CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
DEPARTAMENTO DE MATEMÁTICA

Nome Sobrenome

Métodos Numéricos

SEROPÉDICA
2025

Nome Sobrenome

MÉTODOS NUMÉRICOS

Monografia apresentada à Banca Examinadora da Universidade Federal Rural do Rio de Janeiro, como parte dos requisitos para obtenção do título de Bacharel em Matemática sob orientação do Prof. Dr. Nome Sobrenome do Orientador

SEROPÉDICA

2025

UNIVERSIDADE FEDERAL RURAL DO RIO DE JANEIRO
INSTITUTO DE CIÊNCIAS EXATAS
DEPARTAMENTO DE MATEMÁTICA

COORDENAÇÃO DO CURSO DE GRADUAÇÃO EM MATEMÁTICA

A monografia "Métodos Numéricos", apresentada e defendida por NOME SOBRENOME, matrícula 2022019000-0, foi aprovada pela Banca Examinadora com conceito "X", recebendo o número 000.

Seropédica, 12 de dezembro de 2025

BANCA EXAMINADORA:

Prof. Dr. Presidente da Banca
Orientador

Prof. Dr. Membro 1
Convidado 1

Prof. Dr. Membro 2
Convidado 1

Agradecimentos

Aqui está um agradecimento.

Resumo

Aqui está um resumo.

Abstract

Here is an (optional) abstract.

Sumário

Introdução

Capítulo 1

Pontos Flutuantes

1.1 Sistemas Numéricos

Na matemática, um sistema numérico é um conjunto de regras e símbolos utilizados para representar quantidades através do que chamamos de números. Existem dois tipos de sistemas: os posicionais e os aditivos.

O sistema aditivo é aquele em que os números são representados pelas somas dos valores dos símbolos, geralmente agrupados lado a lado em ordem decrescente, como, por exemplo, os sistemas romano e egípcio. Já o sistema posicional leva em conta não só os dígitos mas também a posição que eles ocupam no número. A quantidade de símbolos diferentes que são utilizados para representar os dígitos está ligada à **base** desse sistema, e cada posição do dígito no número refere-se a uma potência dessa base. Por exemplo, no sistema decimal (base 10), usamos os dígitos de 0 a 9. No sistema binário (base 2), usamos os dígitos 0 e 1. E já no sistema hexadecimal (base 16), usamos de 0 a 9 e as letras A a F (que representam 10 a 15). A quantidade de símbolos diferentes que são utilizados para representar os dígitos está ligada à **base** desse sistema, e cada posição do dígito no número refere-se a uma potência dessa base. Por exemplo, no sistema decimal (base 10), usamos os dígitos de 0 a 9. No sistema binário (base 2), usamos os dígitos 0 e 1. E já no sistema hexadecimal (base 16), usamos de 0 a 9 e as letras A a F (que representam 10 a 15).

Com essas diferentes formas de representar um número, a escolha do sistema depende do contexto e da aplicação. No uso cotidiano, a base decimal é a mais utilizada. Já as bases binária e hexadecimal, são amplamente utilizadas na ciência da computação

em operações aritméticas dos processadores e em algumas linguagens de programação para endereçamento de memória.

Um número N pode ser representado em uma base b no seguinte formato

$$N = \pm \sum_{i=-k}^{n} d_i b^i, \tag{1.1}$$

em que d_i são os dígitos na base b, k é o número de casas decimais à direita do ponto, e $n + 1 + k$ é o número de dígitos significativos. Vejamos alguns exemplos.

Exemplo 1.1.1. Vamos escrever o número 13 nas bases 10 e 2.

- Número na base decimal: $13 = 1 \times 10^1 + 3 \times 10^0 = 13_{10}$
- Número na base binária: $13 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 1101_2$

Exemplo 1.1.2. Agora vamos escrever o número 3,5625 nas bases 10 e 2.

- Número na base decimal:

$$3{,}5625 = 3 \times 10^0 + 5 \times 10^{-1} + 6 \times 10^{-2} + 2 \times 10^{-3} + 5 \times 10^{-4} = 3{,}5625_{10}$$

- Número na base binária:

$$3{,}5625 = 1 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4} = 11{,}1001_2$$

1.2 Aritmética de Ponto Flutuante

A *aritmética de ponto flutuante* é o sistema adotado por computadores para que lidem com números reais utilizando uma notação compacta e eficaz. Essa técnica é utilizada para representar e manipular números reais de forma prática e eficiente. Ela permite representar números de grandezas diversas, que não podem ser armazenados com precisão, utilizando apenas números inteiros.

Um sistema de ponto flutuante F pode ser definido como

$$F(\beta, t, L, U)$$

cuja representação normalizada de um número real N nesse sistema é dada por

$$N = \pm(d_1.d_2...d_t)_\beta \times \beta^e \tag{1.2}$$

em que

- N é o número real;
- β é a base que a máquina opera;
- t é o número de dígitos na mantissa, tal que $0 \leq d_j \leq \beta - 1$, j = 1, ...,t, $d_1 \neq 0$;
- L é o menor expoente inteiro;
- U é o maior expoente inteiro;
- e é o expoente inteiro no intervalo [L,U].

No padrão IEEE 754 (usado na maioria dos sistemas eletrônicos), um número de ponto flutuante é dividido em três partes:

- **Sinal (S)**: 1 bit indicando se o número é positivo ($S = 0$) ou negativo ($S = 1$),
- **Expoente (E)**: campo que representa o expoente com viés (bias),
- **Mantissa (M)**: parte fracionária significativa do número.

A fórmula completa de reconstrução do número é:

$$\text{Valor} = (-1)^S \times (1.M) \times 2^{E-\text{bias}}$$

onde:

- S é o bit de sinal,
- $1.M$ indica que há um bit implícito "1" antes da mantissa nos números normalizados,
- bias é um valor constante que depende da precisão (por exemplo, 127 para 32 bits).

1.2.1 Precisão Simples e Precisão Dupla

Em sistemas computacionais, os números em ponto flutuante podem ser representados em diferentes níveis de precisão. Os dois mais comuns são:

- **Precisão Simples (32 bits)**
- **Precisão Dupla (64 bits)**

Esses formatos seguem o padrão `IEEE 754` de representação binária de números reais.

Comparação entre os formatos

Característica	**Precisão Simples (32 bits)**	**Precisão Dupla (64 bits)**
Bits totais	32	64
Bit de sinal	1	1
Bits de expoente	8	11
Bits de mantissa	23	52
Bias	127	1023
Intervalo do expoente real	-126 a $+127$	-1022 a $+1023$
Precisão (dígitos decimais)	Aproximadamente 7	Aproximadamente 16

Exemplo: Representação em Precisão Simples

Considere o número decimal $x = -12{,}25$. Sua representação em binário é:

$$x = -1100{,}01_2 = -1{,}10001 \times 2^3$$

Formato:

- Sinal: $s = 1$
- Mantissa (sem o bit oculto): 10001000000000000000000
- Expoente: $e = 3 + 127 = 130 = 10000010_2$

Portanto, o número seria representado, em binário de 32 bits, como:

$$\boxed{1\ 10000010\ 10001000000000000000000}$$

s_n	e	m
1	10000010	10001000000000000000000

Exemplo: Representação em Precisão Dupla

Vamos representar o número decimal $x = 12{,}375$ em ponto flutuante com **precisão dupla (64 bits)**.

1. Conversão para binário:

$$12{,}375_{10} = 1100{,}011_2 = 1{,}100011 \times 2^3$$

2. Identificação dos componentes:

- **Sinal (s)**: Como o número é positivo, $s = 0$
- **Expoente real (e)**: 3
- **Bias**: Para precisão dupla, bias = 1023
- **Expoente com bias**: $e + \text{bias} = 3 + 1023 = 1026$
- **Expoente em binário (11 bits)**: $1026_{10} = 10000000010_2$
- **Mantissa (m)**: Os bits após o ponto da parte fracionária normalizada: $100011000000\ldots$ (completando até 52 bits)

3. Representação final (64 bits):

$$\boxed{0\ 10000000010\ 1000110000000000000000000000000000000000000000000000}$$

Essa é a representação de 12,375 em ponto flutuante com precisão dupla.

Resumo:

- **Bits de sinal**: 0
- **Bits do expoente**: 10000000010
- **Bits da mantissa**: 100011 seguidos de zeros até completar 52 bits

Considerações

A escolha entre precisão simples e dupla depende da aplicação:

- **Precisão Simples**: adequada para aplicações com memória limitada e que não exigem alta precisão.
- **Precisão Dupla**: usada em aplicações científicas, cálculos de engenharia, simulações e algoritmos numéricos mais sensíveis. Apesar do ganho de precisão, o uso de precisão dupla demanda mais memória e tempo de processamento.

1.2.2 Representação de Números em Sistemas de Ponto Flutuante

Em máquinas que operam em sistemas de ponto flutuante, apenas um subconjunto finito de $\mathbb{R}$ pode ser representado de maneira exata. Por isso, frequentemente, é necessário limitar a quantidade de dígitos significativos na representação de números a fim de adequá-los ao sistema que a máquina opera. Dois dos principais processos empregados para este fim são o **truncamento** e o **arredondamento**.

O truncamento consiste na supressão de todos os dígitos após uma determinada posição, sem qualquer ajuste adicional no último dígito mantido. Formalmente, dado um número real x, sua aproximação truncada com n dígitos na base b é expressa por

$$T(x) = \sum_{i=-k}^{n-1} d_i b^i$$

onde os dígitos d_i com $i < -k$ são descartados.

O erro introduzido por este processo, dado por $E_T = x - T(x)$, é denominado *erro de truncamento*. Ele é limitado superiormente por

$$|x - T(x)| < b^{-k}. \tag{1.3}$$

Exemplo 1.2.1. Considere a aproximação com oito casas decimais de $\pi = 3,14159265$. Para truncar π com precisão de 4 casas decimais, descartamos todos os termos da sexta casa em diante. Assim, o valor truncado fica $T(\pi) = 3,1415$. O erro do truncamento é, nesse caso, $E_T = 3,14159265 - 3,1415 = 0,00009265$. Diante disso, podemos observar que $|E_T| < 10^{-4}$.

O arredondamento, por outro lado, ajusta o último dígito mantido com base no valor do primeiro dígito descartado, buscando minimizar o erro absoluto da aproximação. No arredondamento simétrico (ou clássico), se o primeiro dígito descartado for maior ou igual a $\dfrac{b^{-n}}{2}$, incrementa-se o último dígito mantido em uma unidade; caso contrário, seu valor permanece inalterado.

Seja x um número real e $R(x)$ sua aproximação arredondada com n dígitos na base b. O erro de arredondamento satisfaz:

$$|x - R(x)| \leq \frac{1}{2} b^{-n}$$

Exemplo 1.2.2. Ainda considerando a mesma aproximação de $\pi = 3,14159265$. Para arredondar π com precisão de 4 casas decimais, vamos analisar o número da próxima casa em que queremos arredondar. Nesse caso, o número na quinta posição é 9, então vamos arredondar a quarta casa para cima

$$R(\pi) = 3,1416.$$

O erro do arredondamento é, nesse caso,

$$E_R = 3,14159265 - 3,1416 = -0,00000735.$$

Diante disso, podemos observar que $|E_R| \leq \dfrac{1}{2} 10^{-4}$.

Em geral, o erro máximo introduzido pelo arredondamento é metade daquele introduzido pelo truncamento, razão pela qual o arredondamento tende a produzir aproximações mais precisas.

Para compreendermos melhor as limitações na representação de números em um sistema de ponto flutuante, vamos explorar o exemplo a seguir. Suponha que uma máquina opere no sistema $F(10, 5, -5, 5)$. Nesse sistema, os números serão representados da seguinte maneira

$$\pm(0.d_1d_2...d_t) \times 10^e, 0 \leq |d_j| \leq 9, d_1 \neq 0, e \in [-5, 5]. \tag{1.4}$$

O menor valor, em módulo, representado nesse sistema é

$$m = 0.10000 \times 10^{-5} = 10^{-6},$$

enquanto que o maior é

$$M = 0.99999 \times 10^5 = 99999.$$

$$G = \{x \in \mathbb{R} \mid x \text{ corresponde a um número representado no sistema } \mathcal{F}(\beta, t, L, U)\}$$

@LucasM: aqui vale a pena trocar por algo do tipo pontos flutuantes e inclusive inserir uma imagem mostrando que trata-se de um conjunto discreto

é o conjunto dos números que são representáveis por esse sistema de ponto flutuante. Nesse conjunto m é a mantissa Dado um número real x, as seguintes situações podem ocorrer:

Caso 1: $x \in G$ **(Número representável)**

Seja $x = 12237,76$.

Na forma normalizada temos $x = 1,223776 \times 10^4$. Porém, esse número não pode ser representado precisamente no sistema F e, portanto, precisamos aplicar uma das técnicas de aproximação. Utilizando o truncamento o resultado é $\bar{x} = 1{,}22377 \times 10^4$. Já com o arredondamento $\bar{x} = 1{,}22378 \times 10^4$.

O número está dentro da faixa de expoente permitida e é representável com perda controlada de precisão.

Caso 2: $|x| < m$ **(Underflow)**

Seja $x = 0{,}582 \times 10^{-6}$.

O expoente é -6 , menor que o limite inferior $L = -5$. Portanto, o número não pode ser representado e ocorre **underflow**. Nesse caso, na maioria das vezes o valor é tratado como zero.

Caso 3: $|x| > M$ **(Overflow)**

Seja $x = 0{,}927 \times 10^{6}$

O expoente é $+6$, maior que $U = 5$. Portanto, ocorre **overflow** e o número não pode ser representado precisamente. Neste caso, o valor pode ser tratado como infinito ou como uma flag indicando o **overflow**.

Vejamos a seguir uma comparação da técnicas de arredondamento e de truncamento.

Considere uma máquina decimal com 3 dígitos na mantissa e expoentes variando de –4 a 4:

Número Real	**Arredondamento**	E_R	**Truncamento**	E_T
5,678	$0{,}568 \times 10^{1}$	$0{,}2 \times 10^{-3}$	$0{,}567 \times 10^{1}$	$0{,}8 \times 10^{-3}$
–192,73	$-0{,}193 \times 10^{3}$	$0{,}27 \times 10^{1}$	$-0{,}192 \times 10^{3}$	$0{,}73 \times 10^{1}$
3,14159	$0{,}314 \times 10^{1}$	$0{,}159 \times 10^{-2}$	$0{,}314 \times 10^{1}$	$0{,}159 \times 10^{-2}$
0,0000063	Underflow			
920000,0	Overflow			

1.2.3 Representação Especial do Zero

Na representação de ponto flutuante, um número real é geralmente expresso na forma normalizada:

$$N = \pm(d_1{,}d_2d_3 \ldots d_t)_\beta \times \beta^e,$$

com $d_1 \neq 0$, garantindo o aproveitamento máximo da precisão disponível e evitando representações redundantes. Contudo, o número zero não pode ser representado nesta forma, pois exigiria $d_1 = 0$, o que contraria a normalização.

Assim, o zero recebe uma *representação especial* denotada por

$$N = \pm(0{,}000\ldots0_t)_\beta \times \beta^{L-1},$$

em que L é o menor expoente permitido no sistema. Este tratamento especial assegura que o zero seja manipulado de forma única e consistente dentro do sistema de ponto flutuante, evitando, assim, a perda de informação ao realizar operações aritméticas que o envolvam. Essa perda de informação será discutida na seção 1.3.

1.3 Erros e Limitações

TODO: adicionar nessa seção uma subseção falando sobre epsilon de maquina, contextualizando para o experimento com as ULPS nos próximos capitulos

Erros em operações com pontos flutuantes podem se propagar e aumentar em cálculos mais complexos. Por exemplo, pequenos erros de arredondamento em etapas iniciais podem afetar significativamente o resultado final, especialmente em somas repetitivas ou subtrações de números muito próximos. Isso torna importante considerar a ordem das operações e o impacto da precisão em aplicações sensíveis.

1.3.1 Erro Absoluto e Relativo

Em cálculos numéricos, frequentemente lidamos com aproximações devido a arredondamentos e truncamentos. Para avaliar a precisão dessas aproximações, utilizamos as métricas de erro absoluto e erro relativo.

O erro absoluto mede a diferença entre o valor real $\boldsymbol{x_r}$ e o valor aproximado $\boldsymbol{x_a}$, ou seja, a quantidade exata de erro na aproximação. Ele é definido como:

$$\text{EA} = |x_a - x_r|. \tag{1.5}$$

Quanto menor for o erro absoluto, mais próximo o valor aproximado está do valor real. No entanto, essa métrica não fornece informações diretas sobre o impacto do erro em relação à magnitude do número em questão.

O erro relativo indica o quão preciso é o valor aproximado em relação ao valor real. Ele pode ser tratado como um decimal ou na forma de porcentagem. Ele é definido como

$$\text{ER} = \frac{|x_a - x_r|}{|x_r|} \quad \text{ou} \quad \frac{|x_a - x_r|}{|x_r|} \times 100\%. \tag{1.6}$$

Isso pode ser reescrito como

$$\text{ER} = \frac{EA}{|x_r|} \quad \text{ou} \quad \frac{EA}{|x_r|} \times 100\%. \tag{1.7}$$

Essa métrica é útil quando lidamos com valores de grandezas muito diferentes. Por exemplo, um erro absoluto de 0.1 pode ser insignificante se estivermos tratando de números na ordem de milhares, mas pode ser relevante se estivermos lidando com valores decimais.

Vamos analisar dois casos com mesmo erro absoluto mas diferentes erros relativos. Considere um valor real $x_{r1} = 10.5$ com uma aproximação de $x_{a1} = 10.3$. O seu erro absoluto é $|10.3 - 10.5| = 0.2$ e seu erro relativo $\dfrac{0.2}{10.5} \approx 0.019$ (ou 1.9%). Agora considere um segundo valor real de $x_{r2} = 0.4$ com uma aproximação de $x_{a2} = 0.2$, seu erro absoluto é $|0.4 - 0.2| = 0.2$, o seu erro relativo será $\dfrac{0.2}{0.4} = 0.5$ ou seja, 50%.

Número Real	**Aproximação**	**Erro Absoluto**	**Erro Relativo**
10.5	10.3	0.2	0.019 ou 1.9%
0.4	0.2	0.2	0.5 ou 50.0%

Apesar dos dois casos terem o mesmo erro absoluto ($EA = 0.2$), o erro relativo relacionado à aproximação no primeiro caso representa menos de 2% do valor real, indicando que a aproximação é razoavelmente precisa. Em contrapartida, no segundo caso, o erro relativo é de 50%, um valor relativamente alto comparado ao primeiro, o que indica uma aproximação deficiente. Dessa forma, o erro relativo é uma ferramenta importante para interpretar melhor a qualidade de uma aproximação, independentemente da escala dos números envolvidos.

1.4 Perda de Significância em Operações com Pontos Flutuantes

A perda de significância (também conhecida como cancelamento catastrófico) ocorre de modo mais evidente quando há grande diferença de ordem de grandeza entre os números envolvidos na operação. Por exemplo, considere:

$$x = 1{,}000 \times 10^4 \quad \text{e} \quad y = 0{,}276 \times 10^{-2}.$$

Para realizar a soma, ambos os operandos precisam ser expressos com o mesmo expoente:

$$x = 1{,}000000000 \times 10^4, \quad y = 0{,}000000276 \times 10^4.$$

A soma exata seria:

$$x + y = (1{,}000000000 + 0{,}000000276) \times 10^4 = 1{,}000000276 \times 10^4.$$

Contudo, devido à precisão limitada do sistema (7 dígitos significativos), a aritmética de ponto flutuante armazena apenas:

$$x + y \approx 1{,}0000002 \times 10^4.$$

Uma parte do termo y é completamente desprezada, e a soma não resulta numericamente igual a $x + y$, evidenciando a perda catastrófica de significância.

Outro caso peculiar é a soma de um número muito grande com uma sequência de números pequenos. Dependendo da ordem em que as somas são realizadas, o número grande pode "mascarar"os pequenos, resultando em diferentes valores finais.

Por exemplo:

$$S = 10^8 + 10^{-1} + 10^{-2} + 10^{-3} + \ldots + 10^{-10}.$$

Se somarmos primeiro o número grande (10^8) e depois os números pequenos, muitos destes podem ser ignorados devido à falta de precisão da mantissa. Por outro lado, ao somar os números pequenos antes, o valor final será mais próximo do esperado.

Para ilustrar, suponha a seguinte ordem de cálculo:

- Caso 1: $S = 10^8 + (10^{-1} + 10^{-2} + \ldots + 10^{-10})$.
- Caso 2: $S = (10^{-1} + 10^{-2} + \ldots + 10^{-10}) + 10^8$.

No primeiro caso, muitos números pequenos são ignorados devido ao arredondamento. No segundo, o somatório dos números pequenos é calculado antes de adicionar o número grande, preservando mais informações significativas.

Prevenção da perda de significância em operações com Zero

O zero possui uma representação especial na aritmética de ponto flutuante devido à sua importância nas operações numéricas e à necessidade de evitar ambiguidades. Essa representação permite o tratamento adequado de operações que envolvem valores nulos, prevenindo erros numéricos e a perda de significância em cálculos sensíveis. Considere um sistema de ponto flutuante $F(10, 7, -5, 5)$. Sejam

$$x = 0{,}000 \times 10^{-6} \quad \text{(o número zero)}, \quad y = 0{,}276 \times 10^{-2}.$$

Na operação de soma:

$$x + y = 0{,}276 \times 10^{-2},$$

como x é exatamente zero, o resultado mantém integralmente os dígitos significativos de y.

@EnzoR: @LucasM por favor, verifique se o exemplo ficou bom

Se o zero não tivesse uma representação especial e fosse tratado como um número subnormal com expoente mínimo, o alinhamento das mantissas poderia comprometer a precisão de y, deslocando seus dígitos significativos e resultando em perda de informação. Nesse sentido, suponha que x não tivesse uma representação especial e fosse tratado como um número subnormal com o expoente mínimo -5, com mantissa ajustada para

$$x' = 0{,}000000 \times 10^{-5}.$$

Para somar x' e y, é necessário alinhar os expoentes, o que implica deslocar a mantissa de y para a direita:

$$y = 0{,}276 \times 10^{-2} = 0{,}00000276 \times 10^{-5}.$$

Ao realizar a soma,

$$x' + y = (0{,}000000 + 0{,}00000276) \times 10^{-5} = 0{,}00000276 \times 10^{-5}.$$

Devido à precisão limitada do sistema (7 dígitos significativos), essa operação pode fazer com que os dígitos significativos de y sejam deslocados para posições menos precisas, causando perda de informação relevante.

$$x' + y = 0{,}0000027 \times 10^{-5}.$$

Portanto, a representação especial do zero evita esse problema, preservando a precisão e assegurando a estabilidade numérica das operações.

1.5 Análise de Instabilidades e Casos Peculiares

@EnzoR: @LuisD preciso que verifique se a seção está coerente e se os exemplos estão corretos.

