

Universidade de São Paulo

Introdução à Física Computacional - Projeto 5

João Victor Dell Agli Floriano - 10799783

1. Tarefa A

Esta tarefa inicial teve o objetivo de implementar o método de Verlet na simulação do efeito de atração gravitacional entre o Sol e os planetas do sistema solar. Considerando que força de atração gravitacional que os planetas sentem por efeito do Sol é:

$$\vec{F}_g = -\frac{GM_S M_P}{|\vec{r}_P - \vec{r}_S|^3}(\vec{r}_P - \vec{r}_S) \quad (1)$$

Que nos permite deduzir a aceleração a partir da Segunda Lei de Newton:

$$\begin{aligned} \vec{F}_g &= M_P \vec{a}_P \\ M_P \vec{a}_P &= -\frac{GM_S M_P}{r^3} \vec{r} \\ \vec{a}_P &= -\frac{GM_S}{r^3} \vec{r} \end{aligned} \quad (2)$$

E, a partir da equação da aceleração, podemos implementar um método numérico de cálculo das posições x e y de determinado planeta em relação ao Sol. Para tal, usamos o método de Verlet, o qual se baseia na expansão de Taylor das posições:

$$x(t_i \pm \Delta t) = x(t_i) \pm \frac{dx}{dt_i} \Delta t \pm \frac{d^2x}{dt_i^2} (\Delta t)^2 \dots$$

Sendo assim, o cálculo iterativo das posições a partir da aceleração é dada por:

$$\begin{aligned} x_{i+1} &= 2x_i - x_{i-1} + \frac{d^2x}{dt^2} (\Delta t)^2 + O(\Delta t^4) \\ y_{i+1} &= 2y_i - y_{i-1} + \frac{d^2y}{dt^2} (\Delta t)^2 + O(\Delta t^4) \end{aligned} \quad (3)$$

Para os demais cálculos em máquina, todos os parâmetros foram utilizados em unidades astronômicas, como o produto $GM_S = 4\pi^2$, assim como os demais dados dos planetas:

Planeta	massa (Kg)	raio (UA)	excentricidade
Mercúrio	$2.4 \cdot 10^{23}$	0.39	0.206
Venus	$4.9 \cdot 10^{24}$	0.72	0.007
Terra	$6.0 \cdot 10^{24}$	1.00	0.017
Marte	$6.6 \cdot 10^{23}$	1.52	0.093
Júpiter	$1.9 \cdot 10^{27}$	5.20	0.048
Saturno	$5.7 \cdot 10^{26}$	9.24	0.056
Urano	$8.8 \cdot 10^{25}$	19.19	0.046
Netuno	$1.03 \cdot 10^{26}$	30.06	0.010
Plutão	$6.0 \cdot 10^{24}$	39.53	0.248

Nos subtópicos a seguir, este método é implementado em contextos diferentes.

1.1. Tarefa a1

Essa tarefa teve como objetivo implementar o método de Verlet descrito acima para todos os planetas do Sistema Solar, considerando o Sol na posição (0, 0), procurando por tentativa e erro velocidades iniciais as quais levariam a órbitas circulares dos mesmos. Além disso, foi também realizada uma verificação da Terceira Lei de Kepler para cada planeta, que prevê que os quadrados dos períodos de translação ao redor do Sol de cada planeta são proporcionais aos cubos dos raios médios de suas órbitas:

$$\frac{T^2}{R^3} = \text{constante} \quad (4)$$

O código inicia os planetas na posição (R_p , 0), com velocidades iniciais experimentais, e após uma quantidade de iterações especificada, verificava-se a partir de suas posições escritas se quando o valor de x fosse próximo de zero, o valor de y era próximo do raio inicial, assim como quando o valor de y fosse próximo de zero se sua posição em x era próxima da sua posição inicial R_p . A partir desse método, foi-se ajustando o chute inicial das velocidades iniciais que aproximasse a posição x final da posição inicial de cada planeta, indicando assim uma órbita circular. O cálculo e verificação da Terceira Lei de Kepler foi feito verificando toda a vez que o planeta completasse uma volta completa, verificando quando o valor de x se aproximava do raio inicial, indicando assim que uma volta foi completa. Feita essa verificação, guardou-se o valor do tempo de tal volta em relação ao início da simulação, e calculou-se o tempo em relação ao cálculo do período anterior. A partir disso, era possível calcular a Terceira Lei elevando este valor ao quadrado e dividindo-o pelo atual raio em relação ao Sol.

Para essa e as demais implementações, foi usado um valor de $\Delta t = 10^{-3}$. O código que implementa o que foi descrito acima é:

```
double precision function d2f(x, r)
  implicit double precision (a-h, o-z)
  !Função para calcular a aceleração gravitacional dos planetas

  pi = 4d0*datan(1d0)
  GMs = 4d0*(pi**2d0) !UA**3/ano**2

  d2f = -(GMs*x)/(r**3d0)

  return
end function d2f

program main
  implicit double precision (a-h, o-z)
  double precision sol(2)

  double precision raios(9)
  double precision velocidades(9)
  integer iters(9)

  pi = 4d0*datan(1d0)

  sol(1) = 0d0
  sol(2) = 0d0

  dt = 1d-3

  raios(1) = 39d-2 !mercurio
  velocidades(1) = 4d0*pi/1252d-3
  iters(1) = 1000

  raios(2) = 72d-2 !venus
  velocidades(2) = 4d0*pi/1705d-3
  iters(2) = 1000

  raios(3) = 1d0 !terra
  velocidades(3) = 4d0*pi/2d0
```

```

iters(3) = 1000

raios(4)      = 152d-2 !marTE
velocidades(4) = 4d0*pi/2467d-3
iters(4) = 5000

raios(5)      = 520d-2 !jupiter
velocidades(5) = 4d0*pi/4562d-3
iters(5) = 30000

raios(6)      = 924d-2 !saturno
velocidades(6) = 4d0*pi/6080d-3
iters(6) = 50000

raios(7)      = 1919d-2 !urano
velocidades(7) = 4d0*pi/8761d-3
iters(7) = 100000

raios(8)      = 3006d-2 !netuno
velocidades(8) = 4d0*pi/10966d-3
iters(8) = 170000

raios(9)      = 3953d-2 !plutao
velocidades(9) = 4d0*pi/12576d-3
iters(9) = 250000


open(25, file = "saida-A1-1-10799783.dat")

do j = 1, 9
    v0x = 0d0
    v0y = velocidades(j)

    x0 = raios(j)
    y0 = 0d0

    x1 = x0 + v0x*dt
    y1 = y0 + v0y*dt

    x = 0d0
    y = 0d0

    !variáveis para a Lei de Kepler
    T0 = 0d0
    rK0 = 0d0
    rK = 1d0

    open(24, file = "saida-A1-2-10799783.dat")

```

```

write(25,*) "planeta ", j
do i = 1, iters(j)

    r = dsqrt((x1 - sol(1))**2 + (y1 - sol(2))**2)

    x = 2d0*x1 - x0 + d2f(x1, r)*(dt**2d0)
    y = 2d0*y1 - y0 + d2f(y1, r)*(dt**2d0)

    x0 = x1
    y0 = y1

    x1 = x
    y1 = y

    write(24,*) x, y

    if(dabs(x - raios(j)) .LT. 1d-1
& .AND. dabs(y) .LT. 4d-3) then

        T = i*dt
        rK = ((T-T0)**2)/(r**3)

        if((rK - rK0) .GT. 1d-5) then
            write(25,*) rK
        end if

        T0 = T
        rK0 = rK
    end if

end do

close(24)
end do
close(25)

stop
end program main

```

Rodando o programa acima, temos:

```
joao@joao-Inspiron-15-7000-Gaming: ~/Área de Trabalho/Intro a Fiscomp/Quinto P...
Arquivo Editar Ver Pesquisar Terminal Ajuda
o Projeto/tarefaA$ f77 A1.f -o A1
(base) joao@joao-Inspiron-15-7000-Gaming:~/Área de Trabalho/Intro a Fiscomp/Quin
o Projeto/tarefaA$ ./A1
(base) joao@joao-Inspiron-15-7000-Gaming:~/Área de Trabalho/Intro a Fiscomp/Quin
o Projeto/tarefaA$
```

Que rende dois arquivos de saída. O primeiro, uma lista da Terceira Lei de Kepler calculada para todos os planetas, nos permite fazer uma tabela:

Planeta	Terceira Lei de Kepler
Mercúrio	0.98000307161416578
Vênus	0.97103170510306791
Terra	0.99806116923752908
Marte	0.99577477489579236
Júpiter	0.99800734818268855
Saturno	0.99927629276237762
Urano	1.0000840929348878
Netuno	0.99960002487738997
Plutão	0.99926362869454466

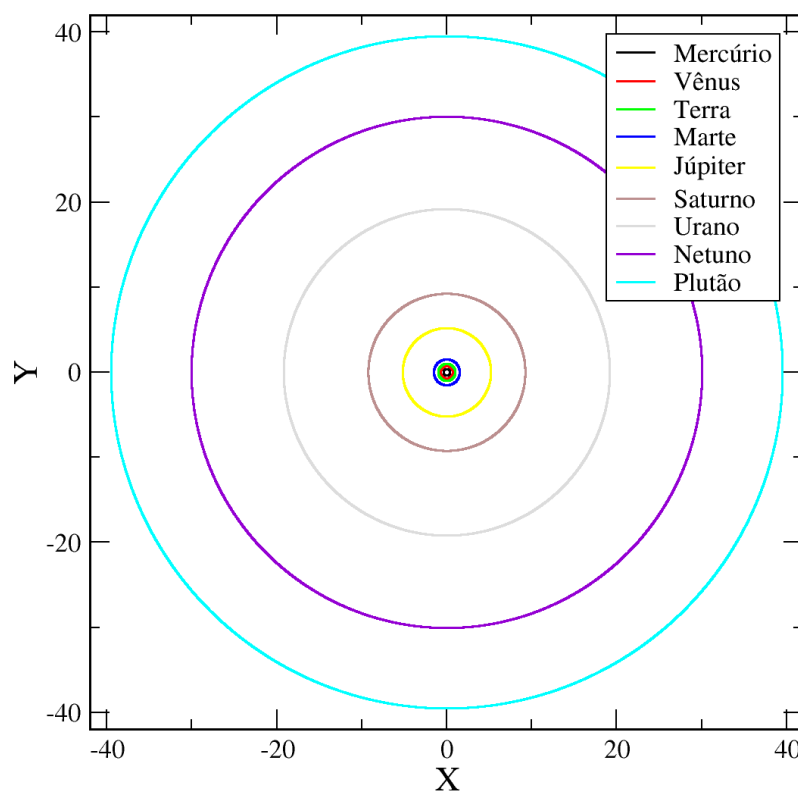
Analisando a tabela, é possível perceber que os valores da Terceira Lei para o Sistema Solar são bem próximos, e que se aproximam aos valores reais esperados, como demonstrado pela tabela a seguir¹:

¹ <https://www.physicsclassroom.com/class/circles/Lesson-4/Kepler-s-Three-Laws>

Planet	Period (yr)	Average Distance (au)	T^2/R^3 (yr ² /au ³)
Mercury	0.241	0.39	0.98
Venus	.615	0.72	1.01
Earth	1.00	1.00	1.00
Mars	1.88	1.52	1.01
Jupiter	11.8	5.20	0.99
Saturn	29.5	9.54	1.00
Uranus	84.0	19.18	1.00
Neptune	165	30.06	1.00
Pluto	248	39.44	1.00

Além da tabela acima, a partir da segunda saída do programa, é possível graficar as posições de cada planeta rodado separadamente:

Posição dos planetas do sistema Solar



Nos permitindo verificar que cada um segue uma trajetória circular, como esperado.

1.2. Tarefa a2

Essa tarefa teve o objetivo de novamente implementar as órbitas dos planetas do sistema solar, agora com órbitas não circulares, mas sim elípticas e fechadas. Para isso, utilizou-se os seus dados fornecidos da excentricidade de cada, que a partir da equação abaixo, possibilitou estimar uma boa velocidade inicial²:

$$v_o = \frac{2\pi a}{T} \left[1 - \frac{1}{4}e^2 - \frac{3}{64}e^4 - \frac{5}{256}e^6 - \frac{175}{16384}e^8 - \dots \right] \quad (5)$$

O código que implementa esses cálculos é:

```
double precision function d2f(x, r)
  implicit double precision (a-h, o-z)
  !Função para calcular a aceleração gravitacional dos planetas

  pi = 4d0*datan(1d0)
  GMs = 4d0*(pi**2d0) !UA**3/ano**2

  d2f = -(GMs*x)/(r**3d0)

  return
end function d2f

double precision function e(ex)
  implicit double precision (a-h, o-z)
  !função para o cálculo do termo da excentricidade no calculo da
  velocidade

  e =(1d0 -(ex**2)/4d0 -(ex**4)*(3d0/64d0) -(ex**6)*(5d0/256d0))

  return
end function e

program main
  implicit double precision (a-h, o-z)
  double precision sol(2)
```

² https://pt.wikipedia.org/wiki/Velocidade_orbital

```

double precision raios(9)
double precision velocidades(9)
integer iters(9)

pi = 4d0*datan(1d0)

sol(1) = 0d0
sol(2) = 0d0

raios(1) = 39d-2 !mercurio
velocidades(1) = (2d0*pi*raios(1)/24d-2)*e(206d-3)
!velocidades(1) = 2d0*pi/dsqrt(raios(1))
iters(1) = 1000

raios(2)      = 72d-2 !venus
velocidades(2) = (2d0*pi*raios(2)/615d-3)*e(7d-3)
iters(2) = 1000

raios(3)      = 1d0 !terra
velocidades(3) = (2d0*pi*raios(3)/1d0)*e(17d-3)
iters(3) = 1000

raios(4)      = 152d-2 !marte
velocidades(4) = (2d0*pi*raios(4)/188d-2)*e(93d-3)
iters(4) = 5000

raios(5)      = 520d-2 !jupiter
velocidades(5) = (2d0*pi*raios(5)/1186d-2)*e(48d-3)
iters(5) = 30000

raios(6)      = 924d-2 !saturno
velocidades(6) = (2d0*pi*raios(6)/29d0)*e(56d-3)
iters(6) = 50000

raios(7)      = 1919d-2 !urano
velocidades(7) = (2d0*pi*raios(7)/8432d-2)*e(46d-3)
iters(7) = 100000

```

```

raios(8) = 3006d-2 !netuno
velocidades(8) = (2d0*pi*raios(8)/165d0)*e(10d-3)
iters(8) = 170000

raios(9) = 3953d-2 !plutao
velocidades(9) = (2d0*pi*raios(9)/248d0)*e(248d-3)
iters(9) = 250000

```

```

open(25, file = "saida-A2-1-10799783.dat")

```

```

do j = 1, 9
    dt = 1d-3
    v0x = 0d0
    v0y = velocidades(j)

```

```

    x0 = raios(j)
    y0 = 0d0

```

```

    x1 = x0 + v0x*dt
    y1 = y0 + v0y*dt

```

```

    x = 0d0
    y = 0d0

```

```

    T0 = 0d0

```

```

    rK0 = 0d0
    rK = 1d0

```

```

open(24, file = "saida-A2-2-10799783.dat")

```

```

write(25,*) "planeta ", j

```

```

do i = 1, iters(j)

```

```

    r = dsqrt((x1 - sol(1))**2 + (y1 - sol(2))**2)

```

```

    x = 2d0*x1 - x0 + d2f(x1, r)*(dt**2d0)
    y = 2d0*y1 - y0 + d2f(y1, r)*(dt**2d0)

```

```

    x0 = x1
    y0 = y1

```

```

        x1 = x
        y1 = y

        write(24,*) x, y

        if(dabs(x - raios(j)) .LT. 1d-3
& .AND. dabs(y) .LT. 5d-3) then

            T = i*dt
            rK = ((T-T0)**2)/(r**3)

            if((rK - rK0) .GT. 1d-2) then
                write(25,*) rK
            end if

            T0 = T
            rK0 = rK
        end if

    end do

    close(24)
end do
close(25)

stop
end program main

```

Que rodando, rende:

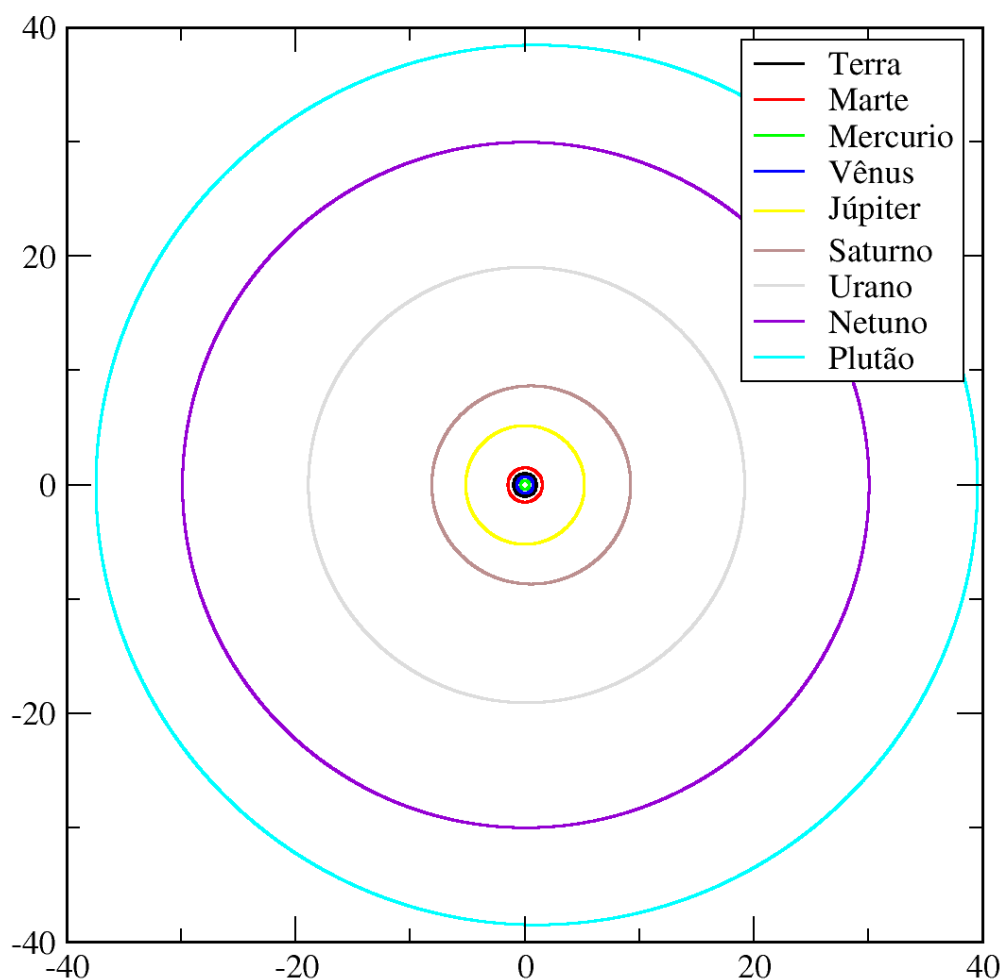
```

joao@joao-Inspiron-15-7000-Gaming: ~/Área de Trabalho/Intro a Fiscomp/Quinto P...
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda
o Projeto/tarefaA$ f77 A2.f -o A2
(base) joao@joao-Inspiron-15-7000-Gaming:~/Área de Trabalho/Intro a Fiscomp/Quin
o Projeto/tarefaA$ ./A2
(base) joao@joao-Inspiron-15-7000-Gaming:~/Área de Trabalho/Intro a Fiscomp/Quin
o Projeto/tarefaA$

```

Criando dois arquivos de saída. Com um deles, conseguimos verificar as posições dos planetas, e graficando, verificar o formato das órbitas:

Posição dos planetas do Sistema Solar



Apesar de se assemelhar com as órbitas circulares, é possível perceber agora que os formatos das órbitas são ligeiramente deslocados do centro (0, 0) indicando que percorrem órbitas elípticas das quais o Sol, nesse mesmo centro, ocupa um dos focos.

Com o outro arquivo de saída, é possível analisar novamente a Terceira Lei de Kepler, rendendo a tabela:

Planeta	Terceira Lei de Kepler
Mercúrio	1.0207454232029234
Vênus	0.95825391839059670

Terra	0.99804836620519599
Marte	0.96723014500581339
Júpiter	0.99514502237161673
Saturno	0.83125739364824724
Urano	0.97885629079006764
Netuno	0.99287768891804296
Plutão	0.92398815224601616

Analisando a tabela, é possível perceber valores levemente diferentes dos apresentados na tarefa anterior, porém valores muito próximos entre si, o que confere com o esperado.

2. Tarefa B

O próximo conjunto de tarefas teve como objetivo implementar o método de Verlet para três ou mais corpos, agora considerando o efeito gravitacional de outros corpos além do sol no efeito da órbita de um planeta. Para os cálculos a seguir, estaremos considerando que $M_S = 10^3 M_J = 3 \cdot 10^5 M_T$.

2.1. Tarefa b1

Esta tarefa teve como objetivo implementar o cálculo das órbitas da Terra e de Júpiter, considerando seus respectivos efeitos gravitacionais um no outro. Para isso, consideramos a velocidade inicial que confere a Júpiter uma órbita circular, e a velocidade da Terra em sua órbita elíptica. O código que implementa essa simulação é:

```
double precision function d2ft(x, xj, r, rj)
  implicit double precision (a-h, o-z)

  !Função para calcular a aceleração da Terra

  pi = 4d0*datan(1d0)
  GMs = 4d0*(pi**2) !UA**3/ano**2
```

```
GMj = GMs/1d3
```

```
GMt = GMs/3d5
```

```
d2ft = -(GMs*x)/(r**3) -GMj*(x-xj)/(rj**3)
```

```
return
```

```
end function d2ft
```

```
double precision function d2fj(x, xj, r, rj)
```

```
implicit double precision (a-h, o-z)
```

```
!Função para calcular a aceleração de Júpiter
```

```
pi = 4d0*datan(1d0)
```

```
GMs = 4d0*(pi**2) !UA**3/ano**2
```

```
GMj = GMs/1d3
```

```
GMt = GMs/3d5
```

```
d2fj = -(GMs*x)/(r**3) -GMt*(x-xj)/(rj**3)
```

```
return
```

```
end function d2fj
```

```
double precision function e(ex)
```

```
implicit double precision (a-h, o-z)
```

```
!função para o cálculo do termo da excentricidade no calculo da  
velocidade
```

```
e =(1d0 -(ex**2)/4d0 -(ex**4)*(3d0/64d0) -(ex**6)*(5d0/256d0))
```

```
return
```

```
end function e
```

```
program main
```

```
implicit double precision (a-h, o-z)
```

```
double precision sol(2)
```

```
double precision ter(2)
```

```

double precision ter1(2)
double precision ter0(2)
double precision velter(2)
double precision jup(2)
double precision jup1(2)
double precision jup0(2)
double precision veljup(2)

dt = 1d-3

pi = 4d0*datan(1d0)

sol(1) = 0d0
sol(2) = 0d0

jup0(1) = 520d-2
jup0(2) = 0d0

veljup(1) = 0d0
veljup(2) = 4d0*pi/4562d-3

jup1(1) = jup0(1) + veljup(1)*dt
jup1(2) = jup0(2) + veljup(2)*dt

ter0(1) = 1d0
ter0(2) = 0d0

velter(1) = 0d0
velter(2) = (2d0*pi*ter0(1)/1d0)*e(17d-3)

ter1(1) = ter0(1) + velter(1)*dt
ter1(2) = ter0(2) + velter(2)*dt

open(20, file = "saida-B1-1-10799783.dat")
open(21, file = "saida-B1-2-10799783.dat")

do i = 1, 25000

    r = dsqrt((ter1(1) - sol(1))**2 + (ter1(2) - sol(2))**2)
    rtj = dsqrt((ter1(1) - jup(1))**2 + (ter1(2) - jup(2))**2)

    ter(1) = 2d0*ter1(1) - ter0(1)
    &+ d2ft(ter1(1), jup1(1), r, rtj)*(dt**2)
    ter(2) = 2d0*ter1(2) - ter0(2)
    &+ d2ft(ter1(2), jup1(2), r, rtj)*(dt**2)

```



```

        rj = dsqrt((jup1(1) - sol(1))**2 + (jup1(2) - sol(2))**2)

        jup(1) = 2d0*jup1(1) - jup0(1)
&+ d2fj(jup1(1), ter1(1), rj, rtj)*(dt**2)
        jup(2) = 2d0*jup1(2) - jup0(2)
&+ d2fj(jup1(2), ter1(2), rj, rtj)*(dt**2)

        ter0 = ter1

        ter1 = ter

        jup0 = jup1

        jup1 = jup

        write(20, *) ter(1), ter(2)
        write(21, *) jup(1), jup(2)

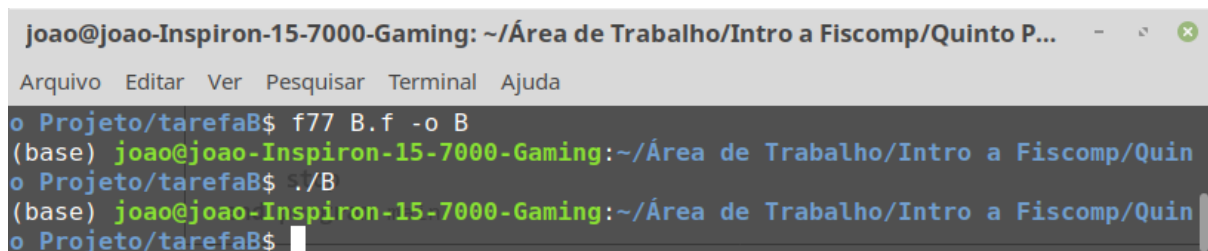
    end do

    close(20)
    close(21)

    stop
end program main

```

Que rodando, rende:



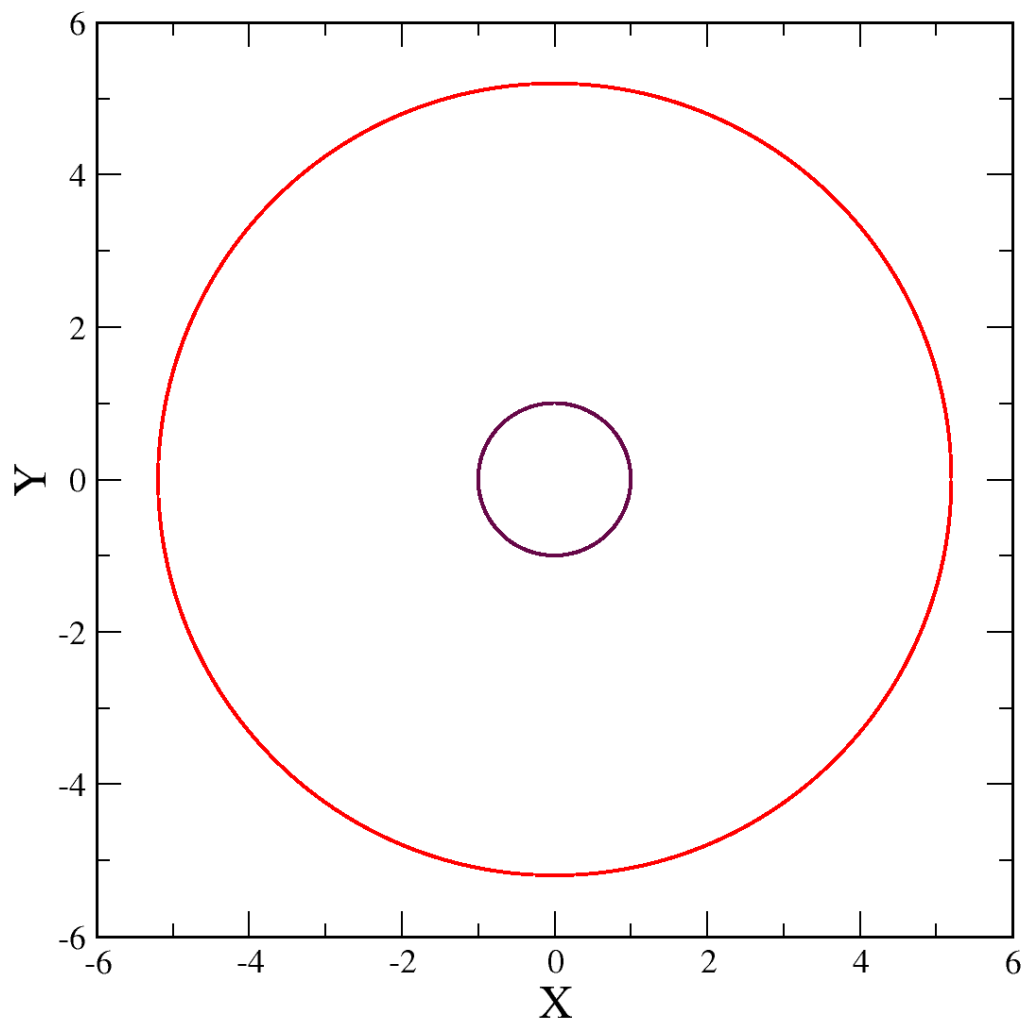
```

joao@joao-Inspiron-15-7000-Gaming: ~/Área de Trabalho/Intro a Fiscomp/Quinto P...
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda
o Projeto/tarefaB$ f77 B.f -o B
(base) joao@joao-Inspiron-15-7000-Gaming:~/Área de Trabalho/Intro a Fiscomp/Quin
o Projeto/tarefaB$ ./B
(base) joao@joao-Inspiron-15-7000-Gaming:~/Área de Trabalho/Intro a Fiscomp/Quin
o Projeto/tarefaB$

```

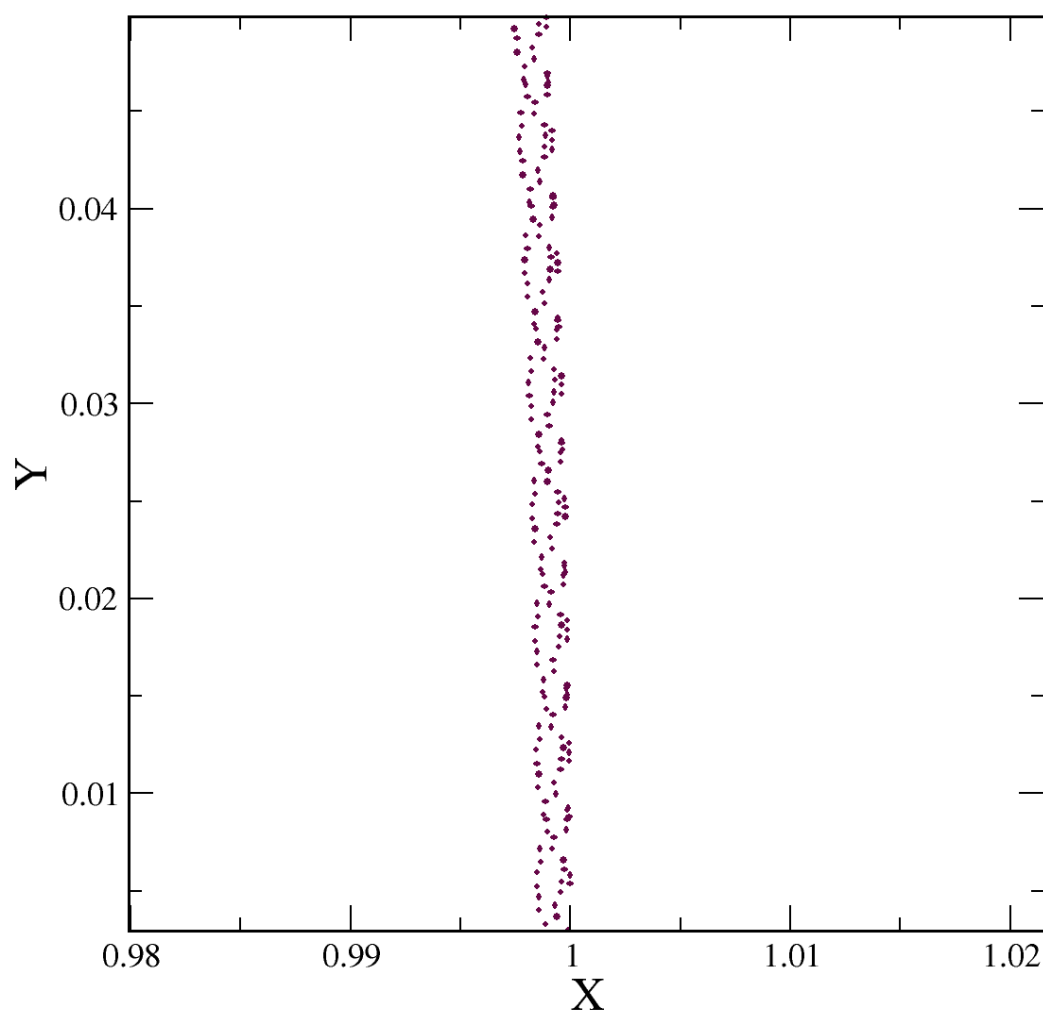
Resultando em dois arquivos de saída, um com as posições da Terra, e outro com as posições de Júpiter. Graficando essas posições, temos:

Posição de Júpiter e Terra



Que à primeira vista não aparenta muita mudança das órbitas da Terra em relação ao cálculo das órbitas sem a influência gravitacional de Júpiter. Porém, se for dado um zoom na órbita da Terra:

Posição de Júpiter e Terra



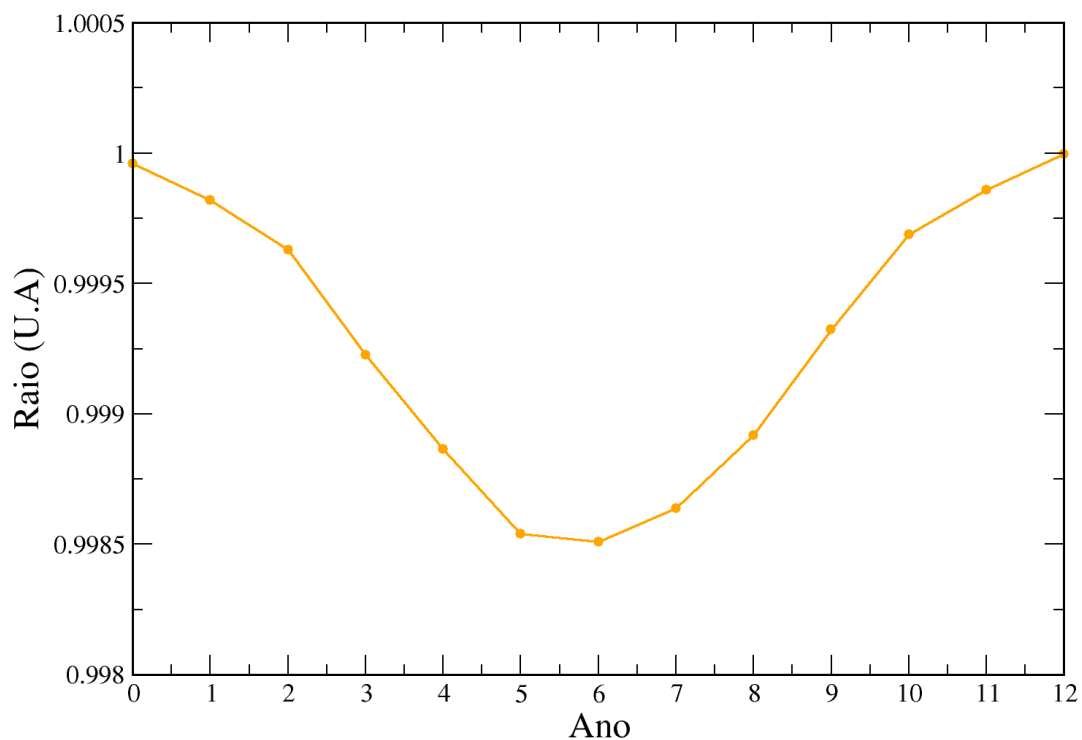
Percebe-se que ela não é mais uma órbita simples e linear, que tem as mesmas posições independente do período completo. Analisando o arquivo com as órbitas da Terra, é possível traçar os raios da órbita da Terra a cada período completo:

Ano	Raio (U.A.)
0	0.99996068971929764
1	0.99982030793357335
2	0.99962842834175869
3	0.99922520143165117

4	0.99886506976581380
5	0.99854015310214428
6	0.99850898940523303
7	0.99863730889823188
8	0.99891838926091026
9	0.99932510692913856
10	0.99968841078622683
11	0.99985823257216599
12	0.99999681974364840

Que, graficando:

Raio da órbita da Terra por ano corrido

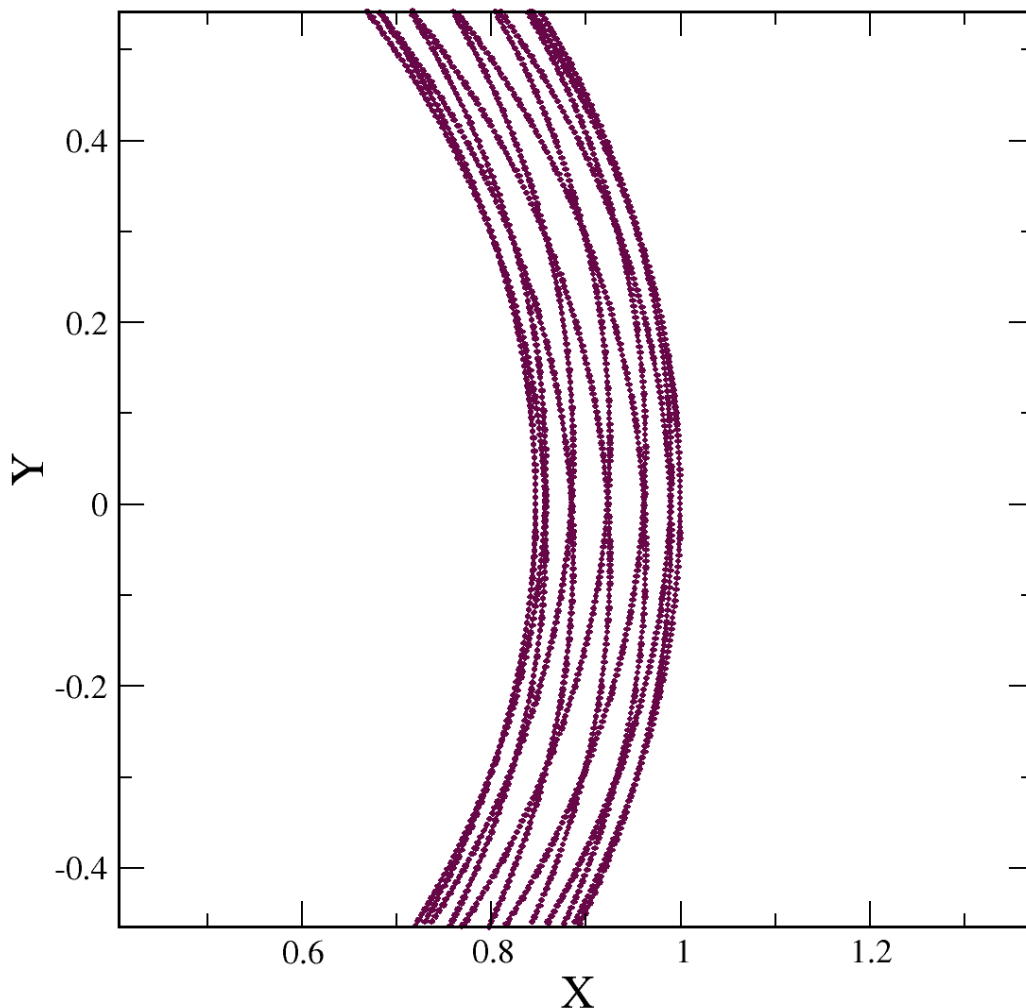


Denuncia um comportamento curioso por parte do raio da órbita da terra, pois demonstra um comportamento oscilatório ao redor de um raio médio, com período de aproximadamente 12 anos.

2.2. Tarefa b2

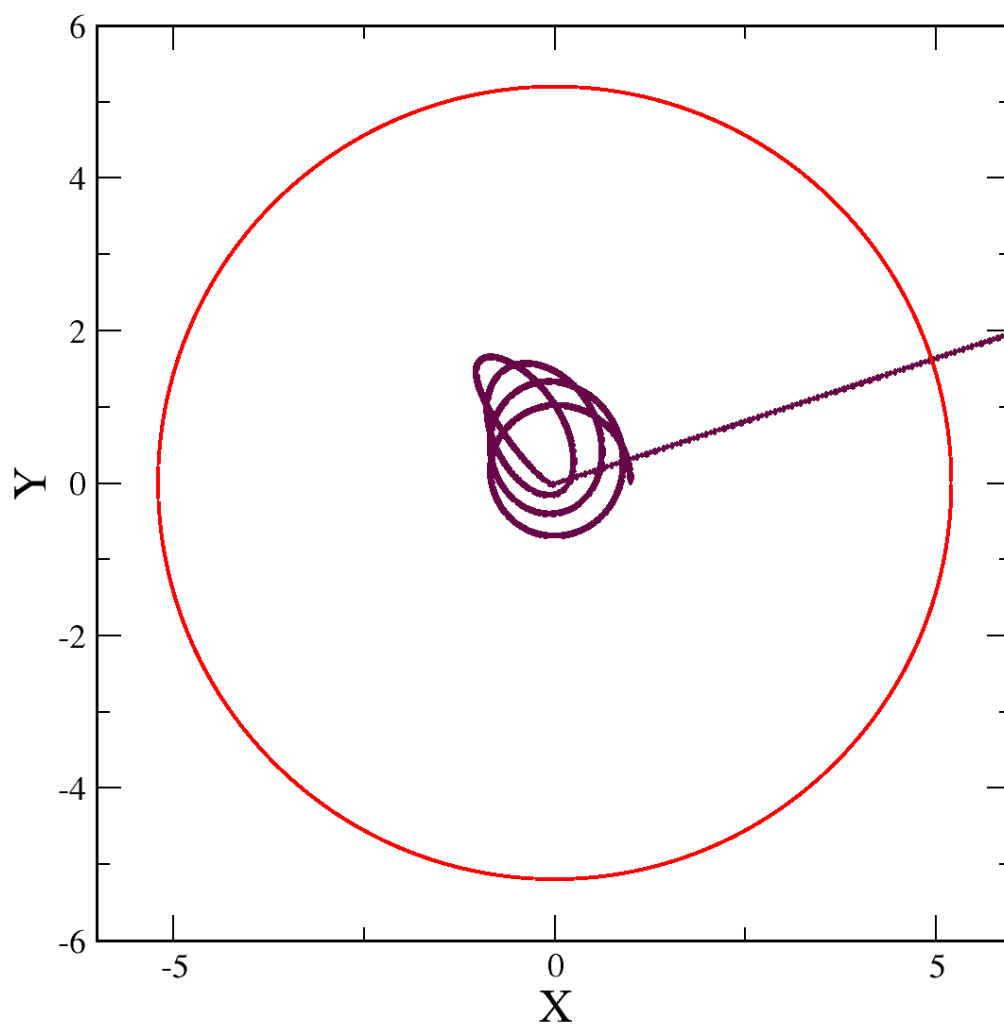
Esta tarefa teve o objetivo de repetir o implementado no programa anterior, porém agora com a massa de Júpiter 100 e 1000 vezes maior. Executando o mesmo programa anterior com a massa de Júpiter 100x maior, temos novamente dois arquivos de saída com as posições da Terra e de Júpiter, que graficando:

Posição de Júpiter e Terra

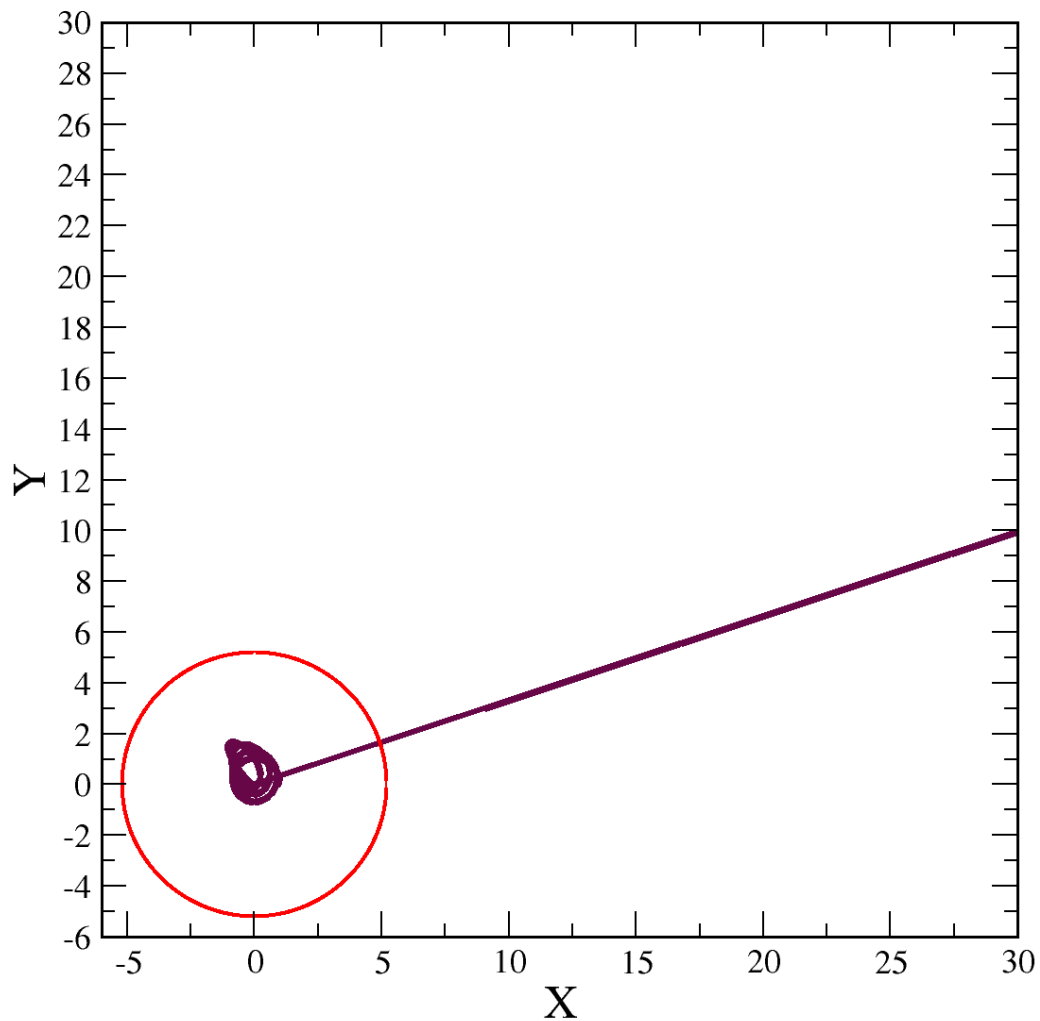


Nos mostra um mesmo comportamento visto no programa anterior, porém mais acentuado, com a Terra variando seu raio orbital médio de maneira mais expressiva. Executando o programa agora com a massa de Júpiter 1000x maior, ou seja, equivalente a massa do Sol, temos:

Posição de Júpiter e Terra



Posição de Júpiter e Terra



Que nos mostra que, com um Júpiter de massa equivalente ao Sol, a Terra é alavancada de sua órbita atual em direção ao infinito.

2.3. Tarefa b3

Essa tarefa teve como objetivo implementar novamente o método de Verlet para um sistema de três ou mais corpos, porém desta vez o cálculo envolveu Júpiter e três asteróides, de valores:

Objeto	raio (UA)	velocidade (UA/ano)
Asteróide I	3.000	3.628
Asteróide II	3.276	3.471
Asteróide III	3.700	3.267

Desta vez, usaremos como velocidade inicial de júpiter $v_0 = 2.755$ U.A./ano. O código que implementa o algoritmo desejado é:

```
double precision function d2ft(x, xj, r, rj)
  implicit double precision (a-h, o-z)

  !Função para calcular a aceleração dos asteroides

  pi = 4d0*datan(1d0)
  Gms = 4d0*(pi**2) !UA**3/ano**2
  GMj = (Gms/1d3)
  GMt = Gms/3d5

  d2ft = -(Gms*x)/(r**3) -GMj*(x-xj)/(rj**3)

  return
end function d2ft

double precision function d2fj(x, r)
  implicit double precision (a-h, o-z)

  !Função para calcular a aceleração de Júpiter

  pi = 4d0*datan(1d0)
  Gms = 4d0*(pi**2) !UA**3/ano**2
  GMj = Gms/1d3
  GMt = Gms/3d5

  d2fj = -(Gms*x)/(r**3)

  return
end function d2fj
```



```

program main
  implicit double precision (a-h, o-z)
  double precision sol(2)
  double precision asterI(2)
  double precision asterI1(2)
  double precision asterI0(2)

  double precision asterII(2)
  double precision asterII1(2)
  double precision asterII0(2)

  double precision asterIII(2)
  double precision asterIII1(2)
  double precision asterIII0(2)

  double precision jup(2)
  double precision jup1(2)
  double precision jup0(2)
  double precision veljup(2)

  dt = 1d-3

  pi = 4d0*datan(1d0)

  sol(1) = 0d0
  sol(2) = 0d0

  jup0(1) = 520d-2
  jup0(2) = 0d0

  veljup(1) = 0d0
  veljup(2) = 2755d-3

  jup1(1) = jup0(1) + veljup(1)*dt
  jup1(2) = jup0(2) + veljup(2)*dt

  asterI0(1) = 3d0
  asterI0(2) = 0d0
  velasterI = 3628d-3

  asterII0(1) = 3276d-3
  asterII0(2) = 0d0
  velasterII = 3471d-3

  asterIII0(1) = 37d-1
  asterIII0(2) = 0d0
  velasterIII = 3267d-3

```

```

asterI1(1) = asterI0(1) + 0d0*dt
asterI1(2) = asterI0(2) + velasterI*dt

asterII1(1) = asterII0(1) + 0d0*dt
asterII1(2) = asterII0(2) + velasterII*dt

asterIII1(1) = asterIII0(1) + 0d0*dt
asterIII1(2) = asterIII0(2) + velasterIII*dt


open(20, file = "saida-B3-1-10799783.dat")
open(21, file = "saida-B3-2-10799783.dat")
open(22, file = "saida-B3-3-10799783.dat")
open(23, file = "saida-B3-4-10799783.dat")

do i = 1, 25000

    r = dsqrt((asterI1(1) - sol(1))**2
&+ (asterI1(2) - sol(2))**2)
    rIj = dsqrt((asterI1(1) - jup(1))**2
&+ (asterI1(2) - jup(2))**2)

    asterI(1) = 2d0*asterI1(1) - asterI0(1)
&+ d2ft(asterI1(1), jup1(1), r, rIj)*(dt**2)
    asterI(2) = 2d0*asterI1(2) - asterI0(2)
&+ d2ft(asterI1(2), jup1(2), r, rIj)*(dt**2)

    r = dsqrt((asterII1(1) - sol(1))**2
&+ (asterII1(2) - sol(2))**2)
    rIIj = dsqrt((asterII1(1) - jup(1))**2
&+ (asterII1(2) - jup(2))**2)

    asterII(1) = 2d0*asterII1(1) - asterII0(1)
&+ d2ft(asterII1(1), jup1(1), r, rIIj)*(dt**2)
    asterII(2) = 2d0*asterII1(2) - asterII0(2)
&+ d2ft(asterII1(2), jup1(2), r, rIIj)*(dt**2)

    r = dsqrt((asterIII1(1) - sol(1))**2
&+ (asterIII1(2) - sol(2))**2)
    rIIIj = dsqrt((asterIII1(1) - jup(1))**2

```

```

&+ (asterIII1(2) - jup(2))**2)

    asterIII(1) = 2d0*asterIII1(1) - asterIII0(1)
&+ d2ft(asterIII1(1), jup1(1), r, rIIIj)*(dt**2)
    asterIII(2) = 2d0*asterIII1(2) - asterIII0(2)
&+ d2ft(asterIII1(2), jup1(2), r, rIIIj)*(dt**2)

    rj = dsqrt((jup1(1) - sol(1))**2 + (jup1(2) - sol(2))**2)

    jup(1) = 2d0*jup1(1) - jup0(1)
&+ d2fj(jup1(1), rj)*(dt**2)
    jup(2) = 2d0*jup1(2) - jup0(2)
&+ d2fj(jup1(2), rj)*(dt**2)

    asterI0 = asterI1

    asterI1 = asterI

    asterII0 = asterIII

    asterIII1 = asterII

    asterIII0 = asterIII1

    asterIII1 = asterIII

    jup0 = jup1

    jup1 = jup

    write(20, *) asterI(1), asterI(2)
    write(21, *) asterII(1), asterII(2)
    write(22, *) asterIII(1), asterIII(2)
    write(23, *) jup(1), jup(2)

end do

close(20)
close(21)
close(22)
close(23)

stop
end program main

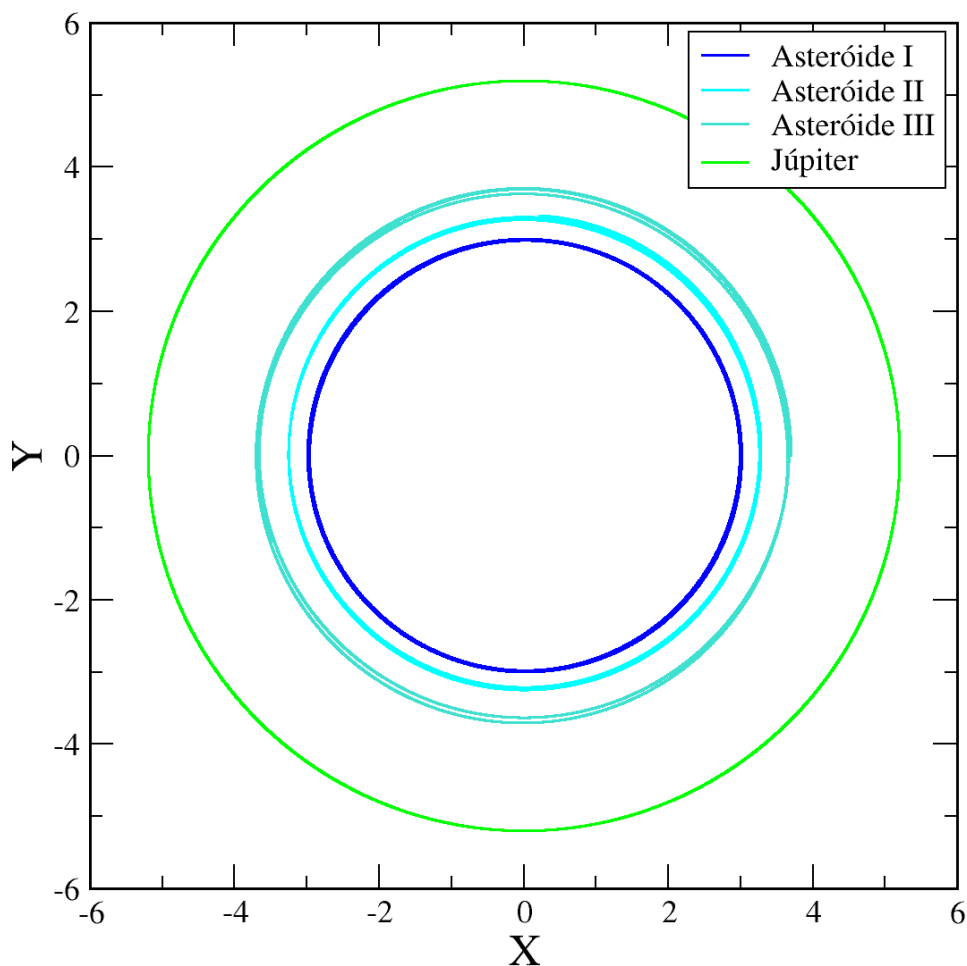
```

Rodando o código, temos:

```
joao@joao-Inspiron-15-7000-Gaming: ~/Área de Trabalho/Intro a Fiscomp/Quinto P...
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda
o Projeto/tarefaB$ f77 B3.f -o B3
(base) joao@joao-Inspiron-15-7000-Gaming:~/Área de Trabalho/Intro a Fiscomp/Quin
o Projeto/tarefaB$ ./B3
(base) joao@joao-Inspiron-15-7000-Gaming:~/Área de Trabalho/Intro a Fiscomp/Quin
o Projeto/tarefaB$
```

Que nos rende quatro arquivos de saída com as posições, um para cada corpo. Graficando as posições, temos:

Posições de Júpiter e dos Asteróides



Este resultado demonstra um comportamento já conhecido como Lacunas de Kirkwood³. As Lacunas de Kirkwood são espaços menos densos, relativamente vazios, do cinturão de asteroides entre a órbita da Terra e de Júpiter. Pelos

³ https://pt.wikipedia.org/wiki/Lacunas_de_Kirkwood

resultados, é possível inferir que os três asteroides estão localizados um em cada segmentação das Lacunas, como é possível conferir a seguir:

