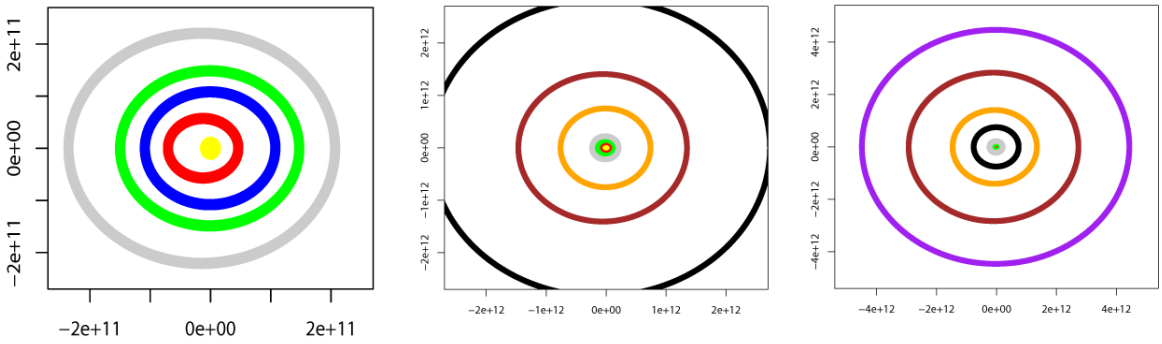
Título: Órbitas no Sistema Solar - visualização e comparação de modelos

PDPD: Michel Kluger

Orientadora: Prof. Dra. Cecilia Chirenti

Centro de Matemática, Computação e Cognição
Universidade Federal do ABC
2011



Profa. Dra. Cecilia Chirenti

Michel Klüger

Resumo

Este trabalho pretende modelar computacionalmente a atração gravitacional, fazendo uma comparação dos modelos de gravitação de Newton e da relatividade geral de Einstein, através da integração das equações de movimento para os dois casos.

Abstract

in this work we computational model the gravitional attraction, making a discussion of the Newton to Einstein transition in physics by means of the numerical integrations of the equations of motion.

Sumário

Lista de Figuras

1 Introdução

No contexto da gravitação, temos dois modelos distintos baseados em princípios teóricos diferentes, que fornecem previsões desejadas e corretas para apropriação do mundo pelo homem.

Isaac Newton, ao formular a sua lei da gravitação universal, podia quantificar as forças de interação gravitacional entre corpos celestes, mas não podia explicar a natureza dessa força, como expresso em suas própria palavras: *"até agora explicamos os fenômenos do Céu e do nosso mar por intermédio do poder da gravidade, mas não atribuímos nenhuma causa a este poder. É certo que deve proceder de uma causa que penetre até os próprios centros do sol e dos planetas... Mas até agora não pude descobrir as causas dessa propriedade da gravidade a partir dos fenômenos, e não forjo hipóteses."*

Em 1915, Albert Einstein muda com a relatividade geral o âmbito da questão,da força para o campo, um corpo com massa gera uma curvatura do espaço-tempo. Uma diferença marcante desses modelos é o intervalo de tempo para a ação da gravidade, para Newton é instantâneo($\Delta t = 0$) e para Einstein é transmitido com a velocidade da luz, e portanto $\Delta t = \frac{\Delta S}{c}$ que é pequeno mas não nulo. E é transmitida coincidentemente com o encurvar do espaço.

Segundo a gravitação universal de Newton, um corpo exerce sobre um outro uma força que é proporcional ao inverso do quadrado da distância, dada pela formula:

$$\vec{F}(r) = -\frac{GmM}{r^2}\hat{r} \tag{1}$$

Onde F é a força$[N]$, r $[m]$ é a distância entre os centros dos corpos, G $[m^3kg^{-1}s^{-2}]$ é a constante de gravitação universal, m e M massa dos corpos que estão interagindo $[kg]$ e $\hat{r}$ $[adim]$ é o versor direção da força, que é radial. como o raio cresce para fora do centro do corpo, adotamos a força negativa.

$$\vec{F} = m\vec{a} \tag{2}$$

igualando as equações temos que:

$$\vec{a} = -\frac{GM}{r^2} \tag{3}$$

Para Einstein um corpo que possui uma massa curva o espaço ao seu redor:.

$$\kappa \ = 16 \cdot G\pi c^{-2}\rho \tag{4}$$

Portanto a curvatura escalar do espaço é proporcional a densidade aparente ρ $[kg^1m^{-3}]$ do corpo. Isso se dá pois a matéria curva o espaço e a curvatura do espaço faz com que a matéria se mova.

Com correções relativisicas, a equação da aceleração fica:

$$\vec{a_{relativisic}} = \frac{GM}{r^2}(1 + \frac{12k^2}{c^2r^2}) \tag{5}$$

Uma consideração importante é que nessa configuração a costante de kepler K é expressa por $\frac{rv}{2}$.[1]

2 Objetivos

Esse projeto tem como objetivos:

- O melhor entendimento da gravidade, sendo possível uma manipulação das equações de força, campo e potencial gravitacional.
- Um aprofundamento do estudo de programação estudando métodos de tratar os dados gerados por um programa, de forma a gerar resultados e analisá-los.
- Complementar os estudos de cálculo e física do BCeT. Fazendo uso de integrais e de derivadas para análise de graficos, integrando equações diferenciais separaveis, e lineares de primeira e segunda ordem e começando o estudo da integrações numericas que só seria visto mais adiante no curso, na disciplina de cálculo numérico.

3 Métodos

Esse projeto de PDPD é pautado na revisão da literatura de conceitos físicos, e no estudo de métodos numéricos tais como Euler e Runge-Kutta, usados para integrar as equações de movimento.

3.1 Métodos numéricos

um método numérico é uma técnica para integrar uma equação, numa função no intervalo L $[L_0, L]$, esse intervalo é a soma das N partes da a serem integradas,distantes entre si por um avanço (passo) h.

Quanto menor o passo melhor a precisão do método menor seu erro.O erro E_x é expresso pela relação $E_x = X - \bar{X}$, onde X é o valor analítico e $\bar{X}$ a aproximação do feita pelo método,outra consideração importante é que como uma função matematica no computador é definida por uma série finita de termos, pois um algoritmo computacional precisa ter por definição um número finito de passos. Essa limitação dos termos considerados leva a um erro, o truncamento.

3.2 Linguagem utilizadas no projeto

3.2.1 c++

- Dev c++ - c++:
  O dev c++ é o programa utilizado para gerar e compilar os códigos dos algoritmos computacionais.Estes por definição são uma sequencia finita de comando encadeados, que transformam dados de entrada em uma saída desejada. No projeto ainda temos a parte onde os dados são externalizados para serem posteriormente processados gráficamente.

3.2.2 R

- R project - R: O R project é utilizado para ler os resultados gerados pela função ofstream no formato .txt e plotar o gráfico. Através do metodo, splicing, colocamos os dados das

colunas gerados pelo c++ em variaveis para plotar conforme o resultado que queremos obter.

