Homework 7 for Math 173A - Fall 2025

For coding questions, you must submit your code and the requested answers. **Your code must be present to receive points.**

1. Indicate whether the following functions are strongly convex. If so, find the strong convex parameter. Justify your answer.

   (a) $f : \mathbb{R} \to \mathbb{R}$ with $f(x) = x$.

   (b) $f : \mathbb{R}^2 \to \mathbb{R}$ with $f(x) = x_1^2 + x_2^2$.

   (c) $f : \mathbb{R} \to \mathbb{R}$ with $f(x) = \log(1 + e^x)$.

2. Here we will prove the PL-inequality used in class to prove fast convergence for strongly convex functions. Let $F(x)$ be a strongly convex function with parameter c, i.e., F is c-strongly convex. Our goal is to show that F satisfies the c-PL inequality:

$$F(x) - F(x^*) \leq \frac{1}{2c}\|\nabla F(x)\|^2 \qquad \text{for all } x \in \mathbb{R}^d. \tag{1}$$

   (a) Fix $x \in \mathbb{R}^d$ and define the quadratic function

$$q(y) = F(x) + \nabla F(x)^T(y - x) + \frac{c}{2}\|x - y\|^2.$$

   Find the y^* that minimizes $q(y)$.

   (b) Show that $q(y^*) = F(x) - \frac{1}{2c}\|\nabla F(x)\|^2$

   (c) Use the above to deduce (1).

   (d) Explain the proof technique in your own words to demonstrate understanding of what we did.

3. Consider Q3 from HW6, we plan to analyze gradient descent on this function, and want to understand in theory how fast GD will converge. In the end, you will be asked to compare it numerically with the accelerated GDs and with the conjugate gradient method.

   (a) Check whether f is L-smooth and α-strongly convex for some $L, \alpha > 0$. Find the corresponding L if f is L-smooth and the α if f is strongly convex.

(b) Give a range of step sizes, i.e., an interval in this case, so the method will converge using the theorem we learned for strongly convex and smooth functions.

(c) Continuing (b), what can we say about $f(x^{(t)}) - f(x^*)$ and its rate of decay according to the theorem we learned? Does it depend on the choice of the stepsize? Which stepsize gives you the best rate? If you want to achieve $f(x^{(t)}) - f(x^*) \leq 10^{-5}$ for $n = 100$ and $f(x^{(0)}) - f(x^*) = 1$, how many iterations do you need using the best rate you found?

(d) **Coding Question**. Program and run the conjugate gradient descent method for f starting at $x^{(0)} = 0$ for $n = 20$ and $n = 100$. Run the method for n iterations. Plot the $f(x^{(t)}) - f(x^*)$ for the conjugate gradient descent method and the three methods you have explored in Q3 from HW6 in the same figure. In a different figure, plot $\|x^{(t)} - x^*\|$ for the four methods. If you encounter a number smaller than 10^{-16}, set it to be 10^{-16}. In both plots, make the logarithmic scale for the vertical axis. Comment on the plots.

4. Consider the following set in $\mathbb{R}^n$ for an integer $s > 0$:

$$B = \{x \in \mathbb{R}^n \mid x_i \geq 0, \text{ for } i = 1, \ldots, n \text{ and } x \text{ has at most } s \text{ nonzeros.}\}.$$

(a) Find an expression for the orthogonal projection of a point $x \in \mathbb{R}^n$ onto B (No need for justification).

(b) For the function

$$f(x) = \frac{1}{2}\|Ax - b\|^2, \tag{2}$$

Write a projected gradient descent algorithm to solve

$$\min_{x \in \Omega} f(x) \tag{3}$$

for $\Omega = B$, with B from part (a). You need to specify the gradient formula and the projection formula. You do not need to specify the step size for this problem.

(c) **Coding Question:** Consider the optimization problem (3), where $A \in \mathbb{R}^{20\times 50}$ and $b \in \mathbb{R}^{20}$ are from the dataset `HW7Ab.csv`. The file `HW7Ab.csv` contains the data A and b. The first 50 columns form the matrix A and the last column is the vector b. The vector b is generated by setting $b = Ax^*$ for a vector $x^* \in \mathbb{R}^{50}$ that has 2 positive nonzeros. Note the linear system $Ax = b$ is underdetermined and has a lot of solutions. Program the projected gradient method for solving (3) with $s = 2$ to find the x^*. You can experiment with the stepsize and initialization to make sure $f(x^{(t)})$ converges to 0. You need to submit the code, the plot of $f(x^{(t)}) - f(x^*) = f(x^{(t)})$ vs iteration number, and the indices and values that has absolute value larger than 10^{-6} of x^* you found.

5. **Coding Question:** We will implement the SVM algorithm with gradient descent to classify two Gaussians in 2D. The dataset is given in `HW7SVM.csv`.

   (a) In `HW7SVM.csv`, the first 100 rows are the data for cluster 1: $(x_i, y_i) \in \mathbb{R}^2 \times \mathbb{R}$, $i = 1, \ldots, 100$, with $y_i = 1$ always. The next 100 rows are the data for cluster 2: $(x_i, y_i) \in \mathbb{R}^2 \times \mathbb{R}$, $i = 101, \ldots, 200$, with $y_i = -1$ always. Create and turn in a scatter plot of the feature vectors, i.e., the x_is, colored by the label, i.e., y_is (blue for 1 and red for -1).

   (b) Create a function for the gradient of the loss

$$L(w) = \frac{1}{2}\|w\|^2 + \sum_{i=1}^{n} \max(0, 1 - y_i\langle x_i, w\rangle)$$

$$\nabla L(w) = w + \sum_{i=1}^{n} -y_i x_i \cdot 1_{1-y_i\langle x_i, w\rangle > 0},$$

   where $1_{1-y_i\langle x_i,w\rangle>0} = \begin{cases} 1 & \text{if } 1 - y_i\langle x_i, w\rangle > 0 \\ 0 & \text{else} \end{cases}$. Also, here $n = 200$. To compute the gradient, you'll have to compute an indicator of whether $1 - y_i\langle x_i, w\rangle$ is positive or negative at every point, and sum up the contribution of this term for all points where it's positive.

   (c) Setting the step size $\mu = 10^{-4}$ and starting at $w^{(0)} = (-1, 1)$, run 1000 iterations of gradient descent. You will create two plots.

      i. Plot the classification error (averaged over all the points) as a function of the iterations. The classification of x_i is determined by $\text{sign}(\langle x_i, w\rangle)$.

      ii. Plot the margin $\frac{2}{\|w\|}$ as a function of the iterations. This shows how much of a gap you have between the classes you've learned.

   (d) Create another scatter plot of your data, but this time color the points by the function $f(x_i) = 1 - y_i \cdot \langle x_i, w\rangle$. The numbers closest to 0 (positive numbers or largest negative numbers) will show you which points were "most important" in determining the classification.

Note: Here we're only defining a subspace classifier (i.e. the classifier goes through the origin). This is fine for our problem as the gaussians are on opposite sides of the origin. If you want to create an intercept term, simply append a vector of all 1's as a column of your data, and now your weight vector will be of dimension 3 instead of 2. This is done the same way as when running least squares regression.

HW7 SABELLA SAULIND

1) Indicate whether the following functions are strongly convex. If so, find the strong convex parameter. Justify your answer.

Def: A function F is strongly convex if $\exists c > 0$ s.t. $\forall x, y \in \mathbb{R}^n$,

$$F(y) \geq F(x) + \nabla F(x)^T (y - x) + \frac{1}{2} c \|y - x\|^2$$

Lemma: A C^2 function $f: \mathbb{R}^n \to \mathbb{R}$ is c-strongly convex iff

$$\forall x, \quad \nabla^2 f(x) - c \cdot I \succeq 0$$

$$\Leftrightarrow \quad \forall x, v, \quad v^T \nabla^2 f(x) v \geq c \|v\|^2$$

$$\Leftrightarrow \quad \forall x, \quad \lambda_{min}(\nabla^2 f(x)) \geq c \quad \text{for} \quad c > 0$$

(a) $f: \mathbb{R} \to \mathbb{R}$ with $f(x) = x$

Check: $\forall x$, $f''(x) \geq c$, where $c > 0$

$f''(x) = 0$, so $\nexists c > 0$ such that $f''(x) \geq c$

So f is not strongly convex.

(b) $f: \mathbb{R}^2 \to \mathbb{R}$ with $f(x) = x_1^2 + x_2^2$

Check: $\forall x$, $\lambda_{min}(\nabla^2 f(x)) \geq c$

$$\nabla f(x) = \begin{bmatrix} 2x_1 \\ 2x_2 \end{bmatrix} \qquad \nabla^2 f(x) = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

$$\lambda_{min}(\nabla^2 f(x)) = 2 \quad \Rightarrow \quad c = 2$$

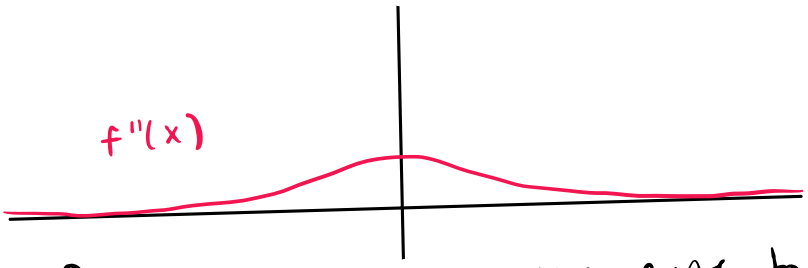
Yes, f is strongly convex with $c = 2$.

(c) $f: \mathbb{R} \to \mathbb{R}$ with $f(x) = \log(1 + e^x)$

$f'(x) = \frac{e^x}{1+e^x}$, $f''(x) = \frac{e^x(1+e^x) - e^x \cdot e^x}{(1+e^x)^2} = \frac{e^x}{(1+e^x)^2} > 0$

Want $\frac{e^x}{(1+e^x)^2} \geq c$, where $c > 0$, for all x.

Graph of $f''(x)$:



As $x \to \pm\infty$, $f''(x) \to 0$ but never actually gets to it (since $e^x > 0$ for any x). Therefore we cannot find some $c > 0$ such that $f''(x) \geq c$ for all values of x.

So f is not strongly convex.

2) Here we will prove the PL-inequality used in class to prove fast convergence for strongly convex functions. Let $F(x)$ be a strongly convex function with parameter c, ie., F is c-strongly convex. Our goal is to show that F satisfies the c-PL inequality:

$$F(x) - F(x^*) \leq \frac{1}{2c} \|\nabla F(x)\|^2 \quad \forall x \in \mathbb{R}^d$$

(a) Fix $x \in \mathbb{R}^d$ and define the quadratic function

$$q(y) = F(x) + \nabla F(x)^T (y - x) + \frac{c}{2} \|x - y\|^2$$

Find the y^* that minimizes $q(y)$.

$$\nabla q(y) = \nabla \left(F(x) + \nabla F(x)^T (y - x) + \frac{c}{2} \|x - y\|^2 \right)$$

$$= \nabla \left(\nabla F(x)^T y - \nabla F(x)^T x + \frac{c}{2} \|x - y\|^2 \right)$$

$$= \nabla F(x) + \nabla \left(\frac{c}{2} \|x - y\|^2 \right)$$

$$= \nabla F(x) + c(y-x)$$

$$\nabla F(x) + c(y-x) = 0 \quad \Rightarrow \quad \nabla F(x) + cy - cx = 0$$

$$\Rightarrow \quad cy = cx - \nabla F(x) \quad \Rightarrow \quad y^* = x - \frac{1}{c}\nabla F(x)$$

(b) Show that $q(y^*) = F(x) - \frac{1}{2c}\|\nabla F(x)\|^2$

$$q(y^*) = q\left(x - \frac{1}{c}\nabla F(x)\right)$$

$$= F(x) + \nabla F(x)^T\left(x - \frac{1}{c}\nabla F(x) - x\right) + \frac{c}{2}\left\|x - x - \frac{1}{c}\nabla F(x)\right\|^2$$

$$= F(x) - \frac{1}{c}\nabla F(x)^T \nabla F(x) + \frac{c}{2}\left\|-\frac{1}{c}\nabla F(x)\right\|^2$$

$$= F(x) - \frac{1}{c}\|\nabla F(x)\|^2 + \left(\frac{c}{2}\cdot\frac{1}{c^2}\right)\|\nabla F(x)\|^2$$

$$= F(x) + \left(\frac{-1}{c} + \frac{1}{2c}\right)\|\nabla F(x)\|^2 = F(x) - \frac{1}{2c}\|\nabla F(x)\|^2$$

(c) Use the above to deduce (1).

(1) $F(x) - F(x^*) \leq \frac{1}{2c}\|\nabla F(x)\|^2$

We know that F is a strongly convex function, thus, by a previous theorem,

$$F(y) \geq F(x) + \nabla F(x)^T(y-x) + \frac{1}{2}c\|y-x\|^2 = q(y)$$

$$\Rightarrow F(y) \geq q(y).$$

We know that x^* minimizes F, and above implies

$$F(x^*) \geq q(x^*)$$

We know that y^* minimizes q, so

$$F(x^*) \geq q(x^*) \geq q(y^*)$$

$$\Rightarrow F(x^*) \geq q(y^*)$$

$$\Rightarrow F(x^*) \geq F(x) - \frac{1}{2c}\|\nabla F(x)\|^2$$

Rearrange to get

$$F(x) - F(x^*) \leq \frac{1}{2c}\|\nabla F(x)\|^2.$$

(d) Explain the proof technique in your own words to demonstrate understanding of what we did.

In (a) & (b), we found the minimizer of the strong convexity lower bound $q(y)$ for $F(y)$, and then found the minimum value of the lower bound $q(y^*)$. Then, we took the minimum value of F, $F(x^*)$, and used the simple logic of minimizers to deduce that $F(x^*) \geq q(y^*)$. Knowing the formula for $q(y^*)$ from (b), we proved the inequality.

3) Consider Q3 from HW6, we plan to analyze GD on this function, and want to understand in theory how fast GD will converge. In the end, you will be asked to compare it numerically with the accelerated GDs and with the conjugate gradient method.

$A \in \mathbb{R}^{n\times n}$ diagonal matrix with diagonal entries $A_{ii} = i$
$b \in \mathbb{R}^n$ vector with all 1 entries. Define

$$f(x) = \frac{1}{2}x^T A x - b^T x.$$

(a) Check whether f is L-smooth and α-strongly convex for some $L, \alpha > 0$. Find the corresponding L if f is L-smooth and the α if f is strongly convex.

L-smooth: f is C^2, so we use $\|\nabla^2 f(x)\|_{op} \leq L \ \forall x$

$$\nabla f(x) = Ax - b \qquad \nabla^2 f(x) = A$$

$$\|A\|_{op} = \lambda_{max}(A) = n$$

Yes, f is L-smooth with $L = n$.

Strongly convex: $\forall x$, $\lambda_{min}(\nabla^2 f(x)) \geq \alpha > 0$

$$\lambda_{min}(A) = 1 \quad \Rightarrow \quad \alpha = 1$$

Yes, f is α-strongly convex with $\alpha = 1$.

(b) Give a range of step sizes, ie., an interval in this case, so the method will converge using the theorem we learned for strongly convex and smooth functions.

Theorem: If F is α-strongly convex and L-smooth, then

$$F(x^{(t)}) - F(x^*) \leq (1 - \mu\alpha)^t \left(F(x^{(0)}) - F(x^*)\right)$$

for any $\mu \in (0, \frac{1}{L}]$

$$\Rightarrow \mu \in (0, \tfrac{1}{n}]$$

(c) Continuing (b), what can we say about $f(x^{(t)}) - f(x^*)$ and its rate of decay according to the theorem we learned? Does it depend on the choice of the stepsize? Which stepsize gives you the best rate? If you want to achieve $f(x^{(t)}) - f(x^*) \leq 10^{-5}$ for $n = 100$ and $f(x^{(0)}) - f(x^*) = 1$, how many iterations do you need using the best rate you found?

From theorem, with our α and L:

$$f(x^{(t)}) - f(x^*) \leq (1 - \mu)^t \left(f(x^{(0)}) - f(x^*)\right), \quad \mu \in (0, \tfrac{1}{n}]$$

The rate of decay definitely depends on μ, since at each iteration the decay increases by a factor of $(1-\mu)$. The "best" rate of decay would mean faster, so we'd want $(1-\mu)$ to be larger. Since $\mu \in [0, \frac{1}{n}]$, we want μ to be the "smallest" in that range so when we subtract from 1 it gives the largest possible value. So $\mu = \frac{1}{n}$ would give the "best" rate.

Want: $f(x^{(t)}) - f(x^*) \leq 10^{-5}$ with $n = 100$, $f(x^{(0)}) - f(x^*) = 1$

$$\Rightarrow f(x^{(t)}) - f(x^*) \leq \left(1 - \frac{1}{100}\right)^t \leq 10^{-5}$$

$$\Rightarrow (0.99)^t \leq 10^{-5}$$

$$\Rightarrow \ln((0.99)^t) \leq \ln(10^{-5})$$

$$\Rightarrow t \geq \ln(10^{-5}) / \ln(0.99)$$

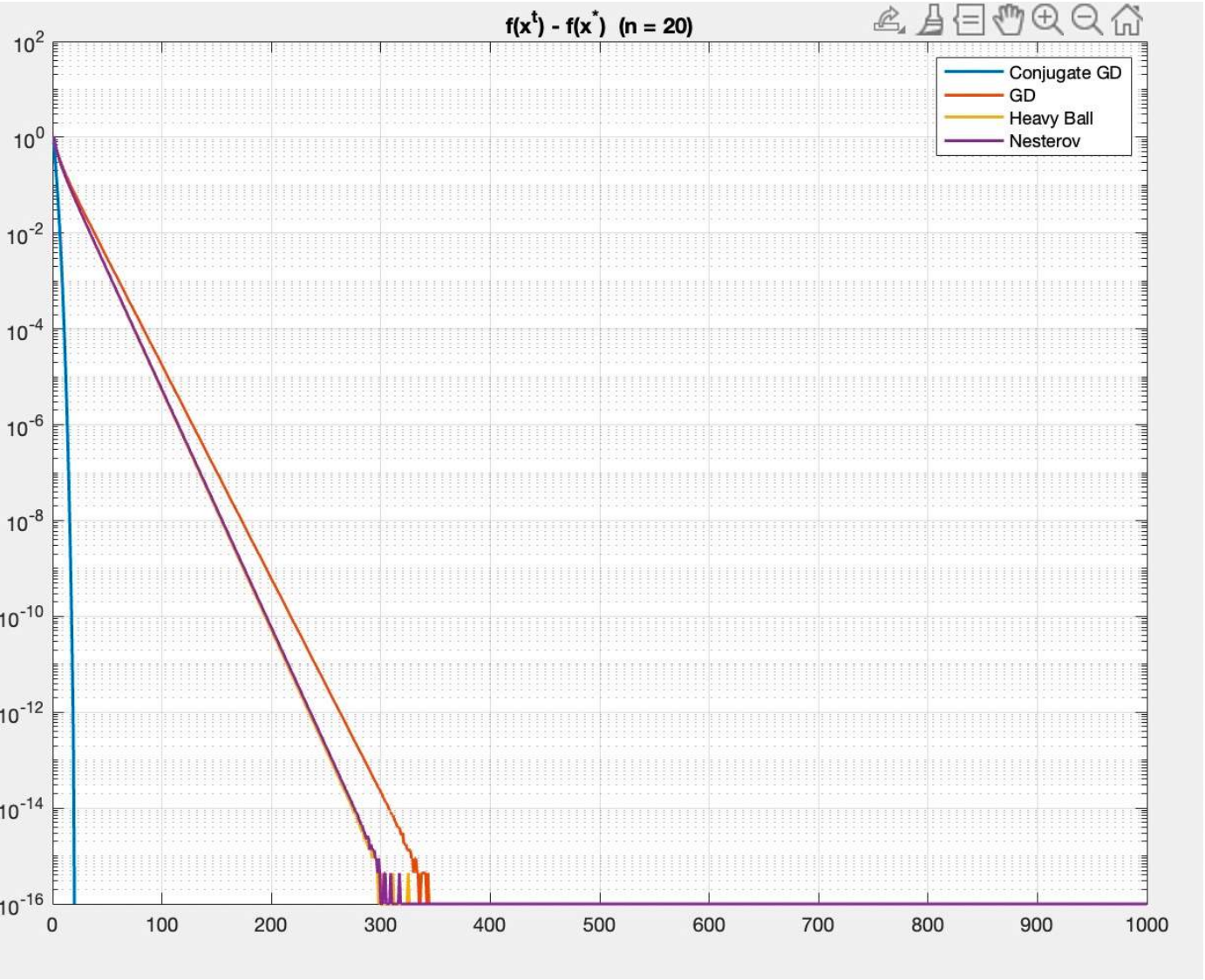
$$\Rightarrow t \geq -11.51 / -0.01$$

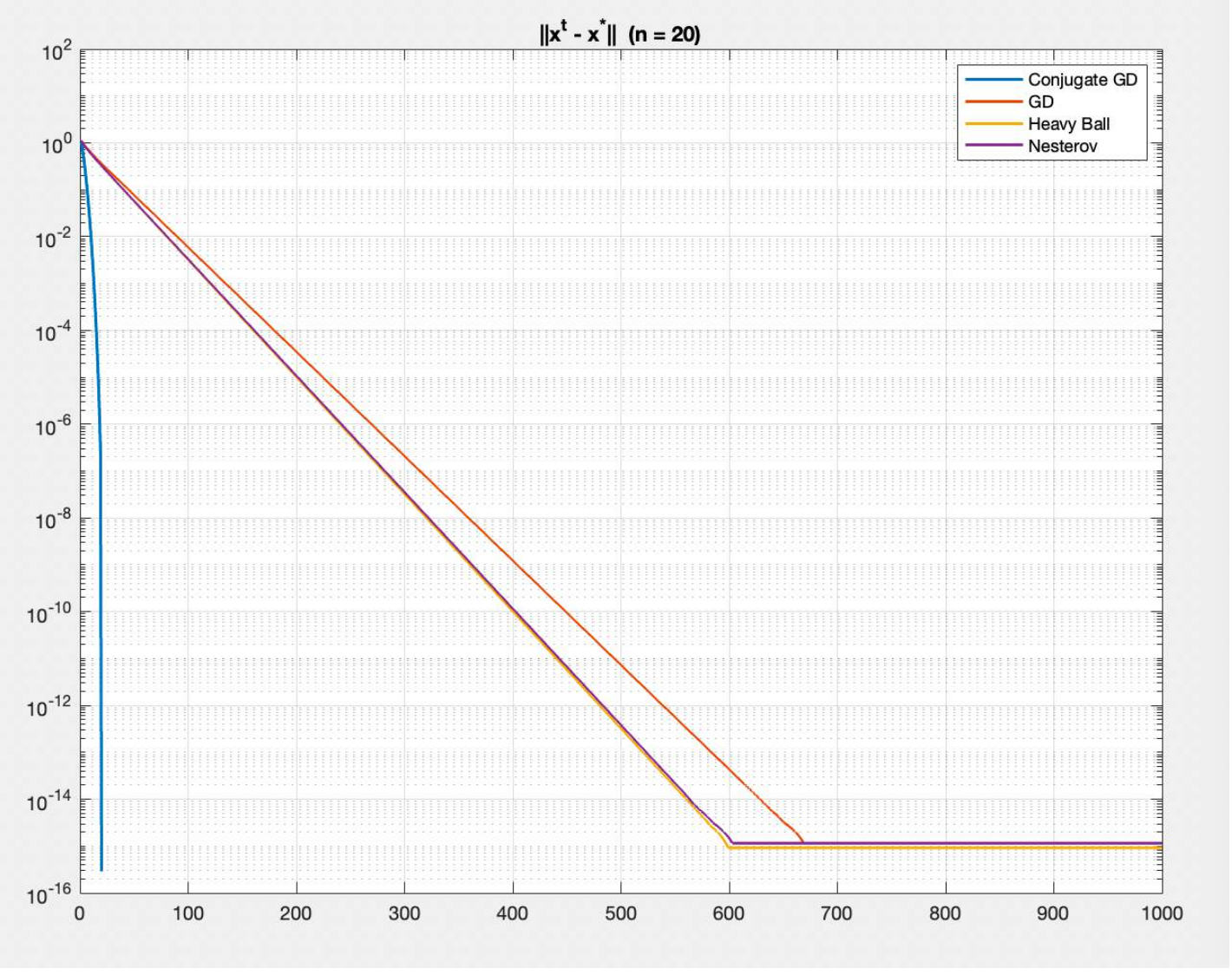
$$\Rightarrow t \geq 1145.53 \quad \Rightarrow t = 1146 \text{ iterations}$$

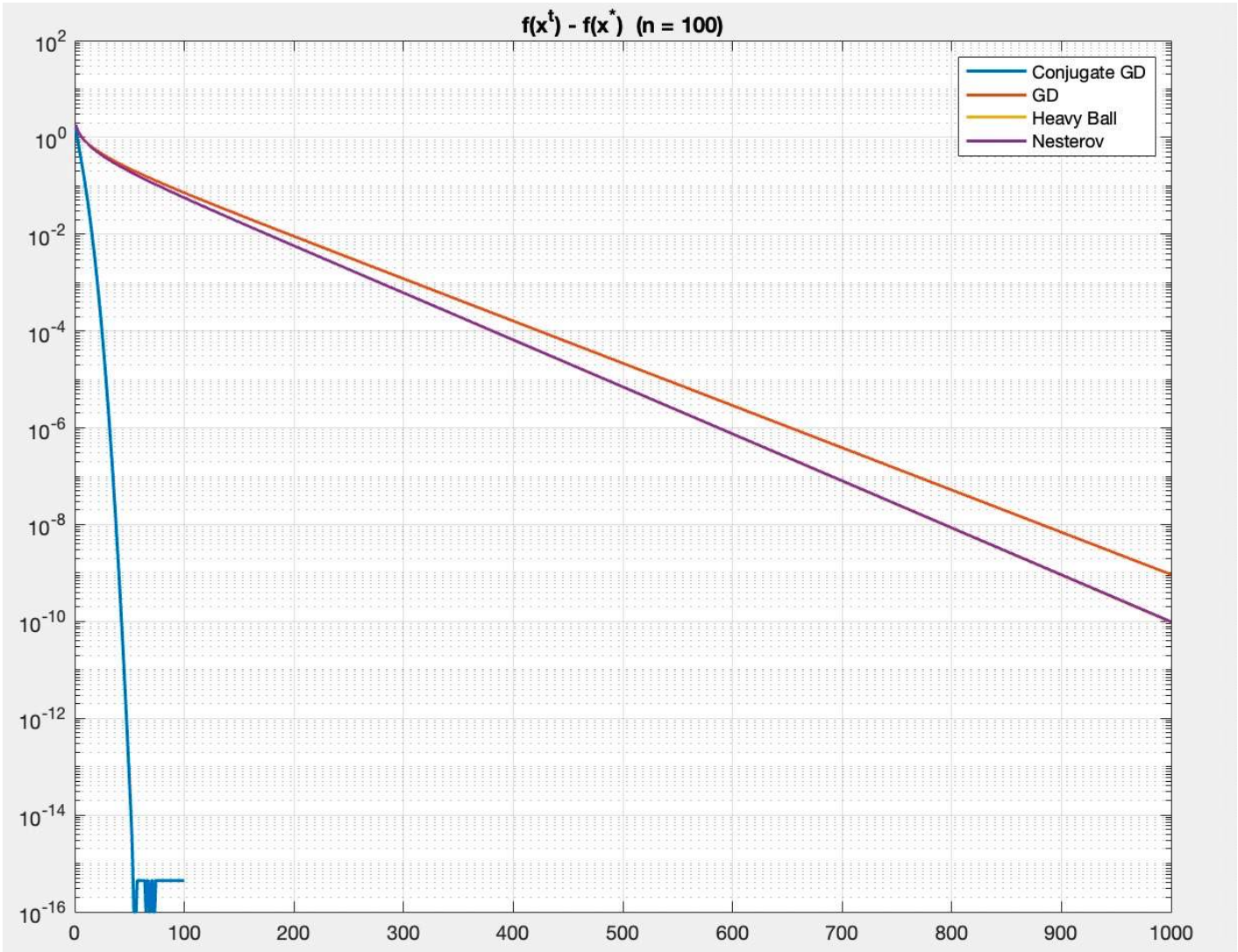
(d) Coding Question. Program and run the conjugate gradient descent method for f starting at $x^{(0)} = 0$ for $n = 20$ and $n = 100$. Run the method for n iterations. Plot the $f(x^{(t)}) - f(x^*)$ for the conjugate gradient descent method and the three methods you have explored in Q3 from HW6 in the same figure. In a different figure, plot $\|x^{(t)} - x^*\|$ for the four methods. If you encounter a number smaller than 10^{-16}, set it to be 10^{-16}. In both plots, make the logarithmic scale for the vertical axis. Comment on the plots.

