非精确一维搜索求解无约束最优化问题

PB19030861 王湘峰

一、问题描述

给定一个无约束最优化问题

$min \quad f(x) = 100(x_3 - x_2^2)^2 + (x_2 - 1)^2 + 100(x_2 - x_1^2)^2 + (x_1 - 1)^2.$

实验使用拟牛顿法中的 DFP 以及 BFGS 作为下降方法，同时采用基于**Wolfe-Powell**准则的非精确一维搜索确定步长α 的值

二、算法原理

① 拟牛顿法

无约束最优化最经典的方法是牛顿法，牛顿法具有二阶收敛的良好性质。

记 $g(x) = \nabla f(x)$，$H(x) = \nabla^2 f(x)$. 牛顿法的迭代形式为

$$x_{k+1} = x_k + H_k^{-1} g_k$$

牛顿法虽然收敛速度快，但是需要计算海塞矩阵的逆矩阵 H^{-1} ，计算工作量较大且目标函数的海森矩阵不能保证正定，从而使得牛顿法失效。为了克服这两个问题，人们提出了拟牛顿法。

拟牛顿条件

对 $\nabla f(x)$ 在 x_k 处做泰勒展开我们得到了以下近似：

$\nabla f(x) = g_k + H_k(x - x_k)$

取 $x = x_{k+1}$ 即可得到 $g_{k+1} - g_k = H_k(x_{k+1} - x_k)$

记 $y_k = g_{k+1} - g_k, \delta_k = x_{k+1} - x_k$， 则

$$y_k = H_k \delta_k$$

此为拟牛顿条件，DFP算法以 G_k 作为对 H_k^{-1} 的近似，并且使其满足拟牛顿条件

DFP方法

DFP算法以 G_k 作为对 H_k^{-1} 的近似，它的迭代格式为：

$$G_{k+1} = G_k + \frac{\delta_k \delta_k^T}{\delta_k^T y_k} - \frac{G_k y_k y_k^T G_k}{y_k^T G_k y_k}$$

BFGS方法

BFGS算法用 B_k 作为 H_k 的近似，与DFP相比，BFGS性能更佳。若记 $G_k = B_k^{-1}$ ，那么应用Sherman-Morrison公式可以得到BFGS的迭代格式为：

$$G_{k+1} = (I - \frac{\delta_k y_k^T}{\delta_k^T y_k}) G_k (I - \frac{\delta_k y_k^T}{\delta_k^T y_k})^T + \frac{\delta_k \delta_k^T}{\delta_k^T y_k}$$

