Atividade - Least Squares

April 10, 2019

Autor: Bruno Almeida da Silva - Matrícula: 190061197

1 Oscilador Harmônico Forçado

Este trabalho é um exemplo da aplicação do modelo ARX de Identificação de Sistemas para resolver a solução transiente de um Oscilador Harmônico Forçado em dois regimes distintos:

1. A Estimação dos parâmetros foi feita no regime de Subamortecimento;
2. Validou-se o modelo no regime de Superamortecimento.

As soluções foram retiradas da Wikipedia.

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        from control import *
        %matplotlib inline
```

```
In [2]: sns.set_color_codes(palette='dark')
        sns.set(style="whitegrid")
```

```
In [3]: np.random.seed(0)
```

O Oscilador Harmônico Forçado mais geral é descrito pela Equação

$$\frac{d^2q}{dt^2} + 2\zeta\frac{dq}{dt} + q = 0$$

onde ζ é o fator de amortecimento e q é uma coordenada generalizada. Adicionando uma força do tipo $f(t) = cos(\omega t)$, tem-se

$$\frac{d^2q}{dt^2} + 2\zeta\frac{dq}{dt} + q = cos(\omega t)$$

Para a solução subamortecida ($\zeta < 1$), utilizou-se a seguinte Equação para estimação dos parâmetros:

$$u(t) = u_{amp}e^{-\zeta t}cos\left(t\sqrt{1-\zeta^2}\right)$$

Para a solução superamortecida ($\zeta > 1$), utilizou-se a seguinte Equação para os cálculos para validação do modelo:

$$u(t) = u_{amp}e^{-\zeta t}e^{\left(-t\sqrt{\zeta^2-1}\right)}$$

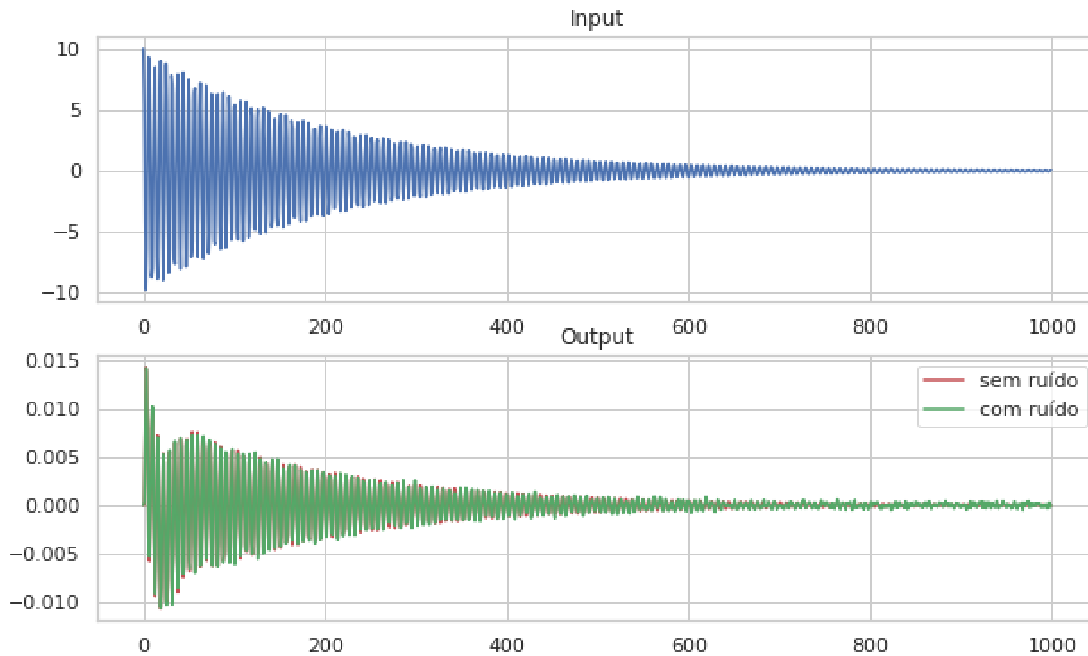
```
In [4]: # Constantes do sistema
        m = 0.1
        k = 10
        c = 0.01
        zt = c/(2*np.sqrt(m*k))
        Ts = 0.01
        print(zt)
```

