Curso de Controle Clássico

Prof: Dennis

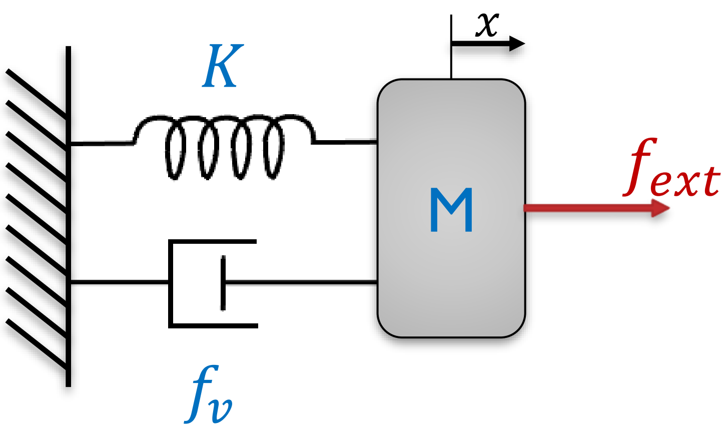
Aula Prática #03C – Simulação de sist de 2ª ordem e PID com a toolbox de controle do python.

O que faremos nesta aula

- Nesta aula continuaremos usando o Python para simular um sistema de 2ª ordem obtendo sua resposta ao degrau em malha aberta e fechada;
- O exemplo escolhido será um sistema mecânico: Massa + mola e amortecedor em paralelo, que é intrinsecamente de 2ª ordem;
- Nas duas primeiras partes desta aula já usamos o Scilab é para encontrar a resposta ao degrau de duas formas:
  - Graficamente, montando o diagrama do sistema no Xcos do Scilab;
  - Diretamente usando um código na linguagem do Scilab;
- Veremos nesta aula como usar um código em python para fazer a mesma simulação.
- Por simplicidade usaremos uma ferramenta web, o "google colab";
- Uma vantagem de usar o python, é que se trata de uma linguagem com muito mais bibliotecas e possibilidades do que o Scilab.
- Obteremos o mesmo resultado pelos 3 métodos, ficando a escolha do estudante qual dos três deseja usar;

Do modelo à função de transferência:

- Do sistema escolhido devemos encontrar a função de transferência no domínio s:



$$f_{ext} = M\frac{d^2x}{dt^2} + f_v\frac{dx}{dt} + Kx$$

$$F_{ext} = Ms^2X + f_vsX + KX$$

$$P(s) = \frac{X}{F_{ext}} = \boxed{\frac{1}{Ms^2 + f_vs + K}}$$

  - Agora vamos atribuir valores as constantes do sistema:

$$M = 0{,}1\,Kg \quad f_v = 0{,}2\,\frac{\mathrm{Ns}}{\mathrm{m}} \quad K = 1{,}6\,\frac{\mathrm{N}}{\mathrm{m}}$$

$$P(s) = \frac{1}{Ms^2 + f_vs + K} = \frac{1}{0{,}1s^2 + 0{,}2s + 1{,}6}$$ Sendo :

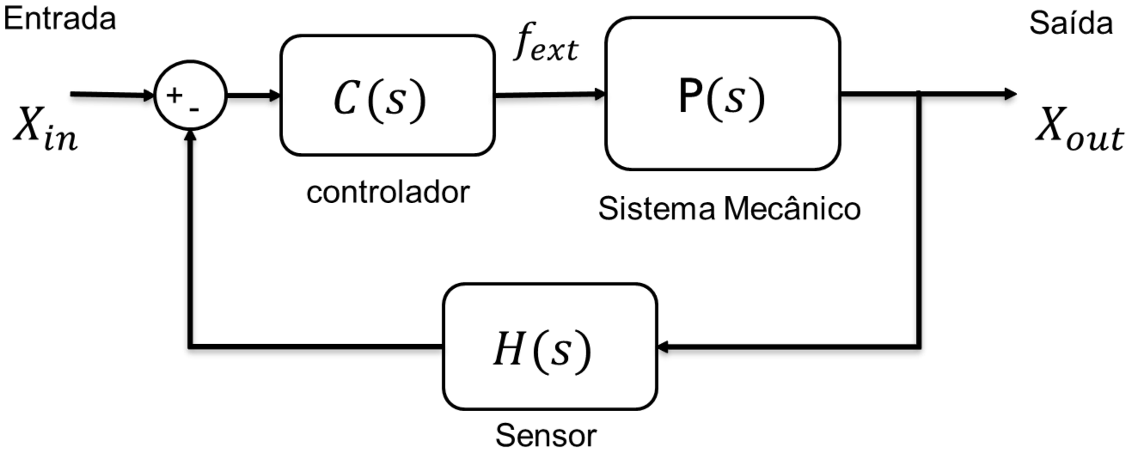
  - Frequência natural do sistema: $w_n = \sqrt{\frac{K}{M}} = 4$
  - Relação de amortecimento: $\xi = \frac{f_v}{2Mw_n} = 0{,}25$ Subamortecido!

