1 绪论

1.1 算法与时间复杂度

算法是信息科学中的基本概念。20 世纪 60 年代 Donald Knuth 出版了《The Art of Computer Programming》，以各种算法研究为主线，确立了算法的重要性，1974 年获得图灵奖。算法问题看似简单，但实际复杂，其中常蕴含深刻的数学原理，并涉及算法设计和数值计算等技巧。

一千多年前波斯人编著了一部数学著作，论述了印度的十进制记数法等算术知识，后来被意大利人翻译为拉丁文，欧洲人才用上了阿拉伯数字。这就是英文 Algorithm（算法）这个词的来历。算法原指用数字实现计算的过程，现指用计算机解决问题的程序或步骤。算法没有简单明确的定义。举一个算法实例，辗转相除法（Euclidean algorithm）是求两个正整数之间最大公约数的算法，出现于欧几里德的《几何原本》和东汉的《九章算术》，可能是最古老的算法。辗转相除就是反复求余数直到余数为 0，例如假设要求 60 和 25 的最大公约数，MOD(60, 25)=10, MOD(25, 10)=5, MOD(10, 5)=0，所以 60 和 25 的最大公约数为 5，这里 MOD()是求余函数。下面给出用辗转相除法计算正整数 A、B 最大公约数的算法流程图：

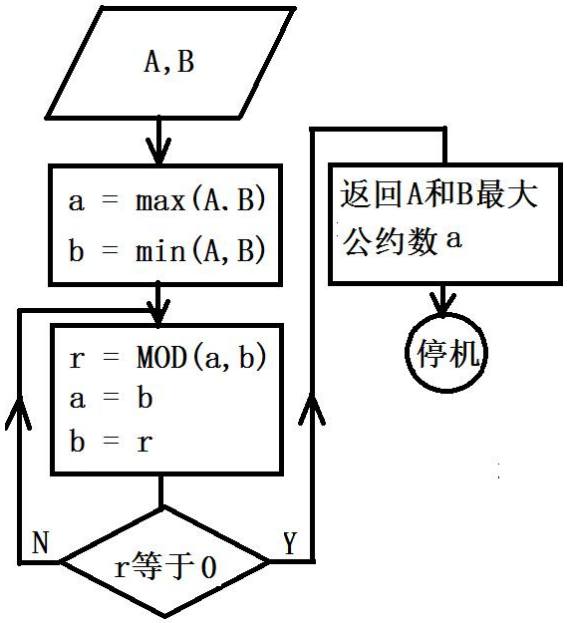


图 1.1 辗转相除法算法流程图

如同建筑设计、产品制造，总需要一些工程图纸一样，在生产实际中常需要进行算法设计。解决工程计算问题的一般过程：工程问题→数学模型→算法设计→程序实现。有些简单算法仅与常见数据结构有关，如冒泡排序法等，有些算法则基于数学或物理原理，并依赖数值计算技巧。平面区域和空间曲面的三角剖分在 CAD、CAM、CAE 等工程领域中有非常广泛的应用。给定平面上一个点集，存在一种三角剖分，使三角形的最小角达到最大，称为 Delaunay 剖分。如何实现这种剖分？这就要介绍俄国数学家 Voronoi 的发现——Voronoi 结构，它在自然界中普遍存在。假设平面上有 n 个点 $\boldsymbol{P}_i$，这些点将平面分为 n 个区域 Ω_i，Ω_i 内的点到 $\boldsymbol{P}_i$ 比到 $\boldsymbol{P}_j$ 近，i 不等于 j, 见下图。Voronoi 图是 Delaunay 三角剖分的对偶图。Voronoi 结构的边是相邻顶点的垂直平分线。

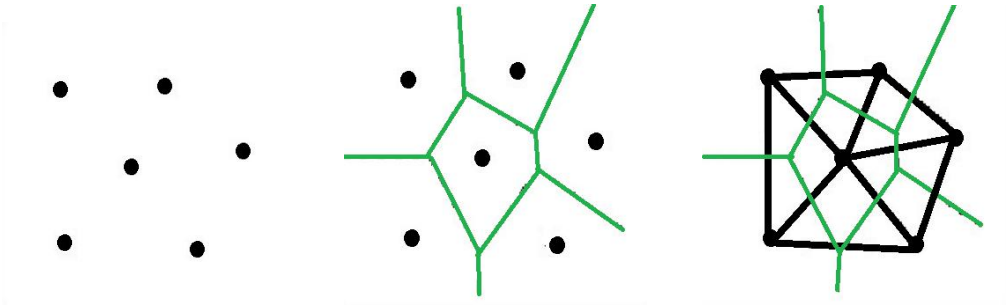
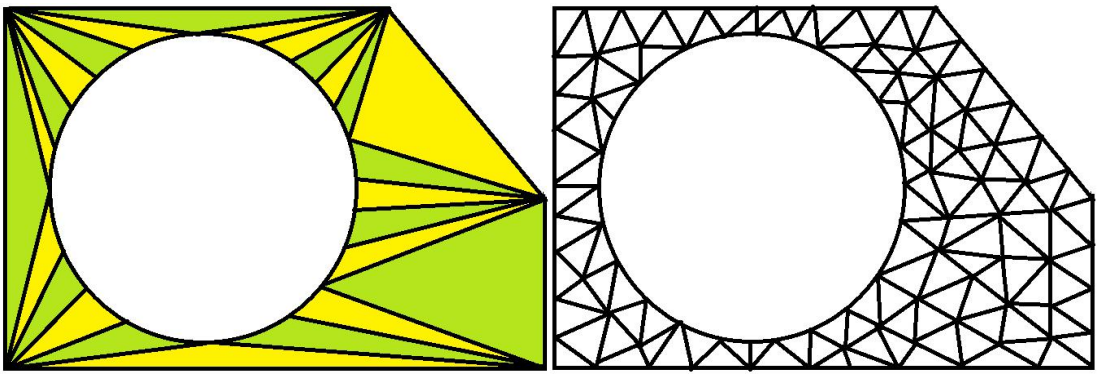


图 1.2 Voronoi 结构与 Delaunay 三角剖分



（a）用于图形显示的三角网格（b）用于有限元分析的三角网格

图 1.3 三角剖分实例

什么是算法？很难给出一个完整的、明确的定义。简单说**算法**是满足下列条件的一系列计算步骤：

1. **有限性**：有限步内必须停止。
2. **确定性**：每一步都是严格定义和确定的动作。
3. **可行性**：每一个动作都能够被精确地机械执行。
4. **输入**：有一个满足给定约束条件的输入。
5. **输出**：满足给定约束条件的结果。

