Introdução à Física Computacional: Projeto 3

Instituto de Física de São Carlos, Universidade de São Paulo

Solano E. S. Felício, n° USP 10288907

23 de abril de 2018

Introdução e equações

Este é um relatório entregue para avaliação da disciplina 7600017 – Introdução à Física Computacional. Cada seção é uma execução direta das tarefas do projeto 3, que trata de simulações de movimento realistas. Os problemas foram resolvidos durante as aulas 5 e 6 do curso. Aqui apresento os programas (em Fortran 90) e resultados (em forma de tabelas ou gráficos). Os gráficos foram feitos em Python com as bibliotecas NumPy e matplotlib.

O primeiro problema é modelar um ciclista que sofre resistência do ar. Se P é a potência gerada pelo ciclista, A é sua área frontal e ρ é a densidade do ar, um modelo razoável dá a velocidade terminal

$$v_{term} = \left(\frac{2P}{\rho A}\right)^{1/3} \tag{1}$$

em que a força gerada pelo ciclista equilibra a resistência do ar.

O segundo problema trata do efeito resistivo do ar no lançamento de projéteis. Resolvemos integrando por método de Euler, com as equações

$$x_{i+1} = x_i + v_{x,i}\Delta t \tag{2}$$

$$v_{x,i+1} = v_{x,i} - \frac{\gamma_2}{m}\left(1 - b\frac{y}{T_0}\right)^{\alpha}\sqrt{v_{x,i}^2 + v_{y,i}^2}\, v_{x,i}\Delta t \tag{3}$$

$$y_{i+1} = y_i + v_{y,i}\Delta t \tag{4}$$

$$v_{y,i+1} = v_{y,i} - g\Delta t - \frac{\gamma_2}{m}\left(1 - b\frac{y}{T_0}\right)^{\alpha}\sqrt{v_{x,i}^2 + v_{y,i}^2}\, v_{y,i}\Delta t \tag{5}$$

cujos símbolos foram definidos no enunciado do projeto.

O último problema é um pêndulo simples fora do regime de pequenas oscilações. Nesse caso, temos

$$T = 4\sqrt{\frac{L}{g}}\int_0^{\pi/2}\frac{du}{\sqrt{1 - k^2\sin^2 u}}, \text{ onde } k = \sin\frac{\theta_0}{2} \tag{6}$$

ou, aproximadamente,

$$T \approx 2\pi\sqrt{\frac{L}{g}}\left(1 + \frac{\theta_0^2}{16}\right) \tag{7}$$

o que se aproxima do valor esperado $2\pi\sqrt{L/g}$ quando $\theta_0 \to 0$.

1 Efeito resistivo do ar em bicicletas

Os itens (a), (b) e (c) podem ser resolvidos com um único programa, dado abaixo. O programa lê os valores de ρ (em kg/m^3) e A (em m^2) e retorna uma tabela com o tempo (de 0 a 300 s), a velocidade, e a posição (obtida pelo método do trapézio). Para o item (a), basta dar $\rho = 0$; para os itens (b) e (c), entramos com $\rho = 1{,}3$ kg/m^3, em (c), variamos os valores de A, que em (b) é 0,333 m^2.

```
program ciclista
implicit none

real*8 v0, vi, vn, xi, xn
real*8 m, P, dt, rho, A
```

```
integer i, imax

parameter (v0 = 4.d0)
parameter (m = 70.d0)
parameter (P = 400.d0)
parameter (dt = 0.1d0)
parameter (imax = 3000) ! 300 segundos / dt

read(*,*) rho, A

vi = v0
xi = 0
do i=0,imax
    write(*,*) i*dt, vi, xi

    vn = vi + (P/(m*vi) - rho*A*vi*vi/(2.d0*m))*dt
    xi = xi + 0.5d0*(vi+vn)*dt

    vi = vn
end do

end program ciclista
```

No caso (a), sem resistência do ar (figura 1), não há dissipação de energia (a potência total do sistema corredor+bicicleta é constante e positiva), de modo que não há limite para a velocidade.

No caso (b), com resistência do ar (figura 2), o ciclista chega a um estado estacionário, movendo-se com a velocidade terminal dada pela equação 1. A velocidade final na simulação (após 5 min) foi igual ao resultado analítico dentro de 10^{-13} m/s. A velocidade terminal foi atingida, com tolerância de 10^{-3} m/s, em aproximadamente 78 s. A distância total percorrida após 5 min foi de 11 794 m.

No caso (c), todas as velocidades terminais calculadas numericamente (velocidades finais após 5 min) foram iguais às analíticas dentro de 10^{-6} m/s, a precisão do resultado melhorando com o aumento de A (as últimas três ficaram dentro de 10^{-13} m/s dos resultados analíticos). Observa-se, em concordância com a equação 1, que a velocidade terminal é maior conforme diminui a área frontal do ciclista. Isto é usado por ciclistas profissionais, que procuram permanecer logo atrás dos competidores, diminuindo sua área frontal efetiva que sofre resistência do ar.

