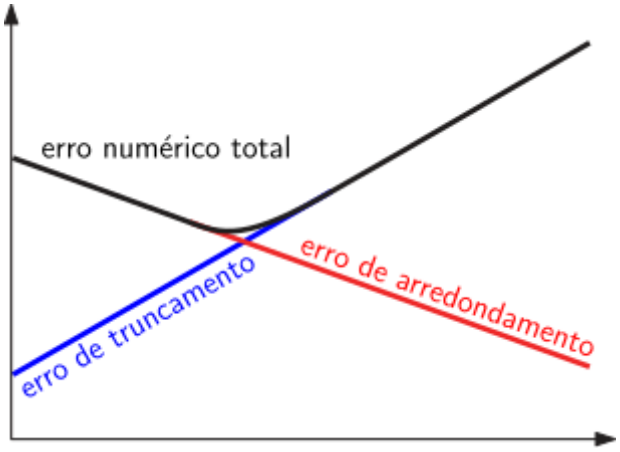
Erros de aproximação



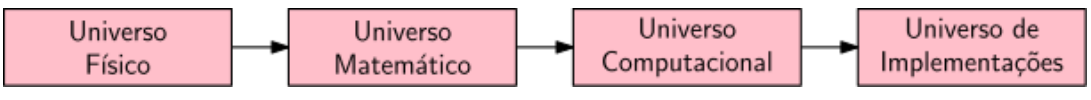
Neste curso, iremos aprender como funcionam diversos métodos numéricos e como podemos aplicá-los para resolver problemas reais. Veremos que dificilmente é possível obter soluções analíticas exatas para os problemas abordados aqui, até mesmo porque muitas vezes não haverá uma única solução. Nestas situações, os métodos numéricos podem nos fornecer soluções alternativas, que chamaremos de soluções *aproximadas*.

Quando escolhemos uma solução aproximada, o ideal é que ela esteja o mais próximo possível daquela dita exata, correta. A distância entre as soluções exata e aproximada é o que chamamos de erro. Os erros presentes em soluções aproximadas podem ser introduzidos pelos próprios métodos numéricos ou por processos de aquisição dos dados por eles manipulados.

1 O paradigma dos quatro universos

Tomaremos emprestado o paradigma dos quatro universos, bastante conhecido no ramo da computação gráfica. Ele no ajudará a compreender com clareza em quais momentos surgirão os famigerados **erros de aproximação**.

Podemos dizer que o processo de modelagem computacional envolve quatro universos distintos de abstração, conforme abaixo.



Para que possamos resolver um problema da vida real utilizando métodos numéricos, precisamos construir uma representação simbólica finita da solução que nos permita simular corretamente o problema original em um computador. Esta representação é o que chamamos de *modelo computacional*, o qual pertence ao *universo computacional*. Mas antes mesmo de chegar nele, ainda precisamos passar por outros dois universos, o *universo físico* e o *universo matemático*, nos quais são estabelecidos os modelos físico (quando for o caso) e matemático.

O *modelo físico* consiste de hipóteses e leis que regem o fenômeno em questão, além de objetos e meios envolvidos. Já o *modelo matemático* consiste de uma descrição simbólica abstrata dos elementos constituintes do modelo físico. Por último, temos o *universo de implementações*, onde são projetados ou escolhidos algoritmos e estruturas de dados, definidos sobre um determinado modelo de computação, necessários para construirmos uma *implementação* do modelo computacional em um computador.

Exemplo. (Pesagem de objetos) Consideremos o problema de modelarmos o peso de objetos. Segundo a lei da gravitação universal, se dois corpos possuem massa, existirá uma força atrativa proporcional ao produto de suas massas e inversamente proporcional ao quadrado da distância entre os corpos. A *força peso* é definida como uma força gravitacional atrativa que a Terra exerce sobre objetos próximos de sua superfície, e isto completa a definição de nosso modelo físico.

