

UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
UNIDADE ACADÊMICA DE SISTEMAS E COMPUTAÇÃO
INSTITUTO DE ESTUDOS EM COMPUTAÇÃO E
INFORMAÇÃO QUÂNTICAS

RELATÓRIO FINAL DE ESTÁGIO INTEGRADO

Construção de uma aplicação para analisar a qualidade de geradores de números aleatórios e pseudo-aleatórios.

ELLOÁ B. GUEDES DA COSTA
(ESTAGIÁRIA)

FRANCISCO M. DE ASSIS, DR.
BERNARDO LULA JR., DR.
(ORIENTADORES)

GILSON O. SANTOS, MSC.
(SUPERVISOR TÉCNICO)

CAMPINA GRANDE, 22 JULHO DE 2009

**CONSTRUÇÃO DE UMA APLICAÇÃO PARA ANALISAR A
QUALIDADE DE GERADORES DE NÚMEROS ALEATÓRIOS E
PSEUDO-ALEATÓRIOS**

APROVADO EM 22 DE JULHO DE 2009

BANCA EXAMINADORA

PROF. DR. FRANCISCO M. DE ASSIS

ORIENTADOR ACADÊMICO

PROF. DR. BERNARDO LULA JR.

ORIENTADOR ACADÊMICO

PROFA. DRA. JOSEANA MACÊDO FECHINE

MEMBRO DA BANCA EXAMINADORA

PROF. DR. TIAGO LIMA MASSONI

MEMBRO DA BANCA EXAMINADORA

Agradecimentos

“Find your passion and follow it. (...)

Your passion must come from the things that fuel you from the inside.”

Randy Pausch

“Existem obstáculos, pontos discutíveis, decepções, mas isso significa apenas aquilo que já sabíamos antes, que nada será dado de presente a você, que pelo contrário terá de lutar, motivo a mais para ser ativo e não abatido.”

Franz Kafka

Ao final de mais uma etapa em minha vida, um misto de nostalgia dos dias passados e desejo por novos desafios, o elo entro o caminho já percorrido e a estrada desconhecida que se põe a frente.

Da nostalgia, a gratidão aos meus pais e irmãos pelo amor e pela motivação constantes, à minha avó pela acolhida, aos meus professores e orientadores pelo conhecimento e exemplo, aos meus amigos pelo companheirismo, pelos risos e pela família que se fizeram nesta nova cidade. Aos projetos, provas e compromissos também cabem agradecimentos por terem me acrescentado e fortalecido meu desejo e empenho em continuar sempre aprendendo.

Dos novos desafios, a imensa vontade de continuar caminhando, crescendo, e poder sentir outras vezes a doce nostalgia de cada etapa vencida.

De todos os tempos, a gratidão pela iluminação e proteção divinas sempre presentes.

Elloá B. Guedes

Julho de 2009.

Apresentação

Como parte das exigências do curso de Ciência da Computação, da Universidade Federal de Campina Grande, para cumprimento da disciplina de Estágio Integrado, apresenta-se o relatório de estágio que descreve as atividades desempenhadas pela aluna *Elloá Barreto Guedes da Costa*, regularmente matriculada na respectiva disciplina durante o semestre letivo 2009.1 (março de 2009 a julho de 2009).

O estágio foi realizado no Instituto de Estudos em Computação e Informação Quânticas, IQuanta, sob a orientação acadêmica do professor Dr. Francisco Marcos de Assis, do Departamento de Engenharia Elétrica da UFCG, e do professor Dr. Bernardo Lula Jr., do Departamento de Sistemas e Computação da UFCG, e sob supervisão técnica de Gilson Oliveira dos Santos, mestre pela Universidade Federal de Campina Grande e doutorando do Curso de Pós-Graduação em Engenharia Elétrica pela UFCG.

O conteúdo deste relatório está distribuído conforme descrição a seguir: no Capítulo 1 são apresentados os objetivos do estágio; no Capítulo 2 é descrito o ambiente no qual o estágio foi realizado; nos Capítulos 3 e 4 é apresentada a fundamentação teórica; no Capítulo 5 há a descrição das atividades realizadas e no Capítulo 6 são apresentadas as considerações finais deste trabalho, seguidas das referências bibliográficas e apêndices, cuja leitura será referenciada ao longo do relatório.

Resumo

Este relatório descreve as atividades desenvolvidas pela aluna Elloá B. Guedes da Costa durante a disciplina de Estágio Integrado que resultaram na implementação de uma aplicação denominada *Sieve*, destinada a análise de qualidade de geradores de números aleatórios e pseudo-aleatórios. A ferramenta resultante, desenvolvida em Java, possui sete testes estatísticos configuráveis, os quais podem ser aplicados à seqüências numéricas, oriundas de geradores, com o intuito de verificar se as mesmas são realizações de uma distribuição uniforme discreta no intervalo $[0, 1]$. O *Sieve* é adequado, principalmente, para fins de simulação, gera relatórios em formato *PDF*, e também possui uma interface gráfica de fácil utilização.

Sumário

1	Introdução	1
1.1	Contextualização	1
1.2	Objetivos Geral	2
1.3	Objetivos Específicos	2
2	Ambiente de Estágio	4
2.1	Ambiente de Estágio	4
2.2	Recursos de Hardware e Software	5
2.3	Orientação Acadêmica e Supervisão Técnica	5
2.3.1	Orientação Acadêmica	6
2.3.2	Supervisão Técnica	6
3	Geração de Números Aleatórios e Pseudo-Aleatórios	7
3.1	Geradores de Números Aleatórios	7
3.2	Geradores de Números Pseudo-Aleatórios	8
3.2.1	Geradores Congruentes Lineares Simples	10
3.2.2	Geradores Congruentes Não-Lineares	13
3.2.3	Geradores com Registradores de Deslocamento Retro-Alimentados	16
4	Análise Estatística de Geradores Aleatórios e Pseudo-Aleatórios	18
4.1	Qualidade de Geradores de Números Aleatórios e Pseudo-Aleatórios	19
4.2	Teste de Hipóteses	19
4.3	Descrição dos Testes	22
4.3.1	Teste da Frequência	23
4.3.2	Teste de Frequência Dentro de um Bloco	24

4.3.3	Teste das Corridas (<i>Runs Test</i>)	24
4.3.4	Teste das Corridas em Blocos	25
4.3.5	Teste Serial	27
4.3.6	Teste Pôquer	27
4.3.7	Teste da Autocorrelação	28
5	Atividades Realizadas	30
5.1	Análise	30
5.1.1	Processo de Desenvolvimento	30
5.1.2	Levantamento de Requisitos	32
5.2	Solução Proposta	34
5.2.1	Arquitetura	35
5.2.2	Tecnologias Utilizadas	37
5.2.3	Interface Gráfica	39
5.2.4	Documentação	43
5.3	Avaliação da Solução Proposta	43
5.3.1	Validação	43
5.3.2	Verificação	46
5.3.3	Trabalhos Relacionados	46
6	Considerações Finais	50
A	Conceitos Matemáticos Utilizados	55
A.1	Aritmética Modular	55
A.1.1	Congruência módulo m	55
A.1.2	Redução Módulo m	56
A.1.3	Classes Residuais	56
A.1.4	Inverso Multiplicativo	57
A.2	Reticulados	58
A.3	Variáveis Aleatórias	58
A.3.1	Valor esperado, Variância e Desvio Padrão	60
A.4	Distribuições de Probabilidade	61

A.4.1	Distribuição Uniforme	61
A.4.2	Distribuição de Bernoulli	62
A.4.3	Distribuição Normal	62
A.4.4	Distribuição Binomial	62
A.4.5	Distribuição χ^2	63
A.5	Funções Matemáticas Auxiliares	63
A.5.1	Função Complementar de Erro	64
A.5.2	Função Gama Complementar Incompleta	64
B	Plano de Estágio	65
C	Cronograma	66
D	Dados da Validação	68
D.1	Nível de Significância	68
D.2	Comparação com o NIST	73

Lista de Siglas e Abreviaturas

erfc Função Complementar de Erro

H_a Hipótese Alternativa

H_0 Hipótese Nula

igamac Função Gama Complementar Incompleta

IQuanta Instituto de Estudos em Computação e Informação Quânticas

JVM Máquina Virtual Java

MVC Padrão Arquitetural *Model-View-Controller*

NIST *National Institute of Standards and Technology*

PRNG Gerador de Números Pseudo-Aleatórios

RNG Gerador de Números Aleatórios

UFCC Universidade Federal de Campina Grande

XP *Extreme Programming*

Lista de Figuras

3.1	Reticulado em \mathbb{R}^2 obtido a partir do gerador congruente linear simples $x_i \equiv (7 \cdot x_{i-1}) \pmod{31}$ alimentado com semente igual a 27.	12
3.2	Perspectiva do Reticulado em \mathbb{R}^3 obtido a partir do gerador congruente linear simples $x_i \equiv (7 \cdot x_{i-1}) \pmod{31}$	12
3.3	Reticulado em 3 dimensões para uma seqüência de 10.000 números gerados com o RANDU [Entacher 2000]. Os números apresentados no reticulado estão normalizados em função do maior valor gerado.	13
3.4	Exemplo de uma entrada para o gerador de registradores de deslocamento com retro-alimentação. A entrada fornecida é o número 181 e o resultado produzido é o número 106, ambos na notação decimal.	17
5.1	Funcionalidades que devem ser providas a um usuário da aplicação.	34
5.2	Arquitetura do <i>Sieve</i>	36
5.3	Utilização dos padrões <i>Facade</i> , <i>Singleton</i> e <i>Factory Method</i> no <i>Sieve</i>	36
5.4	Tela inicial do <i>Sieve</i>	40
5.5	Tela de entrada do arquivo que contém a seqüência numérica.	40
5.6	Tela de seleção e configuração dos testes.	40
5.7	Tela de seleção do nível de significância.	41
5.8	Tela de acesso aos resultados dos testes.	41
5.9	Exemplo de relatório gerado pelo <i>Sieve</i>	42
5.10	Seleção de testes no DIEHARD.	47
5.11	Seleção de testes no DIEHARD.	48
A.1	Exemplo de um reticulado em \mathbb{R}^2 . Na área sombreada destaca-se o domínio fundamental deste reticulado.	59

Lista de Tabelas

4.1	Conclusões que podem ser tomadas a partir de um teste de hipóteses.	20
4.2	O tamanho da entrada n é utilizado para definir o tamanho M dos blocos. . .	26
4.3	Tabulação das contagens das maiores corridas de bits 1's em cada bloco. . .	26
4.4	Quando $m = 5$, é possível fazer uma interpretação dos dígitos dos números como se fossem cartas de baralho em um jogo de pôquer.	28
5.1	Médias e desvios-padrão da ocorrência de Erro Tipo I decorrente da execução dos testes do <i>Sieve</i> sob os Conjuntos 1, 2 e 3 com diferentes níveis de significância.	45
5.2	Tabela comparativa de softwares destinados a análise estatística de seqüências numéricas.	49
A.1	Distribuição de probabilidades da variável aleatória X	60
C.1	Cronograma das atividades realizadas durante o estágio	67
D.1	Resultado da execução do Teste da Frequência com diferentes níveis de significância.	68
D.2	Resultado da execução do Teste da Frequência Dentro de um Bloco com diferentes níveis de significância. Cada bloco possui 10 bits.	69
D.3	Resultado da execução do Teste da Frequência Dentro de um Bloco com diferentes níveis de significância. Cada bloco possui 50 bits.	69
D.4	Resultado da execução do Teste da Frequência Dentro de um Bloco com diferentes níveis de significância. Cada bloco possui 100 bits.	69
D.5	Resultado da execução do Teste das Corridas com diferentes níveis de significância.	70

D.6	Resultado da execução do Teste das Corridas em Blocos com diferentes níveis de significância. Cada bloco possui 128 bits.	70
D.7	Resultado da execução do Teste Serial com diferentes níveis de significância.	70
D.8	Resultado da execução do Teste Pôquer com diferentes níveis de significância. Cada número é composto por 5 bits (mão de pôquer).	71
D.9	Resultado da execução do Teste Pôquer com diferentes níveis de significância. Cada número é composto por 10 bits.	71
D.10	Resultado da execução do Teste da Autocorrelação com diferentes níveis de significância. O shift é de 1 bit.	71
D.11	Resultado da execução do Teste da Autocorrelação com diferentes níveis de significância. O shift é de 10 bits.	72
D.12	Resultado da execução do Teste da Autocorrelação com diferentes níveis de significância. O shift é de 25 bits.	72
D.13	Resultado da execução do Teste da Autocorrelação com diferentes níveis de significância. O shift é de 40 bits.	72
D.14	Resultado da execução dos testes para a expansão binária de π com nível de significância igual a 0.1	73
D.15	Resultado da execução dos testes para a expansão binária de e com nível de significância igual a 0.15	73
D.16	Resultado da execução dos testes para a expansão binária de $\sqrt{2}$ com nível de significância igual a 0.15	74
D.17	Resultado da execução dos testes para a expansão binária de $\sqrt{3}$ com nível de significância igual a 0.15	74

Capítulo 1

Introdução

Neste capítulo é apresentada a problemática que motivou a realização do estágio e os objetivos geral e específicos que o trabalho se propôs a atender.

1.1 Contextualização

Muitos métodos estatísticos utilizados pela Engenharia e Ciências Naturais demandam utilização de *números aleatórios* para simulação de processos físicos e biológicos. A utilização destes números também se aplica à outros domínios, por exemplo, a criptografia – para permitir a troca segura de dados, e aos sistemas operacionais – possibilitando a execução de determinados algoritmos.

Para obtenção de números aleatórios normalmente se utiliza uma fonte geradora de números aleatórios, que consiste em um *hardware* especializado para a captura de números aleatórios gerados em decorrência de algum fenômeno físico, por exemplo, o decaimento de uma substância química. Apesar dos números resultantes serem verdadeiramente aleatórios, o custo deste hardware e a imprevisibilidade dos fenômenos físicos costumam ter um impacto negativo na adoção destes geradores.

Em virtude dos fatores mencionados, a utilização de números aleatórios se torna inviável para algumas situações. Porém, algumas delas, sem prejuízo aos resultados, podem recorrer à utilização de *números pseudo-aleatórios*: seqüências de números obtidas a partir de funções matemáticas, executadas por computadores convencionais, que assemelham-se à seqüências de números aleatórios de uma dada distribuição de probabilidade [Gentle 2003].

A geração de números pseudo-aleatórios possui várias vantagens em relação à geração de números aleatórios, pois (i) costuma ser mais rápida; (ii) não depende de hardware específico, (iii) pode ser facilmente configurada; (iv) várias técnicas diferentes para geração deste tipo de número já foram desenvolvidas e (v) está presente na maioria das linguagens de programação e bibliotecas de sistemas operacionais.

Apesar do grande conjunto de vantagens práticas, quando certas seqüências de números fornecidas por geradores pseudo-aleatórios são analisadas por meio de testes estatísticos, tais seqüências podem fornecer indícios de que seus respectivos geradores podem ser inadequados para determinadas aplicações, por exemplo, na criptografia, em que a previsibilidade da seqüência pode representar uma quebra na segurança dos dados criptografados [Williams and Clearwater 1998].

Assim, a escolha do gerador adequado para uma determinada aplicação não depende somente dos atributos deste gerador (técnica para geração, intervalo dos números geradores, período, etc.), mas também da sua qualidade, que pode ser avaliada por meio de análises estatísticas das amostras geradas. É com este propósito, de avaliar a qualidade de geradores de números aleatórios e pseudo-aleatórios, que as atividades deste estágio foram realizadas.

1.2 Objetivos Geral

O objetivo geral deste estágio foi:

Construir uma aplicação para analisar a qualidade de geradores de números aleatórios e pseudo-aleatórios.

1.3 Objetivos Específicos

Os objetivos específicos deste estágio foram:

1. Desenvolver uma bateria de testes estatísticos para indicar não-aleatoriedade em seqüências numéricas obtidas por meio de geradores aleatórios e pseudo-aleatórios;
2. Produzir documentação e implementação em software destes testes; e

3. Prover orientações no uso e aplicação dos testes estatísticos oferecidos.

As atividades desenvolvidas e objetivos alcançados neste estágio estão inserido nos projetos de pesquisa do Prof. Dr. Francisco M. de Assis, o qual atuou como um dos orientadores do estágio.

Capítulo 2

Ambiente de Estágio

Neste capítulo é apresentado o ambiente no qual o estágio foi desenvolvido, os recursos de hardware disponíveis, assim como as informações de identificação sobre os orientadores acadêmicos e o supervisor técnico.

2.1 Ambiente de Estágio

O IQuanta – Instituto de Estudos em Computação e Informação Quânticas – é uma instituição de direito privado sem fins lucrativos, com direção definida em estatuto e constituída por docentes do quadro permanente da Universidade Federal de Campina Grande (UFCG) lotados nos departamentos de Física, de Sistemas e Computação e de Engenharia Elétrica. As instalações do instituto estão situadas no bloco CZ.A do Campus I da Universidade Federal de Campina Grande, localizada na Avenida Aprígio Veloso, 882, no bairro de Bodocongó, Campina Grande, Paraíba.

O instituto tem por objetivo promover o desenvolvimento de conhecimentos e tecnologias nas áreas da Computação e da Informação Quânticas, incentivando parcerias, a troca de informações e a discussão de temas relacionados ao crescimento brasileiro nessas áreas, em todos os seus aspectos: pesquisa, desenvolvimento, ensino e serviço.

O trabalho desenvolvido neste estágio está alinhado com os objetivos do IQuanta, no tocante ao desenvolvimento de conhecimentos e de tecnologias nas áreas de Computação e Informação Quânticas. Os resultados obtidos com o estágio irão colaborar para uma pesquisa sobre ataques quânticos à geradores aleatórios e pseudo-aleatórios que será iniciada

posteriormente, ao final do estágio.

