

Ciência Computacional: Modelagem e Simulação - 2

Roberto M. Cesar Jr.

rmcesar@usp.br



Blucher

Curso de Física Básica – vol. 1 -
5ª ed. - H. Moysés Nussenzveig

Curso de Física Básica

Volume 1 – Mecânica

5^a Edição, Revista e Atualizada

Capítulo I
Introdução

Blucher

Curso de Física Básica – vol. 1 -
5^a ed. - H. Moysés Nussenzveig

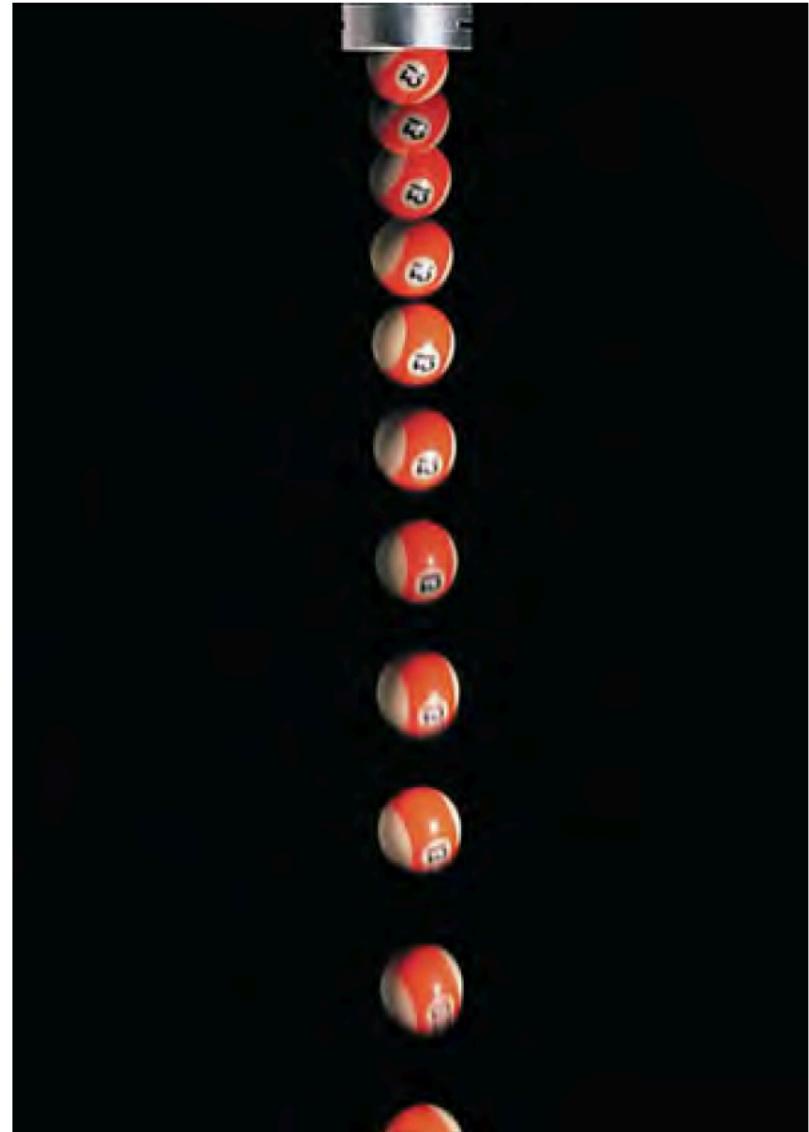
Capítulo 2

MOVIMENTO UNIDIMENSIONAL

2.1 – Velocidade média

A análise do movimento é um problema fundamental em física, e a forma mais simples de abordá-la é considerar primeiro os conceitos que intervêm na descrição do movimento (*cinemática*), sem considerar ainda o problema de como determinar o movimento que se produz numa dada situação física (*dinâmica*). No presente capítulo, para simplificar ainda mais a discussão, vamos-nos limitar ao movimento em uma só dimensão — por exemplo, o movimento de um automóvel em linha reta ao longo de uma estrada. Como muitos aspectos da cinemática são discutidos no curso secundário, vamos restringir o tratamento a apenas alguns tópicos controvérsios.

2.22 Multiflash photo of a freely falling ball.



$$g = 9.8 \text{ m/s}^2 = 980 \text{ cm/s}^2 = 32 \text{ ft/s}^2$$

(approximate value near
the earth's surface)

Exemplo: Na experiência de queda livre da bolinha (Fig. 2.2), o gráfico $x \times t$ tem a forma de uma parábola (Fig. 2.6), $x = \alpha t^2$, onde, para x em m e t em s, o valor de α seria $\approx 5 \text{ m/s}^2$; tomemos

$$x(t) = 5t^2 \quad (2.2.1)$$

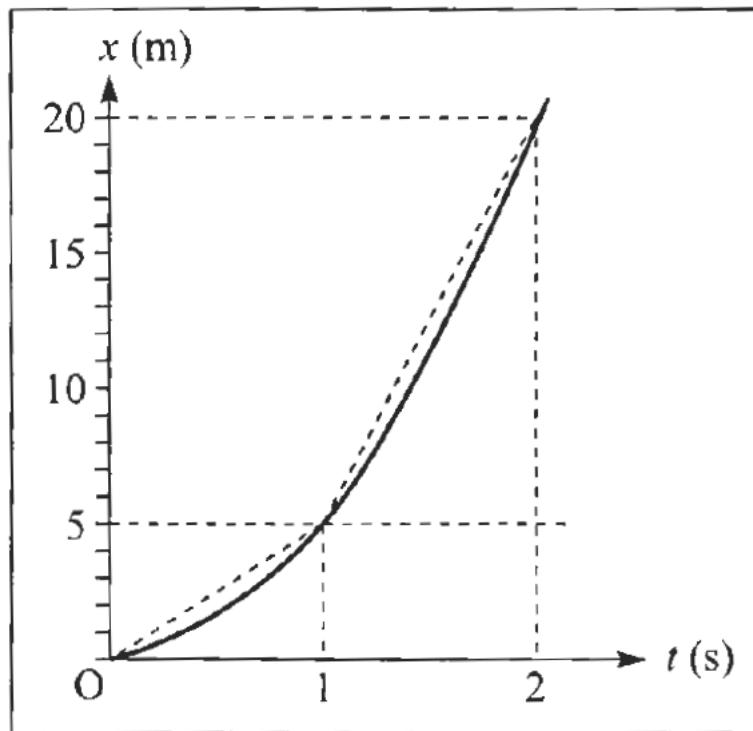


Figura 2.6 Velocidade média na queda livre.

Qual é a velocidade instantânea para $t = 1$ s? Com centro no instante $t = 1$ s, calculemos a velocidade média (2.1.5) a partir de instantes anteriores e para instantes posteriores, tomando $\Delta t = 1$ s, 0,1 s, 0,01 s.

$$\left. \begin{aligned} \bar{v}_{0 \rightarrow 1} &= \frac{x(1) - x(0)}{1 - 0} = \frac{5 - 0}{1 - 0} = 5 \text{ m/s} \\ \bar{v}_{1 \rightarrow 2} &= \frac{x(2) - x(1)}{2 - 1} = \frac{20 - 5}{2 - 1} = 15 \text{ m/s} \end{aligned} \right\} \Delta t = 1 \text{ s}$$

$$\left. \begin{aligned} \bar{v}_{0,9 \rightarrow 1} &= \frac{x(1) - x(0,9)}{1 - 0,9} = \frac{5,00 - 4,05}{1 - 0,9} = 9,5 \text{ m/s} \\ \bar{v}_{1 \rightarrow 1,1} &= \frac{x(1,1) - x(1)}{1,1 - 1} = \frac{6,05 - 5,00}{1,1 - 1} = 10,5 \text{ m/s} \end{aligned} \right\} \Delta t = 0,1 \text{ s}$$

$$\left. \begin{aligned} \bar{v}_{0,99 \rightarrow 1} &= \frac{x(1) - x(0,99)}{1 - 0,99} = \frac{5,0000 - 4,9005}{1,00 - 0,99} = 9,95 \text{ m/s} \\ \bar{v}_{1 \rightarrow 1,01} &= \frac{x(1,01) - x(1)}{1,01 - 1,00} = \frac{5,1005 - 5,0000}{1,01 - 1,00} = 10,05 \text{ m/s} \end{aligned} \right\} \Delta t = 0,01 \text{ s}$$

Para uma função $x(t)$, o limite

$$\lim_{\Delta t \rightarrow 0} \left[\frac{x(t_0 + \Delta t) - x(t_0)}{\Delta t} \right] = \lim_{\Delta t \rightarrow 0} \left(\frac{\Delta x}{\Delta t} \right)_{t=t_0} = \left(\frac{dx}{dt} \right)_{t=t_0}$$

$$\left(\frac{dx}{dt} \right)_{t=1} = 10 \quad (\text{em m / s})$$

A velocidade instantânea $v(t)$ num instante t qualquer, num movimento descrito por $x = x(t)$, é dada por

$$v(t) = \frac{dx}{dt} \quad (2.2.5)$$

2.3 – O problema inverso

Vimos como, conhecendo a "lei horária" de um movimento, ou seja, a função $x = x(t)$, é possível calcular a velocidade instantânea $v(t)$ no decurso do movimento: basta tomar dx/dt . Assim, por exemplo, para a lei horária (2.2.3), a velocidade é dada pela (2.2.4).

Freqüentemente temos de resolver o problema inverso: conhecendo a velocidade instantânea $v(t)$ entre um dado instante inicial t_1 , e um instante final t_2 , calcular o espaço percorrido entre estes dois instantes, ou seja, $x(t_2) - x(t_1)$. Poderíamos pensar num filme do painel de instrumentos de um automóvel que mostrasse simultaneamente o velocímetro e um

relógio, permitindo traçar o gráfico de $v \times t$ entre t_1 e t_2 (tomamos sempre $t_2 > t_1$).

Se o movimento for uniforme, como na (2.1.4), velocidade instantânea e velocidade média se confundem, $v = \bar{v} = \text{constante}$, e o gráfico é uma reta paralela ao eixo das abscissas (Fig. 2.9). Pela definição de velocidade média, o espaço percorrido entre t_1 e t_2 é:

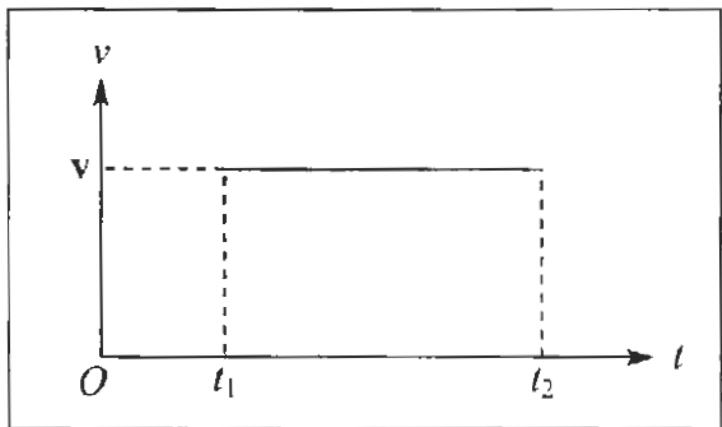


Figura 2.9 Espaço percorrido como área.

$$\Delta t_{t_1 \rightarrow t_2} = x(t_2) - x(t_1) = \bar{v}_{t_1 \rightarrow t_2} \Delta t = \bar{v}_{t_1 \rightarrow t_2} (t_2 - t_1) = \bar{v}(t_2 - t_1) = v(t_2 - t_1) \quad (2.3.1)$$



Delta x, o livro está errado.

$$\Delta x_{t_1 \rightarrow t'_1} = x(t'_1) - x(t_1) = \bar{v}_{t_1 \rightarrow t'_1} \Delta t_1 \approx v(t_1) \Delta t_1$$

$$\Delta x_{t'_1 \rightarrow t'_2} = x(t'_2) - x(t'_1) = \bar{v}_{t'_1 \rightarrow t'_2} \Delta t_2 \approx v(t'_1) \Delta t_2$$

$$\Delta x_{t'_2 \rightarrow t'_3} = x(t'_3) - x(t'_2) = \bar{v}_{t'_2 \rightarrow t'_3} \Delta t_3 \approx v(t'_2) \Delta t_3$$

Somando membro a membro estas 3 relações, obtemos o deslocamento total entre t_1 e t'_3 :

$$x(t'_3) - x(t_1) \approx v(t_1) \Delta t_1 + v(t'_1) \Delta t_2 + v(t'_2) \Delta t_3$$

e é claro que, se prosseguirmos até t_2 , obteremos a soma das contribuições de todos os subintervalos em que $[t_1, t_2]$ foi dividido:

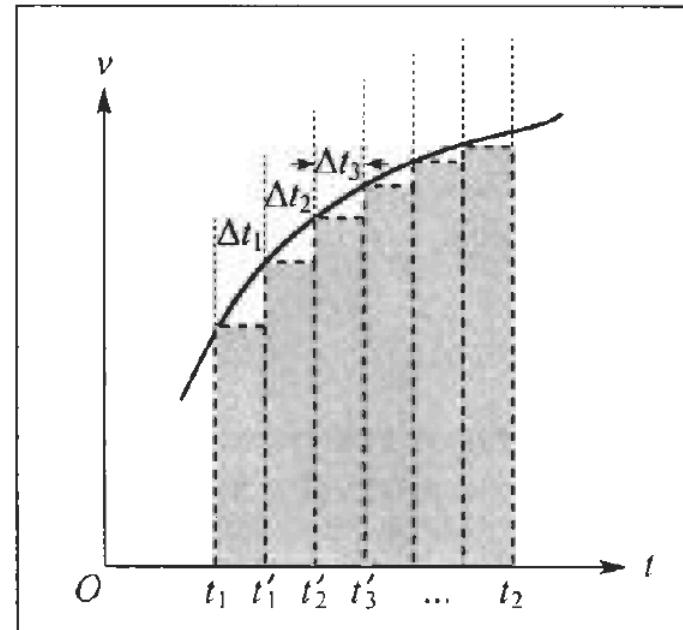


Figura 2.11 Divisão em subintervalos.

