

Sistemas de Ponto Flutuante

CCT - UDESC

[Fernando Deeke Sasse](mailto:fsasse@alumni.uwaterloo.ca)

fsasse@alumni.uwaterloo.ca

Departamento de Matemática, UDESC - Joinville

Sistemas de ponto flutuante em geral

Na aritmética usual temos a liberdade de utilizar números com um número infinito de dígitos não-periódicos. Por exemplo, definimos $\sqrt{3}$ como sendo o único número positivo que multiplicado por ele mesmo resulta o inteiro 3. Na aritmética computacional, entretanto, cada número é representável somente através de um número fixo, finito de dígitos. Como $\sqrt{3}$ não tem uma representação de dígitos finita, uma representação aproximada é escolhida pela máquina, cujo quadrado não é exatamente 3, embora uma boa aproximação em geral. O objetivo desta seção é examinar alguns casos onde a representação finita de dígitos pode trazer problemas não triviais. Erros de arredondamento ocorrem quando uma calculadora ou computador é utilizado para realizar cálculos envolvendo números reais. Em um computador típico, somente um subconjunto relativamente pequeno dos números reais é utilizado para a representação de todos os números reais. Este subconjunto contém somente os números racionais, positivos e negativos. Um fato importante, que veremos logo a seguir, é que certos erros aparentemente inexplicáveis devem-se ao fato de que diferentes sistemas computacionais utilizam diferentes sistemas de aritmética de ponto flutuante, via hardware ou software. Maiores informações podem ser obtidas nas referências [1,2,3,4,5].

Normalmente os dados de entrada são enviados ao computador pelo usuário no sistema decimal. O computador os converte para binário, onde são efetuadas as computações. Os resultados finais são convertidos para o sistema decimal e transmitidos ao usuário. Estes processos de conversão são uma fonte de erro dos cálculos.

A representação de um número no formato de ponto flutuante pode ser dada na forma

$$x = \pm .(d_1 d_2 \dots d_p) B^e$$

sendo d_i dígitos cujos valores podem variar de 0 a $B - 1$, sendo B a base

(normalmente 2, 10 ou 16), p é o número de dígitos (precisão) e e é o expoente inteiro que está dentro de um certo intervalo $e_1 < e < e_2$. O termo $d_1 d_2 \dots d_p$ é denominado mantissa. Outra forma para esta representação é:

$$x = \pm \left(\frac{d_1}{B} + \frac{d_2}{B} + \frac{d_3}{B} + \dots + \frac{d_p}{B^p} \right) B^e.$$

Esta representação é denominada **normalizada** quando o bit de mais alta ordem é 1. No presente caso, $d_1 = 1$. Com estas convenções acima, vamos denotar sistemas de ponto flutuante particulares na forma $F(B, p, e_1, e_2)$. Neste sistema o menor número não nulo, possível de ser representado, é

$$0.1 B^{e_1}$$

e o maior número é (p fatores $(B - 1)$):

$$0.(B-1)(B-1)\dots(B-1)B^{e_2}.$$

Consideremos alguns exemplos. Quando um número é muito pequeno para ser representado pelo sistema da máquina, ele é considerado zero, e temos nesse caso um underflow. Quando ele é grande demais para ser representável pela máquina, temos overflow e a computação pára.

Exemplo 1

Consideremos o sistema F(10, 3, -5, 5). Os números são representados na seguinte forma:

$$0.d_1 d_2 d_3 10^e, \quad 0 \leq d_j \leq 9, \quad -5 \leq e \leq 5.$$

O número de menor magnitude normalizada nesta máquina é dado por

$$n_1 = 0.100 \cdot 10^{-5},$$

e o de maior magnitude é

$$n_2 = 0.999 \cdot 10^5.$$

Consideremos as limitações da máquina com as entradas dos seguintes números:

(a) $x = 235.89 = 0.23589 \cdot 10^5$

Como a máquina só pode representar três dígitos na mantissa, os dígitos 8 e 9 são eliminados, de modo que o número representado é $0.235 \cdot 10^5$, se for utilizado o truncamento, e $0.236 \cdot 10^5$ se for utilizado o arredondamento.

(b) $x = 0.345 \cdot 10^{-7}$. Este número não pode ser representado nesta máquina porque o expoente -7 é menor que o menor admitido -5. Este é um caso de *underflow*, e tais números são considerados como **zero** pela máquina.

(c) $x = 0.562 \cdot 10^8$. Este número não pode ser representado pela máquina pois o expoente +8 é maior do que o máximo +5. Neste caso temos *overflow*, e a computação pára.

Exemplo 2

Consideremos o sistema normalizado definido por F(2, 3, -1, 2). As possíveis mantissas são: 0.100, 0.101, 0.110, 0.111 e os possíveis expoentes são 2.

O menor número nesta representação (binária) é

$$n_1 = 0.100 \cdot 2^{-1},$$

que na representação decimal equivale a

$$1 \cdot 2^{-1} \cdot 2^{-1} = \frac{1}{4}.$$

O maior número nesta representação é

$$n_2 = 0.111 \cdot 2^2,$$

que na representação decimal equivale a

$$(1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3}) \cdot 2^2 = \frac{7}{2}.$$

Portanto, a região entre $-1/4$ e $1/4$ é chamada *região de underflow*, enquanto que a região com números maiores que $7/2$ e menores que $-7/2$ é chamada de *região de overflow*.

