Universidade Federal do Paraná
Departamento de Informática
CI-202

MÉTODOS NUMÉRICOS

Prof. Ionildo José Sanches
Prof. Diógenes Cogo Furlan

E-Mail: ionildo@ionildo.cjb.net
URL: http://www.ionildo.cjb.net/metodos/

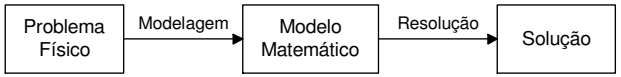
CURITIBA
2007

SUMÁRIO

1 Introdução

Cálculo Numérico é a obtenção da solução de um problema pela aplicação de método numérico; a solução do problema será caracterizada, então, por um conjunto de números, exatos ou aproximados.

Método Numérico é um algoritmo composto por um número finito de operações envolvendo apenas números (operações aritméticas elementares, cálculo de funções, consulta a uma tabela de valores, consulta a um gráfico, arbitramento de um valor, etc.).



Modelagem é a fase de obtenção do modelo matemático que descreve o comportamento do sistema físico.

Resolução é a fase de obtenção da solução através da aplicação de métodos numéricos (este é o objetivo de estudo do **Cálculo Numérico**).

2 Conceito de Erro

2.1 Introdução

A noção de erro está presente em todos os campos do Cálculo Numérico. De um lado, os dados, em si, nem sempre são exatos e, de outro lado, as operações sobre valores não exatos propagam esses erros a seus resultados. Finalmente, os próprios métodos numéricos, freqüentemente métodos aproximados, buscam a minimização dos erros, procurando resultados o mais próximo possível do que seriam valores exatos.

Erro é a diferença entre o valor exato e o valor apresentado.

No próximo capítulo, sobre representação de números reais, iremos analisar várias situações em que ocorrem erros, quando utilizamos o computador para realizar os cálculos. A seguir, analisaremos os erros que ocorrem durante as fases de modelagem e resolução e também sobre erros de arredondamento e erros de truncamento.

2.2 Erros na Fase de Modelagem

Ao se tentar representar um fenômeno do mundo físico por meio de um método matemático, raramente se tem uma descrição correta deste fenômeno. Normalmente, são necessárias várias simplificações do mundo físico para que se tenha um modelo.

Exemplo: Estudo do movimento de um corpo sujeito a uma aceleração constante.

Tem-se a seguinte equação:

$$d = d_o + v_o * t + 1/2 * \alpha * t^2$$

onde:

- d : distância percorrida
- d_o : distância inicial
- v_o : velocidade inicial
- t : tempo
- α : aceleração

Determinar a altura de um edifício com uma bolinha de metal e um cronômetro: 3s

$$d = 0 + 0 * 3 + 1/2 * 9.8 * 3^2 = 44.1\text{m}$$

Este resultado é confiável?

1. Fatores não considerados:
   - resistência do ar
   - velocidade do vento, etc.
2. Precisão dos dados de entrada:
   - Se o tempo fosse 3,5s $\rightarrow$ d = 60.025m
   - Variação de 16,7% no cronômetro $\rightarrow$ 36% na altura.

2.3 Erros na Fase de Resolução

Para a resolução de modelos matemáticos muitas vezes torna-se necessária a utilização de instrumentos de cálculo que necessitam, para o seu funcionamento, que sejam feitas certas aproximações. Tais aproximações podem gerar erros, tais como: conversão de bases, erros de arredondamento e erros de truncamento.

2.4 Erros Absolutos e Relativos

Erro absoluto (*EA*) é a diferença entre o valor exato de um número *N* e o seu valor aproximado *N'*:

$N = N' + EA_N$ $\quad (N > N' \rightarrow EA_N > 0$
$N < N' \rightarrow EA_N < 0)$

$EA_N = N - N'$ ***Erro absoluto***

Por exemplo, sabendo-se que $\pi \in (3.14, 3.15)$ tomaremos para π um valor dentro deste intervalo e teremos, então, $|EA_\pi| = |\pi - \pi'| < 0.01$.

Erro Relativo é definido como o erro absoluto dividido pelo valor aproximado:

$$ER_N = \frac{EA_N}{N'} = \frac{N - N'}{N'} \qquad \textit{Erro Relativo}$$

É claro que EA_N só poderá ser determinado se N for exatamente conhecido; como isso é raro, em cálculos numéricos costuma-se trabalhar com uma limitação máxima para o erro, ao invés do próprio (indicando-se, então, $|E| < \varepsilon$, onde ε é o limite).

