7600017

Introdução à Física Computacional

Projeto 3: Movimento Realístico

Prof: José A. Hoyos

Victor Foscarini Almeida
nUsp: 10728101

São Carlos,2019

Introdução

O método de Euler é uma ferramenta extremamente útil em computação científica para a resolução de equações diferenciais, que são equações que relacionam alguma função com a sua derivada. Com este, calcula-se numericamente, para pequenos valores de tempo (dt) consecutivos a variação de um valor da derivada de maior ordem da função, relacionando-o então com a derivada de menor ordem e assim por diante, até chegar no valor da própria função. Dessa forma, é possível resolver equações diferenciais que, de outra forma, não poderiam facilmente ser resolvidas, dependendo apenas das condições iniciais.

Apesar disso, para casos oscilatórios em que conserva-se a energia, o método de Euler vai propagando os erros e acaba não sendo capaz de conservar a energia. Então, torna-se necessário o uso do método de Euler-Cromer,

Uma das principais aplicações de tais métodos é para resolução de problemas físicos numericamente, fazendo uma análise mais complexa e completa de uma situação que, de outra forma, seria simplificada para tornar os cálculos possíveis de serem realizados por um ser humano. Será utilizado o Fortran 95 para programar as situações e realizar as contas devido a sua eficiência para realizar cálculos numéricos.

Métodos e Resultados

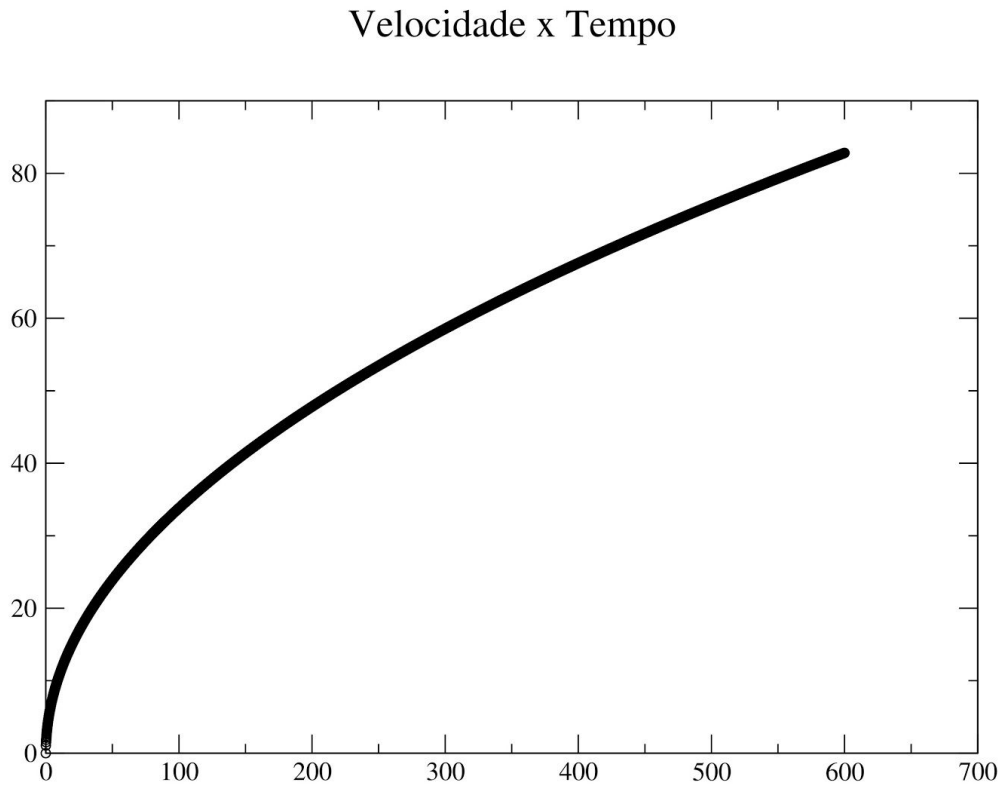
1 - Efeito resistivo do ar em movimento unidimensionais

1.1 - Sem efeito resistivo

O movimento de um ciclista pode ser simplificado para algo unidimensional, aqui será considerado o caso de um atleta de alto desempenho andando em um caminho reto. Sabe-se experimentalmente que um atleta de alto desempenho produz uma potência de 400 W constante. É razoável considerar que ele seja capaz de manter esse movimento por 5 minutos também. Sendo assim, a partir do método

de Euler, relaciona-se a energia cinética com a sua derivada primeira, a potência a partir do método de Euler por intervalos de tempo pequenos(dt).

Considerando primeiramente dt em intervalos de 0.1s, obtém-se o seguinte gráfico:



Ao fazer com intervalos de 0.01s ao invés de com intervalos de 0.1s não obtêm-se nenhuma diferença, visto que foi utilizada a energia cinética K e a potência, onde K = P * t, sendo assim não há diminuição do erro ao utilizar-se períodos menores para plotar.

```
program ciclista

    implicit none

    integer i,n
    real*8 Ec,dt,Pc,m,p,A,K,t

     !declaram-se as variáveis com os valores dados no problema e utiliza-se
um  loop  para  o  cálculo  da  energia  cinética  em  cada  instante  e,  então,
imprime-se o instante e a velocidade tirada da energia cinética.

    K =0.d0
    Pc = 400.d0
```

```
    dt = 0.1d0
    t = 0.d0
    n = 600/dt
    m = 70.d0
    p = 0.d0
    A = 0.d0

    open(10,file='VxT.dat')

    do i=0,n
        K = K + (Pc - p*A*(2/m**3)**0.5*K**1.5)*dt
        write(10,*) t , (2*K/m)**0.5
        t = t + 0.1d0
    enddo

end program
```

Após plotar o gráfico da velocidade em função do tempo, adiciona-se ao código o cálculo da posição a partir do método dos trapézios, onde toma-se a velocidade instantânea e multiplica-se por um intervalo de tempo “dt” pequeno. Quanto menor o “dt”, mais preciso é o processo de integração, desde que a precisão do Fortran seja compatível.

```
program ciclista

    implicit none

    integer i,n
    real*8 Ec,dt,Pc,m,p,A,K,t,v,X

     !aqui foram adicionadas as variaveis v e X relacionadas à velocidade e
posição
     !declaram-se as variáveis com os valores dados no problema e utiliza-se
um loop para o cálculo da energia cinética em cada instante e, então,
imprime-se o instante e a velocidade tirada da energia cinética.

    K =0.d0
    Pc = 400.d0
    dt = 0.1d0
    t = 0.d0
    n = 600/dt
    m = 70.d0
    p = 0.d0
    A = 0.d0
    X = 0.d0
```

```
     !calcula-se a velocidade instantânea e utiliza-se do método do trapézio
para calcular a posição, partindo de X=0
    do i=0,n
        K = K + (Pc - p*A*(2/m**3)**0.5*K**1.5)*dt
        v = (2*K/m)**0.5
        X = X + v*dt
    enddo

    write(*,*)X

end program
```

Obtém-se então um deslocamento de **33135.546552071188 m** utilizando intervalo “dt” de 0.1 s e um deslocamento de **33124.388271131800 m** utilizando um intervalo “dt” de 0.01 s. Como previsto, os valores são muito semelhantes, mudando apenas devido ao erro na integração.

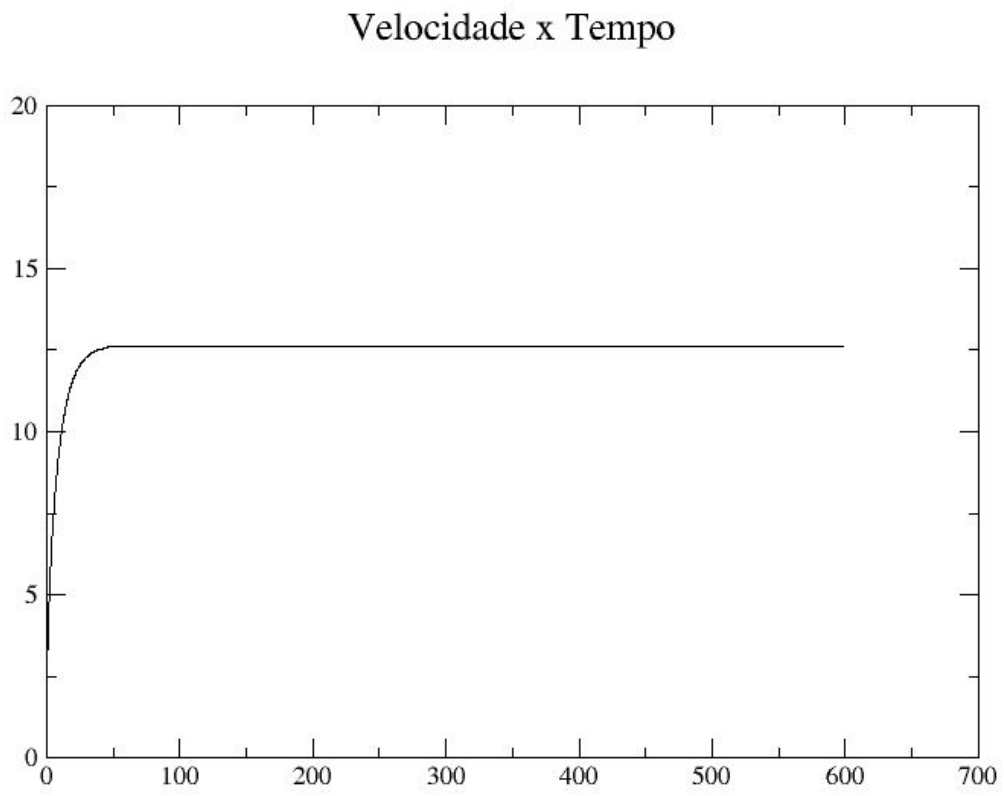
1.2 - Com efeito resistivo

Ao adicionar-se o efeito resistivo do fluido ao problema, nota-se a necessidade do uso do computador para realizar os cálculos. Dessa vez, soma-se à potência gerada pelo ciclista uma potência resistiva provinda do fluido. Esta depende diretamente da velocidade, então o cálculo torna-se mais complexo e a partir da equação, é esperado que para intervalos menores de tempo dt, haverá uma precisão maior.

