COMPUTAÇÕES NUMÉRICAS

1.0– Representação

O sistema de numeração decimal é o mais usado pelo homem nos dias de hoje. O número 10 tem papel fundamental, é chamado de base do sistema. Os símbolos 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, são usados para representar qualquer grandeza. O fato de o sistema decimal ser largamente utilizado tem evidentemente razões históricas, pois na realidade qualquer número inteiro maior que 1 poderia ter sido escolhido. De fato, no mundo dos computadores digitais o sistema binário é o utilizado. O número 2 é a base do sistema e os símbolos 0 e 1 servem para representar uma grandeza qualquer. Ao lado do sistema binário, os sistemas octal, hexadecimal, base 8 e 16 respectivamente, são também utilizados. Isto ocorre pelo fato de que cada símbolo octal e hexadecimal representa um equivalente a três e quatro símbolos no sistema binário e vice-versa.

1-1- Conversão

Dado um número x representado na base N, isto é, na N-representação, e nós queremos saber como representa-lo na base M, isso é, na M-representação. Temos então a equação: $x = a_mN^m + a_{m-1}N^{m-1} + ... = b_nM^n + b_{n-1}M^{n-1} + ...$ onde os coeficientes a_m ,a_{m-1} , ... são conhecidos e os coeficientes b_n , b_{n-1} , ... devem ser determinados. Observe que b_n , b_{n-1} , ... devem ser expressos com símbolos de dígitos da N-representação. Para realizar a conversão dividiremos x em uma parte inteira i e uma parte fracionária f. Nós temos $i = b_nM^n + b_{n-1}M^{n-1} + ... + b_1M^1 + b_0$, e dividindo i por M nós obtemos um quociente q_1 e um resto $r_1 = b_0$. Continuando, dividiremos q_1 por M, nós conseguiremos q_2 e o resto $r_2 = b_1$, e, obviamente, b_0, b_1, b_2, ... são os restos consecutivos quando i é dividido repetitivamente por M. De forma semelhante nós encontramos a parte fracionária como as partes inteiras consecutivas quando f é multiplicado repetitivamente por M e a parte inteira é removida. Os cálculos devem ser feitos na N-representação e M deve ser também dado nesta representação.

Exemplo: Converta o número decimal 261,359 para a representação binária, ternária e octal.

Conversão: Decimal para binário

Inteira: Divisão sucessiva do número decimal por 2

Divisão	Resto
261:2	1
130:2	0
65:2	1
32:2	0
16:2	0
8:2	0
4:2	0
2:2	0

1:2	1

O número inteiro binário é obtido através dos restos das divisões escritos na ordem inversa da sua obtenção. Então $(261)_{10} = (1.0000.0101)_2$.

Fração: Multiplicação sucessiva da fração decimal por 2

Multiplicação	**Sobra**
0,359x2	0
0,718x2	1
0,436x2	0
0,872x2	1
0,774x2	1
0,488x2	0
0,976x2	1
0,952x2	1
0,904x2	1

A fração binária é obtida través das sobras, parte inteira, das multiplicações escritas na ordem direta de sus obtenção. Então $(0,359)_{10} = (0,0101.1011.1 ...)_2$.

Somando-se as partes inteiras e fracionárias dos binários obtidos têm-se

$$(261,359)_{10} = (1.0000.0101, 0101.1011.1 ...)_2$$

Conversão: Decimal para ternário

Inteira: Divisão sucessiva do número decimal por 3

Divisão	**Resto**
261:3	0
87:3	0
29:3	2
9:3	0
3:3	0
1:3	1

O número inteiro ternário é obtido através dos restos das divisões escritos na ordem inversa da sua obtenção. Então $(261)_{10} = (100.200)_3$.

Fração: Multiplicação sucessiva da fração decimal por 3

Multiplicação	**Sobra**
0,359x3	1
0,077x3	0
0,231x3	0

0,693x3	2
0,079x3	0
0,273x3	0
0,711x3	2
0,133x3	0
0,399x3	1

A fração ternária é obtida través das sobras, parte inteira, das multiplicações escritas na ordem direta de sus obtenção. Então $(0,359)_{10} = (0,100.200.201 ...)_3$.