Determinemos todos os números de possível ocorrência nesta máquina. Mostraremos que sua distribuição não é linear:

```
> restart;
> with(combinat);
[Chi, bell, binomial, cartprod, character, choose, composition, conjpart, decodepart,
  encodepart, eulerian1, eulerian2, fibonacci, firstpart, graycode, inttovec, lastpart,
  multinomial, nextpart, numbbcomb, numbbcomp, numbbpart, numbbperm, partition,
  permute, powerset, prevpart, randcomb, randpart, randperm, setpartition, stirling1,
  stirling2, subsets, vectoint]
```

(1.1)

```
> L := [op(permute([1, 0])), [0, 0], [1, 1]];
      L := [[1, 0], [0, 1], [0, 0], [1, 1]]
```

(1.2)

```
> LL := [seq([1, op(L[i])], i = 1 .. 4)];
      LL := [[1, 1, 0], [1, 0, 1], [1, 0, 0], [1, 1, 1]]
```

(1.3)

```
> nm:=nops(LL):v:=0:lnum:={}:p:=3:
> e2:=-1:e1:=2:
> m[1]:=[1,0,0]:m[2]:=[1,0,1]:m[3]:=[1,1,0]:m[4]:=[1,1,1]:
>   for j from e2 to e1 do
>     for i to nm do
>       x[i,j]:=sum(op(k,LL[i])*2^(-k),k=1..p)*2^(j);
>       v:=v+1:
>
>     n[v]:=%%;
>     lnum:=lnum union {n[v]} ;
>   od;
> od;
> print('numeros'=lnum);
      numeros = {1, 2, 3, 1/2, 1/4, 3/2, 3/4, 3/8, 5/2, 5/4, 5/8, 5/16, 7/2, 7/4, 7/8, 7/16}
```

(1.4)

Vejamos como estes números estão distribuídos.

```
> with(plots);
[animate, animate3d, animatecurve, arrow, changecoords, complexplot, complexplot3d,
  conformal, conformal3d, contourplot, contourplot3d, coordplot, coordplot3d,
  densityplot, display, dualaxisplot, fieldplot, fieldplot3d, gradplot, gradplot3d, implicitplot,
  implicitplot3d, inequal, interactive, interactiveparams, intersectplot, listcontplot,
  listcontplot3d, listdensityplot, listplot, listplot3d, loglogplot, logplot, matrixplot, multiple,
  odeplot, pareto, plotcompare, pointplot, pointplot3d, polarplot, polygonplot,
  polygonplot3d, polyhedra_supported, polyhedraplot, rootlocus, semilogplot, setcolors,
  setoptions, setoptions3d, spacecurve, sparsematrixplot, surfdata, textplot, textplot3d,
  tubeplot]
```

```
> lnum;
```

(1.5)

$$\left\{1, 2, 3, \frac{1}{2}, \frac{1}{4}, \frac{3}{2}, \frac{3}{4}, \frac{3}{8}, \frac{5}{2}, \frac{5}{4}, \frac{5}{8}, \frac{5}{16}, \frac{7}{2}, \frac{7}{4}, \frac{7}{8}, \frac{7}{16}\right\} \quad (1.6)$$

```
> Lnum := convert(Lnum, list);
```

$$Lnum := \left[1, 2, 3, \frac{1}{2}, \frac{1}{4}, \frac{3}{2}, \frac{3}{4}, \frac{3}{8}, \frac{5}{2}, \frac{5}{4}, \frac{5}{8}, \frac{5}{16}, \frac{7}{2}, \frac{7}{4}, \frac{7}{8}, \frac{7}{16}\right] \quad (1.7)$$

```
> LnumS := sort(Lnum, '<');
```

$$LnumS := \left[\frac{1}{4}, \frac{5}{16}, \frac{3}{8}, \frac{7}{16}, \frac{1}{2}, \frac{5}{8}, \frac{3}{4}, \frac{7}{8}, 1, \frac{5}{4}, \frac{3}{2}, \frac{7}{4}, 2, \frac{5}{2}, 3, \frac{7}{2}\right] \quad (1.8)$$

