2,txt)
title('Velocidad Angular')
xlabel('t [s]')
ylabel('w(t) [rad/s]')

X = load('Matriz_X');   %datos del ejercicio 1

Fk1 = X.X(:,4);          %datos de fk1
SFk1 = Regla_Simpson_Compuesta(Fk1',a,b,M);
h3=figure(3);
for k=1:1:length(Fk1)      %para graficar de distinto color
    if(Fk1(k)>0)
        Yfk1_pos(k) = Fk1(k);
        Yfk1_neg(k) = 0;
    end
    if(Fk1(k)<0)
        Yfk1_pos(k) = 0;
        Yfk1_neg(k) = Fk1(k);
    end
end
stem(t,Yfk1_pos,'r')
hold on
stem(t,Yfk1_neg,'b')
grid
txt = ['Area de la Fuerza del Resorte 1: ' num2str(SFk1)];
text(2,400,txt)
```

```
title('Fk1')
xlabel('t [s]')
ylabel('Fk1(t) [N]')

Fk2 = X.X(:,1);          %datos de fk2
SFk2 = Regla_Simpson_Compuesta(Fk2',a,b,M);
h4=figure(4);
stem(t,Fk2,'r')
grid
txt = ['Area de la Fuerza del Resorte 2: ' num2str(SFk2)];
text(2,700,txt)
title('Fk2')
xlabel('t [s]')
ylabel('Fk2(t) [N]')
```




Velocidad Angular
Area de la velocidad angular: 0.0079021
w(t) [rad/s]
t [s]
4
3
2
1
0
-1
-2
-3
-4
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5


Fk1
Area de la Fuerza del Resorte 1: -13.7986
Fk1(t) [N]
t [s]
800
600
400
200
0
-200
-400
-600
-800
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5



Conclusión del Ejercicio 4

Podemos observar que tanto el área de la velocidad traslacional como la velocidad angular son cercanas a cero debido que se trata de un sistema subamortiguado por lo que el área positiva y el área negativa se terminan anulando entre ellas.

Luego para el área de Fk1 se observa un comportamiento parecido, pero en este caso tenemos más área negativa que área positiva, por lo tanto el resultado termina siendo negativo.

