9-Construindo Simulador de MRU

April 20, 2021

1 Informes iniciais

1. A interface do VPython, juntamente com as suas figuras, aparece na mesma célula de importação do vpython pela primeira vez após o kernel ser reiniciado. Então, após executar a célula para se gerar as figuras, **o usuário irá vizualizá-las na célula de importação da biblioteca**;
2. Caso o usuário queira "limpar" as figuras na interface, é preciso reiniciar o kernel e as saídas, indo no menu em `Kernel -> Restart & Clear Output`, se estiver no Jupyter Notebook;
3. É preciso lembrar de **sempre** executar a célula para importar a biblioteca VPython após reiniciar o kernel, uma vez que ela é a responsável por proporcionar a criação das figuras e afins.

2 Construindo Simulador de MRU

Para colocarmos em prática o que vimos até agora, iremos construir um simulador do MRU a parir dos códigos desenvolvidos no tutoriais anteriores, adicionando novos elementos, assim como associando-o ao TKinter.

Assim, desenvolveremos o simulador a partir do código ao final do tutorial 7-Construção de Gráficos em Tempo Real:

[ ]:
```
from vpython import * #sempre importar as bibliotecas ao início

graf_pos = graph(xtitle="tempo (s)", ytitle="Posição (m)")
curva_pos = gcurve(color=color.red, label = "componente X")

x_inicial = float(input('Insira a Posição Inicial(m): '))
v = float(input('Insira a Velocidade da Partícula(m/s): '))
t_inicial = float(input('Insira o Tempo Inicial(s): '))
t_final = float(input('Insira o Tempo Final(s): '))

if (v<0):
    p = 10
else:
    p=-10

#Perceba que vamos inserir os parâmetros da trajetória prórpia esfera
```

```
esfera = sphere(pos = vec(p, 0, 0), radius = 0.2, color = color.red, opacity=1,␣
 ↪make_trail=True, interval=10, trail_type="points", trail_color =color.yellow)
texto_x_inicial = label(pos=vec(p, 0, 0), text= str(x_inicial) + " m",␣
 ↪xoffset=0, yoffset=-100, space=30, height=16, border=4)
delt_t = t_final - t_inicial #tempo de execução da simulação (variação do tempo)
t = 0

e = (abs(v*delt_t))/20 #fator de controle de escala

while (t < delt_t):
    rate (100)
    t+=0.01
    esfera.pos = vec(p + (v*t)/e, 0, 0)#Perceba a inserção do fator de escala␣
 ↪apenas na variação da posição
    curva_pos.plot(pos=(t_inicial + t, x_inicial + v*t)) #plotagem do gráfico
```

Reinicie o kernel e limpe as saídas (ver Informes Iniciais) e comece a execução dos códigos a partir da célula abaixo para uma melhor facilidade na vizualização

3 Adicionando Outros Elementos

Para além do que já foi construído, adicinaremos aqui o ponto médio ($\frac{x_0+v\Delta t}{2}$) e o ponto final da trajetória ($x_0 + v\Delta t$) e a posição atual da partícula ao longo do tempo.

