我们已经熟悉，机器学习模型运作的一般流程为：假设出模型的形式（包含未知参数）；基于训练集，构造模型的目标函数及相应的（关于未知参数的）优化问题；求解优化问题（得出未知参数的最优解），得出模型的具体形式；利用得出的模型对新的样本进行预测。

在流程中，求解优化问题是很重要的一步。这些优化问题往往不存在解析解，或很难找到解析解，因此需要使用数值优化方法，迭代地搜索，逐步逼近最优解。

机器学习中用以求解优化问题的数值优化方法有很多种，大体上可以分为非启发式算法和启发式算法。

非启发式算法主要是基于梯度的算法，又可分为一阶梯度算法和二阶梯度算法。

启发式算法是指不是根据严格的函数或泛函理论，而是根据经验和直观构造的，很难分析其效率和效果（算法很难保证给出的解是最优解，只能提供“比较好”的解）的优化算法，包括粒子群算法、蚁群算法、模拟退火算法、遗传算法等。

启发式算法的优点在于它能达到全局最优而不只是局部最优。

优化问题按照是否有约束条件还可以分为约束优化和无约束优化。约束优化方法在SVM笔记中已有介绍，本篇笔记只介绍无约束优化方法。

Part 1、基于梯度的数值优化算法

基于梯度的数值优化算法的基本思想是给定初始参数值，利用梯度的信息，一步一步地搜索，每一步向目标函数值减小（对于极小优化问题）的方向移动一点。最终达到目标函数的极小点。

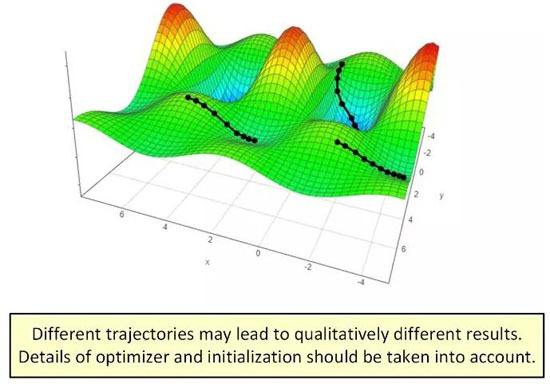


图1 基于梯度的优化算法示意图（二元函数，曲面图）

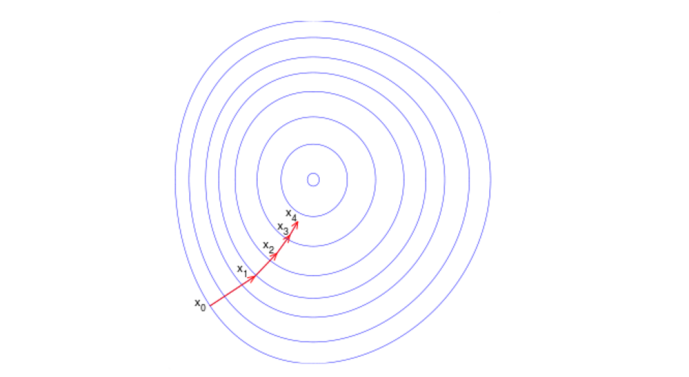


图2 基于梯度的优化算法示意图（二元函数，等高线图）

如图1和图2所示，对于一般的目标函数，存在多个极值点，选取不同的初始值，就会得到不同的局部极值点，一般很难取定得到的局部极值点是否是全局极值点。但对于凸函数（图2所示），则只有一个极值点，此时无论选取哪个初始点，最终都会到达全局极值点。

基于梯度的优化方法从初始点出发，每一步基于梯度（一阶梯度或二阶梯度）提供的信息搜索周围目标函数减小的方向并向该方向移动一小段距离，经过多次搜索后到达极值点的附近。

一、一阶梯度算法

一阶梯度算法是在搜索时只使用一阶梯度提供的信息的算法。最基本的一阶梯度算法是梯度下降法。

**1、梯度下降法**

设目标函数（多元函数）为f(x)，其中x是自变量，它是一个向量，，目标函数的梯度为。梯度也是一个向量，它是由f(x)对x的每个分量的偏导数组成，。梯度方向是目标函数f(x)变化最快的方向，所以梯度下降法针对每一点，向该点的梯度方向搜索。有x更新的公式：



