unesp UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
CAMPUS DE GUARATINGUETÁ

RODOLFO ARAUJO ANDRADE

Integrais e a Teoria de Monte Carlo

Guaratinguetá - SP
2016

Rodolfo Araujo Andrade

Integrais e a Teoria de Monte Carlo

Trabalho de Graduação apresentado ao Conselho de Curso de Graduação em Licenciatura em Matemática da Faculdade de Engenharia do Campus de Guaratinguetá, Universidade Estadual Paulista, como parte dos requisitos para obtenção do diploma de Graduação em Licenciatura em Matemática.

Orientador: Prof. Dr. Rafael Sfair

Guaratinguetá - SP
2016

A553i

Andrade, Rodolfo Araújo
Integrais e a Teoria de Monte Carlo / Rodolfo Araújo Andrade – Guaratinguetá, 2017.
49 f. : il.

Bibliografia: f. 35

Trabalho de Graduação em Licenciatura em Matemática – Universidade Estadual Paulista, Faculdade de Engenharia de Guaratinguetá, 2017.

Orientadora: Prof. Dr. Rafael Sfair

1. Monte Carlo, Método de. 2. Cálculos numéricos. 3. Integrais (Matemática). I. Título

CDU 51

unesp UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
CAMPUS DE GUARATINGUETÁ

RODOLFO ARAUJO ANDRADE

ESTE TRABALHO DE GRADUAÇÃO FOI JULGADO ADEQUADO COMO PARTE DO REQUISITO PARA A OBTENÇÃO DO DIPLOMA DE **"GRADUADO EM LICENCIATURA EM MATEMÁTICA"**

APROVADO EM SUA FORMA FINAL PELO CONSELHO DE CURSO DE GRADUAÇÃO EM LICENCIATURA EM MATEMÁTICA.

Profa. Dra. VIVIAN MARTINS GOMES
Coordenador

BANCA EXAMINADORA:

Prof. Dr. RAFAEL SFAIR
Orientador/UNESP-FEG

Profa. Dra. ANA PAULA MARINS CHIARADIA
UNESP-FEG

Prof. Dr. ERNESTO VIEIRA NETO
UNESP-FEG

DADOS CURRICULARES

RODOLFO ARAUJO ANDRADE

NASCIMENTO	19.02.1994 – São Paulo/ SP
FILIAÇÃO	Eduardo da Silva Andrade Lucia Ineide Araujo Lima de Andrade
2013/2016	Graduado em Licenciatura em Matemática Universidade Estadual Paulista “Júlio Mesquita Filho” - FEG

AGRADECIMENTOS

Ao meu orientador Prof. Rafael Sfair de Oliveira, pela oportunidade de trabalhar neste projeto, pela confiança, incentivo e apoio para a conclusão deste trabalho.

À minha mãe Lúcia Ineide, ao meu pai Eduardo, ao meu irmão Gabriel, aos meus avós Eulina e Francisco que mesmo com todas as muitas dificuldades que estavam enfrentando nunca deixaram de me apoiar e me incentivar, se conquistei algo aqui eles são o principal motivo.

A toda a minha grande família, o que inclui meus amigos também, a todos meus amigos/irmãos do Lauzane, ao Gersin e a Pamzinha que estiveram ao meu lado desde o primeiro ano de faculdade.

A toda minha república Ponto G que me acolheu e me fez parte dessa família.

A moradia que foi de suma importância para a minha manutenção em Guaratinguetá e aonde eu pude fazer muitos amigos, inclusive no último ano onde eu achei que já estava com a lista de amigos cheia, mas surpreendentemente consegui sobrecarregar essa lista ainda mais, por exemplo com o nome das Pris.

A todos outros professores que me proporcionaram a oportunidade de aprender algo com eles, em especial à Ana Paula Chiaradia que esteve me orientando durante esses três últimos anos da minha graduação.

A todos aqueles que contribuíram direta ou indiretamente para a realização deste trabalho.

“É necessário sempre acreditar que o sonho é possível,
Que o céu é o limite e você truta é imbatível.
Que o tempo ruim vai passar é só uma fase,
E o sofrimento alimenta mais a sua coragem.”

Racionais MC’s

RESUMO

Métodos numéricos é um assunto que desperta o interesse em todos os lugares principalmente ao utilizá-los com a intenção de calcular integrais, pois assim suas aplicações ultrapassa qualquer fronteira e alcançam um vasto território para ser trabalhado. O método escolhido por nós para ser estudado, é o chamado Método de Monte Carlo, que se constitui do uso de números aleatórios, auxiliado pelos programas Gnuplot e Devc++, foi feita uma construção do conhecimento acerca do assunto, indo passo a passo até o resultado final, o programa utilizado para fazer o cálculo das integrais utilizando o método sofreu diversas modificações ao longo do trabalho de modo a aprimorá-lo ou adequá-lo para o problema proposto, depois alguns testes foram feitos para validação do método utilizado, através de gráficos mostrou-se de forma numérica e visual a sua eficácia.

PALAVRAS-CHAVE: Métodos Numéricos. Integrais. Método de Monte Carlo. Gnuplot. DevC++.

ABSTRACT

Numerical methods are a subject that arouses interest everywhere, especially when using them with the intention of calculating integrals, so that their applications go beyond any frontier and reach a vast territory to be worked on. The method chosen by us to be studied, is called as Monte Carlo Method, which consists of the use of random numbers, aided by the programs Gnuplot and Devc ++, a construction of the knowledge about the subject was made, going step by step until the result Finally, the program used to calculate the integrals using the method has undergone several modifications throughout the work in order to improve it or adapt it to the proposed problem, after which some tests were done to validate the method used, through graphs Showed its effectiveness numerically and visually.

KEYWORDS: Numerical Methods. Integrals. Monte Carlo method. Gnuplot. DevC ++.

SUMÁRIO

1 INTRODUÇÃO

O método de Monte Carlo (MMC), o qual será estudado neste trabalho, pode ser definido como um método que calcula soluções aproximadas para um problema fazendo uso de simulações estocásticas. Segundo Angelotti (2008), "Para que uma simulação de Monte Carlo esteja presente em um estudo basta que este faça uso de números aleatórios na verificação de algum problema." (ANGELOTTI, 2008, p.2).

O conceito de simulação estocástica envolve a reprodução computacional de diversos números equiprováveis com a finalidade de análise ou solução para algum problema proposto, assim, o conceito de simulação estocástica se enquadra perfeitamente para o MMC.

Com isso vem a questão de como pode-se gerar esses números aleatórios. Uma ótima ideia seria utilizar o avanço tecnológico ao nosso favor e assim produzir computacionalmente esses números. Porém, como garantir que eles sejam realmente aleatórios, ou pelo menos, que sejam suficientemente aleatórios? O que classifica um número como aleatório ou não? Esses tópicos serão tratados no Capítulo 3 deste trabalho.

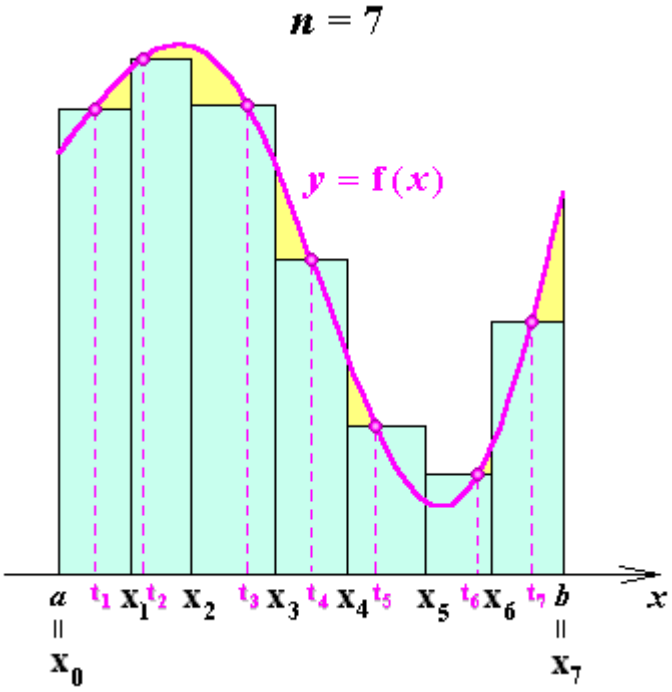
1.1 CÁLCULO DE INTEGRAIS

Integral é um assunto muito estudado nas matérias de cálculo de alguns cursos de graduação, sendo que muitos a chamam de antiderivada pois uma das formas de calculá-la é pensando no processo inverso de uma derivada. Existem diversas maneiras de se resolver uma integral e também inúmeras áreas de aplicações para elas como em engenharia, física, biomedicina, economia, etc.

A integral simples muitas vezes é associada a área da curva integrada no intervalo dado. Segundo o livro de cálculo Stewart volume 1, a definição de integral de Riemann se dá da seguinte forma:

> "Se *f* é uma função contínua definida em $a \leq x \leq b$, dividimos o intervalo *[a,b]* em *n* subintervalos de comprimentos iguais $\Delta x = (b-a)/n$. Sejam $x_0(=a), x_1, x_2, \ldots, x_n(=b)$as extremidades desses subintervalos, escolhemos os pontos amostrais $x_1^*, x_2^*, \ldots, x_n^*$nesses subintervalos, de forma que x_n^* esteja no *i*-ésimo subintervalo $[x_{i-1}, x_i\ ]$. Então a integral definida de *f* de *a* a *b* é $\int_a^b f(x)dx = \lim_{n\to\infty} \sum_{i=1}^{n} f(x_i^*)\Delta x$, desde que este limite exista. Se ele existir, dizemos que f é integrável em *[a.b]*.". (STEWART, 2010, p.345)

Figura 1 - representação da soma das áreas dos subretângulos quando n=7.



Fonte: Lúcia, R. (2016)

Quanto mais aumentamos o número *n*, mais próximos ficamos da área abaixo da curva no intervalo *[a,b]*, sendo assim, quando tendermos *n* para infinito e somarmos as áreas de todos esses retângulos formados, estaremos calculando a área abaixo da curva também e é por isso que a integral definida $\int_a^b f(x)dx$pode ser interpretada como a área de *a* até *b* sob a curva *y=f(x)* (Figura 1).

1.2 FUNÇÃO DISTRIBUIÇÃO DE PROBABILIDADE

Outro ponto importante de ser falado é a respeito das funções de distribuição de probabilidade. No início de estudo sobre probabilidades é comum começar com o cálculo para variáveis discretas, por exemplo: calcular a probabilidade de se obter duas caras em duas jogadas consecutivas de uma moeda comum. Outros tipos de problemas que se utilizam de números inteiros também são tratados como variáveis discretas, como o número tirado no dado, quantidade de pessoas em uma determinada amostra, etc.

Contudo, quando vamos tratar de variáveis contínuas muda-se bastante a forma de trabalhar com a resolução, pois agora estaremos trabalhando com valores que variam em um intervalo contido no conjunto dos números reais, como por exemplo: problemas com o tempo decorrido, altura, temperatura, distância, etc.

