Pós-Graduação em Ciência da Computação

“UMA APLICAÇÃO DA LINGUAGEM JAVA À COMPUTAÇÃO CIENTÍFICA.”

Por

Thiago Fabiano Silva Varjão

Dissertação de Mestrado

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

Recife

2013

Universidade Federal de Pernambuco
Centro de Informática
Pós-graduação em Ciência da Computação

Thiago Fabiano Silva Varjão

"UMA APLICAÇÃO DA LINGUAGEM JAVA À COMPUTAÇÃO CIENTÍFICA."

Trabalho apresentado ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientadora: *Marcilia Andrade Campos*

Recife
2013

Catalogação na fonte
Bibliotecária Joana D'Arc L. Salvador, CRB 4-572

Varjão, Thiago Fabiano Silva.
Uma aplicação da linguagem Java à computação cientifica / Thiago Fabiano Silva Varjão. – Recife: O Autor, 2013.
69 f.: fig., tab.

Orientadora: Marcilia Andrade Campos.
Dissertação (Mestrado) - Universidade Federal de Pernambuco. CIN. Ciência da Computação, 2013.
Inclui referências e apêndices.

1. Computação - Matemática. 2. Java (Linguagem de programação de computador). 3. Computação cientifica. I. Campos, Marcilia Andrade (orientador). II. Título.

004 (22. ed.) MEI 2014-81

Dissertação de Mestrado apresentada por **Thiago Fabiano Silva Varjão** à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título "**UMA APLICAÇÃO DA LINGUAGEM JAVA À COMPUTAÇÃO CIENTÍFICA**" orientada pela **Profa. Marcilia Andrade Campos** e aprovada pela Banca Examinadora formada pelos professores:

Prof. Sílvio de Barros Melo
Centro de Informática / UFPE

Profa. Maria Lencastre Pinheiro de Menezes Cruz
Escola Politécnica/UPE

Profa. Marcilia Andrade Campos
Centro de Informática / UFPE

Visto e permitida a impressão.
Recife, 30 de agosto de 2013

Profa. Edna Natividade da Silva Barros
Coordenadora da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.

Dedico esta dissertação a meus avós,
mestres na vida.

Agradecimentos

Agradeço a Deus, fonte de toda sabedoria, que com sua misericórdia infinita me sustenta e tem derramado graças abundantes em minha vida.

Meu pai e minha MÃE que desde cedo me incentivaram a ultrapassar as barreiras que o mundo me impõe e que nunca me negaram esforço algum.

Meu irmão, Filipe, que sempre acreditou em mim mais do que eu mesmo. Negão esse é seu, e você merece muito mais.

Minha irmã, Fabiana, pelo apoio mesmo distante se faz presente.

Minha sogra, Dona Conceição, pela força e exemplos de dedicação.

Meus cunhados e cunhadas que, a seu modo, sempre contribuiram para essa caminhada.

Professora Marcilia, cuja dedicação e empenho tornaram possível a conclusão dessa árdua jornada.

Minha esposa, Juliana companheira de todos os momentos, suporte nas diculdades e alegria constante, que com seu empurrãozinho de cada dia me trouxe até aqui. Deus sabe o que passamos por isso, e que você nunca me deixou parar.

Arthur e Victor, por quem me levanto a cada dia.

A todos, meu sincero muito obrigado, pois sozinho há muito teria desistido.

"...porque qualquer homem,
mesmo perfeito, entre os homens,
não será nada, se lhe falta a
Sabedoria que vem de vós."

—SAB 9,6

Resumo

No mundo do desenvolvimento de software, Java representa um marco. Concebida na década de 90, alcançou enorme popularidade desde o início de sua utilização. Apresentada inicialmente como uma linguagem que trabalhava em sites para internet, o que não era possível na época, seu amadurecimento levou a criação de muitas implementações, permitindo-lhe, hoje, ser encontrada, além das páginas da internet, em desktops, celulares e diversos outros dispositivos, provendo grande segurança, sendo utilizada em grandes bancos e empresas que necessitam de estabilidade e portabilidade, para trafegar grande quantidade de dados.

Apesar de toda influência da linguagem Java, baseada em seu poder e aplicabilidade, ela não costuma ser usada na computação científica por ter os típicos problemas computacionais acentuados pela forte tipagem de seus tipos primitivos, comprometendo sua aplicação na matemática computacional, mesmo em experimentos feitos, na implementação de bibliotecas intervalares que usam esses tipos primitivos.

Este trabalho apresenta a extensão de uma biblioteca, fundamentada na matemática intervalar e aritmética de exatidão máxima, na linguagem Java. Além das funções potência, raiz quadrada, exponencial, logarítmica e trigonométricas, a nova versão da biblioteca foi incrementada com cálculo de probabilidades para as variáveis aleatórias Uniforme, Exponencial e Pareto. Por fim, foi desenvolvida uma proposta de benchmark para comparação de tecnologias aplicadas à matemática intervalar no qual foram confrontados os desempenhos da extensão aqui proposta com uma biblioteca intervalar em Python.

Palavras-chave: Aritmética de Exatidão Máxima, Java, Matemática Intervalar, Python

Abstract

Java is a milestone in the world of software development. Conceived in the 90´s decade, it has since become hugely popular. Presented initially as a language to work on websites, not possible at the time, Java has evolved and led to the creation of a diversity of implementations that allows it to be found not only on websites but also on desktop computers, mobile phones and several other devices providing security and thus being used in banks and companies that need stability and portability in order to transfer large amounts of data.

In spite of all the influence of Java, due to it´s power and aplicability, it is not commonly used in scientific computing as it has some typical computation problems that are accentuated by the strong typing of its primitive types. This hinders its application in Computational Mathematics even in experiments performed in the implementation of interval libraries that use these kind of primitive types.

This paper presents the extension of a library which is based on interval mathematics and high acuracy arithmetic in Java language. Besides the power function, square root, exponential, logarithm and trigonometric functions, the new version of the library was increased with calculation of probabilities for the ramdon variables Uniform, Exponential and Pareto. Finally, we developed a proposal of a benchmark for comparing technologies applied to interval mathematics in which the performance of the extension proposed in this study was confronted with an interval library in Python.

Keywords: High accuracy arithmetic, Java, Interval Mathematics, Python

Lista de Figuras

Lista de Códigos Fonte

Sumário

1 Introdução

O avanço tecnológico da humanidade e a consequente introdução de computadores, em suas mais diversas variedades, no cotidiano da sociedade, tornaram-nos presentes atualmente desde as atividades mais corriqueiras até em experimentos científicos de grande magnitude. Esse alvorecer computacional trouxe consigo todo um universo de praticidades, mas, apesar de todo avanço, limitações permanecem dificultando a utilização dos computadores em algumas áreas, dentre elas pode-se mencionar o cálculo de probabilidades.

Os computadores representam um número em ponto flutuante com uma quantidade constante de bits. Porém, essa representação é apenas um subconjunto finito dos números racionais. À medida que operações aritméticas são realizadas o erro inerente a esta falha na representação dos números reais é propagado, visto que propriedades algébricas reais são perdidas [18].

As consequências dessas limitações dos sistemas computacionais são erros críticos em cálculos financeiros causando prejuízos, métricas científicas imprecisas comprometendo sua credibilidade ou até desastres como nos casos do sistema antimíssil Patriot, que ao não interceptar um míssil Iraquiano Scud, causou a morte de 28 soldados americanos e deixando cerca de 100 outras pessoas feridas e a plataforma de petróleo Sleipner A que afundou também em 1991 por conta de um erro no cálculo da espessura das paredes que foram insuficientes para suportar o peso da plataforma [3].

O uso de computadores na realização de cálculos proporciona inúmeras vantagens, porém, também podem levar a resultados totalmente errados com aparência de serem corretos, ou seja, um procedimento correto mas com o resultado perdendo seu significado devido à inexatidão da representação numérica e de arredondamentos aplicados nas avaliações das operações e expressões aritméticas em ponto flutuante [18]. No exemplo abaixo, são feitas 10 iterações que adicionam a razão de $0,1$ a variável d.

Exemplo 1. Erro em adição com double

```
1  private static void main(String args[]) {
2     double d = 0.0;
3     for (int i = 0 ; i < 10; i++) {
4        d += 0.1;
5     }
6     System.out.println("Resultado = " + d);
7  }
```

Saída: `Resultado = 0.9999999999999999`

Ao analisar o resultado da execução do trecho de código acima, escrito em linguagem Java[16], o resultado esperado deveria ser 1.0, mas o valor obtido é 0.9999999999999999.

Uma solução para os problemas numéricos decorrentes da representação digital dos números reais como propagação de erros, arredondamento, truncamento e mesmo a falta de precisão do dado em si é a matemática intervalar, que fornece uma ferramenta de estimativa e controle automático dos erros numéricos, aproximando um número x por um intervalo X, tendo números de máquina como limites inferior e superior [8].

Por exemplo, considere a representação de $1/3$ com quatro dígitos significativos. O arredondamento para o mais proximo resulta em 0.3333, o qual pode ser substitu do pelo intervalo $[0.3333, 0.3334]$. Aparentemente esse valor intervalar é mais impreciso que o valor verdadeiro, porém é mais confiável, visto que revela o nível de incerteza presente, e mostra que 0.3333 e uma subestimação do valor exato [8].

No mundo do desenvolvimento de software, Java [16] representa um marco. Concebida na década de 90, alcançou enorme popularidade desde o início de sua utilização. Apresentada inicialmente como uma linguagem que trabalhava em sites para internet, algo inédito na época, seu amadurecimento levou à criação de muitas implementações, permitindo-lhe, hoje, ser encontrada, além das páginas da internet, em desktops, celulares e diversos outros dispositivos, provendo grande segurança, sendo utilizada em grandes bancos e empresas que necessitam de estabilidade e portabilidade, para trafegar grande quantidade de dados [10].

Apesar de toda influência da linguagem Java, baseada em seu poder e aplicabilidade, ela não costuma ser usada na computação científica por ter os típicos problemas computacionais acentuados pela forte tipagem de seus tipos primitivos, comprometendo sua aplicação na matemática computacional, por dificultar, por exemplo, a aplicação de algoritmos de solução semelhantes aos utilizados em outras plataformas como MatLab[19] e Python[24]. Os problemas ficam evidentes, mesmo em experimentos feitos na implementação de bibliotecas intervalares que usam esses tipos primitivos [18].

1.1 Revisão Bibliográfica

Diversos trabalhos têm sido desenvolvidos no âmbito acadêmico no decorrer dos anos, aplicando a matemática intervalar na computação científica. Em vários pode-se identificar a busca pela solução do problema de representação numérica dos racionais, apresentando propostas e suas respectivas aplicações. A seguir é feita uma sucinta apresentação de trabalhos relacionados à matemática intervalar na computação científica.

Em [8], é proposta uma metodologia para estender a probabilidade real de tal forma a possibilitar seu cálculo automático através do uso de intervalos, limitando os erros computacionais e mantendo uma semântica consistente entre a probabilidade e os resultados intervalares obtidos. Propondo uma probabilidade intervalar em substituição à probabilidade real ou clássica.

Um ambiente de alto desempenho, do tipo cluster de computadores é apresentado em [14], através da utilização da biblioteca C-XSC e validado por testes básicos abordando o cálculo do produto escalar, a multiplicação entre matrizes, a implementação de solvers intervalares para matrizes densas e bandas e a implementação de alguns métodos numéricos para a resolução de sistemas de equações lineares com a característica da alta exatidão.

Hickley et al [13] apresenta uma biblioteca intervalar da linguagem Java que usa os tipos primitivos que implementa operadores aritméticos e as funções elementares.

Grigolleti et al[11] apresenta um módulo para matemática intervalar desenvolvido em Python [24], que disponibiliza funcionalidades de operações aritméticas intervalares.

Leite [18] desenvolveu uma biblioteca em Java para representar um novo sistema numérico que utiliza a matemática intervalar e a aritmética de exatidão máxima através do processamento de Strings.

No trabalho de [28] a biblioteca Intlab[26] do MatLab[19] é usada para implementar o método de Simpson Intervalar definido por Caprani [6] para calcular a probabilidade de uma variável aleatória, com distribuições Exponencial, Normal e Uniforme.

Funções matemáticas específicas e cálculo de probabilidades intervalares para variáveis aleatórias discretas e contínuas são implementadas por [34] em uma extensão da biblioteca Intpy [5].

Mendonça [20] propõe métodos, implementados no Matlab [19] utilizando a biblioteca Intlab [26], para a obtenção de intervalos que encapsulam valores de métricas de confiabilidade.

Nesta breve análise de trabalhos relacionados, é possível identificar similaridades entre

os objetivos almejados, dentre os quais destacam-se a busca pela aplicação computacional em problemas matemáticos e o uso da matemática intervalar na solução de problemas em comum.

É possivel identificar ainda, a tímida utilização da linguagem Java perante o amadurecimento de outras tecnologias. Nascendo daí, a motivação do estudo desenvolvido neste trabalho dissertativo.

1.2 Objetivos

A proposta deste trabalho é utilizar a Biblioteca Intervalar de String Numéricos BISN desenvolvida por Leite [18], que fornece um novo sistema numérico utilizando matemática intervalar e aritmética de exatidão máxima, baseando-se no processamento de strings, estendendo suas funções matemáticas especificas ainda não contempladas e a implementação de cálculos de probabilidades intervalares [8, 20, 28, 34], para variáveis aleatórias contínuas, culminando em uma aplicação válida da linguagem Java na computação científica.

A princípio, a Biblioteca Intervalar de Strings Numéricos foi estendida com a implementação das funções de potência, raiz quadrada, logaritmo, exponencial e trigonométricas, propostas no trabalho dissertativo desenvolvido por Leite [18].

