**UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ**
**CORDENAÇÃO DE INFORMÁTICA**
**TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS**

**MARCIO HILGEMBERG ALMEIDA**

# ESTUDO COMPARATIVO DE ALGORITMOS DE INTEGRAÇÃO NUMÉRICA

**TRABALHO DE CONCLUSÃO DE CURSO**

**PONTA GROSSA**
**2011**

**MARCIO HILGEMBERG ALMEIDA**

# ESTUDO COMPARATIVO DE ALGORITMOS DE INTEGRAÇÃO NUMÉRICA

Trabalho de Conclusão de Curso em Tecnologia em Análise e Desenvolvimento de Sistemas da Universidade Tecnológica Federal do Paraná como requisito parcial para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Orientador: Prof. Dr. André Koscianski

**PONTA GROSSA**

**2011**

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Campus Ponta Grossa

Diretoria de Graduação e Educação Profissional

TERMO DE APROVAÇÃO

ESTUDO COMPARATIVO DE ALGORITMOS DE INTEGRAÇÃO NUMÉRICA

por

MARCIO HILGEMBERG ALMEIDA

Este Trabalho de Conclusão de Curso foi apresentado em 10 de novembro de 2011 como requisito parcial para a obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____
Prof. Dr. André Koscianski
Prof. Orientador

_____
Prof. Dr. Marcus Vinicius Drissen Silva
Membro titular

_____
Prof(a). Renata Luiza Stange Carneiro Gomes
Membro titular

_____
Prof(a). Helyane Bronoski Borges
Responsável pelos Trabalhos de Conclusão de Curso

_____
Prof. Dr. André Koscianski
Coordenador do Curso Tecnologia em Análise e Desenvolvimento de Sistemas
UTFPR - Campus Ponta Grossa

À minha Família pela compreensão nas horas ausente do convívio familiar em função da dedicação aos estudos e pesquisas necessárias ao curso.

## Agradecimentos

Primeiramente, agradeço a Deus por ter me dado a oportunidade de estar no mundo.

Aos amigos da UTFPR, que me "aturam" todos os dias. Muitas das pessoas que passaram e passam pelo que eu passei e passo: ficar longe da família em busca de um ideal comum.

Aos amigos internautas, adquiridos nas noites e madrugadas insones e, hoje, conhecidos pessoalmente, deixamos de ser os estranhos do outro lado do computador.

Tenho muito a agradecer e a muitas pessoas. Não cito nomes para não ser injusto com pessoas que me auxiliaram até onde já cheguei. Meus agradecimentos especiais a:

Elisaldo Oliveira Almeida e Sonia Mara Hilgemberg Almeida por me apoiarem e investirem nos meus estudos e ao meu orientador o Prof. Dr. André Koscianski.

## Resumo

ALMEIDA, Marcio Hilgemberg. **Estudo comparativo de alguns algoritmos de integração numérica.** 2011. 30 f. Trabalho de Conclusão de Curso (Análise e Desenvolvimento de Sistemas) - Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2011.

Integração é uma ferramenta muito usada em engenharia. Existem vários algoritmos para calcular integrais numericamente, com o auxílio de computadores. Esses algoritmos variam em relação à velocidade e à precisão. Essas duas características dependem do hardware e do software utilizados.
Este trabalho apresenta três métodos para a realização de cálculos de integrais. Fatores que afetam o desempenho desses métodos são examinados. Os algoritmos foram implementados usando linguagem C. O trabalho apresenta testes e comparações entre os algoritmos.

**Palavras-chave**: cálculo numérico, integração, teste de desempenho.

## ABSTRACT

ALMEIDA, Marcio Hilgemberg. **Comparative study of numerical integration algorithms.** 2011. 30 f. Trabalho de Conclusão de Curso (Análise e Desenvolvimento de Sistemas) - Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2011.

Integration is widely used in engineering. It can be calculated computationally, with the help of several algorithms. These algorithms perform differently with respect to speed and accuracy. These two characteristics depend on the hardware and software used.
This work presents three methods for numerical integration. The main factors affecting the performance of these methods are examined. The algorithms were implemented using C language. The work presents tests and comparisons between the algorithms.



**Keywords**: numerical calculus, integration, benchmarking.

**Lista de figuras**

## Lista de quadros

## Lista de gráficos

**Lista de tabelas**

**Sumário**

# 1. Introdução

A ciência vem ao longo dos anos tentando comprovar varias teorias do inicio do século XX. E isso só é possível através da evolução da tecnologia, que auxilia a resolver cálculos complexos e que são impossíveis de ser resolvidos manualmente.

No ramo da física e da engenharia, um tipo de cálculo muito utilizado é a Integração Numérica. A resolução desse tipo de cálculo é complexa e torna-se mais difícil quando passada para área computacional para ser implementada e executada.

Inúmeros fatores como *hardware* e *software*, podem afetar o resultado final tornando ele impreciso e demorado, pois a maneira que foi implementada torna a velocidade do cálculo baixa. E o que essas áreas querem é a realização de cálculos com números grandes, executando outros inúmeros cálculos simultâneos no menor tempo possível e com precisão.

Este trabalho faz a comparação de três algoritmos para o cálculo de integrais, avaliando a precisão mais aceitável e o melhor desempenho para retornar um resultado e inúmeros resultados. Também são abordados fatores que influenciam negativamente na execução dos testes dos algoritmos e como solucionar ou contornar eles.

# 2. Revisão bibliográfica

## 2.1. Integração

Em cálculo numérico a integral corresponde intuitivamente a uma somatória de parcelas. Quanto maior o numero de parcelas e menor o intervalo entre elas, mais precisa será o resultado da integral.

As integrais podem ser classificadas em indefinidas e definidas.

Uma integral indefinida é uma função "$f(x)$" ou conjunto de funções resultantes da derivada de uma função "$F$". Está função "$F$" é chamada de primitiva.