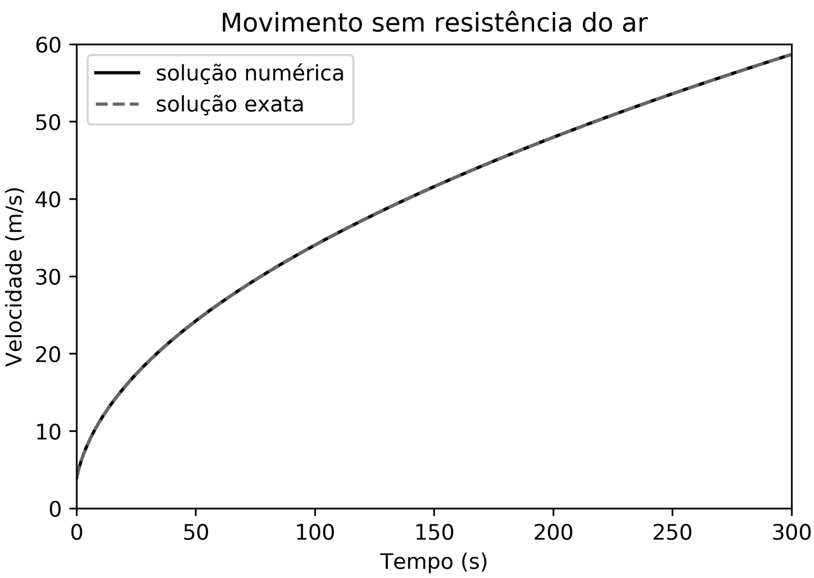

Figura 1: Resultado da parte (a). As curvas analítica e numérica coincidem dentro de 0,4%.

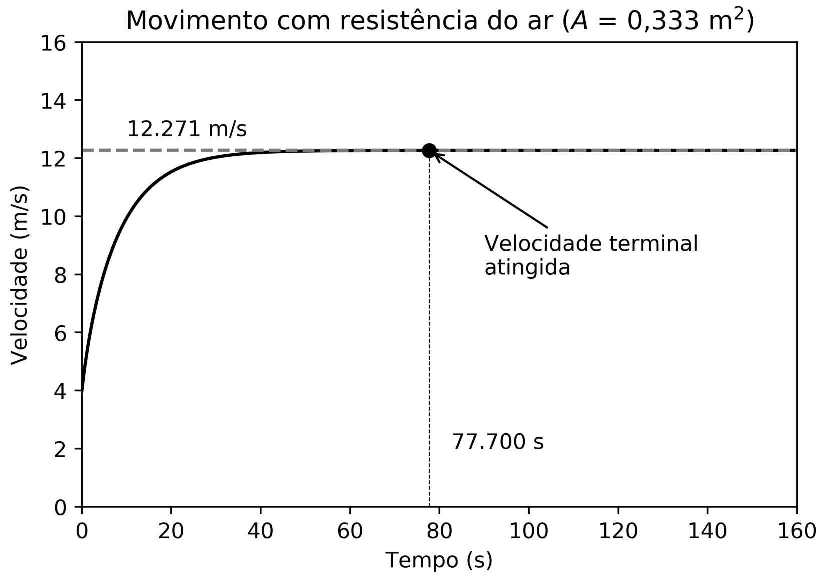

Figura 2: Resultado da parte (b), onde $\rho = 1{,}3\ \mathrm{kg/m^3}$. Considera-se que a velocidade terminal é atingida quando a diferença entre a velocidade atual e a velocidade terminal calculada analiticamente é menor que 10^{-3} m/s.

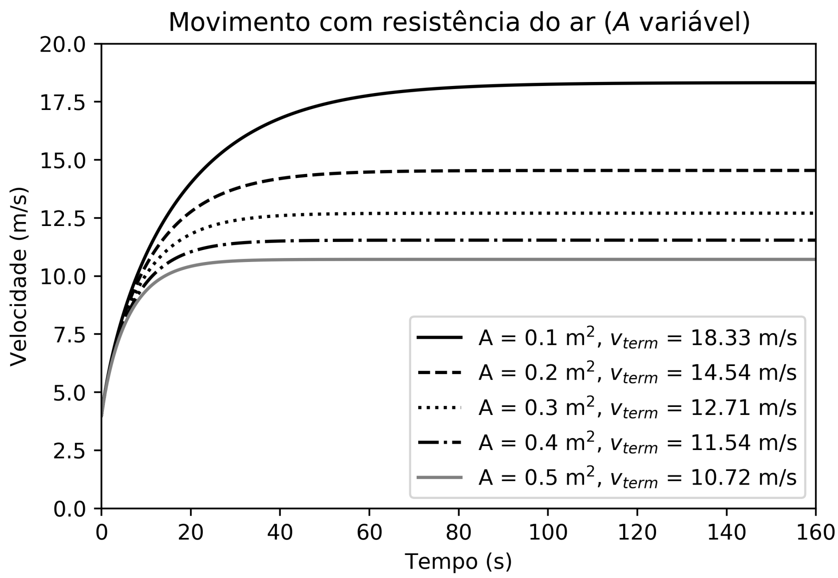

Figura 3: Resultado da parte (c), onde $\rho = 1{,}3\ \mathrm{kg/m^3}$ é fixo e A é variável. Quando menor é A, maior é a velocidade terminal atingida.

