优化理论简介 (上)

侠之大者

2016 年 4 月 7 日

在科学研究和工程实践中，很多棘手的问题都归结为一个优化问题。如何去求解一个优化问题，是每一个科研人员必备的技能。

然而，优化理论的著作浩如烟海，汗牛充栋，其理论也博大精深，深奥难懂，其方法更是层出不穷，令人眼花缭乱。一个优秀的科研人员用一辈子去研究优化理论，也仅能领悟其中之一二。

那么对于我们这些普通人，在遇到优化问题时，如何以最小的成本，花最少的时间找到合适的方法去求解呢?

1 浅薄的看法

对于优化理论和方法，虽然我还没有入门，但我还是想和大家分享一下我的浅见。

第一，简单了解，用时再学。对于优化理论，了解基本思想和框架，在遇到相应问题时，有针对性地去学习。因为大家几乎没有时间和精力去系统地学习每一种优化方法，比如经典的优化理论教材 «Numerical Optimization» 和 «Convex Optimization»，都差不多一千页左右，像一块砖头一样厚。而且这些优化方法还只是传统的算法，新的优化算法还在不断地以论文的形式被发表出来。对于一些新的优化问题，像稀疏表达，优化 L_0 范数，低秩约束，优化矩阵的核范数等等，这些问题的求解算法现在都还没有被整理进入教材，大都零星地出现在最近几年的论文中。

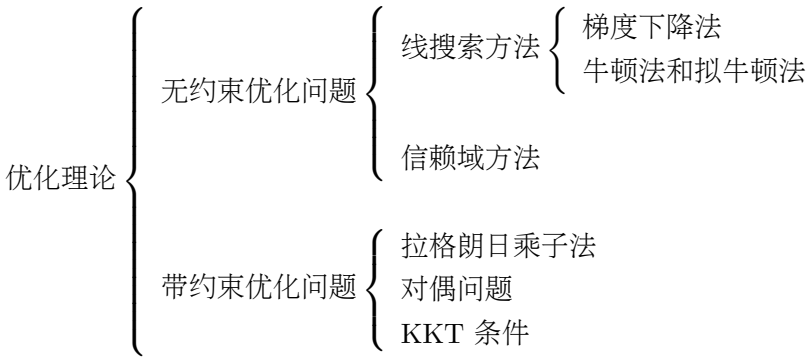
第二，没有最好，够用就行。这里有两层意思。其一，在工程实践中，我们一般只要求得的解满足我们的要求即可，不一定非得最优解。毕竟，得到的解越精确，所付出的代价就越大。其二，求解优化问题的方法不一定要高大上，要非常严格的收敛性证明，能给出结果，达到目的就行。这里有一

个有趣的例子。在 2010 年 TIP 上，有一篇关于图像 FCM 分割的论文，作者在优化模型中增加自己定义的先验约束项，但是给出的求解算法是直觉的、错误的。正确的解法发表在 2013 年的 TIP 上，论文名为 «Comments on“A Robust Fuzzy Local Information C-Means Clustering Algorithm”»。哈哈，不可思议的是，错误的解法得到的分割结果非常不错。

第三，先正确，再优化，不要过早优化。这一点其实非常重要。我们在研究一个问题时，往往会提出一个优化模型。首先，我们应该尽可能使用最简单的方法去求解这个优化模型，看看得到的结果是不是我们想要的。如果得到的结果不好，很可能是模型定义的问题。这时，我们应该去改进优化模型。但是，如果我们一开始就用非常复杂的算法去求解优化模型，即使结果不好，我们也不能确定是优化模型的问题，还是求解算法的问题。相对于简单的算法，高级算法能够减少迭代次数，快速求得最优解，但是也很容易被错误使用。如果定义了一个模型，使用了最简单的优化方法，发现效果不错，至少说明我们定义的模型没有问题。这时，如果还需要提高速度，可以有针对性地尝试比较高级的算法。这样比较胸有成竹，高级算法玩不转，我们还可以全身而退嘛。

