Aritmética computacional

Departamento de Matemática (Dmat)
Universidade do Minho

Carla Ferreira
2023/2024

Aritmética computacional

O objetivo de um método numérico é, como o próprio nome indica, produzir respostas numéricas a problemas matemáticos.

Em calculo numérico a informação é transportada sob a forma de números. É, pois, fundamental ter um bom conhecimento de como os computadores representam e operam com números, quais os erros cometidos e de que forma é que estes se podem avaliar e controlar.

Este capítulo aborda os tópicos da aritmética computacional considerados indispensáveis.

- Representação de números inteiros
- Representação de números reais
  - Representação em vírgula flutuante
  - Arredondamento e operações aritméticas
- A Norma IEEE 754 (IEEE standard)[1]

[1] IEEE - Institute for Electrical and Electronics Engineers

Representação de números naturais numa base b ($b \geq 2$)

Um sistema de numeração (notação usada para representar números) é definido pela base que utiliza. A base é o número de símbolos diferentes, ou algarismos, ou dígitos, necessários para representar um qualquer número.

Num **sistema de numeração posicional** um número é representado por uma sequência de dígitos que assumem um valor que depende da sua posição relativa na representação, correspondendo cada posição a uma determinada potência da base.

Um número natural $N \neq 0$ é representado numa **base $b \geq 2$** por uma expressão do tipo

$$N = (a_n a_{n-1} \cdots a_1 a_0)_b,$$

em que os a_i são os *dígitos* da base, $a_i \in \{0, 1, 2, \ldots, b-1\}$, $a_n \neq 0$.

A conversão à base 10 obtém-se por intermédio da expressão

$$N = a_n \times b^n + a_{n-1} \times b^{n-1} + \cdots + a_1 \times b^1 + a_0 \times b^0,$$

ou seja, o número é expresso como um polinómio em b.

Sistema decimal

Este é, nos nossos dias, o caso mais vulgar.

Tem este nome por possuir 10 dígitos diferentes:

$$0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8 \quad \text{e} \quad 9$$

Exemplo.

$$\begin{aligned} N = (1532)_{10} &= 1 \times 10^3 + 5 \times 10^2 + 3 \times 10^1 + 2 \times 10^0 \\ &= 1000 + 500 + 30 + 2 \end{aligned}$$

Observe-se que os dígitos 2, 3, 5 e 1 são os restos de sucessivas divisões por 10.

É muito comum e conveniente o uso de outras bases quando lidamos com computadores; as mais importantes são a binária (base 2), octal (base 8) e hexadecimal (base 16).

Sistema binário

Num computador um *bit* (binary digit) é um elemento de memória básico que assume dois estados *on* e *off* (dois tipos diferentes de sinais: tensão baixa ou tensão alta) que se associam aos dígitos 0 e 1.

Base binária	0	1	10	11	100	101	110	111	1000	1001
Base decimal	0	1	2	3	4	5	6	7	8	9

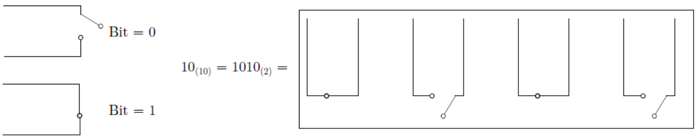

Figura 1: $10_{(10)} = (1010)_2$

Exemplo.

- *Conversão da base binária para a base decimal.*

$$\begin{aligned}(11101)_2 &= 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ &= 2^4 + 2^3 + 2^2 + 1 \\ &= 29\end{aligned}$$

- *Conversão da base decimal para a base binária.*

$$\begin{array}{cccccc} 29 & \multicolumn{1}{|l}{2} & & & & \\ \cline{2-2} 1 & 14 & \multicolumn{1}{|l}{2} & & & \\ \cline{3-3} & 0 & 7 & \multicolumn{1}{|l}{2} & & \\ \cline{4-4} & & 1 & 3 & \multicolumn{1}{|l}{2} & \\ \cline{5-5} & & & 1 & 1 & \multicolumn{1}{|l}{2} \\ \cline{6-6} & & & & 1 & 0 \end{array}$$

$$29 = (11101)_2$$

Mudança da base 10 para a base b

Sucessivas divisões por b

Dado um número natural $N \neq 0$ e uma base $b \geq 2$ pretende-se determinar os dígitos $a_0, a_1, \dots a_n \in \{1, 2, \dots, b-1\}$ tais que

$$N = a_n \times b^n + a_{n-1} \times b^{n-1} + \cdots + a_2 \times b^2 + a_1 \times b^1 + a_0 \times b^0,$$

ou seja, tais que $N = (a_n a_{n-1} \cdots a_1 a_0)_b$.

Observe-se que vamos poder escrever

$$N = \big(a_n \times b^{n-1} + a_{n-1} \times b^{n-2} + \cdots + a_2 \times b + a_1\big) b + \boldsymbol{a_0}.$$

Assim, se considerarmos a divisão inteira de N por b, a_0 será o resto dessa divisão e o quociente será

$$q_0 := a_n \times b^{n-1} + a_{n-1} \times b^{n-2} + \cdots + a_2 \times b + a_1.$$

Se dividirmos novamente

$$q_0 = \big(a_n \times b^{n-2} + a_{n-1} \times b^{n-3} + \cdots + a_2\big) b + \boldsymbol{a_1}$$

por b, obteremos o resto a_1.

Repetindo este procedimento, obtemos sucessivamente os restos

$$\boldsymbol{a_0, a_1, a_2, \dots, a_n}.$$

Sistemas octal e hexadecimal

Base	a_i															
binária	0	1														
octal	0	1	2	3	4	5	6	7								
hexadecimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
											A	B	C	D	E	F

Tabela 1: Bases $b = 2$, $b = 8$ e $b = 16$.

Exemplos.

• *Conversão da base octal para a base decimal.*

$$(231)_8 = 2 \times 8^2 + 3 \times 8^1 + 1 \times 8^0 = 153$$

• *Conversão da base hexadecimal para a base decimal.*

$$\begin{aligned}(A2E)_{16} &= A \times 16^2 + 2 \times 16^1 + E \times 16^0 \\ &= 10 \times 16^2 + 2 \times 16^1 + 14 = 2606.\end{aligned}$$

Conversões binário $\leftrightarrow$ octal e binário $\leftrightarrow$ hexadecimal

Como

$$8 = 2^3 \quad \text{e} \quad 16 = 2^4$$

as conversões binário $\leftrightarrow$ octal e binário $\leftrightarrow$ hexadecimal podem ser feitas de forma quase imediata.

