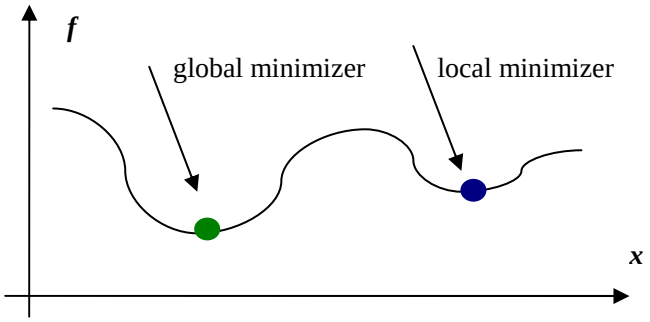
无约束最优化

本篇文档主要介绍无约束最优化问题，同时初步介绍解该类问题目前常用的一种算法即 Quasi-Newton Method (拟牛顿法)。在介绍无约束最优化问题之前，我们首先会从直观上引入无约束最优化的概念，并在此基础上引入解这类问题的两个重要概念：步长和方向。由步长的选择引入重要概念 line search，由方向的选择引入重要概念 Quasi-Newton Method。因此本篇介绍文档主要分为以下几个部分：无约束最优化问题引入，Line Search，Quasi-Newton Method 和算法总结。

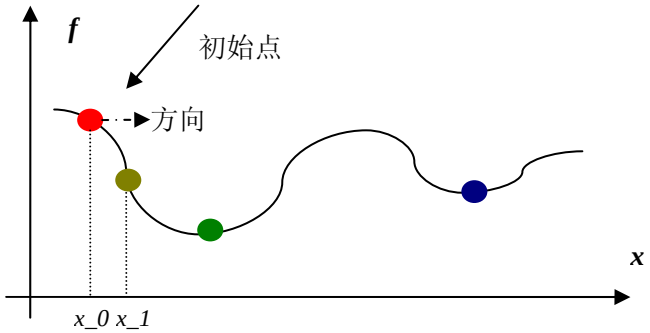
1. 无约束最优化

对无约束最优化不熟悉的读者也许要问，什么是无约束最优化。这里以一个例子来说明该问题。



上图所示为一元函数 *f(x)*的图像，无约束最优化问题，即不对定义域或值域做任何限制的情况下，求解函数 *f(x)*的最小值，上面显示两个最小值点：一个为全局最小值点，另一个为局部最小值点。受限于算法复杂度等问题，目前大部分无约束最优化算法只能保证求取局部最小值点。这时读者不免要问，既然只能求取局部最小值点，那为什么这类算法还能应用呢。这是因为实际应用中，许多情形被抽象为函数形式后均为凸函数，对于凸函数来说局部最小值点即为全局最小值点，因此只要能求得这类函数的一个最小值点，该点一定为全局最小值点。

理解了上面的无约束最优化问题之后，我们就可以开始介绍无约束最优化的求解过程了，对于无约束最优化的求解首先我们需要选择一个初始点 *x_0*，如下所示：



初始点选择好之后，就可以按照各种不同的无约束最优化求解算法，求解最小值点了。求解过程中主要涉及两个概念，即从初始点开始沿“哪个方向”以及“走多远”到达下一个点处。

所谓“走多远”即之前提的“步长”的概念，“哪个方向”即方向概念。

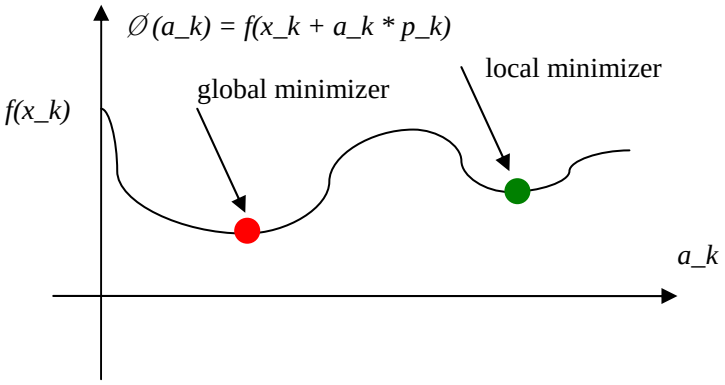
2．Line Search

Line search 主要用于解决之前提到的步长的概念，即方向确定好之后，需要确定从当前点 *x_k* 沿着该方向走多远，以便走到下一个合适的点 *x_k+1*。若用 *p_k* 代表从第 *k* 个点走向第 *k+1* 点的方向，*x_k* 代表当前点，*x_k+1* 代表下一个点，*a_k* 代表步长，则存在如下的等式：

$$x\_k+1 = x\_k + a\_k * p\_k \tag{1}$$

这里简要介绍一下 *p_k*，大部分 line search 方法要求 *p_k* 为下降方向，即从当前点沿着 *p_k* 方向移动后导致函数值减少。由于目标是求取一个函数的最小值，因此最优的情况是求取沿 *p_k* 方向满足 *f(x_k+1)*为全局最小的 *a_k* 值，可用下式表示为：

$$\varnothing(a\_k) = f(x\_k+1) = f(x\_k + a\_k * p\_k) \tag{2}$$



由于直接求取满足 *∅(a_k)*为全局最小值的 *a_k* 涉及到大量 *f(x_k + a_k * p_k)*的计算，若从求导角度计算最小值，还会涉及到 *▽f_k+1* 的计算，计算量较大。因此从计算量角度考虑，可以采用如下较为折中的策略。

