

MAC 2166 – Introdução à Computação

POLI - PRIMEIRO SEMESTRE DE 2007

Material Didático

Prof. Ronaldo Fumio Hashimoto

FÓRMULA DE RECORRÊNCIA E SÉRIES (SOMAS INFINITAS)

Objetivo

O objetivo desta aula é introduzir fórmulas de recorrência e o uso das mesmas para o cálculo de séries (somadas infinitas).

Introdução

Até o momento nos limitamos ao uso do tipo inteiro para variáveis e expressões. Mesmo variáveis do tipo caractere guardam números inteiros. Vamos introduzir agora o tipo real.

Para entender como são representados variáveis do tipo real, vamos falar um pouco da representação de números inteiros.

Fórmula de Recorrência

Uma fórmula de recorrência é uma relação entre os termos sucessivos de uma seqüência numérica. Dessa forma, usando uma fórmula de recorrência, é possível obter o próximo termo da seqüência usando o valor de termos anteriores. Um exemplo clássico é a seqüência de Fibonacci definida pela fórmula de recorrência

$$\begin{cases} F_1 = 1 \\ F_2 = 1 \\ F_i = F_{i-1} + F_{i-2} \quad \text{para } i \geq 3. \end{cases}$$

Note que para determinar o termo F_i é necessário ter os dois termos anteriores F_{i-1} e F_{i-2} .

Exercício de Fórmula de Recorrência: Raiz Quadrada

Dados $x \geq 0$ e ϵ , $0 < \epsilon < 1$, números reais, calcular uma aproximação para raiz quadrada de x através da seqüência de números gerada pela seguinte fórmula de recorrência:

$$\begin{cases} r_0 = x \\ r_{k+1} = (r_k + x/r_k)/2 \quad \text{para } k > 0. \end{cases}$$

Gere a seqüência até um k tal que $|r_k - r_{k-1}| < eps$. A raiz quadrada de x é o último valor da seqüência, isto é, r_k .

Note que de uma certa maneira o real eps controla a precisão da raiz quadrada de x .

Solução:

Gerar os números da seqüência $r_0, r_1, r_2, \dots, r_k$ usando uma repetição.

```

1      float r, x, erro, eps;
2
3      r = x; erro = eps;
4      while (erro >= eps) {
5          r = (r + x / r) / 2;
6          printf ("r = %f\n", r);
7
8          /* atualiza erro */
9          ...
10     }
```

A repetição acima de fato imprime cada elemento r_k da seqüência. No entanto, para calcular o erro= $|r_k - r_{k-1}|$ é necessário guardar o termo anterior. Assim, vamos declarar mais uma variável r_{ant} e fazer a geração da seqüência da seguinte forma:

```

1      float r, rant, x, erro, eps;
2
3      r = x; erro = eps;
4      while (erro >= eps) {
5          r = (rant + x / rant) / 2;
6          printf ("r = %f\n", r);
7
8          /* atualiza erro */
9          erro = r - rant;
10
11         /* atualiza rant */
12         rant = r;
13     }
```

Agora, note que o cálculo do erro no trecho de programa acima está errado, uma vez que erro= $|r_k - r_{k-1}|$ (valor absoluto). Para consertar isso, temos que verificar se erro ficou negativo. Em caso afirmativo, devemos trocar o sinal de erro:

```

1     float r, rant, x, erro, eps;
2
3     r = x; erro = eps;
4     while (erro >= eps) {
5         r = (rant + x / rant) / 2;
6         printf ("r = %f\n", r);
7
8         /* atualiza erro */
9         erro = r - rant;
10        if (erro < 0)
11            erro = -erro;
12
13        /* atualiza rant */
14        rant = r;
15    }

```

Note ainda que devemos garantir que o programa funcione para $x=0$. Como a raiz quadrada de zero é zero, podemos fazer com que quando $x=0$, o programa não entre no laço colocando uma condição ($\text{erro} \geq \text{eps} \ \&\& \ x > 0$).

Assim, a solução final do exercício é:

```

1     # include <stdio.h>
2
3     int main () {
4         float r, rant, x, erro, eps;
5
6         printf ("Entre com x >= 0: ");
7         scanf ("%f", &x);
8
9         printf ("Entre com 0 < eps < 1: ");
10        scanf ("%f", &eps);
11
12        r = x; erro = eps;
13        while (erro >= eps && x > 0) {
14            r = (rant + x / rant) / 2;
15
16            /* atualiza erro */
17            erro = r - rant;
18            if (erro < 0)
19                erro = -erro;
20
21            /* atualiza rant */
22            rant = r;
23        }
24
25        printf ("Raiz de %f = %f\n", x, r);
26
27        return 0;
28    }

```

Erro Absoluto e Erro Relativo

Dado um número x e uma aproximação y para x , o erro (também chamado de erro absoluto) da aproximação y em relação x é definido como $|y - x|$. Quando a grandeza de x não é próxima da de 1, o erro absoluto pode não ser a maneira mais adequada de medir a qualidade da aproximação y . Por exemplo, os erros absolutos de 1.01 em relação a 1.00 e de 0.02 em relação a 0.01 são idênticos, mas é claro que a primeira aproximação é muito melhor que a segunda.

