Home assignment 3

Numerical Optimization and its Applications - Spring 2019

Gil Ben Shalom, 301908877
Tom Yaacov, 305578239

June 13, 2019

1 Equality constrained optimization

(a) The optimization problem that is given can be formulated to a single equality constrained optimization problem:

$$\max_{\mathbf{x}\in\mathbb{R}^3} x_1x_2 + x_2x_3 + x_1x_3 \quad s.t. \quad x_1 + x_2 + x_3 - 3 = 0$$

The Lagrangian of this method is

$$\mathcal{L}(\mathbf{x}, \lambda_1) = x_1x_2 + x_2x_3 + x_1x_3 + \lambda_1(x_1 + x_2 + x_3 - 3)$$

and the solution of this problem is given by

$$\nabla\mathcal{L} = 0 \Rightarrow \begin{cases} x_2 + x_3 + \lambda_1 = 0 \\ x_1 + x_3 + \lambda_1 = 0 \\ x_1 + x_2 + \lambda_1 = 0 \\ x_1 + x_2 + x_3 = 0 \end{cases} \quad \Rightarrow \quad \begin{cases} x_1 = 1 \\ x_2 = 1 \\ x_3 = 1 \\ \lambda_1 = -2 \end{cases}$$

(b) In order to show that this critical point is a maximum point we'll first compute the Hessian of $\mathcal{L}$:

$$\nabla^2_{\mathbf{x}}\mathcal{L}(\mathbf{x}, \lambda_1) = \begin{bmatrix} \frac{\partial^2\mathcal{L}}{\partial x_1^2} & \frac{\partial^2\mathcal{L}}{\partial x_1\partial x_2} & \frac{\partial^2\mathcal{L}}{\partial x_1\partial x_3} \\ \frac{\partial^2\mathcal{L}}{\partial x_2\partial x_1} & \frac{\partial^2\mathcal{L}}{\partial x_2^2} & \frac{\partial^2\mathcal{L}}{\partial x_2\partial x_3} \\ \frac{\partial^2\mathcal{L}}{\partial x_3\partial x_1} & \frac{\partial^2\mathcal{L}}{\partial x_3\partial x_2} & \frac{\partial^2\mathcal{L}}{\partial x_3^2} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

Next we'll show that the Hessian of the Lagrangian is negative

$$\mathbf{y}^T\nabla^2_{\mathbf{x}}\mathcal{L}\mathbf{y} < 0 \quad \forall\mathbf{y}\in\mathbb{R}^3 \quad s.t. \quad \mathbf{y}^T\mathbf{1} = 0, \mathbf{y} \neq \mathbf{0}$$

$$\begin{aligned} \mathbf{y}^T\nabla^2_{\mathbf{x}}\mathcal{L}\mathbf{y} &= \begin{bmatrix} y_1 & y_2 & y_3 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \\ &= \begin{bmatrix} y_2 + y_3 & y_1 + y_3 & y_1 + y_2 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \end{aligned}$$

in addition,

$$\mathbf{y}^T\mathbf{1} = 0 \Rightarrow y_1 + y_2 + y_3 = 0 \Rightarrow \begin{bmatrix} y_2 + y_3 \\ y_1 + y_3 \\ y_1 + y_2 \end{bmatrix} = \begin{bmatrix} -y_1 \\ -y_2 \\ -y_3 \end{bmatrix}$$

therefore, we get

$$\begin{aligned} \mathbf{y}^T\nabla^2_{\mathbf{x}}\mathcal{L}\mathbf{y} &= \begin{bmatrix} y_2 + y_3 & y_1 + y_3 & y_1 + y_2 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \\ &= \begin{bmatrix} -y_1 & -y_2 & -y_3 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \\ &= -y_1^2 - y_2^2 - y_3^2 < 0 \qquad\qquad * \mathbf{y} \neq \mathbf{0} \end{aligned}$$

2 General constrained optimization

(a) The Lagrangian of this method is

$$\mathcal{L}(\mathbf{x}, \lambda_1, \lambda_2, \lambda_3) = (x_1+x_2)^2 - 10(x_1+x_2) + \lambda_1(3x_1+x_2-6) + \lambda_2(x_1^2+x_2^2-5) + \lambda_3(-x_1)$$

