Modelado, linealización y simulación comparativa linealizado/no-lineal de modelos de movimiento transversal de un tiovivo ("al grano")

Este código funcionó con Matlab **R2022b** (Linux)

Presentaciones en vídeo:

http://personales.upv.es/asala/YT/V/tiovl3.html

http://personales.upv.es/asala/YT/V/tiovl4.html

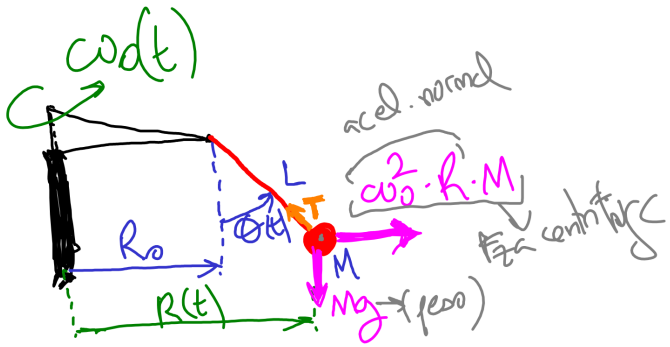
http://personales.upv.es/asala/YT/V/tiovl5.html

Objetivos: Comprender el modelado físico de un tiovivo (sólo la oscilación "lateral" del columpio), y su simulación. Linealizar el mismo y comparar la solución original con la linealizada. Aquí vamos a ir "al grano", con el mínimo código necesario. En otros vídeos se comparaban muchas opciones, pero lo que hay que intentar hacer "cuando ya se ha entendido todo" es lo de este ejemplo.

*Image from jana_lsb0 at Pixabay

Tabla de Contenidos

Modelo dinámico



0

- Parámetros constantes:

```
M=20; L=4; R_0=3; g=9.8;
```

- Entrada:

```
syms omega_0 real %velocidad angular tiovivo
omega_0nominal=pi/4; %valor nominal de diseño (para equilibrio, linealización, etc.)
```

- Otras variables en las ecuaciones del sistema:

```
syms theta  omega_1  real
```

Escribamos el modelo (con fricción) con solo 2 señales... Si las manipulaciones algebraicas a hacer con las ecuaciones son sencillas (sustituir unas cosas en otras), mejor las hacemos "a mano" y menos código y más directo todo.

Ecuación de estado normalizada:

Podemos escribir, por tanto, directamente las ecuaciones de estado

```
dEstado_dt = simplify( [ omega_1; ... %dinámica velocidad ang.
    1/(M*L^2)*(-M*g*L*sin(theta) + omega_0^2*(R_0+L*sin(theta))*L*cos(theta)*M-128*omeg
```

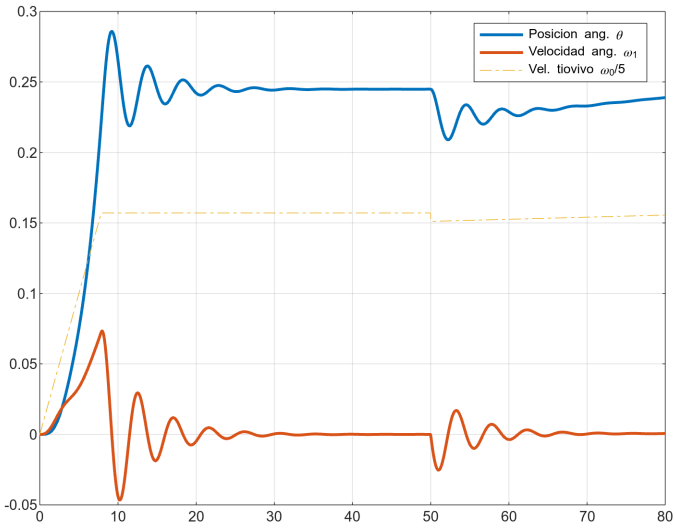
```
dEstado_dt =
```

$$\begin{pmatrix} \omega_1 \\ \frac{\cos(\theta)\ (4\sin(\theta)+3)\ {\omega_0}^2}{4} - \frac{2\,\omega_1}{5} - \frac{49\sin(\theta)}{20} \end{pmatrix}$$

```
Estado=[theta;omega_1]; %pos y vel angulares
```