2 Lançamento de projéteis

Novamente, um único programa é necessário. O programa lê valores de $\lambda_0 = \gamma_2/m$ (em m^{-1}), b (em K/m), v_0 (em m/s), e θ_0 (em graus). Para a parte (a), fazemos $\lambda_0 = 0$; em (b), temos $\lambda_0 = 4 \times 10^{-5}\ \mathrm{m}^{-1}$ e $b = 0$; em (c) e (d), $\lambda_0 = 4 \times 10^{-5}\ \mathrm{m}^{-1}$ e $b = 6{,}5 \times 10^{-3}$ K/m.

Se for dado um argumento de linha de comando qualquer, o programa retorna uma tabela da trajetória (tempo, velocidade em x, velocidade em y, posição em x, posição em y). Caso contrário, retorna apenas o ângulo (em graus) e o alcance. Em qualquer caso, o programa espera uma nova entrada $(\lambda_0, b, v_0, \theta_0)$, encerrando apenas quando recebe um sinal EOF (*end of file*).

Para cada entrada, iteramos em passos `dt` $= 0{,}01$ s. O laço para assim que a posição em y se torna negativa (o projétil atinge o chão) ou o número de iterações `i` supera o limite `imax`. Em cada iteração, calculamos $v = \sqrt{v_x^2 + v_y^2}$ e $\lambda = \lambda_0(1 - by/T_0)^\alpha$ e integramos pelo método de Euler, usando as equações (2) a (5).

```fortran
program projetil
implicit none

real*8 xi,yi,vxi,vyi,vi,v0
real*8 g, theta, dt
real*8 lambda, lambda0, b, T0, alpha
integer i,imax
logical traj     ! escrever trajetoria?

parameter (g=9.8)
parameter (alpha=2.5)
parameter (T0=300)
parameter (dt=0.01)
parameter (imax = 10000000)

traj = iargc()>0

do
    read(*,*,end=10) lambda0,b,v0,theta

    xi = 0
    yi = 0
    vxi = v0*cos(theta*(2.d0*atan(1.d0)/90.d0))
    vyi = v0*sin(theta*(2.d0*atan(1.d0)/90.d0))

    i=0
    do while(yi>=0 .and. i<=imax)
        if (traj) then
            write(*,*) i*dt, vxi, vyi, xi, yi
        end if

        vi = dsqrt(vxi*vxi + vyi*vyi)
        lambda = lambda0*(1.d0-b*yi/T0)**alpha
        xi = xi + vxi*dt
        yi = yi + vyi*dt
        vxi = vxi - lambda*vi*vxi*dt
        vyi = vyi - g*dt - lambda*vi*vyi*dt

        i = i+1
    end do

    if (.not. traj) then
        write(*,*) theta, xi  ! angulo, alcance
```

```
        end if
    end do

    end program projetil
```

Note, na implementação acima, que quando `traj` é falso, o valor do alcance dado na saída é x_{n+1}, onde n é o número de iterações, enquanto no caso em que `traj` é verdadeiro, o último valor de x na saída (interpretado como alcance do projétil) é x_n. Isto é proposital. Como estamos avançando no tempo em passos discretos, não esperamos que o programa pare com y exatamente em 0, mas um pouco abaixo. Estes valores, x_n e x_{n+1}, são as coordenadas x do projétil antes e depois de y mudar de sinal. Interpretamos a média $(x_n + x_{n+1})/2$ como o alcance do projétil no lançamento, e a semidiferença $(x_{n+1} - x_n)/2$ como o erro associado à medida. Isto não foi automatizado apenas por simplicidade. Estimativa semelhante poderia ser feita para o tempo de voo.

Os resultados da parte (a) são exibidos nas figuras 4 e 5. O ângulo de lançamento que corresponde ao alcance máximo, como pode-se mostrar analiticamente, é de 45°. Na simulação, o ângulo de alcance máximo encontrado foi de 44,95°, com alcance de 50 008 ± 3 m. Explica-se a discrepância por erros de arredondamento (foram mais de 10 000 iterações). O tempo de voo correspondente foi de 100,93 s.

Os resultados da parte (b) estão nas figuras 6 e 7. O ângulo de lançamento que dá alcance máximo, agora, é de 38,74° (figura 7). O tempo de voo correspondente é de 69,62 s, e o alcance é de 22 071 ± 1 m. Todos estes valores são menores que no caso (a), o que era esperado: trajetórias com ângulos de lançamento maiores demoram mais, dissipando mais energia no ar, o que explica o ângulo ótimo menor que 45°. A diminuição no alcance e no tempo de voo têm razões evidentes.

Na parte (c) (figuras 8 e 9), o ângulo de alcance máximo aumenta novamente: 43,62°, com alcance de 24 521 ± 1 m e tempo de voo 78,03 s. Isto acontece porque a densidade do ar é menor em altitudes mais elevadas, o que quer dizer que o projétil sofre menos resistência do ar se subir mais.

Finalmente, na parte (d), fixamos θ em 43,62° e variamos v_0 em 1%, isto é, usamos $v_0 = 693$ m/s e $v_0 = 707$ m/s. Os alcances foram, respectivamente, 24 214 ± 1 m e 24 827 ± 1 m, indicando que 1% de variação na velocidade inicial de lançamento leva a uma variação de aproximadamente 300 m no alcance do projétil, ou 1,25%. Sequer levamos em conta o vento e outras complicações. Este problema de acurácia é uma razão porque os *canhões de Paris*, usados pelos alemães em 1918 para bombardear a capital francesa de um ponto a 120 km de distância, só serviam para acertar alvos do tamanho de cidades.