As atividades desenvolvidas neste estágio totalizaram 300 horas, conforme proposto inicialmente no plano de estágio (ver Apêndice B). A maioria destas atividades foi desenvolvida no IQuanta, no horário previamente estabelecido. Outras atividades, porém, foram desenvolvidas no turno da noite, realocadas em virtude de mudanças decorrentes da proximidade do término do curso pela estagiária.

2.2 Recursos de Hardware e Software

O IQuanta conta com infra-estrutura própria – instalação física, máquinas e rede – para o desenvolvimento de seus projetos de pesquisa. Para o desenvolvimento deste estágio foram utilizadas duas máquinas :

1. Um notebook AMD Sempron 3500+, dispendo de 1 GB de memória RAM e 80 GB de disco rígido, no qual foi configurado o ambiente de desenvolvimento; e
2. Um computador desktop Core 2 Duo, dispendo de 2 GB de memória RAM e 160 GB de disco rígido. Neste computador foram executados os testes na aplicação.

Além destes recursos, a sala de reuniões do IQuanta foi utilizada para a realização dos encontros de acompanhamento das atividades do estágio.

Dentre os recursos de software utilizados, destacam-se a IDE Eclipse na qual foi efetuada a implementação da aplicação resultante deste estágio e o \LaTeX , ferramenta para edição de texto.

2.3 Orientação Acadêmica e Supervisão Técnica

Os orientadores acadêmicos atuaram como clientes, definindo os requisitos para a aplicação a ser desenvolvida, e também auxiliando no esclarecimento e diretrizes para a resolução das dúvidas teóricas que surgiram durante o estágio.

O supervisor técnico atuou auxiliando na definição do plano de trabalho, na validação das *user stories* e da implementação, no auxílio da escrita do relatório de estágio e na preparação para as reuniões com os orientadores.

2.3.1 Orientação Acadêmica

1. Professor Dr. Francisco Marcos de Assis

- Informações Gerais: Pós-Doutor em Engenharia Elétrica pela Universidade de Toronto, Canadá, atualmente é professor associado da Universidade Federal de Campina Grande e presidente do IQuanta.
- E-mail: fmarassis@gmail.com

2. Professor Dr. Bernardo Lula Jr

- Informações Gerais: Doutor em Informática pela Université d'Aix-Marseille, França, atualmente é professor associado da Universidade Federal de Campina Grande e membro do IQuanta.
- E-mail: bernardo.lulajr@gmail.com

2.3.2 Supervisão Técnica

1. Gilson Oliveira dos Santos

- Informações Gerais: Mestre em Informática pela Universidade Federal da Paraíba, atua como professor efetivo do Instituto Federal de Educação Tecnológica de Alagoas e atualmente está cursando Doutorado na Universidade Federal de Campina Grande na área de concentração de Processamento da Informação.
- E-mail: gilson10@click21.com.br

Capítulo 3

Geração de Números Aleatórios e Pseudo-Aleatórios

O estudo da definição e das técnicas para a geração de números aleatórios e pseudo-aleatórios constitui parte essencial dos conhecimentos necessários para a fundamentação teórica deste estágio. Com este intuito, o presente capítulo permitirá o conhecimento da definição de geradores aleatórios e pseudo-aleatórios, exemplos de geradores, limitações e também exemplos práticos. Aqueles que desejarem ter uma visão aprofundada deste assunto é sugerida a obra de Gentle [Gentle 2003], que apresenta detalhadamente outras técnicas de geração de seqüências numéricas, como combinar geradores e orientações para a escolha de sementes, módulos, etc.

3.1 Geradores de Números Aleatórios

Muitos métodos estatísticos utilizados pela Engenharia e Ciências Naturais demandam utilização de *números aleatórios* para simulação de processos físicos e biológicos. A utilização destes números também se aplica à outros domínios, por exemplo, na criptografia, para permitir a troca segura de dados, e nos sistemas operacionais, possibilitando a execução de determinados algoritmos.

Uma seqüência de números é dita ser *aleatória* se (i) os bits da representação binária dos números da seqüência assumirem os valores “falso” ou “verdadeiro” de modo *equiprovável* e se (ii) a obtenção de tais números for feita de forma independente [Rukhin et al. 2008].

Uma definição alternativa, enuncia que seqüências aleatórias finitas de números não podem ser descritas por alguma subsequência de si mesmas [L'Ecuyer 1990].

Em quase todos os sistemas para criptografia, por exemplo, é necessária a utilização de números aleatórios, seja para a criação de chaves para assinatura digital ou para a melhoria no processo de encriptação [Hoffstein et al. 2008]. Porém, diante das demandas pela troca segura de dados, os requisitos de aleatoriedade costumam ser mais rigorosos que em outras situações. Em face dos requisitos rigorosos em relação à aleatoriedade, uma característica importante de seqüências de números utilizadas é a *imprevisibilidade*, ou seja, dada uma subsequência, não é possível prever os elementos anteriores nem tampouco os próximos elementos da seqüência. Caso a seqüência fosse previsível, haveria comprometimento do processo, a exemplo de uma troca segura das informações [Gentle 2003].

Para obtenção de números aleatórios utilizam-se *geradores de números aleatórios* (RNG – *Random Numbers Generator*) que, para produzirem seqüências de números, utilizam uma fonte não-determinística (*fonte de entropia*) em conjunto com alguma função de processamento (*processo de destilação da entropia*), necessária para minimizar a geração de números não-aleatórios decorrentes da fonte de entropia.

Vários tipos diferentes de fontes de entropia são utilizadas em RNG's, a citar: decaimento de uma substância química [Gentle 2003], turbulência em discos rígidos [Davis et al. 1994], fenômenos quânticos [Williams and Clearwater 1998], dentre outros. Apesar de gerarem números aleatórios, na prática alguns RNG's demandam tempo para geração de seqüências e os resultados de algumas fontes de entropia podem ser previsíveis, o que torna inviável a utilização deste geradores sem outras técnicas ou tecnologias associadas [Rukhin et al. 2008].

3.2 Geradores de Números Pseudo-Aleatórios

Computadores digitais não são capazes de gerarem números aleatórios e em algumas situações não é conveniente conectar um *hardware* externo para obtenção de números aleatórios. Nestes casos, a utilização de seqüências numéricas advindas de *geradores de números pseudo-aleatórios* (PRNG – *Pseudo-Random Numbers Generator*) são uma alternativa viável [Gentle 2003].

A geração de números pseudo-aleatórios possui várias vantagens em relação à geração

de números aleatórios, pois (i) costuma ser mais rápida; (ii) não depende de hardware específico, (iii) pode ser facilmente configurada; (iv) várias técnicas diferentes para geração deste tipo de números já foram desenvolvidas e (v) está presente na maioria das linguagens de programação e bibliotecas de sistemas operacionais. Além destas vantagens, estudos afirmam que números pseudo-aleatórios normalmente apresentam-se “mais aleatórios” do que números obtidos de RNG’s [Rukhin et al. 2008].

Os tipos mais utilizados de geradores pseudo-aleatórios produzem seqüências de números que assemelham-se a seqüências aleatórias, mas que, na verdade, são simplesmente resultados de computações sob os números previamente gerados. Uma seqüência numérica pseudo-aleatória pode ser comparada a uma seqüência aleatória por meio de uma *distância probabilística*, cuja definição é baseada em um teste estatístico sobre qualquer algoritmo de previsão do próximo número da seqüência [Trevisan 1998]. A distância probabilística é dada por:

$$\max \{|P(A) - Q(A)| : A \subset \{0, 1\}\}$$

cujo resultado representa a maior possível diferença entre as probabilidades de duas distribuições para o mesmo evento.

Matematicamente, um PRNG pode ser representado por uma função recursiva f que toma os k números gerados anteriormente para determinação de um novo número para a seqüência:

$$x_i = f(x_{i-1}, \dots, x_{i-k})$$

em que x_{i-1}, \dots, x_{i-k} são números inteiros.

A quantidade k de números anteriores utilizados denomina-se a *ordem* do gerador e o conjunto de valores que é fornecido inicialmente denomina-se *semente*.

Uma vez que a quantidade de números fornecida é limitada, a seqüência resultante irá repetir. O comprimento da seqüência de números gerada até a primeira repetição denomina-se *período* do gerador.

Como dito anteriormente, existem várias técnicas consagradas para geração de números pseudo-aleatórios. Algumas destas técnicas mais comumente utilizadas e suas características serão apresentadas nas seções a seguir.

3.2.1 Geradores Congruentes Lineares Simples

Os geradores congruentes lineares simples foram propostos em 1951 por Lehmer como uma fonte de números pseudo-aleatórios [Lehmer 1951]. Neste tipo de gerador, um único número determina seu sucessor, através de uma função seguida por uma redução modular¹. Assim, a forma geral destes geradores é:

$$x_i \equiv (a \cdot x_{i-1} + c) \pmod{m}$$

em que $0 \leq x_i < m$. O escalar a é denominado *multiplicador* e c é denominado *incremento*. Nos casos em que $c = 0$, estes geradores são denominados *geradores congruentes multiplicativos*.

A semente para geradores congruentes lineares é somente um valor x_0 para iniciar a recursão e os valores resultantes denominam-se *seqüência de Lehmer*. A escolha da semente e dos escalares a e m são determinantes para definirem as características da seqüência, a exemplo do período.

Por exemplo, um gerador congruente linear simples em que os valores de $m = 31$ e $a = 7$ é da forma:

$$x_i \equiv (7 \cdot x_{i-1}) \pmod{31}$$

Se o valor $x_0 = 19$ é fornecido como semente, obtém-se o seguinte valor:

$$\begin{aligned} x_i &\equiv (7 \cdot 19) \pmod{31} \\ &\equiv 133 \pmod{31} \\ &\equiv 9. \end{aligned}$$

A seqüência é então construída ao se fornecer o número 9 como entrada e assim sucessivamente. A seqüência de Lehmer deste gerador para a semente 19 é:

$$9, 1, 7, 18, 2, 14, 5, 4, 28, 10, 8, 25, 20, 26, 19, \dots$$

¹Conceitos da Aritmética Modular são apresentados na Seção A.1.

É evidente que a seqüência será repetida, pois o último número foi 19, que havia sido fornecido inicialmente como semente. Neste exemplo, o período é igual a 15.

Uma vez que o valor de x_i é determinado por x_{i-1} e só existem m possíveis valores de x , o período máximo de um gerador congruente linear simples é m . Porém, dado que $x_{i-1} = 0$ não se adequa a este tipos de geradores, o maior período possível é reduzido a $m - 1$ números. Nos casos práticos de utilização de geradores congruentes lineares, o período é aproximadamente da ordem de 10^9 , o que significa que o módulo do gerador deve ser um inteiro grande, normalmente entre 10^9 e 10^{15} . Ainda assim, considera-se que para aplicações de simulação períodos desta ordem são relativamente curtos [Gentle 2003].

Além das limitações práticas relativas ao comprimento do período dos geradores congruentes lineares, existem outros aspectos que devem ser levados em consideração na adoção destes geradores. Um destes aspectos diz respeito a estrutura resultante das seqüências numéricas produzidas por geradores congruentes lineares, que se mostra bastante regular.

Marsaglia [Marsaglia 1968] mostrou que a saída de qualquer gerador congruente linear encontra-se disposta em um reticulad² de dimensão k , em que os eixos representam as saídas do gerador. Por exemplo, a seqüência de Lehmer do gerador $x_i \equiv (7 \cdot x_{i-1}) \pmod{31}$ com semente igual a 27 é:

27, 19, 26, 16, 17, 20, 29, 25, 13, 8, 24, 10, 30, 28, 22, 4, 12, 5, 15, 14, 11, 2, 6, 18, 23, 7, 21, 1, 3, 9.

O reticulado em \mathbb{R}^2 para este gerador é dado por $\{a \cdot \vec{v}\}$ em que $\vec{v} = (1, 3)$. Este reticulado encontra-se ilustrado na Figura 3.1.

A existência de um padrão é bastante evidente após a construção do reticulado. Se este padrão não for evidenciado em uma determinada dimensão, pode ser detectado em uma dimensão maior. Por exemplo, para o mesmo gerador sob as mesmas condições de inicialização, o reticulado em \mathbb{R}^3 encontra-se apresentado na Figura 3.2. Neste reticulado também é possível verificar a existência de um padrão de repetição.

Uma inspeção visual é suficiente para constatar a existência de um padrão no reticulado. A partir do conhecimento do padrão de repetição é possível estabelecer quais serão os próximos números a partir de sub-amostras da seqüência, por exemplo, tomando como referência

²Conceitos introdutórios sobre Reticulados disponíveis na Seção A.2.

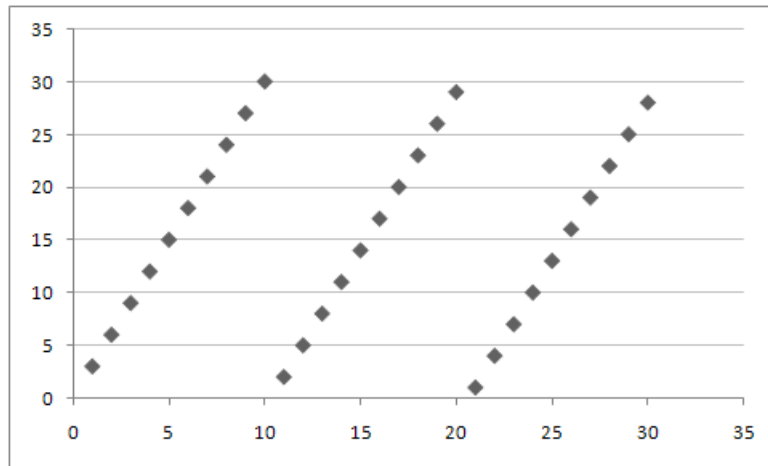


Figura 3.1: Reticulado em \mathbb{R}^2 obtido a partir do gerador congruente linear simples $x_i \equiv (7 \cdot x_{i-1}) \pmod{31}$ alimentado com semente igual a 27.

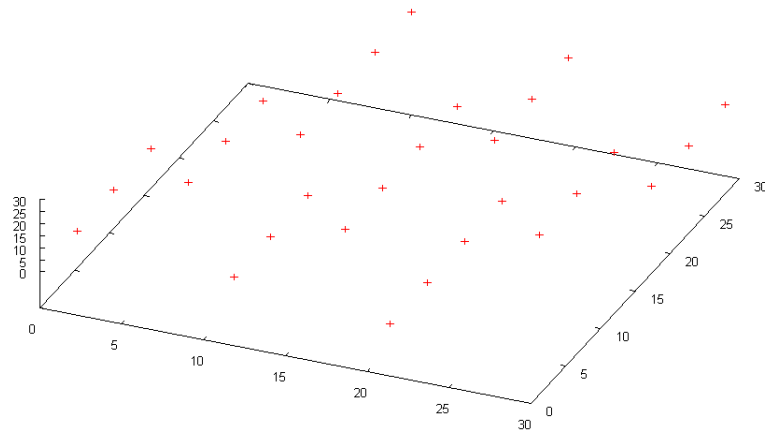


Figura 3.2: Perspectiva do Reticulado em \mathbb{R}^3 obtido a partir do gerador congruente linear simples $x_i \equiv (7 \cdot x_{i-1}) \pmod{31}$.

a distância entre dois pontos no reticulado.

Um exemplo que ilustra a utilização de geradores congruentes lineares simples é o RANDU [IBM 1968]. Este PRNG foi desenvolvido pela IBM e atualmente é parte de algumas bibliotecas científicas de sistemas operacionais [GNU 2008]. O RANDU chegou a ser um dos geradores pseudo-aleatórios mais utilizados em todo o mundo. Este gerador é da forma:

$$x_i \equiv 65539 \cdot x_{i-1} \pmod{2^{31}}$$

O RANDU apresenta boas características quando analisado de acordo com um reticulado de duas dimensões – não há um padrão óbvio de repetição. Mas, quando o reticulado de 3 dimensões é construído, observa-se que os números resultantes agrupam-se em quinze planos [Marsaglia 1968], como ilustrado na Figura 3.3.

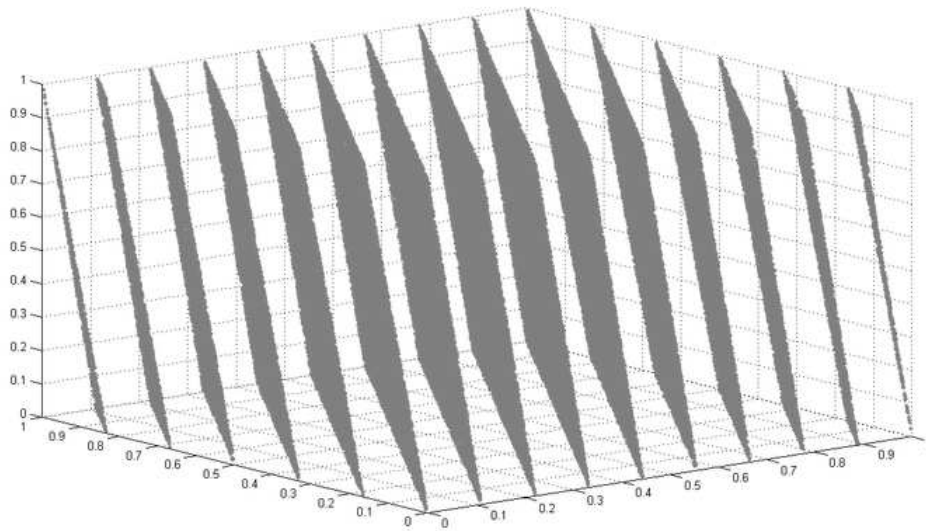


Figura 3.3: Reticulado em 3 dimensões para uma seqüência de 10.000 números gerados com o RANDU [Entacher 2000]. Os números apresentados no reticulado estão normalizados em função do maior valor gerado.