```
0.005
```

```
In [5]: th = np.array([[-1.895, 0.9048, 0.0004833, 0.0004675]]).transpose()
        print(th)
```

```
[[-1.895e+00]
 [ 9.048e-01]
 [ 4.833e-04]
 [ 4.675e-04]]
```

```
In [6]: N = 1000
        t = np.arange(0,N)
        sig = 0.0005

In [7]: uamp = 10
        yr = 1
        u = np.array(uamp*np.exp(-zt*t)*np.cos(np.sqrt(1-zt**2)*t)).reshape((N,1))
        y = np.array(np.zeros((N,1)))

In [8]: for k in range(2, N):
            y[k] = -th[0]*y[k-1] - th[1]*y[k-2] + th[2]*u[k-1] + th[3]*u[k-2]

In [9]: UTRA = u
        YTRA = y + sig*np.random.rand(N,1) - sig*np.random.rand(N,1)

In [10]: fig,axes = plt.subplots(nrows=2, ncols=1, figsize=(10,6))

         axes[0].set_title("Input")
         axes[0].plot(t,UTRA,color='b')

         axes[1].set_title("Output")
         axes[1].plot(t,y,color='r',label='sem ruído')
         axes[1].plot(t,YTRA,color='g',label='com ruído')
         axes[1].legend(loc=1)

         fig.savefig("funcoes.png",dpi=500, bbox_inches='tight')
```



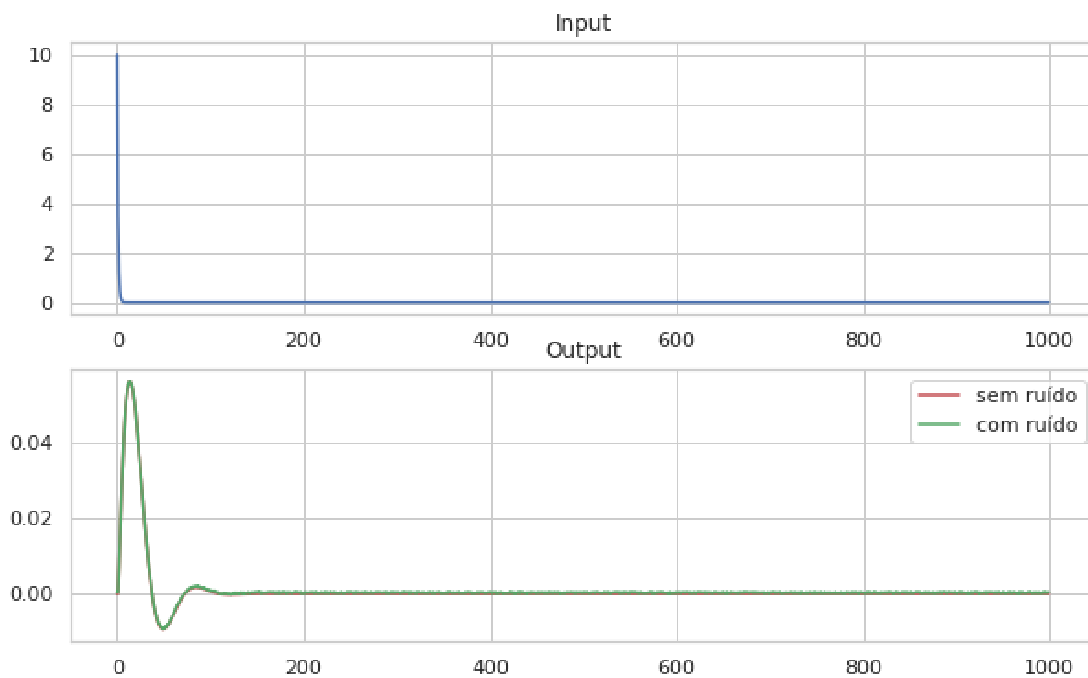

