Centro de Instrução Almirante Wandenkolk - CIAW
Instituto Tecnológico de Aeronáutica - ITA

Curso de Aperfeiçoamento Avançado em Sistemas de Armas

SAB: Simulação e Controle de Artefatos Bélicos

Métodos Numéricos Computacionais

Jozias **Del Rios** Cap Eng
delriosjdrvgs@fab.mil.br
(12) 98177-9921

Abril 2018

AA-811
SIMULAÇÃO E CONTROLE DE ARTEFATOS BÉLICOS
Métodos Numéricos Computacionais

Instrutor: Cap Eng Jozias **DEL RIOS**

Autor do Material: Jozias **DEL RIOS** – rev. 25.mai.2016

TÓPICOS

Métodos Numéricos Computacionais

1. Sistema numérico binário
2. Representação de ponto-flutuante
3. Erros e desempenho
4. Interpoladores
5. Integradores numéricos
6. Geradores de números aleatórios

OBJETIVO

Nesta aula estudaremos algumas noções de

representações computacionais,

operações matemáticas e

ferramentas numéricas

para entender a forma e os óbices em que

os processadores (computadores e embarcado) fazem

cálculos matemáticos com números reais.

MOTIVAÇÃO: SIMULAÇÃO

Em simulações, é comum ocorrer a soma de números muito pequenos: infinitésimos, dx sobre números grandes: variáveis de estado, $x(t)$.

Os erros nos cálculos se acumulam e propagam a cada iteração da simulação, desviando da realidade.

MOTIVAÇÃO: CONTROLE EMBARCADO

Ao implementar as leis de controle em software embarcado, as restrições de capacidade de memória e de processamento obrigam a diminuição da precisão dos cálculos, ou uso de métodos aproximados, em comparação com o simulador, o MATLAB, etc.

SISTEMA NUMÉRICO BINÁRIO

Um número é escrito com algarismos **0** ou **1** que denotam a presença de potências positivas de 2:

Exemplo: número $(181)_{10} = (10110101)_{2}$

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
1	**0**	**1**	**1**	**0**	**1**	**0**	**1**

$$2^7 + 2^5 + 2^4 + 2^2 + 2^0 = \mathbf{181}$$

BINÁRIO: INTEIROS

bit é o nome de cada dígito binário.

byte é o conjunto de 8 bits:

- pode contar de 0 ⇨ 255, ou
- pode contar de –128 ⇨ +127

Regra geral: n bits são necessários para contar 2^n elementos:

- ou contar somente positivos de 0 ⇨ $(2^n - 1)$
- ou contar todos os inteiros de -2^{n-1} ⇨ $+(2^{n-1} - 1)$

Então, caso seja considerado números negativos inteiros, um bit é destinado a indicar o sinal do número "complemento de 2".

BINÁRIO: PONTO-FIXO

As frações são as potências negativas da base 2.

Vamos exemplificar com ponto-fixo 2.6:

2 bits na parte inteira, 6 bits na parte fracionária.

<u>Exemplo</u>: número $(2{,}625)_{10} = (10{,}101000)_{2}$

2^{1}	2^{0}	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}
2	1	0,5	0,25	0,125	0,0625	0,03125	0,015625
1	**0**	**1**	**0**	**1**	**0**	**0**	**0**

$$2^{1} + 2^{-1} + 2^{-3} = 2{,}625$$

BINÁRIO: PONTO-FIXO

Outro exemplo: número $(3{,}14)_{10} \approx (11{,}001001)_2$

2^{1}	2^{0}	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}
2	1	0,5	0,25	0,125	0,0625	0,03125	0,015625
1	**1**	**0**	**0**	**1**	**0**	**0**	**1**

$$2^{1} + 2^{0} + 2^{-3} + 2^{-6} = 3{,}140625 \approx 3{,}14$$

Arredondamento de representação: devido ao número finito de bits (6), o número desejado foi arredondado.

BINÁRIO: PONTO-FIXO

Outro exemplo: número $(3{,}15)_{10} \approx (11{,}001010)_2$

2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}
2	1	0,5	0,25	0,125	0,0625	0,03125	0,015625
1	**1**	**0**	**0**	**1**	**0**	**1**	**0**

$$2^1 + 2^0 + 2^{-3} + 2^{-5} = 3{,}15625 \approx 3{,}15$$

Este número na base 2 teria dízima periódica:

$$(3{,}15)_{10} = (11{,}00\ 1001\ 1001\ \underline{1001}\ \ldots)_2$$

PONTO-FLUTUANTE

Grandezas de engenharia utilizam multiplicadores muito pequenos (μ: 10^{-6}) e muito grandes (M: 10^{6}), o que exigira muitos bits para tratar com ponto-fixo.

A norma IEEE-754 define o ponto-flutuante para representar e armazenar números reais de forma semelhante à notação científica de engenharia:

$$\text{número} = (\text{sinal}) \cdot (\text{mantissa}) \cdot 2^{\text{expoente}}$$

	sinal	mantissa	expoente	total
Precisão simples:	**1** bit	**23** bits	**8** bits	**32** bits
Precisão dupla:	**1** bit	**52** bits	**11** bits	**64** bits

PONTO-FLUTUANTE

Precisão Dupla (64 bits):

- Denominados "`double`" em linguagem C.
- Usado em computação científica: Simuladores, MATLAB, etc.

Precisão Simples (32 bits):

- Denominados "`float`" em linguagem C.
- Usado em software embarcado e onde o desempenho for mais importante que a qualidade do resultado (visualização).
- Overflow: número de módulo maior que 2^{+127}
- Underflow: número de módulo menor que 2^{-128}

PONTO-FLUTUANTE: MANTISSA

$$\text{número} = (\text{sinal}) \cdot (\text{mantissa}) \cdot 2^{\text{expoente}}$$

Mantissa: algarismos significativos, vai de 1,000... até 1,999...

Exemplo: $c_0 = 299\,792\,458$ m/s escrito em precisão simples:

$$c_0 \approx c_{ps} = (+1) \cdot (1{,}116\,813\,898\,086\,547\,9) \cdot 2^{28}$$

Mantissa tem **23 bits**: $(1{,}0001\ 1101\ 1110\ 0111\ 1000\ 010)_2$

Escrevendo de volta em decimal: $c_{ps} = 299\,792\,4\underline{48}$
Portanto, o erro de arredondamento foi de 10 m/s.

Incerteza após o último bit da mantissa é o erro de quantização:

$$\left(2^{-24}\right) \cdot 2^{28} = 2^4 = 16 \text{ m/s}$$

PONTO-FLUTUANTE: TRUNCAMENTO

Operações de adição e subtração contribuem fortemente para a degradação da precisão do resultado. Exemplo: 0,99 + 0,0001:

$$0{,}9900 \approx \left(1{,}11111010111000010100011\right)_2 \cdot 2^{-1}$$

$$0{,}0001 \approx \left(1{,}10100011011011100010111\right)_2 \cdot 2^{-14}$$

$$(0{,}111111010111000010100011)_2 +$$
$$\underline{(0{,}\underbrace{0000000000000}\,11010001101\,1011100010111)_2} =$$
$$(0{,}111111010111011100110000)_2 \quad \text{(truncado/descartado)}$$

Foram perdidos 13 bits da precisão da parcela menor.
Observe que (-1) – (-14) = 13

PONTO-FLUTUANTE: PROCESSAMENTO

Simuladores (computadores) executam 3 bilhões ciclos/segundo (cada núcleo). Hardware embarcado (DSP, Digital Signal Processor) executam 200 milhões ciclos/segundo, único núcleo.

Ambos executam com tempos de processamento aproximadamente como a tabela ao lado.

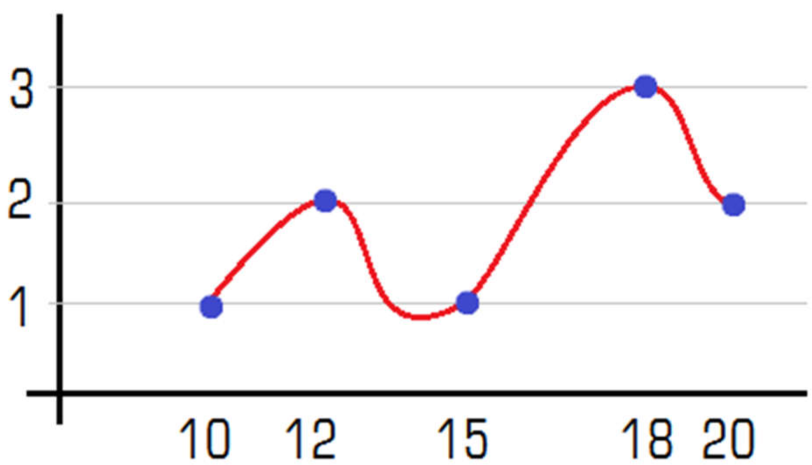
Cálculos mais complexos exigem que o processador compute uma série de Taylor ou faça iterações convergentes.

Operação	Ciclos
Soma / Subtração	1
Multiplicação	1
Divisão	15
Raiz Quadrada	15
Exponencial / Logaritmo	20
Funções Trigonométricas	30

INTERPOLADORES: MOTIVAÇÃO

Em simuladores, é muito comum que os valores dos parâmetros sejam amostrados de curvas, provenientes de experimentos.

As curvas (fórmulas empíricas) em si não são fornecidas. São fornecidos pontos amostrados, portanto uma tabela ordenada:



x	F(x)
10,0	1,0
12,0	2,0
15,0	1,0
18,0	3,0
20,0	2,0

INTERPOLADORES: BUSCA

O primeiro passo é buscar em qual intervalo do domínio pode-se encontrar o valor do parâmetro.