对于一个算法，一般须做如下几个方面的分析：

1. **正确性**：称一个算法是正确的，如果它对于每个输入都最终停止，而且给出正确的输出。调试程序不等同于程序正确性证明。程序调试只能证明程序有错误，不能证明程序正确。

2. **复杂性**：运行一个算法总要消耗一定的时间、占用一定的内存，具体耗费多少的时间和内存与输入规模有关。算法的资源占用量与输入规模构成函数关系，该函数就是算法的复杂度。一般不考虑算法的具体运行环境，仅从时间和空间两方面定性地分析算法的复杂度。

3. **稳定性**：对各种可能的输入，算法能够给出合理的输出。

4. **可读性**：算法的各步骤要容易被人理解。

用记号 O() 定性地表示算法的**时间复杂度**，例如一个算法的时间复杂度为 $O(n)$ 表示该算法的原子操作总数与输入规模 n 成正比。一次加减乘除、赋值等运算都可以看作是一次原子操作。算法的原子操作总数是输入规模 n 的函数，记为 $f(n)$，如果 $\lim\limits_{n\to\infty}\dfrac{f(n)}{n^2}=C$，其中 C 为常数，则称该算法的时间复杂度为 $O(n^2)$。

考虑计算两个 n 阶矩阵乘积的算法，假设矩阵 $A=(a_{ij})$，$B=(b_{ij})$，$C=AB=(c_{ij})$，则根据矩阵乘法的定义，$c_{ij}=\sum\limits_{k=1}^{n}a_{ik}b_{kj}$，算法通过 3 次循环语句完成计算，所以算法的计算量与 n^3 成正比。因此，该算法的时间复杂度为 $O(n^3)$。

算法 1.1 矩阵乘法

Input:矩阵阶数 n，$A=(a_{ij})$，$B=(b_{ij})$

Output:$C=(c_{ij})$

```
Begin
    For i←1 to n, do
        For j←1 to n, do
            cij←0.
            For k←1 to n, do
                cij←cij+aik×bkj
            End For
```

```
        End For
    End For
End
```

当 n 很大时，有不等式 $\log_2 n < n < n\log_2 n < n^2 < n^3 < 2^n < n!$。多项式级的复杂度远远小于指数级复杂度，见图 1.4。

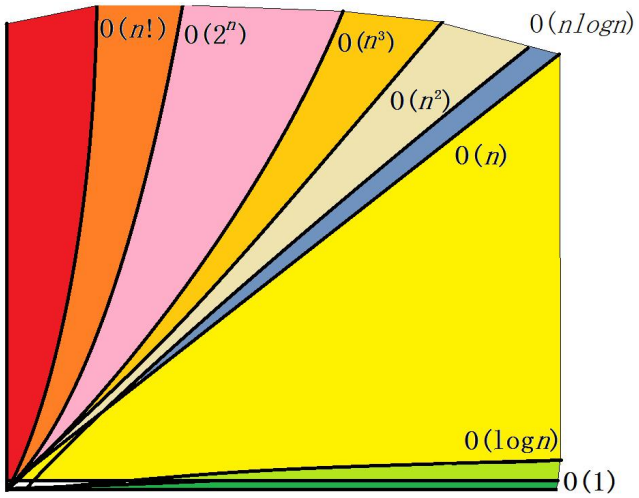
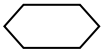


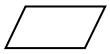
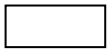
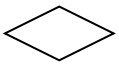
图 1.4 不同量级的算法复杂度

算法就是一系列的计算步骤，具体表现出来可以是一段自然语言、伪代码、程序，或者一个程序框图。简单算法只有几行代码，复杂算法可能需要调用其它成百上千个子算法。算法与控制论中的黑箱子（black box）（见图 1.5），数学中的变换，C 语言中的函数等对象有一定的相似性。常用标准流程图（表 1.1）的形式描述算法。

图 1.5 算法可看作变换或函数

表 1.1 算法的流程图

图形	名称	含义
	开始	算法开始及算法初始化

	数据	输入、输出的数据
	处理	算法处理过程
	判断	算法判断过程
	流程线	处理顺序关系
	终止	算法终止

1.2 图灵机与可计算性

图灵机（Turing Machine）是解决计算问题的一个抽象模型。一般认为可用算法解决的问题等价于能用图灵机解决。从图灵机的角度看，算法就是图灵机上运行的程序，Edmonds 提出评价一个算法优劣的标准是该算法的时间复杂度是否是多项式级别的。

图灵（Alan Turing）在 1936 年发表论文“On Computable Numbers, with an Application to the Entscheidungs problem”，提出了一个抽象的计算模型，即图灵机模型，其核心思路是将计算（推理）看做一系列简单的机械动作。

图灵机由一根两端可无限延长的纸带和一个有读写头的控制器组成。纸带分成相同方格，读写头可在方格上书写一个符号。这些符号构成有穷字母表：$\{a_0, a_1 \cdots a_m\}$。控制器的状态表为：$\{s_0, s_1 \cdots s_n\}$，每一时刻控制器处于一个状态，即当前状态。指令定义为五元组(s_i, a_j, a_k, x, s_e)，其中：

（1）s_i是控制器目前所处的状态；

（2）a_j是读写头从方格中读入的符号；

（3）a_k 是用来代替 a_j 写入方格中的符号；

（4）x 为 R、L、N 标志之一，该三个标志分别表示控制器向右移一格、向左移一格和不移动；

（5）s_e 是控制器的后续状态。

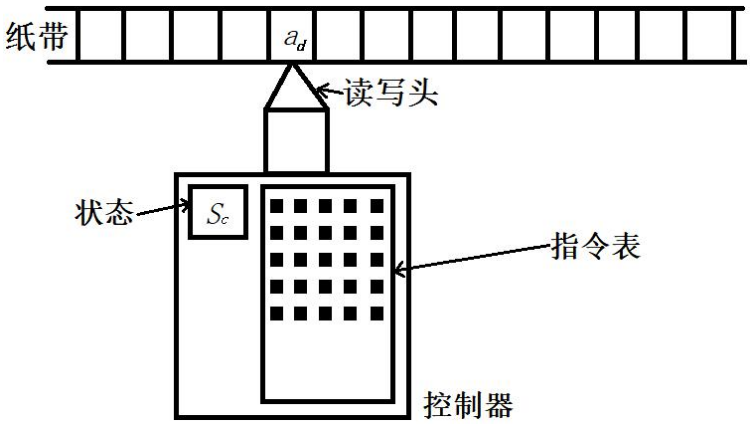


图 1.6 图灵机

图灵机有六个元操作：（1）读当前符号；（2）写当前符号；（3）左移一格；（4）右移一格；（5）改变控制器的当前状态；（6）停止。

图灵机运行前，输入记录了符号串的初始纸带，纸带上某一特定的方格被置于读写头之下，控制器先处于初始状态。图灵机依照控制器内设定的“程序”（亦称为状态转换表，即指令表）运行：根据读写头读取的当前符号和控制器的当前状态，查找指令表得到对应的指令；依据该指令确定(a)要写入的符号，(b)控制器向左或右移动一格（或不动），(c)控制器更新状态。这一系列操作称为图灵机运行的一步。当运行终止时纸带上的符号串就是输出数据。

