Estudo comparativo de geração de números aleatórios

Paulo Henrique da Silveira
Instituto de Física - USP

Gabriel Moraes
Instituto de Física - USP

Hugo Salia
Instituto de Física - USP

15 de Dezembro de 2015

Palavras chave

Gerador de números aleatórios, teste de χ^2, monte carlo, MLC,

1 Introdução

1.1 Propriedade dos números aleatórios

Uma sequência de números aleatórios $x_1, x_2, ...$ deve ser independente e seguir uma distribuição uniforme, i.e., todo x_i é uma amostra independente de uma distribuição uniforme e contínua no intervalo $I = [0, 1]$, portanto a função densidade probabilidade (figura 1) de x é:

$$f(x) = \begin{cases} 1 & \text{se } 0 \leq x \leq 1 \\ 0 & \text{se outro valor} \end{cases} \tag{1}$$

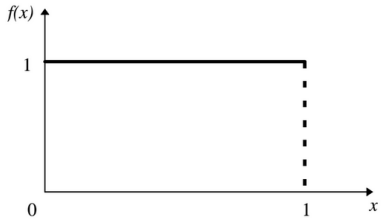

Figura 1: FDP para números aleatórios

Figura 1: *Função densidade de probabilidad (FDP) para números aleatórios*

O valor esperado e a variância para cada x_i é:

$$E(x) =< x >= \int_0^1 x dx = \frac{1}{2} \tag{2}$$

$$V(x) =< x^2 > - < x >^2 = \int_0^1 x^2 dx - [E(x)]^2 = \frac{1}{12} \tag{3}$$

Se um intervalo $I = [0, 1]$ é subdividido em n classes, o valor esperado de observação em cada intervalo será de $\frac{n}{N}$, onde N é o número total de observações.

1.2 Números aleatórios são realmente aleatórios?

Logo temos todas as propriedades básicas para gerar números aleatórios, entretanto cada sequência de valores é igualmente possível aparecer, isso significa que um bom gerador produzirá sequências que não parecem nada aleatórias (figura 2).

Figura 2: *Será que a criatura realmente está dizendo uma sequência aleatória?*[**??**]

O que é dito para Dilbert é totalmente correto, é impossível provar se um gerador é aleatórios, pois a criatura pode estar gerando números perfeitamente aleatórios a anos e Dilbert fora justo na hora errada. Caso ele ficasse mais tempo talvez perceberia a aleatoriedade.

Assim temos o problema técnico de dizer se uma sequência de fato é realmente aleatória, uma maneira é gerar várias sequências e efetuar testes estatísticos, como veremos na sessão 1.6, alguns testes falharão, pois como vimos parte das sequências é não aleatória, o que não prejudica o método, mas se repetido muitas vezes e for detetado esse padrão, devemos nos atentar que tal gerador pode estar enviesado.

Há várias maneiras de gerar números aleatórios uns melhores que outros, como veremos mais a frente.

1.3 Verdadeiros geradores aleatórios (TRNGs)

TRNGs extraem a aleatoriedade de fenômenos físicos, como por exemplo, jogadas de dados não viciados, um jogo de cara ou coroa, roleta de um cassino, até mesmo decaimentos radioativos[1], condições climáticas[2] e clock da CPU de um computador pessoal.

Para exemplificar vamos tomar um dado de 6 faces não viciado, seu espaço amostral é $\Omega = \{1, 2, 3, 4, 5, 6\}$. Seja uma variável aleatório X, temos que:

$$f(x) = p(X = x_i) = 1/6; x_i \in \Omega \tag{4}$$

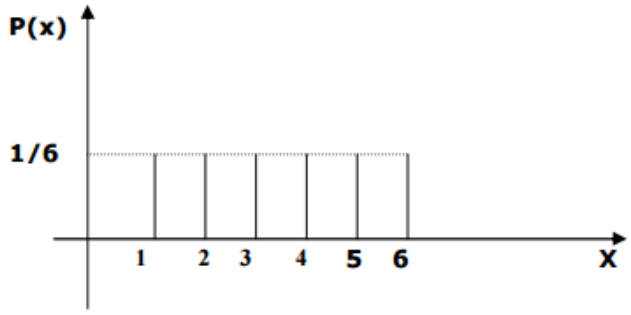

Figura 3: *Função densidade de probabilidade (FDP) de um dado*

1.4 Geradores de Números pseudo aleatórios (PRNGs)

Como o nome "pseudo"diz, esses geradores não são verdadeiros, no sentido similar ao real, isso é, não se comporta como jogar os dados ou saber os números da loteria. São obtidos através de algoritmos computacionais e são determinísticos, isso é, sabendo o método e as condições iniciais, facilmente obtermos a sequência.

