Identificación experimental, ejemplo masa-muelle-amortiguador utilizando algoritmo genético [**ga**] o/y **fmincon**: modelo "**no lineal**" del muelle.

Presentación en vídeo: http://personales.upv.es/asala/YT/V/identganl.html

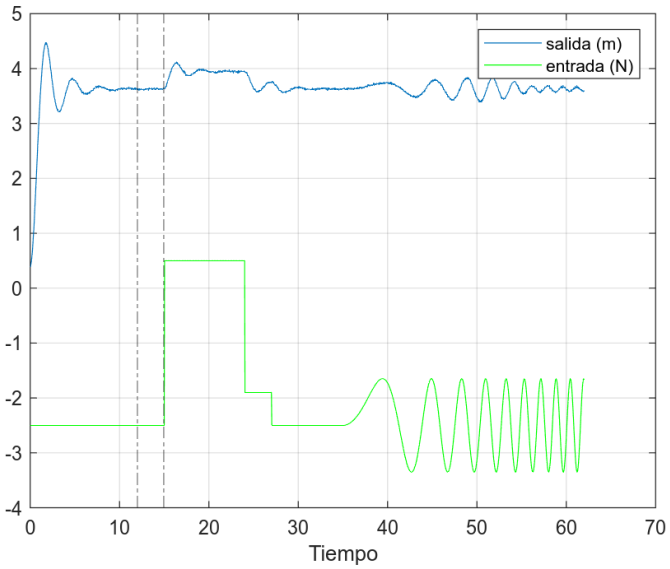
Este código funcionó sin errores en Matlab **R2022b** (Linux)

Objetivo: Calcular los parámetros de un modelo masa-muelle-amortiguador **NO lineal** que "mejor" se ajustan (en sentido de mínimos cuadrados) a unos datos experimentales. Comparar con modelo "*lineal*" a ver si merece la pena.

Tabla de Contenidos

Importación y preprocesado de datos

```
load experimento.mat
plot(datos.T,datos.pos,datos.T,datos.F,'g'), grid on
xline(12,'-.'),xline(14.95,'-.'), legend("salida (m)","entrada (N)"), xlabel("Tiempo")
```



Selección de la zona de datos adecuada y punto de operación

Los primeros 10 segundos son un transitorio "raro" (o al menos, con incrementos grandes)... en teoría de sistemas lineales se busca un modelo alrededor de un punto de operación (equilibrio).

```
i1=find(datos.T>=12,1)
```

```
i1 = 301
```

```
i2=find(datos.T<14.95,1,'last')
```

```
i2 = 374
```

```
ptoeq_y=mean(datos.pos(i1:i2))
```

```
ptoeq_y = 3.6267
```

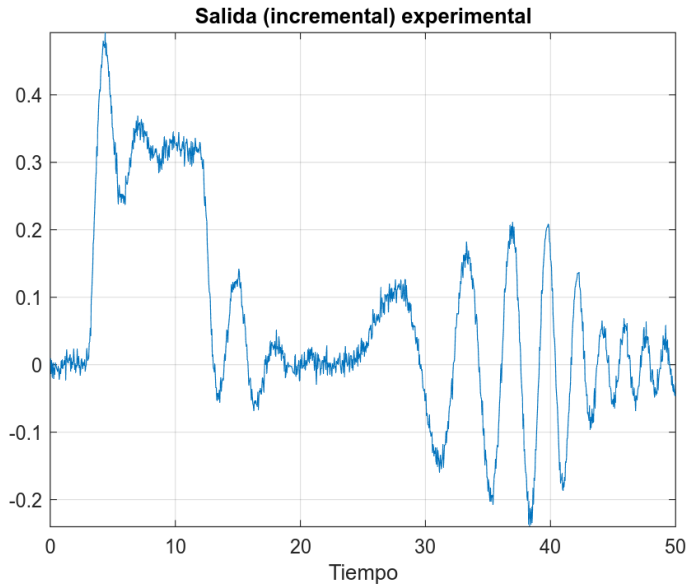
```
ptoeq_F=mean(datos.F(i1:i2))
```

```
ptoeq_F = -2.5000
```

