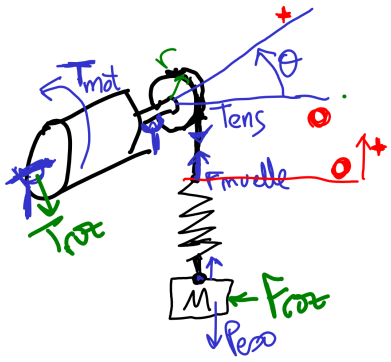
Modelado de un sistema motor-polea-muelle-masa, forma normalizada y simulación

Este código ejecutó en Matlab **R2024b**

Objetivos: modelar y simular el sistema mecánico de la figura inferior.



Presentaciones en vídeo:

https://personales.upv.es/asala/YT/V/mpmm1.html [modelado físico]

https://personales.upv.es/asala/YT/V/mpmm2.html [forma normalizada, sin Matlab]

https://personales.upv.es/asala/YT/V/mpmm3.html [forma normalizada, Symbolic toolbox, Matlab]

https://personales.upv.es/asala/YT/V/mpmm4.html [análisis de equilibrio, simulación ode45]

Tabla de Contenidos

Ecuaciones del modelado dinámico NO normalizadas

Parámetros constantes

Podemos dejarlos "en letra", pero les voy a dar valores numéricos para luego hacer simulaciones y cálculos.

```
Masa=2; Inercia=1; k_muelle=10; r=0.25; g=9.8;
l_natural=0.5;
coef_fric_rodamientos=2; coef_fric_aire=0.3;
```

Variables de entrada (el resto serán incógnitas)

```
syms T_motor      real
Entradas=[T_motor];
n_entradas=length(Entradas)
```

```
n_entradas = 1
```

(1) Ecuaciones elementales

- Masa que se mueve (traslación):

```
syms p dpdt v dvdt F_result     real
Modelo = [dpdt == v;
    dvdt == 1/Masa*F_result    ];
```

- Sólido que gira:

```
syms theta dthetadt omega domegadt T_result     real
Modelo=[Modelo;
    dthetadt == omega;
    domegadt == T_result/Inercia   ]; %añadimos
```

- Muelle:

```
syms F_muelle longitud_muelle       real
Modelo=[Modelo;
    F_muelle == k_muelle*(longitud_muelle-l_natural) ];
```

- Una polea:

```
syms Tension_Cuerda Par_Cuerda      real
Modelo = [Modelo;
    Par_Cuerda == r*Tension_Cuerda   ];
```

- Rozamiento en el giro de motor y polea, y rozamiento con aire

```
syms T_rozamiento  F_rozamiento       real
Modelo=[Modelo;
    T_rozamiento == coef_fric_rodamientos*omega;
    F_rozamiento == coef_fric_aire * v    ];
```

(2) Balances

```
Modelo = [Modelo;
    T_result == T_motor - Par_Cuerda - T_rozamiento;
    Tension_Cuerda == F_muelle;
    F_result == F_muelle - Masa*g - F_rozamiento;
    longitud_muelle == r*theta - p ];
Modelo
```

```
Modelo =
```

$$\begin{pmatrix} \mathrm{dpdt} = v \\ \mathrm{dvdt} = \frac{F_{\mathrm{result}}}{2} \\ \mathrm{dthetadt} = \omega \\ \mathrm{domegadt} = T_{\mathrm{result}} \\ F_{\mathrm{muelle}} = 10\,\mathrm{longitud}_{\mathrm{muelle}} - 5 \\ \mathrm{Par}_{\mathrm{Cuerda}} = \frac{\mathrm{Tension}_{\mathrm{Cuerda}}}{4} \\ T_{\mathrm{rozamiento}} = 2\,\omega \\ F_{\mathrm{rozamiento}} = \frac{3\,v}{10} \\ T_{\mathrm{result}} = T_{\mathrm{motor}} - \mathrm{Par}_{\mathrm{Cuerda}} - T_{\mathrm{rozamiento}} \\ \mathrm{Tension}_{\mathrm{Cuerda}} = F_{\mathrm{muelle}} \\ F_{\mathrm{result}} = F_{\mathrm{muelle}} - F_{\mathrm{rozamiento}} - \frac{98}{5} \\ \mathrm{longitud}_{\mathrm{muelle}} = \frac{\theta}{4} - p \end{pmatrix}$$

