*Lembrand*o:

Dado um sistema linear contínuo, no domínio do tempo, com $x = x(t)$ o vetor de estado do sistema, A a matriz da dinâmica do sistema, B a matriz de controle e $u = u(t)$ o vetor de controle, na forma

$$\dot{x} = Ax + Bu$$

para malha aberta, temos $u = 0$, portanto

$$\dot{x} = Ax$$

no domínio da frequência, com $X = X(s)$

$$Xs = AX$$
$$Xs - AX = 0$$
$$(sI - A)X = 0$$

para X não nulo temos

$$\det(sI - A) = 0$$

ou seja o determinante de $(sI - A)$ deve ser nulo, desta forma encontramos os autovalores da matriz A, que nos fornecem os polos de malha aberta do sistema.

$$det(sI - A) = s^n + a_{n-1}s^{n-1} + a_{n-2}s^{n-2} + \cdots + a_0 = (s - p_1)(s - p_2) \ldots (s - p_n) = 0$$

com n a ordem do sistema (número de estados), a_i os coeficientes do polinômio característico e p_i os polos de malha aberta.

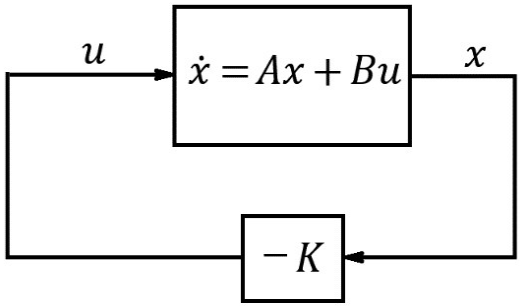
Desejamos agora projetar um controlador para que a dinâmica em malha fechada possua uma atenuação da vibração.

Seja o sistema dinâmico linear e invariante no tempo da forma

$$\dot{x} = Ax + Bu$$

E seja o controle $u(t)$ na forma

$$u = -Kx$$



com K a matriz dos ganhos do controlador a ser determinada, e sua dimensão determinada pelas dimensões de x e u, isto é, K terá o número de linhas de dim(u) e o número de colunas de dim(x), exemplo, para o caso do controle do sistema massa mola amortecedor descrito anteriormente, com duas entradas f_1 e f_2, temos

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} \quad u = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix}$$

portanto

$$K = \begin{bmatrix} k_1 & k_2 & k_3 & k_4 \\ k_5 & k_6 & k_7 & k_8 \end{bmatrix}$$

substituindo na lei de controle

$$u = -Kx \quad \rightarrow \quad \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} = -\begin{bmatrix} k_1 & k_2 & k_3 & k_4 \\ k_5 & k_6 & k_7 & k_8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \dot{x}_1 \\ \dot{x}_2 \end{bmatrix}$$

k_i são as constantes do controlador a serem determinadas.

Voltando na equação do sistema dinâmico e substituindo a lei de controle, temos

$$\dot{x} = Ax - BKx$$
$$\dot{x} = (A - BK)x$$

no domínio da frequência, com $X = X(s)$ e $U = U(s)$

$$Xs = (A - BK)X$$
$$Xs - (A - BK)X = 0$$
$$\big(sI - (A - BK)\big)X = 0$$
$$(sI - A + BK)X = 0$$

para

$$X \neq 0$$

então

$$det(sI - A + BK) = 0$$

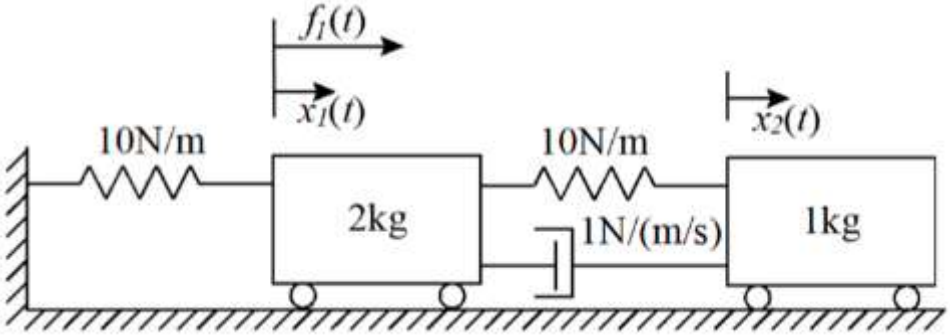
que são os polos de malha fechada. Pode-se então forçar que $det(sI - A + BK)$ coincida com uma dinâmica desejada a ser projetada, fazendo

$$det(sI - A + BK) = (s - p_1)(s - p_2)(s - p_3) \dots (s - p_i)$$

na equação acima os valores dos polos (p_1, p_2, etc) são conhecidos (projetados), ficando apenas como incógnitas os valores das constantes k_i da matriz K.

