一元五次及以上多项式方程的实根求解

吕敬一 (lv-jy22@mails.tsinghua.edu.cn)
潘佳奇 (panjq22@mails.tsinghua.edu.cn)
叶隽希 (yejx22@mails.tsinghua.edu.cn)
2023 年 10 月 10 日

目录

1 问题描述 **2**

2 前置约定 **2**
2.1 定义 . . . 2
2.2 基本事实 . . . 2
2.3 引理 . . . 3

3 算法设计：朴素二分法 **5**
3.1 前提准备 . . . 5
3.2 算法流程 . . . 6
3.3 数值验证 . . . 7

4 算法设计：Sturm-Newton 算法 **8**
4.1 前提准备 . . . 8
4.2 算法流程 . . . 10
4.3 理论证明 . . . 10
4.4 数值验证 . . . 16

5 算法设计：基于笛卡尔符号规则的实根分离 **17**
5.1 前提准备 . . . 17
5.1.1 重根的处理 . . . 17
5.1.2 实根分离 . . . 19
5.2 算法流程 . . . 20
5.2.1 实根分离算法-VAS 法 . . . 20
5.2.2 实根分离算法-二分法 . . . 21
5.3 理论证明 . . . 22

6 对各算法的进一步优化 **24**
6.1 精度优化 . . . 24
6.2 效率优化 . . . 25

7 数值验证与算法改进 **26**
7.1 根正态分布的多项式 . . . 26
7.2 形态更全面的多项式 . . . 28
7.3 测试与改进 . . . 28

8 总结 **31**

1 问题描述

给定一元 5 次以上多项式方程的系数，求解方程实根的存在情况（包括是否存在，是否有重根）以及具体数值，并计算求得 double 类型数值的绝对误差，使其尽量小。

2 前置约定

2.1 定义

首先给出接下来会用到的一些定义：

定义 1: 邻域

$x_0 \in \mathbb{R}$ **的邻域** $U(x_0, \delta) = (x_0 - \delta, x_0 + \delta)$
$x_0 \in \mathbb{R}$ **的去心邻域** $\mathring{U}(x_0, \delta) = (x_0 - \delta, x_0) \cup (x_0, x_0 + \delta)$

定义 2: 多项式的取模与整除

对于多项式 $f(x), g(x)$，记 $f(x)$ 对 $g(x)$ 取模的结果为 $\operatorname{rem}(f(x), g(x))$，则 $\operatorname{rem}(f(x), g(x))$ 是满足如下条件的唯一多项式（$h(x)$ 也是多项式）：

$$\begin{cases} f(x) = g(x)h(x) + \operatorname{rem}(f(x), g(x)) \\ \deg \operatorname{rem}(f(x), g(x)) < \deg g(x) \end{cases}$$

如果 $\operatorname{rem}(f(x), g(x)) = 0$，则称 $f(x)$ **被** $g(x)$ **整除**或 $f(x)$ **包含因子** $g(x)$。

定义 3: 多项式的最大公因式

如果多项式 $f(x)$ 和 $g(x)$ 都被多项式 $h(x)$ 整除，则称 $h(x)$ **是** $f(x)$ **和** $g(x)$ **的公因式**。在 $f(x)$ 和 $g(x)$ 的公因式中次数最高且最高次项系数为 1 的被称为 $f(x)$ **和** $g(x)$ **的最大公因式**，记为 $\gcd(f(x), g(x))$。两个多项式的最大公因式唯一且非零。

2.2 基本事实

在研究过程中，我们注意到如下基本事实：

事实 1: 介值定理

对于 $\mathbb{R}$ 上的连续函数 $f(x)$，若 $f(a)f(b) < 0 (a < b)$，则 (a, b) 中存在 $f(x)$ 的零点。

事实 2: 代数基本定理的推论

任一实系数 n 次多项式至多有 n 个实根（含重根）。

事实 3：带余项的泰勒公式

设 $f(x)$ 在 x_0 的某个邻域内存在 n 阶导数，则当 $x \to x_0$ 时，

$$f(x) = \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!}(x-x_0)^k + o((x-x_0)^n)$$

$$f(x) = \sum_{k=0}^{n-1} \frac{f^{(k)}(x_0)}{k!}(x-x_0)^k + \frac{f^{(n)}(\xi)}{n!}(x-x_0)^n$$

其中 ξ 在 x_0 和 x 之间。

事实 4：最大公因式的基本性质

以下 $f(x), g(x), h(x)$ 为多项式，$a(a \neq 0)$ 为 $f(x)$ 的最高次项系数，k 为任意非零实数。

$$\gcd(f(x), kg(x)) = \gcd(f(x), g(x))$$

$$\gcd(f(x), g(x)) = \gcd(f(x), g(x) + h(x)f(x))$$

$$\gcd(f(x), 0) = \frac{f(x)}{a}$$

$$\gcd(f(x)h(x), g(x)h(x)) = h(x)\gcd(f(x), g(x))$$

事实 5：多项式的微分性质

任意多项式都是 $\mathbb{R}$ 上的连续函数；

任意多项式的导数还是多项式；

若某多项式在 $x = x_0$ 处取极小值，则存在 $\delta > 0$ 使得 $\forall x \in (x_0 - \delta, x_0), f'(x) < 0$ 且 $\forall x \in (x_0, x_0 + \delta), f'(x) > 0$ 。极大值情况同理。

2.3 引理

并给出了如下引理：

引理 1：多项式重根数与高阶导数值的关系

若多项式 $f(x)$ 有实根 x_0 ，则 $f(x)$ 在 $x = x_0$ 处的重根数为 $r(r \leq \deg f)$ 是 $f^{(i)}(x_0) = 0,\ (i \in \{1, 2, \cdots, r-1\})$ 且 $f^{(r)}(x_0) \neq 0$ 的充分条件。

贡献认定：由 **吕敬一** 提供。

下文朴素二分法与 Sturm-Newton 算法中 gb_getRepetition() 函数均为该引理的直接应用，其原理不再赘述。

证明 1：多项式重根数与高阶导数值的关系

据假设，多项式 $f(x)$ 含有因子 $(x-x_0)^r$ ，不妨设 $f(x)=(x-x_0)^r g(x)$ ，其中 $g(x)$ 不含因子 $(x-x_0)$ 。考虑 $f(x)$ 的 $1,2,\cdots,r$ 阶导数：

$$
\begin{aligned}
f^{(1)}(x) &= r(x-x_0)^{r-1}g(x)+(x-x_0)^r g^{(1)}(x)\\
f^{(2)}(x) &= r(r-1)(x-x_0)^{r-2}g(x)+2r(x-x_0)^{r-1}g^{(1)}(x)+(x-x_0)^r g^{(2)}(x)\\
f^{(3)}(x) &= r(r-1)(r-2)(x-x_0)^{r-3}g(x)+3r(r-1)(x-x_0)^{r-2}g^{(1)}(x)+3r(x-x_0)^{r-1}g^{(2)}(x)\\
&\quad +(x-x_0)^r g^{(3)}(x)\\
&\vdots\\
f^{(r-1)}(x) &= \sum_{i=0}^{r-1}\binom{r-1}{i}(x-x_0)^{r-i}g^{(r-1-i)}(x)\prod_{j=0}^{i-1}(r-j)\\
f^{(r)}(x) &= \sum_{i=0}^{r}\binom{r}{i}(x-x_0)^{r-i}g^{(r-i)}(x)\prod_{j=0}^{i-1}(r-j)
\end{aligned}
$$

观察发现 $f^{(1)}(x),f^{(2)}(x),\cdots,f^{(r-1)}(x)$ 中每一项都被 $(x-x_0)$ 整除，从而 $f^{(1)}(x)=f^{(2)}(x)=f^{(r-1)}(x_0)=0$ 。而 $f^{(r)}(x)$ 中恰有一项 $(x-x_0)^0 g(x)r!$ 不含因子 $(x-x_0)$ ，故 $f^{(r)}(x_0)\neq 0$ 。 □

贡献认定：由 **吕敬一** 提供。

引理 2：无重根多项式与其导函数的公因子

对于无重根多项式 $f(x)$，及其导函数 $f'(x)$，有其最大公因子为 1，即 $\gcd(f(x),f'(x))=1$

贡献认定：由 **潘佳奇** 提供。

证明 2：无重根多项式与其导函数的公因子

对于 $f(x)$ 得其分解 $f(x)=(x-x_1)(x-x_2)\dots(x-x_n)$，则 $f(x)$ 只可能有 $(x-x_i)$ 形式的因子。

对于 $f'(x)$，有

$$f'(x)=\sum_{i=1}^{n}-x_i(x-x_1)\cdots(x-x_{i-1})(x-x_{i+1})\cdots(x-x_n).$$

假设 $f'(x)$ 被 $(x-x_i)$ 整除，则有

$$f'(x)\equiv -x_i(x-x_1)\cdots(x-x_{i-1})(x-x_{i+1})\cdots(x-x_n)\pmod{(x-x_i)}.$$

但是因为 $f(x)$ 没有重根，$(x-x_1)\cdots(x-x_{i-1})(x-x_{i+1})\cdots(x-x_n)$ 不被 $(x-x_i)$ 整除，与 $f'(x)$ 被 $(x-x_i)$ 整除矛盾。

