O Planetário

Trabalho final do processo seletivo do LCCV-UFAL
Aluno:
Pedro Henrique Rodrigues Meira Bastos

O projeto

- Montar um software que faça a simulação do movimento de corpos celestes dados pelo usuário;
- O projeto será dividido em três partes:

Créditos da Imagem: Danielle Futselaar

Módulo de leitura:

- Responsável pela leitura dos dados de entrada, para que possam ser usados pelo módulo de análise;
- Os dados de entrada são fornecidos por um arquivo no seguinte molde:

```
#PLANETAS
(número de planetas)
(idPlaneta) (x) (y) (z) (v0x) (v0y) (v0z) (raio) (massa)
#CONTATO_DEM
(número de modelos de contato)
(idModelo de Contato) (Kn)
#CONTATO_INTERACAO
(contato(1,1).id) ... (contato(1,nPlanetas).id)
...
(contato(nPlanetas,2).id) ... (contato(nPlanetas,nPlanetas).id)
#INTEGRADOR
(nome do integrador, VERLET ou EULER)
#PARAMETROS_TEMPO(dt) (tempo_final_simulacao) (numero_de_passos_impressao)
#FIM
```

Créditos da imagem: o autor

Módulo de análise:

- Realiza os cálculos para obter posição e velocidade dos planetas em um número de instantes fornecidos pelo usuário;
- Retorna um “histórico” com um intervalo igualmente espaço determinado pelo usuário;
- Para os cálculos, as seguintes equações são utilizadas:

$$\boldsymbol{F_i} = \sum_{j}^{nPlanetas} \left(\frac{G m_i m_j \boldsymbol{r}}{r^3} + \boldsymbol{F}^{\boldsymbol{ij}}_{\boldsymbol{contato}} \right)$$

Somatório das forças gravitacional e de contato

$$\boldsymbol{F}^{\boldsymbol{ij}}_{\boldsymbol{contato}} = K_n \Delta \boldsymbol{x}$$

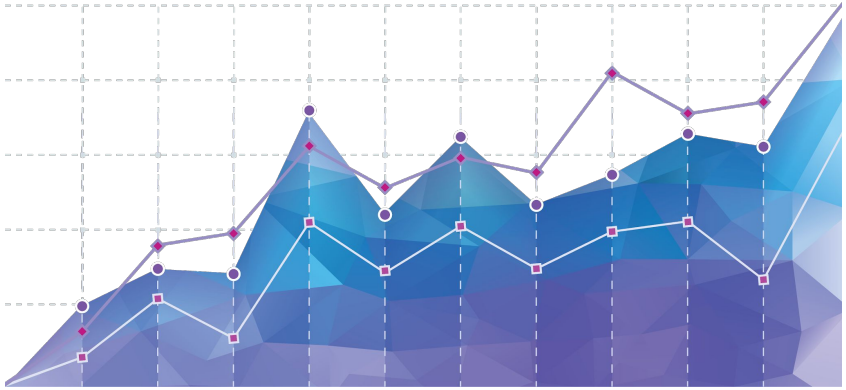
Força de contato expressada como força elástica

$$\boldsymbol{r}_i^n = \boldsymbol{r}_i^{n-1} + \Delta t \boldsymbol{v}_i^n$$
$$\boldsymbol{v}_i^n = \boldsymbol{v}_i^{n-1} + \Delta t \boldsymbol{a}_i^n$$

Método de Euler para achar velocidade e posição

Módulo de visualização

- Permite que o usuário visualize os resultados obtidos a partir de gráficos (pelo menos 2D) mostrando o movimento dos corpos ao longo do tempo estabelecido.



Créditos da Imagem: Analytics Insight

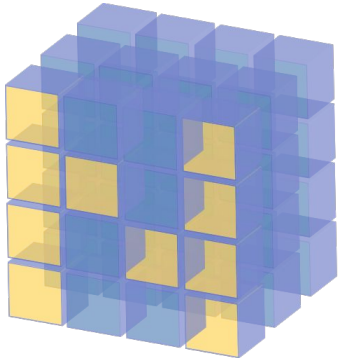
Ferramentas utilizadas:

- Linguagem Python;
- Versão 3.7.3;
- Interpretada;
- De alto nível;
- Permite orientação à objetos.

Créditos da Imagem: Python Foundation

Ferramentas utilizadas:

- Módulo NumPy;
- Versão 1.16.3;
- Faz operações matemáticas mais avançadas que o Python não oferece por padrão (matrizes, por exemplo);
- Backend em C e FORTRAN



NumPy

Créditos da imagem: David Cournapeau

Ferramentas Utilizadas

- Módulo SciPy
- Versão 1.2.1
- Faz operações com um “viés mais científico” que o NumPy, que o Python também não oferece por default
- Backend em C e FORTRAN

Créditos da imagem: Scipy Organization

Ferramentas utilizadas:

- Módulo MatPlotLib;
- Versão 1.16.3;
- Faz operações matemáticas mais avançadas que o Python não oferece por padrão (matrizes, por exemplo);
- Backend em C e FORTRAN

Créditos da imagem: MatPlotLib

Ferramentas utilizadas:

- Módulo Pandas;
- Versão 0.24.2;
- Módulo desenvolvido para aplicações de ciência de dados;
- Compatível com NumPy e MatplotLib