Lembrando que esta teoria está explicada nos vídeos de Controle Clássico e foi revisada na primeira parte desta aula (3A) o objetivo aqui é partir deste resultado e fazer a simulação;

NISE, N. S. **Engenharia de Sistemas de Controle**. São Paulo: LTC 6ª ed. 2012.

Malha fechada:

- Para fechar a malha, vamos precisar de um controlador, um sensor e um bloco para fazer a subtração:



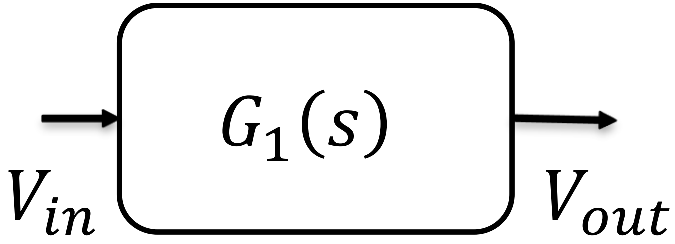
Sistema: $$P(s) = \frac{1}{0{,}1s^2 + 0{,}2s + 1{,}6}$$

Sensor: $$H(s) = 1$$

Controlador PID: $$C(s) = \frac{K_i + K_p \cdot s + K_d \cdot s^2}{s}$$

$$\text{com:}\ \begin{aligned} K_p &= 10 \cdot w_n \\ K_i &= 2 \cdot w_n \\ K_d &= 1 \cdot w_n \end{aligned}$$

Esses parâmetros definem a “sintonia” do controlador PID



$$G_1(s) = \frac{C(s)P(s)}{1 + C(s)P(s)H(s)}$$

Código do python

```
import control as ctl
import matplotlib.pyplot as plt
import numpy as np
# cria a função de transferência em malha aberta
km = 1.6;#constante da mola
massa=.1;# massa do peso
fv=0.2;#const de amortecimento
Wn = np.sqrt(km/massa); eta = (fv/massa)/(2*Wn)
print ('Wn e eta =',[Wn, eta])
Tsim=10;
numerador = [1.]
denominador = [massa, fv, km]
P_s = ctl.tf(numerador, denominador)
print(' FT malha aberta= ' ,P_s)

# controlador PID
Ki=2*Wn;Kp=10*Wn;Kd=1*Wn;
C_s=ctl.tf([Kd, Kp, Ki],[1., 0.])
print(' FT controlador= ' ,C_s)
# sensor unitário
H_s=ctl.tf([1.],[1.])
```

```
#Funcao de transf MF
#G1_s=(C_s*P_s)/(1+C_s*P_s*H_s)
G1_s=ctl.feedback(ctl.series(C_s, P_s), H_s, sign=-1)
print(' FT malha fechada= ' ,G1_s)

#calcula a resposta ao degrau
T, yout = ctl.step_response(P_s, Tsim)
T_mf, yout_mf = ctl.step_response(G1_s, Tsim)

#calcula um degrau unitário
T2=np.linspace(-1.,Tsim,1000)
degrau=np.ones_like(T2)
degrau[T2<0]=0;

#plota os resultados
plt.plot(T,yout,'b-')
plt.plot(T_mf,yout_mf,'k-')
plt.plot(T2,degrau,'r-')
plt.xlabel('tempo (s)')
plt.legend(['resposta em malha aberta','resposta em malha
 fechada','degrau unitário'])
plt.grid(); plt.title('Sistema de 2ª ordem');
```

Código do python (para uma figura bonita)

