UNIVERSIDADE FEDERAL DA FRONTEIRA SUL
CAMPUS CHAPECÓ
PROGRAMA DE MESTRADO PROFISSIONAL EM REDE NACIONAL
PROFMAT

DAVID JESUS DOS REIS SILVEIRA

BACIAS DE NEWTON: UM SOFTWARE GERADOR DE IMAGENS

CHAPECÓ
2024

DAVID JESUS DOS REIS SILVEIRA

BACIAS DE NEWTON: UM SOFTWARE GERADOR DE IMAGENS

Dissertação apresentada ao Programa de Mestrado Profissional em Matemática em Rede Nacional, da Universidade Federal da Fronteira Sul – UFFS como requisito para obtenção do título de Mestre em Matemática sob a orientação do Prof. Dr. Paulo Rafael Bösing.

CHAPECÓ
2024

Bibliotecas da Universidade Federal da Fronteira Sul - UFFS

Silveira, David Jesus dos Reis
Bacias de Newton: Um software gerador de imagens / David Jesus dos Reis Silveira. -- 2024.
73 f.:il.

Orientador: Doutor Paulo Rafael Bosing

Dissertação (Mestrado) - Universidade Federal da Fronteira Sul, Programa de Pós-Graduação Profissional em Matemática em Rede Nacional, Chapecó,SC, 2024.

1. Bacias de Newton. 2. Bacia de Newton. 3. Software. 4. Software gerador de bacias. 5. Software gerador de bacias de Newton. I. Bosing, Paulo Rafael, orient. II. Universidade Federal da Fronteira Sul. III. Título.

Elaborada pelo sistema de Geração Automática de Ficha de Identificação da Obra pela UFFS com os dados fornecidos pelo(a) autor(a).

DAVID JESUS DOS REIS SILVEIRA

BACIAS DE NEWTON: UM SOFTWARE GERADOR DE IMAGENS

Dissertação apresentada ao Programa de Mestrado Profissional em Matemática em Rede Nacional da Universidade Federal da Fronteira Sul – UFFS, para obtenção do título de Mestre em Matemática.

Orientador (a): Prof. Dr. Paulo Rafael Bösing

Aprovado em: _19_/_03_/_2024__

BANCA EXAMINADORA

Documento assinado digitalmente
gov.br
PAULO RAFAEL BOSING
Data: 13/05/2024 11:41:33-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Paulo Rafael Bösing - UFFS

Documento assinado digitalmente
gov.br
FABIO DE SOUZA ALVES
Data: 13/05/2024 11:06:55-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Fábio de Souza Alves - IFSC

Documento assinado digitalmente
gov.br
MILTON KIST
Data: 13/05/2024 15:04:28-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Milton Kist – UFFS

Chapecó/SC, Março de 2024

Dedico este trabalho ao Dr Hannibal Lecter por me ajudar a observar de forma peculiar a natureza humana.

Agradecimentos

- Ao meu orientador minha sincera gratidão por sua orientação excepcional durante todo o meu trabalho de mestrado. Sua expertise, paciência e apoio foram fundamentais para o sucesso deste projeto.

  Ao longo desta jornada acadêmica, pude não apenas aprimorar meus conhecimentos na área, mas também desenvolver habilidades críticas e analíticas, graças à sua orientação perspicaz e incentivo constante.

  Agradeço por dedicar seu tempo e esforço para revisar meu trabalho, fornecer insights valiosos e orientar-me na busca pela excelência acadêmica. Suas sugestões e feedbacks foram inestimáveis e contribuíram significativamente para o aprimoramento da qualidade da pesquisa.

- À SBM que na busca da melhoria do ensino de Matemática na Educação Básica viabilizou a implementação do PROFMAT.

- À UFFS por ter sido parte fundamental da minha jornada acadêmica e por seu compromisso inabalável com a excelência educacional no ensino público.

- Aos meus professores da UFFS pela extraordinária dedicação, apoio e conhecimento que compartilharam comigo ao longo da minha jornada.

- Ao IFSC por apoiar e incentivar meu constante aprimoramento.

Ora, a realidade é constituída por essências e existências particulares e, portanto, o conhecimento verdadeiro tem que ser um conhecimento que preserve o particular sem destruí-lo numa nomenclatura abstrata.

Baruch de Espinosa

Resumo

Este trabalho apresenta o detalhamento, funcionamento e especificidades de um software desenvolvido para gerar imagens coloridas das Bacias de Newton. As Bacias de Newton são regiões do plano de Argand-Gauss (Plano Complexo) que correspondem aos valores dos chutes iniciais z_0 que produzem sequências convergentes para as raízes complexas de uma equação algébrica usando o método de Newton. O software foi desenvolvido em linguagem de programação Python e está disponível no link : Bacias de Newton: Software.

Palavras-chave: Bacias de Newton. Método de Newton. Linguagem Python. Software.

Abstract

This work presents the detailed description, functioning, and specifics of a software developed to generate colored images of Newton's Basins. Newton's Basins are regions in the Argand-Gauss plane (Complex Plane) that correspond to the initial guess values z_0 leading to convergent sequences for the complex roots of an algebraic equation using the Newton's method. The software was developed in the Python programming language and is available at the following link: Bacias de Newton: Software.

Keywords: Newton's Basins. Newton's method. Python programming language. Software.

Lista de ilustrações

Lista de abreviaturas e siglas

SBM: Sociedade Brasileira de Matemática

PROFMAT: Mestrado Profissional em Matemática em Rede Nacional

UFFS: Universidade Federal da Fronteira Sul

IFSC: Instituto Federal de Santa Catarina

Sumário

1 INTRODUÇÃO

As Bacias de Newton, fenômeno enigmático e multifacetado no campo da matemática e da dinâmica não linear, têm atraído a atenção de matemáticos, cientistas e pesquisadores por décadas. Este enigmático padrão de convergência e divergência de iterações do Método de Newton, aplicado a equações polinomiais no plano complexo, tem sido objeto de estudo e fascinação.

Tais fenômenos, muitas vezes retratadas como mapas de cores intricados e belos, são, na realidade, uma expressão visual de uma complexidade matemática profunda. Elas revelam como o chute inicial z_0 para o Método de Newton aplicado a equações polinomiais complexas se comportam com relação a convergência do método. E, ao fazê-lo, desvendam uma riqueza de fenômenos matemáticos, incluindo a existência de conjuntos fractais, a sensibilidade às condições iniciais (tais como o chute inicial z_0 e a equação polinomial) e uma estrutura que transcende os limites da intuição matemática convencional.

Este trabalho se propõe a explorar as Bacias de Newton numa perspectiva de aguçar a curiosidade para o aspecto artístico das imagens, e despertar nosso entendimento dessas estruturas fascinantes ao mesmo tempo. Apresentaremos bacias de Newton para equações polinomiais com coeficientes complexos do primeiro grau até o grau 8.

Apesar de existir uma relação entre as Bacias de Newton e a teoria dos fractais, não nos aprofundaremos nisso. O objetivo é despertar a curiosidade. Para explorar esse ramo veja Burton (2009), Epureanu e Greenside (1998) e Sahari, Djellit et al. (2006).

A dissertação segue uma estrutura lógica e sequencial que visa apresentar de forma clara e progressiva o desenvolvimento da pesquisa. Na Seção 1.1, exploramos a escolha da linguagem Python, destacando sua relevância para a abordagem adotada. O Capítulo 2 aborda aspectos da linguagem Python relacionados à representação de polinômios e números complexos.

O Capítulo 3 adentra o método de Newton, fornecendo não apenas uma descrição do método, mas também contextualizando historicamente a busca por raízes de polinômios. Questões relacionadas à convergência do método de Newton também são discutidas nesta seção. O Capítulo 4 formaliza o conceito de "Bacias de Newton", estabelecendo uma base teórica para o trabalho.

No Capítulo 5 são apresentadas as visualizações das 'Bacias de Newton' para 19 equações polinomiais, variando do grau 1 até o grau 8. Este capítulo serve como o núcleo da pesquisa, oferecendo dados detalhados dos resultados obtidos.

A Conclusão, Capítulo 6, resume os principais achados, discute implicações práticas

e teóricas e oferece sugestões para trabalhos futuros. Ao final, nas Referências, são listadas as fontes utilizadas ao longo da dissertação. O Apêndice contém o código fonte do software gerador das "Bacias de Newton", enquanto os Anexos incluem informações técnicas adicionais sobre as imagens produzidas.

Essa estrutura proporciona uma narrativa coesa, guiando o leitor através do raciocínio e desenvolvimento da pesquisa de maneira lógica e compreensível.

Ao avançar em nosso estudo das Bacias de Newton, esperamos contribuir para o enriquecimento do campo da matemática e das ciências relacionadas, bem como abrir novas perspectivas para a aplicação desses conceitos em contextos práticos. Por meio dessa exploração, pretendemos desvendar ainda mais a profundidade das Bacias de Newton e revelar as muitas facetas de sua influência no mundo da matemática e da ciência contemporânea.

1.1 A escolha pelo Python

Uma justificativa técnica para o uso da linguagem Python em nosso trabalho inclui vários argumentos que destacam as vantagens e benefícios específicos dessa linguagem.

Ampla Comunidade e Ecossistema de Bibliotecas: Python possui uma comunidade de desenvolvedores e cientistas de dados extremamente ativa. Isso resulta em um vasto ecossistema de bibliotecas e recursos prontos para uso, como NumPy, SciPy, pandas, Matplotlib, entre outros, que facilitam a análise e a visualização de dados, bem como a implementação de algoritmos científicos.

As referências e modelos de tais bibliotecas foram obtidas em Ciência (2023) e também Matplotlib (2023).

Legibilidade e Clareza de Código: A sintaxe simples e legível do Python torna o código mais fácil de escrever, ler e manter. Isso é fundamental para a colaboração em projetos de pesquisa e para garantir a compreensão do código ao longo do tempo.

Gratuidade e Código Aberto: Python é uma linguagem de código aberto, o que significa que é gratuito para uso e distribuição. Isso reduz custos associados a licenças de software e permite que outros pesquisadores reproduzam e validem seus resultados sem barreiras financeiras.

Plataforma Cruzada: Python é compatível com várias plataformas, incluindo Windows, macOS e Linux. Isso facilita a colaboração e a portabilidade de projetos de pesquisa entre diferentes sistemas operacionais.

Integração com Outras Linguagens: Python é conhecido por sua capacidade de integração com outras linguagens de programação, como C/C++, Fortran e Java. Isso permite aproveitar bibliotecas existentes e usar código de alto desempenho quando necessário.

Ferramentas de Visualização: Bibliotecas como Matplotlib e Seaborn permitem criar visualizações de dados de alta qualidade e personalizadas, o que é fundamental para a apresentação de resultados em trabalhos científicos.

Suporte à Computação Científica e Aprendizado de Máquina: Python é amplamente utilizado em áreas científicas, desde física e biologia até aprendizado de máquina e inteligência artificial. Isso torna mais fácil encontrar soluções para problemas científicos em diversas disciplinas.

Acesso a Dados e APIs: Python oferece uma variedade de bibliotecas para acessar e processar dados, seja por meio de APIs da web, leitura de arquivos de diferentes formatos ou conexão a bancos de dados.

Documentação Extensa: Python tem uma documentação abrangente e recursos de aprendizado online, tornando-o uma escolha acessível para pesquisadores que desejam aprender e usar a linguagem.

Reprodutibilidade da Pesquisa: O código em Python é altamente reprodutível, permitindo que outros pesquisadores recriem e validem facilmente os resultados de sua pesquisa, o que é fundamental para a ciência aberta e transparente.

Em resumo, o uso do Python em trabalhos científicos é justificado por sua eficácia, versatilidade, suporte da comunidade e capacidade de acelerar o processo de pesquisa, análise de dados e apresentação de resultados. É uma escolha sólida para pesquisadores de diversas disciplinas devido às suas capacidades de resolução de problemas e ao seu amplo conjunto de ferramentas e bibliotecas.

A versão utilizada foi Python 3.8.10 [MSC v.1928 64 bit (AMD64)] IPython 8.12.0 – An enhanced Interactive Python, disponível para download em Python (2019). As bibliotecas para tratamento de imagem usadas no programa foram Matplotlib e Pillow . Para instalá-las é preciso usar um gerenciador de pacotes, como o pip.

A programação do código foi feita no ambiente Spyder IDE 5.4.3 disponível para download em Team (2018).

Para instalar o Pillow, você pode usar o seguinte comando no terminal ou prompt de comando:

```
pip install Pillow
```

Para o Matplotlib, o comando é:

```
pip install Matplotlib
```

Os livros utilizados como consulta para sintaxe de comandos da linguagem foram Matthes (2016) e Sweigart (2015). Os sites utilizados foram Pillow (2011), Ciência (2023), e Matplotlib (2023).

2 Metodologia

Este trabalho utilizou a linguagem de programação Python para criar visualizações das Bacias de Newton, que são as regiões no plano complexo dos chutes iniciais que produzem sequências que convergem para as raízes de equações algébricas usando o Método de Newton. A visualização será realizada com a ajuda das bibliotecas Matplotlib e Pillow, aproveitando as capacidades gráficas dessas ferramentas para criar representações esteticamente agradáveis das Bacias de Newton.

1 - Configuração do Ambiente de Desenvolvimento:

Foi instalada a versão Python 3.8.10 64-bit . Foi instalado e configurado o ambiente de desenvolvimento Spyder na versão 5.4.3.

2 - Instalação das Bibliotecas:

Através do gerenciador de pacotes 'pip' foram instaladas as bibliotecas Matplotlib e Pillow.

3 - Implementação do Código Python:

Foi desenvolvido um programa em Python que varre uma região retangular de 2000 pixels x 1500 pixels, portanto com 3.000.000 pixels. Cada ponto dessa região é usado como z_0 no Método de Newton para uma determinada equação algébrica. O Método de Newton foi utilizado para encontrar as raízes da equação. Foi criada uma matriz representando a região do plano complexo a ser explorada, a esta região, ponto a ponto foi aplicado o Método de Newton usando cada um desses pontos da região complexa como z_0, ou seja, como 'chute' inicial do método.

Atribuímos cores aos chutes iniciais que convergem para determinadas raízes de forma a criar uma imagem esteticamente atraente das Bacias de Newton. O grau do polinômio que é igual a quantidade de raízes complexas de acordo com o Teorema Fundamental da Álgebra, também é igual à quantidade de cores utilizada. Utilizamos no máximo polinômios de grau 8, ou seja, 8 cores, cada uma para o conjunto de pontos que convergem para uma raiz específica.

Quando usamos o Método de Newton pode acontecer do z_0 utilizado não convergir para nenhuma das raízes do polinômio, esse fato é intrínseco ao método e não discutiremos seus pormenores. À estes pontos atribuímos a cor preta no gráfico. Ao redor das raízes do polinômio atribuímos um pequeno círculo de cor branca. Também adicionamos um segundo gráfico para cada polinômio com a representação das suas raízes na forma polar utilizando as mesmas cores do gráfico das Bacias, tornando assim de fácil visualização. Enfim, utilizamos a biblioteca Pillow para criar o gráfico das Bacias de Newton, e posteriormente

a biblioteca Matplotlib para colocar o grid e a legenda. O gráfico das Bacias são gerados com as dimensões de 2000 x 1500 pixels, ou seja, 2000 pixels de largura por 1500 pixels de altura. Depois disso são adicionados os eixos e a legenda mantendo a proporção das dimensões originais.

Para assuntos relacionados a sintaxe da linguagem Python utilizamos Matthes (2016) e Sweigart (2015). A implementação de um número complexo é um tipo de dado intrínseco da biblioteca Numpy do Phyton, ou seja, não usamos par ordenado para representação de um número complexo. Todas as equações usadas no trabalho são complexas, não há equações com variáveis reais. Nos casos de não convergência foram usados dois critérios de parada no método implementado: Quando a derivada dá zero para algum passo iterativo, ou quando se excede a quantidade de passos M=30 .

Nas implementações das Bacias consideramos equações polinomiais do grau 1 até o grau 8. As cores associadas às raízes são apresentadas na tabela abaixo. Nela, a Cor(i) está associada a raiz z_i, i=0,1,2,3,4,5,6,7, da equação polinomial.

Figura 1 – Tabela de cores associadas às raízes

As cores da Figura 1 correspondem respectivamente aos seguintes códigos RGB : [(0,250,154), (255,64,64), (0,255,255), (255,153,18), (191,62,255), (255,255,0), (255,0,255), (30,144,255)].

O tempo de execução do programa depende do hardware e também do grau da equação polinomial em questão. Esse tempo pode variar de 2 minutos a 2 horas e 40 minutos segundo nossos testes. Quanto maior o desempenho do hardware menos tempo se gasta na execução e quanto maior o grau da equação polinomial, mais tempo se gasta na execução.

Além da disponibilização do software em Bacias de Newton: Software, encontra-se em processo também a sua inserção na plataforma eduCAPES (2024).

Em geral foi uma experiência muito gratificante a elaboração deste trabalho sobretudo pela dimensão artística das imagens. Com relação ao tempo de confecção deste trabalho, a maior parte dele foi dedicada à escrita do código, foram gastos 9 meses para se

obter as imagens de acordo com as nossas expectativas.

2.1 Representações em Python

Utilizamos Numpy, que é a biblioteca para computação numérica em Python, os números complexos são representados pelo tipo de dados numpy.complex128. Este tipo de dado representa números complexos usando 128 bits de precisão de ponto flutuante, garantindo uma alta precisão nas operações.

A representação de um número complexo em NumPy é feita usando a notação padrão Python, onde a parte real e a parte imaginária são representadas como números de ponto flutuante, sendo a parte imaginária multiplicada pela unidade imaginária j. Por exemplo:

z1 = (2.5 + 1j)

z2 = (1.0 - 2j)

É importante ressaltar que na linguagem a unidade imaginária 'i' é substituída pelo símbolo 'j' .

Os polinômios são representados através de arrays. Cada elemento do array representa um coeficiente do polinômio, onde o índice do elemento indica a potência correspondente à variável.