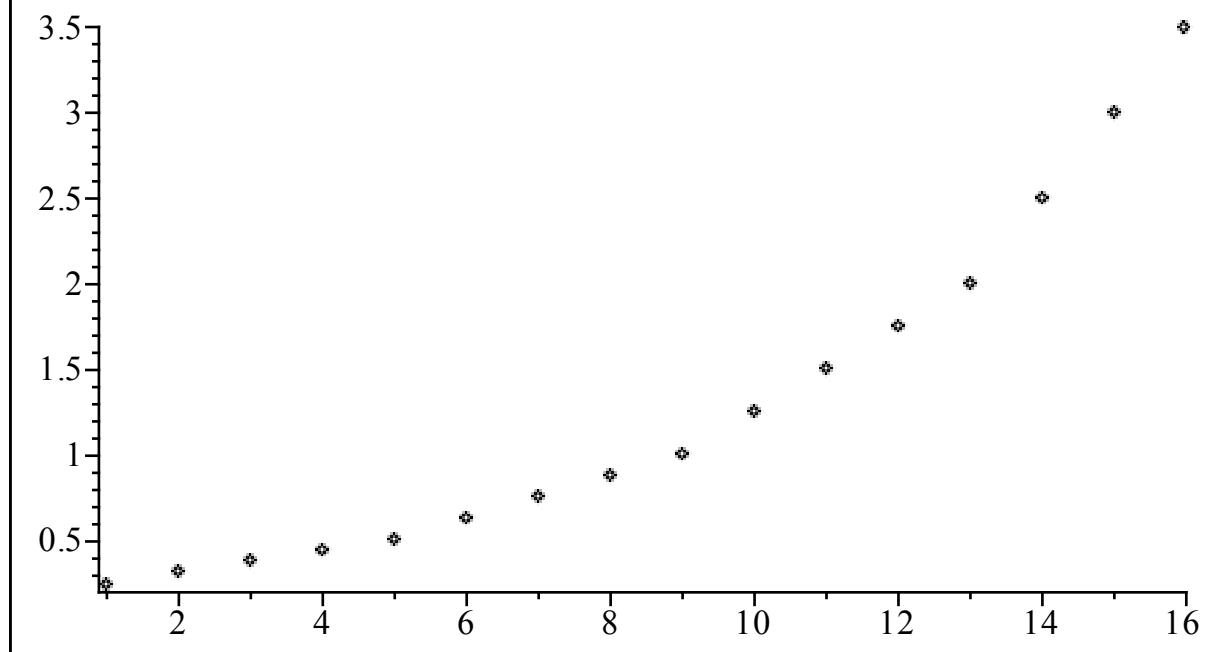
```
> nn := nops(LnumS);
```

$$nn := 16 \quad (1.9)$$

```
> P := [seq([i, LnumS[i]], i = 1 .. nn)];
```

$$P := \left[\left[1, \frac{1}{4}\right], \left[2, \frac{5}{16}\right], \left[3, \frac{3}{8}\right], \left[4, \frac{7}{16}\right], \left[5, \frac{1}{2}\right], \left[6, \frac{5}{8}\right], \left[7, \frac{3}{4}\right], \left[8, \frac{7}{8}\right], [9, 1], \left[10, \frac{5}{4}\right], \left[11, \frac{3}{2}\right], \left[12, \frac{7}{4}\right], [13, 2], \left[14, \frac{5}{2}\right], [15, 3], \left[16, \frac{7}{2}\right]\right] \quad (1.10)$$

```
> pointplot(P);
```



Notemos que o zero não está definido no conjunto acima. Ele é especialmente definido na máquina e não é normalizado. Façamos alguns cálculos envolvendo números representáveis neste sistema. Por exemplo, $1/2 + 7/4 = 9/4$. Este número não pertence ao conjunto dos números decimais representáveis pelo sistema, que determinamos acima. De fato, $9/4$ em binário é dado por

```
> convert(evalf(9/4), binary);
```

$$10.01000000 \quad (1.11)$$

ou seja, $0.1001 \cdot 2^2$, que possui uma mantissa maior que a admitida pelo sistema. Se houver truncamento, este número será aproximado para $0.100 \cdot 2^2$, que representa 2. Se esta soma for feita utilizando-se a representação binária, obteremos