A velocidade máxima esperada teoricamente ocorre quando a parte da equação que multipla “dt” é zero, assim não há incremento da energia cinética e, consequentemente não há incremento da velocidade. O valor obtido teoricamente é **12,70599 m/s**.

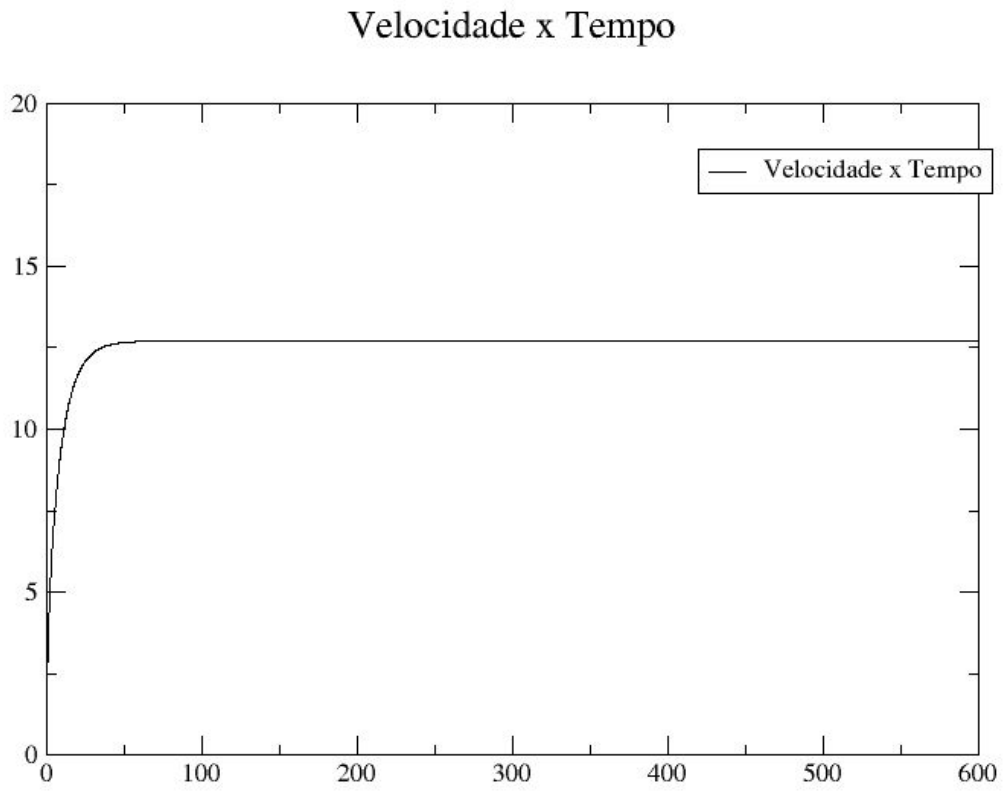
$$Pc = p \cdot A \sqrt{\frac{2}{m^3}} K^{1.5}$$

Considerando intervalos dt de 0.1s, obtém-se o seguinte gráfico:



Nota-se então que a velocidade terminal é **12.705988560341595 m/s**, que é atingida em **306.7s**.

Considerando intervalos “dt” de 0.01s, obtém-se:



Foi obtida a velocidade terminal de **12.705988560341162**, no tempo **286.43 s**. Nota-se então que não fez muita diferença utilizar 0.1s ou utilizar 0.01 s para o cálculo. Ambos tiveram uma precisão boa. Uma observação curiosa aqui é que em 0.1s demorou-se mais para atingir a velocidade terminal do que em 0.01s.

```
program ciclistaresistido

    implicit none

    integer i,n
    real*8 Ec,dt,Pc,m,p,A,K,t

     !declaram-se as variáveis com os valores dados no problema e utiliza-se
um  loop  para  o  cálculo  da  energia  cinética  em  cada  instante  e,  então,
imprime-se o instante e a velocidade tirada da energia cinética
     !dessa vez, simplesmente atualiza-se os valores da densidade do fluido e
da área da secção reta transversa ao movimento

    K =0.d0
    Pc = 400.d0
    dt = 0.01d0 !para 0.1s, muda-se para 0.1d0
    t = 0.d0
    n = 600/dt
    m = 70.d0
```

```
    p = 1.3d0
    A = 0.3d0

    open(10,file='VxT.dat')

    do i=0,n
        K = K + (Pc - p*A*(2/m**3)**0.5*K**1.5)*dt
        write(10,*) t , (2*K/m)**0.5
        t = t + 0.01d0 !para 0.1s, muda-se 0.01d0 para 0.1d0
    enddo

end program
```

1.3 - Variando a área

Após realizar as simulações com um valor arbitrário de área 1.3 m, é interessante analisar valores mais realistas de área de contato para um ciclista. Serão considerados aqui três casos.

Segundo dados do IBGE, para jovens entre 20 e 24 anos, as medidas de altura e peso médio do homem brasileiro são de 1,73 m e 69,4 kg. Já para as mulheres, a média é de 1,61 m e 57,8 kg. Fazendo a média desses dados, obtém-se 1,67 m e 63,6 kg .

https://biblioteca.ibge.gov.br/visualizacao/livros/liv45419.pdf

Tendo esses dados, a primeira área A1 será referente ao brasileiro médio, a segunda a um ciclista de curta distância e a terceira a um ciclista de longa distância. Sabe-se que um ciclista de curta distância apresenta mais força e massa do que a média, visto que eles realizam movimentos rápidos por um tempo menor, assim será considerada que sua altura e peso são 20% maiores do que a média para o cálculo da área A2. Já ciclistas de longa distância apresentam mais resistência muscular e menos massa, visto que sua atividade é mais lenta e por tempos maiores, assim será considerada que sua altura e peso são 20 % menores do que a média para a área A3.

Para o cálculo da área corporal, será utilizada a fórmula de Mosteller, que relaciona a área corporal com a altura e peso de uma pessoa. Tomaremos aqui metade da área como uma aproximação para a área transversal que oferece resistência ao ar.

https://prezi.com/1nzx8jdr3ted/area-da-superficie-do-corpo-humano/

$$A = \sqrt{\frac{massa^{*}\, altura}{3600}}$$

Além disso, é possível inclinar o corpo como um método de reduzir área de contato, um método comum utilizado por ciclistas profissionais e amadores. Considerando possível inclinar o tronco a um ângulo de 45 graus, é possível reduzir a área de contato da parte

superior à metade. Considerando a área superior como metade da área corporal obtida a partir da equação, a área total é reduzida em um quarto.

"No movimento de pedalada, o ciclista adapta a postura em uma posição em que os joelhos medializam por meio de uma adução dos quadris. Esta posição tem como objetivo principal reduzir a área frontal do conjunto ciclista-bicicleta, criando assim, uma estratégia aerodinâmica empregada em competições e treinamentos (STOELBEN et al., 2016). "
https://home.unicruz.edu.br/seminario/anais/anais-2017/XXII%20SEMIN%C3%81RIO%20INTERINSTITUCIONAL%202017%20-%20ANAIS/GRADUA%C3%87%C3%83O%20-%20TRABALHO%20COMPLETO_Ci%C3%AAncias%20Biol%C3%B3gicas%20e%20Sa%C3%BAde/A%20BIOMEC%C3%82NICA%20DO%20CICLISMO%20UMA%20REVIS%C3%83O%20DE%20LITERATURA.pdf

Áreas obtidas:

	Sem adaptação(m2)	Com adaptação(m2)
A1	0,86	0,65
A2	1,03	0,78
A3	0,69	0,52

Utilizando o código da 1.2, é possível encontrar uma aproximação mais realista para velocidade final e deslocamento, ao substituir a área na equação.

Velocidade terminal:

	Sem adaptação	Com adaptação
A1	8,87 m/s	9,73 m/s
A2	8,35 m/s	9,16 m/s
A3	9,54 m/s	10,42 m/s

Deslocamento

	Sem adaptação	Com adaptação
A1	5293.86 m	5804.85 m
A2	4988.22 m	5466.71 m
A3	5691.63 m	6206.78 m

Dessa vez, foram obtidos resultados razoáveis de deslocamento e velocidade para ciclistas, próximos a 5km, o que mostra novamente o quão absurdos são os

valores obtidos na 1a de velocidade final e deslocamento ao desconsiderar-se a resistência do ar, onde o deslocamento seria maior de 30 km e a velocidade terminal superior a 80 m/s.