A integral definida, segundo Flemming e Gonçalves (1992), consiste em uma função "$f$" definida em um intervalo de "$a$" até "$b$", retornando um valor, mas "$a$" e "$b$" devem estar entre "$-\infty$" e "$+\infty$". A representação é dada por $\int_a^b f(x)dx$ .

As integrais são muito utilizadas na resolução de problemas em áreas como física e engenharia. Pode-se ilustrar a idéia de integral com dois exemplos.

1) $\int_1^3 x\ dx = F(x)|_1^3 = \frac{x^2}{2}\Big|_1^3 = \frac{3^2}{2} - \frac{1^2}{2} = \frac{9-1}{2} = 4$
2) Uma fábrica despeja dejetos no leito de um rio. É registrada a quantidade de poluentes lançada em quilogramas por hora. Deseja-se conhecer a quantidade total de dejetos que foram lançados durante certo prazo, como um dia.
3) Uma indústria monitora o consumo de energia elétrica. Para isso um equipamento indica a quantidade de energia consumida em um dado momento e deseja-se saber a quantidade total durante um mês.

A tabela 1 mostra os detalhes do primeiro exemplo.

**Tabela 1 - Exemplos de Informações que precisam ser integrados**

| *Hora* | *Quantidade de Poluentes (Kg/hora)* |
|---|---|
| 08:00 | 2 |
| 10:00 | 3 |
| 13:00 | 4 |
| 17:00 | 1 |

**Fonte: SILVA, Renato S. ALMEIDA, Regina C. Notas de aula do Curso de Iniciação a Modelagem (LNCC/INPA, 2003)**

## 2.2. Precisão numérica

Antes de começar a trabalhar com os algoritmos fundamentais, é necessário entender sobre a precisão dos sistemas computacionais em relação aos números decimais/fracionários no sistema binário.

É impossível converter de forma exata alguns números, pois os mesmos são dizimas periódicas em base binária. Quando o número é convertido da base decimal para binário, se forem usadas poucas casas depois da vírgula, poderá existir uma diferença muito grande entre o numero real e o numero obtido. A solução mais utilizada é trabalhar com o maior numero de casas possíveis, para que se obtenha uma maior precisão.

Para exemplificar, podemos converter o numero $0,3_{10}$ para binário. Se trabalharmos com apenas cinco casas depois da vírgula, obtemos o numero $0,01001_2$ que, convertido novamente para a base decimal é $0,28125_{10}$. Se utilizarmos nove casas o resultado é $0,010011001_2$ que convertido para decimal é igual a $0,298828125_{10}$. Como se pode ver, acrescentando quatro algarismos após a vírgula foi possível melhorar a precisão em 0.017 unidades.

Computadores trabalham com tipos de dado *float* e *double*, que possuem uma boa precisão. Em trabalhos de engenharia em que se usa cálculo numérico, procura-se evitar números muito pequenos justamente para reduzir os erros de precisão numérica. Assim, por exemplo, em vez de

trabalhar com metros pode-se usar como unidade o milímetro (1m = 1000mm), dessa forma reduzindo o problema de conversão entre decimal e binário.

## 2.3. Regras de integração numérica

A regra do trapézio, a regra trapézio estendida ou composta e a regra de Simpson são as mais simples e conhecidas. Foram as regras selecionadas para os testes de desempenho e precisão neste trabalho.

### 2.3.1. Regra do trapézio

Para uma função fixa "$f(x)$" a ser integrada entre os limites "$a$" até "$b$", a regra do trapézio calcula a média da função nessas extremidades "$a$" e "$b$" e multiplica pelo comprimento do intervalo.

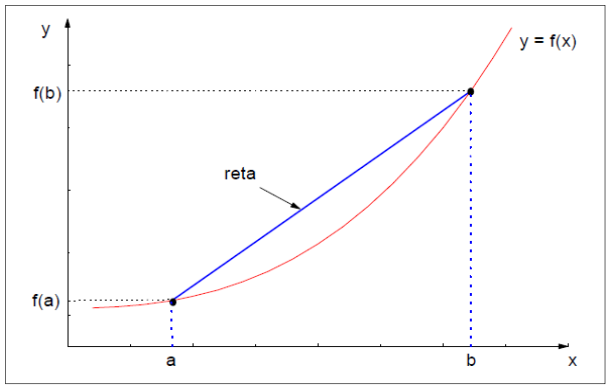


**Figura 1- Regra do trapézio**

**Fonte: adaptado de Cláudio e Marins(2000 p. 269)**

Como essa aproximação fica muito longe da curva original, a regra do trapézio melhora o resultado dividindo o intervalo em vários subintervalos e realizando o mesmo cálculo em cada um deles. O valor da integral é igual a soma dos cálculos feitos em cada intervalo.

O algoritmo do trapézio pode ser definido conforme o código no quadro 1, onde é passada uma função a ser integrada como parâmetro.

```
#define FUNC(x) ((*func)(x))
float trapzd(float (*func)(float), float a, float b, int n)
{
    float x,tnm,sum,del;
    static float s;
    int it,j;
    if (n == 1)
        return (s=0.5*(b-a)*(FUNC(a)+FUNC(b)));
    } else {
        for (it=1,j=1;j<n-1;j++)
            it <<= 1;
        tnm=it;
        del=(b-a)/tnm;
        x=a+0.5*del;
        for (sum=0.0,j=1;j<=it;j++,x+=del)
            sum += FUNC(x);
        s=0.5*(s+(b-a)*sum/tnm);
        return s;
    }
}
```

**Quadro 1 - Algoritmo da regra do trapézio**

**Fonte: Press, Teukolsky e Vetterling (1992)**

Essa rotina também recebe como parâmetro a quantidade de subintervalos desejada. Quanto maior esse número, maior a precisão, embora também aumente o tempo de cálculo.