Somando-se as partes inteiras e fracionárias dos binários obtidos têm-se

$$(261,359)_{10} = (100.200, 100.200.201 ...)_3$$

Conversão: Decimal para hexadecimal

Inteira: Divisão sucessiva do número decimal por 16

Divisão	**Resto**
261:16	5
16:16	0
1:16	1

O número inteiro hexadecimal é obtido através dos restos das divisões escritos na ordem inversa da sua obtenção. Então $(261)_{10} = (105)_{16}$.

Fração: Multiplicação sucessiva da fração decimal por 16

Multiplicação	**Sobra**
0,359x16	5
0,744x16	11
0,904x16	14
0,464x16	7
0,424x16	6
0,784x16	12
0,544x16	8
0,704x16	11
0,264x16	4

A fração hexadecimal é obtida través das sobras, parte inteira, das multiplicações escritas na ordem direta de sus obtenção. Então

$$(0,359)_{10} = (0,5\ \underline{11}\ \underline{14}\ 7\ 6\ \underline{12}\ 8\ \underline{11}\ 4...)_{16}$$

ou utilizando-se os símbolos hexadecimais (tabela 1.1)

$$(0{,}359)_{10} = (0{,}5BE.76C.8B4...)_{16}$$

Somando-se as partes inteiras e fracionárias dos binários obtidos têm-se

$$(261{,}359)_{10} = (105,\ 5BE.76C.8B4...)_{16}$$

Tabela 1.1 – Símbolos Hexadecimais

Grandeza Decimal	**Símbolo hexadecimal**
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F

Programa 1.1 – Conversão de Inteiros na Base 10 para base qualquer

```
IDec(N, bx) := | VS ← ""
               | i ← 0
               | while N > 0
               |     V_i ← mod(N, bx)
               |     N ← trunc(N / bx)
               |     i ← i + 1
               | n ← i − 1
               | V ← reverse(V)
               | for i ∈ 0..n
               |     VS ← concat(VS, num2str(V_i))
               |     VS ← concat(VS, ".")
               | VS
```

Exemplos:

IDec(261, 2) = "1.0.0.0.0.0.1.0.1."

IDec(261, 3) = "1.0.0.2.0.0."

IDec(261, 16) = "1.0.5."

Programa 1.2 – Conversão de Fracionários na Base 10 para base qualquer

```
FDec(F, bx, n) := | VS ← "0,"
                  | i ← 0
                  | while F > 0
                  |     | V ← trunc(F·bx)
                  |     | VS ← concat(VS, num2str(V))
                  |     | VS ← concat(VS, "." )
                  |     | F ← |V − F·bx|
                  |     | i ← i + 1
                  |     | break if i = n
                  | VS
```

Exemplos:

FDec(0.359, 2, 9) = "0,0.1.0.1.1.0.1.1.1."

FDec(0.359, 3, 9) = "0,1.0.0.2.0.0.2.0.1."

FDec(0.359, 16, 9) = "0,5.11.14.7.6.12.8.11.4."

Programa 1.3 – Conversão da Base 10 para base qualquer

```
XDec(X, bx, n) := | N ← trunc(X)
                  | if N > 0
                  |     VS ← ""
                  |     i ← 0
                  |     while N > 0
                  |         V_i ← mod(N, bx)
                  |         N ← trunc(N / bx)
                  |         i ← i + 1
                  |     nx ← i − 1
                  |     V ← reverse(V)
                  |     for i ∈ 0..nx
                  |         VS ← concat(VS, num2str(V_i))
                  |         VS ← concat(VS, ".")
                  | VS ← "0"  otherwise
                  | F ← |X − trunc(X)|
                  | if F > 0
                  |     VS ← concat(VS, ",")
                  |     i ← 0
                  |     while F > 0
                  |         V ← trunc(F·bx)
                  |         VS ← concat(VS, num2str(V))
                  |         VS ← concat(VS, ".")
                  |         F ← |V − F·bx|
                  |         i ← i + 1
                  |         break if i = n
                  | VS
```

Exemplos: XDec(261, 2, 9) = "1.0.0.0.0.0.1.0.1." XDec(0.359, 2, 9) = "0,0.1.0.1.1.0.1.1.1."

XDec(261.359, 2, 9) = "1.0.0.0.0.0.1.0.1.,0.1.0.1.1.0.1.1.1."

XDec(261, 16, 9) = "1.0.5." XDec(0.359, 16, 9) = "0,5.11.14.7.6.12.8.11.4."