方向确定好之后，每一步的 line Search 主要涉及两个问题：1）什么样的 *a_k* 是合理的 2）如何选择 *a_k* 的长度。下面将沿这两个面展开讨论，首先讨论“什么样的 *a_k* 是合理的”，确定了该问题之后，我们就可以在此基础上选择 *a_k* 了。

2.1 *a_k* 合理性讨论

如下将要讨论关于*a_k*需要满足的两个条件，当*a_k*满足这两个条件后，就可以认为从*x_k* 点移动到*x_k+1* 点的步长已经确定下来了。第一个条件为sufficient decrease condition，从直观角度来看，该条件主要要用保证*x_k+1* 点的函数值要小于*x_k*点的函数值，满足该条件后，才有全局收敛[1]的可能性。第二个条件为curvature condition，从直观角度来看，该条件主要用于保证*x_k*点经过步长*a_k*的移动到达*x_k+1* 后， *▽f_k+1* 小于 *▽f_k*。

2.1.1 sufficient decrease condition

a_k 的选择一定要使得函数值满足 sufficient decrease condition，该条件可以用如下不等式描述：

$$f(x\_k+1) \leqslant f(x\_k) + c\_1 * a\_k * \nabla f\_k^T * p\_k \tag{3}$$

将公式（1）代入上式，可得：

$$f(x\_k + a\_k * p\_k) \leqslant f(x\_k) + c\_1 * a\_k * \nabla f\_k^T * p\_k \tag{4}$$

[1] 注：这里以及后面所说的全局收敛，均指的是可以收敛到一个局部最小值处

这里有必要对上面的不等式做一些解释：

a) $f(x\_k)$ 代表函数在第 x_k 点的值

b) ∇f_k 代表函数在第 x_k 点的梯度

c) p_k 代表从第 x_k 点的走到 x_k+1 点的方向

d) a_k 代表从第 x_k 点沿着 p_k 方向走到 x_k+1 点步长

e) c_1[2]为常量，需满足 $0< c\_1 < 1$，一般取c_1 为 1E-4

当 p_k 为函数下降方向时，有：

$$\nabla f\_k * p\_k < 0 \tag{5}$$

因此不等式 4，即要求：

$$f(x\_k+1) < f(x\_k) \tag{6}$$

从图形角度看，函数位于第 k 点时，以上各参数中只有 a_k 为变量，其他均为常量，因此(4)可以用以下不等式重新描述为：

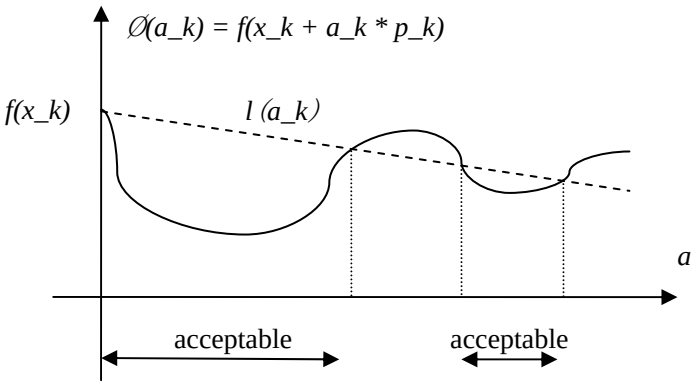
$$\varnothing(a\_k) \leqslant l(a\_k) \tag{7}$$

其中：

$$\varnothing(a\_k) = f(x\_k + a\_k * p\_k)$$

$$l(a\_k) = f(x\_k) + c\_1 * a\_k * \nabla f\_k^T * p\_k$$

以下为不等式（7）的图形化表示：



因此只要步长 a_k 的选择使得函数 $\varnothing(a_k)$位于 acceptable 区间，就满足 sufficient decrease condition。

2.1.2 curvature condition

a_k 的选择一定要使得函数梯度值满足 curvature condition，该条件可以用如下不等式描述：

$$\nabla f(x\_k + a\_k * p\_k)^T * p\_k \geqslant c\_2 * \nabla f(x\_k)^T * p\_k \tag{8}$$

