中国科学技术大学

最优化算法报告

Primal-dual 内点法求解凸优化问题

学生姓名: 陈泽豪

所在院系: 数学科学学院

学生学号: SA22001009

完成时间: 2023 年 6 月 29 日

摘要

本报告会从凸优化问题入手，使用一种求解凸优化问题的标准方法：Primal-dual 内点算法。报告会先介绍算法的具体内容即算法流程介绍，然后给出结果以及一些收敛性分析。

关键词：凸优化，Primal-dual 内点算法

一、Primal-dual 算法介绍

1.1 对 KKT 条件的简要介绍描述

在介绍凸优化的内点算法之前先介绍一下如下的优化问题：

$$\begin{aligned} min \quad & f_0(x) \\ s.t. \quad & f_i(x) \le 0, \quad i = 1, ..., m \\ & h_j(x) = 0 \quad j = 1, ..., p. \end{aligned} \tag{1}$$

其中定义域为:$\mathcal{D} = \cap_{i=0}^{m} \mathbf{dom}(f_i) \cap \cap_{j=1}^{p} \mathbf{dom}(h_j)$ 为非空的，它的最优解用 v^* 表示。类似求解 K-T 问题，定义一个关于这个不等式约束的最优化问题的 Lagrangian 函数：

$$L(x, \lambda, \gamma) = f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{j=1}^{p} \gamma_j h_j(x) \tag{2}$$

其中 λ_i 就是对应于 $f_i(x) \le 0$ 的 Lagrange multiplier，γ_j 为对应与 $h_j(x)$ 的 Lagrange multiplier。此时可以定义 Lagrange dual fuction 如下所示：

$$g(\lambda, \gamma) = \inf_{x \in \mathcal{D}} L(x, \lambda, \gamma) \tag{3}$$

易知又 $g(\lambda, \gamma) \le f_0(x^*)$ 成立。因此如何得到 $f_0(x^*)$ 的最佳下界估计就是求解如下的优化问题：

$$\begin{aligned} max \quad & g(\lambda, \gamma) \\ s.t. \quad & \lambda \ge 0 \end{aligned} \tag{4}$$

这个问题被称为 lagrange dual problem，一定为凸优化问题，设 (λ^*, γ^*) 为 dual optimal 的，并得到一个 g^* 为 optimal value。则有 $g^* \le v^*$ 成立。我们将 $v^* - g^*$ 称为 optimal duality gap，如果此时有 $v^* = g^*$ 成立，就称满足了强对偶条件。

Slater 条件：存在 $x \in \mathbf{relint}\mathcal{D}$ 满足：

$$f_i(x) < 0, \quad i = 1, 2, ..., m, \quad Ax = b. \tag{5}$$

其中有 $\mathbf{relint}\mathcal{D} = \{x \in \mathcal{D} | B(x, r) \cap \mathbf{aff}\mathcal{D} \subset \mathcal{D} \, for \, some \, r > 0\}$。即存在相对可行内点使不等式严格成立。如果原问题为凸优化问题，同时再加上 Slater 条件便有强对偶条件成立，在强对偶成立的情况下有如下的式子推导成立，设 x^* 为原问题 (1) 可行最优解，(λ^*, γ^*) 为 lagrange dual problem (4) 的可行最优解，有：

$$\begin{aligned} f_0(x^*) = g(\lambda^*, \gamma^*) &= \inf_{x \in \mathcal{D}} (f_0(x) + \sum_{i=1}^{m} \lambda_i^* f_i(x) + \sum_{j=1}^{p} \gamma_j^* h_j(x)) \\ &\le f_0(x^*) + \sum_{i=1}^{m} \lambda_i^* f_i(x^*) + \sum_{j=1}^{p} \gamma_j^* h_j(x^*) \\ &\le f(x^*). \end{aligned} \tag{6}$$

