Gravitação no sistema solar

Autor:
Rafael Pauwels

Orientadora:
Cecilia Chirenti

Centro de Matemática, Computação e Cognição
Universidade Federal do ABC
Santo André - Brasil

30 de Agosto de 2015

Gravitação no sistema solar

Rafael Pauwels

Resumo:

Este artigo descreve métodos de estudo sobre a gravitação clássica e prova numericamente as leis de Kepler. Dada a dificuldade encontrada em calcular geodésicas e órbitas através das equações de movimento de forma analítica optou-se pelo uso de métodos numéricos, como o de Euler e Runge-Kutta.

Abstract:

This article outlines the methods about classical gravitation and proves in a numeric way Kepler's laws. Given the difficulty found in calculating geodesics and orbits through the equations of motion in an analytic way it is found necessary the use of numerical methods, such as Euler's and Runge-Kutta's.

Conteúdo

1. Introdução

Desde sua elaboração as leis da gravitação já explicaram diversos fenômenos previamente inexplicáveis, como por exemplo as marés e a forma que os planetas se movem no céu, nos serviu também ao prever a existência de Netuno, que foi descoberto quando percebemos as pequenas variações na órbita de Urano que não podiam ser explicadas somente pelas forças exercidas por Saturno e Júpiter.

Atualmente o estudo de órbitas e geodésicas através das leis da gravitação continuam possuindo grande importância no âmbito exploratório e científico espacial, considerando que desprovido de tais conhecimentos seria impossível a elaboração de missões espacias de décadas atrás como na Vostok 1[1] e na missão Rosetta[2], ou mesmo as mais recentes como as missões Falcon 9[3] que dependem imensamente de extrema precisão e onde praticamente não há margens para erro, uma vez que tais missões, devido ao seu alto custo envolvido, não podem ser facilmente refeitas e muitas vezes são uma parte de uma cadeia maior de eventos.

As missões Falcon e Dragon, por exemplo, são destinadas, em sua maior parte, ao abastecimento da Estação Espacial Internacional, qualquer erro na execução da missão poderia causar a paralisação total de diversas outras pesquisas, comprometendo por vezes não só anos de pesquisa mas também a vida dos pesquisadores. Partindo de tais premissas percebemos assim a importância da realização de estudos em relação ao do tema.

2. Objetivos e metas

Os objetivos deste trabalho são os seguintes:

- Compreender a gravitação clássica.
- Escrever um programa capaz de resolver numericamente E.D.O's relacionando os erros de cada método estudado.
- Reproduzir graficamente os erros absolutos de cada método numérico estudado.
- Deduzir uma expressão discreta a partir da equação de movimento.
- Escrever uma versão do programa de cálculo de E.D.O's dedicado ao cálculo da equação de movimento, gerando gráficos e classificando as diferentes órbitas encontradas.
- Utilizar a estrutura desenvolvida a fim de reproduzir cenários fisicamente interessantes, como as órbitas dos planetas ao redor do sol, a órbita da lua com a Terra e as órbitas de cometas e asteroides.

3. Metodologia

Durante a realização desta pesquisa o conteúdo foi estudado de forma simultânea. No entanto, para melhor ilustrar a metodologia utilizada optou-se por descrever os métodos de forma independente, fazendo-se referências apenas quando absolutamente necessário.

3.1 Gravitação Clássica

A gravitação é uma das quatro únicas interações fundamentais que conhecemos, e é dentre todas a mais fraca, só se manifestando de forma perceptível na escala astronômica. Por este motivo os avanços na área da gravitação sempre estão relacionados aos estudos da astronomia. Para o estudo inicial da gravitação clássica utilizou-se o livro *Lições de Física por Feynman*[4] e posteriormente avançando através do livro *Curso de Física Básica vol. 1 Mecânica*[5].
Dado que a excentricidade das órbitas elípticas para diversos planetas é pequena podemos aproximar os resultados por uma órbita circular. E para uma órbita circular a 2^a lei de Kepler, que descreve a relação entre áreas e velocidade, implica que a velocidade seria constante, ou seja, o movimento nessa órbita é *uniforme*. Como a aceleração é, neste caso, centrípeta, a velocidade angular é dada por

$$\vec{a} = -\omega^2 R\hat{r} = -4\pi^2 \frac{R}{T^2}\hat{r} \tag{3.1}$$

agora se chamarmos de m a massa do planeta, temos que a força que atua sobre ele é

$$\vec{F} = ma = -4\pi^2 m\frac{R}{T^2}\hat{r} \tag{3.2}$$

Pela 3^a lei de Kepler, que relaciona os períodos, temos

$$\frac{R^3}{T^2} = C = constante \tag{3.3}$$

onde C tem o mesmo valor para todos os planetas. Reescrevemos assim a 3.1 como

$$\vec{F} = -4\pi^2 C\frac{m}{R^2}\hat{r} \tag{3.4}$$

Newton foi então, desta forma, levado à expressão

$$\vec{F} = -G\frac{mM}{R^2}\hat{r} \tag{3.5}$$

3.2 Métodos Numéricos

Inúmeros modelos matemáticos podem ser formulados em função da taxa de variação de uma ou mais variáveis, naturalmente nos guiando às equações diferenciais. Porém, muitas destas equações simplesmente não podem ser solucionadas de modo analítico, ou são extremamente difíceis de se resolver, o que nos força a recorrer a métodos numéricos[6], que apesar de serem apenas aproximações são em geral suficientemente bons, tendo como único requisito para o cálculo o conhecimento do valor inicial do problema e da derivada da função incógnita.

3.2.1 Método de Euler

A partir de qualquer ponto de uma curva é possível encontrar uma aproximação de um ponto próximo na curva movendo-se ligeiramente pela tangente à curva. Considerando esta curva como sendo:

$$\dot{y}(t) = f(t, y(t)) \tag{3.6}$$

e tendo seu valor inicial $y(t_o) = y_o$ podemos substituir a derivada $\dot{y}$ pela aproximação

$$\dot{y}(t) \approx \frac{y(t+h) - y(t)}{h} \tag{3.7}$$

rearranjando seus elementos obtemos a seguinte fórmula

$$y(t+h) \approx y(t) + h\dot{y}(t) \tag{3.8}$$

e utilizando 3.6 obtemos

$$\dot{y}(t+h) \approx y(t) + hf(t, y(t)) \tag{3.9}$$

Esta fórmula é aplicada escolhendo um tamanho de passo h e um valor inicial $y(t)$. No desenvolvimento deste projeto utilizou-se esta fórmula de forma recursiva, que pode ser escrita como

$$y_{n+1} = y_n + hf(t_n, y_n) \tag{3.10}$$

3.2.2 Método de Runge-Kutta

Existe uma grande variedade de métodos dentro da família de Runge-Kutta, neste projeto o método utilizado é o chamado RK-4, este método funciona da seguinte forma. Seja o problema

$$\dot{y} = f(t, y), y(t_o) = y_o \tag{3.11}$$

Aqui y é uma função desconhecida que desejamos obter uma aproximação. É dado que $\dot{y}$, a taxa em qual y varia, é uma função de t e de y, enquanto o

valor inicial de y é y_o. A partir disso é escolhido um passo h tal que $h > 0$ e o seguinte é definido por

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$
$$t_{n+1} = t_n + h$$

onde

$$k_1 = f(t_n, y_n)$$
$$k_2 = f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1)$$
$$k_3 = f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2)$$
$$k_4 = f(t_n + h, y_n + hk_3)$$

Aqui y_{n+1} é a aproximação por RK-4 de $y(t_{n+1})$ e é determinado pelo valor atual de y_n adicionado aos quatro incrementos, onde cada incremento é baseado do seguinte

- k_1 é o incremento baseado na inclinação no começo do intervalo, usando y;
- k_2 é o incremento baseado na inclinação no ponto médio do intervalo, usando $y + \frac{h}{2}k_1$;
- k_3 é de novo um incremento baseado na inclinação do ponto médio do intervalo, mas desta vez usando $y + \frac{h}{2}k_2$;
- k_4 é o incremento baseado na inclinação no fim do intervalo, usando $y + hk_3$.

3.3 Programação

Durante o desenvolvimento do projeto foram elaborados oito programas, são eles em ordem de desenvolvimento

Primeira Versão A
: Cálculo numérico pelo método de Euler, não possui interface gráfica e não aceita *input* do usuário.

Primeira Versão B
: Cálculo numérico pelo método de Runge-Kutta, não possui interface gráfica e não aceita *input* do usuário

Segunda Versão
: Cálculo numérico de polinômios por ambos os métodos, não possui interface gráfica mas através de diversas perguntas pelo terminal ele modela e resolve um polinômio de ordem n.

Terceira Versão
Cálculo numérico por ambos os métodos, não possui interface gráfica mas permite agora que o usuário digite a equação a ser calculada.

Quarta Versão
Cálculo numérico por ambos os métodos, possui interface gráfica e permite que usuário digite a equação a ser calculada e selecione através de *checkboxes* o método escolhido. Usando a biblioteca Matplotlib é capaz de gerar um gráfico com as curvas dos dois métodos.

