最优化算法

一般来说，最优化算法研究可以分为：构造最优化模型、确定最优化问题的类型和设计算法、实现算法或调用优化算法软件包进行求解.

- 最优化模型的构造和实际问题紧密相关，在问题(1) 中，目标函数f 和约束函数c_i 都是由模型来确定的.
- 在确定模型之后，我们需要对模型对应的优化问题进行分类，分类的必要性是因为不存在对于所有优化问题的一个统一的算法. 因此我们需要针对具体优化问题所属的类别，来设计或者调用相应的算法求解器.
- 最后就是模型的求解过程. 同一类优化问题往往存在着不同的求解算法. 对于具体的优化问题，需要充分利用问题的结构，并根据问题的需求（求解精度和速度等）来设计相应的算法. 另外，根据算法得到的结果，可以来判别模型构造是否合理或者进一步地改进模型.

优化算法设计

在设计优化算法时，有一些基本的准则或技巧．对于复杂的优化问题，基本的想法是将其转化为一系列简单的优化问题（其最优解容易计算或者有显式表达式）来逐步求解．常用的技巧有：

- **泰勒（Taylor）展开**对于一个非线性的目标或者约束函数，通过其泰勒展开用简单的线性函数或者二次函数来逼近，从而得到一个简化的问题．因为该简化问题只在小邻域内逼近原始问题，所以需要根据迭代点的更新来重新构造相应的简化问题．
- **对偶**每个优化问题都有对应的对偶问题．特别是凸的情形，当原始问题比较难解的时候，其对偶问题可能很容易求解．通过求解对偶问题或者同时求解原始问题和对偶问题，可以简化原始问题的求解，从而设计更有效的算法．

- **拆分**对于一个复杂的优化问题，可以将变量进行拆分，比如$\min_x h(x)+r(x)$，可以拆分成

$$\min_{x,y} \quad h(x)+r(y), \qquad \text{s.t.} \qquad x=y.$$

  通过引入更多的变量，则可以得到每个变量的简单问题（较易求解或者解有显式表达式），从而通过交替求解等方式来得到原问题的解.

- **块坐标下降**对于一个n维空间（n 很大）的优化问题，可以通过逐步求解分量的方式将其转化为多个低维空间中的优化问题. 比如，对于$n=100$，可以先固定第2—100 个分量，来求解x_1；接着固定下标为$1,3$—100 的分量来求解x_2；依次类推.

凸和非凸优化问题

- 凸优化问题是指最小化问题(1) 中的目标函数和可行域分别是凸函数和凸集. 如果其中有任何一个不是凸的，则相应的最小化问题是非凸优化问题. 因为凸优化问题的任何局部最优解都是全局最优解，其相应的算法设计以及理论分析相对非凸优化问题简单很多.
- 若问题(1) 中的min 改为max，且目标函数和可行域分别为凹函数和凸集，我们也称这样的问题为凸优化问题. 这是因为对凹函数求极大等价于对其相反数（凸函数）求极小.
- 在实际问题的建模中，经常更倾向于得到一个凸优化模型. 除此之外,判断一个问题是否是凸问题也很重要. 给定一个非凸优化问题，一种方法是将其转化为一系列凸优化子问题来求解. 此时需要清楚原非凸问题中的哪个或哪些函数导致了非凸性，之后考虑的是如何用凸优化模型来逼近原问题.

全局和局部最优解

在求解最优化问题之前，先介绍最小化问题(1) 的最优解的定义.

定义 (最优解)

对于可行点$\bar{x}$ (即$\bar{x} \in \mathcal{X}$)，定义如下概念：

(1) 如果$f(\bar{x}) \leq f(x), \forall x \in \mathcal{X}$，那么称$\bar{x}$ 为问题(1) 的**全局极小解（点）**，有时也称为（全局）最优解或最小值点；

(2) 如果存在$\bar{x}$ 的一个ε 邻域$N_\varepsilon(\bar{x})$ 使得$f(\bar{x}) \leq f(x), \forall x \in N_\varepsilon(\bar{x}) \cap \mathcal{X}$, 那么称$\bar{x}$ 为问题(1) 的**局部极小解（点）**，有时也称为局部最优解；

(3) 进一步地，如果有$f(\bar{x}) < f(x), \forall x \in N_\varepsilon(\bar{x}) \cap \mathcal{X}$, 且$x \neq \bar{x}$ 成立，则称$\bar{x}$ 为问题(1) 的**严格局部极小解（点）**.

