

PROJETO FINAL

COMENTÁRIOS SOBRE O ALGORITMO PLANETÁRIO NUMÉRICO E DE COMO
PROCEDEU O SEU DESENVOLVIMENTO.

MÓDULO DE ENTRADA

ARQUIVO DE TEXTO .TXT QUE SERÁ LIDO E AS INFORMAÇÕES TRATADAS DE MODO ADEQUADO SERÃO ARMAZENADOS NA MEMÓRIA. TODOS OS DADOS FORAM CONSIDERADOS NO 'SI'.

Processo de Leitura

#PLANETAS

#CONTATO_DEM

#CONTATO_INTERACAO

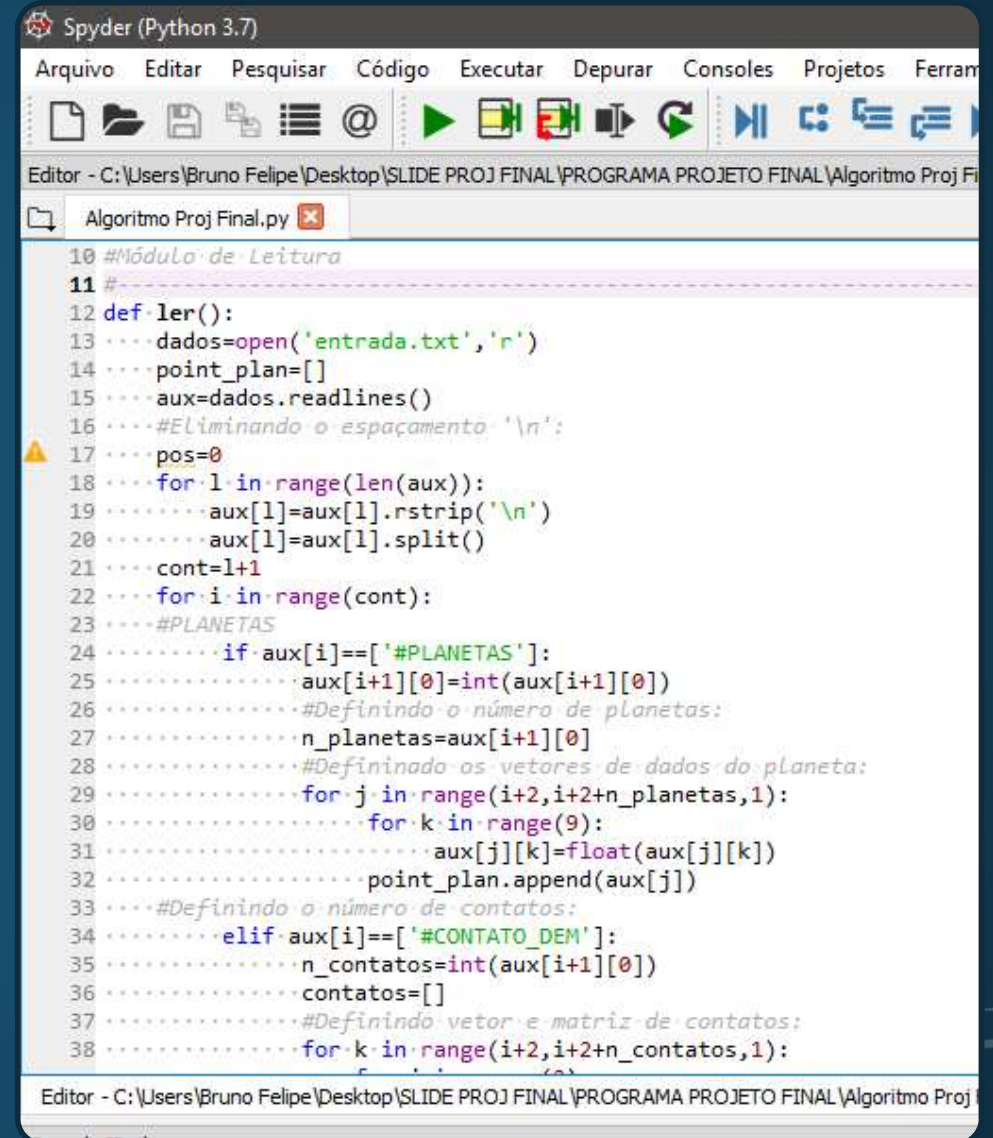
#INTEGRADOR

#FIM

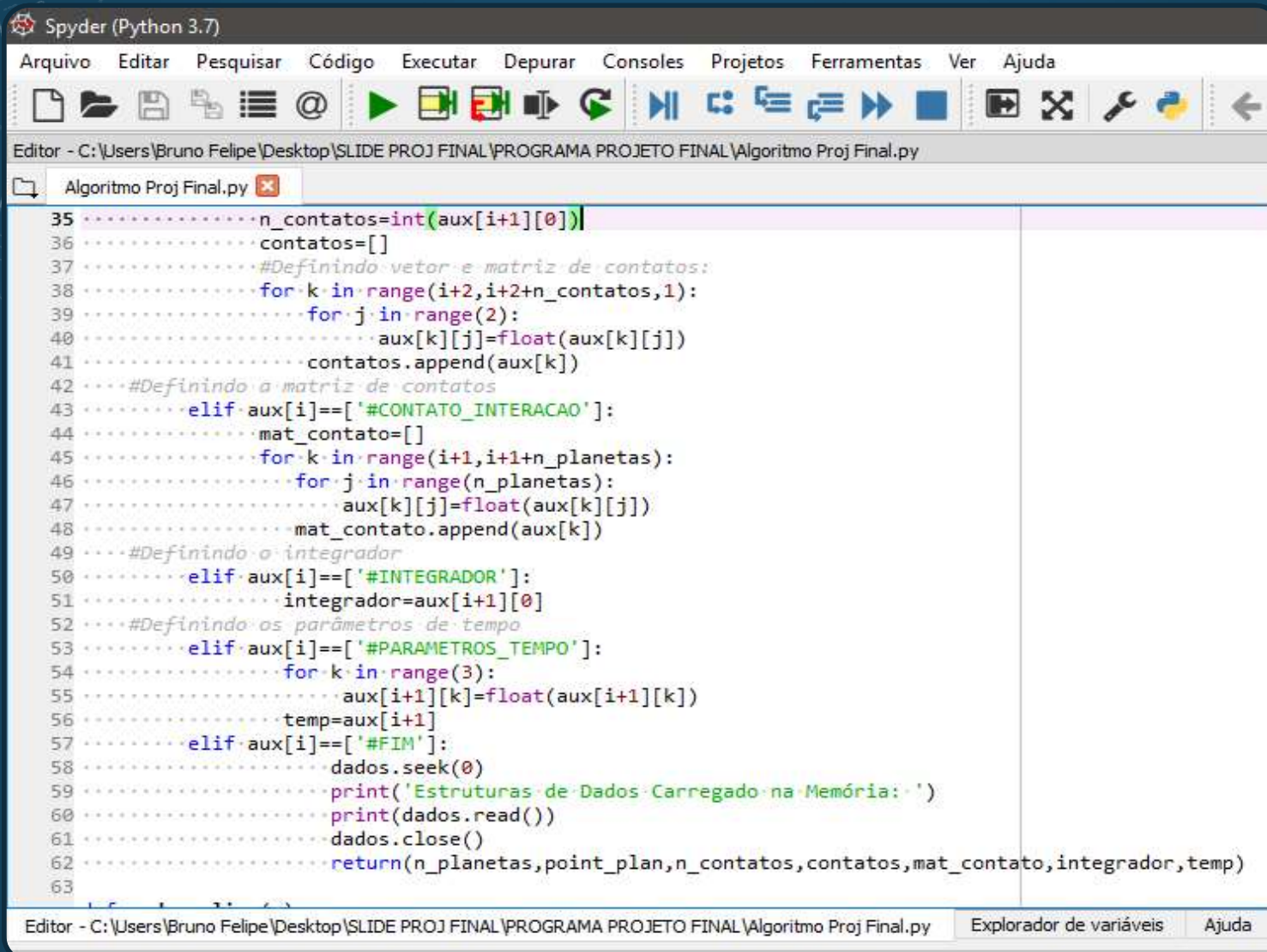


FUNÇÃO LER()

- Abertura de leitura: o nome do arquivo a ser lido deve ser "entrada", e sua extensão: .txt.
- Foi aplicado o readlines(), transformando em lista as informações das linhas de texto.
- Foi aplicado rstrip("\n") e split() para cada elemento da lista obtida pelo readlines(). Além disso foi contado quantas linhas o arquivo de entrada apresentava.
- Por meio da iteração da lista de dados previamente tratado do arquivo entrada.txt, estabeleceu-se uma iteração em relação ao número de linhas, localizando cada tag e o referenciando a sua posição na lista, tomando-se que por padrão o arquivo de entrada será de acordo com o que foi estabelecido na apresentação do projeto.
- Assim o conjunto de dados abaixo de cada tag foi direcionado para as variáveis definidas ao longo da função, para que posteriormente pude-se ser retornada para o módulo de análise de maneira adequada.

The image shows a screenshot of the Spyder Python IDE interface. The main window displays a Python script named 'Algoritmo Proj Final.py'. The code defines a function 'ler()' that reads a file 'entrada.txt' and processes its contents. The function uses 'readlines()' to read the file into a list 'aux'. It then iterates through the list, stripping trailing newlines and splitting each line. It identifies tags like '#PLANETAS' and '#CONTATO_DEM' and extracts numerical data into lists and matrices. The IDE's menu bar includes 'Arquivo', 'Editar', 'Pesquisar', 'Código', 'Executar', 'Depurar', 'Consoles', 'Projetos', and 'Ferramentas'. The status bar at the bottom shows the file path: 'C:\Users\Bruno Felipe\Desktop\SLIDE PROJ FINAL\PROGRAMA PROJETO FINAL\Algoritmo Proj Final\Algoritmo Proj Final.py'.