- O código apresentado é uma expansão dos usados nas aulas 1C e 2C de prática de controle;
- Assim as principais funções da toolbox de controle já foram explicadas;
  - *ctl.tf (ou control.tf)*
  - ctl.step_response (ou control.step_response)
  - ctl.series (ou control.series)
  - ctl.feedback
- Você pode acessar essas explicações nas aulas anteriores, disponíveis nesta playlist;
- As diferenças na simulação desta aula 3C são:
  - Definimos uma função de transferência de 2ª ordem para o sistema mecânico;
  - Usamos novamente o comando "ctl.tf()" para criar a função de transferência do controlador PID, que desta vez é um PID.

```
C_s=ctl.tf([Kd, Kp, Ki],[1., 0.])
```

$$C(s) = \frac{K_d \cdot s^2 + K_p \cdot s + K_i}{1s + 0}$$

Código do python, principais diferenças

```
import control as ctl
import matplotlib.pyplot as plt
import numpy as np
# cria a função de transferência em malha aberta
km = 1.6;#constante da mola
massa=.1;# massa do peso
fv=0.2;#const de amortecimento
Wn = np.sqrt(km/massa); eta = (fv/massa)/(2*Wn)
print ('Wn e eta =',[Wn, eta])
Tsim=10;
numerador = [1.]
denominador = [massa, fv, km]
P_s = ctl.tf(numerador, denominador)
print(' FT malha aberta= ' ,P_s)

# controlador PID
Ki=2*Wn;Kp=10*Wn;Kd=1*Wn;
C_s=ctl.tf([Kd, Kp, Ki],[1., 0.])
print(' FT controlador= ' ,C_s)
# sensor unitário
H_s=ctl.tf([1.],[1.])
```

Polinômio de 2º grau, $massa * s^2 + fv * s + km$, vai gerar uma FT de 2ª ordem.

Polinômios do PID: $\frac{K_d \cdot s^2 + K_p \cdot s + K_i}{1s + 0}$

Entendendo o Código em python

import control as ctl
import matplotlib.pyplot as plt
import numpy as np

$\rightarrow$ *importa as bibliotecas (toolbox) necessárias*

\# cria a função de transferência em malha aberta $\rightarrow$ *comentário*

km = 1.6;#constante da mola
massa=.1;# massa do peso
fv=0.2;#const de amortecimento

$\rightarrow$ $cria\ as\ variáveis\ km, massa, fv\ e\ dá\ valor\ a\ elas$

Wn = np.sqrt(km/massa); eta = (fv/massa)/(2*Wn) $\rightarrow$ $Calcula\ Wn\ e\ eta$

print ('Wn e eta =',[Wn, eta]) $\rightarrow$ $Mostra\ Wn\ e\ eta$

Tsim=10; $\rightarrow$ cria a variavel Tsim que usaremos para definir o tempo de simulação

numerador = [1.]
denominador = [massa, fv, km]

$\rightarrow$ $cria\ os\ vetores\ da\ FT\ em\ MA$

P_s = ctl.tf(numerador, denominador) $\rightarrow$ $usa\ os\ vetores$ para criar P_s

print(' FT malha aberta= ' ,P_s) $\rightarrow$ $Mostra\ a\ função\ de\ transferência\ em\ MA$

\# controlador PID
Ki=2*Wn;Kp=10*Wn;Kd=1*Wn;
C_s=ctl.tf([Kd, Kp, Ki],[1., 0.])

$\rightarrow$ $cria\ a\ fuñção\ de\ transferencia\ do\ controlador\ C\_s$

print(' FT controlador= ' ,C_s) $\rightarrow$ $Mostra\ a\ função\ de\ transferênciado\ controlador\ C\_s$

\# sensor unitário
H_s=ctl.tf([1.],[1.]) $\rightarrow$ $cria\ a\ fuñção\ de\ transferencia\ do\ sensor\ H\_s = 1/1$

Entendendo o Código em python

```
#Funcao de transf MF
#G1_s=(C_s*P_s)/(1+C_s*P_s*H_s)
G1_s=ctl.feedback(ctl.series(C_s, P_s), H_s, sign=-1)
print(' FT malha fechada= ' ,G1_s)
```

→ *Constroi a FT em Malha fechada a partir das demais FT*

→ *Mostra a função de transferência em MF*

```
#calcula a resposta ao degrau
T, yout = ctl.step_response(P_s, Tsim)
T_mf, yout_mf = ctl.step_response(G1_s, Tsim)
```

→ *calcula a resposta ao degrau (step) para Tsim e P_s*

→ *calcula a resposta ao degrau (step) para Tsim e G1_s*

```
#calcula um degrau unitário
T2=np.linspace(-1.,Tsim,1000)
degrau=np.ones_like(T2)
degrau[T2<0]=0;
```

→ *calcula um degrau (step) entre -1 e Tsim, para comparar*

```
#plota os resultados
plt.plot(T,yout,'b-')
plt.plot(T_mf,yout_mf,'k-')
plt.plot(T2,degrau,'r-')
plt.xlabel('tempo (s)')
plt.legend(['resposta em malha aberta','resposta em malha fechada','degrau unitário'])
plt.grid(); plt.title('Sistema de 2ª ordem');
```

faz o *gráfico* destas 3 curvas no mesmo gráfico;

