COMPSCI 371 Homework 2

Group Members: Mayur Sekhar, Jai Kasera, Rithvik Neti

Problem 0 (3 points)

Part 1: A Tiny Linear Regression Problem

Problem 1.1 (Exam Style)

Power losses with odd positive integer powers (e.g., l_p norms for $3, 5, \ldots$) are not adequate measures of loss because they can result in asymmetric penalization of positive and negative errors. Odd powers do not treat overestimation and underestimation equally, which is undesirable in most cases where both types of error are equally important.

In contrast, even powers like $p = 2$ (as in mean squared error) treat positive and negative deviations symmetrically, making them more suitable for measuring loss in many applications

Problem 1.2 (Exam Style)

The z value that would have the smallest residual risk is $z = 0$. In this case, the value of the residual risk when $z = 0$ would also be 0. The associated values of b and w for this given z would also both be 0. If $z = 0$ then the function would simply be the horizontal line of the x axis, and it would would have no loss as all the points of the data would be on the line.

Problem 1.3 (Exam Style)

To solve this problem, we need to write the total risk $L_T(v)$ as a function of the parameters b and w, and then derive the normal equations by setting the gradients with respect to b and w to zero.

The dataset is: $T = \{(-1, 0), (0, z), (1, 0)\}$.

We aim to fit a line $y = mx + b$ and minimize the loss using the power loss p, where p is a positive even integer.

The loss for each point is given by:

For $(x_1, y_1) = (-1, 0)$:

$L_1 = |w(-1) + b - 0|^p = | - w + b|^p$

For $(x_2, y_2) = (0, z)$:

$L_2 = |w(0) + b - z|^p = |b - z|^p$

For $(x_3, y_3) = (1, 0)$:

$L_3 = |w(1) + b - 0|^p = |w + b|^p$

Thus, the total risk $L_T(v)$ is the sum of these losses:

$L_T(v) = \frac{1}{3}| - w + b|^p + |b - z|^p + |w + b|^p$

To minimize $L_T(v)$, we take the partial derivatives of $L_T(v)$ with respect to b and w, and set them to zero.

Derivative with respect to b:

$\frac{\partial L_T}{\partial b} = p \cdot (-w + b)^{p-1} \cdot 1 + p \cdot (b - z)^{p-1} + p \cdot (w + b)^{p-1} = 0$

$\implies p[(-w + b)^{p-1} + (b - z)^{p-1} + (w + b)^{p-1}] = 0$

$\implies (-w + b)^{p-1} + (b - z)^{p-1} + (w + b)^{p-1} = 0$

Derivative with respect to w:

$\frac{\partial L_T}{\partial w} = p \cdot (-w + b)^{p-1} \cdot (-1) + p \cdot (w + b)^{p-1} \cdot 1 = 0$

$\implies p[(-w + b)^{p-1}(-1) + (w + b)^{p-1}] = 0$

$\implies (-w + b)^{p-1}(-1) + (w + b)^{p-1} = 0$

Substitution in when $p = 2$:

$\frac{\partial L_T}{\partial b} = (-w + b)^{2-1} + (b - z)^{2-1} + (w + b)^{2-1} = 0$

$\frac{\partial L_T}{\partial b} = (-w + b)^1 + (b - z)^1 + (w + b)^1 = 0$

$\frac{\partial L_T}{\partial b} = -w + b + b - z + w + b = 0$

$\frac{\partial L_T}{\partial b} = 3b - z = 0$

$\therefore b = \frac{z}{3}$

$\frac{\partial L_T}{\partial w} = (-w + b)^{2-1}(-1) + (w + b)^{2-1} = 0$

$\frac{\partial L_T}{\partial w} = (-w + b)^1(-1) + (w + b)^1 = 0$

$\frac{\partial L_T}{\partial w} = w - b + w + b = 0$

$\frac{\partial L_T}{\partial w} = 2w = 0$

$\therefore w = 0$

Equation for $L_T(v)$:

$L_T(v) = \frac{1}{3}(| - w + b|^p + |b - z|^p + |w + b|^p)$

$L_T(v) = \frac{1}{3}(|0 + \frac{z}{3}|^2 + |\frac{z}{3} - z|^2 + |0 + \frac{z}{3}|^2)$

$L_T(v) = \frac{1}{3}((\frac{z}{3})^2 + (\frac{2z}{3})^2 + (\frac{z}{3})^2)$

$L_T(v) = \frac{2z^2}{9}$

Thus for $p = 2$ and arbitrary z the solution to the normal equations is: $b = \frac{z}{3}$ and $w = 0$. Thus, the best-fitting line is $y = \frac{z}{3}$ Also, $L_T(v) = \frac{2z^2}{9}$.