```
In [11]: # Novas constantes para validação
         c = 20
         zt = c/(2*np.sqrt(m*k))
         print(zt)
```

```
1.0005003753127737
```

```
In [12]: # Função de Validação
         u = np.array(uamp*np.exp(-zt*t)*np.exp(-1*np.sqrt(zt**2-1)*t)).reshape((N,1))
         y = np.array(np.zeros((N,1)))
```

```
In [13]: for k in range(2,N):
             y[k] = -th[0]*y[k-1] - th[1]*y[k-2] + th[2]*u[k-1] + th[3]*u[k-2]
         UVAL = u
         YVAL = y + sig*np.random.rand(N,1)
```

```
In [14]: fig,axes = plt.subplots(nrows=2, ncols=1, figsize=(10,6))

         axes[0].set_title("Input")
         axes[0].plot(t,UVAL,color='b')

         axes[1].set_title("Output")
         axes[1].plot(t,y,color='r',label='sem ruído')
         axes[1].plot(t,YVAL,color='g',label='com ruído')
         axes[1].legend(loc=1)

         fig.savefig("funcoes-2.png",dpi=500, bbox_inches='tight')
```



Verificou-se que aumentando a ordem da entrada em 1, com o modelo ARX de ordem 2 na entrada e 3 na saída, houve melhora nos resultados.

```
In [15]: Phi = np.concatenate((-YTRA[2:-1] -YTRA[1:-2], \
                               -YTRA[0:-3], UTRA[1:-2], UTRA[0:-3]), axis=1)
         PhiVAL = np.concatenate((-YVAL[2:-1] -YVAL[1:-2], \
                                  -YVAL[0:-3], UVAL[1:-2], UVAL[0:-3]), axis=1)
         Y1 = YTRA[2:-1]
         Y2 = YVAL[2:-1]