2 - Efeito resistivo do ar em movimentos bidimensionais

2.1 - Trajetória

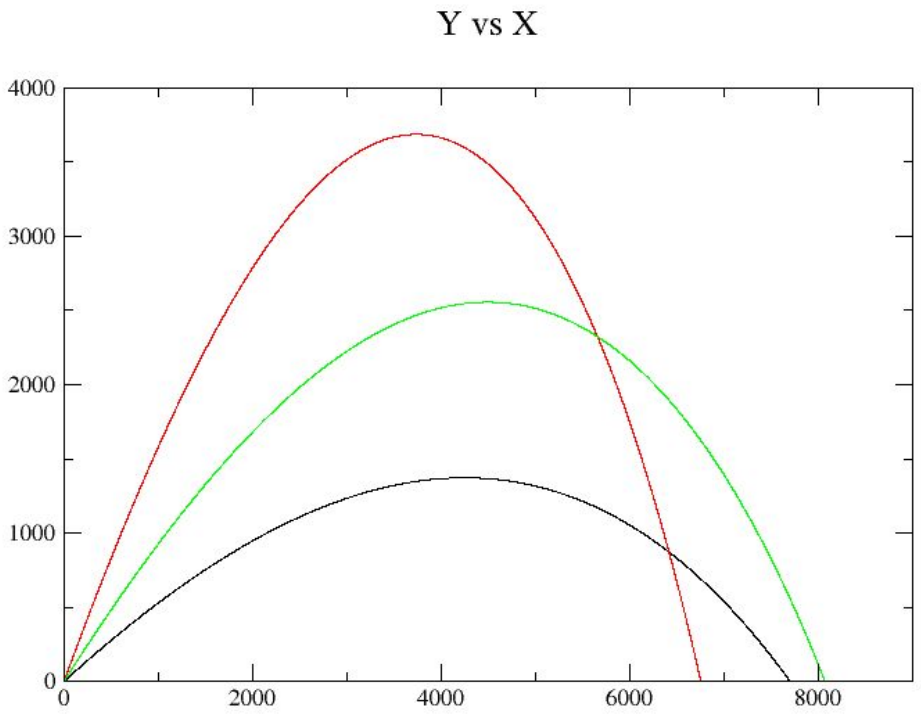
Em alguns casos, não é possível reduzir um movimento para uma dimensão, e é necessário, então, dividi-lo em dois utilizando os eixos XY. Um caso tradicional assim é o lançamento de projéteis, onde define-se um ângulo theta a partir do eixo X em direção ao eixo Y em relação à direção dos projéteis e, a partir do seno e cosseno de tal ângulo, que varia com o tempo, encontra-se a velocidade do projétil em cada eixo.

Então, substituindo o seno e cosseno de theta a partir das componentes da velocidade, e tendo em conta as velocidades iniciais em cada eixo, sabe-se que a gravidade atua apenas no eixo Y e, assim, calcula-se a trajetória dos projéteis.

No caso, será considerado um canhão, onde sabe-se a massa do projétil, a densidade do ar local e o coeficiente de arraste.

Sabendo o diâmetro da esfera(projétil), simplifica-se a área de contato para um círculo, de área $\pi \cdot R^2$.

Utilizando intervalo dt de 0.01 s, obtém-se o gráfico:



$\pi/6$ $\pi/4$ $\pi/3$

Ao mudar o intervalo dt para 0.001s não é notória a diferença no gráfico e os valores obtidos são semelhantes.

```
program projetil

    implicit none

    real*8 theta,pi,v0,m,g,p,Cd,A,dt
    real*8 vx,vy,X,Y,square

     !primeiramente define-se todas as variáveis necessárias
para a simulação, com o mesmo nome que pedido no projeto
```

```
    pi = 4*atan(1.d0)

!a área da secção transversa ao movimento é um círculo
!as outras variáveis têm o valor inicial dado no projeto

       theta = pi/6 !muda-se aqui o valor de theta quando
necessário
    A = pi * (0.1491/2)**2

    v0 = 377
    m = 42
    g = 9.8
    p = 1.3
    Cd = 0.295
    dt = 0.001 !muda-se aqui o dt para 0.001 para a comparação

     !a partir da trigonometria do problema, encontra-se os
valores de vx e vy iniciais

    vx = v0 * cos(theta)
    vy = v0 * sin(theta)

!para o funcionamento do código, é necessário tomar os valores
de X e Y no primeiro intervalo dt

    X = vx*dt
    Y = vy*dt

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

     !aqui será utilizado um loop que calcula as velocidades
vx e vy após o intervalo dt e com essas velocidades,
calcula-se as posições X e Y
     !aqui vale notar que foi necessário utilizar uma variável
chamada squares para que em ambas as velocidades nos eixos X e
Y, utiliza-se os valores anteriores e não os atualizados
durante o loop

    do while (Y>0)
        write(10,*) X,Y
```

```
        square = sqrt(vx**2 + vy**2)

        vx = vx - p*A*Cd/(2*m)*vx*square*dt
        vy = vy - (g + p*A*Cd/(2*m)*vy*square)*dt

        X = X + vx*dt
        Y = Y + vy*dt
    enddo

end program
```

2.2 - Alcance(ângulo máximo)

Para o cálculo do ângulo máximo, que é o ângulo que permite o maior alcance do projétil, utiliza-se o código anterior, tomando o último valor obtido de X no loop como o alcance. Assim, o código vira uma função que é chamada pelo código principal para calcular o alcance e depende apenas de theta, utilizando novamente os outros dados do projeto. Varia-se então o theta para encontrar o ângulo que permite o maior alcance. Sabe-se que o theta estará entre 0 e $\pi/2$ e varia-o até que se encontre o valor máximo de alcance.

Para variá-lo, comparar-se dois métodos, primeiro variando até achar um valor máximo de theta, de forma que o alcance referente ao theta i seja maior do que o referente ao theta i+1, isso valerá se o valor do alcance aumentar com o theta até o ângulo máximo. Então, da outra forma, simplesmente plota-se um gráfico e toma-se o valor máximo.

```
program projetil

    implicit none
    real*8,external :: Alcance
    real*8 theta,X2,X1,dx

       theta = 0.1d0 !muda-se aqui o valor para precisão
diferente
    X1 = Alcance(theta)

    do while (dx>0)
        theta = theta + 0.1d0
```

```
        X2 = Alcance(theta)
        dx = X2 - X1
        X1 = X2
    enddo
          write(*,*) theta - 0.1d0 !muda-se aqui o valor para
precisão diferente
end program



function Alcance(theta)

    implicit none

    real*8 Alcance
    real*8 theta,pi,v0,m,g,p,Cd,A,dt
    real*8 vx,vy,X,Y,square

     !primeiramente define-se todas as variáveis necessárias
para a simulação, com o mesmo nome que pedido no projeto

    pi = 4*atan(1.d0)

    !a área da secção transversa ao movimento é um círculo
    !as outras variáveis têm o valor inicial dado no projeto


    A = pi * (0.1491/2)**2

    v0 = 377
    m = 42
    g = 9.8
    p = 1.3
    Cd = 0.295
    dt = 0.01

     !a partir da trigonometria do problema, encontra-se os
valores de vx e vy iniciais

    vx = v0 * cos(theta)
    vy = v0 * sin(theta)
```

```
     !para o funcionamento do código, é necessário tomar os
valores de X e Y no primeiro intervalo dt

    X = vx*dt
    Y = vy*dt



     !aqui será utilizado um loop que calcula as velocidades
vx e vy após o intervalo dt e com essas velocidades,
calcula-se as posições X e Y
     !aqui vale notar que foi necessário utilizar uma variável
chamada squares para que em ambas as velocidades nos eixos X e
Y, utiliza-se os valores anteriores e não os atualizados
durante o loop

    do while (Y>0)

        square = sqrt(vx**2 + vy**2)

        vx = vx - p*A*Cd/(2*m)*vx*square*dt
        vy = vy - (g + p*A*Cd/(2*m)*vy*square)*dt

        X = X + vx*dt
        Y = Y + vy*dt
    enddo

    Alcance = X

    return
end function
```

Rodando o código, obtém-se o valor de theta máximo de **0.7**. Notando que o código rodou rapidamente, muda-se a precisão de 0.1 para 0.001 e calcula--se novamente o theta máximo. Encontra-se então o valor de **0.679** para o theta máximo.