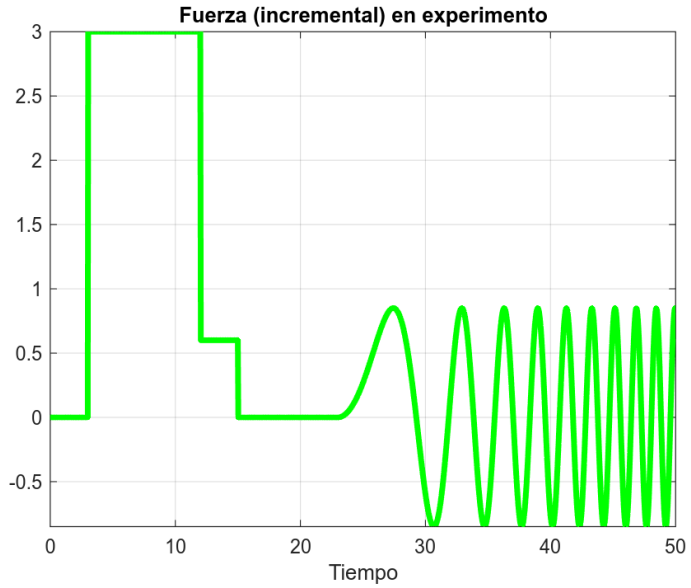
Paso a variables incrementales

Trasladamos origen de tiempos, de fuerzas y de salida (variables "incrementales"):

```
T=datos.T(i1:end)-datos.T(i1); %pongo a cero el cronómetro
F=datos.F(i1:end)-ptoeq_F; %pongo el "cero" de fuerza
y=datos.pos(i1:end)-ptoeq_y; %pongo el "cero" de posición
plot(T,y), grid on, axis tight, title("Salida (incremental) experimental"), xlabel("Tie
```




```
plot(T,F,'g','LineWidth',3), grid on, axis tight, title("Fuerza (incremental) en experi
```



Separación "datos de ajuste" versus "datos de validación"

Para "detectar errores" sería mejor que la recogida de datos para validación del modelo la realizara "otra persona", "otro día" en una aplicación "real". Pero, bueno, aquí vamos a coger 22 segundos de ajuste y el resto de validación. La validación de modelos en control, estadística, inteligencia artificial es un mundo con muchos detalles a explorar.

```
i3=find(T>=22,1)-1;
Tajuste=T(1:i3); Yajuste=y(1:i3); Fajuste=F(1:i3);
Tvalida=T((i3+1):end)-T(i3+1);%empezamos en cero.
Yvalida=y((i3+1):end); Fvalida=F((i3+1):end);
```

Construcción y optimización índice de coste (mínimos cuadrados)

Cálculo indice mínimos cuadrados

```
J=@(params) sum((simulamodelo(Tajuste,Fajuste,params)-Yajuste).^2);
```

Pruebas con valores "al tun tun":

```
J([1 2 4 0])
```

```
ans = 42.2026
```

```
J([2.5 2.5 10 -0.1])
```

```
ans = 0.6913
```

```
J([2 2.5 10 0])
```

```
ans = 0.3506
```

```
J([2.5 1 0.5 .1])
```

```
ans = 2.6207e+03
```

Búsqueda con optimizador global (algoritmo genético, busqueda aleatoria dirigida) + local al final

Modelo 3 parámetros (lineal)

*Búsqueda sólo local

```
tic
[params_opt3,Jopt]=fmincon(@(x) J([x 0]), [2 2 2],[],[],[],[],[.1 .1 .1],[3 3 15])
```

```
Local minimum possible. Constraints satisfied.

fmincon stopped because the size of the current step is less than
the value of the step size tolerance and constraints are
satisfied to within the value of the constraint tolerance.

<stopping criteria details>
params_opt3 = 1×3
    1.6525    1.8169    8.6166
Jopt = 0.2925
```

```
toc
```

```
Elapsed time is 6.334136 seconds.
```

*Búsqueda global (ga) + refino local (por si acaso)

Usamos el "algoritmo genético" de la Global Optimization Toolbox.

```
tic
[params_opt,Jopt]=ga(@(x) J([x 0]), 3,[],[],[],[],[.1 .1 .1],[3 3 15])
```

```
Optimization terminated: average change in the fitness value less than options.FunctionTolerance.
params_opt = 1×3
    1.7485    1.8243    9.3842
Jopt = 0.1131
```

```
toc
```

```
Elapsed time is 55.568720 seconds.
```

```
tic
[params_opt3,Jopt]=fmincon(@(x) J([x 0]), params_opt,[],[],[],[],[.1 .1 .1],[3 3 15])
```

```
Local minimum possible. Constraints satisfied.

fmincon stopped because the size of the current step is less than
the value of the step size tolerance and constraints are
satisfied to within the value of the constraint tolerance.

<stopping criteria details>
params_opt3 = 1×3
    1.7485    1.8243    9.3842
Jopt = 0.1131
```

```
toc
```

```
Elapsed time is 1.011547 seconds.
```