因此等号全部成立的条件就是 $\lambda_i^* f_i(x^*) = 0,\ i = 1, 2, ..., m$。因此若有强对偶条件成立，原问题就变成了如下的 KKT 条件：

$$(KKT) = \begin{cases} f_i(x^*) \le 0, \forall i \\ h_j(x^*) = 0, \forall j \\ \lambda_i^* \ge 0, \forall i \\ \lambda_i^* f_i(x^*) = 0, \forall i \\ \nabla f_0(x^*) + \sum_{i=1}^m \lambda_i^* \nabla f_i(x^*) + \sum_{j=1}^p v_j^* \nabla h_j(x^*) = 0. \end{cases} \tag{7}$$

1.2 Primal-dual 算法

下面就来求解如下所示的凸优化问题：

$$\begin{aligned} min \quad & f_0(x) \\ s.t. \quad & f_i(x) \le 0, i = 1, 2, ..., m \\ & Ax = b \end{aligned} \tag{8}$$

采用 logarithmic barrier $\phi(x) = -\sum_{i=1}^m log(-f_i(x))$。其定义域即为 $\mathbf{dom}\phi = \{x \in \mathbf{R}^n | f_i(x) < 0, i = 1, 2, ..., m\}$，即可行域的一个内点。假设存在相对可行内点，即满足 Slater 条件。将上面的问题 (8) 变成如图所示的优化问题：

$$\begin{aligned} min \quad & f_0(x) + \sum_{i=1}^m -(1/t) log(-f_i(x)) \\ s.t. \quad & Ax = b \end{aligned} \tag{9}$$

将问题 (9) 的两侧乘以 t 就得到了如下的形式：

$$\begin{aligned} min \quad & t f_0(x) + \phi(x) \\ s.t. \quad & Ax = b \end{aligned} \tag{10}$$

因此对 $t > 0$，定义 $x^*(t)$ 为 (10) 的最优解，定义 $\{x^*(t) | t > 0\}$ 为 central points。而我们可以通过此时求出的最优解对比原问题最优解之间的差距。问题 (10) 的 KT 条件推出了如下的等式：

$$\begin{aligned} 0 &= t\nabla f_0(x^*(t)) + \nabla\phi(x^*(t)) + A^T\gamma \\ &= t\nabla f_0(x^*(t)) + \sum_{i=1}^m \frac{1}{-f_i(x^*(t))} \nabla f_i(x^*(t)) + A^T\gamma \end{aligned} \tag{11}$$

因此与问题 (8) 的 KKT 条件相比，有 $\lambda_i^*(t) = 1/-tf_i(x^*(t))$，$\gamma^*(t) = \gamma/t$。即此时的 $x^*(t)$ 极小化了 lagrange 函数 $L(x, \lambda^*, \gamma^*) = f_0(x) + \sum_{i=1}^m \lambda_i^* f_i(x) + (\gamma^*)^T(Ax - b)$。。因此此时有：

$$\begin{aligned} g(\lambda^*(t), \gamma^*(t)) &= \min_x L(x, \lambda^*(t), \gamma^*(t)) \\ &= f_0(x^*(t)) + \sum_{i=1}^m \lambda_i^* f_i(x^*(t)) + (\gamma^*)^T(Ax^*(t) - b) \\ &= f_0(x^*) - m/t. \end{aligned} \tag{12}$$

因此有 $f_0(x^*(t)) - m/t \le v^*$，且当 $t \to +\infty$ 时有 $f_0(x^*(t)) \to v^*$。

于是在加了 barrier 之后的问题 (10) 的 KT 条件实际上就是修改了 KKT 条件的一个等式条件得到如下所示的 modified KKT 条件：

$$(KKT) = \begin{cases} f_i(x^*) \leq 0, \forall i \\ Ax^* = b \\ \lambda_i^* \geq 0, \forall i \\ \lambda_i^* f_i(x^*) = -1/t, \forall i \\ \nabla f_0(x^*) + \sum_{i=1}^m \lambda_i^* \nabla f_i(x^*) + \sum_{j=1}^p v_j^* \nabla h_j(x^*) = 0. \end{cases} \tag{13}$$

接下来就是利用 primal-dual search 方式进行方程组的求解。本质上就是利用了 Newton-Raphson 迭代。定义：

$$r_t(x, \lambda, \gamma) = \begin{pmatrix} \nabla f_0(x) + J(f(x))^T \lambda + A^T \gamma \\ -diag(\lambda) f(x) - (1/t)1 \\ Ax - b \end{pmatrix} \tag{14}$$

