Task 1:

Q1

Find the optimal solution to the following optimization problem:

$\min\limits_{x} x^T Mx + c^T x$

s.t.

$Ax = b$

$M \succ 0$

$A \in \mathbb{R}^{\mathrm{m\times n}}, x \in \mathbb{R}^{\mathrm{n}}, c \in \mathbb{R}^{\mathrm{n}}, b \in \mathbb{R}^{\mathrm{m}}, M \in \mathbb{R}^{\mathrm{n\times n}}$

Solution:

$L(x,\lambda) = x^T Mx + c^T x + \lambda^T (Ax - b)$

$dL(x,\lambda) = \left(dx^T\right) Mx + x^T Mdx + c^T dx + \lambda^T Adx = x^T M^T dx + x^T Mdx + c^T dx + \lambda^T Adx = \left(2x^T M + c^T + \lambda^T A\right) dx = \left(2Mx + c + A^T\lambda\right)^T dx$

$\Rightarrow \nabla_x L(x,\lambda) = 2Mx + A^T\lambda + c$

$\nabla_x L(x,\lambda) = 0 \Rightarrow 2Mx + A^T\lambda + c = 0$

$\Rightarrow \begin{cases} 2Mx + A^T\lambda = -c \\ Ax = b \end{cases} \quad \Rightarrow \begin{cases} x = -\frac{1}{2}M^{-1}c - \frac{1}{2}M^{-1}A^T\lambda \\ Ax = b \end{cases}$

$\Rightarrow b = A\left(-\frac{1}{2}M^{-1}c - \frac{1}{2}M^{-1}A^T\lambda\right) = -\frac{1}{2}AM^{-1}c - \frac{1}{2}AM^{-1}A^T\lambda$

$$\Longrightarrow \lambda = -\left(AM^{-1}A^T\right)^{-1}\left(2b + AM^{-1}c\right)$$

$$\Longrightarrow x = -\frac{1}{2}M^{-1}c + \frac{1}{2}M^{-1}A^T\left(AM^{-1}A^T\right)^{-1}\left(2b + AM^{-1}c\right)$$

Q2

Find the optimal solution to the following optimization problem:

$\min\limits_{x} \|x - c\|_2^2$

s.t.

$Ax = b$

$A \in \mathbb{R}^{\mathrm{m\times n}}, x \in \mathbb{R}^{\mathrm{n}}, c \in \mathbb{R}^{\mathrm{n}}, b \in \mathbb{R}^{\mathrm{m}}$

Solution:

$L\left(x,\lambda\right)=\left\|x-c\right\|_{2}^{2}+\lambda^{T}\left(Ax-b\right)=\left(x-c\right)^{T}\left(x-c\right)+\lambda^{T}\left(Ax-b\right)$

$dL\left(x,\lambda\right)=d\left(x-c\right)^{T}\left(x-c\right)+\left(x-c\right)^{T}d\left(x-c\right)+d\left(\lambda^{T}\left(Ax-b\right)\right)=dx^{T}\left(x-c\right)+\left(x-c\right)^{T}dx+\lambda^{T}\left(Adx-b\right)=\left(2\left(x-c\right)^{T}+\lambda^{T}A\right)dx+\lambda^{T}b=\left(2\left(x-c\right)+A^{T}\lambda\right)^{T}dx+\lambda^{T}b$

$\Rightarrow\nabla L\left(x,\lambda\right)=2\left(x-c\right)+A^{T}\lambda$

$\nabla_{x}L\left(x,\lambda\right)=0\Rightarrow A^{T}\lambda=2\left(c-x\right)$

$\Rightarrow\begin{cases}A^{T}\lambda=2\left(c-x\right)\\ Ax=b\end{cases}\quad\Rightarrow\begin{cases}c-\frac{1}{2}A^{T}\lambda=x\\ Ax=b\end{cases}$

$\Rightarrow b=A\left(c-\frac{1}{2}A^{T}\lambda\right)=Ac-\frac{1}{2}AA^{T}\lambda$

$$\Longrightarrow\lambda=2\left(AA^{T}\right)^{-1}\left(Ac-b\right)$$

$$\Longrightarrow x=c-A^{T}\left(AA^{T}\right)^{-1}\left(Ac-b\right)$$

Q3

Consider the problem:

$\min\limits_{x}\ x^{T}Ax+b^{T}x$

s.t.

