Centro Federal de Educação Tecnológica de Minas Gerais
PROFMAT - Mestrado Profissional em Matemática em Rede Nacional

Edson Bertosa Lopes Santos

Elaboração de uma apostila sobre programação e métodos numéricos para o ensino médio

Belo Horizonte
2024

Edson Bertosa Lopes Santos

ELABORAÇÃO DE UMA APOSTILA SOBRE PROGRAMAÇÃO E MÉTODOS NUMÉRICOS PARA O ENSINO MÉDIO

Dissertação apresentada ao Centro Federal de Educação Tecnológica de Minas Gerais como parte das exigências do Programa de Pós-Graduação Mestrado Profissional em Matemática em Rede Nacional, para obter o título de Mestre.

Orientador
Luis Alberto D'Afonseca

Coorientador
Carlos Magno Martins Cosme

Banca Examinadora
Dênis Emanuel da Costa Vargas
Jônathas Douglas Santos de Oliveira
Mehran Sabeti

Belo Horizonte
2024

Santos, Edson Bertosa Lopes
S237e Elaboração de uma apostila sobre programação e métodos numéricos para o ensino médio / Edson Bertosa Lopes Santos. – 2024.
95 f.

Dissertação de mestrado apresentada ao Programa de Mestrado Profissional em Matemática em Rede Nacional.
Orientador: Luís Alberto D'Afonseca.
Coorientador: Carlos Magno.
Dissertação (mestrado) – Centro Federal de Educação Tecnológica de Minas Gerais.

1. Programação (Computadores) – Teses. 2. Métodos e algoritmos numéricos – Teses. 3. Ensino médio – Teses. 4. Python (Linguagem de programação de computador) – Teses. 5. Funções – Teses. 6. Sistemas não lineares – Teses. I. D'Afonseca, Luís Alberto. II. Cosme, Carlos Magno Martins. III. Centro Federal de Educação Tecnológica de Minas Gerais. IV. Título.

CDD 515.3

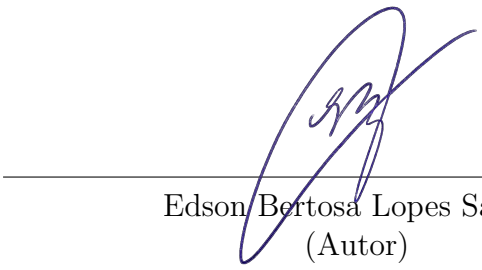
Elaboração da ficha catalográfica pela bibliotecária Jane Marangon Duarte, CRB 6ª 1592 / Cefet/MG

Edson Bertosa Lopes Santos

ELABORAÇÃO DE UMA APOSTILA SOBRE PROGRAMAÇÃO E MÉTODOS NUMÉRICOS PARA O ENSINO MÉDIO

Dissertação apresentada ao Centro Federal de Educação Tecnológica de Minas Gerais como parte das exigências do Programa de Pós-Graduação Mestrado Profissional em Matemática em Rede Nacional, para obter o título de Mestre.

Aprovada: 12 de setembro de 2024.



Edson Bertosa Lopes Santos
(Autor)

Luis Alberto D'Afonseca
(Orientador)

Belo Horizonte
2024

Dedico este trabalho a minha mãe, ao meu filho e a minha esposa por terem me incentivado a caminhar para frente sempre!

Agradecimentos

A Deus, por me conceder saúde, oportunidade e o privilégio de conquistar mais uma vitória em minha carreira ao adquirir esse título de mestre mesmo frente a tantos desafios.
À minha esposa, Marciana, quero expressar meu sincero agradecimento. Você foi meu apoio constante ao longo dessa jornada desafiadora. Seu amor e paciência foram fundamentais para me manter motivado e focado em alcançar meus objetivos. Agradeço de coração por estar ao meu lado, por me encorajar e por me apoiar nos momentos mais difíceis. Sem você, eu não teria conseguido chegar tão longe. Sou imensamente grato por todo o seu amor e dedicação. Obrigado, minha amada Marciana.
À minha mãe Ana, desde os tempos de escola, seu apoio e inspiração foram fundamentais. Sua crença em mim foi essencial para superar os desafios e seguir em busca do conhecimento. Sua confiança constante em minhas habilidades me deu forças para continuar, mesmo quando as coisas ficaram difíceis. Sou profundamente grato por tudo que você fez por mim. Agradeço de coração, minha querida mãe Ana, por todo o amor, dedicação e por sempre acreditar em mim.
A Luis Alberto D'Afonseca, gostaria de expressar minha sincera gratidão pelo seu papel fundamental como meu orientador e coordenador durante meu mestrado. Sua experiência, conhecimento e orientação foram essenciais para o desenvolvimento da minha pesquisa e para o aprimoramento das minhas habilidades acadêmicas. Agradeço também por sua paciência, incentivo e por me desafiar a ir além do que achava ser capaz. Suas contribuições foram valiosas para o sucesso deste projeto. Sou grato por ter tido a oportunidade de trabalhar sob sua orientação e agradeço de coração por todo o apoio e orientação que você me proporcionou ao longo desta jornada.
Por fim, quero deixar meu agradecimento ao meu co-orientador Carlos Magno e a todos os professores que desempenharam seu trabalho de forma excelente.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001.

Resumo

Este trabalho tem como objetivo apresentar uma apostila direcionada aos professores do Ensino Médio, com o propósito de introduzir o Pensamento Computacional, a linguagem de programação Python e a implementação de métodos numéricos na busca por soluções de equações não lineares e de sistemas de equações lineares e não lineares com a utilização de recursos computacionais. Afim de embasar nosso projeto, vamos citar alguns conceitos relacionados a definição de Pensamento Computacional e também algumas competências da Base Nacional Comum Curricular (BNCC) a esse respeito. Com o objetivo de compor um grupo de referências para sustentar as intenções desse projeto, faremos uma análise de trabalhos similares, buscando identificar como os temas acima citados têm sido explorados no contexto da sala de aula. Quanto ao uso dos recursos computacionais, vamos explorar a aritmética computacional e alguns conceitos como erro de arredondamento, número de máquina, algoritmo e estabilidade. Em seguida vamos explorar alguns métodos numéricos clássicos para a busca de zeros de funções de uma variável e para a busca de soluções de sistemas de equações lineares e não lineares. Portanto, a proposta central desse projeto é contribuir para melhoria da qualidade do processo de ensino-aprendizagem na educação básica disponibilizando um material didático que poderá servir como subsídio para fomentar as discussões sobre pensamento computacional e métodos numéricos iterativos usando recursos computacionais na sala de aula.

Palavras-chave: Pensamento Computacional, Python, linguagem de programação, métodos numéricos, zeros de funções, sistemas não lineares.

Abstract

This work aims to present a handbook directed at high school teachers, with the purpose of introducing Computational Thinking, the Python programming language, and the implementation of numerical methods for solving nonlinear equations and systems of linear and nonlinear equations using computational resources. To support our project, we will discuss some concepts related to the definition of Computational Thinking and also some competencies from the National Common Curricular Base (BNCC) in this regard. In order to compile a group of references to support the objectives of this project, we will analyze similar works to identify how the topics mentioned above have been explored in the classroom context. Regarding the use of computational resources, we will explore computational arithmetic and concepts such as rounding error, machine number, algorithm, and stability. We will then explore some classical numerical methods for finding the zeros of single-variable functions and for solving systems of linear and nonlinear equations. Therefore, the central proposal of this project is to contribute to improving the quality of the teaching-learning process in basic education by providing didactic material that can serve as a resource to foster discussions on computational thinking and iterative numerical methods using computational resources in the classroom.

Keywords: Computational Thinking, Python, programming language, numerical methods, function zeros, nonlinear systems.

Sumário

1 Introdução

Nos dias atuais, nossa realidade está ligada à tecnologia, transformando a forma como vivemos e aprendemos. Nesse contexto, o Pensamento Computacional emerge como uma habilidade importante. Sua presença nas salas de aula pode preparar as novas gerações para um futuro impulsionado pela tecnologia. A Base Nacional Comum Curricular (BNCC) [3] reconhece essa importância ao destacar o papel do Pensamento Computacional na educação, ressaltando sua significativa contribuição para a formação dos estudantes e impulsionando o progresso da aprendizagem contemporânea. Neste trabalho, discutiremos como esse pensamento se encaixa na estrutura educacional, conforme descrito na BNCC, e como ele se tornou uma habilidade importante para enfrentar os desafios do mundo moderno. Apresentaremos ainda um estudo sobre a aritmética computacional, abordando números de máquina e erros de aproximação, mostrando como esses conceitos impactam os cálculos e as representações numéricas em computadores. Faremos uma análise de trabalhos similares, dissertações que exploram os mesmos temas abordados aqui, onde serão analisadas algumas dissertações defendidas no programa de mestrado PROFMAT e algumas dissertações defendidas em outros programas de mestrado com o intuito de compor um grupo de referências para sustentar as intenções desse projeto. Falaremos sobre algoritmo e estabilidade, elementos importantes na aplicação de métodos numéricos iterativos com o uso de recursos computacionais e apresentaremos alguns métodos clássicos para a busca de soluções de equações não lineares de uma variável e de sistemas de equações lineares e não lineares.

Em minha experiência como professor do ensino Fundamental II e Ensino Médio, pude perceber que, mesmo nascidos em meio à tecnologia, os alunos não entendem o funcionamento interno das máquinas, usam celulares e executam aplicativos, mas não entendem sua lógica de operação. Além disso, notei que, ao realizarem cálculos com a calculadora e obterem resultados irracionais, os alunos frequentemente se questionam sobre o significado de todos aqueles dígitos e, em muitos casos, acreditam ter cometido um erro. Também levantam questões como: “Como a calculadora consegue calcular todos esses dígitos após a vírgula?” São questões dessa natureza que inspiraram o desenvolvimento deste trabalho, cujo objetivo principal é criar uma apostila destinada a servir como um auxílio

didático para discussões sobre a introdução ao Pensamento Computacional e à linguagem de programação, a utilização de métodos numéricos iterativos na busca de soluções para equações e sistemas de equações não lineares, e o uso de recursos computacionais na execução desses métodos. A intenção é proporcionar aos alunos uma melhor compreensão desses temas no contexto da sala de aula.

Na apostila, as atividades nela propostas destinam-se a estudantes do Ensino Médio, visando a inserção do Pensamento Computacional e a introdução ao conhecimento sobre Linguagem de Programação, que neste projeto será utilizada a linguagem de programação Python, uma linguagem de programação de alto nível, de fácil acesso e compreensão. Destacamos aqui que não é necessário nenhum conhecimento prévio sobre programação por parte dos professores e dos alunos, todos os passos estão descritos na apostila, desde o download e instalação até a execução do programa, além disso, usaremos um aplicativo gratuito no caso do uso do smartphone ou o Colab, que é um ambiente de programação online, no caso do uso do computador, não sendo necessário nenhum tipo de instalação. Quanto aos métodos numéricos, posteriormente resolveremos problemas que envolvem solução de equações ou de sistemas de equações não lineares através de métodos numéricos convencionais utilizando recursos computacionais. Aqui será apresentado o desenvolvimento matemático completo dos métodos, porém na apostila a abordagem é adaptada para o nível básico.

Nesse sentido, nosso trabalho está dividido em duas partes: a dissertação propriamente dita e o produto (apostila). Em relação a primeira parte, que se constitui nesse documento, está estruturado da seguinte maneira: No Capítulo 2, abordaremos o Pensamento Computacional, sua importância na educação, e sua relação com a programação, destacando seu papel no desenvolvimento de habilidades essenciais. Seguindo no Capítulo 3 com a análise de trabalhos similares quanto aos temas abordades neste. No Capítulo 4, abordaremos números de máquina e erros de aproximação dentro da aritmética computacional e também falaremos sobre algoritmo e estabilidade destacando como a escolha e implementação adequada de algoritmos contribuem para a precisão e eficácia das soluções computacionais. Continuando, no Capitulo 5, com a apresentação de alguns métodos numéricos iterativos clássicos para zeros de funções de uma variável e no Capítulo 6, vamos descrever métodos numéricos iterativos para solução de sistemas lineares e não lineares. O Capítulo 7 descreve e apresenta o produto central desse tralho que é a criação de uma apostila, e por fim, no Capitulo 8 apresentamos nossas percepções e considerações observadas durante o desenvolvimento desse trabalho.

O texto completo da apostila segue anexada à este documento.

2 Pensamento Computacional no Ensino

Nesse capítulo vamos apresentar o conceito de Pensamento Computacional e como ele vem sendo abordado nas práticas educacionais. O conceito não é novo, remontando pelo menos ao início da década de 80 quando Papert (1980) [13] já defendia a ideia de que:

> Aprender a se comunicar com um computador pode mudar a forma como as pessoas pensam, favorecendo um outro tipo de aprendizado (PAPERT, 1980, p. 6).

Grover e Pea (2013) [10], por sua vez, consideram que o Pensamento Computacional deve ser parte do conhecimentos que buscamos desenvolver nas crianças por meio de resolução de problemas do cotidiano. Segundo os autores, se reconhecermos a importância de uma alfabetização científica para a compreensão do mundo, é necessário incluir o entendimento do funcionamento interno dos aplicativos e computadores. Eles sustentam a perspectiva de que essas competências são indispensáveis em um cenário digital. Mais recentemente, Wing (2006) [21], o considera como um conjunto de habilidades e competências ligadas à Ciência da Computação, as quais os estudantes devem desenvolver desde os primeiros anos escolares. Wing considera o Pensamento Computacional como uma forma de pensar inerente aos seres humanos, uma vez que envolve diversas habilidades que requerem diferentes níveis de abstração. Essa forma de pensar distingue-se do funcionamento das máquinas. Portanto, o Pensamento Computacional não se limita apenas àqueles que trabalham na área da computação, mas é uma habilidade intelectual fundamental para todos os indivíduos, assim como ler, escrever ou realizar operações matemáticas.

Uma outra visão sobre o tema é dada por foi dada por Bordini (2016) [1]. De acordo com o autor, o Pensamento Computacional possui um conjunto de habilidades específicas, não limitado somente a essas, como:

1. Fazer a formulação de um problema de tal forma que seja possível a sua resolução através de computadores e outras ferramentas.
2. Fazer a organização lógica e a análise de dados;
3. Representar dados através de abstrações, modelos ou simulações;

4. Automatização de soluções através de algoritmos;
5. Identificação, análise e implementação de soluções de forma mais eficiente e eficaz;
6. Fazer a generalização e a transferência da forma de resolução para outros problemas.

O autor destaca a importância de reconhecer o Pensamento Computacional como uma habilidade fundamental para todos os indivíduos, e enfatizam a necessidade de desenvolver habilidades cognitivas, preparando os estudantes para os desafios da sociedade digital contemporânea.

No item 4, sobre o conjunto de habilidades específicas dadas acima, foi citada a palavra algoritmo. Um algoritmo é uma sequência finita de regras a serem aplicadas em uma determinada ordem para um número finito de dados para chegar com certeza (ou seja, sem indeterminação ou ambiguidade), em um número finito de etapas, em um determinado resultado e que independa dos dados (BOUVIER, et.al., 2005) [2]. A concepção de algoritmo não é intrínseca à disciplina da Ciência da Computação, a qual tem suas raízes na Matemática e está historicamente ligada a manipulações numéricas no sistema decimal, como adição, subtração, busca de divisores, e outras operações similares. No contexto atual, um algoritmo pode ser expresso em linguagem natural ou codificado em uma linguagem de programação, resultando em um programa que pode ser executado por um computador.

Ainda de acordo com Bordini (2016), essas habilidades são apoiadas e realçadas por uma série de qualidades ou atitudes que são dimensões essenciais do Pensamento Computacional. Essas qualidades ou atitudes incluem:

1. Confiança em lidar com a complexidade;
2. Persistência no trabalho com problemas difíceis;
3. Capacidade de lidar com problemas em aberto.

Com a implantação da nova Base Nacional Comum Curricular (BNCC), podemos verificar que o Pensamento Computacional e o uso de tecnologias é citado por diversas vezes nesse documento. Abaixo citamos alguns trechos que evidenciam esse fato.

> "Outro aspecto a ser considerado é que a aprendizagem de Álgebra, como também aquelas relacionadas a Números, Geometria e Probabilidade e estatística, podem contribuir para o desenvolvimento do Pensamento Computacional dos alunos, tendo em vista que eles precisam ser capazes de traduzir uma situação dada em outras linguagens, como transformar situações-problema, apresentadas em língua materna, em fórmulas, tabelas e gráficos e vice-versa.
>
> Associado ao Pensamento Computacional, cumpre salientar a importância dos algoritmos e de seus fluxogramas, que podem ser objetos de estudo nas aulas de Matemática.
>
> Outra habilidade relativa à álgebra que mantém estreita relação com o

> Pensamento Computacional é a identificação de padrões para se estabelecer generalizações, propriedades e algoritmos. (BNCC, 2018, p.271)."
>
> "Utilizar, propor e/ou implementar soluções (processos e produtos) envolvendo diferentes tecnologias, para identificar, analisar, modelar e solucionar problemas complexos em diversas áreas da vida cotidiana, explorando de forma efetiva o raciocínio lógico, o Pensamento Computacional, o espírito de investigação e a criatividade. (BNCC, 2018, p. 475)."

Um outro trecho importante da BNCC é:

> "Além disso, a BNCC propõe que os estudantes utilizem tecnologias, como calculadoras e planilhas eletrônicas, desde os anos iniciais do Ensino Fundamental. Tal valorização possibilita que, ao chegarem aos anos finais, eles possam ser estimulados a desenvolver o Pensamento Computacional, por meio da interpretação e da elaboração de algoritmos, incluindo aqueles que podem ser representados por fluxogramas.(BNCC, 2018, p. 528)".

Embora o Pensamento Computacional seja mencionado explicitamente na BNCC, ainda é um tema novo e levanta muitas dúvidas em relação à sua compreensão e inclusão nos currículos escolares. Tais dúvidas geram questionamentos como: será que simplesmente consultar o texto base da BNCC é suficiente para apoiar os professores no desenvolvimento do Pensamento Computacional durante as aulas de Matemática? Será que os professores de Matemática possuem conhecimento sólido sobre o Pensamento Computacional? Eles se sentem confortáveis em utilizar as ferramentas tecnológicas necessárias? Além disso, as escolas públicas de ensino básico possuem os recursos e equipamentos adequados para criar sistemas digitais ou oferecer suporte para o desenvolvimento e a prática do Pensamento Computacional?

Essas questões reforçam a necessidade de uma reflexão mais profunda sobre a integração do Pensamento Computacional no ambiente escolar. Ao concluirmos essa breve discussão, somos levados a concordar com os diversos autores sobre a importância do Pensamento Computacional para os indivíduos na era contemporânea. A exploração das definições e dimensões do Pensamento Computacional ressalta a importância de sua integração no âmbito escolar, considerando um panorama cada vez mais tecnológico que molda nosso mundo. Implantar esse tipo de mentalidade nos alunos pode, não apenas expandir suas perspectivas, mas também ajudar no preparo para enfrentar os desafios e oportunidades do futuro.

3 Estudos Relacionados

Neste capítulo, buscamos identificar como os temas Linguagem de Programação, Métodos Numéricos ou Cálculo Numérico e Sistemas Lineares e Não Lineares têm sido explorados e desenvolvidos no contexto das salas de aula da educação básica. Realizamos essa análise por meio de uma pesquisa bibliográfica em dissertações de mestrado voltadas para o Ensino de Matemática.

Durante a pesquisa, estabelecemos critérios para selecionar dissertações a serem analisadas de maneira mais minuciosa, a fim de compor um grupo de referências que sustentaria as intenções deste projeto. Os critérios adotados foram os seguintes:

1. Como o tema abordado na dissertação analisada se relaciona com a proposta deste trabalho?
2. Como a metodologia utilizada pode contribuir para a elaboração e seleção do método deste projeto?
3. Quais são os tipos de experiências observadas pelos autores?
4. Quais referências foram utilizadas?
5. Quais são os resultados e contribuições para a área?

Iniciamos nossa pesquisa consultando a Lista de Dissertações do PROFMAT, disponível em https://profmat-sbm.org.br/dissertacoes, página do site oficial do programa, um repositório de dissertações do PROFMAT organizado pela Sociedade Brasileira de Matemática (SBM). Utilizamos os temas citados a seguir como critério de busca pelo título das dissertações publicadas, abrangendo todas as dissertações de 2013 à 2023. Ao utilizarmos os termos "linguagem de programação" foram encontramos 7 dissertações, utilizando os termos "métodos numéricos" encontramos 15 dissertações, utilizando os termos "cálculo numérico" encontramos 2 dissertações, utilizando os termos "sistemas lineares" encontramos 70 dissertações e utilizando os termos "sistemas não lineares" não encontramos nenhuma dissertação.

Prosseguindo com a seleção das dissertações a serem analisadas, de acordo com os critérios mencionados anteriormente, optamos por analisar 4 das 7 dissertações encontradas sobre o tema "Linguagem de Programação", 3 das 15 dissertações sobre o tema "Métodos Numéricos", 1 das 2 dissertações sobre "Cálculo Numérico" e 2 das 70 dissertações sobre

"Sistemas Lineares", sendo essas escolhidas de acordo com os critérios mencionados. Na tabela abaixo estão dispostas as dissertações escolhidas e posteriormente suas análises.

Tabela 3.1: Dissertações analisadas do PROFMAT.

Ano	Autor(a)	Título	Instituição
2023	J. E. Souza	O Uso da Linguagem de Programação Python na Resolução de Problemas Matemáticos do Ensino Médio	UECG
2023	W. R. Faustino	Sistemas Lineares Homogêneos de Recorrências Lineares de Primeira Ordem com Duas e Três Variáveis	UECG
2022	J. R. Carvalho	Calculo Numérico de Raízes e sua Aplicação no Ensino Básico	CEFET
2022	J. F. Veronese	Métodos Numéricos de Zero de Funções Aplicados em Problemas de Taxa Interna de Retorno	UTFPR
2022	J. V. T. Cotrim	Sistemas Lineares e Matrizes em Planilhas Eletrônicas	UESB
2021	L. S. Vaz	Relações Métricas em Triângulos Retângulos utilizando a Linguagem de Programação Scratch: Uma Abordagem para Atividades	UTFPR
2019	J. C. O. Marques	Construção de Mosaicos Utilizando a Linguagem de Programação Scratch como Ferramenta para o Ensino de Geometria Plana	UTFPR
2019	S. M. O. Riboldi	A Linguagem de Programação Scratch e o Ensino de Funções: Uma Possibilidade	UFFS
2018	T. M. S. Neto	Ajuste de Curvas Usando Métodos Numéricos	UFG
2016	S. D. Corrêa	O Uso de Métodos Numéricos em Problemas de Otimização: Aplicações No Ensino Médio	UNICAMP

Fonte: o própio autor.

João Evayr de Souza [18], em seu trabalho "O Uso da Linguagem de Programação Python na Resolução de Problemas Matemáticos do Ensino Médio", foca na introdução à programação aplicada ao ensino de Matemática. Ele inicia abordando o uso da linguagem Python, a matemática envolvida e os referenciais curriculares da BNCC. O autor apresenta uma explanação sobre as vantagens e desvantagens de Python, além de instruções de instalação e uma introdução aos conceitos básicos da linguagem, como variáveis e operadores. Souza também discute o uso do Python tanto em computadores, com IDEs como PyCharm e Spyder, quanto em dispositivos móveis, finalizando com exemplos de problemas que utilizam a linguagem para resolver tópicos específicos do Ensino Médio.

Wellington Rodrigues Faustino [8], em seu trabalho "Sistemas Lineares Homogêneos

de Recorrências Lineares de Primeira Ordem com Duas e Três Variáveis", aborda o estudo de sistemas lineares. Ele inicia apresentando os conceitos fundamentais de autovalores, autovetores, semelhança, diagonalização de matrizes e a matriz de Jordan. Em seguida, explora as recorrências lineares de primeira ordem e os sistemas homogêneos de primeira ordem com coeficientes constantes. No Capítulo 4, o autor resolve 10 problemas matemáticos sobre esses tópicos, e no capítulo final, amplia o estudo para recorrências lineares de segunda e terceira ordem, concluindo com exemplos e considerações finais.

Júlio Ribeiro de Carvalho [5], em seu trabalho "Cálculo Numérico de Raízes e sua Aplicação no Ensino Básico", foca em métodos numéricos iterativos. O autor começa com os conceitos básicos de continuidade e derivadas, seguido da apresentação de métodos clássicos para encontrar zeros de funções, como a bisseção, o ponto fixo e o método de Newton. Ele também discute o pensamento computacional no ensino da matemática, referenciando a BNCC e o uso de ferramentas informatizadas. Carvalho encerra com uma sequência didática aplicada em sala, utilizando planilhas eletrônicas para resolver problemas matemáticos com os métodos da bisseção e de Newton.

Jessica Fernandes Veronese [20], em seu trabalho "Métodos Numéricos de Zero de Funções Aplicados em Problemas de Taxa Interna de Retorno", explora métodos numéricos iterativos. A autora começa com um embasamento teórico, demonstrando importantes teoremas de Cálculo Diferencial e Integral, como o Teorema dos Intervalos Encaixantes, o Teorema de Bolzano e o Teorema de Rolle. Em seguida, descreve métodos numéricos para encontrar zeros de funções, como o Método da Bisseção, o Método do Ponto Fixo, o Método de Newton-Raphson e o Método da Secante, apresentando os algoritmos e critérios de parada de cada um. Ela também realiza uma comparação prática entre os métodos de Bisseção, Newton-Raphson e Secante, aplicando-os a problemas de taxa interna de retorno.

João Vitor Teixeira Cotrim [7], em seu trabalho "Sistemas Lineares e Matrizes em Planilhas Eletrônicas", aborda o uso de matrizes e sistemas lineares. O autor começa apresentando conceitos fundamentais sobre esses temas e discute a importância da tecnologia na educação, destacando o Excel como uma ferramenta útil, sua evolução e suas aplicações na matemática. Além disso, ele implementa uma oficina em sala de aula onde os alunos utilizam o Excel para resolver problemas matemáticos relacionados a matrizes e sistemas lineares.

Lucas Dos Santos Vaz [19], em seu trabalho "Relações Métricas em Triângulos Retângulos utilizando a Linguagem de Programação Scratch: Uma Abordagem para Atividades", explora a introdução à programação. O autor começa com uma breve história da geometria, desde suas origens até a geometria contemporânea, passando por períodos como o Egito e a Mesopotâmia. Em seguida, apresenta um embasamento teórico sobre

geometria plana, abordando congruência e semelhança de triângulos, além de discutir a tecnologia da informação e comunicação na educação. Por fim, oferece uma lista de 8 atividades a serem realizadas com o Scratch, que inclui a criação de apresentações, medições de segmentos, construção de ângulos e deduções sobre relações métricas em triângulos retângulos.

Jéssica Cristina De Oliveira Marques [11], em "Construção de Mosaicos Utilizando a Linguagem de Programação Scratch como Ferramenta para o Ensino de Geometria Plana", também aborda a introdução à programação. A autora inicia com uma breve história da geometria plana e a arte dos mosaicos, seguido de um embasamento teórico sobre geometria plana, especialmente polígonos. Em seguida, apresenta uma lista de 9 atividades para serem feitas com o Scratch, que incluem a construção de ângulos e polígonos, a descoberta da soma dos ângulos internos de um triângulo, a dedução da fórmula para a soma dos ângulos internos de um polígono convexo e a criação de mosaicos com polígonos regulares.

Sandra Mara Oselame Riboldi [15], em "A Linguagem de Programação Scratch e o Ensino de Funções: Uma Possibilidade", foca na introdução à programação. A autora inicia sua fundamentação teórica discutindo o construcionismo de Papert e a teoria da aprendizagem significativa de Ausubel, além de abordar o pensamento computacional e a Base Nacional Comum Curricular (BNCC). No desenvolvimento do trabalho, apresenta uma série de atividades desenvolvidas em sala de aula com o uso do Scratch em problemas relacionados a funções. Ela realiza uma pesquisa sobre o desenvolvimento dos alunos em questões matemáticas antes e depois da aplicação de uma série de atividades envolvendo o Scratch no ensino e aprendizagem de matemática, conduzida com uma turma do 9° ano do Ensino Fundamental em uma escola pública estadual de Santa Catarina, caracterizada como pesquisa-ação. Segundo a autora, o objetivo principal é utilizar o Scratch para desenvolver o pensamento computacional, introduzindo conceitos de funções, como lei de formação, domínio e imagem, valor numérico e gráficos de funções, o que representa um diferencial nesta pesquisa.

Theófilo Machado de Sousa Neto [12], em "Ajuste de Curvas Usando Métodos Numéricos", aborda o uso de métodos numéricos. O autor inicia discutindo ajuste polinomial e ajuste por mínimos quadrados no caso linear, apresentando critérios matemáticos que justificam a escolha da curva que melhor se adequa aos pontos dados. Para o caso não linear, ele utiliza o método de Gauss-Newton para determinar os coeficientes da equação proposta, além de abordar a interpolação polinomial com o método de Newton e o método de Lagrange, entre outros. Como aplicação prática, foi proposta uma atividade para uma turma do terceiro ano do ensino médio, visando determinar a resistência elétrica de uma lâmpada incandescente de 70W/220V. O autor finaliza apresentando definições sobre o

sistema de aterramento e uma breve introdução a alguns dos métodos mais utilizados para determinar a resistividade do solo.

Sonia Dourado Corrêa [6], em "O Uso de Métodos Numéricos em Problemas de Otimização: Aplicações no Ensino Médio", também foca no uso de métodos numéricos. A autora inicia seu trabalho estudando pontos de máximos e mínimos em funções quadráticas e não lineares de uma única variável. Para isso, discute limites e derivadas, além de demonstrar teoremas importantes, como o teorema de Weierstrass para valores extremos e o teorema de Rolle, analisando o crescimento e o decrescimento de funções. A autora então inicia a discussão sobre problemas de otimização e explora métodos numéricos para encontrar máximos e mínimos de funções reais. Segundo a própria autora, "a utilização de métodos numéricos parece ser uma boa estratégia para se encontrar máximos e mínimos de funções reais no Ensino Médio, uma vez que são de fácil compreensão e utilizam tecnologia (computadores, planilhas eletrônicas)". Ela finaliza com a apresentação de atividades e problemas de otimização com resolução numérica para serem realizados em sala de aula.

Na segunda parte de nossa pesquisa, fizemos uma busca no Catálogo de Teses e Dissertações da CAPES, disponível em catalogodeteses.capes.gov.br/catalogo-teses. Dissertações oriundas de programas de mestrados profissionais ofertados por instituições públicas e privadas. Utilizando os termos "linguagem de programação", "métodos numéricos" e "sistemas não lineares", foram encontradas várias dissertações, sobre os dois primeiros temas, mas nenhuma dissertação a respeito de sistemas não lineares com o filtro em Educação. Conforme os critérios mencionados anteriormente, optamos por examinar duas das dissertações encontradas no catálogo e uma terceira dissertação que não consta no catálogo da CAPES, porém seu conteúdo está alinhado com o tema aqui abordado. Dessa forma, temos um total de três dissertações, todas listadas na tabela a seguir, juntamente com suas respectivas análises que serão apresentadas posteriormente.

Tabela 3.2: Dissertações analisadas do catálogo da CAPES e outros.

Ano	Autor(a)	Título	Instituição
2020	J. S. Rocha	A Implementação da Linguagem de Programação na Educação Escolar Utilizando o Scratch	UFR
2019	G. M. Pesente	Ensino da Matemática por meio da Linguagem de Programação Python	UTFPR
2015	E. A. Souza	Métodos Iterativos para Problemas Não Lineares	UFF

Fonte: o própio autor.

Jaine Souza da Rocha [16], em "A Implementação da Linguagem de Programação

na Educação Escolar Utilizando o Scratch", investiga os limites e as possibilidades da implementação da linguagem de programação na educação escolar por meio do software Scratch. O autor inicia o trabalho com um enfoque metodológico baseado no estudo de campo, cuja empíria foi obtida por meio de uma oficina de Scratch realizada com alunos do quinto ano do ensino fundamental em uma escola pública de Santarém, Pará. Complementarmente, foram aplicados questionários e entrevistas com os alunos participantes e a professora responsável pela sala informatizada. Como resultado, o autor destaca que o Scratch se apresenta como uma maneira produtiva de introduzir a linguagem de programação na educação, permitindo que os alunos tenham uma experiência ativa na construção do conhecimento, abrangendo tanto a informática e a computação quanto os conteúdos curriculares de matemática.

Guilherme Morais Pesente[14], em "Ensino da Matemática por meio da Linguagem de Programação Python". Nesse trabalho, voltado para a introdução da programação, o autor mostra que problemas na aprendizagem da matemática são realidades presentes em diversas escolas. No decorrer dos anos, de 2013 a 2017, por meio da avaliação do Índice de Desenvolvimento da Educação Básica (IDEB), de acordo com o autor, este fato foi constatado e, por meio dele foi demonstrado o baixo aproveitamento dos alunos nos anos iniciais. O autor buscou investigar, através de um projeto feito por um grupo de 10 alunos do sexto ano do ensino fundamental II, na disciplina de matemática, em uma escola particular do município de Ponta Grossa, no Estado do Paraná, os impactos de se utilizar linguagem de programação Python para complementar os conhecimentos obtidos nos conteúdos em sala de aula. Para a execução do projeto, ainda de acordo com o autor, foi efetuado um levantamento dos problemas apresentados pelos alunos na disciplina de matemática, sendo posteriormente desenvolvidos os códigos para cada conteúdo. Tal levantamento foi obtido por meio de questionários respondidos pela professora da disciplina de matemática. Ao final do projeto, foi desenvolvido uma avaliação processual, visando mensurar o aproveitamento, avanços e melhorias dos alunos no presente trabalho, sendo possível obter significantes resultados da utilização da linguagem Python no processo de desenvolvimento cognitivo. Após essas constatações, o autor afirma que foi possível comprovar que a utilização da linguagem Python como meio de complemento, contribui para o melhor desenvolvimento educacional e lógico dos alunos.

Elienai Alves de Souza[17], em "Métodos Iterativos para Problemas Não Lineares". Nesse trabalho, voltado para métodos numéricos iterativos, sendo um trabalho puramente científico e não voltado para o ensino, o autor apresenta alguns métodos iterativos clássicos e outros mais recentes para a resolução de sistemas de equações não lineares. Apresenta-se também algumas definições básicas como bacias de atração, raio de convergência local,

entre outras, e uma introdução à teoria fractal, reunindo assim informações e ferramentas para a análise dos métodos iterativos apresentados. Técnicas de aceleração de convergência para métodos iterativos estão também presentes neste trabalho, mas apenas para fins informativos, sendo o objetivo geral deste trabalho o estudo comparativo entre os métodos iterativos apresentados, mediante a análise das imagens das bacias de atração, gráficos e tabelas de resultados numéricos obtidos. Os métodos iterativos estão implementados na linguagem MATLAB®, e estes são aplicados a quatro diferentes sistemas não lineares cujas soluções já são previamente conhecidas. Além dos métodos iterativos, estão implementados algoritmos para gerar as imagens das bacias de atração e de seus respectivos conjuntos de Julia, para assim ser possível calcular o raio de convergência local referente a cada raiz e a dimensão fractal de cada conjunto de Julia pelo método Box-Counting. Desta forma, os dados obtidos são organizados em imagens, tabelas e gráficos para a realização da análise comparativa, a fim de avaliar a convergência e a eficiência temporal de cada método, destacando-se, dentre os métodos avaliados, os métodos da Homotopia e Continuação e o método de Newton.

Após a análise dos trabalhos produzidos, fica evidente a ligação entre o ensino de matemática e a integração com a tecnologia, destacando-se a programação e ferramentas computacionais como recursos pedagógicos. Os autores exploram diferentes abordagens, desde a introdução à linguagem Python até o uso de softwares como Scratch e Excel, além de empregarem métodos numéricos e recursos computacionais para a resolução de problemas específicos. Essa abordagem visa, não apenas facilitar a compreensão dos conceitos matemáticos, mas também podem promover uma aprendizagem mais prática e aplicada. Em vários trabalhos, observa-se também um alinhamento com a BNCC, contribuindo para uma educação matemática mais contextualizada e alinhada com as necessidades atuais dos estudantes.

É importante ressaltar que, dentre as pesquisas realizadas na plataforma do PROFMAT e no catálogo da CAPES, não encontramos dissertações específicas sobre O tema 'Sistemas Não Lineares' voltadas para o ensino básico. Este fato, embora não surpreendente, considerando a limitada abordagem ou até mesmo a não abordgem dos sistemas não lineares no ensino médio, nos motiva a contribuir para a área com este estudo. Além disso, constatamos que, embora as dissertações contenham sequências didáticas voltadas para professores interessados, nenhuma delas apresentou uma apostila independente abrangendo não apenas uma sequência didática, mas também conteúdo de estudo e aprimoramento dos docentes em relação aos tópicos abordados nas dissertações. Portanto, o propósito deste trabalho é preencher essa lacuna ao criar uma apostila com esse propósito específico.

4 Aritmética Computacional e Erros de Aproximação

Vamos entrar no universo da aritmética utilizando números finitos de dígitos, o que nos leva a esperar resultados precisos das máquinas para operações como $2 + 2 = 4$ ou $4 \times 8 = 32$, porém não para expressões como $(\sqrt{3})^2 = 3$. Devemos estar cientes de que sempre haverá erros decorrentes da utilização de valores aproximados quando realizamos cálculos com números não representados exatamente de forma finita na base 2. O erro que surge quando utilizamos máquinas para efetuar cálculos com números reais é conhecido como erro de arredondamento.A aritmética realizada em uma máquina opera apenas com números que possuem um número finito de dígitos, o que resulta em cálculos que utilizam representações aproximadas dos valores corretos. Esse fenômeno pode ser especialmente relevante em situações que exigem um alto grau de precisão. Portanto, é importante compreender a natureza do erro de aproximação e aplicar técnicas adequadas para mantê-lo sob controle. Controlar esse erro é de extrema importância, principalmente em cenários que envolvam um grande número de cálculos.

Na próxima seção vamos averiguar como são representados os números em uma máquina e vamos estudar com mais detalhes erros de aproximação.

4.1 Número de Máquina e Erro de Aproximação

No contexto da computação numérica, o termo *número de máquina* desempenha um papel importante na determinação da precisão e da confiabilidade dos cálculos executados em sistemas computacionais. De acordo com (BURDEN, 2016)[4], o conceito de número de máquina refere-se à representação de números no ambiente binário de uma máquina, levando em consideração as limitações impostas pelo hardware e pela aritmética de ponto flutuante, um formato de representação de números em computação, que permite representar números reais de forma aproximada. Em 1985, segundo o autor, o IEEE (Institute of Electrical and Electronic Engineers), entidade técnico-profissional, fundada em 1884, no Estados Unidos, responsável pela definição de padrões mundiais para dispositivos elétrico e eletrônicos,

publicou um relatório, atualizado em 2008, fornecendo dentre outros, os padrões para números binários e decimais de ponto flutuante, e são padrões geralmente seguidos por todos os fabricantes de microcomputadores e hardwares de pontos flutuantes. Segundo o relatório, é usado para um número real uma representação de 64 bits (dígitos binários). O primeiro bit é um indicador de sinal (s=0: positivo, s=1: negativo) denotado por s. Ele é seguido por um expoente de 11 bits, c, chamado de **característica**, e por uma fração binária de 52 bits, f, chamada **mantissa**. A base para o expoente é o número 2.

Visto que 52 algarismos binários correspondem a 16 ou 17 algarismos decimais, podemos considerar pelo menos 16 algarismos de precisão na representação de u número nesse sistema. O expoente de 11 algarismos binários fornece uma faixa de 0 a $2^{11} - 1 = 2047$. Para representação adequada de números de módulo pequeno, 1023 é subtraído da **característica**, de maneira que o intervalo do expoente seja na verdade -1023 até 1024. Uma normatização imposta, fornece um número de ponto flutuante da forma

$$(-1)^s 2^{c-1023}(1+f).$$

O menor número positivo normatizado que pode ser representado tem $s = 0$, $c = 1$ e $f = 0$ equivalente a

$$(-1)^0 \cdot 2^{-1022} \cdot (1+0) \approx 0{,}22251 \cdot 10^{-307}.$$

O maior tem $s = 0$, $c = 2046$ e $f = 1 - 2^{-52}$ equivalente a

$$(-1)^0 \cdot 2^{1023} \cdot (2 - 2^{-52}) \approx 0{,}17977 \cdot 10^{309}.$$

Ainda de acordo com (BURDEN, 2016), números que ocorrem em cálculos com módulo menor que $2^{-2022} \cdot (1+0)$ resultam em **underflow** e são geralmente igualados a zero. Números que são maiores do que $2^{1023} \cdot (2 - 2^{-52})$ resultam em **overflow** e normalmente fazem o cálculo parar (a menos que o programa tenha sido projetado para detectar essa ocorrência). Observe que há duas representações para o número zero; um zero positivo quando $s = 0,\ c = 0\ e\ f = 0$ e um zero negativo quando $s = 1,\ c = 0\ e\ f = 0$.