We will start by activating the first constraint and solve the equation. Meaning, we set $\lambda_2 = \lambda_3 = 0$

$$\nabla\mathcal{L} = 0 \Rightarrow \begin{cases} 2(x_1 + x_2) - 10 + 3\lambda_1 = 0 \\ 2(x_1 + x_2) - 10 + \lambda_1 = 0 \\ 3x_1 + x_2 - 6 = 0 \end{cases} \Rightarrow \begin{cases} 2x_1 + 2x_2 + 3\lambda_1 = 10 \\ 2x_1 + 2x_2 + \lambda_1 = 10 \\ 3x_1 + x_2 = 6 \end{cases}$$

The solution to this linear system is given by

$$x_1 = \frac{1}{2}, x_2 = \frac{9}{2}, \lambda_1 = 0$$

Using these values, we get that the second condition does not hold.

$$0.5^2 + 4.5^2 - 5 = 15 > 0$$

Thus, we need to set the second constraint to be active as well. Thus, only $\lambda_3 = 0$. We get the following equations:

$$\nabla\mathcal{L} = 0 \Rightarrow \begin{cases} 2(x_1 + x_2) - 10 + 3\lambda_1 + 2\lambda_2 x_1 = 0 \\ 2(x_1 + x_2) - 10 + \lambda_1 + 2\lambda_2 x_2 = 0 \\ 3x_1 + x_2 - 6 = 0 \\ x_1^2 + x_2^2 - 5 = 0 \end{cases} \Rightarrow \begin{cases} 2x_1 + 2x_2 + 3\lambda_1 + 2\lambda_2 x_1 = 10 \\ 2x_1 + 2x_2 + \lambda_1 + 2\lambda_2 x_2 = 10 \\ 3x_1 + x_2 = 6 \\ x_1^2 + x_2^2 = 5 \end{cases}$$

The last two equations has 2 solution:

$$x_1 = \frac{18 - \sqrt{14}}{10}, x_2 = \frac{3(2 + \sqrt{14})}{10}$$

$$x_1 = \frac{18 + \sqrt{14}}{10}, x_2 = -\frac{3(-2 + \sqrt{14})}{10}$$

both of the solution holds for all the conditions, but the better solution for the minimization function is when $x_1 = \frac{18-\sqrt{14}}{10} = 1.42, x_2 = \frac{3(2+\sqrt{14})}{10} = 1.72$, We will solve the first two equations to find λ_1 and λ_2:

$$\begin{cases} 2(x_1 + x_2) - 10 + 3\lambda_1 + 2\lambda_2 x_1 = 0 \\ 2(x_1 + x_2) - 10 + \lambda_1 + 2\lambda_2 x_2 = 0 \end{cases} \Rightarrow \begin{cases} 3\lambda_1 + 2.85\lambda_2 = 3.72 \\ \lambda_1 + 3.44\lambda_2 = 3.72 \end{cases}$$

The solution is $\lambda_1 = 0.293, \lambda_2 = 1$ We will validate that all the KKT condition holds:

$$\nabla\mathcal{L}(x^*, \lambda^{eq^*}, \lambda^{ieq^*}) = \begin{bmatrix} 2 * 1.42 + 2 * 1.72 - 10 \\ 2 * 1.42 + 2 * 1.72 - 10 \end{bmatrix} + 0.293 \begin{bmatrix} 3 \\ 1 \end{bmatrix} + 1 \begin{bmatrix} 2 * 1.42 \\ 2 * 1.72 \end{bmatrix} = 0$$

$$c^{eq}(x^*) = 0 :$$

$$3x_1 + x_2 - 6 = 3 * 1.42 + 1.72 - 6 = 0$$

Second constraint is active, and the third not, and indeed $\lambda_2 > 0, \lambda_3 = 0$.

From all the above we can conclude that the KKT condition hold.

