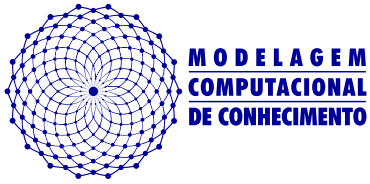

Dissertação de Mestrado

Geradores de Números Aleatórios, um estudo comparativo utilizando a linguagem R

Marcelo Queiroz de Assis Oliveira
marceloqao@gmail.com

Orientadores:

Dr. Alejandro C. Frery
Dr. Heitor Soares Ramos Filho

Maceió, Março de 2015

Marcelo Queiroz de Assis Oliveira

Geradores de Números Aleatórios, um estudo comparativo utilizando a linguagem R

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós-Graduação em Modelagem Computacional de Conhecimento do Instituto de Computação da Universidade Federal de Alagoas.

Orientadores:
Dr. Alejandro C. Frery

Dr. Heitor Soares Ramos Filho

Maceió, Março de 2015

Catalogação na fonte
Universidade Federal de Alagoas
Biblioteca Central
Divisão de Tratamento Técnico
Bibliotecária: Fabiana Camargo dos Santos

T693n Oliveira, Marcelo Queiroz de Assis.

Geradores de Números Aleatórios, um estudo comparativo utilizando a linguagem R / Marcelo Queiroz de Assis Oliveira. – 2015
73 f. : il.

Orientadores: Alejandro C. Frery.
Heitor Soares Ramos Filho.
Dissertação (Mestrado em Modelagem Computacional de Conhecimento) – Universidade Federal de Alagoas. Instituto de Computação. Maceió, 2015.

Bibliografia: f. 69–74.

1. Geradores de Números Aleatórios. 2. Testes Teóricos. 3. Testes Estatísticos.

CDU: 004.932:004.852 – VERIFICAR –

Membros da Comissão Julgadora da Dissertação de Mestrado de Marcelo Queiroz de Assis Oliveira, intitulada “Geradores de Números Aleatórios, um estudo comparativo utilizando a linguagem R”, apresentada ao Programa de Pós-Graduação em Modelagem Computacional de Conhecimento da Universidade Federal de Alagoas em 30 de marco de 2015, às 10h00min, na sala de aula do Mestrado em Modelagem Computacional de Conhecimento. Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Curso de Mestrado em Modelagem Computacional de Conhecimento do Instituto de Computação da Universidade Federal de Alagoas, aprovada pela comissão examinadora que abaixo assina.

Dr. Alejandro C. Frery – Orientador
Instituto de Computação
Universidade Federal de Alagoas

Dr. Heitor Soares Ramos Filho – Orientador
Instituto de Computação
Universidade Federal de Alagoas

Dr. Osvaldo Anibal Rosso – Examinador
Pesquisador Visitante – Instituto de Computação
Universidade Federal de Alagoas

Dr. Raydonal Ospina Martínez – Examinador
Departamento de Estatística
Universidade Federal de Pernambuco

Maceió, Março de 2015

RESUMO

Este trabalho busca comparar alguns geradores de números aleatórios bem conhecidos pela comunidade acadêmica. Geradores de números aleatórios são amplamente utilizados em algoritmos probabilísticos como "Simulated Annealing", Algoritmos Genéticos e GRASP, assumindo assim um importante papel na utilização destes. Primeiramente falaremos sobre o histórico dos geradores de números aleatórios e de geradores em ambientes paralelos. Em seguida faremos a avaliação de alguns geradores segundo as abordagens clássica e numa segunda abordagem baseada na teoria da informação, procurando, desta forma, uma melhoria nos geradores de números pseudoaleatórios. Na sequência, discutiremos a avaliação paralela dos geradores utilizando três geradores disponíveis no R através dos trinta e um testes disponíveis no pacote "Dieharder", além de realizar um teste baseado da teoria da informação, encontrar regiões de confiança no plano (H,C) (Entropia x Complexidade) utilizando como dados de entrada as sequências aleatórias oriundas dos geradores testados. Como contribuição, proporemos uma ferramenta web para realizar os testes de sequências aleatórias.
Palavras-chave: Geradores de Números Aleatórios. Testes Teóricos. Testes Estatísticos.

ABSTRACT

This work aims to compare some Random Numbers Generators well known by the Scientific Community. Random Numbers Generators (RNG's) are used in probabilistic algorithms, such the Simulated Annealing, Genetics algorithms and GRASP. Also, the use of a robust RNG method is key factor to the success of a simulation. To introduce, some theoretic concepts about generators will be showed **Keywords**: Random Number Generators. Theoretical tests.

Statistical Tests.

AGRADECIMENTOS

Marcelo Queiroz de A. Oliveira

LISTA DE FIGURAS

LISTA DE TABELAS

Lista de Equações

SUMÁRIO

1

Introdução

1.1 Definição do Problema

1.2 Revisão Bibliográfica

1.3 Contribuições

1.4 Riscos

Neste capítulo tratamos de aspectos introdutórios do trabalho como a Definição do Problema, uma Revisão Bibliográfica, Contribuições e Riscos envolvidos no mesmo, no próximo capítulo trataremos da metodologia utilizada do desenvolvimento do mesmo.

2

Fundamentação

ESTE capítulo tem como objetivo apresentar a revisão bibliográfica necessária para a realização deste trabalho.

2.1 Quão bons são os geradores de Números Pseudoaleatórios Disponíveis no R ?

No R há alguns geradores de números pseudoaleatórios bem como um pacote de testes, conhecido como Dieharder, para aferir a sua qualidade. Há por outro lado, uma metodologia para análise de séries temporais baseada em ferramentas da teoria da informação que se mostrou útil para avaliar geradores de números pseudoaleatórios. A proposta é avaliar os geradores disponíveis no R comparando os testes disponíveis no Dieharder com o teste baseado em ferramentas da teoria da informação.

2.2 O que são PRNGs

2.2.1 Geradores de Números Aleatórios?

A necessidade de números aleatórios e pseudoaleatórios surge em inúmeras aplicações, como aborda Knuth (1998) :

- *Simulação* - Quando um computador é usado para simular fenômenos naturais, números aleatórios são necessários para fazer as coisas de forma realística. Simulação abrange diversas áreas, desde o estudo de física nuclear (onde particulas são submetidas a colisões aleatórias) até pesquisa operacional (como, por exemplo, a taxa de pessoas que entram num aeroporto em intervalos aleatórios).