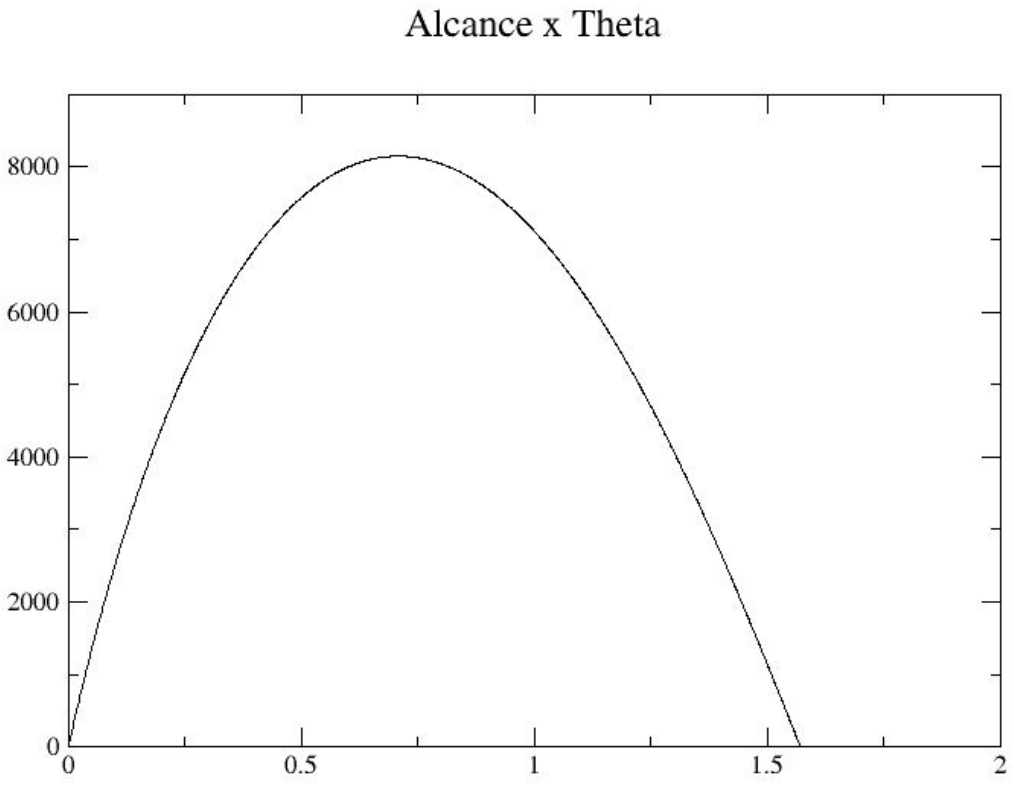
Então, modifica-se novamente o código para plotar os valores de alcance e theta, tomando o valor máximo a partir do gráfico e comparar para descobrir qual é o método mais efetivo.

```
program projetilgraf

    implicit none
    real*8,external :: Alcance
    real*8 theta,X
    integer i

    theta = 0.d0

    open(10,file="Alcancextheta")

    do i=1,1570
        theta = theta + 0.001d0
        X = Alcance(theta)
        write(10,*)theta,X
    enddo


end program



function Alcance(theta)

    implicit none

    real*8 Alcance
    real*8 theta,pi,v0,m,g,p,Cd,A,dt
    real*8 vx,vy,X,Y,square

     !primeiramente define-se todas as variáveis necessárias
para a simulação, com o mesmo nome que pedido no projeto

    pi = 4*atan(1.d0)

    !a área da secção transversa ao movimento é um círculo
    !as outras variáveis têm o valor inicial dado no projeto


    A = pi * (0.1491/2)**2

    v0 = 377
```

```
    m = 42
    g = 9.8
    p = 1.3
    Cd = 0.295
    dt = 0.01

     !a partir da trigonometria do problema, encontra-se os valores de vx e vy iniciais

    vx = v0 * cos(theta)
    vy = v0 * sin(theta)

     !para o funcionamento do código, é necessário tomar os valores de X e Y no primeiro intervalo dt

    X = vx*dt
    Y = vy*dt



     !aqui será utilizado um loop que calcula as velocidades vx e vy após o intervalo dt e com essas velocidades, calcula-se as posições X e Y
     !aqui vale notar que foi necessário utilizar uma variável chamada squares para que em ambas as velocidades nos eixos X e Y, utiliza-se os valores anteriores e não os atualizados durante o loop

    do while (Y>0)

        square = sqrt(vx**2 + vy**2)

        vx = vx - p*A*Cd/(2*m)*vx*square*dt
        vy = vy - (g + p*A*Cd/(2*m)*vy*square)*dt

        X = X + vx*dt
        Y = Y + vy*dt
    enddo

    Alcance = X

    return
```

```
end function
```

Obtém-se então o seguinte gráfico:

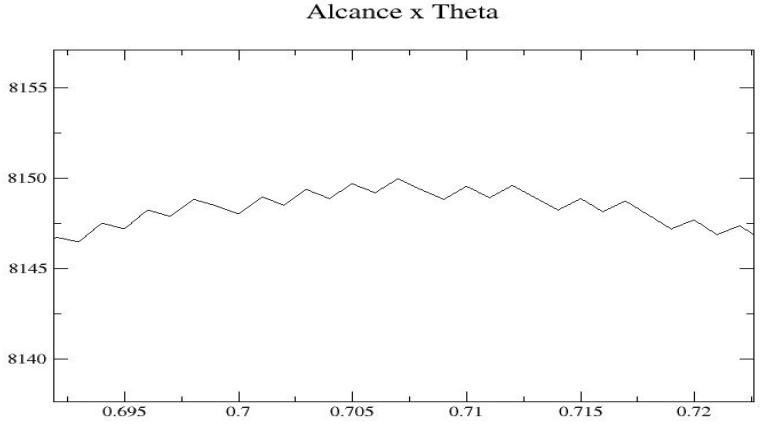


Onde, a partir do gráfico e dos dados obtidos, encontra-se o valor de theta máximo em torno de **0.71**, em radianos.

Analisando os resultados, encontra-se a razão para essa discrepância:

667	0.6670000000000048	8128.1207267642803
668	0.6680000000000048	8129.7250720557568
669	0.6690000000000048	8129.9022026016200
670	0.6700000000000048	8131.4651059676344
671	0.6710000000000048	8131.6042514276414
672	0.6720000000000049	8133.1256869803610
673	0.6730000000000049	8133.2268191742014
674	0.6740000000000049	8134.7067609486376
675	0.6750000000000049	8136.1650170864386
676	0.6760000000000049	8136.2082735913127
677	0.6770000000000049	8137.6249975494593
678	0.6780000000000049	8137.6301704909165
679	0.6790000000000049	8139.0053361064001
680	0.6800000000000049	8138.9723970936875
681	0.6810000000000049	8140.3059781260690
682	0.6820000000000049	8140.2348987093501
683	0.6830000000000050	8141.5268688408623
684	0.6840000000000050	8142.7970977824425
685	0.6850000000000050	8142.6679533462857
686	0.6860000000000050	8143.8965324342626
687	0.6870000000000050	8143.7291766009994
688	0.6880000000000050	8144.9160793192632
689	0.6890000000000050	8144.7104834264373

Os resultados seguem crescendo até o valor de 0.68, onde o alcance diminui um pouco e, logo após, volta a aumentar. Ao dar zoom no gráfico, nota-se que isso acontece constantemente com a função:



2.3 - Densidade do ar e coeficiente de arraste

As equações utilizadas anteriormente para o cálculo nos itens 2.1 e 2.2 são apenas aproximações. O problema real é muito mais complexo, visto que a resistência do ar varia com a altitude e para lançamentos de projéteis com altitudes maiores, a aproximação não é satisfatória. Outro problema é que o coeficiente de arraste depende da velocidade do fluido, então essa aproximação também não é satisfatória para velocidades que variam muito.

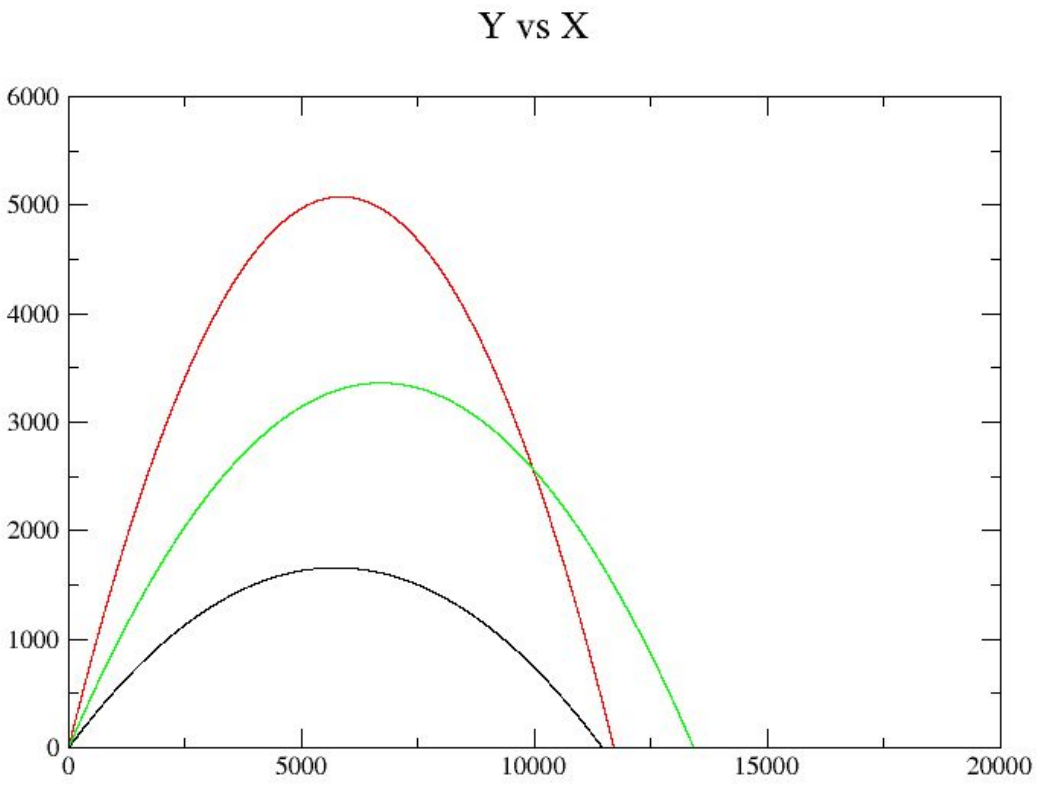
Será utilizada então a seguinte equação para a densidade do ar, onde há uma dependência em relação a Y e algumas constantes:

$$\rho = \rho_0 \left(1 - \frac{\beta}{T_0} y\right)^{\alpha}$$

E, para o coeficiente de arraste, será mantida a aproximação para 0.295 em velocidades menores que a do som (343 m/s) e será adotada uma nova de 0.5 para velocidades maiores.

Para aplicar isso ao código, basta simplesmente adicionar o cálculo da densidade antes do cálculo das velocidades e posições, além de adicionar um “if” para checar a questão do coeficiente de arraste.