(b)

$$f_\mu(x) = (x_1+x_2)^2 - 10(x_1+x_2) + \mu[(3x_1+x_2-6)^2 + (max\{(x_1^2+x_2^2-5), 0\})^2 + (max\{-x_1, 0\})^2]$$

(c) The code that runs the algorithm:

```
from numpy import dot, array, copy, round, array_equal
from random import randint

M_VALUES = [0.01, 0.1, 1.0, 10.0, 100.0]
MAX_ITER = 10000
ALPHA_0 = 1.0
BETA = 0.5
C = 0.0001
X_STAR = [1.42, 1.72]
SEARCH_MAX_ITER = 100


def line_search(f, x, d, gk, alpha0, beta, c, m):
    alpha_j = alpha0
    for j in range(0, SEARCH_MAX_ITER):
        x_temp = x + alpha_j * d
        if f(x_temp, m) <= f(x, m) + alpha_j * c * dot(d, gk):
            break
        else:
```

```
            alpha_j = alpha_j * beta
    return alpha_j


def steepest_decent(f, x_0, max_iter, m):
    x_sd = copy(x_0)
    for k in range(max_iter):
        gk = g(x_sd, m)
        d_sd = -1 * gk
        alpha_sd = line_search(f, x_sd, d_sd, gk, ALPHA_0, BETA, C, m)
        x_last = copy(x_sd)
        x_sd = x_sd + alpha_sd * d_sd
        if array_equal(round(x_sd, 2), round(x_last, 2)):
            break
    return round(x_sd, 2)


def x_func(x, m):
    return pow(x[0] + x[1], 2) - 10 * (x[0] + x[1]) + m * (
           + pow(3 * x[0] + x[1] - 6, 2)
           + pow(max(pow(x[0], 2) + pow(x[1], 2) - 5, 0), 2)
           + pow(max(-1 * x[0], 0), 2)
    )


def x_func_orig(x):
    return round(pow(x[0] + x[1], 2) - 10 * (x[0] + x[1]), 2)


def first_constraint(x):
    return round(3 * x[0] + x[1] - 6, 2)


def second_constraint(x):
    return round(pow(x[0], 2) + pow(x[1], 2) - 5, 2)


def third_constraint(x):
    return round(-1 * x[0], 2)


def check_constraint(x):
    print(f'**** values for x={x}')
    print(f'function value is {pow(x[0] + x[1], 2) - 10 * (x[0] + x[1])}')
    print(f'first constraint is {3 * x[0] + x[1] - 6}')
    print(f'second constraint is {pow(x[0], 2) + pow(x[1], 2) - 5}')
    print(f'third constraint is {-1 * x[0]}')
    print('')


def g(x, m):
    x_g = array([0.0, 0.0])

    x_g[0] = 2 * (x[0] + x[1]) - 10 + m * (
        + 6 * (3 * x[0] + x[1] - 6)
        + 0 if (pow(x[0], 2) + pow(x[1], 2) - 5) <= 0 else 4 * x[0] * (pow(x[0], 2) + pow(x[1], 2) - 5)
        + 0 if (-1 * x[0]) <= 0 else 2 * x[0]
    )

    x_g[1] = 2 * (x[0] + x[1]) - 10 + m * (
        + 2 * (3 * x[0] + x[1] - 6)
        + 0 if (pow(x[0], 2) + pow(x[1], 2) - 5) <= 0 else 4 * x[1] * (pow(x[0], 2) + pow(x[1], 2) - 5)
    )
    return x_g
```

```python
def main():
    x_0 = array([float(randint(0, 9)), float(randint(0, 9))])  # X_STAR
    for _m in M_VALUES:
        x_sol = steepest_decent(x_func, x_0, MAX_ITER, _m)
        print(f'm={_m}, x*={x_sol}, f(x)={x_func_orig(x_sol)}, c1={first_constraint(x_sol)},'
              f' c2={second_constraint(x_sol)}, c3={third_constraint(x_sol)}')


if __name__ == '__main__':
    main()
```

Function and constraint results for each value of μ

```
m=0.01, x*=[2.36 2.31], f(x)=-24.89, c1=3.39, c2=5.91, c3=-2.36
m=0.1, x*=[2.07 1.74], f(x)=-23.58, c1=1.95, c2=2.31, c3=-2.07
m=1.0, x*=[1.61 1.65], f(x)=-21.97, c1=0.48, c2=0.31, c3=-1.61
m=10.0, x*=[1.42 1.74], f(x)=-21.61, c1=0.0, c2=0.04, c3=-1.42
m=100.0, x*=[1.84 0.49], f(x)=-17.87, c1=0.01, c2=-1.37, c3=-1.84
```

3 Box-constrained optimization

(a) The scalar box constrained minimization problem can be formulated to general constrained optimization problem:

$$\min_{x\in\mathbb{R}} \frac{1}{2}hx^2 - gx \quad s.t. \quad \begin{cases} -x + a \leq 0 \\ x - b \leq 0 \end{cases}$$

