Projeto 5 - Leis de Kepler e o Problema de Três Corpos

Instituto de Física de São Carlos
Universidade de São Paulo

Vinícius Bastos Marcos (12556715)

Introdução à Física Computacional
Prof. Francisco Castilho Alcaraz

Dezembro, 2022

Tarefa A

A tarefa A introduz, basicamente, todo ferramental matemático que será usado em todo o projeto. Nela temos o problema de simular a órbita entre dois corpos, no caso uma estrela, nosso Sol, e um planeta, a Terra.

Para isso, usaremos o Método de Verlet, explanado abaixo:

$$t_i = i \cdot \Delta t \rightarrow y(t_i) = y_i$$

e

$$y(t_i \pm \Delta t) = y(t_i) \pm \frac{dy}{dt}\,|_{t_i} \cdot (\Delta t) + \frac{1}{2}\frac{d^2y}{dt^2}\,|_{t_i} \cdot (\Delta t)^2 \pm \frac{1}{6}\frac{d^3y}{dt^3}\,|_{t_i} \cdot (\Delta t)^3 + O((\Delta t)^4)$$

Somando o y_{i-1} e y_{i+1} e reorganizando os termos, temos

$$y_{i+1} = 2y_i - y_{i-1} + \frac{d^2y}{dt^2}\,|_{t_i} \cdot (\Delta t)^2 + O((\Delta t)^4) \tag{1}$$

Mas sabemos que

$$\frac{d^2y}{dt^2} = \frac{F_{Gy}}{M_P} = -\frac{GM_S y}{r^3}$$

em que

$$r = \sqrt{(x_P - x_S)^2 + (y_P - y_S)^2}$$

Mas, percebemos que haverá a necessidade de se calcular o y_1 por meio de outro método antes. Assim, lembrando o projeto anterior, usarei o método de Euler-Cromer.

$$y_1 = y_0 + v_{0y}\Delta t \tag{2}$$

Tudo que foi exposto acima vale para x.

Mas a tarefa pede que mudemos o Δt para verificar o efeito. Assim, escolhi três: o equivalente a um dia terrestre (1/365) em anos, um menor, sendo 0,000001 ano, e outro maior, sendo 0,05 ano. O código implementado está disposto logo abaixo

```
c     tarefa a
      implicit real*8 (a-h, o-z)
      parameter(pi = dacos(-1.d0))
      parameter(dmedia = 1.d0) !distância média que vou alterando em cada
 ↪   caso

c     abrindo arquivo de saída
      iout = 10
      open(iout, FILE='saida-a-certa.dat')

      deltaT = (1.d0/365.d0) !um dia em anos
      GMs = 4 * (pi**2) !em unidades astronômicas
      v0x = 0.d0
      v0y = (2.d0*pi)/(dsqrt(dmedia))
      x0 = dmedia
      y0 = 0.d0
      n = 365 * 3 !tempo de simulação em dias terrestres
```

```fortran

c     método de Euler para x1 e y1
      x1 = x0 + v0x * deltaT
      y1 = y0 + v0y * deltaT
      write(iout,*) x0, y0
      write(iout,*) x1, y1

c     método de Verlet
      do i = 2, n
          r = dsqrt(x1**2 + y1**2)
          !na componente x
          Fx = ((-1)*GMs*x1)/(r**3)
          x2 = 2*x1 - x0 + Fx*(deltaT**2)
          x0 = x1
          x1 = x2
          !na componente y
          Fy = ((-1)*GMs*y1)/(r**3)
          y2 = 2*y1 - y0 + Fy*(deltaT**2)
          y0 = y1
          y1 = y2

          write(iout, *) x2, y2
      end do

      close(iout)

      end
```

Algoritmo 1: código para resolução da tarefa A

Assim, chegamos nas seguintes órbitas, dispostas juntas na figura abaixo. Como a v_{0y} escolhida foi a necessária para que a órbita seja circular (explicarei como chegar nela na próxima tarefa), temos que a melhor escolha para Δt sendo um dia em anos.