Créditos da imagem: WWF

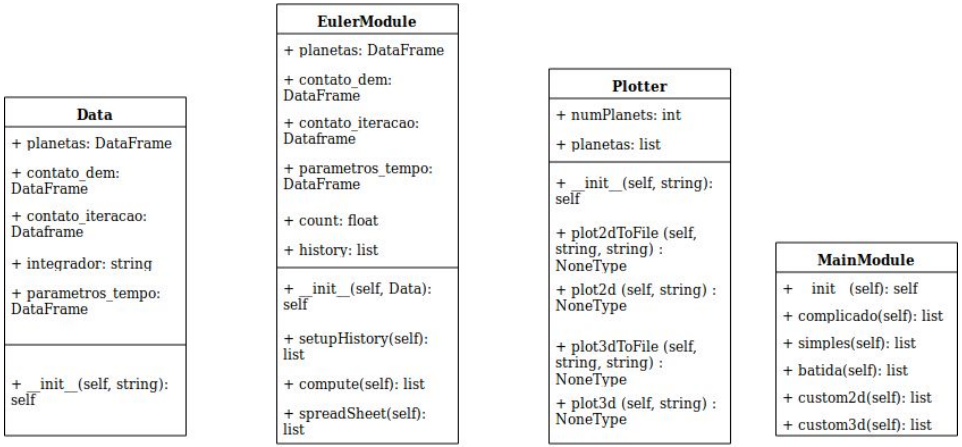
Ferramentas utilizadas:

- Dois módulos tiveram que ser usados com o pandas:
- openpyxl (2.6.2);
- xlrd (1.2.0);
- Usados para leitura e escrita de arquivos .xlsx (Excel)

Créditos da imagem: WWF

Estrutura:

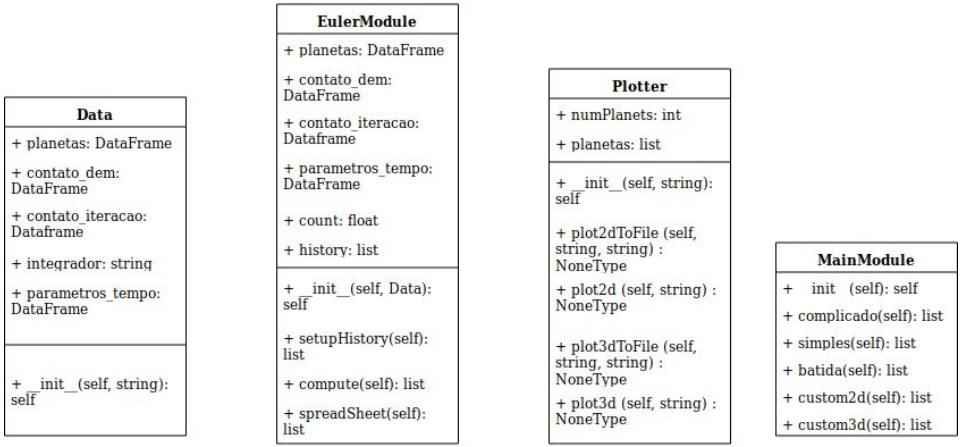
- Cada módulo foi feito em uma classe
- O classe de *Data* lê os dados
- A classe *EulerModule* recebe os dados de uma instância de Data, faz as contas necessárias, e retorna o histórico em uma planilha .xlsx
- A classe *Plotter* lê uma planilha gerada por *EulerModule* e plota as imagens (2d ou 3d)



Créditos da Imagem: o autor

Estrutura:

- Todas os módulos são apropriadamente instanciados no módulo *MainModule*, que não exerce função dentro do processo
  - Apenas “unifica” todo o processo dentro de poucos passos
  - Os métodos *complicado*, *simples* e *batida* são demonstrativos, mostram o algoritmo funcionando



Créditos da Imagem: o autor

Estrutura:

- Além dos três módulos básicos e do *MainModule*, existe um script chamado *support.py* que faz o cálculo das forças gravitacional e de contato entre dois corpos celestes;
- Serão usadas no método *compute()* do EulerModule

```python
from numpy.linalg import norm
import scipy.constants as con

def gravForce(coordA, coordB, mA, mB):
    """Calcula, vetorialmente, a força gravitacional que um corpo B exerce em um corpo A (ou o contrário, dependendo da interpretação).
    a função recebe as coordenadas e massas dos corpos desejados, e retorna uma força em sua forma vetorial (um array, obivamente)"""
    r = (coordA-coordB)
    #print(r)
    escalar = (con.G * mA * mB)/(norm(r)**3)
    return -1*(escalar * (r))

def springForce(coordA, coordB, radA, radB, k):
    """Essa função recebe as coordenadas dos corpos, seus raios, e uma constante elástica. Ela checa primeiro se existe um choque entre os corpos,
    e em seugida calcula, vetorialmente, a força de reação, como se o corpo estivesse sendo impulsionado por uma mola, que foi comprimida uma distancia
    deltaX igual ao quanto um dos corpos entrou no outro. Caso não exista choque, essa força é zero"""
    r = (coordA - coordB)
    if norm(r) < (radA + radB):
        deltaX = norm(r) - (radA+radB)
        return (k * deltaX) * (r/norm(r))
    else:
        return 0

def calculateForce(coordA, coordB, radA, radB, mA, mB, k):
    """Soma as forças gravitacional e de reação ao choque entre um corppo A e B"""
    return (gravForce(coordA, coordB, mA, mB)) - springForce(coordA, coordB, radA, radB, k)
```

Créditos da Imagem: o autor

Problemas

- O tempo de execução do código ficou muito alto;
- O método *compute()* da classe EulerModule ficou muito mal otimizado:
  - Muitos loops encadeados;
  - Acessos constantes à estruturas de dados de grande porte;
- Quanto maior o tempo limite e menor o *dt*, mais demora a execução.