Por exemplo,

$$(231)_8 = (\underbrace{010}_{2}\ \underbrace{011}_{3}\ \underbrace{001}_{1})_2$$

$$(A2E)_{16} = (\underbrace{1010}_{A}\ \underbrace{0010}_{2}\ \underbrace{1110}_{E})_2$$

$$\begin{aligned}(11100110111)_2 &= (\underbrace{011}_{3}\ \underbrace{100}_{4}\ \underbrace{110}_{6}\ \underbrace{111}_{7})_2 = (3467)_8 \\ &= (\underbrace{0111}_{7}\ \underbrace{0011}_{3}\ \underbrace{0111}_{7})_2 = (737)_{16}\end{aligned}$$

Exercícios

1. *Quais as representações na base binária dos dígitos da base hexadecimal?*
2. *Represente os números* $(11111)_2$, $(100000)_2$ *e* $(100101)_2$ *na base decimal.*
3. *Obtenha as representações nas bases octal e hexadecimal dos números* 47 *e* 100.
4. *Represente o número* $(12BC)_{16}$ *na base decimal.*
5. *Obtenha a representação na base octal do número* $(10000111110101)_2$.
6. *Represente os números* $(123)_{16}$ $(1AB4)_{16}$ *na base binária.*

Soluções.

2. $(11111)_2 = 31$, $(100000)_2 = 32$, $(100101)_2 = 37$.
3. $47 = (57)_8 = (2F)_{16}$; $\quad (100)_{10} = (144)_8 = (64)_{16}$
4. $(12BC)_{16} = 4796$
5. $(10000111110101)_2 = (20765)_8$
6. $(123)_{16} = (1\,0010\,0011)_2$; $(1AB4)_{16} = (1\,1010\,1011\,0100)_2$

Representação de números em computadores

Por estarem envolvidos circuitos eletrónicos, um computador usa um número fixo de bits para representar números (8 bits, 16 bits, 32 bits, 64 bits, ...). Temos, portanto, um limite superior (e um limite inferior) para os números que podem ser representados.

Se um número tem menos algarismos binários, o número é complementado com zeros à esquerda. Por exemplo, o número 47, que precisa de 6 bits para ser representado na base binária, poderá ter num computador de 8 bits a representação

$$47 = (00101111)_2$$

(admitindo que o primeiro dígito 0 indica que o número é positivo).

Há necessidade de representar

- números **inteiros, positivos e negativos,** e
- números com **parte fracionária não nula**.

Precisamos de definir **esquemas de representação** usando apenas bits.

Sistemas de representação

Temos, para inteiros com sinal, representações em

- sinal e módulo
- em excesso de k (k também chamado de *offset binário*)
- complemento de 2
- complemento de 1,

e, para números reais, representações em

- vírgula fixa (ou ponto fixo)
- vírgula flutuante (ou ponto flutuante).

Representação de inteiros com *sinal e módulo*

Representação mais intuitiva (similaridade com a notação escrita).

Dado um número inteiro x, se considerarmos uma representação com t bits, o bit mais significativo (bit mais à esquerda) é reservado para o sinal,

$$0 \text{ para o sinal } + \quad \text{e} \quad 1 \text{ para o sinal } -,$$

e o valor absoluto $|x|$ é simplesmente representado em binário com $t-1$ bits.

Neste formato o valor absoluto máximo representável será $2^{t-1}-1$ e teremos a possibilidade de representar números inteiros satisfazendo

$$-\left(2^{t-1}-1\right) \leq x \leq 2^{t-1}-1.$$

Por exemplo, entre

(a) -7 e 7 para $t=4$;

(b) -32767 e 32767 para $t=16$;

(c) -2147483647 e 2147483647 para $t=32$.

Representação de inteiros com *sinal e módulo*

Com $t = 4$ bits podemos representar os inteiros de -7 a 7.

Metade dos números representados são não negativos e a outra metade corresponda a números negativos.

0	0	0	0	$\hookrightarrow +0$	1	0	0	0	$\hookrightarrow -0$
0	0	0	1	$\hookrightarrow +1$	1	0	0	1	$\hookrightarrow -1$
0	0	1	0	$\hookrightarrow +2$	1	0	1	0	$\hookrightarrow -2$
0	0	1	1	$\hookrightarrow +3$	1	0	1	1	$\hookrightarrow -3$
0	1	0	0	$\hookrightarrow +4$	1	1	0	0	$\hookrightarrow -4$
0	1	0	1	$\hookrightarrow +5$	1	1	0	1	$\hookrightarrow -5$
0	1	1	0	$\hookrightarrow +6$	1	1	1	0	$\hookrightarrow -6$
0	1	1	1	$\hookrightarrow +7$	1	1	1	1	$\hookrightarrow -7$

Um problema deste método é que o zero tem duas representações: $+0$ e -0.

Representação de inteiros *em excesso de k*

Numa representação *em excesso (ou enviesada)* é usado um parâmetro positivo chamado de *excesso* ou *offset binário*.

Para um dado valor k deste parâmetro, o menor número representável será $-k$ e o maior dependerá do número de bits t disponível.

Para representar um número x em excesso de k com t bits,

- obtemos um novo número $y = x + k$ e
- representamos y em t bits.

Note-se que teremos sempre $y \geq 0$ uma vez que estamos a assumir que $x \geq -k$.

Representação de inteiros *em excesso de* k

Numa representação em excesso de $k = 7$ com $t = 4$ bits temos possibilidade de representar os inteiros de -7 a 8, sendo que 0 terá uma representação única.

$$0 \leq y = x + 7 \leq 15 \quad \text{representa} \quad -7 \leq x \leq 8$$

0	0	0	0	$\hookrightarrow 0 - 7 = -7$
0	0	0	1	$\hookrightarrow 1 - 7 = -6$
0	0	1	0	$\hookrightarrow 2 - 7 = -5$
0	0	1	1	$\hookrightarrow 3 - 7 = -4$
0	1	0	0	$\hookrightarrow 4 - 7 = -3$
0	1	0	1	$\hookrightarrow 5 - 7 = -2$
0	1	1	0	$\hookrightarrow 6 - 7 = -1$
0	1	1	1	$\hookrightarrow 7 - 7 = 0$

1	0	0	0	$\hookrightarrow 8 - 7 = +1$
1	0	0	1	$\hookrightarrow 9 - 7 = +2$
1	0	1	0	$\hookrightarrow 10 - 7 = +3$
1	0	1	1	$\hookrightarrow 11 - 7 = +4$
1	1	0	0	$\hookrightarrow 12 - 7 = +5$
1	1	0	1	$\hookrightarrow 13 - 7 = +6$
1	1	1	0	$\hookrightarrow 14 - 7 = +7$
1	1	1	1	$\hookrightarrow 15 - 7 = +8$

O 0 é representado por k e $-k$ é representado por 0.