Por exemplo, se $\alpha = 3876.373$ e só desejamos a parte inteira α', o erro absoluto será:

$$\Delta_\alpha = |\alpha - \alpha'| = 0.373$$

Se fizermos o mesmo com o número $\beta = 1.373$, teremos:

$$\Delta_\beta = |\beta - \beta'| = 0.373$$

Obviamente, o efeito de aproximação de β é muito maior do que em α, mas o erro absoluto é o mesmo nos dois casos. O erro relativo, entretanto, pode traduzir perfeitamente este fato, pois:

$$\delta_\alpha = \frac{0{,}373}{3876} \cong 0{,}000096 < 10^{-4} \qquad \delta_\beta = \frac{0{,}373}{1} \cong 0{,}373 < 5*10^{0}$$

2.5 Erro de Arredondamento

Ao se aplicar um método numérico, os erros devidos aos valores iniciais, intermediários e finais conduzem a um erro global (diferença entre o exato e o obtido) também chamado de arredondamento.

Erros iniciais são os cometidos no arredondamento dos dados iniciais. Os erros intermediários são decorrentes dos erros cometidos durante a aplicação do método numérico e os erros finais decorrentes da apresentação final do resultado.

Os tipos de arredondamentos mais conhecidos são:

- Arredondamento para baixo ou por falta;
- Arredondamento para cima ou por excesso;
- Arredondamento para o numero de maquina mais próximo.

Critério de Arredondamento: no cálculo manual, ao registrar um valor aproximado, costuma-se usar a seguinte regra:

1. somar meia unidade após a última casa decimal a conservar;
2. desprezar as demais casas.

Assim, com 2 números significativos tem-se:

$\sqrt{2} = 1.414 ... \cong 1.41 \qquad (1.414 ... + 0.005 = 1.419 ... \rightarrow 1.41)$
$\sqrt[3]{2} = 1.259 ... \cong 1.26 \qquad (1.259 ... + 0.005 = 1.264 ... \rightarrow 1.26)$

O uso deste critério limita o erro a meia unidade da última casa conservada:

$E = \sqrt{2} - 1.41 = 1.41421 ... - 1.41 = 0.00421 < 0.005$

Os valores aproximados obtidos podem ser inferiores (valor aproximado por falta) ou superiores (valor aproximado por excesso) aos exatos; 1.41 é o valor aproximado, por falta, de $\sqrt{2}$; 1.26 é o valor de $\sqrt[3]{2}$, aproximado por excesso.

Para concluir este item de erro de arredondamento, deve-se ressaltar a importância de se saber o número de dígitos significativos do sistema de representação da máquina que está sendo utilizada para que se tenha a noção da precisão do resultado obtido.

Além da precisão decimal, o cálculo do chamado Épsilon da máquina nos dá uma idéia da exatidão da máquina.

O ε da máquina é o menor número de ponto flutuante, tal que: $1 + \varepsilon > 1$. Alguns métodos para cálculo de ε não dão seu valor exato, mas isto nem sempre é necessário, pois o que importa é a sua ordem de grandeza.

O programa abaixo, escrito na linguagem C, calcula uma aproximação do ε da máquina:

```
#include <stdio.h>
int main()
{
  float Eps=1.0;
  while (Eps + 1 > 1)
    Eps = Eps / 2.0;
  printf("A maquina acha que %1.25f vale zero!\n", Eps);
  return 0;
}
```

O programa acima, executado num Pentium, obteve a seguinte resposta:

```
A maquina acha que 0.0000000000000000000542101 vale zero!
```

Logo, o número de dígitos significativos é 19.

2.6 Erro de Truncamento

São erros provenientes da utilização de processos que deveriam ser infinitos ou muito grandes para a determinação de um valor e que, por razões práticas, são truncados.

Estes processos infinitos são muito utilizados na avaliação de funções matemáticas, tais como, exponenciação, logaritmos, funções trigonométricas e várias outras que uma máquina pode ter.

Exemplo: Uma máquina poderia calcular a função seno(x) e exponencial(x) utilizando as seguintes técnicas:

$$seno(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$$

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$$

Fazendo truncamento:

$$seno(x) \cong x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots + (-1)^n \frac{x^n}{n!}$$

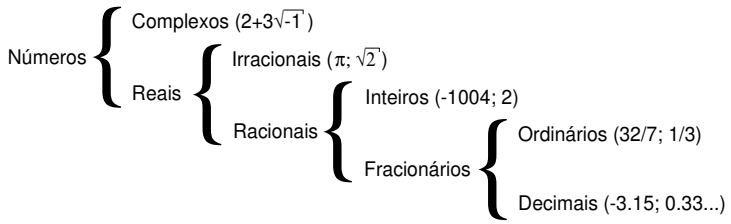
$$e^x \cong 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots + \frac{x^n}{n!}$$

A solução é a de interromper os cálculos quando uma determinada precisão é atingida.

De uma maneira geral, pode-se dizer que o erro de truncamento pode ser diminuído até chegar a ficar da ordem do erro de arredondamento; a partir desse ponto, não faz sentido diminuir-se mais, pois o erro de arredondamento será dominante.