Pesquisas e experimentos mostram que atualmente o RANDU é considerado um gerador ultrapassado e que sua adoção deve ser evitada, por apresentar características não-aleatórias [Park and Miller 1988].

3.2.2 Geradores Congruentes Não-Lineares

Os geradores congruentes lineares são descritos por uma função do tipo $x_i = f(x_{i-1}, \dots, x_{i-k})$ que é linear. Para este tipo de gerador, a computação é feita mais rapidamente e considera-se que a saída é mais tratável. Porém, como apresentado anteriormente, a saída apresenta padrões que podem comprometer a utilização deste gerador em algumas aplicações. Em virtude disto, vários *geradores congruentes não-lineares* foram propostos.

Os geradores congruentes não-lineares baseiam-se no uso de funções não-lineares. Se

uma função g é não-linear, então ela não satisfaz os requisitos a seguir:

1. $g(u + v) \neq g(u) + g(v)$;
2. $g(\lambda \cdot u) \neq \lambda \cdot g(u)$.

Existem várias definições de geradores congruentes não-lineares possíveis. Os *geradores congruentes inversivos*, por exemplo, foram propostos em 1986 e baseiam-se na utilização de inversos multiplicativos para gerar o próximo elemento da seqüência [Eichenauer and Lehn 1986]. Um gerador congruente inversivo é da forma:

$$x_i \equiv (a \cdot x_{i-1}^- + c) \pmod{m}$$

em que $0 \leq x_i < m$ e x^- denota o inverso multiplicativo de x módulo m . Nos casos em que x^- não é definido, x^- denota 0 (zero).

Por exemplo, seja um gerador congruente inversivo da forma $x_i \equiv (5 \cdot x_{i-1}^-) \pmod{11}$. Caso a semente fornecida seja 2, o próximo elemento da seqüência será:

$$\begin{aligned} x_i &\equiv (5 \cdot x_{i-1}^-) \pmod{11} \\ &\equiv (5 \cdot 28) \pmod{11} \\ &\equiv 140 \pmod{11} \\ &\equiv 8. \end{aligned}$$

As seqüências produzidas por geradores congruentes inversivos possuem boas características estatísticas, inclusive em termos de não existência de um padrão óbvio em reticulados. Porém, as dificuldades computacionais para o cálculo do inverso multiplicativo, especialmente para os casos de números grandes, foi comprovada como um fator limitante da utilização destes geradores [Gentle 2003].

Outra definição de geradores congruentes não-lineares foi proposta por Knuth [Knuth 1998] e possui a seguinte forma:

$$x_i \equiv (d \cdot x_{i-1}^2 + a \cdot x_i - 1 + c) \pmod{m}$$

em que $0 \leq x_i < m$. Com este gerador, a idéia de Knuth era utilizar elementos de geradores consagrados e incrementá-los, na tentativa de gerar seqüências numéricas “ainda mais” aleatórias. Embora seja possível utilizar polinômios de grau superior a 2, não foi comprovado que isto aumente a aleatoriedade da seqüência gerada.

O gerador de Blum-Blum-Schub [Blum et al. 1986] é um exemplo de gerador congruente não-linear de ampla utilização. Para a inicialização deste gerador, são fornecidos uma semente x_0 e os valores p_1 e p_2 , os quais devem ser primos e congruentes a 3 mod 4. A forma geral deste gerador é dada por:

$$x_i \equiv x_{i-1}^2 \pmod{m}$$

em que $m = p_1 \cdot p_2$. A saída deste gerador forma uma seqüência de bits b_1, b_2, \dots em que $b_i = 0$ se x_i é ímpar, e $b_i = 1$ em caso contrário. A saída destes geradores é tal que não pode ser previsível em tempo polinomial sem conhecimento de p_1 e p_2 .

Por exemplo, suponha que a saída deste gerador foi 22 e deseja-se saber qual seu predecessor na seqüência. Sabe-se que:

$$22 \equiv x_{i-1}^2 \pmod{m}$$

Sem o conhecimento prévio de m , não é possível estabelecer o predecessor sem que sejam testadas todas as possibilidades. Porém, ao saber que $m = 39$, é fácil descobrir, com a ajuda de conhecimentos da Aritmética Modular, que o predecessor é 10. Em razão desta dificuldade é que se considera este gerador adequado para criptografia.

É interessante observar que este gerador atende às restrições para um gerador congruente não-linear. Basta denotá-lo como uma função g em que, fornecido um parâmetro, o valor é o próximo número da seqüência. Considerando $m = 39$, tal como determinado anteriormente, tem-se:

1. Seja $x_0 = 3 + 9 = 12$. $g(12) = 27$, mas $g(3) + g(9) = 9 + 3 = 12$. Isto significa que a primeira condição não é satisfeita, ou seja, $g(12) \neq g(3) + g(9)$;
2. Seja $x_0 = 2 \cdot 7 = 14$. $g(14) \neq 2 \cdot g(7) \Rightarrow 1 \neq 20$. Neste caso a segunda condição também não é satisfeita, pois $g(14) \neq 2 \cdot g(7)$.

Existe uma vasta literatura sobre geradores congruentes não-lineares: outras definições, análises estatísticas, aplicações práticas, etc. Independentemente do gerador congruente não-linear utilizado, considera-se que a escolha dos parâmetros de inicialização são essenciais para a geração de seqüências numéricas com boas características. Além disso, quando há demanda por muitos números, o ideal é conhecer bem a estrutura do gerador e adquirir hardware específico para acelerar as computações não-triviais destes geradores [Gentle 2003].

3.2.3 Geradores com Registradores de Deslocamento Retro-Alimentados

Os geradores com registradores de deslocamento retro-alimentados (do inglês, *feedback shift registers generators*) foram propostos por Tausworthe [Tausworthe 1965] e, diferentemente dos geradores previamente apresentados, baseiam-se na manipulação individual de bits.

Nestes geradores, os números produzidos são funções lineares de seus antecessores, por meio das operações *ou-exclusivo* (XOR) ou *coincidência* (XNOR) realizadas com os bits que compõem a sua representação binária. As operações ou-exclusivo e coincidência são permitidas por serem as únicas operações bit-a-bit lineares.

A definição da obtenção de números neste tipo de gerador é dada por:

$$b_i \equiv (a_p \cdot b_{i-p} + a_{p-1} \cdot b_{i-p+1} + \dots + a_1 \cdot b_{i-1}) \pmod{2}$$

em que o resultado é um valor binário. Uma possível interpretação esta definição é: os números, representados em binário, são vistos como um vetor de bits, ou seja, um registrador, que pode ser deslocado para a direita ou para a esquerda e re-introduzidos na seqüência resultante por meio das operações ou-exclusivo ou coincidência. Tal conjunto de operações justifica a nomenclatura destes geradores.

Para exemplificar, seja um gerador de registradores de deslocamento com retro-alimentação cuja representação de números é feita por um byte. Neste gerador, o próximo número da seqüência é produzido pelos seguintes passos:

1. Tome o último número gerado;

2. Desloque-o de 1 bit para a esquerda;
3. Tome o bit que foi eliminado, efetue uma soma módulo 2 (\oplus) com o quarto bit menos significativo;
4. O resultado da operação anterior será o bit menos significativo do novo número produzido.

Para ilustrar o funcionamento deste gerador, a obtenção de um número a partir de uma semente é mostrada na Figura 3.4. Neste exemplo o conjunto de passos é bastante simplificado, apenas para ilustração, mas em outras definições de geradores a direção, o número de deslocamentos e os passos para a combinação podem ser mais complexos.

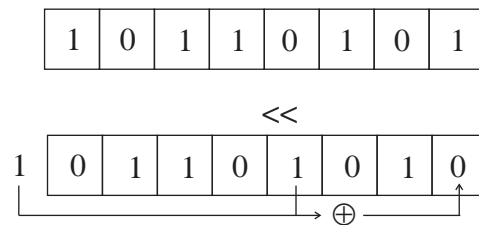


Figura 3.4: Exemplo de uma entrada para o gerador de registradores de deslocamento com retro-alimentação. A entrada fornecida é o número 181 e o resultado produzido é o número 106, ambos na notação decimal.

Estes geradores são bastante utilizados para fins criptográficos, em particular pela facilidade de serem implementados por meio de circuitos. Um exemplo prático é a utilização das seqüências numéricas produzidas para composição de cifras em comunicações da rede móvel GSM e nas troca de dados via Bluetooth. Além do interesse prático, estes geradores possuem uma teoria bastante desenvolvida e estudos empíricos verificaram que as seqüências produzidas por estes geradores possuem boas características, reforçando a viabilidade de utilização [Gentle 2003].

Capítulo 4

Análise Estatística de Geradores

Aleatórios e Pseudo-Aleatórios

O Capítulo 3 apresentou diferentes técnicas para geração de seqüências numéricas por meio de geradores aleatórios e pseudo-aleatórios. Apesar da grande variedade de técnicas, uma questionamento relevante é saber se um dado gerador é adequado para uma determinada aplicação. Uma maneira de esclarecer esta questão é por meio da utilização de *testes estatísticos*.

Testes estatísticos fazem uso de uma amostra de números obtidas de um gerador e, a partir delas, são capazes de determinar quando esta seqüência possui determinados atributos que uma seqüência (verdadeiramente) aleatória viria a apresentar. Um exemplo de atributo observado é a mesma proporção de 0's e 1's na seqüência. Em virtude dos diferentes atributos a serem observados, existem diferentes definições de testes possíveis na literatura [Knuth 1998, Rukhin et al. 2008, L'Ecuyer 1992].

O objetivo deste capítulo é apresentar definições de testes estatísticos relevantes especialmente para fins de simulação e orientar um usuário de geradores de números aleatórios e pseudo-aleatórios a verificar se um gerador é adequado para um determinado fim. Para a melhor compreensão dos conceitos a serem apresentados, recomenda-se a leitura prévia das Seções A.4 e A.5 do Apêndice A, as quais apresentam, respectivamente, as distribuições de probabilidade e as funções auxiliares adotadas.

4.1 Qualidade de Geradores de Números Aleatórios e Pseudo-Aleatórios

Existem muitos atributos que são levados em consideração na utilização de números advindos de geradores aleatórios e pseudo-aleatórios. Alguns destes atributos são a frequência dos bits 0's e 1's, a ausência de padrões sistemáticos, a correlação nula entre as amostras produzidas, etc. Uma vez que todos os atributos analisados são relativos às seqüências produzidas pelos geradores, é necessária uma metodologia empírica de verificação de tais atributos nas seqüências produzidas por um gerador.

A verificação em questão é efetuada por meio de *testes estatísticos*, que determinam se uma seqüência fornecida possui um determinado atributo que um gerador (verdadeiramente) aleatório poderia apresentar. Estes testes geram conclusões a partir da seqüência fornecida como entrada, ou seja, não constituem uma prova formal que ateste aleatoriedade na geração da seqüência nem tampouco resultam em uma conclusão definitiva, mas sim *probabilística* [Menezes et al. 1996].

No escopo deste trabalho, considera-se que um gerador aleatório ou pseudo-aleatório é dito ser de *qualidade* quanto maior for a quantidade de saídas bem sucedidas em testes estatísticos. Além disto, considera-se que somente aquelas saídas bem sucedidas devem sinalizar os geradores a serem utilizados [Gentle 2003, L'Ecuyer 1992].

4.2 Teste de Hipóteses

Para gerar conclusões a respeito dos atributos da seqüência é necessário efetuar um procedimento denominado *teste de hipóteses*, o qual consiste em uma regra de decisão utilizada para aceitar ou rejeitar uma hipótese estatística com base em elementos amostrais.

Um teste de hipóteses é composto pelos seguintes passos:

1. Enunciar as hipóteses nula e alternativa;
2. Fixar o nível de significância;
3. Definir a estatística do teste;

4. Definir a região crítica;
5. Definir a regra de decisão;
6. Efetuar a conclusão.

Um teste de hipóteses é composto por uma *hipótese nula* e uma *hipótese alternativa*, que corresponde à rejeição da hipótese nula. Para os geradores de números aleatórios e pseudo-aleatórios, a hipótese nula (H_0) que está sendo testada é:

H_0 : “Os números da seqüência fornecida encontram-se independente e identicamente distribuídos de acordo com uma distribuição uniforme discreta no intervalo $[0, 1]$ ”

Uma outra hipótese nula que costuma ser usada em testes estatísticos é se os bits que compõem a seqüência são identicamente e independentemente distribuídos como variáveis de Bernoulli com parâmetro igual a $\frac{1}{2}$. Associada à hipótese nula definida está a hipótese alternativa (H_a), que corresponde à negação da hipótese nula, ou seja, a seqüência fornecida não está distribuída conforme especificado.

O teste de hipóteses possui dois possíveis resultados: aceitar H_0 ou aceitar H_a . Para cada teste estatístico a ser realizado, a decisão ou conclusão de aceitar ou rejeitar a hipótese nula baseia-se apenas na seqüência fornecida, ou seja, não é verificada a corretude de um algoritmo de geração de números pseudo-aleatórios ou os atributos do processo físico utilizado para gerar números aleatórios. A Tabela 4.1 mostra as possíveis conclusões a que se pode chegar em um teste de hipóteses.

Tabela 4.1: Conclusões que podem ser tomadas a partir de um teste de hipóteses.

Real Situação	Conclusão	
	Aceitar H_0	Aceitar H_a (rejeitar H_0)
Os dados são aleatórios (H_0 é verdadeira)	Decisão Correta	Erro Tipo I
Os dados não são aleatórios (H_a é verdadeira)	Erro Tipo II	Decisão Correta

Se os dados estão distribuídos conforme especificado, então a conclusão errônea de rejeitar a hipótese nula pode vir a acontecer em uma pequena porcentagem de situações. Esta

conclusão incorreta denomina-se *Erro Tipo I* e a sua probabilidade de ocorrência denomina-se *nível de significância* do teste. Esta probabilidade, denotada por α , pode ser definida *a priori* e representa a probabilidade de classificar uma seqüência como não pertencente à distribuição especificada, quando, na verdade, ela pertence. Isto ocorre nas situações em que a seqüência fornecida apresenta poucas características aleatórias, ainda que tenha sido produzida por um gerador (verdadeiramente) aleatório.

A probabilidade de indicar que uma seqüência não possui distribuição uniforme (ou de Bernoulli) quando, de fato, ela possui, denomina-se *Erro Tipo II*, denotada por β . Ao contrário de α , β não possui um valor fixo porque depende das infinitas maneiras que uma seqüência pode ser pertencer à distribuição esperada e cada uma destas maneiras conduz a um diferente valor de β . A probabilidade de ocorrência do Erro Tipo II é intitulada *poder do teste*.

Para testar as hipóteses propostas, é necessário ir além da mera descrição das mesmas. É também necessário fazer inferências, ou seja, tomar decisões com base em dados colhidos. Para gerar conclusões a respeito de qual hipótese deve ser aceita ou rejeitada são utilizados testes *paramétricos* ou *não-paramétricos*, que resultam em uma *estatística*. Os testes paramétricos são utilizados em duas situações: (i) quando os dados analisados estão distribuídos de forma normal e (ii) quando se conhece o nível intervalar de mensuração. Porém, quando estes requisitos são violados, torna-se impossível conhecer o poder deste teste e o resultado obtido não tem interpretação significativa. Sob tais condições, opta-se pela utilização de testes não-paramétricos [Gonçalves and Lopes 2003, Levin 1987]. No domínio dos geradores aleatórios e pseudo-aleatórios, uma vez que não se tem informações sobre a distribuição da seqüência fornecida para análise, utilizam-se *testes não-paramétricos*.

De acordo com a hipótese nula proposta, deseja-se verificar se as seqüências numéricas fornecidas como entrada provém de um gerador (verdadeiramente) aleatório, ou seja, se são independentes e identicamente distribuídas de acordo com uma distribuição uniforme discreta no intervalo $[0, 1]$. Ao tipo de teste que têm o intuito de verificar se a população de onde foram retiradas as amostras possui determinada distribuição teórica (uniforme, normal, exponencial, etc.), denomina-se *teste de ajustamento*¹ [Gonçalves and Lopes 2003].

Para os testes estatísticos em seqüências advindas de geradores de números aleatórios

¹Do inglês, *goodness-of-fit test*.

e pseudo-aleatórios será aplicado, exceto quando houver ressalva, o teste de ajustamento não-paramétrico χ^2 (chi-quadrado). Este teste ocupa-se essencialmente com a distinção entre frequências esperadas e frequências obtidas (observadas). As frequências esperadas referem-se aos termos da hipótese nula, de acordo com os quais se espera que a frequência relativa (ou proporção) seja a mesma para todos os grupos. Por exemplo, no Teste Serial, que será apresentado na Seção 4.3.5, busca-se verificar, com a utilização do teste χ^2 , se o número de subsequências 00, 01, 10 e 11 é aproximadamente igual, tal como esperado em uma sequência que obedece à distribuição uniforme (de acordo com a hipótese nula proposta previamente).

Como resultado de cada teste, será obtida uma estatística denominada *valor p* (ou *p*-valor), que corresponde ao menor nível de significância que pode ser assumido para rejeitar a hipótese nula. Diz-se que há *significância estatística* quando *p* é menor que o nível de significância adotado. Por exemplo, quando $p = 0,0001$ pode-se afirmar que o resultado é bastante significativo, pois este valor é muito inferior aos níveis de significância usuais. Por outro lado, se $p = 0,048$ pode haver dúvida pois, embora o valor seja inferior, ele está muito próximo ao nível usual de 5%. Usualmente, quanto mais próximo de 0 está *p*, maior a indicação de que a hipótese nula deve ser rejeitada. Quanto mais próximo de 1 está o valor *p*, maior é a evidência de que não se deve rejeitar a hipótese nula. Assim, a conclusão de cada teste baseia-se na comparação $p < \alpha$. Caso seja verdadeira, rejeita-se a hipótese nula (equivalentemente, aceita-se H_a). Em caso contrário, H_0 é aceita.