Por último para el área de Fk2 termina siendo positiva porque el resorte nunca llega a comprimirse más allá de su posición inicial.

Ejercicio 5

Para este ejercicio usamos el método de Aproximación de Raíces Reales, pero también tuvimos que modificarlo para que funcione con valores discretos. Luego calculamos el promedio y desvío estándar de dichas raíces.

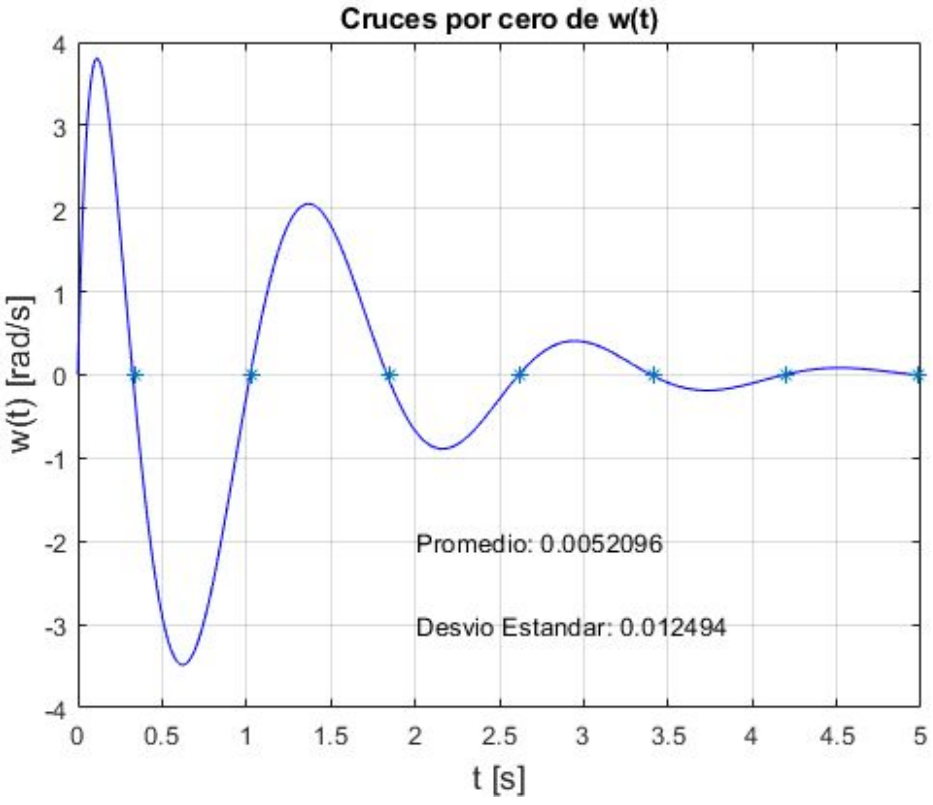
```
Solucion Aproximada de Raices Reales
function [r,A] = Aprox_Raices_Reales(f,delta)
Parámetros de Entrada:
f = vector de valores discretos
delta = error final querido (exactitud del método)
Parámetros de Salida:
r  = vector de raices aproximadas de f(x) para el intervalo [a,b]
A = vector de valores de las raices
```

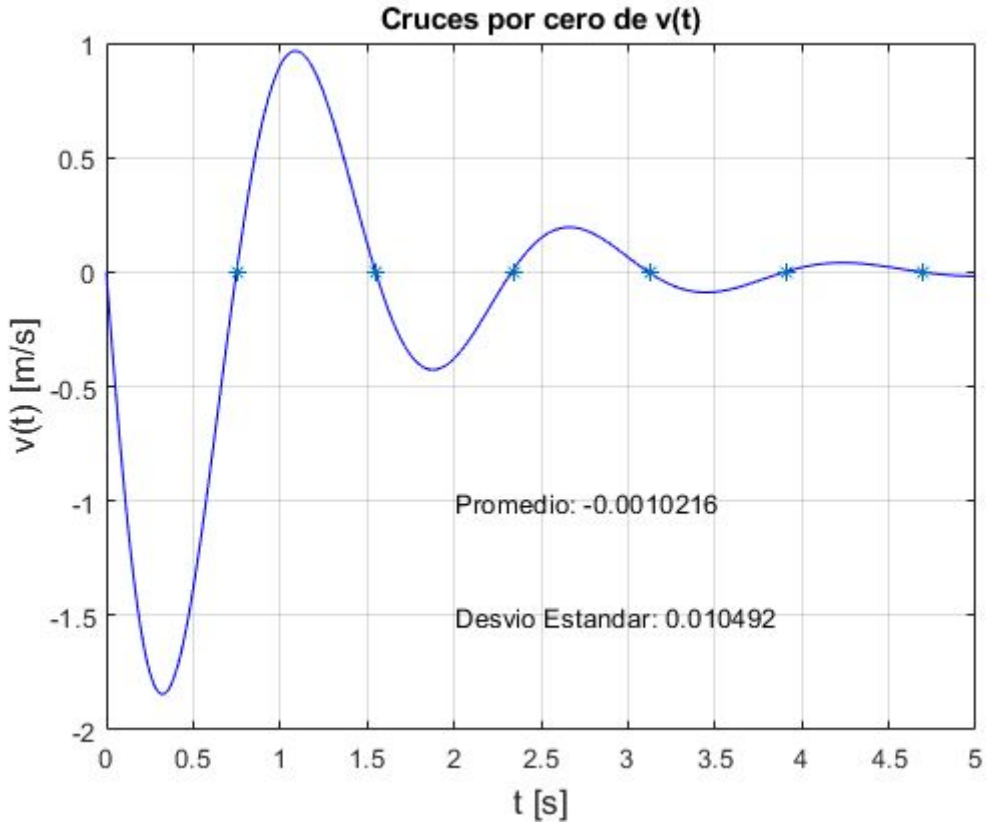
```
function [r,A] = Aprox_Raices_Reales(f,delta)
cont = 1;
r = zeros();
A = zeros();
for k=2:length(f)-1       %cambiamos feval por los valores en los puntos
    y_xk = f(k);          %xk
    y_xk1 = f(k-1);       %xk-1
    y_xk2 = f(k+1);       %xk+1

    if (y_xk*y_xk1) < 0
        r(cont) = (k+k-1)/2;
        A(cont) = (y_xk+y_xk1)/2;
        cont = cont+1;
    elseif ((y_xk2 - y_xk)*(y_xk - y_xk1)) < 0 && abs(y_xk) < delta
        r(cont) = k;
        A(cont) = y_xk;
        cont = cont+1;
    end
end
```

```
Ejercicio 5
```

```
delta = 0.00000000000001;
a=0;
b=5;
Y = load('Matriz_Y.mat');   %valores del ejercicio 1
Yw = Y.Y(2,:);
[r,A] = Aprox_Raices_Reales(Yw,delta);
Prom = mean(A);              %calcula el valor medio
Desvio_est = std(A);         %calcula el desvio estandar
x=0:0.01:5;
z = zeros(1,length(r));
H1 = figure(1);
plot(x,Yw,'b',r.*10^-2,z,'*')
```