Problem 1.4

To find the optimal values of w and b when $p = 4$ in the regression problem, we start by using the same data:

The dataset is: $T = \{(-1, 0), (0, z), (1, 0)\}$.

The linear model is $y = mx + b$, and we aim to minimize the fourth power loss $p = 4$.

The loss for each point is given by:

For $(x_1, y_1) = (-1, 0)$:

$L_1 = |w(-1) + b - 0|^4 = | - w + b|^4$

For $(x_2, y_2) = (0, z)$:

$L_2 = |w(0) + b - z|^4 = |b - z|^4$

For $(x_3, y_3) = (1, 0)$:

$L_3 = |w(1) + b - 0|^4 = |w + b|^4$

Thus, the total risk $L_T(v)$ is the sum of these losses:

$L_T(v) = | - w + b|^4 + |b - z|^4 + |w + b|^4$

To minimize $L_T(v)$, we take the partial derivatives of $L_T(v)$ with respect to b and w, and set them to zero.

b:

$\frac{\partial L_T}{\partial b} = (-w+b)^{4-1} + (b-z)^{4-1} + (w+b)^{4-1} = 0$

$\frac{\partial L_T}{\partial b} = (-w+b)^{3} + (b-z)^{3} + (w+b)^{3} = 0$

w:

$\frac{\partial L_T}{\partial w} = (-w+b)^{4-1}(-1) + (w+b)^{4-1} = 0$

$\frac{\partial L_T}{\partial w} = (-w+b)^{3}(-1) + (w+b)^{3} = 0$

$(-w+b)^3 = (w+b)^3$

Solving $(-w+b)^3 = (w+b)^3$

This implies either:

$-w+b = w+b$ which simplifies to $w = 0$, or

$-w+b = -w-b$ which simplifies to $b = 0$.

Since both can't be true at the same time, we first explore $w = 0$.

Case 1: $w = 0$

Substituting $w = 0$ into the equation for b:

$b^3 + (b-z)^3 + b^3 = 0$

$\Rightarrow 2b^3 + (b-z)^3 = 0$

Using Wolfram Alpha, the only real solution to this cubic equation is
$b = \frac{1}{3}(1 - \sqrt[3]{2} + 2^{\frac{2}{3}})z$

Numerically, this is equivalent to $0.44249z$.

So when $w = 0$, we get:

$b = 0.44249z$

Case 2: $b = 0$

Substituting $b = 0$ into the derivative with respect to b yields:

$(-w)^3 + (-z)^3 + (w)^3 = 0$

$-w^3 + (-z)^3 + w^3 = 0$

$-z^3 = 0 \Rightarrow z = 0$

In this case, if $z = 0$ then $b = 0$ and $w = 0$.

Therefore the analytic solution for $p = 4$ and arbitrary z is:

$w = 0$

$b = \frac{1}{3}(1 - \sqrt[3]{2} + 2^{\frac{2}{3}})z = 0.44249z$

In comparison to $p = 2$ where:

$w = 0$

$b = \frac{z}{3}$

Thus, for $p = 4$, the predictor $y = 0.44249z$ is closer to the second sample $(0, z)$ by 0.10916 than the predictor found for $p = 2$, where $y = \frac{z}{3}$. This is because the $p = 2$ model is farther from the second sample (it's "pulled" more towards the other points at $y = 0$).

