第 3 章 遗传算法

遗传算法（Genetic Algorithm，简称 GA）是智能优化方法中应用最为广泛且成功的一种算法。作为模拟自然选择和遗传机制的优化技术，遗传算法已经在多个领域取得了显著的成果。本文将从遗传算法的起源与发展历程入手，首先介绍遗传算法的基本原理和构成要素，随后详细探讨其基本框架和常见的模型设计。接着，我们将重点分析不同的编码方式。最后，结合具体实例，展示遗传算法在实际应用中的广泛用途和其优越性。

3.1 遗传算法概述

遗传算法是一种基于自然选择和遗传机制的优化搜索算法。它最早由美国密歇根大学的 J. H. Holland 教授于 20 世纪 60 年代末提出，并在 1975 年的《自然与人工系统的适应性》一书中系统阐述了其基本理论和方法。Holland 教授提出的重要理论之一是模板理论（Schema Theory），这一理论对遗传算法的理论发展起到了关键作用。

生物在自然界中的生存繁衍，显示了其对自然环境的优异的自适应能力。遗传算法所借鉴的生物学基础就是生物的进化和遗传。生物的进化是以群体的形式共同进行的，这样的一个团体称为群体（Population），组成群体的单个生物称为个体（Individual），每个个体对其生存环境都有不同的适应能力，这种适应能力称为个体的适应度（Fitness）。按照达尔文的进化论，那些具有较强适应环境变化能力的生物个体具有更高的生存能力，容易存活下来，并有较多的机会产生后代；相反，具有较低生存能力的个体则被淘汰，或者产生后代的机会越来越少，直至消亡。达尔文把这一过程和现象叫做“自然选择，适者生存”。通过这种自然的选择，物种将逐渐地向适应于生存环境的方向进化，从而产生优良的物种。

遗传算法借鉴了达尔文的进化论和孟德尔的遗传学说，本质上是一种并行、高效、全局搜索的方法。它通过模拟生物的进化过程，在搜索过程中自动获取和积累有关搜索空间的知识，并自适应地控制搜索过程以求得最优解。遗传算法操作中使用“适者生存”的原则，通过自然选择、交叉和变异等操作，实现各代种群中个体适应性的提高。它在求解复杂优化问题方面具有巨大的潜力，已经在工

业工程、人工智能、生物工程、自动控制等多个领域得到了广泛的应用，并取得了显著的成果。

3.2 遗传算法的基本思想

遗传算法是一种基于自然选择和遗传机制的全局优化搜索算法，其灵感来源于生物进化过程。它通过模拟生物进化中的选择、交叉和变异等操作，使种群不断进化，逐步逼近最优解。以下是遗传算法基本思想的详细阐述：

3.2.1 初始种群的生成

遗传算法以种群为单位进行搜索，种群由多个个体组成，每个个体代表问题的一个可能解。初始种群的生成通常采用随机方法，这样可以保证种群的多样性和广泛性，增加找到全局最优解的可能性。当然，在某些特定问题中，也可以根据先验知识或问题特点，有针对性地构造初始种群，以提高算法的效率和效果。

3.2.2 适值函数的定义

适值函数是遗传算法中衡量个体优劣的关键指标，它直接决定了个体在进化过程中被选择的概率。适值函数通常根据问题的目标函数进行设计，将目标函数值转化为非负的适应值，且适应值越大表示个体越优秀。例如，在求解最大化问题时，适值函数可以与目标函数值成正比；而在求解最小化问题时，可以对目标函数值进行倒数或其他转换，使其符合适应值的要求。

3.2.3 选择与繁殖机制

选择操作模拟自然选择的过程，根据个体的适应值大小决定其被选中繁殖后代的概率。适应值越大的个体，被选中的概率越高，从而有机会将优秀的基因传递给下一代。常见的选择方法有轮盘赌选择、锦标赛选择等。被选中的个体通过两两配对进行繁殖，生成新的个体组成新一代种群。繁殖过程主要涉及交叉和变异操作。

交叉操作模拟生物有性繁殖中的基因重组过程，通过交换两个个体的部分基因，产生具有新特征的后代。交叉操作可以增加种群的多样性，探索新的解空间。变异操作则模拟生物基因突变现象，随机改变个体的某些基因，防止算法陷入局

部最优解，同时保持种群的活力和创新性。

3.2.4 进化过程与最优解的产生

遗传算法通过不断重复选择和繁殖过程，使种群不断进化。随着代数的增加，种群中优秀个体的比例逐渐上升，适应值也不断提高。经过若干代的进化后，种群中适应值最好的个体所对应的解，即被视为问题的最优解或近似最优解。

3.3 遗传算法的特点

遗传算法是模拟生物在自然环境中的遗传和进化过程而形成的一种并行、高效、全局搜索的方法，它具有以下特点：

3.3.1 编码处理决策变量

遗传算法以决策变量的编码作为运算对象。这种编码处理方式，使得在优化过程中可以借鉴生物学中染色体和基因等概念，模拟自然界中生物的遗传和进化等机制，方便地进行遗传操作。例如，在解决旅行商问题时，可以将路径表示为一个染色体，其中每个基因代表一个城市。通过对染色体进行交叉和变异操作，可以生成新的路径，从而逐步逼近最优解。这种编码处理方式不仅简化了算法的实现，还使得遗传算法能够处理各种类型的优化问题，包括连续优化和组合优化。

3.3.2 直接利用目标函数信息

遗传算法直接以目标函数值作为搜索信息。它仅使用由目标函数值变换来的适应度，就可确定进一步的搜索方向和搜索范围，而不需要目标函数的导数等其他一些辅助信息。在实际应用中，很多问题的目标函数无法用显式表达式表示，或者求导非常困难，甚至根本不存在导数。对于这类问题，遗传算法显示出了其独特的优势，因为它避开了求导的障碍。例如，在工程设计优化中，目标函数可能涉及复杂的仿真计算，遗传算法可以通过直接利用仿真结果进行搜索，而无需关心目标函数的具体数学形式。

3.3.3 群体搜索策略

遗传算法同时使用多个搜索点的群体信息。遗传算法对最优解的搜索过程，是从一个由很多个体所组成的初始群体开始的，而不是从单一的个体开始的。通

过对这个群体所进行的选择、交叉、变异等运算，产生出新一代的群体，其中包括很多群体信息。这些信息可以避免重复搜索一些不必要的点，相当于同时搜索了更多的点，从而提高了搜索效率。例如，在解决大规模组合优化问题时，群体搜索策略可以使遗传算法在广阔的解空间中快速定位潜在的优质解区域，避免陷入局部最优解。

3.3.4 概率搜索机制

遗传算法是一种基于概率的搜索技术。遗传算法属于自适应概率搜索技术，其选择、交叉、变异等运算都是以一种概率的方式来进化的，从而增加了其搜索过程的灵活性。虽然这种概率特性也会使群体中产生一些适应度不高的个体，但随着进化过程的进行，新的群体中总会更高概率地产生出优良的个体。与其他一些算法相比，遗传算法具有较强的鲁棒性，在参数设置不完全合理时仍能维持一定搜索性能。例如，在参数设置不太合理的情况下，遗传算法仍然能够通过概率机制逐步调整搜索方向，找到较好的解，而不会完全失效。

3.3.5 自组织、自适应和学习特性

遗传算法具有自组织、自适应和学习等特性。当遗传算法利用进化过程获得信息进行自身组织时，适应度大的个体具有较高的生存概率，并获得更适应环境的基因结构。同时，遗传算法具有可扩展性，易于同其他的算法相结合，形成综合双方优势的混合算法。例如，在与局部搜索算法结合时，遗传算法可以利用其全局搜索能力快速定位潜在解区域，然后由局部搜索算法进行精细优化，从而在全局和局部两个层面提高优化效果。

3.4 遗传算法的构成要素

遗传算法作为一种模拟自然进化过程的优化算法，其核心构成要素包括种群与个体、编码方法、遗传算子、适值函数以及选择策略等。这些要素相互协作，共同驱动着遗传算法的搜索优化过程。以下对遗传算法的核心构成要素展开说明，技术实现细节将在后续章节深入探讨。

3.4.1 种群与种群规模

在遗传算法框架下，种群由染色体构成，每个染色体对应问题的一个解。种群中个体的数量称为种群规模，后文统一用NP表示，常规场景中多设定为固定常数。虽然理论上更大的种群规模对算法优化更有利，但会显著增加运算耗时，因此实际应用中常将种群规模设为100～1000。特殊情形下，也可采用与遗传代数相关的动态种群规模，以此提升优化效果。

3.4.2 编码方法

编码方法又称基因表达方法。遗传算法运行时，种群内每个个体（即染色体）由基因组成。因此，建立染色体与优化问题解的映射关系，需通过基因完成，即对染色体进行精准编码。准确编码染色体以表示问题解，是遗传算法的基础工作，也是决定算法效果的关键环节。常见的编码方式包括二进制编码、实数编码等。二进制编码通过二进制位串表示个体，例如，在求实数区间$[0，4]$上函数$f(x)$的最大值问题中，我们可以由长度为6的串表示变量x，即从“000000”到“111111”，并将中间的取值映射到实数区间$[0，4]$内。实数编码则直接使用实数值表示个体，实数编码的优点是计算精确度高，便于和经典连续优化算法结合，适用于数值优化问题；但其缺点是适用范围有限，通常只能用于连续变量问题。

3.4.3 遗传算子

遗传算子包含交叉和变异，其模拟了生物繁衍后代的过程，是遗传算法的核心机制。

交叉作为遗传算法中最重要的算子，交叉操作作用于两个染色体，融合二者特征生成新后代。以二进制编码为例，简单交叉方式是在双亲染色体上随机选取断点，交换断点右侧片段，形成两个新个体。遗传算法的性能很大程度依赖交叉运算的效率，而双亲染色体是否执行交叉由交叉率（记为P_c）控制。交叉率定义为各代中通过交叉产生的后代数量与种群个体总数的比值。较高的交叉率可扩大解空间搜索范围，降低算法停滞于非最优解的概率；但交叉率过高，会因搜索冗余解空间消耗大量计算资源。

变异指染色体自发产生的随机变化，简单形式如替换单个或多个基因。在遗

传算法中，变异可引入初始种群未包含的基因，或恢复选择过程中丢失的基因，为种群补充新内容。染色体是否发生变异由变异率（记为P_m）控制，其定义为种群中变异基因数在总基因数中的占比。若变异率过低，有用基因难以进入选择环节；若变异率过高，随机变化过大会导致后代失去从双亲继承的优良特性，使算法丧失从历史搜索中学习的能力。

3.4.4 适值函数

适值函数是衡量个体优劣的标准，它将个体的解映射到一个非负的适应度值，且适应度越大表示个体越优秀。适值函数的设计应紧密结合问题本身的要求，例如，在求解最大化问题时，适值函数可直接取目标函数值；而在求解最小化问题时，可以对目标函数值进行倒数或其他转换。

3.4.5 选择策略

选择策略决定了如何从当前种群中挑选个体以生成交配池，进而产生下一代。正比选择策略是常用的一种方法，它根据个体的适应度值所占比例来决定其被选中的概率，体现了“适者生存，优胜劣汰”的进化思想，确保优良基因能够遗传给下一代。

3.4.6 停止准则

停止准则是决定遗传算法何时终止的条件。常见的停止准则包括达到预定的最大迭代次数、种群多样性降到一定程度或找到满足精度要求的解等。最大迭代次数是一种简单直接的停止方式，但在实际应用中，通常会结合其他条件，以确保算法在找到满意解后及时停止，避免不必要的计算。