Uma vez que todos os elementos que compõem os testes estatísticos a serem efetuados estão caracterizados, é necessário descrever quais testes serão realizados e quais as condições para execução dos mesmos. Tais informações serão tratadas na seção a seguir.

4.3 Descrição dos Testes

Nas subseções a seguir serão apresentados testes estatísticos que podem ser efetuados em sequências numéricas advindas de geradores aleatórios e pseudo-aleatórios. Para cada teste, será apresentado o seu propósito e quais passos são necessários para sua execução, bem como uma análise de como interpretar seus resultados.

A definição de tais testes leva em consideração *escalabilidade* e *consistência*. Escalabili-

dade significa que qualquer teste aplicado a uma seqüência pode ser aplicado a subsequências selecionadas aleatoriamente, que devem passar nos testes em que a seqüência passa; consistência significa que o comportamento de um gerador deve ser consistente em relação aos valores iniciais (sementes). De acordo com a consistência, por exemplo, seria inadequado testar um PRNG levando em consideração apenas seqüências obtidas a partir de uma mesma semente [Rukhin et al. 2008].

A escolha e caracterização dos testes a serem apresentados seguem sugestões de Menezes et al. [Menezes et al. 1996], Gentle [Gentle 2003], Rukhin et al. [Rukhin et al. 2008], L'Ecuyer [L'Ecuyer 1992] e Knuth [Knuth 1998].

4.3.1 Teste da Freqüência

O foco deste teste é avaliar a proporção de 0's e 1's em uma seqüência e o propósito é determinar quando este número é aproximadamente unitário, tal como esperado em uma seqüência aleatória.

Os n bits da seqüência são visto como realizações de *variáveis de Bernoulli* independentes e identicamente distribuídas, com parâmetro igual a $\frac{1}{2}$. Isto significa que será utilizada a hipótese nula de que os dados estão independentemente distribuídos como uma variáveis de Bernoulli com parâmetro igual a $\frac{1}{2}$.

Para um número de bits suficientemente grande, a distribuição tende a uma binomial normalizada por \sqrt{n} , ou seja, uma aproximação da distribuição normal padrão, tal como estabelece o Teorema do Limite Central [Rukhin et al. 2008]. Neste caso, em particular, é utilizado o teste normal padrão, que é adequado quando variáveis aleatórias independentes que possuem a mesma média e variância são somadas, tal como acontece no Teste da Freqüência.

O primeiro passo para a execução deste teste é representar os bits 0 como -1 e os bits 1 como $+1$ e efetuar uma soma cumulativa destes valores, denominada S_n . O módulo do valor de S_n deve ser dividido pela raiz quadrada da quantidade de bits somados, ou seja:

$$S_{obs} = \left(\frac{|S_n|}{\sqrt{n}} \right)$$

A partir deste resultado, computa-se o valor p para o teste:

$$p = \text{erfc} \left(\frac{S_{obs}}{\sqrt{2}} \right)$$

em que erfc é a função complementar de erro.

A conclusão deste teste baseia-se na comparação $p < \alpha$. Caso seja verdadeira, rejeita-se a hipótese nula (equivalentemente, aceita-se H_a). Em caso contrário, H_0 é aceita.

4.3.2 Teste de Frequência Dentro de um Bloco

Este teste é análogo ao teste da frequência, porém baseia-se na escalabilidade: se a hipótese nula é aceita no teste da frequência, provavelmente será aceita neste teste. Embora com definições similares, este teste diferencia-se do teste da frequência por utilizar a distribuição χ^2 como referência.

No teste da frequência dentro de um bloco, a sequência de bits ε fornecida como entrada deve ser particionada em N blocos de tamanho M , em que $N = \lfloor \frac{n}{M} \rfloor$. Após esta etapa, determina-se a proporção de bits 1's em cada bloco, denotada por π_i por meio da equação:

$$\pi_i = \frac{\sum_{j=1}^M \varepsilon_{(i-1)M+j}}{M}$$

para $1 \leq i \leq N$. Em seguida, computa-se a estatística do teste:

$$\chi_{(obs)}^2 = 4 \cdot M \cdot \sum_{i=1}^N \left(\pi_i - \frac{1}{2} \right)^2$$

e a partir deste resultado obtém-se o valor p do teste :

$$p = \text{igamac} \left(\frac{N}{2}, \frac{\chi_{(obs)}^2}{2} \right)$$

em que igamac denota a função gama incompleta.

Caso $p < \alpha$ seja verdadeiro, rejeita-se a hipótese nula (equivalentemente, aceita-se H_a). Em caso contrário, H_0 é aceita.

4.3.3 Teste das Corridas (*Runs Test*)

É um teste bastante utilizado cujo objetivo é a obtenção do número total de bits de um mesmo valor de forma contínua em uma sequência. Uma sequência contínua de comprimento k indica a ocorrência de k bits idênticos delimitados por um bit de valor oposto.

O objetivo deste teste é determinar quando o número de seqüências contínuas apresenta-se como esperado para uma seqüência advinda de um gerador (verdadeiramente) aleatório. Na prática, este teste detecta quando a oscilação entre 0's e 1's é muito rápida ou muito lenta.

Uma seqüência ε é fornecida e a partir dela calcula-se a proporção π de 1's na seqüência de entrada:

$$\pi = \frac{\sum_j \varepsilon_j}{n}$$

isto é equivalente a verificar se a seqüência passou no teste da freqüência (ver Seção 4.3.1).

Em caso afirmativo, verifica-se a seguinte desigualdade:

$$\left| \pi - \frac{1}{2} \right| \geq \tau$$

em que $\tau = \frac{2}{\sqrt{n}}$. Caso a desigualdade seja falsa, atribui-se $p = 0$. Nos demais casos, computa-se a estatística $V_n(obs)$, dada por:

$$V_n(obs) = \sum_{k=1}^{n-1} r(k) + 1$$

em que $r(k) = 0$ se $\varepsilon_k = \varepsilon_{k+1}$ e $r(k) = 1$, em caso contrário.

Um valor grande de $V_n(obs)$ indica que a seqüência de entrada oscila muito rapidamente entre 0's e 1's e um valor muito pequeno indica que esta oscilação ocorre lentamente. Com este resultado, computa-se o valor p por meio do teste normal-padrão:

$$p = \text{erfc} \left(\frac{|V_n(obs) - 2 \cdot n \cdot \pi(1 - \pi)|}{2 \cdot \sqrt{2 \cdot n \cdot \pi \cdot (1 - \pi)}} \right)$$

Caso $p < \alpha$ seja verdadeiro, rejeita-se a hipótese nula (equivalentemente, aceita-se H_a). Em caso contrário, H_0 é aceita.

4.3.4 Teste das Corridas em Blocos

Este teste é análogo ao teste das corridas, apresentado na Seção 4.3.3 e a descrição a ser apresentada segue a implementação sugerida por Rukhin et. al [Rukhin et al. 2008]. No teste das corridas em blocos, a entrada é dividida em blocos de tamanho M nos quais é averiguada a quantidade de corridas de bits 1. Neste teste, utiliza-se a distribuição \mathcal{X}^2 como referência.

O tamanho M dos blocos é função da quantidade n de bits fornecida como entrada, de acordo com a Tabela 4.2.

Tabela 4.2: O tamanho da entrada n é utilizado para definir o tamanho M dos blocos.

Valor de n	Valor de M
Até 128	8
Até 6272	128
Acima de 6272	10^4

Após a divisão da entrada em blocos, constrói-se uma tabela de frequências v_i com os maiores comprimentos de 1's dentro de cada bloco organizados em categorias, em que cada célula contém o número de seqüências contínuas de 1's de um dado comprimento. As categorias são determinadas de acordo com a Tabela 4.3.

Tabela 4.3: Tabulação das contagens das maiores corridas de bits 1's em cada bloco.

v_i	$M = 8$	$M = 128$	$M = 10^4$
v_0	≤ 1	≤ 4	≤ 10
v_1	2	5	11
v_2	3	6	12
v_3	≥ 4	7	13
v_4		8	14
v_5		≥ 9	15
v_6			≥ 16

Computa-se então o $\chi^2(obs) = \sum_{i=0}^K \frac{(v_i - N \cdot \pi_j)^2}{N \cdot \pi_j}$, utilizando os valores de K e π_j como sugeridos por Revesz [Revesz 1990]. A partir deste resultado calcula-se o valor p para o teste:

$$p = igamac\left(\frac{K}{2}, \frac{\chi^2(obs)}{2}\right)$$

Caso $p < \alpha$ seja verdadeiro, rejeita-se a hipótese nula (equivalentemente, aceita-se H_a). Em caso contrário, H_0 é aceita.

4.3.5 Teste Serial

O teste serial analisa dois bits por vez e o propósito deste teste é determinar quando as ocorrências de 00, 01, 10 e 11 possuem aproximadamente o mesmo número. Sejam n_{00} , n_{01} , n_{10} e n_{11} as contagens do número de seqüências 00, 01, 10 e 11, respectivamente. É importante salientar que tais seqüências podem se sobrepor e, por consequência, $n_{00} + n_{01} + n_{10} + n_{11} = (n - 1)$.

A estatística $\nabla \mathcal{X}_2^2$ é usada no teste e o cálculo da mesma é obtido da seguinte forma:

$$\begin{aligned} \nabla \mathcal{X}_2^2 &= \mathcal{X}_2^2 - \mathcal{X}_1^2 \\ &= \left[\frac{4}{n-1} (n_{00}^2 + n_{01}^2 + n_{10}^2 + n_{11}^2) \right] - \left[\frac{2}{n} (n_{00}^2 + n_{11}^2) + 1 \right] \end{aligned}$$

A partir do resultado calcula-se o valor p do teste, da seguinte forma:

$$\begin{aligned} p &= \text{igamac} (2^{m-2}, \nabla \mathcal{X}_m^2) \\ &= \text{igamac} (2^{2-2}, \nabla \mathcal{X}_2^2) \\ &= \text{igamac} (1, \nabla \mathcal{X}_2^2) \end{aligned}$$

Caso $p < \alpha$ seja verdadeiro, rejeita-se a hipótese nula (equivalentemente, aceita-se H_a). Em caso contrário, H_0 é aceita.

4.3.6 Teste Pôquer

Seja m um inteiro positivo tal que $\lfloor \frac{n}{m} \rfloor \geq 5 \cdot (2^m)$ e $k = \lfloor \frac{n}{m} \rfloor$. Dividir a seqüência fornecida na entrada (ε) em k partes não sobrepostas, cada uma com comprimento m . Seja n_i o número de ocorrências do i -ésimo tipo de seqüência de comprimento m , em que $1 \leq i \leq 2^m$. O teste pôquer determinar quando as seqüências de comprimento m aparecem tal como esperado em uma seqüência distribuída uniformemente.

O teste pôquer é uma generalização do teste da frequência. Este teste foi proposto por Knuth [Knuth 1998] e recebe esta denominação por fazer uma analogia com o jogo pôquer,

Tabela 4.4: Quando $m = 5$, é possível fazer uma interpretação dos dígitos dos números como se fossem cartas de baralho em um jogo de pôquer.

Todas diferentes:	$abcde$	Full House:	$aaabb$
Um par:	$aabcd$	Quatro do mesmo tipo:	$aaaab$
Dois pares:	$aabbc$	Cinco do mesmo tipo:	$aaaaa$
Três do mesmo tipo:	$aaabc$		

em que os números podem ser interpretados como cartas de baralho na mão de um jogador quando $m = 5$, tal como descrito na Tabela 4.4.

A estatística utilizada no teste é obtida da seguinte forma:

$$X_1 = \frac{2^m}{k} \cdot \left(\sum_{i=1}^{2^m} n_i^2 \right) - k$$

que segue a distribuição χ^2 com $2^m - 1$ graus de liberdade.

O p valor deste teste é obtido da seguinte forma:

$$p = \text{igamac}(2^m - 1, X_1)$$

Caso $p < \alpha$ seja verdadeiro, rejeita-se a hipótese nula (equivalentemente, aceita-se H_a). Em caso contrário, H_0 é aceita.

4.3.7 Teste da Autocorrelação

A autocorrelação é uma medida que informa o quanto o valor de um dos elementos da seqüência de entrada é capaz de influenciar seus vizinhos. Por exemplo, o quanto a existência de um valor mais alto condiciona valores também altos de seus vizinhos na seqüência. Assim, o objetivo deste teste é verificar a existência e o nível de correlações entre a seqüência ε e em uma versão (não-cíclica) deslocada desta mesma seqüência.

Para tanto, calcula-se o valor $A(d)$ que corresponde aos bits distintos da seqüência original e da sua versão deslocada, em que d um inteiro fixo e $1 \leq d \leq \lfloor \frac{n}{2} \rfloor$. O cálculo de $A(d)$ faz uso da operação XOR, denotada por \oplus , que é igual a 1 quando os operandos são iguais e este cálculo é obtido da seguinte forma:

$$A(d) = \sum_{i=0}^{n-d-1} s_i \oplus s_{i+d}$$

A estatística utilizada no teste é:

$$X_2 = 2 \cdot \frac{\left(A(d) - \frac{n-d}{2}\right)}{\sqrt{n-d}}$$

que aproximadamente segue a distribuição normal no intervalo $(0, 1)$ se $n - d \geq 10$. O cálculo do p valor é efetuado da seguinte forma:

$$p = \text{erfc}(X_2)$$

Caso $p < \alpha$ seja verdadeiro, rejeita-se a hipótese nula (equivalentemente, aceita-se H_a). Em caso contrário, H_0 é aceita.

Capítulo 5

Atividades Realizadas

Este capítulo tem por objetivo descrever as atividades realizadas no estágio para a concretização dos objetivos propostos. O capítulo é dividido em três seções: análise, em que há a descrição do processo de desenvolvimento adotado e dos requisitos; apresentação e descrição da solução proposta, a qual se propõe a atender ao que havia sido solicitado pelos clientes; e avaliação da solução proposta, em termos de validação, verificação e comparativo com outras ferramentas de mesmo propósito.

5.1 Análise

A atividade de análise consistiu (*i*) na caracterização do processo de desenvolvimento a ser adotado e (*ii*) no levantamento de requisitos junto aos clientes. Tais etapas são descritas nas seções a seguir.

5.1.1 Processo de Desenvolvimento

Durante o planejamento das atividades do estágio, apresentado no Apêndice B, foi sugerido que o desenvolvimento priorizasse a implementação de todos os testes estatísticos, deixando para etapas posteriores a realização de testes de unidade e a documentação. Porém, quando a atividade de levantamento de requisitos foi iniciada, observou-se a necessidade de implementação de testes de unidade à medida que os testes estatísticos eram codificados. Além disto, a constante documentação consolidava a compreensão de cada passo que estava sendo

efetuado.

Deste modo, o processo de desenvolvimento proposto inicialmente, com características de um processo de desenvolvimento em cascata, foi substituído por um processo de desenvolvimento ágil, iterativo e incremental, baseado nas práticas sugeridas pelo XP1.

XP1 é um processo de desenvolvimento criado a partir do eXtreme Programming (XP) e foi concebido levando em consideração a execução de projetos de software desenvolvidos no meio acadêmico, no qual há demanda por um processo de desenvolvimento “magro”. É um processo que não requer um número grande de artefatos, mas que apesar disto permite que tarefas essenciais em qualquer processo de desenvolvimento de software sejam executadas; possui um baixo impacto diante de mudanças, em particular naquelas ligadas à elicitação de requisitos; e leva em consideração um tempo semanal de dedicação ao projeto inferior aos processos adotados pela indústria de software, por levar em conta as demais atividades que os alunos desenvolvem [Sauvé et al. 2002].

Dentre os três principais papéis sugeridos por XP1, neste estágio os orientadores acadêmicos e o supervisor técnico atuaram como *clientes* e a estagiária assumiu os papéis de *desenvolvedora* e *gerente*.

No papel de desenvolvedora, a estagiária descreveu as seguintes atividades:

1. Descrição das *user stories* e requisitos não funcionais comunicados oralmente pelos clientes;
2. Validação dos requisitos juntamente ao supervisor técnico;
3. Elaboração do projeto arquitetural, a ser apresentado na Seção 5.2.1;
4. Refatoração periódica no código da aplicação, particularmente priorizando esta atividade, a fim de evitar que a implementação, efetuada por uma só programadora, viesse a desrespeitar boas práticas de programação e *design*;
5. Elaboração de testes de unidade, cuja descrição é feita na Seção 5.3.2.

Como gerente de projeto, a estagiária desenvolveu as seguintes atividades:

1. Planejamento de atividades, iterações e releases junto aos clientes, conforme apresentadas no Apêndice C;

2. Respeito ao cronograma previamente estabelecido;
3. Constante avaliação dos riscos do projeto;
4. Informes periódicos ao supervisor técnico do status do projeto;
5. Consultas aos clientes sobre eventuais dúvidas que surgiram durante a implementação.

Houve a utilização de um SVN hospedado no *GoogleCode* para integração. Além disto, *backups* periódicos das diferentes versões do código foram efetuados utilizando o software *Dropbox*, responsável pela transferência automática dos dados de um computador para um servidor remoto de armazenamento de dados [Dropbox 2009].

Embora o processo de desenvolvimento adotado no estágio possua vários elementos sugeridos por XP1, a elaboração de testes de aceitação foi suprimida, pois a validação das funcionalidades implementadas ocorreu por meio de reuniões com o supervisor técnico, como será apresentado posteriormente.

Como é possível observar, as atividades do estágio contemplaram diversos elementos de XP1, a saber: especificação de user stories, refatoramento, testes de unidade, planejamento de atividades, iterações e releases, avaliação de riscos, reuniões frequentes com os clientes e integração de código.

