Métodos Numéricos para Engenharia

MÓDULO 6 – ÉPSILON DA REPRESENTAÇÃO E CONDICIONAMENTO DE ALGORITMOS

PROFESSOR LUCIANO NEVES DA FONSECA

Soma de Floats

1) Igualar expoentes e deslocar mantissas
   somar 1 ao expoente equivale a deslocar a mantissa de 1 ->
   subtrair 1 ao expoente equivale a deslocar a mantissa de 1 <-
2) Somar as mantissas

Exemplo: Somar (2 + 1) = 3
Com Float(3,4) n=3, m=4

2 = [0] [100] [1] [000]
1 = [0] [011] [1] [000]

```
--> b1=dec2float(2,3,4,%t);
[0] [100] [1.000]
--> b2=dec2float(1,3,4,%t);
[0] [011] [1.000]
```

Somar 1 ao menor expoente
Deslocar mantissa menor de 1 ->:

  2 = [0] [100] [1] [000]
\+ 1(d) = [0] [100] [0] [100]

---

3 = [0] [100] [1] [100]

```
--> soma=SomaFloat_b(b1,b2,3,4,%t);
e1     = [0] [100][1] [000]
e2     = [0] [011][1] [000]
mantissa x2 deslocada -> de 1 bits
e2d    = [0] [100][0] [100]
--------------------
e2d    = [0] [100][0] [100]
e1     = [0] [100][1] [000]
-------------------- +
soma   = [0] [100][1] [100]

--> float2dec(soma,3,4)

  3.
```

$|e_1| > |e_2|$ $e_1 + e_2$

```
function soma=SomaFloat_b(e1,e2,n,m,prt)
  exp1  = e1(2:n+1)  //exp1 e exp2 são os expoentes
  exp2  = e2(2:n+1)
  mant1  = [0 1 e1(n+2:n+m)] //mant1 e mant2 são as mantissas
  mant2  = [0 1 e2(n+2:n+m)]
  desloc=binario2dec1(SomaBinaria(exp1,comple2(exp2),%f))
  mant2d=zeros(1:m+1) //desloca mantissa direita para igualar expoentes
  mant2d(desloc+1:m+1)=mant2(1:m+1-desloc)
  e2_d = [e2(1) exp1 mant2d(3:m+1)]
  mant_s = SomaBinaria(mant1,mant2d,%f) //soma mantissas
  soma = [0 exp1 mant_s(3:m+1)]
  if(mant_s(1)==1)  then  // vai 1 na soma
      soma_d = soma //deslocar mantissa direita e somar 1 ao expoente
      soma = [0 SomaBinaria(exp1,[0 1],%f) mant_s(2:m)]
  end
    :
endfunction
```

Soma de Floats

Exemplo: Somar (9 + 2) = 11
Com Float(3,4) n=3, m=4

9 = [0] [110] [1] [001]
2 = [0] [100] [1] [000]

```
--> b1=dec2float(9,3,4,%t);
[0] [110] [1.001]
--> b2=dec2float(2,3,4,%t);
[0] [100] [1.000]
```

Somar 2 ao menor expoente
Deslocar mantissa menor de 2 ->:

9 = [0] [110] [1] [001]
\+ 2(d) = [0] [111] [0] [010]

---

11 = [0] [110] [1] [011]

```
--> soma=SomaFloat_b(b1,b2,3,4,%t);
e1     = [0] [110][1] [001]
e2     = [0] [100][1] [000]
mantissa x2 deslocada -> de 2 bits
e2d    = [0] [110][0] [010]
--------------------
e2d    = [0] [110][0] [010]
e1     = [0] [110][1] [001]
-------------------- +
soma   = [0] [110][1] [011]

--> float2dec(soma,3,4)

  11.
```

Exemplo: Somar (3 + 2) = 5
Com Float(3,4) n=3, m=4

3 = [0] [100] [1] [100]
2 = [0] [100] [1] [000]

```
--> b1=dec2float(3,3,4,%t);
[0] [100] [1.100]
--> b2=dec2float(2,3,4,%t);
[0] [100] [1.000]
```

\+ 3 = [0] [100] [1] [100]
2 = [0] [100] [1] [000]

5d = [0] [100] [^{1}0] [100]

Vai um ! Deslocar mantissa soma de 1 -> Expoente +1

5 = [0] [101] [1] [010]

```
--> soma=SomaFloat_b(b1,b2,3,4,%t);
e1     = [0] [100][1] [100]
e2     = [0] [100][1] [000]
mantissa x2 deslocada -> de 0 bits
e2d    = [0] [100][1] [000]
--------------------
e2d    = [0] [100][1] [000]
e1     = [0] [100][1] [100]
-------------------- +
soma_d = [0] [100][(+1) 0] [100]
Vai um! mantisa soma delocada de -> 1 bits
soma   = [0] [101][1] [010]

--> float2dec(soma,3,4)

  5.
```

Subtração de Floats

Exemplo: Somar (7 -5) = 2
Com Float(3,4) n=3, m=4

```
--> b1=dec2float(7,3,4,%t);
[0] [101] [1.110]

--> b2=dec2float(5,3,4,%t);
[0] [101] [1.010]
```

Complemento 2 de mantissa negativa, somar

7	= [0] [101] [1] [110]
+ 5(c2)	= [1] [101] [0] [110]
2d	= [0] [101] [0] [100]

Deslocar mantissa soma de 1 <-, expoente -1

2 = [0] [100] [1] [000]

```
--> soma=SubtFloat_b(b1,b2,3,4,%t);
e1      = [0] [101][1] [110]
e2      = [0] [101][1] [010]
mantissa x2 deslocada -> de 2 bits
e2d     = [0] [101][1] [010]
--------------------
e2d(c2)= [1] [101][0] [110]
e1      = [0] [101][1] [110]
-------------------- +
soma_d = [0] [101][0] [100]
mantissa soma deslocada de <- 1 bits
soma    = [0] [100][1] [000]

--> float2dec(soma,3,4)
  2.
```

1) Igualar expoentes e deslocar mantissas
somar 1 ao expoente equivale a deslocar a mantissa de 1 ->
subtrair 1 ao expoente equivale a deslocar a mantissa de 1 <-
2) Subtrair as mantissas com complemento 2