Face à limitada avaliação de uma aproximação conferida pelo erro absoluto, tenta-se definir o erro relativo a y em relação a x como sendo

$$|y - x/x|$$

Assim, nos dois exemplos anteriores, os erros relativos são respectivamente de 0.01 (ou 1%) e 1.00 (ou 100%). Contudo esta definição é incompleta quando $x = 0$. Neste caso, a divisão por 0 não pode ser realizada e adotam-se valores arbitrários para o erro relativo. No caso de também ocorrer que $y = 0$, a aproximação certamente é perfeita e adota-se que o erro é 0. No caso de $y \neq 0$, a aproximação é certamente insatisfatória e adota-se o valor arbitrário 1 para o erro relativo. Assim, definimos

$$errorel(y, x) = \begin{cases} |y - x/x| & \text{se } x \neq 0 \\ 0 & \text{se } x = 0 = y \\ 1 & \text{se } x = 0 \neq y \end{cases}$$

Exercício da Raiz Quadrada com Erro Relativo

Resolver o exercício da raiz quadrada usando erro relativo em vez de erro absoluto, ou seja, gerar a seqüência até um k tal que $errorel(r_k, r_{k-1}) < eps$.

Solução:

A única diferença deste exercício com relação ao anterior é o cálculo do erro. Esse cálculo pode ser feito da seguinte forma:

```
1      float r, rant, x, erro;
2
3      if (rant != 0) {
4          erro = (r - rant) / rant;
5          if (erro < 0)
6              erro = -erro;
7      }
8      else { /* rant == 0 */
9          if (r == 0)
10             erro = 0;
11         else
12             erro = 1;
13     }
```

Assim, a solução final do exercício é:

```

1      # include <stdio.h>
2
3      int main () {
4          float r, rant, x, erro;
5
6          printf ("Entre com x >= 0: ");
7          scanf ("%f", &x);
8
9          printf ("Entre com 0 < eps < 1: ");
10         scanf ("%f", &eps);
11
12         erro = r = x;
13         while (erro >= eps) {
14             r = (rant + x / rant) / 2;
15
16             /* atualiza erro */
17             if (rant != 0) {
18                 erro = (r - rant) / rant;
19                 if (erro < 0)
20                     erro = -erro;
21             }
22             else { /* rant == 0 */
23                 if (r == 0)
24                     erro = 0;
25                 else
26                     erro = 1;
27             }
28
29             /* atualiza rant */
30             rant = r;
31         }
32
33         printf ("Raiz de %f = %f\n", x, r);
34
35         return 0;
36     }

```

Exercício de Cálculo de Séries

Dados x e eps , $0 < eps < 1$, reais, obter uma aproximação da série

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

com precisão eps , isto é, somar os termos da série até aparecer um termo cujo valor absoluto seja menor que eps .

Primeiramente, vamos mostrar uma forma que você não deve usar para resolver este exercício.

- $\left| \frac{x^k}{k!} \right|$ tende a zero quando k tende a $+\infty$.
- Usando um comando de repetição, gerar uma seqüência de números $k = 1, 2, 3, 4, \dots$,
- Calcular $p = x^k$.
- Calcular $fat = k!$.
- Calcular $t = p/fat$.

- Acumular t em uma variável *soma*.
- Repetir estes cálculos até um k tal que $\left| \frac{x^k}{k!} \right| < \text{eps}$.

Esta solução poderia ser escrita como:

```

1      float soma, t, eps, x, pot, fat;
2      int k;
3
4      soma = 1; t = 1; k = 1;
5      while (|t| >= eps) {
6          /* calcule pot = x^k; */
7          /* calcule fat = k!; */
8          t = pot / fat;
9          soma = soma + t;
10         k++;
11     }

```

Para o cálculo de *pot* e *fat*, é possível aproveitar os valores de *pot* e *fat* anteriores da seguinte forma:

```

1      float soma, t, eps, x, pot, fat;
2      int k;
3
4      soma = t = k = 1;
5      pot = fat = 1;
6      while (|t| >= eps) {
7          /* calcule pot = x^k; */
8          pot = pot * x;
9          /* calcule fat = k!; */
10         fat = fat * k;
11         t = pot / fat;
12         soma = soma + t;
13         k++;
14     }

```

Esta solução é ruim, pois como não sabemos até que valor k vai assumir, a variável *fat* que recebe o fatorial de k , pode estourar facilmente, mesmo *fat* sendo uma variável do tipo `float`.

Assim a solução acima não é considerada uma boa solução. Uma boa solução não deve envolver o cálculo do fatorial.