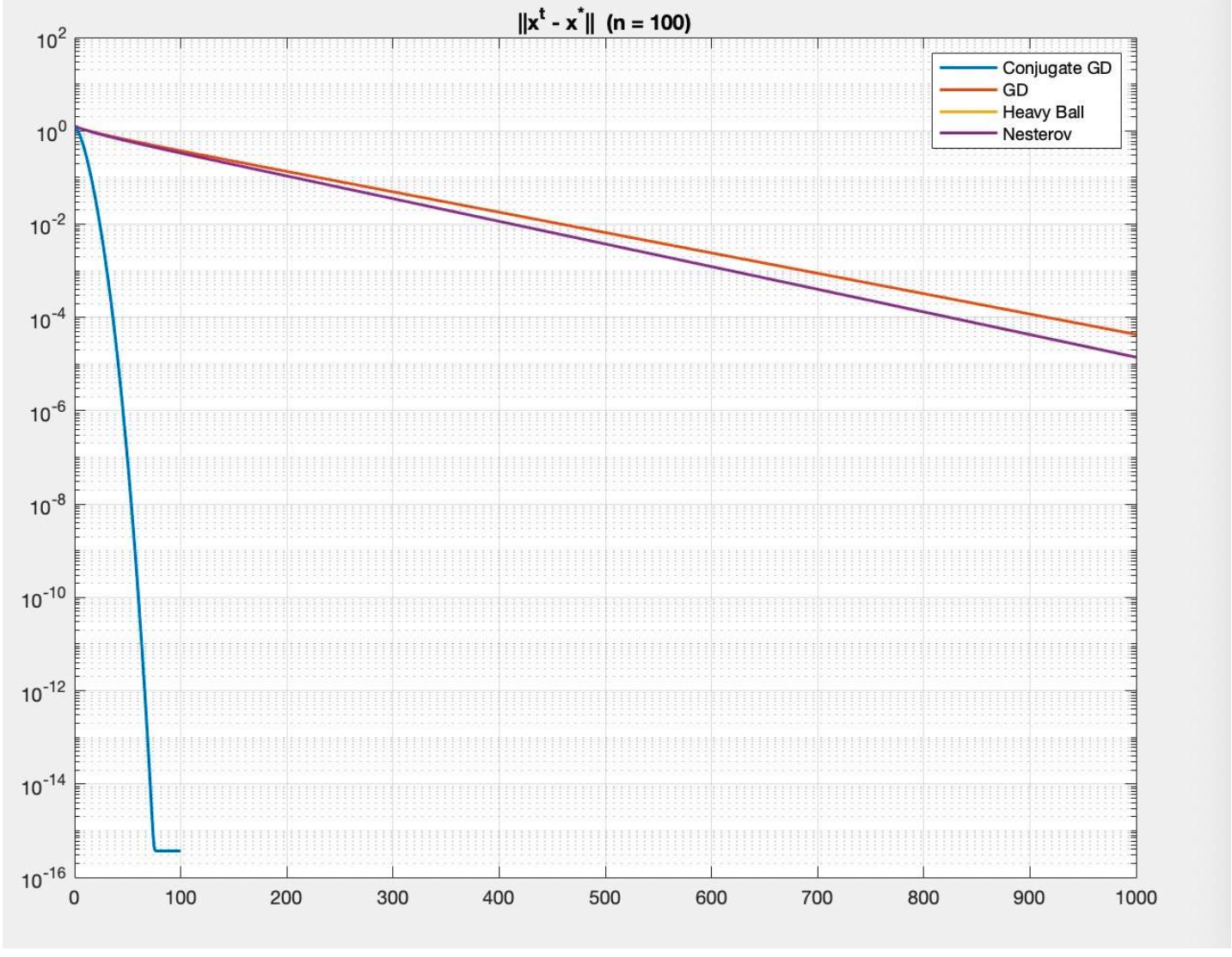
```m
function [f_error, x_error] = conjugate_GD(n)
    A = diag(1:n);
    b = ones(n, 1);
    x0 = zeros(n, 1);
    x_star = A\b;
    f_star = 0.5 * x_star' * A * x_star - b' * x_star;
    f_error = zeros(n,1);
    x_error = zeros(n, 1);
    x = x0;
    r = b - A*x;
    p = r;
    r_old = r' * r;
    for k = 1:n
        Ap = A * p;
        alpha = r_old /(p'*Ap);
        x = x + alpha * p;
        r = r - alpha * Ap;
        r_new = r' * r;
        f_val = 0.5 * x' * A * x - b' * x;
        f_error(k) = f_val - f_star;
        x_error(k) = norm(x - x_star);
        beta = r_new / r_old;
        p = r + beta*p;
        r_old = r_new;
    end
end
```

```m
for n = [20, 100]
    A = diag(1:n);
    b = ones(n,1);
    x_star = A\b;

    [f_CG, x_CG] = conjugate_GD(n);
    [f_GD, x_GD] = gd_hw6(n);
    [f_HB, x_HB] = gd_HB(n);
    [f_NA, x_NA] = gd_NA_2(n);

    f_CG(f_CG < 1e-16) = 1e-16;
    f_GD(f_GD < 1e-16) = 1e-16;
    f_HB(f_HB < 1e-16) = 1e-16;
    f_NA(f_NA < 1e-16) = 1e-16;

    x_CG(x_CG < 1e-16) = 1e-16;
    x_GD(x_GD < 1e-16) = 1e-16;
    x_HB(x_HB < 1e-16) = 1e-16;
    x_NA(x_NA < 1e-16) = 1e-16;

    figure;
    semilogy(f_CG, 'LineWidth', 1.5); hold on;
    semilogy(f_GD, 'LineWidth', 1.5);
    semilogy(f_HB, 'LineWidth', 1.5);
    semilogy(f_NA, 'LineWidth', 1.5);
    title(sprintf("f(x^t) - f(x^*)  (n = %d)", n));
    legend("Conjugate GD", "GD", "Heavy Ball", "Nesterov");
    grid on;

    figure;
    semilogy(x_CG, 'LineWidth', 1.5); hold on;
    semilogy(x_GD, 'LineWidth', 1.5);
    semilogy(x_HB, 'LineWidth', 1.5);
    semilogy(x_NA, 'LineWidth', 1.5);
    title(sprintf("||x^t - x^*||  (n = %d)", n));
    legend("Conjugate GD", "GD", "Heavy Ball", "Nesterov");
    grid on;

end
```


$f(x^t) - f(x^*)$ (n = 20)
Conjugate GD
GD
Heavy Ball
Nesterov
10^{2}
10^{0}
10^{-2}
10^{-4}
10^{-6}
10^{-8}
10^{-10}
10^{-12}
10^{-14}
10^{-16}
0
100
200
300
400
500
600
700
800
900
1000


$\|x^t - x^*\|$ (n = 20)
Conjugate GD
GD
Heavy Ball
Nesterov
10^{2}
10^{0}
10^{-2}
10^{-4}
10^{-6}
10^{-8}
10^{-10}
10^{-12}
10^{-14}
10^{-16}
0
100
200
300
400
500
600
700
800
900
1000


$f(x^t) - f(x^*)$ (n = 100)
Conjugate GD
GD
Heavy Ball
Nesterov
10^{2}
10^{0}
10^{-2}
10^{-4}
10^{-6}
10^{-8}
10^{-10}
10^{-12}
10^{-14}
10^{-16}
0
100
200
300
400
500
600
700
800
900
1000


$\|x^t - x^*\|$ (n = 100)
Conjugate GD
GD
Heavy Ball
Nesterov
10^{2}
10^{0}
10^{-2}
10^{-4}
10^{-6}
10^{-8}
10^{-10}
10^{-12}
10^{-14}
10^{-16}
0
100
200
300
400
500
600
700
800
900
1000

n=20: We can see from the plot that conjugate GD has a very fast convergence compared to normal GD, HB, and NA, which all behave very similarly. In the $f(x^{(t)}) - f(x^*)$ plot, conjugate GD appears to converge to 10^{-16} in exactly n iterations, while GD, HB, and NA all take ~300+ iterations to converge. In this plot, GD, NA, and HB oscillate after reaching convergence, which I think is because I did not tell any of the methods to stop running once they converged. We can still see that regular GD has the slowest convergence, HB/NA have similar rates that are slightly faster than GD, and conjugate GD converges much, much faster than the rest. The same can be said for the $\|x^{(t)} - x^*\|$ plot about the rates of convergence, however, this time GD, HB, NA converge in about 600+ iterations, so the difference between conjugate GD is even greater. There is no oscillation in this plot, so perhaps the oscillation in $f(x^{(t)}) - f(x^*)$ had to do with the stepsize choices being optimal for finding x^*.

n=100: Again, in both $f(x^{(t)}) - f(x^*)$ and $\|x^{(t)} - x^*\|$, we see conjugate GD converge much, much faster than GD, HB, and NA. Conjugate GD actually appears to converge in less than n iterations, oscillating to the 10^{-16} bound due to fast convergence. Compared to n=20, the GD, HB, and NA appear to converge much, much slower, still with similar rates to one another. So, conjugate GD runs "better" with larger n, and GD, HB, NA run "worse" with larger n, with their step sizes being inversely proportional to n.

4) Consider the following set in $\mathbb{R}^n$ for an integer $s > 0$:

$B = \{x \in \mathbb{R}^n \mid x_i \geq 0$, for $i = 1, \ldots, n$ and x has at most s nonzeros$\}$.

(a) Find an expression for the orthogonal projection of a point $x \in \mathbb{R}^n$ onto B (NO need to justify)

$$(\Pi_B(x \in \mathbb{R}^n))_i = \begin{cases} \max(x_i, 0), & x_i \in \{s \text{ largest entries of } x\} \\ 0, & x_i \notin \{s \text{ largest entries of } x\} \end{cases}$$

(b) For the function $f(x) = \frac{1}{2}\|Ax - b\|^2$, write a projected gradient descent alg to solve $\min_{x \in \Omega} f(x)$ for $\Omega = B$. You need to specify the gradient formula and the projection formula. You do not need to specify step size.

alg proj-GD

Initialize $x^{(0)} \in B$

(Initialize stepsize - not specified in problem)

Initialize $s > 0$

For $i = 0, 1, \ldots,$ do

grad $= A^T(Ax^{(i)} - b)$

(update stepsize - not specified in this problem)

$y^{(i+1)} = x^{(i)} - (\text{stepsize}) * \text{grad}$

for $j = 1, \ldots, n$ do

if $y^{(i)}[j] < 0$: $y^{(i)} = 0$ end

end

$x^{(i+1)} = \text{zeros}(n)$

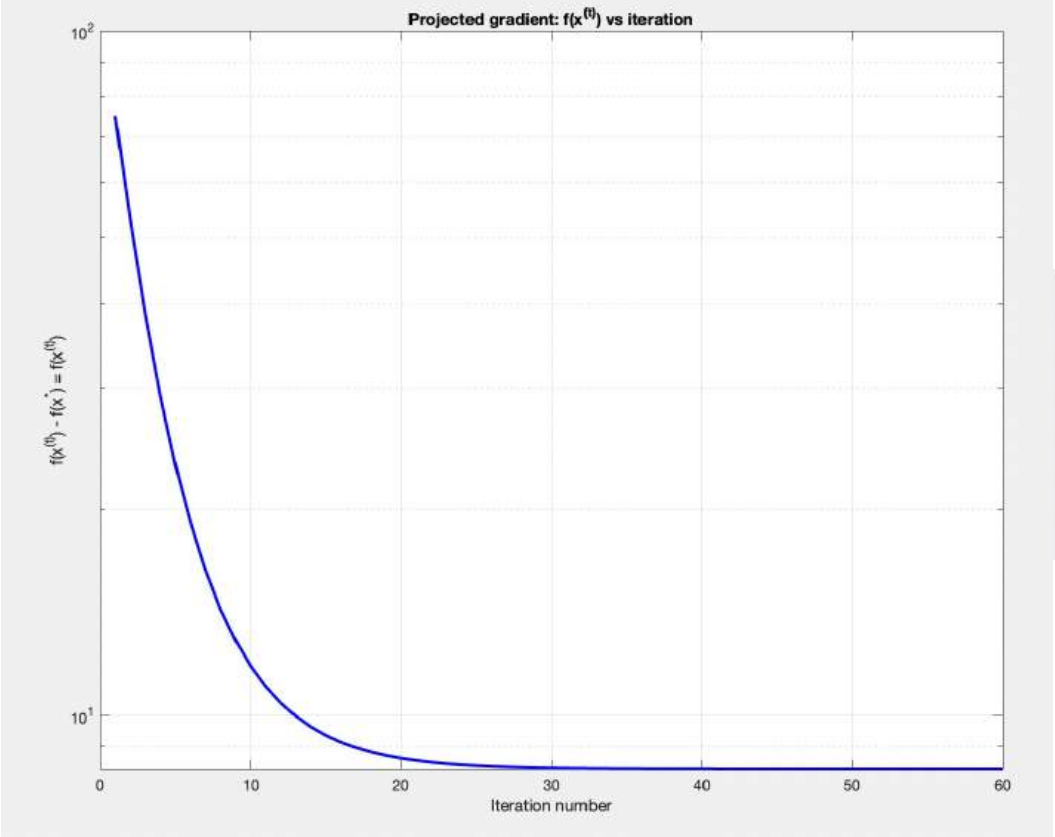
count $= 0$

Initialize empty list L

```
for j = 1, ..., n do
    if y^(i)[j] > 0 : cont ++ end
    add y^(i)[j] to L
end
If cont ≤ s : x^(i+1) = y^(i) end
else:
    Sort L by descending value
    For j = 1, ..., s do
        valve = L[j-1]
        x^(i+1)[j-1] = valve
    end
end
end
```

(c) Coding Question. Consider problem (3), where $A \in \mathbb{R}^{20 \times 50}$ and $b \in \mathbb{R}^{20}$ are from dataset HW7Ab.csv. The csv file contains the data A and b. First 50 columns form A, last col is b. The vector b is generated by setting $b = Ax^*$ for a vector $x^* \in \mathbb{R}^{50}$ that has 2 positive nonzeros. Note $Ax = b$ is underdetermined and has a lot of solutions. Program the projected gradient method for solving (3) with $s = 2$ to find the x^*. You can experiment with the stepsize and initialization to make sure $f(x^{(t)})$ converges to 0. You need to submit the code, the plot $f(x^{(t)}) - f(x^*) = f(x^{(t)})$ vs iteration number, and the indices and values that has abs val larger than 10^{-6} of x^* you found.

```m
function x = HW7_q4()
    data = readmatrix('HW7Ab.csv');
    A = data(:, 1:50);
    b = data(:, 51);
    [~, n] = size(A);
    s = 2;
    x0 = zeros(n, 1);
    x = x0;
    max_iter = 60;
    L = norm(A'*A, 2);
    alpha = 1/L;
    f_vals = zeros(max_iter, 1);
    x_error = zeros(max_iter, 1);
    for i = 1:max_iter
        grad = A' * (A*x-b);
        f_vals(i) = 0.5 * norm(A*x-b)^2;
        y = x - alpha * grad;
        y = max(y, 0);
        [sorted_vals, sorted_idx] = sort(y, 'descend');
        x_new = zeros(n, 1);
        keep = sorted_idx(1:min(s, length(sorted_idx)));
        x_new(keep) = y(keep);
        x_error(i) = norm(x_new - x);
        x = x_new;
    end
    above_threshold = abs(x) > 1e-6;
    if sum(above_threshold) > 0
        fprintf('Index   Value\n');
        for i = 1:n
            if above_threshold(i)
                fprintf('%3d     %12.8f\n', i, x(i));
            end
        end
    end
    figure;
    semilogy(1:length(f_vals), f_vals, 'b-', 'LineWidth', 2);
    xlabel('Iteration number');
    ylabel('f(x^{(t)}) - f(x^{*}) = f(x^{(t)})');
    title('Projected gradient: f(x^{(t)}) vs iteration');
    grid on;
end
```



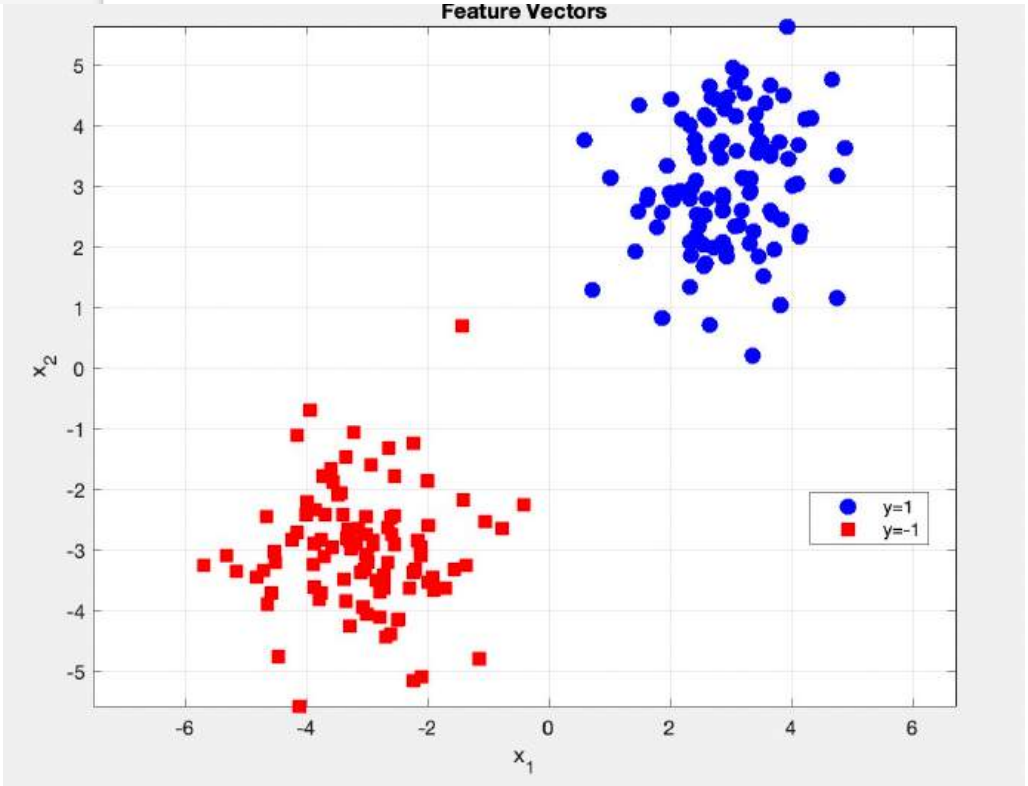
Command Window

```
>> HW7_q4
Index   Value
 15       2.81583484
 17       0.41767449
```

5) Coding question. We will implement the SVM algorithm with GD to classify two Gaussians in 2D. Dataset is HW7SVM.csv.

(a) In HW7SVM.csv, first 100 rows are the data for cluster 1: $(x_i, y_i) \in \mathbb{R}^2 \times \mathbb{R}$, $i = 1, \ldots, 100$, with $y_i = 1$ always. The next 100 rows are the data for cluster 2: $(x_i, y_i) \in \mathbb{R}^2 \times \mathbb{R}$, $i = 101, \ldots, 200$ with $y_i = -1$ always. Create and turn in a scatter plot of the feature vectors, i.e. the x_is, colored by the label, i.e., y_is (blue for 1 and red for -1).

```
data = readmatrix('HW7SVM.csv');
X = data(:, 1:2);
y = data(:, 3);

figure;
plot(X(1:100, 1), X(1:100, 2), 'bo', 'MarkerSize', 8, ...
    'MarkerFaceColor', 'b', 'DisplayName', 'y=1');
hold on;

plot(X(101:200, 1), X(101:200, 2), 'rs', 'MarkerSize', 8, ...
    'MarkerFaceColor', 'r', 'DisplayName', 'y=-1');

xlabel('x_1');
ylabel('x_2');
title('Feature Vectors');
legend('Location', 'best');
grid on;
axis equal;
```



(b) Create a function for the gradient of the loss

$$L(w) = \frac{1}{2}\|w\|^2 + \sum_{i=1}^{n} \max(0, 1 - y_i \langle x_i, w \rangle)$$

$$\Delta L(w) = w + \sum_{i=1}^{n} -y_i x_i \cdot 1_{1 - y_i \langle x_i, w \rangle > 0},$$

where $1_{1 - y_i \langle x_i, w \rangle > 0} = \begin{cases} 1 & \text{if } 1 - y_i \langle x_i, w \rangle > 0, \\ 0 & \text{else} \end{cases}$. Also, $n = 200$.

To compute the gradient, you'll have to compute an indicator of whether $1 - y_i \langle x_i, w \rangle > 0$ at every point, and sum up the contribution of this term for all points where its positive.

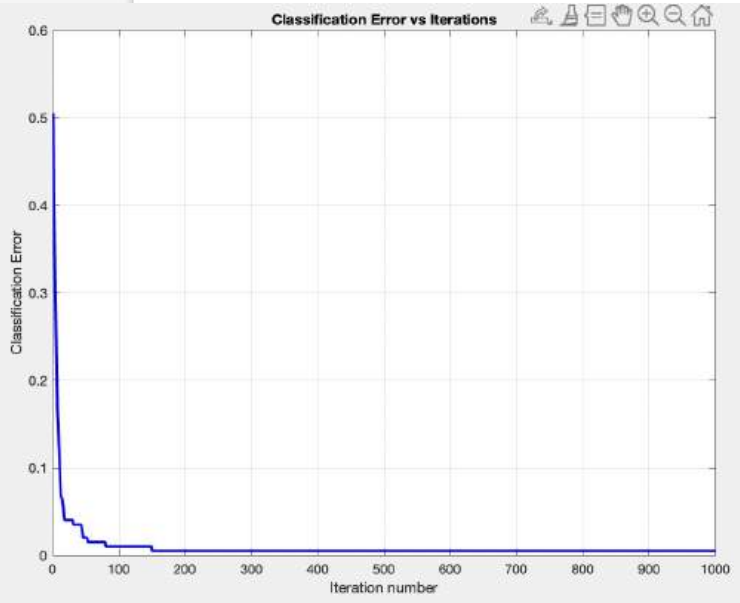
```
function grad = grad(w, X, y)
    [n, ~] = size(X);
    w = w(:);
    grad = w;
    for i = 1:n
        dot_product = X(i, :) * w;
        margin = 1 - y(i) * dot_product;
        if margin > 0
            grad = grad - y(i) * X(i, :)';
        end
    end
end
```

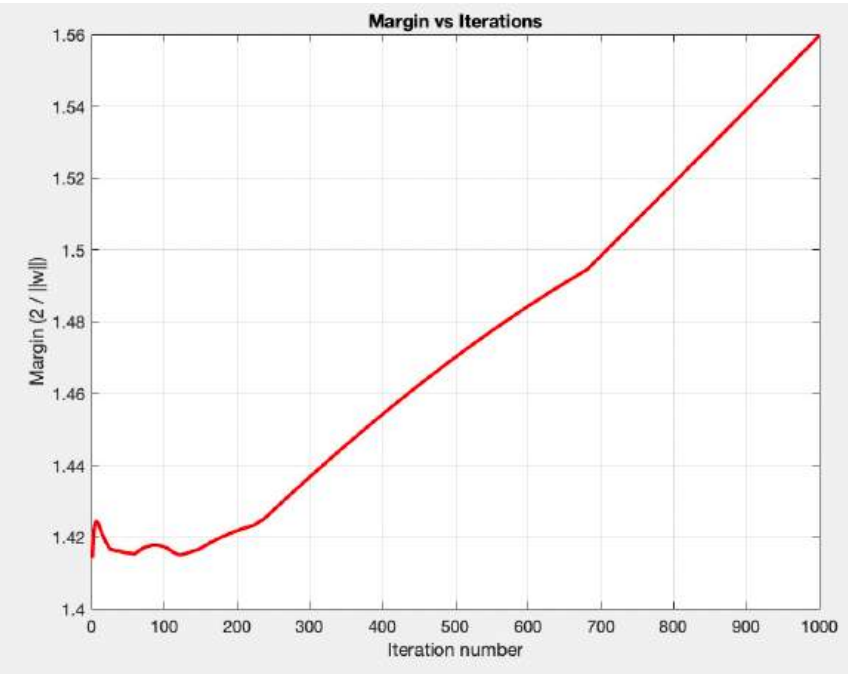
(c) Setting the step size $\mu = 10^{-4}$ and starting at $w^{(0)} = (-1, 1)$, run 1000 iterations of GD. You will create two plots.

i. Plot the classification error (averaged over all points) as a function of the iterations. The classification of x_i is determined by sign $(\langle x_i, w \rangle)$.

ii. Plot the margin $\frac{2}{\|w\|}$ as a function of the iterations. This shows how much of a gap you have between the classes you learned.