所以 $\gcd(f(x),f'(x))=1$。 □

贡献认定：由 **潘佳奇** 提供。

3 算法设计：朴素二分法

3.1 前提准备

根据事实，我们自然地考虑以二分法为主体求解，配合一系列对区间的其他划分、调整操作来处理特殊情况。

问题 1：对可能存在的实根作界的估计

由于我们计划以二分法为主体来进行求解，需要先确定实根的上下界。

贡献认定：由 **吕敬一** 提供。

定理 1：多项式实根的界

多项式方程 $a_nx^n+\cdots+a_1x+a_0=0$ 如果存在实根，则实根必在区间 $[-bound,bound]$ 中，其中

$$bound=1+\max_{i=0}^{n-1}\left|\frac{a_i}{a_n}\right|$$

贡献认定：由 **吕敬一** 摘自维基百科[1]。

证明 3：多项式实根的界

设 $x_0\in\mathbb{R}^*$ 是原多项式的一个实根，$|x_0|\le 1$ 的情形自不必证。下设 $|x_0|>1$，则有

$$a_nx_0^n+a_{n-1}x_0^{n-1}+\cdots+a_0=0$$

[1] https://en.wikipedia.org/wiki/Geometrical_properties_of_polynomial_roots

即

$$\begin{aligned}|a_n x_0^n| &= \left|\sum_{i=0}^{n-1} a_i x^i\right| \\ &\le \sum_{i=0}^{n-1} |a_i x_0^i| \\ &\le \max_{i=0}^{n-1} |a_i| \sum_{i=0}^{n-1} |x_0^i| \\ &= \max_{i=0}^{n-1} |a_i| \frac{|x_0|^n - 1}{|x_0| - 1} \\ &< \max_{i=0}^{n-1} |a_i| \frac{|x_0|^n}{|x_0| - 1}\end{aligned}$$

从而

$$|x_0| - 1 < \frac{\max_{i=0}^{n-1} |a_i|}{|a_n|}$$

即

$$|x_0| < 1 + \max_{i=0}^{n-1} \left|\frac{a_i}{a_n}\right|$$

□

贡献认定：由 **吕敬一** 提供。

在求得实根的界之后，设计了如下算法来求解实根。

3.2 算法流程

算法 1: 朴素二分法

步骤 1. 将区间 $[-bound, bound]$ 作为初始区间插入求解队列 q 中。

步骤 2. 若已经找到的实根数与多项式次数相同，或队列 q 为空，则跳转到 **步骤 9**。否则，取出队列 q 最前端的区间 $[l, r]$ ，若区间未空则重新执行 **步骤 2**，否则计算其端点的函数值 leftVal 与 rightVal ，并与 0 进行比较（此处考虑容差）。

步骤 3. 若左端点或右端点函数值为 0，则调用 `gb_getRepetition()` 计算该点处的重根数并将其加入答案列表中，然后调用 `gb_findNotZero()` 缩小区间，使得端点函数值不为 0.

步骤 4. 若区间长度过短（小于容差值）则取其中点 m 代入函数判断是否是 0，若是则同样计算重根数并加入答案，而后跳转到 **步骤 2**.

步骤 5. 若区间两端函数值异号，则执行**步骤 6、7**。否则，执行 **步骤 8**.

步骤 6. 区间两端函数值异号，则根据介值定理，区间内必有实根存在，尝试找到其中一个根 res 。记当前正在处理的区间为 $I = [l, r]$，取其中点 m，若 m 处函数值为 0 或区间过短（小于容差），则记 res 为 m 并进入下一步；否则，若 l 点和 m 点函数值异号，则将 I 置为 $[l, m]$ 并重新执行这一步；否则，若点 r 和点 m 函数值异号（这必然成立），则将 I 置为 $[m, r]$ 并重新执行这一步。

步骤 7. 计算点 res 处的重根数并将其加入答案列表，然后调用 `gb_findNotZero()` 找到 res 前后最近的函数值不为 0 的实数 r_0, l_0，并将 $[l, r_0]$ 和 $[l_0, r]$ 插入到队列 q 末尾。而后重新回到步骤 2.

步骤 8. 在当前迭代深度不过大的情况下，将当前处理的区间等分为 8 段，并依次加入到队列 q 末尾。而后重新回到 **步骤 2**.

步骤 9. 对于已求出的实根，考虑到过程中的一系列误差可能性，对根进行进一步去重。将答案列表从小到大排序，记上一个保留的根为 x，x 初始为最小的实根。从头遍历，若当前实根减去 x 大于容差，则将 x 更新为当前实根；否则将 x 的重复数加上当前实根的重复数，并设法删除当前实根。

算法结束。此算法由吕敬一进行了代码实现。

贡献认定：由 **吕敬一** 提供。

3.3 数值验证

由于本算法存在明显的正确性问题，暂不对其进行理论证明。关于数值验证涉及的具体方法，见本文第五节：数值验证的方法。

对算法 1 进行了数值验证，注意到三条主要问题：

(1) 偶数次重根会被直接忽略，或者在某些情况下在实际根附近求出若干不精确的根。初步判断为二分法的结构性缺陷（参考引理 2），需要后续改进算法。

例子 1

当多项式为

$$\begin{aligned}
&-0.000000020126899095761600753253138464289595521705678038415499031 5+\\
&0.000002123610183896995262431200632291528052064677467569708824157 7x+\\
&-0.000050364710444953952352391962632438549007929395884275436401367 2x^2+\\
&-0.000570193171519725052441562862526325261569581925868988037109375x^3+\\
&0.015612231183381179017155560018181859049946069717407226562 5x^4+\\
&-0.064944943106635774632273694351169979199767112731933593 75x^5+\\
&-0.129099119753301877500106797924672719091176986694335937 5x^6+\\
&0.712935591561112524949805901997024193406105041503906 25x^7+\\
&0.964516626227685747174689367966493591666221618652343 75x^8+\\
&-2.008020007154295960560830280883237719535827636718 75x^9+\\
&-3.393990517759712943757222092244774103164672851562 5x^{10}+\\
&0.386601472165507331624212383758276700973510742187 5x^{11}+\\
&2.571900337086786336726618173997849225997924804687 5x^{12}+\\
&x^{13}
\end{aligned}$$

时（精确实根为 -1.43822, -0.910655, -0.844453, -0.743001, -0.600247, -0.0570855, 0.0187824, 0.0187824, 0.106847, 0.197466, 0.197466, 0.490085, 0.992328 ），所求出的根为 -1.43822, -0.910654, -0.844448, -0.743005, -0.600246, -0.0570885, 0.106844, 0.49008, 0.992327，不难发现各出现了两次的 0.0187824 和 0.197466 被漏掉了。

当多项式为

$$(x+0.889715718592040460954706304619321599602699279785156 25)^{10}$$

时（精确实根为 -0.889715718592040460954706304619321599602699279785156 25，重根 10 次），所求出的实根为-0.930381 和-0.849622，结果精度非常低。

贡献认定：由 **吕敬一** 提供。

(2) 某些根与实际根的误差超过了容差。有可能是代码实现问题，也有可能是多项式求值时容差选取不够优秀，还有可能是浮点数自身的误差导致。

例子 2

当多项式为

$$\begin{aligned}
&-0.000005806440347306378527559959923376453616583603434264659881591 8+\\
&0.000287495767995746395653128280045507381146308034658432006835937 5x+\\
&-0.006137730445032065981769697771142091369256377220153808593 75x^2+\\
&0.073487291130911971959172035440133186057209968566894531 25x^3+\\
&-0.536381366814627447148211558669572696089744567871093 75x^4+\\
&2.420421284825134744522756591322831809520721435546875x^5+\\
&-6.490042806599969793523996486328542232513427734375x^6+\\
&9.210096528363649781567801255732774734497070312 5x^7+\\
&-5.249481342694291896577851730398833751678466796875x^8+\\
&x^9
\end{aligned}$$

时（精确实根为 7 次重根的 0.144122 和两次重根的 2.12031），求出的实根为 0.145712，其与 0.144122 的误差超过了预设的容差 10^{-5}。

贡献认定：由 **吕敬一** 提供。

(3) 在算法 1 的步骤 8 中进行了将区间等分并递归求解的操作。这一操作是为了在两端函数值同号的区间中找到两端函数值异号的区间以进行二分，但实际上产生了很多不必要的区间，造成了巨大的时间开销，程序运行缓慢。

4 算法设计：Sturm-Newton 算法

从上述问题中可以看出，找到有根区间是一个较大的难点。此外，可能需要考虑二分法以外的数值方法来应对偶数次重根的情况。

4.1 前提准备

问题 2：实根分离

对于没有重根的多项式，实根分离是构造一系列不相交的区间，使得这些区间不相交，且每个区间内有且只有一个实根，并且多项式所有的实根都被区间覆盖到。

贡献认定：由 **叶隽希** 摘自维基百科[2]。

使用实根分离可以更好地使用迭代法，因为每个区间只有一个实根，那么迭代法就能快速地找到多项式所有的实根。我们首先提出了用 Sturm 定理来解决实根分离问题的算法。定理说明如下：

定义 4：Sturm 序列

对于多项式 f，定义 f 的 **Sturm 序列** T 如下:

$$\begin{aligned} T_0 &= f \\ T_1 &= f' \\ T_{n+1} &= -\mathrm{rem}(T_{n-1}, T_n) \quad (n \geq 1,\ \deg T_n \geq 1) \end{aligned}$$

其中 $\mathrm{rem}(f,g)$ 是唯一满足 $f = gh + \mathrm{rem}(f,g)$，且 $\deg \mathrm{rem}(f,g) \leq \deg g - 1$ 的多项式。此处 h 亦为某多项式。

贡献认定：由 **吕敬一** 摘自维基百科[3]。

定义 5：实数在多项式序列中的变号数

对于多项式序列 $T_0, T_1, \cdots, T_m$ 和 $x \in \mathbb{R}$，记数列 $\{v_n\}$ 为 $v_i = T_i(x)\ (0 \leq i \leq m)$。

定义 x **在** T **中的变号数**为 $\{v_n\}$ 中满足 $v_i \neq 0$ 且 v_i 与其前一个非零项异号的正下标 i 的数量，记作 $\mathrm{var}(T, x)$ 。

