

Gravitação no sistema solar

Autor:

Rafael PAUWELS

Orientadora:

Cecilia CHIRENTI

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

29 de Abril de 2015

Gravitação no sistema solar

Rafael PAUWELS

Resumo:

Este projeto tem como objeto de estudo as órbitas e geodésicas em gravitação. Iniciamos com um estudo sobre gravitação clássica e evoluímos para um estudo que envolve a resolução de equações diferenciais ordinárias através dos métodos numéricos de Euler e Runge-Kutta, possibilitando assim a resolução das equações de movimento.

Abstract:

This project aims at studying orbits and geodesics in gravitation. Starting with a study in classical gravitation then evolving to a study that involves the resolution of ordinary differential equations through the Euler and Runge-Kutta numerical methods, thus enabling the solving of the equations of motion.

Conteúdo

Conteúdo	2
1 Introdução	3
2 Objetivos e metas	4
3 Metodologia	5
3.1 Gravitação Clássica	5
3.2 Métodos Numéricos	6
3.2.1 Método de Euler	6
3.2.2 Método de Runge-Kutta	6
3.3 Programação	7
3.3.1 Python	8
3.3.2 Módulos Externos	8
3.4 Balística	9
4 Resultados	10
4.1 Gravitação clássica	10
4.1.1 Massa da lua	10
4.1.2 Buraco negro de Laplace	11
4.2 Métodos numéricos na solução de E.D.O's	12
4.2.1 Comparação entre métodos	12
4.2.2 Erros absolutos	15
4.3 Balística	17
5 Discussão	19
6 Cronograma	20
Bibliografia	20
Apêndice A Código	22

1. Introdução

O estudo de órbitas e geodésicas abertas e fechadas é de extrema importância no âmbito exploratório e científico espacial, considerando que desprovido de tais conhecimentos seria impossível a elaboração de missões espaciais 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 paralização 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 ao redor 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 asteróides.

3. Metodologia

Durante a realização desta pesquisa o conteúdo foi estudado de forma simultânea, já que todo conjunto possui como requisito o entendimento, mesmo que parcial, dos outros dois conjuntos. 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 2ª 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

$$a = -\omega^2 R \hat{\mathbf{r}} = -4\pi^2 \frac{R}{T^2} \hat{\mathbf{r}} \quad (3.1)$$

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

$$F = ma = -4\pi^2 m \frac{R}{T^2} \hat{\mathbf{r}} \quad (3.2)$$

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

$$\frac{R^3}{T^2} = C = \text{constante} \quad (3.3)$$

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

$$F = -4\pi^2 C \frac{m}{R^2} \hat{\mathbf{r}} \quad (3.4)$$

Newton foi então, desta forma, levado a expressão

$$F = -G \frac{mM}{R^2} \hat{\mathbf{r}} \quad (3.5)$$

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.

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:

$$y'(t) = f(t, y(t)) \quad (3.6)$$

e tendo seu valor inicial $y(t_o) = y_o$ podemos substituir a derivada y' pela aproximação

$$y'(t) \approx \frac{y(t+h) - y(t)}{h} \quad (3.7)$$

rearranjando seus elementos obtemos a seguinte fórmula

$$y(t+h) \approx y(t) + hy'(t) \quad (3.8)$$

e utilizando 3.6 obtemos

$$y'(t+h) \approx y(t) + hf(t, y(t)) \quad (3.9)$$

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

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

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

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

Aqui y é uma função desconhecida a qual 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_0 . 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

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

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 sete 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.