- *Amostragem* - É praticamente impossível examinar todos os possíveis casos, porém uma amostra aleatória provê um palpite sobre como é o comportamento típico do fenômeno em questão.
- *Análise Numérica* - Técnicas elaboradas para a solução de complexos problemas numéricos foram desenvolvidas utilizando-se números aleatórios.
- *Programação de Computadores* - Valores aleatórios são uma ótima fonte de dados para testar a eficácia de algoritimos computacionais.
- *Tomada de Decisão* - Existem relatos de que diversos executivos tomam suas decisões lançando moedas ou atirando dardos. Também há rumores de que professores universitários lançam suas notas de forma similar. Em alguns momentos é importante tomar decisões de forma não influenciada por qualquer agente externo.
- *Criptografia* - Uma fonte de bits não viesada é essencial para diversos tipos de comunicações seguras, quando os dados precisam ser mantidos em sigilo.
- *Estética* - Um pouco de aleatoriedade faz com que gráficos e músicas geradas por computador aparentem ser menos artificiais.
- *Diversão* - Rolar dados, embaralhar cartas, girar rodas de roletas, etc., são passatempos facinantes para muitos. Estes usos tradicionais de números aleatórios sugeriram o nome "Método de Monte Carlo".

Existem dois tipos básicos de geradores utilizados para produzir sequências aleatórias: geradores de números aleatórios (RNGs) e geradores de números pseudoaleatórios (PRNGs). Ambos podem produzir um fluxo de zeros e uns, que podem ser divididos em blocos ou subcorrentes de números aleatórios. Uma seqüência de bits aleatórios poderia ser interpretada como o resultado dos lançamentos de uma moeda justa, imparcial com os lados que são identificados como "0" e "1", onde cada caso tem uma probabilidade de exatamente ½. Além disso, o resultado de qualquer lançamento anterior não influencia futuros lançamentos. A moeda imparcial justa é, portanto, o gerador de fluxo de bits aleatórios perfeito. Todos os elementos da sequência são gerados independentemente uns dos outros, e o valor do elemento seguinte na sequência não pode ser previsto, independentemente do número de elementos que já foram produzidos. Obviamente, a utilização de moedas imparciais para fins de criptografia, modelagem computacional ou simulação é impraticável. No entanto, a saída hipotética de um gerador de uma seqüência realmente aleatória serve como um ponto de referência para a avaliação dos geradores de números aleatórios e pseudoaleatórios.

Números aleatórios e pseudoaleatórios gerados para quaisquer aplicações devem ser imprevisíveis. No caso de PRNG, se a semente é desconhecida, o próximo número na sequência de saída deve ser imprevisível, apesar de qualquer conhecimento de números aleatórios

anteriores na sequência. Esta propriedade é conhecida como imprevisibilidade para a frente. Também não deve ser viável para determinar a semente conhecendo os valores gerados (ou seja, a imprevisibilidade para trás também é obrigatória). Nenhuma correlação entre uma semente e qualquer valor gerado a partir da mesma devem ser evidentes; cada elemento da sequência deve parecer ser o resultado de um evento aleatório independente, cuja probabilidade é de 1/2. Para garantir a imprevisibilidade para a frente, o cuidado deve ser exercido na obtenção de sementes. Os valores produzidos por um PRNG são completamente previsíveis se a semente e algoritmo de geração são conhecidos. Uma vez que em muitos casos

2.2.2 Propriedades desejadas de um gerador ideal

Um gerador de números pseudoaleatórios deve possuir algumas propriedades que garantam sua qualidade para um leigo a construção de um gerador de números aleatórios pode parecer uma tarefa simples e alguns programadores tem demonstrado ser relativamente fácil escrever programas que gerem sequências de números aparentemente imprevisíveis. Entretanto, é bastante complexo escrever um bom programa que gere sequências satisfatórias, ou seja, uma sequência virtualmete infinita de números aleatórios estatisticamente independentes entre 0 e 1. Pois sequências aparentemente imprevisíveis não são necessariamente aleatórios

- *D. H. Lehmer (1951)*: "Uma sequência aleatória é uma vaga noção baseada na ideia de uma sequência onde cada termo é imprevisível e cujos dígitos passam em um certo número de testes, tradicionais com estatísticas ou dependendo do uso no qual a sequência será utilizada."

- *J. N. Franklin (1962)*: "A sequência é aleatória se possuir todas as propriedades compartilhadas por todas as infinitas sequências de amostras independentes de variáveis aleatórias sobre a distribuição uniforme.

A afirmação de Franklin essencialmente generaliza a de Lehmer ao dizer que a sequência precisa satisfazer *todos* os testes estatísticos. Esta definição não é precisa e uma interpretação sensata leva-nos a concluir que uma sequência aleatória simplesmente não existe! Portanto, iniciemos com a primeira e menos restritiva afirmação de Lehmer e tentemos torná-la mais precisa. O que realmente queremos é uma pequena lista de propriedades matemáticas, cada uma delas satisfeita por nossas noções intuitivas de uma sequência aleatória; além disso, a lista precisa ser completa o bastante para que qualquer sequência que a satisfaça possa ser considerada aleatória

2.2.3 Histórico

2.2.4 Geradores disponíveis no R

- a
- b
- c

2.3 Verificação clássica das propriedades dos geradores

2.3.1 Repetibilidade, portabilidade, eficiência computacional

2.3.2 Testes

A metodologia adotada para testar Geradores de Números Pseudoaleatórios é baseada em teste de hipóteses. Um teste de hipóteses é um procedimento para determinar se uma afirmação sobre uma característica de uma população é coerente. Neste caso, o teste envolve determinar quando uma específica sequência de zeros e uns é aleatória. Praticamente, apenas uma amostra da sequência de saída do PRNG é submetida a vários testes estatísticos. A tabela 2.1 lista algumas terminologias associadas com o teste de hipóteses.

