Introdução ao Cálculo Numérico

Números inteiros; números reais em ponto-fixo e em ponto-flutuante

Instituto de MATEMÁTICA E ESTATÍSTICA UFRGS

Prof. Rudnei Dias da Cunha

- Precisão x Exatidão
  - **precisão**
    sf (*lat praecisione*) **3** Qualidade daquilo que é exato; exatidão. **6** Exatidão rigorosa em cálculos e ciências. **7** Exatidão, regularidade na execução. *Precisão de um número, Inform:* número de dígitos de um número... [Michaelis]
  - **exatidão**
    sf (*exato+suf lat itudine*) **1** Caráter ou qualidade de exato. **2** Rigor na determinação de medida, peso, valor etc.; precisão. **3** Atenção minuciosa no cálculo; correção. [Michaelis]

- Precisão x Exatidão
  - Em Cálculo Numérico:
  - **Precisão**: quantidade de dígitos com os quais uma determinada operação aritmética é efetuada $\rightarrow$ refere-se ao HARDWARE
  - **Exatidão**: qualidade de um algoritmo numérico e sua implementação $\rightarrow$ refere-se ao SOFTWARE

Representação de números inteiros

- Representação de números inteiros
  - Um número inteiro x é representado no computador na forma de **sinal-e-magnitude** através dos n dígitos

$$x = \pm b_{n-1} b_{n-2} \ldots b_1 b_0$$

que compõem sua representação binária

  - O valor de x em base 10 é calculado através do somatório

$$(x)_{10} = \pm \sum_{i=0}^{n-1} \left(b_i \times 2^i\right)$$

- Representação de números inteiros
  - Evidentemente, um número inteiro será armazenado em uma palavra de m bits, logo, $n \leq m$
  - Tipicamente, o bit mais significativo é reservado para o sinal, e os restantes armazenam os dígitos binários do número; esse formato é conhecido como *sinal-e-magnitude*
  - Outra possibilidade é utilizar armazenamento por *complemento-de-2*

- Representação de números inteiros
  - No formato sinal-e-magnitude, o intervalo de representação dos números inteiros é
    $$[\,-(2^{n-1}-1)\,;2^{n-1}-1\,]$$
  - Por exemplo, para uma variável declarada como INTEGER(KIND=4) em Fortran 90, temos *n = 32*, de onde o intervalo de representação para tal variável é
    $$[\,-2\,147\,483\,647\,;\ 2\,147\,483\,647\,]$$
  - Observe que nesse formato, existem duas representações para o número 0: *+0* e *-0*
  - Não é um problema, pois o hardware toma conta dessa situação e desconsidera o sinal

- Representação de números inteiros
  - No formato complemento-de-2, o bit de sinal é desconsiderado, e números negativos são armazenados negando-se todos os seus dígitos binários e somando 1 ao dígito menos significativo
  - Nesse formato, o intervalo de representação é
$$[\,-2^{n-1}; 2^{n-1}-1\,]$$
  - Existe apenas uma representação para o 0 nesse formato

- Representação de números inteiros
  - De qualquer forma, aplicações como criptografia (usada para fins militares, corporativos e de transações bancárias) exigem a manipulação de números inteiros com, no mínimo, 128 dígitos binários, a fim de tornar (muito) difícil a "quebra" da informação codificada
  - Isso faz com que aqueles dois tipos de representação de números inteiros sejam inadequados para tais fins
  - Uma das maneiras de se resolver esse problema é simular em software uma aritmética de inteiros em multi-precisão; existem pacotes como o MPFUN90 (www.netlib.org) que implementam tal aritmética

Representação de números reais na forma racional

- Representação de números reais na forma racional
  - Uma possível representação de números reais seria na forma *racional*, em que cada número é representado por uma razão entre dois números inteiros

$$x = \pm \frac{a}{b} \in \mathbb{R}$$

onde $a, b \in \mathbb{N}$.

  - Tal forma exige que os números inteiros sejam representados com um grande número de dígitos, a fim de que a forma racional possa ser transformada – se necessário – para um número decimal com precisão adequada.

- Representação de números reais na forma racional
  - Por exemplo, o número

$$0{,}000327739000384884000027738$$

pode ser representado como

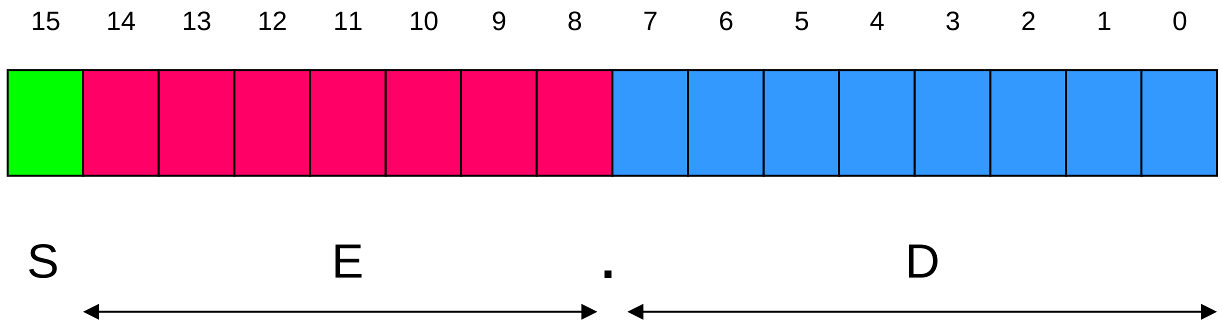
$$\frac{16386950019244200013869}{50000000000000000000000000000}.$$

  - Observe que tanto o numerador como o denominador são números que excedem ao maior número inteiro representável em 64 bits!
  - Isso leva novamente ao uso de *aritmética de inteiros em multiprecisão!*
  - A forma racional é empregada em vários programas conhecidos como *sistemas de computação algébrica*, como Maple e Mathematica.

Representação de números reais em ponto-fixo

- Representação de números reais em ponto-fixo
  - Outra forma de representar números reais no computador é similar à usada no formato sinal-e-magnitude para inteiros, a qual é chamada de *formato de ponto-fixo*
  - Nesse formato, considerando-se uma palavra de m bits, especificam-se dois campos, chamados de E e D, que representam os bits à esquerda e à direita do ponto binário (o qual é implícito, portanto)

- Representação de números reais em ponto-fixo
  - Um dos bits da palavra é reservado para o sinal, de tal forma que $|E|+|D|+1 \leq m$, onde |E| e |D| representam o número de bits destinados aos campos E e D



    - *Campos de uma palavra de ponto-fixo de m=16 bits: S (sinal,1 bit), E (7 bits) e D (8 bits); o ponto decimal está implicitamente entre os bits nº 7 e 8*

- Representação de números reais em ponto-fixo
  - Podemos caracterizar um sistema de ponto-fixo através de algumas constantes, como:
    - A precisão ($\boldsymbol{p}$)
    - O menor e o maior números representáveis (**MINR** e **MAXR**)
    - A separação entre dois números representáveis consecutivos (**ULP**)
    - O "épsilon da máquina", ε, o menor número representável positivo para o qual

$$\text{fixo}(1+\varepsilon) \neq 1$$

onde $\text{fixo}(x)$ é a representação em ponto-fixo de x

- Representação de números reais em ponto-fixo
  - Como exemplo, vamos considerar uma palavra de 8 bits, com |E|=3 e |D|=4
    - $p = |E|+|D| = 7$
    - $MINR = (000.0001)_2 = 2^{-|D|} = 0.0625$
    - $MAXR = (111.1111)_2 = (2^{|E|}-1)+(1- 2^{-|D|})=7.9375$
    - $ULP = (000.0001)_2 = 2^{-|D|} = 0.0625$
    - $\varepsilon = ULP = 0.0625$

  - Agora, podemos fazer as seguintes perguntas:
    1. **O conjunto de números representáveis nesse sistema é enumerável?**
    2. **Se sim, quais são esses números e onde eles se situam na reta dos reais?**

- Representação de números reais em ponto-fixo
    - A resposta para a pergunta nº 1 é *sim* – o que nos leva a dizer que a quantidade de números em ponto-fixo é menor (na prática, muito menor) do que a de números reais (que são infinitos)
    - Para a pergunta nº 2, vamos escrever esses números em ponto-fixo (considerando apenas os valores positivos):

- Representação de números reais em ponto-fixo

$(000.0000)_2 = (0)_{10}$

$(000.0001)_2 = (1x2^{-4}) = 0.0625$

$(000.0010)_2 = (1x2^{-3}) = 0.125$

$(000.0011)_2 = (1x2^{-3} +1x2^{-4}) = 0.1875$

$(000.0100)_2 = (1x2^{-2}) = 0.25$

$(000.0101)_2 = (1x2^{-2} +1x2^{-4}) = 0.3125$

$(000.0111)_2 = (1x2^{-2} +1x2^{-3} +1x2^{-4}) = 0.4375$

...

- Representação de números reais em ponto-fixo

$(000.0111)_2 = (1x2^{-2} + 1x2^{-3} + 1x2^{-4}) = 0.4375$

$(000.1000)_2 = (1x2^{-1}) = 0.5$

$(000.1001)_2 = (1x2^{-1} + 1x2^{-4}) = 0.5625$

$(000.1010)_2 = (1x2^{-1} + 1x2^{-3}) = 0.625$

...