LUT: Look-Up Table

Embarcado, as LUT são o principal consumidor de memória.

A busca consome tempo, e pode ser feita de forma:

- Linear: compara um a um, começando do início: Lento!
- Busca binária: compara com o elemento central, então reinicia a busca no sub-intervalo superior ou inferior: Rápido.
- Cache: lembra-se qual foi o intervalo da última vez que fez a busca, e busca linear em volta dele: Rápido e simples!

INTERPOLADORES: BUSCA LINEAR

Exemplo: buscar o valor de F(x=17)

x	F(x)
10,0	1,0
12,0	2,0
15,0	1,0
18,0	3,0
20,0	2,0

Busca Linear:

$x < 10$? Não.

$x < 12$? Não.

$x < 15$? Não.

$x < 18$? Sim,

Então o intervalo é de 15 à 18.

INTERPOLADORES: BUSCA BINÁRIA

Exemplo: buscar o valor de $F(x=17)$

x	F(x)
10,0	1,0
12,0	2,0
15,0	1,0
18,0	3,0
20,0	2,0

Busca Binária:

Intervalo 10 à 20: elemento central: 15
$x < 15$? Não.

Intervalo 15 a 20, elemento central: 18
$x < 18$? Sim

Então o intervalo é de 15 à 18.

INTERPOLADORES: BUSCA COM CACHE

Exemplo: buscar o valor de $F(x=17)$

x	F(x)
10,0	1,0
12,0	2,0
15,0	1,0
18,0	3,0
20,0	2,0

Busca com Cache:

Tanto em simulador quanto em voo, o valor dos parâmetros devem variar pouco no tempo (aderência ao histórico).

Se o intervalo anteriormente encontrado foi 15 à 18, verificar se $x = 17$ está contido. Se não estiver, pode ser feita uma busca ao redor linear, crescente ou decrescente.

INTERPOLADORES

Foi identificado o intervalo (15 à 18) no domínio:

$$1{,}0 = F(x = 15)$$
$$y = F(x = 17)$$
$$3{,}0 = F(x = 18)$$

pode-se agora usar um dentre vários interpoladores:

- Nearest-Neighbor
- Linear
- Spline
- ...

INTERPOLAÇÃO: NEAREST-NEIGHBOR

Exemplo: buscar o valor de $F(x=17)$

x	F(x)
10,0	1,0
12,0	2,0
15,0	1,0
18,0	3,0
20,0	2,0

Outros nomes:

- Interpolador constante
- Hold de ordem zero (ZOH)
- Escada, Degrau, ...

Utiliza o valor da função no ponto mais próximo.

Ponto no domínio mais próximo de $x=17$ é o $x=18$, então $F(17) = 3,0$

INTERPOLAÇÃO: LINEAR

Amostra na reta que passa entre os pontos do intervalo:

$$y_0 = 1{,}0 = F(x_0 = 15)$$

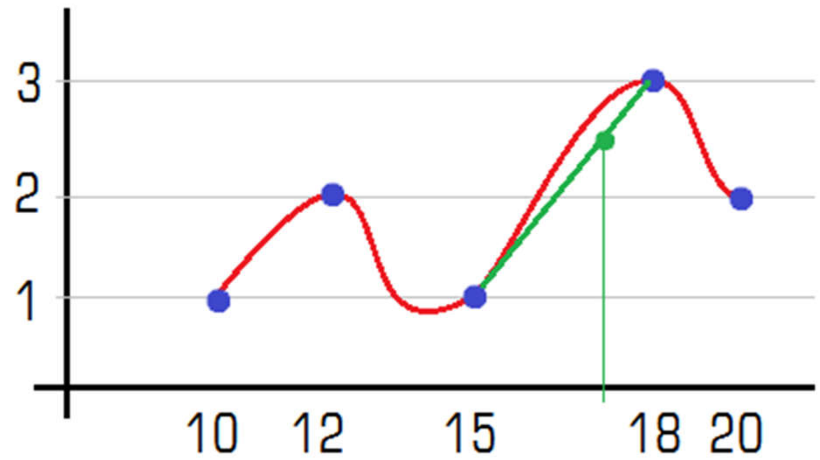
$$y = \boxed{} \qquad F(x = 17)$$

$$y_1 = 3{,}0 = F(x_1 = 18)$$

$$y = y_0 + \frac{x - x_0}{x_1 - x_0}(y_1 - y_0)$$

$$y = 1 + \frac{17 - 15}{18 - 15}(3 - 1)$$

$$y = 2{,}333...$$



INTERPOLAÇÃO: SPLINE

Polinômio cúbico em cada intervalo k, resultando em suavidade:

- Passa pelos pontos das bordas de cada intervalo:

$$P_k(x_k) = y_k \qquad\qquad P_k(x_{k+1}) = y_{k+1}$$

- Continuidade da derivada entre intervalos:

$$P'_k(x_k) = P'_{k-1}(x_k) \qquad\qquad P'_k(x_{k+1}) = P'_{k+1}(x_{k+1})$$

- Inflexão nula nas bordas dos intervalos:

$$P''_k(x_k) = 0 \qquad\qquad P''_k(x_{k+1}) = 0$$

Usando toda a tabela, forma-se um sistema linear. A solução determina os coeficientes de uma cúbica para cada intervalo.

INTERPOLAÇÃO MULTI-DIMENSIONAL

Coeficientes aerodinâmicos dependem de vários parâmetros:

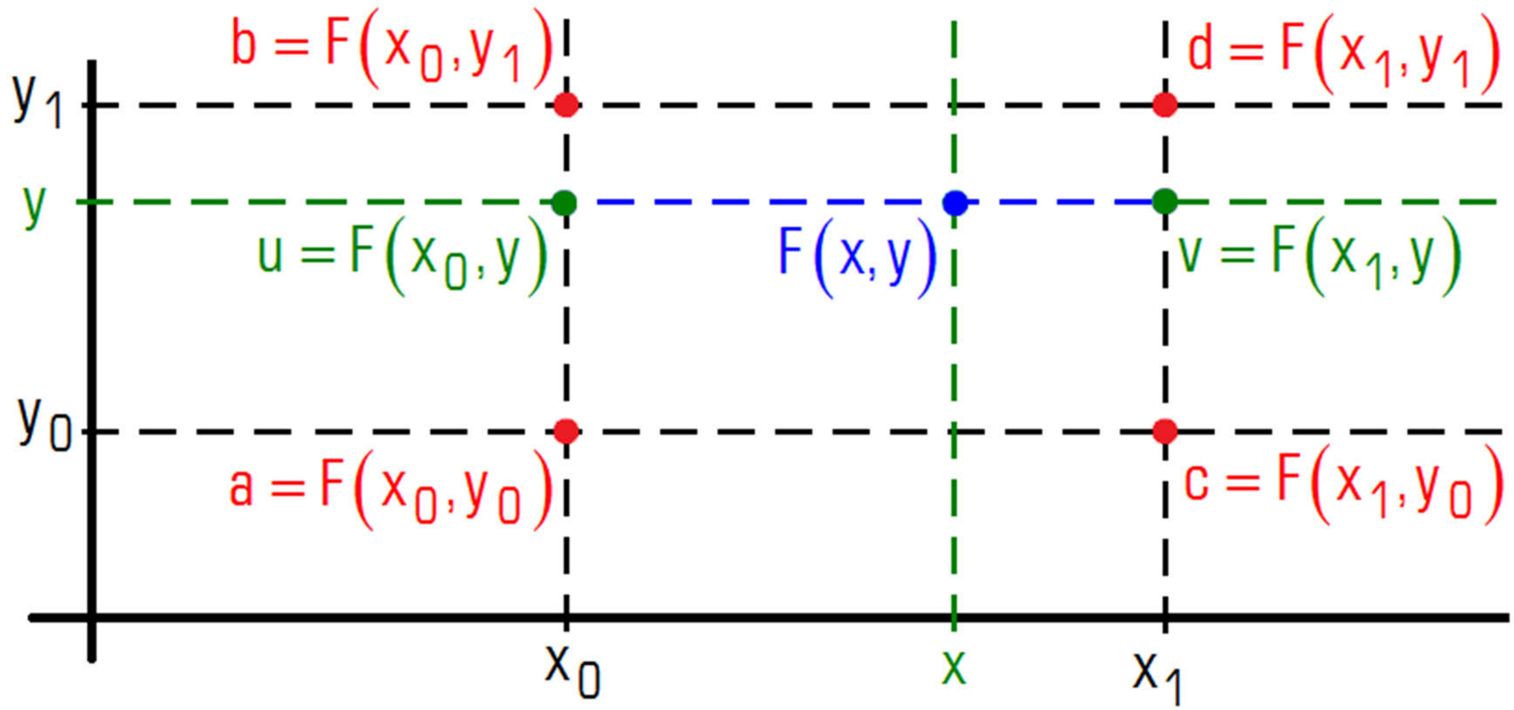
- número de Mach;
- ângulo de ataque;
- posição do CP ou CG;
- interferências de superfícies móveis;
- interferências da presença de pluma de um motor-foguete;
- ...

