UNIVERSIDADE FEDERAL FLUMINENSE -UFF

Animações com Matplotlib

Autor Bruno da Silva Machado

12 de junho de 2017

Introdução

Matplotlib é uma biblioteca de plotagem para o linhagem de programação python e uma extensão do numpy (numerical python). Ele fornece uma API orientada a objetos incorporada em aplicações que precisõa utilizar gráficos. Escolhemos o seu uso por ser uma ferramenta integrada com o python e por sua simplicidade. Nesse trabalho a usaremos para criar animações de modo que possamos ver o comportamento de um pendulo ao longo do tempo.

Animações com Matplotlib

As ferramentas de animação se centram em torno da matplotlib.animation.Animation classe base, que fornece uma estrutura em torno da qual a funcionalidade de animação é construída. As principais interfaces são TimedAnimation e FuncAnimation , nas quais podem ser lidas na documentação $(http://matplotlib.sourceforge.net/api/animation_api.html)$.Nesse trabalho vamos usar a FuncAnimation ferramenta, comumente a mais usada nas animações

Primeiro usaremos FuncAnimation para fazer uma animação básica de um pendulo simples sujeito as seguintes hipóteses:

1. O braço é formado por um fio não flexível que se mantém sempre com o mesmo formato e comprimento.
2. Toda a massa, m , do pêndulo está concentrada na ponta do braço a uma distância constante L do eixo.
3. Não existem outras forças a atuar no sistema senão a gravidade e a força que mantém o eixo do pêndulo fixo. (O movimento é portanto conservativo).
4. O pêndulo realiza um movimento bidimensional no plano xy.

equação do pendulo simples

$$\frac{d^2\theta}{dt^2} + \frac{g}{L}\sin\theta = 0. \tag{1}$$

e por fim animamos um pendulo forçado que é um pêndulo amortecido o torque de uma força periódica $F = F_0\sin(\omega_D t)$,onde ω_D t é a frequência angular da força externa, a segunda lei de Newton para rotação deste movimento torna-se:

$$\ddot{\theta}A\sin(\omega_D t) - \gamma\dot{\theta} - \omega^2\dot{\theta}\sin(\theta) \tag{2}$$

Código em Python

```
fig = plt.figure(facecolor='white')
line1 ,=XxT.plot([],[], 'k-',lw=2)
```

Aqui criamos uma janela de figura, com um único eixo na e criamos nosso objeto de linha que será modificado na animação. Observe que aqui simplesmente traçamos uma linha vazia. Em seguida, criaremos as funções que fazem a animação acontecer. init() É a função que será chamada para criar o quadro base sobre o qual a animação ocorre. Aqui, usamos apenas uma função simples que define os dados da linha para nada. É importante que esta função retorne o objeto de linha, pois isso indica ao animador quais objetos no enredo para atualizar após cada quadro:

```
def init():
    line1.set_data([],[])
    line2.set_data([],[])
    line3.set_data([],[])
    line4.set_data([],[])
    return line1,line2,line3,line4,
```

A próxima peça é a função de animação. É preciso um único parâmetro, o número do quadro i

```
#funcao animacao
def animate(i):
  a = t[:i]
  b = x[:i]
  c = v[:i]
  d = e[:i]


  line1.set_data(a,b)
  line2.set_data(a,c)
  line3.set_data(b,c)
  line4.set_data(a,d)
  return line1,line2,line3,line4,
```

Note-se que novamente aqui devolvemos uma tupla dos objetos do da cena que foram modificados. Isso diz ao quadro de animação quais partes do gráfico devem ser animadas. Finalmente, criamos o objeto de animação:

```
#cria animacao
anim = animation.FuncAnimation(fig,animate,init_func = init, frames=t.
    size,interval=0,blit=True,repeat = False)
```

Referencia

N. J. Giordano & H. Nakanishi, Computational Physics, 2°ed, 2007
3)M. A. Savi, "Dinâmica não-linear e caos", E-papers, Rio de Janeiro, 2006.