# 第七讲　算法概述

* 算法思想
* 线搜索
* 信赖域
* 线搜索方向
* 算法收敛性
* 算法复杂度

**一、算法思想**

“算法”即演算法的大陆中文名称出自《[周髀算经](https://baike.baidu.com/item/%E5%91%A8%E9%AB%80%E7%AE%97%E7%BB%8F" \t "_blank)》；而英文名称Algorithm 来自于9世纪波斯数学家al-Khwarizmi，因为al-Khwarizmi在数学上提出了算法这个概念。“算法”原为"algorism"，意思是阿拉伯数字的运算法则，在18世纪演变为"algorithm"。[欧几里得算法](https://baike.baidu.com/item/%E6%AC%A7%E5%87%A0%E9%87%8C%E5%BE%97%E7%AE%97%E6%B3%95" \t "_blank)被人们认为是史上第一个算法。 第一次编写程序是Ada Byron于1842年为巴贝奇分析机编写求解[伯努利方程](https://baike.baidu.com/item/%E4%BC%AF%E5%8A%AA%E5%88%A9%E6%96%B9%E7%A8%8B" \t "_blank)的程序，因此Ada Byron被大多数人认为是世界上第一位[程序员](https://baike.baidu.com/item/%E7%A8%8B%E5%BA%8F%E5%91%98" \t "_blank)。因为查尔斯·巴贝奇(Charles Babbage)未能完成他的巴贝奇分析机，这个算法未能在巴贝奇分析机上执行。 因为"well-defined procedure"缺少数学上精确的定义，19世纪和20世纪早期的数学家、[逻辑](https://baike.baidu.com/item/%E9%80%BB%E8%BE%91" \t "_blank)学家在定义算法上出现了困难。20世纪的英国数学家[图灵](https://baike.baidu.com/item/%E5%9B%BE%E7%81%B5" \t "_blank)提出了著名的图灵论题，并提出一种假想的计算机的抽象模型，这个模型被称为[图灵机](https://baike.baidu.com/item/%E5%9B%BE%E7%81%B5%E6%9C%BA" \t "_blank)。图灵机的出现解决了算法定义的难题，图灵的思想对算法的发展起到了重要作用。

算法（Algorithm）是指解题方案的准确而完整的描述，是一系列解决问题的清晰[指令](https://baike.baidu.com/item/%E6%8C%87%E4%BB%A4/3225201" \t "_blank)，算法代表着用系统的方法描述解决问题的策略机制。也就是说，能够对一定规范的[输入](https://baike.baidu.com/item/%E8%BE%93%E5%85%A5/32696" \t "_blank)，在有限时间内获得所要求的输出。如果一个算法有缺陷，或不适合于某个问题，执行这个算法将不会解决这个问题。不同的算法可能用不同的时间、空间或效率来完成同样的任务。一个算法的优劣可以用[空间复杂度](https://baike.baidu.com/item/%E7%A9%BA%E9%97%B4%E5%A4%8D%E6%9D%82%E5%BA%A6/9664257" \t "_blank)与[时间复杂度](https://baike.baidu.com/item/%E6%97%B6%E9%97%B4%E5%A4%8D%E6%9D%82%E5%BA%A6/1894057" \t "_blank)来衡量。算法中的指令描述的是一个[计算](https://baike.baidu.com/item/%E8%AE%A1%E7%AE%97/81857" \t "_blank)，当其[运行](https://baike.baidu.com/item/%E8%BF%90%E8%A1%8C" \t "_blank)时能从一个初始状态和（可能为空的）初始输入开始，经过一系列**有限**而清晰定义的状态，最终产生**输出**并**停止**于一个终态。一个状态到另一个状态的转移不一定是确定的。[随机化算法](https://baike.baidu.com/item/%E9%9A%8F%E6%9C%BA%E5%8C%96%E7%AE%97%E6%B3%95/6233182" \t "_blank)在内的一些算法，包含了一些随机[输入](https://baike.baidu.com/item/%E8%BE%93%E5%85%A5/5481954" \t "_blank)。