$1_{3}^{T}x=1$

$x_{3}\leq1$

$A=\begin{bmatrix}1 & 1 & 0\\ 1 & 2 & 0\\ 0 & 0 & 0\end{bmatrix},x\in\mathbb{R}^{3},1_{3}=\begin{bmatrix}1\\ 1\\ 1\end{bmatrix},b=\begin{bmatrix}1\\ -1\\ 1\end{bmatrix}$

1. Is the problem convex?
2. Find and solve the dual problem.

Solution:

1. Let's find A's eigenvalues:

$$P_A(\lambda) = (1-\lambda)(2-\lambda)(-\lambda) + \lambda$$

$$P_A(\lambda) = -\lambda^3 + 3\lambda^2 - \lambda$$

$$\lambda_1 = 0,\ \lambda_{2,3} = \frac{3 \pm \sqrt{5}}{2}$$

$\Longrightarrow \lambda_i \geq 0 \ \Rightarrow \ A \succeq 0 \ \Rightarrow \ f$ is convex
The constraints are linear $\Longrightarrow$ the problem is convex

2. Lagrangian:

$$L(x,\lambda) = x^T Ax + b^T x + \sum \lambda_i g_i(x) = x^T Ax + b^T x + \lambda_1 \left(1_3^T x - 1\right) + \lambda_2 \left(\begin{pmatrix} 0 & 0 & 1 \end{pmatrix}^T x - 1\right)$$

$$\nabla_x L(x,\lambda) = 2Ax + b + \lambda_1 1_3 + \lambda_2 \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}^T$$

KKT:

$$\nabla_x L(x,\lambda) = 0$$

$$\Rightarrow \begin{pmatrix} 2x_1 + 2x_2 + 1 + \lambda_1 \\ 2x_1 + 4x_2 - 1 + \lambda_1 \\ 1 + \lambda_1 + \lambda_2 \end{pmatrix} = 0$$

$$x_2 = 1$$

$$x_1 = -\frac{3+\lambda_1}{2}$$

$$x_3 = 1 - x_2 - x_1 = -x_1 = \frac{3+\lambda_1}{2}$$

$$\Rightarrow x^* = \begin{pmatrix} -\frac{3+\lambda_1}{2} \\ 1 \\ \frac{3+\lambda_1}{2} \end{pmatrix}$$

The dual function is:

$$q(\lambda) = \min_x L(x,\lambda) = L(x^*,\lambda) = x_1^2 + 2x_1x_2 + 2x_2^2 + x_1 - x_2 + x_3 + \lambda_1(x_1 + x_2 + x_3 - 1) + \lambda_2(x_3 - 1)$$

$$= x_1^2 + 3x_1 + 1 + x_3 + \lambda_1(x_1 + x_3) + \lambda_2(x_3 - 1)$$

$$= x_3^2 + 2x_3 + 1 - \lambda_2(x_3 + 1)$$

$$= \left(\frac{2-\lambda_2}{2}\right)^2 + 2\frac{2-\lambda_2}{2} + 1 - \lambda_2\left(\frac{2-\lambda_2}{2} + 1\right)$$

$$= -\lambda_2^2$$

The dual problem:

$$\max q(\lambda)$$

Solve:

$$q'(\lambda_2) = -2\lambda_2$$

$$q'(\lambda_2) = 0 \Rightarrow \lambda_2 = 0 \Rightarrow \lambda_1 = -1 - 0 = -1$$

Solution:

$$\lambda^* = \begin{pmatrix} -1 \\ 0 \end{pmatrix}$$

$$x^* = \begin{pmatrix} -\frac{3-1}{2} \\ 1 \\ \frac{3+0}{2} \end{pmatrix} = \begin{pmatrix} -1 \\ 1 \\ 1 \end{pmatrix}$$

Q4

Write the dual problem for the following linear programming problem:

$\min\limits_x c^T x$

s.t.

$Ax \leq b$

$x \geq 0$

$A \in \mathbb{R}^{\mathrm{m}\times\mathrm{n}}, x \in \mathbb{R}^{\mathrm{n}}, c \in \mathbb{R}^{\mathrm{n}}, b \in \mathbb{R}^{\mathrm{m}}$

Solution:

Denote:

$$f(x) = c^T x$$

$$g_1(x) = Ax - b \leq 0$$

$$g_2(x) = -x \leq 0$$

Lagrangian:

$$L(x,\lambda) = f(x) + \sum \lambda_i^T g_i(x) = c^T x + \lambda_1^T (Ax - b) - \lambda_2^T x$$

$$\nabla_x L(x,\lambda) = c + A^T \lambda_1 - \lambda_2$$

Find optimum:

$$\nabla_x L(x,\lambda) = 0 \ \Rightarrow \ c + A^T \lambda_1 - \lambda_2 = 0 \ \Rightarrow \ c = \lambda_2 - A^T \lambda_1$$

We see that the condition is not dependant on x.

$$L\left(x^{*},\lambda\right)=\left(\lambda_{2}-A^{T}\lambda_{1}\right)^{T}x+\lambda_{1}^{T}\left(Ax-b\right)-\lambda_{2}^{T}x$$
$$=\left(\lambda_{2}^{T}-\lambda_{1}^{T}A\right)x+\lambda_{1}^{T}\left(Ax-b\right)-\lambda_{2}^{T}x$$
$$=\lambda_{2}^{T}x-\lambda_{1}^{T}Ax+\lambda_{1}^{T}Ax-\lambda_{1}^{T}b-\lambda_{2}^{T}x$$
$$=-\lambda_{1}^{T}b$$

