MATH 173B, HW1

Victor Pekkari — epekkari@ucsd.edu

January 23, 2025

Problem 1

A function f (below) is strongly convex iff:

$$f : \Omega \to \mathbb{R}$$
$$\nabla^2 f(x) \succeq \mu I \quad \text{where: } \exists \mu \in \mathbb{R} \quad \text{s.t. } \mu > 0$$

(a)

$$\nabla^2 f(x) = 30 \cdot x^4 \geq 0 \quad \forall x \in \mathbb{R}$$
$$\nabla^2 f(0) = 0$$

Answer: Not Strongly Convex

(b)

$$\nabla^2 f(x) = \frac{e^x + e^{2x}}{1 + e^x} + 2 > 0 \quad \forall x \in \mathbb{R}$$

Answer: Strongly convex

(c)

$$\nabla^2 f(x) = 2e^{x^2} + 4x^2 \cdot e^{x^2} \geq 0 \quad \forall x \in \mathbb{R}$$

Answer: Strongly Convex

(d)

$$\nabla^2 f(x) = \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix} \implies \lambda_1 = 4, \lambda_2 = 0$$

Answer: Not Strongly Convex

Problem 2

(a)

$$f(\vec{x}) = x_1^2 + x_2^2 + x_3^2, \quad \nabla^2 f(\vec{x}) = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad \lambda_{min}(\nabla^2 f(\vec{x})) = 0$$

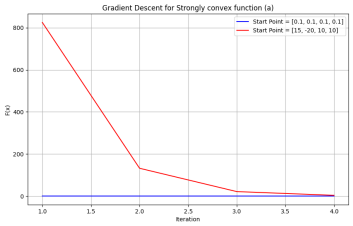
(b)

$$f(\vec{x}) = x_1^2 + x_2^2 + x_3^2 + x_4^2, \quad \nabla^2 f(\vec{x}) = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix}, \quad \lambda_i = 2 > 0 \quad \forall i \in [1, 2, 3, 4]$$

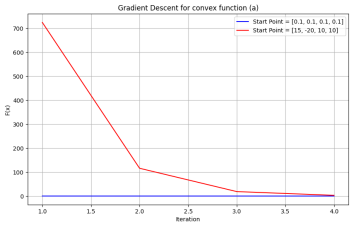
Problem 3

Conclusion: Strong convexity ensures stability which in turn gives us more freedom when choosing the learning rate (had the same lr for gd of the strongly cvx and just cvx this time though).

Strong convexity is furthermore the only thing that ensures exponential convergence, which then makes it reasonable to see that the strongly convex function converges faster.



Convex function (2.a)



Strongly Convex function (2.b)

Code:

```
import numpy as np
import matplotlib.pyplot as plt
from typing import Callable

start1 = np.array([0.1, 0.1, 0.1, 0.1], dtype=float)
start2 = np.array([15, -20, 10, 10], dtype=float)
start_points = [start1, start2]

def cvx(x: np.ndarray) -> float:
    return x[0]**2 + x[1]**2 + x[2]**2

def strongly_cvx(x: np.ndarray) -> float:
    return x[0]**2 + x[1]**2 + x[2]**2 + x[3]**2

def gradient_cvx(x: np.ndarray) -> np.ndarray:
    return np.array([2*x[0], 2*x[1], 2*x[2], 0], dtype=float)

def gradient_strongly_cvx(x: np.ndarray) -> np.ndarray:
    return np.array([2*x[0], 2*x[1], 2*x[2], 2*x[3]], dtype=float)

def gradient_descent(function: Callable[[np.ndarray], float], gradient: Callable[[np.ndarra
    lr = 3e-1
    iteration_nbr = 5
    iterationList = []
    yList = []
    for start_point in start_points:
        x = start_point.astype(float)
        yList.append(function(x))
        iterationList.append(1)
        for i in range(2, iteration_nbr):
            iterationList.append(i)
            x -= lr * gradient(x)
            yList.append(function(x))
    length = len(yList) // 2
    iterationList = iterationList[:length]
    list1 = yList[:length]
    list2 = yList[length:]

    # Plotting list1 and list2
    plt.figure(figsize=(10, 6))
    plt.plot(iterationList, list1, label='Start Point = [0.1, 0.1, 0.1, 0.1]', color='blue'
    plt.plot(iterationList, list2, label='Start Point = [15, -20, 10, 10]', color='red')
    plt.xlabel('Iteration')
```

```
    plt.ylabel('F(x)')
    plt.title('Gradient Descent for Strongly convex function (a)')
    plt.legend()
    plt.grid(True)
    plt.savefig('strong_cvx2.png')
    plt.show()

# Example usage
#gradient_descent(cvx, gradient_cvx)
gradient_descent(strongly_cvx, gradient_strongly_cvx)


# For cvx function, you need to use 4-dimensional start points
# start1 = np.array([0.1, 0.1, 0.1, 0.1], dtype=float)
# start2 = np.array([15, -20, 10, 10], dtype=float)
# start_points = [start1, start2]
# gradient_descent(cvx, gradient_cvx)
```

Problem 4

Coin=c, Heads=h, Tails=t

(a)

Answer: $\Omega = \{[c_1 = h, c_2 = h, c_3 = h], [c_1 = h, c_2 = h, c_3 = t], [c_1 = h, c_2 = t, c_3 = h], [c_1 = h, c_2 = t, c_3 = t], [c_1 = t, c_2 = h, c_3 = h], [c_1 = t, c_2 = h, c_3 = t], [c_1 = t, c_2 = t, c_3 = h], [c_1 = t, c_2 = t, c_3 = t]\}$

(b)

Answer: The event space is the powerset of the sample space; the set of all possible subsets of the sample space

(c)

Answer: $P(event) = \frac{1}{8} \quad \forall event \in \Omega$

(d)

$$X : \Omega \to \mathbb{N}$$

Answer: It takes an element from the sample space ($\omega \in \Omega$) and outputs how many of the coins show heads.

(e)

$\mathbf{p_X(0) = \frac{1}{8}}$

one of our eight events in the sample space fulfills this, when none of the coins show heads.

$\mathbf{p_X(1) = \frac{3}{8}}$

Three of our eight events in the sample space fulfills this, when only coin1, coin2 or coin3 show heads.

$\mathbf{p_X(2) = \frac{3}{8}}$

Three of our eight events in the sample space fulfills this, when only coin1, coin2 or coin3 show tails.

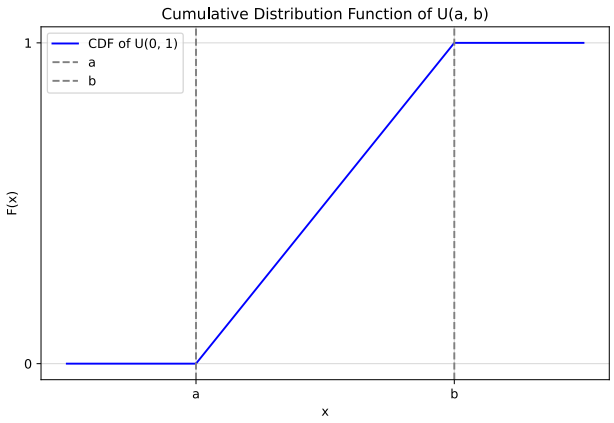
$\mathbf{p_X(3) = \frac{1}{8}}$

One of our eight events in the sample space fulfills this, when all our coins show heads.

Problem 5

Answer:

$$F_X(x) = \begin{cases} 0, & \text{if } x < a, \\ \frac{x-a}{b-a}, & \text{if } a \leq x \leq b, \\ 1, & \text{if } x > b. \end{cases}$$



(b)

Answer: $F_X(\frac{a+b}{2}) = \frac{1}{2}$

(c)

$$f_X(x) = \begin{cases} 1, & \text{for } 0 \leq x \leq 1, \\ 0, & \text{else} \end{cases}$$

$$P(x \in [\frac{1}{3}, \frac{1}{2}]) = \frac{1}{2} - \frac{1}{3} = \frac{1}{6}$$

$$P(x \in [\frac{4}{5}, \frac{5}{6}]) = \frac{5}{6} - \frac{4}{5} = \frac{1}{30}$$

$$P(x \in [\frac{1}{3}, \frac{1}{2}] \cup [\frac{4}{5}, \frac{5}{6}]) = P(x \in [\frac{1}{3}, \frac{1}{2}]) + P(x \in [\frac{4}{5}, \frac{5}{6}])$$

Answer: $P = \frac{1}{5}$.

