Exercise 1

Question 1

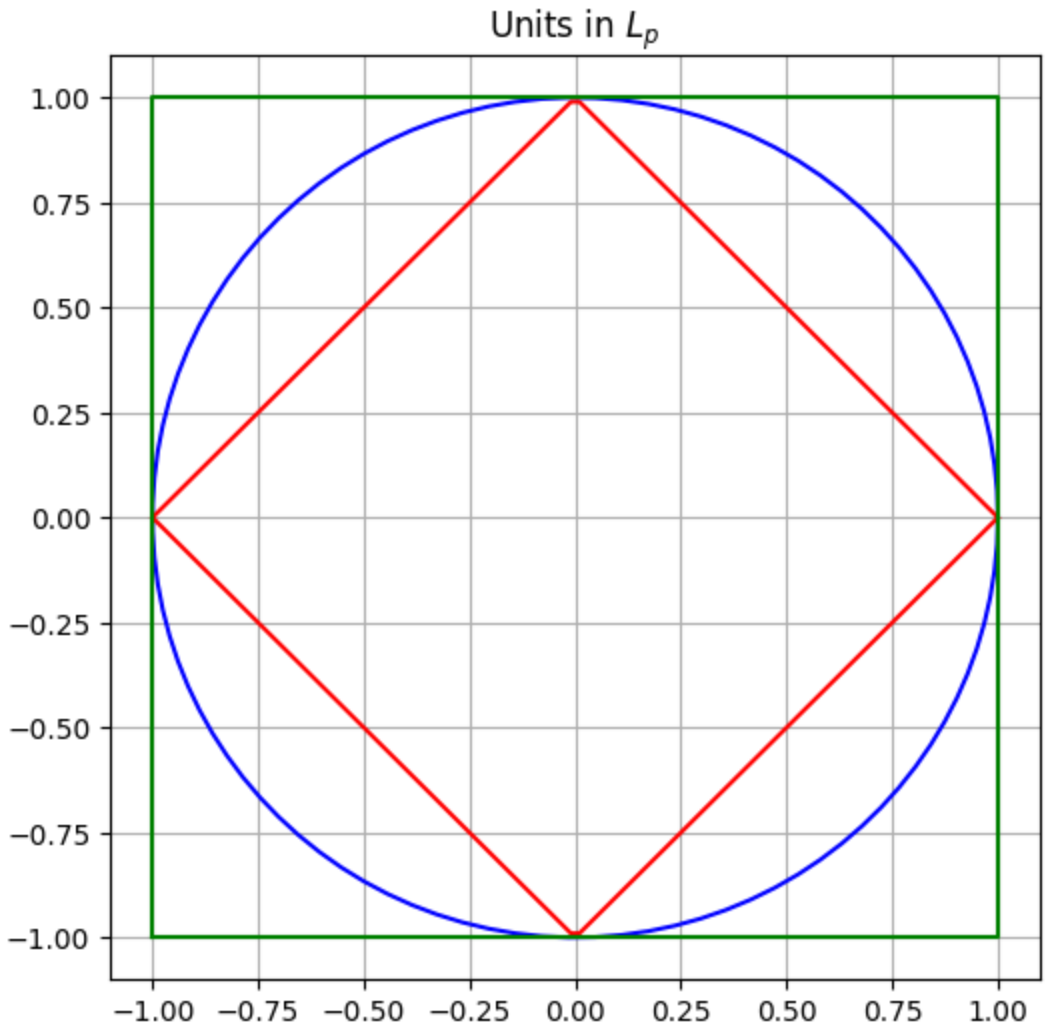
Consider the set of points $\{x \in \mathbb{R}^2 : \|x\|_p = 1\}$

a. Plot $p = 2$. The unit sphere in L^2 (blue)

b. On the same figure, plot the unit sphere in L^1 (red)

c. Same plot do $p = \infty$ (green)

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

a = np.linspace(0, 2*np.pi, 100)
x1_a = np.cos(a)
x2_a = np.sin(a)

b = np.linspace(-1, 1, 100)
x1_b = 1 - np.abs(b)
x2_b = -1 + np.abs(b)

p = np.array([[-1, -1], [-1, 1], [1, 1], [1, -1], [-1, -1]])