O gráfico para os diferentes valores de theta ficam:



π/6 π/4 π/3

Comparando com os gráficos anteriores, nota-se dessa vez que os alcances foram consideravelmente maiores (alguns quilômetros a mais) e o gráfico do lançamento para $\pi/3$ obteve alcance maior do que $\pi/6$, diferentemente do caso anterior. A principal razão para isso é a diminuição da densidade do ar ao atingir altitudes maiores.

E o código para calcular o ângulo fica, então:

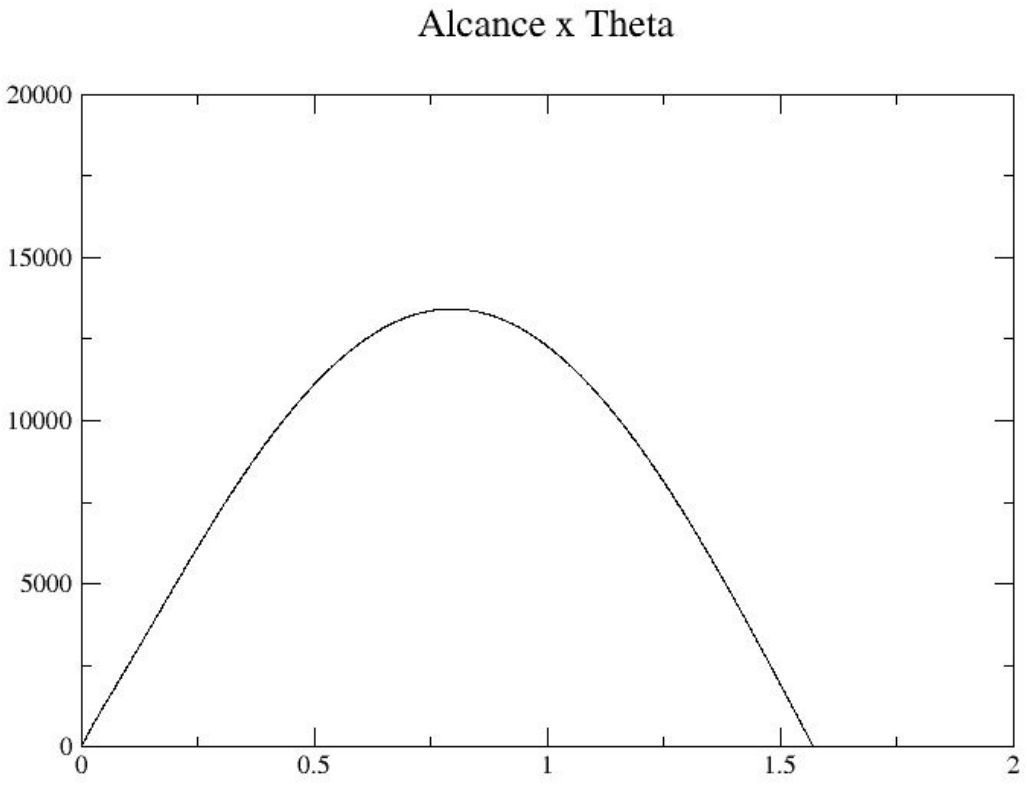
```
program projetilgraf

    implicit none
    real*8,external :: Alcance
    real*8 theta,X
```

```
    integer i

    theta = 0.d0

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

    do i=1,1570
        theta = theta + 0.001d0
        X = Alcance(theta)
        write(10,*)theta,X
    enddo


end program



function Alcance(theta)

    implicit none

    real*8 Alcance
    real*8 theta,pi,v0,m,g,p,Cd,A,dt
    real*8 vx,vy,X,Y,square

   !primeiramente define-se todas as variáveis necessárias
para a simulação, com o mesmo nome que pedido no projeto

    pi = 4*atan(1.d0)

    !a área da secção transversa ao movimento é um círculo
    !as outras variáveis têm o valor inicial dado no projeto


    A = pi * (0.1491/2)**2

    v0 = 377
    m = 42
    g = 9.8
    p = 1.3
    Cd = 0.295
    dt = 0.01
```

```
     !a partir da trigonometria do problema, encontra-se os
valores de vx e vy iniciais

    vx = v0 * cos(theta)
    vy = v0 * sin(theta)

    !para o funcionamento do código, é necessário tomar os
valores de X e Y no primeiro intervalo dt

    X = vx*dt
    Y = vy*dt



     !aqui será utilizado um loop que calcula as velocidades
vx e vy após o intervalo dt e com essas velocidades,
calcula-se as posições X e Y
     !aqui vale notar que foi necessário utilizar uma variável
chamada squares para que em ambas as velocidades nos eixos X e
Y, utiliza-se os valores anteriores e não os atualizados
durante o loop

    do while (Y>0)

        square = sqrt(vx**2 + vy**2)

        if (square > 343) then
            Cd = 0.5
        else
            Cd = 0.295
        end if

        p = p*(1 - 0.00649/300 * Y)**4.256 !adidiona-se ao
codigo a função para a densidade do ar

        vx = vx - p*A*Cd/(2*m)*vx*square*dt
        vy = vy - (g + p*A*Cd/(2*m)*vy*square)*dt

        X = X + vx*dt
        Y = Y + vy*dt
    enddo
```

```
    Alcance = X

    return
end function
```

Obtém-se por fim, o gráfico:



Onde o valor de theta máximo é de **0.795** radianos.

2.4 - Velocidade de lançamento

Tomando-se o valor de theta máximo obtido anteriormente de 0.795 radianos, agora será analisado se uma variação de +- 1% na velocidade muda muito o alcance do projétil.

Utilizando-se da função para o cálculo do alcance do código do item anterior, modifica-se o valor da velocidade anterior e define-se o theta. Por fim, obtém-se:

-1% :13164.231923013407
Normal : 13426.538214657770
+1% : 13688.890526549945

Obteve-se uma diferença de cerca de 524,66 m entre a velocidade com erro de -1% e a velocidade com erro de +-1%. Sendo assim, ao atirar um projétil em um canhão, por exemplo, que tivesse esse erro de 1%, teria uma linha de 524,66 m em que o projétil poderia ter aterrissado. Dessa forma, não parece ser uma atividade trivial atirar com um projétil de longas distâncias, além do que foi feita uma aproximação no coeficiente de arraste que pode ter levado a erros grandes, o que torna o problema ainda mais complexo de resolver. É necessário então um estudo bem detalhado da trajetória e sobre atirar com uma velocidade inicial exata de lançamento.

```
program velocidade

    implicit none

    real*8,external :: Alcance

    write(*,*) Alcance(0.795d0)

end program

function Alcance(theta)

    implicit none

    real*8 Alcance
    real*8 theta,pi,v0,m,g,p,Cd,A,dt
    real*8 vx,vy,X,Y,square
```

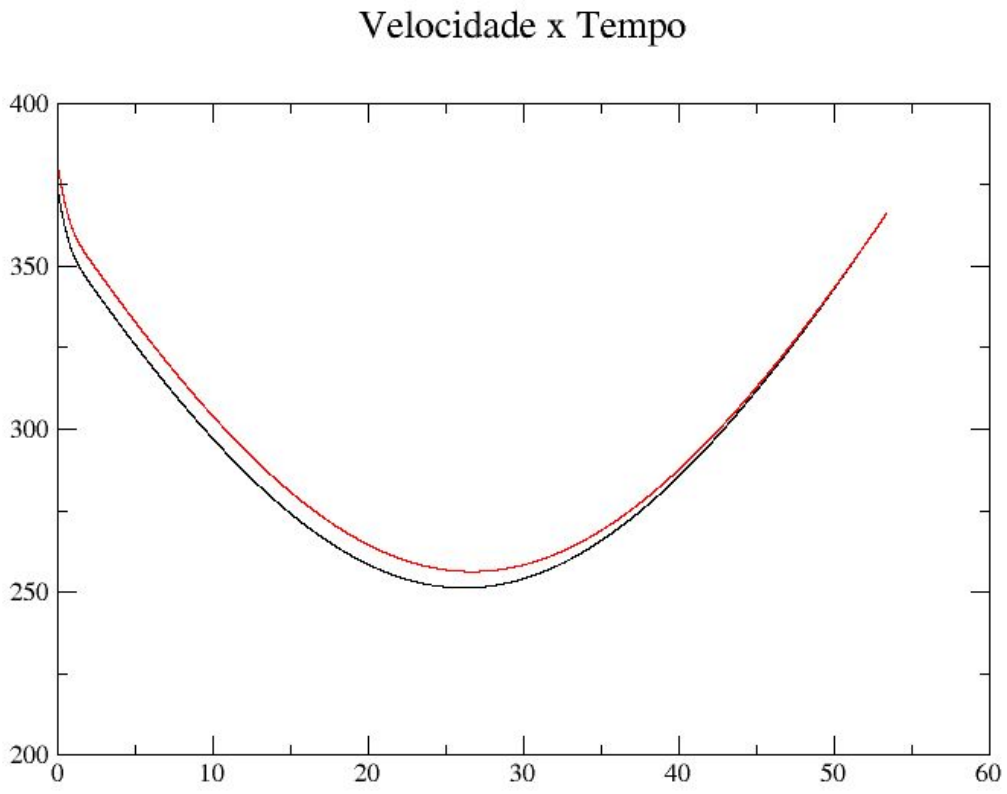
```
     !primeiramente define-se todas as variáveis necessárias
para a simulação, com o mesmo nome que pedido no projeto

    pi = 4*atan(1.d0)

    !a área da secção transversa ao movimento é um círculo
    !as outras variáveis têm o valor inicial dado no projeto


    A = pi * (0.1491/2)**2

     v0 = 377 * 1.01 ! multiplica-se por 1 para a velocidade
inicial, 1.01 para um porcento a mais e 0.99 para um por cento
a menos
    m = 42
    g = 9.8
    p = 1.3
    Cd = 0.295
    dt = 0.01

     !a partir da trigonometria do problema, encontra-se os
valores de vx e vy iniciais

    vx = v0 * cos(theta)
    vy = v0 * sin(theta)

     !para o funcionamento do código, é necessário tomar os
valores de X e Y no primeiro intervalo dt

    X = vx*dt
    Y = vy*dt



     !aqui será utilizado um loop que calcula as velocidades
vx e vy após o intervalo dt e com essas velocidades,
calcula-se as posições X e Y
     !aqui vale notar que foi necessário utilizar uma variável
chamada squares para que em ambas as velocidades nos eixos X e
Y, utiliza-se os valores anteriores e não os atualizados
durante o loop
```

```
    do while (Y>0)

        square = sqrt(vx**2 + vy**2)

        if (square > 343) then
            Cd = 0.5
        else
            Cd = 0.295
        end if

          p = p*(1 - 0.00649/300 * Y)**4.256 !adiciona-se ao
código a função para a densidade do ar

        vx = vx - p*A*Cd/(2*m)*vx*square*dt
        vy = vy - (g + p*A*Cd/(2*m)*vy*square)*dt

        X = X + vx*dt
        Y = Y + vy*dt
    enddo

    Alcance = X

    return
end function
```