*****Parece que "`ga`" ha acabado lo suficientemente cerca de un mínimo (al menos mínimo local) de modo que a "`fmincon`" no le ha merecido la pena seguir. Habría diferencia en problems más complejos, donde el tiempo para "`ga`" estuviera limitado y quisiéramos bajar "localmente" alrededor de lo mejor que hayamos encontrado "tirando el dado".

Probemos con 4 parámetros, a ver si merece la pena complicar el modelo más:

```
pruebaParalelo=true; %quizás ponlo "false" en tus primeras pruebas. El vídeo está así.
if(~pruebaParalelo)
    tic
    [params_opt,Jopt]=ga(J, 4,[],[],[],[], [.1 .1 .1 -1], [3 3 15 4])
```

```
Optimization terminated: average change in the fitness value less than options.FunctionTolerance.
params_opt = 1×4
    1.7186    1.8476    8.4997    2.3652
Jopt = 0.0894
```

```
    toc
```

```
Elapsed time is 55.950467 seconds.
```

NOTA: Opción de procesamiento "en paralelo" en máquinas multicore:

```
else %pruebaParalelo true
    tic
    Opt=optimoptions("ga",UseParallel=true);
    [params_opt,Jopt]=ga(J, 4,[],[],[],[], [.1 .1 .1 -1], [3 3 15 4],[],Opt)
```

```
Optimization terminated: average change in the fitness value less than options.FunctionTolerance.
params_opt = 1×4
    1.7136    1.8144    9.2074    0.4469
Jopt = 0.1024
```

```
    toc
```

```
Elapsed time is 39.428207 seconds.
```

```
end
```

*Problema: requiere "*parallel computing toolbox*" instalado; además, **la primera vez que se ejecuta tarda mucho** en cargar en memoria muchas copias de Matlab "independientes" (al menos con opciones por defecto), por lo que en problemas "fáciles" puede ser "peor el remedio que la enfermedad"... cuando ya estemos hablando de varios minutos de tiempo de ejecución de "ga" entonces podría convenir el usar procesamiento paralelo.

```
tic
[params_opt4,Jopt]=fmincon(J, params_opt,[],[],[],[], [.1 .1 .1 0], [3 3 15 6])
```

```
Local minimum possible. Constraints satisfied.

fmincon stopped because the size of the current step is less than
the value of the step size tolerance and constraints are
satisfied to within the value of the constraint tolerance.

<stopping criteria details>
params_opt4 = 1×4
    1.7136    1.8144    9.2074    0.4469
Jopt = 0.1023
```

```
toc
```

```
Elapsed time is 1.507071 seconds.
```

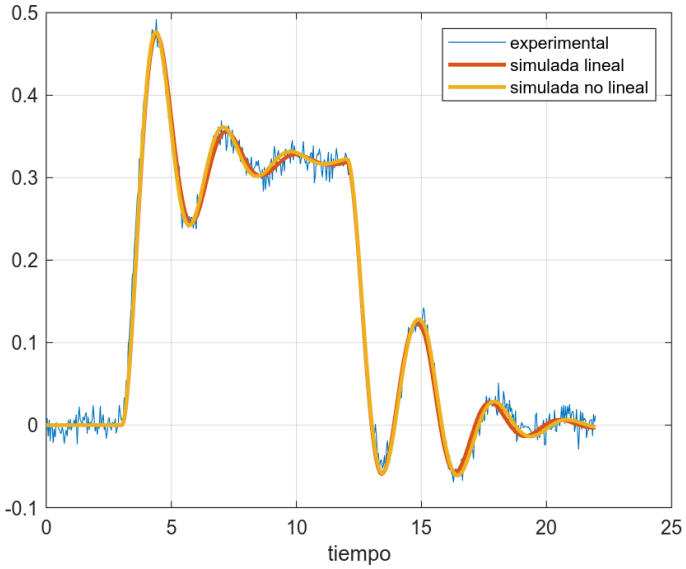
Igualmente, no parece que a "`fmincon`" le quede mucho por hacer una vez ha acabado "`ga`", al menos con opciones por defecto.