XDec(261.359, 16, 9) = "1.0.5.,5.11.14.7.6.12.8.11.4."

1-1-1- Conversão de uma base qualquer para base 10

Para obter-se o número decimal equivalente a um número escrito em qualquer base é só multiplicar cada dígito por sua potência:

Exemplos:

$(261{,}359)_{10} = (1.0000.0101,\ 0101.1011.1\ ...)_2$

Parte Inteira

$$1\cdot 2^8 + 0\cdot 2^7 + 0\cdot 2^6 + 0\cdot 2^5 + 0\cdot 2^4 + 0\cdot 2^3 + 1\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0 = 261$$

Parte fracionária

$$0\cdot 2^{-1} + 1\cdot 2^{-2} + 0\cdot 2^{-3} + 1\cdot 2^{-4} + 1\cdot 2^{-5} + 0\cdot 2^{-6} + 1\cdot 2^{-7} + 1\cdot 2^{-8} + 1\cdot 2^{-9} = 0.3574$$

$\text{Erro} := |261.359 - 261.3574|$ $\quad\quad$ $\text{Erro} = 0.0016$

$(261{,}359)_{10} = (100.200,\ 100.200.201\ ...)_3$

Parte Inteira

$$1\cdot 3^5 + 0\cdot 3^4 + 0\cdot 3^3 + 2\cdot 3^2 + 0\cdot 3^1 + 0\cdot 3^0 = 261$$

Parte fracionária

$$1\cdot 3^{-1} + 0\cdot 3^{-2} + 0\cdot 3^{-3} + 2\cdot 3^{-4} + 0\cdot 3^{-5} + 0\cdot 3^{-6} + 2\cdot 3^{-7} + 0\cdot 3^{-8} + 1\cdot 3^{-9} = 0.35899$$

$\text{Erro} := |261.359 - 261.35899|$ $\quad\quad$ $\text{Erro} = 0.00001$

$(261{,}359)_{10} = (105{,}5\ \underline{11}\ \underline{14}\ 7\ 6\ \underline{12}\ 8\ \underline{11}\ 4...)_{16}$ ou seja,

$(261{,}359)_{10} = (0{,}5BE.76C.8B4...)_{16}$

Parte Inteira

$$1\cdot 16^2 + 0\cdot 16^1 + 5\cdot 16^0 = 261$$

Parte fracionária

$$\frac{5}{16^1} + \frac{11}{16^2} + \frac{14}{16^3} + \frac{7}{16^4} + \frac{6}{16^5} + \frac{12}{16^6} + \frac{8}{16^7} + \frac{11}{16^8} + \frac{4}{16^9} = 0.35899999999674$$

$\text{Erro} := |261.359 - 261.35899999999674|$ $\quad\quad$ $\text{Erro} = 3.24\times 10^{-12}$

1-2 – Representação do virgula-flutuante

Em geral, um número N ponto-flutuante tem a forma seguinte:

$$N = M \cdot \beta^K$$

onde

M = mantissa, um valor que deve ser entre –1 e +1;
β = base, 2 se o sistemas de numeração for binário, 10 se o sistema de numeração for decimal, etc.;
K = expoente, um inteiro.

Exemplo:

$$N = -19{,}2 \cdot 10^{-8}$$

Reescrevendo o número para a forma $N = -0{,}192 \cdot 10^{-6}$, o número fica na representação da virgula-flutuante, o expoente é igual a –6, a mantissa é igual a –0,192 e a base é 10.

Se além da limitação –1< M < 1, M também satisfaz uma das condições seguintes:

$\frac{1}{\beta} \leq |M| < 1$ significa que devemos ter um digito não nulo após a virgula

ou

M = 0

nós podemos dizer que $N = M \cdot \beta^K$ é um número escrito na notação de virgula-flutuante normalizada.

Exemplo:

-0.27072236 $0{,}27143247 \cdot 10^7$

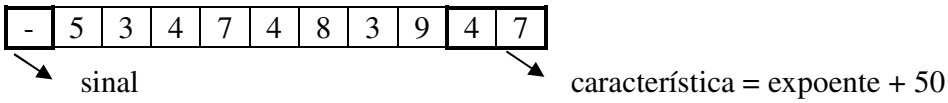
$-0{,}27043247 \cdot 10^7$

$0{,}00071011 \cdot 10^7$

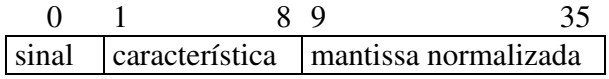
Vemos que a diferença entre estes dois números virgula-flutuante normalizados, resulta num número em virgula-flutuante não normalizado. Podemos, entretanto, normaliza-lo se deslocarmos a virgula três lugares à direita e somar –3 ao expoente, obtendo-se $0{,}71011000 \cdot 10^4$ normalizado.