Simulación ode45 del modelo dinámico no lineal

```
dEstado_num=matlabFunction(dEstado_dt,Vars={Estado,omega_0}); %traducir a función Matla
Tiempofinal=80;
x0=[0;0];
u=@(t) min(0.1*t,omega_0nominal)-0.03*(1-0.025*(t-50)).*(t>50); %entrada
opts=odeset("RelTol",1e-5,"AbsTol",1e-5);
dEstado_paraode45=@(t,VectorDeEstado) dEstado_num(VectorDeEstado,u(t));
[Tiempos,Estados]=ode45(dEstado_paraode45,[0 Tiempofinal],x0,opts);
plot(Tiempos,Estados,LineWidth=2), grid on, hold on
plot(Tiempos, u(Tiempos')/5,'-.'), hold off
legend("Posicion ang. \theta","Velocidad ang. \omega_1","Vel. tiovivo \omega_0/5",Locat
```



Linealización (sin rodeos)

Punto de funcionamiento (equilibrio)

Si la velocidad de giro del tiovivo (entrada) está en el valor nominal, podemos escribir

```
ModelEq=subs(dEstado_dt, omega_0, omega_0nominal);
PtoFuncionamientoNominal=vpasolve(ModelEq==[0;0]) %derivadas a cero
```

```
PtoFuncionamientoNominal = struct with fields:
    omega_1: 0
      theta: 0.24485189138436267811938653070954
```

Con lo que cuando la velocidad de entrada sea "parecida" al nominal, pues theta será parecido a lo de arriba. Lo gastaremos, pues, como "punto de tangencia" para linealizaciones.

***Nota:** La solución del problema no es única (hay un equilibrio "inestable" tipo "motociclista en curva" si el columpio se sujetase con una barra a compresión).

```
PtoFuncionamientoNominalAlternativo=vpasolve(ModelEq==[0;0], [0 -pi])
```

```
PtoFuncionamientoNominalAlternativo = struct with fields:
    omega_1: 0
      theta: -2.991536339633866500783606779354 3
```

```
eval(PtoFuncionamientoNominalAlternativo.theta*180/pi)
```

```
ans = -171.4024
```

Seleccionamos el de "abajo" como punto de funcionamiento.

```
theta_PF=eval(PtoFuncionamientoNominal.theta);
```

Cálculo de derivadas parciales

En otros vídeos se detallaba "mucho" la linealización, para aprender cómo hacerlo "a mano". Aquí vamos, por la vía rápida, a hacer las derivadas parciales con la Symbolic Toolbox.

```
derivsParcialsTMP=jacobian(dEstado_dt, Estado); %respecto a posicion y velocidad, matri
A=eval(subs(derivsParcialsTMP, {theta,omega_0}, {theta_PF,omega_0nominal})) %derivs. pa
```

```
A = 2×2
         0    1.0000
   -1.9447   -0.4000
```

```
derivsParcialsTMP=jacobian(dEstado_dt, omega_0); %respecto a entrada, matriz derivadas
B=eval(subs(derivsParcialsTMP, {theta,omega_0}, {theta_PF,omega_0nominal})) %derivs. pa
```

```
B = 2×1
         0
    1.5124
```

Con lo que el modelo linealizado es $\frac{dx}{dt} = Ax + Bu$, siendo $x_{2\times 1}$ el vector con los "*incrementos de posición y velocidad*", y siendo u los "incrementos de velocidad del tiovivo $u = w_0 - w_{0,PF}$.

Introducción del modelo linealizado en el "control systems toolbox"

Usualmente la respuesta ante entradas sencillas (escalón) y las propiedades de la respuesta (amplitud, frecuencia, duración de oscilaciones) en el modelo linealizado pueden calcularse "analíticamente" con fórmulas de teoría de EDO lineales, en vez de con simulación `ode45`... de

hecho, esa es la justificación más importante de la utilidad de la linealización, porque para simular con ode45 ya tendríamos el no lineal, je.