5.1.2 Levantamento de Requisitos

Por meio de diálogos com os clientes, juntamente com investigações complementares em referências bibliográficas da área, houve a escolha de um conjunto de testes estatísticos adequados para aplicação em seqüências advindas de geradores aleatórios e pseudo-aleatórios. A partir da escolha e utilização destes testes estatísticos, houve a identificação dos requisitos funcionais da aplicação, descritos nas *user stories* a seguir:

1. Entrada de dados: o usuário deve fornecer uma seqüência numérica oriunda de um gerador aleatório ou pseudo-aleatório como entrada. Sob esta seqüência devem ser executados os testes estatísticos;
2. Testes estatísticos: após efetuar a entrada de dados, o usuário deve escolher e configurar pelo menos um dos testes listados a seguir.

- (a) Teste da Frequência;
- (b) Teste de Frequência dentro de um bloco;
- (c) Teste das Corridas;
- (d) Teste das Corridas em blocos;
- (e) Teste Serial;
- (f) Teste Pôquer;
- (g) Teste da Autocorrelação.

A configuração de cada teste consiste na quantidade de bits a serem analisados. Determinados testes necessitam de configurações adicionais. Para os testes dos itens (b) e (d) devem ser configurados o tamanho dos blocos e para o teste (g) deve ser configurado o comprimento do deslocamento.

3. Nível de significância: o usuário deve escolher o nível de significância no intervalo $(0, 1)$ para gerar conclusões para os testes;
4. Saída de dados: as conclusões e os p -valores obtidos a partir da execução dos testes estatísticos selecionados devem ser informados ao usuário.

Tomando como referência os requisitos funcionais descritos, o diagrama de caso de uso ilustrado na Figura 5.1 sintetiza a interação do usuário com a aplicação que avalia a qualidade dos geradores de números aleatórios e pseudo-aleatórios.

Baseando-se em solicitações dos clientes, aspectos observados em aplicações similares e também em geradores de números aleatórios e pseudo-aleatórios, houve a definição dos requisitos não funcionais da aplicação, apresentados a seguir:

1. As seqüências numéricas devem ser fornecidas em arquivos, visto que essa é uma saída de dados comum dos geradores aleatórios e pseudo-aleatórios;
2. Todos os números da seqüência numérica de entrada devem estar representados na mesma base, que pode ser binária ou decimal, ambas saídas típicas dos geradores;
3. Os números da seqüência devem ser utilizados pelos testes na ordem em que aparecem no arquivo. Em caso contrário, os testes podem resultar em conclusões incorretas;

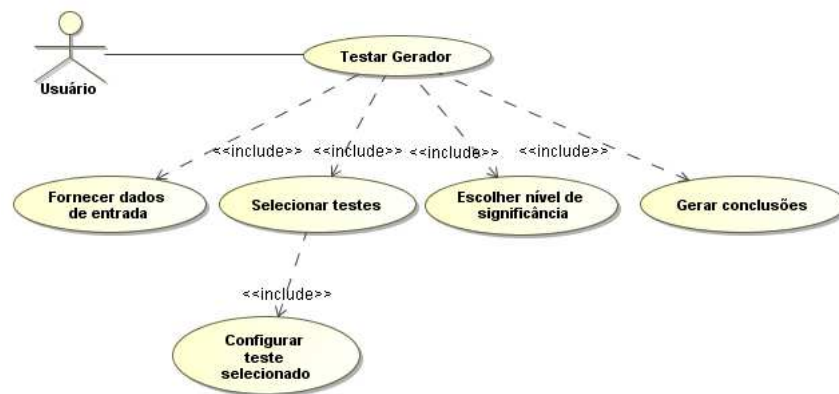


Figura 5.1: Funcionalidades que devem ser providas a um usuário da aplicação.

4. Os números podem vir a ter um número muito grande de dígitos;
5. A interface gráfica deve ser de fácil utilização;
6. A saída de dados deve ser amigável ao usuário, evitando, por exemplo, a impressão dos resultados numa tela orientada à caracteres;
7. A nomenclatura e as conclusões dos testes devem respeitar a linguagem e as asserções feitas no teste de hipótese.

É importante esclarecer que a elicitação dos requisitos foi feita de forma incremental e com a constante validação do supervisor técnico.

5.2 Solução Proposta

Com o intuito de atender aos requisitos funcionais e não funcionais especificados, foi desenvolvido um software, denominado *Sieve*.

O *Sieve* é capaz de testar seqüências produzidas por geradores de números aleatórios e pseudo-aleatórios, fornecidas em arquivo, por meio de um conjunto de sete testes estatísticos, os quais podem ser selecionados e configurados independentemente. As conclusões de boas características das seqüências e, conseqüentemente, dos seus respectivos geradores, é feita em razão de um nível de significância escolhido pelo usuário.

Ao final da execução dos testes, o *Sieve* apresenta um relatório, em formato *PDF*, contemplando o nome do arquivo fornecido como entrada, a data e horário da execução dos testes, o nível de significância escolhido, as conclusões e os *p*-valores obtidos. Este relatório pode ser anexado, por exemplo, na documentação de um projeto de pesquisa ou de uma aplicação que faça uso de geradores de números aleatórios e pseudo-aleatórios.

O *Sieve* é uma aplicação Desktop, multi-plataforma e de código livre, segundo a *GNU General Public License II* [GNU 1991]. A escolha por concebê-la como uma aplicação *open-source* deveu-se em virtude da possibilidade de ter a aplicação continuada por outros desenvolvedores ou grupos de pesquisa, consolidando o incremento de novos testes estatísticos na aplicação. Além disto, a comunidade científica poderia fazer uso do *Sieve* gratuitamente.

Nas seções a seguir serão apresentados em maiores detalhes os elementos que compõem e caracterizam o *Sieve*.

5.2.1 Arquitetura

O projeto arquitetural do *Sieve* foi concebido segundo o padrão arquitetural *Model-View-Controller* (MVC). De acordo com este padrão, a aplicação deve possuir três camadas (*model*, *view* e *controller*) que atuam de forma independente, possibilitando, por exemplo, o reuso da lógica em novas aplicações e minimizando o impacto das alterações de requisitos de interface sobre a camada de domínio.

Na camada *view*, encontram-se todos os componentes responsáveis pela interface gráfica com o usuário. Nesta camada são informados o arquivo de entrada, os testes selecionados e as suas respectivas configurações, além do nível de significância desejado.

As informações de entrada fornecidas pelo usuário são passadas para o *controller*, responsável por criar descritores de testes e repassá-los para as classes responsáveis pela execução. Um descritor de teste é um objeto que possui informações sobre os arquivos de entrada, formato dos dados, e configuração de um teste a ser executado.

Ao receber os descritores de teste, a camada *model* executa-os, gerando os *p*-valores e as conclusões, que são repassados para o *controller* e então para a *view*, na qual são dispostos no relatório, em formato *PDF*, e apresentados para o usuário. A Figura 5.2 apresenta a arquitetura do *Sieve*, em termos de suas camadas e da comunicação entre elas.

Além do padrão arquitetural, os padrões de projeto *Facade*, *Singleton* e *Factory Method*

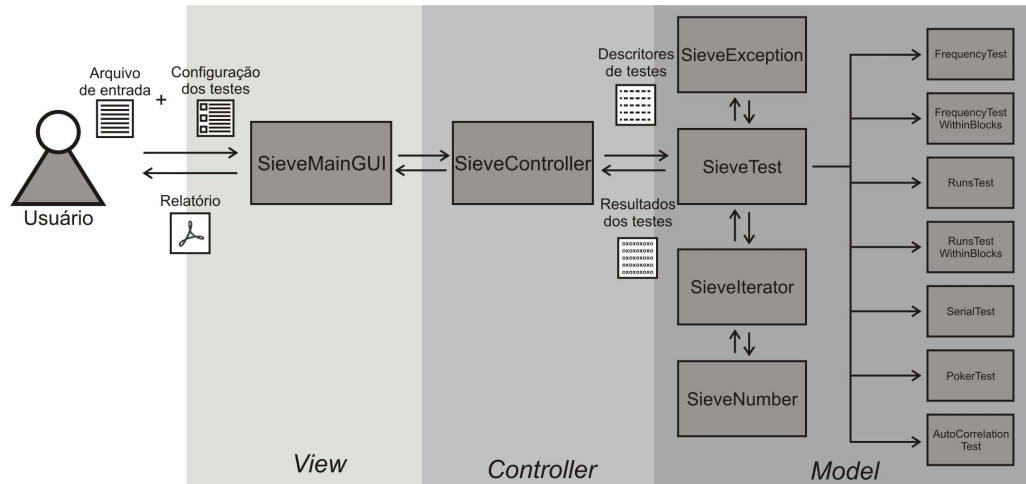


Figura 5.2: Arquitetura do Sieve.

foram adotados na implementação. Os padrões *Facade* e *Singleton* foram utilizado na classe `SieveController` com o intuito de prover uma interface única e simplificada para acesso e execução dos testes estatísticos; e o padrão *Factory Method* foi utilizado para instanciar os diferentes testes estatísticos a partir de uma classe abstrata comum, denominada `SieveTest`. A aplicação dos referidos padrões no *Sieve* é ilustrada na Figura 5.3.

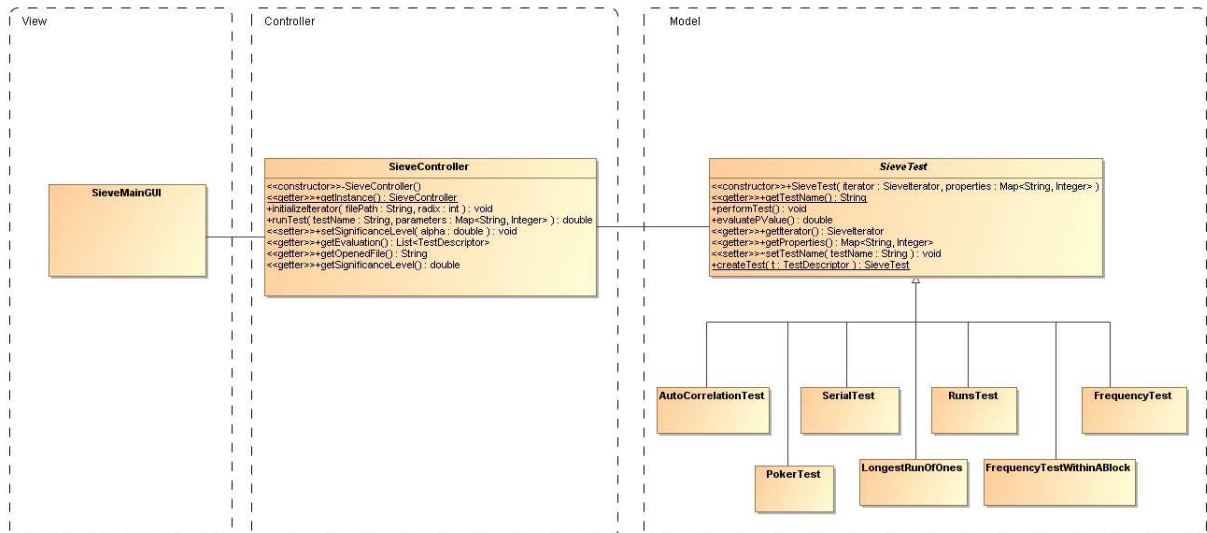


Figura 5.3: Utilização dos padrões *Facade*, *Singleton* e *Factory Method* no Sieve.

Para acessar os arquivos nos quais a seqüência numérica de entrada era fornecida, foi utilizado um iterador, denominado `SieveIterator`, o qual retorna números da seqüência de entrada à medida que a aplicação demanda. Isto evita um *overhead* na utilização de

memória principal, visto que não se conhece a priori o tamanho dos arquivos, e implica, conseqüentemente, em uma melhoria no desempenho da aplicação.

Com o intuito de promover uma uniformidade na representação numérica foi construída uma classe para este propósito, denominada `SieveNumber`. Esta classe é capaz de representar números em diferentes bases e com um grande número de dígitos, encapsulando a complexidade de manipulação dos mesmos. Além disto, também foi construída uma hierarquia de exceções capaz de capturar um conjunto de falhas durante a execução da aplicação.

5.2.2 Tecnologias Utilizadas

Para a implementação do *Sieve* diferentes tecnologias foram utilizadas: bibliotecas para cálculos matemáticos, *frameworks* para testes e construção de relatórios, etc. Todas estas tecnologias são apresentadas nas seções a seguir.

Java

A linguagem de programação Java [Sun Microsystems 2009], desenvolvida e mantida pela *Sun Microsystems*, foi a linguagem adotada para a implementação do *Sieve*. Sua escolha se deu por diferentes fatores:

1. Independência de plataforma: o código Java é executado em uma Máquina Virtual Java (JVM), a qual é capaz de executá-lo independente da arquitetura ou sistema operacional do computador ou dispositivo. Isto significa que o *Sieve* pode ser executado em qualquer ambiente sem a necessidade de compilações ou configurações especiais;
2. Suporte a documentação: uma vez que o *Sieve* está sob a licença GPL II, é imprescindível que seu código fonte esteja bem documentado, permitindo que outros desenvolvedores acrescentem testes ou funcionalidades. Por meio do Javadoc, foi possível documentar a aplicação (métodos, classes, atributos, etc.) de modo a tornar o código fonte facilmente legível por outros programadores;
3. Grande número de bibliotecas e *frameworks*: em virtude da grande aceitação da linguagem Java pelos programadores, foi possível a incorporação de funcionalidades,

- tais como geração de relatórios, cálculos matemáticos, etc., para melhor atender aos requisitos solicitados pelos clientes;
4. Suporte para desenvolvimento de interface gráfica: com as bibliotecas Swing e AWT, padrões de Java, é possível efetuar a construção de aplicações Desktop com diversos elementos que possibilitem uma boa interação com os usuários;
 5. A linguagem já era de conhecimento da estagiária: desde 2006 a estagiária possui experiência no desenvolvimento de aplicações com a linguagem Java. Uma vez que as atividades de implementação do estágio eram executadas sem o auxílio de terceiros, o domínio da tecnologia seria fundamental para a minimização de riscos para o cronograma do estágio.

Colt

A biblioteca Colt [CERN 2004] provê estruturas de dados de propósito geral para dados numéricos, tais como arrays redimensionáveis, arrays multi-dimensionais para representação de matrizes densas e esparsas, etc.; ferramentas matemáticas e estatísticas para análise de dados; geradores de números pseudo-aleatórios de determinadas distribuições; classes para computação paralela, etc. De modo geral, a biblioteca Colt provê infra-estrutura para computação científica em Java.

Neste estágio, a biblioteca Colt foi utilizada como parte integrante de alguns testes, nos quais algumas de suas funções auxiliaram nos cálculos necessários para a geração do p -valor a partir do χ^2 observado.

SSJ – Stochastic Simulation in Java

SSJ [L'Ecuyer 2009] é uma biblioteca Java para para simulação estocástica, desenvolvida pelo Departamento de Informática e Pesquisa Operacional da Universidade de Montreal, Canadá. É organizada em um conjunto de pacotes cujo propósito é facilitar a programação de simulações na linguagem Java.

Apesar desta biblioteca ser voltada para simulações, ela foi utilizada na solução proposta para auxiliar o cálculo dos p -valores que demandavam o cálculo da função complementar de erro.

JasperReports

JasperReports [Danciu 2008] é um *framework open-source* para geração de relatórios. Escrito em Java, apresenta grande habilidade na organização e apresentação de conteúdo, permitindo a geração dinâmica de relatórios em diversos formatos, como *PDF*, *HTML*, *XLS*, *CSV* e *XML*, podendo ser utilizado em qualquer aplicação Java, desktop ou web.

Neste estágio, a biblioteca JasperReports e a ferramenta gráfica para design de relatórios iReport foram utilizados para a criação e geração de relatórios, em formato PDF, contendo o resultado dos testes estatísticos sob as seqüências numéricas fornecidas como entrada.

JUnit

JUnit [Beck et al. 2009] é um *framework* para dar suporte a implementação de testes de unidade na linguagem Java. Com ele, é possível criar casos de teste com o intuito de localizar *bugs* e corrigi-los.

No *Sieve*, o JUnit foi utilizado para testar as principais classes da aplicação. Os `TestCase` construídos para o *Sieve* foram agrupados em uma `TestSuite`, a qual permitia a execução automática de todos os testes construídos. Em associação com o *plugin Coverage* para Eclipse [EclEmma 2009], foi possível verificar o percentual de linhas de código cobertas pelos testes.

5.2.3 Interface Gráfica

Nesta seção serão apresentadas os aspectos referentes a interface gráfica do *Sieve*.

A interface gráfica foi desenvolvida utilizando as bibliotecas AWT e Swing, padrões de Java para o desenvolvimento de interfaces gráficas de aplicações Desktop. Ao iniciar a aplicação, a tela inicial contendo o nome do software, a sua versão e o site são apresentados ao usuário, como pode ser visualizado na Figura 5.4.

Em seguida, é exibida a tela inicial do programa para o usuário, a qual solicita o arquivo de entrada contendo a seqüência numérica. Nessa tela, ilustrada na Figura 5.5, o usuário deve informar qual o tipo de representação numérica adotada (binária ou decimal).