Por exemplo, o array [1, -2, 0, 3] representa o polinômio $P(z) = 1.z^3 - 2.z^2 + 0.z + 3$.

No arquivo de informações técnicas sobre as imagens, usamos a representação 3[1, -2, 0, 3] para representar o mesmo polinômio citado acima, contudo acrescentamos o grau do polinômio '3' (primeiro número na representação antes da abertura do colchete).

3 Método de Newton

Neste capítulo, inicialmente exploraremos uma breve jornada pela história da matemática, focando na evolução da busca por raízes de equações polinomiais. Destacaremos marcos significativos que moldaram essa área do conhecimento ao longo do tempo.

A segunda seção abordará o Método de Newton, apresentando uma descrição de sua aplicação na resolução de equações polinomias e os passos que caracterizam o método.

A sequência do capítulo é dedicada à implementação prática do Método de Newton no código utilizado para gerar as 'Bacias de Newton'. Exemplificaremos como o método é incorporado nesse contexto específico, proporcionando uma compreensão mais tangível de sua aplicação na geração das imagens.

Em seguida, abordaremos questões sobre a convergência do Método de Newton, e indicaremos obras que abordam esse tema com maior rigor.

Para aqueles que desejam se aprofundar no assunto, ao decorrer do capítulo, forneceremos uma lista de referências relevantes. Essas referências incluirão obras que tratam não apenas do Método de Newton, mas também da história da busca por raízes de equações polinomiais e questões associadas à convergência. Essa abordagem visa enriquecer a compreensão do leitor e incentivá-lo a explorar mais a fundo os aspectos apresentados ao longo do capítulo.

3.1 Busca por raízes de equações

As informações históricas mencionadas nesta seção, foram extraídas principalmente de Eves (2011), contudo também nos baseamos em Boyer e Merzbach (2019) e Roque (2012).

A busca por soluções para equações algébricas tem sido uma pedra angular da matemática desde tempos antigos, proporcionando desafios intelectuais que têm evoluído em conjunto com o desenvolvimento do pensamento matemático. Essa investigação não apenas gerou ferramentas essenciais para resolver problemas práticos, mas também estabeleceu os fundamentos para a compreensão profunda de números complexos e sua interação com o mundo real.

O estudo das raízes das equações polinomiais remonta a civilizações antigas, com vestígios de técnicas para a resolução de equações quadráticas datando de 2000 a.C. Porém, a busca por soluções gerais para equações de graus superiores continuou ao longo dos séculos. No século XVI, matemáticos notáveis como Scipione del Ferro, Niccolò Fontana

Tartaglia e Gerolamo Cardano desempenharam papéis cruciais na resolução de equações cúbicas e quartas. O resultado notável foi a fórmula de Cardano, uma expressão que permitia a resolução de equações cúbicas de forma geral, criando um marco significativo na história da matemática.

No entanto, o desafio de encontrar uma fórmula geral para equações de quinto grau e acima permaneceu um enigma matemático por muitos anos. Um esforço intelectual considerável foi dedicado a esse problema, até que, no século XIX, Niels Henrik Abel e Évariste Galois demonstraram que tal fórmula resolutiva não poderia existir. Suas contribuições à teoria das equações algébricas culminaram no famoso "Teorema de Abel-Galois", estabelecendo os limites da resolução analítica dessas equações. Para quem deseja se profundar uma referência clássica na teoria de Galois é o livro Stewart (2015), que oferece uma introdução acessível a essa teoria, incluindo o Teorema de Abel-Galois. Outros livros padrão de álgebra abstrata e teoria de Galois, como Artin (2013) e Garling (1986) também abordam o Teorema de Abel-Galois .

Além disso, ao longo da história, surgiram conceitos fundamentais que são essenciais para a compreensão da teoria das equações algébricas, como os números complexos, a teoria de grupos e a álgebra abstrata. Tais conceitos desempenharam um papel crítico na formulação do Teorema Fundamental da Álgebra, que estabelece que toda equação polinomial com coeficientes complexos possui pelo menos uma raiz complexa. Este teorema não apenas teve implicações profundas na teoria das equações, mas também se tornou um dos resultados mais importantes e ubíquos da matemática. Para explorar o teorema e sua história, é útil consultar livros de história da matemática, como Boyer e Merzbach (2019) e Edwards (2012), que fornecem contextos históricos e detalhes sobre o desenvolvimento desse teorema.

O problema da impossibilidade de encontrar fórmulas resolutivas para equações de grau superior a 4 está relacionado à solução analítica de equações polinomiais. O método de Newton oferece uma abordagem alternativa, baseada em aproximações iterativas, para encontrar raízes de equações, incluindo equações polinomiais. A relevância do método de Newton é acentuada nas seguintes características:

Versatilidade na Aproximação de Raízes: O método de Newton não se limita a equações polinomiais, mas pode ser aplicado a uma ampla variedade de equações. Ele é eficaz na aproximação de raízes reais e complexas de equações transcendentais e polinomiais.

Eficácia na Prática: O método de Newton é amplamente utilizado em ciência, engenharia e computação devido à sua rapidez de convergência. Ele é uma ferramenta fundamental para encontrar raízes em problemas de otimização, análise numérica, simulação e muitos outros campos.

Importância na Computação Numérica: Algoritmos de cálculo numérico muitas

vezes dependem do método de Newton para encontrar soluções de sistemas de equações não lineares. Isso inclui a solução de equações diferenciais, otimização e modelagem numérica, que são fundamentais para a pesquisa científica e tecnológica.

O método de Newton desempenha um papel crítico na superação da impossibilidade de encontrar fórmulas resolutivas gerais para equações polinomiais de grau superior a 4. Ele oferece uma abordagem prática e eficaz para a aproximação de raízes complexas e reais de equações com um nível desejado de precisão, preenchendo a lacuna entre a teoria das equações algébricas e a aplicação prática. Sua versatilidade e eficácia o tornam uma ferramenta essencial em muitos campos da matemática, ciência e engenharia.

As Bacias de Newton, que surgiram como um conceito intrigante na análise de raízes de equações algébricas, têm uma rica história e desempenham um papel importante nas aplicações matemáticas modernas. Seu desenvolvimento ao longo do tempo reflete não apenas o desejo de entender a dinâmica das raízes de equações, mas também suas ligações com diversas áreas da matemática, ciência e engenharia.

Aspectos Históricos:

As Bacias de Newton devem seu nome a Sir Isaac Newton, que formulou o método de Newton no século XVII para a aproximação de raízes de equações. O método de Newton, que é uma técnica iterativa, permitiu o cálculo eficiente de raízes reais de equações complexas e desempenhou um papel fundamental na história da análise numérica.

Durante o século XIX, a compreensão das Bacias de Newton avançou à medida que matemáticos como Pierre Fatou e Gaston Julia investigaram as propriedades das órbitas das iterações de equações polinomiais. Eles exploraram a dinâmica das raízes complexas das equações, o que resultou na descoberta de estruturas fractais nas chamadas "Bacias de Atração". Essas figuras altamente intrincadas, compostas por regiões que convergem para diferentes raízes, estabeleceram um novo paradigma na teoria das equações algébricas.

Aplicações Modernas:

As Bacias de Newton e os fractais associados não são apenas curiosidades matemáticas, mas também têm aplicações modernas em diversas áreas:

Computação Gráfica e Arte Fractal: As imagens das Bacias de Newton e outros fractais são frequentemente usadas na computação gráfica, resultando em paisagens fractais e designs artísticos complexos. Um livro que aborda a relação entre Computação Gráfica, Arte Fractal e Bacias de Newton é Peitgen e Richter (1986). Este livro é uma exploração visual e matemática das belezas estéticas dos fractais e dos sistemas dinâmicos complexos.

Teoria do Caos: O estudo da dinâmica das Bacias de Newton está intimamente relacionado à teoria do caos, que tem aplicações em física, biologia, economia e engenharia. Para aprofundar na relação entre Teoria do Caos e Bacias de Newton leia Gleick (2008) .

Sistemas Dinâmicos Complexos: A compreensão da dinâmica das Bacias de Newton é essencial para analisar sistemas dinâmicos complexos, como sistemas de partículas, osciladores caóticos e sistemas biológicos. Veja Strogatz (1996).

Matemática Experimental: O estudo das Bacias de Newton e dos fractais relacionados é um exemplo de matemática experimental, que é uma abordagem em crescimento na pesquisa matemática. Veja Press (2007).

Ensino e Divulgação Científica: As Bacias de Newton e os fractais são ferramentas visuais poderosas para o ensino e a divulgação científica, ajudando a tornar conceitos matemáticos complexos mais acessíveis. Veja Janos (2009) e Ellenberg (2015).

As Bacias de Newton são um exemplo notável de como os aspectos históricos da matemática se entrelaçam com aplicações modernas em uma variedade de campos. Elas não apenas enriquecem nossa compreensão da teoria das equações, mas também emprestam sua beleza e complexidade a áreas diversas, tornando-as uma área rica e multidisciplinar dentro da matemática e da ciência.

3.2 Método de Newton

O desenvolvimento presente nesta seção foi extraído de Burton (2009), Epureanu e Greenside (1998), Levy (2018), Yau e Ben-Israel (1998), McClure (2006), Hubbard, Schleicher e Sutherland (2001).

Seja uma função complexa $F : \mathbb{C} \to \mathbb{C}$ que assumimos ser diferenciável com $F'(z)$ sua função derivada. Dado uma tolerância (erro) $\varepsilon > 0$, a quantidade máxima de iterações M, e um número complexo z_0 (chute inicial) para iniciar o processo iterativo, buscamos uma aproximação para uma raiz da equação $F(z) = 0$, calculando z_1 através da igualdade :

$$z_1 = z_0 - \frac{F(z_0)}{F'(z_0)} \quad .$$

Depois, calculamos z_2 da seguinte forma:

$$z_2 = z_1 - \frac{F(z_1)}{F'(z_1)} \quad ,$$

e generalizando, z_{n+1} depende de z_n como segue:

$$z_{n+1} = z_n - \frac{F(z_n)}{F'(z_n)} \qquad n \in \mathbb{N}.$$

Nesta sequência de passos iterativos, se tivermos $F'(z_n) = 0$, dizemos que z_0 não gera uma sequência $\{z_n\}$ convergente, pois não existe z_{n+1} e encerramos as iterações para esse z_0.

A cada passo que for possível calcular z_{n+1}, ou seja, que exista z_{n+1}, verificamos duas condições nesta ordem:

1) $|F(z_{n+1})| < \varepsilon$

2) $n + 1 > M$

Se a Condição 1) for verdadeira dizemos que z_0 gera uma sequência $\{z_n\}$ convergente e z_{n+1} é uma raiz aproximada de $F(z) = 0$.

Se a Condição 2) for verdadeira dizemos que z_0 não gera uma sequência convergente e encerramos as iterações para esse z_0.

Com isto descrevemos todos os passos para entender o método aplicado neste trabalho. Todas as aplicações do método são feitas em equações complexas, ou seja, não há aplicações em equações reais neste trabalho.

3.3 Convergência

Como mencionado anteriormente, as causas de convergência ou divergência são um universo a parte. Para o leitor interessado em tal aprofundamento consultar Burden (2011), Dukkipati (2010), Dukkipati (2023) e Epperson (2021).

Vamos citar algumas possíveis causas:

O Método de Newton pode convergir nos números complexos sob certas condições. No entanto, ao contrário do caso real, a convergência nos complexos pode ser mais sensível à escolha do ponto inicial e à complexidade da função.

As condições para a convergência do Método de Newton nos números complexos incluem:

1) Continuidade e Derivabilidade:

A função $F(z)$ deve ser contínua e diferenciável na região que contém a raiz que está sendo procurada.

2) Escolha Adequada do Ponto Inicial:

A convergência pode depender significativamente do ponto inicial. Em alguns casos, o método pode convergir para diferentes raízes ou pode não convergir se o ponto inicial estiver muito distante da raiz desejada.

3) Derivada Não Nula:

A derivada $F'(z)$ não deve ser zero no caminho de iteração. $F'(z) = 0$ resulta em não convergência.

4) Condição de Contratibilidade:

O método requer que a razão $\frac{F(z_n)}{F'(z_n)}$ não cresça indefinidamente durante as iterações. Isso é muitas vezes expresso como a condição de contratibilidade local.

Condições Globais de Convergência:

Condições globais, como a função ser monótona ou ter um comportamento 'simples' na vizinhança da raiz, podem facilitar a convergência. No entanto, essas condições podem variar dependendo do contexto específico. Lembrando que, mesmo sob essas condições, não há garantia de convergência em todos os casos. A escolha adequada do ponto inicial continua sendo um fator crítico, e algumas equações podem apresentar regiões onde o método não converge ou converge para soluções diferentes. A análise da convergência do Método de Newton nos complexos é, muitas vezes, uma tarefa mais complexa e dependente do contexto específico da função em questão.

3.4 Exemplos

Vejamos alguns exemplos de sequências $\{z_n\}$ para equações polinomiais de graus 1, 5 e 7. Todos os exemplos tem uma tolerância (erro) $\varepsilon = 10^{-10}$ e quantidade máxima de iterações M=30.

A Figura 2 trata de uma sequência simples para uma equação polinomial de grau 1 onde com apenas uma iteração se alcança a raiz dentro da precisão estabelecida.

```
1[ 1.+0.j -3.+3.j] --> Converge !

Índice (n)       Número Complexo (zn)    Valor Polinômio P(zn)   Módulo do Valor Polinômio

0                -4.000+3.000j           -7.000+6.000j           9.21954445729288707412
1                3.000-3.000j            0.000+0.000j            0.00000000000000000000
```

Figura 2 – Iterações para a equação $(1 + 0j)z^1 + (-3 + 3j)z^0 = 0$

Na Figura 3 temos a sequência de iterações para uma equação de grau 5 onde foram necessárias 12 iterações até que se atingisse uma das raízes dentro da precisão estabelecida.

```
5[-1.+0.j  0.+0.j  1.+1.j  0.+0.j  0.+0.j  1.+0.j] --> Converge !

Índice (n)       Número Complexo (zn)    Valor Polinômio P(zn)   Módulo do Valor Polinômio

0                -4.000+3.000j           -3188.000+398.000j      3212.74773363860595054575
1                -3.203+2.378j           -1046.007+133.532j      1054.49606606922066021070
2                -2.566+1.875j           -343.378+45.272j        346.34967237971363829274
3                -2.057+1.467j           -112.748+15.548j        113.81480303838532108784
4                -1.652+1.134j           -36.963+5.408j          37.35631108019597235170
5                -1.331+0.862j           -12.013+1.891j          12.16052079955987430537
6                -1.084+0.643j           -3.775+0.648j           3.83004360890834405495
7                -0.908+0.479j           -1.058+0.203j           1.07720067839746747040
8                -0.810+0.380j           -0.207+0.049j           0.21237368747729762353
9                -0.781+0.347j           -0.014+0.005j           0.01528566745188686751
10               -0.779+0.344j           -0.000+0.000j           0.00009775040038534339
11               -0.779+0.344j           -0.000+0.000j           0.00000000406883925229
12               -0.779+0.344j           0.000-0.000j            0.00000000000000011102
```

Figura 3 – Iterações para a equação $(-1 + 0j)z^5 + (-1 + 0j)z^4 + (-1 + 0j)z^3 + (-1 + 0j)z^2 + (-1 + 0j)z^1 + (-1 + 0j)z^0 = 0$

Na sequência da Figura 4 temos um exemplo peculiar retratando um caso de não convergência. Escolhemos esta equação e este z_0 observando a Figura 29 onde fica evidente que z_0 pertence a região onde um conjunto de pontos que convergem para nenhuma raiz (parte inferior esquerda).

```
7[0.1+0.j 1. +1.j 0. +0.j 0. +0.j 0. +0.j 0. +0.j 1. +0.j] --> Não Converge !
```

Índice (n)	Número Complexo (zn)	Valor Polinômio P(zn)	Módulo do Valor Polinômio
0	-15.000-10.000j	15898438.500+6468750.000j	17164063.426263090223073959 35
1	-13.176-9.294j	5418642.367+1829494.595j	5719155.14550936128944158554
2	-11.615-8.912j	1881349.249+366345.424j	1916685.67268255399540066719
3	-10.286-8.978j	681335.429-91456.950j	687446.24562589381821453571
4	-9.479-9.831j	145829.718-323871.498j	355188.75843714806251227856
5	-10.157-9.948j	50835.233+126868.477j	136674.17961620251298882067
6	-10.002-9.989j	8715.913+1897.617j	8920.09424804047739598900
7	-10.000-10.000j	-14.344-40.047j	42.53831997071957005119
8	-10.000-10.000j	0.001+0.000j	0.00095966641476683922
9	-10.000-10.000j	-0.000-0.000j	0.00000000123114341375
10	-10.000-10.000j	0.000-0.000j	0.00000000059050809108
11	-10.000-10.000j	0.000-0.000j	0.00000000059050809108
12	-10.000-10.000j	0.000-0.000j	0.00000000059050809108
13	-10.000-10.000j	0.000-0.000j	0.00000000059050809108
14	-10.000-10.000j	0.000-0.000j	0.00000000059050809108
15	-10.000-10.000j	0.000-0.000j	0.00000000059050809108
16	-10.000-10.000j	0.000-0.000j	0.00000000059050809108
17	-10.000-10.000j	0.000-0.000j	0.00000000059050809108
18	-10.000-10.000j	0.000-0.000j	0.00000000059050809108
19	-10.000-10.000j	0.000-0.000j	0.00000000059050809108
20	-10.000-10.000j	0.000-0.000j	0.00000000059050809108
21	-10.000-10.000j	0.000-0.000j	0.00000000059050809108
22	-10.000-10.000j	0.000-0.000j	0.00000000059050809108
23	-10.000-10.000j	0.000-0.000j	0.00000000059050809108
24	-10.000-10.000j	0.000-0.000j	0.00000000059050809108
25	-10.000-10.000j	0.000-0.000j	0.00000000059050809108
26	-10.000-10.000j	0.000-0.000j	0.00000000059050809108
27	-10.000-10.000j	0.000-0.000j	0.00000000059050809108
28	-10.000-10.000j	0.000-0.000j	0.00000000059050809108
29	-10.000-10.000j	0.000-0.000j	0.00000000059050809108
30	-10.000-10.000j	0.000-0.000j	0.00000000059050809108

Figura 4 – Iterações para a equação $(0.1+0j)z^7+(1+1j)z^6+0jz^5+0jz^4+0jz^3+0jz^2+0jz^1+(1+0j)z^0=0$

4 Bacias de Newton

Uma bacia de atração está relacionada aos métodos iterativos, como o método de Newton, utilizado para encontrar raízes de equações complexas. A ideia é visualizar como diferentes pontos de inicialização (chute inicial z_0) geram sequências que convergem para diferentes raízes da equação no plano complexo.

