dx.doi.org/10.7437/NT2236-7640/2011.02.004
Notas Técnicas, v. 1, n. 2, 2011

Geradores de Números Aleatórios

Gabriela Sena Souza*
Centro Brasileiro de Pesquisas Físicas - CBPF/MCT, Rua Dr. Xavier Sigaud, 150, Urca, Rio de Janeiro, RJ, CEP 22290-180 – Brazil and Universidade do Estado do Rio de Janeiro, UERJ, Rua São Francisco Xavier, 524, Maracanã, Rio de Janeiro, RJ, Cep 20550-900 – Brazil

Nilton Alves Jr.†
Centro Brasileiro de Pesquisas Físicas - CBPF/MCT, Rua Dr. Xavier Sigaud, 150, Urca, Rio de Janeiro, RJ, CEP 22290-180 – Brazil

Esta Nota Técnica consiste no estudo dos principais geradores de números aleatórios existentes: Gerador Linear Congruente, Multiply-With-Carry, Lagged Fibonacci, Mersenne Twister e /dev/random. Neste trabalho também foram comentados dois testes que podem vir a ser realizados para verificar a qualidade dos geradores: teste Monte Carlo e teste Chi-Quadrado. Analisando os métodos que são utilizados pelos geradores, foi possível concluir sobre o desempenho, o tamanho das sequências geradas e principais características de cada um deles.

Palavras-chaves: Geradores, Aleatoriedade, Métodos.

Sumário

*Electronic address: gabisenaa@gmail.com
†Electronic address: naj@cbpf.br

1. INTRODUÇÃO

Em estatística, um número aleatório é aquele que pertence a uma série numérica e que não pode ser previsto a partir dos membros anteriores da série. Sequências de números aleatórios podem ser geradas, por exemplo, através do lançamento de um dado ou de uma moeda. Nessas duas situações sabemos quais são os resultados possíveis, porém não temos como prever qual será o resultado obtido em cada lançamento.

Na história antiga, os conceitos de acaso e de aleatoriedade estavam entrelaçados com o destino. Muitos povos antigos jogavam dados para determinar o destino, e esses jogos mais tarde passaram a ser chamados de jogos de azar.

No século XIX os conceitos de aleatoriedade começaram a ser estudados com mais rigor matemático. Em 1866, John Venn publicou o livro "The Logic of chance"[1]. Este livro contém estudos sobre probabilidade, aleatoriedade e possui um capítulo contendo o ponto de vista do autor sobre a aleatoriedade dos dígitos do número π.

Na primeira parte do século XX observou-se um crescimento muito rápido no estudo formal de aleatoriedade. Os cientistas da computação começaram a perceber que a introdução da aleatoriedade em alguns cálculos poderia ser uma ferramenta eficaz para a obtenção de algoritmos melhores.

A primeira tentativa de fornecer dígitos aleatórios ocorreu em 1927 quando a editora Cambridge University publicou um livro que possuia uma tabela de 41.600 dígitos desenvolvida por Leonard H.C. Tippet. Em 1947 a RAND Corporation gerou uma sequência de números a partir de uma simulação eletrônica de uma roda de roleta. Os resultados foram publicados em 1955 com o título Million Random Digits with 100.000 Normal Deviates.

Um grande desafio para os programadores desde que o estudo sobre aleatoriedade começou, sempre foi conseguir gerar sequências totalmente aleatórias no computador. Os números gerados no computador através de algoritmos são chamados de pseudo-aleatórios, já que os algoritmos geram sequências de números totalmente previsíveis e portanto nenhum aleatório verdadeiro. A palavra "aleatório" é usada somente para um processo físico aleatório, como por exemplo