Assumindo que uma função *f* que descreve a probabilidade *P* de que ocorra um evento podemos definir a função densidade de probabilidade, que nos traz a probabilidade de que uma variável aleatória X esteja entre a e b. A função densidade de probabilidade é dada pela integração de *f* com os limites de integração iguais a e b:

$$P(a \leq X \leq b) = \int_a^b f(x)dx \tag{1.1}$$

Sendo assim, fica ainda mais fácil perceber que quando estamos lidando com variáveis aleatórias contínuas não iremos trabalhar com o cálculo da probabilidade onde a variável é igual a um valor exato, e sim está contida em um intervalo, pois como estamos trabalhando no conjunto dos reais então entre um número e outro sempre haverá infinitos números no meio, portanto a probabilidade de se acertar um número exato é tão próxima de zero quanto queira.

Utilizando Equação (1.1) podemos confirmar a afirmação feita acima. Para calcular a probabilidade de X ser igual a um valor *c*. Faremos:

$$P(X = c) = \int_c^c f(x)dx \tag{1.2}$$

sendo g'(x)=f(x) temos

$$P(X = c) = [g\ \ (x)]_c^c \tag{1.3}$$

logo,

$$P(X = c) = g(c) - g(c) \tag{1.4}$$

portanto

$$P(X = c) = 0 \tag{1.5}$$

Segundo Waner (2014), a função densidade de probabilidade é a função *f* definida no intervalo *(a,b)* e possui as seguintes propriedades, sendo permitido que *a* e *b* sejam infinitos:

(a) $f(x) \geq 0$ para todo *x*.

(b) $\int_a^b f(x)dx = 1$

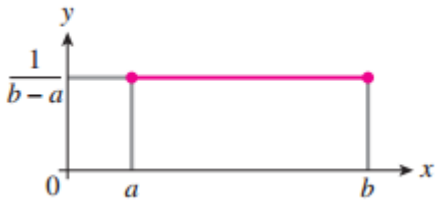
Observando o gráfico ou equação delas podemos classificá-las entre diferentes modelos, cada um com suas respectivas características e padrões, sendo da mais simples (distribuição uniforme) para algumas mais complexas (distribuições exponencial, beta e gama). A seguir listamos algumas dessas funções e seu respectivo modelo de gráfico, todos eles retirados do livro "Calculus Applied".

- Função densidade uniforme - No intervalo *(a,b)* é dado por:

$$f(x) = \frac{1}{b-a} \tag{1.6}$$

E o seu gráfico está representado na Figura 2:

Figura 2 - Figura que representa o gráfico da densidade uniforme.



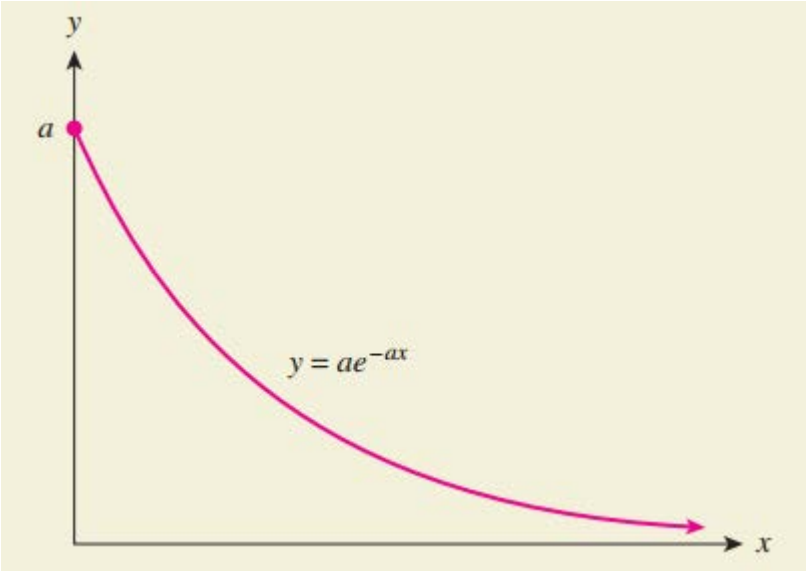
Fonte: Warner, S. (2016)

- Função densidade exponencial - é da forma

$$f(x) = ae^{-ax} \tag{1.7}$$

sendo *a* uma constante positiva. Dado o domínio *[0,+∞],* representa-se com o modelo de gráfico da Figura 3.

Figura 3 - Figura que representa o gráfico da densidade exponencial



Fonte: Warner, S. (2016)

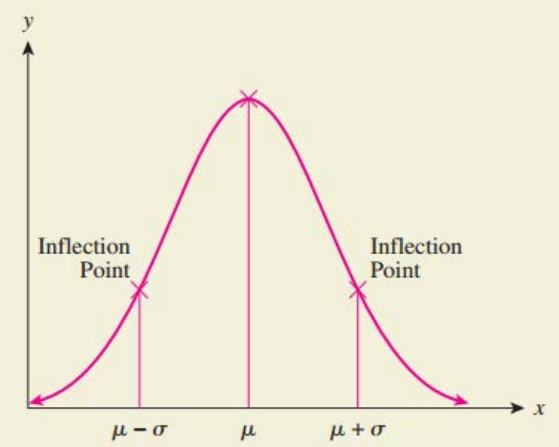
- Função densidade normal - Talvez seja a mais interessante das classes das funções, pois possui diversas propriedades aplicáveis, essa distribuição pode ser aplicada em diversas situações que se utilizam de medições e testes, entre outros motivos foi por isso que a curva de distribuição normal se tornou tão importante para os controles de qualidade, alguns exemplos de casos em que tendem a ser distribuídos normalmente

são repetidas medições imprecisas do comprimento de um único objeto, uma medição feita em muitos itens de uma linha de montagem, notas de alunos em uma prova.

Sendoμo que chamamos de média e podendo ser qualquer número real , enquanto σ é o desvio padrão e pode ser qualquer valor real estritamente positivo, sua forma (Equação 1.8) e seu gráfico (Figura 4) são do tipo:

$$f(x) = \frac{1}{\sigma\sqrt{2}} e^{\frac{(x-\mu)^2}{2\sigma^2}} \qquad (1.8)$$

Figura 4 - Gráfico da função densidade normal

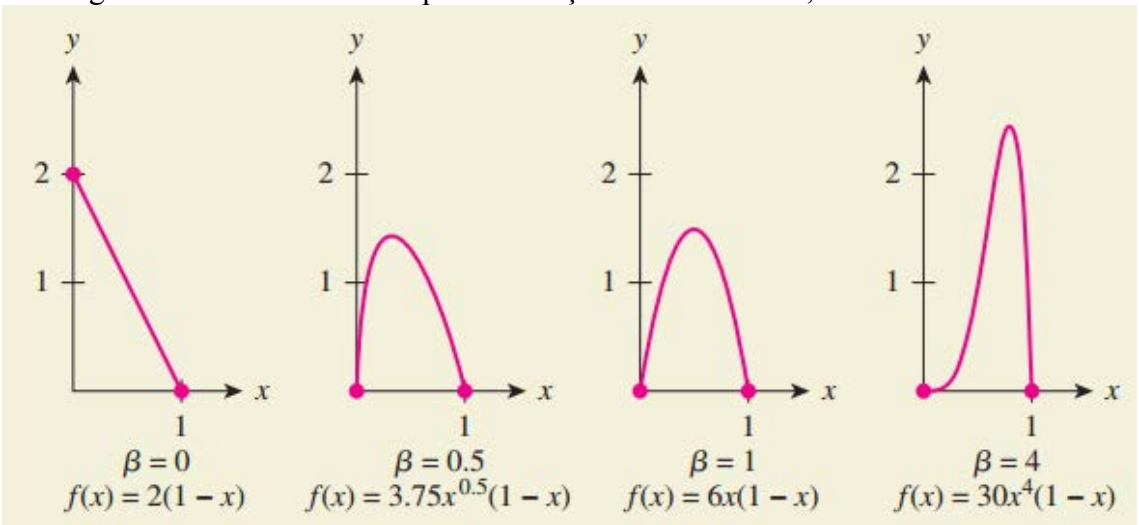


Fonte: Warner, S. (2016)

- Função densidade beta - Sendo seu domínio *[o,1]* e a constante β≥0, sua forma (Equação 1.9) e gráfico (Figura 5) são do tipo:

$$f(x) = (\beta + 1)(\beta + 2)x^{\beta}(1 - x) \qquad (1.9)$$

Figura 5 - Gráficos de exemplos da função densidade beta, variando-se o beta



Fonte: Warner, S. (2016)

1.3 O PROBLEMA

Dados *a* e *b* números reais e *f(x)* uma função, tal que *f(x)* é contínua no intervalo *[a,b]*, nosso problema gira em torno de acharmos uma boa aproximação para a integral definida $\int_a^b f(x)dx$ utilizando uma simulação de números aleatórios para tal.

Ao trabalharmos com números aleatórios equiprováveis, dizemos que as chances desse número cair em qualquer lugar dentro do espaço amostral é a mesma. Assim, gerando números aleatórios equiprováveis e comparando a quantidade de números que caem dentro de uma Região A ou em uma Região B, sendo regiões disjuntas e ambas contidas no espaço amostral, pode-se perceber uma relação direta entre essas quantidades com a área de sua respectiva Região, ou seja, quanto maior for a área dessa região, a tendência é que mais números aleatórios serão gerados dentro dela.

Levando em consideração essa proporcionalidade entre a área e os pontos contidos nela e dentro das condições propostas, ao trabalharmos com um retângulo de área *Ar* e uma curva *f* dentro dele, sendo *Aa* a área abaixo da curva contida neste retângulo, chamaremos de *count* o número de pontos dentro do retângulo que estão abaixo da curva e *i* o número total de pontos dentro dele, podemos afirmar que:

$$\frac{Aa}{Ar} = \frac{count}{i}. \tag{1.10}$$

Isolando-se a variável Aa temos:

$$Aa = Ar.(\frac{count}{i}). \tag{1.11}$$

A Equação 1.11 será de suma importância para nós, pois ao calcularmos área abaixo da função *f(x)* limitada no eixo das abscissas e pelas extremidades do retângulo, consequentemente também estaremos calculando o valor da integral da curva no intervalo *[a,b]* sendo *a* o valor de *x* no início da largura do retângulo e *b* o valor de *x* no fim, portanto:

$$Aa = \int_a^b f(x)dx. \tag{1.12}$$

Considerando a Equação 1.11 e a Equação 1.12, por transitividade temos:

$$\int_a^b f(x)dx = Ar.(\frac{count}{i}). \tag{1.13}$$

Sendo assim, agora podemos calcular a integral de um modo diferente do usual, e utilizando uma simulação estocástica com números aleatórios conseguiremos concretizar esse "experimento" de se gerar números aleatórios e depois contar quantos foram gerados ao todo

e desses quantos ficaram abaixo da função dada, para tal simulação iremos fazer uso da linguagem em C.

1.4 JUSTIFICATIVA

É indiscutível a importância das integrais, sendo elas definidas ou indefinidas, pois possuem uma vasta aplicação em inúmeras áreas, com isso, ao decorrer dos anos muito se foi estudado acerca do assunto, criando-se diversos métodos para a sua resolução, porém, usando-se apenas papel e caneta não conseguimos resolver todas as integrais (como por exemplo: a integral $\int e^{x^2} dx$), mas hoje em dia, não poderíamos deixar de utilizar o avanço tecnológico a nosso favor, e é aí que o Método de Monte Carlo entra, a fim de achar respostas para essas integrais e de aperfeiçoar os métodos anteriores para que possamos obter resultados mais precisos e com um tempo reduzido.