3.2.3 Latex

- Miktex-latextila - LaTeX 2_ε: O Miktex-latextila estão sendo utilizados na hora de escrever o relatório, a opção por latex em detrimento da metodologia WYSIWYG (MS Word), foi feita por ter uma melhor adaptação ao modelo acadêmico e por ter boa compatibilidade com fórmulas matemáticas.

4 Desenvolvimento

4.1 Etapas iniciais

Conforme apresentado no relatório parcial, no início do projeto construímos o programa da força 3d [A.1], o programa se mostrava inadequado por ser estático em relação ao tempo, diferente do problema físico dos dois corpos que é dinâmico. pois a força gera num corpo com massa uma aceleração(2).O programa tambem não contemplava velocidade enquanto condição inicial.

Com base no livro Gravity from the Ground Up [1], e na apostila de c++ da Unesp[4], começamos a montar algumas simulações mais simples, baseadas em problemas classicos da mecânica Newtoniana envolvendo a força de atração da gravidade, modelamos a queda livre e de movimento balístico, Em paralelo estudamos como exportar os dados gerados no c++ para arquivos que poderiam ser armazenados na memória permante do computador. Para isso incluímos e utilizamos a biblioteca fstream.

4.1.1 Forças

O estudo da força gravitacional, se iniciou com o programa de força 3d. Nesse programa dois corpos são posicionados no espaço, o usuario entra com as cordenadas x,y,z e com a massa de cada um deles. Com a fórmula descrita pela equação (1), calcula-se o módulo da força.

4.1.2 Integração de equações

Na biblioteca cmath do c++ existem funções prontas definidas com um grau razoável de precisão, podemos comparar a função $y(x)$ com a função obtida a partir da integral de uma equação y'=f(x). A integração numérica foi realizada utilizando os métodos de Euler e R.K.

4.2 Orbit

4.2.1 Órbitas

Órbitas são trajetórias que um corpo percorre ao redor de outro sob a influência de alguma força, no caso das órbitas celestes é a força de atração gravitacional.A 1ª lei de Kepler diz que todos

os planetas movem-se em órbitas elípticas e que o Sol localiza-se em um dos focos da elipse descrita.

Elipse é o conjunto dos pontos P tais que a soma das distâncias de P a dois pontos fixos F1 e F2 (focos) é constante.

4.2.2 programa orbit

Um problema não contemplado pelo programa da força 3D é que a força gravitacional gera no corpo periférico um movimento, e com a mudança na distância, há uma variação na força consequentemente na aceleração. Para calcular a órbita o programa precisa atualizar as variáveis ao longo do tempo(ao longo da integração).

O programa orbit[6] do livro gravity from ground up[1] cobre esses requesitos descritos acima, e possue três pilares centrais, são eles: o passo *dt* variável, o método preditor-corretor e a análise angular da órbita.

- Passo dt variável:

  Ao integrar uma função é comum estabelecer um passo dt para integrá-la numericamente. Pois analiticamente quando o passo tende a 0 o erro também vai pra 0, e numericamente com um passo pequeno o suficiente podemos obter uma boa aproximação.

  A aceleração ao longo dos intervalos de tempo varia, visto que a posição na órbita vai variar, por se tratar de um elipse, caso não variasse teríamos uma situação confortável de trabalho,por ser um MRUV. Mas se avançarmos um passo dt pequeno o suficiente, a nova posição vai ser próxima o suficiente da inicial, considerando que um corpo de massa m tem sua aceleração relativa à posição, e esta quase não variou e a aceleração quase não vai mudar, e assim podemos considerar que o movimento é um MRUV naquele intervalo.

  Se a órbita fosse redonda, bastaria acertar o tamanho do passo de integração no começo, como a distância se mantém constante a aceleração seria constante e o passo poderia ser constante sem resultar numa variação de aceleração muito grande de x0 para x1 (de fato ela não varia),como as órbitas podem e geralmente são elípticas, o tamanho do passo que é bom no começo pode não ser bom ao longo da órbita que está sendo integrada, então para avançar um passo devemos testar novamente o tamanho do passo para ver se ele será conveniente para manter o MRUV , logo essa parte é feita cada vez que ele vai avançar de t para t + dt.

  Se a nova posição x_1 for próxima o suficiente de x_0, a aceleração quase não vai mudar e então o programa testa para ver se a aceleração variou pouco. Se for verdade, usa esse tamanho de dt1 mesmo. Se a aceleração variou muito, calcula tudo de novo com $\frac{dt}{2}$. Com o passo ajustado para metade do original a variação deve ser menor, tornando aquele movimento de aceleração variável mais parecido com um MRUV.

  As equações abaixo mostram em x, a ligação entre o avanço do passo, a mudança de posição e a variação da velocidade.

$$x_1 = x_0 + dx + ddx_0$$
$$dx = vdt_1$$
$$ddx_0 = \frac{dvdt_1}{2}$$

$$dv = ax_0 dt$$
$$ax_0 = -MGx_0/r^3$$
$$ax_1 = -GMx_1/r^3$$

- Passo preditor-corretor com o dt estabelecido,o programa recalcular melhor a nova posição e a nova aceleração em vez de usar a aceleração constante, usar a média das acelerações (supondo então que a derivada da aceleração é constante) nós comparamos agora ddx0 com ddx1 ddx0 nós tínhamos calculado supondo o MRU, enquanto ddx1 nós calculamos com a média das acelerações. Se ddx0 e ddx1 são bem próximos, então $x_0 = x_0 + dx + ddx_0$ é quase igual a $x_1 = x_0 + dx + ddx_1$, a acelaração ax_1 varia pouco então a nossa aproximação é boa,se $ddx0$ e ddx1 são muito diferentes, aí entra o preditor-corretor. usamos o novo $x_1 = x_0 + dx + ddx_1$ para recalcular ax1 e usamos o novo ax1 para fazer a média das acelerações e calcular o novo ddx1 que comparamos com o antigo. Quando a diferença entre o ddx_1 e o ddx_1 for pequena o suficiente, para a precisão eps2 nós saímos do passo do preditor-corretor e avançamos mais um passo no tempo

- Análise angular da órbita:

  Nossos valores de entrada são todos cartesianos e o corpo central é fixado na origem; para determinar qual o sentido que o corpo orbitante perfaz a órbita,A varaiavel meia órbita é importante para determinar o fim da integração, quando o corpo ja tiver completado a meia órbita e seu angulo com a origem for maior que o inicial no caso de órbita anti-horária e menor no caso horário a órbita terá terminado.quando a órbita fecha, o há uma quebra de looping no programa. Nesta etapa os dados coletados são exportados. Para obter o angulo apartir dos dados cartesianos há a implementação da função matemática-computacional Atan2(x,y) ver figura 1.

