等式约束熵极大化问题

一、问题描述

考虑以下等式约束问题

$$min\ f(x) = \sum_{i=1}^{n} x_i log x_i$$

$$subject\ to\ Ax = b$$

其中$dom\ f = \mathrm{R}^{\mathrm{n}}_{++}$，$A \in R^{p\times n}$，$p < n$。

生成一个$n = 100, p = 30$的问题实例，随机选择A（保证其为满秩），随机选择一个正向量作为$\hat{x}$，然后令$b = A\hat{x}$。

使用以下方法计算该问题的解：

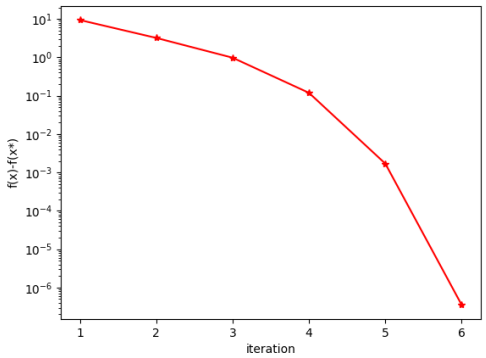
（a）标准 Newton 方法。可以选取初始点为$\hat{x}$。

（b）不可行初始点的 Newton 方法。可以选用初始点为$\hat{x}$，也可以使用初始点为$\mathbf{1}$。

（c）对偶 Newton 方法。即将标准 Newton 方法应用于对偶问题。

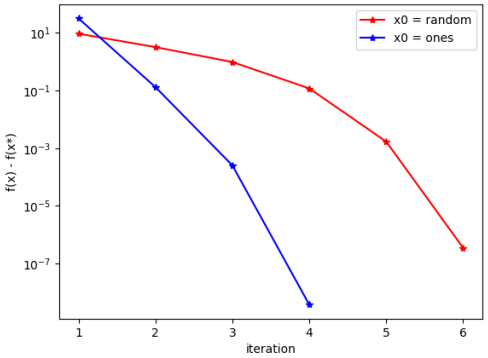
二、标准 Newton 方法

使用从可行点开始的等式约束下的 Newton 方法对此问题进行求解。使用回溯直线搜索进行迭代步长的选择，设置$\alpha = 1, \beta = 0.5$，设置迭代停止条件的$\epsilon = 1\text{e-}7$，具体代码见附件。可以得到如下误差随迭代次数的变化曲线。



三、不可行初始点的 Newton 方法

使用从不可行点开始的等式约束下的 Newton 方法对此问题进行求解。使用回溯直线搜索进行迭代步长的选择，设置$\alpha = 1, \beta = 0.5$，设置迭代停止条件的$\epsilon = 1\text{e-}7$，具体代码见附件。选择不同的初始点（蓝色为$\hat{x}$，红色为$\mathbf{1}$），可以得到如下误差随迭代次数的变化曲线。



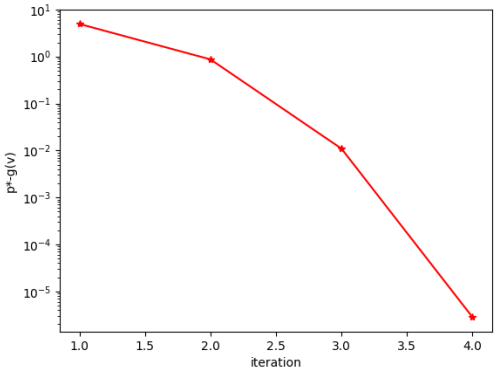
四、对偶 Newton 方法

对偶问题为:

$$max - b^T v - \sum_{i=1}^{n} e^{-a_i^T v - 1}$$

其中a_i是A的第i列。

使用回溯直线搜索进行迭代步长的选择, 设置$\alpha = 1, \beta = 0.5$, 设置迭代停止条件的$\epsilon = 1e\text{-}8$, 可以得到如下误差随迭代次数的变化曲线。



五、分析

以上每种方法得到的结果是相同的, 比如, 设置随机种子为 123, 得到的结果均为-31.85。三种方法得到相同效果的迭代次数是相近的，在不可行点的 Newton 方法中，不同的初始点对收敛性能也会产生一定的影响。

三种方法的计算量是相同的。在标准方法的不可行点 Newton 方法中，我们解系数如下所示的方程:

$$\begin{bmatrix} \nabla^2 f(x) & A^T \\ \mathrm{A} & 0 \end{bmatrix}$$

其中，$\nabla^2 f(x) = diag(x)^{-1}$，分块消除法使得方程的系数减为$A\ diag(x)\ A^T$。

在对偶方法中，我们解系数如下所示的方程:

$$-\nabla^2 g(v) = ADA^T$$

其中，D为对角矩阵$D_{ii} = e^{-a_i^T v - 1}$。

三种方法中，主要的计算为解如下形式的线性方程组:

$$A^T DAv = -g$$

其中，D是对角线元素为正的对角矩阵。

六、代码

（a）标准 Newton 方法

```
import numpy as np
import matplotlib.pyplot as plt

seed = 123
np.random.seed(seed)

MAXITERS = 1000
ALPHA = 0.01
```

```
BETA = 0.5
NTTOL = 1e-7

def optimize(x0, A):
    n = len(x0)
    p = A.shape[0]
    x = x0.copy()
    vals = []
    for _ in range(MAXITERS):
        val = np.dot(x, np.log(x))
        vals.append(val)
        grad = 1 + np.log(x)
        hess = np.diag(1 / x)
        sol = -np.linalg.solve(np.block([[hess, A.T], [A, np.zeros((p, p))]]), np.block([grad,
np.zeros(p)]))
        v = sol[:n]
        fprime = np.dot(grad, v)
        if abs(fprime) < NTTOL:
            break
        t = 1
        while np.min(x + t * v) <= 0:
            t = BETA * t
        while np.dot((x + t * v), np.log(x + t * v)) >= val + t * ALPHA * fprime:
            t = BETA * t
        x = x + t * v

    vals.append(np.dot(x, np.log(x)))
    return x, vals

if __name__ == '__main__':
    n = 100
    p = 30
    # 生成 p*n 的满秩矩阵
    while True:
        A = np.random.rand(p, n)
        if np.linalg.matrix_rank(A) == p:
            break
    x0 = np.random.rand(n)
    b = np.dot(A, x0)
    x, vals = optimize(x0, A)
    res = [vals[i]-vals[-1] for i in range(len(vals)-1)]
    plt.plot([i+1 for i in range(len(res))], res, 'r*-')
    plt.xlabel('iteration')
    plt.ylabel('f(x)-f(x*)')
```

```
    plt.yscale('log')
    plt.show()
```

（b）不可行初始点的 Newton 方法

```
import numpy as np
import matplotlib.pyplot as plt

seed = 123
np.random.seed(seed)

MAXITERS = 1000
ALPHA = 0.01
BETA = 0.5
RESTOL = 1e-7

def optimize(x0, A, b):
    n = len(x0)
    p = A.shape[0]
    x = x0.copy()
    nu = np.zeros(p)
    vals = []
    resdls = []
    for _ in range(MAXITERS):
        val = np.dot(x, np.log(x))
        vals.append(val)
        r = np.concatenate([1 + np.log(x) + np.dot(A.T, nu), np.dot(A, x) - b])
        resdls.append(np.linalg.norm(r))
        sol = -np.linalg.solve(np.block([[np.diag(1.0 / x), A.T], [A, np.zeros((p, p))]]), r)
        Dx = sol[:n]
        Dnu = sol[n:]
        if np.linalg.norm(r) < RESTOL:
            break
        t = 1
        while np.min(x + t * Dx) <= 0:
            t = BETA * t
        while np.linalg.norm(np.concatenate([1 + np.log(x + t * Dx) + np.dot(A.T, nu + t *
Dnu), np.dot(A, x + t * Dx) - b])) > (1 - ALPHA * t) * np.linalg.norm(r):
            t = BETA * t
        x += t * Dx
        nu += t * Dnu
    vals.append(np.dot(x, np.log(x)))
    return x, vals

if __name__ == '__main__':
```

```
    n = 100
    p = 30
    # 生成 p*n 的满秩矩阵
    while True:
        A = np.random.rand(p, n)
        if np.linalg.matrix_rank(A) == p:
            break
    x0 = np.random.rand(n)
    b = np.dot(A, x0)
    x_1, vals_1 = optimize(x0, A, b)
    x0 = np.ones(n)
    x_2, vals_2 = optimize(x0, A, b)
    res_1 = [vals_1[i] - vals_1[-1] for i in range(len(vals_1) - 1)]
    res_2 = [vals_2[i] - vals_2[-1] for i in range(len(vals_2) - 1)]
    plt.plot([i+1 for i in range(len(res_1))], res_1, 'r*-', label='x0 = random')
    plt.plot([i+1 for i in range(len(res_2))], res_2, 'b*-', label='x0 = ones')
    plt.xlabel('iteration')
    plt.ylabel('f(x) - f(x*)')
    plt.yscale('log')
    plt.legend()
    plt.show()
```

（c）对偶 Newton 方法

```
import numpy as np
import matplotlib.pyplot as plt

seed = 123
np.random.seed(seed)

MAXITERS = 100
ALPHA = 0.01
BETA = 0.5
NTTOL = 1e-8

def optimize(A, b):
    nu = np.zeros(p)
    vals = []
    for _ in range(MAXITERS):
        val = np.dot(b.T, nu) + np.sum(np.exp(-np.dot(A.T, nu) - 1))
        vals.append(val)
        grad = b - np.dot(A, np.exp(-np.dot(A.T, nu) - 1))
        hess = np.dot(np.dot(A, np.diagflat(np.exp(-np.dot(A.T, nu) - 1))), A.T)
        v = -np.linalg.solve(hess, grad)
        fprime = np.dot(grad.T, v)
```

```
            if abs(fprime) < NTTOL:
                break
            t = 1
            while np.dot(b.T, nu + t * v) + np.sum(np.exp(-np.dot(A.T, nu + t * v) - 1)) > val + t
* ALPHA * fprime:
                t = BETA * t
            nu = nu + t * v
        vals.append(np.dot(b, nu) + np.sum(np.exp(-np.dot(A.T, nu) - 1)))
        return nu, vals

if __name__ == '__main__':
        n = 100
        p = 30
        #  生成 p*n 的满秩矩阵
        while True:
            A = np.random.rand(p, n)
            if np.linalg.matrix_rank(A) == p:
                break
        x0 = np.random.rand(n)
        b = np.dot(A, x0)
        result, vals = optimize(A, b)
        res = [vals[i] - vals[-1] for i in range(len(vals) - 1)]
        plt.plot([i+1 for i in range(len(res))], res, 'r*-')
        plt.xlabel('iteration')
        plt.ylabel('p*-g(v)')
        plt.yscale('log')
        plt.show()
```