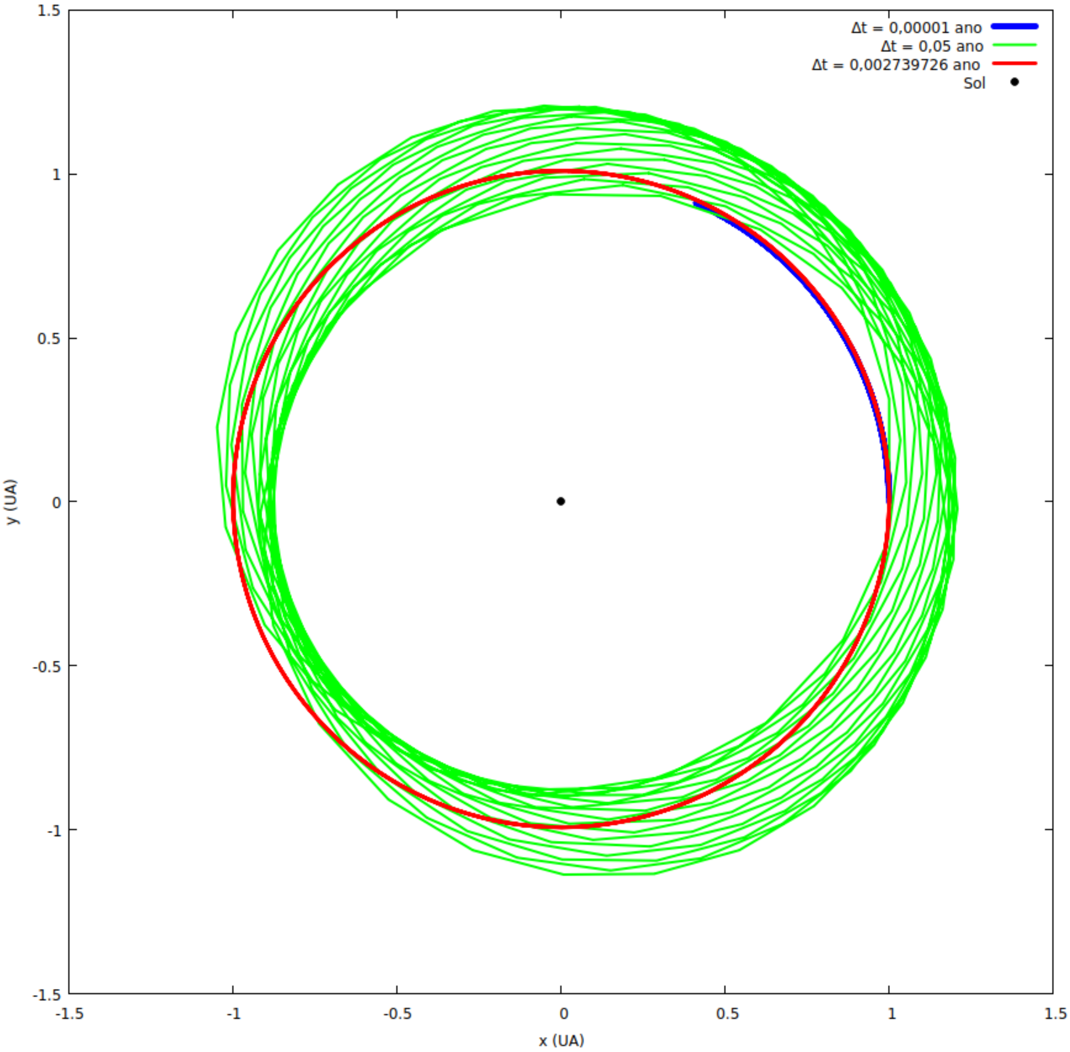

Figura 1: gráfico para diferentes Δt

Tarefa A1

Essa tarefa pede para que calculemos qual a velocidade inicial necessária para que cada planeta do Sistema Solar, com dados de raio médio e massa dados.

Como a órbita é circular, temos que a Força gravitacional será igual a centrípeta, logo:

$$M_P \cdot \frac{v_{0y}^2}{r} = \frac{GM_S M_P}{r^2} \Rightarrow v_{0y} = \sqrt{\frac{GM_S}{r}} \tag{3}$$

como temos que em unidades astronômicas, $GM_S = 4\pi^2$, temo que

$$v_{0y} = \sqrt{\frac{4\pi^2}{r}} = \frac{2\pi}{\sqrt{r}} \tag{4}$$

o que justifica a velocidade escolhida na tarefa anterior.

A excentricidade de uma elipse é dada pela seguinte relação:

$$\varepsilon = \frac{c}{a} = \frac{\sqrt{a^2 - b^2}}{a}$$

Logo, para mostrar que realmente é essa a velocidade para ter uma órbita circular, vou variando a velocidade e colocando no arquivo de saída, em formato de tabela, o valor de a e b, a excentricidade (que é nula quando temos uma circunferência) calculada, o v_{0y} encontrada e a esperada.

O código implementado está logo abaixo.

```fortran
c     tarefa a1 - velocidade de órbita circular
      implicit real*8 (a-h, o-z)
      parameter(pi = dacos(-1.d0))
      dimension dist(9) !quantidade de planetas
      parameter (dist = (/0.39d0, 0.72d0, 1.d0, 1.52d0, 5.2d0, 9.24d0,
     +19.19d0, 30.06d0, 39.53d0/))

c     abrindo arquivo de saida
      iout = 10
      open(iout, FILE='saida-a1-orbita-circular.dat')

      write(iout,*)'       a                          ','b
     +       ','excentricidade                ','v0y
     +  ','v0y esperado'

      do in = 1, 9

      v0y = 0.d0
      do ivel = 1, 20
      v0y = v0y + ivel*(2.d0*pi/(10*dsqrt(dist(in))))

      deltaT = (1.d0/365.d0) !um dia em anos
      GMs = 4 * (pi**2) !em unidades astronômicas
      v0x = 0.d0
      x0 = dist(in)
      y0 = 0.d0
      n = 365 * 1000 !tempo de simulação em dias terrestres

c     método de Euler para x1 e y1
      x1 = x0 + v0x * deltaT
      y1 = y0 + v0y * deltaT

c     método de Verlet
      j = 0
      do i = 2, n
          r = dsqrt(x1**2 + y1**2)
          !na componente x
          Fx = ((-1)*GMs*x1)/(r**3)
          x2 = 2*x1 - x0 + Fx*(deltaT**2)
          x0 = x1
          x1 = x2
          !na componente y
          Fy = ((-1)*GMs*y1)/(r**3)
```

```fortran
         y2 = 2*y1 - y0 + Fy*(deltaT**2)
         !Cálculo do período
         if (mod(j,2)== 0 .and. dabs(x2).LE.dist(in) .and. x2.LT.0 .an
     +d. y1*y2.LE.0) then
                j = j + 1
                b = x2
         else if (mod(j,2).NE.0 .and. y1*y2.LE.0) then
                j = j + 1
         end if
         if (j == 2) then
                a = dist(in)
                exc = (dsqrt(a**2 - b**2))/a
                if (exc .LT. 0.1) then
                  write(iout,*) dist(in), b, exc, v0y, (2.d0*pi/(dsqrt(
     +dist(in))))
                  go to 20
                end if
         end if

         y0 = y1
         y1 = y2

      end do

      end do
20    continue
      end do

      close(iout)

      end
```

Algoritmo 2: código para resolução da tarefa A1, parte das órbitas circulares

A tabela obtida foi a seguinte, na ordem dos planetas que encontramos no Sistema Solar (Mercúrio, Vênus, Terra, Marte, Júpiter, Saturno, Urano, Netuno e Plutão).

a	b	excentricidade	v0y	v0y esperado
0.39000000000000001	-0.38885382033707339	7.6610750952414236E-002	10.061148632539165	10.061148632539163
0.71999999999999997	-0.71968631261329707	2.9515488096399565E-002	7.4048048969306119	7.4048048969306102
1.0000000000000000	-0.99985421388439422	1.7074863912196030E-002	6.2831853071795862	6.2831853071795862
1.5200000000000000	-1.5199396452136567	8.9113756459685278E-003	5.0963362487929960	5.0963362487929960
5.2000000000000002	-5.1999954206630736	1.3271332479395516E-003	2.7553590302269777	2.7553590302269777
9.2400000000000002	-9.2399999362922145	1.1742897399004141E-004	2.0670162113025627	2.0670162113025627
19.190000000000001	-19.189999919254408	9.1735393123783582E-005	1.4343078687082951	1.4343078687082953
30.059999999999999	-30.059999884902933	8.7508982555376663E-005	1.1460020123254904	1.1460020123254901
39.530000000000001	-39.529999994362839	1.6888140644875342E-005	0.99934734082975984	0.99934734082975973

Figura 2: valores tabelados para as velocidades iniciais em uma órbita circular

Assim, vemos que a excentricidade está em uma precisão boa, bem próxima de zero e as velocidades muito precisas em relação às esperadas.

Agora, sabendo as velocidades para uma órbita circular, prossegui e fiz o outro programa que calcula o período de cada planeta e verifica uma das leis de Kepler.

O código implementado está disposto logo abaixo.

```
c     tarefa a1 - orbitas circulares T²/R³
      implicit real*8 (a-h, o-z)
      parameter(pi = dacos(-1.d0))
      dimension dist(9) !quantidade de planetas
      parameter (dist = (/0.39d0, 0.72d0, 1.d0, 1.52d0, 5.2d0, 9.24d0,
     +19.19d0, 30.06d0, 39.53d0/))

c     abrindo arquivo de saida
      iout = 10
      open(iout, FILE='saida-a1-TR.dat')

      write(iout,*)'    Período(T)                   ','Raio          '
     +,'             T²/R³ '

      do in = 1, 9

      deltaT = (1.d0/365.d0) !um dia em anos
      GMs = 4 * (pi**2) !em unidades astronômicas
      v0x = 0.d0
      v0y = (2.d0*pi)/(dsqrt(dist(in)))
      x0 = dist(in)
      y0 = 0.d0
      n = 365 * 3000 !tempo de simulação em dias terrestres

c     método de Euler para x1 e y1
      x1 = x0 + v0x * deltaT
      y1 = y0 + v0y * deltaT

c     método de Verlet
      j = 0
      do i = 2, n
          r = dsqrt(x1**2 + y1**2)
          !na componente x
          Fx = ((-1)*GMs*x1)/(r**3)
          x2 = 2*x1 - x0 + Fx*(deltaT**2)
          x0 = x1
          x1 = x2
          !na componente y
          Fy = ((-1)*GMs*y1)/(r**3)
          y2 = 2*y1 - y0 + Fy*(deltaT**2)
          !Cálculo do período
          if (mod(j,2) == 0 .and. j .NE. 2 .and. y1 * y2 .LE. 0) then
                j = j + 1
          else if (mod(j,2) .NE. 0 .and. y1 * y2 .LE. 0) then
                j = j + 1
          end if
          if (j == 2) then
```

```
                periodo = i/365.d0
                go to 20
            end if

            y0 = y1
            y1 = y2

        end do

20      write(iout,*) periodo, dist(in), (periodo**2)/(dist(in)**3)

        end do

        close(iout)

        end
```

Algoritmo 3: código para resolução da tarefa A1, parte da verificação da lei de Kepler

Assim, temos que o período está em uma precisão boa em relação aos valores esperados, e que a razão calculada é constante e próxima a 1, como Kepler provou e estabeleceu como uma das leis que regem a órbita de um planeta. Os valores estão na próxima figura e confirmam as conclusões dadas aqui.

Período(T)	Raio	T²/R³
0.24657534246575341	0.39000000000000001	1.0249565824121036
0.61369863013698633	0.71999999999999997	1.0090503060485618
1.0027397260273974	1.0000000000000000	1.0054869581535000
1.8767123287671232	1.5200000000000000	1.0029162086727175
11.860273972602739	5.2000000000000002	1.0004131962989156
28.087671232876712	9.2400000000000002	1.0000358114833778
84.065753424657530	19.190000000000001	1.0000306134031456
164.81095890410958	30.059999999999999	1.0000120740364298
248.53698630136986	39.530000000000001	1.0000037134736346

Figura 3: tabela que termina de resolver a tarefa A1

Tarefa B1

A tarefa B1 introduz a problemática de três corpos, temos a Terra, o Sol e Júpiter. A matemática utilizada é a mesma da tarefa A, usarei o método de Euler-Cromer para o primeiro ponto e o de Verlet para o restante. A única diferença é que teremos que computar a força gravitacional entre a Terra e Júpiter. Logo, teremos

$$\frac{d^2 y_T}{dt^2} = -\frac{G M_S y_T}{r_{TS}^3} - \frac{G M_J (y_T - y_J)}{r_{TJ}^3}$$

o equivalente para x, e o análogo para a força resultante de Júpiter em ambas componentes, com seus devidos ajustes (trocando o sinal nas coordenada e a massa de Júpiter pela da Terra).

Para continuar usando $GM_S = 4\pi^2$ e simplificar a implementação no código, vou usar a relação entre as massas do Sol, Júpiter e Terra dadas na tarefa anterior: $M_S = 10^3 M_J = 3 \cdot 10^5 M_T$. Logo

$$GM_T = \frac{GM_S}{3 \cdot 10^5}$$

e

$$GM_J = \frac{GM_S}{10^3} \tag{5}$$

Assim, o código implementado foi o seguinte:

```
c     tarefa b1
       implicit real*8 (a-h, o-z)
       parameter(pi = dacos(-1.d0))
       parameter(dmter = 1.d0)
       parameter(dmjup = 5.2d0)

c      abrindo arquivos de saída
       ioutT = 10
       open(ioutT, FILE='saida-b1-terra.dat')
       ioutJ = 20
       open(ioutJ, FILE='saida-b1-jupiter.dat')

       deltaT = (1.d0/365.d0) !um dia em anos
       n = 365 * 30 !tempo de simulação em dias terrestres
       GMs = 4*(pi**2)
c      Sol: posição
       xs = 0.d0
       ys = 0.d0

c      Terra: iniciando as variáveis
       v0xt = 0.d0
       v0yt = (2.d0*pi)/(dsqrt(dmter))
       x0t = dmter
       y0t = 0.d0

c      Júpiter: iniciando as variáveis
       v0xj = 0.d0
       v0yj = (2.d0*pi)/(dsqrt(dmjup))
       x0j = dmjup
       y0j = 0.d0

c      método de Euler para x1 e y1
       x1t = x0t + v0xt * deltaT
       y1t = y0t + v0yt * deltaT
       write(ioutT,*) x0t, y0t
       write(ioutT,*) x1t, y1t
       x1j = x0j + v0xj * deltaT
       y1j = y0j + v0yj * deltaT
```

```
      write(ioutJ,*) x0j, y0j
      write(ioutJ,*) x1j, y1j

c     método de Verlet
      do i = 2, n
          rts = dsqrt((x1t - xs)**2 + (y1t - ys)**2) !dist Terra-Sol
          rtj = dsqrt((x1t - x1j)**2 + (y1t - y1j)**2) !dist Terra-Jup
          rjs = dsqrt((x1j - xs)**2 + (y1j - xs)**2) !dist Jup-Sol
          !Terra:
          Fxter = (-1)*(((GMs*(x1t-xs))/(rts**3)) + ((GMs*(x1t-x1j))/(1
     +0.d3*(rtj**3))))
          x2t = 2*x1t - x0t + Fxter*(deltaT**2)
          Fyter = (-1)*(((GMs*(y1t-ys))/(rts**3)) + ((GMs*(y1t-y1j))/(1
     +0.d3*(rtj**3))))
          y2t = 2*y1t - y0t + Fyter*(deltaT**2)
          !Júpiter
          Fxjup = (-1)*(((GMs*(x1j-xs))/(rjs**3)) + ((GMs*(x1j-x1t))/(3
     +.d3*(rtj**3))))
          x2j = 2*x1j - x0j + Fxjup*(deltaT**2)
          Fyjup = (-1)*(((GMs*(y1j-ys))/(rjs**3)) + ((GMs*(y1j-y1t))/(3
     +.d3*(rtj**3))))
          y2j = 2*y1j - y0j + Fyjup*(deltaT**2)

          x0t = x1t
          x1t = x2t
          y0t = y1t
          y1t = y2t
          x0j = x1j
          x1j = x2j
          y0j = y1j
          y1j = y2j

          write(ioutT, *) x2t, y2t
          write(ioutJ, *) x2j, y2j
      end do

      close(ioutT)
      close(ioutJ)

      end
```

Algoritmo 4: código para resolução da tarefa B1

Dessa maneira, obtemos a seguinte imagem das órbitas da Terra e Júpiter.

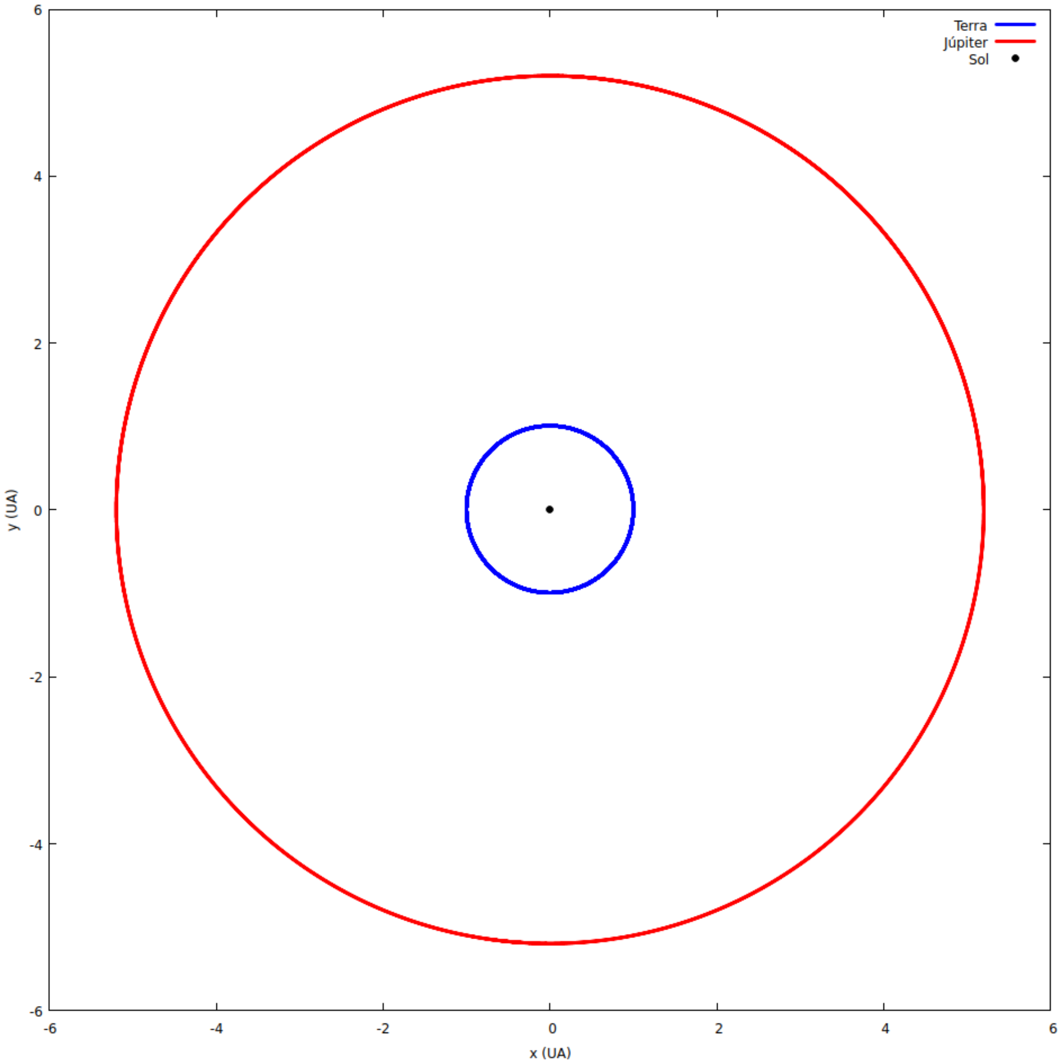

Figura 4: sistema de três corpos: Sol, Júpiter e Terra

Podemos ter a impressão de que a a órbita terrestre sempre passa pelo mesmo caminho, mas se aproximarmos vemos que isso não acontece, como fiz na próxima figura.

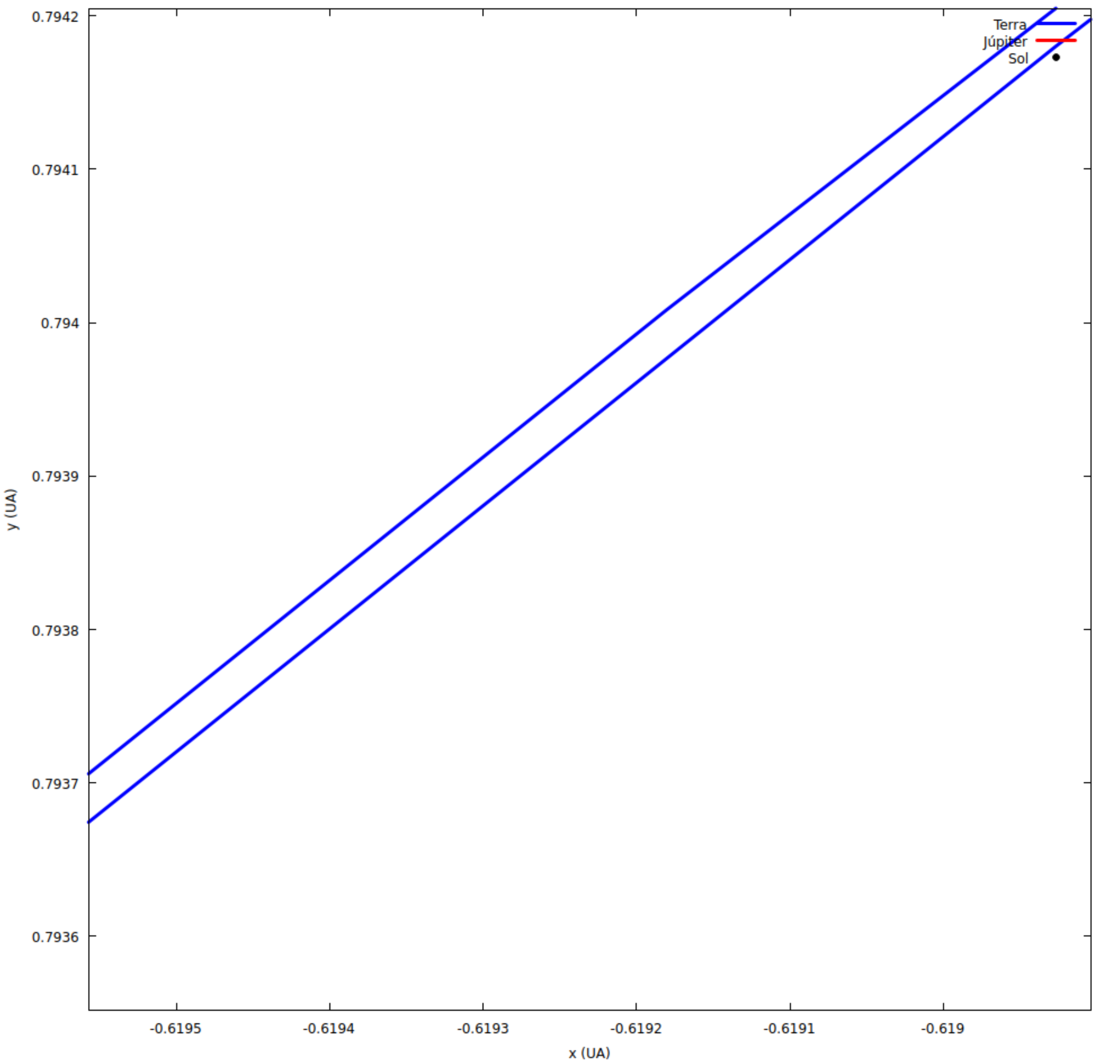

Figura 5: aproximação da imagem anterior para notarmos a não peridiocidade da órbita da terra

Vemos claramente a diferença, e percebemos que vale em torno de $0,00002UA$.

Tarefa B2

A tarefa B2 pede apenas que multipliquemos a massa de Júpiter por cem e por mil e vejamos a acentuação do efeito visto anteriormente.

Para tal, usarei o mesmo código utilizado anteriormente, multiplicando a relação (5) por cem e depois por mil.

```
c     tarefa b2
       implicit real*8 (a-h, o-z)
       parameter(pi = dacos(-1.d0))
       parameter(dmter = 1.d0)
       parameter(dmjup = 5.2d0)

c      abrindo arquivos de saída
```

```fortran
      ioutT = 10
      open(ioutT, FILE='saida-b2-terra1000.dat')
      ioutJ = 20
      open(ioutJ, FILE='saida-b2-jupiter1000.dat')

      deltaT = (1.d0/365.d0) !um dia em anos
      n = 365 * 30 !tempo de simulação em dias terrestres
      GMs = 4*(pi**2)

c     Sol: posição
      xs = 0.d0
      ys = 0.d0

c     Terra: iniciando as variáveis
      v0xt = 0.d0
      v0yt = (2.d0*pi)/(dsqrt(dmter))
      x0t = dmter
      y0t = 0.d0

c     Júpiter: iniciando as variáveis
      v0xj = 0.d0
      v0yj = (2.d0*pi)/(dsqrt(dmjup))
      x0j = dmjup
      y0j = 0.d0

c     método de Euler para x1 e y1
      x1t = x0t + v0xt * deltaT
      y1t = y0t + v0yt * deltaT
      write(ioutT,*) x0t, y0t
      write(ioutT,*) x1t, y1t
      x1j = x0j + v0xj * deltaT
      y1j = y0j + v0yj * deltaT
      write(ioutJ,*) x0j, y0j
      write(ioutJ,*) x1j, y1j

c     método de Verlet
      do i = 2, n
          rts = dsqrt((x1t - xs)**2 + (y1t - ys)**2) !dist Terra-Sol
          rtj = dsqrt((x1t - x1j)**2 + (y1t - y1j)**2) !dist Terra-Jup
          rjs = dsqrt((x1j - xs)**2 + (y1j - xs)**2) !dist Jup-Sol
          !Terra:
          Fxter = (-1)*(((GMs*(x1t-xs))/(rts**3)) + ((GMs*(x1t-x1j))/(1
     +*(rtj**3)))) !x100 e x1000 alterando aqui
          x2t = 2*x1t - x0t + Fxter*(deltaT**2)
          Fyter = (-1)*(((GMs*(y1t-ys))/(rts**3)) + ((GMs*(y1t-y1j))/(1
     +*(rtj**3)))) !x100 e x1000 alterando aqui
          y2t = 2*y1t - y0t + Fyter*(deltaT**2)
          !Júpiter
```

```
          Fxjup = (-1)*(((GMs*(x1j-xs))/(rjs**3)) + ((GMs*(x1j-x1t))/(3
     +.d3*(rtj**3))))
          x2j = 2*x1j - x0j + Fxjup*(deltaT**2)
          Fyjup = (-1)*(((GMs*(y1j-ys))/(rjs**3)) + ((GMs*(y1j-y1t))/(3
     +.d3*(rtj**3))))
          y2j = 2*y1j - y0j + Fyjup*(deltaT**2)

          x0t = x1t
          x1t = x2t
          y0t = y1t
          y1t = y2t
          x0j = x1j
          x1j = x2j
          y0j = y1j
          y1j = y2j

          write(ioutT, *) x2t, y2t
          write(ioutJ, *) x2j, y2j
      end do

      close(ioutT)
      close(ioutJ)

      end
```

Algoritmo 5: código para resolução da tarefa B2

Assim temos o seguintes resultados, evidenciados pelas figuras abaixo.

Vemos, de maneira evidente, que quando multiplicamos por cem a massa de Júpiter a não periodicidade da órbita da terra, que muda de trajetória muitas vezes. Podemos observar melhor na figura aproximada que está abaixo da geral.

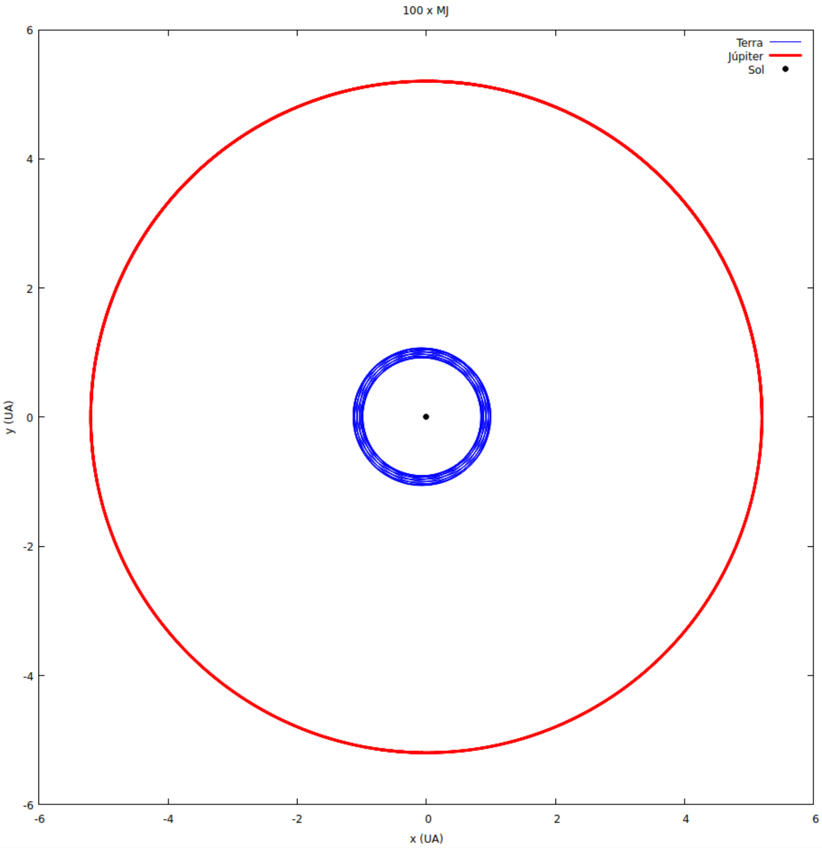

Figura 6: órbitas multiplicando a massa de Júpiter por cem

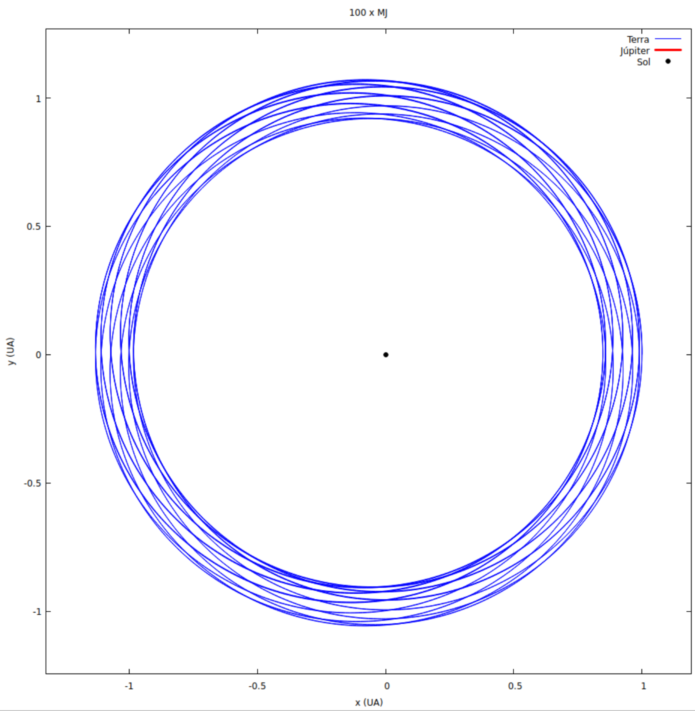

Figura 7: figura aproximada da órbita da Terra

Quando multiplicamos por mil a massa de Júpiter, o que vemos é um desenrolar um tanto negativo para nós, habitantes do pálido ponto azul. A Terra acaba sendo lançada para longe do Sistema Solar, como podemos abservar logo abaixo.

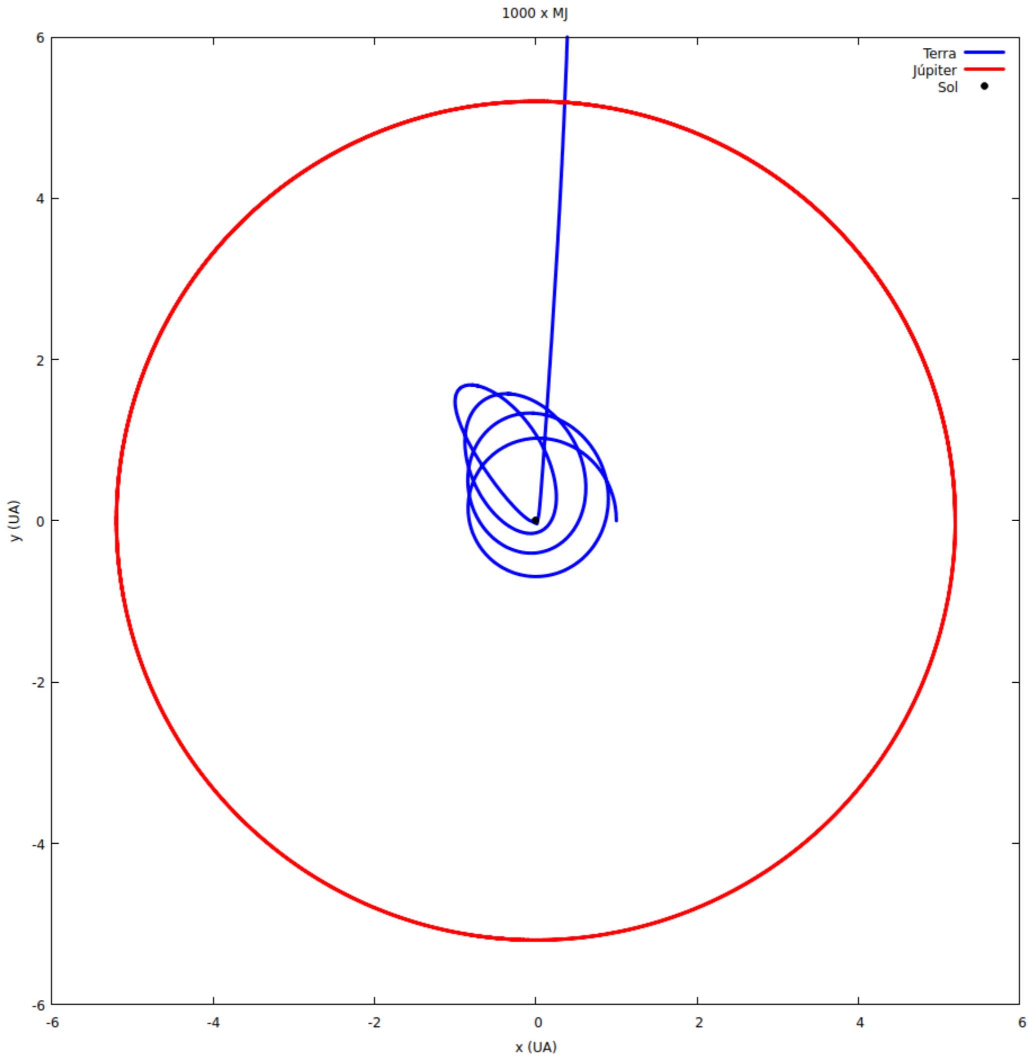

Figura 8: órbitas multiplicando a massa de Júpiter por mil

Tarefa B3

A tarefa B3 pede pede para calcular as órbitas de três asteroides e Júpiter. código implementado está disposto abaixo, e segue a mesma matemática e lógica do anterior.

```
c     tarefa b3
       implicit real*8 (a-h, o-z)
       parameter(pi = dacos(-1.d0))
       parameter(dmjup = 5.2d0)
       parameter(dma1 = 3.d0)
       parameter(dma2 = 3.276d0)
       parameter(dma3 = 3.7d0)

c      abrindo arquivos de saída
```

```
      ioutA1 = 10
      open(ioutA1, FILE='saida-b3-ast1.dat')
      ioutA2 = 20
      open(ioutA2, FILE='saida-b3-ast2.dat')
      ioutA3 = 30
      open(ioutA3, FILE='saida-b3-ast3.dat')
      ioutJ = 40
      open(ioutJ, FILE='saida-b3-jupiter.dat')

      deltaT = (1.d0/365.d0) !um dia em anos
      n = 365 * 25 !tempo de simulação em dias terrestres
      GMs = 4*(pi**2)

c     Sol: posição
      xs = 0.d0
      ys = 0.d0

c     Asteroides: iniciando as variáveis
      !Ast1
      v0xa1 = 0.d0
      v0ya1 = 3.628d0
      x0a1 = dma1
      y0a1 = 0.d0
      !Ast2
      v0xa2 = 0.d0
      v0ya2 = 3.471d0
      x0a2 = dma2
      y0a2 = 0.d0
      !Ast3
      v0xa3 = 0.d0
      v0ya3 = 3.267d0
      x0a3 = dma3
      y0a3 = 0.d0

c     Júpiter: iniciando as variáveis
      v0xj = 0.d0
      v0yj = 2.755d0
      x0j = dmjup
      y0j = 0.d0

c     método de Euler para x1 e y1
      x1a1 = x0a1 + v0xa1 * deltaT
      y1a1 = y0a1 + v0ya1 * deltaT
      write(ioutA1,*) x0a1, y0a1
      write(ioutA1,*) x1a1, y1a1
      x1a2 = x0a2 + v0xa2 * deltaT
      y1a2 = y0a2 + v0ya2 * deltaT
      write(ioutA2,*) x0a2, y0a2
```

```
      write(ioutA2,*) x1a2, y1a2
      x1a3 = x0a3 + v0xa3 * deltaT
      y1a3 = y0a3 + v0ya3 * deltaT
      write(ioutA3,*) x0a3, y0a3
      write(ioutA3,*) x1a3, y1a3
      x1j = x0j + v0xj * deltaT
      y1j = y0j + v0yj * deltaT
      write(ioutJ,*) x0j, y0j
      write(ioutJ,*) x1j, y1j

c     método de Verlet
      do i = 2, n
          ra1s = dsqrt((x1a1 - xs)**2 + (y1a1 - ys)**2) !dist Ast1-Sol
          ra1j = dsqrt((x1a1 - x1j)**2 + (y1a1 - y1j)**2) !dist Ast1-Jup
          rjs = dsqrt((x1j - xs)**2 + (y1j - xs)**2) !dist Jup-Sol
          ra2s = dsqrt((x1a2 - xs)**2 + (y1a2 - ys)**2) !dist Ast2-Sol
          ra2j = dsqrt((x1a2 - x1j)**2 + (y1a2 - y1j)**2) !dist Ast2-Jup
          ra3s = dsqrt((x1a3 - xs)**2 + (y1a3 - ys)**2) !dist Ast3-Sol
          ra3j = dsqrt((x1a3 - x1j)**2 + (y1a3 - y1j)**2) !dist Ast3-Jup
          !Asteroides:
            !Ast1
          Fxa1 = (-1)*(((GMs*(x1a1-xs))/(ra1s**3)) + ((GMs*(x1a1-x1j))/
     +(10.d3*(ra1j**3))))
          x2a1 = 2*x1a1 - x0a1 + Fxa1*(deltaT**2)
          Fya1 = (-1)*(((GMs*(y1a1-ys))/(ra1s**3)) + ((GMs*(y1a1-y1j))/
     +(10.d3*(ra1j**3))))
          y2a1 = 2*y1a1 - y0a1 + Fya1*(deltaT**2)
            !Ast2
          Fxa2 = (-1)*(((GMs*(x1a2-xs))/(ra2s**3)) + ((GMs*(x1a2-x1j))/
     +(10.d3*(ra2j**3))))
          x2a2 = 2*x1a2 - x0a2 + Fxa2*(deltaT**2)
          Fya2 = (-1)*(((GMs*(y1a2-ys))/(ra2s**3)) + ((GMs*(y1a2-y1j))/
     +(10.d3*(ra2j**3))))
          y2a2 = 2*y1a2 - y0a2 + Fya2*(deltaT**2)
            !Ast1
          Fxa3 = (-1)*(((GMs*(x1a3-xs))/(ra3s**3)) + ((GMs*(x1a3-x1j))/
     +(10.d3*(ra3j**3))))
          x2a3 = 2*x1a3 - x0a3 + Fxa3*(deltaT**2)
          Fya3 = (-1)*(((GMs*(y1a3-ys))/(ra3s**3)) + ((GMs*(y1a3-y1j))/
     +(10.d3*(ra3j**3))))
          y2a3 = 2*y1a3 - y0a3 + Fya3*(deltaT**2)
          !Júpiter
          Fxjup = (-1)*(((GMs*(x1j-xs))/(rjs**3)))
          x2j = 2*x1j - x0j + Fxjup*(deltaT**2)
          Fyjup = (-1)*(((GMs*(y1j-ys))/(rjs**3)))
          y2j = 2*y1j - y0j + Fyjup*(deltaT**2)

          x0a1 = x1a1
```

```
          x1a1 = x2a1
          y0a1 = y1a1
          y1a1 = y2a1

          x0a2 = x1a2
          x1a2 = x2a2
          y0a2 = y1a2
          y1a2 = y2a2

          x0a3 = x1a3
          x1a3 = x2a3
          y0a3 = y1a3
          y1a3 = y2a3

          x0j = x1j
          x1j = x2j
          y0j = y1j
          y1j = y2j

          write(ioutA1, *) x2a1, y2a1
          write(ioutA2, *) x2a2, y2a2
          write(ioutA3, *) x2a3, y2a3
          write(ioutJ, *) x2j, y2j
      end do

      close(ioutA1)
      close(ioutA2)
      close(ioutA3)
      close(ioutJ)

      end
```