Tabela 2.1: Teste de Hipóteses

Termo	Definição
Estatística de Teste	
Hipótese Nula	
Hipótese Alternativa	
Nível de Significancia	
Erro Tipo I	
Erro Tipo II	
Intervalo de Confiança	
p-valor	
Valor Crítico	

2.3.3 Testes anterior

2.3.4 Testes Diehard

2.3.5 Testes NIST

Fundado em 1991, o NIST (Instituto Nacional de Padrões e Tecnologia) é uma agência não regulatória do Departamento de Comércio dos Estados Unidos da América (EUA) que tem por missão promover a inovação e a competitividade nos EUA através da ciência de medidas, padrões e tecnologia de forma a alavancar a segurança econômica e melhorar a qualidade de vida do povo americano. A Divisão de Segurança de Computadores (CSD) e o Centro de Pesquisa em Segurança Computacional (CSRC) facilitam a ampla disseminação de práticas e ferramentas de segurança da informação, provendo recursos para a definição de padrões além de identificar recursos de segurança na web para suportar usuários na industria, governo e academia. CSRC é o portal de acesso primário para se ter acesso às publicações de segurança de computadores, padrões e instruções, além de outras informações relacionadas a segurança. Desde 1997, o Grupo de Trabalho Técnico em Geração de Números Aleatórios (RNG-TWG) tem trabalhado no desenvolvimento de uma bateria de testes estatísticos apropriados para a avaliação de geradores de números aleatórios e pseudoaleatórios utilizados em aplicações criptográficas. Os principais objetivos do grupo são:

- Desenvolvimento de uma bateria de testes estatísticos para detectar não aleatoriedade em sequencias binárias construidas através de geradores de números aleatórios e pseudoaleatórios utilizados em aplicações criptográficas;
- Produzir documentação e uma implementação em software destes testes;
- Prover auxílio no uso e aplicação destes testes.

Um total de quinze testes estatísticos foram desenvolvidos, implementados e avaliados:

- **Frequency (Monobits) Test** - The focus of the test is the proportion of zeroes and ones for the entire sequence. The purpose of this test is to determine whether that number of ones and zeros in a sequence are approximately the same as would be expected for a truly random sequence. The test assesses the closeness of the fraction of ones to ½, that is, the number of ones and zeroes in a sequence should be about the same.
- **Test For Frequency Within A Block** - The focus of the test is the proportion of zeroes and ones within M-bit blocks. The purpose of this test is to determine whether the frequency of ones is an M-bit block is approximately M/2.
- **Runs Test** - The focus of this test is the total number of zero and one runs in the entire sequence, where a run is an uninterrupted sequence of identical bits. A run of length

k means that a run consists of exactly k identical bits and is bounded before and after with a bit of the opposite value. The purpose of the runs test is to determine whether the number of runs of ones and zeros of various lengths is as expected for a random sequence. In particular, this test determines whether the oscillation between such substrings is too fast or too slow.

- **Test For The Longest Run Of Ones In A Block** - The focus of the test is the longest run of ones within M-bit blocks. The purpose of this test is to determine whether the length of the longest run of ones within the tested sequence is consistent with the length of the longest run of ones that would be expected in a random sequence. Note that an irregularity in the expected length of the longest run of ones implies that there is also an irregularity in the expected length of the longest run of zeroes. Long runs of zeroes were not evaluated separately due to a concern about statistical independence among the tests.

- **Random Binary Matrix Rank Test** - The focus of the test is the rank of disjoint submatrices of the entire sequence. The purpose of this test is to check for linear dependence among fixed length substrings of the original sequence.

- **Discrete Fourier Transform (Spectral) Test** - The focus of this test is the peak heights in the discrete Fast Fourier Transform. The purpose of this test is to detect periodic features (i.e., repetitive patterns that are near each other) in the tested sequence that would indicate a deviation from the assumption of randomness.

- **Non-Overlapping (Aperiodic) Template Matching Test** - The focus of this test is the number of occurrences of pre-defined target substrings. The purpose of this test is to reject sequences that exhibit too many occurrences of a given non-periodic (aperiodic) pattern. For this test and for the Overlapping Template Matching test, an m-bit window is used to search for a specific m-bit pattern. If the pattern is not found, the window slides one bit position. For this test, when the pattern is found, the window is reset to the bit after the found pattern, and the search resumes.

- **Overlapping (Periodic) Template Matching Test** - The focus of this test is the number of pre-defined target substrings. The purpose of this test is to reject sequences that show deviations from the expected number of runs of ones of a given length. Note that when there is a deviation from the expected number of ones of a given length, there is also a deviation in the runs of zeroes. Runs of zeroes were not evaluated separately due to a concern about statistical independence among the tests. For this test and for the Non-overlapping Template Matching test, an m-bit window is used to search for a specific m-bit pattern. If the pattern is not found, the window slides one bit position.

For this test, when the pattern is found, the window again slides one bit, and the search is resumed.

- **Maurer's Universal Statistical Test** - The focus of this test is the number of bits between matching patterns. The purpose of the test is to detect whether or not the sequence can be significantly compressed without loss of information. An overly compressible sequence is considered to be non-random.

- **Linear Complexity Test** - The focus of this test is the length of a generating feedback register. The purpose of this test is to determine whether or not the sequence is complex enough to be considered random. Random sequences are characterized by a longer feedback register. A short feedback register implies non-randomness.

- **Serial Test** - The focus of this test is the frequency of each and every overlapping m-bit pattern across the entire sequence. The purpose of this test is to determine whether the number of occurrences of the 2m m-bit overlapping patterns is approximately the same as would be expected for a random sequence. The pattern can overlap.

- **Approximate Entropy Test** - The focus of this test is the frequency of each and every overlapping m-bit pattern. The purpose of the test is to compare the frequency of overlapping blocks of two consecutive/adjacent lengths (m and m+1) against the expected result for a random sequence.

- **Cumulative Sum (Cusum) Test** - The focus of this test is the maximal excursion (from zero) of the random walk defined by the cumulative sum of adjusted (-1, +1) digits in the sequence. The purpose of the test is to determine whether the cumulative sum of the partial sequences occurring in the tested sequence is too large or too small relative to the expected behavior of that cumulative sum for random sequences. This cumulative sum may be considered as a random walk. For a random sequence, the random walk should be near zero. For non-random sequences, the excursions of this random walk away from zero will be too large.

