Eduardo Brunaldi dos Santos — 8642515,
Jorge Ashkar Ferreira Simondi — 8517081,
Victor Luiz da Silva Mariano Pereira — 8602444

Trabalho 2
Método de Integração Numérica
Simpson 1/3 Composta

Brasil

2018

Eduardo Brunaldi dos Santos — 8642515,
Jorge Ashkar Ferreira Simondi — 8517081,
Victor Luiz da Silva Mariano Pereira — 8602444

Trabalho 2
Método de Integração Numérica
Simpson 1/3 Composta

Universidade de São Paulo – USP
Instituto de Ciências Matemáticas e de Computação – ICMC
Cálculo Numérico – SME0104

Professor Murilo Francisco Tomé

Brasil
2018

Conteúdo

Introdução

Em cálculo, algumas integrais definidas não são tão simples de calcular, para isso foram desenvolvidos métodos numéricos para calcular essas integrais difíceis. As integrações numéricas são formas para aproximar uma intragral que tenha a caracteristica ou até mesmo quando não se tem a função propriamente dita, mas tem alguns valores da função em alguns pontos.

No caso desse segundo trabalho de Cálculo Numérico, tivemos que implementar um programa em Matlab/Octave ou em C que calculasse a integral de uma dada função utilizando o método de Simpson 1/3 composto.

Também precisamos fazer alguns testes solicitados para porvar a existência de raiz num certo intervalo, a partir dos resultados encontrados na implementação e teste do método.

1 Simpson 1/3 composto

A regra de Simpson 1/3 composto nada mais é que aplicar a regra de Simpson 1/3 a cada dois intervalos, é importante notar que é preciso ter uma quantidade par de subintervalos, ou seja, uma quantidade ímpar de pontos.

2 Instruções de uso

Nesse capítulo mostraremos como compilar e executar os programas que produzimos. Fizemos dois programas, um para calular a integração usando o método de Simpson 1/3 composto e outro para calcular a raiz de uma função utilizando o método de Newton. Porém os dois utilizam programas auxiliares, chamamos de bibliotecas, para modularizar nosso trabalho.

2.1 Arquivos principais

Os arquivos que produzimos são:

- `funcoes.c`
- `funcoes.h`
- `newton.c`
- `newton.h`
- `simpson.c`
- `simpson.h`
- `main_newton.c`
- `main_simpson.c`
- `Makefile`

2.2 Compilando

Para compilar os dois programas utilizaremos a ferramenta `make`, que utiliza como base o arquivo `Makefile`. Para compilar os dois programas basta estar no diretório dos arquivos e rodar os seguintes comandos:

```
1 make main_newton    #Para compilar o programa que utiliza o método de Newton
2 make main_simpson   #Para compilar o programa que utiliza o método de Simpson 1/3
```

2.3 Execução

Para executar os programas há duas formas, automática, a qual é preciso um arquivo de entrada, ou colocando os dados um a um. Como foi solicitado uns testes específicos, deixamos disponíveis, na pasta `inputs/`, os testes para serem executados.

2.3.1 Executando programa sem arquivo de entrada

```
1 make run_newton      #Para executar o método de Newton
2 make run_simpson     #Para executar o método de Simpson
```

2.3.2 Executando programa com arquivo de entrada

```
1 make run_newton < arquivo_de_input      #Para executar o método de Newton
2 make run_simpson < arquivo_de_input     #Para executar o método de Simpson
```

3 Resultados

3.1 Primeira etapa — Simpson 1/3 composto

Inicialmente nos foi pedido para utilizar o programa que criamos para calcular $F(1) \times F(2)$, em que $F(1) = I(f)(1) - 0.45$ e $F(2) = I(f)(2) - 0.45$ e verificar que o resultado dessa multiplicação é menor que zero, ou seja, que existe uma raiz nesse intervalo ($[1, 2]$).

Para realizar esse teste, primeiro temos que calcular a $I(f)(1)$ e $I(f)(2)$, para isso utilizaremos o programa `main_simpson` e dois arquivos auxiliares de entrada, o arquivo `simpson1.in` e o `simpson2.in`, que estão na pasta `inputs/`. Rodando o programa `main_simpson` da forma que foi explicada na subseção 2.3.2, obtivemos as seguintes saídas, respectivamente:

```
1 make run_simpson < inputs/simpson1.in
2 0.3413447460685457
```

```
1 make run_simpson < inputs/simpson2.in
2 0.4772498680518112
```

Agora fazendo a operação solicitada:

$$\begin{aligned} F(1) &= 0.3413447460685457 - 0.45 \\ &= -0.10865525393145431 \\ F(2) &= 0.4772498680518112 - 0.45 \\ &= 0.027249868051811177 \\ F(1) \times F(2) &= -0.10865525393145431 * 0.027249868051811177 \\ &= -0.0029608413327681677 \end{aligned}$$

■

3.2 Segunda etapa — Método de Newton

Nessa segunda etapa foi solicitado que aproximemos a raiz que foi provada a existência na seção anterior, a seção 3.1, utilizando o programa que fizemos.