o tempo decorrido entre os cliques de um Contador Geiger[1] colocado ao lado de uma amostra de um elemento radioativo. Os dados obtidos na leitura do contador Geiger serão totalmente aleatórios e esses dados tratados podem ser usados como semente na geração de números aleatórios. Considerando que o custo computacional para geração de longas cadeias de números aleatórios é alta, métodos recursivos são mais explorados, porque acabam economizando processamento e memória. Vários métodos foram desenvolvidos e podem ser implementados em algoritmos. Chamamos esses métodos de "Geradores de números aleatórios".
Na seção 2 desta Nota Técnica encontramos uma explicação sobre o método utilizado pelos seguintes geradores: Linear Congruente, Multiply-With-Carry, Lagged Fibonacci, Mersenne Twister e /dev/random.
Na seção 3 são descritos dois testes que podem ser utilizados para verificar a qualidade de um gerador: teste Monte Carlo e o teste Chi-quadrado. Neste trabalho, os geradores não foram testados já que o desempenho de cada um deles varia de apliacação pra aplicação. Cada usuário deverá realizar seus próprios testes para saber qual será o melhor gerador para o seu caso.
Na seção 4 podemos comparar os geradores a partir de uma tabela que caracteriza o período, a usabilidade e a implementação de cada um.

2. GERADORES DE NÚMEROS ALEATÓRIOS

Algumas características são indispensáveis para todos os geradores de números aleatórios:

- Os números gerados devem seguir uma distribuição uniforme, ou seja, a probabilidade de se gerar qualquer número em um intervalo contido em um espaço amostral deve ser inversamente proporcional ao tamanho do intervalo.

- O valor de um número na sequência não deve afetar o valor do próximo (na prática a maioria dos geradores usa sequências recursivas, então há essa dependência dos valores anteriores, mas isso não é estatisticamente significativo).

- A sequência não deve se repetir nunca. Isso é teoricamente impossível mas na prática um período de repetição suficientemente grande é o bastante.

- A geração desses números deve ser rápida, de modo a poupar recursos computacionais para as simulações em si.

Vários testes podem ser realizados para verificar a qualidade dos geradores, mas o esperado é que todos os geradores obtenham estatisticamente os mesmos resultados quando acoplados com suas particulares aplicações do programa. Se os geradores não produzirem o mesmo resultado estatístico, significa que pelo menos um deles não é um bom gerador. Por exemplo, quando jogamos dois dados aleatoriamente, a soma das faces podem ser sete e dois, com um número de combinações possíveis seis e um, e a probabilidade $1/6$ e $1/36$, respectivamente. Logo, se fizéssemos um programa para simular esse lançamento, e isso fosse feito um número considerável de vezes, deveríamos obter muito mais vezes o número sete. E qualquer gerador que fosse usado, deveria encontrar o mesmo resultado estatístico, caso contrário, significaria que ele não é um bom gerador. No caso dos dados, nós sabemos qual resultado estatístico seria correto, mas existem muitas situações em que não sabemos qual o melhor resultado, e por isso, é necessário obter um gerador de números aleatórios que nos permita confiar nas sequências geradas. A maior parte das linguagens possuem rotinas para inicializar, e em seguida gerar números aleatórios, algumas delas serão citadas neste trabalho.

2.1. Função Rand

A função *rand*[2] é utilizada pela linguagem C para gerar números aleatórios. Ela é sempre utilizada com a função *srand*, que serve para determinar qual será o número que irá inicializar a sequência de números aleatórios. O gerador de números aleatórios é inicializado chamando-se $srand(iseed)$ com algum valor positivo *iseed*, e cada valor de inicializar resulta numa sequência aleatória diferente. O mesmo valor inicializado de *iseed* sempre vai retornar a mesma sequência aleatória. Obtem-se uma sequência números aleatórios através de sucessivas chamadas da função $rand()$. Essa função retorna um inteiro que é geralmente na faixa de 0 até o maior representante valor positivo do tipo *int*. Esse maior valor é disponível como RAND_MAX. As funções *rand* e *srand* possuem os Códigos 2.1 e 2.2 respectivamente.

```
int rand(void)
{

next = next * 1103515245 + 12345;
return (unsigned int) (next/65536)  % 32768;

}
```

Código 2.1: *Função rand*

```
void srand(unsigned int seed)
{

next = seed;

}
```