Em contra partida os PRNGs são super rápidos e eficientes para gerar grandes sequências de valores, diferente dos TRNGs que tem a limitação do tempo para gerá-los, imagine o tempo para jogar 1000 dados e obter a sequência, enquanto com os PRNGs pode-se obter em questão de segundos.

Como exemplo vamos gerá-los pelo método linear congruente ([3]), um dos métodos recursivos, i.e., $x_n = f(x_{n-1,x_{n-2}})$ mais comuns.

[1]HotBits service at Fourmilab in Switzerland
[2]Random.org
[3]para maiores detalhes ver 2.1

$$x_n = 5x_{n-1} + 1 mod 16 \tag{5}$$

Iniciando a série com $x_0 = 5$ o qual chamamos de seed e calculando os 32 primeiros elementos: 10, 3, 0, 1, 6, 15, 12, 13, 2, 11, 8, 9, 14, 7, 4, 5, 10, 3, 0, 1, 6, 15, 12, 13, 2, 11, 8, 9, 14, 7, 4, 5.

Podemos dizer que essa sequência é aleatória? para facilitar nossa análise, vemos que os números estão entre 0 e 15, dividindo-os por 16 obtemos uma sequência com números entre 0 e 1 (tabela 1)

Tabela 1: *Sequência exemplo de 32 números aleatórios gerados pelo método congruente linear com multiplicado m = 16 e período na 17º posição*

0,6250	0,1875	0,0000	0,0625
0,3750	0,9375	0,7500	0,8125
0,1250	0,6875	0,5000	0,5625
0,8750	0,4375	0,2500	0,3125
0,6250	0,1875	0,0000	0,0625
0,3750	0,9375	0,7500	0,8125
0,1250	0,6875	0,5000	0,5625
0,8750	0,4375	0,2500	0,3125

Vemos que conhecida a função e o seed (x_0) podemos reobter a mesma sequência, logo dizemos que esse método é determinístico, embora a sequência ser aleatória. Vale ressaltar que a partir do 17º posição a sequência torna a se repetir, logo dizemos que o gerador possui comprimento de ciclo de 16 valores. Alguns geradores não repetem os primeiros números sendo esses números chamados de calda, vemos na figura 4tem o que chamam de cauda.

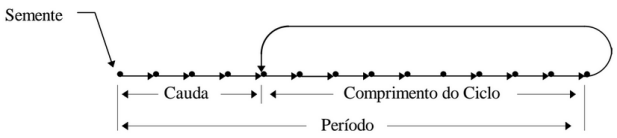

Figura 2: Comprimento do ciclo, cauda e período de gerador de números aleatórios

Figura 4: *Comprimento do ciclo, cauda e período de gerador de números aleatórios*

Portanto qual é a vantagem de usar geradores psedo aleatórios? ora, como vimos esses algoritmos rápidos e eficientes, sendo útil quando for necessário gerar uma grande sequência de dados, garantindo períodos longos, uniformidade e independência dos números, temos excelentes geradores, embora sua usabilidade dependa de qual ambiente se está trabalho, criptografia por exemplo, é um pessímo lugar, entretanto para simulação de dados é excelente.

1.5 Comparação entre PRNGs and TRNGs

Abaixo há um resumo das utilidades e funcionalidades de ambos tipos de geradores.

Tabela 2: *Comparação dos verdadeiros para os pseudo geradores*

Características	PRNG	TRNG
Eficiência	Excelente	Ruim
Determinśtico	Sim	Não
Períodico	Sim	Não

Tabela 3: *Usabilidade dos dois tipos de geradores*

Usos	Geradores Mais Usados
Loteria	TRNG
Jogos e Apostas	TRNG
Simulação e Modelagem	PRNG
Criptografia e Segurança	TRNG

1.6 Testes para os geradores

Os testes tem objetivo de garantir que os geradores são de fato aleatórios, por meio de métodos estatísticos. Há várias maneiras de testá-los, seja por meio de verificar os testes estatísticos por vários métodos diferentes e ao final comparar seus resultados, se for diferente um ou mais dos geradores não é confiável ou através da visualização gráfica do mapa dos pontos e o histograma e para ver os padrões, da imagem, conforme vemos na figura 5 que é o mapa de dois algoritmos aleatórios, se nota um padrão na figura.