1-3- Armazenamento na memória

Para começar vamos representar o número 0,00053474839 num computador decimal. A notação virgula-flutuante normalizada dês número é 0,53474839.10-3. Par evitar expoente negativo, nós adicionamos, arbitrariamente, 50 ao expoente e o número agora é $0{,}53474839 \cdot 10^{47}$. O expoente somado a uma constante arbitrária é chamado de característica. O número pode ser representado, unicamente, através da normalização da notação virgula-flutuante, na memória do computado utilizando o esquema seguinte:



Deve ser observado que a característica coloca o expoente limitado a expressão seguinte: $-50 \le k \le 49$. O número tem o máximo de oito dígitos de precisão e a representação falha quando temos números muito grande ou muito pequeno. De modo análogo, um número binário em virgula-flutuante também pode ser armazenado na memória de um computador digital. Uma palavra armazenada tendo um bit de"sinal" e 35 bits regulares pode representar um número binário virgula-flutuante na forma seguinte



Onde

sinal = sinal do número codificado, 0 se positivo e 1 se negativo;
característica = 128 + expoente (resultado escrito em binário);
mantissa = fração binária normalizada.

Exemplo: Represente o número 12,625 numa palavra de 36 bits conforma acima.

Divisão	**Resto**
12:2	0
6:2	0
3:2	1
1:2	1

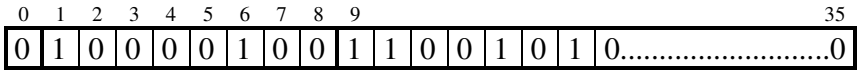
Multiplicação	**Sobra**
0,625x2	1
0,25x2	0
0,5x2	1

Divisão	**Resto**
132:2	0
66:2	0
33:2	1
16:2	0
8:2	0
4:2	0
2:2	0
1:2	1

$12{,}625_{10} = 1100{,}101_2 = 0{,}1100101 \times 2^4$

Ajustando a característica: 4+ 128 = 132

$132_{10} = 10000100_2$



1-4- Armazenamento na memória

Os princípios das operações aritméticas básicas de um computador serão discutidos agora. Para isto iremos considerar que estamos trabalhando num computador decimal com uma palavra de 10 dígitos de comprimento. Princípios semelhantes são utilizados em computadores binários (digitais). Na adição ou subtração de dois números o computador examina a característica ajustada dos dois números. Os seguintes casos são possíveis:

1- Características iguais: Adiciona-se as mantissas e mantém-se a característica.

$$\begin{array}{r} 32109876\ \underline{54} \\ +\ 12340123\ \underline{54} \\ \hline 44449999\ \underline{54} \end{array}$$

2- Quando existe estouro (overflow) na adição das mantissas: O resultado será deslocado uma vez para direita

$$\begin{array}{r} 51319212\ \underline{55} \\ +\ 98756431\ \underline{55} \\ \hline \underline{1}50065643\ \underline{55} \end{array}$$

Resulta em: 15006564 56

característica

estouro

característica

3- Características distintas: Mantém-se a de maior módulo e ajusta-se a de menor valor

$$\begin{array}{r} 31411122\ \underline{55} \\ +\ 12344321\ \underline{53} \\ \hline 44449999\ \underline{54} \end{array} \longrightarrow \begin{array}{r} 31411122\ \underline{55} \\ +\ 00123443\ \underline{55} \\ \hline 31534565\ \underline{55} \end{array}$$

4- Resultado com zero, ou zeros, à esquerda: Normaliza-se o resultado.

$$\begin{array}{r} 34122222\ \underline{73} \\ -\ 34000122\ \underline{73} \\ \hline 00122100\ \underline{73} \end{array}$$ resulta em: 12210000 71

Na multiplicação e divisão as mantissas e características são tratadas separadamente.

