UNIVERSIDADE FEDERAL DO PAMPA

Tiago Domingos dos Santos

Definição da exponencial na aritmética intervalar RDM e aplicação em problemas com distribuição de probabilidade

Alegrete

2018

Tiago Domingos dos Santos

Definição da exponencial na aritmética intervalar RDM e aplicação em problemas com distribuição de probabilidade

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Profa. Ma. Alice Fonseca Finger

Alegrete

2018

Tiago Domingos dos Santos

Definição da exponencial na aritmética intervalar RDM e aplicação em problemas com distribuição de probabilidade

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Trabalho de Conclusão de Curso defendido e aprovado em 28 de junho de 2018

Banca examinadora:

Alice Fonseca Finger

Profa. Ma. Alice Fonseca Finger
Orientadora
UNIPAMPA

Prof. Dr. Marcelo Resende Thielo
UNIPAMPA

Celso Nobre da Fonseca.

Prof. Me. Celso Nobre da Fonseca
UNIPAMPA

AGRADECIMENTOS

"Que posso eu oferecer a Deus, o Senhor, por tudo de bom que ele me tem dado? (Salmos 116:12)". - Minha vida! Minha gratidão! Agradeço, primeiramente, ao meu Deus, a Rocha que me mantém firme diante dos ventos de problemas da vida, que me deu ânimo, sabedoria e fé para concluir este curso. Nada sai do controle de Deus, e tudo corre segundo o conselho da Sua vontade. Todos parágrafos da minha vida seriam pouco para agradecer a Deus. Agradeço aos meus pais, Otávio e Ivone, mais que meus genitores, são meus professores, meusamigos e meus heróis. Agradeço por me ensinarem valores que hoje, infelizmente, são tidos como antiquados, retrógrados e dia após dia caem em desuso, como a lealdade, a sinceridade, a persistência e a honestidade. Prometo a vocês jamais abandonar estes ensinamentos e propagá-los sempre. Agradeço à Prof. Alice, pela oportunidade de trabalhar seu lado, pela compreensão diante dos problemas, pela motivação, pela amizade, pela alegria, pela atenção e pela dedicação que demonstrou em todas conversas, reuniões e sucessivas revisões feitas. À todos aqueles que foram meus professores no Curso de Ciência da Computação da Universidade Federal do Pampa, digo - Nobre missão é a de ensinar, muito obrigado! Por fim, às amizades adquiridas e nutridas ao longo da graduação, digo, sem citar nomes, obrigado!

"Por isso não desanimamos. Embora exteriormente
estejamos a desgastar-nos, interiormente estamos sendo renovados
dia após dia, pois os nossos sofrimentos leves e momentâneos estão produzindo
para nós uma glória eterna que pesa mais do que todos eles. Assim, fixamos
os olhos não naquilo que se vê, mas no que não se vê , pois o que se vê
é transitório, mas o que não se vê é eterno."
(Bíblia Sagrada, 2 Coríntios, 4:16-18)

RESUMO

Problemas numéricos em sistemas digitais acontecem devido a característica destes operarem sobre o conjunto dos números reais, obrigando-os a discretizar este conjunto por intermédio de truncamentos e arredondamentos. Devido a restrições da máquina, nem sempre é possível representar valores em ponto flutuante de maneira exata. A aritmética intervalar surgiu como uma ferramenta que possibilita a solução destes problemas numéricos, tratando imprecisões e propiciando um controle automático de erros, através da aproximação de um número real por um intervalo com limites superior e inferior. Entretanto, o modelo da aritmética intervalar de Moore apresenta algumas limitações, como, por exemplo, o excesso de largura deste intervalo e dependência de dados. Com o objetivo de corrigir estes e outros problemas foi desenvolvida a *Relative-Distance-Measure (RDM) interval arithmetic*. Trabalhos atuais provam que essa nova aritmética retorna resultados mais exatos quando comparados com Moore. O objetivo deste trabalho é definir a exponencial natural para aritmética RDM e aplicar em problemas com distribuição de probabilidade esperando obter resultados mais exatos. Por fim, foi feita a verificação da qualidade dos resultados obtidos através da aplicação de métricas de qualidade ao intervalos resultantes das aplicações desenvolvidas com os métodos de integração de Rall, de Bedregal e de Simpson Intervalar, bem como a comparação destes resultados entre as aritméticas de Moore e RDM e análise de complexidade da exponencial RDM que foi definida neste trabalho.

Palavras-chave: Aritmética intervalar. Aritmética intervalar RDM. Computação científica. Esforço computacional. Função exponencial RDM .

ABSTRACT

Numerical problems in digital systems happen due to their characteristic operate on the set of real numbers, forcing them to discretize this set by means of truncations and rounding. Due to machine constraints, it is not always possible to represent floating point values exactly. Interval arithmetic emerged as tool that allows the solution of these numerical problems, treating inaccuracies and providing an automatic error control, by approaching a real number by a range with upper and lower limits. However, Moore's interval arithmetic model has some limitations, such as the excess width of this interval and dependence on data. In order to correct these and other problems the Relative-Distance-Measure (RDM) interval arithmetic was developed. Current works prove that this new arithmetic returns more accurate results when compared to Moore. The objective of this work is to define the natural exponential for RDM arithmetic and to apply probabilities with probability distribution hoping to obtain more exact results. Finally, we verified the quality of the results obtained by applying quality metrics to the intervals resulting from the applications developed with the integration methods of Rall, Bedregal and Simpson Intervalar, as well as the comparison of these results among the arithmetic of Moore and RDM and complexity analysis of the exponential RDM that was defined in this work.

Key-words: Interval arithmetic. RDM arithmetic. Scientific computing. Computational effort. RDM exponential function.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A análise intervalar surgiu com o objetivo inicial de controlar a propagação de erros numéricos em procedimentos computacionais. Estes erros acontecem em virtude dos sistemas computacionais atuais estarem padronizados para operar com formatos e métodos da aritmética de ponto flutuante (ZURRAS et al., 2008). Assim, números reais são representados por aproximação através de um subconjunto chamado números de máquina.

Devido a esta representação, são gerados dois tipos de erros: o primeiro ocorre quando um valor real de entrada é aproximado para um número de máquina; o segundo é causado pelos resultados intermediários gerados na execução de cada operação e que vão se acumulando (RATSCHEK; ROKNE, 1988). E existe ainda outro tipo de erro, mas está relacionado a incerteza dos dados de entrada provindos, por exemplo, de experimentos físicos.

Sendo assim, a matemática intervalar fornece uma solução aos erros ocasionados por aproximações através do uso de intervalos. Na aritmética intervalar (MOORE, 1966), um valor real x é aproximado por um intervalo fechado $\mathbf{X} = [\underline{x}, \overline{x}]$ onde o $\underline{x}$ e $\overline{x}$ são os limites inferior e superior, respectivamente, do intervalo e x está contido neste intervalo.

Entretanto, a aritmética de Moore possui algumas limitações (LANDOWSKI, 2015) como, por exemplo, o excesso de largura dos intervalos e dificuldade em resolver problemas simples como a adição do inverso intervalar. Procurando resolver estes e outros problemas e limitações, Piegat e Landowski (2012) apresentaram em seu trabalho uma nova aritmética intervalar, a chamada *Relative-Distance-Measure (RDM) interval arithmetic*.

Nesta aritmética RDM um valor real x é aproximado por um intervalo fechado $\mathbf{X} = [\underline{x}, \overline{x}]$, entretanto, x é descrito usando a variável RDM $\alpha_x, \alpha_x \in [0, 1]$ de tal forma que $x = \underline{x} + \alpha_x \cdot (\overline{x} - \underline{x}), \alpha_x \in [0, 1]$.

Esta forma de representar cada variável do problema permite levar em consideração cada propriedade desta variável que a torna única e, caso haja necessidade, considerar todos os valores intermediários dentro de um intervalo, ou seja, cada variável adiciona uma dimensão ao problema, o que torna a aritmética RDM multidimensional.

Além disso, Piegat e Landowski (2012) e Piegat e Landowski (2013) definiram as operações básicas para a aritmética demonstrando, inclusive, as propriedades matemáticas satisfeitas em cada operação RDM e apresentando por intermédio de figuras os resultados obtidos.

Na literatura é possível encontrar a aritmética de Moore empregada em aplicações com distribuições de probabilidade, porém não é conhecido se a aritmética RDM pode retornar resultados melhores que a aritmética de Moore ao resolver esse tipo de problemas.

Diante de uma nova definição pra intervalos e com trabalhos científicos mostrando que tal aritmética retorna resultados mais exatos que a de Moore (PIEGAT; LANDOWSKI, 2012) (PIEGAT; LANDOWSKI, 2013)(LANDOWSKI, 2015), o objetivo deste trabalho é definir a exponencial intervalar RDM, até então, não encontrada na literatura, e submetê-la à problemas com distribuição de probabilidade utilizando os métodos de integração de Rall, Bedregal e Simpson intervalar. Após, analisar os intervalos obtidos como resposta, quanto a sua exatidão e qualidade, com a expectativa de obter intervalos mais exatos quando comparados a operação real e a aritmética de Moore, além de observar qual método de integração, entrega resultados mais confiáveis.

Para alcançar este objetivo, todas as implementações serão realizadas utilizando o pacote de extensão intervalar da linguagem python, chamado IntPy(BARRETO, 2009). A partir de então, fazer a análise de qualidade, por intermédio de métricas de erros dos intervalos obtidos. Concluindo, por fim, qual das aritmética intervalares, Moore ou RDM, retorna resultados mais exatos quando aplicada às distribuições de probabilidades.

1.1 Objetivos

O objetivo deste trabalho é empregar a aritmética intervalar RDM em aplicações com distribuição de probabilidade, esperando encontrar intervalos mais exatos, em virtude das características desta aritmética.

A fim de atingir o objetivo principal, os objetivos específicos a serem executados são listados e detalhados a seguir:

- Definir a operação de exponenciação para a aritmética RDM uma vez que esta não é encontrada nas literaturas para que seja possível o seu emprego nas distribuições de probabilidade utilizando as integrais numéricas.

- Estudar os métodos de integração numérica intervalares, buscando identificar alguma limitação que possam oferecer ao emprego da aritmética RDM e verificar aqueles métodos que retornam, naturalmente, um melhor resultado.

- Identificar as distribuições de probabilidade que já possuam sua extensão intervalar definida para aritmética de Moore, que, preferencialmente, façam uso de da operação exponecial de base natural e que, se possível, já possuam exemplos numéricos com a aritmética intervalar convencional, de maneira a propiciar a certeza do corretude da aritmética RDM aplicada e com o objetivo de comparação prévia de resultados.

- Após o estudo dos métodos de integração intervalar a serem utilizados e do contexto em que serão aplicados, implementar os algoritmos para a sua forma real e intervalar. Utilizar, para isso, a linguagem de programação Python (PYTHON, 2018) operando com o pacote IntPy, desenvolvido por Barreto (2009), o qual possibilita a programação utilizando intervalos.
- Apresentar os resultados de qualidade para a aplicação utilizando medidas de erro absoluto e erro relativo, bem como o cálculo do diâmetro do intervalo, comparando os resultados obtidos com a aritmética de Moore.
- Realizar a análise de complexidade da exponencial intervalar RDM implementada a fim de analisar se o esforço computacional despendido na aplicação é aceitável a ponto de justificar o emprego de intervalos, em especial, da aritmética intervalar RDM.

1.2 Organização deste Trabalho

Este trabalho está organizado da seguinte forma:

O capítulo 2 apresenta uma fundamentação teórica sobre a matemática intervalar de Moore, citando seus principais conceitos, bem como suas operações básicas. Posteriormente, são abordadas as métricas de erro, apresentando as principais fontes de erro dos sistemas digitais e a maneira de quantificá-lo. Além disso, é apresentada uma breve síntese sobre integração numérica. Em seguida, é apresentada a definição de análise de complexidade de algoritmos, suas características e sua importância para este trabalho. Por fim, é apresentado o pacote IntPy, sua arquitetura e seu funcionamento.

Após, no capítulo 3 é apresentada uma síntese sobre a nova aritmética intervalar *Relative-Distance-Measure (RDM) interval arithmetic*, que é a primeira motivação deste trabalho, seus principais conceitos e suas operações básicas definidas.

Em seguida, o capítulo 4 apresenta um extrato de trabalhos desenvolvidos que apresentam relação com os temas: aritmética intervalar, aritmética intervalar RDM e computação científica, além de servirem como alicerce de conhecimento para este trabalho.

Posteriormente, o capítulo 5 apresenta a definição da exponenciação Real e a exponencial intervalar RDM, definida e desenvolvida neste trabalho. Em relação a exponencial RDM, é apresentada a motivação para a criação desta aplicação, bem como o raciocínio matemático utilizado para enfrentar o problema, a solução proposta e a justificativa para esta solução. Por fim é apresentada a análise da complexidade da operação exponencial RDM definida.

O capítulo 6 apresenta a análise dos resultados obtidos, tanto com a aritmética de ponto flutuante, quanto com as aritméticas intervalares RDM e de Moore. Os resultados

são apresentados agrupados por distribuições de probabilidade com os métodos de integração intervalares de Rall, de Bedregal de Simpson. Por fim, são apresentados, ainda, os tempos que cada solução gastou para sua conclusão, bem como as métricas de erro para cada intervalo.

Finalizando este trabalho, no capítulo 7 são realizadas as considerações finais deste trabalho. Este capítulo foi reservado, ainda, para perspectivas de trabalhos futuros, e após isso são apresentados as bibliografias e os anexos deste trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo tem por objetivo apresentar e discutir assuntos necessários a produção e entendimento deste trabalho. Primeiramente, são abordados os fundamentos da teoria da matemática intervalar apresentadas por Moore (1966), bem como são descritas as operações aritméticas intervalares básicas.

Na seção posterior, são abordados as métricas de erro, sua definição e são apresentados as principais fontes de erro e as formas de quantificá-lo. E em seguida, é apresentado o conceito de integrais e as principais integrais intervalares que possivelmente serão utilizadas no trabalho. Posteriormente, é feita uma explanação sobre complexidade de algoritmos que será usado no futuro para avaliar e comparar os resultados obtidos com os métodos de integralização utilizando a aritmética RDM, quanto ao esforço computacional, em relação à aritmética de Moore.

Por fim, será apresentada a biblioteca computacional IntPy, desenvolvida para a linguagem de programação Python, que implementa funções voltadas à matemática intervalar, desenvolvida por Barreto (2009). Esta biblioteca será utilizada neste trabalho para fins de implementação.

2.1 Aritmética Intervalar de Moore

A aritmética intervalar (MOORE, 1966) (MOORE, 1979) (MOORE; KEARFOTT; CLOUD, 2009) utiliza intervalos fechados e reais $[x_1\ ,\ x_2]$, para representar valores infinitos, valores desconhecidos ou ainda para representar valores contínuos, conhecidos ou não (LORETO, 2006).

A ideia de representar resultados de problemas em forma de intervalos objetiva um controle automático de erros com limites confiáveis. Pois, devido a limitação computacional de representar um valor real x, este é aproximado por um intervalo $\mathbf{X}$ que possui um limite inferior $\underline{x}$ e um limite superior $\overline{x}$ de forma que contenha o valor real x.

A definição matemática para esta operação é a seguinte: dada uma função $f(x)$, onde $x \in [x_1\ ,\ x_2]$, e x_1,$x_2 \in \mathbb{R}$, a imagem de f é dada por

$$f(x) = \{y | y = f(x), x_1 \leq x \leq x_2\}\,, \tag{2.1}$$

onde y também pertence ao intervalo $[y_1\ ,\ y_2]$, de tal forma que $f(x) \subseteq \mathbf{Y}$, ou seja, $y_1 \leq f(x) \leq y_2$. Logo, pode-se definir uma função intervalar F associada a f pela transformação do intervalo $[x_1\ ,\ x_2]$ em $[y_1\ ,\ y_2]$, como $f(x) \subseteq F(\mathbf{X}) = \mathbf{Y}$.

A função F é dita extensão intervalar de f, e deve possuir a mínima diferença da imagem $f(\mathbf{X})$, sendo que o erro obtido no cálculo de $f(x)$ a partir do intervalo de $\mathbf{X}$ é calculado por intermédio do diâmetro $w(F(\mathbf{X})) = y_2 - y_1$.

Por fim, é oportuno lembrar que imagem intervalar de uma função e avaliação intervalar são distintas.

A imagem intervalar de uma função f, contínua no intervalo $\mathbf{X}$, é definida como o intervalo limitado pelo mínimo da imagem de $f(x)$ e pelo máximo da imagem de $f(x)$, sendo x um elemento do intervalo $\mathbf{X}$ (OLIVEIRA; DIVERIO; CLAUDIO, 1997) tal que:

$$\mathbf{Im} = I(f, \mathbf{x}) = [min\{f(x)|x \in \mathbf{X}\}, max\{f(x)|x \in \mathbf{X}\}]. \tag{2.2}$$

A função $F : \mathbb{I}(\mathbb{R}) \to \mathbb{I}(\mathbb{R})$ é uma extensão intervalar de uma função $f : \mathbb{R} \to \mathbb{R}$, se para todo $x \in \mathbb{R}, F([x, x]) = [f(x), f(x)]$ (SANTIAGO; BEDREGAL; ACIÓLY, 2006).

2.1.1 Operações Aritméticas Básicas

Assim como na aritmética básica, a matemática intervalar trabalha sobre as variáveis por intermédio de operações elementares simples como a adição, subtração, multiplicação e divisão. Contudo, a diferença é que para a aritmética intervalar as variáveis são intervalos de números, e não apenas um único número como na matemática básica.

Sejam $\mathbf{A}$ e $\mathbf{B}$ dois intervalos pertencentes aos reais $\mathbb{R}$, e $*$ pertencente ao conjunto $\{+, -, \cdot, \div\}$, uma operação binária sobre o conjunto dos números reais, então,

$$\mathbf{A} * \mathbf{B} = \{a * b | a \in \mathbf{A}, b \in \mathbf{B}\}. \tag{2.3}$$

define as operações aritméticas sobre $\mathbb{IR}$, assumindo que $0 \notin \mathbf{B}$ para o caso da divisão.

- Soma intervalar:

  Sejam $\mathbf{A}$ e $\mathbf{B}$ dois intervalos pertencentes a $\mathbb{IR}$, onde $\mathbf{A} = [a, b]$ e $\mathbf{B} = [c, d]$, define-se a soma intervalar como:

  $$\mathbf{A} + \mathbf{B} = [a + c, b + d]. \tag{2.4}$$

  Exemplo: *Sejam os intervalos reais* $\boldsymbol{A}$ *e* $\boldsymbol{B}$*, sendo* $\boldsymbol{A}$ *= [1, 2] e* $\boldsymbol{B}$ *= [3, 4], tem-se a operação aritmética de adição intervalar* $\boldsymbol{A} + \boldsymbol{B} = [4, 6]$.

  As seguintes propriedades algébricas se aplicam a soma de intervalos: fechamento, associatividade, cancelamento, comutatividade e elemento neutro.

- Subtração intervalar:

  Sejam $\mathbf{A}$ e $\mathbf{B}$ dois intervalos pertencentes a $\mathbb{IR}$, onde $\mathbf{A} = [a, b]$ e $\mathbf{B} = [c, d]$, define-se a subtração intervalar como:

$$\mathbf{A} - \mathbf{B} = [a - d, b - c]\,. \tag{2.5}$$

  Exemplo: *Sejam os intervalos reais* $\boldsymbol{A}$ *e* $\boldsymbol{B}$*, sendo* $\boldsymbol{A}$ *= [-1, 2] e* $\boldsymbol{B}$ *= [-3, 4], tem-se a operação aritmética de subtração intervalar* $\boldsymbol{A} - \boldsymbol{B} = [-5, 5]$.

- Multiplicação intervalar:

  Sejam $\mathbf{A}$ e $\mathbf{B}$ dois intervalos pertencentes a $\mathbb{IR}$, onde $\mathbf{A} = [a, b]$ e $\mathbf{B} = [c, d]$, define-se a multiplicação intervalar como:

$$\mathbf{A} \cdot \mathbf{B} = [min\,\{a.c, a.d, b.c, b.d\}\,, max\,\{a.c, a.d, b.c, b.d\}]\,. \tag{2.6}$$

  Exemplo: *Sejam os intervalos reais* $\boldsymbol{A}$ *e* $\boldsymbol{B}$*, sendo* $\boldsymbol{A}$ *= [-1, 2] e* $\boldsymbol{B}$ *= [-3, 4], tem-se a operação aritmética de multiplicação intervalar* $\boldsymbol{A} \cdot \boldsymbol{B} = [-6, 8]$.

  As seguintes propriedades algébricas se aplicam a multiplicação de intervalos: fechamento, associatividade, comutatividade, elemento neutro e distributividade.

- Divisão intervalar:

  Sejam $\mathbf{A}$ e $\mathbf{B}$ dois intervalos pertencentes a $\mathbb{IR}$ onde $\mathbf{A} = [a, b]$ e $\mathbf{B} = [c, d]$, e $0 \notin \mathbf{B}$, define-se a divisão intervalar como:

$$\mathbf{A} \div \mathbf{B} = \left[min\left\{\frac{a}{c}, \frac{a}{d}, \frac{b}{c}, \frac{b}{d}\right\}, max\left\{\frac{a}{c}, \frac{a}{d}, \frac{b}{c}, \frac{b}{d}\right\}\right]. \tag{2.7}$$

  Exemplo: *Sejam os intervalos reais* $\boldsymbol{A}$ *e* $\boldsymbol{B}$*, sendo* $\boldsymbol{A}$ *= [-1, 2] e* $\boldsymbol{B}$ *= [-3, 4], tem-se a operação aritmética de divisão intervalar* $\boldsymbol{A} \div \boldsymbol{B} = [-\frac{2}{3}, \frac{1}{2}]$.

2.2 Métricas de Erros

Na aritmética de ponto flutuante, utilizada pelos sistemas digitais, números reais não são representados diretamente, mas por intermédio de um subconjunto dos números reais, chamado de número de máquina (RATSCHEK; ROKNE, 1988).

O erro é definido como a diferença entre o valor obtido ao usar a representação de máquina e o valor real, dito exato. A seguir são explicitadas algumas fontes de erros mais comuns:

- **Erros inerentes ao modelo e erros inerentes aos dados:** ambos são erros iniciais do problema, ocorrem quando os dados de entrada são incertos, por exemplo, observações de experimentos que usam instrumentos de precisão limitada para medir os dados que serão analisados computacionalmente;
- **Erros de arredondamento:** ocorre quando não é possível representar um determinado número em toda sua extensão em virtude do tamanho da palavra suportada pelo computador, uma vez que a precisão de computador é finita. Em vista disso, após uma operação ser realizada em ponto flutuante, o resultado é normalizado e pode ser necessário arredondar esse valor, caracterizando assim, o erro de arredondamento;
- **Erros de truncamento:** erros de truncamento são causados ao se truncar sequências infinitas de operações aritméticas, após um número finito de etapas. Tanto o erro de arredondamento quanto o de truncamento acontecem em virtude das limitações físicas do dispositivo de computação, mas podem ser tornadas cada vez menores através de uma computação rigorosa e exaustiva.

A matemática intervalar busca estimar e controlar esses erros automaticamente, aproximando um número real x por um intervalo $\mathbf{X} = [\underline{x}, \overline{x}]$ que o contenha, ao invés de aproximá-lo por um número de máquina que sofre com os erros descritos acima. Contudo, é necessário avaliar a qualidade deste intervalo, quanto menor o seu diâmetro, englobando o numero real x, melhor o intervalo é. Para quantificar pode-se usar as medidas de Erro Absoluto e Erro Relativo, conforme as equações a seguir:

- **Erro Absoluto:** seja $m(x) = \frac{(\underline{x}+\overline{x})}{2}$ o ponto médio do intervalo $\mathbf{X}$, e $w(x) = \overline{x} - \underline{x}$ o diâmetro do intervalo, logo:

$$|x - m(x)| < \frac{w(x)}{2}. \tag{2.8}$$

- **Erro Relativo:** seja $m(x) = \frac{(\underline{x}+\overline{x})}{2}$ o ponto médio do intervalo $\mathbf{X}$, e $w(x) = \overline{x} - \underline{x}$ o diâmetro do intervalo, logo:

$$\left|\frac{x - m(x)}{x}\right| \leq \frac{w(x)}{2min|x|}, se\ 0 \notin x. \tag{2.9}$$

2.3 Integrais Numéricas

Dada a equação $I = \int_a^b f(x)dx$, a questão em análise é estimar integrais definidas em intervalos finitos de funções reais, isto é, calcular a área abaixo do gráfico da função f, no intervalo fechado de [a, b].

Três são as principais razões para que seja necessário este cálculo de aproximação, segundo Fernandes (1997):

- A primeira diz respeito aos casos em que a primitiva da função integrando não pode ser expressa em termos de funções elementares;
- A segunda diz respeito a funções integrando que possuem primitivas exatas, mas que devido a sua complexidade fazem com que uma aplicação numérica seja desejável;
- Por fim, pode ocorrer ainda da função integrando ser conhecida em alguns pontos somente, isto é, ela é conhecida para um conjunto discreto de pontos. Neste caso, não há alternativa a não ser o compto numérico da integral.

As técnicas usadas para a resolução de integrais são baseadas em interpolação. Ao invés de resolver de forma direta a equação $I = \int_a^b f(x)dx$, pontos $x_i, i = 0, ..., n$ pertencentes ao intervalo $[a, b]$, são aplicados à $f(x)$. Feito isso, um polinômio interpolador $p_n(x)$ é calculado baseado nos pontos $(x_i, f(x_i))$ e, por fim, calcula-se $\int_a^b p_n(x)dx$ para aproximar o valor de I.

Os métodos de integração, segundo Fernandes (1997), diferem entre si na forma como os pontos $x_i, i = 0, ..., n$ são selecionados, na quantidade de pontos e na maneira como estes são utilizados para se construir o polinômio interpolador. Alguns métodos de integração dividem o intervalo $[a, b]$ em subintervalos e aproximam, para cada subintervalo, um polinômio e ao final os soma. Outros métodos utilizam pontos igualmente espaçados.

Os estudos sobre integrais numéricas intervalares não são novos. Os principais métodos de integração utilizados com a aritmética intervalar são:

- Integral de Rall: (RALL, 1982).
- Integral de Bedegrall: (BEDREGAL; BEDREGAL, 2010).
- Integral de Simpson: (CAPRANI; MADSEN; NIELSEN, 2002).

2.4 Análise de Complexidade de Algoritmos

Um algoritmo é um procedimento que consiste em um conjunto de regras não ambíguas, as quais especificam, para cada entrada, uma sequência finita de operações, terminando com uma saída correspondente (TOSCANI; VELOSO, 2009).

Segundo Medida e Fertig (2006), algumas características são quesitos de qualidade e utilidade de um algoritmo, como por exemplo:

- **Reusabilidade**: propriedade de um algoritmo ou trechos deste, ser utilizados por outros algoritmos.
- **Legibilidade**: grau de facilidade de compreensão de um algoritmo.

- **Corretude**: o algoritmo deve desempenhar corretamente a função para o qual foi projetado.

- **Eficiência**: a propriedade de um algoritmo executar uma tarefa da melhor maneira.

A análise de um algoritmo trabalha principalmente sobre este último ponto, a eficiência, medindo o tempo de resposta do algoritmo. Esta análise do tempo de resposta pode ter duas abordagens: medir o tempo de execução ou expressar o tempo de execução em uma função ou expressão matemática (MEDIDA; FERTIG, 2006).

A primeira abordagem é empírica, pois depende de vários fatores, entre eles, configuração do computador, concorrência de processos, memória e processador. A segunda, por sua vez, é a área abrangida pela complexidade de algoritmos que trata da análise formal do desempenho do programa (MEDIDA; FERTIG, 2006).

Segundo Toscani e Veloso (2009), o termo complexidade de algoritmos diz respeito ao quanto este algoritmo demanda de recursos computacionais para executar as operações que lhe são propostas, gerando ao final uma saída correspondente, isto é, a quantidade de trabalho computacional despendido pelo algoritmo.