```
title('Cruces por cero de w(t)')
xlabel('t [s]','fontsize',13);
ylabel('w(t) [rad/s]','fontsize',13);
txt = ['Promedio: ' num2str(Prom)];
text(2,-2,txt)
txt = ['Desvio Estandar: ' num2str(Desvio_est)];
text(2,-3,txt)
grid

Yv = Y.Y(4,:);
[r,A] = Aprox_Raices_Reales(Yv,delta);
Prom = mean(A);              %calcula el valor medio
Desvio_est = std(A);         %calcula el desvio estandar
x=0:0.01:5;
z = zeros(1,length(r));
H2 = figure(2);
plot(x,Yv,'b',r.*10^-2,z,'*')
title('Cruces por cero de v(t)')
xlabel('t [s]','fontsize',13);
ylabel('v(t) [m/s]','fontsize',13);
txt = ['Promedio: ' num2str(Prom)];
text(2,-1,txt)
txt = ['Desvio Estandar: ' num2str(Desvio_est)];
text(2,-1.5,txt)
grid
```





Conclusión del Ejercicio 5

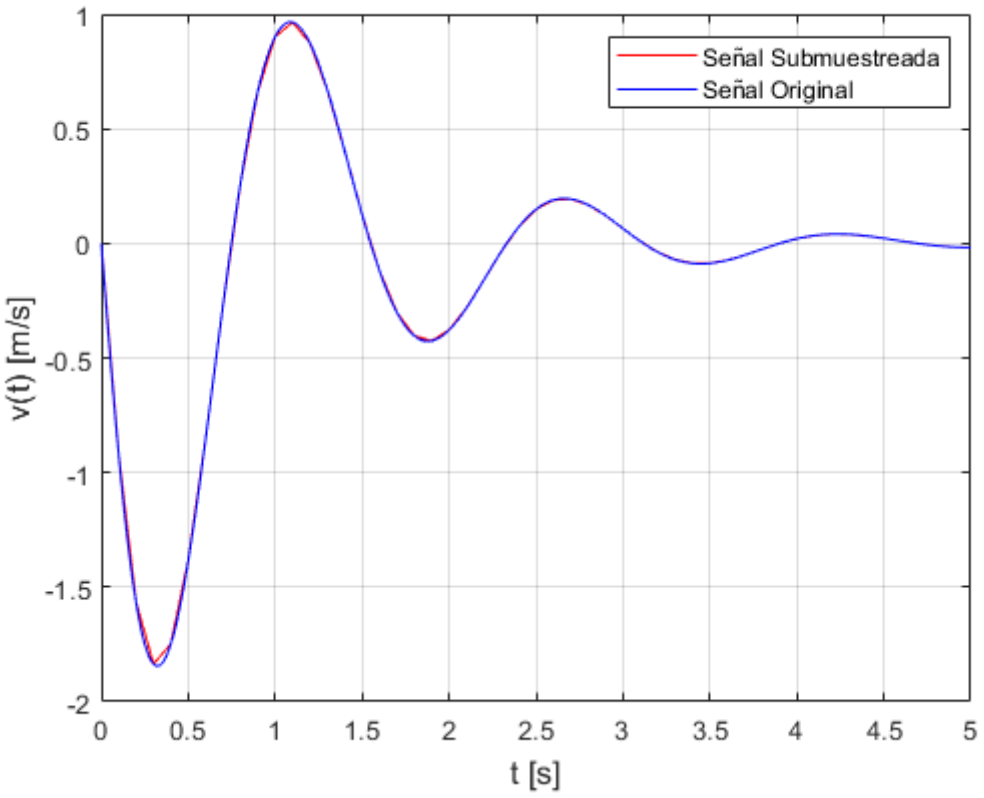
Se puede observar a simple vista que el método utilizado aproxima muy bien las raíces pero asumimos que al trabajar con una nube de puntos y al no tener una función se puede cometer un leve error como se puede observar en el desvío estándar y el promedio.

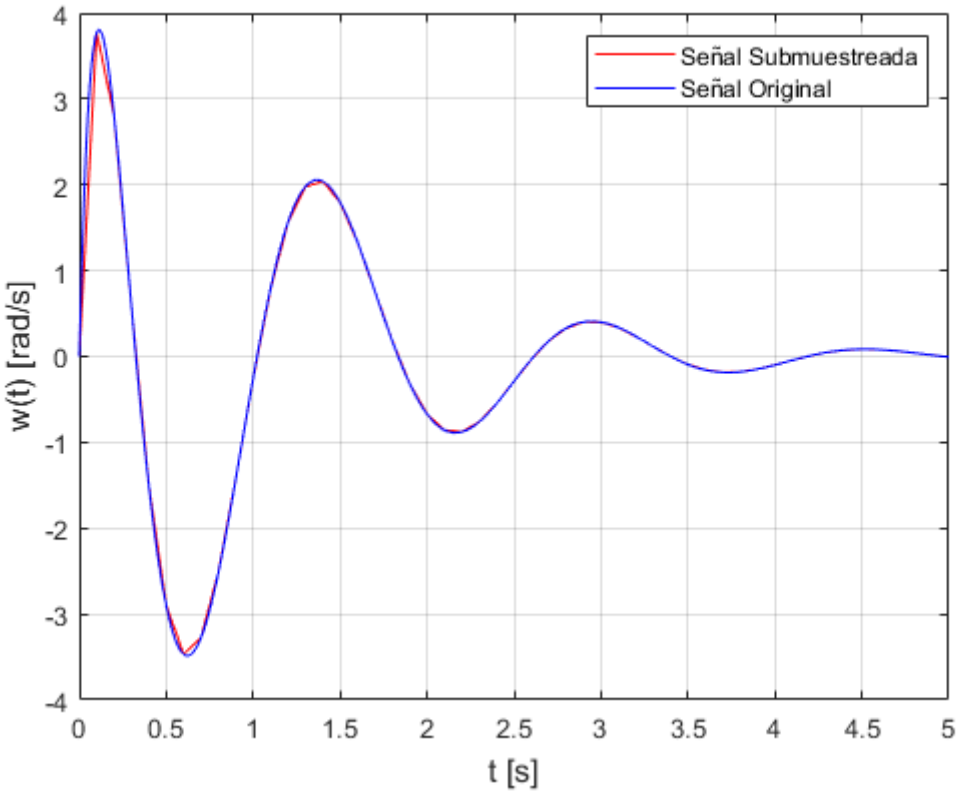
Ejercicio 6

Para este ejercicio reducimos el número de puntos de la función original, tomando un punto cada diez. Por lo tanto pasamos de tener 500 puntos a tener 50 y tenemos menos resolución lo que ocasiona pérdida información.

