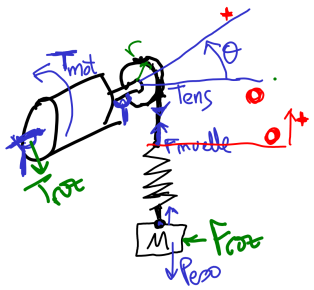
Modelado de sistema eje-polea-muelle-masa, y análisis de propiedades

Objetivos: modelar el sistema mecánico de la figura inferior como $\frac{dx}{dt} = Ax + Bu$, $y = Cx + Du$ (linealizando), y analizar sus propiedades.



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]

https://personales.upv.es/asala/YT/V/mpmm5.html [linealización, forma normalizada MATRICIAL]

https://personales.upv.es/asala/YT/V/mpmm6.html [análisis de estabilidad]

https://personales.upv.es/asala/YT/V/mpmm7.html [análisis de modos de la respuesta libre]

https://personales.upv.es/asala/YT/V/mpmm8.html [inestabilidad interna versus externa]

https://personales.upv.es/asala/YT/V/mpmm9.html [respuesta en frecuencia]

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]; %Single input
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 (o sea, fuerzas y pares resultantes, en mecánica)

```
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 %Objeto symbolic toolbox
```

```
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 =
```

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

```
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, esas letras en particular son 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_muell
e,T_rozamiento,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
,solVarsEliminar);
vpa(EcuacionesDeEstadoNormalizadas)
```

```
ans =
```

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

Nota: en la variable "sol" están todas las posibles **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
_rozamiento,domegadt,dpdt,dthetadt,dvdt,longitud_muelle];
sol_forma2=solve(Modelo,QuitoEntradasyEstadosDeLasLetras);
VectorDeEstados'
```

```
ans = (p   v   θ   ω)
```

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

```
ans =
```

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

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(DerivadasDelEstado==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"** (en la jerga de Física) 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([DerivadasDelEstado==0;theta==1],
[VectorDeEstados;T_motor])
```

```
PtoEquilib1 = struct with fields:
          p: -221/100
          v: 0
      theta: 1
      omega: 0
    T_motor: 49/10
```

NOTA: un sistema con múltiples puntos de equilibrio "indiferente" es "marginalmente inestable" y tendrá "polos en el origen", si sabes a lo que me estoy refiriendo. Se discutirá más adelante en este material.

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(DerivadasDelEstado,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=double(PtoEquilib0.p);
        v_inicial=double(PtoEquilib0.v);
        theta_inicial=double(PtoEquilib0.theta);
        omega_inicial=double(PtoEquilib0.omega);
    case 2 %respuesta "LIBRE"
        T_motor=@(t) T_motor_equilibrio;
        p_inicial=double(PtoEquilib0.p)-1;
        v_inicial=double(PtoEquilib0.v)+0;
        theta_inicial=double(PtoEquilib0.theta)+0;
        omega_inicial=double(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=double(PtoEquilib0.p);
        v_inicial=double(PtoEquilib0.v);
        theta_inicial=double(PtoEquilib0.theta);
        omega_inicial=double(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=double(PtoEquilib0.p)+1;
        v_inicial=double(PtoEquilib0.v)+0;
        theta_inicial=double(PtoEquilib0.theta)+0;
        omega_inicial=double(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:

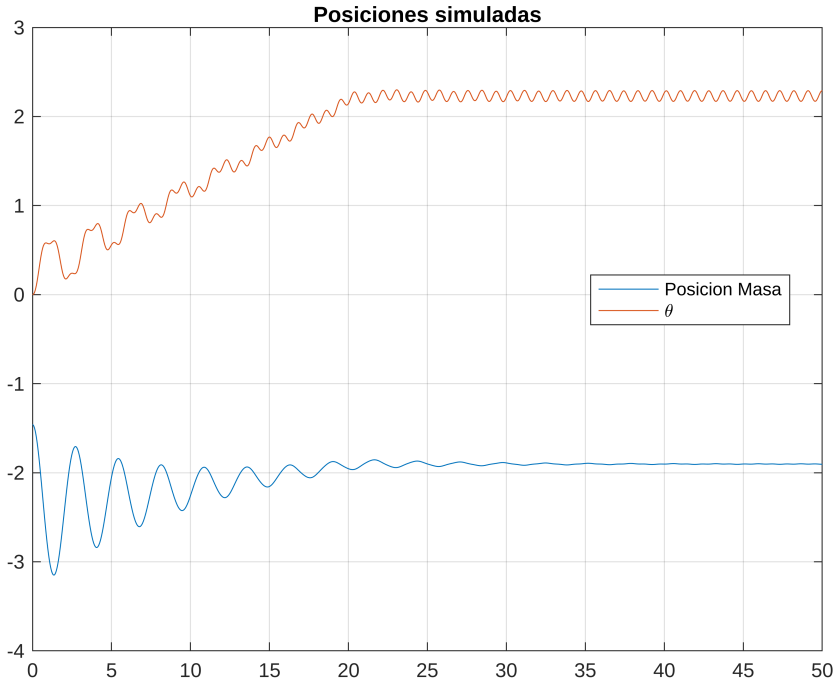
```
size(TiemposSim)
```

```
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 decir algo: las salidas es "lo que queramos que nos interese porque queremos ver qué le pasa a esa señal, porque vamos a instalar un sensor que la mide...), 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})
```

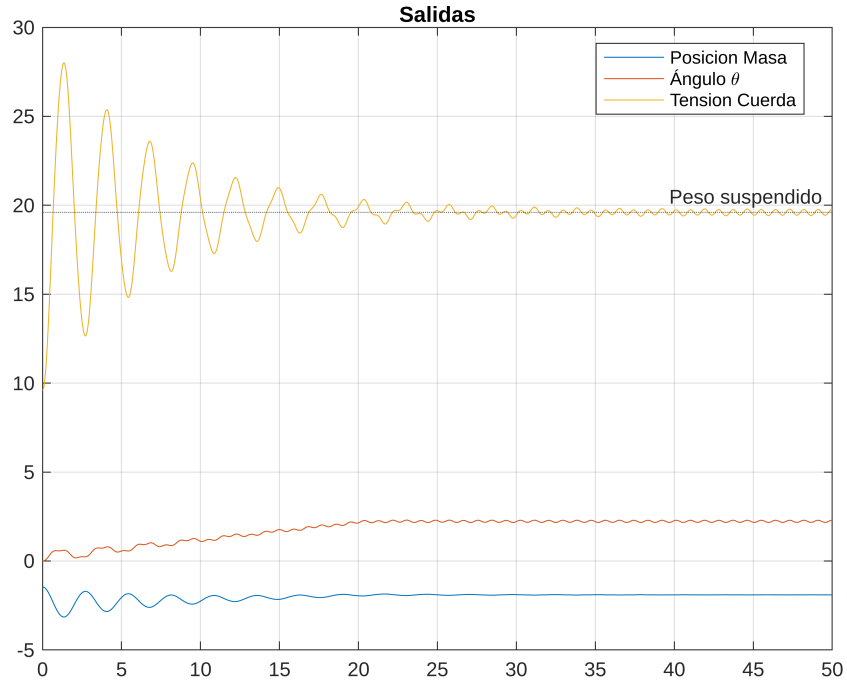
```
EcSalidaNum = function_handle with value:
    @(in1)[in1(1,:);in1(3,:);in1(1,:).*-1.0e+1+in1(3,:).*(5.0./2.0)-5.0]
```

```
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")
```