Em seguida, foram implementados cálculos das distribuições de probabilidade para as variáveis aleatórias contínuas Exponencial, Uniforme e Pareto utilizando a aritmética intervalar [21, 22, 23] e a aritmética de exatidão máxima [17]. Foram ainda submetidos aos cálculos das probabilidades desenvolvidas os parâmetros de entrada dos problemas propostos em [28, 34] que foram desenvolvidos utilizando outras plataformas Matlab [19, 26] e Python [24], implicando na validação dos resultados obtidos com Java.

Por fim, foi desenvolvido um benchmark [29] entre a aplicação da Biblioteca Intervalar de Strings Numéricos em Java e a aplicação desenvolvida em Python por [34].

1.3 Estrutura do trabalho

Esta dissertação está organizada em quatro capítulos sumarizados a seguir.

No Capítulo 2, são apresentados os fundamentos teóricos necessários para o desenvolvimento das propostas deste trabalho. Os intervalos com aritmética de exatidão máxima são apresentados como solução para os problemas causados pela representação numérica na computação, fundamentando-se em seus conceitos. Ainda nesse capítulo

é exposto um breve histórico da linguagem Java com suas principais características e ambientação no mercado de desenvolvimento de software.

O Capítulo 3 traz uma aplicação da computação científica em Java, desenvolvida com a extensão da biblioteca criada por [18]. Foram implementadas as funções propostas como trabalhos futuros por [18] e ainda o cálculo de intervalos encapsuladores de probabilidade para as distribuições uniforme, exponencial e pareto.

O Capítulo 4 propõe um benchmark comparativo de desempenho entre algorítmos de distribuição de probabilidade intervalar em Java na biblioteca [18] e em Python na biblioteca Intpy [5].

No Capítulo 5 é feita uma avaliação da contribuição desta dissertação para a comunidade científica, além de propostas de trabalhos futuros.

2 Fundamentos

Este capítulo apresenta a fundamentação teórica aplicada no desenvolvimento da proposta do trabalho, aduzindo os intervalos com aritmética de exatidão máxima como solução para os problemas oriundos da representação numérica na computação, além de uma explanação de atributos nos quais são baseados a escolha da linguagem Java para a proposta.

2.1 Problemas da computação científica

A limitação de armazenamento das máquinas digitais dificulta a representação de números reais causando erros numéricos na computação científica. Como solução para os problemas de representação dos números reais com controle do erro numérico usa-se a Matemática Intervalar [21, 22, 23], na qual os números são representados por uma faixa de valores intervalares ao invés de um valor pontual.

O uso de métodos da matemática intervalar traz consigo a garantia que a resposta pertence ao intervalo resultante. Segundo Campos [8], os intervalos devem ser confiáveis no sentido que incluem a solução exata do problema original com uma amplitude tão pequena quanto possível.

Conforme Cláudio [7], os resultados obtidos na computação numérica podem conter erros provenientes de três fontes distintas, sobre os quais o uso da aritmética intervalar pode prover uma análise rigorosa e completa. A primeira, e por não ser possível torná-la menor, a mais séria dessas fontes, é a *propagação de erro nos dados iniciais*. Esse erro ocorre quando os dados iniciais do processamento são incertos, causados, por exemplo por uma simplificação de um modelo matemático de determinado sistema ou em medidas de tempo, temperatura, distância, etc., coletados com instrumentos de precisão limitada. As incertezas dos dados de entrada são acumuladas nas respostas e precisam ser assistidas por

uma análise profunda por meio de simulações e por meio da experiência do pesquisador.

As outras duas fontes de erros são *arredondamento* e *truncamento*, ambos gerados pela adequação dos resultados das operações aritméticas a um número finito de dígitos. O uso da matemática intervalar visa fornecer um esquema computacional que torne essas duas últimas espécies de erros tão pequenas quanto possível. De acordo com Acioly [1], um intervalo resultante de uma operação carrega consigo a garantia de sua incerteza, sendo apenas uma estimativa do erro que pode estar presente.

A matemática intervalar [21, 22] fornece uma ferramenta para estimar e controlar erros numéricos automaticamente. Ao invés de aproximar um número real x por um número de máquina, x é aproximado por um intervalo X, tendo números de máquina como limites inferior e superior. Podendo a amplitude do intervalo ser usada como medida de qualidade da aproximação [8]. Contudo, os cálculos, tem que ser efetuados sobre os intervalos ao invés de números reais.

Por exemplo, a representação de $1/3$ com quatro dígitos significativos. O arredondamento para o mais próximo resulta em 0.3333, o qual pode ser substituído pelo intervalo $[0.3333, 0.3334]$, usando arredondamentos direcionados. Apesar de aparentar uma imprecisão em relação ao valor verdadeiro, o valor intervalar é mais confiável, visto que 0.3333 é uma subestimação do valor exato [17].

Segundo Campos [8], a análise intervalar pode ser usada para produzir computação numérica autovalidável, também conhecida como análise automática do erro, através da computação de um intervalo que tem a garantia de conter o valor teoricamente correto mas desconhecido.

2.2 Intervalos

Um *intervalo*, no contexto da matemática intervalar, é um subconjunto dos reais da seguinte forma:

$$[x_1, x_2] = \{x \in \mathbf{R} \mid x_1 \leq x \leq x_2\}. \tag{2.1}$$

A finalidade principal de equação (2.1) é construir uma aproximação dos números reais nele contidos, conforme definição de [21, 22, 23]. A denotação dada para o conjunto de todos os intervalos reais é **IR**. Portanto **IR** é o espaço de todos os intervalos de números reais da forma $[x_1, x_2]$.

O intervalo $[x_1, x_2]$ representa qualquer número real contido entre os números x_1 e x_2.

Por exemplo, π pode ser representado como um intervalo:

$$\pi = [314 \cdot 10^{-2},\ 315 \cdot 10^{-2}], \tag{2.2}$$

onde

$$314 \cdot 10^{-2} \ \leq \ \pi \ \leq \ 315 \cdot 10^{-2}. \tag{2.3}$$

Moore é conhecido como precussor da aritmética intervalar em virtude de seu trabalho monográfico [21] publicado em 1966. Contudo os primeiros estudos relacionados a intervalos datam de 1914, quando Norbert Wiener [36] utilizou intervalos em medições de distâncias e tempo.

As operações aritméticas [1] com intervalos são definidas do seguinte modo. Dados X, $Y \in \mathbf{IR}$ com $X = [x_1, x_2]$ e $Y = [y_1, y_2]$, tem-se que:

$X + Y = [x_1 + y_1,\ x_2 + y_2]$.
$X - Y = [x_1 - y_2,\ x_2 - y_1]$.
$X \cdot Y = [min\{x_1 \cdot y_1, x_1 \cdot y_2, x_2 \cdot y_1, x_2 \cdot y_2\},\ max\{x_1 \cdot y_1, x_1 \cdot y_2, x_2 \cdot y_1, x_1 \cdot y_2\}]$.
$X/Y = [x_1,\ x_2] \cdot [1/y_1,\ 1/y_2]$ se $0 \ \notin \ [y_1,\ y_2]$.

Deste modo, toda operação aritmética com intervalos é, por natureza, mais dispendiosa de processamento, consequentemente de tempo. Este custo de processamento é ainda intensificado pelo tratamento de strings feito na BISN [18].

2.3 Aritmética de Exatidão Máxima

A representação numérica nos computadores é finita, devido às limitações físicas, uma vez que o armazenamento e processamento das informações é realizado com um número fixo de dígitos binários (bits). Esta restrição não é problema para a representação de letras de um alfabeto, números inteiros, notas musicais ou mesmo partes de um grafo. Contudo, a situação não é a mesma quando na representação dos números reais, visto que este conjunto de números é um corpo ordenado completo [8], por isso é necessário substituí-lo por um conjunto que os simule, nesse caso os números de ponto-flutuante [7] **F**.

O sistema de ponto flutuante é $\mathbf{F}(b, l, e_1, e_2)$, onde:

- $b =$ base

[1] Neste trabalho a ênfase é apenas nas operações aritméticas

- $l =$ quantidade de dígitos do significando
- $e_1 =$ menor expoente
- $e_2 =$ maior expoente

O problema do uso dos pontos-flutuantes é que, diferentemente dos reais, eles não têm propriedades algébricas que garantam os resultados calculados, podendo, causar *overflow* em uma operação aritmética quando, por exemplo, se somam números grandes de **F**.

Kulish e Miranker [17] propõem os intervalos de pontos-flutuantes como forma de representação dos números reais em máquinas digitais através da aritmética de exatidão máxima, onde um número real x é representado pelo menor intervalo cujos limites pertencem a **F**. Dado $X = [x_1, x_2]$ que encapsula x tem-se que:

- $x_1 \in \mathbf{F}$.
- $x_2 \in \mathbf{F}$.
- $x_1 \leq x \leq x_2$.

A aritmética de exatidão máxima propõe uma forma de operar números reais representados por números de máquina preservando uma estrutura algébrica chamada de *anelóide* ou *vetoide*, através de *semiformismo* [17].

2.4 Java

A linguagem Java [16] representa um marco no desenvolvimento de software. Apresentada inicialmente como uma linguagem para sites, seu amadurecimento levou à criação de muitas implementações, permitindo-lhe, hoje, ser encontrada, além das páginas da internet, em desktops, celulares e diversos outros dispositivos. A linguagem Java provê extrema segurança, sendo utilizada em grandes bancos e empresas que necessitam de estabilidade e portabilidade, para trafegar grande quantidade de dados, entre as quais podem ser citadas "NASA, IBM, ESPN entre outras como exemplos da confiabilidade que a linguagem Java demonstra em seus utilizadores" [10].

2.4.1 História

Em 1980, a palavra de ordem era Codificação, e como diz Anselmo [2] "a maior preocupação era quanto a escrever programas e, dos programas, escrever sistemas para

computadores. Quanto mais, melhor." As linguagens que estavam no auge eram Fortran, Algol, Cobol e Pascal, linguagens de programação estruturadas, cujos programas são gerados a partir de blocos elementares de código ligados por mecanismos básicos de seqüência, seleção e iteração [25]. Com o surgimento da Structure Query Language (SQL – Linguagem de Consulta Estruturada) em 85, dados passam a ser o objeto da atenção do ambiente de desenvolvimento de software [2].

Em 1990, a palavra da vez era "Produtividade", auge do modelo Client/Server, ferramentas Visual Basic, Delphi e o conceito de Rapid Application Development (RAD) dominavam o mercado. Os softwares deviam ser produzidos o mais rápido e do melhor modo possível. Por fim em 1995, o clamor foi pela "Performance", buscando-se "Computação Distribuída" e "Escalabilidade", conceitos indispensáveis nas aplicações, que passavam a ter a computação descentralizada. Com o crescimento da internet, nasce então a plataforma e linguagem Java.

Iniciado como um projeto de melhoria da linguagem C++, na tentativa de criar uma plataforma de controle universal para eletrodomésticos e computadores em uma residência, recebendo o nome de Java inspirado no nome da cidade de origem de um tipo de café importado que a equipe responsável tomara durante visita em uma cafeteria local.

A plataforma Java foi apresentada como uma linguagem que seria voltada para aplicações para a Internet em 23 de maio de 1995 no "SunWorld", evento que reunia pessoas do mundo inteiro, por John Cage, diretor do Escritório de Ciência da Sun Microsystems, e Marc Andreesen, co-fundador e vice-presidente da Netscape [2].

2.4.2 Características

A linguagem Java possui características fundamentais que norteiam seu funcionamento e expõem a essência de seus conceitos. A seguir serão abordadas estas características a partir de 11 palavras-chave citadas no "Manifesto Java" apresentado por Horstmann e Cornell [15]:

- Simples: A linguagem Java sintetiza-se em apenas seis comandos, sendo dois de decisão e três de repetição, abstraindo assim, o desenvolvedor do cuidado com ponteiros, estruturas, sobrecargas, uniões, dentre outras, além de gerar os arquivos de aplicação pequenos, permitindo-lhes rodar em maquinas de grande ou pequeno porte.

- Orientada a Objetos: A linguagem Java herdou do C++ as funções orientadas a objetos, suportando a aplicação de conceitos como abstração, herança, polimorfismo,

encapsulamento, classes e reuso.

- Distribuída: Java provê facilidade para trabalhar com objetos distribuídos, dando-lhes acessibilidade semelhante à dos sistema de arquivos locais, através de uma biblioteca de classes que trabalham com protocolos de rede.

- Robusta: A linguagem Java permite a criação de aplicações extremamente confiáveis por ser capaz de verificar possíveis problemas em tempo de execução. "[...]Java possui um modelo de ponteiros que elimina a possibilidade de sobrescrever a memória e corromper dados."

- Segura: A linguagem Java tem a segurança como um de seus focos desde sua concepção, por ter sido projetada para desenvolver aplicações voltadas para ambientes de rede e/ou distribuídos.

- Neutra em Relação à Arquitetura: O compilador Java gera instruções bytecodes que são interpretados pela máquina virtual Java, o que permite que os aplicativos nela desenvolvidos sejam executados em diversos processadores mantendo-se independente de arquitetura.

- Portável: As aplicações desenvolvidas em Java são independentes da plataforma, ou seja, é possível uma mesma aplicação rodando no Windows, Macintosh e no Unix pois tem uma interface de portabilidade definida e concisa, onde seus tipos primitivos de dados e seus comportamentos aritméticos são especificados.

- Interpretada: "O interpretador Java pode executar bytecodes Java diretamente em qualquer máquina para qual o interpretador tenha sido portado."