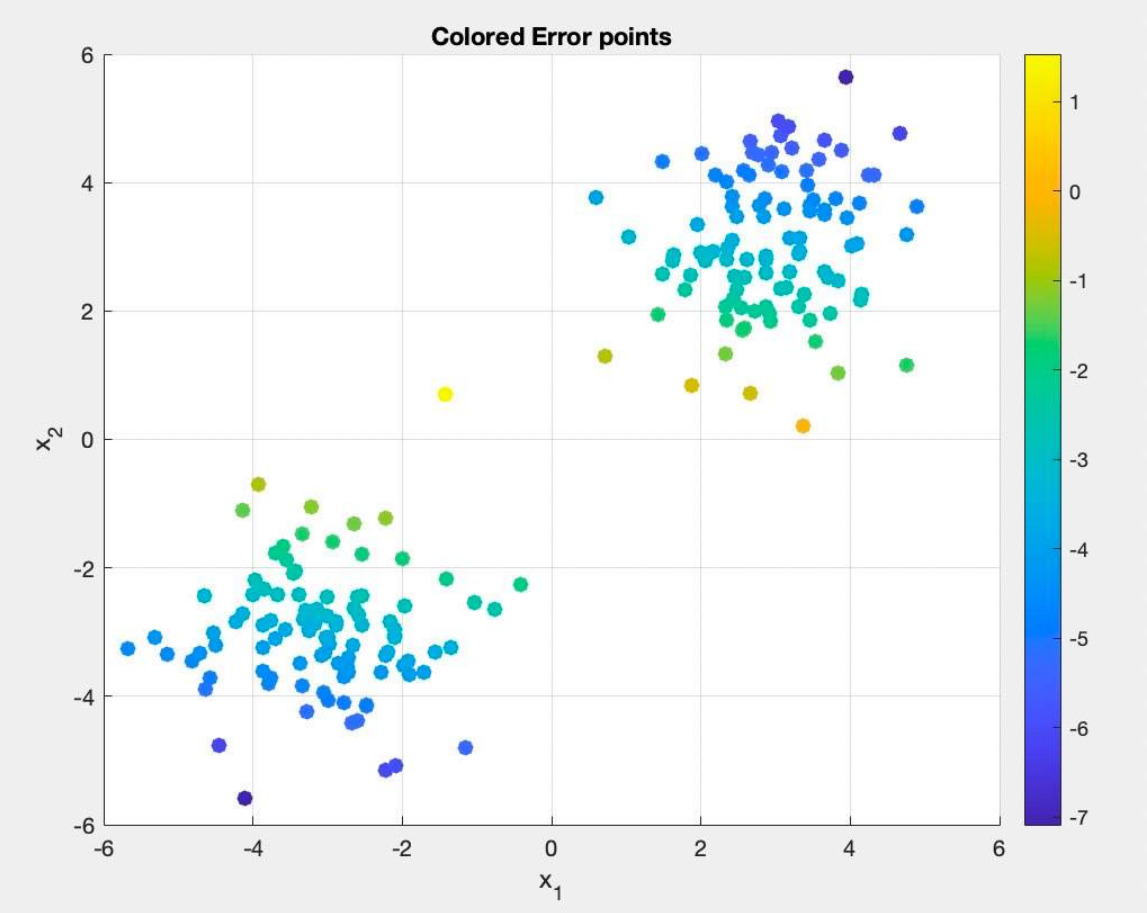
```m
data = readmatrix('HW7SVM.csv');
X = data(:, 1:2);
y = data(:, 3);
mu = 1e-4;
w0 = [-1; 1];
n_iter = 1000;
w = w0;
error = zeros(n_iter, 1);
margin = zeros(n_iter, 1);
for iter = 1:n_iter
    prediction = sign(X * w);
    misclassified = prediction ~= y;
    error(iter) = sum(misclassified) / length(y);

    w_norm = norm(w);
    if w_norm > 0
        margin(iter) = 2 / w_norm;
    else
        margin(iter) = Inf;
    end

    dot_products = X * w;
    margin_loss = 1 - y .* dot_products;

    gradient = grad(w, X, y);

    w = w - mu * gradient;
end


figure;
plot(1:n_iter, error, 'b-', 'LineWidth', 2);
xlabel('Iteration number');
ylabel('Classification Error');
title('Classification Error vs Iterations');
grid on;

figure;
plot(1:n_iter, margin, 'r-', 'LineWidth', 2);
xlabel('Iteration number');
ylabel('Margin (2 / ||w||)');
title('Margin vs Iterations');
grid on;
```





(d) Create another scatter plot of your data, but this time color the points by the function $f(x_i) = 1 - y_i \cdot \langle x_i, w \rangle$. The numbers closest to 0 (positive numbers or largest negative numbers) will show you which points were "most important" in determining the classification.

```
44    f_values = 1 - y .* (X * w);
45    figure;
46    scatter(X(:, 1), X(:, 2), 50, f_values, 'filled');
47    colorbar;
48    xlabel('x_1');
49    ylabel('x_2');
50    title('Colored Error points');
51    grid on;
52
```



Homework 6 for Math 173A - Fall 2025

For coding questions, you must submit your code and the requested answers. **Your code must be present to receive points.**

1. Consider the following optimization problem

$$\Phi(x) = \frac{1}{2} x^T \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} x \quad - \quad \sum_{i=1}^{2} x_i,$$

where $x \in \mathbb{R}^2$. Do the following by hand and clearly write out your steps.

   (a) Find the minimizer of the problem and show that the one you find is indeed the minimizer.

   (b) Is Φ L-smooth? Determine the smooth constant L.

   (c) Perform two steps of the Heavy ball method for Φ starting at $x^{(-1)} = x^{(0)} = (2, 2)$ with stepsize $\eta = 0.5$ and the parameter $\beta = 0.1$. Show your procedures and find what $x^{(2)}$ is.

   (d) Perform gradient descent with Nesterov's acceleration for Φ starting at $x^{(-1)} = x^{(0)} = (2, 2)$ with stepsize $\eta = 0.5$ and the parameter $\beta = 0.1$. Show your procedures and find what $x^{(2)}$ is.

   (e) Perform the conjugate gradient descent method (version 0 or 1), where the conjugate directions are initialized and chosen algorithmically, until it finds the minimizer. Show your procedures.

2. **Coding Questions**: Consider Question 4 from Homework 5. We will differentiate 4's and 9's again. This time, use the following two methods to minimize the loss F:

   (a) Gradient Descent with momentum

   (b) Gradient Descent with Nesterov acceleration

   Plot $F(w)$ per iteration for each. (you can experiment with the parameters until you find something you like)

3. **Coding Questions**: Let $A \in \mathbb{R}^{n \times n}$ be a diagonal matrix with diagonal entries

$$A_{ii} = i, \quad \text{i.e. the entries run from 1 to } n,$$

and let $b \in \mathbb{R}^n$ a vector with all 1 entries. Define the function

$$f(x) = \frac{1}{2}x^T Ax - b^T x.$$

We want to compare the convergence behavior of gradient descent and its accelerated versions . Do the following for $n = 20$ and $n = 100$ with initialization $x^{(0)} = 0$.

(a) Find the optimal solution x^* by solving $Ax = b$ using a Matlab/Python linear equation solver (or by hand and hard-coding the answer).

(b) Program and run the gradient descent method for f with a fixed stepsize. Run the method for 1000 iterations. You may experiment with the stepsize until you see something that works, or use a stepsize dictated by a theorem in the class.

(c) Program and run the heavy ball method for f. Run the method for 1000 iterations.

(d) Program and run gradient descent with Nesterov's acceleration for 1000 iterations.

For items (b) and (c) above, you can experiment with the parameters until you find something you like. Plot the $f(x^{(t)}) - f(x^*)$ for the three methods in the same figure. In a different figure, plot $\|x^{(t)} - x^*\|$ for the three methods. If you encounter a number smaller than 10^{-16}, set it to be 10^{-16}. In both plots, make the logarithmic scale for the vertical axis. Comment on the plots.

HW 6 SABELLA SAVLINO

1) Consider the following optimization problem

$$\Phi(x) = \frac{1}{2} x^T \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} x - \sum_{i=1}^{2} x_i \quad ,$$

where $x \in \mathbb{R}^2$. Do the following by hand and write out steps.

(a) Find the minimizer of the problem and show that the one you find is indeed the minimizer.

$$\Phi(x) = \frac{1}{2} [x_1, x_2] \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - (x_1 + x_2)$$

$$= \frac{1}{2} [2x_1, x_2] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - (x_1 + x_2)$$

$$= \frac{1}{2} (2x_1^2 + x_2^2) - (x_1 + x_2)$$

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

$$\nabla \Phi(x) = \begin{bmatrix} 2x_1 - 1 \\ x_2 - 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Rightarrow \begin{matrix} 2x_1 = 1 \Rightarrow x_1 = 1/2 \\ x_2 = 1 \end{matrix}$$

$$\Rightarrow x^* = \begin{bmatrix} 1/2 \\ 1 \end{bmatrix}$$

$$\nabla^2 \Phi(x) = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 2-\lambda & 0 \\ 0 & 1-\lambda \end{bmatrix} \Rightarrow \begin{matrix} \lambda = 2 \\ \lambda = 1 \end{matrix} \quad \text{both } (+)$$

Hessian is PSD $\Rightarrow \Phi$ is convex, so Φ has unique global minimizer $\Rightarrow x^* = \begin{bmatrix} 1/2 \\ 1 \end{bmatrix}$ is indeed the minimizer

(b) Is Φ L-smooth? Determine the smooth constant L.

$$\|\nabla \Phi(x) - \nabla \Phi(y)\| \le L\|x-y\| \quad \forall x, y \in \mathbb{R}^2$$

$$\nabla \Phi(x) = \begin{bmatrix} 2x_1 - 1 \\ x_2 - 1 \end{bmatrix} \qquad \nabla \Phi(y) = \begin{bmatrix} 2y_1 - 1 \\ y_2 - 1 \end{bmatrix}$$

$$= \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \end{bmatrix} \qquad = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$A \quad x \quad - \quad b \qquad\qquad A \quad y \quad - \quad b$$

$$\Rightarrow \|\nabla \Phi(x) - \nabla \Phi(y)\| = \|(Ax-b) - (Ay-b)\|$$

$$= \|Ax - Ay\|$$

$$= \|A(x-y)\|$$

$$\Rightarrow \|A(x-y)\| \le \lambda_{max}\|x-y\|$$

$$\lambda_{max}(A) = 2 \Rightarrow L = 2$$

Yes, Φ is L-smooth with smooth constant $L = 2$.

(c) Perform 2 steps of the Heavy Ball method for Φ starting at $x^{(-1)} = x^{(0)} = (2,2)$ with stepsize $\eta = 0.5$ and the parameter $\beta = 0.1$. Show your procedures and find what $x^{(2)}$ is.

$$x^{(t+1)} = x^{(t)} - \mu \nabla f(x^{(t)}) + \beta(x^{(t)} - x^{(t-1)}), \quad t \ge 1$$

$$x^{(1)} = (2,2) - 0.5\begin{bmatrix} 2(2) - 1 \\ 2 - 1 \end{bmatrix} + 0.1\left((2,2) - (2,2)\right)$$

$$= (2,2) - (3/2, 1/2) = (1/2, 3/2)$$

$$x^{(1)} = (1/2, 3/2)$$

$$x^{(2)} = (1/2, 3/2) - 0.5\begin{bmatrix} 2(1/2) - 1 \\ 3/2 - 1 \end{bmatrix} + 0.1\left((1/2, 3/2) - (2,2)\right)$$

$-3/2 = -1.5$

$-1/2 = -0.5$

$$= (1/2, 3/2) - (0, 1/4) + 0.1(-3/2, -1/2)$$

$$= (0.5, 1.25) + (-0.15, -0.05) = (0.35, 1.2)$$

$$x^{(2)} = (0.35, 1.2)$$

(d) Perform gradient descent with Nesterov's acceleration for ① starting at $x^{(-1)} = x^{(0)} = (2,2)$ with stepsize $\eta = 0.5$ and the parameter $\beta = 0.1$. Show your procedures and find what $x^{(2)}$ is.

$$y^{(t)} = x^{(t)} + \beta(x^{(t)} - x^{(t-1)})$$

$$x^{(t+1)} = y^{(t)} - \mu \nabla f(y^{(t)})$$

$$y^{(0)} = (2,2) + 0.1\cancel{((2,2) - (2,2))} = (2,2)$$

$$x^{(1)} = (2,2) - (3/2, 1/2) = (1/2, 3/2)$$

$$y^{(1)} = (1/2, 3/2) + 0.1((1/2, 3/2) - (2,2))$$

$$= (0.5, 1.5) + (-0.15, -0.05) = (0.35, 1.45)$$

$$x^{(2)} = (0.35, 1.45) - 0.5\begin{bmatrix} 2(0.35) - 1 \\ 1.45 - 1 \end{bmatrix}$$

$$= (0.35, 1.45) - (-0.15, 0.225) = (1/2, 1.225)$$

$$x^{(2)} = (1/2, 1.225)$$

(c) Perform the conjugate gradient descent method (version 0 or 1), where the conjugate directions are initialized and chosen algorithmically, until it finds the minimizer. Show your procedures.

$$x^{(t+1)} = x^{(t)} + \alpha_t P_t \;, \quad \alpha_t = \arg\min_{a \in \mathbb{R}} \Phi(x^{(t)} + a P_t)$$

need $\Phi(x) = \frac{1}{2} x^T A x - b^T x$: $A = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$, $b = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

Initialize $x^{(0)} = (2,2)$,

$$r_0 = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 2 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 1 \end{bmatrix} = \nabla \Phi(x^{(0)})$$

$$P_0 = \begin{bmatrix} -3 \\ -1 \end{bmatrix} \;, \quad x^{(t+1)} = x^{(t)} + \alpha_t P_t \quad \Rightarrow$$

$$\alpha_0 = \frac{\begin{bmatrix} 3 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ 1 \end{bmatrix}}{\begin{bmatrix} -3 & -1 \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} -3 \\ -1 \end{bmatrix}} = \frac{10}{19}$$

$$x^{(1)} = \begin{bmatrix} 2 \\ 2 \end{bmatrix} + \frac{10}{19} \begin{bmatrix} -3 \\ -1 \end{bmatrix} = \begin{bmatrix} 0.421 \\ 1.474 \end{bmatrix}$$

$$r_1 = \begin{bmatrix} 3 \\ 1 \end{bmatrix} + \frac{10}{19} \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} -3 \\ -1 \end{bmatrix} = \begin{bmatrix} 3 \\ 1 \end{bmatrix} + \begin{bmatrix} -60/19 \\ -10/19 \end{bmatrix} = \begin{bmatrix} -0.158 \\ 0.474 \end{bmatrix}$$

$$\beta_1 = \frac{\begin{bmatrix} -0.158 & 0.474 \end{bmatrix} \begin{bmatrix} -0.158 \\ 0.474 \end{bmatrix}}{\begin{bmatrix} 3 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ 1 \end{bmatrix}} = \frac{0.249}{10} = 0.0249$$

$$P_1 = \begin{bmatrix} 0.158 \\ -0.474 \end{bmatrix} + 0.0249 \begin{bmatrix} -3 \\ -1 \end{bmatrix} = \begin{bmatrix} 0.158 \\ -0.474 \end{bmatrix} + \begin{bmatrix} -0.075 \\ -0.0249 \end{bmatrix} = \begin{bmatrix} 0.083 \\ -0.499 \end{bmatrix}$$

$$\alpha_1 = \frac{0.249}{\begin{bmatrix}0.083 & -0.499\end{bmatrix}\begin{bmatrix}2 & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}0.083\\ -0.499\end{bmatrix}} = \frac{0.249}{0.263} = 0.95$$

$$x^{(2)} = \begin{bmatrix}0.421\\ 1.474\end{bmatrix} + 0.95\begin{bmatrix}0.083\\ -0.499\end{bmatrix}$$

$$= \begin{bmatrix}0.421\\ 1.474\end{bmatrix} + \begin{bmatrix}0.0786\\ -0.472\end{bmatrix} = \begin{bmatrix}0.4996\\ 1.002\end{bmatrix} \approx \begin{bmatrix}1/2\\ 1\end{bmatrix}$$

(rounding errors - $x^{(2)}$ does $= \begin{bmatrix}1/2\\ 1\end{bmatrix}$ by thm)

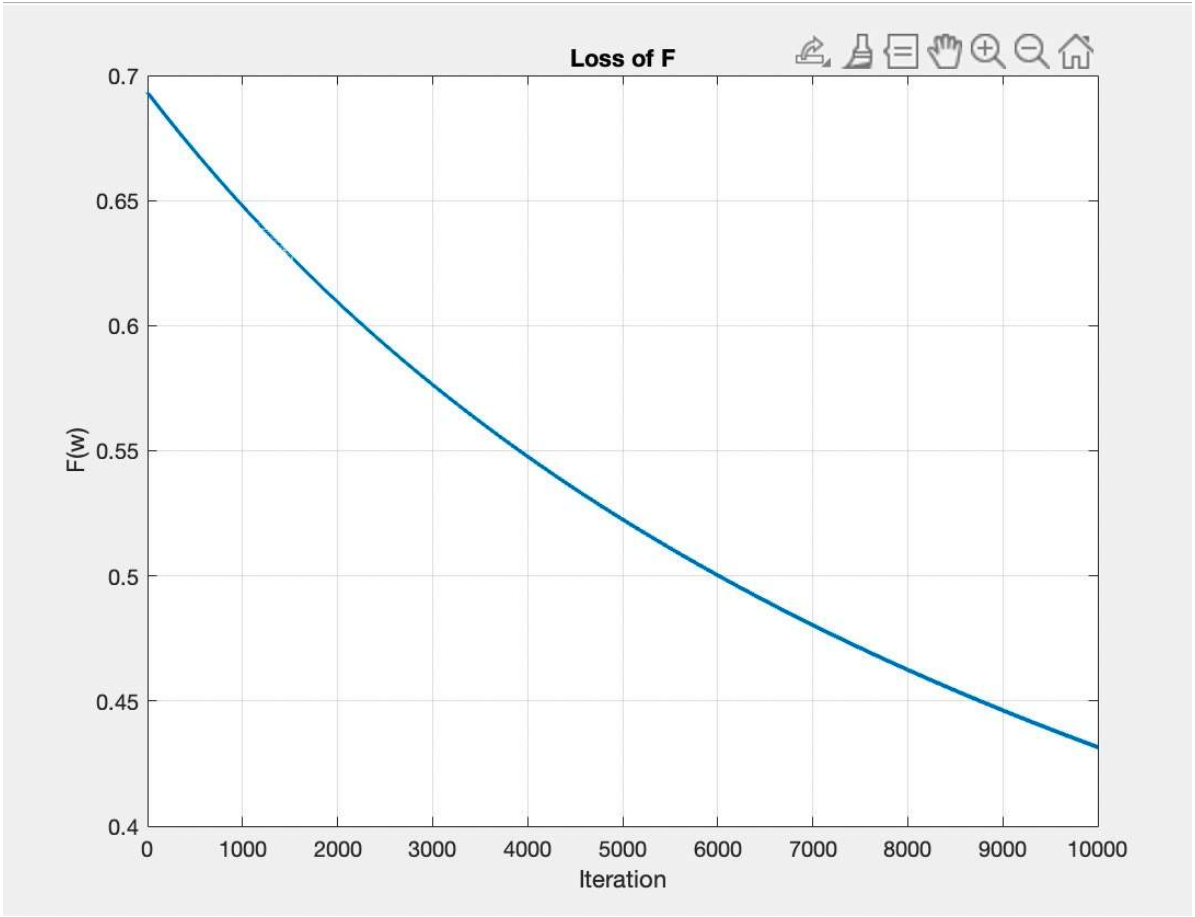
2) Consider Question 4 from Homework 5. We will differentiate Y's and q's again. This time, use the following methods to minimize the loss of F:

(a) Gradient Descent with momentum (HB)

Editor - /Users/sabella/Downloads/gd_momentum.m

form_data.m | mnist_train.csv | mnist_test.csv | gd_momentum.m | gd_NA.m

```
function w = gd_momentum()
    [x,y] = form_data();
    [N, d] = size(x);
    u = 1e-4;
    max_its = 10000;
    F_vals = zeros(max_its,1);
    w = zeros(d,1);
    beta = 0.1;
    momentum = zeros(d, 1);
    for i = 1:max_its
        inner_prod = x * w;
        n = length(y);
        denom = 1 + exp(y .* inner_prod);
        grad = -(x' * (y ./ denom)) / n;
        exp_term = exp(-y .* inner_prod);
        F_vals(i) = (1/N) * sum(log(1 + exp_term));

        momentum = beta * momentum - u * grad;
        w = w + momentum;
    end

    figure;
    plot(1:length(F_vals), F_vals, 'LineWidth', 2);
    xlabel('Iteration');
    ylabel('F(w)');
    title('Loss of F');
    grid on;

end
```

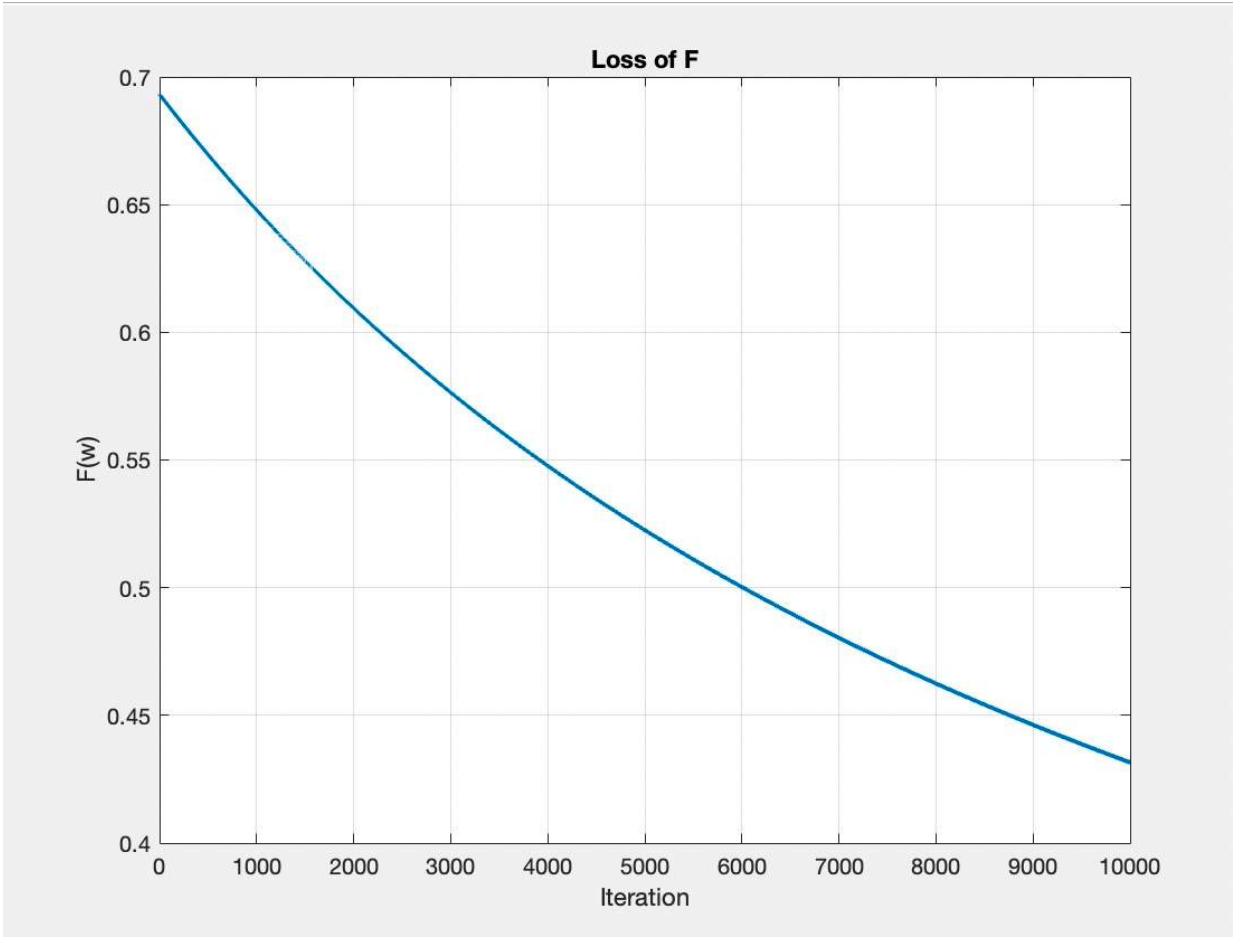


(b) Gradient Descent with Nesterov Acceleration

Editor - /Users/sabella/Downloads/gd_NA.m

form_data.m | mnist_train.csv | mnist_test.csv | gd_momentum.m | gd_NA.m

```
function w = gd_NA()
    [x,y] = form_data();
    [N, d] = size(x);
    u = 1e-4;
    max_its = 10000;
    F_vals = zeros(max_its,1);
    w = zeros(d,1);
    beta = 0.1;
    w_prev = w;
    for i = 1:max_its
        w_y = w + beta * (w - w_prev);
        inner_prod = x * w_y;
        n = length(y);
        denom = 1 + exp(y .* inner_prod);
        grad_lookahead = -(x' * (y ./ denom)) / n;
        inner_prod_current = x * w;
        exp_term_current = exp(-y .* inner_prod_current);
        F_vals(i) = (1/N) * sum(log(1 + exp_term_current));

        w_prev = w;
        w = w_y - u * grad_lookahead;
    end

    figure;
    plot(1:length(F_vals), F_vals, 'LineWidth', 2);
    xlabel('Iteration');
    ylabel('F(w)');
    title('Loss of F');
    grid on;
end
```



3) Let $A \in \mathbb{R}^{n \times n}$ be a diagonal matrix with diagonal entries $A_{ii} = i$, i.e. the entries run from 1 to n, and let $b \in \mathbb{R}^n$ be a vector with all 1 entries. Define

$$f(x) = \frac{1}{2} x^T A x - b^T x.$$

We want to compare the convergence of gradient descent and its accelerated versions. Do the following for $n = 20$ and $n = 100$ with initialization $x^{(0)} = 0$.

(a) Find the optimal solution x^* by solving $Ax = b$ using a Matlab/Python linear equation solver (or by hand and hard-coding the answer).

Editor - /Users/sabella/Downloads/untitled4.m

form_data.m | mnist_train.csv | mnist_test.csv | gd_momentum.m | gd_NA.m | untitled4.m

```
1   n1 = 20;
2   A1 = diag(1:n1);
3   b1 = ones(n1,1);
4   x_star_1 = A1 \ b1;
5
6   n2 = 100;
7   A2 = diag(1:n2);
8   b2 = ones(n2,1);
9   x_star_2 = A2 \ b2;
10
11  fprintf('x* for n = 20: \n')
12  disp(x_star_1)
13  fprintf('x* for n = 100: \n')
14  disp(x_star_2)
```

```
Command Window
>> untitled4
x* for n = 20:
    1.0000
    0.5000
    0.3333
    0.2500
    0.2000
    0.1667
    0.1429
    0.1250
    0.1111
    0.1000
    0.0909
    0.0833
    0.0769
    0.0714
    0.0667
    0.0625
    0.0588
    0.0556
    0.0526
    0.0500
```

```
x* for n = 100:
    1.0000
    0.5000
    0.3333
    0.2500
    0.2000
    0.1667
    0.1429
    0.1250
    0.1111
    0.1000
    0.0909
    0.0833
    0.0769
    0.0714
    0.0667
    0.0625
    0.0588
    0.0556
    0.0526
    0.0500
    0.0476
    0.0455
    0.0435
    0.0417
    0.0400
    0.0385
    0.0370
    0.0357
    0.0345
    0.0333
    0.0323
    0.0312
```

```
0.0303
0.0294
0.0286
0.0278
0.0270
0.0263
0.0256
0.0250
0.0244
0.0238
0.0233
0.0227
0.0222
0.0217
0.0213
0.0208
0.0204
0.0200
0.0196
0.0192
0.0189
0.0185
0.0182
0.0179
0.0175
0.0172
0.0169
0.0167
0.0164
0.0161
0.0159
0.0156
0.0154
0.0152
```

```
0.0149
0.0147
0.0145
0.0143
0.0141
0.0139
0.0137
0.0135
0.0133
0.0132
0.0130
0.0128
0.0127
0.0125
0.0123
0.0122
0.0120
0.0119
0.0118
0.0116
0.0115
0.0114
0.0112
0.0111
0.0110
0.0109
0.0108
0.0106
0.0105
0.0104
0.0103
0.0102
0.0101
0.0100
```

(please forgive the strange layout - trying to save space)

(b) Program and run the GD method for f with a fixed stepsize. Run the method for 1000 iterations. You may experiment with the stepsize until you see something that works, or use a stepsize dictated by a theorem in class.