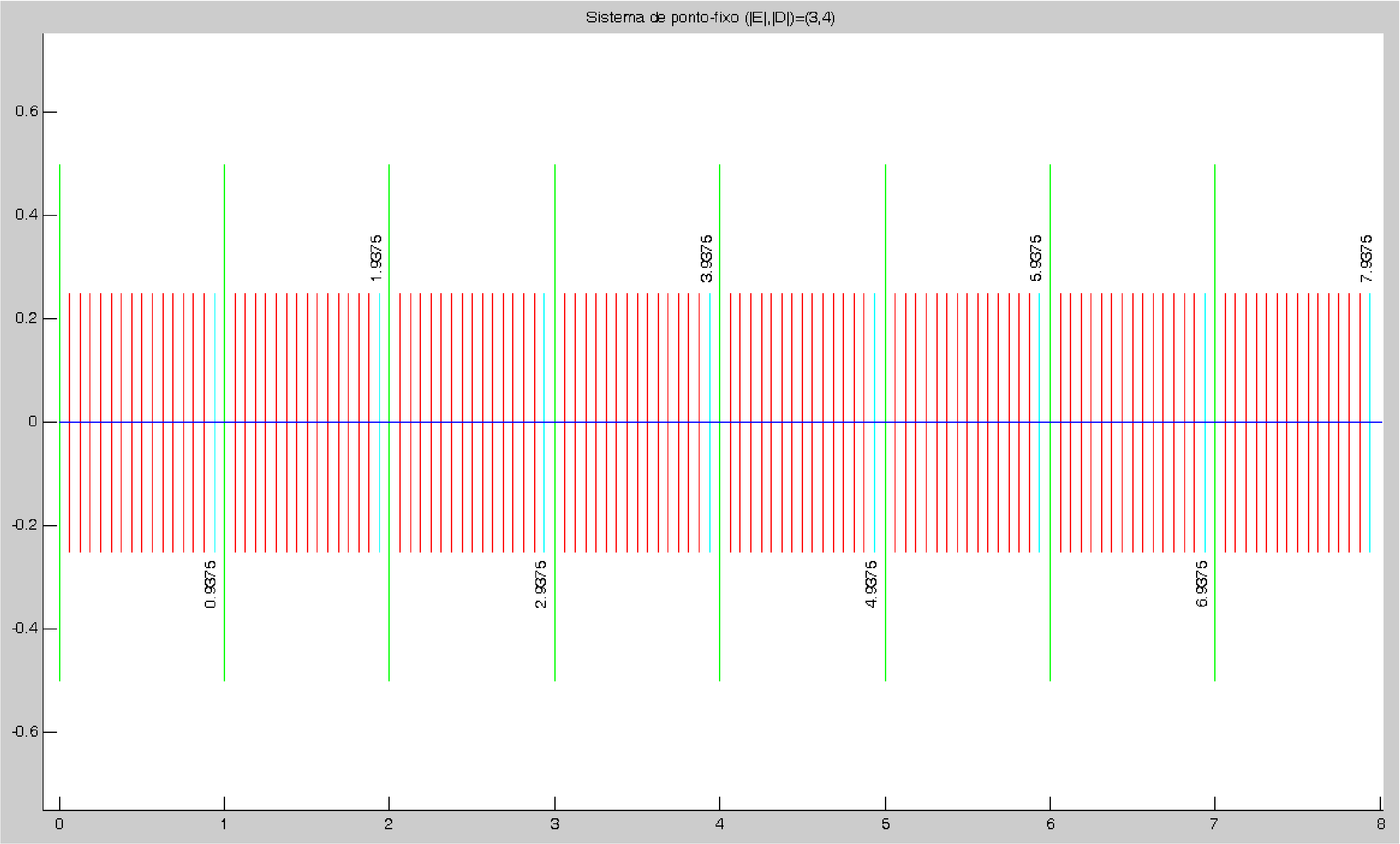
$(000.1111)_2 = (1x2^{-1} + 1x2^{-2} + 1x2^{-3} + 1x2^{-4}) = 0.9375$

$(001.0000)_2 = (1x2^{0}) = 1.0$

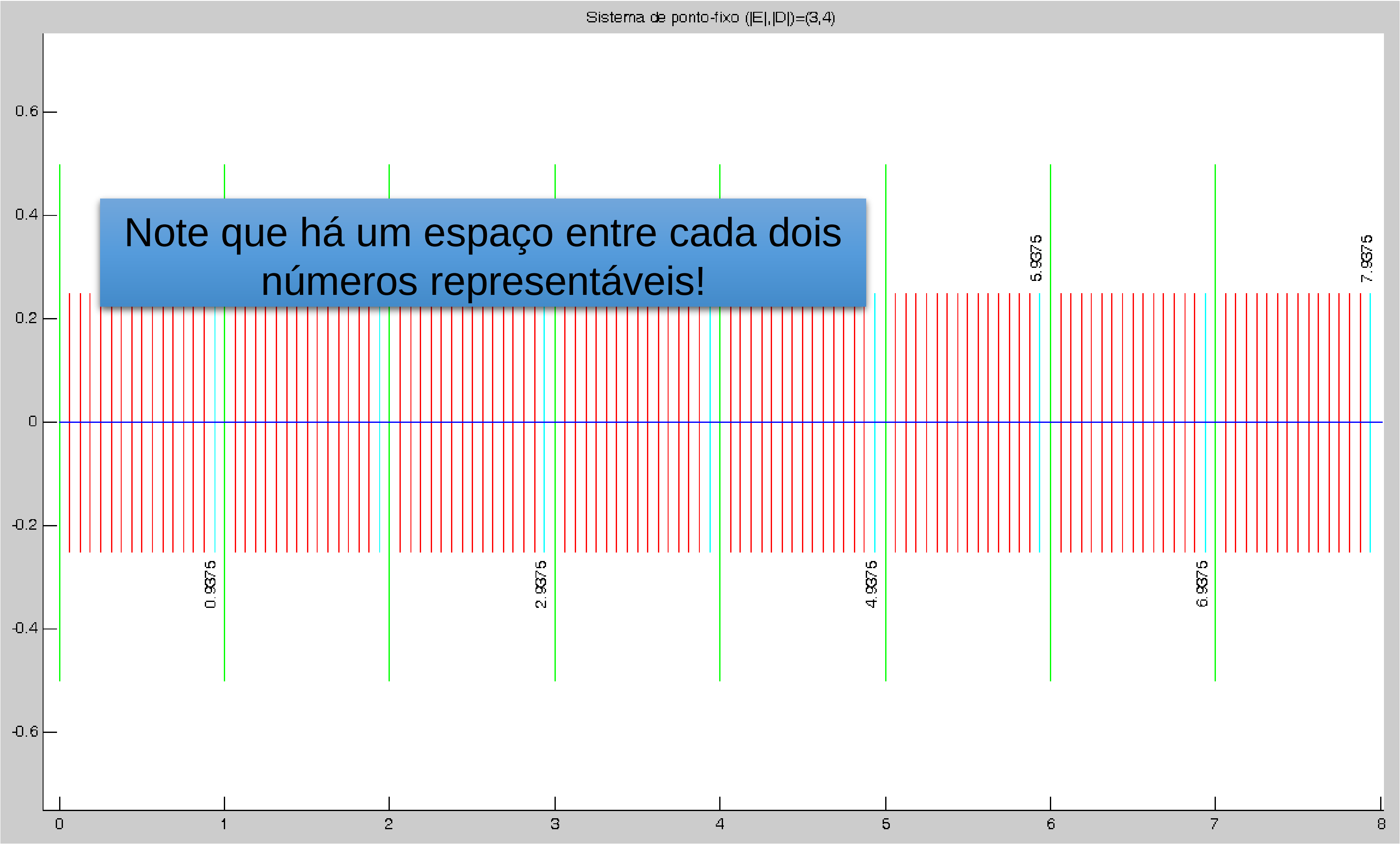
...

$(111.1111)_2 = 7.9375$

- Representação de números reais em ponto-fixo
  - Para esse exemplo, com $|E|=3$ e $|D|=4$, temos um total de:
    - Quinze números, maiores do que 0 e com parte inteira nula
    - Sete números com parte decimal nula
  - De tal forma que podemos dizer que existem $2x(8x15+7)+2 = 256 = 2x2^{|E|+|D|}$ números *representáveis* no *sistema de ponto-fixo* $(|E|,|D|)=(3,4)$, i.e. os números positivos e negativos diferentes de zero, além de duas representações para o zero (+0 e -0)

