O Grimório Proibido dos Jutsus Algébricos: Transmutando Variáveis e Conquistando Equações com Rust, Java e TypeScript

Prefácio: Desvendando o Grimório (Por Que Este Caminho, Jovem Adepto?)

Então, você tropeçou neste... *peculiar* tomo. A maioria corre gritando ao ver um integral, quanto mais um demônio de equação. Você é excepcionalmente corajoso, deliciosamente tolo, ou apenas muito, *muito* perdido. Perfeito. Você se encaixará perfeitamente.

Esqueça aqueles pergaminhos empoeirados e senseis monótonos que tornaram o 'x' mais assustador que o Orochimaru num dia de cabelo ruim. A álgebra não é uma maldição infligida a estudantes desavisados; é um *poder*, uma linguagem secreta para comandar a própria estrutura de... bem, problemas, de qualquer forma. Pense nisto não como um livro didático, mas como um pergaminho proibido, revelando os antigos jutsus da manipulação algébrica. Como disse o sábio Baroni, o humor pode ser o "Cavalo de Troia para o conhecimento" , e é exatamente isso que este grimório pretende ser.

Os alquimistas da antiguidade buscavam transformar chumbo em ouro [2]; nós, meu ávido aprendiz, transformaremos equações confusas em respostas elegantes. É basicamente o mesmo, só que com menos envenenamento por mercúrio. Esta é a "Grande Obra" da álgebra, onde a "pedra" da mente ignorante é transmutada no "ouro" da sabedoria.[2]

Apresentaremos as "três sagradas linguagens de encantamento" – Rust, Java e TypeScript – como as ferramentas para nossa "conjuração de feitiços". Cada linguagem tem seu próprio dialeto, suas próprias peculiaridades, como invocar um demônio sarcástico (o verificador de empréstimo do Rust), um golem verboso (o boilerplate do Java), ou uma kitsune metamorfa (a flexibilidade do TypeScript).

Por que álgebra, você pergunta, com um tom de quem foi forçado a assistir a uma maratona de episódios filler de Naruto? Porque, meu caro adepto, entender álgebra é como dominar os selos de mão básicos.[11] Sem eles, seus jutsus mais poderosos – seja em programação, ciência, ou mesmo para decifrar por que sua pizza extra grande parece menor que o diploma de matemática pura do seu amigo na hora de alimentar uma família de quatro [3] – serão, na melhor das hipóteses, um fiasco.

Alguém uma vez me perguntou o que eu faço. Considere dizer: "Eu comungo com entidades abstratas e as aprisiono com lógica." Mas "sou um matemático/programador" tem menos chances de me levar a um exorcismo. Na maior parte do tempo. Meu pai, por outro lado, afirma que "mata variáveis desgarradas e vende seus coeficientes no mercado negro". Ele é, na verdade, um contador, mas como explicar *isso* para os não iniciados?.[3]

Este grimório é seu primeiro passo no caminho para se tornar um Code-Kage, um Alquimista Algorítmico. Prepare seu chakra (ou café, muito café) e vamos começar.

Capítulo 1: O Chakra Alfanumérico – Entendendo Variáveis e Expressões

1.1 O Que nos Nove Infernos é uma Variável? (O 'x' Marca o Local... ou Não?)

Ah, 'x'. A letra que lançou mil sessões de terapia e assombrou mais pesadelos de estudantes do que um Bijuu com dor de dente. Mas o que *é* essa criatura elusiva? Em sua essência mais pura, uma variável é um símbolo, geralmente uma letra, que representa uma quantidade desconhecida ou uma quantidade que pode mudar.[4] Pense em uma variável como uma caixa de tesouro mística. Você ainda não sabe o que há dentro – pode ser um tesouro, pode ser um duende muito zangado – mas você sabe que ela *guarda* algo. No mundo da programação, essa caixa de tesouro ganha uma forma mais concreta: é um nome que usamos para nos referir a um local específico na memória do computador onde um valor é armazenado.[6]

No reino etéreo da matemática pura, 'x' é um espírito errante, um conceito abstrato. No dojo codificado do seu computador, um comando como let x_espiritual: i32 = 10; (em Rust) é esse espírito aprisionado em um pote bem específico – uma "foca de contenção" para um dado. A transição de uma variável matemática (um conceito) para uma variável de programação (um local de memória concreto com um tipo) é um salto conceitual crítico. É como selar uma entidade poderosa; o nome que você dá (o nome da variável) e o tipo de "pergaminho de selamento" (o tipo de dado) que você usa são cruciais.[6] Dar nomes significativos às variáveis é um ato de poder e controle, muito parecido com o conhecimento do "nome verdadeiro" em muitas tradições ocultas, que concede domínio sobre a entidade nomeada.[7]

Analogia Naruto: As variáveis são como o "chakra" fundamental que os ninjas moldam.[8] É o potencial bruto. Ou, as variáveis são o "alvo" de um jutsu – o desconhecido que você está tentando definir, capturar ou transformar. Antes que Naruto possa aplicar um Rasengan em alguém e mandá-lo para a próxima semana,

ele precisa identificar seu alvo. Esse alvo, em seu estado desconhecido, é nossa variável. O chakra, como as variáveis, é a energia base que pode ser moldada em várias formas (tipos de dados) para executar diferentes técnicas (operações).[8]

Analogia Ocultista: Uma variável é como um sigilo [7] representando uma força ou desejo ainda não manifestado. Ou, na alquimia [2], é um ingrediente cuja quantidade ou natureza precisa é inicialmente desconhecida, mas essencial para a Grande Obra. O 'x' em sua equação é a *prima materia*, a substância bruta e indefinida que você está prestes a transmutar.

1.2 Criando Suas Primeiras Encantações: Expressões Algébricas

Se as variáveis são seu chakra bruto, as expressões algébricas são os primeiros jutsus simples que você aprende – um soco, um chute, uma explosão elemental básica. Uma expressão algébrica é uma combinação de variáveis (nosso "chakra"), constantes ("pontos fixos no cosmos", ou os números imutáveis) e operações matemáticas ("forças elementares" ou "selos de mão básicos").[4]

Exemplos:

- x + 5: chakra x combinado com a essência fixa de 5 através do selo 'mais'.
- 2*y - z: o dobro do chakra y, então repele o chakra z.
- 3*a^2: chakra a triplicado, elevado à segunda potência de intensidade.

Analogia Naruto: Expressões como sequências simples de selos de mão.[10] Cada operação (+, -, *, /) é um selo fundamental. Combiná-los cria um jutsu básico (a expressão). Um selo 'Rato' para adição, um selo 'Tigre' para multiplicação... encadeie-os, e você tem um Katon: Gōkakyū no Jutsu algébrico, pronto para ser avaliado!

Analogia Ocultista: Expressões como receitas alquímicas simples ou componentes de um ritual maior.[2] Uma pitada de 'x' (mercúrio), uma pitada de 'mais' (enxofre) e uma pitada de '5' (sal) – eis uma fórmula transmutadora simples, uma expressão algébrica!

Dentro dessas expressões, encontramos:

- **Termos:** As runas individuais em seu feitiço (ex: em 2x + 3y - 5, os termos são 2x, 3y e -5).
- **Coeficientes:** O multiplicador de poder para o seu chakra (ex: em 2x, o 2 é o coeficiente).
- **Constantes:** As pedras fundamentais inflexíveis do seu ritual (ex: em 2x - 5, o -5

é a constante).

A **Ordem das Operações** (PEMDAS/BODMAS – Parênteses/Colchetes, Expoentes, Multiplicação e Divisão, Adição e Subtração) é a "sintaxe sagrada dos feitiços" ou a "sequência correta dos selos de mão". Se errar a ordem, seu jutsu pode sair pela culatra, invocando um sapo particularmente mal-humorado em vez de uma bola de fogo. O universo (e seu compilador) é muito particular sobre a sintaxe. Este "jutsu que sai pela culatra" é um mnemônico poderoso; a consequência vívida de um erro (um jutsu falho, um demônio invocado por engano) é mais provável de fixar a regra do que o acrônimo sozinho.

As expressões são os blocos de construção. Assim como selos de mão básicos são combinados para jutsus avançados, ou ingredientes alquímicos simples são fundamentais para a Pedra Filosofal, as expressões são a base para equações e funções mais complexas. Você não pode lançar um jutsu de Rank-S se atrapalhar os selos básicos.

1.3 Implementação em Código: Dando Forma ao Amorfo (Variáveis e Operações Básicas)

Agora, vamos traduzir esses conceitos místicos para as linguagens dos programadores modernos.

Tabela 1: Selos de Mão Elementares para Jutsus Aritméticos Básicos

Operador	Simbólico	Exemplo Rust	Exemplo Java	Exemplo TypeScript	Analogia Naruto/Oculta
+	a + b	let soma = a + b;	int soma = a + b;	let soma = a + b;	Selo da Fusão / Combinação de Essências
-	a - b	let diff = a - b;	int diff = a - b;	let diff = a - b;	Selo da Repulsão / Separação de Elementos
*	a * b	let prod = a *	int prod = a *	let prod = a *	Selo da Multiplicaçã

		b;	b;	b;	o (Clone das Sombras) / Potenciação
/	a / b	let quot = a / b; (cuidado com inteiros!)	double quot = (double)a / b;	let quot = a / b;	Selo da Divisão (Corte) / Refinamento da Substância
%	a % b	let rem = a % b;	int rem = a % b;	let rem = a % b;	Selo do Resíduo / Restos da Transmutação

Rust: O Sensei Exigente 13
Rust é como aquele sensei que exige perfeição em cada movimento. Ele quer saber a natureza exata do seu chakra (tipo de dado) desde o início.

Rust

```rust
// main.rs
fn main() {
    // Declarando variáveis (selando o chakra)
    let a_espiritual: i32 = 10; // Um inteiro de 32 bits, imutável por padrão
    let mut b_material: f64 = 5.5; // Um ponto flutuante de 64 bits, mutável

    println!("Chakra A: {}", a_espiritual);
    println!("Essência B: {}", b_material);

    // Operações básicas (Jutsus Elementares)
    let soma_elemental = a_espiritual + 7; // Rust infere o tipo de 7 como i32
    let produto_arcano = a_espiritual as f64 * b_material; // Conversão explícita de 'a' para f64
    let divisao_proibida = b_material / 2.0;
    let resto_mistico = a_espiritual % 3;
```

```
    println!("Soma Elemental (a + 7): {}", soma_elemental); // 17
    println!("Produto Arcano (a * b): {}", produto_arcano); // 55.0
    println!("Divisão Proibida (b / 2.0): {}", divisao_proibida); // 2.75
    println!("Resto Místico (a % 3): {}", resto_mistico); // 1

    b_material = b_material + 0.5; // Modificando uma variável mutável
    println!("Essência B Transmutada: {}", b_material); // 6.0
}
```

Atenção, neófito: Rust é rigoroso com tipos. Misturar i32 e f64 diretamente em operações como * requer uma conversão explícita (ex: a_espiritual as f64). Falhar nisso é como tentar um jutsu de fogo com chakra de água – resultados... úmidos e decepcionantes.

Java: O Dojo Tradicional 14
Java, o mestre do dojo tradicional, também insiste na declaração de tipos, mas sua sintaxe é um pouco mais... cerimonial.

Java

```
// AlquimiaBasica.java
public class AlquimiaBasica {
    public static void main(String args) {
        // Declarando variáveis (definindo os ingredientes)
        int a_ouro_filosofal = 10;
        double b_elixir = 5.5;

        System.out.println("Ouro A: " + a_ouro_filosofal);
        System.out.println("Elixir B: " + b_elixir);

        // Operações básicas (Processos Alquímicos)
        int soma_simples = a_ouro_filosofal + 7;
        double produto_complexo = (double)a_ouro_filosofal * b_elixir; // Conversão para
double
        double divisao_purificadora = b_elixir / 2.0;
        int resto_da_calcinacao = a_ouro_filosofal % 3;
```

```
        System.out.println("Soma Simples (a + 7): " + soma_simples); // 17
        System.out.println("Produto Complexo (a * b): " + produto_complexo); // 55.0
        System.out.println("Divisão Purificadora (b / 2.0): " + divisao_purificadora); // 2.75
        System.out.println("Resto da Calcinação (a % 3): " + resto_da_calcinacao); // 1

        b_elixir = b_elixir + 0.5;
        System.out.println("Elixir B Refinado: " + b_elixir); // 6.0
    }
}
```

Observação do Alquimista: Assim como em Rust, a conversão de tipo é importante em Java ao misturar inteiros e doubles para evitar perda de precisão ou comportamento inesperado.

TypeScript: O Ninja Metamorfo 15
TypeScript, o ninja adaptável, oferece flexibilidade com seus tipos. Você pode ser explícito, ou deixá-lo inferir, mas cuidado – grande flexibilidade vem com grande responsabilidade (de não criar um monstro de bugs).

TypeScript

```
// scriptArcano.ts
// Declarando variáveis (invocando espíritos numéricos)
let a_runa: number = 10;
let b_sigilo: number = 5.5;

console.log(`Runa A: ${a_runa}`);
console.log(`Sigilo B: ${b_sigilo}`);

// Operações básicas (Pequenos Feitiços)
let soma_astral: number = a_runa + 7;
let produto_etereo: number = a_runa * b_sigilo; // TypeScript lida com a mistura de number
let divisao_cosmica: number = b_sigilo / 2.0;
let resto_dimensional: number = a_runa % 3;

console.log(`Soma Astral (a + 7): ${soma_astral}`); // 17
console.log(`Produto Etéreo (a * b): ${produto_etereo}`); // 55
console.log(`Divisão Cósmica (b / 2.0): ${divisao_cosmica}`); // 2.75
```

```
console.log(`Resto Dimensional (a % 3): ${resto_dimensional}`); // 1

b_sigilo = b_sigilo + 0.5;
console.log(`Sigilo B Aprimorado: ${b_sigilo}`); // 6.0

// Para executar:
// 1. Certifique-se de ter Node.js e npm instalados.
// 2. Instale o TypeScript: npm install -g typescript
// 3. Compile: tsc scriptArcano.ts
// 4. Execute: node scriptArcano.js
```

Sussurro do Grimório: TypeScript é construído sobre JavaScript. Embora o TypeScript adicione verificação de tipo em tempo de compilação, o JavaScript subjacente é dinamicamente tipado, o que às vezes pode levar a comportamentos "interessantes" (leia-se: "arrancar os cabelos") se você não for cuidadoso, especialmente com operações entre tipos diferentes, como a concatenação de string com número ao usar +.[16]

Com esses fundamentos, você deu os primeiros passos para moldar o chakra alfanumérico. Agora, prepare-se para encantamentos mais complexos: as equações!