Os algoritmos podem ser classificados pelo tipo de expressão matemática que melhor define seu comportamento. Para isso, é feita a análise do comportamento assintótico da função do tempo "T" em relação ao tamanho de entrada "n". Os comportamentos assintóticos são: constante ($\mathcal{O}(1)$), logarítmico ($\mathcal{O}(\log_2 n)$), linear ($\mathcal{O}(\mathrm{n})$), logarítmo-linear ($\mathcal{O}(n \log_2 n)$), quadrática ($\mathcal{O}(n^2)$), cúbica ($\mathcal{O}(n^3)$) e exponencial ($\mathcal{O}(c^n)$), onde c é constante.

Sendo assim, quanto mais acentuado for o comportamento assintótico mais tempo ele demorará em relação aos outros para resolver uma entrada "n" de dados, ou seja, ele necessita de mais passos para resolver o mesmo problema. O algoritmo será razoável ou não será razoável, dependendo de quantos passos computacionais ele necessitará para chegar a solução de um problema e se ele obtém a solução de um problema em tempo polinomial (KREINOVICH et al., 1998). Além disso, se este algoritmo de tempo polinomial resolver todas as instâncias de um problema, então, este problema é dito tratável, caso contrário, diz-se que é intratável (TOSCANI; VELOSO, 2001).

Por fim, cabe ressaltar que análise da complexidade de um algoritmo é de suma importância, pois, embora os recursos computacionais tenham evoluído em virtude de novas tecnologias, e devido a isso o poder computacional para resolver determinados problemas tenha aumentado, eles não são infinitos e, por vezes, possuem um preço expressivo. A alternativa mais viável para se obter um melhor desempenho, com custos computacionais menores, ao resolver um problema é a criação de um algoritmo eficiente e de baixa complexidade.

2.5 IntPy: Pacote Intervalar na Linguagem Python

O IntPy (BARRETO; CAMPOS, 2008) é um *framework* para a matemática intervalar escrito como software livre usando a linguagem de programação Python (PYTHON, 2018). O fato do IntPy ter sido escrito nesta linguagem de programação, segundo Barreto e Campos (2008), se dá entre outras razões, pelo fato desta ser uma linguagem de cunho científico com sintaxe clara, construções apropriadas ao manejo matemático, multiplataforma e livre distribuição. Estas características fazem do IntPy uma ferramenta viável frente aos pacotes intervalares baseados em sistemas proprietários como por exemplo o INTLAB do MatLab (MATLAB, 2017).

A biblioteca IntPy foi desenvolvida por Barreto (2009), e suporta diversas operações com intervalos, bem como operações de conjunto como: continência, pertinência, união e intersecção. Além disso, possui os arredondamentos direcionados implementados de forma nativa, através do chaveamento dos modos de arredondamento do processador (BARRETO; CAMPOS, 2008).

O pacote IntPy é composto por 2 subpacotes e um módulo que encapsula as classes de excessão, chamado de *errors.py*.

O primeiro subpacote chamado de *support* contém as funcionalidades de suporte ao IntPy, e por sua vez é composto de 3 módulos *roundingmodule.c*, *general.py*, *stdfunc.py*. O *roundingmodule.c* é responsável por gerenciar os modos de arredondamento do processador. O *general.py*, por sua vez, organiza as peças pontuais de software pontuais, como por exemplo, a função *function2fraction*, responsável por converter um valor do tipo *string* (cadeia de caracteres alfanuméricos) de números racionais em outra representação que pode ser mais facilmente manipulável pelo computador.

O segundo subpacote, chamado de *ireal*, é composto por 2 módulos: *ireal.py* e *irmath.py*. O primeiro é responsável pela implementação da classe $\mathbb{I}\mathbb{R}$eal, que representa o tipo Intervalo Real, e de todas as demais operações e funções auxiliares. E o segundo módulo implementa extensões intervalares das funções.

Algumas extensões intervalares foram desenvolvidas por Varjão (2011), com o intuito de complementar o IntPy. As extensões intervalares desenvolvidas foram para funções exponenciais, logarítmicas, potenciação, raiz quadrada e trigonométricas.

Em síntese, este capítulo apresentou, inicialmente, a definição, principais conceitos e operações básicas da aritmética intervalar. Posteriormente, abordou-se o que é e como funciona a análise de complexidade de algoritmos, bem como sua importância para a computação. Por fim, foi apresentado o *framework* IntPy, sua arquitetura e suas funcionalidades.

3 *RELATIVE-DISTANCE-MEASURE (RDM) - INTERVAL ARITHMETIC*

Este capítulo aborda um novo conceito de aritmética intervalar, proposto por Piegat e Landowski (2012), que busca corrigir limitações existentes na aritmética intervalar convencional. Em seguida, são apresentadas suas características e suas operações básicas seguidas de exemplos para elucidá-las.

3.1 Aritmética Intervalar RDM

Segundo Piegat e Landowski (2012) a aritmética intervalar de Moore apresenta limitações evidentes tais como: problema de excesso de largura dos intervalos, dependência de variáveis e introdução de entropia negativa em sistemas (LANDOWSKI, 2015). Porém, ainda assim, é a aritmética mais utilizada no desenvolvimento de pesquisas científicas.

Com a finalidade de apresentar uma alternativa à aritmética de Moore, porém, sem as limitações inerentes a esta aritmética, Piegat e Landowski (2012) desenvolveram e apresentaram a aritmética intervalar multidimensional RDM, com base na teoria da incerteza apresentada por Liu (2010).

A aritmética RDM é multidimensional, pois entende que cada um dos parâmetros possui incertezas em seus dados, e ao passo que seja possível tratar cada intervalo como único, isto é, sem interferir nos demais, por intermédio de variável RDM "α", acrescenta-se uma nova dimensão ao problema. Por sua vez, a aritmética de Moore não leva em questão as peculiaridades dos parâmetros, mas sim apenas os limites do intervalo tão somente, o que a torna unidimensional.

Na aritmética RDM um valor real x pertencente ao intervalo $[x_1 \ , \ x_2]$ é descrito usando a variável RDM α_x, $\alpha_x \in [0 \ , \ 1]$, conforme a equação abaixo:

$$x = \underline{x} + \alpha_x \cdot (\overline{x} - \underline{x}), \tag{3.1}$$

e o intervalo $\mathbf{X} = [\underline{x}, \overline{x}]$ é descrito da seguinte forma:

$$\mathbf{X} = \{x : x = \underline{x} + \alpha_x \cdot (\overline{x} - \underline{x}), \alpha_x \in [0, 1]\}. \tag{3.2}$$

A variável RDM α_x, presente na equação, possibilita a obtenção de qualquer valor entre o limite da esquerda ($\underline{x}$) e o limite da direita ($\overline{x}$) do intervalo $\mathbf{X}$.

3.2 Operações Aritméticas Básicas

A aritmética RDM, assim como a aritmética de Moore, trabalha sobre as variáveis por intermédio de operações elementares simples como a adição, subtração, multiplicação e divisão. Contudo, a diferença reside no fato de que cada nova variável introduzida aumenta uma dimensão à solução do problema, tornando-o multidimensional.

A seguir são apresentadas as operações básicas da aritmética RDM (PIEGAT; LANDOWSKI, 2012)(PIEGAT; LANDOWSKI, 2013).

- Soma intervalar:

  Sejam **A** e **B** dois intervalos pertencentes aos reais, onde:

$$\mathbf{A} = \{a : a = \underline{a} + \alpha_a \cdot (\overline{a} - \underline{a}), \alpha_a \in [0,1]\} \tag{3.3}$$

  e

$$\mathbf{B} = \left\{b : b = \underline{b} + \alpha_b \cdot (\overline{b} - \underline{b}), \alpha_b \in [0,1]\right\}, \tag{3.4}$$

  então:

$$\mathbf{A} + \mathbf{B} = \left\{a + b : a + b = \underline{a} + \alpha_a \cdot (\overline{a} - \underline{a}) + \underline{b} + \alpha_b \cdot (\overline{b} - \underline{b}), \alpha_a, \alpha_b \in [0,1]\right\}. \tag{3.5}$$

  Exemplo: *Sejam os intervalos reais* ***A*** *e* ***B****, sendo*

$$\mathbf{A} = [1,2] = \{a : a = 1 + \alpha_a, \alpha_a \in [0,1]\} \tag{3.6}$$

  e

$$\mathbf{B} = [3,4] = \{b : b = 3 + \alpha_b, \alpha_b \in [0,1]\}, \tag{3.7}$$

  tem-se a operação aritmética de adição intervalar

$$\mathbf{A} + \mathbf{B} = \{a + b : a + b = 4 + \alpha_a + \alpha_b, \alpha_a, \alpha_b \in [0,1]\}. \tag{3.8}$$

As seguintes propriedades algébricas se aplicam a soma de intervalos RDM: fechamento, associatividade, comutatividade, cancelamento elemento inverso e elemento neutro.

- Subtração intervalar:

  Sejam **A** e **B** dois intervalos pertencentes aos reais, onde:

$$\mathbf{A} = \{a : a = \underline{a} + \alpha_a \cdot (\overline{a} - \underline{a}), \alpha_a \in [0,1]\} \tag{3.9}$$

  e

$$\mathbf{B} = \left\{b : b = \underline{b} + \alpha_b \cdot (\overline{b} - \underline{b}), \alpha_b \in [0,1]\right\}, \tag{3.10}$$

então:

$$\mathbf{A}-\mathbf{B}=\left\{a-b : a-b=\underline{a}+\alpha_a\cdot(\overline{a}-\underline{a})-\underline{b}-\alpha_b\cdot(\overline{b}-\underline{b}),\alpha_a,\alpha_b\in[0,1]\right\}. \quad (3.11)$$

Exemplo: *Sejam os intervalos reais* ***A*** *e* ***B****, sendo*

$$\mathbf{A}=[1,2]=\{a : a=1+\alpha_a,\alpha_a\in[0,1]\} \quad (3.12)$$

e

$$\mathbf{B}=[3,4]=\{b : b=3+\alpha_b,\alpha_b\in[0,1]\}, \quad (3.13)$$

tem-se a operação aritmética de subtração intervalar

$$\mathbf{A}-\mathbf{B}=\{a-b : a-b=-2+\alpha_a-\alpha_b,\alpha_a,\alpha_b\in[0,1]\}. \quad (3.14)$$

- Multiplicação intervalar:

  Sejam **A** e **B** dois intervalos pertencentes aos reais, onde:

$$\mathbf{A}=\{a : a=\underline{a}+\alpha_a\cdot(\overline{a}-\underline{a}),\alpha_a\in[0,1]\} \quad (3.15)$$

e

$$\mathbf{B}=\left\{b : b=\underline{b}+\alpha_b\cdot(\overline{b}-\underline{b}),\alpha_b\in[0,1]\right\}, \quad (3.16)$$

então:

$$\mathbf{A}\cdot\mathbf{B}=\left\{a\cdot b : a\cdot b=[\underline{a}+\alpha_a\cdot(\overline{a}-\underline{a})]\cdot[\underline{b}+\alpha_b\cdot(\overline{b}-\underline{b})],\alpha_a,\alpha_b\in[0,1]\right\}. \quad (3.17)$$

Exemplo: *Sejam os intervalos reais* ***A*** *e* ***B****, sendo*

$$\mathbf{A}=[1,2]=\{a : a=1+\alpha_a,\alpha_a\in[0,1]\} \quad (3.18)$$

e

$$\mathbf{B}=[3,4]=\{b : b=3+\alpha_b,\alpha_b\in[0,1]\}, \quad (3.19)$$

tem-se a operação aritmética de multiplicação intervalar

$$\mathbf{A}\cdot\mathbf{B}=\{a\cdot b : a\cdot b=3+3\alpha_a+\alpha_b+\alpha_a\cdot\alpha_b,\alpha_a,\alpha_b\in[0,1]\}. \quad (3.20)$$

As seguintes propriedades algébricas se aplicam a multiplicação de intervalos RDM: fechamento, associatividade, comutatividade, elemento inverso, distributividade e elemento neutro.

- Divisão intervalar:

  Sejam **A** e **B** dois intervalos pertencentes aos reais, onde:

$$\mathbf{A}=\{a : a=\underline{a}+\alpha_a\cdot(\overline{a}-\underline{a}),\alpha_a\in[0,1]\} \quad (3.21)$$

e

$$\mathbf{B} = \left\{ b : b = \underline{b} + \alpha_b \cdot (\overline{b} - \underline{b}), \alpha_b \in [0,1] \right\}, \tag{3.22}$$

então:

$$\mathbf{A} \div \mathbf{B} = \left\{ a \div b : a \div b = [\underline{a} + \alpha_a \cdot (\overline{a} - \underline{a})] \div [\underline{b} + \alpha_b \cdot (\overline{b} - \underline{b})], \alpha_a, \alpha_b \in [0,1] \right\}, se\, 0 \notin \mathbf{B}. \tag{3.23}$$

Exemplo: *Sejam os intervalos reais* ***A*** *e* ***B****, sendo:*

$$\mathbf{A} = [1,2] = \{aa = 1 + \alpha_a, \alpha_a \in [0,1]\} \tag{3.24}$$

e

$$\mathbf{B} = [3,4] = \{b : b = 3 + \alpha_b, \alpha_b \in [0,1]\}, \tag{3.25}$$

tem-se a operação aritmética de divisão intervalar

$$\mathbf{A} \div \mathbf{B} = \left\{ a \div b : a \div b = \frac{1 + \alpha_a}{3 + \alpha_b}, \alpha_a, \alpha_b \in [0,1] \right\}. \tag{3.26}$$

Apenas a propriedade algébrica da divisão pelo elemento inverso se aplica a divisão de intervalos RDM.

Para obter os intervalos resultantes das operações apresentadas o método é semelhante ao usado na aritmética de Moore, conforme as equações a seguir. E a Tabela 1 (LANDOWSKI, 2015) ilustra os possíveis resultados.

- Soma Intervalar:

$$s(\mathbf{A} + \mathbf{B}) = \begin{bmatrix} min(4 + \alpha_a + \alpha_b), & max(4 + \alpha_a + \alpha_b) \\ \alpha_a \in [0,1] & \alpha_a \in [0,1] \\ \alpha_b \in [0,1] & \alpha_b \in [0,1] \end{bmatrix} = [4,6] \tag{3.27}$$

- Subtração Intervalar:

$$s(\mathbf{A} - \mathbf{B}) = \begin{bmatrix} min(-2 + \alpha_a - \alpha_b), & max(-2 + \alpha_a - \alpha_b) \\ \alpha_a \in [0,1] & \alpha_a \in [0,1] \\ \alpha_b \in [0,1] & \alpha_b \in [0,1] \end{bmatrix} = [-3,-1] \tag{3.28}$$

- Multiplicação Intervalar:

$$s(\mathbf{A} \cdot \mathbf{B}) = \begin{bmatrix} min(3 + 3\alpha_a + \alpha_b + \alpha_a \cdot \alpha_b), & max(3 + 3\alpha_a + \alpha_b + \alpha_a \cdot \alpha_b) \\ \alpha_a \in [0,1] & \alpha_a \in [0,1] \\ \alpha_b \in [0,1] & \alpha_b \in [0,1] \end{bmatrix} = [3,8] \tag{3.29}$$

- Divisão Intervalar:

$$s(\mathbf{A} \div \mathbf{B}) = \begin{bmatrix} min(\frac{1+\alpha_a}{3+\alpha_b}), & max(\frac{1+\alpha_a}{3+\alpha_b}) \\ \alpha_a \in [0,1] & \alpha_a \in [0,1] \\ \alpha_b \in [0,1] & \alpha_b \in [0,1] \end{bmatrix} = [\frac{1}{4}, \frac{2}{3}] \tag{3.30}$$

Tabela 1 – Resultado das operações básicas RDM para os dois intervalos $\mathbf{A} = [1, 2]$ e $\mathbf{B} = [3, 4]$, usando valores de fronteira para as variáveis RDM, $\alpha_a \in [0, 1]$ e $\alpha_b \in [0, 1]$ (LANDOWSKI, 2015).

α_a a	0 1	0 1	1 2	1 2
α_b b	0 3	1 4	0 3	1 4
$a + b$	$\underline{a} + \underline{b}$ 4	$\underline{a} + \overline{b}$ 5	$\overline{a} + \underline{b}$ 5	$\overline{a} + \overline{b}$ 6
$a - b$	$\underline{a} - \underline{b}$ -2	$\underline{a} - \overline{b}$ -3	$\overline{a} - \underline{b}$ -1	$\overline{a} - \overline{b}$ -2
$a \cdot b$	$\underline{a} \cdot \underline{b}$ 3	$\underline{a} \cdot \overline{b}$ 4	$\overline{a} \cdot \underline{b}$ 6	$\overline{a} \cdot \overline{b}$ 8
$a \div b$	$\underline{a} \div \underline{b}$ $\frac{1}{3}$	$\underline{a} \div \overline{b}$ $\frac{1}{4}$	$\overline{a} \div \underline{b}$ $\frac{2}{3}$	$\overline{a} \div \overline{b}$ $\frac{1}{2}$

Na Figura 1, retirada do trabalho de Landowski (2015), são apresentados quatro gráficos tridimensionais que ilustram as operações básicas para os intervalos: $\mathbf{A} = [1, 2]$ e $\mathbf{B} = [3, 4]$ e a multidimensionalidade da aritmética RDM, onde dois eixos representam os intervalos e o terceiro eixo ilustra o resultado da operação executada entre estes intervalos, já a área hachurada representa o granulo das soluções possíveis.

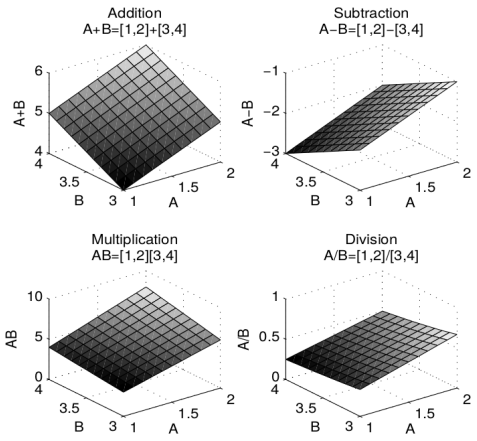

Figura 1 – Ilustração no plano para as operações de adição, subtração, multiplicação e divisão utilizando aritmética multidimensional RDM (LANDOWSKI, 2015).

Em síntese, este capítulo apresentou a aritmética intervalar RDM, mostrando o conceito, as operações matemáticas básicas definidas e as propriedades algébricas desta aritmética, além de apontar as diferenças entre a aritmética RDM e a aritmética de Moore.

4 TRABALHOS RELACIONADOS

Neste capítulo são apresentadas as produções textuais que serviram de alicerce para a construção do conhecimento sobre o assunto deste projeto, em virtude de estarem alinhados com a proposta deste trabalho. Em cada seção é apresentado uma breve síntese de cada produção, ou seja, são apresentados os principais conceitos e resultados que subsidiaram importantes contribuições para o desenvolvimento deste documento.

4.1 Is The Conventional Interval Arithmetic Correct?

O trabalho desenvolvido por Piegat e Landowski (2012) afirma que a aritmética convencional desenvolvida Moore (1966), conhecida também como aritmética de Moore, muito embora seja a mais frequentemente usada pelos pesquisadores, apresenta limitações como a resolução de equações e excesso de largura dos intervalos, por exemplo.

Como solução para estas e outras deficiências de aritmética de Moore, é apresentado um novo modelo aritmético, a *RDM interval arithmetic(Relative-Distance-Measure interval arithmetic)*. Além disso, são definidas as operações de soma e subtração para a aritmética RDM.

Piegat e Landowski (2012) também apresentam uma metodologia de teste para verificar se o resultado de qualquer aritmética intervalar está correta.

Por fim, os autores concluem que os parâmetros incertos e aproximados de um modelo de sistema aumentam sua dimensionalidade, aumentando as dificuldades de cálculo. Entretanto foi verificado que a aritmética RDM apresentou soluções corretas para estes problemas, enquanto não foi possível fazê-los corretamente com a aritmética de Moore. Concluem também que as operações aritméticas RDM apresentadas no artigo são o início de grandes investigações que podem levar a um novo intervalo multidimensional que permitirá resolver muitos problemas complicados contendo incerteza e dados aproximados.

4.2 Two Interpretations of Multidimensional RDM Interval Arithmetic - Multiplication and Division

Neste trabalho, Piegat e Landowski (2013) apresentam duas interpretações sobre a multiplicação e divisão de intervalos com a aritmética RDM, a possibilística e a probabilística. Além disso, eles definem estas operações intervalares para a aritmética RDM e demonstram a forma de realizar estas operações, finalizando a definição das operações básicas desta aritmética iniciado com os mesmos autores (PIEGAT; LANDOWSKI, 2012).

A interpretação possibilística, segundo os autores, é muito usada e de grande valor em sistemas cinzentos, computação de palavras e sistemas *fuzzy*. Outrossim, a visão possibilística é dita incondicional, pois trabalha com variáveis que não possuem parâmetros, como a função de densidade de probabilidade, conhecidos. Isto faz desta interpretação muito aceita, afirmam os autores, pois para os problemas reais, normalmente, não são conhecidos os parâmetros dos dados.

Por sua vez, a interpretação probabilística é de grande valia quando se tem o conhecimento da distribuição de densidade de probabilidade do problema e por isso é dita condicional. Entretanto, o cálculo usando esta interpretação só estará correto se as distribuições de probabilidade se mantiverem constantes nos intervalos.

Como conclusão, eles salientam que a versão possibilística é mais real e mais usada por ser incondicional. Isso deve-se ao fato de, nos problemas práticos, frequentemente, não existir um conhecimento aprofundado sobre distribuições de probabilidade para que sejam aplicadas a versão probabilística. Entretanto, os autores asseveram que estas duas formas de se realizar as operações de multiplicação e divisão de intervalos não são contraditórias, antes, são complementares. Isto é, ambas são corretas, no entanto, são úteis em situações específicas.

4.3 Differences Between Moore and RDM Interval Arithmetic

Neste trabalho de Landowski (2015), o objetivo é apresentar uma comparação entre a aritmética de intervalos de Moore e a aritmética de intervalos RDM. A motivação para isto, conforme o autor, é que a partir do desenvolvimento da Teoria da Incerteza a aritmética intervalar tem sido muito usada para determinar os dados incertos e modelagem de sistemas incertos.

Além disso, o autor sustenta que a aritmética intervalar mais usada, atualmente, é a de Moore. Contudo, foram encontradas limitações e desvantagens neste modelo, tais como: excesso de largura dos intervalos, problemas de dependência, problemas com resoluções de equações simples, entropia negativa e soluções absurdas.

Para realizar esta comparação, ele apresenta e resolve as equações para as operações básicas realizadas em ambos modelos aritméticos e propriedades da aritmética como: associatividade, comutatividade, distributividade, elemento neutro e lei de cancelamento, sempre ressaltando as diferenças entre os modelo aritméticos. Por exemplo, o autor ao demonstrar a adição do elemento inverso ou multiplicação pelo elemento inverso, define a propriedade para a aritmética RDM, como segue:

Na operação de adição, o intervalo inverso **-X**, na rotação RDM, é definido como:

$$\textbf{-X} = -[\underline{x}, \overline{x}] = \{-x\colon\ -x = -\underline{x} - \alpha_x \cdot (\overline{x} - \underline{x}), \alpha_x \in [0,1]\}\,, \tag{4.1}$$

por conseguinte, a adição do elemento inverso na aritmética RDM resulta em:

$$\mathbf{X} - \mathbf{X} = \{x - x\colon x - x = \underline{x} + \alpha_x \cdot (\overline{x} - \underline{x}) - \underline{x} - \alpha_x \cdot (\overline{x} - \underline{x}), \alpha_x \in [0,1]\} = 0. \tag{4.2}$$

E para operação de multiplicação o intervalo inverso $\frac{1}{X}$ é definido como:

$$\frac{1}{\mathbf{X}} = \left\{ \frac{1}{x}\colon \frac{1}{x} = \frac{1}{[\underline{x} + \alpha_x \cdot (\overline{x} - \underline{x})]}, \alpha_x \in [0,1] \right\}. \tag{4.3}$$

Logo, a multiplicação do intervalo $\mathbf{X}$ por seu inverso $\frac{1}{X}$ na aritmética RDM resulta em:

$$\frac{\mathbf{X}}{\mathbf{X}} = \left\{ \frac{x}{x}\colon \frac{x}{x} = \frac{[\underline{x} + \alpha_x \cdot (\overline{x} - \underline{x})]}{[\underline{x} + \alpha_x \cdot (\overline{x} - \underline{x})]}, \alpha_x \in [0,1] \right\} = 1. \tag{4.4}$$

Entretanto na aritmética intervalar de Moore o autor ressalta que não existe a propriedade de adição do elemento inverso, pois:

$$\mathbf{X} - \mathbf{X} = [\underline{x}, \overline{x}] - [\underline{x}, \overline{x}] = [\underline{x}, \overline{x}] + [-\overline{x}, -\underline{x}] = [\underline{x} - \overline{x}, \overline{x}, -\underline{x}], \tag{4.5}$$

logo esta propriedade na aritmética de Moore, retorna uma solução absurda. Além disso, a multiplicação pelo elemento inverso na aritmética de Moore, da mesma forma, não existe pois,

$$\frac{X}{X} = X \cdot \frac{1}{X} = [\underline{x}, \overline{x}] \cdot \left[\frac{1}{\overline{x}}, \frac{1}{\underline{x}}\right] = \begin{cases} [\underline{x}/\overline{x}, \overline{x}/\underline{x}] & \text{para} \quad \underline{x} > 0, \\ [\overline{x}/\underline{x}, \underline{x}/\overline{x}] & \text{para} \quad \overline{x} < 0. \end{cases} \tag{4.6}$$

Por fim, o autor reafirma as limitações do modelo aritmético de Moore que sempre fornece uma resposta unidimensional, sendo que esta solução, por vezes, depende da forma da equação, o que indica que problemas mais complexos não podem ser resolvidos pela aritmética convencional enquanto que para diferentes formas da equação, o modelo multidimensional RDM apresenta os mesmos resultados conforme demonstrado no exemplo a seguir.

Exemplo: *Consideremos os resultados da aritmética do intervalo Moore e RDM para equação não-linear C a seguir onde o intervalo* $\boldsymbol{A}$ *= [0, 2].*

$$C = A - A^2. \tag{4.7}$$

Esta equação pode ser escrita das seguintes formas também,

$$C = A \cdot (1 - A)\ e \tag{4.8}$$

$$C = (A - 1) + (1 - A) \cdot (1 + A). \tag{4.9}$$

Ao calcularmos o valor de C nestas três equações, para $\mathbf{A} = [0,2]$, usando a aritmética de Moore, obtemos:

$$C = [0,2] - ([0,2])^2 = [-4,2]. \tag{4.10}$$

$$C = [0,2] \cdot (1 - [0,2]) = [-2,2]. \tag{4.11}$$

$$C = ([0,2] - 1) + (1 - [0,2]) \cdot (1 + [0,2]) = [-4,4], \tag{4.12}$$

ou seja, existem três resultados distintos para expressões equivalentes. Qual resultado estaria correto?

Entretanto ao calcularmos o valor de C nestas três equações, para $\mathbf{A} = [0,2] = \{a : a = 2 \cdot \alpha_a, \alpha_a \in [0,1]\}$, usando a aritmética RDM, obtemos:

$$C = [2 \cdot \alpha_a] - ([2 \cdot \alpha_a])^2 = \left\{c\colon c = 2 \cdot \alpha_a - 4 \cdot \alpha_a^2, \alpha_a \in [0,1]\right\}. \tag{4.13}$$

$$C = [2 \cdot \alpha_a] \cdot (1 - 2 \cdot \alpha_a) = \left\{c\colon c = 2 \cdot \alpha_a - 4 \cdot \alpha_a^2, \alpha_a \in [0,1]\right\}. \tag{4.14}$$

$$C = ([2 \cdot \alpha_a] - 1) + (1 - [2 \cdot \alpha_a]) \cdot (1 + [2 \cdot \alpha_a]) = \left\{c\colon c = 2 \cdot \alpha_a - 4 \cdot \alpha_a^2, \alpha_a \in [0,1]\right\}. \tag{4.15}$$

A solução calculada pela aritmética RDM possui apenas uma variável RDM $\alpha_a \in [0,1]$ assim é 1-dimensional e tem a forma e o resultado a seguir:

$$C = \begin{bmatrix} min(2 \cdot \alpha_a - 4 \cdot \alpha_a^2), & max(2 \cdot \alpha_a - 4 \cdot \alpha_a^2) \\ \alpha_a \in [0,1] & \alpha_a \in [0,1] \end{bmatrix} = \left[-2, \frac{1}{4}\right]. \tag{4.16}$$

4.4 A New Approach to Diabetic Control - Human Glucose Metabolism Using Interval Arithmetic

O objetivo do trabalho de Tomaszewska (2015), foi aplicar a aritmética intervalar RDM a um problema biomédico, a diabetes. A justificativa para esta ação reside no fato de que este problema não é trivial, uma vez que os modelos são geralmente muito complexos, não-lineares e caracterizados por muitos parâmetros que devem ser identificados para cada paciente.