Observe-se que $k = 2^3 - 1 = 2^{t-1} - 1$

Representação de inteiros *em excesso de k*

Dado um número y em binário numa representação em excesso de k, interpretamos este número convertendo-o para decimal e subtraíndo k, por forma a obter o número *correspondente* x.

Dado um número y com t bits numa representação em excesso de $k = 2^{t-1} - 1$, temos (em decimal)

$$0 \leq y \leq 2^t - 1,$$

e y representa $x = y - (2^{t-1} - 1)$.

Assim, um número inteiro x representável neste formato satisfaz

$$-(2^{t-1} - 1) \leq x \leq 2^t - 1 - (2^{t-1} - 1).$$

ou seja,

$$-(2^{t-1} - 1) \leq x \leq 2^{t-1}$$

Representação de inteiros *em complemento de 2*

O *complemento de dois* (ou complemento para 2) de um número $x \geq 0$ com t bits é definido como o *complemento em relação a* 2^t, ou seja, é o valor

$$y = 2^t - x,$$

ou ainda, dito de outra forma, é o que falta a x para completar 2^t.

Numa representação em complemento de 2 o

número negativo $-x$ é representado por y

ou seja, pelo complemento de 2 do seu módulo.

Nota 1: Observe-se que 2^t em binário é representado por

$$1\underbrace{00\ldots00}_{t \text{ zeros}}.$$

Nota 2: Um outro método para calcular o complemento de 2 de $x \geq 0$ é calcular o *complemento de um* (operação bitwise NOT) e somar 1 ao valor, ou seja, obter $\underbrace{(2^t - 1) - x}_{\text{complemento de 1}} + 1$.

Representação de inteiros *em complemento de* 2

Numa representação em complemento de 2, se usarmos $t = 4$ bits podemos representar os números inteiros de -8 a 7, com unicidade na representação do número 0.

0	0	0	0	$\hookrightarrow 0$
0	0	0	1	$\hookrightarrow +1$
0	0	1	0	$\hookrightarrow +2$
0	0	1	1	$\hookrightarrow +3$
0	1	0	0	$\hookrightarrow +4$
0	1	0	1	$\hookrightarrow +5$
0	1	1	0	$\hookrightarrow +6$
0	1	1	1	$\hookrightarrow +7$

1	0	0	0	$\hookrightarrow 8 - 16 = -8$
1	0	0	1	$\hookrightarrow 9 - 16 = -7$
1	0	1	0	$\hookrightarrow 10 - 16 = -6$
1	0	1	1	$\hookrightarrow 11 - 16 = -5$
1	1	0	0	$\hookrightarrow 12 - 16 = -4$
1	1	0	1	$\hookrightarrow 13 - 16 = -3$
1	1	1	0	$\hookrightarrow 14 - 16 = -2$
1	1	1	1	$\hookrightarrow 15 - 16 = -1$

Se o dígito mais significativo for

- 0, o número é positivo e os bits seguintes representam o módulo do número (tal como numa representação de sinal e módulo);
- 1, o número é negativo e está representado o complemento de 2 do seu módulo, ou seja, para o número $-x$, $x \geq 0$, temos representado

$$y = 2^4 - x \text{ e, portanto, } -x = y - 2^4.$$

Representação de inteiros *em complemento de 2*

Também neste formato o bit mais significativo é o bit indicativo do sinal do número e existe unicidade na representação do zero.

Dado um número y com t bits, temos (em decimal)

$$0 \leq y \leq 2^t - 1.$$

Numa representação em complemento de 2, se

$$0 \leq y \leq 2^{t-1} - 1,$$

y representa o mesmo número positivo, $x = y$.

Se

$$2^{t-1} \leq y \leq 2^t - 1,$$

y representa o número negativo $-x = y - 2^t$, ou seja, o número

$$-2^{t-1} \leq -x \leq -1.$$

Assim, neste formato, com t bits podemos representar números inteiros positivos entre 0 e $2^{t-1} - 1$ e números negativos entre -2^{t-1} e -1, ou seja, números entre

$$-2^{t-1} \text{ e } 2^{t-1} - 1.$$

Representação de inteiros *em complemento de 2*

Uma das vantagens do uso do complemento de 2 é que as regras para a soma e a subtração são as mesmas.

Com $t = 4$, por exemplo, observe-se que a subtração $5 - 5 = 5 + (-5)$ corresponde a

$$5 + (2^4 - 5) = 2^4 \qquad (\textit{overflow}).$$

Em binário,

$$\begin{array}{rlr} 0101 & (5) & 5 \\ +\,1011 & (11) & \text{-}5 \\ \hline 10000 & (0) & 0 \end{array}$$

Descartando o bit mais significativo (só dispomos de 4 bits) obtemos o resultado pretendido.

Outros exemplos:

$$\begin{array}{rlr} 0101 & (5) & 5 \\ +\,1100 & (12) & \text{-}4 \\ \hline 10001 & (1) & 1 \end{array} \qquad \begin{array}{rlr} 0101 & (5) & 5 \\ +\,1010 & (10) & \text{-}6 \\ \hline 01111 & (15) & \text{-}1 \end{array}$$

Observações

- A representação de sinal e módulo foi usada em alguns computadores antigos, mas a duplicidade do valor zero e a "lógica" complicada para certas operações matemáticas resultaram na perda de popularidade (perda de tempo na manipulação dos sinais). Hoje a maior importância desta representação é a de ser a base para a representação em vírgula flutuante de números reais.