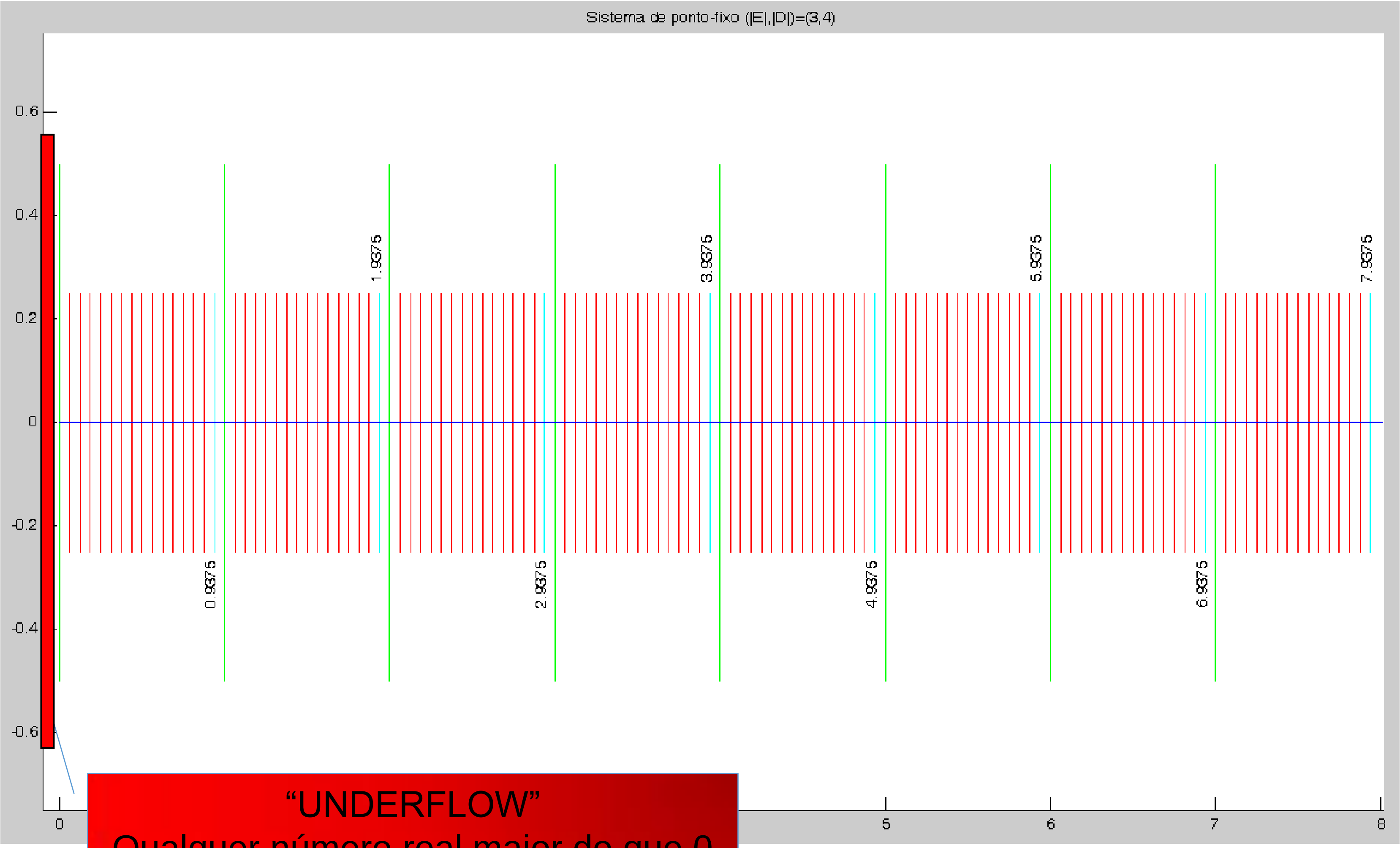

Sistema de ponto-fixo (|E|,|D|)=(3,4)
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
1.9375
3.9375
5.9375
7.9375
0.9375
2.9375
4.9375
6.9375
0
1
2
3
4
5
6
7
8


Sistema de ponto-fixo (|E|,|D|)=(3,4)
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
0.9375
1.9375
2.9375
4.9375
6.9375
0
1
2
3
4
5
6
7
8
Observe como os números representáveis se distribuem ao longo da reta dos números reais

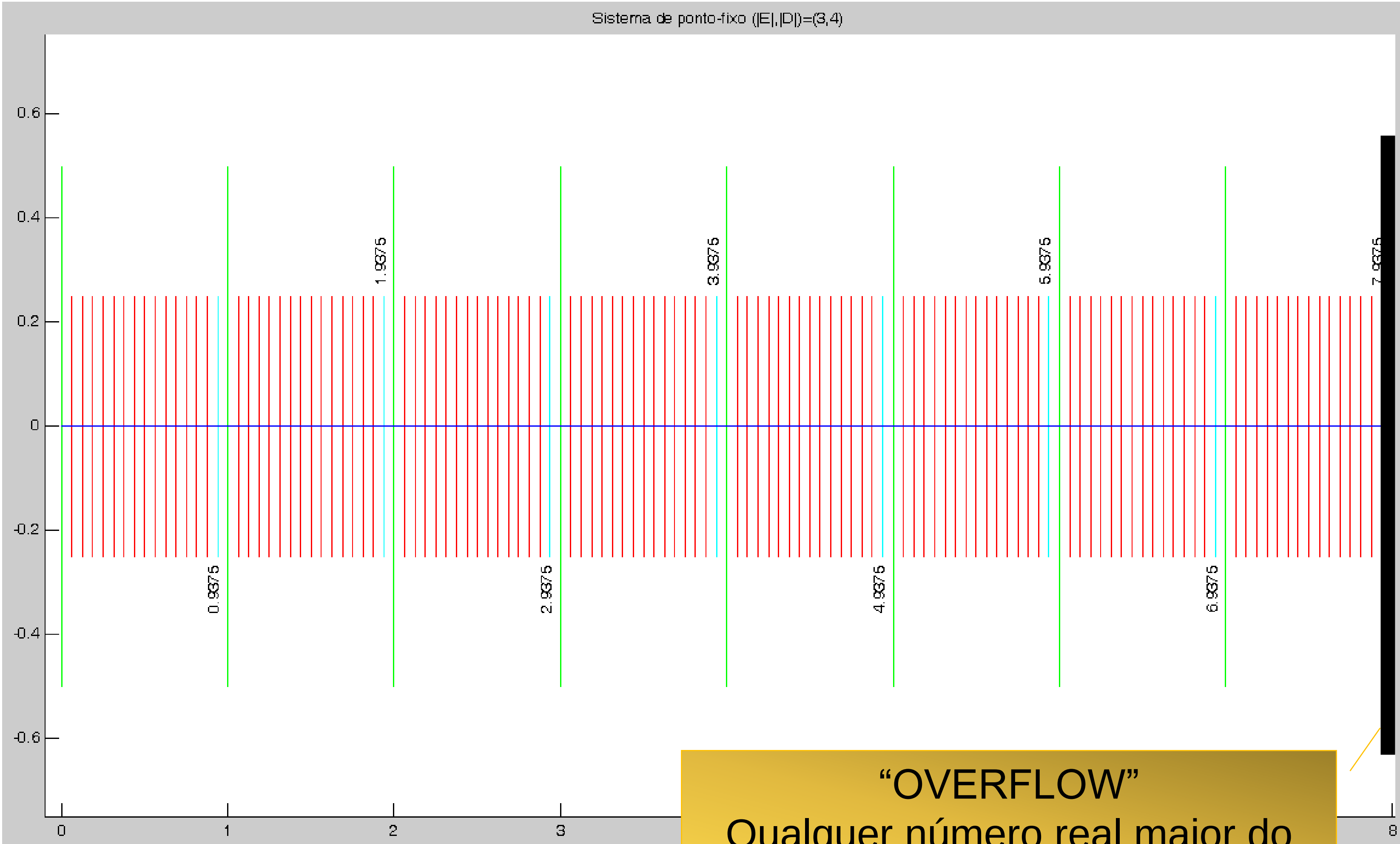


Introdução ao Cálculo Numérico

A distância entre dois números representáveis consecutivos é de 1 ULP = 2^{-4} !



“UNDERFLOW”
Qualquer número real maior do que 0 e menor do que 2^{-4}



“OVERFLOW”
Qualquer número real maior do $(111.1111)_2 = 7.9375$

Introdução ao Cálculo Numérico

Prof. Rudnei Dias da Cunha

- Representação de números reais em ponto-fixo
  - A existência de um espaço entre dois números representáveis consecutivos implica na ocorrência de um *erro* ao se tentar representar qualquer número real que não pertença ao *conjunto de números representáveis* do sistema de ponto-fixo
  - Por exemplo, suponha $x = 0.609375$; os dois números representáveis imediatamente menor e maior do que x são $x_a = 0.5625$ e $x_b = 0.625$
  - Como não existem outros números entre x_a e x_b que possam representar x, teremos de escolher entre um deles

- Representação de números reais em ponto-fixo
  - O processo de escolha entre dois números representáveis consecutivos para armazenar um número real chama-se de *arredondamento*
  - Existem dois tipos de arredondamento:
    - *Por corte*: desprezam-se os bits que encontram-se nas posições além do tamanho do campo D
    - *Por adição*: soma-se 1_2 ao bit $|D|+1$ (i.e. $0{,}5 \times 2^{-|D|} = 2^{-|D|-1}$) e desprezam-se os bits como no arredondamento por corte

- Representação de números reais em ponto-fixo
  - Voltando ao exemplo, observe que a representação binária desses números é:

$x_a = 0.5625 = (0.1001)_2$

$x = 0.609375 = (0.100111)_2$

$x_b = 0.625 = (0.1010)_2$

e os arredondamentos são feitos da seguinte forma:

$$x = (0.100111)_2 \xrightarrow{\textit{Por corte}} \text{fixo}(x) = (0.1001)_2$$

$$\begin{array}{r} x = (0.100111)_2 \\ +(0.00001)_2 \\ \hline x = (0.101001)_2 \end{array} \xrightarrow{\textit{Por adição}} \text{fixo}(x) = (0.1010)_2$$

- Representação de números reais em ponto-fixo
  - Como a representação de um número real no computador é feita por um arredondamento, isso leva a um *erro* na representação
  - Esse erro pode ser mensurado de duas formas:
    - **Erro absoluto**
    - **Erro relativo**
  - O *erro absoluto* e_a entre dois números a e b é definido como

$$e_a(a,b) = |a - b|$$

  - O *erro relativo* e_r entre dois números a e b é definido como

$$e_r(a,b) = \frac{|a - b|}{|a|}, a \neq 0$$

- Representação de números reais em ponto-fixo
  - Usando o exemplo anterior, observe que os *erros relativos* entre $x = 0.609375$ e $x_a = 0.5625$ e entre x e $x_b = 0.625$ são, respectivamente:

$$e_r(x, x_a) = \frac{|x - x_a|}{|x|} = 0.07692308 < 2 \cdot 2^{-4} = 0.125$$

$$e_r(x, x_b) = \frac{|x - x_b|}{|x|} = 0.02564103 < 2 \cdot 2^{-4} = 0.125$$

Em geral, pode-se afirmar que, num sistema de ponto-fixo,

$$e_r\big(x, \mathrm{fixo}(x)\big) < 2 \cdot \mathrm{ULP}$$

- Representação de números reais em ponto-fixo
  - Para se realizar as operações aritméticas (adição, subtração, multiplicação e divisão), é recomendável (dependendo da aplicação) que se utilize o dobro de bits destinados para os campos E e D.
  - Uma vez efetuada a operação, caso não tenha ocorrido um erro de “overflow”, é feito o arredondamento, e o resultado então é armazenado na palavra, desprezando-se os bits usados a mais para a realização da operação.