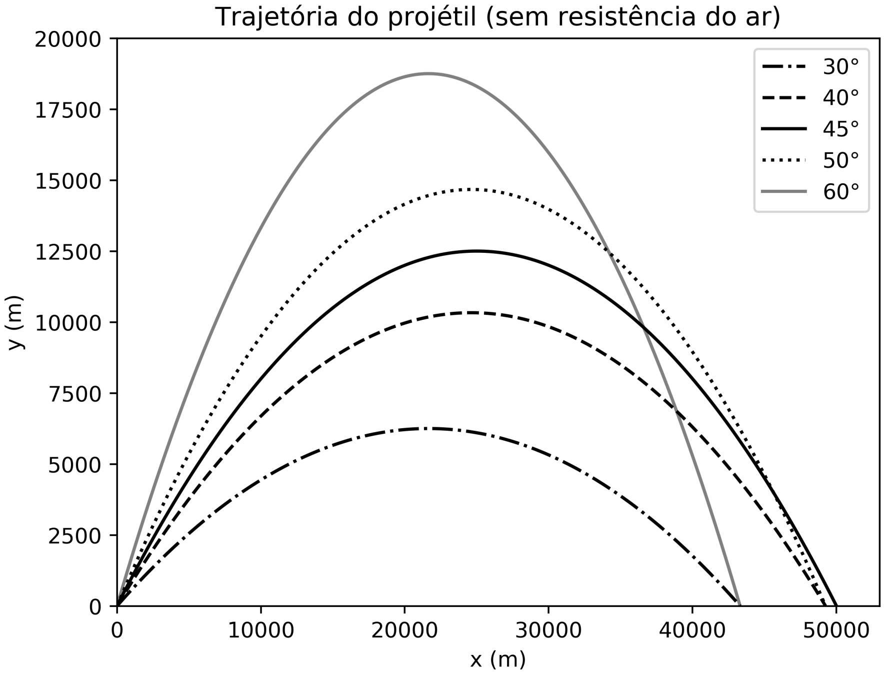

Figura 4: Trajetórias para diferentes ângulos de lançamento na parte (a). Note que o ângulo de lançamento com maior alcance é 45° e que o ângulos “equidistantes” de 45° dão mesmo alcance.

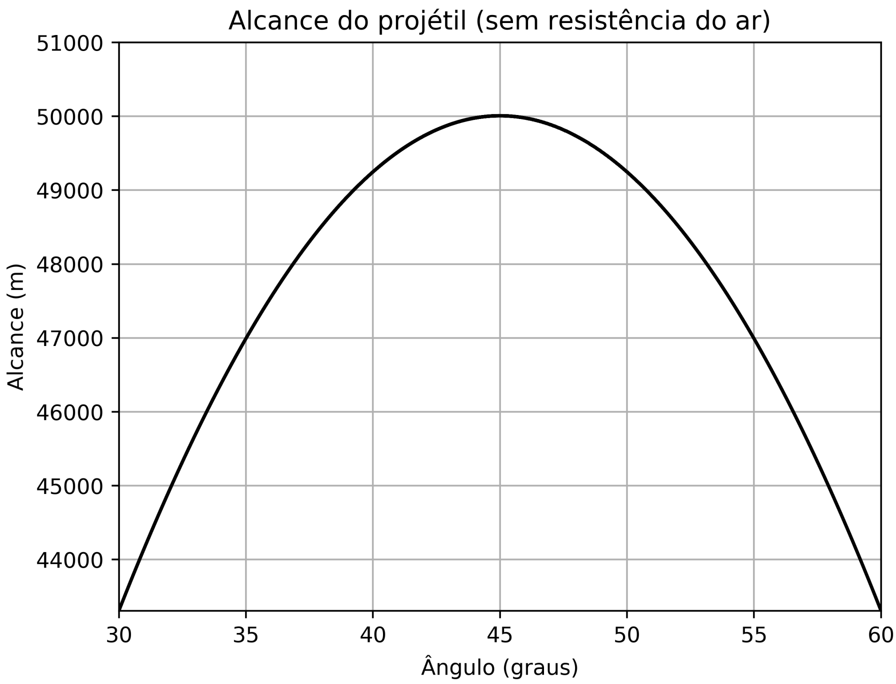

Figura 5: Alcance do projétil como função do ângulo de lançamento na parte (a).