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 decisegundos 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

$$\vec{g} = -\frac{GM}{r^3}\vec{r} \quad (3.12)$$

Tendo como base a equação da velocidade

$$v(t) = v_o + a(t - t_o) \quad (3.13)$$

e sabendo que a velocidade nos eixos podem ser descritas por

$$v_{ox} = v_o \cos\theta \quad v_{oy} = v_o \sin\theta \quad (3.14)$$

e a aceleração em y e em x por

$$\ddot{y} = a_y = -g \quad \ddot{x} = a_x = 0 \quad (3.15)$$

temos ao substituir em 3.13

$$\dot{y} = v_y = v_o \sin\theta - gt \quad \dot{x} = v_x = v_o \cos\theta \quad (3.16)$$

nos dando

$$y = v_o \sin\theta t - \frac{1}{2}gt^2 \quad x = v_o \cos\theta t \quad (3.17)$$

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, gravitação, 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. A seguir a resolução do exercício 1 e 7[5]

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 $velocidade = \frac{\text{espaço}}{\text{tempo}}$ e o espaço percorrido é dado pela circunferência temos que $\text{espaço} = 2\pi R$

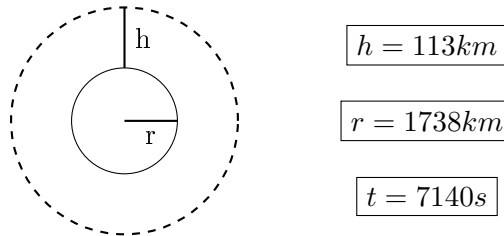
Fazendo as substituições necessárias chegamos em

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

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.



Fazendo as substituições necessárias concluímos que a massa lunar é igual a $7,3638 \cdot 10^{22} kg$.

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 idéia 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 substituímos na fórmula para chegarmos ao raio do buraco negro calculado por Laplace.

Resolução

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

$$E_{antes} = E_{depois} \quad (4.1)$$

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

$$\frac{mv_i^2}{2} - \frac{GmM}{d_i} = \frac{mv_f^2}{2} - \frac{GmM}{d_f} \quad (4.2)$$

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

$$\frac{mv_i^2}{2} = \frac{GmM}{d_i} \quad (4.3)$$

$$v = \sqrt{\frac{2GM}{d}} \quad (4.4)$$

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

$$\mu = \frac{M}{\frac{4}{3}\pi d^3} \quad (4.5)$$

manipulando 4.4 e 4.5 chegamos em

$$v = \sqrt{\frac{2}{3}GM4\pi d^2} \quad (4.6)$$

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

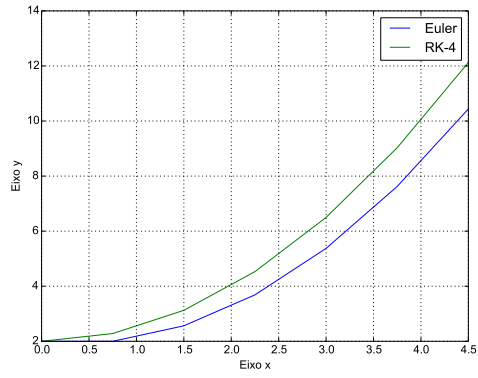
Para a realização do testes a seguir foi utilizado a sexta versão do programa descrito na metodologia, o código está disponível no Apêndice A.

4.2.1 Comparação entre métodos

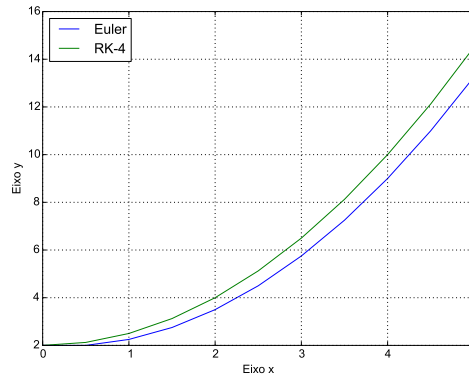
Neste momento da pesquisa utilizou-se diversas E.D.O's para comparação entre os métodos e analisar comportamentos a partir de um passo h tal que $h > 0$.