Representação de números reais em ponto-flutuante

- Representação de números reais em ponto-flutuante
  - Um número real a é representado num *sistema de ponto-flutuante* através da *notação científica normalizada (NCN),* i.e.

$$a = \pm M \times 10^{\pm E},$$

onde

$$\frac{1}{10} \leq M < 1$$

$$E \in \mathbb{N}$$

- Representação de números reais em ponto-flutuante
  - De forma equivalente, um número b em ponto-flutuante é armazenado em binário, na forma
$$b = \pm M \times 2^{\pm E},$$
    onde
$$\frac{1}{2} \leq M < 1$$
$$E \in \mathbb{N}$$
  e M é um número real e E é um número inteiro, ambos expressos em binário

- Representação de números reais em ponto-flutuante
  - O nome *ponto-flutuante* é devido ao fato de que, variando-se o expoente E, o ponto decimal “flutua” de posição entre os dígitos da mantissa M, sem alterar o valor do número:

$$
\begin{aligned}
3.1415926 \quad &= \quad 0.314159260 \times 10^{+1} \\
&= \quad 0.031415926 \times 10^{+2} \\
&= \quad 31.415926000 \times 10^{-1}
\end{aligned}
$$

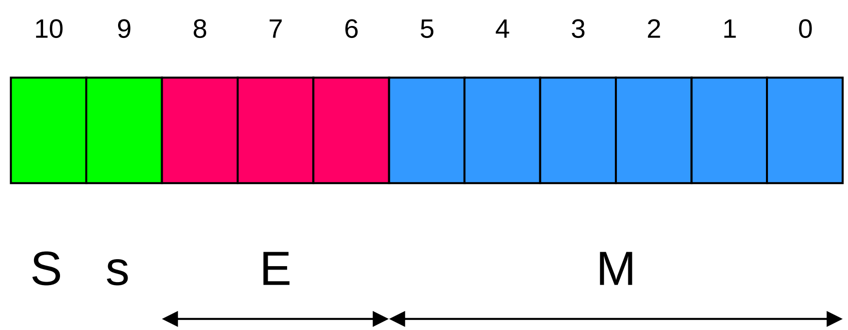
- Representação de números reais em ponto-flutuante
  - A mantissa M é um número binário na forma

$$M = (0.1m_1m_2m_3\ldots)_2$$

já que, por definição, $0.5 \leq M < 1$

  - O expoente E é um número inteiro cujo intervalo de representação depende de quantos bits são alocados para armazená-lo

- Representação de números reais em ponto-flutuante
  - Da mesma forma que no sistema de ponto-fixo, divide-se a palavra em campos, atribuindo-se um número de bits para cada
  - Temos, agora, quatro campos a definir:
    - S: sinal da mantissa
    - s: sinal do expoente
    - M: mantissa
    - E: expoente
  - No diagrama abaixo: S=1, s=1, E=3 e M=6



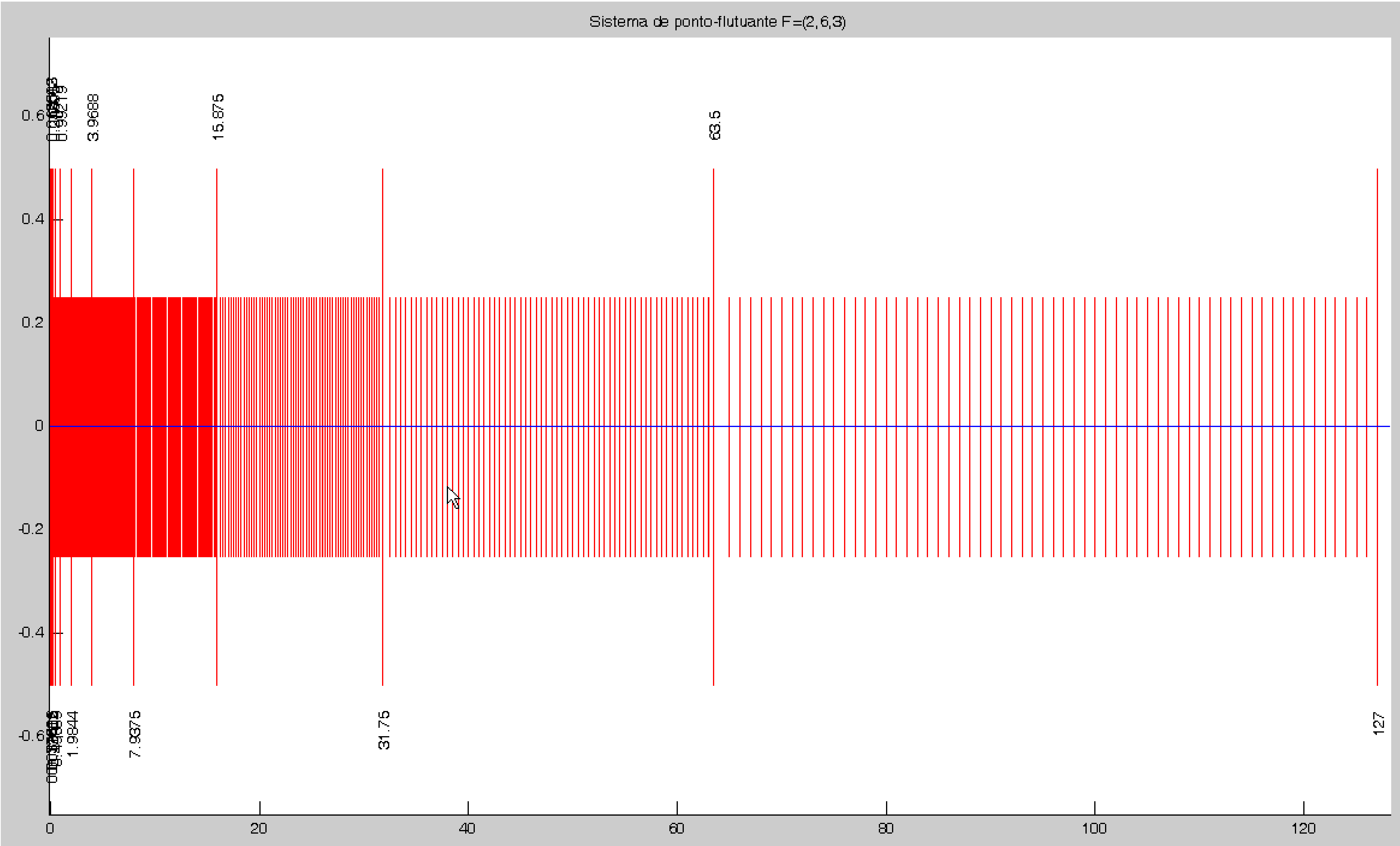
- Representação de números reais em ponto-flutuante
  - Podemos caracterizar um sistema de ponto-flutuante $\mathrm{F}(b, M, E)$, onde b é a base, M é a quantidade de bits na mantissa e E é a quantidade de bits no expoente, através de algumas constantes, como:
    - A precisão, *p*
    - *O menor e o maior expoentes representáveis (MINE e MAXE)*
    - *O menor e o maior números representáveis (MINR* e *MAXR)*
    - *A separação entre dois números representáveis consecutivos (ULP)*
    - *O "épsilon da máquina", o menor número representável positivo para o qual*

$$\mathrm{fl}(1+\varepsilon) \neq 1$$

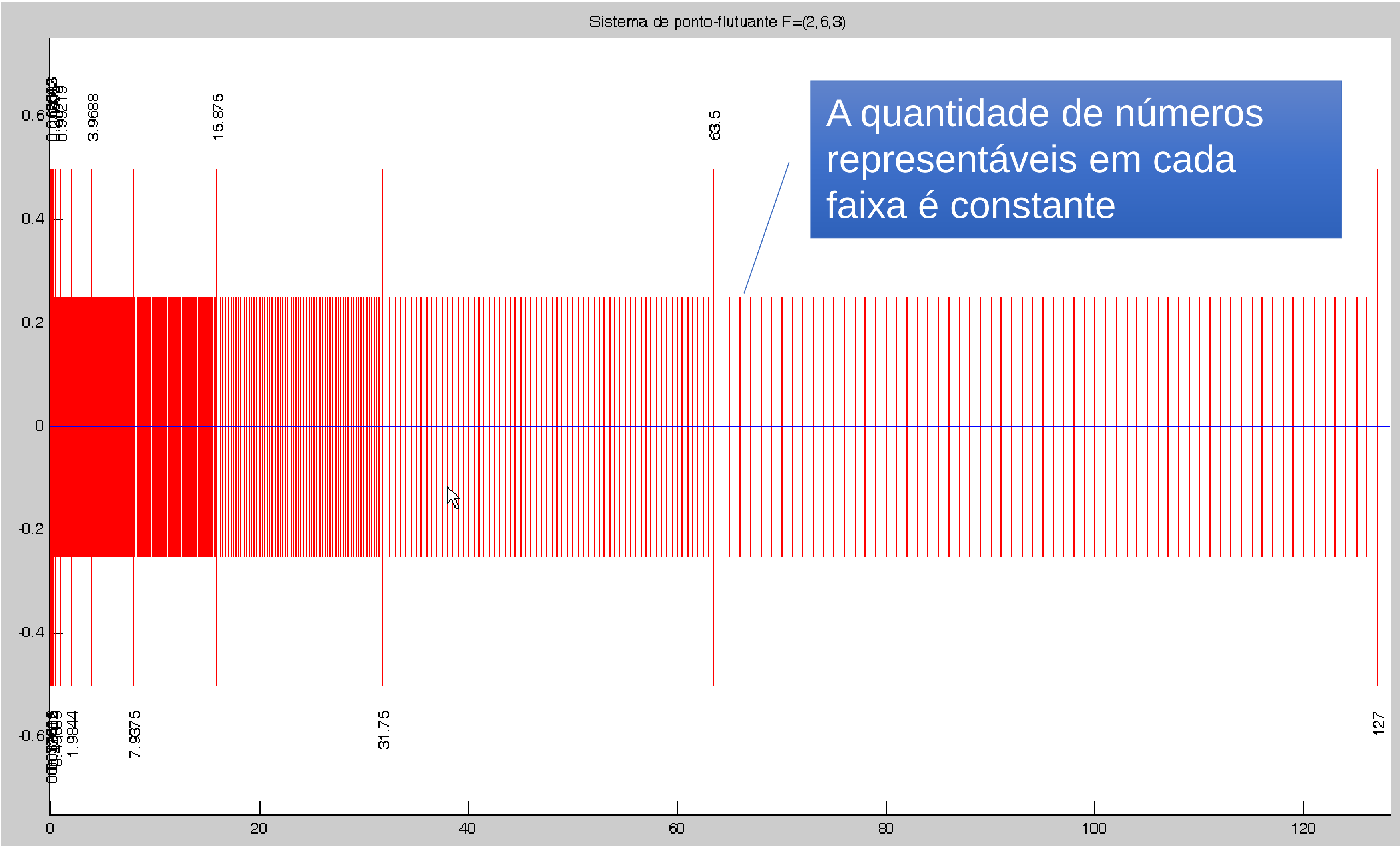
onde $\mathrm{fl}(x)$ *é a representação em ponto-flutuante de* x