A estimação do coeficiente num domínio com mais variáveis precisa de interpoladores de dimensão superior:

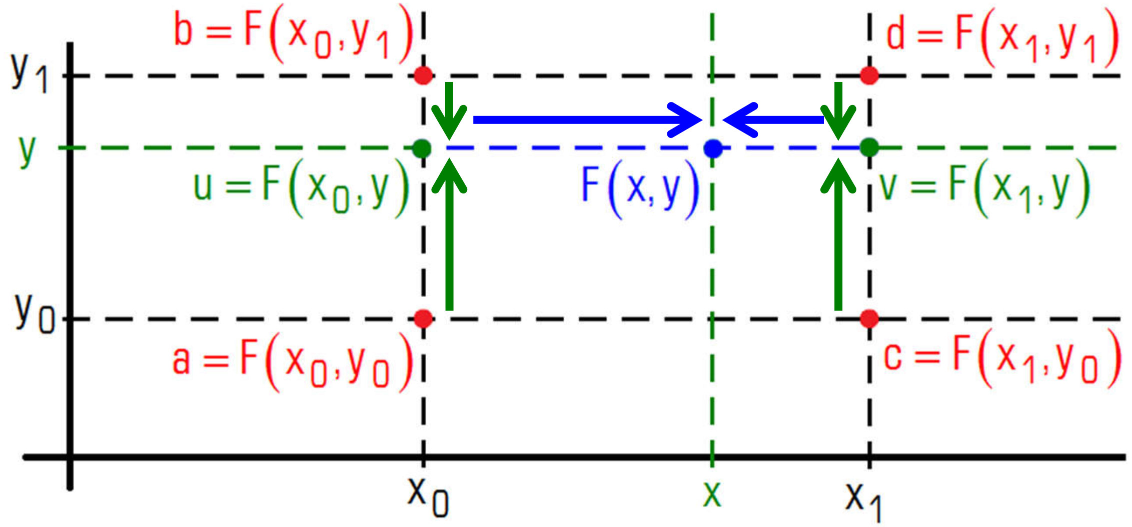
INTERPOLAÇÃO 2D: BILINEAR

Para domínio de duas dimensões F(x,y), primeiramente são encontrados os intervalos no domínio $[x_0; x_1]$ e $[y_0; y_1]$

O interpolador bilinear realiza 3 interpolações lineares.



INTERPOLAÇÃO 2D: BILINEAR



Duas interpolações paralelas e uma interpolação no outro eixo:

$$u = a + \frac{y - y_0}{y_1 - y_0}(b - a) \qquad v = c + \frac{y - y_0}{y_1 - y_0}(d - c)$$

$$F(x,y) = u + \frac{x - x_0}{x_1 - x_0}(v - u)$$

GERADORES DE NÚMEROS ALEATÓRIOS

- Funções **PRNG**: Pseudo-Random Number Generators :
  - Usam cálculos determinísticos.
  - Dependem de um número semente (seed) inicial.
    - Mesma semente → mesma sequência de números.
  - MATLAB: função `rand`
    - Resulta em um valor aleatório entre 0 e 1.
    - Distribuição uniforme: probabilidade constante.
    - Intervalo entre a e b: `x = a + (b-a)*rand;`
    - Setar a semente: `rng(seed);`
    - Semente aleatória: `rng('shuffle');`

GERADORES DE NÚMEROS ALEATÓRIOS

Funções **PRNG**: Pseudo-Random Number Generators:

- Mersenne-Twister (1997):
  - Ótimo, aprovado em vários testes de randomicidade.
  - Amplo uso científico, padrão do MATLAB.
- **LCG** (Linear Congrential Generator):
  - Ruim, mas simples. Padrão Microsoft Excel:

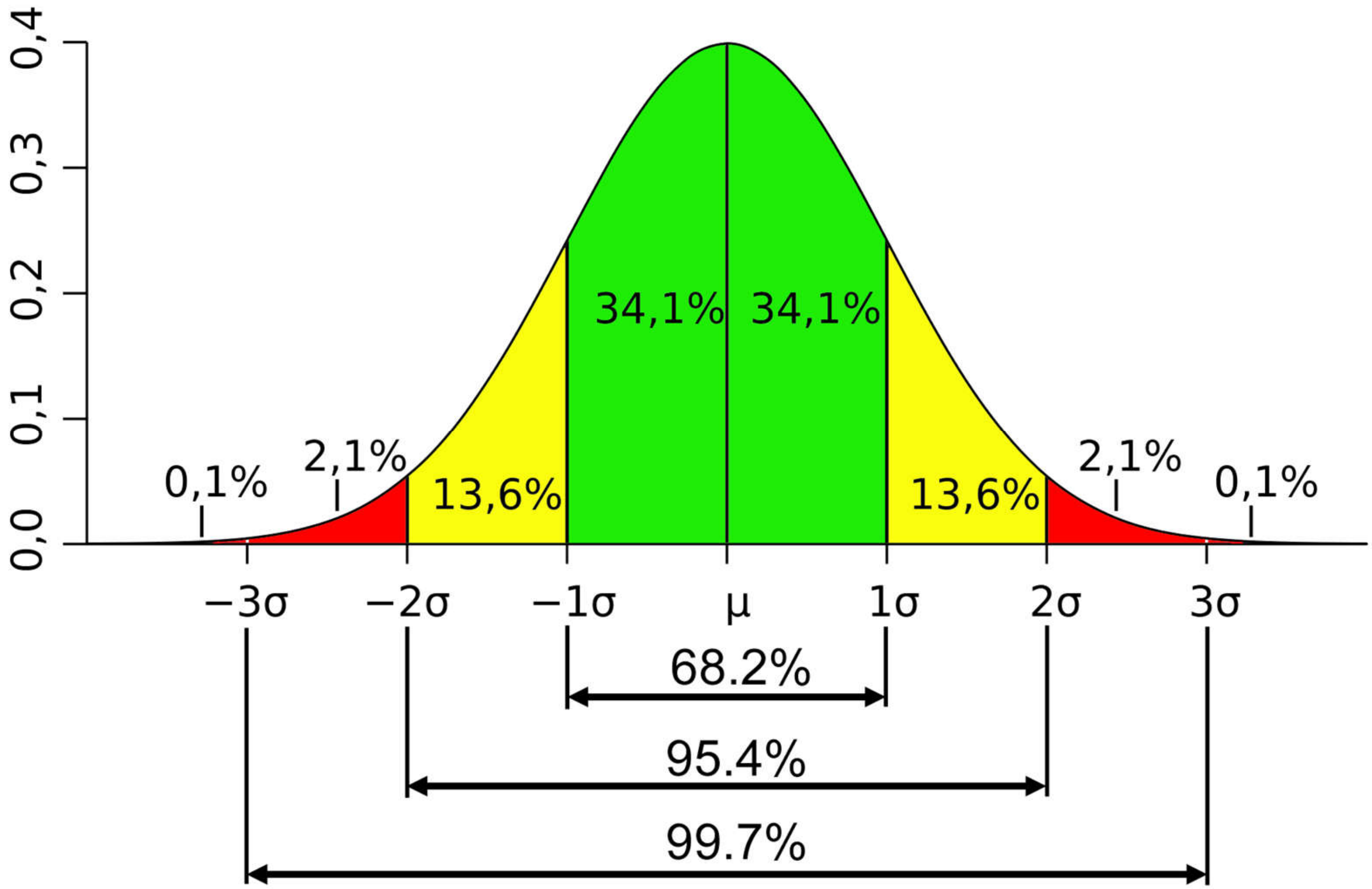
```
x = (9821*x + 0.211327) MOD 1.0
```

- "xorshift*" de Marsaglia. Simples e efetivo:

```
x = x XOR (x << 13);
x = x XOR (x >> 17);
x = x XOR (x <<  5);
```

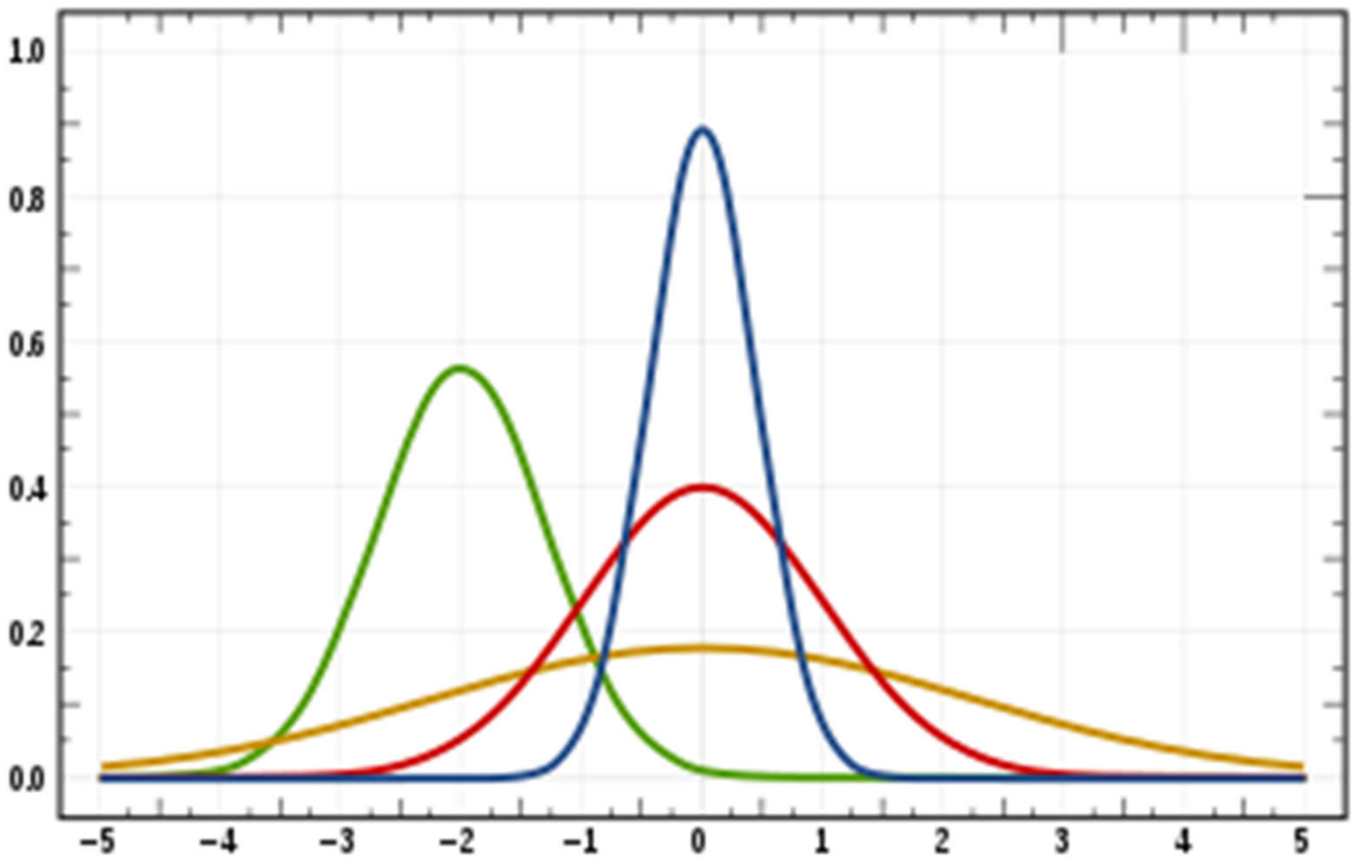
DISTRIBUIÇÃO NORMAL (GAUSSIANA)