$$x(t_2) - x(t_1) \approx \sum_i v(t'_i) \Delta t'_i \quad (2.3.2)$$

A soma (2.3.2) se aproxima tanto mais do resultado exato quanto menores forem as subdivisões $\Delta t'_i$. Logo, no limite em que os $\Delta t'_i$ tendem a zero, devemos obter:

$$x(t_2) - x(t_1) = \lim_{\Delta t'_i \rightarrow 0} \sum_i v(t'_i) \Delta t'_i = \begin{array}{l} \text{Área entre a curva } v \times t \\ \text{e o eixo } Ot, \text{ de } t_1 \text{ a } t_2 \end{array} \quad (2.3.3)$$

O limite (2.3.3) é chamado de *integral definida de $v(t)$ entre os extremos t_1 e t_2* , é representado pela notação

$$\lim_{\Delta t'_i \rightarrow 0} \sum_i v(t'_i) \Delta t'_i = \int_{t_1}^{t_2} v(t) dt \quad (2.3.4)$$

Como aplicação, consideremos um movimento cuja velocidade $v(t)$ é dada pela (2.2.4):

$$v(t) = 2at + b \quad (2.3.5)$$

A área a calcular neste caso é o trapézio sombreado na Fig. 2.12.

$$\frac{dx}{dt} = 2at + b \quad (2.2.4)$$

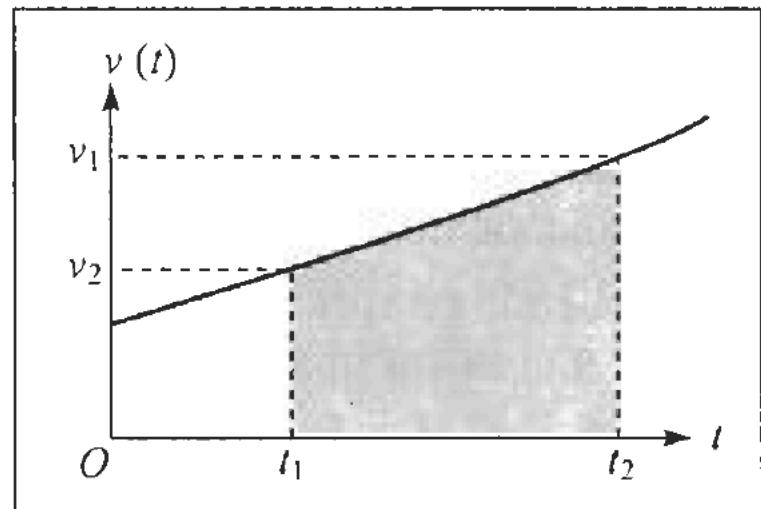


Figura 2.12 Exemplo de integração.

$$\text{Sejam} \begin{cases} v(t_1) = v_1 = 2at_1 + b \\ v(t_2) = v_2 = 2at_2 + b \end{cases}$$

Temos então, pela (2.3.3), $x(t_2) - x(t_1) = \text{Área do trapézio} = \text{Semi-soma das bases} \times \text{Altura} = \frac{1}{2}(v_1 + v_2)(t_2 - t_1)$ o que, comparando com a (2.1.5), implica que, neste movimento,

$$\bar{v}_{t_1 \rightarrow t_2} = \frac{1}{2}[v(t_1) + v(t_2)] \quad (2.3.6)$$

ou seja, que a velocidade média num intervalo é a média aritmética das velocidades nos extremos do intervalo. Substituindo na (2.3.6) os valores de $v(t_1)$ e $v(t_2)$, vem

$$\bar{v}_{t_1 \rightarrow t_2} = a(t_2 + t_1) + b$$

o que dá

$$x(t_2) - x(t_1) = a(t_2^2 - t_1^2) + b(t_2 - t_1) \quad (2.3.7)$$

que coincide com o resultado obtido a partir da lei horária (2.2.3) (e dá o valor da integral definida (2.3.4) quando $v(t)$ é dado pela (2.3.5)).

A (2.3.7) pode ser aplicada, em particular, tomando para t_1 o instante inicial $t_1 = 0$, e para t_2 um instante genérico t . Chamando $x(0) = c$ (valor inicial de x), a (2.3.7) dá então:

$$x(t) = x(0) + at^2 + bt = at^2 + bt + c \quad (2.3.8)$$

ou seja, este processo de "integração" nos permitiu recuperar a lei horária (2.2.3) a partir da expressão (2.2.4) da velocidade e do valor inicial de x .

Matematicamente, a (2.2.4) se chama uma *equação diferencial* para a função incógnita $x(t)$ (porque a derivada da função incógnita aparece na equação). Passamos da (2.2.4) à (2.3.8) integrando a equação diferencial com a *condição inicial* $x(0) = c$.

$$\frac{dx}{dt} = 2 at + b \quad (2.2.4)$$

Aceleração

A aceleração média pode geralmente ser variável durante o movimento, e considerações análogas às da Seção 2.2 levam-nos a definir a *aceleração instantânea* $a(t)$ num instante t por (cf. (2.2.2) e (2.2.5))

$$a(t) = \lim_{\Delta t \rightarrow 0} \left[\frac{v(t + \Delta t) - v(t)}{\Delta t} \right] = \lim_{\Delta t \rightarrow 0} \left(\frac{\Delta v}{\Delta t} \right) = \frac{dv}{dt} \quad (2.4.2)$$

ou seja, a *aceleração instantânea* é a derivada em relação ao tempo da *velocidade instantânea*.

Substituindo $v(t)$ na (2.4.2) pela (2.2.5), obtemos

$$a(t) = \frac{d}{dt} \left(\frac{dx}{dt} \right) = \frac{d^2x}{dt^2} \quad (2.4.3)$$

onde introduzimos a definição de *derivada segunda* de x em relação a t , indicada pela notação d^2x/dt^2 .

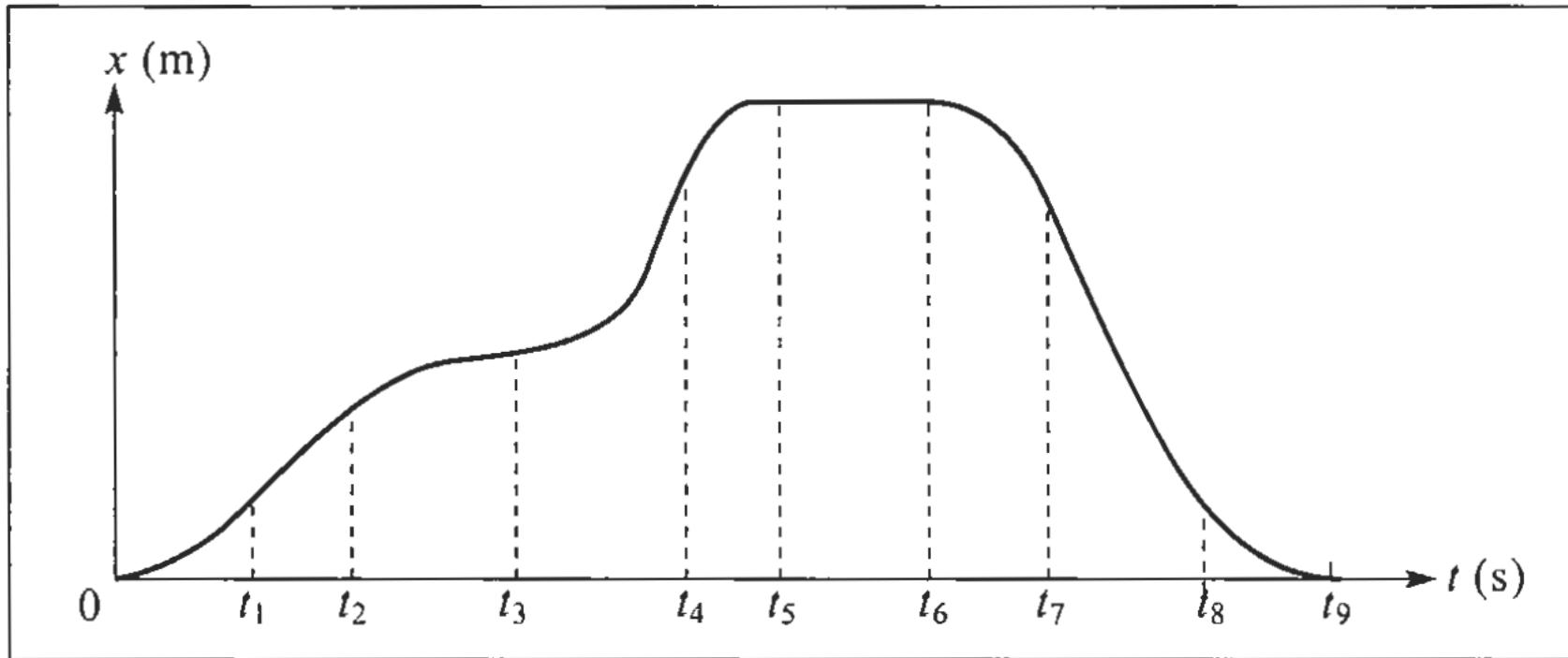


Figura 2.13 Posição em função do tempo.

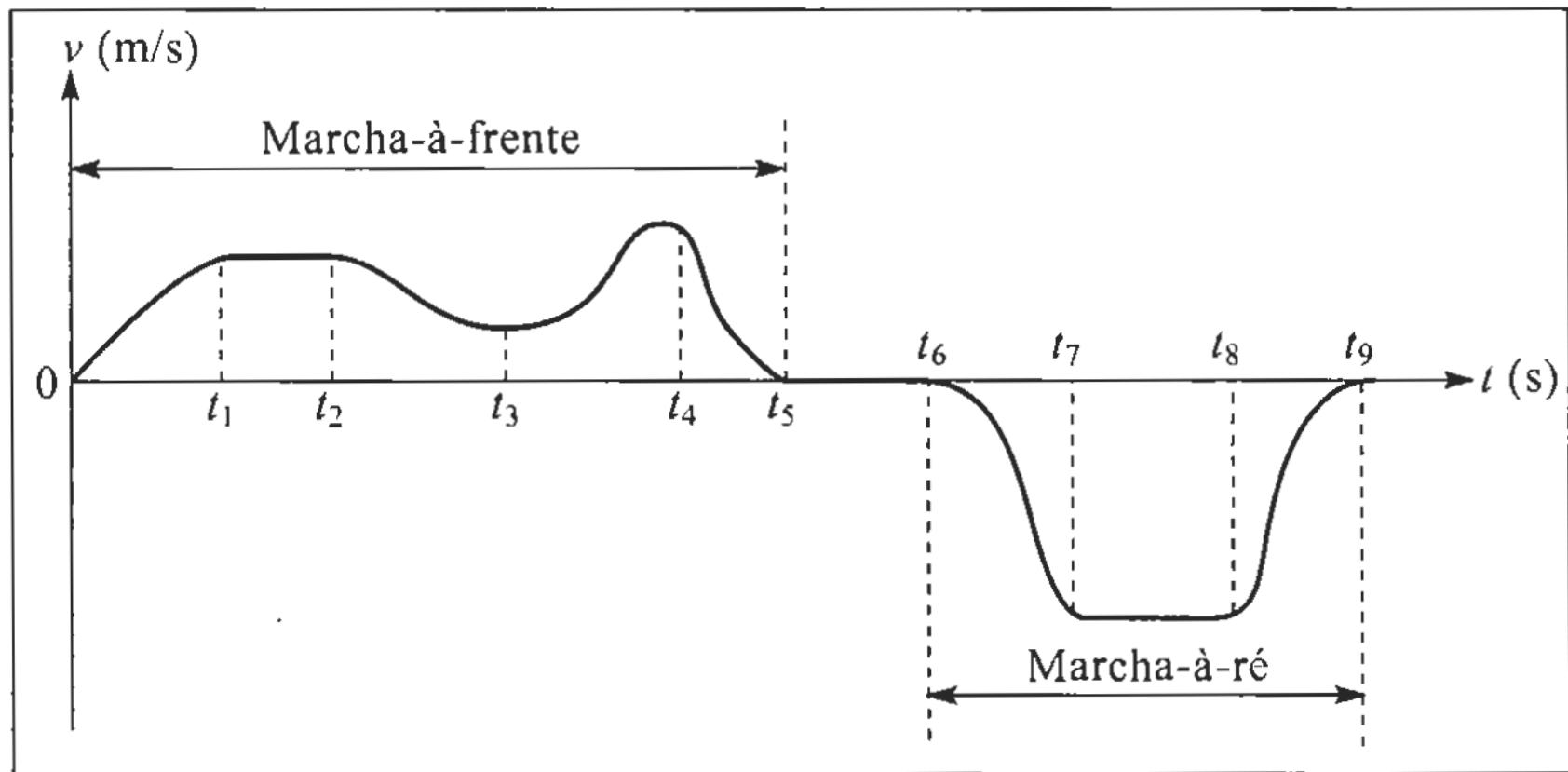


Figura 2.14 Velocidade em função do tempo.

O gráfico da aceleração instantânea se obtém de forma análoga do gráfico $v \times t$:

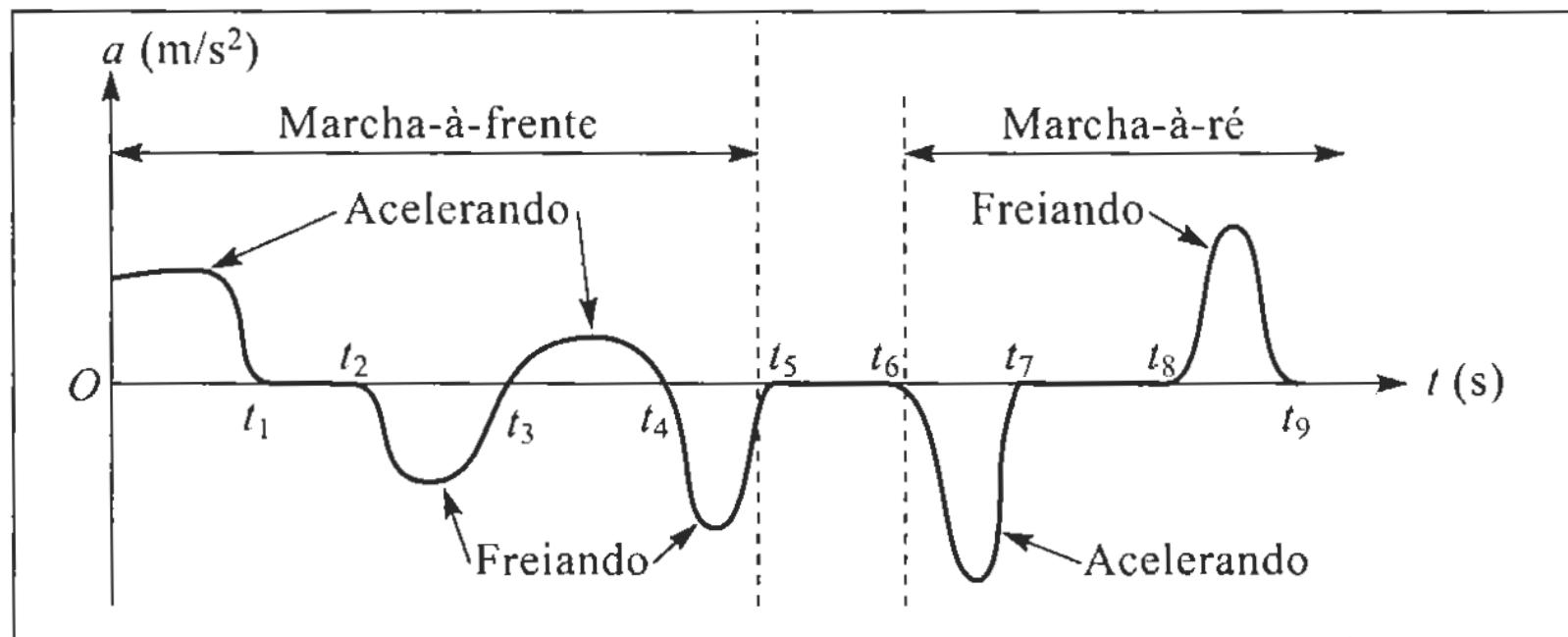


Figura 2.15 Aceleração em função do tempo.

