基于 Wolfe-Powell 准则非精确一维步长搜索的拟牛顿法

张景浩 PB20010399

January 2023

1 问题描述

无约束问题

$$\min_x \ f(x)$$

2 算法原理

（本算法参考自课程 ppt 中《无约束优化》一节）

拟牛顿法的一般迭代格式

(0) 初始化：选取适当的初始点 $x^{(0)}$，令 $H_0 = I$，k=0
(1) 计算搜索方向：$d^{(k)} = -H_k \nabla f(x^{(k)})$
(2) 确定步长因子：采用基于 Wolfe-Powell 准则的非精确一维搜索确定步长因子 α_k
(3) 更新迭代点：$x^{(k+1)} = x^{(k)} + \alpha_k d^{(k)}$
(4) 基于 $x^{(k)}$ 到 $x^{(k+1)}$ 的梯度变化，更新 Hesse 矩阵逆的近似，即确定满足正割条件的 H_{k+1}。令 $k := k+1$，返回 (1)
这里更新 Hesse 矩阵逆的近似我们选择关于 H_k 的 BFGS 校正，即

$$H_{k+1}^{(BFGS)} = H_k + (1 + \frac{y^{(k)^T} H_k y^{(k)}}{s^{(k)^T} y^{(k)}}) \frac{s^{(k)} s^{(k)^T}}{s^{(k)^T} y^{(k)}} - \frac{H_k y^{(k)} s^{(k)^T} + s^{(k)} y^{(k)^T} H_k}{s^{(k)^T} y^{(k)}}$$

基于 Wolfe-Powell 准则的非精确一维搜索算法

(0) 给定初始的一维搜索区间 $[0, \bar{\alpha}]$，以及 $\rho \in (0, \frac{1}{2})$，$\sigma \in (\rho, 1)$。计算 $\varphi_0 = \varphi(0) = f(x^{(k)}$，$\varphi_0' = \varphi'(0) = \nabla f(x^{(k)})^T d^{(k)}$。并令 $a_1 = 0$，$a_2 = \bar{\alpha}$，$\varphi_1 = \varphi_0$，$\varphi_1' = \varphi_0'$。选取适当的 $\alpha \in (a_1, a_2)$
(1) 计算 $\varphi = \varphi(\alpha) = f(x^{(k)} + \alpha d^{(k)})$，若 $\varphi(\alpha) \le \varphi(0) + \rho\alpha\varphi'(0)$，则转 (2)；否则，由 φ_1，φ_1'，φ 构造二次插值多项式，并得其极小值点 $\hat{\alpha}$。令 $a_2 = \alpha$，$\alpha = \hat{\alpha}$，转 (1)
(2) 计算 $\varphi' = \varphi'(\alpha) = \nabla f(x^{(k)} + \alpha d^{(k)})^T d^{(k)}$，若 $\varphi'(\alpha) \ge \sigma\varphi'(0)$，则输出 $\alpha_k = \alpha$，并停止搜索；否则，由 φ，φ'，φ_1' 构造二次插值多项式，并得其极小值点 $\hat{\alpha}$。令 $a_1 = \alpha$，$\alpha = \hat{\alpha}$，转 (1)

Theme 2.1. （BFGS 的全局收敛性）假设初始矩阵 B^0 是对称正定矩阵，目标函数 f(x) 是二阶连续可微函数，且下水平集

$$\mathscr{L} = \{x \in \mathbb{R}^n | f(x) \leq f(x^0)\}$$

是凸的，并且存在正数 m 以及 M 使得对于任意的 $z \in \mathbb{R}^n$ 以及任意的 $x \in \mathscr{L}$ 有

$$m \parallel z \parallel^2 \leq z^T \nabla^2 f(x) z \leq M \parallel z \parallel^2,$$

则采用 BFGS 格式并结合 Wlofe 线搜索的拟牛顿法全局收敛到 f(x) 的极小值点 x*。[1]

3 数据集说明

3.1 数据集 1.

函数：$f(x) = 100(x_2 - x_1^2)^2 + (x_1 - 1)^2$

3.2 数据集 2.

函数：$f(x) = \sum_{i=1}^{6} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$

3.3 数据集 3.