Vamos considerar o método de Newton descrito na Seção 3.2 para encontrar raízes da equação F(z) = 0 no plano complexo. A bacia de atração (ou Bacia de Newton) de uma raiz específica é a região do plano complexo formada pelos chutes iniciais que resultarão em sequências que convergem para essa raiz específica após algumas iterações do método de Newton.

A visualização da bacia de atração no plano complexo pode ser feita colorindo as regiões de acordo com a raiz para a qual os pontos geram sequências que convergem. Cada cor representa uma raiz diferente, e as fronteiras entre as cores mostram onde a transição de uma bacia de atração para outra ocorre.

Essa abordagem ajuda a entender como o método de Newton se comporta para diferentes pontos iniciais e fornece insights sobre as propriedades da função complexa que está sendo analisada.

A definição usada aqui foi extraída de Burton (2009), contudo, alguns símbolos foram substituídos para que haja coerência com os símbolos utilizados em todo o trabalho. Assim essa substituição somente acrescenta na compreensão.

Definição : Seja r uma raiz da equação $F(z) = 0$, a bacia de atração de r denotada por $B(r)$ é o conjunto de todos os números z_0, tais que o Método de Newton com chute inicial z_0 converge para r. Em termos matemáticos, $B(r) = \{z_0 \ / \ \{z_n\} \ \ converge\ para\ r\ \}$ em que $\{z_n\}$ é sequência do método de Newton gerada com chute inicial $z_0 \in \mathbb{C}$.

4.1 Equações Polinomiais e suas Bacias

Nessa seção exploraremos as estruturas das equações polinomiais, o que representa o ponto central de nossa investigação, abrigando não menos que 19 equações polinomiais, cada uma delas acompanhada por uma análise visual enriquecedora nas formas das Bacias de Newton. Ao avançar neste capítulo, convidamos você a apreciar essas imagens impressionantes e descobertas matemáticas. Prepare-se para uma jornada que promete expandir sua compreensão e apreciação pela complexidade e beleza das raízes polinomiais e suas bacias associadas. Temos a oportunidade para estimular nossa percepção das equações polinomiais e suas propriedades únicas. O capítulo é uma jornada guiada, onde cada equação é analisada quanto às suas Bacias de Newton, proporcionando insights valiosos sobre os comportamentos locais de suas raízes.

Ao longo dessas páginas, você encontrará para cada uma das 19 equações, 2 figuras:

1) Representação cartesiana das raízes e bacias

Cada figura é dimensionada para 2000 pixels de largura por 1500 pixels de altura. As equações polinomiais aqui apresentadas vão do grau 1 até o grau 8 em ordem crescente. É crucial notar que todas as análises são conduzidas sob condições padronizadas. A quantidade máxima de iterações é fixada em M = 30, proporcionando uma visão clara e consistente das características dinâmicas das equações. Além disso, a tolerância é mantida no valor $\varepsilon = 10^{-10}$, garantindo precisão nos resultados e consistência nas comparações.

2) Representação polar das raízes

Cada equação aqui apresentada possui também uma representação polar de suas raízes. Essa abordagem não apenas fornece uma perspectiva visual única, mas também enriquece nossa compreensão da distribuição espacial das raízes no plano complexo. Representar raízes complexas em um gráfico polar pode oferecer várias vantagens e insights sobre o comportamento dessas raízes. Aqui estão alguns pontos positivos: Visualização Clara: O gráfico polar oferece uma maneira clara e intuitiva de visualizar as raízes complexas. Ao plotar as raízes em um plano polar, é possível observar sua distribuição, padrões e simetrias de uma forma mais direta do que em um gráfico cartesiano. Compreensão da Modulação e Amplitude: No gráfico polar, a magnitude (ou módulo) de uma raiz complexa é representada pela distância do ponto ao centro do gráfico, enquanto o argumento (ou fase) é representado pelo ângulo formado entre o eixo real positivo e o segmento que une o ponto à origem. Isso permite uma compreensão mais clara da relação entre a modulação (magnitude) e a amplitude (fase) das raízes complexas. Identificação de Padrões e Simetrias: O gráfico polar pode revelar padrões e simetrias nas raízes complexas que podem não ser imediatamente aparentes em um gráfico cartesiano. Por exemplo, certas equações

polinomiais podem ter raízes complexas dispostas simetricamente ao redor do centro do gráfico polar.

4.1.1 Equações Polinomiais de Grau 1

Figura 5 – Bacias de Newton para a equação: $(1+0j)z^1+(-3+3j)z^0=0$; R=[-4.4;4.4]x[-3.3j;3.3j]; Pixels não convergentes: $0 = (0.0\ \%)$; Pixels Cor(0): $3000000 = (100.0\ \%)$; Tempo Execução = 2 min.

Fonte: Do Autor.

Na Figura 5 observamos a bacia de uma equação polinomial de primeiro grau, que é a mais simples que se pode ter. Todos os pontos convergem para a única raiz da equação, assim a figura toda é pintada de uma única cor.

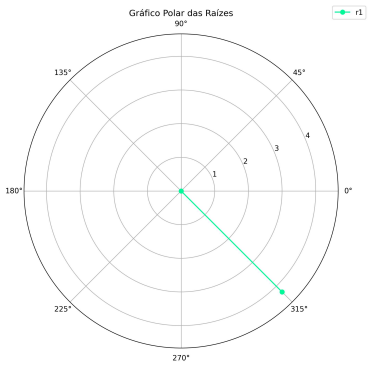

Figura 6 – Representação polar das raízes da equação: $(1+0j)z^1+(-3+3j)z^0=0$

Fonte: Do Autor.

Figura 7 – Bacias de Newton para a equação: $(1+0j)z^1+(1+0j)z^0=0$; R=[-1.1;1.1]x[-0.82j;0.82j]; Pixels não convergentes: $0 = (0.0\ \%)$; Pixels Cor(0): $3000000 = (100.0\ \%)$; Tempo Execução = 2 min.

Fonte: Do Autor.

Observamos na Figura 7 onde novamente temos uma equação polinomial do primeiro grau onde todos os pontos da figura convergem para a única raiz da equação. Assim como na Figura 5 não há pontos não convergentes.

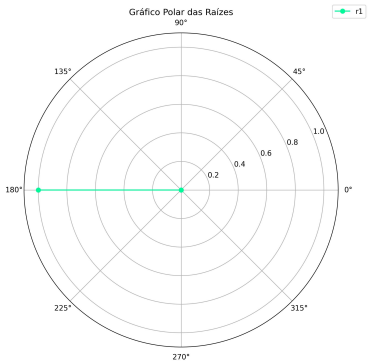

Figura 8 – Representação polar das raízes da equação: $(1+0j)z^1+(1+0j)z^0=0$

Fonte: Do Autor.

4.1.2 Equações Polinomiais de Grau 2

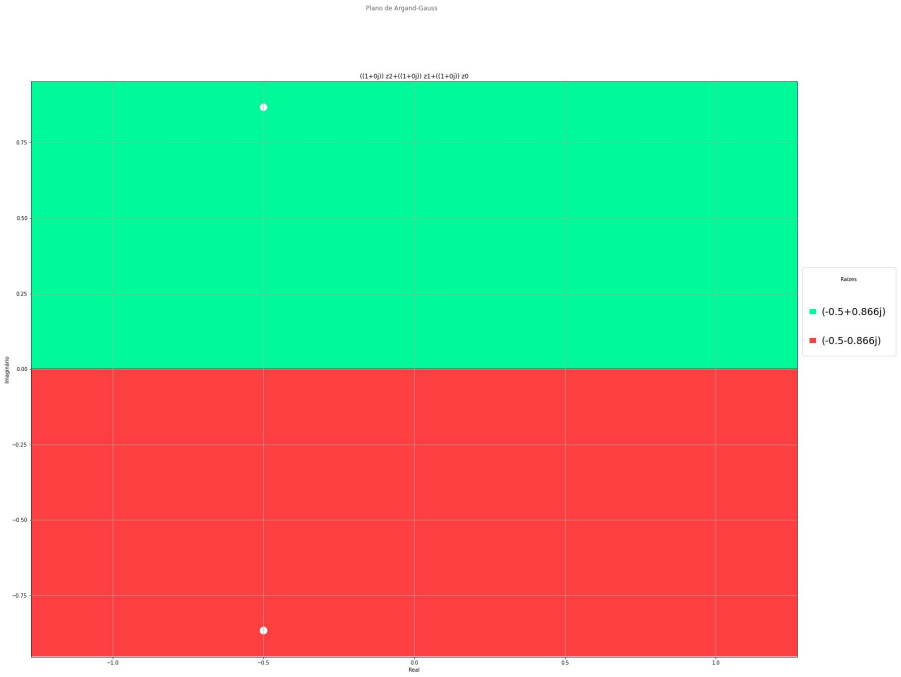

Figura 9 – Bacias de Newton para a equação: $(1+0j)z^2+(1+0j)z^1+(1+0j)z^0=0$; R=[-1.27;1.27]x[-0.95j;0.95j]; Pixels não convergentes: 2000 = (0.1 %); Pixels Cor(0): 1500000 = (50.0 %); Pixels Cor(1): 1498000 = (49.9 %); Tempo Execução = 11 min.

Fonte: Do Autor.

Podemos observar na Figura 9 duas bacias, duas regiões distintas para a equação polinomial de grau 2. Na fronteira entre as duas bacias temos uma linha com 2.000 pixels não convergentes. É possível notar esses pontos pretos na linha de fronteira.

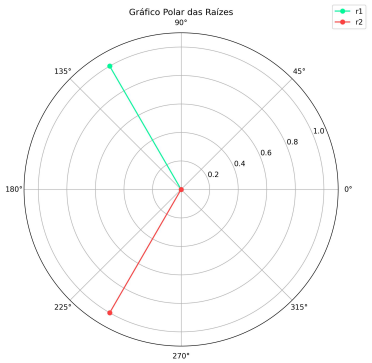

Figura 10 – Representação polar das raízes da equação: $(1+0j)z^2+(1+0j)z^1+(1+0j)z^0=0$

Fonte: Do Autor.

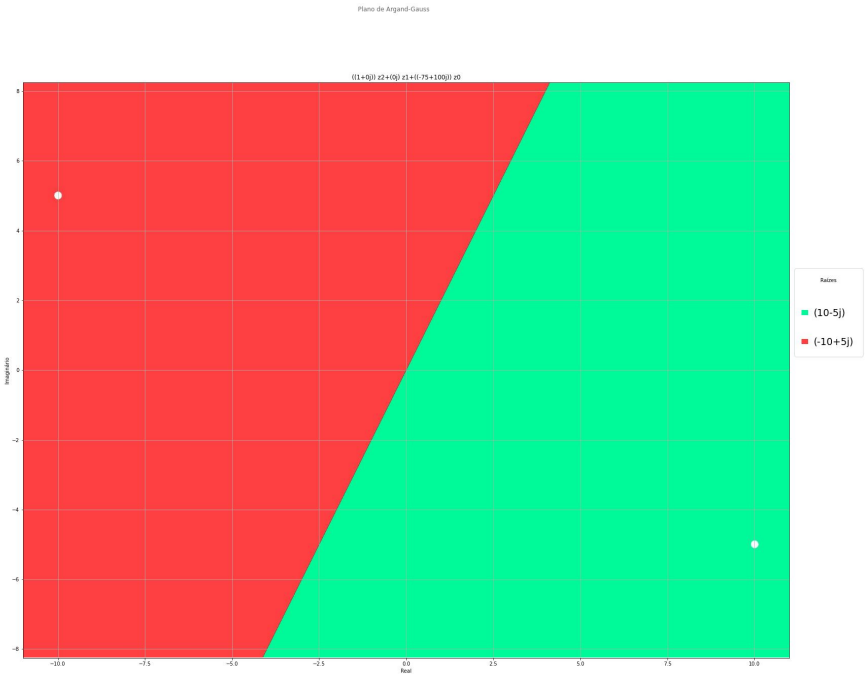

Figura 11 – Bacias de Newton para a equação: $(1+0j)z^2+0jz^1+(-75+100j)z^0=0$; R=[-11.0;11.0]x[-8.25j;8.25j]; Pixels não convergentes: 750 = (0.0 %); Pixels Cor(0): 1498500 = (50.0 %); Pixels Cor(1): 1500750 = (50.0 %); Tempo Execução = 10 min.

Fonte: Do Autor.

Na Figura 11 notamos que nas equações de grau 2 a linha de fronteira entre as duas bacias é a reta mediatriz do segmento que une os dois pontos que representam as duas raízes.

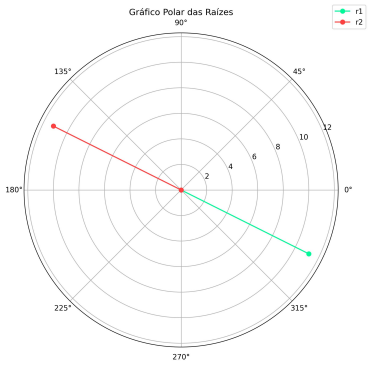

Figura 12 – Representação polar das raízes da equação: $(1+0j)z^2+0jz^1+(-75+100j)z^0=0$

Fonte: Do Autor.

4.1.3 Equações Polinomiais de Grau 3

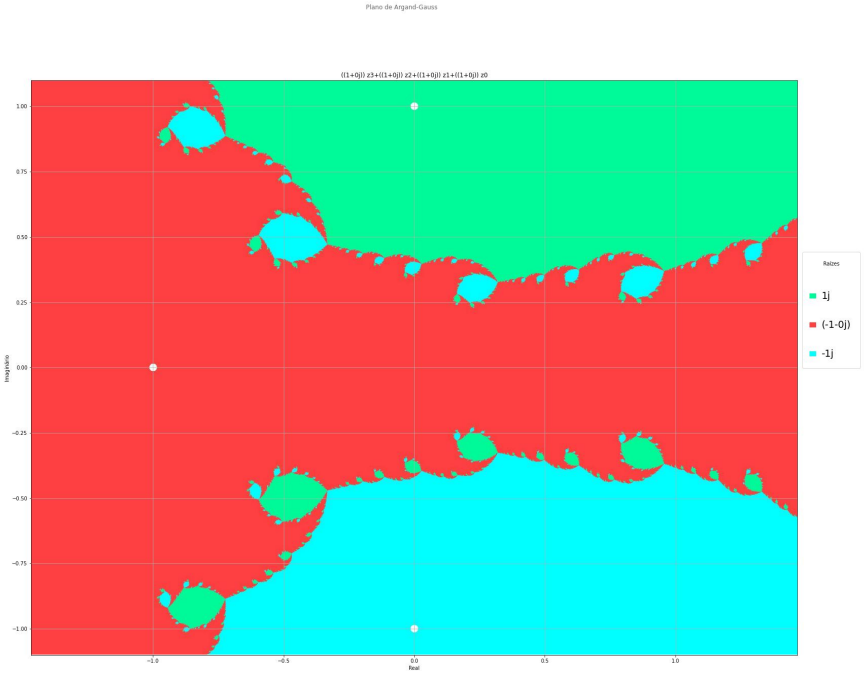

Figura 13 – Bacias de Newton para a equação: $(1+0j)z^3+(1+0j)z^2+(1+0j)z^1+(1+0j)z^0 = 0$; R=[-1.47;1.47]x[-1.1j;1.1j]; Pixels não convergentes: 20 = (0.0 %); Pixels Cor(0): 683324 = (22.8 %); Pixels Cor(1): 1634870 = (54.5 %); Pixels Cor(2): 681786 = (22.7 %); Tempo Execução = 14 min.

Fonte: Do Autor.

Notamos na Figura 13 que quando o grau da equação polinomial é 3 ainda existe uma simetria na forma da imagem e também que a distribuição dos pontos torna-se um tanto caótica. Não há uma linha bem definida na fronteira das bacias.

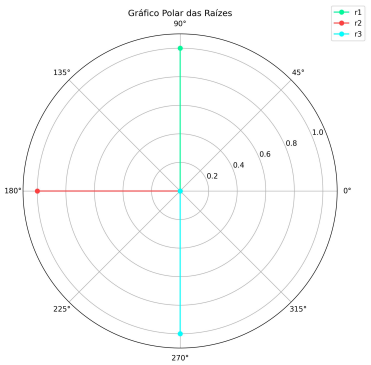

Figura 14 – Representação polar das raízes da equação: $(1+0j)z^3 + (1+0j)z^2 + (1+0j)z^1 + (1+0j)z^0 = 0$

Fonte: Do Autor.