Estes parâmetros são bastante imprecisos e variáveis (TOMASZEWSKA, 2015), dependendo do físico de cada paciente e do tempo que a pessoa convive com esta doença,

além dos modelos matemáticos, que regem o funcionamento do metabolismo da glicose, possuírem valores iniciais como o conteúdo nutricional dos alimentos ingeridos, que só podem ser quantificados com um alto grau de incerteza.

A autora explica ainda que o modelo completo para o influxo de glicose exogênica no sangue consiste em dois submodelos: um para a concentração de carboidratos no estômago e um para a concentração de carboidratos no intestino, e que neste trabalho concentra-se apenas no modelo para a concentração de carboidratos no estômago, que são governadas pelas equações modelo de concentração c_c^S e de volume $V(t)$:

$$\frac{d}{dt}[c_c^S(t) \cdot V(t)] = -\alpha \cdot V \cdot c_c^s, \tag{4.17}$$

$$\frac{dV(t)}{dt} = -\alpha \cdot V + q^* = -q(t) + q^*. \tag{4.18}$$

Para analisar o modelo de concentração c_c^S de carboidratos no estômago e do volume $V(t)$, a autora fez algumas considerações com relação a dois parâmetros incertos neste modelo, o fator f de gastroparesia e a quantidade de carboidratos m_0^c. Ambos tem um grau de incerteza bastante elevado, o primeiro porque depende do físico individual dos pacientes e o segundo porque é bastante difícil determinar com precisão a quantidade de carboidratos nos alimentos ingeridos. E por esta razão estes parâmetros foram modelados na aritmética RDM, sendo definido como:

$$f_g = 0.6 + 0.1 \cdot \alpha_f, \alpha_f \in [0, 1], \tag{4.19}$$

$$m_0^c = 68 + 32 \cdot \alpha_c, \alpha_c \in [0, 1]. \tag{4.20}$$

Ambos fatores tem influência direta sobre a taxa de evacuação α do estômago. A autora então propôs, como mostra a Tabela 2, valores mínimos, médios e máximos para as variáveis RDM.

Tabela 2 – Possíveis Valores de α para diferente valores de variáveis RDM

α_f	0	0	0	0.5	0.5	0.5	1	1	1
α_c	0	0.5	1	0	0.5	1	0	0.5	1
α	0.04818	0.04812	0.04806	0.05220	0.05213	0.05206	0.05621	0.05614	0.05607

Tomaszewska (2015), identificou que valores diferentes do parâmetro α_f têm um impacto maior no valor da taxa de evacuação do estômago do que valores diferentes de α_c, além do α ter uma forte influência na concentração $c_c^S(t)$ de carboidratos no estômago que implica diretamente na diabetes.

A autora conclui que a aritmética intervalar RDM facilita os cálculos de equações complexas com variáveis incertas mostradas neste problema e fornece resultados satisfatórios. Além disso, acredita que este trabalho e a aritmética RDM dão base para realizar

mais pesquisas sobre o metabolismo humano da glicose, melhorando terapias e ajudando no desenvolvimento de uma medicação ideal para diabetes.

4.5 IntPy: Computação Científica Auto Validável em Python

O objetivo do trabalho de Varjão (2011) foi implementar funções específicas de extensão intervalar, como, cálculos de probabilidade intervalares para variáveis aleatórias contínuas, na linguagem de programação Python, usando como base a biblioteca intervalar IntPy desenvolvida por Barreto (2009).

Além disso, analisou o desempenho das rotinas implementadas com o pacote IntPy com as mesmas implementadas na biblioteca intervalar desenvolvida para a linguagem de programação Matlab, denominada Intlab.

Entre as funções implementadas por ele estão: potência do intervalo, logaritmo do intervalo e exponencial do intervalo, entre outras. Todas estas funções estão na classe stdfunc (*standart functions*).

Por fim, ele conclui que o IntPy obteve resultados satisfatórios quando comparado aos resultados encontrados no Intlab, em tempo de processamento e em precisão de máquina como pode ser observado na tabela abaixo extraída de Varjão (2011).

Tabela 3 – Intervalos Encapsuladores e Tempo de Processamento (VARJÃO, 2011)

Probabilidade	**Biblioteca**	**IP**	**TIp**
P($1 \leq x \geq 2$)	IntPy	[0.33333333333331, 0.33333333333337]	0.0005s
	Intlab	[0.33333333333333, 0.33333333333334]	0.0460s
P($\frac{1}{2} \leq x \geq \frac{4}{7}$)	IntPy	[0.02380952380952, 0.02380952380954]	0.0003s
	Intlab	[0.02380952380952, 0.02380952380953]	0.0630s

Na Tabela 3, o autor mostra que, para distribuições uniformes, o tempo de processamento (TIp) com os intervalos encapsuladores (Ip), foi menor com o pacote IntPy desenvolvido por ele do que com o pacote Intlab.

Foram elencados neste capítulo um breve resumo dos principais conceitos das produções textuais que contribuiram e serviram de auxílio teórico ao desenvolvimento deste trabalho.

5 DEFINIÇÃO DA EXPONENCIAL RDM

O presente capítulo tem por objetivo descrever a definição da exponencial natural intervalar na aritmética RDM desenvolvida neste trabalho. Para isso, é descrito, sucintamente, na seção 5.1 a definição da exponenciação para base real, bem como as propriedades que a caracterizam. Além disso, é apresentada na seção 5.2, a generalização da definição da exponencial real para a exponencial intervalar, momento no qual é apresentado o problema mátemático que este trabalho se propôs a resolver, bem como o conhecimento a ser aplicada e a solução proposta. Por fim, na seção 5.3 é apresentada a análise de complexidade da exponencial RDM definida.

5.1 Definição da Exponenciação com Base Real

Seja a um número real e n um número natural. Potência de base a e expoente n é o número a^n tal que (IEZZI; DOLCE; MURAKAMI, 1997):

$$\begin{cases} a^0 = 1 \\ a^n = a^{n-1} \cdot a, \forall\, n, n \geq 1. \end{cases} \tag{5.1}$$

Desta definição decorre as seguintes propriedades das potências:

P_1. $a^m \cdot a^n = a^{m+n}$

P_2. $\frac{a^m}{a^n} = a^{m-n}$

P_3. $(a \cdot b)^n = a^n \cdot b^n$

P_4. $\left(\frac{a}{b}\right)^n = \frac{a^n}{b^n} \quad b \neq 0$

P_5. $(a^m)^n = a^{m \cdot n}$

Se $a \in \mathbb{R}$,$b \in \mathbb{R}$, $m \in \mathbb{N}$ e $n \in \mathbb{N}$ então as propriedades anteriores são válidas para toda e qualquer potência.

Dado um número real a, tal que $0 < a \neq 1$, chama-se função exponencial de base a a função f de $\mathbb{R}$ em $\mathbb{R}$ que associa a cada x real o número a^x, cujas propriedades de P_1 à P_5 são mantidas(IEZZI; DOLCE; MURAKAMI, 1997).

5.2 Exponencial Intervalar

A aritmética intervalar de Moore resolve a exponenciação utilizando a extensão intervalar da potenciação real, isto é, para uma base $\mathbf{X}$, onde $\mathbf{X}$ é um intervalo, tal que

$\mathbf{X} = [\underline{x}, \overline{x}]$, e um expoente $\mathbf{Y}$, onde $\mathbf{Y}$ tambem é um intervalo, tal que $\mathbf{Y} = [\underline{y}, \overline{y}]$, têm-se que:

$$\mathbf{X}^{\mathbf{Y}} = [\underline{x}^{\underline{y}}, \overline{x}^{\overline{y}}] \tag{5.2}$$

Extendendo-se o conceito para aritmética intervalar RDM, é necessário lembrar que a principal diferença desta aritmética para a aritmética de Moore, é a representação de um intevalo $\mathbf{X}$ conforme a equação 3.2, logo a equação 5.2 pode ser reescrita da seguinte forma (PIEGAT; LANDOWSKI, 2012)(PIEGAT; LANDOWSKI, 2013):

Dado uma base $\mathbf{X}$, onde $\mathbf{X}$ é um intervalo RDM, tal que $\mathbf{X} = [\underline{x}+\alpha_x \cdot (\overline{x}-\underline{x}), \underline{x}+\alpha_x \cdot (\overline{x}-\underline{x})]$, com $\alpha_x \in [\alpha_{xi}, \alpha_{xf}]$, e um expoente $\mathbf{Y}$, onde $\mathbf{Y}$ também é um intervalo RDM, tal que $\mathbf{Y} = [\underline{y} + \alpha_y \cdot (\overline{y} - \underline{y}), \underline{y} + \alpha_y \cdot (\overline{y} - \underline{y})]$, com $\alpha_y \in [\alpha_{yi}, \alpha_{yf}]$, postulou-se neste trabalho a seguinte equação:

$$\mathbf{X}^{\mathbf{Y}} = [(\underline{x} + \alpha_{xi} \cdot (\overline{x} - \underline{x}))^{(\underline{y}+\alpha_{yi} \cdot (\overline{y}-\underline{y}))}, (\underline{x} + \alpha_{xf} \cdot (\overline{x} - \underline{x}))^{(\underline{y}+\alpha_{yf} \cdot (\overline{y}-\underline{y}))}] \tag{5.3}$$

e ao fazer $\alpha_x \in [\alpha_{xi}, \alpha_{xf}]$ e $\alpha_y \in [\alpha_{yi}, \alpha_{yf}]$, com $[\alpha_{xi}, \alpha_{xf}]$ e $[\alpha_{yi}, \alpha_{yf}]$ iguais a $[0, 1]$, obtém-se a equação exponencial da aritmética intervalar de Moore 5.4 que legitima o suposto na equação 5.3:

$$\mathbf{X}^{\mathbf{Y}} = [\underline{x}^{(\underline{y}+\alpha_{inicial} \cdot (\overline{y}-\underline{y}))}, \overline{x}^{(\underline{y}+\alpha_{final} \cdot (\overline{y}-\underline{y}))}] = [\underline{x}^{\underline{y}}, \overline{x}^{\overline{y}}] \tag{5.4}$$

Em termos da aritmética RDM, ainda, não se tem explicitamente definida uma forma para se determinar uma função na forma $e^{\mathbf{X}}$, onde $\mathbf{X}$, é um intervalo RDM. Segundo a equação **??** a operação de potência fica, aparentemente, definida de maneira correta, pois, no caso particular, de $\alpha \in [0, 1]$, o resultado retornado é extensão intervalar da exponenciação da aritmética Moore (MOORE, 1979).

Neste ponto, o que este trabalho propôs é encontrar uma maneira de definir a função exponencial de base natural na aritmética RDM, ou seja, $f(\mathbf{X}) = e^{\mathbf{X}}$.

Para aritmética de Moore a potência e soma de um intervalo $\mathbf{X}^n$ está bem definida (MOORE, 1979) (RATSCHEK; ROKNE, 1984), neste ponto pode-se supor que, se uma determinada função puder ser escrita em termos de somas e potências, fica, aparentemente, provado que, baseado no anteriormente visto, podemos sempre recair em um caso da arimética de Moore. Segundo **??**), a função e^x, $\forall x \in \mathbb{R}$, pode ser escrita em função de séries de potências de Taylor, na seguinte forma polinomial:

$$e^x = \sum_{k=0}^{k=n} \frac{x^k}{k!} = 1 + x + \frac{x^2}{2!} + ... + \frac{x^n}{n!} \tag{5.5}$$

Com isto, foi suposto que na forma intervalar pudessemos escrever $e^{\mathbf{X}}$ como sendo:

$$\mathbf{e^X} = \sum_{k=0}^{k=n} \frac{\mathbf{X}^k}{k!} = \mathbf{1} + \mathbf{X} + \frac{\mathbf{X}^2}{2!} + ... + \frac{\mathbf{X}^n}{n!} \tag{5.6}$$

onde $n \in \mathbb{N}$ e $\mathbf{1} = \mathbf{X}^0 = [1, 1]$, (MOORE, 1979) (RATSCHEK; ROKNE, 1984).

Segundo Piegat e Landowski (2012) na aritmética intervalar RDM, um intervalo pode ser escrito como sendo:

$$\mathbf{X} = [\underline{x} + \alpha_i \cdot (\overline{x} - \underline{x}),\ \underline{x} + \alpha_f \cdot (\overline{x} - \underline{x})]$$

com $\alpha \in [0, 1]$.

Desta forma, postulou-se, com intuito de se obter bons resultados, o seguinte:

$$\mathbf{e^X} = [\mathbf{e}^{\underline{x}+\alpha_i \cdot (\overline{x}-\underline{x})},\ \mathbf{e}^{\underline{x}+\alpha_f \cdot (\overline{x}-\underline{x})}] \tag{5.7}$$

onde $\underline{x}$ e $\overline{x} \in \mathbb{R}$, e α_i, $\alpha_f \in [0, 1]$.
Valendo-se também das propriedades operatórias, vistas na seção 5.1, em especial, da propriedade P_1 das funções exponenciais, têm-se:

$$\mathbf{e}^{\underline{x}+\alpha \cdot (\overline{x}-\underline{x})} = \mathbf{e}^{\underline{x}} \cdot \mathbf{e}^{\alpha \cdot (\overline{x}-\underline{x})} \tag{5.8}$$

que pode ser escrita em termos da série definida na equação 5.6:

$$\begin{gathered}\mathbf{e^X} = \sum_{k=0}^{k=n} \frac{x^k}{k!} = 1 + \frac{[\underline{x} + \alpha_i \cdot (\overline{x} - \underline{x}),\ \underline{x} + \alpha_f \cdot (\overline{x} - \underline{x})]}{1!} + \\ \frac{[\underline{x} + \alpha_i \cdot (\overline{x} - \underline{x}),\ \underline{x} + \alpha_f \cdot (\overline{x} - \underline{x})]^2}{2!} + ... + \frac{[\underline{x} + \alpha_i \cdot (\overline{x} - \underline{x}),\ \underline{x} + \alpha_f \cdot (\overline{x} - \underline{x})]^n}{n!}\end{gathered} \tag{5.9}$$

Existem diferentes bases para serem usadas na exponencial, a seguir é elencado em breves comentários, o porquê da utilização da Base Natural e não de outra base a qualquer:

- Para uma base a qualquer, caso se utilizasse a mesma estratégia adotada para a base e, incorreria-se na falta da definição do logarítmo natural intervalar RDM, uma vez que recairiamos na seguinte série de potências:

$$a^{\mathbf{X}} = \sum_{k=0}^{k=n} \frac{\ln^k(a)}{k!} = 1 + \mathbf{X} \cdot \ln(a) + \mathbf{X} \cdot \frac{(\ln(a))^2}{2!} + ... + \mathbf{X} \cdot \frac{(\ln(a))^n}{n!} \tag{5.10}$$

  sendo assim optou-se por definir $e^{\mathbf{X}}$ neste trabalho e propôr a extensão para a^X para um futuro projeto.

- Um número razoável das aplicações da aritmética intervalar faz uso da Base Natural em suas setenças, dentre elas podemos citar as distribuições de probabilidades utilizadas neste trabalho para testar a função definida, bem como a área de economia (KEARFOTT, 1996).

- Alguns trabalhos, ao empregarem a aritmética RDM, utilizam informalmente a aritmética de Moore para potência.

- Nos testes avaliados, a forma $\mathbf{\frac{1}{X}}$ retornou resultados corretos para equações exponenciais do tipo e^{-x}, pois, tomou-se a forma $\mathbf{e^{-x}}$ como sendo $\left[\frac{1}{e^{xi}}, \frac{1}{e^{xf}}\right]$.

5.3 Análise de Complexidade da Exponencial RDM

Nesta seção, apresenta-se a análise de complexidade da operação exponencial RDM definida neste trabalho. É importante ressaltar que para que fosse possível utilizar a aritmética RDM com os métodos de integração e distribuições, foi necessário implementar as operações de soma, subtração, multiplicação e divisão já definidas, de forma teórica, nos trabalhos de Piegat e Landowski (2012) e Piegat e Landowski (2013).

Ao analisarmos a função RDM percebemos que ela recebe como parâmetros de entrada o intervalo a ser utilizado, a precisão n que se deseja que a exponencial tenha e o α que é utilizado no intervalo RDM e que por padrão pertence ao intervalo [0, 1], podendo ser alterado diante da solução que o problema exija. A precisão n existe devido a definição intervalar da exponenciação RDM ser baseada em séries de potência de Taylor, conforme a equação 5.6, ou seja, 'n' é o número de termos da série a serem computados, e que quanto maior for mais termos são somados à série, o que aumenta a precisão da resposta final. Esta definição implica ser necessário um laço de n iterações que calcularia o fatorial de n números, o que na melhor hipótese seria $\mathcal{O}(1)$, com a função fatorial do pacote *math* da linguagem Python.

Foi definido neste trabalho uma função, chamada de "*precisao_taylor* "que recebe a precisão desejada para os resultados e o limite superior do intervalo utilizado e retorna o número de termos n da série de potências de Taylor que será necessário calcular para se alcançar aquela precisão.

Entretanto, após alguns testes simples, foi observado que a partir do 24º termo da série de Taylor calculado, não há alteração do resultado computacional devido a limitação de representação do computador. Sendo assim, foi decisão de projeto construir uma lista com o fatorial dos trinta primeiros números, para que ao invés de carcular o fatorial de um determinado elemento menor que trinta a cada iteração, fosse apenas acessado a lista e obtido o valor. Logo, o esforço computacional da função exponencial RDM é limitado pelo número de iterações necessárias, sendo assim a complexidade desta função é $\mathcal{O}(n)$.

Em síntese, este capítulo apresentou a definição da exponencial intervalar RDM feita neste trabalho, demonstrando o problema enfrentado, a metodologia para solucioná-lo e a solução do mesmo. E, por fim, foi realizada a análise da complexidade da exponencial RDM.

6 ANÁLISE DA QUALIDADE DOS INTERVALOS ENCAPSULADORES

Para atingir os objetivos propostos neste trabalho, após definir a operação de exponenciação para a aritmética RDM, e diante do comprometimento em testar esta nova operação com os métodos de integração, decidiu-se utilizar as distribuições de probabilidade como meio auxiliar de trabalho. Esta escolha pautou-se no fato que além destas distribuições possibilitarem a aplicação dos métodos de integração e utilizarem a operação de exponenciação, em sua grande maioria já foram definidas em sua forma intervalar.

O objetivo deste capítulo é analisar os intervalos encapsuladores obtidos ao aplicar a aritmética RDM para cada variável definida de forma intervalar. Para tanto serão computados exemplos aplicando três aritméticas: aritmética de ponto flutuante, aritmética intervalar de Moore e aritmética intervalar RDM.

As seções do presente capítulo estão divididas por distribuições de probabilidade. Em virtude das distribuições Exponencial, Normal e Gama, utilizarem a operação aritmética definida neste trabalho e, somado a isso, o fato delas estarem definidas na sua forma intervalar, fruto de outros trabalhos, procurou-se expô-las primeiro. Após elas é apresentado a distribuição Normal, que conquanto não faça uso da operação exponencial definida, possibilita verificar o desempenho da aritmética RDM e acrescenta conhecimento e completude a este trabalho.

Além disso, em cada uma destas seções em que este capítulo foi divido, são apresentadas as soluções de implementação com as três aritméticas já citadas anteriormente, utilizando os métodos de integração de Rall, de Bedregal e de Simpson, todas com 100.000 subdivisões. A finalidade é comparar os resultados obtidos em cada solução proposta e apresentar qual retorna um intervalo encapsulador que possua a melhor qualidade.

É exposto, ainda, com o objetivo de avaliar cada solução, o tempo de execução de cada solução, obtido através da média simples entre os tempos instantâneos de 10 execuções de cada algoritmo, ponto flutuante e intervalares. Além disso, foi determinada e apresentada, com o intuito de determinar a qualidade dos intervalos encapsuladores obtidos, a análise de erros verificada por intermédio das métricas de qualidade de intervalos descritas e explicadas na seção 2.2.

Não obstante existir variados ambientes computacionais para o desenvolvimento de algoritmos para cálculos numéricos que utilizam matemática intervalar na sua estrutura, a linguagem utilizada para codificar os algoritmos foi *Python* (PYTHON, 2018) em conjunto com o pacote IntPy (BARRETO, 2009). Os motivos para isso foram descritos

anteriormente na seção 2.5 e por ter demonstrado eficiência em relação as demais, retornando intervalos com melhor qualidade e com menor tempo de processamento (BALBONI et al., 2014).

Para todos os testes e resultados foi utilizado o mesmo computador com as seguintes especificações técnicas: processador Intel Core i7-2670QM 6M Cache,CPU @ 2.20GHz x 8; memória RAM 8 GB, DDR3 @ 1333 MHz; armazenamento HD Sata 1TB, modelo Toshiba MK1059GSM, 5400 RPM; sistema operacional Linux Ubuntu 16.04 LTS, Kernel versão 4.13.0.43.62 amd64. Além disso, todos os resultados intervalares e reais utilizaram o sistema de ponto flutuante F(10, 17, -17, 17).

6.1 Distribuição Exponencial

A função de densidade de probabilidade Exponencial normalmente está relacionada a determinação do tempo de vida útil de equipamentos eletrônicos, pois possui um parâmetro α, que pode ser interpretado da seguinte forma: $\frac{1}{\alpha}$ é o tempo de vida médio do objeto (ROCHA, 2014). No entanto, ela pode ter diversas outras aplicações em variadas áreas do conhecimento humano e, em particular, às variáveis hidrológicas (NAGHETTINI; PINTO, 2007).

Uma variável aleatória contínua assumindo valores não-negativos é dita seguir distribuição exponencial com parâmetro $\alpha > 0$, se sua função de densidade é dada por: (NAGHETTINI; PINTO, 2007)

$$f(x) = \begin{cases} \alpha e^{\alpha}, & \text{x} \geq 0 \\ 0, & \text{x} < 0 \end{cases} \tag{6.1}$$

A probabilidade de que um número $x \in (a, b)$ é:

$$P(a \leq x \leq b) = \alpha \int_a^b e^{-\alpha x} dx = e^{-a\alpha} - e^{-b\alpha} \tag{6.2}$$

Sendo a aplicação da integral a resolução de um número x estar compreendido no intervalo entre os valores a e b.

A função densidade de probabilidade da distribuição Exponencial, teve sua extensão intervalar definida por Santos (2010), utilizando o método de integração Simpson Intervalar, e é calculada da seguinte forma:

$$F_E(X) = \alpha[e^{-\alpha\overline{x}}, e^{-\alpha\underline{x}}], \underline{x} > 0 \tag{6.3}$$

e

$$G_E(X) = \alpha^5[e^{-\alpha\overline{x}}, e^{-\alpha\underline{x}}] \tag{6.4}$$

Para esta distribuição foram propostas três implementações, conforme mencionado anteriormente, ponto flutuante, a forma intervalar de Moore e a forma intervalar RDM, utilizando os métodos de integração de Rall, Bedregal e Simpson. Para a forma de ponto flutuante é sempre usado o método de integração 1/3 de Simpson como comparação, uma vez que os métodos de integração de Rall e Bedregal não contemplam os reais, mas sim os intervalares.

Para obtenção dos resultados foram passadas duas configurações de parâmetros:

- **Configuração 1:** intervalo A = [20.0, 50.0], $\alpha = 0.010$, n = 100.000 subintervalos
- **Configuração 2:** intervalo A = [15.0, 45.0], $\alpha = 0.005$, n = 100.000 subintervalos

6.1.1 Distribuição Exponencial Utilizando o Método de Integração de Rall

As tabelas 4 e 5 apresentam os resultados obtidos a partir da aplicação de cada configuração de parâmetros à distribuição Exponencial, utilizando Rall como método de integração. Além disso, mostra, também, o tempo que cada solução gastou para retornar cada resposta. As tabelas 6 e 7, por sua vez, expõem o diâmetro do intervalo encapsulador, o erro absoluto e o erro relativo de cada solução, servindo para análise dos resultados.

Tabela 4 – Resultados da aplicação da Configuração 1 à distribuição Exponencial

Aritmética	**Resultado**	**Tempo (s)**
Ponto Flutuante	0.2122000933655982	0.04
Intervalar Moore	[0.2121997750643845, 0.2122004116665235]	3.60
Intervalar RDM	[0.2121997750653166, 0.2122004116655970]	12.88

Tabela 5 – Resultados da aplicação da Configuração 2 à distribuição Exponencial

Aritmética	**Resultado**	**Tempo (s)**
Ponto Flutuante	0.1292272675692315	0.04
Intervalar Moore	[0.1292271706482232, 0.1292273644901790]	3.52
Intervalar RDM	[0.1292271706487510, 0.1292273644896515]	12.31

A partir da análise das tabelas 4 e 5 é possível afirmar que tanto o intervalo encapsulador obtido através do uso da aritmética intervalar de Moore quanto o intervalo encapsulador obtido através do uso da aritmética intervalar RDM contém a solução em ponto flutuante. Entretanto, foi observado que os resultados obtidos com o uso da aritmética RDM foram melhores, ou seja, mais estreitos, em média $1,456961248 \cdot 10^{-12}$ menores em relação a aritmética de Moore.

É possível inferir, ainda, que para cem mil subintervalos o tempo médio de execução para as soluções utilizando a aritmética RDM foi em torno de três vezes e meio maior que as soluções da aritmética de Moore. Tal fato, embora possa indicar uma desvantagem, era esperado tendo em vista que a distribuição Exponencial faz grande uso da operação

exponencial RDM definida neste trabalho, função esta que executa mais cálculos que a função exponencial de Moore, e que por vezes, traduz-se em intervalos mais exatos.

Tabela 6 – Métricas de erro para a configuração de parâmetros 1

Aritmética	Diâmetro	Erro Absoluto	Erro Relativo
Intervalar Moore	6.3660213905913920e-07	1.44162459747576e-13 < 3.18301069529569e-07	6.79370387925324e-13 ≤ 1.50000663022848e-06
Intervalar RDM	6.3660028046252931e-07	1.41331391034782e-13 < 3.18300140231264e-07	6.66028882425058e-13 ≤ 1.50000225086614e-06

Tabela 7 – Métricas de erro para a configuração de parâmetros 2

Aritmética	Diâmetro	Erro Absoluto	Erro Relativo
Intervalar Moore	1.9384195579230656e-07	3.03368441478824e-14 < 9.69209778961532e-08	2.34755750226088e-13 ≤ 7.50004642289874e-07
Intervalar RDM	1.9384090046981050e-07	3.01703106941886e-14 < 9.69204502349052e-08	2.33467063582578e-13 ≤ 7.50000559080119e-07

Observando as tabelas 6 e 7 é possível verificar que os valores dos diâmetros obtidos com o uso a aritmética RDM foram menores em relação àqueles obtidos com o uso da aritmética de Moore. Nos dois exemplos, ambas aritméticas apresentam diferença no diâmetro a partir da 12ª casa decimal.

Com relação ao erro absoluto, nos dois exemplos de configuração de parâmetros e nas duas aritméticas intervalares, constatou-se que a condição $|x - m(\mathbf{X})|$ é menor que a metade do diâmetro, $w(\mathbf{X})$, do intervalo. Diante disso é possível afirmar que o critério de estimativa para o erro é satisfeito em ambos os casos. Entretanto, foi constatado que em todos os cenários o erro absoluto acumulado foi menor com uso da aritmética RDM do que com o uso da aritmética de Moore.