FUNÇÃO LER()



```
35 .....n_contatos=int(aux[i+1][0])
36 .....contatos=[]
37 .....#Definindo vetor e matriz de contatos:
38 .....for k in range(i+2,i+2+n_contatos,1):
39 .....    for j in range(2):
40 .....        aux[k][j]=float(aux[k][j])
41 .....        contatos.append(aux[k])
42 .....#Definindo a matriz de contatos
43 .....elif aux[i]==['#CONTATO_INTERACAO']:
44 .....    mat_contato=[]
45 .....    for k in range(i+1,i+1+n_planetas):
46 .....        for j in range(n_planetas):
47 .....            aux[k][j]=float(aux[k][j])
48 .....            mat_contato.append(aux[k])
49 .....#Definindo o integrador
50 .....elif aux[i]==['#INTEGRADOR']:
51 .....    integrador=aux[i+1][0]
52 .....#Definindo os parâmetros de tempo
53 .....elif aux[i]==['#PARAMETROS_TEMPO']:
54 .....    for k in range(3):
55 .....        aux[i+1][k]=float(aux[i+1][k])
56 .....        temp=aux[i+1]
57 .....elif aux[i]==['#FIM']:
58 .....    dados.seek(0)
59 .....    print('Estruturas de Dados Carregado na Memória:')
60 .....    print(dados.read())
61 .....    dados.close()
62 .....    return(n_planetas,point_plan,n_contatos,contatos,mat_contato,integrador,temp)
63
```

Editor - C:\Users\Bruno Felipe\Desktop\SLIDE PROJ FINAL\PROGRAMA PROJETO FINAL\Algoritmo Proj Final.py

Explorador de variáveis Ajuda

dados - Tupla (7 elementos)

Índice	Tipo	Tamanho	Valor
0	int	1	4
1	list	4	[[0.0, 7844000000.0, 0.0, 0.0, -26080.0, ...], [1.0, 0.0, 0.0, 0.0, 0. ...
2	int	1	1
3	list	1	[[1.0, 1e+200]]
4	list	4	[[1.0, 1.0, 1.0, 1.0], [1.0, 1.0, 1.0, 1.0], [1.0, 1.0, 1.0, 1.0], [1. ...
5	str	1	EULER
6	list	3	[50.0, 4000000.0, 200.0]

1 - Lista (4 elementos)

Índice	Tipo	Tamanho	Valor
0	list	9	[0.0, 7844000000.0, 0.0, 0.0, -26080.0, 31080.0, 0.0, 1.0, 7.34e+22]
1	list	9	[1.0, 0.0, 0.0, 0.0, 0.0, 30000.0, 0.0, 16371000.0, 5.97e+26]
2	list	9	[2.0, 149600000000.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.989e+30]
3	list	9	[3.0, -149600000000.0, 0.0, 0.0, 0.0, -25000.0, 0.0, 1.0, 1.989e+30]

Fechar

4 - Lista (4 elementos)

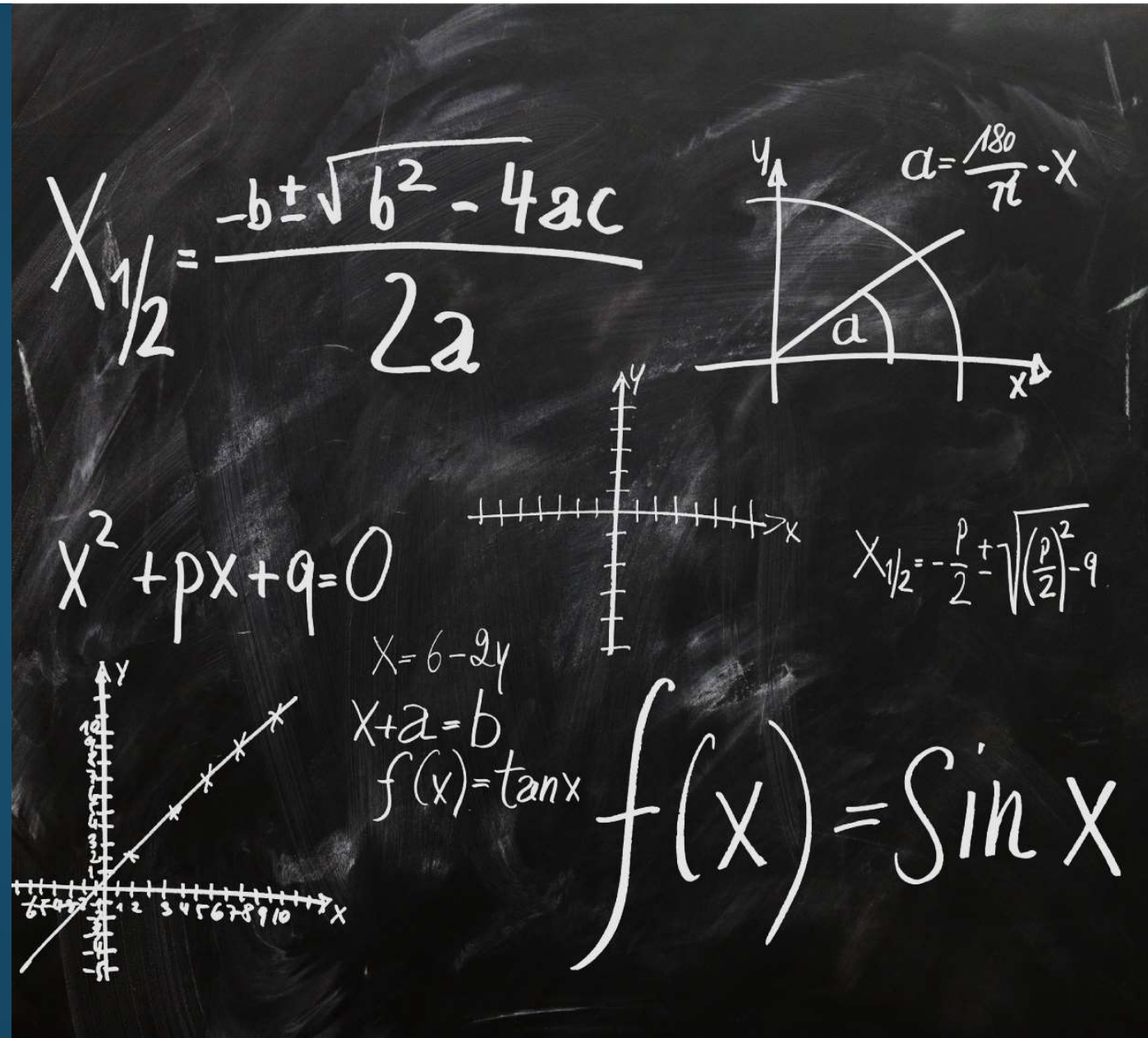
Índice	Tipo	Tamanho	Valor
0	list	4	[1.0, 1.0, 1.0, 1.0]
1	list	4	[1.0, 1.0, 1.0, 1.0]
2	list	4	[1.0, 1.0, 1.0, 1.0]
3	list	4	[1.0, 1.0, 1.0, 1.0]

