MECÂNICA II – PME 3200

MODELAGEM E SIMULAÇÃO NUMÉRICA

TUTORIAL DO *PYTHON*

Prof. Dr. Roberto Spinola Barbosa

1 Introdução

Um sistema mecânico tem seus movimentos descritos por equações diferenciais de segunda ordem a termos constantes. Um sistema massa/mola com um grau de liberdade x e forçamento externo F , tem sua aceleração obtida pelo teorema da resultante e descrita pela seguinte equação diferencial:

$$\ddot{x} = f(x, \dot{x}, t) \qquad \Rightarrow \qquad \ddot{x} = -\frac{c}{m}\dot{x} - \frac{k}{m}x + \frac{F}{m} \tag{1}$$

Em geral os sistemas mecânicos apresentam grandes deslocamentos resultando em expressões não lineares, como por exemplo, o movimento de um pendulo simples:

$$\ddot{\theta} = -\frac{g}{L}\operatorname{sen}\theta \tag{2}$$

A solução de equação diferencial não linear pode ser realizado por integração numérica (integral definida) do tipo:

$$A = \int_a^b f(x)dx \tag{3}$$

Assim para a integração da função $f(x)$ num intervalo de tempo dt é necessário o conhecimento das condições iniciais $x(0)$ e $\dot{x}(0)$ do sistema para $\boldsymbol{t} = \boldsymbol{t}(0)$:

$$\dot{x}(t+dt) = \int_{t=t(0)}^{t=t+dt} \ddot{x}\,dt + \dot{x}(0) \qquad \text{e} \qquad x(t+dt) = \int_{t=t(0)}^{t=t+dt} \dot{x}\,dt + x(0) \tag{4}$$

Existem diversos programas numéricos em linguagem de programação de alto nível, com algoritmos especializados para realizar a tarefa de integração numérica (*Scilab, Octave, Matlab, Mathematica, Mathcad, Python*, etc.).

A edição dos códigos para a resolução das equações diferenciais descritivas do modelo, pode ser realizada diretamente por linhas de comando, com chamadas apropriadas aos algoritmos de integração ou, em ambiente gráfico (*diagramas de blocos*) específicos e disponíveis nos programas como *Xcos (Scilab), Simulink (Matlab)*, respectivamente. Alguns destes programas são de uso gratuito e possuem códigos abertos (*Scilab, Octave, Python*).

2 Modelagem

Considere como exemplo o sistema pendular que se movimenta no plano e portanto, com 1 grau de liberdade, com dissipação viscosa na articulação, conforme mostrado na Figura 1.

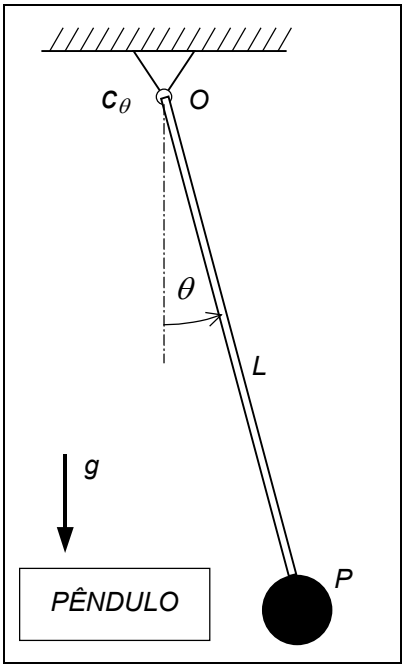


Figura 1 – Sistema pendular simples com 1 grau de liberdade

Utilizando o ***TQMA*** (Teorema da Quantidade de Movimento Angular) descrito por:

$$\dot{\vec{H}}_O = \vec{M}_O \tag{5}$$

e adotando o pólo *O* para um sistema plano e coordenada angular θ, obtêm-se a seguinte equação diferencial de segunda ordem:

$$J_{Oz}\ddot{\theta}\,\vec{k} = -mgL\,\mathrm{sen}\,\theta\,\vec{k} - c_\theta\dot{\theta}\,\vec{k} \qquad \text{ou} \qquad \ddot{\theta} = -\frac{g}{L}\mathrm{sen}\,\theta - \frac{c_\theta}{mL^2}\dot{\theta} \tag{6}$$

com os seguintes parâmetros: massa da partícula m , comprimento do pêndulo L e coeficiente de dissipação viscosa c_θ na articulação *O*, proporcional a velocidade angular $\dot{\theta}$.

3 Implementação Numérica

Para a implementação numérica da integração das equações diferenciais de segunda ordem são necessárias duas etapas de integração. Por conveniência pode-se expressar o sistema na forma de espaço de estados, reduzindo a ordem do sistemas de equações diferenciais.

Seja o vetor de estados $\{y\}$ descrito como:

$$\{y\} = \begin{Bmatrix} y(1) \\ y(2) \end{Bmatrix} = \begin{Bmatrix} \theta \\ \dot{\theta} \end{Bmatrix} \qquad \text{derivando} \quad \Rightarrow \qquad \{\dot{y}\} = \begin{Bmatrix} \dot{y}(1) \\ \dot{y}(2) \end{Bmatrix} = \begin{Bmatrix} \dot{\theta} \\ \ddot{\theta} \end{Bmatrix} \tag{7}$$

A equação diferencial de primeira ordem dupla do sistema resulta em:

$$\begin{aligned} &\dot{\theta} = \dot{\theta} \\ &\ddot{\theta} = -\frac{g}{L}\mathrm{sen}\,\theta - \frac{k_\theta}{mL^2}\dot{\theta} \end{aligned} \quad \Rightarrow \quad \begin{aligned} &\dot{y}(1) = y(2) \\ &\dot{y}(2) = -(g/L)\cdot\mathrm{sen}(y(1)) - (k_\theta/mL^2)\cdot y(2) \end{aligned} \tag{8}$$