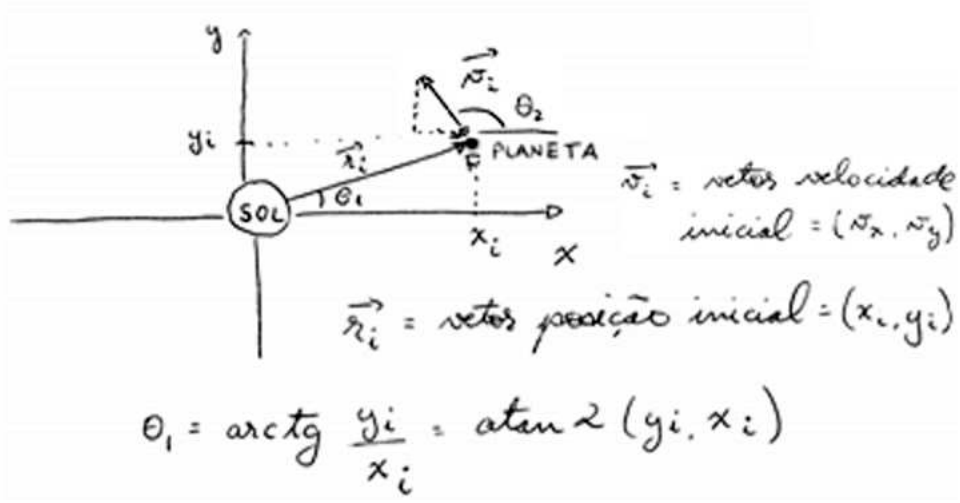


Figura 1: Órbita e aplicação da função Atan2(x,y).

- Extraindo dados do periélio:

  Periélio é o ponto da órbita de um corpo que está mais próximo do Sol. Quando um corpo se encontra no periélio, ele tem a maior velocidade de toda a sua órbita.

  Entrando com a velocidade no periélio e a distancia deste ponto ao sol podemos calcular toda órbita do planeta, a velocidade no periélio poderia ter componentes x e y e suas distancia tambem ter componentes x e y, mas como o sistema solar não está fixo num plano cartsiano tal que podemos posicionar convenientemente o periélio em x é distancia do sol ao periélio da órbita e $y = 0$, como o vetor posição e velocidade tem produto escalar igual a 0.(i.e. são perpendiculares), a velocidade x vale 0, assim tempos $v_y = velocidadenoperielio$.

5 Resultados

5.1 Parametros gerais

uma órbita conforme definido anteriormente é uma tragetória gerada por uma força, portanto um corpo na sua órbita tem alguns paramtros que variam ao longo do tempo, variaveis importantes como energia,posição e velocidade. A figura 2 mostra esses parametros para a órbita de mercurio.

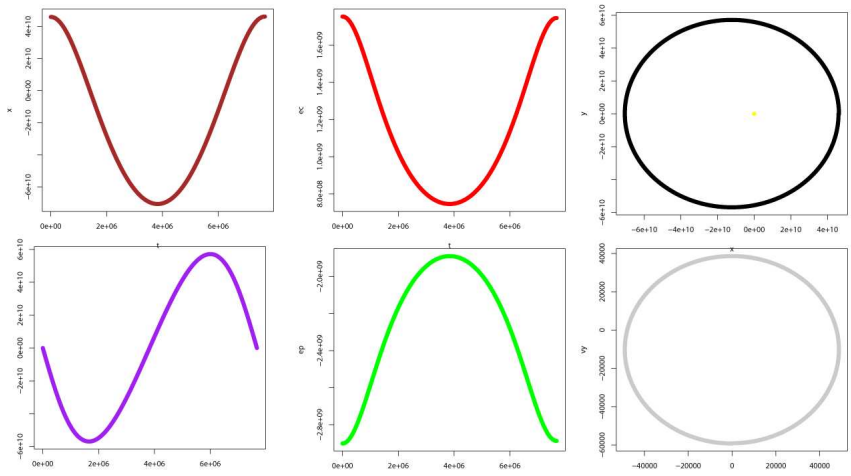


Figura 2: marrom XxT, roxo YxT,vermelho EcxT, verde EpxT, preto XxY,cinza VXxVY

5.2 Verficação da primeira lei de kepler

A elipse obedece a equação $\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$,

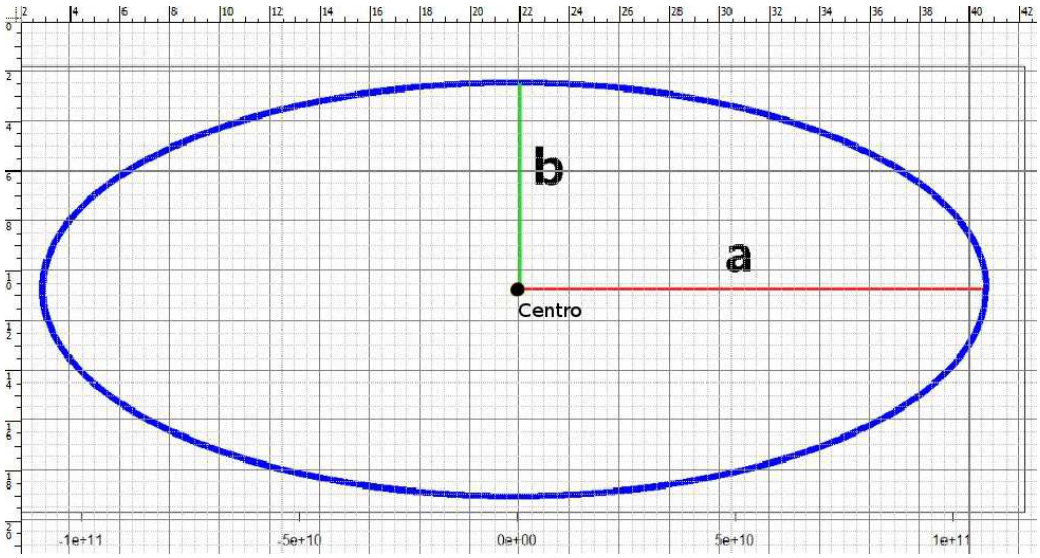


Figura 3: órbita de mercurio plotada sobre escala milimetrada. apesar da excentricidade parecer grande ela vale 0.206

Da figura 3 temos $A = 1.13e + 11$ e $B = 5e + 10$, usando a ferramenta wolframalpha.com plotamos a elipse com os pametros obtidos e novamente obtivemos uma elipse, apesar de alguns parametros da elipse não serem iguais, como a excentricidade, a figura 4 ficou compativel com uma órbita, essa difrença entre as figuras pode ser atribuida a leitura dos dados em papel milimetrado.