La forma normalizada de sistemas lineales en tiempo continuo es:

$$\frac{dx}{dt} = Ax + Bu$$

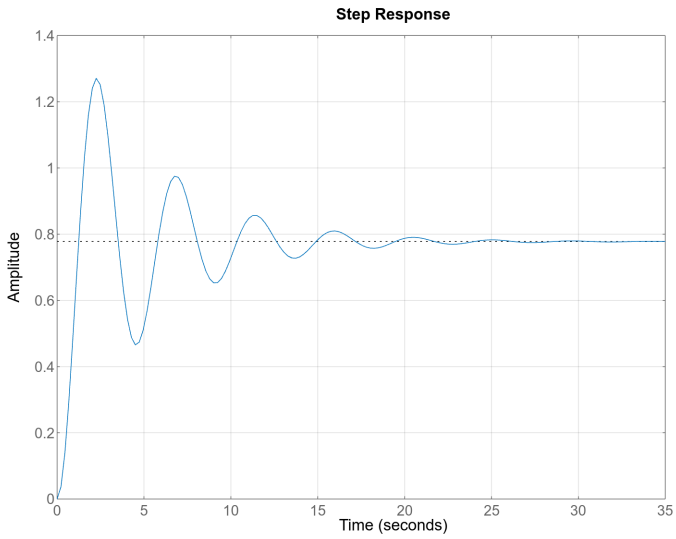
$$y = Cx + Du$$

formando las matrices A, B, C, D con los coeficientes que multiplican a las distintas variables...

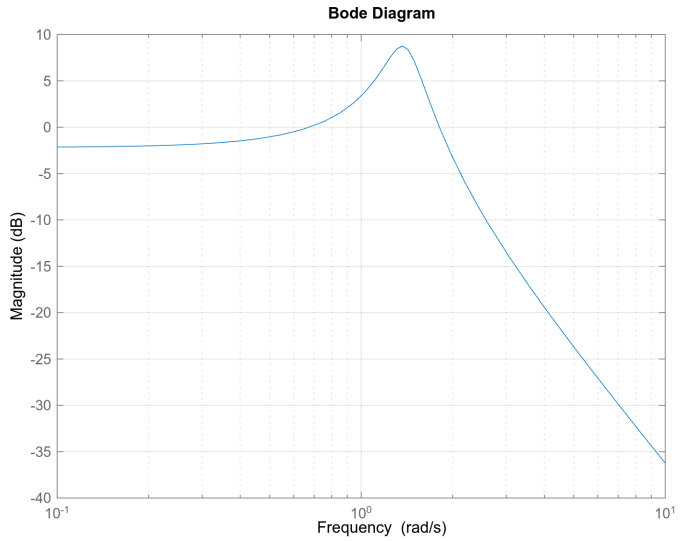
```
C=[1 0]; D=0; %sólo me interesa posición en mi aplicación
syslin=ss(A,B,C,D);
```

Simulacion del modelo linealizado

```
step(syslin), grid on %escalón "unitario"... todo es proporcional, claro, si no es unit
figure()
```



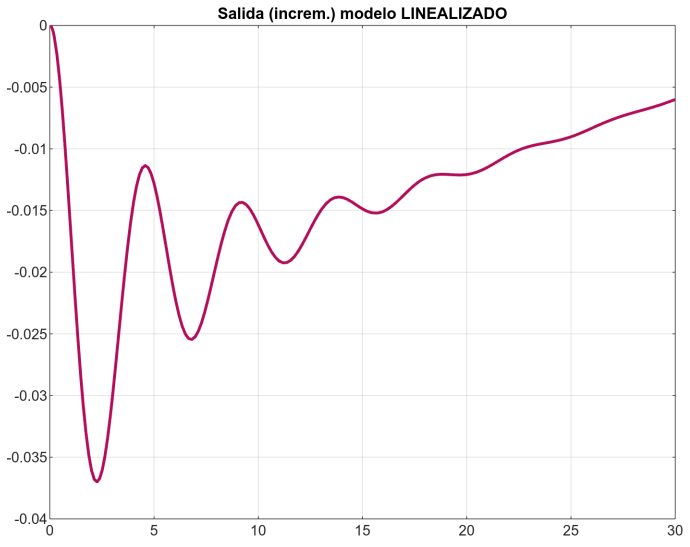

```
bodemag(syslin), grid on %respuesta ante senoides (amplitud, escala logarítmica)
```



Simulación comparada linealizado/no lineal

Para simular el modelo linealizado, usamos `ode45`, como con el no lineal (hay una rutina "`lsim`" especializada en simular sistemas lineales más rápido que `ode45`, pero no vamos a entrar en ese tema aquí, por brevedad):

