MATH 173A: Homework #2

Due on Oct 22, 2024 at 23:59pm

Professor Cloninger

Ray Tsai

A16848188

Problem 1

Using the conditions of optimality, find the extreme points of the following functions and determine whether they are maxima or minima. 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 $f(x_1, x_2) = x_1^4 + 2x_2^4 - 4x_1x_2$

Proof. Note that

$$\nabla f(x) = (4x_1^3 - 4x_2, 8x_2^3 - 4x_1) = 0$$
$$\nabla^2 f(x) = \begin{bmatrix} 12x_1^2 & -4 \\ -4 & 24x_2^2 \end{bmatrix}.$$

Thus, the critical points are $x^* = (0,0)$ or $\pm(2^{-1/8}, 2^{-3/8})$. We can then check

$$\nabla^2 f(0,0) = \begin{bmatrix} 0 & -4 \\ -4 & 0 \end{bmatrix}$$
$$\nabla^2 f(2^{-1/8}, 2^{-3/8}) = \nabla^2 f(-2^{-1/8}, -2^{-3/8}) = \begin{bmatrix} 12 \cdot 2^{-1/4} & -4 \\ -4 & 24 \cdot 2^{-3/4} \end{bmatrix}.$$

Since the eigenvalues of $\nabla^2 f(0,0)$ are ± 4, $(0,0)$ is a saddle point. Since $\det \nabla^2 f(2^{-1/8}, 2^{-3/8}) = \det \nabla^2 f(-2^{-1/8}, -2^{-3/8}) = 128 > 0$ and $\frac{\partial^2 f}{\partial x_1^2} > 0$, the critical points $\pm(2^{-1/8}, 2^{-3/8})$ are local minima. □

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

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

Proof.

$$\nabla f(\vec{x}) = 2A\vec{x} + b,$$
$$\nabla^2 f(\vec{x}) = 2A.$$

Setting $\nabla f(\vec{x}) = 0$ yields

$$\vec{x}^* = -\frac{1}{2}A^{-1}b = \begin{bmatrix} \frac{2}{3} \\ \frac{1}{2} \\ \frac{4}{3} \end{bmatrix}.$$

Since the eigenvalues of $\nabla^2 f(\vec{x}) = 2A$ are $-1, -2, -3$, we know $\nabla^2 f(\vec{x}) \prec 0$ and the critical point is a local maximum. □

Problem 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

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

Proof. We already know f is convex and the gradient of f is

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

Thus, the gradient descent algorithm is

$$\begin{aligned} x^{(0)} &= \text{any vector from } \mathbb{R}^n \\ x^{(t+1)} &= x^{(t)} - \mu^{(t)} \nabla f(x^{(t)}) = x^{(t)} - 2\mu^{(t)} A^T (Ax^{(t)} - b) \end{aligned}$$

where $\mu^{(t)} > 0$ and the terminating condition is of our choice. $\square$

Problem 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

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

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

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

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

Proof. By the chain rule,

$$\nabla F(w) = \frac{1}{N} \sum_{i=1}^N \frac{-y_i e^{-\langle w, x_i \rangle y_i}}{1 + e^{-\langle w, x_i \rangle y_i}} x_i$$
$$\nabla^2 F(w) = \frac{1}{N} \sum_{i=1}^N \frac{y_i^2 e^{-\langle w, x_i \rangle y_i}}{(1 + e^{-\langle w, x_i \rangle y_i})^2} x_i x_i^T.$$

Since $x_i x_i^T \succeq 0$, $\frac{y_i^2 e^{-\langle w, x_i \rangle y_i}}{(1+e^{-\langle w, x_i \rangle y_i})^2} \geq 0$, and the sum of positive semidefinite matrices is positive semidefinite, $\nabla^2 F(w) \succeq 0$. Thus, $F(w)$ is convex. □

(b) Find a gradient descent algorithm for minimizing F.

Proof. The gradient descent algorithm is

$$w^{(0)} = \text{any vector from } \mathbb{R}^d$$
$$w^{(t+1)} = w^{(t)} - \mu^{(t)} \nabla F(w^{(t)}) = w^{(t)} + \frac{\mu^{(t)}}{N} \sum_{i=1}^N \frac{y_i e^{-\langle w^{(t)}, x_i \rangle y_i}}{1 + e^{-\langle w^{(t)}, x_i \rangle y_i}} x_i$$