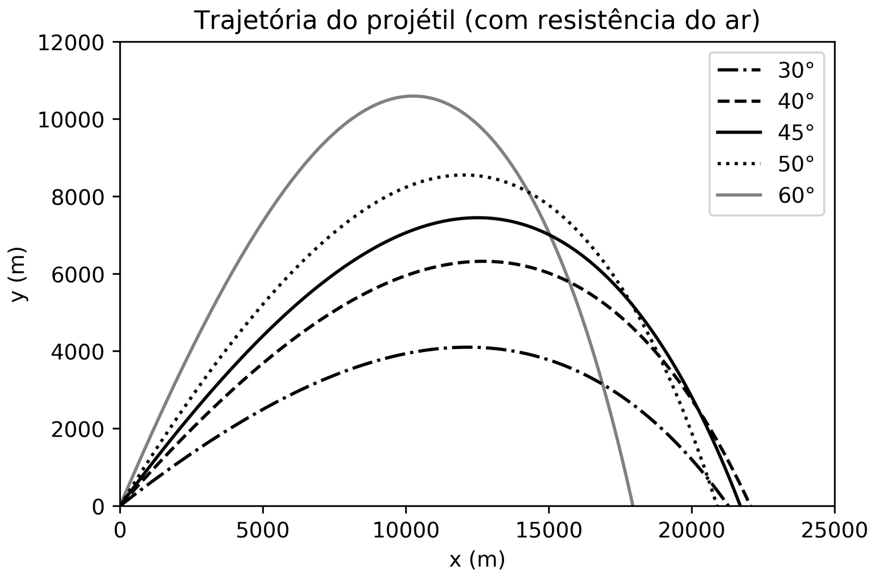

Figura 6: Trajetórias para diferentes ângulos de lançamento na parte (b). Note a assimetria das trajetórias e o alcance menor que na parte (a). 45° não é mais o ângulo de lançamento com maior alcance.

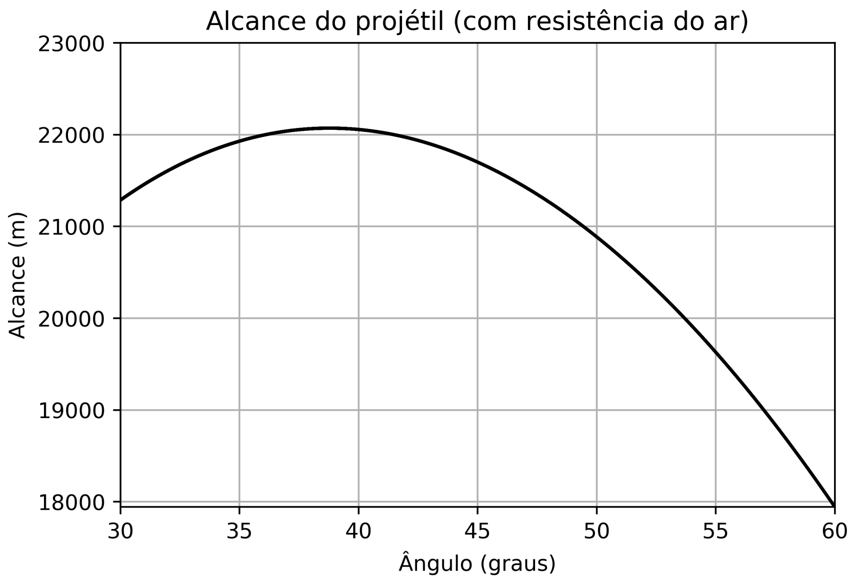

Figura 7: Alcance do projétil como função do ângulo de lançamento na parte (b). O ângulo de alcance máximo agora vale 38,74°.

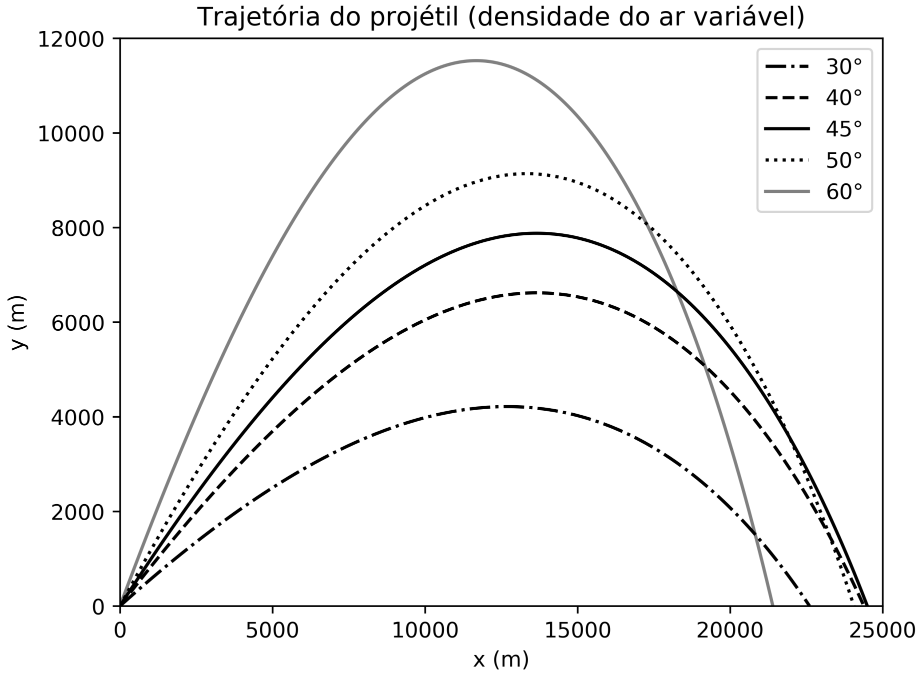

Figura 8: Trajetórias para diferentes ângulos de lançamento na parte (c).