贡献认定：由 **吕敬一** 摘自维基百科。

定理 2：Sturm 定理

若 $a, b\ (a < b)$ 都不是多项式 f 的零点，则 f 在 (a,b) 上的不同零点个数为 $\mathrm{var}(T,a) - \mathrm{var}(T,b)$，其中 T 为 f 的 Sturm 序列。

贡献认定：由 **吕敬一** 摘自维基百科。

根据定理 2，利用上一节中子问题的结果，结合牛顿迭代法（步骤 6、7、8）设计了如下算法：

[2] https://en.wikipedia.org/wiki/Real-root_isolation
[3] https://en.wikipedia.org/wiki/Sturm%27s_theorem

4.2 算法流程

算法 2: Sturm-Newton 算法

步骤 1. 对于输入的多项式 f ，求出其 Sturm 序列 T 备用。

步骤 2. 将区间 $[-bound, bound]$ 作为初始区间插入求解队列 q 中。

步骤 3. 若 q 为空，则跳转至步骤 9；否则，取出队列 q 最前端的区间 $[l, r]$。若 $f(l)$ 在容差允许范围内等于 0，则将 l 加入答案列表，并调用 `gb_findNotZero()` 找到大于 l 的最近的使得 $f(l')$ 在容差范围内不为 0 的浮点数 l'，并令 $l := l'$ 。对于 r 同理亦然。

步骤 4. 利用 Sturm 定理计算 $[l, r]$ 中不同实根的数量 rootNum。若 rootNum 大于等于 2，或 rootNum 等于 1 且 $r - l$ 大于等于给定阈值 L，则进入步骤 5；若 rootNum 等于 0，跳转至步骤 3；否则，跳转至步骤 6。

步骤 5. 若 rootNum 大于等于 2，则将 $[l, r]$ 作二等分；否则将 $[l, r]$ 作 k 等分。把等分所得区间插入队列 q 末尾，回到步骤 2。

步骤 6. 令 $x_0 := \dfrac{l+r}{2}$ 。不断执行步骤 7，直至步骤 7 已被执行 ITER_LIM 次或 $|\dfrac{f(x_0)}{f'(x_0)}| < \dfrac{e}{10}$ 。

步骤 7. 若 $f'(x_0) \neq 0$，则令 $x := x_0 - \dfrac{f(x_0)}{f'(x_0)}$；否则，调用 `gb_findNotZero()` 找到大于 x_0 的最近的使得 $f'(x_1) \neq 0$ 的浮点数 x_1 ，并令 $x := x_1 - \dfrac{f(x_1)}{f'(x_1)}$ 。最后，再置 $x_0 := x$ 。

步骤 8. 将 x_0 加入答案列表，跳转至步骤 3。

步骤 9. 对于答案列表中的每一个根 x，调用 `gb_getRepetition()` 计算其重根数。若 $f'(x) \neq 0$，将 $|\dfrac{f(x)}{f'(x)}|$ 记为其误差；否则，将其误差记为 0。

步骤 10. 按照要求，根据答案列表中根的情况设置变量 flag 的值。

算法结束。此算法由吕敬一进行了代码实现。

贡献认定：由 **吕敬一** 提供。

在实际操作中，取 $L = 0.1, k = 2, \text{ITER\_LIM} = 100$ 。关于 k 的取值，在算法设计时进行了分析，详见下一小节理论证明。

4.3 理论证明

定理 3: 算法 2 中区间等分数 k 的最优值为 2

证明：

对于一个有一个根 x_0 但未作求解的区间 I，设其长度为 l $(l > L)$，设单次求解某实数在 Sturm 序列中变号数的时间开销为 s，每个区间 k 等分需要计算 $k-1$ 个分割点的变号数（区间端点由于上一层分割时已计算过，可用 unordered_map 存储下来，不再计算）。设由 I 最终

求出长度小于 L 的只有一个根的区间的时间开销为 $f(k,l)$，

$$\begin{aligned} f(k,l) &= (k-1)s\log_k \frac{l}{L} \\ &= s\ln\frac{l}{L}\cdot\frac{k-1}{\ln k} \end{aligned}$$

记 $g(x)=\dfrac{x-1}{\ln x}$ $(x\geq 2)$, 有 $g'(x)=\dfrac{\ln x-1+\frac{1}{x}}{(\ln x)^2}$, 注意到 $x\in[3,+\infty)$ 时，$\ln x-1+\dfrac{1}{x}>0$，则 $g(x)$ 在 $[3,+\infty)$ 的最小值为 $g(3)=\ln 3-\dfrac{2}{3}>\ln 2-\dfrac{1}{2}$。而算法流程中 k 为大于等于 2 的整数，故 $k=2$ 时对于任意给定的 l，$f(k,l)$ 取得最小值。 □

贡献认定：由 **吕敬一** 提供。

现补充 Sturm 定理以及步骤 6、7、8 所涉及之牛顿迭代法的证明。

证明 4: Sturm 定理

对于多项式 $f(x)$ 和 $a,b\in\mathbb{R}$ $(a<b)$，记 $f(x)$ 的 Sturm 序列为 T：$T_0,T_1,\cdots,T_m$。假设 a,b 均不是 $f(x)$ 的根。

首先证明 T 的性质：任意 $1\leq i<m,\gcd(T_i,T_{i+1})=\gcd(T_0,T_1)$。

Sturm 序列的性质

对于 $i=1$ 有

$$\begin{aligned} \gcd(T_1,T_2) &= \gcd(T_1,-\mathrm{rem}(T_0,T_1)) \\ &= \gcd(T_1,\mathrm{rem}(T_0,T_1)) \\ &= \gcd(T_1,T_0-h(x)T_1) \\ &= \gcd(T_0,T_1) \end{aligned}$$

其中 $h(x)$ 是某多项式。同理依次可证明 $i=2,3,\cdots,m-1$ 的情形。

不妨设 $d(x)=\gcd(f(x),f'(x))$。考虑将 $f(x)$ 的实根对应的因式分离出来，则 $f(x)$ 可表示为

$$f(x)=g(x)\prod_{i=1}^{k}(x-x_i)^{r_i}$$

其中 $x_1,x_2,\cdots,x_k$ 为 $f(x)$ 的全体实根，$r_1,r_2,\cdots,r_k$ 为对应的重根数。注意到

$$f'(x)=\sum_{i=1}^{k}\frac{r_i}{x-x_i}g(x)\prod_{j=1}^{k}(x-x_j)^{r_j}+g'(x)\prod_{j=1}^{k}(x-x_j)^{r_j}$$

则使得 $f'(x)$ 能被 $(x-x_i)^t$ 整除的最高次数 $t=r_i-1$，进而

$$d(x)=\gcd(f(x),f'(x))=h(x)\prod_{i=1}^{k}(x-x_i)^{r_i-1}$$

其中 $h(x)$ 是不被任意 $(x-x_i)$ 整除的某多项式。由 $f(x)$ 被 $h(x)$ 整除易知 $h(x)$ 无实根。

定义多项式序列 S ： $S_i = \dfrac{T_i}{d(x)}$, $0 \le i \le m$ ，则 $\forall 0 \le i < m, \gcd(S_i, S_{i+1}) = \dfrac{\gcd(T_i, T_{i+1})}{d(x)} = \dfrac{\gcd(T_0, T_1)}{d(x)} = 1$ 。这即说明 $\forall x \in \mathbb{R}$ ， $S_i(x)$ 和 $S_{i+1}(x)$ 不同时为 0。**换言之，S 中任意相邻两项不会同时取值为 0。**

此外，考虑 S 的首项，有

$$S_0 = \frac{T_0}{d(x)} = \frac{g(x)}{h(x)} \prod_{i=1}^{k} (x - x_i)$$

注意到 $g(x)$ 无实根，因而 S_0 的全体不同实根为 $x_1, x_2, \cdots, x_k$ ，**与 $f(x)$ 相同。值得注意的是，S_0 没有重根。**

由于 a 不是 $f(x)$ 的根，$a - x_i \neq 0$ 对 $1 \le i \le k$ 成立。又由 $h(x)$ 无实根知 $d(a) \neq 0$ ，从而 $T_i(a)$ 与 $S_i(a)$ 要么均为 0，要么相差一个非零常数倍，**因而有** $\mathrm{var}(T, a) = \mathrm{var}(S, a)$ 。**同理** $\mathrm{var}(T, b) = \mathrm{var}(S, b)$ 。

故只要证 S_0 在 (a, b) 上的实根数为 $\mathrm{var}(T, a) - \mathrm{var}(T, b)$ **即可。**考虑变量 x 从 a 连续变化至 b 的过程中 $\mathrm{var}(S, x)$ 的变化，不难发现仅当 x 经过某个 S_i 的根时 $\mathrm{var}(S, x)$ 可能发生变化。作如下讨论：

x 经过 S_0 的根

设 x 经过了 S_0 的根 x_0 ，有 $S_0(x_0) = f(x_0) = 0$ ，进而根据事实 2 必存在 x_0 的去心邻域 $\mathring{U}(x_0, \delta)$ 使得该邻域内 $f(x_0)$ 非零。

注意到 $f(x)f'(x) = T_0T_1 = S_0S_1(d(x))^2$ ，知 $f(x)f'(x)$ 与 S_0S_1 同号。又设 $p(x) = \dfrac{1}{2}(f(x))^2$ ，有 $p'(x) = f(x)f'(x)$ 且 $p(x)$ 在 $\mathring{U}(x_0, \delta)$ 恒正。