5.3 Tipos de órbitas

A energia total é a soma da energia cinética e da potencial, as equações 6 e 7 mostram essa relação.

$$E_t = E_c + E_{pot} \tag{6}$$

$$E_t = \frac{mv^2}{2} - \frac{MGm}{r} \tag{7}$$

quando a energia total é 0,temos:

$$\frac{mv^2}{2} = +\frac{MGm}{r} \tag{8}$$

isolando v temos:

$$v = \sqrt{\frac{2MG}{r}} \tag{9}$$

assim fixando r=R para a posição inicial, temos um valor de v_0 no qual a energia é 0 e para esse valor obtemos uma órbita parabólica, acima desse valor a órbita é hiperbólica, abaixo elíptica. os tipos de órbita podem ser vistos na figura 5.

Assim v_0 é:

$$v = \sqrt{\frac{2 * 6.67300e-11 * 2e30}{R}} \tag{10}$$

tomando R da órbita de mercurio que é de 46e9 metros. obtemos como velocidade inicial para um órbita parabólica 76157.73105 metros por segundo.

5.4 Conservação de energia

Conforme visto na seção anterior órbitas tem variação de energia igual ao 0, a figura 6 mostra a energia potencial,cinética e total.

5.5 Lei dos Periodos

A terceira lei de kepler diz que o quadrado do periodo de um corpo orbitante é proporcional ao cubo do eixo maior de sua órbita.

$$\frac{T_1^2}{r_1^3} = \frac{T_2^2}{r_2^3} = \frac{4\pi^2}{MG} \tag{11}$$

Da figura 7 e dos dados gerados temos periodo de mercurio sendo 7.6e6 e o de netuno 5.19e9. assim temos:

$$\frac{7.6e6^2}{46e9^3} = \frac{5.19e9^2}{4487e12^3} = \frac{4\pi^2}{2e30 * 6.673e-11} \tag{12}$$

todos os resultados estão entre 2.9e-19 e 3.1e-19, o que é uma boa faixa de precisão, para nosso métodos.

5.6 RelativisticOrbit

O Relativistic órbit[8] diferentemente do órbit não foi traduzido para C++, apenas para poder comparar os modelos, rodamos o programa direto de seu ambiente nativo o **triana**[7]. Devido as correções relativisticas, a órbita não é mais fechada, observamos a precessão do periélio e o sistema perde energia via emissão de ondas gravitacionais. Podemos ver isso na figura 8.