[ ]:
```
from vpython import * #sempre importar as bibliotecas ao início

graf_pos = graph(xtitle="tempo (s)", ytitle="Posição (m)")
curva_pos = gcurve(color=color.red, label = "componente X")

x_inicial = float(input('Insira a Posição Inicial(m): '))
v = float(input('Insira a Velocidade da Partícula(m/s): '))
t_inicial = float(input('Insira o Tempo Inicial(s): '))
t_final = float(input('Insira o Tempo Final(s): '))

if (v<0):
    p = 10
else:
    p=-10

#Perceba que vamos inserir os parâmetros da trajetória prórpia esfera
delt_t = t_final - t_inicial #tempo de execução da simulação (variação do tempo)
esfera = sphere(pos = vec(p, 0, 0), radius = 0.4, color = color.red, opacity=1,␣
 ↪make_trail=True, interval=10, trail_type="points", trail_color =color.yellow)
texto_x_inicial = label(pos=vec(p, 0, 0), text= str(x_inicial) + " m",␣
 ↪xoffset=0, yoffset=-100, space=30, height=16, border=4) #indica o ponto␣
 ↪inicial
```

```
texto_x_medio = label(pos=vec(0, 0, 0), text= str((v*delt_t+x_inicial)/2)+ "␣
↪m", xoffset=0, yoffset=-100, space=30, height=16, border=4, font='sans')␣
↪#indica o ponto médio
texto_x_final = label(pos=vec(p*-1, 0, 0), text= str(v*delt_t+x_inicial)+ " m",␣
↪xoffset=0, yoffset=-100, space=30, height=16, border=4, font='sans') #indica␣
↪o ponto final
texto_posicao = label(pos=vec(0, 4, 0), text= "Posição: " + str(x_inicial)+ "␣
↪m", space=30, height=16, border=4, font='sans')

t = 0
e = abs(v*delt_t)/20 #fator de controle de escala

while (t < delt_t):
    rate (100)
    t+=0.01
    esfera.pos = vec(p + (v*t)/e, 0, 0)#Perceba a inserção do fator de escala␣
↪apenas na variação da posição
    curva_pos.plot(pos=(t_inicial + t, x_inicial + v*t)) #plotagem do gráfico
    texto_posicao.text = 'Posição: ' + str(round (x_inicial + v*t, 2))+ ' m'
```

Reinicie o kernel e limpe as saídas (ver Informes Iniciais) e comece a execução dos códigos a partir da célula abaixo para uma melhor facilidade na vizualização

4 Associando ao Tkinter

Como dito anteriormente, o Tkinter não é necessariamente uma biblioteca fundamental para o desenvolvimento dos simuladores, mas se destaca como um framework capaz de proporcionar o desenvolvimento de interfaces gráficas de usuário (GUI – Graphic User Interface).

4.1 Criando a janela

Nesse caso, utilizamos a mesma lógica de construção apresentada no tutorial 8-Noções Básicas de Tkinter, criando uma janela que solicite os mesmo dados, mas, agora, em uma GUI. Aqui, foram adicionados alguns novos elementos para dar um melhor visual à GUI:

1. `pady`: Espaçamentos da margem em cima e embaixo;
2. `padx`: Espaçamentos da margem nas laterais;
3. `Frame`: Convencionalmente, antes de inserir elementos na janela, aconcelha-se pôr um `Frame` para que seja possível manipular o plano de fundo com maior facilidade, além de ser possível dividir a janela em diferentes `Frame`'s. Por esse motivo, os elementos são dispostos em cima do Frame;
4. `sticky`: Utilizado para orientar o texto dentro da própria célula - à esquerda (oeste - `W`), à direita (leste - `E`), embaixo (sul - `S`) e em cima (norte `N`). É também possível combinar essas direções para formar outra (ex: nordeste - `N+E`).

[ ]:
```
from tkinter import *
```

```
#O código abaixo se refere exclusivamente à criação de janelas do TKinter
janela = Tk()
janela.geometry('530x500+400+150')
janela.title('SIMULAÇÕES')
janela['pady']=20
janela['padx']=30

container = Frame(janela, highlightbackground='black', highlightthickness=2)
container['pady']=20
container['padx']=35
container.grid(row=0, column=0)

lab_title = Label(container, text='Movimento Retilíneo Uniforme', font='times 20 bold', height=3)
lab_i_pos = Label(container, text='Posição Inicial (m):', font='times 14 ', height=3)
lab_i_vel = Label(container, text='Velocidade Inicial (m/s):', font='times 14', height=3)
lab_ti = Label(container, text='t_inicial(s):', font='times 14 ', height=3)
lab_tf = Label(container, text='t_final (s):', font='times 14 ', height=3)

lab_title.grid(row=0, column=0, columnspan=4)
lab_i_pos.grid(row=1, column=0, columnspan=2, sticky=W)
lab_i_vel.grid(row=2, column=0, columnspan=2, sticky=W)
lab_ti.grid(row=3, column=0, sticky=W)
lab_tf.grid(row=3, column=2, sticky=W)

ed_i_pos = Entry(container, width=26,font='times 12')
ed_i_vel = Entry(container, width=26,font='times 12')
ed_ti = Entry(container, width=7,font='times 12')
ed_tf = Entry(container, width=7,font='times 12')

ed_i_pos.grid(row=1, column=2, columnspan=2, sticky=W)
ed_i_vel.grid(row=2, column=2,  columnspan=2, sticky=W)
ed_ti.grid(row=3, column=1, sticky=W)
ed_tf.grid(row=3, column=3, sticky=W)


lb1 = Label(container, text='', height=2, font='times 12 italic')
lb1.grid(row=4, column=1, columnspan=2)

bt = Button(container, text = 'Gerar Simulação', font='times 14 bold', pady=12)
bt.grid(row=5, column=1, columnspan=2, sticky=S)

janela.mainloop()
```