故根据事实 5 知存在 $\epsilon \in (0, \delta]$ 使得 $\forall x \in (x_0 - \epsilon, x_0), p'(x) = f(x)f'(x) < 0$ 且 $\forall x \in (x_0, x_0 + \epsilon), f(x)f'(x) > 0$ 。

换言之，在 x 经过 x_0 时，$S_0(x)S_1(x)$ 从负变为正，$S_0(x)$ 与 $S_1(x)$ 从异号变为同号，$\mathrm{var}(S, x)$ 减少 1。

x 经过 $S_i (1 \le i \le m)$ 的根

若 $i = m$ 且 $S_m = 0$ 则对 $\mathrm{var}(S, x)$ 没有影响，不作考虑。

否则，设 x 经过了 S_i 的根 x_0 ，有 $S_i(x_0) = 0, S_{i+1}(x_0) \neq 0, S_{i-1}(x_0) \neq 0$ 。根据 Sturm 序列的定义知存在多项式 $q(x)$ 满足

$$T_{i-1} = T_i q(x) - T_{i+1}$$

即

$$S_{i-1}d(x) = S_i d(x) q(x) - S_{i+1}d(x)$$

由于 $d(x) = \gcd(f(x), f'(x))$ 非零，有

$$\begin{aligned} S_{i-1} &= S_i q(x) - S_{i+1} \\ S_{i-1}(x_0) &= S_i(x_0)q(x_0) - S_{i+1}(x_0) = -S_{i+1}(x_0) \end{aligned}$$

这即说明 $S_{i-1}(x_0), S_{i+1}(x_0)$ 异号，进而存在 x_0 的某个邻域 $U(x_0,\delta)$ 使得 S_{i-1}, S_{i+1} 在该邻域不变号且彼此异号，同时 S_i 在该邻域非零。

不难发现当 $x \in \mathring{U}(x_0,\delta)$ 时，无论 $S_i(x)$ 符号如何，$S_{i-1}(x)$ 和 $S_{i+1}(x)$ 中必恰有一者与之同号，一者与之异号，故在该邻域内 (S_{i-1}, S_i, S_{i+1}) 对 $\operatorname{var}(S,x)$ 的贡献总为 1，$\operatorname{var}(S,x)$ 不变。

综上，S_0 在 (a,b) 的实根数恰等于 $\operatorname{var}(S,a) - \operatorname{var}(S,b)$ ，即 $f(x)$ 在 (a,b) 的实根数恰等于 $\operatorname{var}(T,a) - \operatorname{var}(T,b)$ 。 □

贡献认定：由 **吕敬一** 提供。

证明 5：牛顿迭代法的收敛性

设五次及以上多项式 $f(x)$ 有实根 α 。若 α 处有重根，由引理 2，$f'(\alpha)=0$ 。注意到 $f'(x)$ 也是多项式，由事实 2 知存在 α 的去心邻域 $\mathring{U}(\alpha,\delta)$ 满足 $\forall x \in U(\alpha,\delta),\ f'(x) \neq 0$ 。若 α 处无重根，由引理 2，$f'(\alpha) \neq 0$ ，进而由连续性可知存在邻域 $U(\alpha,\delta)$ 使得 $\inf\limits_{x\in U(\alpha,\delta)} |f'(x)| > 0$。

下面的讨论在 $U(\alpha,\delta)$ 或 $\mathring{U}(\alpha,\delta)$ 中进行，并假设 δ 足够小。设牛顿迭代的一系列猜测值为 $x_0, x_1, \cdots$ ，满足 $x_n \in U(\alpha,\delta)$ 。记 $d_n = |x_n - \alpha|$ 。

α 处无重根

根据步骤 7，运用事实 3，有

$$f(x_n) + f'(x_0)(\alpha - x_n) + \frac{f''(\xi)}{2}(\alpha - x_n)^2 = f(\alpha) = 0$$

则

$$\begin{aligned} d_{n+1} &= |x_{n+1} - \alpha| \\ &= \left| x_n - \frac{f(x_n)}{f'(x_n)} - \alpha \right| \\ &= \left| x_n - \alpha - \frac{f(x_n)(x_n-\alpha) - \frac{f''(\xi)}{2}(x_n-\alpha)^2}{f'(x_n)} \right| \\ &= \left| -\frac{f''(\xi)(x_n-\alpha)^2}{2f'(x_n)} \right| \\ &= d_n^2 \left| \frac{f''(\xi)}{2f'(x_n)} \right| \end{aligned}$$

由于 $x_n \in U(\alpha,\delta)$ ，$|f'(x_n)| \geq \inf\limits_{x\in U(\alpha,\delta)} |f'(x)| > 0$ ，故 $\dfrac{f''(\xi)}{2f'(x_n)}$ 有界。

不妨设 $\left| \dfrac{f''(\xi)}{2f'(x_n)} \right| \leq M$ ，立得：

$$d_{n+1} \leq M d_n^2 \tag{1}$$

而 $d_0 = |x_0 - \alpha| < \delta$ 很小，可以认为 $Md_0 < 1$ 。故有 $d_1 \leq Md_0^2 < d_0$ 。假设 $d_n < d_{n-1} < \cdots < d_1 < d_0$ ，则

$$d_{n+1} \leq M d_n^2 < M d_0 d_n < d_n$$

由第二数学归纳法知 d_n 单调递减，进而 $\{d_n\}$ 收敛。
不妨设 $\lim\limits_{n\to\infty} d_n = A(0 \le A < \dfrac{1}{M})$，对 (1) 式两侧取极限，得

$$A \le MA^2$$

解得 $A \in (-\infty, 0] \cup [\dfrac{1}{M}, +\infty)$ 。结合 $0 \le A < \dfrac{1}{M}$ 知 $A = 0$ 。
故牛顿迭代法在 α 附近二阶收敛。

α 处有重根

由引理 2，设重根数为 m，则 $f^m(x) \neq 0$ 。根据步骤 7 和事实 3 有：

$$\begin{aligned}
d_{n+1} &= |x_{n+1} - \alpha| \\
&= \left| x_n - \frac{f(x_n)}{f'(x_n)} - \alpha \right| \\
&= \left| x_n - \alpha - \frac{\dfrac{f^{(m)}(\alpha)(x_n-\alpha)^m}{m!} + o((x_n-\alpha)^m)}{\dfrac{f^{(m)}(\alpha)(x-\alpha)^{m-1}}{(m-1)!} + o((x_n-\alpha)^{m-1})} \right| \\
&= \left| \frac{\dfrac{(m-1)f^{(m)}(\alpha)(x_n-\alpha)^m}{m!} + o((x_n-\alpha)^m)}{\dfrac{f^{(m)}(\alpha)(x-\alpha)^{m-1}}{(m-1)!} + o((x_n-\alpha)^{m-1})} \right| \\
&= \frac{m-1}{m} d_n
\end{aligned}$$

故牛顿迭代法在 α 附近线性收敛。 □

贡献认定：由 **吕敬一** 部分参考自维基百科[4]，大部分独立提出。

这说明了算法 2 中步骤 6、7、8 的有效性：只要实根分离出的区间足够小，便能收敛至区间之内的实根。关于所需区间长度的理论分析，我们没有得到理想的结果，只得到了如下对绝大部分情况适用但实际操作中不易实施的结论：

定理 4：无重根条件下理想收敛结果的初值条件

下面假设多项式 $f(x)$ 没有重根。
假设已知一区间 $[l, r]$ 使得存在恰好一个 $\alpha \in [l, r]$ 是 $f(x)$ 的实根。根据定理 2，$f'(\alpha) \neq 0$ 。**假设**存在 α 的一个邻域 $U(\alpha, \delta)$ 使得邻域内 $f'(x), f''(x), f'''(x)$ 均不变号，即邻域内 $f(x), f'(x), f''(x)$ 单调。
当满足

$$\begin{cases} r - l < \dfrac{4\min\{|f'(l)|, |f'(r)|\}}{\max\{|f''(l)|, |f''(r)|\}} \\ [l, r] \subset U(\alpha, \delta) \end{cases}$$

时，选取初值 $\dfrac{l+r}{2}$，牛顿迭代将收敛至 α 。

[4] https://en.wikipedia.org/wiki/Newton%27s_method

贡献认定：由 **吕敬一** 提供。

证明 6

我们只要证明 $Md_0 < 1$ 即可（M 的定义见证明 5）。
由于 $\alpha \in [l, r]$ ，有 $d_0 = \left|\dfrac{l+r}{2} - \alpha\right| \leq \dfrac{r-l}{2}$ 。
而根据 $f'(x), f''(x)$ 的单调性，不难得到

$$\left|\frac{f''(\xi)}{2f'(x_n)}\right| \leq \frac{\max\{|f''(l)|, |f''(r)|\}}{2\min\{|f'(l)|, |f'(r)|\}}$$

令 $M = \dfrac{\max\{|f''(l)|, |f''(r)|\}}{2\min\{|f'(l)|, |f'(r)|\}}$ ，立得

$$\begin{aligned} Md_0 &= \frac{\max\{|f''(l)|, |f''(r)|\}}{2\min\{|f'(l)|, |f'(r)|\}} \cdot \frac{r-l}{2} \\ &< \frac{\max\{|f''(l)|, |f''(r)|\}}{2\min\{|f'(l)|, |f'(r)|\}} \cdot \frac{2\min\{|f'(l)|, |f'(r)|\}}{\max\{|f''(l)|, |f''(r)|\}} \\ &= 1 \end{aligned}$$