Se o Método de Monte Carlo fosse aplicado apenas para que, de modo prático, se possa achar uma solução numérica aproximada para as integrais definidas ele já teria uma aplicabilidade enorme. Porém o Método ainda pode ser usado em outras áreas também, como por exemplo na previsão dos custos de produção de companhias industriais, Garcia (2010), assim como em análise de incertezas, entre outras diversas aplicações em diferentes ramos "O método de Monte Carlo (MMC) é um método estatístico utilizado em simulações estocásticas com diversas aplicações em áreas como a física, matemática e biologia." (ANGELOTTI, 2008, p.01).

2 O MÉTODO

As integrais, por muitas vezes, possuem uma resolução analítica muito complicada e demorada, isso faz com que busquemos por resoluções alternativas de forma a melhorar e encurtar o processo, umas dessas formas alternativas é a resolução numérica através do Método de Monte Carlo.

O método está inserido no ramo da matemática experimental, a qual utiliza-se de experimentos com números, equações ou outros elementos matemáticos para produzir certo tipo de resultado, em algumas vezes podem até produzir a formulação de uma lei geral que regulará todas as ocorrências do fenômeno em questão. No caso de Monte Carlo faz-se experimentos com números aleatórios com a finalidade de poder tirar alguma conclusão ou aproximação para o problema em que se está trabalhando "A ideia principal por trás de Monte Carlo na abordagem desses problemas, é aproveitar ao máximo a força da análise teórica, e ao mesmo tempo evitar suas fraquezas substituindo a teoria por experimento, onde quer que a primeira falhe." (ROSA et all,2002,p 01).

A ideia básica para o cálculo de integrais pelo método de Monte Carlo é gerar um par ordenado de números aleatórios, para encontrar o que queremos basta dividir a quantidade de pontos que se encontram abaixo da curva a ser integrada pelo número total de pontos gerados e multiplicar pela área do retângulo formado pela amplitude a que os números aleatórios gerados podem chegar, Equação (1.11). Dada a ideia básica, agora vamos detalhar um pouco mais mostrando cada passo do procedimento para esse cálculo.

Pensando em um plano cartesiano, sendo a variável x representado no eixo das abscissas e a variável y no eixo das ordenadas, listamos os passos utilizados para que possamos fazer uso dos números aleatórios com o intuito de calcular a integral de uma função $f(x)$ definida em um intervalo $[a,b]$:

1. Gera-se um aleatório x, dentro intervalo (a,b).
2. Gera-se um aleatório y, de modo que ele esteja compreendido entre zero e o máximo de $f(x)$, denominado por h, no intervalo (a,b).
3. Verifica-se se o ponto (x,y) está abaixo da curva $f(x)$. Se sim, adicionamos 1 unidade a uma variável r inicialmente com o valor zero.
4. Repetem-se os três primeiros passos quantas vezes quanto queira, sendo que quanto mais repetições há mais chances de uma melhor aproximação para o valor da integral.

5. Contam-se quantos pontos *(x,y)* foram gerados. Chamemos essa quantidade de *n*.
6. O valor da integral será dado pela multiplicação da área do retângulo formado pelos números aleatórios gerados $(h \times (b - a))$ e a probabilidade do ponto *(x,y)* estar abaixo da curva $(\frac{r}{n})$:

$$\int_a^b \quad f(x)\, dx = h \times (b - a) \times \frac{r}{n}. \tag{2.1}$$

3 NÚMEROS ALEATÓRIOS

Para se gerar números aleatórios podemos utilizar algoritmos específicos chamados geradores de números aleatórios. Porém, na maioria dos casos, esses geradores se utilizam de uma semente e assim produzem números através de algum procedimento implícito em seu código. Essa semente é um número inicial, um ponto de partida para o gerador , e é através dela que o gerador produz o restante dos números. Com isso se você colocar a mesma semente, ele gera os mesmos números sempre, como há meios de predizer a sequência de números que será gerada, esses números obtidos através dos geradores ganharam o nome de números pseudo-aleatório, deixando o termo aleatório apenas para fatos que sejam realmente aleatórios, como alguns eventos naturais.

Uma forma de tentar otimizar a produção desses números pseudo-aleatórios é utilizar-se de recursos dinâmicos de seu computador, como por exemplo, o contador de tempo pois ele está em constante mudança e assim a semente sempre será diferente, ajudando-nos a melhorar a aleatoriedade dos números gerados.

Devemos nos preocupar também com a questão da distribuição dos números gerados, pois se um número que está dentro do intervalo proposto, tiver maior probabilidade de ser gerado do que outro, não nos terá valia nenhuma. Uma vez que precisamos de um gerador que nos garanta a equiprobabilidade para com os números gerados.

Quanto menor a qualidade do gerador, menor é o ciclo de repetição da sequência produzida, sendo assim, quanto maior for a sua qualidade, mais números ele conseguirá produzir aleatoriamente sem repetir nenhum.

Com isso, então por que não usamos apenas os geradores de maior qualidade sempre? Isso não ocorre pois essa qualidade do gerador é alcançada com um certo custo de tempo, portanto tudo depende da finalidade do uso de seu gerador. Pode ser que você esteja precisando de um gerador simples e mais rápido ou que queira um mais potente e não se importa em despender mais tempo. Através desse pensamento, buscamos um gerador que tenha uma qualidade razoável para nossa pesquisa e que não nos custe tanto em tempo de processamento.

3.1 ANÁLISE DO DESEMPENHO

Para encontrarmos um gerador suficientemente bom para nosso trabalho, nós utilizamo-nos de métodos já desenvolvidos previamente e que podem ser encontrados no livro

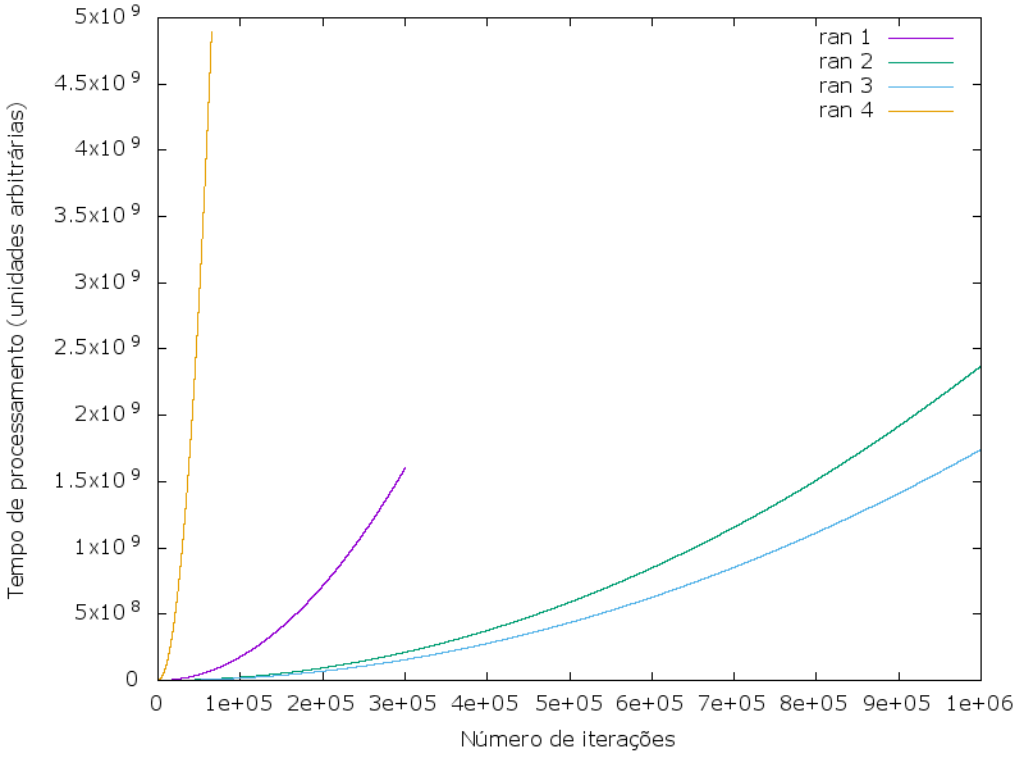
"Numerical Recipes", eles são conhecidos como: ran1, ran2, ran3 e ran4. Em geral, todos eles geram números equiprováveis no intervalo entre 0 e 1. A diferença está na qualidade e complexidade para tal feito: o ran1 é o mais simples entre eles, passando depois pelo ran2, o ran3 e por fim o ran4, que possui o código mais complexo entre esses geradores. O código de todos eles estão presentes nos anexos A, B, C e D, respectivamente.

Foi feito um gráfico (Figura 6) de modo a mostrar os tempos de cada um desses geradores em função do número de pontos gerados. Para tal feito usamos um programa em linguagem c semelhante a todos com a única diferença sendo o gerador de cada um. Esse programa tinha um contador de tempo que se iniciou no começo do código e um outro que se situava no final, para saber o tempo bastou-se subtrair o inicial do final, no meio do código haviam os procedimentos para que fizessem o cálculo de uma integral. Optamos por efetuar os testes considerando a seguinte integral

$$I = \int_0^1 x\sqrt{x}\, dx \tag{3.1}$$

que tem como solução $I = 0{,}40$.

Oo888Figura 6 - Tempo de processamento em função da quantidade de iterações para diferentes geradores de números aleatórios.



Fonte: Autoria própria

Esse gráfico nos mostra uma clara discrepância entre os geradores, trazendo a tona um dos principais motivos de não se escolher imediatamente a implementação da função ran que for mais complexa. Como se pode ver na Figura 6, o algoritmo ran4 possui um custo computacional de execução muito maior que os outros e a curva de tempo dele cresce muito mais rápido do que as das outras versões. Um ponto relevante, que a priori não era esperado, é o de que o tempo de execução do ran1 superou de forma clara e concisa os tempos do ran2 e do ran3, este último se mostrou o mais rápido entre os quatro.

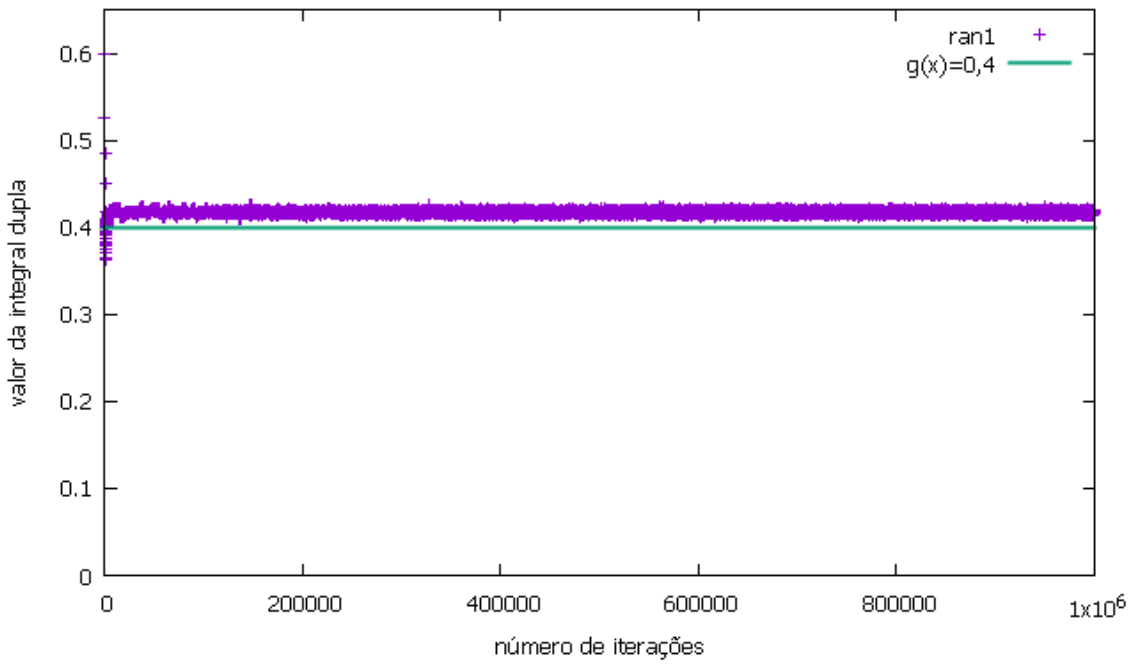
A seguir, discutiremos através da análise de testes e gráficos qual é o melhor ran para ser incluído no nosso programa, a princípio o ran3 e o ran2 nos parecem boas escolhas.