- Alto desempenho: Apesar da interpretação dos bytecodes ter um desempenho aquém dos programas compilados as aplicações desenvolvidas em Java não ficam restritas a essas limitações, pois pode ser realizada uma compilação em tempo de execução (JIT – Just In Time), onde os bytescodes são compilados em código nativo da plataforma.

- Multithreaded: Java facilita significativamente o uso do multithreading, ou seja, o processamento simultâneo de tarefas, além de beneficiar-se de sistemas multiprocessadores desde que o sistema operacional o suporte.

- Dinâmica: A linguagem Java permite ao programador adicionar código em um programa em execução, pois foi projetada para adaptar-se a ambientes em constante evolução.

Essas características fizeram da plataforma Java um verdadeiro fenômeno do universo do desenvolvimento de software. No seu surgimento ela era uma linguagem relativamente simples e direta, entretanto com sua constante evolução, foram-se acrescentando diversos recursos que a tornaram mais complexa, principalmente depois da JEE – Java Platform Enterprise Edition, e mudaram o conceito que os desenvolvedores lhe atribuiam, passando a ser considerada grande, pesada e de difícil aprendizagem [12].

Todavia, como bem advertem Horstmann e Cornell [15], seria impossível encontrar facilidade no aprendizado de uma linguagem de programação tão poderosa quanto Java. Porém, conhecimento básico de seus milhares de recursos é fundamental para que se possa usufruir dessa capacidade em um âmbito realista.

2.4.3 Mercado de desenvolvimento

A plataforma Java foi adotada mais rapidamente do que qualquer outra na história da computação. A expansão da internet e crescente uso de dispositivos móveis a tornaram ainda mais popular. A TIOBE Company <http://www.tiobe.com/>, empresa especializada em qualidade de software, mantém um índice mensal das linguagens de programação desde 2001, no qual a liguagem Java nunca esteve abaixo do segundo lugar, permanecendo na grande maioria desse período no topo do ranking.

O índice TIOBE não tem a pretensão de nomear quais as melhores linguagens de programação, mas sim apresentar indicativos da popularidade das linguagens de programação do mercado. Para isso são usados no cálculo do índice a quantidade de profissionais certificados, cursos e fornecedores das linguagens, além de dados de buscas feitas em sites populares como Google, Bing, Yahoo!, Wikipedia, Amazon, YouTube e Baidu [30].

Apesar de toda influência da linguagem Java, baseada em seu poder e aplicabilidade, ela não costuma ser usada na computação científica por ter os típicos problemas computacionais acentuados pela forte tipagem de seus tipos primitivos, comprometendo sua aplicação na matemática computacional, mesmo em experimentos feitos, na implementação de bibliotecas intervalares que usam esses tipos primitivos [18].

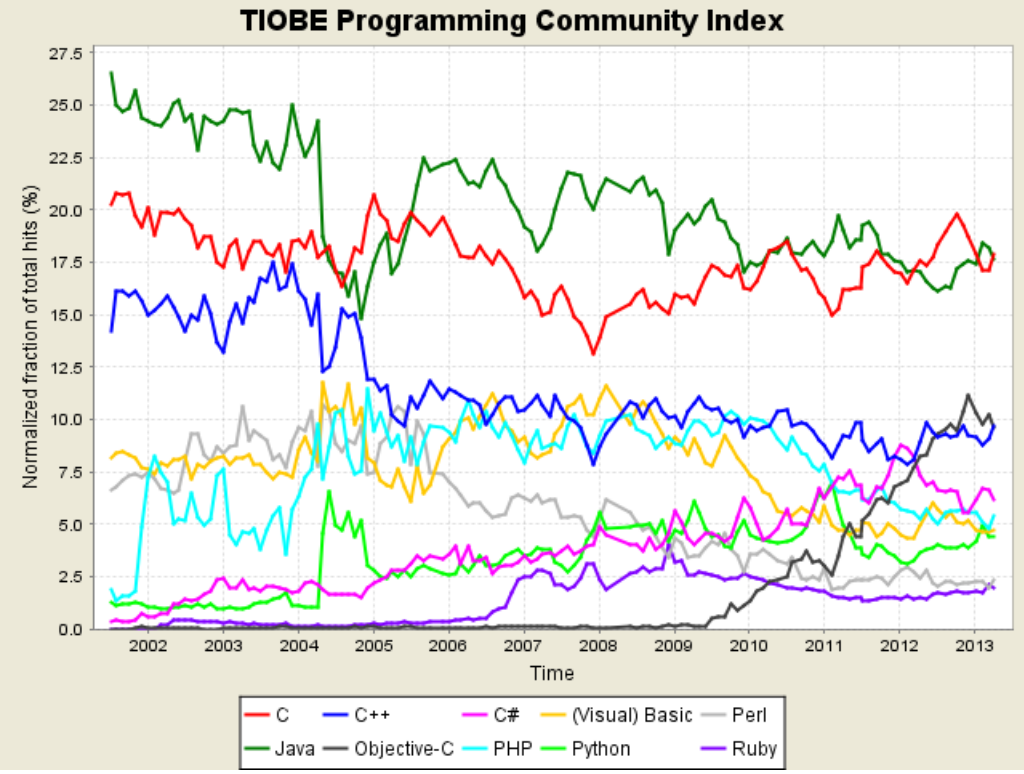

Figura 2.1 Indice TIOBE

2.4.4 Biblioteca Intervalar com Strings Numéricos

Como proposta para resolver os problemas causados pelos erros na representação dos números reais em Java, Leite [18] apresentou uma mudança de paradigma, representando tais números com Strings, ideia essa baseada no conceito observado nos cálculos efetuados manualmente no período escolar, onde, em prol do aprendizado, os alunos efetuam os cálculos sem a ajuda de instrumentos automáticos, tratando-os como uma cadeia de caracteres. Apesar dessa abordagem, por limitações físicas das máquinas, esta representação ainda tem um número finito de caracteres, por isso fez-se necessário criar um sistema intervalar para representação genérica.

A proposta da BISN é representar os números reais com Strings [16], contornando as limitações que levam a utilização de truncamentos e arredondamentos realizadas nas operações aritméticas [18]. A linguagem Java [16] provê métodos de manipulação de Strings como obtenção de um caracteres individuais, comparações entre instancias de Strings, extração de subconjuntos, criação de cópias, concatenção entre outras.

A representação numérica base da BISN é com instâncias da classe `Number`, que possui 5 atributos:

- mantissa: É do tipo String e é o conjunto de algarismos da base do número.
- sinal: Representa o sinal do número e pode tomar o valor de uma das três constantes: `SINAL_NEGATIVO`, `SINAL_ZERO` e `SINAL_POSITIVO`. Apenas o número zero possui o valor `SINAL_ZERO`.
- expoente: Determina a ordem de grandeza de um número. Quantas casas decimais depois da virgula (se o sinal do expoente for negativo) ou qual potência de 10 que multiplica a mantissa. Representa os algarismos do expoente.
- sinalExpoente: Define o sinal do expoente do número pode assumir o valor de uma das três constantes `SINAL_NEGATIVO`, `SINAL_ZERO` e `SINAL_POSITIVO`.
- denominador: Para permitir a representação de um número no formato fracionário o atributo denominador foi criado. Ele é do tipo Number podendo também ter um denominador.

Com a utilização deste modelo obtém-se uma precisão máxima de $2^{31} - 1$ casas decimais na base 10, que é o tamanho máximo que o tipo inteiro em Java pode assumir, pois em Java laços e índices de arrays são indexados por este tipo de dado (`int`). Sendo este também o limite do tamanho do expoente, o que possibilita um módulo máximo do expoente igual a $10^{2147483647} - 1$. Números que fornecem uma margem satisfatória para execução de cálculos sem arredondamentos ou truncamentos [18].

O intervalo por sua vez é representado pela classe `Intervalo` que possui um par de `Number's` referentes aos intervalos inferior e superior.

A Biblioteca Intervalar de Strings Numéricos, BISN, desenvolvida por Leite [18] contém operações aritméticas, operações entre conjuntos além de outras funções como distância entre intervalos, diâmetro, ponto médio e valor absoluto de um intervalo.

A evolução desta biblioteca supõe a aproximação da linguagem Java das características das linguagens XSC [35], levando a experiência dos resultados intervalares da computação científica à possibilidade de utilização no cotidiano dos usuários. Permitindo, por exemplo, o desenvolvimento de aplicações para dispositivos móveis que, entre outros, representam a tendência da evolução da computação na direção da computação pervasiva [4].

3

Ampliando a Biblioteca Intervalar de String Numéricos

Este capítulo apresenta a contribuição deste trabalho, (i) detalhando as ampliações implementadas na Biblioteca Intervalar de Strings Numéricos (BISN) desenvolvida por [18], (ii) incluindo as funções de Exponencial, Logarítmica, Potência, Raiz Quadrada, Trigonométrias e (iii) distribuições de probabilidade implementadas com base em [28] e [34].

3.1 Extensão da classe Intervalo

Para viabilizar o desenvolvimento das contribuições realizadas nesse trabalho, fez-se necessário a implementação de algumas melhorias nas classes que integram a estrutura da BISN. Para tal, basicamente, a classe `Intervalo` que representa as entidades matemática homônimas foi estendida, dando origem à classe `NovoIntervalo` conforme apresentado no diagrama de classes da nova arquitetura da Biblioteca de Intervalar na figura a seguir.

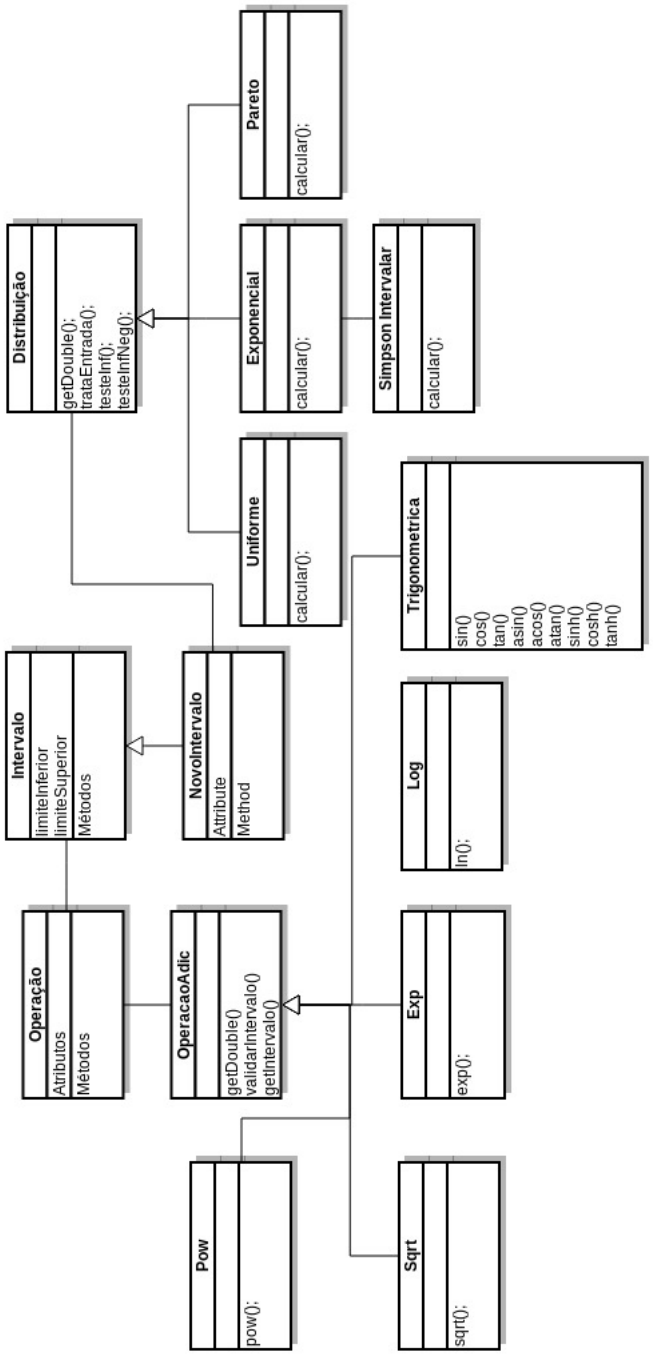

Figura 3.1 Arquitetura da extensão da Biblioteca Intervalar

A extensão da classe `Intervalo` contém métodos construtores que resolvem intervalos degenerados com entradas distintas e um método de verificação da relação de pertinência entre um determinado número real e intervalos.

3.1.1 Construtor de Intervalos com limites do tipo Double

Durante a implementação das distribuições de probabilidade Uniforme, Exponencial e Pareto, surgiu a necessidade de um método construtor de intervalos para entrada do tipo `Double`, em função dos parâmetros definidos por [28] para os métodos propostos.

Código Fonte 3.1 Construtor do Intervalo com parâmetro Double.

```
1 static public Intervalo novoIntervalo(Double limInf, Double limSup) {
2         Number inf = Number.createNumber(String.valueOf(limInf));
3         Number sup = Number.createNumber(String.valueOf(limSup));
4         return new Intervalo(inf, sup);
5 }
```

3.1.2 Intervalos degenerados

Um intervalo degenerado é aquele cujos limites superior e inferior são iguais: $X = [x, x]$.

A classe `Intervalo` original da Biblioteca Intervalar de Strings Numéricos, quando instanciada com um parâmetro entrada único, gera um intervalo degenerado. Contudo, este tipo de intervalo nem sempre é interessante para a aplicação de cálculos da matemática intervalar, visto que o mesmo pode ser visto como um número real [27, 31].