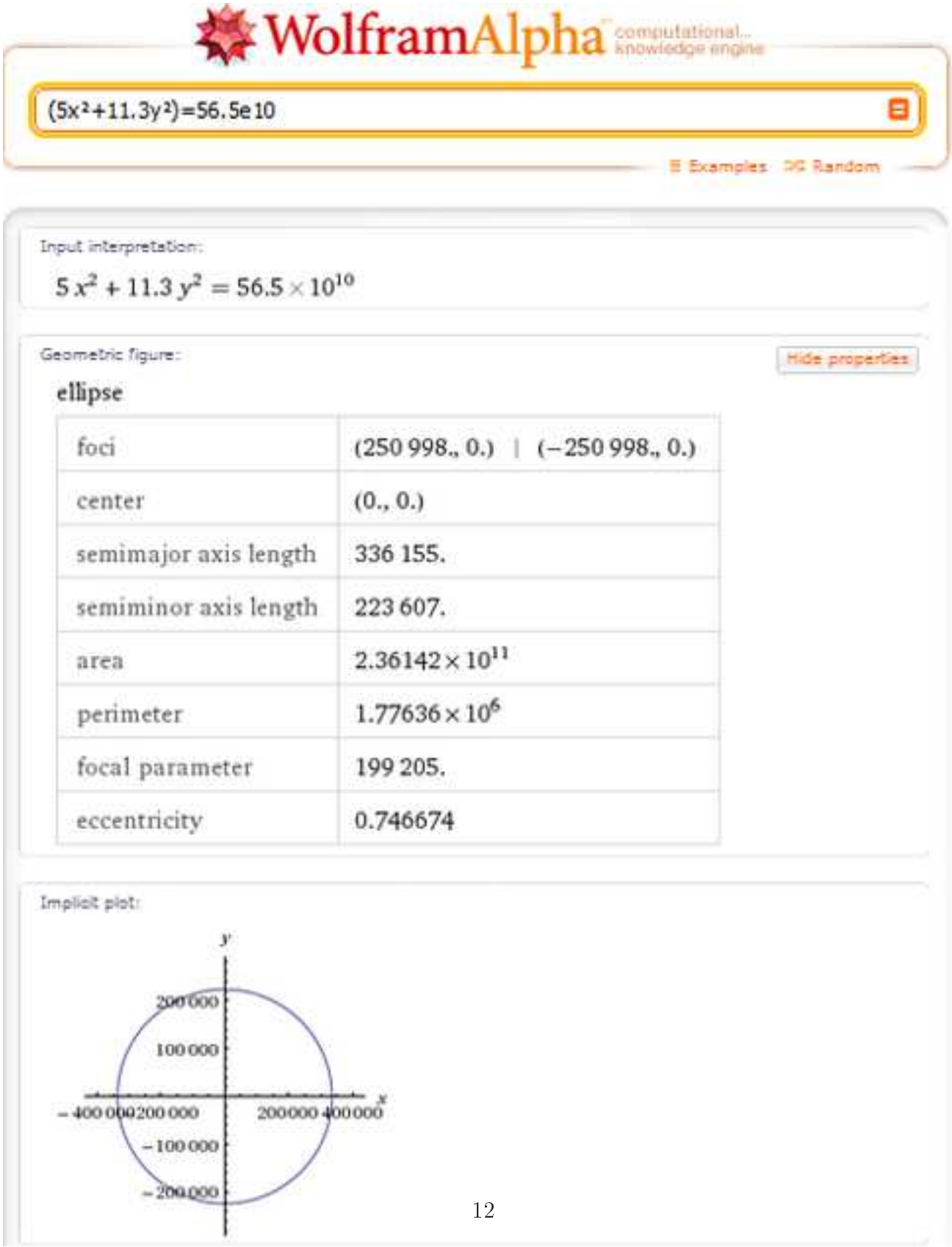


Figura 4: órbita de mercurio plotada com os parametros obtidos

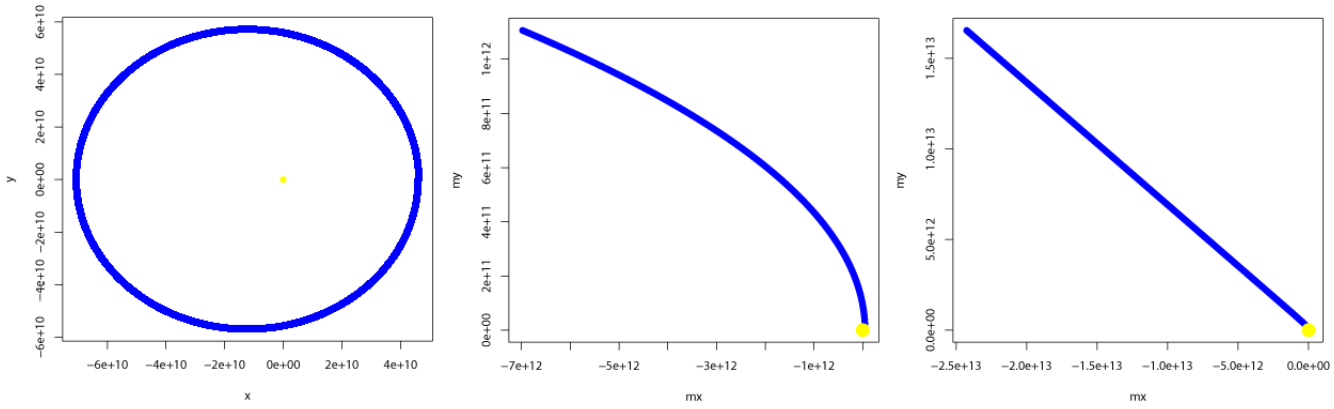


Figura 5: da esquerda para direita:órbita elíptica,parabólica e hiperbólica

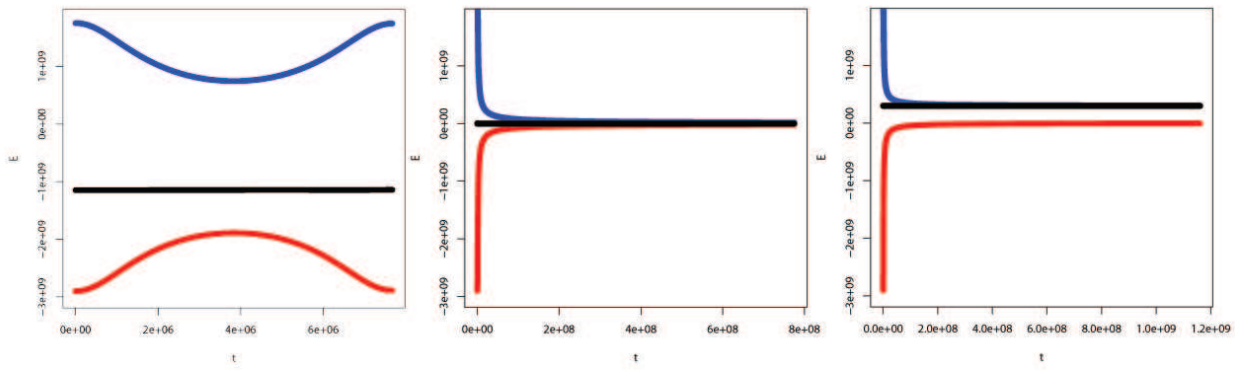


Figura 6: em vermelho energia potencial gravitacional,em azul energia cinética e em preto a energia total.da esquerda para direita:órbita elíptica,parabólica e hiperbólica

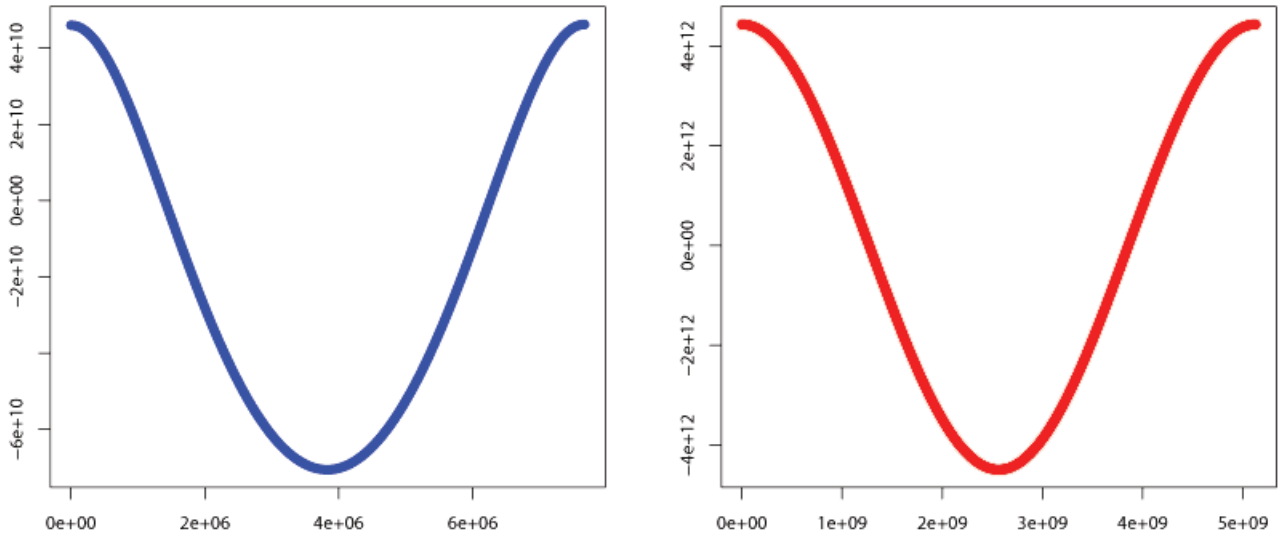


Figura 7: em azul o periodo de mercurio em vermelho o de netuno

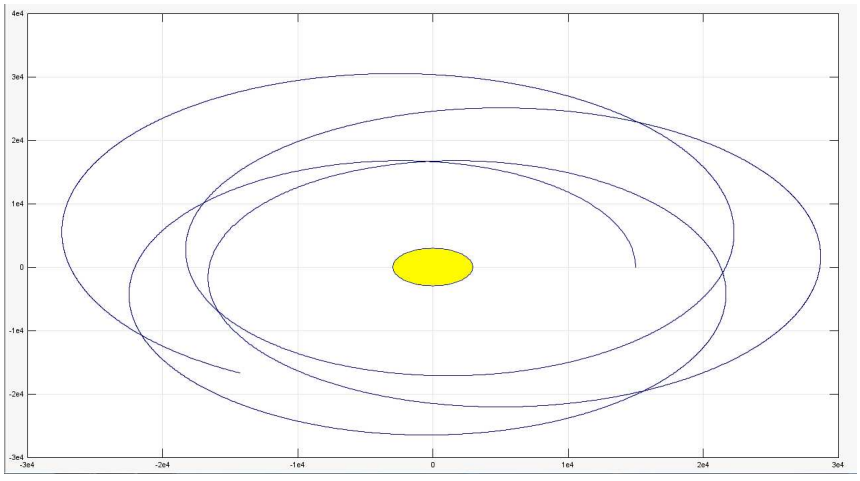


Figura 8: orbita de mercurio com correções relativisticass.

6 Cronograma

Este projeto possuiu a duração de 10 meses de 01/10/2010 a 31/07/2011. No começo do projeto foi proposto o seguinte cronograma:

01/10/2010 a 31/11/2010 Estudo analítico do problema e revisão da literatura.