3.5 遗传算法的流程

以 Holland 提出的基本遗传算法（GA）为例，详细阐述算法的具体实现过程，其算法流程如图所示。

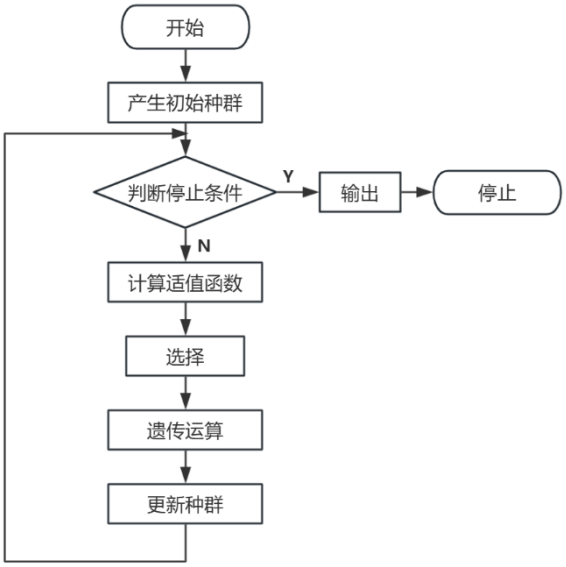

图 3.1 遗传算法流程图

由图 3.1 可以看到遗传算法实现中的各个步骤包括：（1）初始种群的产生；（2）编码；（3）计算适值函数；（4）遗传运算；（5）选择；（6）判断是否达到停止准则。下面就对各个步骤的实现进行具体说明。

3.5.1 初始种群的产生

初始种群通过随机生成，具体生成方式取决于所采用的编码方法。种群的规模受到计算机性能和计算复杂度的影响。例如，0-1 编码的生成方法如下：

随机产生$\xi_i \sim U(0,1)$，若$\xi_i > 0.5$，则$x_i = 1$。若$\xi_i \leqslant 0.5$，则$x_i = 0$。

3.5.2 编码方法——二进制编码

每个染色体可以表示为$X = |(x_1, x_2, \cdots, x_n), 1 \leqslant i \leqslant n$。

染色体的每一位，即x_i是一个基因。每一位的取值称为位值。n称为染色体的长度。一般的遗传算法使用二进制编码，即使用固定长度的$0-1$字符串表示一个染色体，例如$X = (010110)$，就可以表示一个染色体，该个体的染色体长度为$n = 6$。

下面举例说明二进制编码适用情况。

（1）背包问题

n个物品，对物品i，价值为p_i，质量为ω_i，背包容量为 W。如何选取物品装入背包，使背包中的物品的总价值最大。

其编码如下所示：

$$\mathrm{x_i} = \begin{cases} 1, \text{装入物品 i} \\ 0, \text{不装入物品 i} \end{cases}$$

（2）指派问题

指派问题是一类特殊的线性规划问题，其中每个工作与资源的需求是一一对应的。每种资源（如机器、时间段）都被唯一分配给一项工作（如位置、事件）。当资源$i(i = 1,2,\cdots,n)$被分配给工作$j(j = 1,2,\cdots,n)$时，会产生一个相应的费用c_{ij}。该问题的目标是寻找一种资源分配方式，使得总费用最小。通过 0-1 编码的变量x_{ij}来表示资源与工作的对应关系，如下所示：

$$x_{ij} = \begin{cases} 1, \text{资源} i \text{指派给工作} j \\ 0, \text{其他} \end{cases}$$

对于二进制编码，它的缺点在于编码长度较长，可能不利于计算效率，而其优点在于便于进行位值计算，并且能够表示较大的实数范围。

3.5.3 适值函数

在进化论中，适应度通常用于描述一个体对环境的适应能力，并且体现该个体在繁殖后代方面的潜力。对于遗传算法而言，适值函数，也称为评价函数，是用来评估群体中个体优劣的关键指标。该函数通常基于问题的目标函数进行设计，借此评估个体在当前环境下的适应度。在遗传算法的进化过程中，适值函数是唯一依赖的信息源，除了这个评估函数外，一般不需要其他外部数据。通过评估每个个体的优劣，适值函数为遗传操作（如选择、交叉和变异）提供了决策依据。

由于遗传算法的选择过程依赖于个体的适应度，因此，适值函数的值必须是正值，并且在搜索过程中通常要求对个体进行排序，以便计算选择概率。这也表明，在许多情况下，适值函数需要将目标函数转化为求最大值的问题，并确保该函数值是非负的。

适值函数的设计必须满足一定的条件，以确保其有效性。首先，适值函数应

具有单一的值，且是连续、非负，并能被最大化。其次，适值函数必须具备合理性和一致性，使其能够真实地反映个体的优劣。计算效率也是一个重要的设计因素，适值函数的计算量应尽量小，以提升遗传算法的执行速度，尤其在处理大规模问题时尤为重要。最后，适值函数应具有较强的通用性，能够适应不同问题的求解需求。

在实际应用中，适值函数的设计需要根据具体问题的要求来定制，它直接影响到遗传算法的性能和效率。因此，合理的适值函数设计不仅能够提高算法的收敛性，还能有效地提升问题求解的效果。

3.5.4 遗传运算

遗传运算，即交叉和变异，是遗传算法的精髓，也是变化最多的地方。下面就来介绍交叉和变异的具体实现。

（1）交叉

交叉操作中，使用最多的是单点交叉和双点交叉。

Ⅰ.单切点交叉

单切点交叉是由 Holland 提出的最基础的一种交叉方式。从种群中选出两个个体P_1和P_2，随机选择一个切点（Cutting），将切点两侧分别看作两个子串，将右侧的子串分别交换，则得到两个新的个体C_1和C_2，这里，P_1和P_2称为父代染色体，C_1和C_2称为子代染色体。

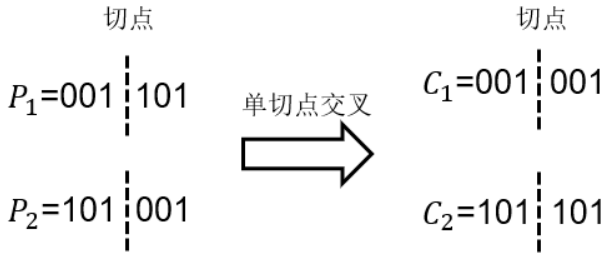

图 3.2 单切点交叉

单切点交叉操作的信息量较少，交叉点的位置选择可能导致较大偏差，同时染色体的末端基因总是会被交换。在实际应用中，双切点交叉被更广泛地采用。

Ⅱ.双切点交叉

对于两个选定的染色体P_1和P_2，随机选取两个切点，交换两个切点之间的子串，如下图所示。

图 3.3 双切点交叉

在算法的构成要素中已经说明，并不是所有的被选中的父代都要进行交叉操作，要设定一个交叉概率P_c，一般取为一个较大的数，比如0.9。

（2）变异

变异是指在种群中根据设定的变异概率P_m随机改变若干基因位的值，对于0-1 编码来说，通常表现为位值的反转。变异实际上是通过小概率扰动引发子代基因的变化。因此，变异概率通常设置为较小的值，通常低于 5%。

3.5.5 选择策略

最常用的选择策略是正比选择（Proportional Selection）策略，即每个个体被选中进行遗传运算的概率为该个体的适应值和群体中所有个体适应值总和的比例。对于个体i，设其适应值为F_i，种群规模为NP，则该个体的选择概率可以表示为

$$P_i = \frac{F_i}{\sum_{j=1}^{NP} F_j},$$

得到选择概率后，采用旋轮法（Roulette Wheel）来实现选择操作。令

$$PP_0 = 0$$
$$PP_i = \sum_{j=1}^{i} P_j,$$

共转轮NP次。每次转轮时，随机产生$\xi_k \in U(0,1)$，若$PP_{i-1} \le \xi_k < PP_i$，则

选择个体i。

下图展示了当 NP=10 时的示意图。整个转轮被分成多个扇面，分别代表不同的个体。每个个体的适应值占所有个体适应值总和的比例不同，这些比例决定了扇面大小。适应值较高的个体对应较大的扇面，而适应值较小的个体则对应较小的扇面。显然，转轮停在某个扇面上的概率与该扇面的圆心角大小成正比。

从图中可以看到，P_4的适应值较高，作为优良的个体，其将获得较多的繁殖机会。而P_1很可能失去繁殖的机会。

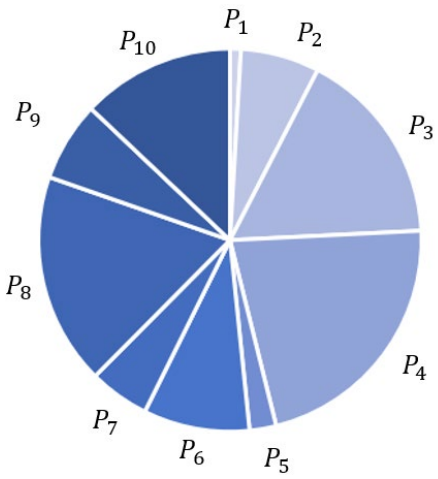

图 3.4 NP=10 时旋轮法示意图

3.5.6 停止准则

遗传算法的停止准则一般是采用设定最大代数的方法，最大代数常表示为 NG（Number of Max Generation）。

例题 3.1 求解以下的无约束优化问题

$$\max f(x) = x^3 - 45x^2 + 600x + 100\ ,\ x \in [0,30].$$

解 首先要将决策变量编码为二进制串形式的染色体。染色体的长度取决于编码精度。设染色体长度为L，问题的定义域为$[a,\ b]$，则编码精度C可以表示为

$$C = \frac{b-a}{2^L - 1},$$

对于本例题，如果要求编码精度为1，则可计算染色体长度如下:

$$\frac{30-0}{2^L-1} \leqslant 1,$$

从而得到$L \geq 5$。

所以取染色体长度为5，即可满足精度要求。下面，用求导的方法来分析该问题的解。令

$$f'(x) = 3x^2 - 90x + 600 = 3(x-10)(x-20) = 0,$$

则可以得到两个极值点:$x_1 = 10, x_2 = 20$。计算$f(x)$的二阶导数

$$f''(x) = 6x - 90,$$

当$x_1 = 10$ 时,$f''(x) < 0$，所以为极大值点。当$x_2 = 20$ 时,$f''(x) > 0$，所以为极小值点。

接着，生成初始种群，设定参数为种群规模 NP=5，最大代数 NG=10，初始时刻 t=0。接着，判断停止准则，若最大代数已达到 NG，则停止算法；否则，继续迭代。在每一代中，计算每个个体的适应值以评估其优劣。然后，使用旋轮法进行正比选择，根据个体适应值的概率进行选择，以确保优良个体更可能进入下一代。通过不断迭代更新种群，最终得到最优解或满足停止准则的结果。

在下表中列出了随机产生的初始种群及相关计算结果。S表示种群中所有个体的适应值和。$\bar{f}$表示种群中所有个体的平均适应值。

表 3.1 例题 3.1 中随机产生的初始种群及相关计算结果

染色体编号 j	编码	染色体 j 对应的解	目标函数值 $f(x_j)$	染色体 j 被选中的概率	PP_j
1	10011	19	2114	0.182	0.182
2	11010	26	2856	0.246	0.428
3	00101	5	2100	0.181	0.609
4	10101	21	2116	0.182	0.791
5	01110	14	2424	0.208	0.999

$$S = \sum f(x_j) = 11610,\ \bar{f} = \frac{S}{5} = 2322, P_j = \frac{f(x_j)}{S}$$