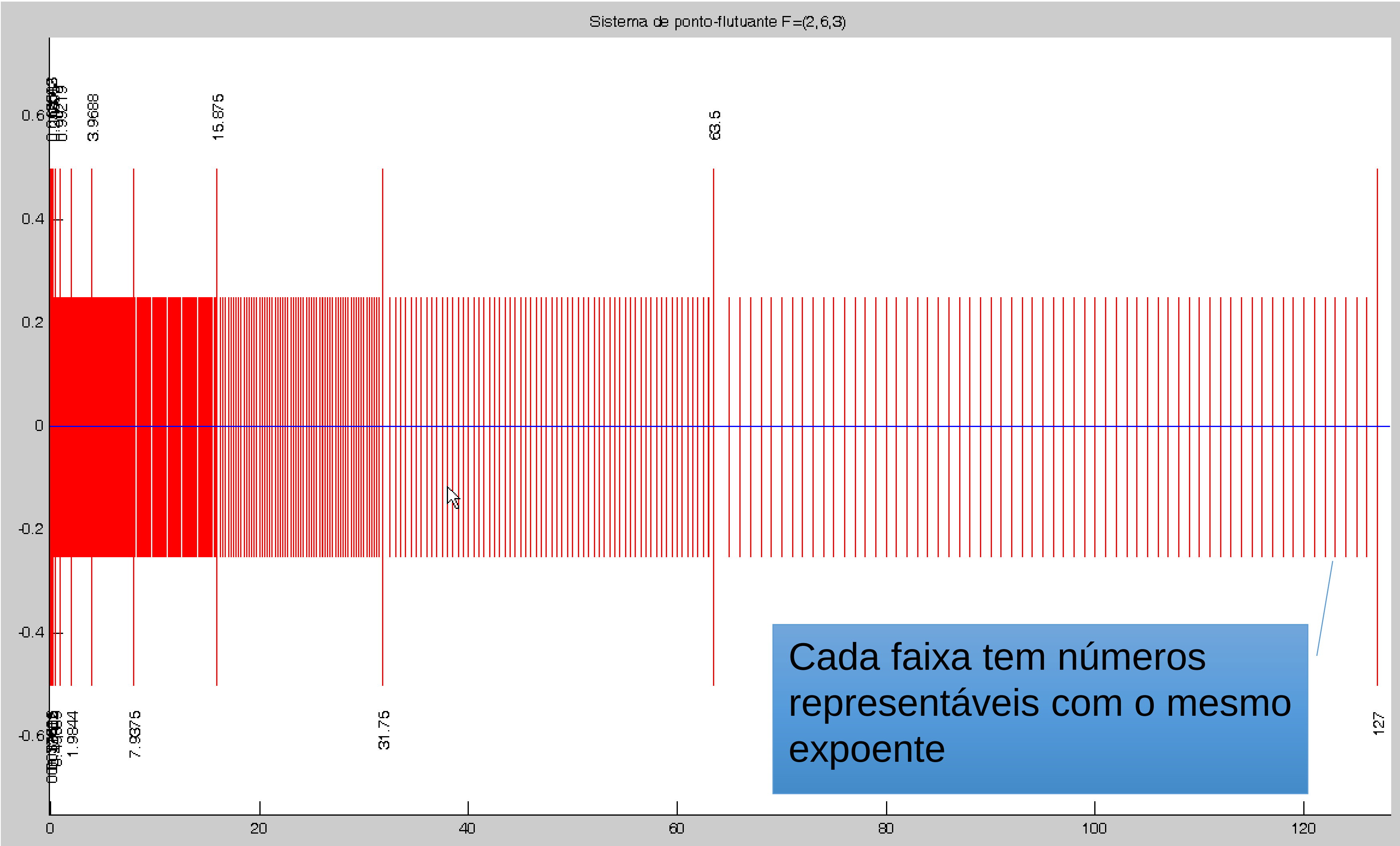
- Representação de números reais em ponto-flutuante
  - Observe que o bit m_0 da mantissa *é sempre 1!*
  - Logo, ele não precisa ser armazenado; isso libera um bit para armazenamento a mais na mantissa (bit menos significativo)
  - Portanto, para determinarmos as constantes que caracterizam o sistema de ponto-flutuante, devemos considerar que a mantissa tem sempre um bit a mais

- Representação de números reais em ponto-flutuante
  - Como exemplo, vamos considerar uma palavra de 11 bits, com |E|=3 e |M|=6
    - p = |M|+1 = 7
    - MINE = $-(111)_2 = -7$
    - MAXE = $(111)_2 = 7$
    - MINR = $(0.100\ 000\ 0\ )_2 \times 2^{\text{MINE}} = 0.5 \times 0.0078125 = 0.00390625$
    - MAXR = $(0.111\ 111\ 1)_2 \times 2^{\text{MAXE}} = 0.9921875 \times 2^7 = 127$
    - ULP = $(0.000\ 000\ 1)_2 \times 2^E = 2^{-p} \times 2^E$
    - O valor de ε pode ser calculado como:
      - $\varepsilon = 2^{-p+1}$, para arredondamento por corte
      - $\varepsilon = 2^{-p}$, para arredondamento por adição



Introdução ao Cálculo Numérico

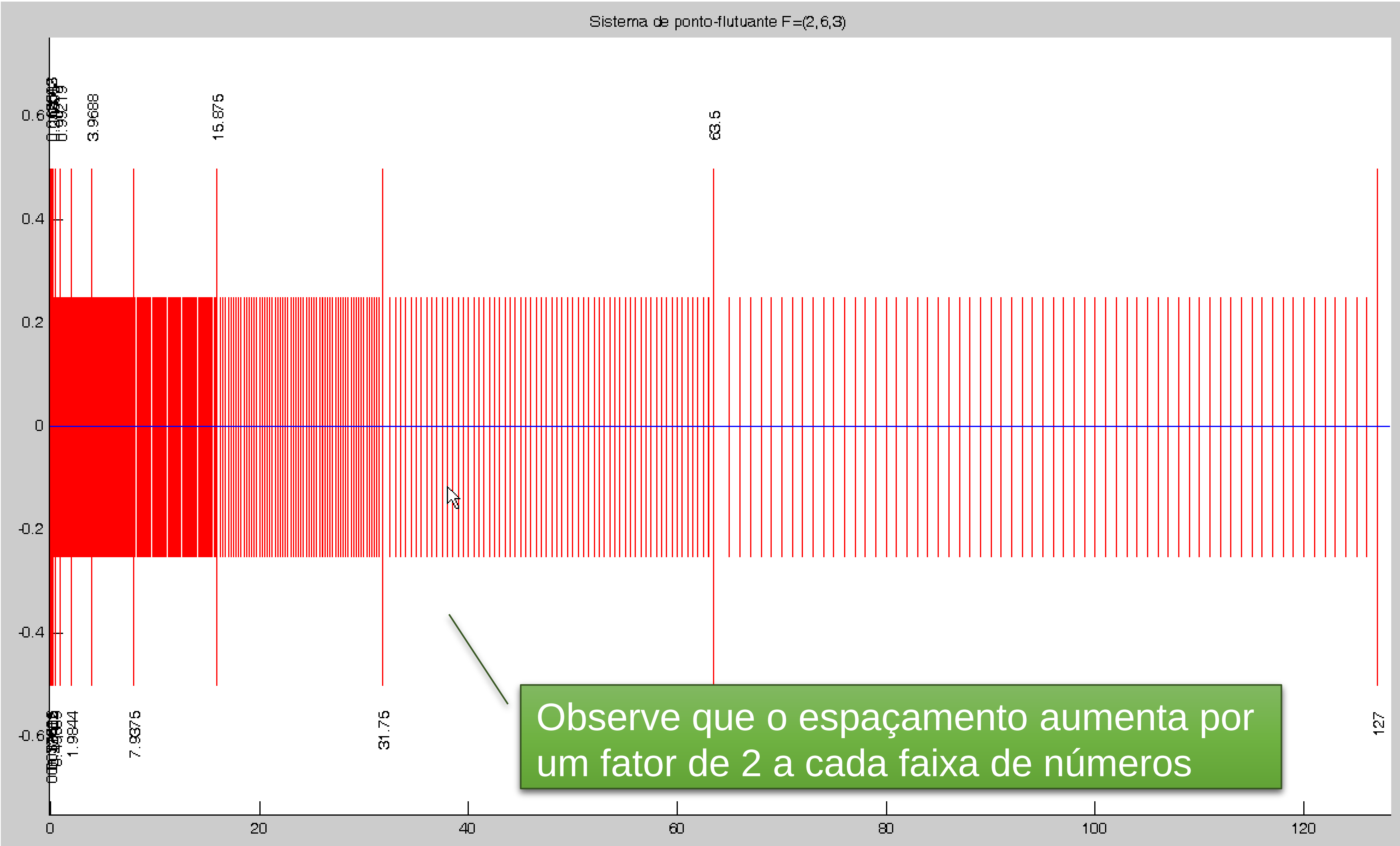

Sistema de ponto-flutuante F=(2,6,3)
A quantidade de números representáveis em cada faixa é constante
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
3.9688
15.875
63.5
1.9844
7.9375
31.75
127
0
20
40
60
80
100
120