- **Random Excursions Test** - The focus of this test is the number of cycles having exactly K visits in a cumulative sum random walk. The cumulative sum random walk is found if partial sums of the (0,1) sequence are adjusted to (-1, +1). A random excursion of a random walk consists of a sequence of n steps of unit length taken at random that begin at and return to the origin. The purpose of this test is to determine if the number of visits to a state within a random walk exceeds what one would expect for a random sequence.

- **Random Excursions Variant Test** - The focus of this test is the number of times that a particular state occurs in a cumulative sum random walk. The purpose of this test is

to detect deviations from the expected number of occurrences of various states in the random walk.

2.3.6 Testes Dieharder

Como substituto aos testes anteriores RGB reescreveu-os numa linguagem portável como C e acrescentou ao conjunto de testes disponíveis no Diehard alguns outros testes do NIST e mais alguns de sua autoria, listados na tabela 2.2 e explicados abaixo.

Tabela 2.2: Testes disponíveis no Dieharder

Seq	Nome	ID
1	diehard birthdays	0
2	diehard operm5	1
3	diehard rank 32x32	2
4	diehardrank 6x8	3
5	diehard bitstream	4
6	diehard opso	5
7	diehard oqso	6
8	diehard dna	7
9	diehard count 1s stream	8
10	diehard count 1s byte	9
11	diehard parking lot	10
12	diehard 2dsphere	11
13	diehard 3dsphere	12
14	diehard squeeze	13
15	diehard sums	14
16	diehard runs	15
17	diehard craps	16
18	marsaglia tsang gcd	17
19	sts monobit	100
20	sts runs	101
21	sts serial	102
22	rgb bitdist	200
23	rgb minimum distance	201
24	rgb permutations	202
25	rgb lagged sum	203
26	rgb kstest test	204
27	dab bytedistrib	205
28	dab dct	206
29	dab filltree	207
30	dab filltree2	208
31	dab monobit2	209

- **“Birthdays” test (modificado). Id(0)** - Cada teste determina o número de intervalos que combinam de 512 “aniversários” (por padrão) tomados num “ano” fictício de 24 bits (por padrão). Este processo é repetido (por padrão) 100 vezes e o resultado é acumulado em um histograma. Intervalos repetidos podem ser distribuidos em uma distribuição Poisson se o gerador em questão for aleatório o suficiente, e em uma Chi Quadrado com o p-valor avaliado relativamente à hipótese nula. É recomendado rodar este teste próximo ou com exatamente 100 amostras por p-valor com `-t 100`. Dois parametros adicionais foram incluidos. No Diehard, nms=512, porém isto pode ser variado e todas as fórmulas de Marsaglia continuam a funcionar. Pode ser ajustado para valores diferentes com `-x nmsvalue`. Similarmente, o parâmetro `nbits`pode ser 24, mas podemos fazê-lo assumir qualquer valor desde que seja menor ou igual a `rmax_bits = 32`. E pode ser atribuido qualquer valor com o parâmetro `-y nbits`. Ambos são padrão para os valores do Diehard se as opções `-x e -y` não forem utilizadas.

- **Diehard Overlapping 5-Permutations Test. Id(1)** - This is the OPERM5 test. It looks at a sequence of one million 32-bit random integers. Each set of five consecutive integers can be in one of 120 states, for the 5! possible orderings of five numbers. Thus the 5th, 6th, 7th, ... numbers each provide a state. As many thousands of state transitions are observed, cumulative counts are made of the number of occurences of each state. Then the quadratic form in the weak inverse of the 120x120 covariance matrix yields a test equivalent to the likelihood ratio test that the 120 cell counts came from the specified (asymptotically) normal distribution with the specified 120x120 covariance matrix (with rank 99). This version uses 1.000.000 integers, twice. Note that Dieharder runs the test 100 times, not twice, by default.

- **Diehard 32x32 Binary Rank Test. Id(2)** - This is the BINARY RANK TEST for 32×32 matrices. A random 32×32 binary matrix is formed, each row a 32bit random integer. The rank is determined. That rank can be from 0 to 32, ranks less than 29 are rare, and their counts are pooled with those for rank 29. Ranks are found for 40.000 such random matrices and a chisquare test is performed on counts for ranks $32, 31, 30 and \leqslant 29$. As always, the test is repeated and a KS test applied to the resulting p –values to verify that they are approximately uniform.

- **Diehard 6x8 Binary Rank Test. Id(3)** - This is the BINARY RANK TEST for 6×8 matrices. From each of six random 32bit integers from the generator under test, a specified byte is chosen, and the resulting six bytes form a 6×8 binary matrix whose rank is determined. That rank can be from 0 to 6, but ranks $0, 1, 2, 3$ are rare; their counts are pooled with those for rank 4. Ranks are found for 100.000 random matrices, and a chi-square test is performed on counts for ranks $6, 5 and \leqslant 4$. As always, the test is repeated and a KS test applied to the resulting p –values to verify that they are approximately uniform.

- **Diehard Bitstream Test. Id(4)** - The file under test is viewed as a stream of bits. Call them b_1,b_2,.... Consider an alphabet with two "letters", 0 and 1 and think of the stream of bits as a succession of 20-letter "words", overlapping. Thus the first word is $b_1\ b_2 \ldots b_{20}$, the second is $b_2\ b_3 \ldots b_{21}$, and so on. The bitstream test counts the number of missing 20letter (20bit) words in a string of 2(21) overlapping 20letter words. There are 2^{20} possible 20 letter words. For a truly random string of $2^{21}+19$ bits, the number of missing words j should be (very close to) normally distributed with mean 141.909 and sigma 428. Thus (j-141909)/428 should be a standard normal variate (z score) that leads to a uniform [0,1) pvalue. The test is repeated twenty times. NOTE WELL! The test is repeated 100 times by default in dieharder, but the size of the sample is fixed (tsamples cannot/should not be varied from the default). The sigma of this test REQUIRES the use of overlapping samples, and overlapping samples are not independent. If one uses the non-overlapping version of this test, sigma = 290 is used instead, smaller because now there are 2^{21} INDEPENDENT samples.