第一项为 dual residual，第二项为 centrality residual，第三项为 primal residual。因此 (13) 的条件等价于求解 $r_t(x, \lambda, \gamma) = 0$。根据 Newton-Raphson 迭代有：$J(r_t(y))\delta_y = -r_t(y)$。即有：

$$\begin{pmatrix} \nabla^2 f_0(x) + \sum_{i=1}^m \lambda_i \nabla^2 f_i(x) & J(f(x))^T & A^T \\ -diag(\lambda) J(f(x)) & -diag(f(x)) & 0 \\ A & 0 & 0 \end{pmatrix} \begin{pmatrix} \delta_x \\ \delta_\lambda \\ \delta_\gamma \end{pmatrix} = \begin{pmatrix} r_{dual} \\ r_{central} \\ r_{primal} \end{pmatrix} \tag{15}$$

收敛终止条件使用 surrogate duality gap $\hat{\eta}(x, \lambda) = -f(x)^T \lambda$。

在求解出了上面的 δ_y 之后，最重要的就是迭代步的选取，一定要使得整体存在下降趋势同时又要落在可行域内，否则内点算法就会出现问题。因此：

1. 步长 α 的选择需要使得 $\lambda + \alpha\delta_\lambda$ 仍为正的，因此有 $\alpha^{max} = sup\{\alpha \in [0,1] | \lambda + \alpha\delta_\lambda \geq 0\} = min\{1, min_i\{-\lambda_i/\delta_{\lambda_i} | \delta_{\lambda_i} < 0\}\}$。
2. 坐 backtracking，$\alpha = 0.99\alpha^{max}$，然后不断乘以 $\beta \in (0,1)$ 直到 $f(x + \alpha\delta_x) < 0$。
3. 继续不断乘以 β 直到 $\|r_t(x + \alpha\delta_x, \lambda + \alpha\delta_\lambda, \gamma + \alpha\delta_\gamma)\|_2 \leq (1 - \tau\alpha)\|r_t(x, \lambda, \gamma)\|_2$，这里 $\tau \in [0.01, 0.1]$。

最终整个算法流程如下所示：

1. 给定 x_0, λ_0, γ_0，其中 x_0 满足 $f_i(x_0) < 0, \forall i; \lambda_0 > 0$，选取 $\theta > 1, \epsilon > 0$。
2. 确定初始的 $t = \theta(m/\hat{\eta})$，这里 m 为不等式约束个数。
3. 计算出 primal-dual search 的 δ_y。
4. 进行迭代步系数 α 的确认，并使得 $y = y + \alpha\delta_y$。
5. 如果满足 $\|r_{primal}\|_2 \leq \epsilon, \|r_{dual}\|_2 \leq \epsilon, \hat{\eta} < \epsilon$，则停止迭代，否则回到第二步。

二、代码处理

2.1 代码处理

计算 Jacobian 矩阵以及 Hessian 矩阵的过程与前一次作业的 SQP 中类似，在代码中使用 sympy 库的方式计算一个函数的符号梯度以及符号 Hessian 阵，并在需要使用的时候转化为数值形式：

```
# 获取雅克比矩阵
def jacobi(self):
self.jac_m = self.funcs.jacobian(self.vars)
self.jac_state = True
return self.jac_m

# 返回numerical jacobi
def n_jacobi(self, x):
if not self.jac_m:
self.jacobi()
return (sy.lambdify(self.vars, self.jac_m, 'numpy'))(x[0], x[1])

# 获取海塞矩阵
def hessian(self):
self.His_m = sy.hessian(self.funcs, self.vars)
self.hes_state = True
return self.His_m

# 返回numerical hessian
def n_hessian(self, x):
if not self.hes_state:
self.hessian()
return (sy.lambdify(self.vars, self.His_m, 'numpy'))(x[0], x[1])
```