2.5 – Movimento retilíneo uniformemente acelerado

Um movimento retilíneo chama-se *uniformemente acelerado* quando a aceleração instantânea é constante (independente do tempo):

$$\boxed{\frac{dv}{dt} = \frac{d^2x}{dt^2} = a = \text{constante}} \quad (2.5.1)$$

Podemos usar as técnicas de solução do "problema inverso" (Seção 2.3) para determinar a lei horária de um movimento uniformemente acelerado.

Para isto, consideremos o movimento durante um intervalo de tempo $[t_0, t]$, onde t_0 é o "instante inicial" (freqüentemente se toma $t_0 = 0$).

A (2.4.4) dá:

$$v(t) - v(t_0) = \int_{t_0}^t a \, dt = a(t - t_0) \quad (2.5.2)$$

que é a área do retângulo hachurado na Fig. 2.16 (compare com a (2.3.1)).

O valor

$$v(t_0) = v_0 \quad (2.5.3)$$

da velocidade no instante inicial chama-se *velocidade inicial*. A (2.5.2) dá então

$$v(t) = v_0 + a(t - t_0) \quad (2.5.4)$$

mostrando que a velocidade é uma função linear do tempo no movimento uniformemente acelerado. Este

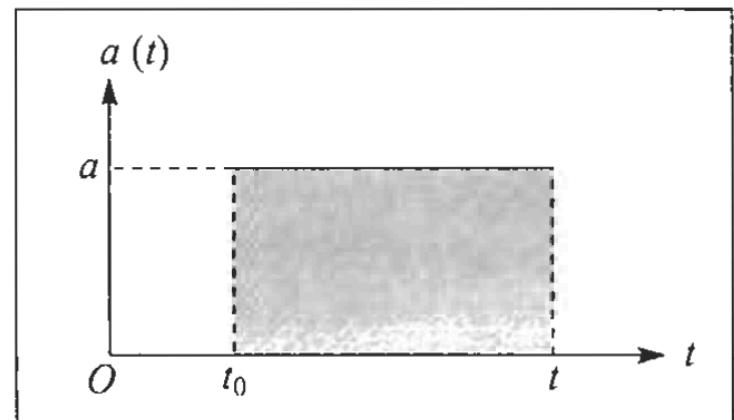


Figura 2.16 Integração da aceleração.

Poderíamos obter a lei horária simplesmente adaptando a (2.3.7) à notação da (2.5.4) (em particular, 2a na (2.3.5) corresponde a a na (2.5.4)), mas é instrutivo recalcular o resultado de forma um pouco diferente. Pelas (2.3.3) e (2.3.4),

$$x(t) - x(t_0) = \int_{t_0}^t v(t') dt' \quad (2.5.5)$$

onde chamamos de t' a variável de integração (veja a discussão após a (2.3.4)) para evitar confusão com t , o extremo superior da integral. A área do trapézio, conforme mostra a Fig. 2.17, pode também ser calculada como a soma da área do retângulo sombreado, que é $v_0(t - t_0)$, com a área do triângulo sombreado, que é

$$\frac{1}{2}a(t - t_0) \cdot (t - t_0)$$

ou seja

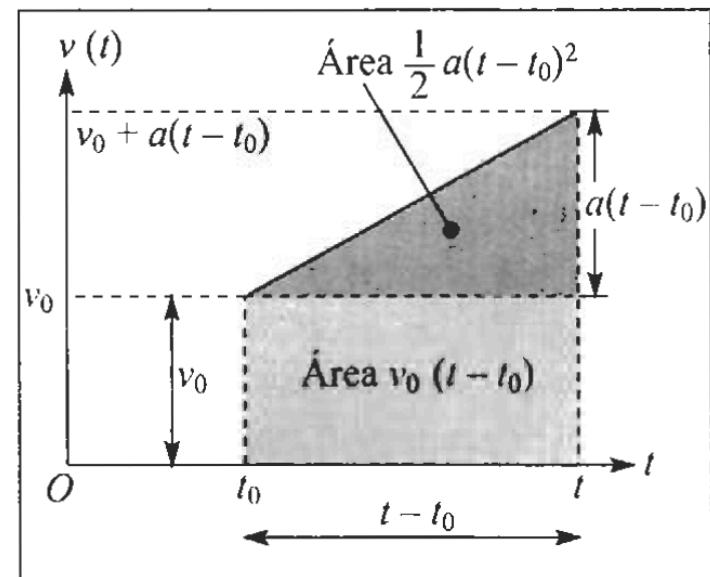


Figura 2.17 Integração da velocidade.

$$x(t) - x(t_0) = v_0(t - t_0) + \frac{1}{2}a(t - t_0)^2 \quad (2.5.6)$$

Analogamente à (2.5.3), definimos

$$x(t_0) = x_0 \quad (2.5.7)$$

como a posição *inicial*. A (2.5.6) dá então finalmente a *lei horária do movimento retilíneo uniformemente acelerado*,

$$x(t) = x_0 + v_0(t - t_0) + \frac{1}{2}a(t - t_0)^2 \quad (2.5.8)$$

em função dos valores iniciais x_0 e v_0 da posição e da velocidade no instante inicial t_0 .

Do ponto de vista matemático, a passagem da (2.5.1) à (2.5.8) corresponde à "integração" da equação diferencial de 2.^a ordem (2.5.1) para a função incógnita $x(t)$ (de 2.^a ordem porque entra a derivada segunda d^2x/dt^2), com as condições iniciais (2.5.3) e (2.5.7). O gráfico $x \times t$ de um movimento uniformemente acelerado é uma parábola.

Freqüentemente interessa também exprimir a velocidade no movimento uniformemente acelerado em função da posição x (em lugar do tempo t). Para obter esta expressão, basta substituir a (2.5.4) na (2.5.8), eliminando $t - t_0$:

$$\begin{aligned} t - t_0 &= \frac{v - v_0}{a} \left\{ x - x_0 = v_0 \left(\frac{v - v_0}{a} \right) + \frac{a}{2} \frac{(v - v_0)^2}{a^2} = \right. \\ &= \frac{v - v_0}{a} \left(v_0 + \frac{v}{2} - \frac{v_0}{2} \right) = \frac{(v - v_0)(v + v_0)}{2a} = \frac{v^2 - v_0^2}{2a} \end{aligned}$$

ou seja,

$$v^2 = v_0^2 + 2a(x - x_0) \quad (2.5.9)$$

que é a expressão procurada.

Algoritmo de Euler

E quando não soubermos a solução analítica?

$$\Delta t_{t_1 \rightarrow t_2} = x(t_2) - x(t_1) = \bar{v}_{t_1 \rightarrow t_2} \Delta t = \bar{v}_{t_1 \rightarrow t_2} (t_2 - t_1) = \bar{v}(t_2 - t_1) = v(t_2 - t_1) \quad (2.3.1)$$



Delta x, o livro está errado.

$$dx/dt = v$$

Aproximação por diferenças finitas: $dx = x(t_2) - x(t_1)$

$$x(t_2) - x(t_1) = v * dt$$

$$x(t_2) = x(t_1) + v * dt$$

Exercício

Implemente em python a solução da equação diferencial $dx/dt = v(t)$ dada por:

$$v(t) = 2at + b \quad (2.3.5)$$

Compare com a solução analítica.



Método de Euler para solução de EDOs

O método de Euler é uma forma de resolver numericamente uma equação diferencial ordinária. Assumem-se conhecidas a derivada de uma função que se quer encontrar ("resolver") e um valor inicial da equação a ser integrada. Por exemplo, no caso do movimento uniformemente acelerado:

$$a = \text{constante}, b = \text{constante}$$

$$v(t) = x'(t) = dx(t)/dt = 2 * a * t + b$$

,

$$x(0) = 0$$

A ideia do método de Euler é substituir a derivada por uma aproximação de Taylor, desprezando-se os termos maiores que segunda ordem. Isto é:

$$x'(t) \sim \frac{x(t + \Delta t) - x(t)}{\Delta t}$$

Se denotarmos os valores de $x(t)$ por x_t , isto é, ao invés da notação de função, usarmos a notação com índices, e assumirmos que os valores de t só podem ser números inteiros (portanto Δt é no mínimo 1, o valor $x(t + \Delta t)$ pode ser escrito como x_1 , para $t = 0$; x_2 , para $t = 1$ e assim por diante.

Desta maneira, o exemplo poderia ser escrito assim (note que já estamos assumindo $\Delta t = 1$):

Capítulo 4

OS PRÍNCIPIOS DA DINÂMICA

4.1 – Forças em equilíbrio

Até aqui discutimos somente a *descrição* de movimentos, sem nos preocuparmos com a determinação do tipo de movimento que terá lugar em dadas circunstâncias físicas. Esta determinação constitui o problema fundamental da dinâmica.

Os princípios básicos da dinâmica foram formulados por Galileu e por Newton. Procuraremos chegar a eles baseando-nos o mais possível em noções intuitivas. Sabemos todos por experiência que o movimento é afetado pela ação do que costumamos chamar de “forças”.

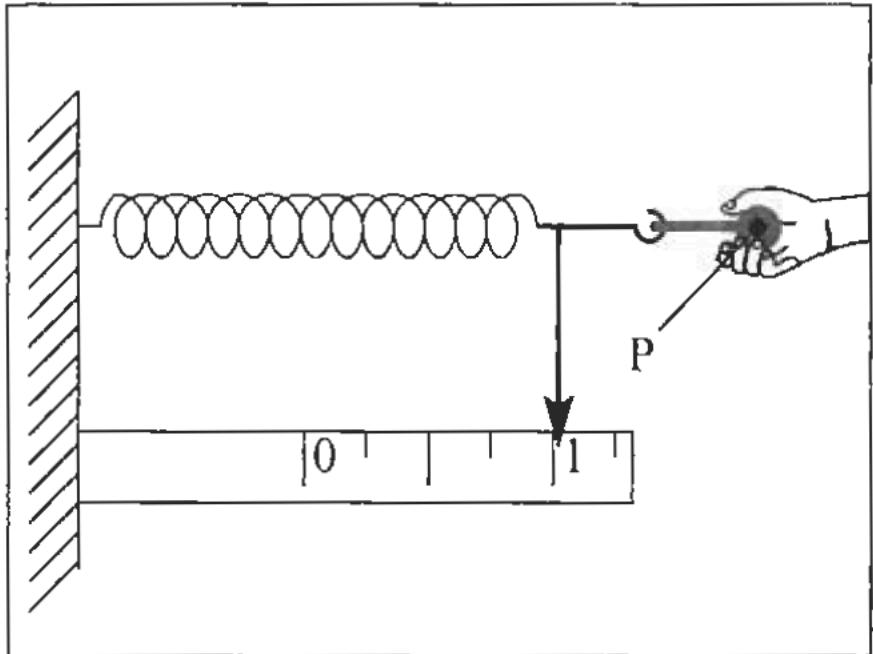


Figura 4.1 Distensão de uma mola.