Observação: Geralmente, na escolha dos polos de malha fechada, deve respeitar a dinâmica de malha aberta, para que a estratégia de controle não se torne muito agressiva, como veremos em exemplo a seguir.

Para o exercício proposto anteriormente



temos o sistema dinâmico, $\dot{x} = Ax + Bu$, com

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \ddot{x}_1 \\ \ddot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -10 & 5 & -0.5 & 0.5 \\ 10 & -10 & 1 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0.5 \\ 0 \end{bmatrix} f_1$$

sendo

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \dot{x}_1 \\ \dot{x}_2 \end{bmatrix}, \qquad u = [f_1], \qquad A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -10 & 5 & -0.5 & 0.5 \\ 10 & -10 & 1 & -1 \end{bmatrix} \quad \text{e} \quad B = \begin{bmatrix} 0 \\ 0 \\ 0.5 \\ 0 \end{bmatrix}$$

podemos obter os polos da dinâmica de malha aberta, conforme exposto acima, por $det(sI - A) = 0$

$$det(sI - A) = det(\begin{bmatrix} s & 0 & -1 & 0 \\ 0 & s & 0 & -1 \\ 10 & -5 & s + 0.5 & -0.5 \\ -10 & 10 & -1 & s + 1 \end{bmatrix})$$

para tanto podemos utilizar a toolbox de simbólico do python, digitando diretamente na linha de comando

```
import numpy as np
import sympy as sp
A = np.array([[0,0,1,0],[0,0,0,1],[-10,5,-0.5,0.5],[10,-10,1,-1]])
B = np.array([[0],[0],[0.5],[0]])
s = sp.symbols('s')
sp.Matrix(s*np.identity(4)-A).det()
```

obtendo

```
1.0*s**4 + 1.5*s**3 + 20.0*s**2 + 5.0*s + 50.0
```

portanto

$$det(sI - A) = 0 \quad \rightarrow \quad s^4 + 1.5s^3 + 20s^2 + 5s + 50 = 0$$

podemos agora encontrar as raízes do polinômio acima utilizando

```
np.roots([1,1.5,20,5,50])
```

retornando

```
[-0.72909881+4.05781174j -0.72909881-4.05781174j -0.02090119+1.71498858j
 -0.02090119-1.71498858j]
```

que são os polos de malha aberta do sistema dinâmico.

A ideia aqui é seguir a teoria e a prática juntas, e não, encontrar de forma automática o resultado. Sim poderíamos encontrar diretamente os polos do sistema dinâmico utilizando o comando

```
print(np.linalg.eigvals(A))
```

retornando o mesmo resultado obtido anteriormente

```
[-0.72909881+4.05781174j -0.72909881-4.05781174j -0.02090119+1.71498858j
 -0.02090119-1.71498858j]
```

temos portanto, os polos de malha aberta em aproximadamente

$$p_1 = -0.0209 + 1.715j \quad p_2 = -0.0209 - 1.715j \quad p_3 = -0.729 + 4.058j \quad p_4 = -0.729 - 4.058j$$

podemos reescrever o polinômio original, na forma

$$s^4 + 1.5s^3 + 20s^2 + 5s + 50 = (s - p_1)(s - p_2)(s - p_3)(s - p_4) = 0$$

de fato, podemos verificar em python

```
p1=-0.0209+1.715j; p2=-0.0209-1.715j; p3=-0.729+4.058j; p4=-0.729-4.058j
print(sp.simplify((s-p1)*(s-p2)*(s-p3)*(s-p4)))
```

```
1.0*s**4 + 1.4998*s**3 + 20.00141121*s**2 + 4.99949296798*s + 50.0047354841371
```