不论是哪种方法，都可以证明，如果初始矩阵 G_0 是正定的，那么迭代过程中的每个矩阵 G_k 都是正定的。一般选取 $G_0 = I$

②非精确一维搜索之Wolfe-Powell准则

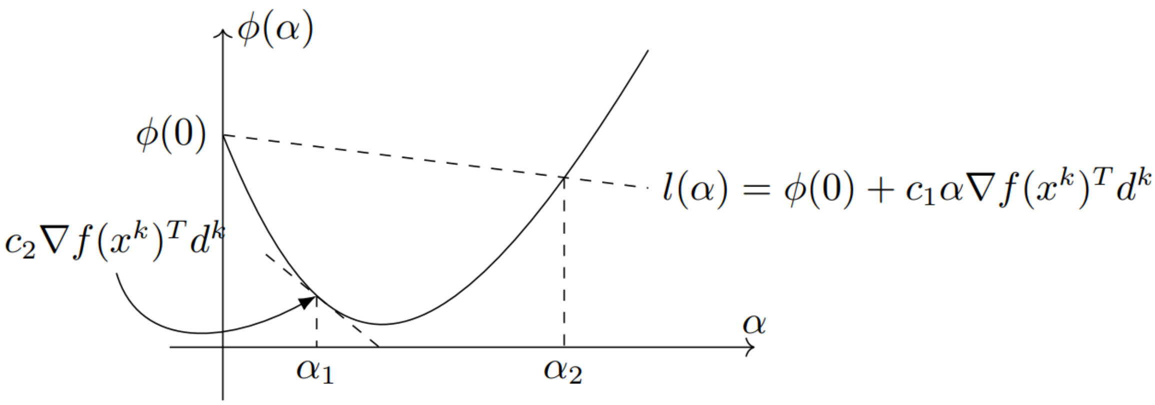
Wolfe-Powell准则

设 d_k 是点 x_k 处的下降方向，若：

$$f(x_k+\alpha d_k) \le f(x_k)+\rho\alpha\nabla f(x_k)^T d_k \qquad condition1$$

$$\nabla f(x_k+\alpha d_k)^T d_k \ge \sigma\nabla f(x_k)^T d_k \qquad condition2$$

则称步长 α 满足**Wolfe**准则，其中 $\rho \in (0,\frac{1}{2}),\sigma \in (\rho,1)$ ，是给定的常数。



当目前所选取的 α 不满足Wolfe准则时，可构造二次插值多项式并求其极小点 $\hat{\alpha}$ 来更新 α 的值。

搜索算法（伪代码）

```
# initiate a1,a2 and original alpha
while iteration_times < iteration_limit:
    if condition1 == True:
        if condition2 == True:
            return alpha
        else:
            alpha = interpolation_method
    else:
        alpha = expolation_method
return alpha
```

其中**interpolation_method(内插法)**公式为

$$\hat{\alpha}=\alpha_1-\frac{\nabla f(x_k)^T d_k(\alpha-\alpha_1)^2}{2[f(x_k+\alpha d_k)-f(x_k)-\nabla f(x_k)^T d_k(\alpha-\alpha_1)]}$$

并更新 $\alpha_2=\alpha,\alpha=\hat{\alpha}$

extrapolation_method(外插法)公式为

$$\hat{\alpha}=\alpha+\frac{\nabla f(x_k+\alpha d_k)^T d_k(\alpha-\alpha_1)}{\nabla f(x_k)^T d_k-\nabla f(x_k+\alpha d_k)^T d_k}$$

并更新 $\alpha_1=\alpha,\alpha=\hat{\alpha}$

三、源码展示

```
def Quasi_Newton(x, method):
    # 初始化
    epsilon, limit = 1e-5, 2000
    n = x.shape[0]
    H = np.eye(n)
    step = 1
    g0 = gradient(x)
    d = -H @ g0
    Y = [f(x)]
    I = np.eye(n)
    while step < limit and np.sqrt(np.sum(g0 ** 2)) > epsilon:
        alpha = Wolfe_Powell(x, d)
        delta = alpha * d
        x += delta
        g1 = gradient(x)
        y = g1 - g0
        if method == '1':
            H += (delta @ delta.T) / (delta.T @ y) - (H @ y @ y.T @ H) / (y.T @
H @ y)
        else:
            H = (I - delta @ y.T / (delta.T @ y)) @ H @ (I - delta @ y.T /
(delta.T @ y)).T + delta @ delta.T / (delta.T @ y)
        d = -H @ gradient(x)
        g0 = g1
        Y.append(f(x))
        step += 1
    print('x = ', x.tolist())
    print('▽f(x) = ', gradient(x).tolist())
    print('f(x) = ', f(x))
    print('迭代次数为', step)
    X = range(step)
    plt.xlabel('Iteration steps')
    plt.ylabel('f(x)')
    plt.plot(X, Y)
    plt.show()
```