MATH 173B, HW2

Victor Pekkari — epekkari@ucsd.edu

February 1, 2025

Problem 1

(a)

$$E(X) = \int_{-\infty}^{\infty} x f_X(x)\, dx.$$

Substitute: $x = (x - \mu) + \mu$:

$$E(X) = \int_{-\infty}^{\infty} [(x - \mu) + \mu] f_X(x)\, dx.$$

$$E(X) = \int_{-\infty}^{\infty} (x - \mu) f_X(x)\, dx + \int_{-\infty}^{\infty} \mu f_X(x)\, dx.$$

The first integral evaluates to zero since $(x - \mu)$ is odd and $f_X(x)$ is even:

$$\int_{-\infty}^{\infty} (x - \mu) f_X(x)\, dx = 0.$$

$$\mu \int_{-\infty}^{\infty} f_X(x)\, dx = \mu \cdot 1 = \mu.$$

Thus, we have shown that:

$$E(X) = \mu.$$

(b)

$$\mathrm{Var}(X) = E(X^2) - (E(X))^2.$$

From the expectation proof, we know $E(X) = \mu$

$$\mathrm{Var}(X) = E(X^2) - \mu^2.$$

The expectation $E(X^2)$ is:

$$E(X^2) = \int_{-\infty}^{\infty} x^2 f_X(x)\, dx.$$

Rewriting x^2 as $(x-\mu)^2 + 2\mu(x-\mu) + \mu^2$, we get:

$$E(X^2) = \int_{-\infty}^{\infty} [(x-\mu)^2 + 2\mu(x-\mu) + \mu^2] f_X(x)\, dx.$$

$$E(X^2) = \int_{-\infty}^{\infty} (x-\mu)^2 f_X(x)\, dx + 2\mu \int_{-\infty}^{\infty} (x-\mu) f_X(x)\, dx + \mu^2 \int_{-\infty}^{\infty} f_X(x)\, dx.$$

The second integral is zero (as shown in the expectation proof), and the last integral evaluates to 1, so:

$$E(X^2) = \int_{-\infty}^{\infty} (x-\mu)^2 f_X(x)\, dx + \mu^2 = \sigma^2 \cdot \int_{-\infty}^{\infty} f_X(x)\, dx + \mu^2$$

(By definition, $\sigma = x - \mu$)

$$E(X^2) = \sigma^2 + \mu^2.$$

Substituting into the variance formula:

$$\mathrm{Var}(X) = (\sigma^2 + \mu^2) - \mu^2 = \sigma^2.$$

Problem 2

(a)

A valid CDF must satisfy:

(1) Monotonicity: $G_Z(z)$ must be non-decreasing.

$$G'_Z(z) = \frac{e^{-z}}{(1+e^{-z})^2} > 0, \quad \forall z \in \mathbb{R}.$$

(2) Limits: the $G(Z) \xrightarrow[z\to\infty]{} 1$ and $G(Z) \xrightarrow[z\to-\infty]{} 0$

$$\lim_{z\to-\infty} G_Z(z) = \frac{1}{1+e^{x}} = 0, \quad \lim_{z\to\infty} G_Z(z) = \frac{1}{1+e^{-x}} = 1.$$

(3) Right-continuity: A function that is continous is also right-continous:

A function $f(x)$ is continuous on the set Ω at a point $x = a$ if:

$$\lim_{x\to a} f(x) = f(a) \quad \forall a \in \Omega \tag{1}$$

To check the continuity of $\sigma(x)$, we evaluate:

$$\lim_{x\to a} \frac{1}{1+e^{-x}} \tag{2}$$

The exponential function e^{-x} is well defined for all real numbers, and the denominator is never going to be zero. We can therefore conclude that the sigmoid-function is contious for all real numbers.

it follows that:

$$\lim_{x \to a} \sigma(x) = \sigma(a), \quad \forall a \in \mathbb{R}.$$

Conclusion: Since everything holds for $G_Z(z)$, it is a valid CDF

(b)

$y = 1$ when $z \geq -a$

$y = -1$ when $z < -a$

Using the definition of a CDF:

$$\mathbb{P}(y = 1) = \mathbb{P}(z \geq -a) = 1 - G_Z(-a)$$

$$\mathbb{P}(y = -1) = G_Z(-a).$$

Since $G_Z(-a) = \frac{1}{1+e^a}$, we get:
Answer:

$$\mathbb{P}(y = 1) = \frac{e^a}{1 + e^a}$$

$$\mathbb{P}(y = -1) = \frac{1}{1 + e^a}$$

(c)

From part (b), we can express the probabilities as:

$$\mathbb{P}(y = 1) = \frac{1}{1 + e^{-a}}, \quad \mathbb{P}(y = -1) = \frac{1}{1 + e^a}.$$

For general $y \in \{-1, 1\}$ we can combine the two functions above as follows:

$$p(y) = \frac{1}{1 + e^{-ay}}$$

(d)

We can rewrite the joint distribution as the following since $y_1, ..., y_n$ are indenpendent:

$$p(y_1, y_2, \ldots, y_N) = \prod_{i=1}^{N} p(y_i) = \prod_{i=1}^{N} \frac{1}{1 + e^{-a_i y_i}}.$$

Problem 3

(a)

Answer: It makes sense to maximize $H(w)$ since by maximizing $H(w)$ we want to maximize every term $\frac{1}{1+e^{-w^T x_i y_i}}$ in the product notation $\prod_{i=1}^{N}$.

To maximize the terms $\frac{1}{1+e^{-w^T x_i y_i}}$ we have to minimize their denominator which mean we want $e^{-w^T x_i y_i}$ to be as small as possible. To keep it small we want to keep the exponent of e negative, $-w^T x_i y_i < 0$ or:

$$w^T x_i y_i > 0 \tag{*}$$

By viewing x_i as different inputs and y_i as our different targets. It is clear that to keep (*) greater than 0 we have to make :

$$sign(w^T x_i) = sign(y_i)$$

and we also want the magnitude of: w^T to be as big as possible. This means that for every training example x_i, the decision boundary defined by w^T should correctly classify y_i by ensuring that $w^T x_i$ has the same sign as y_i. When this condition holds, the probability assigned to the correct class label is high.
Additionally, maximizing $w^T x_i y_i$ not only ensures correct classification but also increases confidence in the prediction. A larger magnitude of $w^T x_i$ results in values closer to 1 in the sigmoid function:

$$\frac{1}{1+e^{-w^T x_i y_i}}$$

pushing the probability of correct classification towards 1.

Since $H(w)$ is the product of these probabilities across all training samples, maximizing $H(w)$ is equivalent to maximizing the likelihood of the observed labels under the logistic regression model. This aligns with the principle of maximum likelihood estimation (MLE), which seeks to find the parameter w that best explains the observed data.

(b)

$$log\left[H(w)\right] = \log\left[\prod_{i=1}^{N} \frac{1}{1+e^{-w^T x_i y_i}}\right]$$

$$= \sum_{i=1}^{N} \log\left[\frac{1}{1+e^{-w^T x_i y_i}}\right]$$

(c)

Maximizing the LHS side is the same as minimizing the RHS, and minimizing the RHS is the same as minimizing $F(w)$ since the only difference is a factor of $\frac{1}{N}$

$$\sum_{i=1}^{N} \log\left[\frac{1}{1+e^{-w^T x_i y_i}}\right] = -\sum_{i=1}^{N} \log\left[1+e^{-w^T x_i y_i}\right]$$

(d)

Answer: To derive an SGD algorithm for minimizing $F(w)$, we first derive the gradient of the objective function:

$$\nabla F(w) = \frac{1}{N}\sum_{i=1}^{N} \nabla \log\left(1+e^{-w^T x_i y_i}\right).$$

The gradient of the individual term is:

$$\nabla \log\left(1+e^{-w^T x_i y_i}\right) = -\frac{y_i x_i e^{-w^T x_i y_i}}{1+e^{-w^T x_i y_i}} = -y_i x_i\left(1-\frac{1}{1+e^{-w^T x_i y_i}}\right).$$

Below is the stochastic gradient descent update step, using index i_t uniformly independently chosen from $\{1, 2, ...N\}$ where N is the number of points. η is the constant step size.
(A constant step size in SGD won't make us converge fully to an optimizer)

$$w_{t+1} = w_t - \eta \nabla f_{i_t}(w_t),$$

where

$$f_{i_t}(w_t) = \log\left(1+e^{-w^T x_i y_i}\right)$$

$$\implies \nabla f_{i_t}(w_t) = -y_{i_t} x_{i_t}\left(1-\frac{1}{1+e^{-w_t^T x_{i_t} y_{i_t}}}\right).$$

(e)

$$F(w) = \sum_{i=1}^{N} f_{i_t}(w) \qquad (*)$$

$$(*) \implies \mathbb{E}\nabla f_{i_t}(w) = \frac{1}{N}\cdot\sum_{i=1}^{N} \nabla f_{i_t}(w) = \nabla F(w)$$

$F(w)$ and f_{i_t}as computed in (d):

$$\nabla F(w) = \sum_{i=1}^{N} -y_{i_t} x_{i_t} \left(1 - \frac{1}{1 + e^{-w_t^T x_{i_t} y_{i_t}}}\right)$$

$$\nabla f_{i_t}(w_t) = -y_{i_t} x_{i_t} \left(1 - \frac{1}{1 + e^{-w_t^T x_{i_t} y_{i_t}}}\right)$$

MATH173B, HW3

Victor Pekkari — epekkari@ucsd.edu

February 8, 2025

Problem 1

SGD with constant step size $\alpha = 10^{-5}$.

