

非精确一维搜索优化算法

吴颖馨¹

Abstract

本次实验从无约束问题出发，实现了非精确一维搜索下步长选择的 Goldstein 方法以及 Wolfe-Powell 方法，以及方向选择的最速下降和牛顿法。分别采用了 Rosenbrock 函数以及二次函数族，以 Scipy 模块中的共轭梯度法为 Baseline，对优化结果进行了定性、定量的全方面分析。结果表明了(1)本次实验所实现的方法的正确性以及有效性；(2)牛顿方向相对于最速下降方向在二次函数上优化的优势；(3)Wolfe-Powell 步长准则相对于 Goldstein 方法的结果差异。

Keywords

无约束最优化，非精确一维搜索，Goldstein，Wolfe-Powell，最速下降方向，牛顿方向

¹Department of Big Data, USTC. wuyxin@mail.ustc.edu.cn

under guidance of PROF. 杨周旺, TAs: 吴凌霄, 王朔

Contents

1 算法总结	1
1.1 一维搜索算法框架	1
1.2 搜索方向	2
1.3 步长因子	2
2 实验设置	2
2.1 Baseline	2
2.2 测试函数	2
2.3 算法微调	3
3 实验结果及分析	3
3.1 求梯度的代码技巧	3
3.2 定量分析	4
3.3 定性分析	5
4 程序指南	6

无约束最优化问题由一个初始点 X_0 出发，通过局部地拓展，得到序列 $\{X_k | k = 1, \dots\}$ ，从而最终收敛到局部极小点乃至全局极小点。一维搜索将这样的拓展分解为两步：(1) 寻求下降方向 d_k ；(2) 基于 d_k 求解步长 α_k ，使得 $f(X_k + \alpha_k d_k)$ 相对于 $f(X_k)$ 有一定程度上的下降。

算法 1 Linear Search Method

输入: target function f , initial point X_0 , allowable error ε , maximum epoches E , and some other parameters

输出: final (locally optimal) point X_k

$k := 0$

while $\|g_k\|_2^2 < \varepsilon$ and $k < D$ **do**

 compute the search direction d_k s.t. $g_k^T d_k < 0$

 determine the step length factor α_k

 s.t. $f(X_k + \alpha_k d_k) \leq f(X_k)$

$X_{k+1} = X_k + \alpha_k d_k$

end while

不同搜索方向策略与不同的步长因子的选择构成了不同的一维搜索算法。其中精确一维搜索通过在求解步长因子 α_k 时采用策略

$$\alpha_k = \arg \min_{\alpha \geq 0} \varphi(\alpha) = f(x_k + \alpha d_k) \quad (2)$$

据文献表明，一般而言，精确一维搜索的算法较非精确一维搜索算法要更差。一方面，精确一维搜索计算量较

1. 算法总结

以下简要总结实验的理论基础。

1.1 一维搜索算法框架

对于问题

$$\min_X f(X) \quad (1)$$

大。另一方面，精确一维搜索可能使得算法的局部性更高，从总体来看不一定使得算法的收敛性更快。

1.2 搜索方向

- 最速下降方向 $d_k = -g_k$, 保证了在 X_k 点附近, 有 $d_k^T g_k = -\|g_k\|_2^2 < 0$, 由 Cauchy-Schwarz 知该方向为使得 f 下降最快的方向。
- Newton 方向 $d_k = -G^{-1}d_k$ 为在 X_k 附近采用二次模型来近似并求其极小值的方向, 其在最优解附近能够快速收敛。但由于求解逆矩阵 G^{-1} 存在一系列数值问题, 故常用近似牛顿方向或拟牛顿法来求解。
- 共轭梯度方向基于共轭方向基本定理, 对于二次函数能够保证在小于维度空间 n 次迭代下算法终止, 与牛顿法同样, 在最优解的局部具有快速收敛性。其实质是将 G 变换到一个新的坐标系, 而在该坐标系中变量相互分离。