其中是梯度下降的步长，又称为学习率，也可以理解为梯度下降的速度。

**2、随机梯度下降法**

随机梯度下降法是针对机器学习中的目标函数的优化问题而提出的对梯度下降法的改进。

在机器学习模型中，目标函数都是基于训练集构建的。给定一个训练集，模型的形式为，其中是模型参数，则模型的目标函数可以写成如下形式：



它是关于模型参数的函数。其中L表示损失函数，它衡量了模型对样本的预测结果与样本的真实标签之间的误差。

如果直接使用梯度下降法，则参数的更新公式为：



可以看出，每次搜索新的点，都需要对整个训练集的样本计算梯度，这使得计算量非常之大。

随机梯度下降法的思想是，每次搜索新的点时，不使用训练集中全部样本，而是使用从训练集中随机抽取的一小部分样本。设第t次迭代随机选取了训练集中个样本（<m）：，则参数的更新公式变为：



随机梯度下降方法每次搜索时，并不是使参数严格地沿着梯度方向移动，而只是沿着与梯度方向相近的方向移动。这也能保证最终的收敛，而且还减少了计算量。

**3、动量梯度下降方法**

动量梯度下降法基于“动量”的概念，对梯度下降法进行改进，以加速收敛。

梯度下降法的一个缺点是，由于每次迭代的参数更新的步长固定，可能导致每次参数点的移动会越过该梯度下降方向的极小点，从而导致下一次参数点的移动会再往回移动，这样，整个迭代过程就形成一种“反复震荡”的形态，使得收敛很慢。（如图3中左图）

动量梯度下降法针对梯度下降法的这个缺点进行改进。梯度下降法每次参数点移动时，单纯地向当前点梯度下降方向移动，而不考虑上一次的梯度下降方向；动量梯度下降法则在每次参数点移动时，保留一部分上一次梯度下降的方向。其参数更新公式为：



可见，每次参数点移动的方向v包含两部分：一部分是当前梯度下降方向，一部分是上一次梯度下降方向。其中用来控制本次参数点移动方向中，上次梯度下降方向的成分的多少。

动量梯度下降法可以使每次参数点移动时，越过极小值的程度减小，从而加快收敛。（如图3中右图）

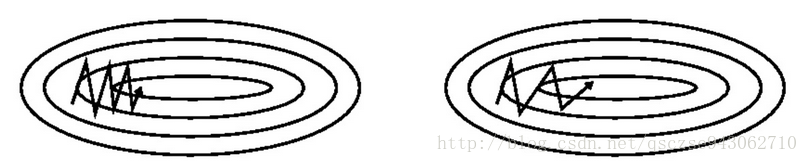


图3 梯度下降与动量梯度下降

**3、Nesterov动量梯度下降方法**

Nesterov动量梯度下降方法与动量梯度下降方法只有很小的区别。下面是Nesterov动量梯度下降方法的参数更新公式：



可以看出，Nesterov动量梯度下降方法在计算梯度时，使用的不是上一步更新后的参数，而是对临时更新成，再计算梯度，接着用这个梯度更新v，进一步更新。这样做的目的是进一步加速收敛。

二、自适应学习率算法

前面介绍了一阶梯度算法，包括简单梯度下降法和它的几个改进版本：为了减少计算量而提出的随机梯度下降法，以及为了加快收敛而提出的动量梯度下降和Nesterov动量梯度下降。

这些方法都使用固定的学习率，即每次参数点移动的步长是不变的。这就造成一些问题，比如上面所说的，参数点移动时很可能越过该梯度下降方向的极小点，导致不必要的移动。因此我们需要在迭代过程中能够自适应调整学习率的优化算法。

**1、AdaGrad**

AdaGrad的基本思想是：对参数的不同分量给以不同的学习率，各分量的学习率的大小依赖于目标函数在该分量上历史偏导数累积平方和的大小。

AdaGrad方法的参数更新公式如下：



其中g表示当前点的梯度，梯度的计算方式与梯度下降法中的计算方式相同；表示对向量逐元素求积；r是一个向量，显然，它表示梯度的累积平方；注意，式子结果也是个向量，它对r逐元素地求加法、求根和求除法。