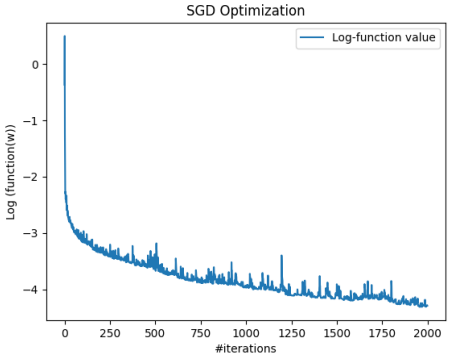


Figure 1: batch size = 30

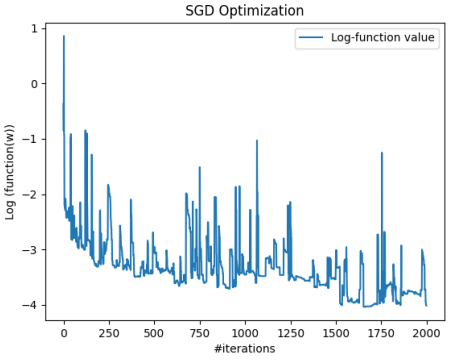


Figure 2: batch size = 5

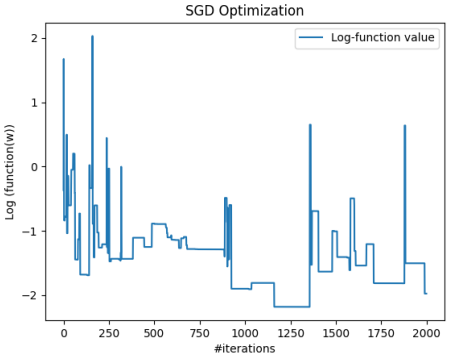


Figure 3: batch size = 1

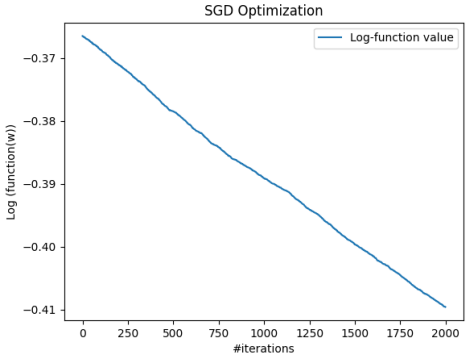


Figure 4: batch size = 1, with normalized input

Comment: (on batch size=1, unnormalized input)

- The log function value $log\,[F(w)]$ does not decrease at every iteration. This is because we don't update our weight in the direction of the negative gradient, we can therefore not guarantee descent in every iteration.
- This SGD fails to converge to a fixed w^*, likely because the gradients we use to update w are too irregular.

These results are very consistent with the theory we have learned in class. We learned that "on average" the function should decrease, but it is not guaranteed to decrease at every iteration.

The variance of $g_j(w) = \left[\frac{1}{m}\sum_{k=1}^{m} f_{i_k}(w_t)\right]_j$ decreases as the batch size increases in the plots above which makes sense:

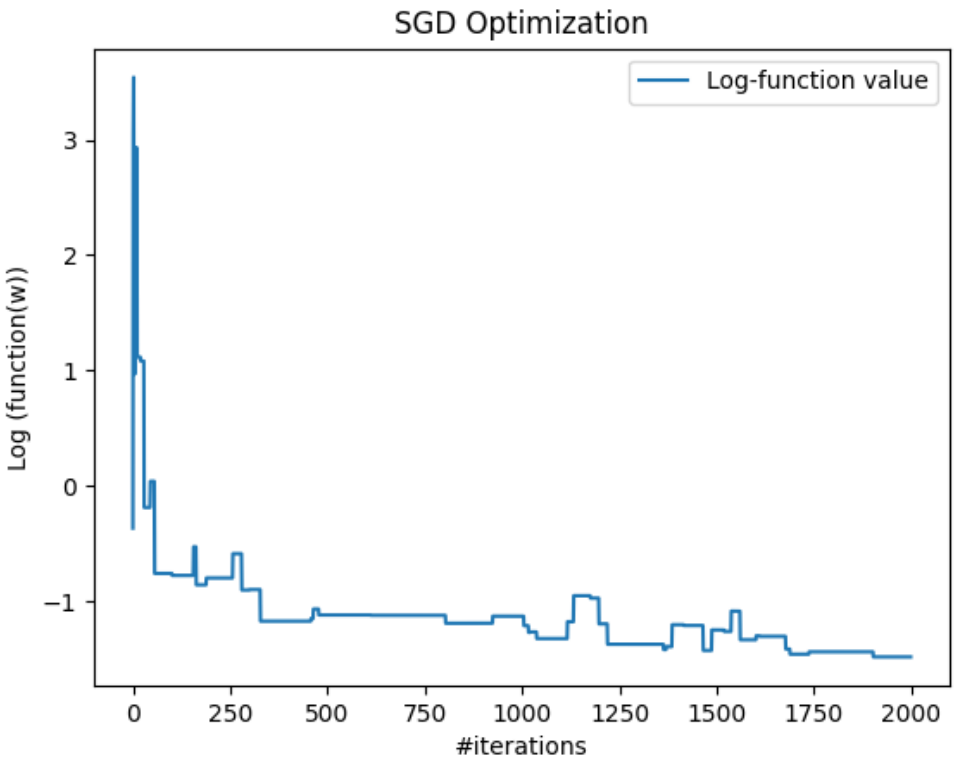
$$\begin{aligned} Var(g_j(x)) &= \mathbb{E}\left[g_j(x)^2\right] - \left(\frac{\partial F}{\partial x_j}(x)\right)^2 \\ &= \frac{1}{m}\mathbb{E}\left[\left(\frac{\partial f_{i_k}}{\partial x_j}(x)\right)^2\right] + \frac{m-1}{m}\left(\frac{\partial F}{\partial x_j}(x)\right)^2 - \left(\frac{\partial F}{\partial x_j}(x)\right)^2 \\ &= \frac{1}{m}\mathbb{E}\left[\left(\frac{\partial f_{i_k}}{\partial x_j}(x)\right)^2\right] - \frac{1}{m}\left(\frac{\partial F}{\partial x_j}(x)\right)^2 \\ &= \frac{1}{m}\cdot\frac{1}{N}\sum_{i=1}^{N}\left(\frac{\partial f_i}{\partial x_j}(x)\right)^2 - \frac{1}{m}\left(\frac{\partial F}{\partial x_j}(x)\right)^2. \end{aligned}$$

(c)

Answer: Error rate was 2% on the test-data and 1.5% on the training-data.

Problem 2

SGD with batch size = 1, and step size: $\alpha_t = 10^{-4} \cdot \sqrt{\frac{1}{1+t}}$



Comment:

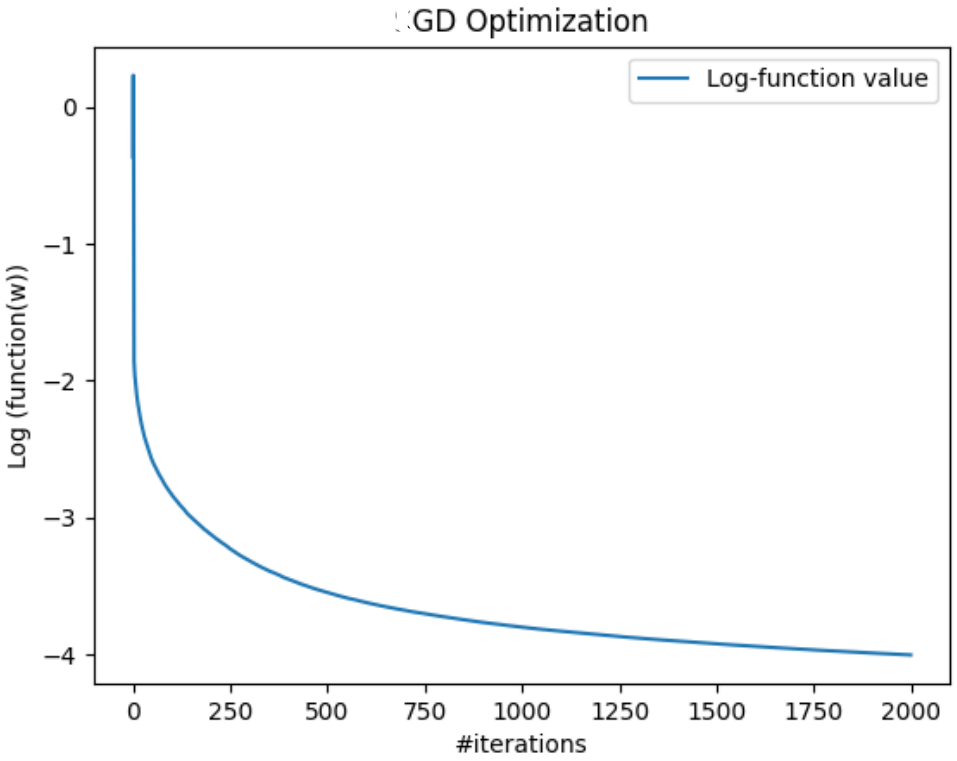
- The log function value $log\,[F(w)]$ does not decrease at every iteration.
- w seems to converge to a fixed w^* as we can see a decreasing function value in the plot, despite fluctuations. This is probably because we decrease our step size after each iteration. Since our step size goes to zero, our updates will converge to zero as well, which means that w will converge to a fixed value to, we just have to hope we are close to a "good" w when we start converging.