本次实验实现了最速下降方向以及 Newton 方向两种搜索方向。

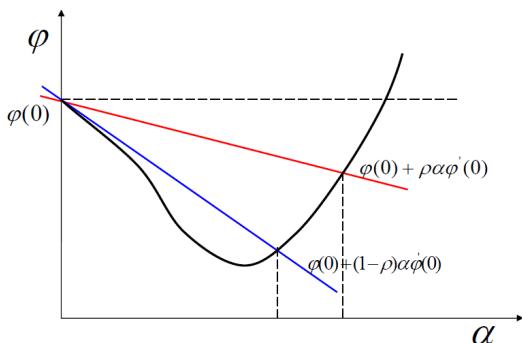
1.3 步长因子

对于非精确一维搜索, 代表性的算法为 Goldstein(1965) 与 Wolfe, Powell 在 1976 年提出的两个准则。

- Goldstein 准则:

$$\varphi(\alpha) \leq \varphi(0) + \rho \alpha \varphi'(0) \quad (3)$$

$$\varphi(\alpha) \geq \varphi(0) + (1 - \rho) \alpha \varphi'(0) \quad (4)$$



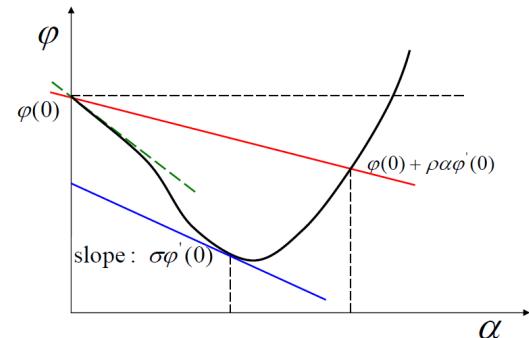
Goldstein 准则的思路很淳朴, 即希望在有限区域 $\{X | f(X) < f(X_k)\}$ 中, 排除掉所有连续区间的左

右端点及其邻域(也是最不可能为最优解的部分)。

- 由于 Goldstein 准则存在排除最优点的可能性, Wolfe & Powell(1976) 对其进一步修改, 将 Goldstein 中的第二式子替换为如下

$$\varphi'(\alpha) \geq \sigma \varphi'(0) \quad (5)$$

可以证明, 总是存在 α 使得上述方程成立。



本次实验实现了 Goldstein 准则与 Wolfe-Powell 准则, 并在实验结果部分进行比较。

2. 实验设置

2.1 Baseline

为了对优化的结果作出验证与比较, 本次实验采用 Python 中 scipy 的优化包中的共轭梯度法最小化作为 baseline, 具体代码如下:

```

1 import scipy.optimize as opt
2 import numpy as np
3
4 L = 100
5 fun = lambda x: f(x)
6 bnds = [(-L, L) for i in range(dim)]
7 res = opt.minimize(fun=fun,
8                     x0=np.zeros(dim),
9                     bounds=bnds)

```

2.2 测试函数

本次实验采取了两个测试函数, 分别为著名的测试函数 Rosenbrock function:

$$f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 \quad (6)$$

正定二次函数：

$$f(X) = \frac{1}{2} X^T G X + c^T X + b \quad (7)$$

在实验中不同的 G 与 c 构成了一组测试函数，为了保证 G 的非负定性，采用先生成一个大小为 $\mathbb{R}^{n \times n}$ ，每个元素为 0 到 1 之间的随机矩阵 \mathcal{A} ，之后取 $G = \mathcal{A}^T \mathcal{A}$ 即可保证非负定性。

2.3 算法微调

为了在实际中得到更好的效果，对算法 1 采取以下几个方面的调整：

- 对于 Goldstein 方法，可以将其看做其为在水平集上对 α_k 的搜索，故实际中采用类似于二分法的搜索方式，即对于 α_k 设置上限界分别为 α_1, α_2 。若条件 1 即式 (3) 不符合，表明目前的 α_k 偏右，因此降低其上界 α_2 ；若条件 2 即式 (4) 不符合，表明目前的 α_k 偏左，因此降低其下界 α_1 。