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

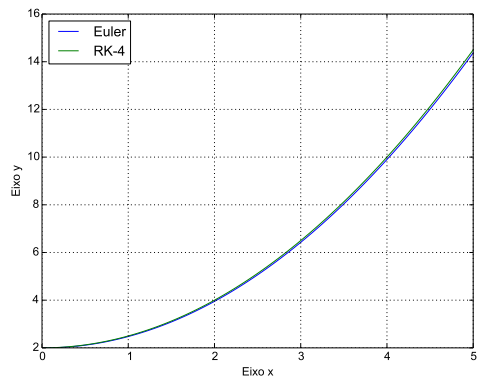
Os métodos analíticos para comparação de resultados serão estudados no decorrer das próximas etapas do projeto junto com a disciplina da UFABC Introdução as Equações Diferenciais Ordinárias. Porém mesmo não comparando com o resultado analítico percebemos claramente que quanto menor h for, menor é a diferença 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 $h \rightarrow 0$, $E_{absoluto} \rightarrow 0$



(a) $h = 0.75$

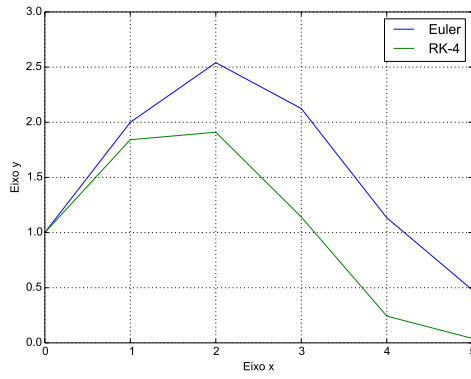


(b) $h = 0.5$

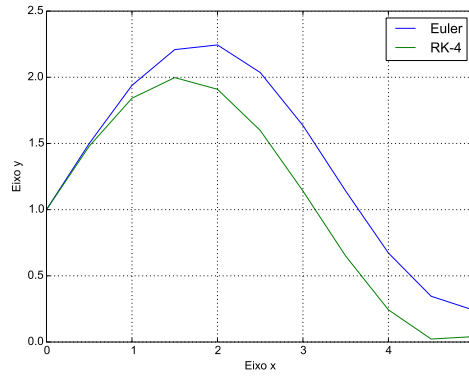


(c) $h = 0.05$

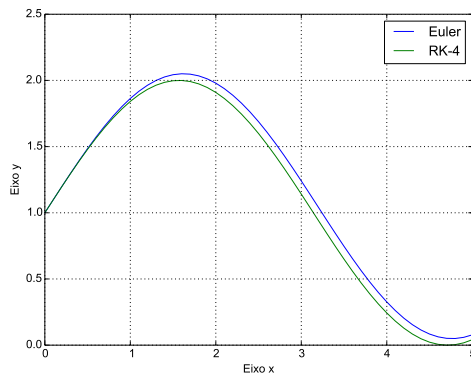
Figura 4.1: Gráficos da solução da E.D.O. $y' = x$ com $y_0 = 2$



(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

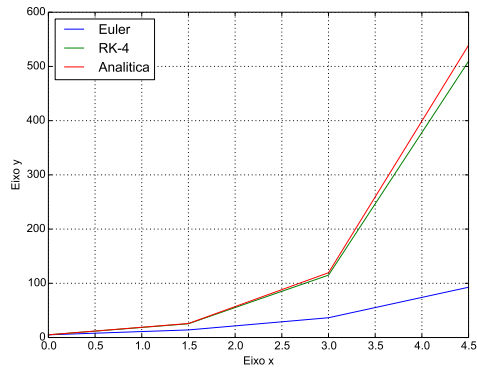
Quando trabalhamos com computadores e calculadoras para resolver equações diferenciais sempre estamos sob um limite físico de informações, isso força o arredondamento de casas decimais, esse arredondamento é chamado de *erro de arredondamento*.

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*.

