

Universidade de São Paulo  
Instituto de Física de São Carlos

## **Relatório 5 - IntroFisComp**

Alexandre de Taunay Voloch

# 1 Tarefa 1

## 1.1 a

Para que a órbita seja circular, precisamos que ela satisfaça a equação do movimento circular, isto é,

$$a = \frac{v^2}{r}$$

Temos  $F = ma$ . Logo,

$$a = \ddot{\rho} = \frac{1}{\rho^2} = \frac{v^2}{\rho}$$

Logo

$$v_0 = \frac{1}{\sqrt{\rho_0}}$$

Como foi dado que  $\rho_0 = (a, 0)$ , a velocidade deve apontar apenas na direção  $y$ . Logo,  $v_0 = (0, \frac{1}{\sqrt{a}})$ .

## 1.2 b

Segue o programa:

```
1      implicit real*8 (a-h, o-z)
2      real*8 p_ec(10000000,2)
3      real*8 v_ec(10000000,2)
4      real*8 p_verlet(10000000,2)
5      real*8 v_verlet(10000000,2)
6
7      pi = 4.d0*datan2(1.d0,1.d0)
8
9      tau_total = 10d0
10     dtau = 1d-4
11     iteracoes = int(tau_total/dtau)
12     a = 1d0
13
14     open(file='tarefa-1-saida-pec.dat', unit=1)
15     open(file='tarefa-1-saida-pverlet.dat', unit=3)
16     open(file='tarefa-1-saida-vec.dat', unit=2)
17     open(file='tarefa-1-saida-verlet.dat', unit=4)
18
```

```

19     p_ec(1,1) = a
20     p_ec(1,2) = 0d0
21     v_ec(1,1) = 0d0
22     v_ec(1,2) = 1d0/dsqrt(a)
23
24     p_verlet(1,1) = a
25     p_verlet(1,2) = 0d0
26
27     do i=2,iteracoes
28         p3 = pcubo(p_ec(i-1,1), p_ec(i-1,2))
29
30         v_ec(i,1) = v_ec(i-1,1) - (p_ec(i-1,1)/p3)*dtau
31         v_ec(i,2) = v_ec(i-1,2) - (p_ec(i-1,2)/p3)*dtau
32
33         p_ec(i,1) = p_ec(i-1,1) + v_ec(i,1)*dtau
34         p_ec(i,2) = p_ec(i-1,2) + v_ec(i,2)*dtau
35
36         write(1,*)p_ec(i,1),p_ec(i,2)
37         write(2,*)v_ec(i,1),v_ec(i,2)
38
39         !verlet
40         if (i.eq.2) then
41             ! pular verlet, usar os valores de euler-cromer
42             p_verlet(i,1) = p_ec(i,1)
43             p_verlet(i,2) = p_ec(i,2)
44         else
45             p3 = pcubo(p_verlet(i-1,1),p_verlet(i-1,2))
46
47             p_verlet(i,1) = 2d0*p_verlet(i-1,1) -
48                 ↪ p_verlet(i-2,1)
49             & - (p_verlet(i-1,1)/p3)*(dtau**2)
50             p_verlet(i,2) = 2d0*p_verlet(i-1,2) -
51             & ↪ p_verlet(i-2,2)
52             - (p_verlet(i-1,2)/p3)*(dtau**2)
53         endif
54
55         write(3,*)p_verlet(i,1), p_verlet(i,2)
56
57     end do
58
59     ! agora calcular as velocidades em Verlet
60     ! vamos usar a derivada simétrica de 5 pontos
61     do i=3,iteracoes-2
62         ! começamos do 3 pq precisamos ter i-2 e i-1 válidos

```

```

61         v_verlet(i,1) = (p_verlet(i-2,1) - 8d0*p_verlet(i-1,1)
62 &         + 8d0*p_verlet(i+1,1) - p_verlet(i+2,1)) / (12d0*dtau)
63         v_verlet(i,2) = (p_verlet(i-2,2) - 8d0*p_verlet(i-1,2)
64 &         + 8d0*p_verlet(i+1,2) - p_verlet(i+2,2)) / (12d0*dtau)
65
66         write(4,*)v_verlet(i,1), v_verlet(i,2)
67     end do
68
69     end
70
71     function pcubo(px,py)
72         real*8 px, py, pcubo
73         pcubo = (px**2d0 + py**2d0)**(3d0/2d0)
74     end function
75

```

Simulando a órbita da terra (isto é, com  $\rho_0 = 1 = v_0$ ) temos

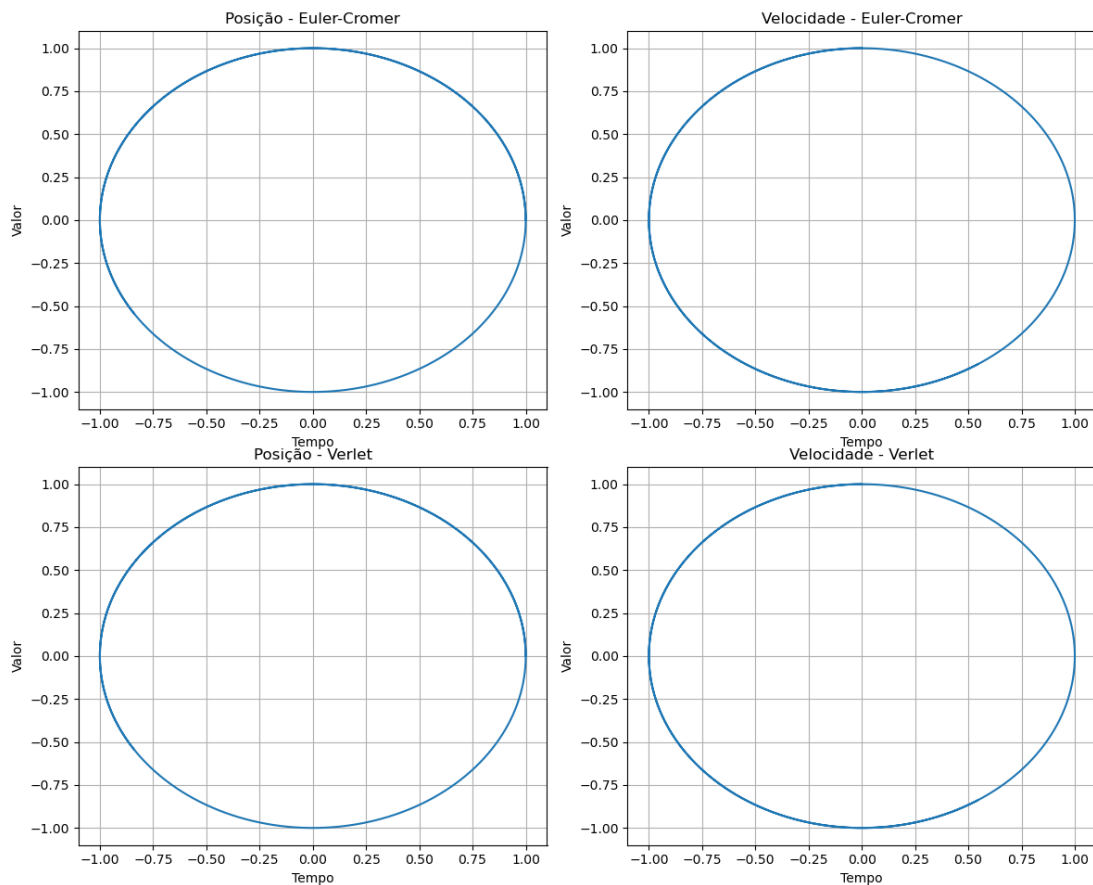


Figura 1: Gráfico da órbita da terra e velocidade calculada nos diferentes métodos.

Como podemos ver, todos os gráficos são praticamente idênticos. Isso é de se esperar, pois a velocidade e a posição têm o mesmo módulo a todo momento, e por ser movimento circular vão apenas oscilando em um círculo, embora com fases diferentes (caso fizéssemos uma animação, veríamos isso).

### 1.3 c

Aqui usamos o seguinte programa para fazer os gráficos:

```

1      implicit real*8 (a-h, o-z)
2      real*8 p_ec(10000000,2)
3      real*8 v_ec(10000000,2)
4      real*8 p_verlet(10000000,2)
5      real*8 v_verlet(10000000,2)
6      real*8 m_planetas(8), a_planetas(8)
7      real*8 d_planetas(800,100000,2) ! armazena (dtau, delta)
      ↪ respectivo pra aquele planeta
8      data m_planetas / 0.055d0, 0.817d0, 1.00d0, 0.107d0,
9      &318d0, 95.2d0, 14.5d0, 17.1d0 /
10     data a_planetas / 0.39d0, 0.72d0, 1.00d0, 1.52d0,
11     &5.20d0, 9.58d0, 19.2d0, 30.1d0/
12     ! data a_planetas / 0.39d0, 0.72d0, 1.00d0, 1.52d0,
13     ! &5.20d0, 6.58d0, 8.2d0, 9.0d0/
14
15     pi = 4.d0*datan2(1.d0,1.d0)
16
17     open(file='tarefa-1c-saida-dec.dat', unit=1)
18     open(file='tarefa-1c-saida-dverlet.dat', unit=2)
19     open(file='dtaumax.dat', unit=3)
20
21     ! iterar nos planetas
22     !do ip=5,60
23     do ip=3,4
24         !a = dble(ip)*0.1d0*a_planetas(3)
25         write(*,*) "Planeta",ip,"a",a
26         a = a_planetas(ip)
27
28         p_ec(1,1) = a
29         p_ec(1,2) = 0d0
30         v_ec(1,1) = 0d0
31         v_ec(1,2) = 1d0/dsqrt(a)
32
33         p_verlet(1,1) = a
34         p_verlet(1,2) = 0d0

```

```

35
36     iquanto=20
37
38     iter_dtau = int(3.5*iquanto)
39
40     !tau_total = 50d0
41     tau_total = 20d0*a
42
43     do j=1,iter_dtau
44         potencia = dble(2*iquanto +j)/dble(iquanto)
45         dtau = 10**(-potencia)
46         iteracoes = int(tau_total/dtau)
47
48         write(*,*) "dtau",dtau,j
49         pmax_ec = 0d0
50         pmin_ec = 10000000d0
51         pmax_v = 0d0
52         pmin_v = 10000000d0
53
54         do i=2,iteracoes
55             p3 = pcubo(p_ec(i-1,1), p_ec(i-1,2))
56
57             v_ec(i,1) = v_ec(i-1,1) - (p_ec(i-1,1)/p3)*dtau
58             v_ec(i,2) = v_ec(i-1,2) - (p_ec(i-1,2)/p3)*dtau
59
60             p_ec(i,1) = p_ec(i-1,1) + v_ec(i,1)*dtau
61             p_ec(i,2) = p_ec(i-1,2) + v_ec(i,2)*dtau
62
63             !verlet
64             if (i.eq.2) then
65                 ! pular verlet, usar os valores de euler-cromer
66                 p_verlet(i,1) = p_ec(i,1)
67                 p_verlet(i,2) = p_ec(i,2)
68             else
69                 p3 = pcubo(p_verlet(i-1,1),p_verlet(i-1,2))
70
71                 p_verlet(i,1) = 2d0*p_verlet(i-1,1) - p_verlet(i-2,1)
72             &         - (p_verlet(i-1,1)/p3)*(dtau**2)
73                 p_verlet(i,2) = 2d0*p_verlet(i-1,2) - p_verlet(i-2,2)
74             &         - (p_verlet(i-1,2)/p3)*(dtau**2)
75             endif
76
77             ! calcular p
78             p_modulo_ec = dsqrt(p_ec(i,1)**2d0 + p_ec(i,2)**2d0)

```

```

79      p_modulo_v = dsqrt(p_verlet(i,1)**2d0 +
      ↪      p_verlet(i,2)**2d0)
80
81      if (p_modulo_ec.gt.pmax_ec) then
82          pmax_ec = p_modulo_ec
83      else if (p_modulo_ec.lt.pmin_ec) then
84          pmin_ec = p_modulo_ec
85      end if
86      if (p_modulo_v.gt.pmax_v) then
87          pmax_v = p_modulo_v
88      else if (p_modulo_v.lt.pmin_v) then
89          pmin_v = p_modulo_v
90
91      endif
92
93
94  end do
95
96      !write(*,*) "Pmax e min:", pmax_ec,pmin_ec,pmax_v,pmin_v
97
98      d_ec = pmax_ec/pmin_ec - 1
99      d_v = pmax_v/pmin_v - 1
100
101
102      ! escrever no grafico valores p/ terra
103
104      if (ip.eq.3) then
105          write(1,*)potencia,d_ec
106          write(2,*)potencia,d_v
107      endif
108
109      d_planetas(ip,j,1) = dtau
110      d_planetas(ip,j,2) = d_ec
111
112      write(*,*) "d ec:",d_ec,"d verlet:",d_v
113
114      !if(d_ec.lt.1d-3) then
115      !    goto 20
116      !endif
117
118      end do
119      20      continue
120  end do
121

```

```

122      ! agora processar os dados de d_planetas para fazer o gráfico
      ⇨ de dtau_min x a
123      write(*,*) "processando"
124      !do ip=5,60
125      !    write(*,*) "planeta",ip
126      !    do j=1,iter_dtau
127      !      if(d_planetas(ip,j,2).le.1d-3)then
128      !        write(*,*) "saimos"
129      !        goto 10
130      !      endif
131      !    end do
132      !10    !
      ⇨ write(3,*)dble(ip)*0.1d0*a_planetas(3),d_planetas(ip,j,1),j
133
134      !end do
135
136      end
137
138      function pcubo(px,py)
139      real*8 px, py, pcubo
140      pcubo = (px**2d0 + py**2d0)**(3d0/2d0)
141      end function
142

```

Primeiro, fazemos o gráfico de  $\delta$  em função de  $\Delta\tau$ . Obtemos o seguinte:



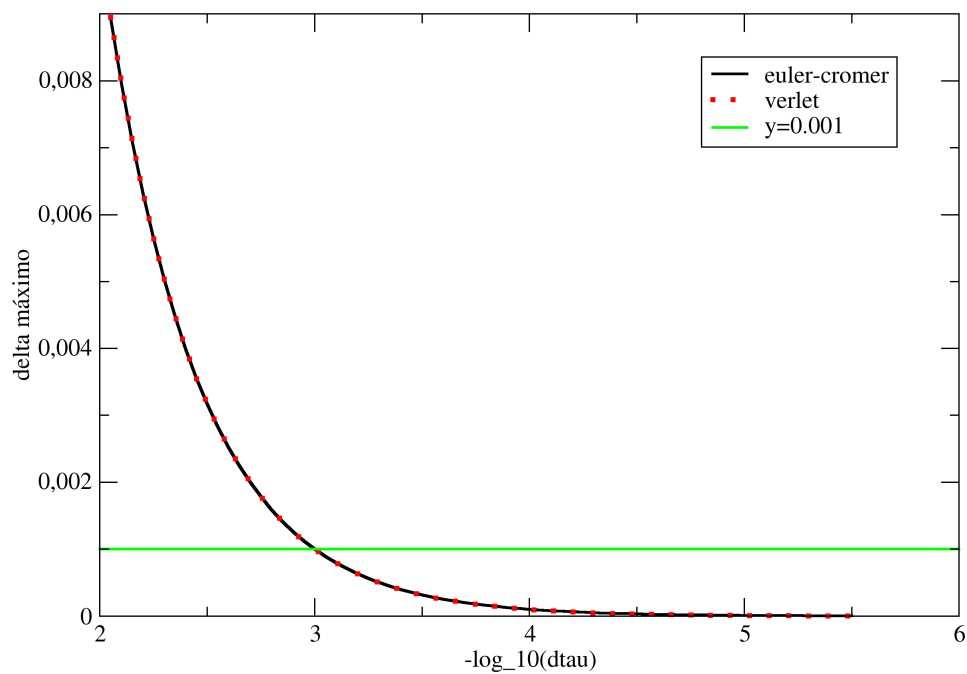


Figura 2: Gráfico de  $\delta$  em função de  $\Delta\tau$ , para a órbita da Terra

Verificamos que, para a órbita da Terra em específico, o delta máximo é praticamente igual ao  $\Delta\tau$ . Portanto, se queremos  $\delta < 10^{-3}$ , este também deve ser o valor de  $\Delta\tau$ .

Agora, fazendo o gráfico para diversos raios diferentes (para isso simulamos vários planetas no nosso programa), obtemos o seguinte:

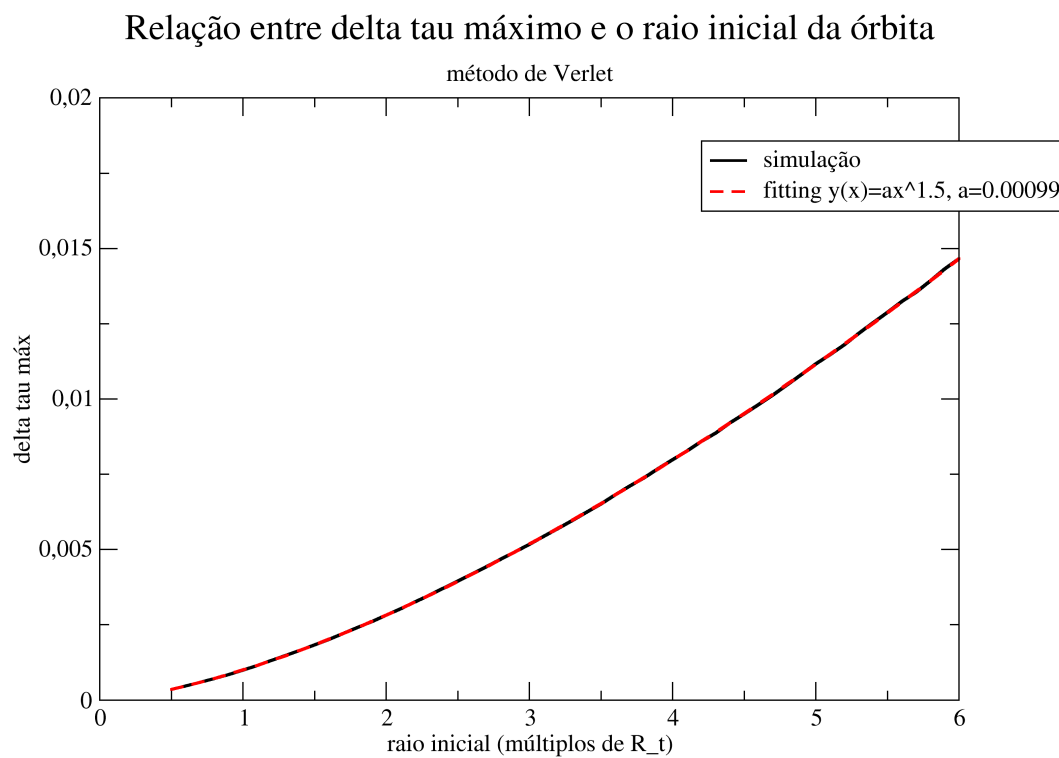


Figura 3: Gráfico de  $\Delta\tau_{max}$  em função do raio inicial da órbita - Verlet

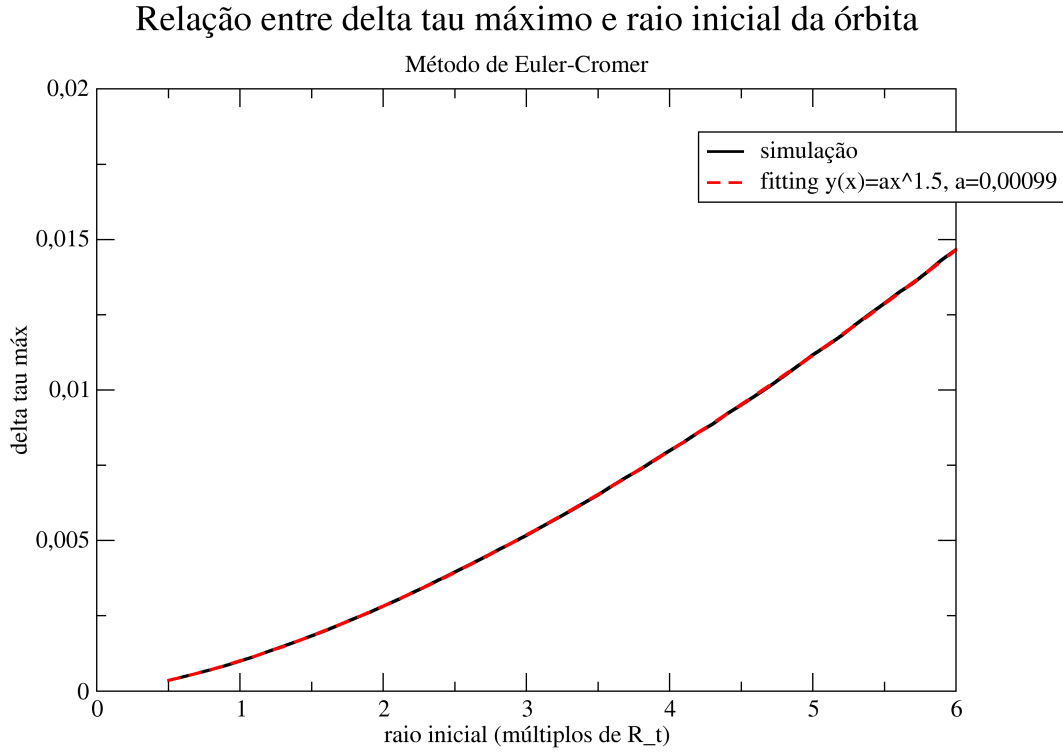


Figura 4: Gráfico de  $\Delta\tau_{max}$  em função do raio inicial da órbita - Euler-Cromer

Ambas as figuras são praticamente idênticas. Percebemos que a curva é fitada perfeitamente por uma função que é proporcional a  $a^{3/2}$ , o que confirma a hipótese feita no projeto.

Para explicar o motivo dessa proporcionalidade, precisamos assumir que  $\Delta\tau_{max} \propto T$ , onde  $T$  nesse caso representa o período de órbita (em unidades de  $\tau$ ). Assim, como a 3a lei de Kepler nos diz que  $T \propto a^{3/2}$ , a proporcionalidade também valerá para qualquer coisa proporcional a  $T$ .

## 1.4 d

Aqui utilizamos um programa muito parecido aos anteriores, mas que calcula a energia. Para chegar na expressão da energia adimensional, fazemos:

$$E = \frac{m\dot{r}^2}{2} + \frac{GM_s m}{r}$$

$$\begin{aligned}
&= \frac{m}{2} \frac{dr}{dt} + \frac{GM_s m}{r} = \frac{m}{2} \left( \frac{UA d\rho}{\frac{ano}{2\pi} d\tau} \right)^2 + \frac{GM_s m}{UA \rho} \\
&= \frac{4\pi^2 m (UA)^2 \dot{\rho}^2}{(ano)^2} + \frac{GM_s m}{UA \rho} \\
&= \frac{GM_s m}{UA} \left( \frac{\dot{\rho}^2}{2} + \frac{1}{\rho} \right)
\end{aligned}$$

Logo, temos nossa expressão para energia adimensional:

$$\epsilon = \frac{\dot{\rho}^2}{2} + \frac{1}{\rho}$$

```

1      implicit real*8 (a-h, o-z)
2      real*8 p_ec(10000000,2)
3      real*8 v_ec(10000000,2)
4      real*8 p_verlet(10000000,2)
5      real*8 v_verlet(10000000,2)
6
7      pi = 4.d0*datan2(1.d0,1.d0)
8
9      tau_total = 200d0
10     dtau = 1d-3
11     iteracoes = int(tau_total/dtau)
12     a = 1d0
13
14     open(file='energia-ec.dat',unit=1)
15     open(file='energia-v.dat',unit=2)
16
17     p_ec(1,1) = a
18     p_ec(1,2) = 0d0
19     v_ec(1,1) = 0d0
20     v_ec(1,2) = 1d0/dsqrt(a)
21
22     p_verlet(1,1) = a
23     p_verlet(1,2) = 0d0
24
25     do i=2,iteracoes
26         tempo = dble(i)*dtau
27         p3 = pcubo(p_ec(i-1,1), p_ec(i-1,2))
28
29         v_ec(i,1) = v_ec(i-1,1) - (p_ec(i-1,1)/p3)*dtau
30         v_ec(i,2) = v_ec(i-1,2) - (p_ec(i-1,2)/p3)*dtau
31

```

```

32      p_ec(i,1) = p_ec(i-1,1) + v_ec(i,1)*dtau
33      p_ec(i,2) = p_ec(i-1,2) + v_ec(i,2)*dtau
34
35      !energia - euler-cromer
36      p_ec_modulo = dsqrt(p_ec(i,1)**2d0 + p_ec(i,2)**2d0)
37      v_ec_modulo = dsqrt(v_ec(i,1)**2d0 + v_ec(i,2)**2d0)
38
39      e_ec = 0.5d0*(v_ec_modulo**2d0) + (1d0/p_ec_modulo)
40      write(1,*) tempo, e_ec
41
42      !verlet
43      if (i.eq.2) then
44          ! pular verlet, usar os valores de euler-cromer
45          p_verlet(i,1) = p_ec(i,1)
46          p_verlet(i,2) = p_ec(i,2)
47      else
48          p3 = pcubo(p_verlet(i-1,1),p_verlet(i-1,2))
49
50          p_verlet(i,1) = 2d0*p_verlet(i-1,1) -
51      &          ↪ p_verlet(i-2,1)
52          - (p_verlet(i-1,1)/p3)*(dtau**2)
53      &          p_verlet(i,2) = 2d0*p_verlet(i-1,2) -
54          ↪ p_verlet(i-2,2)
55          - (p_verlet(i-1,2)/p3)*(dtau**2)
56      endif
57
58      end do
59
60      ! agora calcular as velocidades em Verlet
61      ! vamos usar a derivada simétrica de 5 pontos
62      do i=3,iteracoes-2
63          tempo = dble(i)*dtau
64
65          ! começamos do 3 pq precisamos ter i-2 e i-1 válidos
66          v_verlet(i,1) = (p_verlet(i-2,1) - 8d0*p_verlet(i-1,1)
67      &          + 8d0*p_verlet(i+1,1) - p_verlet(i+2,1)) / (12d0*dtau)
68          v_verlet(i,2) = (p_verlet(i-2,2) - 8d0*p_verlet(i-1,2)
69      &          + 8d0*p_verlet(i+1,2) - p_verlet(i+2,2)) / (12d0*dtau)
70
71          ! energia
72          p_v_modulo = dsqrt(p_verlet(i,1)**2d0 +
73          ↪ p_verlet(i,2)**2d0)
74          v_v_modulo = dsqrt(v_verlet(i,1)**2d0 +
75          ↪ v_verlet(i,2)**2d0)

```

```

72
73         e_v = 0.5d0*(v_v_modulo**2d0) + (1d0/p_v_modulo)
74         write(2,*) tempo, e_v
75     end do
76
77 end
78
79 function pcubo(px,py)
80     real*8 px, py, pcubo
81     pcubo = (px**2d0 + py**2d0)**(3d0/2d0)
82 end function
83

```

Graficando os resultados, temos

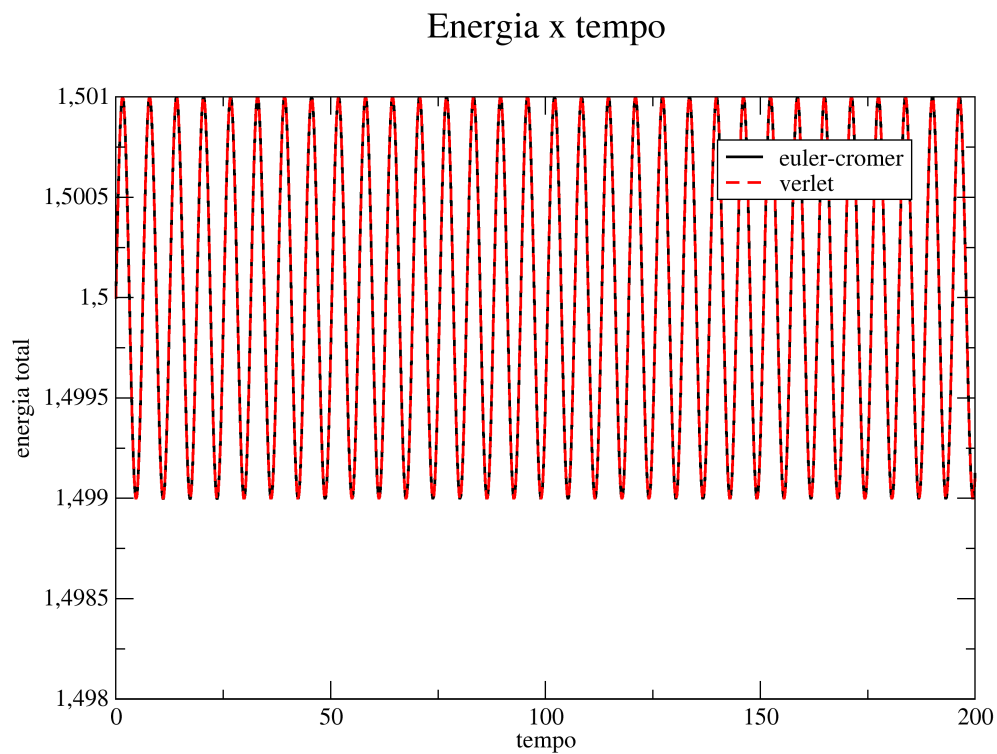


Figura 5: Gráfico da energia adimensional

Percebemos que a energia vai variando bem pouco ao longo do tempo, mas permanece ao redor de 1.5, o que é de se esperar pois estamos simulando a órbita da Terra,

onde  $\epsilon$  de fato vale 1.5. Algo notável é que não percebemos nenhuma diferença entre o método de Euler-Cromer e o de Verlet nessa simulação, como podemos ver no gráfico.

## 2 Tarefa 2

### 2.1 a

Aqui vamos nos limitar a uma análise da órbita de Marte, cuja excentricidade já é conhecida. Temos dois programas, um para as primeiras duas leis e outro para a terceira lei. Seguem os ditos cujos:

```
1      implicit real*8 (a-h, o-z)
2      real*8 p_ec(10000000,2)
3      real*8 v_ec(10000000,2)
4      real*8 p_verlet(10000000,2)
5      real*8 v_verlet(10000000,2)
6
7      pi = 4.d0*datan2(1.d0,1.d0)
8
9      tau_total = 10d0
10     dtau = 1d-4
11     iteracoes = int(tau_total/dtau)
12     a = 1d0
13     a_marte = 1.662d0*a
14     v_marte = 0.739d0
15
16     open(file='orbita.dat', unit=1)
17     open(file='1alei.dat', unit=2)
18     open(file='2alei.dat', unit=3)
19
20     p_ec(1,1) = 0d0
21     p_ec(1,2) = a
22     v_ec(1,1) = v_marte
23     v_ec(1,2) = 0d0
24
25     p_verlet(1,1) = p_ec(1,1)
26     p_verlet(1,2) = p_ec(1,2)
27
28     ymin = 10000d0
29     ymax = 0d0
30
31     ! para verificar se a órbita é elíptica, vamos descobrir
    ↪ qual é o y máximo e o y mínimo
```

```

32      ! esse segundo será a distância do foco do sol, a um dos
33      ↪ cantos da elipse
34      ! o primeiro, subtraído pelo segundo, será a distância da
35      ↪ origem até o segundo foco
36      ! tendo isso, podemos verificar a 1a lei de kepler
37
38      do i=2,iteracoes
39          tempo = dble(i)*dtau
40          p3 = pcubo(p_ec(i-1,1), p_ec(i-1,2))
41
42          !verlet
43          if (i.eq.2) then
44              ! pular verlet, usar os valores de euler-cromer
45              v_ec(i,1) = v_ec(i-1,1) - (p_ec(i-1,1)/p3)*dtau
46              v_ec(i,2) = v_ec(i-1,2) - (p_ec(i-1,2)/p3)*dtau
47
48              p_ec(i,1) = p_ec(i-1,1) + v_ec(i,1)*dtau
49              p_ec(i,2) = p_ec(i-1,2) + v_ec(i,2)*dtau
50
51              p_verlet(i,1) = p_ec(i,1)
52              p_verlet(i,2) = p_ec(i,2)
53          else
54              p3 = pcubo(p_verlet(i-1,1),p_verlet(i-1,2))
55
56              p_verlet(i,1) = 2d0*p_verlet(i-1,1) - p_verlet(i-2,1)
57              & - (p_verlet(i-1,1)/p3)*(dtau**2)
58              p_verlet(i,2) = 2d0*p_verlet(i-1,2) - p_verlet(i-2,2)
59              & - (p_verlet(i-1,2)/p3)*(dtau**2)
60          endif
61
62          write(1,*)p_verlet(i,1), p_verlet(i,2)
63
64          if (p_verlet(i,2).lt.ymin) then
65              ymin = p_verlet(i,2)
66          end if
67          if (p_verlet(i,2).gt.ymax) then
68              ymax = p_verlet(i,2)
69          end if
70
71          area = 0.5d0 * abs(p_verlet(i-1,1)*p_verlet(i,2)
72              & - p_verlet(i,1)*p_verlet(i-1,2))
73          write(3,*)tempo, area

```



```

74     end do
75
76
77     foco = ymax - abs(ymin)
78     write(*,*)ymin,ymax, foco
79
80     ! agora verificar a 1a lei de kepler: calcular as distâncias
81     ↪ em cada ponto a cada um dos focos
82     do i=1,iteracoes
83         tempo = dble(i)*dtau
84         d1 = dsqrt(p_verlet(i,1)**2d0 + p_verlet(i,2)**2d0)
85         d2 = dsqrt(p_verlet(i,1)**2d0 + (foco -
86             ↪ p_verlet(i,2))**2d0)
87         dist = d1+d2
88         write(2,*)tempo,dist
89     end do
90
91     ! agora calcular as velocidades em Verlet
92     ! vamos usar a derivada simétrica de 5 pontos
93     !do i=3,iteracoes-2
94     !     ! começamos do 3 pq precisamos ter i-2 e i-1 válidos
95     !     v_verlet(i,1) = (p_verlet(i-2,1) - 8d0*p_verlet(i-1,1)
96     & !     + 8d0*p_verlet(i+1,1) - p_verlet(i+2,1)) / (12d0*dtau)
97     !     v_verlet(i,2) = (p_verlet(i-2,2) - 8d0*p_verlet(i-1,2)
98     & !     + 8d0*p_verlet(i+1,2) - p_verlet(i+2,2)) / (12d0*dtau)
99
100     ! write(4,*)v_verlet(i,1), v_verlet(i,2)
101 !end do
102
103
104 end
105
106 function pcubo(px,py)
107     real*8 px, py, pcubo
108     pcubo = (px**2d0 + py**2d0)**(3d0/2d0)
109 end function
110
111
112 implicit real*8 (a-h, o-z)
113 real*8 p_ec(10000000,2)
114 real*8 v_ec(10000000,2)
115 real*8 p_verlet(10000000,2)
116 real*8 v_verlet(10000000,2)
117
118 pi = 4.d0*datan2(1.d0,1.d0)

```

```

8
9      tau_total = 10d0
10     dtau = 1d-5
11
12     a = 1d0
13
14     p_ec(1,1) = 0d0
15     p_ec(1,2) = a
16     v_ec(1,1) = 1d0/dsqrt(a)
17     v_ec(1,2) = 0d0
18
19     p_verlet(1,1) = p_ec(1,1)
20     p_verlet(1,2) = p_ec(1,2)
21
22     open(file='3alei.dat',unit=1)
23
24     ! para verificar a 3a lei, vamos usar um exemplo
25     ↪ simplificado para órbitas circulares
26     ! e ir iterando em planetas imaginários até 5ua de raio
27
28     do ip=1,10
29         a = 0.5d0*dble(ip)
30         p_ec(1,1) = a
31         p_ec(1,2) = 0d0
32         v_ec(1,1) = 0d0
33         v_ec(1,2) = 1d0/dsqrt(a)
34
35         p_verlet(1,1) = a
36         p_verlet(1,2) = 0d0
37
38         ! calcular periodo
39         tau_total = 20d0*a
40         iteracoes = int(tau_total/dtau)
41
42         tempo_periodo = 0d0
43
44     do i=2,iteracoes
45         tempo = dble(i)*dtau
46         p3 = pcubo(p_ec(i-1,1), p_ec(i-1,2))
47
48         !verlet
49         if (i.eq.2) then
50             ! pular verlet, usar os valores de euler-cromer
51             v_ec(i,1) = v_ec(i-1,1) - (p_ec(i-1,1)/p3)*dtau

```

```

51         v_ec(i,2) = v_ec(i-1,2) - (p_ec(i-1,2)/p3)*dtau
52
53         p_ec(i,1) = p_ec(i-1,1) + v_ec(i,1)*dtau
54         p_ec(i,2) = p_ec(i-1,2) + v_ec(i,2)*dtau
55
56         p_verlet(i,1) = p_ec(i,1)
57         p_verlet(i,2) = p_ec(i,2)
58     else
59         p3 = pcubo(p_verlet(i-1,1),p_verlet(i-1,2))
60
61         p_verlet(i,1) = 2d0*p_verlet(i-1,1) - p_verlet(i-2,1)
62     &         - (p_verlet(i-1,1)/p3)*(dtau**2)
63         p_verlet(i,2) = 2d0*p_verlet(i-1,2) - p_verlet(i-2,2)
64     &         - (p_verlet(i-1,2)/p3)*(dtau**2)
65     endif
66
67     ! verificar se já deu 1 período
68     if (tempo.gt.1d0)then
69         if ((abs(p_verlet(1,1)-p_verlet(i,1)).lt.1d-3)
70     &         .and.(abs(p_verlet(1,2)-p_verlet(i,2)).lt.1d-3)) then
71             tempo_perodo = tempo
72             goto 10
73         endif
74     endif
75 end do
76
77 10 continue
78 if (tempo_perodo.eq.0d0)then
79     write(*,*)"Erro: nao achamos periodo p/",ip
80 else
81     write(*,*)"periodo p/",ip,tempo_perodo
82
83     write(1,*) (a**3d0), (tempo_perodo**2d0)
84 endif
85
86 end do
87
88 ! agora calcular as velocidades em Verlet
89 ! vamos usar a derivada simétrica de 5 pontos
90 !do i=3,iteracoes-2
91     ! começamos do 3 pq precisamos ter i-2 e i-1 válidos
92     ! v_verlet(i,1) = (p_verlet(i-2,1) - 8d0*p_verlet(i-1,1)
93     &     ! + 8d0*p_verlet(i+1,1) - p_verlet(i+2,1)) / (12d0*dtau)
94     ! v_verlet(i,2) = (p_verlet(i-2,2) - 8d0*p_verlet(i-1,2)

```

```

95      &      !      + 8d0*p_verlet(i+1,2) - p_verlet(i+2,2)) / (12d0*dtau)
96
97      !  write(4,*)v_verlet(i,1), v_verlet(i,2)
98      !end do
99
100
101      end
102
103      function pcubo(px,py)
104          real*8 px, py, pcubo
105          pcubo = (px**2d0 + py**2d0)**(3d0/2d0)
106      end function
107

```

Rodando o primeiro programa e graficando os resultados, obtemos:

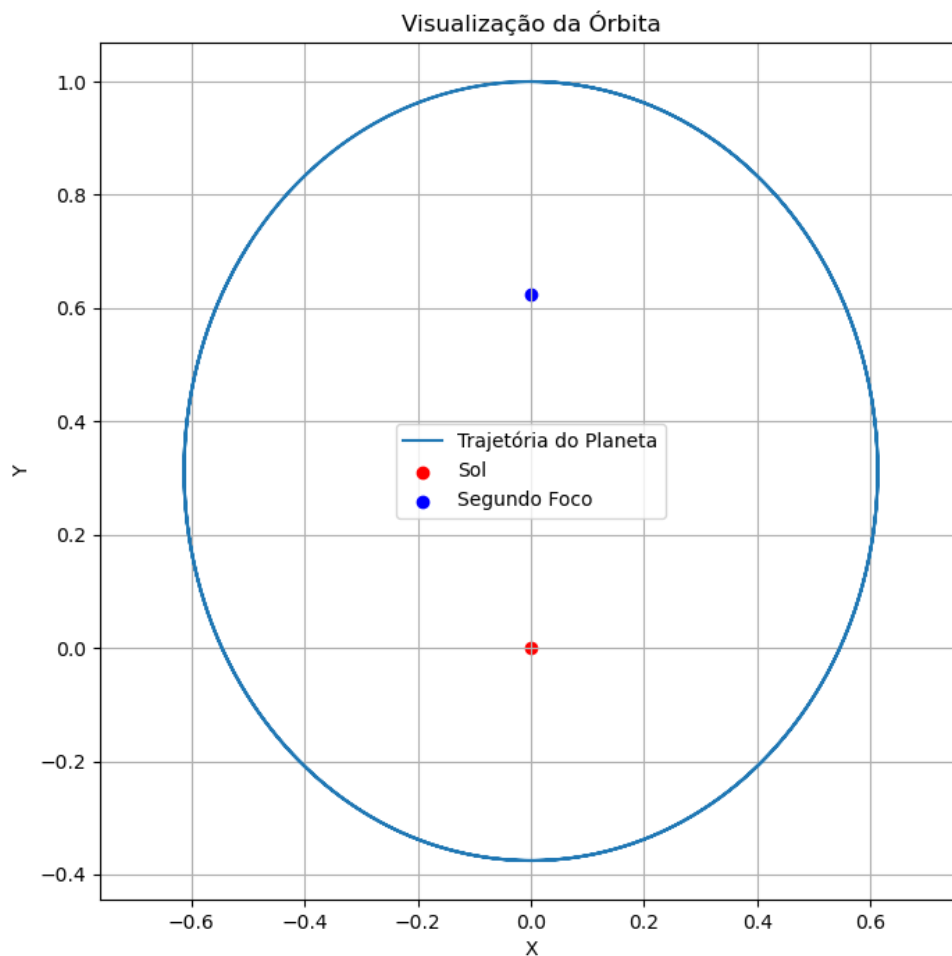


Figura 6: Gráfico da órbita de Marte simulada

Para descobrir a posição do 2o foco no nosso programa, basta apenas descobrir qual é a posição de menor valor de  $y$  - a distância desta posição à origem será igual à distância do ponto de maior  $y$  ao segundo foco. Neste caso, para a órbita de Marte, temos que a posição do sol é  $(0, 0)$  (claro) e a do segundo foco é 0.624.

Agora, podemos verificar a 1a lei de Kepler, utilizando esse foco hipotético e verificando se a distância ao sol e ao foco, somadas, são constantes ao longo do tempo.

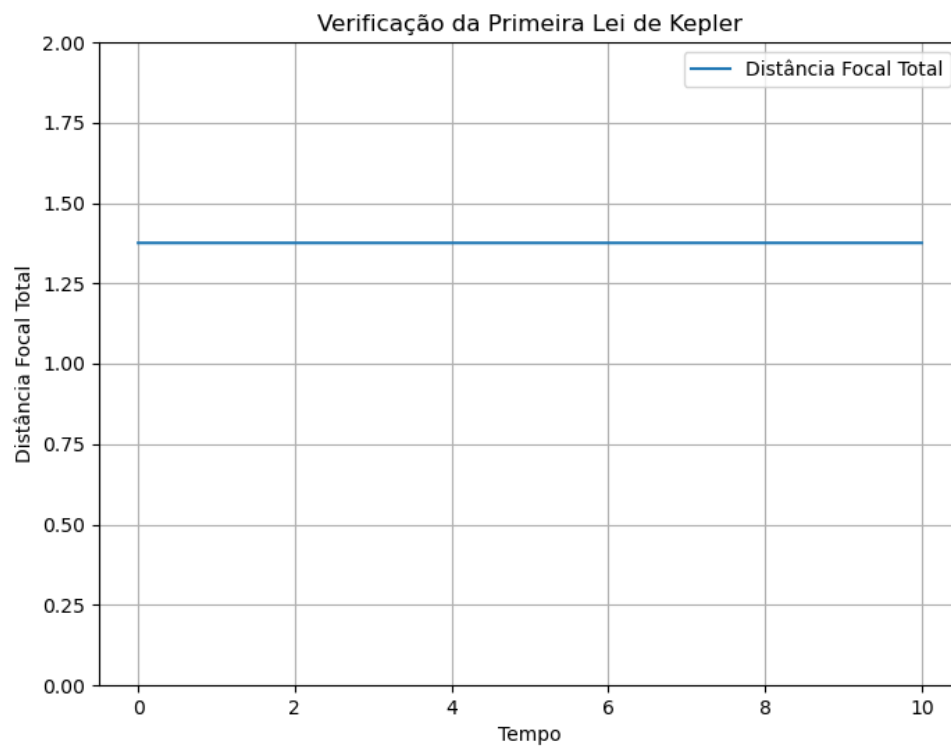


Figura 7: Gráfico da 1a lei de Kepler

De fato, a distância é constante ao longo do tempo! Portanto, a 1a lei está verificada. Para vermos a 2a lei empregamos um procedimento similar, mas dessa vez calculando a área varrida em cada iteração do programa. Graficando, temos

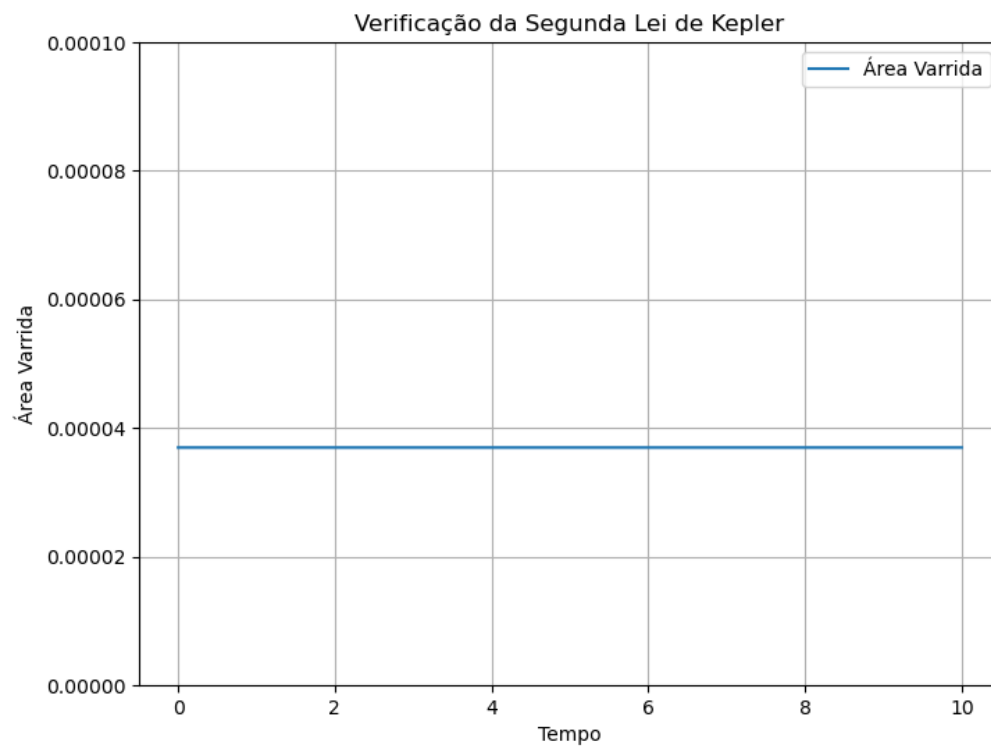


Figura 8: Gráfico da 2a lei de Kepler

Novamente, uma constante. Portanto, a 2a lei está verificada.

Para a 3a lei, simulamos diversos planetas e verificamos os seus períodos. Graficando isso, obtemos

### Verificação da 3a lei de Kepler

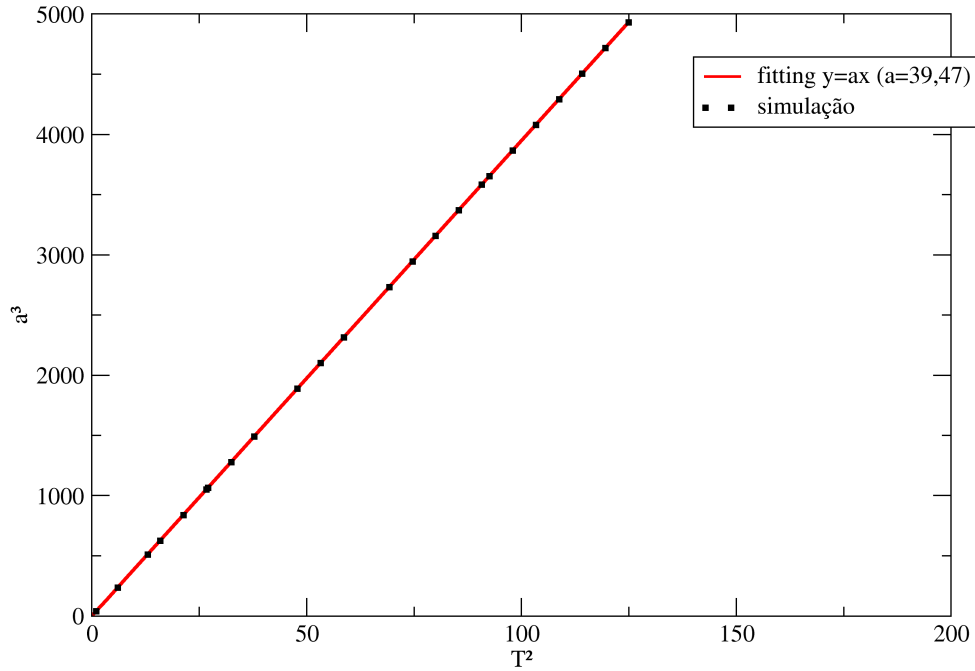


Figura 9: Gráfico da 3a lei de Kepler

Percebemos que a curva obtida é exatamente aquela que esperamos para  $a^3 \propto T^2$ , com a constante de proporcionalidade 39.47, nas unidades adotadas. Portanto, verificamos também a 3a lei.