Algoritmo 6: código para resolução da tarefa B3

Como resultado, obtive a seguinte figura. Nela, podemos observar as Lacunas de Kirkwood. Essas lacunas são espaços um pouco vazios no entre os asteroides, que correspondem a zonas de ressonância onde a atração gravitacional de Júpiter impede a permanência de qualquer corpo celeste. Esse é o principal e indiscutivelmente importante resultado dessa tarefa.

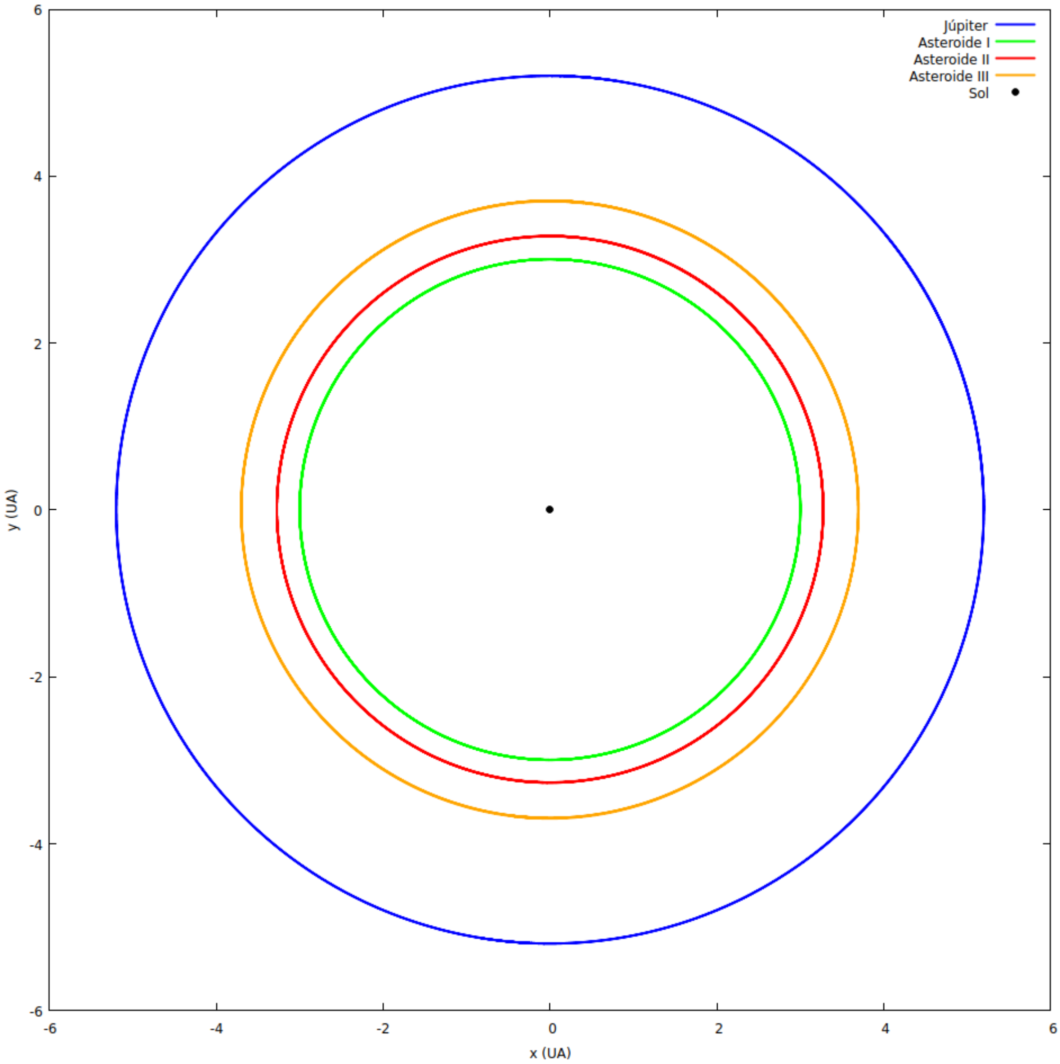

Figura 9: órbitas dos asteroides e Júpiter