Vídeo-presentación disponible en
https://youtu.be/uTBtwf1Q8to
https://youtu.be/qfT0jIgaUsQ

Table of Contents

Modelado sistema engranajes y polea.

Copyright: Antonio Sala, DISA, Univ. Politecnica de Valencia

```
syms th1 th2 w1 w2 x v y dy Fmuelle Famort vy Fcuerda F12 real %salidas
syms dx dth1 dth2 dv dw1 dw2 real %veloc y aceleraciones
syms Tm real %entrada
r1=.1;r2=.2;r3=.5;J1=1;J2=1;M=2;k=30;b=3;
```

Modelo

```
%dinámica

DinEqs=[dx==v, dth1==w1, dth2==w2, dy==vy, ...
    dv==1/M*(-Fmuelle-Famort), dw2==1/J2*(F12*r2+Fcuerda*r3), ...
    dw1==1/J1*(Tm-F12*r1), Fmuelle==k*(x-y), Famort==b*(v-vy),
 Fcuerda==Fmuelle+Famort];

EngranajeyPolea=[r2*th2==r1*th1,  y==r3*th2];

Modelo=[DinEqs, EngranajeyPolea];
```

Laplace

```
syms s real

ModeloLaplace=subs(Modelo,{dx,dth1,dth2,dv,dw1,dw2,dy},
{s*x,s*th1,s*th2,s*v,s*w1,s*w2,s*y});
```

contar ecs. e incógnitas y despejar....

```
num_ecuaciones=length(ModeloLaplace)
letras=symvar(ModeloLaplace)
```

```
length(letras)
```

```
num_ecuaciones =

    12
```

```
letras =

[ F12, Famort, Fcuerda, Fmuelle, Tm, s, th1, th2, v, vy, w1, w2, x, y]
```

```
ans =

    14
```

Conocidas={s,Tm};

```
Incognitas={th1, th2, w1, w2, x, v, y, Fmuelle, Fcuerda, F12, Famort,
 vy};

solLaplace=solve(ModeloLaplace,Incognitas{:});
% vamos a enseñar todas las soluciones por nombre:
[fields(solLaplace) simplify(struct2array(solLaplace))']
```

```
ans =

[     th1, (8*Tm*(2*s^2 + 3*s + 30))/(s^2*(20*s^2 + 33*s + 330))]
[     th2, (4*Tm*(2*s^2 + 3*s + 30))/(s^2*(20*s^2 + 33*s + 330))]
[      w1,   (8*Tm*(2*s^2 + 3*s + 30))/(s*(20*s^2 + 33*s + 330))]
[      w2,   (4*Tm*(2*s^2 + 3*s + 30))/(s*(20*s^2 + 33*s + 330))]
[       x,            (6*Tm*(s + 10))/(s^2*(20*s^2 + 33*s + 330))]
[       v,              (6*Tm*(s + 10))/(s*(20*s^2 + 33*s + 330))]
[       y, (2*Tm*(2*s^2 + 3*s + 30))/(s^2*(20*s^2 + 33*s + 330))]
[ Fmuelle,                         -(120*Tm)/(20*s^2 + 33*s + 330)]
[ Fcuerda,                -(12*Tm*(s + 10))/(20*s^2 + 33*s + 330)]
[     F12,      (10*Tm*(4*s^2 + 9*s + 90))/(20*s^2 + 33*s + 330)]
[  Famort,                         -(12*Tm*s)/(20*s^2 + 33*s + 330)]
[      vy,   (2*Tm*(2*s^2 + 3*s + 30))/(s*(20*s^2 + 33*s + 330))]
```

Función de Transferencia de salidas elegidas:

FdT de la posición de la masa:

```
FdT_x=diff(solLaplace.x,Tm)
```

```
FdT_x =

(6*(s + 10))/(s^2*(20*s^2 + 33*s + 330))
```

FdT de la tensión de la cuerda

```
FdT_Fcuerda=diff(solLaplace.Fcuerda,Tm)
```

```
FdT_Fcuerda =

-(12*(s + 10))/(20*s^2 + 33*s + 330)
```

Respuesta temporal

la tensión de la cuerda ante Tm=sen(3t) será

```
Tm_Laplace=3/(s^2+3^2);

salida_Laplace=FdT_Fcuerda*Tm_Laplace;
%obtengamos la inversa de Laplace,
%simplificada y con 3 cifras significativas
respuestatemporal=vpa(simplify(ilaplace(salida_Laplace)),3)
```

```
respuestatemporal =

0.201*cos(3.0*t) - 0.668*sin(3.0*t) - 0.201*exp(-0.825*t)*(cos(3.98*t)
 - 2.3*sin(3.98*t))
```

representación interna:

```
disp('SS primerintento')
n_ecuaciones=length(Modelo)
letras=symvar(Modelo)
length(letras)
```

```
SS primerintento

n_ecuaciones =

    12


letras =

[ F12, Famort, Fcuerda, Fmuelle, Tm, dth1, dth2, dv, dw1, dw2, dx, dy,
 th1, th2, v, vy, w1, w2, x, y]


ans =

    20
```

aparecen derivadas **dth1, dth2, dw1, dw2, dx, dv, dy**.