Aconteceu conforme o previsto, houve o estudo de alguns problemas gravitacionais [1], o problema da aceleração anômala na Pioneer, houve um estudo de problemas de física Newtoniana fazendo o uso de algoritmos de programação para gerar soluções numéricas em tabelas.

01/12/2010 a 31/01/2009 Estudo dos métodos numéricos e elaboração do programa para integração das órbitas.

Nessa etapa começamos a estudar o programa gráfico para plotar os resultados obtido nas tabelas,houve um aprofundamento na teoria da relatividade geral e devido a ferias houve um pequeno atraso no cronograma.

01/02/2009 a 31/03/2010 Testes numéricos e interpretação dos resultados. Elaboração do Relatório Parcial.

Nessa etapa foram estudados os métodos de Euler e Runge-Kutta estudou-se a diferença de precisão dos métodos e como variando o numero e tamanho de passos as curvas alteravam sua precisão. Houve em paralelo com a disciplina IEDO, um estudo de resoluções das equações diferencias separáveis e o relatório parcial foi elaborado.

01/04/2010 a 31/05/2010 estudamos e implementação do programa orbit,gerando e plotando dados, estudamos os princípios e correção da relativiade geral.

01/06/2010 a 31/07/2010 Extraimos parametros das órbitas apartir do gráficos, como periodo e propiedades da elipse e geramos gráficos do sistema solar e órbitas relativisticas. Elaboramos do Relatório Final.

7 Conclusões

Concluimos que órbitas que utilizam as teorias de newton podem ser hiperbólicas,parabólicas e elípticas, as órbitas mantem sua energia total constante, por se tratar de um sistema fechado. órbitas planetarias são elipticas conforme enunciado na primeira lei de kepler. As órbitas relativisticas não são fechadas e não conservam energia. Também foi possível verificar numericamente a terceira lei de Kepler. Aprendemos durante a elaboração do projeto que funções não analíticas, podem ser integradas através de métodos numéricos e esse resultado apresenta um desvio da realidade, o tamanho desse desvio é determinado pela precisão imposta (eps). O passo de integração que para funções matemáticas é relativamente constante precisa ser constantemente revisto para que o movimento se mantenha uniformemente acelerado para poder usar as equações da cinemática e dinâmica newtoniana.

Referências

[1] B. Schutz, Gravity from the Ground Up, Cambridge Univeristy Press, Cambridge (2003).

[2] H. Moyses Nussenzveig, Curso de Física Básica vol. 1 Mecânica, Edgard Blücher, São Paulo (2002).

[3] C++: como programar/H.M. Deitel e P.J. Deitel trad. Carlos Arthur Lang Lisbôa e Maria Lúcia Lang Lisbôa. - 3.ed. - Porto Alegre : Bookman, 2001.

[4] T. Enrique Camargo,Apostila da Linguagem de Programação C++. Unesp Ilha Solteira. Julho de 2007. http://www.dee.feis.unesp.br/graduacao/disciplinas /langcpp/download/Apostila_C++.pdf

[5] Prosper,Harrison,Introduction to General Relativity,1998.http://www.physics.fsu.edu/courses /spring98/ast3033/Relativity/GeneralRelativity.htm

[6] http://www.gravityfromthegroundup.org/programs/Orbit.html

[7] http://www.gravityfromthegroundup.org/programs/UsingTriana.html

[8] http://www.gravityfromthegroundup.org/programs/RelativisticOrbit.html

A Códigos

A.1 Cálculo de força3d

```
#include <cstdlib>
#include <iostream>
#include<cmath>
#include <fstream>


using namespace std;

int main(int argc, char *argv[])
{
ofstream myfile;
myfile.open("newfile.xls");
const double G=6.67300*pow(10.0,-11.0);
double x1,x2,y1,y2,z1,z2,f,m1,m2,d;
cout<< " programa de calculo de distancia num plano triortonormal \n";
cout<< "insira o valor da cordenada x do ponto 1 \n";
cin>>x1;
cout<< "insira o valor da cordenada x do ponto 2 \n";
cin>>x2;
cout<< "insira o valor da cordenada y do ponto 1 \n";
cin>>y1;
cout<< "insira o valor da cordenada y do ponto 2 \n";
cin>>y2;
cout<< "insira o valor da cordenada z do ponto 1 \n";
cin>>z1;
cout<< "insira o valor da cordenada z do ponto 2 \n";
cin>>z2;
cout<< "insira o valor da massa 1 \n";
cin>>m1;
cout<< "insira o valor da massa 2 \n";
cin>>m2;
d=sqrt(pow(x1-x2,2)+pow(y1-y2,2)+pow(z1-z2,2));
cout<< "a distancia dos pontos vale: \n" <<d<< endl;
f=fabs((G*m1*m2)/(d*d));
cout<< "a atracao gravitacional tem modulo: \n" <<f<< endl;
   myfile<<d<<endl;
myfile<<m1<<endl;
myfile<<m2<<endl;
myfile<<f<<endl;
myfile.close();
```

```
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

A.2 Orbit

```
#include<iostream>
#include<cstdlib>
#include<cmath>
#include<fstream>
#include<sstream>
#include<string>
#include<iomanip>

using namespace std;

int main(int argc, char *argv[])
{

  //variaveis de entrada
  double vx0,vy0,M,dt,t,xi,yi,eps1,eps2;
  int maxpas;
  //contadores de looping
  int j,k,p,pp;
  //variaveis de apoio
  double testepredic, anguloagora;
  //variaveis utilizadas nos calculos
  double ax0,ay0,vx,vy,x0,y0,dt1,r,r3,x1, y1, ax1, ay1, dvx, dvy, dx, dy, ddx0, ddy0, dd
  //vetores alocados como pointer
  double* xcoordenadas;
  double* ycoordenadas;
  double* xvelocidade;
  double* yvelocidade;
  double* energiapotencial;
  double* energiacinetica;
```

```
double* tempo;
//constantes
const double G = 6.6726e-11;
//boleanas
bool antihorario,orbitacompleta,meiaorbita;


//Program code
ofstream myfile1;
myfile1.open("orbit.txt",ios::trunc);

cout<<"\nPROGRAMA ORBIT\n\n";

cout<<"Parametros:\n\n";

//Definicao dos parametros interna
M = 2.0e30;
cout<<"M = "<<M<<" (kg)"<<"\n";
xi = 4.6e10;
cout<<"xi = "<<xi<<" (m)"<<"\n";
yi = 0.0;
cout<<"yi = "<<yi<<" (m)"<<"\n";
vx0 = 0.0;
cout<<"vx0 = "<<vx0<<" (m/s)"<<"\n";
vy0 = -59220.0;
cout<<"vy0 = "<<vy0<<" (m/s)"<<"\n";
dt = 1000.0;
cout<<"dt = "<<dt<<" (s)"<<"\n";
maxpas = 10000;
cout<<"maxpas = "<<maxpas<<"\n";
eps1 = 0.05;
cout<<"eps1 = "<<eps1<<"\n";
eps2 = 1.0e-4;
cout<<"eps2 = "<<eps2<<"\n";