Agora, para um número decimal representado em sua forma de ponto flutuante normatizado, temos

$$\pm\ 0{,}d_1 d_2 ... d_i \times 10^n, \quad 1 \leq d_1 \leq 9 \quad e \quad 0 \leq d_i \leq 9,$$

onde cada $i = 1, 2, \ldots, k$. Os números dessa forma são chamados de números de máquina decimais de k algarismos.

O erro que resulta da substituição de um número por sua forma de ponto flutuante é chamado de erro de arredondamento. A forma em ponto flutuante de y, denotada por $fl(y)$, é obtida terminando a **mantissa** de y em k algarismos decimais. Há duas maneiras de realizar isso; um método chamado truncamento é simplesmente cortar os algarismos restantes (d_{k+1},d_{k+2}, ...), isso produz

$$fl(y) = 0{,}d_1{,}d_2, \ldots, d_k \times 10^n.$$

e o outro chamado arredondamento, adiciona $5 \cdot 10^{n-(k+1)}$ em y e então trunca o resultado para obter

$$fl(y) = 0{,}\alpha_1{,}\alpha_2, \ldots, \alpha_k \times 10^n.$$

Desse modo, ao arredondar, se $d_{k+1} > 5$, adiciona 1 a d_k para obter $fl(y)$, isto é, arredonda para cima. Quando d_{k+1} é menor do que 5, então simplesmente trunca mantendo apenas os primeiros k algarismos, logo arredondamento para baixo.

A definição a seguir descreve os métodos para medir erros de aproximação.

Definição 4.1: Ao supor que P é uma aproximação para P^*, o erro real será $P^* - P$, o erro absoluto será $|P^* - P|$ e o erro relativo será

$$\frac{|P^* - P|}{|P^*|},$$

com $P \neq 0$.

A questão da precisão em cálculos numéricos ganha ainda mais relevância à medida que lidamos com algoritmos que dependem da iteração de operações matemáticas. A acumulação do erro de arredondamento ao longo das iterações pode levar a resultados consideravelmente discrepantes do resultado teoricamente esperado.

Em resumo, o erro de arredondamento é uma característica intrínseca das operações aritméticas em máquinas que trabalham com números reais. Reconhecer sua presença e adotar abordagens que reduzam ou controlem esse erro são fundamentais para a obtenção de resultados corretos e precisos em aplicações computacionais que dependem de cálculos numéricos complexos.

Na próxima seção vamos falar sobre algoritmo, que se trata de uma sequência de passos executados para se chegar a uma solução desejada, método utilizado em máquinas para a execução de tarefas específicas, e a estabilidade de um algoritmo, o que está diretamente relacionada com a confiabilidade das soluções obtidas.

4.2 Algoritmos e Estabilidade

Ao longo deste trabalho, abordaremos métodos numéricos descritos por algoritmos, que são procedimentos que apresentam uma sequência finita e precisa de etapas a serem executadas em uma ordem específica. Seu principal propósito é implementar um procedimento que permita solucionar problemas ou aproximar uma solução para determinadas questões. Para o desenvolvimento desses algoritmos, é importante considerar a precisão das aproximações obtidas e os possíveis erros associados. Muitas vezes, a escolha de um algoritmo adequado pode fazer a diferença entre uma solução viável e uma que não atende aos requisitos necessários.

A busca por algoritmos eficientes e confiáveis é um desafio constante, pois o desenvolvimento de novos métodos ou a adaptação de abordagens existentes pode trazer avanços significativos em diversas áreas de estudo. Além disso, é necessário estar ciente das limitações impostas pelos recursos computacionais disponíveis, como poder de processamento e armazenamento, para garantir que os algoritmos sejam viáveis na prática.

Portanto, neste trabalho, buscaremos analisar e compreender diferentes algoritmos de aproximação, considerando suas aplicações, eficácia e precisão.

A estabilidade de um algoritmo também é algo importante a se considerar. Um critério que desejamos impor aos algoritmos, sempre que possível, é a capacidade de lidar com pequenas variações nos dados iniciais, resultando em alterações correspondentes e limitadas nos resultados finais. Um algoritmo que exibe essa propriedade é conhecido como estável; caso contrário, considera-se instável. É relevante mencionar que alguns algoritmos são estáveis apenas sob certas condições ou escolhas específicas de dados iniciais, classificados como condicionalmente estáveis. É importante caracterizar as propriedades de estabilidade dos algoritmos, sempre que for viável fazê-lo.

A estabilidade dos algoritmos está diretamente relacionada à confiabilidade das soluções obtidas. Algoritmos estáveis têm a vantagem de gerar resultados consistentes, mesmo diante de pequenas variações nos dados de entrada, o que os torna altamente desejáveis em diversas aplicações práticas. Por outro lado, algoritmos instáveis podem levar a resultados imprecisos, sua sensibilidade a perturbações mínimas nos dados de entrada pode comprometer a qualidade e a utilidade das soluções encontradas. A compreensão das propriedades de estabilidade dos algoritmos de aproximação permitirá tomar decisões mais informadas na seleção do método mais adequado para solucionar problemas específicos, o que é essencial para a otimização, eficiência e precisão dos resultados obtidos.

5 Métodos Numéricos para Zeros de Funções

Em muitas situações, encontrar soluções exatas para problemas matemáticos é uma tarefa desafiadora, senão impossível. Nesse âmbito, os métodos numéricos desempenham um papel importante, permitindo a obtenção de soluções aproximadas que podem ser muito úteis em diversas aplicações.

Os métodos numéricos para determinar zeros de funções, citados neste capítulo, funcionam por meio de iterações sequenciais e cálculos fundamentados nas características da função. Quando critérios iniciais são atendidos, essas técnicas conduzem a uma aproximação gradual em direção à solução desejada. A escolha do método apropriado depende da natureza da função e das estimativas iniciais.

Na próxima seção, abordaremos a convergência de um método iterativo, ou seja, se o método produz resultados cada vez mais próximos da solução correta à medida que iterações sucessivas são realizadas. Também discutiremos a ordem de convergência do método, que se refere à medida quantitativa da velocidade com que o método se aproxima dessa solução.

5.1 Convergência e Ordem de Convergência

Em métodos numéricos iterativos, que são processos onde $x_k = f(x_{k-1})$, ou seja, o valor atual é calculado a partir do valor anterior através de uma função e a convergência é a propriedade que demonstra a capacidade de um algoritmo produzir resultados cada vez mais próximos da solução verdadeira à medida que iterações sucessivas são realizadas. Vários fatores podem influenciar na convergência de um método numérico iterativo, assegurar sua convergência é um dos principais objetivos dos pesquisadores e desenvolvedores da área. Para isso, é necessário um estudo cuidadoso do problema em questão, a seleção adequada do método numérico mais apropriado e a utilização de critérios de parada eficazes, que interrompem o processo de iteração quando a precisão desejada é alcançada.

A ordem de convergência de um método numérico é uma medida quantitativa

da rapidez com que o método se aproxima da solução precisa à medida que o número de iterações aumenta. Ela é representada por um número real p e descreve a taxa na qual o erro varia com base no número de iterações. Métodos numéricos com ordem de convergência mais alta geralmente alcançam soluções mais precisas em menos iterações.

Para fazer uma definição formal da ordem de convergência de um método numérico, vamos primeiramente definir o que é um zero de uma função.

Definição 5.1: Se $f\colon [a,\ b] \to \mathbb{R}$ é uma função dada, um ponto $x^* \in [a,\ b]$ é um zero da função se $f(x^*) = 0$.

A ordem de convergência pode ser definida como se segue.

Definição 5.2: Seja x_k o resultado da aplicação de um método iterativo na iteração k e $e_k = x_k - x^*$ o seu erro, onde x^* é o zero da função. Se existirem $p \geq 1$ e uma constante $c > 0$ tais que

$$\lim_{k\to\infty} \frac{|e_{k+1}|}{|e_k|^p} = c,$$

então p é chamado de ordem de convergência e c é a constante de proporção.

Sendo assim, quanto maior o valor de p, mais rapidamente o método se aproxima do zero da função. Quando um método tem ordem de convergência $p = 1$, sua velocidade de convergência é denominada linear, quando $1 < p < 2$, sua ordem de convergência é denominada superlinear e quando $p = 2$, sua ordem de convergência é denominada quadrática. Métodos de alta ordem de convergência são altamente desejáveis, pois podem economizar tempo computacional e recursos, especialmente ao lidar com problemas complexos.

Nas próximas seções vamos apresentar um estudo sobre os métodos denominados por Método da Bisseção, Método do Ponto Fixo, Método de Newton-Raphson e Método da Secante, que são métodos numéricos iterativos clássicos para a busca de soluções de equações não lineares de uma variável, tanto quanto suas demonstrações de convergência e suas ordens de convergência, sob critérios iniciais.

5.2 Método da Bisseção

O Método da Bisseção trata de uma técnica usada para encontrar um zero de uma função contínua em um intervalo pré-estabelecido através de um processo de iteração. O intervalo deve ser selecionado de forma que haja pelo menos um zero da função nele contido. O Método da Bisseção se baseia no Teorema de Bolzano, apresentado formalmente

a seguir.

Teorema 5.3 (Teorema de Bolzano): Se uma função f é contínua em um intervalo fechado $[a,\ b]$ e $f(a)f(b) < 0$, então existe pelo menos um valor $x^* \in (a, b)$ tal que $f(x^*) = 0$.

No processo de iteração do Método da Bisseção, o intervalo contendo um zero da função é dividido pela metade, selecionando o subintervalo onde a função muda de sinal, fundamentado no Teorema de Bolzano 5.3, aproximando-se cada vez mais do zero da função. O processo continua até que a precisão desejada seja alcançada.

Vamos apresentar a fórmula de iteração do Método da Bisseção. Suponha que f é uma função contínua definida no intervalo $[a,\ b]$ tal que $f(a)f(b) < 0$. Pelo Teorema de Bolzano 5.3 temos pelo menos um $x^* \in (a, b)$ onde $f(x^*) = 0$. Vamos dividir o intervalo ao meio e o ponto médio do intervalo será nossa primeira aproximação x_1. Assim sendo, temos

$$x_1 = \frac{a_1 + b_1}{2},$$

onde a_1 e b_1 são os extremos do intervalo inicial. Selecionando agora o subintervalo onde a função muda de sinal nos extremos, faremos a segunda iteração, que será

$$x_2 = \frac{a_2 + b_2}{2},$$

onde a_2 e b_2 são os extremos do novo intervalo escolhido. Assim sendo, temos a fórmula de iteração do Método da Bisseção, que é dado por

$$x_k = \frac{a_k + b_k}{2},$$

onde a_k e b_k são os extremos do intervalo escolhido na iteração k.

A convergência do Método da Bisseção é formalmente apresentado pelo teorema a seguir.

Teorema 5.4 (Convergência do Método da Bisseção): Se f é uma função contínua definida no intervalo $[a,\ b]$ tal que $f(a)f(b) < 0$, então o Método da Bisseção gera uma sequência de intervalos que converge para um ponto x^* onde $f(x^*) = 0$.

Para a demonstração do teorema de convergência acima, vamos analisar o erro obtido a cada iteração. O erro na iteração k é definido como sendo

$$e_k = x_k - x^*,$$

onde x_k é a aproximação na iteração k e x^* é o zero exato da função. Se x^* está no intervalo (a_k, b_k) então

$$a_k \leq x^* \leq b_k.$$

Assim o erro na primeira iteração, dado por e_1, será limitado por

$$|e_1| \leq \frac{b_1 - a_1}{2},$$

e como a_1 e b_1 são os extremos iniciais, temos que

$$|e_1| \leq \frac{b - a}{2}.$$

Agora, o erro na segunda iteração, dado por e_2, será limitado por

$$|e_2| \leq \frac{b_2 - a_2}{2} = \frac{\frac{b-a}{2}}{2} = \frac{b - a}{2^2}.$$

Assim temos que o erro na iteração k, dado por e_k, será limitado por

$$|e_k| \leq \frac{b - a}{2^k}.$$

Ao aplicar limite com k tendendo ao infinito, temos que

$$\lim_{k\to\infty} |e_k| = \lim_{k\to\infty} \frac{b - a}{2^k} = 0.$$

Assim, quando o número de iterações aumenta, o erro tende para zero, demonstrando a convergência do método.

Para determinar a ordem de convergência do Método da Bisseção, precisamos mostrar que, para cada iteração, a diferença entre a aproximação atual e o zero da função é reduzida por um fator constante. Se o valor real x^* está contido no intervalo $[a_k,\ b_k]$, sabemos que

$$a_k \leq x^* \leq b_k,$$

portanto, o erro absoluto na próxima iteração $k + 1$, dado por e_{k+1}, será limitados por

$$|e_{k+1}| \leq \frac{b_{k+1} - a_{k+1}}{2} = \frac{\frac{b_k - a_k}{2}}{2} = \frac{|e_k|}{2}.$$

Chegamos à conclusão que, para k suficientemente grande, o erro absoluto na iteração $k + 1$ será a metade do erro absoluto na iteração k. Dividindo ambos os membros

por $|e_k|$ temos

$$\frac{|e_{k+1}|}{|e_k|} \leq \frac{1}{2}.$$

Isso mostra que a ordem de convergência do Método da Bisseção é $p = 1$ tendo velocidade de convergência linear com uma constante de proporção $c = \frac{1}{2}$.

Agora vamos estimar a quantidade de iterações necessárias para atingir uma aproximação desejada pré-estabelecida.

Suponha que o intervalo inicial seja $[a, b]$ e que após k iterações, o intervalo seja $[a_k, b_k]$. Como o intervalo é dividido ao meio a cada iteração, temos

$$b_k - a_k = \frac{b-a}{2^k}.$$

Para alcançar uma precisão $\epsilon > 0$, queremos que o comprimento do intervalo seja menor que ϵ, ou seja,

$$\frac{b-a}{2^k} < \epsilon.$$

Rearranjando, temos

$$2^k > \frac{b-a}{\epsilon}.$$

Tomando o logaritmo de ambos os lados, verificamos que

$$k \log(2) > \log\left(\frac{b-a}{\epsilon}\right),$$

concluindo então que

$$k > \frac{\log\left(\frac{b-a}{\epsilon}\right)}{\log(2)}.$$

Isso nos dá o número mínimo de iterações necessárias para alcançar a precisão desejada. Essa demonstração estabelece uma relação entre a precisão desejada ϵ, o intervalo inicial $[a,\ b]$ e o número de iterações k necessário para alcançar essa precisão.

Vamos ver um exemplo de aplicação do Método da Bisseção para encontrar um zero aproximado da função $f(x) = x^3 - 3x - 5$.

Exemplo 5.2.1: Considere a função $f(x) = x^3 - 3x - 5$. Como $f(0)f(3) < 0$, pelo Teorema de Bolzano 5.3, temos pelo menos um zero de $f(x)$ no intervalo $[0,\ 3]$.

A seguir, aplicamos o Método da Bisseção por 4 iterações, registrando os valores de x_i, $f(x_i)$, e os intervalos:

Iteração	Intervalo	x_i	$f(x_i)$	Intervalo Atualizado
1	$[0; 3]$	$1; 5$	$-6{,}125$	$[1{,}5; 3]$
2	$[1{,}5; 3]$	$2{,}25$	$-0{,}359$	$[2{,}25; 3]$
3	$[2{,}25; 3]$	$2{,}625$	$6{,}337$	$[2{,}25; 2{,}625]$
4	$[[2{,}25; 2{,}625]$	$2{,}437$	$2{,}061$	$[2{,}25; 2{,}437]$

Após essas iterações, a sequência se aproxima do zero da função, que é $x \approx 2{,}279018$.

Vamos agora verificar quantas iterações seriam necessárias para atingir uma aproximação pré-estabelecida desejada. Queremos encontrar uma solução para a equação não linear $f(x) = 0$ com precisão $\epsilon = 0{,}001$. Suponha que escolhemos o intervalo inicial $[0, 3]$, teremos

$$n > \frac{\log\left(\frac{3-0}{0{,}001}\right)}{\log(2)} \approx 11{,}55.$$

Isso significa que seriam necessárias pelo menos 12 iterações para alcançar uma precisão de 0,001.

O Método da Bisseção, embora lento se comparado a outros métodos, é confiável e garante convergência para um zero de função que satisfaz critérios iniciais e a análise da ordem de convergência nos ajuda a entender o ritmo de aproximação e a estimar o número de iterações necessárias.

5.3 Método do Ponto Fixo

O Método do Ponto Fixo é uma técnica utilizada para aproximar soluções de equações não lineares. Dizemos que um ponto c é um ponto fixo de uma função ϕ se a aplicação da função ϕ sobre c resulta no próprio c. Temos a definição de ponto fixo como se segue.

Definição 5.5: Dizemos que c é um ponto fixo de uma função ϕ se $\phi(c) = c$.

Dado o problema de encontrar uma solução para a equação não linear $f(x) = 0$, o Método do Ponto Fixo busca uma função auxiliar $\phi(x)$ de tal forma que a solução da equação original seja equivalente a encontrar um ponto fixo da função $\phi(x)$, ou seja, $x = \phi(x)$. O processo iterativo então começa com uma estimativa inicial x_0 e calcula iterações subsequentes usando a relação

$$x_{k+1} = \phi(x_k) \tag{5.1}$$

onde x_{k+1} é a próxima estimativa da solução e x_k é a estimativa atual. A sequência de iterações deve convergir para a solução real da equação original, desde que certas condições sejam satisfeitas.

De modo geral, $\phi(x)$ pode ser expressada como

$$\phi(x) = x + A(x)\ f(x).$$

Os critérios de convergência do Método do Ponto Fixo estão relacionados às propriedades da função $\phi(x)$ no qual deve ser diferenciável em uma vizinhança do ponto fixo. Além disso a derivada da função auxiliar $\phi(x_k)$ em relação a x deve ser tal que $|\phi'(x)| < 1$ em todo o intervalo $[a,\ b]$. Isso garante que o método irá convergir. A escolha do ponto inicial x_0 também é relevante. Um ponto inicial próximo à solução desejada pode acelerar a convergência, enquanto um ponto inicial distante pode levar a convergência lenta ou até mesmo a divergência.

Satisfeitos os critérios mencionados, a convergência do Método do Ponto Fixo pode ser estabelecida através do Teorema de Convergência do Método do Ponto Fixo a seguir.

Teorema 5.6: Seja $\phi(x)$ uma função contínua, com derivadas primeira e segunda contínuas num intervalo fechado I da forma $I = (x^* - h, x^* + h)$, cujo centro x^* é solução de $x = \phi(x)$. Seja $x_0 \in I$ e M um limitante da forma $|\phi'(x)| \leq M < 1$ em I. Então:

1. a iteração $x_{k+1} = \phi(x_k)$, $k = 0, 1, \ldots$, pode ser executada indefinidamente, pois $x_k \in I$, $\forall k$.
2. $|x_k - x^*| \to 0$.
3. Se $\phi'(x^*) \neq 0$ ou $\phi'(x^*) = 0$ e $\phi''(x^*) \neq 0$ e se $|x_0 - x^*|$ for suficientemente pequeno, então a sequência $x_1, x_2, \ldots$ será monotônica ou oscilante.

Vamos agora demonstrar o Teorema 5.6, segundo (FRANCO, 2014) [9], que estabelece condições sob as quais uma função contínua, com derivadas contínuas, possui um ponto fixo no intervalo considerado e descreve as propriedades da sequência gerada pelo processo iterativo.

1. Usaremos indução para provar que $x_k \in I$, $\forall k$.
   (a) Por hipótese $x_0 \in I$.
   (b) Supomos que $x_0, x_1, \ldots, x_k \in I$.
   (c) Provemos que $x_{k+1} \in I$. Temos:

$$x_{k+1} - x^* = \phi(x_k) - \phi(x^*).$$

Usando o Teorema do Valor Médio, obtemos:

$$x_{k+1} - x^* = \phi'(\xi_k)(x_k - x^*),$$

onde ξ_k está entre x_k e x^*. Tomando módulo, segue que:

$$|x_{k+1} - x^*| = |\phi'(\xi_k)||x_k - x^*| \leq M|x_k - x^*|,$$

desde que pela hipótese de indução $x_k \in I \implies \xi_k \in I$ e sobre I, $|\phi'(x)| \leq M < 1$. Assim:

$$|x_{k+1} - x^*| \leq M|x_k - x^*|.$$

Como $M < 1$, temos que $x_{k+1} \in I$.

2. Pelo item (a) temos que:

$$|x_k - x^*| \leq M|x_{k-1} - x^*| \leq M^2|x_{k-2} - x^*| \leq \ldots \leq M^k|x_0 - x^*|.$$

Como $M < 1$, passando ao limite, obtemos:

$$\lim_{k\to\infty} M^k \to 0 \quad \text{e portanto} \quad |x_k - x^*| \to 0.$$

3. Aqui dividiremos a prova em duas partes. Assim:
   (a) Seja $\phi'(x^*) \neq 0$. Pelo Teorema da Permanência do Sinal temos que numa vizinhança de x^* suficientemente pequena, $\phi'(x)$ terá o mesmo sinal. Assim de:

$$x_{k+1} - x^* = \phi'(\xi_k)(x_k - x^*),$$

temos que:

- Se $\phi'(x^*) > 0$ e

$$\begin{cases} x_k \leq x^* \implies x_{k+1} \leq x^* \\ x_k \geq x^* \implies x_{k+1} \geq x^* \end{cases}$$

- Se $\phi'(x^*) < 0$ e

$$\begin{cases} x_k \leq x^* \implies x_{k+1} \geq x^* \\ x_k \geq x^* \implies x_{k+1} \leq x^* \end{cases}$$

Como $|x_k - x^*| \to 0$, a convergência será monotônica em (I) e em (II) será oscilante em torno de x^*.

(b) Seja $\phi'(x^*) = 0$ e $\phi''(x^*) \neq 0$. Usando o Teorema do Valor Médio, temos que:

$$\phi'(\xi_k) = \phi'(\xi_k) - \phi'(x^*) = \phi''(\zeta_k)(\xi_k - x^*),$$

onde ζ_k está entre ξ_k e x^*. Assim:

$$x_{k+1} - x^* = \phi''(\zeta_k)(\xi_k - x^*)(x_k - x^*).$$

Pelo Teorema da Permanência do Sinal, $\phi''(x)$ terá o mesmo sinal numa vizinhança suficientemente pequena de x^*. Como $(\xi_k - x^*)(x_k - x^*) \geq 0$, pois ξ_k e x_k encontram-se do mesmo lado de x^*, segue que, se:

$$\begin{cases} \phi''(x^*) > 0 \implies x_{k+1} \geq x^*, \forall k, \\ \phi''(x^*) < 0 \implies x_{k+1} \leq x^*, \forall k. \end{cases}$$

Neste caso, a sequência $x_1, x_2, \ldots$ será monotônica independente do sinal de $x_0 - x^*$. Isso completa a prova do Teorema 5.6.

A velocidade de convergência desse método é influenciada pelo valor absoluto da derivada $\phi'(x)$ em torno da solução x^*. Quanto mais próximo de 0 for esse valor, mais rápida será a convergência, portanto a ordem de convergência do Método do Ponto Fixo é dependente das características intrínsecas da função e do ponto inicial. Porém, podemos verificar que, do Teorema 5.6, temos

$$x_{k+1} - x^* = \phi'(z_k)(x_k - x^*),$$

onde z_k está entre x_k e x^*. Então,

$$\frac{|x_{k+1} - x^*|}{|x_k - x^*|} \leq |\phi'(z_k)| \leq M.$$

Assim, a Definição 5.2 está satisfeita com $p = 1$ e $c = M$, ou seja, a ordem de convergência é linear. Daí o nome de Método Iterativo Linear. Além disso, o erro em qualquer iteração é proporcional ao erro na iteração anterior, sendo que o fator de proporcionalidade é $\phi'(z_k)$.

Vamos ver um exemplo para melhor compreensão do funcionamento do método.

Exemplo 5.3.1: Para ilustrar o Método do Ponto Fixo, consideremos a equação $x^3 - 3x - 5 = 0$ e a função auxiliar $\phi(x) = \sqrt[3]{3x + 5}$. Começaremos com uma estimativa inicial $x_0 = 2$ e ao realizar 4 iterações, temos:

$$x_1 = \phi(x_0) = \sqrt[3]{3 \cdot 2 + 5} \qquad \approx 2{,}223980$$

$$x_2 = \phi(x_1) = \sqrt[3]{3 \cdot 2{,}2239 + 5} \approx 2{,}268372$$
$$x_3 = \phi(x_2) = \sqrt[3]{3 \cdot 2{,}2683 + 5} \approx 2{,}276967$$
$$x_4 = \phi(x_3) = \sqrt[3]{3 \cdot 2{,}28943 + 5} \approx 2{,}278623.$$

Continuando com mais iterações, a sequência de valores se aproxima da solução da equação, que é aproximadamente $x \approx 2{,}279018$.

Agora vamos isolar x de outra forma. Considere a equação não linear $x^3 - 3x - 5 = 0$. Podemos reorganizá-la como $x = \frac{x^3-5}{3}$, o que nos permite aplicar o Método do Ponto Fixo com $\phi(x) = \frac{x^3-5}{3}$. Escolhendo uma estimativa inicial $x_0 = 2$, fazendo 4 iterações sucessivas conforme a relação $x_{n+1} = \phi(x_n)$, temos

$$x_1 = \phi(x_0) = \frac{2^3 - 5}{3} = 1{,}000000$$
$$x_2 = \phi(x_0) = \frac{1^3 - 5}{3} = -1{,}333333$$
$$x_3 = \phi(x_0) = \frac{(-1{,}333333)^3 - 5}{3} \approx -2{,}456790$$
$$x_4 = \phi(x_0) = \frac{(-2{,}456790)^3 - 5}{3} \approx -6{,}609578.$$

Podemos observar que o processo iterativo diverge da solução da equação. Isso se dá porque existem pontos no intervalo onde a derivada da função auxiliar é maior que 1, descumprindo um dos critérios de convergência do método.

No Método do Ponto Fixo para encontrar soluções de equações não lineares, sua eficácia depende da escolha adequada da função auxiliar e do ponto inicial. A demonstração de convergência e a análise da velocidade de convergência fornecem informações sobre a confiabilidade e a eficiência do método. Embora seja um método relativamente simples, tem uma importância teórica relevante pois através deste, outras técnicas se desenvolveram, como por exemplo, o Método de Newton-Raphson, que será abordado logo a seguir.

5.4 Método de Newton-Raphson

O Método de Newton-Raphson é uma técnica amplamente utilizada para encontrar soluções de equações não lineares. Ele é um método iterativo que se baseia na aproximação linear da função em torno de um ponto estimado e utiliza essa aproximação para obter uma melhor estimativa para a solução.

Vamos desenvolver o Método de Newton-Raphson a partir do Método do Ponto Fixo. Para escrever tal método, considere a equação

$$\phi(x) = x + A(x)\ f(x), \quad \text{com } f'(x) \neq 0.$$

A função $A(x)$ deve ser escolhida de tal forma que $A(x^*) \neq 0$, com x^* sendo um zero de f. Como é um método de ponto fixo, temos garantia de convergência se $|\phi'(x)| < 1$ para x em alguma vizinhança de $x^* \in I$. Assim, se escolhermos $A(x)$ tal que $\phi'(x^*) = 0$, teremos $|\phi'(x)| < 1$, garantindo então a convergência do método.

Vamos agora mostrar os passos para encontrar um processo de iteração que será denominado Método de Newton-Raphson. Derivando $\phi(x) = x + A(x)\ f(x)$ em relação a x, temos

$$\phi'(x) = 1 + A'(x)\ f(x) + A(x)\ f'(x).$$

Fazendo $x = x^*$, temos $f(x^*) = 0$, logo

$$\phi'(x^*) = 1 + A(x^*)\ f'(x^*).$$

Colocando $\phi'(x^*) = 0$, temos $0 = 1 + A(x^*)\ f'(x^*)$, o que nos dá

$$A(x^*) = \frac{-1}{f'(x^*)}, \text{ com } f'(x*) \neq 0.$$

Da expressão $\phi(x) = x + A(x)\ f(x)$, tomando então $A(x) = -1/f'(x)$, obtemos

$$\phi(x) = x - \frac{f(x)}{f'(x)},$$

que é o processo de iteração do Método de Newton-Raphson, então definido por

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

O Método de Newton-Raphson, por ser um método de ponto fixo, produz uma sequência convergente, conforme o Teorema 5.6, desde que os critérios iniciais sejam atendidos.

A ordem de convergência do Método de Newton-Raphson é quadrática, dado esse formalmente apresentado pelo teorema a seguir.

Teorema 5.7: Se f, f', f'' são contínuas num intervalo aberto I cujo centro x^* é solução de $f(x^*) = 0$ e $f'(x^*) \neq 0$ então a ordem de convergência do método de Newton-Raphson é quadrática, ou seja, $p = 2$.

Vamos demonstrar o Teorema 5.7, segundo (FRANCO, 2014)[9]. Subtraindo a equação $x^* = \phi(x^*)$ da equação (5.1), obtemos

$$x_{k+1} - x^* = \phi(x_k) - \phi(x^*) \quad \text{onde} \quad \phi(x) = x - \frac{f(x)}{f'(x)}.$$

Desenvolvendo $\phi(x_k)$ em série de Taylor em torno do ponto x^*, obtemos

$$x_{k+1} - x^* = \phi(x^*) + (x_k - x^*)\phi'(x^*) + \frac{(x_k - x^*)^2}{2!}\phi''(\xi_k) - \phi(x^*).$$

Agora, no Método de Newton-Raphson, $\phi'(x^*) = 0$ (condição imposta para a determinação de $A(x)$), e portanto

$$\frac{|x_{k+1} - x^*|}{|x_k - x^*|^2} = \frac{\phi''(\xi_k)}{2!} \leq C.$$

Assim, a Definição 5.2 está satisfeita com $p = 2$ e com o fator de proporcionalidade é $\frac{\phi''(\xi_k)}{2!}$, ou seja, a convergência é quadrática. Isso implica que o erro diminui aproximadamente com o quadrado do erro da iteração anterior.

Vamos ver agora dois exemplos para melhor compreensão do funcionamento do método: um onde o processo converge para o zero da função determinada e outro onde não converge.

Exemplo 5.4.1 (Exemplo de convergência): Considere a função $f(x) = x^3 - 3x - 5$ que, de acordo com o Teorema de Bolzano 5.3, possue um zero no intervalo $(0, 3)$, uma vez que $f(0)f(3) < 0$, e sua derivada $f'(x) = 3x^2 - 3$. Desejamos encontrar um zero dessa função usando o Método de Newton-Raphson. Começamos com uma aproximação inicial $x_0 = 3$. A fórmula de iteração do método para essa função especificamente, será dada por

$$x_{k+1} = x_k - \frac{(x_k)^3 - 3(x_k) - 5}{3(x_k)^2 - 3}.$$

Após realizar 4 iterações, temos:

$$x_1 = 2{,}458333$$

$$x_2 \approx 2{,}294310$$

$$x_3 \approx 2{,}279144$$

$$x_4 \approx 2{,}279019.$$

Após algumas iterações, a sequência se aproxima do zero positivo da função, que é $x = 2{,}279018$, com 6 casas decimais.

Exemplo 5.4.2 (Exemplo de não convergência): Agora, consideremos a mesma função, porém com a aproximação inicial $x_0 = 0$. Realizando 4 iterações, temos:

$$x_1 = -1{,}666666$$

$x_2 \approx -0{,}798269$

$x_3 \approx -3{,}663326$

$x_4 \approx -2{,}504916.$

Neste caso, a sequência não converge para um zero da função. Isso ocorre porque no intervalo $[0, 3]$ há uma mudança de sinal da derivada, pois $f'(0)f'(3) < 0$, o que foge ao cumprimento de um dos critérios de garantia de convergência do método.

A escolha criteriosa da aproximação inicial e a observação dos critérios de convergência são essenciais para a aplicação bem-sucedida do Método de Newton-Raphson.

O método de Newton-Raphson é uma ferramenta poderosa para encontrar soluções de equações não lineares, proporcionando uma convergência rápida quando as condições adequadas são atendidas. No entanto, é importante lembrar que esse método, assim como todos os métodos citados anteriormente, pode não convergir caso todos os critérios de garantia de convergência não forem atendidos, portanto, é essencial verificar a convergência em casos específicos e, se necessário, explorar outras abordagens para garantir resultados confiáveis.

5.5 Método da Secante

O Método da Secante é uma técnica iterativa para encontrar soluções de equações não lineares. Ele é uma aproximação do Método de Newton-Raphson, permitindo que as derivadas não sejam calculadas explicitamente. Esse método utiliza uma secante para fazer uma estimativa da solução. O Método da Secante é especialmente útil quando o cálculo da derivada é inviável.

Vamos deduzir a fórmula de iteração do Método da Secante partindo da fórmula do Método de Newton-Raphson, que é dada por

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

Substituindo a derivada $f'(x_k)$ pelo quociente $\dfrac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$, temos

$$x_{k+1} = x_k - \frac{f(x_k)}{\dfrac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}}.$$

Logo, obtermos

$$x_{k+1} = x_k - \frac{f(x_k)(x_k - x_{k-1})}{f(x_k) - f(x_{k-1})} \tag{5.2}$$

que é a fórmula de iteração do Método da Secante.

Dado o problema de encontrar uma solução da equação não linear $f(x) = 0$, o Método da Secante começa com duas estimativas iniciais x_0 e x_1. A próxima iteração é calculada usando a relação acima onde x_{k+1} é a estimativa da solução.

O Método da Secante é capaz de convergir para a solução x^* da função sob certas condições. Para isso, é necessário que as estimativas iniciais sejam suficientemente próximas da solução e que a função seja bem comportada na região da solução. A seguir, apresentamos o teorema que formaliza essa condição.

Teorema 5.8: Seja $f : [a, b] \to \mathbb{R}$ uma função contínua com uma raiz x^* no intervalo (a, b). Suponha que f é diferenciável e que $f'(x^*) \neq 0$. Dadas duas aproximações iniciais x_0 e x_1 suficientemente próximas de x^*, o Método da Secante gera uma sequência $\{x_k\}$ que converge para x^*.

Para provar o Teorema 5.8, analisaremos o erro na k-ésima iteração, definido como

$$e_k = x_k - x^*.$$

Usando a expansão de Taylor para $f(x_k)$ e $f(x_{k-1})$ em torno de x^*, temos

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

$$f(x_{k-1}) = f'(x^*)(x_{k-1} - x^*) + \frac{f''(\xi_{k-1})}{2}(x_{k-1} - x^*)^2,$$

onde ξ_k e ξ_{k-1} são pontos entre x_k e x^* e entre x_{k-1} e x^*, respectivamente. Substituindo essas expressões na fórmula de iteração do Método da Secante (5.2), obtemos

$$x_{k+1} = x_k - \frac{[f'(x^*)(x_k - x^*) + \frac{f''(\xi_k)}{2}(x_k - x^*)^2](x_k - x_{k-1})}{[f'(x^*)(x_k - x^*) + \frac{f''(\xi_k)}{2}(x_k - x^*)^2] - [f'(x^*)(x_{k-1} - x^*) + \frac{f''(\xi_{k-1})}{2}(x_{k-1} - x^*)^2]}.$$

Para simplificar a análise, consideramos que as aproximações x_k e x_{k-1} estão suficientemente próximas de x^*. Assim, os termos quadráticos podem ser desprezados em primeira ordem de aproximação, resultando em

$$x_{k+1} \approx x_k - \frac{f'(x^*)(x_k - x^*)(x_k - x_{k-1})}{f'(x^*)(x_k - x^*) - f'(x^*)(x_{k-1} - x^*)}.$$

Simplificando, temos

$$x_{k+1} \approx x_k - \frac{(x_k - x^*)(x_k - x_{k-1})}{(x_k - x^*) - (x_{k-1} - x^*)}.$$

$$x_{k+1} \approx x_k - \frac{(x_k - x^*)(x_k - x_{k-1})}{x_k - x_{k-1}}.$$

$$x_{k+1} \approx x_k - (x_k - x^*) = x^*.$$

Isso mostra que x_{k+1} se aproxima de x^*, indicando que $e_{k+1} = x_{k+1} - x^* \to 0$ conforme $k \to \infty$. Portanto, a sequência gerada pelo Método da Secante converge para a raiz x^*.

Vamos agora apresentar o teorema da ordem de convergência do Método da Secante.

Teorema 5.9: Seja f uma função contínua em $[a, b]$ e duas vezes diferenciável em (a, b), com f' e f'' contínuas e com $f(x^*) = 0$ para algum $x^* \in (a, b)$. Se os pontos x_0 e x_1 forem escolhidos suficientemente próximos da raiz x^*, então o Método da Secante é convergente e tem uma ordem de convergência p de $\dfrac{1+\sqrt{5}}{2}$.

Para determinar a ordem de convergência do Método da Secante, começamos com sua definição

$$x_{k+1} = x_k - \frac{(x_k - x_{k-1})f(x_k)}{f(x_k) - f(x_{k-1})}.$$

Usando a expansão de Taylor de $f(x)$ em torno de x^*, temos

$$f(x_k) = f(x^*) + f'(x^*)(x_k - x^*) + \frac{1}{2}f''(x^*)(x_k - x^*)^2 + O((x_k - x^*)^3).$$

Como $f(x^*) = 0$, isto se simplifica para

$$f(x_k) = (x_k - x^*)f'(x^*) + \frac{1}{2}(x_k - x^*)^2 f''(x^*) + O((x_k - x^*)^3).$$

Seja $e_k = x_k - x^*$. Então, podemos escrever

$$f(x_k) = e_k f'(x^*) + \frac{1}{2}e_k^2 f''(x^*) + O(e_k^3).$$

De maneira similar,

$$f(x_{k-1}) = e_{k-1}f'(x^*) + \frac{1}{2}e_{k-1}^2 f''(x^*) + O(e_{k-1}^3).$$

Subtraindo estas equações, temos

$$f(x_k) - f(x_{k-1}) = (e_k f'(x^*) + \frac{1}{2}e_k^2 f''(x^*) + O(e_k^3)) - (e_{k-1}f'(x^*) + \frac{1}{2}e_{k-1}^2 f''(x^*) + O(e_{k-1}^3)).$$

Agrupando os termos comuns,

$$f(x_k) - f(x_{k-1}) = (e_k - e_{k-1})f'(x^*) + \frac{1}{2}(e_k^2 - e_{k-1}^2)f''(x^*) + O(e_k^3 - e_{k-1}^3).$$

Podemos fatorar para obter

$$f(x_k) - f(x_{k-1}) = (e_k - e_{k-1}) \left(f'(x^*) + \frac{1}{2}(e_k + e_{k-1}) f''(x^*) \right) + O(e_k^3 - e_{k-1}^3).$$

Usando $x_k - x_{k-1} = e_k - e_{k-1}$, substituímos na fórmula do Método da Secante:

$$\begin{aligned} x_{k+1} - x_k &= e_{k+1} - e_k \\ &= \frac{-(e_k - e_{k-1}) \left(e_k f'(x^*) + \frac{1}{2} e_k^2 f''(x^*) + O(e_k^3) \right)}{(e_k - e_{k-1}) \left(f'(x^*) + \frac{1}{2}(e_k + e_{k-1}) f''(x^*) \right) + O(e_k^3 - e_{k-1}^3)}. \end{aligned}$$

Simplificando,

$$e_{k+1} = e_k - \frac{e_k (e_k - e_{k-1}) f'(x^*)}{(e_k - e_{k-1}) \left(f'(x^*) + \frac{1}{2}(e_k + e_{k-1}) f''(x^*) \right)} + O(e_k^2).$$

Dividindo por $f'(x^*)$,

$$e_{k+1} = e_k - \frac{e_k}{1 + \frac{1}{2}(e_k + e_{k-1}) \frac{f''(x^*)}{f'(x^*)}} + O(e_k^2).$$

Para pequenos e_k, isso se aproxima de

$$e_{k+1} \approx \frac{1}{2} \frac{f''(x^*)}{f'(x^*)} e_k e_{k-1}.$$

A ordem de convergência ainda não é óbvia nesta equação, e para determiná-la, procuramos uma solução da forma:

$$|e_{k+1}| = c|e_k|^p.$$

Desta análise, também temos

$$|e_k| = c|e_{k-1}|^p,$$

e, portanto

$$|e_{k+1}| = c^{p+1} |e_{k-1}|^{p^2}.$$

Substituindo, temos

$$c^{p+1} |e_{k-1}|^{p^2} = \frac{c}{2} \left| \frac{f''(x^*)}{f'(x^*)} \right| |e_{k-1}|^{p+1}.$$

Igualando o coeficiente e a potência de e_{k-1} resulta em

$$c^p = \frac{1}{2} \left| \frac{f''(x^*)}{f'(x^*)} \right| \qquad \text{e} \qquad p^2 = p + 1.$$

A ordem de convergência do Método Secante é dada por p, portanto, é determinada como a raiz positiva da equação quadrática $p^2 - p - 1 = 0$, ou seja,

$$p = \frac{1+\sqrt{5}}{2} \approx 1.618,$$

que coincidentemente é um famoso número irracional chamado Proporção Áurea e é denotado por Φ. Vemos que o Método da Secante tem uma ordem de convergência superlinear que está entre o Método da Bissecção e o Método de Newton-Raphson.