Com chute inicial de 0.5 e precisão de 10^{-10}Também utilizando a arquivo auxiliar de entrada, no caso o `newton1.in`, obtivemos a seguinte saída:

```
1 make run_newton < inputs/newton1.in
2 Numero de iteracoes do metodo de Newton: 7
3 1.6448536269391639
```

Ou seja, a aproximação que obtivemos utilizando o método de Newton para a raiz foi o ponto 1.6448536269391639 e esse ponto com a precisão solicitada foi encontrado após sete iterações.

Conclusão

Apêndices

APÊNDICE A – Códigos Fonte

A.1 Programas principais

A.1.1 main_simpson.c

```c
/**
 *    Trabalho 2 - Método de Integração Numérica Simpson 1/3 Composta
 *
 *        Cálculo Numérico    SME-0104
 *        Prof.: Murilo Francisco Tomé
 *
 *        Eduardo Brunaldi dos Santos             8642515
 *        Jorge Ashkar Ferreira Simondi           8517081
 *        Victor Luiz da Silva Mariano Pereira    8602444
 */

#include <stdio.h>
#include <funcoes.h>
#include <newton.h>
#include <simpson.h>

int main (int argc, char *argv[]){
    long double x0;
    long double xN;
    long double n;

    scanf("%Lf", &x0);
    scanf("%Lf", &xN);
    scanf("%Lf", &n);

    printf("%.16Lf\n", simpson_composta(x0, xN, n, f_linha));

    return 0;
}
```

A.1.2 main_newton.c

```c
/**
 *    Trabalho 2 - Método de Integração Numérica Simpson 1/3 Composta
 *
 *        Cálculo Numérico    SME-0104
```

```
 *        Prof.: Murilo Francisco Tomé
 *
 *        Eduardo Brunaldi dos Santos              8642515
 *        Jorge Ashkar Ferreira Simondi            8517081
 *        Victor Luiz da Silva Mariano Pereira     8602444
 */

#include <stdio.h>
#include <funcoes.h>
#include <newton.h>
#include <simpson.h>

int main (int argc, char *argv[]){
    long double x0;
    long double e;

    scanf("%Lf", &x0);
    scanf("%Lf", &e);

    printf("%.16Lf\n", newton(x0, e, simpson_composta, f_linha));

    return 0;
}
```

A.2 Programas auxiliares

A.2.1 Makefile

```
main_newton: funcoes newton simpson
    gcc main_newton.c -o main_newton funcoes.o newton.o simpson.o -I . -lm

main_simpson: funcoes newton simpson
    gcc main_simpson.c -o main_simpson funcoes.o newton.o simpson.o -I . -lm

funcoes:
    gcc -c funcoes.c -I .

newton:
    gcc -c newton.c -I .

simpson:
    gcc -c simpson.c -I .

run_newton:
    @./main_newton
```

```
18
19 run_simpson:
20     @./main_simpson
21
22 clean:
23     rm -f *.o main_simpson main_newton
```

A.2.2 *Header* da biblioteca de Funções (`funcoes.h`)

```
1  /**
2   *    Trabalho 2 - Método de Integração Numérica Simpson 1/3 Composta
3   *
4   *        Cálculo Numérico    SME-0104
5   *        Prof.: Murilo Francisco Tomé
6   *
7   *        Eduardo Brunaldi dos Santos             8642515
8   *        Jorge Ashkar Ferreira Simondi           8517081
9   *        Victor Luiz da Silva Mariano Pereira    8602444
10  */
11
12 #ifndef FUNCOES_H
13 #define FUNCOES_H
14
15 /**
16  * Definindo ponteiros para funções, serivrá para poder chamar a função
17  * desejada por parâmetro
18  */
19 typedef long double (*Funcao_Derivada)(long double);
20 typedef long double (*Funcao)(long double, long double, int, Funcao_Derivada);
21
22 /**
23  * Função que calcula o valor de f'(x) dada por:
24  *
25  *        f'(x) = (1/sqrt(2*))*e^((-x^2)/2)
26  *
27  * @param  x ponto x onde será calculada f'(x)
28  * @return   valor da função f'(x) calculada no ponto x
29  */
30 long double f_linha (long double x);
31
32 #endif
```