形式化算法的概念部分源自尝试解决[希尔伯特](https://baike.baidu.com/item/%E5%B8%8C%E5%B0%94%E4%BC%AF%E7%89%B9/172452" \t "_blank)提出的[判定问题](https://baike.baidu.com/item/%E5%88%A4%E5%AE%9A%E9%97%AE%E9%A2%98" \t "_blank)，并在其后尝试定义有效计算性或者有效方法中成形。这些尝试包括[库尔特·哥德尔](https://baike.baidu.com/item/%E5%BA%93%E5%B0%94%E7%89%B9%C2%B7%E5%93%A5%E5%BE%B7%E5%B0%94/10781768" \t "_blank)、Jacques Herbrand和[斯蒂芬·科尔·克莱尼](https://baike.baidu.com/item/%E6%96%AF%E8%92%82%E8%8A%AC%C2%B7%E7%A7%91%E5%B0%94%C2%B7%E5%85%8B%E8%8E%B1%E5%B0%BC/2094842" \t "_blank)分别于1930年、1934年和1935年提出的[递归函数](https://baike.baidu.com/item/%E9%80%92%E5%BD%92%E5%87%BD%E6%95%B0/5634537" \t "_blank)，[阿隆佐·邱奇](https://baike.baidu.com/item/%E9%98%BF%E9%9A%86%E4%BD%90%C2%B7%E9%82%B1%E5%A5%87" \t "_blank)于1936年提出的[λ演算](https://baike.baidu.com/item/%CE%BB%E6%BC%94%E7%AE%97" \t "_blank)，1936年Emil Leon Post的Formulation 1和[艾伦·图灵](https://baike.baidu.com/item/%E8%89%BE%E4%BC%A6%C2%B7%E5%9B%BE%E7%81%B5" \t "_blank)1937年提出的[图灵机](https://baike.baidu.com/item/%E5%9B%BE%E7%81%B5%E6%9C%BA/2112989" \t "_blank)。即使在当前，依然常有直觉想法难以定义为形式化算法的情况。本章中，我们主要学习优化算法。

在过去的四十年中，已经看到了一系列强大算法的开发，这些算法用于平滑函数的无约束优化。我们现在对其主要属性进行更详细的描述。所有用于无约束最小化的算法都要求用户提供起始点，我们通常用表示。具有应用和数据集知识的用户可以选择作为合理的解决方案估计。否则，必须通过系统方法或以某种任意方式，或者算法选择起始点。从算法第一步开始，生成一系列迭代点{}，达到一定的精度时终止。在决定如何从一个迭代移动到下一个迭代时，算法使用关于函数在 的信息，并且还可能使用来自前迭代点, , . . . , 的信息。他们使用此信息来查找具有比更低的函数值，来产生新迭代点。

注：存在非单调算法，不会在每一步都坚持减少，但即使这些算法也要求在经过一些规定的次迭代后减小，即。从当前点移动到新迭代点有两种基本方法。本书中描述的大多数算法都遵循这些方法之一。

**二、线搜索和信赖域方法**  
1.线搜索

在线搜索策略中，算法选择方向并从当前迭代沿该方向搜索具有较低函数值的新迭代。沿着移动的距离可以通过近似求解下面的一维最小化问题来找到步长α：

(2.10)

通过精确地求解（2.10），我们可以从方向中获得最大的好处，但是精确的最小化可能是昂贵的并且通常是不必要的。相反，线搜索算法生成有限数量的试验步长，直到找到松散近似最小值（2.10）的步长。在新的点处，计算新的搜索方向和步长，并重复该过程。

线搜索包括精确线搜索和非精确线搜索。其中精确线搜索包括：试探法（黄金分割法、Fibonacci法、二分法）和函数逼近法（Newton法、割线法、抛物线法、三次插值法）；非精确线搜索包括：Armijo步长规则、Goldstein步长规则、Wolfe步长规则。

2、信赖域

在信赖域的策略中，关于收集的信息用于构造模型函数，其在当前点附近的行为类似于实际目标函数的行为。因为当x远离时，模型可能不是的良好近似，所以我们将搜索的最小化限制在周围的某个区域。换句话说，我们通过近似解决以下子问题找到候选步骤：

, 其中 落在信赖域中. (2.11)

我们现在预览两个主要问题：在线搜索方法中选择搜索方向，以及在信赖域方法中选择Hessian 矩阵。正如我们所观察到的，这些问题密切相关。

**三、线搜索方向**

1、最速下降方向  
最速下降方向−∇是线搜索方向最明显的选择。它很直观; 在我们可以从移动的所有方向中，最速下降方向是函数值下降最快的方向。最速下降法是一种线搜索方法，它在每一步都沿着 =−∇移动。它可以通过多种方式选择步长。 最速下降方向的一个优点是它需要计算梯度∇而不是二阶导数。但是，对于棘手的问题，它可能会非常缓慢。