Agora, suponha que a Terra possui raio constante R e massa M e considere um corpo pontual de massa m localizado a uma distância $h \ll R$ da superfície da Terra. Pela nosso modelo físico, temos que existe uma força peso P obedecendo à relação:

$$P \propto \frac{mM}{(R+h)^2}. \tag{1}$$

Como $h \ll R$, é razoável assumirmos $(R+h) \approx R$, levando à seguinte simplificação:

$$P \propto \frac{mM}{R^2}. \tag{2}$$

Introduzindo uma constante de correção G, conhecida como constante gravitacional, obtemos o seguinte modelo matemático para a força peso:

$$P = m\frac{GM}{R^2}, \tag{3}$$

ou $P = mg$, onde $g = GM/R^2$ é conhecida como a *aceleração da gravidade*.

Observe que P é um número real e, sendo assim, há infinitos valores que não possuem uma representação finita, como no caso dos irracionais, por exemplo. Um modelo computacional (que precisa ser finito) para números reais bastante conhecido é a *representação em ponto flutuante*. Um sistema numérico (finito) decimal de ponto flutuante é definido, basicamente, por três parâmetros inteiros, a precisão $t \geq 2$ e dois valores extremos de expoente $e_{\min} < e_{\max}$. Com isso, qualquer número x deste sistema possuirá a forma:

$$x = \pm m \times 10^e, \tag{4}$$

onde:

- e é um inteiro tal que $e_{\min} \leq e \leq e_{\max}$, denominado o *expoente* de x;
- $m = (d_0,d_1 \ldots d_{t-1})_{10}$, com $d_i \in \{0,1,\ldots,9\}$ e $0 \leq m < 10$, é sua *mantissa*.

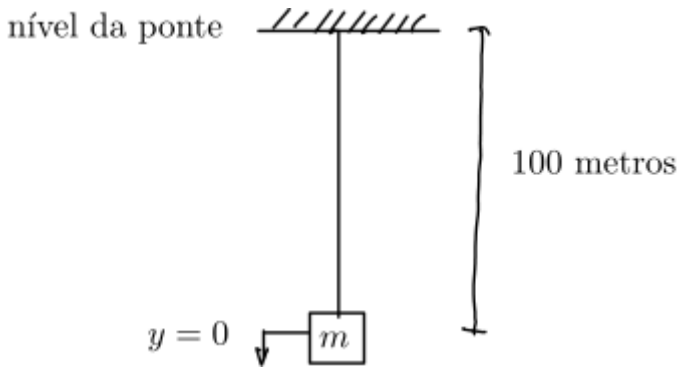
Uma implementação do modelo de ponto flutuante que acabamos de descrever pode ser obtida com qualquer um dos formatos numéricos definidos no padrão IEEE 754. Falaremos mais sobre este padrão em breve.

Agora, consideremos um exemplo um pouco mais sofisticado. Desejamos modelar o comportamento do deslocamento vertical de uma pessoa durante um salto de bungee jump realizado de cima de uma ponte.

Exemplo. (Salto de *bungee jump*) Suponha que uma pessoa irá saltar de *bungee jump* de cima de uma ponte, utilizando um cabo com 100 metros de comprimento. Desejamos prever o deslocamento vertical que o saltador descreverá ao longo de todo o salto.

Por simplicidade, assuma que o saltador descreve um movimento vertical perfeito, sem deslocamentos laterais, sujeito à resistência do ar. Suponha desprezível o peso do cabo. Completando o modelo físico, assumiremos válidas as leis de Newton e a lei de Hooke. As leis de Newton serão úteis para determinarmos as forças atuantes no modelo e a de Hooke nos ajudará a compreender como o cabo se comporta quando for esticado.