Part 2: General Univariate Linear Regression

To show that the least-squares solution given by:

$$b = \frac{s_y s_{xx} - s_x s_{xy}}{N s_{xx} - s_x^2}$$

and

$$w = \frac{N s_{xy} - s_x s_y}{N s_{xx} - s_x^2}$$

is equivalent to the solution of the normal equations:

$$A^T A \begin{bmatrix} b \\ w \end{bmatrix} = A^T y,$$

we proceed as follows:

1. Set Up the Normal Equations

Given:

$$A = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_N \end{bmatrix}, \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

The normal equations for least-squares regression are:

$$A^T A \begin{bmatrix} b \\ w \end{bmatrix} = A^T y$$

2. Compute $A^T A$

First, compute (A^T):

$$A^T = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_N \end{bmatrix}$$

Next, compute (A^T A):

$$A^T A = \begin{bmatrix} \sum_{n=1}^{N} 1 \cdot 1 & \sum_{n=1}^{N} 1 \cdot x_n \\ \sum_{n=1}^{N} x_n \cdot 1 & \sum_{n=1}^{N} x_n \cdot x_n \end{bmatrix} = \begin{bmatrix} N & \sum_{n=1}^{N} x_n \\ \sum_{n=1}^{N} x_n & \sum_{n=1}^{N} x_n^2 \end{bmatrix}$$

3. Compute $A^T y$

$$A^T y = \begin{bmatrix} \sum_{n=1}^{N} 1 \cdot y_n \\ \sum_{n=1}^{N} x_n \cdot y_n \end{bmatrix} = \begin{bmatrix} \sum_{n=1}^{N} y_n \\ \sum_{n=1}^{N} x_n y_n \end{bmatrix}$$

4. Set Up the Normal Equations

Substitute (A^T A) and (A^T y) into the normal equations:

$$\begin{bmatrix} N & \sum_{n=1}^{N} x_n \\ \sum_{n=1}^{N} x_n & \sum_{n=1}^{N} x_n^2 \end{bmatrix} \begin{bmatrix} b \\ w \end{bmatrix} = \begin{bmatrix} \sum_{n=1}^{N} y_n \\ \sum_{n=1}^{N} x_n y_n \end{bmatrix}$$

5. Solve for b and w

The system of equations is:

$$\begin{aligned} Nb + \left(\sum_{n=1}^{N} x_n \right) w &= \sum_{n=1}^{N} y_n \\ \left(\sum_{n=1}^{N} x_n \right) b + \left(\sum_{n=1}^{N} x_n^2 \right) w &= \sum_{n=1}^{N} x_n y_n \end{aligned}$$

Rearrange to solve for b and w:

1. Solve the first equation for b:

$$b = \frac{\sum_{n=1}^{N} y_n - \left(\sum_{n=1}^{N} x_n\right) w}{N}$$

2. Substitute this expression for b into the second equation:

$$\sum_{n=1}^{N} x_n \left(\frac{\sum_{n=1}^{N} y_n - \left(\sum_{n=1}^{N} x_n\right) w}{N} \right) + \left(\sum_{n=1}^{N} x_n^2 \right) w = \sum_{n=1}^{N} x_n y_n$$

$$\frac{\sum_{n=1}^{N} x_n \sum_{n=1}^{N} y_n - \left(\sum_{n=1}^{N} x_n\right)^2 w}{N} + \left(\sum_{n=1}^{N} x_n^2 \right) w = \sum_{n=1}^{N} x_n y_n$$

$$\sum_{n=1}^{N} x_n \sum_{n=1}^{N} y_n - \left(\sum_{n=1}^{N} x_n \right)^2 w + N \left(\sum_{n=1}^{N} x_n^2 \right) w = N \sum_{n=1}^{N} x_n y_n$$

$$\left(N \sum_{n=1}^{N} x_n^2 - \left(\sum_{n=1}^{N} x_n \right)^2 \right) w = N \sum_{n=1}^{N} x_n y_n - \sum_{n=1}^{N} x_n \sum_{n=1}^{N} y_n$$

$$w = \frac{N \sum_{n=1}^{N} x_n y_n - \sum_{n=1}^{N} x_n \sum_{n=1}^{N} y_n}{N \sum_{n=1}^{N} x_n^2 - \left(\sum_{n=1}^{N} x_n \right)^2}$$