可以看出，AdaGrad方法与梯度下降方法的区别仅在于每次迭代时的学习率不同。AdaGrad的学习率不是一个常数，而是对参数的每个分量施加不同的学习率。根据迭代历史统计出每个分量的历史上的偏导数的平方和，对偏导数平方和较大的分量，施以较小的学习率（以减小其越过极小点的可能性和程度）；对偏导数平方和较小的分量，施以较大的学习率。

注意，AdaGrad不仅对参数的不同分量施加不同大小的学习率，而且这些学习率是随着迭代次数增加而递减的（因为偏导数的累积平方和随着迭代次数增加而增加）。

**2、RMSProp**

RMSProp方法是在AdaGrad方法的基础上的改进。上面说到，AdaGrad的学习率会随迭代过程递减，这可能导致在最终收敛到局部极值之前，学习率就变得非常小，使得最终的收敛非常慢，甚至不收敛。

RMSProp通过改造梯度的累积平方r的计算，来解决这个问题。

RMSProp的参数更新公式为：



可以看出，RMSProp方法中，r的更新不再是累加梯度的平方，而是每次迭代时先去掉一部分r，然后再加入当前的梯度（相当于不断地用当前的梯度去稀释历史的梯度）。这就使历史梯度对学习率的影响不会累积，而当前梯度和较近的梯度对学习率的影响总是较大。

**3、Adam**

Adam算法可以看作是结合了动量算法和RMSProp两种算法。

Adam算法相比前面的几种自适应算法稍微复杂一些，这里写出算法每次迭代的所有操作（设当前迭代为第t次迭代）：

（1）计算梯度：

（2）进入下一轮迭代：

（3）更新有偏一阶矩的估计：

（4）更新有偏二阶矩的估计：

（5）修正一阶矩偏差：

（6）修正二阶矩偏差：

（7）更新参数：

可以看出，Adam算法中的相当于是动量的作用；相当于是RMSProp中的r的作用。算法将二者结合在一起。

之所以将和称为一阶矩估计和二阶矩估计，实际上相当于是将多次迭代的梯度g的每个分量看成是一个随机变量，对它们进行一阶矩估计和二阶矩估计（当然，并不是标准的一阶矩估计和二阶矩估计的概念，只是意思意思）。

三、二阶梯度算法

一阶梯度优化算法只使用当前参数点的一阶梯度信息决定参数点的移动方向，而二阶梯度算法还使用二阶梯度信息。

对于多元函数，有多个二阶偏导数，它们构成Hessian矩阵：



可以看出，Hessian矩阵的第i行第j列的元素是f(x)对自变量和的二阶偏导数。根据二阶偏导的对称性（）知，Hessian矩阵是对称矩阵。

**1、牛顿法**

牛顿法可以看作是梯度下降法的一种延伸。梯度下降法相当于在当前点x处用切平面s1(x)代替目标函数曲面f(x)，然后在切平面上找到移动微小距离后使s1(x)下降最多的方向，并向该方向移动微小距离；牛顿法则在当前点处用与目标函数曲面f(x)相切的二次曲面s2(x)代替目标函数曲面，然后在该二次曲面上找到