Créditos da imagem: El País

A classe Data

- Procura os seccionamentos dentro do arquivo original;
  - Busca onde podem ser encontrados jogos da velha;
- Achadas as posições, faz a leitura, cada secção do arquivo original vira um Data Frame (exceto o integrador, porque é só uma palavra).

```python
import pandas as pd

class Data:
    """O módulo de leitura dos dados, recebe as informações num arquivo txt e as organiza em dataframes dos pandas
        para serem acessados e manipulados facilmente pelos outros módulos"""

    def __init__(self, filePath):
        """Aqui é onde ocorre a leitura dos dados a partir do arquivo fornecido pelo usuario, a função recebe os
        o caminho para o arquivo dentro do sistema e retorna um objeto com as informações do arquivo"""
        file = list(open(filePath).read().split("\n"))
        filesize = len(file)
        hashlist = list()

        for x in range(filesize):
            if "#" in file[x]:
                hashlist.append(x)


        #esse aqui lê o #PLANETAS
        self.planetas = pd.read_csv(filePath, skiprows = 2, nrows = hashlist[1]-3, sep = " ", header = None,
                                    names = ["id" ,"x", "y", "z", "v0x", "v0y", "v0z", "raio", "massa"])

        #esse aqui lê o #CONTATO_DEM
        self.contato_dem = pd.read_csv(filePath, skiprows = hashlist[1]+2, nrows = hashlist[2]-hashlist[1]-3, sep = " ", header = None,
                                       names = ["idModelo", "Kn"], index_col = ["idModelo"])
        #self.contato_dem.set_index("idModelo", inplace=True)

        #esse aqui lê o #CONTATO_ITERACAO
        self.contato_iteracao = pd.read_csv(filePath, skiprows = hashlist[2]+1, nrows = hashlist[3]-hashlist[2]-2, sep = " ", header = None)

        #aqui lê o #INTEGRADOR
        self.integrador = file[hashlist[3]+1]

        #aqui lê o PARAMETROS_TEMPO
        self.parametros_tempo = pd.read_csv(filePath, skiprows = hashlist[4]+1, nrows = hashlist[5]-hashlist[4]-2, sep = " ", header = None,
                                            names = ["dt", "tempo_final_simulacao", "numero_de_passos_impressao"])

```

Créditos da imagem: o autor

A Classe EulerModule

- *__init__* recebe os valores obtidos pela instância da classe *Data* fornecida;
- *setupHistory* monta uma lista de *DataFrames*, cada um representando o histórico de um planeta (velocidade e posição) ao longo do tempo determinado.

```python
import pandas as pd
import numpy as np
from support import calculateForce

class EulerModule:
    """Classe recebe os dados lidos pela classe Data, e faz todos os cálculos de posição e velocidade, a partir do método de Euler, podendo salvar os resultado

    def __init__(self, data):
        """Recebe uma instância da classe Data, com as informações retiradas do arquivo txt, e retorna um objeto da classe EulerModule, já feito os cálculos de
        para um sistema, na forma de uma lista de DataFrames do Pandas, cada DataFrame representando um corpo celeste"""
        self.planetas = data.planetas
        self.contato_dem = data.contato_dem
        self.contato_iteracao = data.contato_iteracao
        self.parametros_tempo = data.parametros_tempo

        self.count = (self.parametros_tempo.tempo_final_simulacao[0]/self.parametros_tempo.dt[0])
        self.history = self.compute()

    def setupHistory(self):
        """Monta a lista de DataFrames (cada DataFrame representa um corpo celeste), a ser preenchida pelo método Compute"""
        history = list()
        size = len(self.planetas)

        #aqui é preaprado o índice o dataframe
        index = list()
        aux = len(str(self.parametros_tempo.dt[0]))-2
        for x in range(0, int(self.count)+1):
            index.append(round(x*self.parametros_tempo.dt[0], aux))

        #aqui a lista de dataframes é montada preparado o dataframe é montado
        for x in range(size):
            history.append(pd.DataFrame(index = range(0, len(index)), columns = ["x", "y", "z", "v0x", "v0y", "v0z"])) #dataframe vazio
            history[x].iloc[0] = self.planetas.iloc[x][['x','y','z', 'v0x', 'v0y', 'v0z']] #recebe os valores iniciais
            history[x] = history[x].reindex(index) #dataframe recebe o novo index, ficando então um dataframe cheio de NaNs, a serem preenchidos pelo compute

        return history
```

Créditos da imagem: o autor

A Classe EulerModule

- O método *compute* faz todos os cálculos necessários para preencher a lista retornada pelo *setupHistory*.
- O elefante branco do projeto todo, com suas estruturas de repetição encadeadas, é facilmente o método que mais consome tempo.

```
45 def compute(self):
46     """Faz os calculos de força, aceleração em cada instante para cada planeta, e usando o método de euler, calcula a velocidade e posição para cada ponto no intervalo,
47     prencheendo os DataFrames de cada planeta"""
48     numPlanets = len(self.planetas)
49 
50     history = self.setupHistory()
51 
52     #"iteracoes"
53     for x in range (1, int(self.count)+1):
54         print("iteracao: ", x)
55         #calculos para cada planeta numa dada instancia
56         for m in range (numPlanets):
57             #calcula da força sofrida no planeta
58             force = np.zeros(3)
59             for n in range(numPlanets):
60                 if n != m:
61                     thisPlanet = np.array(history[m].iloc[x-1][["x","y","z"]])
62                     otherPlanet = np.array(history[n].iloc[x-1][["x","y","z"]])
63 
64                     force = force + calculateForce(thisPlanet, otherPlanet,
65                         self.planetas.raio[m], self.planetas.raio[n],
66                         self.planetas.massa[m], self.planetas.massa[n], self.contato_dem.loc[self.contato_iteracao[m][n]][0])
67 
68             #calcula acleração, velocidade, e posição
69             accel = force/self.planetas.massa[m]
70             history[m].iloc[x][["v0x", "v0y", "v0z"]] = history[m].iloc[x-1][["v0x", "v0y", "v0z"]].to_numpy() + (self.parametros_tempo.dt[0] * accel)
71             history[m].iloc[x][["x", "y", "z"]] = history[m].iloc[x-1][["x", "y", "z"]].to_numpy() + (self.parametros_tempo.dt[0] * history[m].iloc[x][["v0x", "v0y", "v0z"]].to
72 
73 
74     return history
```