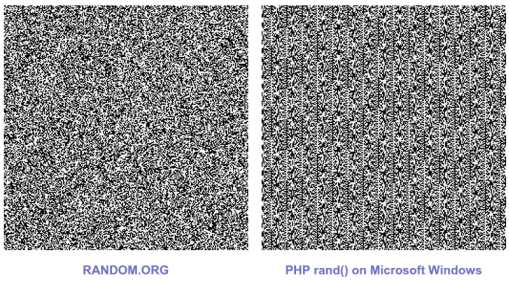

Figura 5: *Mapa dos pontos de um gerador verdadeiro (esq) e um pseudo (dir)*

Há dois tipos de testes, os empíricos e teóricos [**??**], nos empíricos, são gerados várias sequências e testados por métodos estatísticos. Já os teóricos é empregado técnicas mais sofisticadas [4].

[4] técnicas teóricas???

Vamos analisar os algoritmos pelo teste empírico do teste do χ^2 e no cálculo do valor π por Monte Carlo.

1.6.1 teste χ^2

O teste consiste em comparar as possíveis divergências por um gerador em determinado evento e os dados para o mesmo evento. Esse método é útil quando a diferença entre os dados obtidos e os dados esperados se aproxima de zero. Podemos quantificar tal diferença pela seguinte fórmula:

$$\chi^2 = \sum_{i=1}^{n} [\frac{(observado(i) - esperado(i))^2}{esperado(i)}] \quad (6)$$

Para utilizar esse teste é necessário comparar $\chi^2_{calculado}$ com o $\chi^2_{critico}$ tabelado, que são valores fixos (tabel 4). A tabela (4) mostra os valores críticos da probabilidade nas colunas e os graus de liberdade [5] nas linhas.

Se $\chi^2_{calculado} > \chi^2_{crítico}$ então rejeitam-se os dados, caso contrário aceita-se os dados.

Tabela 4: $\chi^2_{tabelado}$, primeira coluna contém os graus de liberdade e na primeria linha a probabilidade de confiabilidade

				χ^2_c	
GL / P	0.975	0.950	...	**0.050**	0.025
1	0.001	0.004		**3.841**	5.024
2	0.051	0.103		**5.991**	7.378
3	0.216	0.352		**7.815**	9.348
4	0.484	0.711		**9.488**	11.143
5	0.831	1.145	...	**11.070**	12.833

Exemplo, no nosso teste do dado não viciado, vamos supor que foram jogados 186 vezes e obtido (tabela 5):

Tabela 5: frequência das faces de um dado não viciado

Face	Frequência
1	34
2	29
3	30
4	32
5	28
6	33
Total	**186**

O valor esperado para cada face é o produto da probabilidade do dado vezes o número total de jogadas (eq.: 7).

[5] Graus de Liberdade = Espaço amostral - 1

$$esperado(i) = p(X)N = \frac{1}{6}186 = 31 \quad (7)$$

$$\chi^2 = \sum_{i=1}^{6}[\frac{(observado(i) - 31)^2}{31}] \quad (8)$$

$$\chi^2 = 0,903 \quad (9)$$

Com $\chi^2 = 0,903$ e comparando com a tabela 4 vemos que $0,903 < 11,143$ logo não rejeitamos o teste e além do mais podemos dizer que temos entre 97,5 e 95 porcento de confiabilidade, visto que $0,831 < 0,903 < 1,145$ e que o dado é honesto.

Vale ressaltar que todas nossas análises dos geradores vamos padronizar 29 graus de liberdade, isso é, subdividiremos em 30 bins dos histogramas. O χ^2_t é apresentado na tabela 6 que será comparado na análise dos nossos números aleatórios.

Tabela 6: chi^2_t para 29 graus de liberdade, para comparação na análise dos geradores.

					χ^2_c
GL P	0.9	0.6	0.5	0.4	**0.05**
29	19.77	26.47	28.34	30.28	**42.55**

1.6.2 Teste por Monte Carlo

Consiste basicamente em calcular o valor no número π por meio da razão entre as áreas, por meio da quantidade do número de pontos (figura 6), tal que A é a área de 1/4 do círculo de raio R e B é área de um quadrado de lado R e o valor de π é obtido pela razão 12.