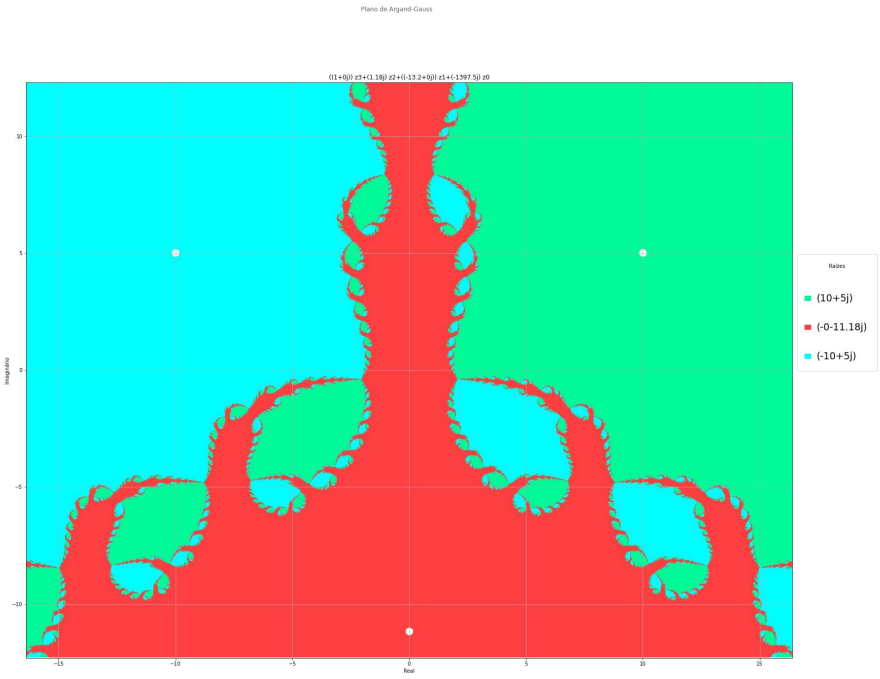

Figura 15 – Bacias de Newton para a equação: $(1+0j)z^3 + 1.18jz^2 + (-13.2+0j)z^1 + -1397.5jz^0 = 0$; R=[-16.4;16.4]x[-12.3j;12.3j]; Pixels não convergentes: 616 = (0.0 %); Pixels Cor(0): 1005940 = (33.5 %); Pixels Cor(1): 986470 = (32.9 %); Pixels Cor(2): 1006974 = (33.6 %); Tempo Execução = 15 min.

Fonte: Do Autor.

Uma característica da Figura 15 é que não conseguimos notar pontos de não convergência (pretos) na imagem, apesar de termos 616 deles que representam bem pouco comparado com o total de pontos. Há uma simetria em relação ao eixo imaginário do plano.

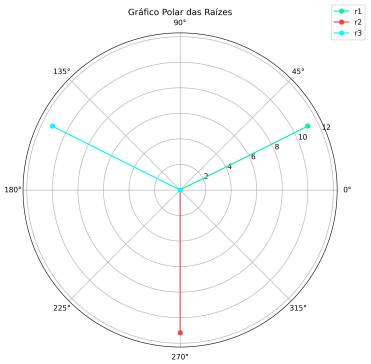

Figura 16 – Representação polar das raízes da equação: $(1+0j)z^3 + 1.18jz^2 + (-13.2+0j)z^1 + -1397.5jz^0 = 0$

Fonte: Do Autor.

4.1.4 Equações Polinomiais de Grau 4

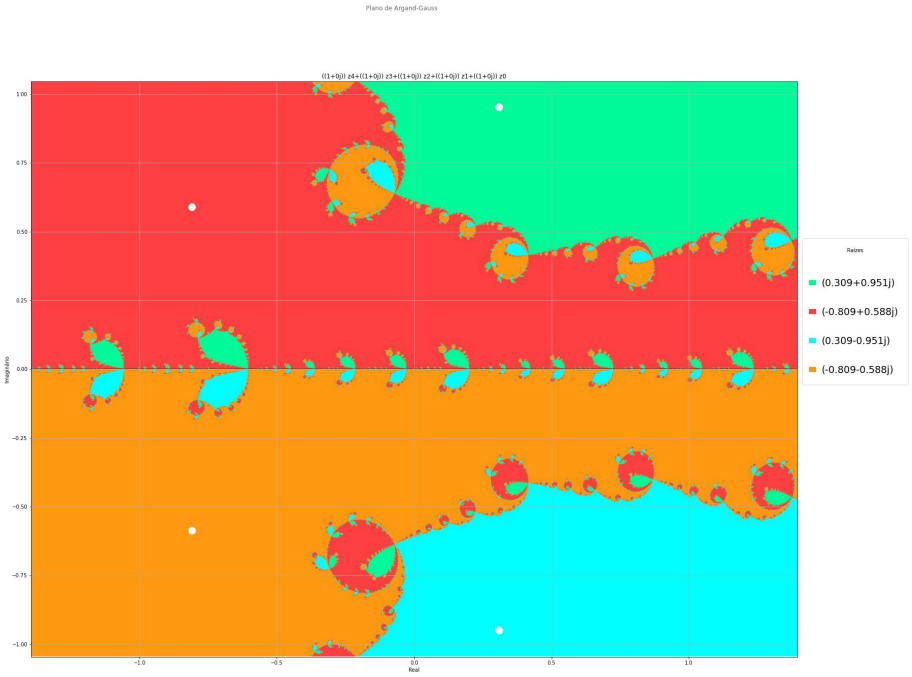

Figura 17 – Bacias de Newton para a equação: $(1+0j)z^4+(1+0j)z^3+(1+0j)z^2+(1+0j)z^1+(1+0j)z^0=0$; R=[-1.39;1.39]x[-1.05j;1.05j]; Pixels não convergentes: 2242 = (0.1 %); Pixels Cor(0): 455365 = (15.2 %); Pixels Cor(1): 1044401 = (34.8 %); Pixels Cor(2): 454219 = (15.1 %); Pixels Cor(3): 1043773 = (34.8 %); Tempo Execução = 16 min.

Fonte: Do Autor.

A Figura 17 que representa as bacias de uma equação polinomial de grau 4 também apresenta simetria com relação à forma, mas não às cores. Aqui também não é possível notar pontos não convergentes.

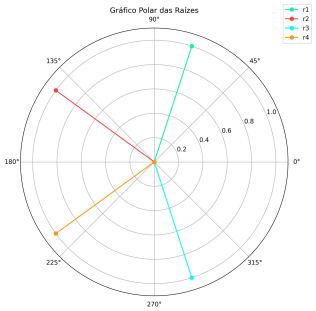

Figura 18 – Representação polar das raízes da equação: $(1+0j)z^4+(1+0j)z^3+(1+0j)z^2+(1+0j)z^1+(1+0j)z^0=0$

Fonte: Do Autor.

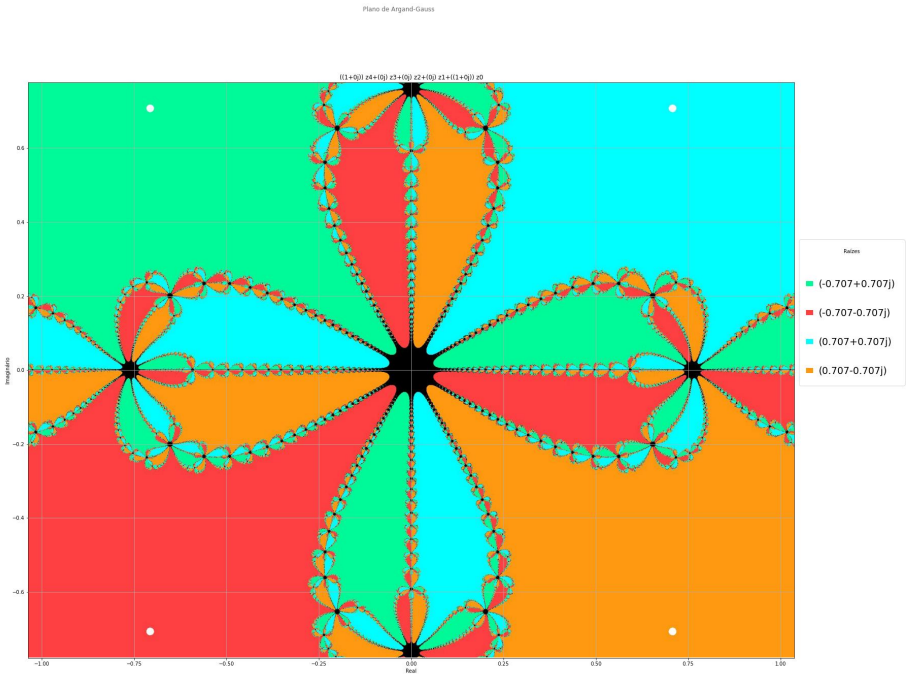

Figura 19 – Bacias de Newton para a equação: $(1+0j)z^4+0jz^3+0jz^2+0jz^1+(1+0j)z^0 = 0$; R=[-1.04;1.04]x[-0.78j;0.78j]; Pixels não convergentes: 64501 = (2.2 %); Pixels Cor(0): 734547 = (24.5 %); Pixels Cor(1): 733638 = (24.5 %); Pixels Cor(2): 734111 = (24.5 %); Pixels Cor(3): 733203 = (24.4 %); Tempo Execução = 21 min.

Fonte: Do Autor.

Para a equação de grau 4 da Figura 19 notamos simetria quanto à forma. Uma novidade é a concentração de pontos não convergentes em algumas regiões.

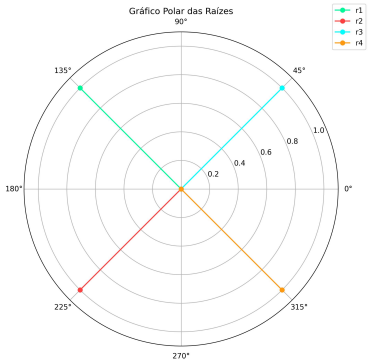

Figura 20 – Representação polar das raízes da equação: $(1+0j)z^4+0jz^3+0jz^2+0jz^1+(1+0j)z^0 = 0$

Fonte: Do Autor.

4.1.5 Equações Polinomiais de Grau 5

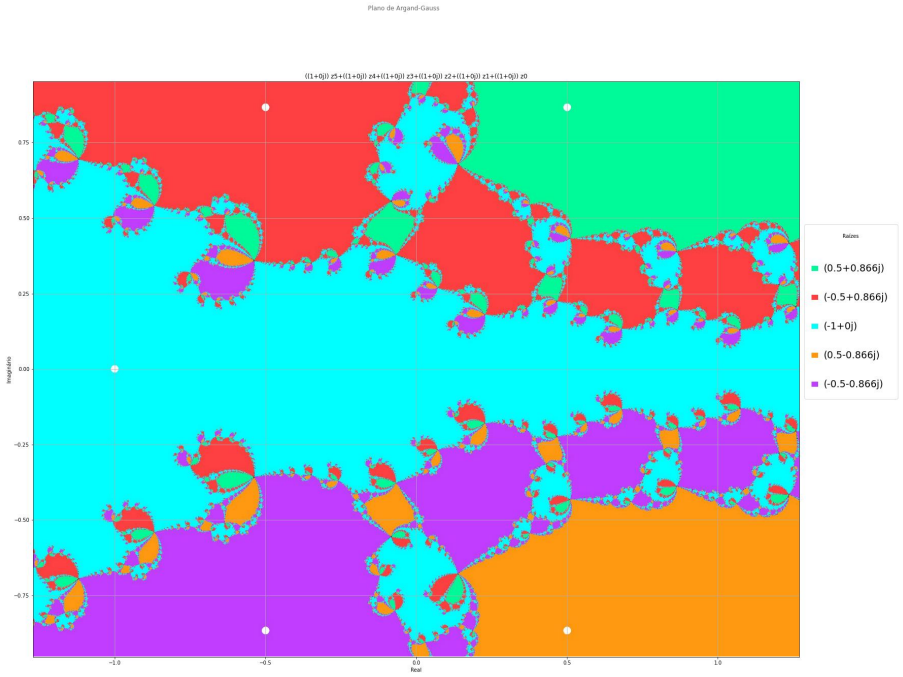

Figura 21 – Bacias de Newton para a equação: $(1+0j)z^5+(1+0j)z^4+(1+0j)z^3+(1+0j)z^2+(1+0j)z^1+(1+0j)z^0=0$; R=[-1.27;1.27]x[-0.95j;0.95j]; Pixels não convergentes: 3247 = (0.1 %); Pixels Cor(0): 381094 = (12.7 %); Pixels Cor(1): 544617 = (18.2 %); Pixels Cor(2): 1147307 = (38.2 %); Pixels Cor(3): 380204 = (12.7 %); Pixels Cor(4): 543531 = (18.1 %); Tempo Execução = 20 min.

Fonte: Do Autor.

Para a equação de grau 5 na Figura 21 notamos que a simetria quando à forma novamente se faz presente. Conforme o grau aumenta e com isso a quantidade de cores, a imagem se torna mais admirável.

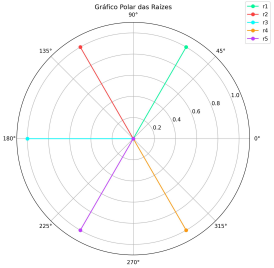

Figura 22 – Representação polar das raízes da equação: $(1+0j)z^5+(1+0j)z^4+(1+0j)z^3+(1+0j)z^2+(1+0j)z^1+(1+0j)z^0=0$

Fonte: Do Autor.

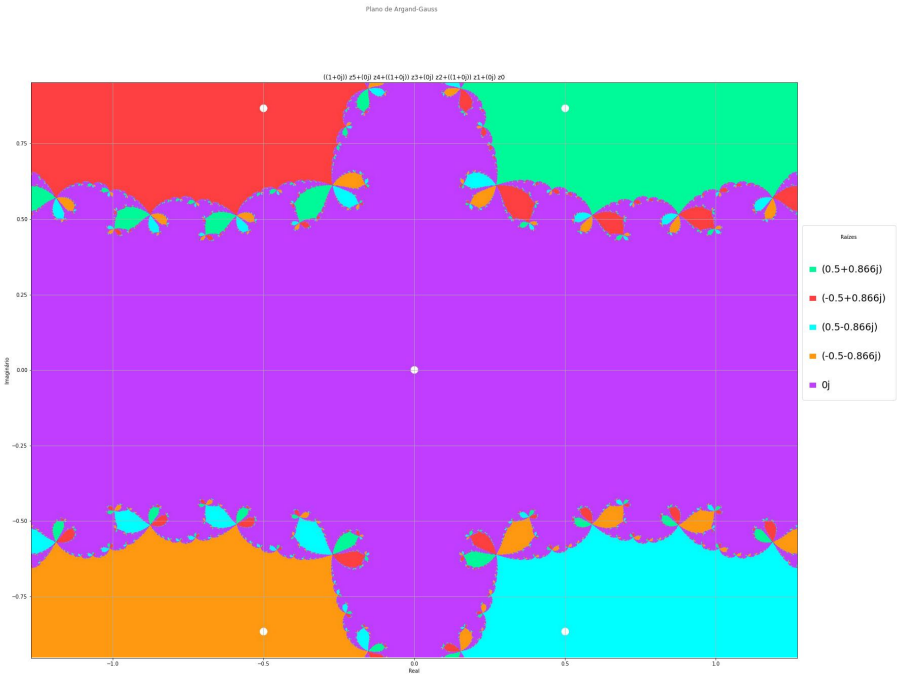

Figura 23 – Bacias de Newton para a equação: $(1+0j)z^5+0jz^4+(1+0j)z^3+0jz^2+(1+0j)z^1+0jz^0=0$; R=[-1.27;1.27]x[-0.95j;0.95j]; Pixels não convergentes: 114 = (0.0 %); Pixels Cor(0): 266514 = (8.9 %); Pixels Cor(1): 266695 = (8.9 %); Pixels Cor(2): 265677 = (8.9 %); Pixels Cor(3): 265857 = (8.9 %); Pixels Cor(4): 1935143 = (64.5 %); Tempo Execução = 15 min.

Fonte: Do Autor.

Podemos notar na Figura 23 a maior parte dos pontos (64.5%) convergindo para raiz 0+0j . Também há uma notável simetria na imagem.

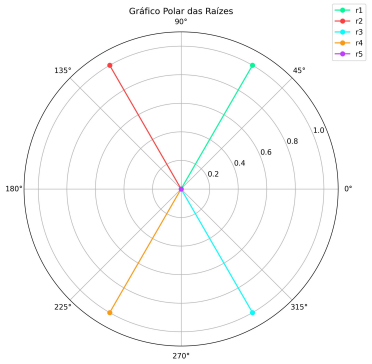

Figura 24 – Representação polar das raízes da equação: $(1+0j)z^5+0jz^4+(1+0j)z^3+0jz^2+(1+0j)z^1+0jz^0=0$

Fonte: Do Autor.

4.1.6 Equações Polinomiais de Grau 6

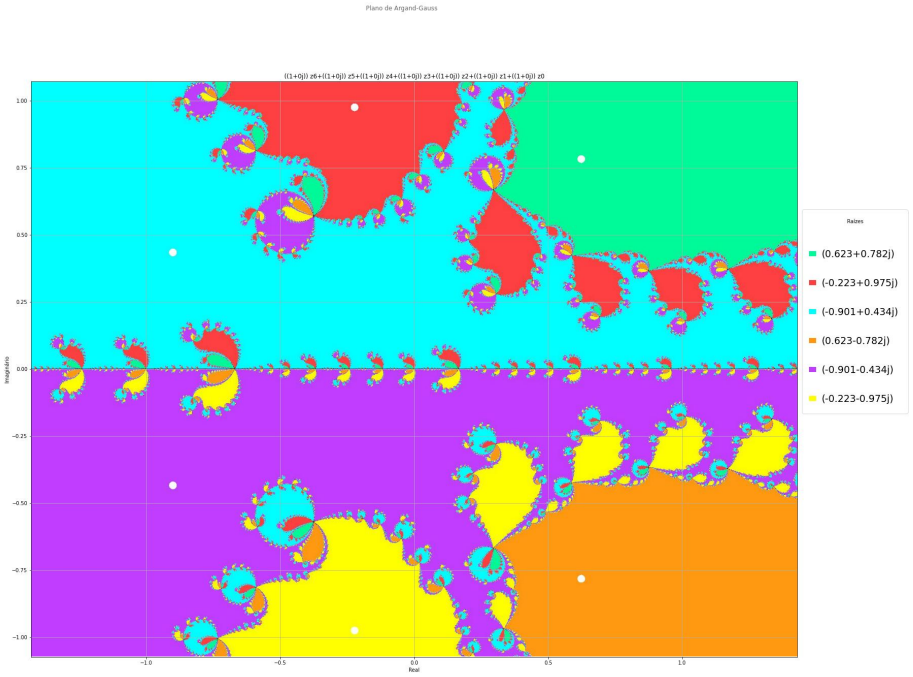

Figura 25 – Bacias de Newton para a equação: $(1+0j)z^6+(1+0j)z^5+(1+0j)z^4+(1+0j)z^3+(1+0j)z^2+(1+0j)z^1+(1+0j)z^0=0$; R=[-1.43;1.43]x[-1.07j;1.07j]; Pixels não convergentes: 6508 = (0.2 %); Pixels Cor(0): 346995 = (11.6 %); Pixels Cor(1): 257520 = (8.6 %); Pixels Cor(2): 893212 = (29.8 %); Pixels Cor(3): 346266 = (11.5 %); Pixels Cor(4): 892604 = (29.8 %); Pixels Cor(5): 256895 = (8.6 %); Tempo Execução = 22 min.