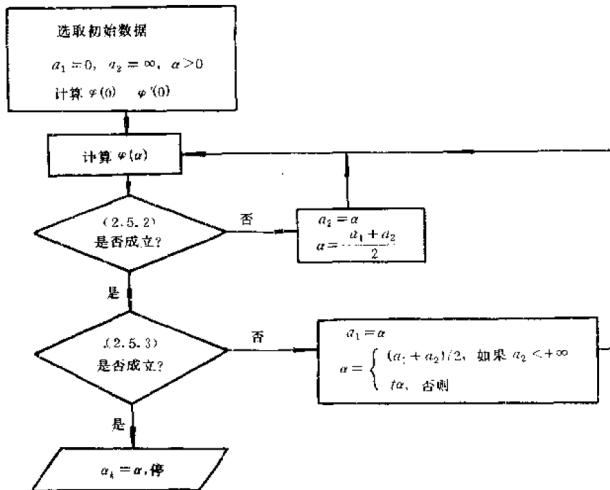


Figure 1. 参考文献[2]中的 Goldstein 算法框图

- 对于 Wolfe-Powell 方法，在条件不满足时则采用插值的办法作为这一步 α_k 的近似。

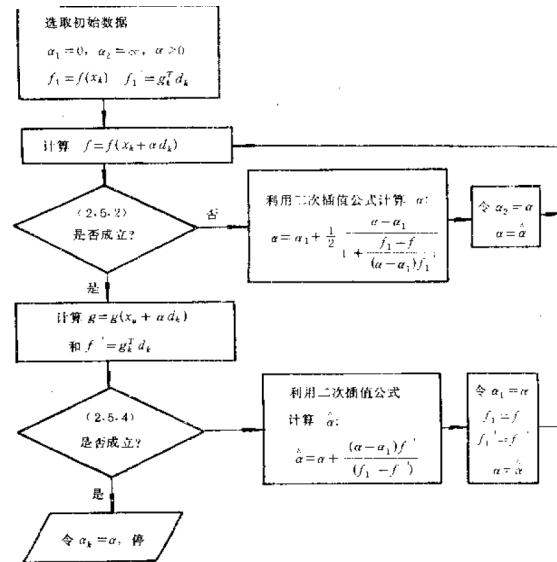


Figure 2. 参考文献[2]中的 Wolfe-Powell 算法框图

- 实际操作时，发现 Wolfe-Powell 方法中存在 $\hat{\alpha}$ 为 nan 的情况，为分母近似为 0 造成的。因此需要在上面的程序框图中加入对 $\hat{\alpha}$ 是否为 nan 的判断，若在第一个判断中出现 nan 的情况，则令 $\alpha = \alpha_1$ ，而 α_1 保持不变；若在第二个判断中出现 nan 的情况，则同理令 $\alpha = \alpha_2$ ，而 α_2 保持不变。
- 迭代求 α_k 的停机准则：为了综合考虑代码效率与结果的精确度，设置计算 α_k 的最大迭代次数 E ，则每次计算 α_k 的停机条件为
 $k >= E$ or conditions are satisfied

3. 实验结果及分析

实验中，取算法一中的 ϵ 值为 10^{-4} ，迭代求 α_k 的最大迭代次数 $E = 100$ ，求 X_k 的最大迭代次数 $D = 300$ ， $\rho = 0.2, \sigma = 0.8$ 。

3.1 求梯度的代码技巧

本次实验设计到对 $f: X \rightarrow v \in \mathbb{R}$ 求梯度，代码技巧是利用 Pytorch 的 autograd 模块，将 X 作为 Variable 加入计算图中，将计算图的结果进行 backward， $X.grad$ 即可获取所需梯度。

```

1 def gradient(f, X):
2     X = Variable(X, requires_grad=True)
3     out = f(X)
4     out.backward()
5     return X.grad

```

3.2 定量分析

(1) 我们先考虑测试函数为 Rosenbrock function 时的运行效果 (此时 Newton 方法由于存矩阵求逆的数值问题, 因此无法给出解, 这也反映了牛顿法的局限性):

```
-----Goldstein + Steepest Descent-----
result: tensor([1.0001, 1.0001])
gradient: 0.00000
fun: 0.00000
time: 0.66
-----Wolfe Powell + Steepest Descent-----
result: tensor([1.0006, 1.0012], grad_fn=<AddBackward0>)
gradient: 0.00000
fun: 0.00000
time: 1.24
-----Spicy Test-----
Optimization terminated successfully.
    Current function value: 0.000000
    Iterations: 21
    Function evaluations: 220
    Gradient evaluations: 55
[0.99999552 0.99999102]
```