移动微小距离后使s2(x)下降最多的方向，并向该方向移动微小距离。

牛顿法每次迭代的参数点移动方向比梯度下降法更接近该参数点指向最终极值点的方向。

牛顿法的收敛速度为二阶收敛，梯度下降法则为一阶收敛。

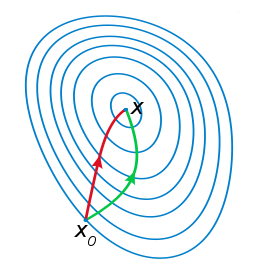


图4 梯度下降法与牛顿法的路径对比

如图所示，绿色为梯度下降法的参数点移动轨迹；红色为牛顿法的参数点移动轨迹。

设目标函数为，则牛顿法每次迭代的流程如下：

（1）计算当前梯度：

（2）计算当前Hessian矩阵：

（3）求Hessian矩阵的逆

（4）更新参数：

可以看出，牛顿法与梯度下降法的一个重大区别是，牛顿法的参数更新公式中没有步长因子（为什么不需要步长因子）。

另外，牛顿法每次迭代需要先计算梯度和Hessian矩阵，并对Hessian矩阵求逆，然后更新参数。

**2、拟牛顿法**

牛顿法虽然具有二阶收敛速度，比梯度下降法要快，但是它每次迭代需要对Hessian矩阵求逆，其计算量相当之大。那么有没有方法，不需要二阶梯度信息，只用一阶梯度信息，近似求取Hessian矩阵的逆，从而既保证收敛速度，又减少计算量呢？

这就是拟牛顿法的思想。

那么应当用什么矩阵来近似Hessian矩阵的逆呢？实际中不止一种近似方法。但它们都满足“拟牛顿条件”，拟牛顿条件是Hessian矩阵的逆近似满足的一个式子（注意，是近似满足，不是严格满足）。

由于Hessian矩阵代表了目标函数的二阶偏导，所以近似地有：



其中下标t和t+1分别表示第t次和第t+1次迭代的结果。该式称为割线方程。

故近似地有：



这就是拟牛顿条件。

（1）DFP算法

DFP算法用矩阵D来近似Hessian矩阵的逆，D的更新公式为：



其中，，即梯度增量，为目标函数的自变量的增量。

（2）BFGS算法

BFGS算法先用矩阵B来近似Hessian矩阵，因此类比DFP算法，得到B的更新公式为：



其中，和的意义与DFP算法的更新公式中的含义相同。可见，B的更新公式只是把DFP算法中D的更新公式中和对换了一下。

然后再利用Sherman-Morrison公式，得到Hessian矩阵的逆的近似G的更新公式：



（3）L-BFGS算法

BFGS算法需要存储矩阵G，G的阶数是模型参数的个数。当模型参数很多时，G会很大。为了减小存储量，提出了BFGS的近似算法：L-BFGS。

L-BFGS通过使用两个向量y和近似计算G，避免了对G的存储，而只需存储向量y和即可。

**3、共轭梯度法**

理论上，共轭梯度法也需要利用目标函数的二阶梯度信息，即Hessian矩阵，因此属于二阶梯度优化算法，但是实际中共轭梯度法可以避免计算Hessian矩阵，以减少计算量。

前面介绍梯度下降法和牛顿法，分别是最基本的一阶梯度优化方法和最基本的二阶梯度优化方法。它们的关键都是在每一次迭代中确定参数点移动的最优方向，只不过梯度下降法是基于目标函数曲面在当前参数点的切面确定最优方向，牛顿法是基于目标函数曲面在当前参数点的二次切曲面确定最优方向。牛顿法比梯度下降法具有更快的收敛速度。共轭梯度法的关键也是在每次迭代中确定参数点移动的最优方向，与牛顿法相同的是，它也基于二次切曲面；但与牛顿法不同的是，它不是直接找目标函数值局部下降最多的方向。

共轭梯度法的基本思想是，梯度下降法每一次迭代只考虑当前参数点处目标函数下降最快的方向，丝毫没有考虑之前迭代步中目标函数在其方向上下降的“成果”。因此可能“撤销”一部分之前迭代步中下降成果。

对于任何目标函数来说，给定初始参数点，在上帝视角下，我们可以只经过一次参数点移动，就到达目标函数极小点。当然，我们不是上帝，不可能那么牛逼，但是我们可以想办法接近上帝。

我们可以设想这样一个过程：将初始参数点到全局极小点的路径分解为多个前后相接的路径（本质上是向量的分解）。直观上，要想使路径数目最少，就应该使每个路径达到在该方向上（从全局来看的）极小，而且不影响之前已走过的路径在相应方向上取得的（从全局来看的）极小。显然，这种思路下最少的路径数为自变量空间的维数，也就是目标函数的自变量（即模型的参数）的个数。