Denote:

$$\eta\left(\lambda\right)=\min_{x}L\left(x,\lambda\right)=\begin{cases}-\lambda_{1}^{T}b & c=\lambda_{2}-A^{T}\lambda_{1}\\ -\infty & otherwise\end{cases}$$

The dual problem:

$$\max_{\lambda\geq0}\eta\left(\lambda\right)=\max_{\lambda\geq0}\left\{-\lambda_{1}^{T}b\right\}$$

s.t.

$$c=\lambda_{2}-A^{T}\lambda_{1}$$

Task 2:

1

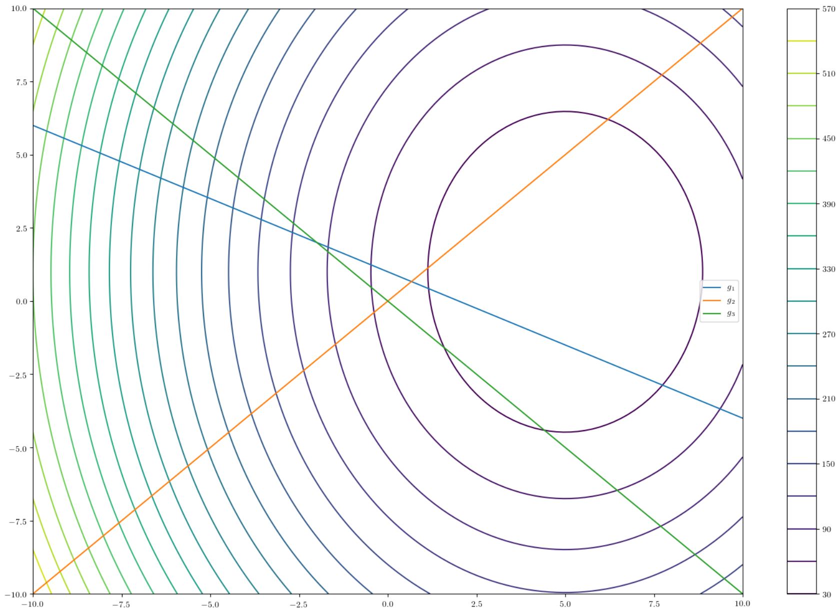
$\min_{x_1,x_2}\left\{2(x_{1}-5)^{2}+(x_{2}-1)^{2}\right\}$

subject to:

$g_{1}\left(x_{1},x_{2}\right)=x_{2}+\frac{x_{1}}{2}-1\leq0$

$g_{2}\left(x_{1},x_{2}\right)=x_{1}-x_{2}\leq0$

$g_{2}\left(x_{1},x_{2}\right)=-x_{1}-x_{2}\leq0$



As we can see in the graph the active constraints are g_1 and g_2.

2

Intersection between g_1 and g_2:

$x_1 + \frac{x_1}{2} - 1 = 0 \Rightarrow x_1 = \frac{2}{3} = x_2$

$\Rightarrow f\left(\frac{2}{3}, \frac{2}{3}\right) = 2\left(\frac{2}{3} - 5\right)^2 + \left(\frac{2}{3} - 1\right)^2 = 37.666$

Intersection between g_1 and g_3:

$-x_1 + \frac{x_1}{2} - 1 = 0 \Rightarrow x_1 = -2 = -x_2$

$\Rightarrow f\left(-2, 2\right) = 2(-2-5)^2 + (2-1)^2 = 99$

Intersection between g_2 and g_3:

$-2x_1 = 0 \Rightarrow x_1 = 0 = x_2$

$\Rightarrow f\left(0, 0\right) = 2(0-5)^2 + (0-1)^2 = 51$

3

$F\left(x_1, x_2, \lambda_1, \lambda_2, \lambda_3\right) = f\left(x_1, x_2\right) + \sum_{i=1}^{3} \lambda_i g_i\left(x_1, x_2\right)$

$= 2(x_1-5)^2 + (x_2-1) + \lambda_1\left(x_2+\frac{x_1}{2}-1\right) + \lambda_2(x_1-x_2) + \lambda_3(-x_1-x_2)$

$$\nabla F = \begin{pmatrix} \frac{\partial F}{\partial x_1} = 4(x_1-5) + \frac{1}{2}\lambda_1 + \lambda_2 - \lambda_3 = 0 \\ \frac{\partial F}{\partial x_2} = 2(x_2-1) + \lambda_1 - \lambda_2 - \lambda_3 = 0 \\ \frac{\partial F}{\partial \lambda_1} = x_2 + \frac{x_1}{2} - 1 \leq 0 \\ \frac{\partial F}{\partial \lambda_2} = x_1 - x_2 \leq 0 \\ \frac{\partial F}{\partial \lambda_3} = -x_1 - x_2 \leq 0 \end{pmatrix} \Rightarrow \begin{matrix} x_1 = -\frac{\lambda_1+2\lambda_2-2\lambda_3}{8}+5 \\ x_2 = -\frac{\lambda_1-\lambda_2-\lambda_3}{2}+1 \end{matrix}$$