A classe `NovoIntervalo`, extensão feita do `Intervalo` da Biblioteca Intervalar de Strings Numéricos, possui métodos construtores que criam intervalos não-degenerados a partir de uma entrada numérica simples, por meio de arredondamentos direcionados sobre a última casa decimal da entrada, dando origem a limites inferior e superior distintos para o intervalo. Os métodos gerados com este objetivos são:

1. Construtor do `NovoIntervalo` com entrada Number

Código Fonte 3.2 Construtor do NovoIntervalo com parâmetro Number.

```
1  public NovoIntervalo(Number number) {
2                  super();
3                  int exp = Integer.parseInt(number.getExpoente());
4                  Number arred = new Number();
5                  arred.setMantissa("1");
6                  arred.setExpoente(String.valueOf(exp));
7                  arred.setSinalExpoente(-1);
8                  arred.setSinal(-1);
9                  Number limInf = Number.adicionar(number, arred);
10                 Number limSup = Number.subtrair(number, arred);
11                 this.setLimiteInferior(limInf);
12                 this.setLimiteSuperior(limSup);
13         }
```

Exemplo 2.

```
1          public class Exemplos {
2                  public static void main(String[] args) {
3                          Number n = Number.createNumber("3.1415");
4                          Intervalo resultado = new NovoIntervalo(n);
5                          System.out.println("Intervalo = " + resultado);
6                  }
7          }
```

Saída: `Intervalo = [31414E-4, 31416E-4]` [1]

2. Construtor do `NovoIntervalo` com entrada String

Código Fonte 3.3 Construtor do NovoIntervalo com parâmetro String.

```
1          public NovoIntervalo(String number) {
2                  super();
3                  Number n = Number.createNumber(number);
4                  int exp = Integer.parseInt(n.getExpoente());
5                  Number arred = new Number();
6                  arred.setMantissa("1");
7                  arred.setExpoente(String.valueOf(exp));
8                  arred.setSinalExpoente(-1);
9                  arred.setSinal(-1);
10                 Number limInf = Number.adicionar(n, arred);
11                 Number limSup = Number.subtrair(n, arred);
12                 this.setLimiteInferior(limInf);
13                 this.setLimiteSuperior(limSup);
14         }
```

[1] $[31414E-4, 31416E-4] = [3.1414, 3.1416]$

Exemplo 3.

```
1  public class Exemplos {
2          public static void main(String[] args) {
3                  String n = "3.1415";
4                  Intervalo resultado = new NovoIntervalo(n);
5                  System.out.println("Intervalo = " + resultado);
6          }
7  }
```

Saída: `Intervalo = [31414E-4, 31416E-4]`

3. Construtor do `NovoIntervalo` com entrada Double

Código Fonte 3.4 Construtor do NovoIntervalo com parâmetro Double.

```
1  public NovoIntervalo(Double number) {
2                  super();
3                  Number n = Number.createNumber(
4                                  String.valueOf(number));
5                  int exp = Integer.parseInt(n.getExpoente());
6                  Number arred = new Number();
7                  arred.setMantissa("1");
8                  arred.setExpoente(String.valueOf(exp));
9                  arred.setSinalExpoente(-1);
10                 arred.setSinal(-1);
11                 Number limInf = Number.adicionar(n, arred);
12                 Number limSup = Number.subtrair(n, arred);
13                 this.setLimiteInferior(limInf);
14                 this.setLimiteSuperior(limSup);
15         }
```

Exemplo 4.

```
1  public class Exemplos {
2          public static void main(String[] args) {
3                  Double n = 3.1415;
4                  Intervalo resultado = new NovoIntervalo(n);
5                  System.out.println("Intervalo = " + resultado);
6          }
7  }
```

Saída: `Intervalo = [31414E-4, 31416E-4]`

3.1.3 Valor contido no intervalo

Uma função fundamental no uso de operações com intervalos é a verificação se um intervalo contém determinado número, isto é, dado $x \in \mathbf{R}$ e $X = [x_1, x_2]$, $x \in X$? Este método, recebe como parâmetro um `Number` e retorna um `boolean` que indica a relação entre o dado número e o intervalo, por meio de comparações originais da Biblioteca Intervalar de Strings Numéricos.

Código Fonte 3.5 Método de verificação da pertinência de x ao intervalo X.

```
1  public boolean contem(Number x) {
2          if (Number.comparar(getLimiteInferior(), x) <= 0
3                          && Number.comparar(getLimiteSuperior(), x) >= 0) {
4                  return true;
5          }
6          return false;
7  }
```

Exemplo 5. Verificar se $3.1415 \in [3.1414, 3.1416]$

```
1          public class Exemplos {
2                  public static void main(String[] args) {
3                          NovoIntervalo X = new NovoIntervalo("3.1414", "3.1416");
4                          Number x = Number.createNumber("3.1415");
5                          if (X.contem(x)) {
6                                  System.out.println("x esta contido no Intervalo X");
7                          } else {
8                                  System.out.println("x nao esta contido no Intervalo X");
9                          }
10                 }
11         }
```

Saída: `x esta contido no Intervalo X`

3.2 Funções Auxiliares

Além dos métodos apresentados de aperfeiçoamento do tipo `Intervalo`, foram implementadas funções auxiliares, decorrentes da refatoração de código, com o intuito de aperfeiçoar a manipulação da BISN.

3.2.1 Extração de um Double do tipo Number

A função `getDouble` foi implementada com o objetivo de extrair um valor `Double` de um dado `Number`, processo utilizado quando é necessário realizar alguma manipulação

com funções básicas da linguagem Java a partir dos tipos criados por [18] na Biblioteca Intervalar de String Numéricos (`Intervalo` e `Number`).

Código Fonte 3.6 Método de extração de Double de um Number.

```
1 private static Double getDouble(Number n) {
2         Double d = Double.parseDouble(n.getMantissa());
3         int sinalExp = n.getSinalExpoente();
4         d = d * Math.pow(10, (sinalExp * Double.parseDouble(n.getExpoente())));
5         d = d * n.getSinal();
6         return d;
7 }
```

3.2.2 Validação de Intervalo

A validação do intervalo é um método que verifica se, dado um intervalo $X = [x_1, x_2]$, o limite inferior (x_1) é menor ou igual ao limite superior (x_2). Como mencionado em [34], por exemplo, para a função seno intervalar, deve-se calcular $sin(x_1)$ e $sin(x_2)$. Se $MAX(sinX)$ for igual a $sin(x_1)$, ou seja, $sin(x_1) > sin(x_2)$, então $sin([x_1, x_2]) = [sin(x_2), sin(x_1)]$, caso contrário, $sin([x_1, x_2]) = [sin(x_1), sin(x_2)]$.

Código Fonte 3.7 Método de Validação de um Intervalo.

```
1 protected static Intervalo validateIntervalo(Intervalo in) {
2         if (Number.comparar(in.getLimiteSuperior(), in.getLimiteInferior()) >= 0) {
3                 return in;
4         }
5         Number temp = in.getLimiteInferior();
6         in.setLimiteInferior(in.getLimiteSuperior());
7         in.setLimiteSuperior(temp);
8         return in;
9 }
```

3.2.3 Obtenção de Intervalo

Para permitir uma generalização das funções implementadas, aumentando a aplicabilidade da BISN 2.0, foram estabelecidas entradas do tipo `Object` na maioria dos métodos desenvolvidos, e para ajudar no tratamento desta entrada foram implementados métodos de validação das mesmas, identificando quando se trata de instâncias de intervalos e, em caso contrário, gerando intervalos a partir da entrada dada.

Código Fonte 3.8 Método de tratamento de parâmetros de entrada.

```
protected static Intervalo getInterval(Object in) {
        if (in instanceof Intervalo) {
                return (Intervalo) in;
        } else {
                Intervalo out = new NovoIntervalo(String.valueOf(in));
                return out;
        }
}
```

3.3 Funções

Com base nas sugestões de trabalhos futuros de [18], foram implementadas as funções potência, raiz quadrada, logarítmica, exponencial além de algumas funções trigonométricas.

No desenvolvimento dessas funções foram utilizados recursos nativos da linguagem Java `java.lang.Math`. Com a aplicação das funções auxiliares, anteriormente apresentadas, valores foram extraídos dos recursos nativos de Java, cálculos foram realizados e intervalos foram gerados, com o devido tratamento. Para permitir a visualização da forma geométrica das funções e suas principais características foram gerados gráficos [37].

3.3.1 Potência

Uma potência é o resultado da multiplicação de um certo número por si mesmo n vezes, ou seja, é a representação de sucessivas multiplicações de um fator, repetido um determinado número de vezes.

A potência de um intervalo, tem seu resultado obtido a partir da aplicação do cálculo sobre seus limites, dando origem a um novo intervalo [21, 34].

$$X^n = [x_1,x_2]^n \begin{cases} [x_1,x_2]^n, & se \quad x_1 > 0, \\ [0, max\{x_1^n, x_2^n\}], & se\ 0 \in X\ e\ n \text{ é par}, \\ [x_1^n, x_2^n], & se\ 0 \in X\ e\ n \text{ é impar}, \\ [x_2^n, x_1^n], & se\ x_2 < 0\ e\ n \text{ é par}, \\ [x_1^n, x_2^n], & se\ x_2 < 0\ e\ n \text{ é impar}. \end{cases}$$

O método desenvolvido recebe entradas do tipo `Object`, garantindo abrangência na sua aplicação. O primeiro parâmetro é a base, neste caso um intervalo, e o segundo o expoente, que é um inteiro positivo.

Código Fonte 3.9 Método de Cálculo da Potência.

```
1  public static Intervalo pow(Object base, Object expoent)
2                          throws ExcecaoOperacaoInvalida {
3                  Intervalo res = getInterval(base);
4                  try {
5                          Integer b = Integer.parseInt(String.valueOf(expoent));
6                          Double inf = getDouble(res.getLimiteInferior());
7                          Double sup = getDouble(res.getLimiteSuperior());
8                          if ((inf < 0 && sup > 0) && b % 2 == 0) {
9                                  Double max = Math.pow(inf, b) > Math.pow(sup, b) ?
10                                                 Math.pow(inf, b) : Math.pow(sup, b);
11                                 res.setLimiteInferior("0");
12                                 res.setLimiteSuperior(String.valueOf(max));
13                         } else {
14                                 res.setLimiteInferior(String.valueOf(Math.pow(inf, b)));
15                                 res.setLimiteSuperior(String.valueOf(Math.pow(sup, b)));
16                         }
17                         res = validateIntervalo(res);
18                 } catch (Exception e) {
19                         throw new ExcecaoOperacaoInvalida();
20                 }
21                 return res;
22         }
```

Exemplo 6. Calcular $[0.25, 0.27]^4$

```
1          public class Exemplos {
2                  public static void main(String[] args) {
3                          Intervalo X = new Intervalo("0.25", "0.27");
4                          Intervalo resultado = Pow.pow(X, 4);
5                          System.out.println("pow(X,4) = " + resultado);
6                  }
7          }
```

Saída: `pow(X,4) = [390625E-8, 53144100000000001E-18]`

Exemplo 7. Calcular $[-0.27, 0.25]^4$

Saída: `pow(X,4) = [0, 53144100000000001E-18]`

Exemplo 8. Calcular $[-0.27, 0.25]^3$

Saída: `pow(X,3) = [-19683000000000006E-18, 15625E-6]`

Exemplo 9. Calcular $[-0.27, -0.25]^4$

Saída: `pow(X,4) = [390625E-8, 53144100000000001E-18]`

Exemplo 10. Calcular $[-0.27, -0.25]^3$

Saída: `pow(X,3) = [-19683000000000006E-18, -15625E-6]`

Se os parâmetros, base ou expoente, não satisfizerem à especificação dada, uma exceção é levantada.

Exemplo 11. Calcular $[0.25, 0.27]^{0.67}$

```
1       public class Exemplos {
2               public static void main(String[] args) {
3                       Intervalo X = new Intervalo("0.25", "0.27");
4                       Intervalo resultado = Pow.pow(X, 0.67);
5                       System.out.println("pow(X,4) = " + resultado);
6               }
7       }
```

Saída: `Operação Inválida`

3.3.2 Raiz Quadrada

O cálculo da raiz quadrada [23, 32, 33, 34], bem como o da potência, é feito com intervalos aplicando a operação sobre os limites inferior e superior de $X = [x_1, x_2]$:

$$\sqrt{X} = [\sqrt{x_1}, \sqrt{x_2}],\ x_1 \geq 0.$$

Código Fonte 3.10 Método de Cálculo da Raiz Quadrada.

```
1  public static Intervalo sqrt(Object in) throws ExcecaoOperacaoInvalida {
2          Intervalo res = getInterval(in);
3          Double inf = getDouble(res.getLimiteInferior());
4          Double sup = getDouble(res.getLimiteSuperior());
5          if (inf <= 0 || sup <= 0) {
6                  throw new ExcecaoOperacaoInvalida();
7          }
8          res.setLimiteInferior(String.valueOf(Math.sqrt(inf)));
9          res.setLimiteSuperior(String.valueOf(Math.sqrt(sup)));
10         res = validateIntervalo(res);
11         return res;
12 }
```

Exemplo 12. Calcular $\sqrt{[1.9,\ 2.2]}$.

```
1       public class Exemplos {
2               public static void main(String[] args) {
3                       Intervalo X = new Intervalo("1.9", "2.2");
4                       Intervalo resultado = Sqrt.sqrt(X);
5               }
6                       System.out.println("sqrt(X) = " + resultado);
7       }
```

Saída: `sqrt(X) = [13784048752090223E-16, 14832396974191326E-16]`

Exemplo 13. Calcular $\sqrt{[-1.9,\ 2.2]}$.

