O sistema Sol-Terra-Lua: Uma breve simulação numérica usando Python

The Sun-Earth-Moon system: A brief numerical simulation using Python

Carlos Eduardo Rufino da Silva
Universidade Federal de Campina Grande - UFCG, Campina Grande, PB
carloseduardorufino7@gmail.com

Rômulo Rodrigues da Silva
Universidade Federal de Campina Grande - UFCG, Campina Grande, PB
romulo@df.ufcg.edu.br

Leonardo Augusto de Lemos Batista
Centro Universitário Tabosa de Almeida - ASCES-UNITA, Caruaru, PE
leoaugusto31@gmail.com

Resumo: Este trabalho tem como objetivo incentivar alunos e professores de graduação, principalmente aqueles que lidam com a chamada área das exatas, a utilizarem a computação numérica como ferramenta de apoio. Para tanto, o problema de três corpos (a saber, Sol, Terra e Lua) foi tomado como motivação. Sob esse foi feita uma implementação em Python, uma vez que esta é uma linguagem de fácil compreensão e acesso, tal como as análises necessárias para o entendimento físico do problema. Assim, este texto pretende através da computação numérica tornar alguns resultados mais claros e objetivos para os estudantes principalmente.

Palavras-chave: simulação numérica; python; odespy; problema de três corpos.

Abstract: This work aims to encourage students and graduation teachers, especially those who deal with the so-called area of the exacts, to use numerical computing as a support tool. For this, the problem of three bodies, under which was made the implementation in Python, was taken as motivation. Thus, this text intends through numerical computation to make some results clearer and more objective for students mainly.

Key words: numerical simulation; python; odespy; problem of three bodies.

1 Introdução

Desde a antiguidade os céus têm sido causa de grande admiração para o ser humano. Ao longo da história muitos empreenderam, com devido sucesso, entender e descrever a trajetória dos corpos celestes. Pode-se entre esses trabalhos destacar Ptolomeu com seus epiciclos, Nicolau Copérnico, Tycho Brahe, Kepler [1], Galileu Galilei com suas observações pioneiras [2], Isaac Newton e tantos outros.

Recebido em 02/01/2018 - Aceito em 13/08/2018

Durante muito tempo esses trabalhos se deram de forma muito laboriosa e demorada, contudo, como consequência do desenvolvimento de computadores com alta capacidade de processamento, houve um grande avanço nas pesquisas relacionadas. Vários softwares que têm por objetivo computar a trajetória dos corpos celestes surgiram. Dentre eles estão o Stellarium [3] e o Celestia [4], que além de serem planetários de código aberto, permite a visualização celeste realista tridimensional correspondente a uma data específica ou em tempo real. Esses softwares também se mostram muito versáteis pelo fato de serem compatíveis com as principais plataformas (Linux, Mac OS X e Windows).

Normalmente, os alunos têm seu primeiro contato com as equações que descrevem a dinâmica celeste em disciplinas introdutórias de Física, como Física Geral I por exemplo, e mais formalmente em cursos avançados como os de Mecânica Clássica. No entanto, geralmente, esses estudos são de natureza puramente analítica, uma vez que, para sistemas com mais de dois corpos se torna uma tarefa essencialmente impossível achar soluções explícitas para o movimento dos mesmos [5]. Dessa forma, fica ao critério dos professores apresentarem para seus alunos uma abordagem mais quantitativa.

Tendo em vista tudo isso, por vezes, a compreensão da matéria acaba por ficar deficitária aos alunos por uma simples falta de visualização gráfica dos resultados que a teoria (equações) apresenta. Para ajudar a resolver esse problema alguns autores sugerem o uso das ferramentas computacionais [6, 7, 8]. Nesse sentido, este texto difere por propor não apenas o uso, mas incentivar a implementação do software de computação numérica. Isso abrirá novas fronteiras para os estudantes, que poderão usá-lo para solução de outros sistemas físicos.

Para tanto, atentando que nem todos estão familiarizados com a programação, dentre as várias linguagens que existem atualmente, o uso de uma linguagem prática e de fácil entendimento quando comparada ao C por exemplo [9], como o Python, achou-se oportuna neste texto. Além de sua simplicidade, pode-se ainda lembrar da facilidade de instalar em um computador qualquer um ambiente de trabalho para o mesmo que, por ser um software de licença livre, se mostra muito atraente para uso pessoal.