# Plot the unit circle
plt.figure(figsize=(6,6))
plt.plot(x1_a, x2_a, color='b')
plt.plot(b, x1_b, b, x2_b, color='r')
plt.plot(p[:, 0], p[:, 1], color='g')

plt.title('Units in $L_p$')
plt.grid(True)
plt.show()
```



d. As the unit spheres increase from 1 to infinity, they get closer and closer to a unit square.

Question 2

Let $x, y \in \mathbb{R}^{n}$'

a. Express $||x - y||_2^2$ using the dot product

$$||x - y||_2^2 = (\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2})^2$$

$$||x - y||_2^2 = \sum_{i=1}^{n}(x_i - y_i)^2$$

$$||x - y||_2^2 = \sum_{i=1}^{n}(x_i - y_i)(x_i - y_i)$$

$$||x - y||_2^2 = \sum_{i=1}^{n}(x_i^2 - 2x_iy_i + y_i^2)$$

$\|x-y\|_2^2 = \sum_{i=1}^n x_i^2 - 2\sum_{i=1}^n x_i y_i + \sum_{i=1}^n y_i^2$

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

Question 3

In linear algebra, a square matrix P is called a projection matrix if $P^2 = P$

(a) Suppose that $V \in \mathbb{R}^{3\times 2}$ is a matrix with orthonormal columns. Recall from class that VV^Tx is the vector projection of a vector $x \in \mathbb{R}^3$ onto the column space of V. Show that VV^T is a projection matrix.

$(VV^T)^T = V^T(V^T)^T = VV^T$ so VV^T is symmetric and thus square (2, 2)

$(VV^T)^2 = (VV^T)(VV^T) = V(V^TV)V^T = VV^T$ so then it is a projection matrix

(b) Projection matricies always exist only in the direction they are projected onto as all inner V^TV cancel to the identity. Meaning, any $P^k = P$.

Question 4

$f(x) = \frac{1}{2}x^T \begin{bmatrix} 1 & 4 \\ 0 & 1 \end{bmatrix} x + \begin{bmatrix} 1 \\ 2 \end{bmatrix}^T x + 1$

(a) Gradient of $f(x)$ would be

$\nabla f(x) = \frac{\partial}{\partial x}(\frac{1}{2}x^T \begin{bmatrix} 1 & 4 \\ 0 & 1 \end{bmatrix} x) + \frac{\partial}{\partial x}(\begin{bmatrix} 1 \\ 2 \end{bmatrix}^T x) + \frac{\partial}{\partial x}(1)$