函数：$\parallel Ax - b \parallel$，其中 $A = \begin{pmatrix} 10 & 1 & 2 & 3 & 4 \\ 1 & 9 & -1 & 2 & -3 \\ 2 & -1 & 7 & 3 & -5 \\ 3 & 2 & 3 & 12 & -1 \\ 4 & -3 & -5 & -1 & 15 \end{pmatrix}$，$b = (12 \quad -27 \quad 14 \quad -17 \quad 12)^T$

4 关键代码展示

```
def quasi_newton_method(f, x):
    """
    BFGS校正的拟牛顿法
    输入：函数f，初始点x
    输出：函数f的极小点x
    """
    n = len(x)
    H = np.zeros([n, n], dtype=float)
    for i in range(n):
        H[i][i] = 1.
    k = 0
    maxk = 10000
    while k < maxk:
        d = np.zeros(n, dtype=float)
```

```
        g = grad(f, x)  # 梯度
        d = -1*np.matmul(H, g)  # 计算搜索方向
        judge, alpha = inexact_one_dimensional_search(f, x, d)  # 确定步长因子
        if not judge:
            return x, k+1, False
            break
        tmpx = x.copy()
        x = x+alpha*d  # 更新迭代点
        s = x-tmpx
        if np.linalg.norm(s, ord=2) < 1e-7:
            return x, k+1, True
        y = grad(f, x)-g
        H = bfgs_correction(H, y, s)
        k += 1
    print("拟牛顿法的迭代次数超过上限！")
    return x, k, False
```

```
def inexact_one_dimensional_search(f, x, d):
    """
    基于Wolfe-Powell准则的非精确一维搜索算法
    """
    k = 0
    maxk = 10000  # 迭代上限
    alpha0 = 1
    rho = 0.25
    sigma = 0.5

    phi0 = f(x)
    phi0_ = np.inner(grad(f, x), d)

    a1 = 0.
    a2 = alpha0
    phi1 = phi0
    phi1_ = phi0_
    alpha = a2*0.5

    while k < maxk:
        phi = f(x+alpha*d)
        if phi > phi0+rho*alpha*phi0_:
            alpha1 = a1+(a1-alpha)**2*phi1_*1. / \
                (2*((phi1-phi)-(a1-alpha)*phi1_))
```

```
            a2 = alpha
            alpha = alpha1
            continue
        phi_ = np.inner(grad(f, x+alpha*d), d)
        if phi_ >= sigma*phi0_:
            return True, alpha
        alpha1 = alpha-(a1-alpha)*phi_*1./(phi1_-phi_)
        a1 = alpha
        alpha = alpha1
        phi1 = phi
        phi1_ = phi_
        k += 1
    print("非精确一维搜索的迭代次数超过上限！")
    return False, alpha
```

5 程序输入输出说明

输入数据集时，将 f(x) 以函数的形式输入，x 以 numpy 数组的形式输入。输出结果时，先输出 f(x) 的初始点，若有解，则输出全局极小值点，再输出对应的值，最后输出拟牛顿法的迭代次数和运行时间。若非精确一维步长搜索迭代次数过大或者拟牛顿法迭代次数过大，则分别返回：”非精确一维搜索的迭代次数超过上限！”或”拟牛顿法的迭代次数超过上限！”。

注：测试所用的数据集 1、2、3 均在.py 文件中写出，以注释的方式选择调用。故程序中不采用以用户输入的形式调用数据集。

6 程序测试结果

6.1 数据集 1.

```
极小值点： [1.00000003 1.00000008]
极小值： 9.019287313028978e-15
拟牛顿法的迭代次数： 46
算法运行时间： 0.008194684982299805 单位：s

初始点： [-2.75280606  4.40176982]
极小值点： [1. 1.]
极小值： 1.7876569348778e-21
拟牛顿法的迭代次数： 41
算法运行时间： 0.007995843887329102 单位：s

初始点： [-7.64067752 -7.4404588 ]
极小值点： [0.99999998 0.99999995]
极小值： 5.398684278792045e-16
拟牛顿法的迭代次数： 69
算法运行时间： 0.012949705123901367 单位：s

初始点： [ 0.99223059 -4.8669427 ]
极小值点： [0.99999997 0.99999993]
极小值： 1.1908303564624533e-15
拟牛顿法的迭代次数： 21
算法运行时间： 0.0038361549377441406 单位：s

初始点： [4.36666029 5.21649744]
极小值点： [1. 1.]
极小值： 1.0532970501951655e-17
拟牛顿法的迭代次数： 40
算法运行时间： 0.0070688724517822266 单位：s
```