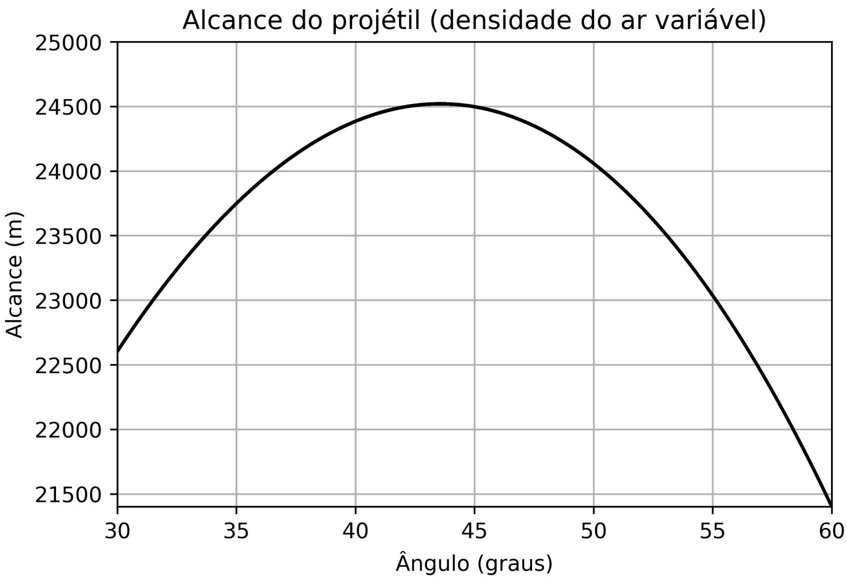

Figura 9: Alcance do projétil como função do ângulo de lançamento na parte (c).

3 Pêndulo simples

Este código, que implementa o método de Euler para a equação de movimento do pêndulo, serve para o item (a):

```
program pendulo_a
implicit none

real*8 omega_i, theta_i
real*8 omega_n, theta_n
real*8 E, g, L, m, theta_0, dt, tmax
integer i

parameter (g=10)
parameter (L=1)
parameter (m=1)
parameter (theta_0=atan(1d0/sqrt(3d0)))
parameter (tmax=20)
parameter (dt=5d-3)

omega_i = 0
theta_i = theta_0
i=0
do while (i*dt <= tmax)
    E = m*(L**2)*(omega_i**2)/2d0 + &
        m*g*L*(1-cos(theta_i))
    write(*,*) i*dt, theta_i, omega_i, E
    omega_n = omega_i - g*sin(theta_i)*dt/L
    theta_n = theta_i + omega_i*dt
    omega_i = omega_n
    theta_i = theta_n
    i = i+1
end do

end program pendulo_a
```

O item (b) se obtém com um programa quase idêntico, mudando apenas a linha que calcula `theta_n` para

```
    theta_n = theta_i + omega_n*dt
```

Veja que o "índice" `n` foi escolhido para representar $i+1$, de maneira que mantemos os valores da iteração anterior armazenados nas variáveis com índice `i`. A cada iteração, o programa escreve na saída o tempo, o ângulo, a velocidade angular e a energia mecânica total.

Os resultados do item (a) são mostrados nas figuras 10 e 11, e os do item (b) nas figuras 12 e 13. A energia mecânica total aumenta consideravelmente em (a), graças à instabilidade inerente do método de Euler. Com o ganho de energia, também aumenta um pouco o tempo de oscilação: veja que aos 20 s o pêndulo (b) está quase um quarto de oscilação à frente do pêndulo (a). No método de Euler-Cromer, usado em (b), a energia varia periodicamente, sendo em média constante a cada oscilação.

Para o item (c), é necessário um novo programa. Este lê, repetidas vezes até chegar num EOF, um intervalo de tempo Δt (`dt`) e um ângulo θ_0 inicial (em graus). Para cada entrada, calcula o período de oscilação por Euler-Cromer (integrando o movimento) e pela integral elíptica na equação 6, usando a regra de Simpson. O cálculo do período por Euler-Cromer é feito definindo-se um inteiro `p`, que conta as trocas de sinal da velocidade angular ω. Quando `p` atinge 10, isto é, ocorrem 5 oscilações, a iteração para e o tempo total que se passou é dividido por 5 para obter o período, armazenado em `periodo1`. O outro período é apenas a integral da função `func` de 0 até $\pi/2$.