//Definicao dos parametros pelo usuario

//cout<<"entre com o valor da massa (kg) do corpo central\n";
//cin>>M;
//cout<<"entre com o valor da cordenada x (m) do corpo em orbita\n";
//cin>>xi;
//cout<<"entre com o valor da cordenada y (m) do corpo em orbita\n";
//cin>>yi;
```

```
//cout<<"entre com o valor da velocidade x (m/s) do corpo em orbita\n";
//cin>>vx0;
//cout<<"entre com o valor da velocidade y (m/s) do corpo em orbita\n";
//cin>>vy0;
//cout<<"entre com o valor do tamanho do passo de integraÃ§Ã£o dt (s) da orbita\n";
//cin>>dt;
//cout<<"entre com o valor do numero maximo de passos\n";
//cin>>maxpas;
//cout<<"entre com o valor de eps1\n";
//cin>>eps1;
//cout<<"entre com o valor de eps2\n";
//cin>>eps2;


//declaracoes e formulas

xcoordenadas = new double[maxpas];
ycoordenadas = new double[maxpas];
xvelocidade = new double[maxpas];
yvelocidade = new double[maxpas];
energiapotencial = new double[maxpas];
energiacinetica = new double[maxpas];
tempo = new double[maxpas];

t  = 0.0;
dt1 = dt;
vx = vx0;
vy = vy0;
x0 = xi;
y0 = yi;
r = sqrt(pow(x0,2)+pow(y0,2));
r3 = pow(r,3);
ax0 = -M*G*x0/r3;
ay0 = -M*G*y0/r3;

xcoordenadas[0] = x0;
ycoordenadas[0] = y0;
xvelocidade[0] = vx0;
yvelocidade[0] = vy0;
energiapotencial[0] = -G*M/r;
energiacinetica[0] = 0.5*(pow(vx,2) + pow(vy,2));

anguloposi = atan2(yi,xi);
```

```
anguloveli = atan2(vy,vx);
difangulo = anguloveli - anguloposi;

cout<<"\n\nDetermina o sentido horario ou antihorario da orbita\n\n";
cout<<"anguloposi = "<<anguloposi<<"\n";
cout<<"anguloveli = "<<anguloveli<<"\n";
cout<<"difangulo = "<<difangulo<<"\n";

//determina o sentido horario ou anti horario na orbita

if ( difangulo > M_PI )
  difangulo = difangulo - 2*M_PI;

//        else if (difangulo < difangulo - 2*M_PI)
else if ( difangulo < -M_PI )
  difangulo = difangulo + 2.0*M_PI;

cout<<"difangulo corrigido = "<<difangulo<<"\n";

antihorario = (difangulo > 0.0 );
orbitacompleta = false;
meiaorbita = false;

if ( antihorario == true )
  cout<<"\nOrbita no sentido antihorario\n";
else if( antihorario == false )
  cout<<"\nOrbita no sentido horario\n";

cout<<"\n\nComeca a integracao\n";


//para terminar a integracao apos uma orbita completa:
for ( j = 1; ( !orbitacompleta && ( j < maxpas )); j++ ) {
//para fazer a integracao ate maxpas:
//for ( j = 1; j < maxpas; j++ ) {

  dvx = ax0*dt1;
  dvy = ay0*dt1;
  dx = vx*dt1;
  dy = vy*dt1;
  ddx0 = dvx/2.0*dt1;
  ddy0 = dvy/2.0*dt1;
  ddx1 = ddx0;
```

```
    ddy1 = ddy0;

    x1 = x0 + dx + ddx0;
    y1 = y0 + dy + ddy0;
    r = sqrt( pow(x1,2)+pow(y1,2));
    r3 = pow(r,3);
    ax1 = -G*M*x1/r3;
    ay1 = -G*M*y1/r3;

    //cout<<"teste: "<<(fabs(ax1-ax0) + fabs(ay1-ay0))/(fabs(ax0) + fabs(ay0))<<"\n";

    if (fabs(ax1-ax0) + fabs(ay1-ay0) > eps1*(fabs(ax0) + fabs(ay0))){
      dt1=dt1/2.0;
      cout<<"diminui dt: dt1 = "<<dt1<<"\n";
      j--;
    }
    else{
      testepredic = fabs(ddx0) + fabs(ddy0);
      for ( k = 0; k < 10; k++ ) {
dvx = (ax0 + ax1)/2.0*dt1;
dvy = (ay0 + ay1)/2.0*dt1;
ddx1 = dvx/2.0*dt1;
ddy1 = dvy/2.0*dt1;

//cout<<"testepredic: "<<k<<"\t"<<(fabs(ddx1-ddx0) + fabs(ddy1-ddy0))/testepredic<<"\n";

if ( fabs(ddx1-ddx0) + fabs(ddy1-ddy0) > eps2 * testepredic){
  ddx0 = ddx1;
  ddy0 = ddy1;
  x1 = x0 + dx + ddx0;
  y1 = y0 + dx + ddy0;
  r = sqrt(pow(x1,2)+pow(y1,2));
  r3 = pow(r,3);
  //ax1 = G*M*x1/pow(r,3);
  ax1 = -G*M*x1/pow(r,3);
  ay1 = -G*M*y1/pow(r,3);
}
else break;
      }

      t=t+dt1;
      x0 = x0 + dx + ddx1;
      y0 =y0+ dy + ddy1;
```

```
      ax0 = ax1;
      ay0 = ay1;
      vx = vx + dvx;
      vy = vy + dvy;
      xcoordenadas[j] = x0;
      ycoordenadas[j] = y0;
      xvelocidade[j] = vx;
      yvelocidade[j] = vy;
      r = sqrt( pow(x0,2) + pow(y0,2));
      energiapotencial[j] = -G*M/r;
      energiacinetica[j] = 0.5*(pow(vx,2) + pow(vy,2));
      tempo[j] = t;


      anguloagora = atan2(y0,x0);
      difangulo = anguloagora - anguloposi;
      if (difangulo > M_PI) difangulo = difangulo - 2*M_PI;
      else if (difangulo < -M_PI) difangulo = difangulo + 2*M_PI;
      if (!meiaorbita)
      {
if (antihorario) meiaorbita = (difangulo < 0);
else meiaorbita = (difangulo > 0);
      }
      else {
if (antihorario) orbitacompleta = (difangulo > 0);
else orbitacompleta = (difangulo < 0);
      }
    }

