基于非精确一维搜索的迭代下降算法

王禹 PB18000145

2021 年 1 月 14 日

1 问题描述

在无约束最优化问题当中, 选定下降方向之后, 需要选定步长. 但精确一维搜索的代价太大, 甚至不可能在有限次计算中求出精确的最优步长. 因此实际操作中, 在得到有足够精度的近似解时, 就用来作为步长. 此时称为非精确一维搜索 (Inexact Line Search). 与精确一维搜索相比，在很多情况下采用非精确一维搜索可以提高整体计算效率.

2 算法原理

这里实现了两种非精确一维搜索的方式, 分别依据 Wolfe-Powell 准则和 Goldstein 准则. 我们将分别阐述其算法.

2.0.1 基于 Goldstein 准则的非精确一维搜索

首先给出 Goldstein 条件 (如图1):

$$\varphi(\alpha) \le \varphi(0) + \rho\alpha\varphi'(0) \tag{1}$$

$$\varphi(\alpha) \ge \varphi(0) + (1-\rho)\alpha\varphi'(0) \tag{2}$$

其中 $\rho \in (0, 1/2)$ 是一个固定参数. 在后面的代码实现中, ρ 取为 0.25. 由此, 给出基于 Goldstein 准则的非精确一维搜索算法:

1. 选取初始数据: $a = 0, b = \overline{\alpha}, \alpha \in (0, \overline{\alpha}), \rho = 0.25, t = 1.75$. 计算出 $\varphi(0), \varphi'(0)$.
2. 计算 $\varphi(\alpha)$. 判断式 (1) 是否成立, 如果成立, 进入下一步, 否则, 令 $b = \alpha$, 进入步 4.
3. 判断式 (2) 是否成立, 如果成立, 则返回当前 α, 如果不成立, 则判断 b 是否小于 $\overline{\alpha}$, 如果小于, 则进入第 4 步, 否则令 $\alpha = t * \alpha$, 返回第 2 步.

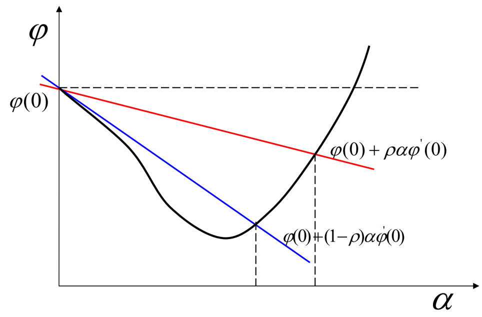

图 1: Goldstein Condition

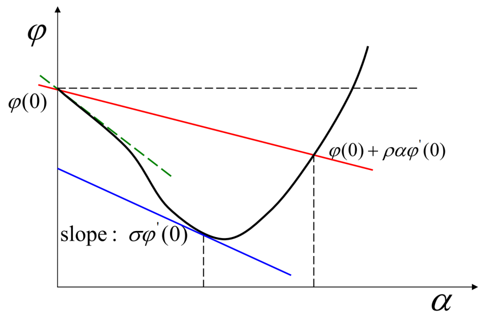

图 2: Wolfe-Powell Condition

4. $\alpha = (a + b)/2$, 返回第 2 步.

2.0.2 基于 Wolfe-Powell 准则

给出 Wolfe-Powell 条件 (如图2):

$$\varphi(\alpha) \le \varphi(0) + \rho\alpha\varphi'(0)$$

$$\varphi'(\alpha) \ge \sigma\varphi'(0)$$

其中 $\rho \in (0, 1/2), \sigma \in (\rho, 1)$.

针对该准则, 可以得到如下算法:

1. 给定初始一维搜索区间 $[0, \overline{\alpha}]$, 以及 $\rho \in (0, 1/2), \sigma \in (\rho, 1)$. 计算出 $\varphi(0), \varphi'(0)$. 令 $a = 0, b = \overline{\alpha}, \alpha \in (0, \overline{\alpha}), \varphi_1 = \varphi_0, \varphi_1' = \varphi_0'$.