- Tipicamente quase todas as máquinas modernas usam a representação em complemento de 2 para a representação de inteiros, pois facilita as operações aritméticas. O algoritmo para tratar os sinais é simples e isso leva a maior rapidez na realização das operações aritméticas. É comum a representação com $t = 32$ bits na qual o número inteiro máximo é $2^{31} - 1 = 2147483647$ e o mínimo $-2^{31} = -2147483648$. (A velocidade das operações aritméticas numa representação em complemento de 1 é mais lenta.)

Observações

- Nos computadores modernos a representação em excesso é importante por fazer parte da representaçao de vírgula flutuante dos números reais. A representação em excesso usa-se para o valor do expoente.

- Numa representação em excesso de k, em princípio pode escolher-se qualquer valor positivo para o parâmetro k. No entanto, quanto maior o valor de k, mais números negativos podem ser representados, mas menos números positivos são possíveis de representar,

$$-k \leq x \leq (2^t - 1) - k.$$

  Geralmente, opta-se por um valor que equilibre a representatividade entre números positivos e números negativos, por exemplo, $k = 2^{t-1}$ ou $k = 2^{t-1} - 1$, quando se usam t bits.

Exercícios

1. *Expresse cada um dos seguintes números inteiros decimais nas representações de sinal e módulo e de complemento de 2, com 16 bits.*

   (a) -32767 (b) $+1024$ (c) -1 (d) $+242$

   Observe que $-32767 = -(2^{15} - 1)$ *e* $1024 = 2^{10}$.

2. *Usando 64 bits, qual o maior e o menor números que podem ser representados, supondo apenas números não negativos? E se utilizarmos a representação de sinal e módulo? E a de complemento de 2?*

Soluções.

1. (a) $sm = 1111111111111111$, $c2 = 1000000000000001$
   (b) $sm = 0000010000000000$, $c2 = 0000010000000000$
   (c) $sm = 1000000000000001$, $c2 = 1111111111111111$
   (d) $sm = 0000000011110010$, $c2 = 0000000011110010$

2. *Números não negativos:*
   $min = 0, \quad max = 2^{64} - 1 = 18446744073709551615$
   Números em SM: $min = -(2^{63} - 1) = -9223372036854775807$,
   $max = +(2^{63} - 1) = +9223372036854775807$
   Números em C2: $min = -2^{63} = -9223372036854775808$,
   $max = +(2^{63} - 1) = +9223372036854775807$

Exercícios

1. *Quais os números (na base decimal) que são representados em excesso de $k = 64$ com 8 bits por*

   (a) 00101101 (b) 00010101 (c) 01110101 (d) 11111111

   Quais são os números máximo e mínimo representáveis?

2. *Repita o exercício anterior mas agora considerando uma representação em excesso de $k = 2^7 - 1 = 127$.*

Soluções.

1. (a) -19
   (b) -43
   (c) 53
   (d) 191

   $min = -64$; $max = 191$

2. (a) -82
   (b) -106
   (c) -10
   (d) 128

   $min = -127$; $max = 128$

Representação de números reais numa base b ($b \geq 2$)

Um número real $x \neq 0$ é representado numa **base $b \geq 2$** por

$$x = *(a_n a_{n-1} \cdots a_1 a_0 . a_{-1} a_{-2} \cdots a_{-k})_b,$$

em que $* \in \{+, -\}$, $a_i \in \{0, 1, \ldots, b-1\}$, $a_n \neq 0$.

A conversão à base 10 obtém-se por intermédio da expressão

$$x = a_n \times b^n + \cdots + a_1 \times b + a_0 + a_{-1} \times b^{-1} + \ldots + a_{-k} \times b^{-k}.$$

Exemplos.

- *Conversão da base octal para a base decimal.*

$$(151.24)_8 = 1 \times 8^2 + 5 \times 8 + 1 + 2 \times 8^{-1} + 4 \times 8^{-2} = 105.3125$$

- *Conversão da base binária para a base decimal.*

$$(111.1011)_2 = 2^2 + 2 + 1 + 2^{-1} + 2^{-3} + 2^{-4} = 7.6875$$

Mudança de um número fracionário para uma base $\boldsymbol{b}$

Dado um número fracionário $x \neq 0$ ($|x| < 1$) na base 10 e uma base $b \geq 2$, pretende-se determinar os dígitos $a_{-1}, a_{-2}, \ldots, a_{-k} \in \{1, 2, \ldots, b-1\}$ tais que

$$x = a_{-1} \times b^{-1} + a_{-2} \times b^{-2} + a_{-3} \times b^{-3} + \ldots + a_{-k} \times b^{-k},$$

ou seja, tais que $x = (0.a_{-1}a_{-2} \cdots a_{-k})_b$.

Observe-se que vamos poder escrever

$$x \times b = a_{-1} + a_{-2} \times b^{-1} + a_{-3} \times b^{-2} + \ldots + a_{-k} \times b^{-k+1}.$$

Assim, se considerarmos a multiplicação de x por b, a_{-1} será a parte inteira do resultado dessa multiplicação e a parte fracionária será

$$x_0 := a_{-2} \times b^{-1} + a_{-3} \times b^{-2} + \ldots + a_{-k} \times b^{-k+1}.$$

Se novamente multiplicarmos x_0 por b,

$$x_0 \times b = a_{-2} + a_{-3} \times b^{-1} + \ldots + a_{-k} \times b^{-k+2},$$

obteremos a_{-2}.

Repetindo este procedimento obtemos sucessivamente

$$\boldsymbol{a_{-1}, a_{-2}, \ldots, a_{-k}}.$$

Representação de números reais numa base b

Exemplos.

- *Conversão de* $x = 0.3125$ *para a base 8:*

$$0.3125 \times 8 = 2.5000$$
$$0.5000 \times 8 = 4.0000$$

$$0.3125 = (0.24)_8$$

- *Conversão de* $x = 0.6875$ *para a base binária.*

$$0.6875 \times 2 = 1.3750$$
$$0.3750 \times 2 = 0.7500$$
$$0.7500 \times 2 = 1.5000$$
$$0.5000 \times 2 = 1.0000$$

$$0.6875 = (0.1011)_2$$

- *Conversão de* $x = 9.6875$ *para a base binária:*

$$9.6875 = (1001.1011)_2$$

Representação de números reais em vírgula flutuante

Precisamos de um esquema de representação específico para números não inteiros ("números com vírgula") que permita lidar facilmente com números muito grandes e números (frações) muito pequenos.