6.2 数据集 2.

```
极小值点： [1. 1. 1. 1. 1. 1.]
极小值： 1.1303940745065455e-19
拟牛顿法的迭代次数： 81
算法运行时间： 0.05706310272216797 单位：s

初始点： [-9.8775814   6.21495264  5.10459269  4.40776785 -6.43604035 -4.72077643]
极小值点： [1. 1. 1. 1. 1. 1.]
极小值： 4.669641448123255e-20
拟牛顿法的迭代次数： 69
算法运行时间： 0.04500126838684082 单位：s

初始点： [ 8.71364596 -1.57203515 -8.30727983 -5.21830791  6.95431863 -8.85793751]
极小值点： [1. 1. 1. 1. 1. 1.]
极小值： 3.4523196976208027e-19
拟牛顿法的迭代次数： 80
算法运行时间： 0.056258440017700195 单位：s

初始点： [ 8.98346134 -1.48175989 -2.49974622  0.97835373 -9.42445875  6.30160195]
极小值点： [1. 1. 1. 1. 1. 1.]
极小值： 1.2172228597350733e-20
拟牛顿法的迭代次数： 72
算法运行时间： 0.05253791809082031 单位：s

初始点： [-8.49798025 -0.91037291 -5.19126608 -4.10935602  3.75962194  2.41568216]
极小值点： [1. 1. 1. 1. 1. 1.]
极小值： 4.390400770427578e-23
拟牛顿法的迭代次数： 77
算法运行时间： 0.057203054428100586 单位：s
```

6.3 数据集 3.

```
极小值点： [ 1.00000003 -2.00000004  2.99999993 -1.99999999  0.99999993]
极小值： 4.534529260484527e-07
拟牛顿法的迭代次数： 47
算法运行时间： 0.04403996467590332 单位：s

初始点： [ 7.05314745 -4.94138182  2.28222251  9.79801768  9.09836635]
极小值点： [ 1.00000004 -2.00000005  2.99999991 -1.99999999  0.99999996]
极小值： 3.453868662855219e-07
拟牛顿法的迭代次数： 49
算法运行时间： 0.04601621627807617 单位：s

初始点： [-1.64590924 -8.79796313  7.68617442 -2.69557571  8.54575533]
极小值点： [ 1.00000006 -2.00000007  2.99999986 -1.99999997  0.99999992]
极小值： 3.3875164034025637e-07
拟牛顿法的迭代次数： 49
算法运行时间： 0.04901576042175293 单位：s

初始点： [-8.13932327 -6.7965282   2.95114781 -3.84603467  9.82264888]
极小值点： [ 0.99999985 -1.99999988  3.0000003  -2.00000005  1.00000017]
极小值： 7.602902990700021e-07
拟牛顿法的迭代次数： 47
算法运行时间： 0.043009042739868164 单位：s

初始点： [ 6.19668134  2.1160511   5.6714458  -4.34491693  8.88930856]
极小值点： [ 1.00000004 -2.00000004  2.99999994 -2.          0.99999996]
极小值： 2.1592448545014121e-07
拟牛顿法的迭代次数： 51
算法运行时间： 0.048021793365478516 单位：s
```

7 分析总结

从数据集的测试结果来看，基于 Wolfe-Powell 准则非精确一维步长搜索的拟牛顿法可以很好的解决函数的最小值问题。本次使用的数据集 1、2 是 Rosenbrock 函数的二元和六元情况，数据集 3 的函数是一个最小二乘问题，本算法成功地解决了这些问题。从测试结果来看，初始点的选取会在误差范围内影响极小值点以及拟牛顿法的迭代次数。而且由于 Rosenbrock 函数本身的非凸性质，部分初始点可能会有无解的情况出现。造成这种情况的可能原因是由于对于函数梯度的计算采用了差商法的通用函数，导致出现了误差。同样的，差商的选取不能太大或者太小，否则积累误差过大会出现误解或者无解。而且由于 Rosenbrock 函数本身的非凸性质，部分初始点可能会有无解的情况出现，这体现了 Theme 2.1. 中本算法全局收敛性的目标函数的要求。

值得注意的是，因为无法确定满足 $f(x^{(k)}+\alpha d^{(k)})=f(x^{(k)})$ 的最小正数 α，故在最初程序的非精确一维步长搜索中，初始的 α 选取依赖了 random.uniform$(0,1)$ 函数，故测试结果中出现了迭代次数过大而导致无解的情况。这说明了这种在全局最小值点附近病态的函数，对初始的 α 选取比较敏感。而算法中使用 random.uniform$(0,1)$ 函数代替某一个常值，一方面使得算法出现了不稳定性，另一方面却也避免了由于初始的 α 选取而导致的无解的情况。初始的 α 选取还会影响解的精度（一般在 1e-8），以及算法迭代的次数。同样的，$\bar{\alpha}$、ρ 和 σ 的选取也可能会对结果产生影响，但本程序中没有进行考虑。为了保证程序运行的稳定性，方便对结果进行总结，本程序最终采用初始的 $\alpha=0.5a_2$。

参考文献

[1] 刘浩洋, 户将, 李勇锋, 文再文, 最优化：建模、算法与理论, 高教出版社, ISBN：9787040550351
H. Liu, J. Hu, Y. Li, Z. Wen, Optimization: Modeling, Algorithm and Theory (in Chinese)