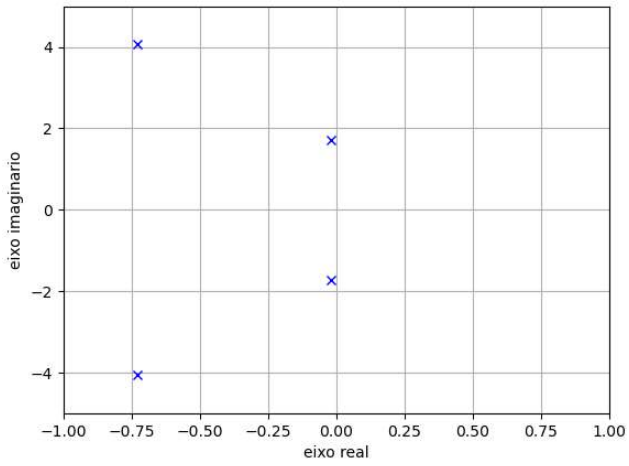
e os erros são devido ao truncamento dos valores reais dos polos.

Plotando os polos do sistema

```
import numpy as np
import matplotlib.pyplot as plt
p1=-0.0209+1.715j; p2=-0.0209-1.715j; p3=-0.729+4.058j; p4=-0.729-4.058j
plt.plot(np.real([p1,p2,p3,p4]),np.imag([p1,p2,p3,p4]),'bx')
plt.axis ([-1,1,-5,5])
plt.xlabel('eixo real')
plt.ylabel('eixo imaginario')
plt.grid(True)
plt.show()
```

Outro código para plotar os polos em malha aberta de A

```
import numpy as np
import matplotlib.pyplot as plt
m1=2.0; m2=1.0; k1=10.0; k2=10.0; b1=0; b2=1.0
A = np.matrix([[0,0,1,0],[0,0,0,1],[-(k1+k2)/m1,k2/m1,-(b1+b2)/m1,b2/m1],[k2/m2,-k2/m2,b2/m2,-b2/m2]])
poles = np.linalg.eigvals(A)
plt.plot(np.real(poles),np.imag(poles),'bx')
plt.axis ([-1,1,-5,5])
plt.xlabel('eixo real')
plt.ylabel('eixo imaginario')
plt.grid(True)
plt.show()
```



Após determinamos os polos de malha aberta podemos projetar a lei de controle baseada no sistema obtido no espaço de estado. Como

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \dot{x}_1 \\ \dot{x}_2 \end{bmatrix}, \qquad u = [f_1], \qquad A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -10 & 5 & -0.5 & 0.5 \\ 10 & -10 & 1 & -1 \end{bmatrix} \quad \text{e} \quad B = \begin{bmatrix} 0 \\ 0 \\ 0.5 \\ 0 \end{bmatrix}$$

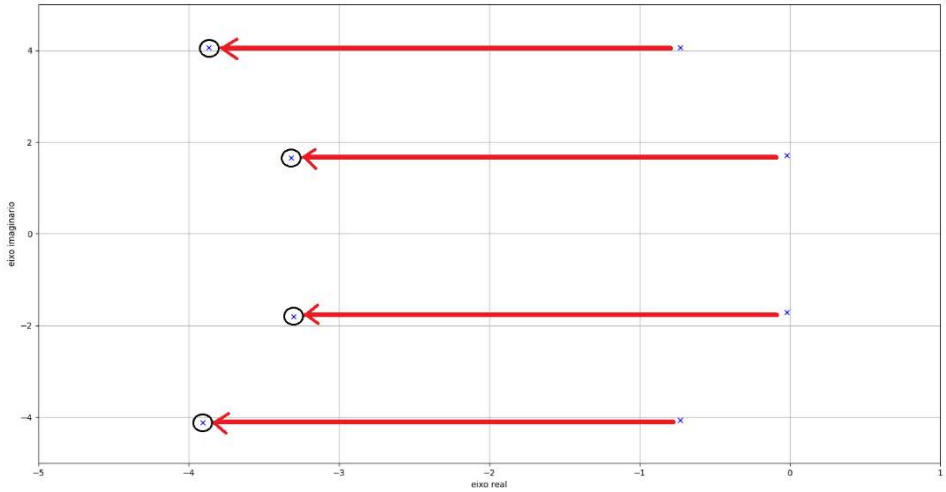
portanto a matriz de controle K, possuirá 1 linha ($dim(u)$) e 4 colunas ($dim(x)$), e terá a forma