Além disso, em relação ao erro relativo, é possível constatar que a desigualdade se manteve válida em todos os resultados obtidos, e assim como no erro absoluto, os algoritmos que usaram a aritmética RDM apresentam menores erros relativos, o que indica a qualidade do intervalo, isto é, os intervalos possuem boa qualidade de aproximação do valor em ponto flutuante.

6.1.2 Distribuição Exponencial Utilizando o Método de Integração de Bedregal

As tabelas 8 e 9 apresentam os resultados obtidos a partir da aplicação de cada configuração de parâmetros à distribuição Exponencial, utilizando Bedregal como método de integração e o tempo que cada solução gastou para retornar cada resposta. As tabelas 10 e 11 expõem, ainda, o diâmetro do intervalo encapsulador, o erro absoluto e o erro relativo de cada solução, servindo para análise dos resultados.

Tabela 8 – Resultados da aplicação da Configuração 1 à distribuição Exponencial

Aritmética	**Resultado**	**Tempo (s)**
Ponto Flutuante	0.2122000933655982	0.04
Intervalar Moore	[0.2121997750644319, 0.2122004116664778]	4.58
Intervalar RDM	[0.2121997750653114, 0.2122004116655919]	14.13

Tabela 9 – Resultados da aplicação da Configuração 2 à distribuição Exponencial

Aritmética	**Resultado**	**Tempo (s)**
Ponto Flutuante	0.1292272675692315	0.04
Intervalar Moore	[0.1292271706482469, 0.1292273644901569]	4.62
Intervalar RDM	[0.1292271706487519, 0.1292273644896542]	13.61

A partir da análise das tabelas 8 e 9 observa-se que tanto o intervalo encapsulador obtido através do uso da aritmética intervalar de Moore quanto o intervalo encapsulador obtido através do uso da aritmética intervalar RDM contém a solução em ponto fluatuante. Contudo, é possível verificar que os resultados obtidos com o uso da aritmética RDM foram melhores. Os algoritmos que usaram a aritmética RDM obtiveram um intervalo mais estreito, em média $1,38652978 \cdot 10^{-12}$ menor em relação a aritmética de Moore.

É possível observar, ainda, que para cem mil subintervalos o tempo médio de execução para as soluções utilizando a aritmética RDM foi em torno de três vezes maior que as soluções da aritmética de Moore. Tal comportamento era esperado assim como no método de integração anterior.

Tabela 10 – Métricas de erro para a configuração de parâmetros 1

Aritmética	**Diâmetro**	**Erro Absoluto**	**Erro Relativo**
Intervalar Moore	6.366020458559163e-07	1.43329792479107e-13 < 3.18301022927958e-07	6.75446415719364e-13 ≤ 1.50000641061617e-06
Intervalar RDM	6.366002805180404e-07	1.46493928099289e-13 < 3.18300140259020e-07	6.90357510102014e-13 ≤ 1.50000225099698e-06

Tabela 11 – Métricas de erro para a configuração de parâmetros 2

Aritmética	**Diâmetro**	**Erro Absoluto**	**Erro Relativo**
Intervalar Moore	1.9384190999560680e-07	2.95319324550291e-14 < 9.69209549978034e-08	2.28527098115789e-13 ≤ 7.50004465095190e-07
Intervalar RDM	1.9384090227392292e-07	2.84494650060196e-14 < 9.69204511369614e-08	2.20150634932974e-13 ≤ 7.50000566060505e-07

Observando as tabelas 10 e 11 é possível verificar que os valores dos diâmetros obtidos com o uso a aritmética RDM foram menores em relação àqueles obtidos com o uso da aritmética de Moore. Nos dois exemplos, ambas aritméticas apresentam diferença no diâmetro a partir da 12ª casa decimal.

Com relação ao erro absoluto, nos dois exemplos de configuração de parâmetros e nas duas aritméticas intervalares, constatou-se que a condição $|x - m(\mathbf{X})|$ é menor que a metade do diâmetro, $w(\mathbf{X})$, do intervalo. Diante disso é possível afirmar que o critério de estimativa para o erro é satisfeito em ambos os casos. Outrossim, foi constatado que na tabela 10 o erro absoluto acumulado foi maior com uso da aritmética RDM, já na tabela 11 a aritmética RDM retornou um erro absoluto menor.

Além disso, em relação ao erro relativo, é possível constatar que a desigualdade se manteve válida em todos os resultados obtidos. E, assim como no erro absoluto, os algoritmos que usaram a aritmética RDM apresentam maior e menor erro relativo acumulado nas tabelas 10 e 11 respectivamente, se comparados aos algoritmos que utilizaram a aritmética de Moore. Por se tratar de valores extremamente pequenos, indicam a qualidade do intervalo, isto é, demonstram que os intervalos possuem boa qualidade de aproximação do valor em ponto flutuante.

6.1.3 Distribuição Exponencial Utilizando o Método de Integração de Simpson

As tabelas 12 e 13 apresentam os resultados obtidos a partir da aplicação de cada configuração de parâmetros à distribuição Exponencial, utilizando o método Simpson de integração, além de apresentar o tempo gasto por cada solução para retornar cada resposta. As tabelas 14 e 15 apresentam o diâmetro do intervalo encapsulador, o erro absoluto e o erro relativo de cada solução, servindo para análise dos resultados.

Tabela 12 – Resultados da aplicação da Configuração 1 à distribuição Exponencial

Aritmética	**Resultado**	**Tempo (s)**
Ponto Flutuante	0.2122000933655982	0.04
Intervalar Moore	[0.2122000933653341, 0.2122000933670999]	6.09
Intervalar RDM	[0.2122000933653342, 0.2122000933670998]	37.89

Tabela 13 – Resultados da aplicação da Configuração 2 à distribuição Exponencial

Aritmética	**Resultado**	**Tempo (s)**
Ponto Flutuante	0.1292272675692315	0.04
Intervalar Moore	[0.12922726756900840, 0.1292272675700170]	6.10
Intervalar RDM	[0.12922726756900843, 0.1292272675700170]	37.16

A partir da análise das tabelas 12 e 13 observa-se, assim como nas subseções 6.1.1 e 6.1.2, que tanto o intervalo encapsulador obtido através do uso da aritmética intervalar de Moore quanto o intervalo encapsulador obtido através do uso da aritmética intervalar RDM contém a solução em ponto flutuante. Porém, foi observado que os resultados obtidos com o uso da aritmética RDM foram praticamente iguais ao obtidos com aritmética de Moore, sendo o resultado do algoritmo que faz uso da aritmética RDM, em média, $4,163336343 \cdot 10^{-17}$ menor que o de Moore.

É possível verificar, ainda, que para cem mil subintervalos o tempo médio de execução para as soluções utilizando a aritmética RDM foi em torno de seis vezes maior que as soluções da aritmética de Moore. Tal fato, embora possa indicar uma desvantagem, era esperado tendo em vista que a distribuição Exponencial faz grande uso da operação exponencial RDM definida neste trabalho.

Tabela 14 – Métricas de erro para a configuração de parâmetros 1

Aritmética	Diâmetro	Erro Absoluto	Erro Relativo
Intervalar Moore	1.7657264539394646e-12	6.18866069501677e-13 < 8.82863226969732e-13	2.91642694254350e-12 ≤ 4.16052233044851e-12
Intervalar RDM	1.7656709427882333e-12	6.18866069501677e-13 < 8.82835471394116e-13	2.91642694254350e-12 ≤ 4.16039153137498e-12

Tabela 15 – Métricas de erro para a configuração de parâmetros 2

Aritmética	Diâmetro	Erro Absoluto	Erro Relativo
Intervalar Moore	1.0086098622963390e-12	2.81219492137552e-13 < 5.04304931148169e-13	2.17616217867404e-12 ≤ 3.90246532821617e-12
Intervalar RDM	1.0085821067207235e-12	2.81219492137552e-13 < 5.04291053360361e-13	2.17616217867404e-12 ≤ 3.90235793766254e-12

Analisando as tabelas 14 e 15 é possível afirmar que os valores dos diâmetros obtidos com o uso a aritmética RDM foram ligeiramente menores em relação àqueles obtidos com o uso da aritmética de Moore. Nos dois exemplos, ambas aritméticas apresentam diferença a partir da 12ª casa decimal.

Com relação ao erro absoluto, nos dois exemplos de configuração de parâmetros e nas duas aritméticas intervalares, constatou-se que a condição $|x - m(\mathbf{X})|$ é menor que a metade do diâmetro, $w(\mathbf{X})$, do intervalo. Diante disso é possível afirmar que o critério de estimativa para o erro é satisfeito em ambos os casos. Foi verificado, ainda, em todos os cenários, que os erros absolutos acumulados além de serem extremamente pequenos, são semelhante para ambas aritméticas.

Além disso, em relação ao erro relativo, é possível constatar que a desigualdade se manteve válida em todos os resultados obtidos, e assim como no erro absoluto, ambos algoritmos retornaram valores semelhantes e com boa qualidade.

6.2 Distribuição Normal

A distribuição Normal, também conhecinhada como distribuição de Gauss, é geralmente utilizada para descrever o comportamento de uma variável aleatória que flutua de forma simétrica em torno de um valor central. Além disso, algumas de suas propriedades matemáticas fazem do modelo Normal a distribuição apropriada à modelagem de variáveis que resultam da soma de um grande número de outras variáveis independentes. (NAGHETTINI; PINTO, 2007)

Além disso, a distribuição Normal está na origem de toda a formulação teórica acerca da construção de intervalos de confiança, testes estatísticos de hipóteses, bem como da teoria de regressão e correlação. (NAGHETTINI; PINTO, 2007)

A variável aleatória X, que assume valores na reta, $-\infty < x < \infty$, tem distribuição Normal se sua função de densidade é da forma (ROCHA, 2014):

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \ , \ -\infty < x < \infty \tag{6.5}$$

onde $-\infty < \mu < \infty$ e $\sigma > 0$, e denota-se $X \sim N(\mu, \sigma^2)$.

A distribuição Normal, assim como a exponencial, teve sua extensão intervalar definida por Santos (2010), utilizando o método de Simpson, da seguinte forma:

$$F_N(X) = \frac{1}{\sqrt{2\pi}} \begin{cases} \left[e^{\frac{-\overline{x}^2}{2}}, e^{\frac{-\underline{x}^2}{2}}\right], & \text{se } \underline{x} > 0, \\ \left[e^{\frac{-\underline{x}^2}{2}}, e^{\frac{-\overline{x}^2}{2}}\right], & \text{se } \overline{x} < 0, \\ \left[e^{-max\left\{\frac{\underline{x}^2}{2}, \frac{\overline{x}^2}{2}\right\}}, 1\right], & \text{se } 0 \in x. \end{cases} \tag{6.6}$$

Foram propostas três implementações, conforme mencionado no capítulo 6, para esta distribuição.

Para obtenção dos resultados foram passadas duas configurações de parâmetro:

- **Configuração 1:** $\sigma = 1$, $\mu = 0$, intervalo A = [-0.90, -0.10], n = 100.000 subintervalos
- **Configuração 2:** $\sigma = 1$, $\mu = 0$, intervalo A = [-0.50, 0.25], n = 100.000 subintervalos

6.2.1 Distribuição Normal Utilizando o Método de Integração de Rall

As tabelas 16 e 17 apresentam os resultados obtidos a partir da aplicação de cada configuração de parâmetros à distribuição Normal, utilizando o método de Rall de integração, além de apresentar o tempo gasto por cada solução para retornar cada resposta. As tabelas 18 e 19 apresentam o diâmetro do intervalo encapsulador, o erro absoluto e o erro relativo de cada solução, servindo para análise dos resultados.

Tabela 16 – Resultados da aplicação da Configuração 1 à distribuição Normal

Aritmética	**Resultado**	**Tempo (s)**
Ponto Flutuante	0.2761120373761500	0.08
Intervalar Moore	[0.2761115139049955, 0.2761125608453779]	10.74
Intervalar RDM	[0.2761115139059979, 0.2761125608443796]	18.31

Tabela 17 – Resultados da aplicação da Configuração 2 à distribuição Normal

Aritmética	**Resultado**	**Tempo (s)**
Ponto Flutuante	0.2901687869569567	0.08
Intervalar Moore	[0.2901685651377956, 0.2901690087735161]	10.75
Intervalar RDM	[0.2901685651389673, 0.2901690087723468]	18.77

Ao observar as tabelas 16 e 17, pode-se concluir que tanto com a configuração de parâmetros 1 quanto com a configuração de parâmetros 2, as duas soluções obtiveram resultado satisfatório, pois ambas aritméticas contêm o valor em ponto fluatuante. No entanto, ao compararmos os dois algoritmos, mais especificamente as duas aritméticas

intervalares, constataremos que a solução que utilizou a aritmética RDM obteve um intervalo mais exato em relação a outra solução. Por exemplo, é possível verificar que já na 12ª casa decimal a aritmética RDM retorna um intervalo mais exato que a aritmética de Moore em ambos exemplos.

Foi verificado, ainda, que o algoritmo que utilizou a aritmética RDM, consumiu em média oito segundos a mais que o algoritmo com a aritmética de Moore. Este maior tempo de processamento, como já dito nas seções anteriores, era esperado.

Tabela 18 – Métricas de erro para a configuração de parâmetros 1

Aritmética	**Diâmetro**	**Erro Absoluto**	**Erro Relativo**
Intervalar Moore	1.0469403823876178e-06	9.63285007316017e-13 < 5.23470191193808e-07	3.488746874167 33e-12 ≤ 1.89586512996312e-06
Intervalar RDM	1.0469383816547050e-06	9.61231094720460e-13 < 5.23469190827352e-07	3.48130818147188e-12 ≤ 1.89586150690393e-06

Tabela 19 – Métricas de erro para a configuração de parâmetros 2

Aritmética	**Diâmetro**	**Erro Absoluto**	**Erro Relativo**
Intervalar Moore	4.4363572049288535e-07	1.30084831795329e-12 < 2.21817860246442e-07	4.48307459804855e-12 ≤ 7.64444832751285e-07
Intervalar RDM	4.4363337953212680e-07	1.29957156147497e-12 < 2.21816689766063e-07	4.47867455043589e-12 ≤ 7.64440798953639e-07

Em relação às tabelas 18 e 19, ao se comparar as métricas de erro para ambos exemplos e aritméticas, é possível ratificar os resultados obtidos nas tabelas 16 e 17. Por exemplo, o diâmetro obtido pelos algoritmos que utilizaram a aritmética RDM foram menores que os demais em ambos os exemplos, isto é, obtiveram maior exatidão que a soluções que utilizaram a aritmética de Moore.

Além disso, o erro absoluto e relativo com a aritmética RDM foram menores que a metade do diâmetro do intervalo nos dois exemplos. Somado-se a isso, estas métricas foram menores que o erro absoluto e relativo obtidos com a aritmética intervalar de Moore, satisfazendo assim o critério de qualidade do intervalo e sobrepondo os resultados da aritmética de Moore.

6.2.2 Distribuição Normal Utilizando o Método de Integração de Bedregal

As tabelas 20 e 21 apresentam os resultados obtidos a partir da aplicação de cada configuração de parâmetros à distribuição Normal, utilizando o método de Bedregal de integração, além de apresentar o tempo gasto por cada solução para retornar cada resposta. As tabelas 22 e 23 além de apresentarem o diâmetro do intervalo encapsulador, expõem o erro absoluto e o erro relativo de cada solução, servindo para análise dos resultados.

Tabela 20 – Resultados da aplicação da Configuração 1 à distribuição Normal

Aritmética	**Resultado**	**Tempo (s)**
Ponto Flutuante	0.2761120373761500	0.08
Intervalar Moore	[0.2761115139049678, 0.2761125608454055]	11.83
Intervalar RDM	[0.2761115139059972, 0.2761125608443797]	19.37

Tabela 21 – Resultados da aplicação da Configuração 2 à distribuição Normal

Aritmética	**Resultado**	**Tempo (s)**
Ponto Flutuante	0.6826882226961494	0.08
Intervalar Moore	[0.2901685651377990, 0.2901690087735067]	11.37
Intervalar RDM	[0.2901685651389701, 0.2901690087723478]	19.74

Ao observar as tabelas 20 e 21, pode-se afirmar que ambos algoritmos, em qualquer uma das configurações de parâmetros obtiveram resultados satisfatórios, pois ambas aritméticas contêm o valor em ponto flutuante. Contudo, ao compararmos os dois algoritmos, conclui-se que a solução que utilizou a aritmética RDM obteve um intervalo mais exato em relação a outra solução, ainda que discreto. É possível, por exemplo, verificar tanto na tabela 20 quanto na tabela 21 que já na 12ª casa decimal a aritmética RDM retornou um intervalo mais exato que a aritmética de Moore.

É possível inferir, ainda, que o algoritmo que utilizou a aritmética RDM, consumiu em torno de oito segundos e meio de tempo a mais que o algoritmo com a aritmética de Moore. Tal fato era esperado pelas razões já mencionadas em seções anteriores.

Tabela 22 – Métricas de erro para a configuração de parâmetros 1

Aritmética	**Diâmetro**	**Erro Absoluto**	**Erro Relativo**
Intervalar Moore	1.046940437676724e-06	9.63340518467248e-13 < 5.23470218838362e-07	3.48894791991585e-12 ≤ 1.89586523008428e-06
Intervalar RDM	1.046938382542835e-06	9.61508650476616e-13 < 5.23469191271441e-07	3.48231341021450e-12 ≤ 1.89586150851231e-06

Tabela 23 – Métricas de erro para a configuração de parâmetros 2

Aritmética	**Diâmetro**	**Erro Absoluto**	**Erro Relativo**
Intervalar Moore	4.4363570778083170e-07	1.30379040896855e-12 < 2.21817853890415e-07	4.49321383819947e-12 ≤ 7.64444810846675e-07
Intervalar RDM	4.4363337764474764e-07	1.29768418233311e-12 < 2.21816688822373e-07	4.47217013222587e-12 ≤ 7.64440795701420e-07

De acordo com as tabelas 22 e 23, conclui-se que o diâmetro obtido pelos algoritmos que utilizaram a aritmética RDM foram menores que os demais em ambos os exemplos, isto é, obtiveram maior exatidão que a soluções que utilizaram a aritmética de Moore.

Outrossim, o erro absoluto e relativo com a aritmética RDM foram menores que a metade do diâmetro do intervalo nos dois exemplos. Somado-se a isso, estas métricas, obtidas com a aritmética RDM, foram menores que o erro absoluto e relativo obtidos com a aritmética intervalar convencional, satisfazendo assim o critério de qualidade do intervalo e superando os resultados da aritmética de Moore.

6.2.3 Distribuição Normal Utilizando o Método de Integração de Simpson

As tabelas 24 e 25 apresentam os resultados obtidos a partir da aplicação de cada configuração de parâmetros à distribuição Normal, utilizando o método de integração de Simpson Intervalar, além de apresentar o tempo gasto por cada solução para retornar cada resposta. As tabelas 26 e 27 apresentam o diâmetro do intervalo encapsulador, o erro absoluto e o erro relativo de cada solução, servindo para análise dos resultados.

Tabela 24 – Resultados da aplicação da Configuração 1 à distribuição Normal

Aritmética	**Resultado**	**Tempo (s)**
Ponto Flutuante	0.2761120373761500	0.08
Intervalar Moore	[0.27611203737548290, 0.2761120373775399]	6.33
Intervalar RDM	[0.27611203737548295, 0.2761120373775398]	48.53

Tabela 25 – Resultados da aplicação da Configuração 2 à distribuição Normal

Aritmética	**Resultado**	**Tempo (s)**
Ponto Flutuante	0.6826882226961494	0.08
Intervalar Moore	[0.29016878695557130, 0.2901687869578997]	6.27
Intervalar RDM	[0.29016878695557136, 0.2901687869578997]	47.77

De acordo com a análise das tabelas 24 e 25, observa-se que tanto o intervalo encapsulador obtido através do uso da aritmética intervalar de Moore quanto o intervalo encapsulador obtido através do uso da aritmética intervalar RDM contém a solução em ponto flutuante. No entanto, foi observado que os resultados obtidos com o uso da aritmética RDM foram praticamente iguais aos obtidos com aritmética de Moore, sendo o resultado do algoritmo que faz uso da aritmética RDM diferente do resultado da solução com Moore, apenas na 17ª e 16ª casa decimal.

É possível verificar, ainda, que para cem mil subintervalos o tempo médio de execução para as soluções utilizando a aritmética RDM foi em torno de sete vezes e meio maior que as soluções da aritmética de Moore. Este maior tempo condiz com o esperado, pelas razões já mencionandas em outras seções.

Tabela 26 – Métricas de erro para a configuração de parâmetros 1

Aritmética	**Diâmetro**	**Erro Absoluto**	**Erro Relativo**
Intervalar Moore	2.0570212200254900e-12	3.61433105666719e-13 < 1.02851061001274e-12	1.30900886865115e-12 ≤ 3.72497562869408e-12
Intervalar RDM	2.0569101977230275e-12	3.61433105666719e-13 < 1.02845509886151e-12	1.30900886865115e-12 ≤ 3.72477458294556e-12

Tabela 27 – Métricas de erro para a configuração de parâmetros 2

Aritmética	**Diâmetro**	**Erro Absoluto**	**Erro Relativo**
Intervalar Moore	2.3284707495463410e-12	2.21156426505331e-13 < 1.16423537477317e-12	7.62164769080201e-13 ≤ 4.01226950351289e-12
Intervalar RDM	2.3283597272438783e-12	2.21156426505331e-13 < 1.16417986362193e-12	7.62164769080201e-13 ≤ 4.01207819709495e-12

Com relação ao erro absoluto, nos dois exemplos de configuração de parâmetros e nas duas aritméticas intervalares, constatou-se que a condição $|x - m(\mathbf{X})|$ é menor que a metade do diâmetro, $w(\mathbf{X})$, do intervalo. Diante disso é possível afirmar que o critério de estimativa para o erro é satisfeito em ambos os casos. Foi verificado, ainda, que na tabela 26 os erros absolutos acumulados foram semelhantes em razão da desigualde não ser a mesma para ambas aritméticas, e na tabela 27, o erro absoluto obtido com a aritmética RDM foi menor quando comparado ao resultado da aritmética convencional.

Além disso, em relação ao erro relativo, é possível constatar que a desigualdade se manteve válida em todos os resultados obtidos, e assim como no erro absoluto, ambos algoritmos retornaram valores semelhantes na tabela 26 e a aritmética RDM retornou

um erro relativo menor que a aritmética de Moore na tabela 27, porém ambos resultados demonstram a boa qualidade dos intervalos.

6.3 Distribuição Gama

A distribuição Gama surge da função $\Gamma = \Gamma(x)$, e é utilizada, segundo Vilches (2011), em fenômenos limitados, como por exemplo, os intervalos de tempo de espera numa fila de banco ou para analisar o tempo de permanência de pacientes em um hospital.

A função de densidade de probabilidade Gama, de parâmetros $\lambda > 0$ e $\upsilon \in \mathbb{R}$, é definida por (VILCHES, 2011):

$$f(x) = \begin{cases} \frac{\lambda^{\upsilon}}{\Gamma(\upsilon)} e^{-\lambda x} x^{\upsilon-1}, & \text{x} > 0 \\ 0, & \text{outro caso.} \end{cases} \tag{6.7}$$

Ainda, segundo Vilches (2011), é possível notar que para $\upsilon = 1$ tem-se a densidade de probabilidade Exponencial. Se utilizar-se a definição da função Gama, obteremos:

$$\int_0^{+\infty} e^{-\lambda x} x^{\upsilon-1} dx = \frac{\Gamma(\upsilon)}{\lambda^{\upsilon}}. \tag{6.8}$$

A função densidade de probabilidade da distribuição Gama, teve sua extensão intervalar definida por Finger (2014), utilizando o método de integração de Simpson Intervalar, e é calculada da seguinte forma:

$$F_G(X) = \frac{\Gamma(\upsilon)}{\lambda \upsilon} \left(e^{[-\lambda \underline{x}, -\lambda \overline{x}]} \cdot \left[x^{\upsilon-1}, x^{-\upsilon-1} \right] \right). \tag{6.9}$$

Para a distribuição Gama, seguiu-se os mesmos métodos de implementação, análise e comparação apresentados nas distribuições anteriores.

Para obtenção dos resultados foram passadas duas configurações de parâmetro:

- **Configuração 1:** $\lambda = 0.05$, $\upsilon = 1.0$, intervalo A = [20.0, 40.0], n = 100.000 subintervalos
- **Configuração 2:** $\lambda = 0.01$, $\upsilon = 0.09$, intervalo A = [5.0, 15.0], n = 100.000 subintervalos

6.3.1 Distribuição Gama Utilizando o Método de Integração de Rall

As tabelas 28 e 29 apresentam os resultados obtidos a partir da aplicação de cada configuração de parâmetros à distribuição Gama, utilizando Rall como método de integração. Além disso, mostra, também, o tempo que cada solução gastou para retornar cada

resposta. As tabelas 30 e 31, por sua vez, expõem o diâmetro do intervalo encapsulador, o erro absoluto e o erro relativo de cada solução, servindo para análise dos resultados.

Tabela 28 – Resultados da aplicação da Configuração 1 à distribuição de Gama

Aritmética	**Resultado**	**Tempo (s)**
Ponto Flutuante	0.2325441579345146	0.05
Intervalar Moore	[0.2325429952150688, 0.2325453206589139]	4.50
Intervalar RDM	[0.2325429952162059, 0.2325453206577853]	14.20

Tabela 29 – Resultados da aplicação da Configuração 2 à distribuição de Gama

Aritmética	**Resultado**	**Tempo (s)**
Ponto Flutuante	0.0758096920543245	0.06
Intervalar Moore	[0.0758092356245852, 0.0758101484883289]	4.56
Intervalar RDM	[0.0758092356249574, 0.0758101484879601]	14.60

Ao observar as tabelas 28 e 29, é possível verificar que ambos algoritmos, em qualquer uma das configurações de parâmetros obtiveram resultados satisfatórios, pois ambas aritméticas contiveram o valor em ponto flutuante. Contudo, ao compararmos os dois algoritmos, mais especificamente as duas aritméticas, conclui-se que a solução que utilizou a aritmética RDM obteve um intervalo mais exato em relação a outra solução. Por exemplo, tanto na tabela 28 quanto na tabela 29, a partir da 12ª casa decimal, a aritmética RDM retorna um melhor intervalo que a aritmética de Moore.

É possível inferir, ainda, que o algoritmo que utilizou a aritmética RDM, gastou aproximadamente oito segundos a mais que o algoritmo com a aritmética de Moore. Este maior consumo de tempo é esperado pelas razões já mencionadas nas distribuições acima.

Tabela 30 – Métricas de erro para a configuração de parâmetros 1

Aritmética	**Diâmetro**	**Erro Absoluto**	**Erro Relativo**
Intervalar Moore	2.3254438451525417e-06	2.47679654563626e-12 < 1.16272192257627e-06	1.06508654856585e-11 ≤ 5.00002987189926e-06
Intervalar RDM	2.3254415794093930e-06	2.48107090428106e-12 < 1.16272078970469e-06	1.06692463329040e-11 ≤ 5.00002500020979e-06

Tabela 31 – Métricas de erro para a configuração de parâmetros 2

Aritmética	**Diâmetro**	**Erro Absoluto**	**Erro Relativo**
Intervalar Moore	9.1286374369337060e-07	2.13255801906342e-12 < 4.56431871846685e-07	2.81304139520215e-11 ≤ 6.02079506654018e-06
Intervalar RDM	9.1286300275827960e-07	2.13430662032720e-12 < 4.56431501379139e-07	2.81534796210197e-11 ≤ 6.02079017967141e-06

Em relação às tabelas 30 e 31, ao se comparar as métricas de erro para ambos exemplos e aritméticas, é possível ratificar os resultados obtidos nas tabelas 28 e 29. Por exemplo, o diâmetro obtido pelos algoritmos que utilizaram a aritmética RDM foram menores que os demais em ambos os exemplos, isto é, obtiveram maior exatidão que a soluções que utilizaram a aritmética de Moore.

Além disso, com relação ao erro absoluto, foi observado que o erro absoluto da aritmética RDM foi discretamente maior que o mesmo erra da aritmética de Moore,

entretanto ambos erros absolutos foram muito menores que a metade do diâmetro do intervalo nos dois exemplos, o que sugere a qualidade do intervalos.