Na aritmética de ponto flutuante, certos casos resultam em erros devido à limitação da precisão e à maneira como os números são representados.

1.5.1 Units in Last Place (ULP's)

Diferentemente dos conjuntos matatemáticos comuns ($\mathbb{R}, \mathbb{Z}$, etc.), todo conjunto de números de um sistema de ponto flutuante (PF) é, por natureza, discreto, ou seja, possui uma quantidade finita de números representáveis. Para cada número real x existe um X que será a sua representação num dado sistema de PF. Definimos $ulp(x)$ pela lacuna entre dois números X_1 e X_2 mais próximos de x, mesmo que x seja um deles. Tome o sistema $F(\beta = 2, t = 3, L = 3, U = 5)$. O menor número que pode ser representado nesse sistema é $1{,}00_2$ x $2^3 = 8_{10}$, o seu sucessor é $1{,}01_2 = 1{,}25_{10}$ x $2^3 = 10_{10}$. Em $x \in [8_{10}, 10_{10}]$ ulp $(x) = 2$.

Para definir para qual PF x o número real x será aproximado, existem várias estratégias, como o truncamento e o arredondamento. A escolha da estratégia de aproximação pode afetar o valor de ulp(x). Por exemplo, se $x = 9{,}6_{10}$, no sistema $F(\beta = 2, t = 3, L = 3, U = 5)$, o valor de ulp($x$) é 2. Se usarmos truncamento, $X = 8_{10}$ e o erro absoluto é $|x - X| = 1{,}6_{10}$, ou seja, $0{,}8 \times ulp(x)$. Se usarmos arredondamento, $X = 10_{10}$ e o erro absoluto é $|x - X| = 0{,}4_{10}$, ou seja, $0{,}2 \times ulp(x)$.

TODO: Adidicionar gráfico ilustrando as ulps

A seguir, descrevemos dois exemplos clássicos que ilustram essas instabilidades.

1.5.2 Imprecisão de operações de Ponto flutuante

Considere o cálculo de $f(x) = x^{10} + 1 - x^{10}$ para $x \in [-60, 60]$. As partes envolvendo x são variáveis e a parte "1"é o literal, ou seja, um valor fixo. Analiticamente, o resultado deveria ser exatamente 1. No entanto, em implementações numéricas, pequenas imprecisões na representação de x^{10} podem levar a resultados instáveis, especialmente para valores de $|x|$ além de um limiar. Isso ocorre devido ao erro relacionado às operações de ponto flutuante, como mostrado na Figura 1.1.

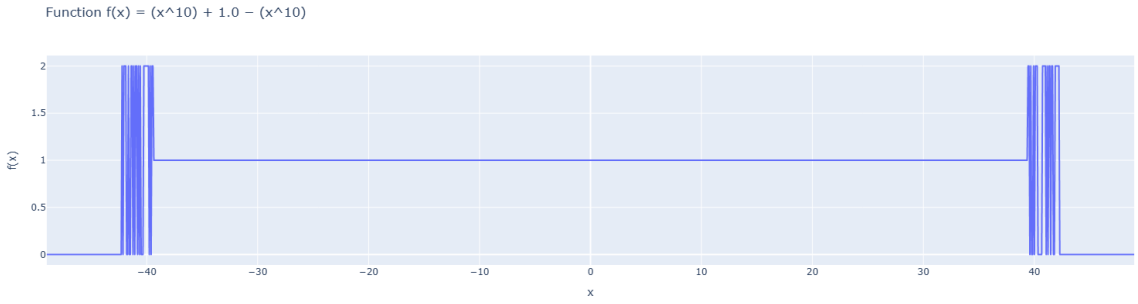


Figura 1.1: Comportamento da expressão $x^{10} + 1 - x^{10}$ no intervalo $[-40, 40]$.

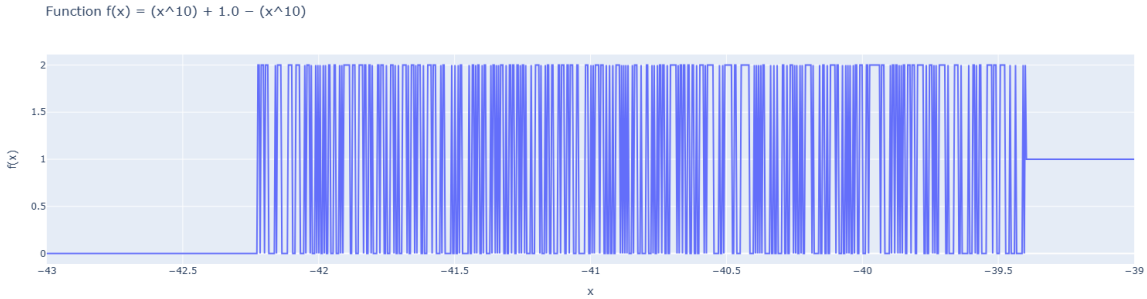


Figura 1.2: Comportamento da expressão $x^{10} + 1 - x^{10}$ no intervalo $[-43, -39]$.

Podemos observar que em um determinado limiar, a função para de se comportar como esperado $f(x) = 1$ e passa a assumir valores os de $f(x) = 2$ e $f(x) = 0$.

Vamos manipular essa expressão e ver como ela se comporta em diferentes situações.

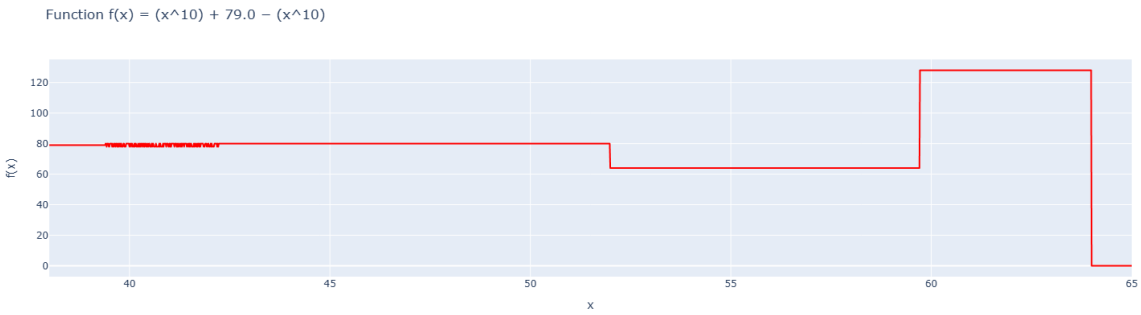


Figura 1.3: Comportamento da expressão $x^{10} + 79 - x^{10}$ no intervalo $[38, 65]$.

Na figura 1.3. é possível observar que a função assume mais de dois valores inesperados para $f(x)$, nesse caso, o conjunto imagem no intervalo $[38, 65]$ é $Im(f) = \{0{,}64{,}78{,}79{,}80{,}128\}$. O seguinte conjunto de valores para o literal foi testado $\{3, 12, 79, 98\}$, e acreditamos que para valores que não podem ser escritos como uma potência de base 2, esse padrão ocorra.

TODO: adicionar explicação

Explicação: blablabla

Uma dúvida comum é se a precisão desses sistemas afeta nessa expressão. Vamos comparar então um sistema de float32 (Cerca de 7-8 bits dígitos decimais de precisão) a um sistema float64 (Cerca de 15-16 dígitos decimais de precisão)

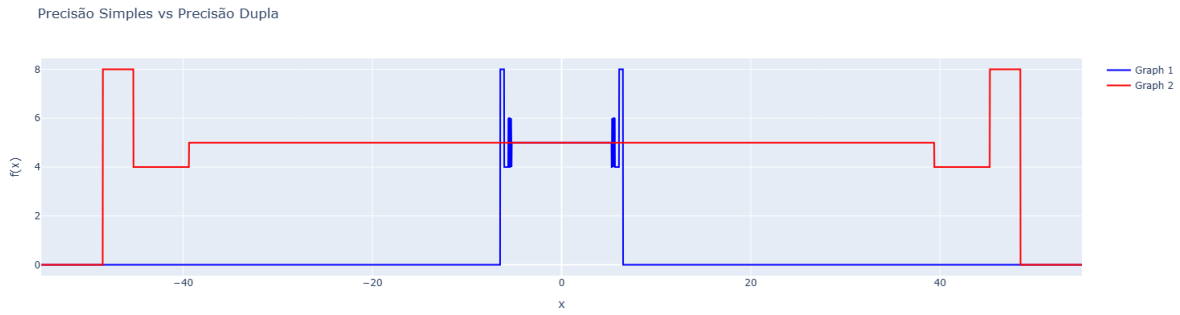


Figura 1.4: Comportamento da expressão $x^{10} + 2 - x^{10}$ no intervalo $[-40, 40]$.

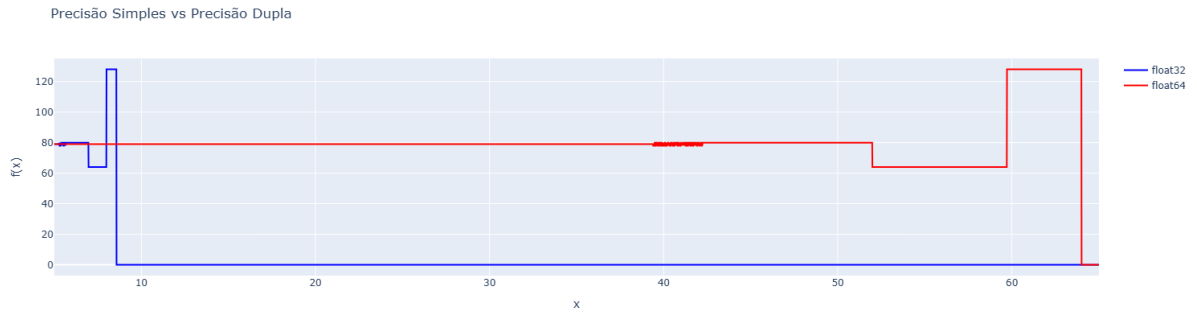


Figura 1.5: Comportamento da expressão $x^{10} + 79 - x^{10}$ no intervalo $[0, 60]$.

Dos testes experimentais, exemplificado nas figuras 1.4 e 1.5, foi concluído que o $|x|$ onde a instabilidade ocorre é menor é menor na precisão dupla.

Outro caso interessante é quando analisamos a função $f(x) = (x \times x \times x \times x \times x \times x \times x \times x \times x \times x) - x^{10}$. Vejamos o gráfico dessa função.

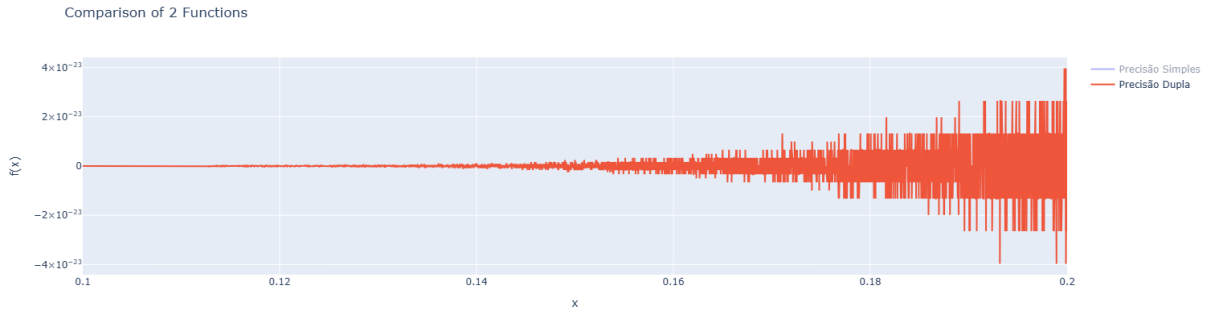


Figura 1.6: Comportamento da expressão $f(x) = (x \times x \times x \times x \times x \times x \times x \times x \times x \times x) - x^{10}$, com precisão dupla, no intervalo $[0{,}1, 0{,}2]$.

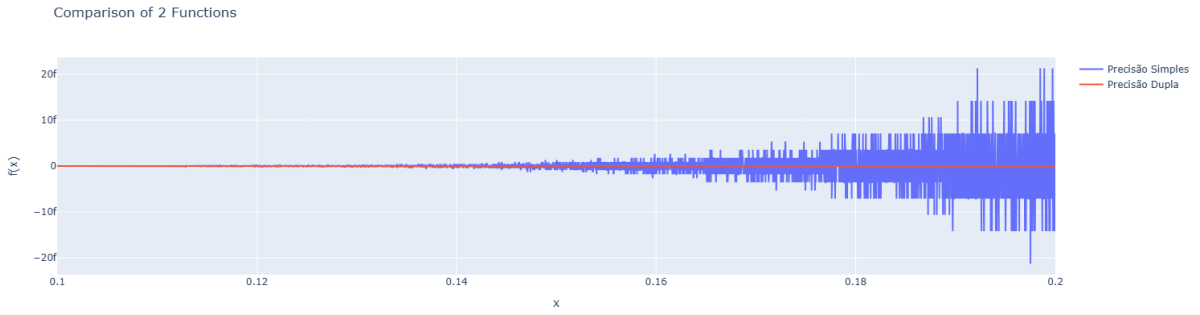


Figura 1.7: Comportamento da expressão $f(x) = (x \times x \times x \times x \times x \times x \times x \times x \times x \times x) - x^{10}$, com precisão simples, no intervalo $[0{,}1, 0{,}2]$.

A instabilidade na precisão dupla (Figura 1.7) produz imagem em torno de $[-4 \times 10^{-23}, 2 \times 10^{-23}]$, enquanto na precisão simples (Figura 1.6) está em torno

de $[-20^{-15}, 20^{-15}]$ ($[-20femto, 20femto]$). Observa-se, portanto, que a diferença de instabilidade é extremamente menor na precisão dupla em comparação com a simples.

Uma outra situação é a diferença de como o computador interpreta algumas funções se forem reescritas de maneiras diferentes. Seja $p(x) = (x-1)^6$ e $q(x) = x^6 - 6x^5 + 15x^4 - 20x^3 + 15x^2 - 6x + 1$, analiticamente essas funções são idênticas, porém existem problemas de cancelamento catastrófico na hora de analisarmos ambas em um sistema de ponto flutuante.

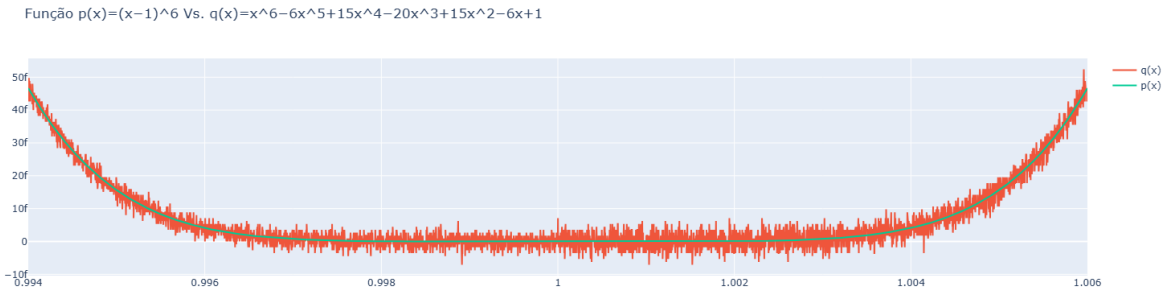


Figura 1.8: Comparação entre as funções $p(x) = (x-1)^6$ e $q(x) = x^6 - 6x^5 + 15x^4 - 20x^3 + 15x^2 - 6x + 1$.

TODO: adicionar explicação

Explicação: yada yada yada

1.5.3 Discussão

Esses exemplos destacam a importância da ordem das operações e da análise cuidadosa ao trabalhar com algoritmos numéricos. Técnicas como a reordenação de cálculos e o uso de formatos de precisão estendida podem ajudar a minimizar esses erros em contextos críticos.

Capítulo 2

Métodos Iterativos para Zeros de Função

Em muitas aplicações, as soluções buscadas se resumem a encontrar os zeros (ou raízes) de uma função. Entretanto, nem sempre é possível fazê-lo analiticamente, devido à natureza das componentes envolvidas na função como, por exemplo, funções polinomiais a partir do 3^{o} grau, somas de funções trigonométricas e logarítmicas, entre outras. Nesse ínterim, recorremos então a maneiras de obter valores aproximados para tais raízes.

Uma classe de métodos utilizados para aproximar raízes de funções são os **métodos iterativos**. A essência desses métodos está em, partindo de um chute inicial e de uma função apropriada φ, obter uma sequência x_k onde cada termo é obtido do anterior recursivamente como $x_{k+1} = \varphi(x_k)$. Essa sequência, sob certas hipóteses, converge para a raiz ξ da função.

Ao longo do capítulo, reservaremos o símbolo ξ para representar raízes de funções.

2.1 Localização de Raízes

Nos métodos que trataremos nesse capítulo, para garantir a convergência da sequência iterativa, é necessário que o primeiro termo esteja suficientemente próximo da raiz e, desse modo, faz-se necessário restringir as funções a intervalos que contenham raízes. Quando as funções envolvidas são contínuas, o resultado a seguir garante a existência de raízes em um intervalo [a,b] desde que as imagens dos extremos tenham sinais opostos.

Proposição 2.1.1. *Seja $f(x)$ uma função contínua no intervalo $[a, b]$. Se $f(a)f(b) < 0$, então há pelo menos uma raiz $\xi \in (a, b)$. Se, além disso, existir $f'(x)$ e $f'(x)$ preservar o sinal em (a, b), então a raiz é única.*

Por exemplo, considere a função $f(x) = x^3 - 9x + 3$. Utilizando a Proposição 2.1.1, observamos que

- $f(0)f(1) = -15 < 0$, portanto há raiz no intervalo (0, 1);
- $f(2)f(3) = -21 < 0$, portanto há raiz no intervalo (2, 3).

Além disso, a derivada de $f(x)$ é $f'(x) = 3x^2 - 9$, cujas raízes são $\pm\sqrt{3}$, então nos intervalos (-∞, -$\sqrt{3}$), (-$\sqrt{3}$, $\sqrt{3}$) e ($\sqrt{3}$, ∞) o sinal da derivada é preservado. Portanto, nos intervalos $(0, 1)$ e $(2, 3)$ há apenas uma raiz.

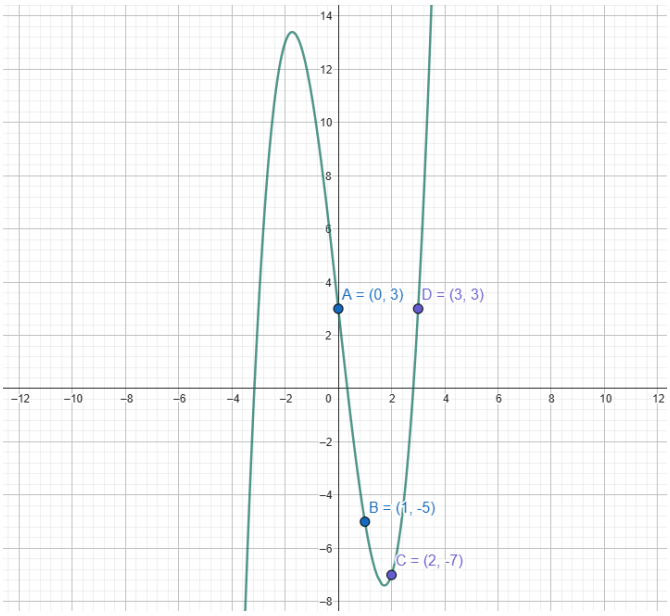


Figura 2.1: Raiz entre A e B, e entre C e D

Outra forma de localizar raízes de uma dada função $f(x)$ é escrevê-la como a diferença entre as funções $g(x) - h(x)$, pois se $f(\xi) = 0$ temos que $g(\xi) - h(\xi) = 0$ ou, equivalentemente, $g(\xi) = h(\xi)$. Graficamente, ξ é a abscissa do ponto de interseção entre as funções g(x) e h(x).

Da mesma função, podemos obter, por exemplo, as interseções entre x e $x^3 - 8x + 3$

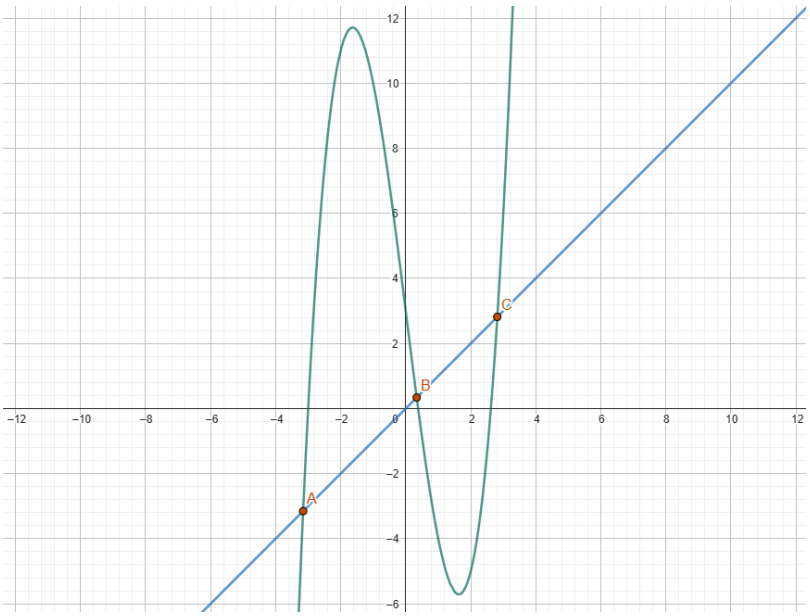


Figura 2.2: Interseções entre x e $x^3 - 8x + 3$

2.2 Critério de Parada

FIXME: Completar

@DanielP: gráficos estilo Vera

O processo é repetido até que a diferença entre duas iterações consecutivas seja inferior a uma tolerância pré-estabelecida ou até que um número máximo de iterações seja atingido.

- $|\overline{x} - \xi| < \epsilon$
- $f(\overline{x}) < \epsilon$

@LucasM: Definir raiz aproximada

2.3 Método do Ponto Fixo

O método do ponto fixo *é um método iterativo* que transforma o problema de buscar as raízes de uma função $f(x)$ no problema de encontrar os pontos fixos de uma outra função $\varphi(x)$, denominada de **função de iteração de ponto fixo**. A partir dessa função de iteração, uma sequência é construída recursivamente começando em um valor inicial x_0 que convergirá para a raiz ξ de $f(x)$, desde que sejam observadas certas condições sob a função $\varphi(x)$ e o dado inicial x_0.