Vamos ver agora um exemplo de aplicação do Método da Secante para encontrar uma aproximação para um zero da função $f(x) = x^3 - 3x - 5$.

Exemplo 5.5.1: Considere a função $f(x) = x^3 - 3x - 5$. Escolhemos as estimativas iniciais $x_0 = 2$ e $x_1 = 3$. A iteração do Método da Secante é dada por

$$x_{k+1} = x_k - \frac{f(x_k)(x_k - x_{k-1})}{f(x_k) - f(x_{k-1})}.$$

Ao realizar 4 iterações, temos

$$x_2 = 2{,}187500$$

$$x_3 \approx 2{,}250619$$

$$x_4 \approx 2{,}270365$$

$$x_5 \approx 2{,}276396.$$

Após algumas iterações, a sequência se aproxima do zero positivo da função, que é $x = 2{,}279018$ com 6 casas decimais.

O Método da Secante pode ser uma alternativa útil ao método de Newton-Raphson, pois não exige o cálculo da derivada da função em questão. Sua convergência superlinear é vantajosa em muitos cenários, embora possa ser mais lenta do que a convergência quadrática do método de Newton-Raphson. A análise da velocidade de convergência e a demonstração de convergência superlinear fortalecem a confiabilidade e a eficiência desse método. No entanto, assim como outros métodos iterativos, a escolha de estimativas iniciais próximas do zero da função é importante para o sucesso da convergência.

5.6 Comparação entre os Métodos

Ao resolver equações não lineares, é importante escolher o método adequado para alcançar resultados mais confiáveis e eficientes. Vamos comparar os Método da Bisseção, do Ponto Fixo, de Newton-Raphson e da Secante com base em critérios iniciais para

convergência, ordem de convergência e outras considerações relevantes.

Iniciando com o Método da Bisseção, é o que possue menos exigências, requerendo apenas a continuidade da função e uma mudança de sinal na função em um intervalo inicial. Isso a torna adequada para funções onde não possuímos uma expressão analítica ou para funções difíceis de derivar. Já os métodos do Ponto Fixo, de Newton-Raphson e da Secante requerem a continuidade da função e de sua derivada em um intervalo inicial. Portanto, a escolha do método dependerá da disponibilidade de informações sobre a função. Em termos de ordem de convergência, o Método da Secante e o Método de Newton-Raphson geralmente se destacam, possuindo convergência superlinear e quadrática respectivamente. O Método do Ponto Fixo depende do valor absoluto da derivada da função de iteração e possui ordem de convergência linear, assim como o Método da Bisseção, ou seja, é necessário um número maior de iterações para atingirem a mesma precisão, se comparados com os Métodos de Newton-Raphson e da Secante.

Para ilustrar a comparação entre os métodos, consideremos a equação $f(x) = x^3 - 2x - 5 = 0$, com zero positivo aproximado em $x \approx 2{,}279018$. Com uma tolerância de 10^{-6}, para o Método da Bisseção usando o intervalo $[1, 3]$, leva cerca de 20 iterações, para o Método do Ponto Fixo, usando $\phi(x) = \sqrt[3]{3x + 5}$ e $x_0 = 2$, leva cerca de 7 iterações. Para o Método de Newton-Raphson, usando $x_0 = 2$, leva cerca de 3 iterações e para o Método da Secante usando $x_0 = 1{,}5$ e $x_1 = 2$, leva cerca de 4 iterações.

Método	Bisseção	Ponto Fixo	Newton-Raphson	Secante
Estimativa Inicial (x_0)	2	2	2	x_0 =1,5 e x_1= 2
Iterações	20	7	3	4

A escolha do método mais apropriado depende das características específicas do problema em questão. O Método da Bisseção é capaz de lidar com maiores variações da função no intervalo, e de sua derivada, assim, quando as informações sobre a função são limitadas, pode ser uma boa escolha, mas pode ser mais lento em termos de convergência. Por outro lado, os Métodos de Newton-Raphson e da Secante são altamente eficazes quando a derivada da função é conhecida e a solução é aproximadamente estimada. O Método do Ponto Fixo oferece uma alternativa válida em muitos casos e pode ser mais simples de implementar. Portanto, a seleção do método ideal envolve considerar as características da função, a disponibilidade de informações derivativas e a relação entre precisão e tempo de computação.

6 Sistemas de Equações Algébricas

Sistemas de equações algébricas desempenham um papel importante em várias áreas do conhecimento, oferecendo uma poderosa ferramenta para modelar e resolver uma ampla variedade de problemas do mundo real. Neste capítulo, exploraremos duas categorias fundamentais de sistemas de equações: os sistemas lineares e os sistemas não lineares. Compreender a distinção entre essas duas categorias é importante, uma vez que a natureza das equações e os métodos de resolução variam significativamente.

Os sistemas lineares consistem em um conjunto de equações lineares, onde cada equação é uma combinação linear de variáveis, geometricamente, são representados por retas ou planos. Os sistemas não lineares, por outro lado, incluem equações que não podem ser representadas por combinações lineares de variáveis. Em vez de retas ou planos, as equações não lineares frequentemente geram curvas ou superfícies complexas no espaço.

A seguir, exploraremos métodos de resolução, propriedades fundamentais e aplicações em sistemas de equações, tanto lineares quanto não lineares. Ao entender com mais detalhes cada categoria, teremos mais ferramentas e abordagens para resolver uma variedade de problemas complexos que envolvem tais sistemas.

6.1 Sistemas de Equações Lineares

Os sistemas de equações lineares consistem em um conjunto de equações que descrevem relações entre variáveis. A classificação de um sistema linear é feita em função do número de soluções que ele admite, da seguinte maneira:

a) Sistema Possível ou Consistente: É todo sistema que possui pelo menos uma solução.
   a.1) Determinado se admite uma única solução, e,
   a.2) Indeterminado se admite mais de uma solução.
b) Sistema Impossível ou Inconsistente: É todo sistema que não admite solução.

Nesta seção, exploraremos alguns métodos de resolução de sistemas lineares. Todos os sistemas podem ser representados por meio de matrizes e vetores, essa representação oferece uma abordagem poderosa para analisar e resolver essas equações.

Existem dois principais grupos de métodos para a resolução de sistemas lineares: os

métodos diretos e os métodos iterativos. Os métodos diretos são caracterizados por uma quantidade finita de passos necessários para obter a solução exata do sistema. Alguns métodos diretos clássicos incluem a Eliminação de Gauss, Decomposição LU e Fatoração QR, os leitores interessados podem ver mais sobre o assunto em [9]. Esses métodos são utilizados devido à sua precisão, mas podem ser computacionalmente intensivos para sistemas de grande porte.

Por outro lado, os métodos iterativos, que será o foco neste trabalho, são caracterizados por se aproximarem gradualmente da solução. Eles são particularmente úteis em sistemas de grande escala, nos quais os métodos diretos podem ser impraticáveis devido ao alto consumo de recursos computacionais. Exemplos de métodos iterativos clássicos incluem o Método de Jacobi, o Método de Gauss-Seidel e o Gradiente Conjugado. Esses métodos atualizam as estimativas iterativamente até que uma solução aceitável seja alcançada.

Na próxima seção, aprofundaremos nosso estudo no Método de Jacobi, examinando sua aplicação, implementação e propriedades, e vamos analisar somente sistemas que admitem uma única solução.

6.1.1 Método de Jacobi

O Método de Jacobi é um método iterativo utilizado para resolver sistemas lineares, sendo particularmente útil para sistemas de grande escala onde os métodos diretos podem ser computacionalmente custosos. Vamos explorar a dedução da fórmula de iteração do Método de Jacobi e apresentar um exemplo prático para compreender melhor o funcionamento desse método.

Suponhamos que temos um sistema linear de equações, representado na forma matricial como $\mathbf{Ax} = \mathbf{b}$, onde $\mathbf{A}$ é uma matriz dos coeficientes, $\mathbf{x}$ é o vetor de incógnitas e $\mathbf{b}$ é o vetor dos termos constantes. O Método de Jacobi divide a matriz $\mathbf{A}$ na forma $\mathbf{A} = \mathbf{L} + \mathbf{D} + \mathbf{U}$, onde $\mathbf{L}$ contém a parte triangular inferior de $\mathbf{A}$, $\mathbf{D}$ contém a diagonal de $\mathbf{A}$ e $\mathbf{U}$ contém a parte triangular superior de $\mathbf{A}$.

Substituindo $\mathbf{A} = \mathbf{L} + \mathbf{D} + \mathbf{U}$ em $\mathbf{Ax} = \mathbf{b}$, temos:

$$(\mathbf{L} + \mathbf{D} + \mathbf{U})\mathbf{x} = \mathbf{b},$$

$$\mathbf{Dx} = \mathbf{b} - (\mathbf{L} + \mathbf{U})\mathbf{x},$$

$$\mathbf{x} = \mathbf{D}^{-1}[\mathbf{b} - (\mathbf{L} + \mathbf{U})\mathbf{x}].$$

Assim, a fórmula de iteração para o Método de Jacobi é dada por

$$\mathbf{x}^{k+1} = \mathbf{D}^{-1}\left(\mathbf{b} - (\mathbf{L} + \mathbf{U})\mathbf{x}^{k}\right),$$

onde $\mathbf{x}^{k+1}$ é o vetor de estimativas atualizadas na $(k+1)$-ésima iteração e $\mathbf{x}^{k}$ é o vetor de estimativas da k-ésima iteração.

Desenvolvendo a fórmula para cada componente do vetor $\mathbf{x}$, temos

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1, j\neq i}^{n} a_{ij}x_j^{(k)}\right) \quad \text{para } i = 1, 2, \ldots, n,$$

onde $x_i^{(k+1)}$ é a estimativa atualizada da variável x_i na $(k+1)$-ésima iteração, n é o número de variáveis, a_{ij} são os elementos da matriz $\mathbf{A}$, e b_i são os elementos do vetor $\mathbf{b}$.

Para garantir a convergência do Método de Jacobi, é necessário que a matriz dos coeficientes $\mathbf{A}$ seja diagonalmente dominante. Isso significa que, para cada linha i do sistema, o valor absoluto do coeficiente diagonal $|a_{ii}|$ deve ser maior do que a soma dos valores absolutos dos outros coeficientes na mesma linha, ou seja,

$$|a_{ii}| > \sum_{j=1, j\neq i}^{n} |a_{ij}|.$$

Se essa condição for atendida, o Método de Jacobi converge para uma solução. Caso contrário, a convergência não é garantida.

O Método de Jacobi, embora menos eficiente em termos de convergência, foi fundamental no desenvolvimento de outros métodos iterativos, inspirando abordagens mais rápidas, como o Método de Gauss-Seidel. Este último acelera a convergência ao utilizar os valores mais recentes das variáveis à medida que são calculados. Na próxima seção, exploraremos o Método de Gauss-Seidel.

6.1.2 Método de Gauss-Seidel

O Método de Gauss-Seidel é uma variação do Método de Jacobi, e ambos são métodos iterativos para resolver sistemas lineares. A principal diferença entre eles reside na forma como as estimativas das variáveis são atualizadas durante o processo iterativo.

No Método de Jacobi, as novas estimativas de todas as variáveis são calculadas simultaneamente, utilizando exclusivamente os valores das iterações anteriores. No entanto, o Método de Gauss-Seidel otimiza esse processo ao utilizar as estimativas mais recentes à medida que elas são calculadas. Assim, as variáveis são atualizadas sequencialmente, e cada nova estimativa pode imediatamente influenciar o cálculo das próximas variáveis na

mesma iteração, o que geralmente resulta em uma convergência mais rápida.

Matematicamente, o Método de Gauss-Seidel pode ser expresso da seguinte forma:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right) \quad \text{para } i = 1, 2, \ldots, n,$$

onde $x_i^{(k+1)}$ é a estimativa atualizada da variável x_i na $(k+1)$-ésima iteração, a_{ij} são os elementos da matriz $\mathbf{A}$, e b_i são os elementos do vetor $\mathbf{b}$.

Para que o Método de Gauss-Seidel funcione eficientemente e converja, é importante que a matriz dos coeficientes $\mathbf{A}$ seja diagonalmente dominante, assim como no Método de Jacobi. Se essa condição for atendida, o Método de Gauss-Seidel tende a convergir mais rapidamente do que o Método de Jacobi, tornando-o uma escolha preferível em muitos casos práticos. Vamos ver um exemplo ilustrativo.

Exemplo 6.1.1: Encontre uma solução aproximada para o sistema de equações lineares a seguir utilizando o Método de Gauss-Seidel.

$$\begin{cases} 3x - 2y + z = 3 \\ -2x + 4y - 2z = 2 \\ x - 3y + 5z = 1 \end{cases}$$

Para iniciar o processo, vamos escrever o o sistema na forma matricial $\mathbf{Ax} = \mathbf{b}$.

$$\mathbf{A} = \begin{bmatrix} 3 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -3 & 5 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}.$$

Observe que a matriz $\mathbf{A}$ é diagonalmente dominante, pois $3 > -2 + 1, 4 > -2 + (-2)$ e $5 > 1 + (-3)$, assim sendo, usando o método de Gauss-Seidel, podemos decompor a matriz $\mathbf{A}$ em $\mathbf{L}$ (triangular inferior), $\mathbf{D}$ (diagonal) e $\mathbf{U}$ (triangular superior):

$$\mathbf{L} = \begin{bmatrix} 0 & 0 & 0 \\ -2 & 0 & 0 \\ 1 & -3 & 0 \end{bmatrix}, \quad \mathbf{D} = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 5 \end{bmatrix}, \quad \mathbf{U} = \begin{bmatrix} 0 & -2 & 1 \\ 0 & 0 & -2 \\ 0 & 0 & 0 \end{bmatrix}.$$

Suponha que começamos com uma estimativa inicial de

$$x^{(0)} = [0 \quad 0 \quad 0]^T$$

Primeira iteração:

$$\mathbf{x}^1 = \mathbf{D}^{-1}[\mathbf{b} - (\mathbf{L} + \mathbf{U})\mathbf{x}^0]$$

$$= \begin{bmatrix} 1/3 & 0 & 0 \\ 0 & 1/4 & 0 \\ 0 & 0 & 1/5 \end{bmatrix} \left[\begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix} - \left(\begin{bmatrix} 0 & -2 & 1 \\ -2 & 0 & -2 \\ 1 & -3 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \right) \right]$$

Efetuando os cálculos elemento a elemento,

$$x_1 = \frac{1}{3}\big[3 - (0 \cdot 0 - 2 \cdot 0 + 1 \cdot 0)\big] = 1$$
$$y_1 = \frac{1}{4}\big[2 - (-2 \cdot 1 + 0 \cdot 0 - 2 \cdot 0)\big] = 1$$
$$z_1 = \frac{1}{5}\big[1 - (1 \cdot 1 - 3 \cdot 1 + 0 \cdot 0)\big] = \frac{3}{5}.$$

Assim temos o vetor de aproximação para a primeira iteração

$$x^{(1)} = [1{,}0000 \quad 1{,}0000 \quad 0{,}6000]^T.$$

Os resultados da primeira iteração são $x = 1$, $y = 1$ e $z = 0{,}6$. Realizando 10 iterações, temos os seguintes resultados.

$$x^{(2)} = [1{,}4667 \quad 1{,}5333 \quad 0{,}8267]^T,$$
$$x^{(3)} = [1{,}7467 \quad 1{,}7867 \quad 0{,}9227]^T,$$
$$x^{(4)} = [1{,}8836 \quad 1{,}9031 \quad 0{,}9652]^T,$$
$$x^{(5)} = [1{,}9470 \quad 1{,}9561 \quad 0{,}9842]^T,$$
$$x^{(6)} = [1{,}9760 \quad 1{,}9801 \quad 0{,}9929]^T,$$
$$x^{(7)} = [1{,}9891 \quad 1{,}9910 \quad 0{,}9968]^T,$$
$$x^{(8)} = [1{,}9951 \quad 1{,}9959 \quad 0{,}9985]^T,$$
$$x^{(9)} = [1{,}9978 \quad 1{,}9982 \quad 0{,}9993]^T,$$
$$x^{(10)} = [1{,}9990 \quad 1{,}9992 \quad 0{,}9997]^T.$$

Podemos observar que os valores encontrados para cada variável estão se aproximando dos valores corretos que são $x = 2$, $y = 2$ e $z = 1$.

O Método de Gauss-Seidel é uma ferramenta para resolver sistemas lineares iterativamente, proporcionando convergência rápida em alguns casos, no entanto, a convergência

não é garantida para todos os sistemas, e a escolha da estratégia iterativa apropriada depende da natureza do problema. O exemplo fornecido ilustra o processo de iteração, mostrando como o método se aproxima da solução a cada passo.

Na próxima seção, abordaremos sistemas não lineares e métodos para resolvê-los. Exploraremos técnicas numéricas para lidar com essas equações, que podem ser complexas em diversos contextos.

6.2 Sistemas de Equações Não Lineares

Neste capítulo, exploraremos métodos numéricos para resolver sistemas de equações não lineares, que consistem em um conjunto de equações que envolvem múltiplas variáveis e funções não lineares. Esses sistemas aparecem em diversas áreas da matemática aplicada, física, engenharia e outras disciplinas científicas. Resolver sistemas de equações não lineares de forma analítica pode ser desafiador ou mesmo impossível em muitos casos, o que torna os métodos numéricos uma ferramenta de grande utilidade para obter soluções aproximadas.

Um sistema de n equações e n incógnitas $x_1, x_2, \ldots, x_n$ é chamado de não linear se uma ou mais equações é não linear. Tem-se uma forma geral que pode ser usada para qualquer sistema não linear,

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

que é a forma homogênea para escrever o sistema, ou simplesmente $F(\mathbf{x}) = 0$, em que $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$ e $F(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_n(\mathbf{x}))^T$. Um vetor $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ que satisfaz $F(\mathbf{x}) = 0$ é denominado solução do sistema não linear.

Um exemplo de sistema não linear é

$$\begin{cases} y^2 + x^2 = 25 \\ y - x^2 = -6. \end{cases} \tag{6.1}$$

Este sistema não linear envolve duas equações com duas incógnitas, x e y. A primeira equação, $y^2 + x^2 = 25$, representa um círculo de raio 5 centrado na origem, enquanto a segunda equação, $y = x^2 - 6$, descreve uma parábola voltada para cima. As soluções do sistema correspondem aos pontos de interseção entre o círculo e a parábola, podendo haver até quatro soluções distintas, onde as curvas se encontram no plano cartesiano.

Existem vários métodos para resolver sistemas não lineares, porém, nesse trabalho vamos abordar o Método de Newton e o Método de Broyden, o segundo fazendo parte de um grupo de métodos de resolução de sistemas não lineares denominado Métodos Quasi-Newton. Também vamos citar a Fórmula de Sherman-Morrison, que é uma técnica que permite atualizar a inversa de uma matriz modificada sem a necessidade de calcular novamente a inversa da matriz original. Cada um desses métodos possui suas próprias características, vantagens e limitações, tornando-os adequados para diferentes situações e tipos de sistemas de equações. Vamos explorar cada método em detalhes e discutir sua implementação, convergência e eficiência.

6.2.1 Método de Newton

O método mais amplamente estudado e conhecido para resolver sistemas de equações não lineares é o Método de Newton. Recordando do Capítulo 5, desenvolvemos o Método de Newton-Raphson para funções de uma variável através de um método de ponto fixo. Considerando a equação

$$\phi(x) = x + A(x)f(x), \quad \text{com} \quad f'(x) \neq 0,$$

dá a convergência quadrática ao método pela escolha de

$$A(x) = -\frac{1}{\phi'(x)},$$

assumindo que $\phi'(x) \neq 0$. A utilização de uma técnica similar no caso n-dimencional envolve a matriz

$$\mathbf{A}(\mathbf{x}) = \begin{bmatrix} a_{11(\mathbf{X})} & a_{12(\mathbf{X})} & \dots & a_{1n(\mathbf{X})} \\ a_{21(\mathbf{X})} & a_{22(\mathbf{X})} & \dots & a_{2n(\mathbf{X})} \\ \vdots & \vdots & & \vdots \\ a_{n1(\mathbf{X})} & a_{n2(\mathbf{X})} & \dots & a_{nn(\mathbf{X})} \end{bmatrix},$$

onde cada entrada a_{ij} é uma função de $\mathbb{R}^n$ em $\mathbb{R}$. Isso requer que a inversa de $\mathbf{A}(\mathbf{x})$ seja encontrada, para que o método de Ponto Fixo gerado por

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

tenha convergência quadrática para a solução $\mathbf{F}(\mathbf{x}) = 0$, assumindo que $\mathbf{A}(\mathbf{x})$ é não-singular.

Uma escolha apropriada para $\mathbf{A}(\mathbf{x})$ é a matriz Jacobiana definida por

$$\mathbf{J}(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(\mathbf{x}) & \frac{\partial f_1}{\partial x_2}(\mathbf{x}) & \cdots & \frac{\partial f_1}{\partial x_n}(\mathbf{x}) \\ \frac{\partial f_2}{\partial x_1}(\mathbf{x}) & \frac{\partial f_2}{\partial x_2}(\mathbf{x}) & \cdots & \frac{\partial f_2}{\partial x_n}(\mathbf{x}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1}(\mathbf{x}) & \frac{\partial f_n}{\partial x_2}(\mathbf{x}) & \cdots & \frac{\partial f_n}{\partial x_n}(\mathbf{x}) \end{bmatrix}.$$

A demonstração desse fato pode ser encontrada em [4]. Sendo assim a função $\mathbf{G}$ é definida por

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

Como $\mathbf{x}^{(k+1)} = \mathbf{G}(\mathbf{x}^{(k)})$, o método de iteração funcional desenvolve-se selecionando $\mathbf{x}^{(0)}$ e gerando, para $k >= 1$,

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \mathbf{J}(\mathbf{x}^{(k)})^{-1}\mathbf{F}(\mathbf{x}^{(k)}).$$

Isso é chamado de Método de Newton para sistemas não lineares, e geralmente é esperado que gere a convergência quadrática, estabelecido um valor inicial suficientemente próximo da solução.

Vamos agora encontrar uma solução aproximada para o sistema (6.1) que é dado por

$$\begin{cases} f(x,y) = x^2 + y^2 - 25, \\ g(x,y) = -x^2 + y + 6. \end{cases}$$

A Jacobiana $\mathbf{J}$ do sistema é

$$\mathbf{J} = \begin{bmatrix} 2x & 2y \\ -2x & 1 \end{bmatrix}.$$

Com as estimativas iniciais $x^{(0)} = 1$ e $y^{(0)} = 2$, a Jacobiana se torna

$$\mathbf{J}^{(0)} = \begin{bmatrix} 2 \cdot 1 & 2 \cdot 2 \\ -2 \cdot 1 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 4 \\ -2 & 1 \end{bmatrix}.$$

A inversa da Jacobiana é então calculada como

$$\mathbf{J}^{-1} = \frac{1}{10}\begin{bmatrix} 1 & -4 \\ 2 & 2 \end{bmatrix} = \begin{bmatrix} 0{,}1 & -0{,}4 \\ 0{,}2 & 0{,}2 \end{bmatrix}. \tag{6.2}$$

Para calcular $\mathbf{F}(x^{(0)}, y^{(0)})$, temos

$$\mathbf{F}^{(0)} = \begin{bmatrix} f(x^{(0)}, y^{(0)}) \\ g(x^{(0)}, y^{(0)}) \end{bmatrix} = \begin{bmatrix} 1^2 + 2^2 - 25 \\ -1^2 + 2 + 6 \end{bmatrix} = \begin{bmatrix} 1 + 4 - 25 \\ -1 + 2 + 6 \end{bmatrix} = \begin{bmatrix} -20 \\ 7 \end{bmatrix}.$$

O vetor de correção é calculado como

$$-\mathbf{J}^{-1} \cdot F^{(0)} = -\begin{bmatrix} 0{,}1 & -0{,}4 \\ 0{,}2 & 0{,}2 \end{bmatrix}\begin{bmatrix} -20 \\ 7 \end{bmatrix} = \begin{bmatrix} 4{,}8 \\ 2{,}6 \end{bmatrix}.$$

Assim, a nova estimativa é

$$\mathbf{x}^{(1)} = \begin{bmatrix} 1{,}0000 \\ 2{,}0000 \end{bmatrix} + \begin{bmatrix} 4{,}8 \\ 2{,}6 \end{bmatrix} = \begin{bmatrix} 5{,}8000 \\ 4{,}6000 \end{bmatrix}.$$

Usamos o mesmo procedimento para calcular mais 3 iterações, os resultados obtidos são

$$\mathbf{x}^{(2)} = [3{,}7567 \quad 3{,}9373],$$
$$\mathbf{x}^{(3)} = [3{,}1944 \quad 3{,}8878],$$
$$\mathbf{x}^{(4)} = [3{,}1448 \quad 3{,}8875].$$

Repetimos o processo até que as estimativas convirjam para uma das soluções do sistema que é aproximadamente $x = 3{,}1444$ e $y = 3{,}8875$. O Método de Newton para sistemas não lineares é uma poderosa ferramenta para encontrar soluções de sistemas complexos. Sua rapidez de convergência torna-o especialmente útil em muitas aplicações em matemática, engenharia e ciência. No entanto, a escolha adequada da aproximação inicial e a consideração das condições de convergência são essenciais para garantir o sucesso do método.

Embora o Método de Newton seja eficaz para resolver sistemas não lineares, ele pode enfrentar dificuldades quando a matriz Jacobiana é inviável ou impossível de calcular. Na próxima seção, exploraremos o Método de Broyden, que não exige o cálculo explícito das derivadas, e a Fórmula de Sherman-Morrison, que facilita a atualização da inversa da matriz Jacobiana com menos recursos após perturbações.

6.2.2 Método de Broyden e a Fórmula de Sherman-Morrison

Uma limitação significativa do Método de Newton na resolução de sistemas não lineares de equações é que, a cada iteração, é necessário calcular a matriz Jacobiana e resolver um sistema linear que envolve essa matriz. Considerando a quantidade de cálculos computacionais envolvidos em cada iteração do Método de Newton, além do fato de que, em muitas situações, o cálculo exato das derivadas parciais pode ser inconveniente, surgiram os Métodos Quasi-Newton. Esses métodos são algoritmos iterativos que buscam resolver sistemas de equações não lineares por meio de aproximações sucessivas, evitando a necessidade de calcular derivadas explicitamente, o que pode ser computacionalmente custoso ou inviável em muitos casos. Nesta seção, exploraremos o Método de Broyden, uma abordagem popular entre os Métodos Quasi-Newton, e discutiremos a Fórmula de Sherman-Morrison, que pode melhorar a eficiência desses métodos.

O Método de Broyden é um exemplo de Método Quasi-Newton que busca aproximar a matriz Jacobiana das equações do sistema por meio de iterações. Para descrever o Método de Broyden, suponha-se que seja dada uma aproximação inicial $\mathbf{x}^{(0)}$ para a solução $\mathbf{p}$ de $\mathbf{F}(\mathbf{x}) = \mathbf{0}$. A próxima aproximação, $\mathbf{x}^{(1)}$, pode ser calculada da mesma forma utilizada no Método de Newton. No entanto, se for difícil calcular as derivadas com precisão, é possível utilizar aproximações por diferenças finitas.

No entanto, para calcular $\mathbf{x}^{(2)}$, o método se afasta do Método de Newton e adota uma estratégia inspirada no Método da Secante para equações não lineares de uma única variável. O Método da Secante utiliza a aproximação

$$f'(x_1) = \frac{f(x_1) - f(x_0)}{x_1 - x_0},$$

como substituto para $f'(x_1)$ no Método de Newton-Raphson, que foi apresentado na Seção 5.5. De maneira semelhante, a matriz aproximada $\mathbf{B}^{(k)}$, que é a matriz que aproxima a Jacobiana no Método de Broyden, é atualizada pela fórmula

$$\mathbf{B}^{(k+1)} = \mathbf{B}^{(k)} + \frac{\left(\Delta\mathbf{y} - \mathbf{B}^{(k)}\Delta\mathbf{x}\right)}{\|\Delta\mathbf{x}\|^2} (\Delta\mathbf{x})^T \tag{6.3}$$

onde $\Delta\mathbf{x} = \mathbf{x}_{k+1} - \mathbf{x}_k$ e $\Delta\mathbf{y} = \mathbf{F}(\mathbf{x}_{k+1}) - \mathbf{F}(\mathbf{x}_k)$.

Na Seção 6.2.1, utilizamos o Método de Newton para o sistema (6.1) dado por

$$\begin{cases} f(x, y) = x^2 + y^2 - 25, \\ g(x, y) = -x^2 + y + 6, \end{cases}$$

com estimativas iniciais $(x_0, y_0) = (1, 2)$. Encontramos a Jacobiana

$$J^{(0)} = \begin{bmatrix} 2 & 4 \\ -2 & 1 \end{bmatrix}$$

e as novas estimativas $(x_1, y_1) = (5.8, 4.6)$. Vamos utilizá-las para executar um passo do método de Broyden e assim encontrar a Jacobiana aproximada para ser utilizada em uma próxima iteração.

Primeiro, calculamos $\Delta\mathbf{x}$ e $\Delta\mathbf{y}$.

$$\Delta\mathbf{x} = \begin{pmatrix} x_1 - x_0 \\ y_1 - y_0 \end{pmatrix} = \begin{pmatrix} 5{,}8 - 1 \\ 4{,}6 - 2 \end{pmatrix} = \begin{pmatrix} 4{,}8 \\ 2{,}6 \end{pmatrix}$$

Para o cálculo de $\Delta\mathbf{y}$, calculamos f e g para (x_0, y_0) e (x_1, y_1).

$$f(x_0, y_0) = 1^2 + 2^2 - 25 = 1 + 4 - 25 = -20$$

$$g(x_0, y_0) = -1^2 + 2 + 6 = -1 + 2 + 6 = 7$$

$$f(x_1, y_1) = 5{,}8^2 + 4{,}6^2 - 25 = 33{,}64 + 21{,}16 - 25 = 29{,}80$$

$$g(x_1, y_1) = -5{,}8^2 + 4{,}6 + 6 = -33{,}64 + 4{,}6 + 6 = -23{,}04.$$

Então

$$\Delta\mathbf{y} = \begin{pmatrix} f(x_1, y_1) - f(x_0, y_0) \\ g(x_1, y_1) - g(x_0, y_0) \end{pmatrix} = \begin{pmatrix} 29{,}80 - (-20) \\ -23{,}04 - 7 \end{pmatrix} = \begin{pmatrix} 49{,}80 \\ -30{,}04 \end{pmatrix}.$$

Agora, para aplicamos a fórmula (6.3) do Método de Broyden, primeiro, calculamos $\mathbf{B}^{(0)}\Delta\mathbf{x}$.

$$\mathbf{B}^{(0)}\Delta\mathbf{x} = \begin{bmatrix} 2 & 4 \\ -2 & 1 \end{bmatrix} \begin{pmatrix} 4{,}8 \\ 2{,}6 \end{pmatrix} = \begin{pmatrix} 2 \cdot 4{,}8 + 4 \cdot 2{,}6 \\ -2 \cdot 4{,}8 + 1 \cdot 2{,}6 \end{pmatrix} = \begin{pmatrix} 20 \\ -7 \end{pmatrix}.$$

Então

$$\Delta\mathbf{y} - \mathbf{B}^{(0)}\Delta\mathbf{x} = \begin{pmatrix} 49{,}80 \\ -30{,}04 \end{pmatrix} - \begin{pmatrix} 20{,}0 \\ -7{,}0 \end{pmatrix} = \begin{pmatrix} 29{,}80 \\ -23{,}04 \end{pmatrix}. \tag{6.4}$$

O quadrado da norma de $\Delta\mathbf{x}$ é

$$\|\Delta\mathbf{x}\|^2 = 4{,}8^2 + 2{,}6^2 = 23{,}04 + 6{,}76 = 29{,}80.$$

A matriz $(\Delta\mathbf{x})^T$ é:

$$(\Delta\mathbf{x})^T = \begin{pmatrix} 4{,}8 & 2{,}6 \end{pmatrix}.$$

Agora, calculamos a atualização

$$\frac{\left(\Delta\mathbf{y} - \mathbf{B}^{(0)}\Delta\mathbf{x}\right)}{\|\Delta\mathbf{x}\|^2}(\Delta\mathbf{x})^T = \frac{\begin{pmatrix} 29{,}80 \\ -23{,}04 \end{pmatrix}}{29{,}80}\begin{pmatrix} 4{,}8 & 2{,}6 \end{pmatrix} = \begin{pmatrix} 1 & -0{,}774 \end{pmatrix}\begin{pmatrix} 4{,}8 & 2{,}6 \end{pmatrix},$$

o valor do produto é

$$\begin{pmatrix} 1 & -0{,}774 \end{pmatrix}\begin{pmatrix} 4{,}8 & 2{,}6 \end{pmatrix} = \begin{pmatrix} 4{,}8 & 2{,}6 \\ -3{,}72 & -2{,}01 \end{pmatrix}.$$

Finalmente, atualizamos $\mathbf{B}^{(0)}$.

$$\mathbf{B}^{(1)} = \mathbf{B}^{(0)} + \begin{pmatrix} 4{,}8 & 2{,}6 \\ -3{,}72 & -2{,}01 \end{pmatrix} = \begin{bmatrix} 2+4{,}8 & 4+2{,}6 \\ -2-3{,}72 & 1-2{,}01 \end{bmatrix} = \begin{bmatrix} 6{,}8 & 6{,}6 \\ -5{,}72 & -1{,}01 \end{bmatrix}.$$

Para encontrar as próximas estimativas da solução do sistema e questão, basta utilizar essa aproximação para a Jacobiana no Método de Newton.

O Método de Broyden é eficiente para resolver sistemas não lineares sem a necessidade de calcular derivadas analíticas, utilizando uma matriz Jacobiana aproximada que é atualizada a cada iteração. No entanto, mesmo com essa aproximação, o método ainda exige a resolução de um sistema linear a cada passo, o que é necessário devido ao cálculo da inversa da matriz Jacobiana aproximada. Esse processo pode ser bastante caro em termos de custo computacional, pois a solução do sistema linear e a atualização da inversa da matriz podem consumir muitos recursos.

Para melhorar a eficiência do Método de Broyden, especialmente quando lidamos com sistemas lineares perturbados – ou seja, sistemas em que a matriz dos coeficientes é alterada por pequenas mudanças – podemos empregar a Fórmula de Sherman-Morrison. Esta fórmula permite atualizar a inversa de uma matriz após uma perturbação, evitando o cálculo completo da inversa, o que pode ser computacionalmente caro.

Suponha que temos uma matriz $\mathbf{B}^{(k)}$ e que após uma perturbação, a matriz é alterada para $\mathbf{B}^{(k+1)}$, tal que

$$\mathbf{B}^{(k+1)} = \mathbf{B}^{(k)} + \Delta\mathbf{x} \cdot (\Delta\mathbf{y})^T,$$

onde $\Delta\mathbf{x}$ e $\Delta\mathbf{y}$ são vetores que representam a perturbação. A Fórmula de Sherman-Morrison afirma que a inversa da matriz perturbada $\left(\mathbf{B}^{(k+1)}\right)^{-1}$ pode ser expressa em termos da inversa da matriz original $\left(\mathbf{B}^{(k)}\right)^{-1}$ e da perturbação aplicada. A fórmula é dada por

$$\left(\mathbf{B}^{(k+1)}\right)^{-1} = \left(\mathbf{B}^{(k)}\right)^{-1} - \frac{\left(\mathbf{B}^{(k)}\right)^{-1} \cdot \left(\Delta\mathbf{y} - \mathbf{B}^{(k)} \cdot \Delta\mathbf{x}\right) (\Delta\mathbf{x})^T \cdot \left(\mathbf{B}^{(k)}\right)^{-1}}{1 + (\Delta\mathbf{x})^T \left(\mathbf{B}^{(k)}\right)^{-1} \cdot (\Delta\mathbf{y} - \mathbf{B}^{(k)} \cdot \Delta\mathbf{x})}. \tag{6.5}$$

Nesta fórmula, $\Delta\mathbf{x}$ é o vetor de diferença nas variáveis e $\Delta\mathbf{y}$ é o vetor correspondente às mudanças na função objetivo. A Fórmula de Sherman-Morrison nos permite calcular a nova inversa $\left(\mathbf{B}^{(k+1)}\right)^{-1}$ de forma eficiente sem ter que recalcular a inversa da matriz perturbada a partir do zero.

Vamos reutilizar o sistema (6.1) e as informações previamente obtidas.

A matriz Jacobiana inicial $\mathbf{B}^{(0)}$ é

$$\mathbf{B}^{(0)} = \begin{bmatrix} 2 & 4 \\ -2 & 1 \end{bmatrix}.$$

A perturbação dada por

$$\mathbf{B}^{(k+1)} = \mathbf{B}^{(k)} + \Delta\mathbf{x} \cdot (\Delta\mathbf{y})^T,$$

onde $\Delta\mathbf{x}$ e $\Delta\mathbf{y}$ são:

$$\Delta\mathbf{x} = \begin{pmatrix} 4{,}8 \\ 2{,}6 \end{pmatrix}$$

$$\Delta\mathbf{y} = \begin{pmatrix} 49{,}80 \\ -30{,}04 \end{pmatrix}.$$

Já temos as informações sobre a inversa de $\mathbf{B}^{(0)}$ que foi calculada em (6.2) na execução do Método de Newton, sendo

$$\left(\mathbf{B}^{(0)}\right)^{-1} = \begin{bmatrix} 0{,}1 & -0{,}4 \\ 0{,}2 & 0{,}2 \end{bmatrix}.$$

Através da execução do Método de Broyden, obtemos em (6.4) a informação que

$$\Delta\mathbf{y} - \mathbf{B}^{(0)} \cdot \Delta\mathbf{x} = \begin{pmatrix} 29{,}80 \\ -23{,}04 \end{pmatrix}.$$

Calculando o produto

$$\left(\mathbf{B}^{(0)}\right)^{-1} \cdot \left(\Delta\mathbf{y} - \mathbf{B}^{(k)} \cdot \Delta\mathbf{x}\right) = \begin{bmatrix} 0{,}1 & -0{,}4 \\ 0{,}2 & 0{,}2 \end{bmatrix} \begin{pmatrix} 29{,}80 \\ -23{,}04 \end{pmatrix} = \begin{pmatrix} 12{,}196 \\ 1{,}352 \end{pmatrix}.$$

Então

$$(\Delta\mathbf{x})^T \cdot \left(\mathbf{B}^{(k)}\right)^{-1} \cdot \left(\Delta\mathbf{y} - \mathbf{B}^{(k)} \cdot \Delta\mathbf{x}\right) = \begin{pmatrix} 4{,}8 & 2{,}6 \end{pmatrix} \begin{pmatrix} 12{,}196 \\ 1{,}352 \end{pmatrix} = 62{,}06.$$

Finalmente, a atualização da inversa é

$$\left(\mathbf{B}^{(1)}\right)^{-1} = \left(\mathbf{B}^{(0)}\right)^{-1} - \frac{\left(\mathbf{B}^{(0)}\right)^{-1} \cdot \left(\Delta\mathbf{y} - \mathbf{B}^{(0)} \cdot \Delta\mathbf{x}\right) (\Delta\mathbf{x})^T \cdot \left(\mathbf{B}^{(0)}\right)^{-1}}{1 + (\Delta\mathbf{x})^T \left(\mathbf{B}^{(0)}\right)^{-1} \cdot \left(\Delta\mathbf{y} - \mathbf{B}^{(0)} \cdot \Delta\mathbf{x}\right)}.$$

Substituindo os valores

$$\left(\mathbf{B}^{(1)}\right)^{-1} = \begin{bmatrix} 0{,}1 & -0{,}4 \\ 0{,}2 & 0{,}2 \end{bmatrix} - \frac{\begin{bmatrix} 12{,}196 \\ 1{,}352 \end{bmatrix} \begin{pmatrix} 4{,}8 & 2{,}6 \end{pmatrix} \cdot \begin{bmatrix} 0{,}1 & -0{,}4 \\ 0{,}2 & 0{,}2 \end{bmatrix}}{1 + 62{,}06},$$

$$\left(\mathbf{B}^{(1)}\right)^{-1} = \begin{bmatrix} 0{,}1 & -0{,}4 \\ 0{,}2 & 0{,}2 \end{bmatrix} - \begin{bmatrix} 0{,}01 & -0{,}39 \\ 1{,}197 & 1{,}197 \end{bmatrix},$$

$$\left(\mathbf{B}^{(1)}\right)^{-1} = \begin{bmatrix} -0{,}09 & -0{,}01 \\ -1 & -1 \end{bmatrix}.$$

Bastando agora usar essa aproximação da inversa da Jacobiana no Método de Newton para encontrar as próximas estimativas para a solução do sistema em questão.