计算复杂性理论发源于二十世纪六十年代，理论基础就是图灵机。如果问题无法在多项式时间（即时间复杂度为多项式级别）内被

图灵机解决，则称该问题为难解问题。对于一个问题，如果存在**求解**该问题的一个算法，且该算法的时间复杂度为多项式级的，称该问题为 **P 问题**；对于一个问题，如果存在**验证**问题解是否正确的一个算法，且该算法的时间复杂度为多项式级的，称该问题为 **NP 问题**。

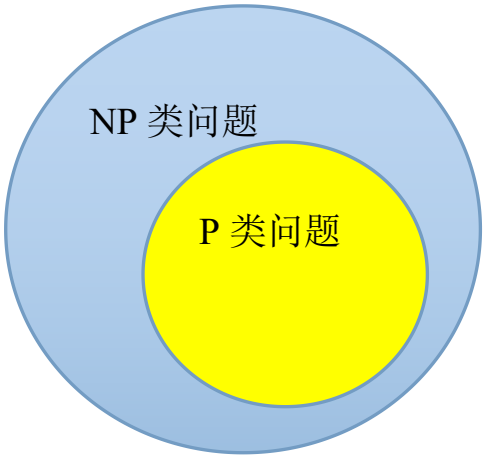


图 1.7 P 问题与 NP 问题

“NP=P 还是 NP>P?”这个问题是美国 Clay 数学研究所 2000 年提出的悬而未决的 7 大千禧年数学问题之一。

两个典型的 NP 问题：

（1）旅行商问题(Traveling Salesman Problem，TSP)：旅行者寻求由起点出发，通过所有给定的城市后，回到起点的最短路径。对于有 n 个城市的旅行商问题，所有可能路径共$(n-1)!$个，通过遍历所有路径得到最优解的算法复杂度是 O$(n!)$。一般用启发式算法求解。启发式算法是指基于直观、经验或自然现象构造的求解问题的算法，如遗传算法、模拟退火算法等。有文献指出蜜蜂是解决旅行商问题的专家。

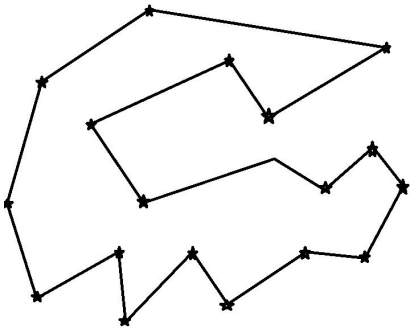


图 1.8 旅行商问题

（2）背包问题(Knapsack problem)是一种组合优化问题。给定若干有一定质量和价值的物品和一个背包，放入背包的物品总质量有确定的上限，如何挑选物品放入背包使背包中的物品总价值最大。解决该问题的算法具有重要的应用价值。

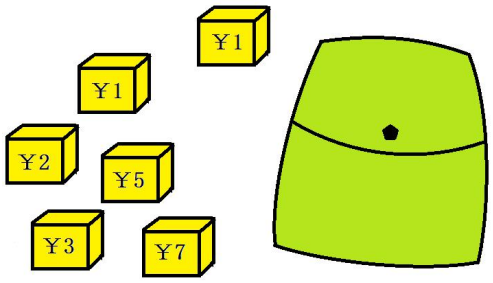


图 1.9 背包问题

以图灵机为理论模型，提供计算能力的硬件平台从小型机、大型机、PC 机、个人工作站、巨型机，到移动终端及 GPU（Graphics Processing Unit，图形处理器）、Google 的 TPU（Tensor Processing Unit，张量处理器）等高性能计算（High performance computing）平台,再到云计算（Cloud Computing）平台，硬件技术日新月异。实现计算功能的数学方法也迅猛发展，除了传统的数值计算方法，20 世纪末涌现出遗传算法等启发式算法，近年来的深度学习算法更是迅猛发展。“计算”（Computing）已逐步渗透到生产、社会生活的各个层面。

基于高性能计算的**模拟仿真**是继理论研究和实验验证之后，人类研究活动的第三类方式。

图 1.10 Tesla C1060 图形处理器

1.3 什么是数值计算

有一本 2013 出版的书《改变未来的九大算法》，作者是 John MacCormick，书中列举了几种影响人类社会的算法：搜索引擎、PageRank、公开密钥加密、纠错码、模式识别、数据压缩、数据库、数字签名等。这本简单的科普书启示人们算法的重要性。事实上，重要的算法还有很多，仅以机械制造领域为例，20 世纪 70、80 年代的数控系统中的运动控制算法、CAD 系统中的曲面、实体造型算法、特征造型算法等都是关键算法。近几年快速发展的机器学习、人工智能算法，是机器人、无人驾驶汽车、无人机等领域的核心技术。在实际应用中，由于各种条件的限制，一般总是寻求工程问题的近似解，而非解析解（精确解）。求解近似解的算法称为**数值算法**，相关计算就是数值计算。实现数值计算的基本方法，例如插值和逼近等，称为**数值方法**（Numerical method）或计算方法。

本教材讲授数值计算中的基本方法。下面给出几个的有关数值计算的具体实例：

（1）ICP（迭代最近点）算法。该算法目标是使空间两个点集实现最佳重合，方法是通过多次旋转、平移一个点集使之与另一个点集逐步配准，配准的目标函数是最近距离点对的距离的平方和，即 $g(R,T)=\sum_{i=0}^{n-1}\|P_i-(RQ_i+T)\|^2$ 达到最小，其中 R、T 是旋转矩阵、平移矢量。如何求解这个优化问题，这就涉及到了各种数值计算中的基本方法。ICP 算法广泛应用于光学、影像测量设备及其相关的软件系统，见图 1.11 和图 1.12。