Quinta Versão
Cálculo numérico por ambos os métodos, possui interface gráfica e permite que usuário digite a equação a ser calculada e selecione através de *checkboxes* o método escolhido, possui também um campo para introdução da resolução da E.D.O., permitindo assim comparar os métodos numéricos com o método analítico. Usando a biblioteca Matplotlib é capaz de gerar um gráfico com as curvas dos três métodos.

Sexta Versão
Cálculo numérico por ambos os métodos, possui interface gráfica e permite que usuário digite a equação a ser calculada e selecione através de *checkboxes* o método escolhido, possui também um campo para introdução da resolução da E.D.O., permitindo assim o cálculo do erro absoluto. Usando a biblioteca Matplotlib é capaz de gerar dois gráficos, um com as três curvas dos métodos e outro com as duas curvas dos erros absolutos.

Programa de órbitas
Dado um passo, tempo de integração, massa e as condições iniciais necessárias para satisfazer a E.D.O. (velocidade e posição em ambos os eixos), o programa plota um gráfico referente a órbita e nos fornece informações como o erro de fechamento, periastro e apoastro.

3.3.1 Python

A linguagem escolhida para o projeto foi Python, que por ser uma linguagem não compilada demonstrou velocidade imensamente inferior quando comparada com linguagens compiladas como C/C++ ou Java. Mas apesar desta desvantagem em velocidade o Python se demonstra infinitamente superior em outro aspecto da linguagem, sua característica intrínseca de ser simples. Quando comparado ao Java os códigos em Python tendem a ser de três a cinco vezes menor, enquanto quando comparado ao C++ essa diferença chega a ser de cinco a dez vezes. Devido a essa característica em ser simples e ao grande e crescente número de bibliotecas o Python tem se mostrado uma linguagem em potencial em inúmeras áreas, entre elas a científica.

3.3.2 Módulos Externos

Uma das maiores vantagens de se utilizar uma linguagem altamente difundida é a sua comunidade, e apesar de ser uma linguagem relativamente nova

a comunidade do Python é imensa, e devido a ela a sua biblioteca de módulos também. A utilização destes módulos nos custa algum tempo de execução na ordem dos deci-segundos mas nos recompensa em tempo de desenvolvimento na ordem de dias ou meses, alguns módulos são tão complexos e longos que se faz necessário a participação de mais de um programador para desenvolver e manter o módulo.

PyGTK

O PyGTK[7] é uma imensa biblioteca desenvolvida para Python2.7 e Python 3.2 que nos permite com relativa facilidade a criação de interfaces gráficas multiplataforma, sem a necessidade de nenhuma alteração no código para que ele funcione em outros sistemas.

Matplotlib

O Matplotlib[8] é uma biblioteca de *plots* 2D que seguem o padrão de softwares como MATLAB e Mathematica, sua extensiva quantidade de módulos nos permite representar praticamente qualquer tipo de gráfico e da forma que desejarmos, é possível até mesmo a elaboração de gráficos em três dimensões, que depois são convertidos em uma imagem 2D para *display*.

3.4 Balística

Antes de começarmos de fato a estudar órbitas, é importante entender como funciona a balística, que difere do estudo em órbitas apenas pelo fato de que em balística temos uma gravidade $\vec{g}$ constante enquanto que na gravitação universal $\vec{g}$ varia seguindo a expressão

$$\vec{g} = -\frac{GM}{r^3}\vec{r} \tag{3.12}$$

Tendo como base a equação da velocidade

$$v(t) = v_o + a(t - t_o) \tag{3.13}$$

e sabendo que a velocidade nos eixos podem ser descritas por

$$v_{ox} = v_o cos\theta \qquad v_{oy} = v_o sen\theta \tag{3.14}$$

e a aceleração em y e em x por

$$\ddot{y} = a_y = -g \qquad \ddot{x} = a_x = 0 \tag{3.15}$$

temos ao substituir em 3.13

$$\dot{y} = v_y = v_o sen\theta - gt \qquad \dot{x} = v_x = v_o cos\theta \tag{3.16}$$

nos dando

$$y = v_o sen\theta t - \frac{1}{2}gt^2 \qquad x = v_o cos\theta t \tag{3.17}$$

3.5 Movimento Planetário

O movimento planetário possui um significado especial para a civilização ocidental devido ao grande impacto desta teoria na sociedade, modificando sua forma de entender o universo.[9]

Devemos grande parte desse conhecimento a Kepler, que a partir da análise dos dados coletados durante anos por Tycho Brahe formulou as três seguintes leis.

1. Os planetas descrevem órbitas elípticas ao redor do Sol, que está em um dos focos dessa elipse.

2. O segmento que une o Sol a um planeta qualquer descreve áreas iguais em intervalos de tempos iguais.

3. A proporção $\frac{T^2}{a^3}$ é a mesma para todos os planetas que orbitam o Sol, onde T é o período do planeta e a é a distância média do Sol.

3.5.1 O modelo

Como demonstrado na seção 3.1 a lei universal da gravitação pode ser escrita vetorialmente da seguinte forma

$$\vec{F_1} = -\frac{Gm_1m_2}{r^2}\hat{r} \tag{3.18}$$

A força no segundo corpo difere somente no sinal. Pela segunda lei de Newton $F = ma$ podemos rescrever a lei universal da seguinte forma

$$\vec{a} = -\frac{Gm_2}{r^2}\hat{r} \tag{3.19}$$

Convertendo a equação 3.19 em uma equação diferencial onde $\vec{x}(t)$ é a posição da partícula num dado momento t temos

$$\frac{d^2\vec{x}}{dt^2} = -\frac{Gm_2}{r^2}\hat{r} \tag{3.20}$$

4. Resultados

4.1 Gravitação clássica

O estudo da gravitação clássica se deu pela leitura do livro de Feynman[4] e através da resolução dos exercícios da capítulo 10, do livro do Moysés[5]. Dada a inviabilidade de reproduzir a resolução de todos os exercícios optei por resolver dois dos problemas. O primeiro é um exercício sobre o estudo da massa de um objeto através de sua órbita e o segundo é baseado da primeira suposição da existência de buracos negros feita em 1795 por Laplace.

4.1.1 Massa da lua

Em 1968, a nave espacial Apolo 8 foi colocada numa órbita circular em torno da Lua, a uma altitude de 113 km acima da superfície. O período observado dessa órbita foi de 1h 59 min. Sabendo que o raio da Lua é de 1 738 km, utilize esses dados para calcular a massa da Lua.

Estratégia

Para resolver este problema precisaremos igualar a força gravitacional à força centrípeta, conhecendo o período e o raio total poderemos, através da fórmula que nós obtemos, calcular a massa.

Resolução

Através da comparação das forças na forma

$$F_{gravitacional} = F_{centrípeta}$$

onde $F_{gravitacional} = \frac{GmM}{R^2}$ e $F_{centrípeta} = \frac{mv^2}{R}$, portanto

$$\frac{GmM}{R^2} = \frac{mv^2}{R}$$

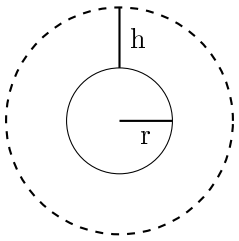
como a velocidade é dada através da relação $velocidade = \frac{\text{espaço}}{tempo}$ e o espaço percorrido é dado pela circunferência temos que espaço $= 2\pi R$
Fazendo as substituições necessárias chegamos em

$$GM = (\frac{2\pi R}{T})^2 R$$

isolando em função da massa obtemos

$$M = \frac{4\pi^2 R^3}{GT^2}$$

onde G é a constante gravitacional universal e R é a soma das distâncias r e h evidenciados na figura.



$$\boxed{h = 113km}$$

$$\boxed{r = 1738km}$$

$$\boxed{t = 7140s}$$

Fazendo as substituições necessárias concluímos que a massa lunar é igual a $7,3638.10^{22}kg$, ou seja, $\frac{1}{81}$ da massa da Terra.

4.1.2 Buraco negro de Laplace

Em 1795, Pierre-Simon de Laplace antecipou a existência de buracos negros, afirmando: "Uma estrela luminosa de mesma densidade que a Terra, cujo diâmetro fosse 250 vezes maior que o do Sol, não permitiria, em consequência de sua atração, que os seus raios luminosos nos atingissem; é possível, portanto, que os maiores corpos luminosos existentes no Universo sejam invisíveis para nós."Embora este raciocínio não-relativístico não se justifique, deduza o resultado de Laplace. Para isto, calcule a velocidade de escape a partir de uma estrela hipotética de mesma densidade que a Terra em função do seu diâmetro e ache o valor crítico do diâmetro.

Estratégia

A ideia para a resolução deste exercício é primeiramente descobrir como calcular a velocidade de escape, uma vez sabendo a velocidade de escape devemos reescrever esta equação em função da densidade. Uma vez em posse de todos os dados basta substituirmos na fórmula para chegarmos ao raio do buraco negro calculado por Laplace.