Modelo linealizado normalizado en forma matricial y análisis de propiedades: estabilidad interna/externa, respuesta en frecuencia [control systems Toolbox]

Revisemos resultado del modelado, ecuación de estado:

```
VectorDeEstados,Entradas
```

VectorDeEstados =

$$\begin{pmatrix} p \\ v \\ \theta \\ \omega \end{pmatrix}$$

Entradas = T_{motor}

```
vpa(DerivadasDelEstado) %en "número real", no en "fracción
simbólica"
```

ans =

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

Linealización

El sistema es "casi" lineal, excepto los términos 12.3 y 1.25 por longitud natural de muelle y peso de la masa colgante.

Pero, en "coordenadas incrementales", desaparecerán. Por ejemplo:

$$\begin{aligned} \frac{d\Delta v}{dt} &= \frac{dv}{dt} - \frac{dv_{eq}}{dt} = \frac{dv}{dt} - 0 = 1.25\theta - 5p - 0.15v - 12.3 = \\ &= 1.25(\theta_{eq} + \Delta\theta) - 5(p_{eq} + \Delta p) - 0.15(v_{eq} + \Delta v) - 12.3 = \\ &= \underbrace{1.25\theta_{eq} - 5p_{eq} - 0.15v_{eq} - 12.3}_{0} + (1.25\Delta\theta - 5\Delta p - 0.15\Delta v) = \\ &= 1.25\Delta\theta - 5\Delta p - 0.15\Delta v \end{aligned}$$

Nota: el resultado es idéntico a la "linealización rápida" de $\frac{dx}{dt} = f(x)$ como $\frac{d\Delta x}{dt} \approx \frac{\partial f}{\partial x}\Delta x$, pero como $f(x)$ era "afín" (lineal + constantes) realmente NO hay ninguna aproximación, sólo un "cambio del origen de coordenadas al punto x_{eq} donde se supone que se verifica $f(x_{eq}) = 0$, de modo que $\frac{dx_{eq}}{dt} = 0$.

La idea general justificando $\frac{d\Delta x}{dt} \approx \frac{\partial f}{\partial x}\Delta x$ era:

$$\frac{d\Delta x}{dt} = \frac{dx}{dt} - \underbrace{\frac{dx_{eq}}{dt}}_{0} = f(x) \approx \underbrace{f(x_{ex})}_{0} + \frac{\partial f}{\partial x} \cdot (x - x_{eq}) = \frac{\partial f}{\partial x}\Delta x$$

Claro, en una función con forma de "recta" (o hiperplano en general) donde derivadas segundas y sucesivas sean cero, la serie de Taylor termina en la derivada primera y se verifica igualdad, sin ninguna aproximación:

$$f(x) = \underbrace{f(x_{ex})}_{0} + \frac{\partial f}{\partial x} \cdot (x - x_{eq})$$

Por tanto, en "incrementos" desde una posición de equilibrio, podremos expresar EXACTAMENTE (en el caso afin) al sistema como $\frac{dx}{dt} = Ax + Bu,\ y = Cx + Du$ esto es, la forma normalizada matricial, siendo A, B, C, y D:

```
A=double(jacobian(DerivadasDelEstado,VectorDeEstados)) %"eval", en
versiones anteriores a 2024b
```

```
A = 4×4
         0    1.0000         0         0
   -5.0000   -0.1500    1.2500         0
         0         0         0    1.0000
    2.5000         0   -0.6250   -2.0000
```

```
B=double(jacobian(DerivadasDelEstado,Entradas))
```

```
B = 4×1
     0
     0
     0
     1
```

Las salidas que se habían considerado "de interés" eran posición lineal, posición angular, y tensión de la cuerda.

En forma normalizada, las ecuaciones de salida eran:

```
vpa(EcSalidaSym)
```

```
ans =
```

$$\begin{pmatrix} p \\ \theta \\ 2.5\,\theta - 10.0\,p - 5.0 \end{pmatrix}$$

Para expresarlas en forma de matriz:

```
C=double(jacobian(EcSalidaSym,VectorDeEstados))
```

```
C = 3×4
    1.0000         0         0         0
         0         0    1.0000         0
  -10.0000         0    2.5000         0
```

```
D=double(jacobian(EcSalidaSym,Entradas))
```

```
D = 3×1
     0
     0
     0
```

Con lo que ya estamos en condiciones de crear un objeto "sistema state space" de la Control Systems Toolbox:

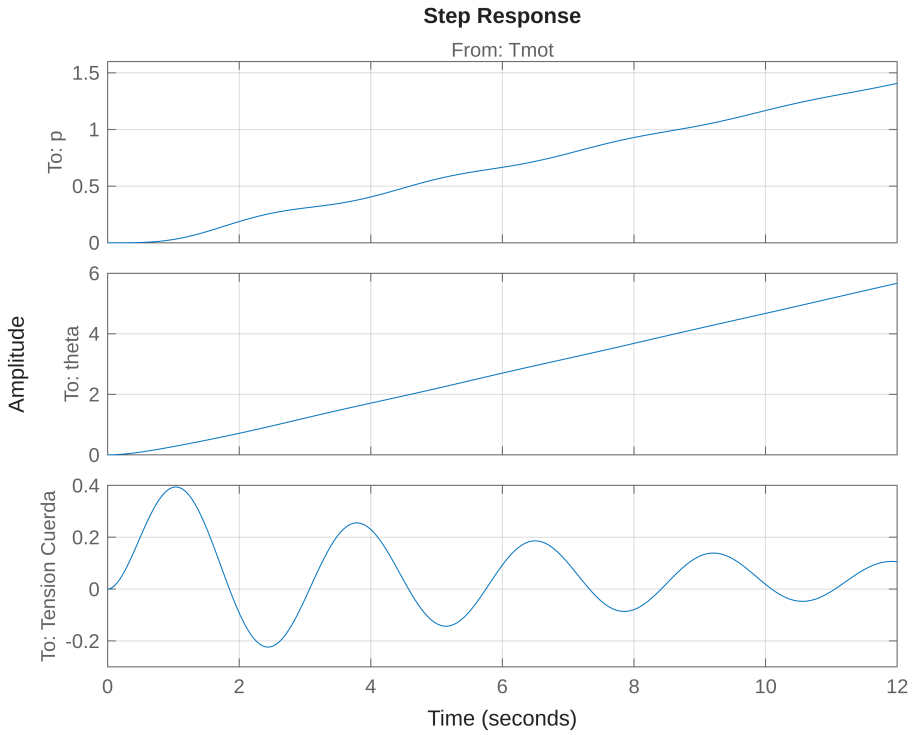
```
sys=ss(A,B,C,D); %objeto Control System Toolbox
sys.InputName='Tmot';
sys.StateName={'p','v','theta','w'};
```

```
sys.OutputName={'p','theta','Tension Cuerda'};
```

Los nombres arriba son meramente "cosméticos", opcionales, para que Matlab etiquete los resultados con ellos.