Outrossim, em relação ao erro relativo, é possível constatar que a desigualdade se manteve válida em todos os resultados obtidos, e assim como no erro absoluto, ambos algoritmos retornaram valores semelhantes nas tabelas 30 e 31 sendo que a aritmética RDM retornou um erro relativo discretamente maior que a aritmética de Moore, contudo, os resultados refletem a boa qualidade dos intervalos obtidos.

6.3.2 Distribuição Gama Utilizando o Método de Integração de Bedregal

As tabelas 32 e 33 apresentam os resultados obtidos a partir da aplicação de cada configuração de parâmetros à distribuição Exponencial, utilizando Rall como método de integração. Além disso, mostra, também, o tempo que cada solução gastou para retornar cada resposta. As tabelas 34 e 35, por sua vez, expõem o diâmetro do intervalo encapsulador, o erro absoluto e o erro relativo de cada solução, servindo para análise dos resultados.

Tabela 32 – Resultados da aplicação da Configuração 1 à distribuição de Gama

Aritmética	**Resultado**	**Tempo (s)**
Ponto Flutuante	0.2325441579345146	0.05
Intervalar Moore	[0.2325429952150885, 0.2325453206589077]	5.89
Intervalar RDM	[0.2325429952162032, 0.2325453206577832]	15.79

Tabela 33 – Resultados da aplicação da Configuração 2 à distribuição de Gama

Aritmética	**Resultado**	**Tempo (s)**
Ponto Flutuante	0.0758096920543245	0.06
Intervalar Moore	[0.0758092356245835, 0.0758101484883360]	5.75
Intervalar RDM	[0.0758092356249570, 0.0758101484879613]	15.88

De acordo com a análise das tabelas 32 e 33, observa-se que tanto o intervalo encapsulador obtido através do uso da aritmética intervalar de Moore quanto o intervalo encapsulador obtido através do uso da aritmética intervalar RDM contêm a solução em ponto flutuante. Não obstante, foi observado que os resultados obtidos com o uso da aritmética RDM foram mais exatos que os resultados obtidos com o emprego da aritmética de Moore, sendo os resultados do algoritmos que fizeram uso da aritmética RDM melhores que os resultados das soluções com Moore, a partir da 12ª casa decimal na tabela 32 e a partir da 13ª casa decimal na tabela 33

É possível verificar, ainda, que para cem mil subintervalos o tempo médio de execução para as soluções utilizando a aritmética RDM foi em média 10 segundos mais lento que as soluções da aritmética de Moore, conforme esperado.

Tabela 34 – Métricas de erro para a configuração de parâmetros 1

Aritmética	Diâmetro	Erro Absoluto	Erro Relativo
Intervalar Moore	2.3254438192565896e-06	2.48354115051085e-12 < 1.16272190962829e-06	1.06798690303380e-11 ≤ 5.00002981621891e-06
Intervalar RDM	2.3254415800477712e-06	2.47865616920250e-12 < 1.16272079002388e-06	1.06588623477718e-11 ≤ 5.00002500158245e-06

Tabela 35 – Métricas de erro para a configuração de parâmetros 2

Aritmética	Diâmetro	Erro Absoluto	Erro Relativo
Intervalar Moore	9.1286375254739930e-07	2.13527806547375e-12 < 4.56431876273699e-07	2.81662938815743e-11 ≤ 6.02079512493709e-06
Intervalar RDM	9.1286300428483620e-07	2.13468132059801e-12 < 4.56431502142418e-07	2.81584222643765e-11 ≤ 6.02079018973985e-06

Com relação ao erro absoluto, nos dois exemplos de configuração de parâmetros e nas duas aritméticas intervalares, foi observado que a condição $|x - m(\mathbf{X})|$ é muito menor que a metade do diâmetro, $w(\mathbf{X})$, do intervalo. Diante disso é possível afirmar que o critério de estimativa para o erro é satisfeito em ambos os casos. Foi verificado, ainda, nas tabelas 34 e 35, que os erros absolutos acumulados foram semelhantes, muito embora, é possível notar que os algoritmos que fizeram uso da aritmética RDM apresenta erros absolutos menores que aquelas soluções que utilizaram a aritmética intervalar convencional.

Sobre o erro relativo, é possível constatar que a desigualdade se manteve válida em todos os resultados obtidos, e assim como no erro absoluto, ambos algoritmos retornaram valores semelhantes nas tabelas 34 e 35 sendo que aritmética RDM, novamente, retornou erros relativos menores que a aritmética de Moore, porém ambos resultados, em virtude de serem muito pequenos, transparecem a boa qualidade dos intervalos.

6.3.3 Distribuição Gama Utilizando o Método de Integração de Simpson

As tabelas 36 e 37 apresentam os resultados obtidos a partir da aplicação de cada configuração de parâmetros à distribuição Exponencial, utilizando Simpson como método de integração. Além disso, mostra, também, o tempo que cada solução gastou para retornar cada resposta. As tabelas 38 e 39, por sua vez, expõem o diâmetro do intervalo encapsulador, o erro absoluto e o erro relativo de cada solução, servindo para análise dos resultados.

Tabela 36 – Resultados da aplicação da Configuração 1 à distribuição de Gama

Aritmética	Resultado	Tempo (s)
Ponto Flutuante	0.2325441579345146	0.05
Intervalar Moore	[0.23254415793339230, 0.2325441579356321]	6.38
Intervalar RDM	[0.23254415793339236, 0.2325441579356320]	40.66

Tabela 37 – Resultados da aplicação da Configuração 2 à distribuição de Gama

Aritmética	Resultado	Tempo (s)
Ponto Flutuante	0.0758096920543245	0.06
Intervalar Moore	[0.07580969205398498, 0.07580969205473444]	6.22
Intervalar RDM	[0.07580969205398504, 0.07580969205473441]	40.58

De acordo com a análise das tabelas 36 e 37, observa-se que tanto o intervalo encapsulador obtido através do uso da aritmética intervalar de Moore quanto o intervalo encapsulador obtido através do uso da aritmética intervalar RDM contêm a solução em ponto flutuante. No entanto, foi observado que os resultados obtidos com o uso da aritmética RDM foram praticamente iguais aos obtidos com aritmética de Moore, na tabela 36 com mais exatidão apenas a partir da 16ª casa decimal. Porém, o segundo exemplo apresentou resultados absolutamente iguais para ambas aritméticas.

É possível observar, ainda, que para cem mil subintervalos o tempo médio de execução para as soluções que utilizaram a aritmética RDM foi em torno de sete vezes e meio maior que as soluções da aritmética de Moore, fato que era esperado pelas razões já mencionadas em seções anteriores.

Tabela 38 – Métricas de erro para a configuração de parâmetros 1

Aritmética	Diâmetro	Erro Absoluto	Erro Relativo
Intervalar Moore	2.2397916854544064e-12	2.38697950294408e-15 < 1.11989584272720e-12	1.02646289812030e-14 ≤ 4.81584165639618e-12
Intervalar RDM	2.2397084187275595e-12	2.38697950294408e-15 < 1.11985420936377e-12	1.02646289812030e-14 ≤ 4.815662622169763e-12

Tabela 39 – Métricas de erro para a configuração de parâmetros 2

Aritmética	Diâmetro	Erro Absoluto	Erro Relativo
Intervalar Moore	7.4945605277321190e-13	3.52079476684252e-14 < 3.74728026386605e-13	4.64425414671195e-13 ≤ 4.94300947852100e-12
Intervalar RDM	7.4937278604636500e-13	3.52218254562330e-14 < 3.74686393023182e-13	4.64608475536260e-13 ≤ 4.94246029592579e-12

Com relação ao erro absoluto, nos dois exemplos de configuração de parâmetros e nas duas aritméticas intervalares, constatou-se que a condição $|x - m(\mathbf{X})|$ é menor que a metade do diâmetro, $w(\mathbf{X})$, do intervalo. Diante disso é possível afirmar que o critério de estimativa para o erro é satisfeito em ambos os casos. Foi verificado, ainda, que na tabela 38 os erros absolutos acumulados foram semelhantes em razão da desigualde não ser a mesma para ambas aritméticas, e na tabela 39, o erro absoluto obtido com a aritmética RDM foi discretamente maior quando comparado ao resultado da aritmética intervalar convencional.

Além disso, em relação ao erro relativo, é possível constatar que a desigualdade se manteve válida em todos os resultados obtidos, e assim como no erro absoluto, ambos algoritmos retornaram valores semelhantes na tabela 38 e a aritmética RDM retornou um erro relativo pouco maior que a aritmética de Moore na tabela 39, porém ambos resultados demonstram e transparecem a boa qualidade dos intervalos obtidos, devido ao seu pequeno tamanho.

6.4 Distribuição de Pareto

Pareto é uma distribuição frequentemente utilizada em Economia no estudo da distribuição de renda de uma população (VILCHES, 2011). Recebeu esse nome devido

ao seu criador, o economista italiano Vilfredo Pareto, em 1987 que usou-a para descrever a distribuição da riqueza na Itália, onde uma minoria possuía a maior parte da riqueza e a maioria da população possuía uma pequena parte. Mais especificamente, ele reparou que 20% da população possuía 80% da riqueza, não somente na Itália, mas também em outros países.

Define-se a função de densidade de probabilidade da distribuição de Pareto, por (VILCHES, 2011):

$$f(x) = \begin{cases} \alpha\beta^{\alpha}, & \text{se x} \geq \beta \\ 0, & \text{se x} < \beta \end{cases} \tag{6.10}$$

onde $\alpha > 1$ e $\beta > 1$. Logo:

$$\int_{-\infty}^{+\infty} f(x)dx = \int_{-\infty}^{+\infty} \frac{\alpha\beta^{\alpha}}{x^{\alpha+1}}dx \tag{6.11}$$

A função densidade de probabilidade da distribuição de Pareto, assim como a distribuição de probabilidade Gama teve sua extensão intervalar definida por Finger (2014), utilizando o método de integração Simpson Intervalar.

Seja $X \sim P[a, b], a < b$,uma variável aleatória com distribuição de Pareto com parâmetros α e c (FINGER, 2014). Para computar o intervalo encapsulador da probabilidade intervalar para esta variável aleatória utiliza-se a seguinte fórmula (FINGER et al., 2012) :

$$F_P(X) = \alpha\frac{c^{\alpha}}{X^{\alpha+1}} = \alpha\left(\left[\frac{c^{\alpha}}{\overline{x}^{\alpha+1}}, \frac{c^{\alpha}}{\underline{x}^{\alpha+1}}\right]\right), \text{ para } 0 \in X = [\underline{x}, \overline{x}]. \tag{6.12}$$

Assim como as demais distribuições, a distribuição Gama seguiu os mesmos métodos de implementação, análise e comparação descritos no capítulo 6

Para obtenção dos resultados foram passadas duas configurações de parâmetro:

- **Configuração 1:** $\alpha = 0.75$, c = 1.0, intervalo A = [1.0, 2.0], n = 100.000 subintervalos
- **Configuração 2:** $\alpha = 0.50$, c = 1.0, intervalo A = [1.0, 2.0], n = 100.000 subintervalos

6.4.1 Distribuição de Pareto Utilizando o Método de Integração de Rall

As tabelas 40 e 41 apresentam os resultados obtidos a partir da aplicação de cada configuração de parâmetros à distribuição de Pareto, utilizando Rall como método de integração. Além disso, mostra, também, o tempo que cada solução gastou para

retornar cada resultado. As tabelas 42 e 43, por sua vez, exibem o diâmetro do intervalo encapsulador, o erro absoluto e o erro relativo de cada solução, servindo para análise dos resultados.

Tabela 40 – Resultados da aplicação da Configuração 1 à distribuição de Pareto

Aritmética	**Resultado**	**Tempo (s)**
Ponto Flutuante	0.4053964425008999	0.05
Intervalar Moore	[0.4053938073863892, 0.4053990776271515]	5.53
Intervalar RDM	[0.4053938073884250, 0.4053990776250840]	5.66

Tabela 41 – Resultados da aplicação da Configuração 2 à distribuição de Pareto

Aritmética	**Resultado**	**Tempo (s)**
Ponto Flutuante	0.2928932188148047	0.05
Intervalar Moore	[0.2928916026999985, 0.2928948349356789]	5.25
Intervalar RDM	[0.2928916027013140, 0.2928948349343586]	5.30

A partir da análise das tabelas 40 e 41 é possível afirmar que tanto o intervalo encapsulador obtido através do uso da aritmética intervalar de Moore quanto o intervalo encapsulador obtido através do uso da aritmética intervalar RDM contêm a solução em ponto flutuante. Entretanto, foi observado que os resultados obtidos com os algoritmos que fizeram uso da aritmética RDM foram melhores, ou seja, mais estreitos, apresentando diferenças significativas a partir da 9ª casa decimal em relação a aritmética de Moore na tabela 41, e na 12ª casa decimal na tabela 40.

A partir das tabelas 40 e 41 é possível verificar, ainda, que para cem mil subintervalos o tempo médio de execução para as soluções que utilizaram a aritmética RDM e aritmética de Moore foram muito próximos. Esta proximidade nos tempos médios indica que para situações onde não há operações envolvendo a exponenciação RDM definida neste trabalho, os algoritmos que utilizam a aritmética RDM, tendem a operar em tempos semelhantes àqueles que utilizam a aritmética intervalar convencional.

Tabela 42 – Métricas de erro para a configuração de parâmetros 1

Aritmética	**Diâmetro**	**Erro Absoluto**	**Erro Relativo**
Intervalar Moore	5.2702407623006490e-06	5.87052628731044e-12 < 2.63512038115032e-06	1.44809516607867e-11 ≤ 6.50014956602121e-06
Intervalar RDM	5.2702366589718610e-06	5.85459458690706e-12 < 2.63511832948593e-06	1.44416525976151e-11 ≤ 6.50014450507161e-06

Tabela 43 – Métricas de erro para a configuração de parâmetros 2

Aritmética	**Diâmetro**	**Erro Absoluto**	**Erro Relativo**
Intervalar Moore	3.2322356804392882e-06	3.03401748169562e-12 < 1.61611784021964e-06	1.03587836344344e-11 ≤ 5.51780189435814e-06
Intervalar RDM	3.2322330445477830e-06	3.03163050219268e-12 < 1.61611652227389e-06	1.03506339766424e-11 ≤ 5.51779739456026e-06

A partir da observação das tabelas 42 e 43, pode-se concluir que os valores dos diâmetros obtidos com o uso a aritmética RDM foram menores em relação àqueles obtidos com o uso da aritmética de Moore em ambas configurações de parâmetros. Nos dois

exemplos, ambas aritméticas apresentam diferenças no diâmetro, sendo o primeiro a partir da 11ª casa decimal e o segundo a partir da 12ª casa decimal.

Com relação ao erro absoluto, em ambas configurações de parâmetros e nas duas aritméticas intervalares, constatou-se que a condição $|x - m(\mathbf{X})|$ é menor que a metade do diâmetro, w($\mathbf{X}$), do intervalo. Diante disso é possível afirmar que o critério de estimativa para o erro é satisfeito em ambos os casos. Entretanto, foi constatado que em todos os cenários o erro absoluto acumulado foi menor com uso da aritmética RDM do que com o uso da aritmética de Moore.

Além disso, em relação ao erro relativo, é possível constatar que a desigualdade se manteve válida em todos os resultados obtidos, e assim como no erro absoluto, os algoritmos que usaram a aritmética RDM apresentam menores erros relativos, o que indica a boa qualidade destes intervalos.

6.4.2 Distribuição de Pareto Utilizando o Método de Integração de Bedregal

As tabelas 44 e 45 apresentam os resultados obtidos a partir da aplicação de cada configuração de parâmetros à distribuição de Pareto, utilizando Bedregal como método de integração. Além disso, mostra, também, o tempo que cada solução gastou para retornar cada resultado. As tabelas 46 e 47, por sua vez, exibem o diâmetro do intervalo encapsulador, o erro absoluto e o erro relativo de cada solução, servindo para análise dos resultados.

Tabela 44 – Resultados da aplicação da Configuração 1 à distribuição de Pareto

Aritmética	**Resultado**	**Tempo (s)**
Ponto Flutuante	0.4053964425008999	0.05
Intervalar Moore	[0.4053938073864714, 0.4053990776270318]	6.43
Intervalar RDM	[0.4053938073884285, 0.4053990776250860]	6.64

Tabela 45 – Resultados da aplicação da Configuração 2 à distribuição de Pareto

Aritmética	**Resultado**	**Tempo (s)**
Ponto Flutuante	0.2928932188148047	0.05
Intervalar Moore	[0.2928916026998845, 0.2928948349357735]	6.25
Intervalar RDM	[0.2928916027013141, 0.2928948349343593]	6.72

Ao se analisar as tabelas 44 e 45 pode-se constatar que tanto o intervalo encapsulador obtido através do uso da aritmética intervalar de Moore quanto o intervalo encapsulador obtido através do uso da aritmética intervalar RDM contêm o resultado em ponto flutuante. Contudo, foi observado que os intervalos encapsuladores obtidos com o algoritmos que fizeram uso da aritmética RDM foram melhores, ou seja, mais exatos, apresentando diferenças significativas a partir da 10ª casa decimal em relação a aritmética de Moore na tabela 45, e na 12ª casa decimal na tabela 44.

A partir das tabelas 44 e 45 é possível verificar, ainda, que para cem mil subintervalos o tempo médio de execução para as soluções que utilizaram a aritmética RDM e aritmética de Moore, novamente, foram muito próximos. Esta semelhança no tempo gasto por cada algoritmo era esperado pelos motivos já citados na seção anterior.

Tabela 46 – Métricas de erro para a configuração de parâmetros 1

Aritmética	**Diâmetro**	**Erro bsoluto**	**Erro Relativo**
Intervalar Moore	5.2702405603510805e-06	5.85176351819427e-12 < 2.63512028017554e-06	1.44346691403965e-11 ≤ 6.50014931694163e-06
Intervalar RDM	5.2702366575285710e-06	5.85737014446863e-12 < 2.63511832876428e-06	1.44484991243001e-11 ≤ 6.50014450329144e-06

Tabela 47 – Métricas de erro para a configuração de parâmetros 2

Aritmética	**Diâmetro**	**Erro Absoluto**	**Erro Relativo**
Intervalar Moore	3.2322358889391722e-06	3.02435854138138e-12 < 1.61611794446958e-06	1.03258059494155e-11 ≤ 5.51780225029381e-06
Intervalar RDM	3.2322330452694280e-06	3.03207459140253e-12 < 1.61611652263471e-06	1.03521501920455e-11 ≤ 5.51779739579219e-06

A partir da observação das tabelas 46 e 47, pode-se concluir que os valores dos diâmetros obtidos com o uso da aritmética RDM foram menores em relação àqueles obtidos com o uso da aritmética de Moore em ambas configurações de parâmetros. Nos dois exemplos, ambas aritméticas apresentam diferenças no diâmetro, sendo o primeiro a partir da 11ª casa decimal e o segundo a partir da 12ª casa decimal.

Com relação ao erro absoluto, em ambas configurações de parâmetros e nas duas aritméticas intervalares, foi verificado que a condição $|x - m(\mathbf{X})|$ é menor que a metade do diâmetro, w($\mathbf{X}$), do intervalo. Diante disso é possível afirmar que o critério de estimativa para o erro é satisfeito em ambos os casos. Entretanto, ao contrário do que foi observado na subseção 6.4.1, foi constatado que, em ambos exemplos, o erro absoluto acumulado foi um pouco menor com uso da aritmética de Moore do que com o uso da aritmética RDM, no entanto todos os erros absolutos são muito pequenos.

Além disso, ao analisar-se o erro relativo, é possível constatar que a desigualdade se manteve válida em todos os resultados obtidos. Não obstante, assim como no erro absoluto, os algoritmos que usaram a aritmética de Moore apresentaram erros relativos pouco menores que aqueles que utilizaram a aritmética RDM. Entretanto, os intervalos de ambas aritméticas satisfazem os critérios de estimativa de erros.

6.4.3 Distribuição de Pareto Utilizando o Método de Integração de Simpson

As tabelas 48 e 49 apresentam os resultados obtidos a partir da aplicação de cada configuração de parâmetros à distribuição de Pareto, utilizando Simpson como método de integração. Além disso, mostra, também, o tempo que cada solução gastou para retornar cada resultado. As tabelas 50 e 51, por sua vez, exibem o diâmetro do intervalo encapsulador, o erro absoluto e o erro relativo de cada solução, servindo para análise dos resultados.

Tabela 48 – Resultados da aplicação da Configuração 1 à distribuição de Pareto

Aritmética	**Resultado**	**Tempo (s)**
Ponto Flutuante	0.4053964425008999	0.05
Intervalar Moore	[0.4053964424981482, 0.40539644250204915]	6.95
Intervalar RDM	[0.4053964424981484, 0.40539644250204904]	12.58

Tabela 49 – Resultados da aplicação da Configuração 2 à distribuição de Pareto

Aritmética	**Resultado**	**Tempo (s)**
Ponto Flutuante	0.2928932188148047	0.05
Intervalar Moore	[0.2928932188131935, 0.29289321881603536]	7.06
Intervalar RDM	[0.2928932188131935, 0.29289321881603536]	12.84

A partir da análise das tabelas 48 e 49, ainda, pode-se constatar que, em ambos exemplos e aritméticas intervalares, os resultados intervalares encontrados contêm a solução em ponto flutuante, o que indica a exatidão das soluções intervalares. Entretanto, ao se observar a tabela 48 é possível verificar que o algoritmo que utilizou a aritmética RDM retornou um intervalo encapsulador discretamente melhor que o algoritmo que utilizou a aritmética de Moore. Por outro lado, ao observar a tabela 49, verifica-se que o resultado obtidos com o uso de ambas aritméticas são exatamente iguais.

A partir das tabelas 48 e 49 é possível verificar, ainda, que para cem mil subintervalos o tempo médio de execução para as soluções utilizando a aritmética RDM foi em média seis segundos maior do que a aritmética de Moore.

Tabela 50 – Métricas de erro para a configuração de parâmetros 1

Aritmética	**Diâmetro**	**Erro bsoluto**	**Erro Relativo**
Intervalar Moore	3.9008796193229500e-12	8.01192445720744e-13 < 1.95043980966147e-12	1.97631839287530e-12 ≤ 4.81119123207496e-12
Intervalar RDM	3.9006575747180250e-12	8.01192445720744e-13 < 1.95032878735901e-12	1.97631839287530e-12 ≤ 4.81091737100756e-12

Tabela 51 – Métricas de erro para a configuração de parâmetros 2

Aritmética	**Diâmetro**	**Erro Absoluto**	**Erro Relativo**
Intervalar Moore	2.8418378761330130e-12	1.90236715269520e-13 < 1.42091893806650e-12	6.49508773331507e-13 ≤ 4.85132070938373e-12
Intervalar RDM	2.8418378761330130e-12	1.90236715269520e-13 < 1.42091893806650e-12	6.49508773331507e-13 ≤ 4.85132070938373e-12

A partir da observação das tabelas 50 e 51, pode-se concluir que os valores dos diâmetros obtidos com o uso a aritmética RDM foram discretamente menores no primeiro exemplo e exatamente igual no segundo exemplo em relação àqueles obtidos com o uso da aritmética de Moore. No primeiro cenário a diferença embora exista é muito pequena, da ordem de $2,220446049 \cdot 10^{-16}$, o que demonstra o nível de exatidão dos intervalos resultantes de ambas aritméticas.

Com relação ao erro absoluto, em ambas configurações de parâmetros e nas duas aritméticas intervalares, foi verificado que a condição $|x - m(\mathbf{X})|$ é menor que a metade do diâmetro, $\mathrm{w}(\mathbf{X})$, do intervalo. Diante disso é possível afirmar que o critério de estimativa para o erro é satisfeito em ambos os casos. Além disso, é possível verificar que o erro absoluto de ambas aritméticas no primeiro exemplo é muito semelhante, não sendo igual

devido ao tamanho médio do diâmetro da aritmética RDM ser discretamente menor que a aritimética intervalar convencional. Contudo, no segundo exemplo, o erro absoluto é exatamente o mesmo em ambas aritméticas.

Além disso, ao analisar-se o erro relativo, é possível constatar que a desigualdade se manteve válida em todos os resultados obtidos. Não obstante, assim como no erro absoluto, a mesma dinâmica foi observada, pois, no primeiro exemplo o erro relativo entre as aritméticas intervalares é muito semelhante e no segundo exemplo são idênticos. Desta forma, embora semelhantes, os intervalos de ambas aritméticas satisfazem os critérios de estimativa de erros.

As tabelas 52, 53 e 54 apresentam um comparativo de todos os resultados obtidos por intermédio dos algoritmos que utilizaram tanto a aritmética RDM quanto a aritmética de Moore, até o momento. Entretanto, os resultados encontram-se dispostos por método de integração intervalar, ao invés de estarem agrupados por distribuição.

Igualmente, as tabelas 55, 56 e 57 expõem os diâmetros dos intervalos obtidos de cada configuração de parâmetros utilizada e em cada distribuição de probabilidade analisada, dispostos pelos métodos de integração utilizados.

Por sua vez, as tabelas 58, 59 e 60 apresentam as métricas de erro apresentadas no decorrer deste capítulo, entretanto, agora dispostos por métodos e integração com o objetivo de mostrar um comparativo entre as métricas de erro dos intervalos.

Tabela 52 – Comparativo dos resultados agrupados por métodos de integração intervalar

Método de integração - Rall

Exemplo	Distribução	Aritmética intervalar	Resultado	Tempo (s)
1	Exponencial	Moore	[0.2121997750643845, 0.2122004116665235]	3.60
1	Exponencial	RDM	[0.2121997750653166, 0.2122004116655970]	12.88
2	Exponencial	Moore	[0.1292271706482232, 0.1292273644901790]	3.52
2	Exponencial	RDM	[0.1292271706487510, 0.1292273644896515]	12.31
1	Normal	Moore	[0.2761115139049955, 0.2761125608453779]	10.74
1	Normal	RDM	[0.2761115139059979, 0.2761125608443796]	18.31
2	Normal	Moore	[0.2901685651377956, 0.2901690087735161]	10.75
2	Normal	RDM	[0.2901685651389673, 0.2901690087723468]	18.77
1	Gama	Moore	[0.2325429952150688, 0.2325453206589139]	4.50
1	Gama	RDM	[0.2325429952162059, 0.2325453206577853]	14.20
2	Gama	Moore	[0.0758092356245852, 0.0758101484883289]	4.56
2	Gama	RDM	[0.0758092356249574, 0.0758101484879601]	14.60
1	Pareto	Moore	[0.4053938073863892, 0.4053990776271515]	5.53
1	Pareto	RDM	[0.4053938073884250, 0.4053990776250840]	5.66
2	Pareto	Moore	[0.2928916026999985, 0.2928948349356789]	5.25
2	Pareto	RDM	[0.2928916027013140, 0.2928948349343586]	5.30

Ao analisar as tabelas 52, 53 e 54, é possível constatar que os algoritmos que utilizaram a aritmética RDM, retornaram resultados muito expressivos, quando comparados à aritmética de Moore. Fica explicito que, para os métodos de Rall e Bedregal, em todos os exemplos e distribuições a aritmética RDM retornou resultados mais exatos, isto é, intervalos menores que contêm o valor em ponto flutuante. Ainda, de acordo com a tabela

Tabela 53 – Comparativo dos resultados agrupados por métodos de integração intervalar

Método de integração - Bedregal

Exemplo	Distribução	Aritmética intervalar	Resultado	Tempo (s)
1	Exponencial	Moore	[0.2121997750644319, 0.2122004116664778]	4.58
1	Exponencial	RDM	[0.2121997750653114, 0.2122004116655919]	14.13
2	Exponencial	Moore	[0.1292271706482469, 0.1292273644901569]	4.62
2	Exponencial	RDM	[0.1292271706487519, 0.1292273644896542]	13.61
1	Normal	Moore	[0.2761115139049678, 0.2761125608454055]	11.83
1	Normal	RDM	[0.2761115139059972, 0.2761125608443797]	19.37
2	Normal	Moore	[0.2901685651377990, 0.2901690087735067]	11.37
2	Normal	RDM	[0.2901685651389701, 0.2901690087723478]	19.74
1	Gama	Moore	[0.2325429952150885, 0.2325453206589077]	5.89
1	Gama	RDM	[0.2325429952162032, 0.2325453206577832]	15.79
2	Gama	Moore	[0.0758092356245835, 0.0758101484883360]	5.75
2	Gama	RDM	[0.0758092356249570, 0.0758101484879613]	15.88
1	Pareto	Moore	[0.4053938073864714, 0.4053990776270318]	6.43
1	Pareto	RDM	[0.4053938073884285, 0.4053990776250860]	6.64
2	Pareto	Moore	[0.2928916026998845, 0.2928948349357735]	6.25
2	Pareto	RDM	[0.2928916027013141, 0.2928948349343593]	6.72

Tabela 54 – Comparativo dos resultados agrupados por métodos de integração intervalar

Método de integração - Simpson intervalar

Exemplo	Distribução	Aritmética intervalar	Resultado	Tempo (s)
1	Exponencial	Moore	[0.2122000933653341, 0.2122000933670999]	6.09
1	Exponencial	RDM	[0.2122000933653342, 0.2122000933670998]	37.89
2	Exponencial	Moore	[0.1292272675690084, 0.1292272675700170]	6.10
2	Exponencial	RDM	[0.1292272675690084, 0.1292272675700170]	37.16
1	Normal	Moore	[0.2761120373754829, 0.2761120373775399]	6.33
1	Normal	RDM	[0.2761120373754829, 0.2761120373775398]	48.53
2	Normal	Moore	[0.2901687869557130, 0.2901687869578997]	6.27
2	Normal	RDM	[0.2901687869557136, 0.2901687869578997]	47.77
1	Gama	Moore	[0.2325441579333923, 0.2325441579356321]	6.38
1	Gama	RDM	[0.2325441579333923, 0.2325441579356320]	40.66
2	Gama	Moore	[0.0758096920539849, 0.0758096920547344]	6.22
2	Gama	RDM	[0.0758096920539850, 0.0758096920547344]	40.58
1	Pareto	Moore	[0.4053964424981482, 0.4053964425020491]	6.95
1	Pareto	RDM	[0.4053964424981484, 0.4053964425020490]	12.58
2	Pareto	Moore	[0.2928932188131935, 0.2928932188160353]	7.06
2	Pareto	RDM	[0.2928932188131935, 0.2928932188160353]	12.84

54, verifica-se que no pior caso a aritmética RDM resultou em intervalos no mínimo iguais aos apresentados pelos algoritmos que utilizaram a aritmética intervalar convencional.