2 Modelagem (Clássica) do problema

Dentre as contribuições que Newton deixou a humanidade, duas das mais conhecidas são sua segunda lei do movimento e sua lei da gravitação universal, a primeira pode ser enunciada como segue:

> *"A mudança de movimento é proporcional a força motora imprimida, e é produzida na direção da linha reta na qual aquela força é imprimida."* [10].

ou ainda, na forma mais usual, essa lei pode ser escrita como,

$$\overrightarrow{F} = \frac{d}{dt}(m \cdot \overrightarrow{v})$$

como aqui não será contemplado o caso em que a massa depende do tempo, temos que,

$$\overrightarrow{F} = m \cdot \frac{d}{dt}(\overrightarrow{v}) = m \cdot \overrightarrow{a} \tag{1}$$

A segunda, a Lei da gravitação universal diz que:

"Cada partícula de matéria atrai qualquer outra partícula com uma força variando diretamente como o produto de suas massas gravitacionais e inversamente como o quadrado da distância entre elas." [10]

por sua vez, essa lei pode ser escrita como,

$$F = G \cdot \frac{Mm}{r^2}$$

ou em sua forma vetorial,

$$\overrightarrow{F}_{21} = -G\frac{m_{g1}m_{g2}}{r^2}\hat{e} = -G\frac{m_{g1}m_{g2}}{r^3} \cdot \overrightarrow{r} \tag{2}$$

Nesta equação, F_{21} denota a força exercida pela massa gravitacional m_{g2} sobre a massa gravitacional m_{g1}, G é a constante de gravitação universal, $\overrightarrow{r}$ é o vetor que aponta de 2 para 1 e r é a distância entre os corpos pontuais.

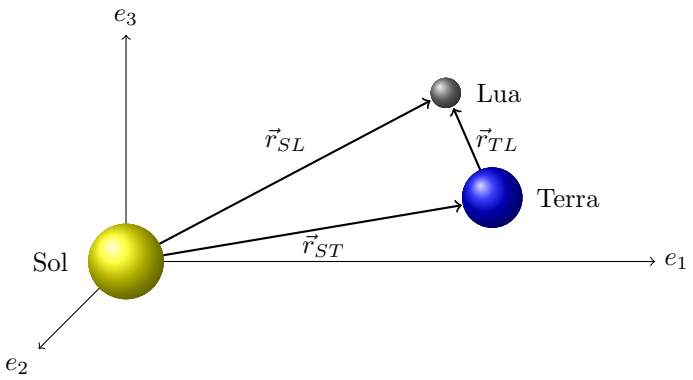

Figura 1. Diagrama de ilustração dos vetores posição.

No sistema internacional (SI), a constante de gravitação universal, G, assume o seguinte valor:

$$G = 6,673 \times 10^{-11}\,\frac{m^3}{Kg \cdot s^2}$$

Por motivos de simplificação, uma vez que a massa do Sol é muito maior que a massa da Terra e da Lua, o Sol será aproximado como o centro de massa do sistema e portanto como um referencial inercial [11] (é importante salientar que dessa forma a modelagem se reduz ao bem conhecido, dos cursos de mecânica clássica, problema de dois corpos [12], o qual pode ser solucionado usando-se uma abordagem um pouco diferente daquela que este texto adota.) como na Figura 1 e os demais corpos serão tratados como pontuais. Assim, usando-se os subíndices (S, T, L) para se referir ao Sol, à Terra e à Lua respectivamente, pode-se, segundo a Eq. (2), escrever as forças agindo sobre cada corpo como:

Força exercida pelo Sol sobre a Terra:

$$\overrightarrow{F}_{ST} = -G\frac{m_T M_S}{r_{ST}^3}\overrightarrow{r}_{ST} \tag{3}$$

Força exercida pelo Sol sobre a Lua:

$$\overrightarrow{F}_{SL} = -G\frac{m_L M_S}{r_{SL}^3}\overrightarrow{r}_{SL} \tag{4}$$

Força exercida pela Terra sobre a Lua:

$$\overrightarrow{F}_{TL} = -G\frac{m_T m_L}{r_{TL}^3}\overrightarrow{r}_{TL} \tag{5}$$

Força exercida pela Lua sobre a Terra:

$$\overrightarrow{F}_{LT} = -G\frac{m_T m_L}{r_{LT}^3}\overrightarrow{r}_{LT} = -\overrightarrow{F}_{TL} \tag{6}$$