□

贡献认定：由 **吕敬一** 提供。

由于假设条件不一定能满足且难以验证，这个结论缺乏实用性，我们在实际算法设计中选择了手动估计区间长度阈值 L 。

另外，算法 2 的步骤 6 和步骤 9 都涉及了根据当前点的多项式值和导数值估计当前的误差。现形式化地补充该定理及其证明：

定理 5：微小误差估计

设 x_0 是多项式 $f(x)$ 的一个实根，$x - x_0$ 充分小且 $f'(x) \neq 0$ ，则

$$x - x_0 \approx \frac{f(x)}{f'(x)}$$

贡献认定：由 **吕敬一** 提供。

证明 7：微小误差估计

运用事实 3，在 x 处对 $f(x)$ 泰勒展开，得

$$f(x_0) = f(x) + f'(x)(x_0 - x) + o(x_0 - x)$$

注意到 x_0 是 $f(x)$ 的根，$f(x_0)=0$，即有

$$x-x_0=\frac{f(x)}{f'(x)}+o(1)$$

由 $x-x_0$ 极小，忽略 $o(1)$ 项即证。 □

贡献认定：由 **吕敬一** 提供。

4.4 数值验证

运用 7.1 节的方法对算法 2 进行了数值验证，注意到三条主要问题：

(1) 在利用 Sturm 定理计算某些区间所含不同实根数时，有时会出现错误，导致最后出现丢根的情况。极端情况下甚至可能将有实根的多项式方程判定为没有实根。经过分析，这种情况几乎只出现在包含重根的区间上。

例子 3

当多项式为例子 1 中的第一个 13 次多项式时，算法 2 得到的实根为

$$\begin{aligned}
&-1.43821581000948595097099769191117957234382629394531 25\\
&-0.91065489381036457405826922695268876850605010986328125\\
&-0.84445284233852291322364180814474821090698242187 5\\
&-0.74300096212108845961097358667757362127304077148437 5\\
&-0.60024708012588556815813944922410883009433746337890625\\
&-0.05708548449600848312401879525168624240905046463012695312 5\\
&0.01878219798816096325344560113990155514329671859741210937 5\\
&0.01878219798816096325344560113990155514329671859741210937 5\\
&0.10684716538494837267947445980098564177751541137695312 5\\
&0.49008462517497325405813057841442059725522994995117187 5\\
&0.99232778605807259175719536870019510388374328613281 25
\end{aligned}$$

与例子 1 中的精确实根对比，两次重根的 0.197466 被遗漏了。

当多项式为

$$(x-1.07820537831261886729805610229959711432456970214843 75)^8$$

时，显然有实根 1.07820537831261886729805610229959711432456970214843 75，但算法 2 返回结果为不存在实根。

贡献认定：由 **吕敬一** 提供。

(2) 对包含一个重根次数比较多的实根的区间进行牛顿迭代法求解时，无论如何增加迭代次数，根的精度都非常低，无法达到要求。

例子 4

当多项式为例子 2 中的多项式时，算法 2 求出的实根为 0.14580241182468195448862502416886854916810989379882812 5，与 0.144122 的误差很大，超过了预设的容差 10^{-6} 。

贡献认定：由 **吕敬一** 提供。

(3) 在步骤 9 中计算重根数时，有时计算结果并不准确。初步分析是因为没有控制好高阶导数值的容差，或是不可避免的浮点运算精度丢失导致。

例子 5

当多项式为

$$(x - 0.42644722833061793343389922483765985816717147827148437 5)^2$$

时（精确实根为 0.42644722833061793343389922483765985816717147827148437 5，重根两次），算法 2 返回了 0.42644722456916933639448075155087281018495559692382812 5，但给出重根数为 1。

贡献认定：由 **吕敬一** 提供。

5 算法设计：基于笛卡尔符号规则的实根分离

5.1 前提准备

5.1.1 重根的处理

根据前述算法的测试结果，多项式的重根会严重影响实根分离算法的表现。为此，提出了如下子问题:

问题 3: 平方因子分解

对于有重根的多项式，其要转化为没有重根的多项式以方便计算。
亦即，要找到一个新多项式，满足两个多项式零点相同，但新多项式方程没有重根。

贡献认定：由 **潘佳奇** 提供。

经过检索与学习，提出了如下算法解决该子问题：

算法 3: Yun's algorithm

对于要分解的非零多项式，假设最终 k 重根 $(x-a)^k$ 被分解到多项式 $a_k(x)$ 里，设

$$f(x) = a_1(x)a_2^2(x)a_3^3(x)\cdots a_k^k(x).$$

则有初始化：

$$\begin{aligned} b_0(x) &:= f(x); \\ d_0(x) &:= f'(x); \\ i &:= 0. \end{aligned}$$

以及迭代过程：重复以下过程直到 $b_i = 1$。

$$\begin{aligned} a_i(x) &:= \gcd(b_i(x), d_i(x))); \\ i &:= i+1; \\ b_i(x) &:= \frac{b_{i-1}(x)}{a_{i-1}(x)}; \\ c_i(x) &:= \frac{d_{i-1}(x)}{a_{i-1}(x)}; \\ d_i(x) &:= c_i(x) - b_i'(x). \end{aligned}$$

计算出的 $a_i(x)$ 即为要求的结果。

贡献认定：由 **潘佳奇** 摘自维基百科 [5]。

证明 8: Yun's algorithm

对于 f 的分解：

$$f(x) = a_1(x)a_2^2(x)a_3^3(x)\cdots a_k^k(x).$$

令 $a_0(x) = a_2^1(x)a_3^2(x)\cdots a_k^{k-1}(x)$，则对于 $f(x)$，有

$$f(x) = a_0(x)a_1(x)\cdots a_k(x)$$

对于 $f'(x)$，有

$$f'(x) = a_0(x)\sum_{i=1}^{k} ia_i'(x)a_1(x)\cdots a_{i-1}(x)a_{i+1}(x)\cdots a_k(x)$$

与前置约定中的『无重根多项式与其导函数最大公因子为 1』同理，因为 $a_1(x)a_2(x)\cdots a_k(x)$ 没有重根，所以有 $gcd(\frac{f(x)}{a_0(x)}, \frac{f'(x)}{a_0(x)}) = 1$。

所以 $\gcd(f(x), f'(x)) = a_0(x)$。

[5] https://en.wikipedia.org/wiki/Square-free_polynomial

于是令 $b_1(x)=\dfrac{f(x)}{a_0(x)}, c_1(x)=\dfrac{f'(x)}{a_0(x)}, d_1(x)=c_1(x)-b_1'(x)$，则

$$d_1(x)=a_1(x)\sum_{i=2}^{k}(i-1)a_i'(x)a_2(x)\cdots a_{i-1}(x)a_{i+1}(x)\cdots a_k(x)$$

再次引用『多项式与其导函数公因子』的引理，于是有 $\gcd(d_1(x),b_1(x))=a_1(x)$
于是由迭代过程，有

$$b_2(x)=a_2(x)\cdots a_k(x),$$

$$c_2(x)=\sum_{i=2}^{k}(i-1)a_i'(x)a_2(x)\cdots a_{i-1}(x)a_{i+1}(x)\cdots a_k(x).$$

由归纳可证，对于任意 $\xi\in\mathbb{N}^+,\xi\le k$，有

$$a_\xi=\gcd(b_\xi,d_\xi)$$

$$b_\xi(x)=a_\xi(x)\cdots a_k(x),$$

$$c_\xi(x)=\sum_{i=\xi}^{k}(i-\xi+1)a_i'(x)a_\xi(x)\cdots a_{i-1}(x)a_{i+1}(x)\cdots a_k(x).$$

即迭代过程每步提取出了正确的 $a_i(i\ge 1)$。 □

贡献认定：由 **潘佳奇** 证明。

5.1.2 实根分离

解决重根问题之后，再给出如下定义与定理，以实现更高效的实根分离：

定义 6：实根数

对于多项式 $f(x)=a_nx^n+\cdots+a_1x+a_0$，定义实根数 $\varrho(f(x))$ 为 $f(x)$ 的实根数（重根计算多次，下同），$\varrho_+(f(x))$ 为 f 的正实根数，$\varrho_-(f(x))$ 为 f 的负实根数。$\varrho_{l,r}(f(x))$ 为区间 $(l,r]$ 之间 f 的实根数。

贡献认定：由 **潘佳奇** 提供。

定义 7：多项式的变号数

对于多项式 $f(x)=a_nx^n+\cdots+a_1x+a_0$，定义变号数 $\mathrm{var}(f)$ 为序列 $[a_n,a_{n-1},\ldots,a_0]$ 的变号次数，即忽略 0 的情况下，正负号变了多少次。

形式化地讲，$\mathrm{var}(f)$ 就是有多少对二元组 (i,j) 满足 $i<j,a_ia_j<0$，且 $\forall i<k<j,a_k=0$。

贡献认定：由 **潘佳奇** 提供。

定理 6: 笛卡尔符号规则（Descartes' rule of signs）

对于任意多项式 $f(x)$，有 $\operatorname{var}(f) \geq \varrho_+(f)$，且 $\operatorname{var}(f) - \varrho_+(f)$ 一定是偶数。

贡献认定：由 **潘佳奇** 提供。

Descartes' rule of signs 只能求出正根的界，对于负根，令 $g(x) = f(-x)$ 即可。

定理 7

$$\operatorname{var}(f) = 0 \iff \varrho_+(f) = 0,$$
$$\operatorname{var}(f) = 1 \implies \varrho_+(f) = 1.$$

贡献认定：由 **潘佳奇** 提供。

定理 8: Budan's theorem

对于多项式 f，取 $a, b \in \mathbb{R}$，$a < b$，则有：

$$\operatorname{var}(f(x+a)) \geq \operatorname{var}(f(x+b))$$
$$\varrho_{a,b}(f) \leq \operatorname{var}(f(x+a)) - \operatorname{var}(f(x+b))$$

并且 $\operatorname{var}(f(x+a)) - \operatorname{var}(f(x+b)) - \varrho_{a,b}(f)$ 一定是偶数。