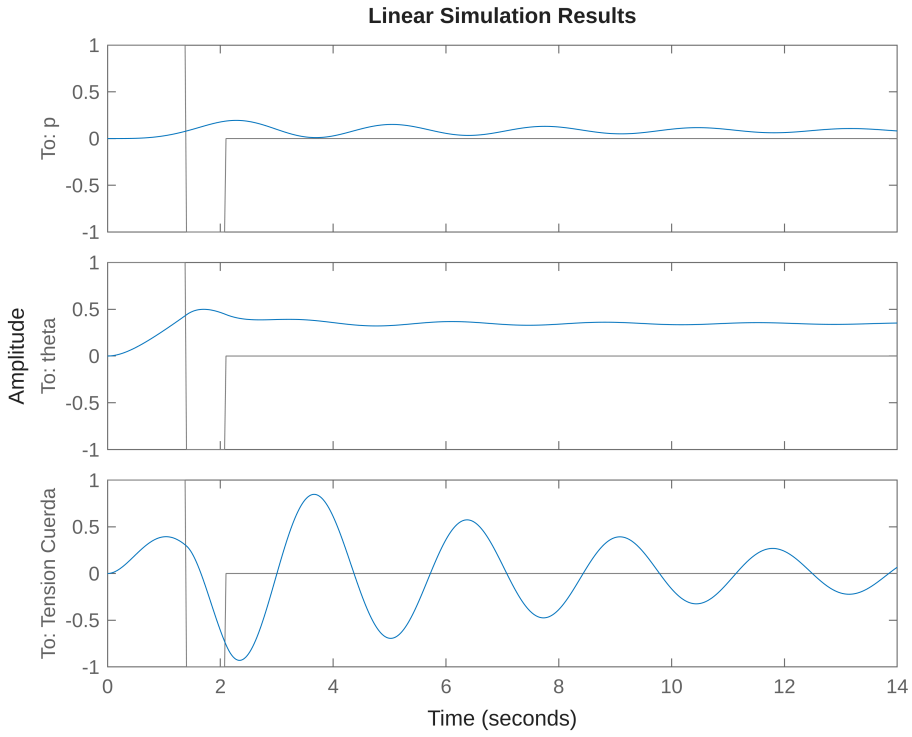
Con él, podremos, por ejemplo, simularlo con código "especializado en sistemas lineales" más eficiente/exacto que algoritmos genéricos (Runge-Kuta).

```
step(sys,12), grid on %12 segundos ante escalón unitario de par
```



Ante formas de entrada no estándar:

```
Tsim=0:0.025:14;
TmotSim = 1*(Tsim<1.4) + (-1)*(Tsim>=1.4 & Tsim<2.1);
lsim(sys,TmotSim,Tsim)
```



Análisis de propiedades

Estabilidad

***repr. interna** $\frac{dx}{dt} = Ax + Bu$

```
lospolos=eig(A)
```

```
lospolos = 4×1 complex
  -0.1404 + 2.3196i
  -0.1404 - 2.3196i
  -1.8692 + 0.0000i
  -0.0000 + 0.0000i
```

*Función/matriz de transferencia

$G(s) = C(sI - A)^{-1}B + D$ resulta en una FdT donde el denominador es $det(sI - A)$, por fórmulas de matriz inversa. Matlab obtiene:

```
systf=zpk(sys) %Los polos son raíces de denominador
```

```
systf =

  From input "Tmot" to output...
                    1.25
   p:  ---------------------------------
       s (s+1.869) (s^2 + 0.2808s + 5.4)

                   (s^2 + 0.15s + 5)
   theta:  ---------------------------------
           s (s+1.869) (s^2 + 0.2808s + 5.4)
```

```
                              2.5 s (s+0.15)
   Tension Cuerda:  --------------------------------
                    s (s+1.869) (s^2 + 0.2808s + 5.4)

Continuous-time zero/pole/gain model.
Model Properties
```

```
systf.P
```

```
ans = 3×1 cell
```

	1
1	[-0.1404 + 2.3196i;-0.1404 - 2.3196i;-1.8692 + 0i;0 + 0i]
2	[-0.1404 + 2.3196i;-0.1404 - 2.3196i;-1.8692 + 0i;0 + 0i]
3	[-0.1404 + 2.3196i;-0.1404 - 2.3196i;-1.8692 + 0i;0 + 0i]

Como hay un polo en cero, el sistema es INESTABLE (marginalmente).

Nótese que la "tensión de la cuerda" NO es inestable (eso lo analizaremos luego).

```
minreal(systf)
```

```
ans =

  From input "Tmot" to output...
                    1.25
   p:  ---------------------------------
       s (s+1.869) (s^2 + 0.2808s + 5.4)

                   (s^2 + 0.15s + 5)
   theta:  ---------------------------------
           s (s+1.869) (s^2 + 0.2808s + 5.4)

                              2.5 (s+0.15)
   Tension Cuerda:  -------------------------------
                    (s+1.869) (s^2 + 0.2808s + 5.4)

Continuous-time zero/pole/gain model.
Model Properties
```

*Los sistemas con equilibrio "indiferente" tienen polos en el origen (su respuesta libre contiene $e^{0t} = 1$ constantes: alejados del "cero" algo se mantiene constante sin volver al "cero", luego lo vemos en detalle).

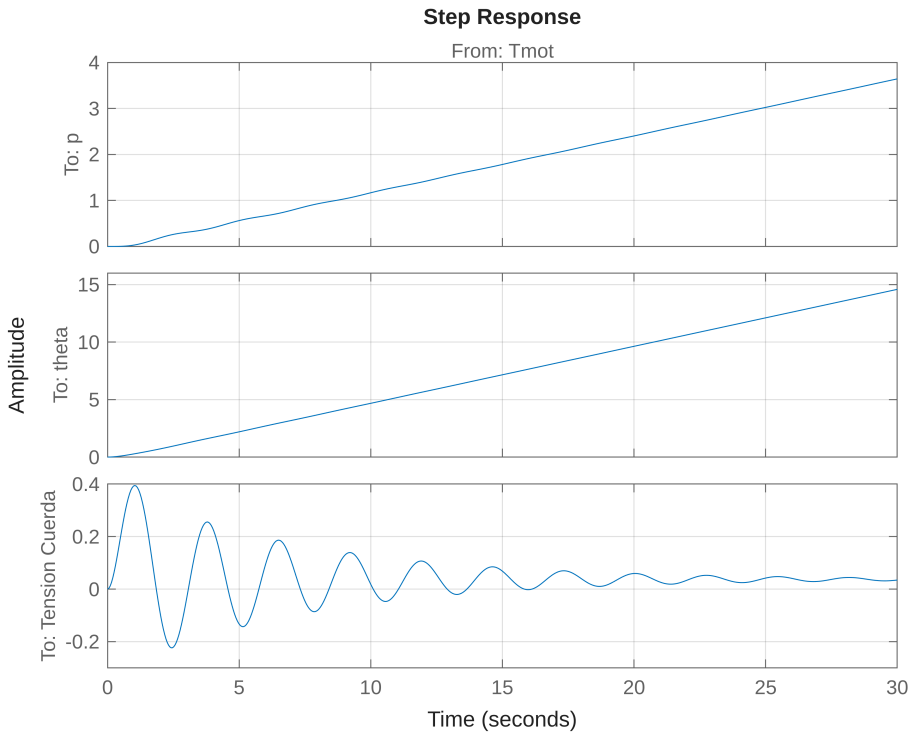
Pero el sistema es INESTABLE "***Bounded Input Bounded Output***" porque existen "entradas acotadas" que resultan en "salidas NO acotadas":