Código 2.2: *Função srand*

Analisando os códigos fonte das funções *rand* e *srand* encontramos alguns problemas. Em primeiro lugar, sabemos que sempre que colocarmos a mesma semente (*iseed*) a sequência será a mesma. Segundo, o maior valor possível que pode ser retornado pela função *rand*, que pode ser chamado de RAND_MAX, é apenas 32.767. Isso pode ser muito ruim em determinadas circunstâncias, como por exemplo para a integração Monte Carlo. Nessa integração você pode querer avaliar 1.000.000 pontos diferentes, mas pode

[1] O contador Geiger serve para medir certas radiações ionizantes (partículas alfa, beta ou radiação gama e raios-X). Ele é constituído de um tubo Geiger-Müller e de um sistema de amplificação e de registro do sinal.

estar avaliando 32.767 pontos 30 vezes cada.
O método utilizado na função *rand* para gerar números aleatórios é conhecido como Método Linear Congruente e ele pode ser utilizado de várias formas diferentes.

2.2. Método Linear Congruente

O método linear congruente[3] é baseado na seguinte relação de recorrência:

$$x_{n+1} = (ax_n + c)\ mod\ m,\ n \geq 0, \tag{1}$$

Nessa relação, m é chamado de módulo[2], a de multiplicador e c de incremento. Se esses valores forem escolhidos de forma errada, o período da sequência de números aleatórios pode ser drasticamente afetado. O período será sempre maior que m, logo, para aumentar o período deve-se ter o maior valor de m possível. Caso os parâmetros a, c e m sejam escolhidos de forma adequada, o algoritmo, apesar de simples, pode se tornar muito poderoso. No livro Numerical Recipes in C [3] encontramos uma tabela com bons valores para as constantes a, c e m.
Além da função rand, existem outras rotinas baseadas no Método Linear Congruente. Algumas rotinas podem ser encontradas neste livro e são elas: Ran1 e Ran2.
A rotina Ran1 é baseada em três geradores lineares congruentes e a escolha de quais constantes da tabela serão utilizadas nessa rotina é arbitrária. Já rotina Ran2 é perfeitamente adequada para quem quer ganhar velocidade, já que ela utiliza apenas um gerador congruente linear. Essas duas rotinas possuem um desempenho melhor que a função *rand*, apesar de utilizarem o mesmo método.
O Método Linear Congruente é muito utilizado e existem algumas variações desse método com o objetivo de melhorá-lo, uma delas é o método Multiply-With-Carry[4].

2.3. Multiply-With-Carry

O Método Mutiply-with-carry (MWC) é uma variação do Método Linear Congruente. A principal vantagem do método MWC é que ele gera sequências muito rapidamente e com períodos imensos variando entre 2^{60} e $2^{2.000.000}$.
Os geradores MWC costumam se comportar melhor que os outros geradores em testes de aleatoriedade.
Na sua forma mais comum, um gerador MWC necessita de uma base b, um multiplicador a, um conjunto de valores aleatórios de sementes e um valor inicial para c.
Uma sequência MWC é então uma sequência de pares x_n:

$$x_n = (ax_{n-r} + c_{n-1})\ mod\ b \tag{2}$$

E c_n é determinado por:

$$c_n = \frac{ax_{n-r} + c_{n-1}}{b} \tag{3}$$

Comparando esse método ao gerador congruente linear podemos perceber que a principal diferença entre eles é que no método MWC o parâmetro c varia ao longo de cada iteração.
O método MWC, com valores adequados para os parâmetros, se torna muito poderoso e passa em testes estatísticos que os geradores congruentes lineares não passam.