贡献认定：由 **潘佳奇** 提供。

5.2 算法流程

5.2.1 实根分离算法-VAS 法

算法 4: 实根分离算法-VAS 法

令没有重根的多项式为 $f(x)$，不妨讨论只求出正根的情况。

维护 S 为算出区间的集合。同时使用 Descartes' rule of signs 和 Budan's theorem 来计算正实根以及区间实根的存在性。

初始化时 $S = \emptyset$。

步骤 1. 先判断特殊情况，当 $f(x)$ 不存在正实数根的时候，将 S 作为返回值结束算法。

步骤 2. 当 $f(0) = 0$ 时，说明 0 是根，将 0 分割出来并且将多项式除以 x，并将 $\{0, 0\}$ 加入 S，返回**步骤 1**。

步骤 3. 当 $f(x)$ 只有一个根的时候，已经达成实数分离的目标，将 $\{(0, +\infty)\}$ 作为结果

返回。

步骤 4. 此时有一个根据多项式系数得到的数 b，粗略地标定根的界。（在此算法里设为 1）。

步骤 5. 对多项式做代入 $x = x' + b$，将 $(b, +\infty)$ 映射到 $(0, +\infty)$，并递归求出 $f(x')$ 的实根分割区间后，还原出对应 $(b, +\infty)$ 上的区间集合 S'，并令 $S \leftarrow S \cup S'$。

步骤 6. 如果在 $(0, b)$ 中有实根，则代入 $x = \dfrac{b}{1 + x'}$，将 $(0, b)$ 映射到 $(0, +\infty)$，并递归求出 $f(x')$ 的实根分割区间后，还原出对应 $(b, +\infty)$ 上的区间集合 S'，并令 $S \leftarrow S \cup S'$。

步骤 7. 将 S 作为返回值返回。

贡献认定：由 **潘佳奇** 摘自 Polynomial Real Root Isolation Using Vincent’s Theorem of 1836 第 143 页，有所改编。

5.2.2 实根分离算法-二分法

算法 5: 实根分离算法-二分法

算法接受一个（不含重根的）多项式 $f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$，并求解其在 $[0, 1)$ 上的实根分割区间。

首先判定多项式常数项 a_0 是否为 0。若是，则 $x = 0$ 是多项式的零点，并将多项式除以 x，直至常数项不为 0。

对于常数项不为 0 的 n 次多项式 $f(x)$，求出 $g(x) = (x + 1)^n \cdot f(\dfrac{1}{x + 1})$。

假设 $f(x)$ 在 $(0, 1)$ 上有某实根 x_0，则 $g(\dfrac{1}{x_0} - 1) = 0$。

同样地，若 $g(x)$ 在 $(0, +\infty)$ 有某实根 x_1，则 $f(\dfrac{1}{x_1 + 1}) = 0$。

亦即，$f(x) = 0$ 在 $(0, 1)$ 上的实根与 $g(x) = 0$ 在 $(0, +\infty)$ 上的实根一一对应。故可对 $g(x)$ 应用 Descartes' rule of signs 以判定 $f(x) = 0$ 在 $(0, 1)$ 上的实根数量。

若 Descartes' rule of signs 确定 $f(x) = 0$ 在 $(0, 1)$ 上只有零或一个实根，则算法结束。否则，构造 $g_0(x) = 2^n f(\dfrac{x}{2})$，$g_1(x) = 2^n f(\dfrac{x + 1}{2})$，它们在 $[0, 1)$ 上的实根分别对应了 $f(x) = 0$ 在 $[0, \dfrac{1}{2}), [\dfrac{1}{2}, 1)$ 上的实根，递归求解即可。

步骤 1. 初始化队列 q,包含一个三元组 $(c, k, f(x))$,代表当前求解的多项式 $f(x)$ 在 $[0, 1)$ 上的根对应最初函数 $[\dfrac{c}{2^k}, \dfrac{c + 1}{2^k})$ 上的根。记输入的多项式为 $f_0(x)$，最初的三元组为 $(0, 0, f_0(x))$。

步骤 2. 如果队列 q 为空则结束算法。否则，从队列 q 中取出一个三元组 $(c, k, f(x))$，判断 $f(x)$ 常数项是否为 0，如果是，则将 $[\dfrac{c}{2^k}, \dfrac{c}{2^k}]$ 置入答案序列中，并将 $f(x)$ 整体除以 x，重复执行这一步直到常数项不为 0。

步骤 3. 计算 $V = \mathrm{var}((x + 1)^n \cdot f(\dfrac{1}{x + 1}))$。

步骤 4. 若 $V = 0$，丢弃该区间并转到 **步骤 2**。

步骤 5. 若 $V = 1$，将该区间 $[\dfrac{c}{2^k}, \dfrac{c + 1}{2^k}]$ 加入到答案序列并转到 **步骤 2**。

步骤 6. 若 $V > 1$，二分该区间，将三元组 $(2c, k + 1, 2^n f(\dfrac{x}{2})), (2c + 1, k + 1, 2^n f(\dfrac{x + 1}{2}))$ 加入队列 q，再转到 **步骤 2**。

贡献认定：由 **叶隽希** 提供。

5.3 理论证明

证明 10: 笛卡尔符号规则（Descartes' rule of signs）

考虑多项式 $f(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$。若 $a_n \neq 1$，不妨将多项式各项系数除以 a_n，得到的新多项式首项系数为 1，且实根数与变号数均未改变。若 $a_0 = 0$，则一定存在某个 x^m，将多项式除以 x^m 后得到新的多项式常数项不为 0。新的多项式**正实根数**与变号数均未改变。故不妨假设 $a_n = 1$ 且 $a_0 \neq 0$。

引理 3

若 $a_0 < 0$，$\mathrm{var}(f(x))$ 为奇数。若 $a_0 > 0$，$\mathrm{var}(f(x))$ 为偶数。

注意到首项系数为正，则若变号奇数次最后为负，变号偶数次最后为正。

引理 4

若 $a_0 < 0$，$\varrho_+(f(x))$ 为奇数。若 $a_0 > 0$，$\varrho_+(f(x))$ 为偶数。

比较直观的感受是，将函数图像在坐标轴上画出，由于首项系数是 1，那么多项式在 x 轴正方向足够远处一定在轴上方（即函数值为正）。

而 a_0 的正负代表了函数图像在 y 轴附近在坐标轴上方或者下方。

注意到每有一个正实根出现，函数图像必定会穿过 x 轴一次（暂且不考虑重根），则函数值正负必定改变一次。

故同样，若有奇数个零点则 a_0 与 $a_n = 1$ 异号，若有偶数个零点则 a_0 与 $a_n = 1$ 同号。

具体证明可以考虑应用归纳法，每次给多项式乘上一个因子 $(x - p)$，p 如果为正根便会改变 a_0 的正负性，如果为负根便不影响（根为零的情况不存在）。

引理 5

$$\mathrm{var}(f(x) \cdot (x - p)) > \mathrm{var}(f(x))(p > 0)$$

在给出证明之前，注意到对于 $p > 0$，有 $\varrho_+(f(x) \cdot (x - p)) = \varrho_+(f(x)) + 1$

而对于任意一个多项式 $f(x)$，其总可以分解为 $f(x) = g(x) \cdot (x - x_1) \cdot (x - x_2) \ldots (x - x_m)$ 的形式，其中 $x_k > 0$ 且 $g(x) = 0$ 无正实根。（即提取出所有正根）

根据定义，$\varrho_+(f(x)) = m$，而 $\mathrm{var}(g(x)) \geq 0$，由引理归纳可得 $\mathrm{var}(f(x)) \geq m = \varrho_+(f(x))$。

又由前两条引理可知 $\mathrm{var}(f(x))$ 与 $\varrho_+(f(x))$ 奇偶性相同。综合可知定理成立。 □

证明 10

考虑对 $\mathrm{var}(f(x))$ 产生贡献的每一个二元组 (i,j)，它满足 $i<j, a_ia_j<0, \forall i<k<j, a_k=0$。

注意到 $f(x)\cdot(x-p)=\cdots+(a_i-pa_{i+1})x^{i+1}+\ldots$。

若 $a_i<0$，则 $a_{i+1}\geq 0$，$a_i-pa_{i+1}<0$

若 $a_i>0$，则 $a_{i+1}\leq 0$，$a_i-pa_{i+1}>0$

对于每一个二元组 $(i_p,j_p)(1\leq p\leq \mathrm{var}(f(x)))$，当 $p>1$ 时显然有 $a_{i_{p-1}}a_{i_p}<0$，故相应的，设 $f(x)\cdot(x-p)=a'_{n+1}x^{n+1}+\cdots+a'_1x+a'_0$，则 $a'_{i_{p-1}+1}a'_{i_p}<0$。

而又有 $a_0a_{i_1}>0, a_na_{i_{\mathrm{var}(f(x))}}>0, a'_0a_0<0$，故存在序列 $(a'_0, a_{i_1+1}, a_{i_2+1}, \ldots, a_{i_{\mathrm{var}(f(x))}}, a'_n)$，这一序列中任意相邻两项均为异号，而该序列长度为 $\mathrm{var}(f(x))+2$，故显然 $\mathrm{var}(f(x)\cdot(x-p))\geq(\mathrm{var}(f(x))+2)-1$。