where $\mu^{(t)} > 0$ and the terminating condition is of our choice. □

Problem 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

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

where

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

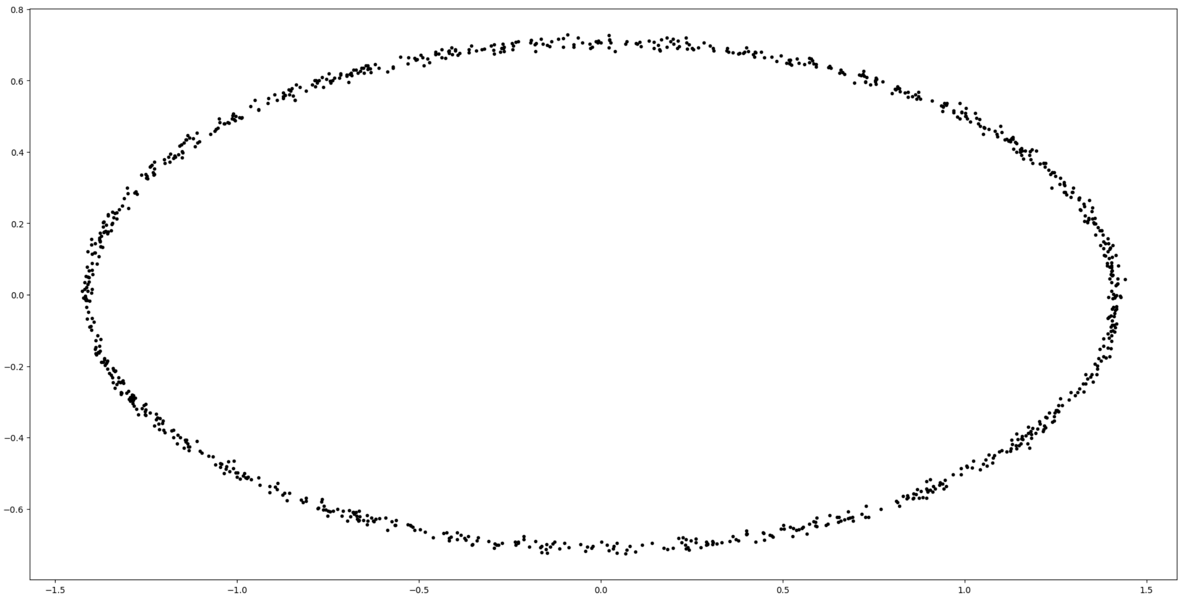
(a) Find an $A \in \mathbb{R}^{N \times 2}$ and $b \in \mathbb{R}^N$ such that $f(\mathbf{a}) = \|A\mathbf{a} - b\|_2^2$. What is A in terms of (x_i, y_i)?

Proof. Put $A = \begin{bmatrix} x_1^2 & y_1^2 \\ \vdots & \vdots \\ x_N^2 & y_N^2 \end{bmatrix}$ and $b = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$ and we have

$$\begin{aligned} f(\mathbf{a}) &= \sum_{i=1}^N \left(a_1 x_i^2 + a_2 y_i^2 - 1\right)^2 \\ &= \sum_{i=1}^N \left(\begin{bmatrix} x_i^2 & y_i^2 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} - 1 \right)^2 \\ &= \|A\mathbf{a} - b\|_2^2. \end{aligned}$$

□

(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).



Problem 4 continued on next page. . .

Code:

```
ellipse = pd.read_csv('HW2_ellipse.csv', header=None)
ellipse.columns = ['x', 'y']
fig = plt.figure(figsize=(24, 12))
ax = fig.add_subplot(1, 1, 1)
ax.plot(ellipse['x'], ellipse['y'], '.', color='black')
```

(c) Using Problem 3, create computer code to compute the gradient descent algorithm on this $f(\mathbf{a})$. The code must include a stopping condition. Use a step-size of $\mu = \frac{1}{2\|\mathbf{A}^T\mathbf{A}\|}$. 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 $\|\mathbf{X}\| = \lambda_{\max}(\mathbf{X})$ is the largest eigenvalue of $\mathbf{X}$, and can be computed using `norm(X,2)` in MATLAB or `np.linalg.norm(X,2)` in Python. (Submit the code)

Code:

```
A = ellipse.values * ellipse.values
b = np.ones((A.shape[0], 1))
mu = 1/(2 * np.linalg.norm(A.T @ A, 2))
a = np.random.rand(2, 1)

def df(x):
    return 2 * A.T @ (A @ x - b)

for i in range(1000):
    a = a - mu * df(a)
```

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

Solution. The solution estimated by the code is $\mathbf{a} \approx \begin{bmatrix} 0.5001 \\ 1.9946 \end{bmatrix}$, with $f(\mathbf{a}) \approx 0.4641$. Since we are given $N = 1000$ points, on average each point is only off by $f(\mathbf{a})/N \approx 0.0004641$, which is a small enough number to me. Given that $\Delta f(\mathbf{a}) \approx 0$, I believe $\mathbf{a}$ has reached a local minimum up to some computational error. Since f is convex, $\mathbf{a}$ is extremely close to the theoretical optimal solution. $\square$