Código do Python – resultado:

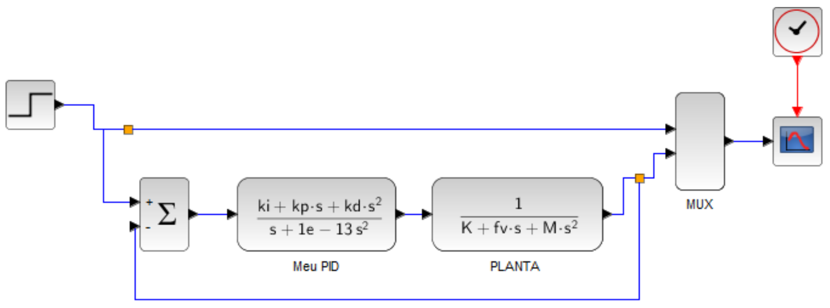
- O resultado é vista a seguir;
- Escolhemos plotar o resultado em malha aberta e fechada no mesmo gráfico;
- Note as cores escolhidas no código;

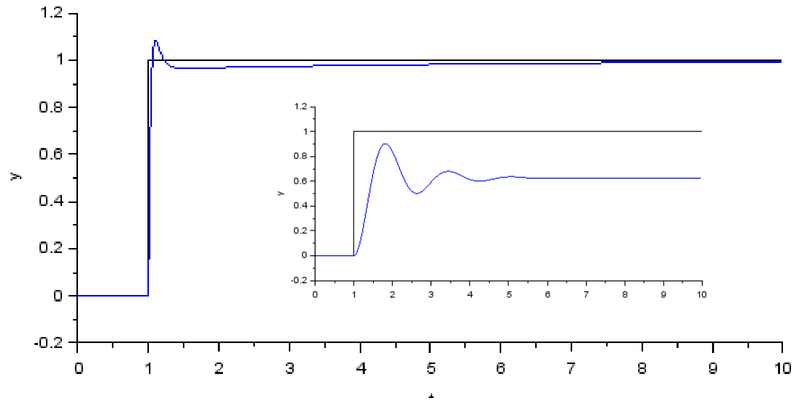
```
:
plt.plot(T,yout,'b-')
plt.plot(T_mf,yout_mf,'k-')
plt.plot(T2,degrau,'r-')
:
```

Comparando Xcos, Scilab e Python:

- Note que o resultado é mesmo. São três caminhos bem distintos. Então, qual deles lhe parece mais adequado?

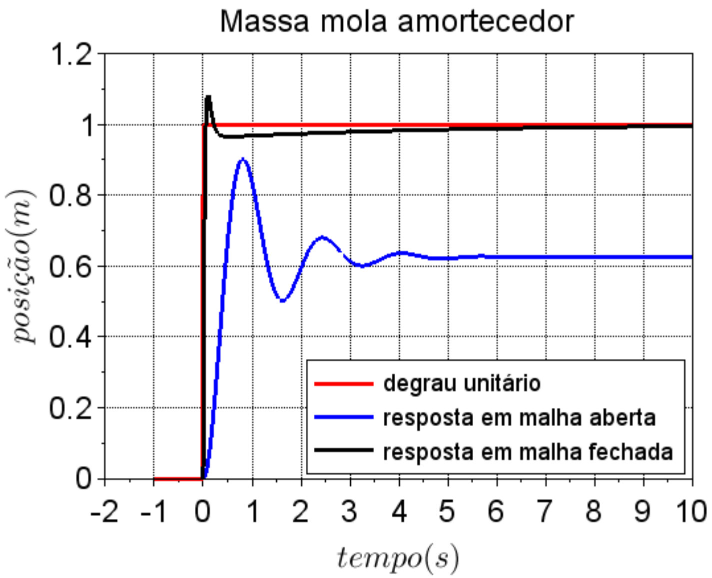
Parte A da aula: Xcox





Parte B da aula: Toolbox do scilab

```
:
// cria a função de transferência do controlador PID
Ki=2*Wn;Kp=10*Wn;Kd=1*Wn;
C_s=syslin('c',poly([Ki Kp Kd],'s','c'),poly([0 1],'s','c'))
disp('FT controlador',C_s)
:
```



Parte C da aula: Toolbox do python

```
:
T, yout = ctl.step_response(P_s, Tsim)
T_mf, yout_mf = ctl.step_response(G1_s, Tsim)
:
```