$\lambda_1\left(x_2+\frac{x_1}{2}-1\right) = 0$

$\Rightarrow \lambda_1\left(-\frac{\lambda_1-\lambda_2-\lambda_3}{2}+1-\frac{\lambda_1+2\lambda_2-2\lambda_3}{16}+\frac{5}{2}-1\right) = 0$

$\Rightarrow \lambda_1(-8\lambda_1+8\lambda_2+8\lambda_3-\lambda_1-2\lambda_2-2\lambda_3+40) = 0$

$\Rightarrow \lambda_1(-9\lambda_1+6\lambda_2+6\lambda_3+40) = 0$

$\lambda_2(x_1-x_2) = 0$

$\Rightarrow \lambda_2\left(-\frac{\lambda_1+2\lambda_2-2\lambda_3}{8}+5+\frac{\lambda_1-\lambda_2-\lambda_3}{2}-1\right) = 0$

$\Rightarrow \lambda_2(-\lambda_1-2\lambda_2-2\lambda_3+40+4\lambda_1-4\lambda_2-4\lambda_3-8) = 0$

$\Rightarrow \lambda_2(3\lambda_1-6\lambda_2-6\lambda_3+32) = 0$

$\lambda_3(-x_1-x_2) = 0$

$\Rightarrow \lambda_3\left(\frac{\lambda_1+2\lambda_2-2\lambda_3}{8}-5+\frac{\lambda_1-\lambda_2-\lambda_3}{2}-1\right) = 0$

$\Rightarrow \lambda_3(\lambda_1+2\lambda_2+2\lambda_3-40+4\lambda_1-4\lambda_2-4\lambda_3-8) = 0$

$\Rightarrow \lambda_3(5\lambda_1-2\lambda_2-2\lambda_3-48) = 0$

(1) $\lambda_1(-9\lambda_1+6\lambda_2+6\lambda_3+40) = 0$

(2) $\lambda_2(3\lambda_1-6\lambda_2-6\lambda_3+32) = 0$

(3) $\lambda_3(5\lambda_1-2\lambda_2-2\lambda_3-48) = 0$

Because g_3 is an inactive constraint $\underline{\lambda_3 = 0}$ therefore, equations now are:

(1) $-9\lambda_1+6\lambda_2+40 = 0$

(2) $3\lambda_1-6\lambda_2+32 = 0$

$-6\lambda_1 = -72 \Rightarrow \underline{\lambda_1 = 12}$

$6\lambda_2 = 68 \Rightarrow \underline{\lambda_2 = 11\frac{1}{3}}$

In summary, we got:

$\underline{(x_1^*, x_2^*, \lambda_1^*, \lambda_2^*, \lambda_3^*) = \left(\frac{2}{3}, \frac{2}{3}, 12, 11\frac{1}{3}, 0\right)}$

$f(x_1^*, x_2^*) = 2\left(\frac{2}{3}-5\right)^2 + \left(\frac{2}{3}-1\right)^2 = \underline{37\frac{2}{3}}$

4

$\eta\left(\lambda_1,\lambda_2,\lambda_3\right)=F\left(\bar{x}_1,\bar{x}_2,\lambda_1,\lambda_2,\lambda_3\right)$

$=2\left(-\frac{\lambda_1+2\lambda_2+2\lambda_3}{8}\right)^2+\left(-\frac{\lambda_1-\lambda_2-\lambda_3}{2}\right)^2+\lambda_1\left(-9\lambda_1+6\lambda_2+6\lambda_3+40\right)+\lambda_2\left(3\lambda_1-6\lambda_2-6\lambda_3+32\right)+\lambda_3\left(5\lambda_1-2\lambda_2-2\lambda_3-48\right)$

The dual problem:

$\max\limits_{\lambda_i\geq 0}\left\{\eta\left(\lambda_1,\lambda_2,\lambda_3\right)\right\}$

5

$\eta\left(12,11\frac{1}{3},0\right)=2\left(-\frac{12+2\cdot 11\frac{1}{3}}{8}\right)^2+\left(-\frac{12-11\frac{1}{3}}{2}\right)^2+12\left(-9\cdot 12+6\cdot 11\frac{1}{3}+40\right)+11\frac{1}{3}\left(3\cdot 12-6\cdot 11\frac{1}{3}+32\right)$