Quando o valor de “n” é igual a “1” (n=1), é executado o cálculo da área do trapézio no modo mais simples. No instante que “n” é maior que um (n>1), é realizado as operações dentro do condicional “else”, que tem o objetivo de dividir o intervalo “$a$” até “$b$” em subintervalos e para cada um que for criado realizar o cálculo da área do trapézio.

O termo “(*func)(float)” é um ponteiro para a função a ser integrada entre as extremidades “$a$” e “$b$”, que é declarada no inicio do código.

Para realizar o teste da rotina, foi utilizado o código mostrado no quadro 2. Quando a rotina principal (main) é executada, a rotina “trapzd” é chamada para ser executada dentro de um laço de repetição. Esse laço tem os objetivos de determinar o valor da variável “n” dentro do método “trapzd”, que determina o numero de subintervalos que serão gerados.

```
main()
{
    float s;
    int a=0, b = 10, m = 15, j;
    for(j=1;j<=m+1;j++){ s=trapzd(func,a,b,j); }
    printf("trapzd = %f\n",s);
    getchar();
}
```

**Quadro 2 - Método que chama a regra do trapézio**
**Fonte: Press, Teukolsky e Vetterling (1992)**

A variável “m” é o numero de vezes que a função será calculada até o resultado mais preciso, e também é a condição de parada do laço.

Para “j = 1” o valor de “n” no método chamado será o mesmo valor e a rotina retorna o valor mais básico da $\int_a^b f(x)dx$ . Nas próximas chamadas do laço com “n = 2”,..., n=m+1, irão melhorar a precisão do resultado.

Na teoria quanto maior o valor da variável “m”, maior será a precisão do resultado. Durante a realização dos testes este fator será colocado à prova.

### 2.3.2. Regra do trapézio estendida ou composta

A regra do trapézio simples é uma rotina simples e bastante robusta para realização de integrais de funções que não são muito suaves. Mas problemas podem surgir especialmente se forem necessários muitos passos na tentativa de alcançar a acurácia necessária. À medida que se usam mais

passos, a regra do trapézio usará mais intervalos. Esses intervalos ficam muito pequenos e, por causa dos erros de precisão numérica, o resultado obtido pode se tornar pior ou a rotina pode não conseguir convergir.

Uma solução é usar o algoritmo da regra trapézio composta. Esta regra decompõe em vários subintervalos pequenos e aplicamos a cada um deles a regra do trapézio.

```
float qtrap(float (*func)(float), float a, float b)
{
    int j;
    float s, olds = -1.0e30;
    for (j=1;j<=JMAX;j++){
        s=trapzd(func,a,b,j);
         if (fabs(s-olds) < EPS*fabs(olds) || (s == 0.0 && olds == 0.0))
                return s;
        olds=s;
    }return 0.0;
}
```

**Quadro 3 - Algoritmos da regra do trapézio estendido**
**Fonte: Press, Teukolsky e Vetterling (1992)**

Quando esse algoritmo é executado, apenas passamos como parâmetros a função e as extremidades. A sub-rotina chama várias vezes à função de integração pela regra do trapézio e compara os resultados. Se a diferença entre as chamadas for muito pequena, a função chega à conclusão de que não vale a pena diminuir o tamanho dos intervalos e retorna o valor calculado. Caso contrário, a rotina continua a chamar a regra do trapézio aumentando o valor de “j”. A comparação do “j” com “JMAX” evita um laço infinito.

A variável “JMAX” é definida na declaração inicial do programa. Ela será abordada na parte dos testes, pois seu valor influencia na precisão do resultado. Ela funciona como a variável “m” citada na regra do trapézio.

O quadro 4 mostra como o método que chama a regra do trapézio composto é mais simples do que da regra do trapézio simples.

```
    s=qtrap(func,a,b);
```

**Quadro 4 - Método que chama a regra do trapézio estendido**
**Fonte: Press, Teukolsky e Vetterling (1992)**

### 2.3.3. Regra de Simpson

A Regra de Simpson é considerada a mais precisa segundo Press, Teukolsky e Vetterling (1992, p139). Ela consiste na aproximação da curva formada da função contínua "$f(x)$" a ser integrada entre os limites "$a$" até "$b$", por uma função de segunda ordem, criando uma parábola e a aproximando de uma curva.

Na regra do trapézio, os pontos "$a$" e "$b$" eram interligados por uma linha reta. Já na regra de Simpson, esses pontos são ligados por uma parábola, que pode se aproximar mais da curva da função a ser integrada.

O próximo quadro mostra o algoritmo.

```
float qsimp(float (*func)(float), float a, float b){
    float trapzd(float (*func)(float), float a, float b, int n);
    int j;
    float s,st,ost,os;
    ost = os = -1.0e30;
    for (j=1;j<=JMAX;j++) {
        st=trapzd(func,a,b,j);
        s=(4.0*st-ost)/3.0;
        if (j > 5)
            if (fabs(s-os) < EPS*fabs(os) || (s == 0.0 && os == 0.0))    return s;
        os=s;
        ost=st;
    } return 0.0;
}
```

**Quadro 5 - Algoritmo da regra de Simpson**
**Fonte: Press, Teukolsky e Vetterling (1992)**

Conforme o quadro acima o algoritmo de Simpson usa a mesma lógica da regra do trapézio composta. O intervalo formado "$a$" e "$b$" é dividido em subintervalos. A cada um deles é aplicado a regra do trapézio simples. A parcial obtida da regra do trapézio é submetida a outra operação que tem o objetivo de retirar a diferença que a regra do trapézio gera e traz o resultado o mais próximo do valor real. Acima da quinta execução do laço, é verificada a precisão do cálculo, se estiver abaixo do esperado, o laço é interrompido e o resultado final retornado.