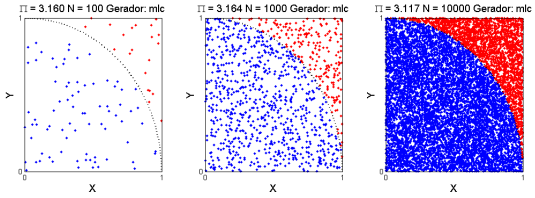

Figura 6: *Exemplo da distribuição dos pontos para o cálculo de π dentro de um quarto de círculo num quadrado*

$$A = \frac{\pi R^2}{4} \quad (10)$$

$$B = R^2 \quad (11)$$

$$\pi = 4\frac{A}{B} \quad (12)$$

2 Geradores de números aleatórios

Tabela 7: Período aproximado de geradores

Gerador	**Período aproximado**
Mersenne twister (default at MATLAB®)	$2^{19937} - 1$
SIMD-oriented fast Mersenne twister	$2^{19937} - 1$
Multiplicative congruential generator	$2^{31} - 1$
Multiplicative lagged Fibonacci generator	2^{124}
Combined multiple recursive generator	2^{127}
Shift-register with linear congruential generator	2^{64}
Modified subtract with borrow generator	2^{1492}

2.1 Método linear congruente

É considerado o mais popular, entre tantos outros métodos geradores. também conhecido como método congruente misto, foi primeiramente divulgado em um trabalho desenvolvido pelo Prof. D. H. Lehmer, em 1951, quando dos expedimentos executados pelo computador ENIAC no MIT[**??**]. Em suas pesquisas ele descobriu que restos de sucessivas potências de um número possuem boas características de aleatoriedade. Ele obtinha o n-ésimo número de uma sequência, tomando o resto da divisão da n-ésima potência de um inteiro a por um outro inteiro m. Isto é:

$$x_n = a^n mod(m) \quad (13)$$

$$x_n = a_{n-1} mod(m) \quad (14)$$

onde a é o modulo e m multiplicador

A popularidade dos geradores baseados neste método devesse ao fato de serem facilmente analisados e de algumas garantias de suas propriedades dadas pela teoria das congruências [**??**].

Na figura 7, são dados os histogramas para 100, 1000 e 10000 números aleatórios entre 0 e 1 gerados a partir do Método Linear Congruente que nós mesmos produzimos no MATLAB. Além disso, foram construídos os mapas de primeiro retorno com o objetivo de verificar se é possível identificar a periodicidade.

É fácil notar as faixas periódicas no mapa de primeiro retorno, já que o método linear congru-

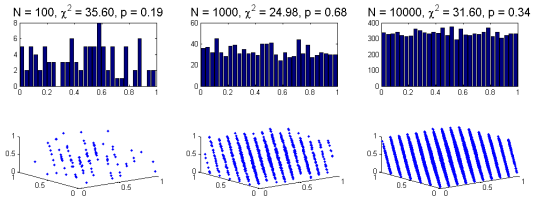

Figura 7: *Na parte superior: Histogramas gerados a partir do Método Linear Congruente Na parte inferior: Respectivos mapas de primeiro retorno*

ente é um método relativamente simples com apenas uma recursividade e uma semente inicial.

Em seguida, foi feito o Teste de Monte Carlo para obter o valor de π e compará-lo ao valor da literatura.Os resultados estão dados na figura 8