3. Substitute w back into the expression for b:

$$b = \frac{\sum_{n=1}^{N} y_n - \left(\sum_{n=1}^{N} x_n\right) \left(\frac{N \sum_{n=1}^{N} x_n y_n - \sum_{n=1}^{N} x_n \sum_{n=1}^{N} y_n}{N \sum_{n=1}^{N} x_n^2 - \left(\sum_{n=1}^{N} x_n \right)^2} \right)}{N}$$

$$b = \frac{\sum_{n=1}^{N} y_n \sum_{n=1}^{N} x_n^2 - \sum_{n=1}^{N} x_n \sum_{n=1}^{N} y_n \sum_{n=1}^{N} x_n}{N \sum_{n=1}^{N} x_n^2 - \left(\sum_{n=1}^{N} x_n \right)^2}$$

The least-squares solutions for (b) and (w) derived from the normal equations match the given formulas:

$$b = \frac{s_y s_{xx} - s_x s_{xy}}{N s_{xx} - s_x^2}$$
$$w = \frac{N s_{xy} - s_x s_y}{N s_{xx} - s_x^2}$$

where:

$$s_x = \sum_{n=1}^{N} x_n$$
$$s_y = \sum_{n=1}^{N} y_n$$
$$s_{xx} = \sum_{n=1}^{N} x_n^2$$
$$s_{xy} = \sum_{n=1}^{N} x_n y_n$$

This confirms that the least-squares solution for the linear regression parameters (b) and (w) is consistent with the derived formulas using the normal equations.

Part 3: Automatic Differentiation

In [1]:
```
from autograd import numpy as anp
from autograd import elementwise_grad as grad
```

In [2]:
```
def bumps(z):
    assert len(z) == 2, 'input must be 2D'
    u, v = z[0], z[1]
    a = (4 - 2.1 * u ** 2 + (u ** 4) / 3) * u ** 2
    b = u * v
    c = (-4 + 4 * v ** 2) * v ** 2
    return a + b + c
```

In [3]:
```
import numpy as snp


x_min, x_max, y_min, y_max = -2.1, 2.1, -1.2, 1.2
n = 301
xx, yy = snp.linspace(x_min, x_max, n), snp.linspace(y_min, y_max, n)
x = snp.array(snp.meshgrid(xx, yy))
```

In [4]:
```
import urllib.request
import ssl
from os import path as osp
import shutil


def retrieve(file_name, semester='fall24', homework=2):
    if osp.exists(file_name):
        print('Using previously downloaded file {}'.format(file_name))
    else:
        context = ssl._create_unverified_context()
        fmt = 'https://www2.cs.duke.edu/courses/{}/compsci371/homework/{}/{}
        url = fmt.format(semester, homework, file_name)
        with urllib.request.urlopen(url, context=context) as response:
            with open(file_name, 'wb') as file:
```

```
                shutil.copyfileobj(response, file)
        print('Downloaded file {}'.format(file_name))
```

In [5]:
```
retrieve('helpers.py')
```

```
Using previously downloaded file helpers.py
```

In [6]:
```
from helpers import fixed_axis_figure
from matplotlib import pyplot as plt
%matplotlib inline
```