## 2.4. Medição de desempenho

Um dos itens que será comparado durante os testes é o tempo que cada algoritmo leva para executar inúmeras vezes o mesmo cálculo. Isso se torna mais difícil estabelecer, pois as tarefas a serem processadas são gerenciadas pelo Sistema Operacional.

O tempo não é preciso durante a execução, porque o sistema operacional pode interromper o processo de cálculo, executar outra tarefa de prioridade mais alta e depois voltar a executar aquele processo. Se a execução do programa for repetida, o tempo de execução provavelmente será diferente, pois pode haver ou não interrupções do sistema nessa nova execução.

As figuras 2 e 3 mostram dois exemplos, no Linux e no Windows. É apresentada nas duas figuras a lista dos processos carregados e como o sistema operacional pode interromper um processo para que outro seja executado.

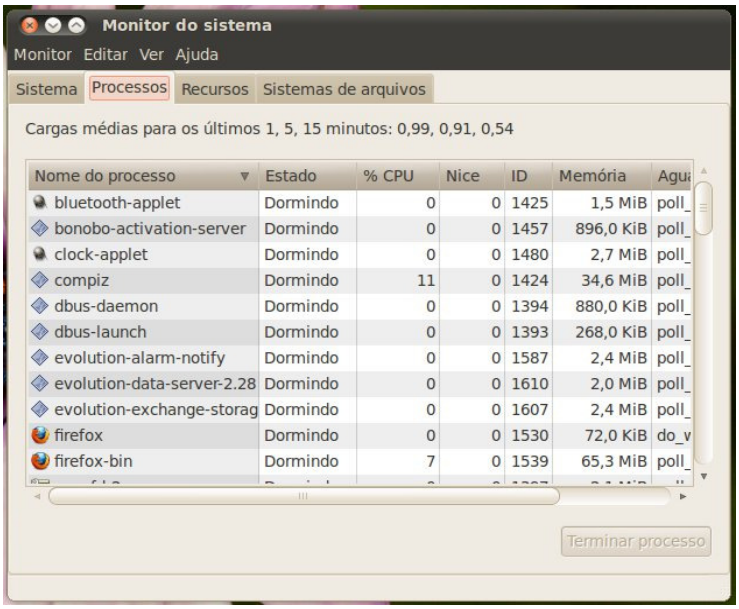


**Figura 2 – Gerenciador de Processos do Sistema Linux**
**Fonte: Autoria Própria**

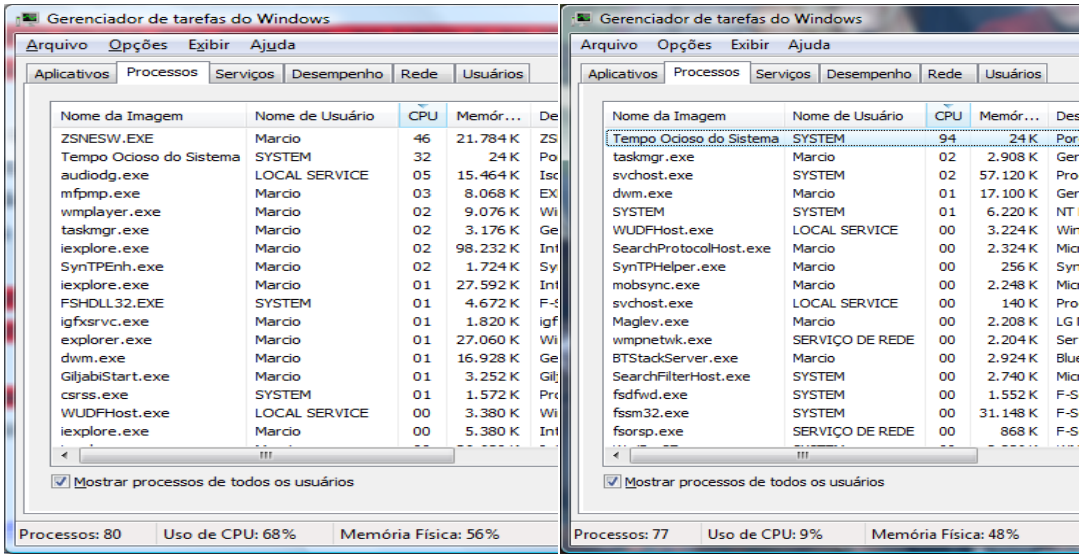


**Figura 3 - Gerenciador de Processos do Sistema Windows em diferentes instantes**
**Fonte: Autoria Própria**

Para medir o tempo de execução pode-se usar a função time(), da biblioteca padrão (stdlib) da linguagem C/C++. Contudo essa função não é muito precisa. Uma medição melhor pode ser obtida, no caso do sistema Windows, fazendo chamadas as funções QueryPerformanceFrequency e QueryPerformanceCounter.

A QueryPerformanceFrequency obtem o valor da freqüência de funcionamento do processador e o armazena em uma variável. Já a QueryPerformanceCounter captura o valor no momento do sinal do clock antes

de iniciar os cálculos e captura novamente após os cálculos. Com a diferença entre o inicial e o final temos o número de sinais de clock do período que ocorreu a execução do programa. O valor da diferença entre o inicial e final é dividido pelo valor da freqüência e obtem-se o tempo total para a execução.

Dependendo dos componentes do equipamento, a unidade de tempo obtida através dessas operações pode variar entre segundos, milissegundos ou milionésimo de segundos[1].