O valor ideal de `dt` é menor que 5×10^{-8} s (o que, com $\theta_0 = 90°$, concorda com o valor da integral até a 8ª casa decimal) e maior que 5×10^{-9} s (que não deu nenhum dígito correto). Computar esse último levou quase 10 min num Core i3 (2 bilhões de iterações!), e por isso não procurei melhorar a estimativa do `dt` ótimo além disto. O período de oscilação em função do ângulo inicial é exibido na figura 14. Qualitativamente, o comportamento é exatamente o esperado pela equação (7).

```
program pendulo_c
implicit none

real*8 omega_i, theta_i, omega_n, theta_n
real*8 g, L, m, theta_0, dt, tmax, t_u
real*8 periodo1,a,b,func,du,integral,periodo2
integer i,p,N

parameter (g=10)
parameter (L=1)
parameter (m=1)
parameter (tmax=20)

do
    read(*,*,end=10) dt, theta_0
    theta_0 = theta_0*2d0*atan(1d0)/90d0

    ! Cálculo do período por Euler-Cromer
    omega_i = 0
    theta_i = theta_0
    i=0
    p = -1
    t_u = 0
    do while (p <= 20 .and. i*dt<=tmax)
        omega_n = omega_i - g*sin(theta_i)*dt/L
        theta_n = theta_i + omega_n*dt

        if (omega_i==0 .or. omega_n/omega_i<0) then
            p = p+1
            t_u = i*dt
        end if

        omega_i = omega_n
        theta_i = theta_n
        i = i+1
    end do
    periodo1 = 2*t_u/dfloat(p)

    ! Cálculo do perído pela integral elíptica
    ! usando método de Simpson
    a = 0
    b = 2d0*atan(1d0)
    N = 2**12
    du = (b-a)/dfloat(N)
    integral = (func(a,theta_0)+func(b,theta_0))*du/3d0
    do i=1,N-1
        if (mod(i,2)==0) then
            integral = integral + &
              2d0*func(a+i*du, theta_0)*du/3d0
        else
            integral = integral + &
              4d0*func(a+i*du, theta_0)*du/3d0
        end if
    end do
```

```
    periodo2 = 4d0*sqrt(L/g)*integral

    write(*,*) theta_0*90d0/(2d0*atan(1d0)),periodo1,periodo2
end do
end program pendulo_c

function func(u,theta_0)
    real*8 func, u, theta_0
    func = 1d0/sqrt(1d0 - &
        (sin(theta_0/2d0)**2) * (sin(u))**2)
    return
end function func
```

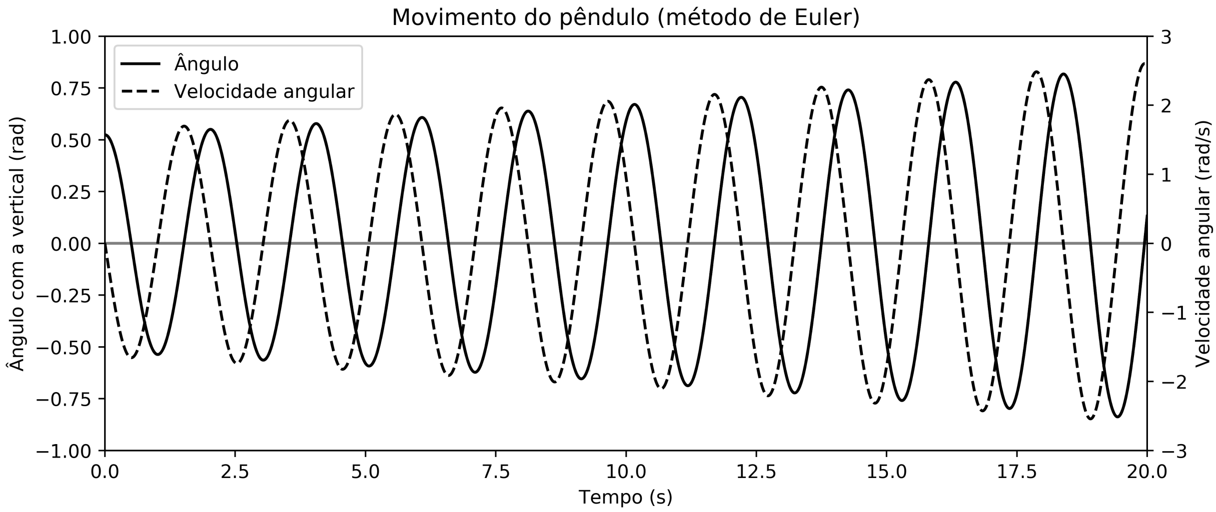

Figura 10: Movimento do pêndulo no item (a).

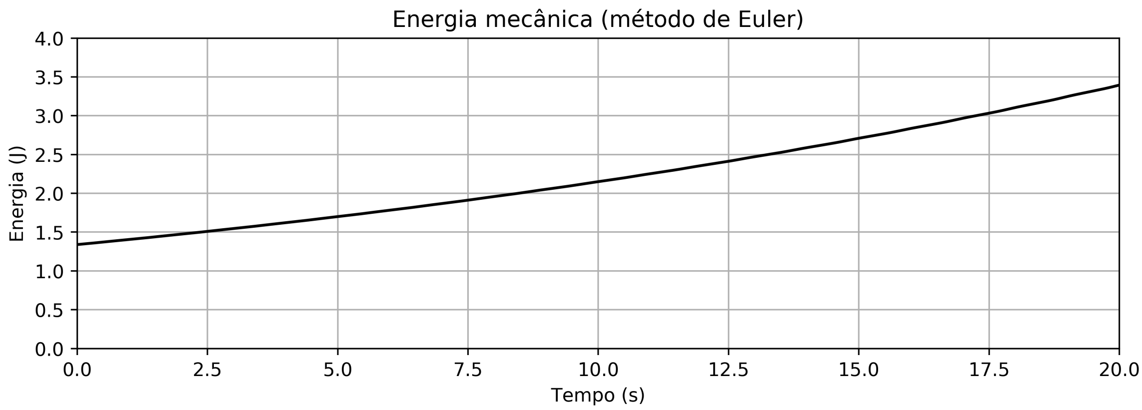

Figura 11: Energia do pêndulo no item (a).