A.2.3 Implementação da biblioteca de Funções (`funcoes.c`)

```c
/**
 *    Trabalho 2 - Método de Integração Numérica Simpson 1/3 Composta
 *
 *        Cálculo Numérico    SME-0104
 *        Prof.: Murilo Francisco Tomé
 *
 *        Eduardo Brunaldi dos Santos             8642515
 *        Jorge Ashkar Ferreira Simondi           8517081
 *        Victor Luiz da Silva Mariano Pereira    8602444
 */

#include <funcoes.h>
#include <math.h>

/**
 * Função que calcula o valor de f'(x) dada por:
 *
 *        f'(x) = (1/sqrt(2*))*e^((-x^2)/2)
 *
 * @param  x ponto x onde será calculada f'(x)
 * @return   valor da função f'(x) calculada no ponto x
 */
long double f_linha (long double x){
    return (1.0/(sqrt(2*M_PI)*1.0))*exp((- pow(x, 2.0))/2.0);
}
```

A.2.4 *Header* da biblioteca de Simpson (`simpson.h`)

```c
/**
 *    Trabalho 2 - Método de Integração Numérica Simpson 1/3 Composta
 *
 *        Cálculo Numérico    SME-0104
 *        Prof.: Murilo Francisco Tomé
 *
 *        Eduardo Brunaldi dos Santos             8642515
 *        Jorge Ashkar Ferreira Simondi           8517081
 *        Victor Luiz da Silva Mariano Pereira    8602444
 */

#ifndef SIMPSON_H
#define SIMPSON_H

/**
 * Definindo ponteiros para funções, serivrá para poder chamar a função
```

```
17  * desejada por parâmetro
18  */
19 typedef long double (*Funcao_Derivada)(long double);
20 typedef long double (*Funcao)(long double, long double, int, Funcao_Derivada);
21
22 /**
23  * Função para calcular a integração de uma certa função f usando o método de
24  * Simpson 1/3 Composta, dada por:
25  *
26  * I^N_S = (h/3) * (f(x0) + f(xN) + 4*SUM(f(x_impares)) + 2*SUM(f(x_pares)))
27  *
28  * Utilizando um intervalo, a quantidade de divisões do intervalo e a função a
29  * ser integrada.
30  *
31  * @param  x0 valor inicial do intervalo
32  * @param  xN valor final do intervalo
33  * @param  n  quantas vezes será dividido o intervalo
34  * @param  f  função a ser integrada
35  * @return    valor aproximado da integral da função f
36  */
37 long double simpson_composta(long double x0, long double xN, int n, Funcao_Derivada
    ↪  f);
38
39 #endif
```

A.2.5 Implementação da biblioteca de Simpson (`simpson.c`)

```
1  /**
2   *    Trabalho 2 - Método de Integração Numérica Simpson 1/3 Composta
3   *
4   *        Cálculo Numérico    SME-0104
5   *        Prof.: Murilo Francisco Tomé
6   *
7   *        Eduardo Brunaldi dos Santos             8642515
8   *        Jorge Ashkar Ferreira Simondi           8517081
9   *        Victor Luiz da Silva Mariano Pereira    8602444
10  */
11
12 #include <funcoes.h>
13 #include <simpson.h>
14
15 /**
16  * Função para calcular a integração de uma certa função f usando o método de
17  * Simpson 1/3 Composta, dada por:
18  *
19  * I^N_S = (h/3) * (f(x0) + f(xN) + 4*SUM(f(x_impares)) + 2*SUM(f(x_pares)))
```

```
20  *
21  * Utilizando um intervalo, a quantidade de divisões do intervalo e a função a
22  * ser integrada.
23  *
24  * @param  x0 valor inicial do intervalo
25  * @param  xN valor final do intervalo
26  * @param  n  quantas vezes será dividido o intervalo
27  * @param  f  função a ser integrada
28  * @return    valor aproximado da integral da função f
29  */
30 long double simpson_composta(long double x0, long double xN, int n, Funcao_Derivada
    ↪  f){
31     // Variáveis auxiliares
32     long double x;
33     long double h;
34     long double resposta;
35     long double pares;
36     long double impares;
37
38     // Iterador
39     int i;
40
41     // Verifica se o n de entrada é par, caso ímpar adiciona 1
42     if (n % 2 != 0)
43         n += 1;
44
45     // Cálculo de h
46     h = (xN - x0)/n;
47
48     // Inicializa valores dos somatórios
49     pares = 0;
50     impares = 0;
51
52     // atualiza o valor de x
53     x = x0 + h;
54
55     // Inicia a resposta com a f(0) e f(x_n)
56     resposta = f(x0) + f(xN);
57
58     // Cálculo dos somatórios de pares e ímpares
59     for (i = 1; i < n; i++){
60         // Caso par
61         if (i % 2 == 0)
62             pares += f(x);
63         // Caso ímpar
64         else
65             impares += f(x);
```

```
66          // Atualiza o x
67          x += h;
68      }
69
70      // Adiciona 4*somatório dos ímpares + 2*somatório dos pares na resposta
71      resposta += 4 * impares + 2 * pares;
72      // Multiplica o valor achado pelas somas por h/3
73      resposta *= h/3;
74
75      // Retorna o valor aproximado da integral utilizando o método de
76      // Simpson 1/3 composto
77      return resposta;
78  }
```