Editor - /Users/sabella/Downloads/gd_hw6.m

+4 | gd_NA.m | untitled4.m | gd_hw6.m | untitled6 * | untitled7 * | untitled8 *

```
1  function x = gd_hw6()
2      n = 20;
3      max_its = 1000;
4      A = diag(1:n);
5      b = ones(n, 1);
6      x = zeros(n, 1);
7      f_vals = zeros(max_its, 1);
8      L = n;
9      u = 1 / L;
10     for i = 1:max_its
11         grad = A * x - b;
12         f_vals(i) = 0.5 * x' * A * x - b' * x;
13         x = x - u * grad;
14     end
15     disp(x);
16 end
```

```
>> gd_hw6  (n = 20)
    1.0000
    0.5000
    0.3333
    0.2500
    0.2000
    0.1667
    0.1429
    0.1250
    0.1111
    0.1000
    0.0909
    0.0833
    0.0769
    0.0714
    0.0667
    0.0625
    0.0588
    0.0556
    0.0526
    0.0500
```

```
>> gd_hw6  (n = 100)
    1.0000
    0.5000
    0.3333
    0.2500
    0.2000
    0.1667
    0.1429
    0.1250
    0.1111
    0.1000
    0.0909
    0.0833
    0.0769
    0.0714
    0.0667
    0.0625
    0.0588
    0.0556
    0.0526
    0.0500
    0.0476
    0.0455
    0.0435
    0.0417
    0.0400
    0.0385
    0.0370
    0.0357
    0.0345
    0.0333
    0.0323
    0.0312
```

```
0.0303
0.0294
0.0286
0.0278
0.0270
0.0263
0.0256
0.0250
0.0244
0.0238
0.0233
0.0227
0.0222
0.0217
0.0213
0.0208
0.0204
0.0200
0.0196
0.0192
0.0189
0.0185
0.0182
0.0179
0.0175
0.0172
0.0169
0.0167
0.0164
0.0161
0.0159
0.0156
0.0154
0.0152
```

```
0.0149
0.0147
0.0145
0.0143
0.0141
0.0139
0.0137
0.0135
0.0133
0.0132
0.0130
0.0128
0.0127
0.0125
0.0123
0.0122
0.0120
0.0119
0.0118
0.0116
0.0115
0.0114
0.0112
0.0111
0.0110
0.0109
0.0108
0.0106
0.0105
0.0104
0.0103
0.0102
0.0101
0.0100
```

(c) Program and run the HB method for f. Run the method for 1000 iterations.

Editor - /Users/sabella/Downloads/gd_HB.m *

mnist_test.csv | untitled4.m | gd_hw6.m | gd_HB.m * | untitled7 * | untitled8 *

```
function x = gd_HB()
    n = 20;
    max_its = 1000;
    A = diag(1:n);
    b = ones(n, 1);
    x = zeros(n, 1);
    f_vals = zeros(max_its, 1);
    L = n;
    u = 1 / L;
    beta = 0.1;
    momentum = zeros(n, 1);
    for i = 1:max_its
        grad = A * x - b;
        f_vals(i) = 0.5 * x' * A * x - b' * x;
        momentum = beta * momentum - u * grad;
        x = x + momentum;
    end
    disp(x);
end
```

(n = 20)

```
>> gd_HB
    1.0000
    0.5000
    0.3333
    0.2500
    0.2000
    0.1667
    0.1429
    0.1250
    0.1111
    0.1000
    0.0909
    0.0833
    0.0769
    0.0714
    0.0667
    0.0625
    0.0588
    0.0556
    0.0526
    0.0500
```

(n = 100)

```
>> gd_HB
    1.0000
    0.5000
    0.3333
    0.2500
    0.2000
    0.1667
    0.1429
    0.1250
    0.1111
    0.1000
    0.0909
    0.0833
    0.0769
    0.0714
    0.0667
    0.0625
    0.0588
    0.0556
    0.0526
    0.0500
    0.0476
    0.0455
    0.0435
    0.0417
    0.0400
    0.0385
    0.0370
    0.0357
    0.0345
    0.0333
    0.0323
    0.0312
    0.0303
    0.0294
    0.0286
    0.0278
    0.0270
    0.0263
    0.0256
    0.0250
    0.0244
    0.0238
    0.0233
    0.0227
    0.0222
    0.0217
    0.0213
    0.0208
    0.0204
    0.0200
    0.0196
    0.0192
    0.0189
    0.0185
    0.0182
    0.0179
    0.0175
    0.0172
    0.0169
    0.0167
    0.0164
    0.0161
    0.0159
    0.0156
    0.0154
    0.0152
    0.0149
    0.0147
    0.0145
    0.0143
    0.0141
    0.0139
    0.0137
    0.0135
    0.0133
    0.0132
    0.0130
    0.0128
    0.0127
    0.0125
    0.0123
    0.0122
    0.0120
    0.0119
    0.0118
    0.0116
    0.0115
    0.0114
    0.0112
    0.0111
    0.0110
    0.0109
    0.0108
    0.0106
    0.0105
    0.0104
    0.0103
    0.0102
    0.0101
    0.0100
```

(d) Program and run GD with NA for 1000 iterations.

Editor - /Users/sabella/Downloads/gd_NA_2.m

form_data.m | mnist_train.csv | mnist_test.csv | untitled4.m | gd_hw6.m | gd_HB.m

```
function x = gd_NA_2()
    n = 20;
    max_its = 1000;
    A = diag(1:n);
    b = ones(n, 1);
    x = zeros(n, 1);
    f_vals = zeros(max_its, 1);
    L = n;
    u = 1 / L;
    beta = 0.1;
    x_prev = x;
    for i = 1:max_its
        y = x + beta*(x-x_prev);
        grad = A * y - b;
        f_vals(i) = 0.5 * x' * A * x - b' * x;

        x_prev = x;
        x = y - u*grad;
    end
    disp(x);
end
```

(n = 20)

```
>> gd_NA_2
    1.0000
    0.5000
    0.3333
    0.2500
    0.2000
    0.1667
    0.1429
    0.1250
    0.1111
    0.1000
    0.0909
    0.0833
    0.0769
    0.0714
    0.0667
    0.0625
    0.0588
    0.0556
    0.0526
    0.0500
```

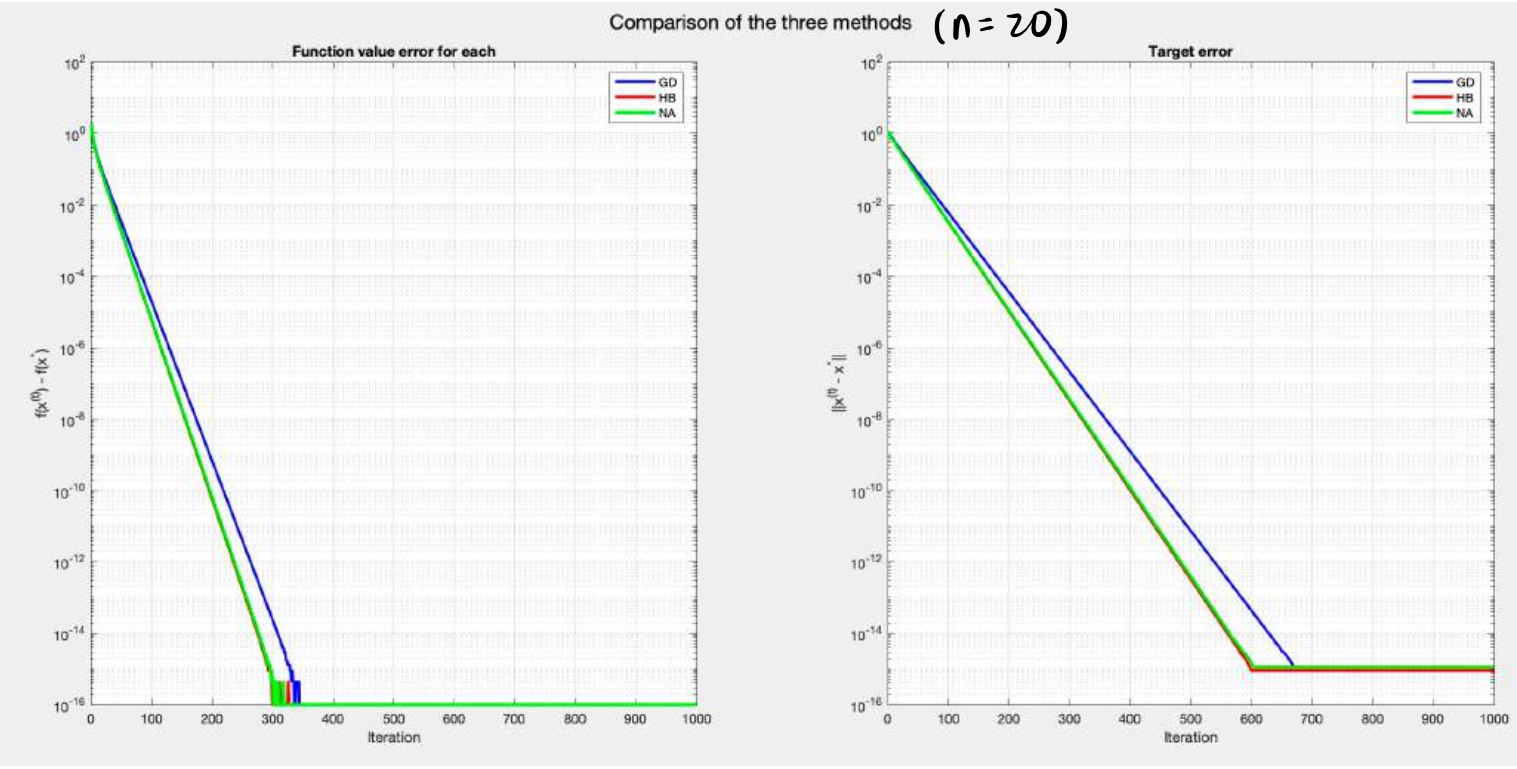
(n = 100)

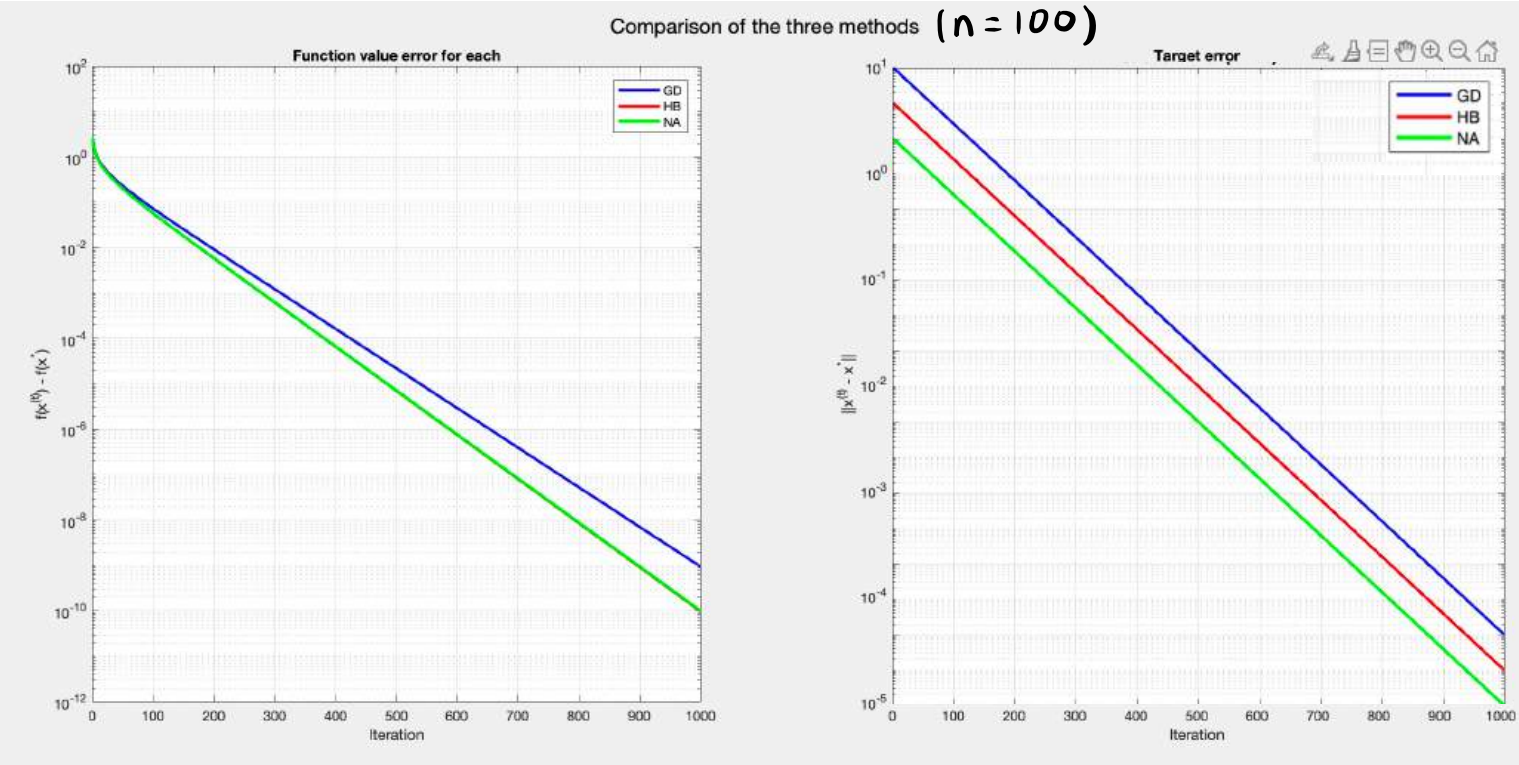
```
>> gd_NA_2
    1.0000
    0.5000
    0.3333
    0.2500
    0.2000
    0.1667
    0.1429
    0.1250
    0.1111
    0.1000
    0.0909
    0.0833
    0.0769
    0.0714
    0.0667
    0.0625
    0.0588
    0.0556
    0.0526
    0.0500
    0.0476
    0.0455
    0.0435
    0.0417
    0.0400
    0.0385
    0.0370
    0.0357
    0.0345
    0.0333
    0.0323
    0.0312
    0.0303
```

```
    0.0294
    0.0286
    0.0278
    0.0270
    0.0263
    0.0256
    0.0250
    0.0244
    0.0238
    0.0233
    0.0227
    0.0222
    0.0217
    0.0213
    0.0208
    0.0204
    0.0200
    0.0196
    0.0192
    0.0189
    0.0185
    0.0182
    0.0179
    0.0175
    0.0172
    0.0169
    0.0167
    0.0164
    0.0161
    0.0159
    0.0156
    0.0154
    0.0152
    0.0149
```

```
    0.0147
    0.0145
    0.0143
    0.0141
    0.0139
    0.0137
    0.0135
    0.0133
    0.0132
    0.0130
    0.0128
    0.0127
    0.0125
    0.0123
    0.0122
    0.0120
    0.0119
    0.0118
    0.0116
    0.0115
    0.0114
    0.0112
    0.0111
    0.0110
    0.0109
    0.0108
    0.0106
    0.0105
    0.0104
    0.0103
    0.0102
    0.0101
    0.0100
```

For items (b) and (c) above, you can experiment with the parameters until you find something you like. Plot the $f(x^{(t)}) - f(x^*)$ for the three methods in the same figure. In a different figure, plot $\|x^{(t)} - x^*\|$ for the three methods. If you encounter a number smaller than 10^{-16}, set it to be 10^{-16}. In both plots, make the logarithmic scale for the vertical axis. Comment on the plots.





For n=20, we see that the regular GD has the highest error during the iterate, for both $f(x^{(t)})$ and $x^{(t)}$, and then rounds out to about the same as HB and NA.

For n=100, the actual value of the target error $\|x^{(t)} - x^*\|$ has a clear spread with NA proving the least error, regular GD proving the most, and HB exactly in the middle. Again, for the function value, NA and HB are almost exactly the same,

while regular GD provides slightly more error.

Editor - /Users/sabella/Downloads/plot_three_methods.m *

form_data.m | mnist_train.csv | mnist_test.csv | untitled4.m | gd_hw6.m | gd_HB.m

```
function plot_three_methods()
    n = 100;
    max_its = 1000;

    A = diag(1:n);
    b = ones(n, 1);
    x_star = A \ b;
    f_star = 0.5 * x_star' * A * x_star - b' * x_star;

    gd_hw6();
    gd_HB();
    gd_NA_2();

    gd_data = load('gd_hw6.mat');
    hb_data = load('gd_HB.mat');
    na_data = load('gd_NA_2.mat');

    f_gd = gd_data.f_history;
    f_hb = hb_data.f_history;
    f_na = na_data.f_history;

    f_gap_gd = max(f_gd - f_star, 1e-16);
    f_gap_hb = max(f_hb - f_star, 1e-16);
    f_gap_na = max(f_na - f_star, 1e-16);

    error_gd = zeros(max_its, 1);
    error_hb = zeros(max_its, 1);
    error_na = zeros(max_its, 1);

    for i = 1:max_its
        error_gd(i) = max(norm(gd_data.x_history(i, :)' - x_star), 1e-16);
        error_hb(i) = max(norm(hb_data.x_history(i, :)' - x_star), 1e-16);
        error_na(i) = max(norm(na_data.x_history(i, :)' - x_star), 1e-16);
    end

    figure('Position', [100, 100, 1200, 500]);

    subplot(1, 2, 1);
    semilogy(1:length(f_gap_gd), f_gap_gd, 'b-', 'LineWidth', 2); hold on;
    semilogy(1:length(f_gap_hb), f_gap_hb, 'r-', 'LineWidth', 2);
    semilogy(1:length(f_gap_na), f_gap_na, 'g-', 'LineWidth', 2);
    xlabel('Iteration');
    ylabel('f(x^{(t)}) - f(x^*)');
    title(sprintf('Function value error for each'));
    legend('GD', 'HB', 'NA', 'Location', 'best');
    grid on;

    subplot(1, 2, 2);
    semilogy(1:length(error_gd), error_gd, 'b-', 'LineWidth', 2); hold on;
    semilogy(1:length(error_hb), error_hb, 'r-', 'LineWidth', 2);
    semilogy(1:length(error_na), error_na, 'g-', 'LineWidth', 2);
    xlabel('Iteration');
    ylabel('||x^{(t)} - x^*||');
    title(sprintf('Target error'));
    legend('GD', 'HB', 'NA', 'Location', 'best');
    grid on;

    sgtitle(sprintf('Comparison of the three methods'));
end
```

Homework 5 for Math 173A - Fall 2025

For coding questions, you must submit your code and the requested answers. **Your code must be present to receive points.**

1. Consider the function $f(x_1, x_2) = (2x_1 - 1)^4 + (x_1 + x_2 - 1)^2$. Recall the minimizer is $(\frac{1}{2}, \frac{1}{2})$ for this problem.

   (a) Starting at $x^{(0)} = (0, 0)$, perform two steps of Newton's method to find $x^{(2)}$. Show your work.

   (b) Can you apply the theorem of convergence for Newton's method learned in the lecture to this problem, even if the initial $x^{(0)}$ is very close to $(\frac{1}{2}, \frac{1}{2})$? Hint: check whether the Hessian at x^* is invertible.

   (c) Consider the function $g(x_1, x_2) = (2x_1 - 1)^2 + (x_1 + x_2 - 1)^2$. Redo the subquestion (a) for this function g.

   Bonus question: If you start Newton's method at $x^{(0)} = (0, 0)$ for the function f, does Newton's method converge to x^*? Justify your answer.

2. **Coding Question:** We'll consider the function $f(x_1, x_2) = 200(x_2 - x_1^2)^2 + (1 - x_1)^2$.

   (a) Write a computer program to run Newton's method on this problem.

   (b) Write a computer program to run Gradient Descent with a fixed step size $\mu = 10^{-3}$ on this problem.

   (c) Write a computer program to run Gradient Descent with backtracking line-search for this problem (you may set β and γ as you wish).

   (d) Starting at the same $x^{(0)}$, run each of the four algorithms and plot $\|x^{(t)} - x^*\|$ for each, in the same figure. In a separate figure, plot $f(x^{(t)}) - f(x^*)$ for each of them. Comment on the performance. Note that the minimizer $x^* = (1, 1)$ for this problem.

3. **Coding Question:** We will redo the MNIST coding question but **for differentiating 4's and 9's**. You can reuse the template for this problem.

   (a) Repeat parts (b), (c), and (e) from Q4 of HW4 for digits of 4s and 9s. Comment on the difference between the results and propose a reason as to why the performance did or did not change.

(b) Implement and run gradient descent with **backtracking line search**. At each iteration t, initialize the step size $\mu = 10^{-1}$, and use $\gamma = 0.5$ and $\beta = 0.8$ to determine the correct $\mu^{(t)}$. Run your algorithm for at least 10,000 iterations and plot the loss curve $F(w^{(t)})$ as a function of t.

4. **Coding Question:** Again, we will redo the MNIST coding question from HW4 but using different versions of gradient descent

$$w^{(t+1)} = w^{(t)} - \mu p^{(t)}.$$

This time, we will **differentiate 4's and 9's**. That is, instead of classifying images of 0 and 1, you classify the images of 4 and 9. You can reuse the template from the previous homework for loading / formatting MNIST.

(a) Implement and run L^∞ gradient descent with step size $\mu = 10^{-8}$. Run your algorithm for at least 10^4 iterations and initialize with $w^{(0)} = 0$ (i.e. the zero vector). **Recall:** L^∞ gradient descent uses steps

$$p^{(t)} = \text{sign}(\nabla F(w^{(t)}))\|\nabla F(w^{(t)})\|_1.$$

(b) Implement and run L^1 gradient descent (aka. coordinate descent) with step size $\mu = 10^{-4}$. Run your algorithm for at least 10^4 iterations and initialize with $w^{(0)} = 0$ (i.e. the zero vector). **Recall:** L^1 gradient descent uses steps

$$p^{(t)} = \text{sign}(\nabla_{j^*} F(w^{(t)}))\|\nabla F(w^{(t)}\|_\infty e_{j^*},$$

where j^* is the location of the largest entry of the gradient, and e_j is the zero vector with a 1 in the j^{th} entry.

(c) Compare these two descent plots of $F(w)$, along with the analogous plot for gradient descent from HW4. Which performs best, and do you have an argument for why? Do you think the performance would change with different step sizes?

(d) For the coordinate descent problem, rerun gradient descent but store a running sum of which entry of $p^{(t)}$ is nonzero at each iteration (not the actual value of the direction vector, just e_{j^*}). This will result in a size 784 vector of mostly zeros, and should have integers at various entries whose sum equals the number of iterations. Reshape this vector to be a 28x28 image and display the result. Why do you think these are the pixels that were chosen in the gradient? How can you use this to interpret the algorithm and its results?

HW 5 SABRINA SALVINO

1) Consider the function $f(x_1, x_2) = (2x_1 - 1)^4 + (x_1 + x_2 - 1)^2$. Recall the minimizer is $(1/2, 1/2)$ for this problem.

(a) Starting at $x^{(0)} = (0,0)$, perform two steps of Newton's method to find $x^{(2)}$.

$$x^{(t+1)} = x^{(t)} - [\nabla^2 f(x^{(t)})]^{-1} \nabla f(x^{(t)})$$

$$\nabla f(x_1, x_2) = \left(8(2x_1 - 1)^3 + 2(x_1 + x_2 - 1),\ 2(x_1 + x_2 - 1)\right)^T$$

$$\nabla^2 f(x_1, x_2) = \begin{bmatrix} 48(2x_1 - 1)^2 + 2 & 2 \\ 2 & 2 \end{bmatrix}$$

$$\nabla f(0,0) = \left(8(0-1)^3 + 2(0+0-1),\ 2(0+0-1)\right) = (-10, -2)^T$$

$$\nabla^2 f(0,0) = \begin{bmatrix} 48(0-1)^2 + 2 & 2 \\ 2 & 2 \end{bmatrix} = \begin{bmatrix} 50 & 2 \\ 2 & 2 \end{bmatrix}$$

$$(\nabla^2 f(0,0))^{-1} = \frac{1}{100 - 4} \begin{bmatrix} 2 & -2 \\ -2 & 50 \end{bmatrix} = \frac{1}{96} \begin{bmatrix} 2 & -2 \\ -2 & 50 \end{bmatrix}$$

$$x^{(1)} = (0,0)^T - \frac{1}{96} \begin{bmatrix} 2 & -2 \\ -2 & 50 \end{bmatrix} \begin{bmatrix} -10 \\ -2 \end{bmatrix}$$

$$= (0,0)^T - \frac{1}{96} \begin{bmatrix} -16 \\ -80 \end{bmatrix} = \frac{1}{96} \begin{bmatrix} 16 \\ 80 \end{bmatrix} = \begin{bmatrix} 1/6 \\ 5/6 \end{bmatrix}$$

$$\nabla f(1/6, 5/6) = \left(8(2(1/6) - 1)^3 + 2(1/6 + 5/6 - 1),\ 2(1/6 + 5/6 - 1)\right)^T$$

$$= \left(8(-2/3)^3, 0\right) = \begin{bmatrix} -64/27 \\ 0 \end{bmatrix}$$

$$\nabla^2 f(1/6, 5/6) = \begin{bmatrix} 48(2(1/6)-1)^2+2 & 2 \\ 2 & 2 \end{bmatrix} = \begin{bmatrix} 48(4/9)+2 & 2 \\ 2 & 2 \end{bmatrix} = \begin{bmatrix} 70/3 & 2 \\ 2 & 2 \end{bmatrix}$$

$$\left(\nabla^2 f(1/6, 5/6)\right) = \frac{1}{140/3 - 4} \begin{bmatrix} 2 & -2 \\ -2 & 70/3 \end{bmatrix} = \frac{3}{128} \begin{bmatrix} 2 & -2 \\ -2 & 70/3 \end{bmatrix}$$

$$x^{(2)} = \begin{bmatrix} 1/6 \\ 5/6 \end{bmatrix} - \frac{3}{128} \begin{bmatrix} 2 & -2 \\ -2 & 70/3 \end{bmatrix} \begin{bmatrix} -64/27 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} 1/6 \\ 5/6 \end{bmatrix} - \frac{3}{128} \begin{bmatrix} -128/27 \\ 128/27 \end{bmatrix} = \begin{bmatrix} 1/6 \\ 5/6 \end{bmatrix} - \begin{bmatrix} -1/9 \\ 1/9 \end{bmatrix} = \begin{bmatrix} 5/18 \\ 13/18 \end{bmatrix}$$

(b) Can you apply the theorem of convergence for Newton's method learned in lecture to this problem, even if the initial $x^{(0)}$ is very close to $(1/2, 1/2)$? Hint: check whether Hessian at x^* is invertible.

$$\nabla^2 f(x^*) = \nabla^2 f(1/2, 1/2) = \begin{bmatrix} 48(2(1/2)-1)^2+2 & 2 \\ 2 & 2 \end{bmatrix} = \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}$$

$$\left(\nabla^2 f(x^*)\right)^{-1} = \frac{1}{4-4} \begin{bmatrix} 2 & -2 \\ -2 & 2 \end{bmatrix} = \text{not invertible},$$

So no, we cannot apply the theorem of convergence for Newton's method, even if initial $x^{(0)}$ is very close to $(1/2, 1/2)$.