Resolução

Para chegarmos à equação da velocidade de escape partimos da ideia da conservação de energia.

$$E_{antes} = E_{depois} \tag{4.1}$$

onde E é dada pela soma de todas as energias, no caso a energia cinética K e a energia potencial gravitacional P. Substituindo então as energias em 4.1, chegamos em

$$\frac{mv_i^2}{2} - \frac{GmM}{d_i} = \frac{mv_f^2}{2} - \frac{GmM}{d_f} \tag{4.2}$$

como a E_{depois} se estende ao infinito temos que $v_f \to 0$ e $d_f \to \infty$, portanto

$$\frac{mv_i^2}{2} = \frac{GmM}{d_i} \tag{4.3}$$

$$v = \sqrt{\frac{2GM}{d}} \tag{4.4}$$

sabendo que a densidade μ é dada por $\mu = \frac{massa}{volume}$ chegamos na seguinte equação para uma esfera

$$\mu = \frac{M}{\frac{4}{3}\pi d^3} \tag{4.5}$$

manipulando 4.4 e 4.5 chegamos em

$$v = \sqrt{\frac{2}{3}GM4\pi d^2} \tag{4.6}$$

onde v é igual a velocidade da luz c de valor aproximado $3.10^8 m/s$. Calculamos que d possui valor aproximado de $1,70698.10^{11} m$. Sabendo que o R_{sol} é igual a $6,958.10^8 m$ podemos calcular a razão

$$\frac{d}{R_{sol}} = 245,3265$$

aproximadamente o resultado obtido por Laplace em 1795.

4.2 Métodos numéricos na solução de E.D.O's

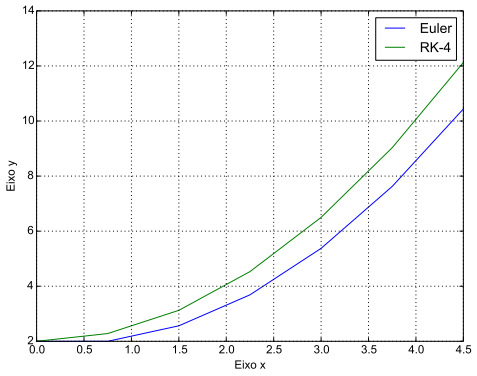
Para a realização das comparações a seguir utilizei a sexta versão do programa descrito na metodologia, o código está disponível no Apêndice A.

4.2.1 Comparação entre métodos

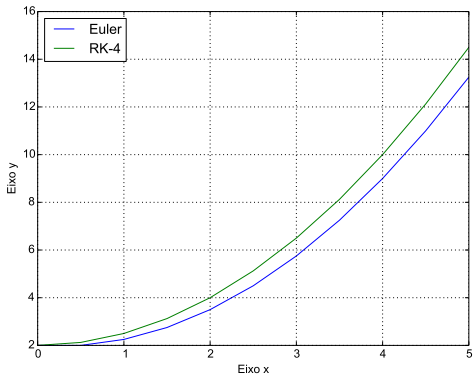
Neste momento da pesquisa utilizou-se diversas E.D.O's para comparação entre os métodos e analisar comportamentos a partir de um passo h tal que $h > 0$.

A pesquisa começou com o teste de algumas E.D.O's simples apenas para comparações de resultado. Exemplificado pelas Figuras 4.1 e 4.2

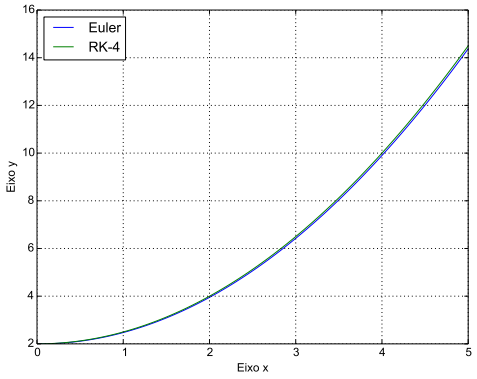
Os métodos analíticos para comparação de resultados serão estudados no decorrer das próximas etapas do projeto junto com a disciplina da UFABC Introdução as Equações Diferenciais Ordinárias. Porém mesmo não comparando com o resultado analítico percebemos claramente que quanto menor h for, menor é a diferença entre a curva pelo método de Euler e pelo método de Runge-Kutta, concluímos assim que há uma maior precisão quando h é menor. Portanto quando $h \to 0, E_{absoluto} \to 0$



(a) $h = 0.75$

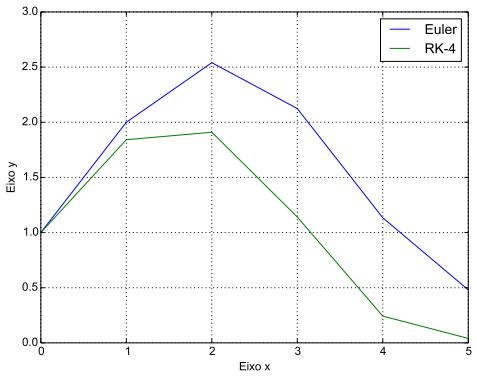


(b) $h = 0.5$

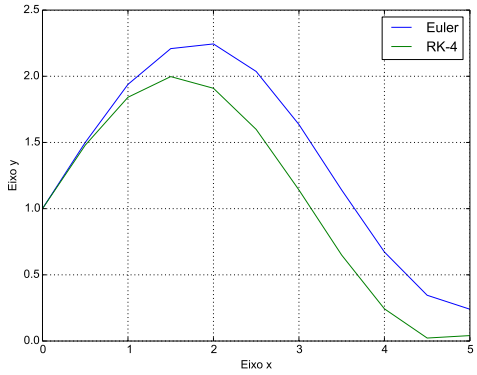


(c) $h = 0.05$

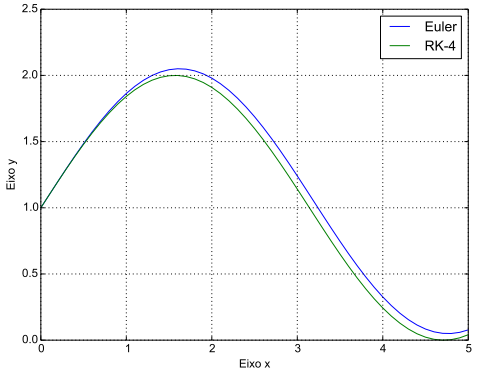
Figura 4.1: Gráficos da solução da E.D.O. $y' = x$ com $y_o = 2$



(a) $h = 1$



(b) $h = 0.5$



(c) $h = 0.1$

Figura 4.2: Gráficos da solução da E.D.O. $y' = cos(x)$ para $y_o = 1$

4.2.2 Erros absolutos

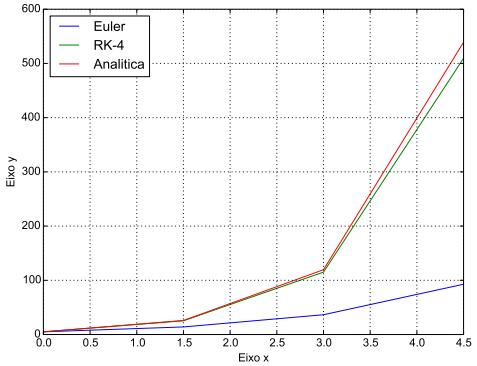
Quando trabalhamos com computadores e calculadoras para resolver equações diferenciais sempre estamos sob um limite físico de informações, isso força o arredondamento de casas decimais, esse arredondamento é chamado de *erro de arredondamento*.

Existe porém outro tipo de erro, que está associado ao método da resolução e que é propagado durante as iterações de y, a este erro damos o nome de *erro de truncamento*.

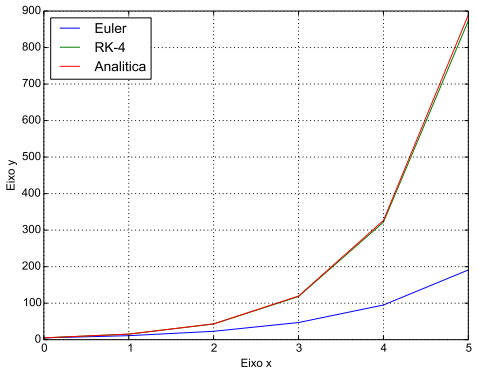
Neste projeto o tipo de erro calculado foi o chamado erro absoluto, que é dado por

$$|f_{analítico} - f_{método}|$$

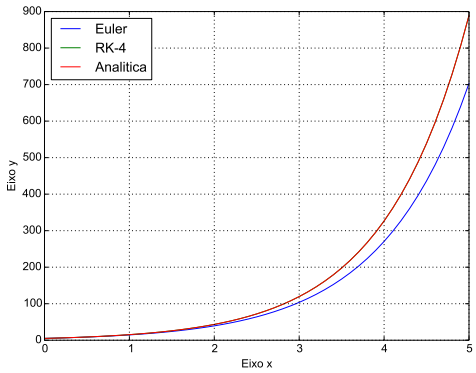
Analisando as figuras 4.3 e 4.4 percebemos o quão impreciso o método de Euler pode ser quando comparado ao RK-4, os erros de Euler neste caso chegam a valores próximos de 700, enquanto o método de Runge-Kutta não ultrapassa a faixa de 30, a curva que representa o passo 0.1 do método RK-4 nem mesmo aparece na figura 4.4, uma vez que seu erro está próximo a 4.10^{-3}.