用旋轮法进行正比选择，根据初始种群的适应值计算得到的概率，选择时所使用的转轮如图所示。

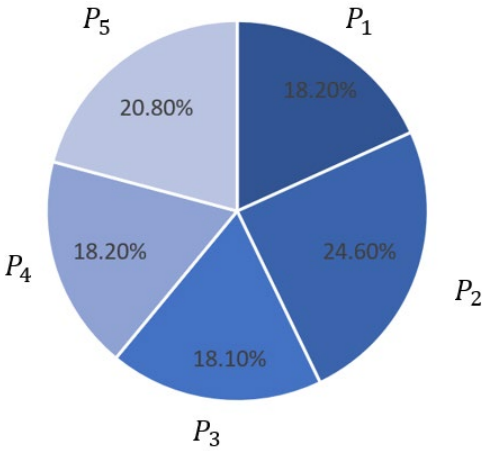

图 3.5 例题 3.1 中旋轮法示意图

对初始种群中挑选出的染色体进行遗传运算时，交叉运算采用单切点交叉，变异操作采用基本位变异操作。交叉概率P_c为0.9，变异概率P_m为0.02。

遗传运算后得到的子种群及相关计算结果列于下表中

表 3.2 例题 3.1 中子种群及相关计算结果

P_1	P_2	切点	变异否	编码	x_j	$f(x_j)$
1	2	4	N	10010	18	2152
5	3	2	N	01101	13	2492
5	2	3	N	01110	14	2424
4	2	1	N	11010	26	2856
2	5	4	N	11010	26	2856

$$S = 12780, \bar{f} = 2556$$

从上述计算结果，可以看到整个种群在改善。种群中所有个体的平均适应值从 2322 增长到 2556。

代码见电子资源。

3.6 本章小结

遗传算法作为一种模拟自然选择和遗传机制的全局优化算法，凭借其独特的全局搜索能力和灵活的求解策略，在许多复杂的优化问题中展现了强大的潜力。通过引入生物进化的概念，遗传算法以群体为基础，利用选择、交叉和变异等操作不断优化种群，最终实现问题的最优解或近似最优解的搜索。

在本章中，我们详细探讨了遗传算法的基本思想，包括初始种群的生成、适值函数的定义、选择与繁殖机制的运作以及遗传算子的实现方法。通过具体的流程步骤，遗传算法不仅能够处理各种类型的优化问题，还能自适应地调整搜索策略，确保算法的全局性和鲁棒性。此外，遗传算法具有显著的优势，尤其是在处理那些无法求导或复杂的目标函数时，能够不依赖显式数学表达式，直接进行高效的搜索。

本章还通过实例展示了遗传算法的应用过程，进一步说明了其在解决实际问题中的有效性和灵活性。从初始种群的产生到最后找到最优解，遗传算法的每个步骤都为问题的求解提供了丰富的探索空间，体现了其强大的自适应能力和全局优化潜力。

研学互通

遗传算法作为进化计算的核心分支，通过模拟自然选择与生物进化机制，为解决复杂优化问题提供了鲁棒性极强的范式。其核心思想——“适者生存”的迭代搜索，已在工程设计、人工智能、金融建模等领域展现出超越传统方法的潜力。本模块精选经典与前沿文献，帮助学习者从理论深化到实践创新，构建完整的算法认知体系。

(1) Holland, J. H. (1975). Adaptation in natural and artificial systems.

作为遗传算法的奠基之作，本书首次系统化提出“模式定理”与“积木块假说”，阐释了生物进化原理如何转化为人工系统的优化机制。读者可通过此书理解 GAs 的核心数学框架及其在适应性系统设计中的哲学基础。

(2) Goldberg, D. E. (1989). Genetic algorithms in search, optimization & machine

learning.

这篇论文规范了遗传算法的工程实现标准，详细拆解二进制编码、选择算子（如轮盘赌）和参数调优策略，并通过大量案例展示其在函数优化、机器学习中的实践路径，是掌握基础实现的必读指南。

(3) Deb, K., et al. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II.

该论文提出的 NSGA-II 算法革新了多目标优化领域，其核心创新“快速非支配排序”与“拥挤度距离比较”机制，显著提升了帕累托前沿的求解效率与分布均匀性，为工程权衡设计（如成本-性能优化）提供了通用工具。

(4) Koza, J. R. (1992). Genetic programming: On the programming of computers by means of natural selection.

开创性地将遗传算法扩展至程序自动生成领域，利用树形结构编码演化出解决符号回归、电路设计等问题的计算机程序。本书揭示了可变长度表示法的强大适应性，推动 AI 迈向自动编程的新阶段。

(5) Whitley, D. (1994). A genetic algorithm tutorial.

这篇权威教程深入浅出地解析了遗传算法的运行机理，重点探讨选择压力控制、早熟收敛预防等关键问题，并通过与传统优化方法（如梯度下降）的对比，帮助读者建立算法选择与参数设置的直觉认知。

(6) Eiben, A. E., & Smith, J. E. (2015). Introduction to evolutionary computing.

作为进化计算的教科书，该书第四章系统梳理了遗传算法的发展脉络，分析其在动态环境中的鲁棒性机制，并对比差分进化、进化策略等变体，为读者构建完整的算法演化知识图谱。

(7) Davis, L. (1991). Handbook of genetic algorithms.

这部早期手册汇集了遗传算法在工业场景的实践案例，包括生产调度、机器人控制、神经网络训练等，展示了实数编码与混合策略（如 GA-模拟退火）如何突破二进制编码局限，是学习问题建模的实用参考。

(8) Holland, J. H. (1992). Genetic algorithms.

这篇论文以通俗语言阐释遗传算法的生物学隐喻，探讨其如何揭示复杂系统的涌现规律。不仅传递核心思想，更启发读者思考自然进化与人工智能的深层联结。

思行经世：遗传算法的“适者生存”与创新精神

遗传算法作为一种模拟自然选择和遗传机制的全局优化算法，其核心理念来源于达尔文的“适者生存”理论。这一思想不仅深刻影响了生物学的演变，也为我们提供了处理复杂问题的新思维方式。在遗传算法的进化过程中，种群中的个体通过选择、交叉和变异等操作，逐渐淘汰不适应环境的解，保留和优化适应性强的解。这个过程本质上反映了自然界的进化规律，也为我们在社会发展、科技创新等领域提供了重要的启示。

在现实社会中，科技创新的过程与遗传算法的进化过程类似。每一次技术突破，都是通过无数次的试错、失败和优化实现的。例如，在中国的科技发展历程中，从最初的“引进来”到如今的“走出去”，中国企业通过不断吸收、创新和自我优化，逐渐走向世界舞台的中央。华为在 5G 技术的研发中，经过了十年的积累和无数的技术迭代，最终推动中国通信标准走向全球，这一过程就如同遗传算法中的“适者生存”，通过不断优化自己的技术和解决方案，最终找到了最适合的路径。

这一过程也提醒我们，在面对社会发展的复杂问题时，要有“长期主义”的思维方式。遗传算法的“迭代进化”告诉我们，科技创新不仅仅是单点突破，更需要整个社会在长期积累中进行不断调整和完善。比如我国在新能源领域的布局，不仅是通过短期的政策引导，更是通过多方协作、持续创新，逐步建立起自己的技术优势。

此外，遗传算法中的“群体协作”也为我们提供了宝贵的经验。在今天这个信息化、全球化的时代，单打独斗已经不再是解决问题的有效方式。无论是企业的技术合作，还是国家间的科技联盟，协作和开放已经成为推动创新和进步的关键。例如，大国之间在科技领域的合作尽管存在许多复杂的挑战，但也表明了全

球创新共同体的力量。合作可以使不同国家、企业利用各自的优势，共同推动科技进步。

总之，遗传算法不仅是一种优化工具，它背后的进化理念、创新精神以及群体协作的思想，深刻影响着我们理解社会发展和科技创新的方式。作为新时代的学习者，我们应当在继承和创新中不断进步，不断在实践中寻找“适合”的道路，让个人成长与社会发展同步进化，为国家的科技强国梦贡献智慧与力量。

习题

习题 3.1 对于编码长度为 7 的 0—1 编码，判断以下编码的合法性。

(1)[1 0 2 0 1 1 0]；

(2)[1 0 1 1 0 0]；

(3)[0 1 1 0 0 1 0]；

(4)[0 0 0 0 0 0 0]；

(5)[2 1 3 4 5 7 6]；

习题3.2 假设选择了两个父代个体，父代1为10101011，父代2为11010010。使用单点交叉方法，交叉点为第 3 位（从左数起）。请给出交叉后的两个子代个体。

习题 3.3 对于函数$\max f(x) = x^3 - 10x^2 + 100x$，$x \in [0,60]$，若采用二进制编码，个体$\boldsymbol{x} = 50$对应的染色体是什么？

习题 3.4 请用遗传算法完整求解该无约束优化问题$f(x) = 2x - 10sin(x) - 2cos(2x)$，$x \in [0,60]$。

第 4 章 数学规划求解器

在当今高度数字化的时代，优化问题已渗透到人类生产生活的方方面面。从国防安全中的资源调度，到能源管理中的电网平衡；从制造企业的生产排程，到物流网络的路径规划；从金融市场的投资组合优化，到通信技术的频谱分配。运筹学作为一门研究如何高效决策的学科，正在通过数学建模与算法求解，为复杂系统的效率提升提供科学支撑。然而，随着现实业务场景的日益复杂，数学模型的规模与复杂度呈现指数级增长。传统的手工计算或简单工具已无法满足需求，数学规划求解器应运而生，成为连接抽象模型与实际问题解决的桥梁。求解器通过集成先进的数值计算方法和优化算法，能够自动化地解析模型、搜索可行解并验证最优性，显著提升了决策的科学性与时效性。本章首先对数学规划求解器进行简要概述，随后以 Gurobi 商业求解器为例，系统阐述其在 Python 环境下的安装配置与建模求解过程。

4.1 数学规划求解器概述

数学规划求解器可分为开源求解器、商业求解器和集成软件内置求解器。以 MATLAB 的 Optimization Toolbox、Excel 的 Solver 插件为代表的集成软件，通过预置数学规划求解功能，成为教育科研领域的常用工具，但是在求解问题的规模、效率或方法上存在一定限制。例如，MATLAB Optimization Toolbox 集成序列二次规划算法，擅长处理中小规模非线性规划问题，当变量规模超过 5000 时计算效率显著下降。Excel Solver 是基于单纯形法的桌面级工具，免费版可求解变量数小于 200、约束条件小于 100 的线性规划、整数规划和非线性规划问题。

开源求解器包括 SCIP、HiGHS、OR-Tools、COIN-OR Linear Programming（CLP）、Soplex、Google Linear Optimization Package（GLOP）、GNU Linear Programming Kit（GLPK）等。其中，SCIP 由柏林的 ZIB 机构开发，被誉为求解混合整数规划（MIP）和混合整数非线性规划（MINLP）问题最快的非商业求解器之一；HiGHS 由爱丁堡大学 Julian Hall 教授团队开发，擅长处理大规模稀疏型线性优化问题，也支持线性规划（LP）、凸二次规划（QP）和混合整数规划（MIP），是目前性能最好的开源求解器。开源求解器通过社区协作模式，为科研人员和小