A capacidade de atualizar a inversa de uma matriz de forma eficiente pode ser importante quando se trabalha com métodos numéricos iterativos como o Método de Broyden, pois reduz significativamente o custo computacional associado à resolução de sistemas lineares perturbados a cada iteração. Assim, a aplicação da Fórmula de Sherman-Morrison contribui para uma melhoria na performance e na eficiência dos métodos quasi-

Newton.

Em resumo, o Método de Broyden e a Fórmula de Sherman-Morrison oferecem uma solução eficaz para sistemas não lineares, combinando a aproximação iterativa da matriz jacobiana com uma atualização eficiente da inversa da matriz após pequenas perturbações. Juntos, eles proporcionam uma abordagem eficiente para encontrar soluções aproximadas, podendo facilitar a resolução de problemas complexos.

Na próxima seção, exploraremos métodos recentes para sistemas não lineares, destacando os avanços significativos que ocorreram nos últimos anos. Esses desenvolvimentos têm levado ao surgimento de técnicas avançadas que aprimoram a resolução de sistemas não lineares, proporcionando soluções mais eficientes e precisas.

6.2.3 Outras Abordagens na Resolução de Sistemas Não Lineares

Nos últimos anos, tem havido avanços significativos no desenvolvimento de métodos avançados para resolver sistemas não lineares. Nesta seção, destacaremos quatro desses métodos, fornecendo uma visão simplificada de suas características e aplicabilidades. Todos os métodos citados aqui dentre outros podem ser encontrados com maiores detalhes em [17].

O Método de Potra-Pták é uma abordagem popular para sistemas não lineares que se baseia na iteração sequencial. Ele utiliza derivadas de primeira ordem, como o Método de Newton, para aproximar a solução. Uma de suas características notáveis é a ordem de convergência quadrática, o que significa que a precisão da solução duplica a cada iteração. Um método eficaz eficaz na resolução de sistemas não lineares. No entanto, a escolha do ponto inicial pode afetar a convergência.

O Método de Homotopia e Continuação é uma abordagem versátil que transforma um sistema não linear em uma sequência de sistemas auxiliares controlados por um parâmetro de continuação. À medida que o parâmetro varia de 0 a 1, o sistema auxiliar se transforma gradualmente no sistema original. Esse método é aplicável a uma ampla gama de problemas não lineares, incluindo sistemas com múltiplas soluções. A ordem de convergência pode variar, normalmente entre linear e quadrática, dependendo das estratégias utilizadas em cada etapa.

O Método do Ponto-Médio combina elementos do Método de Newton e do Método da Secante, proporcionando uma abordagem eficaz para sistemas não lineares, especialmente em alta dimensão. Ele opera iterativamente, atualizando as estimativas da solução com base em aproximações anteriores e na função Jacobiana, que envolve derivadas de primeira ordem. Embora sua ordem de convergência seja tipicamente linear, a simplicidade e o baixo custo computacional o tornam uma escolha interessante em algumas situações práticas,

onde o cálculo das derivadas de primeira ordem é computacionalmente acessível.

O Método de Chun é um método iterativo de quarta ordem para a resolução de equações não lineares, sem a necessidade de calcular derivadas de segunda ordem. Esse método combina características de métodos de descida e do Método de Newton, sendo eficaz na resolução de sistemas não lineares, incluindo problemas mal condicionados e singularidades.

À medida que os sistemas não lineares continuam a desempenhar um papel crítico em diversas disciplinas, o desenvolvimento de métodos eficazes e versáteis para sua resolução é essencial. Cada um dos métodos destacados oferece abordagens únicas para enfrentar problemas não lineares com diferentes características, fazendo uso de derivadas de primeira ordem na maioria dos casos. A escolha do método adequado depende das propriedades do sistema e dos recursos computacionais disponíveis. Com pesquisas contínuas, podemos esperar aprimoramentos e novas abordagens para enfrentar desafios cada vez mais complexos em sistemas não lineares.

7 Construindo uma Apostila

Sob a perspectiva dos estudos realizados anteriormente, elaboramos uma apostila com o objetivo de auxiliar alunos e professores do ensino médio na introdução ao pensamento computacional e no uso de métodos numéricos aplicados à resolução de problemas matemáticos, utilizando a linguagem de programação Python. Nosso propósito foi criar um material didático e acessível, que não apenas apresentasse conceitos fundamentais, mas também proporcionasse uma experiência prática por meio da construção de algoritmos e programas.

A apostila está organizada em cinco capítulos, cada um focado em aspectos específicos do aprendizado de programação e métodos numéricos, sempre com o suporte de recursos computacionais acessíveis, da seguinte forma:

1. Capítulo 1 - Apresentação
2. Capítulo 2 - Programação
3. Capítulo 3 - Funções
4. Capítulo 4 - Métodos Numéricos para Zeros de Funções
5. Capítulo 5 - Métodos Numéricos para Sistemas de Equações

No Capítulo 1 — *Apresentação* — introduzimos o propósito da apostila e a metodologia adotada, destacando a importância de combinar conceitos matemáticos com ferramentas de programação para potencializar o aprendizado.

O Capítulo 2 — *Programação* — é dedicado à uma breve introdução à linguagem de programação Python. Iniciamos apresentando as plataformas de programação que serão utilizadas ao longo da apostila, com destaque para o Google Colab, uma ferramenta online que permite a execução de códigos Python diretamente no navegador, e o aplicativo QPython3L, um aplicativo para smartphones, que não necessita estar conectado à internet para ser executado, possibilitando aos alunos programar em seus dispositivos móveis. A seguir, exploramos comandos básicos de entrada e saída de dados, fundamentais para a interação com o usuário e a manipulação de informações e abordamos também estruturas de repetição e controle de fluxo, essenciais para a criação de algoritmos. Para consolidar o aprendizado, propomos a criação de uma Tabuada Eletrônica Interativa e de um programa

que simula uma Venda Online, atividades que permitem aplicar os conceitos estudados e podem desenvolver a habilidade de construir programas úteis e funcionais.

O Capítulo 3 — *Funções* — concentra-se no estudo de funções, tanto no contexto matemático quanto em Python. Primeiramente, revisamos conceitos matemáticos essenciais relacionados à funções e zeros de funções. Em seguida, avançamos para a criação e utilização de funções em Python, demonstrando como essa prática permite modularizar e organizar códigos de maneira eficiente. Esse capítulo é importante para compreender a importância das funções na programação e sua aplicação em resolver problemas matemáticos.

No Capítulo 4 — *Métodos Numéricos para Zeros de Funções* — iniciaremos apresentando um método numérico para calcular raízes quadradas. Após abordamos métodos clássicos utilizados para calcular zeros de funções, como o *Método de Newton*, o *Método da Bisseção* e o *Método da Secante*. Cada método é explicado em detalhe, seguido de exemplos práticos implementados em Python. O uso desses métodos permite resolver equações complexas de maneira eficiente e precisa. Além disso, incluímos links diretos para o Google Colab, facilitando o acesso dos leitores aos códigos e permitindo a execução e experimentação dos exemplos apresentados.

O Capítulo 5 — *Métodos Numéricos para Sistemas de Equações* — foca na resolução de sistemas de equações, tanto lineares quanto não lineares. Iniciamos discutindo o *Método de Jacobi* para resolver sistemas de equações lineares, detalhando seu funcionamento e aplicabilidade. Em seguida, abordamos o *Método de Newton* para sistemas de equações não lineares, mostrando como essa técnica pode ser adaptada para resolver sistemas mais complexos. Este capítulo é importante para aqueles que buscam entender e aplicar métodos numéricos em problemas que envolvem múltiplas equações inter-relacionadas.

A importância dos temas abordados nesta apostila vem de encontro com o contexto educacional e tecnológico atual. A programação, por exemplo, se tornou uma habilidade importante, não apenas para aqueles que desejam seguir carreiras na área de tecnologia, mas também como uma ferramenta poderosa para resolver problemas em diversas áreas do conhecimento. O conhecimento sobre os métodos numéricos, por sua vez, permite que os alunos compreendam e apliquem técnicas que são amplamente utilizadas na ciência, engenharia, economia, e muitas outras disciplinas, para solucionar problemas complexos que não podem ser resolvidos de forma analítica. Ao capacitar os alunos com essas habilidades, podemos estar não apenas preparando-os para os desafios acadêmicos e profissionais, mas também despertando o interesse e a curiosidade por explorar e inovar em um mundo cada vez mais orientado pela tecnologia.

8 Considerações Finais

Esta dissertação explorou o papel importante do pensamento computacional e dos métodos numéricos na educação matemática, destacando sua relevância na formação de habilidades algorítmicas e lógicas, também citado na Base Nacional Comum Curricular (BNCC). Exploramos superficialmente a evolução do pensamento computacional, suas tendências e lacunas, e mostramos a importância do conhecimento sobre a precisão numérica através da aritmética computacional e erros de aproximação.

Os métodos numéricos discutidos, incluindo técnicas para encontrar zeros de funções e resolver sistemas de equações, foram analisados com maior profundidade, mostrando suas aplicações práticas, vantagens e desvantagens. A criação de uma apostila prática para professores, alunos e interessados mostrou como esses conceitos podem ser integrados no ensino, utilizando a linguagem Python para resolver problemas matemáticos complexos.

Esta dissertação não apenas pretende proporcionar um entendimento teórico, mas também oferece ferramentas práticas para a implementação do pensamento computacional e métodos numéricos no ambiente educacional. Ao unir teoria e prática, esperamos contribuir para a melhoria da educação matemática, podendo capacitar professores e alunos a enfrentar os desafios de um mundo tecnológico em constante evolução.

Referências

1 BORDINI, A. Computação na educação básica no Brasil: o estado da arte. *Revista de Informática Teórica e Aplicada*, v. 23, n. 2, p. 210–238, 2016.

2 BOUVIER, A.; GEORGE, M. *Dictionnaire des mathématiques*. Puf, 2005.

3 BRASIL. *Base Nacional Comum Curricular*. Brasília, 2018.

4 BURDEN, R.; FAIRES, D.; BURDEN, A. *Análise Numérica*. Cengage Learning, 2016.

5 CARVALHO, J. R. *Cálculo Numérico de Raízes e sua Aplicação no Ensino Básico*. 2022. Diss. (Mestrado) – CEFET. PROFMAT – Mestrado Profissional em Matemática em Rede Nacional.

6 CORRÊA, S. D. *O Uso de Métodos Numéricos em Problemas de Otimização: Aplicações No Ensino Médio*. 2016. Diss. (Mestrado) – UNICAMP. PROFMAT – Mestrado Profissional em Matemática em Rede Nacional.

7 COTRIM, J. V. T. *Sistemas Lineares e Matrizes em Planilhas Eletrônicas*. 2022. Diss. (Mestrado) – UESB. PROFMAT – Mestrado Profissional em Matemática em Rede Nacional.

8 FAUSTINO, W. R. *Sistemas Lineares Homogêneos de Recorrências Lineares de Primeira Ordem com Duas e Três Variáveis*. 2023. Diss. (Mestrado) – UECG. PROFMAT – Mestrado Profissional em Matemática em Rede Nacional.

9 FRANCO, N. M. B. *Cálculo Numérico*. Pearson, 2014.

10 GROVER, S.; PEA, R. Computational Thinking in K–12: A Review of the State of the Field. *Educational Researcher*, v. 42, n. 1, p. 38–43, 2013.

11 MARQUES, J. C. O. *Construção de Mosaicos Utilizando a Linguagem de Programação Scratch como Ferramenta para o Ensino de Geometria Plana*. 2019. Diss. (Mestrado) – UTFPR. PROFMAT – Mestrado Profissional em Matemática em Rede Nacional.

12 NETO, T. M. S. *Ajuste de Curvas Usando Métodos Numéricos*. 2018. Diss. (Mestrado) – UFG. PROFMAT – Mestrado Profissional em Matemática em Rede Nacional.

13 PAPERT, S. *Mindstorms: Children, Computers, and Powerful Ideas*. New York, NY: Basic Books, Inc., 1980.

14 PESENTE, G. M. *O Ensino de Matemática por Meio da Linguagem de Programação Python*. 2019. Diss. (Mestrado) – UTFPR. Programa de Pós-graduação em Ensino de Ciência e Tecnologia, Universidade Tecnológica Federal do Paraná.

15 RIBOLDI, S. M. O. *Construção de Mosaicos Utilizando a Linguagem de Programação Scratch como Ferramenta para o Ensino de Geometria Plana*. 2019. Diss. (Mestrado) – UFFS. PROFMAT – Mestrado Profissional em Matemática em Rede Nacional.

16 ROCHA, J. S. *A Implementação da Linguagem de Programação na Educação Escolar Utilizando o Scratch*. 2020. Diss. (Mestrado) – UFOPA. Instituto de Ciências da Educação, Universidade Federal do Oeste do Pará.

17 SOUZA, E. A. *Métodos Iterativos para Problemas Não Lineares*. 2015. Diss. (Mestrado) – UFF. Programa de Pós-graduação em Modelagem Computacional em Ciência e Tecnologia, Universidade Federal Fluminense.

18 SOUZA, J. E. *O Uso da Linguagem de Programação Python na Resolução de Problemas Matemáticos do Ensino Médio*. 2023. Diss. (Mestrado) – UECG. PROFMAT – Mestrado Profissional em Matemática em Rede Nacional.

19 VAZ, L. S. *Relações Métricas em Triângulos Retângulos Utilizando a Linguagem de Programação Scratch: Uma Abordagem para Atividades*. 2021. Diss. (Mestrado) – UTFPR. PROFMAT – Mestrado Profissional em Matemática em Rede Nacional.

20 VERONESE, J. F. *Métodos Numéricos de Zero de Funções Aplicados em Problemas de Taxa Interna de Retorno*. 2022. Diss. (Mestrado) – UTFPR. PROFMAT – Mestrado Profissional em Matemática em Rede Nacional.

21 WING, J. M. Computational Thinking. *Communications of the ACM*, v. 49, n. 3, p. 33–35, 2006.

Anexo I
A Apostila

Segue em anexo o texto completo da apostila desenvolvida neste trabalho.

Métodos Numéricos e Python no Ensino Médio

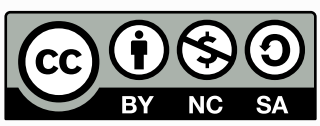
Edson Bertosa Lopes Santos
Luis Alberto D'Afonseca
Carlos Magno Martins Cosme

Métodos Numéricos e Python no Ensino Médio

Edson Bertosa Lopes Santos
Luis Alberto D'Afonseca
Carlos Magno Martins Cosme

Centro Federal dealign Educação Tecnológica de Minas Gerais – CEFET-MG
1 de novembro de 2024

Arte da capa: Fotografia de Designdrunkard baixada de Pexels



Sumário

Prefácio

Com imensa satisfação, apresentamos esta apostila dedicada à exploração dos métodos numéricos aplicados à resolução de desafios matemáticos com a utilização de recursos computacionais. Nosso objetivo é oferecer uma ferramenta versátil e eficaz para enriquecer sua jornada no mundo da matemática e da programação, independentemente de você ser um estudante entusiasta, um professor em busca de novas abordagens ou alguém curioso para explorar novos horizontes.

Como professor de matemática no Ensino Básico, percebo que os alunos frequentemente encontram dificuldades ao usar calculadoras para divisões e cálculos de raízes, como $1/3 \approx 0{,}333333$ ou $\sqrt{7} \approx 2{,}6457513$. Muitos ficam confusos com a precisão dos dígitos ou duvidam de seus próprios resultados. Isso me motivou a criar esta apostila, voltada para o estudo de métodos numéricos, que aborda técnicas para encontrar soluções aproximadas para problemas matemáticos onde cálculos diretos são inviáveis. Esses métodos permitem obter resultados suficientemente próximos dos valores corretos e são amplamente aplicáveis em várias áreas, como engenharia, física, economia e ciências da computação, oferecendo soluções práticas e eficientes para problemas complexos.

Nesta apostila, exploraremos desde os conceitos básicos de Python, uma linguagem de programação versátil e amplamente utilizada, até a aplicação de métodos numéricos para resolver problemas como zeros de funções e soluções de sistemas lineares e não lineares. Para melhor compreensão dos conceitos apresentados, é importante ter um conhecimento básico sobre funções, em nível de ensino médio. No entanto, não é necessário ter nenhum tipo de conhecimento prévio em programação, tornando o material acessível para alunos, professores e qualquer pessoa interessada em Matemática e Programação. A apostila oferece uma abordagem inclusiva e adaptável, com exemplos claros e exercícios práticos.

Agradecemos antecipadamente pelo seu interesse e dedicação em explorar conosco uma parte do vasto mundo da Matemática e da Programação . Venha descobrir

como a matemática e a tecnologia podem trabalhar em harmonia para solucionar problemas de forma eficaz e criativa!

1

Apresentação

Seja bem-vindo(a) à nossa apostila de introdução à linguagem de programação Python e aos métodos numéricos aplicados à resolução de problemas envolvendo zeros de funções e soluções de sistemas lineares e não lineares. Neste capítulo inicial, vamos apresentar de forma sucinta os temas abordados ao longo da apostila, material esse que pode contribuir para o desenvolvimento nessa área essencial, especialmente relevante nos dias atuais devido à crescente importância da tecnologia em diversos campos.

O Que é o Python e Por Que Estamos Trabalhando com Ele

O Python é uma linguagem de programação de alto nível, conhecida por sua simplicidade, legibilidade e versatilidade. É amplamente utilizada em diversas áreas, desde desenvolvimento de páginas para a internet até análise de dados e inteligência artificial.

Para começar a programar em Python, é necessário ter um ambiente de desenvolvimento configurado. Para isso vamos usar um aplicativo chamado QPython3L, para que seja possível programar no smartphone, ou uma plataforma online chamada Google Colab, que funciona tanto no smartphone como no computador, e não necessita de nenhum tipo de instalação.

No Capítulo 2, apresentaremos comandos básicos em Python, incluindo operações matemáticas, impressão de resultados e mensagens personalizadas, entrada e saída de dados, estruturas de repetição e controle de fluxo. Também desenvolveremos dois programas em Python, um que simula uma compra online, calculando o valor

total a ser pago por uma quantidade específica de produtos, e outro que gera a tabuada dinâmica de um número determinado inicialmente. Estes exemplos ajudarão a compreender como criar funções e utilizar estruturas de controle em Python. Posteriormente, exploraremos programas para resolver problemas matemáticos específicos, apresentando o código passo a passo e disponibilizando um link, que estará anexado ao código, para que o leitor possa acessá-lo diretamente na plataforma online do Google Colab.

Esa apostila não é um guia completo sobre Python. Para quem deseja se aprofundar mais na linguagem, recomendamos consultar o site oficial Python.org. Além disso, existem diversos cursos online disponíveis, como o Curso em Vídeo que pode ser uma excelente opção para expandir seus conhecimentos nessa linguagem de programação.

Funções

No Capítulo 3, vamos explorar o conceito de funções tanto na matemática quanto na programação. As funções desempenham um papel fundamental em ambos os campos, mas sua abordagem e aplicação podem variar significativamente.

Começaremos com uma revisão das funções na matemática, abordando conceitos como domínio, contradomínio, imagem, e definição formal de função. Veremos também diferentes tipos de funções, como lineares, quadráticas e trigonométricas, e como representá-las graficamente. Também falaremos sobre equações e um aspecto importante que é a busca por zeros de funções.

Em seguida, faremos uma transição para as funções na programação. Aqui, as funções são blocos de código que realizam tarefas específicas e podem ser reutilizadas em diferentes partes de um programa. Vamos aprender como definir funções em Python, passar argumentos para funções, e retornar valores de funções.

Finalmente, faremos uma comparação entre as funções na matemática e as funções na programação. Embora compartilhem o mesmo nome, as funções em cada domínio têm propósitos e características distintas. Compreender essas diferenças nos ajudará a usar as funções de forma eficaz em contextos matemáticos e de programação.

O Que São Métodos Numéricos Iterativos

Nos Capítulos 4 e 5, vamos explorar métodos numéricos iterativos. Um conjunto de técnicas que buscam encontrar a solução de um problema por meio de passos sucessivos, melhorando a aproximação a cada etapa. Esses métodos geralmente são utilizados quando não é possível encontrar uma solução direta através de métodos analíticos, permitindo-nos obter soluções aproximadas com a precisão desejada.

Nesta apostila, exploraremos uma variedade de métodos numéricos iterativos clássicos para solucionar problemas matemáticos. Usaremos o Método de Newton, o Método da Bisseção e o Método da Secante para a busca de raízes quadradas e zeros de funções, bem como o Método de Jacobi para sistemas lineares e novamente o Método de Newton para encontrar a solução de sistemas não lineares, explorando sua aplicação prática e sua importância no contexto matemático e computacional.

Estamos animados para começar essa jornada de aprendizado juntos. Vamos mergulhar no mundo da programação, métodos numéricos e descobrir como eles podem enriquecer o ensino da matemática!

2

Programação

Neste capítulo, embora não contenha uma introdução completa à programação, exploraremos de forma simplificada os comandos básicos da linguagem de programação Python, conhecida por sua simplicidade e versatilidade em diversas áreas, como ciência de dados, automação de tarefas, desenvolvimento web e de jogos, inteligência artificial, entre outras. Abordaremos as características que a torna uma ferramenta poderosa para resolver problemas matemáticos e científicos, mostrando de forma introdutória como escrever um programas e compreender estruturas de controle e dados para desenvolver algumas aplicações, podendo assim estimular a criatividade, o pensamento lógico e a resolução de problemas.

2.1 Plataformas de Programação

Nesta seção inicial, vamos apresentar as plataformas de programação que serão utilizadas nessa apostila. Após essa introdução às plataformas, seguiremos com os passos para o download, instalação e acesso às mesmas. Não se preocupe se você não tem experiência prévia em programação, este guia foi desenvolvido pensando em facilitar o aprendizado, fornecendo instruções claras e exemplos práticos, ajudando assim na compreensão dos processos executados.

Acesso pelo Smartphone

Para começar a programar em um smartphone, existem vários aplicativos disponíveis, tanto para IOS quanto para Android, tais como Python Code-Pad e Coding Python. No entanto, neste trabalho, optamos por utilizar o QPython3L. Uma plataforma Python gratuita e livre de anúncios, desenvolvida especialmente para dispositivos móveis Android. Uma vez instalado, o QPython3L pode ser utilizado sem a necessidade de conexão com a internet. Essa ferramenta é importante para facilitar o aprendizado da programação, especialmente considerando a grande intimidade que os jovens de hoje têm com seus smartphones. Ao ter acesso à programação através do dispositivo móvel, estamos conectando diretamente com o universo digital em que estamos imersos diariamente, podendo assim tornar o aprendizado mais acessível e dinâmico.

Vamos apresentar nesse momento os passos para fazer a instalação do aplicativo que será usado para escrever e executar o programa pelo smartphone.

1. Vá até a loja de apps do smartphone (Play Store - Android);
2. Procure pelo app 'QPython3L';
3. Clique em 'instalar' e após a instalação, clique em 'abrir';
4. Na página inicial do app, clique em 'Editor'.

A Figura 2.1 mostra todos os passos descritos anteriormente.

Figura 2.1: Instalação e acesso do aplicativo QPython3L.

Nesse momento, teremos uma página em branco onde vamos começar a escrever os códigos. Mas antes, é importante entender o que são código e *script* na programação, especialmente no contexto da linguagem Python.

O código refere-se às instruções escritas em uma linguagem de programação, como o Python, que são interpretadas ou compiladas para executar ações específicas no equipamento utilizado (computador, smatphones, entre outros). Essas instruções podem incluir operações matemáticas, manipulação de dados, controle de fluxo, entre outras funcionalidades. Já um *script*, por sua vez, é um conjunto de instruções ou comandos organizados em um arquivo de texto, geralmente com extensão ".py" no caso do Python. Esse arquivo contém o código que será executado pelo interpretador Python para realizar as tarefas programadas.

Antes de começarmos a escrever os códigos, vamos salvar nosso *script*, assim estamos guardando as instruções que escrevemos em um arquivo para que possamos executá-lo posteriormente e reutilizar o código desenvolvido. Essa prática é fundamental na programação, pois permite organizar e gerenciar projetos de forma eficiente. Para salvar o *script*, siga os passos:

1. Clique em 'salvar';
2. Clique no campo aberto para digitar o nome a ser dado para o arquivo que será salvo;
3. Escreva 'Teste.py';
4. Clique em 'ok'.

A Figura 2.2 mostra todos os passos descritos anteriormente.

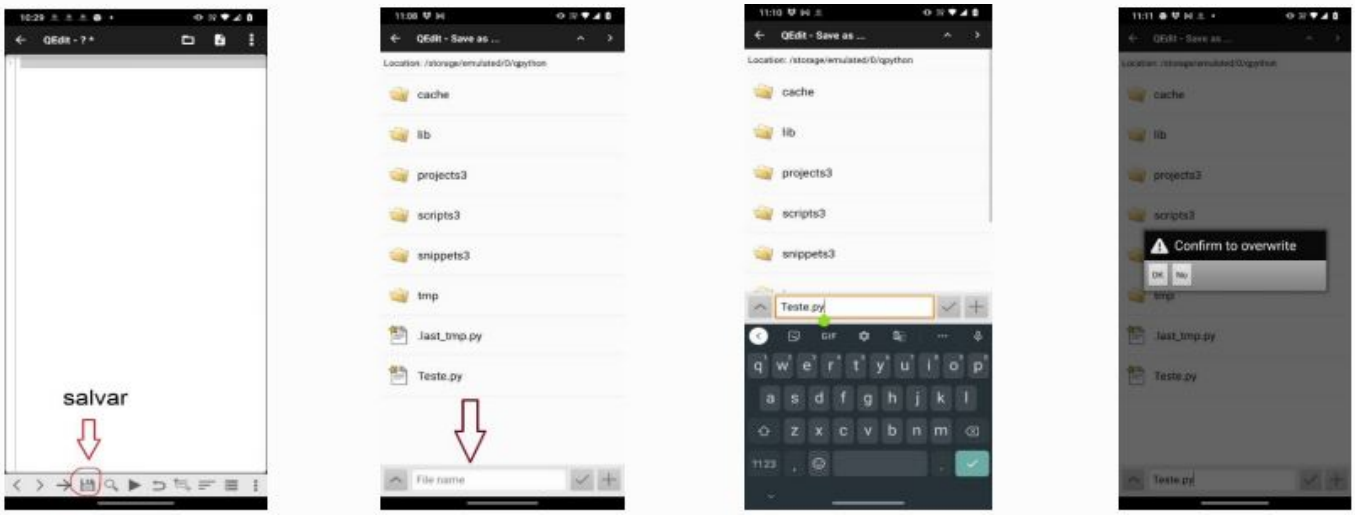


Figura 2.2: Salvando um *script* no aplicativo QPython3L.

Tendo executado os passos descritos, o *script* já estará salvo no smartphone. Você pode criar *scripts* para diferentes propósitos e nomeá-los como desejar. Para começar um novo *script*, com a pagina já aberta, siga os passos a seguir:

1. Click no símbolo +;
2. Click em 'script';
3. Digite 'Tabuada';
4. Click em 'ok'.

A Figura 2.3 mostra todos os passos descritos anteriormente.

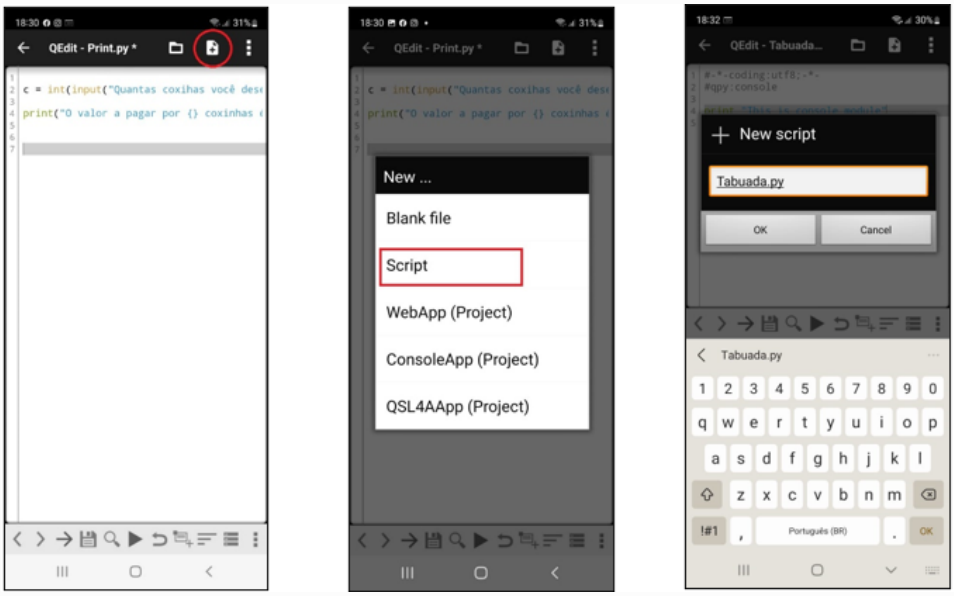


Figura 2.3: Criando um novo *script* no aplicativo QPython3L.

Nesse momento já estamos prontos para começar a programar através do app QPython3L no smartphone.

Acesso Online pelo Computador

Já para o uso online tanto para smartphones quanto para computadores, utilizaremos o Google Colab, uma plataforma que oferece um ambiente de desenvolvimento integrado para Python. A principal vantagem do Colab é a sua acessibilidade, pois não requer a instalação de nenhum software na máquina local, bastando apenas um aparelho com acesso à internet para começar a programar, o que o torna ideal para a sala de informática de uma escola, onde a disponibilidade de recursos técnicos pode variar. Essa abordagem online também promove a colaboração e o compartilhamento de projetos de forma fácil e eficiente.

Para acessar o Google Colab, basta ter uma conta do Google e acessar o site da plataforma pelo link Google Colab.

A Figura 2.4 mostra a tela inicial do Google Colab onde pode-se criar novos notebooks de Python ou abrir notebooks existentes diretamente no navegador.

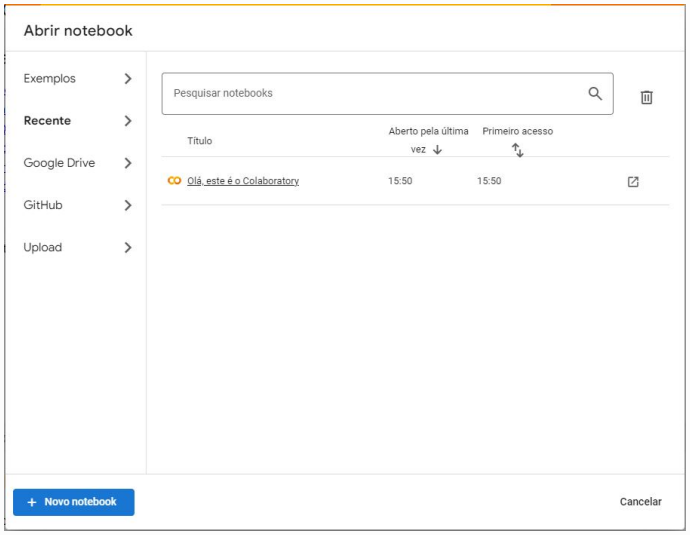


Figura 2.4: Página inicial do Google Colab.

Ao clicar no botão 'novo notebook' ou ao abrir um notebook já existente, teremos a plataforma onde serão escritos os códigos Python, mostrado na Figura 2.5 a seguir.

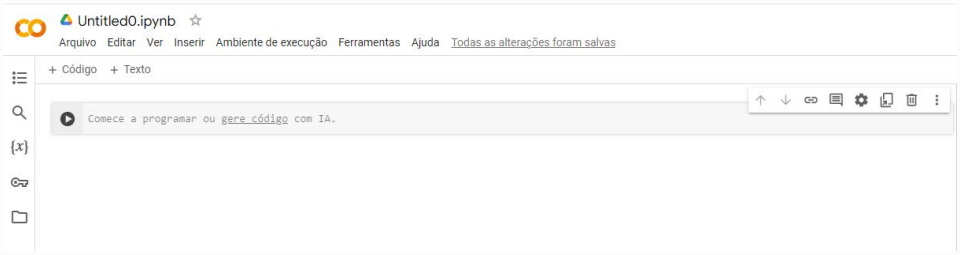


Figura 2.5: Plataforma de programação do Google Colab.

No Google Colab, você pode escrever código Python em células individuais. Para criar uma nova célula, clique no botão **'+ Codigo'** na barra de ferramentas. Em seguida, você pode digitar seu código Python na célula e executá-lo clicando no botão **'Run'** ao lado da célula.

Além de escrever e executar código Python, o Google Colab oferece diversos recursos adicionais, como suporte a bibliotecas populares, gráficos interativos, integração com

o Google Drive, entre outros. Você pode explorar esses recursos conforme avança em seus estudos de programação em Python.

Com esses conceitos básicos, você está pronto para começar a programar em Python utilizando o Google Colab.

Comandos Matemáticos Básicos

Os comandos matemáticos básicos em Python são essenciais para realizar operações numéricas. Abaixo, apresentamos uma tabela com alguns dos principais comandos.

Comando	**Descrição**
`+`	Adição
`-`	Subtração
`*`	Multiplicação
`/`	Divisão
`**`	Potenciação
`%`	Resto da divisão (módulo)
`=`	Atribuição de valor
`==`	Comparação de igualdade
`abs(x)`	Valor absoluto de x

Tabela 2.1: Comandos matemáticos básicos em Python

Esses comandos matemáticos básicos em Python formam a base para a realização de cálculos e manipulações numéricas em seus programas.

2.2 Entrada e Saída de Dados

Nesta seção, exploraremos os comandos básicos de entrada e saída de dados em Python de forma didática e gradual, detalhando todos os passos necessários para entender e aplicar esses conceitos. Iniciaremos com a construção de programas que interagem com o usuário, permitindo a coleta de informações e a exibição de resultados de maneira clara e organizada. Este processo facilita a compreensão desses conceitos fundamentais e pode capacitar o leitor a desenvolver habilidades práticas na criação de programas em Python, independentemente do seu nível de experiência em programação.

O Comando *print*

Vamos começar pelo comando `print`, sua função é mostrar um texto na tela. Vamos escrever um comando para que seja exposto na tela a frase "Olá mundo!". O comando deve incluir o texto entre aspas, que podem ser simples ou duplas. O texto entre aspas é chamado de *string* e o uso das aspas é necessário para que o texto apareça exatamente como foi digitado.

EXEMPLO 2.2.1: Escreva um código Python para exibir a frase "Olá mundo!" (código).

Vamos usar para isso o comando `print` que exibirá o que for escrito ou digitado dentro de aspas dentro de parênteses.

```
1 print("Olá mundo!")
```

Esse código gera o resultado apresentado a seguir.

```
Olá mundo!.
```

Vamos mostrar através do próximo exemplo a diferença entre um texto matemático escrito dentro de aspas e um escrito sem as aspas.

EXEMPLO 2.2.2: Escreva um código Python que mostre na tela a operação 5 + 5 na primeira linha e mostre o resultado da operação 5 + 5 na segunda linha (código).

Vamos usar para isso o comando `print`. Na primeira linha vamos colocar a operação dentro de aspas e na segunda linha vamos colocar a operação sem as aspas.

```
1 print("5+5")
2 print(5+5)
```

Esse código gera o resultado apresentado a seguir.

```
5+5
10
```

O programa exibiu `5 + 5` na primeira linha e `10` na segunda, que é o resultado da operação. Experimente outros exemplos, como `print(7+8)` ou `print(17-4+2)`, para

consolidar o aprendizado.

Agora, introduziremos conceitos algébricos utilizando variáveis como `x` e `y`, atribuindo a cada uma um valor numérico e realizando cálculos aritméticos com essas variáveis. Em Python, as variáveis funcionam como "caixinhas" onde armazenamos valores, que podem ser números, textos ou outros tipos de dados. É importante lembrar que as variáveis devem ser explicitamente atribuídas com um valor antes de serem usadas no programa.

EXEMPLO 2.2.3: Escreva um código Python para exibir o resultado da operação $x + y$, sendo $x = 2$ e $y = 3$ (código).

Vamos primeiramente atribuir os valores para as variáveis `x` e `y` e usar o comando `print` para exibir o resultado da operação.

```
1 x = 2
2 y = 3
3 print(x + y)
```

Esse código gera o seguinte resultado:

```
5
```

O programa exibiu na tela o número 5, resultado da operação $2 + 3$.

Nota: Se você não atribuir valores às variáveis `x` e `y` antes de tentar fazer a operação, o Python não saberá o que elas representam. Nesse caso, será gerado um erro chamado `NameError`, indicando que as variáveis não estão definidas. O erro será algo assim:

```
NameError: name 'x' is not defined
```

Portanto, é sempre necessário definir as variáveis com um valor antes de utilizá-las em operações.

Para o próximo exemplo vamos usar uma expressão um pouco mais complexa.

EXEMPLO 2.2.4: Escreva um código Python para exibir o resultado da operação $2a + 3b$ sendo $a = 4$ e $b = 7$ (código).

Vamos atribuir os valores para as variáveis `a` e `b` e usar o comando `print` para exibir o resultado da operação.

```
1 a = 4
2 b = 7
3 print( 2*a + 3*b)
```

Esse código gera o resultado apresentado a seguir.

```
29
```

O programa exibiu na tela o número 29, resultado de $2 \cdot 4 + 3 \cdot 7 = 8 + 21 = 29$. Se for do interesse do leitor, explore exemplos mudando os valores de `a` e `b` para que se consolide o aprendizado.

O próximo exemplo será mais prático. Vamos escrever um programa que mostra o valor a pagar de acordo com a quantidade de produtos que o cliente comprar.

EXEMPLO 2.2.5: Em uma lanchonete o preço de um salgado é de R$ 3,00. Escreva um programa que mostre o valor a pagar de acordo com a quantidade de salgados que o cliente comprar (código).

Vamos definir `n` como o número de salgados que o cliente vai comprar. Como cada salgado custa R$ 3,00, a expressão para o valor total a pagar será `3*n`. Supondo que o cliente compre 2 salgados, começaremos atribuindo o valor de 2 à `n` e usaremos o comando `print` para exibir o resultado da operação.

```
1 n = 2
2 print( 3*n )
```

Esse código gera o resultado apresentado a seguir.

```
6
```

O programa exibiu na tela o número 6, resultado de $3 \cdot 2 = 6$. Se for do interesse do leitor, explore mais exemplos mudando a quantidade de salgados comprados para que se consolide esse aprendizado.

O Comando *f-string*

Vamos explorar um comando importante em Python que facilita a formatação de saídas em programas: o comando `f-string`.

O comando `f-string` é utilizado para criar frases formatadas dinamicamente, permitindo inserir valores de variáveis diretamente em uma mensagem. Para usar esse comando, basta adicionar a letra `f` antes da *string* e incluir as variáveis dentro de chaves na mensagem. Vamos ver um exemplo para ilustrar como utilizar esse recurso.

EXEMPLO 2.2.6: Escreva um comando em Python que mostre na tela uma frase personalizada com o nome e a idade de uma pessoa (código).

Vamos criar as variáveis `nome` e `idade` e atribuir valores à elas, com o comando `f-string` colocaremos esses valores dentro do texto.

```
1 nome  = "João"
2 idade = 25
3
4 print(f"Olá, meu nome é {nome} e eu tenho {idade} anos.")
```

A saída desse código será:

```
Olá, meu nome é João e eu tenho 25 anos.
```

No exemplo acima, o `f-string` é utilizado para inserir dinamicamente os valores das variáveis `nome` e `idade` na frase, resultando em uma mensagem personalizada. Se trocarmos o valor dado às variáveis, a estrutura da frase não será alterada.

O Comando *input*

Vamos explorar um outro importante comando em Python que facilita a interação com o usuário: o comando `input`.

O comando `input` permite interagir diretamente com o usuário, solicitando que ele insira textos ou valores. Para lidar com números inteiros, use `int` para converter a entrada; para números com casas decimais, use `float`. Vamos ver um exemplo de como utilizar esse comando.

Exemplo 2.2.7: Escreva um código Python onde o programa irá solicitar o nome, a idade e a altura (em metros) do usuário e, ao ser executado, mostrará na tela uma frase personalizada com as informações recebidas (código).

Vamos iniciar o código solicitando ao usuário que digite seu nome, idade e altura, usando o comando `input` para obter esses valores e armazená-los nas variáveis `nome`, `idade` e `altura`. Em seguida, o programa exibirá uma mensagem de boas-vindas contendo essas informações. Para isso, utilizaremos o comando `print` em conjunto com o comando `f-string` para formatar a mensagem.

```
1 print("Digite seu nome e sua idade por favor.")
2 nome = input("Digite seu nome: ")
3 idade = int(input("Digite sua idade: "))
4 altura = float(input("Digite sua altura: "))
5 print(f"Olá, meu nome é {nome}, tenho {idade} anos e tenho
     {altura} metros de altura.")
```

Assumindo que o usuário digite o nome Alberto, para a idade o número inteiro 29 e para a altura o número 1,72, esse código gera o resultado apresentado a seguir.

```
Digite seu nome e sua idade por favor.
nome: Alberto
idade: 29
altura: 1.72
Olá, meu nome é Alberto, tenho 29 anos e
tenho 1.72 metros de altura.
```

Cada comando neste programa desempenha uma função específica. O comando `print` é utilizado para exibir mensagens, solicitando informações ao usuário. O comando `input` captura a entrada de dados, como nome, idade e altura, e retorna uma string. Usamos `int` e `float` para informar o programa sobre o tipo de atribuição que será dada para a variável, que no caso é um número inteiro e um número decimal respectivamente. Por fim, `print` exibe a mensagem final, incorporando as variáveis com a formatação `f-string`.