(a) $h = 1.5$

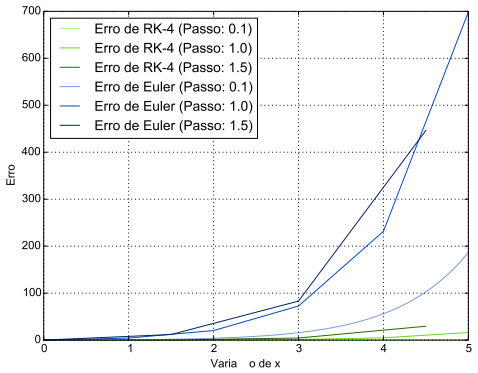


(b) $h = 1$

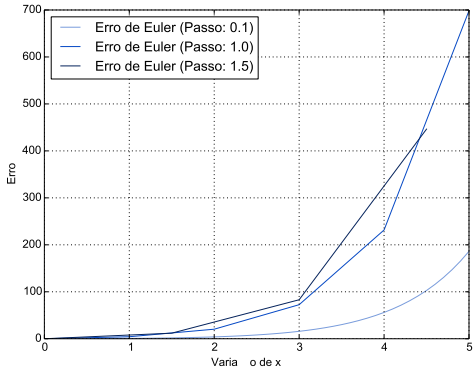


(c) $h = 0.1$

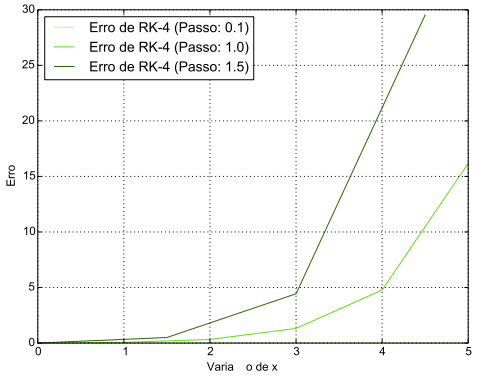
Figura 4.3: Gráficos da solução da E.D.O. $y' = y + 1$ para $y_o = 5$



(a) Todos os erros



(b) Erros de Euler



(c) Erros do RK-4

Figura 4.4: Gráficos dos erros da E.D.O. $y' = y + 1$ para $y_o = 5$ de solução analítica $y(x) = 6e^x - 1$

4.3 Balística

O estudo de balística se deu com auxílio da referência[5]. Começamos com um modelo simples de balística. Um projétil é disparado a partir de um

lançador localizado na origem a uma velocidade fixa, dado o ângulo de elevação devemos determinar quando o projétil atingirá o chão. Por ser um modelo simples estamos descartando influências tais como a resistência do ar e considerando apenas a gravidade como força que age sobre o projétil. Partindo das condições iniciais

$$\begin{aligned} v_o &= 20m/s \\ \theta &= 30^\circ \\ g &= 10m/s \end{aligned}$$

e sabendo que

$$\dot{y} = v_o sen\theta - gt \tag{4.7}$$

substituindo os valores iniciais em 4.7 chegamos em

$$\dot{y} = 10 - 10t \tag{4.8}$$

Utilizando o programa para resolver E.D.O's conseguimos representar graficamente este resultado através da figura 4.5

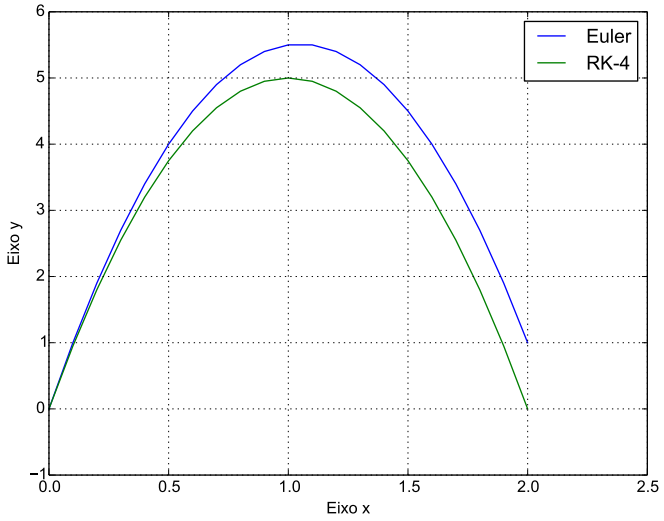


Figura 4.5: Gráfico do deslocamento em x pelo tempo com $h = 0.1$

4.4 Movimento Planetário

Uma vez modelado o problema das equações diferenciais do movimento, basta inserir as condições iniciais da órbita a fim de obter uma simulação de precisão inversamente proporcional ao tamanho do passo. Os valores utilizados como condições iniciais foram retirados da referência[10]. Sendo a

aceleração vetorialmente dada da seguinte forma

$$\frac{d^2\vec{x}}{dt^2} = -\frac{Gm}{r^2}\hat{r} \tag{4.9}$$

podemos reescrever essa aceleração separando o eixo x do y

$$a = -\frac{Gmx}{r^3} \quad ou \quad a = -\frac{Gmy}{r^3} \tag{4.10}$$

consideramos $G \approx 6,67.10^{-11}\frac{N.m^2}{kg^2}$ e r é obtido por

$$r = \sqrt{x^2 + y^2} \tag{4.11}$$

Sabendo que o periélio da Terra é $1,47.10^{11}$ m e sua velocidade máxima é $3,027.10^4$ $m\ s^{-1}$ geramos o gráfico 4.6 para o passo $h = 5000$ s, já o gráfico 4.7 é uma figura gerada com o passo $h = 40000$ s Usando diferentes valores de passos começamos a encontrar diferenças no *erro de fechamento*, esse erro será melhor abordado na próxima subseção.

4.4.1 Erro de fechamento

O método numérico utilizado nas simulações é o método de Runge-Kutta que pode ser definido pelo termo inglês como *not symplectic*, ou seja, o método não conserva energia a cada passo. Esse fator faz com que a órbita não *feche*, em outras palavras, o ponto final em y não é o mesmo que o inicial mesmo considerando uma órbita inteira.

Para melhor entender a relação e a proporcionalidade existente entre o passo e o erro foi gerado o gráfico 4.8 através de 49 dados diferentes que vão do passo $h = 2000$ até o passo $h = 98000$, o eixo y representa o erro e o eixo x o passo.

Podemos ver que o erro é uma reta que apresenta oscilações nos passos maiores devido ao método de comparação utilizado, que se baseia no ponto calculado mais próximo de $x_{inicial}$, que não é necessariamente o mesmo para todos os passos. O erro pode ser aproximado pela equação

$$erro \approx \alpha h + \beta \tag{4.12}$$

conhecendo a expressão 4.12 e usando alguns dos valores do gráfico 4.8, podemos encontrar α e β, prevendo de forma aproximada qual é o erro associado a determinado passo.

4.4.2 Excentricidade orbital

A excentricidade orbital mede o grau de afastamento de uma órbita de um formato circular, a excentricidade normalmente varia entre 0 e 1 mas valores superiores podem ser encontrados.

De forma geral as excentricidades se relacionam à geometria da órbita da seguinte forma

Circular	$e = 0$
Elíptica	$0 < e < 1$
Parabólica	$e = 1$
Hiperbólica	$e > 1$

Netuno é o planeta com a órbita mais circular do sistema solar, enquanto Mercúrio é o que possuí a órbita mais elíptica

Mercúrio	0,2056
Vênus	0,0068
Terra	0,0167
Marte	0,093
Júpiter	0,048
Saturno	0,056
Urano	0,046
Netuno	0,0097

Com o objetivo de comparar o valor calculado com o valor real a mesma simulação feita anteriormente para o erro de fechamento foi feita para a excentricidade da Terra. A figura 4.9 representa a excentricidade encontrada em cada passo, a técnica para se calcular a excentricidade parte da obtenção do periélio e do afélio, representados respectivamente por d_p e d_a

$$e = \frac{d_a - d_p}{d_a + dp} \tag{4.13}$$

Observando o gráfico 4.9 percebemos que a precisão varia de forma linear e por não se tratar de uma medida tão imprecisa quanto o erro de fechamento não vemos dessa vez as oscilações nos resultados.

4.5 Leis de Kepler

A seguir estão as demonstrações numéricas das três leis de Kepler, que como citado em 3.5 são

1. Os planetas descrevem órbitas elípticas ao redor do Sol, que esta em um dos focos dessa elipse.

2. O segmento que une o Sol a um planeta qualquer descreve áreas iguais em intervalos de tempos iguais.

3. A proporção $\frac{T^2}{a^3}$ é a mesma para todos os planetas que orbitam o Sol, onde T é o período do planeta e a é a distância média do Sol.

4.5.1 Primeira Lei

A demonstração dessa lei já foi feita indiretamente 4.4.2, quando trabalhamos com a fórmula 4.13 percebemos que as órbitas do sistema solar apresentam e tal que $0 < e < 1$, ou seja, as órbitas são todas elípticas tendo o Sol com um dos focos dessa elipse.

4.5.2 Segunda Lei

Existe uma grande problemática na prova numérica da segunda lei, que é o modo de se calcular a área, a técnica utilizada nesse projeto é uma aproximação simples porém funcional, ela envolve dividir as áreas em pequenos quadradinhos e então contar manualmente quantos quadrados ocupam a área. A expressão a ser satisfeita é

$$\frac{A_1}{\Delta t} = \frac{A_2}{\Delta t} \tag{4.14}$$

A imagem 4.10 representa o método utilizado, é importante notar que devido à baixa excentricidade da órbita as geometria das áreas são extremamente similares.