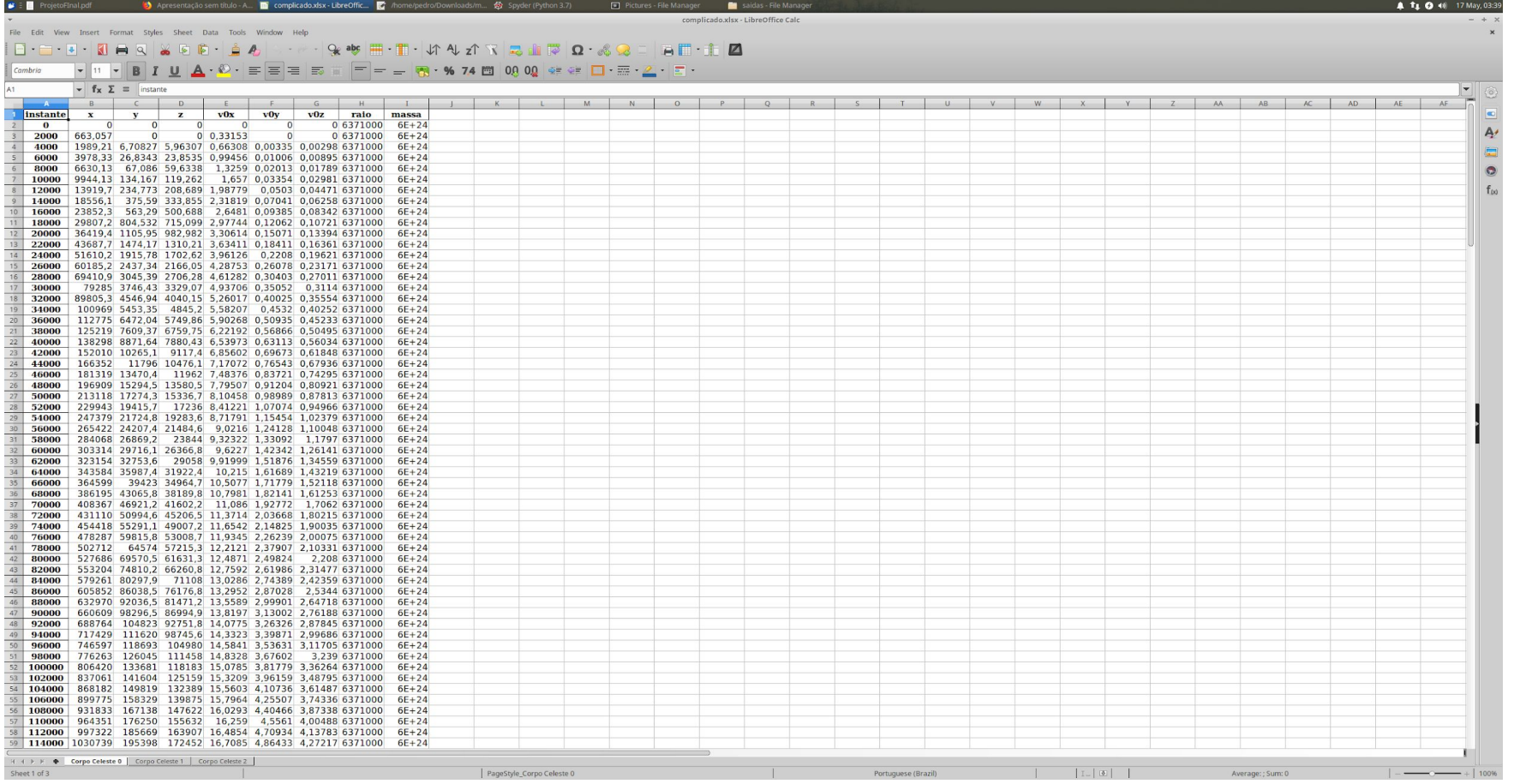
Créditos da imagem: o autor

A Classe EulerModule

- O método *spreadSheet* formata e entrega uma planilha .xlsx (Excel);
- Cada página da planilha representa um corpo celeste, na ordem em que eles estavam dispostos no documento original.

```
76 def spreadSheet(self, filePath):
77     """Adiciona massa e raio de cada planeta para seu respectivo DataFrame, e então joga todo em uma planilha .xlsx com o nome do e diretório desejado pelo usuário"""
78     size = len(self.history)
79 
80     writer = pd.ExcelWriter(filePath, engine = "openpyxl")
81 
82     #quantidade de passos a serem pulados
83     pacing = int((self.parametros_tempo.tempo_final_simulacao[0]/self.parametros_tempo.numero_de_passos_impressao[0])/self.parametros_tempo.dt[0])
84 
85     printHistory = list()
86     for x in range(size):
87         printHistory.append(pd.DataFrame(columns = ["x", "y", "z", "v0x", "v0y", "v0z"]))
88 
89         for y in range(0, int(self.count)+1, pacing):
90             printHistory[x] = printHistory[x].append(self.history[x].iloc[y])
91 
92         #adiciona mais dados e formata para uma exibição melhor no excel
93         printHistory[x]["raio"] = self.planetas.iloc[x].raio
94         printHistory[x]["massa"] = self.planetas.iloc[x].massa
95         printHistory[x].index.name = "instante"
96         name = "Corpo Celeste {}".format(int(self.planetas.iloc[x].id))
97         printHistory[x].to_excel(writer, sheet_name = name)
98 
99     writer.save()
00 
01     return printHistory
```