即为

$$\nabla f\_k+1^T * p\_k \geqslant c\_2 * \nabla f\_k^T * p\_k \tag{9}$$

这里有必要对上面的不等式做一些解释：

a) ∇f_k+1 代表函数在第 x_k+1 点的梯度

b) ∇f_k 代表函数在第 x_k 点的梯度

c) p_k 代表从第 k 点的走到 $k+1$ 点的方向

d) c_2 为常量，需满足 $0< c\_1 < c\_2 < 1$，一般取 c_2 为 0.9

当 p_k 为函数下降方向时，有：

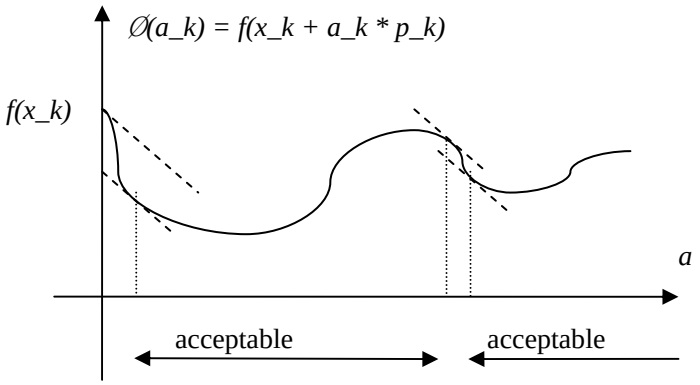
[2] 注：Quasi-Newton Method 中要求 $0< c\_1< 0.5$

$$\nabla f\_k * p\_k < 0$$

因此不等式 9，即要求：

$$\nabla f\_k+1 \geq c\_2 * \nabla f\_k \quad (10)$$

从图形角度看，不等式 10 即要求函数在第 x_k+1 点的变化速度要低于 x_k 点的变化速度，这一点可以从这两点处的梯度处看出，如下图所示。



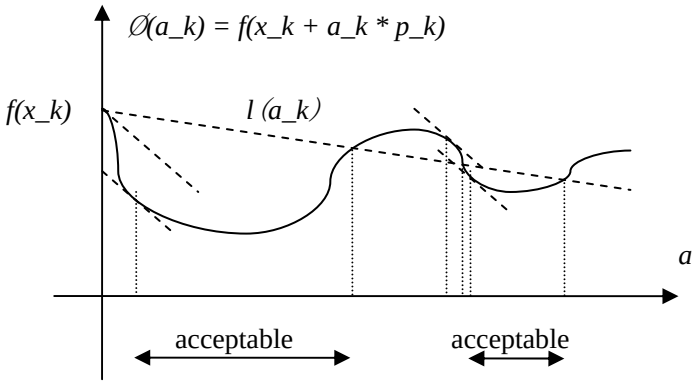
2.1.3 Wolfe conditions

所谓 Wolfe conditions 即 sufficient decrease condition 和 curvature condition 的综合，即 *a_k* 需要同时满足如下两个条件：

$$f(x\_k + a\_k * p\_k) \leq f(x\_k) + c\_1 * a\_k * \nabla f\_k^T * p\_k \quad (11)$$

$$\nabla f(x\_k + a\_k * p\_k)^T * p\_k \geq c\_2 * \nabla f(x\_k)^T * p\_k \quad (12)$$

图形化表示后，如下图所示：



实际应用中，常常会用到由 Wolfe conditions 引申出的 strong Wolfe conditions，即 a_k 需要同时满足如下两个条件：

$$f(x\_k + a\_k * p\_k) \leq f(x\_k) + c\_1 * a\_k * \nabla f\_k^T * p\_k \quad (13)$$

$$|\nabla f(x\_k + a\_k * p\_k)^T * p\_k| \leq c\_2 * |\nabla f(x\_k)^T * p\_k| \quad (14)$$

与 Wolfe condtions 唯一的区别是 strong Wolfe condtions 避免了 $\nabla f(x_k + a_k * p_k)$取较大的正值情况。

2.2 *a_k* 步长的选择

了解了 *a_k* 的合理性之后，就相当于获得了标尺，在此基础上我们可以选择合适的策略来求取 *a_k*。所有的 line search 过程在计算每一步的 *a_k* 时，均需要提供一个初始点 *a_0*，然后再此基础上生成一系列的{*a_i*}，直到 *a_i* 满足 2.1 节所规定的条件为止，此时该 *a_k* 即

被确定为 *a_i*，或者未找到一个合适的 *a_k*。这里我们仅介绍目前常用的策略平方插值和立方插值法。因此本节内容分为两部分，2.2.1 节介绍选择 *a_k* 常用的平方插值和立方插值法，2.2.2 节介绍由 *x_k* 点到 *x_k+1* 点，方向确定为 *p_k* 后，步长 *a_k* 具体计算过程。

2.2.1 平方立方插值法

当给定一个初始步长 a_0，若该初始步长满足 Wolfe conditions（或 strong Wolfe conditions），则 a_k 被确定为 a_0，当前点 x_k 步长计算过程结束，否则，我们可以在此基础上，利用已知的三个信息 $\emptyset(a\_0)$、$\emptyset(0)$、$\emptyset'(0)$，构建一个二次（平方）插值多项式来拟合 $\emptyset(a\_k)$。该二次插值多项式如下所示：

$$\emptyset\_q(a) = ((\emptyset(a\_0) - \emptyset(0) - a\_0*\emptyset'(0)) / a\_0^{2})*a^2 + \emptyset'(0)*a + \emptyset(0) \quad (15)$$

观察上面的二次插值多项式可知，其满足如下插值条件：

$$\emptyset\_q(0) = \emptyset(0) \qquad \emptyset\_q'(0) = \emptyset'(0) \qquad \emptyset\_q(a\_0) = \emptyset(a\_0)$$