Reinicie o kernel e limpe as saídas (ver Informes Iniciais) e comece a execução dos códigos a partir da célula abaixo para uma melhor facilidade na vizualização

4.2 Capturando os Dados Inseridos

Nesse caso, utilizamos como `camanda` função `bt_clic` e capturamos os dados com a função `get()`. Como dito anteriormente, a função `get()` captura os dados como caractér, sendo necessário transformá-los em números reais (`float()`).No entanto, caso o usuário digite um caractér, ocorrerá erro na execução da simulação.

É interessante que se faça um *tratamento de erros* por meio da estrutura `try` e `except`. De modo geral, enquanto o programa estiver dentro da estrutura `try`, será tentado executá-lo sem erros; caso dê algum erro, o programa se direcionará para a estrutura `except` sem que seja finalizada a sua execução. Nesse caso, o programa tentará converter os valores dentro do `try` para `float` e, caso ocorra algum erro, irá para a estrutura `except`, onde será exibida a mensagem de dados inválidos, permitindo ao usuário inseri-los novamente.

[ ]:
```
from tkinter import *

#evento associado ao botão Gerar Simulação
def bt_clic():
        try:
            x_pos=float(ed_i_pos.get())
            v=float(ed_i_vel.get())
            t_inicial=float(ed_ti.get())
            t_final=float(ed_tf.get())
        except:
            lb1['text']='Valores Informados Inválidos!!!'

#O código abaixo se refere exclusivamente à criação de janelas do TKinter
janela = Tk()
janela.geometry('530x500+400+150')
janela.title('SIMULAÇÕES')
janela['pady']=20
janela['padx']=30

container = Frame(janela, highlightbackground='black', highlightthickness=2)
container['pady']=20
container['padx']=35
container.grid(row=0, column=0)

lab_title = Label(container, text='Movimento Retilíneo Uniforme', font='times 20 bold', height=3)
lab_i_pos = Label(container, text='Posição Inicial (m):', font='times 14 ', height=3)
lab_i_vel = Label(container, text='Velocidade Inicial (m/s):', font='times 14', height=3)
```

```
lab_ti = Label(container, text='t_inicial(s):', font='times 14 ', height=3)
lab_tf = Label(container, text='t_final (s):', font='times 14 ', height=3)

lab_title.grid(row=0, column=0, columnspan=4)
lab_i_pos.grid(row=1, column=0, columnspan=2, sticky=W)
lab_i_vel.grid(row=2, column=0, columnspan=2, sticky=W)
lab_ti.grid(row=3, column=0, sticky=W)
lab_tf.grid(row=3, column=2, sticky=W)

ed_i_pos = Entry(container, width=26,font='times 12')
ed_i_vel = Entry(container, width=26,font='times 12')
ed_ti = Entry(container, width=7,font='times 12')
ed_tf = Entry(container, width=7,font='times 12')

ed_i_pos.grid(row=1, column=2, columnspan=2, sticky=W)
ed_i_vel.grid(row=2, column=2,  columnspan=2, sticky=W)
ed_ti.grid(row=3, column=1, sticky=W)
ed_tf.grid(row=3, column=3, sticky=W)


lb1 = Label(container, text='', height=2, font='times 12 italic')
lb1.grid(row=4, column=1, columnspan=2)

bt = Button(container, text = 'Gerar Simulação', font='times 14 bold', pady=12,␣
 ↪command=bt_clic)
bt.grid(row=5, column=1, columnspan=2, sticky=S)

janela.mainloop()
```

Reinicie o kernel e limpe as saídas (ver Informes Iniciais) e comece a execução dos códigos a partir da célula abaixo para uma melhor facilidade na vizualização

4.3 Criando a Simulação

Nesse caso, basta inserir a simulação dentro da estrutura `try`, e ele será gerada. Caso queira, também é possível "destruir" a janela durante a execução da simulação com o comando `janela.destroy()`.