Exemplo:

$$\begin{array}{r} 31313142\ \underline{51} \\ \times\ 12315782\ \underline{65} \\ \hline \end{array}$$

mantissa = 0,31313142x0,12315782 = 0,038564583

característica = 51+65-50 = 66, onde –50 é o desconto para compensar o ajuste +50 em cada ajuste do expoente da representação. A resposta é

31313142 51x 12315782 65 = 038564583 66

com a normalização teremos o resultado 38564583 65

1.5 –Erros

Os erros são definidos como absoluto e relativo. Se x é o número exato e x´ uma aproximação, então temos:

Erro absoluto: $\varepsilon = | x - x´ |$, e

Erro relativo: $\left| \frac{\varepsilon}{x} \right| = \left| 1 - \frac{x´}{x} \right|$

Um número decimal é arredondado na posição n desprezando-se todos dígitos à direita desta posição. O dígito na posição n é deixado inalterado ou acrescido de uma unidade se o dígito da posição n + 1 é um número menor que 5 ou maior que 5. Se o número na posição n + 1 for igual a 5, o digito na posição n é acrescido de uma unidade se ele for par e é deixado inalterado se for impar (a regra pode ser o contrário: o digito na posição n é acrescido de uma unidade se ele for impar e é deixado inalterado se for par) . Freqüentemente é feito o truncamento para n decimais onde todos os dígitos além da posição n são simplesmente desprezados.

Exemplo: 3,1415926535

Arredondamento: 3,14 (2d)
3,141 (3d)
3,1416 (4d)
3,1415927 (7d)
onde (nd) = número de casas decimais.

Nós podemos dizer de forma simplória que dígitos significativos são aqueles que têm informação sobre a dimensão do número sem contar com a parte exponencial. Naturalmente um dígito d localizado mais à esquerda tem mais informação do que um mais à direita. Quando um número é escrito com somente seus dígitos significativos estes formam uma cadeia de símbolos que começa com o primeiro dígito diferente de zero. Portanto se a parte fracionária termina com um ou vários zeros, eles são significativos por definição. Se o número é inteiro e termina com zeros, somente com o conhecimento da situação é que podemos decidir se eles são significativos ou não. Por exemplo, 8630574 escrito com 4 dígitos significativos é 8630000.

Em muitos casos nós estimamos o erro de uma função $f(x_1, x_2, \ldots, x_n)$ com erros individuais nas variáveis $(\Delta x_1, \Delta x_2, \ldots, \Delta x_n)$ conhecidos. Nós encontramos diretamente que

$$\Delta f = \frac{\partial f}{\partial x_1}\Delta x_1 + \frac{\partial f}{\partial x_2}\Delta x_2 + \cdots + \frac{\partial f}{\partial x_n}\Delta x_n$$

onde os termos de ordem superior foram desprezados. O erro máximo é dado por

$$|\Delta f| \le \left|\frac{\partial f}{\partial x_1}\right| \cdot |\Delta x_1| + \left|\frac{\partial f}{\partial x_2}\right| \cdot |\Delta x_2| + \cdots + \left|\frac{\partial f}{\partial x_n}\right| \cdot |\Delta x_n|$$

O limite superior do erro é geralmente bastante pessimista, em computações práticas os erros têm uma tendência a cancelar. Por exemplo, se 20.000 números são arredondados com quatro casa decimais e adicionados, o erro máximo é

$$\frac{1}{2} \times 10^{-4} \times 20.000 = 1$$

Do ponto de vista estatístico é esperado que em 99% de todos os casos o erro total não ultrapasse 0,0003 conforme mostra o resultado das simulações abaixo.

```
Soma(x) := | s ← 0
           | S ← 0
           | x ← rnd(1)
           | for i ∈ 1..20000
           |     x ← rnd(x)
           |     s ← s + x
           |     S ← S + round(x, 4)
           | |S − s|
```

```
MC := | for i ∈ 0..999
      |     x_i ← Soma(i / 100000)
      | x
```

```
X := csort(MC, 0)
```

99% de todas as somas são menores que:

$X_{989} = 0.000258319045482$

$X_{989} = 0.000239490743876$

$X_{989} = 0.000239799524392$

$X_{989} = 0.000262389577904$

$X_{989} = 0.000243062908507$

$X_{989} = 0.000281162372689$

Resultados obtido com cerca de 4 min.