2.4. Geradores Lagged Fibonacci

Os geradores Lagged Fibonacci[5] foram criados para serem uma melhoria aos geradores congruentes lineares. Esses geradores são baseados na sequência Fibonacci, que pode ser descrita pela seguinte relação de recorrência:

$$S_n = S_{n-1} + S_{n-2} \tag{4}$$

A fórmula da sequência Fibonacci nos mostra que cada termo da sequência é igual a soma dos seus dois termos antecessores. Mas podemos generalizar essa fórmula da seguinte forma:

$$S_n = S_{n-j} \star S_{n-k}\ (mod\ m),\ 0 < j < k \tag{5}$$

Nessa fórmula, o novo termo é uma combinação de quaisquer dois termos anteriores. O termo m é normalmente uma potência de 2 ($m = 2^M$) onde M frequentemente é 32 ou 64. O operador $\star$ pode ser uma operação de adição, subtração, multiplicação e pode ser também uma operação binária.
Um exemplo de um gerador Lagged Fibonacci muito utilizado é o Mersenne Twister [6].

2.5. Mersenne Twister

O Mersenne Twister é um gerador de números pseudo-aleatórios que foi desenvolvido em 1997 por Makoto Matsumoto e Takuji Nishimura. Ele é uma variação de um tipo de gerador Lagged Fibonacci e gera números aleatórios de altíssima qualidade. Foi criado para corrigir diversas falhas existentes em algoritmos mais antigos. O nome Mersenne Twister deriva do fato do tamanho do período ser escolhido como um Primo de Mersenne[3].
Esse gerador é baseado numa matriz de recorrência linear e sua fórmula de recorrência é bastante complexa.
Existem duas variações comuns do gerador Mersenne Twister, diferindo apenas no tamanho do Primo de Mersenne usado. São elas: MT19937 (32-bit) e MT19937-64 (64-bit).
Para várias aplicações o Mersenne Twister tem se tornado o

[2] A operação módulo encontra o resto da divisão de um número por outro. Dados dois números a (o dividendo) e b (o divisor), a mod b é o resto da divisão de a por b.

[3] Primo de Mersenne é um número de Mersenne (número na forma $M_n = 2^n - 1$, com n sendo um número natural) que também é um número primo.

gerador de números aleatórios padrão, ele é utilizado por exemplo na linguagem de programação Python.
As principais caracteríticas do Mersenne Twister são o fato de possuir um período muito longo, de $2^{19.937} - 1$, e o fato de ter sido aprovado pelos principais testes de aleatoriedade existentes.

2.6. Geradores /dev/random e /dev/urandom

O sistema operacional Linux foi o primeiro sistema a implementar um gerador de números aleatórios verdadeiro.
Esse gerador de números aleatórios foi implementado pela primeira vez em 1994 por Theodore Ts'o [7]. Na implementação, o gerador faz uma estimativa das interrupções do teclado, cliques do mouse e outros processos e a partir disso gera números aleatórios.
O gerador /dev/random é adequado para usuários que necessitam de um gerador de alta qualidade, como por exemplo para gerar chaves criptográficas.
Quando o computador não estiver sendo usado, a leitura a partir de /dev/random irá bloquear até que um novo ruído ambiental seja recolhido.
Quando a leitura a partir de /dev/random for bloqueada, o gerador /dev/urandom é utilizado. Esse gerador reutiliza o que já foi usado para produzir bits pseudo-aleatórios. Isso significa que a chamada não irá bloquear, mas a saída pode não ser aleatória como é com /dev/random. O gerador /dev/urandom pode ser utilizado para aplicações menos seguras.
Os geradores /dev/random e /dev/urandom também estão disponíveis nos sistemas operacionais Solaris, Mac OS X, NetBSD, etc.

3. TESTES DE GERADORES DE NÚMEROS ALEATÓRIOS

O objetivo dos testes de geradores de números aleatórios é garantir que os geradores testados irão obter resultados estatísticos confiáveis para uma determinada aplicação.
Uma das melhores formas de testar um gerador é verificar se o resultado estatístico obtido por ele é o mesmo que o aqueles obtidos por outros geradores. Caso o resultado não seja o mesmo, significa que pelo menos um dos geradores utilizados não é bom.
Outra forma de avaliar um gerador é colocar os dados obtidos por ele em um histograma, em seguida verificar a proporção de valores que ocorre em cada intervalo do histograma e comparar se essa proporção se aproxima daquela esperada.
Os testes realizados com os geradores podem ser empíricos ou teóricos. Nos testes empíricos os números que são obtidos no gerador são avaliados com técnicas estatísticas. Já os testes teóricos são sofisticados e envolvem ferramentas matemáticas mais complexas. Os testes teóricos indicam a qualidade de um gerador de acordo com algum critério matemático, sem exigir a geração de números. Por exemplo, nos geradores congruentes lineares, os testes avaliam as constantes a, c e m.
Neste trabalho foram estudados dois testes: teste Monte Carlo[8] e teste Chi-Quadrado[8].

