Identificación experimental, ejemplo masa-muelle-amortiguador utilizando algoritmo genético [ga]

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

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

Objetivo: Calcular los parámetros de un modelo masa-muelle-amortiguador lineal que "mejor" se ajustan (en sentido de mínimos cuadrados) a unos datos experimentales.

Tabla de Contenidos

Importación y preprocesado de datos 1
Selección de la zona de datos adecuada y punto de operación 2
Paso a variables incrementales 2
Separación "datos de ajuste" versus "datos de validación" 3
Construcción y optimización índice de coste (mínimos cuadrado) 4
Búsqueda con optimizador global (algoritmo genético, busqueda aleatoria dirigida) 5
Evaluacion del modelo obtenido 6
Apéndice: funciones auxiliares (ec. estado paramétrica, simulación ode45) 7

Importación y preprocesado de datos

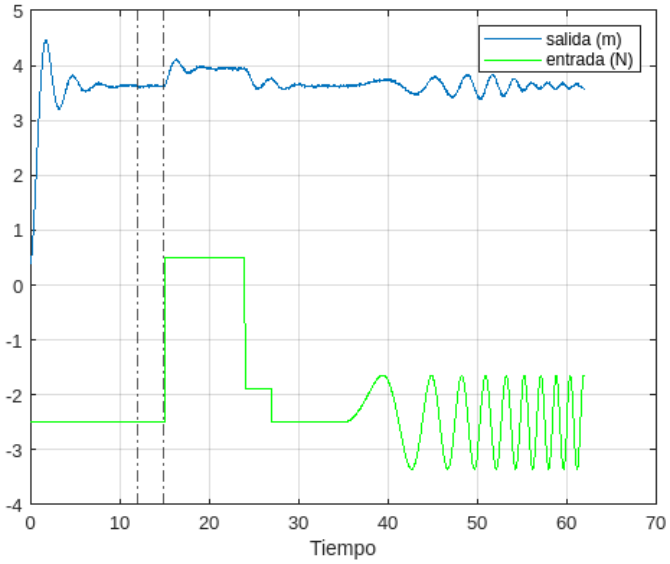
```
load experimento.mat
datos %echamos un vistazo a lo que ha cargado en memoria
```

```
datos = struct with fields:
      T: [0 0.0400 0.0800 0.1200 0.1600 0.2000 0.2400 0.2800 0.3200 0.3600 0.4000 0.4400 0.4800 0.5200 0.5
    pos: [0.4166 0.3919 0.4380 0.4753 0.5284 0.5896 0.6553 0.7601 0.8605 0.9733 1.0621 1.1756 1.3160 1.469
      F: [-2.5000 -2.5000 -2.5000 -2.5000 -2.5000 -2.5000 -2.5000 -2.5000 -2.5000 -2.5000 -2.5000 -2.5000
```

```
plot(datos.T,datos.pos,datos.T,datos.F,'g'), grid on
```

```
Warning: MATLAB has disabled some advanced graphics rendering features by switching to software
OpenGL. For more information, click here.
```

```
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
```

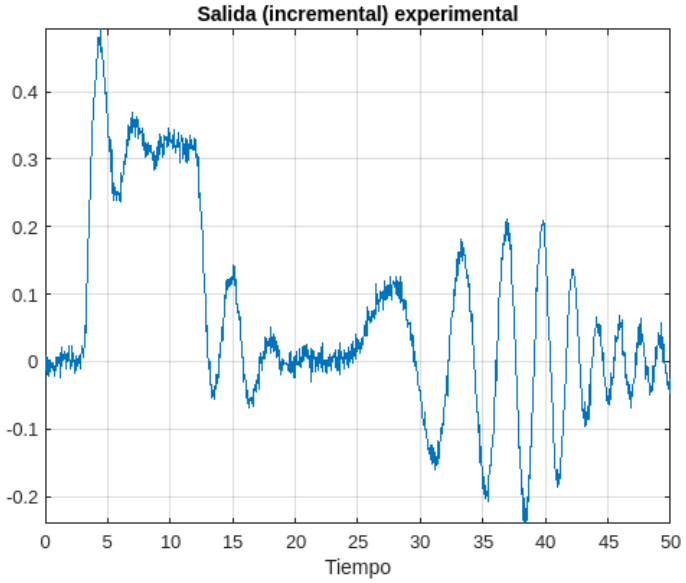
```
ptoeq_F=mean(datos.F(i1:i2))
```

```
ptoeq_F = -2.5000
```