The Lagrangian of this method is

$$\mathcal{L}(x, \lambda_1, \lambda_2) = \frac{1}{2}hx^2 - gx + \lambda_1(-x + a) + \lambda_2(x - b)$$

We'll compute the first order necessary conditions: Suppose that x^* is a local solution of the problem, than the following conditions hold:

(1) $\nabla_x \mathcal{L}(x^*, \lambda_1^*, \lambda_2^*) = hx^* - g - \lambda_1^* + \lambda_2^* = 0$

(2) $-x^* + a \leq 0$

(3) $x^* - b \leq 0$

(4) $\lambda_1^* \geq 0$

(5) $\lambda_2^* \geq 0$

(6) $\lambda_1^*(-x^* + a) = 0$

(7) $\lambda_2^*(x^* - b) = 0$

To know whether a stationary point is a minimum or a maximum we have the second order necessary conditions for general constrained minimization:

$$\nabla_x^2 \mathcal{L}(x^*, \lambda_1^*, \lambda_2^*) = h > 0$$

therefore

$$y\nabla_x^2\mathcal{L}(x^*,\lambda_1^*,\lambda_2^*)y = yhy > 0 \quad \forall y \neq 0$$

and the stationary point is a minimum.
We can divide our solution to 3 cases:

(a) $a \leq \frac{g}{h} \leq b$:
Solution: $x^* = \frac{g}{h}, \lambda_1^* = 0, \lambda_2^* = 0$ we'll check that the first order necessary conditions hold:
(1) $\nabla_x\mathcal{L}(x^*,\lambda_1^*,\lambda_2^*) = hx^* - g - \lambda_1^* + \lambda_2^* = h\frac{g}{h} - g - 0 + 0 = 0$
(2) $-x^* + a = -\frac{g}{h} + a \leq 0$
(3) $x^* - b = \frac{g}{h} - b \leq 0$
(4) $\lambda_1^* = 0 \geq 0$
(5) $\lambda_2^* = 0 \geq 0$
(6) $\lambda_1^*(-x^* + a) = 0(-\frac{g}{h} + a) = 0$
(7) $\lambda_2^*(x^* - b) = 0(\frac{g}{h} - b) = 0$

(b) $a > \frac{g}{h}$:
Solution: $x^* = a, \lambda_1^* = ha - g, \lambda_2^* = 0$ we'll check that the first order necessary conditions hold:
(1) $\nabla_x\mathcal{L}(x^*,\lambda_1^*,\lambda_2^*) = hx^* - g - \lambda_1^* + \lambda_2^* = ha - g - (ha - g) + 0 = 0$
(2) $-x^* + a = -a + a \leq 0$
(3) $x^* - b = a - b \leq 0$
(4) $\lambda_1^* = ha - g > h\frac{g}{h} - g = 0$
(5) $\lambda_2^* = 0 \geq 0$
(6) $\lambda_1^*(-x^* + a) = (ha - g)(-a + a) = 0$
(7) $\lambda_2^*(x^* - b) = 0(a - b) = 0$

(c) $\frac{g}{h} > b$:
Solution: $x^* = b, \lambda_1^* = 0, \lambda_2^* = -hb + g$ we'll check that the first order necessary conditions hold:
(1) $\nabla_x\mathcal{L}(x^*,\lambda_1^*,\lambda_2^*) = hx^* - g - \lambda_1^* + \lambda_2^* = hb - g - 0 - hb + g = 0$
(2) $-x^* + a = -b + a \leq 0$
(3) $x^* - b = b - b \leq 0$
(4) $\lambda_1^* = 0 \geq 0$
(5) $\lambda_2^* = -hb + g > -h\frac{g}{h} + g = 0$
(6) $\lambda_1^*(-x^* + a) = 0(-b + a) = 0$
(7) $\lambda_2^*(x^* - b) = (-hb + g)(b - b) = 0$

(b) Let the following problem

$$\min_{\mathbf{x}\in\mathbb{R}^n} \frac{1}{2}\mathbf{x}^T H\mathbf{x} - \mathbf{x}^T\mathbf{g} \quad s.t. \quad \mathbf{a} \leq \mathbf{x} \leq \mathbf{b}$$