Comparando as equações anteriores com a Eq. (1) e aplicando o princípio de superposição obtém-se o seguinte sistema:

$$\begin{cases} m_T \cdot \overrightarrow{a}_T = -G\frac{m_T M_S}{r_{ST}^3}\overrightarrow{r}_{ST} - G\frac{m_T m_L}{r_{LT}^3}\overrightarrow{r}_{LT} \\ m_L \cdot \overrightarrow{a}_L = -G\frac{m_L M_S}{r_{SL}^3}\overrightarrow{r}_{SL} - G\frac{m_L m_T}{r_{TL}^3}\overrightarrow{r}_{TL} \end{cases}$$

ou ainda,

$$\begin{cases} \frac{d^2}{dt^2}\overrightarrow{r}_T = GM_S\left\{-\frac{\overrightarrow{r}_{ST}}{r_{ST}^3} - \frac{m_L}{M_S}\frac{\overrightarrow{r}_{LT}}{r_{LT}^3}\right\} \\ \frac{d^2}{dt^2}\overrightarrow{r}_L = GM_S\left\{-\frac{\overrightarrow{r}_{SL}}{r_{SL}^3} - \frac{m_T}{M_S}\frac{\overrightarrow{r}_{TL}}{r_{TL}^3}\right\} \end{cases} \tag{7}$$

onde o vetor $\overrightarrow{r}_i$ denota um vetor tridimensional, ou seja,

$$\overrightarrow{r}_i = \sum_{j=1}^{3} x_{ij} \cdot \hat{e}_j \tag{8}$$

Por motivos que ficarão claros mais adiante, serão introduzidas duas novas definições no texto, a saber

$$\begin{cases} x_{ij} \equiv Ru_{ij} \\ t \equiv t' \cdot \tilde{t} \end{cases}$$

onde R, escolhido arbitrariamente, é o raio médio entre a Terra e o Sol, u_{ij} são as novas coordenadas computacionais (entenda-se adimensionais), t' é constante e $\tilde{t}$ é o tempo computacional.

Usando as definições anteriores e a notação com somatório, as Eqs. (7) podem ser reescritas compactamente na forma,

$$\frac{1}{t'^2}\frac{d^2}{d\tilde{t}^2}\sum_{j=1}^{3} u_{ij} \cdot \hat{e}_j = \frac{GM_S}{R^3}\sum_{k\neq i}\sum_{j=i}^{3}\left\{-\frac{M_k}{M_S}\frac{u_{ij} - u_{kj}}{(\sum_{l=1}^{3}(u_{il} - u_{kl})^2)^{\frac{3}{2}}} \cdot \hat{e}_j\right\}, k = S, T, L. \tag{9}$$

nesta equação o índice k é o conjunto de todos os planetas do sistema em questão (S: Sol, T: Terra, L: Lua) e o índice i é o conjunto de planetas sobre os quais as forças são aplicadas,

portanto, $i = \{T, L\}$, nesse caso o Sol não faz parte do conjunto, pois ele foi escolhido como referencial inercial, ou seja, o sistema de coordenadas tem origem em seu centro de massa.

As definições anteriores tornaram possível encontrar um conjunto de equações com índices computacionais, ou seja, com parâmetros normalizados e adimensionais para se evitar trabalhar com números grandes quando da simulação. Para garantir que toda a equação esteja normalizada, t' será escolhido de tal forma que a constante que multiplica a equação seja identicamente igual a 1, isso implica que,

$$t' = \sqrt{\frac{R^3}{GM_S}} \tag{10}$$

portanto, a Eq. (9) assume a forma,

$$\frac{d^2}{d\tilde{t}^2}\sum_{j=1}^{3} u_{ij}\cdot\hat{e}_j = \sum_{k\neq i}\sum_{j=1}^{3}\left\{-\frac{M_k}{M_S}\frac{(u_{ij}-u_{kj})}{(\sum_{l=1}^{3}(u_{il}-u_{kl})^2)^{\frac{3}{2}}}\cdot\hat{e}_j\right\}, k = S, T, L. \tag{11}$$