```
Ejercicio 6
```

```
Y1 = load('Matriz_Y');      %datos del ejercicio 1
T1 = 0:0.01:5;              %tiempo de muestreo original

T2 = 0:0.1:5;                       %tiempo de submuestro
Y2 = Y1.Y(4,1:10:length(Y1.Y));     %datos de v submuestreados
Y3 = Y1.Y(2,1:10:length(Y1.Y));     %datos de w submuestreados

save('V_Sub.mat','Y2');     %guardamos los datos submuestreados para
save('W_Sub.mat','Y3');     %el proximo ejercicio

h1=figure(1);
plot(T2,Y2,'r',T1,Y1.Y(4,:),'b');
grid;
legend('Señal Submuestreada','Señal Original')
xlabel('t [s]','fontsize',12);
ylabel('v(t) [m/s]','fontsize',12);

h2=figure(2);
plot(T2,Y3,'r',T1,Y1.Y(2,:),'b');
grid;
legend('Señal Submuestreada','Señal Original')
xlabel('t [s]','fontsize',12);
ylabel('w(t) [rad/s]','fontsize',12);
```


Señal Submuestreada
Señal Original
v(t) [m/s]
t [s]
1
0.5
0
-0.5
-1
-1.5
-2
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5


Señal Submuestreada
Señal Original
w(t) [rad/s]
t [s]
4
3
2
1
0
-1
-2
-3
-4
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5

Conclusión del ejercicio 6

Al implementar el submuestreo y contrastarlo gráficamente podemos observar que si bien la forma de la señal es la misma, hay una pérdida de información por lo que la resolución es claramente menor a la original y esto se refleja en la poca suavidad de la curva.

Ejercicio 7

Utilizamos Spline Cúbicas para recuperar la información que perdimos en el Ejercicio 6.