除了 Jacobian 矩阵的计算以及 Hessian 矩阵的计算，这一次作业还需要手动实现 Newton 迭代法，并在考虑到保持始终为内点的情况下进行步长的选取，这一部分代码均在 MyConvexSolver.py 文件中进行了实现。对于初始点的选取，所选的例子都较为简单的可以选取合适的初始点，因此直接给了初始迭代点，如果无法轻易看出迭代点的话还需要实现寻找初值的代码。

Newton 迭代法的代码具体如下所示：

```
# Newton迭代法
def newton_iteration(self, x, _lambda, _gamma, t):
n = self.n
m = self.m
p = self.p
total_dim = n + m + p
jac_total_res = np.zeros((total_dim, total_dim))

jac_11 = self.n_hessian(self.test_func_hes, x)
for i in range(m):
jac_11 += _lambda[i] * self.n_hessian(self.cons_func_hes[i], x)

jac_12 = self.ineq_cons_jac(x).T
jac_13 = self.A.T
jac_21 = -np.diag(_lambda) @ self.ineq_cons_jac(x)
jac_22 = -np.diag(self.ineq_cons_val(x))
jac_31 = self.A

jac_total_res[:n, :n] = jac_11
jac_total_res[:n, n:(n+m)] = jac_12
jac_total_res[:n, (n+m):] = jac_13
jac_total_res[n:(n+m), :n] = jac_21
jac_total_res[n:(n+m), n:(n+m)] = jac_22
jac_total_res[(n+m):, :n] = jac_31

# 接下来只需求解方程组 jac_total_res @ delta_(x,lambda,gamma) = - total_res
return np.linalg.solve(jac_total_res, -self.total_res(x, _lambda, _gamma, t))
```

三、实验结果展示

在上述准备工作都完成之后,代码主体部分就放在 MyConvexSolver.py 文件内的 primal_dual_convex algorithm 函数中，实现如下所示：

```
 # 完成 primal-dual 内点算法
def primal_dual_convex_algorithm(self, x0):
n = 2
```

```
m = len(self.cons_with_bounds)
p = len(self.A)
xk = x0.copy()
_lambda = np.ones(m)
_gamma = np.ones(p)

# 记录中间迭代结果
self.myconvex_intermedium_result = []
self.myconvex_intermedium_result.append(xk.copy())

epoch = 200
for i in range(epoch):
t = self.sita * m / self.surrogate_duality_gap(xk, _lambda)

# 进行newton迭代求解
delta_y = self.newton_iteration(xk, _lambda, _gamma, t).flatten()

# 进行迭代步确定
alpha = self.line_search(xk, _lambda, _gamma, t, delta_y)

xk += alpha * delta_y[:n]
_lambda += alpha * delta_y[n:m+n]
_gamma += alpha * delta_y[m+n:]

self.myconvex_intermedium_result.append(xk.copy())

#  如果满足收敛条件，则退出循环
if (np.linalg.norm(self.dual_residual(xk, _lambda, _gamma)) < self.epi and
np.linalg.norm(self.primal_residual(xk)) < self.epi and
self.surrogate_duality_gap(xk, _lambda) < self.epi):
break

return xk, _lambda, _gamma
```

下面我们就三个测试样例来讨论算法的一些性质，并挑选样例三进行一定的收敛性分析。

3.1 测试样例一

下面进行结果展示，首先对于如下的 $test\_1$ 函数以及约束进行实验：

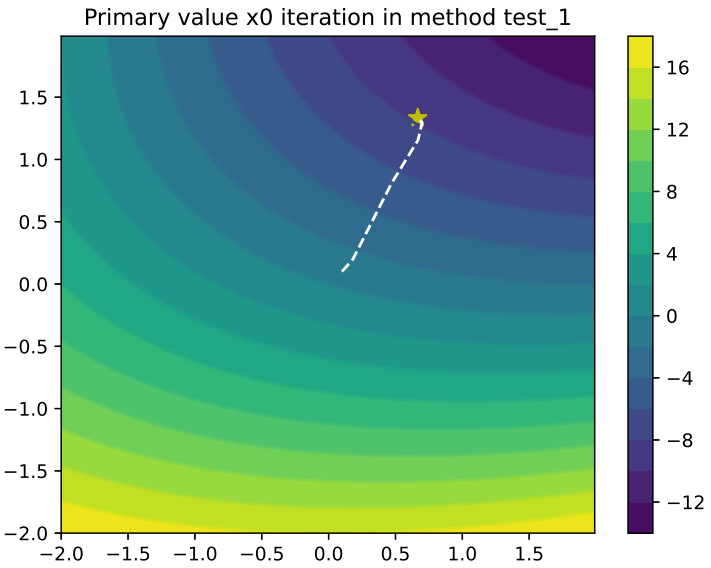
$$\begin{aligned} \min_{x_0,x_1} \quad & 0.5x_0^2 + x_1^2 - x_0x_1 - 2x_0 - 6x_1 \\ & x_0 + x_1 = 2 \\ & 2x_1 - x_0 \leq 2 \\ & 2x_0 + x_1 \leq 3 \\ & 0 \leq x_0 \\ & 0 \leq x_1 \end{aligned} \tag{16}$$