对二次插值多项式(15)求导并令其为零，即可获得使得该多项式取得最小值的 *a* 值，如下所示：

$$a\_1 = -1 * \emptyset'(0) * a\_0^{2} / (2 *(\emptyset(a\_0) - \emptyset(0) - \emptyset'(0)*a\_0)) \quad (16)$$

若 *a_1* 满足 Wolfe conditions（或 strong Wolfe conditions），则 *a_k* 被确定为 *a_1*。否则在此基础上构建一个三次（立方）插值多项式，并求得使得该多项式取最小值的 *a* 值，该 *a* 值的计算公式如下所示：

$$a\_i+1 = a\_i - (a\_i - a\_i-1)*(\emptyset'(a\_i) + d\_2 - d\_1) / (\emptyset'(a\_i) - \emptyset'(a\_i-1) + 2*d\_2) \quad (17)$$

$$d\_1 = \emptyset'(a\_i-1) + \emptyset'(a\_i) - 3*(\emptyset(a\_i-1) - \emptyset(a\_i)) / (a\_i-1 - a\_i)$$

$$d\_2 = sign(a\_i - a\_i-1) * (d\_1^{2} - \emptyset'(a\_i-1) * \emptyset'(a\_i))^{1/2}$$

若 *a_i*+1 满足 Wolfe conditions（或 strong Wolfe conditions），则 *a_k* 被确定为 *a_i+1*，否则一直使用三次插值多项式进行插值拟合。并且选择使用 *a_i+1* 对应的 $\emptyset(a_i+1)$和 $\emptyset'(a\_i+1)$ 替换 *a_i-1* 或 *a_i* 相应的值，一旦确定好 *a_i-1* 或 *a_i* 中的一种后，每次都替换对应的值，如替换 *a_i-1*，则每次都使用新的 *a_i+1* 对应的值替换 *a_i-1*，直到找到满足 Wolfe conditions（或 strong Wolfe conditions）的 *a_i+1* 为止，此时 *a_k* 被确定为 *a_i+1*，或者没有找到合适的 *a_i+1*。

这里有必要简略解释一下为何只使用到三次插值多项式，而没有使用更高阶的插值多项式。原因是三次插值多项式对函数某个点处的具体值有较好的拟合效果，同时又有较好的抗过拟合作用。

最后有必要解释一下初始步长 *a_0* 的选择，对于 Newton 或 quasi-Newton methods 来说，初始步长 *a_0* 总是确定为 1，该选择确保当满足 Wolfe conditions（或 strong Wolfe conditions），我们总是选择单位 1 步长，因为该步长使得 Newton 或 quasi-Newton methods 达到较快的收敛速度。计算第零步长时，将初始步长 *a_0* 使用如下公式确定：

$$a\_0 = 1.0 / (p\_0^{T} * p\_0)^{1/2} \quad (18)$$

第 1 步及其以后的初始步长 *a_0*，直接设定为 1。

2.2.2 步长 *a_k* 计算

本节主要做一个总结，即综合前面步长需要满足的条件及步长迭代计算公式给出步长计算的具体过程。下面假设我们处于 *x_k* 点，因此要从该点选择一个满足 Wolfe (或 strong Wolfe) conditions 的步长 *a_k*，以便走到下一点 *x_k+1*。因此我们选择初始点 *a_0 = 1*，Newton 或 quasi-Newton method 中初始步长总是选择为 1。因此有：

1) 初始化 $a\_x \leftarrow a\_y \leftarrow 0$　$\emptyset(a\_x)$ 、 $\emptyset(a\_y)$ 、$\emptyset'(a\_x)$ 、 $\emptyset'(a\_y)$、*a_min*、*a_max*

2) 初始化 $a\_i \leftarrow a\_0$、$\emptyset'(a\_i)$、$\emptyset(a\_i)$

3) 若 $\varnothing(a\_i) > \varnothing(a\_x)$

说明步长 *a_i* 选择的过大，因此使得 $\varnothing(a_i)$满足 Wolfe(或 strong Wolfe) conditions 的步长 *a_k* 应位于 *a_x* 和 *a_i* 之间，使用平方和立方插值法对 *a_x* 和 *a_i* 分别进行插值，求取两个新的步长 *a_quadratic* 和 *a_cubic*，并取两者之中和 *a_x* 较接近的步长，这里假设为 *a_quadratic*，将新的步长设为 *a_qudratic*，同时进行以下操作：

$a\_y \leftarrow a\_i$

$\varnothing(a\_y) \leftarrow \varnothing(a\_i)$

$\varnothing'(a\_y) \leftarrow \varnothing'(a\_i)$

$a\_i+1 \leftarrow a\_quadrati$

同时在此情况下，说明 *a_k* 位于 *a_x* 和 *a_y* 之间

4) 若 $\varnothing'(a\_i)^* \varnothing'(a\_x) < 0$

说明步长 *a_i* 选择的过大，因此使得 $\varnothing(a_i)$满足 Wolfe(或 strong Wolfe) conditions 的步长 *a_k* 应位于 *a_x* 和 *a_i* 之间，使用平方和立方插值法对 *a_x* 和 *a_i* 分别进行插值，求取两个新的步长 *a_quadratic* 和 *a_cubic*，并取两者之中和 *a_i* 较接近的步长，这里假设为 *a_quadratic*，将新的步长设为 *a_qudratic*，同时进行以下操作：