(3) Contar Ecuaciones e Incógnitas, para que sean iguales en número

```
N_ecuaciones=length(Modelo)
```

```
N_ecuaciones = 12
```

```
Letras=symvar(Modelo)'
```

```
Letras =
```

$$\begin{pmatrix} F_{\text{muelle}} \\ F_{\text{result}} \\ F_{\text{rozamiento}} \\ \text{Par}_{\text{Cuerda}} \\ T_{\text{motor}} \\ T_{\text{result}} \\ T_{\text{rozamiento}} \\ \text{Tension}_{\text{Cuerda}} \\ \text{domegadt} \\ \text{dpdt} \\ \text{dthetadt} \\ \text{dvdt} \\ \text{longitud}_{\text{muelle}} \\ \omega \\ p \\ \theta \\ v \end{pmatrix}$$

```
length(Letras)
```

```
ans = 17
```

NOTA: El modelo está completo porque, aunque son símbolos diferentes para Matlab, realmente:

- "v" y "dvdt" se refieren a la misma incógnita "física" velocidad,
- "p" y "dpdt" también,
- "theta" y "dthetadt" también,
- "omega" y "domegadt" también.

O sea, ocho "syms" que son realmente cuatro "incógnitas" sobre variables del sistema físico (estados).

```
n_incognitas=length(Letras)-4-n_entradas
```

```
n_incognitas = 12
```

```
if N_ecuaciones == n_incognitas
    disp("Modelo BIEN PLANTEADO, COMPLETO")
else
    error("No puedo continuar, el modelo no es correcto")
end
```

```
Modelo BIEN PLANTEADO, COMPLETO
```

Las cuentas coinciden, ¡bien!: **EL MODELO ESTÁ COMPLETO** (no falta ningún fenómeno físico para ser "*resoluble*" por los matemáticos, o "simulable" por métodos numéricos).

(4) Revisión final, signos...

Bueno, ya lo he revisado, je... Están bien!.

***NOTA 1:** Los signos están pensados en sistema de referencia de "desplazamiento positivo hacia **arriba**", "giro positivo **antihorario**".

***NOTA 2:** una cuerda no puede trabajar a compresión, por tanto `Tension_Cuerda` no puede cambiar de signo (positivo o negativo según sistema de referencia, en este caso según gráfica, `Tension_Cuerda` debe ser positiva, de modo que el muelle hará "cero" fuerza si la fórmula de su fuerza resulta negativa... Este tipo de modelos con "cambios estructurales" y ecuaciones con máximos/mínimos, colisiones entre objetos, etc. quedan fuera de los objetivos de la asignatura SAU.

Ec. de estado en forma NORMALIZADA

Debemos despejar únicamente "la derivada de las cosas", aunque Matlab despeja "TODO" lo que se pueda despejar, lo que requerirá enumerar todas esas letras explícitamente.

Separemos las ecuaciones "estáticas" de las "dinámicas" en el modelo

```
ModeloParteEstatica=Modelo(5:end)
```

```
ModeloParteEstatica =
```

$$\begin{pmatrix} F_{\text{muelle}} = 10\,\text{longitud}_{\text{muelle}} - 5 \\ \text{Par}_{\text{Cuerda}} = \frac{\text{Tension}_{\text{Cuerda}}}{4} \\ T_{\text{rozamiento}} = 2\,\omega \\ F_{\text{rozamiento}} = \frac{3\,v}{10} \\ T_{\text{result}} = T_{\text{motor}} - \text{Par}_{\text{Cuerda}} - T_{\text{rozamiento}} \\ \text{Tension}_{\text{Cuerda}} = F_{\text{muelle}} \\ F_{\text{result}} = F_{\text{muelle}} - F_{\text{rozamiento}} - \frac{98}{5} \\ \text{longitud}_{\text{muelle}} = \frac{\theta}{4} - p \end{pmatrix}$$

```
EcuacionesDeEstadoNoNormalizadas=Modelo(1:4)
```

```
EcuacionesDeEstadoNoNormalizadas =
```

$$\begin{pmatrix} \text{dpdt} = v \\ \text{dvdt} = \frac{F_{\text{result}}}{2} \\ \text{dthetadt} = \omega \\ \text{domegadt} = T_{\text{result}} \end{pmatrix}$$