O primeiro passo é gerar funções de iteração φ para $f(x)$, o que pode ser feito isolando x na equação $f(x) = 0$. Por exemplo, manipulando a função $x^3 - 9x + 3$ da seguinte forma

$$x^3 - 8x + 3 = x$$

obtemos a função de iteração $\varphi(x) = x^3 - 8x + 3$. Com a mesma lógica, outras possíveis funções de iteração para f são

a) $\varphi_1(x) = \dfrac{x^3}{9} + \dfrac{1}{3}$

b) $\varphi_2(x) = \sqrt[3]{9x - 3}$

c) $\varphi_3(x) = \dfrac{9}{x} - \dfrac{3}{x^2}$

d) $\varphi_4(x) = \sqrt{9 - \dfrac{3}{x}}$

e) $\varphi_5(x) = -\sqrt{9 - \dfrac{3}{x}}$

f) $\varphi_6(x) = x^3 - 8x + 3$

A forma geral da função de iteração é

$$\varphi(x) = x + A(x)f(x) \tag{2.1}$$

com $A(\xi) \neq 0$. Por exemplo, a $\varphi_1(x) = \dfrac{x^3}{9} + \dfrac{1}{3}$ na forma geral ficaria

$$\varphi_1(x) = x + \frac{1}{9}f(x)$$

em que $A(x) = \dfrac{1}{9}$. Nesse caso, pode-se observar que $\mathrm{A}(\xi) \neq 0$.

O resultado a seguir relaciona a raiz de uma função com pontos fixos de uma função de iteração associada a essa função.

Proposição 2.3.1. *Seja ξ uma raiz de uma função $f(x)$ e seja $\varphi(x)$ uma função de iteração associada a $f(x)$. Então, $f(\xi) = 0$ se, e somente se, $\varphi(\xi) = \xi$.*

Demonstração. ($\Rightarrow$) Pela forma geral da função de iteração temos que $\varphi(\xi) = \xi + A(\xi)f(\xi)$. Uma vez que $f(\xi) = 0$, então $\varphi(\xi) = \xi$.

($\Leftarrow$) Começando novamente pela forma geral da função de iteração, temos que $\varphi(\xi) = \xi + A(\xi)f(\xi)$. Como $\varphi(\xi) = \xi$, concluímos que $A(\xi)f(\xi) = 0$. Tendo como hipótese que $A(\xi) \neq 0$, então $f(\xi) = 0$. □

@LucasM: Aqui, antes de ir para o resultado principal, dar exemplos de funções de iteração que fazem a sequencia convergir e divergir. Inserir gráficos assim como no livro da Vera.

@DanielP: ok! uma redemoinho (φ_6: $[-\sqrt{3}, -\sqrt{\frac{7}{3}}]$), uma escadinha($\varphi_1$: $[0, \sqrt{3}]$) e uma divergente(φ_4 e φ_5; φ_2 e φ_3 I)

Sob condições a respeito da função de iteração, sua derivada e o dado inicial, a convergência da sequência iterativa é garantida, como pode-se observar a seguir.

Teorema 2.3.1. ***Convergência da Sequência Iterativa***
Seja ξ uma raiz de f(x), isolada num intervalo I centrado nessa raiz. Considere uma função de iteração $\varphi(x)$ associada a f(x). Sob as seguintes hipóteses:

i) $\varphi(x)$ *e* $\varphi'(x)$ *são contínuas em I,*

ii) $|\varphi'(x)| \leq M < 1$ *em I,*

iii) $x_0 \in I$*,*

a sequência $x_{k+1} = \varphi(x_k)$ *converge para a raiz* ξ*.*

Demonstração. Como $x_{k+1} = \varphi(x_k)$, subtraindo ξ de ambos os lados da igualdade e usando o fato de que $\varphi(\xi) = \xi$ temos

$$x_{k+1} - \xi = \varphi(x_k) - \varphi(\xi). \tag{2.2}$$

Pelo Teorema do Valor Médio (TVM) podemos escrever

$$\varphi(x_k) - \varphi(\xi) = \varphi'(c_k)(x_k - \xi) \tag{2.3}$$

com c_k entre x_k e ξ. Então, substituindo (2.3) em (2.2), temos

$$\begin{aligned} |x_{k+1} - \xi| &= |(x_k - \xi)\ \varphi'(c_k)| \\ &= |x_k - \xi|\ |\varphi'(c_k)| \\ &< |x_k - \xi| \end{aligned} \tag{2.4}$$

uma vez que $|\varphi'(x)| < 1$. Como $x_0 \in I$, podemos concluir que $x_k \in I$ para todo k já que, por (2.4), $|x_k - \xi| < |x_0 - \xi|$.

Na sequência, provaremos que x_k converge para a raiz ξ. Vamos começar mostrando que

$$|x_1 - \xi| \leq M\ |x_0 - \xi|. \tag{2.5}$$

Observe que, como $x_1 = \varphi(x_0)$, temos que $x_1 - \xi = \varphi(x_0) - \varphi(\xi)$. Pelo Teorema do Valor Médio temos que $\varphi(x_0) - \varphi(\xi) = (x_0 - \xi)\ \varphi'(c_0)$, para algum c_0 entre x_0 e ξ. Uma vez que $|\varphi'(x)| \leq M$ no intervalo I, a seguinte desigualdade é válida

$$\begin{aligned} |x_1 - \xi| &= |x_0 - \xi|\ |\varphi'(c_0)| \\ &\leq M\ |x_0 - \xi| \end{aligned}$$

e provamos a desigualdade (2.5). De modo similar prova-se que $|x_2 - \xi| \leq M\ |x_1 - \xi|$ que, combinado com (2.5), implica que $|x_2 - \xi| \leq M^2\ |x_0 - \xi|$. Repetindo o processo k vezes pode-se concluir que

$$|x_k - \xi| \leq M^k\ |x_0 - \xi|. \tag{2.6}$$

Como $0 < M < 1$, se k tende a infinito, M^k tende a 0 e, portanto, $M^k\ |x_0 - \xi|$ também tende a 0. Assim, provamos que

$$\lim_{k \to \infty} x_k = \xi, \tag{2.7}$$

ou seja, x_k converge para a raiz. □

2.3.1 Ordem de convergência

A ordem de convergência de um método iterativo é uma medida de quão rapidamente a sequência de iterações converge para a solução desejada. Em termos

práticos, isso significa que, se um método tem uma ordem de convergência alta, ele será capaz de reduzir mais o erro de aproximação a cada iteração.

Definição 2.3.1. Seja $\{x_k\}$ uma sequência que converge para ξ e

$$e_k = x_k - \xi \tag{2.8}$$

o erro na k-ésima iteração. Se existirem $p > 1$ e $C > 0$ tais que $\lim_{k\to\infty} \frac{|e_{k+1}|}{|e_k|^p} = C$, então p é chamada de *ordem de convergência* da sequência e C é a *constante assintótica de erro.*

No caso do MPF verificaremos que $p = 1$ portanto a ordem de convergência do método é ao menos linear.

Proposição 2.3.2. *Se*

$$\lim_{k\to\infty} \frac{e_{k+1}}{e_k} = C,\ 0 \leq |C| < 1, \tag{2.9}$$

então a convergência é pelo menos linear. O MPF tem convergência pelo menos linear.

Demonstração. Partindo de (2.2) e (2.3), e tomando o limite com k tendendo a infinito, podemos escrever (2.9) como

$$\lim_{k\to\infty} \frac{x_{k+1} - \xi}{x_k - \xi} = \lim_{k\to\infty} \varphi'(c_k)$$

com c_k entre x_k e ξ . Como por hipótese $|\varphi'(x)| < 1$ então $|C| < 1$, portanto, a convergência do MPF é pelo menos linear. □

2.3.2 Implementações do Método do Ponto Fixo

Imperativa

Algorithm 1 Método do Ponto Fixo Iterativo

1: **procedure** MAIN
2: $x_0 \leftarrow 1.5$ ▷ Chute inicial
3: tol $\leftarrow 10^{-6}$ ▷ Tolerância
4: $N \leftarrow 100$ ▷ Número máximo de iterações
5: Inicializar vetor H de tamanho $N + 1$ ▷ Alocação dinâmica
6: $x \leftarrow x_0$
7: $H[0] \leftarrow x_0$
8: **for** $i \leftarrow 0$ **to** N **do**
9: $x_{\text{next}} \leftarrow \sin(x^2 + \cos(x))$ ▷ Função $g(x)$
10: $H[i + 1] \leftarrow x_{\text{next}}$ ▷ Salva no histórico
11: **if** $|x_{\text{next}} - x| <$ tol **then**
12: **Imprimir** "Raiz: "x_{next}
13: **Imprimir** "Iterações: "$i + 1$
14: **Imprimir** Histórico $H[0 \dots i + 1]$
15: **return** 0 ▷ Sucesso
16: **end if**
17: $x \leftarrow x_{\text{next}}$ ▷ Atualiza para próxima iteração
18: **end for**
19: **Imprimir** "Falha na convergência"
20: Liberar memória de H
21: **return** 1
22: **end procedure**

Funcional

Algorithm 2 Método do Ponto Fixo Recursivo

```
1: function G(x)
2:     return sin(x^2 + cos(x))
3: end function
4: procedure FixedPoint(x_0, tol, maxIters)
5:     function Go(x, iter, acc)                          ▷ Função auxiliar recursiva
6:         if iter ≥ maxIters then
7:             return (x, iter, acc)                      ▷ Limite atingido
8:         end if
9:         x_next ← g(x)
10:        if |x_next − x| < tol then
11:            return (x_next, iter + 1, acc ∪ [x_next])  ▷ Convergiu
12:        else
13:            return Go(x_next, iter + 1, acc ∪ [x_next])  ▷ Recursão
14:        end if
15:    end function
16:    return Go(x_0, 0, [x_0])                           ▷ Chamada inicial com lista [x0]
17: end procedure
```

2.4 Método de Newton-Raphson Unidimensional

TODO: talvez reescrever todas as condições como i) derivada limitada ii) f e g continuas , etc...

@DanielP: conv quadrática

@EnzoR: escolhendo a func de iter

O método de Newton é um caso particular do **Método do Ponto Fixo** amplamente utilizado para encontrar raízes reais de funções não lineares, cuja ordem de convergência é pelo menos quadrática. A função de iteração específica para este método produz uma sequência em que cada termo x_{k+1} corresponde, geometricamente, à **interseção da reta tangente** a $f(x)$ no ponto $(x_k, f(x_k))$ com o eixo x.

2.4.1 Sequência Iterativa

@DanielP: Talvez mudar o nome

No método do ponto fixo, vimos que se a função satisfaz o Teorema 2.3.1 a sequência $x_{k+1} = \varphi(x_k)$ converge para ξ. Além disso, da desigualdade (2.6) temos que quanto menor for $|\varphi'(x)|$, mais rápido a sequência $\{x_k\}$ converge.

Aplicando a derivada na forma geral de φ pela regra da cadeia obtemos $\varphi'(x) = 1 + A'(x)f(x) + A(x)f'(x)$ e, calculando-a na raiz, obtemos $\varphi'(\xi) = 1 + A(\xi)f'(\xi)$. Usando $\varphi'(\xi) = 0$, concluímos que $A(\xi) = \dfrac{-1}{f'(\xi)}$. Generalizando, temos $A(x) = \dfrac{-1}{f'(x)}$. Portanto, desde que $f'(x) \neq 0$, a forma da função de iteração do método de Newton-Raphson é

$$\varphi(x) = x - \frac{f(x)}{f'(x)}. \tag{2.10}$$

Dada uma função $f(x)$, o processo parte de uma estimativa inicial x_0 e aplica a seguinte fórmula iterativa, a partir da escolha da derivada em (2.1)

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, \tag{2.11}$$

onde $f'(x_k)$ é a derivada da função f avaliada em x_k. Para que o método convirja para a raiz correta, é necessário que $f(x)$ seja continuamente diferenciável em uma vizinhança da raiz e que $f'(x) \neq 0$ nessa região. Além disso, a escolha adequada do ponto inicial x_0 é crucial para garantir a convergência do método.

2.4.2 Interpretação Geométrica

@DanielP: ficou paia o alinhamento em

$$\begin{aligned} f'(x_0)(x - x_k) &= y - y_k \\ f'(x_k)(x_{k+1} - x_k) &= -y_k \\ x_{k+1} - x_k &= \frac{-y_k}{f'(x_k)} \\ x_{k+1} &= x_k - \frac{y_k}{f'(x_k)} \end{aligned}$$

2.4.3 Convergência

O método de Newton-Raphson apresenta convergência local quadrática sob certas condições. Isso significa que, se o ponto inicial x_0 estiver suficientemente próximo da raiz simples ξ e f, f' e f'' forem contínuas em uma vizinhança de ξ com $f'(\xi) \neq 0$, então a sequência $\{x_k\}$ definida por $x_{k+1} = x_k - \dfrac{f(x_k)}{f'(x_k)}$ converge para ξ e o erro $e_k = x_k - \xi$ satisfaz

$$|e_{k+1}| \leq C|e_k|^2$$

para algum $C > 0$ e k suficientemente grande. Assim, a cada iteração, o número de dígitos corretos aproximadamente dobra, caracterizando convergência quadrática.

Teorema 2.4.1. *Sejam $f(x)$, $f'(x)$ e $f''(x)$ contínuas num intervalo I que contém a raiz ξ de f, supondo $f'(\xi) \neq 0$. Então, existe um intervalo $\overline{I} \subset I$, contendo a raiz ξ, tal que $x_0 \in \overline{I}$, a sequência x_k gerada pela função de iteração $\varphi(x) = x - \dfrac{f(x_k)}{f'(x_k)}$ convergirá para a raiz.*

Demonstração. Sendo o método de Newton-Raphson um caso particular do MPF, basta provar que para $\varphi(x) = x - \dfrac{f(x)}{f'(x)}$ as hipóteses do Teorema 2.3.1 são satisfeitas.

Primeiramente, observe que

$$\varphi'(x) = \frac{f(x)f''(x)}{[f'(x)]^2}. \tag{2.12}$$

Como $f'(\xi) \neq 0$ e $f'(x)$ é contínua em I, é possível obter $I_1 \subset I$ tal que $f'(x) \neq 0$ no intervalo I_1. Assim, a função f e suas derivadas primeira e segunda são contínuas em I_1 e, consequentemente, a função de iteração e sua derivada também.

Uma vez que a $\varphi'(x)$ é contínua em I_1 e $\varphi'(\xi) = 0$, é possível escolher $I_2 \subset I_1$ de modo que $|\varphi'(x)| < 1$ em I_2 tendo ξ como centro do novo intervalo.

Por fim, tomando, $\overline{I} = I_2$, satisfazem-se as hipóteses do Teorema 2.3.1. □

2.4.4 Ordem de Convergência

No MPF espera-se uma ordem de convergência ao menos linear, entretanto ao escolher uma função de iteração que satisfaça $\varphi'(\xi) = 0$, provaremos que sua ordem de convergência será ao menos quadrática.

Proposição 2.4.1. *A ordem de convergência do método de Newton é pelo menos quadrática.*

Demonstração. Assumiremos todas as hipóteses do Teorema 2.4.1. Partindo de $x_{k+1} = x_k - \dfrac{f(x_k)}{f'(x_k)}$, subtraindo ξ em ambos os lados da igualdade, obtemos

$$e_{k+1} = e_k - \frac{f(x_k)}{f'(x_k)} \tag{2.13}$$

O polinômio de Taylor de grau 2 para $f(x)$ centrado em x_k é

$$f(x) = f(x_k) + f'(x_k)(x - x_k) + \frac{f''(c_k)}{2}(x - x_k)^2$$

com c_k entre x e x_k. Assim, $f(\xi) = f(x_k) - f'(x_k)(x_k - \xi) + \dfrac{f''(c_k)}{2}(x_k - \xi)^2$ dado que $f(\xi) = 0$, dividindo a equação pela derivada de f temos, aplicando a fórmula do erro 2.8 e a da sequência 2.11

$$\begin{aligned} e_k - \frac{f(x_k)}{f'(x_k)} &= \frac{f''(c_k)}{2f'(x_k)}e_k^2 \\ e_{k+1} &= \frac{f''(c_k)}{2f'(x_k)}e_k^2 \\ \frac{e_{k+1}}{e_k^2} &= \frac{1}{2}\frac{f''(c_k)}{f'(x_k)} \end{aligned}$$

Aplicando limite no termo esquerdo da equação acima e, dada a continuidade das funções, nos argumentos à direita, obtemos

$$\lim_{k\to\infty} \frac{e_{k+1}}{e_k^2} = \frac{1}{2}\frac{f''(\xi)}{f'(\xi)}$$

@LucasM: Reescrever isto

@EnzoR: Como?

Calculando a φ'' a partir de φ' 2.12 em ξ obtemos $\frac{[f'(\xi)]^3 f''(\xi)}{[f'(\xi)]^4}$, que é um C concluindo então que a convergência do método de Newton Raphson é ao menos quadrática.

$$\begin{aligned}\lim_{k\to\infty} \frac{e_{k+1}}{e_k^2} &= \frac{1}{2}\varphi''(\xi) \\ &= C\end{aligned}$$

□

2.4.5 Implementações do Método de Newton-Raphson Unidimensional

Imperativa

Algorithm 3 Newton-Raphson Iterativo

```
1: procedure NewtonIter(f, f', x_0, maxIter)
2:     x ← x_0                                   ▷ Inicialização da variável
3:     for i ← 0 to maxIter do
4:         fx ← f(x)
5:         dfx ← f'(x)
6:         x_next ← x − fx/dfx
7:         x ← x_next                            ▷ Atualiza o valor atual
8:     end for
9:     return x
10: end procedure
```

Funcional

Algorithm 4 Newton-Raphson Recursivo

```
1: procedure NewtonRecursive(f, f', x_0, maxIter)
2:     function Iterate(x, n)                          ▷ Função auxiliar (cláusula where)
3:         if n = 0 then
4:             return x                                ▷ Caso base: fim das iterações
5:         else
6:             x_next ← x − f(x)/f'(x)
7:             return Iterate(x_next, n − 1)           ▷ Passo recursivo
8:         end if
9:     end function
10:    return Iterate(x_0, maxIter)                    ▷ Chamada inicial
11: end procedure
```

2.4.6 Casos Atípicos

O Método de Newton é amplamente utilizado na prática devido à sua rapidez e precisão em condições ideais. Contudo, em algumas situações ele pode falhar ou convergir para raízes incorretas se essas condições não forem satisfeitas. Estes problemas geralmente estão associados a fatores como: pontos de máximo e mínimo, pontos de inflexão, multiplicidade da raiz e escolha inadequada do chute inicial.

Caso 1: Alta multiplicidade de raízes

Polinômios de grau elevado tendem a ter alta multiplicidade nas raízes, como no caso do polinômio $x^{10} - 1$. A convergencia lenta é causada pela multiplicidade da raiz. Quando uma raiz tem multiplicidade $m > 1$ — neste caso $m = 10$ — a derivada próxima a raiz se aproxima de zero, aumentando o número de iterações necessárias para convergir para a raiz. Isso faz com que a taxa de convergência do metodo de Newton reduza-se para linear em vez da esperada taxa quadrática.

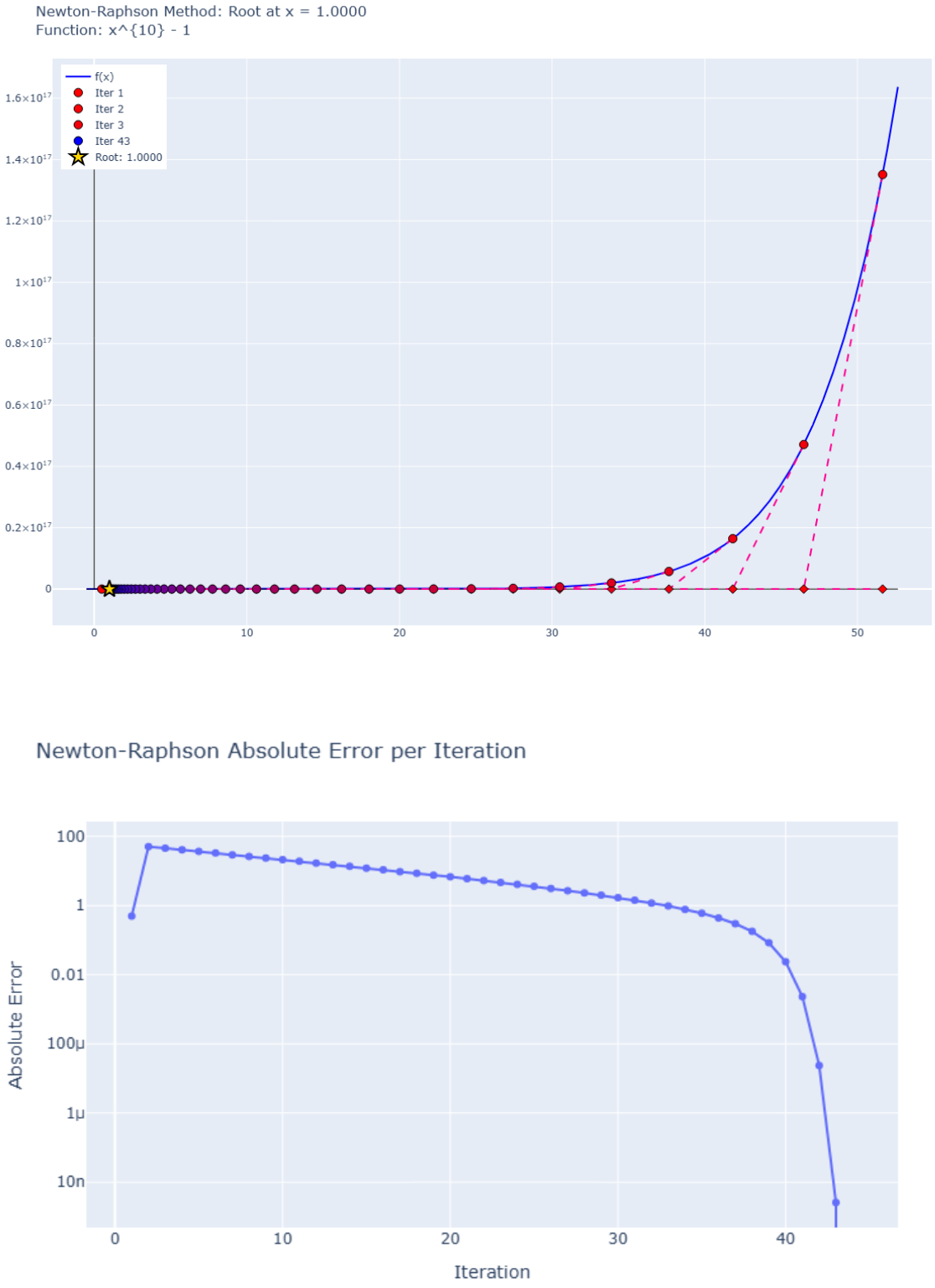


Figura 2.3: Gráfico daCilada relacionada a multiplicidade da raiz.