Las variables de estado son los argumentos de esas derivadas: **th1, th2, w1, w2, x, v, y**. El sistema, por tanto, debería resultar de **orden 7**.

Estados y entradas son conocidas: Conocidas={th1,th2,w1,w2,x,v,y,Tm}; Total 8 letras!!!!

El resto de letras son:

```
Incognitas2={dth1, dth2, dw1, dw2, dx, dv, dy, Fmuelle, Fcuerda, F12,
 Famort, vy};
```

El número de incóngnitas es igual al de ecuaciones:

```
length(Incognitas2)
```

```
ans =

    12
```

por lo que podemos intentar resolver el sistema de ecuaciones Modelo:

```
sol=solve(Modelo,Incognitas2{:}) %NO FUNCIONA es algebraic-
differential eq.
sol.dth1 %devuelve <empty sym>
```

```
sol =

  struct with fields:

       dth1: [0×1 sym]
       dth2: [0×1 sym]
        dw1: [0×1 sym]
        dw2: [0×1 sym]
         dx: [0×1 sym]
         dv: [0×1 sym]
         dy: [0×1 sym]
    Fmuelle: [0×1 sym]
    Fcuerda: [0×1 sym]
        F12: [0×1 sym]
     Famort: [0×1 sym]
         vy: [0×1 sym]


ans =

Empty sym: 0-by-1
```

¿¿¿Por qué no funciona???

realmente th2 NO es variable de estado, ya que depende de th1... tampoco "y" es variable de estado dado que también es función directa de th2.

En efecto, el sistema sólo tiene **2 grados de libertad** mecánicos y debería ser de **orden 4**.

Por tanto, hay que eliminar **th2**, **w2** e **y** de las "conocidas" añadiendo 3 ecuaciones más y **despejándolas de th1 y w1** que son los "verdaderos" estados... Consideramos la restricción cinemática del engranaje y **añadimos sus primera y segunda derivadas**, así como la **primera derivada** de la ecuación de la polea.

```
disp('SS segundo intento')
EngranajeyPoleaDerivs=[r2*w2==r1*w1,r2*dw2==r1*dw1, ...
    dy==r3*dth2];

ModeloAmpliado=[Modelo EngranajeyPoleaDerivs];

EstadosQueNoLoSon={th2,w2,y};

Incognitas3={dth1, dth2, dw1, dw2, dx, dv, dy, Fmuelle, Fcuerda,
 Famort, F12, vy, EstadosQueNoLoSon{:}};


sol=solve(ModeloAmpliado,Incognitas3{:});
%vamos a enseñar todas las soluciones en columna:
[fields(sol) struct2array(sol)']
```

```
SS segundo intento

ans =

[    dth1,                                                  w1]
[    dth2,                                                w1/2]
[     dw1,  (4*Tm)/5 - (3*th1)/2 + (3*v)/5 - (3*w1)/20 + 6*x]
[     dw2, (2*Tm)/5 - (3*th1)/4 + (3*v)/10 - (3*w1)/40 + 3*x]
[      dx,                                                 v]
[      dv,            (15*th1)/4 - (3*v)/2 + (3*w1)/8 - 15*x]
[      dy,                                              w1/4]
[ Fmuelle,                                 30*x - (15*th1)/2]
[ Fcuerda,               3*v - (15*th1)/2 - (3*w1)/4 + 30*x]
[  Famort,                                  3*v - (3*w1)/4]
[     F12,             2*Tm + 15*th1 - 6*v + (3*w1)/2 - 60*x]
[      vy,                                              w1/4]
[     th2,                                             th1/2]
[      w2,                                              w1/2]
[       y,                                             th1/4]
```

En este caso no ha habido ningún problema al resolver, dado que ya no hemos considerado variables "dependientes" de otras como "independientes" que era el error de la primera vez.

Representación normalizada

representación interna linealizada: **derivadas parciales * incrementos**. Realmente ya es lineal. Las derivadas parciales dan el coeficiente que multiplica... resulta "conveniente" para acortar el código.