Com isso, finalizamos esta seção. Na próxima seção, apresentaremos alguns comandos básicos de controle de fluxo, que são essenciais para o desenvolvimento de programas em Python.

2.3 Repetições e Controle de Fluxo

Nesta seção, vamos explorar os comandos básicos de controle de fluxo em Python, como estruturas de decisão (`if`, `elif`, `else`) e estruturas de repetição (`for`, `while`). Esses comandos são fundamentais para criar programas mais dinâmicos e flexíveis, permitindo que o código tome decisões e execute ações repetidamente com base em condições específicas. Vamos aprender como usar essas estruturas de maneira eficiente para resolver problemas comuns de programação e criar programas mais complexos e interativos.

Os Comandos *list* e *range*

Antes de começar com os comandos citados, é importante conhecer um comando que auxilia na criação e execução de programas que envolvem repetições: o comando `range`. Esse comando gera sequências de números, o que é especialmente útil em *loops* e iterações. O `range` pode ser utilizado com um, dois ou três argumentos para criar intervalos variados:

- `range(fim)`: Gera uma sequência de 0 até `fim - 1`.
- `range(inicio, fim)`: Gera uma sequência de `inicio` até `fim - 1`.
- `range(inicio, fim, passo)`: Gera uma sequência de `inicio` até `fim - 1`, com intervalos de `passo`.

Vamos explorar alguns exemplos para entender o funcionamento desse comando (código).

```
1 print(range(6))
2 print(range(3, 10))
3 print(range(1, 20, 3))
```

A resposta do programa será:

```
range(0, 6)
range(3, 10)
range(1, 20, 3)
```

O comando `range` não gera uma lista com os elementos. Para utilizar a sequencia de números gerados, devemos armazená-los em uma lista e, em seguida, usar os comandos de manipulação, como o `print` para imprimir os elementos.

O comando `list` em Python é um tipo de variável que converte uma sequência de itens em uma lista, que é uma coleção ordenada e mutável de elementos de tipos variados, como números, *strings* e outras listas. A sintaxe básica é `list(sequencia)`, onde `sequencia` representa os itens a serem convertidos.

Vamos usar o comando `list` junto com o comando `range` para criar e exibir uma lista com os números de uma sequência (código).

```
1 print(list(range(6)))
2 print(list(range(3, 10)))
3 print(list(range(1, 20, 3)))
```

A resposta do programa será:

```
1 [0, 1, 2, 3, 4, 5]
2 [3, 4, 5, 6, 7, 8, 9]
3 [1, 4, 7, 10, 13, 16, 19]
```

Os programas geralmente não são escritos com essa estrutura; é mais comum atribuir valores a variáveis e trabalhar com elas. Veja o mesmo programa reescrito utilizando variáveis (link).

```
1 r1 = range(6)
2 r2 = range(3, 10)
3 r3 = range(1, 20, 3)
4
5 s1 = list(r1)
6 s2 = list(r2)
7 s3 = list(r3)
8
9 print(s1)
10 print(s2)
11 print(s3)
```

A resposta do programa será:

```
1 [0, 1, 2, 3, 4, 5]
2 [3, 4, 5, 6, 7, 8, 9]
3 [1, 4, 7, 10, 13, 16, 19]
```

- A sequência `s1` gera uma sequência de 0 até 5, porque o `range(6)` inclui números de 0 até 5 (6 é excluído).
- A sequência `s2` gera números de 3 até 9, porque o `range(3, 10)` começa em 3 e vai até 9 (10 é excluído).
- A sequência `s3` gera números de 1 até 19 (20 é excluído) com intervalos de 3, porque o `range(1, 20, 3)` começa em 1 e a cada passo adiciona 3 até alcançar ou ultrapassar 19. Qualquer valor acima de 19 será excluído.

Esses exemplos mostram de forma simplificada o funcionário dos comandos `list` e `range`, porém as listas são amplamente utilizadas na linguagem de programação Python e possuem diversas formas de serem utilizadas ou modificadas. Para mais informações sobre esse comando, uma opção pode ser o vídeo Usando Listas do canal Curso em Vídeo ou buscando informações no site oficial Python.org.

O Comando *for*

O comando `for` em Python é usado para criar um *loop* que itera sobre uma sequência de números. Ele é útil quando é necessário executar um bloco de código um número específico de vezes. Por exemplo, para criar a tabuada do 2, poderíamos usar um comando já estudado na Seção 2.2, o comando `print` e optar por escrever o código a seguir.

```
1 print(2*1)
2 print(2*2)
3 print(2*3)
4 print(2*4)
5 print(2*5)
6 print(2*6)
7 print(2*7)
8 print(2*8)
9 print(2*9)
10 print(2*10)
```

Teríamos na tela todos os resultados da tabuada do 2. Porém, dá muito trabalho repetir todos esses comandos para todos os números da tabuada. Para reduzir tarefas que exigem a repetição de ações similares, usamos o comando `for`. A sintaxe básica é

```
for variavel in range(inicio, fim, passo):,
```

onde `variavel` é a variável que assume o valor de cada item na sequência a cada iteração do *loop*, `inicio` é o número inicial da sequência (incluído no *loop*), `fim` é o

número final da sequência (excluído do loop) e `passo` é o intervalo entre os números na sequência (padrão é 1 se não especificado).

Vamos aplicar o comando `for` para substituir todos os comandos `print` digitados anteriormente para conseguirmos os valores da tabuada do número 2.

EXEMPLO 2.3.1: Escreva um código Python que mostra todos os resultados da tabuada do número 2 (código).

O comando `for x in range(1,11):` faz com que o comando digitado logo abaixo seja repetido e a variável `x` receba os valores de 1 até 10, pois o último valor (11) não é atribuído. Para obtermos os resultados da tabuada do número 2, o código será

```
1 for x in range(1, 11):
2     print(2*x)
```

Esse código gera o resultado apresentado a seguir.

```
2
4
6
8
10
12
14
16
18
20
```

Observe que com esse comando conseguimos todos os resultados da tabuada do 2 de forma mais simples. Esse é um dos comandos principais da linguagem Python.

Para que a tabuada seja mostrada de forma mais completa, o ideal seria ser exposta na forma $2 \times 1 = 2$. Para isso vamos usar o comando `f-string` para mostrar a tabuada de forma mais completa. Veja o exemplo a seguir.

EXEMPLO 2.3.2: Escreva um código Python que mostra a tabuada do número 2 de forma completa (código).

Vamos usar o comando `for x in range` para atribuir os valores à variavel e em seguida vamos usar o comando `print` e o comando `f-string` para que o programa exiba a tabuada formatada.

```
1 for x in range(1,11):
2     print(f"2 x {x} = {2*x}")
```

Esse código gera o resultado apresentado a seguir.

```
2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
2 x 4 = 8
2 x 5 = 10
2 x 6 = 12
2 x 7 = 14
2 x 8 = 16
2 x 9 = 18
2 x 10 = 20
```

O resultado será exatamente como descrevemos. Para ver a tabuada de outros números basta substituir o número 2 pelo número desejado.

Esses exemplos apresentaram de forma simplificada o comando `for`, porém podemos utilizá-lo de diversas formas diferentes. Para mais informações sobre esse comando, uma opção pode ser o vídeo Estrutura de repetição for do canal Curso em Vídeo ou buscar informações no site oficial Python.org.

O Comando *if*

O comando `if` funciona como um filtro que permite a execução de determinadas ações apenas se uma condição específica for verdadeira. É como uma bifurcação na estrada: se a condição for atendida, o programa segue por um caminho específico, executando o bloco de código dentro do `if`; caso contrário, ele continua por outro caminho. Por exemplo, podemos usar o `if` para verificar se um número é positivo ou negativo e executar diferentes ações com base nessa verificação.

Exemplo 2.3.3: Crie um programa em Python que verifica se um número digitado pelo usuário é positivo, negativo ou zero (código).

Vamos utilizar o comando `input` para que o usuário digite um número, o comando `if` para verificar a condição e o comando `print` para imprimir uma mensagem correspondente.

```
numero = int(input("Digite um número: "))

if numero > 0:
    print("O número é positivo.")
elif numero < 0:
    print("O número é negativo.")
else:
    print("O número é zero.")
```

Supondo que o usuário digite o número -3, a resposta do programa será

```
Digite um número: -3
O número é negativo.
```

Nesse código, `input` solicita um número inteiro ao usuário. O comando `if` verifica se o número é maior que zero e imprime "O número é positivo." Se não, `elif` verifica se é menor que zero e imprime "O número é negativo." Caso contrário, `else` imprime "O número é zero." O comando `print` exibe a mensagem correspondente.

Este exemplo mostra de forma simplificada o funcionamento do comando `if`. Para mais informações sobre esse comando, uma opção pode ser o vídeo Condições (Parte 1) do canal Curso em Vídeo ou busque informações no site oficial Python.org.

O Comando *while*

O comando `while` em Python é como uma porta que permanece aberta enquanto uma condição específica for verdadeira. Isso significa que o código dentro do bloco de `while` será repetido continuamente até que a condição não seja mais atendida, momento em que o programa segue para o próximo bloco de código após o `while`. Vamos ver um exemplo de aplicação desse comando.

EXEMPLO 2.3.4: Crie um programa em Python que pede ao usuário para adivinhar um número de 1 a 5. O programa continua solicitando uma entrada até que o usuário acerte o número correto, finalizando com uma mensagem (código).

Vamos usar o *loop* `while` para continuar pedindo ao usuário para adivinhar um número até que ele digite o número correto (neste caso, 3). Uma mensagem de sucesso é exibida quando o usuário acerta.

```
numero_secreto = 3
adivinhacao    = 0

while adivinhacao != numero_secreto:
    adivinhacao = int(input("Adivinhe um número de 1 a 5: "))
    if adivinhacao != numero_secreto:
        print("Tente novamente!")

print("Parabéns! Você adivinhou o número correto.")
```

Supondo que o usuário digitou como tentativas os números 2, 4 e 3, a resposta do programa será

```
Adivinhe um número de 1 a 5: 2
Tente novamente!
Adivinhe um número de 1 a 5: 4
Tente novamente!
Adivinhe um número de 1 a 5: 3
Parabéns! Você adivinhou o número correto.
```

Este código utiliza um *loop* `while` para continuar solicitando ao usuário que adivinhe um número de 1 a 5 até que o número correto seja digitado. O bloco de `if` verifica se a variável `adivinhacao` é diferente do `numero_secreto` e, se for, imprime uma mensagem para tentar novamente. Quando a variável `adivinhacao` se iguala ao `numero_secreto`, o *loop* termina e uma mensagem de parabéns é exibida.

Este exemplo mostra de forma simplificada o funcionamento do comando `while`. Para mais informações sobre esse comando, uma opção pode ser o vídeo Estrutura de repetição while do canal Curso em Vídeo ou busque informações no site oficial Python.org.

2.4 Consolidando o Aprendizado com Exemplos Práticos

Para consolidar o que aprendemos até aqui, vamos criar dois programas práticos que integrarão todos os comandos e conceitos que exploramos até agora. Estes exemplos finais têm o objetivo de mostrar como os comandos básicos de entrada e saída de dados, operadores matemáticos, comandos de repetição e de controle de fluxo podem ser aplicados em situações do dia a dia e resolver problemas concretos.

Tabuada Eletrônica Interativa

Se retomarmos o programa que gera a tabuada de um número usando o comando `for`, conforme descrito em 2.3, podemos melhorar a interação com o usuário ao incluir o comando `input`. Isso permitirá que o usuário digite o número da tabuada desejada. Após exibir a tabuada, podemos utilizar os comandos `if` e `while` para perguntar se o usuário gostaria de ver a tabuada de outro número. Veja o exemplo a seguir.

EXEMPLO 2.4.1: Crie um programa em Python que solicite um número ao usuário e mostre a tabuada desse número. Após a execução do programa, pergunte ao usuário se deseja ver a tabuada de outro número. Se a resposta for sim, o programa será executado novamente; caso contrário, será exibida uma mensagem de encerramento (código).

Vamos chamar de `n` o número do qual o usuário deseja ver a tabuada. Usaremos um `while` para criar um *loop*. O comando `input` solicitará ao usuário o número desejado. O comando `for` criará a tabuada e, após mostrá-la na tela, o programa perguntará se o usuário deseja ver outra tabuada, novamente usando `input`. Se a resposta for "Sim", o programa reinicia. Caso contrário, o programa imprime "Programa encerrado." e sai do *loop*, encerrando a execução.

É importante notar a diferença entre o símbolo de atribuição `=` e o símbolo de comparação `==`: o primeiro é utilizado para atribuir um valor a uma variável, enquanto o segundo é usado para comparar se dois valores são iguais.

```
1 resposta = "Sim"
2
3 while resposta == "Sim":
```

```
4     n = int(input("De qual número você deseja ver a sua
    tabuada? "))
5
6     for x in range(1, 11):
7         print(f"{n} x {x} = {n*x}")
8
9     resposta = input("Deseja ver a tabuada de outro número? ")
10
11 print("Programa encerrado.")
```

O programa inicia perguntando ao usuário de qual número deseja-se ver a tabuada. Assumindo que o usuário deseja ver a tabuada do número 7 e em seguida a tabuada do número 9, o resultado do código apresentado será.

```
De qual número você deseja ver a sua tabuada? 7

7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70

Deseja ver a tabuada de outro número? Sim
De qual número você deseja ver a sua tabuada? 9

9 x 1 = 9
9 x 2 = 18
9 x 3 = 27
9 x 4 = 36
9 x 5 = 45
9 x 6 = 54
9 x 7 = 63
9 x 8 = 72
9 x 9 = 81
9 x 10 = 90
```

```
Deseja ver a tabuada de outro número? Não
Programa encerrado.
```

Agora podemos ver a tabuada do número inicialmente digitado e após, podemos digitar um outro número qualquer que o programa vai gerar uma nova tabuada.

Programa de Compra Online

Nosso exemplo consiste em criar um programa de compra online. O programa informa ao usuário o valor do produto e solicita o número desejado. Após calcular o total a pagar, o programa solicita a confirmação da compra.

EXEMPLO 2.4.2: Crie um programa em Python que informe ao usuário o valor do produto e solicite o número desejado. O programa deve calcular e exibir o valor total a ser pago e pedir a confirmação da compra (código).

Vamos utilizar o comando `input` para capturar o número de produtos desejados, calcular o total multiplicando o número de produtos pelo preço unitário com `*`, e exibir o resultado usando `print`. Em seguida, vamos utilizar `input` novamente para solicitar a confirmação da compra. Vamos utilizar `if-else` para analisar a resposta do usuário.

```
continuar = "Sim"
while continuar == "Sim":
    print("O produto custa 12 reais cada.")
    n= int(input("Quantos produtos o Sr(a) deseja comprar? "))

    total = 12 * n
    print(f"Total a pagar por {n} produtos: {total} reais.")

    resposta = input("Deseja confirmar a compra? (Sim/Não): ")

    if resposta == "Sim":
        print("Compra confirmada. Obrigado!")
        continuar = "Não"
    else:
        print("Compra não confirmada. Reiniciando o
    processo.")
```

Assumindo que o usuário insira inicialmente o número 10 e depois mude para 6 antes de confirmar a compra, o código gera o resultado apresentado abaixo:

```
O produto custa 12 reais cada.
Quantos produtos o Sr(a) deseja comprar? 10
Total a pagar por 10 produtos: 120 reais.
```

```
Deseja confirmar a compra? (Sim/Não): Não
Compra não confirmada. Reiniciando o processo.

O produto custa 12 reais cada.
Quantos produtos o Sr(a) deseja comprar? 6
Total a pagar por 6 produtos: 72 reais.
Deseja confirmar a compra? (Sim/Não): Sim
Compra confirmada. Obrigado!
```

Temos agora um programa que simula uma compra online, informando ao usuário o valor do produto e solicitando o número desejado. Após, calcula o total a pagare solicita a confirmação da compra.

Ao final deste capítulo sobre programação em Python, concluímos nossa jornada desde os comandos de "Entrada e Saída de Dados", onde exploramos conceitos fundamentais como declaração de variáveis e operações aritméticas, até comandos para "Repetições e Controle de Fluxo", que nos permitiu praticar estruturas de repetição e a lógica algorítmica. Nesses programas usamos alguns comandos básicos e de extrema importância para a programação na linguagem Python, outros comandos Python serão apresentados nos capítulos seguintes e também vamos discutir sobre funções e como criar um programa para solucionar alguns problemas matemáticos específicos.

3

Funções

Neste capítulo, exploraremos funções, tanto na Matemática quanto na programação em Python. As funções são uma parte essencial do entendimento do mundo ao nosso redor, permitindo-nos descrever padrões, modelar fenômenos naturais e resolver problemas complexos.

Começaremos com uma revisão de funções na Matemática, abordando conceitos como domínio, contradomínio, imagem e definição formal de função. Em seguida, exploraremos funções afins, quadráticas, exponenciais e trigonométricas, discutindo algumas de suas propriedades e apresentando seus gráficos no plano cartesiano. Após essa revisão, abordaremos equações e zeros de funções, explorando algumas técnicas utilizadas para encontrar os valores de x que satisfazem $f(x) = 0$. Em seguida, faremos a transição para funções em Python, que são blocos de código reutilizáveis para executar tarefas específicas em diferentes partes de um programa.

Ao comparar as funções na Matemática com as funções em Python, entenderemos melhor as semelhanças e diferenças entre esses dois domínios, ampliando nossa capacidade de usar funções de forma eficiente em contextos matemáticos e de programação.

3.1 Funções na Matemática

Uma função é uma relação matemática que associa elementos de um conjunto de origem (domínio) à elementos de um conjunto de destino (contradomínio). Ela descreve como cada elemento do domínio está relacionado a um único elemento do contradomínio. Por exemplo, vamos considerar uma função f que associa mercadorias à seus preços.

$$f(\text{Mercadoria}) = \text{Preço}.$$

Aqui está uma tabela mostrando alguns exemplos de mercadorias e seus preços associados.

Mercadoria(kg)	**Preço (R$)**
Maçã	7,49
Banana	4,79
Laranja	2,49
Arroz	4,99
Açúcar	6,29
Feijão	7,49

Nesta tabela, cada mercadoria do domínio está associada a um único preço no contradomínio, ou seja, uma mercadoria não pode ter dois valores diferentes. Isso significa que cada item está em um e somente um preço, enquanto no contradomínio os preços podem se repetir, como por exemplo, a maçã e o feijão têm preços iguais a R$ 7,49. Além disso, a imagem da função f, que é o conjunto de todos os preços que a função pode produzir a partir dos itens do domínio, pode ser diferente do contradomínio, nesse exemplo o contradomínio eram todos os preços possíveis, porém a imagem são somente os preços mostrados na tabela.

Formalmente, uma função $f\colon A \to B$ é uma regra que associa a cada elemento x do conjunto A um único elemento y do conjunto B. O domínio A é o conjunto de todos os valores possíveis para x, enquanto o contradomínio B é o conjunto de todos os valores possíveis para y. A imagem da função f, denotada por $\text{Im}(f)$ ou $f(A)$, é o conjunto de todos os valores que f produz a partir dos elementos de A.

Esses conceitos são essenciais para compreendermos como as funções operam e como podemos utilizar suas propriedades para resolver problemas matemáticos de forma eficiente e precisa. Vale ressaltar que as funções podem representar fenômenos complexos do mundo real, como a valorização de ações na bolsa ao longo do tempo ou a relação da quantidade de chuva em uma cidade com as estações do ano. Esses exemplos evidenciam a versatilidade das funções e sua aplicabilidade na modelagem de diversos fenômenos reais.

Função Afim

Uma função afim é da forma

$$f(x) = ax + b,$$

onde a e b são constantes reais, sendo $a \neq 0$. Como x pode receber qualquer valor real, a função representa uma reta no plano cartesiano, onde a determina a inclinação da reta em relação ao eixo horizontal (eixo x) e b indica o ponto de interseção com o eixo vertical (eixo y).

A Figura 3.1 apresenta o gráfico da função $f(x) = 2x + 3$ no plano cartesiano (código).

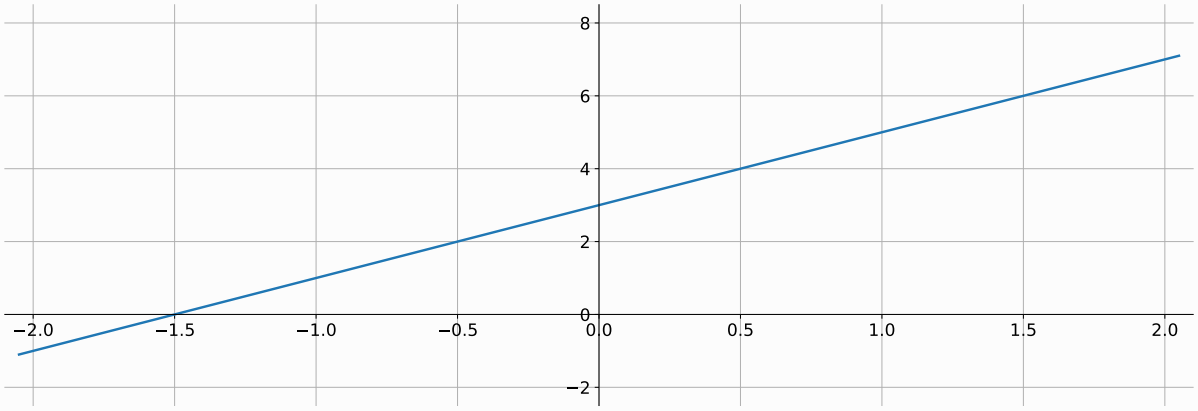


Figura 3.1: Gráfico da função $f(x) = 2x + 3$ no plano cartesiano.

Esse gráfico mostra que se trata de uma reta com inclinação positiva de 2 unidades para cada unidade no eixo x, e intersecta o eixo y em 3. Para determinar os pontos pertencentes ao gráfico dessa função, basta atribuir valores específicos para x e realizar os cálculos correspondentes. Vamos fazer isso no exemplo a seguir.

Exemplo 3.1.1: Determine alguns pontos pertencentes ao gráfico da função $f(x) = 2x + 3$.

Para determinar alguns pontos pertencentes ao gráfico da função $f(x) = 2x + 3$, vamos calcular os valores de $f(x)$ para $x = -2, -1, 0, 1, 2$.

$$\begin{aligned} f(-2) &= 2(-2) + 3 = -1 \\ f(-1) &= 2(-1) + 3 = 1 \\ f(0) &= 2 \cdot 0 + 3 = 3 \\ f(1) &= 2 \cdot 1 + 3 = 5 \\ f(2) &= 2 \cdot 2 + 3 = 7 \end{aligned}$$

Portanto, os pontos $(-2, -1)$, $(-1, 1)$, $(0, 3)$, $(1, 5)$ e $(2, 7)$ pertencem ao gráfico da função $f(x) = 2x + 3$.

Podemos criar um programa em Python para calcular os valores de $f(x)$ como mostra o exemplo a seguir.

Exemplo 3.1.1: Escreva um programa em Python para calcular os valores de $f(x) = 2x + 3$ para para alguns valores específicos de x (código).

Vamos escrever um código para calcular os valores da função $f(x) = 2x + 3$ para $x = -2, -1, 0, 1$ e 2. O código será:

```
1 # Atribuindo valores para x.
2 x_valores = [-2, -1, 0, 1, 2]
3
4 # Efetuando os cálculos correspondentes.
5 for x in x_valores:
6     f_x = 2*x + 3
7     print(f"f({x}) = {f_x}")
```

Este código calcula os valores de $f(x)$ para cada valor de x especificado na lista `x_valores` e imprime os resultados. A resposta do programa será:

```
1 f(-2) = -1
2 f(-1) =  1
3 f(0)  =  3
4 f(1)  =  5
5 f(2)  =  7
```

Assim, podemos ver como a função $f(x) = 2x + 3$ é avaliada para diferentes valores de x usando Python.

Função Quadrática

A função quadrática

$$f(x) = ax^2 + bx + c,$$

onde a, b e c são constantes reais e $a \neq 0$. O gráfico dessa função é uma parábola no plano cartesiano e está é utilizada para modelar diversos fenômenos naturais e artificiais. Vamos explorar as propriedades das funções quadráticas baseado e seus coeficientes.

A concavidade da parábola é determinada pelo coeficiente a. Se $a > 0$, a parábola tem concavidade para cima, e se $a < 0$, a parábola tem concavidade para baixo e c indica o ponto de interseção com o eixo vertical (eixo y).

As Figuras 3.2 e 3.3 mostram o gráfico de duas funções quadráticas, uma com concavidade para cima ($a > 0$) e outra com concavidade para baixo ($a < 0$).

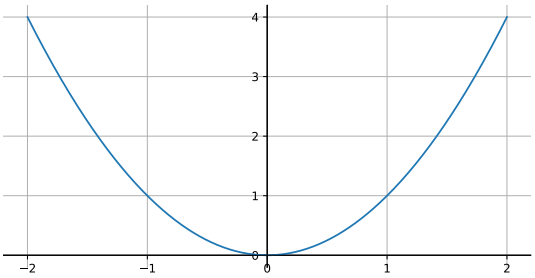


Figura 3.2: Concavidade para cima $(a > 0)$.

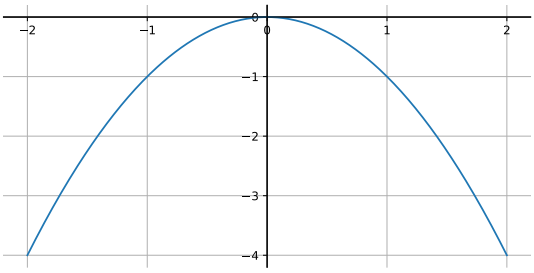


Figura 3.3: Concavidade para baixo $(a < 0)$.

A Figura 3.4 mostra o gráfico da função $f(x) = x^2 - 6x + 5$ no plano cartesiano (código).

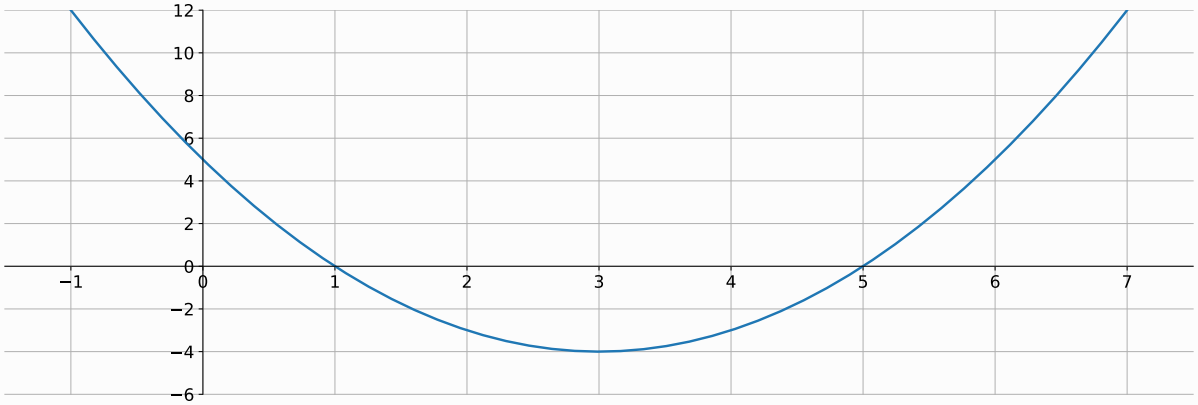


Figura 3.4: Gráfico da função $f(x) = x^2 - 6x + 5$ no plano cartesiano.

Esse gráfico mostra que se trata de uma parábola com concavidade voltada para cima, e intersecta o eixo y em 5. Vamos determinar alguns pontos pertencentes a esse gráfico atribuindo valores específicos para x e realizando os cálculos correspondentes. Vamos fazer isso no exemplo a seguir.

EXEMPLO 3.1.2: Determine alguns pontos pertencentes ao gráfico da função $f(x) = x^2 - 6x + 5$.

Para determinar alguns pontos pertencentes ao gráfico da função $f(x) = x^2 - 6x + 5$, vamos calcular os valores de $f(x)$ para $x = -2, -1, 0, 1, 2$.

$$\begin{aligned} f(-2) &= (-2)^2 - 6(-2) + 5 = 21 \\ f(-1) &= (-1)^2 - 6(-1) + 5 = 12 \\ f(0) &= 0^2 - 6 \cdot 0 + 5 = 5 \\ f(1) &= 1^2 - 6 \cdot 1 + 5 = 0 \\ f(2) &= 2^2 - 6 \cdot 2 + 5 = -3 \end{aligned}$$

Portanto, os pontos $(-2, 21)$, $(-1, 12)$, $(0, 5)$, $(1, 0)$ e $(2, -3)$ pertencem ao gráfico da função $f(x) = x^2 - 6x + 5$.

Podemos criar um programa em Python para calcular os valores de $f(x)$ como mostra o exemplo a seguir.

Exemplo 3.1.2: Escreva um programa em Python para calcular os valores de $f(x) = x^2 - 6x + 5$ para para alguns valores específicos de x (código).

Vamos escrever um código para calcular os valores da função $f(x)$ para $x = -2, -1, 0, 1, 2, 3, 4, 5$. O código será:

```
# Atribuindo valores para x.
x_valores = [-2, -1, 0, 1, 2, 3, 4, 5]

# Efetuando os cálculos correspondentes.
for x in x_valores:
    f_x = x**2 - 6*x + 5
    print(f"f({x}) = {f_x}")
```

Os resultados serão:

```
f(-2) =  21
f(-1) =  12
 f(0) =  5
 f(1) =  0
 f(2) = -3
 f(3) = -4
 f(4) = -3
 f(5) =  0
```

Assim, podemos ver como a função $f(x) = x^2 - 6x + 5$ é avaliada para diferentes valores de x usando Python.

Função Exponencial

As funções exponenciais do tipo $f(x) = ab^x$, onde $b > 0$ e $b \neq 1$, são importantes na matemática e na modelagem de fenômenos como crescimento populacional, decaimento radioativo e propagação de epidemias. O coeficiente a controla a amplitude da função, determinando $f(0)$ e influenciando a altura da curva em relação ao eixo x. A base b define o comportamento da função: quando $b > 1$, $f(x)$ apresenta crescimento exponencial conforme x aumenta; quando $0 < b < 1$, a função representa um decaimento exponencial. As Figuras 3.5 e 3.6 mostra os gráficos das funções $f(x) = 2^x$ e $f(x) = \left(\frac{1}{2}\right)^x$, ilustrando esses comportamentos (código).

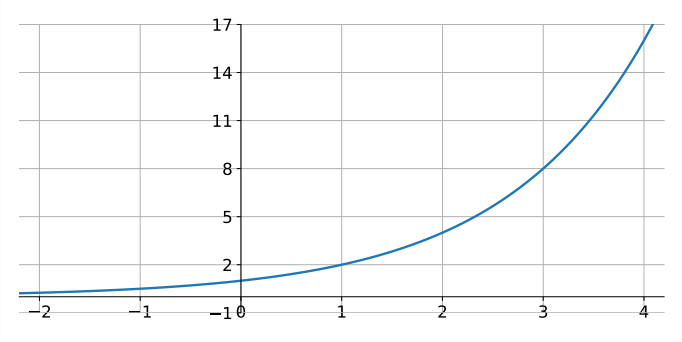


Figura 3.5: Gráfico da função $f(x) = 2^x$.

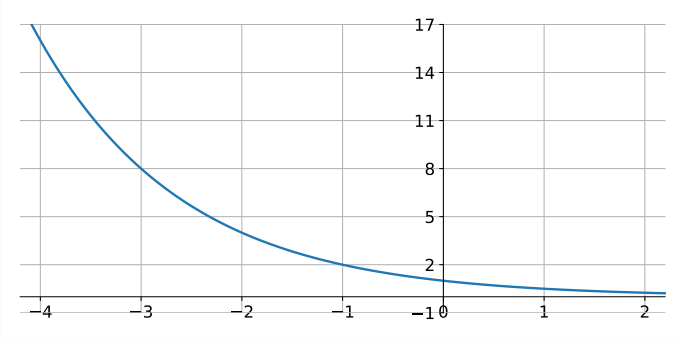


Figura 3.6: $f(x) = (1/2)^x$.

Essas funções exponenciais são amplamente utilizadas na modelagem matemática de fenômenos naturais, financeiros e científicos, sendo uma ferramenta importante em áreas como economia, biologia, física, engenharia, entre outras.

Vamos ver um exemplo considerando a função $f(x) = 2 \cdot 3^x$. A Figura 3.7 apresenta o gráfico dessa função no plano cartesiano (código).

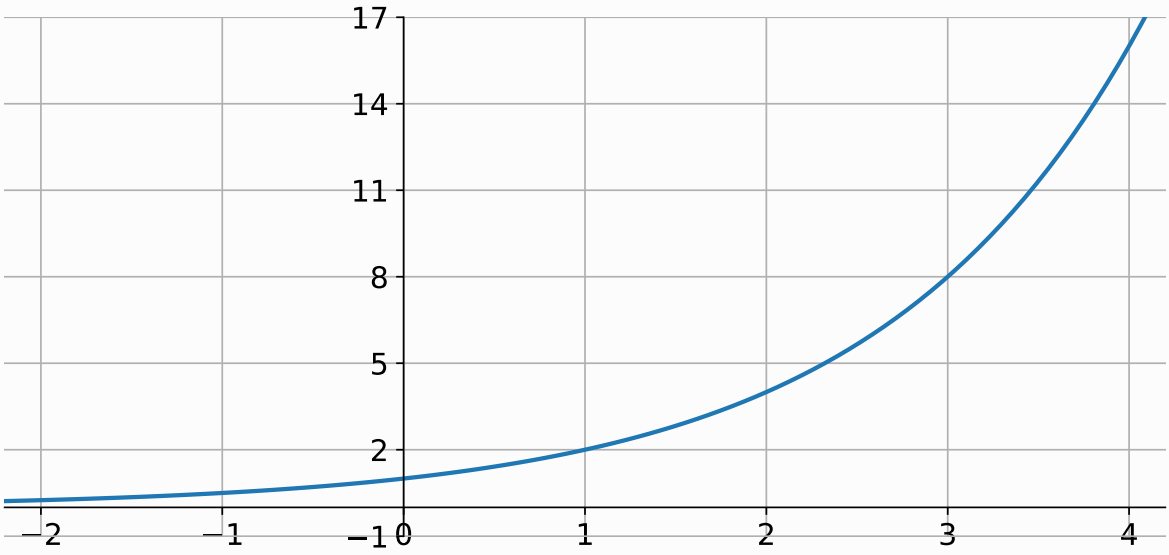


Figura 3.7: Gráfico da função $f(x) = 2 \cdot 3^x$ no plano cartesiano.

O gráfico representa um crescimento exponencial à medida que os valores de x aumentam. Vamos determinar alguns pontos pertencentes a esse gráfico atribuindo valores específicos para x e realizando os cálculos correspondentes. Vamos fazer isso no exemplo a seguir.

EXEMPLO 3.1.3: Determine alguns pontos pertencentes ao gráfico da função $f(x) = 2 \cdot 3^x$.

Para determinar alguns pontos pertencentes ao gráfico da função $f(x) = 2 \cdot 3^x$, vamos calcular os valores de $f(x)$ para $x = -1,\ 0,\ 1, 2,\ 3$.

$$\begin{aligned} f(-1) &= 2 \cdot 3^{-1} = \frac{2}{3} \\ f(0) &= 2 \cdot 3^0 = 2 \\ f(1) &= 2 \cdot 3^1 = 6 \\ f(2) &= 2 \cdot 3^2 = 18 \\ f(3) &= 2 \cdot 3^3 = 54 \end{aligned}$$

Portanto, os pontos $\left(-1, \frac{2}{3}\right)$, $(0, 2)$, $(1, 6)$, $(2, 18)$ e $(3, 54)$ pertencem ao gráfico da função $f(x) = 2 \cdot 3^x$.

Podemos criar um programa em Python para calcular os valores de $f(x)$ como mostra o exemplo a seguir.

EXEMPLO 3.1.3: Escreva um programa em Python para calcular os valores de $f(x) = 2 \cdot 3^x$ para para alguns valores específicos de x (código).

Vamos escrever um código para calcular os valores da função $f(x)$ para $x = -1, 0, 0.25, 1, 1.49, 2, 3$. O código será:

```
1 # Atribuindo valores para x.
2 x_valores = [-1, 0, 0.25, 1, 1.49, 2, 3]
3
4 # Efetuando os cálculos correspondentes.
5 for x in x_valores:
6     f_x = 2 * 3**x
7     print(f"f({x}) = {f_x}")
```

Os resultados serão:

```
1 f(-1)   = 0.6666
2 f(0)    = 2
3 f(0.25) = 2.6321
4 f(1)    = 6
5 f(1.49) = 10.278
```

```
6 f(2)    = 18
7 f(3)    = 54
```

Aqui, podemos observar claramente uma das vantagens da programação. Determinar os valores da função $f(x)$ para $x = 0{,}25$ ou $x = 1{,}49$ não seria uma tarefa simples de realizar manualmente. Utilizando técnicas de programação, conseguimos obter esses valores de forma rápida e precisa, destacando a eficiência da computação em resolver problemas matemáticos complexos.

Funções Trigonométricas

As funções trigonométricas, como o seno (sen), o cosseno (cos) e a tangente (tg), desempenham um papel importante na matemática, ligadas aos ângulos e ao círculo trigonométrico. Elas ajudam a compreender o comportamento periódico dessas funções e suas relações nos diferentes quadrantes. Além de serem importantes na geometria dos triângulos, essas funções são essenciais para a análise de oscilações e fenômenos periódicos em diversas áreas da matemática e das ciências.

Função Seno

A função seno (sen) é uma função trigonométrica que relaciona um ângulo à um comprimento de um lado em um triângulo retângulo. Especificamente, o seno de um ângulo agudo é definido como a razão entre o comprimento do cateto oposto a esse ângulo e o comprimento da hipotenusa. Essa definição é expandida ao círculo unitário, onde o seno de um ângulo é dado pela coordenada y do ponto correspondente no círculo, que possui raio igual a 1.

A função seno é periódica, com um período de 2π , o que implica que $\text{sen}(\theta + 2\pi) = \text{sen}(\theta)$ para qualquer valor de θ . Além disso, essa função varia entre -1 e 1 , sendo amplamente utilizada na modelagem de fenômenos cíclicos, como ondas sonoras e eletromagnéticas, bem como em aplicações em física, engenharia e outras disciplinas.

Sua forma gráfica é uma onda suave que cruza o eixo horizontal em múltiplos de π .

A Figura 3.8 apresenta o gráfico da função $f(x) = \text{sen}(x)$ no plano cartesiano (código).

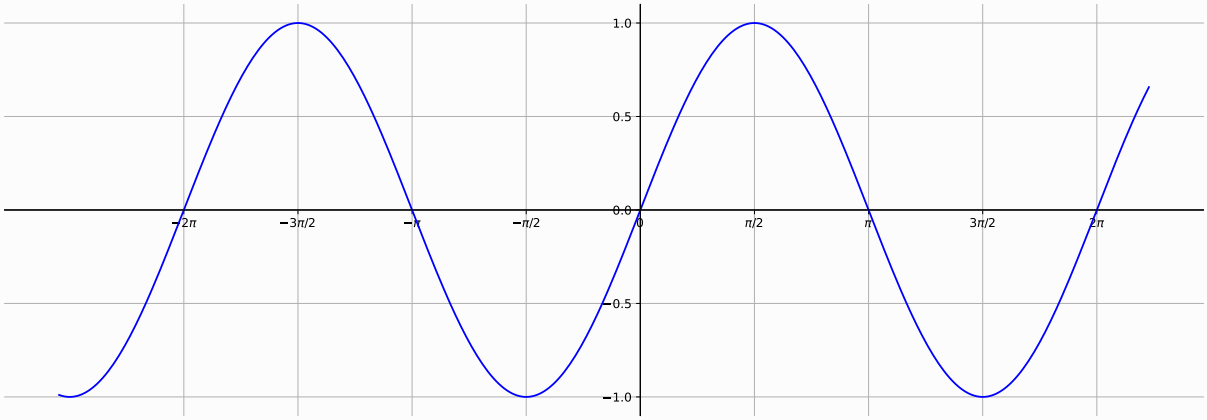


Figura 3.8: Gráfico da função $f(x) = \text{sen}(x)$ no plano cartesiano.

No gráfico da função $f(x) = \text{sen}(x)$, observa-se um padrão periódico de oscilação conforme o valor de x varia. Vamos determinar alguns pontos pertencentes a esse gráfico atribuindo valores específicos para x e realizando os cálculos correspondentes.

EXEMPLO 3.1.4: Calcule o valor da função $f(x) = \text{sen}(x)$ para alguns valores determinados de x.

