14 Fórmula de Recorrência e Séries (Somas Infinitas)

Ronaldo F. Hashimoto e Carlos H. Morimoto

Nessa aula vamos introduzir fórmulas de recorrência e o uso das mesmas para o cálculo de séries (somas infinitas).

Ao final dessa aula você deverá saber:

- Descrever o que são fórmulas de recorrência.
- Descrever o que são erro absoluto e relativo.
- Escrever programas em C a partir de fórmulas de recorrência.

14.1 Fórmula de Recorrência

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

14.2 Exercício de Fórmula de Recorrência: Raiz Quadrada

Dados $x \geq 0$ e eps, $0 < eps < 1$, números reais, calcular uma aproximação para raiz quadrada de x através da sequência 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 sequência até um k tal que $|r_k - r_{k-1}| < eps$. A raiz quadrada de x é o último valor da sequência, 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 sequência $r_0, r_1, r_2, \ldots, r_k$ usando uma repetição.

```
    float r, x, erro, eps;

r = x; erro = eps;
while (erro >= eps) {
  r = (r + x / r) / 2;
  printf ("r = %f\n", r);

  /* atualiza erro */
  ...
}
```

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

```c
float r, rant, x, erro, eps;

r = x; erro = eps;
while (erro >= eps) {
  r = (rant + x / rant) / 2;
  printf ("r = %f\n", r);

  /* atualiza erro */
  erro = r - rant;

  /* atualiza rant */
  rant = r;
}
```

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`:

```c
float r, rant, x, erro, eps;

r = x; erro = eps;
while (erro >= eps) {
  r = (rant + x / rant) / 2;
  printf ("r = %f\n", r);

  /* atualiza erro */
  erro = r - rant;
  if (erro < 0)
    erro = -erro;

  /* atualiza rant */
  rant = r;
}
```

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 `(erro>= eps && x>0)`.

Assim, a solução final do exercício é:

```c
# include <stdio.h>

int main () {
  float r, rant, x, erro, eps;

  printf ("Entre com x >= 0: ");
  scanf ("%f", &x);

  printf ("Entre com 0 < eps < 1: ");
  scanf ("%f", &eps);

  r = x; erro = eps;
  while (erro >= eps && x>0) {
    r = (rant + x / rant) / 2;

    /* atualiza erro */
    erro = r - rant;
    if (erro < 0)
      erro = -erro;

    /* atualiza rant */
    rant = r;
  }

  printf ("Raiz de %f = %f\n", x, r);

  return 0;
}
```

14.3 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

$$|\frac{y-x}{x}|$$

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

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

14.4 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 sequência 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:

```c
float r, rant, x, erro;

if (rant != 0) {
  erro = (r - rant) / rant;
  if (erro < 0)
    erro = -erro;
}
else { /* rant == 0 */
  if (r == 0)
    erro = 0;
  else
    erro = 1;
}
```

Assim, a solução final do exercício é:

```c
# include <stdio.h>

int main () {
  float r, rant, x, erro;

  printf ("Entre com x >= 0: ");
  scanf ("%f", &x);

  printf ("Entre com 0 < eps < 1: ");
  scanf ("%f", &eps);

  erro = r = x;
  while (erro >= eps) {
    r = (rant + x / rant) / 2;

    /* atualiza erro */
    if (rant != 0) {
      erro = (r - rant) / rant;
      if (erro < 0)
        erro = -erro;
    }
    else { /* rant == 0 */
      if (r == 0)
        erro = 0;
      else
        erro = 1;
    }

    /* atualiza rant */
    rant = r;
  }

  printf ("Raiz de %f = %f\n", x, r);

  return 0;
}
```

14.5 Exercício de Cálculo de Séries

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

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

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 sequência de números $k = 1, 2, 3, 4, \ldots,$
- 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 $\left|\frac{x^k}{k!}\right| < eps$.

Esta solução poderia ser escrita como:

```
float soma, t, eps, x, pot, fat;
int k;

soma = 1; t = 1;  k = 1;
while (|t| >= eps) {
  /* calcule pot = x^k; */
  /* calcule fat = k!; */
  t = pot / fat;
  soma = soma + t;
  k++;
}
```

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

```
float soma, t, eps, x, pot, fat;
int k;

soma = t = k = 1;
pot = fat = 1;
while (|t| >= eps) {
  /* calcule pot = x^k; */
  pot = pot * x;
  /* calcule fat = k!; */
  fat = fat * k;
  t = pot / fat;
  soma = soma + t;
  k++;
}
```

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 anterio pelo fator $\frac{x}{k}$.

Assim, uma melhor solução seria:

```
float soma, t, eps, x;
int k;

soma = t = k = 1;
while (|t| >= eps) {
  t = t * x / k;
  soma = soma + t;
  k++;
}
```

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 (linha 21) para calcular o módulo do termo t.

14.6 Outro Exercício de Cálculo de Séries

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

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

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 $\dfrac{-x^2}{(2k) \cdot (2(k+1))}$. Um esboço de uma solução seria:

- Usando um comando de repetição, gerar uma sequência de números $k = 1, 2, 3, 4, \ldots,$
- Calcular $t_k = t_{k-1} \times (-x^2)/\left((2k) \cdot (2(k+1))\right)$.
- 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:

```
# include <stdio.h>

int main () {

  float soma, t, eps, x, abs_t;
  int k;

  printf ("Entre com x: ");
  scanf ("%f", &x);

  printf ("Entre com 0 < eps < 1: ");
  scanf ("%f", &eps);

  soma = abs_t = t = x;
  if (abs_t < 0) abs_t = -abs_t;
  k = 1;

  while (abs_t >= eps) {
    t = - t * x * x / ((2*k)*(2*k+1));
    soma = soma + t;
    k++;
    abs_t = t;
    if (abs_t < 0)
      abs_t = -abs_t;
  }

  printf ("sen(%f) = %f\n", x, soma);

  return 0;
}
```

14.7 Exercícios Recomendados

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

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