□

贡献认定：由 **叶隽希** 提供。

由于朴素的牛顿法要求初始点离根较近，所以需要二分法来确定包含根的较小区间。下面提出区间运算的牛顿迭代。使用区间运算可以规避这个二分。

定义 8: 区间算术

定义区间的加减乘和数除为：

$$[l_1,r_1]+[l_2,r_2]=[l_1+l_2,r_1+r_2]$$

$$[l_1,r_1]-[l_2,r_2]=[l_1-r_2,r_1-l_2]$$

$$a=l_1r_1; b=l_1r_2; c=l_2r_1; d=l_2r_2$$

$$[l_1,r_1]\times[l_2,r_2]=[\min\{a,b,c,d\},\max\{a,b,c,d\}]$$

贡献认定：由 **潘佳奇** 提供。

算法 6: 区间牛顿法

不妨设初始区间只包含正数。

输入：精度 ε，求根区间 $[l,r]$，函数 f。

输出：声明不存在根或者输出满足精度要求的一个根。

令 $g(m,[l,r])=[\min(m-\frac{f(m)}{d_l}, m-\frac{f(m)}{d_r}), \max(m-\frac{f(m)}{d_l}, m-\frac{f(m)}{d_r})]$。

步骤 1. 如果 $l>r$ 说明无解。退出迭代。

步骤 2. 如果 $r-l<\varepsilon$，表示已经区间收敛到要求范围内。

步骤 3. 对于当前区间 $[l,r]$，通过区间算术算出 $[d_l,d_r]=f'([l,r])$。

步骤 4. 令 $m = \dfrac{l+r}{2}$。

步骤 5. 如果 $0 \notin [d_l, d_r]$，则令新区间为 $[l', r'] = [l, r] \cap g(m, [d_l, d_r])$，返回步骤 1 继续迭代。

步骤 6. 此时 $0 \in [d_l, d_r]$，则对 $[l, r] \cap g(m, [d_l, 0))$ 和 $[l, r] \cap g(m, (0, d_r])$ 产生的两个区间进行递归。

贡献认定：由 **潘佳奇** 摘自 julia 文档[6]。

证明 11：区间牛顿法的正确性

对于区间 $x_n = [l, r]$, $m = \dfrac{l+r}{2}$。则 $x_{n+1} = m - \dfrac{f(m)}{f'(x_n)}$。

对于 f 存在 x^* 使得 $f(x^*) = 0$，归纳证明 $x^* \in x_n$。假设对于 n 成立，不妨设 $x^* < m$，存在 $\xi \in (x^*, m) \in (l, r)$，使得 $f'(\xi) = \dfrac{f(m) - f(x^*)}{m - x^*}$，即 $x^* = m - \dfrac{f(m)}{f'(\xi)}$，所以 $x^* \in x_{n+1}$。

贡献认定：由 **潘佳奇** 提供。

当导函数取值不存在零点的时候，该算法二阶收敛至实根，并且必定收敛至实根分离所得初始区间之内的唯一实根。

算法 7：VAS + 区间牛顿迭代

步骤 1. 使用 Yun's Algorithm 进行平方因子分解。

步骤 2. 使用 VAS 算法进行实根分离。

步骤 3. 使用区间牛顿迭代计算每个区间中根。

贡献认定：由 **潘佳奇** 提供。

由于实根估计并不能准确地到达下界，复杂度和二分法相同。

对于区间牛顿迭代，导函数取值不为 0 才能二阶收敛。然而由于多项式区间取值的上下确界不好计算，所以使用了区间算术，这导致很大一部分情况只有一阶的收敛速度。

这个算法在运行效率上没有很大提升，在实际的实现中表现并不好。但是在运算过程中全部使用了区间，避免了牛顿迭代过程中发散的可能。

6 对各算法的进一步优化

6.1 精度优化

不难注意到，提升算法精度的最大瓶颈在于 double 类型本身无法精确表达几乎全体实数。考虑到我们的算法中涉及的运算均对有理数封闭，我们在各个算法中运用了 gmp 高精度有理数和整数库进行计算，并利用 gmp 库生成测试用多项式，这理论上允许我们做到对任意精度的根进行数值求解。

[6] https://juliaintervals.github.io/pages/explanations/explanationNewton/

6.2 效率优化

高精度有理数会带来巨大的算术开销。目前的三个算法的主要区别在于实根分离的手段不同，而在最终求解实根时全部采用牛顿迭代法。牛顿迭代法在大部分条件下二阶收敛至一个实根，其 gmp 有理数的分子分母也二阶增长，这意味着单步迭代的开销会随迭代次数极速上升，严重限制了求解精度的提升。

我们在对答案不影响的情况下进行了粗略的约分，降低计算开销。

算法 8: 粗略约分

输入：正有理数的最简形式 $\frac{a}{b}$，要求精度 $2^{-\varepsilon}$，向下近似还是向上近似。
输出：粗略约分后的有理数。

步骤 1. 找出最小的 k 使得 $2^k \geq \frac{b}{\lfloor \frac{a}{b} \rfloor + 1}$。

步骤 2. 如果 $k \leq \varepsilon + C$，则直接输出 $\frac{a}{b}$。其中 C 是一个不低于 2 的常数。

步骤 3. 令 $M = 2^{k-\varepsilon-C}$，令 $a' = \frac{a}{M}, b' = \frac{b}{M}$。

步骤 4. 如果要求向下近似，输出 $\frac{a'}{b'+1}$，否则输出 $\frac{a'+1}{b'}$。

贡献认定：由 **潘佳奇** 提供。

证明 12: 粗略约分

设 $a = a'M + c, b = b'M + d$，则 $0 \leq c, d < M$。
则有

$$\begin{aligned}
&\left|\frac{a'M+c}{b'M+d} - \frac{a'}{b'}\right| \\
=&\left|\frac{(a'M+c)b' - (b'M+d)a'}{(b'M+d)b'}\right| \\
=&\left|\frac{cb'-da'}{(b'M+d)b'}\right| \\
\leq&\frac{\max(b',a')M}{b'^2M} \\
=&\frac{\max(1,\frac{a'}{b'})}{b'} \\
<&\frac{\max(1,\frac{a+1}{b})}{b'} \\
<&\frac{1}{2^{\varepsilon+1}}
\end{aligned}$$

因为 $\frac{1}{b'} < \frac{1}{2^{\varepsilon+1}}$，所以 $\frac{a'+1}{b'}, \frac{a'}{b'+1}$ 都在容差范围内。
并且有 $\frac{a'}{b'+1} \leq \frac{a}{b} \leq \frac{a'+1}{b'}$。

□

贡献认定：由 **潘佳奇** 提供。

经过验证，使用粗略约分大幅度地加快了牛顿迭代法的速度。

在 Sturm-Newton 方法的实际验证过程中，注意到牛顿迭代的开销过大（实际上粗略约分对该算法的表现改善程度并不大——），故将步骤 4 的阈值 L 从 0.1 调整为 $L = \min\{\sqrt{e}, 10^{-2}\}$，增加了实根分离的精度，减少了迭代次数，平衡了两部分开销，优化效果明显。

贡献认定：由 **吕敬一** 提供。

7 数值验证与算法改进

7.1 根正态分布的多项式

为了验证数值算法所求根的准确性，我们考虑先生成多项式的精确实根，再根据实根构造出对应的多项式，最后将所求出的实根与精确实根比较。在多项式的生成、输入输出中，应当特别注意控制系数的误差。以下例子说明了微小误差可能带来的影响：

例子 6: Wilkinson 多项式

令 $f(x) = (x-1)(x-2)\dots(x-20)$，则有 x^{19} 的系数为 -210。假设这一位在系数计算的时候，有误差 2^{-50}，即系数 x^{19} 的系数变为了 $-210 - 2^{-50}$，则对应多项式的根会有将近 10^{-6} 的绝对误差。

以下根由 matlab 计算，并使用 WolframAlpha 的解方程功能进行验算，保留 30 位有效数字。

$$
\begin{aligned}
x_1 &= 1.0 \\
x_2 &= 2.00000000000000000000000007273 \\
x_3 &= 2.99999999999999999999854887259 \\
x_4 &= 4.00000000000000000194477430517 \\
x_5 &= 4.99999999999999946021669717504 \\
x_6 &= 6.00000000000005173499077404847 \\
x_7 &= 6.99999999997741863637264428 71 \\
x_8 &= 8.00000000053020335215395239 65 \\
x_9 &= 8.99999999254527454249149590 8 \\
x_{10} &= 10.00000006744878904790704052 8
\end{aligned}
$$

$$x_{11} = 10.999999958748937425345988426 6$$
$$x_{12} = 12.000000176306266307312631226$$
$$x_{13} = 12.999999462156173662606196 1$$
$$x_{14} = 14.000001183914676207040977459 2$$
$$x_{15} = 14.999998117893056943804898174 8$$
$$x_{16} = 16.000002138301745240626496067 3$$
$$x_{17} = 16.999998308549882630936118710 7$$
$$x_{18} = 18.000000884257930443720446568 5$$
$$x_{19} = 18.999999725540800164118606122 7$$
$$x_{20} = 20.000000038280312107622753058 4$$

贡献认定：由 **潘佳奇** 提供。

经过考虑，在生成测试集时，使用正态分布随机根的数据。

算法 9: 生成测试多项式

对于多项式的度数，随机一个在 $[1, 20]$ 上的整数，记为 d。以 80% 的概率生成带有重根的多项式，如果没有重根，令 $d' = d$，否则使用标准正态分布随机一个实数 p，并令 $d' = \max(1, d - \lceil \min(|p|, 1)d \rceil)$。对于前 d' 个根，使用标准正态分布获取一个值，对于后 $d - d'$ 个根，分别在前 d 个根中等概率随机选取一个根作为自身的值。获取根后，将根排序，并使用多项式乘法计算出这些根对应的多项式。在输出的数据中，所有浮点数保留 64 位有效数字。