设定 dual residual 与 primal residual 的终止误差 $\epsilon_{feas} = 10^{-5}$，设定 surrogate duality gap 的终止误差 $\epsilon = 10^{-5}$，设定计算参数 $t = \theta(m/\eta)$ 时的 $\theta = 16.0$，做 backtracking 时的参数 $\beta = 0.9, \tau = 0.05$。取完全内点的初值为 $x_0 = [0.1, 0.1]$，最终在经历了 8 次迭代之后在精度范围内接近了精确值，精确值为 [2/3,4/3]，内置 minimize 的数值结果为 [0.66667,1.33333]，实现的 primal-dual 算法结果则如下所示：

迭代次数 k	0	3	6	8
x_k	[0.1,0.1]	[0.66838867 1.15490626]	[0.66753598 1.33246315]	[0.66666684 1.33333316]

表 1: test_1 函数实验中内点法迭代后 x_k 变化表格

具体的可视化等高线图如下所示：



图上的星形表示 minimize 的结果，白线就是我的程序从 [0.1,0.1] 出发接近最终结果的过程。算法具体的初值影响以及参数影响放在收敛性分析中讨论，这里先给出一个正确的结果。

3.2 测试样例二

下面又进行了第二个例子的实验：

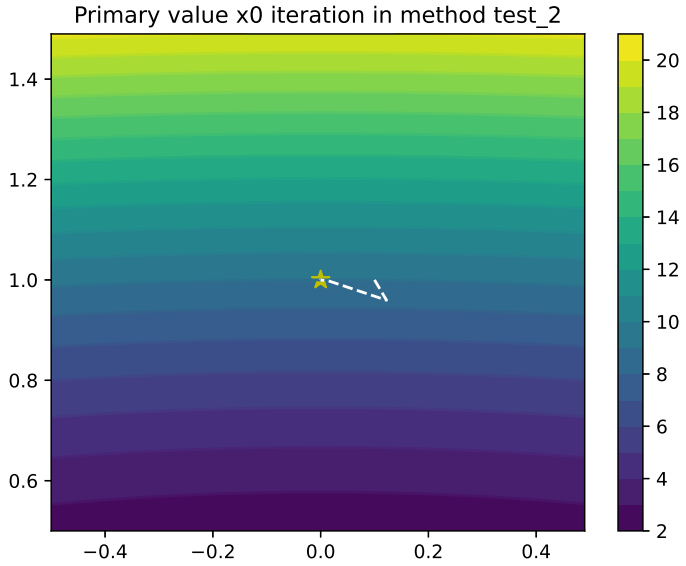
$$\begin{aligned} \min_{x_0,x_1} \quad & x_0^2 + 9x_1^2 \\ & x_0 + x_1 = 1 \\ & x_0 + 3x_1 \geq 3 \\ & x_1 - x_0 \geq 0 \\ & 0 \leq x_0 \\ & 0 \leq x_1 \end{aligned} \tag{17}$$