表 1: 程序文件介绍

程序名	功能
main.py	定义函数以及梯度函数, 并调用其它程序进行梯度下降搜索.
line_search.py	其中包含 GoldStein 和 Wolfe-Powell 两种非精确一维搜索方法.
gradient_descent.py	其中包含两种梯度计算方法 (Steepest, Newton).

2. 计算 $\varphi=\varphi(\alpha)=f(x^{(k)}+\alpha d^{(k)})$. 若 $\varphi(\alpha)\leq\varphi(0)+\rho\alpha\varphi'(0)$, 则转到第 3 步. 否则, 由 $\varphi_1,\varphi_1',\varphi$ 构造两点二次插值多项式 $p^{(1)}(t)$, 并得其极小点:
$$\hat{\alpha}=a_1+\frac{1}{2}\frac{(a_1-\alpha)^2\varphi_1'}{(\varphi_1-\varphi)-(a_1-\alpha)\varphi_1'}$$
于是置 $a_1=\alpha,\alpha=\hat{\alpha}$, 重复这一步.
3. 计算 $\varphi'=\varphi'(\alpha)=\nabla f(x^{(k)}+\alpha d^{(k)})^T d^{(k)}$. 若 $\varphi'(\alpha)\geq\sigma\varphi'(0)$, 则输出 $\alpha_k=\alpha$, 并停止搜索. 否则, 由 $\varphi,\varphi',\varphi_1'$ 构造两点二次插值多项式 $p^{(2)}(t)$, 并得其极小点:
$$\hat{\alpha}=\alpha-\frac{(a_1-\alpha)\varphi'}{\varphi_1'-\varphi'}$$
于是置 $a_1=\alpha,\alpha=\hat{\alpha},\varphi_1=\varphi,\varphi_1'=\varphi'$, 返回 1.

2.1 算法实现

在编程实践中, 使用了最速下降 (Steepest descent) 和牛顿下降 (Newton descent) 两种下降方式, 分别与上面两种非精确一维搜索进行组合. 使用的测试函数为:
$$f(x)=\frac{1}{2}\mathbf{x}^T\mathbf{G}\mathbf{x}+\mathbf{c}^t\mathbf{x}+b \tag{3}$$
其中 $\mathbf{c},\mathbf{x}\in\mathbb{R}^d$, $\mathbf{G}\in\mathbb{R}^{d\times d}$ 为正定阵, $b\in\mathbb{R}$.

3 程序指南

程序文件组成如表1所示. 只需在命令行中输入 `python main.py` 即可. 其中 main 文件中包含两个可选择的参数: seed 和 dim. 其中 seed 为随机种子, dim 为测试函数的维数, 也即式 (3) 中 $\mathbf{x}$ 的维数.

4 程序测试

4.1 总体性能评估

在给定一个随机种子的情况下, 我测试了在测试函数维数不同的情况下的各种方法的性能. 汇总到表格 2,3,4,5 中. 从表格中可以得到如下结论:

表 2: dim = 2, $f(x_0)=0.78472247$

Direction	Line-search	Iteration	Result	Time(s)
Steepest	Wolfe-Powell	35	-1.52139011	111.44482
Steepest	Goldstein	35	-1.52139011	121.71970
Newton	Wolfe-Powell	1	-1.52141726	0.31020
Newton	Goldstein	1	-1.52141726	0.35659

表 3: dim = 5, $f(x_0)=11.83950494$

Direction	Line-search	Iteration	Result	Time(s)
Steepest	Wolfe-Powell	517	-0.25446543	56.13282
Steepest	Goldstein	517	-0.25446543	60.10588
Newton	Wolfe-Powell	2	-0.25447653	0.65440
Newton	Goldstein	2	-0.25447653	0.78093