[ ]:
```
from vpython import *
from tkinter import *

graf_pos = graph(xtitle='tempo (s)', ytitle='Posição (m)')
curva_pos = gcurve(color=color.red, label = 'Coordenada X')

#evento associado ao botão Gerar Simulação
def bt_clic():
        try:
            x_inicial=float(ed_i_pos.get())
```

```
            v=float(ed_i_vel.get())
            t_inicial=float(ed_ti.get())
            t_final=float(ed_tf.get())

            #Simulação
            if (v<0):
                p = 10
            else:
                p=-10

            #Perceba que vamos inserir os parâmetros da trajetória prórpia␣
↪esfera
            delt_t = t_final - t_inicial #tempo de execução da simulação␣
↪(variação do tempo)
            esfera = sphere(pos = vec(p, 0, 0), radius = 0.4, color = color.
↪red, opacity=1, make_trail=True, interval=10, trail_type="points",␣
↪trail_color =color.yellow)
            texto_x_inicial = label(pos=vec(p, 0, 0), text= str(x_inicial) + "␣
↪m", xoffset=0, yoffset=-100, space=30, height=16, border=4) #indica o ponto␣
↪inicial
            texto_x_medio = label(pos=vec(0, 0, 0), text=␣
↪str((v*delt_t+x_inicial)/2)+ " m", xoffset=0, yoffset=-100, space=30,␣
↪height=16, border=4, font='sans') #indica o ponto médio
            texto_x_final = label(pos=vec(p*-1, 0, 0), text=␣
↪str(v*delt_t+x_inicial)+ " m", xoffset=0, yoffset=-100, space=30, height=16,␣
↪border=4, font='sans') #indica o ponto final
            texto_posicao = label(pos=vec(0, 4, 0), text= "Posição: " +␣
↪str(x_inicial)+ " m", space=30, height=16, border=4, font='sans')

            t = 0
            e = abs(v*delt_t)/20 #fator de controle de escala
            janela.destroy()

            while (t < delt_t):
                rate (100)
                t+=0.01
                esfera.pos = vec(p + (v*t)/e, 0, 0)#Perceba a inserção do fator␣
↪de escala apenas na variação da posição
                curva_pos.plot(pos=(t_inicial + t, x_inicial + v*t)) #plotagem␣
↪do gráfico
                texto_posicao.text = 'Posição: ' + str(round (x_inicial + v*t,␣
↪2))+ ' m'
        except:
            lb1['text']='Valores Informados Inválidos!!!'
```

```
#O código abaixo se refere exclusivamente à criação de janelas do TKinter
janela = Tk()
janela.geometry('530x500+400+150')
janela.title('SIMULAÇÕES')
janela['pady']=20
janela['padx']=30

container = Frame(janela, highlightbackground='black', highlightthickness=2)
container['pady']=20
container['padx']=35
container.grid(row=0, column=0)

lab_title = Label(container, text='Movimento Retilíneo Uniforme', font='times 20 bold', height=3)
lab_i_pos = Label(container, text='Posição Inicial (m):', font='times 14 ', height=3)
lab_i_vel = Label(container, text='Velocidade Inicial (m/s):', font='times 14', height=3)
lab_ti = Label(container, text='t_inicial(s):', font='times 14 ', height=3)
lab_tf = Label(container, text='t_final (s):', font='times 14 ', height=3)

lab_title.grid(row=0, column=0, columnspan=4)
lab_i_pos.grid(row=1, column=0, columnspan=2, sticky=W)
lab_i_vel.grid(row=2, column=0, columnspan=2, sticky=W)
lab_ti.grid(row=3, column=0, sticky=W)
lab_tf.grid(row=3, column=2, sticky=W)

ed_i_pos = Entry(container, width=26,font='times 12')
ed_i_vel = Entry(container, width=26,font='times 12')
ed_ti = Entry(container, width=7,font='times 12')
ed_tf = Entry(container, width=7,font='times 12')

ed_i_pos.grid(row=1, column=2, columnspan=2, sticky=W)
ed_i_vel.grid(row=2, column=2,  columnspan=2, sticky=W)
ed_ti.grid(row=3, column=1, sticky=W)
ed_tf.grid(row=3, column=3, sticky=W)


lb1 = Label(container, text='', height=2, font='times 12 italic')
lb1.grid(row=4, column=1, columnspan=2)

bt = Button(container, text = 'Gerar Simulação', font='times 14 bold', pady=12, command=bt_clic)
bt.grid(row=5, column=1, columnspan=2, sticky=S)

janela.mainloop()
```