É possível verificar que os melhores resultados, ou seja, os intervalos mais exatos obtidos por ambas aritméticas são encontrados naqueles algoritmos que utilizaram o método de integração de Simpson, em qualquer uma das distribuições apresentadas.

Com relação as tabelas 52, 53 e 54, foi observado que para 100.000 subdivisões, a aritmética RDM foi mais lenta em comparação a aritmética de Moore em todas os exemplos, tendo sua maior variação nos algoritmos que utilizaram o método de integração de Simpson e a menor naqueles que utilizaram o método de integração de Bedregal.

A maior variação de tempo por distribuição foi verificada na Exponencial, fato esse já esperado, pois faz um largo uso da função exponencial RDM definida neste trabalho, função esta que, quando comparada com a mesma função na matemática intervalar de

Moore, demanda um tempo um pouco maior de processamento, devido a matemática aplicada e a quantidade de cálculos executados. Em contrapartida, a menor variação de tempo, entre as aritméticas, foi verificada na distribuição de Pareto, o que também era esperado, uma vez que ela não faz uso da operação de exponenciação RDM.

Tabela 55 – Diâmetro dos resultados agrupados por métodos de integração intervalar

Método de integração - Rall

Exemplo	Distribução	Aritmética	Diâmetro
1	Exponencial	Moore	6.3660213905913920e-07
1	Exponencial	RDM	6.3660028046252931e-07
2	Exponencial	Moore	1.9384195579230656e-07
2	Exponencial	RDM	1.9384090046981050e-07
1	Normal	Moore	1.0469403823876178e-06
1	Normal	RDM	1.0469383816547050e-06
2	Normal	Moore	4.4363572049288535e-07
2	Normal	RDM	4.4363337953212680e-07
1	Gama	Moore	2.3254438451525417e-06
1	Gama	RDM	2.3254415794093930e-06
2	Gama	Moore	9.1286374369337060e-07
2	Gama	RDM	9.1286300275827960e-07
1	Pareto	Moore	5.2702407623006490e-06
1	Pareto	RDM	5.2702366589718610e-06
2	Pareto	Moore	3.2322356804392882e-06
2	Pareto	RDM	3.2322330445477830e-06

Tabela 56 – Diâmetro dos resultados agrupados por métodos de integração intervalar

Método de integração - Bedregal

Exemplo	Distribução	Aritmética	Diâmetro
1	Exponencial	Moore	6.3660204585591630e-07
1	Exponencial	RDM	6.3660028051804040e-07
2	Exponencial	Moore	1.9384190999560680e-07
2	Exponencial	RDM	1.9384090227392292e-07
1	Normal	Moore	1.0469404376767244e-06
1	Normal	RDM	1.0469383825428835e-06
2	Normal	Moore	4.4363570778083170e-07
2	Normal	RDM	4.4363337764474764e-07
1	Gama	Moore	2.3254438192565896e-06
1	Gama	RDM	2.3254415800477712e-06
2	Gama	Moore	9.1286375254739930e-07
2	Gama	RDM	9.1286300428483620e-07
1	Pareto	Moore	5.2702405603510805e-06
1	Pareto	RDM	5.2702366575285710e-06
2	Pareto	Moore	3.2322358889391722e-06
2	Pareto	RDM	3.2322330452694280e-06

As tabelas 55, 56 e 57, reforçam as conclusões expressas pelas tabelas 52, 53 e 54. Ao se analisar as mesmas, é possivel observar que em todas as configurações de parâmentos utilizadas na distribuição Exponencial, Normal, Gama e Pareto, a aritmética RDM obteve um diâmentro menor que a aritmética de Moore, ou no mínimo igual como é o caso do exemplo 2, da distribuição de Pareto apresentada na tabela 57.

Tabela 57 – Diâmetro dos resultados agrupados por métodos de integração intervalar

Método de integração - Simpson

Exemplo	Distribução	Aritmética	Diâmetro
1	Exponencial	Moore	1.7657264539394646e-12
1	Exponencial	RDM	1.7656709427882333e-12
2	Exponencial	Moore	1.0086098622963390e-12
2	Exponencial	RDM	1.0085821067207235e-12
1	Normal	Moore	2.0570212200254900e-12
1	Normal	RDM	2.0569101977230275e-12
2	Normal	Moore	2.3284707495463410e-12
2	Normal	RDM	2.3283597272438783e-12
1	Gama	Moore	2.2397916854544064e-12
1	Gama	RDM	2.2397084187275595e-12
2	Gama	Moore	7.4945605277321190e-13
2	Gama	RDM	7.4937278604636500e-13
1	Pareto	Moore	3.9008796193229500e-12
1	Pareto	RDM	3.9006575747180250e-12
2	Pareto	Moore	2.8418378761330130e-12
2	Pareto	RDM	2.8418378761330130e-12

Tabela 58 – Métricas de erros agrupadas por métodos de integração intervalar

Método de integração - Rall

Exemplo	Distribução	Aritmética	Erro Absoluto	Erro Relativo
1	Exponencial	Moore	1.44162459747576e-13 < 3.18301069529569e-07	6.79370387925324e-13 ≤ 1.50000663022848e-06
1	Exponencial	RDM	1.41331391034782e-13 < 3.18300140231264e-07	6.66028882425058e-13 ≤ 1.50000225086614e-06
2	Exponencial	Moore	3.03368441478824e-14 < 9.69209778961532e-08	2.34755750226088e-13 ≤ 7.50004642289874e-07
2	Exponencial	RDM	3.01703106941886e-14 < 9.69204502349052e-08	2.33467063582578e-13 ≤ 7.50000559080119e-07
1	Normal	Moore	9.63285007316017e-13 < 5.23470191193808e-07	3.48874687416733e-12 ≤ 1.89586512996312e-06
1	Normal	RDM	9.61231094720460e-13 < 5.23469190827352e-07	3.48130818147188e-12 ≤ 1.89586150690393e-06
2	Normal	Moore	1.30084831795329e-12 < 2.21817860246442e-07	4.48307459804855e-12 ≤ 7.64444832751285e-07
2	Normal	RDM	1.29957156147497e-12 < 2.21816689766063e-07	4.47867455043589e-12 ≤ 7.64440798953639e-07
1	Gama	Moore	2.47679654563626e-12 < 1.16272192257627e-06	1.06508654856585e-11 ≤ 5.00002987189926e-06
1	Gama	RDM	2.48107090428106e-12 < 1.16272078970469e-06	1.06692463329040e-11 ≤ 5.00002500020979e-06
2	Gama	Moore	2.13255801906342e-12 < 4.56431871846685e-07	2.81304139520215e-11 ≤ 6.02079506654018e-06
2	Gama	RDM	2.13430662032720e-12 < 4.56431501379139e-07	2.81534796210197e-11 ≤ 6.02079017967141e-06
1	Pareto	Moore	5.87052628731044e-12 < 2.63512038115032e-06	1.44809516607867e-11 ≤ 6.50014956602121e-06
1	Pareto	RDM	5.85459458690706e-12 < 2.63511832948593e-06	1.44416525976151e-11 ≤ 6.50014450507161e-06
2	Pareto	Moore	3.03401748169562e-12 < 1.61611784021964e-06	1.03587836344344e-11 ≤ 5.51780189435814e-06
2	Pareto	RDM	3.03163050219268e-12 < 1.61611652227389e-06	1.03506339766424e-11 ≤ 5.51779739456026e-06

Tabela 59 – Métricas de erros agrupadas por métodos de integração intervalar

Método de integração - Bedregal

Exemplo	Distribução	Aritmética	Erro Absoluto	Erro Relativo
1	Exponencial	Moore	1.43329792479107e-13 < 3.18301022927958e-07	6.75446415719364e-13 ≤ 1.50000641061617e-06
1	Exponencial	RDM	1.46493928099289e-13 < 3.18300140259020e-07	6.90357510102014e-13 ≤ 1.50000225099698e-06
2	Exponencial	Moore	2.95319324550291e-14 < 9.69209549978034e-08	2.28527098115789e-13 ≤ 7.50004465095190e-07
2	Exponencial	RDM	2.84494650060196e-14 < 9.69204511369614e-08	2.20150634932974e-13 ≤ 7.50000566060505e-07
1	Normal	Moore	9.63340518467248e-13 < 5.23470218838362e-07	1.30900886865115e-12 ≤ 3.72497562869408e-12
1	Normal	RDM	9.61508650476616e-13 < 5.23469191271441e-07	1.30900886865115e-12 ≤ 3.72477458294556e-12
2	Normal	Moore	1.30379040896855e-12 < 2.21817853890415e-07	7.62164769080201e-13 ≤ 4.01226950351289e-12
2	Normal	RDM	1.29768418233311e-12 < 2.21816688822373e-07	7.62164769080201e-13 ≤ 4.01207819709495e-12
1	Gama	Moore	2.48354115051085e-12 < 1.16272190962829e-06	1.06798690303380e-11 ≤ 5.00002981621891e-06
1	Gama	RDM	2.47865616920250e-12 < 1.16272079002388e-06	1.06588623477718e-11 ≤ 5.00002500158245e-06
2	Gama	Moore	2.13527806547375e-12 < 4.56431876273699e-07	2.81662938815743e-11 ≤ 6.02079512493709e-06
2	Gama	RDM	2.13468132059801e-12 < 4.56431502142418e-07	2.81584222643765e-11 ≤ 6.02079018973985e-06
1	Pareto	Moore	5.85176351819427e-12 < 2.63512028017554e-06	1.44346691403965e-11 ≤ 6.50014931694163e-06
1	Pareto	RDM	5.85737014446863e-12 < 2.63511832876428e-06	1.44484991243001e-11 ≤ 6.50014450329144e-06
2	Pareto	Moore	3.02435854138138e-12 < 1.61611794446958e-06	1.03258059494155e-11 ≤ 5.51780225029381e-06
2	Pareto	RDM	3.03207459140253e-12 < 1.61611652263471e-06	1.03521501920455e-11 ≤ 5.51779739579219e-06

A partir da análise das métricas de erro, divididas por métodos de integração intervalar e apresentadas nas tabelas 58, 59 e 60, pode-se observar que em alguns momentos os

Tabela 60 – Métricas de erros agrupadas por métodos de integração intervalar

Método de integração - Simpson

Exemplo	Distribução	Aritmética	Erro Absoluto	Erro Relativo
1	Exponencial	Moore	6.18866069501677e-13 < 8.82863226969732e-13	2.91642694254350e-12 ≤ 4.16052233044851e-12
1	Exponencial	RDM	6.18866069501677e-13 < 8.82835471394116e-13	2.91642694254350e-12 ≤ 4.16039153137498e-12
2	Exponencial	Moore	2.81219492137552e-13 < 5.04304931148169e-13	2.17616217867404e-12 ≤ 3.90246532821617e-12
2	Exponencial	RDM	2.81219492137552e-13 < 5.04291053360361e-13	2.17616217867404e-12 ≤ 3.90235793766254e-12
1	Normal	Moore	9.63285007316017e-13 < 5.23470191193808e-07	3.48874687416733e-12 ≤ 1.89586512996312e-06
1	Normal	RDM	9.61231094720460e-13 < 5.23469190827352e-07	3.48130818147188e-12 ≤ 1.89586150690393e-06
2	Normal	Moore	1.30084831795329e-12 < 2.21817860246442e-07	4.48307459804855e-12 ≤ 7.64444832751285e-07
2	Normal	RDM	1.29957156147497e-12 < 2.21816689766063e-07	4.47867455043589e-12 ≤ 7.64440798953639e-07
1	Gama	Moore	2.38697950294408e-15 < 1.11989584272720e-12	1.02646289812030e-14 ≤ 4.81584165639618e-12
1	Gama	RDM	2.38697950294408e-15 < 1.11985420936377e-12	1.02646289812030e-14 ≤ 4.815662622169763e-12
2	Gama	Moore	3.52079476684252e-14 < 3.74728026386605e-13	4.64425414671195e-13 ≤ 4.94300947852100e-12
2	Gama	RDM	3.52218254562330e-14 < 3.74686393023182e-13	4.64608475536260e-13 ≤ 4.94246029592579e-12
1	Pareto	Moore	8.01192445720744e-13 < 1.95043980966147e-12	1.97631839287530e-12 ≤ 4.81119123207496e-12
1	Pareto	RDM	8.01192445720744e-13 < 1.95032878735901e-12	1.97631839287530e-12 ≤ 4.81091737100756e-12
2	Pareto	Moore	1.90236715269520e-13 < 1.42091893806650e-12	6.49508773331507e-13 ≤ 4.85132070938373e-12
2	Pareto	RDM	1.90236715269520e-13 < 1.42091893806650e-12	6.49508773331507e-13 ≤ 4.85132070938373e-12

erros abosluto e relativo encontrados pelos algoritmos que utilizaram a aritmética RDM foram menores que na aritmética de Moore, e em outras vezes um pouco maior, embora essa comparação não seja expressiva devida ao fato das desigualdes, normalmente, não serem iguais para ambas aritméticas, é fácil concluir que todos os resultados apresentados cumpriram as condições das métricas de erro, e assim demonstraram a sua boa qualidade.

Este capítulo apresentou os resultados obtidos ao empregar-se a aritmética RDM, juntamente com a função exponencial RDM definida neste trabalho, aos métodos de integração.

Após análise de todos os resultados, ficou evidente que quando se deseja trabalhar com as distribuições de probabilidade, independentemente de qual método de integração intervalar será usado e se deseja uma solução mais exata, a melhor opção é utilizar a aritmética RDM ao invés da aritmética de Moore. Pois, ainda que se use o método de Simpson intervalar, método que retornou soluções mais semelhantes entre as duas aritméticas, a aritmética RDM retorna no mínimo um resultado igual à aritmética de Moore. Por exemplo, no caso do exemplo 2, da distribuição de Pareto com o método de integração de Simpson intervalar em que os resultados foram iguais, a melhor alternativa é utilizar a aritmética de Moore devido ao menor tempo de execução. No entanto, como não se é possível saber que os resultados serão iguais antes da comparação entre ambas aritméticas, o mais seguro seria utilizar a aritmética RDM, pois de 8 exemplos em quatro distribuições de probabilidade, 7 exemplos retornaram resultados mais exatos com a aritmética RDM e 1 exemplo retornou o resultado igual para ambas aritméticas intervalares.

É notório que, conquanto a aritmética intervalar convencional tenha sido processada pelos algoritmos em um tempo um pouco menor que os algoritmos que utilizaram aritmética RDM, devido a aplicação da operação exponencial RDM, definida neste trabalho, os resultados de aritmética de Moore estão aquém em exatidão, confiabilidade e qualidade em relação os resultados da aritmética intervalar RDM.

7 CONSIDERAÇÕES FINAIS

Ao trabalhar com computação numérica, um fator importante é a exatidão dos resultados dos cálculos, isto é, o que se procura são resultados com o menor erro contido possível. Neste contexto surgiu a aritmética intervalar (MOORE, 1966) com o objetivo principal de realizar um controle automático de erros dos cálculos, retornando respostas com maior exatidão através de intervalos.

Entretanto, a arimética de Moore possui algumas limitações e problemas que, por vezes, não a permitem ter a máxima exatidão (LANDOWSKI, 2015) ou a tornam falha, como, por exemplo, largura excessiva dos intervalos e soluções absurdas ao resolver uma operação aritmética simples como a adição do elemento inverso.

Esse comportamento da aritmética de Moore, foi a motivação para que Piegat e Landowski (2012) apresentassem a definição de uma nova matemática intervalar, a chamada aritmética RDM. O modelo RDM, segundo o autor, corrige estes outros problemas e limitações da aritmética de Moore, retornando resultados mais exatos e confiáveis.

Dentro deste contexto, o presente trabalho definiu a operação de exponenciação para aritmética RDM, até então não encontrada na literatura, e aplicou a aritmética intervalar RDM a alguns métodos de integração intervalar, por intermédio das distribuições de probabilidade Exponencial, Normal, Gama e Pareto. Estas distribuições foram implementadas tanto na forma de ponto flutuante quanto nas formas intervalares de Moore e RDM. Além disso, analisou-se os resultados obtidos quanto a qualidade e exatidão dos mesmos.

Foi verificado que os algoritmos que fizeram uso da aritmética RDM, retornaram intervalos mais exatos, com o método de integração intervalar de Rall e Bedregal, quando comparados a aritmética de Moore. Assim como nos demais métodos de integração, aqueles algoritmos que utilizaram o método de Simpson Intervalar aliado à aritmética RDM retornaram resultados melhores, embora discretos, ou no mínimo iguais que os demais algoritmos com aritmética intervalar convencional.

Ficou evidente, também, que os algoritmos que utilizaram a aritmética intervalar de Moore resolveram as distribuições de probabilidade em tempo menor que aquelas soluções que empregaram a aritmética RDM. Este maior tempo deve-se ao fato da função exponencial RDM ser definida em função de uma expansão em séries de potências de Taylor. Este maior tempo pode-se, possívelmente, ser reduzido se aplicarmos a técnica de paralelização no cálculo da série de Taylor.

Sendo assim, fica claro que caso seja necessário trabalhar com distribuições de

probabilidade intervalares, independente se o método de integração for Rall, Bedregal, ou Simpson intervalar, utilizar a aritmética RDM é mais vantajoso. Ainda que tenha sido verificado que a aritmética RDM resolveu as distribuições em um tempo comedidamente maior que a aritmética de Moore, é evidente a maior exatidão dos resultados da aritmética RDM.

7.1 Trabalhos Futuros

Em vista da definição da operação exponencial RDM de base (e) e do emprego desta operação nas distribuições de probabilidade neste trabalho, surgem as seguintes indicações de trabalhos futuros:

- Definir a operação logarítmica, para a aritmética RDM;
- Após a definição da operação logarítmica RDM, extender a definição da função exponencial RDM para qualquer base;
- Realizar a aplicação de todas as operações aritméticas RDM desenvolvidas, em outros problemas que possibilitem o maior uso dos recursos da aritmética RDM como a variação do α.

REFERÊNCIAS

BALBONI, M. D. C. et al. Critérios para análise e escolha de ambientes intervalares. **Revista Jr de Iniciação Científica em Ciências Exatas e Engenharia**, Rio Grande - RS, n. 7, 2014. Citado na página 52.

BARRETO, R. M. **IntPy**. 2009. Disponível em: <http://pypi.python.org/pypi/IntPy/0.1.3>. Citado 6 vezes nas páginas 22, 23, 25, 31, 44 e 51.

BARRETO, R. M.; CAMPOS, M. A. Intpy: Um framework intervalar em python. **5º Encontro Regional de Matemática Aplicada e Computacional**, 2008. Citado na página 31.

BEDREGAL, B. R.; BEDREGAL, R. C. A generalization of the moore and yang integral approach. **Computing**, 2010. Citado na página 29.

CAPRANI, O.; MADSEN, K.; NIELSEN, H. B. Introduction to interval analysis. **IMM - Informatics and Mathematical Modelling**, 2002. Citado na página 29.

FERNANDES, E. M. da G. P. **Computação Numérica**. 2ª ed.. ed. [S.l.]: Reprografia e Publicações da Universidade do Minho, 1997. Pag. 227-229. Citado 2 vezes nas páginas 28 e 29.

FINGER, A. F. **Extensão intervalar para as variáveis aleatórias com distribuições Uniforme, Normal, Gama, Exponencial e Pareto**. Dissertação (Mestrado) — UNIVERSIDADE FEDERAL DE PELOTAS, 2014. Citado 2 vezes nas páginas 62 e 67.

FINGER, A. F. et al. A complexidade computacional dos problemas de computar intervalos encapsuladores para as variáveis aleatórias uniforme, exponencial e pareto. **CLEI**, 2012. Citado na página 67.

IEZZI, G.; DOLCE, O.; MURAKAMI, C. **Fundamentos da Matemática Elementar - Logarítmos**. 3. ed. Rua José Antônio Coelho, 785 - São Paulo: Editora Atual, 1997. v. 4. Citado na página 45.

KEARFOTT, R. B. Interval computations: Introduction, uses and resources. **Euromath Bulletin**, v. 2, n. 1, p. 95–112, 1996. Citado na página 48.

KREINOVICH, V. et al. **Computational Complexity and Feasibility of Data Processing and Interval Computations**. [S.l.]: Dordrecht: Kluwer, 1998. Citado na página 30.

LANDOWSKI, M. Differences between moore and rdm interval arithmetic. **Intelligent Systems**, v. 322, p. 331–340, 2015. Citado 8 vezes nas páginas 13, 21, 22, 33, 36, 37, 40 e 77.

LIU, B. Uncertainty theory. In: _____. **Uncertainty Theory: A Branch of Mathematics for Modeling Human Uncertainty**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. p. 1–79. ISBN 978-3-642-13959-8. Disponível em: <http://dx.doi.org/10.1007/978-3-642-13959-8_1>. Citado na página 33.

LORETO, A. B. **Análise da Complexidade Computacional de Problemas de Estatística Descritiva com Entradas Intervalares**. Tese (Doutorado) — Universidade Federal do Rio Grande do Sul, Porto Alegre, 2006. Citado na página 25.

MATLAB. 2017. Disponível em: <http://www.mathworks.com>. Citado na página 31.

MEDIDA, M.; FERTIG, C. **Algoritmos e programação: Teoria e prática**. [S.l.: s.n.], 2006. Citado 2 vezes nas páginas 29 e 30.

MOORE, R.; KEARFOTT, M.; CLOUD, J. **Introduction to Interval Analysis**. Philadelphia: Studies in Applied and Numerical Mathematics (SIAM), 2009. 213 p. Citado na página 25.

MOORE, R. E. **Interval Analysis**. Englewood Cliffs, NJ: Prentice Hall, 1966. 159 p. Citado 4 vezes nas páginas 21, 25, 39 e 77.

MOORE, R. E. **Methods and Applications of Interval Analysis**. 2. ed. Madison, Wisconsin: Studies in Applied and Numerical Mathematics (SIAM), 1979. 200 p. Citado 3 vezes nas páginas 25, 46 e 47.

NAGHETTINI, M.; PINTO, E. **Hidrologia Estatística**. [S.l.]: CPRM Serviço Geológico do Brasil, 2007. Citado 2 vezes nas páginas 52 e 57.

OLIVEIRA, P.; DIVERIO, T.; CLAUDIO, D. **Fundamentos de Matemática Intervalar**. Porto Alegre, Brasil: Sagra-Luzzato, 1997. 93 p. Citado na página 26.

PIEGAT, A.; LANDOWSKI, M. Is the conventional interval arithmetic correct? **Journal of Theoretical and Applied Computer Science**, 2012. Citado 9 vezes nas páginas 21, 22, 33, 34, 39, 46, 47, 48 e 77.

PIEGAT, A.; LANDOWSKI, P. M. Two interpretations of multidimensional rdm interval arithmetic - multiplication and division. **International Journal of Fuzzy Systems**, 2013. Citado 6 vezes nas páginas 21, 22, 34, 39, 46 e 48.

PYTHON. 2018. Disponível em: <http://www.python.org>. Citado 3 vezes nas páginas 23, 31 e 51.

RALL, L. B. Integration of interval functions ii - the finite case*. **Society for Industrial and Applied Mathematics**, 1982. Citado na página 29.

RATSCHEK, H.; ROKNE, J. **Computer Methods for the Range of Functions**. New York: Halsted Press, Wiley, 1984. Citado 2 vezes nas páginas 46 e 47.

RATSCHEK, H.; ROKNE, J. **New Computer Methods for Global Optimization**. Chichester, United Kingdom: Limited, 1988. 229 p. Citado 2 vezes nas páginas 21 e 27.

ROCHA, A. V. **Probabilidade e Estatística**. Caixa Postal 5081 – Cidade Universitária João Pessoa – Paraíba – Brasil CEP: 58.051 – 970: EDITORA DA UFPB, 2014. Citado 2 vezes nas páginas 52 e 57.

SANTIAGO, R. H. N.; BEDREGAL, B. R. C.; ACIÓLY, B. M. Formal aspects of correctness and optimality of interval computations. **Formal Aspects of Computing**, Springer-Verlag, v. 18, p. 231–243, 2006. Citado na página 26.

SANTOS, M. G. **Probabilidades Autovalidáveis para as Variáveis Aleatórias Exponencial, Normal e Uniforme**. Tese (Doutorado) — Universidade Federal de Pernambuco, Recife, 2010. Citado 2 vezes nas páginas 52 e 58.

TOMASZEWSKA, K. A new approach to diabetic control – human glucose metabolism using interval arithmetic. **PhD interdisciplinary Journal**, 2015. Citado 2 vezes nas páginas 42 e 43.

TOSCANI, L.; VELOSO, P. **Complexidade de Algoritmos: análise, projetos e métodos**. [S.l.]: Sagra-Luzzato, 2001. Citado na página 30.

TOSCANI, L. V.; VELOSO, P. A. S. **Complexidade de Algoritmos**. [S.l.]: Bookman, 2009. v. 13. (Livros Didáticos Informática UFRGS, 9788540701397). Citado 2 vezes nas páginas 29 e 30.

VARJÃO, F. R. G. **IntPy: Computação Científica Auto Validável em Python**. Dissertação (Mestrado) — Universidade Federal de Pernambuco, Recife, 2011. Citado 2 vezes nas páginas 31 e 44.

VILCHES, M. A. **Cálculo para Economia e Administração. Acessado em: 1 de junho de 2018. Disponível em: http://www.ime.uerj.br/ calculo/reposit/ecomat.pdf calculo.** 2011. Citado 3 vezes nas páginas 62, 66 e 67.

ZURRAS, D. et al. Ieee standard for floating-point arithmetic. **IEEE Std 754-2008**, p. 1–70, 2008. Citado na página 21.