```
Interpolacion por Splines Cubicas
Se recibe un conjunto de (N+1) puntos a interpolar:
(x0,y0),(x1,y1),(x2,y2)...(xk,yk),...,(xN,yN)
Se arma un sistema de N-1 ecuaciones para las derivadas
segundas de S(xk), es decir, S''(xk), contando con
un sistema lineal tridiagonal y se entrega una matriz de
N filas por 4 columnas conteniendo los coeficientes en
potencias decrecientes de x de la funci?n S(x) con cada
uno de los polinomios interpoladores de k=0,1,2,...,N-1
S(x)=Sk(x)=s3,k*(x-xk)^3+s2,k(x-xk)^2+s1,k(x-xk)+s0,k
Se trabaja con Spline cubica natural, es decir, con
la condicion S''(x0=a)=S''(xN=b)=0
```

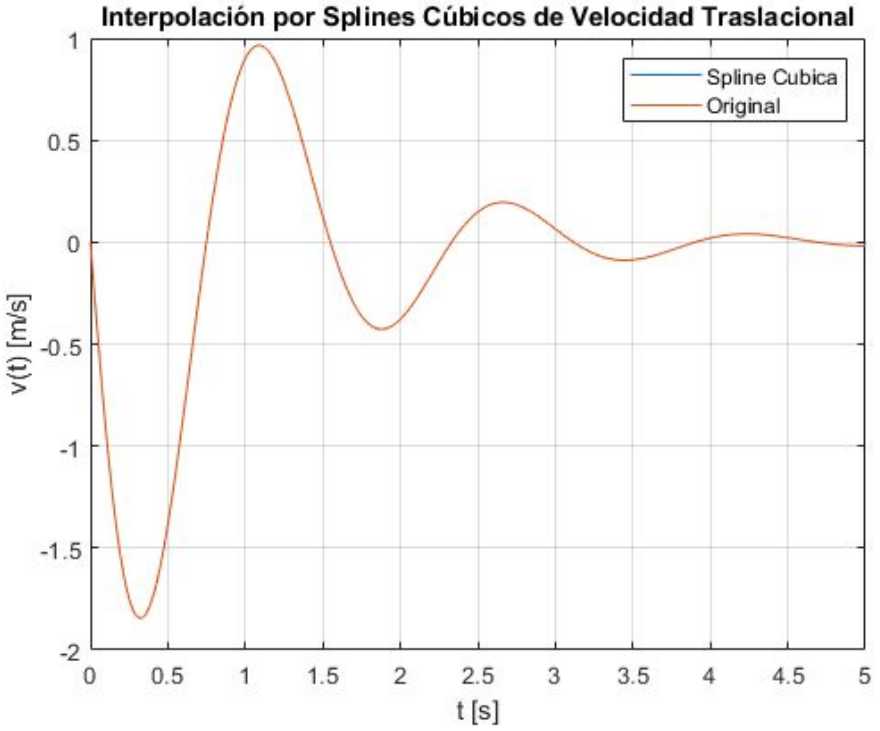
```
function Mat_S=Spline_Cubica(X,Y)

N=length(X)-1; % N+1 cantidad de puntos a interpolar
hk=diff(X);   % Armado de h(k)=x(k+1)-x(k)
dk=diff(Y)./hk; % Armado de dk=(y(k+1)-y(k))/h(k)
m0=0; % Constantes para Spline suavizada, es decir,
mN=0; % derivadas segundas en extremos nulas.
a=hk(2:N-1); % Armado de diagonal inf. para N-1 ecuaciones de mk
b=2*(hk(1:N-1)+hk(2:N)); % armado de diagonal principal
c=hk(2:N-1); % Armado de diagonal sup. para N-1 ecuaciones de mk
uk=6*diff(dk); % Vector Ind. Otra forma: uk=6*(dk(2:N)-dk(1:N-1));
B=uk;
B(1)=uk(1)-hk(1)*m0;    % Condiciones para una
B(N-1)=uk(N-1)-hk(N-1)*mN; % Spline cubica suavizada
%%%%% Calculo de m1,m2,mN-1 para el sistema tridiagonal:
mk=zeros(N+1,1);
mk(1)=m0;
mk(N+1)=mN;
N1=N-1;
for k=2:N1
    Piv=a(k-1)/b(k-1);
    b(k)=b(k)-Piv.*c(k-1); % Triangulacion inferior del
    B(k)=B(k)-Piv*B(k-1);  % Sistema tridiagonal
end
X=zeros(N1,1);
X(N1)=B(N1)/b(N1);
for k=N1-1:-1:1   % Sustitucion hacia atras para calculo de mk
    X(k)=(B(k)-c(k)*X(k+1))/b(k);
end
mk(2:N)=X;
Mat_S=zeros(N,4); % Inicializ. de matriz de constantes de Spline
for k=0:N-1
```

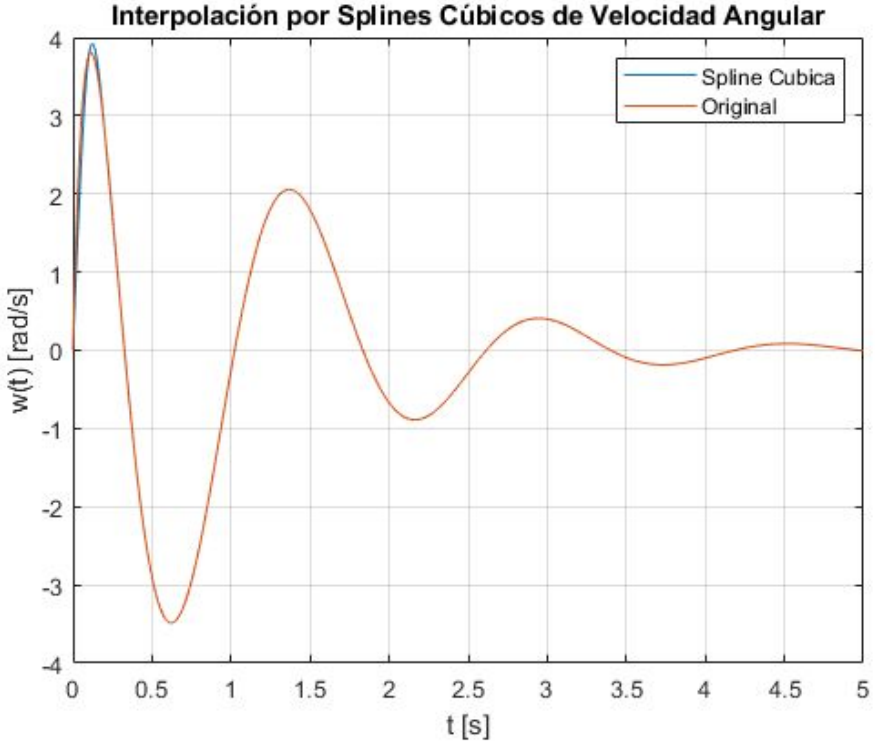
```
    Mat_S(k+1,1)=(mk(k+2)-mk(k+1))/(6*hk(k+1));
    Mat_S(k+1,2)=mk(k+1)/2;  % Armado de los Ck del spline cubico
    Mat_S(k+1,3)=dk(k+1)-hk(k+1)*(2*mk(k+1)+mk(k+2))/6;
    Mat_S(k+1,4)=Y(k+1);
end
```

Ejercicio 7