[1] http://support.microsoft.com/kb/172338

## 3. Implementação e testes

Depois de conhecer as regras mais comuns, foi realizado a implementação e os testes dos algoritmos dessa regras. Pois. O teste para os três algoritmos foi utilizando uma função do segundo grau "$f(x) = x^2 + 2x + 1$" com os limites entre zero e dez. Calculando algebricamente essa integral, temos:

$$\int_a^b f(x)\,dx = \int_0^{10} x^2 + 2\,x + 1\,dx$$

$$F(x)|_0^{10} = F(10) - F(0)$$

$$\frac{x^3}{3} + 2\frac{x^2}{2} + x = \frac{10^3}{3} + 10^2 + 10 - \frac{0^3}{3} + 0^2 + 0$$

$$333{,}33333333 + 100 + 10 = 443{,}33333333$$

$$\int_0^{10} x^2 + 2\,x + 1\,dx = 443{,}33333333$$

Para a implementação dos códigos foi necessário primeiro declarar a função utilizada dentro da sub-rotina "float func (float x)", convertendo da forma algébrica $x^2 + 2x + 1$ para linguagem de programação.

```
float func (float x){
    return (x*x)+(2*x)+1;
}
```

**Quadro 6 - Função de entrada**
**Fonte: Press, Teukolsky e Vetterling (1992)**

Depois, foi aplicada a sub-rotina responsável pela regra de integração e abaixo dela o método principal. Essa rotina é responsável pela inicialização do algoritmo de integração foi completada com algumas linhas de código. As primeiras linhas adicionadas foram de um laço de repetição, com a intenção de realizar o cálculo inúmeras vezes. Depois foram adicionadas antes e depois do laço de repetição, as linhas para a contagem do tempo. Nessas linhas a variável "freq" obtêm a frequência do processador e as variáveis "start" e "stop" armazenam o inicio e o final da contagem. Com a diferença entre o inicio e o

final, dividido pela freqüência temos o tempo aproximado de execução do algoritmo

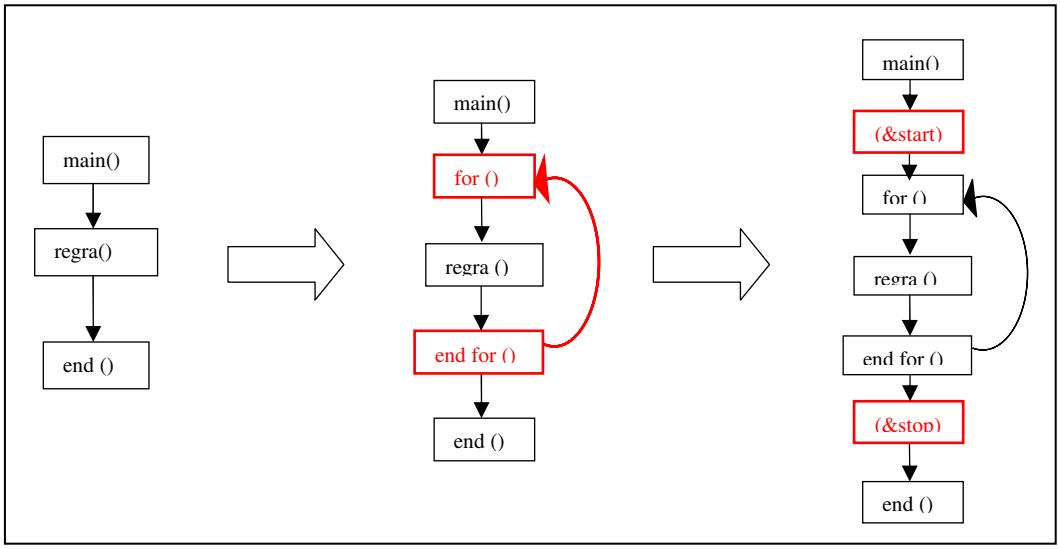


**Figura 4 – Fluxo do método principal**
**Fonte: Autoria Própria**

Para a Regra do Trapézio, o método “main()” ficou organizada conforme o quadro 7:

```
float t;
int j, vz, a=0, b=10, m=15;
__int64 freq,start,stop;
double tempo;
QueryPerformanceFrequency((LARGE_INTEGER *)&freq);
QueryPerformanceCounter((LARGE_INTEGER *)&start);
for(vz = 0; vz <1000; vz++){
    for(j=1;j<=m+1;j++)  t=trapzd(func,a,b,j);
    printf("%d: trapzd = %f\n",vz, t);
}
QueryPerformanceCounter((LARGE_INTEGER *)&stop);
tempo = ((double)stop-(double)start) / (double)freq;
printf("tempo: %f", tempo);
```

**Quadro 7 - Regra do trapézio com a contagem do tempo**
**Fonte: adaptado de Press, Teukolsky e Vetterling (1992)**

Na regra do trapézio o laço de repetição mais externo representa o numero de vezes que o cálculo deveria ser feito. Nesse caso o segundo laço é para o refinamento do resultado. Foram realizados testes com quinze e vente repetições (m=15 e m=20).

Para os outros dois algoritmos foi necessário apenas um laço de repetição no método de inicial. Mas antes do método principal foi necessário definir o valor da variável “JMAX” com a linha de código “#define JMAX 20”.

```
float t, q;
 int vz, a=0, b = 10;
 __int64 freq,start,stop;
 double tempo;
 QueryPerformanceFrequency((LARGE_INTEGER *)&freq);
 QueryPerformanceCounter((LARGE_INTEGER *)&start);
 for(vz = 0; vz <1000; vz++){
       t=qtrap(func,a,b);
       printf("%d: qtrap= %f\n",vz, t);
 }
 QueryPerformanceCounter((LARGE_INTEGER *)&stop);
 tempo = ((double)stop-(double)start) / (double)freq;
 printf("tempo: %f", tempo);
```

**Quadro 8 - Regra do trapézio estendido com a contagem do tempo**
**Fonte: adaptado de Press, Teukolsky e Vetterling (1992)**