Reinicie o kernel e limpe as saídas (ver Informes Iniciais) e comece a execução dos códigos a partir da célula abaixo para uma melhor facilidade na vizualização

4.4 Adicionando Outra Janela

Caso haja o interesse em adicionar uma janela para pôr outras informações na simulação, é possível adicionar um botão à GUI que, ao ser clicado, abre uma outra janela construída por meio da função `ajuda()`. Nesse caso, utilizamos a janela de **ajuda**, contendo uma breve explicação do movimento e da simulação.

[ ]:
```
from vpython import *
from tkinter import *

graf_pos = graph(xtitle='tempo (s)', ytitle='Posição (m)')
curva_pos = gcurve(color=color.red, label = 'Coordenada X')

def ajuda():
    newWindow = Toplevel(janela)
    newWindow.geometry('520x480+100+150')
    newWindow.title('AJUDA')
    newWindow['pady']=20
    newWindow['padx']=30
    lab_title2 = Label(newWindow, text='Movimento Retilíneo Uniforme',␣
↪font='times 20 bold', height=3)
    lab_title2.grid(row=0, column=0)
    a = 'O movimento retilíneo uniforme é caracterizado por um corpo se movendo␣
↪em '
    a = a + 'velocidade constante durante toda a sua trajetória (retilínea),␣
↪resultando em uma '
    a = a + 'velocidade instantânea igual à velocidade média. Com a posição␣
↪inicial, velocidade inicial e '
    a = a + 'intervalo de tempo (tempo inicial e tempo final), é possível se␣
↪ter uma descrição exata do  movimento. '
    a = a + '\n\nNa simulação é possível visualizar a trajetória do corpo␣
↪formada por pontos que '
    a = a + 'aparecem em intervalos de tempos iguais. Nela é possível verificar␣
↪que o espaço de um ponto '
    a = a + 'a outro é simétrico, o que indica que a variação da posição do␣
↪corpo em relação ao tempo é '
    a = a + 'constante para toda trajetória. (Em alguns casos, no início da␣
↪simulação a trajetória pode conter '
    a = a + 'algumas incoerências devido a um atraso na inicialização da␣
↪simulação). Outrossim, é possível '
    a = a + 'visualizar a origem, o final e o ponto médio da trajetória, assim␣
↪como a posição em que o corpo '
    a = a + 'se encontra em instantes diferentes, possibilitando uma melhor␣
↪compreensão das escalas '
```

```
    a = a + 'utilizada em cada caso. '
    lab_desc = Label(newWindow, font='times 12 ',text=a,  wraplength=450)
    lab_desc.grid(row=1, column=0)


#evento associado ao botão Gerar Simulação
def bt_clic():
        try:
            x_inicial=float(ed_i_pos.get())
            v=float(ed_i_vel.get())
            t_inicial=float(ed_ti.get())
            t_final=float(ed_tf.get())

            #Simulação
            if (v<0):
                p = 10
            else:
                p=-10

            #Perceba que vamos inserir os parâmetros da trajetória prórpia␣
 ↪esfera
            delt_t = t_final - t_inicial #tempo de execução da simulação␣
 ↪(variação do tempo)
            esfera = sphere(pos = vec(p, 0, 0), radius = 0.4, color = color.
 ↪red, opacity=1, make_trail=True, interval=10, trail_type="points",␣
 ↪trail_color =color.yellow)
            texto_x_inicial = label(pos=vec(p, 0, 0), text= str(x_inicial) + "␣
 ↪m", xoffset=0, yoffset=-100, space=30, height=16, border=4) #indica o ponto␣
 ↪inicial
            texto_x_medio = label(pos=vec(0, 0, 0), text=␣