(c) Consider the function $g(x_1, x_2) = (2x_1 - 1)^2 + (x_1 + x_2 - 1)^2$. Redo (a) for this function g.

$$x^{(t+1)} = x^{(t)} - \left[\nabla^2 g(x^{(t)})\right]^{-1} \nabla g(x^{(t)})$$

$$\nabla g(x_1, x_2) = (10x_1 + 2x_2 - 6,\ 2x_1 + 2x_2 - 2)^T$$

$$\nabla^2 g(x_1, x_2) = \begin{bmatrix} 10 & 2 \\ 2 & 2 \end{bmatrix}$$

$$\nabla g(0,0) = (-6, -2)^T \qquad \nabla^2 g(0,0) = \begin{bmatrix} 10 & 2 \\ 2 & 2 \end{bmatrix}$$

$$(\nabla^2 g(0,0))^{-1} = \frac{1}{20-4} \begin{bmatrix} 2 & -2 \\ -2 & 10 \end{bmatrix} = \frac{1}{16} \begin{bmatrix} 2 & -2 \\ -2 & 10 \end{bmatrix}$$

$$x^{(1)} = (0,0) - \frac{1}{16} \begin{bmatrix} 2 & -2 \\ -2 & 10 \end{bmatrix} \begin{bmatrix} -6 \\ -2 \end{bmatrix} = -\frac{1}{16} \begin{bmatrix} -8 \\ -8 \end{bmatrix} = \begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix} = x^*$$

2) Consider the function $f(x_1, x_2) = 200(x_2 - x_1^2)^2 + (1 - x_1)^2$

(a)

Editor - /Users/sabella/Downloads/Newton.m *

Newton.m *

```
function x_min = Newton(x0, tol, max_its)
    x = x0;
    i = 0;

    while i < max_its
        x1 = x(1);
        x2 = x(2);
        g1 = -400 * x1 * (x2 - x1^2) - 2 * (1 - x1);
        g2 = 400 * (x2 - x1^2);
        g = [g1; g2];
        if norm(g) < tol
            break;
        end
        H11 = 1200 * x1^2 - 400 * x2 + 2;
        H12 = -400 * x1;
        H21 = H12;
        H22 = 400;
        H = [H11, H12; H21, H22];

        d = -H \ g;
        x = x + d;
        i = i + 1;
    end
    x_min = x;
end
```

(b)

Editor - /Users/sabella/Downloads/GradDesc2.m

Newton.m | GradDesc2.m

```
function x_min = GradDesc2(x0, tol, max_its)
    x = x0;
    i = 0;
    u = 0.001;
    while i < max_its
        x1 = x(1);
        x2 = x(2);
        g1 = -400 * x1 * (x2 - x1^2) - 2 * (1 - x1);
        g2 = 400 * (x2 - x1^2);
        g = [g1; g2];
        if norm(g) < tol
            break;
        end
        x = x - u*g;
        i = i + 1;
    end
    x_min = x;
end

```

(c)

Editor - /Users/sabella/Downloads/gd_backtrack.m

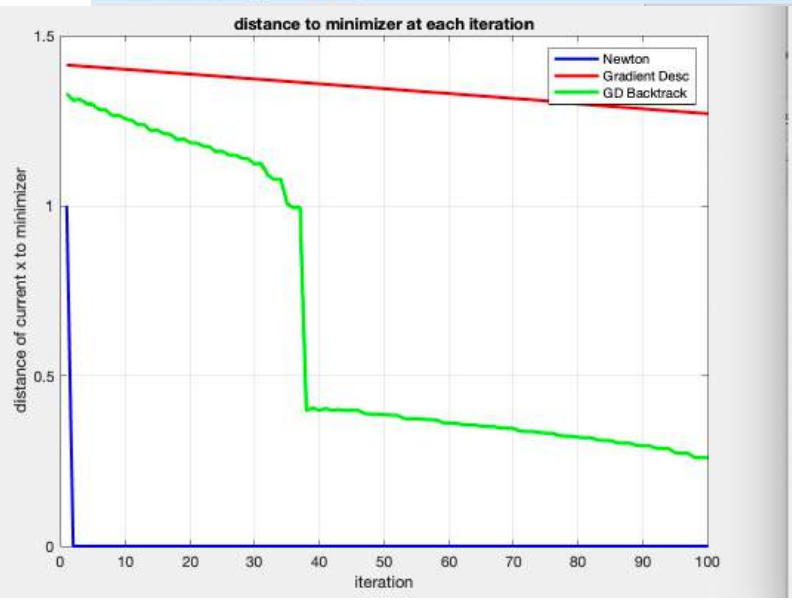
Newton.m | GradDesc2.m | gd_backtrack.m

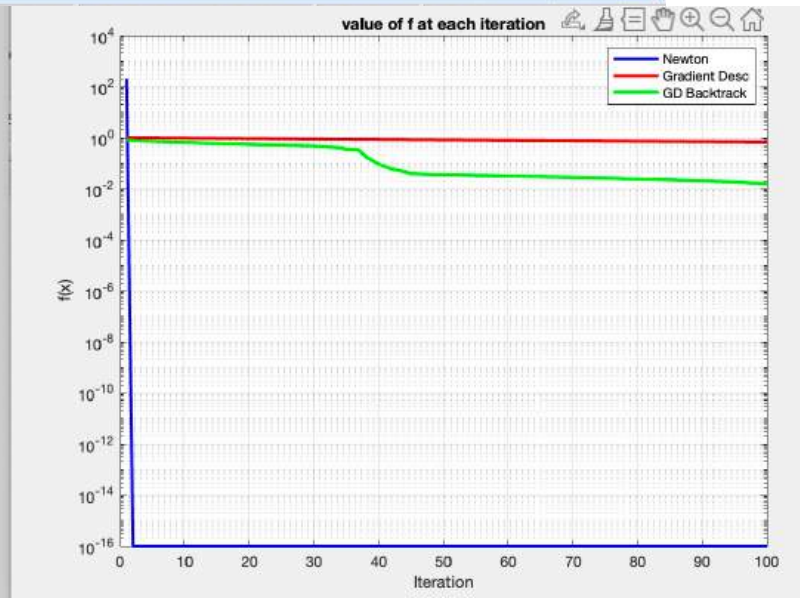
```
function x_min = gd_backtrack(x0, max_its)
    beta = 0.5;
    gamma = 0.1;
    x = x0;
    for i = 1:max_its
        x1 = x(1);
        x2 = x(2);
        g1 = -400 * x1 * (x2 - x1^2) - 2 * (1 - x1);
        g2 = 400 * (x2 - x1^2);
        g = [g1; g2];
        alpha = 1.0;
        f_curr = 200*(x2-x1^2)^2 + (1-x1)^2;

        while true
            x_new = x - alpha * g;
            x1_new = x_new(1);
            x2_new = x_new(2);
            f_new = 200*(x2_new-x1_new^2)^2 + (1-x1_new)^2;
            if f_new <= f_curr - gamma * alpha * (g' * g)
                break;
            end
            alpha = beta * alpha;
        end
        x = x - alpha * g;
    end
    x_min = x;
end
```

(d) Starting at the same $x^{(0)}$, run each of the four algs and plot $\|x^{(t)} - x^*\|$ for each, in the same figure. In a separate figure, plot $f(x^{(t)}) - f(x^*)$ for each of them. Comment on performance. Note that minimizer $x^* = (1, 1)$ for this problem.

```
clear; clc;
x_star = [1; 1];
x0 = [0; 0];
max_its = 100;

d_newton = zeros(1, max_its);
d_GradDesc2 = zeros(1, max_its);
d_gd_backtrack = zeros(1, max_its);

newton_vals = zeros(1, max_its);
GradDesc2_vals = zeros(1, max_its);
gd_backtrack_vals = zeros(1, max_its);

x = x0;
for i = 1:max_its
    x = Newton(x, 1e-6, 1);
    d_newton(i) = norm(x - x_star);
    x1 = x(1);
    x2 = x(2);
    newton_vals(i) = 200*(x2-x1^2)^2 + (1-x1)^2;
end

x = x0;
for i = 1:max_its
    x = GradDesc2(x, 1e-6, 1);
    d_GradDesc2(i) = norm(x - x_star);
    x1 = x(1);
    x2 = x(2);
    GradDesc2_vals(i) = 200*(x2-x1^2)^2 + (1-x1)^2;
end
```

```
x = x0;
for i = 1:max_its
    x = gd_backtrack(x, 1);
    d_gd_backtrack(i) = norm(x - x_star);
    x1 = x(1);
    x2 = x(2);
    gd_backtrack_vals(i) = 200*(x2-x1^2)^2 + (1-x1)^2;
end

it = 1:max_its;
figure(1);
plot(it, d_newton, 'b-', 'LineWidth', 2);
hold on;
plot(it, d_GradDesc2, 'r-', 'LineWidth', 2);
plot(it, d_gd_backtrack, 'g-', 'LineWidth', 2);
legend('Newton', 'Gradient Desc', 'GD Backtrack');
xlabel('iteration');
ylabel('‖x - x*‖');
title('distance to minimizer');
grid on;

figure(2);
semilogy(it, max(newton_vals, 1e-16), 'b-', 'LineWidth', 2);
hold on;
semilogy(it, GradDesc2_vals, 'r-', 'LineWidth', 2);
semilogy(it, gd_backtrack_vals, 'g-', 'LineWidth', 2);
legend('Newton', 'Gradient Desc', 'GD Backtrack');
xlabel('Iteration');
ylabel('f(x)');
title('value of f at each iteration');
grid on;
```





The plots show that Newton's method converges the quickest, converging in about 2 iterations. Then neither GD nor GD with backtracking converge to the minimizer [1,1] in 100 iterations, showing that both are slower to converge. GD with backtracking has a slightly faster than GD alone, but still slow, rate of convergence. GD alone has an extremely slow rate of convergence. Looking at the plot on the right, we can visualize the actual function values at each iteration as we supposedly get closer to the minimum, and the slopes for GD and GD with backtracking

3) We will redo the MNIST coding question but for differentiating 4s and 9s. Can reuse template for problem.

(a) Repeat parts (b), (c), and (e) from Q4 of HW4 for digits of 4s and 9s. Comment on the difference between the results and propose a reason as to why the performance did or did not change.

Command Window

Editor - form_data.m

form_data.m | mnist_train.csv | mnist_test.csv | untitled2 *

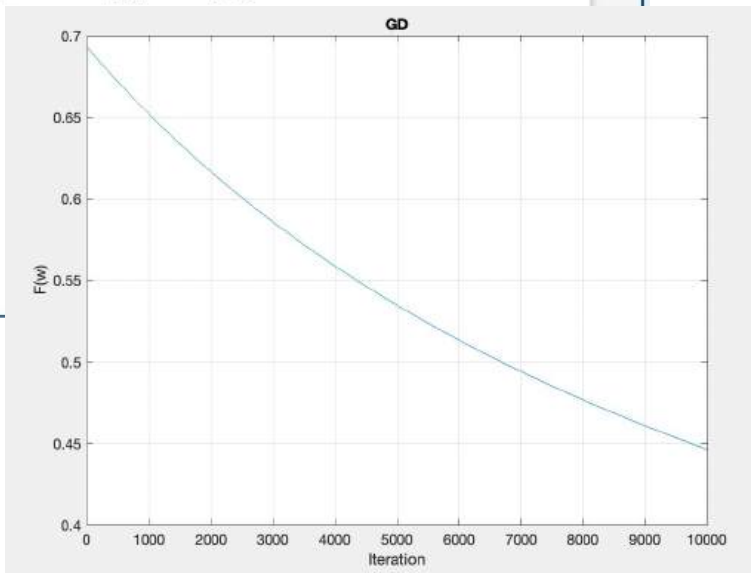
```
function [x,y] = form_data()
    data = readmatrix('mnist_train.csv');
    labels = table2array(data(:,1));
    pixels = table2array(data(:,2:end)) / 255.0;

    fours_idx = find(labels == 4, 500);
    nines_idx = find(labels == 9, 500);
    x = [pixels(fours_idx,:); pixels(nines_idx,:)];
    y = [ones(500,1); -ones(500,1)];
end
```

Command Window

Editor - GD_hw5.m

+1 | form_data.m | mnist_train.csv | mnist_test.csv | GD_hw5.m | GradDesc2.m

```
function w = GD_hw5()
    [x,y] = form_data();
    [N, d] = size(x);
    u = 1e-4;
    max_its = 10000;
    F_vals = zeros(max_its,1);
    w = zeros(d,1);

    for i = 1:max_its
        inner_prod = x*w;
        exp_term = exp(-y .* inner_prod);
        log_term = exp_term ./ (1 + exp_term);
        grad = -(1/N) * (x' * (y .* log_term));
        w = w - u * grad;
        F_vals(i) = (1/N) * sum(log(1 + exp_term));
    end
    figure;
    plot(1:max_its, F_vals);
    xlabel('Iteration');
    ylabel('F(w)');
    title('GD');
    grid on;
end
```



```
24      %added to end of GD file from previous part
25      test_data = readtable('mnist_test.csv');
26      test_labels = table2array(test_data(:,1));
27      test_pixels = table2array(test_data(:,2:end)) / 255.0;
28      fours_idx_test = find(test_labels == 4, 500);
29      nines_idx_test = find(test_labels == 9, 500);
30      x_test = [test_pixels(fours_idx_test,:); test_pixels(nines_idx_test,:)];
31      y_test = [ones(500,1); -ones(500,1)];
32      classify_test = x_test * w;
33      test_pred = sign(classify_test);
34      test_err = sum(test_pred ~= y_test) / length(y_test);
35      train_pred = sign(x_test * w);
36      train_err = sum(train_pred ~= y_test) / length(y_test);
37
38      fprintf('Training error rate: %.4f\n', train_err);
39      fprintf('Test error rate: %.4f\n', test_err);
40  end
```

```
>> GD_hw5
Training error rate: 0.0810
Test error rate: 0.1260
```

Classification error rates shown above. As you can see, the error in the test data is slightly greater than the error in the training data, which is expected and usual, since the GD model was trained with the training data. The difference in errors for each data set is nothing alarming, its a small difference that is explained by the above. Both error percentages are relatively low. The performance in this run, when looking for distinctions between 4s and 9s, rather than 0s and 1s, is maybe not as good as the previous model. Meaning, the model is not as good at distinguishing between 4s and 9s as it is at distinguishing between 0s and 1s. However, this model had a more expected relationship between the training and testing errors, with the training error less than testing error, which the first model did not (possibly an issue with my implementation).

(b) Implement and run GD with backtracking line search. At each iteration t, initialize the step size $\mu = 10^{-1}$, and use $\gamma = 0.5$ and $\beta = 0.8$ to determine the correct $\mu^{(t)}$. Run your algorithm for at least 10000 iterations and plot the loss curve $F(w^{(t)})$ as a function of t.

Editor - /Users/sabella/Downloads/GD_backtracking_hw5.m

+1 | GD_hw5.m | mnist_train.csv | mnist_test.csv | GD_backtracking_hw5.m | +

```
function w = GD_backtracking_hw5()
    [x,y] = form_data();
    [N, d] = size(x);
    max_its = 10000;
    beta = 0.8;
    gamma = 0.5;
    F_vals = zeros(max_its,1);
    w = zeros(d,1);

    for i = 1:max_its
        inner_prod = x * w;
        exp_term = exp(-y .* inner_prod);
        F_current = (1/N) * sum(log(1 + exp_term));
        log_term = exp_term ./ (1 + exp_term);
        grad = -(1/N) * (x' * (y .* log_term));
        u = 0.1;
        while true
            w_new = w - u * grad;
            inner_prod_new = x * w_new;
            exp_term_new = exp(-y .* inner_prod_new);
            F_new = (1/N) * sum(log(1 + exp_term_new));
            if F_new <= F_current - gamma * u * (grad' * grad)
                break;
            end
            u = beta * u;
        end
        w = w - u * grad;
        F_vals(i) = F_current;
    end
    figure;
    plot(1:max_its, F_vals, 'LineWidth', 2);
    xlabel('Iteration');
    ylabel('F(w^{(t)})');
    title('GD Backtracking');
    grid on;
end
```

4) Again, we will redo the MNIST coding question from HW4 but using different versions of GD $w^{(t+1)} = w^{(t)} - \mu p^{(t)}$.
This time, we will differentiate 4s and 9s (we classify images of 4 and 9).

(a) Implement and run L^∞ GD with step size $\mu = 10^{-8}$. Run your alg for at least 10^4 iterations and initialize with $w^{(0)} = 0$.
Recall: L^∞ GD uses steps $p^{(t)} = \text{sign}(\nabla F(w^{(t)})) \| \nabla F(w^{(t)}) \|_1$.

Editor - /Users/sabella/Downloads/L_inf_GD.m

form_data.m | mnist_train.csv | mnist_test.csv | L_inf_GD.m

```
function w = L_inf_GD()
    [x,y] = form_data();
    [N, d] = size(x);
    max_its = 10000;
    u = 1e-8;
    F_vals = zeros(max_its,1);
    w = zeros(d,1);

    for i = 1:max_its
        inner_prod = x * w;
        exp_term = exp(-y .* inner_prod);
        log_term = exp_term ./ (1 + exp_term);
        grad = -(1/N) * (x' * (y .* log_term));
        grad_L1_norm = norm(grad, 1);
        p = sign(grad) * grad_L1_norm;

        w = w - u * p;
        F_vals(i) = (1/N) * sum(log(1 + exp_term));
    end
    figure;
    plot(1:max_its, F_vals, 'LineWidth', 2);
    xlabel('Iteration');
    ylabel('F(w^{(t)})');
    title('L (infinity) GD');
    grid on;
end
```

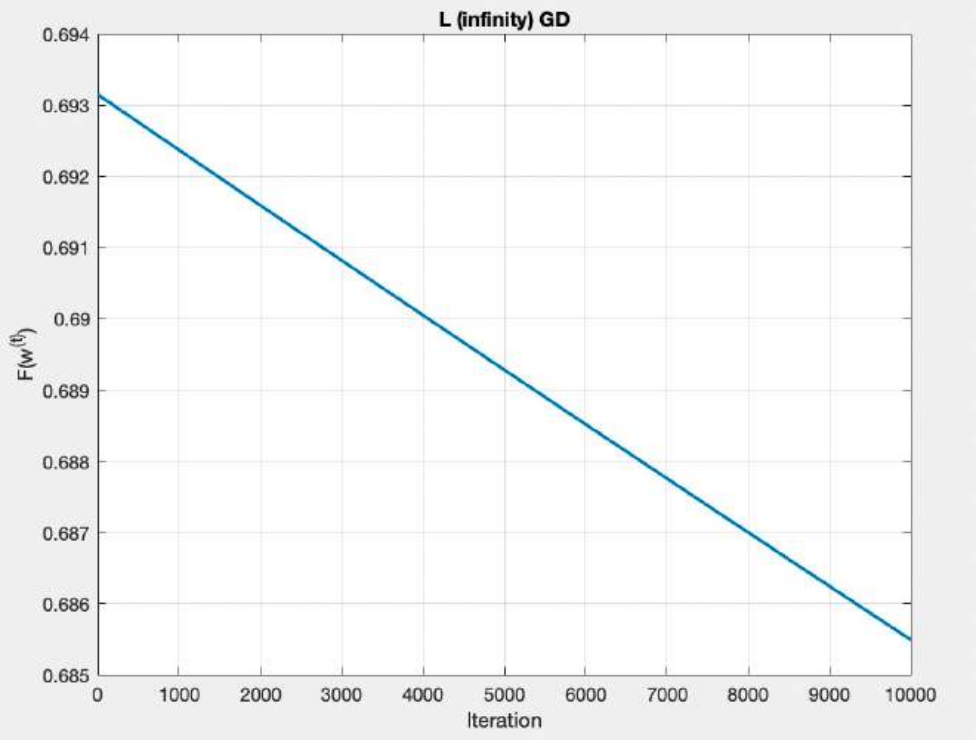
(b) Implement and run L^2 GD with step size $\mu = 10^{-4}$. Run your alg for at least 10^4 iterations and initialize with $w^{(0)} = 0$.
Recall: L^2 GD uses steps $p^{(t)} = \text{sign}(\nabla_{j^*} F(w^{(t)})) \| \nabla F(w^{(t)}) \|_\infty e_{j^*}$, where j^* is the location of the largest entry of the gradient, and e_j is the zero vector with a 1 in the jth entry.

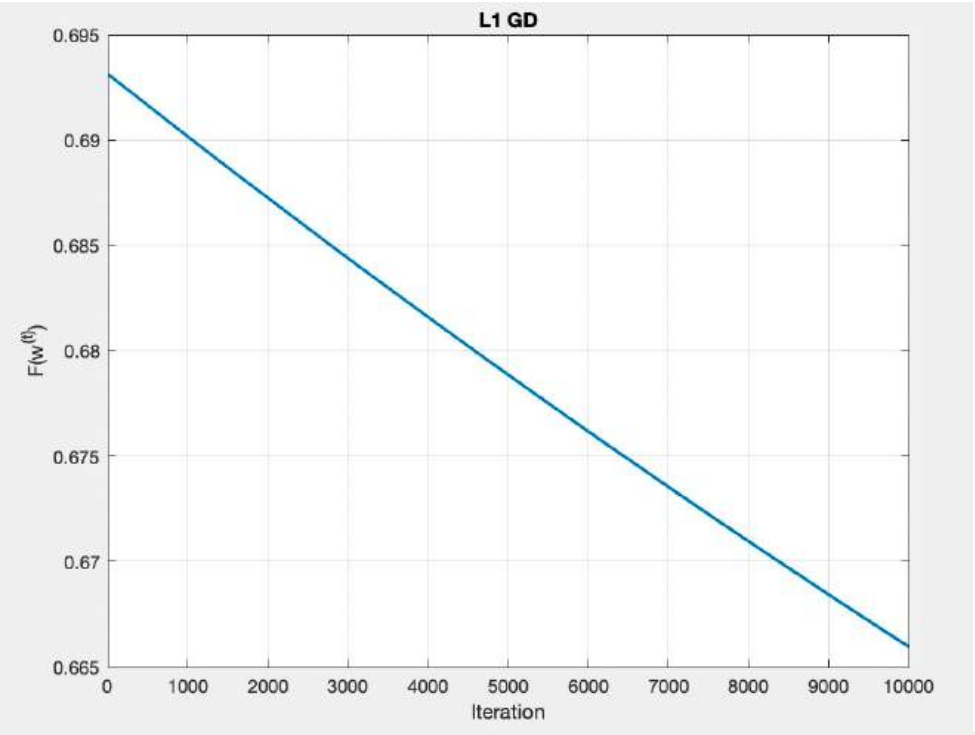
Editor - /Users/sabella/Downloads/L1_GD.m *

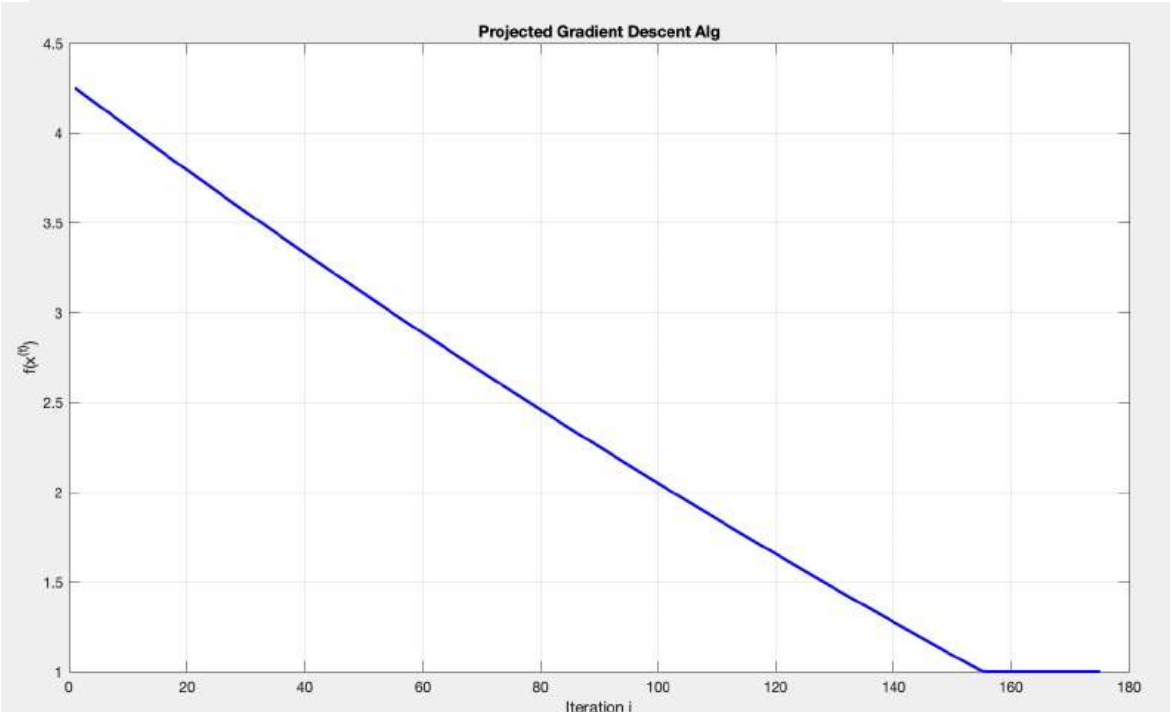
form_data.m | mnist_train.csv | mnist_test.csv | L_inf_GD.m | L1_GD.m *

```
function w = L1_GD()
    [x,y] = form_data();
    [N, d] = size(x);
    max_its = 10000;
    mu = 1e-4;
    F_vals = zeros(max_its,1);
    w = zeros(d,1);

    for i = 1:max_its
        inner_prod = x * w;
        exp_term = exp(-y .* inner_prod);
        log_term = exp_term ./ (1 + exp_term);
        grad = -(1/N) * (x' * (y .* log_term));

        [~, j_star] = max(abs(grad));
        grad_inf_norm = norm(grad, inf);
        p = zeros(d,1);
        p(j_star) = sign(grad(j_star)) * grad_inf_norm;

        w = w - mu * p;
        F_vals(i) = (1/N) * sum(log(1 + exp_term));
    end
    figure;
    plot(1:max_its, F_vals, 'LineWidth', 2);
    xlabel('Iteration');
    ylabel('F(w^{(t)})');
    title('L1 GD');
    grid on;
end
```

(c) Compare these two descent plots of F(w), along with the analogous plot for GD from HW4. Which performs best, and do you have an argument for why? Do you think performance would change with different step sizes?