3 Representação dos Números Reais

3.1 Introdução



Algumas das propriedades básicas da aritmética real não valem mais quando executadas no computador, pois, enquanto na matemática alguns números são representados por infinitos dígitos, no computador isso não é possível, pois uma palavra de memória é finita e a própria memória também.

Exemplos: $\sqrt{2}$, $\sqrt{3}$, π e $\frac{1}{3}$.

Se desejássemos calcular a área de uma circunferência de raio 100m, obteríamos os seguintes resultados:

a) $A = 31400\text{m}^2$
b) $A = 31416\ \text{m}^2$
c) $A = 31415.92654\ \text{m}^2$

Como justificar as diferenças entre os resultados? É possível obter o valor exato desta área?

Os erros ocorridos dependem da representação dos números na máquina utilizada. A representação de um número depende da base escolhida ou disponível na máquina em uso e do número máximo de dígitos usados na sua representação.

O número π, por exemplo, não pode ser representado através de um número finito de dígitos decimais. No exemplo mostrado acima, o número π foi escrito como 3.14, 3.1416 e 3.141592654 respectivamente nos casos (a), (b) e (c). Em cada um deles foi obtido um resultado diferente, e o erro neste caso depende exclusivamente da aproximação escolhida para π, Qualquer que seja a circunferência, a sua área nunca será obtida exatamente, uma vez que π é um número irracional.

Como neste exemplo, qualquer cálculo que envolva números que não podem ser representados através de um número finito de dígitos não fornecerá como resultado um valor exato. Quanto maior o número de dígitos utilizados, maior será a precisão obtida. Por isso, a melhor aproximação para o valor da área da circunferência é aquela obtida no caso (c).

Além disso, um número pode ter representação finita em uma base e não-finita em outras bases. A base decimal é a que mais empregamos atualmente. Um computador opera normalmente no sistema binário.

Observe o que acontece na interação entre o usuário (ou dados do programa) e o computador: os dados de entrada são enviados ao computador pelo usuário no sistema decimal; toda esta informação é convertida para o sistema binário, e as operações todas serão efetuadas neste sistema. Os resultados finais

serão convertidos para o sistema decimal e, finalmente, serão transmitidos ao usuário. Todo este processo de conversão é uma fonte de erros que afetam o resultado final dos cálculos.

Na próxima seção, iremos estudar os processos de conversão de números do sistema decimal para o sistema binário e vice-versa. Estudaremos também a forma de armazenamento feita pelos computadores digitais.

3.2 Sistema de Numeração

Existem vários sistemas numéricos, dentre os quais destacam-se o sistema decimal (base 10), o octal (base 8) e o hexadecimal (base 16).

Em um sistema numérico com base β, existem β dígitos e o maior é $\beta - 1$. De um modo geral, um número na base β, $(a_j a_{j-1}...a_2 a_1 a_0)_\beta$, $0 \le a_k \le (\beta - 1)$, $k = 1, 2, ..., j$, pode ser escrito na forma polinomial:

$$a_j\beta^j + a_{j-1}\beta^{j-1} + ... + a_2\beta^2 + a_1\beta^1 + a_0\beta^0$$

Com esta representação, podemos facilmente converter um número representado em qualquer sistema para o sistema decimal.

3.2.1 Sistema de Numeração Decimal

No sistema de numeração usual, o sistema decimal, usamos dez dígitos 0, 1, ..., 9. Um número maior que 9 é representado usando uma convenção que atribui significado à posição ou lugar ocupado por um dígito. Por exemplo, em virtude das posições ocupadas pelos dígitos individuais no número 2015, este número tem significado numérico calculado como:

$$2015 = 2*10^3 + 0*10^2 + 1*10^1 + 5*10^0 = 2000 + 0 + 10 + 5 = 2015$$

Notamos que um número é expresso como uma soma de potências de 10 multiplicadas por coeficientes apropriados. No sistema decimal, 10 é a base do sistema. Existem 10 dígitos, o maior sendo 9. Em um sistema numérico com base β, existem β dígitos e o maior é β -1.

3.2.2 Sistema de Numeração Binário

No sistema binário existem apenas 2 dígitos: **0** e **1**. Como os circuitos eletrônicos usados no computador apresentam 2 estados possíveis, convencionou-se chamar o estado desligado de 0 e o estado ligado de 1. Cada dígito de um número representado no sistema binário é denominado ***bit*** (contração de ***BI****nary digi****T***), o conjunto de 4 *bits* é denominado *nibble* e o de 8 *bits* de *byte*, termo bastante utilizado na área de informática.