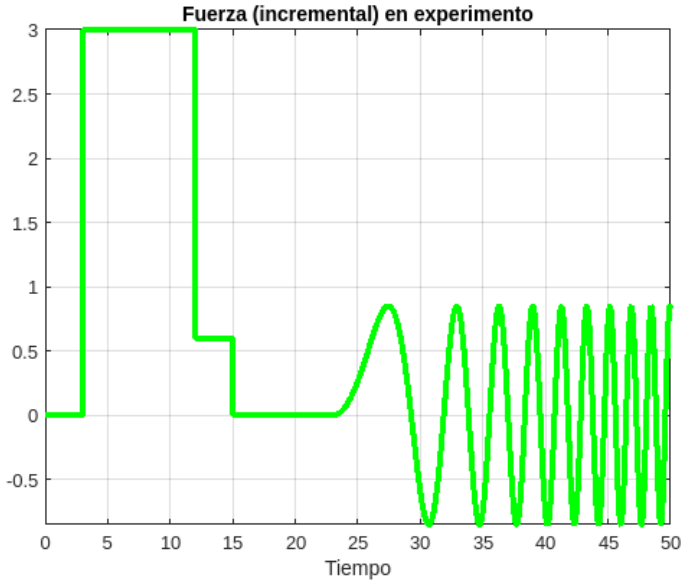
Paso a variables incrementales

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"

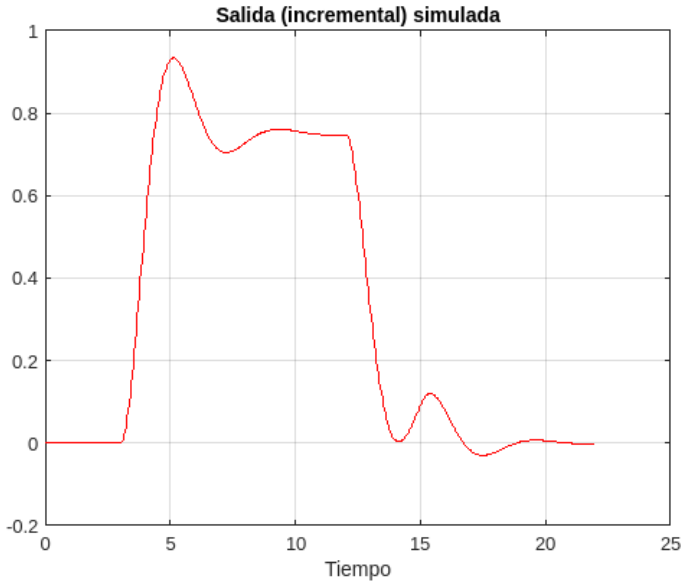
Para "detectar errores" sería mejor que la recogida de datos para validación del modelo la realizara "otra persona", "otro día" en una aplicación "real". Pero, bueno, aquí vamos a coger 22 segundos de ajuste y el resto de validación. La validación de modelos en control, estadística, inteligencia artificial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

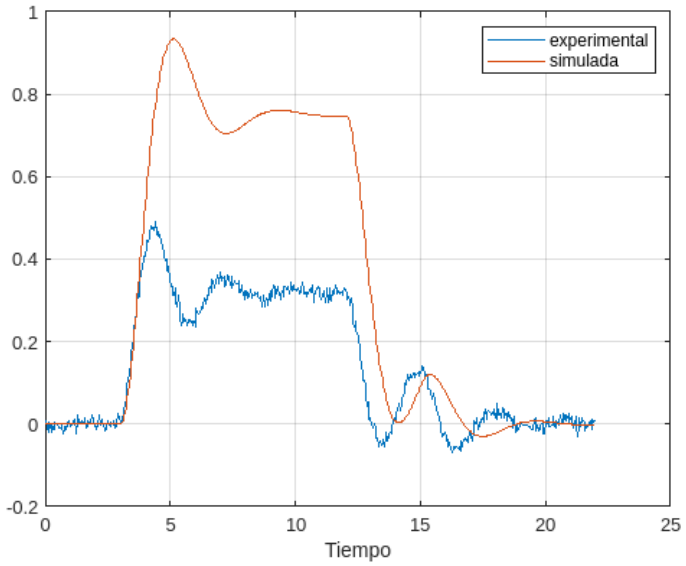
El modelo se puede simular con distintos parámetros de masa, amort., muelle:

```
Yprueba=simulamodelo(Tajuste,Fajuste,[1.5 2 4]);
plot(Tajuste,Yprueba,'r'), grid on, title("Salida (incremental) simulada"), xlabel("Tie
```



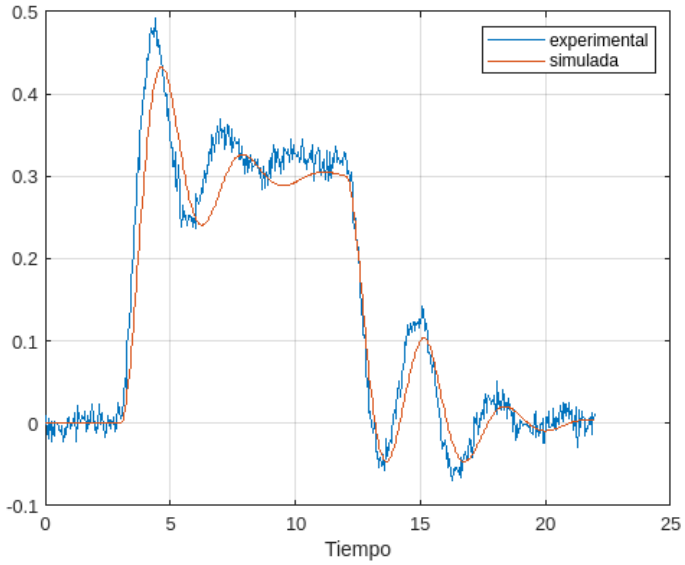
El objetivo es que "se parezca" al experimento...

```
plot(Tajuste,[Yajuste; Yprueba]), grid on, xlabel("Tiempo"), legend("experimental","sim
```



este se parece más:

```
Yprueba=simulamodelo(Tajuste,Fajuste,[2.5 2.5 10]);
plot(Tajuste,[Yajuste; Yprueba]), grid on, xlabel("Tiempo"), legend("experimental","sim
```




```
sum((Yajuste-Yprueba).^2)
```

```
ans = 0.6889
```