Representaremos o saltador por um corpo pontual de massa m localizado no ponto de fixação do cabo em sua cintura. Seja $y(t)$ sua posição no instante $t \geq 0$, y variando verticalmente, com sentido positivo apontando para baixo. Assuma que a ponte está localizada no nível $y = -100$ m. Portanto, em $y = 0$ o cabo estará esticado, mas em repouso. A figura abaixo ilustra este modelo.



Assim que o salto é iniciado, existirão apenas a força da gravidade e a resistência do ar atuando sobre o saltador. Assumiremos a força de amortecimento devido à resistência do ar proporcional à velocidade do saltador. Quando ele ultrapassar o ponto de repouso do cabo, entrará em cena uma força de reação. Pela lei de Hooke, a força no cabo $F(y)$ pode ser representada por:

$$F(y) = \begin{cases} ky, & \text{se } y < -200 \\ 0, & \text{se } -200 \leq y < 0 \\ -ky, & \text{se } y \geq 0 \end{cases} \tag{5}$$

onde k é o *módulo de elasticidade* do cabo, também conhecido como *módulo de Young*.

Agora, a segunda lei de Newton nos fornece a equação:

$$my'' = mg + F(y) - cy', \tag{6}$$

onde m é a massa do saltador, g a aceleração da gravidade e c o coeficiente de amortecimento do ar.

No início do salto, o saltador não tomará impulso. Com isso, sua velocidade inicial será nula, ou seja, $y'(0) = 0$. Acabamos assim de montar o seguinte problema de valor inicial:

$$y'' = g + \frac{1}{m}\left(F(y) - cy'\right), \qquad t > 0 \tag{7}$$
$$y(0) = -100 \tag{8}$$
$$y'(0) = 0 \tag{9}$$

Por conveniência, reescrevemos a equação diferencial ordinária (EDO) acima como um sistema de EDOs de primeira ordem. Isto facilitará a aplicação do método de Euler. Definindo $x_1 = y$ e $x_1' = x_2$, obtemos:

$$x_1' = x_2 \tag{10}$$
$$x_2' = g + \frac{1}{m}\left(F(x_1) - cx_1\right) \tag{11}$$
$$x_1(0) = -100 \tag{12}$$
$$x_2(0) = 0 \tag{13}$$

Seja $t_0 = 0$ e $t_k = t_0 + kh$, com $h > 0$ e $k \in \mathbb{N}$. O método de Euler consiste em usar uma aproximação linear para calcular valores aproximados $x_1^{(k)}$ e $x_2^{(k)}$ para $x_1(t_k)$ e $x_2(t_k)$, respectivamente. As equações do método são:

$$x_1^{(k+1)} = x_1^{(k)} + x_1'(t_k)h \tag{14}$$
$$x_2^{(k+1)} = x_2^{(k)} + x_2'(t_k)h \tag{15}$$
$$\tag{16}$$

o que nos fornece:

$$x_1^{(k+1)} = x_1^{(k)} + hx_2^{(k)} \tag{17}$$
$$x_2^{(k+1)} = x_2^{(k)} + h\left[g + \frac{1}{m}\left(F(x_1^{(k)}) - cx_1^{(k)}\right)\right] \tag{18}$$
$$x_1^{(0)} = -100 \tag{19}$$
$$x_2^{(0)} = 0 \tag{20}$$

Uma implementação sequencial do método de Euler necessitará novamente de um formato do padrão IEEE 754. Nenhum algoritmo ou estrutura de dados especial é requerida.

2 Erro computacional

Na passagem entre cada um dos universos da modelagem, podem acontecer perdas de informação. Neste curso, estaremos particularmente interessados nas perdas que ocorrem entre os modelos matemático e o computacional (*erros de truncamento*), e entre este último e sua implementação (*erros de arredondamento*). Este interesse particular é justificado pela dificuldade que existe em controlar os erros que surgem nas outras interfaces entre universos.

O **erro computacional** ou **erro numérico total** é a soma do *erro de trucamento total*, surgido na passagem entre os modelos matemático e computacional, e o *erro de arredondamento total*, que acontece na implementação do modelo computacional.