La forma "normalizada" será de orden 4:

```
VectorDeEstados=[p; v; theta; omega];
```

Forma 1, la que hay que hacer "en el examen"

Debemos ser capaces de despejar "todo lo que no es estado ni entrada" en función de estados y entradas, manipulando las ecuaciones "sin derivadas":

```
VariablesAEliminar= ...

[F_muelle,F_result,Par_Cuerda,Tension_Cuerda,T_result,longitud_muelle,T_rozam
iento,F_rozamiento];
length(VariablesAEliminar)
```

```
ans = 8
```

```
length(ModeloParteEstatica)
```

```
ans = 8
```

Todo cuadra, "¡bingo!"

```
solVarsEliminar=solve(ModeloParteEstatica,VariablesAEliminar)
```

```
solVarsEliminar = struct with fields:
           F_muelle: (5*theta)/2 - 10*p - 5
           F_result: (5*theta)/2 - 10*p - (3*v)/10 - 123/5
         Par_Cuerda: (5*theta)/8 - (5*p)/2 - 5/4
     Tension_Cuerda: (5*theta)/2 - 10*p - 5
           T_result: T_motor - 2*omega + (5*p)/2 - (5*theta)/8 + 5/4
    longitud_muelle: theta/4 - p
       T_rozamiento: 2*omega
       F_rozamiento: (3*v)/10
```

Por lo que ya podemos sustituir eso en el lado derecho de las ecuaciones de estado:

```
EcuacionesDeEstadoNormalizadas=subs(EcuacionesDeEstadoNoNormalizadas,solVarsE
liminar);
vpa(EcuacionesDeEstadoNormalizadas)
```

```
ans =
```

$$\begin{pmatrix} \mathrm{dpdt} = v \\ \mathrm{dvdt} = 1.25\,\theta - 5.0\,p - 0.15\,v - 12.3 \\ \mathrm{dthetadt} = \omega \\ \mathrm{domegadt} = T_{\mathrm{motor}} - 2.0\,\omega + 2.5\,p - 0.625\,\theta + 1.25 \end{pmatrix}$$

En la variable "sol" están las **ecuaciones de "salida"** si alguna de las variables "eliminadas" fuera de interés para la aplicación tecnológica concreta.

Forma 2, sin separar las ecuaciones con/sin d·/dt, diciendo "Despeja las derivadas" a Matlab

```
QuitoEntradasyEstadosDeLasLetras= ...

[F_muelle,F_result,Par_Cuerda,Tension_Cuerda,T_result,T_rozamiento,F_rozamien
to,domegadt,dpdt,dthetadt,dvdt,longitud_muelle];
sol_forma2=solve(Modelo,QuitoEntradasyEstadosDeLasLetras);
```

```
VectorDeEstados'
```

```
ans = (p   v   θ   ω)
```

```
EcsEstado=[sol_forma2.dpdt;
    sol_forma2.dvdt;
    sol_forma2.dthetadt;
    sol_forma2.domegadt    ]; %en el mismo orden que vector de estados, ojo!
vpa(EcsEstado)
```

```
ans =
```

$$\begin{pmatrix} v \\ 1.25\,\theta - 5.0\,p - 0.15\,v - 12.3 \\ \omega \\ T_{\text{motor}} - 2.0\,\omega + 2.5\,p - 0.625\,\theta + 1.25 \end{pmatrix}$$

Ejemplo análisis del sistema: equilibrio

El equilibrio ("**estática**") se alcanzará cuando la fuerza del muelle sea igual al peso y esa `fuerza*radio` sea igual al par `Tmotor`.

```
Peso=Masa*g
```

```
Peso = 19.6000
```

```
T_motor_equilibrio=Peso*r %par producido por el peso en la polea
```

```
T_motor_equilibrio = 4.9000
```