- **Diehard Overlapping Pairs Sparse Occupance (OPSO). Id (5)** - The OPSO test considers 2letter words from an alphabet of 1024 letters. Each letter is determined by a specified ten bits from a 32bit integer in the sequence to be tested. OPSO generates 2^{21} (overlapping) 2letter words (from $2^{21}+1$ "keystrokes") and counts the number of missing words that is 2letter words which do not appear in the entire sequence. That count should be very close to normally distributed with mean $141.909, \sigma 290$. Thus $(missingwrds 141909)/290$ should be a standard normal variable. The OPSO test takes 32 bits at a time from the test file and uses a designated set of ten consecutive bits. It then restarts the file for the next designated 10 bits, and so on. Note $2^{21} = 2097152$, tsamples cannot be varied.

- **Diehard Overlapping Quadruples Sparce Occupancy (OQSO) Test. Id(6)** - Similar, to OPSO except that it considers 4letter words from an alphabet of 32 letters, each letter determined by a designated string of 5 consecutive bits from the test file, elements of which are assumed 32bit random integers. The mean number of missing words in a sequence of 2^{21} fourletter words, ($2^{21}+3$ "keystrokes"), is again 141909, with $\sigma = 295$. The mean is based on theory, σ comes from extensive simulation. Note $2^{21} = 2097152$, tsamples cannot be varied.

- **Diehard DNA Test. Id(7)** - The DNA test considers an alphabet of 4 letters: `C`,`G`,`A`,`T`, determined by two designated bits in the sequence of random integers being tested. It considers 10letter words, so that as in OPSO and OQSO, there are 2^{20} possible words, and the mean number of missing words from a string of 2^{21} (over-lapping) 10letter words ($2^{21}+9$ "keystrokes") is 141909. The standard deviation σ=339 was determined as for OQSO by simulation. (σ for OPSO, 290, is the true value (to three places), not

determined by simulation. Note 2^{21} = 2097152. Note also that we don't bother with overlapping keystrokes (and sample more rands rands are now cheap).

- **Diehard Count the 1s (stream) (modified) Test. Id(8)** - Consider the file under test as a stream of bytes (four per 32 bit integer). Each byte can contain from 0 to 8 1's, with probabilities 1,8,28,56,70,56,28,8,1 over 256. Now let the stream of bytes provide a string of overlapping 5-letter words, each "letter" taking values A,B,C,D,E. The letters are determined by the number of 1's in a byte: 0, 1, or 2 yield A, 3 yields B, 4 yields C, 5 yields D and 6, 7 or 8 yield E. Thus we have a monkey at a typewriter hitting five keys with various probabilities (37,56,70,56,37 over 256). There are 5^5 possible 5letter words, and from a string of 256.000 (over-lapping) 5letter words, counts are made on the frequencies for each word. The quadratic form in the weak inverse of the covariance matrix of the cell counts provides a chisquare test: Q5-Q4, the difference of the naive Pearson sums of $(OBS-EXP)^2/EXP$ on counts for 5 and 4-letter cell counts.

- **Diehard Count the 1s Test (byte) (modified). Id(9)** - This is the COUNTTHE1's TEST for specific bytes. Consider the file under test as a stream of 32bit integers. From each integer, a specific byte is chosen, say the leftmost: bits 1 to 8. Each byte can contain from 0 to 8 1's, with probabilitie 1, 8, 28, 56, 70, 56, 28, 8, 1 over 256. Now let the specified bytes from successive integers provide a string of (overlapping) 5letter words, each "letter" taking values A,B,C,D,E. The letters are determined by the number of $1's$, in that byte: $0,1,or2 \rightarrow A,3 \rightarrow B,4 \rightarrow C,5 \rightarrow D,and6,7or8 \rightarrow E$. Thus we have a monkey at a typewriter hitting five keys with with various probabilities: 37, 56, 70, 56, 37 over 256. There are 5^5 possible 5letter words, and from a string of 256.000 (overlapping) 5letter words, counts are made on the frequencies for each word. The quadratic form in the weak inverse of the covariance matrix of the cell counts provides a chisquare test: Q5Q4, the difference of the naive Pearson sums of $(OBS-EXP)^2/EXP$ on counts for 5 and 4-letter cell counts. Note: We actually cycle samples over all 031 bit offsets, so that if there is a problem with any particular offset it has a chance of being observed. One can imagine problems with odd offsets but not even, for example, or only with the offset 7. tsamples and psamples can be freely varied, but you'll likely need $tsamples \gg 100.000$ to have enough to get a reliable kstest result.

- **Diehard Parking Lot Test (modified). Id(10)** - This tests the distribution of attempts to randomly park a square car of length 1 on a 100x100 parking lot without crashing. We plot n (number of attempts) versus k (number of attempts that didn't "crash"because the car squares overlapped and compare to the expected result from a perfectly random set of parking coordinates. This is, alas, not really known on theoretical grounds so instead we compare to n=12,000 where k should average 3523 with sigma 21.9 and is very close to normally distributed. Thus (k-3523)/21.9 is a standard normal variable,

which converted to a uniform p-value, provides input to a KS test with a default 100 samples.

- **Diehard Minimum Distance (2d Circle) Test Id(11)** - It does this 100 times: choose n = 8.000 random points in a square of side 10.000. Find d, the minimum distance between the $(n^2 - n)/2$ pairs of points. If the points are truly independent uniform, then d^2, the square of the minimum distance should be (very close to) exponentially distributed with mean 0.995. Thus $1 - exp(-d^2/0.995)$ should be uniform on [0,1) and a KSTEST on the resulting 100 values serves as a test of uniformity for random points in the square. Test $numbers = 0 mod 5$ are printed but the KSTEST is based on the full set of 100 random choices of 8.000 points in the $10.000x10.000$ square. This test uses a fixed number of samples tsamples is ignored. It also uses the default value of 100 psamples in the final KS test, for once agreeing precisely with Diehard.

- **Diehard 3d Sphere (Minimum Distance) Test. Id(12)** - Choose 4.000 random points in a cube of edge 1.000. At each point, center a sphere large enough to reach the next closest point. Then the volume of the smallest such sphere is (very close to) exponentially distributed with mean $120\pi/3$. Thus the radius cubed is exponential with mean 30. (The mean is obtained by extensive simulation). The 3DSPHERES test generates 4.000 such spheres 20 times. Each min radius cubed leads to a uniform variable by means of $1 - exp(-r^3/30)$, then a KSTEST is done on the 20 p-values. This test ignores tsamples, and runs the usual default 100 psamples to use in the final KS test.