O *erro de truncamento total* é a diferença entre o valor calculado por uma implementação usando aritmética exata e a solução exata do modelo matemático. Já o *erro de arredondamento total* pode ser definido como sendo a diferença entre os resultados de uma implementação usando aritmética com precisão finita (de ponto flutuante) e outra usando aritmética exata. A seguir, iremos ilustrar que, em algumas situações, é possível obter um limite superior para o erro de truncamento. A análise de erros de arredondamento será abordada em aulas posteriores.

Exemplo. (Aproximação de funções) Suponha que $f:[a,b]\to\mathbb{R}$ possui suas derivadas contínuas até a ordem $n+1$ em $[a,b]$, para algum inteiro $n\geq 0$. Pelo teorema de Taylor (com resto de Lagrange), se $x,x_0\in[a,b]$, então existe $\xi(x)$ entre x e x_0 tal que:

$$f(x)=P_n(x)+R_n(x), \tag{21}$$

onde:

$$P_n(x)=f(x_0)+f'(x_0)(x-x_0)+\frac{f''(x_0)}{2!}(x-x_0)^2 \tag{22}$$

$$+\frac{f'''(x_0)}{3!}(x-x_0)^3+\cdots+\frac{f^{(n)}(x_0)}{n!}(x-x_0)^n \tag{23}$$

é o *polinômio de Taylor de f de ordem n definido em torno de x_0* e

$$R_n(x)=\frac{f^{(n+1)}(\xi(x))}{(n+1)!}(x-x_0)^{n+1} \tag{24}$$

é o resto de Lagrange ou o erro de truncamento associado à aproximação P_n de f.

Considere o problema de aproximarmos o valor de $f(x)=\cos(x)$ em $[0,1]$ por meio de seu polinômio de Taylor de grau 2 expandido em torno de $x_0=0$. Com efeito, temos

$$P_2(x)=f(x_0)+f'(x_0)(x-x_0)+\frac{f''(x_0)}{2!}(x-x_0)^2 \tag{25}$$

$$R_2(x)=\frac{f'''(\xi(x))}{3!}(x-x_0)^3 \tag{26}$$

Como sabemos que

$$f'(x) = -\operatorname{sen} x, \qquad f''(x) = -\cos x, \qquad f'''(x) = \operatorname{sen} x,$$

substituindo essas expressões e o valor $x_0 = 0$ na expansão de Taylor, obtemos:

$$P_2(x) = f(0) + f'(0)(x-0) + \frac{f''(0)}{2!}(x-0)^2 \tag{27}$$

$$= \cos(0) - \operatorname{sen}(0)(x-0) + \frac{-\cos(0)}{2!}(x-0)^2 \tag{28}$$

$$= 1 - \frac{1}{2}x^2 \tag{29}$$

e (30)