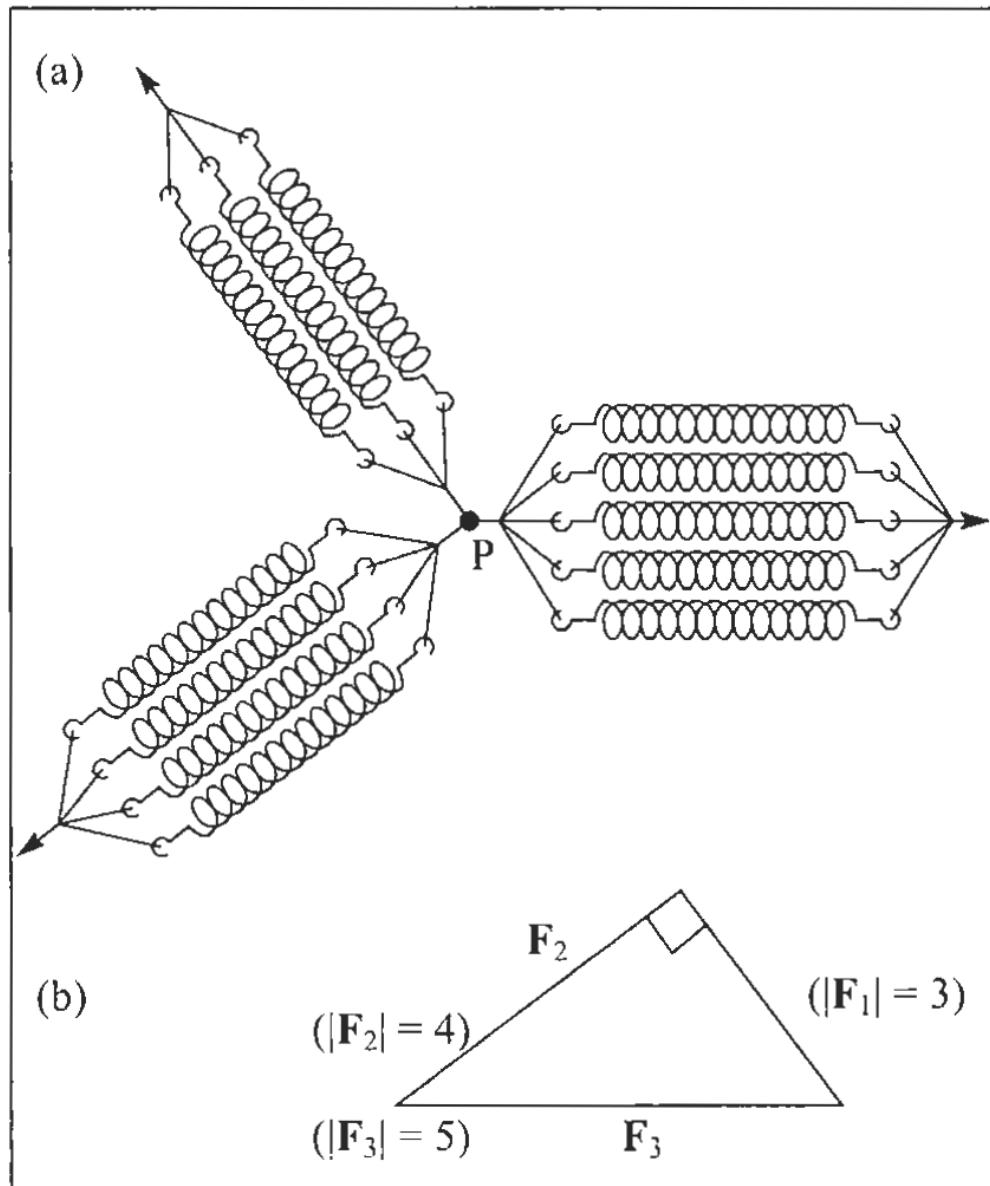


Figura 4.3 Equilíbrio de forças.

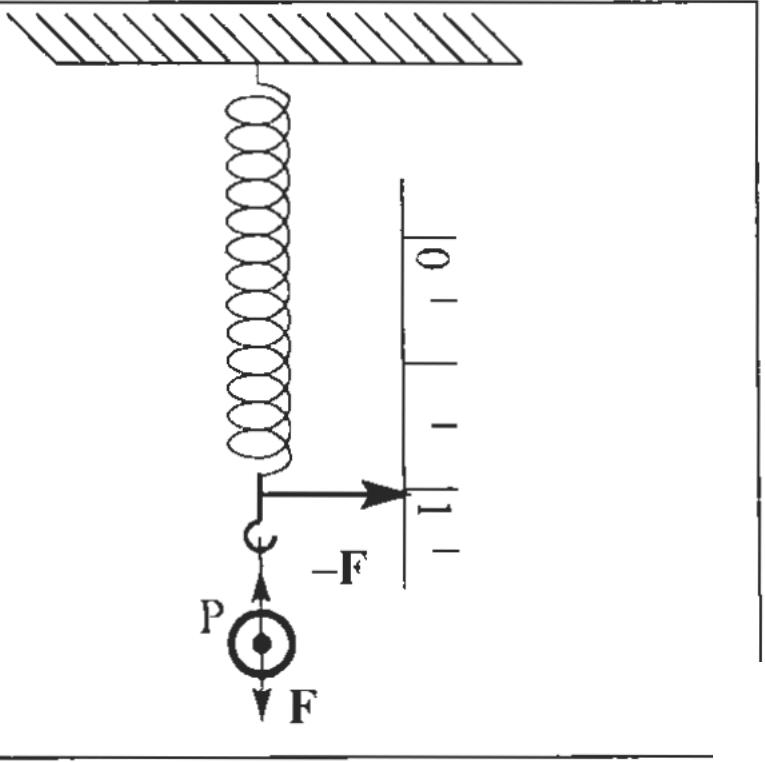


Figura 4.4 Força-peso.

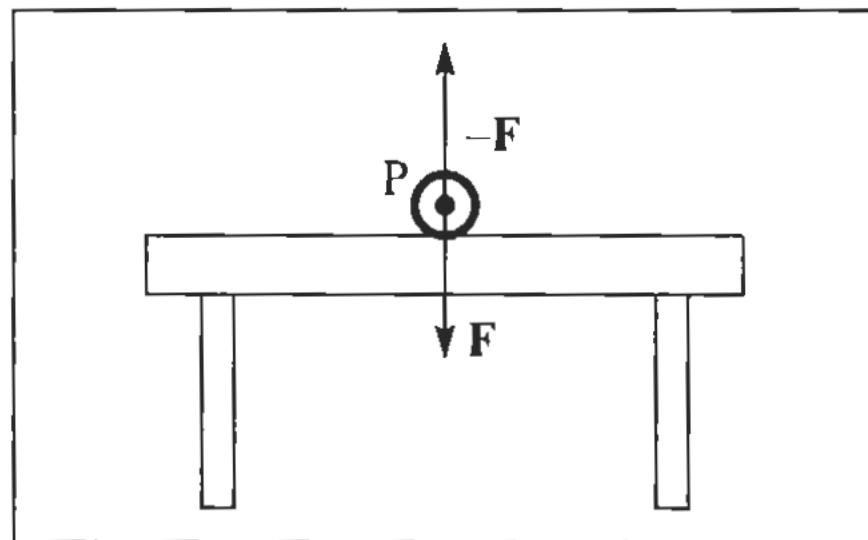


Figura 4.5 Reação de contato.

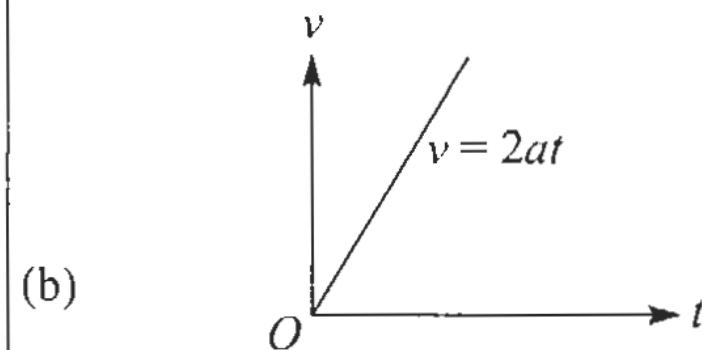
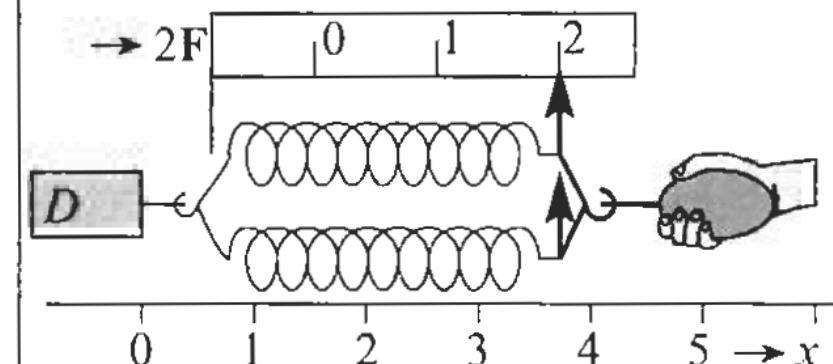
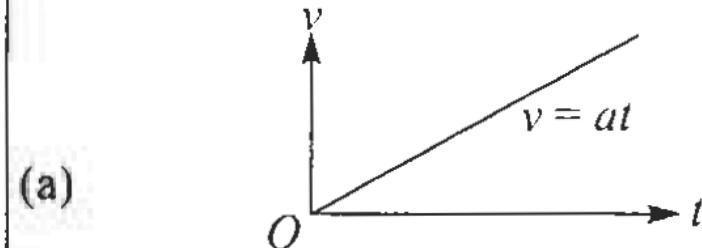
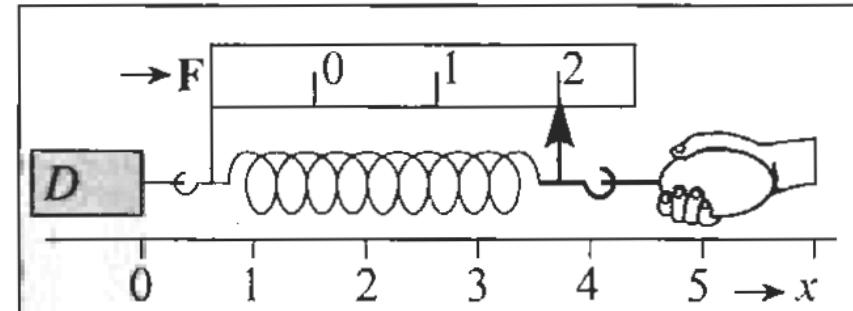
Em seu monumental tratado “Os Princípios Matemáticos da Filosofia Natural”, publicado em 1687, Newton formulou três “Axiomas ou Leis do Movimento”.

A 1.^a Lei é a Lei da Inércia:

“Todo corpo persiste em seu estado de repouso, ou de movimento retilíneo uniforme, a menos que seja compelido a modificar esse estado pela ação de forças impressas sobre ele”.

nos permitem inferir assim a 2.^a Lei de Newton

$$\boxed{F = ma}$$



A (4.3.3) não corresponde à formulação original de Newton da 2.^a lei. Newton começou definindo o que chamou de “quantidade de movimento”, também conhecido como *momento linear*, ou simplesmente *momento*. A definição de Newton foi:

“A quantidade de movimento é a medida do mesmo, que se origina conjuntamente da velocidade e da massa”.

Ou seja: o momento (linear) de uma partícula é o produto de sua massa por sua velocidade:

$$\boxed{\mathbf{p} = mv} \quad (4.4.2)$$

Decorre imediatamente desta definição que \mathbf{p} é um vetor.

Se m não varia com o tempo, ou seja, se excluirmos sistemas de massa variável, obtemos, derivando em relação ao tempo ambos os membros da (4.4.2) (cf.(2.2.5)),

$$\frac{d\mathbf{p}}{dt} = m \frac{d\mathbf{v}}{dt} = m\mathbf{a} \quad (4.4.3)$$

e, comparando com a (4.3.3),

$$\boxed{\frac{d\mathbf{p}}{dt} = \mathbf{F}} \quad (4.4.4)$$

o que corresponde à formulação de Newton da 2.^a lei:

“A variação do momento é proporcional à força impressa, e tem a direção da força”.

Exemplo 1: Força-peso: Substituindo a (3.6.1) na (4.3.3), vemos que a força P que atua sobre um corpo na vizinhança da superfície da Terra devido à atração gravitacional por ela exercida sobre o corpo é

$$P = mg \quad (4.4.5)$$

onde m é a massa inercial do corpo e g a aceleração da gravidade, vertical, dirigida para baixo e de magnitude g . A (4.4.5) chama-se *força-peso*; pode ser medida em equilíbrio pela balança de mola (Seç. 4.1). Para uma partícula em queda livre, a 2^a lei de Newton leva à (3.6.1),

$$\mathbf{a} = \mathbf{g} \quad (4.4.6)$$

A proporcionalidade da força-peso à massa inercial é uma peculiaridade notável dessa força, que voltaremos a discutir no capítulo sobre gravitação. É graças a ela que a aceleração da gravidade é a mesma para qualquer partícula (cf. (4.4.6) e (2.6.1)). É também graças a ela que podemos medir a massa inercial pelo peso, por exemplo, por pesagem com uma balança de mola. É importante, porém, evitar confusão entre os conceitos de massa e peso, que são totalmente diferentes. Num ponto muito distante da superfície da Terra (na superfície da Lua, por exemplo), o peso de uma partícula, indicado pela distensão da balança de mola, seria muito diferente, embora sua massa não se tenha alterado. Aliás, o peso sofre pequenas variações mesmo de ponto a ponto da superfície da Terra, devido às variações locais de g .

Em engenharia, é comum utilizar como unidade de força o *quilograma-força* (kgf), definido como a força-peso sobre uma massa de 1 kg ao nível do mar e na latitude de 45°N (onde $g \approx 9,81 \text{ m/s}^2$). Na prática, podemos tomar: $1 \text{ kgf} \approx 9,8 \text{ N}$.

Chapter 2

Tools for Doing Simulations

We introduce some of the core syntax of Java in the context of simulating the motion of falling particles near the Earth's surface. A simple algorithm for solving first-order differential equations numerically is also discussed.

Capítulo 2

- Partícula em queda livre

$$F_g = -mg,$$

- Segunda lei de Newton

$$m \frac{d^2y}{dt^2} = F,$$

we set $F = F_g$, (2.1) and (2.2) lead to

$$\frac{d^2y}{dt^2} = -g.$$

$$y(t) = y(0) + v(0)t - \frac{1}{2}gt^2$$

$$v(t) = v(0) - gt.$$

$$\begin{aligned}\frac{dy}{dt} &= v \\ \frac{dv}{dt} &= -g,\end{aligned}$$

$$\begin{aligned}\frac{y(t + \Delta t) - y(t)}{\Delta t} &= v(t) \\ \frac{v(t + \Delta t) - v(t)}{\Delta t} &= -g.\end{aligned}$$

Exercício em sala

Diferenças finitas: Algoritmo de Euler

$$y(t + \Delta t) = y(t) + v(t)\Delta t$$

$$v(t + \Delta t) = v(t) - g\Delta t,$$

Programar comparando
as soluções numérica e
analítica.

Exercício em sala

Diferenças finitas: Algoritmo de Euler

$$y(t + \Delta t) = y(t) + v(t)\Delta t$$

$$v(t + \Delta t) = v(t) - g\Delta t,$$

Programar comparando
as soluções numérica e
analítica.

```
# rmcesar@usp.br
# cap 2 Comp Phys, Gold et al

# y: position; v: velocity
# calcula free fall usando formula analitica e algoritmo de Euler

def g():
    return 9.80665

# Analitica
def positionA(y0, v0, t):
    return y0 + v0*t - (g() * t**2) / 2.0

def velocityA(v0, t):
    return v0 - g() * t

def freeFallAnalytical(y0, v0, tf, step):

    t = 0
    v = []
    y = []
    while (t<=tf):
        y.append(positionA(y0,v0,t))
        v.append(velocityA(v0,t))
        t += step
    return y,v

# Euler
def positionE(y0, v0, dt):
    return y0 + v0*dt

def velocityE(v0, dt):
    return v0 - g()*dt
```

Exercício em sala

Resolva o problema abaixo:

$$\frac{dy}{dx} = f(x),$$

$$y_{n+1} = y_n + f(x_n)\Delta x.$$

Implemente o algoritmo de Euler para:

$$f(x) = 2x \text{ and } y(x=0) = 0. \quad y(x) = x^2,$$

Plote os gráficos da solução analítica e aproximada pelo algoritmo de Euler.