2 简明的概述

在这里，我们给大家介绍优化理论中最基础的概念和方法。下面这张图可以说是优化理论中最基本的内容，再也没办法简化了。



首先，优化问题一般分为两类，无约束和带约束优化问题。其中无约束优化问题是基础，带约束优化问题可以先转换成无约束优化问题再去求解。

无约束优化问题的一般形式如下：

$$min \quad f(x)$$

其中 $x \in R^n$，是大小为 $n \times 1$ 的列向量，$f(x)$ 是目标函数，从 R^n 映射到 R。

带约束优化问题的一般形式如下：

$$\begin{aligned} & min \quad f(x) \\ s.t. \quad & h_i(x) = 0, \qquad i = 1, ..., m \\ & g_j(x) \le 0, \qquad j = 1, ..., n \end{aligned}$$

其中 $h_i(x)$ 为等式约束，$g_j(x)$ 为不等式约束。

这一次，我们主要介绍无约束优化问题的求解方法，关于带约束优化问题的讨论，留到下一次再说。

在下面的讨论中，我们假设 $f(x)$ 可导，并且是凸函数，这样可以保证局部最优解一定是全局最优解。

另外，这里对于数学概念和定理，我们只作简单的介绍，具体的内容和推导，大家可以参考有关的教材。

3 线搜索方法

线搜索方法是我们在解优化问题时最常用的方法，它的思想非常简单，归结成一句话，就是“先方向，后步长”。

对于当前点 x_k，我们首先确定目标函数 $f(x)$ 下降的方向 d_k，再估计下降的步长 α_k，这样便可以到达下一个迭代点 x_{k+1}，即

$$x_{k+1} = x_k + \alpha_k d_k$$

这样依次循环迭代，逐渐到达目标函数的最低点。

如何确定每一次迭代时的方向与步长，是线搜索方法的关键。

3.1 梯度与下降方向

我们知道梯度方向是函数值在当前点处上升最快的方向，它的数学定义为:

$$f(x): R^n \mapsto R \qquad \nabla f(x): R^n \mapsto R^n$$

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \qquad \nabla f(x) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix}$$

这里 x_i 指的是自变量 x 的第 i 个分量的值，梯度 $\nabla f(x)$ 是函数 $f(x)$ 对每一个分量的偏导数形成的列向量。

那么下降方向该如何选择呢?

理论上来说，在当前点 x_k 处，d_k 为下降方向的充要条件为：

$$\nabla f(x_k)^T d_k < 0$$

即只要我们所选的下降方向 d_k 与当前的梯度方向 $\nabla f(x_k)$ 的夹角超过 90 度即可，如下图 (1) 所示的 3 个方向都满足下降条件。

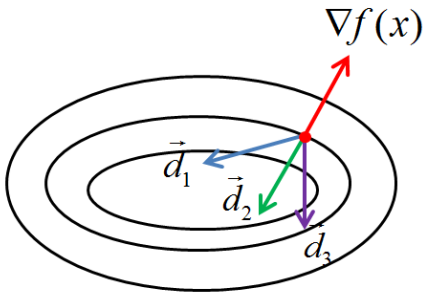


图 1: 梯度与下降方向

这里为什么强调当前点，因为梯度稍微一移动，便会发生变化。我们确定下降方向，是根据当前迭代点的梯度方向 $\nabla f(x_k)$ 来决定的，而不是全局考虑的。

另外，每一种线搜索方法都有自己的规则去确定下降方向。

3.2 估计步长

当我们确定好下降方向后，如何决定在这个方向上走多远?