$$R_2(x) = \frac{f'''(\xi(x))}{3!}(x-0)^3 \tag{31}$$

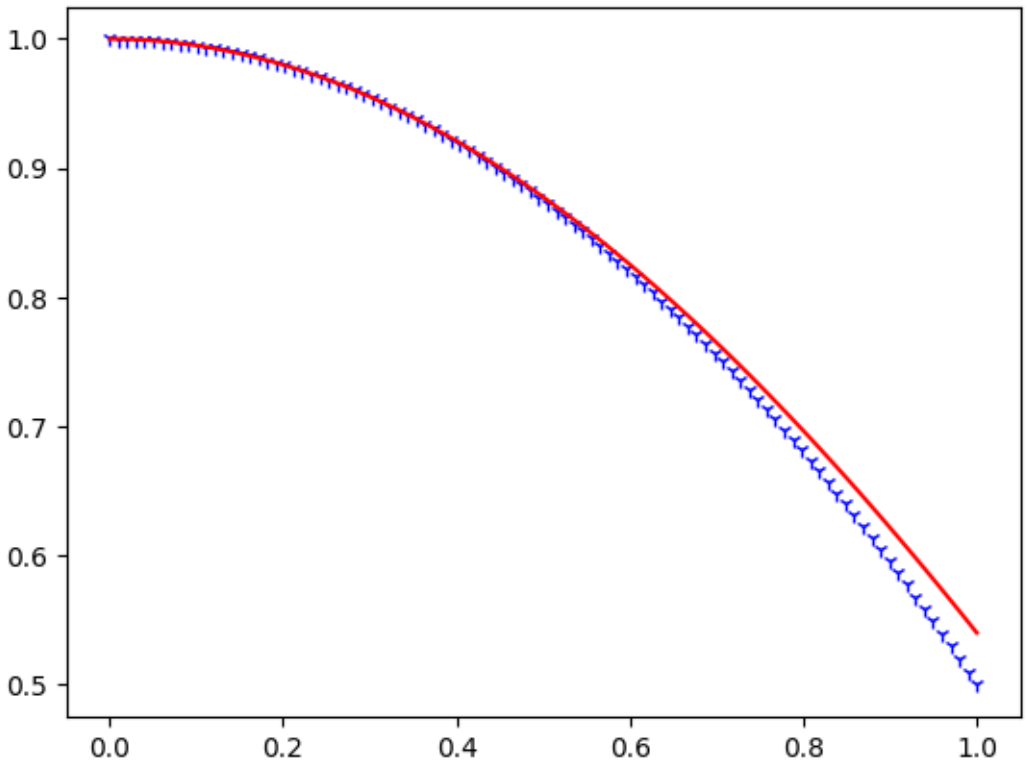
$$= \frac{\operatorname{sen}(\xi(x))}{3!}(x-0)^3 \tag{32}$$

$$= \frac{1}{6}x^3 \operatorname{sen}(\xi(x)) \tag{33}$$

Aqui, P_2 servirá de modelo computacional para f, com erro de truncamento $P_2(x) - f(x) = -R_2(x)$. Vejamos seus gráficos:

[31]:
```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt

fig, ax = plt.subplots() # cria uma figura e com um único sistema de␣
 ↪eixos
ax.set_aspect('equal') # fixa a razão de aspecto do gráfico
plt.close() # evita que o gráfico seja exibido antes de desenhado

def f(x):
    return np.cos(x)

def P2(x):
    return 1 - 0.5*(x**2)

x = np.linspace(0.0, 1.0, 100)
plt.plot(x,P2(x),'b1')
plt.plot(x,f(x),'r-')
```

[31]:
```
[<matplotlib.lines.Line2D at 0x7f55cc1bf460>]
```



Observe que quanto mais próximo x for de $x_0 = 0$, melhor será a aproximação de f por P_2. Por outro lado, à medida que afastamos x de x_0, maior será o erro associado a P_2. Embora esta análise visual seja importante, ela tem pouco valor matemático, pois carece de dados quantitativos.

Por definição, o **erro de truncamento** será:

$$P_2(x) - f(x) = -R_2(x) = -\frac{1}{6}x^3 \operatorname{sen}(\xi(x)). \tag{34}$$

Como não conhecemos o ξ, buscaremos um limitante para $|-R_2(x)|$. Tomando o módulo em ambos os lados da expressão anterior, obtemos

$$|P_2(x) - f(x)| = \frac{1}{6}\left|x^3\right| |\operatorname{sen}(\xi(x))|, \tag{35}$$

Para $0 \leq x \leq 1$, sabemos que $\left|x^3\right| \leq 1$ e $|\operatorname{sen}(\xi(x))| \leq \operatorname{sen}(1)$. Logo,

$$|P_2(x) - f(x)| \leq \frac{1}{6}\operatorname{sen}(1) \approx 0{,}140245164. \tag{36}$$

Podemos então nos perguntar: > Qual o grau do polinômio de Taylor que a gente deveria utilizar para se ter uma aproximação de $f(x)$ com erro de truncamento de no máximo 10^{-8}?