Após a abertura bem sucedida do arquivo, na tela seguinte, ilustrada na Figura 5.6, o usuário deve selecionar e configurar os testes estatísticos disponíveis. Ao selecionar um teste,

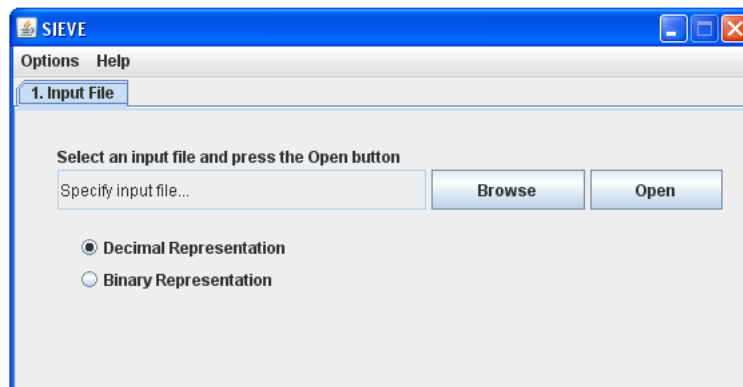
Figura 5.4: Tela inicial do *Sieve*.

Figura 5.5: Tela de entrada do arquivo que contém a seqüência numérica.

uma nova janela é aberta solicitando que os parâmetros de configuração sejam fornecidos.

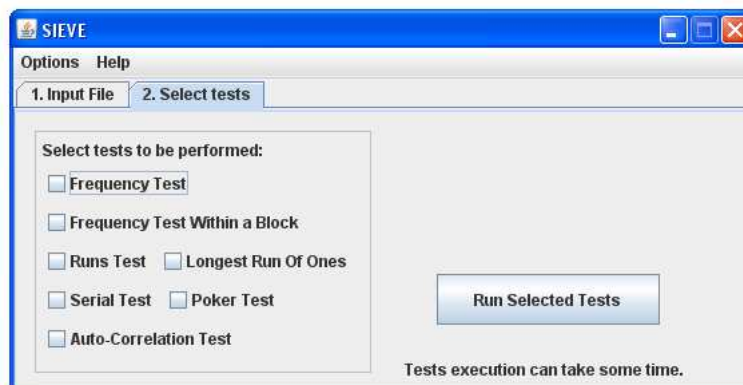


Figura 5.6: Tela de seleção e configuração dos testes.

Após a etapa de seleção dos testes, o usuário deve informar o nível de significância que deseja utilizar, conforme apresentado na Figura 5.7. O nível de significância, conforme descrito anteriormente, é um número no intervalo $[0, 1]$ que representa a probabilidade de

ocorrência do Erro Tipo *I*. Quanto maiores as exigências do usuário em relação à qualidade dos geradores, menor deve ser o nível de significância escolhido.

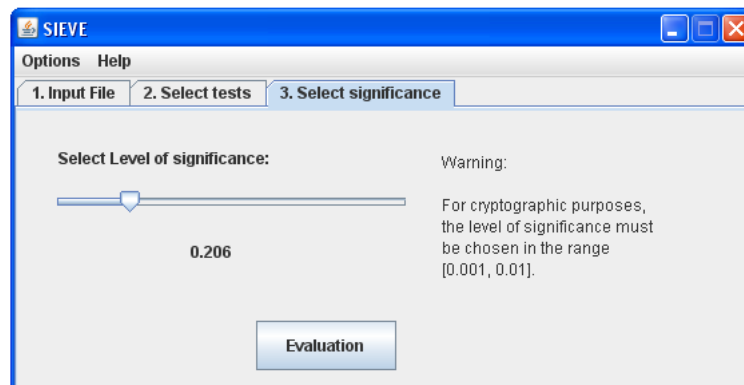


Figura 5.7: Tela de seleção do nível de significância.

Por fim, é apresentada a tela de resultados, ilustrada na Figura 5.8. Nesta tela o usuário tem acesso ao relatório da execução dos testes.

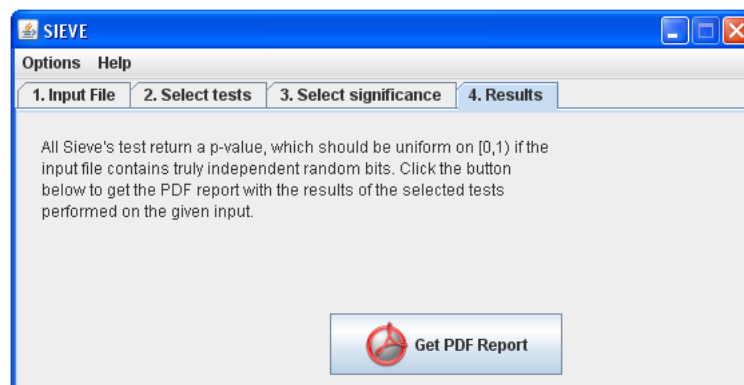


Figura 5.8: Tela de acesso aos resultados dos testes.

O relatório da execução dos testes consiste em um arquivo *PDF* contendo diversas informações relativas aos testes realizados. Um exemplo de relatório gerado é apresentado na Figura 5.9. Neste relatório, para uma sequência de entrada fornecida no arquivo denominado *excel*, foram executados os testes da Frequência, Corridas e Serial e o nível de significância escolhido foi 0.011. Para os dois primeiros testes, a hipótese nula foi aceita, ou seja, pode-se considerar que os números fornecidos pertencem a uma distribuição uniforme. No teste Serial não foi possível chegar a esta conclusão e, portanto, a hipótese nula foi rejeitada.

**Statistical Analysis of Numerical Sequences from Random
and Pseudorandom Number Generators - Version 1.0**<http://sieve.googlecode.com>

Report

Input File: C:\Documents and Settings\lolloa\workspace\SieveProject\files\demo\excel**Date:** Fri Jul 03 23:55:46 BRT 2009**Level of Significance:** 0.011

Results

Test Name	P-Value	Conclusion
Frequency Test	0.5485062355001472	The null hypothesis is accepted
Runs Test	0.6751754610804859	The null hypothesis is accepted
Serial Test	4.399606231546935E-35	The null hypothesis is rejected

Figura 5.9: Exemplo de relatório gerado pelo *Sieve*.

As telas apresentadas nesta seção ilustram a utilização do *Sieve*. Embora não tenham sido apresentadas neste relatório, existem alertas e mensagens exibidas em situações de erro, orientando o usuário em como proceder. Há ainda um barra de menus que permite reiniciar o processo de testes, impedindo que a aplicação precise ser reiniciada caso tenha mais de um arquivo para testar, e provendo acesso à ajuda e informações sobre a ferramenta.

5.2.4 Documentação

A geração de documentação faz referência à documentação do código-fonte produzido e à documentação de suporte ao usuário.

A documentação do código foi gerada por meio de Javadoc durante a implementação da aplicação, caracterizando cada classe, método e atributo criados. Esta documentação auxilia desenvolvedores na compreensão do funcionamento do *Sieve* e também na manutenção e atualização da ferramenta, características desejáveis em uma aplicação *open-source*.

A documentação de suporte ao usuário também foi gerada e estará disponível de forma *online* no site que hospeda o projeto. Tal disponibilização ainda não foi efetuada, pois a divulgação de informações a respeito da ferramenta poderia comprometer a publicação de um artigo que se tem em vista.

5.3 Avaliação da Solução Proposta

Esta seção se propõe a avaliar o *Sieve*. Esta avaliação será realizada de acordo com três critérios: (i) validação – a ferramenta atende ao que foi solicitado pelos clientes; (ii) verificação – a ferramenta está desenvolvida corretamente; e (iii) comparação com outros trabalhos relacionados.

5.3.1 Validação

A validação do *Sieve* foi efetuada por meio de três procedimentos os quais abordaram diferentes aspectos: validação junto ao cliente, a qual teve o intuito de avaliar se a aplicação estava de acordo com as necessidades do cliente; validação em relação ao nível de significância adotado, a qual teve o intuito de avaliar se os resultados dos testes respeitavam os

níveis de significância estabelecidos; e validação comparativa, a fim de avaliar a geração de conclusões do *Sieve* em comparação com outras ferramentas. A divisão da validação em procedimentos permitiu que diferentes aspectos fossem abordados e executados em paralelo e também facilitou o processo de re-trabalho, quando este foi necessário.

O primeiro procedimento foi efetuado de forma incremental, durante o desenvolvimento do *Sieve*, por meio de reuniões com o Supervisor Técnico do Estágio. A aplicação era executada passo-a-passo e o Supervisor Técnico solicitava correções e fazia sugestões, que eram incorporadas até o próximo encontro para execução da aplicação. Havia também reuniões periódicas com os demais clientes, que acompanhavam a evolução da ferramenta e faziam sugestões ou correções.

O segundo procedimento, cujos dados são reportado em detalhes na Seção D.1, verificou o respeito ao nível de significância estabelecido. Foi utilizado uma seqüências de números aleatórios com distribuição uniforme, oriundos da biblioteca Colt [CERN 2004], e esta seqüência foi subdividida em três conjuntos (Conjuntos 1, 2 e 3, respectivamente) com 100 arquivos de 1.000 bits cada.

Os testes do *Sieve* verificam a hipótese de a seqüência fornecida como entrada ser uma realização de uma variável aleatória com distribuição uniforme. Porém, neste segundo procedimento de validação, sabe-se *a priori* que os dados de entrada para os testes do *Sieve* possuem distribuição uniforme e, com isso, espera-se que o percentual de erro respeite o nível de significância estabelecido pelo usuário.

Neste segundo procedimento, cujos dados concludentes estão dispostos na Tabela 5.1, verifica-se que a média de ocorrências do Erro Tipo *I* no *Sieve* não superou o nível de significância estabelecido pelo usuário, conforme desejado. O resultado bem sucedido da execução deste procedimento consolidou mais um elemento da validação do *Sieve*.

O terceiro procedimento diz respeito à comparação da saída do *Sieve* com outras ferramentas. Para implementá-lo, foi utilizada um software análogo: a suíte de testes do NIST, que será apresentada em detalhes na Seção 5.3.3.

A suíte do testes do NIST possui 5 testes em comum com o *Sieve*, são eles: teste da freqüência, teste de freqüência dentro de um bloco, teste das corridas, teste das corridas em blocos e o teste serial. Embora a nomenclatura dos testes seja a mesma, diferem em alguns procedimentos internos e, em virtude disto, este terceiro procedimento de validação

Tabela 5.1: Médias e desvios-padrão da ocorrência de Erro Tipo *I* decorrente da execução dos testes do *Sieve* sob os Conjuntos 1, 2 e 3 com diferentes níveis de significância.

	Média	Desvio-Padrão
$\alpha = 0.1$	10,35	5,15
$\alpha = 0.3$	26,94	7,45
$\alpha = 0.5$	42,79	10,75
$\alpha = 0.9$	78,56	13,43

considera apenas a conclusão ao nível de significância de 15%.

Foram utilizados 4 arquivos de entrada contendo as expansões binárias de π , e , $\sqrt{2}$ e $\sqrt{3}$. Cada arquivo possui 1.000.000 de bits e são oriundos da etapa de validação do software do NIST [Rukhin et al. 2008].

Para cada teste a ser executado em ambas as ferramentas, foram utilizadas as mesmas condições: 10.000 bits de entrada e observação do p -valor em função do nível de significância (α) pré-estabelecido: se $p < \alpha$, rejeita-se a hipótese nula de que a seqüência de entrada possui distribuição uniforme; em caso contrário, aceita-se a hipótese nula. Nesta comparação, não foram levadas em consideração variáveis como o tempo de execução dos testes.

Os resultados obtidos, reportados em detalhes na Seção D.2, mostram que o *Sieve* rejeitou a hipótese nula em 17 ocorrências, enquanto que a suíte do NIST rejeitou tal hipótese em 16 ocorrências. Isto indica que houve conclusão análoga em 94, 11% dos testes executados.

O percentual de Erros Tipo *II* no *Sieve* foi igual a 15%, enquanto que na suíte de testes do NIST foi igual a 20%. Embora o percentual de erros no software do NIST tenha sido maior que o do *Sieve*, não se pode fazer afirmações em relação a ausência ou presença de erros de implementação na suíte de testes do NIST – tais investigações fogem ao escopo deste trabalho. A partir da execução do terceiro procedimento, pode-se concluir que o *Sieve* é capaz de gerar conclusões corretas ao mesmo nível de uma ferramenta análoga já consolidada, a suíte de testes do NIST.

A finalização do terceiro procedimento consolida a validação do *Sieve*, que se deu em três aspectos: validação junto ao cliente; validação em relação ao nível de significância estabelecido; e validação comparativa com outra ferramenta.

5.3.2 Verificação

Para a verificação do *Sieve* foram utilizados testes caixa-branca considerando a cobertura de linhas de código implementadas. Neste contexto, foi atingido 90% de cobertura nas classes que implementam os testes estatísticos. As faltas encontradas neste processo foram corrigidas.

O processo de verificação por meio de testes de unidade foi efetuado da seguinte forma: inicialmente, as funcionalidades foram segmentadas em atividades e estas, por sua vez, foram divididas em pequenas tarefas de implementação, testadas à medida que eram codificadas.

A construção dos testes de unidade foi efetuada utilizando o *framework* de testes JUnit e o auxílio do *plugin* Coverage, responsável por fornecer o percentual de linhas de código cobertas. Ao passo que os testes de unidade eram implementados, passavam a ser adicionados a uma *suíte* de testes, a qual possibilitou a execução dos mesmos de forma conjunta e automática.

5.3.3 Trabalhos Relacionados

Nesta seção serão apresentados dois softwares que realizam testes estatísticos em seqüências numéricas, o DIEHARD e a suíte de testes do NIST. Para ambos, será apresentada uma descrição geral, a quantidade de testes que possuem e algumas de suas características.

DIEHARD

Desenvolvida por Marsaglia [Marsaglia 1995], DIEHARD consiste em uma bateria de 18 testes estatísticos executados sob números inteiros que supostamente são realizações de uma variável aleatória discreta uniforme.

A entrada de dados no DIEHARD é feita por meio de arquivos binários nos quais os números pertencem ao intervalo $(0, 2^{31} - 1)$ e devem estar representados como inteiros de 32 bits sem sinal.

Esta aplicação não é configurável: os testes recebem uma seqüência de números com comprimento pré-estabelecido, resultando em erro em tempo de execução caso sejam menores que o especificado; e o usuário não informa o nível de significância, pois o DIEHARD retorna apenas os *p*-valores, deixando a cargo do usuário a geração de conclusões. Além

disto, os p -valores resultantes diferem dos comumente utilizados: quanto menor este valor, maior é a indicação de que a seqüência possui as características desejadas.

A interface gráfica do DIEHARD é orientada a caracteres e o resultado dos testes é informado em arquivos de saída de dados. A Figura 5.10 ilustra a seleção dos testes estatísticos disponíveis: o usuário informa uma seqüência de números em que 1 indica que o respectivo teste deve ser executado. No exemplo em questão, estão selecionados os testes 1, 3, 4, 7 e 9.

```

C:\WINDOWS\system32\cmd.exe - DIEHARD.EXE
with good RNG's. So keep in mind that "p happens".
Which tests do you want performed?
For all tests, enter 1's:
iiiiiiiiiiiiiiii
For, say, tests 1,3,7 and 14, enter
10100010000010
HERE ARE YOUR CHOICES:
1 Birthday Spacings
2 Overlapping Permutations
3 Ranks of 31x31 and 32x32 matrices
4 Ranks of 6x8 Matrices
5 Monkey Tests on 20-bit Words
6 Monkey Tests OPSO, OQSO, DNA
7 Count the 1's in a Stream of Bytes
8 Count the 1's in Specific Bytes
9 Parking Lot Test
10 Minimum Distance Test
11 Random Spheres Test
12 The Squeeze Test
13 Overlapping Sums Test
14 Runs Test
15 The Craps Test
Enter your choices, 1's yes, 0's no. using 15 columns:
123456789012345
101100101000000

```

Figura 5.10: Seleção de testes no DIEHARD.

Suíte de Testes do NIST

Desenvolvido pelo Departamento de Comércio do Instituto de Padrões e Tecnologias dos Estados Unidos (NIST), apresenta um conjunto de 12 testes estatísticos implementados na linguagem C [Rukhin et al. 2008]. Tais testes foram desenvolvidos para fins criptográficos e recebe como entrada valores binários com até 10^7 bits.

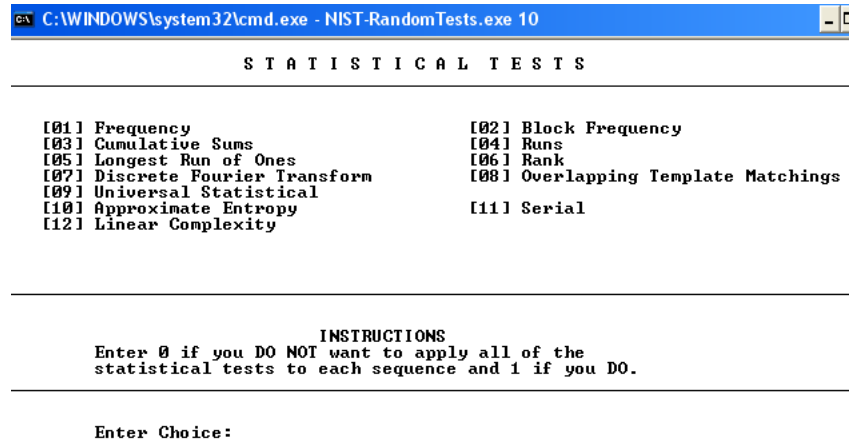
Assim como o DIEHARD, apresenta interface gráfica para o usuário orientada a caracteres e não permite que o usuário informe o nível de significância desejado. Portanto, deixa a cargo do usuário a conclusão da execução dos testes.

A suíte de testes do NIST permite que sejam configurados o formato de entrada dos arquivos (ASCII ou binário) e a quantidade de bits a serem testados. Os resultados são apresentados em um arquivo denominado `finalAnalysisReport` que é salvo na máquina após a execução dos testes.

Uma vantagem desta aplicação é que apresenta um conjunto de geradores pseudo-

aleatórios implementados e que, após a configuração das sementes, podem ser testados pela própria aplicação.