表 4: dim = 8, $f(x_0)=36.31404550$

Direction	Line-search	Iteration	Result	Time(s)
Steepest	Wolfe-Powell	7651	-2.08442358	150.61806
Steepest	Goldstein	7651	-2.08442358	158.96693
Newton	Wolfe-Powell	2	-2.08444315	0.51064
Newton	Goldstein	2	-2.08444315	0.50365

表 5: dim = 10, $f(x_0)=49.02904982$

Direction	Line-search	Iteration	Result	Time(s)
Steepest	Wolfe-Powell	227329	-68.13945032	2218.96683
Steepest	Goldstein	227329	-68.13945032	2668.69832
Newton	Wolfe-Powell	2	-68.13977449	0.58218
Newton	Goldstein	2	-68.13977449	0.55789

1. 无论用最速下降方向还是用牛顿迭代方向, 得到的结果几乎都是相同的, 所不同的只有时间. 从上面几个例子中看出来, 其实两种非精确以为搜索的时间相差不大, Wolfe-Powell 稍微快一点.
2. 比较最速下降方向和牛顿方向, 可以看到, 随着维数增长, 牛顿法的步数一直稳定在个位数, 随着 x 的维数增长, 迭代次数的差距也越来越大.

表 6: dim = 2, $f(x_0) = 0.78472247$

Direction	Line-search	Iteration	Result	Time(s)
Steepest	Wolfe-Powell	25	-1.52139998	0.05945
Steepest	Goldstein	333	-1.52139523	0.41189
Newton	Wolfe-Powell	1	-1.52141783	0.00199
Newton	Goldstein	5	-1.52141726	0.00598

表 7: dim = 5, $f(x_0) = 11.83950494$

Direction	Line-search	Iteration	Result	Time(s)
Steepest	Wolfe-Powell	219	-0.25445810	0.24937
Steepest	Goldstein	200	-0.25446001	0.24933
Newton	Wolfe-Powell	1	-0.25447653	0.00096
Newton	Goldstein	7	-0.25447652	0.01102

表 8: dim = 8, $f(x_0) = 36.31404550$

Direction	Line-search	Iteration	Result	Time(s)
Steepest	Wolfe-Powell	7542	-2.08442183	8.07380
Steepest	Goldstein	6744	-2.08441635	11.34400
Newton	Wolfe-Powell	1	-2.08444315	0.00303
Newton	Goldstein	8	-2.08444315	0.01396

表 9: dim = 10, $f(x_0) = 49.02904982$

Direction	Line-search	Iteration	Result	Time(s)
Steepest	Wolfe-Powell	225036	-68.13943257	265.65643
Steepest	Goldstein	170909	-68.13927772	315.69130
Newton	Wolfe-Powell	1	-68.13977449	0.00100
Newton	Goldstein	9	-68.13977448	0.01396

4.2 细节比较

对于这两种非精确一维搜索, 其中有一个 $\overline{\alpha}$ 是可以进行修改的. 一种方式是直接令 $\overline{\alpha}$ 为一个较大的值, 例如 100; 另一种方式是令 $\overline{\alpha}$ 从零开始, 每次加一个比较小的值, 直到 $f(x+\overline{\alpha}d) > f(x)$ 为止. 具体过程为:

```
// step = 0.001
alpha_bar = step
while(f(x + alpha_bar * d) < f(x)):
    alpha_bar += step
```

在上面的实现当中, 使用的是后面这种方式. 为了探究修改 alpha 对两种非精确一维搜索的影响, 我又进行了下面的一些实验.

将两种非精确一维搜索都改为直接令 $\overline{\alpha} = 100$. 得到的结果如表格 6,7,8,9 所示. 从这些表格中可以发现, 这样的实现过程少了很多计算 $\overline{\alpha}$ 的过程, 因此计算速度快了很多. 比较结果的值, 也几乎一致.

5 结论

在这次实验中, 我尝试使用了两种梯度下降方式, 结合两种非精确一维搜索方式进行求解了最优化问题, 使用不同维度的二次函数进行试验并比较了结果. 可见 Wolfe-Powell 和 GoldStein 两种非精确一维搜索的有效性.