Ficaria difícil representar na notação de ponto fixo valores como

$$2138000000000000000000000000000$$
$$0,00000000000000000000000000091$$

e mais ainda a sua adição ou multiplicação.

Por ser necessário uma quantidade de bits maior, precisamos de usar uma notação especial a que chamamos notação científica ou notação de vírgula (ou ponto) flutuante.

Representação de números reais em vírgula flutuante

Dado $x \in \mathbb{R}$, $x \neq 0$, e fixada uma base $b \geq 2$, x admite uma representação da forma

$$\begin{aligned} x &= * \left(\sum_{k=1}^{\infty} d_k \times b^{-k} \right) \times b^{e} \\ &= (d_1 \times b^{-1} + d_2 \times b^{-2} + d_3 \times b^{-3} + \cdots) \times b^{e} \end{aligned}$$

onde $* \in \{+, -\}$, $d_k \in \{0, 1, \ldots, b-1\}$, $d_1 \neq 0$, $e \in \mathbb{Z}$.

$$x = *(\underbrace{0.d_1 d_2 d_3 \ldots}_{\text{mantissa}})_b \times b^{e} \longleftarrow \text{expoente}$$

Sistema de numeração de vírgula flutuante

Num computador o número de dígitos da mantissa é fixo (t dígitos) e o expoente é limitado por um valor mínimo (m) e um valor máximo (M).

Assim, os números de máquina são os que podem ser escritos na forma

$$x = *(\underbrace{0.d_1 d_2 \dots d_{t-1} d_t}_{\text{mantissa}})_b \times b^{\boldsymbol{e}}, \quad d_1 \neq 0, \quad m \leq \boldsymbol{e} \leq M,\ \boldsymbol{e} \in \mathbb{Z},$$

juntamente com o número zero.

Estes são chamados os números normalizados.

Sistema de numeração de vírgula flutuante

Um sistema de numeracão de vírgula flutuante

$$F(b, t, m, M)$$

é assim caracterizado por quatro parâmetros: a base b, o número de dígitos da mantissa t, o valor mínimo m e o valor máximo M para o expoente.

Um sistema de numeração pode ainda admitir os chamados números desnormalizados ou subnormais, que são os números que se obtêm deixando de impor a condição $d_1 \neq 0$ quando o expoente assume o valor mínimo, ou seja, os números da forma

$$x = *(0.0d_2 \dots d_{t-1}d_t)_b \times b^{\boldsymbol{m}}.$$

Sistema de numeração $F(b,t,m,M)$

- O maior número positivo de $F(b,t,m,M)$ é

$$\begin{aligned}\Omega &:= \big(0.(b-1)(b-1)\dots(b-1)(b-1)\big)_b \times b^{\boldsymbol{M}} \\ &= (1-b^{-t})b^{\boldsymbol{M}}\end{aligned}$$

  a que chamamos nível de overflow.

  Observe que

$$\sum_{k=1}^{t}(b-1)b^{-k} = (b-1)\sum_{k=1}^{t} b^{-k} = (b-1)\cdot b^{-1}\cdot \frac{1-\left(b^{-1}\right)^{t}}{1-b^{-1}} = 1-b^{-t}.$$

- O menor número positivo normalizado, chamado nível de underflow, é

$$\begin{aligned}\omega &:= \big(0.10\dots00\big)_b \times b^{\boldsymbol{m}} \\ &= b^{-1}\times b^{\boldsymbol{m}} = b^{\boldsymbol{m}-1}.\end{aligned}$$

- Ao conjunto

$$R_F := [-\Omega, -\omega] \cup \{0\} \cup [\omega, \Omega]$$

  damos o nome de conjunto dos números representáveis.

Sistemas de numeração $F(b, t, m, M)$

- O menor número positivo de um sistema que admite números desnormalizados é

$$\omega^* = (0.00\ldots01)_b \times b^{m} = b^{m-t}.$$

Se nada for dito em contrário, quando nos referimos a um sistema $F(b, t, m, M)$, consideramos apenas os números normalizados.

Sistemas de numeração $F(b, t, m, M)$

Exemplos.

1. $F(10, 3, -2, 4)$

   $\Omega = 0.999 \times 10^4 = (1 - 10^{-3}) \times 10^4 = 9990$

   $\omega = 0.100 \times 10^{-2} = 10^{-2-1} = 10^{-3} = 0.001$

2. $F(2, 4, -2, 2)$

   $\Omega = (0.1111)_2 \times 2^2 = (1 - 2^{-4}) \times 2^2 = 3.75$

   $\omega = (0.1000)_2 \times 2^{-2} = 2^{-2-1} = 2^{-3} = 0.125$

   Quais os números deste sistema que têm expoente igual a -1?

$$
\begin{array}{ll}
(0.1000)_2 \times 2^{-1} & (0.1100)_2 \times 2^{-1} \\
(0.1001)_2 \times 2^{-1} & (0.1101)_2 \times 2^{-1} \\
(0.1010)_2 \times 2^{-1} & (0.1110)_2 \times 2^{-1} \\
(0.1011)_2 \times 2^{-1} & (0.1111)_2 \times 2^{-1}
\end{array}
$$

Arredondamento

Seja $F = F(10, 3, -2, 4)$ e $x = 0.53142$. Observe-se que

$$x \in R_F \text{ mas } x \notin F,$$

ou seja, $F \subseteq R_F$.

Dado $x \in R_F$, como representá-lo no sistema?

Dado $x \in R_F$, $fl(x)$ designa o número de $F(b, t, m, M)$ obtido (salvo indicação em contrário)

$$\text{somando } \frac{1}{2}b^{-t} \text{ à mantissa de } x$$

e truncando o resultado para t dígitos.

Em caso de underflow, isto é, quando $|x| < \omega$, consideramos $fl(x) = 0$.

Arredondamento

Observe que

$$\frac{1}{2}b^{-t} = \frac{b}{2} \cdot b^{-t-1} = \frac{b}{2} \cdot b^{-(t+1)}$$

e, portanto, o que fazemos é a soma

$$\begin{array}{r} 0.d_1 d_2 \ldots d_{t-1} d_t d_{t+1} d_{t+2} \ldots \\ +\ 0.0\ \ 0\ \ \ldots\ \ \ 0\ \ \ \ 0\ (b/2) \end{array}$$

e depois truncamos o resultado para t dígitos.

Este arredondamento corresponde ao arredondamento para o número de F mais próximo de x e, em caso de empate, ao arredondamento por excesso.