Créditos da imagem: o autor



instante	x	y	z	v0x	v0y	v0z	raio	massa
0	0	0	0	0	0	0	6371000	6E+24
2000	663,057	0	0	0,33153	0	0	6371000	6E+24
4000	1989,21	6,70827	5,96307	0,66308	0,00335	0,00298	6371000	6E+24
6000	3978,33	26,8343	23,8535	0,99456	0,01006	0,00895	6371000	6E+24
8000	6630,13	67,086	59,6338	1,3259	0,02013	0,01789	6371000	6E+24
10000	9944,13	134,167	119,262	1,657	0,03354	0,02981	6371000	6E+24
12000	13919,7	234,773	208,689	1,98779	0,0503	0,04471	6371000	6E+24
14000	18556,1	375,59	333,855	2,31819	0,07041	0,06258	6371000	6E+24
16000	23852,3	563,29	500,688	2,6481	0,09385	0,08342	6371000	6E+24
18000	29807,2	804,532	715,099	2,97744	0,12062	0,10721	6371000	6E+24
20000	36419,4	1105,95	982,982	3,30614	0,15071	0,13394	6371000	6E+24
22000	43687,7	1474,17	1310,21	3,63411	0,18411	0,16361	6371000	6E+24
24000	51610,2	1915,78	1702,62	3,96126	0,2208	0,19621	6371000	6E+24
26000	60185,2	2437,34	2166,05	4,28753	0,26078	0,23171	6371000	6E+24
28000	69410,9	3045,39	2706,28	4,61282	0,30403	0,27011	6371000	6E+24
30000	79285	3746,43	3329,07	4,93706	0,35052	0,3114	6371000	6E+24
32000	89805,3	4546,94	4040,15	5,26017	0,40025	0,35554	6371000	6E+24
34000	100969	5453,35	4845,2	5,58207	0,4532	0,40252	6371000	6E+24
36000	112775	6472,04	5749,86	5,90268	0,50935	0,45233	6371000	6E+24
38000	125219	7609,37	6759,75	6,22192	0,56866	0,50495	6371000	6E+24
40000	138298	8871,64	7880,43	6,53973	0,63113	0,56034	6371000	6E+24
42000	152010	10265,1	9117,4	6,85602	0,69673	0,61848	6371000	6E+24
44000	166352	11796	10476,1	7,17072	0,76543	0,67936	6371000	6E+24
46000	181319	13470,4	11962	7,48376	0,83721	0,74295	6371000	6E+24
48000	196909	15294,5	13580,5	7,79507	0,91204	0,80921	6371000	6E+24
50000	213118	17274,3	15336,7	8,10458	0,98989	0,87813	6371000	6E+24
52000	229943	19415,7	17236	8,41221	1,07074	0,94966	6371000	6E+24
54000	247379	21724,8	19283,6	8,71791	1,15454	1,02379	6371000	6E+24
56000	265422	24207,4	21484,6	9,0216	1,24128	1,10048	6371000	6E+24
58000	284068	26869,2	23844	9,32322	1,33092	1,1797	6371000	6E+24
60000	303314	29716,1	26366,8	9,6227	1,42342	1,26141	6371000	6E+24
62000	323154	32753,6	29058	9,91999	1,51876	1,34559	6371000	6E+24
64000	343584	35987,4	31922,4	10,215	1,61689	1,43219	6371000	6E+24
66000	364599	39423	34964,7	10,5077	1,71779	1,52118	6371000	6E+24
68000	386195	43065,8	38189,8	10,7981	1,82141	1,61253	6371000	6E+24
70000	408367	46921,2	41602,2	11,086	1,92772	1,7062	6371000	6E+24
72000	431110	50994,6	45206,5	11,3714	2,03668	1,80215	6371000	6E+24
74000	454418	55291,1	49007,2	11,6542	2,14825	1,90035	6371000	6E+24
76000	478287	59815,8	53008,7	11,9345	2,26239	2,00075	6371000	6E+24
78000	502712	64574	57215,3	12,2121	2,37907	2,10331	6371000	6E+24
80000	527686	69570,5	61631,3	12,4871	2,49824	2,208	6371000	6E+24
82000	553204	74810,2	66260,8	12,7592	2,61986	2,31477	6371000	6E+24
84000	579261	80297,9	71108	13,0286	2,74389	2,42359	6371000	6E+24
86000	605852	86038,5	76176,8	13,2952	2,87028	2,5344	6371000	6E+24
88000	632970	92036,5	81471,2	13,5589	2,99901	2,64718	6371000	6E+24
90000	660609	98296,5	86994,9	13,8197	3,13002	2,76188	6371000	6E+24
92000	688764	104823	92751,8	14,0775	3,26326	2,87845	6371000	6E+24
94000	717429	111620	98745,6	14,3323	3,39871	2,99686	6371000	6E+24
96000	746597	118693	104980	14,5841	3,53631	3,11705	6371000	6E+24
98000	776263	126045	111458	14,8328	3,67602	3,239	6371000	6E+24
100000	806420	133681	118183	15,0785	3,81779	3,36264	6371000	6E+24
102000	837061	141604	125159	15,3209	3,96159	3,48795	6371000	6E+24
104000	868182	149819	132389	15,5603	4,10736	3,61487	6371000	6E+24
106000	899775	158329	139875	15,7964	4,25507	3,74336	6371000	6E+24
108000	931833	167138	147622	16,0293	4,40466	3,87338	6371000	6E+24
110000	964351	176250	155632	16,259	4,5561	4,00488	6371000	6E+24
112000	997322	185669	163907	16,4854	4,70934	4,13783	6371000	6E+24
114000	1030739	195398	172452	16,7085	4,86433	4,27217	6371000	6E+24