型企业提供了低成本甚至免费的选择。尽管它们在功能完整性和技术支持上存在不足，但其开放性与灵活性仍推动了运筹学技术的普及。

对比开源求解器，商业求解器具备全职开发团队，能够快速响应和全方位支持算法调优和硬件配置，在求解速度、求解能力上往往更优秀。此外，商业求解器往往可提供更丰富的功能如多平台支持、多样化 API 接口、云计算集成和高级优化技术。目前，全球商业求解器三巨头仍然是 Gurobi、Cplex 和 Xpress。1983 年，Xpress 由英国公司 Dash Optimization 开发，于 2008 年被美国公司 FICO 收购。IBM ILOG CPLEX Optimization Studio（CPLEX）由美国数学家 Robert E. Bixby 开发，于 1988 年由 CPLEX 商业出售，2003 年被 IBM 收购并整合到 IBM ILOG CPLEX Optimization Studio 中。Gurobi 成立于 2008 年，以其创始人顾宗浩、Edward Rothberg 和 Robert Bixby 三人的姓命名，是美国 Gurobi Optimization 公司开发的新一代大规模优化器。凭借多年的算法积累和商业经验，三巨头占据了国际市场 80%~90%的份额，并在求解线性规划、整数规划和混合整数规划问题上处于前三名。除了这三款商业求解器，市场上还有其他一些高性能的商业数学规划求解器，包括 Mosek、COPT、MindOpt、LINGO、BARON 等。

数学规划求解器市场几乎被欧美国家垄断，一旦遭遇技术封锁，工业制造、国防安全等关键领域将面临系统性瘫痪风险。近年来，为了应对全球竞争带来的“卡脖子”风险，在国家“十四五”规划与“创新驱动发展”战略指引下，中国团队陆续开发了具有自主知识产权的国产求解器，包括中科院团队开发的 CMIP、杉数科技的 COPT、华为 AI 求解器 OptVerse，以及阿里巴巴达摩院的 MindOpt。这些自主创新的求解器不仅展现出与国际领先产品相媲美的实力，而且在实际应用中展现了显著的价值。例如，杉数科技基于 COPT 为中国南方电网电力现货市场示范出清系统打造了一套智能出清优化系统，使整体出清计算时间效率提升 70%，并有效降低了平均发电成本。阿里达摩院与中国南方电网电力调度控制中心合作发布“电力调度智能决策平台”，在保证总体效益的同时，帮助南网总调实现从 15 分钟到秒级的调度。

数学规划求解器的应用边界也在不断拓展。以 Gurobi 为代表的商业求解器，凭借其全类型优化问题的处理能力与多目标优化、分布式计算等高级功能，持续

引领行业技术革新。目前，Gurobi 求解器在全球数学规划优化器测评排行中多项位列第一，在科研和业界被广泛使用。应用最广泛的数学规划问题类型包括混合整数线性规划（MILP）、混合整数二阶锥规划（MISOCP）、混合整数二次凸规划（凸 MIQP/MIQCQP）、混合整数二次非凸规划（非凸 MIQP/MIQCQP）以及混合整数非线性规划（MINLP）。本章以 Gurobi 为例，介绍其安装、建模、优化求解的具体过程。

4.2 Gurobi 学术版安装

Gurobi 支持多种平台，包括 Windows、Linux、ARM-Linux、MacOS 等，并且提供了方便轻巧的接口，支持 C、C＋＋、Java、Python、.Net、MATLAB 和 R 等编程接口，内存消耗少。本章以 Gurobi 9.5.2 为例，说明在 Anaconda 3.7 环境下的 Gurobi 的离线安装过程，包括下载、安装到 Python 环境、激活、配置环境变量、测试五部分内容。

4.2.1 软件下载

在安装 Gurobi 前，需要注意所下载的版本是否与当前的 Python 环境兼容。Gurobi 9.5.2 支持的 Python 版本包括 3.7、3.8、3.9 和 3.10。读者可访问 Gurobi 官网（www.gurobi.com），根据 Python 环境下载对应的 Gurobi 版本，并将其安装到本地计算机。安装过程中，如果安装向导提示是否重启电脑，可以选择不重启（No）。

4.2.2 Python 模块库安装

软件下载安装到本地后，找到安装目录地址，即“./gurobi952/win64”，见图 4.1。

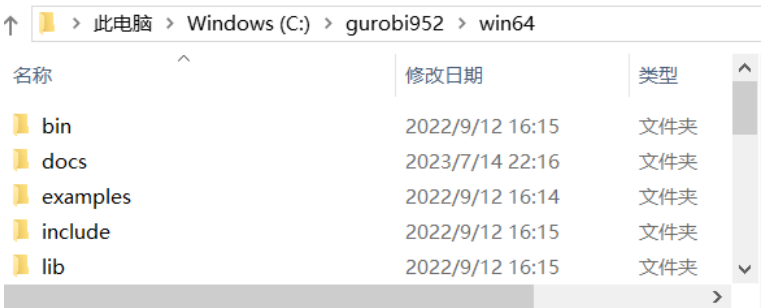

图 4.1 安装目录

首先，进入到激活环境的 Python 命令行窗口（如果安装到 Anaconda，以管理员权限进入到 Anaconda Prompt 窗口并进入激活环境；如果安装到 Pycharm，点击 Pycharm 界面左下角 Terminal 打开命令行窗口，进入对应环境；如果是其他 IDE，进入激活环境的命令行窗口），然后进行以下操作：

步骤一：进入到 Gurobi 软件下载后的安装目录，即“./gurobi952/win64”。具体来说，若安装目录与当前 Python 激活环境的盘符一致（例如，本示例中安装目录位于 C 盘，Anaconda Prompt 的环境同样位于 C 盘），则可以忽略步骤一。若不一致，输入安装目录所在的盘符加冒号。例如，“C:”，“D:”，“E:”，如图 4.2 第一行所示。

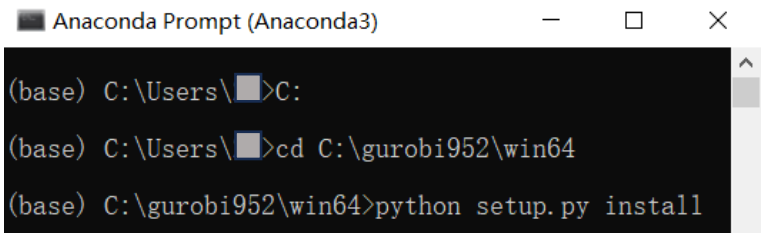

图 4.2 Python 模块库安装过程

成功切换盘符后，通过 cd 操作切换至安装目录。若安装目录的绝对地址为“C:\gurobi952\win64”，则输入“cd C:\gurobi952\win64”，如图 4.2 第二行所示。

步骤二：Python 模块库安装。输入“python setup.py install”，如图 4.2 第三行所示。运行后可以发现 gurobipy 被成功安装到“./Anaconda/Lib/site-packages”文件夹。

4.2.3 激活

Gurobi 为学校教师和学生免费提供使用许可。符合条件的读者可前往官网（www.gurobi.com），验证学术机构 IP 地址，获取激活码，自动申请学术许可。读者也可前往 Gurobi 中国官网（www.gurobi.cn）申请免 IP 学术许可，获取激活码。

首先，在联网状态下，打开命令提示符（cmd）。然后，将当前目录切换至

“./gurobi952/win64/bin”文件夹，其操作与 Python 模块库安装的步骤一类似。接下来，在命令行中输入激活码，具体操作如图 4.3 的第三行所示。需要注意的是，每个用户所申请的激活码一旦被激活，便不可再次使用。如果激活码已过期，请重新申请新的激活码。

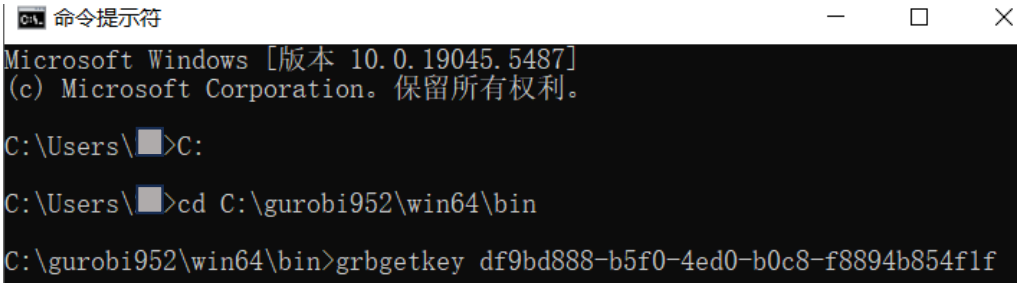

图 4.3 激活过程

运行第三行命令后，根据图 4.4，命令提示符界面中输出信息提示[hit Enter to store it in C:\Users\xxx]，要求指定学术许可文件“gurobi.lic”的下载地址。若点击“Enter”，则选择系统默认文件地址。若读者想要将该文件下载至其他地址，可在提示信息[hit Enter to store it in C:\Users\xxx]后，输入下载地址。本示例点击“Enter”选择默认地址后，许可文件被下载到本地“C:\Users\xxx\gurobi.lic”。

注意，若下载地址处已有“gurobi.lic”文件，命令提示符输出信息提示“Continue?”。若输入“Y”，则会生成新的“gurobi.lic”文件并覆盖原同名文件，若输入“n”，则放弃生成许可文件。

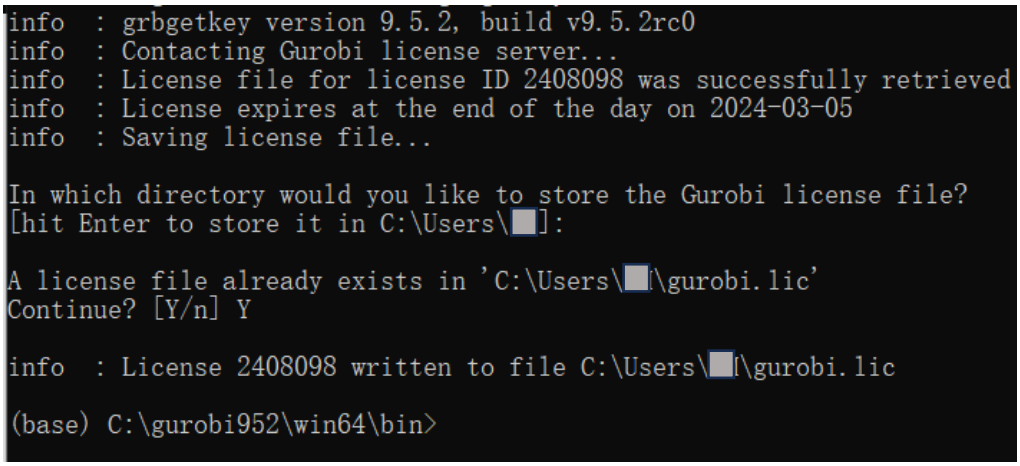

图 4.4 输入许可文件下载地址

4.2.4 配置环境变量

进入计算机“设置”，点击“系统设置”，点击“关于”，选择“高级系统设置”，见图 4.5。

图 4.5 高级系统设置

首先，配置许可文件环境变量。具体来说，点击“环境变量”，在“系统变量”下，点击“新建”。如图 4.6 所示，变量名为“GRB_LICENSE_FILE”，变量值为“gurobi.lic”文件的存储位置（本示例为“C:\Users\xxx\gurobi.lic”）。点击“确定”。