- Concentração em torno de um valor médio μ.
- Distribuição com dispersão em duas caudas, desvio-padrão σ.



DISTRIBUIÇÃO NORMAL (GAUSSIANA)

- Concentração em torno de um valor médio μ.
- Distribuição com dispersão em duas caudas, desvio-padrão σ.



$\mu=0 \quad \sigma=0{,}5$

$\mu=0 \quad \sigma=1{,}0$

$\mu=0 \quad \sigma=2{,}0$

$\mu=-2 \quad \sigma=0{,}7$

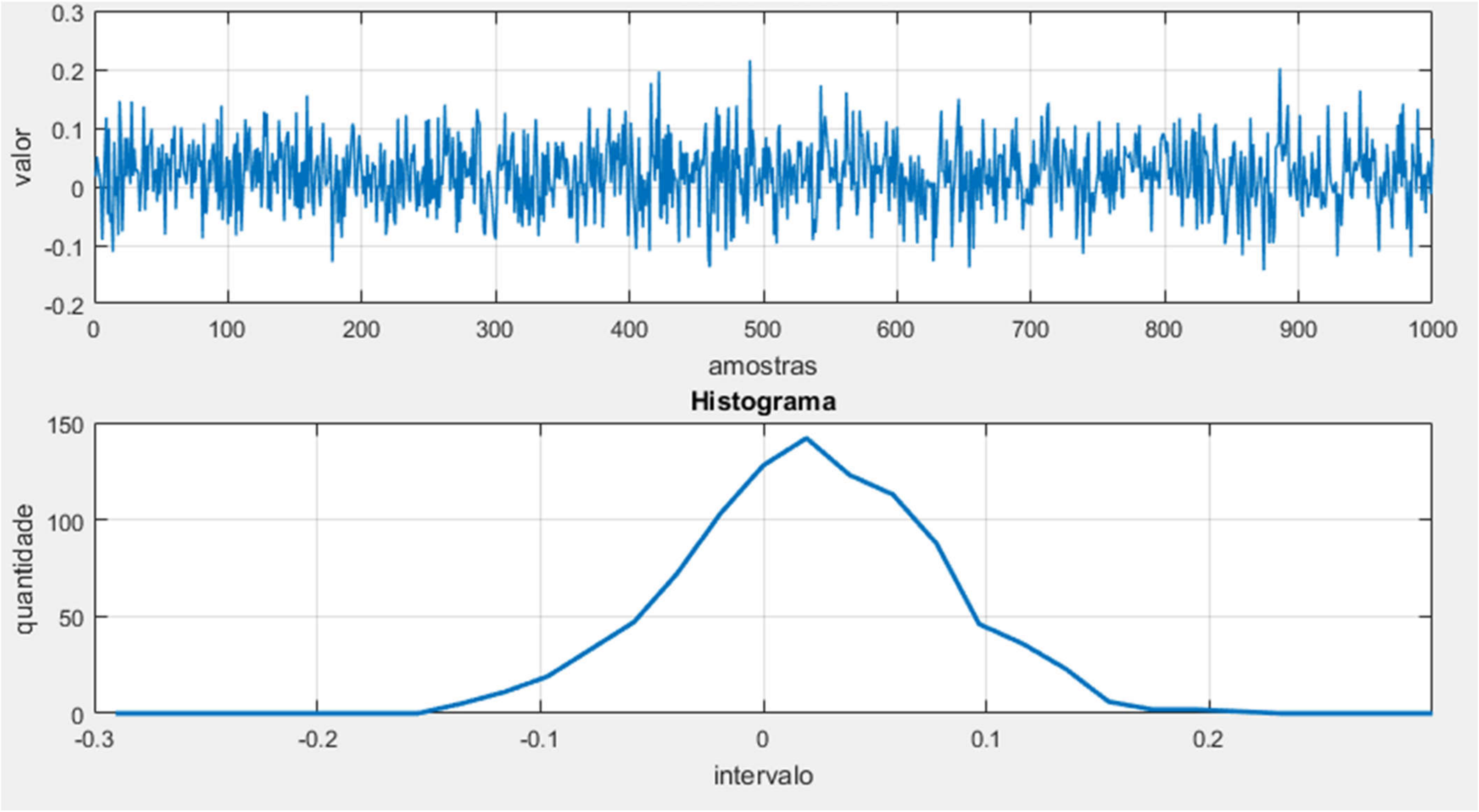
TEOREMA DO LIMITE CENTRAL

Soma de amostras de distribuição uniforme gera a normal:

```
% MATLAB: acumulação de 25 amostras uniformes:
x = 0;
for i = 0:25
    x = x + rand;
end

% faz a média e desloca para o centro:
x = x/25;
x = x - 0.5;

% agora x é uma amostra de uma distribuição normal.
```

TEOREMA DO LIMITE CENTRAL

Soma de amostras de distribuição uniforme gera a normal:



DISTRIBUIÇÃO NORMAL – BOX-MULLER

Par (u,v) de números com distribuição uniforme (intervalo 0...1),

Gera par (x,y) de números com distribuição normal: (m, s)

```
% MATLAB: Normal PRNG Box-Muller Transform
u = rand;
v = rand;
w = sqrt(-2 * log(u));
x = w * cos(2*pi*v);
y = w * sin(2*pi*v);
x = m + s*x;
y = m + s*y;
```

INTEGRADORES NUMÉRICOS: MOTIVAÇÃO

As Leis da física serão descritas no tempo por:

- cinemática (posição e velocidade) (linear e angular), e
- dinâmica (forças e momentos/torques, 2ª Lei de Newton)

São duas Equações Diferenciais Ordinárias no tempo, acopladas:

$$\begin{cases} \frac{d}{dt}\vec{x}(t) = \dot{\vec{x}}(t) = \vec{v}(t) \\ \frac{d}{dt}\vec{v}(t) = \dot{\vec{v}}(t) = a(t) = \frac{1}{m}\sum \vec{F}_{ext}(v,x,t) \end{cases}$$

com condições iniciais definidas (Problema do Valor Inicial)

INTEGRADORES NUMÉRICOS: MOTIVAÇÃO

Se uma solução analítica existir e for encontrada, então a posição e a velocidade dos corpos é obtida para qualquer instante de tempo.

Alternativamente... simula a execução das leis.

As equações diferenciais definem a aceleração de cada corpo.

Integrando a aceleração no tempo, obtém a velocidade

Integrando a velocidade no tempo, obtém a posição.

A integral numérica tem um erro que depende do tamanho de passo de tempo Δt na integração.

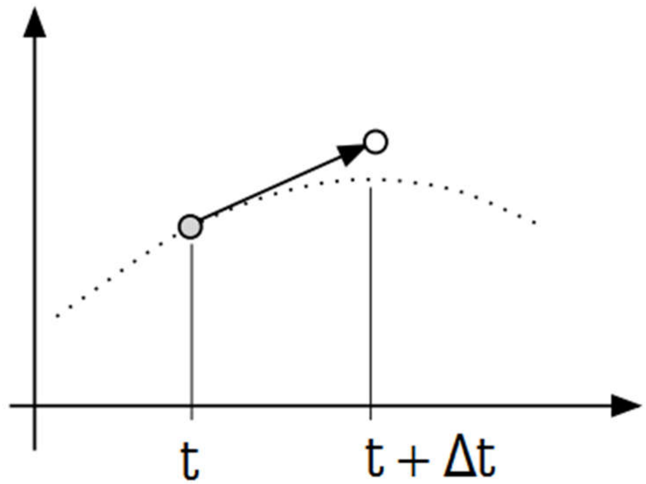
O resultado final (trajetória) deverá ter aderência à realidade.