Open Source Physics Eclipse Workspace

The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** Java - osp_csm/src/org/opensourcephysics/sip/ch02/FirstFallingBallApp.java - Eclipse - /Users/cesar/doc/courses/modelagem-simulacao/eclipse/workspace_compadre
- Toolbar:** Standard Eclipse toolbar with various icons for file operations, search, and preferences.
- Package Explorer:** Shows the project structure:
 - osp
 - osp_csm
 - src
 - org.opensourcephysics.sip
 - ch01
 - ch02
 - BouncingBall.java
 - BouncingBallApp.java
 - CalculationApp.java
 - Complex.java
 - ComplexApp.java
 - FallingBall.java
 - FallingBallApp.java
 - FallingParticle.java
 - FallingParticleApp.java
 - FallingParticleCalcApp.java
 - FallingParticlePlotApp.java
 - FirstFallingBallApp.java
 - Particle.java
 - PlotFrameApp.java
 - SHO.java
 - SHOParticle.java
 - SimulationApp.java
 - ch03
 - ch04
 - ch05
 - ch06
 - ch07
 - ch08
 - ch09
 - ch10
 - ch11
 - ch12
 - ch13
 - Editor:** Displays the Java code for FirstFallingBallApp.java. The code is a simple application that calculates the time for a ball to fall 10 meters using a loop and a time step of 0.01 seconds.
 - Outline View:** Shows the class structure with the main method highlighted.
 - Task List:** An empty list of tasks.
 - Problems:** No problems listed.
 - Console:** Shows the output of the application: <terminated> FirstFallingBallApp [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_25.jdk/Contents/Home/bin/java (18/07/2015 12:02:55)
 - Status Bar:** Writable | Smart Insert | 36 : 3

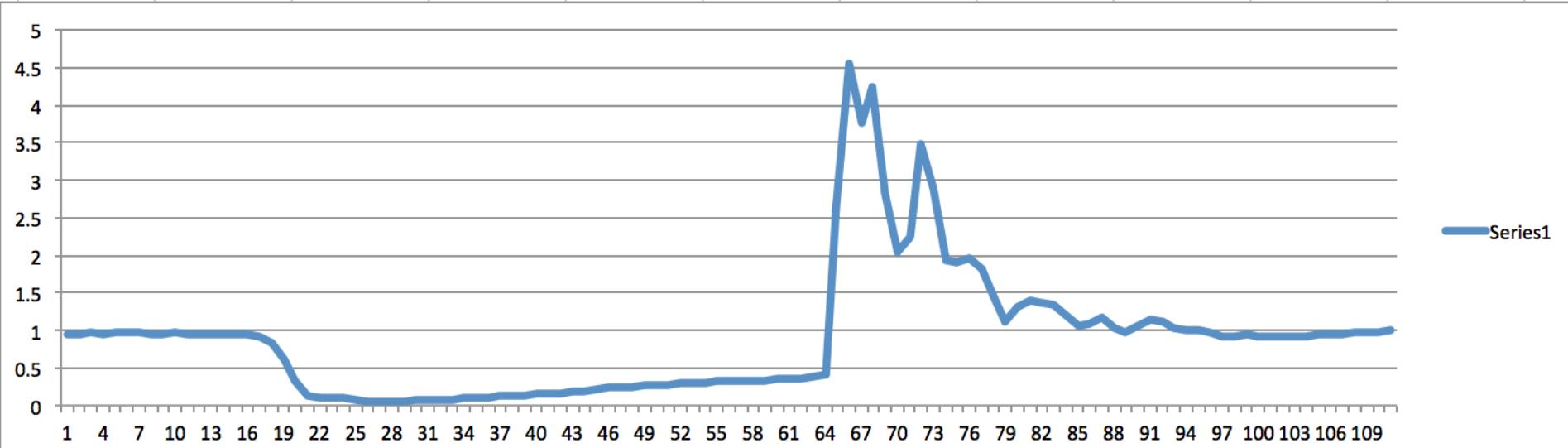
```
public class FirstFallingBallApp {           // beginning of class definition
    public static void main(String[] args) { // beginning of method definition
        // braces {} used to group statements.
        // indent statements within a block so that they can be easily identified
        // following statements form the body of main method
        double y0 = 10;      // example of declaration and assignment statement
        double v0 = 0;       // initial velocity
        double t = 0;        // time
        double dt = 0.01;   // time step
        double y = y0;
        double v = v0;
        double g = 9.8;     // gravitational field
        for(int n = 0;n<100;n++) { // beginning of loop , n++ equivalent to n+1
            // repeat following three statements 100 times
            y = y+v*dt; // indent statements in loop for clarity
            v = v-g*dt; // use Euler algorithm
            t = t+dt;
        }               // end of for loop
        System.out.println("Results");
        System.out.println("final time = "+t);
        // display numerical result
        System.out.println("y = "+y+" v = "+v);
        // display analytic result
        double yAnalytic = y0+v0*t-0.5*g*t*t;
        double vAnalytic = v0-g*t;
        System.out.println("analytic y = "+yAnalytic+" v = "+vAnalytic);
    } // end of method definition
} // end of class definition
```

Exercise 2.4. Exploring FirstFallingBallApp

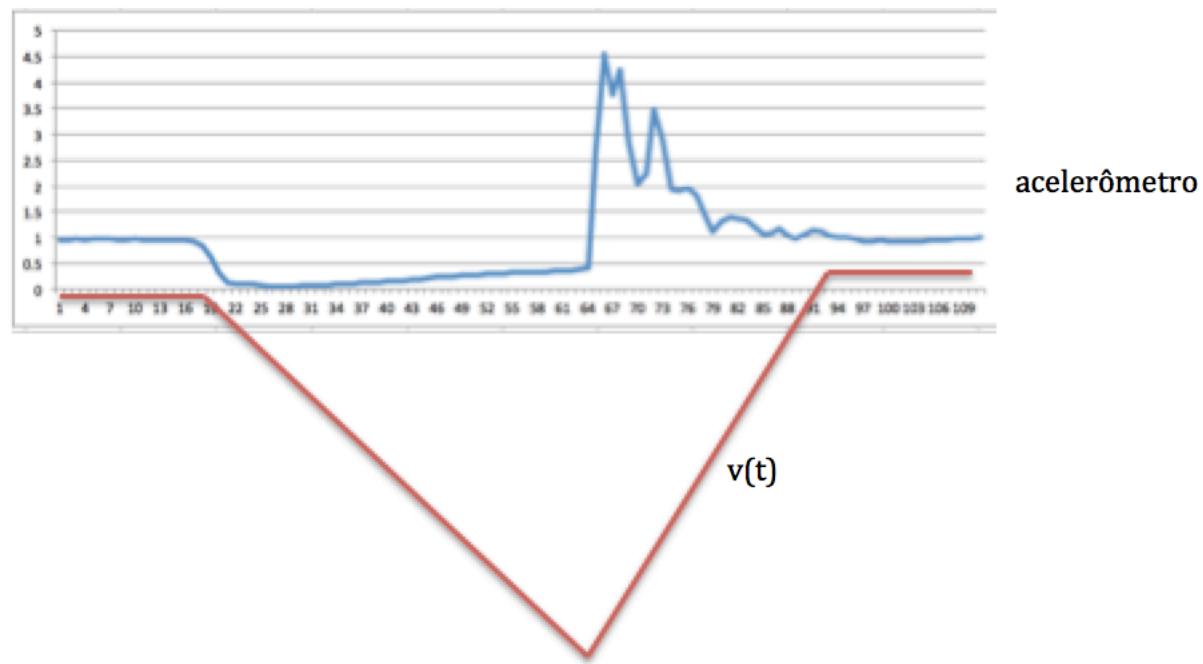
- a. Run `FirstFallingBallApp` for various values of the time step Δt . Do the numerical results become closer to the analytic results as Δt is made smaller?
- b. Use an acceptable value for Δt and run the program for various values for the number of iterations. What criteria do you have for acceptable? At approximately what time does the ball hit the ground at $y = 0$?
- c. What happens if you replace the `System.out.println` method by the `System.out.print` method?
- d. What happens if you try to access the value of the counter variable `n` outside the for loop? The *scope* of `n` extends from its declaration to the end of the loop block; `n` is said to have *block scope*. If a loop variable is not needed outside the loop, it should be declared in the initialization expression so that its scope is limited.

Exercícios de implementação no CEC

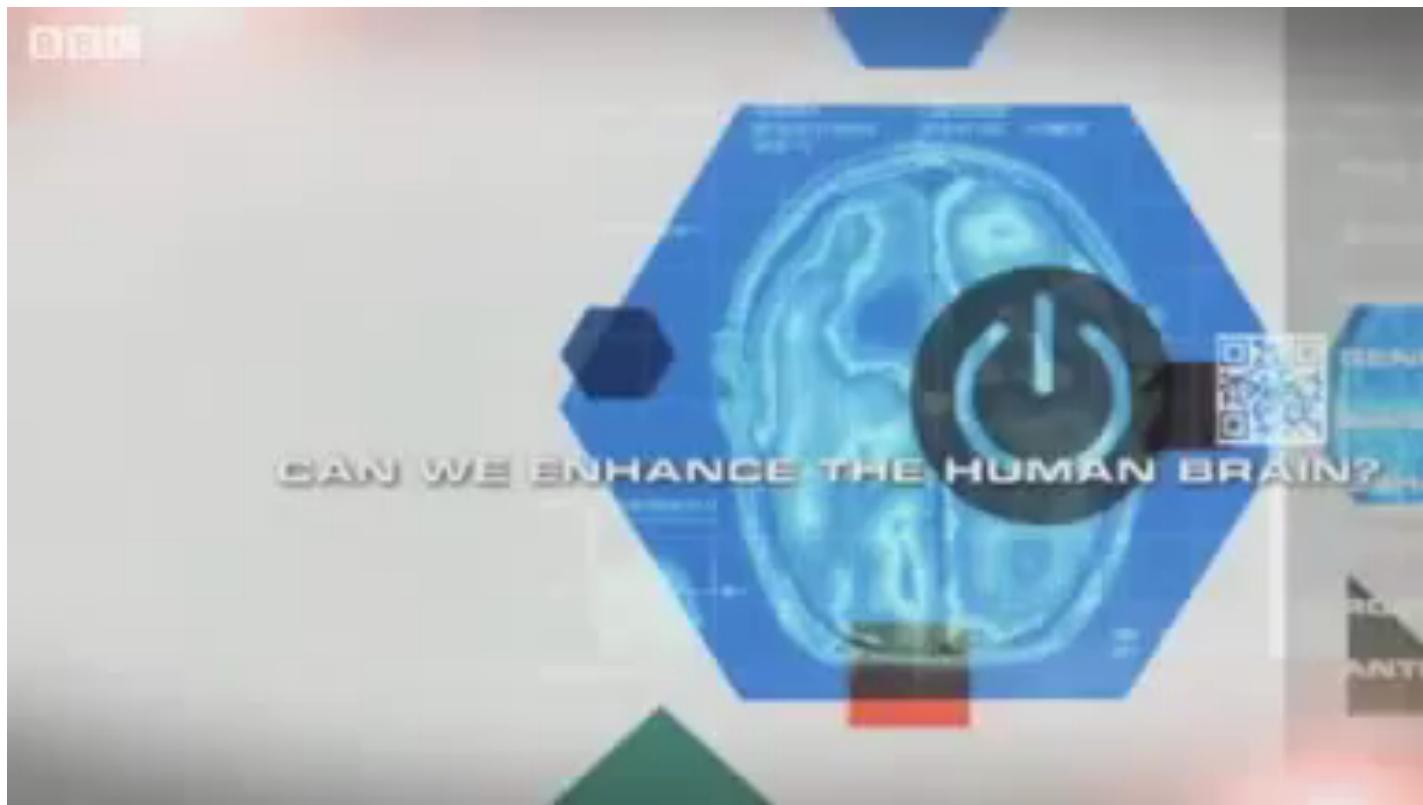
- Refazer os 2 exercícios do Jupyter Movimento 1D
- Implementar $dv/dt = -g$ (queda livre)
- Implementar $dy/dt = 2t$
- Usando os dados da queda livre do Bob Esponja, calcular $v(t)$ e sobrepor aos dados do acelerômetro



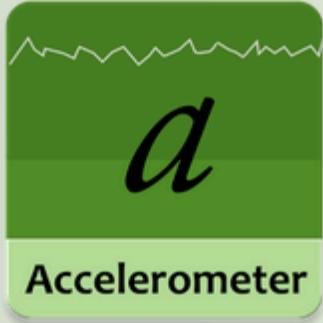
Usando os dados da queda
livre do Bob Esponja, calcular
v(t) e sobrepor aos dados do
acelerômetro