A redução de ordem para representação em espaço de estados, permite realizar de uma só vez a integração do sistema de primeira ordem duplo.

$$\{y\}(t+dt) = \int_{t=t(0)}^{t=t+dt} \{\dot{y}\}dt + \{y\}(0) \tag{9}$$

Considerando conhecidas as condições inicias de posição angular $y(1) = \theta(0)$ e velocidade angular $y(2) = \dot{\theta}(0)$ como nulas, pode-se descrever o vetor de estados $y = f(\theta, \dot{\theta}, t)$ na linguagem *Python* como:

```
# condições iniciais:
theta0 = 0.0
thetadot0 = 0.0
# criação do vetor de estados considerando as condições inicial
y =[theta0,thetadot0]
```

Para realizar a integração da função *pendulo* utiliza-se a rotina *ode* da biblioteca numérica do *Python* (num.py) que tem como parâmetros de entrada o vetor de estados *y*, instantes de tempo *t* e os argumentos *par*.

```
# integrador
Y = ode(pendulo, y, t, args=(par,))
```

O parâmetro ***t*** de entrada da função ***ode*** deve ser um vetor com um número finito de instantes de tempo τ espaçados de d***t*** em um intervalo τ ∈ [*t*i; *t*f] para os quais os valores de ***y*** serão numericamente calculados.

```
# parâmetros de  tempo
ti=0.0                  # instante de tempo inicial
tf=10.0                 # instante de tempo final
h = 0.01                # intervalo de tempo
t = np.arange(ti,tf,h)  # gera o vetor de instantes de tempo
```

Finalmente a função pendulo que contem a descrição do sistema (equação diferencial) é descrita como:

```
#função
    def pendulo (y,t,par):
# separa os parâmetros do sistema
    g = par[0]
    l = par[1]
# cria o vetor de saída  ydot  vazio
    ydot=[0,0]
# equações dinâmicas
    ydot[0] = y[1]
    ydot[1] = -g/l*np.sin(y[0]) - c*y[1]/(m*l*l)
    return ydot  # retorna o vetor ydot
```

Os parâmetros do sistema devem ser definidos no corpo do programa principal.

```
#parâmetros do sistema
    m = 10.0
    g = 9.8
    L = 1.0
    par = [m, g, l]
```

Para desenhar gráficos com os resultados da simulação utilize a função "*graf.plot*" conforme apresentado na listagem do programa em anexo.

Como exemplo a simulação do sistema apenas com condições iniciais diferentes não nulas foi realizado e os resultados estão apresentados a seguir:

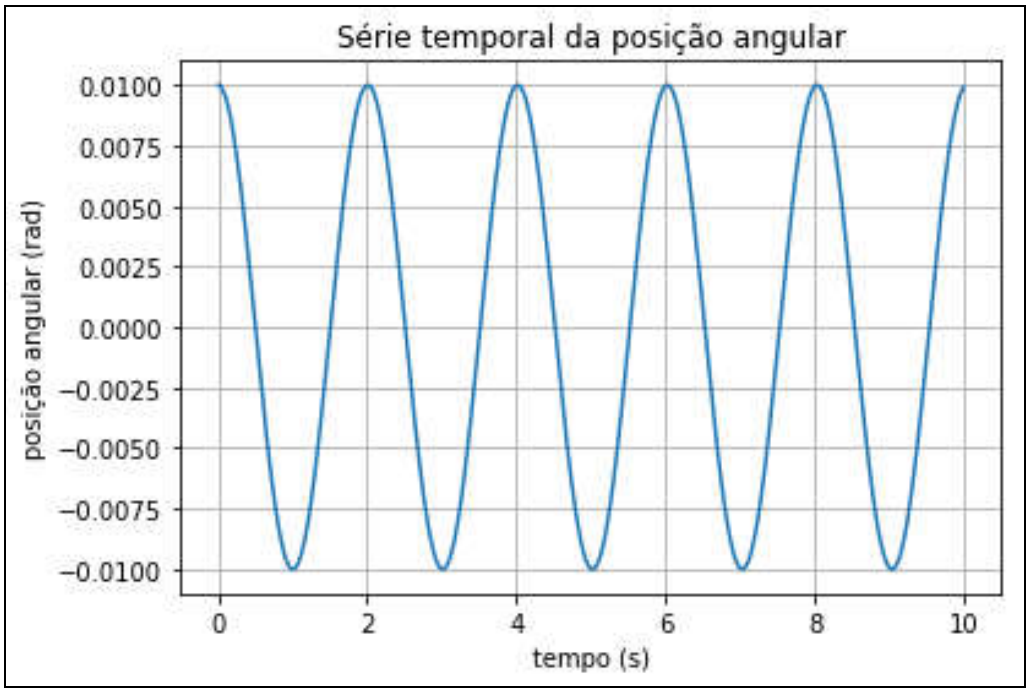


Figura 2 – História Temporal da Posição Angular ($\theta_0 = 0.01$)