- **Diehard Squeeze Test. Id(13)** - Random integers are floated to get uniforms on [0,1). Starting with $k = 2^{31}$=2147483647, the test finds j, the number of iterations necessary to reduce k to 1, using the reduction $k = ceiling(k \times U)$, with U provided by floating integers from the file being tested. Such j's are found 100.000 times, then counts for the number of times j was $\leq 6, 7, \ldots, 47, \geq 48$ are used to provide a chisquare test for cell frequencies.

- **Diehard Sums Test Id(14)** - Integers are floated to get a sequence $U(1), U(2), \ldots$ of uniform [0,1) variables. Then overlapping sums, $S(1) = U(1) + \cdots + U(100), S2 = U(2) + \cdots + U(101), \ldots$ are formed. The S's are virtually normal with a certain covariance matrix. A linear transformation of the S's converts them to a sequence of independent standard normals, which are converted to uniform variables for a KSTEST. The pvalues from ten KSTESTs are given still another KSTEST. Comments At this point I think there is rock solid evidence that this test is completely useless in every sense of the word. It is broken, and it is so broken that there is no point in trying to fix it. The problem is that the transformation above is not linear, and doesn't work. Don't use it. For what it is worth, rgb_lagged_sums with ntuple 0 tests for exactly the same thing, but scalably

and reliably without the complication of overlapping samples and covariance. Use it instead.

- **Diehard Runs Test. Id(15)** - This is the RUNS test. It counts runs up, and runs down, in a sequence of uniform [0,1) variables, obtained by floating the 32-bit integers in the specified file. This example shows how runs are counted: .123,.357,.789,.425,.224,.416,.95 contains an up-run of length 3, a down-run of length 2 and an up-run of (at least) 2, depending on the next values. The covariance matrices for the runs-up and runs-down are well known, leading to chisquare tests for quadratic forms in the weak inverses of the covariance matrices. Runs are counted for sequences of length 10.000. This is done ten times. Then repeated. In Dieharder sequences of length tsamples = 100000 are used by default, and 100 p-values thus generated are used in a final KS test.

- **Diehard Craps Test. Id(16)** - This is the CRAPS TEST. It plays 200.000 games of craps, finds the number of wins and the number of throws necessary to end each game. The number of wins should be (very close to) a normal with mean 200000p and variance 200000p(1-p), with p=244/495. Throws necessary to complete the game can vary from 1 to infinity, but counts for all > 21 are lumped with 21. A chi-square test is made on the number-of-throws cell counts. Each 32-bit integer from the test file provides the value for the throw of a die, by floating to [0,1), multiplying by 6 and taking 1 plus the integer part of the result.

- **Marsaglia and Tsang GCD Test. Id(17)** - 10^7 tsamples (default) of uint rands u, v are generated and two statistics are generated: their greatest common divisor (GCD) (w) and the number of steps of Euclid's Method required to find it (k). Two tables of frequencies are thus generated - one for the number of times each value for k in the range 0 to 41 (with counts greater than this range lumped in with the endpoints). The other table is the frequency of occurrence of each GCD with k is be distributed approximately binomially, but this is useless for the purposes of performing a stringent test. Instead four "good" RNGs (gfsr4, mt19937_1999, rndlxs2, taus2) were used to construct a simulated table of high precision probabilities for k (a process that obviously begs the question as to whether or not THESE generators are "good" wrt the test). At any rate, they produce very similar tables and pass the test with each other's tables (and are otherwise very different RNGs). The table of probabilities for the gcd distribution is generated dynamically per test (it is easy to compute). Chisq tests on both of these binned distributions yield two pvalues per test, and 100 (default) pvalues of each are accumulated and subjected to final KS tests and displayed in a histogram.

- **STS Monobit Test. Id(100)** - Very simple. Counts the 1 bits in a long string of random uints. Compares to expected number, generates a p-value directly from erfc(). Very ef-

fective at revealing overtly weak generators; Not so good at determining where stronger ones eventually fail.

- **STS Runs Test. Id(101)** - Counts the total number of 0 runs + total number of 1 runs across a sample of bits. Note that a 0 run must begin with 10 and end with 01. Note that a 1 run must begin with 01 and end with a 10. This test, run on a bitstring with cyclic boundary conditions, is absolutely equivalent to just counting the 01 + 10 bit pairs. It is therefore totally redundant with but not as good as the rgb_bitdist() test for 2-tuples, which looks beyond the means to the moments, testing an entire histogram of 00, 01, 10, and 11 counts to see if it is binomially distributed with p = 0.25.

- **STS Serial Test. Id(102)** - Accumulates the frequencies of overlapping n-tuples of bits drawn from a source of random integers. The expected distribution of n-bit patterns is multinomial with p = $2^{(-n)}$ e.g. the four 2-bit patterns 00 01 10 11 should occur with equal probability. The target distribution is thus a simple chisq with $2^n - 1$ degrees of freedom, one lost due to the constraint that: p_00 + p_01 + p_01 + p_11 = 1 With overlap, though the test statistic is more complex. For example, given a bit string such as 0110100111000110 without overlap, it becomes 01|10|10|01|11|00|01|10 and we count 1 00, 3 01s, 3 10s, and 1 11. WITH overlap we get all of these patterns as well as (with cyclic wrap): 0|11|01|00|11|10|00|11|0 and we count 4 00s, 4 01s, 4 10s, and 3 11s. There is considerable covariance in the bit frequencies and a simple chisq test no longer suffices. The STS test uses target statistics that are valid for overlapping samples but which require multiple orders to generate. It is much easier to write a test that doesn't use overlapping samples and directly checks to ensure that the distribution of bit ntuples is consistent with a multinomial distribution with uniform probability $p = 1/2^n$, e.g. 1/8 for n = 3 bit, 1/16 for n = 4 bit NON-overlapping samples, and the rgb_bitdist is just such a test. This test doesn't require comparing different orders. An open research question is whether or not test sensitivity significantly depends on managing overlap testing software RNGs where it is presumed that generation is cheap and unlimited. This question pertains to related tests, such as overlapping permutations tests (where non-overlapping permutation tests are isomorphic to non-overlapping frequency tests, fairly obviously). This test does all the possible bitlevel tests from n=1 to n=24 bits (where n=1 is basically sts_monobit, and n=2 IMO is redundant with sts_runs). However, if I understand things correctly it is not possible to fail a 2 bit test and pass a 24 bit test, as if 2 bits are biased so that (say) 00 occurs a bit too often, then 24 bit strings containing 00's MUST be imbalanced as well relative to ones that do not, so we really only need to check n=24 bit results to get all the rest for free, so to speak.