如果一个点是局部极小解，但不是严格局部极小解，则称为**非严格局部极小解**. 在图5 中，我们以一个简单的函数为例，指出了其全局与局部极小解.

全局和局部最优解

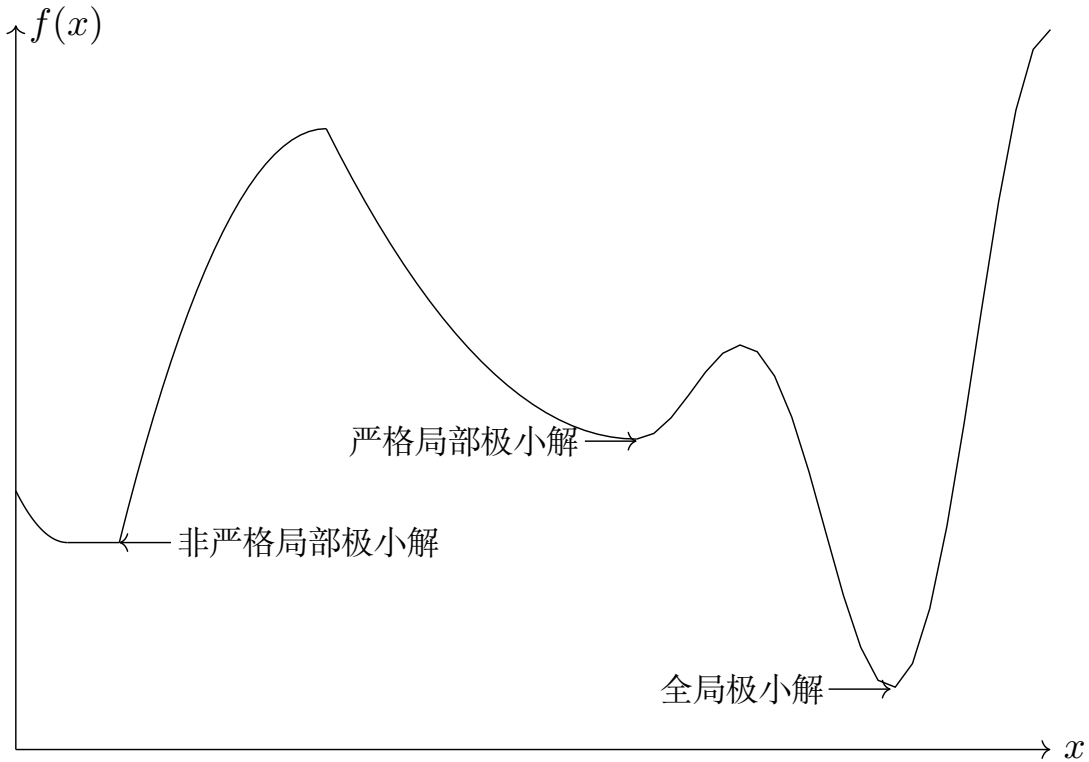


Figure: 函数的全局极小、严格局部极小和非严格局部极小解

在问题(1) 的求解中，我们想要得到的是其全局最优解，但是由于实际问题的复杂性，往往只能够得到其局部最优解.

优化算法收敛性

由于实际问题往往是没有办法显式求解的，因此常采用迭代算法.

- 迭代算法的基本思想是：从一个初始点x^0出发，按照某种给定的规则进行迭代，得到一个序列$\{x^k\}$. 如果迭代在有限步内终止，那么最后一个点就是优化问题的解. 如果迭代点列是无穷集合，那么希望该序列的极限点（或者聚点）则为优化问题的解.
- 在算法设计中，还需考虑算法产生的点列是否收敛到优化问题的解. 给定初始点x^0，记算法迭代产生的点列为$\{x^k\}$. 如果$\{x^k\}$在某种范数$\|\cdot\|$的意义下满足$\lim_{k\to\infty}\|x^k - x^*\| = 0$,且收敛的点$x^*$为一个局部（全局）极小解，那么我们称该点列收敛到局部（全局）极小解，相应的算法称为是**依点列收敛到局部（全局）极小解**的.