直觉的思维是走到这个方向上的最低点最好，即

$$\alpha_k = argmin \quad f(x_k + \alpha d_k)$$

这是精确估计步长，其实没有必要。因为当前的下降方向只是局部选择的方向，如果在这个方向上一条道走到黑，不仅会花费太多时间，而且可能欲速而不达。

实践中常常用到的是下面的两种非精确估计步长的方法。它们的思想是只要在下降方向上，目标函数值有一定的下降即可，不必劳师动众地去精确估计步长。

3.2.1 Armijo 条件

在确定下降方向 d_k 后，只要我们选择的步长 α 满足 Armijo 条件即可，不必恰好到达最低点。Armijo 条件定义为：

$$f(x_k + \alpha d_k) \leq f(x_k) + c\alpha \nabla f(x_k)^T d_k$$

其中 $c \in (0, 1)$，一般可取 $c = 0.1$。

Armijo 步长搜索算法如下：

Algorithm 1 Armijo Search

Input: x_k, d_k, MAX_ITER_NUM

initialize $\alpha = 1.0, \quad c = 0.1$
for $(i = 0; i < MAX\_ITER\_NUM; i = i + 1)$ **do**
  if $f(x_k + \alpha d_k) \leq f(x_k) + c\alpha \nabla f(x_k)^T d_k$ **then**
    break;
  else
    $\alpha = \alpha/2$;
  end if
end for
return α;

3.2.2 Wolfe 条件

Wolfe 条件是在 Armijo 条件的基础上，又增加了一条关于下一个迭代点 x_{k+1} 的梯度约束：

$$f(x_k + \alpha d_k) \leq f(x_k) + c_1 \alpha^T d_k$$

$$\nabla f(x_k + \alpha d_k)^T d_k \geq c_2 \nabla f(x_k)^T d_k$$

其中 $0 < c_1 < c_2 < 1$，一般取 $c_1 = 0.1, c_2 = 0.9$。

为什么要增加这样一个条件?

因为在一个非常著名的拟牛顿 BFGS 方法中，为了保证更新矩阵的正定性，需要位移差 $s_k = x_{k+1} - x_k$ 和梯度差 $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$ 满足条件

$$y_k^T s_k > 0$$

我们将 Wolfe 条件的第二个约束变换一下，有

$$(\nabla f(x_k + \alpha d_k) - \nabla f(x_k))^T * (\alpha d_k) \geq \alpha(c_2 - 1)\nabla f(x_k)^T d_k > 0$$

即我们使用 Wolfe 条件搜索步长满足 BFGS 方法的先决条件。

Wolfe 步长搜索算法如下：

Algorithm 2 Wolfe Search

Input: x_k, d_k, MAX_ITER_NUM

initialize $\alpha = 1.0, c_1 = 0.1, c_2 = 0.9, left = 0, right = 1.0$
for $(i = 0; i < MAX\_ITER\_NUM; i = i + 1)$ **do**
  if $f(x_k + \alpha d_k) > f(x_k) + c_1 \alpha \nabla f(x_k)^T d_k$ **then**
    $right = \alpha$;
    $\alpha = (left + right)/2$;
  else if $\nabla f(x_k + \alpha d_k)^T d_k < c_2 \nabla f(x_k)^T d_k$ **then**
    $left = \alpha$;
    $\alpha = (left + right)/2$;
  else
    return α;
  end if
end for

3.3 基本框架

这里，常见的线搜索方法的基本框架如下：

Algorithm 3 Linear Search

1: initialize $x_0, k = 0$
2: **while** 不满足迭代停止条件 **do**
3: 确定当前点 x_k 处的下降方向d_k;
4: 利用 Armijo 或 Wolfe 条件估计步长α_k;
5: 更新当前迭代点$x_{k+1} = x_k + \alpha_k d_k$;
6: $k = k + 1$;
7: **end while**
8: return x_k;

常用的迭代停止条件有：
1. 当前迭代点 x_k 的梯度小于阈值 $||\nabla f(x_k)|| < \epsilon$
2. 位移的绝对误差小于阈值 $||x_{k+1} - x_k|| < \epsilon$
3. 目标函数的绝对误差小于阈值 $|f(x_{k+1}) - f(x_k)| < \epsilon$

3.4 梯度下降法

梯度下降法是大家最熟悉的优化方法。它使用负梯度方向作为下降方向，即

$$d_k = -\nabla f(x_k)$$

而对于步长的估计，我们可以采用前面介绍的 Armijo 或 Wolfe 条件来搜索。

大家最常见到的梯度下降法的公式是：

$$x_{k+1} = x_k - \alpha \nabla f(x_k)$$

其中 α 是一个固定值，一般可取 $\alpha = 0.5$ 等。这个公式中每一次选取的步长是固定值吗?