Definindo-se,

$$\begin{cases} x_{ij0} \equiv u_{ij} \\ x_{ij1} \equiv \frac{d}{dt}u_{ij} \end{cases} \tag{12}$$

e substituindo-se na Eq. (11), obtém-se

$$\begin{cases} \frac{d}{d\tilde{t}}\sum_{j=1}^{3} x_{ij0}\cdot\hat{e}_j = \sum_{j=1}^{3} x_{ij1}\cdot\hat{e}_j \\ \frac{d}{d\tilde{t}}\sum_{j=1}^{3} x_{ij1}\cdot\hat{e}_j = \sum_{k\neq 1}\sum_{j=1}^{3}\left\{-\frac{M_k}{M_S}\frac{(x_{ij0}-x_{kj0})}{(\sum_{l=1}^{3}(x_{il0}-x_{kl0})^2)^{\frac{3}{2}}}\cdot\hat{e}_j\right\}, k = S, T, L. \end{cases} \tag{13}$$

Somando-se a Eq. (13) em i tem-se finalmente o sistema de equações diferenciais de primeira ordem que descreve a dinâmica dos corpos em questão,

$$\begin{cases} \frac{d}{d\tilde{t}}\sum_{j=1}^{3} x_{Tj0}\cdot\hat{e}_j = \sum_{j=1}^{3} x_{Tj1}\cdot\hat{e}_j \\ \frac{d}{d\tilde{t}}\sum_{j=1}^{3} x_{Lj0}\cdot\hat{e}_j = \sum_{j=1}^{3} x_{Lj1}\cdot\hat{e}_j \\ \frac{d}{d\tilde{t}}\sum_{j=1}^{3} x_{Tj1}\cdot\hat{e}_j = \sum_{k\neq t}\sum_{j=1}^{3}\left\{-\frac{M_k}{M_S}\frac{(x_{Tj0}-x_{kj0})}{(\sum_{l=1}^{3}(x_{Tl0}-x_{kl0})^2)^{\frac{3}{2}}}\cdot\hat{e}_j\right\}, k = S, T, L. \\ \frac{d}{d\tilde{t}}\sum_{j=1}^{3} x_{Lj1}\cdot\hat{e}_j = \sum_{k\neq L}\sum_{j=1}^{3}\left\{-\frac{M_k}{M_S}\frac{(x_{Lj0}-x_{kj0})}{(\sum_{l=1}^{3}(x_{Ll0}-x_{kl0})^2)^{\frac{3}{2}}}\cdot\hat{e}_j\right\} \end{cases} \tag{14}$$

3 Implementação do Código em Python

O Python é uma linguagem de programação de alto nível criada no início dos idos anos 90 por Guido van Rossum [13], no Stichting Mathematisch Centrum (CIT) na Holanda, como um sucessor de uma linguagem chamada ABC. É uma linguagem interpretada, de script, orientada à objetos, indentada e de tipagem dinâmica. Atualmente possui um modelo de desenvolvimento comunitário, aberto e gerenciado pela organização sem fins lucrativos Python Software Foundation[1].

A linguagem conta com estruturas de alto nível tais como listas, dicionários, data, hora e muitas outras. Também possui uma vasta coleção de módulos prontos para uso, além de frameworks de terceiros que podem ser adicionados posteriormente. Ela possui recursos

[1] https://www.python.org/psf/

como geradores, metaclasses, introspecção, persistência e unidades de teste, os quais também estão presentes em outras linguagens. A linguagem é interpretada através do bytecode pela máquina virtual Python, isso possibilita a portabilidade do código [14].

Feitas essas considerações, a implementação foi feita de maneira didática não se importando com a elegância do algorítimo. Três pacotes foram importados no código:

- Matplotlib: Esse pacote (ou biblioteca) oferece ferramentas para geração de gráficos.
- Toolkits: Consiste de uma coleção de ferramentas específicas que estendem a biblioteca Matplotlib
- Numpy: Esse pacote provê ferramentas para vários tipos de processamento numérico.
- Odespy: Consiste em um pacote de ferramentas para resolver equações diferencias ordinárias (EDOs), pode ser obtido na Ref. [15]. Um abrangente leque de opções é oferecido ao usuário, dentre os métodos nela presentes estão, o Adams-Bashforth-Moulton explícito de segunda ordem, o Adams-Bashforth explícito de quarta ordem, o explícito de Heun, o explícito Runge-Kutta de quarta ordem (Este já bem conhecido, foi escolhido para a integração numérica deste trabalho) entre muitos outros.

```
# -*- coding: utf-8 -*-

import odespy, numpy
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Dados (todos os dados estão em unidades do S.I. e foram obtidos
# no site nssdc.gsfc.nasa.gov/planetary/factsheet/):

G = 6.673e-11       # Constante Newtoniana da gravitação universal

Msol = 1.98892e30   # Massa do Sol
Mt   = 5.9724e24    # Massa da Terra
Mlua = 0.07346e24   # Massa da Lua

R = 1.4709e11       # Raio da Terra no periélio
V = 30.29e3         # Velocidade da Terra no periélio
Rm= 1.496e11        # Raio médio Sol-Terra
Vm= 29.78e3         # Velocidade média na trajetória

R_lua = R + 0.3633e9    # Raio Sol-Lua no perigeu
V_lua = 1.076e3 + V     # Velocidade da Lua no perigeu
Rm_lua= Rm + 0.3633e9   # Raio médio (Sol-Lua)
Vm_lua= 1.02207e3       # Velocidade média da trajetória

# Normalização dos parâmetros

Rt = R/Rm           # Raio Sol-Terra
Vt = V/Vm           # Velocidade da Terra

Rlua = R_lua/Rm     # Raio Sol-Lua
Vlua = V_lua/Vm     # Velocidade da Lua

##########################################################
```

```
35 # ================================================== #
36
37 Ano     = 1.0                           # Ano
38 Ano_s   = 31536000                      # 1 ano em segundos
39 T_met   = numpy.sqrt(Rm**3/(G*Msol)) # Tempo métrico
40 T_comp = Ano_s/T_met                   # Tempo computacional/normalizado
41 To      = 0                             # Tempo inicial
42 Tf      = Ano * T_comp                  # Tempo final
43 dt      = 1000                          # intervalo dt em segundos
44
45 # ================================================== #
46 ############################################################
47
48 N = int(Ano * Ano_s/dt) # Número de pontos
49
50 # Condições iniciais
51 def condicaoInicial():
52     '''
53     Esta função retorna as condições iniciais do sistema
54     '''
55     # Terra
56     x0 = Rt      # x(0) da Terra
57     vx0= 0.0     # vx(0) da Terra
58     y0 = 0.0     # y(0) da Terra
59     vy0= Vt      # vy(0) da Terra
60     z0 = 0.0     # z(0) da Terra
61     vz0= 0.0     # vy(0) da Terra
62
63     # Lua
64     x0l = Rlua     # x(0) da Lua
65     vx0l= 0.0      # vx(0) da Lua
66     y0l = 0.0      # y(0) da Lua
67     vy0l= Vlua     # vy(0) da Lua
68     z0l = 0.0      # z(0) da Lua
69     vz0l= 0.0033   # vz(0) da Lua
70
71     return [x0,vx0,y0,vy0,z0,vz0,x0l,vx0l,y0l,vy0l,z0l,vz0l]
72
73 # ================================================== #
74 # Termos constantes das equações
75
76 C1 = Mlua/Msol  #Const. das equações da Lua
77 C2 = Mt/Msol    #Const. das equações da Terra
78
79 # Equações de movimento
80 def f(u,t):
81     '''
82     Esta função retorna as equações que serão computadas, as quais