En efecto, la masa desenrollará la polea hasta el infinito si en la realidad pesa "un gramo más" que el equilibrio calculado... o el motor enrollará totalmente la cuerda si la masa pesa "un gramo menos" que el equilibrio calculado.

El cálculo de T_equilibrio NUNCA será exacto y, sin un sistema de "control en bucle cerrado", el proceso no podrá ser estabilizado en la práctica, aunque las simulaciones digan que sí poniendo "exactamente" el par de giro que compensa al peso.

La inestabilidad es "marginal": no se irá a infinito "exponencialmente" sino que, ante un par de entrada constante ligeramente desequilibrado, la solución de determinadas variables contendrá la integral de esta entrada (o sea, tendrá forma de "rampa").

```
step(sys,30), grid on
```



La justificación, en el dominio de Laplace es la siguiente: si $G(s) = G_1(s)\frac{1}{s}$, entonces la respuesta escalón $G(s) \cdot \frac{1}{s}$ será $G_1(s)\frac{1}{s^2}$ y las fracciones simples serán en la forma $\frac{M}{s} + \frac{N}{s^2}$ +términos dependientes de las raíces de G_1. El término N/s^2 antitransforma a $N \cdot t$, que va a infinito.

Modos de la respuesta libre

Los autovectores nos darán información adicional, porque si $Av = \lambda v$, entonces significa que si $x(0) = v$, en $t = 0$ tenemos $\frac{dx}{dt} = \lambda v$, y como $\frac{dx}{dt}$ no cambia la direccion de v, $x(t)$

continuara siendo un autovector en el futuro si $x(0) = v$ lo es, de modo que la solución de las EDO será $x(t) = e^{\lambda t} \cdot x(0) = e^{\lambda t} \cdot v$.

```
[V,Dia]=eig(A);
V
```

```
V = 4×4 complex
  -0.0223 - 0.3676i  -0.0223 + 0.3676i  -0.0710 + 0.0000i   0.2425 + 0.0000i
   0.8557 + 0.0000i   0.8557 + 0.0000i   0.1327 + 0.0000i  -0.0000 + 0.0000i
  -0.0824 + 0.1177i  -0.0824 - 0.1177i  -0.4664 + 0.0000i   0.9701 + 0.0000i
  -0.2613 - 0.2077i  -0.2613 + 0.2077i   0.8717 + 0.0000i  -0.0000 + 0.0000i
```

```
diag(Dia)'
```

```
ans = 1×4 complex
  -0.1404 - 2.3196i  -0.1404 + 2.3196i  -1.8692 + 0.0000i  -0.0000 + 0.0000i
```

El cambio de variable $Tx_{new} = x$ resulta

$$\frac{dx_{new}}{dt} = T^{-1}\frac{dx}{dt} = T^{-1}(Ax + Bu) = \underbrace{T^{-1}AT}_{A_{new}} \cdot x_{new} + \underbrace{T^{-1}B}_{B_{new}} \cdot u$$

$$y = Cx + Du = \underbrace{CT}_{C_{new}} \cdot x_{new} + Du$$

con T siendo la matriz de vectores propios da lugar a la "forma canónica modal con coeficientes complejos" con A_{new} diagonal (poco útil en "ingeniería" que preferimos números reales, pero con cierta "interpretación" que vamos a discutir):

```
Anew=inv(V)*A*V
```

```
Anew = 4×4 complex
  -0.1404 + 2.3196i  -0.0000 + 0.0000i   0.0000 - 0.0000i  -0.0000 - 0.0000i
  -0.0000 - 0.0000i  -0.1404 - 2.3196i   0.0000 + 0.0000i  -0.0000 + 0.0000i
   0.0000 - 0.0000i   0.0000 + 0.0000i  -1.8692 + 0.0000i   0.0000 - 0.0000i
   0.0000 - 0.0000i   0.0000 + 0.0000i   0.0000 - 0.0000i  -0.0000 + 0.0000i
```

```
Bnew=inv(V)*B
```

```
Bnew = 4×1 complex
  -0.0873 - 0.0650i
  -0.0873 + 0.0650i
   1.1259 - 0.0000i
   0.5106 + 0.0000i
```

```
Cnew=C*V
```

```
Cnew = 3×4 complex
  -0.0223 - 0.3676i  -0.0223 + 0.3676i  -0.0710 + 0.0000i   0.2425 + 0.0000i
  -0.0824 + 0.1177i  -0.0824 - 0.1177i  -0.4664 + 0.0000i   0.9701 + 0.0000i
   0.0164 + 3.9699i   0.0164 - 3.9699i  -0.4561 + 0.0000i  -0.0000 + 0.0000i
```

```
Dnew=D
```

```
Dnew = 3×1
```

```
     0
     0
     0
```

Los elementos igual a cero en Bnew y Cnew detectan, respectivamente, modos "no controlables" (la entrada no los puede mover) o "no observables" (esa salida no se desviará del equilibrio aunque el estado asociado al modo sí esté desviado).

*Con valores propios repetidos (forma de Jordan, triangular) hay que refinar la discusión, no objeto de este material introductorio.

```
sys_modal=canon(sys,'modal') %Variación para tener coeficientes
reales, conceptualmente parecida a la "compleja"
```

```
sys_modal =

  A =
              x1         x2         x3         x4
   x1    -0.1404       2.32          0          0
   x2      -2.32    -0.1404          0          0
   x3          0          0     -1.869          0
   x4          0          0          0  -1.73e-16

  B =
        Tmot
   x1  1.345
   x2  2.189
   x3  3.552
   x4  4.431

  C =
                        x1          x2          x3          x4
   p                -0.03118  -0.0009035     -0.0225     0.02795
   theta            0.009304    0.007846     -0.1478      0.1118
   Tension Cuer       0.3351     0.02865     -0.1446           0

  D =
                 Tmot
   p                0
   theta            0
   Tension Cuer     0

Continuous-time state-space model.
Model Properties
```

```
modo_a_analizar=1;
exp_asociada=diag(Dia(modo_a_analizar,modo_a_analizar))
```

```
exp_asociada =
-0.1404 + 2.3196i
```

```
direccionmodo=V(:,modo_a_analizar)
```

```
direccionmodo = 4×1 complex
  -0.0223 - 0.3676i
   0.8557 + 0.0000i
  -0.0824 + 0.1177i
  -0.2613 - 0.2077i
```

Los modos 1:2 son modos "oscilatorios" (parte imaginaria no nula). Los 3 y 4 no lo son.