INTEGRADORES: MÉTODO DE EULER

Integrador de primeira ordem:

(série de Taylor truncada)

$$x(t+\Delta t) = x(t) + \underbrace{\dot{x}(t)\cdot\Delta t}_{\Delta x(t)}$$



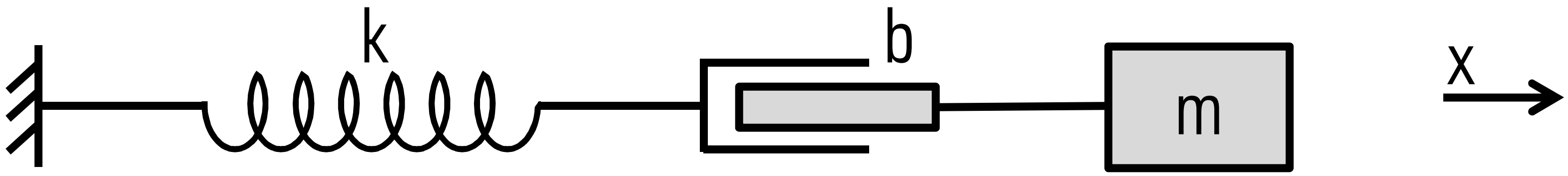
Para trajetória retilínea, o método de Euler é exato.

Método de Euler usa a derivada no início do passo de integração.

Erros aumentam com o tamanho do passo de integração Δt.

Diminuindo o passo, serão necessários mais passos e a precisão numérica pode diminuir: soma do infinitésimo Δx em $x(t)$.

PROBLEMA MASSA-MOLA-AMORTECEDOR



Equação Diferencial: $m a(t) + b v(t) + k x(t) = 0$

Solução: $x(t) = A \cdot \exp(-\zeta \omega_n t) \cdot \cos(\omega_d t + \varphi)$

Coef. de amortecimento: $\zeta = \dfrac{b}{2\sqrt{k m}}$

Freq. natural: $\omega_n = \sqrt{\dfrac{k}{m}}$

Freq. amortecida: $\omega_d = \omega_n \sqrt{1 - \zeta^2}$

Condições iniciais:

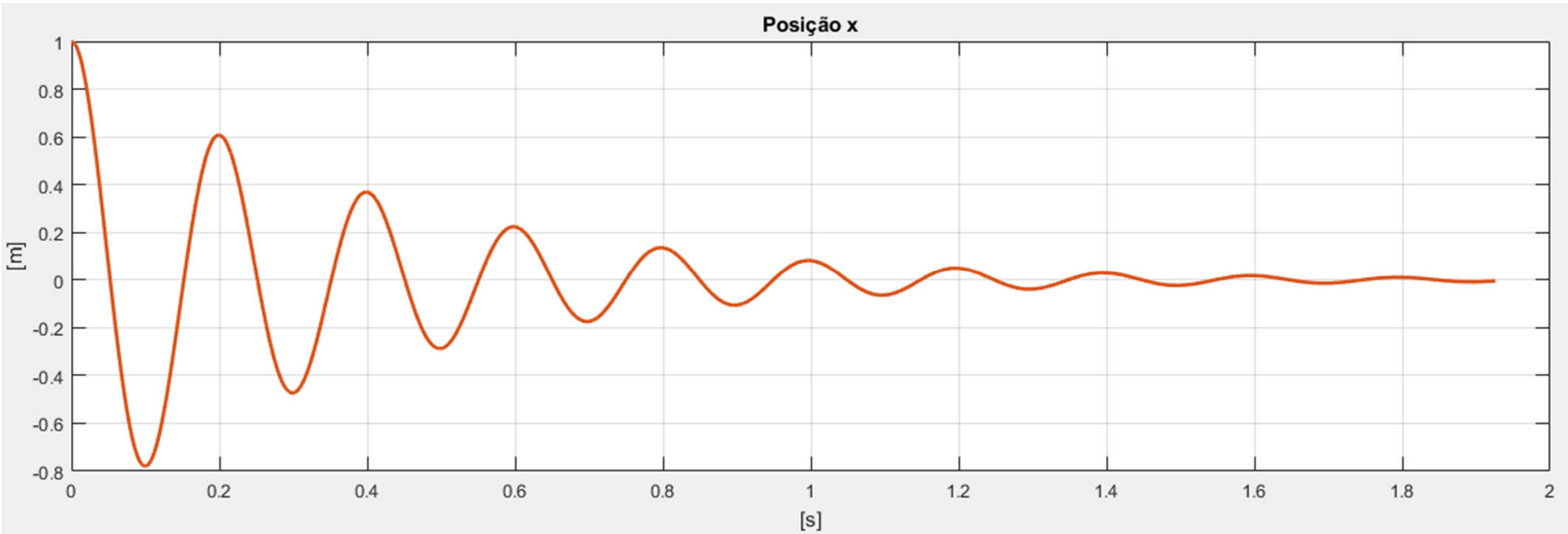
$$\varphi = -\mathrm{atan}\left(\frac{\zeta \omega_n x_0 + v_0}{x_0 \omega_d}\right)$$

$$A = x_0 \sec \varphi$$

$$\boxed{\ddot{x} + 2\zeta \omega_n \dot{x} + \omega_n^2 x = 0}$$

AA-811 – Simulação e Controle de Artefatos Bélicos
Métodos Numéricos Computacionais

PROBLEMA MASSA-MOLA-AMORTECEDOR



$m = 1\,\text{kg}$
$k = 1000\,\text{N/m}$
$b = 5\,\text{kg/s}$

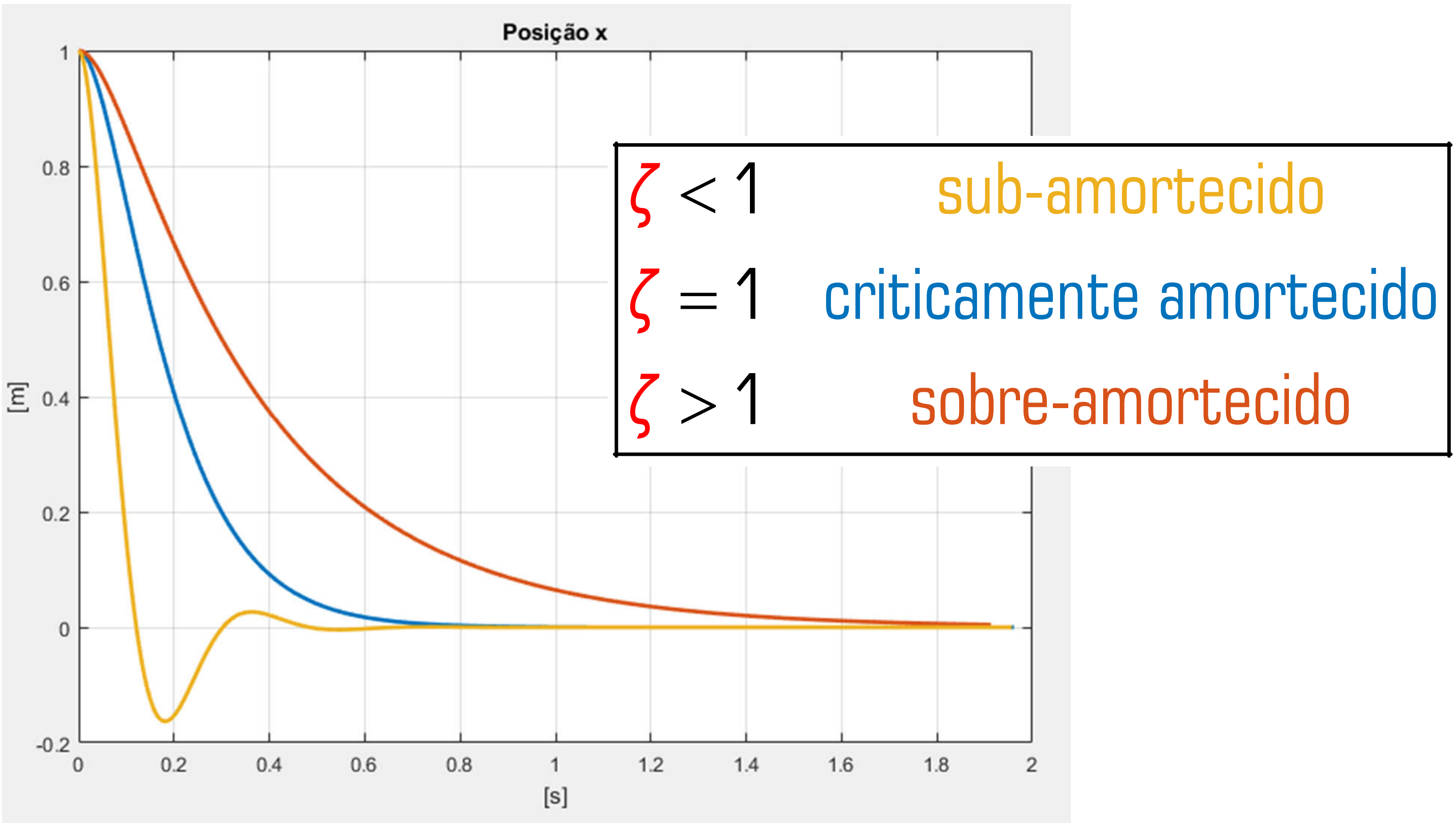
$x_0 = 1\,\text{m}$
$v_0 = 0\,\text{m/s}$

$\zeta \cong 0{,}0791$
$\omega_n \cong 31{,}62\,\text{rad/s}$
$\omega_d \cong 31{,}52\,\text{rad/s}$
$A \cong 1{,}0031\,\text{m}$
$\varphi \cong -0{,}5^{o}$

$(\omega_n \zeta) \cong 0{,}40\,\text{s}$
$2\pi/\omega_d \cong 0{,}199\,\text{s}$

SISTEMA DE SEGUNDA ORDEM

COEFICIENTE DE AMORTECIMENTO



INTEGRADORES: MÉTODO DE EULER

$$\begin{cases} x(t+\Delta t) = x(t) + v(t)\cdot\Delta t \\ v(t+\Delta t) = v(t) + a(t)\cdot\Delta t \end{cases}$$

$$a(t) = -\frac{k}{m}x(t) - \frac{b}{m}v(t)$$

$$v(0) = 0 \qquad x(0) = A$$

PASSO DE INTEGRAÇÃO

Tamanho do passo de integração Δt deve:

- Assegurar a precisão numérica;
- Assegurar a estabilidade numérica;
- Limitar/Controlar o erro cometido em cada passo;
- Capturar descontinuidades de funções;
- Capturar eventos (exemplo: colisão)

Um passo pode ser encurtado para obter o tempo exato do evento.