Capítulo 2: O Jutsu da Igualdade – Resolvendo Equações Lineares

2.1 A Transmutação de Dois Elementos: Desvendando ax + b = 0

Chegamos à nossa primeira forma de encantamento sério, a equação linear: ax+b=0. Não se deixe enganar por sua aparente simplicidade; dominar isso é como aperfeiçoar o controle básico de chakra – essencial para todo jutsu mais complexo que virá. Nosso objetivo aqui é "isolar o espírito elusivo 'x'", ou, em termos menos dramáticos, encontrar o valor de x que torna a afirmação verdadeira.[17]

Pense nisso como um ritual de equilíbrio alquímico.[2] A equação ax+b=0 afirma que a combinação das forças 'a multiplicada por x' e 'b' resulta em um estado de nulidade, de equilíbrio perfeito. Para encontrar 'x', devemos reverter os processos, mantendo o equilíbrio em cada etapa. É como tentar descobrir o ingrediente secreto em uma poção que resultou em... nada.

O processo é uma dança de opostos:

1. Neutralizar 'b' (O Selo da Reversão): Se 'b' foi adicionado, nós o removemos (subtraímos) de ambos os lados da equação para manter o equilíbrio cósmico (e

matemático).
ax+b−b=0−b
ax=−b
"Para cada ação, uma reação igual e oposta," sussurram os antigos grimórios da física e, aparentemente, da álgebra.

2. Liberar 'x' de 'a' (O Selo do Desprendimento): Se 'x' está sendo multiplicado por 'a', nós dividimos ambos os lados por 'a' (assumindo que 'a' não seja zero – dividir por zero é um kinjutsu que nem este grimório ousa ensinar, pois rasga o próprio tecido da realidade matemática!).
aax=a−b
x=a−b

E voilà! O espírito 'x' foi isolado, sua verdadeira forma revelada.

Casos Especiais (Quando os Elementos se Comportam de Maneira Estranha):

- **Se a = 0 e b = 0:** A equação se torna 0x+0=0, ou 0=0. Isso é verdade para *qualquer* valor de x. É como um jutsu que funciona em todos, sempre. Temos infinitas soluções. No mundo dos programadores, isso pode ser uma receita para um loop infinito se não for tratado com cuidado.
- **Se a = 0 e b ≠ 0:** A equação se torna 0x+b=0, ou b=0. Mas dissemos que b僾=0! Isso é uma contradição, um paradoxo que faria a lógica de um Nara entrar em colapso. Não há solução. É um jutsu que sempre falha.

Dominar esses casos é crucial, pois eles representam as bordas do nosso mapa alquímico, os limites do nosso jutsu.

2.2 Conjurando Soluções: Código para Equações Lineares

Agora, vamos inscrever esses rituais nas linguagens sagradas. Criaremos funções (ou métodos, se você for um purista de Java) que pegam os coeficientes a e b e nos devolvem a alma de x, ou um aviso se a transmutação for impossível ou levar ao caos (infinitas soluções).

Rust: Precisão e Segurança do Ferrugem 18
Rust, com sua obsessão por segurança e prevenção de desastres (como dividir por zero sem querer), nos força a sermos explícitos sobre o que pode dar errado. Usaremos Option<f64> para indicar que uma solução pode ou não existir.

Rust

```rust
// linear_solver_rust/src/main.rs

// Função para resolver ax + b = 0
// Retorna Option<f64> para lidar com casos sem solução única.
fn solve_linear_equation(a: f64, b: f64) -> Result<f64, String> {
    println!("\nInvocando o Jutsu de Resolução Linear em Rust para: {}x + {} = 0", a, b);
    if a.abs() < 1e-9 { // Quase zero, para lidar com imprecisões de ponto flutuante
        if b.abs() < 1e-9 { // Quase zero
            // 0x + 0 = 0  => Infinitas soluções
            // Para este grimório, vamos dizer que é um tipo especial de "sucesso"
            // mas que requer interpretação do Mestre Alquimista (você!).
            // Em um cenário real, você poderia retornar um enum mais descritivo.
            println!("O Oráculo de Rust sussurra: 'Infinitas essências satisfazem este equilíbrio...'");
            return Err("Infinitas soluções (0x + 0 = 0)".to_string());
        } else {
            // 0x + b = 0 (com b!= 0) => Nenhuma solução
            println!("O Oráculo de Rust adverte: 'Este caminho leva ao vazio... Nenhuma solução!'");
            return Err("Nenhuma solução (0x + b = 0, b!= 0)".to_string());
        }
    }
    // ax + b = 0  => x = -b / a
    let x = -b / a;
    println!("Transmutação bem-sucedida em Rust! x = {}", x);
    Ok(x)
}

fn main() {
    // Teste 1: Equação normal
    match solve_linear_equation(2.0, 4.0) { // 2x + 4 = 0 => x = -2
        Ok(x) => println!("Resultado da Transmutação: x = {}", x),
        Err(e) => println!("Falha na Transmutação: {}", e),
    }

    // Teste 2: 'a' é zero, 'b' não é zero (sem solução)
    match solve_linear_equation(0.0, 5.0) { // 0x + 5 = 0 => Sem solução
        Ok(x) => println!("Resultado da Transmutação: x = {}", x),
        Err(e) => println!("Falha na Transmutação: {}", e),
    }
```

```
    // Teste 3: 'a' é zero, 'b' é zero (infinitas soluções)
    match solve_linear_equation(0.0, 0.0) { // 0x + 0 = 0 => Infinitas soluções
        Ok(x) => println!("Resultado da Transmutação: x = {}", x), // Não será alcançado neste setup de Err
        Err(e) => println!("Falha na Transmutação: {}", e),
    }

    // Teste 4: Coeficientes negativos
    match solve_linear_equation(-3.0, 9.0) { // -3x + 9 = 0 => x = 3
        Ok(x) => println!("Resultado da Transmutação: x = {}", x),
        Err(e) => println!("Falha na Transmutação: {}", e),
    }
}
```

Encantamento do Ferrugem: Note o uso de Result<f64, String>. Em Rust, erros são valores, e Result é a forma idiomática de lidar com operações que podem falhar. Ok(valor) significa sucesso, Err(mensagem) significa falha. Isso evita os temidos NullPointerExceptions que assombram outros reinos. Comparar floats com zero diretamente (a == 0.0) pode ser arriscado devido a como os computadores armazenam números de ponto flutuante; usar uma pequena tolerância (a.abs() < 1e-9) é uma prática mais robusta, um pequeno "selo de proteção" contra os caprichos dos bits.

Java: A Tradição Orientada a Objetos 17
Java, com sua estrutura de classes, nos encoraja a encapsular a lógica. Poderíamos criar uma classe LinearEquationSolver, mas para simplicidade, usaremos um método estático. Java não tem um Option nativo como Rust, então frequentemente se usa null para indicar ausência de valor, ou se lança uma exceção.

Java

```
// LinearEquationSolver.java
public class LinearEquationSolver {

    // Método para resolver ax + b = 0
    // Retorna Double (objeto) para permitir 'null' como ausência de solução única.
    // Ou poderia lançar uma exceção para casos especiais.
    public static String solveLinear(double a, double b) {
```

```
        System.out.println("\nInvocando o Ritual de Resolução Linear em Java para: " + a + "x + " + b + " = 0");
        // Usando uma pequena tolerância para comparação com zero para doubles
        double epsilon = 1e-9;

        if (Math.abs(a) < epsilon) {
            if (Math.abs(b) < epsilon) {
                // 0x + 0 = 0 => Infinitas soluções
                System.out.println("O Oráculo de Java proclama: 'Inúmeras almas numéricas atendem a este chamado!'");
                return "Infinitas soluções";
            } else {
                // 0x + b = 0 (com b!= 0) => Nenhuma solução
                System.out.println("O Oráculo de Java lamenta: 'Os caminhos se fecham. Nenhuma solução existe.'");
                return "Nenhuma solução";
            }
        }
        // ax + b = 0  => x = -b / a
        double x = -b / a;
        System.out.println("Alquimia concluída em Java! x = " + x);
        return "x = " + x;
    }

    public static void main(String args) {
        // Teste 1: Equação normal
        System.out.println("Resultado da Alquimia: " + solveLinear(2.0, 4.0)); // 2x + 4 = 0 => x = -2

        // Teste 2: 'a' é zero, 'b' não é zero (sem solução)
        System.out.println("Resultado da Alquimia: " + solveLinear(0.0, 5.0)); // 0x + 5 = 0

        // Teste 3: 'a' é zero, 'b' é zero (infinitas soluções)
        System.out.println("Resultado da Alquimia: " + solveLinear(0.0, 0.0)); // 0x + 0 = 0

        // Teste 4: Coeficientes negativos
        System.out.println("Resultado da Alquimia: " + solveLinear(-3.0, 9.0)); // -3x + 9 = 0 => x = 3
    }
}
```

Pergaminho de Java: Usar Double (a classe wrapper) em vez de double (o tipo primitivo) permitiria retornar null. Aqui, optamos por retornar uma String para descrever o resultado, o que pode ser mais informativo para o usuário do grimório. Lançar exceções personalizadas (ex: NoSolutionException, InfiniteSolutionsException) seria uma abordagem mais robusta em software de produção.

TypeScript: A Flexibilidade Dinâmica (com Tipos!) 19
TypeScript nos dá a flexibilidade do JavaScript com a segurança dos tipos. Podemos definir que nossa função retorna um number, null (se não houver solução), ou uma string (para infinitas soluções).

TypeScript

```
// linearSolver.ts

// Função para resolver ax + b = 0
// Retorna number, null (sem solução) ou string (infinitas soluções)
function solveLinearEquationTS(a: number, b: number): number | null | string {
    console.log(`\nExecutando o Jutsu de Resolução Linear em TypeScript para: ${a}x + ${b} = 0`);
    const epsilon = 1e-9; // Pequena tolerância para comparação com zero

    if (Math.abs(a) < epsilon) {
        if (Math.abs(b) < epsilon) {
            // 0x + 0 = 0 => Infinitas soluções
            console.log("O Kage do TypeScript declara: 'Este jutsu afeta todas as coisas!'");
            return "Infinitas soluções";
        } else {
            // 0x + b = 0 (com b!= 0) => Nenhuma solução
            console.log("O Kage do TypeScript avisa: 'Este jutsu não tem alvo!'");
            return null;
        }
    }
    // ax + b = 0  => x = -b / a
    const x = -b / a;
    console.log(`Jutsu executado com sucesso em TypeScript! x = ${x}`);
    return x;
}
```

```
// Testes
// Teste 1: Equação normal
console.log(`Resultado do Jutsu: ${solveLinearEquationTS(2.0, 4.0)}`); // 2x + 4 = 0 => x = -2

// Teste 2: 'a' é zero, 'b' não é zero (sem solução)
console.log(`Resultado do Jutsu: ${solveLinearEquationTS(0.0, 5.0)}`); // 0x + 5 = 0

// Teste 3: 'a' é zero, 'b' é zero (infinitas soluções)
console.log(`Resultado do Jutsu: ${solveLinearEquationTS(0.0, 0.0)}`); // 0x + 0 = 0

// Teste 4: Coeficientes negativos
console.log(`Resultado do Jutsu: ${solveLinearEquationTS(-3.0, 9.0)}`); // -3x + 9 = 0 => x = 3

/*
Para executar:
1. Salve como linearSolver.ts
2. Instale o TypeScript se ainda não o fez: npm install -g typescript
3. Compile: tsc linearSolver.ts
4. Execute com Node.js: node linearSolver.js
*/
```

Manuscrito TypeScript: O tipo de retorno number | null | string é um tipo de união, uma característica poderosa do TypeScript que permite que uma função retorne diferentes tipos de valores dependendo do resultado da computação.

Com estes encantamentos, você agora pode resolver a mais fundamental das equações. Mas o caminho do saber é longo, e demônios mais complexos aguardam...

Capítulo 3: A Dança das Três Entidades – Equações Quadráticas e o Reino Imaginário

3.1 A Trindade dos Coeficientes: Enfrentando $ax^2+bx+c=0$

Avançamos para um desafio mais... interessante. A equação quadrática, $ax^2+bx+c=0$, onde 'a', 'b' e 'c' são nossos coeficientes (e 'a' não ousa ser zero, ou voltamos à simplicidade linear, o que seria terrivelmente anticlimático agora). Esta forma é como um jutsu de três selos de mão, mais complexo, mas com potencial para resultados muito mais variados. Resolver isso é como decifrar um enigma mais profundo, um que pode ter duas respostas, uma resposta disfarçada de duas, ou respostas que... bem,

não são deste mundo.

A chave para desvendar este mistério é a lendária Fórmula Quadrática, também conhecida como Fórmula de Bhaskara em alguns cantos do multiverso:
$x = 2a - b \pm b^2 - 4ac$
Essa fórmula é o nosso mapa do tesouro. Mas antes de cavar, precisamos consultar o **Discriminante**, o termo $b^2 - 4ac$, que denotaremos pelo símbolo arcano Δ. O Discriminante é como o Olho do Oráculo; seu sinal revela a natureza das raízes antes mesmo de as calcularmos completamente.[24]

Tabela 2: O Veredito do Oráculo – Interpretando os Sinais do Discriminante

Valor do Discriminante (Δ)	Natureza das Raízes	Analogia Naruto	Analogia Ocultista
$\Delta > 0$	Duas raízes reais e distintas	"Dois caminhos revelados pelo Byakugan!"	"A transmutação produz dois resultados distintos e tangíveis."
$\Delta = 0$	Uma raiz real (dupla, ou duas raízes reais e iguais)	"Um único caminho, mas com uma forte assinatura de chakra (raiz dupla)."	"A Pedra Filosofal! Uma única essência perfeita (mas com poder dobrado)."
$\Delta < 0$	Duas raízes complexas conjugadas (sem raízes reais)	"Raízes ocultas no Genjutsu do Infinito, invisíveis ao olho comum."	"Raízes invocadas do Reino Espiritual/Astral, não manifestas no plano material."

Se $\Delta \geq 0$, as raízes são "reais", pertencentes ao nosso plano de existência numérica familiar. Mas se $\Delta < 0$... ah, aí é que a diversão (ou o terror, dependendo da sua afinidade com o bizarro) começa. Entramos no reino dos **números complexos**.

3.2 Espiando o Reino Imaginário: Introdução aos Números Complexos

Quando o Discriminante ousa ser negativo, a fórmula quadrática nos pede para tirar a raiz quadrada de um número negativo. Os matemáticos de outrora teriam fugido gritando "Bruxaria!" ou, pior, "Isso não está no currículo!".[27] Mas nós, adeptos deste grimório, somos mais ousados.

Introduzimos a unidade imaginária, **'i'**, o espírito rebelde da matemática, definido como $i=-1$. Isso significa que $i2=-1$.[4] Com 'i', podemos agora dar sentido a raízes quadradas de números negativos. Por exemplo, $-9=9\times-1=9\times-1=3i$.