Nos dois algoritmos acima, os programas principal são parecidos, com exceção do segundo laço de repetição na regra do trapézio simples.

```
float t;
int j, vz, a=0, b=10;
__int64 freq,start,stop;
double tempo;
QueryPerformanceFrequency((LARGE_INTEGER *)&freq);
QueryPerformanceCounter((LARGE_INTEGER *)&start);
for(vz = 0; vz <1000; vz++){
        t=qsimp(func,a,b);
        printf("%d: qsimp= %f\n",vz, t);
}
QueryPerformanceCounter((LARGE_INTEGER *)&stop);
tempo = ((double)stop-(double)start) / (double)freq;
printf("tempo: %f", tempo);
```

**Quadro 9 - Regra do Simpson com a contagem do tempo**
**Fonte: adaptado de Press, Teukolsky e Vetterling (1992)**

Depois de implementado os algoritmos, passamos para a realização de testes com os objetivos de verificar a precisão, o desempenho, estabelecer o impacto no resultado e o valor mais ideal para “m” usado na regra do trapézio e de “JMAX” utilizado nas outras duas regras.

## 4. Resultados obtidos dos algoritmos

Quando realizamos $\int_0^{10} x^2 + 2\,x + 1\,dx$ utilizando lápis e papel, conforme foi abordado no capitulo anterior, obtemos o valor 443,333333. Com a utilização dos algoritmos este é o valor esperado ou o mais próximo possível.

Os resultados que foram obtidos mostram os algoritmos mais rápidos e mais precisos. Foram realizados testes utilizando dois equipamentos diferentes para obter um resultado mais preciso. O primeiro equipamento é um laptop com processador Intel Pentium Dual-Core T3400, com 2,16 GHz, memória RAM de 2 GB 667MHz e sistema operacional Windows Vista Home Premium 32 bits. O segundo equipamento é um desktop com processador AMD Athlon 64 X2 4000+, com 2,11 GHz, memória RAM de 2 GB 800 MHz e sistema operacional Windows 7 Profissional 64 bits.

Durante essa série de testes no primeiro equipamento, a regra do trapézio foi testada com a variável “m=15” e com “m=20”. Se considerar um valor para “m” entre um e dez, será um valor baixo, e a precisão será baixa também. Mas é preciso saber o ponto em que “m” é mais preciso e o ponto em que ele começa a prejudicar o resultado. As tabelas abaixo mostram a média dos resultados do primeiro equipamento.

**Tabela 2 – Resultados com ciclos de 1 a 100 - laptop**

| **Função usada:** | (x*x)+(2*x)+1 | ***Processador: Pentium Dual-core 2 GB de RAM*** | | |
|---|---|---|---|---|
| **Valor esperado:** | 443,333333 | ***SO: Windows Vista Home Premium - 32 bits*** | | |
| **Algoritmo** | **Valor Obtido** | **Numero de Repetições do algoritmo / tempo obtido em segundos** | | |
| | | **1** | **10** | **100** |
| **Regra do Trapézio (m=20)** | 443,328186 | 0,021981 | 0,209993 | 2,133894 |
| **Regra do Trapézio (m=15)** | 443,333435 | 0,000846 | 0,007478 | 0,123434 |
| **Trapézio Composto** | 443,334015 | 0,000357 | 0,002841 | 0,109341 |
| **Regra de Simpson** | 443,333344 | 0,000487 | 0,002687 | 0,103022 |

**Fonte: Autoria Própria**

**Tabela 3 – Resultados com ciclos de 1000 a 7500 - laptop**

| **Função usada:** | (x*x)+(2*x)+1 | ***Processador: Pentium Dual-core 2 GB de RAM*** | | | |
|---|---|---|---|---|---|
| **Valor esperado:** | 443,333333 | ***SO: Windows Vista Home Premium - 32 bits*** | | | |
| **Algoritmo** | **Valor Obtido** | **Numero de Repetições do algoritmo / tempo obtido em segundos** | | | |
| | | **1000** | **5000** | **7000** | **7500** |
| **Regra do Trapézio (m=20)** | 443,328186 | 22,60986 | 109,91261 | 146,76779 | 158,195082 |
| **Regra do Trapézio (m=15)** | 443,333435 | 1,069000 | 4,973886 | 6,574480 | 7,523262 |
| **Trapézio Composto** | 443,334015 | 0,512031 | 1,610510 | 2,171756 | 2,434635 |
| **Regra de Simpson** | 443,333344 | 0,487154 | 1,609131 | 2,280298 | 2,410346 |

**Fonte: Autoria Própria**

**Tabela 4 - Resultados com ciclos de 10000 a 525000 - laptop**

| **Função usada:** | (x*x)+(2*x)+1 | ***Processador: Pentium Dual-core 2 GB de RAM*** | | | |
|---|---|---|---|---|---|
| **Valor esperado:** | 443,333333 | ***SO: Windows Vista Home Premium - 32 bits*** | | | |
| **Algoritmo** | **Valor Obtido** | **Numero de Repetições do algoritmo / tempo obtido em segundos** | | | |
| | | **10000** | **20000** | **30000** | **525000** |
| **Regra do Trapézio (m=20)** | 443,328186 | 215,98735 | 299,704884 | 433,43586 | - |
| **Regra do Trapézio (m=15)** | 443,333435 | 10,019810 | 18,412630 | 27,817042 | 489,129435 |
| **Trapézio Composto** | 443,334015 | 3,065738 | 6,152937 | 8,436001 | 154,782723 |
| **Regra de Simpson** | 443,333344 | 3,030082 | 6,006165 | 8,357705 | 150,826371 |

**Fonte: Autoria Própria**

O teste com 525 mil repetições não foi realizado para a regra do trapézio com “m=20”, pois o resultado dos outros ciclos mostra que será um valor muito alto e não afetará a análise dos resultados. O resultado fica mais expressivo através do gráfico 1.