Ao analisar a função geometricamente, é possível observar que a inclinação da tangente à curva em torno da raiz é muito próxima de zero, o que resulta em saltos minúsculos a cada iteração, causando uma convergencia lenta.

Iteração	Valor de x_k	$f(x)$	Erro e_k
1	$x = 0.50000000000000000$	$f(x) = -0.99902343750000000$	0.50000000000000000
2	$x = 51.64999999999999858$	$f(x) = 135114904483913696$	50.64999999999999858
3	$x = 46.48499999999999943$	$f(x) = 47111654129711536$	45.48499999999999943
4	$x = 41.83650000000000091$	$f(x) = 16426818072478544$	40.83650000000000091
5	$x = 37.65285000000000082$	$f(x) = 5727677301318307$	36.65285000000000082
⋮	⋮	⋮	⋮
43	$x = 1.00000000257760013$	$f(x) = 0.00000002577600156$	0.00000000257760013
44	$x = 1.0$	$f(x) = 0.0$	0.0

Caso 2: Proximidade a extremos de função

Outro fator que é um problema quando se trata de taxa de convergência são os pontos de máximo e mínimo da função. Quando uma iteração é iniciada perto de um ponto de máximo ou mínimo um dos problemas que podem ocorrer é a função de iteração convergir para um ponto de máximo ou mínimo ao invés da raiz. Isso pode causar uma diminuição na taxa de convergência.

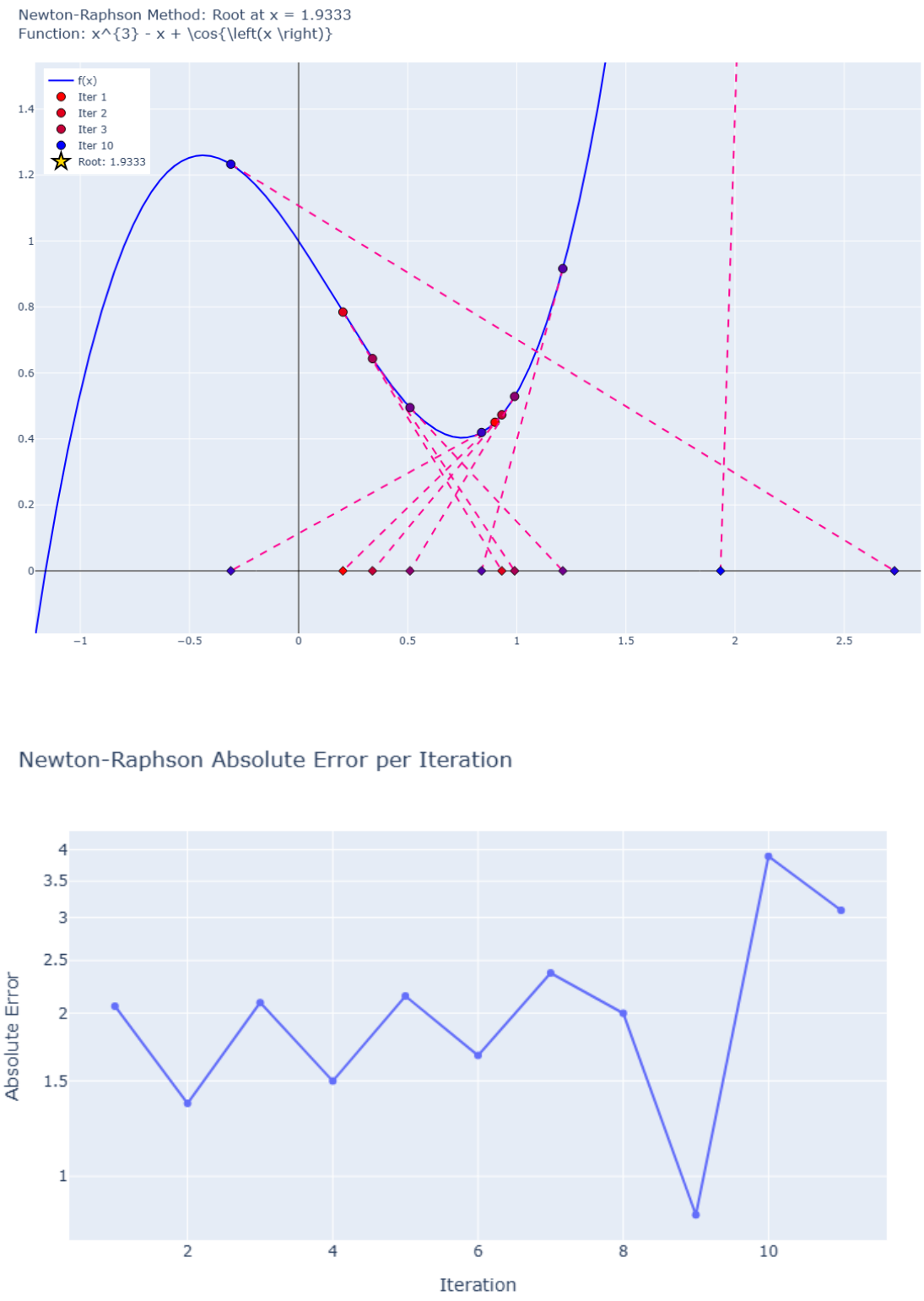


Figura 2.4: Cilada de ponto de Mínimo diminuindo a taxa de convergência.

Iteração	Valor de x_k	$f(x)$	Erro e_k
1	$x = 0.90000000000000002$	$f(x) = 0.45060996827066446$	2.05960580450169983
2	$x = 0.20318738327105890$	$f(x) = 0.78462959591676906$	1.36279318777275882
3	$x = 0.93108680836493152$	$f(x) = 0.47305585213356149$	2.09069261286663144
4	$x = 0.33865524784873213$	$f(x) = 0.64338650465768399$	1.49826105235043205
5	$x = 0.98975277280548202$	$f(x) = 0.52871601875798535$	2.14935857730718194
6	$x = 0.51038365868141022$	$f(x) = 0.49512408543110442$	1.66998946318311026
7	$x = 1.21066335888968579$	$f(x) = 0.91621158441747408$	2.37026916339138571
8	$x = 0.83841140043105933$	$f(x) = 0.41958113044930967$	1.99801720493275914
9	$x = -0.31043606990472872$	$f(x) = 1.23271962647527022$	0.84916973459697120
10	$x = 2.73020451984058488$	$f(x) = 16.70421900417551697$	3.88981032434228480
11	$x = 1.93332993606072678$	$f(x) = 4.93835804267838796$	3.09293574056242671

Em casos extremos, na vizinhança de um ponto de máximo ou mínimo, a intercessão da reta tangente de uma iteração pode coincidir com a coordenada da abscissa da iteração anterior, fazendo com que a sequência não convirja.

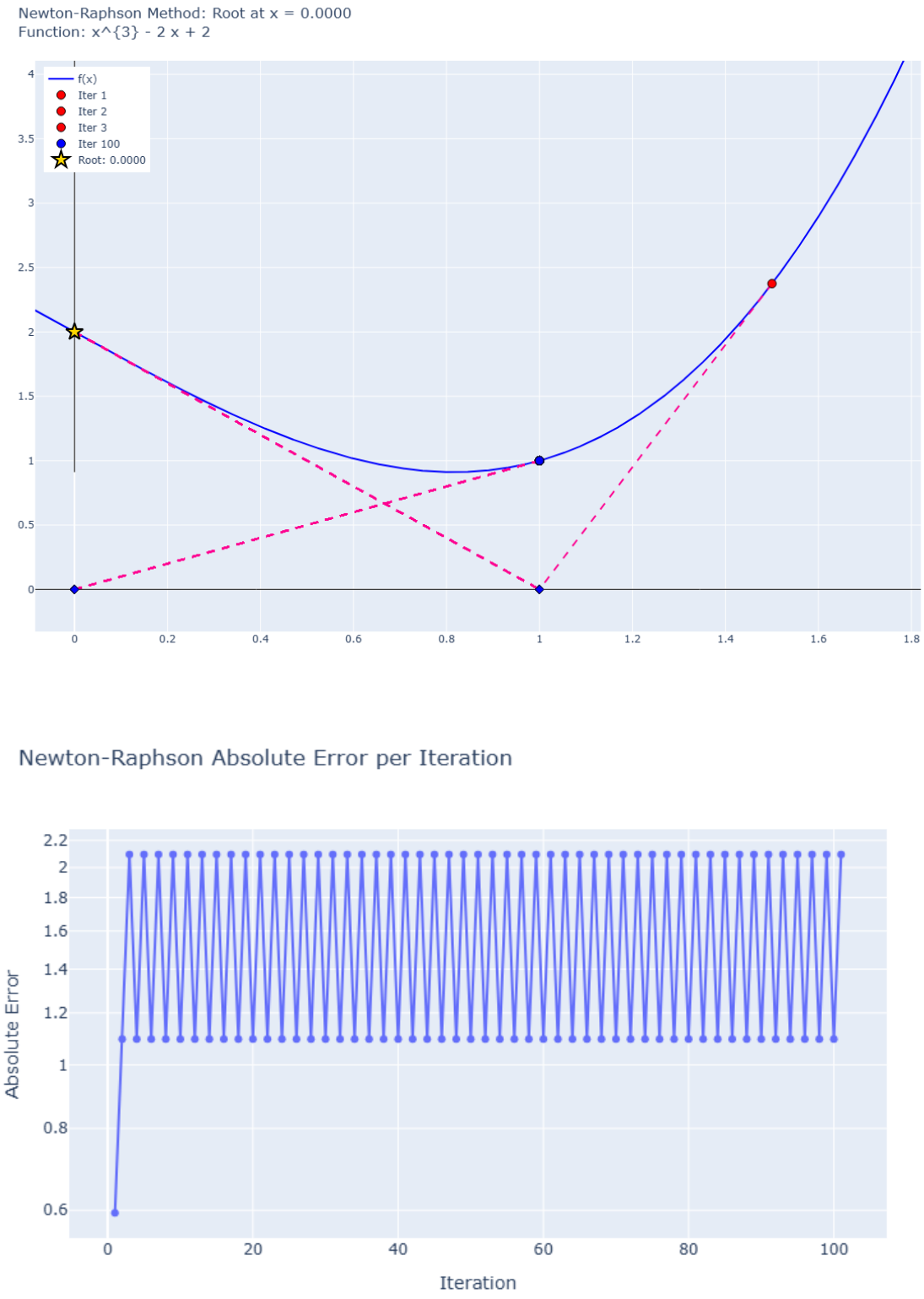


Figura 2.5: Cilada de ponto de Mínimo presa.

Iteração	Valor de x_k	$f(x)$	Erro e_k
1	$x = 1.50000000000000000$	$f(x) = 2.37500000000000000$	0.59455148154232651
2	$x = 1.00000000000000000$	$f(x) = 1.00000000000000000$	1.09455148154232651
3	$x = 0.00000000000000000$	$f(x) = 2.00000000000000000$	2.09455148154232651
4	$x = 1.00000000000000000$	$f(x) = 1.00000000000000000$	1.09455148154232651
5	$x = 0.00000000000000000$	$f(x) = 2.00000000000000000$	2.09455148154232651
⋮	⋮	⋮	⋮
100	$x = 1.00000000000000000$	$f(x) = 1.00000000000000000$	1.09455148154232651
101	$x = 0.00000000000000000$	$f(x) = 2.00000000000000000$	2.09455148154232651

Caso 3: Proximidade a Pontos de Inflexão

Pontos de inflexão também podem causar problemas no método de Newton. Se a raiz estiver próxima de um ponto de inflexão da função $f(x)$, onde a derivada segunda muda de sinal, o método pode não convergir para a raiz desejada. Isso acontece porque a tangente à curva nesse ponto pode não fornecer uma boa aproximação da raiz, resultando em saltos grandes na iteração e afastando-a da raiz.

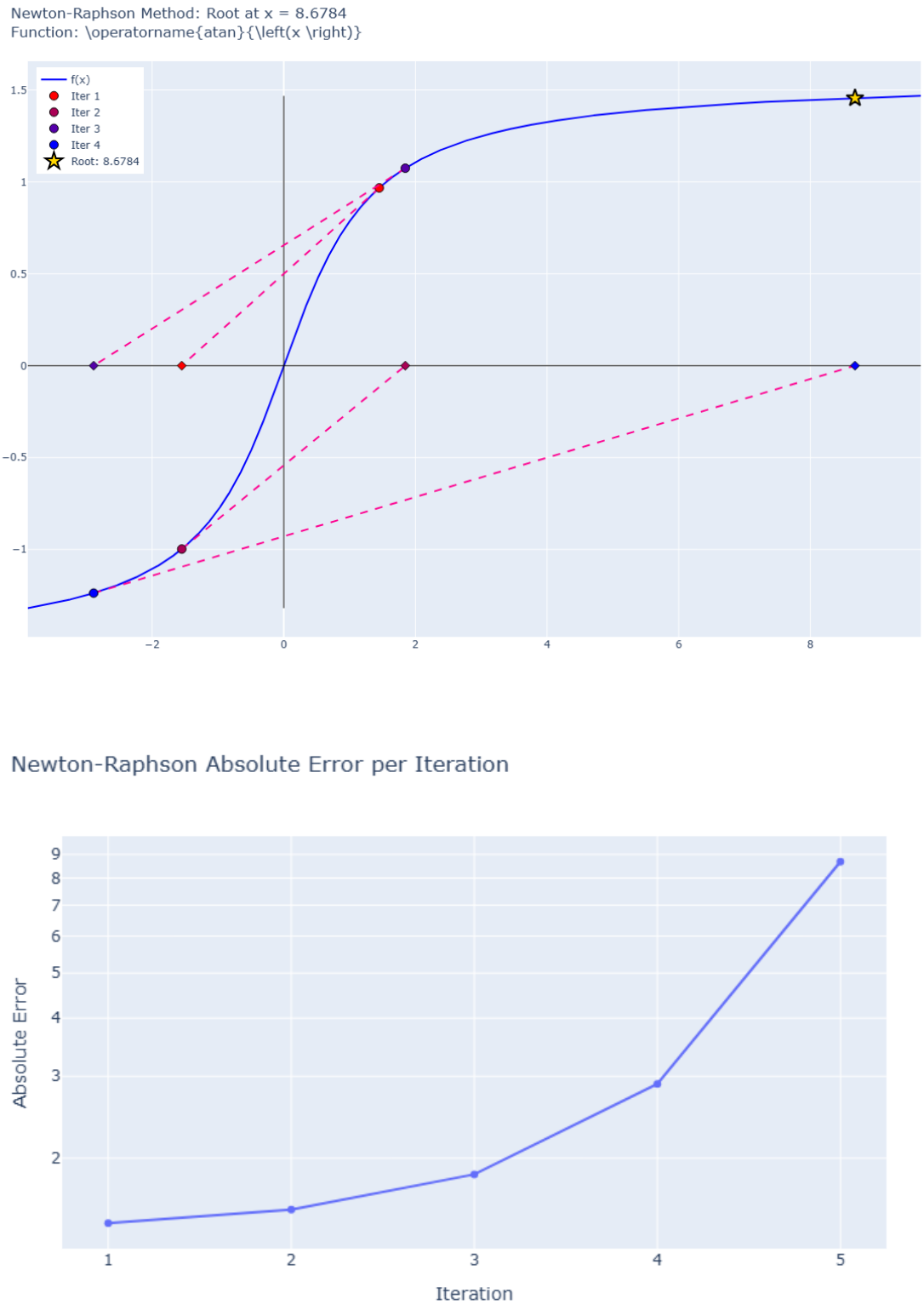


Figura 2.6: Cilada de ponto de Inflexão.

Ao analisar o gráfico é possivel notar que conforme as iterações avançam, a função se afasta da raiz, resultando em saltos cada vez maiores. Isso ocorre porque a tangente à

curva no ponto de inflexão não fornece uma boa aproximação da raiz, levando o método a divergir.

Iteração	Valor de x_k	$f(x)$	Erro e_k
1	$x = 1.44999999999999996$	$f(x) = 0.96704699339746025$	1.44999999999999996
2	$x = -1.55026329701562049$	$f(x) = -0.99790755802460773$	1.55026329701562049
3	$x = 1.84593175119723552$	$f(x) = 1.07432318743154331$	1.84593175119723552
4	$x = -2.88910905408613594$	$f(x) = -1.23757558204700402$	2.88910905408613594
5	$x = 8.67844942653632145$	$f(x) = 1.45607432393228908$	8.67844942653632145

Caso 4: Derivada próxima de zero

Além disso, quando a derivada de uma iteração é muito próxima de zero, o metodo vai ter dificuldades em convergir. Isso ocorre porque a divisão por zero ou por um número muito pequeno pode resultar em saltos grandes na iteração, afastando-a da raiz.

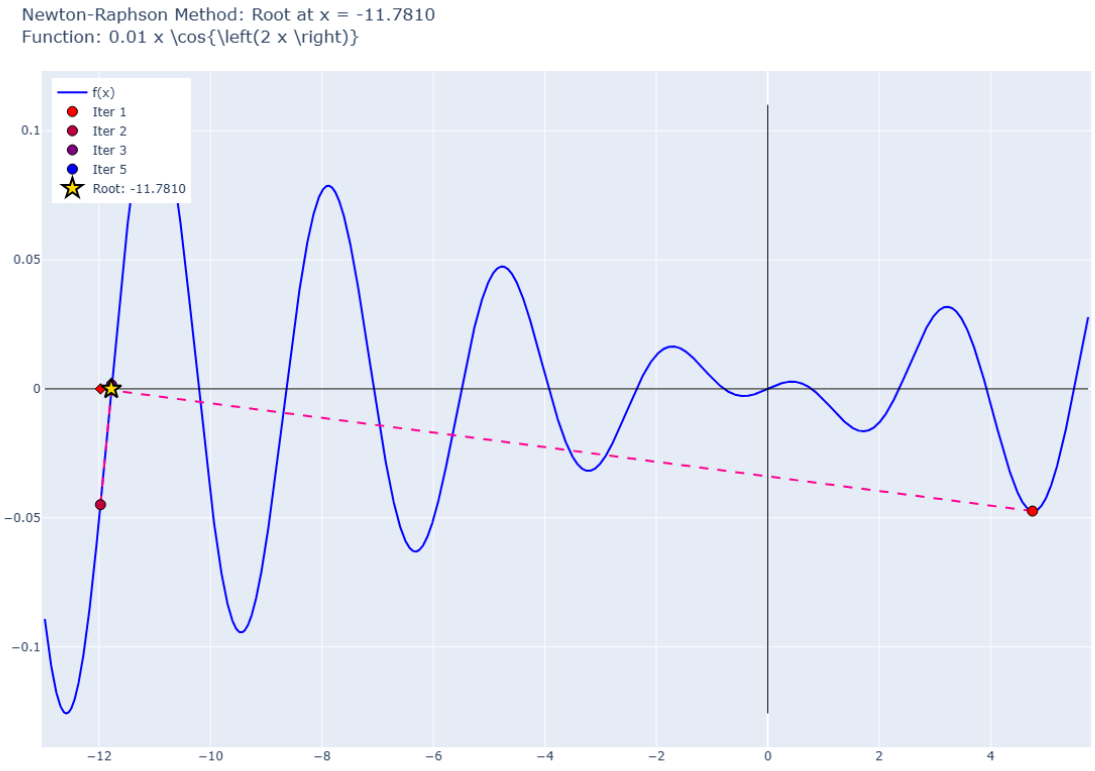


Figura 2.7: Cilada de derivada próxima de zero.

Podemos perceber que a inclinação da tangente à curva em torno da raiz é muito próxima de zero, o que resulta em um salto enorme, levando a iteração para longe da raiz mais próxima.

Iteração	Valor de x_k	$f(x)$
1	$x = 4.75000000000000000$	$f(x) = -0.04736567741932798$
2	$x = -11.97301255116330410$	$f(x) = -0.04486365737362013$
3	$x = -11.77429070649470333$	$f(x) = 0.00157340920400228$
4	$x = -11.78097664313937010$	$f(x) = -0.00000098775893849$
5	$x = -11.78097245096321544$	$f(x) = -0.00000000000035126$
6	$x = -11.78097245096172507$	$f(x) = -0.00000000000000010$

Em resumo, para evitar os problemas apresentados — como convergência lenta, divergência ou oscilação — é fundamental analisar previamente o comportamento da função. Recomenda-se identificar regiões próximas a pontos de máximo, mínimo, inflexão, derivadas próximas de zero e raízes múltiplas. Além disso, a escolha do chute inicial x_0 deve ser feita de modo criterioso: o ponto deve estar suficientemente próximo da raiz desejada e em uma região onde a derivada não seja nula ou muito pequena, garantindo assim maior probabilidade de convergência rápida e correta do método.

2.5 Método de Newton-Raphson N-dimensional

O método de Newton-Raphson pode ser estendido para resolver sistemas de equações não lineares em múltiplas variáveis. Um sistema de equações não lineares ser representado da seguinte maneira:

$$\begin{cases} f_1(x_1, x_2, \ldots, x_n) = 0 \\ f_2(x_1, x_2, \ldots, x_n) = 0 \\ \qquad\qquad \vdots \qquad\qquad \vdots \\ f_n(x_1, x_2, \ldots, x_n) = 0 \end{cases}$$

onde f_i são as funções não lineares e x_i são as variáveis desconhecidas.

Podemos escrever esse sistema de n variáveis como uma função $F : \mathbb{R}^n \to \mathbb{R}^n$, da seguinte forma:

$$F((x_1, x_2, \ldots, x_n)) = (f_1(x_1, x_2, \ldots, x_n), f_2(x_1, x_2, \ldots, x_n), \ldots, f_n(x_1, x_2, \ldots, x_n))^t$$

Portanto, de forma mais compacta, o sistema pode ser escrito na forma vetorial como $F(X) = 0$, onde F é uma função vetorial e X é um vetor de variáveis.

2.5.1 Definições

Durante a construção do método de Newton-Raphson unidimensional (2.4), definimos a função de iteração

$$\varphi(x) = x - A(x)f(x).$$

com isso achamos que a função $A(x)$ que satisfazia $\varphi'(\xi) = 0$ é $A(x) = \dfrac{-1}{f'(x)}$ desde que $f'(x) \neq 0$. Similarmente, para o caso n-dimensional, generalizamos a função de iteração como

$$G(\mathbf{x}) = x - A(x)^{-1}F(x)$$

em que $A(x)$ é a matriz não singular que satisfaz $G'(\xi) = 0$. A matriz $A(x)$ pode ser representada como

$$A(x) = \begin{bmatrix} a_{11}(x) & a_{12}(x) & \ldots & a_{1n}(x) \\ a_{21}(x) & a_{22}(x) & \ldots & a_{2n}(x) \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1}(x) & a_{n2}(x) & \ldots & a_{nn}(x) \end{bmatrix} \tag{2.14}$$

onde cada entrada $a_{ij}(x)$ é uma função de $\mathbb{R}^n \to \mathbb{R}$. Seja b_{ij} as entradas da matriz inversa $A^{-1}(x)$. As funções coordenadas $g_i(\mathbf{x})$ são da forma

$$g_i(\mathbf{x}) = x_i - \sum_{j=1}^{n} b_{ij}(x)f_j(x) \tag{2.15}$$

e

$$\frac{\partial g_i(\mathbf{x})}{\partial x_k} = \delta_{ik} - \sum_{j=1}^{n} \left(\frac{\partial b_{ij}(x)}{\partial x_k} f_j(x) + b_{ij}(x) \frac{\partial f_j(x)}{\partial x_k} \right) \tag{2.16}$$

onde δ_{ik} é o delta de Kronecker, que é 1 se $i = k$ e 0 caso contrário.