Um número complexo é uma entidade que possui uma parte real e uma parte imaginária, geralmente escrita na forma $a+bi$, onde 'a' é a parte real e 'b' é a parte imaginária.[28]

- Se $b=0$, temos um número puramente real (ex: $5+0i=5$).
- Se $a=0$ e $b優=0$, temos um número puramente imaginário (ex: $0+3i=3i$).

Analogia Naruto: Pense nos números reais como o mundo físico que os ninjas habitam. Os números complexos são como uma dimensão paralela, como o Kamui de Obito, acessível através de um "jutsu" específico (a introdução de 'i'). As raízes complexas de uma equação são soluções que existem nessa dimensão alternativa.

Analogia Ocultista: Números reais são o plano material. Números complexos são o plano astral ou etéreo.[30] Eles não são "menos reais", apenas existem em uma vibração diferente, acessível através de um entendimento mais profundo das leis cósmicas (matemáticas). A transmutação que leva a raízes complexas é aquela que transcende o puramente físico.

Operações com Números Complexos (Jutsus de Manipulação Dimensional):
Sejam $z1=a+bi$ e $z2=c+di$.

- Adição/Subtração (Fusão/Separação Dimensional):
  $z1\pm z2=(a\pm c)+(b\pm d)i$
  (Some/subtraia as partes reais e as partes imaginárias separadamente)
- Multiplicação (Entrelaçamento Dimensional):
  $z1\times z2=(a+bi)(c+di)=ac+adi+bci+bdi2=ac+adi+bci-bd=(ac-bd)+(ad+bc)i$
  (Multiplique como binômios, lembrando que $i2=-1$)
- Divisão (Desembaraçamento Dimensional - mais complexo, geralmente envolve o conjugado):
  O conjugado de um número complexo $z=a+bi$ é $z¯=a-bi$. Multiplicar um número complexo por seu conjugado resulta em um número real:
  $zz¯=(a+bi)(a-bi)=a2-(bi)2=a2-(-b2)=a2+b2$.
  Para dividir $z2z1$, multiplicamos o numerador e o denominador pelo conjugado do denominador:
  $z2z1=c+dia+bi\times c-dic-di=c2+d2(a+bi)(c-di)=c2+d2(ac+bd)+(bc-ad)i$

Agora que podemos navegar pelo reino imaginário, estamos prontos para enfrentar as

equações quadráticas que geram raízes complexas.

3.3 Código para Números Complexos e Solucionadores Quadráticos

Vamos implementar nossos próprios tipos ComplexNumber (ou usar bibliotecas existentes) e, em seguida, as funções para resolver equações quadráticas, lidando com todos os tipos de raízes.

Rust: Números Complexos com num_complex e Feitiçaria Quadrática 28
Rust tem uma excelente crate chamada num-complex para lidar com números complexos. Primeiro, adicione ao seu Cargo.toml:

```
Ini, TOML
```

```ini
[dependencies]
num-complex = "0.4" # Verifique a versão mais recente
```

Agora, o código para a estrutura ComplexNumber (se quiséssemos fazer manualmente, mas usaremos num_complex) e o solucionador:

```
Rust
```

```rust
// complex_solver_rust/src/main.rs
use num_complex::Complex; // O tipo Complex da crate num_complex

// Estrutura para representar as raízes de uma equação quadrática
#
struct QuadraticRoots {
    root1: Complex<f64>,
    root2: Option<Complex<f64>>, // Option para o caso de raiz única (discriminante == 0)
    message: String,
}

// Função para calcular o discriminante
fn calculate_discriminant(a: f64, b: f64, c: f64) -> f64 {
    b * b - 4.0 * a * c
```

```rust
}

// Função para resolver ax^2 + bx + c = 0
fn solve_quadratic_equation_rust(a: f64, b: f64, c: f64) -> Result<QuadraticRoots, String> {
    println!("\nConjurando o Jutsu Quadrático em Rust para: {}x^2 + {}x + {} = 0", a, b, c);

    if a.abs() < 1e-9 { // Coeficiente 'a' muito próximo de zero
        // Isso degenera para uma equação linear bx + c = 0
        if b.abs() < 1e-9 { // Coeficiente 'b' também próximo de zero
            if c.abs() < 1e-9 { // 0x^2 + 0x + 0 = 0
                return Err("Infinitas soluções (equação degenerada para 0 = 0)".to_string());
            } else { // 0x^2 + 0x + c = 0 (c!= 0)
                return Err("Nenhuma solução (equação degenerada para c = 0, c!= 0)".to_string());
            }
        }
        // Equação linear: bx + c = 0 => x = -c / b
        let linear_root = -c / b;
        println!("Equação degenerou para linear. Raiz linear: {}", linear_root);
        return Ok(QuadraticRoots {
            root1: Complex::new(linear_root, 0.0),
            root2: None,
            message: "Equação degenerada para linear, uma raiz real encontrada.".to_string(),
        });
    }

    let discriminant = calculate_discriminant(a, b, c);
    println!("O Oráculo do Discriminante revela: Δ = {}", discriminant);

    let roots_result: QuadraticRoots;

    if discriminant > 1e-9 { // Discriminante positivo (duas raízes reais distintas)
        let sqrt_discriminant = discriminant.sqrt();
        let r1 = (-b + sqrt_discriminant) / (2.0 * a);
        let r2 = (-b - sqrt_discriminant) / (2.0 * a);
        println!("Dois caminhos reais se abrem: x1 = {}, x2 = {}", r1, r2);
        roots_result = QuadraticRoots {
            root1: Complex::new(r1, 0.0),
            root2: Some(Complex::new(r2, 0.0)),
            message: "Duas raízes reais e distintas.".to_string(),
```

```
        };
    } else if discriminant.abs() < 1e-9 { // Discriminante zero (uma raiz real dupla)
        let r = -b / (2.0 * a);
        println!("Um único caminho real, mas poderoso: x = {}", r);
        roots_result = QuadraticRoots {
            root1: Complex::new(r, 0.0),
            root2: None, // Ou Some(Complex::new(r, 0.0)) se preferir listar duas vezes
            message: "Uma raiz real (dupla).".to_string(),
        };
    } else { // Discriminante negativo (duas raízes complexas conjugadas)
        let real_part = -b / (2.0 * a);
        let imag_part = (-discriminant).sqrt() / (2.0 * a);
        println!("Portais para o Reino Imaginário se abrem: x1 = {} + {}i, x2 = {} - {}i", real_part,
imag_part, real_part, imag_part);
        roots_result = QuadraticRoots {
            root1: Complex::new(real_part, imag_part),
            root2: Some(Complex::new(real_part, -imag_part)),
            message: "Duas raízes complexas conjugadas.".to_string(),
        };
    }
    Ok(roots_result)
}

fn main() {
    // Teste 1: Duas raízes reais (x^2 - 3x + 2 = 0 => (x-1)(x-2)=0 => x=1, x=2)
    match solve_quadratic_equation_rust(1.0, -3.0, 2.0) {
        Ok(roots) => println!("Resultado: {:?}\n", roots),
        Err(e) => println!("Erro: {}\n", e),
    }

    // Teste 2: Uma raiz real dupla (x^2 - 2x + 1 = 0 => (x-1)^2=0 => x=1)
    match solve_quadratic_equation_rust(1.0, -2.0, 1.0) {
        Ok(roots) => println!("Resultado: {:?}\n", roots),
        Err(e) => println!("Erro: {}\n", e),
    }

    // Teste 3: Raízes complexas (x^2 + x + 1 = 0)
    // x = (-1 ± sqrt(1-4))/2 = (-1 ± sqrt(-3))/2 = (-1 ± i*sqrt(3))/2
    // x = -0.5 ± 0.866i
```

```
    match solve_quadratic_equation_rust(1.0, 1.0, 1.0) {
        Ok(roots) => println!("Resultado: {:?}\n", roots),
        Err(e) => println!("Erro: {}\n", e),
    }

    // Teste 4: 'a' é zero (degenerada para linear: 2x + 4 = 0 => x = -2)
    match solve_quadratic_equation_rust(0.0, 2.0, 4.0) {
        Ok(roots) => println!("Resultado: {:?}\n", roots),
        Err(e) => println!("Erro: {}\n", e),
    }

    // Teste 5: 'a' e 'b' são zero, 'c' não é zero (0x^2 + 0x + 5 = 0 => Nenhuma solução)
    match solve_quadratic_equation_rust(0.0, 0.0, 5.0) {
        Ok(roots) => println!("Resultado: {:?}\n", roots),
        Err(e) => println!("Erro: {}\n", e),
    }
}
```

Magia de Rust: A crate num_complex::Complex<f64> simplifica muito o trabalho. A função retorna um Result<QuadraticRoots, String> para um tratamento de erros mais robusto, especialmente para os casos degenerados onde a=0. A estrutura QuadraticRoots ajuda a retornar as raízes de forma clara. Usamos uma pequena tolerância 1e-9 para comparar números de ponto flutuante com zero, uma prática comum para evitar problemas de precisão.

Java: Criando a Classe ComplexNumber e o Solucionador Quadrático 29
Em Java, criaremos nossa própria classe ComplexNumber para representar números complexos.

Java

```
// ComplexNumber.java
import java.util.Objects;

public class ComplexNumber {
    private final double real;
    private final double imag;
```

```
    public ComplexNumber(double real, double imag) {
        this.real = real;
        this.imag = imag;
    }

    public double getReal() {
        return real;
    }

    public double getImag() {
        return imag;
    }

    public ComplexNumber add(ComplexNumber other) {
        return new ComplexNumber(this.real + other.real, this.imag + other.imag);
    }

    public ComplexNumber subtract(ComplexNumber other) {
        return new ComplexNumber(this.real - other.real, this.imag - other.imag);
    }

    public ComplexNumber multiply(ComplexNumber other) {
        double result_real = this.real * other.real - this.imag * other.imag;
        double result_imag = this.real * other.imag + this.imag * other.real;
        return new ComplexNumber(result_real, result_imag);
    }

    public ComplexNumber divide(ComplexNumber other) {
        double denominator = other.real * other.real + other.imag * other.imag;
        if (Math.abs(denominator) < 1e-9) { // Evitar divisão por zero complexo
            // Lidar com erro, talvez lançar uma ArithmeticException
            System.err.println("Erro de transmutação: Divisão por número complexo zero!");
            return new ComplexNumber(Double.NaN, Double.NaN); // Ou outra representação de
erro
        }
        double result_real = (this.real * other.real + this.imag * other.imag) / denominator;
        double result_imag = (this.imag * other.real - this.real * other.imag) / denominator;
        return new ComplexNumber(result_real, result_imag);
```

```
    }

    public static ComplexNumber fromReal(double realPart) {
        return new ComplexNumber(realPart, 0);
    }

    @Override
    public String toString() {
        if (Math.abs(imag) < 1e-9) { // Considerar como puramente real
            return String.format("%.2f", real);
        }
        if (Math.abs(real) < 1e-9 && Math.abs(imag) >= 1e-9) { // Considerar como puramente
imaginário
            return String.format("%.2fi", imag);
        }
        return String.format("%.2f %c %.2fi", real, (imag < 0? '-' : '+'), Math.abs(imag));
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null |
| getClass()!= o.getClass()) return false;
        ComplexNumber that = (ComplexNumber) o;
        // Comparar com uma tolerância para doubles
        double epsilon = 1e-9;
        return Math.abs(that.real - real) < epsilon &&
               Math.abs(that.imag - imag) < epsilon;
    }

    @Override
    public int hashCode() {
        return Objects.hash(real, imag); // Cuidado com hashCode para doubles se a igualdade usa
tolerância
    }
}

// QuadraticSolver.java
import java.util.ArrayList;
```

```
import java.util.List;

public class QuadraticSolver {

    private static final double EPSILON = 1e-9;

    public static class QuadraticSolution {
        public List<ComplexNumber> roots;
        public String message;

        public QuadraticSolution(List<ComplexNumber> roots, String message) {
            this.roots = roots;
            this.message = message;
        }

        @Override
        public String toString() {
            return "Mensagem: " + message + ", Raízes: " + roots;
        }
    }

    public static double calculateDiscriminant(double a, double b, double c) {
        return b * b - 4 * a * c;
    }

    public static QuadraticSolution solve(double a, double b, double c) {
        System.out.println("\nIniciando o Ritual Quadrático em Java para: " + a + "x^2 + " + b + "x + "
+ c + " = 0");
        List<ComplexNumber> roots = new ArrayList<>();
        String message;

        if (Math.abs(a) < EPSILON) {
            if (Math.abs(b) < EPSILON) {
                message = (Math.abs(c) < EPSILON)? "Infinitas soluções (0 = 0)" : "Nenhuma
solução (c = 0, c!= 0)";
                System.out.println("O Oráculo de Java decreta: " + message);
                return new QuadraticSolution(roots, message);
            }
            // Equação linear: bx + c = 0
```

```
            double linearRoot = -c / b;
            roots.add(ComplexNumber.fromReal(linearRoot));
            message = "Equação degenerada para linear. Raiz: " + linearRoot;
            System.out.println(message);
            return new QuadraticSolution(roots, message);
        }

        double discriminant = calculateDiscriminant(a, b, c);
        System.out.println("O Olho do Oráculo (Java) revela o Discriminante: Δ = " + discriminant);

        if (discriminant > EPSILON) {
            double sqrtDiscriminant = Math.sqrt(discriminant);
            double r1 = (-b + sqrtDiscriminant) / (2 * a);
            double r2 = (-b - sqrtDiscriminant) / (2 * a);
            roots.add(ComplexNumber.fromReal(r1));
            roots.add(ComplexNumber.fromReal(r2));
            message = "Duas raízes reais e distintas encontradas.";
            System.out.println(message + " x1 = " + r1 + ", x2 = " + r2);
        } else if (Math.abs(discriminant) < EPSILON) {
            double r = -b / (2 * a);
            roots.add(ComplexNumber.fromReal(r));
            message = "Uma raiz real (dupla) encontrada.";
            System.out.println(message + " x = " + r);
        } else {
            double realPart = -b / (2 * a);
            double imagPart = Math.sqrt(-discriminant) / (2 * a);
            roots.add(new ComplexNumber(realPart, imagPart));
            roots.add(new ComplexNumber(realPart, -imagPart));
            message = "Duas raízes complexas conjugadas materializadas.";
            System.out.println(message + " x1 = " + roots.get(0) + ", x2 = " + roots.get(1));
        }
        return new QuadraticSolution(roots, message);
    }

    public static void main(String args) {
        // Teste 1: Duas raízes reais
        System.out.println(solve(1.0, -3.0, 2.0));

        // Teste 2: Uma raiz real dupla
```

```
        System.out.println(solve(1.0, -2.0, 1.0));

        // Teste 3: Raízes complexas
        System.out.println(solve(1.0, 1.0, 1.0));

        // Teste 4: 'a' é zero
        System.out.println(solve(0.0, 2.0, 4.0));

        // Teste 5: 'a' e 'b' são zero, 'c' não é zero
        System.out.println(solve(0.0, 0.0, 5.0));
    }
}
```

Enciclopédia Java: A classe ComplexNumber encapsula os dados (partes real e imaginária) e as operações. O método solve na classe QuadraticSolver usa ComplexNumber para retornar as raízes. O uso de List<ComplexNumber> permite flexibilidade no número de raízes retornadas (embora para quadráticas seja sempre 0, 1 ou 2 distintas).