83     tem a forma:
84
85             d
86           ---- x(t) = f(t).
87             dt
88
89     Note o leitor que por motivos didáticos as variáveis foram
```

```
90      renomeadas intuitivamente para facilitar a implementação das
91      equações.
92      '''
93      xt = u[0] ; xt1= u[1] # x_T00 ; x_T01
94      yt = u[2] ; yt1= u[3] # x_T10 ; x_T11
95      zt = u[4] ; zt1= u[5] # x_T20 ; x_T21
96
97      xl = u[6] ; xl1= u[7] # x_L00 ; x_L01
98      yl = u[8] ; yl1= u[9] # x_L10 ; x_L11
99      zl = u[10] ; zl1= u[11] # x_L20 ; x_L21
100
101     # Equações de movimento da Terra
102     Eq0 = xt1
103     Eq1 = -xt/(xt**2 + yt**2 +zt**2)**(3./2) \
104           - C1*(xt-xl)/((xt-xl)**2 + (yt-yl)**2 + (zt-zl)**2)**(3./2)
105     Eq2 = yt1
106     Eq3 = -yt/(xt**2 + yt**2 +zt**2)**(3./2) \
107           - C1*(yt-yl)/((xt-xl)**2 + (yt-yl)**2 + (zt-zl)**2)**(3./2)
108     Eq4 = zt1
109     Eq5 = -zt/(xt**2 + yt**2 +zt**2)**(3./2) \
110           - C1*(zt-zl)/((xt-xl)**2 + (yt-yl)**2 + (zt-zl)**2)**(3./2)
111
112     # Equações de movimento da Lua
113     Eq6 = xl1
114     Eq7 = -xl/(xl**2 + yl**2 + zl**2)**(3./2) \
115           - C2*(xl-xt)/((xt-xl)**2 + (yt-yl)**2 + (zt-zl)**2)**(3./2)
116     Eq8 = yl1
117     Eq9 = -yl/(xl**2 + yl**2 + zl**2)**(3./2) \
118           - C2*(yl-yt)/((xt-xl)**2 + (yt-yl)**2 + (zt-zl)**2)**(3./2)
119     Eq10= zl1
120     Eq11= -zl/(xl**2 + yl**2 + zl**2)**(3./2) \
121           - C2*(zl-zt)/((xt-xl)**2 + (yt-yl)**2 + (zt-zl)**2)**(3./2)
122
123     return[Eq0,Eq1,Eq2,Eq3,Eq4,Eq5,Eq6,Eq7,Eq8,Eq9,Eq10,Eq11]
124
125 #=======================================================#
126 # Solução das equações de movimento
127
128 # Aqui é definido que:
129 #   * O método a ser usado será o RungeKutta4
130 #   * f é oconjunto de equações que serão computadas
131 #   * rtol é a tolerância relativa
132 #   * atol é a tolerância absoluta
133 solver = odespy.RungeKutta4(f, rtol= 1e-12, atol= 1e-12)
134 # Define condições iniciais
135 solver.set_initial_condition(condicaoInicial())
136 # Intervalo de integração
137 t_points = numpy.linspace(To,Tf,N)
138 # Computa soluções discretas de u para o problema nos pontos t_points
139 u, t = solver.solve(t_points)
140
141 #=======================================================#
142 #=======================================================#
143 # Solucões
144
```

```
145 # Posição; Velocidade da Terra
146 x = u[:,0]; vx = u[:,1]
147 y = u[:,2]; vy = u[:,3]
148 z = u[:,4]; vz = u[:,4]
149
150 # Posição; Velocidade da Lua
151 xl = u[:,6];   vxl = u[:,7]
152 yl = u[:,8];   vyl = u[:,9]
153 zl = u[:,10]; vzl = u[:,11]
154
155 # ================================================ #
156
157 def plot(titulo, X, Y, planeta):
158     '''
159     Esta função tem como objetivo gerar gráficos a partir das soluções.
160     '''
161     plt.figure(titulo)
162     plt.plot(X , Y, label=titulo)
163     plt.xlabel(r'$x_{%s00}$'%planeta)
164     plt.ylabel(r'$x_{%s10}$'%planeta)
165     plt.grid()
166     plt.legend(loc=1)
167     plt.savefig('%s.eps'%(titulo))
168
169 plot(u'Solucao 2D para a Terra', x, y, 'T')
170 plot(u'Solucao relativa 2D para a Lua', (xl-x), (yl-y), 'L_R')
171
172 fig = plt.figure()
173 ax = Axes3D(fig)
174 ax.plot(x,y,z,'b-', label= 'Terra')
175 ax.plot(xl,yl,zl,'g-', label= 'Lua')
176 ax.legend()
177 ax.set_xlabel('$x_{u00}$')
178 ax.set_ylabel('$x_{u10}$')
179 ax.set_zlabel('$x_{u20}$')
180 plt.savefig('Solar_3D.eps')
181
182 plt.show()
```