$|e_1| > |e_2|$ $e_1 - e_2$

```
function soma=SubtFloat_b(e1,e2,n,m,prt)
    exp1  = e1(2:n+1)  //exp1 e exp2 são os expoentes
    exp2  = e2(2:n+1)
    mant1  = [0 1 e1(n+2:n+m)] //mant1 e mant2 são as mantissas
    mant2  = [0 1 e2(n+2:n+m)]
    desloc=binario2dec1(SomaBinaria(exp1,comple2(exp2),%f))
    mant2d=zeros(1:m+1) //desloca mantissa direita para igualar expoentes
    mant2d(desloc+1:m+1)=mant2(1:m+1-desloc)
    e2_d = [e2(1) exp1 mant2d(3:m+1)]
    mant_s = SomaBinaria(mant1,comple2(mant2d),%f) //subtair mantissas
    soma_d = [0 exp1 mant_s(3:m+1)]
    lista_uns = find(mant_s(1:m+1)==1)
    if(lista_uns==[]) // mantissa zerada
        soma = [0 zeros(1:n+m-1)]
    else  //deslocar mantissa para esquerda e subtrair expoente
        desloc=lista_uns(1) -1 // encontrar primeiro '1'
        mant_sd=zeros(1:m+1)
        mant_sd(1:m+2-desloc)=mant_s(desloc:m+1)
        exp_s = SomaBinaria(exp1,dec2binario3(-desloc+1,n),%f)
        soma = [0 exp_s mant_sd(3:m+1)]
    end
      ⋮
endfunction
```

Subtração de Floats

Exemplo: Somar (20 -16) = 4
Com Float(3,4) n=3, m=4

```
--> b1=dec2float(20,3,4,%t);
[0] [111] [1.010]

--> b2=dec2float(16,3,4,%t);
[0] [111] [1.000]
```

Complemento 2 de mantissa negativa, somar

```
   20      = [0] [111] [1] [010]
+  16(c2)  = [1] [111] [1] [000]
----------------------------------
   4d  = [0] [111] [0] [010]
```

Deslocar mantissa soma de 2 <-, expoente -2

4 = [0] [101] [1] [000]

```
--> soma=SubtFloat_b(b1,b2,3,4,%t);
e1      = [0] [111][1] [010]
e2      = [0] [111][1] [000]
mantissa x2 deslocada -> de 3 bits
e2d     = [0] [111][1] [000]
--------------------
e2d(c2)= [1] [111][1] [000]
e1      = [0] [111][1] [010]
-------------------- +
soma_d = [0] [111][0] [010]
mantissa soma deslocada de <- 2 bits
soma    = [0] [101][1] [000]

--> float2dec(soma,3,4)
  4.
```

Exemplo: Somar (20 -18) = 2
Com Float(3,4) n=3, m=4

```
--> b1=dec2float(20,3,4,%t);
[0] [111] [1.010]

--> b2=dec2float(18,3,4,%t);
[0] [111] [1.001]
```

Complemento 2 de mantissa negativa, somar

```
   20      = [0] [111] [1] [010]
+  18(c2)  = [1] [111] [0] [111]
----------------------------------
   2d  = [0] [111] [0] [001]
```

Deslocar mantissa soma de 3 <-, expoente -3

2 = [0] [100] [1] [000]

```
--> soma=SubtFloat_b(b1,b2,3,4,%t);
e1      = [0] [111][1] [010]
e2      = [0] [111][1] [001]
Underflow: deslocamento -> maior que 4
mantissa x2 deslocada -> de 4 bits
e2d     = [0] [111][1] [001]
--------------------
e2d(c2)= [1] [111][0] [111]
e1      = [0] [111][1] [010]
-------------------- +
soma_d = [0] [111][0] [001]
mantissa soma deslocada de <- 3 bits
soma    = [0] [100][1] [000]

--> float2dec(soma,3,4)
  2.
```

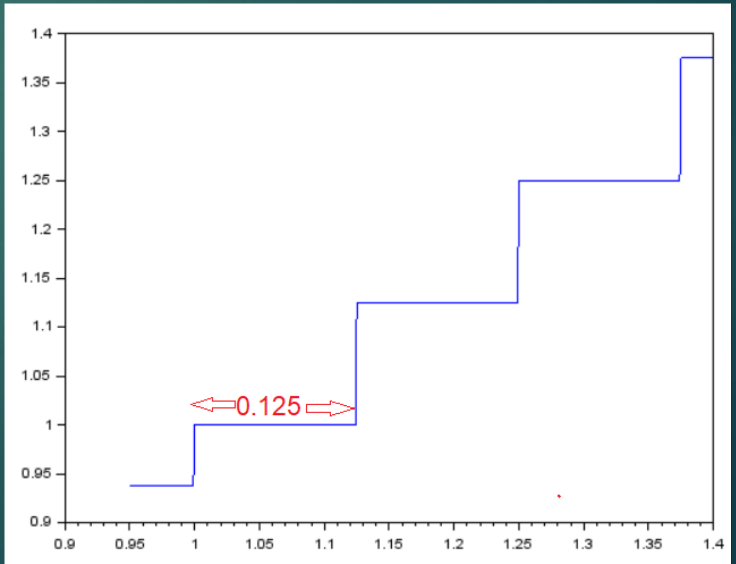
Épsilon da representação de Ponto Flutuante

- Épsilon de uma representação de ponto flutuante é o intervalo entre o número 1 (nesta representação) e o próximo número, maior que 1, que seja distinto de 1 no sistema numérico usado.
- Em outras palavras, se o número 1 estiver armazenado na memória do computador, o número real x original pode estar entre $1-\varepsilon < x < 1+\varepsilon$
- Em um float(n,m) $\quad \varepsilon = 2^{-m+1}$

```
function eps_def=epsilon(n,m,prt)
    eps_def = 2^-(m-1)
    bias    = 2^(n-1) -1;
    expl=dec2binario1(0.0 + bias , n)
    mant1=[zeros(1:m-1) 0]
    e1=[0 expl mant1]
    mant2=[zeros(1:m-2) 1]
    e2=[0 expl mant2]
    eps_sub_float=SubtFloat_b(e2,e1,n,m,prt)
    eps_sub=float2dec(eps_sub,n,m)
    printf("Epsilon Subtração = %.3e\n",eps_sub)
    printf("Epsilon 2^-(m-1)  = %.3e\n",eps_def)
endfunction
```