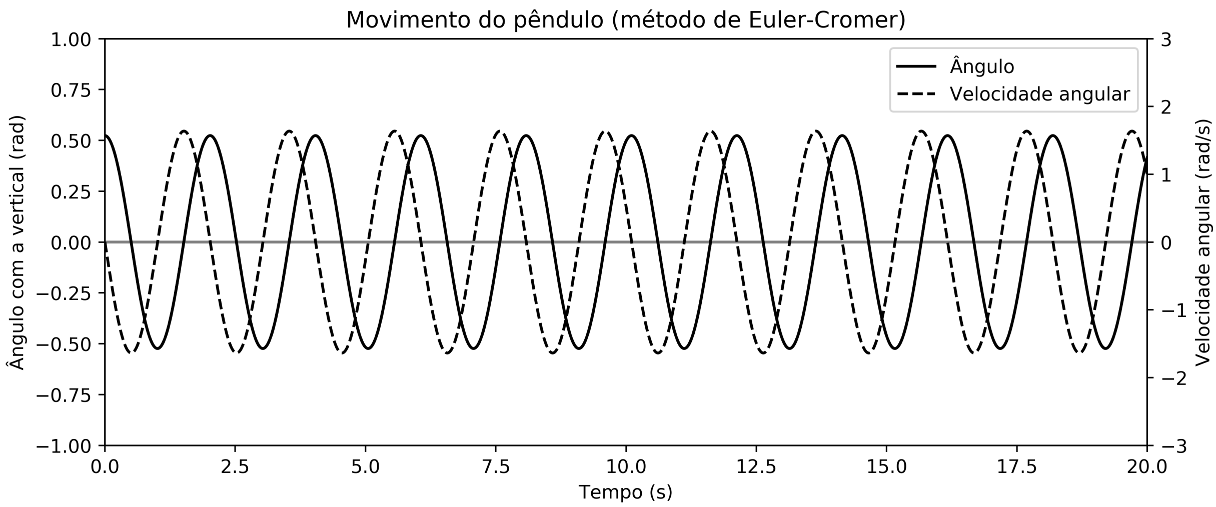

Figura 12: Movimento do pêndulo no item (b).

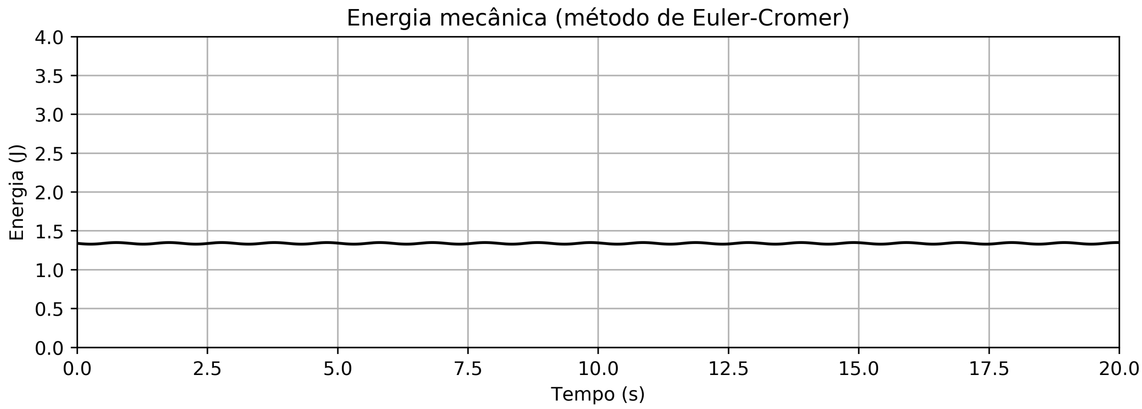

Figura 13: Energia do pêndulo no item (b).

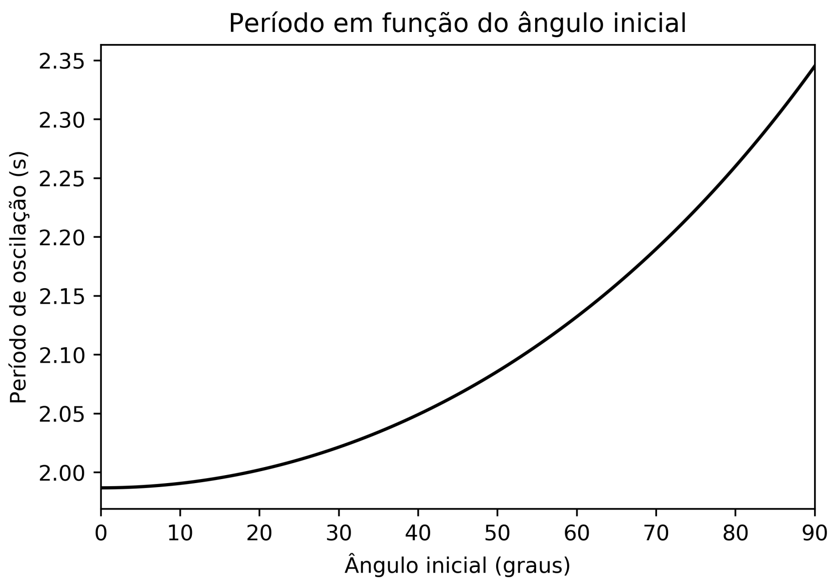

Figura 14: Período de oscilação em função do ângulo inicial do pêndulo simples. Compare com a equação (7).