3.2 ANÁLISE DA PRECISÃO

3.2.1 RAN1

Com isso fizemos alguns testes para analisar a precisão, primeiramente temos um gráfico (Figura 7) que mostra valor de Ida Equação (3.1) conforme aumentamos o número de pontos gerados:

Figura 7 - Valor de $\int_0^1 x\sqrt{x}\,dx$em função do número de iterações feita utilizando o ran1.



Fonte: Autoria própria

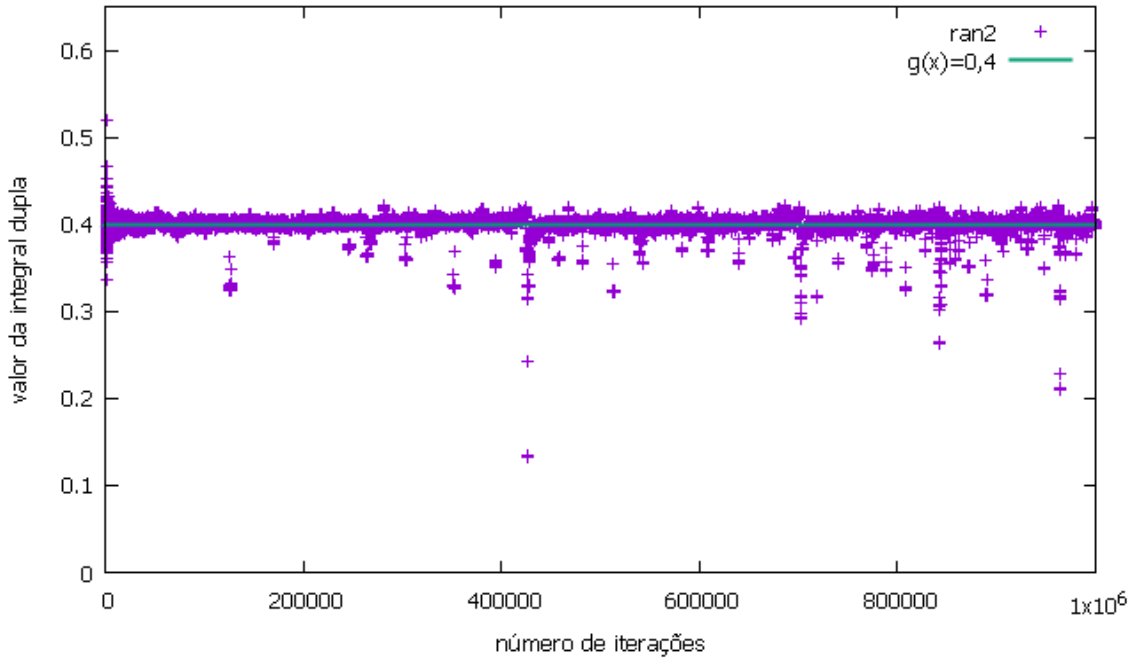
Para gerar o gráfico da Figura 7 foi feita uma tabela de pontos em que a primeira coluna representa a quantidade de números aleatórios gerados no intervalo entre 0 e 1, e a segunda coluna contém o valor final da integral definida de x*sqrt(x) com os limites de integração sendo 0 e 1, ou seja, Equação (3.1).

Foi feito um procedimento análogo para os outros rans também. Pudemos perceber que a estabilidade do ran1 é muito grande, bem no começo ele apresenta certa variação no valor da integral, porém rapidamente alcança uma boa estabilidade. O problema é que essa estabilidade está aproximadamente em torno do número 0,42 sendo que através de outros métodos podemos calcular o valor exato dessa integral que é de 0,40.

3.2.2 RAN2

Pelo teste feito com o tempos de todos os rans (Figura 6) podemos ver que o ran2 é uma boa opção no quesito de custo temporal. A seguir (Figura 8) mostraremos o número de iterações pelo valor de$I$ da Equação (3.1).

Figura 8 - Valor de $\int_0^1 x\sqrt{x}\,dx$ em função do número de iterações feita utilizando o ran2.



Fonte: Autoria própria

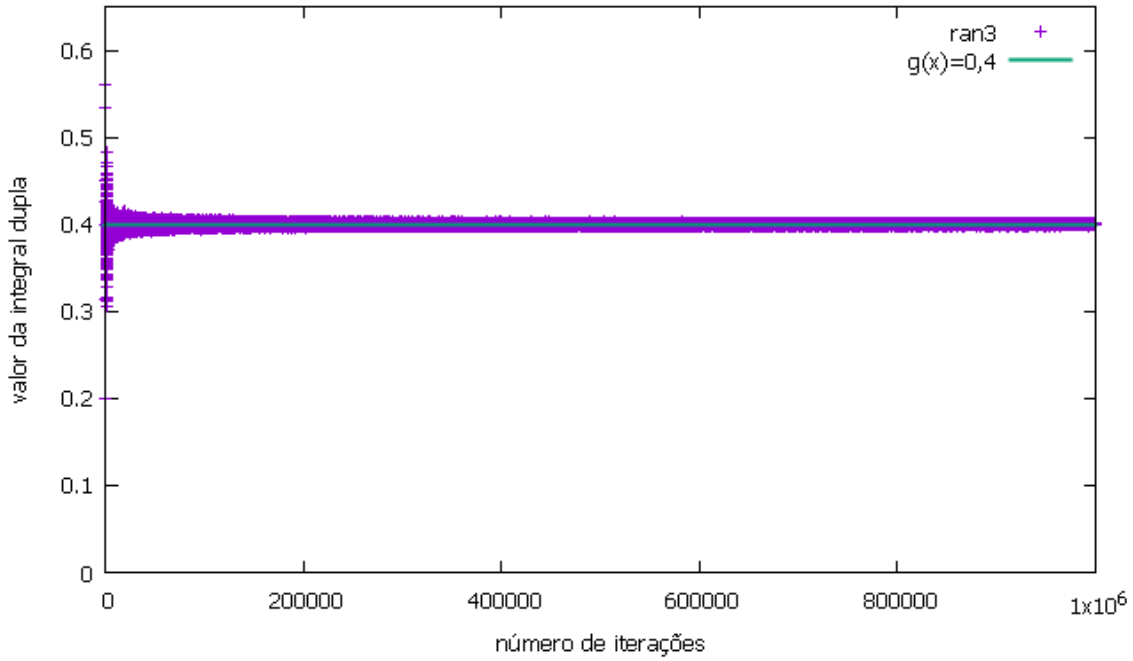
Analisando o gráfico pode-se perceber que o ran2 possui uma estabilidade menor do que a do ran1, mas a precisão é melhor, o valor da integral tende a estar em torno de 0,40. Um

ponto interessante a ser notado, que possui um efeito contrário ao que se é imaginado anteriormente aos testes, é que conforme aumentamos o número de iterações, o valor da integral ao invés de ficar mais estável, ele está tendendo a ficar mais espalhado.

3.2.3 RAN3

A seguir, na Figura 9, temos o teste de estabilidade e precisão do ran3 que até então disputa como a melhor opção junto com o ran2:

Figura 9 - Valor de $\int_0^1 x\sqrt{x}\,dx$ em função do número de iterações feita utilizando o ran3



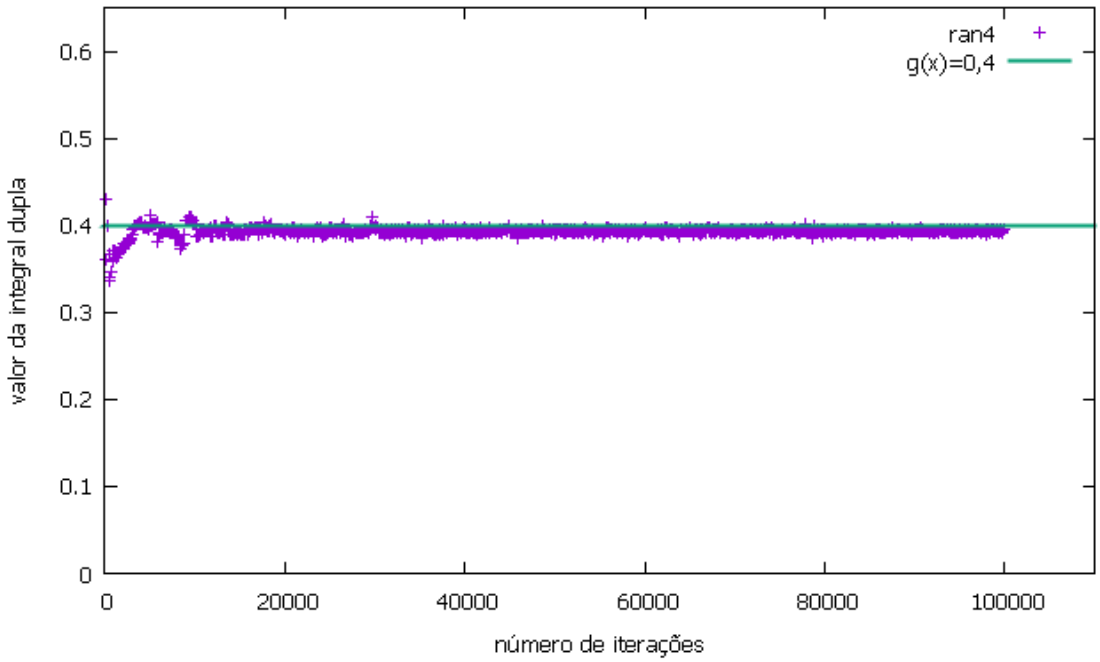
Fonte: Autoria própria

O ran3 nos mostrou um ótimo resultado. Ao chegar a duzentos mil pontos, o valor da integral já está estabilizado. Outro destaque é a sua precisão fazendo com que o valor da integral fique bem próximo à 0,40. Assim como no teste do custo temporal, nesse teste o ran3 apresentou os melhores resultados, portanto é a nossa escolha mais óbvia.

3.2.4 RAN4

Apresentamos ainda o teste do ran4 também, já que o custo de tempo dele é muito alto, ele dificilmente será escolhido por nós como o melhor gerador. A Figura 10 nos mostra o gráfico do valor de I da Equação (3.1) em função do número de iterações.

Figura 10 - Valor de $\int_0^1 x\sqrt{x}\,dx$ em função do número de iterações feita utilizando o ran4



Fonte: Autoria própria

O custo de tempo do ran4 para cem mil pontos supera o dobro do custo de ran2 e ran3 com um milhão de pontos (Figura 6), logo não foi preciso analisar sua precisão para mais pontos. Neste teste o ran4 se mostrou mediano, não apresentando nada de extraordinário que compensasse o seu alto custo temporal, com isso a nossa escolha óbvia continuou sendo o ran3, que apresentou melhores resultados em todos os testes.