Capturando dados com sensores



Video: Future Technology will



Physics Toolbox Accelerometer

Free

Vieyra Software | February 17, 2015 | Tools

Like 0

g+1 0

Tweet 0

Accelerometer

Used this app? Rate it

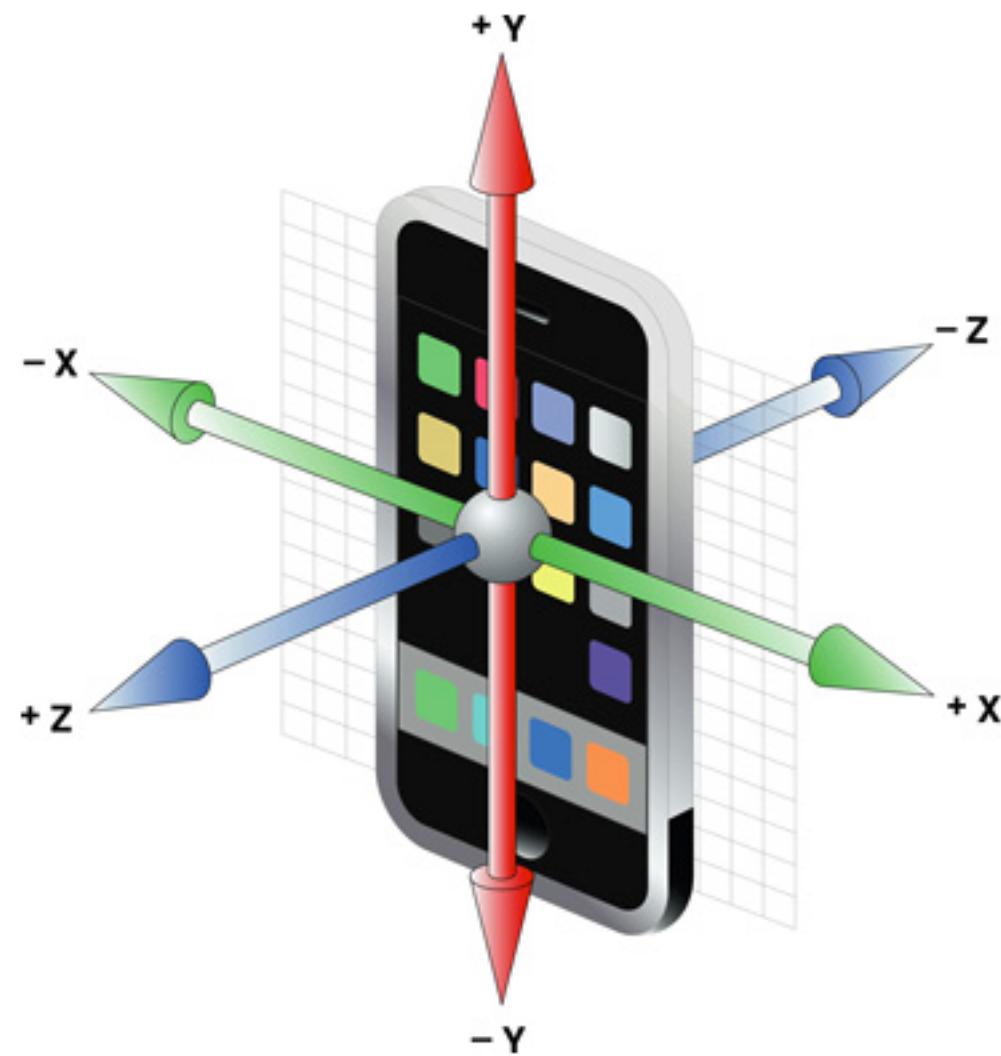


9 users
installed this application

Physics Toolbox Video

Analyzing free fall with a smartphone acceleration sensor

Patrik Vogt and Jochen Kuhn, Department of Physics/
Didactics of Physics, University of Kaiserslautern, Erwin-Schrödinger-Str.,
67663 Kaiserslautern, Germany; vogt@physik.uni-kl.de



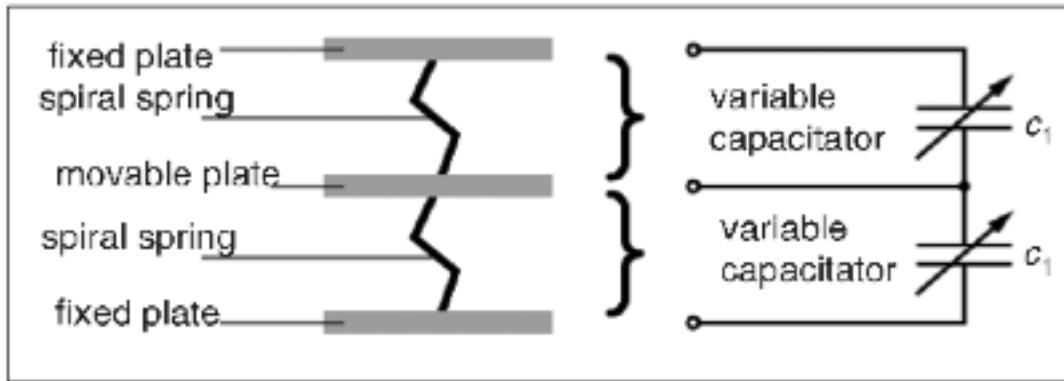


Fig. 2. Design and mode of operation of acceleration sensors.⁵

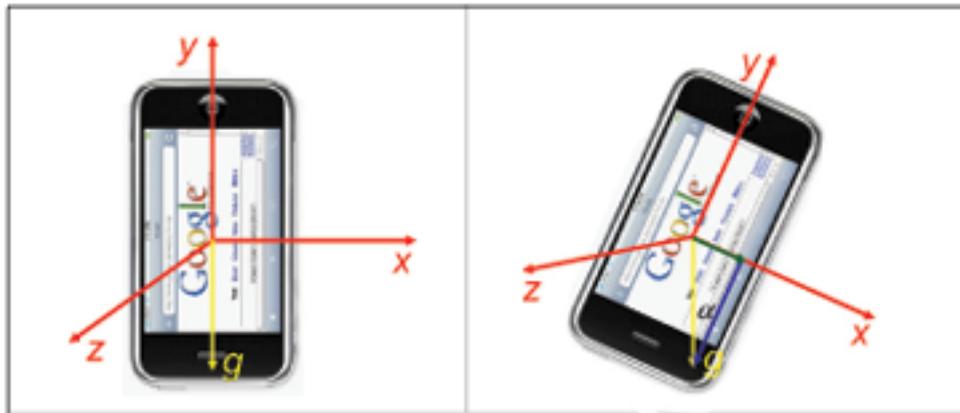


Fig. 3. The orientation of the three independent acceleration-sensors of an iPhone or iPod touch; the sensors measure the acceleration in the direction of the three plotted axes.



É HORA DE ...

...EXPERIMENTO!

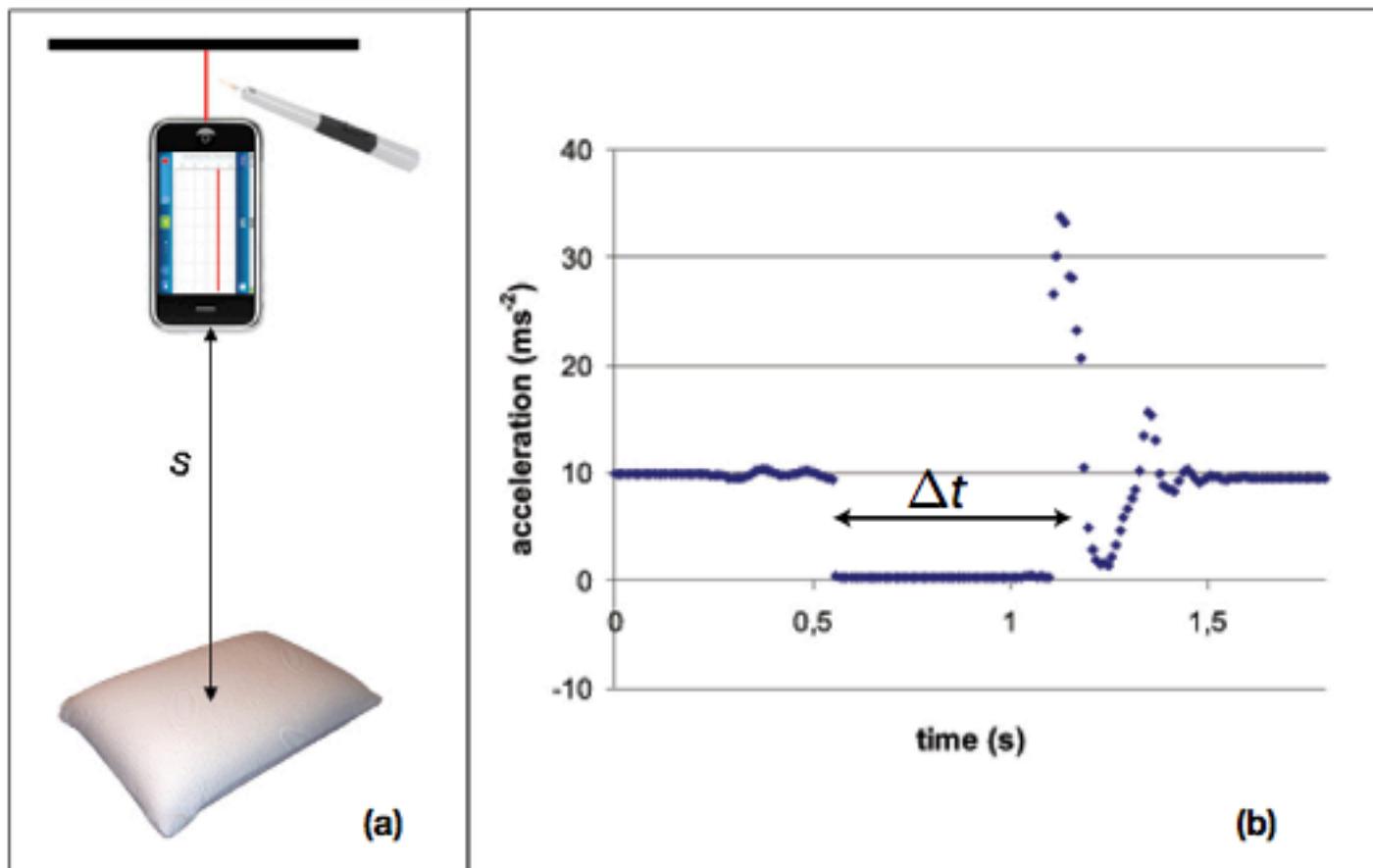
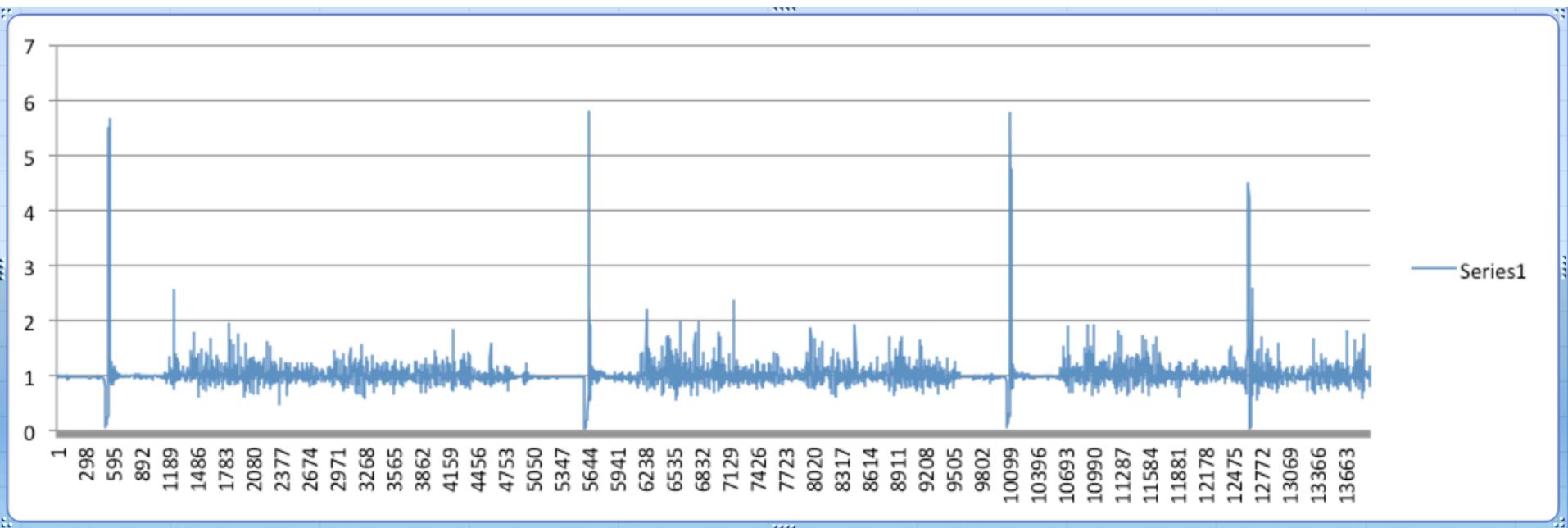


Fig. 4. Free fall: (a) Experimental setup and acceleration process. (b) Presentation of measurements after the export of data from the smartphone into MS Excel.

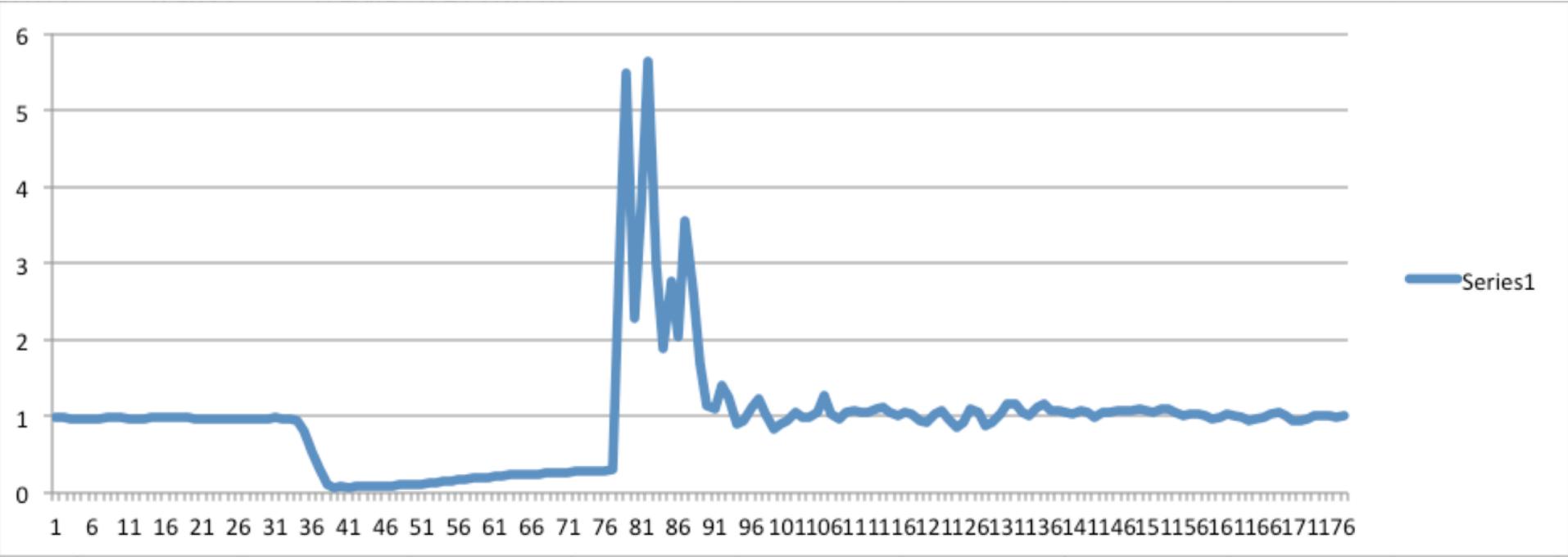
Exercício

- Analise os dados do acelerômetro:
- Use o algoritmo de Euler para calcular o intervalo de queda livre.
- Calcule e plote os gráficos de velocidade e posição ao longo do tempo.
- Calcule e compare a velocidade média experimental com a aproximada numericamente.