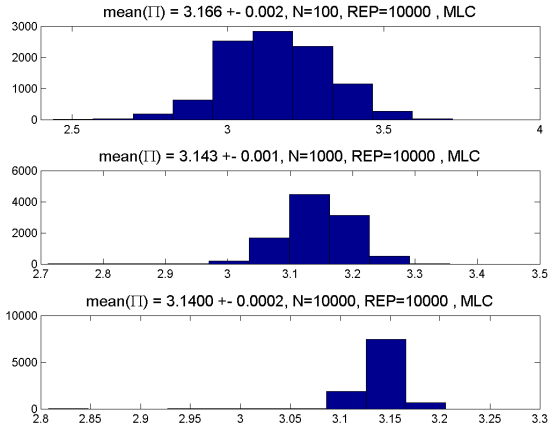

Figura 8: *Histograma dos valores de π para 10000 repetições*

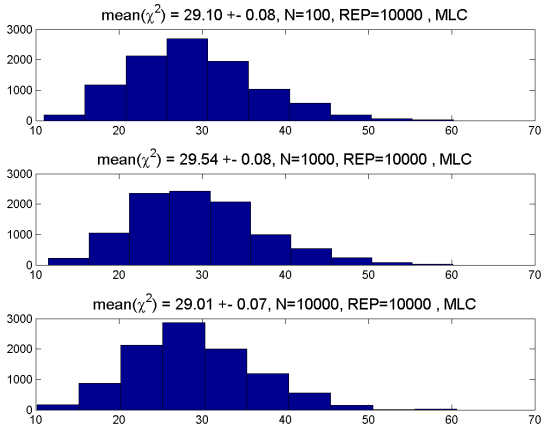

Figura 9: *Histograma para distribuição do χ^2 para 10 000 repetições pelo MLC*

2.2 Multiply-With-Carry

O Método Multiple With Carry é semelhante ao Método Linear Congruente. Neste último utilizávamos uma fórmula recursiva:

$$x_{n+1} = (a * x_{n-r} + c_{n-1}) mod(m) \qquad (15)$$

Em que as constantes a e n, e as sementes iniciais x_0 e c_0 são fornecidas pelo administrador do método (no caso anterior, utilizamos c=0). No método Multiply with Carry, a constante c também é dada recursivamente:

$$c_n = (a * x_{n-r} + c_{n-1})/m \; para \; n \geq 0 \qquad (16)$$

A vantagem do método Multiply With Carry deve-se a utilização de duas sementes iniciais, o que aumenta o período para geração do número aleatório.

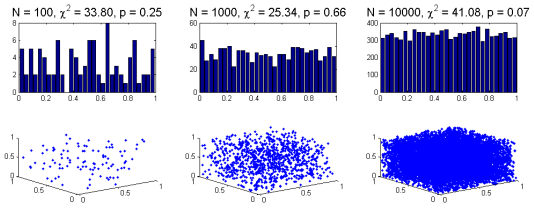

Figura 10: *Na parte superior: Histogramas gerados a partir do Método Multiply With Carry Na parte inferior: Respectivos mapas de primeiro retorno*

No Método Multiply With Carry é mais difícil identificar as faixas periódicas no mapa de primeiro retorno na figura 10, já que neste método temos duas sementes e duas recursividades.

Novamente foi feito o Teste de Monte Carlo para obter o valor de π e compará-lo ao valor da literatura. Os resultados estão dados na figura 11

2.3 Geradores Lagged Fibonacci

O método Lagged Fibonacci, como o próprio nome sugere, foi inspirado na sequência de Fibonacci, a qual apresentava valores a partir da soma de dois números anteriores na sequência:

$$x_n = x_{n-1} + x_{n-2} \qquad (17)$$

Generalizando, podemos trocar a operação de soma por outras operações:

$$x_n = x_{n-l} \star x_{n-k} \; tal \; que \; l > k > 0 \qquad (18)$$

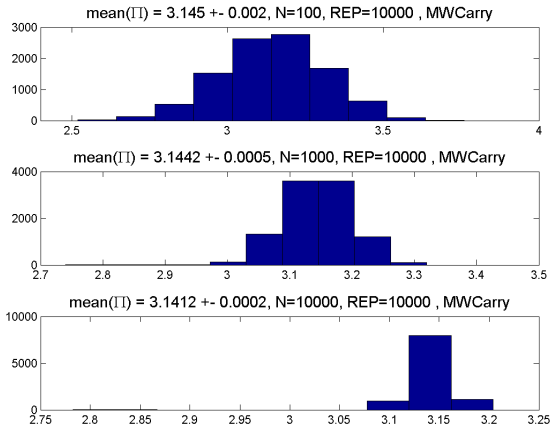

Figura 11: *Histograma dos valores de π para 10000 repetições*

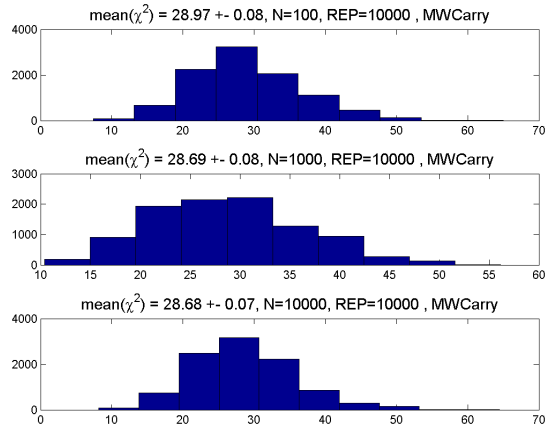

Figura 12: *Histograma para distribuição do χ^2 para 10 000 repetições pelo MWCarry*

Onde $\star$ representa as operações de soma, subtração, multiplicação ou operação de binários. A Multiplicative Lagged Fibonacci é a sequência em que se utiliza a operação de multiplicação. Para obter os resultados referentes a ela, utilizou-se a função do MATLAB que proporcionou os resultados dados nas figuras 13 e 14.