Arredondamento

Exemplos.

1. $F(10, 3, -2, 4)$

   $fl(0.53142) = fl(0.53142 \times 10^0) = 0.531 \times 10^0$

   $fl(12.252) = fl(0.12252 \times 10^2) = 0.123 \times 10^2$

   $fl(0.045601) = fl(0.45601 \times 10^{-1}) = 0.456 \times 10^{-1}$

2. $F(2, 4, -2, 2)$

   $fl\big((0.110001)_2 \times 2^{-1}\big) = (0.1100)_2 \times 2^{-1}$

   $fl\big((0.110011)_2 \times 2^{-1}\big) = (0.1101)_2 \times 2^{-1}$

   $fl\big((10.0111)_2\big) = fl\big((0.100111)_2 \times 2^2\big) = (0.1010)_2 \times 2^2$

Epsilon da máquina

Chama-se epsilon da máquina e denota-se por ε, a diferença entre o número de $F(b, t, m, M)$ imediatamente superior a 1 e o número 1, isto é,

$$\varepsilon := b^{1-t}.$$

De facto, temos

$$1 = (1)_b = \underbrace{(0.10\ldots00)_b}_{t \text{ dígitos}} \times b^1$$

e o número imediatamente superior a 1 é o número

$$\underbrace{(0.10\ldots01)_b}_{t \text{ dígitos}} \times b^1.$$

Ou seja, a diferença é igual a

$$\underbrace{(0.000\ldots01)_b}_{t \text{ dígitos}} \times b^1 = b^{1-t}.$$

Unidade de erro de arredondamento

A unidade de erro de arredondamento do sistema (ou precisão da máquina) é

$$\mu := \frac{1}{2}\varepsilon = \frac{1}{2}b^{1-t} = \frac{b}{2} \cdot b^{-t}$$

Podemos provar que dado um número $x \in R_F$ e sendo $fl(x)$ o número do sistema que representa x (a aproximação para x do sistema), o erro relativo (em valor absoluto) desta aproximação é majorado por μ, ou seja, temos sempre

$$\left| \frac{x - fl(x)}{x} \right| \leq \mu$$

qualquer que seja a grandeza de x.

Exemplos.

1. $F(10, 3, -2, 4)$

   $\varepsilon = 10^{1-3} = 10^{-2} = 0.01$ *e* $\mu = \frac{1}{2} \times 10^{-2} = 0.005$

   Repare-se que neste sistema, 1 *tem a representação* 0.100×10^1 *e o número imediatamente superior é*

   $$0.101 \times 10^1 = 0.100 \times 10^1 + 0.001 \times 10^1 = 1 + \varepsilon.$$

2. $F(2, 4, -2, 2)$

   $\varepsilon = 2^{1-4} = 2^{-3} = 0.125$ *e* $\mu = \frac{1}{2} \times 2^{-3} = 0.0625$

   Para este exemplo, temos $(1)_{10} = (0.1000)_2 \times 2^1$ *e número imediatamente a seguir é*

   $$(0.1001)_2 \times 2^1 = (0.1000)_2 \times 2^1 + (0.0001)_2 \times 2^1 = 1 + \varepsilon.$$

Operações de vírgula flutuante

As operações num computador não são realizadas de forma exata.

Representaremos as operações de vírgula flutuante pelo símbolo usual rodeado por um círculo; por exemplo, $\oplus$, $\otimes$.

Temos o seguinte modelo: admitimos que o resultado de uma operação de vírgula flutuante é obtido por arredondamento do resultado da operação exata, isto é,

$$x \oplus y = fl(x + y), \qquad x \otimes y = fl(x \times y), \quad \text{etc.}$$

As operações de máquina de adição e subtração exigem que os números sejam por vezes temporariamente desnormalizados para que os operandos estejam numa representação com o mesmo expoente, isto para que as mantissas possam ser alinhadas.

Depois de realizada a operação, o resultado volta a ser normalizado. Neste processo podem perde-se dígitos e por conseguinte precisão, uma vez que o número de dígitos da mantissa está fixo.

O modelo que assumimos exige que se tenha um bit extra ou *bit de guarda* (*guard digit*) para a mantissa.

Exemplo.

Sejam $F = F(10, 4, -99, 99)$ *e* $x = 0.5289$, $y = 0.8012$ *e* $z = 0.6024$.

Quais os resultados das operações

$$x \oplus (y \oplus z) \quad \text{e} \quad (x \oplus y) \oplus z?$$

$$\begin{aligned} y \oplus z &= fl(0.8012 + 0.6024) = fl(1.4036) \\ &= fl(0.14036 \times 10^1) = 0.1404 \times 10^1 \end{aligned}$$

$$\begin{aligned} x \oplus 0.1404 \times 10^1 &= fl(0.5289 + 0.1404 \times 10^1) \\ &= fl(0.05289 \times 10^1 + 0.1404 \times 10^1) \\ &= fl(0.19329 \times 10^1) = 0.1933 \times 10^1 \end{aligned}$$

Assim, $x \oplus (y \oplus z) = 0.1933 \times 10^1$. *Observe-se que para se obter este resultado foi preciso conservar um dígito extra quando se desnormalizou.*

De forma semelhante obtemos

$(x \oplus y) \oplus z = 0.1932 \times 10^1$.

Norma IEEE 754

Em 1985 uma norma (ou padrão) de um sistema binário de vírgula flutuante foi publicada com o objetivo de uniformizar as operações nos sistemas de vírgula flutuante. Esta norma ficou conhecida como a Norma IEEE, uma vez que foi desenvolvida pelo Institute for Electrical and Electronics Engineers, e, mais importante, foi seguida cuidadosamente pelos fabricantes de computadores.

A Norma IEEE tem três requisitos muito importantes:

- representação consistente de números de vírgula flutuante em todas as máquinas que adoptem esta norma;
- arredondamento correto de operações aritméticas;
- tratamento consistente de situações excecionais tais como uma divisão por zero.