$=37\frac{5}{9}+\frac{1}{9}+0+0=37\frac{2}{3}$

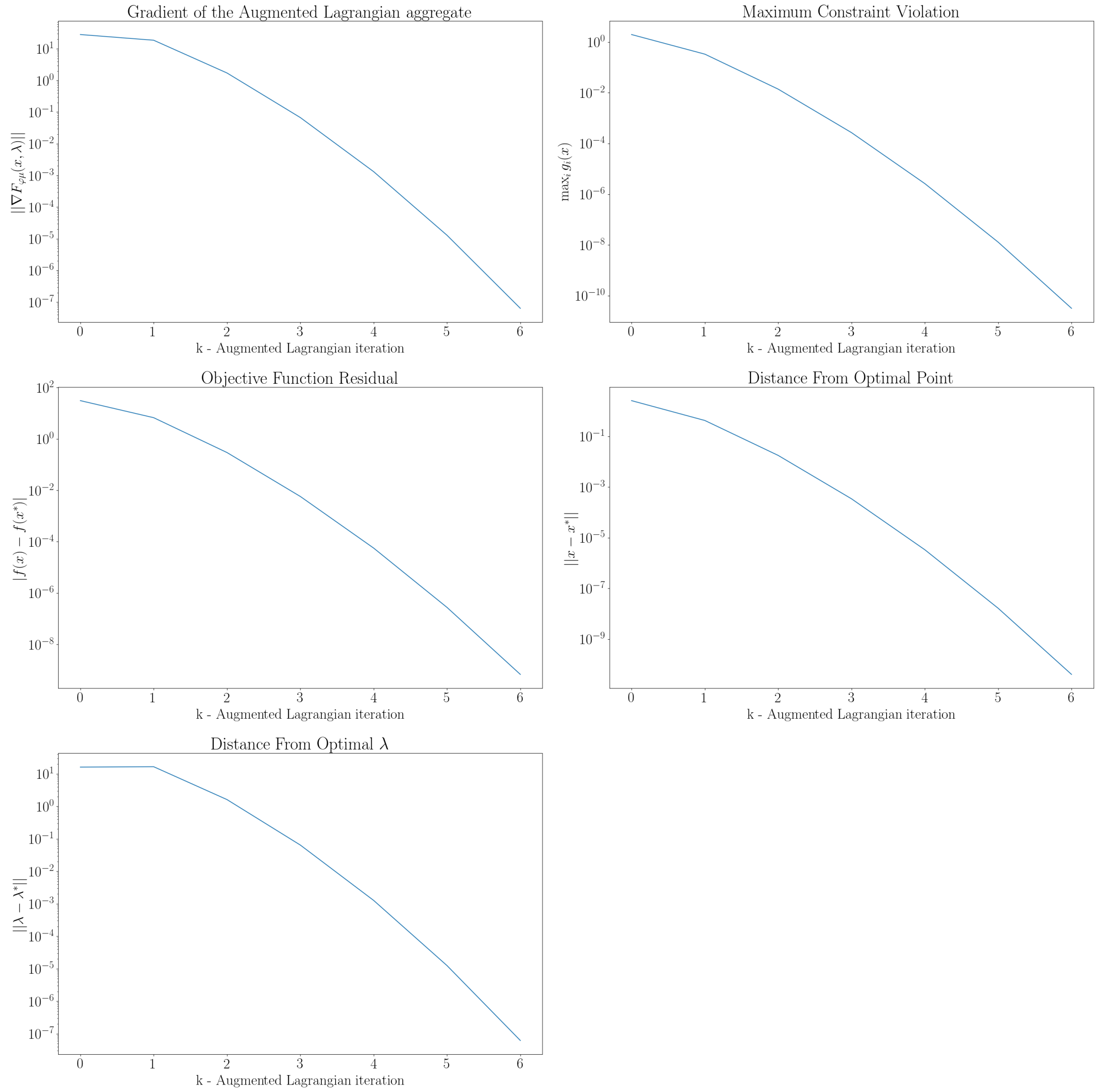
The optimum of the dual problem is achieved, because strong duality conditions hold and therefore the optimum of the primal problem is the optimum of the dual problem.

6 + 7

Implemented in our code.