图 1.11 ATOS 三维光学扫描检测仪

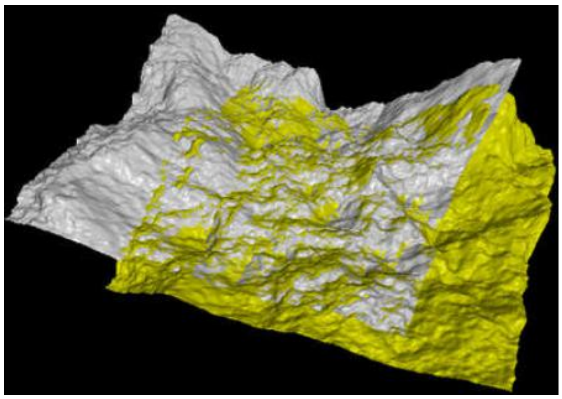


图 1.12 点云配准

（2）三维几何建模与仿真。CAD 造型过程中涉及大量曲线、曲面插值、优化及求交。求交就是判断曲线、曲面之间交点，本质是就是求解非线性方程组。下面各应用都以数值计算为基础。

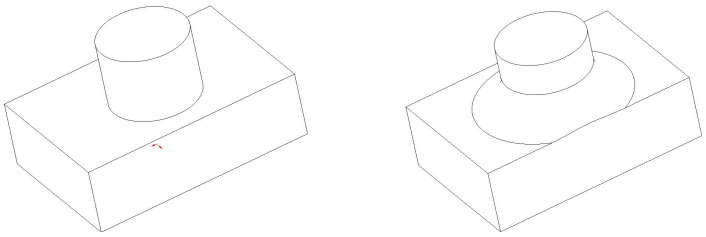


图 1.13 渡曲面的边界裁剪

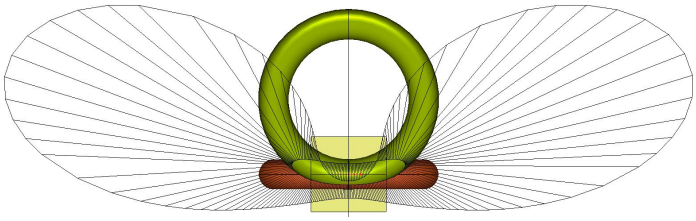
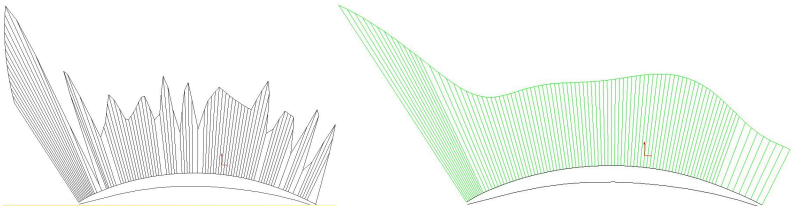
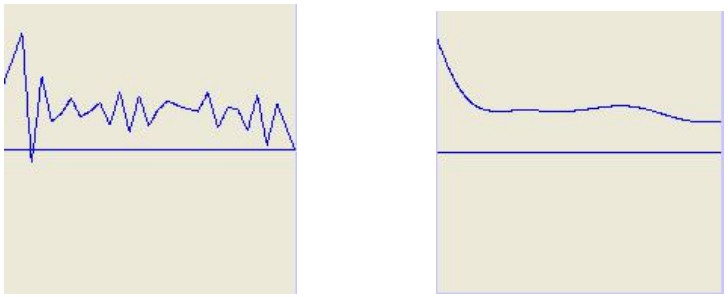


图 1.14 两圆环面垂直相交（有切圆）的交线曲率梳

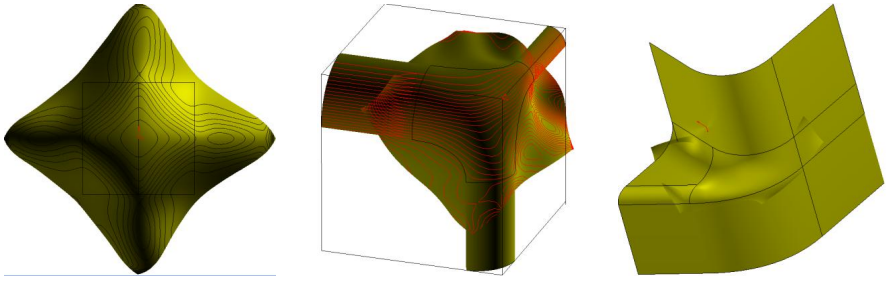


（a）光顺前的曲率梳图　　（b）光顺后的曲率梳图

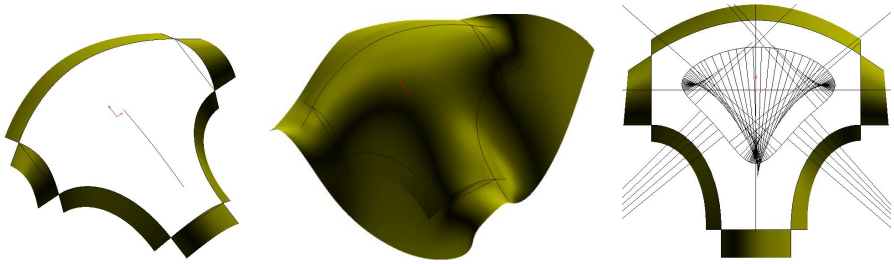


（c）光顺前的曲率图　　（d）光顺后的曲率图

图 1.15 叶片截面线的光顺前后效果对比



(a)4 边域光顺插值（等照度线）(b)六边域光顺插值（等照度线 (c)复杂过渡曲面



(d)原始边界条件 (e)光顺曲面（G0 误差 0.0012mm，G1 误差 1.1 度）(f)截面线的曲率

图 1.16 曲面填充（插值与光顺）

图 1.17 管件的弯制与干涉仿真

（3）草图求解。在三维特征造型过程中，大约 80%的时间用于绘制草图，提高草图绘制效率的关键技术就是参数化草图求解技术，即将尺寸约束、几何约束转化为非线性方程组，再用各种特殊方法求解非线性方程组。图 1.18 所示约束求解器 DCube 应用于 solidworks 等 CAD 系统。

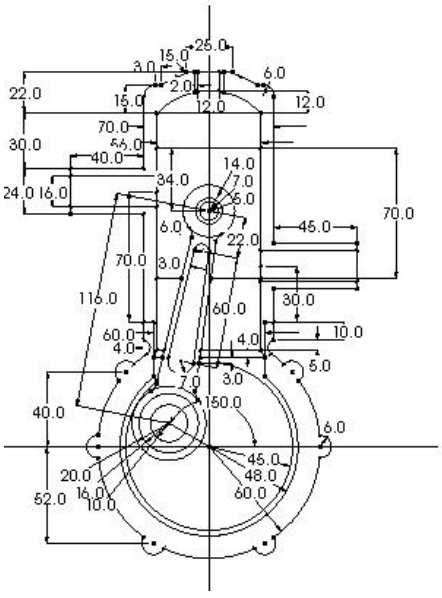


图 1.18 Simens 的 DCube 求解器计算实例

（4）线性规划。线性规划就是求解满足多个不等式约束的目标函数为线性函数的优化问题。George Dantzig 于 1947 年提出了求解线性规划问题的**单纯形算法**（Simplex algorithm）。单纯形是 n 维空间中的由 n+1 个顶点构成的凸包（凸包中任何两点的连线包含于该凸包），一维直线上的一个线段，二维平面上的一个三角形，三维空间中的一个四面体等都是单纯形。线性规划中的不等式约束实际上定义了一个凸包，线性目标函数的最优值一定在顶点取得，所以其优化过程就是从一个顶点到另一顶点，不断寻找最优值。图 1.19 是两个变量的线性规划问题对应的几何图形。