Planilha .xlsx representando o movimento de três corpos celestes (créditos da imagem: o autor)

A classe Plotter

```python
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D #vem com o matplotlib

class Plotter:
   """Módulo de visualização do projeto, ao ser instanciado, faz a leitura de um arquivo gerado pelo módolo de análise para então realizar a plotagem"""
   def __init__(self, filePath):
        """Faz a leitura dos fornecidos pela planilha criada pelo módulo de análise"""
        xlsx = pd.ExcelFile(filePath, engine = "xlrd")
        sheetList = xlsx.sheet_names
        self.NumPlanets = len(sheetList)
        self.planetas = list()
        [self.planetas.append(pd.read_excel(xlsx, sheetList[x])) for x in range (self.NumPlanets)]
```

*_init_* lê a planilha página à página e salva os dados em um lista de *DataFrames* (créditos da imagem: o autor)

A classe Plotter

- O método *plot2d* separa os pontos de início e fim da trajetória, as plotando como pontos, e a trajetória entre eles como uma linha semi-transparente
- Também existe plot2dToFile, que faz exatamente a mesma coisa, exceto que ao invés de abrir uma janela, ela salva direto no armazenamento do computador.

```
def plot2d(self, colors):
    """Faz a plotagem 2d e mostra o gráfico numa janela (caso seja rodado direto no iPython). Recebe uma
    lista com as cores desejadas pelo usuário para cada corpo celeste."""
    ax = plt.gca()

    for x in range (self.NumPlanets):
        firstPos = self.planetas[x].head(1)
        lastPos = self.planetas[x].tail(1)

        trajectory = self.planetas[x].drop(self.planetas[x].head(1).index)
        trajectory = trajectory.drop(trajectory.tail(1).index)


        firstPos.plot(kind = "scatter", x = "x", y="y", color = colors[x], alpha = 0.5, ax=ax, zorder = 2)
        trajectory.plot(kind = "line", x = "x", y="y", color = "black", alpha = 0.25, ax=ax, legend = False, zorder = 1)
        lastPos.plot(kind = "scatter", x = "x", y="y", color = colors[x], ax=ax, zorder = 3)

    plt.show()
```

Créditos da imagem: o autor

A classe Plotter

- O método *plot3d* faz a mesma coisa que o 2d, exceto que para as três dimensões
- Também existe uma *plot3dToFile*, análoga à *plot2dToFile*

```python
def plot3d(self, colors):
    """Faz a plotagem 3d e mostra o gráfico numa janela (caso seja rodado direto no iPython). Recebe uma
    lista com as cores desejadas pelo usuário para cada corpo celeste."""
    fig = plt.figure()
    ax3d = fig.add_subplot(111, projection = "3d")


    for x in range (self.NumPlanets):
        firstPos = self.planetas[x].head(1)
        lastPos = self.planetas[x].tail(1)

        trajectory = self.planetas[x].drop(self.planetas[x].head(1).index)
        trajectory = trajectory.drop(trajectory.tail(1).index)

        ax3d.scatter(firstPos["x"], firstPos["y"], firstPos["z"], color = colors[x], alpha = 0.5)
        ax3d.plot(trajectory["x"], trajectory["y"], trajectory["z"], color = "black", alpha = 0.25)
        ax3d.scatter(lastPos["x"], lastPos["y"], lastPos["z"], color = colors[x])

    plt.show()
```

Créditos da imagem: o autor

A classe MainModule

- Essa classe apenas “condensa” todas as outras, tornando fácil o uso do programa.
- *batida, simples* e *complicado* usam dados já presentes no diretório, com caráter demonstrativo

```
"""
from readingModule import Data
from plottingModule import Plotter
from analysisModule import EulerModule

class MainModule():
    """Esta classe condensa os modulos, de forma a tornar os testes mais convenientes."""
    def __init__(self):
        pass

    def complicado(self):
        """Roda o teste demonstrativo de três corpos celestes movendo-se no espaço. O método
        retorna instâncias do módulo de leitura, de análise e de visualização já carregados com os dados."""
        dadosTeste = Data("entradas/complicado.txt")
        print("dados lidos")

        eulerTeste = EulerModule(dadosTeste)
        print("calculos feitos")

        eulerTeste.spreadSheet("saidas/complicado.xlsx")
        print("planilha salva")

        plotagem = Plotter("saidas/complicado.xlsx")
        print("planilha lida")

        #plotagem.plot3dToFile(["red", "green", "blue"], "saidas/complicado.pdf")
        plotagem.plot3d(["red", "blue", "green"])
        print("plotagem feita")

        return dadosTeste, eulerTeste, plotagem

    def simples(self):
        """Roda o teste demonstrativo de dois corpos celestes movendo-se no plano. O método
        retorna instâncias do módulo de leitura, de análise e de visualização já carregados com os dados."""
        dadosTeste = Data("entradas/simples.txt")
        print("dados lidos")

        eulerTeste = EulerModule(dadosTeste)
        print("calculos feitos")

        eulerTeste.spreadSheet("saidas/simples.xlsx")
        print("planilha salva")

        plotagem = Plotter("saidas/simples.xlsx")
        print("planilha lida")

        #plotagem.plot2dToFile(["red", "blue"], "saidas/simples.pdf")
        plotagem.plot2d(["red", "blue"])
        print("plotagem feita")

        return dadosTeste, eulerTeste, plotagem

    def batida(self):
        """Roda o teste demonstrativo de dois corpos celestes chocando-se no plano. O método
        retorna instâncias do módulo de leitura, de análise e de visualização já carregados com os dados."""
        dadosTeste = Data("entradas/batida.txt")
        print("dados lidos")

        eulerTeste = EulerModule(dadosTeste)
        print("calculos feitos")

        eulerTeste.spreadSheet("saidas/batida.xlsx")
        print("planilha salva")

        plotagem = Plotter("saidas/batida.xlsx")
        print("planilha lida")

        #plotagem.plot2dToFile(["red", "blue"], "saidas/batida.pdf")
        plotagem.plot2d(["red", "blue"])
        print("plotagem feita")

        return dadosTeste, eulerTeste, plotagem
```