Fechar

Fechar

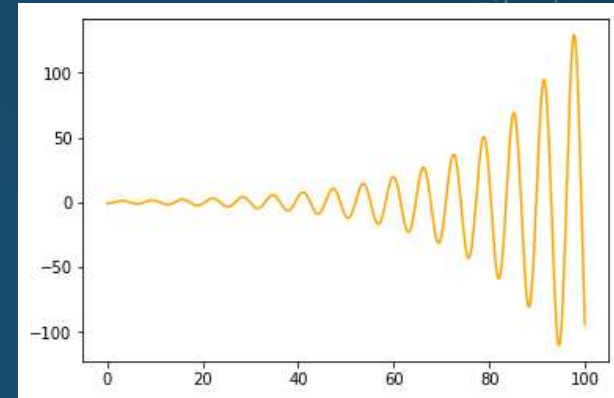
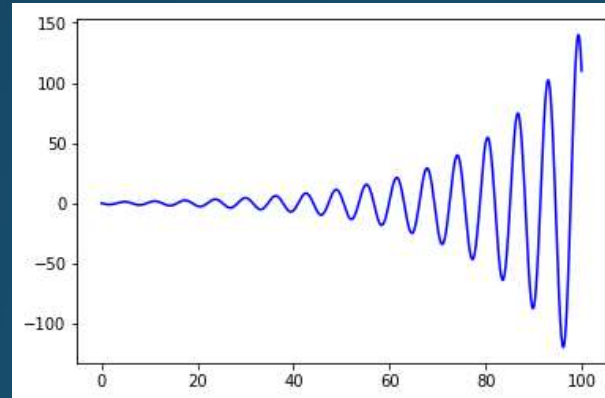
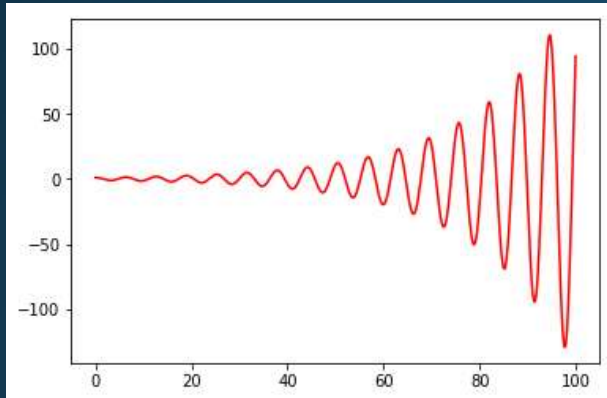
MÓDULO DE ANÁLISE

O algoritmo foi implementado considerando a posição, velocidade, força e aceleração como vetores.

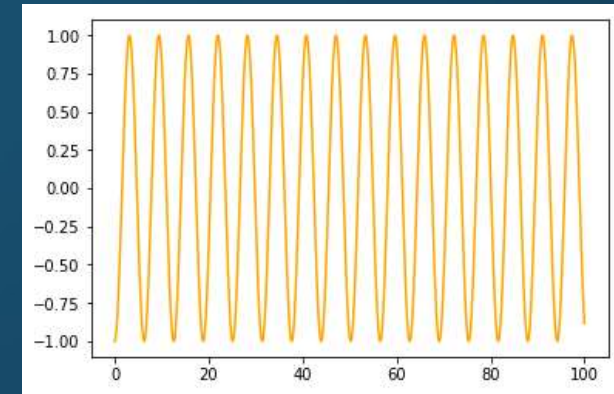
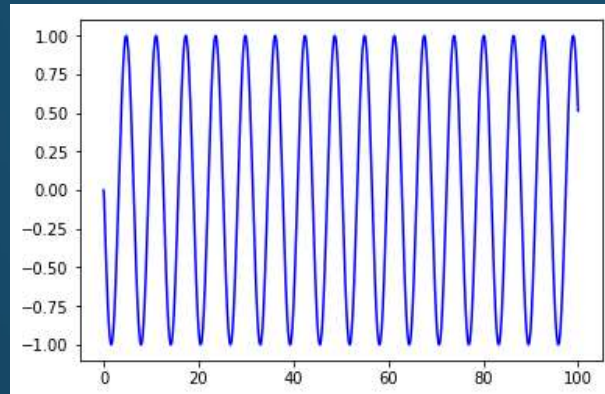
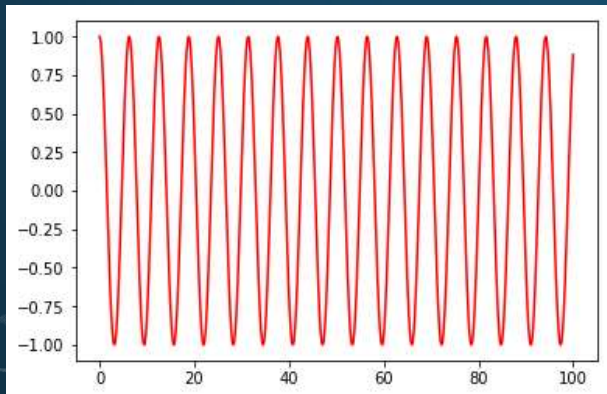