Problem 3.1

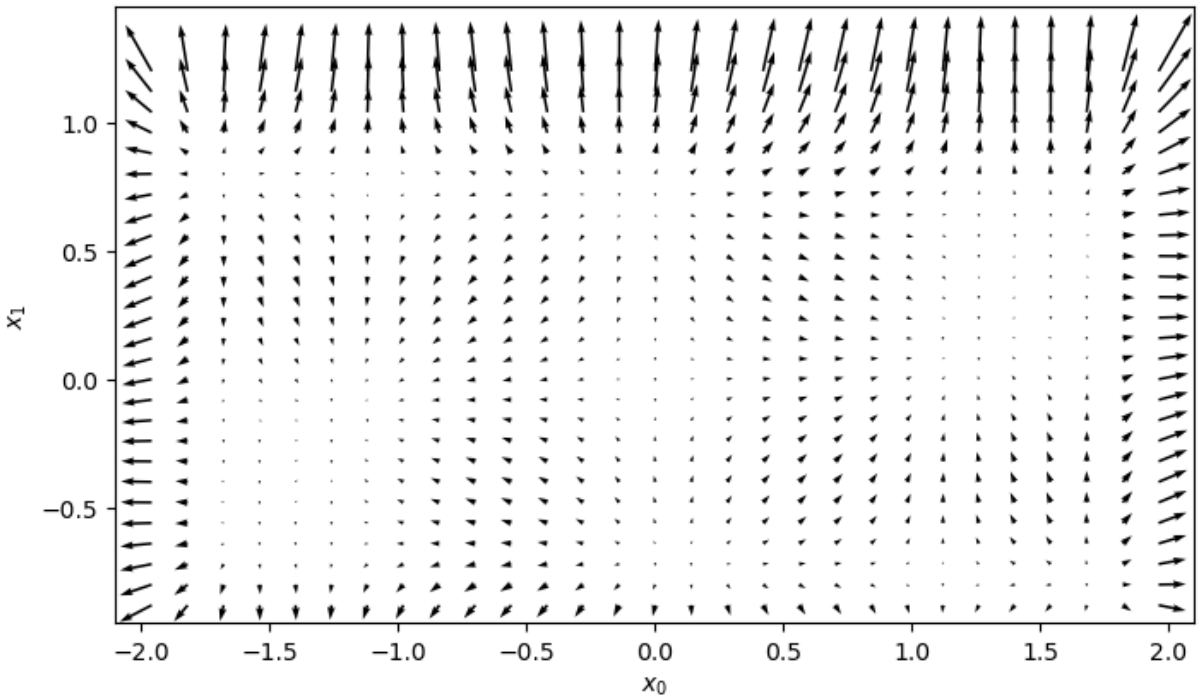
In [7]:
```
step = 10
s = x[:, ::step, ::step]
```

In [8]:
```
margin = 0.25
quiver_box = snp.array([x_min, x_max, y_min, y_max])
quiver_box[2:] = quiver_box[2:] + margin

g = grad(bumps)
gradients = g(s)

ax = fixed_axis_figure(quiver_box, scale=1.5)[1]
ax.quiver(s[0], s[1], gradients[0], gradients[1])
plt.xlabel(r'$x_0$')
plt.ylabel(r'$x_1$')
```

Out[8]:
```
Text(0, 0.5, '$x_1$')
```



Part 4: Gradient Descent

In [9]:
```
from helpers import Stepper
from autograd import numpy as anp


def gradient_descent(
    f, z, alpha, min_step=1.e-6, max_iter=10000, history=False, **kwargs
):
    step = Stepper(f, z, alpha, history=history, **kwargs)
    z, fz, gz = anp.copy(z), step.fz0, step.gz0
    for k in range(max_iter):
        s, z, fz, gz = step(z, **kwargs)
        if anp.linalg.norm(s) < min_step:
            break
    step.show_history()
    return fz, z, k
```

In [10]:
```
alpha = 1.e-3
```

Problem 4.1

In [11]:
```
v0 = anp.array((1, 1), dtype=float)
```

In [12]:
```
x = anp.array([-1, 0, 1], dtype=float)[:, None]
```

In [13]:
```
y = anp.array([0, 1, 0], dtype=float)
```

In [14]:
```
def p_risk(v, x=x, y=y, p=2):
    b, w = v[0], v[1:]
    y_pred = b + w * x.flatten()
    return anp.mean((y_pred - y)**p)

for p in (2,4,6):
    risk_ast, v_ast, iterations =  \
        gradient_descent(p_risk, v0, alpha, x=x, y=y, p=p)
    with snp.printoptions(precision=4, suppress=True):
        print(f'p = {p}: [b, w] = {v_ast} {risk_ast: .4f}')
```

```
p = 2: [b, w] = [0.3333 0.0007]  0.2222
p = 4: [b, w] = [0.4425 0.0006]  0.0578
p = 6: [b, w] = [0.4654 0.0011]  0.0146
```