Fonte: Do Autor.

As bacias da Figura 25 representam uma equação de grau 6, nessas a última cor a entrar é a amarela. As regiões com essa cor destacam-se entre as outras.

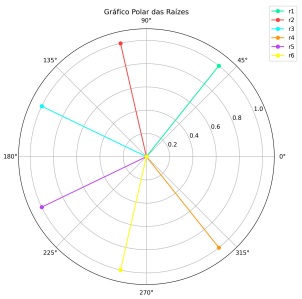

Figura 26 – Representação polar das raízes da equação: $(1+0j)z^6+(1+0j)z^5+(1+0j)z^4+(1+0j)z^3+(1+0j)z^2+(1+0j)z^1+(1+0j)z^0=0$

Fonte: Do Autor.

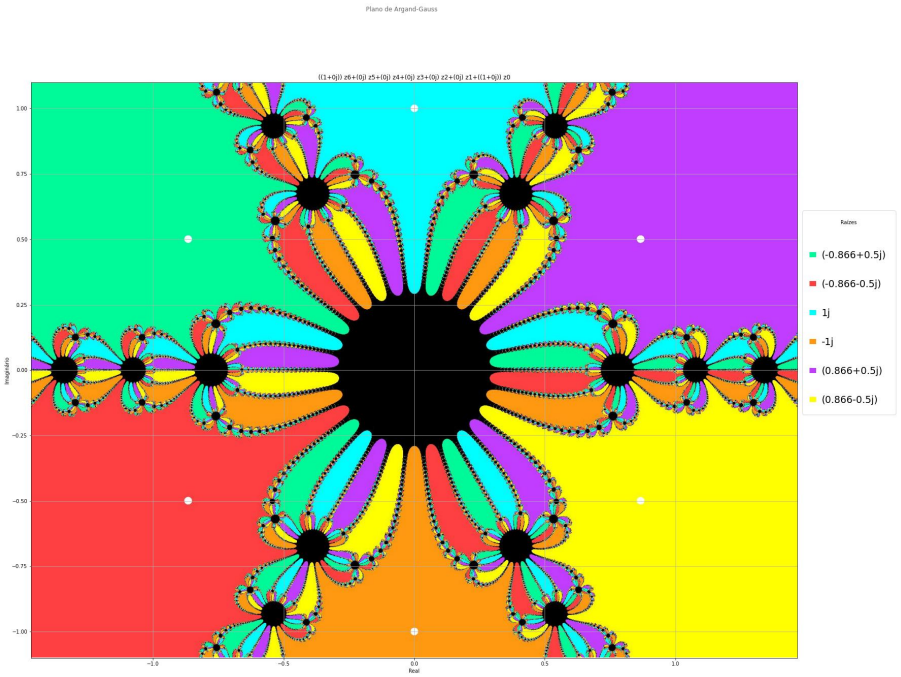

Figura 27 – Bacias de Newton para a equação: $(1+0j)z^6+0jz^5+0jz^4+0jz^3+0jz^2+0jz^1+(1+0j)z^0=0$; R=[-1.47;1.47]x[-1.1j;1.1j]; Pixels não convergentes: 326758 = (10.9 %); Pixels Cor(0): 524981 = (17.5 %); Pixels Cor(1): 524521 = (17.5 %); Pixels Cor(2): 288088 = (9.6 %); Pixels Cor(3): 287387 = (9.6 %); Pixels Cor(4): 524362 = (17.5 %); Pixels Cor(5): 523903 = (17.5 %); Tempo Execução = 30 min.

Fonte: Do Autor.

Na Figura 27 observamos que se a quantidade de pontos não convergentes e o grau do polinômio aumentam, o tempo de execução aumenta também. .

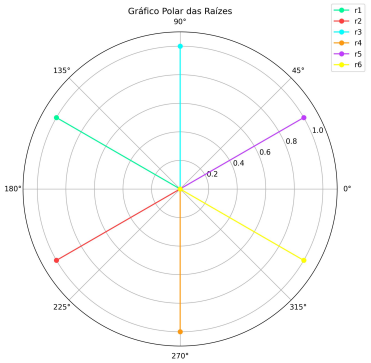

Figura 28 – Representação polar das raízes da equação: $(1+0j)z^6+0jz^5+0jz^4+0jz^3+0jz^2+0jz^1+(1+0j)z^0=0$

Fonte: Do Autor.

4.1.7 Equações Polinomiais de Grau 7

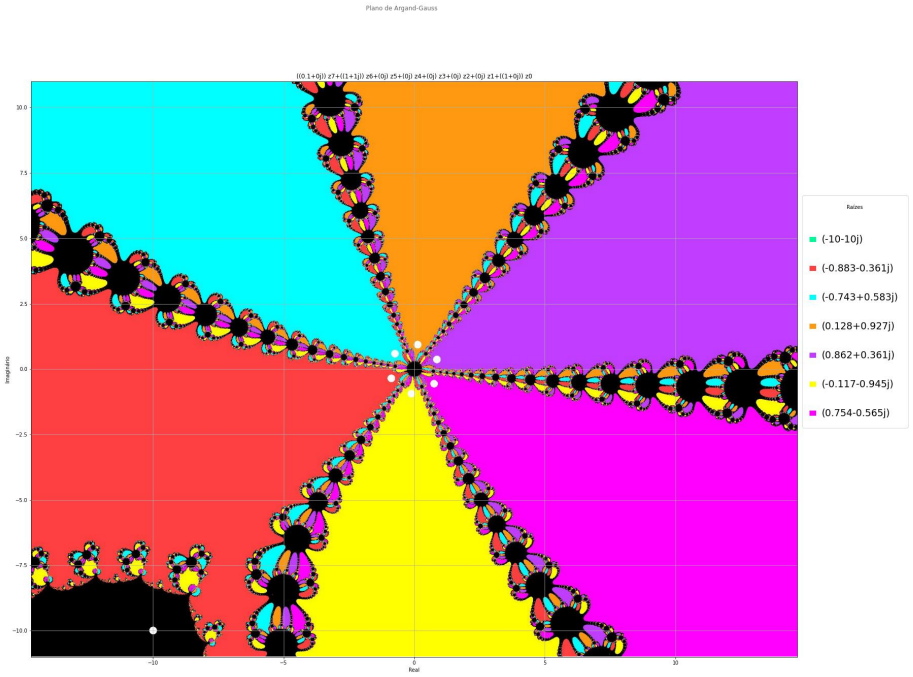

Figura 29 – Bacias de Newton para a equação: $(0.1+0j)z^7+(1+1j)z^6+0jz^5+0jz^4+0jz^3+0jz^2+0jz^1+(1+0j)z^0=0$; R=[-14.67;14.67]x[-11.0j;11.0j]; Pixels não convergentes: 347054 = (11.6 %); Pixels Cor(0): 0 = (0.0 %); Pixels Cor(1): 554136 = (18.5 %); Pixels Cor(2): 473088 = (15.8 %); Pixels Cor(3): 291006 = (9.7 %); Pixels Cor(4): 509060 = (17.0 %); Pixels Cor(5): 312631 = (10.4 %); Pixels Cor(6): 513025 = (17.1 %); Tempo Execução = 49 min.

Fonte: Do Autor.

Reforçamos na Figura 29 a relação entre um número alto de pontos não convergentes e um alto tempo de execução. A equação desta figura foi usada no exemplo da Figura 4.

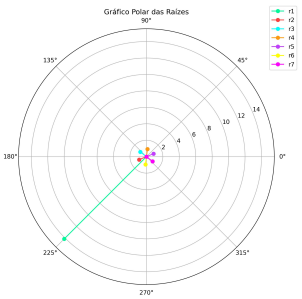

Figura 30 – Representação polar das raízes da equação: $(0.1+0j)z^7+(1+1j)z^6+0jz^5+0jz^4+0jz^3+0jz^2+0jz^1+(1+0j)z^0=0$

Fonte: Do Autor.

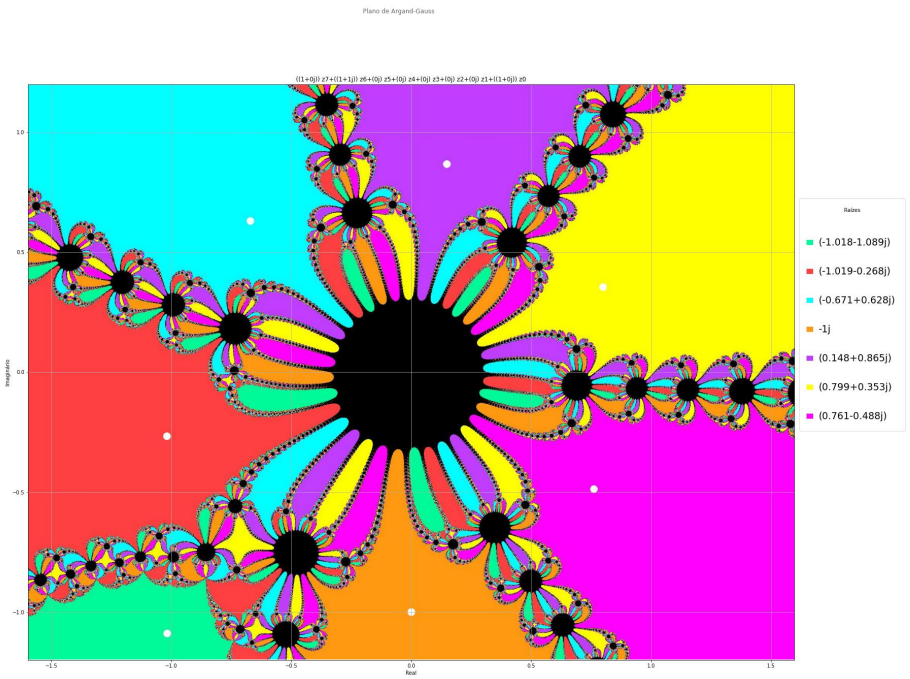

Figura 31 – Bacias de Newton para a equação: $(1+0j)z^7+(1+1j)z^6+0jz^5+0jz^4+0jz^3+0jz^2+0jz^1+(1+0j)z^0=0$; R=[-1.6;1.6]x[-1.2j;1.2j]; Pixels não convergentes: 357625 = (11.9 %); Pixels Cor(0): 184086 = (6.1 %); Pixels Cor(1): 425273 = (14.2 %); Pixels Cor(2): 466003 = (15.5 %); Pixels Cor(3): 254354 = (8.5 %); Pixels Cor(4): 292900 = (9.8 %); Pixels Cor(5): 517681 = (17.3 %); Pixels Cor(6): 502078 = (16.7 %); Tempo Execução = 33 min.

Fonte: Do Autor.

Notamos que a Figura 31 é caracterizada por uma grande quantidade de pontos não convergentes (11.9%), alto tempo de execução e nos lembra uma flor. Não há simetria por conta do terceiro quadrante.

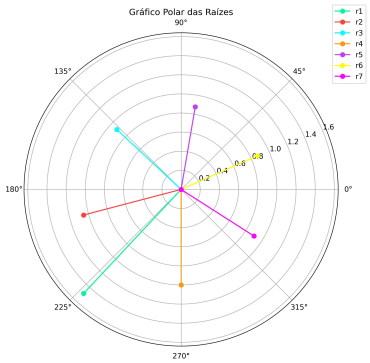

Figura 32 – Representação polar das raízes da equação: $(1+0j)z^7+(1+1j)z^6+0jz^5+0jz^4+0jz^3+0jz^2+0jz^1+(1+0j)z^0=0$

Fonte: Do Autor.

4.1.8 Equações Polinomiais de Grau 8

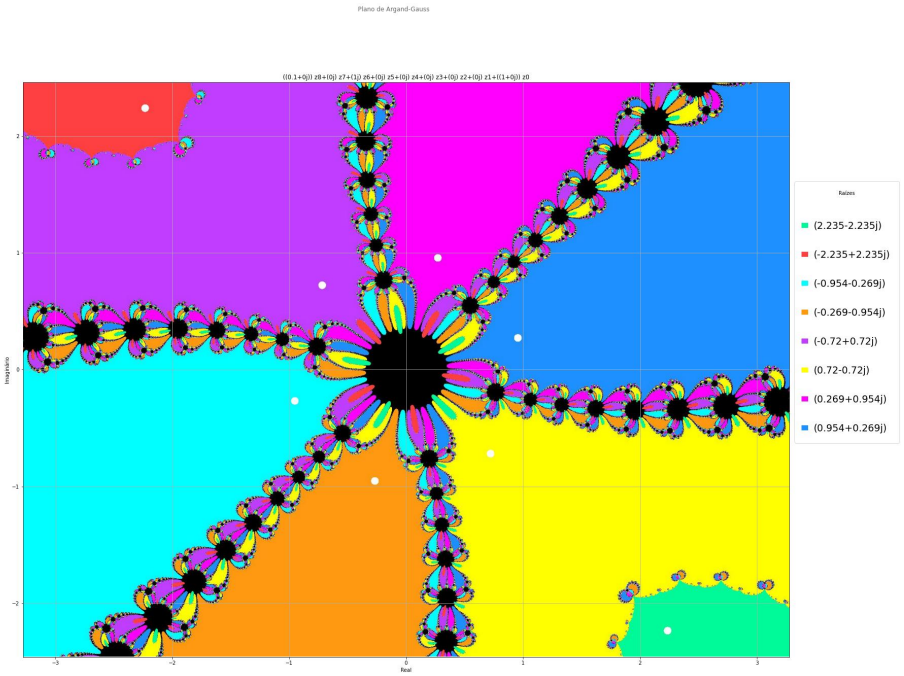

Figura 33 – Bacias de Newton para a equação: $(0.1 + 0j)z^8 + 0jz^7 + 1jz^6 + 0jz^5 + 0jz^4 + 0jz^3 + 0jz^2 + 0jz^1 + (1 + 0j)z^0 = 0$; R=[-3.28;3.28]x[-2.46j;2.46j]; Pixels não convergentes: 211640 = (7.1 %); Pixels Cor(0): 95874 = (3.2 %); Pixels Cor(1): 96443 = (3.2 %); Pixels Cor(2): 481264 = (16.0 %); Pixels Cor(3): 301337 = (10.0 %); Pixels Cor(4): 515875 = (17.2 %); Pixels Cor(5): 514993 = (17.2 %); Pixels Cor(6): 302006 = (10.1 %); Pixels Cor(7): 480568 = (16.0 %); Tempo Execução = 34 min.

Fonte: Do Autor.

Observamos que a Figura 33 representa as bacias de uma equação de grau 8. As imagens mais interessantes do trabalho são as que retratam equações polinomiais do grau 8. Um motivo é a presença de 8 cores, maior quantidade de cores entre todas.

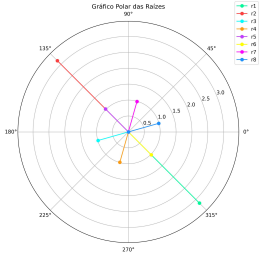

Figura 34 – Representação polar das raízes da equação: $(0.1 + 0j)z^8 + 0jz^7 + 1jz^6 + 0jz^5 + 0jz^4 + 0jz^3 + 0jz^2 + 0jz^1 + (1 + 0j)z^0 = 0$

Fonte: Do Autor.

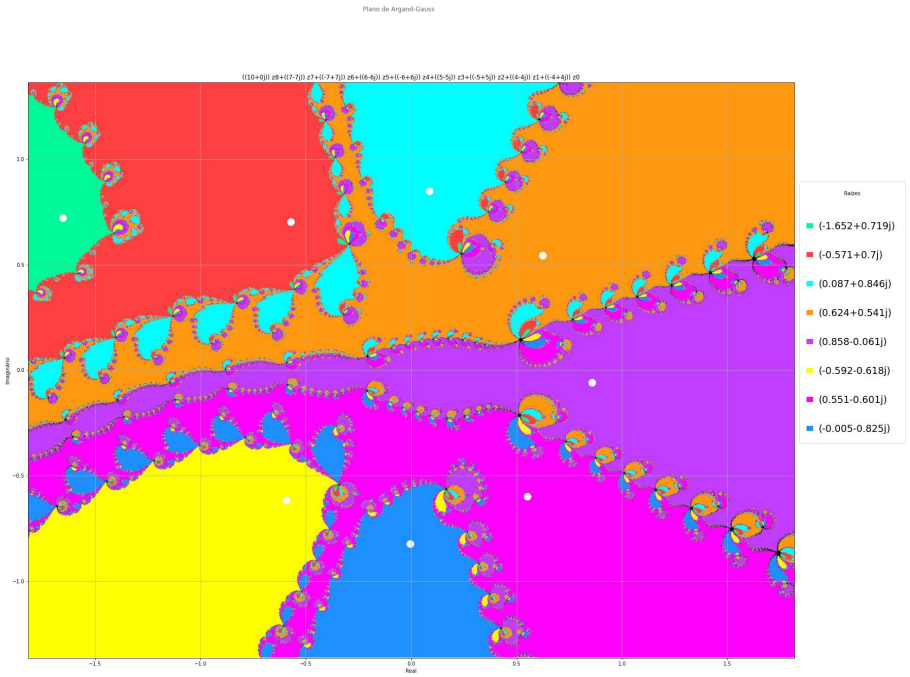

Figura 35 – Bacias de Newton para a equação: $(10+0j)z^8+(7-7j)z^7+(-7+7j)z^6+(6-6j)z^5+(-6+6j)z^4+(5-5j)z^3+(-5+5j)z^2+(4-4j)z^1+(-4+4j)z^0=0$; R=[-1.82;1.82]x[-1.36j;1.36j]; Pixels não convergentes: 22642 = (0.8 %); Pixels Cor(0): 75249 = (2.5 %); Pixels Cor(1): 399267 = (13.3 %); Pixels Cor(2): 241616 = (8.1 %); Pixels Cor(3): 630353 = (21.0 %); Pixels Cor(4): 498553 = (16.6 %); Pixels Cor(5): 358627 = (12.0 %); Pixels Cor(6): 558119 = (18.6 %); Pixels Cor(7): 215574 = (7.2 %); Tempo Execução = 30 min.