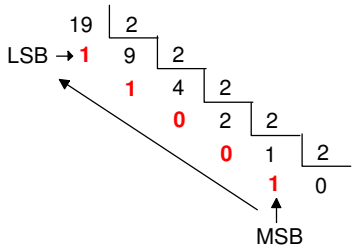
3.2.2.1 Conversão do Sistema Binário para Decimal

Quando um número é escrito no sistema binário, os dígitos individuais representam os coeficientes de potências de 2. Por exemplo, o número decimal 19 é escrito em representação binária como 10011, pois este arranjo de dígitos binários significa:

$$10011 = 1*2^4 + 0*2^3 + 0*2^2 + 1*2^1 + 1*2^0 = 16 + 0 + 0 + 2 + 1 = 19$$

3.2.2.2 Conversão do Sistema Decimal para Binário

A conversão de um número decimal para binário é feita da seguinte forma:



$$19_{(10)} = 10011_{(2)}$$

O *bit* menos significativo de um número binário recebe a notação de LSB (*Least Significant Bit*) e o *bit* mais significativo de MSB (*Most Significant Bit*).

3.2.2.3 Conversão de Números Binários Fracionários em Decimais

Consideremos agora a conversão de um número fracionário binário (base 2) para um número decimal (base 10).

$$0.125_{10} = 0*10^0 + 1*10^{-1} + 2*10^{-2} + 5*10^{-3} = 0.1 + 0.02 + 0.005 = 0.125_{10}$$

$$0.001_2 = 0*2^0 + 0*2^{-1} + 0*2^{-2} + 1*2^{-3} = 0 + 0 + 0 + 0.125 = 0.125_{10}$$

$$0.101_2 = 0*2^0 + 1*2^{-1} + 0*2^{-2} + 1*2^{-3} = 0 + 0.5 + 0 + 0.125 = 0.625_{10}$$

3.2.2.4 Conversão de Números Decimais Fracionários em Binários

Consideremos agora a conversão de um número fracionário da base 10 para a base 2. Um número real entre 0 e 1 pode ter representação finita no sistema decimal, mas representação infinita no sistema binário.

No caso geral, seja r um número entre 0 e 1 no sistema decimal e $(0.d_1d_2...d_j...)_2$ sua representação no sistema binário. Os dígitos binários $d_1, d_2, ..., d_j, ...$ são obtidos através do seguinte algoritmo:

Passo 0: $r_1 = r$; $k = 1$

Passo 1: Calcule $2r_k$.
Se $2r_k = 1$, faça: $d_k = 1$,
caso contrário, faça: $d_k = 0$

Passo 2: Faça $r_{k+1} = 2r_k - d_k$
Se $r_{k+1} = 0$, pare.
Caso contrário:

Passo 3: $k = k + 1$.
Volte ao passo 1.

Observar que o algoritmo pode ou não terminar após um número finito de passos. Para $r = (0.125)_{10}$ teremos: $r_1 = 0.125$.

$k = 1$ $\quad 2r_1 = 0.25 \quad \Rightarrow \quad d_1 = \underline{\mathbf{0}}$
$\qquad\qquad\qquad\qquad\qquad\quad r_2 = 0.25 - d_1 = 0.25$

$k = 2$ $\quad 2r_2 = 0.5 \quad \Rightarrow \quad d_2 = \underline{\mathbf{0}}$

$r_3 = 0.5$

$k = 3 \quad 2r_3 = 1.0 \quad \Rightarrow \quad d_3 = \underline{\mathbf{1}}$
$r_4 = 0$

Temos então $0.125_{10} = 0.001_2$, sendo portanto a representação binária finita. Já para $r = 0.1_{10}$, teremos: $r_1 = 0.1$

$k = 1 \quad 2r_1 = 0.2 \quad \Rightarrow \quad d_1 = \underline{\mathbf{0}}$
$r_2 = 0.2$

$k = 2 \quad 2r_2 = 0.4 \quad \Rightarrow \quad d_2 = \underline{\mathbf{0}}$
$r_3 = 0.4$

$k = 3 \quad 2r_3 = 0.8 \quad \Rightarrow \quad d_3 = \underline{\mathbf{0}}$
$r_4 = 0.8$

$k = 4 \quad 2r_4 = 1.6 \quad \Rightarrow \quad d_4 = \underline{\mathbf{1}}$
$r_5 = 0.6$

$k = 5 \quad 2r_5 = 1.2 \quad \Rightarrow \quad d_5 = \underline{\mathbf{1}}$
$r_6 = 0.2 = r_2$

Como $r_6 = r_2$, teremos que os resultados para k de 2 e 5 se repetirão e então: $r_{10} = r_6 = r_2 = 0.2$ e assim indefinidamente.

Concluímos que: $(0.1)_{10} = (0.00011001100111\mathbf{0011...})_2$ e, portanto, o número $(0.1)_{10}$ não tem representação binária finita.

O fato de um número não ter representação finita no sistema binário pode acarretar a ocorrência de erros aparentemente inexplicáveis em cálculos efetuados em sistemas computacionais binários.