4 VALIDAÇÃO

Precisamos então validar o método de cálculo de integrais. A priori escolhemos a função:

$$f(x) = \frac{1}{1-x^2} \tag{4.1}$$

que pode ser calculada através da decomposição parcial de frações. Segue abaixo uma forma de resolução da $\int_a^b \frac{1}{1-x^2}\,dx$:

Fazemos algumas manipulações algébricas na Equação (4.1) de modo a colocar-la como a soma de duas outras frações, após isso fazemos o cálculo da $\int_a^b f(x)dx$.

$$\frac{1}{1-x^2} = \frac{1}{(1-x).(1+x)} \tag{4.2}$$

$$\frac{1}{(1-x).(1+x)} = \frac{A}{(1-x)} + \frac{B}{(1+x)} \tag{4.3}$$

$$\frac{A}{(1-x)} + \frac{B}{(1+x)} = \frac{A(1+x)+B(1-x)}{(1-x).(1+x)} \tag{4.4}$$

$$\frac{A(1+x)+B(1-x)}{(1-x).(1+x)} = \frac{A+Ax+B-Bx}{(1-x).(1+x)} \tag{4.5}$$

$$\frac{A+Ax+B-Bx}{(1-x).(1+x)} = \frac{A+B+(A-B)x}{(1-x).(1+x)} \tag{4.6}$$

$$\frac{A+B+(A-B)x}{(1-x).(1+x)} = \frac{1+0.x}{(1-x).(1+x)} \tag{4.7}$$

Formamos o seguinte sistema:

$$\begin{cases} A+B=1 \\ A-b=0 \end{cases} \tag{4.9}$$

(4.10)

Isolando *A* na Equação (4.10), temos:

$$A = B \tag{4.11}$$

Substituindo a Equação (4.11) na Equação (4.9):

$$B+B = 1 \tag{4.12}$$

$$2B = 1 \tag{4.13}$$

$$B = \frac{1}{2}. \tag{4.14}$$

Substituindo *B* na Equação (4.11):

$$A = \frac{1}{2}. \tag{4.15}$$

Logo,

$$\frac{1}{1-x^2} = \frac{1}{2.(1-x)} + \frac{1}{2.(1+x)}. \tag{4.16}$$

Aplicando em ambos os lados a integral definida com limites *a* e *b*, temos:

$$\int_a^b \frac{1}{1-x^2}dx = \int_a^b \left(\frac{1}{2.(1-x)} + \frac{1}{2.(1+x)}\right)dx = \int_a^b \frac{1}{2.(1-x)}dx + \int_a^b \frac{1}{2.(1+x)}dx \tag{4.17}$$

$$\int_a^b \frac{1}{2.(1-x)}dx + \int_a^b \frac{1}{2.(1+x)}dx = \frac{1}{2}\int_a^b \frac{1}{(1-x)}dx + \frac{1}{2}\int_a^b \frac{1}{(1+x)}dx \tag{4.18}$$

$$\frac{1}{2}\int_a^b \frac{1}{(1-x)}dx + \frac{1}{2}\int_a^b \frac{1}{(1+x)}dx = \frac{1}{2}[-ln(1-x) + ln(1+x)]_b^a. \tag{4.19}$$

Assim,

$$\int_a^b \frac{1}{1-x^2}dx = \frac{1}{2}[-ln(1-x) + ln(1+x)]_b^a. \tag{4.20}$$

Como se pode ver não é tão simples chegar à Equação (4.20) de imediato. Lembrando que essa função não é definida em *x=1*, temos que tomar cuidado para não escolher limites de integração *a* e *b* de modo que *[a,b]* contenha 1, pois caso não seja respeitada essa condição, não terá como calcular a integral.

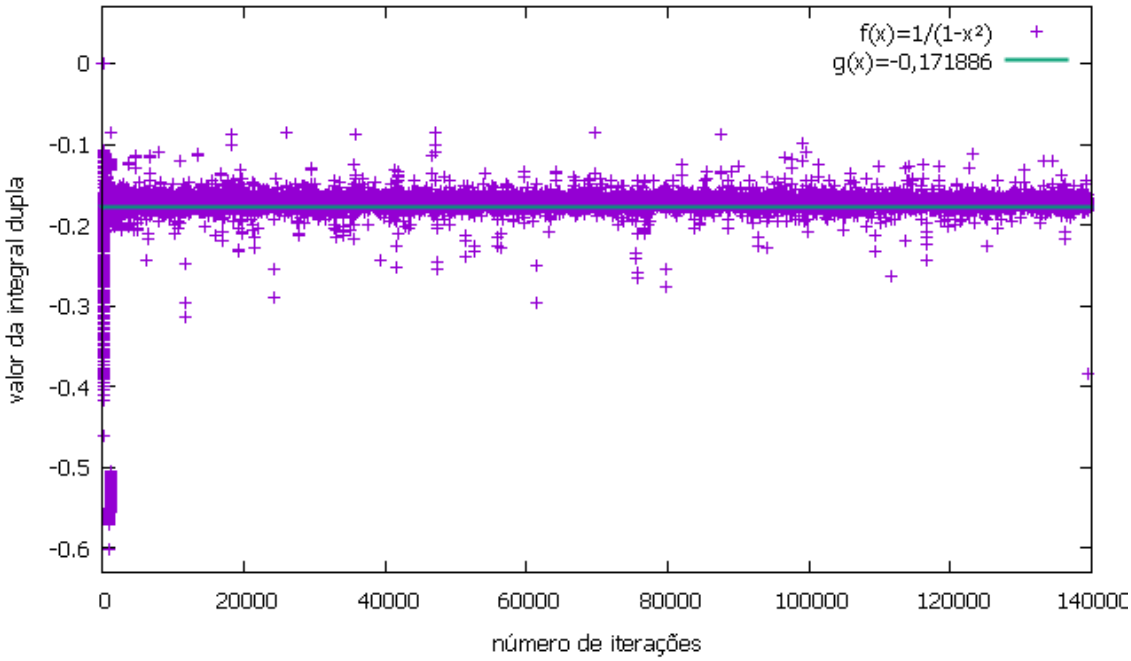
Nesse ponto decidimos avançar um pouco e fazer modificações para que o programa também possa resolver integrais em que *f(x)<0*. Assim aproveitando esse teste resolvemos incluir um diferencial a mais a ele, escolhemos *a* e *b* de modo que o resultado final da integral seja negativo e assim poderemos ver se o programa também funciona para o caso de *f(x)<0*. Dentro dessas condições decidimos arbitrariamente por *a=2,3* e *b=3,5*. A integral a ser testada é:

$$\int_{2,3}^{3,5} \frac{1}{1-x^2}\, dx, \tag{4.21}$$

que tem como solução o valor de -0,17188.

Através do nosso programa decidimos por montar um gráfico (Figura 11), assim como nos testes dos rans, mostrando a evolução do valor da integral pelo aumento do número de pontos aleatórios gerados, veja a figura a seguir:

Figura 11 - Valor de $\int_{2,3}^{3,5} \frac{1}{1-x^2}\, dx$ em função do número de iterações



Fonte: Autoria própria

A Figura 11 nos mostra que o valor da integral calculado pelo programa é uma boa aproximação para $\int_{2,3}^{3,5} \frac{1}{1-x^2}\, dx$ e se mostra eficiente no cálculo de integral quando *f(x)<0*. Esse gráfico foi gerado até um número máximo de aproximadamente cento e quarenta mil pontos apenas, o que já deu para se ter uma estabilidade boa acerca do valor real da integral. O programa utilizado para produzir os dados que auxiliaram a produção do gráfico da Figura 11 se encontra no Apêndice A.

Já que o teste utilizando a Equação (4.1) nos deu um bom resultado, tentaremos agora com uma função um pouco mais complicada e escolhemos a função gaussiana $y = e^{-x^2}$. Essa função está intimamente ligada às funções erro, "erf(x)" e "erfc(x)", e a integral gaussiana, sendo essa última definida como:

$$\int_{-\infty}^{+\infty} e^{-x^2} dx, \tag{4.22}$$

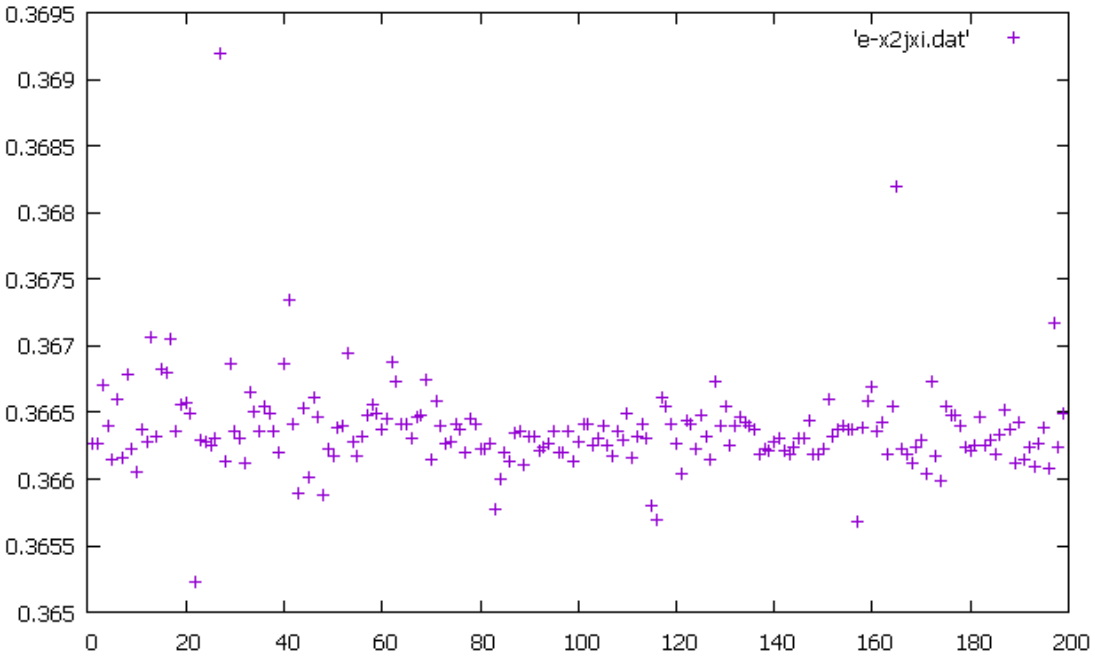
cujo resultado será $\sqrt{\pi}$.

Sendo assim, a função escolhida possui diversas aplicações, principalmente nas áreas de física e estatística. Além disso, a integral definida dessa função não é resolvida através de métodos tradicionais, necessitando do auxílio do computador para a resolução. Para a validação dos nossos resultados, utilizaremos integradores já conhecidos e confiáveis como o Wolfram Alpha, os limites de integração foram escolhidos arbitrariamente sendo eles 0,3 e 0,8. Desta forma nossa integral será:

$$\int_{0,3}^{0,8} e^{-x^2} dx \tag{4.23}$$

Para tal, o valor máximo possível para os números aleatórios gerados será de 3, ou seja, nosso retângulo terá base 0,5 e altura 3. Como já fizemos o valor da integral em função do número de pontos, é interessante para nós agora fixarmos o número de pontos em algum que esteja suficientemente bom e calcularmos diversas vezes para vermos o valor médio da integral. Pensando nisso escolhemos calcular duzentas vezes com um milhão de pontos. A cada rodada foi registrada o valor da integral, com esses dados fizemos a dispersão dos pontos em um gráfico (Figura 12) e após isso calculamos o valor médio entre os valores da integral e obtivemos 0,36637 como resposta, sendo que o valor calculado no Wolfram Alpha para a mesma integral com os mesmos limites de integração foi de 0,36643.