Para calcular derivadas parciales en bloque: **jacobian**. **eval** transforma de simbólico a numérico.

```
ladoderecho_ecuacion_estado=[sol.dth1;sol.dw1;sol.dx;sol.dv];
vector_estado=[th1,w1,x,v];
```

Calculemos matriz estado-estado A:

```
A=eval(jacobian(ladoderecho_ecuacion_estado,vector_estado))
```

```
A =

         0    1.0000         0         0
   -1.5000   -0.1500    6.0000    0.6000
         0         0         0    1.0000
    3.7500    0.3750  -15.0000   -1.5000
```

matriz estado-entrada B:

```
vector_entradas=[Tm];
B=eval(jacobian(ladoderecho_ecuacion_estado,vector_entradas))
```

```
B =

         0
    0.8000
         0
         0
```

salidas: posición de la masa y tensión cuerda (arbitrarias)

```
ec_salida=[x,sol.Fcuerda];
```

matriz salida-estado:

```
C=eval(jacobian(ec_salida,vector_estado))
```

```
C =

         0         0    1.0000         0
   -7.5000   -0.7500   30.0000    3.0000
```

matriz salida-entrada:

```
D=eval(jacobian(ec_salida,vector_entradas))
```

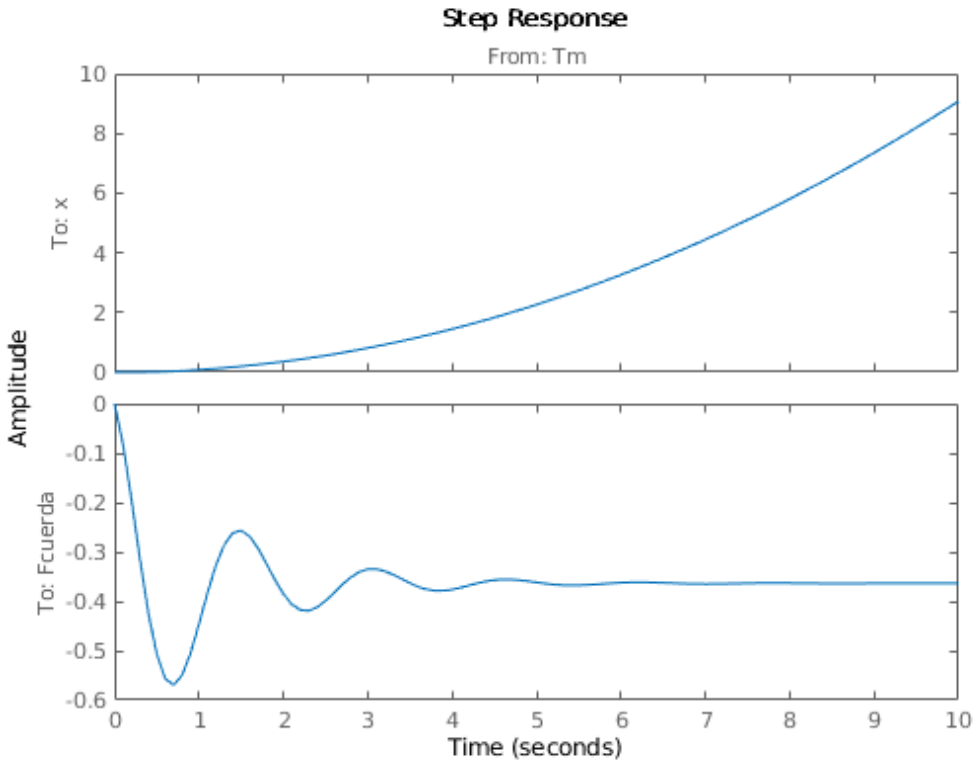
```
D =

     0
     0
```

simulamos ante un par de entrada de 1 Nm durante 10 segundos.

```
sistemamecanico=ss(A,B,C,D);
```

```
sistemamecanico.InputName={'Tm'};
sistemamecanico.OutputName={'x','Fcuerda'};
step(sistemamecanico,10)
```



analisis de la respuesta de la tensión cuerda:

FdT entre (salida **2**, entrada **1**), **minreal** simplifica numéricamente.

```
tf2=minreal(tf(sistemamecanico(2,1)))
```

```
tf2 =

  From input "Tm" to output "Fcuerda":
                   -0.6 s^3 - 6 s^2
  ---------------------------------------------------
  s^4 + 1.65 s^3 + 16.5 s^2 - 9.39e-15 s - 4.622e-15

Continuous-time transfer function.
```

polos:

```
rr=roots(tf2.den{:})
```

```
rr =
```

```
  -0.8250 + 3.9774i
  -0.8250 - 3.9774i
  -0.0000 + 0.0000i
   0.0000 + 0.0000i
```

tiempo de establecimiento:

```
te95=-3/real(rr(1))
```

```
te95 =

    3.6364
```

Frecuencia propia de oscilaciones:

```
woscil_propia=imag(rr(1))
```

```
woscil_propia =

    3.9774
```

Ganancia estática (salida en equilibrio cuando entrada vale 1 constante)

```
gan=dcgain(tf2)
```

```
gan =

     0
```

sobreoscilación, la obtenemos examinando la gráfica antes escalón unidad:

```
valorfinal=gan;
valormaximo=-.57;
valorinicial=0;

sobreoscilacion_tanto_x_uno=(valormaximo-valorfinal)/(valorfinal-
valorinicial)
```

```
sobreoscilacion_tanto_x_uno =

  -Inf
```

Published with MATLAB® R2019a