$$0.100 \cdot 2^0 + 0.111 \cdot 2^1 = 0.100 \cdot 2^2 = 2$$

Exemplo 3

Consideremos o sistema normalizado definido por $F(2, 4, -3, 3)$. Determinemos todos os números possíveis deste sistema e a sua distribuição na forma decimal.

```
> restart;  
> with(combinat);  
[Chi, bell, binomial, cartprod, character, choose, composition, conjpart, decodepart, encodepart, eulerian1, eulerian2, fibonacci, firstpart, graycode, inttovec, lastpart, multinomial, nextpart, numbbcomb, numbbcomp, numbbpart, numbbperm, partition, permute, powerset, prevpart, randcomb, randpart, randperm, setpartition, stirling1, stirling2, subsets, vectoint]
```

```
> L := [op(permute([1, 0, 0])), op(permute([1, 0, 1])), op(permute([1, 1, 0])), [0, 0, 0], [1, 1, 1]];  
L := [[1, 0, 0], [0, 1, 0], [0, 0, 1], [1, 0, 1], [1, 1, 0], [0, 1, 1], [1, 1, 0], [1, 0, 1], [0, 1, 1], [0, 0, 0], [1, 1, 1]]
```

```
> nm := nops(L);  
nm := 11
```

```
> LL := [seq([1, op(L[i])], i = 1 .. nm)];  
LL := [[1, 1, 0, 0], [1, 0, 1, 0], [1, 0, 0, 1], [1, 1, 0, 1], [1, 1, 1, 0], [1, 0, 1, 1], [1, 1, 1, 0], [1, 1, 0, 1], [1, 0, 1, 1], [1, 0, 0, 0], [1, 1, 1, 1]]
```

```
> v:=0:lnum:={}:e1:=-3:e2:=3:  
> e2:=-1:e1:=2:p:=4:  
> for j from e2 to e1 do  
> for i to nm do  
> x[i, j]:=sum(op(k, LL[i])*2^(-k), k=1..p)*2^(j);  
> v:=v+1:  
> n[v]:=%%;  
> lnum:=lnum union {n[v]} ;  
> od;  
> od;  
> print('numeros'=lnum);
```

```
numeros = {1, 2, 3,  $\frac{1}{2}$ ,  $\frac{1}{4}$ ,  $\frac{3}{2}$ ,  $\frac{3}{4}$ ,  $\frac{3}{8}$ ,  $\frac{5}{2}$ ,  $\frac{5}{4}$ ,  $\frac{5}{8}$ ,  $\frac{5}{16}$ ,  $\frac{7}{2}$ ,  $\frac{7}{4}$ ,  $\frac{7}{8}$ ,  $\frac{7}{16}$ ,  $\frac{9}{4}$ ,  $\frac{9}{8}$ ,  $\frac{9}{16}$ ,  
 $\frac{9}{32}$ ,  $\frac{11}{4}$ ,  $\frac{11}{8}$ ,  $\frac{11}{16}$ ,  $\frac{11}{32}$ ,  $\frac{13}{4}$ ,  $\frac{13}{8}$ ,  $\frac{13}{16}$ ,  $\frac{13}{32}$ ,  $\frac{15}{4}$ ,  $\frac{15}{8}$ ,  $\frac{15}{16}$ ,  $\frac{15}{32}$ }
```

```
> with(plots);  
[animate, animate3d, animatecurve, arrow, changecoords, complexplot, complexplot3d, conformal, conformal3d, contourplot, contourplot3d, coordplot, coordplot3d, densityplot, display, dualaxisplot, fieldplot, fieldplot3d, gradplot, gradplot3d, implicitplot, implicitplot3d, inequal, interactive, interactiveparams, intersectplot, listcontplot, listcontplot3d, listdensityplot, listplot, listplot3d, loglogplot, logplot, matrixplot, multiple, odeplot, pareto, plotcompare, pointplot, pointplot3d, polarplot, polygonplot, polygonplot3d, polyhedra_supported, polyhedraplot, rootlocus, semilogplot, setcolors, setoptions, setoptions3d, spacecurve, sparsematrixplot, surfdata, textplot, textplot3d, tubeplot]
```

$$\begin{aligned}
 &> \text{lnum}; \\
 &\left\{ 1, 2, 3, \frac{1}{2}, \frac{1}{4}, \frac{3}{2}, \frac{3}{4}, \frac{3}{8}, \frac{5}{2}, \frac{5}{4}, \frac{5}{8}, \frac{5}{16}, \frac{7}{2}, \frac{7}{4}, \frac{7}{8}, \frac{7}{16}, \frac{9}{4}, \frac{9}{8}, \frac{9}{16}, \frac{9}{32}, \frac{11}{4}, \right. \\
 &\quad \left. \frac{11}{8}, \frac{11}{16}, \frac{11}{32}, \frac{13}{4}, \frac{13}{8}, \frac{13}{16}, \frac{13}{32}, \frac{15}{4}, \frac{15}{8}, \frac{15}{16}, \frac{15}{32} \right\} \quad (1.18)
 \end{aligned}$$