A idéia é calcular um termo da série usando o termo anterior. Observe que

$$t_{k-1} = \frac{x^{k-1}}{(k-1)!}$$

O próximo termo t_k é

$$t_k = \frac{x^k}{k!} = \frac{x^{k-1} \times x}{(k-1)! \times k} = \frac{x^{k-1}}{(k-1)!} \times \frac{x}{k} = t_{k-1} \times \frac{x}{k}$$

Assim, para calcular o próximo termo da série, basta multiplicar o termo anterior pelo fator $\frac{x}{k}$.

Assim, uma melhor solução seria:

```

1      float soma, t, eps, x;
2      int k;
3
4      soma = t = k = 1;
5      while (|t| >= eps) {
6          t = t * x / k;
7          soma = soma + t;
8          k++;
9      }

```

Note que esta solução não envolve diretamente o cálculo de fatorial. A solução completa seria:

Solução:

```

1      # include <stdio.h>
2
3      int main () {
4
5          float soma, t, eps, x, abs_t;
6          int k;
7
8          printf ("Entre com x: ");
9          scanf ("%f", &x);
10
11         printf ("Entre com 0 < eps < 1: ");
12         scanf ("%f", &eps);
13
14         soma = abs_t = t = k = 1;
15
16         while (abs_t >= eps) {
17             t = t * x / k;
18             soma = soma + t;
19             k++;
20             abs_t = t;
21             if (abs_t < 0)
22                 abs_t = -abs_t;
23         }
24
25         printf ("exp(%f) = %f\n", x, soma);
26
27         return 0;
28     }

```

Note o if no final da repetição para calcular o módulo do termo t .

Outro Exercício de Cálculo de Séries

Dados x e ϵ reais, $\epsilon > 0$, calcular uma aproximação para $\text{sen } x$ através da seguinte série infinita

$$\text{sen } x = \frac{x}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \dots + (-1)^k \frac{x^{2k+1}}{(2k+1)!} + \dots$$

incluindo todos os termos até que $\frac{|x^{2k+1}|}{(2k+1)!} < \epsilon$.

Solução:

Neste exercício, temos que calcular o termo seguinte em função do termo anterior. Assim, o termo anterior é

$$t_{k-1} = (-1)^{k-1} \cdot \frac{x^{2(k-1)+1}}{(2(k-1)+1)!} = (-1)^{k-1} \cdot \frac{x^{2k-1}}{(2k-1)!}$$

$$\begin{aligned} t_k &= (-1)^k \cdot \frac{x^{2k+1}}{(2k+1)!} = (-1)^{k-1} \cdot (-1) \cdot \frac{x^{(2k-1)+2}}{(2k-1)! \cdot (2k) \cdot (2(k+1))} = \\ &= (-1)^{k-1} \cdot \frac{x^{2k-1}}{(2k-1)!} \cdot \frac{-x^2}{(2k) \cdot (2(k+1))} = \\ &= t_{k-1} \cdot \frac{-x^2}{(2k) \cdot (2(k+1))} \end{aligned}$$

Assim, neste exercício, podemos calcular o termo seguinte em função do termo anterior apenas multiplicando-o pelo fator $\frac{-x^2}{(2k) \cdot (2(k+1))}$. Um esboço de uma solução seria:

- Usando um comando de repetição, gerar uma seqüência de números $k = 1, 2, 3, 4, \dots$,
- Calcular $t_k = t_{k-1} \times (-x^2) / ((2k) \cdot (2(k+1)))$.
- Acumular t em uma variável *soma*.
- Repetir estes cálculos até um k tal que $|t_k| < \epsilon$.

Note um par de parênteses a mais no divisor do fator multiplicativo. Este par de parênteses é necessário para fazer a divisão corretamente. Uma solução completa seria:

Solução:

```

1      # include <stdio.h>
2
3      int main () {
4
5          float soma, t, eps, x, abs_t;
6          int k;
7
8          printf ("Entre com x: ");
9          scanf ("%f", &x);
10
11         printf ("Entre com 0 < eps < 1: ");
12         scanf ("%f", &eps);
13
14         soma = abs_t = t = x;
15         if (abs_t < 0) abs_t = -abs_t;
16         k = 1;
17
18         while (abs_t >= eps) {
19             t = - t * x * x / ((2*k)*(2*k+1));
20             soma = soma + t;
21             k++;
22             abs_t = t;
23             if (abs_t < 0)
24                 abs_t = -abs_t;
25         }
26
27         printf ("sen(%f) = %f\n", x, soma);
28
29         return 0;
30     }
```

Exercício

Dados x real e N natural, calcular uma aproximação para $\cos x$ através dos N primeiros termos da seguinte série:

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots + (-1)^k \frac{x^{2k}}{(2k)!} + \dots$$

Dúvidas

Dúvidas deste material pode ser enviadas para o “Fórum para assuntos específicos da turma WEB”.