$$MC := \left| \begin{array}{l} \text{for } i \in 0..9999 \\ x_i \leftarrow \text{Soma}\left(\frac{i}{100000}\right) \\ x \end{array} \right.$$

$$X := \text{csort}(MC, 0)$$

99% de todas as somas são menores que:

$X_{9899} = 0.00025942089183$

Resultado obtido com cerca de 40 min.

$$MC := \left| \begin{array}{l} \text{for } i \in 0..99999 \\ x_i \leftarrow \text{Soma}\left(\frac{i}{100000}\right) \\ x \end{array} \right.$$

$$X := \text{csort}(MC, 0)$$

99% de todas as somas são menores que:

$X_{98999} = 0.000261343918953$

Normalmente nós classificamos os erros em computações numéricas para estudar suas fontes e seus crescimentos individuais. Enquanto as fontes têm uma natureza que é essencialmente estática, o crescimento é puramente dinâmico. Apesar de erros "grosseiros" terem freqüentemente um papel destacado em cálculo numérico, nós não trataremos dele aqui. Sendo assim, restam essencialmente três fontes de erro:

1. Erros iniciais;
2. Erros de truncamento;
3. Erros de arredondamento.

Erros inicias são erros nos dados iniciais. Os erros de truncamento surgem quando um processo infinito (em algum sentido) é trocado por um finito. Erros de arredondamento surgem do fato que durante uma computação numérica os números devem ser arredondados até um certo número de dígitos. Geralmente uma computação numérica é feita com muitos passos. Cada passo nós temos as aproximações x´ e y´ dos números exatos x e y e nós obtemos a aproximação z´ de Z com o uso de uma das quatro operações aritméticas. Por exemplo, se $x' = x + \delta$ e $y' = y + \gamma$ e se calcula $z = \frac{x}{y}$, calcula-se na realidade

$$z' = \frac{x'}{y'} = \frac{x+\delta}{y+\gamma} + \varepsilon$$

então

$$z' \cong z + \frac{1}{y}\cdot\delta - \frac{x}{y^2}\gamma + \varepsilon$$

Exemplos:

$x := 8 \quad \delta := 0.009 \quad y := 5 \quad \gamma := 0.04$

$X := x + \delta \quad Y := y + \gamma$

$\frac{x}{y} = 1.6$

$\frac{X}{Y} = 1.589 \quad \frac{x}{y} + \frac{1}{y}\cdot\delta - \frac{x}{y^2}\cdot\gamma = 1.589$

$x := 8 \quad \delta := -0.07 \quad y := 5 \quad \gamma := 0.08$

$X := x + \delta \quad Y := y + \gamma$

$\frac{x}{y} = 1.6$

$\frac{X}{Y} = 1.561 \quad \frac{x}{y} + \frac{1}{y}\cdot\delta - \frac{x}{y^2}\cdot\gamma = 1.56$

O erro em z´ é constituído dos erros propagados de x e y e um novo erro de arredondamento.

È interessante ilustrar diferentes tipos de erro mais explicitamente. Suponha que nós queremos computar f(x) para um dado x real. No cálculo prático x é aproximado por x´ pois o computador tem uma palavra finita. A diferença entre |x´- x| é o erro inicial, enquanto $\varepsilon_1 = f(x') - f(x)$ é o erro propagado correspondente. Normalmente f é trocado por uma função mais simples f_1 (freqüentemente uma série de potência truncada). A diferença $\varepsilon_2 = f_1(x') - f(x')$ é o erro de truncamento. Os cálculos são feitos por um computador, portanto não são exatos. Na realidade calculamos $f_2(x')$ no lugar de $f_1(x')$, o qual é um valor calculado errado de uma função errada com argumento errado. A diferença $\varepsilon_3 = f_2(x') - f_1(x')$ é o erro de arredondamento propagado. O erro total é $\varepsilon = f_2(x') - f(x) = \varepsilon_1 + \varepsilon_2 + \varepsilon_3$.