如果每次走固定的长度，显然不会收敛！

这里我们一直模糊了步长的概念。对于上式，可以换一种写法：

$$x_{k+1} = x_k + \alpha ||\nabla f(x_k)|| \times \frac{-\nabla f(x_k)}{||\nabla f(x_k)||}$$

可以看出，每次沿下降方向的行走距离为 $\alpha||\nabla f(x_k)||$，正比于梯度的模长。可以想象，随着迭代的进行，梯度值会越来越小，步长也会越来越小，这符合我们的预期。这个公式也许是最朴素的步长估计方法。

在机器学习中，我们一般使用随机梯度下降法。因为机器学习中的优化函数一般定义在样本集上，要想得到这个优化函数的梯度，需要遍历所有的样本数据，这样计算量太大。随机梯度下降法是指每次选取优化函数在某一个样本点上的梯度去代替整个样本集上的梯度，去训练模型。

3.5 牛顿法和拟牛顿法

我们知道梯度下降法是线性收敛的，而牛顿法是二次收敛的。

3.5.1 牛顿法的由来

在当前点 x_k 处，我们可以利用 x_k 的一阶导数 (梯度 $\nabla f(x_k)$) 和二阶导数 (Hesse 矩阵 $\nabla^2 f(x_k)$) 去拟合目标函数 $f(x)$，那么有

$$\hat{f}(x) = f(x_k) + \nabla f(x_k)(x - x_k) + \frac{1}{2}(x - x_k)^T \nabla^2 f(x_k)(x - x_k)$$

这里 $\hat{f}(x)$ 也可以看做目标函数 $f(x)$ 的二阶泰勒展开式，因为拟合函数 $\hat{f}(x)$ 是二次的，很容易得到它的最值点

$$\nabla \hat{f}(x) = \nabla f(x_k) + \nabla^2 f(x_k)(x - x_k) = 0$$

如果我们将这个最值点当作下一次的迭代点 x_{k+1}，那么有

$$x_{k+1} = x_k - (\nabla^2 f(x_k))^{-1} \nabla f(x_k)$$

这样的话，如果我们以线搜索方法的角度来看这个更新公式:

$$\alpha_k = 1, \quad d_k = -(\nabla^2 f(x_k))^{-1} \nabla f(x_k)$$

3.5.2 牛顿法的缺陷

可惜的是牛顿法只猜中了开头，没有猜中结局。

工程实践中，几乎没有人会使用牛顿法去解一个优化问题，因为它有明显的缺陷:

第一，如果 Hesse 矩阵 $\nabla^2 f(x_k)$ 不可逆，迭代公式就没法求解。

第二，即使 Hesse 矩阵 $\nabla^2 f(x_k)$ 可逆，也不一定正定。如果 $\nabla^2 f(x_k)$ 不是正定矩阵的话，那么 $d_k = -(\nabla^2 f(x_k))^{-1}\nabla f(x_k)$ 不是下降方向。当迭代点远离目标函数的最小值点时，这种情况很容易出现。

如下图 (2) 所示，当前点 x_k 处拟合的二次函数开口向下，最值点为最大值点。如果 Hesse 矩阵 $\nabla^2 f(x_k)$ 是正定矩阵，可以形象地理解为保证拟合的高维空间的曲面开口朝上，那么 d_k 一定是下降方向。