Podemos também avaliar Graficamente o Épsilon

```
--> plot_epsilon_floating_point(3,4)
```



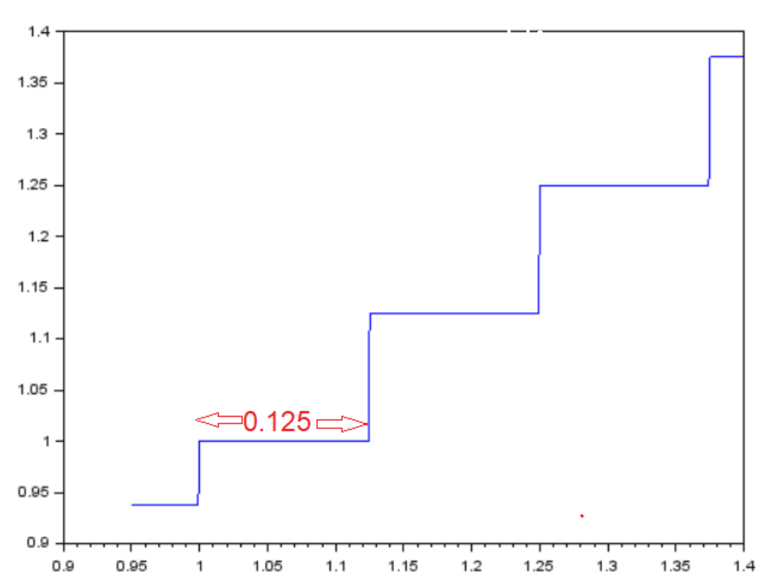
Cálculo exato do Épsilon de uma máquina com ponto flutuante por subtração e pela definição:

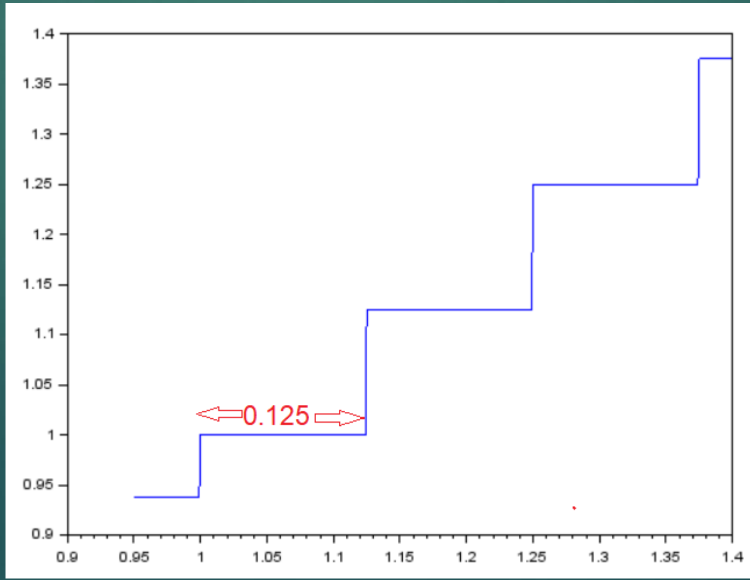
```
--> eps=epsilon(3,4,%t);
e1      = [0] [011][1] [001]
e2      = [1] [011][1] [000]
mantissa x2 deslocada -> de 0 bits
e2d     = [1] [011][1] [000]
--------------------
e2d(c2)= [0] [011][1] [000]
e1      = [0] [011][1] [001]
-------------------- +
soma_d = [0] [011][0] [001]
mantissa soma deslocada de <- 3 bits
soma    = [0] [000][1] [000]
Soma binária ( 3,4)      = 0.125000000000
Epsilon Subtração = 1.250e-01
Epsilon 2^-(m-1)   = 1.250e-01

--> plot_epsilon_floating_point(3,4)
```

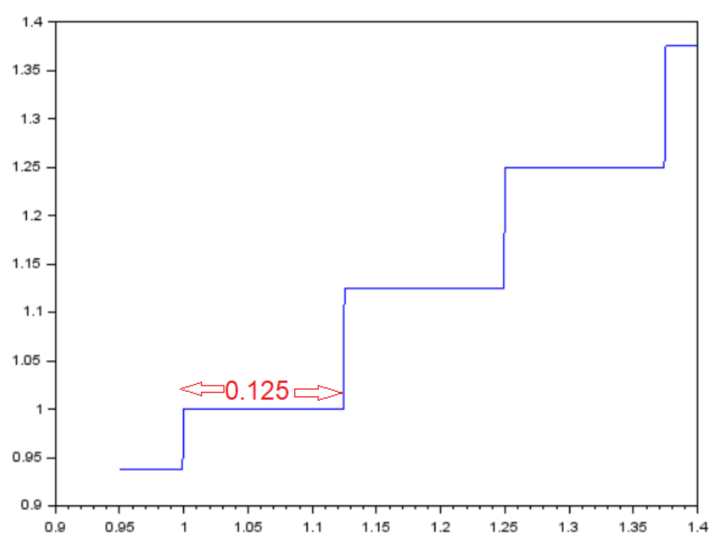
```
--> eps=epsilon(8,4,%t);
e1      = [0] [01111111][1] [001]
e2      = [1] [01111111][1] [000]
mantissa x2 deslocada -> de 0 bits
e2d     = [1] [01111111][1] [000]
--------------------
e2d(c2)= [0] [01111111][1] [000]
e1      = [0] [01111111][1] [001]
-------------------- +
soma_d = [0] [01111111][0] [001]
mantissa soma deslocada de <- 3 bits
soma    = [0] [01111100][1] [000]
Soma binária ( 8,4)      = 0.125000000000
Epsilon Subtração = 1.250e-01
Epsilon 2^-(m-1)   = 1.250e-01

--> plot_epsilon_floating_point(8,4)
```

```
--> eps=epsilon(16,4,%t);
e1      = [0] [0111111111111111][1] [001]
e2      = [1] [0111111111111111][1] [000]
mantissa x2 deslocada -> de 0 bits
e2d     = [1] [0111111111111111][1] [000]
--------------------
e2d(c2)= [0] [0111111111111111][1] [000]
e1      = [0] [0111111111111111][1] [001]
-------------------- +
soma_d = [0] [0111111111111111][0] [001]
mantissa soma deslocada de <- 3 bits
soma    = [0] [0111111111111100][1] [000]
Soma binária ( 16,4)     = 0.125000000000
Epsilon Subtração = 1.250e-01
Epsilon 2^-(m-1)   = 1.250e-01

--> plot_epsilon_floating_point(16,4)
```