- **RGB Bit Distribution Test. Id(200)** - Accumulates the frequencies of all n-tuples of bits in a list of random integers and compares the distribution thus generated with the

theoretical (binomial) histogram, forming chisq and the associated p-value. In this test n-tuples are selected without WITHOUT overlap (e.g. 01|10|10|01|11|00|01|10) so the samples are independent. Every other sample is offset modulus of the sample index and ntuple_max. This test must be run with -n ntuple for ntuple > 0. Note that if ntuple > 12, one should probably increase tsamples so that each of the 2^{ntuple} bins should end up with an average of around 30 occurrences. Note also that the memory requirements and CPU time requirements will get quite large by e.g. ntuple = 20 - use caution when sampling the distribution of very large ntuples.

- **THE GENERALIZED MINIMUM DISTANCE TEST. Id(201)** - This is the generalized minimum distance test, based on the paper of M. Fischler in the doc directory and private communications. This test utilizes correction terms that are essential in order for the test not to fail for large numbers of trials. It replaces both diehard_2dsphere.c and diehard_3dsphere.c, and generalizes the test itself so that it can be run for any d = 2,3,4,5. There is no fundamental obstacle to running it for d = 1 or d 5, but one would need to compute the expected overlap integrals (q) for the overlapping d-spheres in the higher dimensions. Note that in this test there is no real need to stick to the parameters of Marsaglia. The test by its nature has three controls: n (the number of points used to sample the minimum distance) which determines the granularity of the test - the approximate length scale probed for an excess of density; p, the usual number of trials; and d the dimension. As Fischler points out, to actually resolve problems with a generator that had areas 20% off the expected density (consistently) in $d = 2, n = 8000$ (Marsaglia's parameters) would require around 2500 trials, where p = 100 (the old test default) would resolve only consistent deviations of around 1.5 times the expected density. By making both of these user selectable parameters, dieharder should be able to test a generator pretty much as thoroughly as one likes subject to the generous constraints associated with the eventual need for still higher order corrections as n and p are made large enough.

- **RGB Permutations Test. Id(202)** - This is a non-overlapping test that simply counts order permutations of random numbers, pulled out n at a time. There are n! permutations and all are equally likely. The samples are independent, so one can do a simple chisq test on the count vector with n! - 1 degrees of freedom. This is a poor-man's version of the overlapping permutations tests, which are much more difficult because of the covariance of the overlapping samples.

- **RGB Lagged Sums Test. Id(203)** - This package contains many very lovely tests. Very few of them, however, test for lagged correlations – the possibility that the random number generator has a bitlevel correlation after some fixed number of intervening bits. The lagged sums test is therefore very simple. One simply adds up uniform deviates

sampled from the rng, skipping lag samples in between each rand used. The mean of tsamples samples thus summed should be 0,5*tsamples. The standard deviation should be sqrt(tsamples/12). The experimental values of the sum are thus converted into a p-value (using the erf()) and a ks-test applied to psamples of them.

- **The Kolmogorov-Smirnov Test Test. Id(204)** - This test generates a vector of tsamples uniform deviates from the selected rng, then applies an Anderson-Darling or Kuiper KS test to it to directly test for uniformity. The AD version has been symmetrized to correct for weak left bias for small sample sets; Kuiper is already ring-symmetric on the interval. The AD code corresponds roughly to what is in R (thanks to a correction sent in by David Bauer). As always, the test is run pvalues times and the (same) KS test is then used to generate a final test pvalue, but the real purpose of this test is to test ADKS and KKS, not to test rngs. This test clearly reveals that kstests run on only 100 test values (tsamples, herein) are only approximately accurate; their pvalues are distinctly high-biased (but less so than Kuiper or KS before the fix). This bias is hardly visible for less than 1000 trivals (psamples, herein) but will constently cause failure for -t 100, -p 10000 or higher. For -t 1000, it is much more difficult to detect, and the final kstest is approximately valid for the test in question.

- **DAB Byte Distribution Test. Id(205)** - Extract n independent bytes from each of k consecutive words. Increment indexed counters in each of n tables. (Total of 256 × n counters.) Currently, n=3 and is fixed at compile time. If n>=2, then the lowest and highest bytes will be used, along with n-2 bytes from the middle. If the generator's word size is too small, overlapped bytes will be used. Current, k=3 and is fixed at compile time. Use a basic chisq fitting test (chisq_pearson) for the p-value. Previous version also used a chisq independence test (chisq2d); it was found to be slightly less sensitive. I envisioned this test as using a small number of samples and large number of separate tests. Experiments so far show that keeping -p 1 and increasing -t performs best.

- **DCT (Frequency Analysis) Test. Id(206)** - This test performs a Discrete Cosine Transform (DCT) on the output of the RNG. More specifically, it performs tsamples transforms, each over an independent block of ntuple words. If tsamples is large enough, the positions of the maximum (absolute) value in each transform are recorded and subjected to a chisq test for uniformity/independence. [1] (A standard type II DCT is used.) If tsamples is smaller than or equal to 5 times ntuple then a fallback test will be used, whereby all DCT values are converted to p-values and tested for uniformity via a KS test. This version is significantly less sensitive, and is not recommended. Power: With the right parameters, this test catches more GSL generators than any other; however, that count is biased by each of the randomNNN generators having three copies. Limitations: ntuple is required to be a power of 2, because a radix 2 algorithm is used to calculate the

DCT. False positives: targets are (mostly) calculated exactly, however it will still return false positives when ntuple is small and tsamples is very large. For the default ntuple value of 256, I get bad scores with about 100 million or more tsamples (psamples set to 1). [1] The samples are taken as unsigned integers, and the DC coefficient is adjusted to compensate for this.