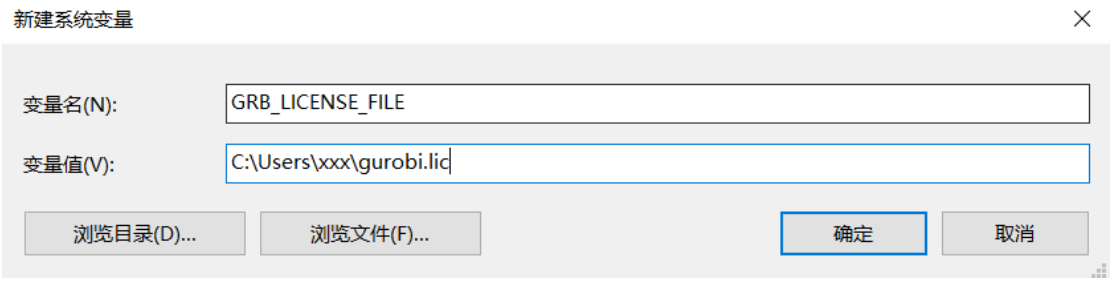

图 4.6 新建系统变量 1

然后，配置 Gurobi 环境变量。在“系统变量”下，再次点击“新建”。如图 4.7 所示，设置变量名为“GUROBI_HOME”，变量值为 gurobi 安装目录（本示例为“C:\gurobi952\win64”），点击“确定”。

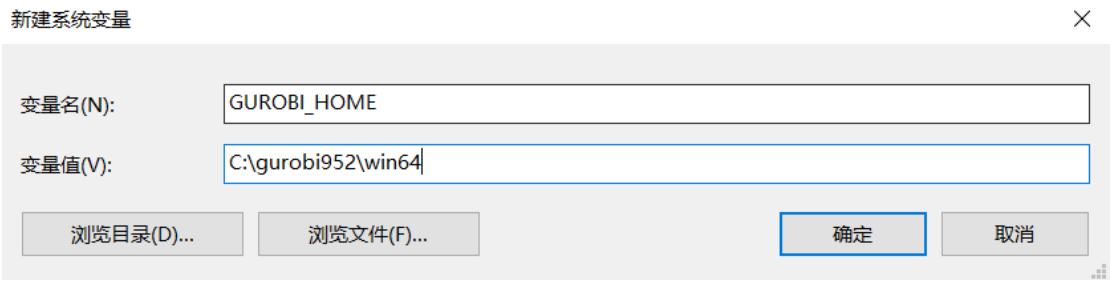

图 4.7 新建系统变量 2

自动返回至“系统变量”窗口，点击“确定”；自动返回到“环境变量”窗口，点击“确定”。最后，重启电脑。

4.2.5 测试

重启电脑后，在 Python 中运行下列代码进行安装测试。若没有报错，则安装成功。

示例 4.1 安装测试。

```
import gurobipy as gp

from gurobipy import GRB

m = gp.Model()
```

4.3 Gurobi 入门

4.3.1 基本流程

本章以模型（4.1）为例，说明如何使用 Python 调用 Gurobi 求解该问题。基本流程包括：创建模型对象、定义决策变量、设置目标函数、添加约束条件、设置求解参数、优化求解和查看求解结果。

$$\begin{aligned} \max\ & 3y_1 + 5y_2 \\ \text{s.t.}\ & y_1 + 2y_2 \le 4 \\ & 3y_1 - y_2 \ge 1 \\ & y_1 \in R_+ \\ & y_2 \in \{0,1\} \end{aligned} \tag{4.1}$$

（1）创建模型对象

Model()函数可用于创建一个 Gurobi 模型对象，该函数的基本语法格式如下。其中，参数 name 用于指定模型的名称，该名称将以 ASCII 字符串形式存储，为可选参数。

```
Model(name = ' ')      #此处只说明了部分参数
```

示例 4.2 实现模型对象创建。

```
import gurobipy as gp
from gurobipy import GRB
model = gp.Model('new_model')
```

上述代码用于创建一个模型对象 model，其名称为 new_model。注意，在 Python 中调用 Gurobi 求解器前，需要导入 gurobipy 模块。

（2）定义决策变量

Gurobi 建模时，两个常见函数 addVar()和 addVars()均可向创建的模型中添加决策变量。addVar 方法每次可添加一个变量，其基本语法格式如下：

```
addVar(lb = 0.0, ub = float('inf'), obj=0.0, vtype = GRB.CONTINUOUS)
#此处只说明了部分参数
```

上述函数的参数含义如下：

- lb：定义新变量取值的下界，默认值为 0；
- ub：定义新变量取值的上界，默认值为正无穷大；
- obj：定义该变量在目标函数中的系数，默认值为 0；
- vtype：定义变量类型，默认值为 GRB.CONTINUOUS（连续型变量），可选项包括：GRB.CONTINUOUS（连续型）、GRB.BINARY（0-1 整数）、GRB.INTEGER（整数）、GRB.SEMICONT（半连续型）。

示例 4.3 使用 addVar 添加变量。

```
y_1 = model.addVar(lb = 0, ub = GRB.INFINITY, vtype = GRB.CONTINUOUS)
y_2 = model.addVar(vtype = GRB.BINARY)
```

上述代码先后向模型 model 中添加了变量 y_1 和 y_2。通过设置 lb、ub、vtype 的取值，定义了两个变量的取值范围，即$y_1 \in R_+$，$y_2 \in \{0,1\}$。注意，若不设置函数中的参数取值，则采用系统默认设置。

addVars 方法可批量添加多个决策变量，其基本语法格式如下，其中参数*indices 用于定义批量创建的变量规模，其余参数含义与 addVar 函数的参数类似。

```
addVars(*indices, lb = 0.0, ub = float('inf'), obj=0.0, vtype = GRB.CONTINUOUS)
#此处只说明了部分参数
```

当同时创建多个决策变量时，若这些变量的某个参数取值均相同，则相应参数取值可设为一个标量。若某个参数取值不同，则相应参数可采用一个列表赋值（列表非唯一赋值方法）。

示例 4.4 使用 addVars 批量添加变量。

```
y = model.addVars(2, lb = 0, ub = [GRB.INFINITY, 1], vtype = [GRB.CONTINUOUS, GRB.BINARY])
```

运行上述代码，同时创建 2 个决策变量 y[0]和 y[1]。二者的 lb 参数取值均值 0，参数 ub 采用列表[GRB.INFINITY, 1]赋值，表示 y[0]变量的取值上界为正无穷大，y[1]变量的取值上界为 1。参数 vtype 取值类似。

示例 4.5 addVars 方法创建多维变量的结果输出。

```
x = model.addVars(2, 3)
print(x)
```

注意，示例 4.5 创建了一个 2 行 3 列的二维变量 x，每一个变量分量 x[0,0]、

x[0,1]、x[0,2]、x[1,0]、x[1,1]和 x[1,2]均为正连续型变量。

（3）设置目标函数

setObjective()是最常见的设置目标函数的方法，其基本语法结构如下：

```
setObjective(expr, sense = GRB.MINIMIZE)
```

上述函数的参数含义如下：

（1）expr 定义目标函数表达式，要求单目标只能是线性或者二次表达式；

（2）sense 定义优化方向，可取值 GRB.MINIMIZE 或 GRB.MAXIMIZE，前者使目标函数最小化，后者使目标函数最大化。

示例 4.6 setObjective 方法设置目标函数。

```
model.setObjective(3*y_1 + 5*y_2, sense = GRB.MAXIMIZE)
```

setObjective 并非设置目标函数的唯一方法。添加决策变量时（addVar，addVars），obj 参数可定义该变量在目标函数中的系数。尤其地，当目标函数是一次线性函数时，该方式被普遍使用。然而，设置 obj 参数只定义了目标函数表达式，无法指定优化方向，此时模型采用默认优化方向（GRB.MINIMIZE）。因此，若待优化问题是最大化问题，需要将其转化为最小化问题，更新变量的目标函数系数。

示例 4.7 设置 obj 参数定义目标函数。

```
y_1 = model.addVar(obj = -3, vtype = GRB.CONTINUOUS)
y_2 = model.addVar(obj = -5, vtype = GRB.BINARY)
```

示例 4.7 展示了在创建变量时通过设置 obj 参数来定义目标函数。模型（4.1）的目标函数为 max 3y_1+5y_2，此时需要转化为 min -3y_1-5y_2。然后，定义 2 个变量 y_1 和 y_2 的 obj 参数分别为-3 和-5。

（4）添加约束条件

addConstr()方法可模型添加一条约束，其基本语法结构如下。其中，expr 定义约束表达式，支持≤，≥和==关系。

```
addConstr(expr)      #此处只说明了部分参数
```

示例 4.8 使用 addConstr 方法添加约束条件。

```
model.addConstr(y_1 + 2*y_2 <= 4)
model.addConstr(3*y_1 - y_2 >= 1)
```

上述 2 行代码向模型 model 中分别添加了约束 y_1+2y_2≤4 和 3y_1-y_2≥1。由于 addConstr 方法每次只能添加一条约束，因此当模型的约束比较多时，需要调用多次该方法。

addConstrs()方法可批量添加多个约束条件，适合基于矩阵的建模方式，其基本语法结构如下：

```
addConstrs(generator)      #此处只说明了部分参数
```

示例 4.9 使用 addConstrs 方法添加约束条件。

```
A = [[1, 2], [-3, 1]]
b = [4, -1]
m = 2
n = 2
y = model.addVars(n, lb = 0, ub = [GRB.INFINITY, 1], vtype = [GRB.CONTINUOUS, GRB.BINARY])
model.addConstrs(sum(y[j]*A[i][j] for j in range(n)) <= b[i] for i in range(m))
```

示例 4.9 将模型（4.1）重表示为矩阵表达的形式，把约束条件重写为 **Ay**≤

b。首先，创建变量 **y**；然后调用 addConstrs 方法，通过 for 循环迭代，批量生成并添加 m 条约束。

（5）设置求解参数

在优化求解之前，可设置模型求解的相关参数（非必选项），用于影响 Gurobi 的优化求解过程。可选择的参数详见官方文档。

示例 4.10 设置求解参数。

```
model.Params.TimeLimit = 600
model.Params.OutputFlag = 0
```

示例 4.10 设置参数“model.Params.TimeLimit = 600”用于控制求解时间不超过 600 秒。设置“model.Params.OutputFlag = 0”，表示不输出求解过程；若取值 1 表示输出求解过程。

（6）优化求解

optimize()方法用于求解模型，status 属性用于查看模型的求解状态。状态值包括 GRB.OPTIMAL（找到全局最优解），GRB.INF_OR_UNBD（模型不可行或有无界解），GRB.INFEASIBLE（模型不可行），GRB.UNBOUNDED（模型有无界解）和 GRB.TIME_LIMIT（达到时间限制）等。

示例 4.11 优化求解并输出结果。

```
model.optimize()
if model.status == GRB.OPTIMAL:
    print(f"最优解: {model.objVal}")
    print(f"opt_y = {y_1.x}, opt_z = {y_2.x}")
elif model.status == GRB.INF_OR_UNBD:
    print('模型不可行或有无界解')
elif model.status == GRB.INFEASIBLE:
    print('模型不可行')
elif model.status == GRB.UNBOUNDED:
    print('模型有无界解')
else:
    print('模型状态为%d' % model.status)
```

示例 4.11 首先对模型求解，然后判断模型的求解状态“model.status”。若当前求解状态为“GRB.OPTIMAL”，则打印最优目标函数值“model.objVal”。变量 y_1 和 y_2 的最优值可通过属性“x”查看，即“y_1.x”和“y_2.x”。其中，“objVal”是模型对象的属性，“x”是变量对象的属性。