线搜索方法可以使用除最速下降方向之外的搜索方向。 通常，只要步长足够小，任何下降方向保证与−∇方向之间的角度的严格小于π/ 2，这种条件下会保证函数值的减小。

2、牛顿方向

另一个重要的搜索方向是牛顿方向。该方向源自对的二阶泰勒级数近似。 当真实函数与其二次模型之间的差异不太大时，牛顿方向是可靠的。与最速下降方向不同， 牛顿方法的大多数线搜索实现在可能的情况下使用单位步长α= 1，并且当函数值减少的情况没有达到满意的程度时下，我们去调整α。当不是正定时，牛顿方向可能无法定义。 即使被定义，它也可能不满足下降，在这种情况下它不适合作为搜索方向。 在这些情况下，线搜索方法修改的定义以使其满足下降条件，同时保留 中包含的二阶信息。使用牛顿方向的方法具有快速的局部收敛速度，通常是二次的。在达到解的邻域之后，通常仅在几次迭代中发生高精度的收敛。牛顿方向的主要缺点是需要Hessian矩阵。这种二阶导数矩阵的显式计算有时可能是一个麻烦，容易出错且昂贵的过程。所描述的有限差分和自动微分技术可用于避免手动计算二阶导数的需要。

拟牛顿搜索方向提供了牛顿方法的有吸引力的替代方案，因为它们不需要计算Hessian阵并且仍然获得超线性收敛速率。他们使用近似代替真正的Hessian矩阵 ，在每一步之后更新，以考虑在该步骤中获得的额外知识。利用了梯度的变化提供关于沿搜索方向的的二阶导数的信息的事实。

(2.16)

我们选择新的Hessian矩阵的近似，使其符合真正的Hessian阵的性质（2.16），也就是说，我们要求它满足以下条件，称为割线方程：

(2.17)

这里 。

通常，我们对施加附加条件，例如对称性，以及要求连续近似和之间的差具有低秩性。  
用于更新的两个最流行的公式是SR1公式和BFGS公式。

（1）Symmetric-rank-one（SR1）公式，定义为

（2）BFGS公式，以其发明者Broyden，Fletcher，Goldfarb和Shanno命名，定义为

(2.19)

注意，矩阵和之间的差异在（2.18）的情况下是秩-1矩阵，在（2.19）的情况下是秩-2矩阵。 两个更新都满足割线方程并且都保持对称性。 可以证明，只要初始为正定且，BFGS更新（2.19）就会产生正定的近似矩阵。通过使用代替公式（2.15）中的精确Hessian来获得拟牛顿搜索方向。拟牛顿方法的实现避免了在每次迭代时分解来更新的逆而不是。实际上，（2.18）和（2.19）的等价公式适用于逆近似。

, (2.21)

然后可以使用公式来执行的计算。  
3、共轭梯度方向

最后一类搜索方向是由共轭梯度方法生成的。它们的形式为

，

其中是一个标量，确保和是共轭的，这是二次函数最小化的一个重要概念。共轭梯度法最初设计用于求解线性方程组Ax = b，其中系数矩阵A是对称的和正定的。解决这个线性系统的问题等同于求解最小化凸二次函数的问题，因此将这些算法的扩展研究到更一般类型的无约束最小化问题是很自然的。通常，非线性共轭梯度方向比最速下降方向更有效，并且几乎同样易于计算。这些方法没有达到牛顿或拟牛顿方法的快速收敛速度，但它们具有不需要存储矩阵的优点。

到目前为止讨论的所有搜索方向都可以直接在线搜索框架中使用。它们随之产生了最速下降法、牛顿法、拟牛顿法和共轭梯度线搜索法。

注：如果我们在（2.12）中设置= 0并使用欧几里德范数定义信赖域，则信赖域子问题（2.11）变得容易解决。这只是一个最速下降步骤，其中步长由信赖域半径决定; 在这种情况下，信赖域和线搜索方法基本相同。

通过选择近似为精确的Hessian来获得更有趣的信赖域算法。信赖域Newton方法在实践中证明是非常有效的。如果（2.12）的二次模型函数中的矩阵是通过拟牛顿近似来定义的，我们获得了一个信赖域拟牛顿法。