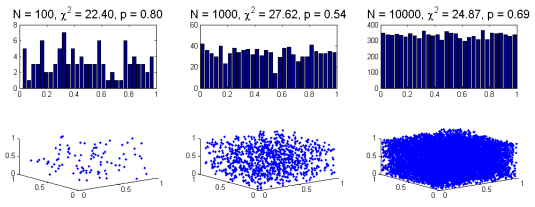

Figura 13: *Na parte superior: Histogramas gerados a partir do Método Multiplicative Lagged Fibonacci*
Na parte inferior: Respectivos mapas de primeiro retorno

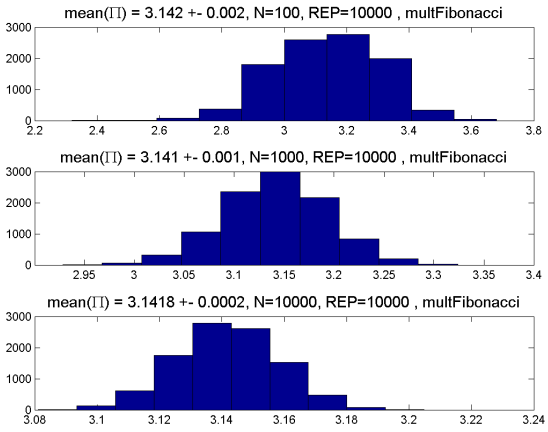

Figura 14: *Histograma dos valores de π para 10000 repetições*

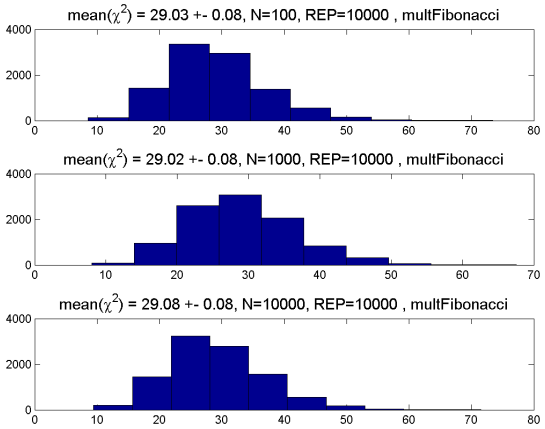

Figura 15: *Histograma para distribuição do χ^2 para 10 000 repetições pelo Método de Lagged Fibonnaci*

2.4 Mersenne Twister

O Mersenne Twister é um Gerador de Números Pseudo aleatórios (PRNG), seu nome Mersenne é devido ao seu período ser um Mersenne Prime[6], que são números primos estudados pelo francês Marin Mersenne, dados por uma potência de dois menos um $2^2 - 1$, o Mersenne Twister é atualmente um dos mais eficientes PRNGs, pois foi desenvolvido justamente para cobrir as falhas dos PRNGs anteriores com a proposta de ser um PRNG rápido e eficiente, ou seja, rápido para gerar números pseudo aleatórios e com um longo período. As versões do Mersenne Twister são o MT19937, cujo padrão de rotina é baseado em um comprimento de palavra de 32 bits já a outra versão, que é o MT19937-64, utiliza um comprimento de palavra de 64 bits. O Mersenne

[6] ??