Vemos que o Épsilon não depende do tamanho do expoente





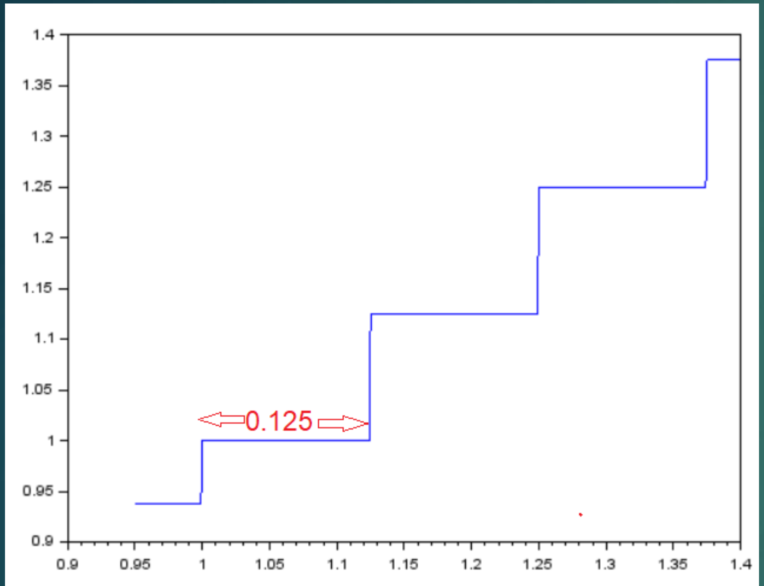


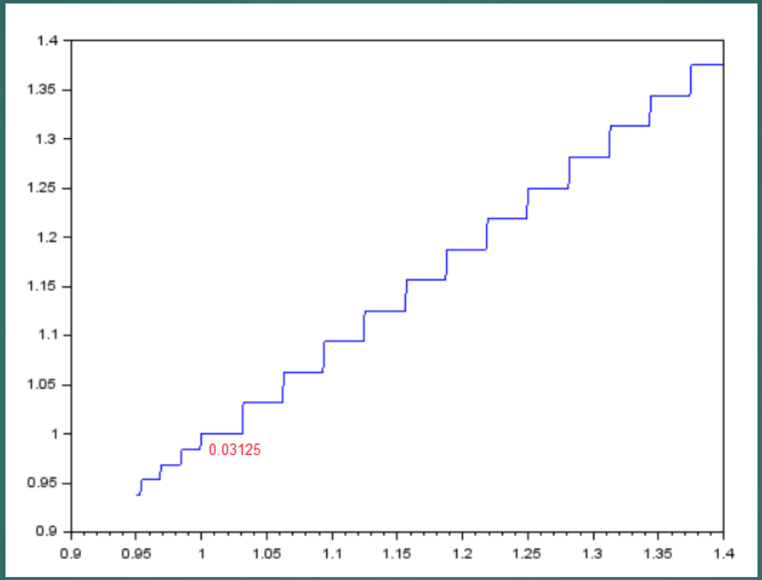
O Epsilon tem porém uma forte dependência com o tamanho m da mantissa

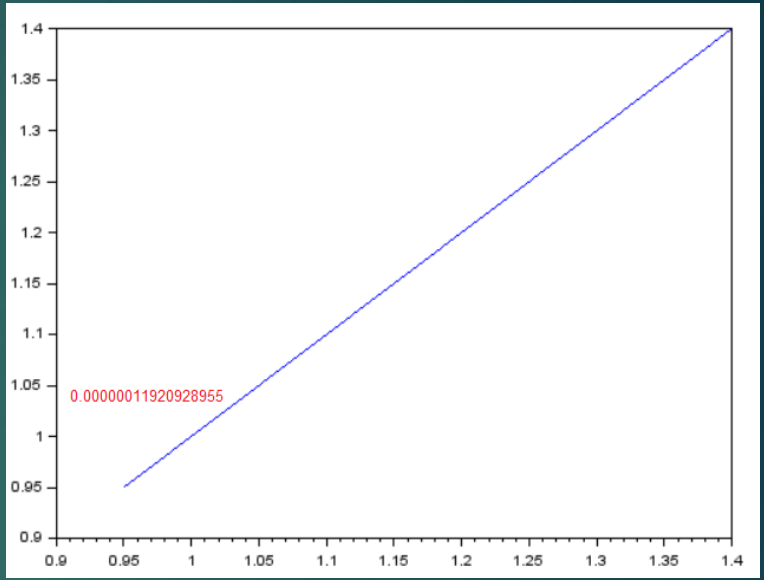
```
--> eps=epsilon(3,4);
Epsilon Subtração = 1.250e-01
Epsilon 2^-(m-1)  = 1.250e-01

--> plot_epsilon_floating_point(3,4)
```

```
--> eps=epsilon(3,6);
Epsilon Subtração = 3.125e-02
Epsilon 2^-(m-1)  = 3.125e-02

--> plot_epsilon_floating_point(3,6)
```

```
--> eps=epsilon(3,24);
Epsilon Subtração = 1.192e-07
Epsilon 2^-(m-1)  = 1.192e-07

--> plot_epsilon_floating_point(3,24)
```








```
--> eps=epsilon(8,24);
Epsilon Subtração = 1.192e-07
Epsilon 2^-(m-1)  = 1.192e-07
```

```
--> eps=epsilon(11,53);
Epsilon Subtração = 2.220e-16
Epsilon 2^-(m-1)  = 2.220e-16
```

No padrão IEEE o épsilon do float (n=8,m=24) $= 1.19\ 10^{-7} = 2^{-23}$

No padrão IEEE o épsilon do double (n=11, m= 53) $= 2.22\ 10^{-16} = 2^{-52}$

SomaFloat ()

recebe como argumento quaisquer x_1 e x_1

Para isso força:

Força $|x_1| > |x_2|$

Força $|x_1| > 0$

Antes de chamar SomaFloat_b() ou SubtFloat_b()

```
function soma_nm=SomaFloat(x1,x2,n,m,prt)
  soma_exata = x1+x2
  if (prt)  printf("(x1,x2)=(%.2f,%.2f)\n",x1,x2)  end
  if(abs(x2)>abs(x1)) then  // Forçar abs(x1)>abs(x2)
      temp=x1;
      x1=x2;
      x2=temp;
      if (prt)
          printf("Trocar x1 por x2\n")
          printf("(x1,x2)=(%.2f,%.2f)\n",x1,x2)
      end
  end
  trocar_sinais=0;
  if( x1<0 ) then  //Forçar x1>0
       trocar_sinais=1;
       x1=-x1;
       x2=-x2;
       if (prt)
          printf("Trocar Sinais de x1 e x2\n",x1)
          printf("(x1,x2)=(%.2f,%.2f)\n",x1,x2)
       end
  end
  e1=dec2float(x1,n,m,%f) //e1 e e2 são os floats equivalente
  e2=dec2float(x2,n,m,%f)
  if (x1*x2>=0)
       somafloat_nm=SomaFloat_b(e1,e2,n,m,prt);
  else
       somafloat_nm=SubtFloat_b(e1,e2,n,m,prt);
  end
  if (trocar_sinais)  somafloat_nm(1)=bitcmp(somafloat_nm(1),1) end
  soma_nm=float2dec(somafloat_nm,n,m)
      ⋮
 endfunction
```

Fontes de erro de arredondamento
1) Somar um número muito pequeno a um número grande

$1024 + 0.1$

```
--> SomaFloat(1024,0.1,8,24,%t)
(x1,x2)=(1024.000000,0.100000)
e1     = [0] [10001001][1] [00000000000000000000000]
e2     = [0] [01111011][1] [10011001100110011001100]
mantissa x2 deslocada -> de 14 bits
e2d    = [0] [10001001][0] [00000000000001100110011]
--------------------

e2d    = [0] [10001001][0] [00000000000001100110011]
e1     = [0] [10001001][1] [00000000000000000000000]
-------------------- +

soma   = [0] [10001001][1] [00000000000001100110011]
Soma binária ( 8,24)     = 1024.099975585938
Soma exata   (11,53)     = 1024.100000000000
```

$1024 + 0.001$

```
--> SomaFloat(1024,0.001,8,24,%t)
(x1,x2)=(1024.000000,0.001000)
e1     = [0] [10001001][1] [00000000000000000000000]
e2     = [0] [01110101][1] [00000110001001001101110]
mantissa x2 deslocada -> de 20 bits
e2d    = [0] [10001001][0] [00000000000000000001000]
--------------------

e2d    = [0] [10001001][0] [00000000000000000001000]
e1     = [0] [10001001][1] [00000000000000000000000]
-------------------- +

soma   = [0] [10001001][1] [00000000000000000001000]
Soma binária ( 8,24)     = 1024.000976562500
Soma exata   (11,53)     = 1024.001000000000
```

$1024 + 0.0001$

```
--> SomaFloat(1024,0.0001,8,24,%t)
(x1,x2)=(1024.000000,0.000100)
e1     = [0] [10001001][1] [00000000000000000000000]
e2     = [0] [01110001][1] [10100011011011100010111]
Underflow: deslocamento -> maior que 24
mantissa x2 deslocada -> de 24 bits
e2d    = [0] [10001001][0] [00000000000000000000000]
--------------------

e2d    = [0] [10001001][0] [00000000000000000000000]
e1     = [0] [10001001][1] [00000000000000000000000]
-------------------- +

soma   = [0] [10001001][1] [00000000000000000000000]
Soma binária ( 8,24)     = 1024.000000000000
Soma exata   (11,53)     = 1024.000100000000
```

- Notar que o erro de arredondamento aumenta com o deslocamento do segundo operando
- Notar que quando o deslocamento do segundo operando é maior que 23 bits, a sua mantissa será zerada, o causa um grande erro de arredondamento (a soma é ignorada!)

Epsilon(8,24)=$2^{-23} = 1.192\ \ 10^{-7}$

$1 + 1.193\ 10^{-7}$

```
--> SomaFloat(1,1.193e-7,8,24,%t)
(x1,x2)=(1.000000e+00,1.193000e-07)
e1      = [0] [01111111][1] [00000000000000000000000]
e2      = [0] [01101000][1] [00000000001100011101111]
mantissa x2 deslocada -> de 23 bits
e2d     = [0] [01111111][0] [00000000000000000000001]
--------------------
e2d     = [0] [01111111][0] [00000000000000000000001]
e1      = [0] [01111111][1] [00000000000000000000000]
-------------------- +
soma    = [0] [01111111][1] [00000000000000000000001]
Soma binária ( 8,24)     = 1.000000119209
Soma exata   (11,53)     = 1.000000119300
```

$1 + 1.192\ 10^{-7}$

```
--> SomaFloat(1,1.192e-7,8,24,%t)
(x1,x2)=(1.000000e+00,1.192000e-07)
e1      = [0] [01111111][1] [00000000000000000000000]
e2      = [0] [01100111][1] [11111111111101011100100]
Underflow: deslocamento -> maior que 24
mantissa x2 deslocada -> de 24 bits
e2d     = [0] [01111111][0] [00000000000000000000000]
--------------------
e2d     = [0] [01111111][0] [00000000000000000000000]
e1      = [0] [01111111][1] [00000000000000000000000]
-------------------- +
soma    = [0] [01111111][1] [00000000000000000000000]
Soma binária ( 8,24)     = 1.000000000000
Soma exata   (11,53)     = 1.000000119200
```

- Quando o primeiro operando tem valor 1 e o segundo operando for exatamente igual ao épsilon do sistema numérico, o deslocamento será 24 bits, o que vai zerar a mantissa e causar grandes erros de arredondamento
- Então épsilon é o menor número que podemos somar ao número 1
- De maneira similar, x*épsilon é menor número que podemos soma a um número x.