$\nabla f(x) = \frac{1}{2}(\begin{bmatrix} 1 & 4 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 4 \\ 0 & 1 \end{bmatrix}^T)x + \begin{bmatrix} 1 \\ 2 \end{bmatrix}$

$\nabla f(x) = \frac{1}{2}(\begin{bmatrix} 1 & 4 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 4 & 1 \end{bmatrix})x + \begin{bmatrix} 1 \\ 2 \end{bmatrix}$

$\nabla f(x) = \frac{1}{2}(\begin{bmatrix} 2 & 4 \\ 4 & 2 \end{bmatrix})x + \begin{bmatrix} 1 \\ 2 \end{bmatrix}$

$$\nabla f(x) = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} x + \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

(b) Hessian of $f(x)$ would be

$$\nabla^2 f(x) = \frac{1}{2}(\begin{bmatrix} 1 & 4 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 4 \\ 0 & 1 \end{bmatrix}^T)$$

$$\nabla^2 f(x) = \frac{1}{2}(\begin{bmatrix} 1 & 4 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 4 & 1 \end{bmatrix})$$

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

(c) $f(x)$ is convex if eigenvalues arent 0

$$\nabla^2 f(x) = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} = Q\Lambda Q^T$$

$$\Lambda = \begin{bmatrix} 3 & 0 \\ 0 & -1 \end{bmatrix}$$

Thus $f(x)$ is convex

Question 5

In Newtons method, we also need to compute the inverse of the Hessian in order to find delta. The advantage is that it is very fast to approximate the vertex once you have these. The disadvantage is that you need to compute the Hessian.

Exercise 2

Question 1

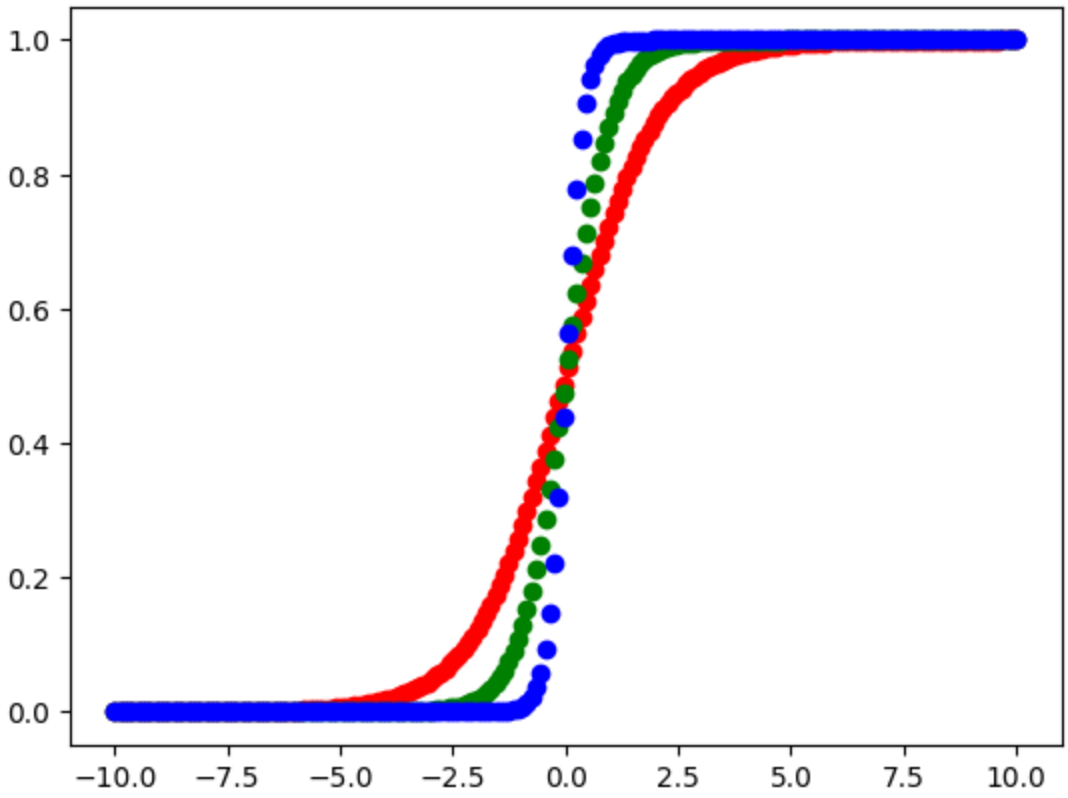
Logistic regression is modelled by

$$P(Y=1|x;\theta) = g(\theta^T x) = \frac{1}{1+e^{-\theta^T x}}$$

(a) Suppose $x, \theta \in \mathbb{R}$. Plot $h_\theta(x) = g(\theta^T x)$ for $\theta = 1$, 2, and 5.