以上为函数Quasi_Newton的执行过程，其中while循环中的部分即是函数值下降的过程；最大迭代次数为2000次。

```
def Wolfe_Powell(x, d):
    # 初始化
    step, limit = 0, 1000
    rho, sigma = 0.15, 0.75
    Alpha = 100
    a1, a2 = 0, Alpha
    alpha = (a1 + a2) / 2
    phi1 = f(x)
    phi1_ = gradient(x).T @ d
    while step < limit:
        step += 1
        phi = f(x + alpha * d)
        if phi <= phi1 + rho * alpha * phi1_:  # condition1
            phi_ = gradient(x + alpha * d).T @ d
            if phi_ >= sigma * phi1_:  # condition2
```

```
                return alpha
            else:  # 外插法
                new = alpha + phi_ * (alpha - a1) / (phi1_ - phi_)
                a1 = alpha
                alpha = new
                phi1 = phi
                phi1_ = phi_
        else:  # 内插法
            new = a1 - 0.5 * phi1_ * (alpha - a1) ** 2 / (phi - phi1 - (alpha - a1) * phi1_)
            a2 = alpha
            alpha = new
    return alpha
```

以上代码为应用Wolfe-Powell准则的非精确一维搜索的函数，最大迭代次数为1000次。

四、用户指南

环境配置

本程序使用Python 3.8编写，请使用Python3及以上编译器运行。本程序需要调用numpy库和matplotlib库，安装方式为，在cmd中输入

```
pip install numpy
pip install matplotlib
```