Fonte: Do Autor.

Notamos que a Figura 35 é uma das imagens mais belas pela distribuição peculiar de suas bacias e também por ter poucos pontos não convergentes.

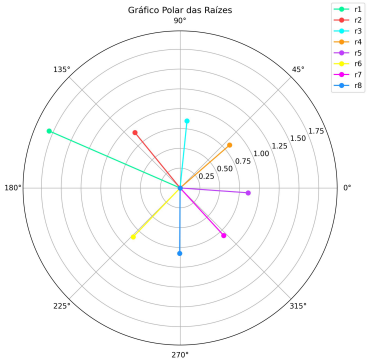

Figura 36 – Representação polar das raízes da equação: $(10+0j)z^8+(7-7j)z^7+(-7+7j)z^6+(6-6j)z^5+(-6+6j)z^4+(5-5j)z^3+(-5+5j)z^2+(4-4j)z^1+(-4+4j)z^0=0$

Fonte: Do Autor.

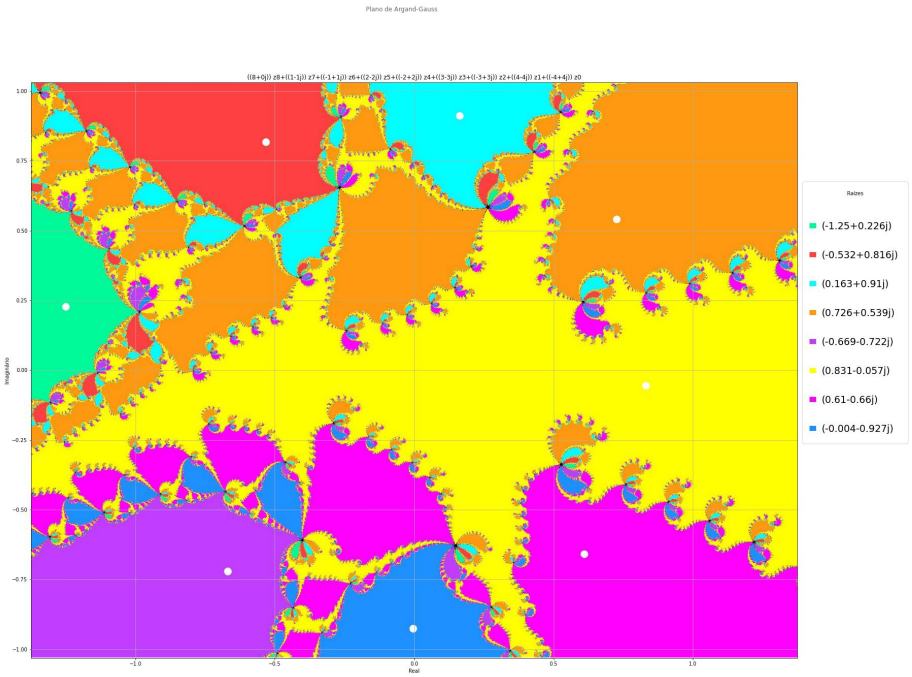

Figura 37 – Bacias de Newton para a equação: $(8+0j)z^8+(1-1j)z^7+(-1+1j)z^6+(2-2j)z^5+(-2+2j)z^4+(3-3j)z^3+(-3+3j)z^2+(4-4j)z^1+(-4+4j)z^0=0$; R=[-1.37;1.37]x[-1.03j;1.03j]; Pixels não convergentes: 12204 = (0.4 %); Pixels Cor(0): 95233 = (3.2 %); Pixels Cor(1): 199224 = (6.6 %); Pixels Cor(2): 171986 = (5.7 %); Pixels Cor(3): 626363 = (20.9 %); Pixels Cor(4): 252328 = (8.4 %); Pixels Cor(5): 1030112 = (34.3 %); Pixels Cor(6): 470125 = (15.7 %); Pixels Cor(7): 142425 = (4.7 %); Tempo Execução = 28 min.

Fonte: Do Autor.

A Figura 37 é a imagem mais bela de todo trabalho. Essa ramificação amarela no centro que se espalha pelas outras bacias acrescentou um aspecto artístico ímpar.

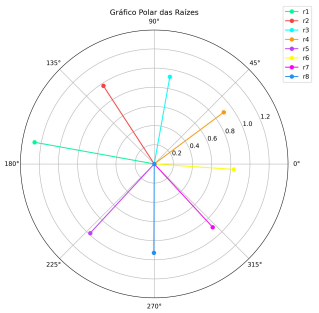

Figura 38 – Representação polar das raízes da equação: $(8+0j)z^8+(1-1j)z^7+(-1+1j)z^6+(2-2j)z^5+(-2+2j)z^4+(3-3j)z^3+(-3+3j)z^2+(4-4j)z^1+(-4+4j)z^0=0$

Fonte: Do Autor.

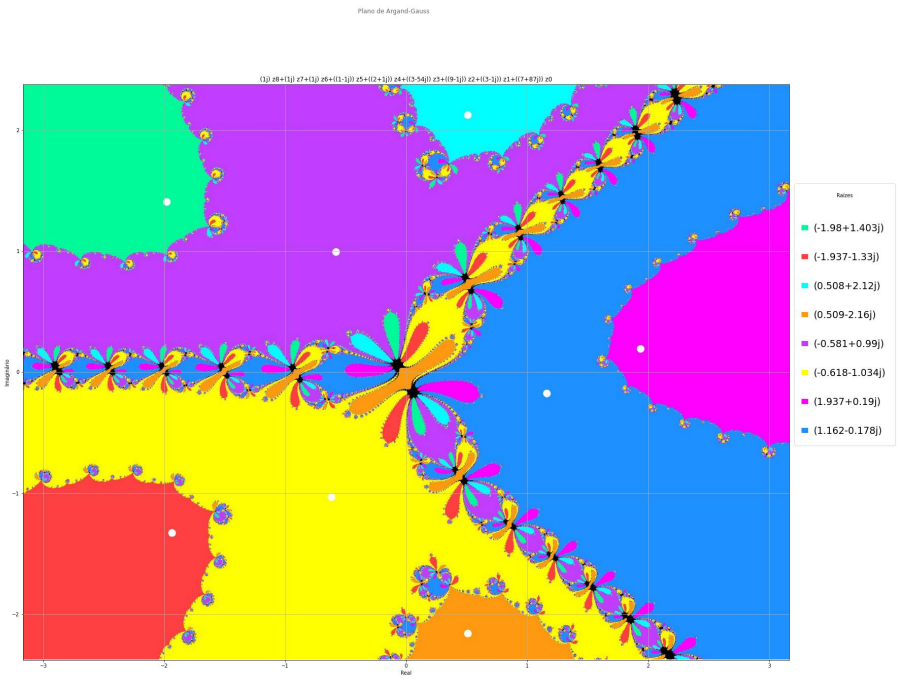

Figura 39 – Bacias de Newton para a equação: $1jz^8 + 1jz^7 + 1jz^6 + (1 - 1j)z^5 + (2 + 1j)z^4 + (3 - 54j)z^3 + (9 - 1j)z^2 + (3 - 1j)z^1 + (7 + 87j)z^0 = 0$; R=[-3.17;3.17]x[-2.38j;2.38j]; Pixels não convergentes: 24760 = (0.8 %); Pixels Cor(0): 220676 = (7.4 %); Pixels Cor(1): 240604 = (8.0 %); Pixels Cor(2): 86860 = (2.9 %); Pixels Cor(3): 97765 = (3.3 %); Pixels Cor(4): 664567 = (22.2 %); Pixels Cor(5): 649344 = (21.6 %); Pixels Cor(6): 244175 = (8.1 %); Pixels Cor(7): 771249 = (25.7 %); Tempo Execução = 26 min.

Fonte: Do Autor.

Na Figura 39 temos 8 cores muito pujantes distribuídas de forma única. Aqui é possível observar pequenas regiões onde se encontram pontos não convergentes.

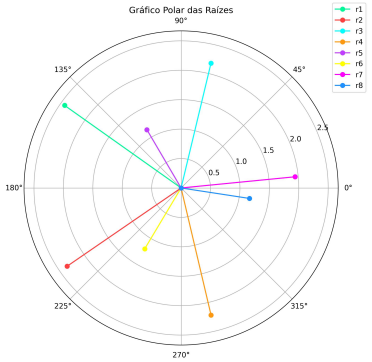

Figura 40 – Representação polar das raízes da equação: $1jz^8 + 1jz^7 + 1jz^6 + (1 - 1j)z^5 + (2 + 1j)z^4 + (3 - 54j)z^3 + (9 - 1j)z^2 + (3 - 1j)z^1 + (7 + 87j)z^0 = 0$

Fonte: Do Autor.

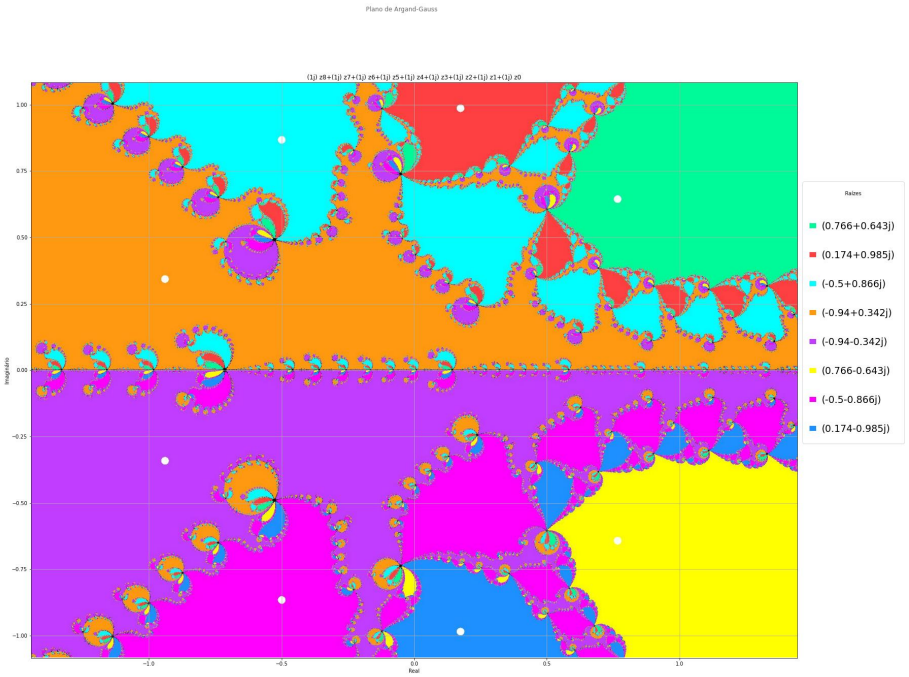

Figura 41 – Bacias de Newton para a equação: $1jz^8 + 1jz^7 + 1jz^6 + 1jz^5 + 1jz^4 + 1jz^3 + 1jz^2 + 1jz^1 + 1jz^0 = 0$; R=[-1.44;1.44]x[-1.08j;1.08j]; Pixels não convergentes: 21340 = (0.7 %); Pixels Cor(0): 291873 = (9.7 %); Pixels Cor(1): 139306 = (4.6 %); Pixels Cor(2): 349799 = (11.7 %); Pixels Cor(3): 709265 = (23.6 %); Pixels Cor(4): 709161 = (23.6 %); Pixels Cor(5): 291417 = (9.7 %); Pixels Cor(6): 349030 = (11.6 %); Pixels Cor(7): 138809 = (4.6 %); Tempo Execução = 28 min.

Fonte: Do Autor.

Na equação da Figura 41 todos os coeficientes do polinômio são 1j. Foi feito um teste em que todos os coeficientes são 1. A imagem gerada foi a mesma e as raízes também foram as mesmas.

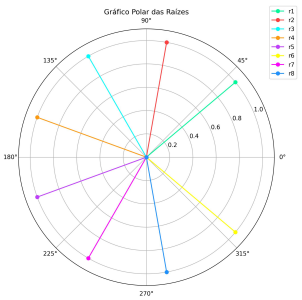

Figura 42 – Representação polar das raízes da equação: $1jz^8 + 1jz^7 + 1jz^6 + 1jz^5 + 1jz^4 + 1jz^3 + 1jz^2 + 1jz^1 + 1jz^0 = 0$

Fonte: Do Autor.

5 Conclusão

Exploramos o fascinante mundo das Bacias de Newton, um fenômeno matemático complexo e envolvente. Utilizando um programa desenvolvido em linguagem Python, pudemos visualizar e analisar as peculiares estruturas das Bacias de Newton geradas a partir da aplicação do Método de Newton em polinômios complexos.

Nossos esforços revelaram que as Bacias de Newton são verdadeiramente obras de arte matemáticas, exibindo uma riqueza de detalhes e complexidade que desafiam a intuição. Essas estruturas intricadamente entrelaçadas não apenas demonstram a beleza da matemática, mas também têm aplicações práticas em uma variedade de campos, desde a otimização até a física teórica.

Além disso, nosso trabalho nos permitiu perceber a relação entre o Método de Newton e os fractais, apesar de não nos aprofundarmos nisso. Essa conexão entre a análise numérica e a geometria fractal expande nossa compreensão das complexas interações entre matemática pura e matemática aplicada.

Concluímos que o estudo das Bacias de Newton, com a ajuda de ferramentas computacionais e programação em Python, não apenas enriquece nosso conhecimento matemático, mas também estimula o pensamento criativo e a exploração interdisciplinar. As imagens geradas pelo programa demonstram a beleza intrínseca e a complexidade subjacente à matemática, inspirando uma apreciação mais profunda das maravilhas do mundo matemático.

Como pesquisadores, acreditamos que o potencial das Bacias de Newton é vasto e que essa área de estudo oferece oportunidades emocionantes para pesquisas futuras. Ao continuar investigando suas propriedades e aplicabilidades, podemos desvendar novos mistérios matemáticos e ampliar nosso entendimento da matemática em seu estado mais puro.

Portanto, este trabalho representa não apenas uma exploração das Bacias de Newton, mas também um convite para que outros pesquisadores mergulhem nesse emocionante campo de estudo. À medida que continuamos a explorar as fronteiras da matemática e da computação, esperamos que as Bacias de Newton continuem a ser uma fonte inesgotável de descobertas e inspiração.

Referências

ARTIN, Michael. *Algebra*. Pearson Higher Ed, out. 2013. Citado 1 vez na página 24.

BOYER, Carl B; MERZBACH, Uta C. *História da matemática*. Editora Blucher, 2019. Citado 2 vezes nas páginas 23, 24.

BURDEN, Richard L. *Numerical analysis*. Brooks/Cole Cengage Learning, 2011. Citado 1 vez na página 28.

BURTON, Aaron. *Newton's method and fractals*. Citeseer, 2009. Citado 3 vezes nas páginas 16, 26, 31.

CIÊNCIA. *Ciência Programada - Ciência e programação ao seu alcance*. 2023. Disponível em: https://cienciaprogramada.com.br/. Acesso em: 20 nov. 2023. Citado 2 vezes nas páginas 17, 19.

DUKKIPATI, Rao V. *Numerical Methods*. 2010. Citado 1 vez na página 28.

DUKKIPATI, Rao V. *Numerical Methods Fundamentals*. Stylus Publishing, LLC, 2023. Citado 1 vez na página 28.

EDUCAPES. *eduCAPES*. 2024. https://educapes.capes.gov.br/. (Accessed on 05/12/2023). Citado 1 vez na página 21.

EDWARDS, C.H.Jr. *The Historical Development of the Calculus*. Springer Science & Business Media, dez. 2012. Citado 1 vez na página 24.

ELLENBERG, Jordan. *How not to be wrong: The power of mathematical thinking*. Penguin, 2015. Citado 1 vez na página 26.

EPPERSON, James F. *An introduction to numerical methods and analysis*. John Wiley & Sons, 2021. Citado 1 vez na página 28.

EPUREANU, Bogdan I; GREENSIDE, Henry S. Fractal basins of attraction associated with a damped Newton's method. *SIAM review*, JSTOR, p. 102–109, 1998. Citado 2 vezes nas páginas 16, 26.

EVES, Howard. Introdução à história da matemática, trad. *Higyno H. Domingues. Brasil: Editora UNICAMP*, 2011. Citado 1 vez na página 23.

GARLING, H. *A Course in Galois Theory*. Cambridge University Press, 1986. Citado 1 vez na página 24.

GLEICK, James. *Chaos: Making a new science*. Penguin, 2008. Citado 1 vez na página 25.

HUBBARD, John; SCHLEICHER, Dierk; SUTHERLAND, Scott. How to find all roots of complex polynomials by Newton's method. *Inventiones mathematicae*, Springer, v. 146, n. 1, p. 1–33, 2001. Citado 1 vez na página 26.

JANOS, Michel. Matemática e natureza. *São Paulo: Editora Livraria da Física*, 2009. Citado 1 vez na página 26.

LEVY, Adam B. *Attraction in Numerical Minimization: Iteration Mappings, Attractors, and Basins of Attraction*. Springer, 2018. Citado 1 vez na página 26.