SISTEMA MASSA MOLA – EULER E VERLET

EULER



VERLET



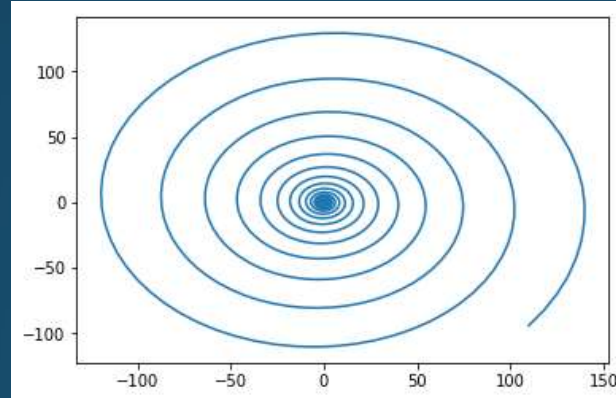
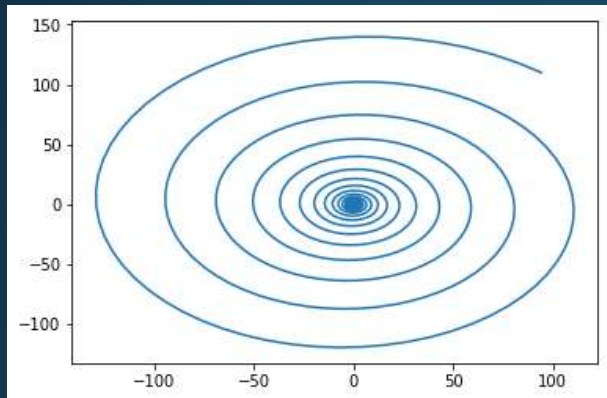
TEMPO X DESLOCAMENTO

TEMPO X VELOCIDADE

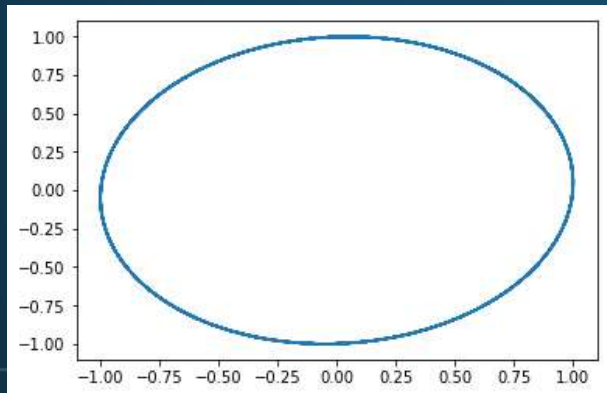
TEMPO X ACELERAÇÃO

SISTEMA MASSA MOLA – EULER E VERLET

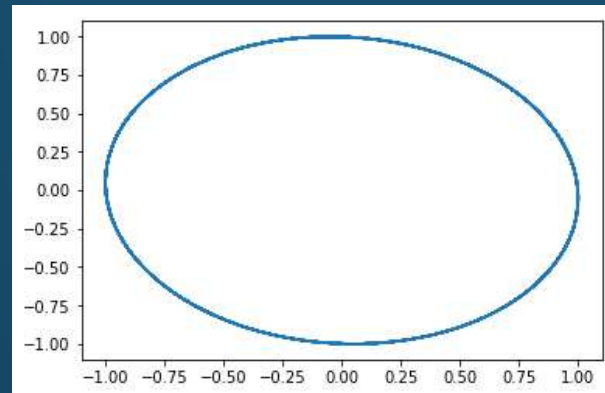
EULER



VERLET



DESLOCAMENTO X VELOCIDADE



VELOCIDADE X ACELERAÇÃO

VERLET MANTÉM O SISTEMA CONSERVATIVO.