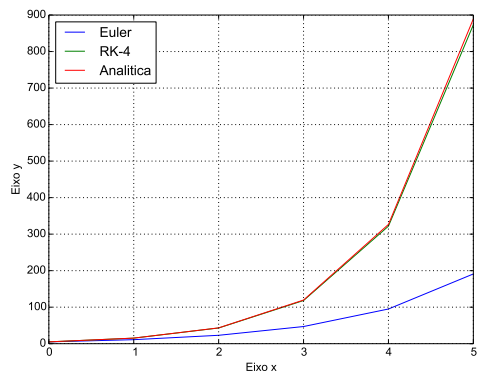
Neste projeto o tipo de erro calculado foi o chamado erro absoluto, que é dado por

$$|f_{\text{analítico}} - f_{\text{método}}|$$

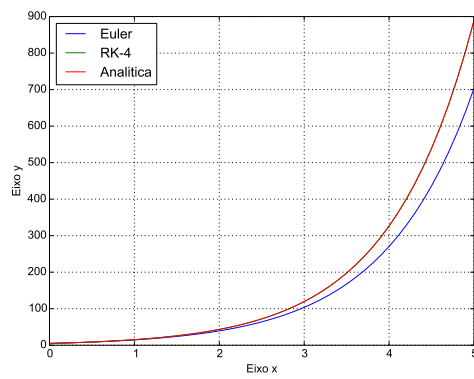
Analisando as figuras 4.3 e 4.4 percebemos o quão impreciso o método de Euler pode ser quando comparado ao RK-4, os erros de Euler neste caso chegam a valores próximos de 700, enquanto o método de Runge-Kutta não ultrapassa a faixa de 30, a curva que representa o passo 0.1 do método RK-4 nem mesmo aparece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



(a) $h = 1.5$

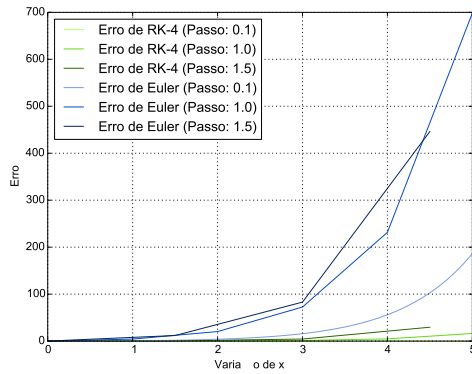


(b) $h = 1$

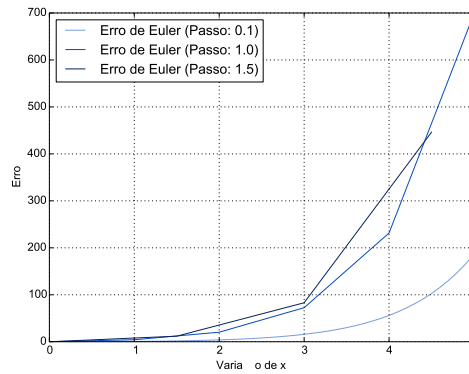


(c) $h = 0.1$

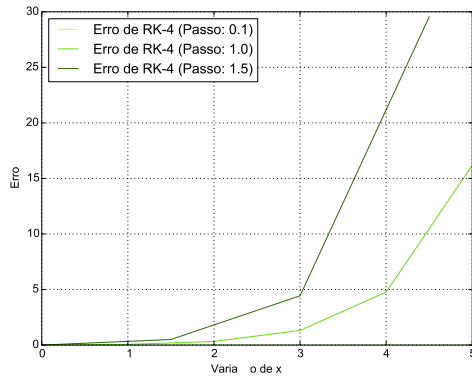
Figura 4.3: Gráficos da solução da E.D.O. $y' = y + 1$ para $y_0 = 5$