Um computador que opera no sistema binário irá armazenar uma aproximação para $(0.1)_{10}$, uma vez que possui uma quantidade fixa de posições para guardar os dígitos de mantissa de um número, e esta aproximação será usada para realizar os cálculos. Não se pode, portanto, esperar um resultado exato.

Podemos agora entender melhor por que o resultado da operação:

$$S = \sum_{n=1}^{1000} 0.1$$

não é obtido corretamente num computador. Supondo uma máquina digital que trabalhe com apenas 9 dígitos na mantissa, o número $(0.1)_{10}$ seria armazenado como $(0.000110011)_2$ e este número representa exatamente $(0.099609375)_{10}$. Portanto, todas as operações que envolvem $(0.1)_{10}$ seriam realizadas com esta aproximação. Veremos na próxima seção a representação de números em aritmética de ponto flutuante com o objetivo de se entender melhor a causa de resultados imprecisos em operações numéricas.

O programa a seguir permite calcular $\sum_{i=1}^{1000} 0.1$, sendo 100 o valor exato dessa somatória.

```
#include <stdio.h>
int main()
{
  int i;
  float x=0;
  for (i=1;i<=1000; i++)
     x = x + 0.1;
  printf("x = %.20f", x);
  return 0;
}
```


Quando essa somatória é efetuada utilizando o computador o valor é: **99.990046325683593800000**. Se trocarmos o tipo *float* para *double* (maior precisão) o resultado será **99.99999999998593100000**.

3.3 Aritmética de Ponto Flutuante

Usa-se, rotineiramente, duas formas para fazer o armazenamento dos números em máquinas: ponto fixo (para valores inteiros) e ponto flutuante (para valores reais).

Uma máquina de calcular, ou um computador digital, representa um número real no sistema denominado aritmética de *ponto flutuante*. Neste sistema, o número x é representado na forma:

$$x = \pm \left[\frac{d_1}{\beta^1} + \frac{d_2}{\beta^2} + \frac{d_3}{\beta^3} + \ldots + \frac{d_t}{\beta^t} \right] \bullet \ \beta^e$$

onde:

β: é a base em que a máquina opera;
d_i: são números inteiros contidos no intervalo $0 \le d_i \le (\beta - 1)$; $i = 1, 2, ..., t$; $d_1 \neq 0$;
e: representa o expoente de β e assume valores entre $I \le e \le S$;
I, S: limite inferior e limite superior, respectivamente, para a variação do expoente.

$\left[\frac{d_1}{\beta^1} + \frac{d_2}{\beta^2} + \frac{d_3}{\beta^3} + \ldots + \frac{d_t}{\beta^t} \right]$ é chamada de mantissa e é a parte do número que representa os seus dígitos significativos e t é o número de dígitos significativos do sistema de representação, comumente chamado de precisão da máquina.

Exemplo 1: No sistema de base $\beta = 10$ (decimal), tem-se:

$$0.125_{10} = \left(\frac{1}{10^1} + \frac{2}{10^2} + \frac{5}{10^3} \right) \bullet \ 10^0$$

$$3.1415_{10} = 0.31415 \bullet 10^1 = \left(\frac{3}{10^1} + \frac{1}{10^2} + \frac{4}{10^3} + \frac{1}{10^4} + \frac{5}{10^5} \right) \bullet \ 10^1$$

Os números assim representados estão normalizados, isto é, a mantissa é um valor entre 0 e 1. A forma normalizada é utilizada nas operações envolvendo ponto flutuante em computadores digitais.

No sistema de base $\beta = 2$ (binário), tem-se:

$$10_{10} = 1010_2 = 0.101 \bullet 2^4 = \left(\frac{1}{2^1} + \frac{0}{2^2} + \frac{1}{2^3} \right) \bullet \ 2^4$$

$$4_{10} = 100_2 = 0.1 * 2^3 = \frac{1}{2} * 2^3$$

Exemplo 2: Numa máquina de calcular cujo sistema de representação utilizado tenha β=2, $t = 10$, $I = -15$ e $S = 15$, o número 25_{10} e 3.5_{10} é, assim representado:

$25_{10} = 11001_2 = 0.11001 * 2^5 = 0.11001 * 2^{101}$

$$\left(\frac{1}{2^1} + \frac{1}{2^2} + \frac{0}{2^3} + \frac{0}{2^4} + \frac{1}{2^5} + \frac{0}{2^6} + \frac{0}{2^7} + \frac{0}{2^8} + \frac{0}{2^9} + \frac{0}{2^{10}}\right) * 2^{101}$$

ou, de uma forma mais compacta:

1100100000	0101
Mantissa	expoente

Cada dígito é chamado de bit, portanto, nesta máquina são utilizados 10 bits para a mantissa, 4 para o expoente e mais um bit para o sinal da mantissa (se bit=0 positivo, se bit=1 negativo) e um bit para o sinal do expoente, resultando, no total, 16 bits, que são assim representados:

$25_{10} =$

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

Valor da Mantissa — Sinal da Mantissa — Expoente — Sinal Exp.