4.5.3 Terceira Lei

A terceira lei não é inteiramente provável numericamente, uma vez que ela se originou a partir de uma dedução matemática e não da simples observação do movimento planetário.

$$T^2 = Ca^3 \tag{4.15}$$

No entanto é possível fazer o cálculo de verificação da seguinte forma. O T em 4.15 nada mais é que o tempo gasto para completar uma elipse na simulação e a é a média simples entre o periastro (condição inicial) e o apoastro (numericamente calculado). O valor da constante varia para cada planeta, para a Terra por exemplo, a constante encontrada é $C \approx 3.10^{-19}$ já para Mercúrio a constante vale $C \approx 8,96.10^{-20}$.

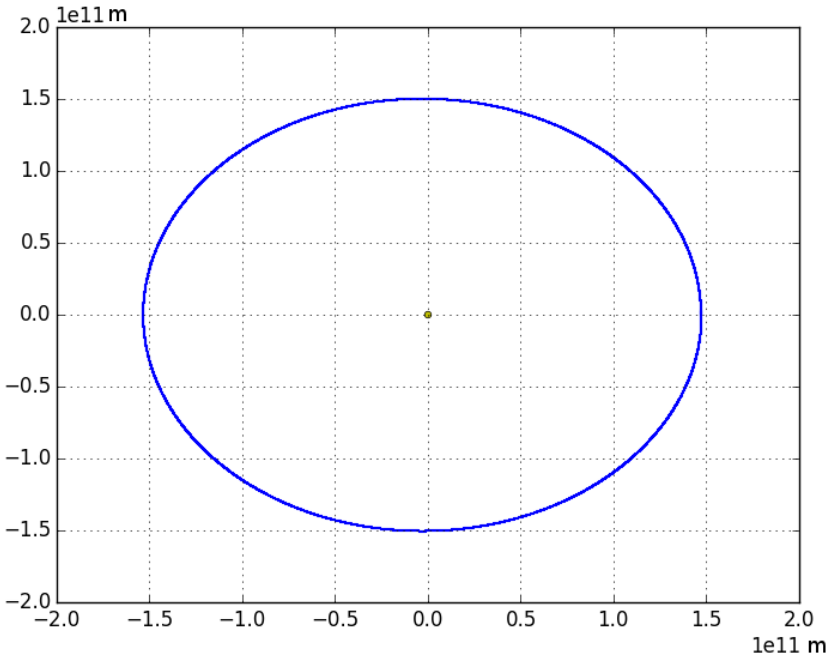


Figura 4.6: Órbita terrestre com $h = 5000$

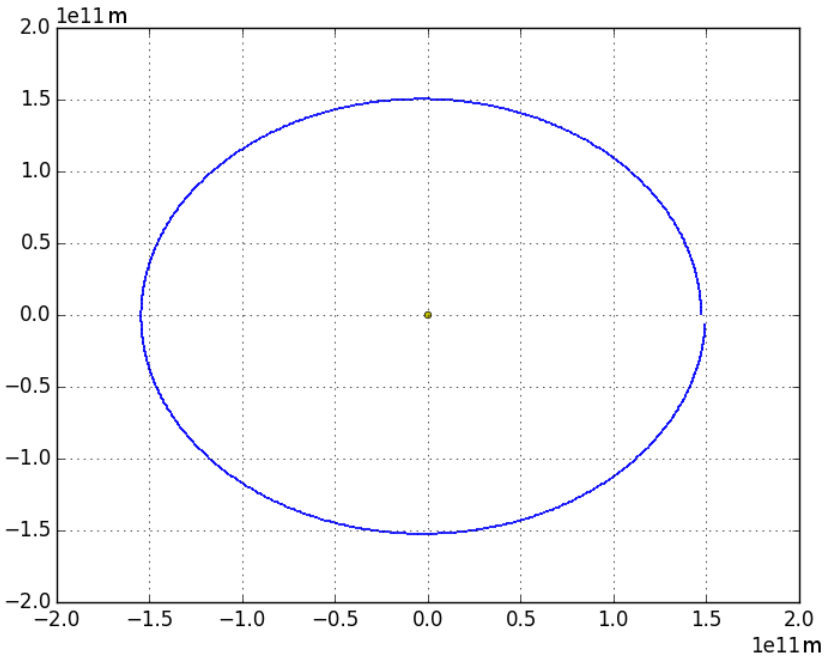


Figura 4.7: Órbita terrestre com $h = 40000$

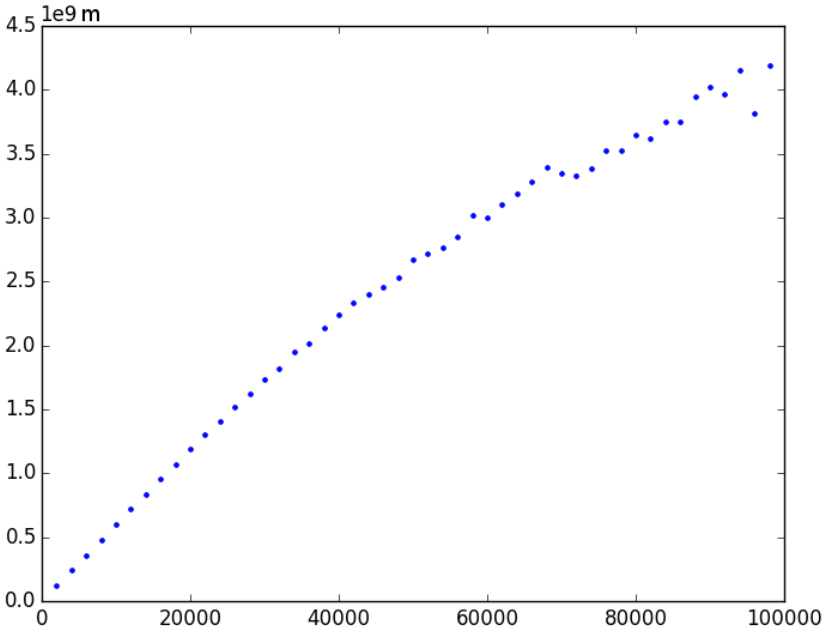


Figura 4.8: Relação entre passo e erro de fechamento

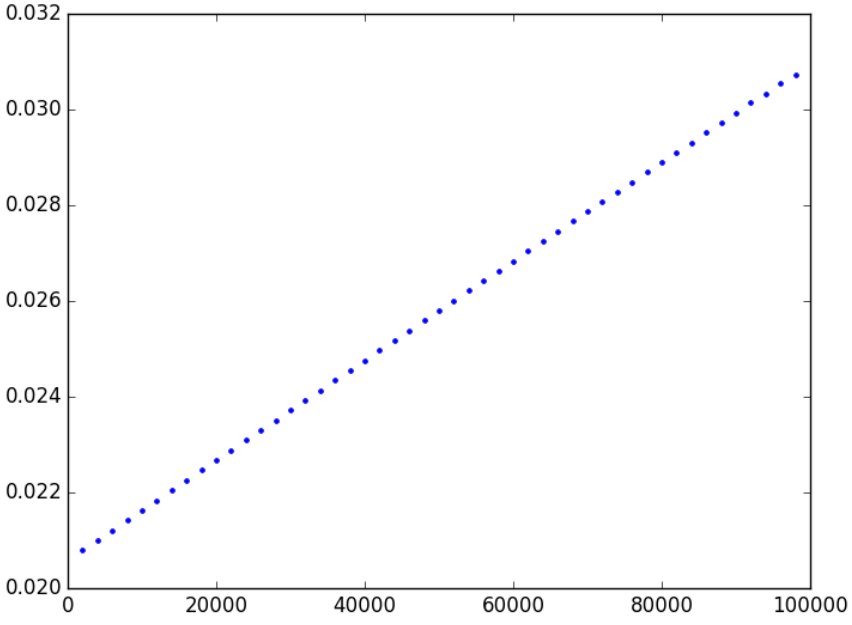


Figura 4.9: Relação entre excentricidade e passo

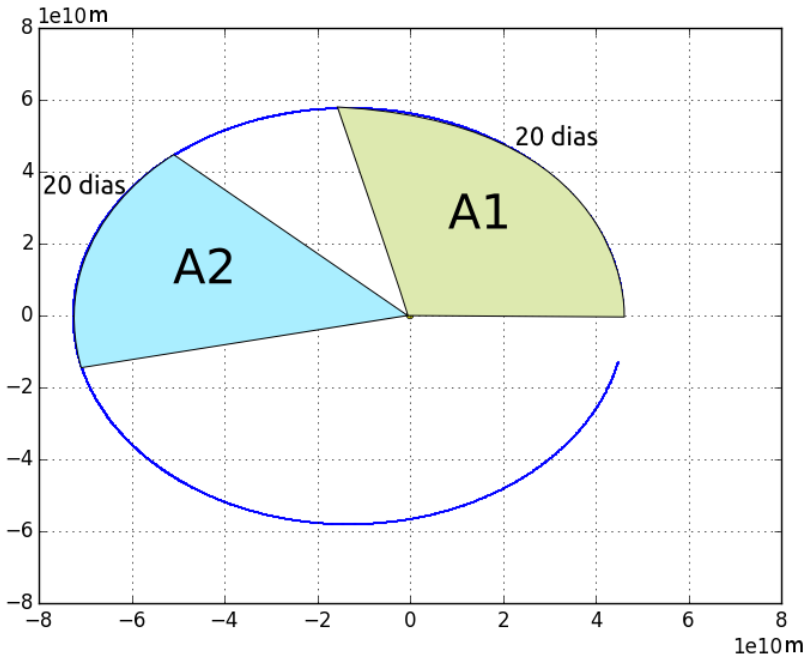


Figura 4.10: Órbita de Mercúrio com duas áreas iguais marcadas

5. Discussão

O desenvolvimento deste projeto possibilitou o estudo de três diferentes áreas do conhecimento, a Física, a Matemática e a Computação.