Cálculo indice mínimos cuadrados

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

Pruebas con valores "al tun tun":

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

```
ans = 42.2026
```

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

```
ans = 0.6889
```

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

```
ans = 0.3506
```

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

```
ans = 8.2510e+03
```

Búsqueda con optimizador global (algoritmo genético, busqueda aleatoria dirigida)

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

```
tic
[params_opt,Jopt]=ga(J,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 25.551302 seconds.
```

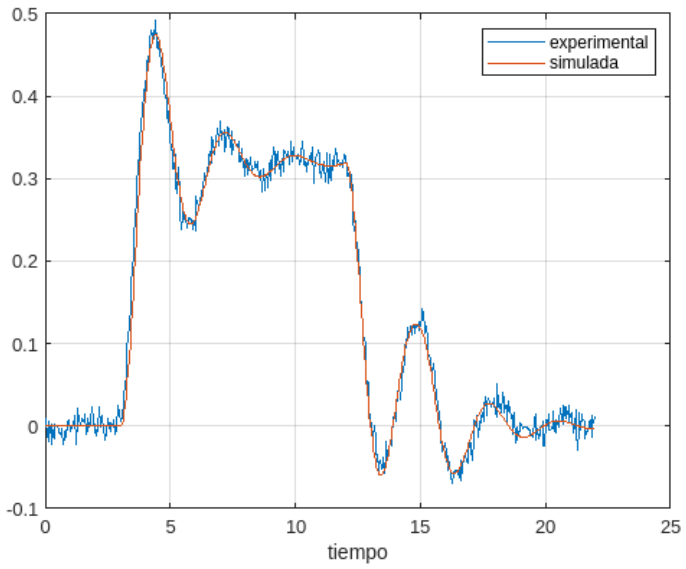
Parece que este es el mejor.

Evaluacion del modelo obtenido

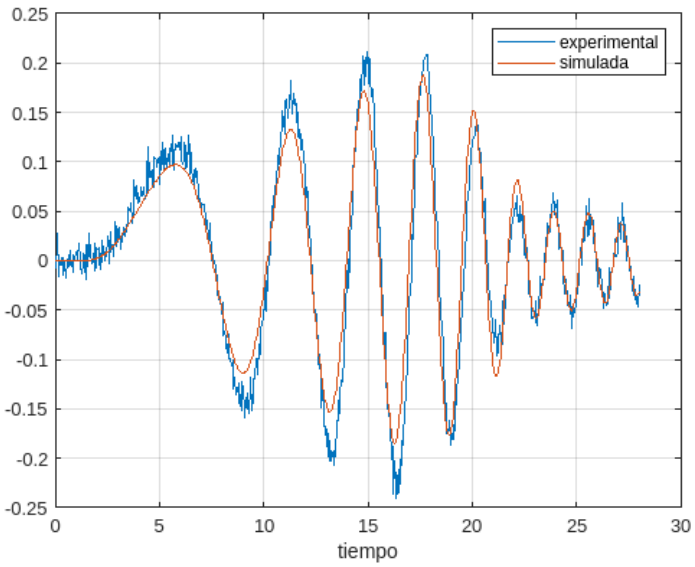
```
J(params_opt)
```

```
ans = 0.1131
```

```
Yprueba=simulamodelo(Tajuste,Fajuste,params_opt);
plot(Tajuste,[Yajuste; Yprueba]), grid on, legend("experimental","simulada"), xlabel("t
```




```
Yprueba2=simulamodelo(Tvalida,Fvalida,params_opt);
plot(Tvalida,[Yvalida; Yprueba2]), grid on, legend("experimental","simulada"), xlabel('
```



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

```
function dxdt=ec_estado(x,F,params)
   %masa, amort., muelle son los parámetros
   M=params(1); b=params(2); k=params(3);
   pos=x(1); v=x(2); %vector de estado es [pos; v]
   dxdt=[ v; ...
        (-k*pos-b*v+F)/M ]; %ecuación de estado (lineal)
end

function Ysim=simulamodelo(tiempos,entrada,params)
   F=@(t) interp1(tiempos,entrada,t);
   modelosimulacion=@(t,x) ec_estado(x,F(t),params);
   [~,X]=ode45(modelosimulacion,tiempos,[0;0]);
   Ysim=X(:,1)'; %ecuación de salida: la salida es el primer estado
end
```