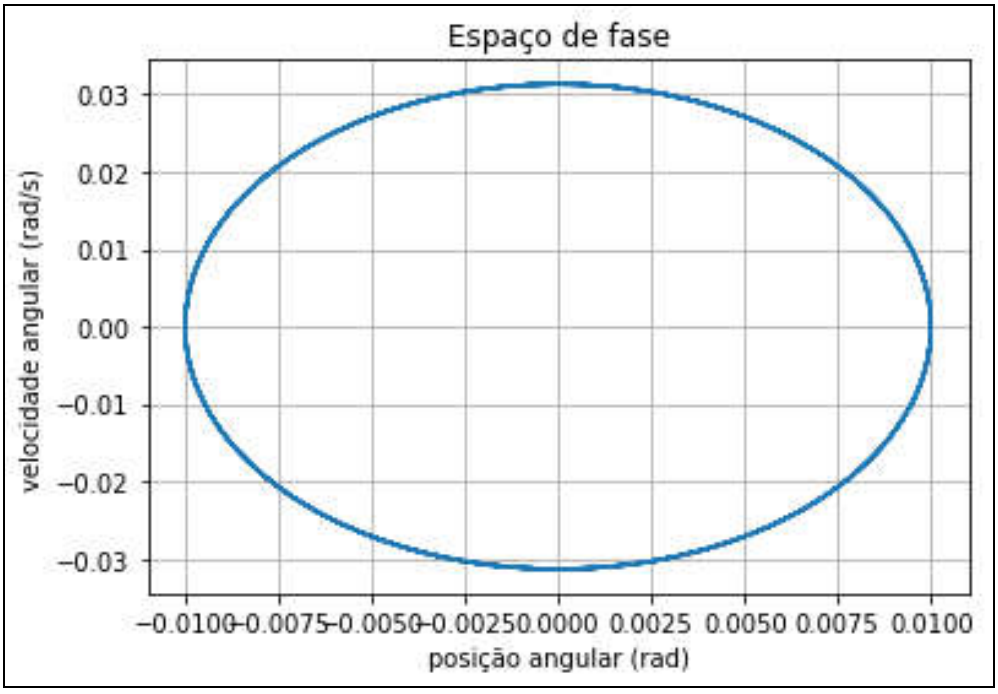


Figura 3 – Espaço de Fase ($\theta_0 = 0.01$)

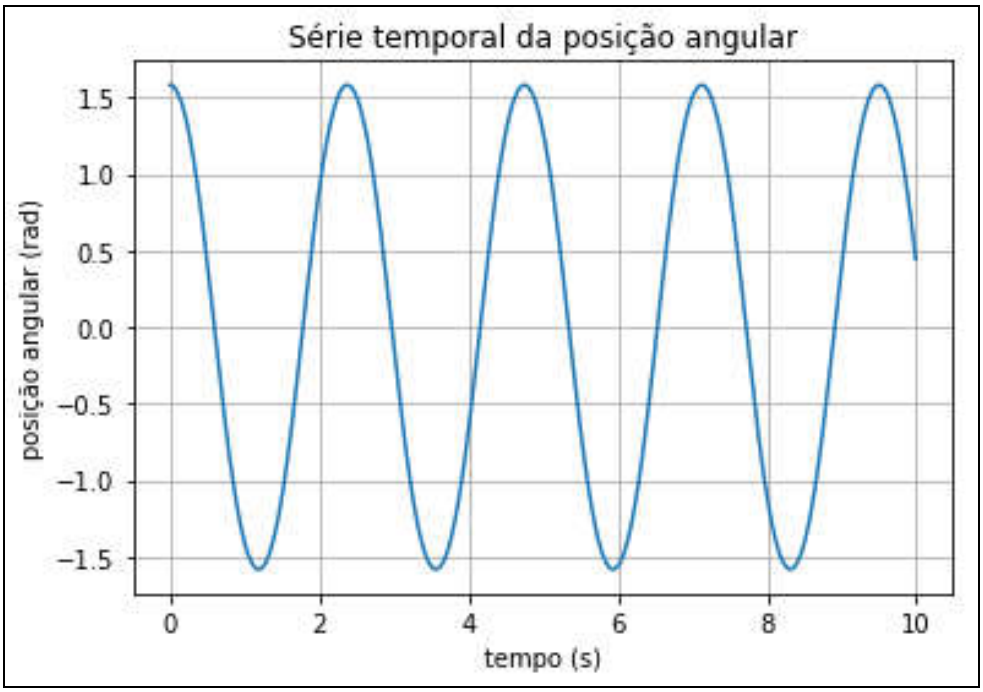


Figura 4 – História Temporal da Posição Angular ($\theta_0 = 0.01 + \pi/2$)

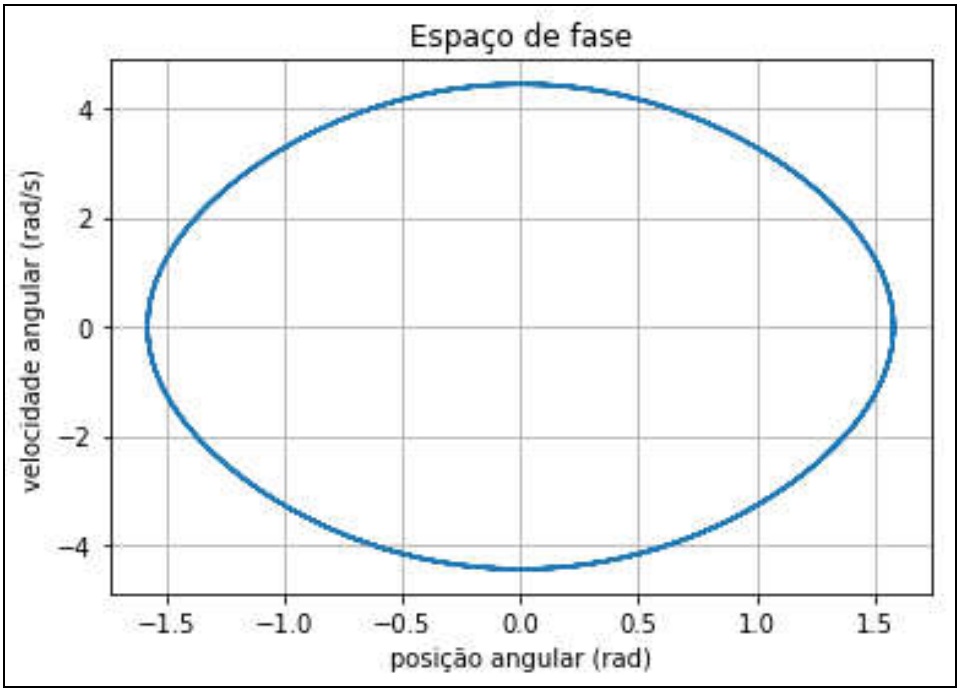


Figura 5 – Espaço de Fase ($\theta_0 = 0.01 + \pi/2$)