Com base no que foi dito anteriormente, o erro de truncamento *absoluto* correspondente ao polinômio de Taylor de grau $n-1$ é:

$$|P_{n-1}(x) - f(x)| = \frac{1}{n!}\left|f^{(n)}(\xi)\right| |x - x_0|^n \tag{37}$$

$$\leq \frac{1}{n!} \max_{a\leq t\leq b} |t - x_0|^n \max_{a\leq t\leq b} \left|f^{(n)}(t)\right|. \tag{38}$$

Para o exemplo em estudo, tem-se

$$|P_{n-1}(x) - f(x)| \leq \frac{1}{n!} \max_{0\leq t\leq 1} |t|^n \max_{0\leq t\leq 1} \left|f^{(n)}(t)\right| = \frac{1}{n!}. \tag{39}$$

Usando indução matemática, é possível mostrar que $n! > 2^n$, para $n \geq 4$, ou seja:

$$|P_{n-1}(x) - f(x)| \leq \frac{1}{n!} < \frac{1}{2^n}.$$

Para asseguramos o erro de truncamento limitado por 10^{-8}, resolvemos:

$$\frac{1}{2^n} \leq 10^{-8},$$

cuja solução é $n \geq 8\log_2(10) \approx 26{,}5754248$. Como n precisa ser inteiro, é suficiente tomarmos $n = 27$ e o grau do polinômio será $n - 1 = 26$.

Por causa do comportamento cíclico das derivadas do cosseno, seu polinômio de Taylor de grau 26 em torno $x_0 = 0$ assume a forma:

$$P_{26}(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \cdots + \frac{x^{24}}{24!} - \frac{x^{26}}{26!} \tag{40}$$

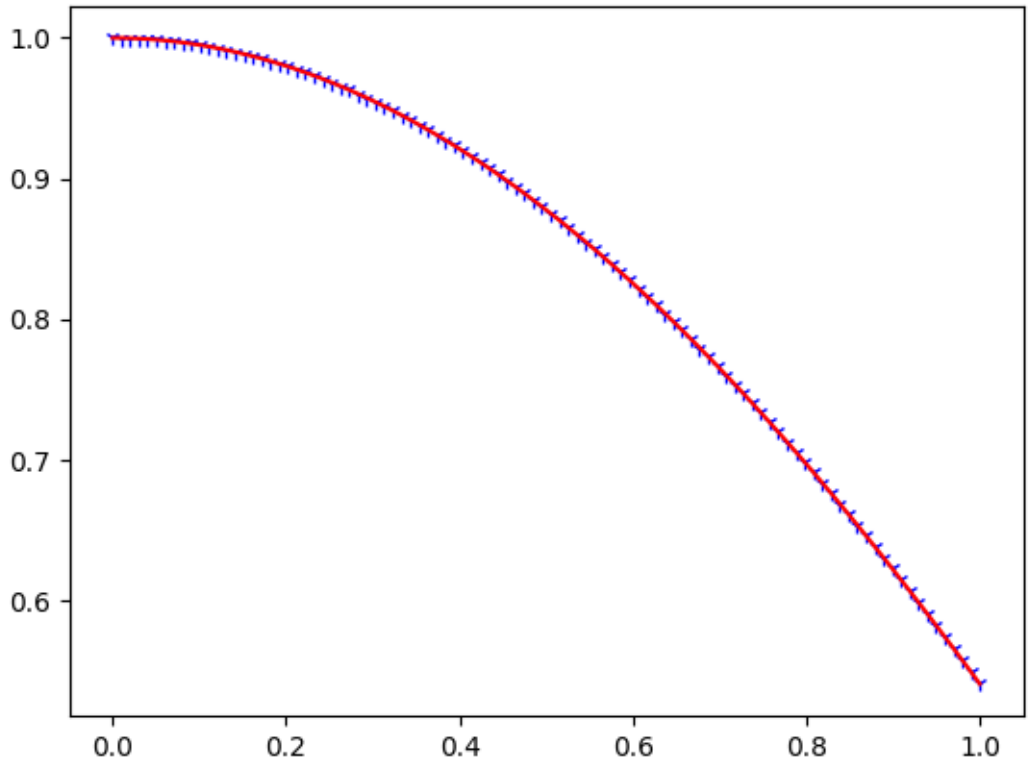
$$= \sum_{i=0}^{13} \frac{(-1)^i}{(2i)!} x^{2i} \tag{41}$$