 ↪str((v*delt_t+x_inicial)/2)+ " m", xoffset=0, yoffset=-100, space=30,␣
 ↪height=16, border=4, font='sans') #indica o ponto médio
            texto_x_final = label(pos=vec(p*-1, 0, 0), text=␣
 ↪str(v*delt_t+x_inicial)+ " m", xoffset=0, yoffset=-100, space=30, height=16,␣
 ↪border=4, font='sans') #indica o ponto final
            texto_posicao = label(pos=vec(0, 4, 0), text= "Posição: " +␣
 ↪str(x_inicial)+ " m", space=30, height=16, border=4, font='sans')

            t = 0
            e = abs(v*delt_t)/20 #fator de controle de escala
            janela.destroy()

            while (t < delt_t):
                rate (100)
                t+=0.01
```

```
                esfera.pos = vec(p + (v*t)/e, 0, 0)#Perceba a inserção do fator␣
↪de escala apenas na variação da posição
                curva_pos.plot(pos=(t_inicial + t, x_inicial + v*t)) #plotagem␣
↪do gráfico
                texto_posicao.text = 'Posição: ' + str(round (x_inicial + v*t,␣
↪2))+ ' m'
        except:
            lb1['text']='Valores Informados Inválidos!!!'

#O código abaixo se refere exclusivamente à criação de janelas do TKinter      ␣
↪
janela = Tk()
janela.geometry('530x500+400+150')
janela.title('SIMULAÇÕES')
janela['pady']=20
janela['padx']=30

container = Frame(janela, highlightbackground='black', highlightthickness=2)
container['pady']=20
container['padx']=35
container.grid(row=0, column=0)

lab_title = Label(container, text='Movimento Retilíneo Uniforme', font='times␣
↪20 bold', height=3)
lab_i_pos = Label(container, text='Posição Inicial (m):', font='times 14 ',␣
↪height=3)
lab_i_vel = Label(container, text='Velocidade Inicial (m/s):', font='times 14␣
↪', height=3)
lab_ti = Label(container, text='t_inicial(s):', font='times 14 ', height=3)
lab_tf = Label(container, text='t_final (s):', font='times 14 ', height=3)

lab_title.grid(row=0, column=0, columnspan=4)
lab_i_pos.grid(row=1, column=0, columnspan=2, sticky=W)
lab_i_vel.grid(row=2, column=0, columnspan=2, sticky=W)
lab_ti.grid(row=3, column=0, sticky=W)
lab_tf.grid(row=3, column=2, sticky=W)

ed_i_pos = Entry(container, width=26,font='times 12')
ed_i_vel = Entry(container, width=26,font='times 12')
ed_ti = Entry(container, width=7,font='times 12')
ed_tf = Entry(container, width=7,font='times 12')

ed_i_pos.grid(row=1, column=2, columnspan=2, sticky=W)
ed_i_vel.grid(row=2, column=2,  columnspan=2, sticky=W)
ed_ti.grid(row=3, column=1, sticky=W)
ed_tf.grid(row=3, column=3, sticky=W)
```

```
lb1 = Label(container, text='', height=2, font='times 12 italic')
lb1.grid(row=4, column=1, columnspan=2)

bt = Button(container, text = 'Gerar Simulação', font='times 14 bold', pady=12,␣
 ↪command=bt_clic)
bt.grid(row=5, column=1, columnspan=2, sticky=S)
buttonExample = Button(container, text='Ajuda', font='times 12 bold',␣
 ↪command=ajuda) #Botão de ajuda
buttonExample.grid(row=5, column=0, columnspan=2, sticky=W)

janela.mainloop()
```

Apesar de se diferenciar quanto às equações e formas utilizadas, todos os outros simuladores seguem essa mesma lógica. Ou seja, compreendendo a lógica de construção dos simuladores, juntamente com o conhecimento em Física, é possível se desenvolver uma série de outros simuladores.

[ ]: