7600017

Introdução à Física Computacional

Projeto 4: O problema de Kepler
Força Central

Prof: José A. Hoyos

Victor Foscarini Almeida
nUsp: 10728101

São Carlos,2019

Introdução

O método de Runge-Kutta é um dos métodos mais eficientes para se realizar a integração numérica. Ele demora mais que outros métodos como o método de Simpsons e do trapézio para ser programado, porém compensa em eficiência.

Métodos e Resultados

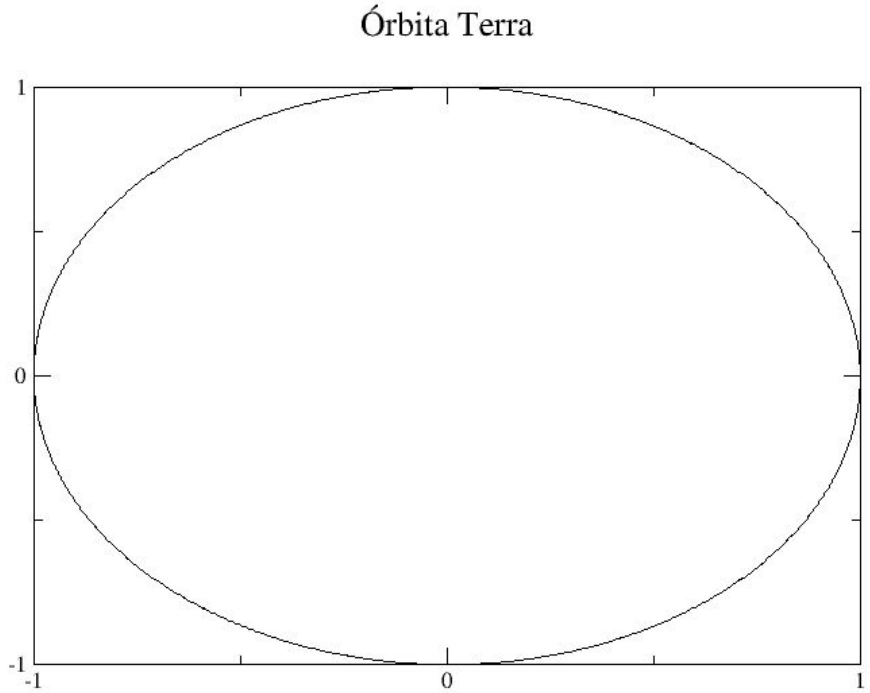
1- O problema de dois corpos

Dada uma força gravitacional entre dois corpos, é possível simplificá-lo ao tomar que um corpo é muito mais massivo que o outro, assim o problema vira um corpo na ação de uma força central. Esse simplificação é muito útil para casos como o da Terra e do Sol, onde a massa do sol é muito maior do que a da Terra.

1.1 - Força central e planeta Terra em órbita

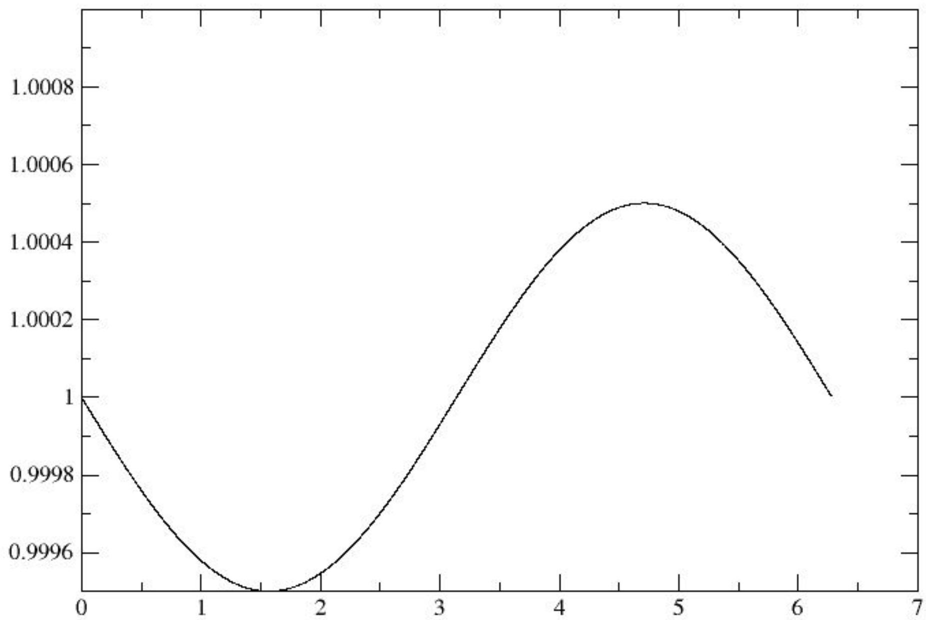
Criou-se um código para simular a órbita de um planeta em volta de uma estrela, considerando essa estrela tão mais massiva que o planeta que a força de atração entre ambas pode ser considerada uma força central, onde a estrela fica parada e o planeta orbita a estrela em um movimento circular.

Plotando as posições, a órbita fica assim:



Note que apesar de parecer uma elipse, ela é circular, isso ocorre apenas devido ao layout do Grace.

Já o raio em função do tempo fica, então:



```fortran
program orbitaTerra

    implicit none

    integer i
    real*8 pi
    real*8 vx,vy,p,px,py,T,dT

    p = 1.d0 !p = 1/UA * r, no caso 1 para a Terra

    pi = 4*atan(1.d0) !a definição de pi do fortran

    !define-se o intervalo de T(período), note que 2*pi/ano, para
a terra 2*pi

    dT =  0.001d0 !utiliza-se o intervalo de 1/1000 de ano


    !note que T e dT se referem a tau e p se refere a rô

    !vamos definir o planeta Terra saindo de px = p e py = 0, ou
seja, do ponto onde a posição no eixo y é nula
    px = p
    py = 0

    !dadas as condições iniciais, utiliza-se a velocidade da Terra

    vx = 0
    vy = 1.d0 !tem-se v = 2*pi*r/ano, ou v= r/UA em p/s, sendo v=1
para a Terra e vy máxima pela geometria

    !utiliza-se um loop implementando o método de EC

    open(10,file='orbitas.dat')
    open(20,file='RaioxTempo.dat')

    do i=1,int(1 * 2*pi/dT)
         vx = vx - px/p**3 * dT
         px = px + vx * dT

         vy = vy - py/p**3 * dT
         py = py + vy *dT

         p = (px**2 + py**2)**0.5
         T = T + dT
```

```
        write(10,*) px,py
        write(20,*) T,p

    enddo


end program
```

1.2 - Velocidade e dT máximo

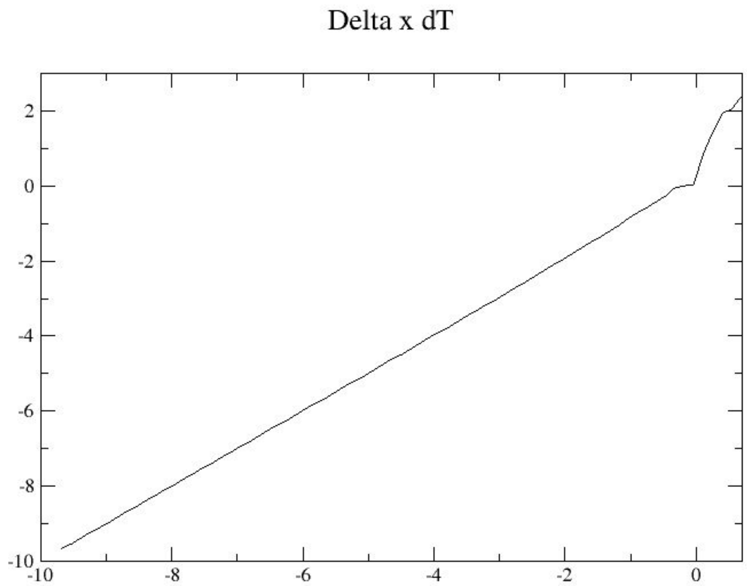
Primeiramente, cria-se um programa que, dado o raio e a massa de um planeta que orbita o sol, retorna a velocidade que esse planeta deveria ter para que a órbita seja circular. Utiliza-se o semieixo maior e aplica-se $v = \sqrt{G * M/R}$. Note que a entrada é a massa em massas da Terra e o semieixo maior em unidades astronômicas.

Planeta	Velocidade(m/s)
Mercúrio	11186.55
Vênus	10022.13
Terra	29788.41
Marte	7903.46
Júpiter	232948.10
Saturno	93903.76
Urano	25886.94
Netuno	22452.36

```
program velocidade

    implicit none

    real*8 M,R
    real*8 Ms,G,UA

    Ms = 1.989 !* 10**30
```

```
    G = 6.67408 !* 10**(-11)
    UA = 1.496 !* 10**11

      !será realizada a conta das potências de 10 por fora do fortran, visto que ele não consegue calcular com tamanha precisão


    write(*,*)"Massa do planeta: "
    read(*,*) M
    M = M * Ms

    write(*,*) "Semieixo maior: "
    read(*,*) R
    R = R*UA

     !conta das potências = -11 + 30 + (-11) =  8 , sendo a raiz de 10 a oitava 10 a quarta

    write(*,*)"velocidade:", sqrt(G*M/R)*10000,"m/s"

end program
```

Agora, para a segunda parte do problema, obteve-se o gráfico de delta em função de dT, em logxlog e o valor de dTmax como 10^{-3}, devido à linearidade do gráfico obtido no formato y=x. Note que para valores muito grandes(canto superior direito) não há precisão suficiente, então a função não se comporta como esperado, mas para valores menores nota-se claramente a relação.

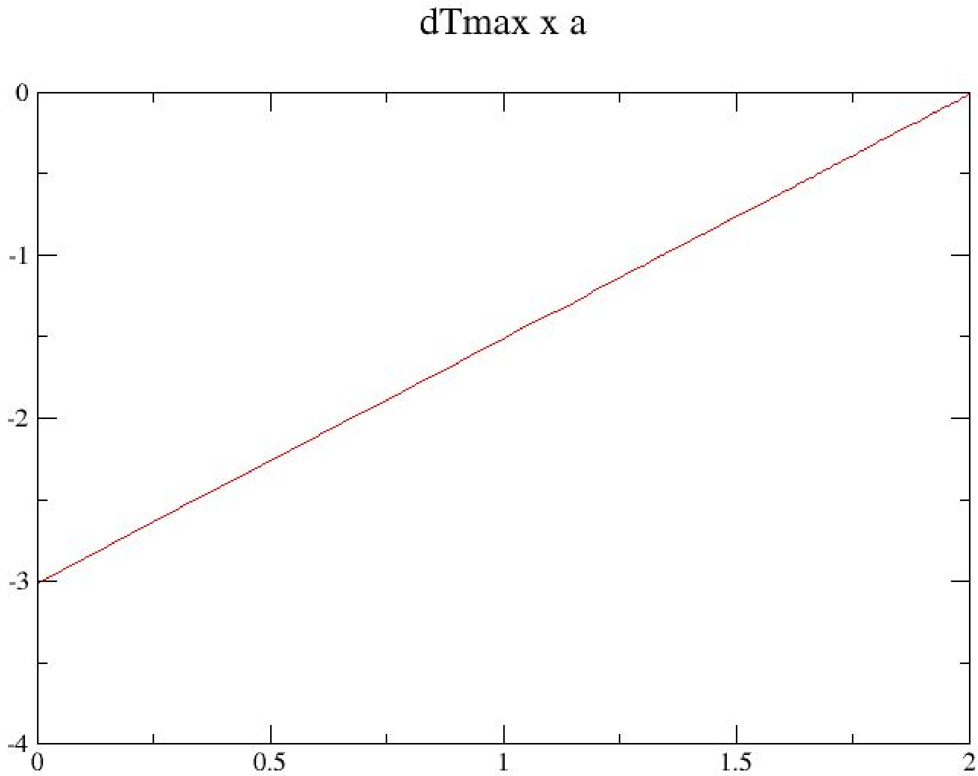


```
program deltaxdT

    implicit none

    integer i,j
    real*8 pi
    real*8 vx,vy,p,px,py,T,dT,pmax,pmin,delta,jr

    pi = 4*atan(1.d0) !a definição de pi do fortran

    open(10,file="deltaxdT.dat")

    do j=8,80

            !diferentemente do codigo anterior,T e dT aqui serao
variáveis

        !note que T e dT se referem a tau e p a rô

        p = 1.d0 !p = 1/UA * r, no caso 1 para a Terra

         !vamos definir o planeta Terra saindo de px = p e py = 0,
ou seja, do ponto onde a posição no eixo y é nula
        px = p
        py = 0

          !dadas as condições iniciais, utiliza-se a velocidade da
Terra
        vx = 0
         vy = 1.d0 !tem-se v = 2*pi*r/ano, ou v= r/UA em p/s, sendo
v=1 para a Terra e vy máxima pela geometria

        !utiliza-se um loop implementando o método de EC

         !define-se o pmax e o pmin como sendo 1, visto que esse é
o valor de p inicial e, a partir de um if, encontra-se o pmax e o
pmin
        pmax = 1.d0
        pmin = 1.d0

             !define-se o intervalo de T(período), note que T =
2*pi/ano, para a terra T = 2*pi

        jr = real(j,8)/16

        T = 2*pi
```

```
        dT = T / 10**(jr)

        do i=1,10**(jr)

           vx = vx - px/p**3 * dT
           px = px + vx * dT

           vy = vy - py/p**3 * dT
           py = py + vy *dT

           p = sqrt(px**2 + py**2)

           if (p > pmax) then
               pmax = p
           else if (p < pmin) then
               pmin = p
           endif

        enddo

        delta = pmax/pmin - 1

        write(10,*)dT,delta

    enddo


end program
```

Após isso, então, busca-se o valor de dT max em função de a, na esperança de revelar algo acerca da terceira lei de Kepler.

Plota-se então dT max em função de a, a função y = -3.0143 + 1.5 * x, onde o coeficiente angular 1,5 é resultado direto da análise dimensional da lei de Kepler, da proporcionalidade do tempo com o raio elevado a 3/2. Isso ocorre pois dTmax é proporcional ao período e a variação sofrida durante o ciclo é proporcional à razão entre o passo temporal e o período.




```
program dTmaxxa

    implicit none

    integer i,j,k,Ninteracoes
    real*8 pi
    real*8 vx,vy,p,px,py,T,dT,pmax,pmin,delta,jr

    pi = 4*atan(1.d0) !a definição de pi do fortran

    open(10,file="dTxa.dat")


    do k=1,100

        delta = 1.d0 !valor inicial para iniciar o loop
         j = 8 !o j é utilizado para calcular o valor de interações
para dT atingir T, ou seja, uma órbita
```

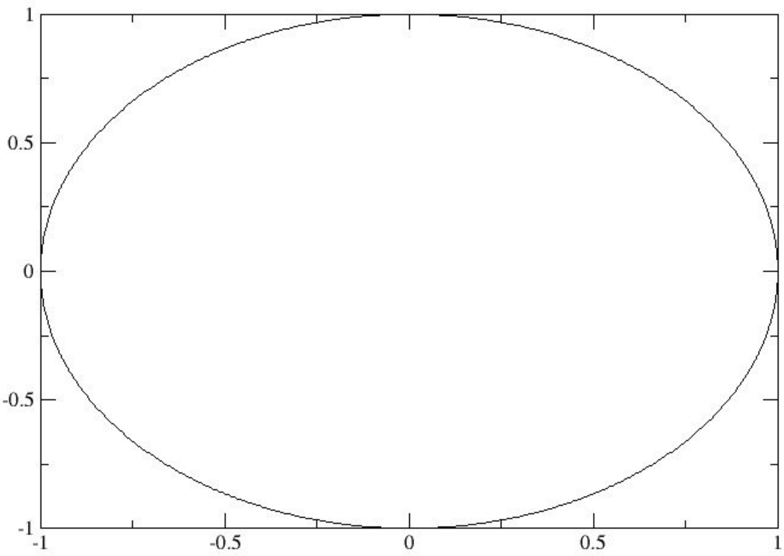
```
        do while(delta > 0.001d0)

            j = j + 1

              !diferentemente do codigo anterior,T e dT aqui serao
variáveis

            !note que T e dT se referem a tau e p a rô

            p = 1.d0 * k !p = 1/UA * r, no caso 1 para a Terra


             !vamos definir o planeta Terra saindo de px = p e py =
0, ou seja, do ponto onde a posição no eixo y é nula
            px = p
            py = 0

              !dadas as condições iniciais, utiliza-se a velocidade
da Terra
            vx = 0
             vy = 1.d0 / sqrt(p) !tem-se v = 2*pi*r/ano, ou v= r/UA
em p/s, sendo v=1 para a Terra e vy máxima pela geometria

            !utiliza-se um loop implementando o método de EC

               !define-se o pmax e o pmin como sendo 1, visto que
esse é o valor de p inicial e, a partir de um if, encontra-se o
pmax e o pmin
            pmax = p
            pmin = p

               !define-se o intervalo de T(período), note que T =
2*pi/ano, para a terra T = 2*pi

            jr = real(j,8)/16

              T = 2*pi * p**(1.5) !note que utilizou-se a terceira
lei de Kepler para fazer uma proporção para o novo período em
relação ao período da Terra
            dT = T / 10**(jr)

            Ninteracoes = ceiling(10**jr,4)

            do i=1,Ninteracoes

                vx = vx - px/p**3 * dT
```

```
            px = px + vx * dT

            vy = vy - py/p**3 * dT
            py = py + vy *dT

            p = sqrt(px**2 + py**2)

            if (p > pmax) then
                pmax = p
            else if (p < pmin) then
                pmin = p
            endif

        enddo

        delta = pmax/pmin - 1


    enddo

    write(10,*)log10(p),log10(dT)

  enddo

end program
```

1.3 - Planeta em órbita com força central: método Runge-Kutta

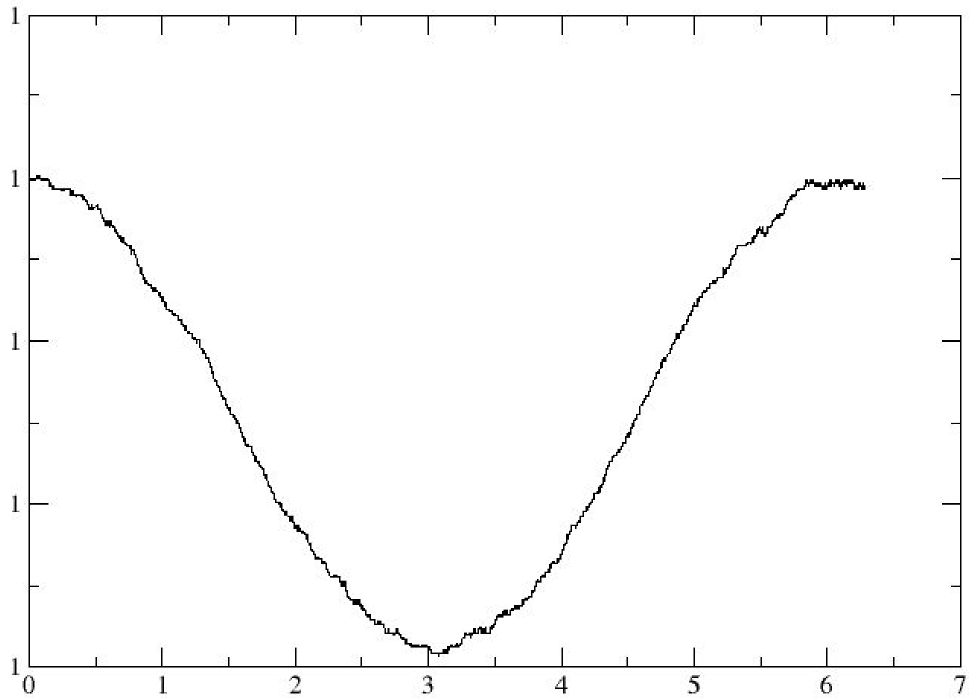
Aqui será feito o mesmo processo do item 2.1, mas dessa vez utilizando o método de Runge-Kutta, já discutido anteriormente na introdução do projeto.

Órbita Terra



Por fim, colocando o raio em função do tempo, pelo método de Runge-Kutta,obtém-se:

Raio x Tempo : Runge-Kutta



```
program orbitarungekutta

    implicit none

    integer i,j,k
    real*8 pi,ano
    real*8 vx,vy,p,px,py,r,T,dT
    real*8 vx2,vy2,px2,py2
    real*8 F1(5),F2(5),F3(5),F4(5)

    p = 1.d0 !p = 1/UA * r, no caso 1 para a Terra

    pi = 4*atan(1.d0) !a definição de pi do fortran

    !define-se o intervalo de T(período), note que 2*pi/ano, para
a terra é 2*pi

    T = 0.d0
    dT =  0.001

    !note que T e dT se referem a tau e p a rô

    !vamos definir o planeta Terra saindo de px = p e py = 0, ou
seja, do ponto onde a posição no eixo y é nula
    px = p
    py = 0

    !dadas as condições iniciais, utiliza-se a velocidade da Terra

    vx = 0
    vy = 1.d0 !tem-se v = 2*pi*r/ano, ou v= r/UA em p/s, sendo v=1
para a Terra e vy máxima pela geometria

    open(10,file='orbitas.dat')
    open(20,file="raioxtempo.dat")

    !utiliza-se um loop implementando o método de Runge-Kutta

    T = 0.d0

    do j=1,int(1 * 2*pi/dT)

        F1(1) = vx
        F2(1) = vy
        F3(1) = - px/p**3
        F4(1) = - py/p**3
```

```
        do i=2,4

            if (i<4) then
                px2 = px + dT/2 * F1(i-1)
                py2 = py + dT/2 * F2(i-1)
                vx2 = vx + dT/2 * F3(i-1)
                vy2 = vy + dT/2 * F4(i-1)
            else
                px2 = px + dT * F1(i-1)
                py2 = py + dT * F2(i-1)
                vx2 = vx + dT * F3(i-1)
                vy2 = vy + dT * F4(i-1)
            endif

            call Rk(i,F1,F2,F3,F4,px2,py2,vx2,vy2)

        enddo

      px = px + dT/6 * ( F1(1) + 2*F1(2) + 2*F1(3) + F1(4) )
        py = py + dT/6 * ( F2(1) + 2*F2(2) + 2*F2(3) + F2(4) )
        vx = vx + dT/6 * ( F3(1) + 2*F3(2) + 2*F3(3) + F3(4) )
        vy = vy + dT/6 * ( F4(1) + 2*F4(2) + 2*F4(3) + F4(4) )

          p = (px**2.d0 + py**2.d0)**0.5d0
          T = T +dT


         write(10,*) px,py
         write(20,*) p,T

    enddo


end program

subroutine Rk(i,F1,F2,F3,F4,px2,py2,vx2,vy2)

    integer i
    real*8 F1(5),F2(5),F3(5),F4(5)
    real*8 px2,py2,vx2,vy2

    F1(i) = vx2
    F2(i) = vy2
    F3(i) = -px2/((px2**2 + py2**2)**1.5)
    F4(i) = -py2/((px2**2 + py2**2)**1.5)
```

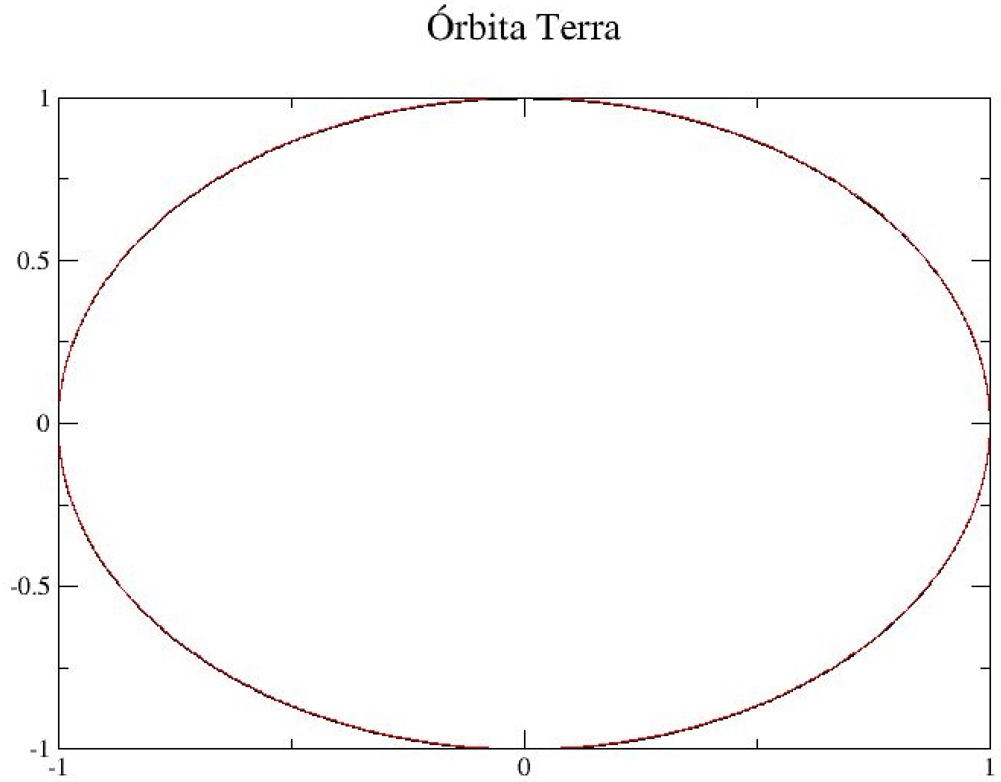
```
end subroutine
```

1.4 - Traçando o gráfico de p(T)

Traçando os dois métodos num mesmo gráfico, obtém-se:

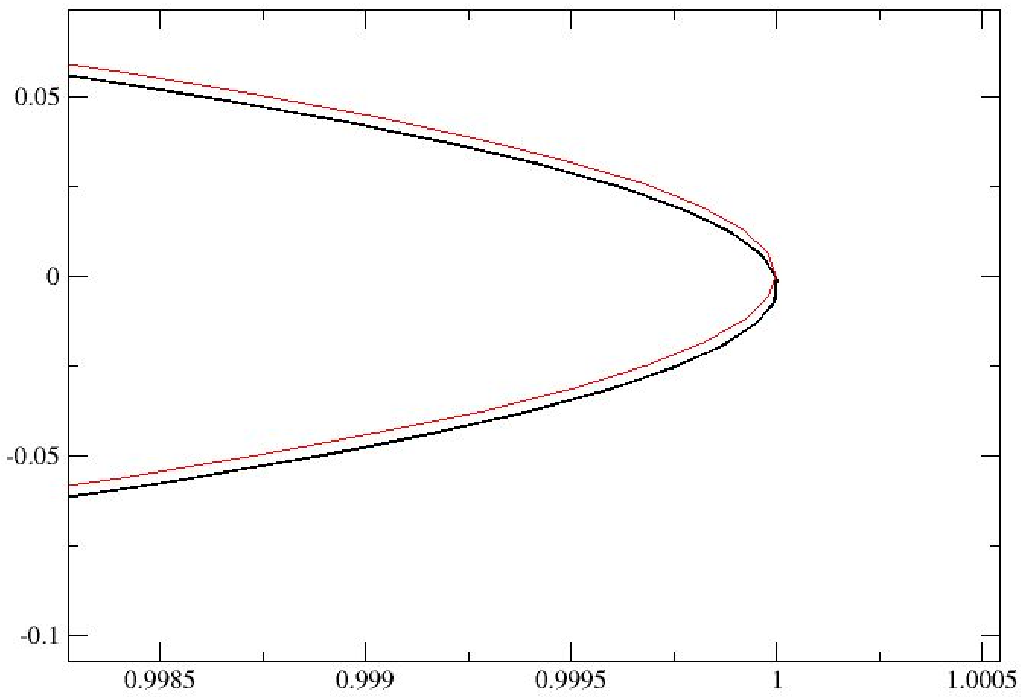
Vermelho : Runge-Kutta

Preto : método de EC



Os gráficos parecem ser iguais, porém ao dar zoom é possível notar a diferença:

Órbita Terra



Checando os valores de X e Y num ponto em que Y é máximo (1) e X é nulo, nota-se que o método de runge-kutta chega mais próximo a 1 do que o método de EC, sendo esperado p=1 no máximo, então Y=1 e X=0.

Por Runge-Kutta:

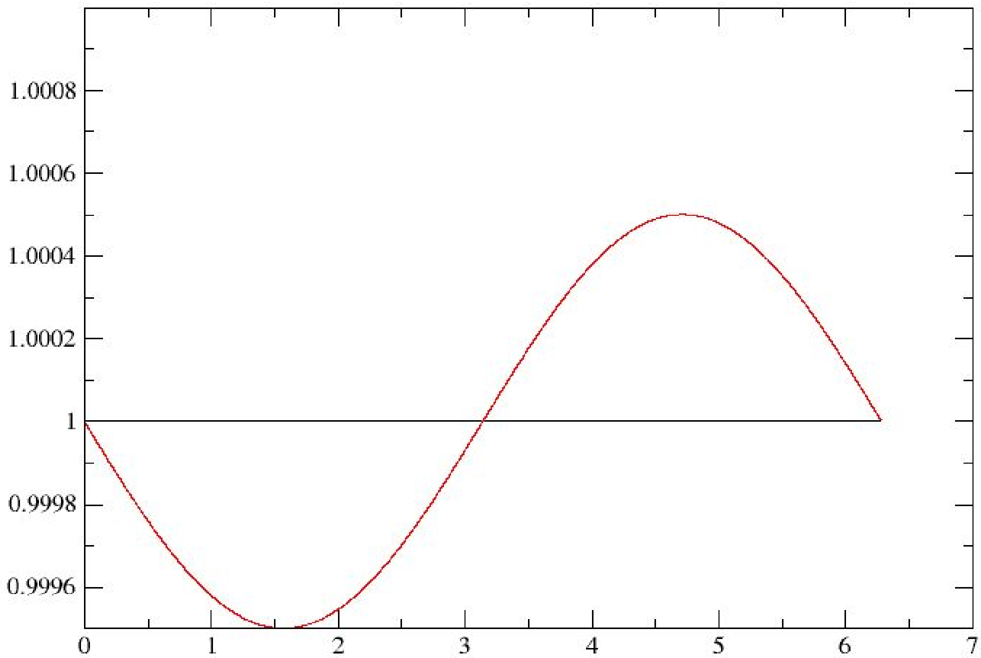
```
  2.5130093952902520E-002  0.99968418928679981
  1.8848438224682403E-002  0.99982235237483674
  1.2566038392394813E-002  0.99992104418836880
  6.2831424744279598E-003  0.99998026083121205
 -1.4912453372845080E-009  0.99999999996559474
 -6.2831454568597562E-003  0.99998026081224978
 -1.2566041374649979E-002  0.99992104415044503
 -1.8848441206643196E-002  0.99982235231795291
 -2.5130096934451205E-002  0.99968418921095803
 -3.1410760569061766E-002  0.99950656028390328
```

Por EC:

2.6815390762469193E-002	0.99651824677984091
2.0514268145960329E-002	0.99666747617824236
1.4212328027161093E-002	0.99677698795480529
7.9098215405982789E-003	0.99684677771310515
1.6069998438959397E-003	0.99687684265457133
-4.6958858925163034E-003	0.99686718157860466
-1.0998584495695523E-002	0.99681779488262667
-1.7300844800477846E-002	0.99672868456206165
-2.3602415659770438E-002	0.99659985421025055

O mesmo se repete em todas as pontas do círculo em que um eixa é nulo e o outro máximo, então é notório que o método de Runge-Kutta é mais preciso que o método de EC.

Tem-se, por fim, os gráficos de raio em função do tempo plotados na mesma escala para Runge-Kutta e pelo método de EC para notar a diferença de forma mais clara:



2 - Sistema com três corpos: Terra,Júpiter,Sol

Aqui será feita uma simulação mais próxima do sistema solar, considerando os efeitos de Júpiter sobre a Terra e sobre o Sol e vice-versa.

2.1 - Análogo de Runge-Kutta

Aplica-se, para cada objeto:

$$p_{x,i+1} = p_{x,i} + \tfrac{\Delta T}{6}(F^{(1)}_{1,i} + 2F^{(2)}_{1,i} + 2F^{(2)}_{1,i} + F^{(4)}_{1,i})$$
$$p_{y,i+1} = p_{y,i} + \tfrac{\Delta T}{6}(F^{(1)}_{2,i} + 2F^{(2)}_{2,i} + 2F^{(2)}_{2,i} + F^{(4)}_{2,i})$$
$$v_{x,i} = v_{x,i} + \tfrac{\Delta T}{6}(F^{(1)}_{3,i} + 2F^{(2)}_{3,i} + 2F^{(2)}_{3,i} + F^{(4)}_{3,i})$$
$$v_{y,i} = v_{y,i} + \tfrac{\Delta T}{6}(F^{(1)}_{4,i} + 2F^{(2)}_{4,i} + 2F^{(2)}_{4,i} + F^{(4)}_{4,i})$$

Onde:

$$F^{(j)}_{1,i} = v_{x,i}$$
$$F^{(j)}_{2,i} = v_{y,i}$$
$$F^{(j)}_{3,i} = \ddot{p}^{(j)}_{x,i}$$
$$F^{(j)}_{4,i} = \ddot{p}^{(j)}_{y,i}$$

E, então:

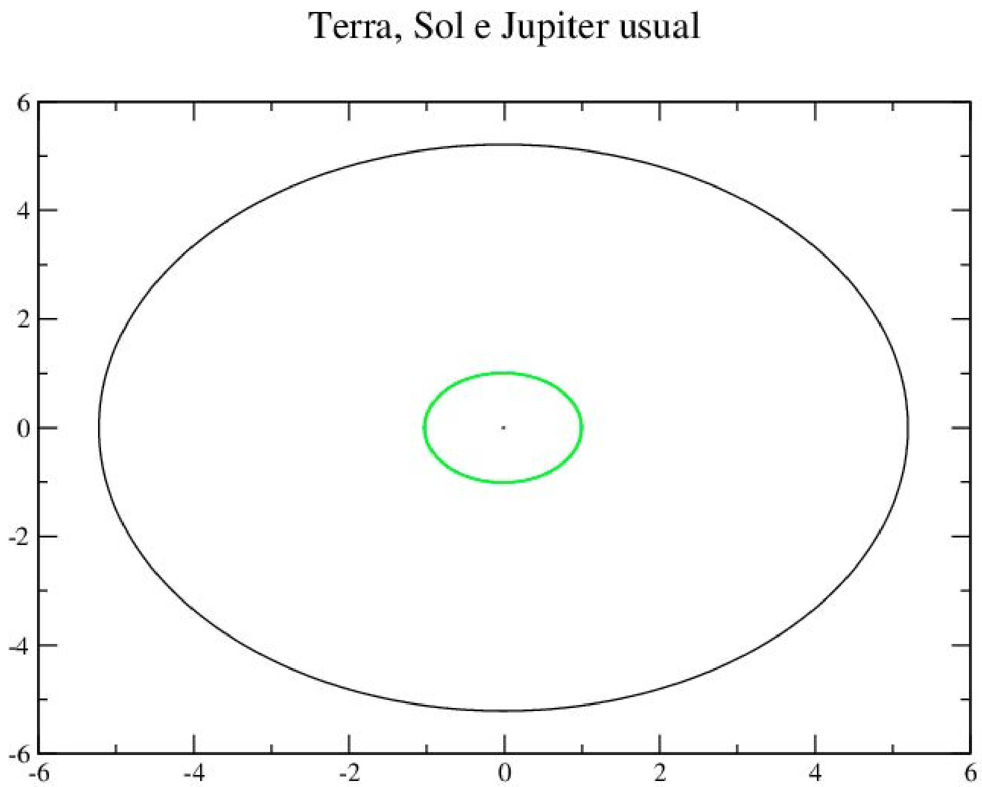
$$p_{x,i} = p_{x,i} + B^{(j)}\Delta T F^{(j-1)}_{1,i}$$
$$p_{y,i} = p_{y,i} + B^{(j)}\Delta T F^{(j-1)}_{2,i}$$
$$v_{x,i} = v_{x,i} + B^{(j)}\Delta T F^{(j-1)}_{3,i}$$
$$v_{y,i} = p_{y,i} + B^{(j)}\Delta T F^{(j-1)}_{4,i}$$

E:

$$p_i = \sqrt{p^2_{x,i} + p^2_{y,i}}$$
$$B^{(1)} = 0,2 \;\; ; \;\; 2B^{(2)} = 2B^{(3)} = B^{(4)} = 1$$

2.2- Sol,Terra e Júpiter

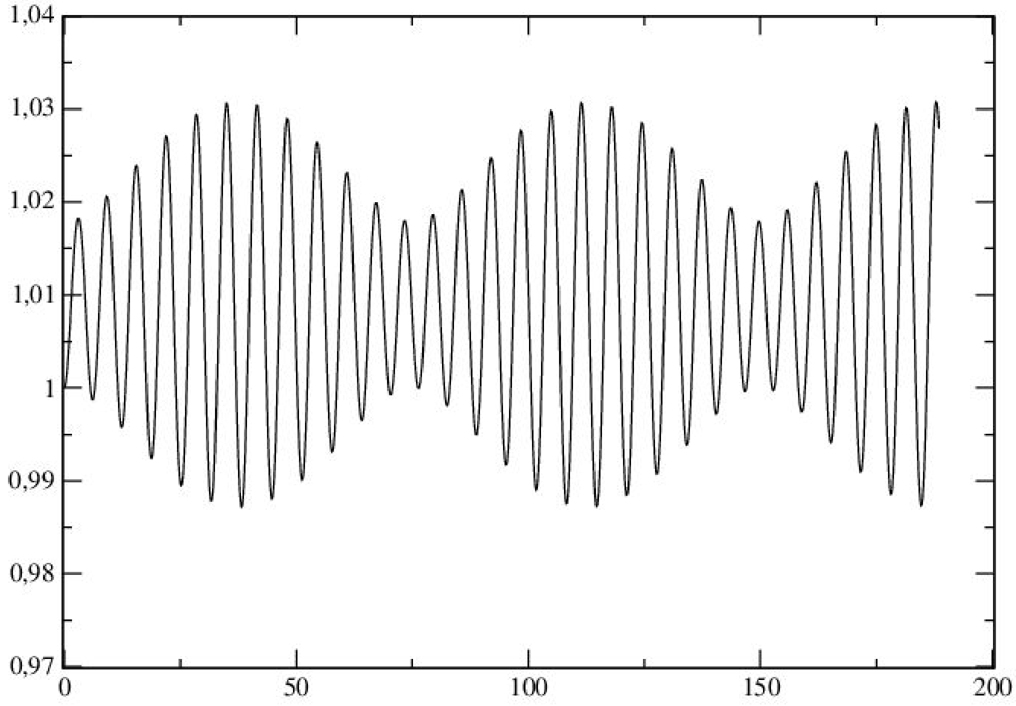
Ao deixar de ser um problema de uma força central e um planeta em órbita(Terra e Sol) e adicionar-se Júpiter no problema, a simulação torna-se mais complexa. sendo que dessa vez é necessária a simulação, visto que não há uma solução analítica simples para o problema de três corpos assim como há para o problema de dois corpos.



É notório que o sistema está em equilíbrio, o que era esperado visto que foram utilizados os dados reais do sistema solar. Obtivemos então algo muito semelhante ao sistema solar real, considerando esses três corpos.

Nota-se que o raio da terra(em relação ao centro do sistema, que no caso coincide como Sol) apresentou uma variação durante a órbita:

Raio da Terra x Tempo



Por fim, a razão obtida de pmax/pmin para a Terra a partir dos dados da simulação foi de **1,04357**. Esse valor é um pouco maior do que o obtido diretamente com os dados da Terra: **1,03399**, mostrando então que o método de Runge-Kutta consegue simular bem a órbita, mesmo que seja elíptica.

```
program TerraSolJupiter
      implicit none

      real*8 :: dT, ms, pi
      real*8 :: pxs, pys, xs, ys, vxs, vys, vx_s, vy_s
      real*8 :: p1, px1, py1, x1, y1, vx1, vy1, vx_1, vy_1
      real*8 :: px2, py2, x2, y2, vx2, vy2, vx_2, vy_2
      real*8 :: m1, m2, T
      real*8,dimension(4,4) :: f1(4,4), f2(4,4), fs(4,4)
      real*8, dimension(4) :: B(4)
      integer :: i, j

      dT = 0.001d0
      ms = 1.d0 !soma-se a massa do sol como base
      pi = 4.d0*atan(1.d0)
```

```
      m1=1.d0/(3.33d5) !massa da Terra
      m2=318.d0*m1 !massa de Júpiter

      !note que a Terra e Júpiter sairão das respectivas pontas
direitas da órbita(onde o eixo X é máximo e Y nulo) e o sol do
ponto central

      px1=1.d0
      py1=0.d0
      vx1=0.d0
      vy1=1.d0
      p1=(px1**2.d0+py1**2.d0)**0.5d0

      px2=5.2d0
      py2=0.d0
      vx2=0.d0
      vy2=1.d0/((5.2d0)**(0.5d0))

      pxs=-m1*px1-m2*px2
      pys=0.d0
      vxs=0.d0
      vys=-m1*vy1+m2*vy2

      T=0.d0
      B=[0.2d0, 2.d0, 2.d0, 1.d0] !diferentemente dos códigos
anteriores, foi definido um vetor que guarda os valores de B para
facilitar

      !arquivos que armazenam as trajetórias dos três corpos
      open(10,file='traj_terra.dat')
      open(20,file='traj_jupiter.dat')
      open(30,file='traj_sol.dat')

      open(40,file='p_terra.dat') !armazena o raio da órbita
terrestre


      do i=1,int(30*2*pi/dT)

            write(10,*)px1, py1
            write(20,*)px2, py2
            write(30,*)pxs, pys
            write(40,*)T, p1

            T=T+dT
```

```
            !aplica-se então o método de Runge-Kutta como descrito
no projeto

            f1(1,1) = vx1
            f1(1,2) = vy1
            f1(1,3) =
((pxs-px1)/(((pxs-px1)**2.d0+(pys-py1)**2.d0)**1.5d0)) -
(m2*(px1-px2)/(((px1-px2)**2.d0+(py1-py2)**2.d0)**1.5d0))
            f1(1,4) =
((pys-py1)/(((pxs-px1)**2.d0+(pys-py1)**2.d0)**1.5d0)) -
(m2*(py1-py2)/(((px1-px2)**2.d0+(py1-py2)**2.d0)**1.5d0))

            f2(1,1) = vx2
            f2(1,2) = vy2
            f2(1,3) =
((pxs-px2)/(((pxs-px2)**2.d0+(pys-py2)**2.d0)**1.5d0)) +
(m1*(px1-px2)/(((px1-px2)**2.d0+(py1-py2)**2.d0)**1.5d0))
            f2(1,4) =
((pys-py2)/(((pxs-px2)**2.d0+(pys-py2)**2.d0)**1.5d0)) +
(m1*(py1-py2)/(((px1-px2)**2.d0+(py1-py2)**2.d0)**1.5d0))

            fs(1,1) = vxs
            fs(1,2) = vys
            fs(1,3) =
(m1*(pxs-px1)/(((pxs-px1)**2.d0+(pys-py1)**2.d0)**1.5d0)) +
(m2*(pxs-px2)/(((pxs-px2)**2.d0+(pys-py2)**2.d0)**1.5d0))
            fs(1,4) =
(m1*(pys-py1)/(((pxs-px1)**2.d0+(pys-py1)**2.d0)**1.5d0)) +
(m2*(pys-py2)/(((pxs-px2)**2.d0+(pys-py2)**2.d0)**1.5d0))


                  do j=2,4

                  x1 = px1 + dT*f1(j-1,1)/B(j)
                  y1 = py1 + dT*f1(j-1,2)/B(j)
                  vx_1 = vx1 + dT*f1(j-1,3)/B(j)
                  vy_1 = vy1 + dT*f1(j-1,4)/B(j)

                  x2 = px2 + dT*f2(j-1,1)/B(j)
                  y2 = py2 + dT*f2(j-1,2)/B(j)
                  vx_2 = vx2 + dT*f2(j-1,3)/B(j)
                  vy_2 = vy2 + dT*f2(j-1,4)/B(j)

                  xs = pxs + dT*fs(j-1,1)/B(j)
                  ys = pys + dT*fs(j-1,2)/B(j)
```

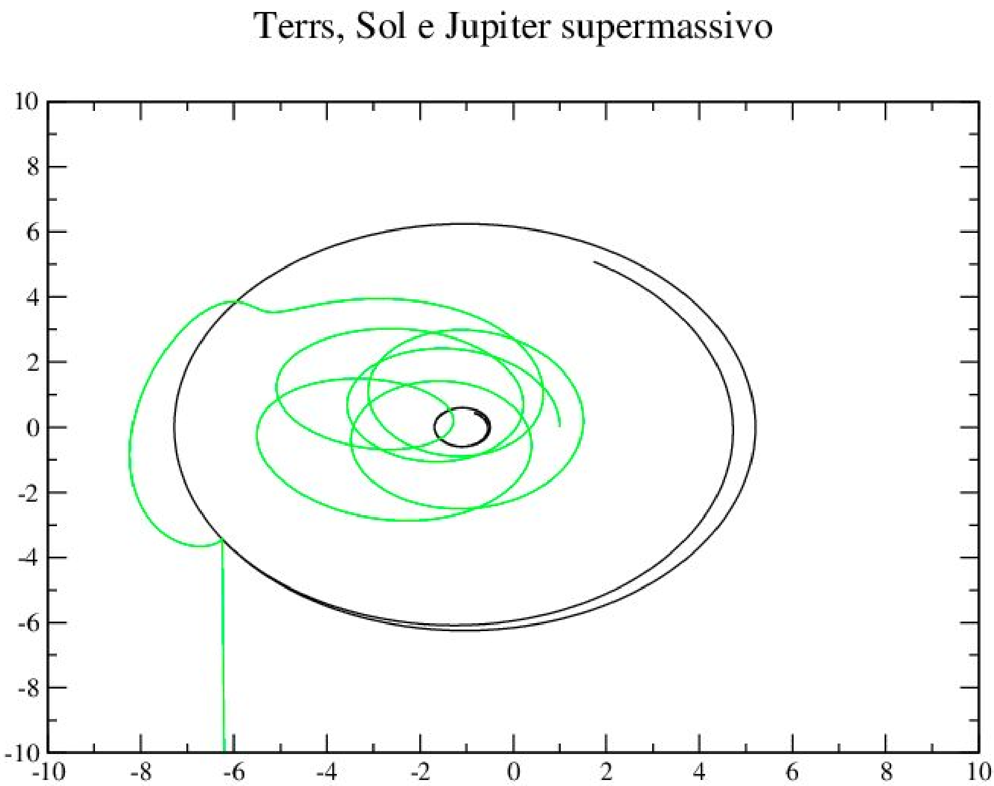
```
                  vx_s = vxs + dT*fs(j-1,3)/B(j)
                  vy_s = vys + dT*fs(j-1,4)/B(j)

                  f1(j,1) = vx_1
                  f1(j,2) = vy_1
                  f1(j,3) =
((xs-x1)/(((xs-x1)**2.d0+(ys-y1)**2.d0)**1.5d0)) -
(m2*(x1-x2)/(((x1-x2)**2.d0+(y1-y2)**2.d0)**1.5d0))
                  f1(j,4) =
((ys-y1)/(((xs-x1)**2.d0+(ys-y1)**2.d0)**1.5d0)) -
(m2*(y1-y2)/(((x1-x2)**2.d0+(y1-y2)**2.d0)**1.5d0))

                  f2(j,1) = vx_2
                  f2(j,2) = vy_2
                  f2(j,3) =
((xs-x2)/(((xs-x2)**2.d0+(ys-y2)**2.d0)**1.5d0)) +
(m1*(x1-x2)/(((x1-x2)**2.d0+(y1-y2)**2.d0)**1.5d0))
                  f2(j,4) =
((ys-y2)/(((xs-x2)**2.d0+(ys-y2)**2.d0)**1.5d0)) +
(m1*(y1-y2)/(((x1-x2)**2.d0+(y1-y2)**2.d0)**1.5d0))

                  fs(j,1) = vx_s
                  fs(j,2) = vy_s
                  fs(j,3) =
(m1*(xs-x1)/(((xs-x1)**2.d0+(ys-y1)**2.d0)**1.5d0)) +
(m2*(xs-x2)/(((xs-x2)**2.d0+(ys-y2)**2.d0)**1.5d0))
                  fs(j,4) =
(m1*(ys-y1)/(((xs-x1)**2.d0+(ys-y1)**2.d0)**1.5d0)) +
(m2*(ys-y2)/(((xs-x2)**2.d0+(ys-y2)**2.d0)**1.5d0))

                  enddo

            px1 = px1 +
(dT/6.d0)*(f1(1,1)+2.d0*(f1(2,1)+f1(3,1))+f1(4,1))
            py1 = py1 +
(dT/6.d0)*(f1(1,2)+2.d0*(f1(2,2)+f1(3,2))+f1(4,2))
            vx1 = vx1 +
(dT/6.d0)*(f1(1,3)+2.d0*(f1(2,3)+f1(3,3))+f1(4,3))
            vy1 = vy1 +
(dT/6.d0)*(f1(1,4)+2.d0*(f1(2,4)+f1(3,4))+f1(4,4))

            px2 = px2 +
(dT/6.d0)*(f2(1,1)+2.d0*(f2(2,1)+f2(3,1))+f2(4,1))
            py2 = py2 +
(dT/6.d0)*(f2(1,2)+2.d0*(f2(2,2)+f2(3,2))+f2(4,2))
```

```
          vx2 = vx2 +
(dT/6.d0)*(f2(1,3)+2.d0*(f2(2,3)+f2(3,3))+f2(4,3))
          vy2 = vy2 +
(dT/6.d0)*(f2(1,4)+2.d0*(f2(2,4)+f2(3,4))+f2(4,4))

          pxs = pxs +
(dT/6.d0)*(fs(1,1)+2.d0*(fs(2,1)+fs(3,1))+fs(4,1))
          pys = pys +
(dT/6.d0)*(fs(1,2)+2.d0*(fs(2,2)+fs(3,2))+fs(4,2))
          vxs = vxs +
(dT/6.d0)*(fs(1,3)+2.d0*(fs(2,3)+fs(3,3))+fs(4,3))
          vys = vys +
(dT/6.d0)*(fs(1,4)+2.d0*(fs(2,4)+fs(3,4))+fs(4,4))

          p1 = (px1**2.d0+py1**2.d0)**0.5d0

     enddo


end program
```

2.3- Sol,Terra e Júpiter supermassivo

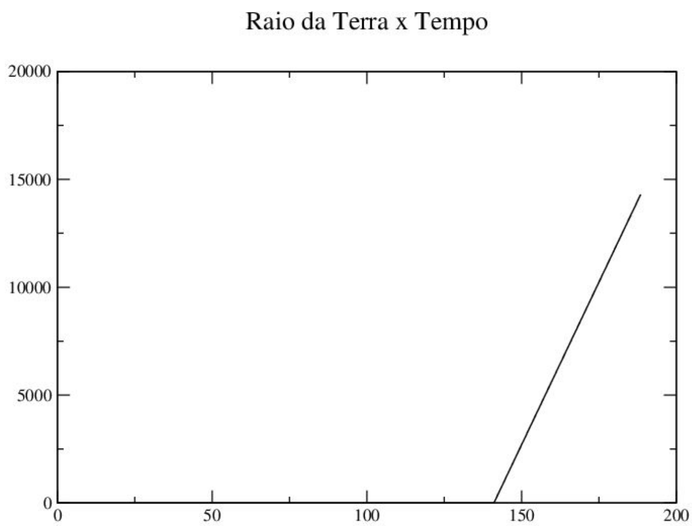
No problema anterior, notou-se que o Sol não se move muito e age praticamente como uma força central. Dessa vez, porém, será testado se, ao aumentar a massa de júpiter em 10 vezes, haverá alguma mudança na órbita. Espera-se checar uma movimentação maior do Sol dessa vez e algum efeito na Terra.

Obtém-se, então, o gráfico abaixo, onde é notório um efeito de Júpiter sobre a Terra, que apresenta um movimento confuso e o sol, que parece orbitar o ponto central do gráfico. E, a ocorrência mais é importante é o fato de que a Terra é ejetada do sistema solar após alguns anos

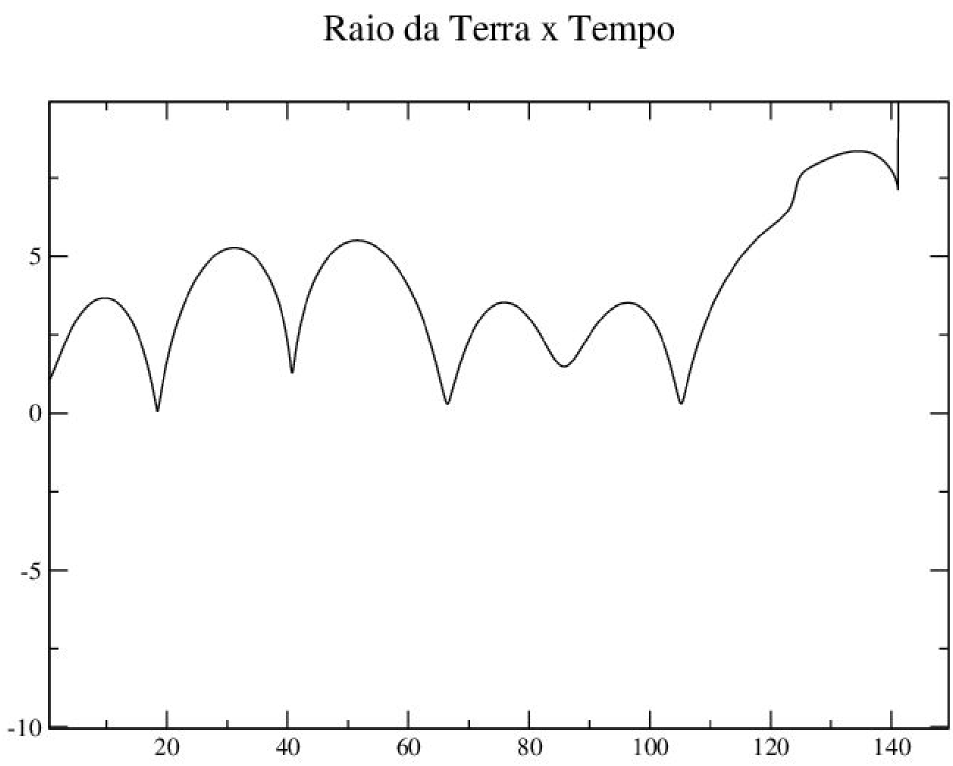


Já quanto ao raio da Terra (em relação ao centro do sistema), nota-se que ele não cresce como no item anterior e, apesar de manter o movimento meio senóide em períodos pequenos de tempo, para períodos grandes a variação é confusa e então a Terra é ejetada do sistema solar, o que parece mostrar que o sistema não está estável como anteriormente.

O raio da Terra em função do tempo fica, como esperado:



Ao dar zoom, obtemos um resultado mais interessante:



Note que no código as mudanças estão marcadas em vermelho e assim será feito nos próximos:

```
program TerraSolJupitersupermassivo
      implicit none

      real*8 :: dT, ms, pi
      real*8 :: pxs, pys, xs, ys, vxs, vys, vx_s, vy_s
      real*8 :: p1, px1, py1, x1, y1, vx1, vy1, vx_1, vy_1
      real*8 :: px2, py2, x2, y2, vx2, vy2, vx_2, vy_2
      real*8 :: m1, m2, T
      real*8,dimension(4,4) :: f1(4,4), f2(4,4), fs(4,4)
      real*8, dimension(4) :: B(4)
      integer :: i, j

      dT = 0.001d0
      ms = 1.d0 !soma-se a massa do sol como base
      pi = 4.d0*atan(1.d0)

      m1=1.d0/(3.33d5) !massa da Terra
      m2=318.d0*m1 * 100 !massa de Júpiter multiplicada por 100
```

```
      !condições iniciais dos três corpos
      !note que a Terra e Júpiter sairão das respectivas pontas
direitas da órbita(onde o eixo X é máximo e Y nulo)

      px1=1.d0
      py1=0.d0
      vx1=0.d0
      vy1=1.d0
      p1=(px1**2.d0+py1**2.d0)**0.5d0

      px2=5.2d0
      py2=0.d0
      vx2=0.d0
      vy2=1.d0/((5.2d0)**(0.5d0))

      pxs=-m1*px1-m2*px2
      pys=0.d0
      vxs=0.d0
      vys=-m1*vy1+m2*vy2

      T=0.d0
      B=[0.2d0, 2.d0, 2.d0, 1.d0] !diferentemente dos códigos
anteriores, foi definido um vetor que guarda os valores de B para
facilitar

      !arquivos que armazenam as trajetórias dos três corpos
      open(10,file='traj_terra.dat')
      open(20,file='traj_jupiter.dat')
      open(30,file='traj_sol.dat')

      open(40,file='p_terra.dat')  !armazena o raio da órbita
terrestre


      do i=1,int(30*2*pi/dT)

            write(10,*)px1, py1
            write(20,*)px2, py2
            write(30,*)pxs, pys
            write(40,*)T, p1

            T=T+dT

            !aplica-se então o método de Runge-Kutta como descri1o
no projeto
```

```
          f1(1,1) = vx1
          f1(1,2) = vy1
          f1(1,3)                                                    =
((pxs-px1)/(((pxs-px1)**2.d0+(pys-py1)**2.d0)**1.5d0))              -
(m2*(px1-px2)/(((px1-px2)**2.d0+(py1-py2)**2.d0)**1.5d0))
          f1(1,4)                                                    =
((pys-py1)/(((pxs-px1)**2.d0+(pys-py1)**2.d0)**1.5d0))              -
(m2*(py1-py2)/(((px1-px2)**2.d0+(py1-py2)**2.d0)**1.5d0))

          f2(1,1) = vx2
          f2(1,2) = vy2
          f2(1,3)                                                    =
((pxs-px2)/(((pxs-px2)**2.d0+(pys-py2)**2.d0)**1.5d0))              +
(m1*(px1-px2)/(((px1-px2)**2.d0+(py1-py2)**2.d0)**1.5d0))
          f2(1,4)                                                    =
((pys-py2)/(((pxs-px2)**2.d0+(pys-py2)**2.d0)**1.5d0))              +
(m1*(py1-py2)/(((px1-px2)**2.d0+(py1-py2)**2.d0)**1.5d0))

          fs(1,1) = vxs
          fs(1,2) = vys
          fs(1,3)                                                    =
(m1*(pxs-px1)/(((pxs-px1)**2.d0+(pys-py1)**2.d0)**1.5d0))           +
(m2*(pxs-px2)/(((pxs-px2)**2.d0+(pys-py2)**2.d0)**1.5d0))
          fs(1,4)                                                    =
(m1*(pys-py1)/(((pxs-px1)**2.d0+(pys-py1)**2.d0)**1.5d0))           +
(m2*(pys-py2)/(((pxs-px2)**2.d0+(pys-py2)**2.d0)**1.5d0))


               do j=2,4

               x1 = px1 + dT*f1(j-1,1)/B(j)
               y1 = py1 + dT*f1(j-1,2)/B(j)
               vx_1 = vx1 + dT*f1(j-1,3)/B(j)
               vy_1 = vy1 + dT*f1(j-1,4)/B(j)

               x2 = px2 + dT*f2(j-1,1)/B(j)
               y2 = py2 + dT*f2(j-1,2)/B(j)
               vx_2 = vx2 + dT*f2(j-1,3)/B(j)
               vy_2 = vy2 + dT*f2(j-1,4)/B(j)

               xs = pxs + dT*fs(j-1,1)/B(j)
               ys = pys + dT*fs(j-1,2)/B(j)
               vx_s = vxs + dT*fs(j-1,3)/B(j)
               vy_s = vys + dT*fs(j-1,4)/B(j)

               f1(j,1) = vx_1
```

```
               f1(j,2) = vy_1
               f1(j,3)                                                =
((xs-x1)/(((xs-x1)**2.d0+(ys-y1)**2.d0)**1.5d0))                      -
(m2*(x1-x2)/(((x1-x2)**2.d0+(y1-y2)**2.d0)**1.5d0))
               f1(j,4)                                                =
((ys-y1)/(((xs-x1)**2.d0+(ys-y1)**2.d0)**1.5d0))                      -
(m2*(y1-y2)/(((x1-x2)**2.d0+(y1-y2)**2.d0)**1.5d0))

               f2(j,1) = vx_2
               f2(j,2) = vy_2
               f2(j,3)                                                =
((xs-x2)/(((xs-x2)**2.d0+(ys-y2)**2.d0)**1.5d0))                      +
(m1*(x1-x2)/(((x1-x2)**2.d0+(y1-y2)**2.d0)**1.5d0))
               f2(j,4)                                                =
((ys-y2)/(((xs-x2)**2.d0+(ys-y2)**2.d0)**1.5d0))                      +
(m1*(y1-y2)/(((x1-x2)**2.d0+(y1-y2)**2.d0)**1.5d0))

               fs(j,1) = vx_s
               fs(j,2) = vy_s
               fs(j,3)                                                =
(m1*(xs-x1)/(((xs-x1)**2.d0+(ys-y1)**2.d0)**1.5d0))                   +
(m2*(xs-x2)/(((xs-x2)**2.d0+(ys-y2)**2.d0)**1.5d0))
               fs(j,4)                                                =
(m1*(ys-y1)/(((xs-x1)**2.d0+(ys-y1)**2.d0)**1.5d0))                   +
(m2*(ys-y2)/(((xs-x2)**2.d0+(ys-y2)**2.d0)**1.5d0))

               enddo

          px1                 =                 px1                   +
(dT/6.d0)*(f1(1,1)+2.d0*(f1(2,1)+f1(3,1))+f1(4,1))
          py1                 =                 py1                   +
(dT/6.d0)*(f1(1,2)+2.d0*(f1(2,2)+f1(3,2))+f1(4,2))
          vx1                 =                 vx1                   +
(dT/6.d0)*(f1(1,3)+2.d0*(f1(2,3)+f1(3,3))+f1(4,3))
          vy1                 =                 vy1                   +
(dT/6.d0)*(f1(1,4)+2.d0*(f1(2,4)+f1(3,4))+f1(4,4))

          px2                 =                 px2                   +
(dT/6.d0)*(f2(1,1)+2.d0*(f2(2,1)+f2(3,1))+f2(4,1))
          py2                 =                 py2                   +
(dT/6.d0)*(f2(1,2)+2.d0*(f2(2,2)+f2(3,2))+f2(4,2))
          vx2                 =                 vx2                   +
(dT/6.d0)*(f2(1,3)+2.d0*(f2(2,3)+f2(3,3))+f2(4,3))
          vy2                 =                 vy2                   +
(dT/6.d0)*(f2(1,4)+2.d0*(f2(2,4)+f2(3,4))+f2(4,4))
```

```
        pxs             =             pxs          +
(dT/6.d0)*(fs(1,1)+2.d0*(fs(2,1)+fs(3,1))+fs(4,1))
        pys             =             pys          +
(dT/6.d0)*(fs(1,2)+2.d0*(fs(2,2)+fs(3,2))+fs(4,2))
        vxs             =             vxs          +
(dT/6.d0)*(fs(1,3)+2.d0*(fs(2,3)+fs(3,3))+fs(4,3))
        vys             =             vys          +
(dT/6.d0)*(fs(1,4)+2.d0*(fs(2,4)+fs(3,4))+fs(4,4))

        p1 = (px1**2.d0+py1**2.d0)**0.5d0

    enddo


end program
```

3-Coreografias celestes

Serão estudadas aqui formas de órbitas celestes diferenciadas em que é possível simular as órbitas dos planetas.

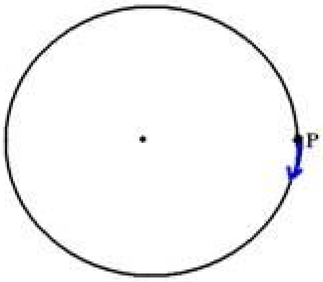
3.1 - Três corpos em um triângulo equilátero

Modifica-se então, os valores iniciais no código assim como pedido:

```
    !note aqui que 1 e 2 se referem às respectivas partículas 1 e
2, além do que 3 se refere ao Sol
    !as três partículas tem a mesma massa
    !são feitas aqui as substituições como pedido no projeto

    dT = 0.001d0
    ms = 1.d0
    pi = 4.d0*atan(1.d0)

    m1=1.d0
    m2=1.d0


    px1=1.d0
    py1=0.d0
    vx1=0.d0
    vy1=1.d0*V0

    px2=-0.5d0
```

```
py2=(3.d0**0.5d0)/2.d0
vx2=(-3.d0**0.5d0)/2.d0*V0
vy2=-0.5d0*V0

pxs=-0.5d0
pys=(-3.d0**0.5d0)/2.d0
vxs=(3.d0**0.5d0)/2.d0*V0
vys=-0.5d0*V0
```

Espera-se encontrar então uma trajetória circular:



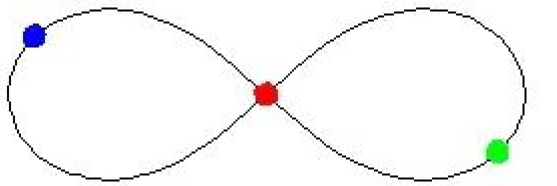
3.2 - Solução de C. Moore: o oito

Modifica-se então, os valores iniciais no código assim como pedido:

```
      !note aqui que 1 e 2 se referem às respectivas partículas 1 e
2, além do que 3 se refere ao Sol
      !as três partículas tem a mesma massa
      !são feitas aqui as substituições como pedido no projeto

      dT = 0.001d0
      ms = 1.d0
      pi = 4.d0*atan(1.d0)

      m1=1.d0
      m2=1.d0


      px1=0.97000436d0
      py1=-0.24308753d0
      vx1=0.466203685d0
      vy1=0.43236573d0

      px2=-0.97000436d0
      py2=0.24308753d0
      vx2=0.466203685d0
```

```
vy2=0.43236573d0

pxs=0.d0
pys=0.d0
vxs=-0.93240737d0
vys=-0.86473146d0
```

Espera-se encontrar uma órbita em forma de oito:



3.3 - Solução de C. Moore modificada

Modificando como pedido a posição inicial em X da partícula 1.

```
!note aqui que 1 e 2 se referem às respectivas partículas 1 e 2, além do que 3 se refere ao Sol
!as três partículas tem a mesma massa
!são feitas aqui as substituições como pedido no projeto

dT = 0.001d0
ms = 1.d0
pi = 4.d0*atan(1.d0)

m1=1.d0
m2=1.d0


px1=0.95000436
py1=-0.24308753d0
```

```
vx1=0.466203685d0
vy1=0.43236573d0

px2=-0.97000436d0
py2=0.24308753d0
vx2=0.466203685d0
vy2=0.43236573d0

pxs=0.d0
pys=0.d0
vxs=-0.93240737d0
vys=-0.86473146d0
```

Espera-se encontrar uma órbita em forma de 8, porém com fase defasada, ou seja, a aparência seria de vários 8 deslocados um pouco para o lado um do outro.