$3.5_{10} = 0.111 * 2^{10}$

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

O maior valor representado por esta máquina descrita no exemplo 2 seria:

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

que, na base decimal, tem o seguinte valor: $0.1111111111 * 2^{1111} = 32736_{10}$

E o menor valor seria: $-0.1111111111 * 2^{1111} = -32736_{10}$

Logo, os números que podem ser representados nesta máquina estariam contidos no intervalo [−32736; 32736].

Nesta máquina, ainda, o valor zero seria representado por:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

O próximo número positivo representado seria:

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

$0.1 * 2^{-15} = 0.000015259$

O subsequente seria:

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

$0.1000000001 * 2^{-15} = 0.000015289$

Através desses exemplos pode-se concluir que o conjunto dos números representáveis neste sistema é um subconjunto dos números reais, dentro do intervalo mostrado anteriormente.

Considere, por exemplo, uma máquina que opera no sistema:

$$\beta = 10; \qquad t = 3; \qquad e \in [-5,5].$$

Os números serão representados da seguinte forma nesse sistema:

$$0.d_1d_2d_3 * 10^e,\ 0 \le d_j \le 9,\ d_1 \ne 0,\ e \in [-5, 5]$$

O menor número (m), em valor absoluto, representado nesta máquina é:

$m = 0.100 * 10^{-5} = 10^{-6}$

e o maior número (M), em valor absoluto, é:

$M = 0.999 * 10^5 = 99900$

Considere o conjunto dos números reais $\mathbb{R}$ e o seguinte conjunto:

$G = \{x \in \mathbb{R} \mid m \le |x| \le M\}$

Dado um número real x, várias situações poderão ocorrer:

1. $x \in G$: por exemplo, $x = 235.89 = 0.23589 * 10^3$. Observe que este número possui 5 dígitos na mantissa. Estão representados exatamente nesta máquina os números: $0.235*10^3$ e $0.236*10^3$. Se for usado o truncamento, x será representado por $0.235*10^3$ e, se for usado o arredondamento, x será representado por $0.236*10^3$. Na próxima seção, sobre erros, estudaremos o truncamento e o arredondamento;
2. $|x| < m$: por exemplo, $x = 0.345*10^{-7}$. Este número não pode ser representado nesta máquina porque o expoente e é menor que −5. Esta é uma situação em que a máquina acusa a ocorrência de *underflow*;
3. $|x| > M$: por exemplo, $x = 0.875*10^9$. Neste caso o expoente e é maior que 5 e a máquina acusa a ocorrência de *overflow*.

Algumas linguagens de programação permitem que as variáveis sejam declaradas em precisão dupla. Neste caso, esta variável será representada no sistema de aritmética de ponto flutuante da máquina, mas com aproximadamente o dobro de dígitos disponíveis na mantissa. É importante observar que, neste caso, o tempo de execução e requerimento de memória aumentam de forma significativa.

O C fornece três tipos para números de ponto flutuante. Cada tipo tem um intervalo e uma precisão especificada:

Tipo	Nº de bits	Intervalo	
		Inicio	Fim
`float`	32	`3.4E-38`	`3.4E+38`
`double`	64	`1.7E-308`	`1.7E+308`
`long double`	80	`3.4E-4932`	`3.4E+4932`

O tipo long double é o tipo de ponto flutuante com maior precisão. É importante observar que os intervalos de ponto flutuante, na tabela acima, estão indicados em faixa de expoente, mas os números podem assumir valores tanto positivos quanto negativos.

3.4 Propagação de Erros

Durante as operações aritméticas de um método, os erros dos operandos produzem um erro no

resultado da operação; sendo A, a, B, b os valores exatos e aproximados, respectivos, e E_a e E_b, os erros dos operandos.

$A + B = (a + E_a) + (b + E_b) = a + b + E_a + E_b \quad \therefore EA_{A+B} = E_a + E_b$
$A - B = (a + E_a) - (b + E_b) = a - b + E_a - E_b \quad \therefore EA_{A-B} = E_a - E_b$
$A * B = (a + E_a)\,(b + E_b) = ab + aE_b + bE_a + E_b{*}E_a \quad \therefore EA_{A*B} = aE_b + bE_a + E_b{*}E_a$

Vejamos através de um exemplo, como os erros descritos anteriormente podem influenciar o desenvolvimento de um cálculo.