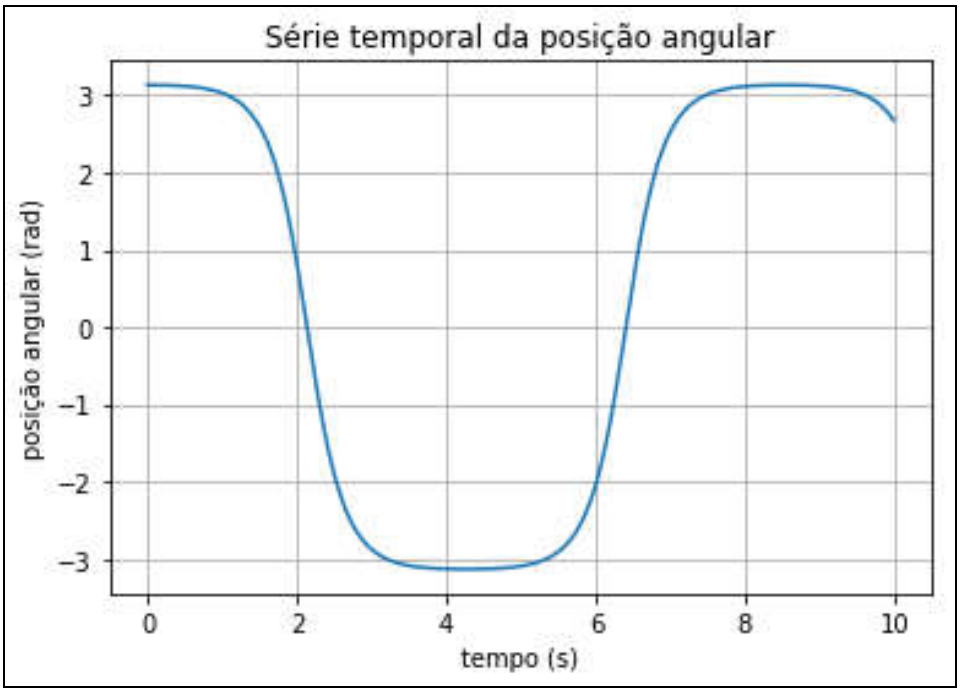


Figura 6 - História Temporal da Posição Angular ($\theta_0 = 0.01 + \pi$)

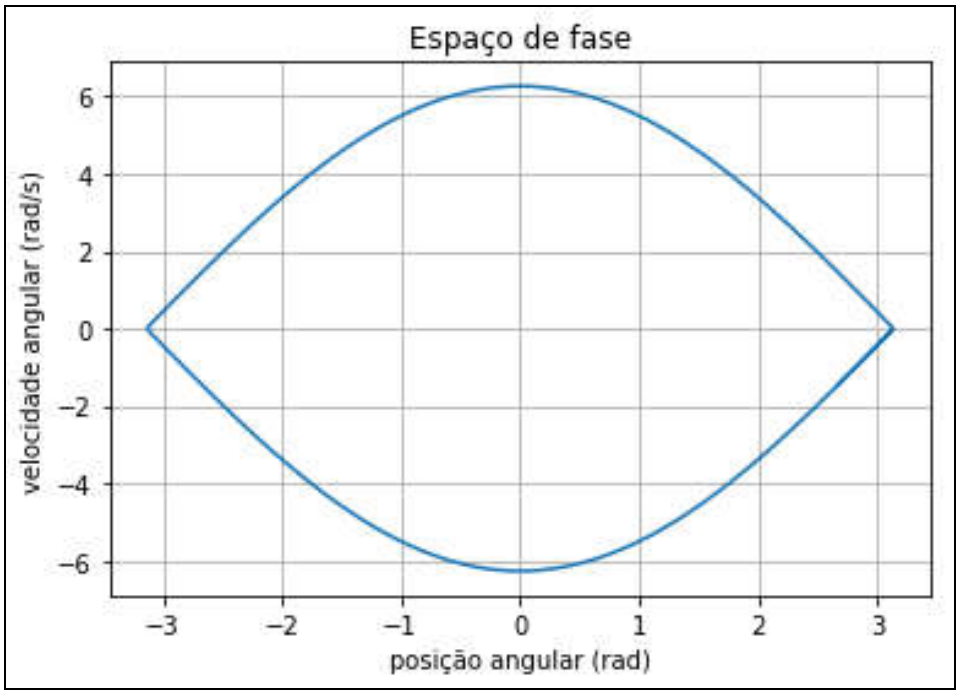


Figura 7 - Espaço de Fase ($\theta_0 = 0.01 + \pi$)

4 Agradecimentos

Agradecimentos aos alunos: Mateus Fujita Siveira e Gustavo Lopes Oliveira monitores de disciplina Mecânica II (PME-3200) que realizaram a implementação dos códigos em linguagem *Python*.

5 Referencias Bibliográficas

França, L. N. F. *Matsumura*, A. Z. (2011) Mecânica Geral. Editora Edgard Blücher, 3ª edição, ISBN: 9788521205784, p. 316.