Figura 12 - Dispersão do valor de $\int_{0,3}^{0,8} e^{-x^2} dx$ em função da quantidade de vezes calculada



Fonte: Autoria própria

Comparando o nosso resultado com o do wolfram alpha, temos uma variação de apenas 0,016% aproximadamente, o que mostra que nosso método e programa possuem uma boa aproximação, validando o resultado obtido.

5 INTEGRAIS DUPLAS

5.1 CÁLCULO DE VOLUMES

Dando um passo um pouco mais além, decidimos por nos aventurar com o cálculo das integrais duplas, então algumas outras adaptações tiveram que ser feitas (ver o programa no Apêndice B). De toda forma, a ideia principal continua a mesma, com a diferença de que estaremos trabalhando com um gráfico em três dimensões. Sendo assim, será preciso que tenhamos um ponto no espaço *(x, y, z)* ao invés de um ponto no plano *(x,y)* e portanto precisaremos de três aleatórios a cada rodada (um para cada variável). Depois basta fazer a verificação para cada um deles e calcular a integral multiplicando a área do paralelepípedo formado pela probabilidade do ponto estar abaixo da curva, ou seja:

1. Gera-se um aleatório *x*, dentro do intervalo *(a,b)*.
2. Gera-se um aleatório *y*, dentro do intervalo *(c,d)*.
3. Gera-se um aleatório *z*, de modo que ele esteja compreendido entre zero e o máximo de *f(x,y)*. Chamemo-lo de *h*, no intervalo *[a,b]*.
4. Verifica-se se o ponto *(x,y,z)* está abaixo da curva *f(x,y)*. Se sim, adicionamos 1 unidade a uma variável *r* inicialmente com o valor zero.
5. Repetem-se os quatro primeiros passos quantas vezes quanto queira, sendo que quanto mais repetições há mais chances de uma melhor aproximação para o valor da integral.
6. Contam-se quantos pontos *(x,y,z)* foram gerados, chamemos essa quantidade de *n*.

O valor da integral dupla $\int_a^b \int_c^d f(x,y)dydx$ será dado pela multiplicação do volume do paralelepípedo formado pelos números aleatórios gerados $(b-a) \times (d-c) \times h$ e a probabilidade do ponto *(x,y,z)* estar abaixo da superfície $\frac{r}{n}$, logo:

$$\int_a^b \int_c^d f(x,y)dydx = (b-a).(d-c).h.(\frac{r}{n}) \tag{5.1}$$

Como *a, b, c* e *d* são dados (limites de integração), *r* e *n* são respectivamente o número de pontos que estão abaixo da curva e o número de pontos totais que foram gerados, e *h* é fixado desde que $h \geq f(x,y), \forall x \in]a,b[\,, \forall y \in]c,d[$. Sendo assim, após a simulação

teremos o valor de todas as incógnitas do lado direito da Equação 5.1, tornando possível calcularmos o valor de $\int_a^b \int_c^d f(x,y)dydx$.

5.1.1 Validação para o Cálculo de Volumes

Idem à integral simples, devemos fazer a validação dos resultados para a integral dupla. Escolhemos

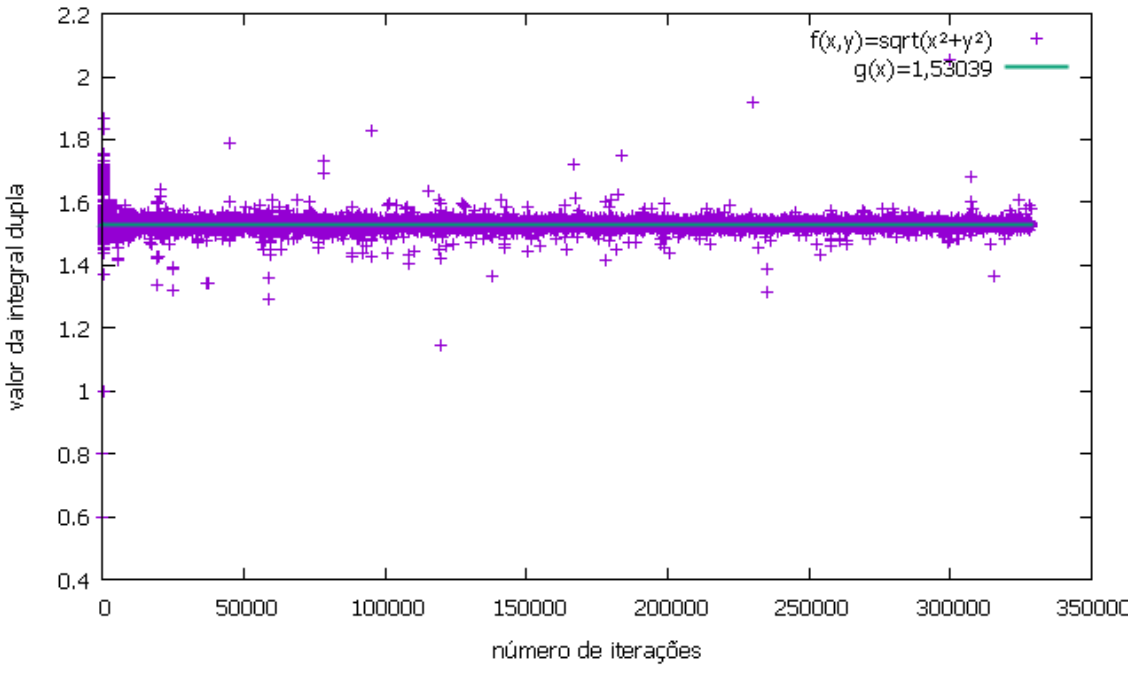
$$f(x,y) = \sqrt{x^2 + y^2}, \tag{5.2}$$

com *a=-1*, *b=1*, *c=0* e *d=1*, portanto a integral a ser calculada é

$$\int_{-1}^{1} \int_{0}^{1} \sqrt{x^2 + y^2}dydx. \tag{5.3}$$

Fazendo novamente o gráfico do valor da integral em função do número de pontos aleatórios gerados, temos a Figura 13:

Figura 13 - Gráfico do valor de $\int_{-1}^{1} \int_{0}^{1} \sqrt{x^2 + y^2}dydx$ em função do número de iterações



Fonte: Autoria própria

Segundo o Wolfram Alpha, o resultado correto é 1,5304 e devido a isso plotamos uma função *g(x)=1,5304* no mesmo gráfico (Figura 13) para fazer a comparação dos resultados.

Pela Figura 13 ficou nítido que nosso programa traz boas aproximações para a integral dupla proposta, mantendo o mesmo padrão de quando resolvemos integrais simples.

5.2 CÁLCULO DE ÁREAS

Segundo Stewart (2006), também podemos interpretar as integrais duplas sobre uma região *D* como sendo a área *A* dessa região. Essa interpretação ocorre quando

$$f(x,y)=1. \tag{5.4}$$

Sendo assim temos:

$$\int \quad \int_D \quad dydx = A \tag{5.5}$$

O programa utilizado para fazer esse cálculo se encontra no Apêndice C e a seguir, na Seção 5.2.1, faremos a resolução de dois problemas propostos que se baseiam no cálculo de áreas através de integrais duplas. Para essa resolução continuaremos utilizando o método de Monte Carlo.

5.2.1 Validação para o Cálculo de Áreas

1. Determinar a área da região limitada pelas curvas $y = x^3$ e $y = 4x$ no 1º Quadrante.

Resolução: Para determinarmos os limites de integração devemos, primeiramente, achar onde as duas equações se encontram, para isso vamos substituir o *y* de uma na outra:

$$x^3 = 4x \tag{5.6}$$

$$x^3 - 4x = 0. \tag{5.7}$$

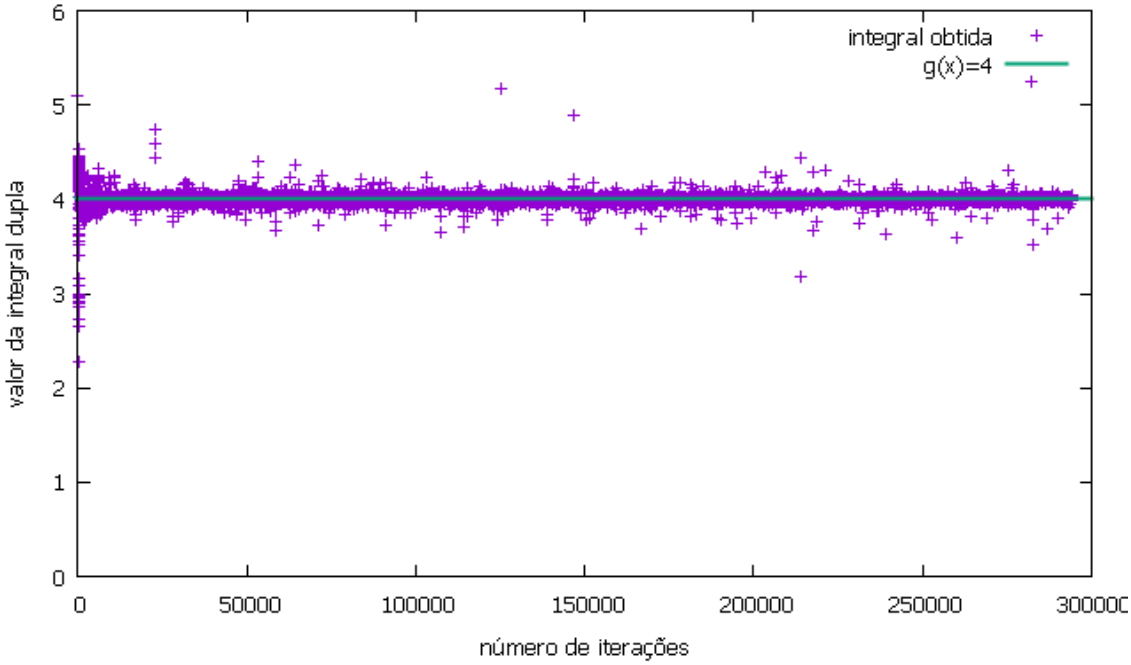
Cujas raízes são *0*, *2* e *-2*, como estamos tratando apenas do 1º Quadrante então temos que nosso *x* varia de *0* até *2*, ou seja, $0 \leq x \leq 2$, e como nosso *y* deve estar entre as curvas, então $x^3 \leq y \leq 4x$. A área *A* delimitadas pelas curvas $y = x^3$ e $y = 4x$ será dada pela integral dupla

$$\int_0^2 \quad \int_{x^3}^{4x} \quad dydx. \tag{5.8}$$

Vamos entrar com os valores dos limites de integração no nosso programa e fazer o cálculo dessa integral dupla através de Monte Carlo.

A Figura 14 mostra o gráfico gerado pelos dados do nosso programa.