$a\_y \leftarrow a\_x$

$\varnothing(a\_y) \leftarrow \varnothing(a\_x)$

$\varnothing'(a\_y) \leftarrow \varnothing'(a\_x)$

$a\_x \leftarrow a\_i$

$\varnothing(a\_x) \leftarrow \varnothing(a\_i)$

$\varnothing'(a\_x) \leftarrow \varnothing'(a\_i)$

$a\_i+1 \leftarrow a\_quadrati$

同时在此情况下，说明 *a_k* 位于 *a_x* 和 *a_y* 之间

5) 若 $|\varnothing'(a\_i)| < |\varnothing'(a\_x)|$

说明步长 *a_i* 选择的过小，因此使得 $\varnothing(a_i)$满足 Wolfe(或 strong Wolfe) conditions 的步长 *a_k* 应位于 *a_i* 和 *a_y* 之间，使用平方和立方插值法对 *a_x* 和 *a_i* 分别进行插值，求取两个新的步长 *a_quadratic* 和 *a_cubic*，并取两者之中和 *a_i* 较接近的步长，这里假设为 *a_quadratic*，将新的步长设为 *a_qudratic*，同时进行以下操作：

$a\_x \leftarrow a\_i$

$\varnothing(a\_x) \leftarrow \varnothing(a\_i)$

$\varnothing'(a\_x) \leftarrow \varnothing'(a\_i)$

$a\_i+1 \leftarrow a\_quadrati$

同时在此情况下，说明 *a_k* 位于 *a_x* 和 *a_y* 之间

5) 若 $|\varnothing'(a\_i)| \geqslant |\varnothing'(a\_x)|$

说明步长 a_i 选择的过小，若之前已经确定出了 *a_k* 所属的范围 *a_x* 和 *a_y*，则使得 $\varnothing(a\_i)$ 满足 Wolfe(或 strong Wolfe) conditions 的步长 *a_k* 应位于 *a_i* 和 *a_y* 之间，使用立方插值法对 *a_i* 和 *a_y* 进行插值，求取新的步长 *a_cubic*，新的步长设为 *a_cubic*；否则若 *a_x* 小于 *a_i*，说明的新的步长不够大，因此将新的步长设置为 *a_max*，若 *a_x* 大于等于 *a_i*，则说明新的步长不够小，将其设为 *a_min*，同时进行以下操作：

$a\_x \leftarrow a\_i$

$\emptyset(a\_x) \leftarrow \emptyset(a\_i)$
$\emptyset'(a\_x) \leftarrow \emptyset'(a\_i)$

if 之前已经确定出了 *a_k* 所属的范围 *a_x* 和 *a_y*
    a_i+1 ← *a_cubic*
else if *a_x* < *a_i*
    a_i+1 ← *a_max*
else if *a_x* ≥ *a_i*
    a_i+1 ← *a_min*

6) 计算判断若 *a_i+1* 使得以下两式(strong Wolfe condition)均成立：

$$f(x\_k + a\_i+1 * p\_k) \leq f(x\_k) + c\_1 * a\_i+1 * \nabla f\_k^T * p\_k$$

$$|\nabla f(x\_k + a\_i+1 * p\_k)^T * p\_k| \leq c\_2 * |\nabla f(x\_k)^T * p\_k|$$

则找到合理的步长 a_k，将其设定为 a_i+1

$a\_k \leftarrow a\_i+1$

则 *x_k* 点步长 *a_k* 计算结束。

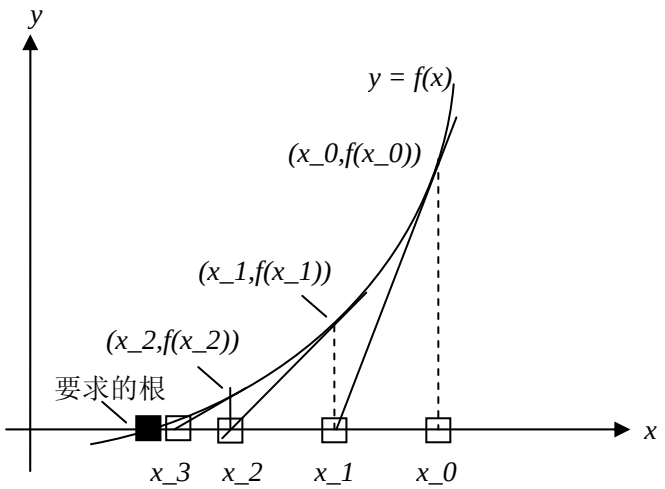
否则转到 2）继续计算合理步长。

3．Quasi-Newton Method

在第 2 节中我们了解了步长的概念，以及从 x_k 走到 x_k+1 点使用 line search 方法计算步长的方法。不过我们在那里忽略了一个重要的概念，即“方向”。从第 2 节，我们了解到从每一点 x_k 走到下一点 x_k+1 时，需要给出要走的“方向”，只有“方向”确定好之后，才能在此基础上应用 line search 方法找到对应的“步长”，因此在解决了“步长”计算问题之后，这里我们将和大家一起了解一下每一步的“方向”如何确定。本节分为 2 大部分，首先我们通过 newton method 引入方向的概念，在此基础上引入 quasi-newton method。然后引入 quasi-newton method 中的一种重要方法 BFGS method，并在 BFGS method 的基础上介绍用于大规模计算的 LBFGS method 算法，同时以此结束本节的所有内容。