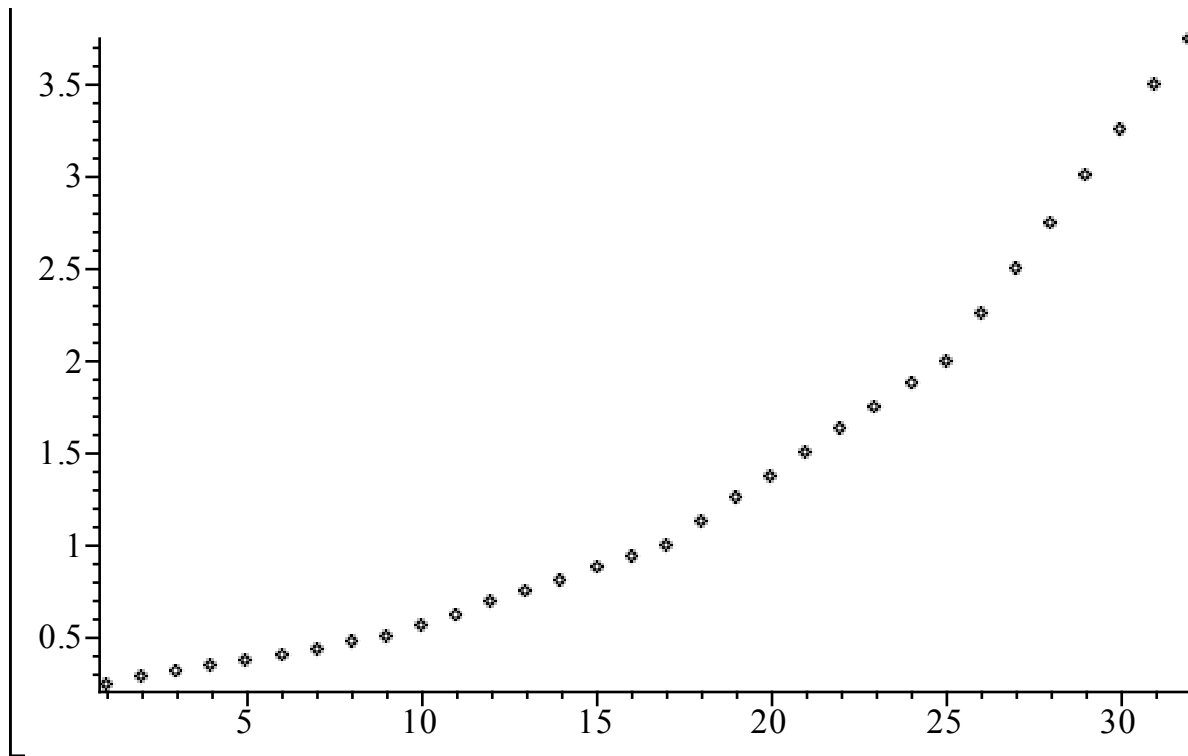
$$\begin{aligned}
 &> \text{Lnum} := \text{convert}(\text{lnum}, \text{list}); \\
 \text{Lnum} &:= \left[1, 2, 3, \frac{1}{2}, \frac{1}{4}, \frac{3}{2}, \frac{3}{4}, \frac{3}{8}, \frac{5}{2}, \frac{5}{4}, \frac{5}{8}, \frac{5}{16}, \frac{7}{2}, \frac{7}{4}, \frac{7}{8}, \frac{7}{16}, \frac{9}{4}, \frac{9}{8}, \frac{9}{16}, \right. \\
 &\quad \left. \frac{9}{32}, \frac{11}{4}, \frac{11}{8}, \frac{11}{16}, \frac{11}{32}, \frac{13}{4}, \frac{13}{8}, \frac{13}{16}, \frac{13}{32}, \frac{15}{4}, \frac{15}{8}, \frac{15}{16}, \frac{15}{32} \right] \quad (1.19)
 \end{aligned}$$

$$\begin{aligned}
 &> \text{LnumS} := \text{sort}(\text{Lnum}, \text{'<'}); \\
 \text{LnumS} &:= \left[\frac{1}{4}, \frac{9}{32}, \frac{5}{16}, \frac{11}{32}, \frac{3}{8}, \frac{13}{32}, \frac{7}{16}, \frac{15}{32}, \frac{1}{2}, \frac{9}{16}, \frac{5}{8}, \frac{11}{16}, \frac{3}{4}, \frac{13}{16}, \frac{7}{8}, \frac{15}{16}, 1, \right. \\
 &\quad \left. \frac{9}{8}, \frac{5}{4}, \frac{11}{8}, \frac{3}{2}, \frac{13}{8}, \frac{7}{4}, \frac{15}{8}, 2, \frac{9}{4}, \frac{5}{2}, \frac{11}{4}, 3, \frac{13}{4}, \frac{7}{2}, \frac{15}{4} \right] \quad (1.20)
 \end{aligned}$$

$$\begin{aligned}
 &> \text{nn} := \text{nops}(\text{LnumS}); \\
 &\qquad \qquad \qquad \text{nn} := 32 \quad (1.21)
 \end{aligned}$$

$$\begin{aligned}
 &> \text{P} := [\text{seq}([i, \text{LnumS}[i]], i = 1 .. \text{nn})]; \\
 \text{P} &:= \left[\left[1, \frac{1}{4} \right], \left[2, \frac{9}{32} \right], \left[3, \frac{5}{16} \right], \left[4, \frac{11}{32} \right], \left[5, \frac{3}{8} \right], \left[6, \frac{13}{32} \right], \left[7, \frac{7}{16} \right], \left[8, \frac{15}{32} \right], \left[9, \right. \right. \\
 &\quad \left. \frac{1}{2} \right], \left[10, \frac{9}{16} \right], \left[11, \frac{5}{8} \right], \left[12, \frac{11}{16} \right], \left[13, \frac{3}{4} \right], \left[14, \frac{13}{16} \right], \left[15, \frac{7}{8} \right], \left[16, \frac{15}{16} \right], \left[17, \right. \\
 &\quad \left. 1 \right], \left[18, \frac{9}{8} \right], \left[19, \frac{5}{4} \right], \left[20, \frac{11}{8} \right], \left[21, \frac{3}{2} \right], \left[22, \frac{13}{8} \right], \left[23, \frac{7}{4} \right], \left[24, \frac{15}{8} \right], \left[25, 2 \right], \\
 &\quad \left[26, \frac{9}{4} \right], \left[27, \frac{5}{2} \right], \left[28, \frac{11}{4} \right], \left[29, 3 \right], \left[30, \frac{13}{4} \right], \left[31, \frac{7}{2} \right], \left[32, \frac{15}{4} \right] \right] \quad (1.22)
 \end{aligned}$$

$$> \text{pointplot}(\text{P});$$



O menor número (em magnitude) de pontos flutuante que, quando adicionado ao número 1.0 produz um número de ponto flutuante diferente de 1.0 é denominado **acurácia (ou exatidão) de máquina** ϵ_m . Um computador típico com $B = 2$ e comprimento de palavra de 32 bits tem ϵ_m em torno de 10^{-8} . Em outras palavras, a acuracidade de máquina ϵ_m é a acuracidade fracional na qual números de pontos flutuantes são representados, correspondendo a uma mudança de 1 no bit menos significativo da mantissa. Em geral qualquer operação aritmética entre números de pontos flutuantes pode ser pensada como sendo a introdução de um erro fracional de ao menos ϵ_m (erro de arredondamento).

No exemplo anterior, somando $1 + 1/4$, obtemos $5/4$, de modo que a precisão de máquina é $1/4$ (coincidindo com o menor número representável no sistema.)

É importante notar que ϵ_m não é o menor número de ponto flutuante que pode ser representado em uma dada máquina. Este último depende de quantos bits podem ser armazenados no expoente, enquanto que ϵ_m depende de quantos bits há na mantissa. A acuracidade (acurácia, exatidão) de máquina de um sistema de ponto flutuante pode ser determinada por um simples algoritmo. Em Maple podemos definir o procedimento que usa o sistema de ponto flutuante de hardware (IEEE-754)

```
> Digits:=20:
  epsilon:=1:
  > while evalhf(epsilon+1)>1 do
  >   epsilon:=evalhf(epsilon/2);
  > od:
  > evalhf(epsilon);
```

$1.11022302462515654 \cdot 10^{-16}$

(1.23)

Notemos que neste caso o sistema de ponto de flutuante de hardware é do tipo estendido (80bits). O comando Digits é mais forte que evalhf, de modo que devemos estabelecer a precisão como sendo a

maior possível. Não faria diferença se usássemos `Digits:=25`, por exemplo. No sistema que usa sistema de ponto flutuante de software (decimal) temos:

```
> Digits := 20 :
> epsilon:=1:
> while evalf(epsilon+1)>1 do
>   epsilon:=evalf(epsilon/2) ;
> od:
> epsilon;
```

$2.7105054312137610850 \cdot 10^{-20}$ (1.24)

As diferenças nestes valores se devem ao fato de que representações em diferentes bases são usadas, sendo que no sistema de base decimal do Maple não há limites baixos para a precisão.

Exercícios

🔍: Resolva manualmente (no máximo com calculadora)

👑: Resolva no computador

🔍 1. Seja o sistema de ponto flutuante $F(10,4, -14,15)$. Determine o maior e o menor número do sistema (normalizados).

🔍 2. Seja o sistema de ponto flutuante $F(2,6, -12,13)$. Determine o maior e o menor número (normalizados). Quantos dígitos decimais de precisão este sistema oferece?

👑 2. Seja o sistema de ponto flutuante $F(2, 26, -14,15)$. Determine o maior e o menor número (normalizados). Quantos dígitos decimais de precisão este sistema oferece?

👑 3. Considere o sistema normalizado definido por (i) $F(2, 4, -6, 7)$, (iii) $F(10, 3, 2, -3, 3)$. Determine, em cada caso, todos os números possíveis deste sistema e a sua distribuição na forma decimal em forma gráfica.

Representação de ponto flutuante binária IEEE 754

Quando em 1985 a Intel decidiu introduzir um coprocessador de ponto flutuante para seu novo microprocessador 8086, denominado 8087 FPU, um novo sistema de ponto flutuante binário foi introduzido (Kahn, Coonan, Stone), sendo logo adotado como padrão pelo IEEE (Institute for Electrical and Electronic Engineers). Foram definidos os formatos denominados simples, duplo e estendido.

Formato simples

O formato simples IEEE, ou precisão simples, denominado "float" em C, é definido de tal modo que cada número é armazenado em uma palavra de 32 bit (4 bytes) na seguinte forma:

$$(-1)^s 2^{E-127} \cdot \left(1 + \sum_{i=1}^{23} 2^{-i} \right),$$

sendo a distribuição de bits dada por

sinal	expoente	bit implícito	mantissa
[s]	[eeee eeee]	[1]	[mmm···

			$\cdot mmm]$
1 bit	8 bits		23 bits

Notemos que a mantissa está entre 1 e 2. O bit 1 mais à esquerda é já implícito, e portanto não necessita ser armazenado.

Valores Normalizados

Vejamos o expoente. Embora um número binário de 8 bits possa ter valores decimais entre $(0000\ 0000)_2$ e $(1111\ 1111)_2 = 255_{10}$, estes valores extremos são reservados para números especiais, como veremos logo adiante. Para números **normais** os bits do expoente podem estar entre $(0000\ 0001)_2$ e $(1111\ 1110)_2$. Ou seja, $1 < E < 254$, pois

```
[ > restart
  > E := sum(2^k, k = 1 .. 7);
                                     E := 254 (2.1.1)
```

```
[ > 2^7;
                                     128 (2.1.2)
```

```
[ > E-127;
                                     127 (2.1.3)
```

O número 127 é denominado *bias* ou **polarização** do expoente e foi introduzido para permitir expoentes negativos. Ou seja, o expoente $E_B = E - 127$ pode estar no intervalo $-126 < E_B < 127$.