Experimento realizado no CCSL: três quedas livres seguidas.

Zoom nos dados em uma das quedas



MOVIMENTO HARMÔNICO SIMPLES

**H. Moysés
Nussenzveig**



*Fluidos
Oscilações e Ondas
Calor*

2



CURSO DE
**FÍSICA
BÁSICA**

Blucher

Blucher

BÁSICA

Curso de Física Básica – Fluidos,
Oscilações e Ondas, Calor – H.

Blucher

Curso de Física Básica
Fluidos, Oscilações e Ondas, Calor – Vol. 2
5^a Edição

Capítulo 3
O oscilador harmônico

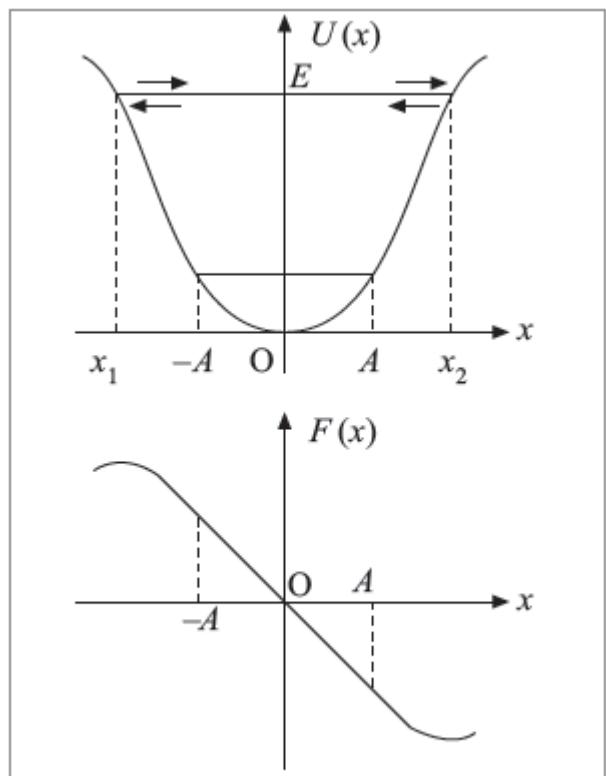


Figura 3.1 Energia potencial U e força F num movimento oscilatório.

$$F(x) = - \frac{dU}{dx} \quad (3.1.1)$$

Vemos também que, para *pequenos desvios da posição de equilíbrio estável*, o gráfico de $F(x)$ é aproximadamente linear. Assim, na fig. ao lado, para $-A \leq x \leq A$, temos aproximadamente

$$F(x) = -kx \quad (3.1.2)$$

$$U(x) = \frac{1}{2} kx^2 \quad (3.1.3)$$

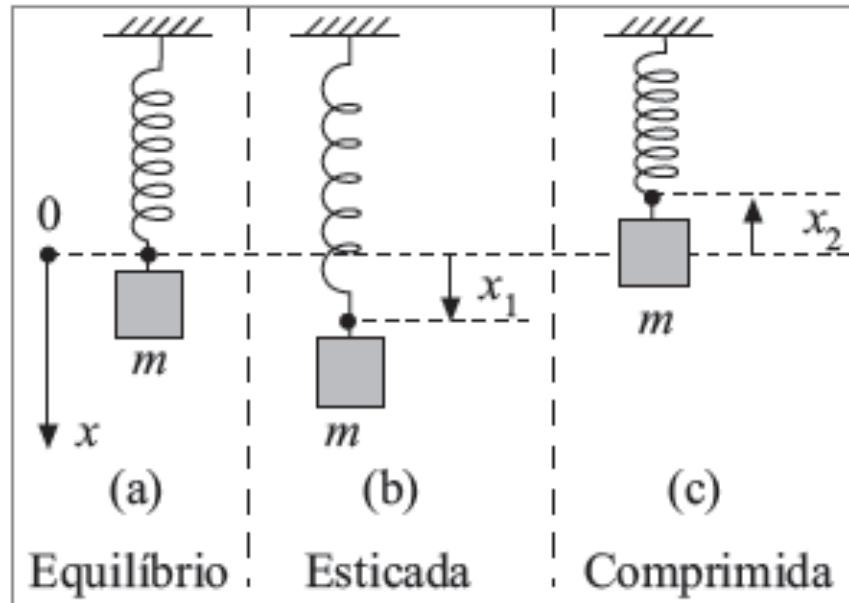


Figura 3.2 Massa e mola.

A equação de movimento correspondente é

$$m \ddot{x} = F(x) = -kx \quad (3.1.4)$$

ou seja,

$$\ddot{x} = \frac{d^2 x}{dt^2} = -\omega^2 x \quad (3.1.5)$$

$$\omega = \sqrt{\frac{k}{m}} \quad (3.1.6)$$

(a) *Soluções*

O movimento de um oscilador harmônico chama-se *movimento harmônico simples* (MHS). Para obter a lei horária do MHS, temos de resolver a equação de movimento (3.1.5) em relação à função incógnita $x(t)$. A (3.1.5) é uma *equação diferencial ordinária* para $x(t)$, porque contém derivadas de x em relação a t . Ela é de 2^a *ordem*, porque a derivada mais elevada que aparece é a 2^a.

Para resolver uma equação diferencial mais geral , como a (3.1.5) , dadas as condições iniciais, podemos sempre procurar soluções aproximadas por métodos numéricos. Na prática, isto se faz efetivamente, com o auxílio de computadores. Para Δt suficientemente pequeno, podemos aproximar

$$\frac{d^2 x}{dt^2}(t) \approx \frac{1}{\Delta t} \left[\frac{dx}{dt}(t + \Delta t) - \frac{dx}{dt}(t) \right] \quad (3.2.6)$$

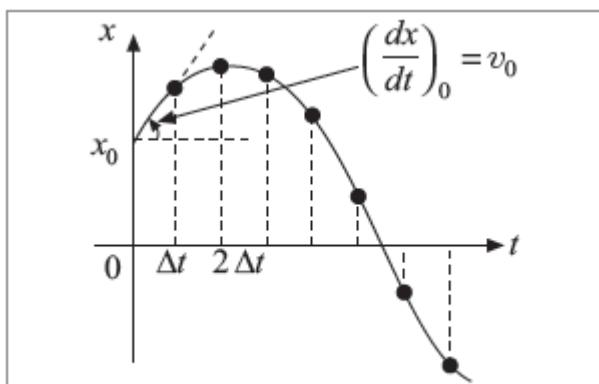


Figura 3.3 Resolução numérica.

A (3.1.5) se escreve

$$\frac{d}{dt} \left(\frac{dx}{dt} \right) = -\omega^2 x \quad (3.2.10)$$

O gráfico aproximado lembra uma senóide, sugerindo considerar soluções do tipo $\sin(ct)$, $\cos(ct)$ onde c é uma constante a ser ajustada. Temos:

$$\frac{d}{dt} [\sin(ct)] = c \cos(ct) \quad \left\{ \frac{d^2}{dt^2} [\sin(ct)] = -c^2 \sin(ct) \right.$$

$$\frac{d}{dt} [\cos(ct)] = -c \sin(ct) \quad \left\{ \frac{d^2}{dt^2} [\cos(ct)] = -c^2 \cos(ct) \right.$$

Logo, tomando $c = \omega$, obtemos as seguintes soluções da (3.1.5):

$$x_1(t) = \cos(\omega t)$$

$$x_2(t) = \sin(\omega t)$$

(3.2.11)

(b) *Linearidade e princípio de superposição*

A (3.1.5) é uma equação diferencial *linear*, ou seja, só contém termos *lineares* na função incógnita e suas derivadas : não comparecem termos em x^2 , x^3 , ... , $(dx/dt)^2$, ... , $(d^2x/dt^2)^3$, A equação diferencial linear de 2^a ordem mais geral é da forma

$$A \frac{d^2x}{dt^2} + B \frac{dx}{dt} + Cx = F \quad . \quad (3.2.12)$$

onde os coeficientes A , B , C e F não dependem de x , mas poderiam, em geral, depender de t . Na (3.1.5), esses coeficientes são *constantes*. Além disso, a (3.1.5) é uma equação *homogênea*, ou seja, com

$$F = 0 \quad (3.2.13)$$

Qualquer *equação diferencial linear de 2^a ordem homogênea* tem as seguintes propriedades fundamentais, cuja verificação é imediata:

(i) Se $x_1(t)$ e $x_2(t)$ são soluções, $x_1(t) + x_2(t)$ também é.

(ii) Se $x(t)$ é solução, $a x(t)$ ($a = \text{const.}$) também é.

Note que estes resultados não seriam válidos para equações não-lineares: por exemplo, se o 2º membro da (3.1.5) fosse proporcional a x^2 em lugar de x (verifique!).

Combinando (i) e (ii), vemos que, se $x_1(t)$ e $x_2(t)$ são soluções, qualquer *combinação linear*

$$x(t) = a x_1(t) + b x_2(t) \quad (3.2.14)$$

onde a e b são constantes arbitrárias, é solução.

Este resultado é uma forma do *princípio de superposição*. Resultados análogos valem para equações diferenciais lineares de ordem qualquer, como é fácil ver.

Uma consequência imediata é que, se $x_1(t)$ e $x_2(t)$ são duas soluções *independentes*, ou seja, se $x_2(t)$ não é múltipla de $x_1(t)$, a (3.2.14) é a *solução geral*, pois depende de duas constantes arbitrárias a e b [se $x_2(t)$ fosse múltipla de $x_1(t)$, $x_2(t) = c x_1(t)$, a (3.2.14) ficaria : $x(t) = (a + b c) x_1(t) = d x_1(t)$, ou seja, só teríamos uma constante efetivamente ajustável].

Aplicando a (3.2.14) à (3.2.11), obtemos a forma geral das *oscilações livres do oscilador harmônico* :

$$x(t) = a \cos(\omega t) + b \sin(\omega t) \quad (3.2.15)$$

o que também podemos escrever de outra forma equivalente:

$$x(t) = A \cos(\omega t + \varphi) \quad (3.2.16)$$

onde as duas constantes arbitrárias passam a ser A e φ .

E fácil obter a relação entre essas duas formas de escrever a solução. Como $\cos(\omega t + \varphi) = \cos(\omega t)\cos\varphi - \sin(\omega t)\sin\varphi$, vem

$$a = A \cos \varphi, \quad b = -A \sin \varphi \quad (3.2.17)$$

Inversamente, dados a e b , podemos obter A e φ :

$$\boxed{A = \sqrt{a^2 + b^2}} \\ \boxed{\cos \varphi = \frac{a}{\sqrt{a^2 + b^2}}, \quad \sin \varphi = -\frac{b}{\sqrt{a^2 + b^2}}} \quad (3.2.18)$$

o que determina φ a menos de um múltiplo de 2π , ou seja, de forma consistente com a (3.2.16), onde φ só é definido a menos de um múltiplo de 2π .

(c) *Interpretação física dos parâmetros*

Pela (3.2.16), vemos que $x(t)$ oscila entre os valores extremos $-A$ e A (cf. pg. 40). Logo, $A = |x(t)|_{\max}$ = *amplitude de oscilação*.

Como $\cos(\omega t + \varphi)$ é uma função periódica de ωt de período 2π , vemos que o *período* de oscilação é

$$\tau = \frac{2\pi}{\omega} = \frac{1}{v} \quad (3.2.19)$$

onde v , a *freqüência* de oscilação, se mede em ciclos por segundo ou hertz (Hz). A grandeza $\omega = 2\pi v$ chama-se *freqüência angular* e se mede em rad/s ou simplesmente s⁻¹. O argumento do cosseno na (3.2.16),

$$\theta = \omega t + \varphi \quad (3.2.20)$$

chama-se *fase* do movimento, e φ é a *constante de fase* ou *fase inicial* (valor da fase para $t = 0$).

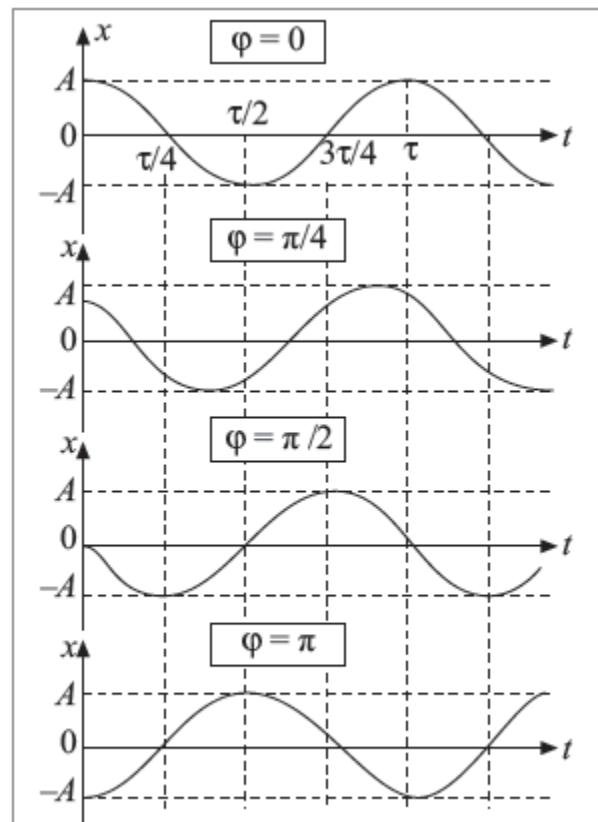


Figura 3.4 Variação da fase inicial.