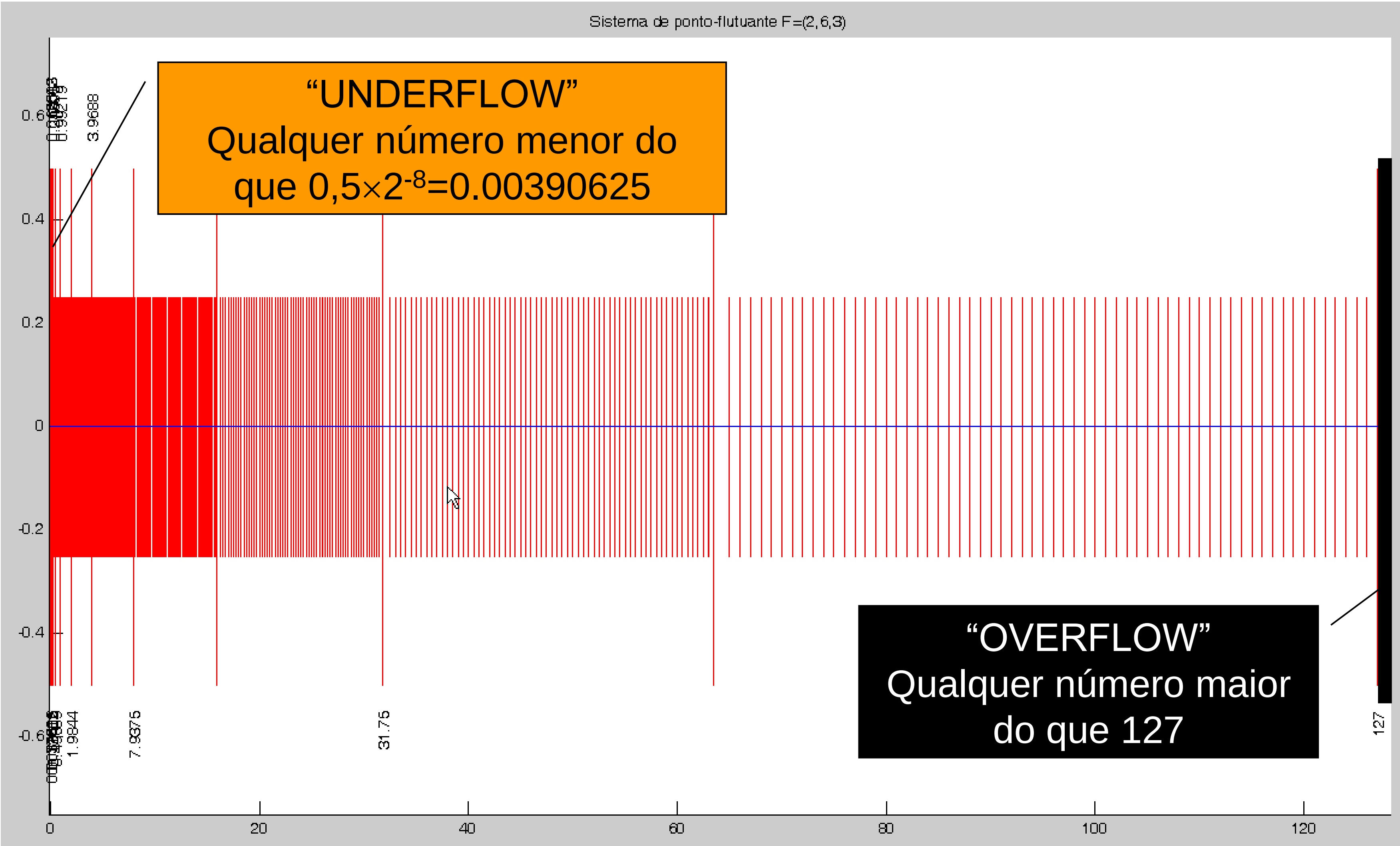


Introdução ao Cálculo Numérico

Instituto de MATEMÁTICA E ESTATÍSTICA UFRGS

Prof. Rudnei Dias da Cunha


Sistema de ponto-flutuante F=(2,6,3)
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
0
20
40
60
80
100
120
3.9688
15.875
63.5
1.9844
7.9375
31.75
127
Observe que o espaçamento aumenta por um fator de 2 a cada faixa de números


Sistema de ponto-flutuante F=(2,6,3)
"UNDERFLOW"
Qualquer número menor do
que 0,5×2-8=0.00390625
"OVERFLOW"
Qualquer número maior
do que 127
3.9688
1.9844
7.9375
31.75
127
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
0
20
40
60
80
100
120

- Representação de números reais em ponto-flutuante
  - Da mesma forma que na representação em ponto-fixo, note que o conjunto de números representáveis é finito
  - Logo, para representarmos um número real que não pertença a esse conjunto, haverá a necessidade de escolhermos um elemento do conjunto para representá-lo

- Representação de números reais em ponto-flutuante
  - Esse processo de escolha é feito novamente através do *arredondamento*, seja ele por corte ou por adição
  - No caso da representação em ponto-flutuante, podemos estimar os erros absoluto e relativo associados a esses arredondamentos:
    - **Erro absoluto**:
$$e_a\big(x, \mathrm{fl}(x)\big) < 2^{E-(p+1)}$$
    - **Erro relativo**:
$$e_r\big(x, \mathrm{fl}(x)\big) < 2^{-p}$$

- Representação de números reais em ponto-flutuante
  - Observe que 2^{-p} é o chamado *épsilon da máquina,* no arredondamento por adição (comumente usado)
  - Podemos então dizer que cada número real, ao ser armazenado no computador, sofre um arredondamento, de onde podemos escrever

$$e_r\big(x, \mathrm{fl}(x)\big) < \mathcal{E}$$

  - Ou, ainda,

$$\mathrm{fl}(x) = x(1 + \delta), |\delta| \le \varepsilon$$

$$\delta = \frac{\mathrm{fl}(x) - x}{x}$$

- Representação de números reais em ponto-flutuante
  - Uma medida comum para se avaliar a qualidade da resposta de um processo numérico é a quantidade de *dígitos significativos exatos* (DIGSE)
  - Aplicando logaritmos nos dois lados da inequação anterior, obtemos

$$-\log_{10}\frac{|x-\mathrm{fl}(x)|}{|x|} \geq -\log_{10}\varepsilon$$

  - E definimos DIGSE como o lado esquerdo da desigualdade

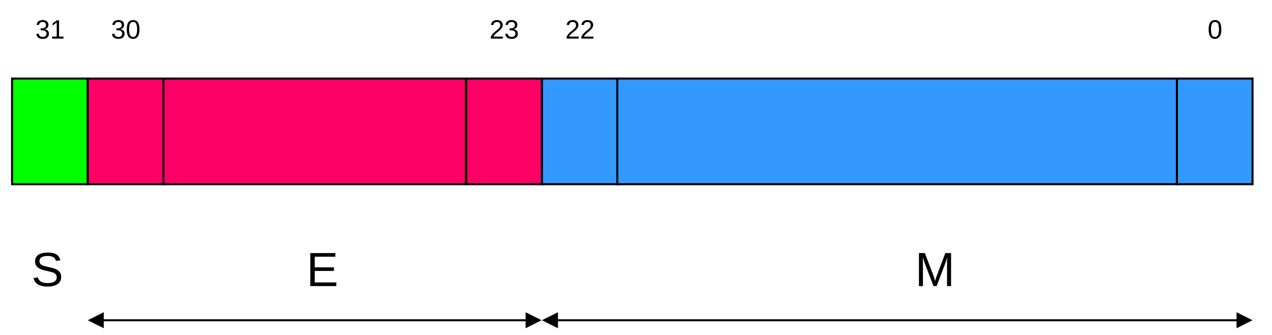
$$DIGSE = -\log_{10}\frac{|x-\mathrm{fl}(x)|}{|x|}$$

- Representação de números reais em ponto-flutuante
  - O DIGSE também pode ser usado como um critério de parada em algoritmos numéricos de busca; suponha que uma sequência de valores $x_0, x_1, \dots, x_k, x_{k+1}$ tende a um valor $\tilde{x}$; então, pode-se parar a geração dos valores x_k quando

$$DIGSE = -\log_{10} \frac{|x_k - x_{k+1}|}{|x_k|} \geq 7,$$

para cálculos em precisão simples (32 bits).

- Representação de números reais em ponto-flutuante – Padrão IEEE-754
  - O padrão IEEE-754 define como um número real é armazenado em ponto-flutuante num computador
  - Tal padrão é adotado universalmente pelos fabricantes de processadores
  - Em precisão simples – i.e. 32 bits – a palavra é dividida da seguinte forma:
    - *S (1 bit), E (8 bits, em complemento-de-2) e M (23 bits)*



- Representação de números reais em ponto-flutuante – Padrão IEEE-754
  - O expoente E é armazenado adicionando-se a ele o valor +127 (denominado de "bias")
  - Observe que o bit $m_0=1$ não é armazenado; logo, os bits 0, 1, ..., 22 da palavra são usados para armazenar os bits $m_1, m_2, \dots, m_{23}$. Daí,

$$M = \frac{1}{2} + \sum_{i=0}^{22} \left(m_i \times 2^{-(i+2)} \right), \frac{1}{2} \leq M < 1$$

  - O padrão introduziu também dois valores, NaN (*Not a Number*) e Inf (*Infinite*), que correspondem a certos padrões de bits armazenados na palavra

- Representação de números reais em ponto-flutuante – Padrão IEEE-754
  - Dependendo dos bits armazenados nos bits que formam os campos E e M, o valor de $\mathrm{fl}(x)$, armazenado na palavra, é:

E	M	$\mathrm{fl}(x)$	Valores representáveis
$0 < E < 255$	$M \neq 0$	$(-1)^S(1.M) \times 2^{E-127}$	$0.1175 \times 10^{-37} \leq \mathrm{fl}(x) \leq 0.3403 \times 10^{+39}$
$E = 0$	$M \neq 0$	$(-1)^S(0.M) \times 2^{E-126}$	$5.8775 \times 10^{-39} \leq \mathrm{fl}(x) \leq 1.1755 \times 10^{-38}$
$E = 0$	$M = 0$	$(-1)^S\ 0$	$\mathrm{fl}(x) = -0$ ou $\mathrm{fl}(x) = +0$
$E = 255$	$M \neq 0$	NaN	-
$E = 255$	$M = 0$	$(-1)^S\ \mathrm{Inf}$	$\mathrm{fl}(x) = -\infty$ ou $\mathrm{fl}(x) = +\infty$

Padrão de bits que corresponde a $\mathrm{fl}(x) = +\infty$, pois $S = 0$ (positivo), $E = 255$ e $M = 0$:

0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- Representação de números reais em ponto-flutuante – Padrão IEEE-754
  - Note que, quando *0<E<255*, deve-se adicionar 1.0 a M, para calcular o valor de $\mathrm{fl}(x)$
  - Ainda, como

$$\text{DIGSE} \geq -\log_{10} \varepsilon,$$

para $\varepsilon = 2^{-24}$ (a precisão usada no padrão IEEE-754 em 32 bits), teremos no mínimo 7 casas decimais de precisão

- Operações aritméticas com números reais em ponto-flutuante
  - Numa operação aritmética em ponto-flutuante, as seguintes etapas são efetuadas:
    1. A operação é feita de forma *“correta”*, i.e., com o dobro do número de bits da mantissa
    2. O resultado é normalizado
    3. É feito o arredondamento, de forma que o número possa ser armazenado na palavra (cuja mantissa tem a metade do número de bits da mantissa do resultado normalizado)

- Operações aritméticas com números reais em ponto-flutuante
  - Como os números em ponto-flutuante são armazenados em notação científica, as operações aritméticas executadas na primeira etapa são feitas da forma mostrada a seguir:
    - Na *adição* e *subtração*, os expoentes devem ser iguais; para tal, seleciona-se o operando com *maior* dos dois expoentes, e a mantissa e o expoente do outro operando tem seus bits deslocados convenientemente de forma a igualar os expoentes
    - Na *multiplicação* e *divisão*, basta efetuar a multiplicação ou divisão sobre as mantissas dos operandos e somar ou subtrair os expoentes, respectivamente

- Operações aritméticas com números reais em ponto-flutuante
  - Note que a normalização deve ser feita **antes do arredondamento**, c.c., poderá ocorrer a *perda catastrófica de dígitos*
  - Exemplo:

Seja $x = 0{,}45230 \times 10^{-2}$ e $y = 0{,}25470 \times 10^{-3}$. Calculando o produto xy, com 10 dígitos na mantissa, temos

$$xy = 0{,}00000\ 11520.$$

Se fizermos o arredondamento *antes* de normalizar xy, o resultado armazenado será

$$0{,}0000\ !!!$$

- Operações aritméticas com números reais em ponto-flutuante
  - Vejamos através de um exemplo uma medida para o erro relativo envolvido em cada operação aritmética:
  - A mantissa é de 5 dígitos decimais, i.e., $\varepsilon = 10^{-5}$
  - Os operandos são

$$x = 0{,}31426 \times 10^3 \text{ e } y = 0{,}92577 \times 10^5$$

  - Efetuando as quatro operações aritméticas, com o dobro de dígitos na mantissa, temos:

$x + y = 0{,}92891\ 00000 \times 10^5$

$x - y = -0{,}92262\ 74000 \times 10^5$

$x \times y = 0{,}29093\ 24802 \times 10^8$

$x \div y = 0{,}33945\ 79647 \times 10^{-2}$

$fl(x+y) = 0{,}92891 \times 10^5$, $e_r = 8{,}5 \times 10^{-6} < 10^{-5}$

$fl(x-y) = -0{,}92263 \times 10^5$, $e_r = 2{,}3 \times 10^{-6} < 10^{-5}$

$fl(x \times y) = 0{,}29093 \times 10^8$, $e_r = 2{,}8 \times 10^{-6} < 10^{-5}$

$fl(x \div y) = 0{,}33946 \times 10^{-2}$, $e_r = 6{,}0 \times 10^{-6} < 10^{-5}$

- Operações aritméticas com números reais em ponto-flutuante
  - Podemos então generalizar e dizer que uma operação aritmética em ponto-flutuante, $\diamond$, sempre incorre em um erro:

$$\mathrm{fl}(x \diamond y) = (x \diamond y)(1 + \delta), |\delta| \leq \varepsilon$$

se x e y são números representáveis. Se ambos não são números de máquina, então

$$\mathrm{fl}\big(\mathrm{fl}(x) \diamond \mathrm{fl}(y)\big) = \big(x(1 + \delta_1) \diamond y(1 + \delta_2)\big)(1 + \delta_3),$$
$$|\delta_{1,2,3}| \leq \varepsilon$$

- Operações aritméticas com números reais em ponto-flutuante
  - Além disso, se encadearmos várias operações aritméticas em sequência, é possível que o erro se torne de tal grandeza que a resposta esteja completamente errada!
  - Existe um teorema que prova que, no cálculo do produto interno entre dois vetores, cujos *n* elementos sejam todos números representáveis positivos, o erro acumulado pode ser de aproximadamente $n\varepsilon$:
    - Se $\varepsilon = 10^{-7}$ e $n = 100000$, o erro acumulado poderá ser da ordem de 10^{-2}, o qual é considerado muito grande

- Operações aritméticas com números reais em ponto-flutuante
  - Porém, mesmo que n seja pequeno, ainda é possível que a resposta seja completamente errada!
  - Considere o exemplo abaixo: sejam os vetores

$$x=[10^{20}, 1223, 10^{18}, 10^{15}, 3, -10^{12}]$$

e

$$y=[10^{20}, 2, -10^{22}, 10^{13}, 2111, 10^{16}].$$

Calculando à mão as parcelas do produto interno, obtemos:

$$x \bullet y=\cancel{10^{40}}+2446-\cancel{10^{40}}+\cancel{10^{28}}+6333-\cancel{10^{28}}=8779$$

- Operações aritméticas com números reais em ponto-flutuante
  - Mas, se calcularmos num computador, obtemos como resultado... 0!
  - A razão está na ordem em que se faz as somas das parcelas:

$x_1*y_1+ x_2*y_2=10^{20}*10^{20}+1223*2=10^{40}+2446=10^{40}$

$(x_1*y_1+ x_2*y_2)+x_3*y_3=10^{40}+10^{18}*(-10^{22})=10^{40}-10^{40}=0$

$(x_1*y_1+ x_2*y_2+x_3*y_3)+x_4*y_4=0+10^{15}*10^{13=}\ 10^{28}$

$(x_1*y_1+ x_2*y_2+x_3*y_3+x_4*y_4)+x_5*y_5= 10^{28}+3*2111=10^{28}+6333=10^{28}$

$(x_1*y_1+ x_2*y_2+x_3*y_3+x_4*y_4+x_5*y_5)+x_6*y_6=10^{28}+(-10^{12})*10^{16}=$

$=10^{28}-10^{28}=0$

- Operações aritméticas com números reais em ponto-flutuante
  - Se trocarmos os elementos de lugar nos vetores *x* e *y*:

$$x=[10^{20}, 10^{18}, 10^{15}, -10^{12}, 1223, 3]$$

    e

$$y=[10^{20}, -10^{22}, 10^{13}, 10^{16}, 2, 2111]$$

    e calcularmos o produto interno no computador, obteremos, como resposta, $x \bullet y=8779$, que é a resposta correta!
  - Esse exemplo mostra como podemos identificar possíveis fontes de erro em programas!

- O *épsilon da máquina*
  - Já havíamos aludido a tal constante na representação em ponto-fixo, porém como o hardware de que dispomos implementa apenas a representação em ponto-flutuante, agora é possível estudarmos com mais cuidado tal constante
  - Relembrando: o ε da máquina é o menor número representável positivo para o qual $\mathrm{fl}(1+\varepsilon) \neq 1$

- O *épsilon da máquina*
  - Essa definição implica na existência de números representáveis, menores do que ε e maiores do que 0 (em módulo) que, somados a 1, resultam em 1!
  - Evidentemente, tal situação só pode ocorrer devido à *maneira* como as operações aritméticas são efetuadas em ponto-flutuante, envolvendo normalizações e arredondamentos

- O *épsilon da máquina*
  - Para ver como tal ocorre, suponha um sistema de ponto-flutuante $\mathrm{F} = (2,2,2)$, i.e. com *p=3* dígitos binários na mantissa (incluindo o bit mais significativo, igual a 1, não armazenado), e 2 dígitos no expoente
  - Os números que podem ser representados nesse sistema, em valor absoluto, são:

- O *épsilon da máquina*
  - Números representáveis em $\mathrm{F} = (2,2,2)$

$0.100 \times 2^{-11} = 0.0625$	$0.100 \times 2^{-10} = 0.125$	$0.100 \times 2^{-01} = 0.25$
$0.101 \times 2^{-11} = 0.078125$	$0.101 \times 2^{-10} = 0.15625$	$0.101 \times 2^{-01} = 0.3125$
$0.110 \times 2^{-11} = 0.09375$	$0.110 \times 2^{-10} = 0.1875$	$0.110 \times 2^{-01} = 0.375$
$0.111 \times 2^{-11} = 0.109375$	$0.111 \times 2^{-10} = 0.21875$	$0.111 \times 2^{-01} = 0.4375$
$0.100 \times 2^{00} = 0.5$	$0.100 \times 2^{01} = 1.0$	$0.100 \times 2^{10} = 2.0$
$0.101 \times 2^{00} = 0.625$	$0.101 \times 2^{01} = 1.25$	$0.101 \times 2^{10} = 2.5$
$0.110 \times 2^{00} = 0.75$	$0.110 \times 2^{01} = 1.5$	$0.110 \times 2^{10} = 3.0$
$0.111 \times 2^{00} = 0.875$	$0.111 \times 2^{01} = 1.75$	$0.111 \times 2^{10} = 3.5$
$0.100 \times 2^{11} = 4.0$		
$0.101 \times 2^{11} = 5.0$		
$0.110 \times 2^{11} = 6.0$		
$0.111 \times 2^{11} = 7.0$		