Na área da física pode-se estudar mais sobre a história da gravitação clássica[4, 5], entendendo desta forma o surgimento das fórmulas e constantes tanto utilizadas durante os cálculos, que por sua vez também foram estudados e trabalhados em forma de exercícios[5].

Para o desenvolvimento da parte física do projeto também foi necessário um desenvolvimento na parte matemática, desde o começo se fez uso de conceitos de cálculo e geometria analítica importantes, dando assim uma aplicação fisicamente útil a conceitos previamente desenvolvidos.

Desenvolveu-se também diversos programas em Python com dois principais objetivos, resolver E.D.O's numericamente e solucionar as equações de movimento e gerar órbitas e geodésicas graficamente. Tais programas exigiram um estudo tanto na linguagem quanto nos métodos computacionais[11].

6. Cronograma

Este projeto possui a duração de 9 meses, de 01/09/2014 a 31/07/2015 e foi concluído dentro do prazo estipulado.

- **01/09/2014 a 30/10/2014** Estudo analítico do problema e revisão da literatura.
  Seguiu como o planejado. Foi feito a revisão da literatura[4, 5].

- **01/11/2014 a 31/12/2014** Estudo dos métodos numéricos e elaboração do programa.
  Durante este momento foi estudado os métodos numéricos de Euler e Runge-Kutta enquanto era desenvolvido as versões iniciais do programa.

- **01/01/2015 a 28/02/2015** Testes numéricos iniciais e interpretação dos resultados.
  Realizou-se os primeiros testes numéricos e as primeiras interpretações, levando ao aprofundamento e aprimoramento do programa. Foi neste período em que os estudos relacionados a erros foi iniciado.

- **01/03/2015 a 30/04/2015** Estudo numérico das leis de Kepler. Elaboração do relatório parcial.
  Término dos estudos com erros e início do estudo de movimento balístico, afim de posteriormente progredir o estudo para órbitas. Ocorreu também a elaboração do relatório parcial.

- **01/05/2015 a 30/06/2015** Estudo numérico das órbitas no sistema solar. Visualização dos resultados.
  Estudo dos erros associados à Lei de Kepler e desenvolvimento do programa capaz de resolver equações de segunda ordem numericamente.

- **01/07/2015 a 31/07/2015** Elaboração do relatório final. Término do estudo e elaboração deste relatório.

Bibliografia

[1] *Vostok 1, NASA*. http://nssdc.gsfc.nasa.gov/nmc/spacecraftDisplay.do?id=1961-012A.

[2] *Rosetta, European Space Agency*. http://sci.esa.int/rosetta/.

[3] *Falcon 9, SpaceX*. http://www.spacex.com/falcon9.

[4] R. P. Feynman. *Lições de Física de Feynman*. Porto Alegre, Brasil: Bookman, 2008.

[5] H. Moysés Nussenzveig. *Curso de Física Básica vol. 1 Mecânica*. São Paulo, Brasil: Edgard Blücher, 2002.

[6] Douglas Quinney. *An Introduction to the Numerical Solution of Differential Equations*. Hertfordshire, Inglaterra: Research Studies Press LTD, 1985.

[7] *PyGTK*. http://www.pygtk.org/.

[8] *Matplotlib*. http://matplotlib.org/.

[9] Gould. H e Tobochnik. J e Christian. W. *An Introduction to Computer Simulation Methods*. Addison-Wesley, 2006.

[10] Bernard Schutz. *Gravity from the ground up*. United Kingdom: Cambridge University Press, 2003.

[11] Jaan Kiusalaas. *Numerical Methods in Engineering With Python 3*. Nova York, Estados Unidos da América: Cambridge University Press, 2013.