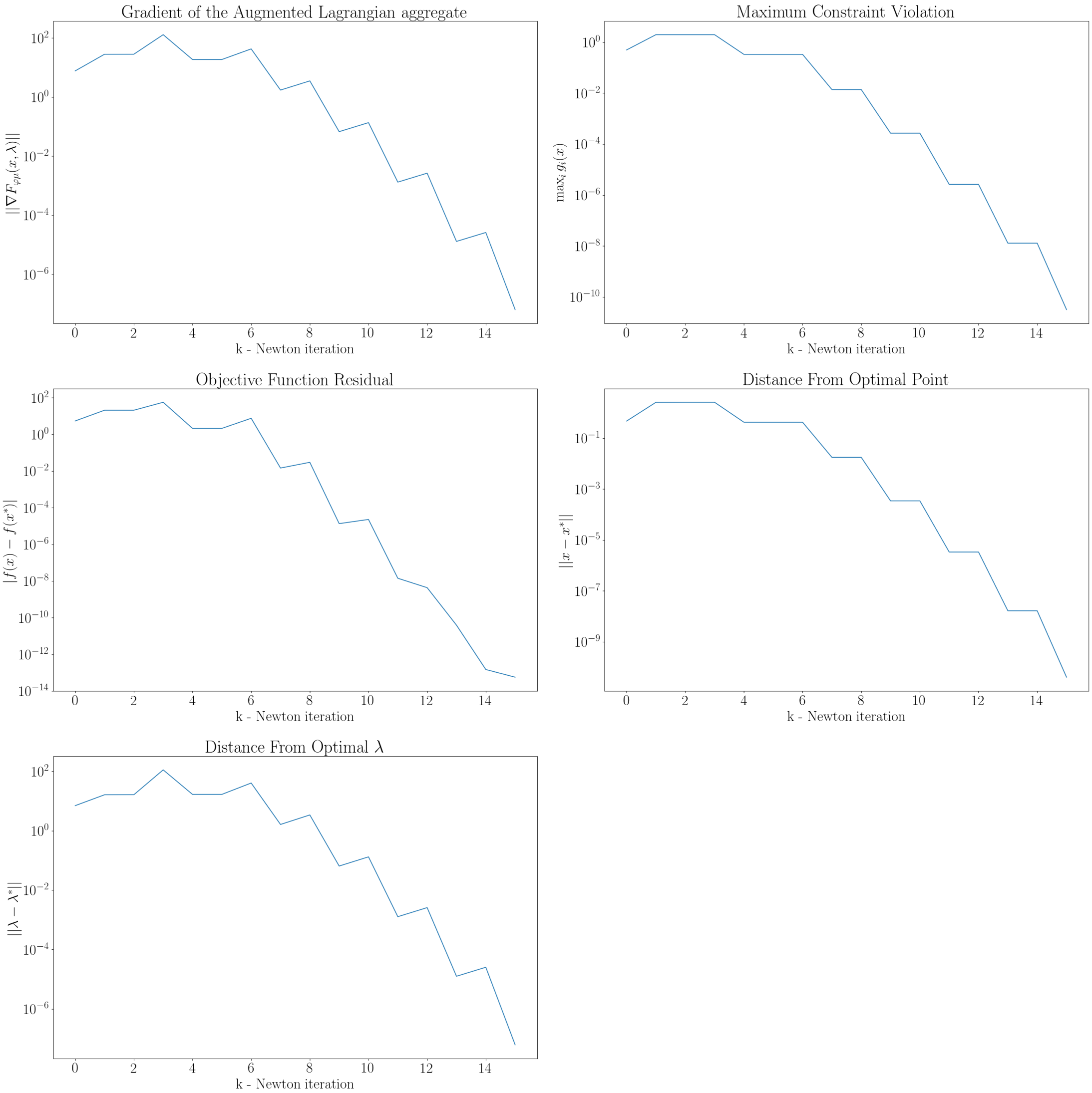
8

As function of outer Augmented Lagrangian



iteration

As function of inner Newton



iteration

HW2.py - For Newton Method wit Armijo line search

```
import numpy as np
import matplotlib.pyplot as plt
from HW2.mcholmz import modified_chol
from scipy.io import loadmat
from functools import partial


def call_foreach(funcs, x):
    try:
        return tuple(map(lambda f: f(x), funcs))
    except TypeError:
        pass
    return funcs(x, 2)


class FunctionAt:
    def __init__(self, x, derivative_sequence):
        self.x = x
        self.der_i_at_x = call_foreach(derivative_sequence, x)


def armijo(f,
           g_x,
           x,
           d,
           alpha=1.,
           beta=0.5,
           sigma=0.25):
    try:
        f0 = f[0]
    except TypeError:
        f0 = lambda v: f(v, 1)
    f_at_x = f0(x)
    df = np.dot(g_x.T, d)

    # comparing (f0(x + alpha * d) - f_at_x) > (sigma * df * alpha) caused a
        numerical bug!!
    while (f0(x + alpha * d) - f_at_x) - (sigma * df * alpha) > np.finfo(np.
        float64).eps:
        # print(f0(x + alpha * d) - f_at_x, ":", sigma * df * alpha)
        alpha *= beta

    return x + alpha * d


def find_newton_direction(f_der_x, _):
    g, H = f_der_x[1:3]
```

```
    L, d, e = modified_chol(H)
    y = substitution(L, -g, direction=FORWARD_SUBSTITUTION)
    z = y.reshape(-1, 1) / d
    return substitution(L.T, z, direction=BACKWARD_SUBSTITUTION).reshape(-1,
        1)


FORWARD_SUBSTITUTION = 1
BACKWARD_SUBSTITUTION = -1


def substitution(L, b, direction=FORWARD_SUBSTITUTION):
    rows = len(L)
    x = np.zeros(rows, dtype=L.dtype)
    row_sequence = reversed(range(rows)) if direction == BACKWARD_SUBSTITUTION
         else range(rows)
    for row in row_sequence:
        delta = b[row] - np.dot(L[row], x)
        cur_x = delta / L[row, row]
        x[row] = cur_x
    return x


def iterative_minimization(get_direction,
                           f_derivatives_sequence,
                           initial_guess=np.zeros((10, 1)),
                           alg_data=None,
                           update_data=None,
                           epsilon=1e-5,
                           return_x_series=False):
    x = initial_guess
    x_history = []
    f_history = []
    f_der_x_prev = None

    while True:
        x_history.append(x)
        f_der_x = FunctionAt(x, f_derivatives_sequence)
        f_history.append(f_der_x.der_i_at_x[0])

        # printing for debug
        # print("#", len(f_history), " f:", f_der_x.der_i_at_x[0], " g:", np.
            linalg.norm(f_der_x.der_i_at_x[1]))

        if update_data:
            alg_data = update_data(alg_data, f_der_x, f_der_x_prev)

        if np.linalg.norm(f_der_x.der_i_at_x[1]) < epsilon:  # ||g(x)|| < e
            break
```

```
        f_der_x_prev = f_der_x

        # Armijo line search
        x = armijo(f=f_derivatives_sequence,
                   g_x=f_der_x.der_i_at_x[1],
                   x=x,
                   d=get_direction(f_der_x.der_i_at_x, alg_data))

    return (x, np.array(f_history)) if not return_x_series else (x, f_history,
        x_history)


def newton_method(f_derivatives_sequence, initial_guess=np.zeros((10, 1)),
    return_x_series=False):
    return iterative_minimization(get_direction=find_newton_direction,
                                  f_derivatives_sequence=
                                      f_derivatives_sequence,
                                  initial_guess=initial_guess,
                                  return_x_series=return_x_series)
```