A.2.6 *Header* da biblioteca de Newton (`newton.h`)

```
1   /**
2    *    Trabalho 2 - Método de Integração Numérica Simpson 1/3 Composta
3    *
4    *        Cálculo Numérico    SME-0104
5    *        Prof.: Murilo Francisco Tomé
6    *
7    *        Eduardo Brunaldi dos Santos             8642515
8    *        Jorge Ashkar Ferreira Simondi           8517081
9    *        Victor Luiz da Silva Mariano Pereira    8602444
10   */
11
12  #ifndef NEWTON_H
13  #define NEWTON_H
14
15  /**
16   * Definindo ponteiros para funções, serivrá para poder chamar a função
17   * desejada por parâmetro
18   */
19  typedef long double (*Funcao_Derivada)(long double);
20  typedef long double (*Funcao)(long double, long double, int, Funcao_Derivada);
21
22  /**
23   * Função que calcula a raiz de uma função utilizando o método de Newton
24   * @param  x0       Valor do chute inicial
25   * @param  e        Precisão esperada
26   * @param  f        Função que será analisada
27   * @param  f_linha Derivada da função que será analisada
28   * @return          Aproximação da raiz da função dada, com uma certa precisão
29   */
30  long double newton(long double x0, long double e, Funcao f, Funcao_Derivada f_linha);
```

```
31
32 #endif
```

A.2.7 Implementação da biblioteca de Newton (`newton.c`)

```
1  /**
2   *    Trabalho 2 - Método de Integração Numérica Simpson 1/3 Composta
3   *
4   *        Cálculo Numérico    SME-0104
5   *        Prof.: Murilo Francisco Tomé
6   *
7   *        Eduardo Brunaldi dos Santos             8642515
8   *        Jorge Ashkar Ferreira Simondi           8517081
9   *        Victor Luiz da Silva Mariano Pereira    8602444
10  */
11
12 #include <newton.h>
13 #include <stdio.h>
14 #include <math.h>
15 #include <funcoes.h>
16 #include <simpson.h>
17
18 /**
19  * Função que calcula a raiz de uma função utilizando o método de Newton
20  * @param  x0      Valor do chute inicial
21  * @param  e       Precisão esperada
22  * @param  f       Função que será analisada
23  * @param  f_linha Derivada da função que será analisada
24  * @return         Aproximação da raiz da função dada, com uma certa precisão
25  */
26 long double newton(long double x0, long double e, Funcao f, Funcao_Derivada f_linha){
27     // Variáveis auxiliares
28     long double x_atual;
29     long double x_anterior;
30     int iteracoes = 0;
31
32     // Atualiza o valor de x_n+1
33     x_atual = x0;
34
35     do{
36         // Atualiza valor de x_n
37         x_anterior = x_atual;
38         // Calcula novo valor de x_n+1
39         x_atual = x_anterior - ((f(0, x_anterior, 200,
           ↪  f_linha)-0.45)/f_linha(x_anterior));
40         // Atualiza contador de iterações
```

```
41          iteracoes += 1;
42      }while(fabs(x_atual - x_anterior) > e); // Verifica condição da precisão
43
44      // Se não quiser imprimir o número de iterações do método de Newton, basta
45      // comentar a próxima linha
46      printf("Numero de iteracoes do metodo de Newton: %d\n", iteracoes);
47
48      // Retorna a raiz calculada
49      return x_atual;
50  }
```