(c)

Answer: Error rate was 1% on the test-data and 1.4% on the training-data.

Problem 3

GD with step size: $\alpha_t = 10^{-4} \cdot \sqrt{\frac{1}{1+t}}$



Comment:

- The log function value $log\,[F(w)]$ now decreases at every iteration. That is because we always update the weight in the direction of the negative gradient, unlike in SGD when that is not a guarantee.
- w seems to converge to a fixed w^* as we can in all plots see a decreasing function value, despite fluctuations.
- It is possible to see how there are smaller and smaller changes to the value of the function. Likely due to the decrease of the step size and the gradient

Answer: Error rate was 0.06% on the test-data and 0.05% on the training-data.

Problem 4

(a)

Answer: Say we have a problem where we want to run T number of iterations, we have d number of features for $w \in R^d$, and we have N number of datapoints.

SGD(batch size=1): here we have a time complexity of

$$O(T \cdot d)$$

to run the algorithm. Since we have to do T iterations, and every iteration we have to calculate the gradient, to calculate the gradient we have N terms (data-points) to compute with, and every term has d dimensions.

GD: here we have a time complexity of

$$O(T \cdot N \cdot d)$$

to run the algorithm. Since we have to do T iterations, and every iteration we have to calculate the gradient, to calculate the gradient we have 1 term (data-points) to compute, and every this term has d dimensions.

Conclusion: The big upside with SGD compared to GD computationally, is that SGD becomes a lot faster when N is huge,

(b & c)

Answer: (b)

(1): $F(w) = 0.1563$, (2): $F(w) = 0.2448$, (3): $F(w) = 0.01792$

Gradient descent (3) got closest to the minimizer, it does it at a much more expensive computational cost however. The decay in step size might not let SGD with adaptive step size converge in time.

Comment: Gradient descent got the highest accuracy on both the training data-set and the test-set. SGD with adaptive step size can achieve better accuracy but a worse function value because accuracy depends only on correct classification (a discrete measure), while loss also depends on prediction confidence (a continuous measure). As the adaptive step size shrinks, updates become too small to fully minimize the loss, even if the decision boundary is already well-positioned for high accuracy. In contrast, SGD with a constant step size continues optimizing loss but may not generalize as well.

It is very likely that the SGD with adaptive step size would have overtaken the other SGD in loss function value if we only ran it for more iterations, since a smaller step size requiers us to do more updates.

Python code for question 1,2 and 3:

```python
import math
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

csv_file = "mnist_train.csv"
test_csv = "mnist_test.csv"
data = pd.read_csv(csv_file)

labels = data.iloc[:, 0].to_numpy()
pixels = data.iloc[:, 1:].to_numpy()
ones_column = np.ones((pixels.shape[0], 1))
pixels = np.hstack((pixels, ones_column))
normalize = False

mask = (labels == 1) | (labels == 2)
X = pixels[mask].astype(float)
y = labels[mask]

y = np.where(y == 1, 1, -1)

if normalize:
    X /= 255.0

X_1 = X[y == 1][:2000]
X_2 = X[y == -1][:2000]
y_1 = y[y == 1][:2000]
y_2 = y[y == -1][:2000]

X = np.concatenate((X_1, X_2), axis=0)
y = np.concatenate((y_1, y_2), axis=0)


lr = 1e-5

N = 4000

# 1a
# w = sgd(lr='a', save_fig=False, nbr_iterations=2000)

# 2a
# w = sgd(lr='b', save_fig=False, nbr_iterations=2000)

# 3a
# w = sgd(lr='b', save_fig=True, nbr_iterations=2000, batch_size=4000)

# print(function(weights=w))
# print(classify(w=w))


def gradient(weights:np.ndarray[float], batch_size:int=1) -> np.ndarray[float]:
    indices = np.random.choice(np.arange(0, 4000), size=batch_size, replace=True)
    grad = np.zeros_like(weights)
    for index in indices:
        x_multiplied_y = (-y[index]) * X[index]
        denominator = 1 + np.exp(-np.dot(weights, X[index]) * y[index])
        grad += (1 / batch_size) * x_multiplied_y * (1 - (1 / denominator))
    return grad

def load_test_data():
    test_data = pd.read_csv(test_csv)
    test_labels = test_data.iloc[:, 0].to_numpy()
    test_pixels = test_data.iloc[:, 1:].to_numpy()
```

```
64
65      ones_column = np.ones((test_pixels.shape[0], 1))
66      test_pixels = np.hstack((test_pixels, ones_column))
67
68      test_mask = (test_labels == 1) | (test_labels == 2)
69      test_X = test_pixels[test_mask].astype(float)[:500]
70
71      test_y = test_labels[test_mask][:500]
72
73      test_y = np.where(test_y == 1, 1, -1)
74
75      if normalize:
76          test_X /= 255.0
77      return test_X, test_y
78
79  def classify(w):
80      x_test, y_test = load_test_data()
81      print("test_data: ", classify_test(x_test, y_test, weights=w))
82      print("training_data: ", classify_test( X, y, weights=w))
83
84  def classify_test(test_X, test_y, weights:np.ndarray[float]) -> float:
85
86      predictions = np.dot(test_X, weights)
87      predicted_labels = np.sign(predictions)
88      summation = 0
89      for index in range(test_y.shape[0]):
90          if predicted_labels[index] == test_y[index]:
91              summation += 1
92      return summation / test_y.shape[0]
93
94  def log_function(weights:np.ndarray[float]) -> float:
95      return np.log(function(weights))
96
97  def function(weights:np.ndarray[float]) -> float:
98      #return (1 / N) * np.sum(np.log(1 + np.exp(-y[:N] * np.dot(X[:N], weights))))
99      return (1 / N) * np.sum(np.log(1 + np.exp(-y[i] * np.dot(weights, X[i])))) for i in range(N))
100
101 def learning_rate(kind, t) -> float:
102     if kind == 'a':
103         return 1e-5
104     else:
105         return 1e-4 * np.sqrt(1/(1+t))
106
107 def sgd(lr:str, save_fig:bool=False, nbr_iterations:int = 2000, batch_size=1) -> None:
108     weights = np.zeros(X.shape[1])
109     function_list = []
110     iteration_list = []
111     weigh_list = []
112     for i in range(nbr_iterations):
113         iteration_list.append(i)
114         function_list.append(log_function(weights))
115         weights -= learning_rate(kind=lr, t=i) * gradient(weights, batch_size=batch_size)
116
117     if save_fig:
118         plt.plot(iteration_list, function_list, label='Log-function value')
119         plt.xlabel('#iterations')
120         plt.ylabel('Log (function(w))')
121         plt.title('SGD Optimization')
122         plt.legend()
123         plt.savefig('sgd_optimization_plot2.png')
124         plt.show()
125     print(function_list[-1])
126     return weights
```

MATH 173B, HW4

Victor Pekkari — epekkari@ucsd.edu

February 20, 2025

Problem 1

By definition of the dual function:

$$F(u_i) \leq f(x) + u_i^T g(x) \quad \forall x \in \mathbb{R}^n \tag{*}$$

Let $u = \alpha u_1 + (1-\alpha)u_2$ where $\alpha \in [0,1], \quad u_1, u_2 \in \mathbb{R}^m, m, n \in \mathbb{N}$,
and x^* be the minimizer of $L(x,u)$

$$F(u) = L(x^*, u) = f(x^*) + u^T g(x^*) = f(x^*) + \alpha \cdot u_1^T g(x^*) + (1-\alpha) \cdot u_2^T g(x^*)$$

$$F(u) = \left[\alpha \cdot f(x^*) + u_1^T g(x^*)\right] + \left[(1-\alpha) \cdot f(x^*) + u_2^T g(x^*)\right]$$

$$(*) \implies F(u) \geq [\alpha F(u_1)] + [(1-\alpha)F(u_2)]$$

Problem 2

(a)

$\mathcal{L}[x, \lambda, v] = x_1 + \frac{1}{2}(x_1^2 + 4x_2^2) - \lambda_1 \cdot x_1 - \lambda_2 \cdot x_2 + v_1 \cdot (x_1 + 2x_2 - 1)$

$\mathcal{L}[x, \lambda, v] = (1 + v_1 - \lambda_1)x_1 + (2v_1 - \lambda_2)x_2 + \frac{1}{2}(x_1^2 + 4x_2^2) - v_1$

(b)

Since the optimizer is convex in x (linear + second degree is convex) we can set the gradient in x equal to zero.

$$\nabla \mathcal{L}(x, \lambda, v) = \begin{bmatrix} (1 + v_1 - \lambda_1) + x_1 \\ (2v_1 - \lambda_2) + 4x_2 \end{bmatrix}$$

minimizer x^* is:

$$x_1 = \lambda_1 - 1 - v_1$$
$$x_2 = \frac{1}{4}(\lambda_2 - 2v_1)$$

Putting back into $\mathcal{L}(x, \lambda, v)$ yields:

$$F(\lambda, v) = (1+v_1-\lambda_1)\cdot(\lambda_1-v_1-1)+(2v_1-\lambda_2)\cdot\frac{1}{4}(\lambda_2-2v_1)-v_1+\frac{1}{2}(\lambda_1^2-2\lambda_1-2\lambda_1 v_1+1+2v_1+v_1^2)+\frac{1}{8}(\lambda_2^2-4\lambda_2 v_1+v_1^2$$