Anexos

ANEXO A – IMPLEMENTAÇÃO DAS OPERAÇÕES MATEMÁTICAS DA ARITMÉTICA RDM

Todas as implementações descritas nesta foram realizadas utilizando o ambiente de desenvolvimento intervalar composto pela linguagem de programação Python e o pacote de extensão intervalar IntPy.

```
# -*- coding: utf-8 -*-

from intpy import *
from math import *

# ============================================================================================
# Operacao de Soma RDM
# ============================================================================================
def RDM_soma(a, b, alpha_a=None, alpha_b=None):
    alpha_i = alpha_a or IReal(0, 1)
    alpha_s = alpha_b or IReal(0, 1)
    lista = [
        (a.inf + b.inf + alpha_a.sup * (a.sup - a.inf) + alpha_b.sup * (b.sup - b.inf)),
        (a.inf + b.inf + alpha_a.inf * (a.sup - a.inf) + alpha_b.inf * (b.sup - b.inf)),
        (a.inf + b.inf + alpha_a.inf * (a.sup - a.inf) + alpha_b.sup * (b.sup - b.inf)),
        (a.inf + b.inf + alpha_a.sup * (a.sup - a.inf) + alpha_b.inf * (b.sup - b.inf))
    ]
    s = IReal(min(lista), max(lista))
    return s

# ============================================================================================
# Operacao de Subtracao RDM
# ============================================================================================
def RDM_sub(a, b, alpha_a=None, alpha_b=None):
    alpha_a = alpha_a or IReal(0, 1)
    alpha_b = alpha_b or IReal(0, 1)
    lista = [
        (a.inf - b.inf + alpha_a.sup * (a.sup - a.inf) - alpha_b.sup * (b.sup - b.inf)),
        (a.inf - b.inf + alpha_a.inf * (a.sup - a.inf) - alpha_b.inf * (b.sup - b.inf)),
        (a.inf - b.inf + alpha_a.inf * (a.sup - a.inf) - alpha_b.sup * (b.sup - b.inf)),
        (a.inf - b.inf + alpha_a.sup * (a.sup - a.inf) - alpha_b.inf * (b.sup - b.inf))
```

```
    ]
    s = IReal(min(lista), max(lista))
    return s

# ============================================================================================
# Operacao de Multiplicacao RDM
# ============================================================================================
def RDM_mult(a, b, alpha_a=None, alpha_b=None):
    alpha_a = alpha_a or IReal(0, 1)
    alpha_b = alpha_b or IReal(0, 1)
    lista = [
        ((a.inf + alpha_a.sup * (a.diameter())) * (b.inf + alpha_b.sup * (b.diameter()))),
        ((a.inf + alpha_a.sup * (a.diameter())) * (b.inf + alpha_b.inf * (b.diameter()))),
        ((a.inf + alpha_a.inf * (a.diameter())) * (b.inf + alpha_b.sup * (b.diameter()))),
        ((a.inf + alpha_a.inf * (a.diameter())) * (b.inf + alpha_b.inf * (b.diameter())))
    ]
    s = IReal(min(lista), max(lista))
    return s

# ============================================================================================
# Operacao de Divisao RDM
# ============================================================================================
def RDM_div(a, b, alpha_a=None, alpha_b=None):
    alpha_a = alpha_a or IReal(0, 1)
    alpha_b = alpha_b or IReal(0, 1)
    lista = [
        ((a.inf + alpha_a.sup * (a.diameter())) / (b.inf + alpha_b.sup * (b.diameter()))),
        ((a.inf + alpha_a.sup * (a.diameter())) / (b.inf + alpha_b.inf * (b.diameter()))),
        ((a.inf + alpha_a.inf * (a.diameter())) / (b.inf + alpha_b.sup * (b.diameter()))),
        ((a.inf + alpha_a.inf * (a.diameter())) / (b.inf + alpha_b.inf * (b.diameter())))
    ]
    s = IReal(min(lista), max(lista))
    return s

# ============================================================================================
# Operacao de Exponenciacao 'e' RDM
# ============================================================================================
def RDM_exp(xi, xs=None, n=None, alpha_i=None, alpha_s=None):
    alpha_i = alpha_i or 0
    alpha_s = alpha_s or 1
    if xs is None:  xs = xi
    t = t or 30
    resultinf = 0
    resultsup = 0
```

```
    s = 0
    listfat = [
        1.0, 1.0, 2.0, 6.0, 24.0, 120.0, 720.0, 5040.0, 40320.0, 362880.0, 3628800.0,
        39916800.0, 479001600.0, 6227020800.0, 87178291200.0, 1307674368000.0,
        20922789888000.0, 355687428096000.0, 6402373705728000, 121645100408832000,
        2432902008176640000, 51090942171709440000, 1124000727777607680000,
        25852016738884976640000, 620448401733239439360000, 15511210043330985984000000,
        403291461126605635584000000, 10888869450418352160768000000,
        304888344611713860501504000000, 8841761993739701954543616000000
    ]
    if xi < 0 and xs < 0:
        x = IReal(-xi, -xs)
        for i in range(n):
            resultinf = resultinf + (pow(x.inf + alpha_i * (x.sup - x.inf), i)
                                                        / int(listfat[i]))
            resultsup = resultsup + (pow(x.inf + alpha_s * (x.sup - x.inf), i)
                                                        / int(listfat[i]))
        s = IReal(1 / resultinf, 1 / resultsup)
        return s
    elif xi < 0 <= xs:
        xinf = -xi
        xsup =  xs
        for i in range(n):
            resultinf = resultinf + (pow(xinf + alpha_i * (xsup - xinf), i)
                                                       / int(listfat[i]))
            resultsup = resultsup + (pow(xinf + alpha_s * (xsup - xinf), i)
                                                       / int(listfat[i]))
        s = IReal(1 / resultinf, resultsup)
        return s
    else:
        x = IReal(xi, xs)
        for i in range(n):
            resultinf = resultinf + (pow(x.inf + alpha_i * (x.sup - x.inf), i)
                                                        / int(listfat[i]))
            resultsup = resultsup + (pow(x.inf + alpha_s * (x.sup - x.inf), i)
                                                        / int(listfat[i]))
        s = IReal(resultinf, resultsup)
    return s

# ================================================================================
# Precisao de 'n' para Exponenciacao 'e' RDM
# ================================================================================
def precisao_taylor(prec, xsup):
    n = 0
```

```
soma = 2
while soma >= prec:
    soma = (math.pow(math.ceil(xsup), n + 1)
    / math.factorial(n + 1)) * pow(3, math.ceil(xsup))
    n = n + 1
return n
```

ANEXO B – IMPLEMENTAÇÃO DAS APLICAÇÕES DOS MÉTODOS DE INTEGRAÇÃO ÀS ARIMÉTICAS RDM E MOORE PARA DISTRIBUIÇÃO DE PROBABILIDADE EXPONENCIAL

Todas as implementações descritas nesta foram realizadas utilizando o ambiente de desenvolvimento intervalar composto pela linguagem de programação Python e o pacote de extensão intervalar IntPy.

B.1 Distribuição Exponencial utilizando o Método de Integração de Rall

```
# -*- coding: utf-8 -*-
from DefOpRDM import *
from intpy import *
import time

def powI(dado, exp):  # Definicao de potencia intervalar.
    result = IReal(dado.inf ** exp.inf, dado.sup ** exp.sup)
    return result

# ============================================================================================
# DISTRIBUICAO EXPONENCIAL EM PONTO FLUTUANTE
# ============================================================================================
def ExponencialRS(alp, a, b):
    begin_time = time.time()
    total = 0
    n = 100000
    tamanho = (a - b) / float(n)
    total = total + (alp * (exp(-a * alp)))
    total = total + (alp * (exp(-b * alp)))
    aux = b
    for i in range(n - 1):
        if i % 2 == 0:
```

```
            aux = aux + tamanho
            total = total + 4 * (alp * (exp(-aux * alp)))
        else:
            aux = aux + tamanho
            total = total + 2 * (alp * (exp(-aux * alp)))
    total = (tamanho / 3.0) * total
    end_time = time.time()
    print ("TEMPO:", end_time - begin_time)
    return (-total)

# ============================================================================================
# DISTRIBUICAO EXPONENCIAL INTERVALAR - ARITMETICA INTERVALAR DE MOORE
# ============================================================================================
def Moore_exponencialIR(alp, a, b):
    begin_time = time.time()
    listaInt = []
    e = IReal(2.71828182845904523536028747135266249775724709369995 9)
    x = IReal(a, b)
    n = 100000
    totalaux = IReal(0)
    alp = IReal(alp)
    h = (x.sup - x.inf) / float(n)
    aux = x.inf
    for i in range(n):
        listaInt += [IReal(aux, aux + h)]
        aux += h
    for i in range(n):
        a1 = powI(e, (-listaInt[i] * alp))
        totalaux = totalaux + (alp * a1)
    end_time = time.time()
    print ("TEMPO:", end_time - begin_time)
    print("DIAMETRO:", (totalaux * h).diameter())
    return totalaux * h

# ============================================================================================
# DISTRIBUICAO EXPONENCIAL INTERVALAR - ARITMETICA INTEVALAR RDM
# ============================================================================================
def RDM_exponencialIR(alp, a, b):
    begin_time = time.time()
    listaInt = []
    x = IReal(a, b)
    n = 100000
    totalaux = IReal(0)
    alp = IReal(alp)
```

```
    h = (x.sup - x.inf) / float(n)
    aux = x.inf
    for i in range(n):
        listaInt += [IReal(aux, aux + h)]
        aux += h
    for i in range(n):
        a1 = RDM_mult(-listaInt[i], alp)
        a2 = RDM_exp(a1.inf, a1.sup)
        a3 = RDM_mult(alp, a2)
        totalaux = RDM_soma(totalaux, a3)
    totalaux = RDM_mult(totalaux, IReal(h))
    end_time = time.time()
    print ("TEMPO:", end_time - begin_time)
    print("DIAMETRO:", totalaux.diameter())
    return totalaux
```

B.2 Distribuição Exponencial utilizando o Método de Integração de Bedregal

```
# -*- coding: utf-8 -*-
from DefOpRDM import *
from intpy import *
import time

def powI(dado, exp):  # Definicao de potencia intervalar.
    result = IReal(dado.inf ** exp.inf, dado.sup ** exp.sup)
    return result

# ============================================================================================
# DISTRIBUICAO EXPONENCIAL EM PONTO FLUTUANTE
# ============================================================================================
def exponencialRS(alp, a, b):
    begin_time = time.time()
    total = 0
    n = 100000
    tamanho = (a - b) / float(n)
    total = total + (alp * (exp(-a * alp)))
    total = total + (alp * (exp(-b * alp)))
    aux = b
    for i in range(n - 1):
        if i % 2 == 0:
            aux = aux + tamanho
            total = total + 4 * (alp * (exp(-aux * alp)))
```

```
        else:
            aux = aux + tamanho
            total = total + 2 * (alp * (exp(-aux * alp)))
    total = (tamanho / 3.0) * total
    end_time = time.time()
    print ("TEMPO:", end_time - begin_time)
    return (-total)

# ============================================================================================
# DISTRIBUICAO EXPONENCIAL INTERVALAR - ARITMETICA INTERVALAR DE MOORE
# ============================================================================================
def Moore_exponencialIB(alp, a, b):
    begin_time = time.time()
    listaInt = []
    e = IReal(2.71828182845904523536028747135266249775724709369995 95)
    totalaux = IReal(0)
    x = IReal(a)
    y = IReal(b)
    alp = IReal(alp)
    n = 100000
    h = (y.inf - x.inf) / float(n)
    aux = x.inf
    for i in range(n):
        listaInt += [IReal(aux, aux + h)]
        aux += h
    for i in range(n):
        a1 = powI(e, (-listaInt[i] * alp))
        totalaux = totalaux + (alp * a1) * h
    limite_inf = abs(y.inf - x.inf)
    limite_sup = abs(y.sup - x.sup)
    if limite_inf > limite_sup:
        dM = limite_inf
    else:
        dM = limite_sup
    bedrego = dM / (y.inf - x.inf)
    total = totalaux * bedrego
    end_time = time.time()
    print ("TEMPO:", end_time - begin_time)
    print("DIAMETRO:", total.diameter())
    return total

# ============================================================================================
# DISTRIBUICAO EXPONENCIAL INTERVALAR - ARITMETICA INTERVALAR RDM
# ============================================================================================
```

```
def RDM_exponencialIB(alp, a, b):
    begin_time = time.time()
    listaInt = []
    n = 100000
    total = IReal(0)
    totalaux = IReal(0)
    x = IReal(a)
    y = IReal(b)
    alp = IReal(alp)
    h = (y.inf - x.inf) / float(n)
    aux = x.inf
    for i in range(n):
        listaInt += [IReal(aux, aux + h)]
        aux += h
    for i in range(n):
        a1 = RDM_mult(-listaInt[i], alp)
        a2 = RDM_exp(a1.inf, a1.sup)
        a3 = RDM_mult(alp, a2)
        a4 = RDM_mult(a3, IReal(h))
        totalaux = RDM_soma(a4, totalaux)
    limite_inf = abs(y.inf - x.inf)
    limite_sup = abs(y.sup - x.sup)
    if limite_inf > limite_sup:
        dM = limite_inf
    else:
        dM = limite_sup
    bedrego = dM / (y.inf - x.inf)
    total = RDM_mult(IReal(bedrego), totalaux)
    end_time = time.time()
    print ("TEMPO:", end_time - begin_time)
    print("DIAMETRO:", total.diameter())
    return total
```

B.3 Distribuição Exponencial utilizando o Método de Integração de Simpson Intervalar

```
# -*- coding: utf-8 -*-
from DefOpRDM import *
from intpy import *
import time
from math import *
```

```
def powI(dado, exp):  # Definicao de potencia intervalar.
    result = IReal(dado.inf ** exp.inf, dado.sup ** exp.sup)
    return result

# ============================================================================================
# DISTRIBUICAO EXPONENCIAL EM PONTO FLUTUANTE
# ============================================================================================
def exponencialRS(alp, a, b):
    begin_time = time.time()
    total = 0
    n = 100000
    tamanho = (a - b) / float(n)
    total = total + (alp * (exp(-a * alp)))
    total = total + (alp * (exp(-b * alp)))
    aux = b
    for i in range(n - 1):
        if i % 2 == 0:
            aux = aux + tamanho
            total = total + 4 * (alp * (exp(-aux * alp)))
        else:
            aux = aux + tamanho
            total = total + 2 * (alp * (exp(-aux * alp)))
    total = (tamanho / 3.0) * total
    end_time = time.time()
    print ("TEMPO:", end_time - begin_time)
    return (-total)

# ============================================================================================
# DISTRIBUICAO EXPONENCIAL INTERVALAR - ARITMETICA INTERVALAR DE MOORE
# ============================================================================================
def Moore_exponencialIS(alp, a, b):
    begin_time = time.time()
    listaInt = []
    e = 2.71828182845904523536028747135266249775724709369995 95
    totalI = IReal(0)
    total = 0
    n = 100000
    x = IReal(a, b)
    h = (x.sup - x.inf) / float(n)
    aux = x.inf
    for i in range(n):
        listaInt += [IReal(aux, aux + h)]
        aux = aux + h
    for i in range(n):
```

```
        totalaux = IReal(0)
        totalaux = totalaux + (alp * e ** (-listaInt[i].inf * alp))
        totalaux = totalaux + (alp * e ** (-listaInt[i].sup * alp))
        aux = (listaInt[i].inf + listaInt[i].sup) / 2
        totalaux = totalaux + (4 * alp * e ** (-aux * alp))
        totalaux = totalaux * ((listaInt[i].sup - listaInt[i].inf) / 6)
        total = totalaux + total
    end_time = time.time()
    print ("TEMPO:", end_time - begin_time)
    print("DIAMETRO:", total.diameter())
    return total

# ============================================================================================
# DISTRIBUICAO EXPONENCIAL INTERVALAR - ARITMETICA INTERVALAR RDM
# ============================================================================================
def RDM_exponencialIS(alp, a, b):
    begin_time = time.time()
    listaInt = []
    n = 100000
    x = IReal(a, b)
    total = IReal(0)
    alp = IReal(alp)
    h = (x.sup - x.inf) / float(n)
    aux = x.inf
    for i in range(n):
        listaInt += [IReal(aux, aux + h)]
        aux = aux + h
    for i in range(n):
        totalaux = IReal(0)
        temp = RDM_mult(-listaInt[i], alp)
        totalaux = RDM_soma(totalaux, (RDM_mult(alp, RDM_exp(temp.inf))))
        totalaux = RDM_soma(totalaux, (RDM_mult(alp, RDM_exp(temp.sup))))
        aux = (listaInt[i].inf + listaInt[i].sup) / 2
        temp = RDM_mult(alp, IReal(-aux))
        totalaux = RDM_soma(totalaux, (RDM_mult(IReal(4), RDM_mult(alp, RDM_exp(temp.inf)))))
        totalaux = RDM_mult(totalaux,  IReal((listaInt[i].sup - listaInt[i].inf) / 6))
        total = total + totalaux
    end_time = time.time()
    print ("TEMPO:", end_time - begin_time)
    print("DIAMETRO:", total.diameter())
    return total
```

ANEXO C – IMPLEMENTAÇÃO DAS APLICAÇÕES DOS MÉTODOS DE INTEGRAÇÃO ÀS ARIMÉTICAS RDM E MOORE PARA DISTRIBUIÇÃO DE PROBABILIDADE NORMAL

C.1 Distribuição Normal utilizando o Método de Integração de Rall

Todas as implementações descritas nesta foram realizadas utilizando o ambiente de desenvolvimento intervalar composto pela linguagem de programação Python e o pacote de extensão intervalar IntPy.

```
# -*- coding: utf-8 -*-
from DefOpRDM import *
from math import *
from intpy import *
import time

def powI(dado, exp):  # Definicao de potencia intervalar.
    result = IReal(dado.inf ** exp.inf, dado.sup ** exp.sup)
    return result

# ============================================================================================
# DISTRIBUICAO NORMAL EM PONTO FLUTUANTE
# ============================================================================================
def normalRS(sigma, mi, a, b):
    begin_time = time.time()
    n = 100000
    e = 2.71828182845904523536028747135266249775724709369995 95
    total = 0
    tamanho = (a - b) / float(n)
    total = total + ((1 / (sigma * sqrt(2 * math.pi))) * e ** (-(a - mi) ** 2
                                                               / (2 * sigma ** 2)))
    total = total + ((1 / (sigma * sqrt(2 * math.pi))) * e ** (-(b - mi) ** 2
                                                               / (2 * sigma ** 2)))
    aux = b
```

```
    for i in range(n - 1):
        if i % 2 == 0:
            aux = aux + tamanho
            total = total + (4 * ((1 / (sigma * sqrt(2 * math.pi))) * e ** (
                                    -(aux - mi) ** 2 / (2 * sigma ** 2))))
        else:
            aux = aux + tamanho
            total = total + (2 * ((1 / (sigma * sqrt(2 * math.pi))) * e ** (
                                    -(aux - mi) ** 2 / (2 * sigma ** 2))))
    total = (tamanho / 3.0) * -total
    end_time = time.time()
    print ("TEMPO:", end_time - begin_time)
    return float128(total)


# ============================================================================================
# DISTRIBUICAO NORMAL INTERVALAR - ARITMETICA INTERVALAR DE MOORE
# ============================================================================================
def Moore_normalIR(sigma, mi, a, b):
    begin_time = time.time()
    e = IReal(2.71828182845904523536028747135266249775724709369995 9)
    listaInt = []
    totalaux = IReal(0)
    sigma = IReal(sigma)
    mi = IReal(mi)
    x = IReal(a, b)
    n = 100000
    h = (x.sup - x.inf) / float(n)
    aux = x.inf
    for i in range(n):
        listaInt += [IReal(aux, aux + h)]
        aux += h
    for i in range(n):
        aux1 = (IReal(1) / (sigma * IReal(sqrt(2 * math.pi))))
        aux2 = (- powI(listaInt[i] - mi, IReal(2))) / (IReal(2)
                                    * powI(sigma, IReal(2)))
        aux3 = powI(e, aux2)
        aux4 = aux1 * aux3
        totalaux = totalaux + aux4
    total = totalaux * h
    end_time = time.time()
    print ("TEMPO:", end_time - begin_time)
    print("DIAMETRO:", total.diameter())
    return total
```

```
# ============================================================================================
# DISTRIBUICAO NORMAL INTERVALAR - ARITMETICA INTERVALAR RDM
# ============================================================================================
def RDM_normalIR(sigma, mi, a, b):
    begin_time = time.time()
    pi = 3.14159265358979323846264338327950288419716939937510
    listaInt = []
    totalaux = IReal(0)
    sigma = IReal(sigma)
    mi = IReal(mi)
    x = IReal(a, b)
    n = 100000
    h = (x.sup - x.inf) / float(n)
    aux = x.inf
    for i in range(n):
        listaInt += [IReal(aux, aux + h)]
        aux += h
    for i in range(n):
        aux1 = RDM_div(IReal(1), (RDM_mult(sigma, IReal(sqrt(2 * pi)))))
        aux2 = RDM_div((powI(RDM_sub(listaInt[i], mi), IReal(2))),
                       RDM_mult(IReal(2), powI(sigma, IReal(2))))
        aux3 = RDM_exp(-aux2.inf, -aux2.sup)
        aux4 = RDM_mult(aux1, aux3)
        totalaux = RDM_soma(totalaux, aux4)
    total = RDM_mult(totalaux, IReal(h))
    end_time = time.time()
    print ("TEMPO:", end_time - begin_time)
    print("DIAMETRO:", total.diameter())
    return total
```

C.2 Distribuição Normal utilizando o Método de Integração de Bedregal

Todas as implementações descritas nesta foram realizadas utilizando o ambiente de desenvolvimento intervalar composto pela linguagem de programação Python e o pacote de extensão intervalar IntPy.

```
# -*- coding: utf-8 -*-
from DefOpRDM import *
from math import *
from intpy import *
import time
```

```
def powI(dado, exp):  # Definicao de potencia intervalar.
    result = IReal(dado.inf ** exp.inf, dado.sup ** exp.sup)
    return result

# ============================================================================================
# DISTRIBUICAO NORMAL EM PONTO FLUTUANTE
# ============================================================================================
def normalRS(sigma, mi, a, b):
    begin_time = time.time()
    n = 100000
    e = 2.71828182845904523536028747135266249775724709369995 95
    total = 0
    tamanho = (a - b) / float(n)
    total = total + ((1 / (sigma * sqrt(2 * pi))) * e ** (-(a - mi) ** 2
                                                          / (2 * sigma ** 2)))
    total = total + ((1 / (sigma * sqrt(2 * pi))) * e ** (-(b - mi) ** 2
                                                          / (2 * sigma ** 2)))
    aux = b
    for i in range(n - 1):
        if i % 2 == 0:
            aux = aux + tamanho
            total = total + (4 * ((1 / (sigma * sqrt(2 * pi))) * e ** (
                                  -(aux - mi) ** 2 / (2 * sigma ** 2))))
        else:
            aux = aux + tamanho
            total = total + (2 * ((1 / (sigma * sqrt(2 * pi))) * e ** (
                                  -(aux - mi) ** 2 / (2 * sigma ** 2))))
    total = (tamanho / 3.0) * -total
    end_time = time.time()
    print ("TEMPO:", end_time - begin_time)
    return (total)

# ============================================================================================
# DISTRIBUICAO NORMAL INTERVALAR - ARITMETICA INTERVALAR DE MOORE
# ============================================================================================
def Moore_normalIB(sigma, mi, a, b):
    begin_time = time.time()
    e = IReal(2.71828182845904523536028747135266249775724709369995 9)
    listaInt = []
    totalaux = IReal(0)
    sigma = IReal(sigma)
    mi = IReal(mi)
    x = IReal(a)
    y = IReal(b)
```

```
    n = 100000
    h = (y.inf - x.inf) / float(n)
    aux = x.inf
    for i in range(n):
        listaInt += [IReal(aux, aux + h)]
        aux += h
    for i in range(n):
        aux1 = (IReal(1) / (sigma * IReal(sqrt(2 * pi))))
        aux2 = (- powI(listaInt[i] - mi, IReal(2))) / (IReal(2)
                                        * powI(sigma, IReal(2)))
        aux3 = powI(e, aux2)
        aux4 = aux1 * aux3
        totalaux = totalaux + (aux4 * h)
    limite_inf = abs(y.inf - x.inf)
    limite_sup = abs(y.sup - x.sup)
    if limite_inf > limite_sup:
        dM = limite_inf
    else:
        dM = limite_sup
    bedrego = dM / (y.inf - x.inf)
    total = totalaux * bedrego
    end_time = time.time()
    print ("TEMPO:", end_time - begin_time)
    print("DIAMETRO:", total.diameter())
    return total

# ============================================================================================
# DISTRIBUICAO NORMAL INTERVALAR - ARITMETICA INTERVALAR RDM
# ============================================================================================
def RDM_normalIB(sigma, mi, a, b):
    begin_time = time.time()
    pi = 3.14159265358979323846264338327950288419716939937510
    listaInt = []
    totalaux = IReal(0)
    sigma = IReal(sigma)
    mi = IReal(mi)
    x = IReal(a)
    y = IReal(b)
    n = 100000
    h = (y.inf - x.inf) / float(n)
    aux = x.inf
    for i in range(n):
        listaInt += [IReal(aux, aux + h)]
        aux += h
```

```
    for i in range(n):
        aux1 = RDM_div(IReal(1), (RDM_mult(sigma, IReal(sqrt(2 * pi)))))
        aux2 = RDM_div((powI(RDM_sub(listaInt[i], mi), IReal(2))),
                       RDM_mult(IReal(2), powI(sigma, IReal(2))))
        aux3 = RDM_exp(-aux2.inf, -aux2.sup)
        aux4 = RDM_mult(aux1, aux3)
        totalaux = RDM_soma(totalaux, RDM_mult(aux4, IReal(h)))
    limite_inf = abs(y.inf - x.inf)
    limite_sup = abs(y.sup - x.sup)
    if limite_inf > limite_sup:
        dM = limite_inf
    else:
        dM = limite_sup
    bedrego = dM / (y.inf - x.inf)
    total = RDM_mult(totalaux, IReal(bedrego))
    end_time = time.time()
    print ("TEMPO:", end_time - begin_time)
    print("DIAMETRO:", total.diameter())
    return total
```