Saída: `Operação Inválida`

3.3.3 Exponencial

A função exponencial é aquela cuja variável sobre a qual está determinada a dependência da incógnita resultante encontra-se no expoente e é utilizada na representação de situações em que a variação é considerada muito grande, como em juros complexos da matemática financeira [27].

O método desenvolvido calcula e^X, onde e é a constante neperiana, ou número de Euler e X é o intervalo ao qual a constante está sendo elevada [23, 32, 33, 34]. Assim como nas funções anteriores o cálculo é desenvolvido sobre os limites do intervalo:

$$\begin{aligned} e^X &= [e^{x_1}, e^{x_2}], \\ e^{-X} &= [e^{-x_2}, e^{-x_1}]. \end{aligned}$$

Código Fonte 3.11 Método de Cálculo da Exponencial.

```
1 public static Intervalo exp(Object in) {
2         ...
3         res.setLimiteInferior(String.valueOf(Math.exp(inf)));
4         res.setLimiteSuperior(String.valueOf(Math.exp(sup)));
5         res = validateIntervalo(res);
6         return res;
7 }
```

Exemplo 14. Calcular $e^{[1,2]}$

```
1         public class Exemplos {
2                 public static void main(String[] args) {
3                         Intervalo X = new Intervalo("1.0", "2.0");
4                         Intervalo resultado = Exp.exp(X);
5                         System.out.println("exp(X) = " + resultado);
6                 }
7         }
```

Saída: `exp(X) = [27182818284590455E-16, 738905609893065E-14]`

Exemplo 15. Calcular $e^{[-2,-1]}$

```
1  public class Exemplos {
2      public static void main(String[] args) {
3          Intervalo X = new Intervalo("-2.0", "-1.0");
4          Intervalo resultado = Exp.exp(X);
5          System.out.println("exp(X) = " + resultado);
6      }
7  }
```

Saída: `exp(X) = [1353352832366127E-16, 36787944117144233E-17]`

3.3.4 Logaritmica

O cálculo do logaritmo é usado para identificar o expoente que uma base precisa para produzir determinada potência [23, 32]. A base dos cálculos de logaritmos adotada neste trabalho é sempre e. Com isso, o método recebe como parâmetro um intervalo sobre o qual o cálculo será aplicado:

$$ln\, X = [ln\, x_1, ln\, x_2],\ x_1 > 0.$$

Código Fonte 3.12 Método de Cálculo Logaritmico.

```
1  public static Intervalo log(Object in) {
2          ...
3          res.setLimiteInferior(String.valueOf(Math.log(inf)));
4          res.setLimiteSuperior(String.valueOf(Math.log(sup)));
5          res = validateIntervalo(res);
6          return res;
7  }
```

Exemplo 16. Calcular $ln\ [3.14, 3.15]$

```
1  public class Exemplos {
2      public static void main(String[] args) {
3          Intervalo X = new Intervalo("3.14", "3.15");
4          Intervalo resultado = Log.ln(X);
5          System.out.println("ln(X) = " + resultado);
6      }
7  }
```

Saída: `ln(X) = [1144222799920162E-15, 11474024528375417E-16]`

Exemplo 17. Calcular $ln\ [-3.14, 3.15]$

Saída: `Operação Inválida`

3.3.5 Seno

O cálculo do seno intervalar se dá de modo semelhante às funções anteriores [23, 34] e é definido por:

$$sen\ X = [sen(x_1), sen(x_2)],\ X \subseteq [\frac{-\pi}{2}, \frac{\pi}{2}].$$

Segundo [23], propriedades da função seno podem ser usadas para calcular extensões intervalares para qualquer argumento intervalar.

As funções trigonométricas para intervalos apresentam restrições por não serem globalmente monotônicas. Por exemplo, para determinar a função seno intervalar, deve-se calcular $sen(x_1)$ e $sen(x_2)$. Se $MAX(senX)$ for igual a $sen(x_1)$, ou seja, $sen(x1) > sen(x_2)$, então $sen([x_1, x_2]) = [sen(x_2), sen(x_1)]$, caso contrário, $sen([x_1, x_2]) = [sen(x_1), sen(x_2)]$ [34].

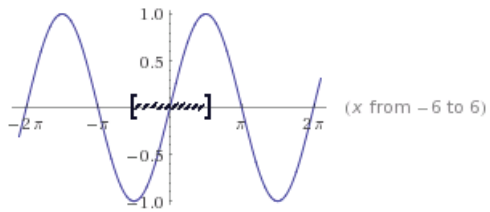

Figura 3.2 Gráfico do Seno

Código Fonte 3.13 Método de Cálculo do Seno.

```
1  public static Intervalo sin(Object in)
2                  throws ExcecaoOperacaoInvalida {
3          Intervalo res = getInterval(in);
4          Double inf = getDouble(res.getLimiteInferior());
5          Double sup = getDouble(res.getLimiteSuperior());
6          if (inf <= -Math.PI / 2 || sup >= Math.PI / 2) {
7                  throw new ExcecaoOperacaoInvalida();
8          }
9          res.setLimiteInferior(String.valueOf(Math.sin(inf)));
10         res.setLimiteSuperior(String.valueOf(Math.sin(sup)));
11         res = validateIntervalo(res);
12         return res;
13 }
```

Exemplo 18. Calcular $sen[-1.56, 1.56]$

```
1       public class Exemplos {
2               public static void main(String[] args) {
3                       Intervalo X = new Intervalo("-1.56", "1.56");
4                       Intervalo resultado = Trigonometrica.sin(X);
5                       System.out.println("sin(X) = " + resultado);
6               }
7       }
```

Saída: `sin(X) = [-9999417202299663E-16, 9999417202299663E-16]`

Exemplo 19. Calcular $sen[-3.15, -3.14]$

Saída: `Operação Inválida`

3.3.6 Cosseno

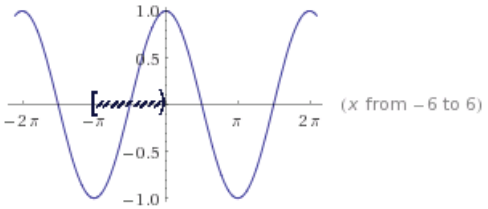

Figura 3.3 Gráfico do Cosseno

A expressão intervalar para o cálculo do cosseno é:

$$cos\, X = [cos(x_1), cos(x_2)],\ X \subseteq [-\pi, 0].$$

Código Fonte 3.14 Método de Cálculo do Cosseno.

```
1  public static Intervalo cos(Object in)
2                  throws ExcecaoOperacaoInvalida {
3          Intervalo res = getInterval(in);
4          Double inf = getDouble(res.getLimiteInferior());
5          Double sup = getDouble(res.getLimiteSuperior());
6          if (inf < -Math.PI || sup > 0) {
7                  throw new ExcecaoOperacaoInvalida();
8          }
9          res.setLimiteInferior(String.valueOf(Math.cos(inf)));
10         res.setLimiteSuperior(String.valueOf(Math.cos(sup)));
11         res = validateIntervalo(res);
12         return res;
13 }
```

Exemplo 20. Calcular $cos[-3.14, 0]$

```
1  public class Exemplos {
2          public static void main(String[] args) {
3                  Intervalo X = new Intervalo("-3.14", "3.15");
4                  Intervalo resultado = Trigonometrica.cos(X);
5                  System.out.println("cos(X) = " + resultado);
6          }
7  }
```

Saída: `cos(X) = [-9999987317275395E-16, 10E-1]`

Exemplo 21. Calcular $cos[-3.14, 3.15]$

Saída: `Operação Inválida`

3.3.7 Tangente

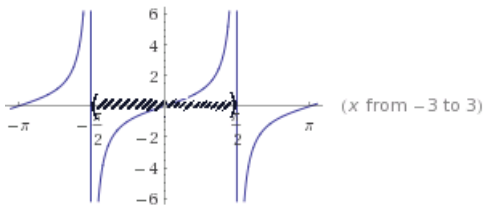

Figura 3.4 Gráfico da Tangente

A extensão intervalar para a tangente é:

$$tg\, X = [tg(x_1),\, tg(x_2)],\, X \subset (\frac{-\pi}{2}, \frac{\pi}{2}).$$

Código Fonte 3.15 Método de Cálculo da Tangente.

```
1  public static Intervalo tan(Object in)
2                  throws ExcecaoOperacaoInvalida {
3          Intervalo res = getInterval(in);
4          Double inf = getDouble(res.getLimiteInferior());
5          Double sup = getDouble(res.getLimiteSuperior());
6          if (inf < -Math.PI / 2 || sup > Math.PI / 2) {
7                  throw new ExcecaoOperacaoInvalida();
8          }
9          res.setLimiteInferior(String.valueOf(Math.tan(inf)));
10         res.setLimiteSuperior(String.valueOf(Math.tan(sup)));
11         res = validateIntervalo(res);
12         return res;
13 }
```

Exemplo 22. Calcular $tg\ [0, 1.57]$

```
1       public class Exemplos {
2               public static void main(String[] args) {
3                       Intervalo X = new Intervalo("0", "1.57");
4                       resultado = Trigonometrica.tan(X);
5                       System.out.println("tan(X) = " + resultado);
6               }
7       }
```

Saída: `tan(X) = [0, 12557655915007897E-13]`

Exemplo 23. Calcular $tg\ [-1.58, 1.57]$

Saída: `Operação Inválida`

3.4 Intervalos encapsuladores (enclosures) para probabilidades de variáveis aleatórias contínuas

Para efetivar a implementação do cálculo de intervalos encapsuladores, P_e, para as variáveis aleatórias contínuas Uniforme, Exponencial e Pareto [34] foi desenvolvida uma classe com métodos que suportam o tratamento das possíveis entrada de dados.

Todos os enclosures são calculados para $x_1 \leq x \leq x_2$. Todavia, esta não é uma restrição, porque a Uniforme tem densidade diferente de zero em $[a, b]$ e na Exponencial e na Pareto a densidade é diferente de zero em $[0, +\infty)$.

Para permitir que os métodos tenham uma maior abrangência na aceitação de entrada de dados, os mesmo recebem parâmetros do tipo `Object` e estes por sua vez são analisados nos métodos de tratamento de entrada.

Código Fonte 3.16 Método de tratamento dos parâmetros de entrada.

```
1  protected static Double trataEntrada(Object in) {
2          if (in == null) {
3                  return null;
4          } else if (in instanceof String && in.equals("inf")) {
5                  return null;
6          }
7          return Double.parseDouble(String.valueOf(in));
8  }
9
10 protected static boolean testeInf(Object in) {
11         if (in instanceof String && in.equals("inf")) {
12                 return true;
13         } else {
14                 return false;
```

```
15          }
16  }
17
18  protected static boolean testeInfNeg(Object in) {
19          if (in instanceof String && in.equals("-inf")) {
20                  return true;
21          } else {
22                  return false;
23          }
24  }
```

Esses métodos encontram-se na super classe `Distribuicao`, estando disponíveis, consequentemente, nas classes de todas as classes dela derivadas.

3.4.1 Distribuiçao Uniforme

A distribuição Uniforme é a mais simples distribuição contínua, porém é fundamental na teoria de probabilidade sendo utilizada para modelar a ocorrência de eventos cuja probabilidade é constante em intervalos de mesma dimensão.

A função de densidade de uma uniforme X no intervalo $A = [a,b]$ é dada por:

$$f(x) = \begin{cases} \frac{1}{b-a}, & x \in [a,b], \\ 0, & x \notin [a,b]. \end{cases}$$

Para aplicação da distribuição Uniforme na matemática Intervalar com suporte na Aritmética de Exatidão Máxima, fornecendo um intervalo encapsulador das probabilidades de menor amplitude [28] usa-se a expressão:

$$UNIF(c,d) = \left[\underline{\frac{w([c,d] \cap [a,b])}{b-a}}, \overline{\frac{w([c,d] \cap [a,b])}{b-a}} \right], [c,d] \cap [a,b] \neq \emptyset.$$

Onde w é o diâmetro (ou amplitude) do intervalo e os parâmetros de entrada do método de cálculo da distribuição Uniforme são [28]:

$$\begin{aligned} a &= \text{limite inferior do intervalo } A, \\ b &= \text{limite superior do intervalo } A, \\ c &= \text{limite inferior do intervalo } B, \\ d &= \text{limite superior do intervalo } B. \end{aligned}$$

Portanto, o algoritmo (código) calcula

$$P_e(c < X < d) = \int_c^d \frac{1}{b-a}\, dx = \texttt{Uniforme.calcular(a, b, c, d)}$$

Código Fonte 3.17 Método de cálculo de enclosures para a Uniforme.

```
1  public static Intervalo calcular(Object a, Object b, Object c, Object d) {
2          Intervalo resultado = null;
3
4          NovoIntervalo x = new NovoIntervalo(trataEntrada(a), trataEntrada(b));
5          NovoIntervalo y = new NovoIntervalo(trataEntrada(c), trataEntrada(d));
6          ListNumber lista1 = new ListNumber();
7          ListNumber lista2 = new ListNumber();
8          if (y.encapsula(x.getLimiteSuperior())
9                          || y.encapsula(x.getLimiteInferior())
10                         || x.encapsula(x.getLimiteSuperior())
11                         || x.encapsula(y.getLimiteInferior())) {
12                 Number l1[] = { x.getLimiteInferior(), x.getLimiteSuperior(),
13                                 y.getLimiteInferior(), y.getLimiteSuperior() };
14                 lista1.addAll(l1);
15                 lista2.addAll(l1);
16                 Number l2[] = { lista2.min(), lista2.max() };
17                 lista1.clear();
18                 lista1.addAll(l2);
19                 lista2.remove(lista2.min());
20                 lista2.remove(lista2.max());
21         }
22         if (Number.comparar((Number) lista2.get(0), (Number) lista1.get(0)) > 0
23                         && Number.comparar((Number) lista2.get(1),
24                                         (Number) lista1.get(1)) < 0) {
25                 Interseccao inter = new Interseccao(x, y);
26                 Intervalo z = inter.executarOperacao();
27                 Number diam = z.getDiametro();
28                 Number s1 = Number.dividir(diam, x.getDiametro(), false, 10);
29                 return NovoIntervalo.novoIntervalo(s1);
30         } else {
31                 resultado = new Intervalo(Double.toString(EPSILON));
32         }
33
34         return resultado;
35 }
```

Exemplo 24.