MATPLOTLIB. *Matplotlib documentation — Matplotlib 3.5.0 documentation*. 2023. Disponível em: https://matplotlib.org/stable/. Acesso em: 20 nov. 2023. Citado 2 vezes nas páginas 17, 19.

MATTHES, Eric. *Curso Intensivo de Python: Uma introdução prática e baseada em projetos à programação*. Novatec Editora, 2016. Citado 2 vezes nas páginas 19, 21.

MCCLURE, Mark. Newton's method for complex polynomials. *Mathematica in Education and Research*, v. 11, n. 2, 2006. Citado 1 vez na página 26.

PEITGEN, Heinz-Otto; RICHTER, Peter H. *The beauty of fractals: images of complex dynamical systems*. Springer Science & Business Media, 1986. Citado 1 vez na página 25.

PILLOW. *Pillow — Pillow (PIL Fork) 6.2.1 documentation*. 2011. Disponível em: https://pillow.readthedocs.io/en/stable/. Acesso em: 20 nov. 2023. Citado 1 vez na página 19.

PRESS, William H. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007. Citado 1 vez na página 26.

PYTHON. *Python*. Python.org, mai. 2019. Disponível em: https://www.python.org/. Acesso em: 20 nov. 2023. Citado 1 vez na página 18.

ROQUE, Tatiana. *História da matemática*. Editora Schwarcz - Companhia das Letras, set. 2012. Citado 1 vez na página 23.

SAHARI, ML; DJELLIT, I et al. Fractal newton basins. *Discrete Dynamics in Nature and Society*, Hindawi, v. 2006, 2006. Citado 1 vez na página 16.

STEWART, Ian. *Galois theory*. Crc Press, Taylor & Francis Group, 2015. Citado 1 vez na página 24.

STROGATZ, Steven H. Nonlinear dynamics and chaos, 1996. Citado 1 vez na página 26.

SWEIGART, Al. Automatize tarefas maçantes com Python. *São Paulo: Novatec*, 2015. Citado 2 vezes nas páginas 19, 21.

TEAM, Spyder. *Spyder Website*. 2018. Disponível em: https://www.spyder-ide.org/. Acesso em: 20 nov. 2023. Citado 1 vez na página 18.

YAU, Lily; BEN-ISRAEL, Adi. The Newton and Halley methods for complex roots. *The American Mathematical Monthly*, Taylor & Francis, v. 105, n. 9, p. 806–818, 1998. Citado 1 vez na página 26.

Apêndices

APÊNDICE A – Código fonte do programa

```python
# -*- coding: utf-8 -*-
"""
Created on Mon Mar 20 18:30:49 2023

@author: David Jesus dos Reis Silveira
"""

import matplotlib
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image




def rgb_to_hex(rgb):
    return '#%02x%02x%02x' % rgb


#recebe uma lista de complexos e devolve uma outra com valores arredondados
def arredondaraizes(raizes,decimais):
    lista = []
    tam = len(raizes)
    for i in range(tam):
        lista.append(complex(round(raizes[i].real,decimais),round(raizes[i].imag,decimais)))
    return lista

#rotina de entrada dos coeficientes do polinomio numa lista
def lepol():
    print('Entre com a ordem do polinômio menor ou igual a 8:')
    ordem = int(input())
    lista=[]
    tam = ordem + 1
    for i in range(tam):
        lista.append(complex(0.0,0.0))
    for i in range(tam):
        print("Entre com o coeficiente complexo de x ^",ordem - i,"na forma a+bj"+":")
        lista[i] = complex(input())
    return lista


def crianome(polin):
    lista = ""
    tam = len(polin)
    j = tam
    for i in range(tam):
        #Aqui alteramos os caracteres do nome acima da imagem com grid
        lista = lista+"("+str(polin[i])+")"+" z"+str(j-1)
        if j>1:
            lista = lista + "+"
        j = j-1
    return lista


def mostrapol(polin):
    tam = len(polin)
    j = tam
    for i in range(tam):
        print(polin[i],'* x ^',j-1,'+')
        j = j-1
    print("Fim mostrapol")

```

```
 63 def deriva(polin):
 64     lista = polin[:]
 65     tam = len(polin)-1
 66     for i in range(tam):
 67         lista[i]=(tam-i)*polin[i]
 68     lista.pop()#tira ultimo elemento da lista
 69     #print("Fim deriva")
 70     return lista
 71 
 72 
 73 def newton(f, df, x0, tol, maxi):
 74 
 75     x = x0
 76 
 77     for i in range(maxi):
 78         fx = np.polyval(f,x)
 79         if abs(fx) < tol:
 80             return x
 81         dfx = np.polyval(df,x)
 82         if dfx == 0+0j:
 83             return x0
 84             #raise ValueError("Derivada zero. Não é possível continuar a
    iteração.")
 85         x = x - np.polyval(f,x)/np.polyval(df,x)
 86     return x0
 87     #raise ValueError("O método de Newton não convergiu.")
 88 
 89 
 90 def criaimagem(nome,formato,largura,altura,pol,dpol,raizesarredondadas,fator):
 91     nomearq = (nome+".jpg",formato)
 92     pillow_obj = Image.new("RGB", (largura,altura))
 93 
 94     pixel_set = pillow_obj.load()
 95     color = (0,0,0)
 96     branco = (255,255,255)
 97     contbranco = 0
 98     tol = 1e-10
 99     max_iter = 30
100     tam = len(raizesarredondadas)
101     contcores=[]
102 
103     for i in range(tam):
104         contcores.append(0)
105 
106     naoconvergiu = 0
107 
108     #cores=[(118,238,0),(255,64,64),(152,245,255),(255,153,18),(191,62,255),
    (255,215,0),(255,0,255),(64,224,208)]
109     cores=[(0,250,154),(255,64,64),(0,255,255),(255,153,18),(191,62,255),
    (255,255,0),(255,0,255),(30,144,255)]
110 
111 
112     for a in range(altura):
113         for l in range (largura):
114 
115 
116 
117             x=(l-largura//2)*fator
118             y=(altura//2-a)*fator
119             #se fator=1 cada inteiro vale um pixel
120             x0=complex(x,y)
121 
122             raiz = newton(pol, dpol, x0, tol, max_iter)
123             raiz1 = complex(round(raiz.real,3),round(raiz.imag,3))
```

```
124
125             try:
126                 indice = raizesarredondadas.index(raiz1)
127             except:
128                 indice = -1
129
130             if indice == -1:
131                 naoconvergiu = naoconvergiu + 1
132                 pixel_set[l,a] = color
133
134
135             for i in range(tam):
136                 if indice == i:
137                     contcores[i]=contcores[i]+1
138                     pixel_set[l,a]=cores[indice]
139                 if(abs(x0-raizesarredondadas[i])<fator*10):
140                     contbranco = contbranco + 1
141                     pixel_set[l,a] = branco
142
143                 print("Total = ",(a+1)*100/altura,"%")
144
145
146     #Salva a imagem no disco
147     pillow_obj.save(*nomearq)
148
149     #inicio da colocação dos eixos na imagem
150     im = plt.imread(*nomearq)
151     fig , ax = plt.subplots()
152
153     inicio_x = -(largura//2)*fator
154     final_x = (largura//2)*fator
155
156     inicio_y = -(altura//2)*fator
157     final_y = (altura//2)*fator
158
159     ax.grid('on')
160
161     # dpi = 96 ou 100 também encontrados
162     dpi = 72
163     fig.set_size_inches(largura/dpi,altura/dpi)
164     fig.suptitle('Plano de Argand-Gauss', color='dimgray')
165     nomestr = crianome(pol)
166     ax.set_title(nomestr)
167     ax.imshow(im, extent=[inicio_x,final_x,inicio_y,final_y],aspect=1)
168     ax.set_xlabel('Real')
169     ax.set_ylabel('Imaginário')
170
171
172
173     for i in range(tam):
174         texto = str(raizesarredondadas[i])
175         cor = cores[i]
176         cor = rgb_to_hex(cor)
177         line1, = ax.plot([0], label=texto, linewidth = 10, color = cor )
178         #line1, = ax.plot([0], label=texto, linewidth = 10, color = rgb_to_hex
    (118,238,0) )
179
180
181
182     ax.legend(
183
184               ncol=1,
185               title="Raízes\n",
186               loc="center left",
```

```
187
188             #largura da legenda
189             handlelength=0.2,
190
191             #distância da borda da legenda até o texto dos 4 lados
192             borderpad=1,
193
194             #espaço entre as labels na vertical
195             labelspacing=2.0,
196
197             #amplia e reduz a legenda toda proporcionalmente
198             prop={'size':19},
199
200             bbox_to_anchor=(1.0 , 0.1 , 0.5 , 1)
201
202
203            )
204
205
206
207 #aqui a imagem carregada entra nos eixos que foram criados por fig , ax =
    plt.subplots()
208     ax.imshow(im, extent=[inicio_x,final_x,inicio_y,final_y],aspect=1)
209
210
211
212     #Salva a imagem com eixos no disco
213     plt.savefig(nome+'~eixos.jpg')
214     #fim da colocação dos eixos na imagem
215
216
217
218     #INÍCIO DO GRÁFICO POLAR
219
220     # Números complexos (exemplo)
221     complex_numbers = raizesarredondadas
222
223     # Cores RGB correspondentes (exemplo)
224     colors_rgb = cores
225
226     # Criação do gráfico polar
227     plt.figure(figsize=(8, 8))
228     ax = plt.subplot(111, projection='polar')
229
230     # Plotagem dos números complexos com cores RGB
231     for i, z in enumerate(complex_numbers):
232         angulo = np.angle(z)  # Ângulo do número complexo
233         r = np.abs(z)  # Módulo do número complexo
234         color = tuple(np.array(colors_rgb[i]) / 255.0)  # Normaliza os valores RGB
    para 0-1
235         plt.polar([0,angulo],[0,r],marker='o',color=color,label=f'r{i+1}')
236         #ax.scatter(angle, r, c=[color], s=100, label=f'r{i+1}')
237
238     # Configurações estilísticas
239     ax.set_rmax(max([np.abs(z) for z in complex_numbers])*1.1)  # Define o valor
    máximo do raio
240
241     # Adiciona uma legenda
242     ax.legend(loc='upper right', bbox_to_anchor=(1.1, 1.1))
243
244     # Exibe o gráfico
245     plt.title('Gráfico Polar das Raízes')
246
247     # Salva o gráfico no disco
```

```
248     plt.savefig(nome+"~Polar"+".jpg", bbox_inches='tight', dpi=300)
249 
250     # Mostra o gráfico
251     #plt.show()
252 
253 
254     #FIM DO GRÁFICO POLAR
255 
256     #inicio da criação do arquivo com informações técnicas da imagem
257     caminho_arquivo = nomecurto + '.txt'
258     arquivo = open(caminho_arquivo, "w")
259 
260     print('\n' * 10)
261     totaldepixels = altura * largura
262     print(nome + "\n\n")
263     print("Total de pixels = " + str(totaldepixels) + "\n")
264     print("Pixels não convergentes = " + str(naoconvergiu) + " = " + str
    ((naoconvergiu/totaldepixels)*100) + " % " + "\n")
265     for i in range(tam):
266         print("Quantidade de pixels cor " + str(i) + " = " + str(contcores[i]) + "
    = " + str(contcores[i]/totaldepixels*100) + "%" + "\n" )
267     print("Abscissa mínima = " + str(inicio_x) + "\n")
268     print("Abscissa máxima = " + str(final_x) + "\n")
269     print("Ordenada mínima = "+ str(inicio_y) + "\n")
270     print("Ordenada máxima = "+ str(final_y) + "\n")
271     #print("Fator de escala = " + str(fator))
272 
273     #arquivo.write("Figura " + str(indfigura) + "\n\n")
274     arquivo.write('Polinômio = ' + nome + '\n')
275     arquivo.write('Raízes = ' + str(raizesarredondadas))
276     #arquivo.write(nome + "     Figura " + str(indfigura) + "\n\n")
277     arquivo.write("\nLargura da imagem = " + str(largura) + ' pixels' + "\n")
278     arquivo.write("Altura da imagem = " + str(altura) + ' pixels' + "\n")
279     arquivo.write("Total de pixels = " + str(totaldepixels) + "\n")
280     arquivo.write("Quantidade de pixels não convergentes = " + str(naoconvergiu) +
    " = " + str((naoconvergiu/totaldepixels)*100) + " % " + "\n")
281     for i in range(tam):
282         arquivo.write("Quantidade de pixels cor(" + str(i) + ") = " + str(contcores
    [i]) + " = " + str(contcores[i]/totaldepixels*100) + " % " )
283         arquivo.write("\n")
284 
285     arquivo.write("Abscissa mínima = " + str(inicio_x))
286     arquivo.write("\n")
287     arquivo.write("Abscissa máxima = " + str(final_x))
288     arquivo.write("\n")
289     arquivo.write("Ordenada mínima = "+ str(inicio_y))
290     arquivo.write("\n")
291     arquivo.write("Ordenada máxima = "+ str(final_y))
292     arquivo.write("\n")
293     #arquivo.write("Fator de escala = " + str(fator))
294     arquivo.write("\n\n\n\n")
295 
296     arquivo.close()
297     #fim da criação do arquivo com informações técnicas da imagem
298 
299     #fim da criaimagem
300 
301 
302 
303 
304 #recebe uma lista de complexos e devolve o mínimo real
305 def minreal(listcomp):
306     lista = []
307     tam = len(listcomp)
```

```
308     for i in range(tam):
309         lista.append(listcomp[i].real)
310     minreal = min(lista)
311     return minreal
312 
313 
314 #recebe uma lista de complexos e devolve o máximo real
315 def maxreal(listcomp):
316     lista = []
317     tam = len(listcomp)
318     for i in range(tam):
319         lista.append(listcomp[i].real)
320     maxreal = max(lista)
321     return maxreal
322 
323 #recebe uma lista de complexos e devolve o mínimo imaginário
324 def minimag(listcomp):
325     lista = []
326     tam = len(listcomp)
327     for i in range(tam):
328         lista.append(listcomp[i].imag)
329     minimag = min(lista)
330     return minimag
331 
332 #recebe uma lista de complexos e devolve o máximo imaginário
333 def maximag(listcomp):
334     lista = []
335     tam = len(listcomp)
336     for i in range(tam):
337         lista.append(listcomp[i].imag)
338     maximag = max(lista)
339     return maximag
340 
341 
342 
343 print('\n' * 10)
344 
345 pol = lepol()
346 #pol = [1,0,-21-4j,0,20-48j,0,492-3344j]
347 #pol = [1,-3+3j]
348 #pol = [1j,1j,1j,1j,1j,1j,1j,1j,1j]
349 
350 
351 largura = 200
352 altura = 150
353 
354 
355 dpol = deriva(pol)
356 raizes = np.roots(pol)
357 raizesarredondadas = arredondaraizes(raizes,3)
358 
359 grau = len(raizesarredondadas)
360 
361 
362 minx = minreal(raizes)
363 maxx = maxreal(raizes)
364 miny = minimag(raizes)
365 maxy = maximag(raizes)
366 maximo = max(abs(minx),abs(maxx),abs(miny),abs(maxy))
367 maximodx = max(abs(minx),abs(maxx))
368 maximody = max(abs(miny),abs(maxy))
369 
370 if altura > largura:
371     fator = maximo/(altura//2)
```

```
372     if maximody < maximodx:
373         fator = maximo/(largura//2)
374     if maximodx > (largura//2)*fator:
375         fator = (maximodx/((largura//2)*fator))*fator
376 
377 if altura <= largura:
378     fator = maximo/(largura//2)
379     if maximody >= maximodx:
380         fator = maximo/(altura//2)
381     if maximody > (altura//2)*fator:
382         fator = (maximody/((altura//2)*fator))*fator
383 
384 
385 
386 fator = 1.1*fator
387 
388 
389 fatorp = fator*100
390 raix = str(arredondaraizes(raizes,2))
391 larguras = str(largura)
392 alturas = str(altura)
393 fatorp = round(fatorp,3)
394 fators = str(fatorp)
395 nome=",".join(str(x) for x in pol)
396 
397 nomecurto=str(grau)+"["+nome+"]"
398 
399 nome=str(grau)+"["+nome+"]"+"~"+fators+"%"+"~"+larguras+"x"+alturas+"~"+raix
400 
401 nomedaimagem = nomecurto
402 
403 
404 
405 criaimagem(nomedaimagem,"JPEG",largura,altura,pol,dpol,raizesarredondadas,fator)
406 
407 #criaimagem("david01.png","PNG",1024,512,pol,dpol,raizesarredondadas)
408 
409 
410 
411 
```

Apêndices

APÊNDICE B – Informações técnicas das imagens

Figura 5

Polinômio = 1[(1+0j),(-3+3j)]
Raízes = [(3-3j)]
Largura da imagem = 2000 pixels
Altura da imagem = 1500 pixels
Total de pixels = 3000000
Quantidade de pixels não convergentes = 0 = 0.0 %
Quantidade de pixels cor(0) = 3000000 = 100.0 %
Abscissa mínima = -4.4
Abscissa máxima = 4.4
Ordenada mínima = -3.3000000000000003
Ordenada máxima = 3.3000000000000003
Tempo de execução = 3 minutos

Figura 7

Polinômio = 1[(1+0j),(1+0j)]
Raízes = [(-1+0j)]
Largura da imagem = 2000 pixels
Altura da imagem = 1500 pixels
Total de pixels = 3000000
Quantidade de pixels não convergentes = 0 = 0.0 %
Quantidade de pixels cor(0) = 3000000 = 100.0 %
Abscissa mínima = -1.1
Abscissa máxima = 1.1
Ordenada mínima = -0.8250000000000001
Ordenada máxima = 0.8250000000000001
Tempo de execução = 3 minutos

Figura 9

Polinômio = 2[(1+0j),(1+0j),(1+0j)]
Raízes = [(-0.5+0.866j), (-0.5-0.866j)]
Largura da imagem = 2000 pixels
Altura da imagem = 1500 pixels
Total de pixels = 3000000
Quantidade de pixels não convergentes = 2000 = 0.06666666666666667 %
Quantidade de pixels cor(0) = 1500000 = 50.0 %
Quantidade de pixels cor(1) = 1498000 = 49.93333333333334 %
Abscissa mínima = -1.2701705922171769
Abscissa máxima = 1.2701705922171769
Ordenada mínima = -0.9526279441628827
Ordenada máxima = 0.9526279441628827
Tempo de execução = 13 minutos