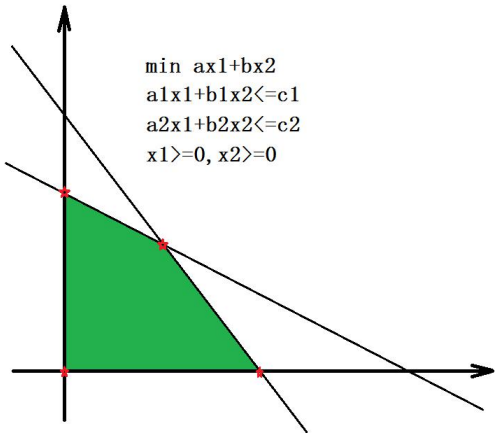


图 1.19 两个变量的线性规划

（5）非线性优化。非线性优化就是求解约束条件或目标函数为非线性函数的优化问题。钻孔问题：目的是钻 n 个孔 $\boldsymbol{P}_0,\boldsymbol{P}_1,\ldots,\boldsymbol{P}_{n-1}$，使总路径 L 最短，就是要得到 n 个孔的一个优化排序，见图 1.20。优化下料问题：分线材、矩形、异型等情况，图 1.21 是钢格板优化下料工艺图，图 1.22 是玻璃切割优化下料工艺图。五坐标数控加工编程算法中的刀轴姿态计算也是非线性优化问题，见图 1.23。

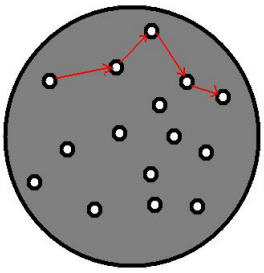


图 1.20 钻孔加工

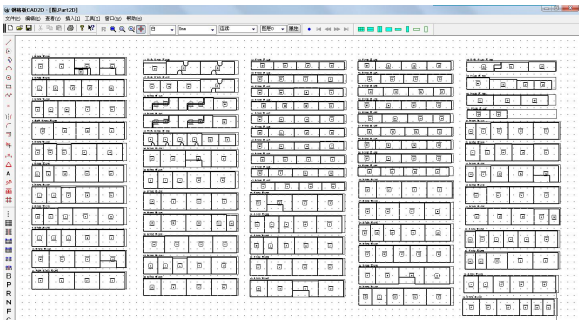


图 1.21 钢格板工艺优化排料图

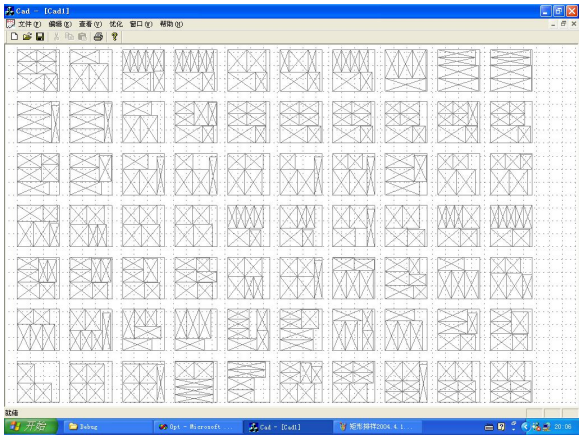


图 1.22 矩形优化下料

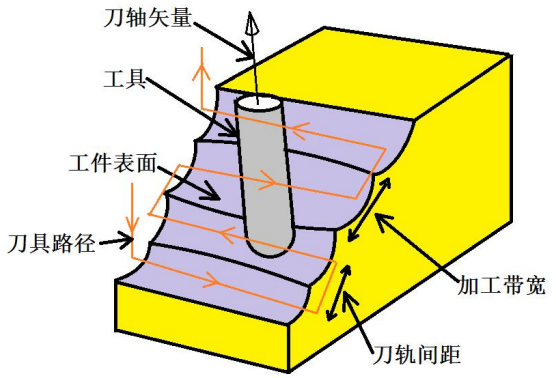


图 1.23 宽行加工中的刀轴矢量优化

还有很多应用，图像识别（文字、指纹、人脸）、数据压缩、加

密、数据挖掘、计算机视觉、自然语言处理、机器学习等，都涉及到数值计算中的基本方法。在实际工程中，存在大量的数值计算问题，可以说举不胜举。20 世纪中期随着计算机的诞生与发展，计算方法也飞速发展，并与各学科交叉渗透，形成了计算物理、计算化学、计算生物、统计物理学、计算几何、计算生物学、计算流体力学等。2008 年美国国家科学基金会（NSF）宣布将投资 1600 万美元用于建立国立数学生物学综合研究所（NIMBioS）。NSF 为此专门启动了一项“定量的环境与整合生物学”项目。美国国立卫生研究院(NIH)也设立了一项“计算生物学”的重大项目。

1.4 误差与浮点数

误差是数值计算中的基本概念。某个量的近似值 x^* 与真实值 x 之差称为误差。

定义:绝对误差：设 x 是精确值，x^* 是它的一个近似值，称 $e=x^*-x$ 是近似值 x^* 的绝对误差。若 $|x^*-x|\le\varepsilon$，称 ε 是近似值 x^* 的**绝对误差上限**。

定义:相对误差：称 $\frac{e}{x}=\frac{x^*-x}{x}$ 为近似值 x^* 的相对误差，记作 e_r。若 $|e_r|\le\varepsilon_r$，称 ε_r 为近似值 x^* **相对误差上限**。在实际应用中，有时可用 x^* 来替代 x，即 $e_r=\frac{e}{x^*}=\frac{x^*-x}{x^*}$。

定义:若近似值 x^* 的误差上限是 $\frac{1}{2}\times 10^{-n}$，其中 n 是正整数，则称 x^* 准确到小数点后 n 位，从第一个非零的数字到该位的所有数字称为有