Therefore, the minimization for each scalar x_i is given by

$$\begin{aligned}
argmin_{x_i \in \mathbb{R}} \frac{1}{2}\mathbf{x}^T H\mathbf{x} - \mathbf{x}^T\mathbf{g} &= argmin_{x_i \in \mathbb{R}} \frac{1}{2}\Big(\sum_i \sum_j x_i x_j h_{i,j}\Big) - \sum_i x_i g_i \\
&= argmin_{x_i \in \mathbb{R}} \frac{1}{2}x_i^2 h_{i,i} + \frac{1}{2}\Big(\sum_{j \neq i} x_i x_j h_{i,j} + x_i x_j h_{j,i}\Big) - x_i g_i \quad * \text{removing consants-} x_j, j \neq i \\
&= argmin_{x_i \in \mathbb{R}} \frac{1}{2}x_i^2 h_{i,i} + \frac{1}{2}2\Big(\sum_{j \neq i} x_i x_j h_{i,j}\Big) - x_i g_i \quad * H \text{ symmetric} \\
&= argmin_{x_i \in \mathbb{R}} \frac{1}{2}x_i^2 h_{i,i} + \Big(\Big(\sum_{j \neq i} x_j h_{i,j}\Big) - g_i\Big)x_i
\end{aligned}$$

In addition, given that the rest are known:

$$\mathbf{a} \leq \mathbf{x} \leq \mathbf{b} \Rightarrow a_i \leq x_i \leq b_i$$

therefore, we get that the minimization for each scalar x_i is given by

$$\min_{x_i \in \mathbb{R}} \frac{1}{2}x_i^2 h_{i,i} + \Big(\Big(\sum_{i \neq j} x_j h_{i,j}\Big) - g_i\Big)x_i \quad s.t. \quad a_i \leq x_i \leq b_i$$

In order to show the expression for the update of projected coordinate descent, we need to define the projection operation with respect to some norm and then set

$$x_i^{(k+1)} = \Pi_\Omega(x_i^{(k)} - \alpha \nabla f(x_i^{(k)}))$$

We will choose the squared l_2 norm. The lagrangian is given by

$$\mathcal{L}(x_i, \lambda_1, \lambda_2) = \frac{1}{2}||x_i - y_i||_2^2 + \lambda_1(-x + a) + \lambda_2(x - b)$$

and its gradient is given by

$$\nabla_{x_i}\mathcal{L}(x_i, \lambda_1, \lambda_2) = x_i - y_i - \lambda_1 + \lambda_2$$

The problem is separable, so if $a_i \leq x_i \leq b_i$, then we can set $x_i^* = y_i$ without breaking the constraint, and hence $\lambda_1^* = \lambda_2^* = 0$, because the constraints are inactive. If $y_i < a_i$, then the lower bound constraint is active and the upper bound is not. We set $x_i^* = a_i$ and

$$x_i^* - y_i - \lambda_1 = 0 \Rightarrow \lambda_1 = a_i - y_i > 0$$

We get a positive Lagrange multiplier, which is what needs to be. If $y_i > b_i$, then the upper bound constraint is active and the lower bound is not. We set $x_i^* = b_i$ and

$$x_i^* - y_i + \lambda_2 = 0 \Rightarrow \lambda_2 = y_i - b_i > 0$$

The gradient is defined by

$$(\nabla f(x^{(k)}))_{(i)} = h_{i,i}x_i + \sum_{j \neq i} h_{i,j}x_j - g_i$$

and the step

$$z_i = x_i^{(k)} - \alpha(\nabla f(x^{(k)}))_{(i)}$$

Overall, the projected steepest descent step is given by:

$$x_i^{(k+1)} = \begin{cases} a_i & z_i < a_i \\ b_i & z_i > b_i \\ z_i & otherwise \end{cases}$$

(c) Following is an implementation for projected coordinate descent algorithm:

```python
from copy import copy
from numpy.linalg import norm
from numpy import matmul


def projected_coordinate_descent(H, g, a, b, x_0, alpha, max_iter, epsilon):
    x = copy(x_0)
    for _ in range(max_iter):
        prev_x = copy(x)
        grad = matmul(H, x) - g
        z = x - alpha * grad
        for i in range(x.shape[0]):
            if z[i] < a[i]:
                x[i] = a[i]
            elif z[i] > b[i]:
                x[i] = b[i]
            else:
                x[i] = z[i]
        if norm(x - prev_x) < epsilon:
            break
    return x


def objective(H, x, g):
    return 0.5 * matmul(matmul(x.T, H), x) - matmul(x.T, g)
```