时间复杂度：$O(d^2)$。

贡献认定：由 **潘佳奇** 提供。

猜想 1

对于能够保证生成的多项式精度情况较好，使算法在读入的时候带来的误差对根影响较小。

贡献认定：由 **潘佳奇** 提供。

其中根据正态分布获取随机数的算法如下：

算法 10: Marsaglia 极坐标法

设正态分布的均值为 μ，方差为 σ^2。

步骤 1. 不断地随机二元组 (s, v)（其中 u, v 均服从 $(-1, 1)$ 上的均匀分布），直到 $0 < u^2 + v^2 < 1$。

步骤 2. 令 $s = u^2 + v^2, t = \sqrt{\dfrac{-2 \ln s}{s}}$，返回两个数 $\mu + \sigma t\mu, \mu + \sigma tv$ 作为随机数结果。

时间复杂度：$O(1)$。

贡献认定：由 **潘佳奇** 提供。

7.2 形态更全面的多项式

由于在后续算法优化中使用了 gmp 库，我们用 gmp 库重写了测试方法，包含在 testlib.cpp 和 testlib.h 中。目前的测试方法支持三种功能：随机生成指定数量的根整数、根为实数的多项式，以及复用上一次生成过的多项式。生成的多项式数量 nTests 和容差要求 e 需在代码中修改，而生成模式由 cin 输入。

算法 11：测试方法（简略）

输入：一个整数 mode，代表测试模式。0 为复用，1 为整根多项式，2 为实根多项式。

步骤 1. 若 mode $= 0$ ，则从 poly.rawin 中读取上一次生成的精确原始多项式，以 gmp 高精度有理数形式存储并输出。否则，进入步骤 2。

步骤 2. 随机生成一个 $[2, 19]$ 的整数 n ，代表多项式的次数。随后按照 mode 要求，在 $[-10, 10]$ 范围内随机生成 n 个实根，或是在 $[-100, 100]$ 范围内随机生成 n 个不等整根，将其存储为 gmp 高精度有理数格式。

步骤 3. 把生成的根展开为多项式，获得 gmp 高精度有理数格式的多项式系数。将 unsigned long long 格式的根和系数输出至 poly.raw 中。

步骤 4. 将多项式分别传入三种不同实现的 gb_solve() 方法，获得三组不同结果。将三组结果之间分别与生成的标准根比较，输出超出容差的根组数；再将三组结果彼此互相比较，输出超出容差的根组数。

步骤 5. 如果希望下一次测试复用某一组数据，则在 poly.raw 中找到待用数据，复制进 poly.rawin 中。

步骤 6. 若当前测试多项式数量已达到 nTests，则退出；否则跳转至步骤 2。

算法结束。

贡献认定：由 **叶隽希** 提供。

7.3 测试与改进

读入误差

初步测试之后，注意到的首要问题是三种不同算法所求出实根彼此的误差均在所给定容差范围内，但和标准根对比，其均有相同数量的根超出容差。

例子 7：读入误差 1

在求解如下多项式方程时

$$
\begin{aligned}
&-601.87247977829235878743929788470268249511718 75\\
&-372.28844745806571836510556749999523162841796875x\\
&+202.58729945165558206099376548081636428833007812 5x^2\\
&+52.03053344897322318729493417777121067047119140625x^3\\
&-16.69698959372401603218349919188767671585083007812 5x^4\\
&+x^5
\end{aligned}
$$

测试程序提示标准根与算法 2, 4 ,5 生成的实根各有两个误差超过容差，而算法 2, 4, 5 生成的实根彼此没有误差超过容差。具体而言，标准根应当是

$$
\begin{aligned}
&-2.537151028527214524643795812153257429599761962890625 00\\
&-1.181240419666571561663204192882403731346130371093750 00\\
&2.510420324399699421036302737775258719921112060546875 00\\
&8.565239496195045987292360223364084959030151367187500 00\\
&9.339721221323058486518675636034458875656127929687500 00
\end{aligned}
$$

算法 2, 4, 5 给出的实根是

$$
\begin{aligned}
&-2.537151028527214968733005662215873599052429199218750 00\\
&-1.181240419666571783707809117913711816072463989257812 50\\
&2.510420324399699421036302737775258719921112060546875 00\\
&8.565239496195074409001790627371519804000854492187500 00\\
&9.339721221323030064809245232027024030685424804687500 00
\end{aligned}
$$

贡献认定：由 **吕敬一** 提供。

对 Sturm-Newton 算法调整了迭代终止条件，强制增加了迭代次数，发现输出结果和标准答案的误差完全没有改变，基本说明 gb_solve() 的求解过程并未导致过大的误差。

由于数据生成中采用了无误差的高精度数，初步判断该误差来源是将生成数据输入 gb_solve() 时，由于 gb_solve() 要求输入多项式系数为 double 类型，发生了丢失精度的类型转换。根据先前 Wilkinson 多项式的例子，在次数较高时，多项式系数的微小扰动也会带来实根明显的误差。考虑到 double 类型丢失精度不可避免，故认为这并不属于算法涉及的明显缺陷。

例子 8：读入误差 2

在 mode = 1 下生成了具有如下 30 个实根的多项式：

$$
-97, -88, -86, -84, -83, -82, -77, -59, -55, -42, -40, -39, -34, -28, -22, -14, -11, 3, 24, 28, 47, 52, 59, 64, 66, 73, 80, 88, 89, 91
$$

注意到实根中有 10 的倍数（40 和 80），展开后的多项式常数项应当为 10 的倍数。然而输入数据显示 double 类型的多项式系数中常数项为

$$-113260460306063369995491810216948300274714967801856$$

显然发生了矛盾。

贡献认定：由 **吕敬一** 提供。

这个例子表明 double 类型存储多项式系数对于过高的次数或者过大的系数也会产生显著的读入误差，且很难避免。

舍入误差

当我们将测试数据调整为度数不超过 10，精确根为绝对值不超过 100 的整数，容差设置为 10^{-15} 时，发现算法的输出出现了问题。

例子 9：舍入误差

在求解如下多项式方程时：

$$108 - 21x + x^2 = 0$$

算法 5 给出了 `double` 类型下的结果：

$$x_1 = 8.999999999999998223643160599749535322189331054687 5, x_2 = 12$$

误差超过了给定的 10^{-15} 。

贡献认定：由 **吕敬一** 提供。

由于之前测试之中 10^{-15} 的容差在长期测试下并未出现问题，这一组简单数据却超出了容差，这引起了我们的注意。

起初怀疑是算法精度不足，但在同时增加二分层数、牛顿迭代次数、粗略约分的精度后，算法运行时间虽已数十倍地增长，但给出的结果并未改变。

在经历了更多的调试后我们偶然发现，算法给出的 x_1 与实际的根 9 在 IEEE 754 标准的 `double` 意义下实际是相邻的。

代码 1

```
#include <iostream>
#include <iomanip>
#include <bitset>
unsigned long long to_ull(double x) {
    return *reinterpret_cast<unsigned long long*>(&x);
}
```

```
std::bitset<64> to_bitset(double x) {
    return std::bitset<64>(to_ull(x));
}
double to_double(unsigned long long x) {
    return *reinterpret_cast<double*>(&x);
}
void print(double x) {
    std::cout << to_bitset(x) << std::endl;
}
int main() {
    std::cout << std::fixed << std::setprecision(50);
    double x = 9;
    double y = to_double(to_ull(x) - 1);
    std::cout << x << '\n'; print(9);
    std::cout << y << '\n'; print(y);
}
```

这一段代码给出的输出为

```
9.00000000000000000000000000000000000000000000000000
0100000000100010000000000000000000000000000000000000000000000000
8.99999999999999822364316059974953532218933105468750
0100000000100001111111111111111111111111111111111111111111111111
```

随即我们意识到，问题产生于我们使用的高精度库 gmp，由于我们在计算过程中采用的是高精度有理数，而返回时需要转化成 `double` 类型，故而我们使用了 gmp 库提供的 `get_d()` 方法。然而，查阅资料后发现，这一方法返回时并未四舍五入，而是采用了向零取整。而由于算法原理的原因，我们计算出的当前实根近似值虽然非常接近真实值 9，但却始终不到 9，向零取整后便变为了现在给出的 x_1。采用了自己实现的四舍五入方法后，这一问题便改善了（但没有完全解决）。

8 总结

在小组成员的算法设计中，三个人各采用了三种算法。

吕敬一采用了 Sturm 区间分割达到足够小区间以使用 Newton 迭代求根的方法。

潘佳奇采用了基于 Descartes 符号规则的分割区间的方法，在一个区间内只有一个实根的时候使用区间 Newton 迭代求根。

叶隽希采用了基于 Descartes 符号规则的二分分割区间的方法，在区间足够小时，使用 Newton 迭代求根。

在小组内的实现中，吕敬一的实现运行效率最高。潘佳奇的实现使用了区间 Newton 迭代法，相对普通 Newton 迭代法，采用区间运算可以保证最后不丢根且收敛。叶隽希的实现提供了一种十分简明的求根方法，其代码复杂度是三者最低的。

由于优化不够充分，小组实现的基于 Descartes 符号规则的方法未能发挥其理论上的效率。在小组的实现中有过度依赖高精度有理数等问题。经过测试使用高精度实数能带来更高的效率，但是会大量增加调试和测试的负担。采用精细实现的基于高精度实数的算法，是该项目一个可选的改进方向。

在这个项目中，各成员的主要贡献如下：

- 吕敬一：文档编写、基础定理证明、项目代码整合、用户接口。
- 潘佳奇：文献收集、推导和编写公用函数、寻找优化方向、引入 gmp 库。
- 叶隽希：维护项目结构、小组研究方向、数据测试系统、管理外部依赖以及项目编译。

得益于三人合作，最终得到一个完整的项目。小组三人在这门课中收获颇丰。为此，感谢雍俊海老师的教导。