Tempo final: 100

Passo adotado: 0.1

FUNÇÃO MOD_ANALISE()

```
64 def mod_analise(a):
65     #
66     import numpy as np
67     import math
68     a=dados
69     cont=0
70     contP=0
71     #Constante Gravitacional Universal (SI)
72     G=6.67384*10**(-11) #m³.Kg³.s^(-2) ou N.m².Kg²
73     #Tempo de iteração
74     t=0
75     #Tempo Final
76     tf=a[6][1]
77     #Incremento de Tempo
78     dt=a[6][0]
79     #Número de Planetas
80     numplan=a[0]
81     #Passo
82     Passo=a[6][2]
83     #Matriz de Vetores de posição, velocidade e aceleração Iniciais
84     r0=np.zeros((numplan,3),dtype=float)
85     v0=np.zeros((numplan,3),dtype=float)
86     v1=np.zeros((numplan,3),dtype=float)
87     r1=np.zeros((numplan,3),dtype=float)
88     raioP=np.zeros((numplan,1),dtype=float)
89     massa=np.zeros((numplan,1),dtype=float)
90     for i in range(numplan):
91         for j in range(3):
92             r0[i][j]=a[1][i][j+1]
93             v0[i][j]=a[1][i][j+4]
94             raioP[i]=a[1][i][7]
95             massa[i]=a[1][i][8]
96         #
97     for i in range(numplan):
98         for j in range(numplan):
99             for k in range(a[2]):
100                 if a[3][k][0]==a[4][i][j] and i!=j:
101                     a[4][i][j]=a[3][k][1]
102     if a[5]=='VERLET':
```

- Foi estabelecido o valor da constante Gravitacional Universal.
- Tomou-se um tem $t = 0$ como referência para contagem do tempo, onde este é incrementado por um dt a cada iteração até finalizar em t_f , que é o tempo final.
- O número de planetas foi armazenado.
- Criou-se os vetores r_0 , v_0 , $raioP$, massa para receberem os valores iniciais correspondentes as posições e velocidades iniciais de cada planeta por linha, o raio de cada planeta por linha, como também a massa.
- Criou-se o vetor r_1 , que é um auxiliar, para receber os valores atualizados de cálculo das posições por linha de cada planeta e a partir dele, substituir no vetor posição que irá iterar novamente com EULER ou VERLET.
- A próxima iteração em i , j e k foi realizada para acessar os dados da constante da mola com os seus referidos ids, essa estrutura, desconsiderará a diagonal principal da matriz de contato dos planetas, e irá substituir os valores fora dessa diagonal pelo respectivo valor do id associado.
- Inicia-se então a condição de VERLET, caso o usuário tenha fornecido essa string no arquivo txt.

FUNÇÃO MOD_ANALISE()

- No começo da condição do algoritmo de VARLET foi estabelecido os vetores força e a_c , isto é, força e aceleração dos n planetas que sofrerão constantes atualizações e também foi preferível tomar o vetor r , que recebe o vetor inicial posição dos planetas, e que sofrerá atualizações ao longo das iterações de tempo.

- Com o while loop, condicionando sempre que $t < t_f$ e t a cada iteração ficará recebendo o seu próprio valor mais o incremento de tempo dt , temos:

1. Os N arquivos de gravação gerados de cada planeta receberão o tempo t no instante de cálculo, além da posição do planeta naquele instante, isto é, as coordenadas x, y, z . A gravação respeita a quantidade de passos fornecido no arquivo de entrada, assim foi estabelecido dois contadores, contP que conta o número de iterações de cálculo e o cont , que recebe o seu próprio valor mais o passo, caso haja a igualdade de contP com cont . Esse padrão foi definido em virtude de agilizar a escrita do arquivo e dependendo das dimensões do problema e da precisão, os parâmetros de passo precisão podem ser alterados.
2. A próxima etapa é a verificação da distância entre os planetas i, j de interesse, caso esse valor seja menor do que a soma dos seus raios, então será calculado a distância dos planetas, a soma dos raios, obteremos o Valor da norma do vetor dX pela diferença da distância da eminência de contado pela nova distância após a penetração. Essa norma é multiplicada pelo vetor direção que une os pontos de posição dos planetas naquele instante. Por fim, o vetor dX recebe esse cálculo e o mesmo é multiplicado pelo valor da mola referente a posição da matriz de contato. E essa força, Chamada força de contato, é somada a força principal da gravitação.

```
C:\Users\Bruno Felipe\Desktop\SLIDE PROJ FINAL\PROGRAMA PROJETO FINAL
Editor - C:\Users\Bruno Felipe\Desktop\SLIDE PROJ FINAL\PROGRAMA PROJETO FINAL\Algoritmo Proj Final.py
Algoritmo Proj Final.py
102 ...if a[5]=='VERLET':
103 .....forca=[]
104 .....ac=[]
105 .....for i in range(numplan):
106 .....forca.append([0])
107 .....ac.append([''])
108 .....r=np.copy(r0)
109 .....#-----
110 .....while t<=tf:
111 .....if contP==cont:
112 .....for i in range(numplan):
113 .....f=open('planeta-Nº-{}.txt'.format(i+1),'a')
114 .....f.write('{}-{}-{}-{}\n'.format(t,r[i][0],r[i][1],r[i][2]))
115 .....f.close()
116 .....cont=cont+Passo
117 .....dX=0
118 .....for i in range(numplan):
119 .....for j in range(numplan):
120 .....if i!=j:
121 .....distplan=np.linalg.norm(r[j]-r[i])
122 .....somRaios=raioP[i]+raioP[j]
123 .....vetdir=(r[j]-r[i])/np.linalg.norm(r[j]-r[i])
124 .....normdX=abs(distplan-somRaios)
125 .....if distplan<somRaios:
126 .....dX=normdX*vetdir
127 .....else:
128 .....dX=0
129 .....forca[i]=forca[i]+G*massa[i]*massa[j]*(r[j]-r[i])/
((np.linalg.norm(r[j]-r[i]))**3)+a[4][i][j]*dX
130 .....ac[i]=forca[i]/massa[i]
131 .....if t==0:
132 .....k=dt/2
133 .....else:
134 .....k=dt
135 .....v1[i]=v0[i]+k*ac[i]
136 .....r1[i]=r[i]+dt*v1[i]
137 .....v0[i]=np.copy(v1[i])
138 .....r[i]=np.copy(r1[i])
139 .....forca[i]=0
140 .....t=round(t+dt,10)
141 .....contP=contP+1
142 .....#-----
```