         print(Phi.shape)
```

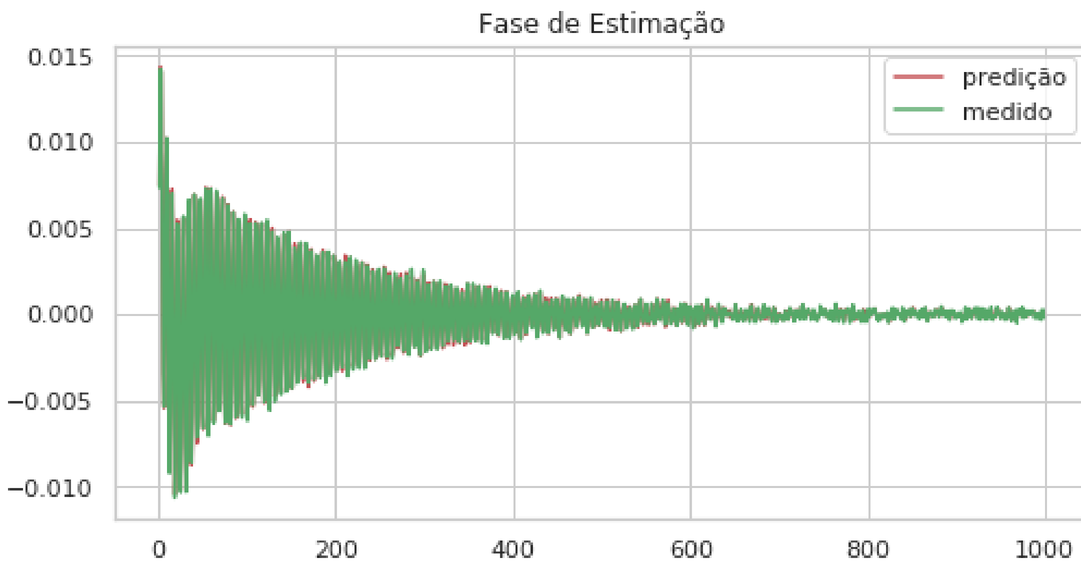
```
(997, 4)
```

```
In [16]: # Precisa obter a pseudo inversa (pinv)
         th_hat = np.dot(np.linalg.pinv(np.dot(Phi.conj().transpose(),Phi)), \
                         np.dot(Phi.conj().transpose(),Y1))
```

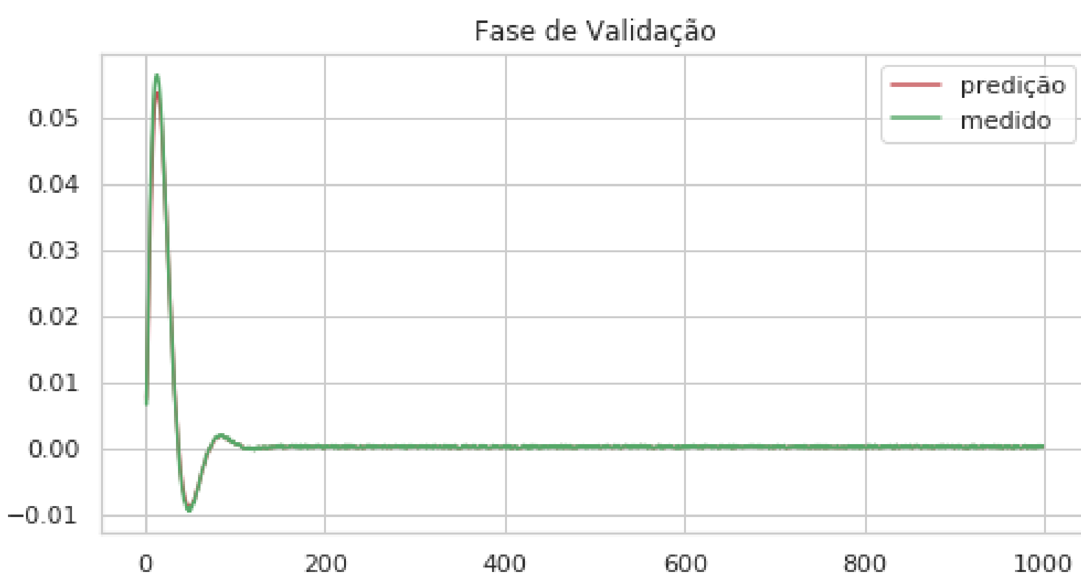
```
In [17]: yhat_TRA_OSA = np.dot(Phi,th_hat)
         yhat_VAL_OSA = np.dot(PhiVAL,th_hat)
```

```
In [18]: fig,axes = plt.subplots(figsize=(8,4))

         axes.set_title("Fase de Estimação")
         axes.plot(t[2:999],yhat_TRA_OSA,color='r',label='predição')
         axes.plot(t[2:999],YTRA[2:999],color='g',label='medido')
```

```
axes.legend(loc=1)

fig.savefig("resultado_training.png",dpi=500, bbox_inches='tight')
```



In [19]:
```
fig,axes = plt.subplots(figsize=(8,4))

axes.set_title("Fase de Validação")
axes.plot(t[2:999],yhat_VAL_OSA,color='r',label='predição')
axes.plot(t[2:999],YVAL[2:999],color='g',label='medido')
axes.legend(loc=1)

fig.savefig("resultado_validate.png",dpi=500, bbox_inches='tight')
```



```
In [20]: fig, [ax1, ax2] = plt.subplots(2, 1, sharex=True, figsize=(8,8))
         ax1.xcorr(Y1[:,0]-yhat_TRA_OSA[:,0], UTRA[2:999,0], \
                   usevlines=True, maxlags=500, normed=True, lw=2)
         ax1.grid(True)
         ax1.axhline(0, color='black', lw=2)

         ax2.acorr(Y1[:,0]-yhat_TRA_OSA[:,0], usevlines=True, normed=True, maxlags=500, lw=2)
         ax2.grid(True)
         ax2.axhline(0, color='black', lw=2)

         fig.savefig("resultado_corr.png",dpi=500, bbox_inches='tight')
```