- **DAB Fill Tree Test. Id(207)** - This test fills small binary trees of fixed depth with words from the the RNG. When a word cannot be inserted into the tree, the current count of words in the tree is recorded, along with the position at which the word would have been inserted. The words from the RNG are rotated (in long cycles) to better detect RNGs that may bias only the high, middle, or low bytes. The test returns two p-values. The first is a Pearson chi-sq test against the expected values (which were estimated empirically). The second is a Pearson chi-sq test for a uniform distribution of the positions at which the insert failed. Because of the target data for the first p-value, ntuple must be kept at the default (32).

- **DAB Fill Tree 2 Test. Id(208)** - Bit version of Fill Tree test. This test fills small binary trees of fixed depth with "visited"markers. When a marker cannot be placed, the current count of markers in the tree and the position that the marker would have been inserted, if it hadn't already been marked. For each bit in the RNG input, the test takes a step right (for a zero) or left (for a one) in the tree. If the node hasn't been marked, it is marked, and the path restarts. Otherwise, the test continues with the next bit. The test returns two p-values. The first is a Pearson chi-sq test against the expected values (which were estimated empirically. The second is a Pearson chi-sq test for a uniform distribution of the positions at which the insert failed. Because of the target data for the first p-value, ntuple must be kept at the default (128).

- **DAB Monobit 2 Test. Id(209)** - Block-monobit test. Since we don't know what block size to use, try multiple block sizes. In particular, try all block sizes of 2^k words, where k=0 . . . n. The value of n is calculated from the word size of the generator and the sample size used, and is shown as ntuple.

2.4 Verificação das propriedades com ferramentas da teoria da informação

2.4.1 aaa

Neste capítulo tratamos da Revisão Bibliográfica realizada para o desenvolvimento do trabalho, no capítulo seguinte tratamos da metodologia utilizada do desenvolvimento do mesmo.

3

Metodologia

ESTE capítulo tem como objetivo apresentar os materiais e métodos utilizados no trabalho. Como principal objetivo, este capítulo visa fornecer subsídios suficientes e para que o mesmo possa ser reproduzido e a continuidade do mesmo na pesquisa e desenvolvimento dos problemas deixados em aberto possa ser alcançada.

3.1 Materiais e Métodos

Em relação a fundamentação teórica, utilizou-se como principal fonte de pesquisa a área de indexação de periódicos científicos ISI *Web of Knowledge*, onde foram obtidas a grande maioria das referências, usando como parâmetros o fator de impacto dos periódicos pesquisados, a quantidade de citações de cada publicação, o grau de relevância para o tema pesquisado e o nível de produtividade (fator-H) dos autores envolvidos. O apoio em livros, surveys, lecture notes e ferramentas complementares de busca, como o *google acadêmico* foram utilizadas para complementar esta pesquisa.

Para organizar, catalogar e facilitar a consulta a todo material obtido, as referências foram gerenciadas com a ferramenta *Mendeley*. Um gerenciador de referências bibliográficas multiplataforma gratuito que permite organizar de maneira centralizada vários vínculos entre as referências utilizadas, bem como visualizar e anotar as mesmas dentro da própria ferramenta. Quanto à editoração eletrônica do trabalho, fez-se uso da plataforma LaTeX, com editor de textos *Kile* de código aberto. Este trabalho foi desenvolvido num equipamento com as seguintes configurações:

Arquitetura	`Intel i7 64 bits`
S.O.	Linux Mint 17.1 - kernel 3.13.0 – 43 – $generic$
Editor	Kile versão 2.1.3 e LaTeXtexlive 2013.20140215 – 1

Do ponto de vista técnico deste trabalho, com ênfase em Geradores de Números Pseudoaleatórios, é utilizada a plataforma de análise estatística R . Esta plataforma foi desenvolvida originalmente por Ross Ihaka e Robert Gentleman, com o intuito de ser uma linguagem de código aberto voltada para a análise estatística e, consequentemente, a precisão numérica com fortes características funcionais (**?**). Por este motivo, ela será utilizada para a geração e manipulação das sequências pseudoaleatórias, análise dos dados e geração dos gráficos desse trabalho. A precisão numérica desta ferramenta, sendo aferida por Almiron et al. (2009), é adequada para essa abordagem.

As ferramentas utilizadas no desenvolvimento deste trabalho, são preferencialmente multiplataforma e código aberto com licença de uso *GNU General Public License* (GPL).

Todos esses aplicativos, métodos e informações obtidas, forneceram grandes contribuições para o traçado da linha mestra deste trabalho, indicando que o mesmo está na fronteira do conhecimento produzindo um estado da arte fidedigno aos temas e ferramentas adotadas para norteá-lo.

Neste capítulo tratamos da metodologia utilizada do desenvolvimento do trabalho, no capítulo seguinte analisaremos os impactos esperados com a realização do mesmo.

4

Resultados Esperados

UMA vez com a modelagem concluída e os dados para simulação, serão realizados os experimentos afim de comprovar a eficácia do modelo proposto.

4.1 Resultados Esperados

Neste capítulo tratamos os resultados esperados com a realização do trabalho enquanto que no próximo faremos a conclusão alcançada com o mesmo.

5

Conclusão

A análise dos dados deve evidenciar a importância de trabalhar a redução dos dados numa rede de sensores, visando a redução do consumo de energia e maximizando o tempo de vida da rede.

5.1 Impactos Esperados

A análise dos dados deve evidenciar a importância de trabalhar a redução dos dados numa rede de sensores, visando a redução do consumo de energia e maximizando o tempo de vida da rede.

Apêndice A

Ambiente Reprodutível e Computacional

O?

REFERÊNCIAS BIBLIOGRÁFICAS

Almiron, M. G., Almeida, E. S. & Miranda, M. N. (2009), 'The reliability of statistical functions in four software packages freely used in numerical computation', *Brazilian Journal of Probability and Statistics* **23**(2), 107–119.

Knuth, D. E. (1998), *The Art of Computer Programming, Volume 2: (2Nd Ed.) Seminumerical Algorithms*, Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA.

Este trabalho foi redigido em LaTeX utilizando uma modificação do estilo IC-UFAL. As referências bibliográficas foram preparadas no Mendeley e administradas pelo BibTeX com o estilo LaCCAN. O texto utiliza fonte Fourier-GUT*enberg* e os elementos matemáticos a família tipográfica Euler Virtual Math, ambas em corpo de 12 pontos.