FUNÇÃO MOD_ANALISE()

- Δx assume zero, caso a distância dos planetas seja maior que a soma dos seus raios.
- O vetor 'força' em i receberá o seu próprio valor, em que i se mantém fixo até o final da variação de j , realizando-se o somatório de todas as forças dos planetas j sobre o planeta i . Lembrando-se ainda que toda essa condição deve valer para o valor i diferente de j . Em seguida, a força em i é dividida pela massa em i , obtendo-se então a aceleração do planeta i . Nessa condição, para VERLET, temos uma condicional sobre k , e esse k substitui o Δt na equação da velocidade. Pois caso o t seja nulo, temos então que k recebe $\Delta t/2$, conforme a condição inicial do método. E quando t incrementa em Δt , k assume Δt , assim respeitando a equação para VERLET em um tempo não nulo.
- Em seguida é calculado a nova posição do planeta i , e os vetores da velocidade inicial v_o e r são atualizados em i .
- Por fim, o vetor 'força' em i recebe um valor nulo, para que este funcione como elemento neutro da adição e assim obter o valor da nova força em i Para uma nova somatória.

EULER:

O que diferencia esse algoritmo para o de VERLET, é que k é uma constante e igual a dt no tempo zero, na equação da velocidade.

C:\Users\Bruno Felipe\Desktop\SLIDE PROJ FINAL\PROGRAMA PROJETO FINAL

Editor - C:\Users\Bruno Felipe\Desktop\SLIDE PROJ FINAL\PROGRAMA PROJETO FINAL\Algoritmo Proj Final.py

Algoritmo Proj Final.py

```
102 ...if a[5]=='VERLET':
103     forca=[]
104     ac=[]
105     for i in range(numplan):
106         forca.append([0])
107         ac.append([''])
108     r=np.copy(r0)
109     #-----
110     while t<=tf:
111         if contP==cont:
112             for i in range(numplan):
113                 f=open('planeta-Nº-{}.txt'.format(i+1),'a')
114                 f.write('{}-{}-{}\n'.format(t,r[i][0],r[i][1],r[i][2]))
115                 f.close()
116                 cont=cont+Passo
117             dX=0
118             for i in range(numplan):
119                 for j in range(numplan):
120                     if i!=j:
121                         distplan=np.linalg.norm(r[j]-r[i])
122                         somRaios=raioP[i]+raioP[j]
123                         vetdir=(r[j]-r[i])/np.linalg.norm(r[j]-r[i])
124                         normdX=abs(distplan-somRaios)
125                         if distplan<somRaios:
126                             dX=normdX*vetdir
127                         else:
128                             dX=0
129                         forca[i]=forca[i]+G*massa[i]*massa[j]*(r[j]-r[i])/
((np.linalg.norm(r[j]-r[i]))**3)+a[4][i][j]*dX
130                     ac[i]=forca[i]/massa[i]
131                 if t==0:
132                     k=dt/2
133                 else:
134                     k=dt
135                 v1[i]=v0[i]+k*ac[i]
136                 r1[i]=r[i]+dt*v1[i]
137                 v0[i]=np.copy(v1[i])
138                 r[i]=np.copy(r1[i])
139                 forca[i]=0
140                 t=round(t+dt,10)
141                 contP=contP+1
142     #-----
```

Permissões: RW Fim de linha: CRLF Codificação: UTF-8 Linha: 293 Coluna: 20 Memória: 46 %