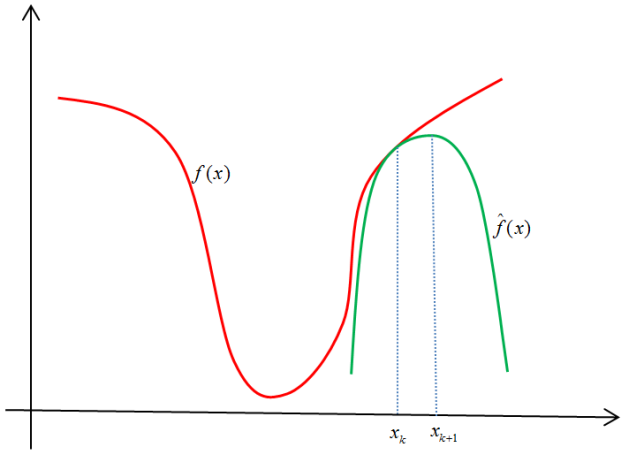


图 2: 牛顿法的缺陷

第三，在得到下降方向 d_k 时，需要重新计算当前点的 Hesse 矩阵 $\nabla^2 f(x_k)$，并且还要去解一个线性方程组 $\nabla^2 f(x_k)d = -\nabla f(x_k)$，这样计算量比较大。

3.5.3 拟牛顿法

为了克服牛顿法的缺点，同时想要具有类似牛顿法的比较快的收敛速度，有大量的拟牛顿法 (Quasi-Newton methods) 被提出。

这类算法的基本思路是用一个矩阵 H_k 去近似 Hesse 矩阵的逆 $(\nabla^2 f(x_k))^{-1}$, 同时保证：

第一，矩阵 H_k 是正定的，这样可以保证得到的方向是下降的；

第二，矩阵 H_k 的更新规则比较简单，最好可以迭代求出。

最著名的拟牛顿法就是 BFGS 算法，它是由 Broyden, Fletcher, Goldfarb 和 Shanno 四个人于 1970 年提出来的。

BFGS 方法中矩阵 H 的更新公式如下：

$$H_{k+1} = H_k + \frac{s_k s_k^T}{y_k^T s_k}(1 + \frac{y_k^T H_k y_k}{y_k^T s_k}) - \frac{s_k y_k^T H_k + H_k y_k s_k^T}{y_k^T s_k}$$

其中 $s_k = x_{k+1} - x_k$ 为位移差, $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$ 为梯度差。这样，每一次迭代的下降方向为

$$d_k = -H_k \nabla f(x_k)$$

一般来说，BFGS 方法会搭配上 Wolfe 条件估计步长，这样可以保证矩阵 H_k 的正定性。这其中具体的理论推导，大家参考任何一本优化理论的教材。

BFGS 算法流程如下：

Algorithm 4 BFGS method

1: initialize $x_0, H_0 = I, k = 0$
2: **while** $||\nabla f(x_k)|| > \epsilon$ **do**
3: $d_k = -H_k \nabla f(x_k)$;
4: obtain α_k using Wolfe search;
5: $x_{k+1} = x_k + \alpha_k d_k$;
6: update H_k;
7: $k = k + 1$;
8: **end while**
9: return x_k;

4 信赖域方法

信赖域方法的思想不同于前面所讲的线搜索方法。线搜索方法是“两步走”战略，先方向再步长。而信赖域方法是直接确定位移，得到下一次迭代点的位置。

信赖域方法的基本思想是，首先以当前迭代点 x_k 为中心，给出一个信赖域半径 r_k,，确定一个信赖域 $\{x : ||x - x_k|| \leq r_k\}$。

接着，在这个信赖域中，求解一个目标函数 $f(x)$ 的近似子问题

$$\begin{aligned} min \quad & \hat{f}(x) = f(x_k) + \nabla^T f(x_k)(x - x_k) + \frac{1}{2}(x - x_k)^T \nabla^2 f(x_k)(x - x_k) \\ s.t. \quad & ||x - x_k|| \leq r_k \end{aligned} \tag{1}$$

我们将这个子问题的最优解当做下一次迭代点的候选点，记为 $\hat{x}_{k+1}$。

最后，通过定义一个比值 ρ_k 来判断这个候选点 $\hat{x}_{k+1}$ 好不好，并决定是否扩大信赖域半径。这个比值是目标函数的实际下降量与近似函数的预测下降量之比，即

$$\rho_k = \frac{f(x_k) - f(\hat{x}_{k+1})}{\hat{f}(x_k) - \hat{f}(\hat{x}_{k+1})}$$