输入说明

本程序支持二维和三维情况下的**Rosenbrock**函数，如果输入为两个坐标则自动识别为二维函数；输入三个坐标则按照三维情况处理。

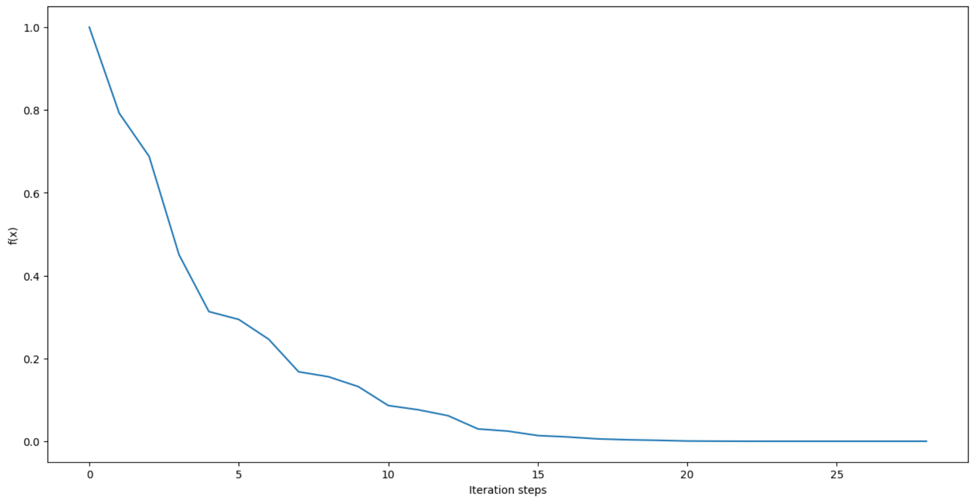
输出说明

程序输出结果为：算法终止时的坐标、该点的梯度、该点的函数值、迭代次数，以及函数值的下降曲线。

五、程序测试结果

测试1：初始点：$(0,0)$，下降方法：DFP

下降曲线如下：

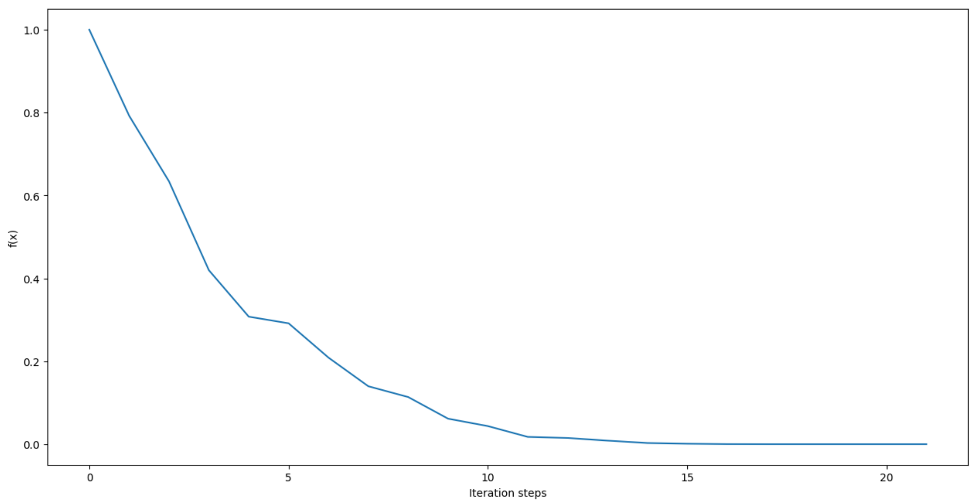


测试2：初始点：$(0,0)$，下降方法：BFGS

```
运行:    无约束最优化
"D:\python env\Scripts\python.exe" D:/干点正事/大三上/运筹学/实验报告/2/无约束最优化.py
--- 程序名称：非精确一维搜索求解无约束最优化 ---
------ 王湘峰PB19030861 ------
请输入初始点坐标（以空格作为分割）
0 0
请选择下降方法：1 → DFP 2 → BFGS
2
x =  [[1.0000000012632027], [1.000000002466945]]
∇f(x) =  [[2.6310491149163326e-08], [-1.1892042905969902e-08]]
f(x) =  [1.94923266e-18]
迭代次数为 22

进程已结束，退出代码为 0
```

下降曲线如下：

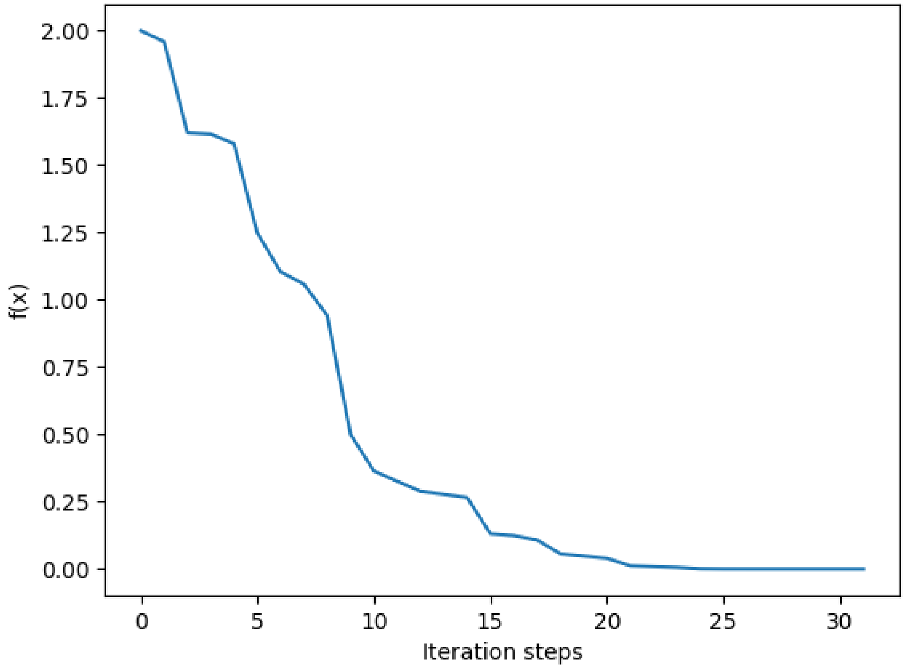


测试3：初始点：$(0,0,0)$，下降方法：DFP

```
运行:  无约束最优化
"D:\python env\Scripts\python.exe" D:/干点正事/大三上/运筹学/实验报告/2/无约束最优化.py
--- 程序名称：非精确一维搜索求解无约束最优化 ---
------ 王湘峰PB19030861 ------
请输入初始点坐标（以空格作为分割）
0 0 0
请选择下降方法：1 → DFP 2 → BFGS
1
x =  [[1.00000002874374], [1.0000000577446164], [1.0000001277888944]]
∇f(x) =  [[-4.536669247257103e-08], [-4.7529472387737115e-06], [2.459931636167312e-06]]
f(x) =  [1.92954143e-14]
迭代次数为 32

进程已结束，退出代码为 0
```