3.1 Newton Method

我想我们还依稀记得，微积分中用于求一元函数根的 Newton 法。该方法可用如下所示的图形来描述：



首先我们选择一个初始点 *x_0* 并计算获得其对应的 *f(x_0)*，然后该方法用曲线 y = f(x)在 *(x_0,f(x_0))*的切线近似该曲线，把切线和 x 轴的交点记作 x_1，点 x_1 通常 x_0 更接近所要

求得根，同样方法用点 x_1 处的切线近似该曲线，并求取切线和 x 轴的交点 x_2，一直迭代下去，直到找到满足我们所需的充分接近真实跟的解为止。因此 Newton 法从第 k 次近似值 x_k 求得第 k+1 次近似值 x_k+1 即为求 x_k 点处的切线和 x 轴的交点。

切线方程为：

$$y - f(x_k) = \nabla f(x_k)(x - x_k)$$

$$=> \quad 0 - f(x_k) = \nabla f(x_k)(x - x_k)$$

$$=> \quad \nabla f(x_k) * x = \nabla f(x_k) * x_k - f(x_k)$$

$$=> \quad x = x_k - f(x_k) / \nabla f(x_k)$$

因此 $x_{k+1} = x_k - f(x_k) / \nabla f(x_k)$ (19)

从上面使用 Newton Method 求函数根的过程可以发现，首先需要选择一个初始点，并在点处构建一个模型来近似该函数。

切线模型： $M_k(x_k + p_k) = f(x_k) + \nabla f(x_k)^T * p_k$

上面使用了相应点处的切线模型来近似函数，然后求取该近似模型的根以便求得更接近函数根的下一个点，该过程一直迭代下去，直到找到根为止。从上面构建每个点处的近似模型可以发现，该模型相对原函数来说简化了很多，因此求解要容易一些。

现在来考虑求取函数的最小值问题，方法类似。首先开始我们选择一个初始点 x_0，并构建函数在该点处的一个近似模型，上面求函数根时，我们构建的近似模型为切线模型。这里我们构建一个抛物线模型：

抛物线模型：$M_k(x_k + p_k) = f(x_k) + \nabla f(x_k)^T * p_k + 1/2 * p_k^T * \nabla^2 f(x_k) * p_k$

并求解该模型的梯度，同时令其为零，即：$\nabla M_k(x+) = 0$，在此基础上求得 $x+$，该值即使得近似模型取得最小值的点。

对抛物线模型 M_k 求导并令其为零后，可得以下公式：

$$p_k = -\nabla^2 f(x_k)^{-1} * \nabla f(x_k) \quad (20)$$

因此 $x_{k+1} = x_k - \nabla^2 f(x_k)^{-1} * \nabla f(x_k)$ (21)

从上式可见，使用Newton法时，每一步的方向为p_k，而步长为 1。从Newton法方向公式我们不难看出，若使用Newton法，则从每个最小值近似点x_k走到下一个近似点x_k+1 的过程中将涉及函数Hessian矩阵 $\nabla\nabla f(x_k)$ (二阶导)计算，而该Hessian矩阵无法保证在每个点处都是正定[3]的，对正定矩阵来说存在如下不等式：

$$p_k^T * \nabla^2 f(x_k) * p_k > 0 \quad (22)$$

由于矩阵无法保证为正定矩阵，因此下式中

$$-\nabla f(x_k)^T * p_k = p_k^T * \nabla^2 f(x_k) * p_k \quad (23)$$

p_k 无法保证总是为下降方向，即负梯度方向。

从上面的分析可见，Newton Method 面临两个问题：

1) Hessian 矩阵计算量较大
2) Hessian 矩阵无法保证在迭代的过程中始终处于正定状态

为了解决这两个问题，我们引入 Quasi-Newton Method。

3.2 Quasi-Newton Method

Quasi-Newton Method 每一步计算过程中仅涉及到函数值和函数梯度值计算，这样有效避免了 Newton Method 中涉及到的 Hessian 矩阵计算问题。于 Newton Method 不同的是 Quasi-Newton Method 在每点处构建一个如下的近似模型：

[3] 关于矩阵正定可以查阅 Linear Algebra and Its Applications 一书的最后一章

$$M\_k(x\_k + p\_k) = f(x\_k) + \nabla f(x\_k)^T * p\_k + 1/2 * p\_k^T * B\_k * p\_k$$