Agora, vejamos como fica o gráfico dessa aproximação:

[30]:
```
fig, ax = plt.subplots() # cria uma figura e com um único sistema de
 ↪eixos
ax.set_aspect('equal') # fixa a razão de aspecto do gráfico
plt.close() # evita que o gráfico seja exibido antes de desenhado

def P26(x):
    soma = 0.0
    for i in range(0,14):
        sinal = (-1)**i
        num   = x**(2*i)
        den   = np.math.factorial(2*i)
        soma  = soma + sinal*num/den
```

```
    return soma

x = np.linspace(0.0, 1.0, 100)
plt.plot(x,P26(x),'b1')
plt.plot(x,f(x),'r-')
```

[30]: [<matplotlib.lines.Line2D at 0x7f55cc287f70>]



Podemos aumentar a acurácia do modelo computacional do exemplo anterior utilizando um polinômio de Taylor de ordem maior do que 26. Isto pode até fazer com que o erro de truncamento diminua, mas temos que ter cautela, pois serão necessárias mais operações aritméticas e isto pode levar ao aumento do erro de arredondamento total. Na prática, a precisão dos sistemas numéricos encontrados nos computadores é normalmente suficiente para manter o erro de arredondamento dentro de limites aceitáveis, deixando o erro numérico total dominado pelo erro de truncamento.

Prever teoricamente o erro computacional é uma tarefa bastante complicada, até mesmo para modelos razoavelmente pequenos. Geralmente, realizamos testes com a implementação e verificamos se os resultados obtidos atendem restrições e equações presentes no modelo computacional. Uma outra técnica usada para avaliar a qualidade dos resultados de uma implementação é conhecida como *análise de sensibilidade*. Ela consiste em verificar como e quanto a solução calculada é afetada quando alteramos os dados de entrada do problema. Isto pode ser feito experimentalmente, executando o programa com diversas combinações de entrada, ou teoricamente, como veremos a seguir.

3 Saiba mais

- Estas anotações refletem principalmente o conteúdo da Seção 1.2 do livro do professor Michael Heath, *Scientific Computing: An Introductory Survey*, 2ª edição revisada, SIAM, 2018.
- O paradigma dos quatro universos foi adaptado de Jonas Gomes e Luiz Velho, Abstraction paradigms for computer graphics, *The Visual Computer*, v. 11, p. 227-239, 1995.
- Por curiosidade, dê uma olhadinha no verbete do Wikipedia sobre o padrão IEEE 754. A versão em inglês está bem mais completa e possui diversas referências interessantes.
- Há diversos trabalhos sobre a modelagem de saltos de bungee jump na internet. Um que me chamou a atenção pela apresentação concisa e elegante foi o de dois alunos de graduação, Eric Moon e John Thompson, apresentado ao final de uma disciplina sobre equações diferenciais do *College of the Redwoods*.
- Definimos o *erro computacional* ou *erro numérico total* baseando-se na naquela presente no livro do Chapra et al., *Métodos Numéricos para Engenharia*, 7ª edição, McGraw-Hill Brasil, 2016. É um excelente livro para buscar ideias de projetos aplicados. Há alguns exemplares dele na biblioteca do campus Juazeiro do Norte.

UFCA | Centro de Ciências e Tecnologia