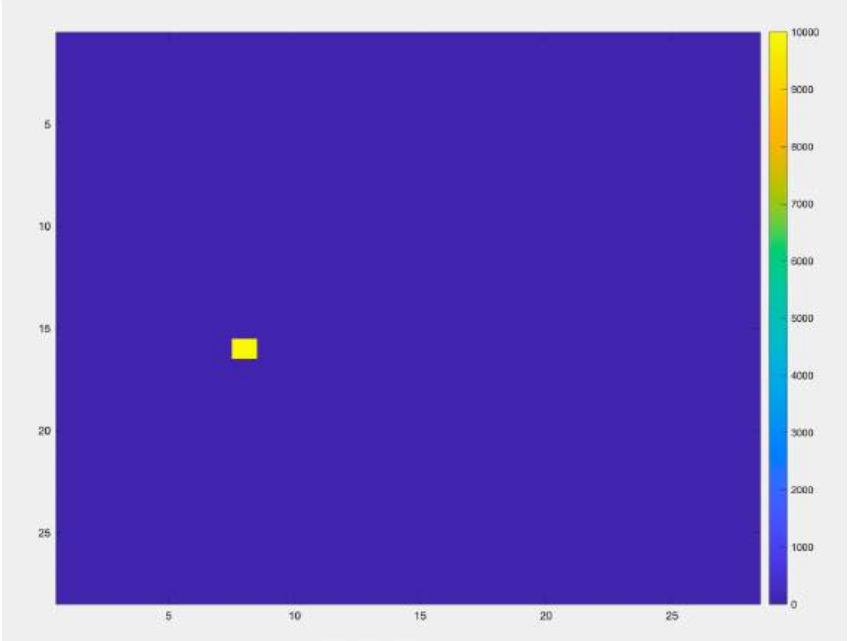
The GD implementation from HW4 appears to perform the best, and an argument for why could be that the regular GD always moves in the steepest direction of descent. L^∞ GD moves depending on each coordinate sign, which can possibly "confuse" the descent. L^2 GD moves one coordinate at a time in the direction of j^*, which could be a problem since it doesn't take into account all coordinates. Yes, I do think performance would change with different step sizes, because both L^∞ and L^2 look to maybe have too small of a step and are converging too slowly.

All GDs rely on step size that essentially scale $\hat{p}^{(t)}$ in the update. So, a larger step may lead to faster convergence for L^2 & L^∞ GD. Clearly, the analogous GD converges fastest by a large margin.

(d) For the coordinate descent problem, rerun GD but store a running sum of which entry of $p^{(t)}$ is nonzero at each iteration (not the actual value of the direction vector, just e_{j^*}). This will result in a size 784 vector of mostly zeros, and should have integers at various entries whose sum equals the number of iterations. Reshape the vector to be a 28 × 28 image and display the result. Why do you think these are the pixels that were chosen in the gradient? How can you use this to interpret the alg and its result?

mnist_train.csv | mnist_test.csv | L_inf_GD.m | L1_GD.m | GradDesc.m | L1_GD_sum.m *

```
function coordinates = L1_GD_sum()
    [x,y] = form_data();
    [N, d] = size(x);
    max_its = 10000;
    u = 1e-4;
    w = zeros(d,1);
    coordinates = zeros(d,1);

    for i = 1:max_its
        inner_prod = x * w;
        exp_term = exp(-y .* inner_prod);
        log_term = exp_term ./ (1 + exp_term);
        grad = -(1/N) * (x' * (y .* log_term));
        [~, j_star] = max(abs(grad));
        grad_inf_norm = norm(grad, inf);
        p = zeros(d,1);
        p(j_star) = sign(grad(j_star)) * grad_inf_norm;

        w = w - u * p;
        coordinates(j_star) = coordinates(j_star) + 1;
    end
    figure;
    vector_image = reshape(coordinates, 28, 28);
    imagesc(vector_image);
    colorbar;
end
```



From the image, the coords chosen were from rows ~15-17 and columns ~7-9. I think that these coords was chosen due to the magnitude of their gradient being the largest, meaning that these coords are the most influential in helping the model distinguish between 4s and 9s. This probably means that the algorithm repeatedly looks at this pixel in the data sets in order to help make decisions since it has the largest magnitude of gradient. Since our alg was L_1 GD, only one coord. updates at each iteration, so it makes sense that we just have one pixel for a small number of coordinates that get the large update from the large magnitude of gradient. So, the model sort of relies on a few coordinates and a small chunk of data to make its decisions, so it may not have the most accurate differentiation.

Homework 4 for Math 173A - Fall 2025

For coding questions, you must submit your code and the requested answers. **Your code must be present to receive points.**

1. (a) Find an expression for the orthogonal projection of a point $x \in \mathbb{R}^n$ onto the convex set

$$B = \{z \in \mathbb{R}^n : 0 \leq z_i \leq 1 \text{ for each } i = 1, ..., n\}.$$

You need to show your work, and justify your answer. The expression can be written piecewise, and per dimension if it's easier / more compact.
Hint: It might be helpful to sketch B, when $n = 2$ (i.e., in 2 dimensions), and use the sketch to help you figure out what the projection should be.

(b) Let $f : \mathbb{R}^n \to \mathbb{R}$ be given by

$$f(x) = \|Ax\|_2^2 + a^T x$$

where $A \in \mathbb{R}^{n \times n}$ is a positive definite matrix, and $a \in \mathbb{R}^n$. Write a projected gradient descent algorithm to solve

$$\min_{x \in \Omega} f(x)$$

for $\Omega = B$, with B from part (a). You do not need to specify the step size for this problem.

(c) Repeat part (b) but for $\Omega = B_2^n = \{z \in \mathbb{R}^n : \|z\|_2 \leq 1\}$.

2. Consider the *hollow* sphere S in $\mathbb{R}^n$, i.e., the set $S := \{x \in \mathbb{R}^n : \|x\|_2^2 = 1\}$. Consider the function $f : \mathbb{R}^n \to \mathbb{R}$ given by

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

where Q is an $n \times n$ symmetric matrix. For this problem you may use the fact that $\nabla f(x) = 2Qx$.

(a) For an arbitrary point $y \in \mathbb{R}^n$, $\Pi(y)$ be the projection of y onto S. Find an expression for $\Pi(y)$ and give a short argument (i.e., proof) for why this is the correct expression. Make sure to handle the case $y = 0$ (i.e., the zero vector).
Hint: Recall that the projection minimizes $\|x - y\|$ for $y \in S$. One approach would be to consider the reverse triangle inequality $\|x - y\| \geq |\|x\| - \|y\||$ and find a projection formula that achieves the global lower bound (i.e. equality instead of inequality). This would prove you've found the projection.

(b) Is S a convex set?

(c) Write a projected gradient descent algorithm, with constant step size μ, for

$$\min_{x \in \mathbb{R}^n} x^T Q x \qquad \text{subject to} \qquad \|x\|_2^2 = 1.$$

(d) Is the projected gradient descent algorithm guaranteed to converge to the solution for small enough μ? If not, can you give an example of Q and an initialization $x^{(0)}$ where the algorithm won't converge? **Hint:** Consider a diagonal matrix Q where not all entries are equal.

3. **Coding Question:** Consider the function $f : \mathbb{R}^2 \to \mathbb{R}$ with $f(x_1, x_2) = (x_1 - 2)^2 + (x_2 - 2)^2 - x_1 x_2$ and the following constrained optimization problem:

$$\min_{x_1, x_2} f(x_1, x_2) \quad \text{subject to} \quad 0 \le x_i \le 1, \ i = 1, 2.$$

Program a projected gradient descent algorithm, with constant step size $\mu = 0.001$ starting at $(0.5, 0.5)$ for 175 iterations, for the above optimization problem. Submit your code and plot the function value $f(x^{(t)})$ against the iteration t.

4. **Coding Question** For this question and the next question, you will need to download the MNIST data set. It is a data set of images of size 28x28 pixels. Each one is an image of a handwritten digit from 0-9. You can find some description of the dataset here: `https://en.wikipedia.org/wiki/MNIST_database`.

We've uploaded a Jupyter notebook template to the HW folder (HW4_Q4.ipynb), and it provides a basic set of code for downloading / formatting MNIST and blocks for where to put your code / results / answers. You may need to twist the code a bit if it does not work for you.

We have also provided you directly the training and testing datasets, `mnist_train.csv` in the Assignment page. The `mnist_train.csv` file contains the 60,000 training examples and labels. The `mnist_test.csv` contains 10,000 test examples and labels. Each row consists of 785 values: the first value is the label (a number from 0 to 9) and the remaining 784 values are the pixel values (a number from 0 to 255).

In HW2 you considered the classification problem with logistic regression

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

You also wrote down a gradient descent algorithm for it.

(a) Display one randomly selected image from your data (either the one from the template if you are using the template, or the training dataset `mnist_train.csv` if you directly use the dataset we provided) for each digit class. Provide the index number for each image.

(b) Continuing (a), select the first 500 examples of 0's and 1's for this example, those will form the training data $(x_i, y_i) \in \mathbb{R}^{784} \times \{-1, 1\}$, $i = 1, ..., 1000$. Assign label $y_i = 1$ for 1s and $y_i = -1$ for 0s. Also, renormalize your x_i so that the pixel values are floats between 0 and 1, instead of ints from 0 to 255.

Note: To get from images of size 28×28 pixels to vectors in $\mathbb{R}^{784}$, you just need to "vectorize" the image. This means you can concatenate each of the 28 columns of the original image into one long vector of length 784. In Matlab, this is done with the command $x(:)$, simllarly in Python it's numpy.vectorize(x) or x.reshape().

(c) Implement and run a Gradient Descent algorithm, with step-size $\mu = 10^{-4}$, to optimize the function (1) associated with this setup. You should run your algorithm for at least $T = 10,000$ iterations, but if your computer can handle it, try $T = 100,000$ or choose a reasonable stopping criterion. Provide a plot showing the value of $F(w)$ at each iteration. Also, feel free to adjust μ to be larger / smaller if the plot does not match your expectations.

(d) Comment on the resulting plot. In particular, does the shape of $F(w)$ suggest you've successfully converged to a local or global minimum? Does it appear you chose a good stopping criteria? Explain whether your answers to these questions are consistent with the theory we discussed in class (and in the notes). Be specific i.e., point to a specific theorem (or theorems) and indicate why it does or does not explain the behavior of the algorithm. Would the theory dictate a different choice of μ than the one we used?

(e) Now, use the w you found from part (c) to classify the next 500 data points associated to each of the 0 and 1 handwritten digits if you use the template we provided. The next 500 data points serve as test data. The idea is that you deploy your model on data you didn't use to train the model, as a check of its quality. If you use the datasets we provided, then you use the w you found from part (c) to classify the first 500 *test data points* associated to each of the 0 and 1 handwritten digits in `mnist_test.csv`. Recall that you need to use the function $y = \text{sign}(w^T x)$ to classify. What was the classification error rate associated with the two digits on the test data (this should be a number between 0 and 1)? What was it on the training data? Does this relationship make sense?

HW 4

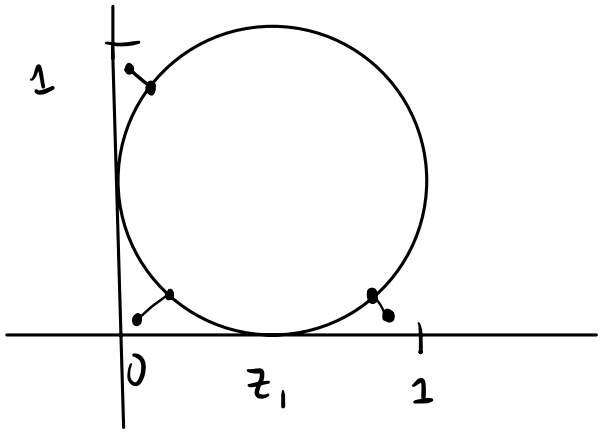
Sabella Savlind

1) (a) Find an expression for the orthogonal projection of a point $x \in \mathbb{R}^n$ onto the convex set

$$B = \{z \in \mathbb{R}^n : 0 \le z_i \le 1 \text{ for each } i = 1, \dots, n\}$$

Show work, justify. Expression can be written piecewise and per dimension if it's easier/more compact. Hint: sketch B, when $n = 2$ (2 dimensions), use sketch to figure out projection should be

When $n = 2$: $B = \{z \in \mathbb{R}^2 : 0 \le z_i \le 1 \text{ for } i = 1, 2\}$



$$\Pi_{\mathcal{R}}(x) \in \arg\min_{x \in \mathcal{R}} \|x - z\|_2$$

$$\Rightarrow \Pi_B(x) = \min_{\substack{x_i \in B \\ 0 \le z_i \le 1}} \sum_{i=1}^{n} (x_i - z_i)$$

$$z_i = \begin{cases} 0, & x_i < 0 \\ x_i, & 0 \le x_i \le 1 \\ 1, & x_i > 1 \end{cases}$$

$$\Rightarrow \Pi_B(x) = \begin{cases} 0, & x < 0 \\ x, & 0 \le x \le 1 \\ 1, & x > 1 \end{cases}$$

Justification: $(\Pi_B(x) \in B)$

For any x, $0 \le \Pi_B(x) \le 1$. Thus, $\Pi_B(x) \in B$.

Minimality: $(\forall z \in B \quad |x - z| \ge \|\Pi_B(x) - x\|_2)$

Suppose $z \in \mathbb{R}^n$: $0 \le z_i \le 1$ for $i = 1, \dots, n$ $(z \in B)$.

Then RHS: $\|x - x\|_2 = 0$

Suppose $x < 0$. Then LHS is abs val, so $|x - z| \ge 0$ is true.

Suppose $0 \le x \le 1$. Then LHS is $|x - x| = 0 \Rightarrow |0| \ge |0|$ is true.

Suppose $x > 1$. Then LHS is $|x - 1| \ge 0$ which is true since

$(x > 1) - 1 \geq 0$.

Set B is convex and closed, so $\Pi_B(x)$ is unique.

(b) Let $f: \mathbb{R}^n \to \mathbb{R}$ be given by $f(x) = \|Ax\|_2^2 + a^T x$, where $A \in \mathbb{R}^{n \times n}$ is a positive definite matrix, and $a \in \mathbb{R}^n$. Write a projected GD algorithm to solve $\min_{x \in \Omega} f(x)$ for $\Omega = B$, with B from part (a). You do not need to specify step size.

$$x^{(t+1)} = \arg\min_{x \in B} \|y^{(t+1)} - x\|_2 \quad , y^{(t+1)} = x^{(t)} - \mu^{(t)} \nabla f(x^{(t)})$$

$$\Leftrightarrow \quad x^{(t+1)} = \Pi_B\left(x^{(t)} - \mu^{(t)} \nabla f(x^{(t)})\right)$$

$$f(x) = \|Ax\|_2^2 + a^T x = \sum_{i=1}^{n} (Ax)_i^2 + a^T x$$

$$\Rightarrow \nabla f(x) = \nabla\left(\sum_{i=1}^{n} (Ax)_i^2 + a^T x\right) = 2A^T A x + a$$

alg GD

Initialize $x^{(0)} \in B$

(Initialize stepsize - not specified in problem)

For $i = 0, 1, \ldots,$ do

(update stepsize - not specified in this problem)

grad_it $= 2A^T A x^{(i)} + a$

$y^{(i+1)} = x^{(i)} - (\text{stepsize}) * \text{grad\_it}$

If $y^{(i+1)} < 0$:

$x^{(i+1)} = 0$

else if $y^{(i+1)} > 1$:

$x^{(i+1)} = 1$

else: $x^{(i+1)} = y^{(i+1)}$

} $\left(x^{(i+1)} = \Pi_B(y^{(i+1)})\right)$

(c) Repeat part (b) bu for $\Omega = B_2^n = \{z \in \mathbb{R}^n : \|z\|_2 \leq 1\}$

By a previous example, $\Pi_{B_2^n}(x) = \begin{cases} x, & \|x\| \leq 1 \\ \frac{x}{\|x\|}, & \|x\| > 1 \end{cases}$

alg GD

Initialize $x^{(0)} \in B$

(Initialize stepsize - not specified in problem)

For $i = 0, 1, \ldots,$ do

(update stepsize - not specified in this problem)

$grad\_it = 2A^TAx^{(i)} + a$

$y^{(i+1)} = x^{(i)} - (stepsize) * grad\_it$

$mag\_y = \|y^{(i+1)}\|_2$ // "2 norm of $y^{(i+1)}$"

If $mag\_y \leq 1$:

$x^{(i+1)} = y^{(i+1)}$

else: $x^{(i+1)} = y^{(i+1)} / mag\_y$

} $(x^{(i+1)} = \Pi_{B_2^n}(y^{(i+1)}))$

2) Consider the hollow sphere S in $\mathbb{R}^n$, i.e. the set $S := \{x \in \mathbb{R}^n : \|x\|_2^2 = 1\}$. Consider the function $f: \mathbb{R}^n \to \mathbb{R}$ given by $f(x) = x^T Q x$, where Q is $n \times n$ symmetric matrix. For this problem you may use the fact that $\nabla f(x) = 2Qx$

(a) For an arbitrary point $y \in \mathbb{R}^n$, $\Pi(y)$ be the projection of y onto S. Find an expression for $\Pi(y)$ and give a short argument (i.e. proof) for why this is the correct expression. Make sure to handle the case $y = 0$ (i.e. the zero vector). Hint: Recall that the projection minimizes $\|x-y\|$ for $y \in S$. One approach would be to consider the reverse triangle inequality $\|x-y\| \geq \|x\| - \|y\|$ and find a projection formula that achieves the global lower bound (i.e. equality instead of inequality). This would prove you've found the projection.

$$\Pi_S(y) \in \arg\min_{x \in S} \|x-y\|_2$$

By reverse triangle inequality, $\|x-y\|_2 \geq |\|x\| - \|y\||$. We know that for any $x \in S$, $\|x\|_2^2 = 1 \Rightarrow \|x\|_2 = 1$

$$\Rightarrow \|x-y\|_2 \geq |1 - \|y\|_2|$$

We want $\|x-y\|_2 = |1 - \|y\|_2|$ for any $x \in S$.

Starting from our goal $|1 - \|y\|_2|$, we have

$$|1 - \|y\|_2| = \left|\frac{\|y\|_2}{\|y\|_2} - \|y\|_2\right|$$

$$= \|y\|_2 \left|\left(\frac{1}{\|y\|_2} - 1\right)\right|$$

$$= \left\|\frac{y}{\|y\|_2} - y\right\|_2 \quad \Rightarrow \quad x = \frac{y}{\|y\|_2}$$

Projection: $\left\| \frac{y}{\|y\|_2} - y \right\|_2 = \left\| y \left(\frac{1}{\|y\|_2} - 1 \right) \right\|_2$

$= \|y\|_2 \left| \frac{1}{\|y\|_2} - 1 \right|$

$= \left| 1 - \|y\|_2 \right|$

Feasability: $(\Pi_S(y) \in S)$

$\|x\|_2^2 = \left\| \frac{y}{\|y\|_2} \right\|_2^2 = 1$ ✓

When $y = 0$, $\|x - 0\|_2 \geq 1 \Rightarrow \|x\|_2 \geq 1$, which holds true for any $x \in S$. Hence in this case any $x \in S$ is a minimizer.

(b) Is S a convex set?

S is the set of vectors in $\mathbb{R}^n$ whose magnitude (length) squared equals one. Let $v = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \in S$ and $w = \begin{bmatrix} 0 \\ -1 \end{bmatrix} \in S$ (both have magnitude 1). Then $\alpha \begin{bmatrix} 1 \\ 0 \end{bmatrix} + (1-\alpha) \begin{bmatrix} 0 \\ -1 \end{bmatrix}$

$= \begin{bmatrix} \alpha \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \alpha - 1 \end{bmatrix} = \begin{bmatrix} \alpha \\ \alpha - 1 \end{bmatrix}$

Suppose $\alpha = \frac{1}{2} \in [0, 1]$. Then $\alpha v + (1-\alpha) w = \begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix}$

$\left\| \begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix} \right\|_2 = \sqrt{1/2} \neq 1$. So $\alpha v + (1-\alpha) w \notin S$. Hence S is not convex.

(c) Write a projected GD algorithm, with constant step size μ, for $\min_{x \in \mathbb{R}^n} x^T Q x$ subject to $\|x\|_2^2 = 1$.

We have $\nabla f(x) = 2Qx$,

$$x^{(t+1)} = \arg\min_{x \in S} \|y^{(t+1)} - x\|_2 \quad , y^{(t+1)} = x^{(t)} - \mu^{(t)} \nabla f(x^{(t)})$$

$$\Leftrightarrow \quad x^{(t+1)} = \Pi_S\left(x^{(t)} - \mu^{(t)} \nabla f(x^{(t)})\right) \quad \text{where}$$

$$\Pi_S(y) = \begin{cases} \frac{y}{\|y\|_2} & , y \neq 0 \\ \text{any } x \in S & , y = 0 \end{cases}$$

alg GD

Initialize $x^{(0)} \in S$

Initalize stepsize U

For $i = 0, 1, \ldots$ do

```
grad = 2 * Q * x^(i)
y^(i+1) = x^(i) - U * grad
mag-y = ||y^(i+1)||_2

If y^(i+1) = 0:
    x^(i+1) = any x ∈ R^n with ||x||_2 = 1
else:
    x^(i+1) = y^(i+1) / mag-y
```

(d) Is the projected GD algorithm graventeed to converge to the solution for small enough μ? If not, can you give an example of Q and an initialization $x^{(0)}$ where the algorithm won't converge? Hint: consider a diagonal matrix Q where not all entries are equal.

Since S is not convex, the projected GD algorithm is not graventeed to converge. Lets use

$$Q = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}, \quad x^{(0)} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$\Rightarrow y^{(1)} = x^{(0)} - 2\mu Q x^{(0)}$$

$$= \begin{bmatrix} 0 \\ 1 \end{bmatrix} - 2\mu \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 \\ 1 \end{bmatrix} - \begin{bmatrix} 2\mu & 0 \\ 0 & 4\mu \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 \\ 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 4\mu \end{bmatrix} = \begin{bmatrix} 0 \\ 1 - 4\mu \end{bmatrix}$$

$$\| y^{(1)} \|_2 = \sqrt{(1-4\mu)^2} = |1 - 4\mu|$$

If $\mu \neq 1/4$, $y^{(1)} \neq 0$, so

$$x^{(1)} = y^{(1)} / \| y^{(1)} \|_2 = \begin{bmatrix} 0 \\ 1-4\mu \end{bmatrix} \Big/ |1-4\mu| = \begin{bmatrix} 0 \\ \text{sign}(1-4\mu) \end{bmatrix}$$

Suppose $0 < \mu < 1/4$. Then $1 - 4\mu > 0$ and thus

$$x^{(1)} = \begin{bmatrix} 0 \\ +1 \end{bmatrix} = x^{(0)}$$

So one iteration, even with $0 < \mu < 1/4$, does not move $x^{(i)}$ at all $\Rightarrow$ does not converge

Suppose $\mu > 1/4$. Then $1 - 4\mu < 0$, so $x^{(1)} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$

$$\Rightarrow y^{(2)} = \begin{bmatrix} 0 \\ -1 \end{bmatrix} - \begin{bmatrix} 2\mu & 0 \\ 0 & 4\mu \end{bmatrix} \begin{bmatrix} 0 \\ -1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 \\ -1 \end{bmatrix} - \begin{bmatrix} 0 \\ -4\mu \end{bmatrix} = \begin{bmatrix} 0 \\ -(1-4\mu) \end{bmatrix}$$

$$\Rightarrow x^{(2)} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = x^{(0)} \Rightarrow \text{does not converge}$$

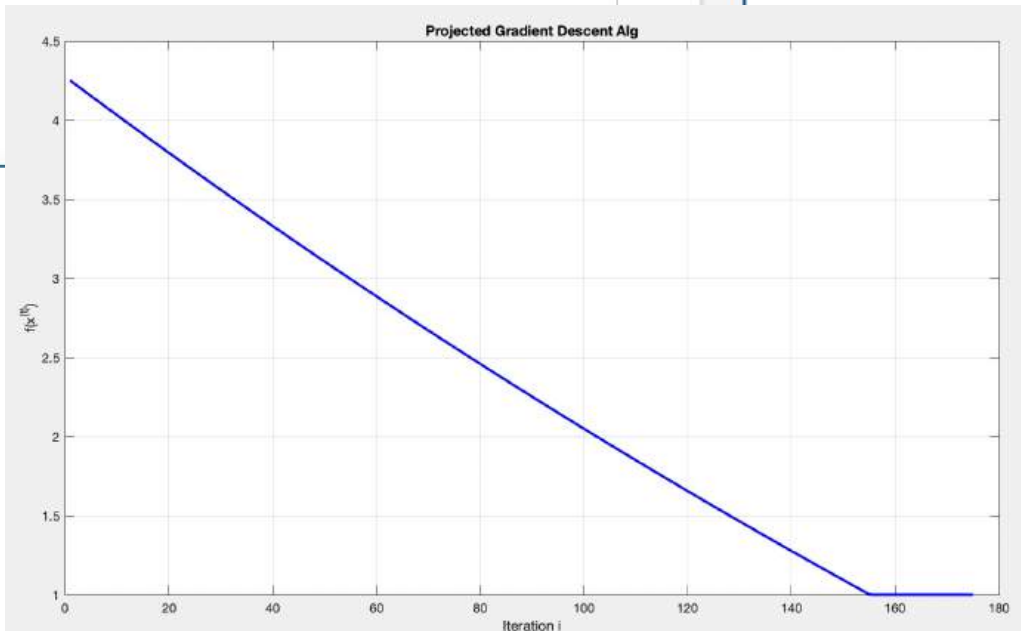
3) Coding Question: Consider the function $f: \mathbb{R}^2 \to \mathbb{R}$ with $f(x_1, x_2) = (x_1 - 2)^2 + (x_2 - 2)^2 - x_1 x_2$ and the following constrained optimization problem: $\min_{x_1, x_2} f(x_1, x_2)$ subject to $0 \leq x_i < 1$, $i = 1, 2$.

Program a projected GD alg, with constant step size $\mu = 0.001$ starting at $(0.5, 0.5)$ for 175 iterations, for the above. Submit code and plot function value $f(x^{(t)})$ against iteration t.