从上面的近似模型我们可以看出，该模型用 *B_k* 代替了 Newton Method 中近似模型中涉及到的 Hessian 矩阵。因此 Quasi-Newton Method 中方向计算公式如下所示：

$$p\_k = -B\_k^{-1} * \nabla f(x\_k) \tag{24}$$

这里有必要解释一下用于近似 Hessian 矩阵的 *B_k* 可行性，及一个指导性方案。根据 Taylor(泰勒)级数可知如下公式：

$$\nabla f(x\_k + p\_k) = \nabla f(x\_k) + \nabla^2 f(x\_k)^T * p\_k + \int [\nabla^2 f(x\_k + t * p\_k) - \nabla^2 f(x\_k)] p\_k dt$$

由于函数 $\nabla f(.)$连续，因此上式可以表示为：

$$\nabla f(x\_k+1) = \nabla f(x\_k) + \nabla^2 f(x\_k) * (x\_k+1 - x\_k) + o(||x\_k+1 - x\_k||)$$

$$\nabla^2 f(x\_k) * (x\_k+1 - x\_k) \approx \nabla f(x\_k+1) - \nabla f(x\_k) \tag{25}$$

因此每一选择 Hessian 矩阵的近似 B_k+1 时，可以像式(24)那样模仿真实的 Hessian 矩阵的性质。得到下式：

$$B\_k+1 * s\_k = y\_k \tag{26}$$

其中：

$$s\_k = x\_k+1 - x\_k \qquad y\_k = \nabla f(x\_k+1) - \nabla f(x\_k) \tag{27}$$

同时要求 *B_k+1* 为对称正定矩阵。

3.2.1 BFGS Method

从 Quasi-Newton Method 方向公式 (24) 中，可以看到每一步计算方向的过程中均涉及到 *B_k+1* 矩阵求逆的问题，为了避免该计算，通过分析公式（26）可知，我们可以构建一个近似 *H_k+1*，该近视满足如下方程：

$$H\_k+1 * y\_k = s\_k \tag{28}$$

同时要求 *H_k+1* 为对称正定矩阵。因此 BFGS Method 中，每个点处的方向由如下公式计算：

$$p\_k = -H\_k * \nabla f(x\_k) \tag{29}$$

在此基础上，BFGS 方向迭代公式如下所示：

$$H\_k+1 = (1 - \rho\_k * s\_k * y\_k^T) H\_k (1 - \rho\_k * y\_k * s\_k^T) + \rho\_k * s\_k * s\_k^T \tag{30}$$

其中 ρ_k 为一个标量：

$$\rho\_k = 1 / (y\_k^T * s\_k)$$

有了上面(30)的 *H_k* 迭代公式后，还有一个问题就是初始的 *H_0* 如何计算，目前常用的方法是初始的 *H_0* 直接设为单位矩阵 *I*。因此 BFGS Method 用于解无约束最优化的过程可以表示为如下过程：

1 选择一个初始点 x_0，并选择收敛判断条件 $\varepsilon > 0$，及初始 H_0 (单位矩阵 I)

2 k ← 0

3 while || $\nabla f(x\_k)$|| > ε

4 计算从当前点 x_k 走到下一个点 x_k+1 的方向

$$p\_k = -H\_k * \nabla f(x\_k)$$

5 采用 line search 策略计算步长 a_k

6 $x\_k+1 = x\_k + a\_k * p\_k$

7 $s\_k = x\_k+1 - x\_k$ $y\_k = \nabla f(x\_k+1) - \nabla f(x\_k)$

8 $H\_k+1 = (1 - \rho\_k * s\_k * y\_k^T) H\_k (1 - \rho\_k * y\_k * s\_k^T) + \rho\_k * s\_k * s\_k^T$

9 $k \leftarrow k+1$

3.2.2 LBFGS Method

上一节所介绍的 BFGS Method 比较适合解决中小规模无约束最优化问题，但是 BFGS 算法产生的 Hessian 近似矩阵 H_k 为 n * n 的，同时该矩阵非稀疏，因此当 n 的规模较大时将面临两个问题：

1） 存储问题：n 规模较大时，n*n 矩阵对内存的消耗将较大；

2） 计算问题：n 规模较大，同时 n*n 矩阵非稀疏时，计算复杂度将较高；

为了解决以上问题，引申出了 Limited-Memory Quasi-Newton Method，目前使用较多的 LBFGS 算法即属于该类算法。为了减少 H_k 矩阵的存储，LBFGS 算法利用最近几代的 curvature 信息来构建 Hessian 矩阵的近似。由 BFGS Method 我们知道：

$$x\_k+1 = x\_k + a\_k * H\_k * \nabla f(x\_k)$$

其中 a_k 为步长，H_k 为 Hessian 矩阵的近似，可以通过如下迭代公式计算：

$$H\_k+1 = V\_k * H\_k * V\_k + \rho\_k * s\_k * s\_k \tag{31}$$

其中：

$$\rho\_k = 1 / (y\_k^T * s\_k) \qquad V\_k = 1 - \rho\_k * y\_k * s\_k^T$$

$$s\_k = x\_k+1 - x\_k \qquad y\_k = \nabla f(x\_k+1) - \nabla f(x\_k)$$

从上面的 H_k 的迭代计算公式可知，H_k 会慢慢由稀疏矩阵转变为稠密矩阵，因此存储该矩阵以及进行该矩阵和向量的相乘运算的消耗将较大。为了避免该问题，LBFGS 算法在 BFGS 算法的基础上从两点进行了改进：