A. Diferentes métodos e passos

```
#!usr/bin/env python
#-*- coding: utf-8 -*-

from __future__ import print_function, division
from matplotlib.backends.backend_gtk import FigureCanvasGTK as FigureCanvas
from operator import sub
from matplotlib.figure import Figure
from matplotlib import cm
import numpy as np
from math import *
import matplotlib.pyplot as plt
import gtk, parser, sys

# Calcula a função passada (em string) nos pontos dados (em float)
def no_ponto(x,y,funcao):
    code=parser.expr(funcao).compile()
    x=float(x)
    y=float(y)
    return eval(code)

# Calcula a função passada através do método de euler
def euler(xi,yi,funcao,termino,passoi):
    x=float(xi)
    y=float(yi)
    passo=float(passoi)

    global listaYEulerA, listaYEulerB, listaYEulerC
    listaYEulerA, listaYEulerB, listaYEulerC=[],[],[]

    while x <= termino:
        listaYEulerA.append(y)

        y+=passo*no_ponto(x,y,funcao)
        x+=passo

    x=float(xi)
    y=float(yi)
    passo=float(passoi)*0.1

    while x <= termino:
        listaYEulerB.append(y)
        y+=passo*no_ponto(x,y,funcao)
        x+=passo

    x=float(xi)
    y=float(yi)
    passo=float(passoi)*1.5

    while x <= termino:
        listaYEulerC.append(y)
        y+=passo*no_ponto(x,y,funcao)
        x+=passo

# Calcula a função passada através do método de runge-kutta de quarta ordem
def kutta(xi,yi,funcao,termino,passoi):
    x=float(xi)
```

```
57      y=float(yi)
58      passo=float(passoi)
59
60      global listaYKuttaA, listaYKuttaB, listaYKuttaC
61      listaYKuttaA,listaYKuttaB,listaYKuttaC=[],[],[]
62
63      while x <= termino:
64          listaYKuttaA.append(y)
65
66          k1=no_ponto(x,y,funcao)
67          k2=no_ponto(x+passo*0.5,y+passo*0.5*k1,funcao)
68          k3=no_ponto(x+passo*0.5,y+passo*0.5*k2,funcao)
69          k4=no_ponto(x+passo,y+passo*k3,funcao)
70
71          y+=passo*(k1+2*k2+2*k3+k4)/6.0
72          x+=passo
73
74      x=float(xi)
75      y=float(yi)
76      passo=float(passoi)*0.1
77
78      while x <= termino:
79          listaYKuttaB.append(y)
80
81          k1=no_ponto(x,y,funcao)
82          k2=no_ponto(x+passo*0.5,y+passo*0.5*k1,funcao)
83          k3=no_ponto(x+passo*0.5,y+passo*0.5*k2,funcao)
84          k4=no_ponto(x+passo,y+passo*k3,funcao)
85
86          y+=passo*(k1+2*k2+2*k3+k4)/6.0
87          x+=passo
88
89      x=float(xi)
90      y=float(yi)
91      passo=float(passoi)*1.5
92
93      while x <= termino:
94          listaYKuttaC.append(y)
95
96          k1=no_ponto(x,y,funcao)
97          k2=no_ponto(x+passo*0.5,y+passo*0.5*k1,funcao)
98          k3=no_ponto(x+passo*0.5,y+passo*0.5*k2,funcao)
99          k4=no_ponto(x+passo,y+passo*k3,funcao)
100
101         y+=passo*(k1+2*k2+2*k3+k4)/6.0
102         x+=passo
103
104 # Calcula a função de forma analítica
105 def analitico(xi,funcao,termino,passoi):
106     x=float(xi)
107     passo=float(passoi)
108
109     global listaYAnaliticaA,listaYAnaliticaB,listaYAnaliticaC
110     listaYAnaliticaA,listaYAnaliticaB,listaYAnaliticaC=[],[],[]
111
112     while x <= termino:
113         listaYAnaliticaA.append(no_ponto(x,0,funcao))
114         x+=passo
115
116     x=float(xi)
117     passo=float(passoi)*0.1
118     while x <= termino:
```

```
119         listaYAnaliticaB.append(no_ponto(x,0,funcao))
120         x+=passo
121 
122     x=float(xi)
123     passo=float(passoi)*1.5
124     while x <= termino:
125         listaYAnaliticaC.append(no_ponto(x,0,funcao))
126         x+=passo
127 
128 # Calcula o erro de cada ponto comparando a solução analitica (exata) com o
129 # método de runge-kutta de quarta ordem (aproximado)
130 def kutta_error(solucao_analiticaA,solucao_analiticaB,solucao_analiticaC,solucao_kuttaA,solucao_kuttaB,solucao_ku
131     global listaErroKuttaA,listaErroKuttaB,listaErroKuttaC
132     listaErroKuttaA = map(sub, solucao_analiticaA, solucao_kuttaA)
133     listaErroKuttaB = map(sub, solucao_analiticaB, solucao_kuttaB)
134     listaErroKuttaC = map(sub, solucao_analiticaC, solucao_kuttaC)
135 
136     i=0
137     while i < len(listaErroKuttaA):
138         if listaErroKuttaA[i]<0:
139             listaErroKuttaA[i]=listaErroKuttaA[i]*(-1)
140         i+=1
141     i=0
142     while i < len(listaErroKuttaB):
143         if listaErroKuttaB[i]<0:
144             listaErroKuttaB[i]=listaErroKuttaB[i]*(-1)
145         i+=1
146     i=0
147     while i < len(listaErroKuttaC):
148         if listaErroKuttaC[i]<0:
149             listaErroKuttaC[i]=listaErroKuttaC[i]*(-1)
150         i+=1
151 
152 # Calcula o erro de cada ponto comparando a solução analitica (exata) com o
153 # método de euler (aproximado)
154 def euler_error(solucao_analiticaA,solucao_analiticaB,solucao_analiticaC,solucao_eulerA,solucao_eulerB,solucao_eu
155     global listaErroEulerA,listaErroEulerB,listaErroEulerC
156     listaErroEulerA = map(sub, solucao_analiticaA, solucao_eulerA)
157     listaErroEulerB = map(sub, solucao_analiticaB, solucao_eulerB)
158     listaErroEulerC = map(sub, solucao_analiticaC, solucao_eulerC)
159 
160     i=0
161     while i < len(listaErroEulerA):
162         if listaErroEulerA[i]<0:
163             listaErroEulerA[i]=listaErroEulerA[i]*(-1)
164         i+=1
165     i=0
166     while i < len(listaErroEulerB):
167         if listaErroEulerB[i]<0:
168             listaErroEulerB[i]=listaErroEulerB[i]*(-1)
169         i+=1
170     i=0
171     while i < len(listaErroEulerC):
172         if listaErroEulerC[i]<0:
173             listaErroEulerC[i]=listaErroEulerC[i]*(-1)
174         i+=1
175 # Função que permite a criação de "ranges" com números float e em intervalos
176 # predeterminados
177 def xfrange(inicio,fim,passo):
178     while inicio<=fim:
179         yield inicio
180         inicio+=passo
```

```
181
182 # Constrói o eixo x do gráfico principal
183 def construir_x(inicio,fim,passo):
184     global listaXA,listaXB,listaXC
185     listaXA,listaXB,listaXC=[],[],[]
186     for i in xfrange(inicio,fim,passo):
187         listaXA.append(i)
188     for i in xfrange(inicio,fim,passo*0.1):
189         listaXB.append(i)
190     for i in xfrange(inicio,fim,passo*1.5):
191         listaXC.append(i)
192
193 def quit(widget):
194     sys.exit()
195
196 def erro(warn):
197     erroDialog.show()
198     erroLabel.set_text(warn)
199
200 def erro_hide(widget):
201     erroDialog.hide()
202
203 def atualizar(widget):
204     # Verificação dos tipos dos inputs
205     global funcaoSTR, inicioFLOAT, fimFLOAT, passoFLOAT, yinicialFLOAT
206     try:
207         funcaoSTR=str(funcao.get_text())
208         try:
209             inicioFLOAT=float(inicio.get_text())
210             fimFLOAT=float(fim.get_text())
211             try:
212                 passoFLOAT=float(passo.get_text())
213                 try:
214                     yinicialFLOAT=float(yinicial.get_text())
215                     try:
216                         atualizar_grafico_principal(widget)
217                     except:
218                         erro("Há um erro na função ou intervalo.")
219                         raise
220                 except ValueError:
221                     erro("Há um erro no passo definido. (ValueError)")
222                     raise
223             except ValueError:
224                 erro("Há um erro com o passo definido. (ValueError)")
225                 raise
226         except ValueError:
227             erro("Há um erro no intervalo definido. (ValueError)")
228             raise
229     except ValueError:
230         erro("Há um erro com a função definida. (ValueError)")
231         raise
232
233 def atualizar_grafico_principal(widget):
234     try:
235         if eulerCheckbox.get_active()==True:
236             euler(inicioFLOAT,yinicialFLOAT,funcaoSTR,fimFLOAT,passoFLOAT)
237         if kuttaCheckbox.get_active()==True:
238             kutta(inicioFLOAT,yinicialFLOAT,funcaoSTR,fimFLOAT,passoFLOAT)
239         if analiticoCheckbox.get_active()==True:
240             analitico(inicioFLOAT,solucaoAnaliticaSTR,fimFLOAT,passoFLOAT)
241         try:
242             if eulerCheckbox.get_active()==False:
```

```
                global listaYEulerA, listaYEulerB, listaYEulerC
                listaYEulerA, listaYEulerB, listaYEulerC=[],[],[]
            if kuttaCheckbox.get_active()==False:
                global listaYKuttaA, listaYKuttaB, listaYKuttaC
                listaYKuttaA,listaYKuttaB,listaYKuttaC=[],[],[]
            if analiticoCheckbox.get_active()==False:
                global listaYAnalitica
                listaYAnalitica=[]
            graficoPrincipal.remove(graficoPrincipal.get_child())
            grafico_principal(inicioFLOAT,fimFLOAT,passoFLOAT,listaYEulerA,listaYKuttaA, listaYAnaliticaA)
            try:
                try:
                    grafico2DErroXx.remove(grafico2DErroXx.get_child())
                except:
                    pass
                if eulerCheckbox.get_active()==True or kuttaCheckbox.get_active()==True:
                    graficos_erros(listaYEulerA,listaYEulerB,listaYEulerC,listaYKuttaA,listaYKuttaB,listaYKuttaC,
            except:
                erro("Ocorreu um erro ao plotar o gráfico dos erros.")
                raise
        except:
            erro("Ocorreu um erro ao plotar o gráfico.")
            raise
    except:
        erro("Ocorreu um erro ao executar os métodos númericos.\nConfira seus dados.")
        raise
    if analiticoCheckbox.get_active()==True:
        try:
            analitico(inicioFLOAT,solucaoAnaliticaSTR,fimFLOAT,passoFLOAT)
        except:
            erro("Não foi possível fazer o cálculo analítico.\nConfira sua solução.")
            raise

def grafico_principal(inicio,fim,passo,listaEuler=[],listaKutta=[],listaAnalitica=[],listaErroKutta=[],listaErroE
    construir_x(inicio,fim,passo)
    grafico1=Figure(figsize=(8,6), dpi=72)
    axis1=grafico1.add_subplot(111)
    axis1.grid(True)
    axis1.set_xlabel("Eixo x")
    axis1.set_ylabel("Eixo y")

    if eulerCheckbox.get_active()==True:
        axis1.plot(listaXA, listaEuler, label="Euler",color="b")
    if kuttaCheckbox.get_active()==True:
        axis1.plot(listaXA, listaKutta, label="RK-4",color="g")
    if analiticoCheckbox.get_active()==True:
        axis1.plot(listaXA, listaAnalitica, label="Analitica",color="r")

    axis1.legend(loc='best')

    canvas1=FigureCanvas(grafico1)
    canvas1.set_size_request(700,500)
    graficoPrincipal.add_with_viewport(canvas1)
    janelaPrincipal.show_all()

def graficos_erros(listaEulerA=[],listaEulerB=[],listaEulerC=[],listaKuttaA=[],listaKuttaB=[],listaKuttaC=[],list
    if analiticoCheckbox.get_active()==True:
        #GRAFICO 2 INFERIOR DIREITO
        grafico2=Figure(figsize=(8,6), dpi=72)
        axis2=grafico2.add_subplot(111)
        axis2.grid(True)
        axis2.set_xlabel("Variação de x")
```

```
305             axis2.set_ylabel("Erro")
306 
307             if kuttaCheckbox.get_active()==True:
308                 kutta_error(listaAnaliticaA,listaAnaliticaB,listaAnaliticaC,listaKuttaA,listaKuttaB,listaKuttaC)
309             if eulerCheckbox.get_active()==True:
310                 euler_error(listaAnaliticaA,listaAnaliticaB,listaAnaliticaC,listaEulerA,listaEulerB,listaEulerC)
311 
312             if kuttaCheckbox.get_active()==True:
313                 label1a="Erro de RK-4 (Passo: "+str(passoFLOAT)+")"
314                 label1b="Erro de RK-4 (Passo: "+str(passoFLOAT*0.1)+")"
315                 label1c="Erro de RK-4 (Passo: "+str(passoFLOAT*1.5)+")"
316                 axis2.plot(listaXB, listaErroKuttaB, label=label1b,color="#B1FF79")
317                 axis2.plot(listaXA, listaErroKuttaA, label=label1a,color="#67DA15")
318                 axis2.plot(listaXC, listaErroKuttaC, label=label1c,color="#2C6703")
319             if eulerCheckbox.get_active()==True:
320                 label2a="Erro de Euler (Passo: "+str(passoFLOAT)+")"
321                 label2b="Erro de Euler (Passo: "+str(passoFLOAT*0.1)+")"
322                 label2c="Erro de Euler (Passo: "+str(passoFLOAT*1.5)+")"
323                 axis2.plot(listaXB, listaErroEulerB, label=label2b,color="#799CDC")
324                 axis2.plot(listaXA, listaErroEulerA, label=label2a,color="#0649C5")
325                 axis2.plot(listaXC, listaErroEulerC, label=label2c,color="#05235C")
326 
327             axis2.legend(loc='best')
328 
329             canvas2=FigureCanvas(grafico2)
330             canvas2.set_size_request(500,500)
331             grafico2DErroXx.add_with_viewport(canvas2)
332 
333             janelaPrincipal.show_all()
334 
335     def analitico_dialog(widget):
336         try:
337             labelSolucaoAnalitica.set_text("Solução da E.D.O.\nPara y(" + str(float(inicio.get_text())) + ") = " + st
338         except:
339             labelSolucaoAnalitica.set_text("Solução da E.D.O.\nPara certo y(xo)")
340             raise
341         if(analiticoCheckbox.get_active()==True):
342             solucaoAnaliticaDialog.show()
343 
344     def cancelar(widget):
345         solucaoAnaliticaDialog.hide()
346         analiticoCheckbox.set_active(False)
347 
348     def add_solucao_analitica(widget):
349         global solucaoAnaliticaSTR
350         try:
351             solucaoAnaliticaSTR=str(solucaoAnaliticaEntrada.get_text())
352             solucaoAnaliticaDialog.hide()
353         except:
354             erro("Digite a solução")
355             raise
356 
357     def main():
358         builder=gtk.Builder()
359         builder.add_from_file("gui.glade")
360 
361         #erroDialog
362         global erroDialog, erroLabel
363         erroDialog=builder.get_object("erroDialog")
364         erroLabel=builder.get_object("erroLabel")
365         #janelaPrincipal
366         global janelaPrincipal, funcao, inicio, fim, passo, yinicial
```

```
367     janelaPrincipal=builder.get_object("janelaPrincipal")
368     funcao=builder.get_object("funcaoEntrada")
369     inicio=builder.get_object("inicioEntrada")
370     fim=builder.get_object("fimEntrada")
371     passo=builder.get_object("passoEntrada")
372     yinicial=builder.get_object("yinicialEntrada")
373     #janelaPrincipal - Métodos
374     global analiticoCheckbox, eulerCheckbox, kuttaCheckbox
375     analiticoCheckbox=builder.get_object("analiticoCheckbox")
376     eulerCheckbox=builder.get_object("eulerCheckbox")
377     kuttaCheckbox=builder.get_object("kuttaCheckbox")
378     #janelaPrincipal - Gráficos
379     global graficoPrincipal, grafico2DErroXx
380     graficoPrincipal=builder.get_object("graficoPrincipal")
381     grafico2DErroXx=builder.get_object("grafico2DErroXx")
382     #solucaoAnalitica
383     global solucaoAnaliticaDialog, labelSolucaoAnalitica, solucaoAnaliticaEntrada
384     solucaoAnaliticaDialog=builder.get_object("solucaoAnalitica")
385     labelSolucaoAnalitica=builder.get_object("labelSolucaoAnalitica")
386     solucaoAnaliticaEntrada=builder.get_object("entradaSolucaoAnalitica")
387
388
389     # Conexões dos botões
390     dic={
391     "on_janelaPrincipal_destroy" : quit,
392     "on_okErroBotao_clicked" : erro_hide,
393     "on_botaoAtualizar_clicked" : atualizar,
394     "on_analiticoCheckbox_toggled" : analitico_dialog,
395     "on_botaoCancelar_clicked" : cancelar,
396     "on_botaoAddSolucao_clicked" : add_solucao_analitica,
397     }
398
399     builder.connect_signals(dic)
400     # Loop
401     grafico_principal(0,10,1)
402     janelaPrincipal.show_all()
403     gtk.main()
404
405
406 main()
```