Figure 3. 采用Rosenbrock function的实验结果

可见两种非精确一维搜索的方法都能够得到很好的结果, 与最优解 $[1.0, 1.0]$ 之间的差距很小, 且 Wolfe-Powell 的策略比 Goldstein 的策略用时更久一些, 主要原因是求梯度以及插值带来的计算量。

(2)对于函数 $f(X) = \frac{1}{2}X^T GX + c^T X + b$, 以下为在 $G = \begin{bmatrix} 0.973 & 0.547 \\ 0.547 & 0.527 \end{bmatrix}$, $c = [0.275, 0.159]$, $b = 0$ 下的运行结果:

```
-----Goldstein + Steepest Descent-----
result: tensor([-0.2538, -0.0519])
gradient: 0.00006
fun: -0.03873
time: 0.12
-----Goldstein + Newton-----
result: tensor([-0.2751, -0.0188])
gradient: 0.00001
fun: -0.03885
time: 0.01
-----Wolfe Powell + Steepest Descent-----
result: tensor([-0.2639, -0.0181], grad_fn=<AddBackward0>)
gradient: 0.00009
fun: -0.03882
time: 0.00
-----Wolfe Powell + Newton-----
result: tensor([-0.2719, -0.0186], grad_fn=<AddBackward0>)
gradient: 0.00000
fun: -0.03885
time: 0.00
-----Spicy Test-----
Optimization terminated successfully.
    Current function value: -0.033459
    Iterations: 1
    Function evaluations: 8
    Gradient evaluations: 2
[-0.27488953 -0.15853733]
```

Figure 4. 采用二元二次函数的实验结果

可以看到本次实现的方法与 Scipy 包都基本上找到了全局极小值, 其中 Goldstein + Newton 与 Wolfe-Powell + Newton 得到的极小值为 -0.03885, 为所有方法中最小的结果, 猜想为 Newton 法的快速收敛性在此处起到了效果。

为了验证这种猜想, 我们来考虑一个更高阶的二次函数, 其 Hesse 阵、线性系数以及实验的结果如下所示

```
G: tensor([[3.3700, 1.9223, 1.8095, 2.3672, 3.0315, 2.3587, 3.6224, 3.0782],
           [1.9223, 1.5561, 1.3197, 1.5108, 1.6839, 1.4833, 2.0327, 1.9816],
           [1.8095, 1.3197, 1.4466, 1.5742, 1.7748, 1.4775, 1.7107, 1.9662],
           [2.3672, 1.5108, 1.5742, 3.0430, 1.7418, 1.6568, 2.6449, 2.4653],
           [3.0315, 1.6839, 1.7748, 1.7418, 3.2812, 2.1795, 3.1762, 2.8928],
           [2.3587, 1.4833, 1.4775, 1.6568, 2.1795, 2.2344, 2.1687, 2.5648],
           [3.6224, 2.0327, 1.7107, 2.6449, 3.1762, 2.1687, 4.4389, 3.1325],
           [3.0782, 1.9816, 1.9662, 2.4653, 2.8928, 2.5648, 3.1325, 3.274611]])
c: tensor([0.1668, 0.7344, 0.6702, 0.4440, 0.0520, 0.6648, 0.9726, 0.8527])
-----Goldstein + Steepest Descent-----
result: tensor([ 6.3627,  5.0684, -9.6607,  3.8329,  5.4364, -2.4223, -7.5095, -2.1137])
gradient: 0.00010
fun: -5.18745
time: 0.74
-----Goldstein + Newton-----
result: tensor([ 5.6236,  5.4733, -9.8181,  4.3044,  6.0919, -1.3703, -7.2835, -3.5456])
gradient: 0.00001
fun: -5.19563
time: 0.01
-----Wolfe Powell + Steepest Descent-----
result: tensor([ 6.4122,  4.9018, -9.4608,  3.7043,  5.2540, -2.5344, -7.4333, -1.9062],
             grad_fn=<AddBackward0>)
gradient: 0.00010
fun: -5.18468
time: 1.80
-----Wolfe Powell + Newton-----
result: tensor([ 5.6124,  5.4619, -9.7979,  4.2954,  6.0791, -1.3680, -7.2687, -3.5377],
             grad_fn=<AddBackward0>)
gradient: 0.00000
fun: -5.19565
time: 0.00
```