(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 $y_0 = 5$ 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, tomando em relação a angulação 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

$$v_o = 20m/s$$

$$\theta = 30^\circ$$

$$g = 10m/s$$

e sabendo que

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

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

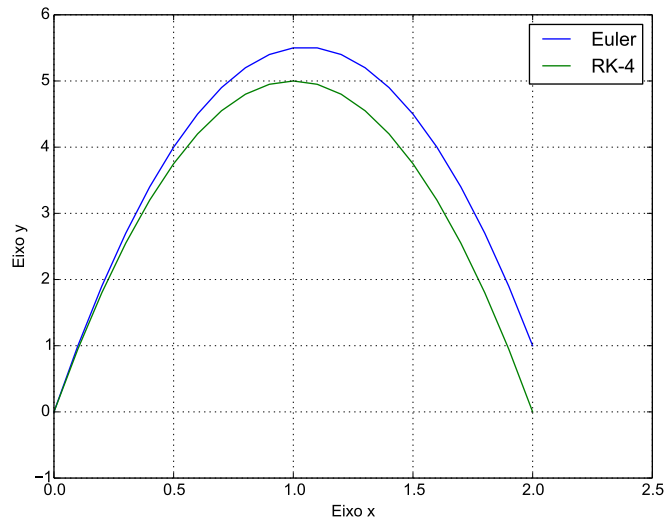


Figura 4.5: Gráfico do deslocamento em y pelo tempo com $h = 0.1$

5. Discussão

Durante o desenvolvimento deste projeto foi possível a realização de estudos em diferentes áreas.

Na área da física pode-se estudar mais sobre a história da gravitação clássica[4, 5], entendendo desta forma de onde vem as 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 álgebra, cálculo e geometria analítica importantes, dando assim uma aplicação fisicamente útil a conceitos previamente desenvolvidos.

Por fim houve também a criação de um programa em Python para a resolução de E.D.O's, tal programa exigiu um estudo tanto na linguagem em si quanto em métodos computacionais[9] e desenvolvimento de interfaces gráficas.

6. Cronograma

Este projeto possui a duração de 9 meses, de 01/09/2014 a 31/07/2015, sendo este apenas um relatório parcial. O projeto se encontra dentro do cronograma.

- **01/09/2014 a 30/10/2014** Estudo analítico do problema e revisão da literatura.
Seguiu como o planejado. Foi feita 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 com erros foi iniciado.
- **01/03/2015 a 30/04/2015** Estudo numérico das leis de Kepler. Elaboração do relatório parcial.
Termino dos estudos com erros e inicio do estudo de movimento balístico, afim de posteriormente progredir o estudo para órbitas. Ocorreu também a elaboração deste relatório.
- **01/05/2015 a 30/06/2015** Estudo numérico das órbitas no sistema solar. Visualização dos resultados.
- **01/07/2015 a 31/07/2015** Elaboração do relatório final.

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

A. Código

```
1  #!/usr/bin/env python
2  ##- coding: utf-8 -*-
3
4  from __future__ import print_function, division
5  from matplotlib.backends.backend_gtk import FigureCanvasGTK as FigureCanvas
6  from operator import sub
7  from matplotlib.figure import Figure
8  from matplotlib import cm
9  import numpy as np
10 from math import *
11 import matplotlib.pyplot as plt
12 import gtk, parser, sys
13
14 # Calcula a função passada (em string) nos pontos dados (em float)
15 def no_ponto(x,y,funcao):
16     code=parser.expr(funcao).compile()
17     x=float(x)
18     y=float(y)
19     return eval(code)
20
21 # Calcula a função passada através do método de euler
22 def euler(xi,yi,funcao,termino,passoi):
23     x=float(xi)
24     y=float(yi)
25     passo=float(passoi)
26
27     global listaYEulerA, listaYEulerB, listaYEulerC
28     listaYEulerA, listaYEulerB, listaYEulerC=[],[],[]
29
30     while x <= termino:
31         listaYEulerA.append(y)
32
33         y+=passo*no_ponto(x,y,funcao)
34         x+=passo
35
36     x=float(xi)
37     y=float(yi)
38     passo=float(passoi)*0.1
39
40     while x <= termino:
41         listaYEulerB.append(y)
42         y+=passo*no_ponto(x,y,funcao)
43         x+=passo
44
45     x=float(xi)
46     y=float(yi)
47     passo=float(passoi)*1.5
48
49     while x <= termino:
50         listaYEulerC.append(y)
51         y+=passo*no_ponto(x,y,funcao)
52         x+=passo
53
54 # Calcula a função passada através do método de runge-kutta de quarta ordem
55 def kutta(xi,yi,funcao,termino,passoi):
56     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(passo)*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 analítica (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 analítica (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 xrange(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:

```

```

243         global listaYEulerA, listaYEulerB, listaYEulerC
244         listaYEulerA, listaYEulerB, listaYEulerC=[], [], []
245     if kuttaCheckbox.get_active()==False:
246         global listaYKuttaA, listaYKuttaB, listaYKuttaC
247         listaYKuttaA, listaYKuttaB, listaYKuttaC=[], [], []
248     if analiticoCheckbox.get_active()==False:
249         global listaYAnalitica
250         listaYAnalitica=[]
251     graficoPrincipal.remove(graficoPrincipal.get_child())
252     grafico_principal(inicioFLOAT, fimFLOAT, passoFLOAT, listaYEulerA, listaYKuttaA, listaYAnaliticaA)
253     try:
254         try:
255             grafico2DErroXx.remove(grafico2DErroXx.get_child())
256         except:
257             pass
258         if eulerCheckbox.get_active()==True or kuttaCheckbox.get_active()==True:
259             graficos_errores(listaYEulerA, listaYEulerB, listaYEulerC, listaYKuttaA, listaYKuttaB, listaYKuttaC,
260             except:
261                 erro("Ocorreu um erro ao plotar o gráfico dos erros.")
262                 raise
263     except:
264         erro("Ocorreu um erro ao plotar o gráfico.")
265         raise
266 except:
267     erro("Ocorreu um erro ao executar os métodos numéricos.\nConfira seus dados.")
268     raise
269 if analiticoCheckbox.get_active()==True:
270     try:
271         analitico(inicioFLOAT, solucaoAnaliticaSTR, fimFLOAT, passoFLOAT)
272     except:
273         erro("Não foi possível fazer o cálculo analítico.\nConfira sua solução.")
274         raise
275
276 def grafico_principal(inicio, fim, passo, listaEuler=[], listaKutta=[], listaAnalitica=[], listaErroKutta=[], listaErroEuler=[])
277     construir_x(inicio, fim, passo)
278     grafico1=Figure(figsize=(8,6), dpi=72)
279     axis1=grafico1.add_subplot(111)
280     axis1.grid(True)
281     axis1.set_xlabel("Eixo x")
282     axis1.set_ylabel("Eixo y")
283
284     if eulerCheckbox.get_active()==True:
285         axis1.plot(listaXA, listaEuler, label="Euler", color="b")
286     if kuttaCheckbox.get_active()==True:
287         axis1.plot(listaXA, listaKutta, label="RK-4", color="g")
288     if analiticoCheckbox.get_active()==True:
289         axis1.plot(listaXA, listaAnalitica, label="Analitica", color="r")
290
291     axis1.legend(loc='best')
292
293     canvas1=FigureCanvas(grafico1)
294     canvas1.set_size_request(700,500)
295     graficoPrincipal.add_with_viewport(canvas1)
296     janelaPrincipal.show_all()
297
298 def graficos_errores(listaEulerA=[], listaEulerB=[], listaEulerC=[], listaKuttaA=[], listaKuttaB=[], listaKuttaC=[], listaErroEuler=[])
299     if analiticoCheckbox.get_active()==True:
300         #GRAFICO 2 INFERIOR DIREITO
301         grafico2=Figure(figsize=(8,6), dpi=72)
302         axis2=grafico2.add_subplot(111)
303         axis2.grid(True)
304         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     grafico2DErroX.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())) + ") = " + str(float(fim.get_text())) + ")")
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()

```