优化算法收敛性

- 进一步地，如果从任意初始点x^0出发，算法都是依点列收敛到局部（全局）极小解的，我们称该算法是**全局依点列收敛到局部（全局）极小解**的．记对应的函数值序列$\{f(x^k)\}$，可以定义算法的**（全局）依函数值收敛到局部（全局）极小值**的概念．
- 对于凸优化问题，因为其任何局部最优解都为全局最优解，算法的收敛性都是相对于其全局极小而言的．
- 除了点列和函数值的收敛外，实际中常用的还有每个迭代点的最优性条件（如无约束优化问题中的梯度范数，约束优化问题中的最优性条件违反度等等）的收敛．
- 对于带约束的情形，给定初始点x^0，算法产生的点列$\{x^k\}$不一定是可行的（即$x^k \in \mathcal{X}$未必对任意k成立）．考虑到约束违反的情形，我们需要保证$\{x^k\}$在收敛到x^*的时候，其违反度是可接受的．除此要求之外，算法的收敛性的定义和无约束情形相同．

算法的渐进收敛速度

设$\{x^k\}$为算法产生的迭代点列且收敛于x^*

- 算法（点列）**Q-线性收敛**：对充分大的k有

$$\frac{\|x^{k+1}-x^*\|}{\|x^k-x^*\|}\leq a,\quad a\in(0,1)$$

- 算法（点列）**Q-超线性收敛**：对充分大的k有

$$\lim_{k\to\infty}\frac{\|x^{k+1}-x^*\|}{\|x^k-x^*\|}=0,$$

- 算法（点列）**Q-次线性收敛**：对充分大的k有

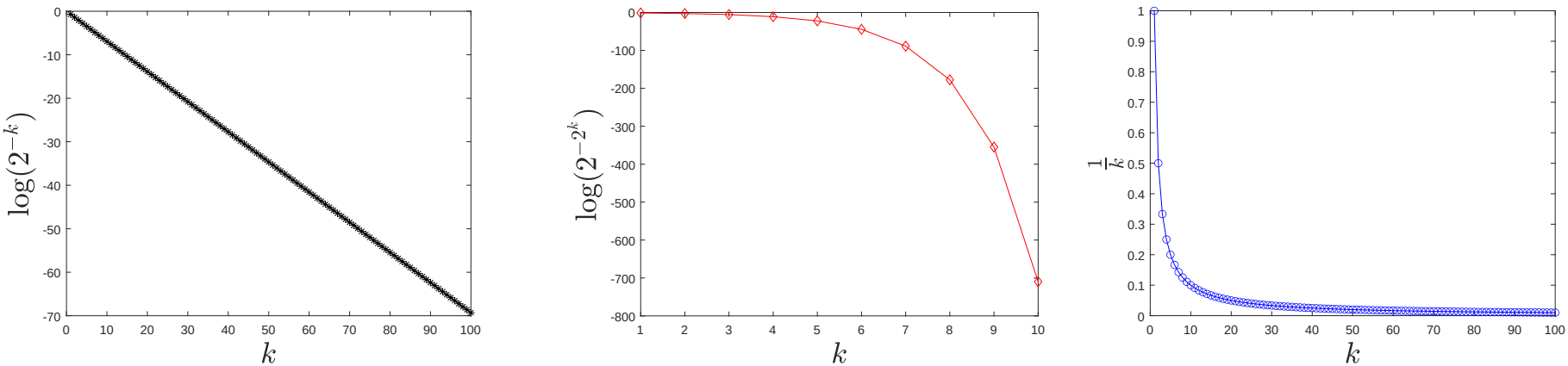
$$\lim_{k\to\infty}\frac{\|x^{k+1}-x^*\|}{\|x^k-x^*\|}=1,$$