示例 4.12 模型（4.1）完整建模求解过程。

```
import gurobipy as gp
from gurobipy import GRB
model = gp.Model("new_model")
y_1 = model.addVar(lb=0, ub=GRB.INFINITY, vtype=GRB.CONTINUOUS)
y_2 = model.addVar(vtype=GRB.BINARY)
model.setObjective(3*y_1 + 5*y_2, sense = GRB.MAXIMIZE)
model.addConstr(y_1 + 2*y_2 <= 4)
model.addConstr(3*y_1 - y_2 >= 1)
model.Params.TimeLimit = 600
model.Params.OutputFlag = 0
model.optimize()
```

读者可根据本章节介绍的多种创建决策变量、约束条件、目标函数的方式，尝试其他建模求解方法。

4.3.2 常见属性

（1）变量的常见属性

在 addVar、addVars 等方法创建完决策变量后，若需要修改变量的相应属性值，可直接对其属性重新赋值。

示例 4.13 修改变量属性值。

```
m = gp.Model()
x = m.addVar(lb = -4, ub = 3, vtype = GRB.CONTINUOUS, name = "x")
x.Lb = 1       # 修改下界
x.Ub = 5       # 修改上界
x.VarName = "new_x"       # 修改变量名称
x.Obj = 2       # 修改目标函数系数
```

示例 4.13 首先创建了模型 m，并通过 addVar 方法定义决策变量 x。然后，展示了如何修改变量 x 的属性，包括上界、下界、名称和目标函数系数。值得一提的是，在 Python 调用 Gurobi 时，属性名可以小写，但是方法名需要区分大小写。例如，x.lb, x.ub, model.addVars()等。

示例 4.14 查看变量的其他属性值（假设 x 是决策变量）。

```
m.optimize()
print('Optimal value of x:', x.X)       # 查看变量值
print('Reduced cost of x:', x.RC)       # 查看缩减成本
```

当模型求解结束后，属性“X”用于查看变量的最优值。若模型是线性规划，那么属性“RC”可查看变量的缩减成本（reduced cost）。

（2）约束的常见属性

向模型添加完约束条件后，若需要修改约束的相应属性值，可直接对其属性重新赋值。

示例 4.15 修改约束属性值。

```
m = gp.Model()
x = m.addVar(lb = -4, ub = 3, vtype = GRB.CONTINUOUS, name = "x")
model.addConstr(x >= 1, name = "constraint1")
constr = model.getConstrByName("constraint1")       # 根据约束名称，获取约束
constr.RHS = 4       # 修改约束右值
```

示例 4.15 展示了如何改变约束的右值。具体来说，使用模型的 getConstrByName 方法，根据约束的名字获取对应约束 constr，并修改该约束的右值。

示例 4.16 查看约束的其他属性值（假设 constr 是一条约束）。

```
m.optimize()
print('Slack for constraint1:', constr.Slack)       # 查看松弛变量值
print('Dual price for constraint1:', constr.Pi)      # 查看影子价格
```

示例 4.16 查看约束的“Slack”和“Pi”属性，前者返回对应该约束的松弛变量值，后者返回对应该约束的影子价格。

4.3.3 数独问题实战

章节 4.3.1 介绍了 Gurobi 建模求解的基本流程，最后给出了基于 addVar 和 addConstr 方法的完整建模代码。本节以数独问题为例详细介绍如何使用 addVars 和 addConstrs 方法。

给定一个 9×9 矩阵，该矩阵被分为 9 个 3×3 的九宫格，部分方格中的数字已提前给定，见图 4.7。玩家需要在空白格子中填入数字 1-9，使得：（1）在每一行，数字 1-9 只能出现一次；（2）在每一列，数字 1-9 只能出现一次；（3）在每一个小的 3×3 的九宫格中，数字 1-9 只能出现一次。针对该数独问题，请找到一种满足规则要求的可行方案。

	2							
5				2				
4					7			3
8					3	4		
	1				9			
				1				
3	4		2			1		
	5	9	6	4			8	
		1		9	5			2

图 4.7 数独问题

设决策变量$x_{i,j,k}$表示第 i 行、第 j 列的方格内是否填写数字 k，定义如下：

$$x_{i,j,k} = \begin{cases} 1, \text{第 i 行、第 j 列的方格内填写数字 k} \\ 0, \text{第 i 行、第 j 列的方格内不填写数字 k} \end{cases}, \quad \forall i, j, k \in \{1, \dots, 9\}.$$

本问题找到一个可行方案即可，因此设置目标函数为一个常数，建立数学模型（4.2）：

$$
\begin{aligned}
&\min 0 \\
&\text{s.t.} \sum_{k=1}^{9} x_{i,j,k} = 1, \forall i, j = 1, \dots, 9 && (1) \\
&\sum_{j=1}^{9} x_{i,j,k} = 1, \forall i, k = 1, \dots, 9 && (2) \\
&\sum_{i=1}^{9} x_{i,j,k} = 1, \forall j, k = 1, \dots, 9 && (3) \\
&\sum_{i'=1}^{3} \sum_{j'=1}^{3} x_{i'+i,j'+j,k} = 1, \forall i, j \in \{0,3,6\}, k = 1, \dots, 9 && (4) \\
&x_{i,j,k} = 1, \forall (i,j,k) \in G && (5) \\
&x_{i,j,k} \in \{0,1\}, \forall\, i,j,k \in \{1, \dots, 9\} && (6)
\end{aligned}
\tag{4.2}
$$

约束（1）表示一个格子只能填写一个数字。约束（2）表示每一行数字 1-9 只能出现一次。约束（3）表示每一列数字 1-9 只能出现一次。约束（4）表示对于每一个小的 3×3 的九宫格，数字 1-9 只能出现一次。

约束（5）中的参数 G = [(1, 2, 2), (2, 1, 5), (2, 5, 2), (3, 1, 4), (3, 6, 7), (3, 9, 3), (4, 1, 8), (4, 6, 3), (4, 7, 4), (5, 2, 1), (5, 6, 9), (6, 5, 1), (7, 1, 3), (7, 2, 4),(7, 4, 2), (7, 7, 1), (8, 2, 5), (8, 3, 9), (8, 4, 6), (8, 5, 4), (8, 8, 8), (9, 3, 1), (9, 5, 9), (9, 6, 5), (9, 9, 2)]。G ∈{(i, j, k): i∈{1,⋯,9}, j∈{1,⋯,9}, k∈{1,⋯,9}}，记录了已填充的信息，表示第 i 行、第 j 列方格内已填有数字 k。因此，相应的决策变量取值为 1。

示例 4.17 求解数独问题。

```
import gurobipy as gp

from gurobipy import GRB

import numpy as np

G = [(1, 2, 2), (2, 1, 5), (2, 5, 2), (3, 1, 4), (3, 6, 7), (3, 9, 3), (4, 1, 8), (4, 6, 3), (4, 7, 4), (5, 2, 1), (5, 6, 9), (6, 5, 1), (7, 1, 3), (7, 2, 4),(7, 4, 2), (7, 7, 1), (8, 2, 5), (8, 3, 9), (8, 4, 6), (8, 5, 4), (8, 8, 8), (9, 3, 1), (9, 5, 9), (9, 6, 5), (9, 9, 2)]


# 步骤 1：创建模型对象

m = gp.Model("sudoku")

# 步骤 2：添加决策变量

x = m.addVars([(i, j, k) for i in range(1, 10) for j in range(1, 10) for k in range(1, 10)], vtype=GRB.BINARY)

# 步骤 3：添加约束条件

## 约束（1）

m.addConstrs((sum(x[i, j, k] for k in range(1, 10)) == 1 \

                    for i in range(1, 10) for j in range(1, 10)))

## 约束（2）

m.addConstrs((sum(x[i, j, k] for i in range(1, 10)) == 1 \

                    for j in range(1, 10) for k in range(1, 10)))

## 约束（3）

m.addConstrs((sum(x[i, j, k] for j in range(1, 10)) == 1 \

                    for i in range(1, 10) for k in range(1, 10)))

## 约束（4）

m.addConstrs((sum(x[i + i_, j + j_, k] for i_ in range(3) for j_ in range(3)) == 1 \

                    for i in range(1, 10, 3) for j in range(1, 10, 3) for k in range(1, 10)))

## 约束（5）

m.addConstrs((x[i, j, k] == 1 for i, j, k in G))

# 步骤 4：优化求解

m.optimize()

sol = np.zeros((9, 9))
```

示例 4.17 创建模型对象 m，并向模型中批量添加了 x[1,1,1]，x[1,1,2]，……，x[9,9,9]共 9×9×9 个决策变量。

接下来，以约束（1）为例，addConstrs 方法对 i 和 j 的取值嵌入 2 层 for 循环，共批量添加共 9×9 个约束条件。步入某次迭代，已知 i, j 取值，通过 sum 函数写入以下约束表达式。约束（2）和（3）添加方式类似。

$$\sum_{k=1}^{9} x_{i,j,k} = 1$$

对于约束（4），addConstrs 方法对 i，j 和 k 的取值嵌入 3 层 for 循环，共批量添加共 3×3×9 个约束条件。步入某次迭代中，已知 i，j，k 取值，通过 sum 函数写入约束表达式：

$$\sum_{i'=1}^{3}\sum_{j'=1}^{3} x_{i'+i,j'+j,k} = 1$$

对于约束（5），addConstrs 方法遍历 G 中每个元素，批量添加约束。步入某次迭代后，已知 i，j，k 取值，令$x_{i,j,k} = 1$。最后，调用优化求解，并将解决方案存储在数组 sol 中。

4.4 Gurobi 进阶

Gurobi 可以精确求解的问题类型包括：线性约束和目标模型（连续变量、混合整数）、二阶锥模型（连续变量、混合整数）、二次凸约束和目标模型（连续变量、混合整数）、二次非凸（双线性、二次等式约束、分母带变量、高阶多项式等）约束和目标模型（连续变量、混合整数）、以及非线性模型（除式、高阶多项式、指数、对数、三角函数、范数、逻辑函数等）（连续变量、混合整数）。Gurobi 还支持约束和目标中带有最大、最小、绝对值等数学函数，或者带有 AND、OR、INDICATOR 逻辑条件的模型（连续变量、混合整数）。

变量、约束和目标是 Gurobi 建模的三个核心要素。章节 4.3 简单介绍了一些基本的添加变量、约束和目标的方法，如 addVar()、addVars()、addConstr()、addConstrs()和 setObjective()。本章节为了满足更高的建模需求，对相关知识进行

补充。

4.4.1 变量

Gurobi 不区分决策变量、辅助变量、中间变量、松弛变量等。所有变量都通过 addVar()、addVars()和 addMVar()等方法来定义。其中，addMVar()方法用于创建多维数组形式的决策变量（如矩阵、向量），和 addVars()有一定相似之处，但它们适用于不同类型的需求和使用场景。下面，通过实例来对比两种方法。

（1）addVars()

该方法适合为每个组合的索引创建单独的变量。它是基于字典的方式添加变量，通常用于那些可以由一组离散的索引标识的情况。例如，如果模型中变量代表不同工厂生产不同产品的数量，可以为每种产品和每个工厂的组合创建一个变量。通过传递索引（或多个索引）给 addVars()方法，自动为所有可能的索引组合生成变量。

示例 4.18 addVars 方法创建变量。

```
model = gp.Model()
x = model.addVars({"工厂 1", "工厂 2"}, {"牛奶", "面包"})
""" 返回值：
{('工厂 2', '牛奶'): <gurobi.Var C0>,
 ('工厂 2', '面包'): <gurobi.Var C1>,
 ('工厂 1', '牛奶'): <gurobi.Var C2>,
 ('工厂 1', '面包'): <gurobi.Var C3>} """
```