Figura 14 - Gráfico do valor da integral dupla $\int_0^2 \int_{x^3}^{4x} dydx$em função do número de iterações



Fonte: Autoria própria

Percebe-se pela Figura 14 que o programa calculou boas aproximações para a integral dupla da Equação (5.8), validando tanto o método, quanto o método utilizado neste exercício.

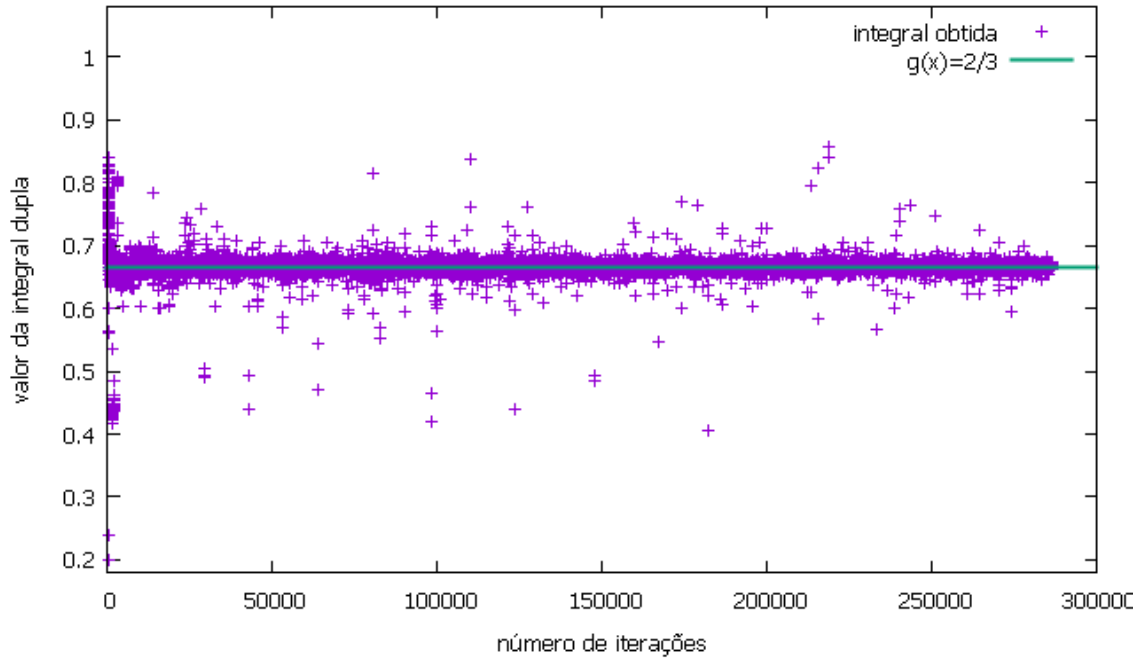
2. Determinar a área da região limitada pelas curvas $y = \sqrt{2x}$ e $y = x$no 1º Quadrante.

Resolução: Fazendo-se um procedimento análogo ao item anterior, obteremos as raízes 0 e 2, e novamente $0 \leq x \leq 2$. Como queremos a região limitada pelas curvas $y = \sqrt{2x}$ e $y = x$, então $x \leq y \leq \sqrt{2x}$, logo a área desejada é dada por

$$\int_0^2 \int_x^{\sqrt{2x}} dydx. \tag{5.9}$$

Ao calcularmos esta integral no nosso programa, foi possível fazer o gráfico da Figura 15:

Figura 15 - Gráfico do valor de $\int_0^2 \int_x^{\sqrt{2x}} dydx$ em função do número de iterações.



Fonte: Autoria própria

Percebe-se que o programa trouxe boas aproximações para a integral dupla da Equação (5.9), porém aparentemente o programa parece demorar um pouco mais para ficar com uma boa estabilidade, necessitando então de um maior número de iterações.

6 CONCLUSÃO

Neste trabalho analisamos um método numérico para estimar o valor de integrais. Para isso usamos simulações de números pseudo-aleatórios que nos traziam a probabilidade de que esses números estivessem em uma região proposta. Fizemos uso da linguagem em C para realizar essas simulações. Obtivemos bons resultados, validando o método utilizado.

Temos várias desvantagens e melhorias a serem feitas, porém atualmente alcançamos uma boa aproximação para as integrais definidas calculadas. Uma das desvantagens é que para que mantenha a qualidade da aproximação, quanto maior for o retângulo ou paralelepípedo máximo formado pelos números aleatórios gerados, maior terá que ser a quantidade desses números.

Uma melhoria a ser feita em trabalhos futuros é que o programa procure o ponto máximo dentro do intervalo de integração e o faça ser a altura h, de modo a otimizar e manter nosso espaço amostral o menor possível.

Como o método que utilizamos se baseia apenas com números aleatórios e se eles estão ou não abaixo de uma função a qual se quer integrar, desde que essa função seja contínua no intervalo proposto, não há problemas quanto a função a ser integrada, ela pode ser quão complicada quanto se queira que o método funciona da mesma forma, ele não faz distinção entre funções fáceis ou difíceis, basta apenas que ela tenha nenhuma indeterminação no intervalo dado.

REFERÊNCIAS

COSTA NETO, P. L. de O. **Estatística.** São Paulo: Edgard Blucher, 2002.

COSTA NETO, P. L. de O.; CYMBALISTA, M. **Probabilidades.** 2. ed. São Paulo: Edgard Blucher, 2005.

PRESS, W. H. **Numerical recipes in Fortran 77**. Volume I, 2. ed. Melbourne: Press Syndicate of the University of Cambridge, 1997.

STEWART, J. **Cálculo**. 6.ed. São Paulo: Cengage Learning, 2010. Volume I.

STEWART, J. **Cálculo**. 5.ed. São Paulo: Thomson Learning, 2007. Volume II.

BUFFONI, S. S. O. **Integral dupla:** exercícios resolvidos. Volta Redonda: Universidade Federal de Fluminense. Disponível em: http://www.professores.uff.br/salete/cdiii/Calculo21.pdf. Acesso em: 31/11/2016. (apostila)

WANER, S. **Applied calculus:** Calculus applied to probability and statistics. Disponível em: http://www.cengage.com/resource_uploads/downloads/1439049254_242719.pdf. Acesso em: 25/10/2016.

ANGELOTTI, W. **Uma abordagem simplificada do método Monte Carlo quântico.** Campinas: Universidade Estadual de Campinas, 2008.

APÊNDICE

APÊNDICE A – CÁLCULO DE ÁREAS PELA INTEGRAL SIMPLES

```
1.  // Programa para testar o calculo de integrais
2.  // atraves do metodo de montecarlo
3.  //
4.  //  by sfair and Rodolfo (110416.1556)
5.
6.  #include <stdio.h>
7.  #include <math.h>
8.  #include <stdlib.h>
9.  #include <time.h>
10. #include "ran3.h"
11.
12. // funcao que vai ser integrada
13. double fx(double x)
14. {
15.     return (1/(1-x*x)) ;
16. }
17.
18. int main()
19. {
20.     unsigned long int   i,j ; // contadores
21.     float   count           ;
22.     float   a, b            ; // intervalo de integracao
23.     double  x               ; // variavel independente
24.     double  y               ; // variavel dependente
25.     float   h=2.0           ; // altura do retângulo
26.     double  v               ;
27.     static long idum            ; // random seed
28.
29.     FILE    *out                ; // arquivo para guardar os pontos
30.
31.     // ---------------------------------------------------------------------
32.     // abrindo arquivos
33.     out = fopen("1(1-xx)nxi.dat","w") ;
34.
35.     // ---------------------------------------------------------------------
36.     // intervalo de integracao
37.
38.     a = 2.3 ;
39.     b = 3.5 ;
40.
41.
42.     count = 0.0 ;
43.
44.     // ---------------------------------------------------------------------
45.     for (j=2;j<1e6;j++){
46.     count = 0.0 ;
47.     for (i=1; i < j; i++)
48.     {
49.         // aqui escolhe quem eh aleatorio
50.         idum = ((i)*time(NULL) - time(NULL))     ;
51.
52.
```

```
53.         x = (b-a) * ran3(&idum) +   a       ;
54.         y = h * (2 * ran3(&idum) -1 )               ;
55.
56.
57.
58.         if(x >= a && x <= b){           // x esta no intervalo de integracao?
59.             if (fx(x) < 0){         // a função é negativa?
60.                 if (y >= fx(x) && y <= 0) //y está entre 0 e a função?
61.                 {
62.                     count--     ;
63.                 }
64.             }
65.             if (fx(x) >= 0){                // a função é positiva?
66.             if (y>=0 && y <= fx(x))         //y está entre 0 e a função?
67.             {
68.                 count++             ;
69.             }
70.             }
71.             }
72.         }
73.         v=(2*h*(b-a))*(1.0*count)/i       ;     //área do retângulo vezes a
    probabilidade
74.         fprintf(out,"%d %f \n",j,v) ;
75. }
76.     v=(2*h*(b-a))*(1.0*count)/i          ;      //área do retângulo vezes a
    probabilidade
77.
78.     printf("integral = %1.6f \n", v) ;
79.
80.
81.     // ---------------------------------------------------------------------
82.     // fechando arquivos
83.     fclose(out) ;
84.
85.
86.     return 0    ;
87. }
```

APÊNDICE B – CÁLCULO DE VOLUME

```
1.  // Programa para testar o calculo de integrais duplas
2.  // atraves do metodo de montecarlo
3.  //
4.  //  by sfair and Rodolfo (110416.1556)
5.  // os comandos que estiverem como comentários,
6.  // foram utilizados como complementos para que fossem feitos os testes
7.  #include <stdio.h>
8.  #include <math.h>
9.  #include <stdlib.h>
10. #include <time.h>
11. #include "ran3.h"
12.
13. // funcao que vai ser integrada
14. double fxy(double x, double y)
15. {
16.     return (x+y*2) ;
17. }
18.
19. int main()
20. {
21.     unsigned long int   i,j ; // contadores
22.     float   count           ;
23.     float   a, b, c, d      ; // intervalo de integracao
24.     double  x               ; // variavel independente
25.     double  y               ; // variavel independente
26.     double  z               ; // variavel dependente
27.     float   h=4.0           ; // altura do paralelepípedo
28.     double  v               ; // valor final da integral
29.     static long idum        ; // random seed
30.
31.     FILE    *out            ; // arquivo para guardar os pontos
32.
33.     // ------------------------------------------------------------------
34.     // abrindo arquivos
35.     out = fopen("sqrt(xx+yy)nxi.dat","w") ;
36.
37.     // ------------------------------------------------------------------
38.     // intervalo de integracao
39.
40.     a = -1.0    ;
41.     b = 1.0 ;
42.     c = 0.0 ;
43.     d = 2.0 ;
44.
45.     count = 0.0 ;
46.
47.     // ------------------------------------------------------------------
48. //  for (j=10; j<1e6; ){
49. //  count = 0.0 ;
50.     for (i=1; i < 1e6; i++)
51.     {
52.         // aqui escolhe quem eh aleatorio
53.         idum = ((i)*time(NULL) - time(NULL))     ;
54.
55.
```

```
56.         x = (b-a) * ran3(&idum) +   a       ;
57.         y = (d-c) * ran3(&idum) +   c       ;
58.         z = h * (2 * ran3(&idum) -1 )       ;
59.
60.
61.
62.         if(x >= a && x <= b){           // x esta no intervalo de integracao?
63.         if(y >= c && y <= d){           // y esta no intervalo de integracao?
64.             if (fxy(x,y) < 0){          // a função é negativa?
65.                 if (z >= fxy(x,y) && z <= 0) //z está entre 0 e a função?
66.                 {
67.                     count--     ;
68.                 }
69.             }
70.             if (fxy(x,y) >= 0){             // a função é positiva?
71.             if (z>=0 && z <= fxy(x,y))          //z está entre 0 e a função?
72.                 {
73. //              fprintf(out,"%f %f \n",x,y) ;
74.                 count++             ;
75.                 }
76.             }
77.         }
78.         }
79.     }
80. //       v=(2*h*(b-a)*(d-c))*(1.0*count)/i    ;    //área do retângulo vezes a
    probabilidade
81. //      fprintf(out,"%d %f \n",j,v) ;
82. //      j=j+5;
83. //}
84.     v=(2*h*(b-a)*(d-c))*(1.0*count)/i    ;    //área do paralelepípedo vezes a
    probabilidade
85.
86.
87.
88.     printf("integral = %1.6f \n", v/h) ;
89.
90.
91.     // ------------------------------------------------------------------
92.     // fechando arquivos
93.     fclose(out) ;
94.
95.
96.     return 0    ;
97. }
```