Exemplo: Suponha-se que as operações abaixo sejam processadas em uma máquina com 4 dígitos significativos e fazendo-se: $x_1 = 0.3491*10^4$ e $x_2 = 0.2345*10^0$ tem-se:

$$\begin{aligned}(x_2 + x_1) - x_1 &= (0.2345*10^0 + 0.3491*10^4) - 0.3491*10^4 \\ &= 0.3491*10^4 - 0.3491*10^4 \\ &= 0.0000\end{aligned}$$

$$\begin{aligned}x_2 + (x_1 - x_1) &= 0.2345*10^0 + (0.3491*10^4 - 0.3491*10^4) \\ &= 0.2345 + 0.0000 \\ &= 0.2345\end{aligned}$$

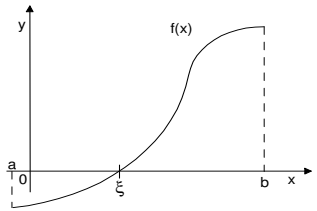
Os dois resultados são diferentes, quando não deveriam ser, pois a adição é uma operação distributiva. A causa desta diferença foi um arredondamento feito na adição ($x_2 + x_1$), cujo resultado tem 8 dígitos. Como a máquina só armazena 4 dígitos, os menos significativos foram desprezados.

Ao se utilizar máquinas de calcular deve-se estar atento a essas particularidades causadas pelo erro de arredondamento, não só na adição mas também nas outras operações.

4 Zeros de Equações Transcendentes e Polinomiais

4.1 Introdução

Seja $F(x)$ uma função real definida num intervalo [a, b]. Chama-se raiz(es) desta função em [a, b] a todo ξ (*csi*) $\in$ (a, b) tal que $F(\xi) = 0$, como mostra a figura abaixo.



4.1.1 Derivada de uma função num ponto

A função $f : A \to \mathbf{R}$ diz-se derivável no ponto de acumulação $a \in A$ quando existe e é finito o limite:

$$\lim_{\Delta x \to 0} \frac{\Delta y}{\Delta x} = \lim_{x \to a} \frac{f(x) - f(a)}{x - a} = \lim_{\Delta x \to 0} \frac{f(a + \Delta x) - f(a)}{\Delta x}$$

Quando f é derivável em a, o limite é chamado derivada de f no ponto a.

4.1.2 Tipos de Métodos

Pode-se dizer que são dois os métodos para se achar a(s) raiz(es) de uma equação:

Método direto: quando fornece solução em apenas um único passo. Esta raiz é exata, a menos de erros de arredondamento.

Exemplo: Seja $F(x) = x^2 - 3x + 2$. A solução direta pode ser obtida através da fórmula de Baskara com a expressão: $X = \dfrac{-b \pm \sqrt{b^2 - 4ac}}{2a}$, que terá como conjunto solução {1, 2}.

```
PROGRAM Baskara;
VAR   a, b, c : INTEGER;
      delta : INTEGER;
      x1, x2 : REAL;
BEGIN
      a := 1; b := -3; c := 2;            {f(x) = x^2 - 3*x + 2}
      delta := b*b - 4*a*c;
      IF delta >= 0 THEN
      BEGIN
            x1 := (-b + SQRT(delta)) / (2*a);
            x2 := (-b - SQRT(delta)) / (2*a);
            WRITELN('x1 = ',x1);
            WRITELN('x2 = ',x2);
      END
      ELSE WRITELN('Nao possui raizes reais');
END.
```

***Método iterativo* ou indireto**: é um processo de cálculo infinito, recursivo, em que o valor obtido a cada passo depende de valores obtidos em passos anteriores. Este tipo de método, na maioria das vezes, não obtém solução exata para as raízes, mas sim uma solução aproximada dentro de uma faixa de erro considerada aceitável.

É importante salientar, que normalmente, os métodos iterativos são mais precisos quando executados em um computador que permite agilizar os cálculos matemáticos, obtendo assim uma melhor precisão.

Exercício: Calcular $\sqrt{4}$ e de $\sqrt{2}$ usando o Método de Newton definido por:

$$x_n = \frac{\left(\frac{x}{x_{n-1}} + x_{n-1}\right)}{2}, \text{ para } n = 1, 2, 3, ...$$

onde: x: o número a ser calculado a raiz
x_0: uma atribuição inicial qualquer diferente de zero (por exemplo, $x_0 = 1$).

Como vimos anteriormente, o cálculo das duas raízes de uma equação do segundo grau, colocada sob a forma $ax^2 + bx + c = 0$, são facilmente obtidas pela fórmula de Baskara. Entretanto, se colocarmos uma expressão em que apareça uma equação transcendente, a solução já não é tão simples, como demonstram os exemplos abaixo:

$e^x + x = 0$
$cos(x) - x = 0$
$ln(x) + x - 2 = 0$

Mesmo um polinômio de grau maior que três já não tem uma solução algébrica simples como a da equação do segundo grau, a não ser em casos particulares. Vamos analisar como enfrentar esse problema, tão comum em diversas áreas da engenharia, da economia, das ciências, da física, entre tantas outras.