(d) Running the implementation over the given parameters and outputing the results:

```python
from numpy import array
from py_files.part_3_c import projected_coordinate_descent, objective
from numpy.random import uniform

# parameters setting
H = array([[5, -1, -1, -1, -1],
           [-1, 5, -1, -1, -1],
           [-1, -1, 5, -1, -1],
           [-1, -1, -1, 5, -1],
           [-1, -1, -1, -1, 5]])
g = array([18, 6, -12, -6, 18])
```

```
a = array([0, 0, 0, 0, 0])
b = array([5, 5, 5, 5, 5])

# algorithm parameters setting
epsilon = 1e-4
alpha = 0.1
max_iter = 1000
x_0 = uniform(a, b, a.shape[0])

# running the algorithm
x = projected_coordinate_descent(H, g, a, b, x_0, alpha, max_iter, epsilon)
print(x, r"\\")
print(objective(H, x, g), r"\\")

[ 5. 3.66676844 0.66676844 1.66676844 5. ]
-111.999999953
```

4 Projected Gradient Descent for the LASSO regression

(a)

Lemma 1. *let* $\mathbf{u}, \mathbf{v}$ *be the vectors that minimizes the objective* $||A(\mathbf{u}-\mathbf{v})-\mathbf{b}||_2^2+\lambda(1^T(\mathbf{u}+\mathbf{v})) \Rightarrow \mathbf{u}, \mathbf{v}$ *are from the following form:* $\forall i$ *if* $v_i > 0 \Rightarrow u_i = 0$, *if* $u_i > 0 \Rightarrow v_i = 0$, *else* $u_i = v_i = 0$

Proof. Assume, for the sake of contradiction that there exists $\mathbf{u}$,$\mathbf{v}$ that minimizes the objective $||A(\mathbf{u}-\mathbf{v})-\mathbf{b}||_2^2+\lambda(1^T(\mathbf{u}+\mathbf{v}))$, and $\exists$ i s.t $u_i > 0$ and $v_i > 0$. Let $x = min(u_i, v_i)$. By Setting $u_i' = u_i - x, v_i' = v_i - x$, we keep the value of $\mathbf{u}-\mathbf{v}$ the same, thus the expression $||A(\mathbf{u}-\mathbf{v})-\mathbf{b}||_2^2$ doesn't change as well. But we reduce the value of $\lambda(1^T(\mathbf{u}+\mathbf{v}))$ in $2x\lambda$, and this a contradiction to the minimization of $\mathbf{u}$ and $\mathbf{v}$. □

Lemma 2. *For* $\mathbf{u}, \mathbf{v}$ *in the form we saw in Lemma1* $\Rightarrow$ *let* $\mathbf{x} = \mathbf{u} - \mathbf{v}$, *than* $\lambda(1^T(\mathbf{u}+\mathbf{v})) = \lambda||\mathbf{x}||_1$

Proof.

$$\lambda||\mathbf{x}||_1 = \lambda\sum_{i=1}^{n}|x_i| = \lambda\sum_{i=1}^{n}max\{x_i, -x_i\} = \lambda\sum_{i=1}^{n}max\{u_i - v_i, v_i - u_i\}$$

From Lemma 1 we can conclude that

$$max\{u_i - v_i, v_i - u_i\} = max\{u_i, v_i\} = u_i + v_i$$

Thus:

$$\lambda\sum_{i=1}^{n}max\{u_i - v_i, v_i - u_i\} = \lambda\sum_{i=1}^{n}(u_i + v_i) = \lambda(1^T(\mathbf{u}+\mathbf{v}))$$