Créditos da imagem: o autor

```
def custom2dplot(self, txtPath, xlsxPath, colors, imagePath):
    """Teste customizado, recebe o nome do arquivo .txt a ser lido, o nome da planilha .xlsx a ser testada, uma lista com as cores desejadas pelo usuário
    e o nome do imagem a ser salva em memória secundária. Ele faz a checagem nas entradas do .txt e do .xlsx, levantando exceções caso não funcionem. O método
    retorna instâncias do módulo de leitura, de análise e de visualização já carregados com os dados. Faz a plotagem dos dados em 2d (plano x0y)."""

    if ".txt" not in txtPath:
        raise Exception("Formato errado, tem que ser .txt!")
    elif ".xlsx" not in xlsxPath:
        raise Exception("Formato errado, tem que ser .txt!")
    else:
        dadosTeste = Data(txtPath)
        print("dados lidos")

        eulerTeste = EulerModule(dadosTeste)
        print("calculos feitos")

        eulerTeste.spreadSheet(xlsxPath)
        print("planilha salva")

        plotagem = Plotter(xlsxPath)
        print("planilha lida")

        plotagem.plot2dToFile(colors, imagePath)
        plotagem.plot2d(colors)
        print("plotagem feita")

        return dadosTeste, eulerTeste, plotagem

def custom3dplot(self, txtPath, xlsxPath, colors, imagePath):
    """Teste customizado, recebe o nome do arquivo .txt a ser lido, o nome da planilha .xlsx a ser testada, uma lista com as cores desejadas pelo usuário
    e o nome do imagem a ser salva em memória secundária. Ele faz a checagem nas entradas do .txt e do .xlsx, levantando exceções caso não funcionem. O método
    retorna instâncias do módulo de leitura, de análise e de visualização já carregados com os dados. Faz a plotagem dos dados em 3d."""

    if ".txt" not in txtPath:
        raise Exception("Formato errado, tem que ser .txt!")
    elif ".xlsx" not in xlsxPath:
        raise Exception("Formato errado, tem que ser .txt!")
    else:
        dadosTeste = Data(txtPath)
        print("dados lidos")

        eulerTeste = EulerModule(dadosTeste)
        print("calculos feitos")

        eulerTeste.spreadSheet(xlsxPath)
        print("planilha salva")

        plotagem = Plotter(xlsxPath)
        print("planilha lida")

        #plotagem.plot3dToFile(colors, imagePath)
        plotagem.plot3d(colors)
        print("plotagem feita")

        return dadosTeste, eulerTeste, plotagem
```

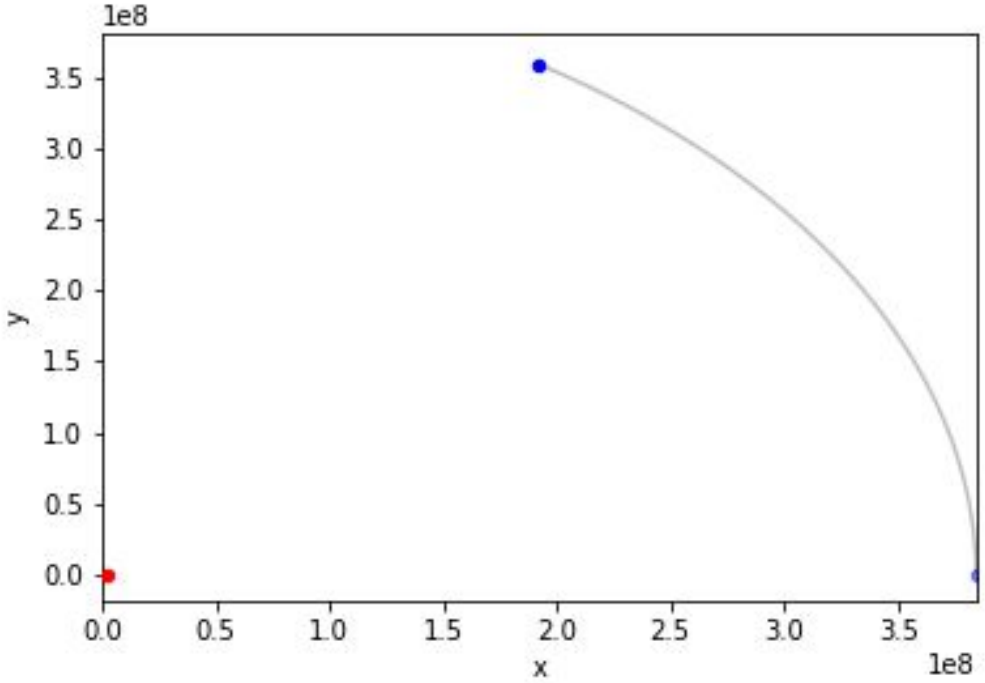
MainModule também aceita dados próprios do usuário (créditos da imagem: o autor)

```
#PLANETAS
2
0 0 0 0 0 0 0 6371000 5.97e24
1 384400000 0 0 0 1080 0 1737100 7.34e22

#CONTATO_DEM
1
1 1e7

#CONTATO_INTERACAO
1 1
1 1

#INTEGRADOR
euler

#PARAMETROS TEMPO
2000 400000 200

#FIM
```