Cuando esté todo en equilibrio, las variables se mantendrán constantes... y sus derivadas serán cero:

```
PtoEquilib0=solve(EcsEstado==0,[VectorDeEstados; T_motor])
```

```
PtoEquilib0 = struct with fields:
          p: -123/50
          v: 0
      theta: 0
      omega: 0
    T_motor: 49/10
```

Matlab encuentra un punto de equilibrio... pero el sistema tiene equilibrio indiferente y hay "infinitos" puntos de equilibrio, NO hay que fiarse de las máquinas. Estamos resolviendo 4 ecuaciones con 5 incógnitas, ¡OJO!.

Si añadimos, por ejemplo, posición angular prefijada, resulta un equilibrio diferente:

```
PtoEquilib1=solve([EcsEstado==0;theta==1],[VectorDeEstados;T_motor])
```

```
PtoEquilib1 = struct with fields:
          p: -221/100
          v: 0
      theta: 1
```

```
      omega: 0
    T_motor: 49/10
```

SIMULACIÓN del modelo resultante

Expresamos el modelo en forma "numérica" y no como "objeto simbólico": no es lo mismo el "objeto simbólico con un carácter "2", otro carácter "+" y otro carácter "2" que el "número en coma flotante 4.0000".

```
EcsEstadoNUM=matlabFunction(EcsEstado,Vars={VectorDeEstados,Entradas});
```

Por supuesto, si hemos hecho los cálculos en "*Lápiz y Papel*" y no tenemos "expresiones simbólicas", entonces tendríamos que teclear en "código Matlab" una función como sigue:

```
EcsEstadoNUM_LapizYPapel=@(p,v,theta,omega,Tmotor) ...
      [ v;
        1.25*theta-5*p-0.15*v-12.3;
        omega;
        Tmotor-2*omega+2.5*p-0.625*theta+1.25 ];
```

*Esto segundo es lo que harás en las prácticas de laboratorio donde no utilizamos la manipulación simbólica.

La simulación necesita de unos valores explícitamente introducidos de la entrada y de un estado inicial al principio de la simulación (energía/información almacenada del "pasado"):

```
ejemplo=4;
switch ejemplo %CUATRO EJEMPLOS DIFERENTES
    case 1 %La simulación no debería moverse nada
        T_motor=@(t) T_motor_equilibrio;
        p_inicial=eval(PtoEquilib0.p);
        v_inicial=eval(PtoEquilib0.v);
        theta_inicial=eval(PtoEquilib0.theta);
        omega_inicial=eval(PtoEquilib0.omega);
    case 2 %respuesta "LIBRE"
        T_motor=@(t) T_motor_equilibrio;
        p_inicial=eval(PtoEquilib0.p)-1;
        v_inicial=eval(PtoEquilib0.v)+0;
        theta_inicial=eval(PtoEquilib0.theta)+0;
        omega_inicial=eval(PtoEquilib0.omega)+0;
    case 3%respuesta "FORZADA" desde cond.inic. de equilibrio
        T_motor=@(t) T_motor_equilibrio + 0.2*(t<20) + 3*sin(7*t);
        p_inicial=eval(PtoEquilib0.p);
        v_inicial=eval(PtoEquilib0.v);
        theta_inicial=eval(PtoEquilib0.theta);
        omega_inicial=eval(PtoEquilib0.omega)+0;
    case 4%respuesta forzada genérica desde cond.inic. arbitrarias
        T_motor=@(t) T_motor_equilibrio + 0.2*(t<20) + 3*sin(7*t);
        p_inicial=eval(PtoEquilib0.p)+1;
        v_inicial=eval(PtoEquilib0.v)+0;
        theta_inicial=eval(PtoEquilib0.theta)+0;
        omega_inicial=eval(PtoEquilib0.omega)+0;
```

```
end
CondIniciales=[p_inicial;
               v_inicial;
              -theta_inicial;
               omega_inicial];
```