TypeScript: Classe ComplexNumber e Solucionador Quadrático Flexível 34
TypeScript nos permite definir uma classe ComplexNumber e uma função que pode retornar um array dessas instâncias.

TypeScript

```
// complexSolver.ts

class ComplexNumberTS {
    constructor(public real: number, public imag: number) {}

    add(other: ComplexNumberTS): ComplexNumberTS {
        return new ComplexNumberTS(this.real + other.real, this.imag + other.imag);
    }

    subtract(other: ComplexNumberTS): ComplexNumberTS {
        return new ComplexNumberTS(this.real - other.real, this.imag - other.imag);
    }
```

```
    multiply(other: ComplexNumberTS): ComplexNumberTS {
        const result_real = this.real * other.real - this.imag * other.imag;
        const result_imag = this.real * other.imag + this.imag * other.real;
        return new ComplexNumberTS(result_real, result_imag);
    }

    divide(other: ComplexNumberTS): ComplexNumberTS | string {
        const denominator = other.real * other.real + other.imag * other.imag;
        if (Math.abs(denominator) < 1e-9) {
            return "Erro de Invocação: Divisão por zero complexo!";
        }
        const result_real = (this.real * other.real + this.imag * other.imag) / denominator;
        const result_imag = (this.imag * other.real - this.real * other.imag) / denominator;
        return new ComplexNumberTS(result_real, result_imag);
    }

    toString(): string {
        if (Math.abs(this.imag) < 1e-9) {
            return `${this.real.toFixed(2)}`;
        }
        if (Math.abs(this.real) < 1e-9 && Math.abs(this.imag) >= 1e-9) {
            return `${this.imag.toFixed(2)}i`;
        }
        const sign = this.imag < 0? '-' : '+';
        return `${this.real.toFixed(2)} ${sign} ${Math.abs(this.imag).toFixed(2)}i`;
    }
}

interface QuadraticSolutionTS {
    roots: ComplexNumberTS;
    message: string;
}

function calculateDiscriminantTS(a: number, b: number, c: number): number {
    return b * b - 4 * a * c;
}

function solveQuadraticEquationTypeScript(a: number, b: number, c: number): QuadraticSolutionTS {
```

```
    console.log(`\nLançando o Jutsu Quadrático em TypeScript para: ${a}x^2 + ${b}x + ${c} = 0`);
    const roots: ComplexNumberTS =;
    let message: string;
    const epsilon = 1e-9;

    if (Math.abs(a) < epsilon) {
        if (Math.abs(b) < epsilon) {
            message = (Math.abs(c) < epsilon)? "Infinitas soluções (selo quebrado, 0 = 0)" :
"Nenhuma solução (paradoxo temporal, c = 0 onde c!= 0)";
            console.log("O Conselho dos Kages (TypeScript) delibera: " + message);
            return { roots, message };
        }
        const linearRoot = -c / b;
        roots.push(new ComplexNumberTS(linearRoot, 0));
        message = `Equação reduzida a linear. Raiz única: ${linearRoot}`;
        console.log(message);
        return { roots, message };
    }

    const discriminant = calculateDiscriminantTS(a, b, c);
    console.log(`O Sharingan do Discriminante (TS) prevê: Δ = ${discriminant}`);

    if (discriminant > epsilon) {
        const sqrtDiscriminant = Math.sqrt(discriminant);
        const r1 = (-b + sqrtDiscriminant) / (2 * a);
        const r2 = (-b - sqrtDiscriminant) / (2 * a);
        roots.push(new ComplexNumberTS(r1, 0));
        roots.push(new ComplexNumberTS(r2, 0));
        message = "Duas realidades (raízes) distintas se manifestam.";
        console.log(`${message} x1 = ${r1}, x2 = ${r2}`);
    } else if (Math.abs(discriminant) < epsilon) {
        const r = -b / (2 * a);
        roots.push(new ComplexNumberTS(r, 0));
        message = "Uma única realidade (raiz dupla) se solidifica.";
        console.log(`${message} x = ${r}`);
    } else {
        const realPart = -b / (2 * a);
        const imagPart = Math.sqrt(-discriminant) / (2 * a);
        roots.push(new ComplexNumberTS(realPart, imagPart));
```

```
        roots.push(new ComplexNumberTS(realPart, -imagPart));
        message = "Duas sombras do mundo imaginário (raízes complexas) são conjuradas.";
        console.log(`${message} x1 = ${roots.toString()}, x2 = ${roots.toString()}`);
    }
    return { roots, message };
}

// Testes
// Teste 1: Duas raízes reais
console.log(solveQuadraticEquationTypeScript(1.0, -3.0, 2.0));

// Teste 2: Uma raiz real dupla
console.log(solveQuadraticEquationTypeScript(1.0, -2.0, 1.0));

// Teste 3: Raízes complexas
console.log(solveQuadraticEquationTypeScript(1.0, 1.0, 1.0));

// Teste 4: 'a' é zero
console.log(solveQuadraticEquationTypeScript(0.0, 2.0, 4.0));

// Teste 5: 'a' e 'b' são zero, 'c' não é zero
console.log(solveQuadraticEquationTypeScript(0.0, 0.0, 5.0));

/*
Para executar:
1. Salve como complexSolver.ts
2. Compile: tsc complexSolver.ts
3. Execute com Node.js: node complexSolver.js
*/
```

Feitiçaria TypeScript: A classe ComplexNumberTS e a função solveQuadraticEquationTypeScript demonstram a tipagem forte do TypeScript e sua capacidade de interagir bem com o JavaScript subjacente. A interface QuadraticSolutionTS ajuda a estruturar o retorno.

Com estes jutsus, você agora pode não apenas resolver equações que permanecem no plano real, mas também aquelas cujas soluções residem no misterioso reino complexo. A álgebra, como você vê, é uma ponte entre mundos.

Capítulo 4: Além do Quadrado – Um Vislumbre de Horizontes Algébricos Mais

Amplos

Jovem adepto, você dominou as artes lineares e desvendou os mistérios das equações quadráticas, inclusive aquelas com raízes que espreitam no reino complexo. Mas o universo da álgebra é vasto, como o deserto de Suna, e há bestas muito maiores à espreita. Este capítulo é um mapa para territórios ainda mais arcanos, onde os polinômios se tornam mais... poli.

4.1 As Bestas de Muitas Faces: Polinômios de Grau Superior

Um polinômio é uma expressão que consiste em variáveis (ou indeterminadas) e coeficientes, envolvendo apenas as operações de adição, subtração, multiplicação e exponenciação inteira não negativa de variáveis.4 Já vimos os de grau 1 (lineares) e grau 2 (quadráticos). Mas e os de grau 3 (cúbicos), grau 4 (quárticos) e além?
$a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 = 0$
Analogia Naruto: Se equações lineares são genins e quadráticas são chunins, polinômios de grau superior são os jounins e Kages da álgebra – cada um com suas próprias peculiaridades e níveis de poder (complexidade para resolver).

Para polinômios de grau 3 e 4, existem fórmulas gerais (como a de Cardano para cúbicas), mas são tão monstruosamente complexas que usá-las é como tentar invocar Gamabunta para esmagar uma mosca. É um exagero, e muitas vezes mais doloroso para o invocador do que para o problema.

E para grau 5 ou superior? Aqui reside uma das grandes tragédias (ou belezas, dependendo da sua inclinação masoquista) da matemática: o **Teorema de Abel-Ruffini** afirma que não existe uma solução algébrica geral (ou seja, uma fórmula usando apenas operações aritméticas e radicais – raízes) para equações polinomiais de grau cinco ou superior.[37] É como um kinjutsu (técnica proibida) que simplesmente não pode ser dominado por meios convencionais. Galois, com sua teoria de grupos, nos deu um vislumbre do porquê disso, conectando a solubilidade de equações às propriedades de simetria de suas raízes.[37] É uma história de partir o coração para qualquer um que sonhava com uma "Fórmula Mestra Universal Para Todos os Polinômios".

Representando Polinômios em Código:
A forma mais comum de representar um polinômio em código é através de um array ou lista de seus coeficientes. O índice do array geralmente corresponde ao grau do termo.
Por exemplo, $P(x) = 6x^3 + 0x^2 + 10x + 5$ pode ser representado como:

- Rust (usando Vec<f64>): let p_coeffs: Vec<f64> = vec![5.0, 10.0, 0.0, 6.0]; (coeficiente do termo constante primeiro).[40]

- Java (usando double): double pCoeffs = {5.0, 10.0, 0.0, 6.0};.[42]
- TypeScript (usando number): let pCoeffs: number = [5.0, 10.0, 0.0, 6.0];.[42]

Operações com Polinômios (Jutsus Polinomiais):

- **Adição/Subtração:** Some/subtraia os coeficientes dos termos correspondentes. Se um polinômio P(x) é representado por coeffsP e Q(x) por coeffsQ, o polinômio soma S(x)=P(x)+Q(x) terá coeficientes coeffsS[i] = coeffsP[i] + coeffsQ[i] (cuidando dos diferentes tamanhos dos arrays).
- **Multiplicação:** Mais complexo. Se P(x)=Σaixi e Q(x)=Σbjxj, então P(x)Q(x)=Σckxk onde ck=Σi+j=kaibj. Isso envolve um loop aninhado. O grau do produto é a soma dos graus dos polinômios originais.[4]
- **Divisão:** O algoritmo de divisão de polinômios é análogo à divisão longa de números, resultando em um quociente e um resto.[45]

Implementação de Adição e Multiplicação de Polinômios:

Rust (usando Vec<f64> para coeficientes): [40]

Rust

```
// polynomial_ops_rust/src/main.rs

// Representa um polinômio como um vetor de coeficientes f64,
// onde o índice é o grau. Ex: vec![c0, c1, c2] para c0 + c1*x + c2*x^2
#
struct Polynomial {
    coeffs: Vec<f64>,
}

impl Polynomial {
    // Cria um novo polinômio a partir de um vetor de coeficientes
    fn new(coeffs: Vec<f64>) -> Self {
        // Remove zeros à direita para normalizar (opcional, mas bom para consistência)
        let mut c = coeffs;
        while c.last().map_or(false, |&val| val.abs() < 1e-9) && c.len() > 1 {
            c.pop();
        }
```

```
        Polynomial { coeffs: c }
    }

    // Retorna o grau do polinômio
    fn degree(&self) -> usize {
        if self.coeffs.is_empty() |
| (self.coeffs.len() == 1 && self.coeffs.abs() < 1e-9) {
            0 // Ou defina como -1 para polinômio zero, por convenção
        } else {
            self.coeffs.len() - 1
        }
    }

    // Adiciona dois polinômios
    fn add(&self, other: &Polynomial) -> Polynomial {
        let max_len = self.coeffs.len().max(other.coeffs.len());
        let mut result_coeffs = vec![0.0; max_len];

        for i in 0..self.coeffs.len() {
            result_coeffs[i] += self.coeffs[i];
        }
        for i in 0..other.coeffs.len() {
            result_coeffs[i] += other.coeffs[i];
        }
        Polynomial::new(result_coeffs)
    }

    // Multiplica dois polinômios
    fn multiply(&self, other: &Polynomial) -> Polynomial {
        if self.coeffs.is_empty() |
| other.coeffs.is_empty() |
|
            (self.coeffs.len() == 1 && self.coeffs.abs() < 1e-9) ||
            (other.coeffs.len() == 1 && other.coeffs.abs() < 1e-9) {
            return Polynomial::new(vec![0.0]); // Multiplicação por zero resulta em zero
        }

        let self_deg = self.degree();
        let other_deg = other.degree();
```

```
        let result_deg = self_deg + other_deg;
        let mut result_coeffs = vec![0.0; result_deg + 1];

        for i in 0..=self_deg {
            for j in 0..=other_deg {
                result_coeffs[i + j] += self.coeffs[i] * other.coeffs[j];
            }
        }
        Polynomial::new(result_coeffs)
    }

    // Converte para uma string legível
    fn to_string_poly(&self) -> String {
        if self.coeffs.is_empty() |
| (self.coeffs.len() == 1 && self.coeffs.abs() < 1e-9) {
            return "0".to_string();
        }
        let mut s = String::new();
        for (i, &coeff) in self.coeffs.iter().enumerate().rev() {
            if coeff.abs() < 1e-9 &&!(self.coeffs.len() == 1 && i == 0) { // Ignora termos zero, a
menos que seja o único termo
                continue;
            }
            if!s.is_empty() && coeff > 0.0 {
                s.push_str(" + ");
            } else if!s.is_empty() && coeff < 0.0 {
                 s.push_str(" - ");
            } else if coeff < 0.0 { // Primeiro termo e é negativo
                s.push_str("-");
            }


            let abs_coeff = coeff.abs();
            if abs_coeff!= 1.0 |
| i == 0 {
                 s.push_str(&format!("{:.2}", abs_coeff));
            } else if s.is_empty() && abs_coeff == 1.0 && i > 0 {
                // Não imprime "1" para o primeiro termo se o coeficiente for 1 e não for constante
            }
```

```
            if i > 0 {
                s.push('x');
                if i > 1 {
                    s.push_str(&format!("^{}", i));
                }
            }
        }
        if s.is_empty() &&!self.coeffs.is_empty() && self.coeffs.abs() < 1e-9 { // Polinômio era só
0.0
            return "0".to_string();
        }
        s
    }
}

fn main() {
    let p1 = Polynomial::new(vec![1.0, 2.0, 3.0]); // 3x^2 + 2x + 1
    let p2 = Polynomial::new(vec![-1.0, 0.0, 1.0, 2.0]); // 2x^3 + x^2 - 1

    println!("Polinômio P1(x) = {}", p1.to_string_poly());
    println!("Grau de P1(x) = {}", p1.degree());
    println!("Polinômio P2(x) = {}", p2.to_string_poly());
    println!("Grau de P2(x) = {}", p2.degree());

    let sum_p = p1.add(&p2);
    // (3x^2 + 2x + 1) + (2x^3 + x^2 - 1) = 2x^3 + 4x^2 + 2x
    println!("P1(x) + P2(x) = {}", sum_p.to_string_poly());

    let prod_p = p1.multiply(&p2);
    // (3x^2 + 2x + 1)(2x^3 + x^2 - 1) = 6x^5 + 3x^4 - 3x^2 + 4x^4 + 2x^3 - 2x + 2x^3 + x^2 - 1
    // = 6x^5 + 7x^4 + 4x^3 - 2x^2 - 2x - 1
    println!("P1(x) * P2(x) = {}", prod_p.to_string_poly());

    let p_zero = Polynomial::new(vec![0.0]);
    println!("Polinômio Zero: {}", p_zero.to_string_poly());
    let prod_with_zero = p1.multiply(&p_zero);
    println!("P1(x) * 0 = {}", prod_with_zero.to_string_poly());
```