O número de dígitos significativos da parte fracionária da mantissa, representáveis na base decimal é

```
[ > fsolve(2^23 = 10^x, x);
                                     6.923689900 (2.1.4)
```

ou seja, aproximadamente 7 dígitos.

O maior número (positivo) **normalizado** desta representação é dado por $[0] [1111\ 1110] [1]$. $[1111\ 1111\ 1111\ 1111\ 1111\ 1111]$

A mantissa é dada por

```
[ > Digits := 8;
                                     Digits := 8 (2.1.5)
```

```
[ > m[max] := evalf(sum(2^(-k), k = 1 .. 23));
                                     m_max := 0.99999988 (2.1.6)
```

Portanto, o maior número decimal que pode ser representado com precisão simples é :

```
[ > x[max] := (1+m[max])*2^(E-127);
                                     x_max := 3.4028234 10^38 (2.1.7)
```

O menor número (negativo) normal ou **normalizado** desta representação é dado por $[1] [0000\ 0001] [1]$. $[0000\ 0000\ 0000\ 0000\ 0000\ 0000]$

$$\begin{aligned} &> \mathbf{x[\min] := 1.*2^{(1-127)}; } \\ & \qquad \qquad \qquad x_{\min} := 1.1754943 \cdot 10^{-38} \end{aligned} \quad (2.1.8)$$

Notemos que embora x_{\max} e x_{\min} sejam representados por 8 dígitos, a aritmética é realizada a 7 dígitos. A acurácia de máquina deve ser da ordem de 2^{-23} , pois o número 1 é representado por $1.0000\ 0000\ 0000\ 0000\ 000 \cdot 2^0$. O próximo número que este sistema consegue distinguir de 1 é $1.0000\ 0000\ 0000\ 0000\ 001 \cdot 2^0$, que corresponde a 2^{-23} ou

$$\begin{aligned} &> 2^{(-23)} \\ & \qquad \qquad \qquad 1.1920929 \cdot 10^{-7} \end{aligned} \quad (2.1.9)$$

Valores especiais

Zero não é representado diretamente, devido ao bit embutido que sempre tem valor 1. O zero é um valor especial com um campo de expoentes zero (E) e um campo de mantissa zero. Os valores -0 e $+0$ são distintos. Valores em ponto flutuante especiais admitidos pelo padrão são resumidos a seguir:

$+ \textit{Infinity}$	[0] [1111 1111] [1] . [0000 0000 0000 0000 0000 000]
$- \textit{Infinity}$	[1] [1111 1111] [1] . [0000 0000 0000 0000 0000 000]
\textit{NaN} (Not a Number)	[1] [1111 1111] [1] . [1000 0000 0000 0000 0000 000]
$\textit{Positive zero}$	[0] [0000 0000] [1] . [0000 0000 0000 0000 0000 000]
$\textit{Negative zero}$	[1] [0000 0000] [1] . [0000 0000 0000 0000 0000 000]

Valores denormalizados ou subnormais

Números normalizados neste padrão podem ser escritos na forma $2^{k-23} n = \pm 2^k (1 + f)$ com $-2^{24} \leq n \leq 2^{24}$, $1 - 2^7 < k < 2^7$ e fração não-negativa $f < 1$. O fator n é chamado *significando*. Além destes números, o padrão IEEE 754 admitem os chamados números subnormais, que não existiam nos sistemas aritméticos computacionais anteriores e permitem que o underflow seja gradual. Eles são números não-zero com um *significando* n não normalizado e o expoente mínimo $k = 2 - 2^7 = -126$:

$$2^{k-23} n = \pm 2^k (0 + f), \quad 0 < |n| < 2^{23}, \quad f < 1.$$

Se o expoente tem bits todos 0, mas a mantissa é não zero, então o valor é um número subnormal, que não possui um bit 1 antes do ponto binário.

Alguns valores denormalizados são mostrados abaixo:

<i>Smallest positive (non-zero) float</i>	[0][0000 0000][1] .[0000 0000 0000 0000 0000 001]
<i>Smallest negative (non-zero) float</i>	[1][0000 0000][1].[0000 0000 0000 0000 0000 001]
<i>Largest denormalized float</i>	[1][0000 0000][1].[1111 1111 1111 1111 1111 111]

Uma discussão aprofundada sobre este assunto pode ser encontrada no artigo de Kahan [6].

Formato duplo

Conhecido em C como "double", este formato é implementado em 64 bits (8 bytes) da seguinte forma:

$$(-1)^s 2^{E-1023} \cdot \left(1 + \sum_{i=1}^{52} 2^{-i} \right),$$

sendo a distribuição de bits dada por

sinal	expoente	bit implícito	mantissa
[s]	[eeee eeee eee]	[1]	[mmm· ·mmm]
1 bit	11 bits		52 bits

Consideremos o expoente. Embora um número binário de 11 bits possa ter valores decimais entre $(0000\ 0000\ 000)_2$ e $(1111\ 1111\ 111)_2 = 2047_{10}$, estes valores extremos são reservados para números especiais, como veremos logo adiante. Para números normais os bits do expoente podem estar entre $(0000\ 0000\ 001)_2$ e $(1111\ 1111\ 110)_2$. Ou seja, $1 < E < 2046$, pois

```
[> restart
> EM := sum(2^k, k = 1 .. 10);
EM := 2046 (2.2.1)
```

O *bias* do expoente é 1023. Ou seja, o expoente $E_b = E - 1023$ pode estar no intervalo $-1022 < E_b < 1023$.