2.5.2 Calculo do Jacobiano

Método de Newton-Raphson N-dimensional.

Proposição 2.5.1. *Seja* $\mathbf{p}$ *um ponto fixo de* $\mathbb{G}$. *Se existe um* $\delta > 0$ *tal que*

i) $\dfrac{\partial g_i}{\partial x_j}$ *é contínua em uma vizinhança* $N_\delta = \{x \in \mathbb{R}^n : ||x - p|| < \delta\}$ *para todo* $i, j = 1, 2, \ldots, n$

ii) $\dfrac{\partial^2 \mathbb{G}_i}{\partial x_j \partial x_k}$ *é contínua e* $|\dfrac{\partial^2 g_i(x)}{\partial x_j \partial x_k}| \leq M$

iii) $\dfrac{\partial g_i(p)}{\partial x_k} = 0$, *para todo* $i = 1, 2, \ldots, n$

Então existe $\bar{\delta} \leq \delta$ *de tal forma que a sequência gerada por* $x_{k+1} = \mathbb{G}(x_k)$ *converge para* $\mathbf{p}$ *de forma quadrática, para qualquer escolha de* x_0 *que satisfaz* $||x_0 - \mathbf{p}|| \leq \bar{\delta}$

@EnzoR: Tarefa aos Matematicos: Encontrar onde isso ta demonstrado

A demonstração é análoga à do Teorema 2.4.1. E está descrita extensivamente em ref.

A partir de (2.16) vamos caracterizar a matriz A. Obseve que, de $\dfrac{\partial g_i(\mathbf{p})}{\partial x_k} = 0$ para todo $i = 1, 2, \ldots, n$, temos que

$$0 = 1 - \sum_{j=1}^{n} \left(b_{ij}(\mathbf{p}) \frac{\partial f_j(\mathbf{p})}{\partial x_i} \right) \quad \text{para i = j} \tag{2.17}$$

e

$$0 = - \sum_{j=1}^{n} \left(b_{ij}(\mathbf{p}) \frac{\partial f_j(\mathbf{p})}{\partial x_k} \right) \quad \text{para i} \neq \text{j.} \tag{2.18}$$

A matriz Jacobiana J de F é a matriz das derivadas dada a seguir:

$$J(\mathbf{p}) = \begin{bmatrix} \dfrac{\partial f_1(\mathbf{p})}{\partial x_1} & \dfrac{\partial f_1(\mathbf{p})}{\partial x_2} & \cdots & \dfrac{\partial f_1(\mathbf{p})}{\partial x_n} \\ \dfrac{\partial f_2(\mathbf{p})}{\partial x_1} & \dfrac{\partial f_2(\mathbf{p})}{\partial x_2} & \cdots & \dfrac{\partial f_2(\mathbf{p})}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \dfrac{\partial f_n(\mathbf{p})}{\partial x_1} & \dfrac{\partial f_n(\mathbf{p})}{\partial x_2} & \cdots & \dfrac{\partial f_n(\mathbf{p})}{\partial x_n} \end{bmatrix} \tag{2.19}$$

As condições (2.17) e (2.18) implicam que

$$A(\mathbf{p})^{-1}J(\mathbf{p}) = I$$

$$A(\mathbf{p})A(\mathbf{p})^{-1}J(\mathbf{p}) = A(\mathbf{p})I$$

$$IJ(\mathbf{p}) = A(\mathbf{p})$$

$$J(\mathbf{p}) = A(\mathbf{p})$$

Portanto, uma escolha natural para $A(x)$ é a matriz Jacobiana $J(x)$ de F. Assim, a função de iteração do método de Newton-Raphson n-dimensional é dada por

$$G(\mathbf{x}) = x - J(x)^{-1}F(x). \tag{2.20}$$

A sequência iterativa do MNR n-dimensional é

$$x_{k+1} = x_k - J(x_k)^{-1}F(x_k) \tag{2.21}$$

onde $\mathbf{x}_k$ é a aproximação atual da solução, $J(x_k)$ é a matriz Jacobiana avaliada em x_k, e $F(\mathbf{x}_k)$ é o vetor de funções avaliado em x_k. É esperado que a sequência $\{x_k\}$ convirja, **quadraticamente**, para a solução do sistema de equações não lineares, desde que o chute inicial x_0 esteja suficientemente próximo da solução e que a matriz Jacobiana avaliada em $J(x_k)$ seja invertível nesse ponto. Vejamos um exemplo para n = 2. Considere o sistema de equações não lineares:

$$\begin{cases} x^2 + y^2 - 4 = 0 \\ x^2 - y - 1 = 0 \end{cases}$$

Podemos definir a função $F : \mathbb{R}^2 \to \mathbb{R}^2$ como

$$F((x, y)) = \begin{bmatrix} x^2 + y^2 - 4 \\ x^2 - y - 1 \end{bmatrix}$$

A matriz Jacobiana $J(x, y)$ de F é dada por

$$J((x,y)) = \begin{bmatrix} \dfrac{\partial(x^2+y^2-4)}{\partial x} & \dfrac{\partial(x^2+y^2-4)}{\partial y} \\ \dfrac{\partial(x^2-y-1)}{\partial x} & \dfrac{\partial(x^2-y-1)}{\partial y} \end{bmatrix}$$

$$J((x,y)) = \begin{bmatrix} 2x & 2y \\ 2x & -1 \end{bmatrix}$$

A iteração do método de Newton-Raphson n-dimensional é então dada por

$$\begin{aligned} \begin{bmatrix} x_{k+1} \\ y_{k+1} \end{bmatrix} &= \begin{bmatrix} x_k \\ y_k \end{bmatrix} - J((x_k, y_k))^{-1} F((x_k, y_k)) \\ &= \begin{bmatrix} x_k \\ y_k \end{bmatrix} - \begin{bmatrix} 2x_k & 2y_k \\ 2x_k & -1 \end{bmatrix}^{-1} \begin{bmatrix} x_k^2 + y_k^2 - 4 \\ x_k^2 - y_k - 1 \end{bmatrix} \\ &= \begin{bmatrix} x_k \\ y_k \end{bmatrix} - \frac{1}{-2x_k - 4x_k y_k} \begin{bmatrix} -1 & -2y_k \\ -2x_k & 2x_k \end{bmatrix} \begin{bmatrix} x_k^2 + y_k - 4 \\ x_k^2 - y_k - 1 \end{bmatrix} \end{aligned}$$

Vamos escolher um chute inicial $(x_0, y_0) = (2, 1)$ e aplicar o método iterativamente:

Iteração	x_k	y_k
0	2.0000000000000000	1.0000000000000000
1	1.5000000000000000	0.2500000000000000
2	1.3095238095238095	0.7142857142857143
3	1.2727272727272727	0.6181818181818182
4	1.2679491924311228	0.6180339887498950
5	1.2679491924311228	0.6180339887498950

Após algumas iterações, o método converge para a solução aproximada $(x, y) \approx (1.2679491924311228, 0.618033988749895)$, que satisfaz o sistema de equações não lineares.

@EnzoR: @LucasM, por favor, verifique se era assim que você desejava.

Do ponto de vista computacional, calcular explicitamente a inversa da matriz Jacobiana pode ser ineficiente e numericamente instável, especialmente para sistemas de

grande porte. Por isso, a sequência iterativa (2.21) é normalmente reescrita de forma mais eficiente, evitando a inversão direta da matriz. Introduz-se uma variável auxiliar γ que representa a solução do sistema linear:

$$J(x_k)\,\gamma = -F(x_k)$$

Assim, a atualização da aproximação é feita por

$$x_{k+1} = x_k + \gamma$$

Dessa forma, em cada iteração, resolve-se um sistema linear envolvendo a matriz Jacobiana, o que é computacionalmente mais eficiente e estável do que calcular sua inversa.

$$\begin{aligned} \gamma &= -J(x_k)^{-1}F(x_k) \\ J(x_k)\gamma &= J(x_k)J(x_k)^{-1}F(x_k) \\ J(x_k)\gamma &= -F(x_k) \end{aligned}$$

Subistituindo γ em (2.21) temos

$$x_{k+1} = x_k + \gamma$$

2.5.3 Fractais de Newton

A aplicação do método de Newton para os pontos do ***Plano Complexo*** gera os **Fractais de Newton**.

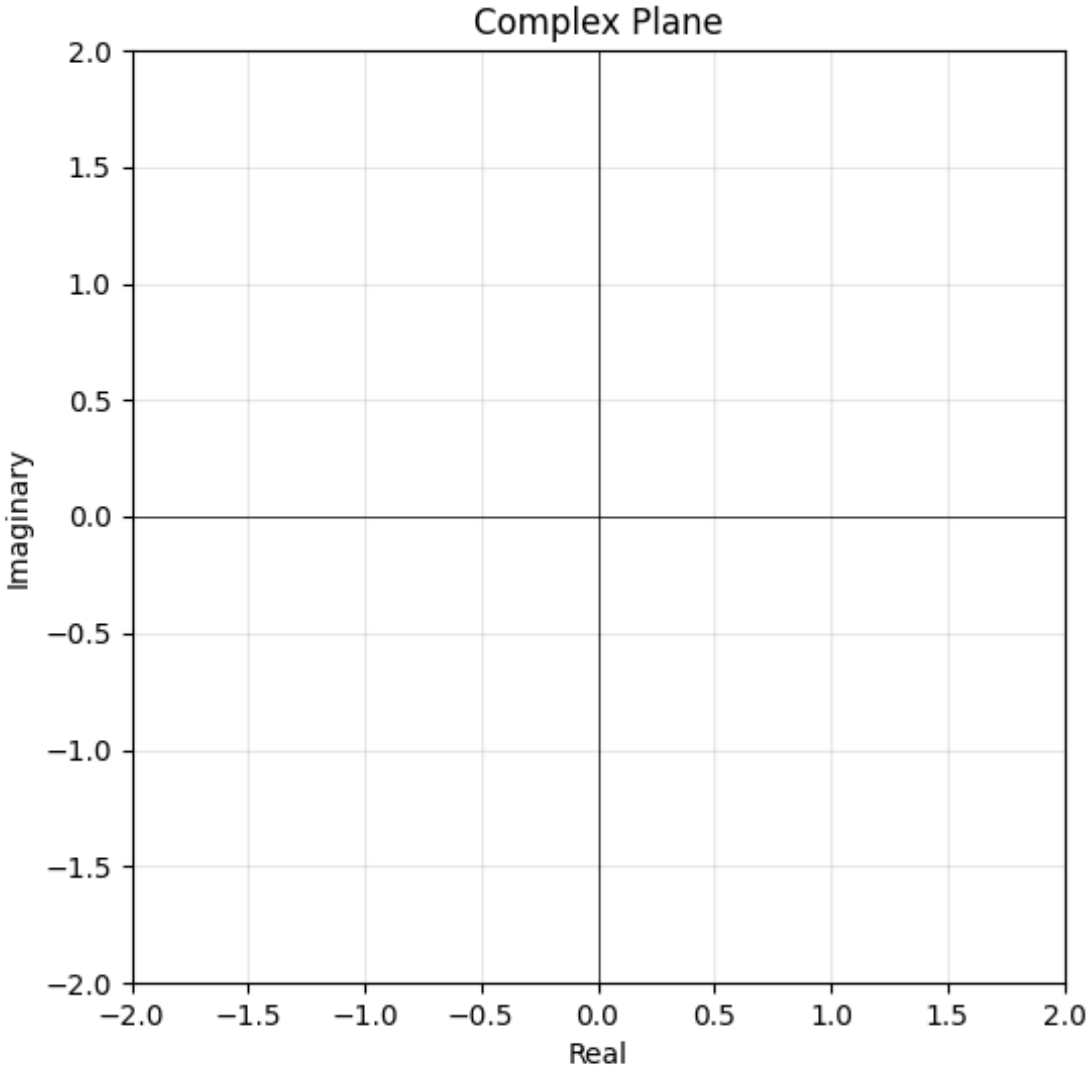
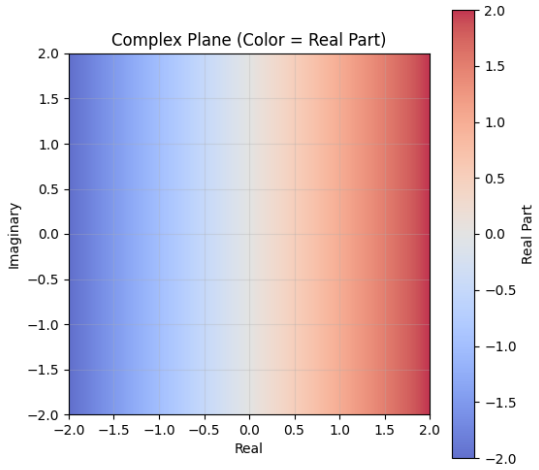
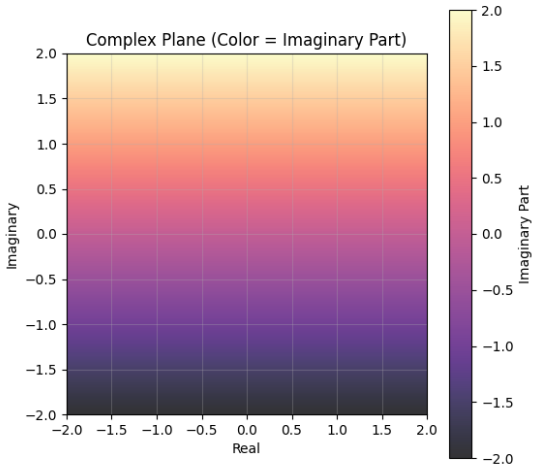


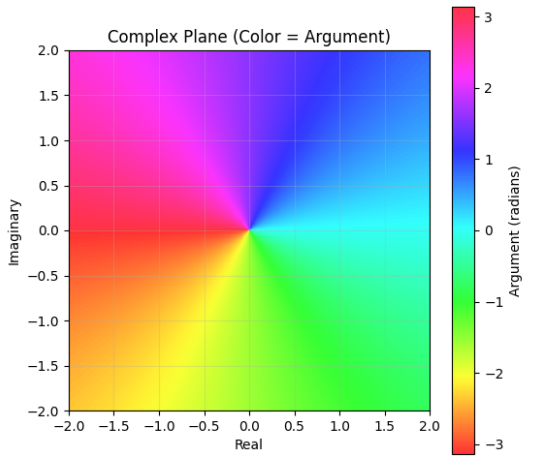
Figura 2.8: Plano Complexo



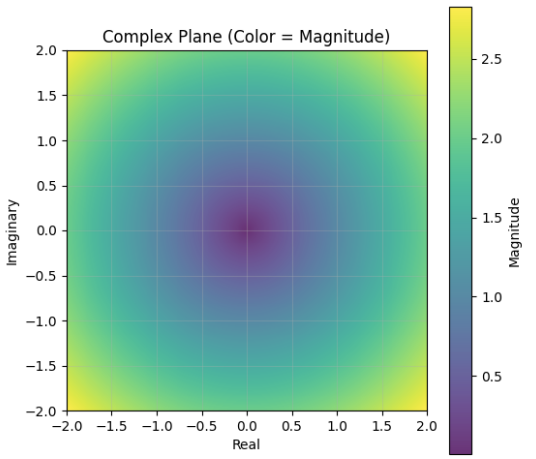
Parte Real



Parte Imaginária



Parte Angular



Parte Magnitude

Figura 2.9: Propriedades dos Números Complexos

O método de Newton-Raphson no plano complexo é aplicado a funções complexas $f : \mathbb{C} \to \mathbb{C}$. A fórmula de iteração é semelhante à do caso real, mas agora z é um número complexo:

$$z_{n+1} = z_n - \frac{f(z_n)}{f'(z_n)}$$

onde z_n é a aproximação atual, $f(z_n)$ é o valor da função no ponto z_n, e $f'(z_n)$ é a derivada da função no ponto z_n.

asdklsahdoisahop A ideia básica para gerar um fractal de Newton é aplicar o método de Newton-Raphson a cada ponto em uma região do plano complexo e colorir o ponto com base na raiz para a qual ele converge e o número de iterações que levou para convergir. Se um ponto não convergir para nenhuma raiz dentro da tolerância estabelecida, ele é colorido de uma forma diferente. Nos gráficos utilizados nessa seção, quanto mais escura uma cor, mais tempo levou para convergir, e preto indica que não convergiu.

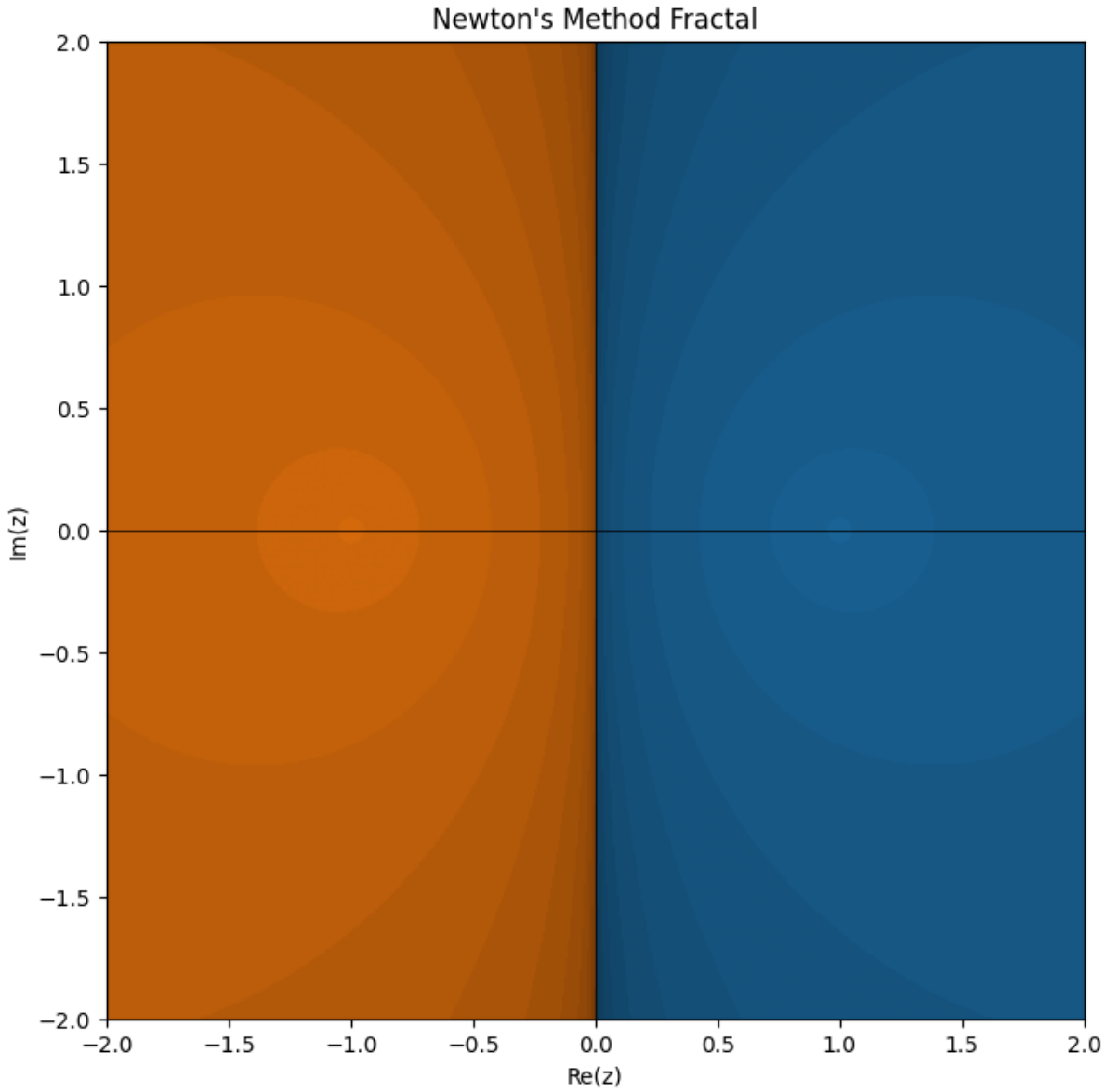


Figura 2.10: Fractais de Newton – 2 Unidades da raiz.

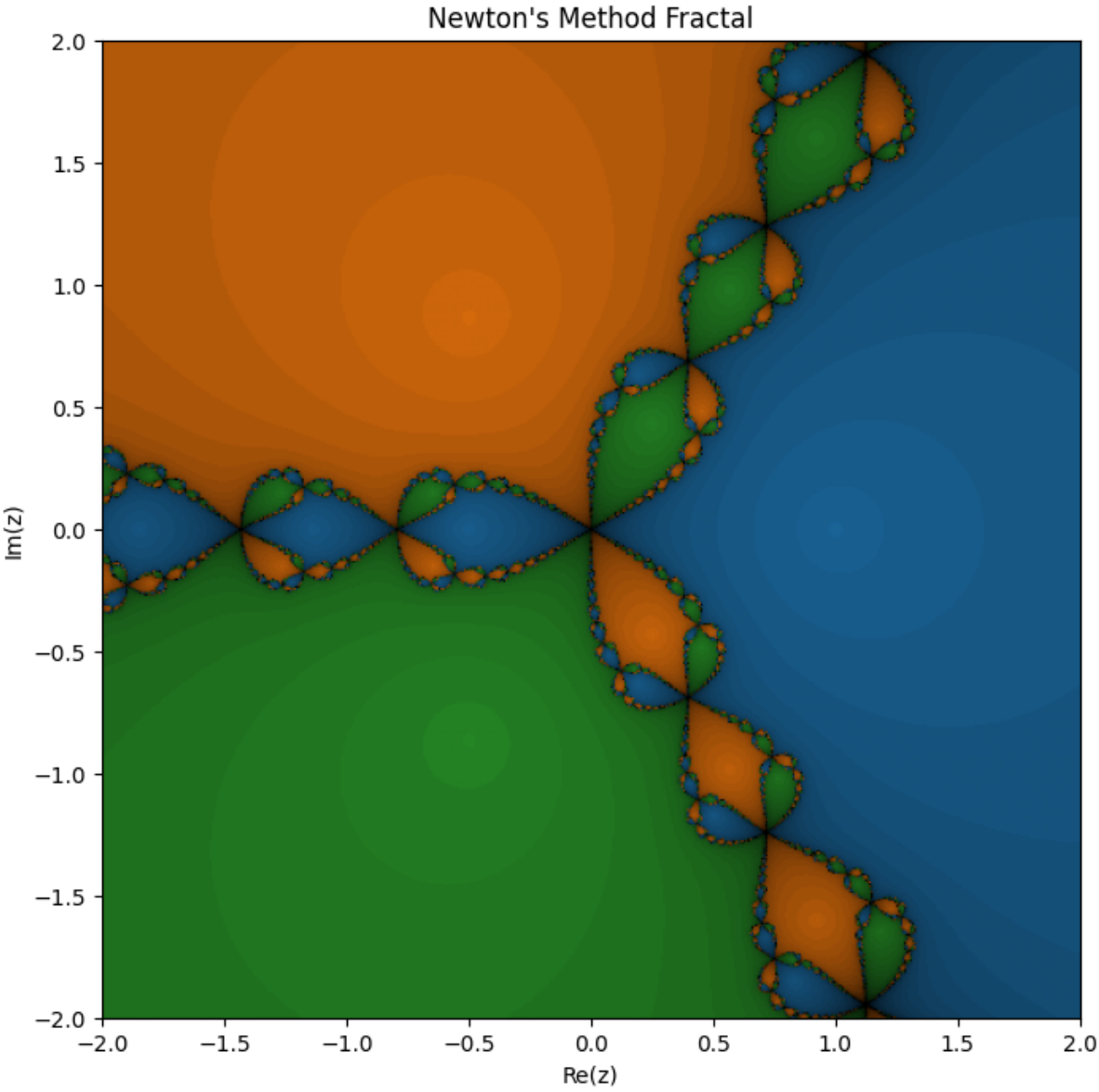


Figura 2.11: Fractais de Newton - 3 Unidades da raiz.