}

Java (usando double para coeficientes): [42]

Java

```
// Polynomial.java
import java.util.Arrays;
import java.text.DecimalFormat;

public class Polynomial {
    private double coeffs; // coeffs[i] é o coeficiente de x^i

    // Construtor
    public Polynomial(double coeffs) {
        // Remove zeros à direita para normalizar
        int deg = coeffs.length - 1;
        while (deg > 0 && Math.abs(coeffs[deg]) < 1e-9) {
            deg--;
        }
        this.coeffs = Arrays.copyOf(coeffs, deg + 1);
    }

    public int degree() {
        if (coeffs.length == 1 && Math.abs(coeffs) < 1e-9) return 0; // Polinômio zero
        return coeffs.length - 1;
    }

    public Polynomial add(Polynomial other) {
        int maxDeg = Math.max(this.degree(), other.degree());
        double resultCoeffs = new double;

        for (int i = 0; i <= this.degree(); i++) {
            resultCoeffs[i] += this.coeffs[i];
        }
        for (int i = 0; i <= other.degree(); i++) {
            resultCoeffs[i] += other.coeffs[i];
```

```
        }
        return new Polynomial(resultCoeffs);
    }

    public Polynomial multiply(Polynomial other) {
        if ((this.coeffs.length == 1 && Math.abs(this.coeffs) < 1e-9) ||
            (other.coeffs.length == 1 && Math.abs(other.coeffs) < 1e-9)) {
            return new Polynomial(new double{0.0});
        }

        int resultDeg = this.degree() + other.degree();
        double resultCoeffs = new double; // Inicializa com zeros

        for (int i = 0; i <= this.degree(); i++) {
            for (int j = 0; j <= other.degree(); j++) {
                resultCoeffs[i + j] += this.coeffs[i] * other.coeffs[j];
            }
        }
        return new Polynomial(resultCoeffs);
    }

    @Override
    public String toString() {
        if (coeffs.length == 0 |
| (coeffs.length == 1 && Math.abs(coeffs) < 1e-9)) {
            return "0";
        }
        StringBuilder sb = new StringBuilder();
        DecimalFormat df = new DecimalFormat("#.##");

        for (int i = degree(); i >= 0; i--) {
            double coeff = coeffs[i];
            if (Math.abs(coeff) < 1e-9 &&!(degree() == 0 && i == 0)) {
                continue;
            }

            if (sb.length() > 0) {
                if (coeff > 0) {
                    sb.append(" + ");
```

```
            } else {
                sb.append(" - ");
            }
        } else if (coeff < 0) {
            sb.append("-");
        }


        double absCoeff = Math.abs(coeff);
        if (absCoeff!= 1.0 |
| i == 0) {
            sb.append(df.format(absCoeff));
        } else if (sb.length() == 0 && absCoeff == 1.0 && i > 0){
            // Não imprime "1" para o primeiro termo se o coeficiente for 1 e não for constante
        }


        if (i > 0) {
            sb.append("x");
            if (i > 1) {
                sb.append("^").append(i);
            }
        }
    }
     if (sb.length() == 0 && coeffs.length > 0 && Math.abs(coeffs) < 1e-9) { // Polinômio
era só 0.0
        return "0";
    }
    return sb.toString();
  }

  public static void main(String args) {
    Polynomial p1 = new Polynomial(new double{1.0, 2.0, 3.0}); // 3.0x^2 + 2.0x + 1.0
    Polynomial p2 = new Polynomial(new double{-1.0, 0.0, 1.0, 2.0}); // 2.0x^3 + 1.0x^2 - 1.0

    System.out.println("Polinômio P1(x) = " + p1);
    System.out.println("Grau de P1(x) = " + p1.degree());
    System.out.println("Polinômio P2(x) = " + p2);
    System.out.println("Grau de P2(x) = " + p2.degree());
```

```
        Polynomial sum_p = p1.add(p2);
        System.out.println("P1(x) + P2(x) = " + sum_p);

        Polynomial prod_p = p1.multiply(p2);
        System.out.println("P1(x) * P2(x) = " + prod_p);
    }
}
```

TypeScript (usando number para coeficientes): [42]

TypeScript

```
// polynomialOps.ts
class PolynomialTS {
    coeffs: number; // coeffs[i] é o coeficiente de x^i

    constructor(coeffs: number) {
        let deg = coeffs.length - 1;
        while (deg > 0 && Math.abs(coeffs[deg]) < 1e-9) {
            deg--;
        }
        this.coeffs = coeffs.slice(0, deg + 1);
    }

    degree(): number {
        if (this.coeffs.length === 1 && Math.abs(this.coeffs) < 1e-9) return 0;
        return this.coeffs.length - 1;
    }

    add(other: PolynomialTS): PolynomialTS {
        const maxDeg = Math.max(this.degree(), other.degree());
        const resultCoeffs: number = new Array(maxDeg + 1).fill(0);

        for (let i = 0; i <= this.degree(); i++) {
            resultCoeffs[i] += this.coeffs[i];
        }
```

```
        for (let i = 0; i <= other.degree(); i++) {
            resultCoeffs[i] += other.coeffs[i];
        }
        return new PolynomialTS(resultCoeffs);
    }

    multiply(other: PolynomialTS): PolynomialTS {
        if ((this.coeffs.length === 1 && Math.abs(this.coeffs) < 1e-9) ||
            (other.coeffs.length === 1 && Math.abs(other.coeffs) < 1e-9)) {
            return new PolynomialTS([0.0]);
        }

        const resultDeg = this.degree() + other.degree();
        const resultCoeffs: number = new Array(resultDeg + 1).fill(0);

        for (let i = 0; i <= this.degree(); i++) {
            for (let j = 0; j <= other.degree(); j++) {
                resultCoeffs[i + j] += this.coeffs[i] * other.coeffs[j];
            }
        }
        return new PolynomialTS(resultCoeffs);
    }

    toString(): string {
        if (this.coeffs.length === 0 |
| (this.coeffs.length === 1 && Math.abs(this.coeffs) < 1e-9)) {
            return "0";
        }
        let s = "";
        for (let i = this.degree(); i >= 0; i--) {
            const coeff = this.coeffs[i];
            if (Math.abs(coeff) < 1e-9 &&!(this.degree() === 0 && i === 0)) {
                continue;
            }

            if (s.length > 0) {
                if (coeff > 0) {
                    s += " + ";
                } else {
```

```
                s += " - ";
            }
        } else if (coeff < 0) {
            s += "-";
        }

        const absCoeff = Math.abs(coeff);
         if (absCoeff!== 1.0 |
| i === 0) {
            s += absCoeff.toFixed(2);
        } else if (s.length === 0 && absCoeff === 1.0 && i > 0){
            // Não imprime "1" para o primeiro termo se o coeficiente for 1 e não for constante
        }


        if (i > 0) {
            s += "x";
            if (i > 1) {
                s += `^${i}`;
            }
        }
    }
     if (s.length === 0 && this.coeffs.length > 0 && Math.abs(this.coeffs) < 1e-9) {
        return "0";
    }
    return s;
  }
}

let p1_ts = new PolynomialTS([1.0, 2.0, 3.0]); // 3.00x^2 + 2.00x + 1.00
let p2_ts = new PolynomialTS([-1.0, 0.0, 1.0, 2.0]); // 2.00x^3 + 1.00x^2 - 1.00

console.log(`Polinômio P1(x) = ${p1_ts.toString()}`);
console.log(`Grau de P1(x) = ${p1_ts.degree()}`);
console.log(`Polinômio P2(x) = ${p2_ts.toString()}`);
console.log(`Grau de P2(x) = ${p2_ts.degree()}`);

let sum_p_ts = p1_ts.add(p2_ts);
console.log(`P1(x) + P2(x) = ${sum_p_ts.toString()}`);
```

```
let prod_p_ts = p1_ts.multiply(p2_ts);
console.log(`P1(x) * P2(x) = ${prod_p_ts.toString()}`);
```

4.2 O Olho Que Tudo Vê do Sábio Algorítmico: Sistemas de Álgebra Computacional (CAS)

Para os jutsus polinomiais verdadeiramente épicos, ou quando a preguiça bate forte demais para calcular manualmente, os ninjas-alquimistas-codificadores se voltam para os **Sistemas de Álgebra Computacional (CAS)**. Pense neles como ferramentas de invocação poderosas, ou grimórios sencientes, capazes de realizar manipulações simbólicas complexas.[47] Esses sistemas não lidam apenas com números; eles manipulam os próprios símbolos e expressões, como um mestre de marionetes controlando os fios da álgebra.

O que são CAS?
Um CAS é um software que facilita o cálculo simbólico. A principal diferença entre um CAS e uma calculadora tradicional é sua capacidade de trabalhar com equações e fórmulas simbolicamente, em vez de numericamente.49 Eles podem:

- **Simplificar expressões:** Reduzir expressões complexas a formas mais simples ou canônicas.[49] Por exemplo, transformar (x+1)2 em x2+2x+1 (expansão) ou o inverso (fatoração).
- **Diferenciação e Integração Simbólica:** Calcular derivadas e integrais de funções simbolicamente, não apenas numericamente.[49]
- **Resolução de Equações (e Sistemas):** Encontrar soluções simbólicas para vários tipos de equações.[49]
- **Manipulação de Matrizes:** Realizar operações com matrizes simbolicamente.[49]

Exemplos de CAS incluem Mathematica, Maple, Maxima (descendente do Macsyma) e SymPy (uma biblioteca Python).[48] A computação simbólica, o campo por trás dos CAS, visa "fazer matemática por computador" com respostas exatas ou validadas.[50]

Como Funciona a Simplificação em CAS? (Um Feitiço de Clarividência) 51
Simplificar uma expressão como sin(x)2+cos(x)2 para 1 parece magia, mas é resultado de algoritmos e heurísticas.

- **Formas Canônicas e Padrão:** Muitos CAS tentam converter expressões para uma forma canônica (padrão). Por exemplo, expandir todos os polinômios.
- **Regras de Reescrita:** O sistema possui um vasto banco de dados de identidades matemáticas (como sin(x)2+cos(x)2=1) que aplica como regras de reescrita.[56]
- **Árvores de Expressão:** Internamente, expressões são frequentemente representadas como árvores. A simplificação envolve transformar essas árvores.[57]

- **Heurísticas:** Como "mais simples" não é um termo matematicamente bem definido, os CAS usam heurísticas (regras práticas) para decidir qual forma é "melhor". Às vezes, simplify() pode não dar o que você espera, e funções mais específicas como factor(), trigsimp(), ou powsimp() são necessárias.[51] O SymPy, por exemplo, usa uma função de medida como count_ops para determinar a complexidade e pode não modificar uma expressão se o resultado for considerado mais complexo pela métrica padrão.[55]

Diferenciação Simbólica (O Jutsu da Análise Instantânea): 53
A diferenciação simbólica aplica as regras de derivação (regra da cadeia, do produto, do quociente, etc.) recursivamente à estrutura da expressão (geralmente uma árvore de expressão).
Por exemplo, para derivar f(x)=x2sin(x):

1. Reconhece uma multiplicação: u(x)=x2, v(x)=sin(x).
2. Aplica a regra do produto: f′(x)=u′(x)v(x)+u(x)v′(x).
3. Calcula recursivamente u′(x)=2x e v′(x)=cos(x).
4. Substitui: f′(x)=2xsin(x)+x2cos(x).

Integração Simbólica (O Ritual da Reversão Temporal): 59
A integração simbólica é significativamente mais difícil que a diferenciação. Não existe um algoritmo que resolva todas as integrais elementares.

- **Tabelas de Integrais Estendidas:** Os CAS possuem vastas tabelas de integrais conhecidas.
- **Métodos Heurísticos e de Correspondência de Padrões:** Tentam transformar a integral dada em uma forma conhecida na tabela, usando substituições, integração por partes, expansão em frações parciais, etc. O algoritmo de Moses, fundamental no Macsyma/Maxima, usa uma abordagem de múltiplos estágios, começando com testes simples e correspondência de padrões antes de recorrer a métodos mais complexos.[59]
- **Algoritmo de Risch:** Para integrais de funções elementares, o algoritmo de Risch (e suas variações) pode determinar se uma integral elementar existe e, em caso afirmativo, encontrá-la. É um algoritmo muito complexo.
- **Funções Especiais:** Se uma integral elementar não puder ser encontrada, o CAS pode retornar a resposta em termos de funções especiais (ex: função Erro, integral elíptica).

Resolução de Equações Simbólicas (A Palavra de Poder): [62]

- **Equações Polinomiais:** Para graus baixos (1 a 4), podem usar fórmulas diretas. Para graus mais altos ou sistemas, técnicas como Bases de Gröbner são usadas.
- **Equações Não Polinomiais/Transcendentes:** Envolve uma mistura de técnicas:

- **Transformações Algébricas:** Isolar a variável, aplicar logaritmos, exponenciais, etc.
- **Uso de Funções Inversas:** Se f(x)=y, então x=f−1(y).
- **Métodos Específicos:** Para equações trigonométricas, exponenciais, logarítmicas.
- **Retorno de Soluções Implícitas ou Numéricas:** Se uma solução simbólica fechada não puder ser encontrada.

Os CAS revolucionaram a pesquisa matemática e a engenharia, permitindo a manipulação de expressões e a resolução de problemas que seriam intratáveis manualmente.[47] Eles automatizam cálculos tediosos e propensos a erros, liberando o pesquisador para focar nos aspectos conceituais.

4.3 Os Pergaminhos Proibidos de Buchberger: Uma Breve Incursão às Bases de Gröbner

Para sistemas de equações polinomiais multivariadas – pense em múltiplos Kages tentando coordenar um jutsu combinado terrivelmente complicado – precisamos de uma ferramenta mais poderosa: as **Bases de Gröbner**. Este é um tópico avançado, um verdadeiro *kinjutsu* da álgebra computacional, desenvolvido por Bruno Buchberger.[68]

Em essência, uma base de Gröbner é um conjunto especial de geradores para um ideal polinomial (o conjunto de todas as combinações polinomiais de um conjunto inicial de equações) que possui propriedades "agradáveis" para a resolução.[68] A principal delas é que permite uma forma de "divisão polinomial" multivariada que tem um resto único, o que é crucial para determinar se um polinômio pertence a um ideal (ou seja, se uma equação é consequência de outras).[68]

O **Algoritmo de Buchberger** é o método para calcular uma base de Gröbner a partir de um conjunto de polinômios.[68] Ele funciona iterativamente:

1. Comece com o conjunto inicial de polinômios F.
2. Para cada par de polinômios fi,fj no conjunto atual, calcule seu **S-polinômio** (um tipo especial de combinação projetada para cancelar os termos líderes).
3. Reduza (divida) este S-polinômio pelos polinômios no conjunto atual.
4. Se o resto não for zero, adicione-o ao conjunto e repita.
5. O algoritmo termina quando todos os S-polinômios se reduzem a zero em relação ao conjunto.