B. Calculo de órbitas

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from __future__ import print_function
from matplotlib.pyplot import * # Plotar os gráficos
from math import * # Função sqrt
import csv # Para análise de dados

constanteG=6.67e-11

massa=1.98e30 # Massa do astro

veloci= 30.27*10**3 # Velocidade em m/s do planeta
raio = 147.1*10**9 # Distância em metros

tempo=(370*24*60*60) # Duração da integração em segundos
dt=1.0*5000 # Passo em segundos

print(sqrt(constanteG*massa/raio)) # Print da velocidade necessária para formar uma órbita circular perfeita

gm=constanteG*massa
def accelx(xx,xy,vx,vy,dt):
    dsq = xx*xx + xy*xy  # distância ao quadrado
    dr = sqrt(dsq)  # distância
    return -gm*xx/dr**3

def accely(xx,xy,vx,vy,dt):
    dsq = xx*xx + xy*xy  # distância ao quadrado
    dr = sqrt(dsq)  # distância
    return -gm*xy/dr**3

def rk4(xx, xy, vx, vy, ax, ay, dt):
        xx1 = xx
        vx1 = vx
        xy1 = xy
        vy1 = vy
        ax1 = ax(xx1,xy1,vx1,vy1,0)
        ay1 = ay(xx1,xy1,vx1,vy1,0)

        xx2 = xx + 0.5*vx1*dt
        vx2 = vx + 0.5*ax1*dt
        xy2 = xy + 0.5*vy1*dt
        vy2 = vy + 0.5*ay1*dt
        ax2 = ax(xx2, xy2, vx2, vy2, dt/2.0)
        ay2 = ay(xx2, xy2, vx2, vy2, dt/2.0)

        xx3 = xx + 0.5*vx2*dt
        vx3 = vx + 0.5*ax2*dt
        xy3 = xy + 0.5*vy2*dt
        vy3 = vy + 0.5*ay2*dt
        ax3 = ax(xx3, xy3, vx3, vy3, dt/2.0)
        ay3 = ay(xx3, xy3, vx3, vy3, dt/2.0)

        xx4 = xx + 0.5*vx3*dt
        vx4 = vx + 0.5*ax3*dt
        xy4 = xy + 0.5*vy3*dt
        vy4 = vy + 0.5*ay3*dt
```

```
57          ax4 = ax(xx4, xy4, vx4, vy4, dt)
58          ay4 = ay(xx4, xy4, vx4, vy4, dt)
59
60          xxf = xx + (dt/6.0)*(vx1 + 2*vx2 + 2*vx3 + vx4)
61          xyf = xy + (dt/6.0)*(vy1 + 2*vy2 + 2*vy3 + vy4)
62          vxf = vx + (dt/6.0)*(ax1 + 2*ax2 + 2*ax3 + ax4)
63          vyf = vy + (dt/6.0)*(ay1 + 2*ay2 + 2*ay3 + ay4)
64
65          return xxf, xyf, vxf, vyf
66
67  t = 0
68
69  estado = raio, 0., 0., veloci # Posição x, posição y, velocidade x, velocidade y
70
71  colidiu = False
72
73  # Cria o gráfico e adiciona o sol em (0, 0)
74  fig = figure()
75  ax = fig.add_subplot(111)
76  ax.grid()
77  ax.plot(0, 0, 'oy', markersize=4)
78
79  estadoInicial = estado[0] # Posição X em o planeta começou (periastro)
80  estadoApo = estado[0] # Usado para calcular o apoastro
81  estadoFinal = estado[0] # Inicia com um valor alto, usado para calcular o X quando Y esta próximo de 0 (erro de f
82
83  # Roda por x tempo ou até chegar muito próximo da estrela
84  while t < tempo and not colidiu:
85          t += dt
86          estado = rk4(estado[0], estado[1], estado[2], estado[3], accelx, accely, dt)
87          ax.plot(estado[0], estado[1], 'b.', markersize=2)
88          if estadoApo < -1*estado[0]: # IF para calcular o apoastro
89                  estadoApo = -1*estado[0]
90          if estado[1] < sqrt(estadoFinal**2) and estado[0] > estadoInicial: # IF que calcula o erro de fechamento
91                  estadoFinal = estado[1]
92                  estadoFinalX = estado[0]
93          if sqrt(estado[0]**2+estado[1]**2) < 1e6:
94                  print("O planeta colidiu.")
95                  colidiu=True
96
97  #Excentricidade é calculada por e = r_a-r_p/r_a+r_p
98
99  excen=(estadoApo-estadoInicial)/(estadoApo+estadoInicial)
100 fecha=estadoFinalX-estadoInicial
101
102 print("Passo: ",dt,"Apoastro: ",estadoApo,"Periastro: ",estadoInicial,"Excentricidade: ", excen )
103 print("Erro de fechamento: ", fecha)
104
105 with open('dados.csv', 'ab') as csvfile:
106     spamwriter = csv.writer(csvfile, delimiter=',',
107                             quotechar='|', quoting=csv.QUOTE_MINIMAL)
108     spamwriter.writerow([dt, estadoInicial, estadoApo, excen, fecha])
109
110 fig.show()
111 raw_input("...")
```

C. Grapher

```
1  import csv
2  import matplotlib.pyplot as plt
3
4  with open('dados.csv') as f:
5      data=[tuple(line) for line in csv.reader(f)]
6
7  print data
8
9  print len(data)
10
11 print data[7][0]
12
13 n=0
14 eixoy=[]
15 eixox=[]
16 while n < len(data):
17         eixoy.append(data[n][0])
18         n+=1
19 n=0
20 while n < len(data):
21         eixox.append(data[n][3])
22         n+=1
23
24
25 plt.plot(eixoy, eixox, 'b.')
26
27 plt.show()
28 raw_input('...')
```