- 算法（点列）**Q-二次收敛**：对充分大的k有

$$\frac{\|x^{k+1}-x^*\|}{\|x^k-x^*\|^2}\leq a,\quad a>0,$$

算法的渐进收敛速度

我们举例来更直观地展示不同的Q-收敛速度，参见下图（图中对所考虑的点列作了适当的变换）．点列$\{2^{-k}\}$是Q-线性收敛的，点列$\{2^{-2^k}\}$是Q-二次收敛的（也是Q-超线性收敛的），点列$\{\frac{1}{k}\}$是Q-次线性收敛的．一般来说，具有Q-超线性收敛速度和Q-二次收敛速度的算法是收敛较快的



(a) Q-线性收敛　(b) Q-二次收敛　(c) Q-次线性收敛

Figure: 不同Q-收敛速度比较

算法的渐进收敛速度

- 算法（点列）**R-线性收敛**: 设$\{x^k\}$为算法产生的迭代点且收敛于x^*，若存在Q-线性收敛于0 的非负序列t_k 并且

$$\|x^k - x^*\| \leq t_k$$

对任意的k成立．类似地，可定义**R-超线性收敛**和**R-二次收敛**等收敛速度．从R-收敛速度的定义可以看出序列$\{\|x^k - x^*\|\}$ 被另一趋于0 的序列$\{t_k\}$控制．当知道t_k的形式时，我们也称算法（点列）的收敛速度为$\mathcal{O}(t_k)$.

- **算法复杂度**．设x^*为全局极小点，某一算法产生的迭代序列$\{x^k\}$满足

$$f(x^k) - f(x^*) \leq \frac{c}{\sqrt{k}}, \quad \forall\, k > 0,$$

其中$c > 0$为常数。如果需要计算算法满足精度$f(x^k) - f(x^*) \leq \varepsilon$所需的迭代次数，只需令$\frac{c}{\sqrt{k}} \leq \varepsilon$ 则得到$k \geq \frac{c^2}{\varepsilon^2}$，因此该优化算法对应的（迭代次数）复杂度为$N(\varepsilon) = \mathcal{O}\left(\frac{1}{\varepsilon^2}\right)$.

优化算法的收敛准则

为了使算法能在有限步内终止，一般会通过一些收敛准则来保证迭代停在问题的一定精度逼近解上.

- 对于无约束优化问题，常用的收敛准则有

$$\frac{f(x^k)-f^*}{\max\{|f^*|,1\}} \le \varepsilon_1, \quad \|\nabla f(x^k)\| \le \varepsilon_2, \tag{11}$$

其中$\varepsilon_1, \varepsilon_2$为给定的很小的正数，$\|\cdot\|$表示某种范数（这里可以简单理解为$\ell_2$范数：$\|x\|_2 = \left(\sum_{i=1}^n x_i^2\right)^{1/2}$，$f^*$为函数$f$的最小值以及$\nabla f(x^k)$表示函数$f$在点$x$处的梯度.

- 对于约束优化问题，还需要考虑约束违反度. 也即要求最后得到的点满足

$$c_i(x^k) \le \varepsilon_3, \ i = 1, 2, \cdots, m,$$
$$|c_i(x^k)| \le \varepsilon_4, \ i = m+1, m+2, \cdots, m+l,$$

其中$\varepsilon_3, \varepsilon_4$为很小的正数，用来刻画$x^k$的可行性.

优化算法的收敛准则

- 除了约束违反度之外，也要考虑x^k与最优解之间的距离，如(11)式中给出的函数值与最优值的相对误差．由于一般情况下事先并不知道最优解，在最优解唯一的情形下一般使用某种基准算法来得到x^*的一个估计，之后计算其与x^k的距离以评价算法的性能．
- 因为约束的存在，不能简单地用目标函数的梯度来判断最优性，实际中采用的判别准则是点的最优性条件的违反度．
- 对于具体的算法，根据其设计的出发点，不一定能得到一个高精度的逼近解．为了避免无用的计算开销，需要一些停机准则来及时停止算法的进行．常用的停机准则有

$$\frac{\|x^{k+1}-x^k\|}{\max\{\|x^k\|,1\}}\le\varepsilon_5,\quad \frac{|f(x^{k+1})-f(x^k)|}{\max\{|f(x^k)|,1\}}\le\varepsilon_6,$$

其分别表示相邻迭代点和其对应目标函数值的相对误差很小．

- 在算法设计中，这两个条件往往只能反映迭代点列接近收敛，但不能代表收敛到优化问题的最优解．