3.1. Teste Monte Carlo

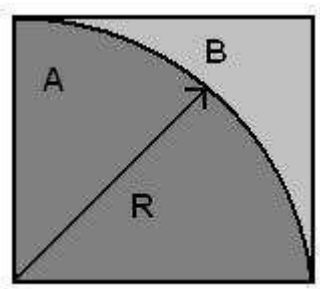

Figura 1: Circunferência circunscrita num quadrado.

Este teste utiliza o método Monte Carlo, que é um método estatístico utilizado em simulações estocásticas, tem aplicação em diferentes áreas como física e matemática e pode ser usado de diversas formas diferentes.
O método Monte Carlo foi desenvolvido por John von Neumann, Stanislaw Ulam e Nicholas Metropolis enquanto investigavam as distâncias percorridas por nêutrons dentro de determinados materiais. Depois de realizarem vários experimentos, perceberam que os cálculos dessas distâncias percorridas pelos nêutrons não poderiam ser feitos analiticamente. Ulam, então, sugeriu que fossem feitos experimentos aleatórios, e através desses experimentos surgiu o método Monte Carlo. O nome do método é uma referência ao cassino Monte Carlo que existe em Monaco, onde o tio de Ulam costumava gastar todo seu dinheiro.
O método Monte Carlo necessita de uma grande quantidade de números aleatórios para funcionar corretamente, logo, se utilizarmos esse método como um teste, poderemos saber se o gerador de números aleatórios que foi utilizado é aceitável.
Uma forma de realizar este teste é obter valores para π através de uma razão entre as áreas da Figura 1. A razão entre as áreas dessa figura é calculada através de dados obtidos por um gerador de números aleatórios. Se os valores encontrados para π após os cálculos se aproximarem do valor real, significa que o gerador é confiável.
Podemos realizar o teste da seguinte forma: primeiramente "jogamos"pontos aleatórios numa circunferência circunscrita num quadrado (Fig. 1), ou seja, utilizamos um gerador de números aleatórios para lançar pontos na figura. Depois, fazemos uma razão entre o número de pontos dentro do quadrado e o número de pontos dentro da circunferência, e assim podemos encontrar uma estimativa para o valor de π.
Sabendo que a área de $1/4$ de círculo é:

$$A = \frac{\pi R^2}{4} \tag{6}$$

E sabendo que a área do quadrado é:

$$B = R^2 \tag{7}$$

Podemos dizer que π será:

$$\pi = \frac{4A}{B} \tag{8}$$

Assim, quanto mais aleatoriamente conseguirmos jogar os pontos na circunferência, obteremos uma melhor estimativa para a razão entre as áreas A e B e consequentemente para π.

3.2. Teste Chi-Quadrado

Este é o teste mais utilizado e é denotado por χ^2.
O método utilizado neste teste consiste em comparar as possíveis divergências entre os dados obtidos por um gerador para um determinado evento e os dados esperados para o mesmo evento. Podemos dizer que um gerador é bom quando as diferenças entre os dados obtidos e os dados esperados se aproximam de zero.
Karl Pearson propôs a seguinte fórmula para medir as possíveis discrepâncias entre os dados obtidos e os dados esperados:

$$\chi^2 = \sum_{i=1}^{n} [\frac{(o_n - e_n)^2}{e_n}] \tag{9}$$

onde:
o = dados obtidos; e = dados esperados.
Quando os dados obtidos são muito próximos aos esperados, o valor de χ^2 é pequeno. Mas, quando as divergências são grandes, a diferença $o - e$ passa a ser também grande e consequentemente, χ^2 assume valores altos.
Para utilizar esse teste, é necessário obter duas estatísticas denominadas χ^2 e χ^2_c tabelado. O χ^2_c tabelado são valores fixos que estão contidos na Tabela 1. A Tabela 1, conhecida como a tabela de Chi Quadrado, mostra o valor da probabilidade nas colunas e o número de Graus de liberdade $(G.L.)$ nas linhas, que é calculado da seguinte forma:

$$G.L. = Espaço\ Amostral - 1 \tag{10}$$

Na coluna referente a 5% de probabilidade encontra-se o valor crítico de χ^2_c, com o qual deve ser comparado o valor calculado de χ^2.

				χ^2_c			
GL / P	**0,990**	**0,950**	**...**	**0,050**	**0,020**	**0,010**	**0,001**
1	0,000	0,004		**3,841**	5,412	6,635	10,827
2	0,020	0,103		**5,991**	7,824	9,210	13,815
3	0,115	0,352		**7,815**	9,837	11,345	16,266
4	0,297	0,711		**9,488**	11,668	13,277	18,467
5	0,554	1,145		**11,070**	13,388	15,080	20,515
...							

Tabela I: Na tabela podemos encontrar na primeira linha e na primeira coluna, os valores para a probabilidade e o grau de liberdade respectivamente. Os valores que se encontram na coluna de 0,050 de probabilidade são os valores do χ^2_c.

O χ^2 calculado é obtido a partir dos dados experimentais, levando-se em consideração os valores obtidos e esperados. Já o χ^2_c tabelado depende do número de graus de liberdade.
A decisão é feita comparando os dois valores de χ^2:

- Se o χ^2 calculado for maior ou igual que o χ^2_c tabelado, então rejeitam-se os dados.
- Se o χ^2 calculado for menor que o χ^2_c tabelado, então aceitam-se os dados.

Por exemplo, se um dado for jogado 6 vezes, a probabilidade de cair qualquer face é $1/6$, já que espera-se obter uma vez cada face (1, 2, 3, 4, 5 e 6).
Supondo que um dado foi jogado 186 vezes e se obteve: face $1 = 34$; face $2 = 29$; face $3 = 30$, face $4 = 32$; face $5 = 28$; face $6 = 33$.
O resultado esperado para cada face será calculada da seguinte forma:
A probablidade de se obter cada face multiplicada pelo número de vezes que o dado foi lançado: $1/6\ .\ 186 = 31$.
Qual será o valor de χ^2?
Calculando,

$$\chi^2 = \sum_{i=1}^{n} [\frac{(o_n - 31)^2}{31}] \tag{11}$$

obteve-se:

$$\begin{aligned} \chi^2 &= (0,2903 + 0,1290 + 0,0322 + 0,0322 + \\ &+ 0,2903 + 0,1290) = 0,903 \end{aligned} \tag{12}$$

Como se pode interpretar esse valor?
Sabendo que o espaço amostral neste caso é 6, já que o dado possui seis faces, utilizando a equação 10, chegamos a conclusão que G.L. = 5.
Verificando-se a Tabela 1 na linha em que G.L. = 5 encontra-se χ^2_c igual a 11,070. Como o valor de Chi-quadrado obtido (0,903) foi menor que o esperado ao acaso (11,070) admite-se que o dado seja honesto.

4. TABELA COMPARATIVA

Na tabela 2 encontramos uma comparação entre os geradores estudados neste trabalho. Ela possui o período de todos os geradores, com exceção de /dev/random e Lagged Fibonacci, já que não possuem um período fixo. Além do período, encontramos a usabilidade, ou seja, em quais linguagens, sistemas operacionais e softwares os geradores são utilizados. Através da tabela também é possível verificar se é fácil implementar os geradores.