(c)

Using the function from part (b)

$$\max_{\lambda, v} F(\lambda, v) \quad \text{s.t.} \quad \lambda \geq 0.$$

Problem 3

(a)

Introducing the dual variables $\lambda \geq 0$ for the inequality constraints, the Lagrangian function is:

$$\mathcal{L}(x, \lambda) = \|x\|^2 + \lambda^T(Ax - b).$$

(b)

Taking the gradient with respect to x and setting it to zero:

$$\nabla_x \mathcal{L}(x, \lambda) = 2x + A^T\lambda = 0 \quad \Rightarrow \quad x^* = -\frac{1}{2}A^T\lambda.$$

Substituting x^* into $\mathcal{L}(x, \lambda)$:

$$F(\lambda) = -\frac{1}{4}\lambda^T AA^T\lambda - \lambda^T b.$$

(c)

The dual problem is given by:

$$\max_{\lambda \geq 0}\left(-\frac{1}{4}\lambda^T AA^T\lambda - \lambda^T b\right) \quad s.t. \lambda_i \geq 0 \quad \forall i \in \{1, 2, .., m\}$$

Problem 4

$$\mathcal{L}(x, y, \lambda, v) = \frac{1}{2}||y||^2 + v^T(Ax - b - y)$$

(b)

To find the dual function, we minimize the Lagrangian with respect to x and y:
Minimize with respect to y:

$$\frac{\partial L}{\partial y} = y - v = 0 \quad \Rightarrow \quad y = v.$$

Substitute $y = v$ into the Lagrangian:

$$L(x, v, \lambda, v) = \frac{1}{2}\|v\|^2 + v^T(Ax - b - v).$$

We end up with:

$$L(x, v, \lambda, v) = -\frac{1}{2}\|v\|^2 + v^T(Ax - b).$$

Minimize with respect to x:

$$\inf_x v^T Ax = \begin{cases} 0, & \text{if } A^T v = 0, \\ -\infty, & \text{otherwise.} \end{cases}$$

Answer:

$$F(\lambda, v) = \begin{cases} -\frac{1}{2}\|v\|^2 - v^T b, & \text{if } A^T v = 0, \\ -\infty, & \text{otherwise.} \end{cases}$$

(c)

The dual optimization problem is:

$$\max_{v \in \mathbb{R}^m} \quad -\frac{1}{2}\|v\|^2 - v^T b \quad \text{subject to} \quad A^T v = 0.$$

Problem 5

(a)

The Lagrangian associated with this problem is given by:

$$L(x, \lambda, v) = \frac{1}{2}x^T Qx + c^T x + \lambda^T(b - Ax), \quad \lambda \geq 0$$

(b)

$$\frac{\partial L}{\partial x} = Qx + c - A^T\lambda = 0 \quad \Rightarrow \quad x = -Q^{-1}(c - A^T\lambda).$$

$$L(-Q^{-1}(c - A^T\lambda), \lambda, v) = \frac{1}{2}(c - A^T\lambda)^T Q^{-1}(c - A^T\lambda) - \lambda^T b.$$

Answer: Lagrangian dual function is:

$$F(v, \lambda) = -\frac{1}{2}(c - A^T\lambda)^T Q^{-1}(c - A^T\lambda) - \lambda^T b.$$

(c)

$$\max_{\lambda \geq 0} \quad -\frac{1}{2}(c - A^T\lambda)^T Q^{-1}(c - A^T\lambda) - \lambda^T b.$$

MATH 173B, HW5

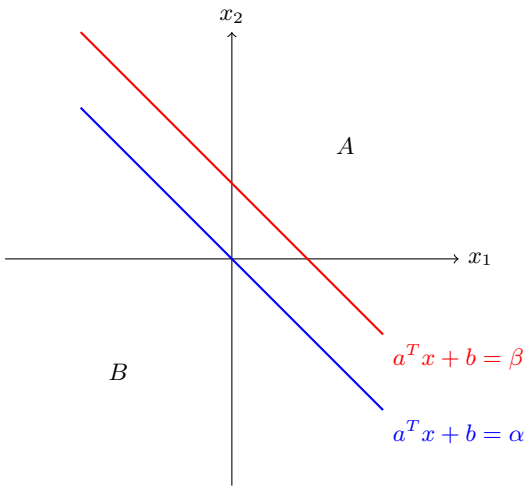
Victor Pekkari — epekkari@ucsd.edu

March 1, 2025

Problem 1

(a)

The A-region is above (including) the red line, and the B-region is beneath (including) the blue line.



(b)

We assume the point $z \in B$ and $z + \lambda a \in A$.

$$a^T z + b = \alpha \tag{1}$$

$$a^T (z + \lambda a) + b = \beta \tag{2}$$

Expanding (2) $\implies$

$$a^T z + \lambda a^T a + b = \beta$$

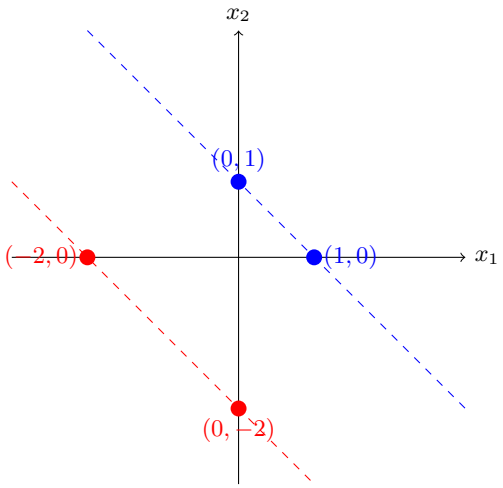
$$\alpha + \lambda \|a\|^2 = \beta \qquad (\alpha = a^T z + b)$$

$$\lambda = \frac{\beta - \alpha}{\|a\|^2}$$

The distance between the two hyperplanes is the length of the array we had to add to point $z \in B$ to get to hyperplane A.

$$w = \|\lambda a\| = \left\| \frac{\beta - \alpha}{\|a\|^2} a \right\| = \frac{\beta - \alpha}{\|a\|}$$

Problem 2



$$A = \{(1,0),(0,1)\}, \quad B = \{(-2,0),(0,-2)\}$$

(a)

Primal SVM optimization problem:

$$\min_{w,b} \frac{\|w\|^2}{\|\beta - \alpha\|^2} \quad s.t. \begin{cases} \beta - w^T x - b \leq 0, & \forall x \in \{(0,1),(1,0)\} \\ w^T x + b - \alpha \leq 0, & \forall x \in \{(-2,0),(0,-2)\} \end{cases}$$

Dual SVM optimization problem:

$$\mathcal{L}(w,b,\lambda,\mu) = \frac{\|w\|^2}{\|\beta - \alpha\|^2} + \lambda_1 \left(\beta - \left(w^T(0,1) + b\right)\right) + \lambda_2 \left(\beta - \left(w^T(1,0) + b\right)\right)$$
$$+\mu_1 \left(w^T(-2,0) + b - \alpha\right) + \mu_2 \left(w^T(0,-2) + b - \alpha\right)$$
$$\mathcal{L}(w,b,\lambda,\mu) = \frac{\|w\|^2}{\|\beta - \alpha\|^2} + \sum_{x \in A} \lambda_i(\beta - w^T x - b) + \sum_{x \in B} \mu_i(w^T + b - \alpha)$$

$$\mathcal{L}(w,b,\lambda,\mu) = \frac{\|w\|^2}{\|\beta-\alpha\|^2} + \langle a, \sum_{x\in B}\mu_x\cdot x - \sum_{x\in A}\lambda_x\cdot x\rangle + \langle b, \sum_{x\in B}\mu_x - \sum_{x\in A}\lambda_x\rangle + \sum_{x\in B}\mu_x - \sum_{x\in A}\lambda_x$$

$$F(\lambda,\mu) = \min_{w,b}\mathcal{L}(w,b,\lambda,\mu) = \mathcal{L}(w^*,b^*,\lambda,\mu)$$

Find $F(\lambda,\mu) = \min_{w,b}\mathcal{L}(w,b,\lambda,\mu)$

Dual optimization problem: $\max_{\lambda,\mu} F(\lambda,\mu)$

$$\max_{\lambda,\mu}\frac{\|w^*\|^2}{\|\beta-\alpha\|^2} + \langle w^{*T}, \sum_{x\in A}\mu_x\cdot x - \sum_{x\in B}\lambda_x\cdot x\rangle + \langle b^*, \sum_{x\in A}\mu_x - \sum_{x\in B}\lambda_x\rangle + \sum_{x\in A}\mu_x - \sum_{x\in B}\lambda_x$$

where w^*, b^* are the minimizers for $\mathcal{L}$
Subject to $\lambda,\mu \geq 0$ and $\sum_i \lambda_i = \sum_j \mu_j$

(b)