→ Busca iterativa

Na prática, escolhe-se dois passos, compara-se o erro numérico ou visual, então toma-se uma decisão, com base na demora para simular e segurança dos resultados.

MÉTODO DE EULER: MATLAB

Implementando em script MATLAB, para os valores:

$$m = 2\,\text{kg} \qquad k = 5\,\text{N/m} \qquad b = 0\,\text{kg/s} \qquad A = 1\,\text{m}$$

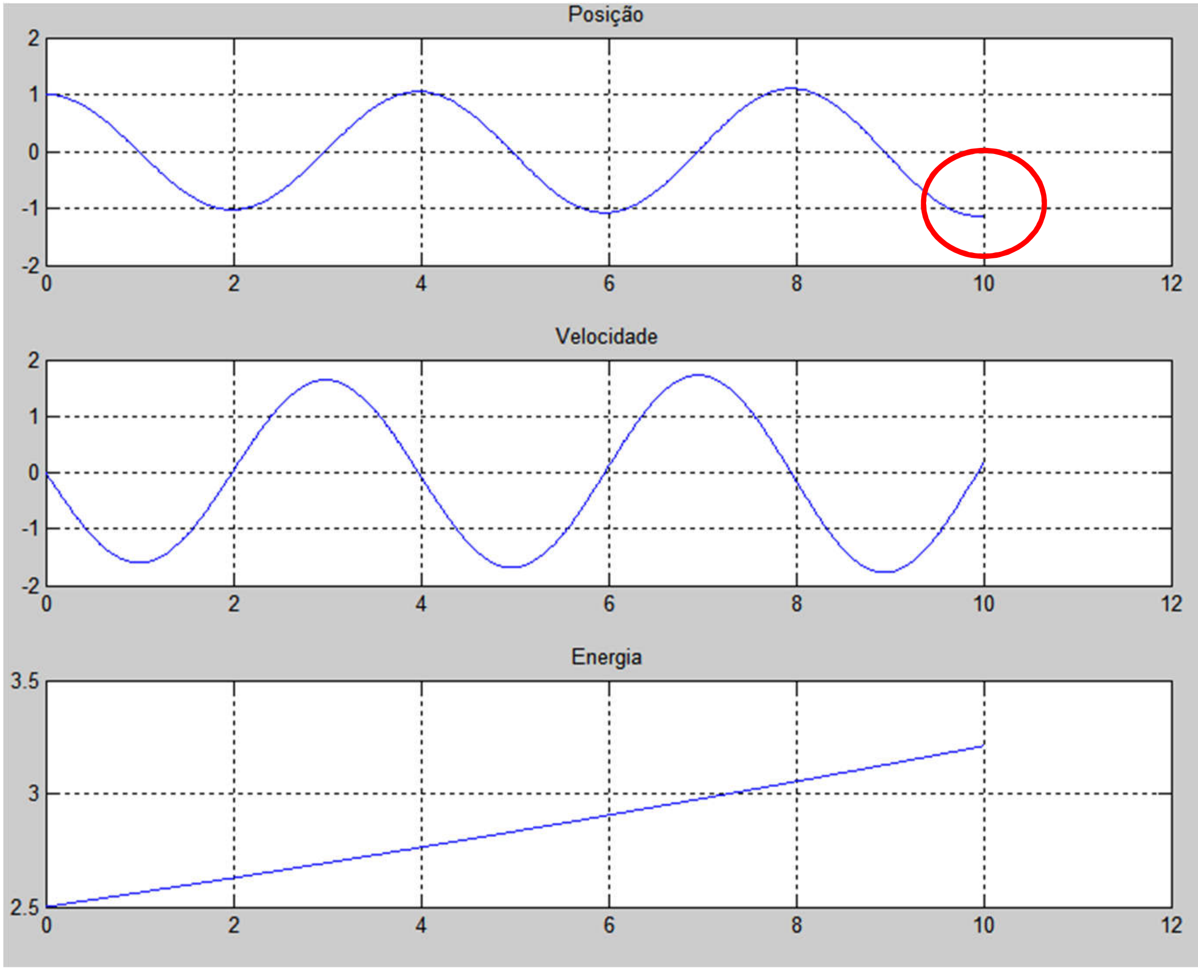
```
% valores dos parâmetros
m = 2;
b = 0;
k = 5;

% valores das condições iniciais
v0 = 0;
x0 = 1;

% Simulação por 10 segundos, passo fixo
t_inicial = 0;
t_final = 10;
dt = 0.01;

% preenchendo condições iniciais
i = 1;
x = [x0];
v = [v0];
t = [t_inicial];
```

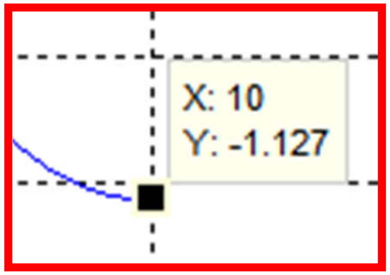
```
while t(i) < t_final

    % cálculo do somatório das forças externas
    Fext = 0;

    % cálculo de aceleração
    a = -k/m * x(i) - b/m * v(i) + Fext/m;

    % integração por Euler obtem a velocidade
    v(i + 1) = v(i) + dt * a;

    % integração por Euler obtem a posição
    x(i + 1) = x(i) + dt * v(i);

    % próximo passo
    t(i + 1) = t(i) + dt;
    i = i + 1;
end
```

```
figure;
subplot(3,1,1); plot(t, x); grid on; title('Posição');
subplot(3,1,2); plot(t, v); grid on; title('Velocidade');
subplot(3,1,3); plot(t, k*x.^2/2 + m*v.^2/2); grid on; title('Energia');
```

MÉTODO DE EULER: MATLAB

Gráficos com um passo de integração $\Delta t = 10$ ms:



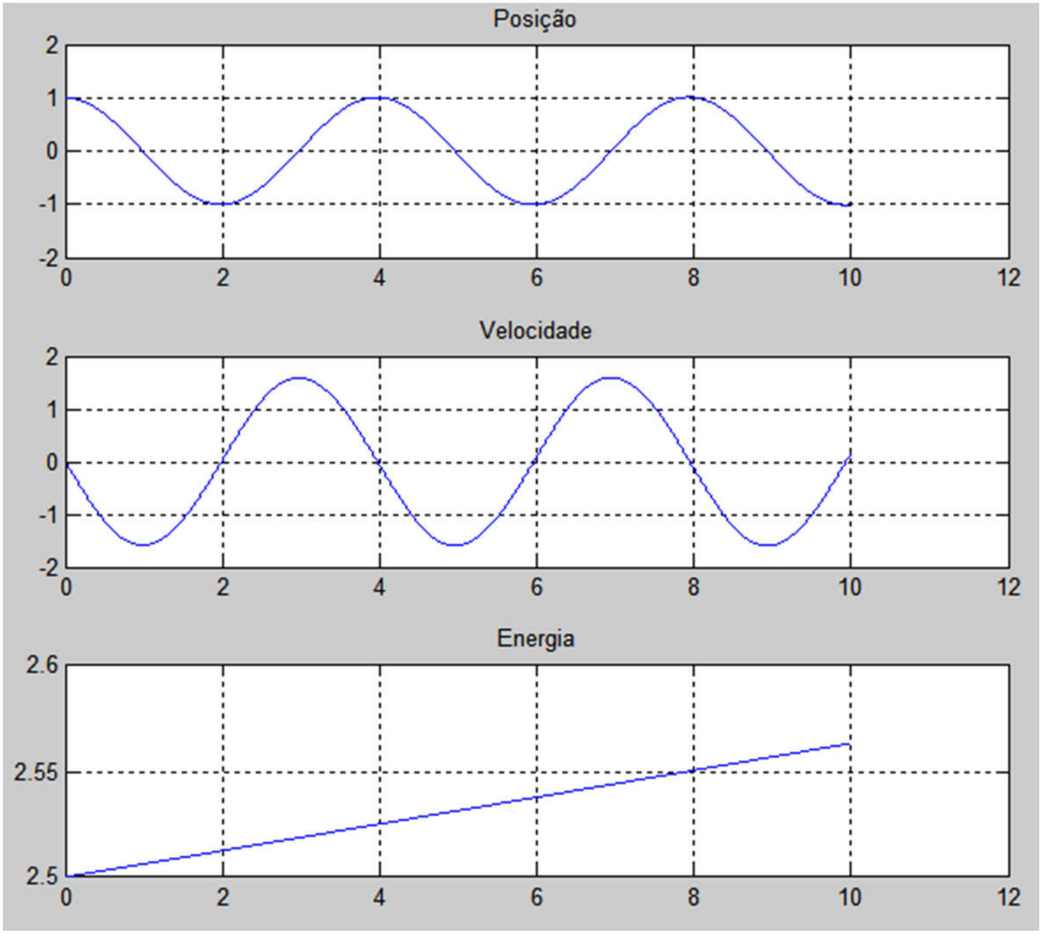


$$T = 2\pi\sqrt{\frac{m}{k}}$$

$$T = 3{,}97\ \text{s}$$

MÉTODO DE EULER: MATLAB

Gráficos com um passo de integração $\Delta t = 1$ ms:



MÉTODO DE EULER: INSTABILIDADE

Não há amortecimento ($b=0$), nem forças externas.