5. CONCLUSÃO

Através do estudo que foi realizado sobre os principais geradores de números aleatórios e os seus respectivos métodos, foi possível identificar quais são as principais características de cada gerador e quais são os geradores que possuem melhor desempenho.
O gerador da própria linguagem C (função rand) pode ser considerado bom para gerar sequências pequenas mas quando se deseja uma sequência muito grande e de alta

Tabela II: Tabela Comparativa

GERADORES DE NÚMEROS ALEATÓRIOS	PERÍODO	USABILIDADE	IMPLEMENTAÇÃO
Função rand	-	C e C++	Fácil
Método Linear Congruente	$>$m	Rand, Ran1, Ran2	Fácil
Multiply-With-Carry	2^{60} até $2^{2.000.000}$	-	Média
Lagged Fibonacci	-	Mersenne Twister	Fácil
Mersenne Twister	$2^{19.937-1}$	Python	Fácil
Gerador /dev/random	-	Linux, Mac OS X	Média

qualidade, esse gerador não deve ser utilizado, já que ele possui um período curto.

Os geradores Ran1 e Ran2, que utilizam o Método Congruente Linear, possuem uma performance boa, mas não foram aprovados em alguns testes de aleatoriedade considerados importantes. O Método Congruente Linear possui um rendimento razoável quando seus parâmetros são escolhidos de forma adequada.

O gerador Mersenne Twister foi criado para corrigir diversas falhas em algoritmos antigos e entre todos os geradores de números pseudo-aleatórios existentes ele pode ser considerado um dos melhores. Foi aprovado nos principais testes de aleatoriedade.

O método Multiply-with-Carry pode ser considerado uma evolução do Método Linear Congruente. Ele é capaz de gerar sequências muito longas e de grande qualidade se os parâmetros adequados forem escolhidos.

O gerador /dev/random obtem números totalmente aleatórios, porém, depende da utilização do computador para que sejam obtidos os dados de cliques do mouse, interrupções do teclado e etc. É aconselhável que esse método seja utilizado apenas na obtenção de uma semente para inicializar um outro gerador.

Os testes citados neste trabalho são os mais utilizados, mas existem muitos outros testes que podem ser aplicados.

Entre todos os geradores que foram estudados, concluiu-se que os que possuem melhor desempenho são o Mersenne Twister e o Multiply-with-Carry. O fato destes geradores possuírem melhor desempenho, não significa que os outros geradores não possam ser utilizados. Toda vez que for necessário utilizar um gerador de números aleatórios, deve ser feita uma análise do método que é utilizado por ele e deve ser estudado se aquele gerador é capaz de obter resultados estatísticos corretos para a função que ele vai exercer. Através desse estudo é que torna-se possível saber se o gerador terá um bom desempenho ou não para a aplicação específica.

[1] Venn, J., "The Logic of Chance", Cambridge University Press, 1866.

[2] Kernighan, B. W., Ritchie, D. M., "The C Programming Language", PTR Prentice Hall, Engleawood Cliffs, New Jersey, 2th edition, ISBN 0-13-110362-8, 1988.

[3] Press, W. H., Teukolsky, S. A., Flannery, B. P., Vetterling W. T., "Numerical Recipes in C: The Art of Scientific Computing", Cambridge University Press, 1th edition, ISBN 0-521-35465-X book, 1988.

[4] Goresky, M., Klapper, A., "Efficient Multiply-with-Carry Random Number Generators with Maximal Period", Institute for Advanced Study, University of Kentucky, CM165A-01, 2003.

[5] Rosa, F. H. F. P., Junior, V. A. P., "Gerando Números Aleatórios", Laboratório de Matemática Aplicada, 21 de novembro de 2002.

[6] Vieira, C. E. C., Souza R. C., Ribeiro C. C. C., "Um Estudo Comparativo entre Três Geradores de Números Aleatórios", PUC-RioInf.MCC16/04 Junho, 2004.

[7] Gutterman, Z., Pinkas, B., Reinman, T., "Analysis of the Linux Random Number Generator", The Hebrew University of Jerusalem, University of Haifa, March 6, 2006.

[8] Zeeb, C. N., "Random Number Generator Recommendation", Colorado State University, Departament of Mechanical Engineering, Fort Collins 80523.