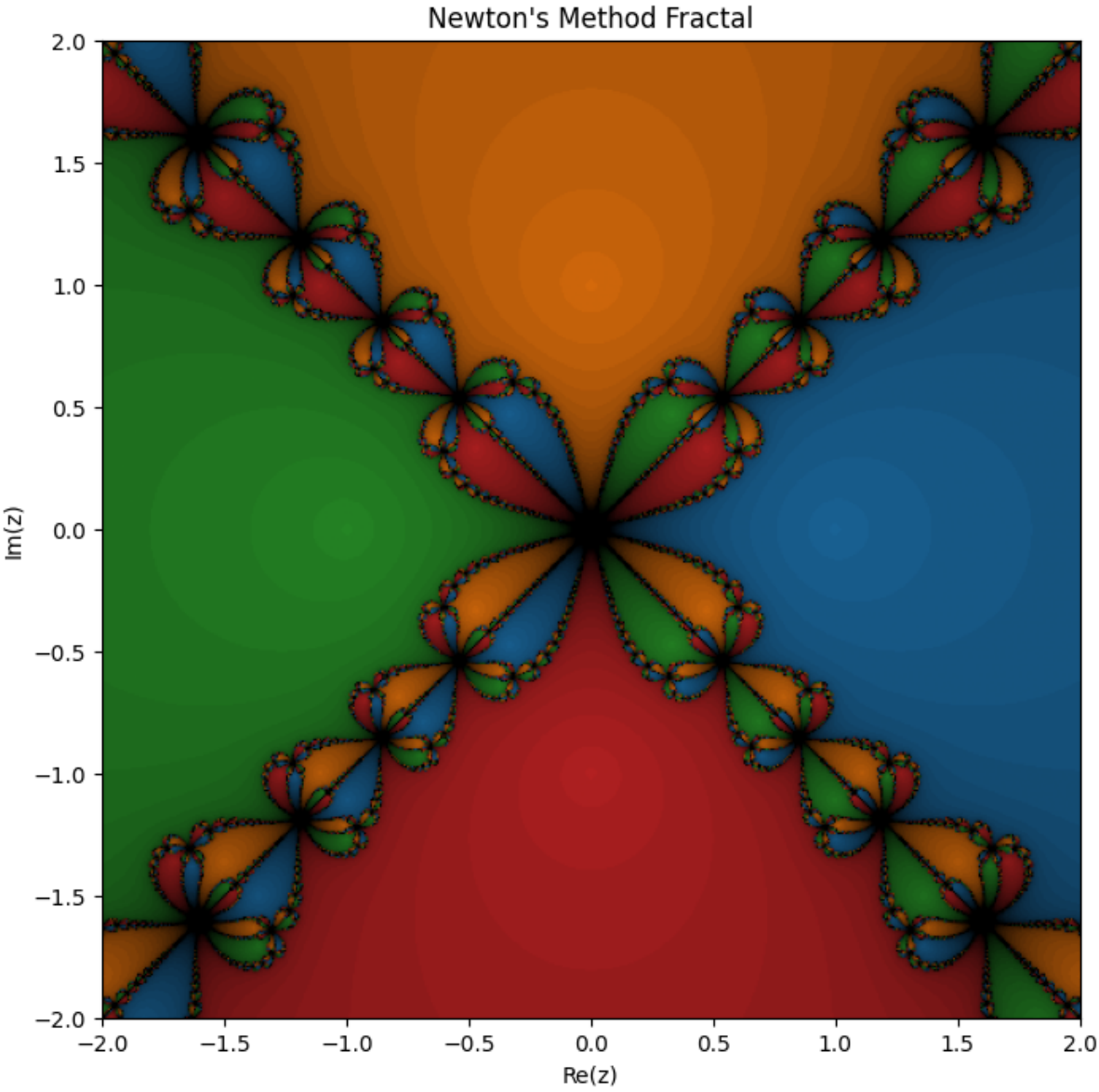


Figura 2.12: Fractais de Newton - 4 Unidades da raiz.

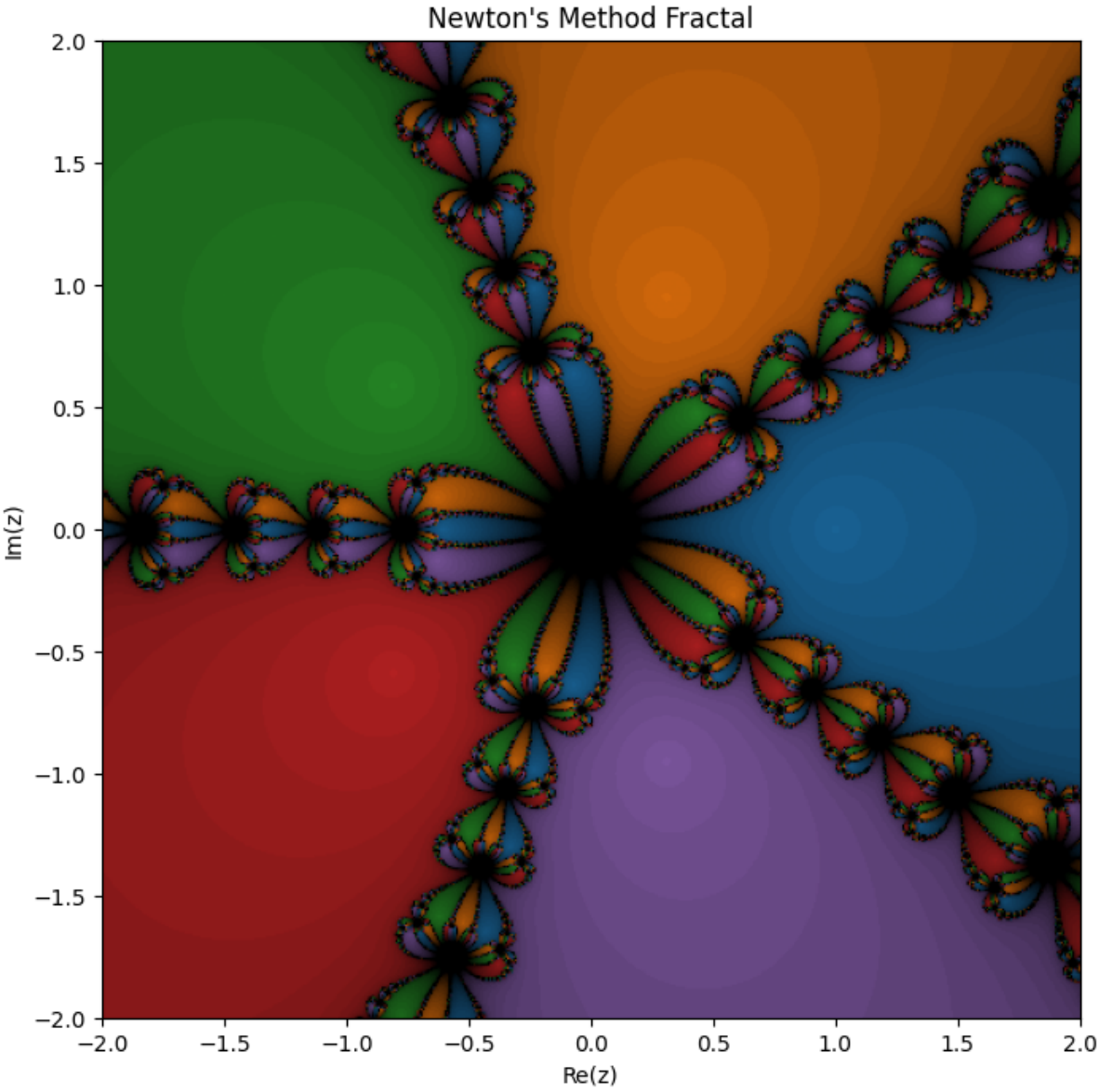


Figura 2.13: Fractais de Newton - 5 Unidades da raiz.

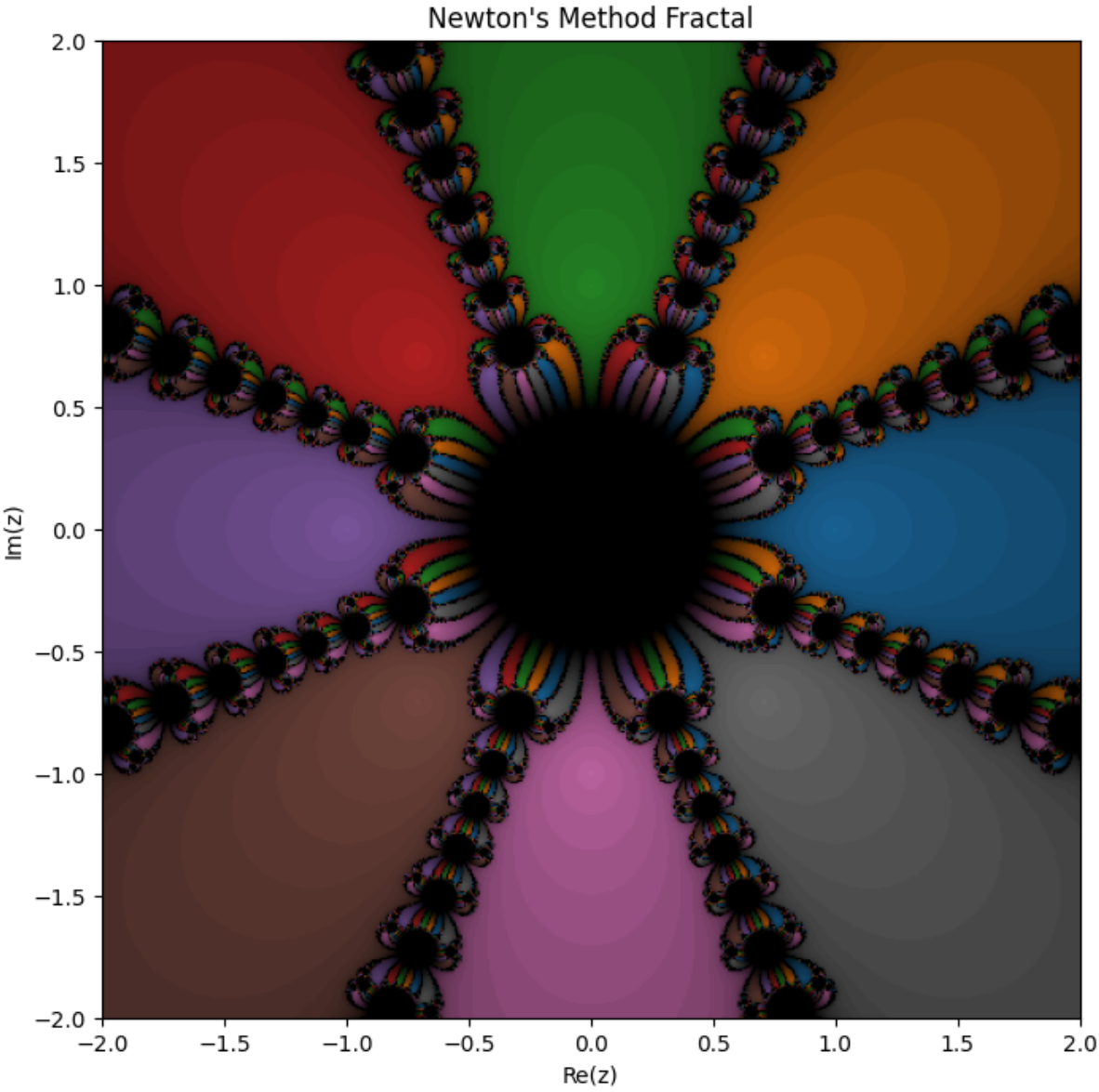


Figura 2.14: Fractais de Newton - 8 Unidades da raiz.

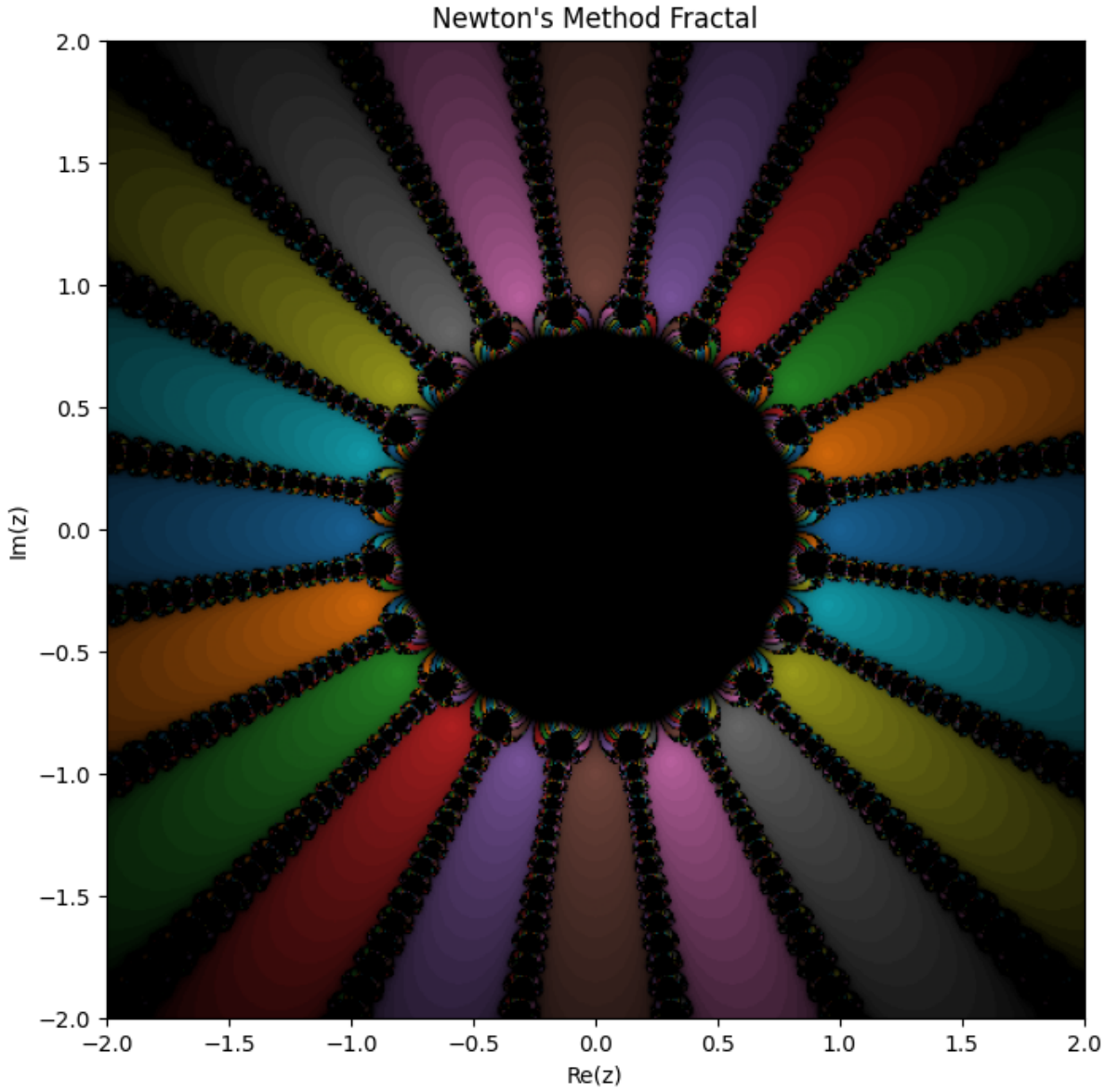


Figura 2.15: Fractais de Newton - 20 Unidades da raiz.

TODO: COLOCAR PolinomioS

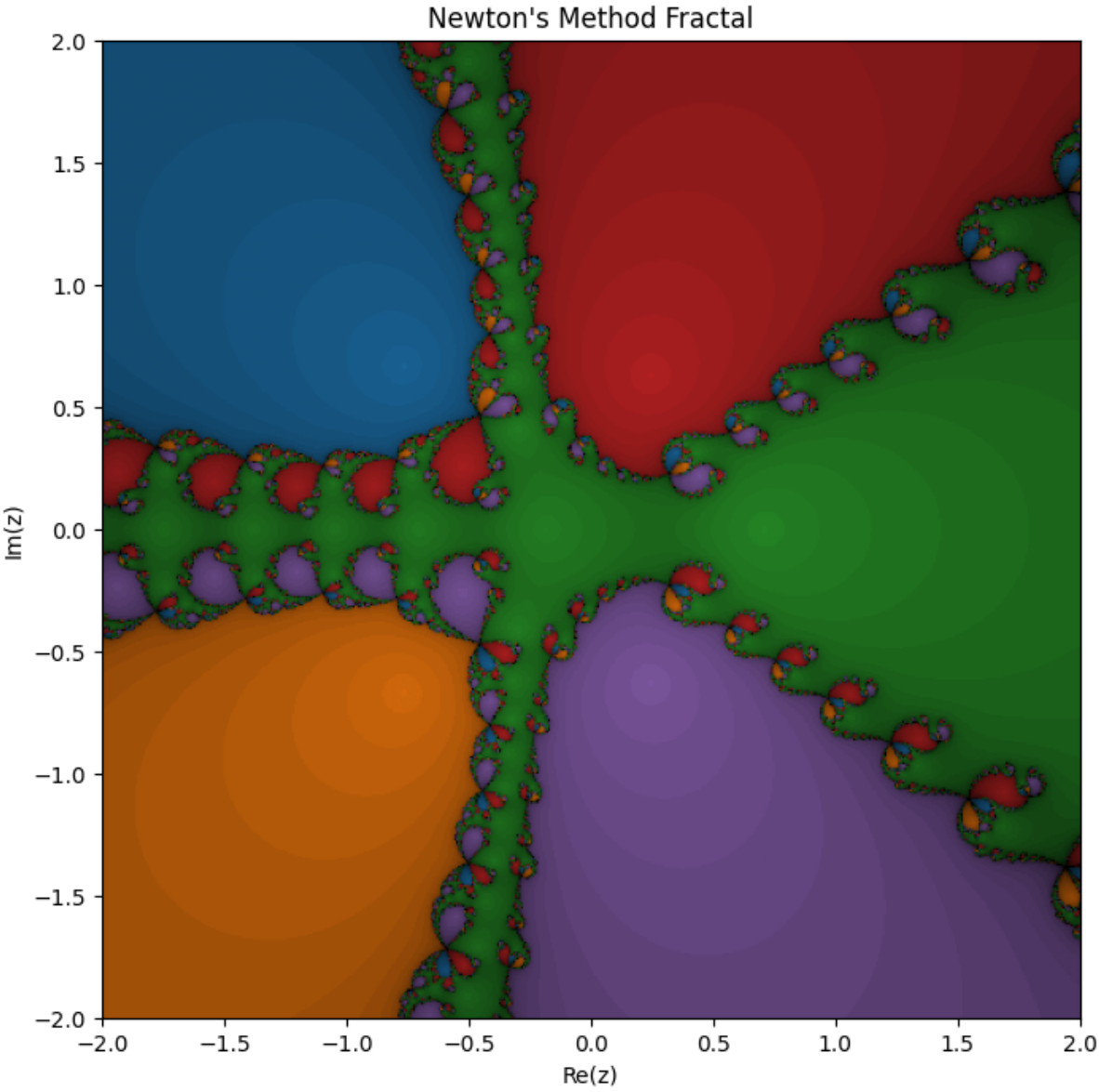


Figura 2.16: Fractais de Newton - Polinomio ABCD.