Consideremos a função $f(x) = \text{sen}(x)$. Podemos calcular os valores da função para alguns ângulos. Vamos fazer isso para os seguintes ângulos no intervalo de 0 a $\frac{\pi}{2}$:

$$\begin{aligned} f(0) &= \text{sen}(0) &&= 0 \\ f\left(\frac{\pi}{6}\right) &= \text{sen}\left(\frac{\pi}{6}\right) &&= \frac{1}{2} \\ f\left(\frac{\pi}{4}\right) &= \text{sen}\left(\frac{\pi}{4}\right) &&= \frac{\sqrt{2}}{2} \\ f\left(\frac{\pi}{3}\right) &= \text{sen}\left(\frac{\pi}{3}\right) &&= \frac{\sqrt{3}}{2} \\ f\left(\frac{\pi}{2}\right) &= \text{sen}\left(\frac{\pi}{2}\right) &&= 1 \end{aligned}$$

Portanto, os pontos $(0, 0)$, $\left(\frac{\pi}{6}, \frac{1}{2}\right)$, $\left(\frac{\pi}{4}, \frac{\sqrt{2}}{2}\right)$, $\left(\frac{\pi}{3}, \frac{\sqrt{3}}{2}\right)$, $\left(\frac{\pi}{2}, 1\right)$ pertencem ao gráfico da função $f(x) = \text{sen}(x)$.

Agora veremos um código em Python para calcular os valores da função $f(x) = \text{sen}(x)$

para alguns valores determinados de x. Para isso vamos utilizar uma biblioteca específica para cálculos matemáticos mais avançados, a biblioteca `math`.

Em Python, nem todas as funcionalidades estão disponíveis por padrão. Algumas, como funções matemáticas avançadas, precisam ser importadas de bibliotecas externas. Bibliotecas em Python são conjuntos de ferramentas que oferecem funcionalidades específicas, permitindo aos programadores escreverem códigos de maneira mais eficiente. Para utilizá-las, é necessário importá-las com a palavra-chave `import` seguida do nome da biblioteca.

Nesta seção, vamos importar a biblioteca `math`, que fornece acesso a várias funções matemáticas, como trigonometria, exponenciais e logaritmos, facilitando cálculos mais complexos.

Função	**Descrição**
`math.sqrt(x)`	Retorna a raiz quadrada de x
`math.pow(x, y)`	Retorna x elevado a y
`math.sin(x)`	Retorna o seno de x (em radianos)
`math.cos(x)`	Retorna o cosseno de x (em radianos)
`math.tan(x)`	Retorna a tangente de x (em radianos)
`math.log(x)`	Retorna o logaritmo natural de x

Tabela 3.1: Comandos básicos da biblioteca `math`

Vamos ver o exemplo a seguir.

EXEMPLO 3.1.4: Escreva um programa em Python para calcular os valores de $f(x) = \text{sen}(x)$ para para alguns valores específicos de x (código).

Vamos escrever um código para calcular os valores da função $f(x)$ para $x = -1$, 0, 0.3, 1, 1.78, 2, 3. O código será:

```
1 # Importando a biblioteca math
2 import math
3
4 # Atribuindo valores para x
5 angulos = [-1, 0, 0.3, 1, 1.78, 2, 3]
6
7 # Efetuando os cálculos correspondentes
8 for angulo in angulos:
9     seno_angulo = math.sin(angulo)
```

```
10     print(f"sen({angulo}) = {seno_angulo}")
```

Os resultados serão:

```
1 sen(-1)   = -0.8414
2 sen(0)    = 0.0
3 sen(0.3)  = 0.2955
4 sen(1)    = 0.8414
5 sen(1.78) = 0.9781
6 sen(2)    = 0.9092
7 sen(3)    = 0.1411
```

Dessa forma, podemos observar como a função $f(x) = \text{sen}(x)$ é avaliada para diferentes valores de x, utilizando Python.

Função Cosseno

A função cosseno (cos) é uma função trigonométrica que relaciona um ângulo a uma razão geométrica. No contexto do círculo trigonométrico, o cosseno de um ângulo é definido como a coordenada x do ponto correspondente no círculo unitário. Assim, para um ângulo θ, temos $\cos(\theta) = x$, onde x é a projeção do ponto no círculo sobre o eixo x.

A função cosseno é periódica, com período 2π, o que significa que $\cos(\theta+2\pi) = \cos(\theta)$ para qualquer valor de θ. A imagem desta função está no intervalo entre -1 e 1, apresentando um comportamento similar ao da função seno, e é amplamente utilizada em diversas áreas, como física e engenharia, para modelar fenômenos que envolvem ciclos e oscilações.

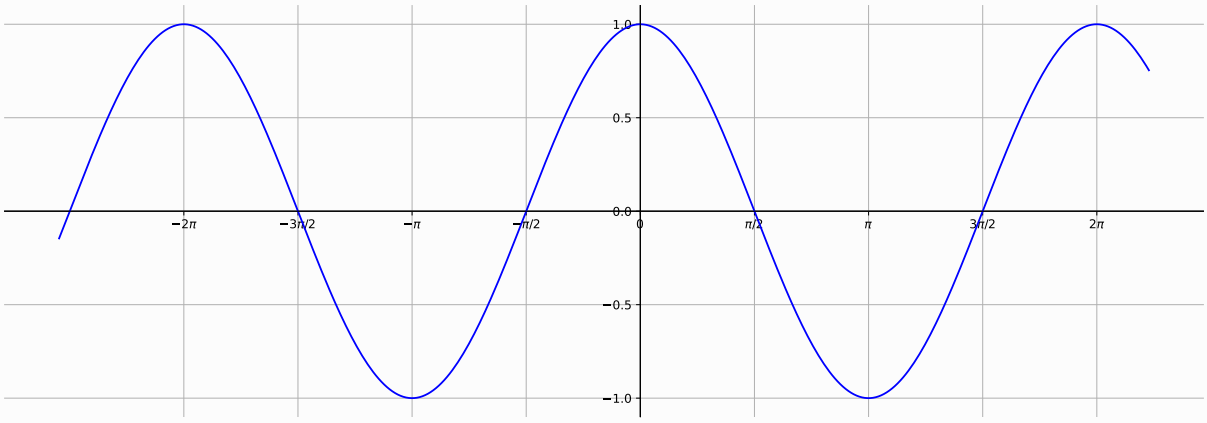


Figura 3.9: Gráfico da função $f(x) = \cos(x)$ no plano cartesiano.

A Figura 3.9 apresenta o gráfico da função $f(x) = \cos(x)$ no plano cartesiano (código).

No gráfico da função $f(x) = \cos(x)$, observa-se um padrão periódico de oscilação conforme o valor de x varia. Vamos determinar alguns pontos pertencentes a esse gráfico atribuindo valores específicos para x e realizando os cálculos correspondentes.

EXEMPLO 3.1.5: Calcule o valor da função $f(x) = \cos(x)$ para alguns valores determinados de x.

Consideremos a função $f(x) = \cos(x)$. Podemos calcular os valores da função para alguns ângulos. Vamos fazer isso para os seguintes ângulos no intervalo de 0 a 2π:

$$
\begin{aligned}
f(0) &= \cos(0) &&= 1 \\
f\left(\frac{\pi}{6}\right) &= \cos\left(\frac{\pi}{6}\right) &&= \frac{\sqrt{3}}{2} \\
f\left(\frac{\pi}{4}\right) &= \cos\left(\frac{\pi}{4}\right) &&= \frac{\sqrt{2}}{2} \\
f\left(\frac{\pi}{3}\right) &= \cos\left(\frac{\pi}{3}\right) &&= \frac{1}{2} \\
f\left(\frac{\pi}{2}\right) &= \cos\left(\frac{\pi}{2}\right) &&= 0
\end{aligned}
$$

Portanto, os pontos $(0, 1)$, $\left(\frac{\pi}{6}, \frac{\sqrt{3}}{2}\right)$, $\left(\frac{\pi}{4}, \frac{\sqrt{2}}{2}\right)$, $\left(\frac{\pi}{3}, \frac{1}{2}\right)$, $\left(\frac{\pi}{2}, 0\right)$ pertencem ao gráfico da função $f(x) = \cos(x)$.

Agora, vamos ver um código em Python para calcular os valores da função $f(x) = \cos(x)$ utilizando a biblioteca `math`.

EXEMPLO 3.1.5: Escreva um programa em Python para calcular os valores de $f(x) = \cos(x)$ para para alguns valores específicos de x (código).

Vamos escrever um código para calcular os valores da função $f(x) = \cos(x)$ para $x = -1$, 0, 0.3, 1, 1.78, 2, 3. O código será:

```
1 # Importando a biblioteca math
2 import math
```

```
3
4 # Atribuindo valores para x
5 angulos = [-1, 0, 0.3, 1, 1.78, 2, 3]
6
7 # Efetuando os cálculos correspondentes
8 for angulo in angulos:
9     cosseno_angulo = math.cos(angulo)
10    print(f"cos({angulo}) = {cosseno_angulo}")
```

Os resultados serão:

```
1 cos(-1)   = 0.5403
2 cos(0)    = 1.0
3 cos(0.3)  = 0.9553
4 cos(1)    = 0.5403
5 cos(1.78) = -0.2076
6 cos(2)    = -0.4161
7 cos(3)    = -0.9899
```

Dessa forma, podemos observar como a função $f(x) = \cos(x)$ é avaliada para diferentes valores de x, utilizando Python.

Função Tangente

A função tangente (tg) é uma das funções trigonométricas fundamentais. Ela é definida como a razão entre o seno e o cosseno de um ângulo. Matematicamente, podemos expressar a tangente da forma

$$\text{tg}(x) = \frac{\text{sen}(x)}{\cos(x)},$$

onde $x \neq \frac{\pi}{2} + k\pi$, com $k \in \mathbb{Z}$. Isso ocorre porque a tangente não está definida para esses valores de x, onde o cosseno é igual a zero, resultando em uma divisão por zero. A função tangente é periódica, com período π, e varia de $-\infty$ a $+\infty$.

Essa definição é válida tanto no contexto de um triângulo retângulo quanto no círculo trigonométrico. No triângulo retângulo, a tangente representa a razão entre o comprimento do cateto oposto e o comprimento do cateto adjacente ao ângulo. No círculo trigonométrico, ela é a razão entre as coordenadas y (seno) e x (cosseno) do ponto correspondente ao ângulo sobre a circunferência.

No plano cartesiano, o gráfico da função tangente apresenta características distintas, como suas assíntotas verticais e a periodicidade da função. A Figura 3.10 apresenta o gráfico da função $f(x) = \text{tg}(x)$ no intervalo de 0 a 2π (código) .

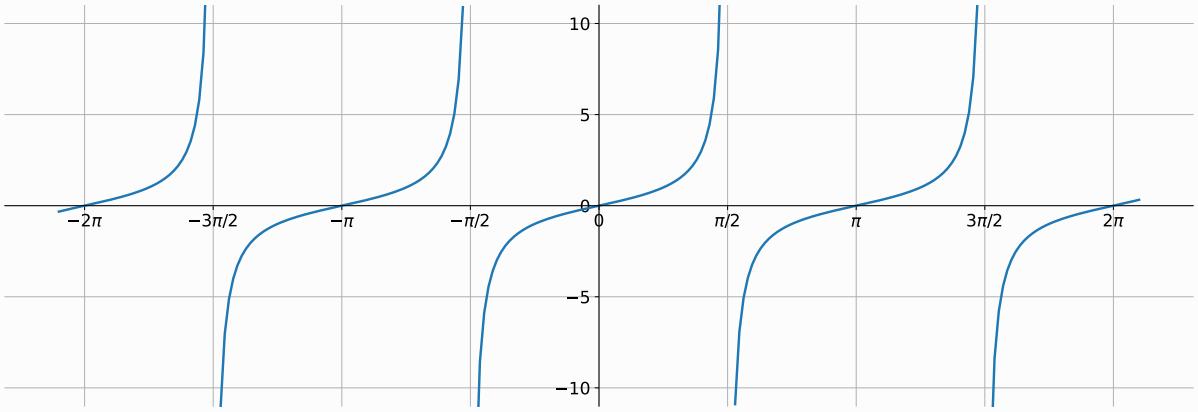


Figura 3.10: Gráfico da função $f(x) = \text{tg}(x)$ no plano cartesiano.

No gráfico da função $f(x) = \text{tg}(x)$, observe as assíntotas verticais nos pontos onde $x = \dfrac{\pi}{2} + k\pi$, indicando os pontos onde a tangente é indefinida. Vamos determinar alguns pontos pertencentes a esse gráfico atribuindo valores específicos para x e realizando os cálculos correspondentes.

EXEMPLO 3.1.6: Calcule o valor da função $f(x) = \text{tg}(x)$ para alguns valores determinados de x.

Consideremos a função tangente $f(x) = \text{tg}(x)$. Podemos calcular os valores da função para alguns ângulos. Vamos fazer isso para os seguintes ângulos no intervalo de 0 a $\dfrac{\pi}{2}$:

$$\begin{aligned}
f(0) = \text{tg}(x) &= 0 \\
f\left(\frac{\pi}{6}\right) = \text{tg}\left(\frac{\pi}{6}\right) &= \frac{\sqrt{3}}{3} \\
f\left(\frac{\pi}{4}\right) = \text{tg}\left(\frac{\pi}{4}\right) &= 1 \\
f\left(\frac{\pi}{3}\right) = \text{tg}\left(\frac{\pi}{3}\right) &= \sqrt{3} \\
f\left(\frac{\pi}{2}\right) = \text{tg}\left(\frac{\pi}{2}\right) &= \text{indefinido}
\end{aligned}$$

Portanto, os pontos $(0,0)$, $\left(\frac{\pi}{6}, \frac{\sqrt{3}}{3}\right)$, $\left(\frac{\pi}{4}, 1\right)$, $\left(\frac{\pi}{3}, \sqrt{3}\right)$ pertencem ao gráfico da função $\text{tg}(x)$.

Agora, vamos ver um código em Python para calcular os valores da função $f(x) = \text{tg}(x)$ utilizando a biblioteca `math`.

EXEMPLO 3.1.6: Escreva um programa em Python para calcular os valores de $f(x) = \text{tg}(x)$ para alguns valores específicos de x (código).

Vamos escrever um código para calcular os valores da função $f(x)$ para $x = -1$, 0, 0.3, 1, 1.78, 2, 3. O código será:

```
# Importando a biblioteca math
import math

# Atribuindo valores para x
angulos = [-1, 0, 0.3, 1, 1.78, 2, 3]

# Efetuando os cálculos correspondentes
for angulo in angulos:
    tangente_angulo = math.tan(angulo)
    print(f"tg({angulo}) = {tangente_angulo}")
```

Os resultados serão:

```
tg(-1)   = -1.557
tg(0)    = 0.0
tg(0.3)  = 0.3093
tg(1)    = 1.5574
tg(1.78) = -4.7100
tg(2)    = -2.1850
tg(3)    = -0.1425
```

Dessa forma, podemos observar como a função $f(x) = \text{tg}(x)$ é avaliada para diferentes valores de x utilizando Python.

Para concluir, as funções trigonométricas, como o seno, cosseno e tangente, são fundamentais na matemática e nas ciências devido à sua capacidade de modelar oscilações e padrões periódicos que aparecem em diversos fenômenos naturais e

artificiais. Essas funções não apenas ajudam a compreender a geometria dos triângulos e a circularidade dos ângulos, mas também são essenciais para a análise de ondas, sinais e movimentos cíclicos. Utilizando ferramentas computacionais como Python e suas bibliotecas, como `math`, podemos calcular, visualizar e explorar essas funções de maneira eficiente, permitindo em muitos casos uma análise detalhada e precisa de comportamentos angulares e periódicos em várias áreas do conhecimento, desde a física até a engenharia e além.

3.2 Equações e Zeros de Funções

Uma equação é uma sentença matemática aberta, uma afirmação matemática que expressa a igualdade entre duas expressões. De forma geral, uma equação envolve uma ou mais variáveis, e encontrar as soluções da equação significa determinar os valores dessas variáveis que tornam a igualdade verdadeira. Quando há apenas uma variável envolvida, a equação pode ser escrita na forma $f(x) = 0$, onde $f(x)$ é uma função matemática que depende da variável x. Os valores de x que satisfazem essa equação são chamados de zeros da função $f(x)$, pois são os pontos onde o gráfico de $f(x)$ intercepta o eixo x. Em outras palavras, resolver uma equação é equivalente a encontrar os zeros da função associada a essa equação. A seguir, exploraremos alguns métodos para encontrar esses zeros e resolver equações de maneira eficiente.

Existem diversos métodos para encontrar zeros de funções, cada um com suas características e aplicabilidades. Vamos abordar dois desses métodos.

Método Analítico: Neste método, escrevemos $f(x) = 0$ e resolvemos a equação resultante para encontrar a solução ou as soluções. Existem equações em que pode ser bastante complicado ou até mesmo pode não ser possível encontrar uma solução de forma analítica.

Método Numérico: Os métodos numéricos abrangem diversas técnicas para aproximar soluções de problemas matemáticos complexos ou difíceis de serem solucionados analiticamente. Dentro desta categoria, os métodos iterativos utilizam algoritmos que repetem cálculos até encontrar uma solução satisfatória, começando com uma estimativa inicial e refinando-a a cada iteração. Esses métodos são especialmente úteis quando soluções analíticas são inviáveis ou complicadas, e dependem de computadores para realizar cálculos de forma eficiente e precisa. Portanto, enquanto os métodos numéricos são amplos, os métodos iterativos se focam em refinar estimativas até alcançar a solução desejada.

Nesse capítulo, vamos explorar os métodos analíticos para encontrar zeros de algumas funções. No Capítulo 4, vamos abordar métodos numéricos para encontrar os zeros de uma função específica, fornecendo mais detalhes e explicações.

Para ilustrar os métodos analíticos, consideremos exemplos com diferentes tipos de funções.

EXEMPLO 3.2.1: Encontre o zero da função afim $f(x) = 2x - 4$ de forma analítica.

Escrevendo a equação na forma $f(x) = 0$, para encontrar essa solução podemos executar os passos a seguir.

$$\begin{aligned} f(x) &= 0 \\ 2x - 4 &= 0 \\ 2x &= 4 \\ x &= \frac{4}{2} \\ x &= 2 \end{aligned}$$

Assim encontrando a solução da equação que é $x = 2$, que se trata de um zero da função $f(x) = 2x - 4$, pois $f(2) = 2 \cdot 2 - 4 = 0$. Uma Função Afim admite apenas um valor como sendo um zero da função.

A Figura 3.11 apresenta o gráfico da função $f(x) = 2x - 4$ destacando seu zero (código).

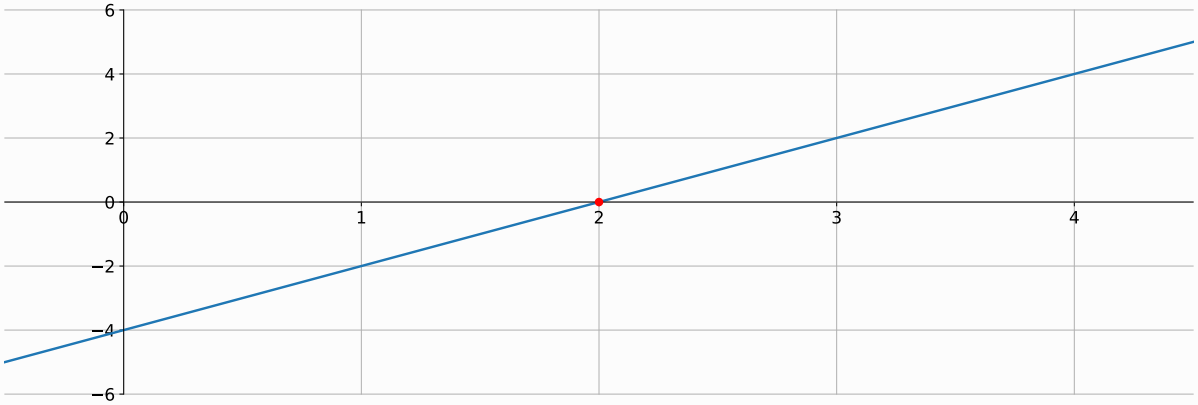


Figura 3.11: Gráfico de $f(x) = 2x - 4$ com seu zero destacado.

Vamos agora analisar as funções quadráticas. No caso do método analítico, para

encontrar as soluções da equação $ax^2 + bx + c = 0$, onde a, b e c são constantes reais e $a \neq 0$. Podemos usar a fórmula resolutiva de Bhaskara dada por

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}, \tag{3.1}$$

onde a, b e c são constantes. Vamos ver um exemplo de como encontrar as soluções de uma equação quadrática de forma analítica utilizando essa fórmula.

EXEMPLO 3.2.2: Encontre os zeros da função $f(x) = x^2 - 6x + 5$ de forma analítica.

Escrevendo a equação na forma $x^2 - 6x + 5 = 0$, podemos resolvê-la aplicando a fórmula de Bhaskara, conforme apresentado na equação (3.1).

$$x = \frac{-(-6) \pm \sqrt{(-6)^2 - 4 \cdot 1 \cdot 5}}{2 \cdot 1}$$

$$x = \frac{6 \pm \sqrt{36 - 20}}{2}$$

$$x = \frac{6 \pm \sqrt{16}}{2}$$

$$x = \frac{6 \pm 4}{2}.$$

Portanto, as soluções são:

$$x_1 = \frac{6 + 4}{2} = 5$$

$$x_2 = \frac{6 - 4}{2} = 1.$$

A Figura 3.12 apresenta o gráfico da função $f(x) = x^2 - 6x + 5$ destacando seus zeros (código).

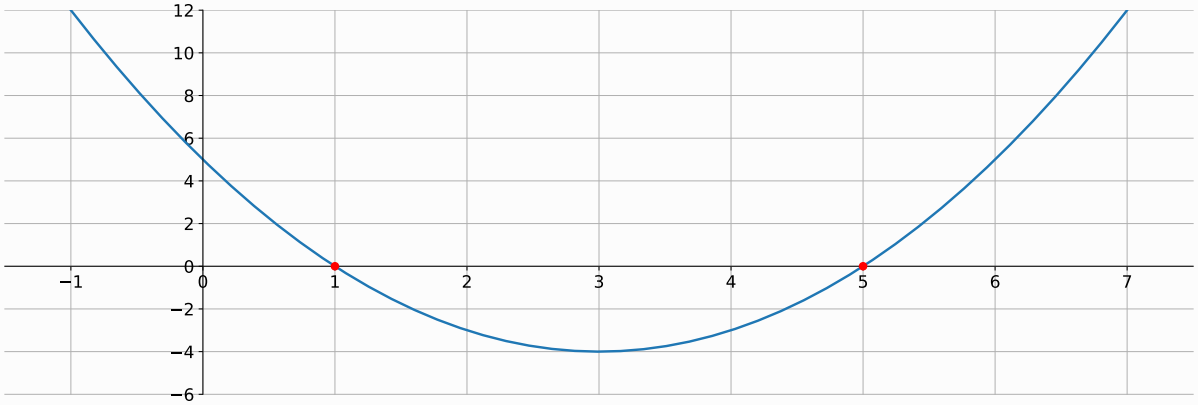


Figura 3.12: Gráfico de $f(x) = x^2 - 5x + 6$ com seus zeros destacados.

Vamos analisar agora a função trigonométrica $f(x) = \text{sen}(x)$. Conhecemos os zeros dessa função com base em suas propriedades (círculo trigonométrico). Os zeros da função seno ocorrem em múltiplos de π, ou seja, $x = -\pi,\ 0,\ \pi,\ 2\pi,\ 3\pi,\ \ldots$.

A Figura 3.13 apresenta o gráfico da função $f(x) = \text{sen}(x)$ destacando seus zeros (código).

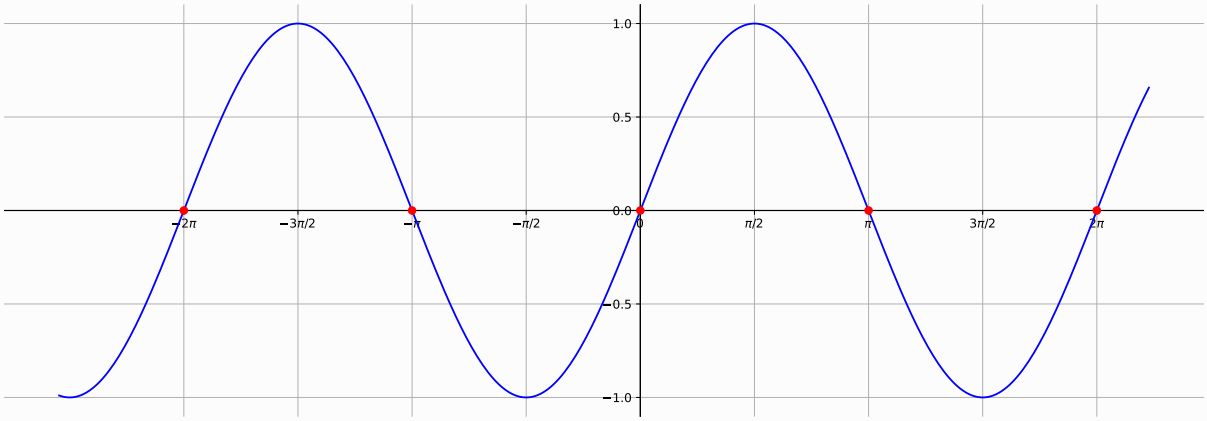


Figura 3.13: Gráfico de $f(x) = \text{sen}(x)$ com seus zeros destacados.

Vamos considerar agora uma função mais complexa, como $f(x) = x - \text{sen}(x)$. Diferente da função seno simples, onde os zeros são múltiplos de π, os zeros de $f(x) = x - \text{sen}(x)$ são difíceis de serem encontrados analiticamente. Exemplos como esse ilustram a importância dos métodos numéricos, que serão explorados no Capítulo 4, métodos esses que permitem aproximar soluções de forma prática para funções complexas onde métodos analíticos não são viáveis.

Ao compreendermos os conceitos fundamentais das funções na matemática, como domínio, contradomínio, imagem, zeros, entre outros, ampliamos nossa capacidade de

resolver problemas de forma eficiente e precisa. Nesta seção, exploramos as funções na matemática, agora, daremos um passo adiante onde vamos explorar as funções em Python. Na próxima seção, faremos uma apresentação breve sobre esse tipo de função utilizando a linguagem Python para criar e manipular funções, abrindo um novo horizonte de possibilidades para resolver problemas matemáticos e computacionais.

3.3 Funções em Python

As funções em Python são blocos de código que realizam tarefas específicas e podem ser reutilizadas em diferentes partes de um programa. Para definir uma função em Python, utilizamos a palavra-chave `def` seguida pelo nome da função e parênteses que podem ou não conter os argumentos da função. Esses argumentos são os valores que a função recebe para executar suas ações. Para utilizar uma função, basta chamá-la pelo nome, seguido pelos parênteses que podem conter os argumentos necessários para a execução da função, caso ela os exija. Essa abordagem ajuda a organizar o código, tornando-o mais legível e reutilizável. Além disso, as funções podem retornar valores, que são os resultados das operações realizadas pela função. Isso permite que o programa receba e manipule esses valores de retorno conforme necessário.

Uma função em Python segue a sintaxe:

```
1 def nome_da_funcao(argumentos):
2     # Corpo da função
3     return resultado
```

Cada elemento do código desempenha um papel específico, vamos explicar o significado de cada um deles a seguir.

- `def`: palavra-chave para definir uma função.
- `nome_da_funcao`: nome que você escolhe para sua função.
- `argumentos`: valores que a função recebe para executar suas ações.
- `return`: palavra-chave para retornar um resultado da função.

Vamos explorar vários exemplos para melhor compreendermos na prática o funcionamento das funções em Python.

Inicialmente vamos ver um exemplo onde escreveremos um código Python que define uma função de soma simples.

EXEMPLO 3.3.1: Escreva um código Python que define uma função de soma simples (código).

A função de soma simples pode ser definida da seguinte forma.

```
1 # Definindo a função
2 def soma(a, b):
3   return a + b
```

No código, a palavra-chave `def` é usada para definir a função `soma(a, b)` que receberá dois argumentos, `a` e `b`, e retornará a soma desses dois valores. A partir de agora podemos reutilizá-la em qualquer outro código, apenas chamando-a pelo nome e fornecendo os valores a serem atribuídos para os argumentos, por exemplo

```
1 # Chamando a função
2 resultado = soma(3, 5)
3 print("A soma de 3 e 5 é:", resultado)
```

Esse código gera a resposta apresentada a seguir.

```
A soma de 3 e 5 é: 8
```

Dentro da função, a instrução `return a + b` especifica que o resultado da soma de `a` e `b` deve ser retornado quando a função é chamada.

Vamos ver um exemplo onde a função é definida e retorna os zeros de uma função quadrática.

EXEMPLO 3.3.2: Escreva um código Python que define uma função para calcular os zeros da função quadrática $f(x) = ax^2 + bx + c$ (código).

Para definirmos uma função em Python que recebe os coeficientes a, b e c de uma função quadrática e retorna os zeros da função, devemos primeiro verificar o discriminante, que nos dirá se a equação possui duas raízes reais e distintas, duas raízes reais e iguais ou não possui raízes reais. Em seguida, calcularemos as raízes, se existirem. Podemos utilizar o código a seguir.

```
1 # Importando a biblioteca math
2 import math
3
4 # Definindo a função
5 def bhaskara(a, b, c):
```

```
    delta = b**2 - 4*a*c
    if delta > 0:
        x1 = (-b + math.sqrt(delta)) / (2*a)
        x2 = (-b - math.sqrt(delta)) / (2*a)
        return x1, x2
    elif delta == 0:
        x = -b / (2*a)
        return x
    else:
        return "Não há raízes reais"
```

O código define uma função chamada `bhaskara` que calcula as raízes de uma equação quadrática $ax^2+bx+c=0$ utilizando a fórmula de Bhaskara (3.1). A função começa importando o módulo `math` para usar a função `sqrt`, que calcula a raiz quadrada. Em seguida, define-se a função `bhaskara` que aceita três parâmetros: `a`, `b` e `c`, que são os coeficientes da equação quadrática. Dentro da função, o discriminante (Δ) é calculado como `b**2 - 4*a*c`. Se o discriminante for positivo (`delta > 0`), a função calcula e retorna as duas raízes reais `x1` e `x2`. Se o discriminante for zero (`delta == 0`), a função calcula e retorna a raiz real única `x`. Se o discriminante for negativo, a função retorna a mensagem `"Não há raízes reais"`.

A partir de agora podemos reutilizá-la em qualquer outro código, apenas chamando-a pelo nome e fornecendo os valores a serem atribuídos para os argumentos. Vamos ver o exemplo a seguir.

EXEMPLO 3.3.3: Escreva um programa em Python que utilize a função `bhaskara` para encontrar os zeros das funções $f(x)=x^2-7x+6$, $g(x)=x^2-4x+4$ e $h(x)=x^2-x+10$ (código).

Para encontrar os zeros das funções quadráticas $f(x)$, $g(x)$ e $h(x)$ utilizando a função `bhaskara` em Python, é necessário primeiro atribuir os valores dos coeficientes a, b e c de cada equação quadrática. Em seguida, chamamos a função `bhaskara` com esses valores para calcular as raízes da equação.

```
# Chamando a função e atribuindo os argumentos

raízes_f = bhaskara(1, -7, 6)
print(f"Os zeros da função f(x) são: {raízes_f}")

raízes_g = bhaskara(1, -4, 4)
```

```
print(f"Os zeros da função g(x) são: {raízes_g}")

raízes_h = bhaskara(1, -1, 10)
print(f"Os zeros da função h(x) são: {raízes_h}")
```

Esse código gera o resultado apresentado a seguir.

```
Os zeros da função f(x) são: (6, 1)
Os zeros da função g(x) são: 2
Os zeros da função h(x) são: Não há raízes reais
```

Este código Python demonstra como utilizar a função `bhaskara` para calcular os zeros das funções quadráticas $f(x)$, $g(x)$ e $h(x)$. Cada chamada da função `bhaskara` é feita com os coeficientes específicos de cada função quadrática, e os resultados são impressos para indicar os zeros de cada função, mostrando também a mensagem adequada quando não há raízes reais para a função como no caso da função $h(x)$.

Ao compreender como definir funções em Python, passar argumentos para elas e receber valores de retorno, os programadores têm uma ferramenta poderosa para organizar e reutilizar código de maneira eficiente. É importante notar a distinção entre funções na matemática e em Python: enquanto as primeiras representam relações entre conjuntos, as segundas são blocos reutilizáveis que encapsulam tarefas específicas.

No próximo capítulo, exploraremos métodos numéricos para encontrar zeros de funções. Estes métodos são fundamentais para resolver equações complexas que não podem ser resolvidas de forma analítica, oferecendo técnicas eficazes e algoritmos poderosos que são amplamente utilizados em diversas áreas da ciência e engenharia.

4

Métodos Numéricos para Zeros de Funções

Os métodos numéricos são técnicas computacionais utilizadas para resolver problemas matemáticos que podem não ter solução analítica direta. Eles são particularmente úteis quando lidamos com equações complexas ou funções onde seus zeros não podem ser encontrados facilmente. Neste trabalho, serão utilizados métodos numéricos iterativos na busca por zeros de funções.

Os métodos numéricos iterativos são uma categoria importante desses métodos, que utilizam estimativas iniciais para as soluções, que são valores aproximados iniciais fornecidos como ponto de partida para o processo iterativo. Estas estimativas iniciais são aprimoradas repetidamente até alcançar uma precisão desejada. Esse aprimoramento é realizado por meio de iterações, que consistem no processo de repetir cálculos para melhorar a precisão da solução. Durante as iterações, os métodos numéricos empregam técnicas específicas para ajustar as estimativas, movendo-as em direção à solução correta. Esse movimento em direção à solução é chamado de convergência, e acontece gradualmente até que a solução atinja um nível aceitável de precisão.

Esses métodos são fundamentais em diversas áreas, como engenharia, física, estatística e ciência da computação, proporcionando uma abordagem prática e eficiente para resolver problemas matemáticos complexos.

4.1 Método para Calcular a Raiz Quadrada

Vamos começar explorando um método para calcular a raiz quadrada de um determinado número. No contexto dos métodos numéricos iterativos, utilizamos um índice n para indicar a posição de cada estimativa na sequência. Por exemplo, x_n representa nossa estimativa atual e x_{n+1} é a próxima estimativa. Essa nova estimativa é então usada na próxima iteração do método. A letra n como índice ajuda a indicar a ordem das estimativas na sequência, facilitando a compreensão de como cada estimativa se baseia na anterior.

Suponha que desejamos encontrar a raiz quadrada de um número a. Inicialmente, fazemos uma estimativa inicial x_0. Em cada iteração, atualizamos nossa estimativa usando a fórmula

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right). \tag{4.1}$$

Essa fórmula é derivada de um método matemático chamado Método de Newton, do qual falaremos com maiores detalhes na Seção 4.2. Ela deve ser aplicada repetidamente até que a diferença entre duas interações consecutivas seja suficientemente pequena para considerarmos a solução como precisa o bastante.

Vamos calcular a raiz quadrada de 9 usando a fórmula (4.1). É claro que sabemos que a resposta é 3, mas vamos usar esse exemplo para entender o funcionamento do método.

Exemplo 4.1.1: Calcular o valor aproximado da raiz quadrada de 9 com 4 casas decimais de precisão.

Vamos calcular a raiz quadrada do número 9 usando como estimativa inicial $x_0 = 4$ e vamos repetir os cálculos com as novas estimativas até obtermos 4 casas decimais precisas.

$$x_0 = 4$$

$$
\begin{aligned}
x_1 &= \frac{1}{2}\left(4 + \frac{9}{4}\right) && = 3{,}125 \\
x_2 &= \frac{1}{2}\left(3{,}125 + \frac{9}{3{,}125}\right) && = 3{,}0025 \\
x_3 &= \frac{1}{2}\left(3{,}0025 + \frac{9}{3{,}0025}\right) && = 3{,}0000
\end{aligned}
$$

Após três iterações, obtivemos a resposta com 4 casas decimais de precisão.

Observe que em cada cálculo realizado, empregamos o valor da estimativa anterior, evidenciando assim a natureza iterativa desse método. Agora, vamos aplicar essa ideia para calcular a raiz quadrada de 7 usando a fórmula (4.1). Como a raiz quadrada de 7 é irracional, podemos ver como a fórmula iterativa é eficiente para aproximar a solução desejada.

EXEMPLO 4.1.2: Calcular o valor aproximado da raiz quadrada de 7 com 4 casas decimais de precisão.

Vamos calcular a raiz quadrada do número 7 usando como estimativa inicial $x_0 = 3$ e vamos repetir os cálculos com as novas estimativas até obtermos 4 casas decimais precisas.

$$
\begin{aligned}
x_0 &= 3 \\
x_1 &= \frac{1}{2}\left(3 + \frac{7}{3}\right) && = 2{,}6666 \\
x_2 &= \frac{1}{2}\left(2{,}6666 + \frac{7}{2{,}6666}\right) && \approx 2{,}6458 \\
x_3 &= \frac{1}{2}\left(2{,}6458 + \frac{7}{2{,}6458}\right) && \approx 2{,}6457 \\
x_4 &= \frac{1}{2}\left(2{,}6457 + \frac{7}{2{,}6457}\right) && \approx 2{,}6457
\end{aligned}
$$

Observe que da terceira para a quarta iteração os algarismos das primeiras 4 casas decimais não foram alterados, logo, após quatro iterações temos uma estimativa da raiz quadrada de 7, sendo $2{,}6457$, com precisão de 4 casas decimais. Podemos verificar que $2{,}6457^2 = 6{,}9997$, ou seja, bem próximo de 7.

Vamos agora utilizar a programação para realizar esses cálculos de forma computacional. Uma das vantagens de usar o computador é a capacidade de realizar muitas

iterações rapidamente, alcançando a precisão desejada de maneira eficiente.

Primeiramente vamos definir a função `raiz_quadrada` que se utilizará da fórmula (4.1). Essa função será responsável para efetuar os cálculos até que se obtenha um resultado satisfatório (código).

```
# Definindo a função raiz quadrada
def raiz_quadrada(a, x0):
    print(f'Estimativa inicial: x0 = {x0}')
    estimativa_x = x0

    # Realiza até 100 iterações para calcular a raiz quadrada
    for iteracao in range(1, 100):
        x0 = 0.5 * (x0 + a / x0)
        print(f'Iteração {iteracao}: x{iteracao} = {x0}')

        # Verifica se a aproximação é sufuciente
        if abs(estimativa_x - x0) < 1e-4:
            break
        estimativa_x = x0

    # Retorna a estimativa final da raiz quadrada
    return x0
```

A função `raiz_quadrada` recebe dois argumentos: `a`, que é o número para o qual se deseja encontrar a raiz quadrada, e `x0`, a estimativa inicial. A função começa imprimindo a estimativa inicial. Em seguida, o laço `for` realiza até 100 iterações para calcular a nova estimativa da raiz quadrada utilizando a fórmula iterativa `x0 = 0.5 * (x0 + a / x0)`. A condição de parada verifica, utilizando `if`, se a diferença absoluta entre `estimativa_x` e `x0` é menor que `1e-4`, o que equivale a 0,0001, ou seja, 4 casas decimais de precisão. Se essa condição for atendida, a sequência é interrompida utilizando `break`. A função retorna a estimativa final da raiz quadrada.

Agora vamos escrever um código que se utiliza da função `raiz_quadrada` para calcular a raiz quadrada de um número fornecido pelo usuário.

> Exemplo 4.1.1: Escreva um código Python que solicita ao usuário um número e uma estimativa inicial, e utiliza a função `raiz_quadrada` para calcular a raiz quadrada desse número (código).

Ao iniciar o programa, é solicitado ao usuário digitar o número para encontrar a raiz quadrada e posteriormente digitar a estimativa inicial. Após, o programa mostra a estimativa inicial e gera todos os resultados encontrados de todas as iterações até que se obtenha a precisão desejada.

```
# Solicita um número do usuário e a estimativa inicial
a = float(input('Digite o número para encontrar a raiz
    quadrada: '))
x0 = float(input('Digite a estimativa inicial: '))

# Chama a função raiz_quadrada e executa o método
resultado = raiz_quadrada(a, x0)

# Exibe o resultado
print(f'\nA raiz quadrada de {a} é aproximadamente
    {resultado}')
```

O código solicita ao usuário que insira um número e uma estimativa inicial usando o comando `input`. Esses valores são armazenados nas variáveis `a` e `x0`, respectivamente. A função `raiz_quadrada` é então chamada com esses argumentos, realizando o cálculo da raiz quadrada. O resultado é armazenado na variável `resultado` e exibido ao usuário com o comando `print`, mostrando a raiz quadrada aproximada do número fornecido.