Manual do *Python* – *Python* é um programa aberto (*open-source*) e livre para uso sendo distribuído pela *Python Software Foundation*. https://www.python.org/doc/

Manual do *Scilab* - *Scilab* é um programa aberto (*open-source*) e livre para uso sendo distribuído pela *INRIA* (até 2003) e atualmente pela *SCILAB Enterprises*. https://www.scilab.org/tutorials

Manual do *Octave* – *https://www.gnu.org/software/octave/*

Manual do *Matlab* - *https://www.mathworks.com/help/matlab/*

Manual do *Mathematica* - *http://www.wolfram.com/mathematica/*

ANEXO A – CÓDIGOS DE PROGRAMA DO PENDULO

```
# Comandos de importação das bibliotecas do phyton necessárias para este programa
from scipy.integrate import odeint as ode
import matplotlib.pyplot as graf
import numpy as np

# parâmetros do sistema
g = 9.8
l = 1.0
par = [g,l]

# condições iniciais:
theta0 = np.pi-0.01
thetadot0 = 0.0
y =[theta0,thetadot0] #vetor de estados inicial

# função pendulo
def pendulo(y,t,par):
    # separa os parâmetros do sistema
    g = par[0]
    l = par[1]
    # cria o vetor de saida ydot vazio
    ydot=[0,0]
    # equações dinâmicas
    ydot[0] = y[1]
    ydot[1] = -g/l*np.sin(y[0])
    return ydot  # retorna como saida o vetor ydot

#parâmetros do tempo
ti=0.
tf=10.
h = 0.01
t = np.arange(ti,tf,h) # gera o vetor de instantes de tempo

#integrador da função pendulo
Y = ode(pendulo,y,t,args=(par,))

# separa posição e velocidade
Xn = Y[:,0]
Vn = Y[:,1]

# desenha os gráficos
graf.plot(t,Xn)
graf.title("Série temporal da posição angular")
graf.xlabel("tempo (s)")
graf.ylabel("posição angular (rad)")
graf.grid()
graf.show()

graf.plot(t,Vn)
graf.title("Série temporal da velocidade angular")
graf.xlabel("tempo (s)")
graf.ylabel("velocidade angular (rad/s)")
graf.grid()
```

```
graf.show()

graf.plot(Xn,Vn)
graf.title("Espaço de fase")
graf.xlabel("posição angular (rad)")
graf.ylabel("velocidade angular (rad/s)")
graf.grid()
graf.show()
```

ANEXO C – SINTAXE DO PROGRMA PYTHON

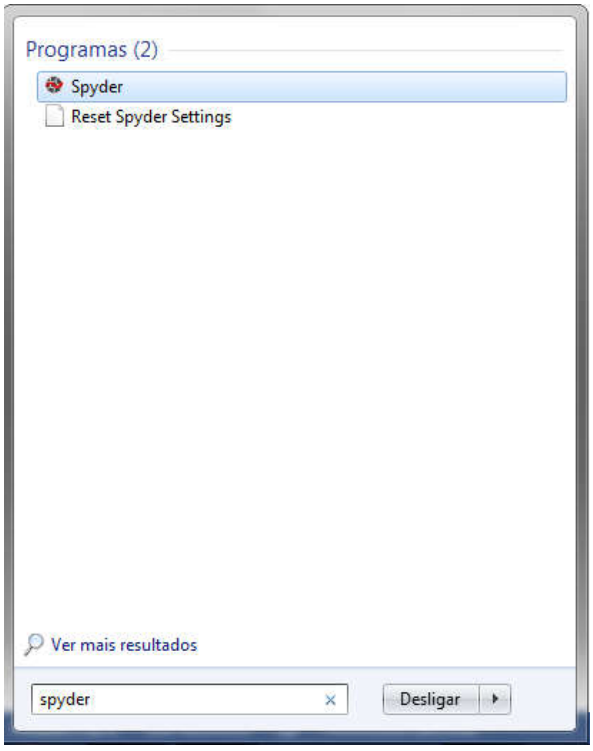
Alguns aspectos da sintaxe da linguagem *Python* são relembradas:

C1. Anaconda/Spyder

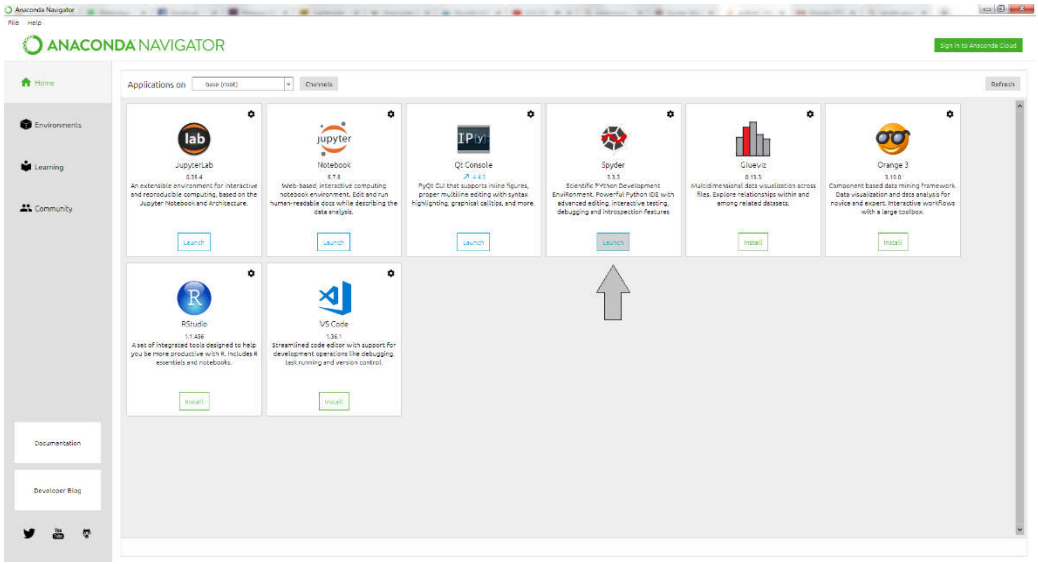
Existem vários ambientes disponíveis para se programar em *Python*. Recomenda-se o uso do Spyder, um ambiente que vem da *Anaconda Distribuition*. As bibliotecas necessárias para realizar as atividades propostas já estão disponíveis.