Figure 5. 采用八元二次函数的实验结果

从时间上看，Newton 法给出的方向比梯度方向效果要更好！其中 Wolfe-Powell + Steepest Descent 所花的时间为 1.8 s，而 Wolfe-Powell + Newton 只用了不足 0.01 s。在八元二次函数下，Scipy 的最终优化结果为 -1.126998，未达到全局最优解。

3.3 定性分析

(1) 以下可视化测试函数为 Rosenbrock function 时的优化过程：

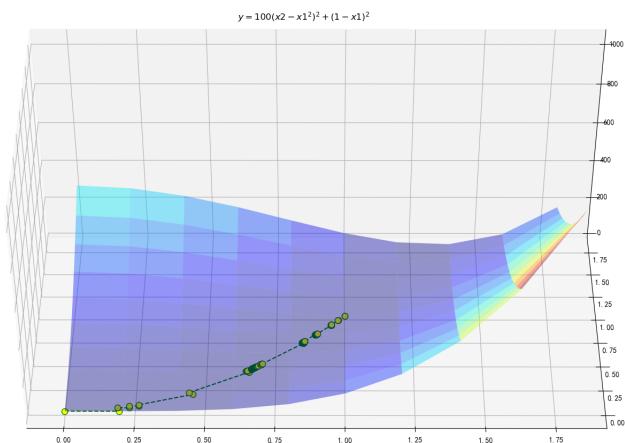


Figure 6. Goldstein + Steepest Descent

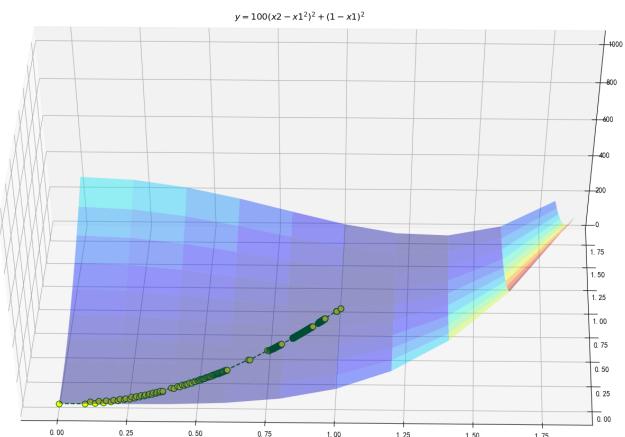


Figure 7. Wolfe-Powell + Steepest Descent

可以看到，由于 Wolfe-Powell 比起 Goldstein 的下降要更为保守，迭代的步长更小，迭代次数更高。

(1) 接下来看一个很有意思的例子，对于二次函数， $G = \begin{bmatrix} 1.699 & 1.201 \\ 1.201 & 0.955 \end{bmatrix}$ ， $c = [0.009, 0.948]$ ， $b = 0$ 时，将四个方法的迭代点过程可视化如下：