La frecuencia de las oscilaciones propias del modo analizado es:

```
imag(exp_asociada)
```

```
ans =
2.3196
```

y la exponencial de la parte real, como es estrictamente negativa, decae un 98% en

$e^{Re(polo)\cdot t_{est}} = 0.02$, o sea $Re(polo) \cdot t_{est} = 0.02$

```
log(0.02)./real(exp_asociada) %aprox -4/parte_real_polo
```

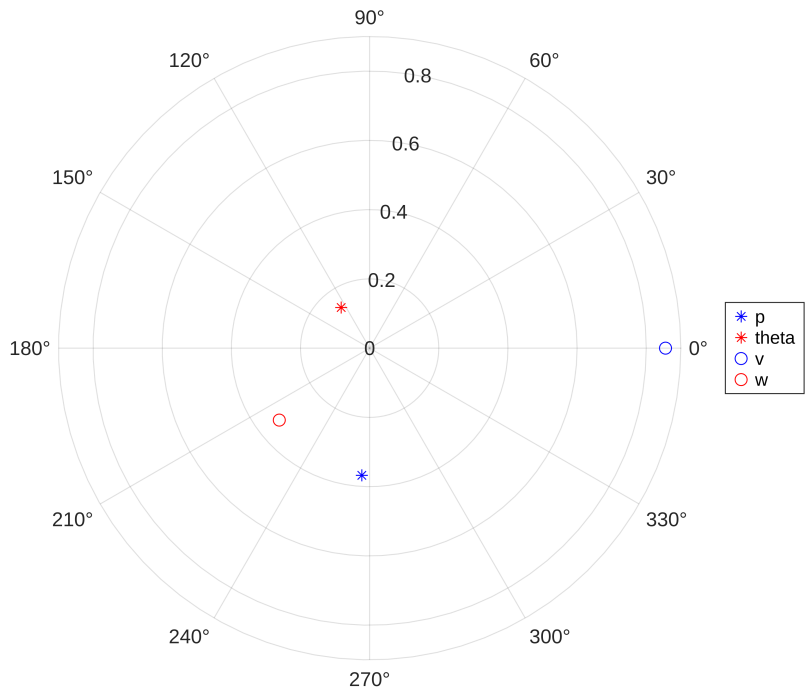
```
ans =
27.8614
```

```
%log(0.02)
```

*Modos 1 y 2: un modo es el "conjugado" del otro, es "hacia adelante" o "hacia atrás" en el sentido de giro, pero son físicamente lo mismo.

Las amplitudes relativas (caen exponencialmente) y desfases de posiciones y velocidades las podemos ver en un diagrama polar:

```
figure()
polarplot(direccionmodo(1,1),'*b'), hold on
polarplot(direccionmodo(3,1),'*r')
polarplot(direccionmodo(2,1),'ob')
polarplot(direccionmodo(4,1),'or')
hold off
legend("p","theta","v","w")
```



Relación entre posición y velocidades:

$\xi = C \cdot e^{(a+bj)t}$ su derivada es $\xi' = (a+bj) \cdot \xi$, estaría desfasada 90º si $a = 0$, como en las ecuaciones de condensador y bobina ideal sin resistencias, pongamos, o en sistemas mecánicos sin rozamiento. El desfase será "cero" o "180" en modos no oscilatorios ($b = 0$).

```
angle(exp_asociada)*180/pi
```

```
ans =
93.4641
```

```
angle(direccionmodo(2,1)/direccionmodo(1,1))*180/pi
```

```
ans =
93.4641
```

```
angle(direccionmodo(4,1)/direccionmodo(3,1))*180/pi
```

```
ans =
93.4641
```

```
abs(exp_asociada)
```

```
ans =
2.3238
```

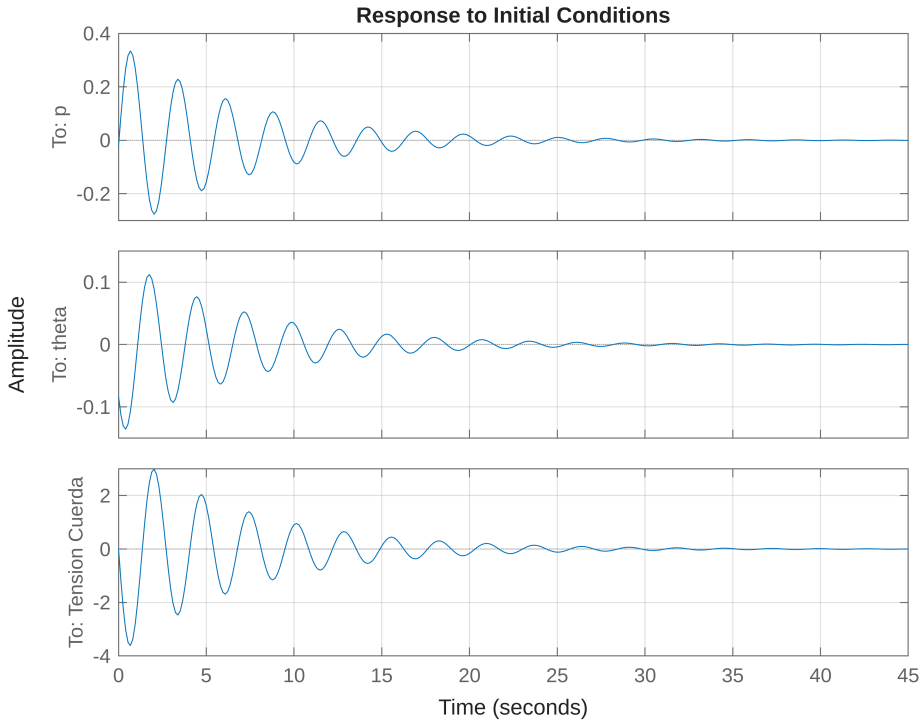
```
abs(direccionmodo(2,1)/direccionmodo(1,1))
```

```
ans =
2.3238
```

```
abs(direccionmodo(4,1)/direccionmodo(3,1))
```