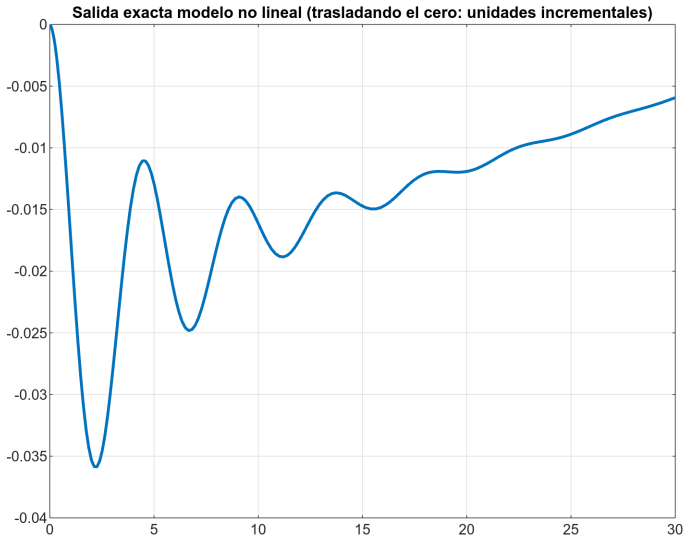
```
EcEstadoLinealizadoNum=@(x,u) A*x+B*u;
uinc=@(t) u(t+50)-omega_0nominal; %entrada INCREMENTAL
[TiemposLin,EstadosLin]=ode45(@(t,x) EcEstadoLinealizadoNum(x,uinc(t)),[0 30],[0;0],opt
plot(TiemposLin,EstadosLin(:,1),Color=[0.7 0.05 0.35],LineWidth=2), grid on, title("Sal
```



Comparación "A": en unidades incrementales

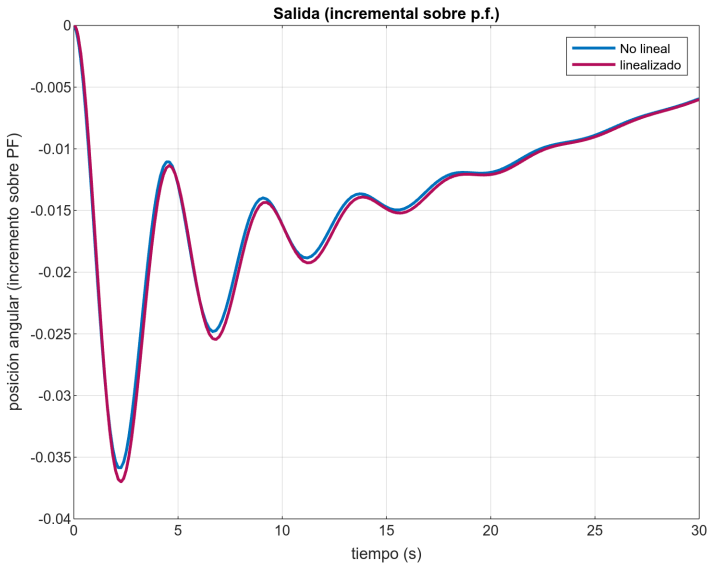
Vamos a superponerlo con el "no lineal" en incrementales

```
i1=find(Tiempos>=50,1);
plot(Tiempos(i1:end)-Tiempos(i1),Estados(i1:end,1)-theta_PF,LineWidth=2), grid on %camb
title("Salida exacta modelo no lineal (trasladando el cero: unidades incrementales)")
```



Para comparar ambos modelos, superponemos y todo se ve más claro:

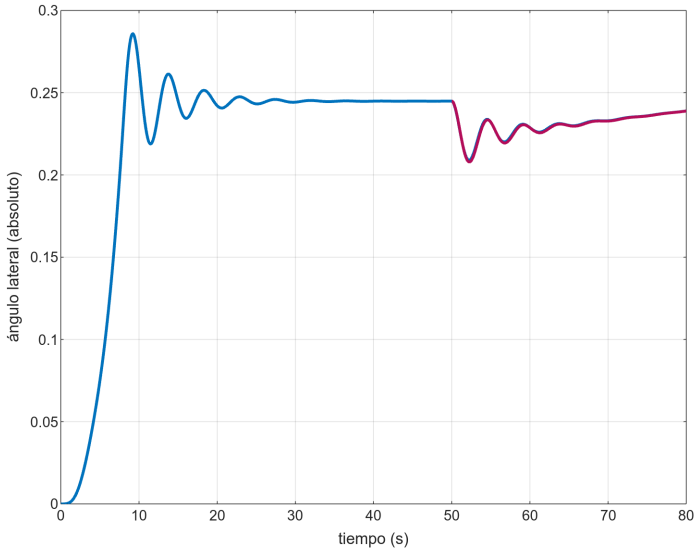
```
plot(Tiempos(i1:end)-Tiempos(i1),Estados(i1:end,1)-theta_PF,LineWidth=2), grid on %camb
hold on
plot(TiemposLin,EstadosLin(:,1),Color=[0.7 0.05 0.35],LineWidth=2) %superponemos lineal
hold off, legend("No lineal","linealizado"), title("Salida (incremental sobre p.f.)")
xlabel("tiempo (s)"), ylabel("posición angular (incremento sobre PF)")
```



Comparación "B": en unidades absolutas

Como alternativa, obviamente equivalente, sería trasladar la salida "incremental" del modelo linealizado a unidades físicas "absolutas" como las del modelo no lineal original y superponer gráficas:

```
plot(Tiempos,Estados(:,1),LineWidth=2), grid on, hold on %ode45 no lineal
plot(TiemposLin+50,EstadosLin(:,1)+theta_PF,Color=[0.7 0.05 0.35],LineWidth=2), hold of
xlabel("tiempo (s)"),ylabel("ángulo lateral (absoluto)")
```



Al final es lo mismo, excepto el origen del sistema de referencia... Si nos interesan las "pequeñas vibraciones alrededor del equilibrio", calcularemos la amplitud de las mismas en "incrementales", si nos interesa saber si se chocará con una valla al llegar a 0.3 radianas, entonces nos interesarán las unidades "absolutas".

Comentarios finales: ¿linealización por cambio de variable (inversión de no-linealidad)?

En el código de arriba se ha seguido, "sin hacerse preguntas filosóficas", la metodología estándar de linealización.

Pero, dado que la entrada siempre sale como ω_0^2, podríamos haber pensado en hacer un cambio de variable y haber llamado $v := \omega_0^2$, donde v será la nueva entrada porque si ω_0 puede tener un valor arbitrario, también lo podrá tener v (bueno, debe ser positivo).

El resultado sería un modelo:

```
syms v real
Modelo2=subs(dEstado_dt,omega_0^2,v)
```

```
Modelo2 =
```

$$\begin{pmatrix} \omega_1 \\ \frac{v\cos(\theta)\ (4\sin(\theta)+3)}{4} - \frac{49\sin(\theta)}{20} - \frac{2\,\omega_1}{5} \end{pmatrix}$$

Obviamente, seguiría siendo no lineal por los senos y cosenos de θ y por el producto $v\cos(\theta)$, pero *al menos una causa de error* (cambiar parábola por recta) *la habríamos eliminado.*

Es una **opción RECOMENDADA**, pues, seguir este camino, en muchos casos.

Obviamente, quedaría pendiente linealizar el resto de la dinámica, esto es:

```
A2=eval(subs(jacobian(Modelo2,Estado),{omega_1,theta,v},{0,theta_PF,omega_0nominal^2}))
```

```
A2 = 2×2
         0    1.0000
   -1.9447   -0.4000
```

```
B2=eval(subs(jacobian(Modelo2,v),{omega_1,theta,v},{0,theta_PF,omega_0nominal^2}))
```

```
B2 = 2×1
         0
    0.9628
```

*****El probema es que en aplicaciones de control o filtrado podrían ser necesarios bloques que hicieran ***cuadrados*** o ***raíces cuadradas***; por ejemplo, si para posicionar en cierto "ángulo deseado" calculamos que se necesita $v = 2$, ello implica $\omega_0 = \sqrt{2}$.

Esto no es problema si es implementado tecnológicamente como "*control por computador*" o "*procesado digital de señal*", pero podría serlo si la lógica debe ser realizada con "*electrónica analógica*" o "*electromecánica*" (circuitos y amplificadores lineales, palancas, ...). En un examen de "lápiz y papel" preguntad al profesor si el truco del "cambio de variable" está o no permitido en la resolución.