遗憾的是，我们无法在任意形式的目标函数上实现这个思路，但是对于二次目标函数来说，很容易实现之。这就是共轭梯度法。共轭梯度法通过保证各个路径的方向保持“共轭”来实现这一思路。它能对任意的二次目标函数和任意的初始参数点，完美地实现上述思路。

每次迭代时，共轭梯度法计算与先前路径方向共轭的方向，并计算在该方向上的步长以便达到该方向上的（从全局上来看的）极小。

这里有三点要注意，一是我们谈到的“每个方向上取得极小”和“不影响之前路径的方向上取得的极小”中，所谓的“取得极小”，都是从全局（这里全局的意思，是针对各路径方向而言的，而不是针对非凸函数的多个局部极值而言的）来看的，而不是局部上来看的；二是只对于二次目标函数才能用共轭梯度法完美地实现上述思路。三是要保证每个路径方向不影响之前方向上的极小成果，就要每次计算共轭方向，而又要保证每次在当前方向上取得全局的极小，就还要计算步长。

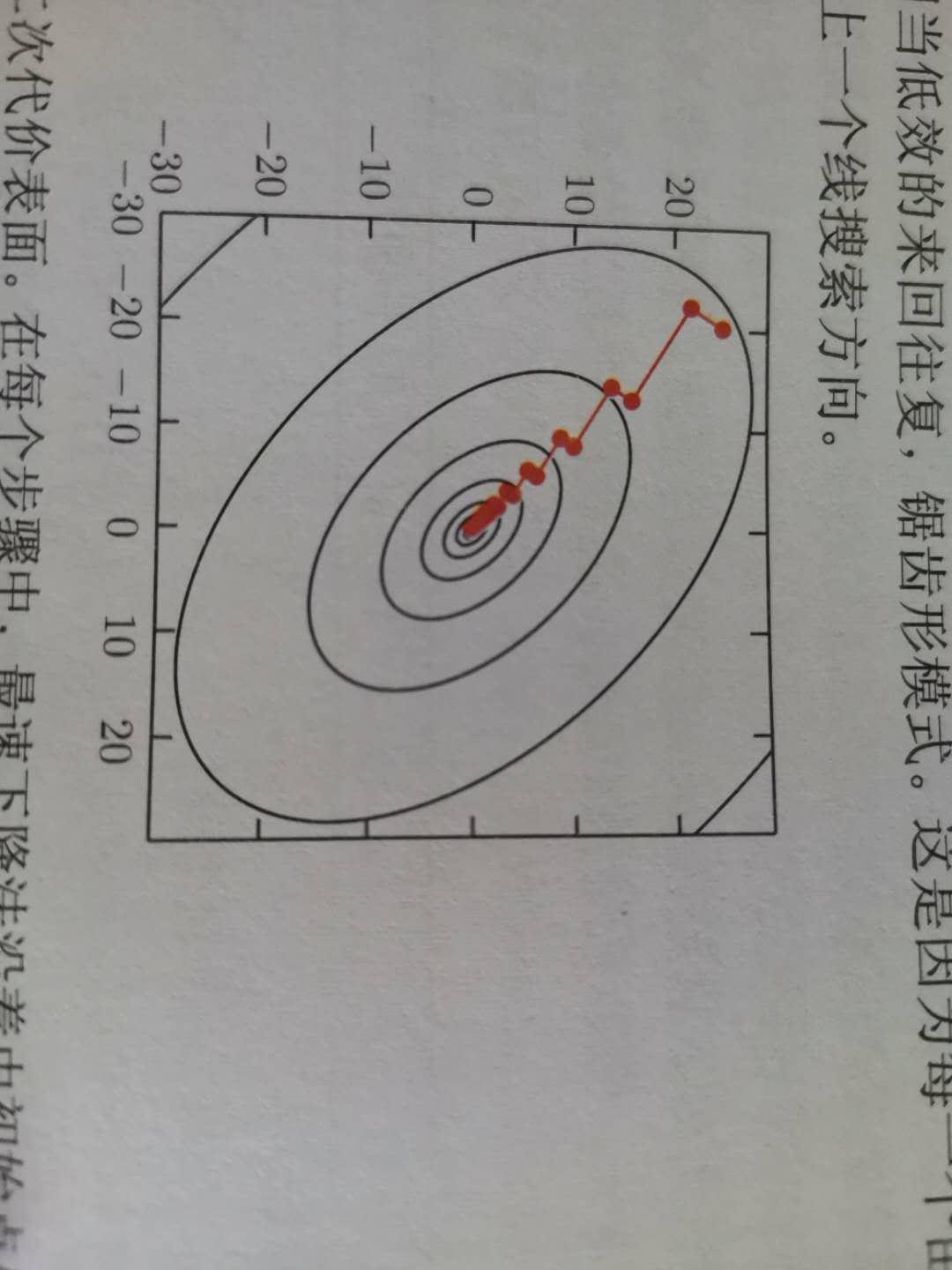


图5 二次目标函数的梯度下降法示意图

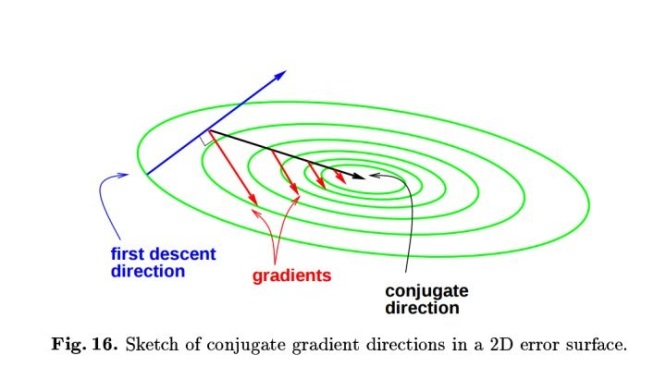


图6 二次目标函数的共轭梯度法示意图

如图5和图6为二元二次目标函数的梯度下降法和共轭梯度法的搜索路径。可以看出，从全局上看，梯度下降法每次迭代的搜索方向都会撤销一部分之前的搜索方向上的（全局来看的）极小化的成果；而共轭梯度法不仅不会撤销，而且保证每次能达到当前方向上的（全局来看的）极小，因此只需两次迭代即可达到全局极小。