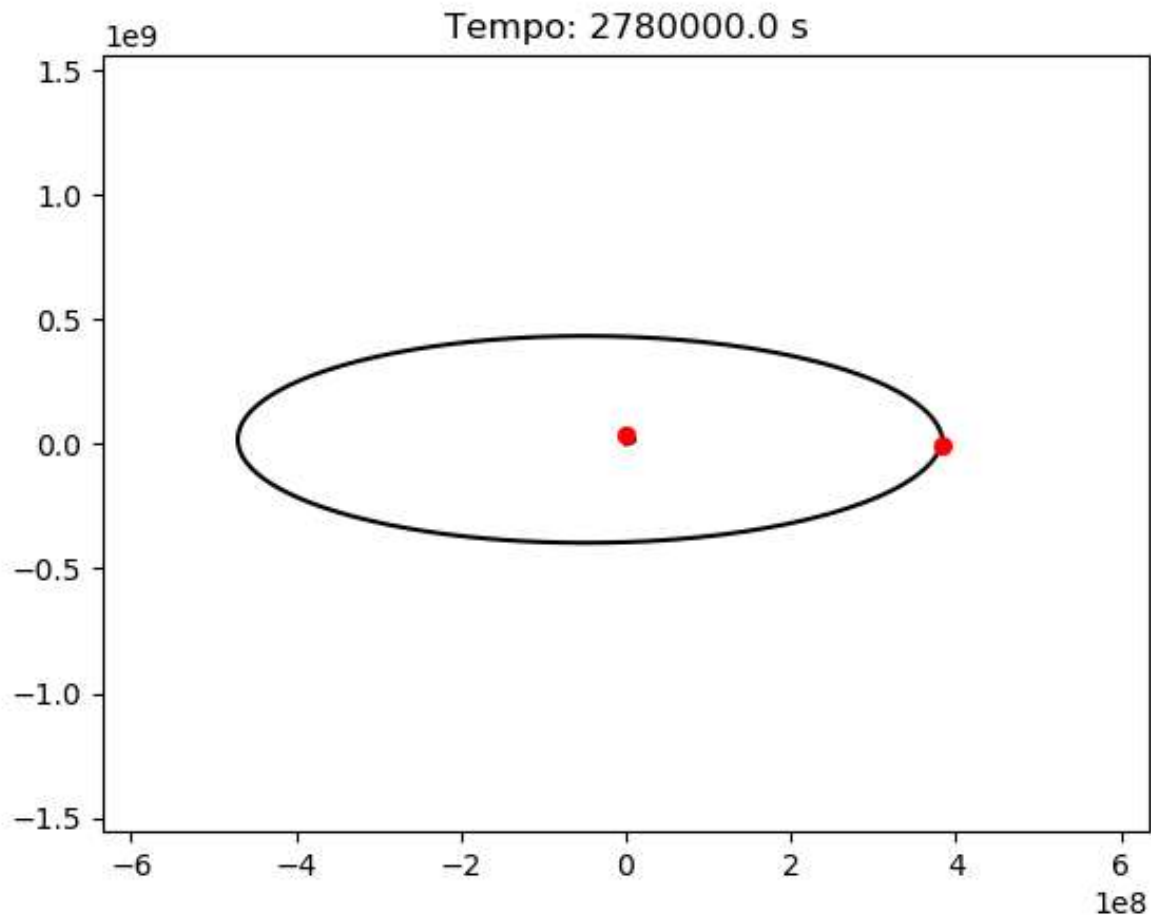
FUNÇÃO VISUALIZACAO()

- Os dados de textos dos planetas armazenados no diretório do algoritmo são lidos, o usuário deverá entrar com o tipo de plot que o mesmo deseja verificar.
- Após a remoção da quebra linha de texto e de transformar os elementos da linha do arquivo texto em vetor, assim os dados de tempo e de posição são salvas na matriz de vetores planetas. Os dados a partir daqui começam a serem tratados. O vetor temp receberá todos os tempos de iteração salvos em um dos arquivos lidos, e excluirá toda Informação de tempo da matriz de vetores planetas.
- É definido os vetores x,y e z que receberão datax, datay e dataz respectivamente em cada iteração. Os vetores datax, datay e dataz armazenarão todos os elementos da posição do planeta 1, por exemplo, que são uma lista de elementos, assim os vetores x,y e z guardam essa lista. Logo, cada elemento dos vetores x,y e z correspondem os dados completos do planeta em questão.
- Por meio dessa ideia foi possível plotar cada elemento e uma lista limitada de x,y e z para obter a plotagem de todos os N planetas, e também de suas trajetórias ao longo da iteração.

```
184 def visualizacao(dados):
185     numplan=dados[0]
186     modelo=input('Entre com o tipo de plot (2d) ou (3d): ')
187     import matplotlib.pyplot as plt
188     from mpl_toolkits import mplot3d
189     import math
190     import time
191     planetas=[]
192     numelem=0
193     temp=[]
194     for i in range(numplan):
195         f=open('planeta N°.{}.txt'.format(i+1), 'r')
196         linha=f.readlines()
197         aux=[]
198         for j in range(len(linha)):
199             if i==0:
200                 numelem=numelem+1
201                 linha[j]=linha[j].rstrip('\n')
202                 linha[j]=linha[j].split()
203                 for k in range(4):
204                     linha[j][k]=float(linha[j][k])
205                 planetas.append(linha)
206             for i in range(numplan):
207                 for j in range(len(linha)):
208                     if i==0:
209                         temp.append(planetas[i][j][0])
210                         del(planetas[i][j][0])
211                     else:
212                         del(planetas[i][j][0])
213                 x=[]
214                 y=[]
215                 z=[]
216                 if modelo=='2d':
217                     fig=plt.figure('Planetas')
218                     ax=plt.subplot()
219                     for i in range(numplan):
220                         datax=[]
221                         datay=[]
222                         for j in range(len(linha)):
223                             datax.append(planetas[i][j][0])
224                             datay.append(planetas[i][j][1])
225                             x.append(datax)
```

Permissões: RW Fim de linha: CRLF Codificação: UTF-8 Linha: 293 Coluna: 20 Memória: 48 % 13:07

DIFICULDADES.



The background is a solid dark blue. It features several faint, light blue technical diagrams. In the top right, there is a large circular gauge with concentric circles and radial lines, resembling a speedometer or a dial. In the bottom right, there is a diagram of concentric circles with arrows indicating a clockwise direction. In the bottom left, there is a partial view of a similar circular diagram. The text "OBRIGADO!" is centered in the middle of the image in a white, sans-serif font.

OBRIGADO!