→ Energia total é constante (Movimento Harmônico Simples)

A simulação com o método de Euler realizada está divergente:
Amplitude aumentando com o passar do tempo! (drift)

O passo de integração Δt não faz parte do problema...

...Mas a sua redução melhora a aderência à realidade.

...Porém, não anulará as deficiências deste método: instabilidade.

MÉTODOS SIMPLÉTICOS: VERLET

Integradores simpléticos são usados em sistemas conservativos.

Isso é primordial para a simulação de trajetórias interplanetárias:

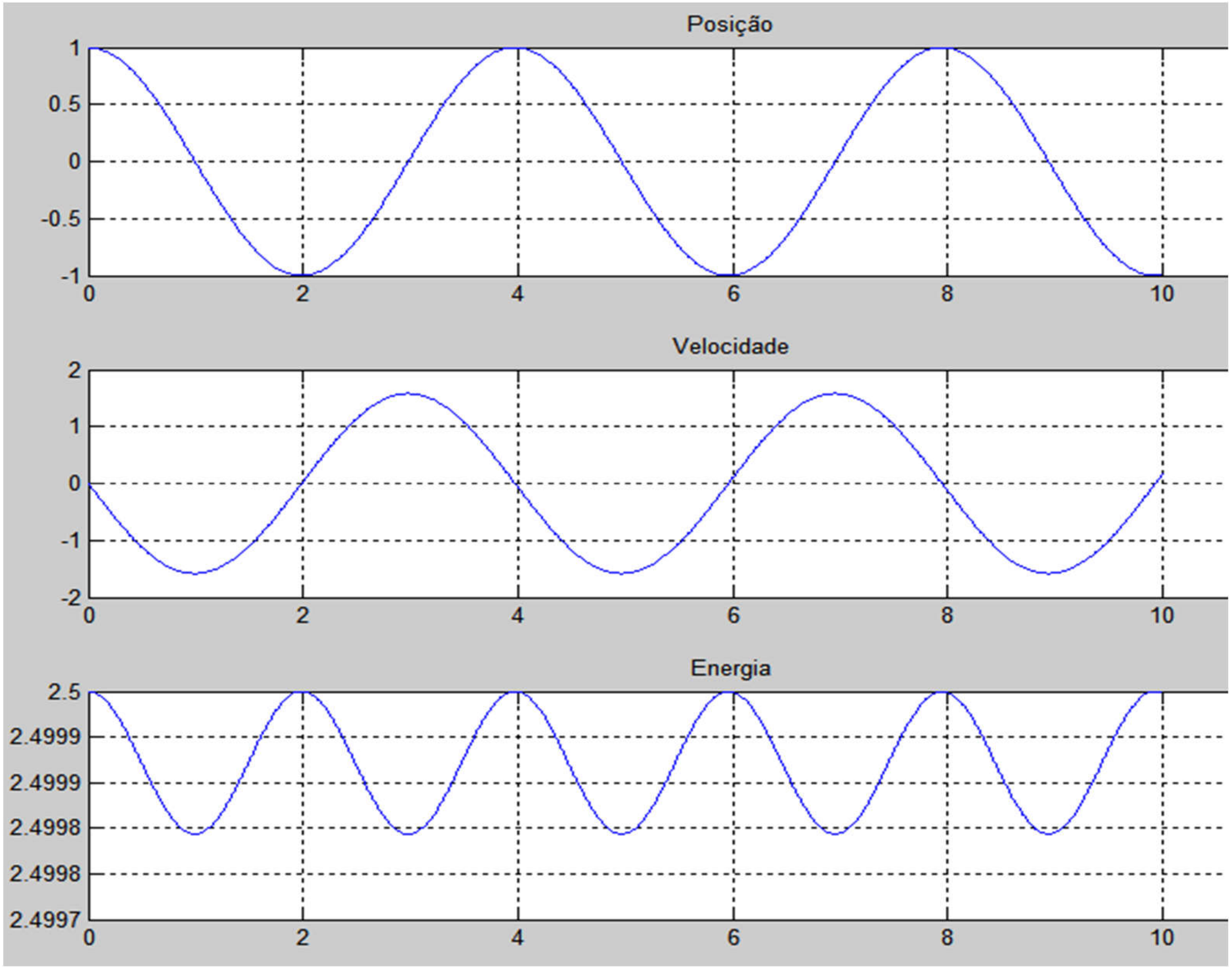
- Conserva a energia (potencial + cinética).
- Longo tempo de simulação.

Método Verlet é de primeira ordem que conserva a energia:

$$x(t+\Delta t) = x(t) + v(t)\cdot\Delta t + \frac{1}{2}a(t)\cdot\Delta t^2$$

$$v(t+\Delta t) = v(t) + \frac{a(t)+a(t+\Delta t)}{2}\cdot\Delta t$$

MÉTODOS SIMPLÉTICOS: VERLET (10ms)



MÉTODOS DE ORDEM SUPERIOR

Métodos simpléticos não podem depender da velocidade.

→ inadequado para simulações de veículos na atmosfera:

→ envolvem forças de arrasto

Métodos de integração de ordem superior diminuem bastante o erro cometido em cada passo.

Método **R**unge-**K**utta de quarta ordem (**RK4**):

Estimativa da derivada em vários pontos e ponderação,

Usa amostras intermediárias no passo de tempo.

MÉTODO RK4 – RUNGE-KUTTA ORDEM 4

Genericamente, o método é descrito a seguir:

$$k_1 = \Delta t \cdot \dot{x}\left(x, t\right)$$

$$k_2 = \Delta t \cdot \dot{x}\left(x + \tfrac{1}{2}k_1,\ t + \tfrac{1}{2}\Delta t\right)$$

$$k_3 = \Delta t \cdot \dot{x}\left(x + \tfrac{1}{2}k_2,\ t + \tfrac{1}{2}\Delta t\right)$$

$$k_4 = \Delta t \cdot \dot{x}\left(x + k_3,\quad t + \Delta t\right)$$

$$x\left(t + \Delta t\right) = x\left(t\right) + \frac{k_1 + 2k_2 + 2k_3 + k_4}{6}$$

MÉTODO RK4: SIMULAÇÃO FÍSICA

Aplicando numa simulação física: $\begin{cases} \dot{x}(t) = v(t) \\ \dot{v}(t) = a(v,x,t) \end{cases}$

$$k_1 = \Delta t \cdot a(v, x, t)$$

$$k_2 = \Delta t \cdot a\left(v + \tfrac{1}{2}k_1,\ x + \tfrac{1}{2}k'_1,\ t + \tfrac{1}{2}\Delta t\right)$$

$$k_3 = \Delta t \cdot a\left(v + \tfrac{1}{2}k_2,\ x + \tfrac{1}{2}k'_2,\ t + \tfrac{1}{2}\Delta t\right)$$