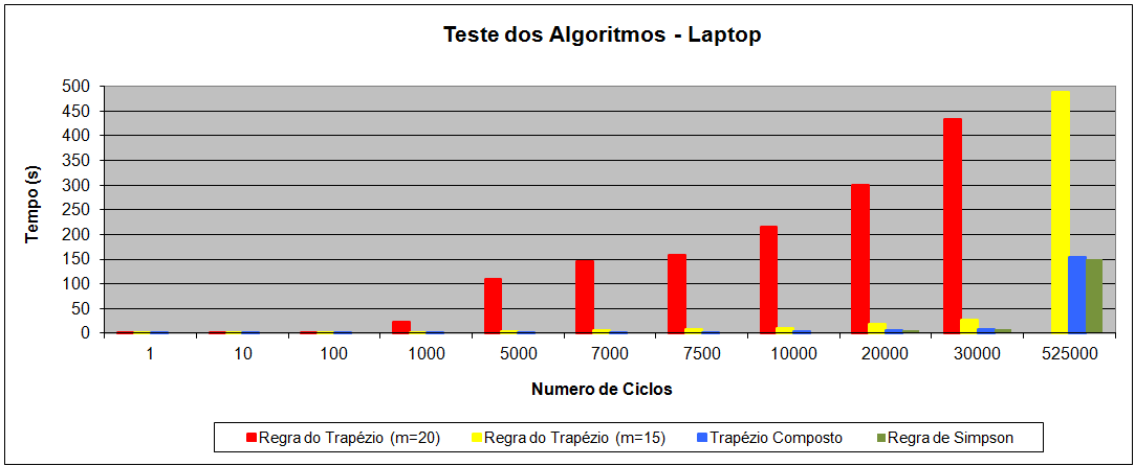


**Gráfico 1 - Teste dos Algoritmos – laptop**

**Fonte: Autoria Própria**

Com os resultados dos testes no primeiro equipamento, foi possível estipular o valor de “m=15” como melhor opção para a regra do trapézio simples.

Os testes no segundo equipamento foram realizados da mesma forma que no primeiro. Abaixo os resultados desta série de testes.

**Tabela 5 – Resultados com ciclos de 1 a 100 - desktop**

| **Função usada:** | (x*x)+(2*x)+1 | ***Processador: AMD Athlon 64 X2 2 GB de RAM*** | | |
|---|---|---|---|---|
| **Valor esperado:** | 443,333333 | ***SO: Windows 7 Profissional - 64 bits*** | | |
| **Algoritmo** | **Valor Obtido** | **Numero de Repetições do algoritmo / tempo obtido em segundos** | | |
| | | **1** | **10** | **100** |
| **Regra do Trapézio** | 443,333435 | 0,000596 | 0,004692 | 0,185067 |
| **Trapézio Composto** | 443,334015 | 0,000296 | 0,001153 | 0,162376 |
| **Regra de Simpson** | 443,333344 | 0,000247 | 0,000990 | 0,142135 |

**Fonte: Autoria Própria**

**Tabela 6 – Resultados com ciclos de 1000 a 7500 – desktop**

| **Função usada:** | (x*x)+(2*x)+1 | ***Processador: AMD Athlon 64 X2 2 GB de RAM*** | | | |
|---|---|---|---|---|---|
| **Valor esperado:** | 443,333333 | ***SO: Windows 7 Profissional - 64 bits*** | | | |
| **Algoritmo** | **Valor Obtido** | **Numero de Repetições do algoritmo / tempo obtido em segundos** | | | |
| | | **1000** | **5000** | **7000** | **7500** |
| **Regra do Trapézio** | 443,333435 | 0,951106 | 3,323739 | 4,336755 | 5,585195 |
| **Trapézio Composto** | 443,334015 | 0,753291 | 1,443778 | 2,003543 | 2,354922 |
| **Regra de Simpson** | 443,333344 | 0,581802 | 1,762205 | 1,959243 | 1,974867 |

**Fonte: Autoria Própria**

**Tabela 7 – Resultados com ciclos de 10000 a 525000 - desktop**

| **Função usada:** | (x*x)+(2*x)+1 | ***Processador: AMD Athlon 64 X2 2 GB de RAM*** | | | |
|---|---|---|---|---|---|
| **Valor esperado:** | 443,333333 | ***SO: Windows 7 Profissional - 64 bits*** | | | |
| **Algoritmo** | **Valor Obtido** | **Numero de Repetições do algoritmo / tempo obtido em segundos** | | | |
| | | **10000** | **20000** | **30000** | **525000** |
| **Regra do Trapézio** | 443,333435 | 7,650800 | 12,606424 | 18,352598 | 333,985246 |
| **Trapézio Composto** | 443,334015 | 3,268766 | 5,184294 | 7,411758 | 139,844330 |
| **Regra de Simpson** | 443,333344 | 3,065535 | 5,308277 | 7,278819 | 128,199360 |

**Fonte: Autoria Própria**

Para o segundo equipamento também foi gerado um gráfico com as três regras e os ciclos para uma melhor análise.