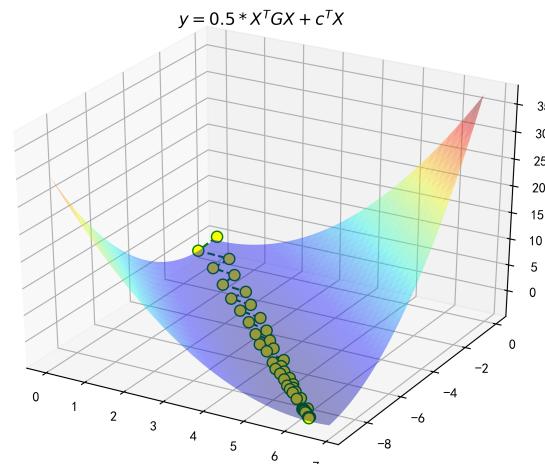


Figure 8. Goldstein + Steepest Descent

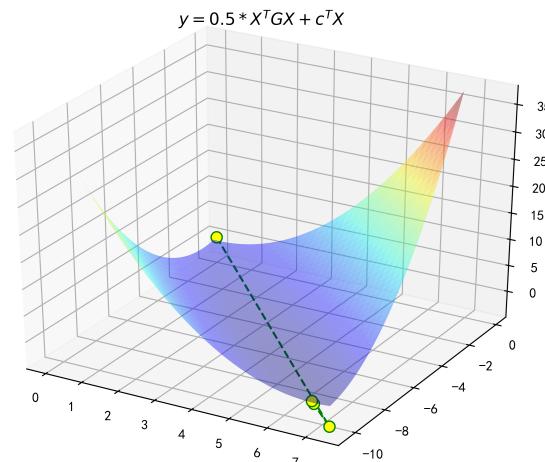


Figure 9. Goldstein + Newton

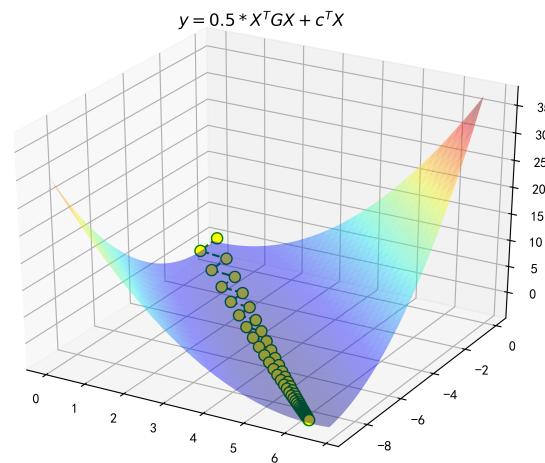


Figure 10. Wolfe-Powell + Steepest Descent

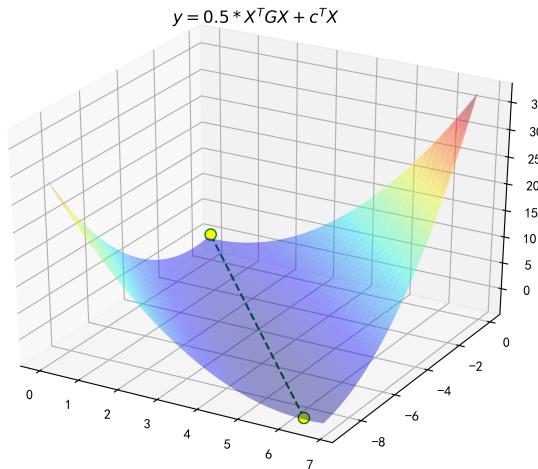


Figure 11. Wolfe-Powell + Newton

可以看到，在模型本身为二次模型时，Newton 法的方向效果大大好于最速下降法的方向，在Wolfe-Powell + Newton 中，甚至一步就能达到最优解。同时，从细节处对比 Goldstein + Steepest Descent(图8) 与 Wolfe-Powell + Steepest Descent(图9) 也可以看到，在非精确一维搜索下，Wolfe-Powell 产生的点列要更为规整，而 Goldstein 产生的点列则参差不齐，体现了在更严格的约束下， X_{k+1} 的选取对于整个路径的影响。在某些场景中，这也许会影响到优化的全局最优性。

4. 程序指南

文件	用途
main.py	实现输入输出，函数调用
target_function/convex.py	定义凸二次函数
target_function/rosenbrock.py	实现 rosenbrock 函数
direc/newton.py	计算牛顿方向
direc/steepest_descent.py	计算负梯度方向
step/goldstein.py	计算 goldstein 步长
step/wolfe_powell.py	计算 wolfe-powell 步长

命令行运行 “python main.py” 即可。

参考文献

[1] Thomas V. Mikosch Sidney I. Resnick Stephen M. Robinson. Springer Series in Operations Research and Financial Engineering

[2] 袁亚湘, 孙文瑜, 最优化理论与方法 (1997版)