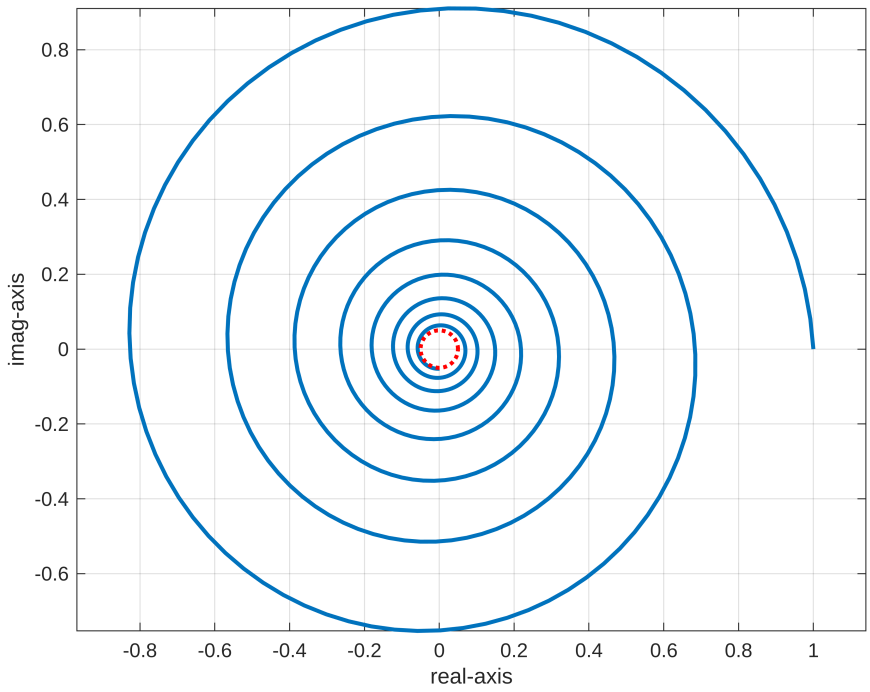
```
ans =
2.3238
```

```
figure()
initial(sys,abs(direccionmodo(:,1)).*cos(angle(direccionmodo(:,1))))
grid on
```



Las trayectorias en el plano complejo son espirales logarítmicas en el caso de polos complejos:

```
timerange=linspace(0,21,600);
%theta =  % Angle parameter
radius = exp((Dia(modo_a_analizar,modo_a_analizar)) * timerange); %
Radius as a function of theta
if modo_a_analizar==4
    plot(real(radius),imag(radius),LineWidth=2,Marker="o")
else
    plot(real(radius),imag(radius),LineWidth=2)
end
grid on
hold on
theta_circle = linspace(0, 2*pi, 100);
radius_in = 0.05;
x_circle = radius_in * cos(theta_circle);
y_circle = radius_in * sin(theta_circle);
plot(x_circle, y_circle, 'r:', 'LineWidth', 2), grid on
axis equal
xlabel('real-axis')
ylabel('imag-axis')
hold off
```



Estabilidad "de los estados" (interna) versus estabilidad "de un par entrada-salida concreto"

Los valores propios de A son las raíces de denominador de las funciones de transferencia, porque $(sI - A)^{-1} = \frac{1}{det(sI - A)} \cdot adj(\,(sI - A)^T)$:

```
zpk(sys) %hace el cálculo C(sI-A)^{-1}B+D y factoriza la fracción resultante
```

```
ans =

  From input "Tmot" to output...
                    1.25
   p:  ---------------------------------
       s (s+1.869) (s^2 + 0.2808s + 5.4)

                   (s^2 + 0.15s + 5)
   theta:  ---------------------------------
           s (s+1.869) (s^2 + 0.2808s + 5.4)

                              2.5 s (s+0.15)
   Tension Cuerda:  ---------------------------------
                    s (s+1.869) (s^2 + 0.2808s + 5.4)

Continuous-time zero/pole/gain model.
Model Properties
```

Pero no todas las "**salidas**" son inestables: la **velocidad** de las cosas y la **tensión** de la cuerda (salida 3 arriba) llegan a un equilibrio.

```
minreal(zpk(sys)) %cancelamos polos y ceros (simplificamos fracciones)
```

```
ans =

  From input "Tmot" to output...
                    1.25
   p:  ---------------------------------
       s (s+1.869) (s^2 + 0.2808s + 5.4)

                   (s^2 + 0.15s + 5)
   theta:  ---------------------------------
           s (s+1.869) (s^2 + 0.2808s + 5.4)

                              2.5 (s+0.15)
   Tension Cuerda:  -------------------------------
                    (s+1.869) (s^2 + 0.2808s + 5.4)

Continuous-time zero/pole/gain model.
Model Properties
```

Comprobemos lo de las velocidades, viendo las funciones de transferencia:

```
sys2=ss(A,B,[0 1 0 0;0 0 0 1;C(3,:)],zeros(3,1));
sys2.InputName='Tmot';
sys2.OutputName={'v','omega','Tension_Cuerda'};
zpk(sys2)
```

```
ans =

  From input "Tmot" to output...
                    1.25 s
   v:  ---------------------------------
       s (s+1.869) (s^2 + 0.2808s + 5.4)

                  s (s^2 + 0.15s + 5)
   omega:  ---------------------------------
           s (s+1.869) (s^2 + 0.2808s + 5.4)

                              2.5 s (s+0.15)
   Tension_Cuerda:  -----------------------------------
                    s (s+1.869) (s^2 + 0.2808s + 5.4)

Continuous-time zero/pole/gain model.
Model Properties
```

```
minreal(zpk(sys2))
```

```
ans =

  From input "Tmot" to output...
                    1.25
   v:  -------------------------------
       (s+1.869) (s^2 + 0.2808s + 5.4)

                  (s^2 + 0.15s + 5)
   omega:  -------------------------------
           (s+1.869) (s^2 + 0.2808s + 5.4)

                              2.5 (s+0.15)
   Tension_Cuerda:  -------------------------------
                    (s+1.869) (s^2 + 0.2808s + 5.4)

Continuous-time zero/pole/gain model.
Model Properties
```

Aunque la FdT de cierta salida (después de "cancelar" la "s" en numerador y denominador) sea estable, el sistema "internamente" **no** lo es: hay estados (variables internas) que pueden irse a infinito **NO OBSERVABLES** sólo con medidas de ciertas salidas.
Se dice que el sistema es **"externamente" estable** entrada/salida (**para las salidas mencionadas**: velocidades, tensión cuerda), pero **"internamente" inestable**, o en la jerga, existe una "**inestabilidad (marginal) interna no detectable**" con solamente sensores de velocidad o tensión de la cuerda. Ello indicaría, que en una aplicación práctica de "monitorización" o "control", utilizar sólo estos "sensores" sería una decisión de ingeniería errónea.

```
sys3=minreal(sys2);
```

```
1 state removed.
```

```
polosestables=pole(sys3)
```

```
polosestables = 3×1 complex
  -0.1404 + 2.3196i
  -0.1404 - 2.3196i
  -1.8692 + 0.0000i
```