Uma vez que você tem uma base de Gröbner (especialmente uma com uma ordem

monomial lexicográfica), resolver o sistema de equações pode se tornar mais simples, muitas vezes reduzindo-se a resolver equações univariadas sequencialmente (propriedade de eliminação).[69]

Aplicações das Bases de Gröbner: São vastas e assustadoras [70]:

- Resolver sistemas de equações polinomiais (seu propósito original!).
- Geometria Algébrica (estudar formas definidas por polinômios).
- Teoria da Codificação (projetar e decodificar códigos corretores de erros).
- Robótica (resolver problemas de cinemática inversa – descobrir as posições das juntas de um robô para alcançar um ponto).
- Criptografia (em ataques algébricos a certos criptosistemas).
- Otimização, Biologia Computacional, e até prova automática de teoremas geométricos.

Este é apenas um arranhão na superfície. As Bases de Gröbner são um campo profundo e poderoso, mas mesmo saber que existem é como saber que Madara Uchiha tem um Rinnegan – você pode não entender todos os detalhes, mas sabe que é algo com que não se brinca.

4.4 Quando a Magia Pura Falha: A Arte da Aproximação (Métodos Numéricos)

Às vezes, nem mesmo os feitiços simbólicos mais potentes podem fornecer uma solução exata e elegante, especialmente para aqueles polinômios de alto grau teimosos ou equações transcendentes perversas. Quando a "Pedra Filosofal" da solução exata está fora de alcance, os alquimistas e ninjas se voltam para a arte da aproximação: os **métodos numéricos**.[36]

Estes métodos não lhe dão a resposta *exata*, mas podem aproximá-la com um grau de precisão assustadoramente bom. É como usar um clone da sombra para testar uma armadilha em vez de pular de cabeça.

- **Método de Newton (ou Newton-Raphson):** Um clássico. Começa com uma estimativa inicial para a raiz e, iterativamente, refina essa estimativa usando a tangente da função no ponto atual. A fórmula é $x_{k+1} = x_k - p'(x_k)p(x_k)$.[75] É rápido, mas pode falhar se o palpite inicial for ruim ou perto de pontos onde a derivada é zero (como um Chidori que erra o alvo e atinge uma parede).
- **Método de Müller:** Uma técnica mais global que usa uma interpolação quadrática (uma parábola) através de três pontos anteriores para prever a próxima aproximação da raiz. Pode encontrar raízes reais e complexas.[75]
- **Deflação:** Uma vez que uma raiz r é encontrada, podemos "deflacionar" o

polinômio dividindo-o por (x−r) para encontrar as raízes restantes no polinômio de grau inferior resultante. Útil, mas erros podem se acumular, então as raízes encontradas dessa forma muitas vezes precisam ser "polidas" (refinadas) usando o polinômio original.[75]

Esses métodos são um domínio inteiro por si só (Cálculo Numérico ou Matemática Computacional [76]), mas é bom saber que eles existem como um plano B quando a álgebra simbólica pura atinge seus limites – ou quando você simplesmente precisa de um número, não de uma bela fórmula cheia de π e raízes quadradas de coisas estranhas.

Capítulo 5: O Livro de Receitas do Alquimista / Manual de Campo Shinobi – Aplicações Arcanas

Jovem adepto, você aprendeu a manipular as energias primordiais da álgebra, a conjurar soluções de equações que antes pareciam impenetráveis. Mas para que serve todo esse poder? Assim como um ninja não aprende jutsus apenas para exibir selos de mão estilosos, e um alquimista não transmuta elementos por mero tédio, a álgebra é uma ferramenta para desvendar os segredos de outros domínios, muitas vezes inesperados. Este capítulo revela algumas das aplicações mais potentes – e, ocasionalmente, sinistras – de seus novos conhecimentos.

5.1 Criptografia: Selando Segredos com Jutsus de Números Primos e Curvas Etéreas

No mundo sombrio da espionagem e das comunicações secretas, a álgebra reina suprema. A necessidade de proteger informações – seja o plano de ataque de uma vila oculta ou a fórmula secreta da Coca-Cola – impulsionou o desenvolvimento de técnicas criptográficas incrivelmente sofisticadas.

- **RSA (O Jutsu dos Três Sábios – Rivest, Shamir, Adleman):** Este é um dos pilares da criptografia moderna de chave pública.[78] Sua segurança reside na dificuldade infernal de fatorar números muito grandes que são o produto de dois números primos gigantescos.
    - **A Alquimia:**
        1. Escolha dois números primos enormes, p e q (mantidos em segredo, como a localização da Akatsuki).
        2. Calcule $n=p\times q$ (parte da chave pública).
        3. Calcule $\phi(n)=(p-1)(q-1)$ (a função totiente de Euler, um segredo intermediário).

4. Escolha um expoente e (parte da chave pública) tal que e seja relativamente primo a $\phi(n)$.
5. Calcule o expoente d (a chave privada secreta!) tal que $ed \equiv 1 (\mod \phi(n))$. Encontrar d é como encontrar o inverso multiplicativo modular – um truque da teoria dos números.

- **Criptografando uma mensagem** M **(transformada em número):** $C = M^e \pmod n$.
- **Descriptografando** C**:** $M = C^d \pmod n$.
- A magia (e a segurança) vem do fato de que, sem conhecer p e q (ou seja, sem poder fatorar n), é computacionalmente inviável encontrar $\phi(n)$ e, portanto, d, mesmo conhecendo n e e.[79] É como tentar reverter um jutsu complexo sem conhecer os selos de mão originais.

- **Criptografia de Curva Elíptica (ECC) (O Jutsu da Geometria Astral):** Uma abordagem mais moderna que oferece segurança comparável ao RSA, mas com chaves muito menores.[78] Em vez de se basear na fatoração, a ECC usa as propriedades de grupos de pontos em curvas elípticas sobre corpos finitos.
  - **A Alquimia (Simplificada):** Pense em uma curva com uma forma específica. Existe uma maneira de "adicionar" dois pontos na curva para obter um terceiro ponto na curva. O "problema do logaritmo discreto" em curvas elípticas – dado um ponto base P e outro ponto $Q = kP$ (P "adicionado" a si mesmo k vezes), encontrar k – é extremamente difícil.
  - As chaves públicas e privadas são derivadas desses pontos e do multiplicador k.
  - A ECC é como um taijutsu mais eficiente – atinge o mesmo impacto com menos "chakra" (tamanho da chave).

A álgebra, especialmente a teoria dos números e a álgebra abstrata (corpos finitos, grupos), é a espinha dorsal desses métodos.[78]

5.2 Teoria da Codificação: Selos de Correção de Erros para Transmissões Impecáveis

Imagine enviar um pergaminho secreto através de um campo de batalha. Há uma boa chance de ele ser danificado – uma mancha de lama aqui, um buraco de kunai ali. Como garantir que a mensagem ainda possa ser lida? A teoria da codificação, fortemente dependente da álgebra, lida com isso.

- **Códigos Corretores de Erros:** O objetivo é adicionar redundância à mensagem original de forma inteligente, para que erros possam ser detectados e, idealmente, corrigidos.

- **Polinômios sobre Corpos Finitos (O Alfabeto Secreto):** Muitas técnicas de codificação representam blocos de dados como polinômios, e as operações são realizadas em **corpos finitos** (também conhecidos como Campos de Galois, GF(q)).[80] Um corpo finito é um conjunto com um número finito de elementos onde você pode adicionar, subtrair, multiplicar e dividir (exceto por zero) como de costume. GF(2), com elementos {0, 1}, é fundamental para dados binários.
    - **Exemplo – Códigos Cíclicos:** Uma classe de códigos onde um deslocamento cíclico de uma palavra-código válida ainda é uma palavra-código válida. Eles são definidos por um **polinômio gerador** g(x). Uma mensagem (polinômio m(x)) é codificada multiplicando-a por g(x) (ou através de um processo de divisão).[80] A detecção de erros envolve verificar se a palavra-código recebida é divisível por g(x).
    - **Códigos Reed-Solomon:** Um tipo poderoso de código cíclico usado em CDs, DVDs, códigos QR e comunicações espaciais. Eles são excelentes para corrigir "rajadas" de erros (muitos erros consecutivos).[80] Suas operações também ocorrem sobre corpos finitos.

A álgebra computacional é usada para encontrar bons polinômios geradores, realizar as operações de codificação e, crucialmente, para os algoritmos de decodificação, que podem ser bastante complexos (como o algoritmo de Berlekamp-Massey).[80]

5.3 Outros Reinos de Poder Algébrico (Um Tour Rápido pelas Dimensões):

A influência da álgebra e da computação simbólica se estende muito além:

- **Robótica (Marionetes de Metal e Lógica):** No projeto de robôs, a **cinemática inversa** – calcular os ângulos das juntas necessários para que a mão do robô alcance um ponto específico – frequentemente se reduz a resolver sistemas de equações polinomiais. As Bases de Gröbner são uma ferramenta para essa tarefa.[70]
- **Otimização (Encontrando o Pico da Montanha Hokage):** Muitos problemas de otimização (encontrar o melhor resultado dadas certas restrições) podem ser formulados como encontrar os mínimos ou máximos de funções polinomiais. Técnicas de otimização polinomial, às vezes usando Soma de Quadrados (SOS) e Programação Semidefinida (que têm conexões com álgebra), são usadas aqui.[84]
- **Biologia Computacional (Decifrando os Pergaminhos da Vida):** Modelar redes bioquímicas, como vias metabólicas ou redes reguladoras de genes, pode envolver sistemas de equações diferenciais ou polinomiais que descrevem as interações entre moléculas.[70] A computação simbólica pode ajudar a analisar esses modelos.

- **Física e Química (As Leis Elementares da Matéria):** Da mecânica à química quântica, equações polinomiais e diferenciais surgem constantemente para descrever fenômenos físicos e estruturas moleculares.[70]
- **Geometria Algébrica (As Formas Ocultas do Universo):** Este é o estudo de formas geométricas definidas por equações polinomiais. As Bases de Gröbner são uma ferramenta central aqui.[70]

A capacidade de manipular símbolos e resolver equações de forma algorítmica, fornecida pela álgebra computacional, transformou a maneira como a pesquisa é feita em inúmeros campos científicos e de engenharia.[47]

Tabela 3: Referência Rápida do Alquimista/Shinobi: Assinaturas do Solucionador de Equações

Tipo de Equação	Assinatura da Função Rust (Exemplo)	Assinatura do Método Java (Exemplo)	Assinatura da Função TypeScript (Exemplo)
Linear: ax+b=0	fn solve_linear(a: f64, b: f64) -> Result<f64, String>	public static String solveLinear(double a, double b)	`function solveLinearTS(a: number, b: number): number \
Quadrática: ax2+bx+c=0	fn solve_quadratic(a: f64, b: f64, c: f64) -> Result<QuadraticRoots, String> (onde QuadraticRoots contém Vec<Complex<f64>>)	public static QuadraticSolution solve(double a, double b, double c) (onde QuadraticSolution contém List<ComplexNumber>)	function solveQuadraticTS(a: number, b: number, c: number): QuadraticSolutionTS (onde QuadraticSolutionTS contém ComplexNumberTS)

Este capítulo apenas arranhou a superfície das aplicações. Cada um desses campos é um universo de estudo em si. Mas agora você sabe que a álgebra que aprendeu é mais do que meros exercícios – é uma chave mestra.

Epílogo: O Caminho do Sábio-Código Continua

E assim, jovem adepto, chegamos ao fim (por ora) deste grimório. Você viajou das planícies elementares das variáveis – o chakra bruto da matemática – até os picos

tempestuosos das equações quadráticas, espiando até mesmo o abismo vertiginoso das raízes complexas. Você aprendeu a transmutar símbolos com a precisão de um alquimista e a invocar soluções com a destreza de um mestre de jutsus.

Você viu como os selos de mão da aritmética se combinam para formar expressões, e como essas expressões, quando unidas pelo sinal de igualdade, se tornam equações – enigmas esperando para serem resolvidos, demônios esperando para serem nomeados e controlados. Você aprendeu que o discriminante é o seu Byakugan, revelando a natureza oculta das raízes antes mesmo de serem totalmente conjuradas. E você empunhou o poder de três linguagens arcanas – Rust, Java e TypeScript – para dar forma tangível a esses conceitos abstratos.

Mas lembre-se das palavras do grande sábio matemático que, ao contemplar a vastidão de seu conhecimento, disse: "Tenho uma piada de matemática realmente maravilhosa para compartilhar com vocês, mas infelizmente esta caixa de comentários é muito pequena para contê-la.".[3] Da mesma forma, este grimório, por mais denso que seja com encantamentos e analogias obscuras, é apenas um fragmento de um pergaminho muito maior. A matemática, como um pergaminho infinito ou a jornada de um sábio, é uma árvore de conhecimento extensivamente interligada, onde campos aparentemente não relacionados se conectam, e com esforço e tempo, tudo pode ser compreendido.[87]

O caminho da álgebra se estende para reinos ainda mais profundos: a elegância austera da Álgebra Linear [88], onde matrizes dançam como exércitos bem coreografados; a beleza abstrata da Teoria dos Grupos, Anéis e Corpos [38], onde as próprias estruturas da matemática são postas sob o microscópio; a complexidade numérica da Teoria dos Números [79], que guarda os segredos dos primos e da criptografia.

Não se desespere com a vastidão. Cada conceito dominado é um novo jutsu em seu arsenal, cada problema resolvido é um demônio subjugado. Que este grimório sirva não como um ponto final, mas como um portal. Continue explorando, continue questionando, continue codificando. Pois o verdadeiro poder não reside apenas em conhecer os feitiços, mas em entender os princípios por trás deles.

Agora vá, e que suas variáveis sejam sempre bem definidas, seus loops nunca infinitos (a menos que você queira), e que seus algoritmos sejam tão elegantes quanto um Haiku e tão poderosos quanto um Bijuudama. O pergaminho é infinito, jovem adepto. Há sempre mais jutsus poderosos, segredos alquímicos mais profundos e códigos

mais elegantes para escrever. A jornada do Sábio-Código apenas começou.

"Querida Álgebra, pare de nos pedir para encontrar seu X. Ela não vai voltar. E não nos pergunte Y." [3] (A menos, é claro, que seja um sistema de equações. Nesse caso, jogo justo.)