效数字。

【例 1】$\pi \approx 3.14159265$，3.1416 有五位有效数字，误差上限为 0.00005。

下面给出有效数字与误差的关系。若 x^* 有 n 位有效数字，其标准形式为 $x^* = \pm\, a_1.a_2 \cdots a_n \times 10^m$，其中“.”是小数点，例如 123.4 表示为 1.234×10^2，则有 $\varepsilon = |x - x^*| \le \frac{1}{2}\times 10^{m-n+1}$，且其相对误差上限 $\varepsilon_r \le \frac{1}{2a_1}\times 10^{-(n-1)}$。

利用用 Taylor 公式讨论函数 $f(x)$ 的误差与 x 的误差之间的关系。设 x^* 是 x 的近似值，且有绝对误差上限 $|x^* - x| \le \varepsilon$。如果 $f(x)$ 可微，则有 $f(x) - f(x^*) = f'(x^*)(x - x^*) + \frac{f''(\xi)}{2}(x - x^*)^2$，$\xi$ 介于 x 与 x^* 之间，于是 $|f(x) - f(x^*)| \le |f'(x^*)|\varepsilon + \frac{|f''(\xi)|}{2}\varepsilon^2$。

绝对误差上限的四则运算公式：

$$\varepsilon(x_1^* \pm x_2^*) \le \varepsilon(x_1^*) + \varepsilon(x_2^*) \tag{1.1}$$

$$\varepsilon(x_1^* x_2^*) \le |x_1^*|\varepsilon(x_2^*) + |x_2^*|\varepsilon(x_1^*) \tag{1.2}$$

$$\varepsilon\left(\frac{x_1^*}{x_2^*}\right) \approx \frac{|x_1^*|\varepsilon(x_2^*) + |x_2^*|\varepsilon(x_1^*)}{|x_2^*|^2}, x_2^* \ne 0 \tag{1.3}$$

以乘积的绝对误差上限为例分析上面的公式，因为

$$\varepsilon(x_1^* x_2^*) = x_1^* x_2^* - x_1 x_2$$

$$= x_1^* x_2^* - x_1^* x_2 + x_1^* x_2 - x_1 x_2$$

$$= x_1^*(x_2^* - x_2) + x_2(x_1^* - x_1)$$

故

$$\left|\varepsilon(x_1^*x_2^*)\right| \le \left|x_1^*\right|\left|x_2^* - x_2\right| + \left|x_2\right|\left|x_1^* - x_1\right|$$

在推导过程中，用 x_2^* 近似地替代 x_2 得到对应的公式

$$\left|\varepsilon(x_1^*x_2^*)\right| \le \left|x_1^*\right|\left|\varepsilon(x_2^*)\right| + \left|x_2^*\right|\left|\varepsilon(x_1^*)\right|$$

误差分类：

- **模型误差**：工程问题转化为数学模型时产生的误差（比如由复杂现象、物理原理的简化而导致的误差），
- **测量误差**：测量物理量等数据时，由于测量设备的精度有限所导致的误差，
- **截断误差**：在设计算法时，经常要进行近似处理（如取 Taylor 展开的前有限项等），这样必然引入一定量级的误差，
- **舍入误差**：由于计算机的存储空间、字长是有限的，用浮点数近似表示实数，浮点运算要进行四舍五入处理所导致的误差。

【例 2】人体模型在动画、仿真等领域有重要应用。通常将人体模型的关节简化为球形（见图 1.24），旋转时有一定的角度限制。由于人体关节非常复杂，这种简化必然会导致误差的存在，这就是模型误差。

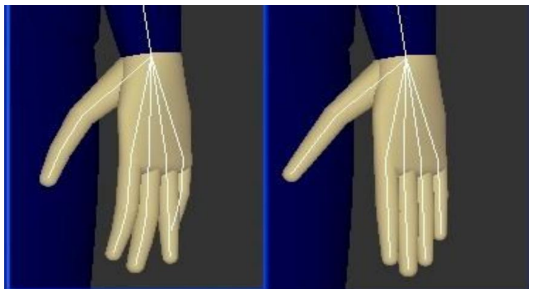


图 1.24 人体骨骼模型

【例 3】各种观测设备都有误差（见图 1.25），如接触式三坐标测量机的误差可能是几个μm，激光测量机误差可能是十几个μm，光学测量机误差可能是 100 个μm 左右。

图 1.25 便携式关节臂测量误差 0.02mm

【例 4】对函数 $f(x)$ 实施 Taylor 展开，用多项式 $P_n(x)=f(0)+\frac{f'(0)}{1!}x+\frac{f''(0)}{2!}x^2+\cdots+\frac{f^{(n)}(0)}{n!}x^n$ 近似代替原函数，则截断误差为 $R_n(x)=f(x)-P_n(x)=\frac{f^{(n+1)}(\xi)}{(n+1)!}x^{n+1}$。

【例 5】$\cos x=1-\frac{x^2}{2}+\frac{x^4}{4!}-\frac{x^6}{6!}+\cdots+\frac{(-1)^n x^{2n}}{(2n)!}+\cdots$，当 $|x|$ 很小时，可用 $1-\frac{x^2}{2}$ 作为 $\cos x$ 的近似值，这种近似方法所导致的误差就是截断误差，其值小于 $\frac{x^4}{24}$。

只有“很少”一部小数可以用计算机精确表示，由此造成的误差就是舍入误差。C 语言或者 Java 语言都实现了 IEEE754 的**浮点数**标准。比如 C 语言或者 Java 语言中的 float 和 double 分别对应单精度浮点数和双精度浮点数。**单精度浮点数**由 32 位二进制位组成（4 个

字节），其中 1 位符号位，8 位指数位，23 位底数位；**双精度浮点数**由 64 位二进制位组成（8 个字节），其中 1 位符号位，11 位指数位，52 位底数位。浮点数的表示是基于二进制的科学计数法，二进制和十进制小数的对应关系见表 1.2：

表 1.2 二进制小数和十进制小数的对应关系

二进制小数	...	1	1	1	1	.	1	1	1	1	...
十进制小数	...	8	4	2	1	.	0.5	0.25	0.125	0.0625	...