Suponha que X tem distribuição Uniforme em $(0,3)$. Calcular
$P_e(1/2 \leq X \leq 4/7)$. Então, $a = 0$, $b = 3$, $c = 1/2$, $d = 4/7 \equiv 0.57142$. Logo
$P_e(1/2 \leq X \leq 0.57142)$ = `Uniforme.calcular(0, 3, 1/2, 0.57142)`
= [2380666666666E-14, 2380666666668E-14].

```
1  public class Exemplos {
2      public static void main(String[] args) {
3          Double a = 0.0;
4          Double b = 3.0;
5          Double c = 0.5;
6          Double d = 0.57142;
7          Intervalo resultado = Uniforme.calcular(a, b, c, d);
8          System.out.println("Resultado = " + resultado);
9      }
10 }
```

Saída: `Resultado = [2380666666666E-14, 2380666666668E-14]`

3.4.2 Distribuiçao Exponencial

A variável aleatória contínua com função densidade

$$f(x) = \begin{cases} \alpha e^{-\alpha x}, & 0 \leq x < \infty, \\ 0, & x < 0, \end{cases}$$

com $\alpha > 0$, é uma variável aleatória Exponencial.

Seja $X = [x_1, x_2] \subset \mathbf{R}$. Para calcular $P_e(x_1 < x \leq x_2)$, é usado o método de Simpson Intervalar [20, 28, 34]. Este método foi implementado em uma classe a parte, passando a compor a biblioteca, permitindo assim, sua aplicação futura no desenvolvimendo de outras funções.

Código Fonte 3.18 Método de cálculo de Simpson Intervalar.

```
1  public class Simpson {
2
3          //  Metodo intervalar que implementa o Metodo de Simpson
4          //  Intervalar
5          //  a = limite inferior do intervalo
6          //  b = limite superior do intervalo
7          //  m = ponto medio do intervalo
8          //  d = amplitude do intervalo
9          //  z = intervalo referente a extensao intervalar da derivada
10         //  de ordem 4
11
12         public static Intervalo calcula(Double a, Double b, Double d, Double m,
13                         Intervalo z) {
14                 DecimalFormat deci = new DecimalFormat("0.000000000000000");
15
16                 Double s1 = Double.parseDouble(deci.format((d / 6) * (a + 4 * m + b))
17                                 .replace(",", "."));
18
19                 Double s2 = Double.parseDouble(deci.format(Math.pow(d, 5) / 2880)
20                                 .replace(",", "."));
21
22                 Intervalo i1 = new Intervalo(String.valueOf(s1));
23
24                 Intervalo i2 = new Intervalo(String.valueOf(s2));
25
26                 Intervalo resultado = new Subtracao(i1, new Multiplicacao(i2, z));
27
28                 return resultado;
29         }
30 }
```

Os parâmetros de entrada do método de cálculo da distribuição exponencial seguem o modelo definido por [28].

$limInf =$ limite inferior do intervalo A,
$limSup =$ limite superior do intervalo A,
$param =$ parâmetro α da função densidade f,
$subint =$ número de subintervalos da partição de A.

Código Fonte 3.19 Método de cálculo de enclosures para a Exponencial.

```
1  public static Intervalo calcular(Object limInf, Object limSup,
2                  Object param, Object subint) {
3          Double a = 0.0, b = 0.0, c = 0.0;
4          int p = Integer.parseInt(subint.toString());
5          a = trataEntrada(limInf);
6          b = trataEntrada(limSup);
7          c = trataEntrada(param);
8
9          Intervalo resultado = null;
10         if (a != null && b != null) {
11                 double h = (b - a) / p;
12                 double temp = a;
13                 ArrayList<String> listaInt = new ArrayList<String>();
14                 for (int i = 0; i < p; i++) {
15                         listaInt.add("item");
16                 }
17                 double s = 0;
18                 Intervalo si = new Intervalo("0");
19                 int ct = 0;
20                 DecimalFormat deci = new DecimalFormat("0.000000000000000");
21                 for (String item : listaInt) {
22                         double mi = (a + b) / 2;
23                         Double yyi = Double.parseDouble(deci.format(
24                                         c * Math.exp(-c * a)).replace(",", "."));
25                         Double yyii = Double.parseDouble(deci.format(
26                                         c * Math.exp(-c * b)).replace(",", "."));
27                         Double yymi = Double.parseDouble(deci.format(
28                                         c * Math.exp(-c * mi)).replace(",", "."));
29                         Double li1 = Double.parseDouble(deci.format(
30                                         Math.pow(c, 5) * Math.exp(-c * b))
31                                         .replace(",", "."));
32                         Double li2 = Double.parseDouble(deci.format(
33                                         Math.pow(c, 5) * Math.exp(-c * a))
34                                         .replace(",", "."));
35                         Intervalo yi = new Intervalo(String.valueOf(li1), String
36                                         .valueOf(li2).replace(",", "."));
37
38                         si = Simpson.calcula(a, b, h, yymi, yi);
39                         ct++;
40                 }
41                 Intervalo sub1 = new Intervalo(String.valueOf(s));
42                 resultado = new Subtracao(sub1, si);
43                 return resultado;
44         } else if (a == null && b > 0.0) {
45                 double s1 = 1 - Math.exp(-c * b);
46                 double s2 = 1 - Math.exp(-c * b);
47
48                 resultado = new Intervalo(String.valueOf(s1), String.valueOf(s2));
49         } else if (b == null && a > 0) {
50                 double s1 = Math.exp(-c * a);
51                 double s2 = Math.exp(-c * a);
52
```

```
53                  resultado = new Intervalo(String.valueOf(s1), String.valueOf(s2));
54          } else if (a < 0 && b < 0) {
55                  resultado = new Intervalo("0");
56          }
57          return resultado;
58  }
```

Assim, a probabilidade calculada, usando Caprani et al [6], Santos [28] e Varjão [34] é $P_e(a < X < b) = \int_a^b \alpha e^{-\alpha x} dx =$ `Exponencial.calcular(limInf, limSup, param, subint).`

Exemplo 25. Seja X uma variável aleatória com uma distribuição exponencial de parâmetro $\alpha = 0.001$. Calcular $P_e(20 < X < 50)$. Então, $a = 20$, $b = 50$ e o número de subintervalos da partição é 50.

```
1   public class Exemplos {
2           public static void main(String[] args) {
3                   Double limInf = 20.0;
4                   Double limSup = 50.0;
5                   Double param = 0.01;
6                   int subint = 50;
7                   Intervalo resultado = Exponencial.calcular(limInf, limSup,
8                                                   param, subint);
9                   System.out.println("Resultado = " + resultado);
10          }
11  }
```

Saída: `Resultado = [21220068858313947145OE-21, 21220068858316811845OE-21]`

Exemplo 26. Seja X uma variável aleatória com uma distribuição exponencial de parâmetro $\alpha = 0.01$. Calcular $P_e(0 < X < 50)$. Então, $a = 0$, $b = 50$ e o número de subintervalos da partição é 50.

Saída: `Resultado = [39347781599811338888889000E-26, 39347781599879649652778451700E-30]`

Exemplo 27. Seja X uma variável aleatória com uma distribuição exponencial de parâmetro $\alpha = 0.002$. Calcular $P_e(0 < X < 1250)$. Então, $a = 0$, $b = 1250$ e o número de subintervalos da partição é 50.

Saída: `Resultado = [41661862428881757778080OE-23, 4166728777610398E-15]`

Exemplo 28. Seja X uma variável aleatória com uma distribuição exponencial de parâmetro $\alpha = 0.002$. Calcular $P_e(0 < X < 1000)$. Então, $a = 0$, $b = 1000$ e o número de subintervalos da partição é 50.

Saída: `Resultado = [33337608948227222222400E-22, 33339386726005E-13]`

3.4.3 Distribuição de Pareto

A distribuição de Pareto nasceu da economia a partir da observação da relação entre população e renda. Tem sua função densidade definida por:

$$f(x) = \alpha.\frac{c^{\alpha}}{x^{\alpha+1}} \quad para \quad x \geq c.$$

Onde c e α são constantes positivas. Assim,

$$P_e(a < X < b) = \int_a^b \frac{\alpha c^{\alpha}}{x^{\alpha+1}}\, dx = \texttt{Pareto.calcular(a, b, c, d).}$$

Deste modo, para calcular o intervalo que encapsula a probabilidade para uma dada variável X, que segue uma distribuição de Pareto, define-se a função densidade como a seguinte extensão intervalar [34]:

$$F(X) = \alpha.\frac{c^{\alpha}}{X^{\alpha+1}} = \alpha\left(\left[\frac{c^{\alpha}}{x_2^{\alpha+1}}, \frac{c^{\alpha}}{x_1^{\alpha+1}}\right]\right),\ para\ 0 \notin X = [x_1, x_2].$$

A seguir estão definidos os parâmetros de entrada para aplicação no método de cálculo da distribuição de Pareto:

$limInf =$ limite inferior do intervalo A,
$limSup =$ limite superior do intervalo A,
$param =$ parâmetro da pareto (expoente) c,
$xInicial =$ parâmetro da pareto (x inicial) .

Código Fonte 3.20 Método de cálculo de enclosures para a Pareto.

```
1  public static Intervalo calcular(Object limInf, Object limSup, Object param,
2                  Object xInicial) {
3          double a = 0, b = 0, c = 0, x0 = 0;
4          a = trataEntrada(limInf);
5          b = trataEntrada(limSup);
6          c = trataEntrada(param);
7          x0 = trataEntrada(xInicial);
8
9          Intervalo resultado = null;
10         if (a > b) {
11                 System.out.println("Erro de entrada, intervalo improprio.");
12                 return null;
13         } else if (limSup != null && a >= x0) {
14                 Intervalo s1 = new Intervalo(String.valueOf(Math.pow(x0, c)));
15                 Intervalo s2 = new Intervalo(String.valueOf(Math.pow(b, c)
16                                 - Math.pow(a, c)));
17                 Intervalo s3 = new Intervalo(String.valueOf(Math.pow(b * a, c)));
18                 Intervalo s = new Multiplicacao(s1, s2);
19                 Number n = Number.dividir(s.getLimiteInferior(),
20                                 s3.getLimiteInferior(), false, 16);
21                 Number d = Number.dividir(s.getLimiteSuperior(),
22                                 s3.getLimiteSuperior(), false, 16);
23                 resultado = new Intervalo(n, d);
24         } else if (b > x0) {
25                 double s1 = EPSILON + (1 - (Math.pow(x0, c)) / Math.pow(b, c));
26                 double s2 = EPSILON + (1 - (Math.pow(x0, c)) / Math.pow(b, c));
27                 resultado = new Intervalo(String.valueOf(s1), String.valueOf(s2));
28         } else if (a >= x0 && limSup == null) {
29                 Intervalo s1 = new Intervalo(String.valueOf(Math.pow(x0, c)));
30                 Intervalo s2 = new Intervalo(String.valueOf(Math.pow(a, c)));
31                 resultado = new Divisao(s1, s2);
32         } else if (a < x0 && limSup == null) {
33                 double s1 = 1 - 60 * EPSILON;
34                 double s2 = 1 + 60 * EPSILON;
35                 resultado = new Intervalo(String.valueOf(s1), String.valueOf(s2));
36         }
37
38         if (Number.comparar(resultado.getLimiteInferior(),
39                         resultado.getLimiteSuperior()) == 0) {
40                 resultado = NovoIntervalo.novoIntervalo(resultado
41                                 .getLimiteInferior());
42         }
43         return resultado;
44 }
```

Exemplo 29. Seja X uma variável aleatória com distribuição de Pareto com parâmetros $\alpha = 0.25$ e $c = 1$. Calcular $P_e(1 < X < 2)$ =

`Pareto.calcular(1.0, 2.0, 0.25, 1.0)`

```
1   public class Exemplos {
2       public static void main(String[] args) {
3           Double limInf = 1.0;
4           Double limSup = 2.0;
5           Double param = 0.25;
6           Double xInicial = 1.0;
7           Intervalo resultado = Pareto.calcular(limInf, limSup,
8                                   param, xInicial);
9           System.out.println("Resultado = " + resultado);
10      }
11  }
```

Saída: `Resultado = [15910358474628543 49E-19, 15910358474628543 51E-19]`

Exemplo 30. Seja X uma variável aleatória com distribuição de Pareto com parâmetros $\alpha = 0.50$ e $c = 1$. Calcular

$$P_e(1 < X < 2) = \texttt{Pareto.calcular(1.0, 2.0, 0.50, 1.0)}$$

Saída: `Resultado = [29289321881345 2536E-18, 29289321881345 2538E-18]`

Exemplo 31. Seja X uma variável aleatória com distribuição de Pareto com parâmetros $\alpha = 0.75$ e $c = 1$. Calcular

$$P_e(1 < X < 2) = \texttt{Pareto.calcular(1.0, 2.0, 0.75, 1.0)}$$

Saída: `Resultado = [40539644249863950E-18, 40539644249863952E-18]`

Exemplo 32. Seja X uma variável aleatória com distribuição de Pareto com parâmetros $\alpha = 1$ e $c = 1$. Calcular

$$P_e(1 < X < 2) = \texttt{Pareto.calcular(1.0, 2.0, 1.0, 1.0)}$$

Saída: `Resultado = [4E-1, 6E-1]`

Exemplo 33. Seja X uma variável aleatória com distribuição de Pareto com parâmetros $\alpha = 1.25$ e $c = 1$. Calcular

$$P_e(1 < X < 2) = \texttt{Pareto.calcular(1.0, 2.0, 1.25, 1.0)}$$

Saída:
```
Resultado =
[57955179237314269E-17, 57955179237314271E-17]
```

Exemplo 34. Seja X uma variável aleatória com distribuição de Pareto com parâmetros $\alpha = 1.3$ e $c = 1$. Calcular

$$P_e(1 < X < 2) = \texttt{Pareto.calcular(1.0, 2.0, 1.3, 1.0)}$$

Saída:
```
Resultado =
[59387380182188223E-17, 59387380182188225E-17]
```

4

Benchmark para as Bibliotecas Intervalares Java e Python

Este capítulo apresenta um benchmark realizado entre as bibliotecas intervalares IntPy [5, 34] e BISN 2.0, a respeito dos cálculos de intervalos encapsuladores para probabilidades de variáveis aleatórias contínuas [28, 34], através da comparação de tempo de processamento despendido.