```
def log_reg(x, theta):
  proba = 1/(1 + np.exp(-np.dot(theta, x)))
  return proba


x = np.linspace(-10, 10, 200)
y1 = log_reg(x, 1)
y2 = log_reg(x, 2)
y5 = log_reg(x, 5)

plt.scatter(x,y1, color='r')
plt.scatter(x,y2 , color='g')
plt.scatter(x,y5, color='b')
```

```
<matplotlib.collections.PathCollection at 0x7fe9035604c0>
```



As we increase our theta, our shape becomes sharper and sharper. At theta=infinity, it is a discontinuous jump between the two classes at the border.

(b) If we choose an extremely large theta (closer to infinity or infinity) then our model will be separated by a line/hyperplane.

Question 2

(a) if $m = 1$ then $\ell(\theta) = ylogh_\theta(x) + (1-y)log(1-h_\theta(x))$

And $h_\theta(x) = g(\theta^T x)$

$\ell(\theta) = y \log g(\theta^T x) + (1-y) \log(1 - g(\theta^T x))$

$\frac{\partial}{\partial \theta_j} \ell(\theta) = \frac{\partial}{\partial \theta_j} (y \log g(\theta^T x) + (1-y) \log(1 - g(\theta^T x)))$

$\frac{\partial}{\partial \theta_j} \ell(\theta) = \frac{\partial}{\partial \theta_j} y \log g(\theta^T x) + \frac{\partial}{\partial \theta_j} (1-y) \log(1 - g(\theta^T x))$

$\frac{\partial}{\partial \theta_j} \ell(\theta) = \frac{y}{g(\theta^T x)} \frac{\partial}{\partial \theta_j} g(\theta^T x) + \frac{1-y}{1 - g(\theta^T x)} \frac{\partial}{\partial \theta_j} (1 - g(\theta^T x))$

As $g'(z) = g(z)(1 - g(z))$

$\frac{\partial}{\partial \theta_j} \ell(\theta) = \frac{y}{g(\theta^T x)} g(\theta^T x)(1 - g(\theta^T x)) x_j + \frac{1-y}{1 - g(\theta^T x)} - g(\theta^T x)(1 - g(\theta^T x) x_j)$

$\frac{\partial}{\partial \theta_j} \ell(\theta) = y(1 - g(\theta^T x)) - (1-y) g(\theta^T x) x_j$

$\frac{\partial}{\partial \theta_j} \ell(\theta) = (y - yg(\theta^T x) - g(\theta^T x) + yg(\theta^T x)) x_j$

$\frac{\partial}{\partial \theta_j} \ell(\theta) = (y - g(\theta^T x)) x_j$

Finally, as $h_\theta(x) = g(\theta^T x)$

$\frac{\partial}{\partial \theta_j} \ell(\theta) = (y - h_\theta(x)) x_j$

(b) Each component should be given by the above where $\frac{\partial}{\partial \theta_m}\ell(\theta) = (y - h_\theta(x))x_m$ so $((y - h_\theta(x))x_1, (y - h_\theta(x))x_2, \ldots, (y - h_\theta(x))x_m)$

Question 3

(a)

```
import numpy as np
X = np.genfromtxt('./sample_data/xvals.dat')
print(X.shape)
```

```
(99, 2)
```

X has 99 datapoints and each datapoint has 2 features.

```
yvals = np.genfromtxt('./sample_data/yvals.dat')
print(yvals.shape)
```

```
(99,)
```

Add constant feature of 1 at the front

```
X_b = np.insert(X, 0, 1, axis=1)
```

```
print(X_b.shape)
```

```
(99, 3)
```

(c) Gradient Ascent

```
def g(z):
  return 1/(1+np.exp(-z))
```

```
def gradient_ascent(X, y, alpha):
  tolerance = 1e-6
  m, n = X.shape
  theta = np.zeros(n)
  steps = 0
  while True:
    z = np.dot(X, theta)
    grad = np.mean((y - g(z))*X.T, axis=1)
    if np.linalg.norm(grad, 2) <= tolerance:
      break
    theta += alpha*grad
    steps+=1

  return theta, steps


np.random.seed(0)
theta, steps = gradient_ascent(X_b, yvals, 0.01)


print(steps)
```

```
62103
```

With a learning rate of 0.01 we needed 62103 steps

```
theta, steps = gradient_ascent(X_b, yvals, 0.1)
print(steps)
```

```
6206
```

With a learning rate of 0.1 we only needed 6206 steps

```
theta, steps = gradient_ascent(X_b, yvals, 0.001)
print(steps)
```

```
621080
```

With a learning rate of 0.001 it took 621080 steps, much more.

The smaller the learning rate, the greater number of steps it needs potentiallly needs to take to converge if it starts far away from convergence

```
print(theta)
```

```
[-2.62045303  0.76035866  1.17194243]
```

(c) and (d)

Because our red point is above the decision boundary we expect it to be in class 1, the blue class.

Thus $P(y = 1|x = [2\ 1]^T; \theta^*) = 1$ and $P(y = 1|x = [2\ 1]^T; \theta^*) = 0$

```
plt.scatter(X_b[np.where(yvals == 1)][:, 1], X_b[np.where(yvals == 1)][:, 2], color='b')
plt.scatter(X_b[np.where(yvals == 0)][:, 1], X_b[np.where(yvals == 0)][:, 2], color='g')
x1 = np.array([np.min(X_b[:, 1]), np.max(X_b[:, 1])])
plt.plot(x1, -(theta[0] + theta[1]*x1)/theta[2])

plt.scatter(2, 1, color='r')
```

```
<matplotlib.collections.PathCollection at 0x7fe902077f10>
```