1）估算每一步对应的 Hessian 近似矩阵时，给出一个当前步的初始 Hessian 矩阵估计 H_k^0

2) 利用过去当前代及过去 m-1 代的 curvature 信息修正初始 Hessian 矩阵估计 H_k^0，得到最终的 Hessian 矩阵近似估计 H_k。

计算式如下所示：

$$H\_k = (V\_k\text{-}1^T \dots V\_k\text{-}m^T) * H\_k^0 * (V\_k\text{-}m \dots V\_k\text{-}1)$$
$$+ \rho\_k\text{-}m * (V\_k\text{-}1^T \dots V\_k\text{-}m+1^T) * s\_k\text{-}m * s\_k\text{-}m^T * (V\_k\text{-}m+1 \dots V\_k\text{-}1)$$
$$+ \rho\_k\text{-}m+1 * (V\_k\text{-}1^T \dots V\_k\text{-}m+2^T) * s\_k\text{-}m+1 * s\_k\text{-}m+1^T * (V\_k\text{-}m+2 \dots V\_k\text{-}1)$$
$$+ \dots$$
$$+ \rho\_k\text{-}1 * s\_k\text{-}1 * s\_k\text{-}1^T \quad (32)$$

上述计算式(32)，可以通过公式(31)递归计算获取。公式(32)可以用以下算法表示：

```
1  q  ←  ▽f(x_k)
2  for i = k-1, k-2, …, k-m
3        α_i  ← ρ_i * s_i^T * q;
4        q    ← q - α_i * y_i
5  end
6  r  ← H_k^0 * q;
7  for i = k-m, k-m+1, …, k-1
8        β  ← ρ_i * y_i^T * r
9        r  ← r + s_i*(α_i – β)
10 end
11 stop with result H_k* ▽f(x_k) = r
```

从上面计算 H_k 的公式(32)可知，要估算每个点 x_k 处的 Hessian 矩阵近似，需要给出初始估计 H_k^0，H_k^0 一般通过以下公式计算：

$$H\_k^0 = r\_k * I$$
$$r\_k = (s\_k\text{-}1^T * y\_k\text{-}1)/(y\_k\text{-}1^T * y\_k\text{-}1)$$

有了上面的方向计算算法后，LBFGS 算法用于解无约束最优化问题，可以表示为如下算法：

1 选择一个初始点 x_0，并选择收敛判断条件 $\varepsilon > 0$，以及常量 m(代表过去代数)一般为 6

2 k ← 0 H_0 ← I，因此 $r = H\_0 * \nabla f(x\_0) = \nabla f(x\_0)$

3 while $\| \nabla f(x\_k) \| > \varepsilon$

4 计算从当前点 x_k 走到下一个点 x_k+1 的方向

$$p\_k = -r$$

5 采用 line search 策略计算步长 a_k

6 $x\_k+1 = x\_k + a\_k * p\_k$

7 if k > m

删除 LBFGS 计算 H_k 时用不上的向量对(s_k-m, y_k-m)

8　计算并保存 $s\_k = x\_k+1 – x\_k$　　$y\_k = \nabla f(x\_k+1) - \nabla f(x\_k)$

9　采用 LBFGS Hessian 矩阵近似算法计算 r

10　$k \leftarrow k+1$

4．算法总结

用于解无约束优化算法的 Quasi-Newton Method 中的 LBFGS 算法到这里总算初步介绍完了，不过这里笔者要承认的是这篇文档省略了许多内容，包括算法收敛性的证明以及收敛速度证明等许多内容。因此读者若希望对这一块有一个更深入的认识可以参考以下两本书：

1) Numerical Methods for Unconstrained Optimization and Nonlinear Equations
   J.E. Dennis Jr.　Robert B. Schnabel
2) Numerical Optimization
   Jorge Nocedal　Stephen J. Wright

同时我想引用一下侯捷老师的话：

大哉问。学习需要明师。但明师可遇不可求，所以退而求其次你需要好书，并尽早建立自修的基础。迷时师渡，悟了自渡，寻好书看好书，就是你的自渡法门。切记，徒学不足以自行，计算机是实作性很强的一门科技，你一定要动手做，最忌讳眼高手低。学而不思则罔，思而不学则殆，一定要思考、沉淀、整理。

因此这里附上一个我阅读后感觉代码还比较美观的 LBFGS 实现（C++）：

http://www.chokkan.org/software/liblbfgs/

Naoaki Okazaki

最后我不得不承认这篇文档，对许多读者来说也许是几张废纸，甚至使读者昏昏欲睡，或烦躁。因为该文档从头至尾并未涉及到一个例子。-_- ||

老天啊，饶恕我吧，毕竟愿望是美好的，但愿它能对你理解 LBFGS 算法有一点帮助，哪怕是一点点…

jianzhu
jzhu.nlp@gmail.com
2009-11-1-2009-12-1