示例 4.18 中，有两家工厂 1 和 2，每家工厂生产 2 种产品，分别是牛奶和面包。现通过 addVars()方法，分别为每个工厂和产品的组合创建一个变量。从变量返回值可以看出，变量 x 存储为字典形式，其中创建的每一个分量由工厂和产品的组合进行索引。

（2）addMVar()

该方法用来添加矩阵变量（或多维数组形式的变量）。当处理需要以矩阵或更高维度数组形式表示决策变量的问题时，该方法尤其适用。例如，在解决线性规划、二次规划或混合整数规划问题时，如果模型中的变量自然地形成一个矩阵或多维数组结构，那么使用 addMVar()可以更方便地定义这些变量，并简化后续对这些变量的操作。值得注意的是，addMVar()方法返回的对象支持 NumPy 切片操作。

示例 4.19 addMVar 方法创建变量。

```
model = gp.Model()
y = model.addMVar(3, vtype = GRB.BINARY)       #返回值：<(3,) matrix variable>
z = model.addMVar((3,4), vtype = GRB.BINARY)       #返回值：<(3, 4) matrix variable>
print(z[:,1:3])       #返回值：<(3, 2) matrix variable>
```

示例 4.19 使用 addMVar()方法创建了一个 3 行 1 列的矩阵变量 y 和一个 3 行 4 列的矩阵变量 z。

综上，如果变量适合用矩阵或多维数组表示，建议使用 addMVar()方法。如果变量需要根据一个或多个索引集来定义，并且希望以键值对的形式访问每个变量，建议使用 addVars()方法。

4.4.2 约束

（1）线性约束和二次约束

约束是由变量通过表达式构成的。常数*变量、常数*变量*变量的任何加总构成线性或者二次表达式。不论变量的类型（Gurobi 支持二次凸和非凸模型），只要最终展开形式不超过二次，就可以直接通过 addConstr()和 addConstrs()等函数写入模型中，和常规书写方式没有区别。此外，对于二次约束，也可以通过 addQConstr()写入模型。

如果表达式中的阶次超过二次，无法直接向模型添加约束。注意，以展开后的已定义变量的阶次为标准判断约束是否超过二次。例如，x、y、z 是已定义的

变量，且$z = x^2(x + y)$。由于约束$z <= 10$的阶次超过二次，因此不能直接通过addConstr()添加约束。

示例 4.20 降阶解决约束超过二次的问题（模型对象 model）。

```
model.addConstr(w == x**2)
model.addConstr(z == w*(x+y))
model.addConstr(z <= 10)
```

示例 4.20 展示了如何进行降阶处理，解决约束表达式中的阶次超过二次的问题。具体来说，通过 addVar 创建一个中间变量 w，将原 z 的表达式重表示为$z = w(x + y)$，从而使约束表达式降到二次，可直接向模型添加约束。

注意，线性和二次表达式中不能混入到 SOS 和广义函数表达式，往往需要创建新的中间变量，并配合约束条件，进行降阶、消除非线性项。例如，目标函数是 $\min\ x + y + z^{0.3} + \max(x, z)$这样线性、混合指数函数和取最大值函数的复杂表达式。由于 Gurobi 要求单目标表达式只能是线性或者二次表达式，此时无法通过 setObjective 方法设置目标函数。

示例 4.21 表达式消除非线性项（模型对象 model）。

```
u = model.addVar()
v = model.addVar()
model.setObjective(x+y+u+v, sense = GRB.MINIMIZE)
model.addGenConstrPow(z, u, 0.3)
model.addConstr(v == gurobipy.max_(x, z))
```

示例 4.21 说明了如何定义目标函数 $\min\ x + y + z^{0.3} + \max(x, z)$。首先，创建两个中间变量 u 和 v，通过广义约束的方式定义$u = z^{0.3}$，$v = \max(x, z)$。然后，通过 setObjective 设置目标函数表达式为 x+y+u+v 即可。

（2）广义约束

前文提到 addConstr 和 addConstrs 仅能处理线性约束和二次约束。Gurobi 接受一些额外的约束类型，我们将其统称为广义约束。Gurobi 支持的广义约束包括：最大（addGenConstrMax）、最小（addGenConstrMin）、绝对值（addGenConstrAbs）、AND（addGenConstrAnd）、OR（addGenConstrOr）、INDICATOR 逻辑函数（addGenConstrIndicator）、多项式（addGenConstrPoly）、指数（addGenConstrExp，addGenConstrExpA）、对数（addGenConstrLog，addGenConstrLogA）、幂函数（addGenConstrPow）、三角函数（addGenConstrSin、addGenConstrCos、addGenConstrTan）。广义约束具体构建方法详见官方文档。

4.4.3 回调

Gurobi 的回调（callback）允许用户自定义函数，并由 Gurobi 在优化的不同阶段定期调用该自定义函数，用于监控优化进度，调整优化器的行为。具体来说，对模型实例化后，自定义一个具有 model 和 where 两个参数的函数（也称回调函数），并将该函数作为参数向 optimize()函数中传递。

示例 4.22 回调使用方法。

```
def my_callback(model, where):
    if where == GRB.Callback.MIP:
        model._step += 1
        nodecnt = model.cbGet(GRB.Callback.MIP_NODCNT)
        solcnt = model.cbGet(GRB.Callback.MIP_SOLCNT)
        if nodecnt >= 10 and solcnt:
            model.terminate()

m = gp.Model("mip1")
x = m.addVars(2, lb = 1, ub = 10, vtype = GRB.INTEGER)
m.setObjective(3*x[0] - 5*x[1], GRB.MAXIMIZE)
m.addConstr(x[0] - 2*x[1] <= 30)
m._step = 0
m.optimize(my_callback)
```

示例 4.22 创建模型 m，并向其添加了变量、约束条件和目标函数。命令“m.optimize(my_callback)”表示在求解模型时调用 my_callback 函数，从而影响求解过程。另外，可使用“._”的方式将模型外的参数传入到 my_callback 函数中使用。例如，“m._step”是模型 m 的一个自定义属性，在 my_callback 函数中使用，用于记录某些条件下回调触发次数。

在自定义函数 my_callback(model, where)中，参数 model 表示当前求解的模型对象，参数 where 表示回调触发位置（由 Gurobi 自动传入）。触发该回调函数后，首先 if 语句判断参数 where 是否等于 GRB.Callback.MIP，仅当回调发生在 MIP 求解阶段时才会执行后续操作。“model._step”是一个计步器，用于记录触发位置是“GRB.Callback.MIP”的次数。

Gurobi 回调时，使用一对参数：where 和 what。where 参数表示在 Gurobi 优化器中调用回调函数的位置。调用该函数时，若希望获得关于优化状态的更详细信息时，可以将 what 参数传递到 cbGet()函数中。注意，what 的取值依赖于 where。

示例 4.22 将“GRB.Callback.MIP_NODCNT”传入 cbGet()函数，可返回已探索的分支定界节点数。将“GRB.Callback.MIP_SOLCNT”传入 cbGet()函数，用于返回当前找到的可行解数量。最后，若节点数不小于 10 且已经至少找到一个可行解，通过“model.terminate()”终止求解，实现干扰求解过程的目的。

Gurobi 中回调函数的参数 where 取值包括 presolve, simplex, barrier, MIP, MIPSOL, MIPNODE, MESSAGE 和 BARRIER。每个 where 取值下，可传入 cbGet 函数的参数 what 的取值有所差异，更多 where 和 what 参数的有效组合详见官方文档。

除了 cbGet 方法，模型还有一些比较重要的方法。例如，若创建一个混合整数规划的模型对象 model，优化求解回调时可能涉及获取当前节点解的值（通过 model.cbGetNodeRel 或 model.cbGetSolution 方法）、导入基于现有解构建的新启发式解（通过 model.cbSetSolution）、或在分支定切框架下添加新的约束（使用 model.cbCut 或 model.cbLazy）以改进解的质量。

4.4.4 旅行商问题实战

旅行商问题的一般性描述是指，一个推销员要到 n 个城市推销商品，他要找出一个包含所有城市的具有最短路程的环路。本章节以一个对称旅行商问题（旅行商在两个城市之间往返的旅行成本相同）为例，逐步演示如何使用 Gurobi 求解器对该问题求解，帮助读者巩固 Gurobi 建模求解方法。

[n]={1, 2, …, n}代表所有城市的集合，矩阵 **d** 记录了不同城市之间的距离。考虑到本问题的对称性，仅针对 i>j 时定义决策变量：

$$x_{ij} = \begin{cases} 1, & \text{解中包含从点 i 到点 j 的线段} \\ 0, & \text{解中不包含从点 i 到点 j 的线段} \end{cases}, \quad \forall i, j \in [n], i > j.$$

本问题的优化目标是最小化旅行成本，即

$$\min \sum_{i\in[n]} \sum_{j\in[n],i>j} d_{ij}\, x_{ij}.$$

下面约束确保每个城市恰好有两条边与之相连，即一条进入该城市，另一条离开该城市，从而形成一个闭合的环路，并且每个城市都被访问到一次。

$$\sum_{j\in[n],i>j} x_{ij} + \sum_{j\in[n],j>i} x_{ji} = 2, \forall i \in [n]$$

若仅考虑上述约束，根据决策变量的定义，至少 3 个点便可能形成子回路。因此，添加下面约束来破除子回路。

$$\sum_{i\in S} \sum_{j\in S,i>j} x_{ij} \leq |S| - 1, \forall S \subseteq [n], |S| \geq 3$$

该对称旅行商问题的完整数学模型如下：

$$\begin{aligned} \min & \sum_{i\in[n]} \sum_{j\in[n],i>j} d_{ij}\, x_{ij} \\ \text{s.t.} & \sum_{j\in[n],i>j} x_{ij} + \sum_{j\in[n],j>i} x_{ji} = 2, \forall i \in [n] \\ & \sum_{i\in S} \sum_{j\in S,i>j} x_{ij} \leq |S| - 1, \forall S \subseteq [N], |S| \geq 3 \\ & x_{ij} \in \{0,1\}, \forall i, j \in [n], i > j \end{aligned} \tag{4.3}$$

示例 4.23 随机生成旅行商问题算例。

```
import math
import random
import gurobipy as gp
from gurobipy import GRB
n = 10
random.seed(1)
points = [(random.randint(0, 100), random.randint(0, 100)) for i in range(n)]
d = {(i, j): math.sqrt(sum((points[i][k]-points[j][k])**2 for k in range(2)))
            for i in range(n) for j in range(i)}
```

示例 4.23 随机生成了一组算例，其中城市数量 n=10，列表 pionts 存储了 n 个城市的坐标。定义字典 d，用以存储任意两个城市间的欧几里得距离，其中键(i, j)表示城市 i 和城市 j 之间的边（i>j）。

（1）创建决策变量

根据章节 4.4.1，如果变量需要由一个或多个索引集来定义，并且希望以键值对的形式访问每个变量，建议使用 addVars()方法。

示例 4.24 创建决策变量。

```
m = gp.Model()
x = m.addVars(d.keys(), obj = d, vtype = GRB.BINARY)
```

示例 4.24 通过 addVars 方法对所有城市对的组合创建 0-1 决策变量 x[i, j]，并定义每个决策变量的目标系数是城市 i 和 j 间的距离。注意，模型（4.3）的目标函数表达式是线性函数，可在定义决策变量时设置参数 obj 取值，从而达到同时定义目标函数的作用。