$$K = [k_1 \quad k_2 \quad k_3 \quad k_4]$$

sendo a lei de controle

$$u = -Kx \qquad \rightarrow \qquad [f_1] = -[k_1 \quad k_2 \quad k_3 \quad k_4]\begin{bmatrix} x_1 \\ x_2 \\ \dot{x}_1 \\ \dot{x}_2 \end{bmatrix}$$

para determinar os valores das constantes do controlador precisamos determinar os polos em malha fechada. Uma maneira de projetar os novos polos de malha fechada é olhar para os polos de malha aberta e respeitando a dinâmica do sistemas e suas frequências, mexemos apenas na parte real, afastando-a do *eixo y* conforme figura abaixo, de forma a mantermos as mesmas frequências do sistema



para facilitar o estudo, vamos colocar todos os polos de malha fechada em '*-4*', isto é, $p_1 = p_2 = p_3 = p_4 = -4$, portanto

$$det(sI - A + BK) = (s - p_1)(s - p_2)(s - p_3)(s - p_4)$$
$$det(sI - A + BK) = (s + 4)(s + 4)(s + 4)(s + 4)$$

```
import sympy as sp
s = sp.symbols('s')
p1=-4; p2=-4; p3=-4; p4=-4
print(sp.expand((s-p1)*(s-p2)*(s-p3)*(s-p4)))
```

obtendo

```
s**4 + 16*s**3 + 96*s**2 + 256*s + 256
```

ou seja

$$det\left(\begin{bmatrix} s & 0 & 0 & 0 \\ 0 & s & 0 & 0 \\ 0 & 0 & s & 0 \\ 0 & 0 & 0 & s \end{bmatrix} - \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -10 & 5 & -0.5 & 0.5 \\ 10 & -10 & 1 & -1 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0.5 \\ 0 \end{bmatrix}[k_1 \quad k_2 \quad k_3 \quad k_4]\right) = s^4 + 16s^3 + 96s^2 + 256s + 256$$

$$det\left(\begin{bmatrix} s & 0 & -1 & 0 \\ 0 & s & 0 & -1 \\ 10 & -5 & s + 0.5 & -0.5 \\ -10 & 10 & -1 & s + 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ k_1 & k_2 & k_3 & k_4 \end{bmatrix}\right) = s^4 + 16s^3 + 96s^2 + 256s + 256$$

$$det\left(\begin{bmatrix} s & 0 & -1 & 0 \\ 0 & s & 0 & -1 \\ 10 & -5 & s + 0.5 & -0.5 \\ k_1 - 10 & k_2 + 10 & k_3 - 1 & s + k_4 + 1 \end{bmatrix}\right) = s^4 + 16s^3 + 96s^2 + 256s + 256$$

Fazendo isso agora podemos determinar os ganhos do controlador, pela equação acima.
Podemos utilizar o toolbox simbólico do python para determinar $det(sI - A + BK)$, conforme abaixo

```
import numpy as np
import sympy as sp
```

```
m1=2.0; m2=1.0; k1=10.0; k2=10.0; c1=0; c2=1.0;
A = np.matrix([[0,0,1,0],[0,0,0,1],[-(k1+k2)/m1,k2/m1,-(c1+c2)/m1,c2/m1],[k2/m2,-
k2/m2,c2/m2,-c2/m2]])
B=[[0],[0],[1/m1],[0]]
s = sp.symbols('s')
k1,k2,k3,k4 = sp.symbols(['k1','k2','k3','k4'])
K=[[k1,k2,k3,k4]]
print(sp.collect(sp.Matrix(s*np.identity(4)-A+np.dot(B,K)).det(),s))
```

```
5.0*k1 + 5.0*k2 + 1.0*s**4 + s**3*(0.5*k3 + 1.5) + s**2*(0.5*k1 + 0.5*k3 + 0.5*k4 +
20.0) + s*(0.5*k1 + 0.5*k2 + 5.0*k3 + 5.0*k4 + 5.0) + 50.0
```

