

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	3
1	SIMPSON 1/3 COMPOSTO	5
2	RESULTADOS	7
	Conclusão	9
	 APÊNDICES	 11
	APÊNDICE A – CÓDIGOS FONTE	13
A.1	Programa principal (main.c)	13
A.2	Biblioteca auxiliar	13
A.2.1	<i>Header da biblioteca (simpson_e_newton.h)</i>	13
A.2.2	<i>Implementação da biblioteca (simpson_e_newton.c)</i>	15

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 característica 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.

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.

1 Simpson $1/3$ composto

2 Resultados

Conclusão

Apêndices

APÊNDICE A – Códigos Fonte

A.1 Programa principal (main.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 <stdio.h>
13  #include <simpson_e_newton.h>
14
15  int main (int argc, char *argv[]){
16      long double x0;
17      long double xN;
18      long double n;
19
20      scanf("%Lf", &x0);
21      scanf("%Lf", &xN);
22      scanf("%Lf", &n);
23
24      printf("Valor da integral, utilizando Simpson 1/3 Composta: %.16Lf\n",
25             ↪ simpson_composta(x0, xN, n, f_linha));
26      printf("Valor da raiz pelo metodo de Newton: %.16Lf\n", newton(0.5, 0.0000000001,
27             ↪ simpson_composta, f_linha));
28      return 0;
29  }
```

A.2 Biblioteca auxiliar

A.2.1 Header da biblioteca (simpson_e_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 SIMPSON_E_NEWTON_H
13 #define SIMPSON_E_NEWTON_H
14
15 /**
16  * Definindo ponteiros para funções, servirá 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  $f(x)$ , sendo  $f(x)$  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 /**
33  * Função para calcular a integração de uma certa função f usando o método de
34  * Simpson 1/3 Composta, dada por:
35  *
36  *       $I_N^S = (h/3) * (f(x_0) + f(x_N) + 4 * \text{SUM}(f(x_{\text{impares}})) + 2 * \text{SUM}(f(x_{\text{pares}})))$ 
37  *
38  * Utilizando um intervalo, a quantidade de divisões do intervalo e a função a
39  * ser integrada.
40  *
41  * @param x0 valor inicial do intervalo
42  * @param xN valor final do intervalo
43  * @param n quantas vezes será dividido o intervalo
44  * @param f função a ser integrada
45  * @return valor aproximado da integral da função f
46  */
47 long double simpson_composta(long double x0, long double xN, int n, Funcao_Derivada
48 ↪ f);
49 /**

```

```

50  * Função que calcula a raiz de uma função utilizando o método de Newton
51  * @param x0      Valor do chute inicial
52  * @param e        Precisão esperada
53  * @param f        Função que será analisada
54  * @param f_linha  Derivada da função que será analisada
55  * @return         Aproximação da raiz da função dada, com uma certa precisão
56  */
57  long double newton(long double x0, long double e, Funcao f, Funcao_Derivada f_linha);
58
59  #endif

```

A.2.2 Implementação da biblioteca (simpson_e_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 <stdio.h>
13  #include <math.h>
14  #include <simpson_e_newton.h>
15
16  /**
17   * Função que calcula o valor de f(x), sendo f(x) dada por:
18   *
19   *    $f(x) = (1/\sqrt{2}) * e^{((-x^2)/2)}$ 
20   *
21   * @param x ponto x onde será calculada f(x)
22   * @return valor da função f(x) calculada no ponto x
23   */
24  long double f_linha (long double x){
25      return (1.0/(sqrt(2*M_PI)*1.0))*exp((- pow(x, 2.0))/2.0);
26  }
27
28  /**
29   * Função para calcular a integração de uma certa função f usando o método de
30   * Simpson 1/3 Composta, dada por:
31   *
32   *  $I \approx (h/3) * (f(x_0) + f(x_N) + 4 * SUM(f(x_{impares})) + 2 * SUM(f(x_{pares})))$ 
33   *

```

```
34  * Utilizando um intervalo, a quantidade de divisões do intervalo e a função a
35  * ser integrada.
36  *
37  * @param x0 valor inicial do intervalo
38  * @param xN valor final do intervalo
39  * @param n quantas vezes será dividido o intervalo
40  * @param f função a ser integrada
41  * @return valor aproximado da integral da função f
42  */
43  long double simpson_composta(long double x0, long double xN, int n, Funcao_Derivada
↪ f){
44      // Variáveis auxiliares
45      long double x;
46      long double h;
47      long double resposta;
48      long double pares;
49      long double impares;
50
51      // Iterador
52      int i;
53
54      // Verifica se o n de entrada é par, caso ímpar adiciona 1
55      if (n % 2 != 0)
56          n += 1;
57
58      // Cálculo de h
59      h = (xN - x0)/n;
60
61      // Inicializa valores dos somatórios
62      pares = 0;
63      impares = 0;
64
65      // atualiza o valor de x
66      x = x0 + h;
67
68      // Inicia a resposta com a f(0) e f(x_n)
69      resposta = f(x0) + f(xN);
70
71      // Cálculo dos somatórios de pares e ímpares
72      for (i = 1; i < n; i++){
73          // Caso par
74          if (i % 2 == 0)
75              pares += f(x);
76          // Caso ímpar
77          else
78              impares += f(x);
79          // Atualiza o x
```

```
80     x += h;
81 }
82
83 // Adiciona 4*somatório dos ímpares + 2*somatório dos pares na resposta
84 resposta += 4 * impares + 2 * pares;
85 // Multiplica o valor achado pelas somas por h/3
86 resposta *= h/3;
87
88 // Retorna o valor aproximado da integral utilizando o método de
89 // Simpson 1/3 composto
90 return resposta;
91 }
92
93 /**
94  * Função que calcula a raiz de uma função utilizando o método de Newton
95  * @param x0      Valor do chute inicial
96  * @param e        Precisão esperada
97  * @param f        Função que será analisada
98  * @param f_linha  Derivada da função que será analisada
99  * @return         Aproximação da raiz da função dada, com uma certa precisão
100 */
101 long double newton(long double x0, long double e, Funcao f, Funcao_Derivada f_linha){
102     // Variáveis auxiliares
103     long double x_atual;
104     long double x_anterior;
105     int iteracoes = 0;
106
107     // Atualiza o valor de x_n+1
108     x_atual = x0;
109
110     do{
111         // Atualiza valor de x_n
112         x_anterior = x_atual;
113         // Calcula novo valor de x_n+1
114         x_atual = x_anterior - ((f(0, x_anterior, 200,
115             ↪ f_linha)-0.45)/f_linha(x_anterior));
116         // Atualiza contador de iterações
117         iteracoes += 1;
118     }while(fabs(x_atual - x_anterior) > e); // Verifica condição da precisão
119
120     // Se não quiser imprimir o número de iterações do método de Newton, basta
121     // comentar a próxima linha
122     printf("Numero de iteracoes do metodo de Newton: %d\n", iteracoes);
123
124     // Retorna a raiz calculada
125     return x_atual;
126 }
```