Figura 11

Polinômio = 2[(1+0j),0j,(-75+100j)]
Raízes = [(10-5j), (-10+5j)]
Largura da imagem = 2000 pixels
Altura da imagem = 1500 pixels
Total de pixels = 3000000
Quantidade de pixels não convergentes = 750 = 0.025 %
Quantidade de pixels cor(0) = 1498500 = 49.95 %
Quantidade de pixels cor(1) = 1500750 = 50.025 %
Abscissa mínima = -11.0
Abscissa máxima = 11.0

Ordenada mínima = -8.25
Ordenada máxima = 8.25
Tempo de execução = 14 minutos

Figura 13

Polinômio = 3[(1+0j),(1+0j),(1+0j),(1+0j)]
Raízes = [1j, (-1-0j), -1j]
Largura da imagem = 2000 pixels
Altura da imagem = 1500 pixels
Total de pixels = 3000000
Quantidade de pixels não convergentes = 20 = 0.0006666666666666666 %
Quantidade de pixels cor(0) = 683324 = 22.777466666666665 %
Quantidade de pixels cor(1) = 1634870 = 54.495666666666665 %
Quantidade de pixels cor(2) = 681786 = 22.7262 %
Abscissa mínima = -1.4666666666666677
Abscissa máxima = 1.4666666666666677
Ordenada mínima = -1.1000000000000008
Ordenada máxima = 1.1000000000000008
Tempo de execução = 16 minutos

Figura 15

Polinômio = 3[(1+0j),1.18j,(-13.2+0j),-1397.5j]
Raízes = [(10+5j), (-0-11.18j), (-10+5j)]
Largura da imagem = 2000 pixels
Altura da imagem = 1500 pixels
Total de pixels = 3000000
Quantidade de pixels não convergentes = 616 = 0.020533333333333334 %
Quantidade de pixels cor(0) = 1005940 = 33.531333333333336 %
Quantidade de pixels cor(1) = 986470 = 32.882333333333335 %
Quantidade de pixels cor(2) = 1006974 = 33.5658 %
Abscissa mínima = -16.397333333333343
Abscissa máxima = 16.397333333333343
Ordenada mínima = -12.298000000000007
Ordenada máxima = 12.298000000000007
Tempo de execução = 17 minutos

Figura 17

Polinômio = 4[(1+0j),(1+0j),(1+0j),(1+0j),(1+0j)]
Raízes = [(0.309+0.951j), (-0.809+0.588j), (0.309-0.951j), (-0.809-0.588j)]
Largura da imagem = 2000 pixels
Altura da imagem = 1500 pixels
Total de pixels = 3000000
Quantidade de pixels não convergentes = 2242 = 0.07473333333333333 %
Quantidade de pixels cor(0) = 455365 = 15.178833333333333 %
Quantidade de pixels cor(1) = 1044401 = 34.81336666666667 %
Quantidade de pixels cor(2) = 454219 = 15.140633333333334 %
Quantidade de pixels cor(3) = 1043773 = 34.792433333333335 %
Abscissa mínima = -1.3948828905662265
Abscissa máxima = 1.3948828905662265
Ordenada mínima = -1.04616216792467
Ordenada máxima = 1.04616216792467
Tempo de execução = 16 minutos

Figura 19

```
Polinômio = 4[(1+0j),0j,0j,0j,(1+0j)]
Raízes = [(-0.707+0.707j), (-0.707-0.707j), (0.707+0.707j), (0.707-
0.707j)]
Largura da imagem = 2000 pixels
Altura da imagem = 1500 pixels
Total de pixels = 3000000
Quantidade de pixels não convergentes = 64501 = 2.1500333333333335 %
Quantidade de pixels cor(0) = 734547 = 24.4849 %
Quantidade de pixels cor(1) = 733638 = 24.454600000000003 %
Quantidade de pixels cor(2) = 734111 = 24.470366666666667 %
Quantidade de pixels cor(3) = 733203 = 24.4401 %
Abscissa mínima = -1.03708994574027
Abscissa máxima = 1.03708994574027
Ordenada mínima = -0.7778174593052026
Ordenada máxima = 0.7778174593052026
Tempo de execução = 21 minutos
```

Figura 21

```
Polinômio = 5[(1+0j),(1+0j),(1+0j),(1+0j),(1+0j),(1+0j)]
Raízes = [(0.5+0.866j), (-0.5+0.866j), (-1+0j), (0.5-0.866j), (-0.5-
0.866j)]
Largura da imagem = 2000 pixels
Altura da imagem = 1500 pixels
Total de pixels = 3000000
Quantidade de pixels não convergentes = 3247 = 0.10823333333333333 %
Quantidade de pixels cor(0) = 381094 = 12.703133333333334 %
Quantidade de pixels cor(1) = 544617 = 18.1539 %
Quantidade de pixels cor(2) = 1147307 = 38.243566666666666 %
Quantidade de pixels cor(3) = 380204 = 12.673466666666666 %
Quantidade de pixels cor(4) = 543531 = 18.1177 %
Abscissa mínima = -1.270170592217178
Abscissa máxima = 1.270170592217178
Ordenada mínima = -0.9526279441628835
Ordenada máxima = 0.9526279441628835
Tempo de execução = 21 minutos
```

Figura 23

```
Polinômio = 5[(1+0j),0j,(1+0j),0j,(1+0j),0j]
Raízes = [(0.5+0.866j), (-0.5+0.866j), (0.5-0.866j), (-0.5-0.866j), 0j]
Largura da imagem = 2000 pixels
Altura da imagem = 1500 pixels
Total de pixels = 3000000
Quantidade de pixels não convergentes = 114 = 0.0038000000000000004 %
Quantidade de pixels cor(0) = 266514 = 8.8838 %
Quantidade de pixels cor(1) = 266695 = 8.889833333333334 %
Quantidade de pixels cor(2) = 265677 = 8.8559 %
Quantidade de pixels cor(3) = 265857 = 8.8619 %
Quantidade de pixels cor(4) = 1935143 = 64.50476666666667 %
Abscissa mínima = -1.270170592217178
Abscissa máxima = 1.270170592217178
Ordenada mínima = -0.9526279441628835
Ordenada máxima = 0.9526279441628835
Tempo de execução = 15 minutos
```

Figura 25

```
Polinômio = 6[(1+0j),(1+0j),(1+0j),(1+0j),(1+0j),(1+0j),(1+0j)]
Raízes = [(0.623+0.782j), (-0.223+0.975j), (-0.901+0.434j), (0.623-
0.782j), (-0.901-0.434j), (-0.223-0.975j)]
Largura da imagem = 2000 pixels
Altura da imagem = 1500 pixels
Total de pixels = 3000000
Quantidade de pixels não convergentes = 6508 = 0.21693333333333334 %
Quantidade de pixels cor(0) = 346995 = 11.5665 %
Quantidade de pixels cor(1) = 257520 = 8.584 %
Quantidade de pixels cor(2) = 893212 = 29.773733333333336 %
Quantidade de pixels cor(3) = 346266 = 11.5422 %
Quantidade de pixels cor(4) = 892604 = 29.753466666666668 %
Quantidade de pixels cor(5) = 256895 = 8.563166666666666 %
Abscissa mínima = -1.429894271200009
Abscissa máxima = 1.429894271200009
Ordenada mínima = -1.0724207034000068
Ordenada máxima = 1.0724207034000068
Tempo de execução = 22 minutos

Figura 27

Polinômio = 6[(1+0j),0j,0j,0j,0j,0j,(1+0j)]
Raízes = [(-0.866+0.5j), (-0.866-0.5j), 1j, -1j, (0.866+0.5j), (0.866-
0.5j)]
Largura da imagem = 2000 pixels
Altura da imagem = 1500 pixels
Total de pixels = 3000000
Quantidade de pixels não convergentes = 326758 = 10.891933333333334 %
Quantidade de pixels cor(0) = 524981 = 17.499366666666667 %
Quantidade de pixels cor(1) = 524521 = 17.484033333333333 %
Quantidade de pixels cor(2) = 288088 = 9.602933333333333 %
Quantidade de pixels cor(3) = 287387 = 9.579566666666667 %
Quantidade de pixels cor(4) = 524362 = 17.47873333333333 %
Quantidade de pixels cor(5) = 523903 = 17.463433333333334 %
Abscissa mínima = -1.4666666666666677
Abscissa máxima = 1.4666666666666677
Ordenada mínima = -1.1000000000000008
Ordenada máxima = 1.1000000000000008
Tempo de execução = 30 minutos

Figura 29

Polinômio = 7[(0.1+0j),(1+1j),0j,0j,0j,0j,0j,(1+0j)]
Raízes = [(-10-10j), (-0.883-0.361j), (-0.743+0.583j), (0.128+0.927j),
(0.862+0.361j), (-0.117-0.945j), (0.754-0.565j)]
Largura da imagem = 2000 pixels
Altura da imagem = 1500 pixels
Total de pixels = 3000000
Quantidade de pixels não convergentes = 347054 = 11.568466666666668 %
Quantidade de pixels cor(0) = 0 = 0.0 %
Quantidade de pixels cor(1) = 554136 = 18.4712 %
Quantidade de pixels cor(2) = 473088 = 15.7696 %
Quantidade de pixels cor(3) = 291006 = 9.7002 %
Quantidade de pixels cor(4) = 509060 = 16.968666666666667 %
Quantidade de pixels cor(5) = 312631 = 10.421033333333334 %
Quantidade de pixels cor(6) = 513025 = 17.100833333333334 %
Abscissa mínima = -14.666668499999282
```

Abscissa máxima = 14.666668499999282
Ordenada mínima = -11.000001374999462
Ordenada máxima = 11.000001374999462
Tempo de execução = 49 minutos

Figura 31

Polinômio = 7[(1+0j),(1+1j),0j,0j,0j,0j,0j,(1+0j)]
Raízes = [(-1.018-1.089j), (-1.019-0.268j), (-0.671+0.628j), -1j,
(0.148+0.865j), (0.799+0.353j), (0.761-0.488j)]
Largura da imagem = 2000 pixels
Altura da imagem = 1500 pixels
Total de pixels = 3000000
Quantidade de pixels não convergentes = 357625 = 11.920833333333333 %
Quantidade de pixels cor(0) = 184086 = 6.1362 %
Quantidade de pixels cor(1) = 425273 = 14.175766666666668 %
Quantidade de pixels cor(2) = 466003 = 15.533433333333333 %
Quantidade de pixels cor(3) = 254354 = 8.478466666666666 %
Quantidade de pixels cor(4) = 292900 = 9.763333333333334 %
Quantidade de pixels cor(5) = 517681 = 17.256033333333335 %
Quantidade de pixels cor(6) = 502078 = 16.73593333333333 %
Abscissa mínima = -1.5974461713320227
Abscissa máxima = 1.5974461713320227
Ordenada mínima = -1.198084628499017
Ordenada máxima = 1.198084628499017
Tempo de execução = 33 minutos

Figura 33

Polinômio = 8[(0.1+0j),0j,1j,0j,0j,0j,0j,0j,(1+0j)]
Raízes = [(2.235-2.235j), (-2.235+2.235j), (-0.954-0.269j), (-0.269-
0.954j), (-0.72+0.72j), (0.72-0.72j), (0.269+0.954j), (0.954+0.269j)]
Largura da imagem = 2000 pixels
Altura da imagem = 1500 pixels
Total de pixels = 3000000
Quantidade de pixels não convergentes = 211640 = 7.054666666666666 %
Quantidade de pixels cor(0) = 95874 = 3.1958 %
Quantidade de pixels cor(1) = 96443 = 3.2147666666666663 %
Quantidade de pixels cor(2) = 481264 = 16.042133333333332 %
Quantidade de pixels cor(3) = 301337 = 10.044566666666666 %
Quantidade de pixels cor(4) = 515875 = 17.195833333333333 %
Quantidade de pixels cor(5) = 514993 = 17.166433333333334 %
Quantidade de pixels cor(6) = 302006 = 10.066866666666668 %
Quantidade de pixels cor(7) = 480568 = 16.018933333333333 %
Abscissa mínima = -3.2779212270911287
Abscissa máxima = 3.2779212270911287
Ordenada mínima = -2.4584409203183464
Ordenada máxima = 2.4584409203183464
Tempo de execução = 34 minutos

Figura 35

Polinômio = 8[(10+0j),(7-7j),(-7+7j),(6-6j),(-6+6j),(5-5j),(-5+5j),(4-
4j),(-4+4j)]
Raízes = [(-1.652+0.719j), (-0.571+0.7j), (0.087+0.846j), (0.624+0.541j),
(0.858-0.061j), (-0.592-0.618j), (0.551-0.601j), (-0.005-0.825j)]
Largura da imagem = 2000 pixels
Altura da imagem = 1500 pixels

Total de pixels = 3000000
Quantidade de pixels não convergentes = 22642 = 0.7547333333333333 %
Quantidade de pixels cor(0) = 75249 = 2.5083 %
Quantidade de pixels cor(1) = 399267 = 13.308900000000001 %
Quantidade de pixels cor(2) = 241616 = 8.053866666666666 %
Quantidade de pixels cor(3) = 630353 = 21.011766666666666 %
Quantidade de pixels cor(4) = 498553 = 16.618433333333332 %
Quantidade de pixels cor(5) = 358627 = 11.954233333333333 %
Quantidade de pixels cor(6) = 558119 = 18.603966666666665 %
Quantidade de pixels cor(7) = 215574 = 7.1858 %
Abscissa mínima = -1.8174001459735836
Abscissa máxima = 1.8174001459735836
Ordenada mínima = -1.3630501094801877
Ordenada máxima = 1.3630501094801877
Tempo de execução = 30 minutos

Figura 37

Polinômio = 8[(8+0j),(1-1j),(-1+1j),(2-2j),(-2+2j),(3-3j),(-3+3j),(4-4j),(-4+4j)]
Raízes = [(-1.25+0.226j), (-0.532+0.816j), (0.163+0.91j), (0.726+0.539j), (-0.669-0.722j), (0.831-0.057j), (0.61-0.66j), (-0.004-0.927j)]
Largura da imagem = 2000 pixels
Altura da imagem = 1500 pixels
Total de pixels = 3000000
Quantidade de pixels não convergentes = 12204 = 0.40679999999999994 %
Quantidade de pixels cor(0) = 95233 = 3.1744333333333334 %
Quantidade de pixels cor(1) = 199224 = 6.6408 %
Quantidade de pixels cor(2) = 171986 = 5.732866666666666 %
Quantidade de pixels cor(3) = 626363 = 20.878766666666667 %
Quantidade de pixels cor(4) = 252328 = 8.410933333333332 %
Quantidade de pixels cor(5) = 1030112 = 34.337066666666665 %
Quantidade de pixels cor(6) = 470125 = 15.670833333333334 %
Quantidade de pixels cor(7) = 142425 = 4.7475000000000005 %
Abscissa mínima = -1.374516733593255
Abscissa máxima = 1.374516733593255
Ordenada mínima = -1.0308875501949413
Ordenada máxima = 1.0308875501949413
Tempo de execução = 28 minutos

Figura 39

Polinômio = 8[1j,1j,1j,(1-1j),(2+1j),(3-54j),(9-1j),(3-1j),(7+87j)]
Raízes = [(-1.98+1.403j), (-1.937-1.33j), (0.508+2.12j), (0.509-2.16j), (-0.581+0.99j), (-0.618-1.034j), (1.937+0.19j), (1.162-0.178j)]
Largura da imagem = 2000 pixels
Altura da imagem = 1500 pixels
Total de pixels = 3000000
Quantidade de pixels não convergentes = 24760 = 0.8253333333333333 %
Quantidade de pixels cor(0) = 220676 = 7.355866666666666 %
Quantidade de pixels cor(1) = 240604 = 8.020133333333334 %
Quantidade de pixels cor(2) = 86860 = 2.8953333333333333 %
Quantidade de pixels cor(3) = 97765 = 3.258833333333333 %
Quantidade de pixels cor(4) = 664567 = 22.15223333333333 %
Quantidade de pixels cor(5) = 649344 = 21.6448 %
Quantidade de pixels cor(6) = 244175 = 8.139166666666666 %
Quantidade de pixels cor(7) = 771249 = 25.7083 %
Abscissa mínima = -3.168137497611443

Abscissa máxima = 3.168137497611443
Ordenada mínima = -2.376103123208582
Ordenada máxima = 2.376103123208582
Tempo de execução = 26 minutos

Figura 41

Polinômio = 8[1j,1j,1j,1j,1j,1j,1j,1j,1j]
Raízes = [(0.766+0.643j), (0.174+0.985j), (-0.5+0.866j), (-0.94+0.342j), (-0.94-0.342j), (0.766-0.643j), (-0.5-0.866j), (0.174-0.985j)]
Largura da imagem = 2000 pixels
Altura da imagem = 1500 pixels
Total de pixels = 3000000
Quantidade de pixels não convergentes = 21340 = 0.7113333333333334 %
Quantidade de pixels cor(0) = 291873 = 9.7291 %
Quantidade de pixels cor(1) = 139306 = 4.643533333333334 %
Quantidade de pixels cor(2) = 349799 = 11.659966666666667 %
Quantidade de pixels cor(3) = 709265 = 23.642166666666668 %
Quantidade de pixels cor(4) = 709161 = 23.6387 %
Quantidade de pixels cor(5) = 291417 = 9.7139 %
Quantidade de pixels cor(6) = 349030 = 11.634333333333334 %
Quantidade de pixels cor(7) = 138809 = 4.626966666666667 %
Abscissa mínima = -1.4443847044179057
Abscissa máxima = 1.4443847044179057
Ordenada mínima = -1.0832885283134293
Ordenada máxima = 1.0832885283134293
Tempo de execução = 29 minutos

Tempo total de execução = 429 min