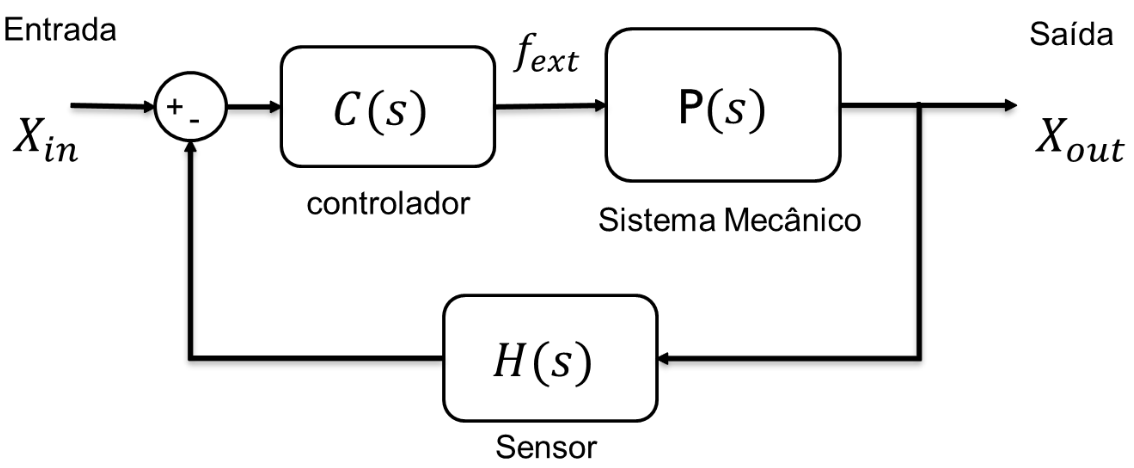
Exercícios:

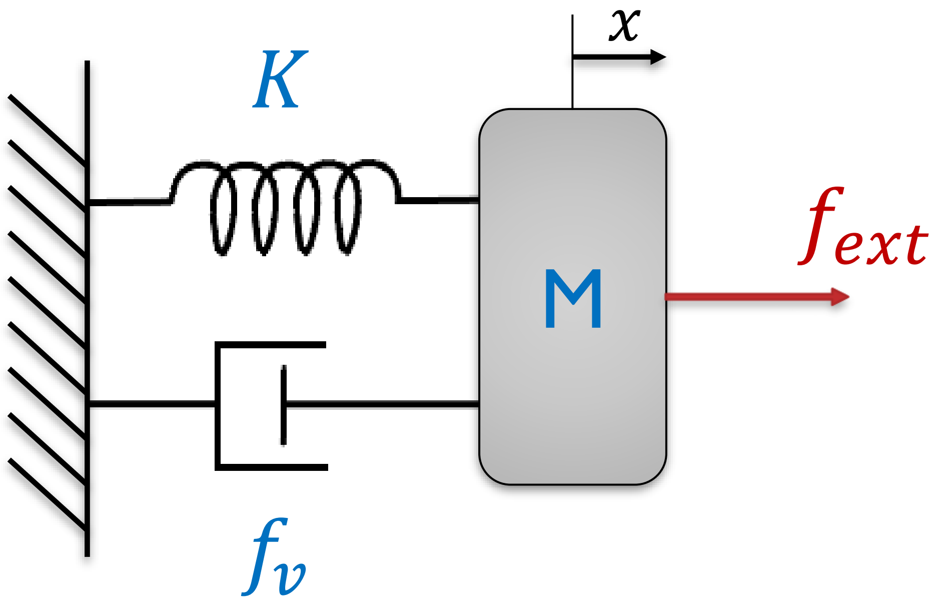
- 1- Refaça as simulações (MA e MF) com a toolbox de controle do python para o mesmo sistema, porém com os seguintes comportamentos:
    a. Superamortecido ($\xi > 1,\ sugestão\colon f_v = 1{,}6$)
    b. Criticamente amortecido ($\xi = 1,\ sugestão\colon f_v = 0{,}8$)
    c. Não amortecido ($\xi = 0 \Rightarrow f_v = 0$)
- Experimente manter a sintonia do PID, para comparar.
- Os comportamentos observados condizem com o esperado? Por que?

Exercícios:

- 2- Agora refaça as 4 simulações em malha fechada, a do exemplo e as do exercício 1 porém aumente o fator integral do controlador PID para $K_i = 10 \cdot w_n$. O que aconteceu com a resposta ao degrau, especialmente com a diferença desta para o degrau da entrada?

Para o caso do exemplo da aula, você deve encontrar este resultado:





Python

Simulação de controle 100% online e free!

Controle na prática #03C: Simulação de sistema mecânico e controlador PID com a toolbox de controle do Python.