不难看出，$(0.1)_2=(0.5)_{10}$，$(0.3)_{10}=(0.01001...)_2$，十进制中的 8, 4, 2, 1, 0.5, 0.25 等可以用浮点数精确表示，而十进制中的 0.3 这样的数就不能用浮点数精确表示。用科学计数法表示的二进制小数，只要指数位和底数位长度不超过浮点数定义的限制，可直接表示为浮点数：

$$(-1)^{\text{sign}}(1.M)_2\times 2^{E-1023} \tag{1.4}$$

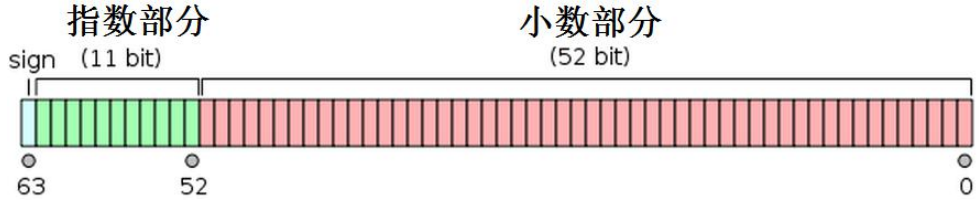


图 1.26 双精度数的表示

浮点数标准规定（参考图 1.26）：

（1）符号位 sign：“0”为正，“1”为负，

（2）指数部分 E：为无符号整数，E 为 8 位或 11 位时，其真实值为 E - 127 或 E - 1023。E 不全为 0 且不全为 1 时，小数部分 M 前加上 1；E 全为 0 时，M 前不再加上 1；E 全为 1 时，如果 M 全为 0，表示±无穷大（正负由符号位 sign 决定），否则表示不是一个数（NaN）。

（3）小数部分 M：一般情况下 1<=1+M<2，即 $M = 0.d_{51}d_{50}\cdots d_1d_0$ 。

程序示例 1.1 计算机只能精确表示“很少”一部分小数

```
#include "stdafx.h"
#include "math.h"

int main(int argc, char* argv[])
{
	double a = 0.2, b = 0.3, h = 0.5 ;

	return 0;
}
```

表 1.3 在 debug 状态下程序示例 1.1 的运行结果：

a	0.20000000000000001
b	0.29999999999999999
h	0.50000000000000000

下面的程序示例 1.2 可以取出单精度浮点数“9.f”的二进制表示，并存于数组 bits 中。

程序示例 1.2 单精度浮点数“9.f”的表示

```
#include "stdafx.h"

int main(int argc, char* argv[])
{
	int i, n, bits[32], bbb = 1 ;
	float* p ;

	p = (float*)&n ;
	*p = 9.f ;
	for( i = 0 ; i < 32 ; i++ )
	{
		if( n&bbb )
			bits[i] = 1 ;
		else
			bits[i] = 0 ;
		bbb = bbb<<1 ;
	}
```

```
    return 0;
}
```

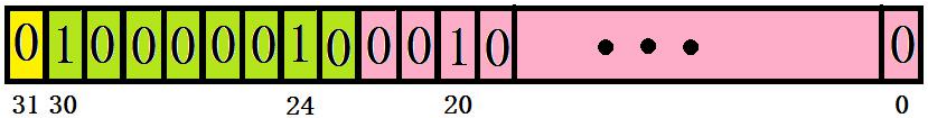


图 1.27 单精度浮点数”9.f”的二进制表示（32 位）

根据 9.f 的二进制表示，sign=0，E=130，M=$(0.001)_2$，所以它表示的浮点数=$(-1)^0(1.001)_2\times2^3$=9，见图 1.27。

程序示例 1. 3 双精度浮点数 1. 对分多少次变为 0.

```
#include "stdafx.h"

int main(int argc, char* argv[])
{
    int i ;
    double d = 1. ;

    for( i = 0 ; i < 1075 ; i++ )
    {
        d /= 2 ;
    }

    return 0;
}
```

根据浮点数的性质：

（1）**避免大数加减小数**，比如 1.+1.0e-20 还是 1.0，

（2）**不能直接判断两个浮点数是否相等。**0.2*3 等于 0.6 吗？见下面的程序：

程序示例 1. 4 在计算机中 0. 2×3 不等于 0. 6

```
#include "stdafx.h"

int main(int argc, char* argv[])
{
    double d ;

    d = 0.2*3-0.6 ;
```

```
    return 0;
}
```

程序示例 1.4 的运行结果是 d=1.1102230246251565e-016，表明在计算机中 0.2×3 不精确等于 0.6。

程序示例 1.5 在计算机中 0.25×2 等于 0.5

```
#include "stdafx.h"

int main(int argc, char* argv[])
{
    double d ;

    d = 0.25*2-0.5 ;

    return 0;
}
```

程序示例 1.5 的运行结果是 d=0，表明在计算机中 0.25×2 精确等于 0.5。

1.5 数值稳定性

舍入误差能控制在某个范围内的算法称之为数值稳定，否则称之为不稳定。

几个关于数值稳定性的基本原则：

（1）两个相近的数相减，有效数字损失很大。 $x-y$ 的相对误差为 $e_r(x-y)=\dfrac{e(x)-e(y)}{x-y}$，当 x 与 y 很接近时，两数之差 $x-y$ 的相对误差会很大。

【例 1】 $2-\sqrt{3}\approx 0.2679491924311227$，如用四位有效数字计算：$2-\sqrt{3}\approx 2-1.732=0.268$，结果只有两位精确数字；若改为：$2-\sqrt{3}=\dfrac{1}{2+\sqrt{3}}\approx\dfrac{1}{2+1.732}\approx 0.267952$，则有四位有效数字。

【例 2】用四位浮点数计算 $\frac{1}{759}-\frac{1}{760}$。 方法（1）：$\frac{1}{759}-\frac{1}{760}\approx 0.1318\times10^{-2}-0.1316\times10^{-2}=0.2\times10^{-5}$，只有一位有效数字；方法（2）：$\frac{1}{759}-\frac{1}{760}=\frac{1}{759\times760}\approx\frac{1}{0.5768\times10^{6}}=0.1734\times10^{-5}$，有四位有效数字。

（2）避免除数的绝对值远小于被除数的绝对值。$\varepsilon\left(\frac{x}{y}\right)=\frac{|x|\varepsilon(y)+|y|\varepsilon(x)}{|y|^2}$，当$|x|>>|y|$时，舍入误差会扩大。

（3）大数吃小数。如$a=10^9+1$，如果计算机只能表示 8 位小数，则算出$a=0.1\times10^{10}$。

【例 3】一元二次方程$x^2-(10^9+1)x+10^9=0$其精确解为$x_1=10^9,x_2=1$。如直接用求根公式：$x_{1,2}=\frac{-b\pm\sqrt{b^2-4ac}}{2a}$在仅支持 8 位有效数字的计算机上求解，有 $\sqrt{b^2-4ac}=\sqrt{10^{18}-4\times10^9}\approx\sqrt{10^{18}}=10^9$，及 $10^9+1\approx10^9$；则 $x_1\approx\frac{-(-10^9)+10^9}{2}=10^9$，$x_2\approx\frac{-(-10^9)-10^9}{2}=0$。$x_2$的值与精确解差别很大。实际上，

$$x_2=\frac{-b-\sqrt{b^2-4ac}}{2a}=\frac{2c}{-b+\sqrt{b^2-4ac}}\qquad(1.5)$$

若用(1.5)式计算可得精度更好的解

$$x_2\approx\frac{2\times10^9}{-(-10^9)+10^9}=1\qquad(1.6)$$

（4）减少运算次数。

【例 4】计算x^{255}的值。如果逐个相乘，则要用 254 次乘法；如果重写 $x^{255}=x\cdot x^2\cdot x^4\cdot x^8\cdot x^{16}\cdot x^{32}\cdot x^{64}\cdot x^{128}$，计算$x^{255}$只需 14 次