2.5 - Velocidade em função do tempo

Para graficar a velocidade em função do tempo, toma-se o mesmo código anterior, mas dessa vez adiciona-se ao loop uma tarefa para imprimir a velocidade e o tempo num arquivo, além do próprio cálculo do tempo, uma nova variável “t” que deve ser adicionada.

Obtém-se o seguinte gráfico:



Nota-se que o formato do gráfico é o mesmo, mudando apenas a posição da curva. Nota-se que em preto começou-se com a velocidade menor e em vermelho com a velocidade maior.

```
program velocidade

    implicit none

    real*8,external :: Alcance

    write(*,*) Alcance(0.795d0)

end program

function Alcance(theta)

    implicit none

    real*8 Alcance
```

```
        real*8  theta,pi,v0,m,g,p,Cd,A,dt,t  !a  variavel  t  é
adicionada
    real*8 vx,vy,X,Y,square

     !primeiramente define-se todas as variáveis necessárias
para a simulação, com o mesmo nome que pedido no projeto

    pi = 4*atan(1.d0)

    !a área da secção transversa ao movimento é um círculo
    !as outras variáveis têm o valor inicial dado no projeto


    A = pi * (0.1491/2)**2

     v0 = 377 * 0.99 ! multiplica-se por 1 para a velocidade
inicial, 1.01 para um porcento a mais e 0.99 para um por cento
a menos
    m = 42
    g = 9.8
    p = 1.3
    Cd = 0.295
    dt = 0.01
    t = dt !t começa no primeiro intervalo dt

     !a partir da trigonometria do problema, encontra-se os
valores de vx e vy iniciais

    vx = v0 * cos(theta)
    vy = v0 * sin(theta)

      !para o funcionamento do código, é necessário tomar os
valores de X e Y no primeiro intervalo dt

    X = vx*dt
    Y = vy*dt



     !aqui será utilizado um loop que calcula as velocidades
vx  e  vy  após  o  intervalo  dt  e  com  essas  velocidades,
calcula-se as posições X e Y
```

```
     !aqui vale notar que foi necessário utilizar uma variável chamada squares para que em ambas as velocidades nos eixos X e Y, utiliza-se os valores anteriores e não os atualizados durante o loop

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

    do while (Y>0)

          square = sqrt(vx**2 + vy**2) !note aqui que square nada mais é do que o módulo da velocidade

        write(10,*) t,square

        if (square > 343) then
            Cd = 0.5
        else
            Cd = 0.295
        end if

          p = p*(1 - 0.00649/300 * Y)**4.256 !adiciona-se ao código a função para a densidade do ar

        vx = vx - p*A*Cd/(2*m)*vx*square*dt
        vy = vy - (g + p*A*Cd/(2*m)*vy*square)*dt

        X = X + vx*dt
        Y = Y + vy*dt

        t = t + dt !a cada iteração o tempo aumenta
    enddo

    Alcance = X

    return
end function
```

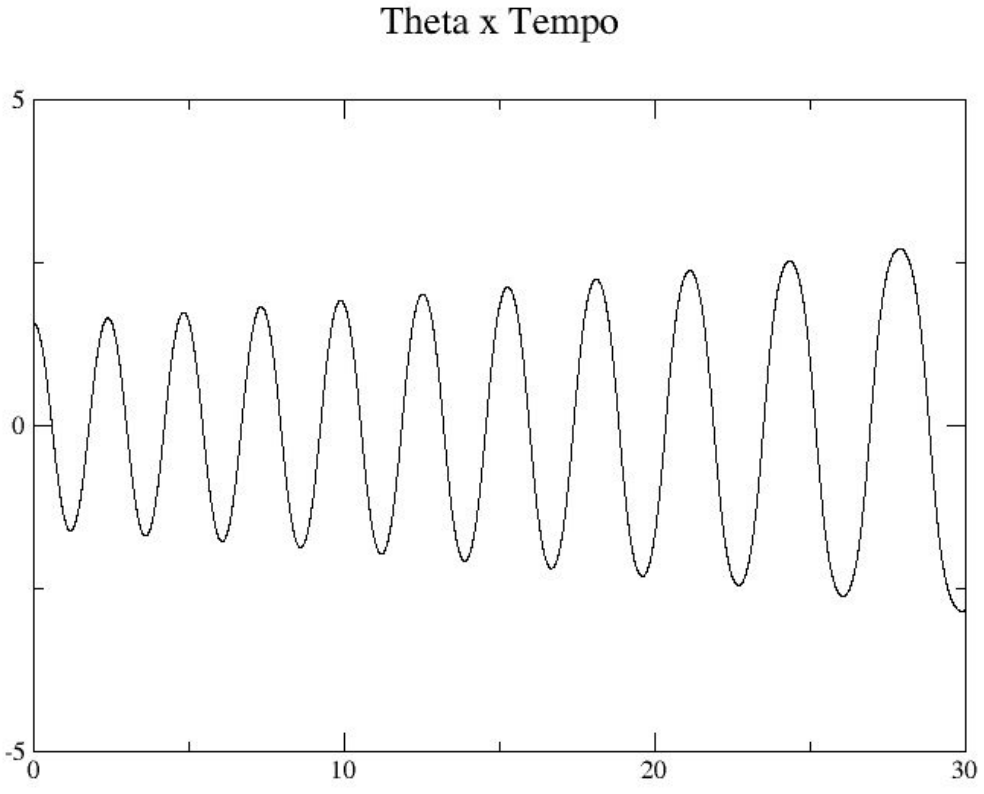
3 - Pêndulo Simples

O problema do pêndulo simples é um problema tradicional da física que pode ser resolvido facilmente com uma aproximação, mas torna-se mais complexo ao tentar resolvê-lo de uma forma mais realística.

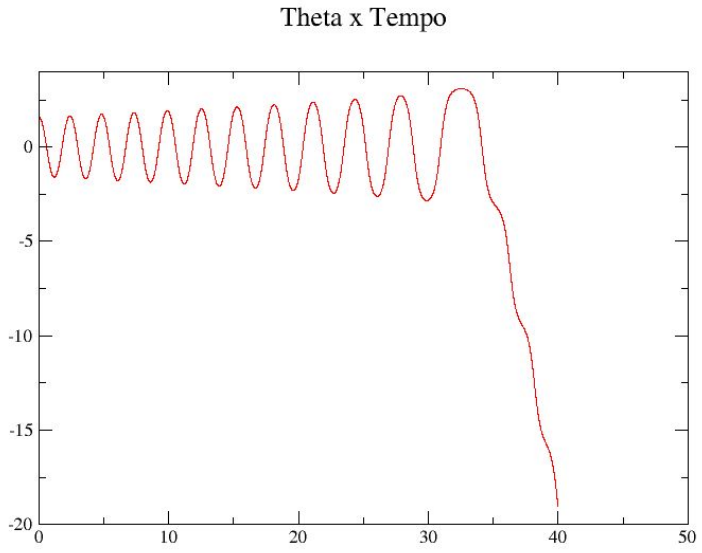
3.1 - Ângulo e Energia mecânica

Utiliza-se de um loop de forma semelhante ao segundo problema, sendo necessário definir um ângulo ómega temporário para aplicar o método de Euler, visto que o cálculo do ângulo depende de ómega e deve-se utilizar o ómega anterior(i) para o cálculo do ângulo i+1.