Twister gera uma sequência de vetores de palavras(strings) como uniformes pseudo aleatórios inteiros entre 0 à $2^2 - 1$, onde w é a dimensão do vetor linha sobre o campo binário finito F_2. Ele é baseado na equação de recorrência (19)

$$x_{k+n} := x_{k+n} \otimes (x_k^u | x_{k+1}^l) A \tag{19}$$

Onde: n é o grau da recorrência. m é um inteiro, $1 \leq m \leq n$. A é uma matriz quadrada (w_x, w) (eq. 20) constante escolhida de moda a facilitar o produto de matrizes efetuado na equação (eq. 19).

$$A = \begin{bmatrix} 0 & 1 & & \\ 0 & 0 & 1 & \\ 0 & \dots & \dots & \\ & & & 1 \\ a_{w-1} & a_{w-2} & & a_0 \end{bmatrix} \tag{20}$$

K é $0, 1, 2, \dots$ x_n é um vetor de linha de uma palavra com tamanho w que é gerado com $k = 0$. $x_0, x_1, \dots . x_{n-1}$ são as sementes iniciais. x_{k+1}^l são os mais baixos ou mais à direita r bits de x_{k+1}. x_k^u são os mais altos ou os mais a esquerda $(w - r)$ bits de x_k. $\otimes$é referente a adição bit a bit do circuito XOR, da algebra Booleana. $|$ é referente a operação de concatenção. $(x_k^u | x_{k+1}^l)$ é o vetor obtido concatenando os vetores mais altos $(w - r)$ bits de x_k com o mais baixo (r)bits de x_{k+1}^l. Nessa ordem, então a matriz A pela direita por esse vetor. Finalmente a adição bit a bit $\otimes$ é para fazer a adição do vetor obtido com x_{k+m} , para assim gerar o próximo o próximo vetor x_{k+n}. A multiplicação de $(x_k^u | x_{k+1}^l) A$ pode ser mostrada utilizando uma operação de deslocamento de bit como mostrada a seguir(eq. 21):

$$xA = \begin{cases} x \gg 1 & \text{se } x_0 = 0 \\ (x \gg 1) \otimes a & \text{se } x_0 = 1 \end{cases} \tag{21}$$

Onde: x é o vetor linha $(x_0, x_1 \dots . x_{n-1})$. a é o vetor linha inferior de A. $\gg$ é o deslocamento bit a bit para a direita. Assim o cálculo da recorrência (19), é realizado através de deslocamentos de bits, os bit a bit são baseados nas operações lógicas: bit a bit $EXCLUSIVE-OR$, bit a bit OR e bit a bit AND. O Mersenne Twister possui um equidistribuição $(n - 1)$ dimensional, oque é um boa característica para um gerator de números pseudoaleatórios. Para o MT19937 com 32bits, por exemplo possui uma precisão de 32 bits para cada $1 \leq k \leq 623$, de acordo com o texte da k-distribuição.

Os resultados estão dados nas figuras 16 e 17.

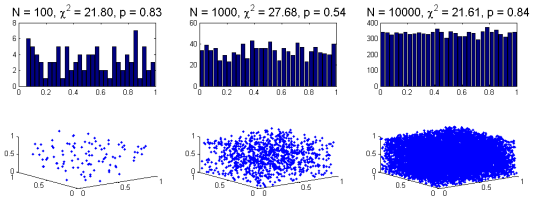

Figura 16: *Na parte superior: Histogramas gerados a partir do Método Mersenne Twister*
Na parte inferior: Respectivos mapas de primeiro retorno

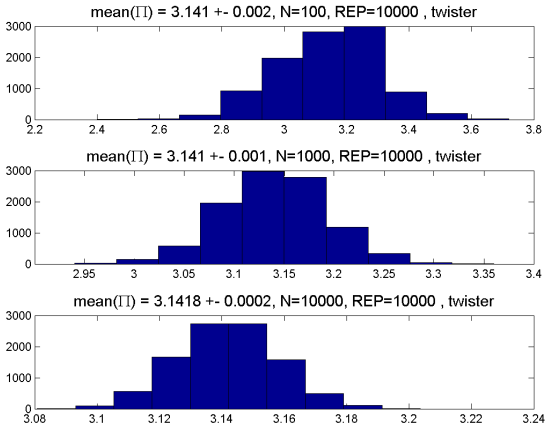

Figura 17: *Histograma dos valores de π para 10000 repetições*

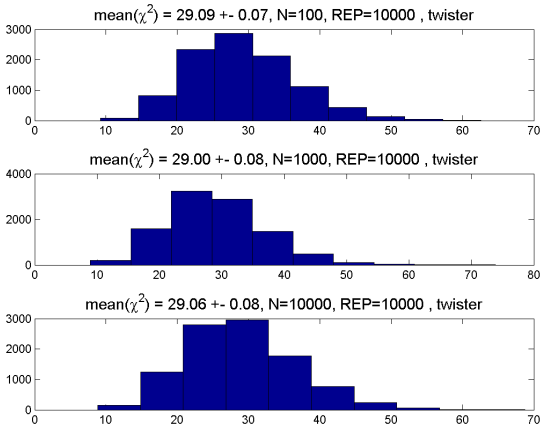

Figura 18: *Histograma para distribuição do χ^2 para 10 000 repetições pelo Twister*