乘法。

【例 5】 计算多项式的值：$P_n(x)=a_nx^n+a_{n-1}x^{n-1}+\cdots+a_1x+a_0$。

方法（1）：如若按 a_kx^k 有 k 次乘法运算，计算 $P_n(x)$ 共需 $1+2+\cdots+n=\frac{n(n+1)}{2}$ 次乘法和 n 次加法运算；

方法（2）：如写成 $P_n(x)=(\cdots((a_nx+a_{n-1})x+a_{n-2})x+\cdots+a_1)x+a_0$，用递推法：$u_0=a_n, u_k=u_{k-1}x+a_{n-k}$，$k=1,2\cdots n$，可得 $P_n(x)=u_n$，共需 n 次乘法和 n 次加法运算。

1.6 教学内容与要求

经典计算方法理论主要是针对工程、物理中常见的数学问题，研究对象主要是各种方程与函数，给出其数值近似解，主要方法是迭代、插值、逼近、变步长等。现代计算方法结合了生物进化策略、人工智能理论以及并行计算技术等，研究对象扩展到大规模方程、复杂网络与系统，寻求实际问题的整体解决方案，见图 1.28。

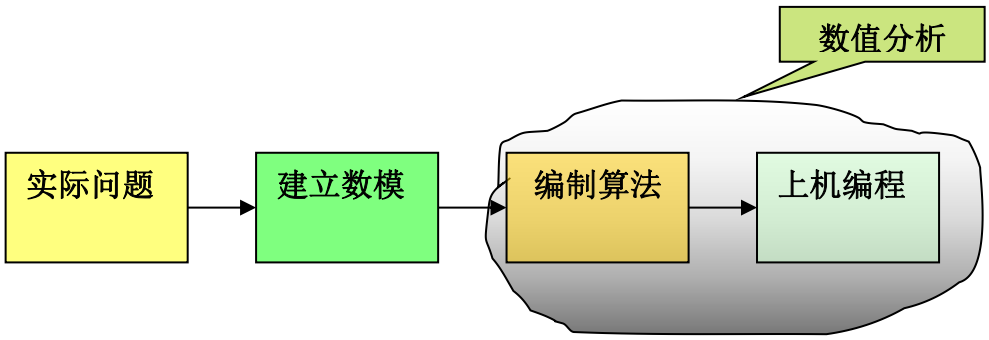


图 1.28 解决科学和工程问题的步骤

教学内容：本课程针对工程应用中最基本的数学问题，给出其数值计算方法，内容包括非线性方程数值解、线性方程组的直接法与迭代法、函数的插值与逼近、数值积分等。具体内容如下：

- 数值计算基本概念，
- 方程求解：方程包括线性方程，非线性方程等，
- 插值与逼近，
- 数值积分，
- 非线性最优化，
- 启发式算法。

教学特点：

- 注重理论分析（收敛性、稳定性、误差分析），
- 强调数值方法在工程实践中的应用。

教学要求：

- 理解基本的数值分析原理，如迭代的构造、收敛性判断等，
- 掌握本课程中给出的经典数值算法，
- 能编程实现基本的数值算法，会用 C 语言编程求解常见的工程计算问题。

练习题：

1.什么是相对误差和绝对误差？

2.什么是舍入误差？如何理解有效数字？为何在浮点数计算的过程中要尽量“避免相近的浮点数相减”？

3.叙述双精度浮点数的基本定义，并说明浮点数的加法是否满足结合律、交换律？（假设用 $f(a)$ 表示实数 a 对应的浮点数，$\oplus$ 表示浮点数加法，则 $(f(a)\oplus f(b))\oplus f(c)$ 是否等于 $f(a)\oplus(f(b)\oplus f(c))$，$f(a)\oplus f(b)$ 是否等于 $f(b)\oplus f(a)$ ）

4.分析下面代码的运行结果：

程序示例 1.6 简单一元二次方程求解公式

```
#include "stdafx.h"
#include "math.h"
int main()
{
        double a = 1.,
                    b = -(1e7+1e-7),
                    c = 1.,
                    d,
                    x1,
                    x2 ;
        d = b*b-4*a*c ;
        d = sqrt(d) ;
        x1 = (-b+d)/(2*a) ;
        x2 = (-b-d)/(2*a) ;
       return 0;
}
```

基于“避免相近的浮点数相减”原则改进一元二次方程求根公式，绘制出算法框图，并用 C 语言编程实现，要求：输入参数 a,b,c 表示方程 $f(x)=ax^2+bx+c=0$，输入容差参数 $e>0$（若 $|f(x_0)|<e$，则 x_0 可能是根，见图 1.32），输出：根的总数及方程的根。

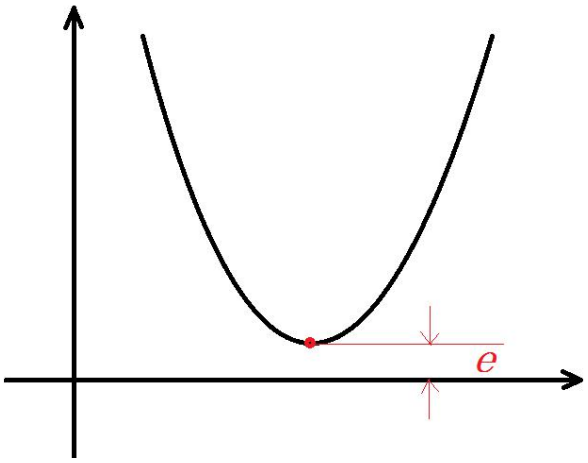


图 1.29 方程的根与容差的关系

用以下数据进行测试：

表 1.4 一元二次方程求解的测试数据

a	b	c
6×10^{154}	5×10^{154}	-4×10^{154}
0	1	1
1	-10^{5}	1
1	$-(10^{8}+10^{-8})$	1
10^{-155}	-10^{155}	10^{155}
1	-4	3.999999

5.试构造一个算法，其时间复杂度为$O(2^{2^n})$。

6.给出程序示例 1.3 中的“1075”含义？