Método Euler

Cuanta más exactitud queramos, más coste computacional tendremos.

```
PasoDeIntegracion=0.01;
```

Ya podemos hacer la simulación (integración numérica) por ejemplo por Euler (lo más sencillo posible):

```
[TiemposSim,EstadosSim]=...
    odeEuler(@(t,x) EcsEstadoNUM(x,T_motor(t)), [0 30], CondIniciales,
PasoDeIntegracion);
size(TiemposSim)
```

```
ans = 1×2
        3001           1
```

```
size(EstadosSim)
```

```
ans = 1×2
        3001           4
```

Método ode45 (Runge-Kuta)

En principio este método es mejor... vamos a "machacar" los resultados anteriores.

Cuanta más exactitud queramos, más coste computacional tendremos:

```
opts=odeset('RelTol',1e-5,'AbsTol',1e-5);
```

Ya podemos hacer la simulación (integración numérica) por ejemplo por Runge-Kuta RK45:

```
[TiemposSim,EstadosSim]=...
    ode45(@(t,x) EcsEstadoNUM(x,T_motor(t)), [0 50], CondIniciales, opts);
```

Gráficas de resultados

Analicemos y representemos el resultado:

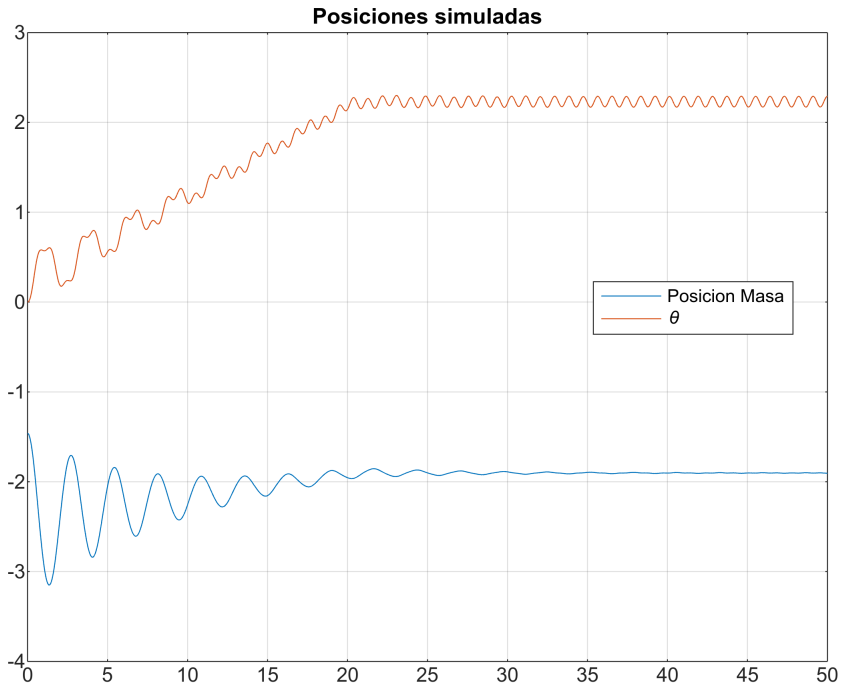
```
size(TiemposSim)
```

```
ans = 1×2
        1713           1
```

```
size(EstadosSim)
```

```
ans = 1×2
        1713           4
```

```
%plot(Tiempos,Tmotor(Tiempos)), grid on, title("Par motor (Entrada)")
plot(TiemposSim,EstadosSim(:,[1 3])), grid on
legend("Posicion Masa","\theta",Location="best"), title("Posiciones
simuladas")
```



Como otra "salida de interés" aparte de las posiciones, nos gustaría representar la tensión de la cuerda, por lo que vamos a crear una ecuación de salida:

```
EcSalidaSym=[p;theta;solVarsEliminar.Tension_Cuerda]
```

```
EcSalidaSym =
```

$$\begin{pmatrix} p \\ \theta \\ \frac{5\,\theta}{2} - 10\,p - 5 \end{pmatrix}$$

```
EcSalidaNum=matlabFunction(EcSalidaSym,Vars={VectorDeEstados})
```