4 Resultados

Na Figura 2 tem-se a solução normalizada para a dinâmica da terra, no período de um ano, projetada nos eixos das abscissas e ordenadas. A excentricidade obtida através dos dados numéricos foi ≈ 0.0181 que possui um erro relativo de aproximadamente 8.4% quando comparado com o valor médio atual que é de aproximadamente 0.0167 [16]. Semelhantemente a Figura 3 apresenta a solução para o movimento relativo (à Terra) da Lua em uma volta completa ao redor da terra. A excentricidade numérica de sua órbita foi de $e \approx 0.0366$, valor muito próximo ao valor $e \approx 0.039 \pm 0.006$ obtido por Kevin em um experimento relativamente simples [17], com erro relativo de 33.3% com relação ao valor médio atual que é de aproximadamente 0.0549 [18]. Na Figura 4 tem-se a solução tridimensional para o sistema como um todo.

Através dos resultados numéricos é possível entender de forma mais clara o significado físico que as equações de movimento desse sistema carregam em si. As órbitas elípticas obtidas, por exemplo, provam que esse sistema está de acordo com a primeira lei de Kepler.

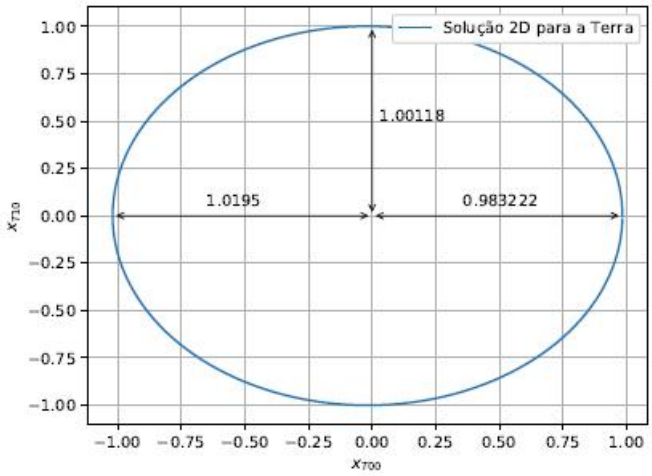

Figura 2. Solução bidimensional para a Terra no período de um ano, que corresponde a uma volta completa ao redor do Sol. A excentricidade da órbita obtida foi de $e \approx 0.0181$.

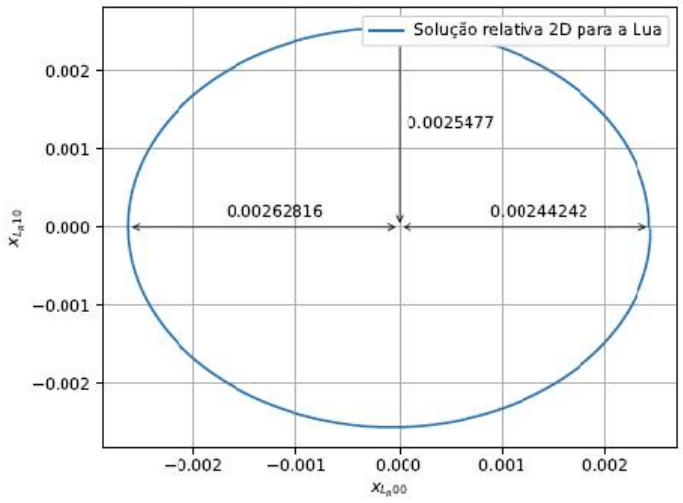

Figura 3. Solução bidimensional para a Lua no período de aproximadamente 27 dias, que corresponde a uma volta ao redor da Terra. A excentricidade da órbita foi de $e \approx 0.0366$.