```
Y = load('Matriz_Y.mat');

Y1 = load('V_Sub.mat');
Yv = Y1.Y2;
Xo = 0:0.1:5;
A = Spline_Cubica(Xo,Yv);

X = 0:0.01:5;
N = length(Yv)-1;
Poli_Int = zeros(size(X)); % Inicialización de valores del polinomio
k=1;
Ind=1;
x2 = 0;
cont = 1;
while k<=N
    if X(k)<X(Ind+1)
        x1 = x2;
        x2 = x2 + 0.09;
        for j=x1:0.01:x2      % Evaluamos cada polinomio en los puntos omitidos
            Poli_Int(cont)=Eval_Polinomio_single(j,A(Ind,:),x1);
            cont = cont+1;
        end
        x2 = x2+0.01;
        k=k+1;
    else
        Ind=Ind+1;
    end
    if Ind>N
        k=N+1;
    end
end
save('V_Int.mat','Poli_Int');
H1=figure(1);
plot(X,Poli_Int,X,Y.Y(4,:)) % Nube de puntos originales y Spline Cúbica
title('Interpolación por Splines Cúbicos de Velocidad Traslacional');
xlabel('t [s]')
ylabel('v(t) [m/s]')
legend('Spline Cubica','Original')
grid
```

```
Y = load('W_Sub.mat');
Yw = Y.Y3;
X = 0:0.1:5;
A = Spline_Cubica(X,Yw);
X = 0:0.01:5;
N = length(Yw)-1;
Poli_Int = zeros(size(X)); % Inicialización de valores del polinomio
k=1;
Ind=1;
x2 = 0;
cont = 1;
while k<=N
    if X(k)<X(Ind+1)
        x1 = x2;
        x2 = x2 + 0.09;
        for j=x1:0.01:x2 % Evaluamos cada polinomio en los puntos omitidos
            Poli_Int(cont)=Eval_Polinomio_single(j,A(Ind,:),x1);
            cont = cont+1;