Norma IEEE 754

- Formato simples (32 *bits*)

s	e_1	e_2	$\cdots$	e_7	e_8	d_1	d_2	d_3	$\cdots$	d_{21}	d_{22}	d_{23}

1 bit para o sinal ($s = 0$ - número positivo; $s = 1$ - número negativo)
8 bits para o expoente (em excesso de 127)
23 bits para a *mantissa*

Os números admitem uma representação da forma

$$x = *(\underbrace{1.d_1 d_2 \ldots d_{22} d_{23}}_{\text{mantissa}})_2 \times 2^{\mathbf{e}} \overset{}{\longleftarrow} \text{expoente} \qquad (d_0 = 1;\ \text{bit implícito})$$

$$= (-1)^s (1 + d_1 \times 2^{-1} + d_2 \times 2^{-2} + \cdots + d_{22} \times 2^{-22} + d_{23} \times 2^{-23}) \times 2^{\mathbf{e}}$$

com $* \in \{+, -\}$, $d_k \in \{0, 1\}$, $-126 \leq \mathbf{e} \leq 127$, $\mathbf{e} \in \mathbb{Z}$.

É usada uma representação em excesso de $2^7 - 1 = 127$ para armazenar o expoente. Assim, $\mathbf{e} = (e_1 e_2 \cdots e_7 e_8)_2 - 127$.

Norma IEEE 754

- Formato duplo (64 *bits*)

s	e_1	e_2	$\cdots$	e_{10}	e_{11}	d_1	d_2	d_3	$\cdots$	d_{50}	d_{51}	d_{52}

1 bit para o sinal ($s = 0$ - número positivo; $s = 1$ - número negativo)
11 bits para o expoente (em excesso de 1023)
52 bits para a *mantissa*

Os números admitem uma representação da forma

$$x = *(\underbrace{1.d_1 d_2 \ldots d_{51} d_{52}}_{\text{mantissa}})_2 \times 2^{\mathbf{e}} \xleftarrow{} \text{expoente} \qquad (d_0 = 1;\ \text{bit implícito})$$

$$= (-1)^s (1 + d_1 \times 2^{-1} + d_2 \times 2^{-2} + \cdots + d_{51} \times 2^{-51} + d_{52} \times 2^{-52}) \times 2^{\mathbf{e}}$$

com $* \in \{+, -\}$, $d_k \in \{0, 1\}$, $-1022 \leq \mathbf{e} \leq 1023$, $\mathbf{e} \in \mathbb{Z}$.

É usada uma representação em excesso de $2^{10} - 1 = 1023$ para armazenar o expoente. Assim, $\mathbf{e} = (e_1 e_2 \cdots e_{10} e_{11})_2 - 1023$.

Alocação de bits

	formato simples	formato duplo
bits do expoente	8	11
bits da mantissa $(t-1)$	23	52
$\mathbf{e}_{min}$	-126	-1022
$\mathbf{e}_{max}$	127	1023

Não há bit reservado para o sinal do expoente. Usa-se uma representação em excesso (temos, portanto, um *expoente enviesado*).

No formato simples temos 8 bits para o expoente.
Numa representação em excesso de $k = 2^7 - 1 = 127$ podemos representar inteiros entre -127 e 128.

Os expoentes -127 e 128 (que correspondem às representações sem sinal 0 e 255) assinalam, respetivamente, um número desnormalizado e "exceções da aritmética IEEE" como $+$Inf, -Inf e NaN. O expoente 128 e $d_1 = d_2 = \ldots = d_{23} = 0$ assinalam $\pm\infty$; e se algum $d_i \neq 0$, NaN.

No formato duplo temos 11 bits para o expoente e numa representação em excesso de $k = 2^{10} - 1 = 1023$ podemos representar inteiros entre -1023 e 1024. De forma análogo ao formato simples, os expoentes -1023 e 1024 estão reservados para assinalar situações especiais.

Norma IEEE 754 - formato simples

s	e_1	e_2	$\cdots$	e_7	e_8	d_1	d_2	d_3	$\cdots$	d_{21}	d_{22}	d_{23}

$$x = (-1)^s (1.d_1 d_2 \dots d_{51} d_{52})_2 \times 2^{\mathbf{e}}, \qquad \mathbf{e} = (e_1 e_2 \cdots e_7 e_8)_2 - 127$$

$e_1 e_2 \cdots e_7 e_8$	valor representado
$(00000000)_2 = (0)_{10}$	$\pm(0.d_1 d_2 \dots d_{22} d_{23})_2 \times 2^{\text{-126}}$
$(00000001)_2 = (1)_{10}$	$\pm(1.d_1 d_2 \dots d_{22} d_{23})_2 \times 2^{\text{-126}}$
$(00000010)_2 = (2)_{10}$	$\pm(1.d_1 d_2 \dots d_{22} d_{23})_2 \times 2^{\text{-125}}$
$\vdots$	$\vdots$
$(01111111)_2 = (127)_{10}$	$\pm(1.d_1 d_2 \dots d_{22} d_{23})_2 \times 2^{\mathbf{0}}$
$(10000000)_2 = (128)_{10}$	$\pm(1.d_1 d_2 \dots d_{22} d_{23})_2 \times 2^{\mathbf{1}}$
$\vdots$	$\vdots$
$(11111101)_2 = (253)_{10}$	$\pm(1.d_1 d_2 \dots d_{22} d_{23})_2 \times 2^{\mathbf{126}}$
$(11111110)_2 = (254)_{10}$	$\pm(1.d_1 d_2 \dots d_{22} d_{23})_2 \times 2^{\mathbf{127}}$
$(11111111)_2 = (255)_{10}$	$\pm\infty$ se $d_1 = d_2 = \cdots = d_{23} = 0$ NaN, nos outros casos

Sistemas de vírgula flutuante correspondentes

$$\begin{aligned} x &= *(1.d_1 d_2 \dots d_{t-2} d_{t-1})_2 \times 2^{\mathbf{e}} \\ &= *(0.1 d_1 d_2 \dots d_{t-2} d_{t-1})_2 \times 2^{\mathbf{e+1}} \longleftarrow \text{expoente} \end{aligned}$$

- Formato simples $(t = 24)$: $F(2, 24, -125, 128)$
- Formato duplo $(t = 53)$: $F(2, 53, -1021, 1024)$

Observação. Nestes sistemas de numeração de vírgula flutuante, com a base binária, o primeiro dígito da mantissa é sempre igual a 1 e o hardware pode ser construído assumindo este bit sem ser necessário representá-lo. Por esta razão se diz que o bit é implícito.