Editor - /Users/sabella/Downloads/projected_GD.m

projected_GD.m

```
function projected_GD()
    x = [0.5; 0.5];
    u = 0.001;
    max_it = 175;
    f_values = zeros(max_it, 1);

    for i = 1:max_it
        f_values(i) = f(x);
        g = grad(x);
        x = x - u * g;
        x = proj_f(x);
    end

    figure;
    plot(1:max_it, f_values, 'b-', 'LineWidth', 2);
    xlabel('Iteration i');
    ylabel('f(x^{(t)})');
    title('Projected Gradient Descent Alg');
    grid on;

end

function val = f(x)
    val = (x(1) - 2)^2 + (x(2) - 2)^2 - x(1)*x(2);
end

function g = grad(x)
    g = [2*x(1) - x(2) - 4;
         2*x(2) - x(1) - 4];
end

function x_proj = proj_f(x)
    x_proj = max(0, min(1, x));
end
```

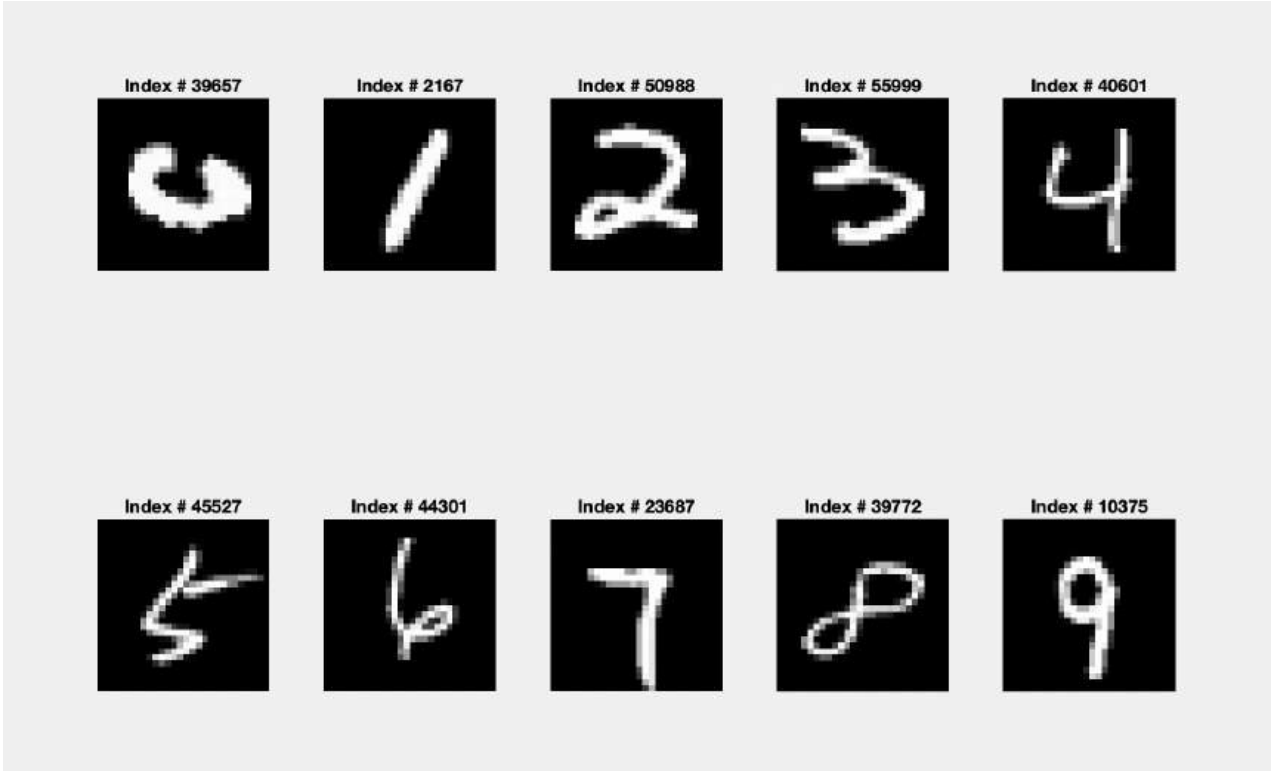


4) (a)

Editor - /Users/sabella/Downloads/display.m

projected_GD.m | display.m | mnist_train.csv

```
function display()
    data = readtable('mnist_train.csv');
    labels = table2array(data(:,1));
    pixels = table2array(data(:,2:end));

    figure('Position', [100, 100, 1200, 600]);
    for digit = 0:9
        indices = find(labels == digit);
        if isempty(indices), continue; end
        random_idx = indices(randi(length(indices)));

        img = reshape(pixels(random_idx, :), 28, 28)';
        subplot(2, 5, digit+1);
        imshow(img, []);
        title(sprintf('Index # %d', random_idx));
    end
end
```



(b)

Editor - /Users/sabella/Downloads/form.m

projected_GD.m | display.m | mnist_train.csv | form.m | untitled6

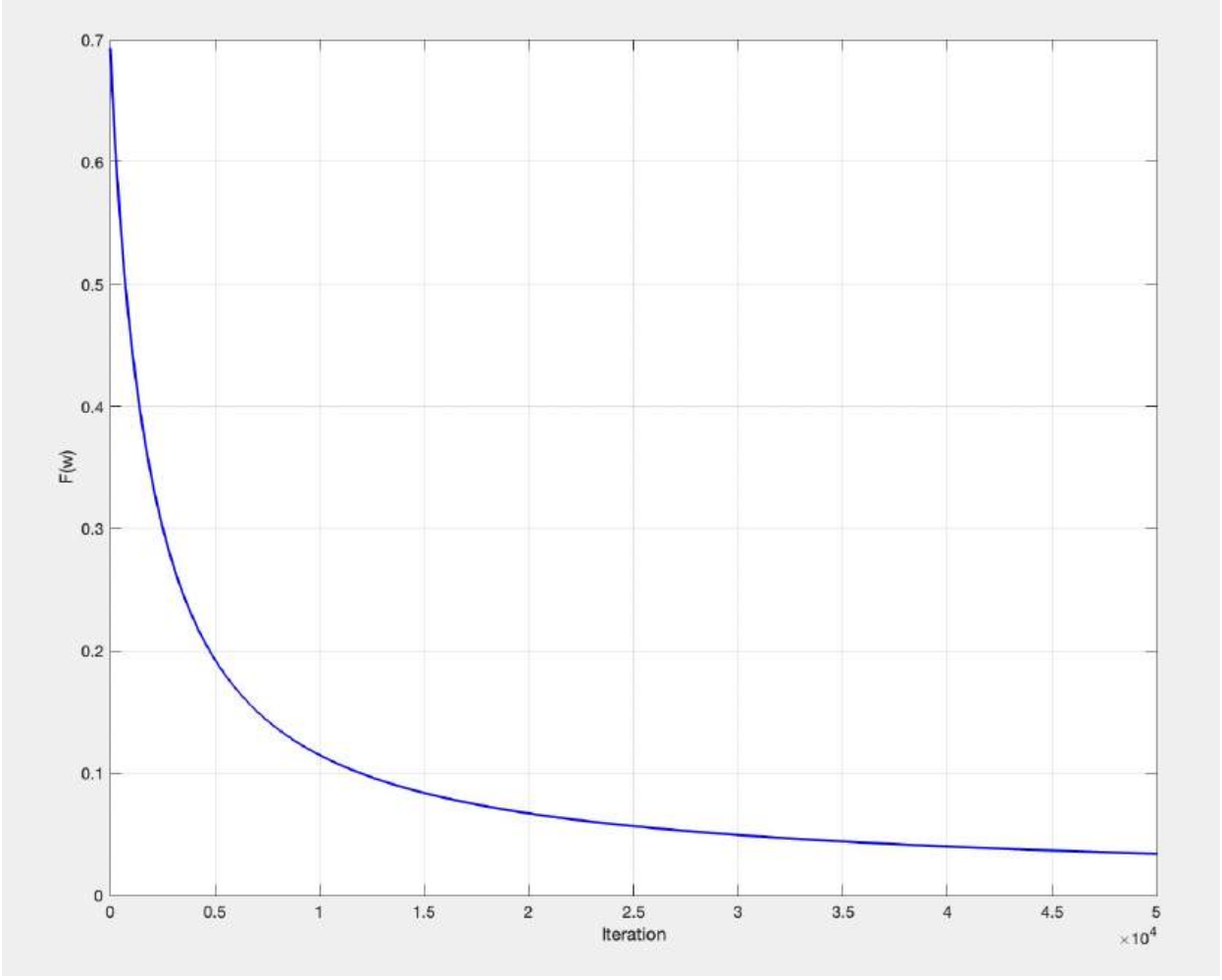
```
function [x, y] = form()
    data = readtable('mnist_train.csv');
    labels = table2array(data(:,1));
    pixels = table2array(data(:,2:end)) / 255.0;

    zeros_idx = find(labels == 0, 500);
    ones_idx = find(labels == 1, 500);
    x = [pixels(zeros_idx,:); pixels(ones_idx,:)];
    y = [-ones(500,1); ones(500,1)];
end
```

(c)

Editor - /Users/sabella/Downloads/GradDesc.m

projected_GD.m | display.m | mnist_train.csv | form.m | GradDesc.m

```
function [w, F_vals] = GradDesc()
    [x, y] = form();
    [~,d] = size(x);
    u = 1e-4;
    max_its = 50000;
    F_vals = zeros(max_its, 1);

    w = zeros(d, 1);

    for i = 1:max_its
        inner_prod = x * w;
        F_vals(i) = mean(log(1 + exp(-y .* inner_prod)));
        n = length(y);
        denom = 1 + exp(y .* inner_prod);
        grad = -(x' * (y ./ denom)) / n;

        w = w - u * grad;
    end

    figure;
    plot(1:max_its, F_vals, 'b-', 'LineWidth', 1.5);
    xlabel('Iteration');
    ylabel('F(w)');
    grid on;
end
```



(d) The shape of F(w) does appear to show convergence to some minimum. At no point does the curve appear to increase at any iteration, and the curve descends steeply before rounding

but to a somewhat clear value. We found in HW2 that the function F is convex, and thus by a previous theorem any local minimizer is also a global minimizer. We also know that given a viable step size, GD converges to a global minimum, which explains the behavior of my GD alg. This thing does mention step size; I chose 50000 iterations since the step size of $\mu = 10^{-7}$ seemed to not quite be enough to fully visualize the convergence. If we could show that F is Lipschitz (too much work for me right now) then a previous theorem would give us $\mu \leq 1/2$, which may lead to a quicker convergence and less iterations of GD. I believe I chose a good stopping criteria, since I knew 100000 would be a lot for my computer, but also 10000 did not seem to be enough. I chose an iteration count somewhere in the middle, and it visualizes the GD nicely.

(c)

Editor - /Users/sabella/Downloads/GradDesc.m

dish.m | mnist_train.csv | form.m | GradDesc.m | mnist_test.csv

```
23        grid on;
24
25        %added to end of GD function from part (c)
26        classify = sign(x * w);
27        train_err = mean(classify ~= y);
28
29        data_test = readtable('mnist_test.csv');  %"form" but with test data
30        labels_test = table2array(data_test(:,1));
31        pixels_test = table2array(data_test(:,2:end)) / 255.0;
32        zeros_test = find(labels_test == 0, 500);
33        ones_test = find(labels_test == 1, 500);
34        x_test = [pixels_test(zeros_test,:); pixels_test(ones_test,:)];
35        y_test = [-ones(500,1); ones(500,1)];
36
37        classify_test = sign(x_test * w);
38        test_err = mean(classify_test ~= y_test);
39
40        fprintf('Training Error: %.4f, Test  Error: %.4f\n', train_err, test_err);
41    end
42
43
```

Command Window

```
     0

>> GradDesc
Training Error: 0.0030, Test  Error: 0.0000
```

Classification error rates shown above. As you can see, the test error is 0, which is unusual and lower than the training error. I do find this relationship unusual, since I'd expect the test error to not fit the model as well as the training. However

although unusual, this implies that the weight w0 applies to new data very, very well producing very no errors for this sample size. I think its possible that yes, the relationship is a bit unusual, but its due to the fact that this model is very accurate and trained well.

Homework 3 for Math 173A - Fall 2025

1. Determine whether each function is Lipschitz, and if so find the smallest possible Lipschitz constant for the function. For all problems, $\|\cdot\|$ represents the Euclidean norm (2-norm).
   (a) $f : \mathbb{R}^n \to \mathbb{R}$ for $f(x) = \|x\|$
   (b) $f : \mathbb{R}^n \to \mathbb{R}$ for $f(x) = \|x\|^2$
   (c) $\rho : \mathbb{R} \to \mathbb{R}$ for $\rho(x) = \frac{1}{1+e^{-x}}$.
   (d) $f : \mathbb{R}^n \to \mathbb{R}$ for $f(x) = \rho(w^T x + b)$ for some weight vector $w \in \mathbb{R}^n$, $b \in \mathbb{R}$, and ρ from part (c).

2. Let f be a convex and differentiable. Let x^* be the global minimum and suppose $x^{(0)}$ is the initialization such that $\|x^* - x^{(0)}\| \leq 5$.
   (a) Let f be $L-$Lipschitz function where $L = 3$. Determine the step size μ and number of steps needed to satisfy
   $$\left\| f\left(\frac{1}{t}\sum_{s=0}^{s-1} x^{(s)}\right) - f\left(x^*\right)\right\| \leq 10^{-4}.$$
   (b) Let f be $L-$smooth where $L = 3$. Determine the step size μ and number of steps needed to satisfy
   $$\left\| f\left(x^{(t)}\right) - f\left(x^*\right)\right\| \leq 10^{-4}.$$

3. Consider the function $f(x_1, x_2) = (2x_1 - 1)^4 + (x_1 + x_2 - 1)^2$.
   (a) Find the global minimum of f, and justify your answer.
   (b) Starting at $x^{(0)} = (0, 0)$, perform gradient descent with backtracking line-search.
      i. Starting at $x^{(0)} = (0, 0)$ with stepsize, which is also called learning rate in the machine learning community, $\mu^{(0)}$, write down the gradient descent equation for $x^{(1)}$.
      ii. Suppose we want to set $\mu^{(0)}$ using backtracking line search with $\gamma = 0.2$ and Armijo's condition $f(x^{(1)}) \leq f(x^{(0)}) - \mu^{(0)}\gamma\|\nabla f(x^{(0)})\|_2^2$. Find a value of $\mu^{(0)}$ that satisfies this.
      iii. Suppose instead you started with $\mu^{(0)} = 1$ and an update of $\mu^{(0)} \leftarrow \frac{1}{2}\mu^{(0)}$ (i.e. $\beta = \frac{1}{2}$). In the worst case, how many steps of back-tracking would you have to take before accepting $x^{(1)}$?

HW 3 SABRINA SAVIND

1) Determine whether each function is lipschitz, if so find the smallest possible lipschitz constant for the function. For all problems, $\|\cdot\|$ represents the Euclidean norm (2-norm).

Lipschitz Condition: $f: \mathbb{R}^n \to \mathbb{R}$, $\forall x, y \in \mathbb{R}^n$, $L > 0$:

$$|f(x) - f(y)| \leq L \cdot \|x - y\|_2$$

If f is C^1, f is lip iff $\|\nabla f(x)\|_2 \leq L$

(a) $f: \mathbb{R}^n \to \mathbb{R}$ for $f(x) = \|x\|$

$$|f(x) - f(y)| = |\|x\|_2 - \|y\|_2| \leq \|x - y\|_2 \quad \forall x, y \in \mathbb{R}^n$$

by Reverse Triangle Inequality

$\Rightarrow$ Yes lipschitz, $L = 1$

WA $y = 0$, $x \neq 0$: $\|f(x) - f(0)\| = \|\|x\|_2 - \|0\|_2\| = \|x\|_2$

$\Rightarrow \|x\|_2 \leq L \cdot \|x - 0\|_2 \Rightarrow \|x\|_2 \leq L \cdot \|x\|_2 \Rightarrow L \geq 1$

$\Rightarrow L = 1$ smallest possible

(b) $f: \mathbb{R}^n \to \mathbb{R}$ for $f(x) = \|x\|^2$

$$|f(x) - f(y)| = |\|x\|_2^2 - \|y\|_2^2|$$

$$\|x\|_2^2 - \|y\|_2^2 = (x^T x) - (y^T y)$$

$$= (x + y)(x - y)$$

$$|(x+y)(x-y)| \leq \|x+y\| \, \|x-y\|$$

$$\Rightarrow |\|x\|_2^2 - \|y\|_2^2| \leq (\|x\|_2 + \|y\|_2) \|x - y\|_2$$

since $x, y \in \mathbb{R}^n$, $L = \|x\|_2 + \|y\|_2$ varies over $\mathbb{R}^n$

$\Rightarrow$ f is NOT lipschitz

(c) $\rho : \mathbb{R} \to \mathbb{R}$ for $\rho(x) = \frac{1}{1+e^{-x}} = (1+e^{-x})^{-1}$

$$\nabla\rho(x) = -(1+e^{-x})^{-2}(-e^{-x}) = \frac{e^{-x}}{(1+e^{-x})^2}$$

$$= \frac{e^{-x}}{(1+e^{-x})(1+e^{-x})}$$

$$= \frac{1}{(1+e^{-x})}\left(1 - \left(\frac{1}{1+e^{-x}}\right)\right)$$

$$= \rho(x)(1-\rho(x))$$

For $\rho \in (0,1)$: $f(\rho) = \rho(1-\rho) = \rho - \rho^2$

$\nabla f(\rho) = 1 - 2\rho = 0 \Rightarrow \rho = \frac{1}{2}$

$f(\frac{1}{2}) = \frac{1}{2}(1-\frac{1}{2}) = \frac{1}{4}$, $f(0) = 0$, $f(1) = 0$

$\Rightarrow \max \nabla\rho(x) = \frac{1}{4} \Rightarrow L = \frac{1}{4}$ yes lipschitz, smallest

(d) $f : \mathbb{R}^n \to \mathbb{R}$ for $f(x) = \rho(w^T x + b)$ for some weight vector $w \in \mathbb{R}^n$, $b \in \mathbb{R}$, and ρ from part (c).

$$|f(x) - f(y)| = |\rho(g(x)) - \rho(g(y))| \leq \frac{1}{4}\|\rho(g(x)) - \rho(g(y))\|_2$$

$$|g(x) - g(y)| = |(w^T x - b) - (w^T y - b)| = |w^T(x-y)|$$

by cauchy schwartz $|w^T(x-y)| \leq \|w\|\|x-y\|$

$\Rightarrow L = \|w\|$, $w \in \mathbb{R}^n$

$$|f(x) - f(y)| \leq \frac{1}{4}\|w\|\|x-y\|_2$$

$\Rightarrow L = \frac{1}{4}\|w\|$ yes lipschitz, smallest

2. Let f be convex and differentiable. Let x^* be the global minimum and suppose $x^{(0)}$ is the initialization such that $\|x^* - x^{(0)}\| \le 5$.

(a) Let f be L-Lipschitz function where $L=3$. Determine the step size μ and number of steps needed to satisfy

$$\left\| f\left(\frac{1}{t}\sum_{s=0}^{t-1} x^{(s)}\right) - f(x^*) \right\| \le 10^{-4}$$

$R=5, L=3$, $\|\nabla f\| \le 3$, $f(y^{(t)}) - f(x^*) \le \frac{RL}{\sqrt{t}}$

Choose $\mu = \frac{R}{L\sqrt{t}} = \frac{5}{3\sqrt{t}}$

$f(y^{(t)}) - f(x^*) \le \frac{15}{\sqrt{t}}$ } say $\le 10^{-4}$

$\frac{15}{\sqrt{t}} \le 10^{-4} \Rightarrow \frac{15}{10^{-4}} \le \sqrt{t}$ let $\sqrt{t} = 150000$

$\Rightarrow t = 2.25 \times 10^{10}$

$\Rightarrow \mu = \frac{5}{3 \cdot 150000} \approx 1.11 \times 10^{-5}$

(b) Let f be L-smooth where $L=3$. Determine the step size μ and number of steps needed to satisfy

$$\|f(x^{(t)}) - f(x^*)\| \le 10^{-4}.$$

$\mu \in (0, \frac{1}{3}]$ $f(x^{(t)}) - f(x^*) \le \frac{\|x_0 - x^*\|^2}{2\mu t} = 10^{-4}$, $\forall t \ge 1$

$\Rightarrow \|x_0 - x^*\|^2 = 2 \cdot 10^{-4} \mu t \Rightarrow \mu = \frac{\|x_0 - x^*\|^2}{2 \cdot 10^{-4} t}$

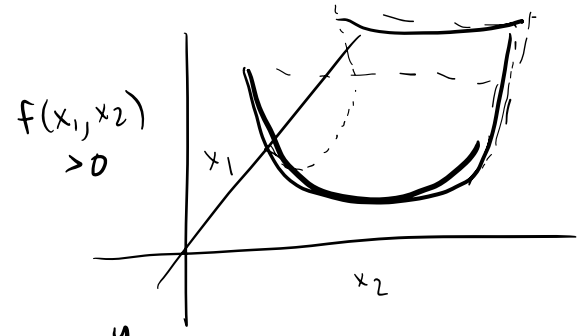
Let $\boxed{\mu = \frac{1}{3}} = \frac{\|x_0 - x^*\|^2}{2 \cdot 10^{-4} t} \Rightarrow t = \frac{3\|x_0 - x^*\|^2}{2 \cdot 10^{-4}}$, $\|x_0 - x^*\|^2 \le 25$

$\Rightarrow t \ge \frac{75}{2 \cdot 10^{-4}} \Rightarrow t \ge 375000$

$\mu = \frac{1}{3}$, $t = 375000$

3. Consider the function $f(x_1, x_2) = (2x_1 - 1)^4 + (x_1 + x_2 - 1)^2$

(a) Find the global min of f, justify your answer



$f(x_1, x_2) \geq 0 \quad \forall x_1, x_2$

$\Rightarrow$ find min by setting $f(x_1, x_2) = 0$

$(2x-1)^4 = 0 \Rightarrow 2x_1 - 1 = 0 \Rightarrow x_1 = \frac{1}{2}$

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

$\Rightarrow x^* = (\frac{1}{2}, \frac{1}{2})$

$\nabla (2x-1)^4 = 8(2x_1 - 1)^3 \quad \nabla^2 (2x-1)^4 = 48(2x_1 - 1)^2 > 0 \Rightarrow$ convex

$\nabla (x_1 + x_2 - 1)^2 = \begin{bmatrix} 2(x_1 + x_2 - 1) \\ 2(x_1 + x_2 - 1) \end{bmatrix} \quad \nabla^2 (x_1 + x_2 - 1) = \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix} = PD \Rightarrow$ convex

(Prev. Theorem: x^* is global min iff $\nabla f(x^*) = 0$)

$$\nabla f = \begin{pmatrix} 8(2x-1)^3 + 2(x_1 + x_2 - 1) \\ 2(x_1 + x_2 - 1) \end{pmatrix}$$

$$\nabla f(\tfrac{1}{2}, \tfrac{1}{2}) = \begin{pmatrix} 8(0)^3 + 2(\frac{1}{2} + \frac{1}{2} - 1) \\ 2(\frac{1}{2} + \frac{1}{2} - 1) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \checkmark$$

$\Rightarrow x^* = (\frac{1}{2}, \frac{1}{2})$ is a global min since $f(x_1, x_2)$ is the sum of two non-negative convex functions and $\nabla f(\frac{1}{2}, \frac{1}{2}) = (0, 0)$

(b) Starting at $x^{(0)} = (0, 0)$, perform gradient descent with backtracking line-search.

i. Starting at $x^{(0)} = (0,0)$ with stepsize, aka learning rate in ML community, $\mu^{(0)}$, write down the gradient descent equation for $x^{(1)}$.

$$\nabla f(x_1, x_2) = \begin{bmatrix} 8(2x_1 - 1)^3 + 2(x_1 + x_2 - 1) \\ 2(x_1 + x_2 - 1) \end{bmatrix}$$

$$\nabla f(0,0) = \begin{bmatrix} 8(-1)^3 + 2(-1) \\ 2(-1) \end{bmatrix} = \begin{bmatrix} -10 \\ -2 \end{bmatrix}$$

$$x^{(1)} = x^{(0)} - \mu^{(0)} \nabla f(x^{(0)})$$

$$= (0,0) - \mu^{(0)} \cdot \begin{bmatrix} -10 \\ -2 \end{bmatrix} = \mu^{(0)} \begin{bmatrix} 10 \\ 2 \end{bmatrix} = \begin{bmatrix} 10\mu^{(0)} \\ 2\mu^{(0)} \end{bmatrix}$$

ii. Suppose we want to set $\mu^{(0)}$ using backtracking line search with $\gamma = 0.2$ and Armijo's condition

$$f(x^{(1)}) \leq f(x^{(0)}) - \mu^{(0)} \gamma \| \nabla f(x^{(0)}) \|_2^2 .$$

Find a value of $\mu^{(0)}$ that satisfies this.

$$f(x^{(0)}) = f(0,0) = (2(0)-1)^4 + (0+0-1)^2 = 1 + 1 = 2$$

$$\| \nabla f(x^{(0)}) \|_2^2 = \left\| \begin{bmatrix} -10 \\ -2 \end{bmatrix} \right\|_2^2 = 100 + 4 = 104$$

$$\Rightarrow f(x^{(1)}) \leq 2 - \mu^{(0)} \cdot 0.2 \cdot 104$$

$$\leq 2 - 20.8\mu^{(0)}$$

$$\Rightarrow f\left(\mu^{(0)} \begin{bmatrix} 10 \\ 2 \end{bmatrix}\right) \leq 2 - 20.8\mu^{(0)}$$

$$f\left(\mu^{(0)} \begin{bmatrix} 10 \\ 2 \end{bmatrix}\right) = f\left(\begin{bmatrix} 10\mu^{(0)} \\ 2\mu^{(0)} \end{bmatrix}\right) = (2(10\mu^{(0)}) - 1)^4 + (10\mu^{(0)} + 2\mu^{(0)} - 1)^2$$

$$= (20\mu^{(0)} - 1)^4 + (12\mu^{(0)} - 1)^2$$

$$\Rightarrow (20\mu^{(0)} - 1)^4 + (12\mu^{(0)} - 1)^2 \leq 2 - 20.8\mu^{(0)}$$

Test $\mu^{(0)} = 1$: $(20-1)^4 + (12-1)^2 \leq 2 - 20.8$

$\Rightarrow (19)^4 + 11^2 \leq 2 - 20.8 \quad \Rightarrow \not\leq$

Test $\mu^{(0)} = 0.01$:

$(20 \cdot 0.01 - 1)^4 + (12(0.01) - 1)^2 \leq 2 - 20.8(0.01)$

$\Rightarrow (0.2 - 1)^4 + (0.12 - 1)^2 \leq 2 - 0.208$

$\Rightarrow (-.8)^4 + (-.88)^2 \leq 1.792$

$\Rightarrow 0.4096 + 0.7744 \leq 1.792$

$\Rightarrow 1.184 \leq 1.792$ ✓ $\mu^{(0)} = 0.01$ satisfies

iii. Suppose instead you started with $\mu^{(0)} = 1$ and an update of $\mu^{(0)} \leftarrow \frac{1}{2}\mu^{(0)}$ (i.e. $\beta = \frac{1}{2}$). In the worst case, how many steps of back-tracking would you have to take before accepting $x^{(1)}$?

$\mu^{(0)}$ cuts in half each iteration, i.e.