下降曲线如下：

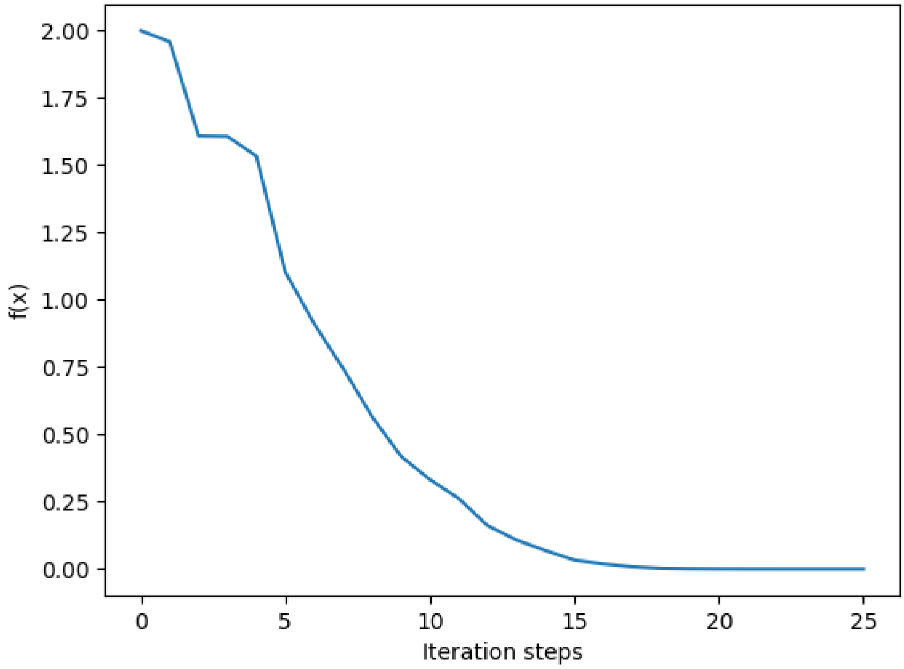


测试4：初始点：$(0,0,0)$ ，下降方法：BFGS

```
运行:  无约束最优化
"D:\python env\Scripts\python.exe" D:/干点正事/大三上/运筹学/实验报告/2/无约束最优化.py
--- 程序名称：非精确一维搜索求解无约束最优化 ---
------ 王湘峰PB19030861 ------
请输入初始点坐标（以空格作为分割）
0 0 0
请选择下降方法：1 → DFP 2 → BFGS
2
x =  [[0.9999999985474646], [0.9999999971984604], [0.9999999943915685]]
∇f(x) =  [[-4.431758864977693e-08], [1.724413389258829e-08], [-1.070477040343576e-09]]
f(x) =  [1.10332207e-17]
迭代次数为 26

进程已结束，退出代码为 0
```

下降曲线如下：



由**Rosenbrock**函数的性质知，函数有且仅有一个极小值点$(1,1,1)$，二维时为$(1,1)$.这与程序给出的结果一致，证明了程序的正确性。

同时发现，在相同的初始点和参数的条件下，BFGS方法要比DFP法迭代次数要少，同时目标函数的值也更小，因此BFGS方法效更佳。

六、总结与分析

本次实验同时实现了基于Wolfe-Powell准则的非精确一维搜索和基于拟牛顿法的DFP下降方法。从下降曲线中可以看出该方法收敛速度是很快的；同时由于下降函数的性质，不同的初始点最终得到的结果会有略微不同（偏"左"或者偏"右"），但是条件 $||\nabla f(x)|| < \epsilon$ 的存在保证了解的误差在可接受的范围内。

总之，本次实验使我更加直观的认识了运筹学算法的巧妙之处，也加深了我对算法的理解，锻炼了动手能力，收获良多。