Fontes de erro de arredondamento
2) Subtrair dois números da mesma ordem de grandeza

$1024.1 - 1024$

```
--> SomaFloat(1024.1,-1024,8,24,%t)
(x1,x2)=(1024.100000,-1024.000000)
e1      = [0] [10001001][1] [00000000000001100110011]
e2      = [1] [10001001][1] [00000000000000000000000]
mantissa x2 deslocada -> de 0 bits
e2d     = [1] [10001001][1] [00000000000000000000000]
--------------------
e2d(c2)= [1] [10001001][1] [00000000000000000000000]
e1      = [0] [10001001][1] [00000000000001100110011]
-------------------- +
soma_d = [0] [10001001][0] [00000000000001100110011]
mantissa soma deslocada de <- 14 bits
soma    = [0] [01111011][1] [10011001100000000000000]
Soma binária ( 8,24)     = 0.099975585938
Soma exata    (11,53)     = 0.100000000000
```

$1024.001 - 1024$

```
--> SomaFloat(1024.001,-1024,8,24,%t)
(x1,x2)=(1024.001000,-1024.000000)
e1      = [0] [10001001][1] [00000000000000000001000]
e2      = [1] [10001001][1] [00000000000000000000000]
mantissa x2 deslocada -> de 0 bits
e2d     = [1] [10001001][1] [00000000000000000000000]
--------------------
e2d(c2)= [1] [10001001][1] [00000000000000000000000]
e1      = [0] [10001001][1] [00000000000000000001000]
-------------------- +
soma_d = [0] [10001001][0] [00000000000000000001000]
mantissa soma deslocada de <- 20 bits
soma    = [0] [01110101][1] [00000000000000000000000]
Soma binária ( 8,24)     = 0.000976562500
Soma exata    (11,53)     = 0.001000000000
```

$1024.0001 - 1024$

```
--> SomaFloat(1024.0001,-1024,8,24,%t)
(x1,x2)=(1024.000100,-1024.000000)
e1      = [0] [10001001][1] [00000000000000000000000]
e2      = [1] [10001001][1] [00000000000000000000000]
mantissa x2 deslocada -> de 0 bits
e2d     = [1] [10001001][1] [00000000000000000000000]
--------------------
e2d(c2)= [1] [10001001][1] [00000000000000000000000]
e1      = [0] [10001001][1] [00000000000000000000000]
-------------------- +
soma_d = [0] [10001001][0] [00000000000000000000000]
Underflow: matissa nula após deslocamento <-
soma    = [0] [00000000][1] [00000000000000000000000]
Soma binária ( 8,24)     = 0.000000000000
Soma exata    (11,53)     = 0.000100000000
```

- Notar que a subtração de dois número próximos também causa grande deslocamento da mantissa
- Notar que o erro de arredondamento aumenta com o deslocamento do segundo operando
- Notar que quando o deslocamento do segundo operando for maior que 23 bits, a sua mantissa será zerada, o causa um grande erro de arredondamento (o resultado será nulo)

$$\text{Epsilon}(8,24) = 2^{-23} = 1.192\ \ 10^{-7}$$

$$(1 + 1.193\ 10^{-7})\ - 1$$

```
--> SomaFloat(1+1.193e-7,-1,8,24,%t)
(x1,x2)=(1.000000,-1.000000)
e1      = [0] [01111111][1] [00000000000000000000001]
e2      = [1] [01111111][1] [00000000000000000000000]
mantissa x2 deslocada -> de 0 bits
e2d     = [1] [01111111][1] [00000000000000000000000]
--------------------
e2d(c2)= [0] [01111111][1] [00000000000000000000000]
e1      = [0] [01111111][1] [00000000000000000000001]
-------------------- +
soma_d = [0] [01111111][0] [00000000000000000000001]
mantissa soma deslocada de <- 23 bits
soma    = [0] [01101000][1] [00000000000000000000000]
Soma binária ( 8,24)     = 0.000000119209
Soma exata   (11,53)     = 0.000000119300
```

$$(1 + 1.192\ 10^{-7})\ - 1$$

```
--> SomaFloat(1+1.192e-7,-1,8,24,%t)
(x1,x2)=(1.000000,-1.000000)
e1      = [0] [01111111][1] [00000000000000000000000]
e2      = [1] [01111111][1] [00000000000000000000000]
mantissa x2 deslocada -> de 0 bits
e2d     = [1] [01111111][1] [00000000000000000000000]
--------------------
e2d(c2)= [0] [01111111][1] [00000000000000000000000]
e1      = [0] [01111111][1] [00000000000000000000000]
-------------------- +
soma_d = [0] [01111111][0] [00000000000000000000000]
Underflow: matissa nula após deslocamento <-
soma    = [0] [00000000][1] [00000000000000000000000]
Soma binária ( 8,24)     = 0.000000000000
Soma exata   (11,53)     = 0.000000119200
```

- Quando o primeiro operando tem valor (1+épislon)e o segundo operando for (1), o deslocamento será 24 bits, o que vai zerar a mantissa e causar grandes erros de arredondamento
- Então delta=épsilon é a menor diferença detectável entre (1+delta) e (1)]

Condicionamento de algoritmos

- Reduzir os efeitos das principais fontes de erros de arredondamento.
- 1) Overflow - Pode ser evitado com um expoente n maior
- 2) Underflow - Pode ser evitado com uma mantissa m maior
- 3) Somar um número muito pequeno a um número muito grande
- 4) Subtrair dois números muito próximos.

Entre os erro 3 e 4, qual o mais problemático?

1) Somatória: somar n vezes um número muito pequeno (dx) a um número grande (x)

Para condicionarmos este algoritmo, a) ou substituímos a soma iterativa por uma multiplicação. b) Ou somamos primeiro os números pequenos entre si, para depois somarmos este resultado parcial ao numero maior.
Assim evitamos somar números com ordens de grandeza muito diferentes

```
function soma1_cond(N,n,m)
    x=1.0
    dx=1.0/N;
    S=x
    for i = 1:N
        S=SomaFloat(S,dx,n,m,%f)
    end
    Soma_cond = x + N*dx
    printf("Soma Iterativa     = %.8f\n",S)
    printf("Soma Condicionada  = %.8f\n",Soma_cond)
endfunction
```