APÊNDICE C - CÁLCULO DE ÁREAS POR INTEGRAIS DUPLAS

```
1.  // Programa para testar o calculo de áreas através de integrais  duplas
2.  // atraves do metodo de montecarlo
3.  //
4.  //  by sfair and Rodolfo (110416.1556)
5.
6.  #include <stdio.h>
7.  #include <math.h>
8.  #include <stdlib.h>
9.  #include <time.h>
10. #include "ran3.h"
11.
12. // funcao que vai ser integrada
13. /*double fxy(double x, double y)
14. {
15.     return (x+y*2) ;
16. }
17. */
18. double fx1(double x) // a menor função do lmite de integração de y
19. {
20.     return x ;
21. }
22. double fx2(double x) // a maior função do limite de integração de y
23. {
24.     return sqrt(x*2) ;
25. }
26.
27. int main()
28. {
29.     unsigned long int   i,j ; // contadores
30.     float   count           ;
31.     float   a, b, c, d      ; // intervalo de integracao
32.     double  x               ; // variavel independente
33.     double  y               ; // variavel independente
34.     double  z               ; // variavel dependente
35.     float   h=0.0           ; // altura do paralelepípedo
36.     double  v               ; // valor final da integral
37.     static long idum        ; // random seed
38.
39.     FILE    *out            ; // arquivo para guardar os pontos
40.
41.     // ------------------------------------------------------------------
42.     // abrindo arquivos
43.     out = fopen("sqrt(xx+yy)nxi.dat","w") ;
44.
45.     // ------------------------------------------------------------------
46.     // intervalo de integracao
47.
48.     a = 0.0 ;
49.     b = 2.0 ;
50.     c = 0.0 ;
51.     d = 3.0 ;
52.
53.     count = 0.0 ;
54.
55.     // ------------------------------------------------------------------
```

```
56. //  for (j=10; j<1e6; ){
57. //  count = 0.0 ;
58.     for (i=1; i < 1e6; i++)
59.     {
60.         // aqui escolhe quem eh aleatorio
61.         idum = ((i)*time(NULL) - time(NULL))     ;
62.
63.
64.         x = (b-a) * ran3(&idum) +   a       ;
65.         y = (d-c) * ran3(&idum) +   c       ;
66. //      z = h * (2 * ran3(&idum) -1 )       ;
67.
68.
69.
70.         if(x >= a && x <= b){           // x esta no intervalo de integracao?
71.         if(y >= fx1(x)  && y <= fx2(x)){           // y esta no intervalo de
    integracao?
72. //          if (fxy(x,y) < 0){          // a função é negativa?
73. //              if (z >= fxy(x,y) && z <= 0) //z está entre 0 e a função?
74. //              {
75. //                  count--     ;
76. //              }
77. //          }
78. //          if (fxy(x,y) >= 0){             // a função é positiva?
79. //          if (z>=0 && z <= fxy(x,y))          //z está entre 0 e a função?
80. //              {
81. //              fprintf(out,"%f %f \n",x,y) ;
82.                 count++             ;
83.                 }
84.             }
85.         }
86. //      }
87. //  }
88. //       v=(2*h*(b-a)*(d-c))*(1.0*count)/i    ;    //área do retângulo vezes a
    probabilidade
89. //      fprintf(out,"%d %f \n",j,v) ;
90. //      j=j+5;
91. //}
92.     v=((b-a)*(d-c))*(1.0*count)/i     ;     //área do paralelepípedo vezes a
    probabilidade
93.
94.
95.
96.     printf("integral = %1.6f \n", v) ;
97.
98.
99.     // ---------------------------------------------------------------------
100.         // fechando arquivos
101.         fclose(out) ;
102.
103.
104.         return 0    ;
105.     }
```

ANEXOS

ANEXO A – RAN1

```
1.       /*************************************************************
2.  This is the random number generator ran1() from "Numerical Recipes
3.  in C"
4.  *************************************************************/
5.
6.  #define IA 16807
7.  #define IM 2147483647
8.  #define AM (1.0/IM)
9.  #define IQ 127773
10. #define IR 2836
11. #define NTAB 32
12. #define NDIV (1+(IM-1)/NTAB)
13. #define EPS 1.2e-7
14. #define RNMX (1.0-EPS)
15.
16. float ran1(long *idum)
17. /* Minimum random number generator of Park an Miller with Bays-Durham
18. shuffle and added safeguards.  Returns uniform random deviate between
19. 0.0 and 1.0 (exclusive of the endpoint values).  Call with idum a
20. negative integer to initialize; thereafter, do not alter idum between
21. successive deviates in a sequence.  RNMX should approximate the largest
22. floating value that is less than 1. */
23. {
24.      int j;
25.      long k;
26.      static long iy=0;
27.      static long iv[NTAB];
28.      float temp;
29.
30.      if(*idum <= 0 || !iy){
31.         if(-(*idum) < 1) *idum=1;
32.         else *idum = -(*idum);
33.         for(j = NTAB+7;j >= 0;j--){
34. k = (*idum)/IQ;
35.             *idum = IA*(*idum - k*IQ) - IR*k;
36.             if(*idum < 0) *idum += IM;
37.             if(j < NTAB) iv[j] = *idum;
38.         }
39.         iy = iv[0];
40.      }
41.      k = (*idum)/IQ; /* Start here when not initializing */
42.      *idum = IA*(*idum - k*IQ) - IR*k; /* Compute idum = (IA*idum) % IM
```

```
43.                                          without overflows by Schrage's
44.                                          method. */
45.      if(*idum < 0) *idum += IM;
46.      j = iy/NDIV; /* Will be in the range 0..NTAB-1. */
47.      iy = iv[j]; /* Output previously stored value and refill the
48.                     shuffle table. */
49.      iv[j] = *idum;
50.      if((temp = AM*iy) > RNMX) return RNMX; /* Because users don't expect
51. endpoint values. */
52.      else return temp;
53.      } /* end ran1 */
```

ANEXO B – RAN2

```
#include <math.h>
#define M 714025
#define IA 1366
#define IC 150889
float ran2(long *idum)
{
    static long iy,ir[98];
    static int iff=0;
    int j;
    void nrerror();

    if (*idum < 0 || iff == 0) {
        iff=1;
        if ((*idum=(IC-(*idum)) % M) < 0) *idum = -(*idum);
        for (j=1;j<=97;j++) {
            *idum=(IA*(*idum)+IC) % M;
            ir[j]=(*idum);
        }
        *idum=(IA*(*idum)+IC) % M;
        iy=(*idum);
    }
    j=1 + 97.0*iy/M;
    if (j > 97 || j < 1) nrerror("RAN2: This cannot happen.");
    iy=ir[j];
    *idum=(IA*(*idum)+IC) % M;
    ir[j]=(*idum);
    return (float) iy/M;
}
#undef M
#undef IA
#undef IC
```

ANEXO C – RAN3

```
#define MBIG 1000000000
#define MSEED 161803398
#define MZ 0
#define FAC (1.0/MBIG)
#include "nrutil.h"
float ran3(idum)
long *idum;
{
    static int inext,inextp;
    static long ma[56];
    static int iff=0;
    long mj,mk;
    int i,ii,k;

    if (*idum < 0 || iff == 0) {
        iff=1;
        mj=MSEED-(*idum < 0 ? -*idum : *idum);
        mj %= MBIG;
        ma[55]=mj;
        mk=1;
        for (i=1;i<=54;i++) {
            ii=(21*i) % 55;
            ma[ii]=mk;
            mk=mj-mk;
            if (mk < MZ) mk += MBIG;
            mj=ma[ii];
        }
        for (k=1;k<=4;k++)
            for (i=1;i<=55;i++) {
                ma[i] -= ma[1+(i+30) % 55];
                if (ma[i] < MZ) ma[i] += MBIG;
            }
        inext=0;
        inextp=31;
        *idum=1;
    }
    if (++inext == 56) inext=1;
    if (++inextp == 56) inextp=1;
    mj=ma[inext]-ma[inextp];
    if (mj < MZ) mj += MBIG;
    ma[inext]=mj;
    return mj*FAC;
}
```

ANEXO D – RAN4

```
#define IM 11979
#define IA 430
#define IC 2531
#define NACC 24
#define IB1 1L
#define IB3 4L
#define IB4 8L
#define IB32 0x80000000L
#define MASK IB1+IB3+IB4
#include "nrutil.h"

typedef struct IMMENSE {unsigned long l,r;} immense;


float ran4(idum)
int *idum;
{
    static int newkey,iff=0;
    static immense inp,key,jot;
    static double pow[66];
    unsigned long isav,isav2;
    int j;
    double r4;
    void des();

    if (*idum < 0 || iff == 0) {
        iff=1;
        *idum %= IM;
        if (*idum < 0) *idum += IM;
        pow[1]=0.5;
        key.r=key.l=inp.r=inp.l=0L;
        for (j=1;j<=64;j++) {
            *idum = ((long) (*idum)*IA+IC) % IM;
            isav=2*(unsigned long)(*idum)/IM;
            if (isav) isav=IB32;
            isav2=(4*(unsigned long)(*idum)/IM) % 2;
            if (isav2) isav2=IB32;
            if (j <= 32) {
                key.r=(key.r >>= 1) | isav;
                inp.r=(inp.r >>= 1) | isav2;
            } else {
                key.l=(key.l >>= 1) | isav;
```

```
44.                 inp.l=(inp.l >>= 1) | isav2;
45.             }
46.             pow[j+1]=0.5*pow[j];
47.         }
48.         newkey=1;
49.     }
50.     isav=inp.r & IB32;
51.     if (isav) isav=1L;
52.     if (inp.l & IB32)
53.         inp.r=((inp.r ^ MASK) << 1) | IB1;
54.     else
55.         inp.r <<= 1;
56.     inp.l=(inp.l << 1) | isav;
57.     des(inp,key,&newkey,0,&jot);
58.     r4=0.0;
59.     for (j=1;j<=NACC;j++) {
60.         if (jot.r & IB1) r4 += pow[j];
61.         jot.r >>= 1;
62.     }
63.     return r4;
64. }
65.
66. #undef IM
67. #undef IA
68. #undef IC
69. #undef NACC
70. #undef IB1
71. #undef IB3
72. #undef IB4
73. #undef IB32
74. #undef MASK
```