$\mu^{(0)} = \frac{1}{2}, \mu^{(1)} = \frac{1}{4}, \mu^{(2)} = \frac{1}{8}, \ldots$

$\Rightarrow \mu^{(t)} = \frac{1}{2^t}$

Have $(20\mu^{(0)} - 1)^4 + (12\mu^{(0)} - 1)^2 \leq 2 - 20.8\mu^{(0)}$ satisfied

by $\mu^{(0)} = 0.01 \quad \Rightarrow \frac{1}{2^t} \leq 0.01$

$\Rightarrow 2^t \geq 100$

$\Rightarrow t \geq 7$

$2^7 = 128 \Rightarrow \mu^{(t)} = \frac{1}{128} = 0.0078125$

$\Rightarrow t = 7$

Homework 2 for Math 173A - Fall 2025

1. Using the conditions of optimality, find the local maximizers and local minimizers, of the following functions and determine whether they are local maximizers or local minimimizers. You may use a computer to find the eigenvalues, but these questions should have easily accessible eigenvalues by hand.

   (a) $f : \mathbb{R}^2 \to \mathbb{R}$ for $f(x_1, x_2) = x_1^4 + 2x_2^4 - 4x_1x_2$

   (b) $f : \mathbb{R}^3 \to \mathbb{R}$ for $f(\vec{x}) = \vec{x}^T A\vec{x} + b^T\vec{x}$, where

   $$A = \begin{bmatrix} -1 & 0 & 1/2 \\ 0 & -1 & 0 \\ 1/2 & 0 & -1 \end{bmatrix}, \qquad b = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}$$

   Bonues question: Are the local maximizers/minimizers you find also global maximizers/minimizers? If so, why?

2. Consider the problem $f : \mathbb{R}^n \to \mathbb{R}$ for $f(x) = \|Ax - b\|_2^2$ for $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$. (Note: this was on the last homework). Write down the gradient descent algorithm to solve the optimization

   $$\min_{x\in\mathbb{R}^n} f(x).$$

   This doesn't have to be a computer program, just something of the form

   $$x^{(0)} = ...$$
   $$x^{(t+1)} = ... \text{ (where the right hand side is in terms of } x^{(t)}).$$

   You need to specify the formula of the gradient, but you do not need to specify the exact choice of the stepsize.

3. **Implementing Classification Model:** First some background for classification:

   - You are given labeled data $\{(x_i, y_i)\}_{i=1}^N$ for $x_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$.
   - Logistic regression involves choosing a label according to

     $$y = \text{sign}(\langle w, x\rangle).$$

     Note we ignore the y-intercept term here, so we only need the optimal $w \in \mathbb{R}^d$.

- It turns out the correct function to minimize to find the weights is

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

Questions:

(a) Is $F(w)$ a convex function?

(b) Write down the gradient descent algorithm for minimizing F. You need to specify the starting point, the formula of the gradient, but you do not need to specify the exact choice of the stepsize.

4. **Coding Question:** Recall that the equation for an ellipse in $\mathbb{R}^2$ is

$$a_1 x^2 + a_2 y^2 = 1.$$

Given data $\{(x_i, y_i)\}_{i=1}^N \subset \mathbb{R}^2$ that lie on (or near) the ellipse, you can find the best fit ellipse by solving

$$\min_{a\in\mathbb{R}^2} f(a)$$

where

$$f(a) = \sum_{i=1}^{N} (a_1 x_i^2 + a_2 y_i^2 - 1)^2$$

(a) Find an $A \in \mathbb{R}^{N\times 2}$ and $b \in \mathbb{R}^N$ such that $f(a) = \|Aa - b\|_2^2$. What is A in terms of (x_i, y_i)?

(b) Download the data provided on the HW page (called HW2_ellipse.csv), and create a scatter plot of the points (submit your code and the plot).

(c) Using Problem 2, create computer code to compute the gradient descent algorithm on this $f(a)$. The code must include a stopping condition. Use a step-size of $\eta = \frac{1}{2\|A^T A\|}$. Note, you cannot use a built-in gradient descent algorithm it must be written with a while or for loop. Also note, the norm of a matrix $\|X\| = \lambda_{\max}(X)$ is the largest eigenvalue (in magnitude) of X, and can be computed using "norm(X,2)" in matlab or "np.linalg.norm(X,2)" in Python. (submit the code)

(d) Using the data provided and your gradient descent code, estimate the solution a. Report a and $f(a)$. Given $f(a)$ and N, do you think you fit the data well or poorly? Given the convexity of f, do you think this is the optimal a?

HW 2 SABELLA SAULINO

1) Using the conditions of optimality, find the local maximizers and minimizers, of the following functions & determine whether they are local maximizers or local minimizers.

(a) $f: \mathbb{R}^2 \to \mathbb{R}$ for $f(x_1, x_2) = x_1^4 + 2x_2^4 - 4x_1x_2$

$\nabla f(x) = 0 \Rightarrow \nabla f(x_1, x_2) = \begin{pmatrix} 4x_1^3 - 4x_2 \\ 8x_2^3 - 4x_1 \end{pmatrix} = 0$

$4x_1^3 - 4x_2 = 0 \Rightarrow 4x_1^3 = 4x_2 \Rightarrow x_1^3 = x_2$

$8x_2 - 4x_1 = 0 \Rightarrow 8x_2^3 - 4x_1$

using $x_1^3 = x_2$: $8(x_1^3)^3 - 4x_1 = 8x_1^9 - 4x_1 = 0$

$= 2x_1^9 - x_1 = 0$

$= x_1(2x_1^8 - 1) = 0$

$\Rightarrow x_1 = 0 \Rightarrow x_2 = 0, \quad 2x_1^8 = 1 \Rightarrow x_1^8 = \frac{1}{2} \Rightarrow x_1 = \pm\left(\frac{1}{2}\right)^{1/8}$

$\Rightarrow x_2 = \pm\left(\frac{1}{2}\right)^{3/8}$

$x_1^* = (0,0)$

$x_2^* = \left(\left(\frac{1}{2}\right)^{1/8}, \left(\frac{1}{2}\right)^{3/8}\right)$

$x_3^* = \left(-\left(\frac{1}{2}\right)^{1/8}, -\left(\frac{1}{2}\right)^{3/8}\right)$

$$\nabla^2 f(x_1, x_2) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1 x_1} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f}{\partial x_2 x_1} & \frac{\partial^2 f}{\partial x_2 x_2} \end{bmatrix} = \begin{bmatrix} 12x_1^2 & -4 \\ -4 & 24x_2^2 \end{bmatrix}$$

$$\nabla^2 f(0,0) = \begin{bmatrix} 0 & -4 \\ -4 & 0 \end{bmatrix} \quad \begin{bmatrix} -\lambda & -4 \\ -4 & -\lambda \end{bmatrix} \Rightarrow \lambda^2 - 16 = 0$$

$\Rightarrow \lambda_1 = -4, \lambda_2 = +4$

$\Rightarrow$ saddle pt

$$\nabla^2 f\left(\left(\tfrac{1}{2}\right)^{1/8}, \left(\tfrac{1}{2}\right)^{3/8}\right) = \begin{bmatrix} 12\left(\tfrac{1}{2}\right)^{1/4} & -4 \\ -4 & 24\left(\tfrac{1}{2}\right)^{3/4} \end{bmatrix} \quad \left(12\left(\tfrac{1}{2}\right)^{1/4} > 0\right) \checkmark$$

$$\det(\nabla^2 f) = \left(12\left(\tfrac{1}{2}\right)^{1/4} \cdot 24\left(\tfrac{1}{2}\right)^{3/4}\right) - 16$$

$$= 288 \cdot \left(\tfrac{1}{2}\right) - 16 = 144 - 16 = 128 > 0$$

$$\nabla^2 f\left(-\left(\tfrac{1}{2}\right)^{1/8}, -\left(\tfrac{1}{2}\right)^{3/8}\right) = \begin{bmatrix} 12\left(\tfrac{1}{2}\right)^{1/4} & -4 \\ -4 & 24\left(\tfrac{1}{2}\right)^{3/4} \end{bmatrix} \text{ same as above}$$

$\nabla^2 f\left(\left(\tfrac{1}{2}\right)^{1/8}, \left(\tfrac{1}{2}\right)^{3/8}\right)$ & $\nabla^2 f\left(-\left(\tfrac{1}{2}\right)^{1/8}, -\left(\tfrac{1}{2}\right)^{3/8}\right)$ both PD,

so $x_2^* = \left(\left(\tfrac{1}{2}\right)^{1/8}, \left(\tfrac{1}{2}\right)^{3/8}\right)$ & $x_3^* = \left(-\left(\tfrac{1}{2}\right)^{1/8}, -\left(\tfrac{1}{2}\right)^{3/8}\right)$

are local minima

(6) $f: \mathbb{R}^3 \to \mathbb{R}$ for $f(\vec{x}) = \vec{x}^T A \vec{x} + b^T \vec{x}$, where

$$A = \begin{bmatrix} -1 & 0 & 1/2 \\ 0 & -1 & 0 \\ 1/2 & 0 & -1 \end{bmatrix}, \quad b = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix} \qquad f(\vec{x}) = \tfrac{1}{2} x^T (2A) \vec{x} + b^T \vec{x}$$

$$\Rightarrow \nabla f(\vec{x}) = 2A\vec{x} + b = 0$$

$$\Rightarrow 2Ax = -b \Rightarrow Ax = -\tfrac{1}{2} b = -\tfrac{1}{2} \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 0 \\ -1/2 \\ -1 \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} -1 & 0 & 1/2 \\ 0 & -1 & 0 \\ 1/2 & 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ -1/2 \\ -1 \end{bmatrix}$$

$$\Rightarrow -x_1 + \tfrac{1}{2} x_3 = 0 \Rightarrow x_1 = \tfrac{1}{2} x_3$$

$$-x_2 = -\tfrac{1}{2} \Rightarrow \underline{x_2 = 1/2}$$

$$\tfrac{1}{2} x_1 - x_3 = -1$$

$$\Rightarrow \frac{1}{2}\left(\frac{1}{2}x_3\right) - x_3 = -1$$

$$\Rightarrow -\frac{3}{4}x_3 = -1 \Rightarrow \underline{x_3 = \frac{4}{3}} \Rightarrow \underline{x_1 = \frac{2}{3}}$$

$$\Rightarrow x^* = \begin{bmatrix} 2/3 \\ 1/2 \\ 4/3 \end{bmatrix}$$

$$\nabla^2 f(\vec{x}) = 2A = \begin{bmatrix} -2 & 0 & 1 \\ 0 & -2 & 0 \\ 1 & 0 & -2 \end{bmatrix}$$

$$\det\left(\begin{bmatrix} -2-\lambda & 0 & 1 \\ 0 & -2-\lambda & 0 \\ 1 & 0 & -2-\lambda \end{bmatrix}\right) = (-2-\lambda)\left((-2-\lambda)^2 - 1\right) = 0$$

$$\Rightarrow -2-\lambda = 0 \Rightarrow \underline{\lambda_1 = -2}$$

$$(-2-\lambda)^2 = 1 \Rightarrow -2-\lambda = \pm 1$$

$$+1: \quad -2-\lambda = 1 \Rightarrow \underline{\lambda_2 = -3}$$

$$-1: \quad -2-\lambda = -1 \Rightarrow \underline{\lambda_3 = -1}$$

$\lambda_1, \lambda_2, \lambda_3 < 0$ so $\nabla^2 f$ neg def $\Rightarrow$

$x^* = \begin{bmatrix} 2/3 \\ 1/2 \\ 4/3 \end{bmatrix}$ is local maximizer

(Prev. Theorem: x^* is global min iff $\nabla f(x^*) = 0$

(a) $x_2^* = \left(\left(\frac{1}{2}\right)^{1/8}, \left(\frac{1}{2}\right)^{3/8}\right)$ & $x_3^* = \left(-\left(\frac{1}{2}\right)^{1/8}, -\left(\frac{1}{2}\right)^{3/8}\right)$

$$\nabla f(x_2^*) = \begin{pmatrix} 4\left(\frac{1}{2}\right)^{3/8} - 4\left(\frac{1}{2}\right)^{3/8} \\ 8\left(\frac{1}{2}\right)^{9/8} \quad -4\left(\frac{1}{2}\right)^{1/8} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$x_2^* = \left(\left(\frac{1}{2}\right)^{1/8}, \left(\frac{1}{2}\right)^{3/8} \right)$ is a global min

$$\nabla f(x_3^*) = \begin{pmatrix} -4\left(\frac{1}{2}\right)^{3/8} + 4\left(\frac{1}{2}\right)^{3/8} \\ -8\left(\frac{1}{2}\right)^{9/8} + 4\left(\frac{1}{2}\right)^{1/8} \end{pmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$x_3^* = \left(-\left(\frac{1}{2}\right)^{1/8}, -\left(\frac{1}{2}\right)^{3/8} \right)$ is a global min

(ignore)

2) Consider the problem $f: \mathbb{R}^n \to \mathbb{R}$ for $f(x) = \|Ax - b\|_2^2$ for $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. Write down the gradient descent algorithm and solve the optimization $\min_{x \in \mathbb{R}^n} f(x)$.

Doesn't have to be comp program, just something of form

$x^{(0)} = \ldots$

$x^{(t+1)} = \ldots$ (terms of $x^{(t)}$)

Specify formula of gradient, do not need to spec exact choice of step size.

From last hw: $\nabla f(x) = 2A^T(Ax - b)$

Gradient Descent (GD)

Choose $x^{(0)} \in \mathbb{R}^n$

For $t = 1, 2, \ldots$, (or until stopping criterion met)

set $x^{(t)} = x^{(t-1)} - \mu^{(t-1)} \nabla f(x^{(t-1)})$

(Based on Piazza @21) $x^{(t)} = x^{(t-1)} - \mu^{(t-1)} \nabla f(x^{(t-1)})$ for $t \geq 0$

Let $x^{(0)} = 0$. (chosen initialization)

$$\Rightarrow x^{(1)} = x^{(0)} - \mu^{(0)} \nabla f(x^{(0)})$$

$$= 0 - \mu^{(0)}(2A^T(A \cdot 0 - b))$$

$$= \underline{0 + \mu^{(0)} \cdot 2A^T b} = \underline{2\mu A^T b}$$

$$x^{(2)} = x^{(1)} - \mu^{(1)} \nabla f(x^{(1)})$$

$$= \mu^{(0)} 2A^T b - \mu^{(1)}(2A^T(A(\mu^{(0)} 2A^T b) - b))$$

$$= \mu^{(0)} 2A^T b - \mu^{(1)}(2A^T(\mu^{(0)} 2AA^T b - b))$$

$$= \cancel{\mu^{(0)} 2A^T b} - \mu^{(1)}(4\mu^{(0)} A^T A A^T b - \cancel{2A^T b})$$

$$= \underline{4\mu A^T b - 4\mu^2 (A^T A) A^T b}$$

$$\vdots$$

$$x^{(t+1)} = x^{(t)} - \mu(2A^T(Ax^{(t)} - b))$$

3) You are given labeled data $\{(x_i, y_i)\}_{i=1}^N$ for $x_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$.

Logistic regression involves choosing a label according to

$$Y = \text{sign}(\langle w, x \rangle).$$

Note we ignore y-intercept term here, only need optimal $w \in \mathbb{R}^d$.

Correct function to minimize to find the weights is

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

(a) Is $F(w)$ a convex function?

$$\nabla f = \frac{\partial}{\partial \langle w, x_i \rangle y_i} \log\left(1 + e^{-\langle w, x_i \rangle y_i}\right)$$

$$= \frac{1}{1 + e^{-\langle w, x_i \rangle y_i}} \left(e^{-\langle w, x_i \rangle y_i}\right) = \frac{-e^{-\langle w, x_i \rangle y_i}}{1 + e^{-\langle w, x_i \rangle y_i}} = \frac{-1}{1 + e^{\langle w, x_i \rangle y_i}}$$

$$\nabla^2 f = \frac{\partial}{\partial \langle w, x_i \rangle y_i} \left(\frac{-1}{1 + e^{\langle w, x_i \rangle y_i}}\right) = \frac{e^{\langle w, x_i \rangle y_i}}{\left(1 + e^{\langle w, x_i \rangle y_i}\right)^2} > 0$$

$\forall w, x_i, y_i \in \mathbb{R}$

since $e^x > 0 \ \forall x \in \mathbb{R}$.

Since $\nabla^2 f > 0$, f is convex. The sum of convex functions is also convex, so $F = \frac{1}{N} \sum_{i=1}^{N} f$ is convex.

(b) Write down gradient descent alg for minimizing F. You must specify starting point, formula of gradient, not step size μ.

Choose $w^{(0)} \in \mathbb{R}^n$

For $t = 1, 2, \ldots$, (or until stopping criterion met)

set $w^{(t+1)} = w^{(t)} - \mu^{(t)} \nabla F(w^{(t)})$

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

(ignore for now)

$$= \frac{1}{1 + e^{-\langle w, x_i \rangle y_i}} \cdot e^{-\langle w, x_i \rangle y_i} \cdot (-x_i y_i)$$

$$= \frac{-x_i y_i}{1+e^{\langle w, x_i\rangle y_i}} \Rightarrow \nabla F(w) = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{-x_i y_i}{1+e^{\langle w, x_i\rangle y_i}}\right)$$

Gradient Descent (GD)

$w^{(0)} = 0$ // Choose $w^{(0)} \in \mathbb{R}^n$

For $t = 1, 2, \cdots,$

$$w^{(t+1)} = w^{(t)} - \mu^{(t)} \frac{1}{N}\sum_{i=1}^{N}\left(\frac{-x_i y_i}{1+e^{\langle w^{(t)}, x_i\rangle y_i}}\right)$$

4) Coding: Recall that the equation for ellipse in $\mathbb{R}^2$ is

$$a_1 x^2 + a_2 y^2 = 1.$$

Given data $\{(x_i, y_i)\}_{i=1}^N \subset \mathbb{R}^2$ that lie on (or near) the ellipse, you can find the best fit ellipse by solving $\min_{a \in \mathbb{R}^2} f(a)$, where

$$f(a) = \sum_{i=1}^{N}(a_1 x_i^2 + a_2 y_i^2 - 1)^2.$$

(a) Find an $A \in \mathbb{R}^{N\times 2}$ and $b \in \mathbb{R}^N$ such that $f(a) = \|Aa - b\|_2^2$. What is A in terms of (x_i, y_i)?

$$f(a) = \sum_{i=1}^{N}(a_1 x_i^2 + a_2 y_i^2 - 1)^2 = \|a_1 x_i^2 + a_2 y_i^2 - 1\|_2^2$$

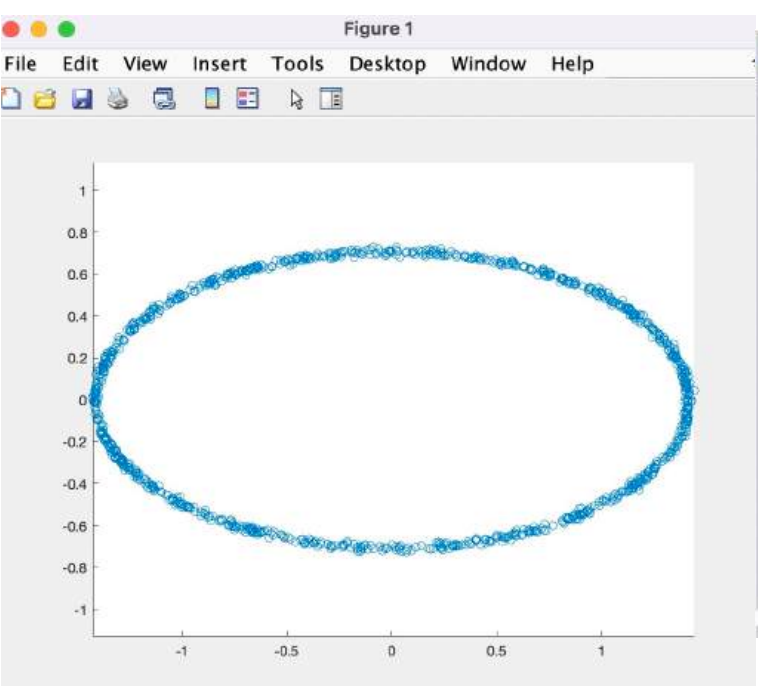
$$= \left\| \begin{bmatrix} x_i^2 & y_i^2 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} - 1 \right\|_2^2$$

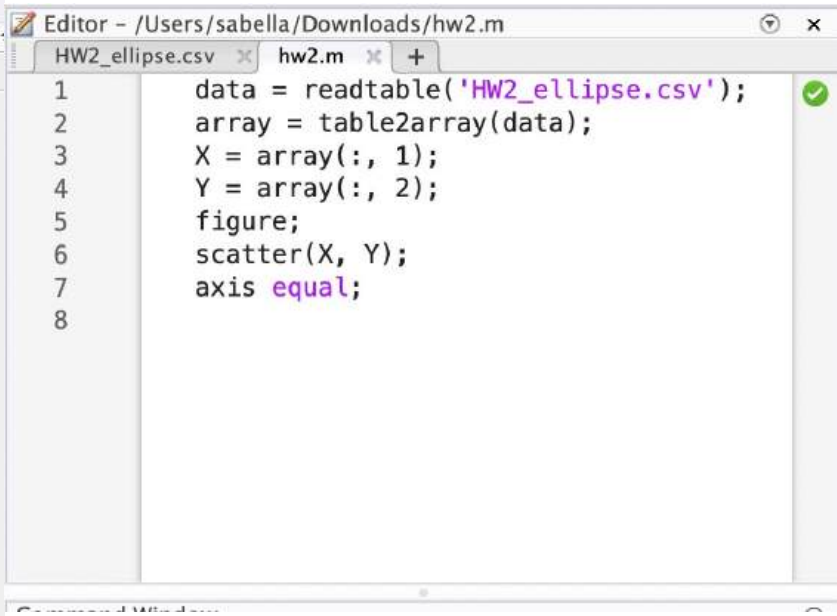
for $i = 1, \cdots, N$:

$$\begin{bmatrix} x_1^2 & y_1^2 \\ x_2^2 & y_2^2 \\ \vdots & \\ x_N^2 & y_N^2 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

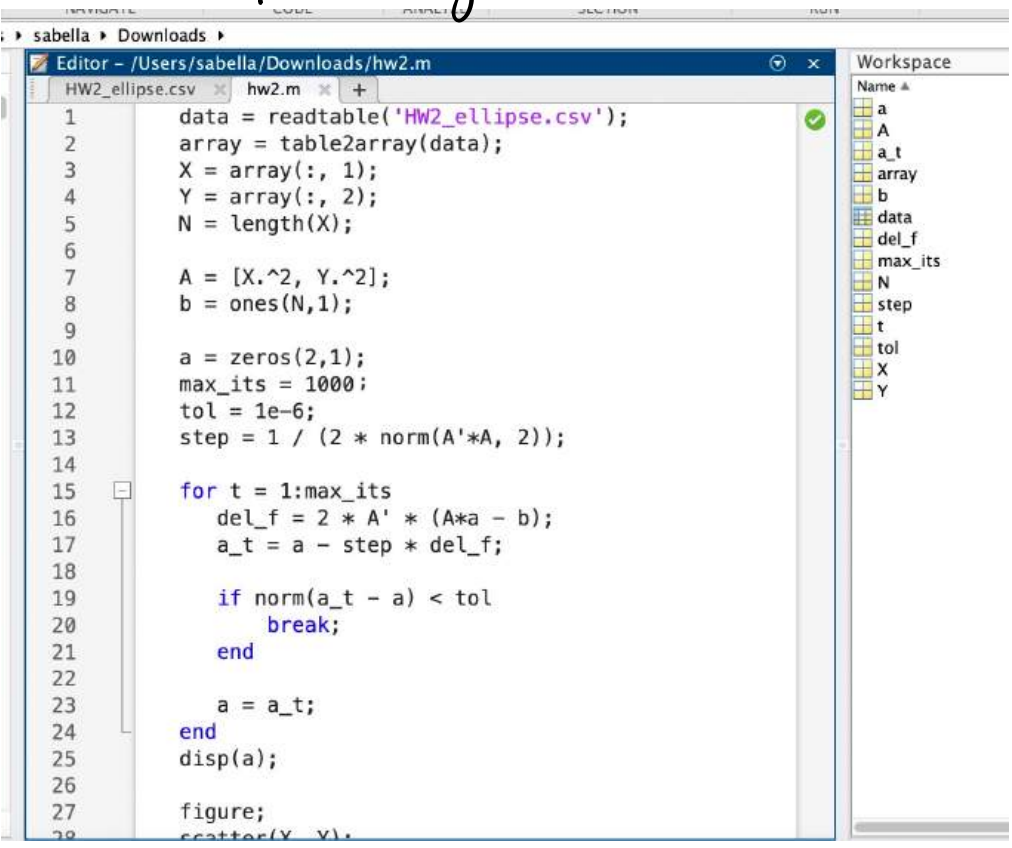
$$\Rightarrow \quad A\ (\in \mathbb{R}^{N\times 2}) \quad a \quad - \quad b\ (\in \mathbb{R}^N)$$

(b) Download the data provided, create scatter plot & submit code & plot





(c) Using Problem 2, create computer code to compute the gradient descent alg on this f(a). The code must include a stopping condition. Use a step-size of $\eta = \frac{1}{2\|A^T A\|}$. Cannot use built-in GD alg, must include for/while loop. Norm of a matrix $\|X\| = \lambda_{max}(X)$, can be computed using "norm(X,2)" in matlab



(d) Using the data provided and your gradient descent code, estimate the solution a. Report a and f(a). Given f(a) and N, do you think you fit the data well or poorly? Given the convexity of f, do you think this is the optimal a?

The estimate for a I found given the data and my code is $a = \begin{bmatrix} 1/2 \\ 2 \end{bmatrix}$.

Command Window

```
>> hw2
   [0.5001]
   [1.9946]  = a

    0.4641   = f(a)

fx >>
```

$$\Rightarrow f(a) = \sum_{i=1}^{N} (a_1 x_i^2 + a_2 y_i^2 - 1)^2$$

$$f\left(\begin{bmatrix} 0.5001 \\ 1.9946 \end{bmatrix}\right) = \sum_{i=1}^{1000} (0.5001 x_i^2 + 1.9946 y_i^2 - 1)^2 = 0.4641$$

we want this = 1, so

$$= \sum_{i=1}^{1000} ((\text{estimate}) - (\text{actual}))^2$$

$$= \sum_{i=1}^{1000} (\text{total error})^2 = 0.4641$$

$$\Rightarrow \frac{f(a)}{N} = \frac{0.4641}{1000} = \frac{\text{total squared error}}{\#\text{ points}} = \underline{0.0004641}$$

The average squared error per point is 0.0004641, which is considerably small compared to the values of (x_i^2, y_i^2) from observing the csv file. Hence I think I fit the data well.

Since $f(a) = \|Aa - b\|_2^2$, we found previously (HW 1, problem 5c and problem 6a) that $f(a)$ is convex. Hence, there is only one global minimizer that minimizes $f(a)$. We found this a by using GD on $f(a)$, so yes, I do believe this a is optimal.