We have the primal:

$$\mathcal{L}(w,b,\lambda,\mu) = \frac{\|w\|^2}{\|\beta-\alpha\|^2} + \langle a, \sum_{x\in A}\mu_x\cdot x - \sum_{x\in B}\lambda_x\cdot x\rangle + \langle b, \sum_{x\in A}\mu_x - \sum_{x\in B}\lambda_x\rangle + \sum_{x\in A}\mu_x - \sum_{x\in B}\lambda_x$$

Since the primal is convex with respect to w, b, taking its gradient with respect to w, b will yield the minimizer.

$$\nabla_w\mathcal{L} = \frac{2w}{\|\beta-\alpha\|^2} + \sum_{x\in A}\mu_x\cdot x - \sum_{x\in B}\lambda_x\cdot x$$

$$\nabla_b\mathcal{L} = \sum_{x\in A}\mu_x - \sum_{x\in B}\lambda_x$$

Thus, we obtain the system:

$$\begin{cases} \frac{2w}{\|\beta-\alpha\|^2} + \sum_{x\in A}\mu_x\cdot x - \sum_{x\in B}\lambda_x\cdot x & = 0 \\ \sum_{x\in A}\mu_x - \sum_{x\in B}\lambda_x & = 0 \end{cases}$$

Rearranging,

$$\frac{2w}{\|\beta-\alpha\|^2} + \sum_{x\in A}\mu_x\cdot x - \sum_{x\in B}\lambda_x\cdot x + \sum_{x\in A}\mu_x - \sum_{x\in B}\lambda_x = 0$$

$$\frac{2w}{\|\beta-\alpha\|^2} + \sum_{x\in A}(1+x)\cdot\mu_x - \sum_{x\in B}(1+x)\cdot\lambda_x = 0$$

From this, solving for w leads to:

$$w = \frac{\|\beta - \alpha\|^2}{2}\left(\sum_{x\in B}(1+x)\cdot\lambda_x - \sum_{x\in A}(1+x)\cdot\mu_x\right)$$

Finally, the optimal separating hyperplane equation is:

$$w^T x + b = 0 \implies b = \sum_{x\in B}\lambda_x - \sum_{x\in A}\mu_x$$

Answer: This means that the optimal hyperplane to use as a separator is (put in the points if you want to get the numeric value for w and b):

$$\frac{\|\beta - \alpha\|^2}{2}\left(\sum_{x\in B}(1+x)\cdot\lambda_x - \sum_{x\in A}(1+x)\cdot\mu_x\right)^T x + \sum_{x\in B}\lambda_x - \sum_{x\in A}\mu_x = 0.$$

(c)

Answer: There is no unique pair of closest points between the two convex hulls of A and B.

Dual problem: Since Slater's condition holds for this problem (like we proved in class), the optimal value of the original probelm is the same as the optimal value of the dual problem. This is like said in clase the other day; the closest two point between set A and B, where the points are computed through a weighted average. One of the two points is computed through a weighted average of the points in A, and the other one is computed through a weighted average of the points in B.

$$\min_{\lambda,\nu} \|\sum_{x\in B}\lambda_x\cdot x - \sum_{x\in A}\nu_x\cdot x\|^2 \quad s.t. \begin{cases} \lambda & \geq 0 \\ \nu & \geq 0 \\ \sum_{x\in B}\lambda_x\cdot x & = \sum_{x\in A}\nu_x\cdot x = 1 \end{cases}$$

If we insert our points into the equation above we get that we **can't** find two unique closest points between the convex hulls of A and B. We get all pairwise-closest points on between the red and blue line in the graph above.

Problem 3

(a)

Define the Lagrangian function as:

$$\mathcal{L}(x_1, x_2, \lambda, \nu_1, \nu_2) = f(x) + \sum_{i}^{m}\lambda_i\cdot g_i(x) + \sum_{i}^{P}\nu_i\cdot h_i(x)$$

where:

$$f(x) = 2x_2 + (x_1^2 + x_2^2)$$
$$h(x) = x_1 + x_2 - 1$$
$$g_1(x) = -x_1, \quad g_2(x) = -x_2$$

All functions above are convex hence setting the gradients equal to zero will give us the minimizer:

1. Stationarity:

$$\nabla_{x_1}\mathcal{L} = 2x_1 + \nu - \lambda_1 = 0 \implies x_1^* = \frac{1}{2}(\lambda_1 - \nu)$$

$$\nabla_{x_2}\mathcal{L} = 2x_2 + 2 + \nu - \lambda_2 = 0 \implies x_2^* = \frac{1}{2}(\lambda_2 - 2 - \nu)$$

2. Primal Feasibility:

$$x_1 + x_2 = 1, \quad x_1 \geq 0, \quad x_2 \geq 0$$

3. Dual Feasibility:

$$\lambda_1 \geq 0, \quad \lambda_2 \geq 0$$

4. Complementary Slackness:

$$\lambda_1 x_1 = 0, \quad \lambda_2 x_2 = 0$$

(b)

Using complementary slackness tells us that if $x_i > 0 \implies \lambda_i = 0$:
Assume $x_1 > 0$ and $x_2 > 0$, then $\lambda_1 = 0$ and $\lambda_2 = 0$. Using equations from points (1):

$$2x_1 + \nu = 0$$
$$2 + 2x_2 + \nu = 0$$

From (3), $x_1 + x_2 = 1$. Solving for x_1, x_2, ν:

$$\nu = -2x_1, \quad \nu = -2 - 2x_2$$

$$-2x_1 = -2 - 2x_2 \Rightarrow 2x_1 = 2 + 2x_2 \Rightarrow x_1 - x_2 = 1$$

$$\begin{cases} x_1 + x_2 = 1 \\ x_1 - x_2 = 1 \end{cases} \Rightarrow x_1 = 1, \quad x_2 = 0$$

Then, $\nu = -2(1) = -2$

(c)

The found point $(x_1, x_2) = (1, 0)$ satisfies the KKT conditions. This means it does solve the original optimization problem

Problem 4

(a)

$$\mathcal{L}(x, \lambda, \mu) = c^T x + \mu^T (Ax - b) - \lambda^T x,$$

The KKT conditions are:

1. **Stationarity:** The gradient of the Lagrangian must be zero:

$$\nabla_x \mathcal{L} = c + A^T \lambda - \mu = 0.$$

2. **Primal feasibility:** The constraints must hold:

$$Ax - b = 0, \quad x \geq 0.$$

$$-x_i \leq 0 \quad \forall i \in \{1, ..., n\}$$

3. **Dual feasibility:** The multipliers must be non-negative:

$$\lambda \geq 0.$$

4. **Complementary slackness:** Each component must satisfy:

$$-\lambda_i x_i = 0, \quad \forall i \in \{1, \ldots, n\}$$

(b)

Answer: Yes, if a primal feasibe point x^* and a dual feasible (λ^*, ν^*) satisifes the KKT-conditions, x^* will be optimal for the linear optimization problem.

Problem 5

Inital note: The Lagrangian is convex, and Dual feasibility holds since there are no inequality constraints in the problems

$$\mathcal{L}(x, y, \lambda, \nu) = \|x\|^2 + \mu(a^T x - b) \tag{1}$$

The gradient with respect to x must be 0:

$$\nabla_x \mathcal{L} = 2x + \mu a \implies x = -\frac{1}{2}\mu a$$

The constraint must be satisfied for primal feasibilty to hold:

$$a^T \left[-\frac{1}{2}\mu a \right] = b$$

Solving for λ gives:

$$\lambda = -\frac{2b}{\|a\|^2}$$

Computing the minimizer x^*

$$x^* = -\frac{1}{2}\left[-\frac{2b}{\|a\|^2}\right]a \implies x^* = \frac{b}{\|a\|^2} \cdot a$$

Note: The KKT-conditions now hold for the optimization problem

Interpretation: x^* is the closest point in the hyperplane $a^T x = b$ to the origin.

MATH 173B, HW6

Victor Pekkari — epekkari@ucsd.edu

March 8, 2025

Problem 1

(a)

$$\mathcal{L}(x, \lambda, \nu) = \left[x_1^2 + 2x_2^2\right] + \lambda_1(4 - x_1 - 2x_2) - \lambda_2(x_1) - \lambda_3(x_2)$$

$$= x_1^2 + 2x_2^2 - (\lambda_1 + \lambda_2)x_1 - (2\lambda_1 + \lambda_3)x_2 + 4\lambda_1$$

(b)

Since the function is convex in x_1, x_2, setting the gradient equal to zero will give us the minimizer.

$$\nabla_{x_1}\mathcal{L} = 2x_1 - \lambda_1 - \lambda_2 = 0 \implies x_1^* = \frac{1}{2}(\lambda_1 + \lambda_2)$$

$$\nabla_{x_2}\mathcal{L} = 4x_2 - 2\lambda_1 - \lambda_3 = 0 \implies x_2^* = \frac{1}{4}(2\lambda_1 + \lambda_3)$$

Lagrangian dual function $F(\lambda, \nu)$ is:

$$F(\lambda, \nu) = \min_x \mathcal{L}(x, \lambda, \nu) = \mathcal{L}(x^*, \lambda, \nu)$$