HW3.py - Augmented Lagrangian

```
import numpy as np
import matplotlib.pyplot as plt
from HW2 import hw2
from functools import partial


def part1():
    def function_f(x1, x2):
        return 2 * (x1 - 5) ** 2 + (x2 - 1) ** 2

    x1 = np.linspace(-10, 10, 100)
    X1, X2 = np.meshgrid(x1, x1)
    vectorized_function = np.vectorize(function_f)
    levels = vectorized_function(X1, X2)
    plt.figure(figsize=(15, 10))
    plt.rc('text', usetex=True)
    plt.rc('font', family='serif')
    c = plt.contour(X1, X2, levels, 20)
    plt.colorbar()
    g1 = np.vectorize(lambda x1: 1 - x1 / 2)(x1)
    g2 = np.vectorize(lambda x1: x1)(x1)
    g3 = np.vectorize(lambda x1: -x1)(x1)
    plt.plot(x1, g1, label=r'$g_1$')
    plt.plot(x1, g2, label=r'$g_2$')
    plt.plot(x1, g3, label=r'$g_3$')
```

```
    plt.legend()
    plt.show()


def phi_f(x: np.ndarray):
    return ((x ** 2) / 2 + x) if x >= -0.5 else (-(np.log(-2 * x)) / 4 - 3 /
        8)


def phi_g(x: np.ndarray):
    return (x + 1) if x >= -0.5 else (-1 / (4 * x))


def phi_H(x: np.ndarray):
    return 1 if x >= -0.5 else (1 / (4 * x ** 2))


def phi_p_mu(x: np.ndarray, p, mu: np.ndarray, G):
    y = 0
    for i in range(G.shape[0]):
        y += mu[i] / p * phi_f(p * (G[i, :2].dot(x) - G[i, 2].reshape(-1, 1)))
    return y
    # return (mu / p).T.dot(np.vectorize(phi_f)(p * (G[:, :2].dot(x) - G[:,
        2].reshape(-1, 1)))).reshape(-1)


def phi_p_mu_grad(x: np.ndarray, p, mu: np.ndarray, G):
    grad = np.zeros((x.size, 1))
    for i in range(G.shape[0]):
        grad += mu[i] * phi_g(p * (G[i, :2].dot(x) - G[i, 2].reshape(-1, 1)))
            * G[i, :2].reshape(-1, 1)
    return grad
    # return (mu * np.vectorize(phi_g)(p * (G[:, :2].dot(x) - G[:, 2].reshape
        (-1, 1)))).T.dot(G[:, :2]).reshape(-1, 1)


def phi_p_mu_hess(x: np.ndarray, p, mu: np.ndarray, G):
    hess = np.zeros((x.size, x.size))
    for i in range(G.shape[0]):
        hess += mu[i] * p * phi_H(p * (G[i, :2].dot(x) - G[i, 2].reshape(-1,
            1))) * G[i, :2].reshape(-1, 1).dot(
             G[i, :2].reshape(-1, 1).T)
    return hess
    # return np.sum(mu * p * np.vectorize(phi_H)(p * (G[:, :2].dot(x) - G[:,
        2].reshape(-1, 1)))  * G[:, :2].T)


def F_p_mu(f, p, mu: np.ndarray, G, nargout=1):
    assert 1 <= nargout <= 3
```

```
    F = lambda x: f[0](x) + partial(phi_p_mu, p=p, mu=mu, G=G)(x)
    if nargout == 1:
        return F

    grad_F = lambda x: f[1](x) + partial(phi_p_mu_grad, p=p, mu=mu, G=G)(x)
    if nargout == 2:
        return F, grad_F

    hess_F = lambda x: f[2](x) + partial(phi_p_mu_hess, p=p, mu=mu, G=G)(x)
    return F, grad_F, hess_F


def quad_f(x, Q, d, e):
    return (0.5 * x.T.dot(Q).dot(x) + d.T.dot(x) + e).reshape(-1)


def quad_g(x, Q, d):
    return Q.dot(x) + d


def quad_h(x, Q):
    return Q


def quad(Q, d, e):
    return (partial(quad_f, Q=Q, d=d, e=e),
            partial(quad_g, Q=Q, d=d, ),
            partial(quad_h, Q=Q))


def augmented_lagrangian(f, G, initial_guess, p=2, P=1e3, alpha=2, epsilon=1e
    -6):
    mu = np.ones((G.shape[0], 1))
    x = initial_guess
    x_series = []
    f_series = []
    lambda_series = []
    max_contraint_violation_series = []
    lagrangian_gradient_series = []
    while p <= P:
        F = F_p_mu(f, p, mu, G, nargout=3)