4.1 Desenvolvimendo do Benchmark

Um benchmark funciona numa comunidade científica como suporte dos paradigmas, sendo um paradigma um conjundo de conhecimentos necessários para o funcionamento de determinada disciplina, captando o consenso da comunidade em relação aos problemas dignos de estudo para determinar as soluções científicas aceitas. O papel de um benchmark é operacionalizar um paradigma levando um conceito de abstrato a concreto, servindo como um guia de ação [29].

A comparação realizada no benchmark delimita o problema e as medidas de desempenho demonstram as melhores soluções, trazendo consigo a vantagem de levar a comunidade científica a trabalhar em conjunto, posto que aponta um direcionamento para as pesquisas. Contudo, também limita, mesmo que temporariamente, os horizontes da comunidade, uma vez que por definição, a seleção de um paradigma não inclui outros [29].

As condições prévias apresentadas por [29] para a criação de um benchmark bem sucedido são (i) um nível mínimo de maturidade da disciplina de pesquisa e (ii) um desejo da comunidade científica de trabalhar em conjunto em busca da solução de problemas comuns, em outras palavras, uma cultura de trabalho colaborativo. Ambos demonstrados

nos diversos trabalhos desenvolvidos na busca de soluções intervalares para aplicação da matemática computacional [8, 18, 20, 28, 34].

No processo de desenvolvimento do benchmark aqui proposto foram observadas as propriedades definidas por [29] como necessárias para a criação de um benckmark bem sucedido. São elas:

1. Acessibilidade: O benchmark deve ser fácil de usar e de ser reproduzido, podendo ser reaplicado por qualquer um para comparar com outras ferramentas ou técnicas. Quando um benchmark é de fácil entendimento é menos provável que seja interpretado de maneira incorreta, o que aumenta sua credibilidade. As bibliotecas [5][18] podem ser acessadas, manipuladas e utilizadas pelos interessados; além da simplicidade das comparações feitas no benchmark permitem replicação do mesmo.

2. Disponibilidade: Os custos devem ser proporcionais aos benefícios. O ponto de equilíbrio para essa relação varia de acordo com a maturidade da tecnologia e do status de resultados de benchmark [29]. O benchmark desenvolvido possui custo baixo, sendo desenvolvido por apenas uma pessoa, tendo sido feito entre duas bibliotecas das diversas desenvolvidas [9], consequentemente um maior esforço pode permitir uma maior amplitude comparativa.

3. Clareza: A definição do benchmark deve ser clara e o mais curta possível de modo a garantir que não haja lacunas no seu entendimento. As medidas adotadas no benchmark aqui proposto foram os tempos de processamento e a precisão dos resultados para problemas predefinidos por [28] para o cálculo de probabilidades intervalares.

4. Relevância: A medida de desempenho deve ser pertinente para as comparações feitas e as tarefas devem ser representativas de um ambiente relativamente real. Esta é uma propriedade tão difícil de satisfazer quanto importante no processo. O tempo de processamento, adotado como medida comparativa, é fundamental em termos de avaliação de desempenho de software [14].

5. Resolubilidade: A tarefa de domínio deve ser possível de ser completada, isto é, uma tarefa possível, mas não trivial é uma oportunidade de um sistema mostrar suas capacidades e dependências. As funções comparadas no benchmark calculam intervalos encapsuladores de distribuições de probabilidade contínuas, que como

especificado não são cálculos triviais e representam a capacidade das bibliotecas comparadas.

6. Portabilidade: O benchmark deve ser especificado em um nível suficientemente alto de abstração para garantir que ele é portável para diferentes ferramentas e técnicas. Uma implicação dessa propriedade é que o benchmark pode precisar ser implementado mais de uma vez para diferentes plataformas ou para diferentes arquiteturas. Em função da simplicidade de sua especificação, o benchmark desenvolvido pode ser facilmente portado para outra plataforma sendo aplicado, por exemplo, à biblioteca [26].

7. Escalabilidade: As tarefas do benchmark devem ser escaláveis para ferramentas de diferentes níveis de maturidade, podendo ser aplicadas a protótipos ou produtos comerciais. Da mesma forma que foi aplicado no ambiente proposto, o benchmark pode ser escalado para ferramentas mais maduras como [19] ou ferramentas futuramente desenvolvidas.

De acordo com [29], um benchmark de sucesso é aquele que causa impacto significativo na área de pesquisa, Assim, o sucesso deste benchmark só poderá ser averiguado posteriormente com a avaliação e consequente adoção ou não da comunidade científica envolvida.

4.2 Especificação dos ambientes de testes

Os testes comparativos foram executados em computadores com especificações diferentes. Os resultados finais apresentados de cada ambiente sofreram influência desta distinção nas configurações, contudo foram mantidas as proporções relacionadas às tecnologias usadas:

1. **Computador 1:** Processador IntelCore i3 2.26 Ghz com 3.0 GB de memória RAM, utilizando sistema operacional Ubuntu 12.04 LTS 64bits, Linux Kernel 3.2.0-49-generic

   (a) Eclipse Juno Service Release 1; Java Oracle 7

   (b) Python 2.7.3

2. **Computador 2:** Processador IntelCore i3-2100 3.10GHz × 4 com 8.0 GB de memória RAM, utilizando sistema operacional Ubuntu 12.04 LTS 64bits, Linux Kernel 3.2.0-49-generic

(a) Eclipse Helios Service Release 2; Java Oracle 6

(b) Python 2.7.3

4.3 Escopo de testes

Foram selecionados 3 dos problemas apresentados por [34], um para cada algorítimo de cálculo de distribuição: (i) Uniforme, (ii) Exponencial e (iii) Pareto. A escolha dessas entradas serve por sua vez para validar os resultados de saída, visto que os mesmos foram originalmente sugeridos por [28], sendo assim calculados com IntPy e BISN 2.0.

Parâmetros aplicados ao cálculo da distribuição Uniforme,

$$a = 0.0,$$
$$b = 3.0,$$
$$c = 1.0,$$
$$d = 2.0,$$

isto é, $Pe(1 < X < 2)$.

Parâmetros aplicados ao cálculo da distribuição Exponencial,

$$limInf = 1000,$$
$$limSup = 1250,$$
$$param = 0.002,$$
$$subint = 100,$$

isto é, $Pe(1000 < X < 1250)$.

Parâmetros aplicados ao cálculo da distribuição de Pareto,

$$limInf = 1.0,$$
$$limSup = 2.0,$$
$$param = 0.25,$$
$$xInicial = 1.0,$$

isto é, $Pe(1 < X < 2)$.

Cada teste selecionado foi executado 2000 vezes em cada um dos ambientes disponíveis em cada uma das bibliotecas comparadas [5][18]. Esse número de repetições tem origem na observação de [18] na estabilidade de tempo de cálculos alcançada a partir da milésima repetição.

4.4 Resultados

A aplicação dos exemplos selecionados aos algoritmos implementados deu origem a resultados semelhantes, conforme pode ser visto a seguir:

- Distribuição Uniforme: $Pe(1 < X < 2)$.

  BISN 2.0: $[3333333332E-10,\ 3333333334E-10]$,

  Intpy: $[0.3333333333333333,\ 0.33333333333333337]$.

- Distribuição Exponencial: $Pe(1000 < X < 1250)$.

  BISN 2.0: $[5325143167568646663194444440000E-31,$
  $53251431675689857473958333300 0E-31]$,

  Intpy: $[0.05325143167564931,\ 0.053251431675655096]$.

- Distribuição de Pareto: $Pe(1 < X < 2)$.

  BISN 2.0: $[1591035847462854349E-19, 15910358474628543 51E-19]$,

  Intpy: $[0.15910358474628541,\ 0.15910358474628544]$.

Os tempos apresentados estão em milisegundos, e mostram a média de execução dos algoritmos, sendo obtidos com a execução dos scripts do Apendice A e B. Os scripts deram origem a arquivos de texto que foram posteriormente tabulados em planilhas para extração do tempo médio.

Tabela 4.1 Tempo médio de processamento

	Uniforme		Exponencial		Pareto	
	IntPy	BISN 2.0	IntPy	BISN 2.0	IntPy	BISN 2.0
Computador 1	0.1016	0.5365	4.6596	8.5745	0.0649	1.5155
Computador 2	0.0761	0.2180	3.5175	5.9480	0.0467	0.9315

Observa-se em todos os testes executados o melhor desempenho da biblioteca Intpy em relação a BISN 2.0 mantida nos dois ambientes de testes. Esta diferença no desempenho pode ser atribuida a características próprias das linguagens, além do desgaste de desempenho consequente do processamento de Strings da BISN 2.0 como visto nas análises de [18].

5

Conclusões, Contribuições e Trabalhos Futuros

Este capítulo apresenta as conclusões obtidas neste trabalho, assim como as perspectivas de trabalhos futuros.

5.1 Conclusões

Este trabalho se propôs a apresentar uma aplicação da matemática computacional na linguagem Java a partir da ampliação da BISN desenvolvida por [18], aplicando-a em cálculos mais elaborados da matemática intervalar. Além elaboração de um benchmark como proposta para a comunidade científica da área.

A extensão desenvolvida deu origem a BISN 2.0 com as sugestões de trabalhos futuros de [18], tendo acrescidas as funções potência, raiz quadrada, exponencial, logarítmica e trigonométricas validando seus cálculos com [37]. A nova versão da biblioteca, foi ainda incrementada com o cálculo de probabilidades para as variáveis aleatórias com base nos trabalhos de [28] e [34].

Durante o processo de implementação da ampliação biblioteca foram feitas experimentos, de implementação das distribuições Normal e Weibull executadas nos trabalhos de [20, 28]. Contudo, estas implementações foram abandonadas em virtude das dificuldades no cálculo de derivadas necessárias, que por sua vez, no MatLab [19], são obtidas com apenas um comando nativo (`diff`). Na linguagem Java [16] a função precisa ser totalmente manipulada para obtenção de sua derivada e mesmo adotando funções fixas o processo de desenvolvimento tornou-se muito dispendioso, inviabilizando sua conclusão em tempo hábil.

Em paralelo ao processo de implementação das funções, foi criado ainda um ambi-

ente web para aplicação das funções desenvolvidas, cujo objetivo seria fornecer uma ferramenta de simples utilização onde se pudesse testar os recursos da BISN 2.0 sem a necessidade efetiva de programação, comportando-se basicamente como uma calculadora intervalar. Entretanto, devido ao estreitamento do prazo de conclusão do trabalho dissertativo, este também teve que ser removido do escopo do projeto.

Por fim, foi desenvolvida uma proposta de benchmark para comparação de tecnologias aplicadas a matemática computacional. O comparativo foi feito entre a Intpy [5] e BISN 2.0 em termos de performance, onde ficou demonstrada a vantagem da biblioteca Python [24], com tempos de cálculos menores em todos os contextos aplicados.

Apesar das dificuldades encontradas, todos os cálculos obtiveram o sucesso almejado tendo seus resultados validados pela aplicação dos exemplos usados por [28] e [34]. Com isso, prova-se a viabilidade da utilização da linguagem Java [16] na computação científica.

5.2 Contribuições

As contribuições deste trabalho são:

(i) Ampliação do trabalho de Leite [18] com a inclusão das funções intervalares Exponencial, Logaritmica, Potência, Raiz Quadrada, Seno, Cosseno e Tangente.

Enclosures para as variáveis aleatórias Uniforme, Exponencial e Pareto.

(ii) Proposta de definição de um benchmark para bibliotecas intervalares.

(iii) Um passo a mais para caracterizar Java como Java-XSC [1] (eXtensions Scientific Computation), isto é, Java com os tipos intervalo, matrizes de intervalos, entre outros e operações sobre os tipos, com recursos necessários para o desenvolvimento de softwares numéricos [35]

5.3 Trabalhos Futuros

A seguir são apresentados possíveis continuações deste trabalho:

- Faz-se importante uma continuidade do desenvolvimento da BISN 2.0, levando-a mais níveis de aplicabilidade. Analisando-se por exemplo os problemas de performance causados pelo processamento das Strings [18].