Putting in x_1^*, x_2^* into $\mathcal{L}$ yields the Lagrangian dual function:

$$F(\lambda, \nu) = \frac{1}{4}(\lambda_1 + \lambda_2)^2 + \frac{1}{8}(2\lambda_1 + \lambda_3)^2 - \frac{1}{2}(\lambda_1 + \lambda_2)^2 - \frac{1}{4}(2\lambda_1 + \lambda_3)^2 + 4\lambda_1$$

$$= -\frac{1}{2}(\lambda_1 + \lambda_2)^2 - \frac{1}{4}(2\lambda_1 + \lambda_3)^2 + 4\lambda_1$$

(c)

If the Hessian of $F(\lambda, \nu)$ is negative semidefinite the function is concave:

$$\frac{\partial F}{\partial \lambda_1} = -(\lambda_1 + \lambda_2) - \frac{1}{2}(2\lambda_1 + \lambda_3) + 4$$

$$\frac{\partial F}{\partial \lambda_2} = -(\lambda_1 + \lambda_2)$$

$$\frac{\partial F}{\partial \lambda_3} = -\frac{1}{2}(2\lambda_1 + \lambda_3)$$

We compute the Hessian matrix:

$$H = \begin{bmatrix} -2 & -1 & -\frac{1}{2} \\ -1 & -1 & 0 \\ -1 & 0 & -\frac{1}{2} \end{bmatrix} \implies \lambda(H) = \{0, \frac{1}{4}(\sqrt{17} - 7), \frac{1}{4}(-7 - \sqrt{17})\}$$

Conclusion: We can see that all eigenvalues of the hessian are less than or equal to zero, This means that the Hessian is Negative Semidefinite. The NSD Hessian implies that the function $F(\lambda, \nu)$ is concave.

(d)

Since the Dual function is convex in λ, ν we can take it's gradient and set it equal to zero

$$\frac{\partial F}{\partial \lambda_1} = -(\lambda_1 + \lambda_2) - \frac{1}{2}(2\lambda_1 + \lambda_3) + 4 = 0$$

$$\frac{\partial F}{\partial \lambda_2} = -(\lambda_1 + \lambda_2) = 0$$

$$\frac{\partial F}{\partial \lambda_3} = -\frac{1}{2}(2\lambda_1 + \lambda_3) = 0$$

The equations above give us the following linear system to solve for λ_i:

$$\begin{cases} -2\lambda - \lambda_2 - \frac{1}{2}\lambda_3 + 4 &= 0 \\ -\lambda_1 + \lambda_2 &= 0 \\ -\lambda_1 - \frac{1}{2}\lambda_3 &= 0 \end{cases} \implies \begin{cases} \lambda_1^* = -8 \\ \lambda_2^* = 8 \\ \lambda_3^* = 24 \end{cases}$$

Putting λ_i^* back into the dual function yields the following:

$$F(\lambda^*, \nu) = -48$$

Answer: -48 is the minimizer for the dual optimization problem

(e)

Strong duality implies that the maximizer for the dual optimization problem is equal to the minimizer for the primal optimization problem. We can check wether strong duality holds between the primal and dual optimization problems by checking if Slater's condition holds.

Slater's condition is a sufficient condition for strong duality to hold for a convex optimization problem. We can see that Slater's condition holds for our optimization problem since:

(1) f(x) is convex:

$$f(x) = x_1^2 + 4x_2^2$$

(2) Conditions for the $g_i(x) \leq 0$ constraints holds:

$$g_1(x) = 4 - x_1 - 2x_2, \quad g_2(x) = -x_1, \quad g_3(x) = -x_2$$

(2.1) All $g_i(x)$ are linear, which means they are convex

(2.2) $\exists x \quad [g_i(x) < 0 \quad \forall g_i]$ for example $x = (1, 1)$ satisfies this condition.

(3) Doesn't exist $h_i(x) = 0$ constraints

This means all conditions that must hold for the $h_i(x)$ automatically holds

Answer: Since strong duality holds, the primal objective and the dual objective are equal, which means the primal minimizer will be the same as the dual maximizer

Problem 2

$$\min_{x \in \mathbb{R}^n} x^T x$$

$$\text{s.t.} \quad Ax = b, \quad A \in \mathbb{R}^{m \times n}$$

(a)

We have the following Lagrangian of the optimization problem:

$$\mathcal{L}(x, y) = f(x) + y^T(Ax - b)$$

where $f(x) = x^T x$, so the Lagrangian becomes:

$$\mathcal{L}(x, y) = x^T x + y^T(Ax - b)$$

The dual function is obtained by minimizing $\mathcal{L}(x, y)$ over x:

$$F(y) = \min_x \mathcal{L}(x, y) = \min_x \left(x^T x + y^T(Ax - b)\right)$$

$$\nabla_x \mathcal{L}(x, y) = 2x + A^T y = 0$$

Solving for x: $\implies x^* = -\frac{1}{2}A^T y$

Dual Ascent Algorithm:

$$\begin{cases} x^{(t+1)} = -\frac{1}{2}A^T y^{(t)} & (= arg\min_x \mathcal{L}(x, y^{(t)})) \\ y^{(t+1)} = y^{(t)} + \alpha_t(Ax^{(t+1)} - b) \end{cases}$$

(b)

$$\mathcal{L}_\rho(x, y) = x^T x + y^T(Ax - b) + \frac{\rho}{2}\|Ax - b\|^2$$

$$\min_x \mathcal{L}(x, y) \implies \nabla_x \mathcal{L}_\rho = 2x + A^T y + \rho A^T(Ax - b) = 0$$

Solving for x:

$$x^{(t+1)} = \frac{\rho A^T b - A^T y}{2 + \rho\|A\|^2}$$

$$y^{(t+1)} = y^{(t)} + \rho(Ax^{(t+1)} - b)$$

Method of multipliers:

$$\begin{cases} x^{(t+1)} = \frac{\rho A^T b - A^T y}{2 + \rho\|A\|^2} & (= arg\min_x \mathcal{L}_\rho(x, y^{(t)}) \\ y^{(t+1)} = y^{(t)} + \rho(Ax^{(t+1)} - b) \end{cases}$$

(c)

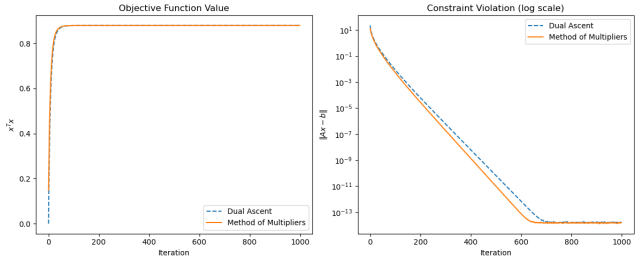


Figure 1: rho=270, alpha=0.001

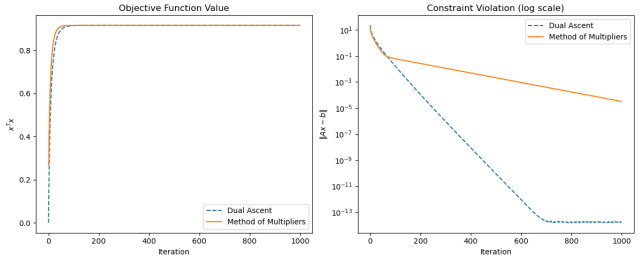


Figure 2: rho=350, alpha=0.001

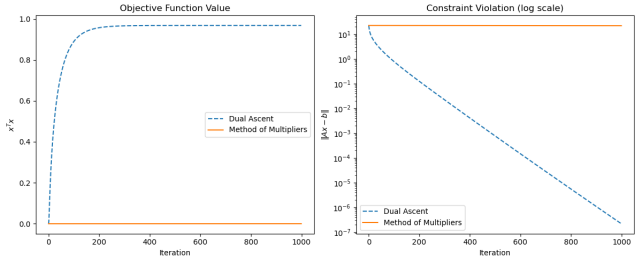


Figure 3: default start values

Answer: From my results I can deduce that MM performed well for a wider range of ρ-values. For example when I put the α-value too small the function value and the norm exploded for the dual ascent (didn't include this graph). MM didn't perform well for a small $\rho = 0.01$ I had too increase it a lot for MM to perform well. The optimum contained from the the two algorithms was approximately equal.
A note about computational efficiency is that Dual ascent is highly parallelisable. It would be possible to parallelise this function, i didn't though. This

parallelizability is indeed one advantage of dual ascent over MM, as MM requires sequential computations that cannot be easily distributed.