## 3 Tarefa 3

### 3.1 a

As equações de movimento são

$$\vec{\rho}_t = \vec{a}_{ts} + \mu_j \vec{a}_{tj}$$

$$\vec{\rho}_j = \vec{a}_{js} + \mu_t \vec{a}_{jt}$$

Para aplicar isso no método de Verlet basta apenas substituir na fórmula que já te-



mos, isto é

$$\vec{\rho}_{i+1} = 2\vec{\rho}_i - \vec{\rho}_{i-1} + \vec{\rho}_i(\Delta\tau)^2$$

Nosso programa utiliza subrotinas para atualizar as acelerações da terra e de jupiter a cada iteração. Segue o programa:

```
1      implicit real*8 (a-h, o-z)
2      real*8 p_t(100000000, 2)
3      real*8 p_j(100000000, 2)
4      real*8 at(2), aj(2)
5
6      open(file='terra.dat',unit=1)
7      open(file='jupiter.dat', unit=2)
8
9      pi = 4.d0*datan2(1.d0,1.d0)
10
11     tau_total = 2d0*pi*30d0 ! 30 anos
12     dtau = 1d-3
13
14     p_t(1,1) = 1d0
15     p_t(1,2) = 0d0
16     p_j(1,1) = 5.2d0
17     p_j(1,2) = 0d0
18
19     v_tx = 0d0
20     v_ty = 1d0
21     v_jx = 0d0
22     v_jy = 1d0/dsqrt(5.2d0)
23
24     iteracoes = tau_total/dtau
25
26     iunidade = 10000
27
28     ! fazer iterações ser divisível por iunidade (p ficar bonitinha
29     ↪ a divisao no progresso)
30     if (mod(iteracoes,iunidade).ne.0) then
31         iteracoes = (iteracoes/iunidade + 1)*iunidade
32     endif
33
34     do i=2,iteracoes
35         if (mod(i,iunidade).eq.0) then
36             write(*,*) 'progresso:',int((dble(i)/dble(iteracoes))*100)
37         endif
```

```

38      ! equações de movimento p/ terra e júpiter
39      call acel_t(p_t(i-1,1),p_t(i-1,2),p_j(i-1,1),p_j(i-1,2), at)
40      call acel_j(p_j(i-1,1),p_j(i-1,2),p_t(i-1,1),p_t(i-1,2), aj)
41
42      if (i.eq.2) then
43          ! usar euler-cromer
44
45          ! terra
46          v_tx = v_tx - at(1)*dtau
47          v_ty = v_ty - at(2)*dtau
48          p_t(i,1) = p_t(i-1,1) + v_tx*dtau
49          p_t(i,2) = p_t(i-1,2) + v_ty*dtau
50
51          ! jupiter
52          v_jx = v_jx - aj(1)*dtau
53          v_jy = v_jy - aj(2)*dtau
54          p_j(i,1) = p_j(i-1,1) + v_jx*dtau
55          p_j(i,2) = p_j(i-1,2) + v_jy*dtau
56
57      else
58          ! terra
59          p_t(i,1) = 2d0*p_t(i-1,1) - p_t(i-2,1) +
60      &      at(1)*(dtau**2d0)
61          p_t(i,2) = 2d0*p_t(i-1,2) - p_t(i-2,2) +
62      &      at(2)*(dtau**2d0)
63
64          !jupiter
65          p_j(i,1) = 2d0*p_j(i-1,1) - p_j(i-2,1) +
66      &      aj(1)*(dtau**2d0)
67          p_j(i,2) = 2d0*p_j(i-1,2) - p_j(i-2,2) +
68      &      aj(2)*(dtau**2d0)
69      endif
70
71      write(1,*)p_t(i,1),p_t(i,2) !, at
72      write(2,*)p_j(i,1),p_j(i,2)
73
74  end do
75
76  end
77
78  subroutine acel_t(px,py,pjx,pjy,at)
79      ! modifica a aceleração da terra dados p_t e p_j
80      real*8 px,py,pjx,pjy,atsx,atsy,mu_j,mu_t,atjx,atjy,p3
81      real*8 at(2)

```

```

82
83      mu_t = 1d0/(3.33d5)
84      mu_j = 318d0/(3.33d5)
85
86      p3 = (px**2d0 + py**2d0)**1.5d0
87      p3j = ( (px-pjx)**2d0 + (py-pjy)**2d0 )**1.5d0
88
89      atsx = -px/p3
90      atsy = -py/p3
91
92      atjx = -(px-pjx)/p3j
93      atjy = -(py-pjy)/p3j
94
95      at(1) = atsx + mu_j*atjx
96      at(2) = atsy + mu_j*atjy
97
98  end subroutine
99
100  subroutine acel_j(px,py,ptx,pty,aj)
101      ! modifica a aceleração de jupiter dados p_t e p_j
102      real*8 px,py,ptx,pty,ajsx,ajsy,mu_j,mu_t,ajtx,ajty,p3
103      real*8 aj(2)
104
105      mu_t = 1d0/(3.33d5)
106      mu_j = 318d0/(3.33d5)
107
108      p3 = (px**2d0 + py**2d0)**1.5d0
109      p3t = ( (px-ptx)**2d0 + (py-pty)**2d0 )**1.5d0
110
111      ajsx = -px/p3
112      ajsy = -py/p3
113
114      ajtx = -(px-ptx)/p3t
115      ajty = -(py-pty)/p3t
116
117      aj(1) = ajsx + mu_j*ajtx
118      aj(2) = ajsy + mu_j*ajty
119
120  end subroutine

```

## 3.2 b

Aqui utilizaremos o menor  $\Delta\tau$  de que precisaríamos de acordo com os resultados da tarefa 1c, que, no caso, é de  $10^{-3}$ .

Graficando a órbita da Terra durante 30 anos, temos

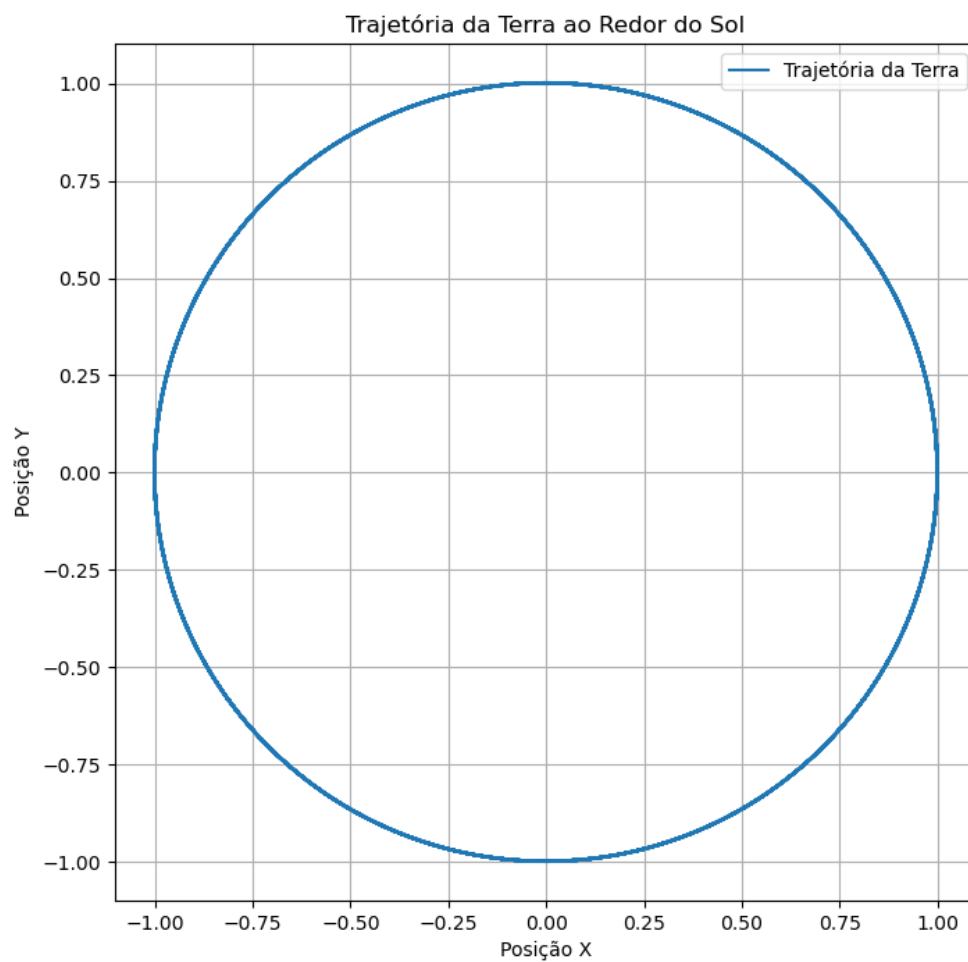


Figura 10: Órbita de terra com Júpiter no sistema

Percebemos que, se dermos um grande zoom, a órbita está variando ao longo do tempo:

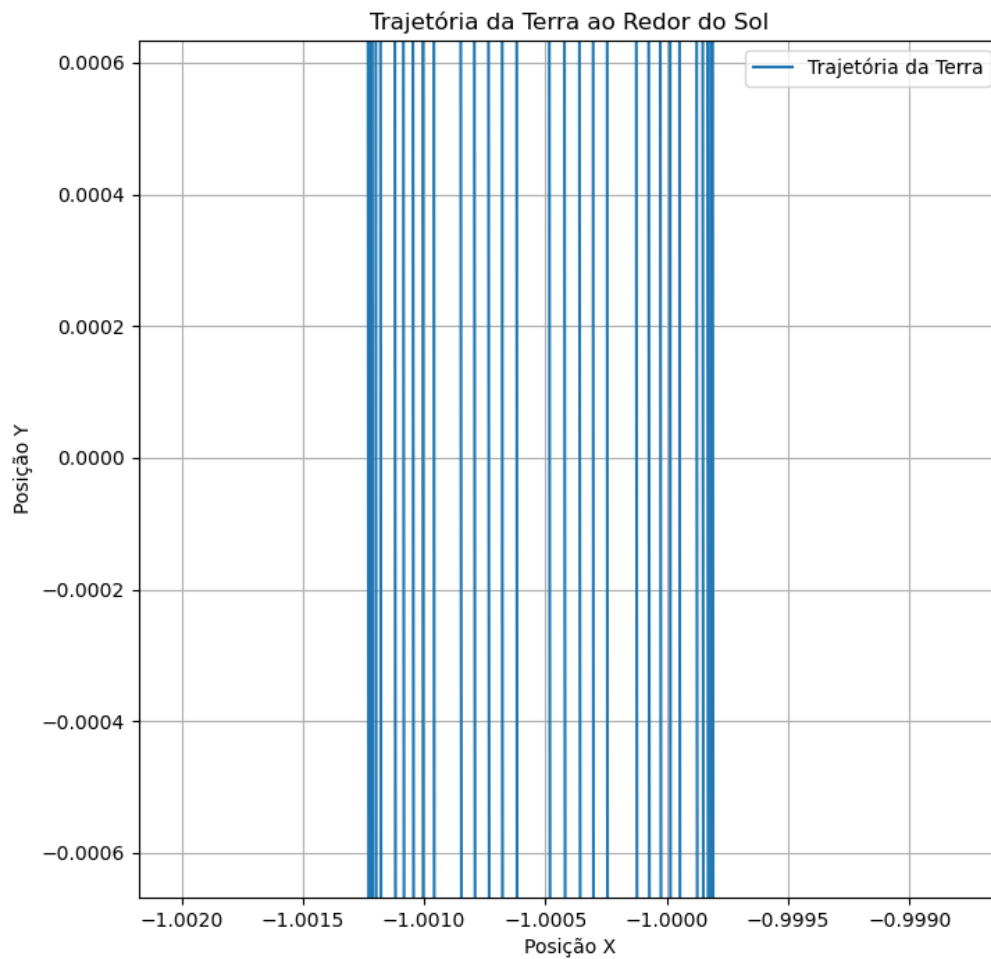


Figura 11: Órbita de terra com Júpiter no sistema - com zoom

Isto ocorre devido à atração gravitacional de Júpiter.

### 3.3 c

Multiplicando a massa de Júpiter por 100, achei legal fazer uma animação do resultado, que pode ser vista nesse link:

<https://youtu.be/CYUy6Q7UrLI>

Sem animação, temos os seguintes dois gráficos:

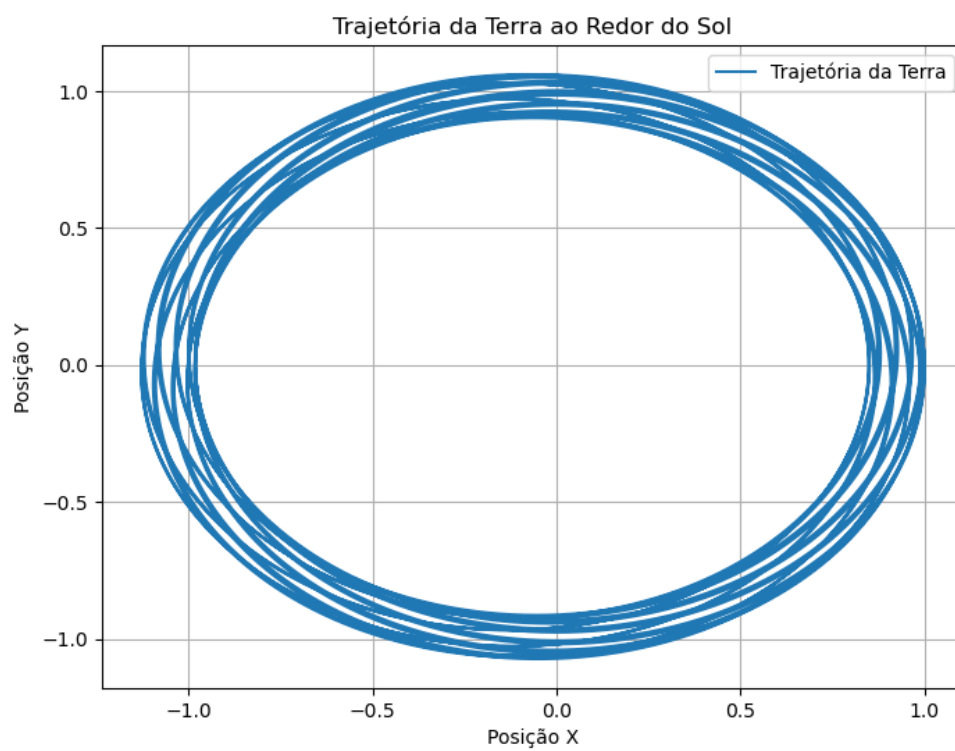


Figura 12: Órbita de terra com júpiter 100x mais massivo

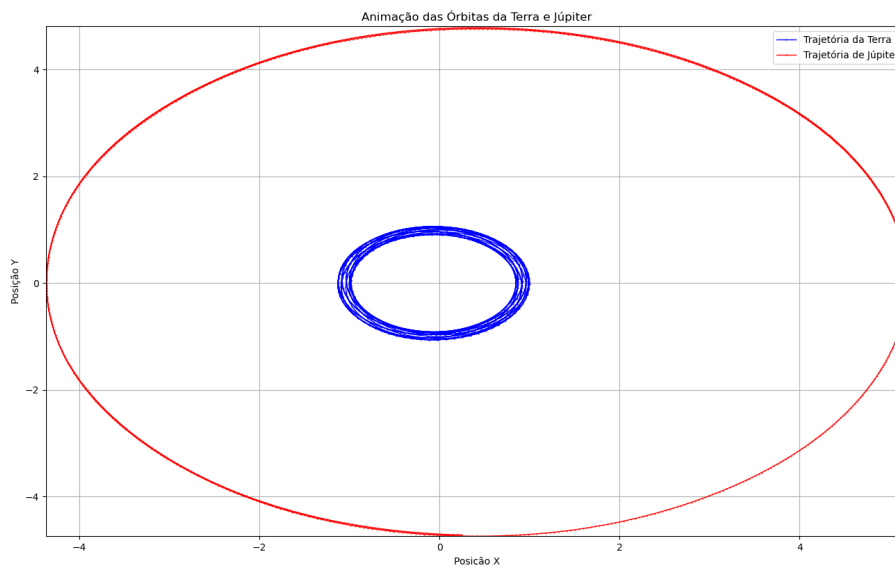


Figura 13: Órbita de terra e de júpiter, com júpiter 100x mais massivo

Percebemos que a órbita da terra vai oscilando dependendo de onde fica a posição de Júpiter, o que faz com que ela não tenha uma "órbita" específica estável. Porém, parece que ela não sai muito da área onde ela está, apenas fica oscilando em sua órbita. Isso pode ser visualizado melhor na animação.

Agora, multiplicamos a massa de Júpiter por 1000 ao invés de 100, e temos um resultado muito interessante:

<https://www.youtube.com/watch?v=CA9KG9WjM50>

O sistema é caótico! Agora não temos mais sistema solar, mas mesmo assim a animação é muito bonita e legal de assistir.