设定 dual residual 与 primal residual 的终止误差 $\epsilon_{feas} = 10^{-5}$，设定 surrogate duality gap 的终止误差 $\epsilon = 10^{-5}$。设定计算参数 $t = \theta(m/\eta)$ 时的 $\theta = 16.0$，做 backtracking 时的参数 $\beta = 0.9, \tau = 0.05$。取完全内点的初值为 $x_0 = [0.1, 1.0]$，最终在经历了 7 次迭代之后在精度范围内接近了精确值，精确值为 [0,1]，内置 minimize 的数值结果为 [-1.47e-13,1.0]，实现的 primal-dual 算法结果则如下所示：

迭代次数 k	0	2	4	7
x_k	[0.1,1.0]	[0.01460842 0.99811035]	[2.21422743e-06 1.0]	[3.36383320e-08 1.0]

表 2: test_2 函数实验中内点法迭代后 x_k 变化表格

具体的可视化等高线图如下所示：



图上的星形表示 minimize 的结果，白线就是我的程序从 [0.1,1.0] 出发接近最终结果的过程。

3.3 测试样例三

下面又进行了第三个例子的实验：

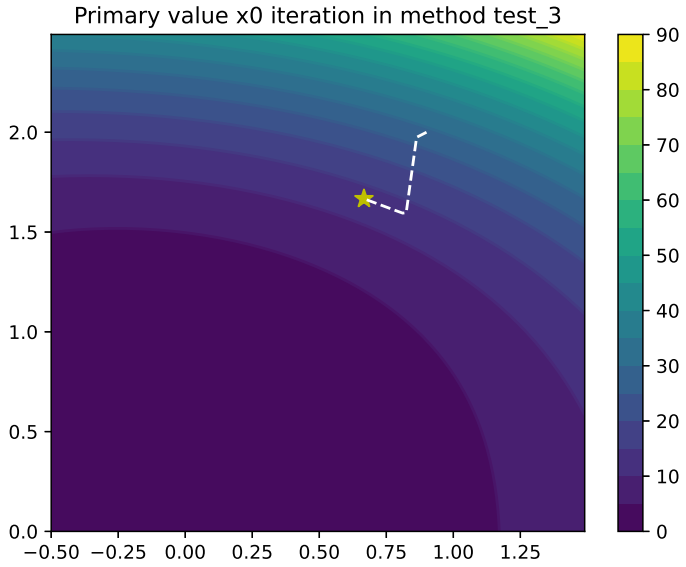
$$\begin{aligned} \underset{x_0,x_1}{min} \quad & 3x_0^2 + x_1^4 + exp(x_0 + x_1) - 2x_0 - 3x_1 \\ & 2x_0 + x_1 = 3 \\ & x_0^2 + x_1^2 \leq 5 \\ & x_0 + 2x_1 \geq 4 \end{aligned} \tag{18}$$

设定 dual residual 与 primal residual 的终止误差 $\epsilon_{feas} = 10^{-5}$，设定 surrogate duality gap 的终止误差 $\epsilon = 10^{-5}$。设定计算参数 $t = \theta(m/\eta)$ 时的 $\theta = 16.0$，做 backtracking 时的参数 $\beta = 0.9, \tau = 0.05$。取完全内点的初值为 $x_0 = [0.9, 2.0]$，最终在经历了 8 次迭代之后在精度范围内接近了精确值，精确值为 [2/3,5/3]，内置 minimize 的数值结果为 [0.66666667 1.66666667]，实现的 primal-dual 算法结果则如下所示：

迭代次数 k	0	3	6	8
x_k	[0.9,2.0]	[0.81094787 1.59479304]	[0.66715369 1.66642765]	[0.6666667 1.66666667]