Supondo que o usuário tenha digitado o número 17 para se calcular a raiz quadrada e tenha digitado o número 8 para a estimativa inicial, o programa gera a reposta a seguir.

```
Digite o número para encontrar a raiz quadrada: 17
Digite a estimativa inicial: 8
Estimativa inicial: x0 = 8,0
Iteração 1: x1 = 5,0625
Iteração 2: x2 = 4,2103
Iteração 3: x3 = 4,1240
Iteração 4: x4 = 4,1231
Iteração 5: x5 = 4,1231

A raiz quadrada de 17,0 é aproximadamente 4,1231.
```

Observe que o programa interrompe as iterações quando os 4 primeiros dígitos após a vírgula não se alteraram, tendo assim uma precisão de 4 casas decimais. Podemos

verificar que $4{,}1231^2 = 16{,}9999$, ou seja, bem próximo de 17.

Podemos também utilizar esse programa para o cálculo da raiz quadrada de números decimais. Supondo que o usuário tenha digitado o número decimal 19,4 para se calcular a raiz quadrada e tenha digitado o número 5,8 para a estimativa inicial, o programa gera a reposta a seguir.

```
Digite o número para encontrar a raiz quadrada: 19.4
Digite a estimativa inicial: 5.8
Estimativa inicial: x0 = 5.8
Iteração 1: x1 = 4.5724
Iteração 2: x2 = 4.4076
Iteração 3: x3 = 4.4045
Iteração 4: x4 = 4.4045

A raiz quadrada de 19.4 é aproximadamente 4.4045.
```

O programa obteve 4 casas decimais de precisão. Podemos verificar que $4{,}4045^2 = 19{,}3996$, ou seja, bem próximo de $19{,}4$.

A programação de cálculos iterativos oferece várias vantagens, incluindo rapidez, precisão e automação, evitando erros manuais e facilitando a análise de problemas complexos. O código apresentado, que calcula a raiz quadrada com precisão de quatro casas decimais, demonstra a eficácia desse método computacional. Executar o código no Google Colab realça sua praticidade, pois elimina a necessidade de instalações adicionais, tornando a programação mais acessível para todos.

Encerramos esta seção com uma pergunta instigante: como calcular a raiz cúbica de um número ou qualquer outra raiz? Essa questão será respondida na próxima seção, onde exploraremos o Método de Newton como uma ferramenta poderosa para lidar não somente com o cálculo de raízes de números reais, mas também para solucionar uma vasta quantidade de problemas matemáticos.

4.2 Método de Newton

Como podemos resolver outras raízes e equações além da raiz quadrada? Essa é uma pergunta natural que surge quando exploramos métodos iterativos como a fórmula (4.1). Podemos resolver diversas raízes e equações utilizando fórmulas prontas. Aqui estão algumas delas.

Raiz Quadrada: Para encontrar a raiz quadrada de um número a, podemos usar a fórmula

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right).$$

Raiz Cúbica: A fórmula para a raiz cúbica de um número a é

$$x_{n+1} = \frac{1}{3}\left(2x_n + \frac{a}{x_n^2}\right).$$

Raiz n-ésima: A fórmula geral para a raiz n-ésima de um número a é

$$x_{n+1} = \frac{1}{n}\left((n-1)x_n + \frac{a}{x_n^{n-1}}\right).$$

Essas fórmulas foram geradas através do Método de Newton e nos permitem encontrar raízes de maneira eficiente. O Método de Newton foi desenvolvido inicialmente pelo grande Sir Isaac Newton, um dos maiores cientistas de todos os tempos. Se trata de um procedimento matemático iterativo usado para encontrar aproximações para um zero de uma função. Começamos com uma estimativa inicial e, em cada iteração, usamos a derivada da função para calcular uma nova estimativa mais próxima do zero desejado. A fórmula geral do Método de Newton para encontrar um zero de uma função $f(x)$ é dada por

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)},$$

onde $f'(x)$ é a derivada de $f(x)$. No caso das raízes n-ésimas, aplicamos essa fórmula específica à função apropriada para obter as fórmulas mencionadas anteriormente.

No entanto, esse conhecimento matemático envolve cálculos mais avançados, normalmente estudados no curso superior. Para nosso propósito, é suficiente utilizar as

fórmulas prontas sem a necessidade de compreender detalhadamente o processo de como foram derivadas.

Podemos expressar o cálculo de raízes como sendo zeros de funções específicas, por exemplo, a raiz quadrada de a pode ser representada como o zero da função $f(x) = x^2 - a$. Quando encontramos o valor de x que torna $f(x) = 0$, obtemos a raiz quadrada de a, ou seja,

$$\begin{aligned} x^2 - a &= 0 \\ x^2 &= a \\ |x| &= \sqrt{a} \\ x &= \pm\sqrt{a} \end{aligned}$$

Da mesma forma, podemos encontrar a raiz cúbica de um número a como o zero da função $g(x) = x^3 - a$. Para encontrar a raiz cúbica, procuramos o valor de x que zera $g(x)$, ou seja,

$$\begin{aligned} x^3 - a &= 0 \\ x^3 &= a \\ x &= \sqrt[3]{a} \end{aligned}$$

Além disso, outras funções podem ser tratadas da mesma maneira. Por exemplo, para encontrar o zero da função trigonométrica $f(x) = \text{sen}(x)$, usamos o Método de Newton para encontrar o valor de x que faz $\text{sen}(x) = 0$. O Método de Newton pode ser aplicado a uma variedade de funções, incluindo polinômios, funções trigonométricas, exponenciais, e muitas outras.

Geometricamente, o Método de Newton pode ser interpretado como o processo de encontrar a interseção da reta tangente à curva da função em um ponto dado inicialmente com o eixo x. A cada iteração, a reta tangente é recalculada com base na inclinação da curva no ponto atual, aproximando-se cada vez mais do zero da função.

A Figura 4.1 mostra o gráfico da função $f(x) = x^2 - 4$ e as retas tangentes geradas pelo Método de Newton. Iniciando pela estimativa inicial $x_0 = 4$, a cada iteração, a interceção da reta tangente com o eixo x se aproxima cada vez mais do zero da função (código).

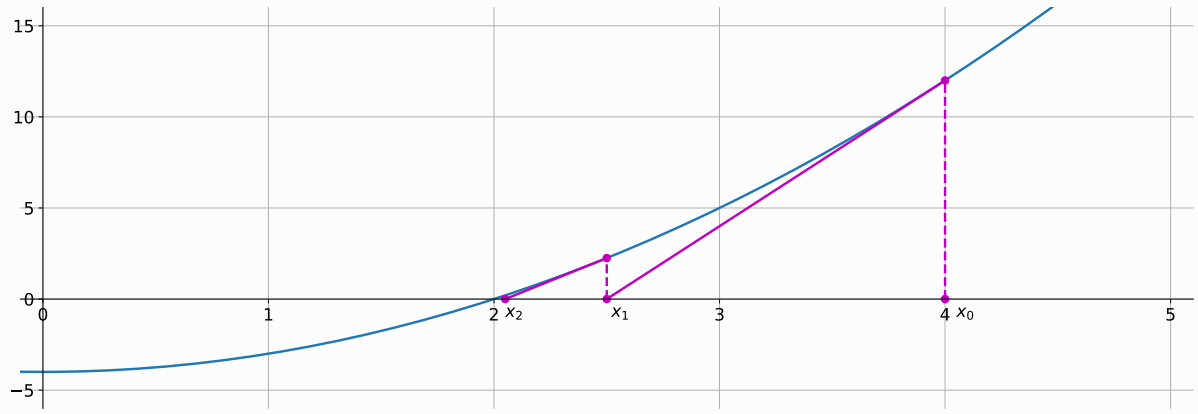


Figura 4.1: Interpretação geométrica do Método de Newton.

Podemos assim utilizar o Método de Newton para funções mais complexas. Vamos fazer um exemplo com uma função trigonométrica.

EXEMPLO 4.2.1: Escreva um código Python que encontre um zero da função $f(x) = \text{sen}(x)$ utilizando o Método de Newton (código).

Sabemos que o $\text{sen}(\pi) = 0$, logo as iterações do Método de Newton vão aproximar o valor da estimativa inicial ao valor de π ou à algum múltiplo de π. A fórmula de iteração específica para esse cálculo será

$$x_{n+1} = x_n - \frac{\text{sen}(x_n)}{\cos(x_n)}.$$

Apresentaremos a seguir o código que encontra um zero da função $f(x) = \text{sen}(x)$ através de uma estimativa inicial suficientemente próxima utilizando o Método de Newton.

```
1  # Importa a biblioteca math
2  import math
3
4  #Define o Método de Newton
5  def metodo_newton(aprox_inicial, precisao):
6      x = aprox_inicial
7      iteracao = 0
8      while True:
9          iteracao += 1
10         x_novo = x - math.sin(x) / math.cos(x)
11         print(f"Iteração {iteracao}: x_{iteracao} = {x_novo}")
12         if abs(x_novo - x) < precisao:
```

```
            return x_novo
        x = x_novo

# Solicita ao usuário as informações iniciais
ap_inicial = float(input('Digite a estimativa inicial: '))
precisao = float(input('Digite a precisão desejada: '))

# Executa o método e imprime os resultados
resultado = metodo_newton(ap_inicial, precisao)
print(f'\nO resultado final aproximado é {resultado}')
```

Ao executar este código, informe a estimativa inicial e a precisão. Se o valor inicial estiver suficientemente próximo de um dos zeros da função seno, o programa gerará uma sequência que se aproxima cada vez mais do zero da função.

No código, o usuário fornece uma estimativa inicial utilizando a função `input`, que é armazenada na variável `ap_inicial`, e a precisão desejada, armazenada na variável `precisao`. A função `metodo_newton` utiliza esses valores para iterar até que a diferença entre a estimativa atual e a nova seja menor que a precisão. A cada iteração, a variável `x` é atualizada com a fórmula `x_novo = x - math.sin(x) / math.cos(x)` e o progresso é impresso. Quando a condição de precisão é satisfeita, a função retorna `x_novo` como a aproximação do zero da função, que é exibida com a função `print`.

Supondo que o usuário tenha digitado o número 2 para a estimativa inicial e 0,0001 para a precisão desejada, o programa gera o seguinte resultado.

```
Digite a estimativa inicial: 2
Digite a precisão desejada: 0.0001
Iteração 1: x_1 = 4.1850
Iteração 2: x_2 = 2.4679
Iteração 3: x_3 = 3.2662
Iteração 4: x_4 = 3.1409
Iteração 5: x_5 = 3.1416
Iteração 6: x_6 = 3.1416

O resultado final aproximado é 3.1416.
```

Observe que com essa estimativa inicial foram necessárias 6 iterações para obter a aproximação de 4 casas decimais, se a estimativa inicial for mais próxima da solução, menos iterações são necessárias para atingir o objetivo.

Agora se o usuário digitar o número 6 para a estimativa inicial e 0,0001 para a precisão desejada, o programa gera o seguinte resultado.

```
1 Digite a estimativa inicial: 6
2 Digite a precisão desejada: 0.0001
3 Iteração 1: x_1 = 6.2910
4 Iteração 2: x_2 = 6.2831
5 Iteração 3: x_3 = 6.2831
6
7 O resultado final aproximado é 6.2831.
```

Observe que, se o usuário digitar o número 6, o programa se aproximaria da resposta 6,2831, que é um outro zero da função seno(aproximadamente 2π). Ou seja, o Método de Newton se aproxima do zero mais próximo da estimativa inicial.

A Figura 4.2 mostra geometricamente como três iterações do Método de Newton geraram tangentes em que o ponto de interceção com o eixo x se aproximam graduadamente do zero da função (código).

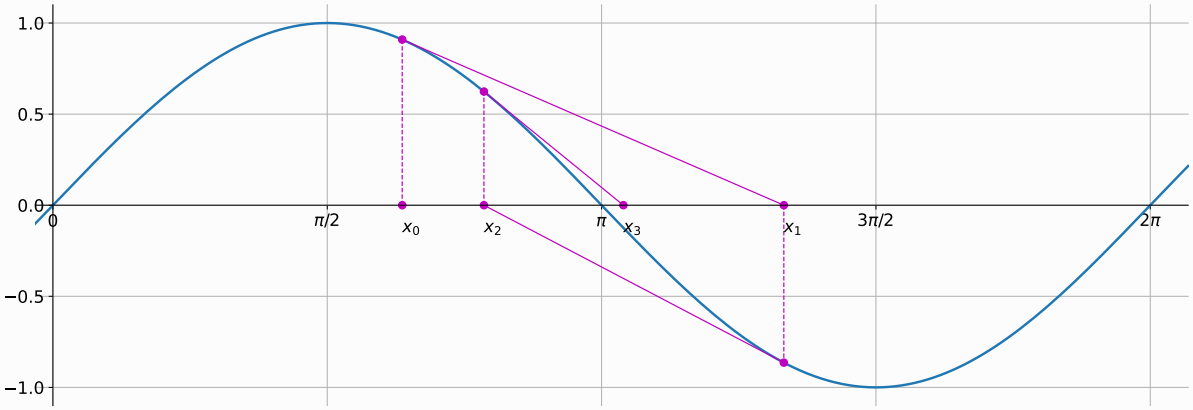


Figura 4.2: Tangentes geradas pelo Método de Newton.

Em alguns casos, dependendo da estimativa inicial, o Método de Newton não converge para o zero da função. Por exemplo, se a estimativa inicial for $\frac{\pi}{2}$, a reta tangente gerada pelo Método de Newton não será direcionada para o zero da função.

A Figura 4.3 mostra geometricamente como a reta tangente não vai em direção ao zero da função (código).

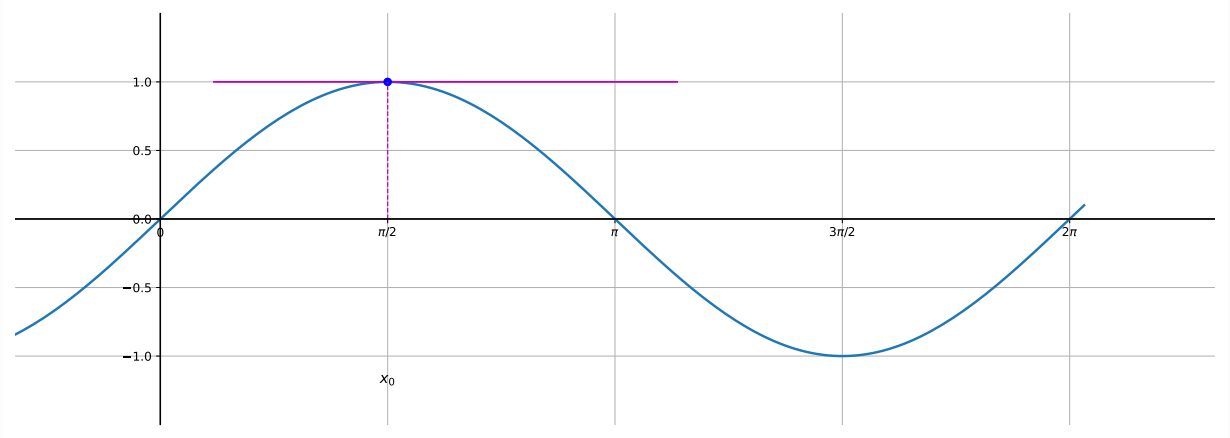


Figura 4.3: Caso em que o Método de Newton não converge.

É importante que a estimativa inicial esteja suficientemente próxima da raiz para que o Método de Newton convirja de forma eficiente. Quando não temos informações suficientes sobre a função para escolher uma estimativa adequada, isso pode ser uma desvantagem significativa do Método de Newton, levando à divergência ou à convergência para uma raiz diferente da desejada.

Aplicar o Método de Newton de forma eficaz exige compreender como calcular a inclinação do gráfico em um ponto, relacionado ao conceito de derivada do cálculo avançado, o que está fora do escopo desta apostila. Nas próximas seções, apresentaremos o Método da Bisseção e o Método da Secante, que, embora exijam mais iterações, não necessitam da derivada. Esses métodos são alternativas para encontrar zeros de funções quando a derivada não é facilmente calculável ou se deseja evitar técnicas matemáticas mais avançadas.

4.3 Método da Bisseção

O Método da Bisseção é uma técnica simples e eficaz para encontrar um zero de uma função em um intervalo conhecido. Ele se baseia no fato de que, se a função é contínua, ou seja, não possui interrupções, buracos ou saltos no gráfico, e muda de sinal entre os extremos do intervalo, deve haver pelo menos um zero dentro desse intervalo. O método utiliza essa propriedade das funções contínuas para localizar sistematicamente o zero da função no intervalo fornecido.

A ideia básica do método é dividir o intervalo pela metade repetidamente até que se obtenha uma aproximação satisfatória do zero da função. A fórmula de iteração do

Método da Bisseção é dada por

$$x_{n+1} = \frac{a_n + b_n}{2}, \tag{4.2}$$

onde x_{n+1} é o próximo ponto intermediário, a_n e b_n são os extremos do intervalo anterior. Essa técnica aproveita a mudança de sinal da função nos extremos do intervalo para garantir que o zero da função está contido no novo intervalo resultante da bisseção.

Ao realizar várias iterações do Método da Bisseção, podemos aproximar o zero da função com a precisão desejada, sem a necessidade de calcular derivadas ou outros conceitos avançados de análise matemática.

Vamos exemplificar o método buscando a raiz quadrada de 9, que é equivalente a encontrar o zero da função $f(x) = x^2 - 9$. Embora já saibamos que a raiz quadrada de 9 é 3, utilizaremos este exemplo para demonstrar como o Método da Bisseção funciona. Consideramos o intervalo entre 1 e 4, onde sabemos que a raiz está contida. Em cada iteração, ajustamos os extremos do intervalo para garantir que o zero da função permaneça dentro dele, ou seja, um extremo deve ser menor que 3 e o outro maior que 3. Assim, o Método da Bisseção permite refinar a estimativa da raiz de forma sistemática.

EXEMPLO 4.3.1: Encontre um valor aproximado para um zero da função $f(x) = x^2 - 9$, utilizando o Método da Bisseção no intervalo entre 1 e 4.

Vamos buscar o zero da função $f(x) = x^2 - 9$, que se trata de buscar o valor da raiz quadrada de 9 no intervalo entre 1 e 4. Para isso, avaliamos $f(1)$ e $f(4)$:

$$f(1) = 1^2 - 9 = -8 \quad \text{e} \quad f(4) = 4^2 - 9 = 7.$$

Como $f(1) < 0$ e $f(4) > 0$, sabemos que existe uma raiz no intervalo entre 1 e 4.

A primeira iteração consiste em calcular o ponto médio

$$x_1 = \frac{1+4}{2} = 2{,}5.$$

Agora avaliamos $f(2{,}5)$.

$$f(2{,}5) = 2{,}5^2 - 9 = -2{,}75.$$

Como $f(2{,}5) < 0$ e $f(4) > 0$, o novo intervalo será $[2{,}5; 4]$.

Na segunda iteração, calculamos o novo ponto médio

$$x_2 = \frac{2{,}5 + 4}{2} = 3{,}25.$$

Avaliaremos $f(3{,}25)$:

$$f(3{,}25) = 3{,}25^2 - 9 = 1{,}562.$$

Como $f(2{,}5) < 0$ e $f(3{,}25) > 0$, o novo intervalo será $[2{,}5; 3{,}25]$.

E assim sucessivamente, até encontrarmos um valor que esteja dentro da aproximação desejada para o zero da função. As próximas iterações mostram os seguintes resultados.

$$\begin{aligned}
x_3 &= \frac{2{,}5 + 3{,}25}{2} &&= 2{,}875 \\
x_4 &= \frac{2{,}875 + 3{,}25}{2} &&= 3{,}0625 \\
x_5 &= \frac{2{,}875 + 3{,}0625}{2} &&= 2{,}96875 \\
x_6 &= \frac{2{,}96875 + 3{,}0625}{2} &&= 3{,}0156 \\
x_7 &= \frac{2{,}96875 + 3{,}0156}{2} &&= 2{,}9921 \\
x_8 &= \frac{2{,}9921 + 3{,}0156}{2} &&= 3{,}0039.
\end{aligned}$$

Podemos observar que após 8 iterações conseguimos a aproximação para a raiz quadrada de 9 com duas casas decimais de precisão.

Geometricamente o Método da Bisseção pode ser apresentado pelas Figuras 4.4, 4.5, 4.6 e 4.7, apresentando quatro iterações no gráfico da função $f(x) = x^2 - 9$ com um zero em $x = 3$ (código).

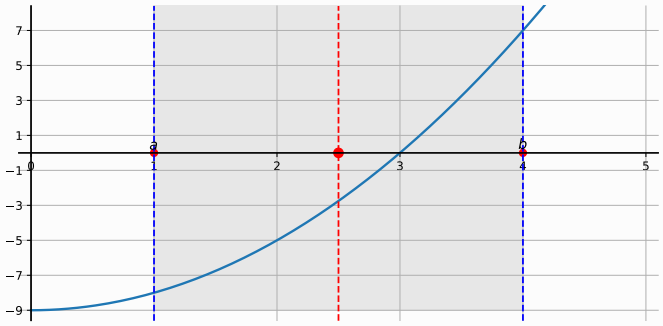


Figura 4.4: Iteração 1: $x_1 = 2{,}5$.

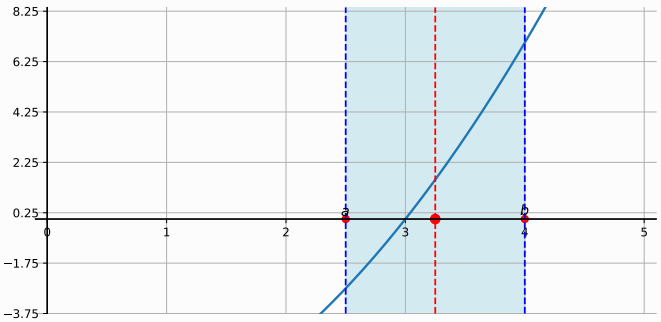


Figura 4.5: Iteração 2: $x_2 = 3{,}25$.

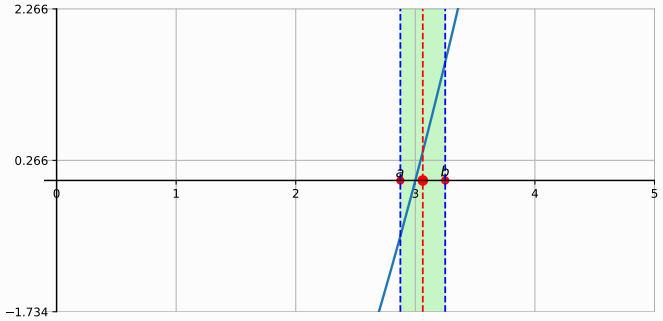


Figura 4.6: Iteração 3: $x_2 = 2{,}875$.

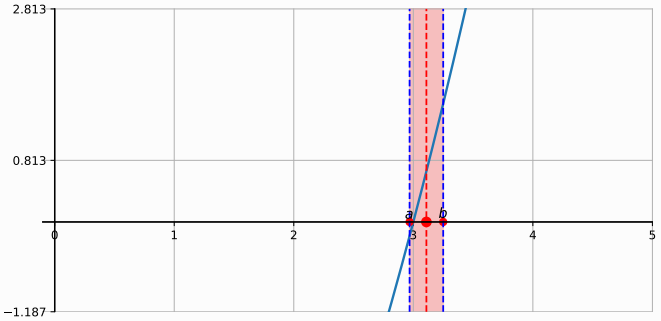


Figura 4.7: Iteração 4: $x_2 = 3{,}065$.

Podemos observar que os intervalos estão diminuindo em torno do zero da função, ilustrando claramente a convergência do método.

Vamos agora mostrar um exemplo com um grau de dificuldade maior, vamos em busca do zero da função exponencial $f(x) = 2^x - 3$.

Podemos verificar que $f(1) = 1^2 - 9 = -8$ e $f(4) = 4^2 - 9 = 7$. Como a função é contínua e há uma mudança de sinal nos extremos do intervalo, existe pelo menos um zero da função nesse intervalo. Este exemplo seria mais difícil com o Método de Newton, pois requer o cálculo da derivada, que envolve conceitos avançados. Portanto, usaremos o Método da Bisseção e aplicaremos um código especialmente desenvolvido para resolver esse exemplo (código).

```
1 # Definindo a função
2 def f(x):
3     return 2**x - 3
4
5 # Definindo o Método da Bisseção
6 def bissecao(a, b, num_iter):
7     for i in range(num_iter):
```

```
        c = (a + b) / 2
        print(f"Iteração {i + 1}: Valor de c = {c}")

        if f(c) == 0:
            return c
        elif f(a) * f(c) < 0:
            b = c
        else:
            a = c

    return (a + b) / 2
# Intervalos iniciais e as iterações
a = 1
b = 2
iteracoes = 8

# Chama a função Bisseção e imprime os resultados
zero = bissecao(a, b, iteracoes)
print(f"O zero da função é aproximadamente {zero}")
```

No código, definimos a função $f(x) = 2^x - 3$ e a função `bissecao(a, b, iter)`, que recebe os extremos do intervalo `a` e `b`, e o número de iterações. Usamos um *loop* `for` para iterar o Método da Bisseção. Calculamos o ponto médio `c` do intervalo e verificamos `f(c)`. Se `f(c) == 0`, retornamos `c`. Caso contrário, atualizamos os extremos do intervalo conforme `f(a)*f(c)`. Após as iterações, retornamos o ponto médio final como a aproximação do zero da função. No exemplo, `a=1`, `b=2`, e são realizadas 8 iterações. O resultado final é impresso como `O zero da função é aproximadamente {zero}`.

Ao executar esse código, após 8 iterações obtemos a seguinte resposta do programa.

```
Iteração 1: Valor de c = 1.50000
Iteração 2: Valor de c = 1.75000
Iteração 3: Valor de c = 1.62500
Iteração 4: Valor de c = 1.56250
Iteração 5: Valor de c = 1.59375
Iteração 6: Valor de c = 1.57812
Iteração 7: Valor de c = 1.58594
Iteração 8: Valor de c = 1.58203
O zero da função é aproximadamente 1.58398
```

Mostrando assim que uma aproximação para um zero da função será 1,58398. Podemos verificar isso substituindo esse valor na função e verificando se o resultado

se aproxima de zero. Temos então que $2^{1,58398} - 3 = -0{,}002$ que é bem próximo de zero.

O Método da Bisseção é conhecido por sua simplicidade e acessibilidade na busca por zeros de funções dentro de um intervalo específico, embora demande mais iterações para alcançar a precisão desejada em comparação a outros métodos. Por outro lado, o Método de Newton oferece uma convergência mais rápida, geralmente exigindo menos iterações, mas requer o cálculo da derivada da função, o que pode ser um obstáculo para quem não está familiarizado com conceitos matemáticos avançados.

Na próxima seção, apresentaremos o Método da Secante como uma alternativa ao Método de Newton. Embora a Secante precise de mais iterações que o Método de Newton, ela é mais rápida que a Bisseção e eficaz em diversas análises numéricas, especialmente quando o cálculo da derivada não é prático.

4.4 Método da Secante

Nesta seção, apresentaremos o Método da Secante, uma técnica iterativa para encontrar zeros de funções sem precisar calcular a derivada. Diferente do Método de Newton, a Secante aproxima a derivada usando dois pontos próximos, tornando-o mais acessível para obter soluções aproximadas dentro de um intervalo.

A fórmula de iteração do Método da Secante é dada por

$$x_{n+1} = x_n - f(x_n)\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}, \tag{4.3}$$

onde x_{n+1} é a próxima estimativa da solução, x_n e x_{n-1} são as estimativas atual e anterior, $f(x_n)$ e $f(x_{n-1})$ são os valores da função nos pontos x_n e x_{n-1}.

Vamos usar o Método da Secante para encontrar um zero da função $f(x) = x^2 - 9$ com estimativas iniciais de $x_0 = 1$ e $x_1 = 4$. Se trata do problema de encontrar a raiz quadrada de 9 no intervalo entre 1 e 4, esse problema é solucionado com os métodos de Newton e da Bisseção, porém vamos começar com esse exemplo para apresentar o Método da Secante e também nos servirá como forma de comparação dos métodos.

EXEMPLO 4.4.1: Encontre um valor aproximado para um zero da função $f(x) = x^2 - 9$, utilizando o Método da Secante.

Vamos calcular um zero da função $f(x) = x^2 - 9$ utilizando o Método da Secante com estimativas iniciais $x_0 = 1$ e $x_1 = 4$. As iterações serão

$$x_2 = 4 - \frac{(4^2 - 9) \cdot (4 - 1)}{(4^2 - 9) - (1^2 - 9)} = 2{,}6$$

$$x_3 = 4 - \frac{(4^2 - 9) \cdot (4 - 2{,}6)}{(4^2 - 9) - (2{,}6^2 - 9)} = 2{,}9393$$

$$x_4 = 4 - \frac{(4^2 - 9) \cdot (4 - 2{,}9394)}{(4^2 - 9) - (2{,}9394^2 - 9)} = 2{,}9912$$

$$x_5 = 4 - \frac{(4^2 - 9) \cdot (4 - 2{,}9912)}{(4^2 - 9) - (2{,}9912^2 - 9)} = 2{,}9987$$

$$x_6 = 4 - \frac{(4^2 - 9) \cdot (4 - 2{,}9987)}{(4^2 - 9) - (2{,}9987^2 - 9)} = 2{,}9998.$$

Após 5 iterações, obtemos uma estimativa $2{,}9998$ para a raiz quadrada de 9, ou seja, bem próximo de 3.

Vamos agora encontrar uma aproximação para um zero da função $f(x) = 3^{x/3} - 4$. Vamos então usar o Método da Secante, mas dessa vez usaremos a programação para isso. O código a seguir é um código escrito especialmente para resolver esse exemplo usando o Método da Secante no intervalo entre 1 e 7 (código).

```
# Define a função f(x)
def f(x):
    return 3**(x/3) - 4

# Implementa o Método da Secante
def secante(x0, x1, iteracoes):
    for i in range(iteracoes):
        x2 = x1 - f(x1) * (x1 - x0) / (f(x1) - f(x0))
        print(f" x{i+2} = {x2}")
        x0 = x1
        x1 = x2
    return x2

# Configuração inicial e execução do método
x0 = 1
x1 = 7
```

```
17 iteracoes = 7
18
19 resultado = secante(x0, x1, iteracoes)
20
21 # Imprime o resultado final
22 print(f"O zero da função é aproximadamente {resultado}")
```

Neste código, definimos a função `f(x)` cujo zero desejamos encontrar, $f(x) = 3^{x/3} - 4$. A função `secante(x0, x1, iteracoes)` executa o Método da Secante com duas estimativas iniciais `x0` e `x1` e um número especificado de iterações. Dentro do *loop* `for`, calculamos as aproximações do zero usando a fórmula do método e atualizamos `x0` e `x1` a cada iteração. O programa inicia com `x0=1` e `x1=7` e realiza 7 iterações, retornando e imprimindo a aproximação do zero como `O zero da função é aproximadamente {resultado}`.

Com esses dados, o programa gera o resultado a seguir.

```
1 x2 = 2.33008
2 x3 = 3.05592
3 x4 = 4.00844
4 x5 = 3.75494
5 x6 = 3.78434
6 x7 = 3.78559
7 x8 = 3.78558
8 O zero da função é aproximadamente 3.78558
```

Observe que a partir da sétima iteração, as quatro primeiras casas decimais não se alteram, ou seja, temos uma aproximação com quatro casas decimais de precisão. Geometricamente, a Figura 4.8 mostra como as interseções das secantes com o eixo x se aproximam gradualmente do zero da função (código).

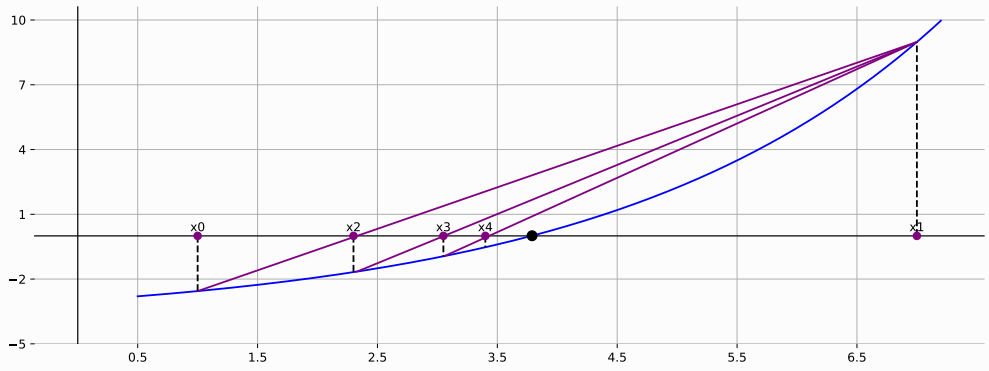


Figura 4.8: Interpretação geométrica do Método da Secante.

Ao comparar os métodos de Newton, Bisseção e Secante, observamos características distintas:

- **Método de Newton**: Converge rapidamente e geralmente requer menos iterações para encontrar uma solução. No entanto, exige o cálculo da derivada, o que pode ser um obstáculo para quem não está familiarizado com cálculo avançado. É importante notar que o Método de Newton pode não convergir se a escolha inicial estiver longe da solução ou se a função não atender às condições necessárias.
- **Método da Bisseção**: Simples e robusto, não necessita da derivada e é aplicável a uma ampla gama de funções. Se a função for contínua no intervalo considerado, o Método da Bisseção garante a convergência, embora possa exigir muitas iterações para alcançar a precisão desejada.
- **Método da Secante**: Oferece uma convergência mais rápida do que a Bisseção, sem precisar calcular a derivada, tornando-o uma alternativa intermediária em termos de velocidade e complexidade. No entanto, o Método da Secante também pode não convergir em alguns casos, dependendo da escolha inicial e das características da função.

Esses métodos possuem suas vantagens e desvantagens, tornando-os mais ou menos adequados dependendo das necessidades específicas do problema.

No próximo capítulo, abordaremos métodos numéricos iterativos para sistemas de equações lineares e não lineares. Esses métodos são essenciais para resolver sistemas complexos encontrados na prática, envolvendo múltiplas variáveis e relações não lineares. Eles são fundamentais em matemática, ciência e engenharia.

5

Métodos Numéricos para Sistemas de Equações

Sistemas de equações algébricas desempenham um papel importante em várias áreas do conhecimento, oferecendo uma poderosa ferramenta para modelar e resolver uma ampla variedade de problemas do mundo real. Neste capítulo, exploraremos duas categorias fundamentais de sistemas de equações: os sistemas lineares e os sistemas não lineares. Compreender a distinção entre essas duas categorias é importante, uma vez que a natureza das equações e os métodos de resolução variam significativamente.

Sistemas de equações lineares envolvem equações com várias variáveis, onde cada variável é multiplicada por uma constante. Para resolver esses sistemas, o número de equações deve ser igual ao número de variáveis, permitindo encontrar os valores que satisfazem todas as equações ao mesmo tempo. No caso de duas variáveis, cada equação representa uma reta, e a solução do sistema é o ponto onde essas retas se cruzam. Com três variáveis, as equações representam planos, e a solução é o ponto onde esses planos se encontram.

Vamos abordar a seguir algumas propriedades e métodos de resolução de sistemas de equações, lineares e não lineares. Ao entender com mais detalhes cada categoria, teremos mais ferramentas e abordagens para resolver uma variedade de problemas complexos que envolvem tais sistemas.

5.1 Sistemas de Equações Lineares

Os sistemas de equações lineares consistem em um conjunto de equações lineares que envolvem as mesmas variáveis. Cada equação no sistema é uma combinação linear dessas variáveis, por exemplo, uma equação com três variáveis pode ser expressa na forma $a_1x + a_2y + a_3z = b$, onde a_1, a_2, a_3 e b são coeficientes constantes. A resolução de um sistema de equações lineares busca encontrar os valores das variáveis que satisfazem simultaneamente todas as equações no sistema. A classificação desses sistemas é baseada no número de soluções que eles admitem. Essa classificação é importante para entender a natureza das soluções e como elas podem ser encontradas.

Sistema Possível: Um sistema possível é aquele que possui pelo menos uma solução, ou seja, existe um conjunto de valores para as variáveis que satisfaz todas as equações do sistema simultaneamente. Este tipo de sistema pode ser classificado de duas formas: determinado e indeterminado. Um sistema é determinado quando possui exatamente uma solução única, o que significa que graficamente as retas que representam as equações se intersectam em um único ponto. Por outro lado, um sistema é indeterminado quando possui infinitas soluções, e graficamente as retas coincidem, indicando que todos os pontos ao longo da reta são soluções do sistema.

Sistema Impossível: Um sistema impossível é aquele que não possui solução. Isso ocorre quando não existe um conjunto de valores para as variáveis que satisfaça todas as equações do sistema simultaneamente, o que acontece graficamente quando as retas são paralelas e não se intersectam.

Vamos analisar três exemplos de sistemas de equações lineares, cada um representando um caso diferente, e explorar sua classificação e representação geométrica no plano cartesiano. Para simplificar a apresentação, focaremos em sistemas 2×2, ou seja, sistemas com duas equações e duas variáveis.

O primeiro exemplo abordará um sistema possível e determinado, ilustrando sua representação no plano cartesiano.

EXEMPLO 5.1.1: Classifique o sistema linear abaixo através da análise da sua representação geométrica no plano cartesiano.

$$\begin{cases} x + y = 6 \\ x - y = 2 \end{cases} \tag{5.1}$$

A Figura 5.1 apresenta geometricamente esse sistema, mostrando as duas retas geradas pelas equações do sistema, bem como o único ponto de interseção entre elas (código).

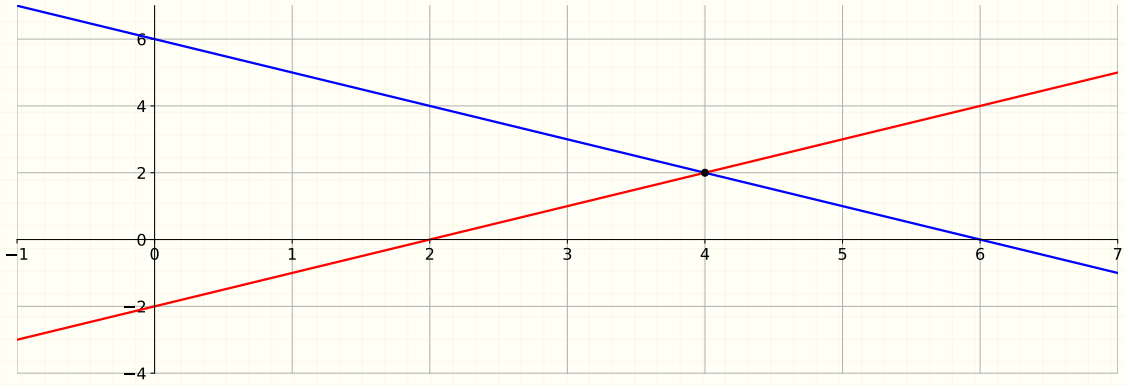


Figura 5.1: Representação geométrica do sistema (5.1).

Há somente um ponto que satisfaz as duas equações simultaneamente. Logo, o sistema (5.1) é um sistema possível e determinado.

No próximo exemplo vamos ver um caso onde o sistema é possível, porém indeterminado. Apresentando também sua representação geométrica no plano cartesiano.

EXEMPLO 5.1.2: Classifique o sistema linear abaixo através da análise da sua representação geométrica no plano cartesiano.

$$\begin{cases} x + y = 1 \\ 2x + 2y = 2 \end{cases} \tag{5.2}$$

A Figura 5.2 apresenta geometricamente esse sistema, mostrando as duas retas geradas pelas equações do sistema, e elas estão sobrepostas, ou seja, todos os pontos pertencentes a uma das retas também pertencem à outra reta (código).

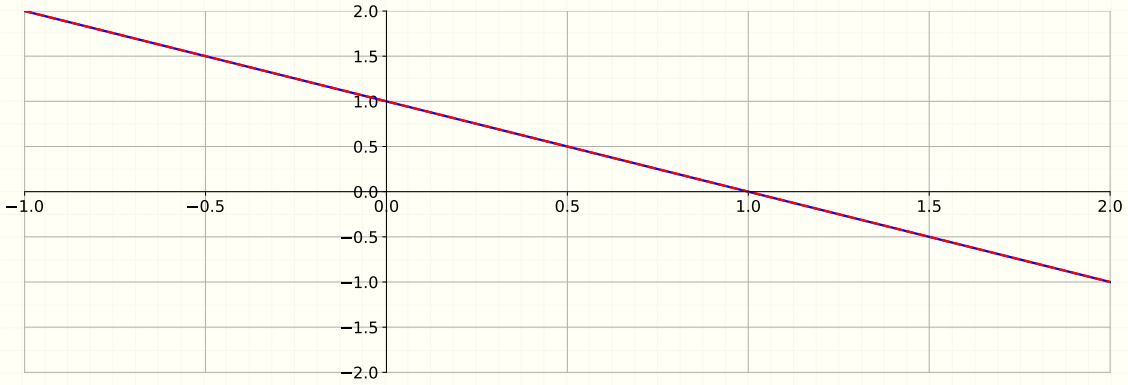


Figura 5.2: Representação geométrica do sistema (5.2).