  }

  cout<<"\n\n\nIntegracao terminada.\n\n";


  for (p=0;p<j;p++){
    myfile1 << tempo[p] << "\t"<<xcoordenadas[p] << "\t"<<ycoordenadas[p] << "\t" << xve
  }

  for(pp=0;pp<j/50;pp++){
    ostringstream filename;
    filename << "dados/file_" << setfill('0') << setw(4) << pp << ".txt";
    ofstream mySaveFile;
    mySaveFile.open(filename.str().c_str(),ios::trunc);
```

```
    mySaveFile<<tempo[pp*50]<<"\t"<<xcoordenadas[pp*50]<<"\t"<<ycoordenadas[pp*50]<<"\n"
    mySaveFile.close();
  }

  delete [] xcoordenadas;
  delete [] ycoordenadas;
  delete [] xvelocidade;
  delete [] yvelocidade;
  delete [] energiapotencial;
  delete [] energiacinetica;
  delete [] tempo;

  myfile1.close();

  //system("PAUSE");
  return EXIT_SUCCESS;
}
```

A.3 Plots de mercurio

```
script R
#jogo da velha é o no comentario no R.
#selecionar diretorio.
setwd(C:\Users\***\Documents")
initial.dir=getwd()

#leitura dos arquivos
orbit=read.table("orbit.txt")
mt=orbit[,1]
mx=orbit[,2]
my=orbit[,3]
mvx=orbit[,4]
mvy=orbit[,5]
mep=orbit[,6]
mec=orbit[,7]

#plot do arquivo 1, com parametros de controle do grÃ¡fico.
plot(mt,mx,col="brown")
plot(mt,my,col="purple")
plot(mx,my,col="black")
points(0,0,col="yellow",pch=19)
plot(mvx,mvy,col="grey")
plot(mt,mec,col="blue",ylim=range(-3e09,+1.8e09),ylab="")
```

```
par(new=TRUE)
plot(mt,mep,axes=FALSE,col="red",ylim=range(-3e09,+1.8e09),ylab="E")
par(new=TRUE)
plot(mt,mep+mec,axes=FALSE,col="black",ylim=range(-3e09,+1.8e09),ylab="")
```

A.4 Sistema solar

```
#script R
#jogo da velha e o comentario no R.
#selecionar diretorio.
setwd(C:\Users\Duda\Documents")
initial.dir=getwd()

#leitura dos arquivos
orbitademercurio=read.table("orbitademercurio.txt")
#mt=orbitademercurio[,1]
mx=orbitademercurio[,2]
my=orbitademercurio[,3]
#mvx=orbitademercurio[,4]
#mvy=orbitademercurio[,5]
#mep=orbitademercurio[,6]
#mec=orbitademercurio[,7]

#leitura dos arquivos
orbitadevenus=read.table("orbitadevenus.txt")
#vt=orbitadevenus[,1]
vx=orbitadevenus[,2]
vy=orbitadevenus[,3]
#vvx=orbitadevenus[,4]
#vvy=orbitadevenus[,5]
#vep=orbitadevenus[,6]
#vec=orbitadevenus[,7]


#leitura dos arquivos
orbitadaterra=read.table("orbitadaterra.txt")
#tt=orbitadaterra[,1]
tx=orbitadaterra[,2]
ty=orbitadaterra[,3]
#tvx=orbitadaterra[,4]
#tvy=orbitadaterra[,5]
#tep=orbitadaterra[,6]
#tec=orbitadaterra[,7]
```

```
#leitura dos arquivos
orbitademarte=read.table("orbitademarte.txt")
#mat=orbitademarte[,1]
max=orbitademarte[,2]
may=orbitademarte[,3]
#mavx=orbitademarte[,4]
#mavy=orbitademarte[,5]
#maep=orbitademarte[,6]
#maec=orbitademarte[,7]

#leitura dos arquivos
orbitadejupiter=read.table("orbitadejupiter.txt")
#jt=orbitadejupiter[,1]
jx=orbitadejupiter[,2]
jy=orbitadejupiter[,3]
#jvx=orbitadejupiter[,4]
#jvy=orbitadejupiter[,5]
#jep=orbitadejupiter[,6]
#jec=orbitadejupiter[,7]

#leitura dos arquivos
orbitadesaturno=read.table("orbitadesaturno.txt")
#st=orbitadesaturno[,1]
sx=orbitadesaturno[,2]
sy=orbitadesaturno[,3]
#svx=orbitadesaturno[,4]
#svy=orbitadesaturno[,5]
#sep=orbitadesaturno[,6]
#sec=orbitadesaturno[,7]

#leitura dos arquivos
orbitadeurano=read.table("orbitadeurano.txt")
#ut=orbitadeurano[,1]
ux=orbitadeurano[,2]
uy=orbitadeurano[,3]
#uvx=orbitadeurano[,4]
#uvy=orbitadeurano[,5]
#uep=orbitadeurano[,6]
#uec=orbitadeurano[,7]

#leitura dos arquivos
orbitadeurano=read.table("orbitadeurano.txt")
```

```
#ut=orbitadeurano[,1]
ux=orbitadeurano[,2]
uy=orbitadeurano[,3]
#uvx=orbitadeurano[,4]
#uvy=orbitadeurano[,5]
#uep=orbitadeurano[,6]
#uec=orbitadeurano[,7]

#leitura dos arquivos
orbitadenetuno=read.table("orbitadenetuno.txt")
#nt=orbitadenetuno[,1]
nx=orbitadenetuno[,2]
ny=orbitadenetuno[,3]
#nvx=orbitadenetuno[,4]
#nvy=orbitadenetuno[,5]
#nep=orbitadenetuno[,6]
#nec=orbitadenetuno[,7]

#plots

plot(mx,my,col="blue",ylim=range(-2.5e+13,2.5e+13),xlim=range(-2.5e+13,2.5e+13))
points(0,0,col="yellow",pch=19,lwd="10")
par(new=TRUE)
plot(vx,vy,col="red",axes=FALSE,xlab="",ylab="",xlim=range(-2.5e+13,2.5e+13),ylim=range(
par(new=TRUE)
plot(tx,ty,col="green",axes=FALSE,xlab="",ylab="",xlim=range(-2.5e+13,2.5e+13),ylim=rang
par(new=TRUE)
plot(max,may,col="grey",axes=FALSE,xlab="",ylab="",xlim=range(-2.5e+13,2.5e+13),ylim=ran
par(new=TRUE)
plot(jx,jy,col="black",axes=FALSE,xlab="",ylab="",xlim=range(-2.5e+13,2.5e+13),ylim=rang
par(new=TRUE)
plot(sx,sy,col="orange",axes=FALSE,xlab="",ylab="",xlim=range(-2.5e+13,2.5e+13),ylim=ran
par(new=TRUE)
plot(ux,uy,col="brown",axes=FALSE,xlab="",ylab="",xlim=range(-2.5e+13,2.5e+13),ylim=rang
par(new=TRUE)
plot(nx,ny,col="purple",axes=FALSE,xlab="",ylab="",xlim=range(-2.5e+13,2.5e+13),ylim=ran
```