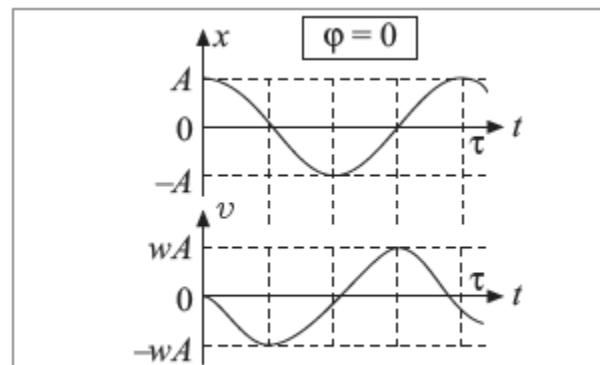


Figura 3.5 Deslocamento e velocidade.

(e) *Energia do oscilador*

Pela (3.2.23), a energia cinética do oscilador no instante t é

$$T(t) = \frac{1}{2} m \dot{x}^2(t) = \frac{1}{2} m \omega^2 A^2 \sin^2(\omega t + \varphi) \quad (3.2.28)$$

e, pelas (3.1.3) e (3.2.16), a energia potencial é

$$U(t) = \frac{1}{2} k x^2 = \frac{1}{2} m \omega^2 x^2 = \frac{1}{2} m \omega^2 A^2 \cos^2(\omega t + \varphi) \quad (3.2.29)$$

onde utilizamos também a (3.2.21).

Somando membro a membro, obtemos a energia total E , que se conserva:

$$\frac{1}{2} m \dot{x}^2 + \frac{1}{2} k x^2 = E = \frac{1}{2} m \omega^2 A^2 = \text{const.} \quad (3.2.30)$$

Vemos que a energia total é proporcional ao quadrado da amplitude, e também ao quadrado da freqüência.

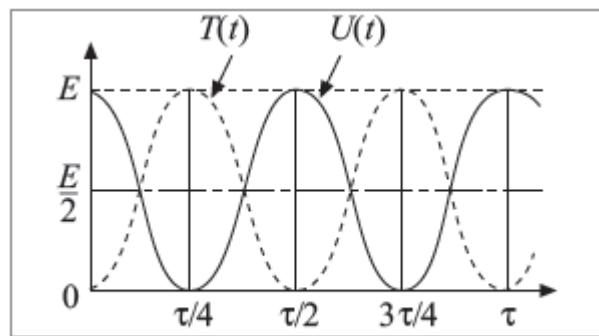


Figura 3.6 Valores médios de T e U .

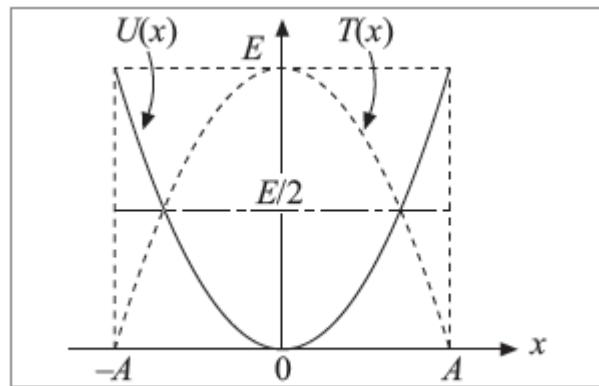
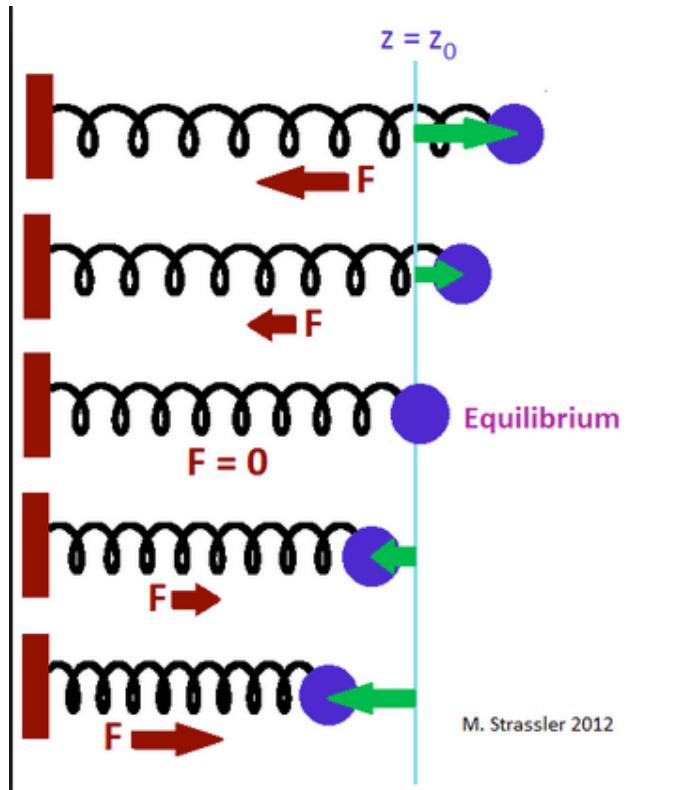


Figura 3.7 U e T em função de x .

It is easy to create additional models for other kinds of motion. Cut and paste the code in the `FallingBall` into a new file named `SHO.java` and change the code to solve the following two first-order differential equations for a ball attached to a spring:

$$\frac{dx}{dt} = v \quad (2.11a)$$

$$\frac{dv}{dt} = -\frac{k}{m}x, \quad (2.11b)$$



M. Strassler 2012

```
double x, v, t;                                // the dynamical variables
double dt;
double k = 1.0;                                  // spring constant
double omega0 = Math.sqrt(k); // assume unit mass

public SHO() {                                     // constructor
    System.out.println("A new harmonic oscillator object is created.");
}

public void step() {
    // modified Euler algorithm
    v = v-k*x*dt;
    x = x+v*dt; // note that updated v is used
    t = t+dt;
}

public double analyticPosition(double y0, double v0) {
    return y0*Math.cos(omega0*t)+v0/omega0*Math.sin(omega0*t);
}

public double analyticVelocity(double y0, double v0) {
    return -y0*omega0*Math.sin(omega0*t)+v0*Math.cos(omega0*t);
}
```

Exercise 2.9. Simple harmonic oscillator

- a. Explain how the implementation of the Euler algorithm in the `step` method of class `SHO` differs from what we did previously.
- b. The general form of the analytical solution of (2.11) can be expressed as

$$y(t) = A \cos \omega_0 t + B \sin \omega_0 t, \quad (2.12)$$

where $\omega_0^2 = k/m$. What is the form of $v(t)$? Show that (2.12) satisfies (2.11) with $A = y(t = 0)$ and $B = v(t = 0)/\omega_0$. These analytical solutions are used in class `SHO`.

- c. Write a target class called `SHOApp` that creates an `SHO` object and solves (2.11). Start the ball with displacements of $x = 1$, $x = 2$, and $x = 4$. Is the time it takes for the ball to reach $x = 0$ always the same?

Exercise 2.12. Extending classes

- a. Extend the `FallingParticle` and `SHOParticle` classes and give them names such as `FallingParticleEC` and `SHOParticleEC`, respectively. These subclasses should redefine the `step` method so that it first calculates the new velocity and then calculates the new position using the new velocity, that is,

```
public void step() {  
    v = v - g*dt;           // falling ball  
    y = y + v*dt; f  
    t = t + dt;  
}  
  
public void step() {  
    v = v - k*x*dt;        // harmonic oscillator  
    x = x + v*dt;  
    t = t + dt;  
}
```

Methods can be redefined (overloaded) in the subclass by writing a new method in the subclass definition with the same name and parameter list as the super class definition.

- b. Confirm that your new `step` method is executed instead of the one in the superclass.
- c. The algorithm that is implemented in the redefined step method is known as the *Euler-Cromer* algorithm. Compare the accuracy of this algorithm to the original Euler algorithm for both the falling particle and the harmonic oscillator. We will explore the Euler-Cromer algorithm in more detail in Problem 4.1.

Herança

Listing 2.5: Particle class.

```
package org.opensourcephysics.sip.ch02;
abstract public class Particle {
    double y, v, t;          // instance variables
    double dt;                // time step

    public Particle() { // constructor
        System.out.println("A new Particle is created.");
    }

    abstract protected void step();
    abstract protected double analyticPosition();
    abstract protected double analyticVelocity();
}
```

Listing 2.6: FallingParticle class.

```
package org.opensourcephysics.sip.ch02;
public class FallingParticle extends Particle {
    final static double g = 9.8;                                // constant
    private double
        y0 = 0, v0 = 0;                                         // initial position and velo
                                                                // constructor
    public FallingParticle(double y, double v) { System.out.println("A new FallingParticle object is created.");
        this.y = y; // instance value set equal to passed value
        this.v = v; // instance value set equal to passed value
        y0 = y;      // no need to use "this" because there is only one y0
        v0 = v;
    }
    public void step() {
        y = y+v*dt; // Euler algorithm
        t = t+dt;
    }
    public double analyticPosition() {
        return y0+v0*t-(g*t*t)/2.0;
    }
    public double analyticVelocity() {
        return v0-g*t;
    }
}
```

Listing 2.7: FallingParticleApp class.

```
package org.opensourcephysics.sip.ch02;
public class FallingParticleApp {           // beginning of class definition
    public static void main(String[] args) { // beginning of method definition
        Particle ball = new FallingParticle(10, 0); // declaration and instanti
        ball.t = 0;
        ball.dt = 0.01;
        while(ball.y>0) {
            ball.step();
        }
        System.out.println("Results");
        System.out.println("final time = "+ball.t);
        System.out.println("y = "+ball.y+" v = "+ball.v);           // numeri
        System.out.println("y analytic = "+ball.analyticPosition()); // analyt
    }
} // end of class definition
```

GUI E ANIMAÇÃO

Listing 2.13: BouncingBall class.

```
package org.opensourcephysics.sip.ch02;
import org.opensourcephysics.display.Circle;

public class BouncingBall extends Circle { // Circle is a class that can dr
    final static double g = 9.8; // const
    final static double WALL = 10;
    private double x, y, vx, vy; // init

    public BouncingBall(double x, double vx, double y, double vy) { // consi
        this.x = x; // sets instance value equal to passed value
        this.vx = vx; // sets instance value equal to passed value
        this.y = y;
        this.vy = vy;
        setXY(x, y); // sets the position using setXY in Circle superclass
    }

    public void step(double dt) {
        x = x+vx*dt; // Euler algorithm for numerical solution
        y = y+vy*dt;
        vy = vy-g*dt;
        if(x>WALL) {
            vx = -Math.abs(vx); // bounce off right wall
        } else if(x<-WALL) {
            vx = Math.abs(vx); // bounce off left wall
        }
        if(y<0) {
            vy = Math.abs(vy); // bounce off floor
        }
        setXY(x, y);
    }
}
```

Java - osp_csm/src/org/opensourcephysics/sip/ch02/BouncingBallApp.java - Eclipse - /Users/cesar/doc/courses/modelagem-simulacao/eclipse/workspace_compadre

The screenshot shows the Eclipse IDE interface with the following components:

- Package Explorer:** Shows the project structure under the package `org.opensourcephysics.sip.ch02`, including files like `BouncingBall.java`, `BouncingBallApp.java`, `CalculationApp.java`, etc.
- Code Editor:** Displays the `BouncingBallApp.java` file with Java code for a simulation. The code includes imports for Open Source Physics classes and defines a class `BouncingBallApp` that extends `AbstractSimulation`. It initializes a window, creates an array of `BouncingBall` objects, and sets boundaries and initial conditions.
- View Window:** A separate window titled "Bouncing Balls" shows a 2D plot of red circular particles representing balls. The x-axis ranges from -10 to 10, and the y-axis ranges from -4 to 14. The particles are scattered across the plot area.
- Console:** Shows the command line output for running the application: "BouncingBallApp [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_25.jdk/Contents/Home/bin/java (18/07/2015 14:22:24)"
- Properties View:** A floating dialog titled "BouncingBallApp Controller" shows input parameters: `dt` (0.1), `number of balls` (40), and `speed` (10). Buttons for `Stop`, `Step`, and `New` are available.
- Status Bar:** Shows status indicators for the editor: "Writable", "Smart Insert", and "2 : 1".

```

public void initialize() {
    // sets boundaries of window in world coordinates
    frame.setPreferredMinMax(-10.0, 10.0, 0, 10);
    time = 0;
    frame.clearDrawables(); // removes old particles
    int n = control.getInt("number of balls");
    int v = control.getInt("speed");
    ball = new BouncingBall[n]; // instantiates array of n BouncingBall
    for(int i = 0;i<n;i++) {
        double theta = Math.PI*Math.random(); // random angle
        // instantiates the ith BouncingBall object
        ball[i] = new BouncingBall(0, v*Math.cos(theta), 0, v*Math.sin(theta));
        frame.addDrawable(ball[i]); // adds ball to frame so it can be drawn
    }
    // decimalFormat instantiated in superclass and used to format numbers
    frame.setMessage("t = "+decimalFormat.format(time)); // appears in label
}

public void doStep() { // invoked every 1/60 second
    for(int i = 0;i<ball.length;i++) {
        ball[i].step(dt);
    }
    time += dt;
    frame.setMessage("t="+decimalFormat.format(time));
}

public void startRunning() { // invoked when start or step button is pressed
    dt = control.getDouble("dt");
} // gets time step

public void reset() { // invoked when reset button is pressed
    control.setAdjustableValue("dt", 0.1); // allows dt to be changed after
    control.setValue("number of balls", 40);
    control.setValue("speed", 10);
}

public static void main(String[] args) { // sets up animation control
    SimulationControl.createApp(new BouncingBallApp());
}

```

- a. Run `BouncingBallApp` and try the different buttons and note how they affect the input parameters.
- b. Add the statement, `enableStepsPerDisplay(true)` to the `reset` method, and run your program again. You should see a new input in the control window that lets you change the number of simulation steps that are computed between redrawing the frame. Vary this input and note what happens.
- c. What is wrong with the physics of the simulation?
- d. Add a method to the `BouncingBall` class to calculate and return the total energy. Sum the energy of the balls in the program's `doStep` method and display this value in the message box. Does the simple model make sense?
- e. Look at the source code for the `setXY` method. If you are using an Integrated Development Environment (IDE), finding the method and looking at the source code is easy. What would you need to do to change the radius of the circle that is drawn?

Crie uma versão do programa que detecte colisão das bolinhas e altere suas velocidades vetoriais.

Material de trabalho

- Leia o Capítulo 2 do livro texto.
- Resolva os exercícios desse capítulo.
- Procure o monitor ou o professor para suas dúvidas.