A Figura 5.11 ilustra a tela de seleção dos testes estatísticos na suíte do NIST e a Tabela 5.2 sintetiza as características dos softwares apresentados e do *Sieve*.



```
C:\WINDOWS\system32\cmd.exe - NIST-RandomTests.exe 10

S T A T I S T I C A L   T E S T S

[01] Frequency                [02] Block Frequency
[03] Cumulative Sums          [04] Runs
[05] Longest Run of Ones      [06] Rank
[07] Discrete Fourier Transform [08] Overlapping Template Matchings
[09] Universal Statistical      [11] Serial
[10] Approximate Entropy
[12] Linear Complexity

INSTRUCTIONS
Enter 0 if you DO NOT want to apply all of the
statistical tests to each sequence and 1 if you DO.

Enter Choice:
```

Figura 5.11: Seleção de testes no DIEHARD.

Tabela 5.2: Tabela comparativa de softwares destinados a análise estatística de seqüências numéricas.

Critérios	Software		
	DIEHARD	NIST	SIEVE
Entrada de dados	Arquivo	Geradores ou Arquivo	Arquivo
Quantidade de Testes	18	12	7
Configuração dos Testes	Não	Sim	Sim
Escolha de nível de significância	Não	Não	Sim
Geração de conclusões	Não	Não	Sim
Notação e convenções	Próprios	Padrão	Padrão
Interface com o usuário	Orientada a caracteres	Orientada a caracteres	Gráfica
Saída de dados	Arquivo-texto	Arquivo-texto	Relatório em formato PDF
Última versão	1995	2008	2009

Capítulo 6

Considerações Finais

O estágio foi uma oportunidade de lidar com uma nova área de pesquisa, pouco aprofundada durante a graduação e que demandou a articulação conjunta de conhecimentos vistos em diversas disciplinas do curso, a citar: Métodos Estatísticos, Engenharia de Software, Métodos e Software Numéricos, Sistemas de Informação, Probabilidade e Estatística, etc. Ou seja, embora tenha resultado em um software, pode-se considerar que foi consequência da fundamentação teórica sólida adquirida durante todo o estágio, acompanhada constantemente pelos orientadores e pelo supervisor, preocupados em assegurar que cada passo dado estava sendo corretamente compreendido

No tocante à implementação, o desafio de construir sozinha uma ferramenta, da especificação à interface gráfica, consolidou o aprendizado e permitiu a vivência de todas as etapas do desenvolvimento de um software, uma vez que na maioria das atividades da graduação estas atividades são compartilhadas com outros colegas de curso.

Deste modo, o estágio contribuiu de forma bastante significativa para o crescimento profissional e pessoal da estagiária, pois proporcionou um maior nível de auto-conhecimento, entendimento da capacidade de aprendizado, do esforço de desenvolvimento necessário para atingir certas metas e respeitar os prazos estabelecidos.

Espera-se que a ferramenta desenvolvida seja utilizada pela comunidade científica e de desenvolvimento de software, para dar suporte à utilização de geradores, e que também tenha funcionalidades acrescentadas por outros programadores, tornando o *Sieve* mais adequado para diferentes aplicações e contextos.

Referências Bibliográficas

- [Barbetta et al. 2008] Barbetta, P. A., Reis, M. M., and Bornia, A. C. (2008). *Estatística para cursos de Engenharia e Informática*. Editora Atlas.
- [Beck et al. 2009] Beck, K., Gamma, E., and Saff, D. (2009). Junit. Disponível para download em <http://www.junit.org/>. Último acesso em Julho de 2009.
- [Blum et al. 1986] Blum, L., Blum, M., and Shub, M. (1986). A simple unpredictable pseudorandom number generator. *SIAM Journal of Computing*, 15:364–383.
- [CERN 2004] CERN (2004). Colt – Open Source Libraries for High Performance Scientific and Technical Computing in Java. Disponível para download em <http://acs.lbl.gov/~hoschek/colt/index.html>.
- [Danciu 2008] Danciu, T. (2008). Jasper reports. Disponível para download em <http://ireport.sourceforge.net>.
- [Davis et al. 1994] Davis, D., Ihaka, R., and Fenstermacher, P. (1994). Cryptographic randomness from air turbulence in disk drives. In *CRYPTO '94: Proceedings of the 14th Annual International Cryptology Conference on Advances in Cryptology*, pages 114–120, London, UK. Springer-Verlag.
- [Dropbox 2009] Dropbox (2009). Secure backup, sync and sharing made easy. Acessado em Julho de 2009.
- [EclEmma 2009] EclEmma (2009). Java Code Coverage for Eclipse. Disponível em <http://www.eclEmma.org/> – Acessado em Julho de 2009.
- [Eichenauer and Lehn 1986] Eichenauer, J. and Lehn, J. (1986). A non-linear congruential pseudorandom number generator. *Statistische Hefte*, 27:315–326.

- [Entacher 2000] Entacher, K. (2000). A collection of classical pseudorandom number generators with linear structures – Advanced Version. Disponível em <http://crypto.mat.sbg.ac.at/results/karl/server/server.html>. Acessado em 14 de março de 2009.
- [Gentle 2003] Gentle, J. E. (2003). *Random Number Generation and Monte Carlo Methods*. Springer.
- [GNU 1991] GNU (1991). General Public License II. Disponível em <http://www.gnu.org/licenses/gpl-2.0.html> – Acessado em Julho de 2009.
- [GNU 2008] GNU (2008). The GNU Scientific Library – other random number generators. Disponível em <http://www.gnu.org/software/gsl/manual/>.
- [Gonçalves and Lopes 2003] Gonçalves, E. and Lopes, N. M. (2003). *Estatística – Teoria Matemática e Aplicações*. Escolar Editora.
- [Hoffstein et al. 2008] Hoffstein, J., Pipher, J., and Silverman, J. H. (2008). *An Introduction to Mathematical Cryptography*. Springer Publishing Company, Incorporated.
- [IBM 1968] IBM (1968). Randu – System/360 Scientific Subroutine package, Version III – Programmer’s Manual. Technical report, IBM, New York. p. 77.
- [Knuth 1998] Knuth, D. E. (1998). *The Art of Computer Programming – Volume 2 – Seminumerical Algorithms*. Addison-Wesley Publishing Company.
- [Koshy 2007] Koshy, T. (2007). *Elementary Number Theory with Applications*. Elsevier.
- [L’Ecuyer 1990] L’Ecuyer, P. (1990). Random numbers for simulation. *Communications of the ACM*, 33(10):85–97.
- [L’Ecuyer 1992] L’Ecuyer, P. (1992). Testing random number generators. In *Proceedings of the 1992 Winter Simulation Conference*, pages 305–313.
- [L’Ecuyer 2009] L’Ecuyer, P. (2009). Ssj: A Java Library for a Stochastic Simulation API Specification. Disponível para download em <http://www.iro.umontreal.ca/~simardr/ssj/>.

- [Lehmer 1951] Lehmer, D. H. (1951). Mathematical methods in large-scale computing units. *Proceedings of the Second Symposium on Large Scale Digital Computing Machinery*, 1:141–146.
- [Levin 1987] Levin, J. (1987). *Estatística Aplicada a Ciências Humanas*. Harbra.
- [Marsaglia 1968] Marsaglia, G. (1968). Random numbers fall mainly in the planes. *Proceedings of the National Academy of Sciences*, 61:25–28.
- [Marsaglia 1995] Marsaglia, G. (1995). The Marsaglia Random Number CDROM, including the DIEHARD Battery of Tests of Randomness. Disponível em <http://stat.fsu.edu/>. Último acesso em Julho de 2009.
- [Menezes et al. 1996] Menezes, A., van Oorschot, P., and Vanstone, S. (1996). *Handbook of Applied Cryptography*. CRC Press.
- [Park and Miller 1988] Park, S. and Miller, K. (1988). Random number generators: good ones are hard to find. *Communications of the ACM*, 31:1192–1201.
- [Revesz 1990] Revesz, P. (1990). *Random Walks in a Random and Non-Random Environments*. World Scientific. Cingapura.
- [Rukhin et al. 2008] Rukhin, A., Soto, J., Nechvatal, J., Smid, M., Barker, E., Leigh, S., Leveson, M., Vangel, M., Banks, D., Heckert, A., Dray, J., and Vo, S. (2008). A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. Technical report, National Institute of Standards and Technology – Technology Administration – U. S. Department of Commerce.
- [Sauvé et al. 2002] Sauvé, J. P., Silva, A. F., Neto, F. R. A., and Cabral, L. W. (2002). Xp1: Um processo de desenvolvimento. Disponível em <http://www.dsc.ufcg.edu.br/jacques/cursos/2002.2/projii/xp1/xp1.html>, acessado em Julho de 2009.
- [Sun Microsystems 2009] Sun Microsystems (2009). Java programming language. Disponível em <http://java.sun.com>, acessado em Julho de 2009.
- [Tausworthe 1965] Tausworthe, R. C. (1965). Random Numbers generated by linear recurrence modulo two. *Mathematics of Computation*, 19:201–209.

- [Trevisan 1998] Trevisan, L. (1998). Constructions of near-optimal extractors using pseudo-random generators. In *Electronic Colloquium on Computational Complexity*, volume 55.
- [Williams and Clearwater 1998] Williams, C. P. and Clearwater, S. H. (1998). *Explorations in Quantum Computing*. The Electronic Library of Science.

Apêndice A

Conceitos Matemáticos Utilizados

O texto relativo a Aritmética Modular é baseado nas obras de Gentle e Koshy [Gentle 2003, Koshy 2007] e os conceitos sobre variáveis aleatórias foram obtidos na obra de Barbeta et. al [Barbeta et al. 2008]. Estas obras podem ser consultadas para obtenção do conteúdo com um maior nível de detalhamento, além de outros exemplos resolvidos e exercícios para fixação.

A.1 Aritmética Modular

Alguns dos conceitos mais importantes da Teoria dos Números são relativos a Aritmética Modular, introduzida e desenvolvida pelo matemático alemão Karl Friedrich Gauss.

Os métodos congruentes e de registradores de deslocamento para a geração de números pseudo-aleatórios utilizam alguns dos conceitos da aritmética modular na geração de seqüências. Em virtude disto, os conceitos de interesse serão abordados nas subseções a seguir.

A.1.1 Congruência módulo m

Seja m um inteiro positivo ($m \in \mathbb{Z}_+$). Um inteiro a é congruente a um inteiro b módulo m se a diferença entre eles é de um inteiro divisível por m . Esta relação é denotada por:

$$a \equiv b \pmod{m}$$

Por exemplo, 5 e 14 são congruentes módulo 3, ou seja, $5 \equiv 14 \pmod{3}$, pois $14/3 =$

$3 \cdot 4 + 2$ e $5/3 = 3 + 2$. Mas, 18 e -6 não são congruentes módulo 7 ($18 \not\equiv -6 \pmod{7}$), pois $18 = 7 \cdot 2 + 4$ e $-6 = (-1) \cdot 7 + 1$, ou seja, $4 \not\equiv 1$

A congruência módulo m apresenta três propriedades:

1. Simetria

$$a \equiv b \pmod{m} \Leftrightarrow b \equiv a \pmod{m}$$

2. Reflexividade

$$a \equiv a \pmod{m}, \forall a$$

3. Transitividade

$$a \equiv b \pmod{m}, b \equiv c \pmod{m} \Rightarrow a \equiv c \pmod{m}$$

A.1.2 Redução Módulo m

Dado um inteiro b , o inteiro a que satisfaz as condições $a \equiv b \pmod{m}$ e $0 \leq a < m$ é a redução módulo m de b .

A redução módulo m de b pode ser também definida como:

$$a = b - \left\lfloor \frac{b}{m} \right\rfloor \cdot m$$

em que $\lfloor \cdot \rfloor$ indica o maior inteiro menor ou igual ao argumento (função chão).

Por exemplo, a redução módulo 4 de 18 é igual a 2, pois $2 \equiv 18 \pmod{4}$ e $0 \leq 2 < 4$. Embora 18 seja congruente módulo 4 a 22, isto é $22 \equiv 18 \pmod{4}$, 18 não é a redução módulo 4 de 22, pois $0 \leq 18 \not< 22$.

A.1.3 Classes Residuais

Como consequência da redução módulo m , o conjunto dos inteiros pode ser particionado em m classes não-vazias e disjuntas, denominadas classes residuais.

Por exemplo, para $m = 4$, existem 4 classes residuais:

$$\{\dots, -8, -4, 0, 4, 8, \dots\}$$

$$\{\dots, -9, -5, 1, 5, 9, \dots\}$$

$$\{\dots, -10, -6, 2, 6, 10, \dots\}$$

$$\{\dots, -11, -7, 3, 7, 11, \dots\}$$

As classes residuais cujos membros são primos relativos a m são os valores da função totiente de Euler, denotada por $\phi(m)$. Existem três situações em que é possível utilizar regras para facilitar o cálculo da função ϕ :

1. Quando m é primo, então

$$\phi(m) = m - 1$$

2. Quando m é primo e e é um inteiro positivo

$$\phi(m^e) = m^{e-1}(m - 1)$$

3. Quando m e n são primos relativos

$$\phi(m \cdot n) = \phi(m) \cdot \phi(n)$$

A.1.4 Inverso Multiplicativo

Anteriormente ao conceito de inverso multiplicativo, faz-se necessário o entendimento de primos relativos. Dois números a e b são primos relativos entre si caso o máximo divisor comum (gcd – greatest common divisor) entre eles seja igual a 1 ($\text{gcd}(a, b) = 1$).

O inverso multiplicativo de x módulo m , denotado por x^- , é definido para todos os valores x primos relativos de m da seguinte forma:

$$1 \equiv x \cdot x^- \pmod{m}$$

Por exemplo, seja $m = 11$. Os primos relativos de m são os possíveis valores de x : 2, 3, 5 e 7. Para $x = 2$, o valor de x^- , se houver, é tal que:

$$1 \equiv 2 \cdot x^- \pmod{11}$$

O valor de x^{-} que satisfaz esta equivalência é 28, pois:

$$\begin{aligned} 1 &\equiv 2 \cdot x^{-} \pmod{11} \\ &\equiv 2 \cdot 28 \pmod{11} \\ &\equiv 56 \pmod{11} \\ &\equiv 1. \end{aligned}$$

A.2 Reticulados

Sejam $v_1, \dots, v_n \in \mathbb{R}^n$ um conjunto de vetores linearmente independentes. Um *reticulado* R gerado por v_1, \dots, v_n com coeficientes em \mathbb{Z} é:

$$R = \{a_1 \cdot v_1 + a_2 \cdot v_2 + \dots + a_n \cdot v_n \mid a_1, \dots, a_n \in \mathbb{Z}\}.$$

Uma *base* para R é qualquer conjunto de vetores linearmente independentes que gera R . A *dimensão* de um reticulado é a quantidade de vetores que compõe uma base deste reticulado. O *domínio fundamental* de R relativo a base v_1, \dots, v_n é o conjunto:

$$\mathcal{F}(v_1, \dots, v_n) = \{t_1 \cdot v_1 + t_2 \cdot v_2 + \dots + t_n \cdot v_n \mid 0 \leq t_1 < 1\}$$

Um reticulado é similar a um espaço vetorial exceto pelo fato de ser gerado por todas as combinações lineares dos vetores da base utilizando coeficientes inteiros. Nos espaços vetoriais, os coeficientes são números reais. É comum denotar reticulados como um conjunto ordenado de pontos em \mathbb{R}^n , em que cada ponto é o extremo de cada vetor. Por exemplo, a Figura A.1 ilustra um reticulado em \mathbb{R}^2 .

A.3 Variáveis Aleatórias

Uma variável aleatória discreta pode ser entendida como uma variável quantitativa, cujo resultado (valor) depende de fatores aleatórios.

São exemplos de variáveis aleatórias:

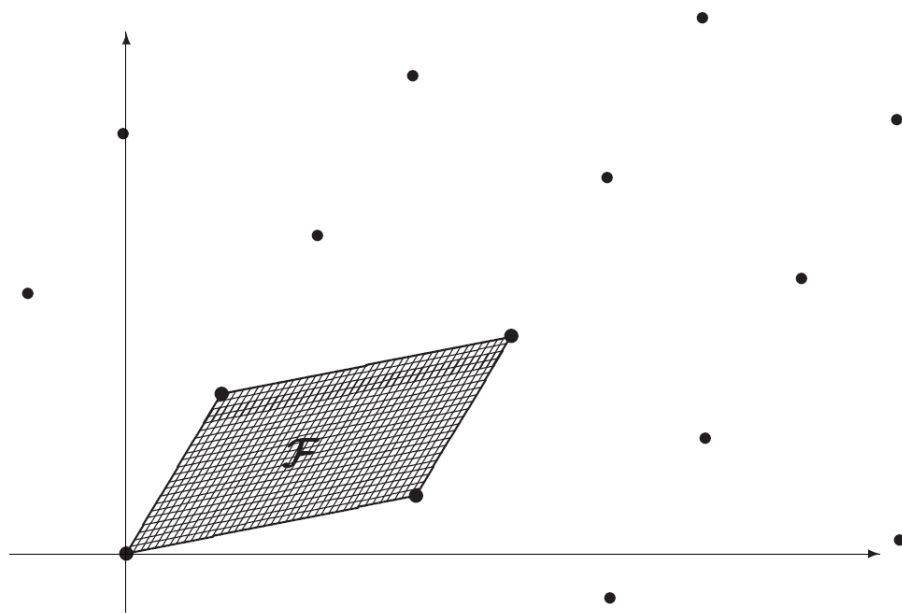


Figura A.1: Exemplo de um reticulado em \mathbb{R}^2 . Na área sombreada destaca-se o domínio fundamental deste reticulado.

1. Número de coroas obtido no lançamento de duas moedas;
2. Número de pessoas que visitam um determinado site, num certo período de tempo;
3. Tempo de resposta de um sistema computacional, etc.