Referências Selecionadas (Fontes de Poder Adicional):

- Para piadas matemáticas que podem ou não aliviar a dor existencial de estudar:.[3]
- Para a importância do humor no ensino (a justificativa secreta deste grimório):.
- Para os mistérios da Alquimia e Transmutação:.[2]
- Para os segredos do Chakra e Jutsus de Naruto:.[8]
- Para a natureza das Variáveis e Expressões Algébricas:.[4]
- Para a resolução de Equações Lineares e Quadráticas:.[17]
- Para Polinômios e suas Operações:.[4]
- Para o mundo da Computação Simbólica e CAS:.[47]
- Para as profundezas das Bases de Gröbner:.[68]
- Para Métodos Numéricos (quando a magia exata precisa de uma mãozinha):.[36]
- Para Aplicações em Criptografia e Teoria da Codificação:.[78]
- Para conexões entre Matemática e Ocultismo/Misticismo:.[30]
- Para Estruturas Algébricas (Grupos, Anéis, Corpos):.[38]
- Para instalação e configuração das linguagens: [13] (Rust), [14] (Java), [15] (TypeScript).
- Para implementações de Números Complexos e Polinômios:.[16]
- Para exemplos de resolução de equações:.[17]

Works cited

1. O humor pode ser o Cavalo de Troia para o conhecimento ..., accessed May 23, 2025, http://www.matematica.seed.pr.gov.br/modules/noticias/article.php?storyid=604
2. Alquimia – Wikipédia, a enciclopédia livre, accessed May 23, 2025, https://pt.wikipedia.org/wiki/Alquimia
3. Me dê suas piadas matemáticas favoritas. : r/math - Reddit, accessed May 23, 2025, https://www.reddit.com/r/math/comments/u81uha/give_me_your_favorite_math_jokes/?tl=pt-br
4. formiga.ifmg.edu.br, accessed May 23, 2025, https://formiga.ifmg.edu.br/documents/2019/Cursos/Matematica/Livro_algebra-revisado.pdf
5. O que é uma variável? (vídeo) | Álgebra | Khan Academy, accessed May 23, 2025, https://pt-pt.khanacademy.org/math/7ano/xf46753cc3e03cd2f:algebra7/xf46753cc3e03cd2f:expressoes-alg/v/what-is-a-variable

6. Introdução às variáveis e expressões aritméticas - IME-USP, accessed May 23, 2025, https://www.ime.usp.br/~leo/intr_prog/introducao_var.html
7. Occult Languages and Alphabets, accessed May 23, 2025, https://digitaloccultlibrary.commons.gc.cuny.edu/occult-languages-and-alphabets/
8. Naruto & Boruto's Chakra System, Explained - CBR, accessed May 23, 2025, https://www.cbr.com/naruto-boruto-chakra-system-explained/
9. O que são Expressões Algébricas? - Matemática Básica - Professora Angela - YouTube, accessed May 23, 2025, https://www.youtube.com/watch?v=MauV62jWBSI&pp=0gcJCdgAo7VqN5tD
10. Hand Seals and One-Handed Seals, how do (you think) they work? : r/NarutoFanfiction - Reddit, accessed May 23, 2025, https://www.reddit.com/r/NarutoFanfiction/comments/m3emj0/hand_seals_and_onehanded_seals_how_do_you_think/
11. Naruto: The Mythological Roots of Hand Seals - Game Rant, accessed May 23, 2025, https://gamerant.com/naruto-hand-seals-mythological-roots/#:~:text=What%20is%20the%20Purpose%20of,chakra%20for%20techniques%20and%20jutsu.
12. Alchemical symbol - Wikipedia, accessed May 23, 2025, https://en.wikipedia.org/wiki/Alchemical_symbol
13. Getting Started with Rust - Programiz, accessed May 23, 2025, https://www.programiz.com/rust/getting-started
14. Getting Started with Java - Dev.java, accessed May 23, 2025, https://dev.java/learn/getting-started/
15. How to set up TypeScript - TypeScript, accessed May 23, 2025, https://www.typescriptlang.org/download/
16. Mathematics in TypeScript - Codefinity, accessed May 23, 2025, https://codefinity.com/courses/v2/a5c23211-8dc2-4c4a-9a83-07a51e843ab6/efb368f0-c318-4bc7-8bed-2bb1386ec5f9/9490c419-5b81-4f90-a7e7-46c8d831b8a6
17. Prove that for all integers a and b, if a ≠ 0, the equation ax + b has a rational solution, accessed May 23, 2025, https://www.geeksforgeeks.org/prove-that-for-all-integers-a-and-b-if-a-%E2%89%A0-0-the-equation-ax-b-has-a-rational-solution/
18. Solve ax+b=0 - Microsoft Math Solver, accessed May 23, 2025, https://mathsolver.microsoft.com/en/solve-problem/ax%2Bb%3D0
19. How to Solve a Linear Equation (ax + b = 0) in C - LabEx, accessed May 23, 2025, https://labex.io/tutorials/c-solve-a-linear-equation-ax-b-0-in-c-435195
20. ndarray_linalg::solve - Rust - Docs.rs, accessed May 23, 2025, https://docs.rs/ndarray-linalg/latest/ndarray_linalg/solve/index.html
21. f64 - Rust Documentation, accessed May 23, 2025, https://doc.rust-lang.org/beta/std/primitive.f64.html
22. Solve the linear system Ax = B - Mathematics Stack Exchange, accessed May 23, 2025, https://math.stackexchange.com/questions/3128295/solve-the-linear-system-ax-b

23. program that prints the result to linear equation of : ax+b = 0 - Stack Overflow, accessed May 23, 2025, https://stackoverflow.com/questions/37966852/program-that-prints-the-result-to-linear-equation-of-axb-0
24. mathsathome.com, accessed May 23, 2025, https://mathsathome.com/the-discriminant-quadratic/#:~:text=To%20calculate%20the%20discriminant%20of,%2D3%20and%20c%20%3D%204.
25. Intro-to-Java-Programming/Exercise_03/Exercise_03_01/Exercise_03_01.java at master · jsquared21/Intro-to-Java-Programming - GitHub, accessed May 23, 2025, https://github.com/jsquared21/Intro-to-Java-Programming/blob/master/Exercise_03/Exercise_03_01/Exercise_03_01.java
26. How to find the Discriminant of a Quadratic Equation? - GeeksforGeeks, accessed May 23, 2025, https://www.geeksforgeeks.org/how-to-find-the-discriminant-of-a-quadratic-equation/
27. www.ufrgs.br, accessed May 23, 2025, https://www.ufrgs.br/espmat/disciplinas/tcc/exemplotextoprontonormas.pdf
28. Implement Complex Numbers in Rust - Earvin, accessed May 23, 2025, https://earvinkayonga.com/posts/implement-complex-numbers-in-rust/
29. Java Program to Add Two Complex Numbers by Passing Class to a ..., accessed May 23, 2025, https://docs.vultr.com/java/examples/add-two-complex-numbers-by-passing-class-to-a-function
30. Sacred Geometry - Occult Science - LibGuides at Monmouth University, accessed May 23, 2025, https://guides.monmouth.edu/Occult/sacred_geometry
31. Complex in num_complex - Rust - Docs.rs, accessed May 23, 2025, https://docs.rs/num-complex/latest/num_complex/struct.Complex.html
32. Java Program to Find all Roots of a Quadratic Equation - Programiz, accessed May 23, 2025, https://www.programiz.com/java-programming/examples/quadratic-roots-equation
33. Java Program to Find the Roots of a Quadratic Equation | Baeldung, accessed May 23, 2025, https://www.baeldung.com/roots-quadratic-equation
34. How to solve quadratic equations with typescript code - CoSpaces Edu Forum - Delightex, accessed May 23, 2025, https://forum.edu.delightex.com/t/how-to-solve-quadratic-equations-with-typescript-code/11760
35. @rawify/rootfinder - npm, accessed May 23, 2025, https://www.npmjs.com/package/@rawify/rootfinder
36. Roots of Polynomials Formula - BYJU'S, accessed May 23, 2025, https://byjus.com/maths/roots-of-polynomials/
37. An introduction to Galois theory | NRICH, accessed May 23, 2025, https://nrich.maths.org/articles/introduction-galois-theory
38. Álgebra abstrata – Wikipédia, a enciclopédia livre, accessed May 23, 2025,

https://pt.wikipedia.org/wiki/%C3%81lgebra_abstrata
39. www.bienasbm.ufba.br, accessed May 23, 2025, http://www.bienasbm.ufba.br/M18.pdf
40. poly_it - Rust - Docs.rs, accessed May 23, 2025, https://docs.rs/poly_it
41. Arithmetic circuits in Rust - NP Labs, accessed May 23, 2025, https://np.engineering/posts/arithmetic-circuits/
42. Program to add two polynomials | GeeksforGeeks, accessed May 23, 2025, https://www.geeksforgeeks.org/program-add-two-polynomials/
43. a polynomial class for java implementing the basic operations + ..., accessed May 23, 2025, https://gist.github.com/derlin/40545e447fffe7599d26d0a435d9b113
44. Polynomial Array — Amplify documentation, accessed May 23, 2025, https://amplify.fixstars.com/en/docs/amplify/v0/array.html
45. cesad.ufs.br, accessed May 23, 2025, https://cesad.ufs.br/ORBI/public/uploadCatalago/15043916022012Estruturas_Algebricas_II_aula_3.pdf
46. Multiply two polynomials | GeeksforGeeks, accessed May 23, 2025, https://www.geeksforgeeks.org/multiply-two-polynomials-2/
47. casopisi.junis.ni.ac.rs, accessed May 23, 2025, http://casopisi.junis.ni.ac.rs/index.php/FUMathInf/article/download/7177/pdf
48. Sistemas de computação algebricas - mtm.ufsc.br, accessed May 23, 2025, http://mtm.ufsc.br/~daniel/amcom/CAS/p5_task.html
49. Sistema algebraico computacional - Wikipedia, la enciclopedia libre, accessed May 23, 2025, https://es.wikipedia.org/wiki/Sistema_algebraico_computacional
50. Symbolic Computation - People | Department of Mathematics - NC ..., accessed May 23, 2025, https://math.sciences.ncsu.edu/group/symbolic-computation/
51. Simplification - SymPy 1.14.0 documentation, accessed May 23, 2025, https://docs.sympy.org/latest/tutorials/intro-tutorial/simplification.html
52. Progress in Symbolic Differential Equations - Wolfram Video Archive, accessed May 23, 2025, https://www.wolfram.com/broadcast/video.php?sx=christopher%20wolfram&c=104&v=3821&disp=list&p=52&&ob=title&o=ASC
53. Mathematical Equations as Executable Models of Mechanical Systems* - Aaron Ames, accessed May 23, 2025, http://ames.caltech.edu/Walid_paper.pdf
54. Macsyma - Wikipedia, accessed May 23, 2025, https://en.wikipedia.org/wiki/Macsyma
55. Simplify - SymPy 1.14.0 documentation, accessed May 23, 2025, https://docs.sympy.org/latest/modules/simplify/simplify.html
56. Ten commandments for good default expression simplification - JSXGraph, accessed May 23, 2025, https://jsxgraph.uni-bayreuth.de/~alfred/jsxdev/Simplify/1-s2.0-S0747717110001471-main.pdf
57. (PDF) Ways to implement computer algebra compactly - ResearchGate, accessed May 23, 2025, https://www.researchgate.net/publication/235941537_Ways_to_implement_computer_algebra_compactly

58. Richard J. Fateman PhD Professor Emeritus at University of California, Berkeley - ResearchGate, accessed May 23, 2025, https://www.researchgate.net/profile/Richard-Fateman-2
59. Symbolic Integration: The Algorithms - Maxima Tutorial, accessed May 23, 2025, https://maxima.sourceforge.io/docs/tutorial/en/gaertner-tutorial-revision/Pages/SI001.htm
60. INTELIGÊNCIA ARTIFICIAL (IA) GENERATIVA E COMPETÊNCIA EM INFORMAÇÃO: HABILIDADES INFORMACIONAIS NECESSÁRIAS AO USO DE FERRA - SciELO, accessed May 23, 2025, https://www.scielo.br/j/pci/a/GVCW7KbcRjGVhLSrmy3PCng/?format=pdf&lang=pt
61. Inteligência Artificial (Peter Norvig, Stuart Russell).pdf - Kufunda.net, accessed May 23, 2025, https://www.kufunda.net/publicdocs/Intelig%C3%AAncia%20Artificial%20(Peter%20Norvig,%20Stuart%20Russell).pdf
62. Full text of "BYTE-1993-03" - Internet Archive, accessed May 23, 2025, https://archive.org/stream/BYTE-1993-03/BYTE-1993-03_djvu.txt
63. AA Introduction To MATLAB Applications in Chemical Engineering | PDF - Scribd, accessed May 23, 2025, https://www.scribd.com/document/604713496/AA-Introduction-to-MATLAB-Applications-in-Chemical-Engineering
64. A resolução de equações do segundo grau com ênfase no método de completar quadrados - FACCAT, accessed May 23, 2025, https://www2.faccat.br/portal/sites/default/files/12%20OF.pdf
65. AS DIFERENTES ESTRATÉGIAS DE RESOLUÇÃO DA EQUAÇÃO DO SEGUNDO GRAU - Ufersa, accessed May 23, 2025, https://ufersa.edu.br/wp-content/uploads/sites/58/2016/02/Disserta%C3%A7%C3%A3o-Alberton-Fagno.pdf
66. POLINÔMIOS E EQUAÇÕES, accessed May 23, 2025, https://wp.ufpel.edu.br/profpassos/files/2016/08/Economia-computacional-com-o-Mathematica%E2%84%A2-Polin%C3%B4mios-e-equa%C3%A7%C3%B5es.pdf
67. Computer Algebra Techniques in Object-Oriented Mathematical Modelling - Open Research Online, accessed May 23, 2025, https://oro.open.ac.uk/66102/1/27696813.pdf
68. www.ime.usp.br, accessed May 23, 2025, https://www.ime.usp.br/~iusenko/ensino_2020_1/MAT5737/trabalhos/Bases_de_Grobner.pdf
69. www.risc.jku.at, accessed May 23, 2025, https://www.risc.jku.at/people/buchberger/papers/2001-02-19-A.pdf
70. www.journal-cand.com, accessed May 23, 2025, https://www.journal-cand.com/article_210937_d0c6638ec9b1b9c90eb5d3af9c5fbb50.pdf
71. www.andrew.cmu.edu, accessed May 23, 2025, https://www.andrew.cmu.edu/course/15-355/lectures/lecture11.pdf
72. Buchberger's algorithm - Wikipedia, accessed May 23, 2025, https://en.wikipedia.org/wiki/Buchberger%27s_algorithm