        end
        x2 = x2+0.01;
        k=k+1;
    else
        Ind=Ind+1;
    end
    if Ind>N
        k=N+1;
    end
end
save('W_Int.mat','Poli_Int');
Y1 = load('Matriz_Y');      %datos del ejercicio 1
H2=figure(2);
plot(X,Poli_Int,X,Y1.Y(2,:))      % Nube de puntos originales y Spline Cúbica
title('Interpolación por Splines Cúbicos de Velocidad Angular');
xlabel('t [s]')
ylabel('w(t) [rad/s]')
legend('Spline Cubica','Original')
grid
```





Conclusión del ejercicio 7

Utilizando el método de Interpolación por Splines Cúbicas, realizado por el profesor Franco Pessana, pudimos recuperar gran parte de la información perdida y esto se refleja en la similitud de nuestra interpolación con la señal original, sin embargo notamos que en la velocidad angular si bien se tuvo una curva muy parecida, no se ajusta perfectamente a la original.

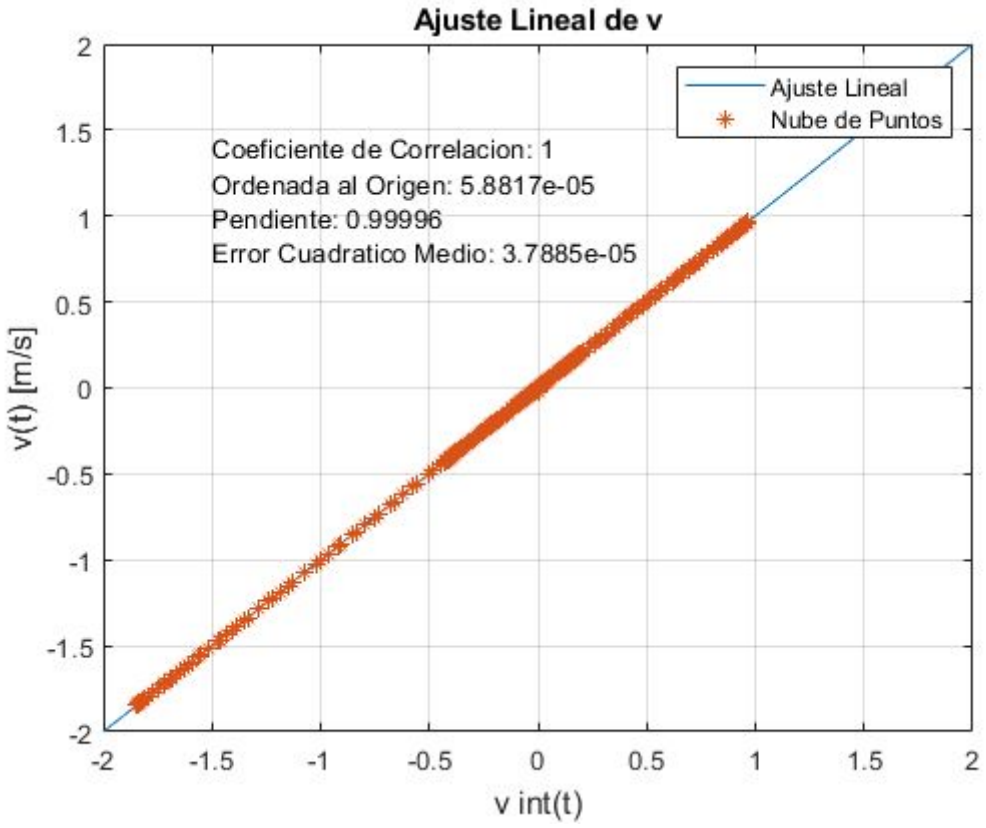
Ejercicio 8

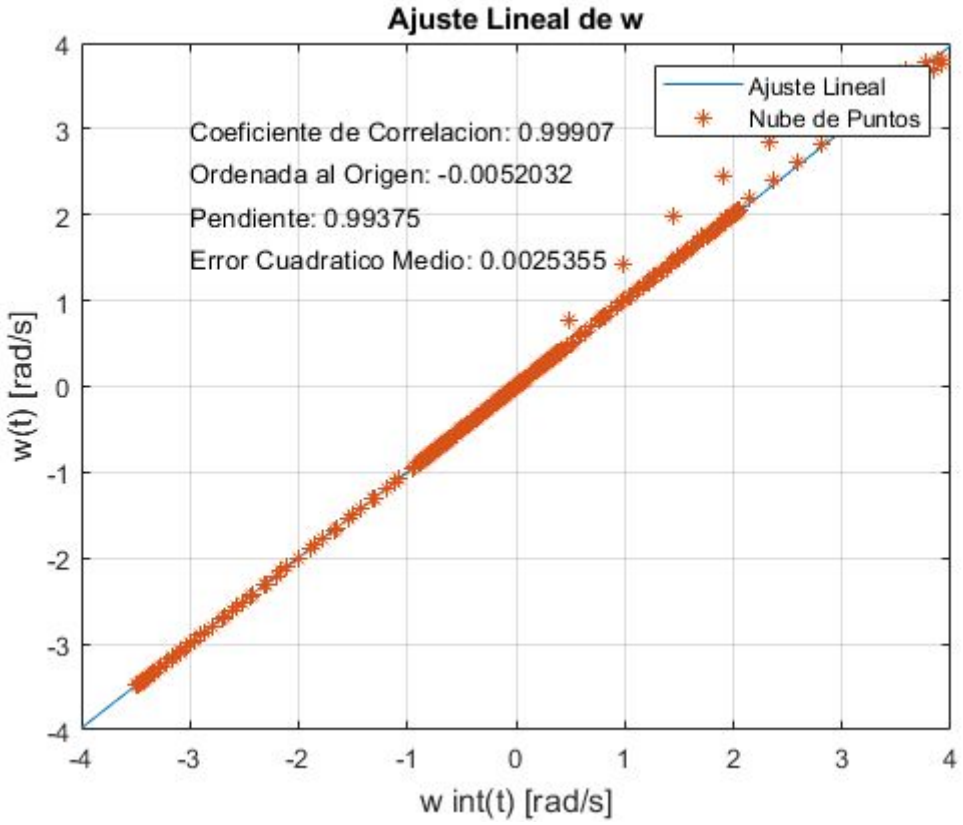
Para este ejercicio utilizamos un ajuste lineal por mínimos cuadrados para poder comparar los resultados del Ejercicio 1 con los resultados del Ejercicio 7. Obtuvimos la pendiente, la ordenada al origen, el coeficiente de correlación y calculamos el error cuadrático medio de cada ajuste. El método de Aproximación Lineal utilizado es el mismo que en el Ejercicio 3.