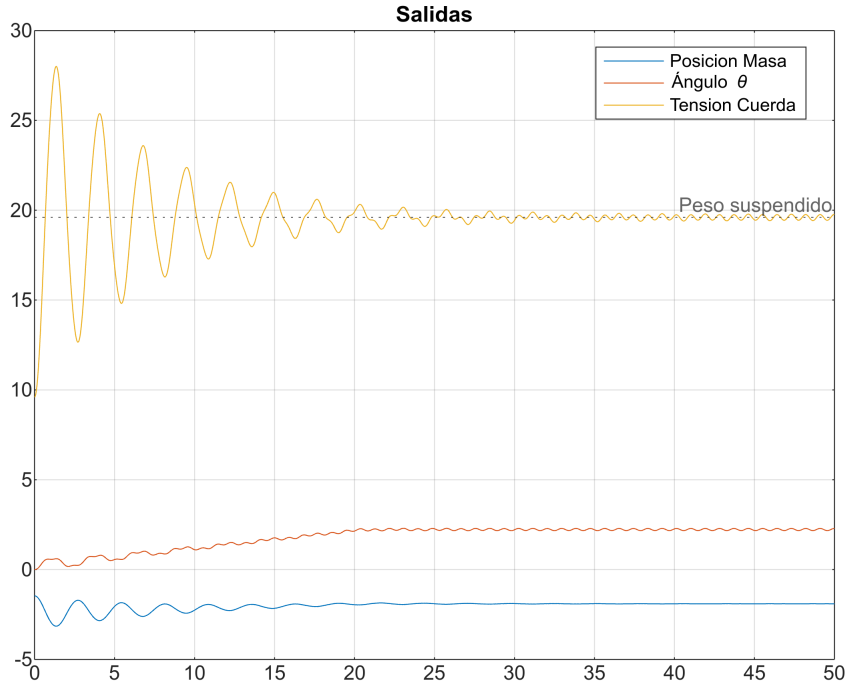
```
EcSalidaNum = function_handle with value:
    @(in1)[in1(1,:);in1(3,:);in1(1,:).*-1.0e+1+in1(3,:).*(5.0./2.0)-5.0]
```

```
SalidasPlot=zeros(length(TiemposSim),length(EcSalidaSym));
size(SalidasPlot)
```

```
ans = 1×2
        1713           3
```

```
for k=1:size(EstadosSim,1)
    SalidasPlot(k,:)=EcSalidaNum(EstadosSim(k,:)')';
end
```

```
plot(TiemposSim,SalidasPlot), title("Salidas"), grid on
yline(Peso,':',Label="Peso suspendido")
legend("Posicion Masa","Ángulo \theta","Tension Cuerda",Location="best")
```



Apéndice (func. auxiliares), Método Euler

¡NO USAR!: Esto solamente tiene una función didáctica... precisamente los códigos de simulación y animación "serios" están hechos para solventar el mal compromiso entre exactitud/eficiencia computacional de éste método "trivialmente sencillo" Euler explícito/forward.

Euler "explícito" paso fijo

```
function
[TiemposSIM,EstadosSIM]=odeEuler(dEstadodt,Tintervalo,EstadoInicial,PasoDeInt
egracion)
 TiemposSIM=Tintervalo(1):PasoDeIntegracion:Tintervalo(2);
 Npasos=length(TiemposSIM);
 OrdenSistema=length(EstadoInicial);
 EstadosSIM=zeros(OrdenSistema,Npasos);
 EstadosSIM(:,1)=EstadoInicial;
```

Esto de abajo es el bucle principal de simulación:

```
 for k=1:(Npasos-1)
    EstadosSIM(:,k+1)= EstadosSIM(:,k) ...
            + dEstadodt(TiemposSIM(k),EstadosSIM(:,k)) * PasoDeIntegracion;
 end
 EstadosSIM=EstadosSIM'; %cosmético, para coincidir con ode45 el formato.
 TiemposSIM=TiemposSIM';
```

```
end
```