Formalmente, uma variável aleatória é uma função que associa elementos do espaço amostral ao conjunto de números reais. Variáveis aleatórias discretas são aquelas que podem ser representadas como variáveis indicativas, por exemplo, 0 ou 1.

Os valores que uma variável aleatória X pode assumir são (x_1, x_2, \dots) , que significam os resultados que podem ocorrer. Diz-se que x_1, x_2, \dots pertencem a um alfabeto, denotado por \mathcal{X} .

Para cada valor que X pode assumir está associada uma probabilidade de ocorrência:

$$p(x_i) = P(X = x_i) (i = 1, 2, \dots)$$

A associação de uma probabilidade a cada possível ocorrência de X caracteriza uma função de probabilidade, que deve satisfazer dois requisitos:

Tabela A.1: Distribuição de probabilidades da variável aleatória X .

Valores Possíveis	Probabilidades
x_1	p_1
x_2	p_2
x_3	p_3
...	...
x_k	p_k
Total	1

1. $p(x_i) \geq 0$, ou seja, as probabilidades associadas a cada ocorrência de X não devem ser negativas;
2. $\sum_i p(x_i) = 1$, ou seja, o somatório das probabilidades associadas às ocorrências de X deve ser igual a 1.

A.3.1 Valor esperado, Variância e Desvio Padrão

Seja uma variável aleatória X e sua distribuição de probabilidades, descrita na tabela A.1.

A média (ou valor esperado) de X é dado por:

$$\mu = E(X) = \sum_{j=1}^k x_j \cdot p_j$$

Seja Y uma outra variável aleatória e c uma constante qualquer. A média satisfaz as seguintes propriedades:

1. $E(c) = c$;
2. $E(X + c) = E(X) + c$;
3. $E(c \cdot X) = c \cdot E(X)$;
4. $E(X + Y) = E(X) + E(Y)$;
5. $E(X - Y) = E(X) - E(Y)$.

A variância de X pode ser calculada por:

$$\sigma^2 = V(X) = \sum_{j=1}^k (x_j - \mu)^2 \cdot p_j$$

A variância de uma variável aleatória também satisfaz algumas propriedades, apresentadas abaixo:

1. $V(c) = 0$;
2. $V(X + c) = V(X)$;
3. $V(c \cdot X) = c^2 V(X)$.

O desvio padrão de X é dado por:

$$\sigma = DP(X) = \sqrt{Var(X)}$$

O desvio padrão satisfaz apenas uma propriedade: $DP(c \cdot X) = |c| \cdot DP(X)$.

A.4 Distribuições de Probabilidade

Esta seção tem por objetivo apresentar uma caracterização das distribuições de probabilidades que serão utilizadas. Os conceitos matemáticos são oriundos das obras de Gonçalves e Lopes [Gonçalves and Lopes 2003] e Barbetta et al. [Barbetta et al. 2008], cuja consulta é sugerida para uma explicação mais aprofundada destes conceitos.

A.4.1 Distribuição Uniforme

Uma variável aleatória X possui distribuição uniforme de parâmetros α e β , sendo $\beta > \alpha$, se a sua função de distribuição de probabilidades é dada por:

$$f(x) = \begin{cases} \frac{1}{\beta - \alpha}, & \text{para } x \in [\alpha, \beta] \\ 0, & \text{para } x \notin [\alpha, \beta] \end{cases}$$

O valor esperado de uma distribuição uniforme é exatamente o ponto médio do intervalo $[\alpha, \beta]$:

$$E(X) = \frac{\alpha + \beta}{2}$$

e a variância representa o centro de gravidade da função de densidade de probabilidade de uma distribuição uniforme, cujo valor é dado por:

$$V(X) = \frac{(\beta - \alpha)^2}{12}$$

A.4.2 Distribuição de Bernoulli

Seja uma variável aleatória discreta X que pode assumir 2 valores: 0 que representa insucesso e o valor 1 que representa o sucesso. Ao sucesso está associada a probabilidade p e ao insucesso a probabilidade $q = 1 - p$.

Diz-se que X tem distribuição de Bernoulli, e denota-se por $X \sim B(p)$ em que $p \in]0, 1[$, se a função de probabilidade desta variável é dada por:

$$P[X = x] = f(x) \begin{cases} p, & \text{se } x = 1 \\ q, & \text{se } x = 0 \\ 0, & \text{se } x \text{ assume outros valores} \end{cases}$$

A distribuição de Bernoulli só tem um parâmetro: p que satisfaz a condição $0 < p < 1$.

A.4.3 Distribuição Normal

Diz-se que a variável aleatória contínua X segue a lei normal, e denota-se por $X \sim N(m, \sigma)$ se a sua função de distribuição de probabilidades for dada por:

$$f(x) = \frac{1}{\sigma \cdot \sqrt{2 \cdot \pi}} \cdot e^{-\frac{1}{2} \left(\frac{x-m}{\sigma} \right)^2}$$

em que $-\infty < x < +\infty$. Os parâmetros caracterizadores da distribuição, $-\infty < m < +\infty$ e $\sigma > 0$, representam, respectivamente, a média e o desvio-padrão da distribuição.

A.4.4 Distribuição Binomial

A distribuição binomial corresponde a um esquema probabilístico adaptado a situações em que se pretende analisar um conjunto finito (ou amostra) de indivíduos/objectos que possuem

determinado atributo com probabilidade p ou que não o possuem com uma probabilidade $q = 1 - p$.

Diz-se que a variável aleatória discreta X – numero de sucessos em n provas de Bernoulli – tem distribuição binomial, denotada por $X \sim B(n, p)$ em que $p \in]0, 1[$ e $n \in \mathbb{N}$, se a sua se a sua função de probabilidade for dada por:

$$P[X = x] = f(x) \begin{cases} \binom{n}{x} \cdot p^x \cdot (1 - p)^{(n-x)}, & \text{se } x = 0, 1, \dots, n \\ 0, & \text{se } x \text{ assume outros valores} \end{cases}$$

em que n e p são os parâmetros caracterizadores desta distribuição. O parâmetro n corresponde ao numero de provas de Bernoulli a efetuar, sendo n qualquer inteiro positivo, e p corresponde a probabilidade associada ao sucesso, com $0 < p < 1$.

A.4.5 Distribuição \mathcal{X}^2

Diz-se que uma variável aleatória segue a distribuição \mathcal{X}^2 (chi-quadrado) com v graus de liberdade, $v \in]0, +\infty[$, quando a sua função densidade de probabilidade tem a forma:

$$f(x) = \frac{1}{2^{\frac{v}{2}} \cdot \Gamma\left(\frac{v}{2}\right)} \cdot e^{-\frac{x}{2}} \cdot x^{\frac{v}{2}-1}$$

em que $x \geq 0$ e Γ denota a função:

$$\Gamma(y) = \int_0^{+\infty} e^{-x} \cdot x^{y-1} dx, y > 0$$

O parâmetro caracterizador desta distribuição é v , que indica o numero de graus de liberdade. É importante salientar que a distribuição \mathcal{X}^2 é um caso particular de uma outra distribuição, denominada distribuição gama.

A.5 Funções Matemáticas Auxiliares

Duas funções matemáticas auxiliarem foram utilizadas: a função complementar de erro e a função gama completa incompleta. Tais funções encontram-se distribuídas no intervalo $[0, 1]$ e possuem distribuições idênticas à função de distribuição normal-padrão cumulativa e \mathcal{X}^2 com a graus de liberdade, respectivamente.

A.5.1 Função Complementar de Erro

A função complementar de erro, denotada por $erfc$, é definida por:

$$erfc(z) = \frac{2}{\sqrt{\pi}} \int_z^{\infty} e^{-u^2} du$$

A função complementar de erro é essencialmente idêntica à função de distribuição normal-padrão cumulativa.

A.5.2 Função Gama Complementar Incompleta

A função gama complementar incompleta, denotada por $igamac$, é definida por:

$$igamac(a, x) \equiv 1 - P(a, x) \equiv \frac{\Gamma(a, x)}{\Gamma(a)} \equiv \frac{1}{\Gamma(a)} \int_x^{\infty} e^{-t} t^{a-1} dt$$

em que $igamac(a, 0) = 1$ e $igamac(a, \infty) = 0$.

A função gama complementar incompleta é essencialmente idêntica à função de distribuição \mathcal{X}^2 com a graus de liberdade.

Apêndice B

Plano de Estágio

Apêndice C

Cronograma

O cronograma das atividades realizadas durante o estágio encontra-se na Tabela C.1. As atividades estão organizadas em iterações, com suas respectivas datas de início e término.

As iterações de construção dos testes estatísticos compreenderam não somente a implementação dos mesmos, como também a documentação do código, os testes de unidade e a incorporação na interface gráfica. Além disto, a escrita e a revisão do relatório de estágio foram feitas de forma incremental durante todo o trabalho.

Tabela C.1: Cronograma das atividades realizadas durante o estágio

Iteração	Atividade	Data de início	Data de término
Iteração 0	Fundamentação Teórica	16/mar	3/abr
Iteração 0	Definição e priorização das <i>user stories</i>	6/abr	10/abr
Iteração 0	Estudos dos procedimentos para desenvolvimento de aplicações científicas com a linguagem Java	13/abr	17/abr
Iteração 1	Definição da arquitetura e padrões adotados na aplicação	20/abr	22/abr
Iteração 1	Implementação das classes principais	23/abr	1/mai
Iteração 2	Implementação da interface gráfica	4/mai	15/mai
Iteração 3	Implementação do Teste da Frequência	18/mai	22/mai
Iteração 4	Implementação do Teste da Frequência dentro de um bloco	25/mai	29/mai
Release 1			
Iteração 5	Refatoramento	30/mai	31/mai
Iteração 5	Implementação do Teste das Corridas	1/jun	6/jun
Iteração 6	Implementação do Teste das Corridas Dentro de um Bloco	8/jun	12/jun
Iteração 6	Refatoramento	13/jun	14/jun
Iteração 7	Implementação do Teste Serial	15/jun	19/jun
Iteração 8	Implementação do Teste Pôquer	22/jun	26/jun
Iteração 9	Implementação do Teste da Autocorrelação	29/jun	3/jul
Release 2			
Iteração 10	Refatoramento	4/jul	5/jul
Iteração 10	Execução de testes adicionais na aplicação	6/jul	10/jul
Iteração 11	Avaliação da aplicação	6/jul	10/jul
Iteração 12	Melhorias na interface gráfica	13/jul	18/jul
Iteração 12	Refatoramento	19/jul	20/jul
Release Final			

Apêndice D

Dados da Validação

Neste apêndice são apresentados os dados que possibilitaram as conclusões apresentadas na seção de Validação, no Capítulo 5.

D.1 Nível de Significância

As tabelas apresentadas a seguir correspondem aos resultados da execução dos testes estatísticos pelo *Sieve*. A entrada de dados e as conclusões dos testes são apresentadas na Seção 5.3.1. Cada conjunto no qual os testes foram executados é composto de 100 arquivos com 1000 bits cada.

Tabela D.1: Resultado da execução do Teste da Frequência com diferentes níveis de significância.

	Conjunto 1	Bloco 2	Conjunto 3
$\alpha = 0.1$	8	17	11
$\alpha = 0.3$	29	35	32
$\alpha = 0.5$	49	50	49
$\alpha = 0.9$	93	93	87

Tabela D.2: Resultado da execução do Teste da Frequência Dentro de um Bloco com diferentes níveis de significância. Cada bloco possui 10 bits.

	Conjunto 1	Conjunto 2	Conjunto 3
$\alpha = 0.1$	3	9	7
$\alpha = 0.3$	32	35	30
$\alpha = 0.5$	45	54	55
$\alpha = 0.9$	86	94	93

Tabela D.3: Resultado da execução do Teste da Frequência Dentro de um Bloco com diferentes níveis de significância. Cada bloco possui 50 bits.

	Conjunto 1	Conjunto 2	Conjunto 3
$\alpha = 0.1$	10	11	10
$\alpha = 0.3$	28	39	35
$\alpha = 0.5$	46	59	59
$\alpha = 0.9$	90	95	91

Tabela D.4: Resultado da execução do Teste da Frequência Dentro de um Bloco com diferentes níveis de significância. Cada bloco possui 100 bits.

	Conjunto 1	Conjunto 2	Conjunto 3
$\alpha = 0.1$	12	13	9
$\alpha = 0.3$	31	33	32
$\alpha = 0.5$	53	60	55
$\alpha = 0.9$	86	95	93

Tabela D.5: Resultado da execução do Teste das Corridas com diferentes níveis de significância.

	Conjunto 1	Conjunto 2	Conjunto 3
$\alpha = 0.1$	9	3	5
$\alpha = 0.3$	17	14	19
$\alpha = 0.5$	30	24	26
$\alpha = 0.9$	46	44	43

Tabela D.6: Resultado da execução do Teste das Corridas em Blocos com diferentes níveis de significância. Cada bloco possui 128 bits.

	Conjunto 1	Conjunto 2	Conjunto 3
$\alpha = 0.1$	10	9	6
$\alpha = 0.3$	25	30	18
$\alpha = 0.5$	48	46	37
$\alpha = 0.9$	93	89	92

Tabela D.7: Resultado da execução do Teste Serial com diferentes níveis de significância.

	Conjunto 1	Conjunto 2	Conjunto 3
$\alpha = 0.1$	12	3	6
$\alpha = 0.3$	22	16	21
$\alpha = 0.5$	31	28	27
$\alpha = 0.9$	64	70	69

Tabela D.8: Resultado da execução do Teste Pôquer com diferentes níveis de significância. Cada número é composto por 5 bits (mão de pôquer).

	Conjunto 1	Conjunto 2	Conjunto 3
$\alpha = 0.1$	19	18	16
$\alpha = 0.3$	34	32	35
$\alpha = 0.5$	45	45	47
$\alpha = 0.9$	72	77	81

Tabela D.9: Resultado da execução do Teste Pôquer com diferentes níveis de significância. Cada número é composto por 10 bits.

	Conjunto 1	Conjunto 2	Conjunto 3
$\alpha = 0.1$	21	22	23
$\alpha = 0.3$	31	40	40
$\alpha = 0.5$	58	57	57
$\alpha = 0.9$	77	78	79

Tabela D.10: Resultado da execução do Teste da Autocorrelação com diferentes níveis de significância. O shift é de 1 bit.

	Conjunto 1	Conjunto 2	Conjunto 3
$\alpha = 0.1$	7	12	12
$\alpha = 0.3$	15	22	24
$\alpha = 0.5$	31	37	39
$\alpha = 0.9$	77	81	70

Tabela D.11: Resultado da execução do Teste da Autocorrelação com diferentes níveis de significância. O shift é de 10 bits.

	Conjunto 1	Conjunto 2	Conjunto 3
$\alpha = 0.1$	5	8	9
$\alpha = 0.3$	23	23	30
$\alpha = 0.5$	37	36	41
$\alpha = 0.9$	76	72	76

Tabela D.12: Resultado da execução do Teste da Autocorrelação com diferentes níveis de significância. O shift é de 25 bits.

	Conjunto 1	Conjunto 2	Conjunto 3
$\alpha = 0.1$	10	10	6
$\alpha = 0.3$	24	22	18
$\alpha = 0.5$	36	38	33
$\alpha = 0.9$	74	71	70

Tabela D.13: Resultado da execução do Teste da Autocorrelação com diferentes níveis de significância. O shift é de 40 bits.

	Conjunto 1	Conjunto 2	Conjunto 3
$\alpha = 0.1$	12	7	4
$\alpha = 0.3$	24	17	14
$\alpha = 0.5$	39	35	27
$\alpha = 0.9$	78	73	76

D.2 Comparação com o NIST

As tabelas apresentadas a seguir correspondem aos resultados da execução dos testes estatísticos utilizando o *Sieve* comparados com os resultados da suíte do NIST, para os testes que estão presentes em ambas as ferramentas. Para cada teste executado, foram fornecidos 10.000 bits de entrada. As demais informações e conclusões são apresentadas na Seção 5.3.1.

Tabela D.14: Resultado da execução dos testes para a expansão binária de π com nível de significância igual a 0.1

	Sieve	NIST
Teste da Frequência	Rejeita	Rejeita
Teste de Frequência Dentro de um Bloco ($m = 128$)	Aceita	Aceita
Teste das Corridas	Aceita	Rejeita
Teste das Corridas em Blocos	Rejeita	Aceita
Teste Serial	Rejeita	Rejeita

Tabela D.15: Resultado da execução dos testes para a expansão binária de e com nível de significância igual a 0.15

	Sieve	NIST
Teste da Frequência	Rejeita	Rejeita
Teste de Frequência Dentro de um Bloco ($m = 128$)	Rejeita	Rejeita
Teste das Corridas	Rejeita	Rejeita
Teste das Corridas em Blocos	Rejeita	Rejeita
Teste Serial	Rejeita	Rejeita

Tabela D.16: Resultado da execução dos testes para a expansão binária de $\sqrt{2}$ com nível de significância igual a 0.15

	Sieve	NIST
Teste da Frequência	Rejeita	Rejeita
Teste de Frequência Dentro de um Bloco ($m = 128$)	Aceita	Rejeita
Teste das Corridas	Rejeita	Rejeita
Teste das Corridas em Blocos	Rejeita	Aceita
Teste Serial	Rejeita	Rejeita

Tabela D.17: Resultado da execução dos testes para a expansão binária de $\sqrt{3}$ com nível de significância igual a 0.15

	Sieve	NIST
Teste da Frequência	Rejeita	Rejeita
Teste de Frequência Dentro de um Bloco ($m = 128$)	Rejeita	Rejeita
Teste das Corridas	Rejeita	Rejeita
Teste das Corridas em Blocos	Rejeita	Rejeita
Teste Serial	Rejeita	Aceita