4、尺度化  
算法的性能可能主要取决于如何制定问题。问题表达中的一个重要问题是尺度化。 在无约束优化中，如果在某个固定的方向上，我们改变时，函数值的变化与在另一个方向上相比，具有较大的方差，则称该问题性质较差。对尺度化不敏感的算法是优选的，因为它们可以以更稳健的方式处理较难的问题。在设计完整算法时，我们尝试将比例不变性纳入算法的所有方面，包括线搜索或信赖域方法和收敛测试。一般而言，保持线搜索算法的尺度不变性比信赖域算法更容易。

**四、算法收敛性**

1.全局收敛、局部收敛

设为解集合，为算法映射。给定一个集合,若对于任意的初始点,算法A所产生的序列{}中任一收敛序列的极限都属于则称算法映射A在Y上收敛。

若集合Y是任意选取的（该集合不必限定在解集合的很小邻域内），则相应的收敛性称为全局收敛性（global convergence）.

若集合Y只能取接近的点集，则相应的收敛性称为局部收敛性（local convergence）.

2、收敛准则

（1） 或者 ，

（2） 或者，

（3）(适用于无约束最优化)。

3、收敛速率

设序列{}收敛于，定义满足

（）

的非负数p的上确界为序列{}的收敛级。收敛级p越大，序列收敛得越快；当收敛级p相同时，收敛比β越小，序列收敛得越快。

（1）若序列的收敛级为p,则称序列是p级收敛的。

（2）若p=1，且，则称序列是以收敛比的线性收敛的。

（3）若p>1,或者p=1，且，则称序列是超线性收敛的。

4、算法的二次终止性

若某个算法对任意的正定二次函数，从任意的初始点出发，都能经有限步迭代达到其极小点，则称该算法具有二次终止性。用二次终止性作为判断算法优劣的原因：

(1)正定二次函数具有某些较好的性质，因此一个好的算法应能够在有限步内达到其极小点；

(2)对于一般的目标函数，若在其极小点处Hessian矩阵正定，由Taylor展开式可以猜想，对正定二次函数好的算法，对于一般目标函数也应具有较好的性质。

**五、算法复杂性**

在计算机科学中，同一问题可用不同算法解决，而一个算法的质量优劣将影响到算法乃至程序的效率。[算法分析](https://baike.baidu.com/item/%E7%AE%97%E6%B3%95%E5%88%86%E6%9E%90" \t "_blank)的目的在于选择合适算法和改进算法。算法复杂性描述算法的存储要求和运行时间要求，分为算法的空间复杂性和算法的时间复杂性。利用算法需要的初等运算次数表示算法的时间复杂性。算法的时间复杂度是指执行算法所需要的计算工作量。一般来说，计算机算法是问题规模 的函数。算法的空间复杂度是指算法需要消耗的内存空间。其计算和表示方法与时间[复杂度](https://baike.baidu.com/item/%E5%A4%8D%E6%9D%82%E5%BA%A6" \t "_blank)类似，一般都用复杂度的渐近性来表示。同时间复杂度相比，空间复杂度的分析要简单得多。

1、时间复杂性

求解实例的算法的基本计算总次数是实例输入长度的一个函数，该函数被另一个函数控制，即存在一个函数和一个常数，使得



成立。

根据时间复杂性分为多项式时间算法与指数时间算法。

若给定该问题的一个实例，存在多项式函数，使得成立，则称该算法对实例是多项式时间算法。当为指数函数时，称相应的算法为指数时间算法。

2、多项式时间算法的优点

（1）随着问题输入规模的增加，算法的计算量（即算法复杂性）呈多项式增长.

（2）一个多项式时间算法利用另一个多项式时间算法作为其“子程序”，构造一个新的复合型算法，则新算法仍是多项式时间算法。

3、单纯形算法的复杂性



上例用单纯形算法需要次迭代。

**总结**：从下节开始，我们逐渐深入学习使用导数的最优化方法，包括一阶梯度算法和二阶牛顿算法，以及目前大规模计算的有效方法。我们主要介绍迭代算法，针对不同的模型，给出相应常用算法的基本框架。然后分析算法的收敛性结果和相应的优缺点，最后给出具体算法实例。

**六、习题**

参考文献：

1. 《Convex Optimization Algorithm》，Dimitri P. Bertsekas 2015.
2. 《Convex Optimization Theory》Dimitri P. Bertsekas， 2009
3. 《Numerical Optimization》Jorge Nocedal, Stephen Wright， 1999(2006)
4. 《Convex Optimization》 Stephen Boyd, Lieven Vandenberghe， 2004
5. 《最优化理论与算法》陈宝林，1989(2005)编著