2.5 dev/urandom

O dev/random é uma função geradora de números aleatórios, sendo a única deste trabalho

que gera números realmente aleatórios. É utilizada pelos sistemas operacionais do UNIX, como por exemplo o Linux que foi o primeiro sistema operacional a utilizar uma função que gerasse dados aleatórios para fins de segurança, criptografia, geração de senhas ou simulações. Esses dados aleatórios são oriundos do ruído do ambiente e obtidos através de drivers de dispositivo e de outras origens, como por exemplo, impulsos elétricos do sistema (o simples movimento do mouse pode fornecer o impulso necessário para gerar dados aleatórios). O número estimado de bits de ruído é mantido pelo gerador no pool de entropia, a partir do qual os números aleatórios são criados. Assim, quando solicitada, a função retorna os bytes aleatórios de acordo com número de bits estimados pelo pool de entropia. Por fim, esses dados são gerados através do comando "*RANDOM*" direto no terminal do linux, ou pela função "urand" do Octave, sendo que esta ultima foi a utilizada para a seguinte análise.

Os resultados estão dados nas figuras 19 e 20.

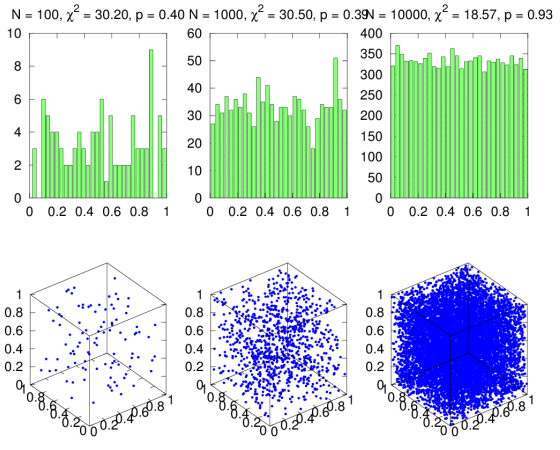

Figura 19: *Na parte superior: Histogramas gerados a partir do urand*
Na parte inferior: Respectivos mapas de primeiro retorno

2.6 Outros geradores

- Legacy MATLAB 5.0 uniform generator - Legacy MATLAB 5.0 normal generator - Legacy MATLAB 4.0 generator

3 Conclusão

Como explicitado anteriormente, a maioria dos geradores de números aleatórios apresentados eram pseudoaleatórios (apenas o gerador urand do Linux se aproxima de um gerador realmente aleatório). Ainda assim, são muito utilizados pois

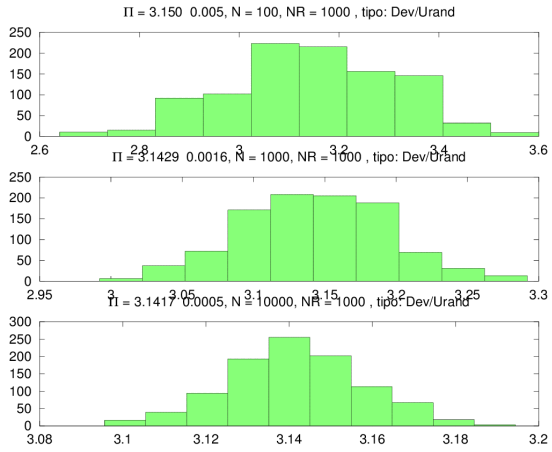

Figura 20: *Histograma dos valores de π para 10000 repetições*

a maioria possui períodos extremamente grandes. Em criptografia, sua utilização é essencial também, já que para decodificar algum tipo de sistema, ele não pode ser totalmente aleatório.

O gerador MLC, como verificado na análise de resultados, é o gerador mais simples e de menor periodicidade.

4 Bibliografia

RESENDE, Maria Fernanda Araujo de, SANTOS, Antonio Domingues dos, ITRI, Rosangela. *Efeito Fotoelétrico, Notas de Aula.* Laboratório de Física Moderna, 2015.

, conforme citado por Jain, [1991]

[Dudewicz e Karian (1985)]