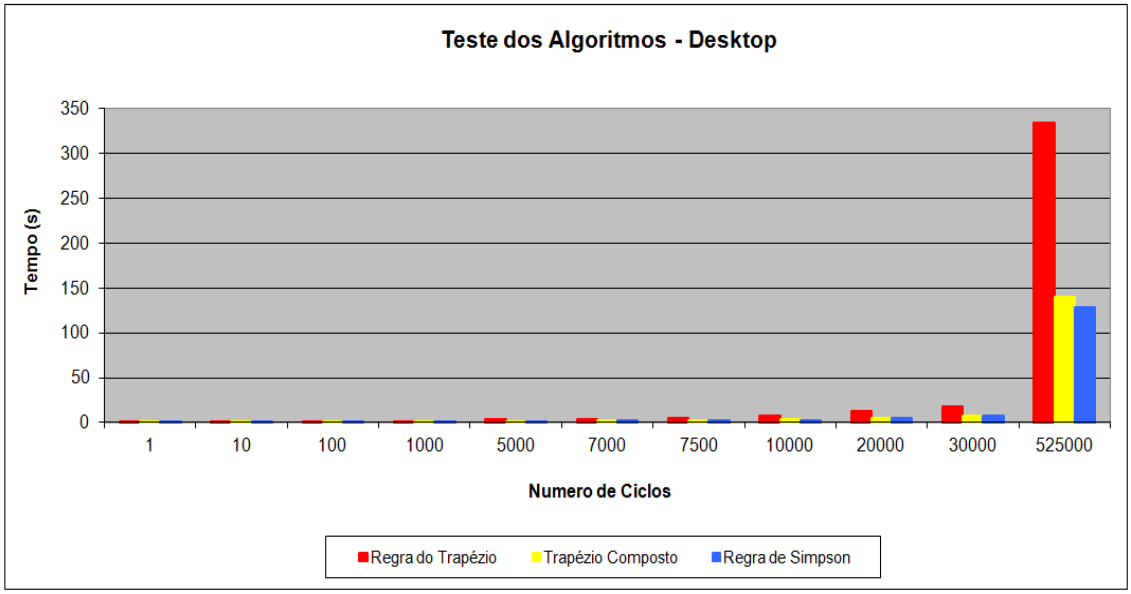


**Gráfico 2 - Teste dos Algoritmos – desktop**
**Fonte: Autoria Própria**

Analisando as tabelas e os gráficos são notáveis que a regra do trapézio simples, mesmo definindo o numero da variável “m”, tem sua precisão e velocidade muito baixa. Quando ela ultrapassa o numero de ciclos ideais, ela deixa o valor do resultado menor e impreciso, e tem um desempenho muito baixo.

Já os outros dois algoritmos, observa-se que eles tem quase a mesma velocidade, Durante os testes com números de repetição do algoritmo igual a 1 e 7000 no *laptop* e igual a 5000 e 20000 no *desktop,* regra do trapézio composto mostrou-se mais rápido que a regra de Simpson. Isso mostra a influencia do escalonador de processos do sistema operacional, mas sua precisão é inferior a da regra de Simpson.

Considerando as interferências do sistema operacional e da precisão, a regra de Simpson mostra um valor mais perto do valor esperado (Simpson = 443,333344 / Valor esperado = 443,33333333) e obteve os menores tempos de execução.

## Conclusão

Este trabalho apresentou um estudo comparativo entre três algoritmos de resolução de Integrais. Sobre eles, ficou claro que algoritmo da regra do trapézio simples mostrou ser de fácil implementação, mas sua precisão é difícil de acertar e sua velocidade mostrou muito ruim. A do trapézio composto mostrou-se mais rápida e mais precisa que a do trapézio simples, mas em relação à próxima regra, ela acaba sendo menos precisa.

A regra de Simpson mostrou que pode trazer uma precisão aceitável e um desempenho muito considerável, sendo assim a mais indicada dentre as três regras.

Também foi possível verificar na prática alguns problemas computacionais como problema de trabalhar com números binários e dizima periódica, onde a solução é trabalhar com o maior numero de casas depois da vírgula e realizar o arredondamento.

Outro problema encontrado foi a questão do gerenciamento de processos do sistema operacional e diferença que ele causa na medição de quanto tempo leva a execução de uma aplicação, mais específico, o tempo que leva para o algoritmo executar o cálculo na quantidade necessária. A solução para isso é utilizar a mesma técnica de um benchmark de computador, que calcula a média de varias execução. Muito parecido como em Física experimental que é calculado o desvio padrão de um experimento.

O trabalho também ajudou no enriquecimento do conhecimento em linguagem C, como o uso do operador de rotação (<<=), dentro da regra do trapézio, e o uso de ponteiros de função, onde era declarada a função a ser integrada.

E através dos testes e aplicando tudo o que foi citado acima foi possível realizar a comparação entre os algoritmos e observar qual obteve o melhor desempenho para que possa ser utilizado em aplicações que dependem de cálculos de integrais.

Para trabalhos futuros pode se buscar a comparação desses algoritmos com outros algoritmos de integração numérica e também avançar para outros problemas de como equação diferencial.

## Referências

CLÁDIO, Dalcidio M.; MARINS, Jussara M. **Cálculo Numérico Computacional**: Teoria e Prática. 3 ed. São Paulo: Atlas, 2000.

FLEMMING, Diva Marília; GONÇALVES, Mirian Buss. **Cálculo A**: Funções, Limite, Derivação, Integração. 5 ed. rev. e atual. São Paulo: Makron Books, 1992.

KERNIGHAN, Brian W. RITCHIE, Dennis M. **C; A linguagem de Programação.** 3 ed. Rio de Janeiro: Elsevier Editora LTDA, 1987.

PRESS, William H.; TEUKOLSKY, Saul A.; VETTERLING, William T.; FLANNERY, Brian P. **Numerical Recipes in C**: The Art of Scientific Computing. 2 ed. New York, NY – USA: Cambridge University Press, 1992.

TANENBAUM, Andrew S. **Sistemas Operacionais Modernos**, 2 ed. São Paulo: Prentice-Hall, 2003.

SILBERSCHATZ, Abraham et al. **Fundamentos de Sistemas Operacionais.** 8 ed. Rio de Janeiro: LTC, 2010.

SILVA, Renato S. ALMEIDA, Regina C. Notas de aula do Curso de Iniciação a Modelagem. Curso promovido pelo LNCC e INPA, 2003. Disponivel em: http://www.geoma.lncc.br/pdfs/integracao.pdf Acesso em: 24 set. 2011.