表 3: test_3 函数实验中内点法迭代后 x_k 变化表格

具体的可视化等高线图如下所示：

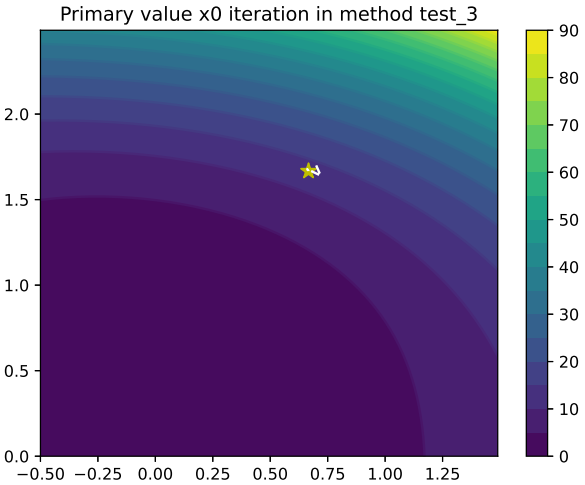


图上的星形表示 minimize 的结果，白线就是我的程序从 [0.9,2.0] 出发接近最终结果的过程。

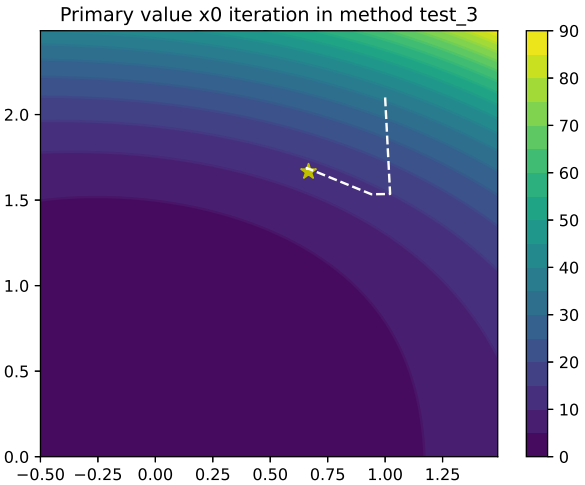
3.4 在测试样例三上进行算法收敛性分析

3.4.1 初始点对迭代的影响

如果换一个距离最终精确点较近的迭代初始点例如 $x_0 = [0.7, 1.7]$ 时，迭代到满足需要的误差处需要循环次数为 6 次，与上面的 8 次相差不多，但确实更快的收敛了，结果如下所示：



如果选超过可行域的初始迭代点，选取初始迭代点为 $x_0 = [1.0, 2.1]$，则最终得到结果如下：



也还是可以收敛到最终的点，因此实际上算法对于初始点是否在可行域内具有一定的忍耐性，但是如果离可行域较远就无法得到结果，例如选取初始点为 $x_0 = [-1.0, 2.5]$ 的时候，算法无法收敛。

3.4.2 几个不同参数对迭代的影响

首先考虑我们在计算 $t = \theta(m/\eta)$ 时的参数 θ，具体变化对应收敛次数如下所示：

θ	1.1	1.5	2.0	4.0	8.0	16.0	32.0
迭代次数	126	33	20	11	9	8	10

表 4: test_3 函数实验中调整变量 θ 后迭代次数变化

可以清楚的发现这个参数对于最终迭代次数影响很大，适当的取大可以极大的加速整个算法的收敛，但是取的太大又会导致迭代次数增加。之所以要取得比较大就是因为 t 只有在趋向于无穷大的时候才会接近初值，因此需要快速的接近更大的量。

其次再考虑做 line search 时取的 β 对迭代次数的影响，此时的 $\theta = 16.0$，初始迭代点为 $x_0 = [0.9, 2.0]$：

β	0.95	0.9	0.8	0.7
迭代次数	8	8	9	11

表 5: test_3 函数实验中调整变量 β 后迭代次数变化

可以发现取 0.9 左右时最优，再往小取的话 α 变化太快，反而使得迭代次数增加。

其次考虑做 line search 时 τ 对迭代次数的影响，此时的 $\theta = 16.0$，初始迭代点为 $x_0 = [0.9, 2.0]$，$\beta = 0.9$：

τ	0.01	0.05	0.08	1.0
迭代次数	8	8	8	无法收敛

表 6: test_3 函数实验中调整变量 τ 后迭代次数变化

因此对于 $\tau \in [0.01, 0.1]$，在小于 0.1 时对迭代次数几乎没有影响，而等于 0.1 时却导致了无法收敛。

四、实验总结

本次实验尝试编写了凸优化算法中的 primal-dual 内点算法，研究了不同参数对于迭代的影响，可以发现对于凸优化问题，使用 primal-dual 算法比 python 内置的 SQP 算法更快。同时对于 KKT 条件的求解方式有了更深一步的了解。