四次方程（Quartic）处理方案

本文档给出在工程环境中求解一般四次方程的实用处理方案。目标是在“够用且稳健”的前提下，兼顾解析解与数值解，并明确特殊情形和数值稳定性注意事项。

通用形式：

- 一般式：a x^4 + b x^3 + c x^2 + d x + e = 0（a ≠ 0）
- 规范化：将方程两边同除以 a，得到 x^4 + A x^3 + B x^2 + C x + D = 0
  - 其中 A = b/a, B = c/a, C = d/a, D = e/a

处理策略概览

- 优先特判（降阶/因式分解迹象）：
  - 边界系数：e = 0 → 提取根 x = 0，余下为三次方程
  - 无一次项：C = 0 → 尝试配方或改写为双二次（biquadratic）
  - 双二次结构：x^4 + p x^2 + q = 0 → 令 y = x^2，降为二次
  - 对称/倒数对称：若系数满足对称结构，可作变量替换 y = x + 1/x 等
- 解析解（Ferrari 法）用于通用情形：
  - 通过 Tschirnhaus 变换消去三次项，得到凹陷四次（depressed quartic）
  - 构造并求解伴随的解三次（resolvent cubic）
  - 由解三次的根回代，开方组合得到四次方程的根
- 数值解作为稳健后盾：
  - 对复杂或病态系数，优先使用数值多项式根算法（如伴随矩阵特征值、Durand–Kerner、Jenkins–Traub）
  - 双精度计算下对重根/近重根需谨慎，必要时使用高精度

C++ 示例：Durand–Kerner 数值求根（自包含）

下面给出一个不依赖第三方库的 C++ 实现，使用 Durand–Kerner 迭代法求解四次多项式的 4 个复根。该方法对一般情形稳健，适合工程使用，并含有简单的缩放与残差校验。

```
#include <array>
#include <complex>
#include <vector>
#include <cmath>
#include <limits>
#include <iostream>

using cd = std::complex<double>;

// 评估 P(x) = a x^4 + b x^3 + c x^2 + d x + e
static inline cd eval_poly(cd x, double a, double b, double c, double d, double e)
{
    return (((a*x + b)*x + c)*x + d)*x + e;
}
```

```cpp
// Durand-Kerner 求根（四次）。返回 4 个复根。
std::array<cd,4> solve_quartic_dk(double a, double b, double c, double d, double
e,
                                  int max_iter = 200, double tol = 1e-14) {
    // 归一化，避免数值放大：令 P(x)/=a
    if (a == 0.0) throw std::invalid_argument("a must not be 0");
    double A = b/a, B = c/a, C = d/a, D = e/a;

    // 初始值：单位圆上等间隔点（使用复单位根的扰动）
    const double pi = 3.14159265358979323846;
    std::array<cd,4> r = {
        cd(1.0, 0.0),
        std::polar(1.0, 2*pi/4),
        std::polar(1.0, 4*pi/4),
        std::polar(1.0, 6*pi/4)
    };
    // 轻微扰动避免对称退化
    for (int i=0;i<4;++i) r[i] += cd(1e-3*i, -1e-3*i);

    auto eval_mon = [&](cd x){ return (((x + A)*x + B)*x + C)*x + D; }; // x^4 + A
x^3 + B x^2 + C x + D

    for (int it=0; it<max_iter; ++it) {
        double max_step = 0.0;
        for (int i=0; i<4; ++i) {
            cd denom(1.0, 0.0);
            for (int j=0; j<4; ++j) if (j != i) denom *= (r[i] - r[j]);
            cd delta = eval_mon(r[i]) / denom; // 已规范化后的多项式
            r[i] -= delta;
            max_step = std::max(max_step, std::abs(delta));
        }
        if (max_step < tol) break;
    }

    return r;
}

int main() {
    // 示例：x^4 - 5x^2 + 4 = 0 → x = ±1, ±2
    double a=1, b=0, c=-5, d=0, e=4;
    auto roots = solve_quartic_dk(a,b,c,d,e);

    std::cout.setf(std::ios::fixed); std::cout.precision(12);
    for (int i=0;i<4;++i) {
        std::cout << "root[" << i << "] = " << roots[i] << "\n";
        auto res = eval_poly(roots[i], a,b,c,d,e);
        std::cout << "  residual = " << std::abs(res) << "\n";
    }
    return 0;
}
```

要点：

- 系数先做首项归一化，改善条件数。
- 迭代停止条件基于步长阈值 `tol`，可根据需求调整。
- 可对根进行后排序或筛选近似实根（判 $|\mathrm{Im}| < eps$）。

Ferrari 法（思路提要）

设规范化后：$x^4 + A x^3 + B x^2 + C x + D = 0$。

1. 消去三次项（Tschirnhaus 变换）

- 令 $x = y - A/4$，得到凹陷四次：
  - $y^4 + p y^2 + q y + r = 0$
  - 系数：
    - $p = B - 3A^2/8$
    - $q = C - A B/2 + A^3/8$
    - $r = D - A C/4 + A^2 B/16 - 3A^4/256$

2. 构造解三次（Resolvent Cubic）并取一实根 $z0$

3. 反求平方根并分解为两个二次，解得 y 后回代 $x = y - A/4$。

注：不同教材/实现对 resolvent cubic 与后续 R、S、T 的表达式存在等价变形；实现时选取一套自洽配方并进行充足的数值测试。

特殊形态与简化

- 双二次（biquadratic）：$x^4 + p x^2 + q = 0$
  - 令 $y = x^2$，解 $y^2 + p y + q = 0 \to y1, y2$
  - 若 $yk \geq 0$，则 $x = \pm \mathrm{sqrt}(yk)$；若 $yk < 0$，则对应复根
- 缺失常数项（$D = 0$）：$x = 0$ 为一根，余下三次可用 Cardano 法或数值法
- 两两平方和形式：$x^4 + a x^2 + b = (x^2 + u x + v)(x^2 - u x + w) \to$ 比较系数解 u,v,w
- 有理根可检验：若系数为整数，可先用有理根定理测试，从而降阶

数值方法与稳定性提示

- 伴随矩阵特征值法在库支持到位时更高效；无外部库时可优先选用 Durand–Kerner。
- 近重根建议用更严格的 `tol` 与更高 `max_iter`，必要时切换到高精度库。
- 求得根后回代检查残差，常用阈值 1e-10 ~ 1e-12。

测试用例建议

- $x^4 - 5x^2 + 4 = 0 \to x=\pm1, \pm2$
- $x^4 - 1 = 0 \to x=\pm1, \pm i$
- $x^4 + 2x^3 - x - 2 = 0 \to$ 可检测有理根 $x=1, -2$
- 病态：近重根、系数量级差异大（如 1e8 与 1e-8 混合）

参考资料

- Ferrari’s method for quartic equations
- Jenkins–Traub, Durand–Kerner 多项式求根
- Numerical Recipes, Roots of Polynomials
- Higham, Accuracy and Stability of Numerical Algorithms

> 注：解析解与数值法可并存；工程实现中建议默认数值法，必要时补充解析分支并进行残差验证。