（2）添加约束并优化求解

对于破解子回路约束，需遍历所有城市数量大于等于 3 个的集合[n]的子集，导致所涉及的约束数量呈指数级增长。本节重点介绍如何不考虑其涉及的全部约束，使用 Gurobi 的回调功能向模型中动态添加约束来消除子回路。

示例 4.25 添加约束条件。

```
m.addConstrs(sum(x[i,j] for j in range(n) if i > j) + sum(x[j,i] for j in range(n) if i < j) == 2 for i in range(n))


m._x = x
m.Params.LazyConstraints = 1
m.optimize(my_callback)
```

首先，示例 4.25 使用 addConstrs 方法向模型批量添加下面 n 条约束。

$$\sum_{j\in[n],i>j} x_{ij} + \sum_{j\in[n],j>i} x_{ji} = 2, \forall i \in [n]$$

然后，将自定义回调函数 my_callback 传入 optimize 方法。在优化求解过程中，每当 my_callback 函数识别出子回路，则向模型添加一条惰性约束（lazy constraint）消除子回路。当触发到 my_callback 函数时，若发现当前最好可行解中不存在子回路，则该问题求到最优。

注意，Gurobi 使用 cbLazy 方法向模型添加惰性约束，在优化求解之前需要设置参数“m.Params.LazyConstraints = 1”。

示例 4.26 自定义回调函数 my_callback。

```
def my_callback(model, where):
    if where == GRB.Callback.MIPSOL:
        curr_x = model.cbGetSolution(model._x)
        S = subtour(curr_x)       # 最短子回路
        if len(S) < n:
            model.cbLazy(sum(model._x[i, j] for i in S for j in S if i > j)
                         <= len(S)-1)
```

示例 4.26 自定义了名为 my_callback 的函数，它接受两个参数：model（当前优化模型）和 where（指示回调被调用的原因）。注意，使用 Gurobi 求解时，只有当回调函数上的 where 值为 GRB.Callback.MIPSOL（表示识别一个新的混合整数规划问题的最好可行解）或 GRB.Callback.MIPNODE（表示正在探索一个混合整数规划问题的节点）时，才能调用 cbLazy 方法来添加惰性约束。

命令“model.cbGetSolution(model._x)”获取当前最好解。然后，调用辅助函数 subtour 来寻找该解中存在的最小子环（读者可自行实现该函数功能，本书不具体展示）。if 条件语句用于判断是否存在子回路，若存在，使用 cbLazy 方法添加以下约束，使当前产生子回路的解不满足以下约束。通过不断触发回调函数动态添加该惰性约束，直到当前最好解中不存在子回路，则该问题求到最优。

$$\sum_{i \in S} \sum_{j \in S, i>j} x_{ij} \leq |S| - 1$$

4.5 本章小结

本章首先对数学规划求解器进行了概述，然后以 Gurobi 求解器为例，系统说明其安装配置过程，并通过具体的示例展示了如何使用 Python 进行建模与求解。具体来说，本章从 Gurobi 入门和进阶两个阶段展开：

入门阶段：首先介绍 Gurobi 在 Python 环境下的安装配置，然后详细讲解了 Gurobi 建模求解流程，包括创建模型对象、定义决策变量、设置目标函数、添加约束条件和优化求解等步骤，并通过数独问题实战案例加深理解。

进阶阶段：深入探讨了更复杂的变量类型和约束条件的定义方法。介绍了 Gurobi 提供的 Callbacks 回调功能，并通过对称旅行商问题实战案例展示了如何利用这些功能增强模型的灵活性和求解效率。

研学互通

科学研究与工程实践中，常常会遇到大量复杂的组合优化问题，其中多数属于计算复杂性理论中的 NP 难问题。这类问题在规模增大时，解空间呈指数级增长，导致求解难度急剧上升。直接调用 Gurobi、Cplex 等数学规划求解器对此类大规模数学规划模型进行精确求解时，需要耗费大量时间，甚至无法在可接受的时间范围内找到可行解，严重影响了实际应用的效率和可行性。

因此，如何在保证解质量的前提下提升求解效率，成为优化领域亟需解决的关键问题。为应对这一挑战，学术界提出了多种高效的大规模优化方法，包括 Benders 分解算法、列生成、拉格朗日松弛、分支定价、分支定价切割等精确算法，以及基于这些精确算法设计的启发式策略。如何设计高效的大规模优化算法也成为当前优化领域研究的核心问题之一。这些方法通过对问题结构的合理拆分、问题重构或智能搜索机制，在显著提高求解效率的同时，保持了解的可靠性与实用性，具有重要的理论价值和广泛的应用前景。

本章主要介绍了如何使用求解器直接求解数学规划模型。为进一步了解大规模精确算法以及数学规划求解器在其中如何调用，推荐读者阅读以下经典文献：

(1) Geoffrion, A. M. (1972). Generalized benders decomposition. Journal of optimization theory and applications, 10, 237-260.

该文章介绍了 Benders 分解法，强调其适用于具有复杂变量（当这些变量暂时固定时，会使剩余的优化问题变得相当易于处理）的数学规划问题。A .M. Geoffrion 提出了广义的 Benders 分解法，要求固定复杂变量后的数学规划问题

（也称子问题）满足对偶理论。该算法框架中，可直接调用求解器求解被分解的问题。

(2) Angulo, G., Ahmed, S., & Dey, S. S. (2016). Improving the integer L-shaped method. INFORMS Journal on Computing, 28(3), 483-499.

整数 L-shaped 方法基于分支定切框架，是求解两阶段随机整数规划的常用方法，要求第一阶段变量是 0-1 整数，第二阶段变量是混合整数。该文章首先详细介绍了整数 L-shaped 方法，然后，提出了整数 L-shaped 方法的两种改进策略。其中一种改进策略——交替切割策略已在大量的研究中证实其有效性。这篇文章可以帮助读者更好地理解分支定切框架中如何使用回调功能添加约束。

(3) Feillet, D. (2010). A tutorial on column generation and branch-and-price for vehicle routing problems. 4or, 8(4), 407-424.

分支定价法是分支定界和列生成方法的组合，在交通运输领域，尤其是车辆路径规划问题中被广泛应用，要求问题能够重建模为集覆盖（set covering）或集划分（set partition）模型。该文章首先概述了这些方法的基本原理和理论基础。此外，讨论了一些如何加强算法效率的问题，包括松弛问题的加强。该文章可以作为科研人员学习列生成和分支定价方法的入门教程。

(4) Costa, L., Contardo, C., & Desaulniers, G. (2019). Exact branch-price-and-cut algorithms for vehicle routing. Transportation Science, 53(4), 946-985.

分支定价切割算法综合了分支定界、列生成和割平面技术，是目前运筹优化领域最前沿的大规模精确算法之一。该文章回顾了自 1990 年以来，分支定价和分支定价切割算法在车辆路径规划领域的方法论和建模贡献，有助于科研人员进一步改进现有最先进算法。具体讨论内容包括分支定价切割算法中定价、割平面、分支、问题上下界等步骤的改进策略。

思行经世：自主创新破垄断，产研融合强根基

高端数学规划求解器曾如精密齿轮般嵌入现代工业与能源系统的核心，其长期被欧美巨头垄断，成为制约我国工业智能化进程与能源安全的“卡脖子”之痛。

它不仅影响局部技术发展，更折射出基础工具缺失对国家安全的深层威胁——犹如数字时代的“工业血液”受制于人，关键技术短板一旦遭遇封锁，整个产业链便有“停摆”之虞。

面对这一严峻挑战，国家“十四五”规划高瞻远瞩，以“创新驱动发展”为战略核心，着力构建“高校-科研机构-企业”三位一体的协同创新体系。这一体系不仅致力于知识融通，更意在打通“基础研究-技术攻关-产业落地”的全链条，将理论探索的星火点燃为产业升级的燎原烈焰。

在这一国家战略的精准指引下，杉数科技以“十年磨一剑”的坚韧，投身于混合整数规划算法与高性能计算架构的攻坚战场。混合整数规划，这一处理复杂决策优化的数学利器，曾如天堑横亘；而支撑其高效运行的高性能架构，更是要求超强的并行计算与资源调度能力。杉数科技团队深耕算法本质，突破理论瓶颈，创新架构设计，终于淬炼出我国首款完全自主知识产权的工业级优化求解器——COPT。其问世一举填补国内空白，更在权威的国际测评中跻身第一梯队，标志着我国优化技术实现了从“跟跑”到“并跑”的历史性跃升。COPT 的诞生，是产学研深度融合的成果，也是国家战略与市场需求的共振。

当前，我国自主研发的多款数学规划求解器已经在航空航天制造、双碳目标下的能源转型、乡村振兴、物流优化等 20 余个关键领域得到应用，成为高质量发展的“隐形引擎”。国产求解器的崛起为全球科技竞争中的中国方案写下有力注脚——在核心技术的无人区，中国智慧正开拓属于自己的坐标，驱动着高质量发展的巨轮破浪前行。

习题

习题 4.1 使用 Gurobi 求解下列数学模型，得出最优解。

$$
\begin{aligned}
&\max 10.8x_1 + 7.8x_2 + 3.2x_3 + 6.3x_4 + 5.4x_5 \\
&\text{s.t. } 1.1\mathrm{x}_1 + 1.2x_2 + 1.3x_3 + 1.4x_4 + 1.5x_5 \le 4.8 \\
&\quad 2.1\mathrm{x}_1 + 1.8x_2 + 1.4x_3 + 2.4x_4 + 2.1x_5 \le 6.1 \\
&\quad 1.8\mathrm{x}_1 + 0.6x_2 + 0.48x_3 + 1.2x_4 + 2.0x_5 \le 4.2 \\
&\quad 0.1\mathrm{x}_1 + 0.2x_2 + 1.7x_3 + 1.3x_4 + 1.5x_5 \le 4.4 \\
&\quad 1.5\mathrm{x}_1 + 0.8x_2 + 0.4x_3 + 1.3x_4 + 1.8x_5 \le 3.6
\end{aligned}
$$

习题 4.2 使用 Gurobi 求解下列数学模型，得出最优解。

$$\min \sum_{i=1}^{8}\sum_{j=1}^{12} c_{ij}x_{ij} + 5\sum_{i}^{8} y_i$$

$$\text{s.t.} \sum_{i}^{8} x_{i,j} = 1, \quad \forall j = 1, \dots, 12$$

$$\sum_{j}^{12} x_{i,j} \le 12y_i, \quad \forall i = 1, \dots, 8$$

$$x_{ij} \in \{0,1\}, y_i \in \{0,1\} \quad \forall i = 1, \dots, 8; j = 1, \dots, 12,$$

其中，目标函数中系数 c = [[3, 10, 2, 5, 2, 8, 8, 8, 7, 4, 2, 8],

[1, 7, 7, 10, 1, 8, 5, 4, 10, 2, 6, 1],

[1, 1, 9, 1, 7, 4, 7, 1, 9, 4, 8, 8],

[9, 4, 6, 4, 4, 8, 5, 1, 7, 9, 2, 3],

[5, 2, 6, 9, 7, 9, 4, 5, 5, 10, 8, 9],

[7, 10, 1, 8, 4, 7, 7, 3, 6, 9, 6, 2],

[8, 9, 2, 3, 9, 7, 6, 8, 1, 8, 1, 5],

[10, 10, 10, 7, 3, 3, 9, 4, 1, 4, 9, 9]]

习题 4.3 使用 Gurobi 求解习题 4.2 数学模型的线性松弛问题，得出最优解，并给出其对偶变量值（影子价格）。