□

Now, we proof the claim based on the 2 lemmas.
$\Rightarrow$
Let $\mathbf{x}$ be the vector the minimizes the objective $||A\mathbf{x}-\mathbf{b}||_2^2+\lambda||\mathbf{x}||_1$. We define $\mathbf{u},\mathbf{v}$ s.t $\mathbf{x}=\mathbf{u}-\mathbf{v}$. $\forall$ i if $x_i>0 \Rightarrow u_i=x_i, v_i=0$, if $x_i<0 \Rightarrow v_i=x_i, u_i=0$, else $u_i=v_i=0$.
Claim: The chosen $\mathbf{u},\mathbf{v}$ Minimizes $||A(\mathbf{u}-\mathbf{v})-\mathbf{b}||_2^2+\lambda(1^T(\mathbf{u}+\mathbf{v}))$.
Proof: Assume for the sake of contradiction the there exists $\mathbf{u}',\mathbf{v}'$ s.t $||A(\mathbf{u}'-\mathbf{v}')-\mathbf{b}||_2^2+\lambda(1^T(\mathbf{u}'+\mathbf{v}'))<||A(\mathbf{u}-\mathbf{v})-\mathbf{b}||_2^2+\lambda(1^T(\mathbf{u}+\mathbf{v}))$, than, from Lemma1 and Lemma2, that means we can define $\mathbf{x}'=\mathbf{u}'-\mathbf{v}'$ and get that $||A\mathbf{x}'-\mathbf{b}||_2^2+\lambda||\mathbf{x}'||_1=||A(\mathbf{u}'-\mathbf{v}')-\mathbf{b}||_2^2+\lambda(1^T(\mathbf{u}'+\mathbf{v}'))<||A(\mathbf{u}-\mathbf{v})-\mathbf{b}||_2^2+\lambda(1^T(\mathbf{u}+\mathbf{v}))=||A\mathbf{x}-\mathbf{b}||_2^2+\lambda||\mathbf{x}||_1$ in contradiction to the minimization of $||\mathbf{x}||_1$.
$\Leftarrow$
let $\mathbf{u},\mathbf{v}$ be the vectors that minimize the objective $||A(\mathbf{u}-\mathbf{v})-\mathbf{b}||_2^2+\lambda(1^T(\mathbf{u}+\mathbf{v}))$. let $\mathbf{x}=\mathbf{u}-\mathbf{v}$.
Claim: $\mathbf{x}$ minimizes the objective $||A\mathbf{x}-\mathbf{b}||_2^2+\lambda||\mathbf{x}||_1$.
Proof: Assume for the sake of contradiction that there exists $\mathbf{x}'$ s.t $||A\mathbf{x}'-\mathbf{b}||_2^2+\lambda||\mathbf{x}'||_1<||A\mathbf{x}-\mathbf{b}||_2^2+\lambda||\mathbf{x}||_1$, We can declare $\mathbf{u}',\mathbf{v}'$ as seen in previous proof and get that $||A(\mathbf{u}'-\mathbf{v}')-\mathbf{b}||_2^2+\lambda(1^T(\mathbf{u}'+\mathbf{v}'))=||A\mathbf{x}'-\mathbf{b}||_2^2+\lambda||\mathbf{x}'||_1<||A\mathbf{x}-\mathbf{b}||_2^2+\lambda||\mathbf{x}||_1=||A(\mathbf{u}-\mathbf{v})-\mathbf{b}||_2^2+\lambda(1^T(\mathbf{u}+\mathbf{v}))$ in contradiction to the minimization of $\mathbf{u},\mathbf{v}$.

(b) The code that solves the problem:

```
from numpy import dot, matmul, ones, zeros, copy
from numpy.linalg import norm
import numpy as np

ALPHA_0 = 1.0
BETA = 0.5
C = 0.0001
SEARCH_MAX_ITER = 10
X_SIZE = 200


def get_projection(w):
    return np.vectorize(lambda x: max(x, 0))(w)


def line_search(f, A, b, x, d, gk, alpha0, beta, c, lam):
    alpha_j = alpha0
    for j in range(0, SEARCH_MAX_ITER):
        x_temp = get_projection(x + alpha_j * d)
        if f(A, b, x_temp, lam) <= f(A, b, x, lam) + alpha_j * c * dot(d, gk):
            break
        else:
            alpha_j = alpha_j * beta
    return alpha_j


def steepest_decent(A, b, x_0, lam, max_iter, epsilon):
    objs = list()
    x_sd = copy(x_0)
    for k in range(max_iter):
        gk = g(A, b, x_sd, lam)
        d = -1 * gk
```

```
        alpha = line_search(x_func, A, b, x_sd, d, gk, ALPHA_0, BETA, C, lam)
        x_sd = get_projection(x_sd + alpha * d)
        objs.append(x_func(A, b, x_sd, lam))
        if norm(gk) < epsilon:
            print('****************** converged ************************')
            break
    return x_sd[0:X_SIZE] - x_sd[X_SIZE:2 * X_SIZE], objs


def x_func(A, b, x, lam):
    return norm(matmul(A, x) - b, 2) + lam * dot(ones(x.shape), x)


def g(A, b, x, lam):
    g_x = 2 * matmul(A.transpose(), matmul(A, x) - b) + lam * ones(x.shape)
    return g_x / norm(g_x)
```