O número de dígitos significativos da parte fracionária da mantissa, representáveis na base decimal é

```
[> fsolve(2^52 = 10^x, x);
15.65355977 (2.2.2)
```

ou seja, entre 15 e 16 dígitos. O maior valor da mantissa é dado por

```
[> Digits := 16;
Digits := 16 (2.2.3)
```

```
[> m[max] := evalf(sum(2^(-k), k = 1 .. 52));
m_max := 0.9999999999999998 (2.2.4)
```

Portanto, o maior número decimal que pode ser representado com precisão simples é :

$$\begin{aligned} > \mathbf{x[\max]} := (1+m[\max]) * 10^{(EM-1023)} ; \\ & \qquad \qquad \qquad x_{\max} := 2.0000000000000000 10^{1023} \end{aligned} \quad (2.2.5)$$

O menor número neste sistema é

$$\begin{aligned} > x_{\min} := 1 \cdot 2^{-1022} \\ & \qquad \qquad \qquad x_{\min} := 2.225073858507201 10^{-308} \end{aligned} \quad (2.2.6)$$

A acuracidade de máquina é $(1.00 \dots 001)_2 \cdot 2^0 = 2^{-52}$

$$\begin{aligned} > 2^{(-52)} \\ & \qquad \qquad \qquad 2.220446049250313 10^{-16} \end{aligned} \quad (2.2.7)$$

Formato estendido

Conhecido em C como "long double", este formato é implementado em 80 bits (10 bytes) da seguinte forma:

$$\begin{aligned} & (-1)^s \cdot 2^{E-16383} * (m), \\ & (-1)^s 2^{E-16383} \cdot \left(1 + \sum_{i=1}^{63} 2^{-i} \right) \end{aligned}$$

sendo o a distribuição de bits dada por

sinal	expoente	bit implícito	mantissa
[s]	[eeee...eee]	[1]	[mmm...mmm]
1 bit	15 bits		63 bits

O número de dígitos significativos da parte fracionária da mantissa, representáveis na base decimal é

$$\begin{aligned} > \mathit{evalf}(63 * \log_{10}(2)); \\ & \qquad \qquad \qquad 18.96488972683081 \end{aligned} \quad (2.3.1)$$

Ou seja, a precisão decimal na parte fracionária da mantissa é de 18 ou 19 dígitos. Esta é a precisão utilizada para cálculos realizados por chips Intel.

Exemplos e exercícios

Exemplo1. Converter o número -11.5 para o formato de ponto flutuante IEEE de precisão simples (32 bits)

Passos:

1. Converter para a base binária:

$$\begin{aligned} > \mathbf{convert(11, binary)} ; \\ & \qquad \qquad \qquad 1011 \end{aligned} \quad (2.4.1)$$

Portanto, $(-11.5)_{10} = (-1011.1)_2$.

2. Converter para a notação binária científica normalizada:

$$-1011.1 = -1.011 \cdot 2^3 = -1.011.1 \cdot 2^{E-127}$$

3. Determinar m(24bits), e(8bits) , s(1bit):

$$m = 1.011000000000000000000000$$

$$E - 127 = 3 = 00000011$$

$$s = 1 ,$$


Note que o expoente a ser armazenado é $E = 130$ ou 10000010 .


4. Montar a palavra de 32 bits:


$$- 1.011000000000000000000000 E_2 00000011 \text{ ou}$$


$$[1] [1000 0010] [1] . [0110 0000 0000 0000 0000 000] .$$


Exercícios:


: Resolva manualmente (no máximo com calculadora)

: Resolva no computador.

 1. Converter o número -212.125 para o formato de ponto flutuante IEEE 754 de precisão simples (32 bits).

 2. Converter o número -22212.25 para o formato de ponto flutuante IEEE 754 de precisão simples (32 bits).

 3. Converter o número -2423342.1245 para o formato de ponto flutuante IEEE 754 de precisão dupla (64 bits).

 4. Converta o número π para o formato de ponto flutuante IEEE 754 de precisão estendida. Use truncamento.

Referências

1. IEEE Computer Society (1985) IEEE Standard for Binary Floating-Point Arithmetic, IEEE Std 754 -1985.
2. Comparing floating point numbers, Bruce Dawson. <http://www.cygnus-software.com/papers>
3. W. Kahan, Lecture Notes on the Status of. IEEE Standard 754 for Binary Floating-Point Arithmetic, www.cs.berkeley.edu/~wkahan/ieee754status/ieee754.ps
4. Prof. W. Kahan's web pages, www.cs.berkeley.edu/~wkahan/
5. An Interview with the Old Man of Floating-Point, <http://www.cs.berkeley.edu/~wkahan/ieee754status/754story.html>
6. W. Kahan, IEEE Standard 754 for Binary Floating-Point Arithmetic, www.cs.berkeley.edu/~wkahan/ieee754status/ieee754.ps