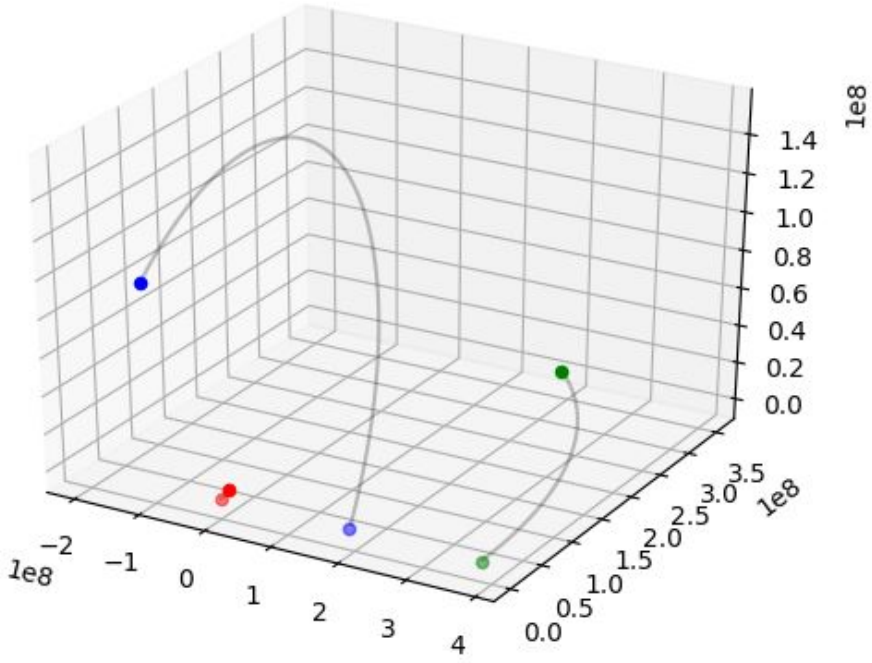
```
#PLANETAS
3
0 0 0 0 0 0 0 6371000 5.97e24
1 384400000 0 0 0 1080 0 1737100 7.34e22
2 192200000.0 0 0 0 1080 1080 1737100 7.34e22

#CONTATO_DEM
1
1 1e7

#CONTATO_INTERACAO
1 1 1
1 1 1
1 1 1

#INTEGRADOR
euler

#PARAMETROS TEMPO
2000 400000 200

#FIM
```

```
#PLANETAS
2
0 0 0 0 0 0 0 6371000 5.97e24
1 3845 0 0 0 0 0 6371000 5.97e24

#CONTATO_DEM
1
1 1e7

#CONTATO_INTERACAO
1 1
1 1

#INTEGRADOR
euler

#PARAMETROS TEMPO
2000 400000 200

#FIM
```

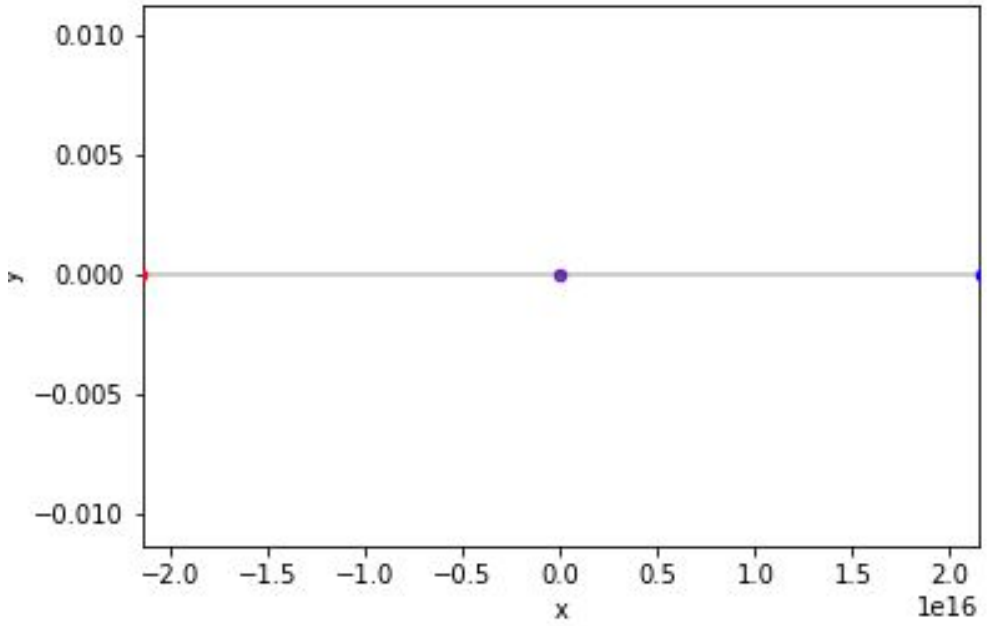
Da esuqerda para a direita: *simples.txt*, *complicado.txt e batida.txt* (créditos da imagem: o autor)



Caso Simples (créditos da imagem: o autor)



Caso Complicado (créditos da imagem: o autor)



Caso de Choque (créditos da imagem: o autor)

Conclusão: o algoritmo mais simples nem sempre é o mais eficiente.