[1] http://www.cin.ufpe.br/ javaxsc/

- Implementando o cálculo intervalar de todas as funções trigonométricas, utilizando as propriedades reais das mesmas.
- Sugere-se também a implementação das distribuições Normal e Weilbull desenvolvidas por [20] aumentando a complexidadde da aplicação e o arcabouço para aplicação do benchmark.
- O cálculo de probabilidades através dos métodos usuais do Cálculo (na reta) envolve ∞. Por exemplo, considerando a variável aleatória do Exemplo 25,

$$P(X > 30) = \int_{30}^{\infty} 0.01\, e^{-0.01x}\, dx = e^{-0.3}.$$

  Como tratar ∞ em Java?
- Como definido por [29], um benchmark é considerado significativo para determinada área científica quando é validado pela comunidade envolvida. Com isso, propõem-se novas aplicações do benchmark aqui desenvolvido paralelizando outras tecnologias aplicadas a matemática computacional, como [26].

Referências

[1] ACIOLY, B. M. (1991): "Fundamentação computacional da matemática intervalar"Universidade Federal do Rio Grande do Sul. Instituto de Informática. Curso de Pós-Graduação em Ciência da Computação. Disponível em http://hdl.handle.net/10183/18234.

[2] ANSELMO, F. "Aplicando Lógica Orientada a Objetos em Java.- 2a. Ed Atual e ampl. - Florianópolis: VisualBooks, 2005.

[3] ARNOLD, D. N. “Some disasters attributable to bad numerical computing” Disponível em: http://www.ima.umn.edu/ arnold/disasters/disasters.html. Acessado em 21/02/2012.

[4] AUGUSTIN, I., "Abstrações para uma Linguagem de Programação Visando Aplicações Móveis em um Ambiente da Pervasive Computing"Universidade Federal do Rio Grande do Sul. Instituto de Informática. Curso de Pós-Graduação em Ciência da Computação, 2004. Disponível em http://hdl.handle.net/10183/3866

[5] BARRETO R. M., "IntPy". Open Source. Disponível em http://pypi.python.org/pypi/IntPy/ 0.1.3, 2009. Acessado em 02/10/2012.

[6] CAPRANI, O., Madsen, K., Nielsen, H. B. "Introduction to Interval Analysis."IMM - Informatics and Mathematical Modelling, pp. 82. Technical University of Denmark, DTU, 2002.

[7] CLÁUDIO, D. M., MARTINS, J. M."Cálculo Numérico Computacional - Teoria e Prática- São Paulo - Editora Atlas S.A., 1994 Editora: ATLAS

[8] CAMPOS, M. A. "Uma Extensão Intervalar para a Probabilidade Real."1997. 127 f. Tese, Departamento de Informática, Universidade Federal de Pernambuco, Recife, 1997.

[9] C-XSC, Disponível em: http://www2.math.uni-wuppertal.de/ xsc/xsc/cxsc.html. Acessado em 17/09/2012.

[10] GONÇALVES, E. "Desenvolvendo Aplicações Web com JSP, Servlets, Javaserver Faces, Hibernate, EJB 3 Persistence e Ajax."Rio de Janeiro – Editora Ciência Moderna, 2007.

[11] GRIGOLETTI, P. S., DIMURO, G. P., BARBOZA, L. V. "Módulo Python para Matemática Intervalar". TEMA Tend. Mat. Apl. Comput., 8, No. 1, 2007.

[12] HEMRAJANI, A. "Desenvolvimento ágil em Java com Spring, Hibernate e Eclipse."Tradução Edson Furmankiewicz e Sandra Figueiredo, revisão técnica Nivaldo Foresti. São Paulo, Pearson Prentice Hall, 2007

[13] HICKEY, T. J., QIU, Z, EMDEN, M. H. "Interval Constraint Plotting for Interactive Visual Exploration of Implicitly Defined Relations"in the special issue on Reliable Geometric Computations, in Reliable Computing, Vol 6., No. 1, 2000.

[14] HÖLBIG, C. A. "Ambiente de Alto Desempenho com Alta Exatidão para a Resolução de Problemas". Tese, Porto Alegre: PPGC da UFRGS, 2005.

[15] HORSTMANN, C., CORNELL G. "Core Java 2."Rio de Janeiro - Editora Alta Books, 2005.

[16] Java, Disponível em: http://www.oracle.com/technetwork/java/index.html. Acessado em 21/02/2012.

[17] KULISH, U. W. and MIRANKER W. L. “Computer Arithmetic in Theory and Practive”, Academic Press, New York, 1981.

[18] LEITE, I. O. B. “Uma Biblioteca Intervalar baseada em Processamento de Strings”, Dissertação de Mestrado, Pós-Graduação em Ciência da Computação, Centro de Informática/UFPE, Recife, PE, 2007.

[19] MATLAB, Disponível em: http://www.mathworks.com/. Acessado em 21/01/2012.

[20] MENDONÇA, A. F. “Intervalos Autovalidáveis para Métricas de Confiabilidade”, Dissertação de Mestrado, Pós-Graduação em Ciência da Computação, Centro de Informática/UFPE, Recife, PE, 2012.

[21] MOORE R. E., "Interval Analysis". Prentice Hall, Englewood Clifs, NJ, 1966.

[22] MOORE, R. E., “Methods and Applications of Interval Analysis”. SIAM Studies in Applied Mathematics, Philadelphia, 1979.

[23] MOORE, R. E., KEARFOTT, R. B. Kearfott, CLOUD, M. J. “Introduction to Interval Analysis”. SIAM, 2009.

[24] PYTHON, Disponível em: http://www.python.org/. Acessado em 21/01/2012.

[25] RICARTE, I. L. M. "Programação de Sistemas: Uma Introdução- UNICAMP (Universidade Estadual de Campinas), 2003. Disponível em http://www.dca.fee.unicamp.br/cursos/EA876/apostila/HTML/node1.html.

[26] RUMP S. M. “INTLAB-INTerval LABoratory”, Dordrecht, Institute for Reliable Computing, Alemanha: Kluwer Academic Publishers, 77-104, 1999.

[27] SANTANA, F. T. "Uma Fundamentação para Sinais e Sistemas Intervalares", Tese - Universidade Federal do Rio Grande do Norte. Centro de Tecnologia. Programa de Pós-Graduação em Engenharia Elétrica e de Computação. - Natal, RN, 2011.

[28] SANTOS, M. G. “Uma Extensão Intervalar para Variáveis Aleatórias Contínuas”, Tese, Pós-Graduação em Matemática Computacional /UFPE, Recife, 2010.

[29] SIM, S. E.; EASTERBROOK, S.; HOLT, R. C. "Using Benchmarking to Advance Research: A Challenge to Software Engineering.", 25th International Conference on Software Engineering, Portland, Oregon, 2003.

[30] TIOBE Company, Disponível em http://www.tiobe.com/

[31] TRINDADE, R. M. P. "Uma fundamentação matemática para processamento digital de sinais intervalares", Tese - Universidade Federal do Rio Grande do Norte. Centro de Tecnologia. Programa de Pós-Graduação em Engenharia Elétrica e de Computação. - Natal, RN, 2009.

[32] VARJÃO F. R. G., CAMPOS, M. A. “Uma Extensão para IntPy com as Funções Exponencial, Logarítmo, Potência, Raiz Quadrada e Trigonométricas”, de 21 a 23 de outubro, João Pessoa - PB, Encontro Regional de Matemática Aplicada e Computacional, 2009.

[33] VARJÃO F. R. G., CAMPOS, M. A. “Implementando extensões intervalares de funções em Python”, de 20 a 23 de setembro, Águas de Lindóia - SP, XXXIII Congresso Nacional de Matemática Aplicada e Computacional, 2010.

[34] VARJÃO, F. R. G. “IntPy: Computação Científica Auto Validável em Python”, Dissertação de Mestrado, Pós-Graduação em Ciência da Computação, Centro de Informática/UFPE, Recife, PE, 2012.

[35] XSC Languages, http://www.xsc.de/. Acessado em 17/09/2012.

[36] WIENER, N. "A contribution to the theory of relative position", Proc. Cambridge Philos. Soc., Vol. 17, pp. 441-449, 1914

[37] WOLFRAMALPHA, Disponível em: http://www.wolframalpha.com/. Acessado em 12/08/2012.

A

Scripts de aferição de tempo BISN

A.1 Aferição de tempo da distribuição Uniforme

```
1  public void marcaTempo(int repeticoes) {
2
3          PrintWriter file = null;
4          File arquivo = new File(nomeArquivo);
5          Intervalo resultado;
6
7          try {
8                  arquivo.createNewFile();
9                  file = new PrintWriter(arquivo);
10
11                 for (int i = 0; i < repeticoes; i++) {
12                         tempoInicial = System.currentTimeMillis();
13
14                         resultado = Uniforme.calcular(0.0, 3.0, 1.0, 2.0);
15
16                         tempoFinal = System.currentTimeMillis();
17
18                         file.println("Uniforme.calcular(0.0, 3.0, 1.0, 2.0);"
19                                         + resultado + ";" + tempoInicial + ";"
20                                         + tempoFinal+ ";"
21                                         + (tempoFinal - tempoInicial));
22                 }
23                 file.flush();
24                 file.close();
25         } catch (IOException e) {
26                 System.out.println("Problemas na criacao do arquivo");
27                 System.out.println(e.getMessage());
28         } finally {
29                 file.flush();
30                 file.close();
31         }
32         System.out.println("Processo Finalizado!");
33 }
```

A.2 Aferição de tempo da distribuição Exponencial

```
1  public void marcaTempo(int repeticoes) {
2
3          PrintWriter file = null;
4          File arquivo = new File(nomeArquivo);
5          Intervalo resultado;
6
7          try {
8                  arquivo.createNewFile();
9                  file = new PrintWriter(arquivo);
10
11                 for (int i = 0; i < repeticoes; i++) {
12                         tempoInicial = System.currentTimeMillis();
13
14                         resultado = Exponencial.calcular(1000, 1250, 0.002, 50);
15
16                         tempoFinal = System.currentTimeMillis();
17
18                         file.println("Exponencial.calcular(1000, 1250, 0.002, 50);"
19                                         + resultado + ";" + tempoInicial + ";"
20                                         + tempoFinal+ ";"
21                                         + (tempoFinal - tempoInicial));
22                 }
23                 file.flush();
24                 file.close();
25         } catch (IOException e) {
26                 System.out.println("Problemas na criacao do arquivo");
27                 System.out.println(e.getMessage());
28         } finally {
29                 file.flush();
30                 file.close();
31         }
32         System.out.println("Processo Finalizado!");
33 }
```

A.3 Aferição de tempo da distribuição Pareto

```
1  public void marcaTempo(int repeticoes) {
2
3          PrintWriter file = null;
4          File arquivo = new File(nomeArquivo);
5          Intervalo resultado;
6
7          try {
8                  arquivo.createNewFile();
9                  file = new PrintWriter(arquivo);
10
11                 for (int i = 0; i < repeticoes; i++) {
12                         tempoInicial = System.currentTimeMillis();
```

```
13
14                          resultado = Pareto.calcular(1.0, 2.0, 0.25, 1.0);
15
16                          tempoFinal = System.currentTimeMillis();
17
18                          file.println("Pareto.calcular(1.0, 2.0, 0.25, 1.0);"
19                                  + resultado + ";" + tempoInicial + ";"
20                                  + tempoFinal+ ";"
21                                  + (tempoFinal - tempoInicial));
22                  }
23                  file.flush();
24                  file.close();
25          } catch (IOException e) {
26                  System.out.println("Problemas na criacao do arquivo");
27                  System.out.println(e.getMessage());
28          } finally {
29                  file.flush();
30                  file.close();
31          }
32          System.out.println("Processo Finalizado!");
33  }
```

B

Scripts de aferição de tempo IntPy

B.1 Aferição de tempo da distribuição Uniforme

```
1  import time
2  from intpy import *
3  import math
4
5  arq = open('Uniforme_py.csv', 'w')
6  print "Iniciando processo..."
7  for i in range(1, 2001):
8      t1 = time.time()
9      res = dunif(0.0, 3.0, 1.0, 2.0)
10     t2 = time.time()
11     texto = "dunif(0.0, 3.0, 1.0, 2.0)"+str(res)+";"
12                 +str(t1)+";"+str(t2)+";"+
13                  str(t2-t1).replace('.',',')+";\n"
14     arq.write(texto)
15 arq.close()
16 print "Processo Finalizado!"
```

B.2 Aferição de tempo da distribuição Exponencial

```
1  import time
2  from intpy import *
3  import math
4
5  arq = open('Exoponencial_py.csv', 'w')
6  print "Iniciando processo..."
7  for i in range(1, 2001):
8      t1 = time.time()
9      res = dexp(1000,1250,0.002,100)
10     t2 = time.time()
11     texto = "dexp(1000,1250,0.002,100);"+str(res)+";"
12                 +str(t1)+";"+str(t2)+";"+
```

```
13                  str(t2-t1).replace('.',',')+";\n"
14      arq.write(texto)
15  arq.close()
16  print "Processo Finalizado!"
```

B.3 Aferição de tempo da distribuição Pareto

```
1   import time
2   from intpy import *
3   import math
4
5   arq = open('Pareto_py.csv', 'w')
6   print "Iniciando processo..."
7   for i in range(1, 2000):
8       t1 = time.time()
9       res = dpareto(1.0, 2.0, 0.25, 1.0)
10      t2 = time.time()
11      texto = "dpareto(1.0, 2.0, 0.25, 1.0);"+str(res)+";"
12                  +str(t1)+";"+str(t2)+";"+
13                  str(t2-t1).replace('.',',')+";\n"
14      arq.write(texto)
15  arq.close()
16  print "Processo Finalizado!"
```