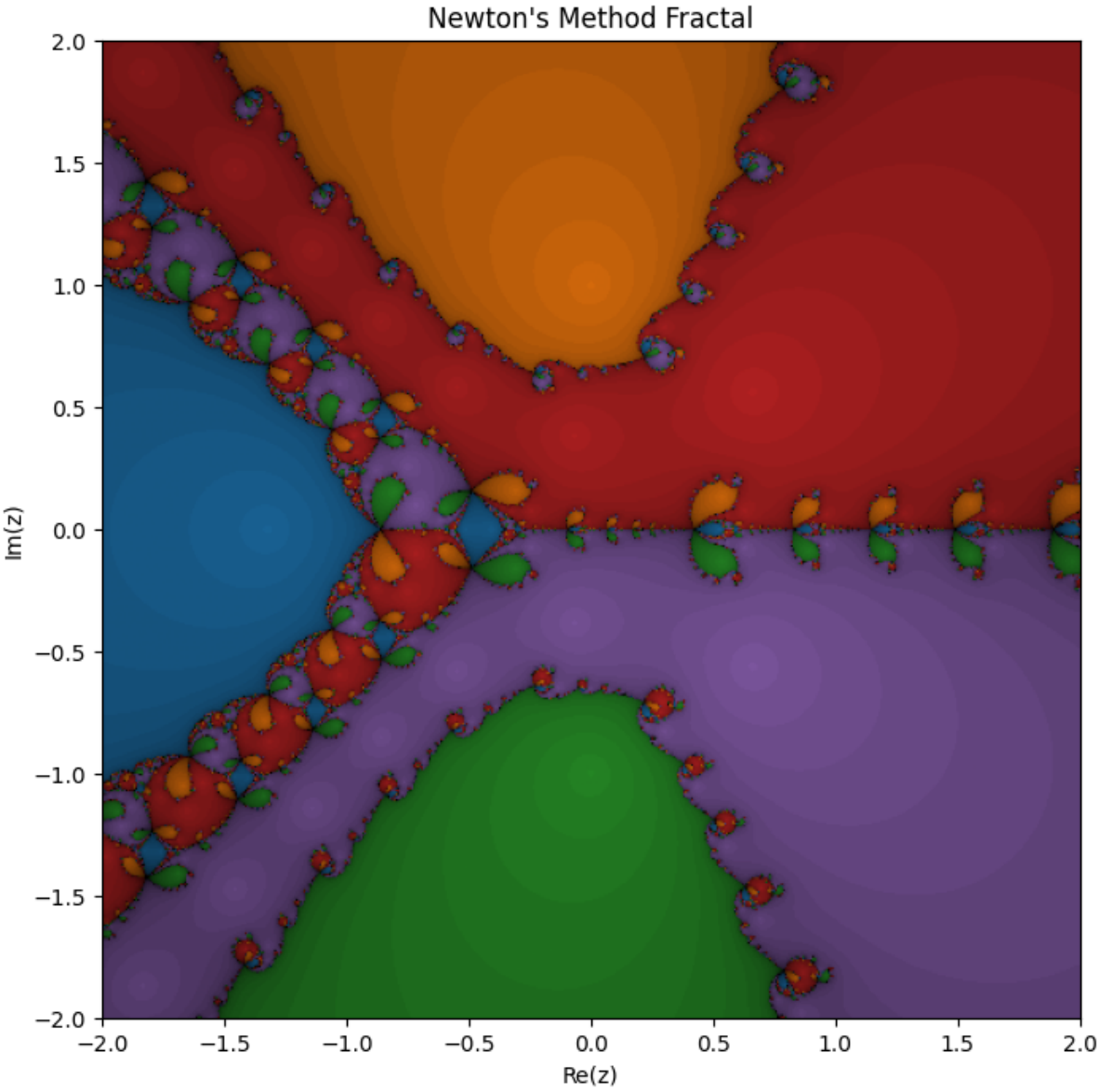


Figura 2.17: Fractais de Newton - Polinomio ABCD.

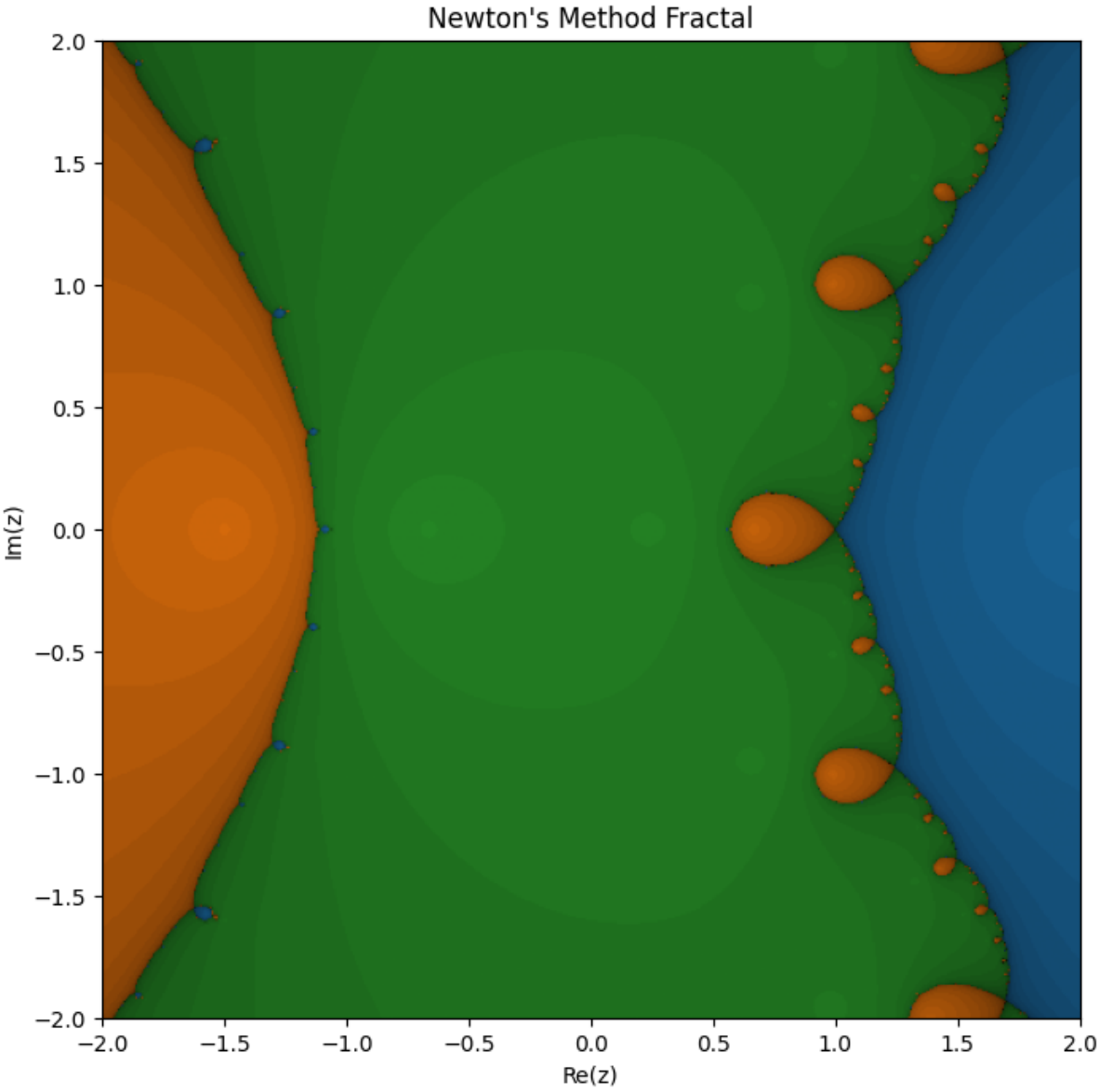


Figura 2.18: Fractais de Newton - Polinomio ABCD.

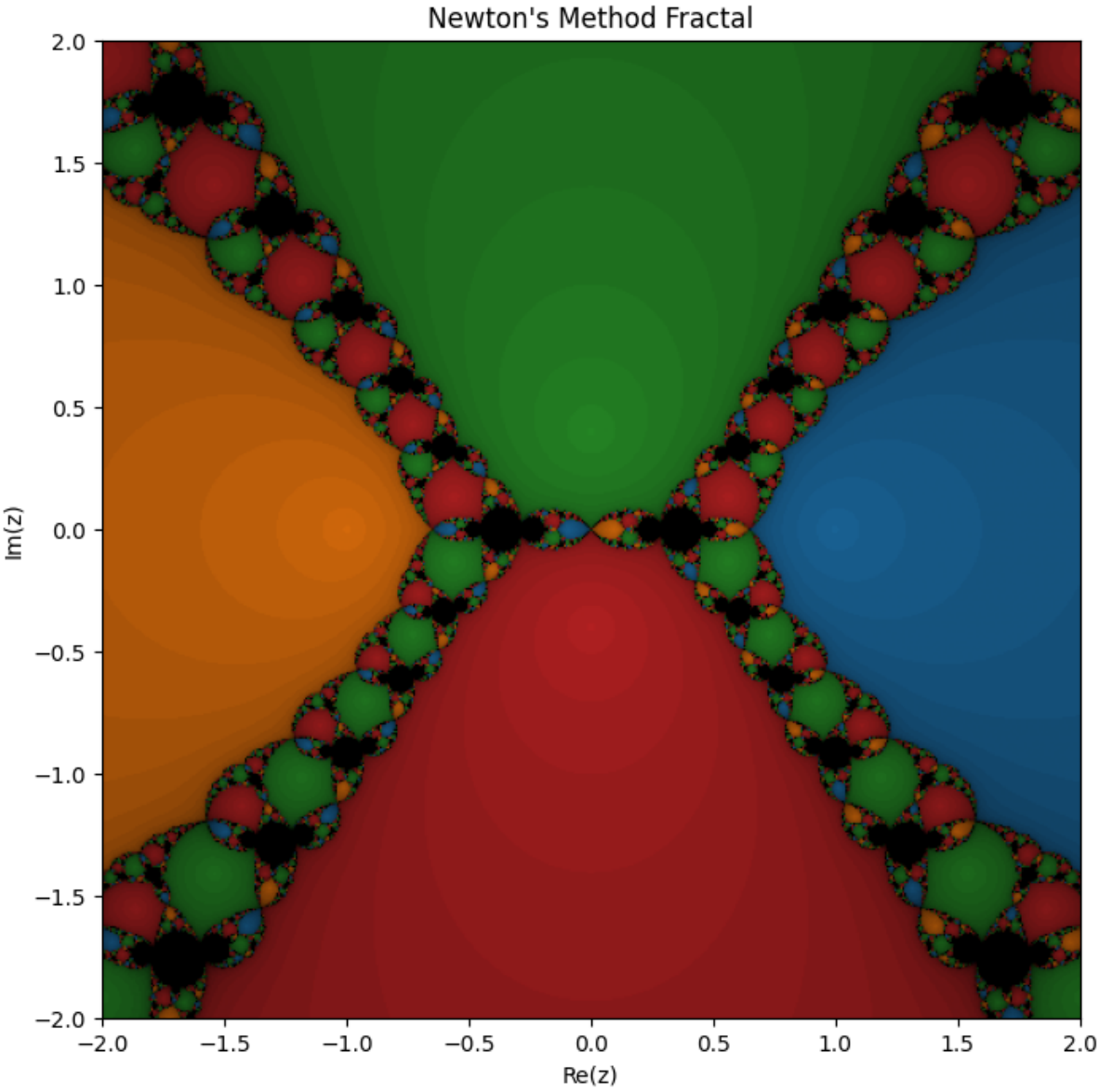


Figura 2.19: Fractais de Newton - Polinomio ABCD.

Capítulo 3

Problemas de Valor Inicial para Equações Diferenciais Ordinárias

3.1 Equações Diferenciais Ordinárias

Muitos dos princípios por traz das leis da natureza são relações que estão diretamente ligados com a taxa de variação de certas quantidades. Matematicamente, essas relações são equações e as taxas de variação são representadas por derivadas. Equações que envolvem derivadas são chamadas de equações diferenciais, que em alguns casos são chamadas de Modelos Matemáticos. Equações diferenciais podem ser classificadas em dois tipos principais: equações diferenciais ordinárias (EDOs) e equações diferenciais parciais (EDPs). Uma equação diferencial ordinária (EDO) é uma equação que envolve uma função desconhecida e suas derivadas em relação a uma única variável independente. As EDOs são amplamente utilizadas para modelar fenômenos em diversas áreas do conhecimento, como física, engenharia, biologia e economia. As EDOs podem ser classificadas de várias maneiras. Uma equação diferencial ordinária de primeira ordem é uma equação do tipo:

$$y' = F(t, y) \tag{3.1}$$

onde t é a variável independente, y é a função desconhecida, e y' é a derivada de y em relação a t.

Exemplo:

$$\frac{dy}{dt} = sen(t) \tag{3.2}$$

A solução geral dessa EDO é dada por:

$$y(t) = -cos(t) + C, \quad C \in \mathbb{R} \tag{3.3}$$

3.2 Definição de Problema de Valor Inicial

Um problema de valor inicial (PVI) para uma equação diferencial ordinária consiste em encontrar uma função desconhecida $y(t)$ que satisfaça uma equação diferencial e que atenda a uma condição inicial especificada em um ponto t_0. Para a análise de existência e unicidade de soluções consideramos as condições de Lipschitz e continuidade. Em termos gerais, um PVI pode ser formulado como:

$$\begin{cases} y'(t) = f(t, y(t)), \quad t \in [a, b] \\ y(t_0) = y_0 \end{cases} \tag{3.4}$$

onde f é uma função dada, $y'(t)$ é a derivada de $y(t)$ em relação a t, e y_0 é o valor inicial da função no ponto t_0.

A solução de um PVI é uma função $y(t)$ que satisfaz tanto a equação diferencial quanto a condição inicial. A existência e unicidade de soluções para PVIs são garantidas sob certas condições, como as condições de Lipschitz e continuidade. No geral, uma solução de uma EDO é uma família de funções, mas ao especificar uma condição inicial, obtemos uma solução única que passa pelo ponto inicial dado. Exemplo:

$$\frac{dy}{dt} = sen(t) \tag{3.5}$$

Com a condição inicial:

$$y(0) = 0 \tag{3.6}$$

A solução particular desse PVI é dada por:

$$y(t) = -cos(t) + 1 \tag{3.7}$$

3.3 Métodos Numéricos para Resolver PVIs

Existem vários métodos numéricos para resolver problemas de valor inicial para equações diferenciais ordinárias. Neste capítulo, discutiremos alguns dos métodos mais comuns, incluindo o método de Euler, o método de Runge-Kutta.

3.3.1 Teorema de Existência e Unicidade

O teorema de existência e unicidade para PVIs estabelece condições sob as quais uma solução única existe para um problema de valor inicial. Essas condições geralmente envolvem a continuidade da função $f(t, y)$ e a satisfação da condição de Lipschitz em relação à variável y.

Teorema 3.3.1 (PVI). *Suponha que a função $f(t, y)$ seja contínua em um retângulo $R = \{(t, y)|a \leq t \leq b, c \leq y \leq d\}$ e satisfaça a condição de Lipschitz em y na mesma região. Então, para qualquer ponto inicial (t_0, y_0) dentro de R, existe um intervalo $[t_0 - h, t_0 + h]$ em que o PVI*

$$\begin{cases} y'(t) = f(t, y(t)), & t \in [a, b] \\ y(t_0) = y_0 \end{cases} \tag{3.8}$$

tem uma única solução em D.

3.3.2 Boa Colocação do PVI

O problema

$$\begin{cases} y'(t) = f(t, y(t)), & t \in [a, b] \\ y(t_0) = y_0 \end{cases} \tag{3.9}$$

é dito estar bem colocado se existe uma solução única que depende continuamente dos dados do problema, ou seja, da função f e da condição inicial y_0. E para todo $\epsilon > 0$, existe um $\delta > 0$ tal que, se $||\tilde{f} - f|| < \delta$ e $|\tilde{y_0} - y_0| < \delta$, então a solução $\tilde{y}(t)$ do PVI perturbado satisfaz $||\tilde{y}(t) - y(t)|| < \epsilon$ para todo $t \in [a, b]$, uma solução única z(t) para o problema

$$\begin{cases} z'(t) = \tilde{f}(t, z(t)) + \delta(t), & t \in [a, b] \\ z(t_0) = \tilde{y_0} \end{cases} \tag{3.10}$$

A boa colocação é uma propriedade desejável para garantir que pequenas variações nos dados do problema não resultem em grandes mudanças na solução.

3.3.3 Método de Euler

O método de Euler é um dos métodos numéricos mais simples para resolver PVIs. Ele é baseado na ideia de aproximar a solução da EDO usando uma série de passos pequenos.

Capítulo 4

Comparação entre Paradigmas de Programação

Os métodos numéricos podem ser implementados utizando diferentes paradigmas de programação, como o imperativo e o funcional. No paradigma imperativo, o foco está na sequência de comandos que modificam o estado do programa, enquanto no paradigma funcional, a ênfase está na aplicação de funções e na imutabilidade dos dados. Ao implementar métodos numéricos, o paradigma imperativo pode ser mais intuitivo para aqueles familiarizados com a manipulação direta de variáveis e estruturas de controle, como loops e condicionais. Por outro lado, o paradigma funcional pode oferecer vantagens em termos de clareza e concisão, especialmente ao lidar com operações matemáticas complexas e recursivas. No entanto, a escolha do paradigma pode influenciar o desempenho e a legibilidade do código, dependendo do contexto e dos requisitos específicos do problema numérico em questão. Portanto, é importante considerar as características de cada paradigma ao implementar métodos numéricos para garantir eficiência e manutenção do código.

Neste capítulo, apresentamos o pseudocódigo dos métodos abordados anteriormente, em ambos os paradigmas de programação: imperativo e funcional. A seguir, discutimos as diferenças e semelhanças entre as implementações, destacando as vantagens e desvantagens de cada abordagem. A implementação real em C, Python e Haskell pode ser encontrada no repositório Github do projeto.

4.1 Comparação Entre Paradigmas

A comparação entre os paradigmas de programação imperativo e funcional na implementação de métodos numéricos revela diferenças significativas em termos de estrutura, legibilidade e manutenção do código. No paradigma imperativo, o foco está na sequência de comandos que modificam o estado do programa, o que pode tornar o código mais intuitivo para aqueles familiarizados com a manipulação direta de variáveis e estruturas de controle, como loops e condicionais. Por exemplo, no método do ponto fixo iterativo, a utilização de um loop explícito facilita a compreensão do fluxo de iterações. Por outro lado, o paradigma funcional enfatiza a aplicação de funções e a imutabilidade dos dados, o que pode resultar em um código mais conciso e expressivo. No método do ponto fixo recursivo, a ausência de estados mutáveis e a utilização de funções auxiliares tornam o código mais fácil de entender em termos matemáticos. No entanto, a recursão pode levar a problemas de desempenho e consumo de memória em casos de iterações profundas. Em termos de manutenção, o código funcional pode ser mais fácil de modificar, pois as funções são independentes e não dependem de estados compartilhados. Em resumo, a escolha entre os paradigmas deve considerar o contexto do problema, a familiaridade do programador com cada abordagem e os requisitos específicos de desempenho e legibilidade do código.

A seguir discutiremos uma análise quantitativa realizada através de métricas como contagem de linhas de código, nível de otimização do compilador, uso de memória, processamento e tempo de execução para avaliar o desempenho das implementações em diferentes cenários. Para garantir a melhor comparação possível, todas as implementações foram otimizadas ao máximo, utilizando as melhores práticas de cada linguagem e paradigma. As medições foram realizadas em um ambiente controlado, utilizando a mesma máquina e condições de teste para minimizar variações externas. Além disso, foram considerados diferentes casos de teste, incluindo funções simples e complexas, para avaliar a robustez e eficiência das implementações em situações variadas.

O uso de linguagens como C para o paradigma imperativo e Haskell para o funcional permitiu explorar as características intrínsecas de cada abordagem. A análise dos resultados revelou que, embora o código imperativo em C tenha apresentado melhor desempenho em termos de tempo de execução e uso de memória para funções simples, o código funcional em Haskell mostrou-se mais eficiente em termos de legibilidade

e manutenção, especialmente para funções matemáticas complexas. Além disso, a capacidade do Haskell de lidar com recursão e funções de ordem superior facilitou a implementação de algoritmos numéricos mais sofisticados. Em conclusão, a escolha do paradigma deve levar em consideração não apenas o desempenho bruto, mas também a clareza do código e a facilidade de manutenção a longo prazo.

Teste realizados no metódo do ponto fixo foram coletados e analisados, gerando os gráficos a seguir:

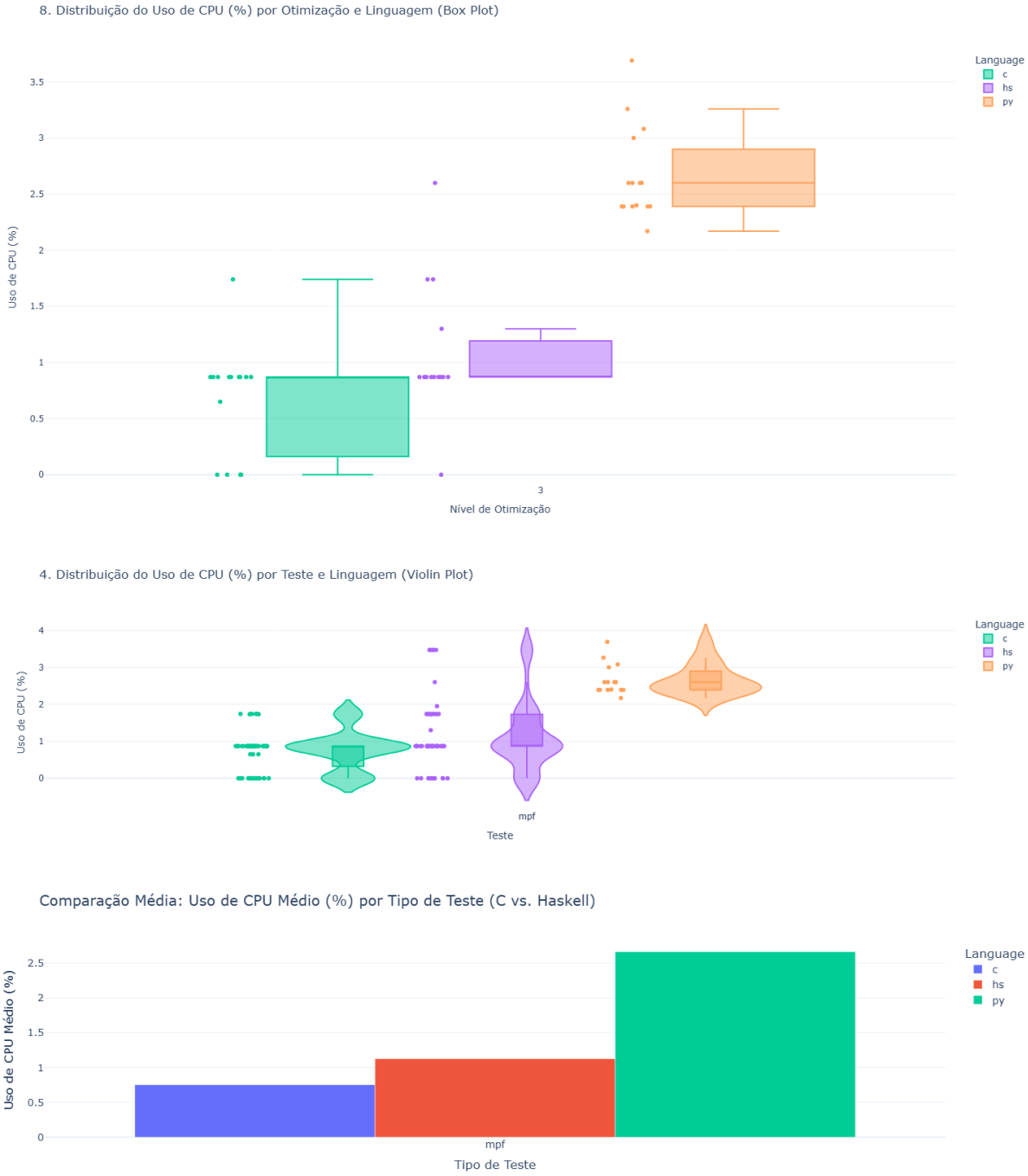


Figura 4.1: Comparação do Uso de CPU entre Implementações Iterativa e Recursiva.