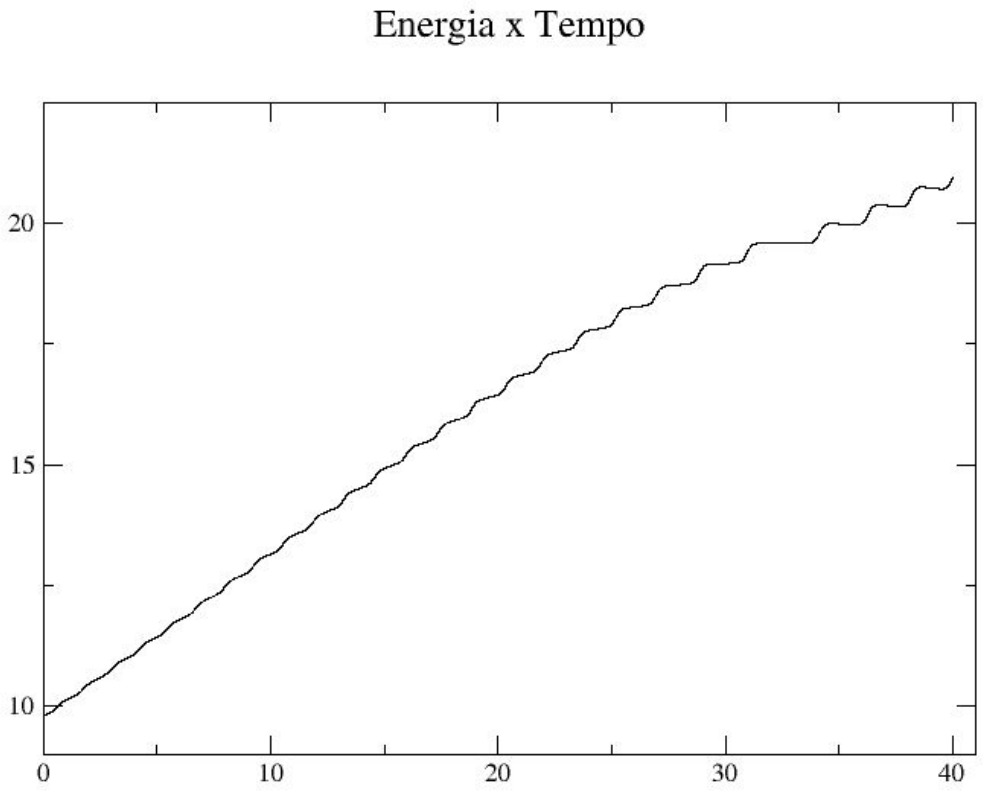
O ângulo de theta em função do tempo fica, então, uma senóide como esperado:



Nota-se também que a partir de cerca de 30s, na última fase, o processo deixa de ser efetivo:



E a Energia mecânica em função do tempo fica:



Nota-se que a energia mecânica não é conservada a partir do método de Euler.

```
program pendulo

    implicit none

    integer i,n
    real*8 w,wi,theta,E,dt,t
    real*8 g,l,m,pi

    m = 1.d0
    l = 1.d0
    g = 9.8d0
    pi = 4*atan(1.d0)

    w = 0.d0
    dt = 0.005d0
    theta = pi/2.d0

    n = 40.d0/0.005d0

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

    do i=1,n

          wi = w ! aqui nota-se que é utilizado o método de
Euler
        w = w - g/l * sin(theta) * dt
        theta = theta + wi * dt

        E = 0.5d0*m*(l*w)**2.d0 + m*g*l*(1 - cos(theta))
        t = t+dt

         write(10,*) w.theta !modifica-se aqui para o gráfico
desejado (X,Y)

    enddo

end program
```

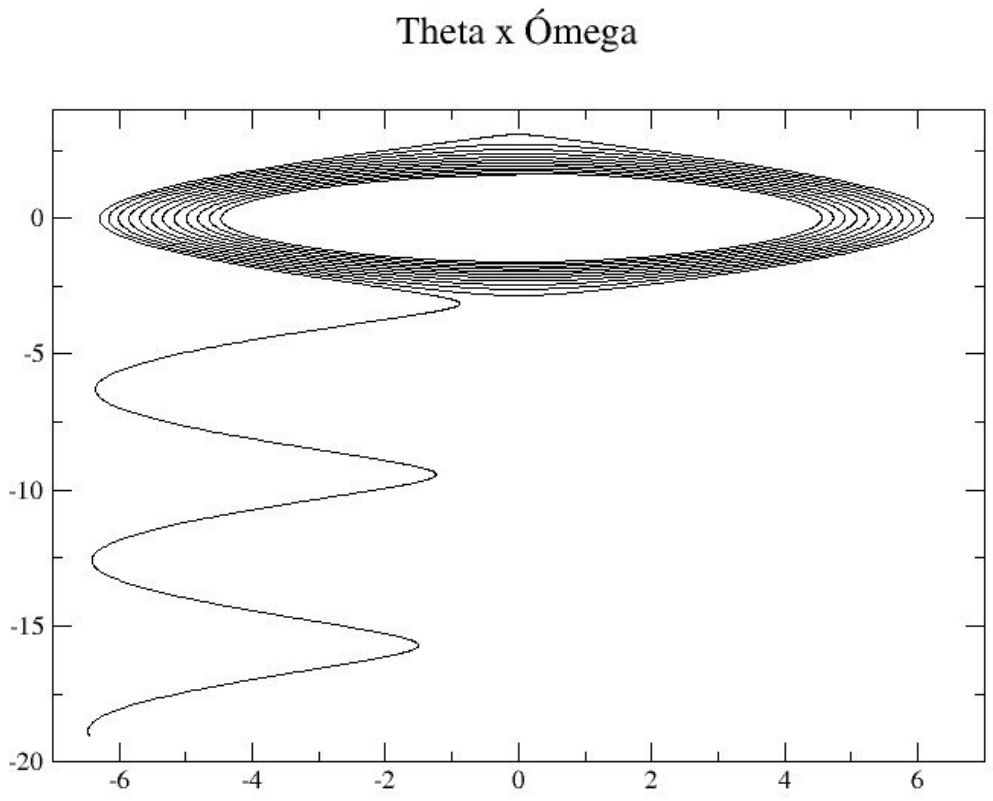
3.2 - Espaço de fase e Theta vs Ómega

Para encontrar o espaço de fase do movimento pendular, toma-se o tempo dividido pelo número de fases no gráfico do ângulo theta, obtendo-se um espaço de fase de **2,72s**. é fácil notar pelo gráfico que o ângulo aumenta com o tempo, visto que calculando o espaço de fase para a primeira fase obtém-se 2,37s.

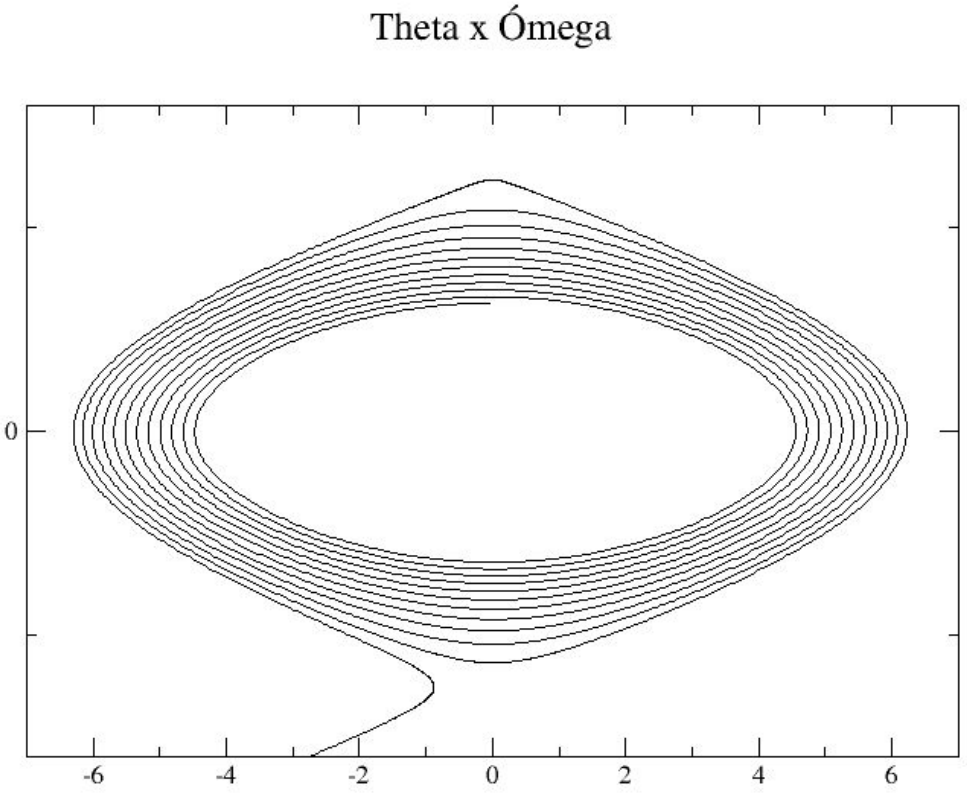
Para graficar o ângulo theta em função de ómega em função de theta, basta modificar as variáveis que são imprimidas no código do item anterior.

O gráfico fica então:

Onde, na parte inferior do eixo Y está a região onde o cálculo do theta deixou de ser preciso (acima de 30s).



Tomando apenas a parte superior do eixo Y do gráfico:



Nota-se então que a energia mecânica não é conservada, visto que não formou-se uma única elipse(theta e ómega são as variáveis que afetam a energia mecânica).