$$k_4 = \Delta t \cdot a\left(v + k_3,\ x + k'_3,\ t + \Delta t\right)$$

$$k = \tfrac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right)$$

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

$$k'_1 = \Delta t \cdot v$$

$$k'_2 = \Delta t \cdot \left(v + \tfrac{1}{2}k_1\right)$$

$$k'_3 = \Delta t \cdot \left(v + \tfrac{1}{2}k_2\right)$$

$$k'_4 = \Delta t \cdot \left(v + k_3\right)$$

$$k' = \tfrac{1}{6}\left(k'_1 + 2k'_2 + 2k'_3 + k'_4\right)$$

$$x(t + \Delta t) = x(t) + k'$$

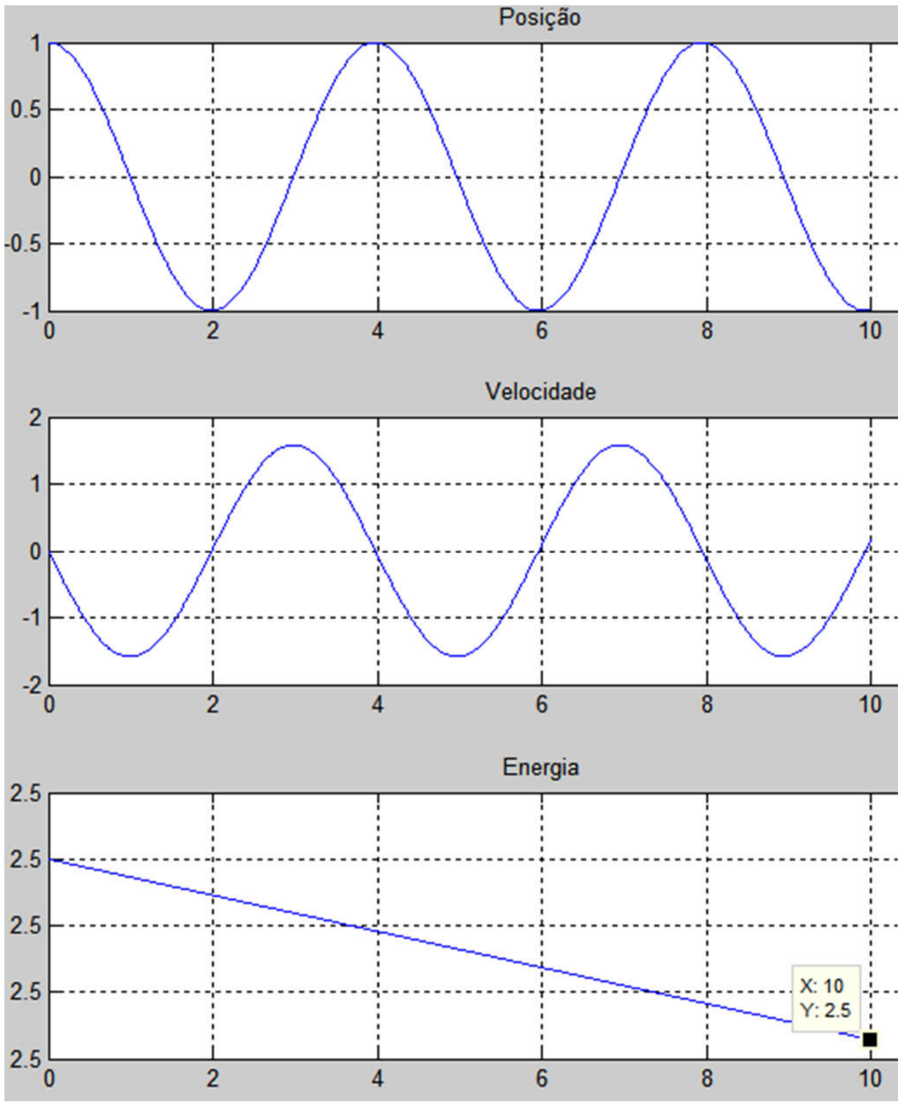
MÉTODO RK4: MATLAB

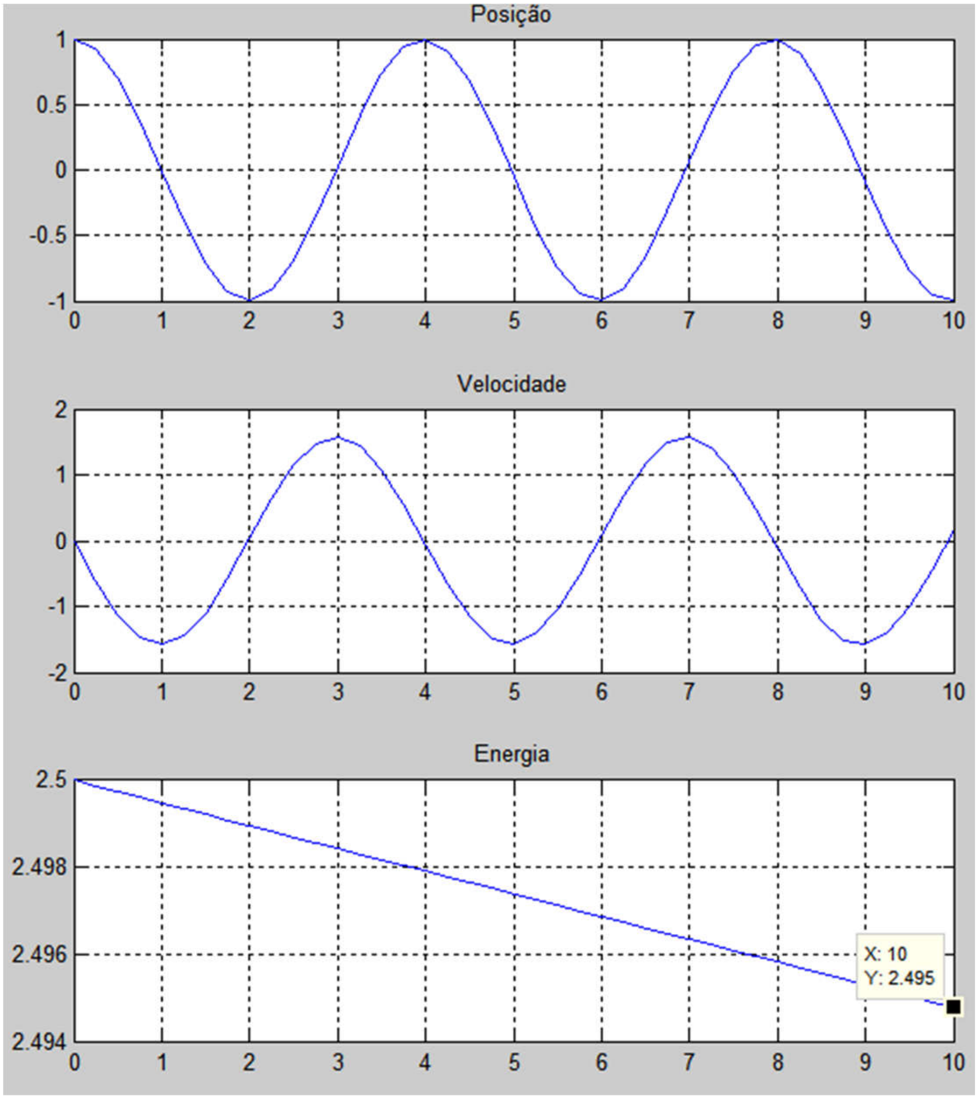
Função cálculo da aceleração:

```
function a = a(v, x, t)
    global m b k;

    % cálculo do somatório das forças externas
    Fext = 0;

    % cálculo de aceleração
    a = -k/m * x -b/m * v + Fext/m;
end
```

```
while t(i) < t_final

    kv1 = dt * a( v(i), x(i), t(i) );
    kp1 = dt * v(i);

    kv2 = dt * a( v(i) + kv1/2, x(i) + kp1/2, t(i) + dt/2 );
    kp2 = dt * (v(i) + kv1/2);

    kv3 = dt * a( v(i) + kv2/2, x(i) + kp2/2, t(i) + dt/2 );
    kp3 = dt * (v(i) + kv2/2);

    kv4 = dt * a( v(i) + kv3, x(i) + kp3, t(i) + dt );
    kp4 = dt * (v(i) + kv3);

    % ponderando velocidades e posições
    kv = (kv1 + 2*kv2 + 2*kv3 + kv4) / 6;
    kp = (kp1 + 2*kp2 + 2*kp3 + kp4) / 6;

    % integrando finalmente
    v(i + 1) = v(i) + kv;
    x(i + 1) = x(i) + kp;

    % próximo passo
    t(i + 1) = t(i) + dt;
    i = i + 1;
end
```

MÉTODO RK4: MATLAB (10ms e 250ms)





PASSO ADAPTATIVO

Escolha do passo <u>automática</u> e <u>dinâmica</u>, controlando:

- Limite superior do erro absoluto;
- Limite superior erro relativo;

A cada passo, os erros são avaliados...

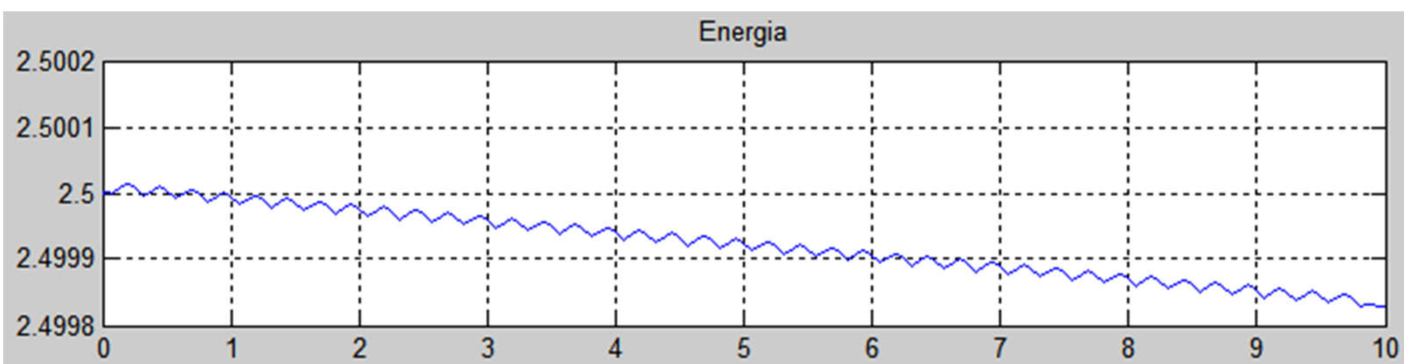
- Se o <u>erro for excessivo</u>,
  - invalida o passo;
  - meia o tamanho passo;
  - refaz o passo;
- Se o <u>erro for muito pequeno</u> (grande margem de segurança):
  - aceita o passo;
  - dobra o tamanho do passo (acelera a simulação);

ODE45: DORMAND-PRICE

Implementado e difundido pela função ode45 do MATLAB

- Runge-Kutta de quinta e quarta ordem (calcula ambas)
- A diferença é uma estimativa do erro → Passo adaptativo
- Realiza 7 cálculos da derivada por passo validado,
  - Mas o último cálculo é igual ao primeiro do próximo passo, então efetivamente são realizados 6 cálculos.

Exemplo para massa-mola (média de 100ms por passo)



MÉTODO ODE45: MATLAB

```
function ode45_test

% valores dos parâmetros
global m b k;
m = 2; b = 0; k = 5;

% valores das condições iniciais e tempo de simulação por 10 s
v0 = 0; x0 = 1;
t_inicial = 0; t_final = 10;

options = odeset('MaxStep', 0.25 );
[t, s] = ode45(@f, [t_inicial t_final], [x0 v0], options);

figure;
subplot(3,1,1); plot(t, s(:,1)); grid on; title('Posição');
subplot(3,1,2); plot(t, s(:,2)); grid on; title('Velocidade');

end
```

```
function dsdt = f(t, s)

    % expande a variável de estados
    x = s(1);
    v = s(2);

    % cálculo do somatório das forças externas
    global m b k;
    Fext = 0;
    % cálculo de aceleração
    a = -k/m * x - b/m * v + Fext/m;

    % preenche a variação da variável de estados
    dsdt = [v; a];
end
```