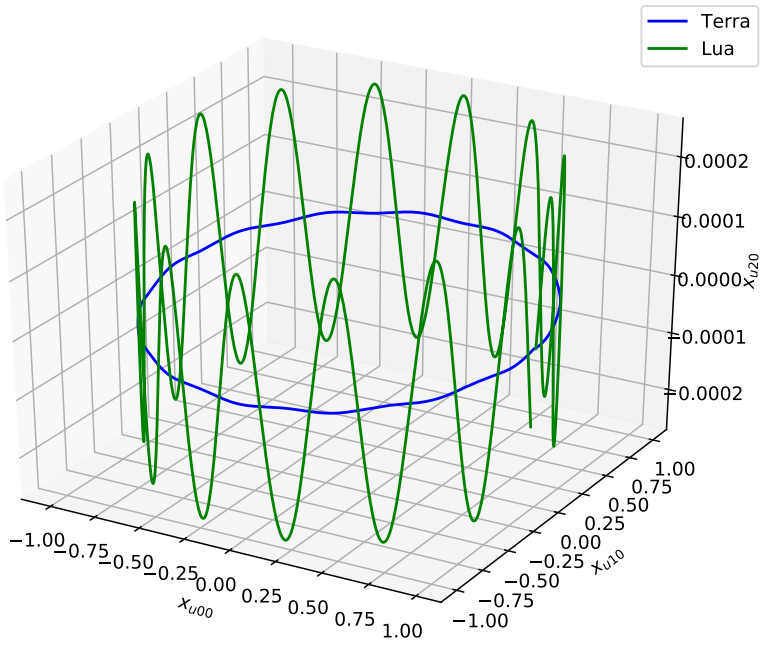

Figura 4. Solução tridimensional. O subíndice "u" representa o acoplamento entre a Terra e a Lua.

5 Conclusões

Neste artigo a computação numérica foi utilizada como ferramenta de apoio na resolução do sistema clássico Sol-Terra-Lua. Através dela foi possível solucionar o sistema de equações diferenciais associado tornando claro seu significado físico. Foram obtidos resultados aceitáveis para o modelo levando-se em consideração que o sistema foi modelado da maneira mais simples possível. Além disso, o código pode ser facilmente adaptado para resolução de outros sistemas com os mais variados níveis de complexidade.

Referências

[1] F. Damasio, "O imcio da revolução científica: questões acerca de copérnico e os epicíclos, kepler e as órbitas elípticas," *Revista Brasileira de Ensino de Física*, vol. 33, no. 3, p. 3602, 2011.

[2] R. Cuzinatto, E. de Morais, and C. N. de Souza, "As observações galileanas dos planetas mcdiccanos de júpiter e a equivalência do mhs e do mcu.," *Caderno Brasileiro de Ensino de Física*, vol. 36, no. 3, 2014.

[3] "Stellarium." http://stellarium.org. Acessado em 31 de Maio de 2018.

[4] "Celestia." http://celestia.space. Acessado em 31 de Maio de 2018.

[5] S. Strogatz, *Nonlinear Dynamics And Chaos*. Studies in nonlinearity, Sarat Book House, 2007.

[6] M. d. P. Guimarães and B. B. Gnecco, "Teaching astronomy and celestial mechanics through virtual reality," *Computer Applications in Engineering Education*, vol. 17, no. 2, pp. 196–205, 2009.

[7] C. Fiolhais and J. Trindade, "Física no computador: o computador como uma ferramenta no ensino e na aprendizagem das ciências físicas," *Revista Brasileira de Ensino de Física*, vol. 25, no. 3, pp. 259–272, 2003.

[8] L. D. d. D. Menezes et al., "Tecnologia no ensino de astronomia na educação básica: análise do uso de recursos computacionais na ação docente," 2011.

[9] A. d. A. Barbosa, D. Í. Ferreira, and E. B. Costa, "Influência da linguagem no ensino introdutório de programação," *in Brazilian Symposium on Computers in Education* (Simpósio Brasileiro de Informática na Educação-SBIE), vol. 25, p. 612, 2014.

[10] A. K. T. Assis, *Mecânica Relacional: e Implementação do Princípio de Mach com a Força de Weber Gravitacional*. C. Roy Keys, Incorporated, 2013.

[11] J. B. Marion, *Classical dynamics of particles and systems*. Academic Press, 2013.

[12] E. Figueiredo and A. S. de Castro, "Um problema de três corpos analiticamente solúvel," *Revista Brasileira de Ensino de Física*, vol. 23, no. 3, pp. 289–293, 2001.

[13] "Python reference manual." http://docs.python.org/2.0/ref/node92.html. Acessado em 31 de Maio de 2018.

[14] L. E. Borges, *Python para Desenvolvedores: Aborda Python 3.3*. Novatec Editora, 2014.

[15] H. P. Langtangen and L. Wang, "Odespy software package." https://github.com/hplgit/odespy. Acessado em 31 de Maio de 2018.

[16] "Excentricidade da órbita terrestre." https://nssdc.gsfc.nasa.gov/planetary/factsheet/Acessado em 14 de Julho de 2018.

[17] K. Krisciunas, "Determining the eccentricity of the moon's orbit without a telescope," *American Journal of Physics*, vol. 78, no. 8, pp. 834–838, 2010.

[18] "Excentricidade da órbita lunar." https://nssdc.gsfc.nasa.gov/planetary/factsheet/Acessado em 14 de Julho de 2018.