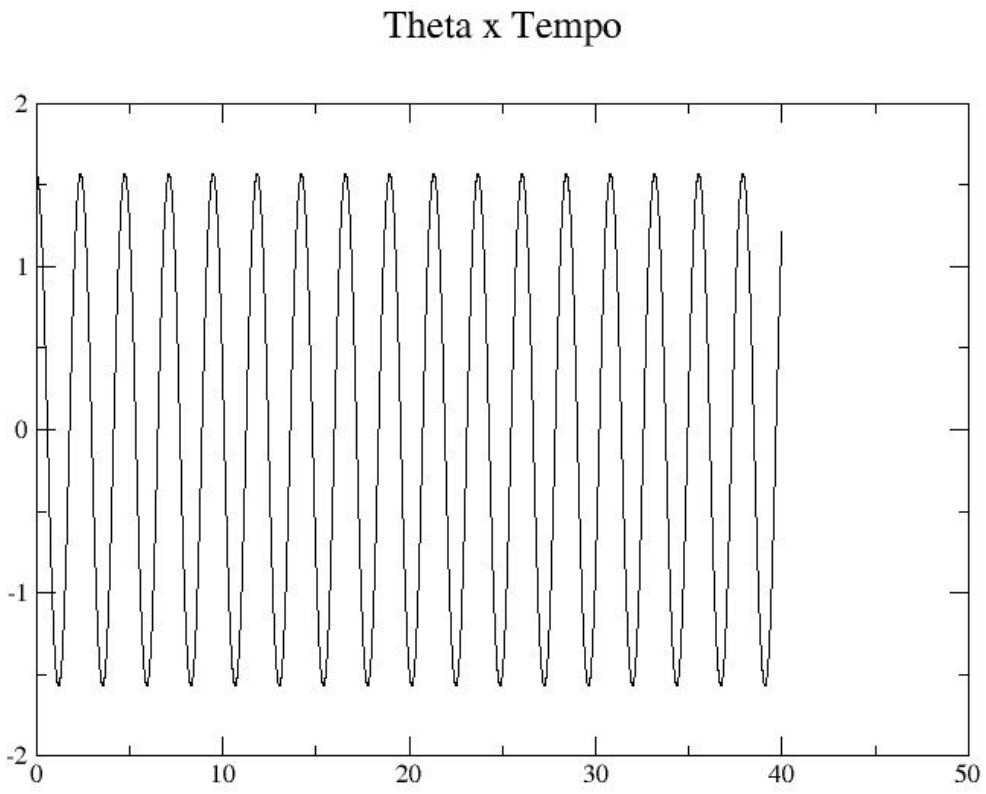
```
program pendulo

    implicit none

    integer i,n
    real*8 w,wi,theta,E,dt,t
    real*8 g,l,m,pi

    m = 1.d0
    l = 1.d0
    g = 9.8d0
    pi = 4*atan(1.d0)

    w = 0.d0
    dt = 0.005d0
```

```
    theta = pi/2.d0

    n = 40.d0/0.005d0

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

    do i=1,n

          wi = w ! aqui nota-se que é utilizado o método de
Euler
        w = w - g/l * sin(theta) * dt
        theta = theta + wi * dt

        E = 0.5d0*m*(l*w)**2.d0 + m*g*l*(1 - cos(theta))
        t = t+dt

         write(10,*) w,theta !modifica-se aqui para o gráfico
desejado (X,Y)

    enddo

end program
```

3.3 - Método de Euler-Cromer

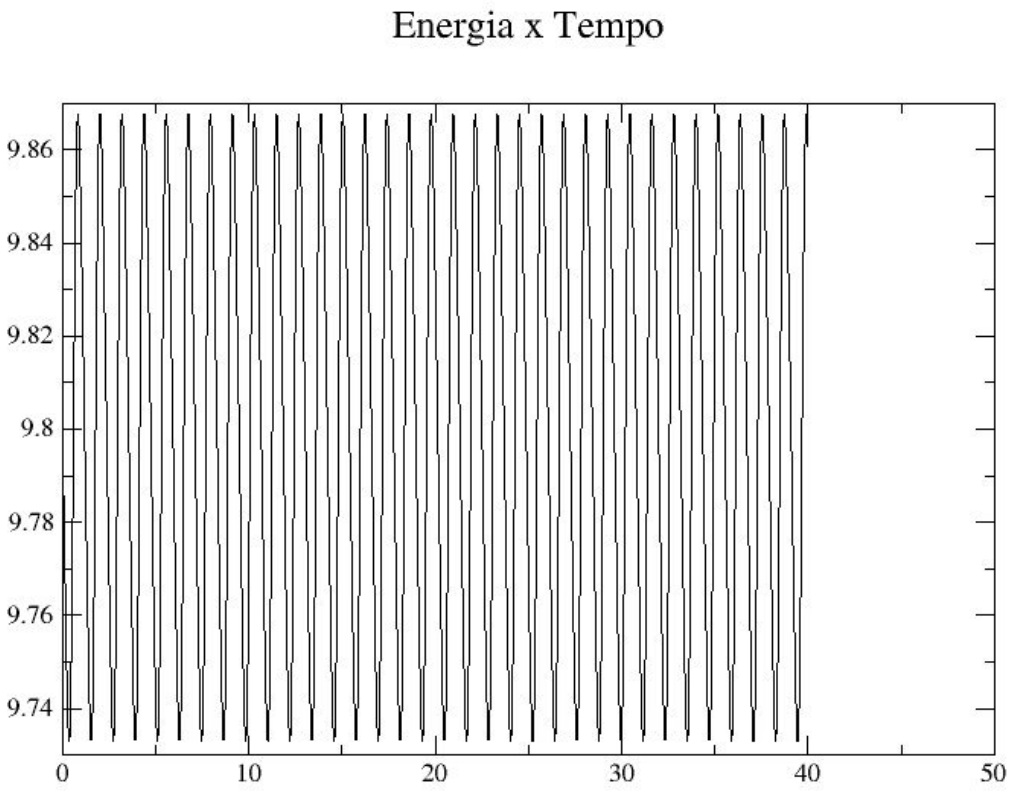
O método de Euler-Cromer é diferente do anterior por não necessitar de um ómega temporário e o seu maior benefício é a conservação da energia mecânica.

Para o ângulo em função do tempo, obtém-se o seguinte gráfico, uma função senoidal que varia entre $\pi/2$ e $-\pi/2$ como esperado:

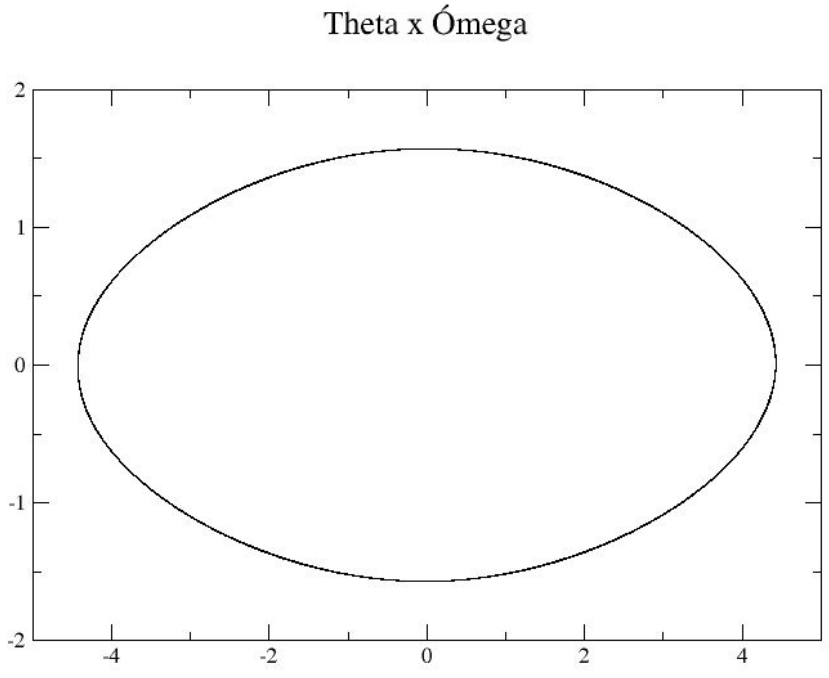


Vale notar aqui que o espaço de fase manteve-se constante, no valor de **2,35s**, semelhante ao encontrado na primeira fase do item anterior .

Já para a Energia mecânica em função do tempo, obtemos um gráfico parecido com o anterior, porém menos preciso(senoidal) e variando em valores muito menores no eixo Y , o que mostra que a energia mecânica no geral é conservada ao tomar-se intervalos de tempo maiores, mas varia em intervalos de tempo menores. Nota-se que essa variação é muito pequena, entre 9.74 e 9.86.



Graficando theta em função de ómega, dessa vez nota-se uma única elipse, mostrando que a energia mecânica foi conservada:



```
program pendulo

    implicit none

    integer i,n
    real*8 w,theta,E,dt,t
    real*8 g,l,m,pi

    m = 1.d0
    l = 1.d0
    g = 9.8d0
    pi = 4*atan(1.d0)

    w = 0.d0
    dt = 0.005d0
    theta = pi/2.d0

    n = 40.d0/0.005d0

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

    do i=1,n

        w = w - g/l * sin(theta) * dt
        theta = theta + w * dt

        E = 0.5d0*m*(l*w)**2.d0 + m*g*l*(1 - cos(theta))
        t = t+dt

         write(10,*) t,E,theta !modifica-se aqui para o gráfico
desejado (X,Y)

    enddo

end program
```

```
program pendulo

    implicit none

    integer i,n
    real*8 w,theta,E,dt,t
    real*8 g,l,m,pi

    m = 1.d0
    l = 1.d0
    g = 9.8d0
    pi = 4*atan(1.d0)

    w = 0.d0
    dt = 0.005d0
    theta = pi/2.d0

    n = 40.d0/0.005d0

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

    do i=1,n

        w = w - g/l * sin(theta) * dt
        theta = theta + w * dt

        E = 0.5d0*m*(l*w)**2.d0 + m*g*l*(1 - cos(theta))
        t = t+dt

          write(10,*) w,theta !modifica-se aqui para o gráfico
desejado (X,Y)

    enddo

end program
```