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

# Problem parameters
m, n = 500, 1000
A = np.random.randn(m, n)
b = np.ones(m)
alpha = 0.001
rho = 270
max_iters = 1000


def dual_ascent(A, b, alpha, max_iters):
    x_vals = []
    infeasibility = []

    y = np.zeros(m)
    x = np.zeros(n)

    for _ in range(max_iters):
        x = -0.5 * A.T @ y
        y = y + alpha * (A @ x - b)
        x_vals.append(x.T @ x)
        infeasibility.append(np.linalg.norm(A @ x - b))

    return x_vals, infeasibility

def method_of_multipliers(A, b, rho, max_iters):
    x_vals = []
    infeasibility = []

    y = np.zeros(m)
    x = np.zeros(n)

    for _ in range(max_iters):
        x = (rho * (A.T @ b) - A.T @ y) / (2 + (np.linalg.norm(A)**2))
        y = y + rho * (A @ x - b)

        x_vals.append(x.T @ x)
        infeasibility.append(np.linalg.norm(A @ x - b))

    return x_vals, infeasibility

x_vals_dual, infeasibility_dual = dual_ascent(A, b, alpha, max_iters)
x_vals_mm, infeasibility_mm = method_of_multipliers(A, b, rho, max_iters)

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(x_vals_dual, label='Dual Ascent', linestyle='dashed')
plt.plot(x_vals_mm, label='Method of Multipliers')
plt.xlabel("Iteration")
plt.ylabel(r"$x^T x$")
plt.title("Objective Function Value")
plt.legend()
```

```
plt.subplot(1, 2, 2)
plt.plot(infeasibility_dual, label='Dual Ascent', linestyle='dashed')
plt.plot(infeasibility_mm, label='Method of Multipliers')
plt.xlabel("Iteration")
plt.ylabel(r"$\|Ax - b\|$")
plt.yscale("log")
plt.title("Constraint Violation (log scale)")
plt.legend()

plt.tight_layout()
plt.show()
```

MATH 173B, HW7

Victor Pekkari — epekkari@ucsd.edu

March 14, 2025

Problem 1

(a)

We try to minimize the following function in P2:

$$\min_{x\in\mathbb{R}^n, z\in\mathbb{R}^m} f(z) + \lambda g(x) \tag{*}$$

From the constraint in P2 we know that:

$$Ax = z$$

This means we can do substitute z for Ax in **(*)** to get P1:

$$\min_{x\in\mathbb{R}^n} f(Ax) + \lambda g(x)$$

This ensures that every point every point-pair (a, b) that we put into $g(a)$ and $f(b)$ will have the same relationship for P1 and P2.

(b)

We can write the augmented Lagrangian as:

$$\mathcal{L}(x, z, y) = f(x) + g(x) + y^T(Ax - z) + \frac{\rho}{2}\|Ax - z\|^2$$

This give us the following ADMM-algorithm:

Initialize: $\rho > 0, x^{(0)} \in \mathbb{R}^n, z^{(0)}, y^{(0)} \in \mathbb{R}$

$$\begin{cases} x^{(t+1)} = \arg\min_x \mathcal{L}(x, z_t, y_t) \\ z^{(t+1)} = \arg\min_z \mathcal{L}(x_{t+1}, z, y_t) \\ y^{(t+1)} = y^{(t)} + \rho(Ax^{(t+1)} - z^{(t+1)}) \end{cases}$$

We can't get a closed form solution for the argmin since we don't know if the functions $f(x)$ and $g(x)$ are convex or not. This means we can't simplify this further. We can use numerical methods to approximate the argmin, many numerical approximator rely on the fact that the function is convex, so our approximator algorithm won't be that effective.

Problem 2

(a)

$$\min_{w,b,z} \sum_{i=1}^{N} \frac{1}{2}(z_i - y_i)^2 + \frac{1}{2}\|w\|^2 \qquad s.t \quad z_i = w^T x_i + b$$

(b)

$$\mathcal{L}_\rho(w, b, z, \nu) = \sum_{i=1}^{N} \left[\frac{1}{2}(z_i - y_i)^2 + \frac{\lambda}{2}\|w\|^2 + \nu_i(w^T x_i + b - z_i) + \frac{\rho}{2}\|w^T x_i + b - z_i\|^2 \right]$$

(c)

The ADMM updates consist of the following steps:
Initialize the variables: $w^{(0)} \in \mathbb{R}^d, b^{(0)}, z_i^{(0)}, u_i^{(0)} \in \mathbb{R}$

Do the thing below for desired number of iterations:

Update w, b

$$(w^{(t+1)}, b^{(t+1)}) = \arg\min_{w,b} \frac{\lambda}{2}\|w\|_2^2 + \frac{\rho}{2}\sum_{i=1}^{N}(w^T x_i + b - z_i^t + u_i^t)^2.$$

Solving for w,

$$w^{(t+1)} = \left(\lambda I + \rho \sum_{i=1}^{N} x_i x_i^T\right)^{-1} \left(\rho \sum_{i=1}^{N} x_i (z_i^t - u_i^t - b^{(t+1)})\right).$$

Solving for b,

$$b^{(t+1)} = \frac{1}{N}\sum_{i=1}^{N}(z_i^t - u_i^t - x_i^T w^{(t+1)}).$$

Update z

$$z_i^{(t+1)} = \arg\min_{z} \frac{1}{2}(z_i - y_i)^2 + \frac{\rho}{2}(z_i - w^{(t+1)} x_i - b^{(t+1)} + u_i^t)^2.$$

Solving for z_i,

$$z_i^{(t+1)} = \frac{y_i + \rho(w^{(t+1)} x_i + b^{(t+1)} - u_i^t)}{1 + \rho}.$$

Update the dual variable u

$$u_i^{(t+1)} = u_i^t + (w^{(t+1)} x_i + b^{(t+1)} - z_i^{(t+1)}).$$

This iterative process continues until convergence...

Problem 3

(a)

This function is convex in both w and b.

$$\min_{w,b} \frac{1}{N} \sum_{i=1}^{N} \log(1 + e^{-w^T x_i y_i}) + \frac{\lambda}{2} \|z\|^2$$

We construct the augmented Lagrangian:

$$\mathcal{L}(w, z, y) = \frac{1}{N} \sum_{i=1}^{N} \left[\log(1 + e^{-w^T x_i y_i}) + \frac{\lambda}{2} \|z\|^2 \right] + y^T (w - z) + \frac{\rho}{2} \|w - z\|^2$$

For the w-update, we solve:

$$w^{(t+1)} = \arg\min_{w} \frac{1}{N} \sum_{i=1}^{N} \log(1 + e^{-w^T x_i y_i}) + \frac{\rho}{2} \|w - z^{(t)} + \frac{y^{(t)}}{\rho}\|^2. \tag{1}$$

Since there is no closed-form solution, we use gradient descent to solve it, since we've been working with GD for 2 quarters I'll leave it out from this solution, it is implemented in my code however...

For the z-update:

$$z^{(t+1)} = \arg\min_{z} \frac{\lambda}{2} \|z\|^2 + \frac{\rho}{2} \|w^{(t+1)} - z + \frac{y^{(t)}}{\rho}\|^2. \tag{2}$$

Taking the derivative and setting it to zero (we can do this because the function is convex in z), we get:

$$z^{(t+1)} = \frac{\rho w^{(t+1)} + y^{(t)}}{\rho + \lambda}. \tag{3}$$

Finally, the dual variable y is updated as:

$$y^{(t+1)} = y^{(t)} + \rho(w^{(t+1)} - z^{(t+1)}). \tag{4}$$

Thus, the ADMM updates are summarized as:

$$\begin{cases} w^{(t+1)} = \arg\min_w \frac{1}{N} \sum_{i=1}^{N} \log(1 + e^{-w^T x_i y_i}) + \frac{\rho}{2} \|w - z^{(t)} + \frac{y^{(t)}}{\rho}\|^2, \\ z^{(t+1)} = \frac{\rho w^{(t+1)} + y^{(t)}}{\rho + \lambda}, \\ y^{(t+1)} = y^{(t)} + \rho(w^{(t+1)} - z^{(t+1)}). \end{cases}$$

Where the w is solved through GD

(b) Performance Evaluation

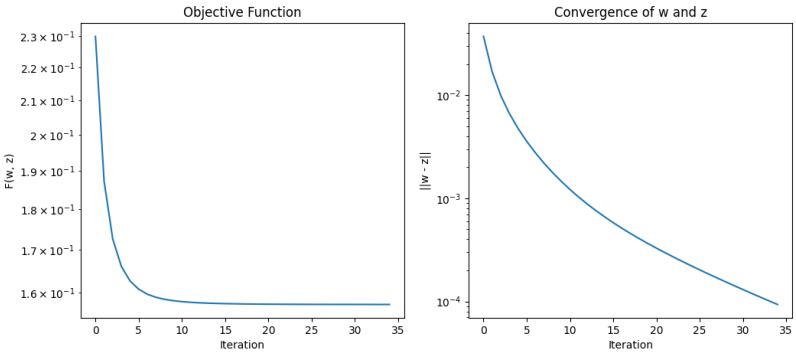


Figure 1: Objective function and convergence of w and z over iterations.

Results:
Training error: 0.0325
Test error: 0.0340

(ii) Performance and Convergence

The objective function shows a fast decrease in the beginning. The decease never stops, though the pace at which the function decreases slows down. The plot of $\|w - z\|$ in log-scale shows an exponential decay. The training error (0.0325) and test error (0.0340) are close, meaning the model generalizes well.

The computational complexity per iteration is primarily determined by the gradient descent step in the w-update. all other operations "inside the iteration" have constant timecomplexity. So the timecomplexity is $O(\#GD - iterations)$.

(iii) Advantages and Disadvantages of ADMM

Advantages:

- Support parallelisation
- Cost function doesn't have to be differentiable
- It provides robust convergence guarantees

Disadvantages:

- Each iteration involves solving a subproblem for w, which can be computationally expensive. If we for example have to run GD for many iterations
- Convergence can be slow