73. (PDF) Gröbner Bases and Systems Theory - ResearchGate, accessed May 23, 2025, https://www.researchgate.net/publication/226788924_Grobner_Bases_and_Systems_Theory
74. Gröbner Bases: A Powerful Tool in Algebra - Algor Cards, accessed May 23, 2025, https://cards.algoreducation.com/en/content/6kRn0Axy/grobner-bases-algebra
75. faculty.sites.iastate.edu, accessed May 23, 2025, https://faculty.sites.iastate.edu/jia/files/inline-files/polyroots.pdf
76. Matemática Computacional | Instituto Mauá de Tecnologia, accessed May 23, 2025, https://maua.br/graduacao/cursos/engenharia-quimica/matematica-computacional/disciplina
77. ccet.ufs.br, accessed May 23, 2025, https://ccet.ufs.br/uploads/page_attach/path/9850/PPC_-_Matem_tica_Aplicada_e_Computacional__1_.pdf
78. www.ic.unicamp.br, accessed May 23, 2025, https://www.ic.unicamp.br/~rdahab/cursos/mo422-mc938/2018-2s/Welcome_files/RD-VisaoGeralCripto.pdf
79. repositorio.ufrn.br, accessed May 23, 2025, https://repositorio.ufrn.br/bitstreams/c7d294a3-ab5a-411b-a047-e86bb19d9735/download
80. www.fccdecastro.com.br, accessed May 23, 2025, http://www.fccdecastro.com.br/pdf/cs4.pdf
81. periodicos.ufms.br, accessed May 23, 2025, https://periodicos.ufms.br/index.php/porandu/article/view/7321/6084
82. Polynomial Factoring Algorithms and their Computational Complexity, accessed May 23, 2025, https://digitalcommons.lib.uconn.edu/cgi/viewcontent.cgi?article=1380&context=srhonors_theses
83. 52 Congreso Nacional Sociedad Matemática Mexicana, accessed May 23, 2025, https://www.smm.org.mx/files/2019/programa_extenso_2019.pdf
84. repositorio.ufpe.br, accessed May 23, 2025, https://repositorio.ufpe.br/bitstream/123456789/40731/1/DISSERTA%C3%87%C3%83O%20Manoel%20de%20S%C3%A1%20Jardim%20Neto.pdf
85. Symbolic Computation: Applications to Scientific Computing (Frontiers in Applied Mathematics, Series Number 5) - Amazon.com, accessed May 23, 2025, https://www.amazon.com/Symbolic-Computation-Applications-Scientific-Mathematics/dp/0898712394
86. citeseerx.ist.psu.edu, accessed May 23, 2025, https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=90b34a5c2ded25ea04bf5567fdfe6651ce37c693
87. Galois Theory | Hacker News, accessed May 23, 2025, https://news.ycombinator.com/item?id=41255456
88. www2.ufjf.br, accessed May 23, 2025, https://www2.ufjf.br/quimicaead/wp-content/uploads/sites/224/2013/05/%C3%81l

gebra-Linear-I_Vol-1.pdf
89. www.ufrgs.br, accessed May 23, 2025, https://www.ufrgs.br/reamat/AlgebraLinear/livro/livro.pdf
90. ANÉIS E CORPOS - Faculdade de Ciências da Universidade de Lisboa, accessed May 23, 2025, https://ciencias.ulisboa.pt/sites/default/files/fcul/dep/dm/Livro-GG-Aneis-e-Corpos.pdf
91. INTRODUC¸˜AO`A´ALGEBRA: QUEST˜OES COMENTADAS E RESOLVIDAS, accessed May 23, 2025, https://mat.ufpb.br/lenimar/textos/intalgebra_lna.pdf
92. AlgÉbrica - eduCAPES, accessed May 23, 2025, https://educapes.capes.gov.br/bitstream/capes/429304/2/EstAlgebrica-livro.pdf
93. cesad.ufs.br, accessed May 23, 2025, https://cesad.ufs.br/ORBI/public/uploadCatalago/09122709042014Estruturas_Algebricas_I_Aula_4.pdf
94. INTRODUÇÃO À ÁLGEBRA ABSTRATA - IC/UFAL, accessed May 23, 2025, https://ic.ufal.br/professor/jaime/livros/Introducao%20a%20Algebra%20Abstrata%2022020.pdf
95. Jutsu (Naruto) – Wikipédia, a enciclopédia livre, accessed May 23, 2025, https://pt.wikipedia.org/wiki/Jutsu_(Naruto)
96. What are the most advanced Maths jokes you know? - Reddit, accessed May 23, 2025, https://www.reddit.com/r/math/comments/49yv6a/what_are_the_most_advanced_maths_jokes_you_know/
97. Really bad math jokes - Reddit, accessed May 23, 2025, https://www.reddit.com/r/math/comments/7tefrg/really_bad_math_jokes/
98. List of alchemical substances - Wikipedia, accessed May 23, 2025, https://en.wikipedia.org/wiki/List_of_alchemical_substances
99. leafninja.com, accessed May 23, 2025, https://leafninja.com/chakra.php#:~:text=Chakra%20flows%20through%20the%20body's,the%20ninja%20to%20mold%20it.
100. [11th grade math: Quadratic Equations (ax^2+bx+c=0)] solve via factoring. A does not equal 1. - Reddit, accessed May 23, 2025, https://www.reddit.com/r/HomeworkHelp/comments/1i5tzv6/11th_grade_math_quadratic_equations_ax2bxc0_solve/
101. Calculate the Discriminant Value | GeeksforGeeks, accessed May 23, 2025, https://www.geeksforgeeks.org/calculate-discriminant-value/
102. 6.2: Quadratic Formula - Mathematics LibreTexts, accessed May 23, 2025, https://math.libretexts.org/Bookshelves/Algebra/Advanced_Algebra/06%3A_Solving_Equations_and_Inequalities/602%3A_Quadratic_Formula
103. Quadratic Equations with Imaginary Solutions (Complex Roots) | Algebra 2 - YouTube, accessed May 23, 2025, https://m.youtube.com/watch?v=fGFh-LHD874&pp=ygURI2NvbXBsZXhfZXF1YXRpb24%3D
104. Quadratic Function with Complex Numbers - Stack Overflow, accessed May 23, 2025,

https://stackoverflow.com/questions/46148277/quadratic-function-with-complex-numbers
105. How to solve imaginary quadratic roots in java? - Stack Overflow, accessed May 23, 2025, https://stackoverflow.com/questions/62741863/how-to-solve-imaginary-quadratic-roots-in-java
106. Complex Roots | GeeksforGeeks, accessed May 23, 2025, https://www.geeksforgeeks.org/complex-roots/
107. Lessons on Datasets and Paradigms in Machine Learn- ing for Symbolic Computation - arXiv, accessed May 23, 2025, https://arxiv.org/pdf/2401.13343
108. Progress in Symbolic Differential Equations - Wolfram Video Archive, accessed May 23, 2025, https://www.wolfram.com/broadcast/video.php?c=104&v=3821&disp=list&p=53
109. Rubi: Solving Integrals, One Rule at a Time - Wolfram Video Archive, accessed May 23, 2025, https://www.wolfram.com/broadcast/video.php?sx=christopher%20wolfram&c=104&v=3969&ob=date&o=DESC&disp=grid&p=38
110. Rubi: Solving Integrals, One Rule at a Time - Wolfram Video Archive, accessed May 23, 2025, https://www.wolfram.com/broadcast/video.php?c=104&ob=title&o=ASC&v=3969&p=55
111. Symbolic Scientific Computing for Multiscale Systems, accessed May 23, 2025, https://battiato.stanford.edu/research/symbolic-scientific-computing-multiscale-systems
112. Teaching Mechanics With Maple | PDF | Numerical Analysis | Equations - Scribd, accessed May 23, 2025, https://www.scribd.com/document/347944197/Teaching-Mechanics-With-Maple
113. Code Generation - SymPy 1.14.0 documentation, accessed May 23, 2025, https://docs.sympy.org/latest/modules/codegen.html
114. Hidden verification for computational mathematics - CiteSeerX, accessed May 23, 2025, https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=b8294237dcd2a967d0732d1afe584a1c6fa088ab
115. CQ Master of Electrical Engineering (Leuven) - Afdruk studiegids, accessed May 23, 2025, https://onderwijsaanbod.kuleuven.be/oa/print/get_printCQ.php?objid=50657365&jaar=2024
116. Using Symbolic Computation, Visualization and Computer Simulation Tools to Enhance Teaching and Learning of Engineering Electromagnetics - ASEE PEER, accessed May 23, 2025, https://peer.asee.org/using-symbolic-computation-visualization-and-computer-simulation-tools-to-enhance-teaching-and-learning-of-engineering-electromagnetics.pdf
117. Grim symbolic computation, accessed May 23, 2025,

https://fredrikj.net/blog/2020/01/grim-symbolic-computation/
118. CQ Master of Mathematical Engineering (Leuven) - Afdruk studiegids, accessed May 23, 2025, https://onderwijsaanbod.kuleuven.be/oa/print/get_printCQ.php?objid=52357094&jaar=2024
119. arXiv:2505.13980v1 [math.OC] 20 May 2025, accessed May 23, 2025, https://arxiv.org/pdf/2505.13980
120. SymPy With Aaron Meurer - The Python Podcast.init, accessed May 23, 2025, https://www.pythonpodcast.com/episode-42-sympy-with-aaron-meurer/
121. Symbolic and Numerical Tools for L_∞-Norm Calculation - arXiv, accessed May 23, 2025, https://arxiv.org/html/2505.13980v1
122. How to compute a Gröbner basis --- CAG L14.1 - YouTube, accessed May 23, 2025, https://www.youtube.com/watch?v=KzT2S9er93k
123. Algorithms for computing greatest common divisors of parametric multivariate polynomials, accessed May 23, 2025, http://www.mmrc.iss.ac.cn/~dwang/papers/21jsc.pdf
124. Unrestricted dynamic Gröbner Basis algorithms - Lume UFRGS, accessed May 23, 2025, https://lume.ufrgs.br/bitstream/handle/10183/194287/001093045.pdf?sequence=1
125. Determining of Level Sets for a Fuzzy Surface Using Gröbner Basis - IGI Global, accessed May 23, 2025, https://www.igi-global.com/viewtitle.aspx?TitleId=126448&isxn=9781466679962
126. Untitled, accessed May 23, 2025, https://www.instituto-camoes.pt/images/stories/tecnicas_comunicacao_em_portugues/Matematica/Matematica%20-%20O%20misticismo%20dos%20numeros.pdf
127. Símbolo do Infinito: Como Surgiu e Significado na Matemática, accessed May 23, 2025, https://mentalidadesmatematicas.org.br/como-surgiu-o-simbolo-do-infinito/
128. GA 35. Mathematics and Occultism - Rudolf Steiner Archive, accessed May 23, 2025, https://rsarchive.org/Lectures/MatOcc_index.html
129. Number Mysticism - Astrodienst Astrowiki, accessed May 23, 2025, https://www.astro.com/astrowiki/en/Number_Mysticism
130. Synchronicity and the Meaning of Numbers (0-12) - LonerWolf, accessed May 23, 2025, https://lonerwolf.com/meaning-of-numbers/
131. From Zero, All: Deriving the Hermetic Principles from Euler's Identity | ChatGPT4o, accessed May 23, 2025, https://bsahely.com/2025/04/05/from-zero-all-deriving-the-hermetic-principles-from-eulers-identity-chatgpt4o/
132. periodicos.pucminas.br, accessed May 23, 2025, https://periodicos.pucminas.br/matematicaeciencia/article/download/22099/16174/79356
133. complexible - Rust - Docs.rs, accessed May 23, 2025, https://docs.rs/complexible
134. Complex Numbers - Rust Cookbook, accessed May 23, 2025,

https://rust-lang-nursery.github.io/rust-cookbook/science/mathematics/complex_numbers.html?highlight=comple
135. Adding two complex numbers using java generics - Stack Overflow, accessed May 23, 2025, https://stackoverflow.com/questions/71050626/adding-two-complex-numbers-using-java-generics
136. Program to add and Subtract Complex Numbers using Class in Java - GeeksforGeeks, accessed May 23, 2025, https://www.geeksforgeeks.org/program-to-add-and-subtract-complex-numbers-using-class-in-java/
137. Learn TypeScript: Complex Types Cheatsheet - Codecademy, accessed May 23, 2025, https://www.codecademy.com/learn/learn-typescript/modules/learn-typescript-complex-types/cheatsheet
138. Handbook - Classes - TypeScript, accessed May 23, 2025, https://www.typescriptlang.org/docs/handbook/classes.html
139. Operators - Learn TypeScript - Free Interactive TypeScript Tutorial, accessed May 23, 2025, https://www.learn-ts.org/en/Operators
140. TypeScript Math utility class for mathematical operations - w3resource, accessed May 23, 2025, https://www.w3resource.com/typescript-exercises/typescript-class-and-oop-exercise-22.php
141. Parsing and Multiplying Numbers in TypeScript | CodeSignal Learn, accessed May 23, 2025, https://codesignal.com/learn/courses/practicing-string-operations-and-type-conversions-in-typescript/lessons/parsing-and-multiplying-numbers-in-typescript
142. discriminant in std::mem - Rust, accessed May 23, 2025, https://doc.rust-lang.org/std/mem/fn.discriminant.html
143. System of Linear Equations | GeeksforGeeks, accessed May 23, 2025, https://www.geeksforgeeks.org/system-linear-equations/
144. Quadratic Equations with Imaginary Solutions (Complex Roots) | Algebra 2 - YouTube, accessed May 23, 2025, https://www.youtube.com/watch?v=fGFh-LHD874
145. num-complex - Rust Package Registry - Crates.io, accessed May 23, 2025, https://crates.io/crates/num_complex/0.1.40/dependencies
146. The Quadratic Formula In R - GitHub Pages, accessed May 23, 2025, https://dk81.github.io/dkmathstats_site/rmath-quad-formula-r.html
147. JavaScript Program to Solve Quadratic Equation - Vultr Docs, accessed May 23, 2025, https://docs.vultr.com/javascript/examples/solve-quadratic-equation
148. Quadratic Complex roots solver - Stack Overflow, accessed May 23, 2025, https://stackoverflow.com/questions/21853509/quadratic-complex-roots-solver
149. num::complex::Complex - Rust, accessed May 23, 2025, https://docs.piston.rs/skeletal_animation/num/complex/struct.Complex.html
150. Quadratic Equations With Complex Solutions | Intermediate Algebra - Lumen Learning, accessed May 23, 2025,

https://courses.lumenlearning.com/intermediatealgebra/chapter/read-quadratic-equations-with-complex-solutions/
151. JavaScript Program to Solve a Quadratic Equation (4 Ways), accessed May 23, 2025, https://www.wscubetech.com/resources/javascript/programs/quadratic-equation
152. Roots of a quadratic function - Rosetta Code, accessed May 23, 2025, https://rosettacode.org/wiki/Quadratic_equation#Rust