$$
\begin{aligned}
&S = x\\
&\quad for\ (i = 1{:}\,N)\\
&\quad S = S + dx\\
&end
\end{aligned}
$$

```
function soma2_cond(N,n,m)
    x=1.0
    dx=1.0/N;
    S=x
    S_cond=dx
    for i = 1:N-1
        S=SomaFloat(S,dx,n,m,%f)
        S_cond=SomaFloat(S_cond,dx,n,m,%f)
    end
    S = SomaFloat(S,dx,n,m,%f)
    S_cond = SomaFloat(S_cond,x,n,m,%f)
    printf("Soma Iterativa     = %.8f\n",S)
    printf("Soma Condicionada  = %.8f\n",S_cond)
endfunction
```

```
--> N=1000;

--> soma1_cond(N,8,24)
Soma Iterativa      = 1.99992752
Soma Condicionada   = 2.00000000
```

$$S_{cond} = x + (N)dx$$

$$
\begin{aligned}
&S_{cond} = dx\\
&for\ (i = 1{:}\,N - 1)\\
&\quad S_{cond} = S_{cond} + dx\\
&end\\
&S_{cond} = S_{cond} + x
\end{aligned}
$$

```
--> N=1000;

--> soma2_cond(N,8,24)
Soma Iterativa      = 1.99992752
Soma Condicionada   = 1.99998832
```

```
--> N=1000000;

--> soma1_cond(N,8,24)
Soma Iterativa      = 1.95367432
Soma Condicionada   = 2.00000000
```

```
--> N=1000000;

--> soma2_cond(N,8,24)
Soma Iterativa      = 1.95367432
Soma Condicionada   = 1.97246170
```

2) Subtrair dois números da mesma ordem de grandeza

Para condicionar este algoritmo temos que substituir a subtração por um cálculo alternativo.

Raízes de $y = x^2 + bx + c$

$$y = x^2 - 100.0001x + 0.01$$

```
--> ps=poly([0.01,-100.0001,1],"s","coeff")
 ps  =

  0.01 -100.0001s +s²
```

$$x_1 = \left(-\frac{b}{2a}\right) + \left(\frac{\sqrt{b^2 - 4ac}}{2a}\right) = 50.001 + 49.9999$$

$$x_2 = \left(-\frac{b}{2a}\right) - \left(\frac{\sqrt{b^2 - 4ac}}{2a}\right) = 50.001 - 49.9999$$

$$mas \;\; x_1 x_2 = \frac{c}{a}, \qquad então: \;\; x_{2\_cond} = \frac{c}{a x_1}$$

```
--> baskara_cond(ps,8,24)
ps=(1.000)x^2+(-100.000)*x+(0.010)
b2      = -b/(2a)=50.000050
delta2 =  sqrt(b^2 - 4ac)/(2a)=49.9999
x1 = b2 + delta2 =50.0001  + 49.9999
x2 = b2 - delta2 =50.0001  - 49.9999
             (subtração 2 números próximos!!
condicionamento x1*x2 = c/a ->  x2=c/(a*x1)
raizes exatas  = 100.00000000   0.00010000      (condicionamento)
raizes ( 8,24) = 100.00000000   0.00009918      (0.00010000)
```

```
--> baskara_cond(ps,8,20)
ps=(1.000)x^2+(-100.000)*x+(0.010)
b2      = -b/(2a)=50.000050
delta2 =  sqrt(b^2 - 4ac)/(2a)=49.9999
x1 = b2 + delta2 =50.0001  + 49.9999
x2 = b2 - delta2 =50.0001  - 49.9999
             (subtração 2 números próximos!!
condicionamento x1*x2 = c/a ->  x2=c/(a*x1)
raizes exatas  = 100.00000000   0.00010000      (condicionamento)
raizes ( 8,20) = 99.99987793    0.00006104      (0.00010000)
```

```
function baskara_cond(ps,n,m)
    coef = coeff(ps)
    a=coef(3)
    b=coef(2)
    c=coef(1)
    b2     = -(b)/(2*a)
    delta2 =   sqrt(   SomaFloat(b^2,-4*a*c,n,m) )  /(2*a)
    x1 = SomaFloat(b2,+delta2,n,m)
    x2 = SomaFloat(b2,-delta2,n,m)
    x2_cond = c/(a*x1)
    .
    .
    .
endfunction
```

3) Subtrair dois números da mesma ordem de grandeza

Para condicionar este algoritmo utilizamos série de Taylor

$$y(x) = \frac{1}{dx}(\sin(x+dx) - \sin(x))$$

$$\sin(x+dx) \cong \sin(x) + (dx)\cos(x) + \frac{(dx)^2}{2}\sin(x)$$

$$y(x) = \frac{1}{dx}(\sin(x) + (dx)\cos(x) + \frac{(dx)^2}{2}\sin(x) - \sin(x))$$

$$y_{cond}(x) \approx \cos(x) + \frac{dx}{2}\sin(x)$$

```
--> taylor_cond(1e-6,8,24)
Calcular y=1/dx*(sin(x+dx)-sin(x)), dx pequeno
subtração 2 números próximos!!
Condicionamento por Taylor y_cond= cos(x)-0.5*dx*sin(x)
dx=1.0e-06
y( 8,24) = 0.5364418029785156    (0.5403018593788147)
```

```
--> taylor_cond(1e-7,8,24)
Calcular y=1/dx*(sin(x+dx)-sin(x)), dx pequeno
subtração 2 números próximos!!
Condicionamento por Taylor y_cond= cos(x)-0.5*dx*sin(x)
dx=1.0e-07
y( 8,24) = 0.5960464477539063    (0.540302276611328l)
```

```
--> taylor_cond(1e-8,8,24)
Calcular y=1/dx*(sin(x+dx)-sin(x)), dx pequeno
subtração 2 números próximos!!
Condicionamento por Taylor y_cond= cos(x)-0.5*dx*sin(x)
dx=1.0e-08
y( 8,24) = 0.0000000000000000    (0.540302276611328l)
```

```
function taylor_cond(dx,n,m)
    x=1
    y = 1/dx*SomaFloat(sin(x+dx),-sin(x),n,m)
    y_cond = SomaFloat(cos(x),-0.5*dx*sin(x),n,m)
        :
endfunction
```

4) Subtrair f(x) com argumento próximos

Para condicionar este algoritmo utilizamos a derivada

$$y = f(x + dx) - f(x)$$

$$f'(x) \approx \frac{f(x + dx) - f(x)}{dx}$$

$$y_{cond} \approx f'(x) * dx$$

$Exemplo$:

$f(x) = \text{atan}(x^2)$

```
--> deff("y=f(x)","y=atan(x^2)")
```

```
--> derivada_cond(2,1e-6,f,8,24)
Calcular y=f(x+dx)-f(x), dx pequeno
subtração 2 números próximos!!
Condicionamento por derivada y_cond= (dx)f'(x)
y( 8,24) = 2.38418579e-07         (2.35294118e-07)
```

```
--> derivada_cond(2,1e-7,f,8,24)
Calcular y=f(x+dx)-f(x), dx pequeno
subtração 2 números próximos!!
Condicionamento por derivada y_cond= (dx)f'(x)
y( 8,24) = 5.87747175e-39         (2.35294118e-08)
```

```
--> derivada_cond(2,1e-8,f,8,24)
Calcular y=f(x+dx)-f(x), dx pequeno
subtração 2 números próximos!!
Condicionamento por derivada y_cond= (dx)f'(x)
y( 8,24) = 5.87747175e-39         (2.35294118e-09)
```

```
function derivada_cond(x,dx,f,n,m)
    y = SomaFloat(f(x+dx),-f(x),n,m)
    y_cond = dx*numderivative(f,x)
              :
endfunction
```

5) Subtrair dois números da mesma ordem de grandeza

$$y = \frac{1 - \cos(x)}{\sin(x)}$$

Para condicionar este algoritmo utilizamos trigonometria

$$\sin^2\left(\frac{x}{2}\right) = \frac{1 - \cos(x)}{2} \qquad \cos^2\left(\frac{x}{2}\right) = \frac{1 + \cos(x)}{2}$$

$$y_{cond} = \frac{2\sin^2\left(\frac{x}{2}\right)}{\sin(x)}$$

```
--> trig_cond(1e-2,8,24)
Calcular y=(1-cos(x))/sin(x), x pequeno
subtração 2 números próximos!!
Condicionamento por trigonometria y_cond= 2*sin(x/2)^2/sin(x)
y( 8,24) = 0.005007      (0.005000)
```

```
function trig_cond(dx,n,m)
    y = 1/sin(dx)*SomaFloat(1,-cos(dx),n,m)
    y_cond = 2*sin(dx/2)^2/sin(dx)
    ⋮
endfunction
```

```
--> trig_cond(1e-3,8,24)
Calcular y=(1-cos(x))/sin(x), x pequeno
subtração 2 números próximos!!
Condicionamento por trigonometria y_cond= 2*sin(x/2)^2/sin(x)
y( 8,24) = 0.000596      (0.000500)
```

```
--> trig_cond(1e-4,8,24)
Calcular y=(1-cos(x))/sin(x), x pequeno
subtração 2 números próximos!!
Condicionamento por trigonometria y_cond= 2*sin(x/2)^2/sin(x)
y( 8,24) = 0.001192      (0.000050)
```

6) Subtrair dois números da mesma ordem de grandeza

Para condicionar este algoritmo propriedades do logaritmo

```
--> log_cond(2,1e-5,8,24)
Calcular y=log(x+dx)-log(x), x pequeno
subtração 2 números próximos!!
Condicionamento por logaritmo  y_cond= log((x+dx)/x)
dx=1.0e-05
y( 8,24) = 5.0068e-06   (5.0000e-06)
```

```
--> log_cond(2,1e-6,8,24)
Calcular y=log(x+dx)-log(x), x pequeno
subtração 2 números próximos!!
Condicionamento por logaritmo  y_cond= log((x+dx)/x)
dx=1.0e-06
y( 8,24) = 5.3644e-07   (5.0000e-07)
```

```
--> log_cond(2,1e-7,8,24)
Calcular y=log(x+dx)-log(x), x pequeno
subtração 2 números próximos!!
Condicionamento por logaritmo  y_cond= log((x+dx)/x)
dx=1.0e-07
y( 8,24) = 5.9605e-08   (5.0000e-08)
```

```
--> log_cond(2,1e-9,8,24)
Calcular y=log(x+dx)-log(x), x pequeno
subtração 2 números próximos!!
Condicionamento por logaritmo  y_cond= log((x+dx)/x)
dx=1.0e-09
y( 8,24) = 5.8775e-39   (5.0000e-10)
```

$$y = \log(x + dx) - \log(x)$$

$$\log(a) - \log(b) = \log\left(\frac{a}{b}\right)$$

$$y_{cond} = \log\left(\frac{x + dx}{x}\right)$$

```
function log_cond(x,dx,n,m)
    y = SomaFloat(log(x+dx),-log(x),n,m)
    y_cond = log((x+dx)/x)
        ⋮
endfunction
```

7) Subtrair dois números da mesma ordem de grandeza

Para condicionar este algoritmo utilizamos o produto notável
$(a+b)(a-b)=(a^2-b^2)$

$$y=\sqrt{x^2+1}-1$$

$$y_{\mathrm{cond}}=\left(\sqrt{x^2+1}-1\right)\frac{\left(\sqrt{x^2+1}+1\right)}{\left(\sqrt{x^2+1}+1\right)}=\frac{\mathrm{x}^2}{\left(\sqrt{x^2+1}+1\right)}$$

```
--> produtonotavel_cond(0.01,8,24)
Calcular y=sqrt(x^2+1)-1, x muito pequeno
subtração 2 números próximos!!
Condicionamento por produto notável  y_cond= x^2/(sqrt(x^2+1)+1)
dx=1.0e-02
y( 8,24) = 4.995e-05    (5.000e-05)
```

```
--> produtonotavel_cond(0.001,8,24)
Calcular y=sqrt(x^2+1)-1, x muito pequeno
subtração 2 números próximos!!
Condicionamento por produto notável  y_cond= x^2/(sqrt(x^2+1)+1)
dx=1.0e-03
y( 8,24) = 4.768e-07    (5.000e-07)
```

```
--> produtonotavel_cond(0.0001,8,24)
Calcular y=sqrt(x^2+1)-1, x muito pequeno
subtração 2 números próximos!!
Condicionamento por produto notável  y_cond= x^2/(sqrt(x^2+1)+1)
dx=1.0e-04
y( 8,24) = 5.877e-39    (5.000e-09)
```

```
function produtonotavel_cond(dx,n,m)
    y = SomaFloat(sqrt(dx^2+1),-1,n,m)
    y_cond = dx^2/SomaFloat(sqrt(dx^2+1),1,n,m)
    :
endfunction
```