NOTA: números em F com mantissa e expoente representados em binário

- O *épsilon da máquina*
  - Como estimá-lo?
    - O procedimento a ser seguido é o seguinte: usando uma variável denominada “s”, inicializada com 1.0, dividimo-la repetidas vezes por 2, até que 1.0+s/2 seja menor ou igual a 1.0:

```
s:= 1.0
s_novo:= s/2
enquanto 1.0+s_novo>1.0 faça
    s:=s_novo
    s_novo:= s/2
fim enquanto
```

    - O último valor de *s* será o valor de ε
    - Esse procedimento estima ε por um fator de 2

- O *épsilon da máquina*
  - Processo de estimação do *épsilon da máquina:*

$1.\ s=0.100x2^{01} = 1.0$

$1.0+s/2.0 = 0.100x2^{01}+0.100x2^{00}$

$$
\begin{array}{l}
0.100\ 000 \times 2^{01} \\
\underline{0.010\ 000 \times 2^{01}\ +} \\
0.110\ \cancel{000} \times 2^{01} = 1.5 > 1.0 \rightarrow repete
\end{array}
$$

Nota: observe que o ajuste dos expoentes é feito escolhendo-se o número que tem maior expoente!

- O *épsilon da máquina*
  - Processo de estimação do *épsilon da máquina:*

$$2.\ s = 0.100 \times 2^{00} = 0.5$$

$$1.0 + s/2.0 = 0.100 \times 2^{01} + 0.100 \times 2^{-01}$$

$$\begin{array}{l} 0.100\ 000 \times 2^{01} \\ \underline{0.001\ 000 \times 2^{01}\ +} \\ 0.101\ \cancel{000} \times 2^{01} = 1.25 > 1.0 \rightarrow \textit{repete} \end{array}$$

- O *épsilon da máquina*
  - Processo de estimação do *épsilon da máquina:*

3. $s = 0.100 x 2^{-01} = 0.25$

$1.0+s/2.0 = 0.100x2^{01}+0.100x2^{-10}$

$$
\begin{array}{l}
0.100\ 000 \times 2^{01} \\
\underline{0.000\ 100 \times 2^{01}\ +} \\
0.100\ \cancel{100} \times 2^{01} = 1.0 = 1.0 \rightarrow \varepsilon = 0.25
\end{array}
$$

- O *épsilon da máquina*
  - Com efeito, se efetuarmos a soma entre 1.0 e o maior número representável menor do que ε, teremos:

$$1.0+0.21875: 0.100x2^{01}+0.111x2^{-10}$$

$$\begin{array}{l} 0.100\ 000 \times 2^{01} \\ \underline{0.000\ 111 \times 2^{01} +} \\ 0.100\ 111 \times 2^{01} = 1.0 \end{array}$$

e, portanto, todos os demais números representáveis menores do que ε e maiores do que 0, se somados a 1.0, resultam em 1.0

- O *épsilon da máquina*
  - Como estimá-lo?
    - Como usamos arredondamento por corte, ε é dado pela fórmula $2^{-p+1} = 2^{-3+1} = 0,25$, o mesmo valor obtido pelo algoritmo
    - Para computadores usando o padrão IEEE-754, temos:

precisão	p	ε
simples (32 bits)	23	$1,1920928955 \times 10^{-7}$
dupla (64 bits)	53	$2,2204460492503131 \times 10^{-16}$
quádrupla (128 bits)	113	$1,925929944387235853055977942584927 32 \times 10^{-34}$

- Perda de dígitos significativos
  - Sempre que dois números em ponto-flutuante forem muito próximos, a subtração entre eles irá acarretar a perda de dígitos significativos
  - Inúmeras vezes, tal situação ocorre do emprego de expressões aritméticas que são inadequadas para seu uso num computador
  - Por exemplo, considere

    $x = 0{,}3721478693$ e $y = 0{,}3720230572$ e a sua subtração, $x - y = \ 0{,}0001248121$

    Se tivermos apenas 5 dígitos na mantissa:

    $\mathrm{fl}(x) = 0{,}37215$ e $\mathrm{fl}(y) = 0{,}37202$

    $$\mathrm{fl}(x) - \mathrm{fl}(y) = 0{,}00013$$

    - *O erro relativo será de 4%, bastante alto*

- Perda de dígitos significativos
  - Por exemplo, no cálculo das raízes de uma equação de 2º grau, $ax^2 + bx + c = 0$, poderá ocorrer tal perda se $b^2 \gg 4ac$, usando a fórmula de Báskara para calcular ambas as raízes
  - A solução para remediar em grande parte tal perda é determinar fórmulas alternativas para calcular uma das raízes de forma a evitar tal erro numérico
  - Vejamos como obter a fórmula de Báskara e uma alternativa a ela:

- Perda de dígitos significativos
  - Multiplicando $ax^2 + bx + c = 0$ por $4a$ e completando os quadrados, obtemos $(2ax + b)^2 + (4ac - b^2) = 0$, de onde obtemos a fórmula de Báskara

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \qquad (1)$$

  - Se $b^2 \gg 4ac$, ao se calcular numericamente $b^2 - 4ac$, é possível que o resultado obtido seja igual a b^2 e, nesse caso, a raiz calculada com o sinal negativo no numerador será igual (erroneamente) a 0.

- Perda de dígitos significativos
  - Por outro lado, dividindo $ax^2 + bx + c = 0$ por x^2, obtemos

$$a + b\frac{1}{x} + c\frac{1}{x^2} = 0$$

de onde, completando os quadrados, obtemos

$$\frac{1}{x} = \frac{-b \mp \sqrt{b^2 - 4ac}}{2c}$$

ou,

$$x = \frac{2c}{-b \mp \sqrt{b^2 - 4ac}} \qquad (2)$$

- Perda de dígitos significativos
  - Como as equações (1) e (2) são matematicamente equivalentes, isso sugere que devemos usar (1) ou (2) para calcular uma das raízes, de forma a evitar a subtração por valores muito próximos entre si
  - A outra raiz é calculada usando a relação $x_1 x_2 = c/a$
  - Assim, teremos:

$$\boxed{b \geq 0}: x_1 = \frac{-b - \sqrt{b^2 - 4ac}}{2a} \qquad (3)$$

$$\boxed{b < 0}: x_1 = \frac{2c}{-b + \sqrt{b^2 - 4ac}} \qquad (4)$$

- Perda de dígitos significativos
  - Ou seja, quando $b^2 \gg 4ac$, $\sqrt{b^2 - 4ac} = b$ e, para evitarmos a subtração $b - \sqrt{b^2 - 4ac}$, usaremos a equação (3) quando $b \geq 0$, pois o numerador será calculado como – $b - b$
  - Da mesma forma, se $b < 0$, usaremos a equação (4), pois o denominador será calculado como $b + b$
  - Vejamos dois exemplos ilustrativos:

- Perda de dígitos significativos
  - Exemplo 1: $2x^2 + 10x + 2 = 0$

Forma de cálculo	x_1	$2{x_1}^2 + 10x_1 + 2$	x_2	$2{x_2}^2 + 10x_2 + 2$
instável	-4.7912878474779195	0.0	-0.20871215252208009	$-8.8817841970012523 \times 10^{-16}$
estável	-4.7912878474779195	0.0	-0.20871215252208003	$-4.4408920985006262 \times 10^{-16}$

  - Exemplo 2: $2x^2 + 4 \times 10^6 x + 2 = 0$

Forma de cálculo	x_1	$2{x_1}^2 + 4 \times 10^6 x_1 + 2$	x_2	$2{x_2}^2 + 4 \times 10^6 x_2 + 2$
instável	-199999.99999500002	0.0	$-4.9999944167211652 \times 10^{-6}$	$2.2333615337100099 \times 10^{-6}$
estável	-199999.99999500002	0.0	$-5.0000000001249996 \times 10^{-6}$	0.0

Ambos os exemplos foram calculados no Scilab, o qual usa precisão dupla ($\varepsilon = 2.220446049 \times 10^{-16}$).

- Perda de dígitos significativos
  - Outro exemplo: suponha a função

$$f(x) = \sqrt{1+x} - \sqrt{1-x}, |x| \ll 1$$

  - Usando um programa com aritmética de precisão dupla, obtemos:

$$f(10^{-10}) = 10^{-10}$$
$$f(10^{-20}) = 0.000 \text{ (!!!!)}$$

- Perda de dígitos significativos
  - O que ocorre é que os dois termos com raízes quadradas são muito próximos de 1, para tais valores de x
  - Para $x = 10^{-10}$, obtivemos $f(10^{-10}) = 10^{-10}$:

$$\boldsymbol{\sqrt{1+x} = 1.00000000005000000}$$
$$\boldsymbol{\sqrt{1-x} = 0.99999999995000000}$$

  - Para $x = 10^{-20}$, obtivemos $f(10^{-20}) = 0.00000$:

$$\boldsymbol{\sqrt{1+x} = 1.00000000000000000}$$
$$\boldsymbol{\sqrt{1-x} = 1.00000000000000000}$$

- Perda de dígitos significativos
  - Como evitar isso? Reescrevendo a expressão de tal forma que a subtração seja eliminada:

$$f(x) = \left(\sqrt{1+x} - \sqrt{1-x}\right) \times \frac{\sqrt{1+x} + \sqrt{1-x}}{\sqrt{1+x} + \sqrt{1-x}} =$$

$$\frac{1+x-1+x}{\sqrt{1+x} + \sqrt{1-x}} = \boxed{\frac{2x}{\sqrt{1+x} + \sqrt{1-x}}}$$

  - Usando essa expressão para a função, obteremos

$$f(10^{-10}) = 10^{-10}$$
$$f(10^{-20}) = 10^{-20}$$