Exemplo: Calcular fazendo todos os cálculos com 4 decimais. (Obs. Os cálculos foram feitas com uma calculadora)

$$\varepsilon_1 = e^{0,3333} - e^{1/3} = -0,000465196$$

$$e^x \cong f_1(x) = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!}$$

para x´ = 0,3333

$$\varepsilon_2 = f_1(x´) - f(x´) = -\left(\frac{0,3333^5}{5!} + \frac{0,3333^6}{6!} + \cdots \right) = -0,0000362750$$

$f_2(x´) = 1 + 0,3333 + 0,0555 + 0,0062 + 0,0005 = 1,3955$
$f_1(x´) = 1,3955296304$ obtidos com 10 decimais

$$\varepsilon_3 = -0,0000296304$$

$$\varepsilon = 1.3955 - e^{1/3} = \varepsilon_1 + \varepsilon_2 + \varepsilon_2 = -0,0001124250$$

1-5– Cancelamento numérico

Devido ao comprimento limitado das palavras em computadores, e em conseqüência do uso da aritmética da vírgula flutuante normalizada, as leis da aritmética elementar não são satisfeitas. Os efeitos do uso da aritmética da vírgula flutuante serão vistos em alguns exemplos que seguem.

Os exemplos a seguir violam a lei associativa da adição.

Exemplo 1: (quando usamos uma máquina com quatro dígitos decimais na representação)

x = 9,909 y = 1,000 z = -0,990

(x + y) + z = 10,90 + (-0,990) = 9,910

x + (y + z) = 9,909 + (0,010) = 9,919

Exemplo 2: (quando usamos uma máquina com quatro dígitos decimais na representação)

x = 4561 y = 0,3472
(y + x) - x = (-0,3472 + 4561) - 4561

= 4561 - 4561 = 0,0000

y + (x - x) = 0,3472 + (4561 – 4561)

$= 0{,}3472 + 0{,}0000 = 0{,}3472$

Vejamos agora um exemplo (quando usamos uma máquina com quatro dígitos decimais na representação) que viola a lei distributiva

$x = 9909 \qquad y = -1{,}000 \qquad z = 0{,}999$

$$(x \times y) + (x \times z) = -9909 + 9899 = -10{,}00$$

$$x \times (y + z) = 9909 + (-0{,}001) = -9{,}909$$

A equação do segundo grau $x^2 - bx + \varepsilon = 0$ tem duas soluções:

$$x_1 = \frac{b + \sqrt{b^2 - 4\varepsilon}}{2} \qquad \text{e} \qquad x_2 = \frac{b - \sqrt{b^2 - 4\varepsilon}}{2}$$

Se $b < 0$ e $\varepsilon << b$, x2 é expresso como a diferença de dois números praticamente iguais e poderá perder muitos dígitos significativos. Se nós reescrevemos

$$x_2 = \frac{\varepsilon}{x_1} = \frac{2\varepsilon}{b + \sqrt{b^2 - 4\varepsilon}}$$

a raiz é aproximadamente $\frac{\varepsilon}{b}$ sem perda de dígitos significativos.

Exemplo: (quando usamos uma máquina com quatro dígitos decimais na representação)

$b = 300{,}0 \qquad \text{e} \qquad \varepsilon = 1{,}000$

$$\sqrt{90000 - 4{,}000} = 300{,}0$$

$$x_1 = \frac{600{,}0}{2{,}000} = 300{,}0$$

$$x_2 = \frac{300{,}0 - 300{,}0}{2{,}000} = \frac{0{,}000}{2{,}000} = 0{,}000$$

usando a relação $x_2 = \frac{\varepsilon}{x_1} = \frac{1{,}000}{300{,}0} = 0{,}003$ é o resultado mais preciso.

Sabe-se que para x grane senh(x) ≅ cosh(x) ≅ $\frac{e^{-x}}{2}$. Se quisermos calcular e^{-x} podemos dizer que e^{-x} = cosh(x) – senh(x), o que conduz a um cancelamento perigoso. Por outro lado $e^{-x} = \frac{1}{\cosh(x) + \text{senh}(x)}$ fornece resultados mais precisos.