O sistema (5.2) é possível e indeterminado, pois possui infinitas soluções.

O próximo exemplo trata de um sistema impossível, pois não há nenhum ponto em comum para ambas as retas.

Exemplo 5.1.3: Classifique o sistema linear abaixo através da análise da sua representação geométrica no plano cartesiano.

$$\begin{cases} x + y = 1 \\ x + y = 4 \end{cases} \tag{5.3}$$

A Figura 5.3 apresenta geometricamente esse sistema, mostrando as duas retas geradas pelas equações do sistema, que são paralelas entre si, ou seja, não possuem nenhum ponto em comum (código).

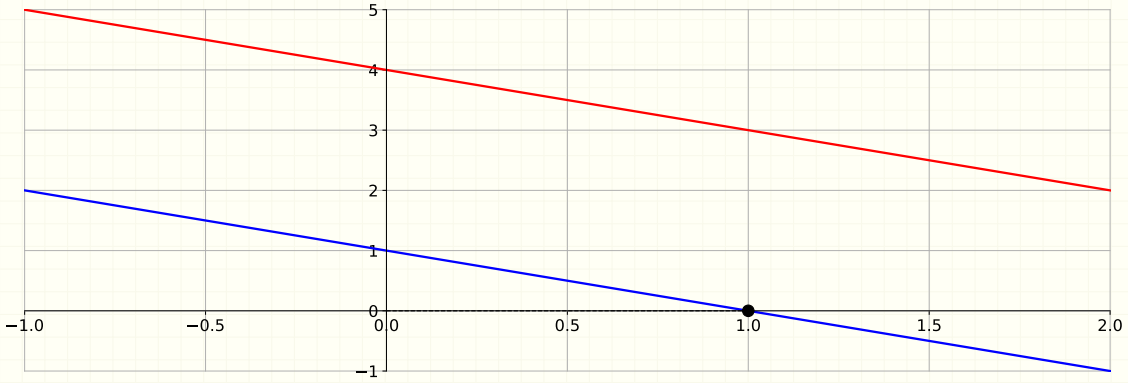


Figura 5.3: Representação geométrica do sistema (5.3)

O sistema (5.3) é impossível, pois não há nenhum ponto que satisfaça as duas equações simultaneamente.

Na matemática e na ciência, frequentemente encontramos sistemas de equações lineares que precisam ser resolvidos para solucionar problemas do mundo real. No entanto, resolver esses sistemas de forma direta e analítica nem sempre é viável, especialmente quando são grandes e complexos. É nesse contexto que os métodos numéricos se tornam essenciais. A seguir vamos explorar o Método de Jacobi, uma técnica computacional iterativa clássica que nos permite encontrar soluções aproximadas para sistemas lineares, facilitando a resolução de problemas que seriam intratáveis por métodos analíticos diretos.

Método de Jacobi

O Método de Jacobi é uma técnica iterativa usada para resolver sistemas de equações lineares, especialmente os grandes e complexos. Baseia-se na atualização sucessiva das estimativas das variáveis, utilizando as equações do sistema para aproximar as soluções. É útil quando métodos diretos são impraticáveis, permitindo obter soluções precisas através de iterações sucessivas. Para simplificar a apresentação, mostraremos os passos do método em um sistema 2×2, ou seja, sistemas com duas equações e duas variáveis, porém o método pode ser aplicado em sistemas maiores.

O passo a passo do Método de Jacobi começa com a inicialização das estimativas para x e y. Para cada iteração, isola-se x na primeira equação e utiliza-se as estimativas atuais de y para calcular um novo valor de x. Da mesma forma, isola-se y na segunda equação e usa-se as estimativas atuais de x para calcular um novo valor de y. Repete-se essas etapas até que as estimativas de x e y se estabilizem o suficiente para serem consideradas a solução. No Método de Jacobi, x e y são atualizados separadamente a cada iteração, com base nas equações do sistema e isolando cada variável em sua respectiva equação.

Considere o sistema de equações lineares a seguir:

$$\begin{cases} 2x - y = 1 \\ x + 2y = 3. \end{cases} \tag{5.4}$$

A Figura 5.4 apresenta geometricamente esse sistema, mostrando as duas retas geradas pelas equações do sistema, bem como o único ponto de interseção entre elas (código).

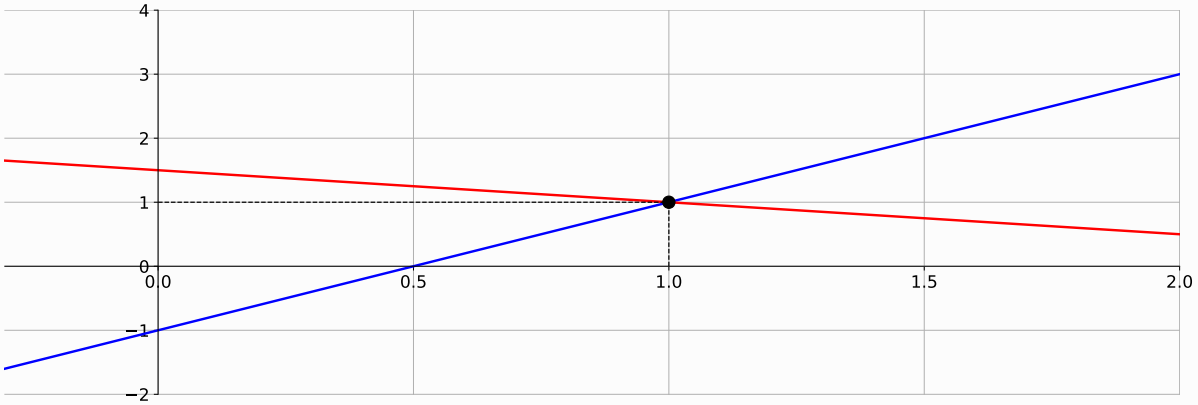


Figura 5.4: Representação geométrica do sistema (5.4).

Podemos verificar que a solução para o sistema é o par ordenado $(1, 1)$, ou seja, $x = 1$ e $y = 1$, pois substituindo esses valores nas equações do sistema

$$\begin{cases} 2 \cdot 1 - 1 = 1 \\ 1 + 2 \cdot 1 = 3, \end{cases}$$

verificamos que ambas as igualdades são verdadeiras, validando a solução.

Vamos encontrar a solução aproximada do sistema (5.4) como exemplo, aplicando o Método de Jacobi. Isso nos ajudará a entender como o método usa estimativas iniciais e as aproxima cada vez mais da solução correta do sistema.

EXEMPLO 5.1.4: Encontre uma solução aproximada para o sistema de equações lineares a seguir usando o Método de Jacobi.

$$\begin{cases} 2x - y = 1 \\ x + 2y = 3. \end{cases}$$

Vamos aplicar o Método de Jacobi para aproximar a solução desse sistema de equações lineares.

Podemos reescrever o sistema da seguinte forma:

$$\begin{cases} x = \dfrac{y + 1}{2} \\ y = \dfrac{3 - x}{2}. \end{cases}$$

Assim o processo de iteração do Método de Jacobi será:

$$x_{n+1} = \frac{y_n + 1}{2} \tag{5.5}$$

$$y_{n+1} = \frac{3 - x_n}{2}. \tag{5.6}$$

Iniciaremos o processo de iteração do método com as estimativas iniciais $x_0 = 0$ e $y_0 = 0$. Para cada iteração subsequente, utilizaremos os valores encontrados na iteração anterior para atualizar as estimativas de x e y, conforme as fórmulas das equações (5.5) e (5.6). Vamos executar 5 iterações para observar como as estimativas se aproximam da solução do sistema.

1. Primeira iteração, usando as estimativas iniciais $x_0 = 0$ e $y_0 = 0$:

$$x_1 = \frac{y_0 + 1}{2} = \frac{0 + 1}{2} = 0{,}5$$

$$y_1 = \frac{3 - x_0}{2} = \frac{3 - 0}{2} = 1{,}5.$$

2. Segunda iteração, usando $x_1 = 0{,}5$ e $y_1 = 1{,}5$:

$$x_2 = \frac{y_1 + 1}{2} = \frac{1{,}5 + 1}{2} = 1{,}25$$

$$y_2 = \frac{3 - x_1}{2} = \frac{3 - 0{,}5}{2} = 1{,}25.$$

3. Terceira iteração, usando $x_2 = 1{,}25$ e $y_2 = 1{,}25$:

$$x_3 = \frac{y_2 + 1}{2} = \frac{1{,}25 + 1}{2} = 1{,}125$$

$$y_3 = \frac{3 - x_2}{2} = \frac{3 - 1{,}25}{2} = 0{,}875.$$

4. Quarta iteração, usando $x_3 = 1{,}125$ e $y_3 = 0{,}875$:

$$x_4 = \frac{y_3 + 1}{2} = \frac{0{,}875 + 1}{2} \approx 0{,}937$$

$$y_4 = \frac{3 - x_3}{2} = \frac{3 - 1{,}125}{2} \approx 0{,}937.$$

5. Quinta iteração, usando $x_4 \approx 0{,}937$ e $y_4 \approx 0{,}937$:

$$x_5 = \frac{y_4 + 1}{2} = \frac{0{,}937 + 1}{2} \approx 0{,}968$$

$$y_5 = \frac{3 - x_4}{2} = \frac{3 - 0{,}937}{2} \approx 1{,}031.$$

Podemos observar que a cada iteração os valores se aproximam mais da solução do sistema que é $x = 1$ e $y = 1$. Repetimos o processo até atingir a aproximação desejada.

A Figura 5.5 mostra geometricamente como os resultados das iterações vão se aproximando da solução do sistema (código).

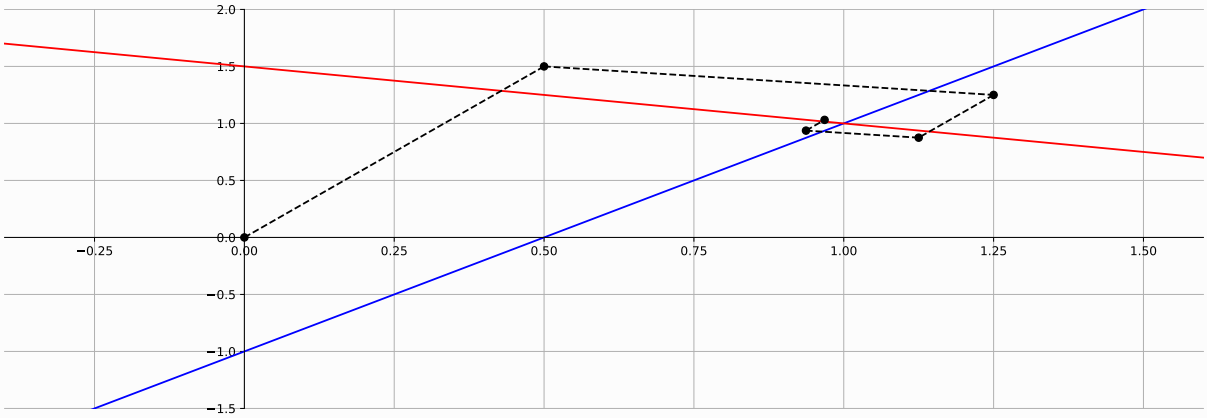


Figura 5.5: Representação geométrica do Método de Jacobi.

Para atingir uma melhor aproximação vamos escrever um programa especialmente para esse sistema usando o Método de Jacobi. Para isso o programa solicitará ao usuário a estimativa inicial para x e a estimativa inicial para y e em seguida a aproximação desejada (código).

```
1 #Programa para aproximar a solução do sistema com o Método de
    Jacobi
2
3 # Função para calcular o próximo valor de x usando o Método de
    Jacobi
4 def jacobi_iteracao(x, y):
5     x_proximo = (y + 1) / 2
6     y_proximo = (3 - x) / 2
7     return x_proximo, y_proximo
8
```

```
# Entrada das estimativas iniciais e da tolerância
x0 = float(input("Digite a estimativa inicial para x: "))
y0 = float(input("Digite a estimativa inicial para y: "))
tolerancia = float(input("Digite a tolerância desejada: "))

# Execução das iterações do Método de Jacobi
max_iteracoes = 100  # Número máximo de iterações
iteracao = 0
x = x0
y = y0
while iteracao < max_iteracoes:
    x_proximo, y_proximo = jacobi_iteracao(x, y)
    print(f"Iteração {iteracao + 1}: x = {x_proximo}, y = {y_proximo}")
    if abs(x_proximo - x) < tolerancia and abs(y_proximo - y) < tolerancia:
        break
    x = x_proximo
    y = y_proximo
    iteracao += 1

# Resultado das iterações
print(f"\nResultado após {iteracao} iterações:")
print(f"x = {x}, y = {y}")
```

O programa utiliza o Método de Jacobi para aproximar a solução de um sistema de equações. A função `jacobi_iteracao` calcula os próximos valores de `x` e `y` com base nas fórmulas do método. O usuário fornece as estimativas iniciais e a tolerância desejada através dos comandos `input`. O *loop* `while` executa o método até que a diferença entre os valores sucessivos de `x` e `y` seja menor que a tolerância ou o número máximo de iterações seja alcançado. Os novos valores são exibidos a cada iteração. Quando a precisão desejada é atingida, o programa mostra os valores finais e o número total de iterações realizadas.

Supondo que o usuário digite $x_0 = 0$ e $y_0 = 0$ para os valores iniciais e 0.001 para a tolerância, ou seja, desejando uma aproximação de 3 casas decimais de precisão, a resposta do programa será

```
Digite a estimativa inicial para x: 0
Digite a estimativa inicial para y: 0
Digite a tolerância desejada: 0.001
Iteração 1: x = 0.5, y = 1.5
```

```
Iteração 2: x = 1.25, y = 1.25
Iteração 3: x = 1.125, y = 0.875
Iteração 4: x = 0.9375, y = 0.9375
Iteração 5: x = 0.96875, y = 1.03125
Iteração 6: x = 1.015625, y = 1.015625
Iteração 7: x = 1.0078125, y = 0.9921875
Iteração 8: x = 0.99609375, y = 0.99609375
Iteração 9: x = 0.998046875, y = 1.001953125
Iteração 10: x = 1.0009765625, y = 1.0009765625
Iteração 11: x = 1.00048828125, y = 0.99951171875
Iteração 12: x = 0.999755859375, y = 0.999755859375

Resultado após 11 iterações:
x = 1.00048828125, y = 0.99951171875
```

Podemos observar que o programa encerrou as iterações para $x = 1{,}000$ e $y = 0{,}999$, ou seja, com 3 casas decimais de precisão como foi solicitado.

O Método de Jacobi, embora atualmente tenha sido substituído por técnicas mais avançadas, ainda é um ponto de partida importante para compreender a resolução de sistemas de equações lineares. Ele serviu de inspiração para o desenvolvimento de métodos numéricos iterativos mais recentes.

Na próxima seção, vamos abordar método numéricos para aproximar soluções de sistemas de equações não lineares e vamos apresentar o Método de Newton, expandindo ainda mais nosso conjunto de técnicas para lidar com diferentes tipos de problemas matemáticos envolvendo sistemas de equações.

5.2 Sistemas de Equações Não Lineares

Nesta seção, exploraremos os sistemas de equações não lineares. Diferentemente das equações lineares, cujas relações entre as variáveis são representadas por retas, as equações não lineares envolvem relações mais complexas, como quadráticas, cúbicas, exponenciais, entre outras formas. Esses sistemas são frequentemente encontrados em áreas como física, biologia, economia e engenharia. Em seguida, apresentaremos o Método de Newton, uma técnica poderosa para resolver sistemas de equações não lineares. É importante destacar que este método é uma generalização do Método de Newton, discutido no Capítulo 4, que foi originalmente desenvolvido para resolver uma única equação não linear. A generalização permite aplicar a técnica ao contexto

de sistemas de equações não lineares, oferecendo uma abordagem eficaz para lidar com a complexidade matemática desses problemas.

Método de Newton para Sistemas Não Lineares

O Método de Newton para sistemas não lineares começa com uma estimativa inicial para a solução do sistema. Com base nessa estimativa, calculamos uma nova estimativa aplicando a fórmula do método, que envolve a matriz Jacobiana, uma matriz formada pelas derivadas das funções do sistema e seu nome "matriz Jacobiana" é em homenagem ao matemático Jacobi. O método usa a fórmula iterativa para atualizar a estimativa: a nova estimativa é obtida subtraindo a inversa da matriz Jacobiana multiplicada pelo vetor das funções avaliadas na estimativa atual. Cada nova estimativa é então usada para recalcular a matriz Jacobiana e o vetor das funções, e o processo é repetido. Cada iteração geralmente aproxima a estimativa da solução correta, e o processo continua até que a diferença entre as estimativas sucessivas seja suficientemente pequena, alcançando a precisão desejada.

Todo esse conhecimento matemático geralmente faz parte do cálculo avançado, estudado no ensino superior. No entanto, vamos tentar mostrar os passos desse método com funções mais simples, para que, mesmo sem esse conhecimento avançado, seja possível entender o funcionamento do método.

Vamos considerar um sistema não linear 2×2 para ilustrar o Método de Newton. Considere o seguinte sistema:

$$\begin{cases} y = -x^2 - 2x + 7 \\ y = 2^x + 1. \end{cases} \tag{5.7}$$

Observe que este sistema é difícil de ser solucionado de forma analítica. A Figura 5.6 apresenta geometricamente esse sistema e mostra o gráfico das duas equações e os pontos de interceção entre elas. Observe que se trata de duas curvas, ou seja, equações não lineares (código).

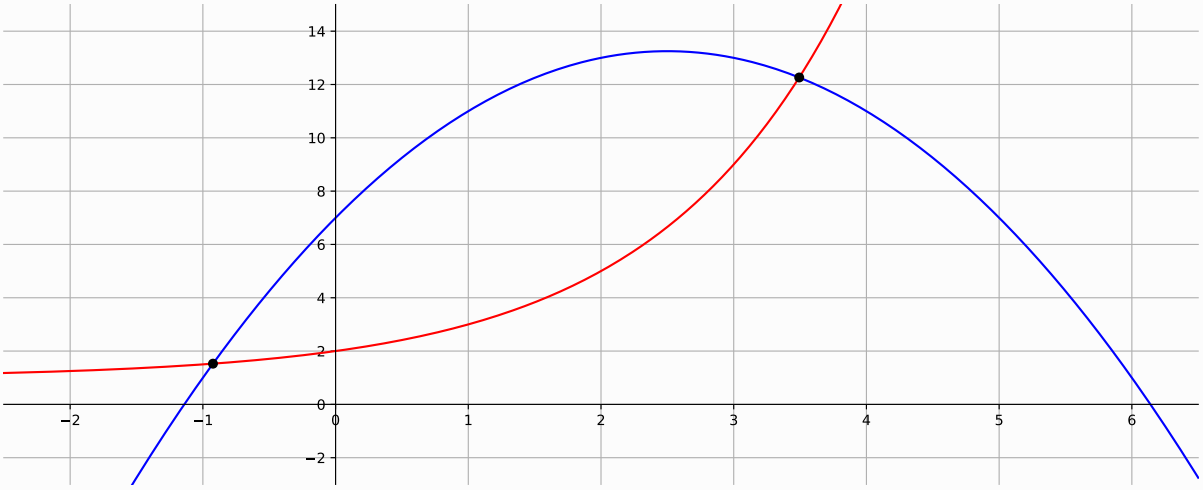


Figura 5.6: Representação geométrica do sistema (5.7).

Podemos observar que existem dois pontos de intercessão entre as curvas geradas pelas funções mostradas no gráfico, ou seja, o sistema possui duas soluções: uma onde o valor de x está entre -1 e 0 e y está entre 1 e 2, e outra onde o valor de x está entre 3 e 4 e y está entre 12 e 13. A utilização de métodos gráficos é muito cara computacionalmente. Por exemplo, para construir esse gráfico, foram necessários 400 pontos. Para funções grandes e complexas, isso pode exigir muitos recursos computacionais. Uma das vantagens dos métodos numéricos é que eles requerem uma quantidade muito menor de iterações para se aproximar da solução desejada.

Vamos, no exemplo a seguir aplicar um passo do Método de Newton para aproximar as estimativas iniciais à uma das soluções do sistema.

EXEMPLO 5.2.1: Considere o sistema não linear (5.7) dado por

$$\begin{cases} y = -x^2 + 5x + 7 \\ y = 2^x + 1. \end{cases}$$

Desenvolva um passo do Método de Newton para aproximar as estimativas iniciais $x_0 = 3$ e $y_0 = 12$, de uma das soluções do sistema.

Primeiro, definimos as funções f_1 e f_2 do sistema (5.7) como:

$$\begin{cases} f_1(x, y) = -x^2 + 5x + 7 - y \\ f_2(x, y) = 2^x + 1 - y. \end{cases}$$

Calculamos as derivadas parciais e com elas construimos a matriz Jacobiana

$$J = \begin{bmatrix} -2x + 5 & -1 \\ 2^x \ln(2) & -1 \end{bmatrix}.$$

Começamos com as estimativas iniciais $x_0 = 3$ e $y_0 = 12$ e as substituímos nas funções do sistema.

$$f_1(3, 12) = -3^2 + 5 \cdot 3 + 7 - 12 = 1$$

$$f_2(3, 12) = 2^3 + 1 - 12 = -3.$$

Substituímos $x_0 = 3$ na matriz Jacobiana.

$$J = \begin{bmatrix} -2 \cdot 3 + 5 & -1 \\ 2^3 \ln(2) & -1 \end{bmatrix} = \begin{bmatrix} -1 & -1 \\ 5.544 & -1 \end{bmatrix}.$$

A inversa da matriz Jacobiana J é

$$J^{-1} \approx \begin{bmatrix} -0.153 & 0.153 \\ -0.848 & -0.153 \end{bmatrix}.$$

A fórmula de iteração do Método de Newton para sistemas não lineares é dada por

$$\begin{bmatrix} x_{n+1} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} x_n \\ y_n \end{bmatrix} - J^{-1} \begin{bmatrix} f_1(x_n, y_n) \\ f_2(x_n, y_n) \end{bmatrix}.$$

Substituímos os valores e calculando o produto, obtemos as novas estimativas.

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} 3.612 \\ 12.389 \end{bmatrix}$$

Assim, após a primeira iteração, temos as estimativas atualizadas $x_1 \approx 3.612$ e $y_1 \approx 12.389$.

Como o Método de Newton envolve vários passos e cálculos complexos, a programação é uma ferramenta eficaz para automatizar todo esse processo. O código a seguir implementa o Método de Newton para encontrar soluções aproximadas para sistemas de equações não lineares. Ele define as funções do sistema e suas derivadas parciais, utilizando uma função que aplica o Método de Newton com base em estimativas

iniciais fornecidas pelo usuário. Isso facilita a resolução prática e eficiente de sistemas complexos, proporcionando soluções numéricas de forma automatizada.

O código que realiza esse processo é apresentado abaixo (código).

```
def Metodo_Newton(eq1, eq2, partial_eq1_x, partial_eq1_y, partial_eq2_x, partial_eq2_y, x0, y0, tolerance):
    iterations = 0
    results = []

    while True:
        iterations += 1
        f1 = eq1(x0, y0)
        f2 = eq2(x0, y0)

        J = np.array([[partial_eq1_x(x0), partial_eq1_y()],
            [partial_eq2_x(x0), partial_eq2_y()]])

        inv_J = np.linalg.inv(J)
        delta = np.dot(inv_J, np.array([-f1, -f2]))
        x1 = x0 + delta[0]
        y1 = y0 + delta[1]

        results.append((x1, y1))
        print(f"x_{iterations} = {x1:.4f}, y_{iterations} = {y1:.4f}")

        if abs(x1 - x0) < tolerance and abs(y1 - y0) < tolerance:
            break

        x0, y0 = x1, y1

    return x1, y1, iterations
```

Este código define a função `Metodo_Newton`, que aplica o Método de Newton para resolver sistemas de equações não lineares. A função recebe como parâmetros as equações do sistema (`eq1` e `eq2`), suas derivadas parciais (`partial_eq1_x`, `partial_eq1_y`, `partial_eq2_x` e `partial_eq2_y`), estimativas iniciais para `x` e `y`, e a tolerância para a convergência. Em cada iteração, a função calcula os valores das equações, constrói e inverte a matriz Jacobiana para encontrar os incrementos necessários nas variáveis. Esses incrementos são aplicados até que as mudanças entre iterações sejam menores que a tolerância especificada. A função imprime as estimativas de cada iteração e retorna as estimativas finais e o número de iterações realizadas.

Agora, considerando o sistema de equações não lineares (5.7) dado por

$$\begin{cases} y = -x^2 + 5x + 7 \\ y = 2^x + 1, \end{cases}$$

vamos escrever um código para definir as funções que compõem o sistema de equações e suas funções derivadas e especificar as estimativas iniciais e a tolerância desejada. Em seguida, utilizaremos a função Método de Newton, que foi definida anteriormente, para encontrar as soluções aproximadas desse sistema.

O código que realiza esse processo é apresentado abaixo (código).

```
1  import math
2
3  # Funções do sistema de equações
4  def eq1(x, y):
5      return -x**2 + 5*x + 7 - y
6
7  def eq2(x, y):
8      return 2**x + 1 - y
9
10 # Derivadas parciais das funções
11 def partial_eq1_x(x):
12     return -2*x + 5
13
14 def partial_eq1_y():
15     return -1
16
17 def partial_eq2_x(x):
18     return 2**x * math.log(2)
19
20 def partial_eq2_y():
21     return -1
22
23 # Coleta de dados do usuário e chamada da função
24 if __name__ == "__main__":
25     x0 = float(input("Digite a estimativa inicial para x: "))
26     y0 = float(input("Digite a estimativa inicial para y: "))
27     tolerance = float(input("Digite a tolerância para as
       aproximações: "))
28
29     x_final, y_final, num_iterations = Metodo_Newton(
30         eq1, eq2,
```

```
        partial_eq1_x, partial_eq1_y,
        partial_eq2_x, partial_eq2_y,
        x0, y0, tolerance
    )

    print(f"\nApós {num_iterations} iterações temos:")
    print(f"x_{num_iterations} = {x_final:.4f}")
    print(f"y_{num_iterations} = {y_final:.4f}")
```

Este código coleta as entradas do usuário e chama a função `Metodo_Newton`. Ele solicita ao usuário as estimativas iniciais para `x` e `y`, e a tolerância desejada. As funções do sistema de equações e suas derivadas parciais são definidas conforme o sistema a ser resolvido. Após coletar os dados, o código chama a função `Metodo_Newton` com os parâmetros fornecidos e exibe o resultado final, incluindo a solução para `x` e `y` e o número de iterações necessárias para alcançar a precisão desejada.

Vamos iniciar tentando aproximar a solução onde o valor de x se encontra no intervalo entre 3 e 4. Para isso vamos iniciar com as estimativas $x_0 = 3$ e $y_0 = 12$ e com uma aproximação de 3 casas decimais. Após a execução, o programa gera os resultados a seguir.

```
Digite a estimativa inicial para x: 3
Digite a estimativa inicial para y: 12
Digite a tolerância para as aproximações: 0.001
x_1 = 3.6111, y_1 = 12.3889
x_2 = 3.4985, y_2 = 12.2657
x_3 = 3.4935, y_3 = 12.2629
x_4 = 3.4935, y_4 = 12.2629

Após 4 iterações temos:
x_4 = 3.4935
y_4 = 12.2629
```

Após 4 iterações o programa encontrou os resultados aproximados $x = 3{,}493$ e $y = 12{,}263$. Geometricamente a Figura 5.7 mostra como os valores obtidos pelas iterações vão se aproximando da solução do sistema (código).

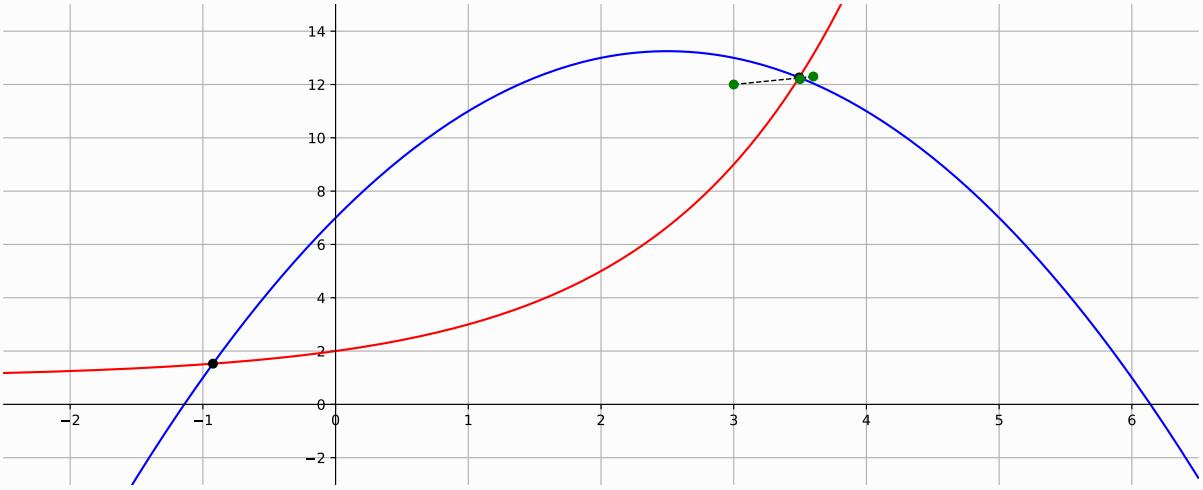


Figura 5.7: Representação geométrica do sistema (5.7) e dos pontos gerados pelo Método de Newton.

Agora vamos iniciar tentando aproximar a solução onde o valor de x se encontrar no intervalo entre -1 e 0. Para isso vamos iniciar com as estimativas $x_0 = 0$ e $y_0 = 0$ e com uma aproximação de 3 casas decimais. Com esses dados iniciais, o programa gera os resultados a seguir.

```
Digite a estimativa inicial para x: 0
Digite a estimativa inicial para y: 0
Digite a tolerância para as aproximações: 0.001
x_1 = -1.1609, y_1 = 1.1953
x_2 = -0.9328, y_2 = 1.5179
x_3 = -0.9239, y_3 = 1.5271
x_4 = -0.9239, y_4 = 1.5271

Após 4 iterações temos:
x_4 = -0.9239
y_4 = 1.5271
```

Após 4 iterações o programa encontrou os resultados aproximados $x = -0{,}924$ e $y = 1{,}527$. Geometricamente a Figura 5.8 mostra como os valores obtidos pelas iterações vão se aproximando da solução do sistema (código).

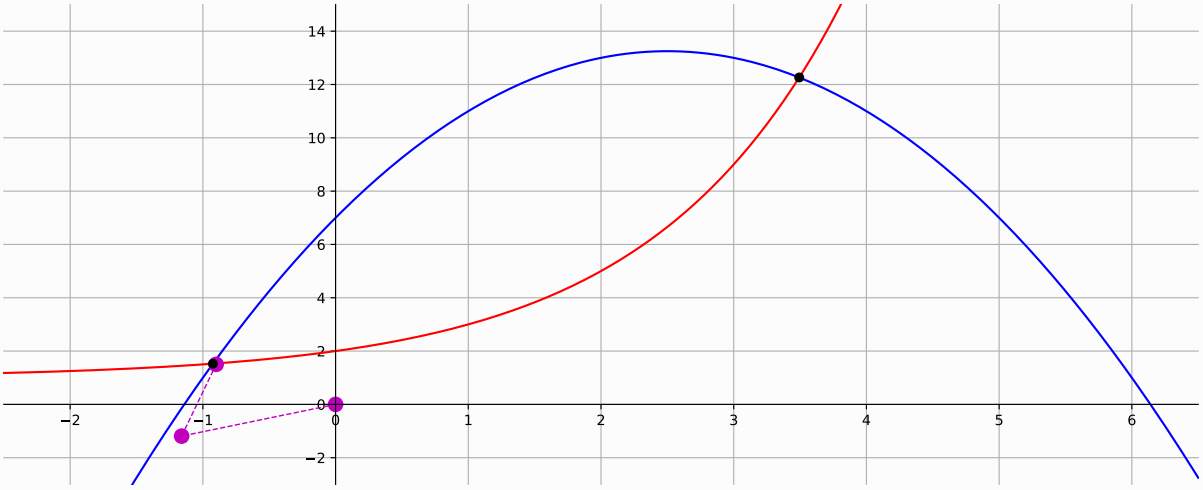


Figura 5.8: Representação geométrica do sistema (5.7) e dos pontos gerados pelo Método de Newton.

Agora, utilizando os programas que desenvolvemos, vamos aplicar o Método de Newton para resolver um exemplo prático de sistema não linear. Este exemplo ilustrará como as soluções aproximadas podem ser obtidas e aplicadas a problemas reais, demonstrando a eficácia do método em contextos práticos.

EXEMPLO 5.2.1: Dois países estão em guerra, um deles lança um míssil que sai da posição $x = 9$ se se movimentando segundo a função $f(x) = -x^2 + 12x - 27$ e tem por objetivo atingir o ponto $x = 3$ de território inimigo. O outro país por sua vês lança um míssil interceptador saindo da posição $x = 0$ se movimentando segundo a função $f(x) = -x^2 + 4x$. Encontre o ponto onde os mísseis se encontraram (código).

O problema se trata de encontrar a solução de um sistemas de equações não lineares escrito como

$$\begin{cases} y = -x^2 + 4x \\ y = -x^2 + 12x - 27 \end{cases}$$

O código que realiza esse processo é apresentado abaixo.

```
1 import numpy as np
2 import math
3
4 # Funções do sistema de equações
5 def eq1(x, y):
```

```
    return -x**2 + 4*x - y

def eq2(x, y):
    return -x**2 + 12*x - 27 - y

# Derivadas parciais das funções
def partial_eq1_x(x):
    return -2*x + 4

def partial_eq1_y():
    return -1

def partial_eq2_x(x):
    return -2*x + 12

def partial_eq2_y():
    return -1

# Função Metodo_Newton foi definida em outro lugar

# Coleta de dados do usuário e chamada da função
if __name__ == "__main__":
    x0 = float(input("Digite a estimativa inicial para x: "))
    y0 = float(input("Digite a estimativa inicial para y: "))
    tolerance = float(input("Digite a tolerância para as
   aproximações: "))

    # Chamada da função Metodo_Newton
    x_final, y_final, num_iterations = Metodo_Newton(
        eq1, eq2,
        partial_eq1_x, partial_eq1_y,
        partial_eq2_x, partial_eq2_y,
        x0, y0, tolerance
    )

    print(f"\nApós {num_iterations} iterações temos:")
    print(f"x_{num_iterations} = {x_final:.6f}")
    print(f"y_{num_iterations} = {y_final:.6f}")
```

Esse código é similar ao código do exemplo anterior, mudando apenas as funções e suas derivadas. Nesse programa será solicitado ao usuário a estimativa iniciais para `x` e `y` e em seguida a aproximação desejada. O programa chamará a função

`Metodo_Newton` que executará os passos de iteração do Método de Newton até que se obtenha a aproximação desejada.

Para a execução do programa, vamos iniciar com as estimativas $x_0 = 3$ e $y_0 = 0$ que é o ponto de impacto do míssil agressor, e com uma aproximação de 6 casas decimais. Após a execução, o programa apresentou os resultados a seguir.

```
Digite a estimativa inicial para x: 3
Digite a estimativa inicial para y: 0
Digite a tolerância para as aproximações: 0.000001
x_1 = 3.3750, y_1 = 2.2500
x_2 = 3.3750, y_2 = 2.1094
x_3 = 3.3750, y_3 = 2.1094

Após 3 iterações temos:
x_3 = 3.375000
y_3 = 2.109375
```

O programa executou 3 iterações obtendo os resultados $x = 3{,}375$ e $y = 2{,}109375$.

A Figura 5.9 mostra geometricamente a movimentação dos dois mísseis e como as iterações do Método de Newton se aproximam cada vez mais do ponto de encontro entre eles (código).

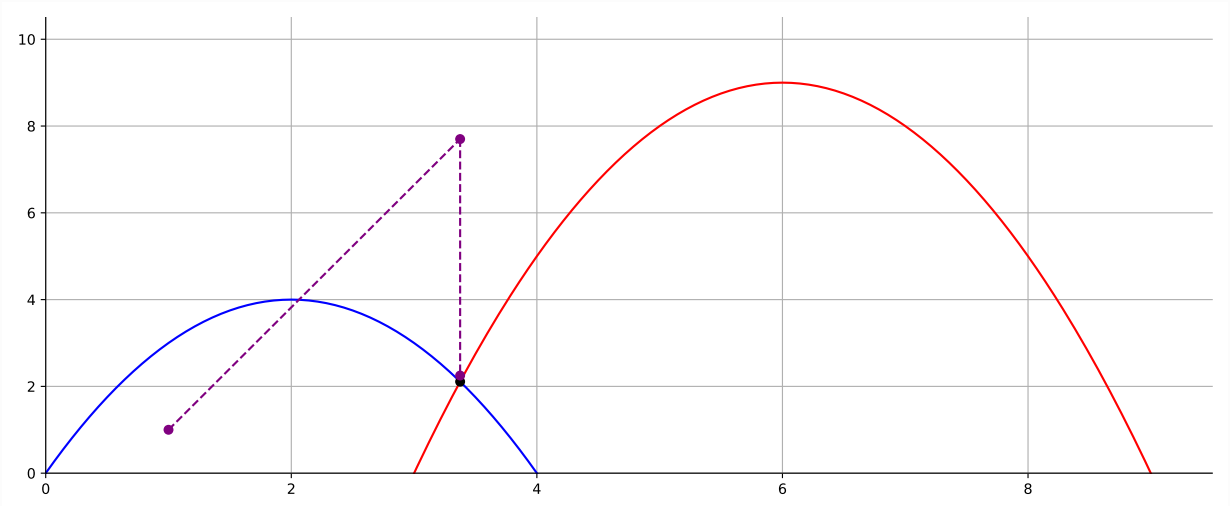


Figura 5.9: Representação geométrica do movimento dos mísseis e dos pontos gerados pelo Método de Newton.

Ao concluir a seção sobre o Método de Newton para sistemas não lineares, é importante ressaltar sua eficácia em resolver equações complexas. Adaptando esse método, podemos abordar uma ampla gama de problemas matemáticos e científicos com relações não lineares. Sua aplicação é importante em várias áreas da ciência e engenharia, superando as limitações dos métodos tradicionais e oferecendo soluções aproximadas de maneira eficiente e precisa.

Encerrando o capítulo sobre métodos iterativos para sistemas lineares e não lineares, podemos destacar a importância dessas técnicas na resolução de problemas complexos e de grande escala. Os métodos iterativos, como o Método de Jacobi e o Método de Newton, proporcionam ferramentas poderosas para encontrar soluções aproximadas de sistemas de equações, sejam eles lineares ou não lineares. Sua eficiência e versatilidade tornam essas técnicas indispensáveis em diversas áreas da matemática aplicada, ciências naturais, engenharias e outras disciplinas que lidam com modelagem e análise de sistemas complexos.

6

Considerações Finais

Ao longo deste trabalho, exploramos desde os conceitos básicos de Python até a aplicação prática de métodos numéricos na resolução de problemas matemáticos, como zeros de funções e sistemas de equações. Python facilita a aprendizagem da programação e permite a implementação direta de algoritmos matemáticos, como o cálculo de raízes quadradas e a solução de sistemas de equações.

As funções são fundamentais tanto na matemática quanto na programação em Python, onde, em Python, elas são blocos de código reutilizáveis que executam tarefas específicas, permitindo uma organização e estruturação mais eficientes do código. Compreender o conceito teórico de funções é essencial, pois isso facilita a implementação prática em Python, tornando a resolução de problemas matemáticos mais clara e eficaz. Essa combinação de teoria e prática é crucial para qualquer programador que busca aplicar a matemática em suas soluções.

Métodos numéricos, como o Método de Newton, são essenciais para encontrar soluções de problemas matemáticos de forma computacional. O Método de Newton, por exemplo, é eficiente na busca por zeros de funções, utilizando uma abordagem iterativa com rápida convergência, desde que a condição inicial seja adequada. A Bisseção, apesar de mais lenta, é um método simples e exige menos critérios para convergência, e o Método da Secante, uma variação do Método de Newton, é útil quando a derivada da função não está disponível.

Na resolução de sistemas lineares, o Método de Jacobi é uma técnica iterativa de fácil entendimento e aplicação, enquanto para sistemas não lineares, o Método de Newton se mostra uma ferramenta poderosa, utilizando a matriz Jacobiana para alcançar a convergência.

Este material oferece uma base para que você aplique esses conhecimentos em suas próprias pesquisas e projetos futuros. Esperamos que este conteúdo tenha proporcionado tanto a teoria quanto a prática necessárias para seu início na programação e na matemática computacional. Boa sorte em sua jornada!

Métodos Numéricos e Python no Ensino Médio

Edson Bertosa Lopes Santos
Luis Alberto D'Afonseca
Carlos Magno Martins Cosme

Centro Federal dealign Educação Tecnológica de Minas Gerais – CEFET-MG
1 de novembro de 2024

Arte da capa: Fotografia de Designdrunkard baixada de Pexels