portanto

$$s^4 + (0.5k_3 + 1.5)s^3 + (0.5k_1 + 0.5k_3 + 0.5k_4 + 20)s^2 + (0.5k_1 + 0.5k_2 + 5k_3 + 5k_4 + 5)s + 5k_1 + 5k_2 + 50 = s^4 + 16s^3 + 96s^2 + 256s + 256$$

$$\begin{gathered} 0.5k_3 + 1.5 = 16 \\ 0.5k_1 + 0.5k_3 + 0.5k_4 + 20 = 96 \\ 0.5k_1 + 0.5k_2 + 5k_3 + 5k_4 + 5 = 256 \\ 5k_1 + 5k_2 + 50 = 256 \end{gathered}$$

Que representa um sistema de equações lineares na forma

$$\begin{bmatrix} 0 & 0 & 0.5 & 0 \\ 0.5 & 0 & 0.5 & 0.5 \\ 0.5 & 0.5 & 5 & 5 \\ 5 & 5 & 0 & 0 \end{bmatrix} \begin{bmatrix} k_1 \\ k_2 \\ k_3 \\ k_4 \end{bmatrix} = \begin{bmatrix} 14.5 \\ 76 \\ 251 \\ 206 \end{bmatrix}$$

que podemos colocar na forma matricial combinada

$$\begin{bmatrix} 0 & 0 & 0.5 & 0 & 14.5 \\ 0.5 & 0 & 0.5 & 0.5 & 76 \\ 0.5 & 0.5 & 5 & 5 & 251 \\ 5 & 5 & 0 & 0 & 206 \end{bmatrix}$$

e resolver usando a forma escalonada reduzida

```
from sympy import Matrix
print(Matrix([[0,0,0.5,0,14.5],[0.5,0,.5,.5,76],[.5,.5,5,5,251],[5,5,0,0,206]]).rref(
))
```

```
(Matrix([
[1, 0, 0, 0, 105.92],
[0, 1, 0, 0, -64.72],
[0, 0, 1, 0,   29.0],
[0, 0, 0, 1,  17.08]]), (0, 1, 2, 3))
```

portanto

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 105.92 \\ 0 & 1 & 0 & 0 & -64.72 \\ 0 & 0 & 1 & 0 & 29 \\ 0 & 0 & 0 & 1 & 17.08 \end{bmatrix}$$

isto é

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} k_1 \\ k_2 \\ k_3 \\ k_4 \end{bmatrix} = \begin{bmatrix} 105.92 \\ -64.72 \\ 29 \\ 17.08 \end{bmatrix}$$

$$k_1 = 105.92$$
$$k_2 = -64.72$$
$$k_3 = 29$$
$$k_4 = 17.08$$

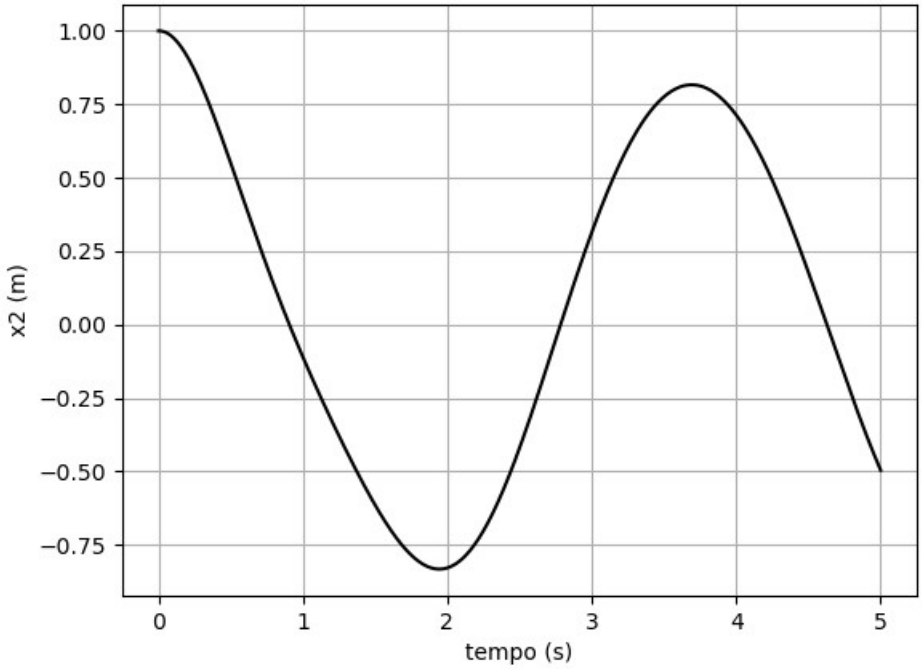
encontramos portanto a matriz de ganhos do controlador