下面对 ρ_k 分情况讨论：

如果 ρ_k 较大，接近于 1.0，表示在当前信赖域中，这个近似函数较好地逼近了目标函数，可以采纳候选点 $\hat{x}_{k+1}$ 作为下一步迭代点，并考虑下一步扩大信赖域的半径。

如果 ρ_k 较小，接近于 0，表示在当前信赖域中，近似函数 $\hat{f}(x)$ 与目标函数 $f(x)$ 差别较大，拒绝接受候选点 $\hat{x}_{k+1}$，同时缩小信赖域半径并重新求解子问题。

如果 ρ_k 适中，可以考虑认可候选点 $\hat{x}_{k+1}$，但下一步不扩大信赖域半径。

信赖域方法的大致流程如下，其中的参数设置可以自行调整。

Algorithm 5 Trust Region method

initialize $x_0, r_0, \beta_1 = 0.05, \beta_2 = 0.75, \mu_1 = 0.5, \mu_2 = 2, k = 0$
while $||\nabla f(x_k)|| > \epsilon$ **do**
  obtain candidate $\hat{x}_{k+1}$;
  compute ρ_k;
  if $\rho_k \leq \beta_1$ **then**
    $r_{k+1} = \mu_1 r_k, \quad x_{k+1} = x_k$;
  else if $\beta_1 < \rho_k < \beta_2$ **then**
    $r_{k+1} = r_k, \quad x_{k+1} = \hat{x}_{k+1}$;
  else if $\rho_k \geq \beta_2$ **then**
    $r_{k+1} = \mu_2 r_k, \quad x_{k+1} = \hat{x}_{k+1}$;
  end if
  $k = k + 1$;
end while
return x_k;

这时，大家可能会有疑问，这个信赖域上的子问题是不是在推导牛顿法时见过?

其实，这两个问题是不同的。在牛顿法中，我们选取的是近似函数的稳定点 (梯度为 0 的点)，而这里，我们求解的是在信赖域上的近似函数的最小值，不一定是稳定点。如下图 (3) 所示，大家稍微看一下，就会明白。

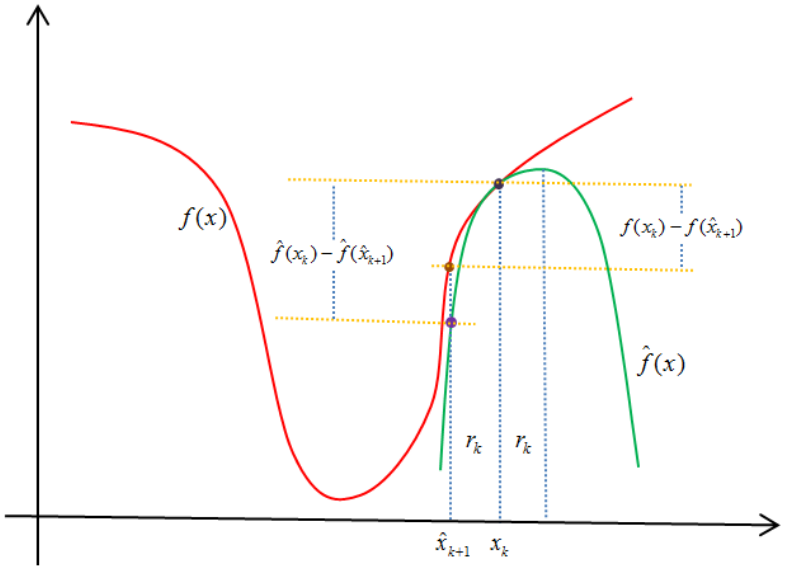


图 3: 信赖域方法的示意图

这里还有一个问题，如何去求解给定信赖域上的近似函数 (二次函数) 的最小值。这个问题看上去不难，其实已经被很好地解决了。如果大家以后需要使用信赖域方法的话，可以参考优化理论的教材去学习一下，这里就不再介绍了。