Link do Spyder: https://www.spyder-ide.org/
Link de Download: https://www.anaconda.com/distribution/

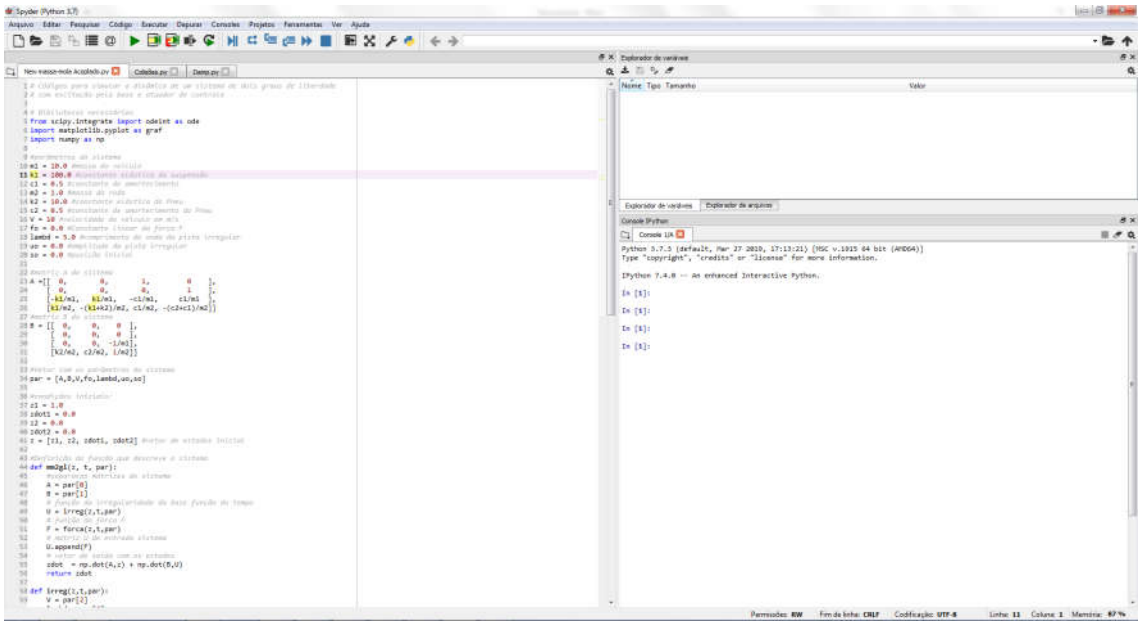
Uma vez baixado o pacote, para abrir o Spyder há 2 opções. Uma delas é acessar diretamente pelo menu iniciar:



Caso não encontre, também há a opção de acessar diretamente pelo ***Anaconda Navigator***.



Após abrir o Spyder, o seguinte ambiente deverá aparecer:



A esquerda, um editor de texto onde o programa deverá ser escrito. O console inferior à direita apresenta os resultados calculados pelo programa. Cada programa escrito deverá ser salvo antes de executado. Por questão de organização, recomenda-se a criação de uma pasta para esses arquivos.

C2. Importação de Pacotes

Além de sua programação padrão, vista em grande parte na disciplina **MAC2166**, o *Python* também conta com **pacotes** que aumentam a sua variedade de comandos e códigos possíveis.
No *Anaconda*, grande parte desses pacotes já vêm instalados, e basta apenas importá-los em seu código. Como exemplo, considere o pacote **numpy**, que será usado nesse tutorial. Para importá-lo, basta usar o seguinte código:

```
import numpy
```

Usando este comando antes do código do programa, todos os comandos de numpy são importados e estarão prontos para serem usados.
Por simplicidade pode-se renomear as bibliotecas. Por exemplo vamos chamar o numpy de np da seguinte forma:

```
import numpy as np
```

Assim, por exemplo, a função seno do numpy será escrita como:

```
np.sin(2)
```

E retorna:

```
0.9092974268256817
```

Em resumo: sempre que você for usar alguma função ou comando de algum pacote, é necessário escrever o nome do pacote como prefixo e um ponto "." antes da função. Como renomeamos o pacote "numpy" como "np", todo comando desse pacote deve ser precedido pela prefixo "np.".

C3. Pacotes Adicionais

Para o cálculo de integração numérica, usaremos 3 pacotes adicionais: numpy, matplotlib.pyplot e scipy. A importação e o uso serão descritos nos próximos tópicos.

3.1. Numpy

O numpy é um modulo do *Python* que, essencialmente, trabalha com vetores e matrizes, além de contar com ferramentas sofisticadas de funções e álgebra linear. Vamos apresentar apenas alguns comandos úteis e como usar vetores e matrizes. Para importar o *numpy*, usar o seguinte código:

```
import numpy as np
```

E cada função requerida, deverá ser precedida pela prefixo "np.".