No quadro seguinte estão resumidas as características destes sistemas de numeração.

	formato simples $F(2, 24, -125, 128)$	formato duplo $F(2, 53, -1021, 1024)$
epsilon da máquina	$2^{1-24} = 2^{-23}$	$2^{1-53} = 2^{-52}$
maior número normalizado	$(1 - 2^{-24})2^{128}$	$(1 - 2^{-53})2^{1024}$
menor número positivo normalizado	$2^{-1-125} = 2^{-126}$	$2^{-1-1021} = 2^{-1022}$
menor número positivo desnormalizado	$2^{-24-125} = 2^{-149}$	$2^{-53-1021} = 2^{-1074}$

Sistema de numeração IEEE

- O sistema IEEE admite números desnormalizados, implementando, assim, a técnica de *underflow gradual*.

  Quando é necessário desnormalizar um número, o sistema tem de assinalar esta situação e para isso usa um expoente de bits todos iguais a zero. Neste caso o expoente toma o valor mínimo, $\mathbf{e} = 2^{-126}$.

- Temos duas representações diferentes para o zero (-0 e $+0$) mas as comparações são definidas de forma que sejam iguais; alega-se que a inclusão destes *números especiais* torna mais fácil obter precisão numérica em alguns problemas críticos.

- As regras de arredondamento a utilizar são também especificadas. Por defeito, é utilizado o chamado arredondamento para par, isto é, dado um número $x \in R_F$, $fl(x)$ é escolhido como o número de máquina mais próximo de x, sendo, em caso de empate, escolhido aquele que tem o último bit da mantissa igual a zero.

Sistema de numeração IEEE

O sistema IEEE é um sistema fechado: toda a operação aritmética produz um resultado, quer seja matematicamente esperada ou não, e "operações excecionais" são assinaladas. As respostas por defeito são mostradas na tabela seguinte.

exceção	exemplo	resultado
operação inválida	$0/0$, $0 \times \infty$, ∞/∞, $\sqrt{-1}$	NaN (Not a Number)
overflow		$\pm\infty$ ($\pm$Inf)
Divisão por zero	número finito/0	$\pm\infty$
underflow		números desnormalizados

Overflow: se $x > \Omega$, $fl(x) = \text{Inf}$ e se $x < -\Omega$ $fl(x) = -\text{Inf}$.

Underflow: se $2^{(\mathsf{e}_{min}+1)-t} \leq |x| < \omega$, $fl(x)$ será o número desnormalizado mais próximo de x; se $|x| < 2^{\mathsf{e}_{min}+1-t}$, $fl(x) = 0$.

Nota. $+$Inf e $-$Inf podem surgir em resultados intermédios e o resultado final estar correto; se NaN surgir como resultado intermédio, o resultado final será sempre NaN.

Exemplos.

Qual o número na base decimal com a seguinte representação no formato de precisão simples especificado pela norma IEEE 754?

(a) 0 1 0 0 0 0 1 0 0 0 0 1 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

(b) 1 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

(c) 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

(d) 0 1 1 1 1 1 1 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Solução.

(a) *Sinal:* $+$

Expoente: $(10000100)_2 - 127 = (2^7 + 2^2) - 127 = 132 - 127 = 5$

Mantissa: $(1.00110011)_2 = 1 + 2^{-3} + 2^{-4} + 2^{-7} + 2^{-8}$

$$\begin{aligned} x &= (1 + 2^{-3} + 2^{-4} + 2^{-7} + 2^{-8}) \times 2^5 \\ &= 2^5 + 2^2 + 2^1 + 2^{-2} + 2^{-3} = 38.375 \end{aligned}$$

(b) *Como o expoente é zero, trata-se de um número desnormalizado.*

$$x = -(0.1101)_2 \times 2^{-126}$$

(c) *Como todos os bits do expoente são iguais a 1, os bits da mantissa são iguais a zero e o bit de sinal é igual a zero, trata-se da indicação da exceção* $+\infty$.

(d) *Como todos os bits do expoente são iguais a 1 e os bits da mantissa não são todos iguais a zero, trata-se da indicação da exceção NaN.*

Exercício

1. *Obtenha a representação dos seguintes números decimais no formato simples especificado pela norma IEEE 754.*

   1.1 $+0.0625$ 1.2 -12.1 1.3 -21322.0 1.4 $+140.9375$

   Solução.

   1.1 0 01111011 00000000000000000000000
   1.2 1 10000010 10000011001100110011010
   1.3 1 10001101 01001101001010000000000
   1.4 0 10000110 00011001111000000000000

Exercício

2. *Considere o formato simples (32 bits) especificado pela norma IEEE 754.*

   2.1 *Obtenha o número na base decimal com a seguinte representação:*

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

   2.2 *O número 2^{-130} é representável neste sistema? Em caso afirmativo, escreva a sua representação.*

   2.3 *Qual o resultado da operação de máquina $2 \bigotimes 2^{127}$? A que representação corresponde?*

Solução.

2.1 212.75

2.2 0 00000000 00010000000000000000000

2.3 0 11111111 00000000000000000000000

Por defeito, o Matlab trabalha no sistema de numeração IEEE em formato duplo, isto é, no sistema $F(2, 53, -1021, 1024)$. No entanto, é possível também trabalhar em precisão simples e com dados do tipo inteiro. Por exemplo, as respostas ao exercício anterior podem ser obtidas no Matlab usando os seguintes comandos,

```
>> format hex
>> single(0.0625)
>> single(-12.1)
>> single(-21322.0)
>> single(140.9375)
```

Obtemos as respostas

1.1 3d800000
1.2 c141999a
1.3 c6a69400
1.4 430cf000

que são as representações dos números no computador, num sistema de vírgula flutuante de precisão simples IEEE apresentadas num *formato de display hexadecimal*. Para obter as sequências de bits basta substituir cada dígito hexadecimal pela sua representação binária (em 4 bits).

Bibliografia

1. *Métodos numéricos*.
   Heitor Pina. McGraw-Hill, D.L. (2003). [Capítulo 1]

2. *Métodos numéricos, complementos e guia prático*.
   Carlos Lemos e Heitor Pina. Instituto Superior Técnico (2006).

3. *Numerical Computing with IEEE floating point arithmetic.*
   Michael L. Overton. SIAM editor (2001).

4. *Accuracy and Stability of Numerical Algorithms.*
   Nicholas J. Higham. SIAM editor (1996). [Capítulo 2]

5. Help Center Documentation, The Mathworks Inc.
   `https://www.mathworks.com/help/matlab/index.html`