C.3 Distribuição Normal utilizando o Método de Integração de Simpson

Todas as implementações descritas nesta foram realizadas utilizando o ambiente de desenvolvimento intervalar composto pela linguagem de programação Python e o pacote de extensão intervalar IntPy.

```
# -*- coding: utf-8 -*-
from DefOpRDM import *
from intpy import *
from math import *
import time

def powI(dado, exp):  # Definicao de potencia intervalar.
    result = IReal(dado.inf ** exp.inf, dado.sup ** exp.sup)
    return result

# ============================================================================
# DISTRIBUICAO NORMAL EM PONTO FLUTUANTE
# ============================================================================
def normalRS(sigma, mi, a, b):
    begin_time = time.time()
    n = 100000
```

```
    e = 2.7182818284590452353602874713526624977572470936999595
    total = 0
    tamanho = (a - b) / float(n)
    total = total + ((1 / (sigma * sqrt(2 * pi))) * e ** (-(a - mi) ** 2
                                                    / (2 * sigma ** 2)))
    total = total + ((1 / (sigma * sqrt(2 * pi))) * e ** (-(b - mi) ** 2
                                                    / (2 * sigma ** 2)))
    aux = b
    for i in range(n - 1):
        if i % 2 == 0:
            aux = aux + tamanho
            total = total + (4 * ((1 / (sigma * sqrt(2 * pi))) * e ** (
                                -(aux - mi) ** 2 / (2 * sigma ** 2))))
        else:
            aux = aux + tamanho
            total = total + (2 * ((1 / (sigma * sqrt(2 * pi))) * e ** (
                                -(aux - mi) ** 2 / (2 * sigma ** 2))))
    total = (tamanho / 3.0) * -total
    end_time = time.time()
    print ("TEMPO:", end_time - begin_time)
    return (total)

# ============================================================================================
# DISTRIBUICAO NORMAL INTERVALAR - ARITMETICA INTERVALAR DE MOORE
# ============================================================================================
def Moore_normalIS(sigma, mi, a, b):
    begin_time = time.time()
    listaInt = []
    n = 100000
    e = 2.7182818284590452353602874713526624977572470936999595
    total = 0
    x = IReal(a, b)
    h = (x.sup - x.inf) / float(n)
    aux = x.inf
    for i in range(n):
        listaInt += [IReal(aux, aux + h)]
        aux = aux + h
    for i in range(n):
        totalaux = IReal(0)
        totalaux = totalaux + ((1 / (sigma * sqrt(2 * pi))) * e ** (
                -((listaInt[i].inf - mi) ** 2) / (2 * sigma ** 2)))
        totalaux = totalaux + ((1 / (sigma * sqrt(2 * pi))) * e ** (
                -((listaInt[i].sup - mi) ** 2) / (2 * sigma ** 2)))
        aux = (listaInt[i].inf + listaInt[i].sup) / 2
```

```
        totalaux = totalaux + (4 * ((1 / (sigma * sqrt(2 * pi))) *
                        e ** (-(aux - mi) ** 2 / (2 * sigma ** 2))))
        totalaux = totalaux * ((listaInt[i].sup - listaInt[i].inf) / 6)
        total = totalaux + total
    end_time = time.time()
    print ("TEMPO:", end_time - begin_time)
    return total

# ============================================================================================
# DISTRIBUICAO NORMAL INTERVALAR - ARITMETICA INTERVALAR RDM
# ============================================================================================
def RDM_normalIS(sigma, mi, a, b):
    begin_time = time.time()
    listaInt = []
    n = 100000
    total = IReal(0)
    x = IReal(a, b)
    h = (x.sup - x.inf) / float(n)
    aux = x.inf
    sigma = IReal(sigma)
    for i in range(n):
        listaInt += [IReal(aux, aux + h)]
        aux = aux + h
    for i in range(n):
        totalaux = IReal(0)
        aux1 = RDM_div(IReal(1), (RDM_mult(sigma, IReal(sqrt(2 * pi)))))
        aux2 = RDM_mult(IReal(2), powI(sigma, IReal(2)))
        aux3 = RDM_div(powI(RDM_sub(IReal(listaInt[i].inf), IReal(mi)),
                                                    IReal(2)), aux2)
        aux4 = RDM_div(powI(RDM_sub(IReal(listaInt[i].sup), IReal(mi)),
                                                    IReal(2)), aux2)
        aux5 = RDM_exp(-aux3.inf, -aux3.sup)
        aux6 = RDM_exp(-aux4.inf, -aux4.sup)
        totalaux = RDM_soma(totalaux, RDM_mult(aux1, aux5))
        totalaux = RDM_soma(totalaux, RDM_mult(aux1, aux6))
        aux = (listaInt[i].inf + listaInt[i].sup) / 2
        aux = IReal(aux - mi)
        aux7 = RDM_mult(aux1, IReal(4))
        aux8 = RDM_div((-powI(aux, IReal(2))), aux2)
        aux9 = RDM_exp(aux8.inf, aux8.sup)
        totalaux = RDM_soma(totalaux, RDM_mult(aux7, aux9))
        totalaux = RDM_mult(totalaux, IReal((listaInt[i].sup - listaInt[i].inf) / 6))
        total = totalaux + total
    end_time = time.time()
```

```
    print ("TEMPO:", end_time - begin_time)
    return total
```

ANEXO D – IMPLEMENTAÇÃO DAS APLICAÇÕES DOS MÉTODOS DE INTEGRAÇÃO ÀS ARIMÉTICAS RDM E MOORE PARA DISTRIBUIÇÃO DE PROBABILIDADE GAMA

D.1 Distribuição Gama utilizando o Método de Integração de Rall

Todas as implementações descritas nesta foram realizadas utilizando o ambiente de desenvolvimento intervalar composto pela linguagem de programação Python e o pacote de extensão intervalar IntPy.

```
# -*- coding: utf-8 -*-
from DefOpRDM import *
from scipy import integrate
from scipy import inf
from intpy import *
import time

def powI(dado, exp):  # Definicao de potencia intervalar.
    result = IReal(dado.inf ** exp.inf, dado.sup ** exp.sup)
    return result

# ============================================================================================
# DISTRIBUICAO GAMA EM PONTO FLUTUANTE
# ============================================================================================
def gamaRS(lamb, v, a, b):
    begin_time = time.time()
    n = 100000
    total = 0
    tamanho = (a - b) / float(n)
    f = lambda x, a: exp(-lamb * x) * x ** (v - 1)
    y, err = integrate.quad(f, 0, inf, args=(1,))
    y = y ** -1
    total += (exp(-lamb * a) * a ** (v - 1))
    total += (exp(-lamb * b) * b ** (v - 1))
```

```
    aux = b
    for i in range(n - 1):
        if i % 2 == 0:
            aux = aux + tamanho
            total = total + 4 * ((exp(-lamb * aux) * aux ** (v - 1)))
        else:
            aux = aux + tamanho
            total = total + 2 * ((exp(-lamb * aux) * aux ** (v - 1)))
    total = (tamanho / 3.0) * total
    end_time = time.time()
    print('TEMPO:', end_time-begin_time)
    return (-total * y)

# ================================================================================================
# DISTRIBUICAO GAMA INTERVALAR - ARITMETICA INTERVALAR DE MOORE
# ================================================================================================
def Moore_gamaIR(lamb, v, a, b):
    begin_time = time.time()
    e = IReal(2.71828182845904523536028747135266249775724709369995 95)
    listaInt = []
    x = IReal(a, b)
    totalaux = IReal(0)
    total = IReal(0)
    n = 100000
    h = (x.sup - x.inf) / float(n)
    aux = x.inf
    f = lambda x, a: exp(-lamb * x) * x ** (v - 1)
    z, err = integrate.quad(f, 0, inf, args=(1,))
    z = z ** -1
    lamb = IReal(lamb)
    for i in range(n):
        listaInt += [IReal(aux, aux + h)]
        aux += h
    for i in range(n):
        a1 = powI(e, (-lamb * listaInt[i]))
        a2 = powI(listaInt[i], IReal(v - 1))
        totalaux = totalaux + (a1 * a2)
    total = totalaux * IReal(h) * IReal(z)
    end_time = time.time()
    print ("TEMPO:", end_time - begin_time)
    print("DIAMETRO:", total.diameter())
    return total
```

```
# ================================================================================================
# DISTRIBUICAO GAMA INTERVALAR - ARITMETICA INTERVALAR RDM
# ================================================================================================
def RDM_gamaIR(lamb, v, a, b):
    begin_time = time.time()
    listaInt = []
    x = IReal(a, b)
    totalaux = IReal(0)
    total = IReal(0)
    n = 100000
    h = (x.sup - x.inf) / float(n)
    aux = x.inf
    f = lambda x, a: exp(-lamb * x) * x ** (v - 1)
    z, err = integrate.quad(f, 0, inf, args=(1,))
    z = z ** -1
    lamb = IReal(lamb)
    for i in range(n):
        listaInt += [IReal(aux, aux + h)]
        aux += h
    for i in range(n):
        a1 = RDM_mult(-lamb, listaInt[i])
        a2 = RDM_exp(a1.inf, a1.sup)
        a3 = powI(listaInt[i], RDM_sub(IReal(v), IReal(1)))
        a4 = RDM_mult(a2, a3)
        totalaux = RDM_soma(totalaux, a4)
    totalaux = RDM_mult(totalaux, RDM_mult(IReal(h), IReal(z)))
    total = totalaux
    end_time = time.time()
    print ("TEMPO:", end_time - begin_time)
    print("DIAMETRO:", total.diameter())
    return total
```

D.2 Distribuição Gama utilizando o Método de Integração de Bedregal

Todas as implementações descritas nesta foram realizadas utilizando o ambiente de desenvolvimento intervalar composto pela linguagem de programação Python e o pacote de extensão intervalar IntPy.

```
# -*- coding: utf-8 -*-
from DefOpRDM import *
from scipy import integrate
```

```
from scipy import inf
from intpy import *
import time

def powI(dado, exp):  # Definicao de potencia intervalar.
    result = IReal(dado.inf ** exp.inf, dado.sup ** exp.sup)
    return result

# ============================================================================================
# DISTRIBUICAO GAMA EM PONTO FLUTUANTE
# ============================================================================================
def gamaRS(lamb, v, a, b):
    begin_time = time.time()
    n = 100000
    total = 0
    tamanho = (a - b) / float(n)
    f = lambda x, a: exp(-lamb * x) * x ** (v - 1)
    y, err = integrate.quad(f, 0, inf, args=(1,))
    y = y ** -1
    total += (exp(-lamb * a) * a ** (v - 1))
    total += (exp(-lamb * b) * b ** (v - 1))
    aux = b
    for i in range(n - 1):
        if i % 2 == 0:
            aux = aux + tamanho
            total = total + 4 * ((exp(-lamb * aux) * aux ** (v - 1)))
        else:
            aux = aux + tamanho
            total = total + 2 * ((exp(-lamb * aux) * aux ** (v - 1)))
    total = (tamanho / 3.0) * total
    end_time = time.time()
    print('TEMPO:', end_time-begin_time)
    return (-total * y)

# ============================================================================================
# DISTRIBUICAO GAMA INTERVALAR - ARITMETICA INTERVALAR DE MOORE
# ============================================================================================
def Moore_gamaIB(lamb, v, a, b):
    begin_time = time.time()
    e = IReal(2.7182818284590452353602874713526624977572470936999595)
    listaInt = []
    x = IReal(a)
    y = IReal(b)
    totalaux = IReal(0)
```

```
    total = IReal(0)
    n = 100000
    h = (y.inf - x.inf) / float(n)
    aux = x.inf
    f = lambda x, a: exp(-lamb * x) * x ** (v - 1)
    z, err = integrate.quad(f, 0, inf, args=(1,))
    z = z ** -1
    lamb = IReal(lamb)
    for i in range(n):
        listaInt += [IReal(aux, aux + h)]
        aux += h
    for i in range(n):
        a1 = powI(e, (-lamb * listaInt[i]))
        a2 = powI(listaInt[i], IReal(v - 1))
        totalaux = totalaux + (a1 * a2) * h
    limite_inf = abs(y.inf - x.inf)
    limite_sup = abs(y.sup - x.sup)
    if limite_inf > limite_sup:
        dM = limite_inf
    else:
        dM = limite_sup
    bedrego = dM / (y.inf - x.inf)
    total = totalaux * bedrego * z
    end_time = time.time()
    print ("TEMPO:", end_time - begin_time)
    print("DIAMETRO:", total.diameter())
    return total

# ============================================================================================
# DISTRIBUICAO GAMA INTERVALAR - ARITMETICA INTERVALAR RDM
# ============================================================================================
def RDM_gamaIB(lamb, v, a, b):
    begin_time = time.time()
    listaInt = []
    x = IReal(a)
    y = IReal(b)
    totalaux = IReal(0)
    total = IReal(0)
    n = 100000
    h = (y.inf - x.inf) / float(n)
    aux = x.inf
    f = lambda x, a: exp(-lamb * x) * x ** (v - 1)
    z, err = integrate.quad(f, 0, inf, args=(1,))
    z = z ** -1
```

```
    lamb = IReal(lamb)
    for i in range(n):
        listaInt += [IReal(aux, aux + h)]
        aux += h
    for i in range(n):
        a1 = RDM_mult(-lamb, listaInt[i])
        a2 = RDM_exp(a1.inf, a1.sup)
        a3 = powI(listaInt[i], RDM_sub(IReal(v), IReal(1)))
        a4 = RDM_mult(a2, a3)
        totalaux = RDM_soma(totalaux, RDM_mult(a4, IReal(h)))
    limite_inf = abs(y.inf - x.inf)
    limite_sup = abs(y.sup - x.sup)
    if limite_inf > limite_sup:
        dM = limite_inf
    else:
        dM = limite_sup
    bedrego = dM / (y.inf - x.inf)
    totalaux = RDM_mult(totalaux, IReal(bedrego))
    total = RDM_mult(totalaux, IReal(z))
    end_time = time.time()
    print ("TEMPO:", end_time - begin_time)
    print("DIAMETRO:", total.diameter())
    return total
```

D.3 Distribuição Gama utilizando o Método de Integração de Simpson

Todas as implementações descritas nesta foram realizadas utilizando o ambiente de desenvolvimento intervalar composto pela linguagem de programação Python e o pacote de extensão intervalar IntPy.

```
# -*- coding: utf-8 -*-
from DefOpRDM import *
from scipy import integrate
from scipy import inf
import time

def powI(dado, exp):  # Definicao de potencia intervalar.
    result = IReal(dado.inf ** exp.inf, dado.sup ** exp.sup)
    return result

# ================================================================================
# DISTRIBUICAO GAMA EM PONTO FLUTUANTE
```

```
# ============================================================================================
def gamaRS(lamb, v, a, b):
    begin_time = time.time()
    n = 100000
    total = 0
    tamanho = (a - b) / float(n)
    f = lambda x, a: exp(-lamb * x) * x ** (v - 1)
    y, err = integrate.quad(f, 0, inf, args=(1,))
    y = y ** -1
    total = total + (exp(-lamb * a) * a ** (v - 1))
    total = total + (exp(-lamb * b) * b ** (v - 1))
    aux = b
    for i in range(n - 1):
        if i % 2 == 0:
            aux = aux + tamanho
            total = total + 4 * (exp(-lamb * aux) * aux ** (v - 1))
        else:
            aux = aux + tamanho
            total = total + 2 * ((exp(-lamb * aux) * aux ** (v - 1)))
    total = (tamanho / 3.0) * total
    end_time = time.time()
    print('TEMPO:', end_time - begin_time)
    return (-total * y)

# ============================================================================================
# DISTRIBUICAO GAMA INTERVALAR - ARITMETICA INTERVALAR DE MOORE
# ============================================================================================
def Moore_gamaIS(lamb, v, a, b):
    begin_time = time.time()
    listaInt = []
    n = 100000
    f = lambda x, a: exp(-lamb * x) * x ** (v - 1)
    y, err = integrate.quad(f, 0, inf, args=(1,))
    y = y ** -1
    total = IReal(0)
    x = IReal(a, b)
    h = (x.sup - x.inf) / float(n)
    aux = x.inf
    for i in range(n):
        listaInt += [IReal(aux, aux + h)]
        aux += h
    for i in range(n):
        totalaux = IReal(0)
        totalaux = totalaux + (e ** (-lamb * listaInt[i].inf) * listaInt[i].inf ** (v - 1))
```

```
        totalaux = totalaux + (e ** (-lamb * listaInt[i].sup) * listaInt[i].sup ** (v - 1))
        aux = (listaInt[i].inf + listaInt[i].sup) / 2
        totalaux = totalaux + 4 * (e ** (-lamb * aux) * aux ** (v - 1))
        totalaux = totalaux * ((listaInt[i].sup - listaInt[i].inf) / 6)
        total = totalaux + total
    end_time = time.time()
    print('TEMPO:', end_time - begin_time)
    return total * y

# ============================================================================================
# DISTRIBUICAO GAMA INTERVALAR - ARITMETICA INTERVALAR RDM
# ============================================================================================
def RDM_gamaIS(lamb, v, a, b):
    begin_time = time.time()
    listaInt = []
    n = 100000
    total = IReal(0)
    f = lambda x, a: exp(-lamb * x) * x ** (v - 1)
    y, err = integrate.quad(f, 0, inf, args=(1,))
    y = y ** -1
    lamb = IReal(lamb)
    x = IReal(a, b)
    h = (x.sup - x.inf) / float(n)
    aux = x.inf
    for i in range(n):
        listaInt += [IReal(aux, aux + h)]
        aux += h
    for i in range(n):
        totalaux = IReal(0)
        aux1 = (RDM_mult(-lamb, listaInt[i]))
        aux2 = powI(listaInt[i], IReal(v - 1))
        totalaux = RDM_soma(totalaux, RDM_mult(RDM_exp(aux1.inf), IReal(aux2.inf)))
        totalaux = RDM_soma(totalaux, RDM_mult(RDM_exp(aux1.sup), IReal(aux2.sup)))
        aux = (listaInt[i].inf + listaInt[i].sup) / 2
        aux3 = RDM_mult(-lamb, IReal(aux))
        totalaux = RDM_soma(totalaux, RDM_mult(IReal(4), RDM_mult(
        RDM_exp(aux3.inf, aux3.sup), powI(IReal(aux), IReal(v - 1)))))
        totalaux = RDM_mult(totalaux, IReal((listaInt[i].sup - listaInt[i].inf) / 6))
        total = totalaux + total
    total = RDM_mult(total, IReal(y))
    end_time = time.time()
    print('TEMPO:', end_time - begin_time)
    return total
```

ANEXO E – IMPLEMENTAÇÃO DAS APLICAÇÕES DOS MÉTODOS DE INTEGRAÇÃO ÀS ARIMÉTICAS RDM E MOORE PARA DISTRIBUIÇÃO DE PROBABILIDADE DE PARETO

E.1 Distribuição de Pareto utilizando o Método de Integração de Rall

Todas as implementações descritas nesta foram realizadas utilizando o ambiente de desenvolvimento intervalar composto pela linguagem de programação Python e o pacote de extensão intervalar IntPy.

```
# -*- coding: utf-8 -*-
from DefOpRDM import *
from intpy import *
import time

def powI(dado, exp):  # Definicao de potencia intervalar.
    result = IReal(dado.inf ** exp.inf, dado.sup ** exp.sup)
    return result

# ================================================================================
# DISTRIBUICAO PARETO EM PONTO FLUTUANTE
# ================================================================================
def paretoRS(alpha, beta, a, b):
    begin_time = time.time()
    n = 100000
    total = 0
    tamanho = (a - b) / float(n)
    total = total + ((alpha * beta ** alpha) / (a ** (alpha + 1)))
    totalaux = total + ((alpha * beta ** alpha) / (a ** (alpha + 1)))
    total = total + ((alpha * beta ** alpha) / (b ** (alpha + 1)))
    aux = b
    for i in range(n - 1):
```

```
        if i % 2 == 0:
            aux += tamanho
            total += 4 * ((alpha * beta ** alpha) / (aux ** (alpha + 1)))
        else:
            aux += tamanho
            total += 2 * ((alpha * beta ** alpha) / (aux ** (alpha + 1)))
    total = (tamanho / 3.0) * total
    end_time = time.time()
    print ("TEMPO:", end_time - begin_time)
    return (-total)

# ============================================================================================
# DISTRIBUICAO PARETO INTERVALAR - ARITMETICA INTERVALAR DE MOORE
# ============================================================================================
def Moore_paretoIR(alp, beta, a, b):
    begin_time = time.time()
    listaInt = []
    totalaux = IReal(0)
    beta = IReal(beta)
    alp = IReal(alp)
    x = IReal(a, b)
    n = 100000
    h = (x.sup - x.inf) / float(n)
    aux = x.inf
    for i in range(n):
        listaInt += [IReal(aux, aux + h)]
        aux += h
    for i in range(n):
        aux1 = powI(beta, alp)
        aux2 = powI(listaInt[i], alp + IReal(1))
        totalaux = totalaux + alp * (aux1 / aux2)
    total = totalaux * h
    end_time = time.time()
    print ("TEMPO:", end_time - begin_time)
    print("DIAMETRO:", total.diameter())
    return total


# ============================================================================================
# DISTRIBUICAO PARETO INTERVALAR - ARITMETICA INTERVALAR RDM
# ============================================================================================
def RDM_paretoIR(alp, beta, a, b):
    begin_time = time.time()
    listaInt = []
```

```
    totalaux = IReal(0)
    beta = IReal(beta)
    alp = IReal(alp)
    x = IReal(a, b)
    n = 100000
    h = (x.sup - x.inf) / float(n)
    aux = x.inf
    for i in range(n):
        listaInt += [IReal(aux, aux + h)]
        aux += h
    for i in range(n):
        aux1 = powI(beta, alp)
        aux2 = powI(listaInt[i], RDM_soma(alp, IReal(1)))
        aux3 = RDM_div(RDM_mult(alp, aux1), aux2)
        totalaux = RDM_soma(totalaux, aux3)
    total = RDM_mult(totalaux, IReal(h))
    end_time = time.time()
    print ("TEMPO:", end_time - begin_time)
    print("DIAMETRO:", total.diameter())
    return total
```

E.2 Distribuição de Pareto utilizando o Método de Integração de Bedregal

Todas as implementações descritas nesta foram realizadas utilizando o ambiente de desenvolvimento intervalar composto pela linguagem de programação Python e o pacote de extensão intervalar IntPy.

```
# -*- coding: utf-8 -*-
from DefOpRDM import *
from numpy import float128
from intpy import *
import time


def powI(dado, exp):  # Definicao de potencia intervalar.
    result = IReal(dado.inf ** exp.inf, dado.sup ** exp.sup)
    return result

# ============================================================================================
# DISTRIBUICAO PARETO EM PONTO FLUTUANTE
# ============================================================================================
def paretoRS(alpha, beta, a, b):
```

```
    begin_time = time.time()
    n = 100000
    total = 0
    tamanho = (a - b) / float(n)
    total = total + ((alpha * beta ** alpha) / (a ** (alpha + 1)))
    totalaux = total + ((alpha * beta ** alpha) / (a ** (alpha + 1)))
    total = total + ((alpha * beta ** alpha) / (b ** (alpha + 1)))
    aux = b
    for i in range(n - 1):
        if i % 2 == 0:
            aux += tamanho
            total += 4 * ((alpha * beta ** alpha) / (aux ** (alpha + 1)))
        else:
            aux += tamanho
            total += 2 * ((alpha * beta ** alpha) / (aux ** (alpha + 1)))
    total = (tamanho / 3.0) * total
    end_time = time.time()
    print ("TEMPO:", end_time - begin_time)
    return float128(-total)

# ================================================================================
# DISTRIBUICAO PARETO INTERVALAR - ARITMETICA INTERVALAR DE MOORE
# ================================================================================
def Moore_paretoIB(alp, beta, a, b):
    begin_time = time.time()
    listaInt = []
    totalaux = IReal(0)
    x = IReal(a)
    y = IReal(b)
    n = 100000
    h = (y.inf - x.inf) / float(n)
    aux = x.inf
    beta = IReal(beta)
    alp = IReal(alp)
    for i in range(n):
        listaInt += [IReal(aux, aux + h)]
        aux += h
    for i in range(n):
        aux1 = powI(beta, alp)
        aux2 = powI(listaInt[i], alp + IReal(1))
        totalaux = totalaux + alp * (aux1 / aux2) * h
    limite_inf = abs(y.inf - x.inf)
    limite_sup = abs(y.sup - x.sup)
    if limite_inf > limite_sup:
```

```
        dM = limite_inf
    else:
        dM = limite_sup
    bedrego = dM / (y.inf - x.inf)
    total = totalaux * bedrego
    end_time = time.time()
    print ("TEMPO:", end_time - begin_time)
    print("DIAMETRO:", total.diameter())
    return total

# ============================================================================================
# DISTRIBUICAO PARETO INTERVALAR - ARITMETICA INTERVALAR RDM
# ============================================================================================
def RDM_paretoIB(alp, beta, a, b):
    begin_time = time.time()
    listaInt = []
    totalaux = IReal(0)
    x = IReal(a)
    y = IReal(b)
    n = 100000
    h = (y.inf - x.inf) / float(n)
    aux = x.inf
    beta = IReal(beta)
    alp = IReal(alp)
    for i in range(n):
        listaInt += [IReal(aux, aux + h)]
        aux += h
    for i in range(n):
        aux1 = powI(beta, alp)
        aux2 = powI(listaInt[i],  RDM_soma(alp, IReal(1)))
        aux3 = RDM_div(RDM_mult(alp, RDM_mult(IReal(h), aux1)), aux2)
        totalaux = RDM_soma(totalaux, aux3)
    limite_inf = abs(y.inf - x.inf)
    limite_sup = abs(y.sup - x.sup)
    if limite_inf > limite_sup:
        dM = limite_inf
    else:
        dM = limite_sup
    bedrego = dM / (y.inf - x.inf)
    total = RDM_mult(totalaux, IReal(bedrego))
    end_time = time.time()
    print ("TEMPO:", end_time - begin_time)
    print("DIAMETRO:", total.diameter())
    return total
```

E.3 Distribuição de Pareto utilizando o Método de Integração de Simpson

Todas as implementações descritas nesta foram realizadas utilizando o ambiente de desenvolvimento intervalar composto pela linguagem de programação Python e o pacote de extensão intervalar IntPy.

```
from DefOpRDM import *
from intpy import *
import time

def powI(dado, exp):  # Definicao de potencia intervalar.
    result = IReal(dado.inf ** exp.inf, dado.sup ** exp.sup)
    return result

# ============================================================================================
# DISTRIBUICAO PARETO EM PONTO FLUTUANTE
# ============================================================================================
def paretoRS(alpha, beta, a, b):
    begin_time = time.time()
    n = 100000
    total = 0
    tamanho = (a - b) / float(n)
    total = total + ((alpha * beta ** alpha) / (a ** (alpha + 1)))
    totalaux = total + ((alpha * beta ** alpha) / (a ** (alpha + 1)))
    total = total + ((alpha * beta ** alpha) / (b ** (alpha + 1)))
    aux = b
    for i in range(n - 1):
        if i % 2 == 0:
            aux += tamanho
            total += 4 * ((alpha * beta ** alpha) / (aux ** (alpha + 1)))
        else:
            aux += tamanho
            total += 2 * ((alpha * beta ** alpha) / (aux ** (alpha + 1)))
    total = (tamanho / 3.0) * total
    end_time = time.time()
    print ("TEMPO:", end_time - begin_time)
    return float128(-total)

# ============================================================================================
# DISTRIBUICAO PARETO INTERVALAR - ARITMETICA INTERVALAR DE MOORE
# ============================================================================================
def Moore_paretoIS(alpha, beta, a, b):
    begin_time = time.time()
```

```
    listaInt = []
    n = 100000
    totalI = IReal(0)
    total = 0
    x = IReal(a, b)
    h = (x.sup - x.inf) / float(n)
    aux = x.inf
    for i in range(n):
        listaInt += [IReal(aux, aux + h)]
        aux = aux + h
    for i in range(n):
        totalaux = IReal(0)
        totalaux1 = IReal(0)
        totalaux = totalaux + ((alpha * beta ** alpha) / (listaInt[i].inf ** (alpha + 1)))
        totalaux1 = totalaux1 + ((alpha * beta ** alpha) / (listaInt[i].sup ** (alpha + 1)))
        totalaux = totalaux + totalaux1
        aux = (listaInt[i].inf + listaInt[i].sup) / 2
        totalaux = totalaux + 4 * ((alpha * beta ** alpha) / (aux ** (alpha + 1)))
        totalaux = totalaux * ((listaInt[i].sup - listaInt[i].inf) / 6)
        total = totalaux + total
    end_time = time.time()
    print ("TEMPO:", end_time - begin_time)
    return total

# ============================================================================================
# DISTRIBUICAO PARETO INTERVALAR - ARITMETICA INTERVALAR RDM
# ============================================================================================
def RDM_paretoIS(alpha, beta, a, b):
    begin_time = time.time()
    listaInt = []
    n = 100000
    totalI = IReal(0)
    total = IReal(0)
    x = IReal(a, b)
    h = (x.sup - x.inf) / float(n)
    aux = x.inf
    for i in range(n):
        listaInt += [IReal(aux, aux + h)]
        aux = aux + h
    for i in range(n):
        totalaux = IReal(0)
        totalaux1 = IReal(0)
        totalaux2 = IReal(0)
        a1 = powI(IReal(beta), IReal(alpha))
```

```
        a2 = powI(listaInt[i], IReal(alpha + 1))
        a3 = RDM_mult(IReal(alpha), a1)
        a4 = RDM_div(a3, a2)
        totalaux1 = RDM_soma(totalaux1, IReal(a4.inf))
        totalaux2 = RDM_soma(totalaux2, IReal(a4.sup))
        totalaux = RDM_soma(totalaux1, totalaux2)
        aux = (listaInt[i].inf + listaInt[i].sup) / 2
        a = (4 * ((alpha * beta ** alpha) / (aux ** (alpha + 1))))
        totalaux = RDM_soma(totalaux, IReal(a))
        totalaux = RDM_mult(totalaux, IReal((listaInt[i].sup - listaInt[i].inf) / 6))
        total = totalaux + total
    end_time = time.time()
    print ("TEMPO:", end_time - begin_time)
    return total
```