(c) We will use the steepest decent algorithm by setting $A' = [A \ -A], \mathbf{x} = [\mathbf{u}\ \mathbf{v}]$. $\lambda = 3$ leads to a solution with approximately 10% non zero values. The code that run the simulation:

```
from numpy import matmul, ones, zeros
from numpy.linalg import norm
import numpy as np
import matplotlib.pyplot as plt
from py_files.part_4_b import steepest_decent

LAM_VALUES = [2.0, 4.0, 8.0, 16.0]
X_SIZE = 200
MAX_ITER = 1000
EPSILON = 0.01


def save_plot(results, x_label, y_label, name):
    plt.figure()
    for lambda_res in results:
        plt.plot(results[lambda_res], label=lambda_res)
    plt.legend()
    plt.xlabel(x_label)
    plt.ylabel(y_label)
    plt.savefig(f'{name}.pdf', bbox_inches="tight")


def main():
    # measurements
    results_objs = dict()

    # init params
    A_orig = np.random.normal(0, 1, (100, X_SIZE))
    x = zeros(X_SIZE)
    indices = np.random.choice(X_SIZE, 20)
    x[indices] = np.random.normal(1, 1, 20)
    b = matmul(A_orig, x) + np.random.normal(0, 0.05, 100)

    # params for sd
    x_0 = np.concatenate((ones(X_SIZE), zeros(X_SIZE)), axis=0)
    A = np.concatenate((A_orig, -A_orig), axis=1)

    for lam in LAM_VALUES:
        print(f'starting sd for lambda={lam}...')
        x_sol, objs = steepest_decent(A, b, x_0, lam, MAX_ITER, EPSILON)
        print(f'lam={lam}, {np.count_nonzero(x_sol)} non zero, norm(x_sol-x)={norm(x_sol-x)}')
        results_objs[f'lam={lam}'] = objs
```

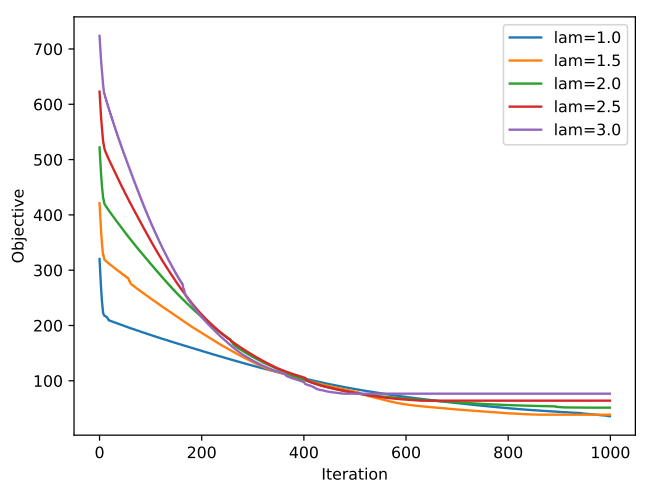
```
    save_plot(results_objs, 'Iteration', 'Objective2', 'Objectives2')


if __name__ == '__main__':
    main()
```

The distance of x from the algorithm, from the original x and number of non-zero values for each lambda:

```
starting sd for lambda=1.0...
lam=1.0, 145 non zero, norm(x_sol − x) = 2.8534896472493636
starting sd for lambda=1.5...
lam=1.5, 39 non zero, norm(x_sol − x) = 0.06859359732580143
starting sd for lambda=2.0...
lam=2.0, 36 non zero, norm(x_sol − x) = 0.07995981722707671
starting sd for lambda=2.5...
lam=2.5, 32 non zero, norm(x_sol − x) = 0.09473570144559718
starting sd for lambda=3.0...
lam=3.0, 33 non zero, norm(x_sol − x) = 0.11751662426815705
```

The objective value of each iteration, for each lambda:



(d) (non-mandatory)

(e) (non-mandatory)