3.1.1. Criação de vetores e matrizes

Os vetores e matrizes em numpy são idênticos às listas em *Python*. Vamos, por exemplo, criar a matriz:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 5 & 4 & 3 \\ 5 & 6 & 7 \end{bmatrix}$$

Em código do *Python*, isso seria escrito como:

```
A = [[1,2,3],[5,4,3],[5,6,7]]
```

Ou então, de maneira mais elegante (inserindo quebras de linhas e espaços):

```
A = [[1,2,3],
     [5,4,3],
     [5,6,7]]
```

Em resumo: cada linha deve ser escrita entre colchetes e separadas por vírgulas entre elementos e entre linhas. No fim, deve-se fechar a matriz com colchetes. Portanto para vetores basta criar uma lista simples com os números/elementos separados por virgula:

```
v = [1,2,3]
```

Em que v é o vetor $v = [1 \quad 2 \quad 3]^t$.

3.1.2. Manipulação de vetores e matrizes

Algumas funções úteis para manipular vetores e matrizes:

Adição de matrizes:

```
np.add(A,B)
```

Recebe duas matrizes/vetores/escalares de mesma dimensão e soma seus elementos. Exemplo:

```
#define matrizes
A = [[1,2,3],[4,5,6],[2,3,4]]
B = [[3,2,1],[6,5,4],[4,3,2]]
#soma
np.add(A,B)
```

E a resposta é:

```
array([[ 4,  4,  4],
       [10, 10, 10],
       [ 6,  6,  6]])
```

Em numpy, há uma categoria diferente das listas para se escrever matrizes e vetores, que é o array. As funções de numpy podem funcionar com ambas, mas sempre retorna uma array.

Produto de Matrizes:

```
np.dot(A,B)
```

- Recebe duas matrizes $A_{n\times m}$ e $B_{m\times p}$ e faz a multiplicação.
- Recebe dois vetores de mesma dimensão e calcula seu produto escalar.

Exemplo de multiplicação de matrizes:

```
#define matrizes
A = [[1,2,3],[4,5,6],[2,3,4]]
B = [[3,2,1],[6,5,4],[4,3,2]]
#multiplicação
np.dot(A,B)
```

E a resposta é:

```
array([[27, 21, 15],
       [66, 51, 36],
       [40, 31, 22]])
```

Exemplo de multiplicação de vetores:

```
#define vetores
A = [1,2,3]
B = [3,2,1]
#produto escalar
np.dot(A,B)
```

E a resposta é:

```
10
```

Produto Termo a Termo

A função `np.multiply` ou "*" faz uma simples multiplicação termo a termo de duas matrizes ou vetores de tamanhos compatíveis.

```
#define matrizes
A = [[1,2,3],[4,5,6],[2,3,4]]
B = [[3,2,1],[6,5,4],[4,3,2]]
#multiplicação
np.multiply(A,B)
```

(ou `#multiplicação  A*B )`

E a resposta é:

```
array([[3, 4, 3],
       [24, 25, 24],
       [8, 9, 8]])
```

Observação: Caso queira ver mais funções de numpy para vetores e matrizes, consulte o site onde está documentado as diversas funções do numpy:
https://docs.scipy.org/doc/numpy-1.15.1/index.html.

3.1.3. Funções

Aqui serão apresentados algumas funções que serão úteis em numpy. A maioria pode ser feita com escalares e com elementos de vetores e matrizes.

```
np.pi
```

Retorna π com 16 algarismos significativos.

```
np.e
```

Retorna e com 16 algarismos significativos.

```
np.sin(a)
np.cos(a)
np.tan(a)
```

Funções trigonométricas. Recebem um escalar, vetor ou matriz e retorna o valor de seno, cosseno e tangente respectivamente do escalar ou dos elementos do vetor/matriz. Exemplo:

```
a = [np.pi, 0, 2]
np.sin(a)
```

Que retorna:

```
array([1.22464680e-16, 0.00000000e+00, 9.09297427e-01])
```

A notação e-16, por exemplo, significa que é o número anterior vezes 10^{-16}.
Esperava-se que o $\sin(\pi)$ fosse zero, mas obteve-se um valor da ordem de 10^{-16}. Isso é normal, visto que o valor de π usado só tem 15 casas decimais e que todas as operações são feitas numericamente, usando Série de Taylor. Esse tipo de erro, no entanto, não afeta significantemente os resultados esperados.

```
np.exp(a)
```

Função exponencial. Recebe um escalar, vetores ou matrizes e retorna a exponencial do escalar ou dos elementos do vetores e matrizes.

Exemplo:

```
a = [0, 1, np.log(1)]
np.exp(a)
```

Que retorna:

```
array([1.        , 2.71828183, 1.        ])
```

Nesse mesmo exemplo, usou-se a função `np.log(a)` que calcula o $\ln(a)$.

```
np.sqrt(a)
```

Função raiz quadrada. Recebe um escalar, vetores ou matrizes e retorna a raiz quadrada do escalar ou dos elementos do vetores e matrizes.

```
np.power(a,n)
```

Função potência. Recebe um escalar, vetor ou matriz a e um outro escalar n. Retorna n-esima potência do escalar ou dos elementos do vetores e matrizes.

```
np.arange(ti,tf,dt)
```

Essa função cria um vetor com números dentro do intervalo $[t_i, t_f[$ espaçados de dt, começando por t_i. Exemplo:

```
np.arange(1, 3, 0.1)
```

Com resposta:

```
array([1. , 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8,
1.9, 2. , 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9])
```

3.2. Matplotlib.Pyplot

O pacote Matplotlib é responsável por criar gráficos. Para importar essa biblioteca, basta escrever o comando:

```
import matplotlib.pyplot as graf
```

Ou seja, todos os comandos desse pacote devem ser precedidos do prefixo “graf.”. A documentação de comandos e exemplos do pacote podem ser vistos em:
https://matplotlib.org/.