Essas equações, com enorme freqüência, nos levam a raízes reais não racionais que, ao serem representadas no computador, necessariamente, o serão de forma aproximada, pelas razões já expostas no capítulo anterior, tendo em vista que necessitariam de infinitos dígitos, em suas mantissas, para serem representadas.

Além disso, em geral, estamos interessados em obter esses valores, essas raízes, com uma determinada precisão, com um erro tolerável, com algumas casas decimais, sem a pretensão de obter valores exatos. Isso é mais do que suficiente, para a maioria dos problemas práticos encontrados.

Os métodos numéricos a serem apresentados, partindo de valores inicialmente propostos, buscam aprimorar esses valores, diminuindo os erros, aproximando-se, assim, dos valores das raízes procuradas, até que os erros sejam aceitáveis, podendo-se garantir que sejam erros inferiores a valores pré-definidos.

Para se calcular uma raiz duas etapas devem ser seguidas:

- Isolar a raiz, ou seja, achar um intervalo [a, b], o menor possível, que contenha uma e somente uma raiz da equação $f(x) = 0$;
- Melhorar o valor da raiz aproximada, isto é, refiná-la até o grau de exatidão requerido. Com a abordagem iterativa precisamos determinar um intervalo inicial para construirmos a seqüência $\{x_i\}$ e teremos que a raiz x' será dada por:

$$x' = \lim_{i \to \infty} x_i$$

Além disto, temos que estipular critérios de parada, pois na pratica não calcularemos infinitos termos, mas apenas o suficiente para atingirmos a exatidão desejada.

4.1.3 Isolamento de Raízes

Nesta fase é feita uma análise teórica e gráfica da função $f(x)$. Para tal fim, usa-se freqüentemente um importante teorema da álgebra.

***Teorema*:** Se uma função $f(x)$ contínua num intervalo [a, b] assume valores de sinais opostos nos pontos extremos deste intervalo, isto é, $f(a).f(b) < 0$, então o intervalo conterá, no mínimo, uma raiz da equação $f(x) = 0$; em outras palavras haverá, no mínimo, um número $\xi \in$ (a, b) tal que $f(\xi) = 0$.

4.1.3.1 Número de Raízes Reais

Na seção anterior vimos como delimitar as raízes reais de $F(x) = 0$. Agora iremos verificar quantas raízes existem no intervalo delimitado. Os métodos a seguir dão uma boa indicação sobre o número de raízes do intervalo.

***Teorema de Bolzano*:** Seja $F(x) = 0$ uma equação algébrica com coeficientes reais e $x \in$ (a, b):

- Se $F(a).F(b) < 0$, então existe um número **impar** de raízes reais (contando suas multiplicidades) no intervalo (a, b).

- Se $F(a).F(b) > 0$, então existe um número **par** de raízes reais (contando suas multiplicidades) ou **não existe** raízes reais no intervalo (a, b).

A determinação do número de raízes de equações transcendentes geralmente é quase impossível, pois algumas equações podem ter um número infinito de raízes.

Não faremos maiores considerações sobre este importante tópico, por não ser o objeto de estudo neste momento, e por merecer um trabalho a parte, devido a extensão de seu conteúdo. Entretanto, podemos salientar que o problema de isolar raízes constitui-se da *enumeração*, *localização* e *separação* das mesmas.

4.1.3.2 Refinamento

Depois de isolar a raiz no intervalo [a, b], passa-se a calculá-la através de métodos numéricos. Como veremos adiante, estes métodos devem fornecer uma seqüência $\{x_i\}$ de aproximação, cujo limite é a raiz exata ξ. Em cada aproximação x_n, da raiz exata ξ, usa-se um destes critérios e compara-se o resultado com a tolerância ε pré-fixada.

A verificação, de que x_n está "suficientemente" próxima da raiz, pode ser feita de dois modos diferentes (que podem levar a resultados diferentes):

$$| f(x_n) | \leq \varepsilon \qquad \text{(abordagem pelo eixo y)}$$

$$| x_n - x_{n-1} | \leq \varepsilon \text{ (abordagem pelo eixo x)}$$

Observa-se que dependendo dos números envolvidos é aconselhável usar os testes de erro relativo:

$$\frac{| x_n - x_{n-1} |}{| x_{n-1} |} \leq \varepsilon$$

4.1.4 Classificação dos métodos

<u>Métodos de quebra</u>: Os métodos de quebra são os mais intuitivos geometricamente; contudo, são os que convergem mais lentamente. Estes métodos são assim chamados porque a partir de um intervalo que contenha uma raiz da função, vai-se particionando este intervalo em outros menores, que ainda contenham a raiz. Dependendo da escolha do ponto de quebra do intervalo, poderemos ter diferentes métodos, tais como.

- Método da Bisseção;