$$K = [105.92 \quad -64.72 \quad 29 \quad 17.08]$$

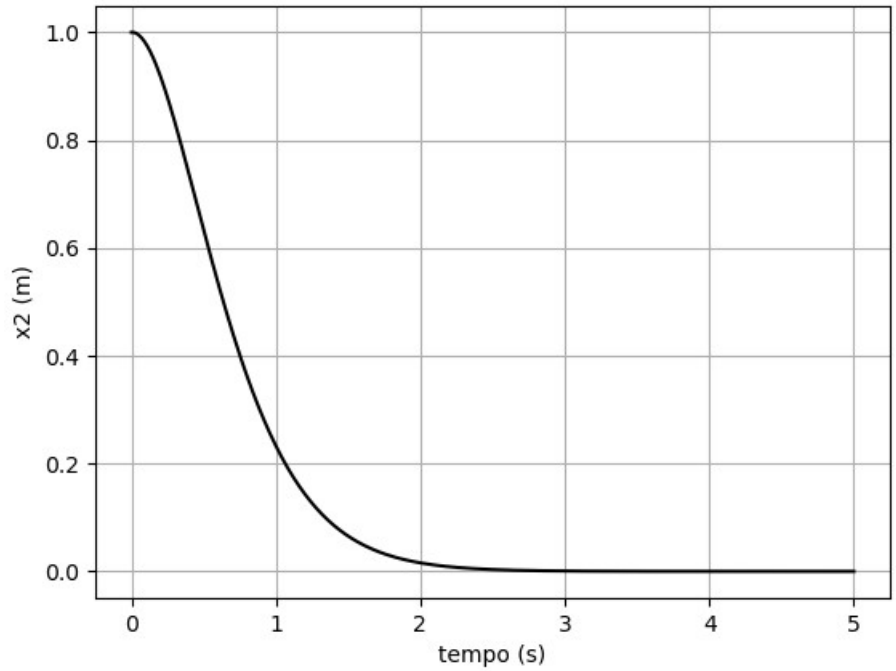
Simulando a dinâmica em malha fechada com

$$u = -Kx$$

```
import numpy as np

dx = lambda x, u, dt: (A@x+B@[u])*dt

m1=2.0; m2=1.0; k1=10.0; k2=10.0; c1=0; c2=1.0;
A = np.array([[0,0,1,0],[0,0,0,1],[-(k1+k2)/m1,k2/m1,-(c1+c2)/m1,c2/m1],[k2/m2,-
k2/m2,c2/m2,-c2/m2]])
B = np.array([[0],[0],[1/m1],[0]])
C = np.array([0,1,0,0])
K = np.array([105.92,-64.72,29,17.08])

t, tf, dt, u, x = 0, 5, .01, 0, np.array([[0.5],[1],[0],[0]]),
X, U, T = x, u, t

for i in range(int((tf-t)/dt)):
    u = -K@x
    t, x = t + dt, x + dx(x,u,dt)
    X = np.append(X,x,axis=1)
    U = np.append(U,u)
    T = np.append(T,t)

Y = C@X
import matplotlib.pyplot as plt
plt.plot(T,Y,'k')
plt.xlabel('tempo (s)')
plt.ylabel('x2 (m)')
plt.grid(True)
plt.show()
```



Simulação em malha aberta



Simulação em malha fechada, com controle $u = -Kx$

```
from control import acker, place
K = acker(A, B, np.array([-4,-4,-4,-4]))

matrix([[105.92, -64.72,  29. ,  17.08]])

K = place(A, B, np.array([-3.98,-3.99,-4,-4.01]))
matrix([[105.60632599, -64.66264583,  28.95999977,  16.95347153]])
```