3.2.1. Criando um Gráfico

Para se criar um gráfico é necessário duas atividades básicas:

- Dos pares ordenados de cada ponto x e y
- Algum comando para visualizar o gráfico

Inicialmente duas listas/vetores são selecionados: uma contendo as coordenadas x de cada ponto do gráfico e outra contendo as coordenadas y desses mesmos pontos do gráfico. Vamos, como um exemplo, criar um gráfico contendo os seguintes pontos:

$$A = (0,1), B = (1,2), C = (2,3), D = (3,4)$$

Para organizar o vetor x e y, devemos fazer:

```
X = [0, 1, 2, 3]
Y = [1, 2, 3, 4]
```

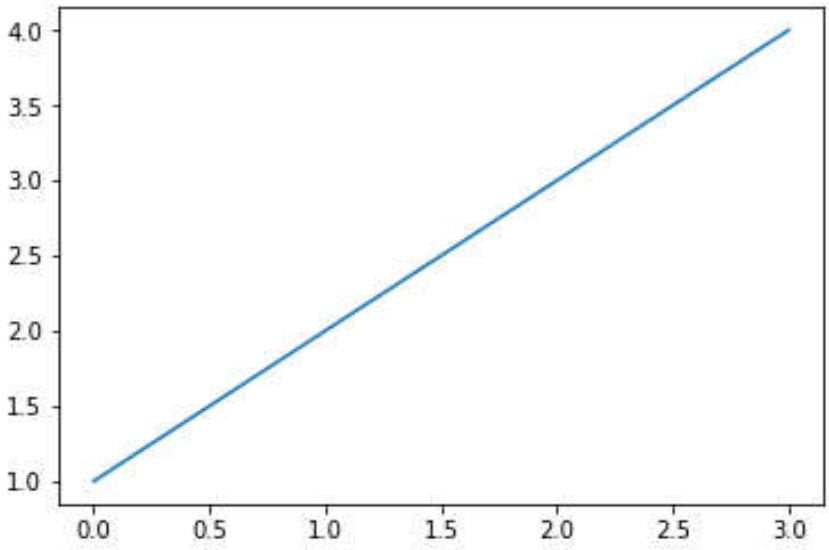
E para criar o gráfico, basta usar a seguinte função:

```
graf.plot(X,Y)
```

Para ver o gráfico diretamente no console do Spyder, basta, usar o comando:

```
graf.show()
```

O seguinte gráfico deveria ser visualizado:



3.2.2. Detalhes

Alguns detalhes dos gráficos são apresentados abaixo:

- Título no gráfico

- Legendas nos eixos
- Linhas de grades

Um código modelo pode ser visto abaixo, em que cada um desses elementos foi comentado (usando # antes do comentário) ao lado:

```
graf.plot(X,Y) #cria o gráfico
graf.title('Título do Gráfico') #Título do gráfico
graf.xlabel('Legenda eixo x') #legenda eixo x
graf.ylabel('Legenda eixo y') #Legenda eixo y
graf.grid() #grades
graf.show() #mostra o gráfico no console
```

O que está em verde é uma ***string***. É importante que o texto que será mostrado no gráfico esteja entre aspas “ ” ou apóstrofos ‘ ’, o que define uma ***string***. Usando os mesmos vetores x e y do tópico anterior e o código acima, o gráfico fica da seguinte forma:

3.2.3. Múltiplos gráficos

Pode imprimir mais de um gráfico nos mesmos eixos. Pode-se adicionar legenda para identificá-los. Observe o seguinte exemplo modelo:

```
X1 = [0, 1, 2, 3]
Y1 = [1, 2, 3, 4]
X2 = [1, 2, 3, 4]
Y2 = [0, 1, 2, 3]
graf.plot(X1,Y1, label = 'gráfico 1') #cria o gráfico 1
(com adição de legenda)
graf.plot(X2,Y2, label = 'gráfico 2') #cria o gráfico 2
(com adição de legenda)
graf.title('Título do Gráfico') #Título do gráfico
graf.xlabel('Legenda eixo x') #legenda eixo x
graf.ylabel('Legenda eixo y') #Legenda eixo y
graf.grid() #grades
```

```
graf.legend() #insere legenda no gráfico
graf.show() #mostra o gráfico no console
```

Aqui, houve a inserção de do gráfico 2 e das legendas no mesmo comando de criação do gráfico. Para tal, basta escrever `label = 'legenda desejada'`.
Finalmente foi acrescentado também o comando `graf.legend()`. Esse comando simplesmente acrescenta uma legenda no gráfico. Ao final da execução do código, o gráfico resultante terá o seguinte aspecto:

3.3. Odeint (Scipy.integrate)

Será utilizada apenas uma função **odeint** da biblioteca scipy.integrate. Por isso, é desnecessário importar a biblioteca inteira. Portanto será utilizado o seguinte comando de importação:

```
from scipy.integrate import odeint as ode
```

Portanto, importou-se apenas a função odeint renomeada por **ode**. Para chamar a função basta escrever `ode()`. A utilização da função ODE foi apresentada no texto.

4. A função ODE

O papel da função ode, é realizar a integração numérica. A função ode recebe os seguintes parâmetros:

```
Y = ode(F,x0, t, args = (par,))
```

- `F` – Função do vetor de estados. Antes de chamar a ODE, é importante criar uma função que recebe o vetor de estados x, o instante de tempo t e os parâmetros relevantes, como constantes físicas, e retorna o vetor de estados seguinte.

- `X0` – Vetor de estados inicial, em $t = 0$. Contém posição e velocidade iniciais

- `T` – Vetor com os instantes de tempo a integrar, entre $[t_i, t_f[$ com passo δt. Pode ser criado usando a função `np.arange(ti, tf, dt)`.

- `args = (par,)` - argumentos extras que devem entrar na função F. Nesse caso, inclui-se um vetor `par` contendo as constantes do problema (elástica, gravidade, amortecimento, etc).