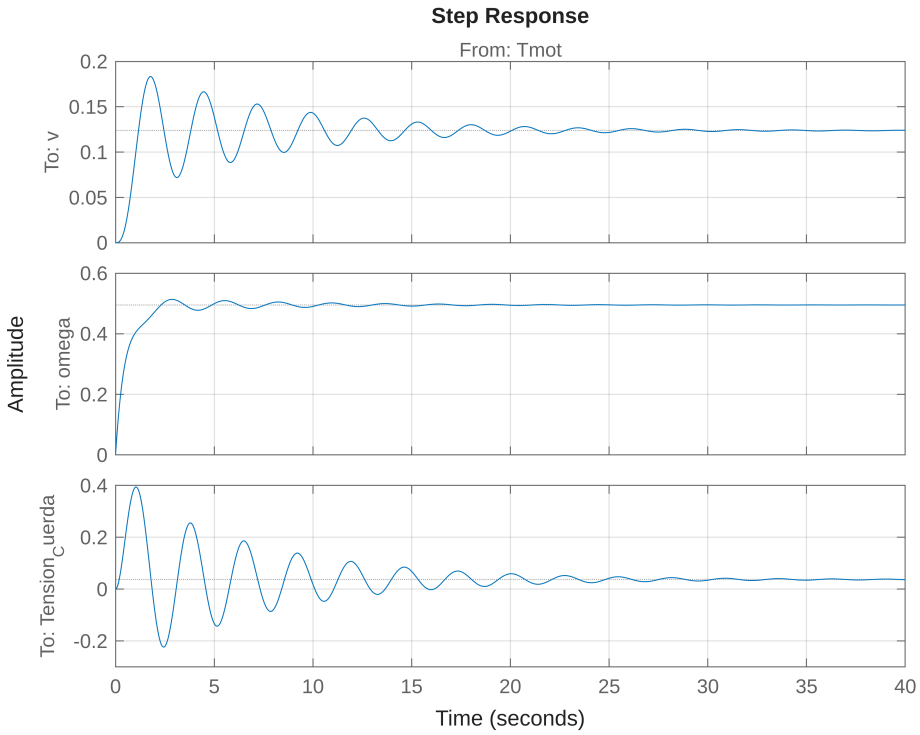
```
4./real(polosestables) %Tiempo de Establecimiento
```

```
ans = 3×1
  -28.4880
  -28.4880
   -2.1400
```

```
imag(polosestables) %Frecuencia de las oscilaciones propias
```

```
ans = 3×1
    2.3196
   -2.3196
         0
```

```
step(sys3), grid on
```




```
dcgain(sys3) %Valor final (por unidad de Tmot de entrada)
```

```
ans = 3×1
    0.1238
    0.4954
    0.0372
```

*Pero, recordemos que estos cálculos NO sirven de NADA: el sistema es "**internamente inestable**" y, por tanto, si lo hacemos funcionar tal y como está, algo se romperá en la práctica, habrá colisiones de la carga con el suelo o con la polea tarde o temprano.

Será **obligatorio** utilizar un "**sistema de control***" para "**estabilizarlo**" y dicho sistema de control deberá tener sensores donde ese modo "*s=0*" sea "**observable**", y la inestabilidad pueda ser "**detectada**" a partir de las medidas.

***El sistema de control puede, obviamente, ser un "**operador humano**" (control "**manual**"), dado que con su sensor de "visión" podrá "**observar**" la posición del sistema y "**detectar**" que el sistema se está cayendo sin control y subir el par para que pare de caer, o viceversa (bajar el par si la masa está subiendo sin parar y colisionará con la polea si no se actúa).

Respuesta forzada en frecuencia

Podríamos dibujar el Diagrama de Bode graficando $G(j\omega)$ siendo $G(s)$ la función de transferencia, que, a partir de una representación interna es $G(s) = C \cdot (sI - A)^{-1} \cdot B + D$.

A una frecuencia en concreto:

```
Frdat=freqresp(sys,0.25)
```

```
Frdat = 3×1 complex
  -0.0723 - 0.4914i
  -0.2709 - 1.9432i
   0.0459 + 0.0560i
```

```
abs(Frdat)
```

```
ans = 3×1
    0.4967
    1.9620
    0.0724
```

```
angle(Frdat)*180/pi
```

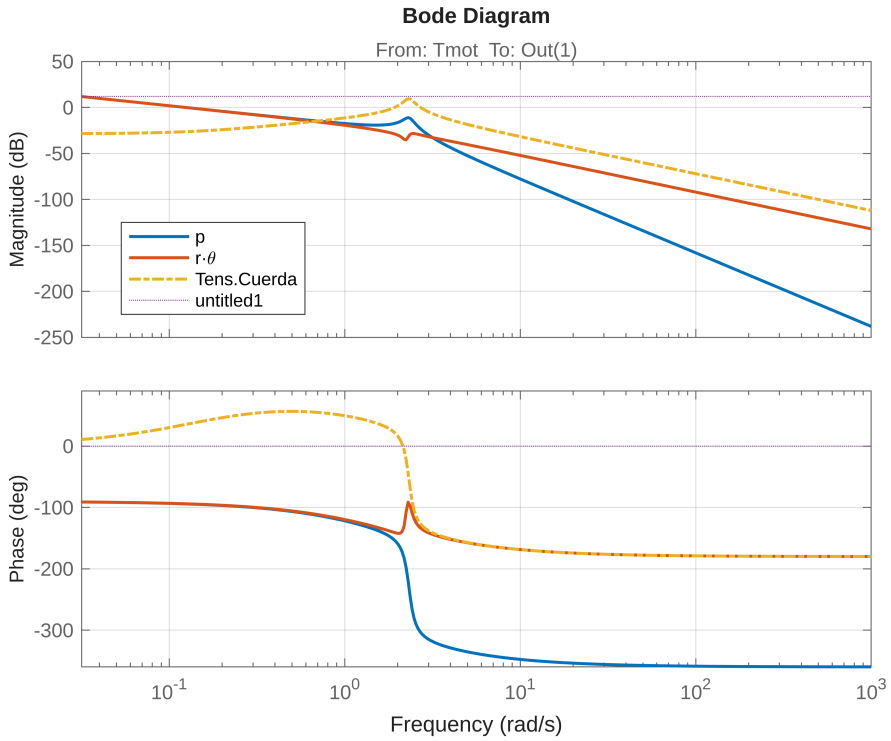
```
ans = 3×1
  -98.3716
  -97.9364
   50.6647
```

```
(Frdat(1)-r*Frdat(2))*k_muelle %Comprobamos tensión cuerda
```

```
ans =
-0.0459 - 0.0560i
```

En el Bode, voy a "escalar" multiplicando θ por el radio, para tener las posiciones en "metros" (o sea, posición de la punta de la "cuerda", incremental).

```
h=bodeplot(sys(1),r*sys(2),sys(3),tf(1/r),logspace(-1.5,3,400));
grid on
h.Responses(1).LineWidth=1.5;
h.Responses(2).LineWidth=1.5;
h.Responses(3).LineWidth=1.5;h.Responses(3).LineStyle='-.';
h.Responses(4).LineStyle=':';
legend("p","r·\theta","Tens.Cuerda",'',Location="best")
```



*La inestabilidad marginal se nota como que la amplitud de la respuesta en frecuencia **tiende a infinito** a baja frecuencia... Pero, en un caso general, **podría ser que la inestabilidad** "exponencial" **no se viera en la respuesta en frecuencia** $G(j\omega)$: el sistema $G(s) = 1/(s-2)$ tiene una respuesta en frecuencia acotada a todas las frecuencias $1/(j\omega - 1)$, pero el transitorio exponencial es inestable.