以上分析过程是基于目标函数为二次函数的情况。对于任意的目标函数，可以使用共轭梯度法的扩展。此时不可能像二次目标函数那样有完美的方案，因为不再能从“全局”去考虑各方向的极小化了。此时只能在局部用二次切曲面代替目标函数曲面（这一点与牛顿法的思想类似），然后应用（局部意义的）共轭梯度法。这时也不能保证迭代n次（n是自变量的个数）就达到全局极小点。因此在迭代过程中要视情况重置方向。

下面介绍共轭梯度法的细节。首先要给出两个方向共轭的定义，对于二次函数，两个向量和共轭的定义是：



其中H是目标函数的Hessian矩阵，对于二次函数来说Hessian矩阵是个常数矩阵（在任意点处都相同）。

共轭梯度法要保证任何迭代步中的搜索方向都与之前所有方向共轭，则有：



其中，是第t次迭代的搜索方向，是第t-1次迭代的搜索方向，（注意，有多种计算方式）。

注意，直接根据共轭的定义计算是需要对Hessian矩阵求特征向量的，但是实际上可以避免之。

则共轭梯度法的完整流程如下（第t步迭代）：

（1）计算当前梯度：

（2）计算

（3）计算搜索方向：

（4）计算该方向上的步长：

（使改方向上目标函数达到极小）

（5）更新参数：

可以看到，共轭梯度法在每次迭代时，不仅要求出共轭方向，还要求出在该共轭方向上移动的步长。

注意，共轭梯度法与牛顿法都是基于目标函数曲面的二次切曲面近似（因此都需要用到二阶梯度信息），对于目标函数为二次函数的情况，牛顿法只需一次迭代即可达到极小，而共轭梯度法需要n次（n是目标函数的自变量个数，也就是模型的参数个数），但是牛顿法和共轭梯度法的思路是不同的，因此对于非二次函数的目标函数，无法断定牛顿法的性能仍然高于共轭梯度法。

Part 2、启发式数值优化算法

（待写）