Os gráficos mostram uma melhor performance em relação ao uso de CPU de execução do C, seguido pelo Haskell e Python.

Figura 4.2: Comparação do Uso de Memória entre Implementações Iterativa e Recursiva.

O grafico de Memória Privada mostra o total de memória alocada para o processo, enquanto o gráfico de Memória Física mostra a quantidade de memória realmente utilizada durante a execução. Em relação ao uso de memória, o C também apresentou a melhor performance, seguido pelo Python e Haskell.

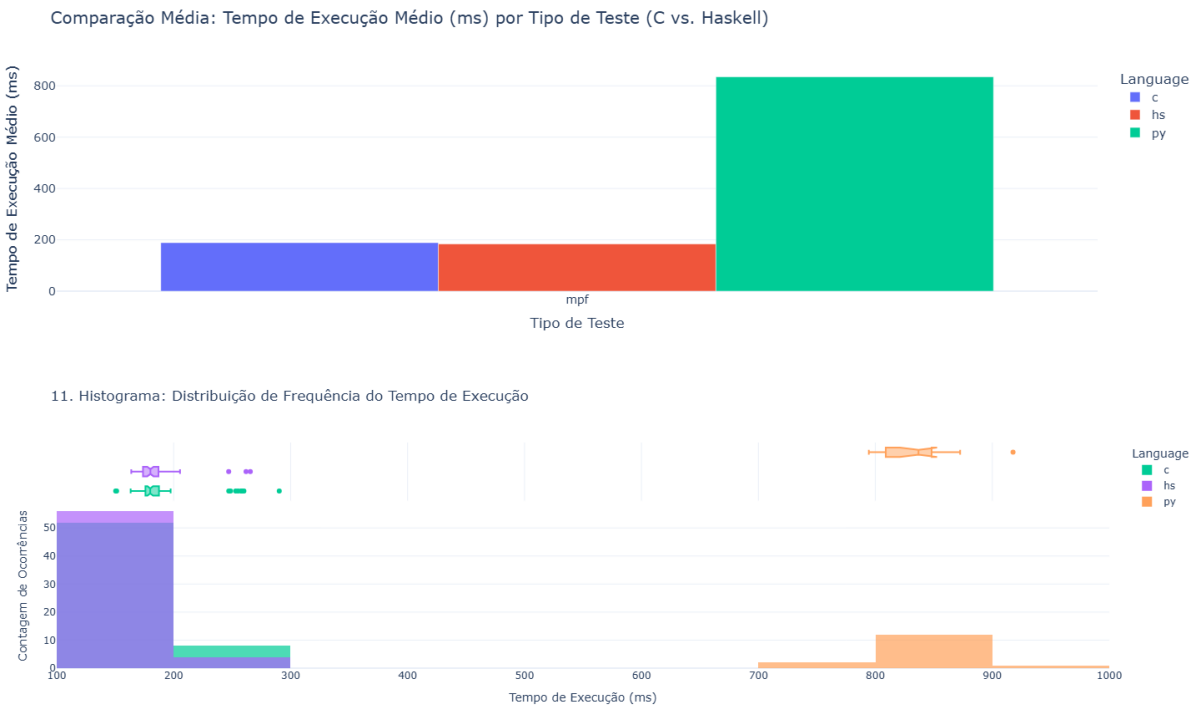


Figura 4.3: Comparação do Tempo de Execução entre Implementações Iterativa e Recursiva.

Em relação ao tempo de execução, o C novamente apresentou a melhor performance que foi muito parecida com a do Haskell. O Python por sua vez apresentou um desempenho muito pior em relação as outras linguagens.

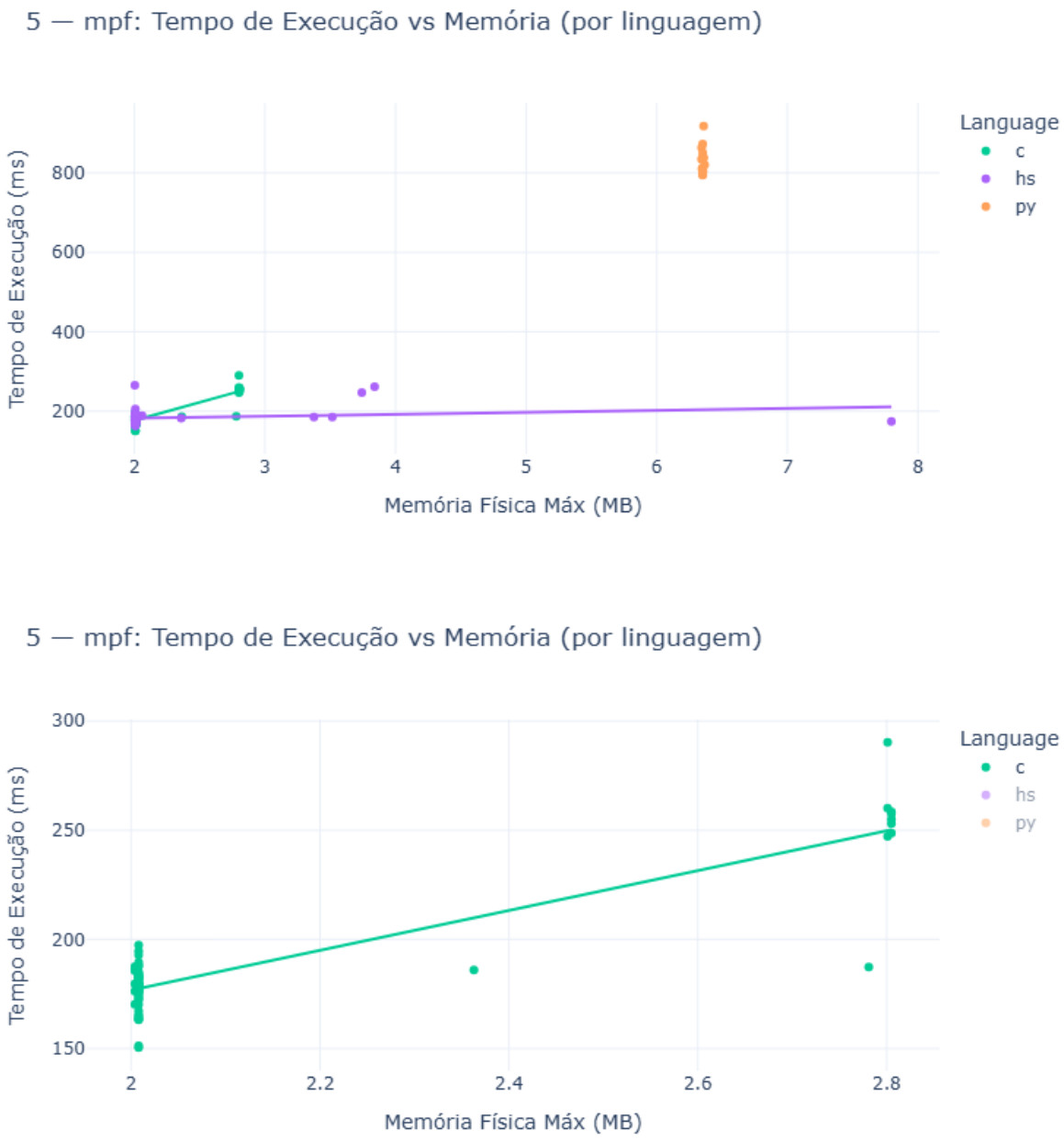


Figura 4.4: Análise de Tendência do Tempo de Execução entre Implementações Iterativa e Recursiva. Parte 1

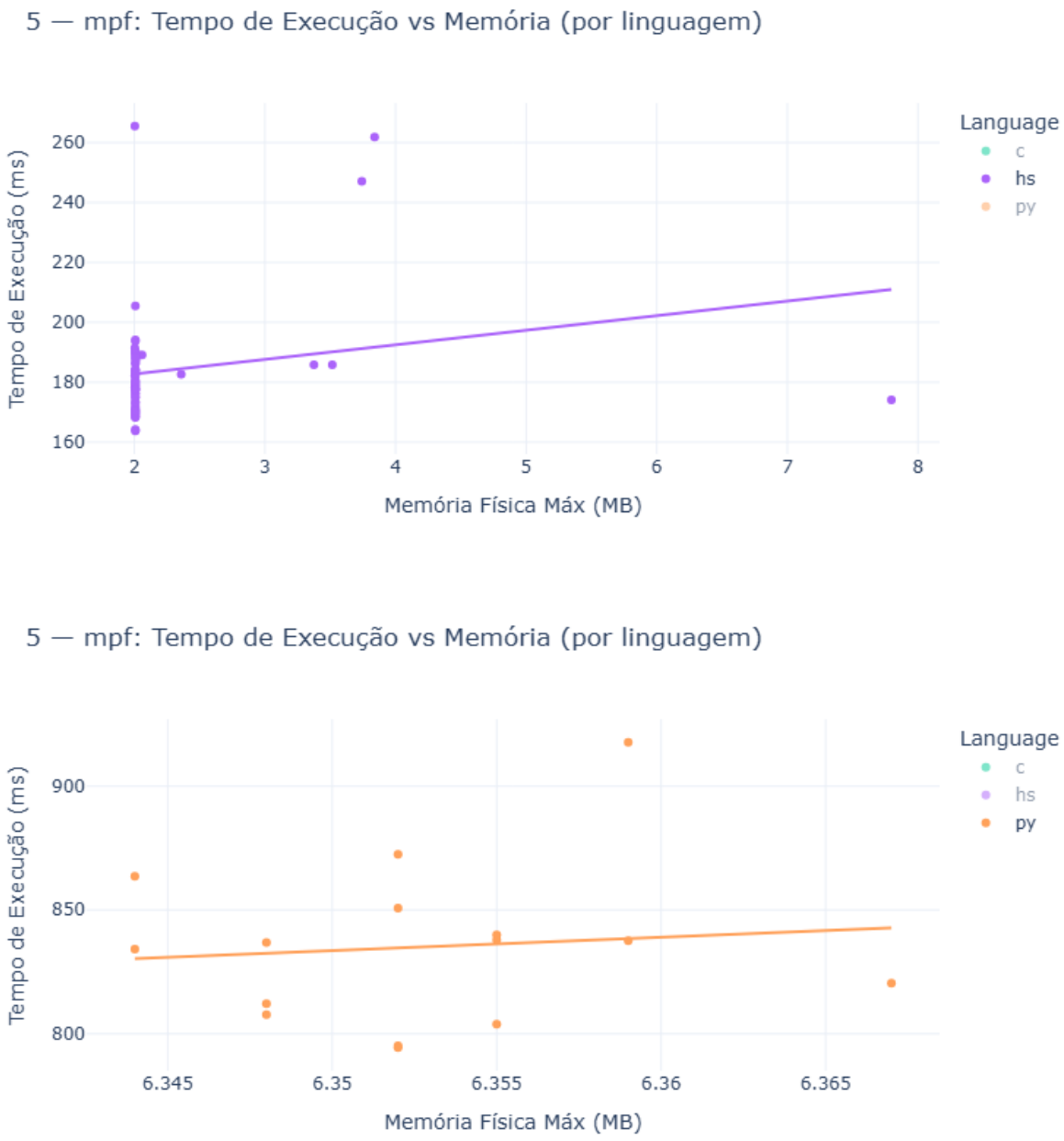


Figura 4.5: Análise de Tendência do Tempo de Execução entre Implementações Iterativa e Recursiva. Parte 2

O gráfico de Trend mostra a tendencia do uso do tempo de execução em relação à memoria física utilizada. Podemos observar que o Python apresenta uma tendência mais estável. O Haskell apresenta uma tendência mais acentuada, enquanto o C apresenta uma tendência intermediária entre as duas.

Testes realizados no metódo de Newton-Raphson unidimensional foram coletados e analisados tanto iterativamente e recursivamente, vale ressaltar que a linguagem Haskell foi implementada apenas na forma recursiva, gerando os gráficos a seguir:

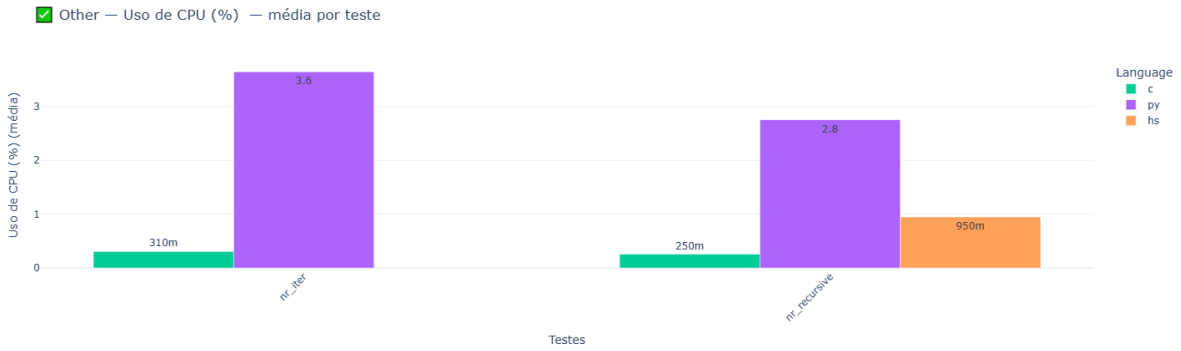


Figura 4.6: Comparação do Uso de CPU entre Implementações Iterativa e Recursiva.

Comparando o uso de CPU entre as implementações iterativa e recursiva, podemos observar que a implementação iterativa em C apresenta um desempenho muito superior em relação à implementação iterativa em Python. A implementação recursiva em Haskell, por sua vez, mostra um desempenho intermediário, ficando atrás do C, mas à frente do Python quando implementados recusivamente. Comparando C e Python iterativamente e recursivamente, ambas tem o desempenho melhor na forma recursiva, gastando menos CPU, porém o C ainda é muito mais eficiente que o Python.

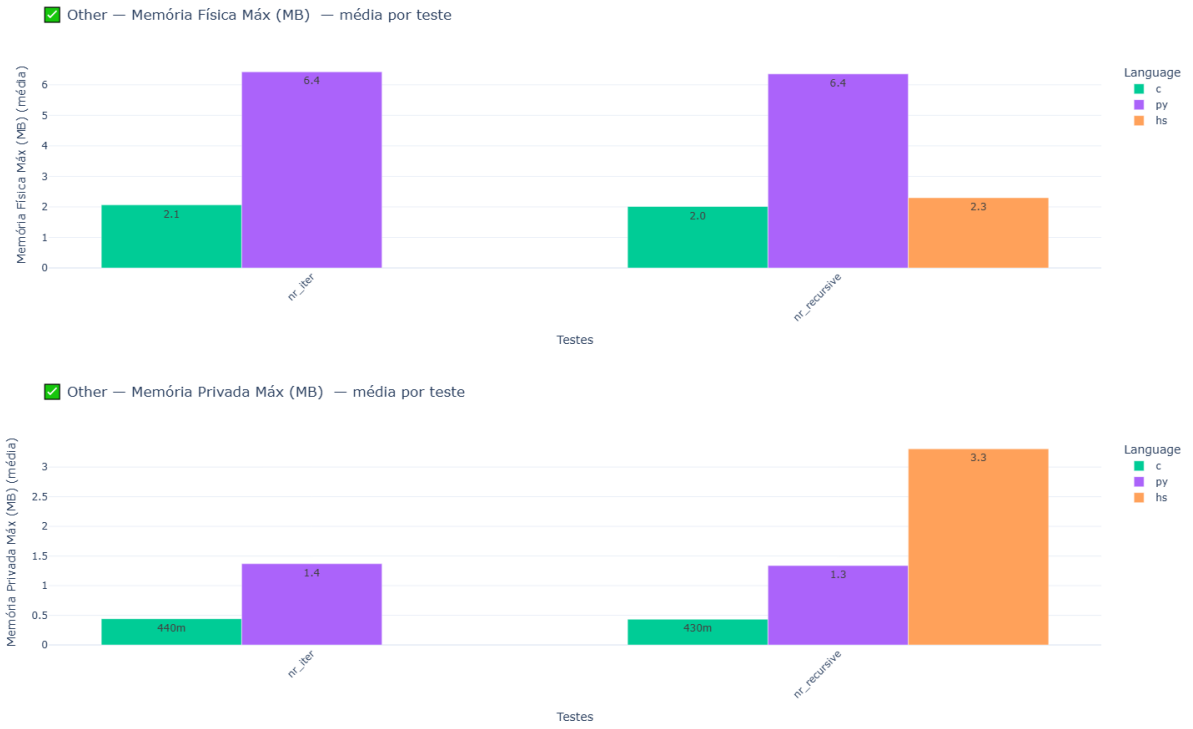


Figura 4.7: Comparação do Uso de Memória entre Implementações Iterativa e Recursiva.

A analise da memória privada mostra que a implementação iterativa em C e python não mostraram muitas diferenças quando comparadas com suas contrapartes recursivas. Já a implementação recursiva em Haskell apresentou um uso de memória privada significativamente maior, o que pode ser atribuído à natureza da recursão e à forma como o Haskell gerencia a memória para chamadas de função recursivas.

Já a memória física utilizada mostrou que a implementação iterativa em C utilizou levemente menos memória física em comparação com a implementação recursiva. A implementação iterativa em Python, por outro lado, utilizou uma quantidade de memória física que não se alterou mas foi muito maior do que ambas outras linguagens nas duas implementações, tanto iterativa quanto recursiva.

Figura 4.8: Comparação do Tempo de Execução entre Implementações Iterativa e Recursiva.

O tempo de execução mostra que a implementação iterativa em C foi a mais rápida, seguida pela implementação recursiva em C que por sua vez é seguida da implementação em Haskell. A implementação iterativa em Python foi a mais lenta entre as três, mas ainda assim apresentou um desempenho muito melhor quando é implementada recursivamente.

Figura 4.9: Violin Plot do Tempo de Execução entre Implementações Iterativa e Recursiva.

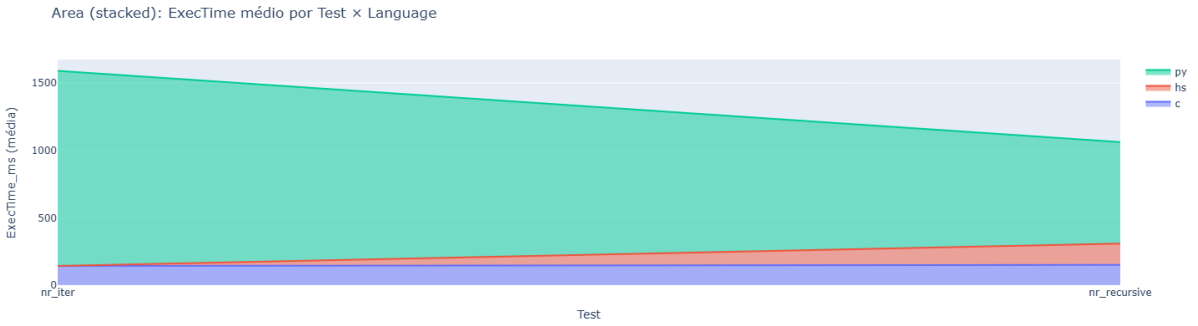


Figura 4.10: Gráfico Empilhado do Tempo de Execução entre Implementações Iterativa e Recursiva.

Ambos os graficos de Violin Plot e Área Empilhada reforçam a análise anterior, mostrando claramente a superioridade do C em termos de tempo de execução, seguido pelo Haskell e Python. Apesar disso, na implementação recursiva, o Haskell mostrou-se ligeramente mais estável do que o C. O python por sua vez apresentou uma variação maior no tempo de execução em ambas implementações, mas a recursiva ainda melhor que a iterativa.

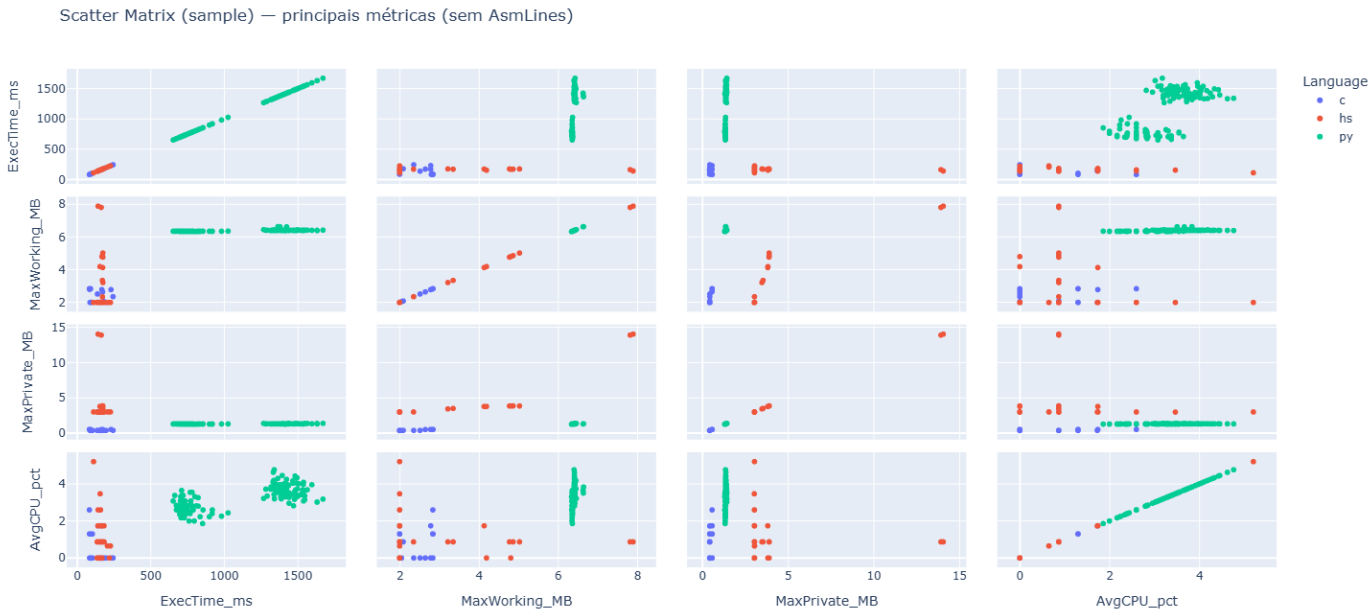


Figura 4.11: Matriz de Dispersão das Métricas Coletadas.

A matriz de dispersão das métricas coletadas oferece uma visão abrangente das relações entre diferentes variáveis, como uso de CPU, memória física, memória privada e tempo de execução. Observa-se que há correlações positivas entre o uso de CPU e o tempo de execução, indicando que implementações que consomem mais CPU tendem a levar mais tempo para completar.

Referências Citadas

Como exemplo, citamos [1].

Referências Bibliográficas

[1] Márcia Aparecida Gomes Ruggiero and Vera Lúcia da Rocha Lopes. *Cálculo numérico: aspectos teóricos e computacionais*. Pearson/Makron, Sao Paulo, 1998.