        x, f_values, x_values = hw2.newton_method(F, x, return_x_series=True)

        # update mu after getting the new x from newton method
        mu = mu * np.vectorize(phi_g)(p * (G[:, :2].dot(x) - G[:, 2].reshape
            (-1, 1)))
        F = F_p_mu(f, p, mu, G, nargout=3)
```

```
        # x_series += x_values
        # f_series += f_values
        x_series.append(x)
        f_series.append(hw2.FunctionAt(x, f).der_i_at_x[0])

        # max_contraint_violation_series += [np.max(G[:, :2].dot(x) - G[:, 2].
            reshape(-1, 1)) for x in x_values]
        max_contraint_violation_series.append(np.max(G[:, :2].dot(x) - G[:,
            2].reshape(-1, 1)))
        lagrangian_gradient_series.append(np.linalg.norm(hw2.FunctionAt(x, F).
            der_i_at_x[1]))
        lambda_series.append(mu * np.vectorize(phi_g)(p * (G[:, :2].dot(x) - G
            [:, 2].reshape(-1, 1))))

        p *= alpha

        # break if ||g(x)|| < epsilon
        if np.linalg.norm(hw2.FunctionAt(x, F).der_i_at_x[1]) < epsilon:
            break
    return x_series, f_series, lambda_series, max_contraint_violation_series,
        lagrangian_gradient_series


def plot_convergence(values, title, xlabel, ylabel, plot_num):
    if plot_num != 0:
        plt.subplot(3, 2, plot_num)
    plt.title(title)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.semilogy(values)


def part2():
    f = quad(Q=np.array([[4, 0], [0, 2]]), d=np.array([[-20], [-2]]), e=51)

    # G contains A & b of the constrains of the form Ax - b <= 0
    # every constrain is a row in which 2 first columns are A and the 3rd (
        last) column is b
    G = np.hstack((np.vstack(([0.5, 1], [1, -1], [-1, -1])), np.vstack((1, 0,
        0))))

    f_opt = 37 + (2 / 3)
    x_opt = np.array([[2 / 3], [2 / 3]])
    lambda_opt = np.array([[12], [11 + (1 / 3)], [0]])

    # initial_guess = np.array([[0.5], [0.5]])
    initial_guess = np.ones((2, 1))

    x_series, f_series, lambda_series, max_constraint_violation_series,
```

```
        lagrangian_gradient_series = augmented_lagrangian(
        f, G, initial_guess=initial_guess)

    plt.figure(figsize=(30, 30))
    plt.rc('text', usetex=True)
    plt.rc('font', family='serif', size=28)

    i = 1
    newton_iter = r'k - Newton iteration'
    augmented_iter = r'k - Augmented Lagrangian iteration'
    for title, xlabel, ylabel, values in (
            (r'Gradient of the Augmented Lagrangian aggregate', augmented_iter
             ,
             r'$||\nabla F_{\varphi\mu}(x,\lambda)||$',
                lagrangian_gradient_series),
            (r'Maximum Constraint Violation', augmented_iter, r'$\max_{i} {g_i
                (x)}$', max_constraint_violation_series),
            (r'Objective Function Residual', augmented_iter, r'$|f(x)-f(x^*)|$
             ',
             [np.abs(f - f_opt).reshape(-1) for f in f_series]),
            (r'Distance From Optimal Point', augmented_iter, r'$||x-x^*||$',
             [np.linalg.norm(x - x_opt) for x in x_series]),
            (r'Distance From Optimal $\lambda$', augmented_iter, r'$||\lambda
                -\lambda^*||$',
             [np.linalg.norm(lambda_i - lambda_opt) for lambda_i in
                lambda_series])
    ):
        plot_convergence(values, title, xlabel, ylabel, i)
        i += 1
    plt.show()


if __name__ == '__main__':
    part1()
    part2()
```