1-6– Exercícios

1) Converta os seguintes números decimais para sua forma binária:

a) 35 b) 2345 c) 0.1218 d) 67,67 e) 95 f) 2500

g) 2000 h) 655 i) 722 j) $3,6 \times 10^{21}$ l) 231 m) $2,5 \times 10^{-18}$

2) Converta os números binários para suas formas octal, hexadecimal e decimal:

a) 101101_2 b) -110101011_2 c) -0.1101_2 d) 0.111111101_2 e) 0.0000101_2

f) 101101_2 g) -110101011_2 h) -0.1101_2 i) 0.111111101_2 j) 0.0000101_2

3) Reescreva os números seguintes na forma geral da vírgula flutuante, tal que a mantissa fique ~~entre 1 e –1:~~ normalizada.

a) 27,534 b) –89,901 c) 18×10^{21} d) $1,3756 \times 10^{-7}$

e) $11,0111_2$ f) $-111,0101_2$ g) $0,00101_2$ h) 111010101_2

~~4) Qual o valor de cada expoente se os números da questão 3 forem colocados na representação vírgula-flutuante normalizada? Qual é o valor do expoente ajustado em cada caso se nós adicionamos arbitrariamente 25 ao expoente original?~~

5) Seja o número seguinte em vírgula-flutuante num computador de 32 bits:

0010.0101.0000.0001.0001.1001.1100.1110

Se o primeiro bit é o sinal do número, os oito seguintes a característica obtida com adição de 128 ao expoente do número vírgula flutuante, e os 23 restantes são a mantissa, responda às questões seguintes:

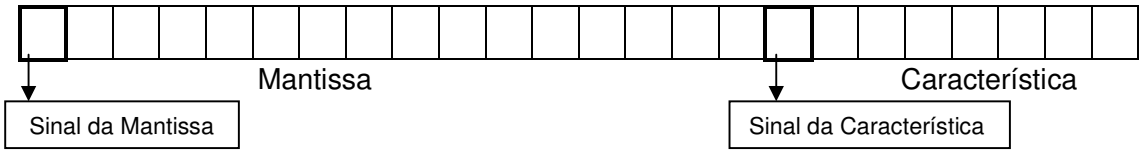
a) O número está normalizado? Se não o normalize.
b) Qual o sinal do número?
c) O valor absoluto do número é menos que 1.

6) Repita a questão 5 com o número

1000.0000.0110.1101.1010.1101.1011.0110

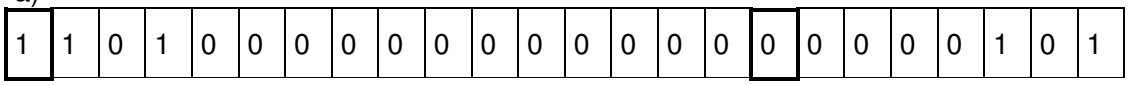
7) Para a representação da questão 5, quais são aproximadamente o maior e o menos número, o menor número positivo e o próximo menor número positivo.

8) Represente os números binários da questão 2) na maquina binária que utiliza o seguinte esquema de representação de ponto flutuante:

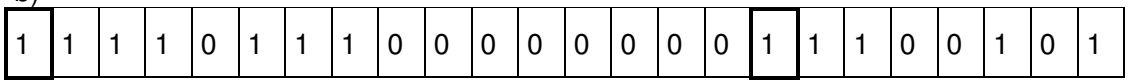


9) Converter para base 10 os valores representados na máquina binária:

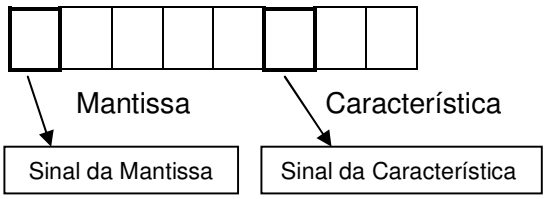
a)



b)



9) Seja um sistema de aritmética de ponto flutuante na base decimal com quatro dígitos na mantissa e dois na característica, 1 digito de sinal da mantissa e 1 digito sinal da característica.



Dados os números:

$x = 0.77237$ $y = 0.2145 \times 10^{-3}$ $z = 0.2585 \times 10^{1}$

Efetue as seguintes operações:

a) x+y+z b) x-y-z c) x/y d) (xy)/z e) x(y/z)

10) Use a aritmética da vírgula-flutuante par somar e subtrair os seguintes pares de números:

11) Use a aritmética da vírgula-flutuante para realizar as operações aritméticas seguintes:

12) Calcular as cotas dos erros absolutos e relativos que se comete ao se tomar como valores de π:

a) 22/7 b) 333/116 c) 355/113 d) $\sqrt{3}+\sqrt{2}$

13) Use a aritmética da vírgula-flutuante par somar e subtrair os seguintes pares de números?

14) Use a aritmética da vírgula-flutuante par somar e subtrair os seguintes pares de números?

Bibliografia