Evaluacion del modelo obtenido, conclusiones

```
td3=J([params_opt3 0]);
sqrt([td3, J(params_opt4)]/td3) %porc. desv. típica
```

```
ans = 1×2
    1.0000    0.8918
```

```
Yprueba3p=simulamodelo(Tajuste,Fajuste,[params_opt3 0]);
Yprueba=simulamodelo(Tajuste,Fajuste,params_opt4);
plot(Tajuste,Yajuste), hold on
plot(Tajuste,[Yprueba3p; Yprueba],LineWidth=2), hold off
grid on, legend("experimental","simulada lineal","simulada no lineal"), xlabel("tiempo'
```



A lo mejor la complejidad del modelo nos da sorpresas sobre datos de validación (generalizar a nuevos datos)?

```
Jval=@(params) sum((simulamodelo(Tvalida,Fvalida,params)-Yvalida).^2);
tva3=Jval([params_opt3 0])
```

```
tva3 = 0.4644
```

A veces, cuando la complejidad del modelo es "artificial" y se añaden parámetros que siguen el ruido pero no incorporan algo que "físicamente sea importante" puede ocurrir que sobre datos de validación funcione "peor" que modelos más sencillos.

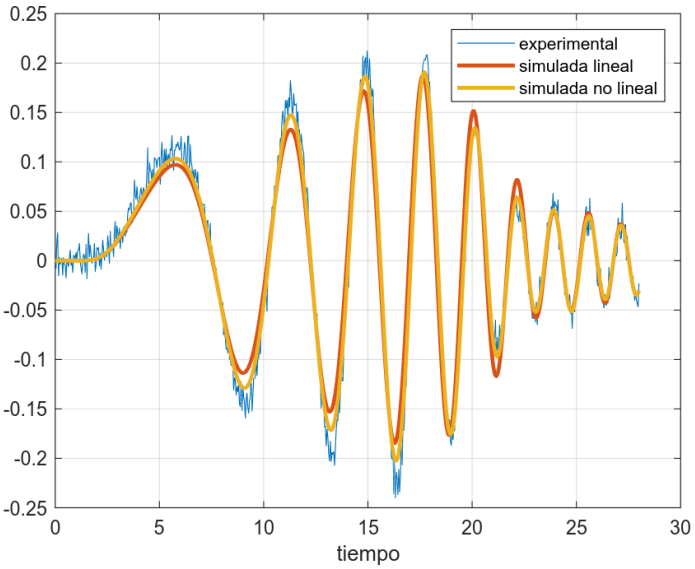
```
tva4=Jval(params_opt4)
```

```
tva4 = 0.1492
```

```
sqrt([tva3, tva4]/tva3) %porc. desv típica
```

```
ans = 1×2
    1.0000    0.5668
```

```
Yprueba3p2=simulamodelo(Tvalida,Fvalida,[params_opt3 0]);
Yprueba2=simulamodelo(Tvalida,Fvalida,params_opt4);
plot(Tvalida,Yvalida), hold on
plot(Tvalida,[Yprueba3p2; Yprueba2],LineWidth=2), hold off
grid on, legend("experimental","simulada lineal","simulada no lineal"), xlabel("tiempo"
```



En este caso sí que parece que añadir no-linealidad tiene mejoras en la calidad de la reproducción de los datos de validación.

Apéndice: funciones auxiliares (ec. estado paramétrica, simulación ode45)

```
function dxdt=ec_estado(x,F,params)
   %masa, amort., cte muelle (dos) son los 4 parámetros ajustables
   M=params(1); b=params(2); k_lin=params(3); k_nolin=params(4);
   pos=x(1); v=x(2); %vector de estado es [pos; v]
   if(abs(pos)>100) % si la simulación se inestabiliza... no seguir a infinito
       dxdt=[0;0]; %paramos el movimiento del estado.
   else
     dxdt=[ v; ...
        (-k_lin*pos-k_nolin*pos^2-b*v+F)/M ]; %ecuación de estado (no lineal)
   end
end

function Ysim=simulamodelo(tiempos,entrada,params)
   F=@(t) interp1(tiempos,entrada,t); %interpolamos en instantes intermedios no en tabl
   modelosimulacion=@(t,x) ec_estado(x,F(t),params); %formato para ode45
   [~,X]=ode45(modelosimulacion,tiempos,[0;0]);
   Ysim=X(:,1)'; %ecuación de salida: la salida es el primer estado
end
```