También se observa una resonancia mecánica alrededor de 2.3 rad/s, que es la frecuencia "propia" de las oscilaciones de la respuesta libre: aunque las frecuencias de resonancia no coinciden "exactamente" con las "*propias*" del sistema suelen estar muy cerca en sistemas poco amortiguados.

Si quisiéramos ver la aceleración lineal y angular en el rango de frecuencias entre 10^{-2} y 10^{2}, por ejemplo, haríamos:

```
s=tf('s');
Acels=minreal(s^2*sys(1:2,:)); %Minreal cancela "s" en numerador y
denominador
```

```
4 states removed.
```

```
Acels.OutputName={'Acel. p','Acel. theta'};
zpk(Acels)
```

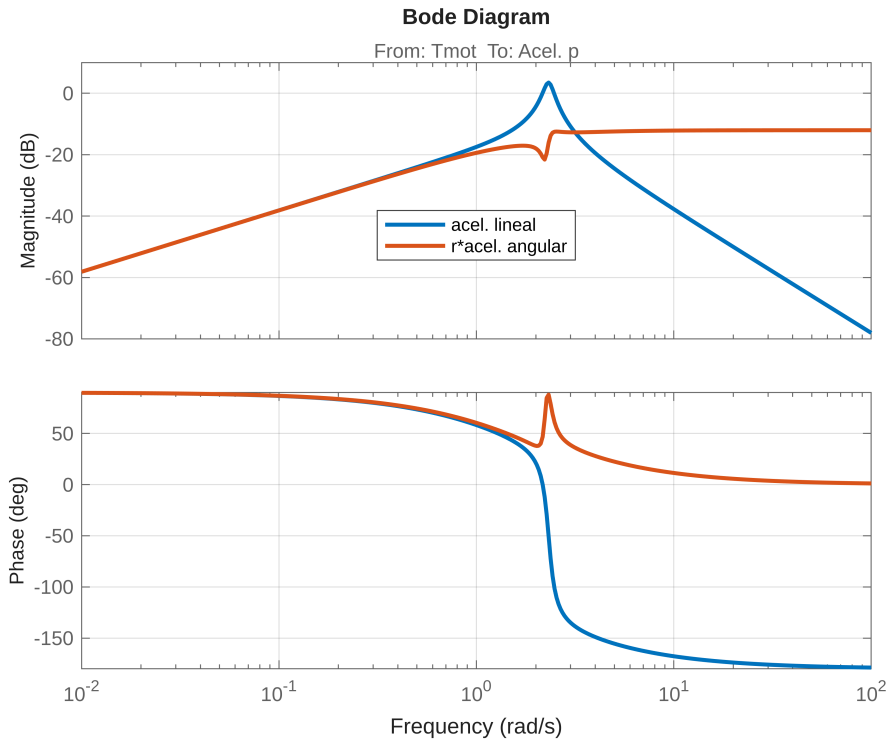
```
ans =

  From input "Tmot" to output...
                          1.25 s
   Acel. p:  -------------------------------
             (s+1.869) (s^2 + 0.2808s + 5.4)
```

```
                        s (s^2 + 0.15s + 5)
   Acel. theta:  -------------------------------
                 (s+1.869) (s^2 + 0.2808s + 5.4)

Continuous-time zero/pole/gain model.
Model Properties
```

```
h=bodeplot(Acels(1),r*Acels(2),logspace(-2,2,400));
h.Responses(1).LineWidth=2;
h.Responses(2).LineWidth=2;
grid on
legend("acel. lineal","r*acel. angular",location="best")
```



Las amplitudes de "posiciones" en resonancia servirían, por ejemplo, para comprobar si no habrá colisiones al ser excitado a determinada frecuencia, las aceleraciones (o sea, fuerzas) están relacionadas con el confort de ocupantes de un ascensor, por ejemplo... Cada aplicación tecnológica tiene unas "salidas de interés" diferentes.

Impedancia mecánica

Es la resp. en frecuencia de la dinámica entre velocidad angular y par del punto de "interfaz rotacional" con el exterior.

```
sys_Tmot_to_w=minreal(ss(A,B,[0 0 0 1],0));
```

```
1 state removed.
```

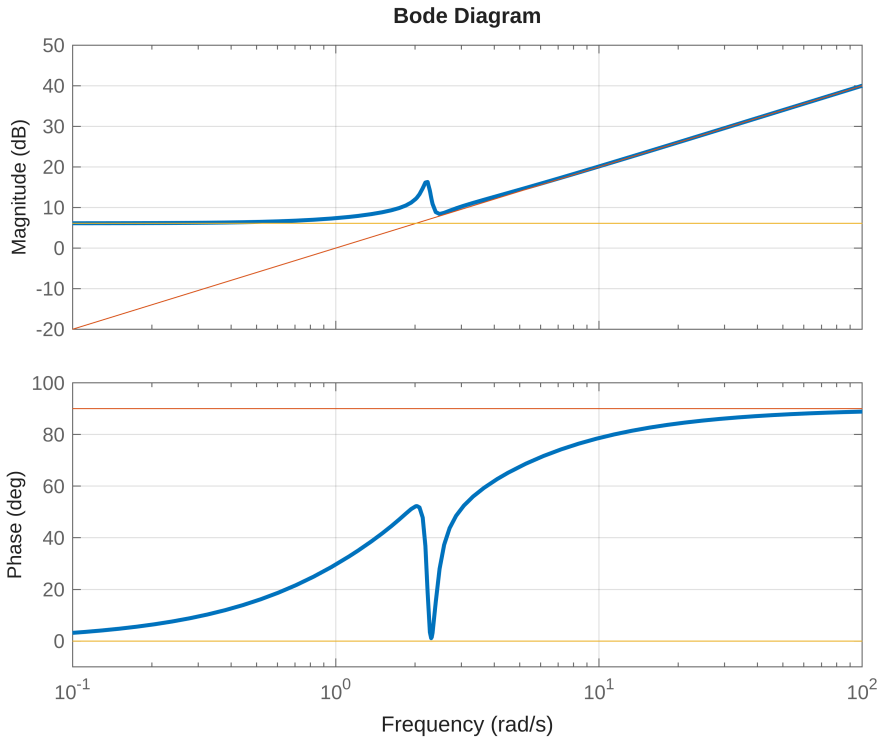
```
ImpedanciaMecanica=1/sys_Tmot_to_w;
dcgain(ImpedanciaMecanica)
```

```
ans =
2.0187
```

```
coef_fric_equiv=coef_fric_rodamientos+r*r*coef_fric_aire
```

```
coef_fric_equiv =
2.0187
```

```
h=bodeplot(ImpedanciaMecanica,Inercia*s,tf(coef_fric_equiv));
h.Responses(1).LineWidth=2; grid on
```



Apéndice (func. auxiliares), Método Euler

¡NO USAR Euler!: Esto solamente tiene una función didáctica... precisamente los códigos de simulación y animación "profesionales" ("*production ready*" en la jerga informática) 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,
PasoDeIntegracion)
 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
```