```
Ejercicio 8
```

```
X = load('Matriz_Y');   %valores del ejercicio 1
Xv = X.Y(4,:);          %valores de v

Y = load('V_Int');      %valores del ejercicio 7 de v
V_int = Y.Poli_Int;
[A,B,CC] = Aproximacion_Lineal(Xv',V_int');
x2=-2:0.01:2;
Ajuste_v = A*x2 + B;
E2 = sqrt(sum((Xv-V_int).^2))/length(Xv);  %error cuadratico medio
H2=figure(2);
plot(x2,Ajuste_v,V_int,Xv,'*');
title('Ajuste Lineal de v')
xlabel('derivada y [m/s]','fontsize',12)
ylabel('v(t) [m/s]','fontsize',12)
legend('Ajuste Lineal','Nube de Puntos')
txt = ['Pendiente: ' num2str(A)];
text(-1.5,1,txt)
txt = ['Ordenada al Origen: ' num2str(B)];
text(-1.5,1.2,txt)
txt = ['Coeficiente de Correlacion: ' num2str(CC)];
text(-1.5,1.4,txt)
txt = ['Error Cuadratico Medio: ' num2str(E2)];
text(-1.5,0.8,txt)
grid

Xw = X.Y(2,:);
Y = load('W_Int');     %valores del ejercicio 7 de w
W_int = Y.Poli_Int;

[A,B,CC] = Aproximacion_Lineal(Xw',W_int');
x1=-4:0.01:4;
Ajuste_w = A*x1 + B;
E1 = sqrt(sum((Xw-W_int).^2))/length(Xw);      %error cuadratico medio
H1=figure(1);
plot(x1,Ajuste_w,W_int,Xw,'*');
title('Ajuste Lineal de w')
xlabel('derivada \theta [rad/s]','fontsize',12)
ylabel('w(t) [rad/s]','fontsize',12)
legend('Ajuste Lineal','Nube de Puntos')
```

```
txt = ['Pendiente: ' num2str(A)];
text(-3,1.5,txt)
txt = ['Ordenada al Origen: ' num2str(B)];
text(-3,2,txt)
txt = ['Coeficiente de Correlacion: ' num2str(CC)];
text(-3,2.5,txt)
txt = ['Error Cuadratico Medio: ' num2str(E1)];
text(-3,1,txt)
grid
```





Conclusión del Ejercicio 8

Observando que en ambos ajustes el coeficiente de correlación y pendiente nos da muy cercano a uno, que el error cuadrático medio y la ordenada nos dan muy cercano a cero podemos concluir que el ajuste es bueno y por lo tanto podemos corroborar que los resultados del ejercicio anterior están bien.

Podemos notar que en la velocidad angular hay una pequeña dispersión que corresponderia a que el ajuste por spline cúbicas no fue tan exacto, esto se ve reflejado en la diferencia en el gráfico de velocidad angular del Ejercicio 6.

CONCLUSIÓN GENERAL

Mediante la realización de este trabajo pudimos comprobar por nuestros propios medios que se puede modelizar y analizar el comportamiento de sistemas físicos de una forma eficiente mediante métodos numéricos y elementos computacionales aprendidos a lo largo de la cursada, ya que al analizar dichos sistemas de otra manera resultaría muy complejo y requeriría mucho tiempo.