Akash Patel
MFML-HMWK 10

2. Suppose the random variables (X, Y), $X \in \mathbb{R}^2, Y \in \{1,2\}$, have joint distribution given by
$P(Y=1) = P(Y=2) = \frac{1}{2}$ $\quad f_X(x \mid Y=k) = \frac{1}{2\pi\sqrt{|\Sigma_k|}} \exp\left(-\frac{1}{2}(x-\mu_k)^T \Sigma_k^{-1}(x-\mu_k)\right)$,
where $\mu_1 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$, $\mu_2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, $\Sigma_1 = \begin{bmatrix} 3 & -6 \\ -6 & 24 \end{bmatrix}$, $\Sigma_2 = \begin{bmatrix} 16 & -6 \\ -6 & 8 \end{bmatrix}$
Draw the regions $\Gamma_1(h^*)$ and $\Gamma_2(h^*)$ that correspond to the Bayes classifier.

$\Gamma_1(h^*) = \{x : h^*(x) = 1\}$ $\qquad \Gamma_2(h^*) = \{x : h^*(x) = 2\}$

$h^*(x) = \underset{k \in \{1,\ldots,K\}}{\arg\max}\ P(Y=k \mid X=x) = \underset{k \in \{1,\ldots,K\}}{\arg\max} \frac{P(X=x \mid Y=k)P(Y=k)}{P(X=x)} = \underset{k \in \{1,\ldots,K\}}{\arg\max}\ P(X=x \mid Y=k)P(Y=k)$

3. a) The file hw10p3data contains two arrays: X1 and X2. These are samples from an unknown distribution, where X1 has been assigned "class 1", and X2 has been assigned "class 2". Implement the nearest neighbor algorithm, and sketch the decision regions Γ_1 and Γ_2 that it defines.

b) In actuality, the data in the last part was generated using the model from Problem 2. Estimate the generalization error $R(h)$ for both the Bayes classifier (problem 2) and the nearest-neighbor rule (part a), and compare the two. This will require the generation of many Gaussian random vectors with specified covariance matrices.

4. Let $X_1, X_2, \ldots$ be independent Gaussian random variables with mean 0 and variance 1. Let

$$Z_M = \max_{1 \le m \le M} |X_m|.$$

a) Using Monte Carlo simulation, estimate $E[Z_m]$ for $M = 1, 2, 5, 10, 20, 50, 100, \ldots, 10^5, 2 \cdot 10^5, 5 \cdot 10^5, 10^6$. Turn in a plot of $E[Z_M]$ versus M on appropriately scaled (log) axes.

b) It is a fact that

$$\frac{1}{\sqrt{2\pi}} \int_u^\infty \exp(-t^2/2)\, dt \le \frac{1}{2} \exp(-u^2/2),$$

and so $P(|X_m| > u) \le \min(1, e^{-u^2/2})$,

for the $X_m \sim \text{Normal}(0,1)$ as above. Using this and the Boole inequality, find a bound on $P(Z_M > u)$.

Boole Inequality: $P\left(\bigcup_{m=1}^{M} A_m\right) \le \sum_{m=1}^{M} P(A_m)$

For our case A_m corresponds to $|X_m| > u$

$$\Rightarrow P\left(\bigcup_{m=1}^{M} |X_m| > u\right) \le \sum_{m=1}^{M} P(|X_m| > u)$$

$$\le \sum_{m=1}^{M} \min(1, e^{-u^2/2})$$

$$\boxed{P(Z_M > u) \le \min(1, M e^{-u^2/2})}$$

c) It is also a fact that if Z is a positive-valued random variable, then

$$E[Z] = \int_0^\infty P(Z > u)\, du.$$

Use this along with your answer to part (b) to get an analytical upper bound on $E[Z_M]$. Note that if $f(u)$ is a positive monotonically decreasing function, then

$$\int_0^\infty \min(1, f(u))\, du = \gamma + \int_\gamma^\infty f(u)\, du,$$

where γ is the point where $f(\gamma) = 1$. You will find that fact handy along with another application of (1).

$$E[Z_M] = \int_0^\infty P(Z_M > u)\,du \le \int_0^\infty \min(1, Me^{-u^2/2})\,du$$

$$= \sqrt{\ln(M^2)} + \int_{\sqrt{\ln(M^2)}}^\infty Me^{-u^2/2}$$

$$\le \sqrt{\ln(M^2)} + M\frac{\sqrt{2\pi}}{2}\exp\left(-\left(\sqrt{\ln(M^2)}\right)^2/2\right)$$

$$= \boxed{\sqrt{\ln(M^2)} + \frac{\sqrt{2\pi}}{2}}$$

$$Me^{-u^2/2} = 1$$

$$E[Z_M] \le \sqrt{\ln(M^2)} + \frac{\sqrt{2\pi}}{2}$$

$$\frac{1}{\sqrt{2\pi}}\int_u^\infty e^{-t^2/2}\,dt \le \frac{1}{2}e^{-u^2/2}$$

$$\int_u^\infty e^{-t^2/2}\,dt \le \frac{\sqrt{2\pi}}{2}e^{-u^2/2}$$

$$Me^{-u^2/2} = 1$$

$$e^{-u^2/2} = \frac{1}{M}$$

$$\ln(e^{-u^2/2}) = \ln\left(\frac{1}{M}\right)$$

$$-u^2/2 = \ln\left(\frac{1}{M}\right)$$

$$-u^2 = 2\ln\left(\frac{1}{M}\right)$$

$$u^2 = -2\ln\left(\frac{1}{M}\right)$$

$$u^2 = 2\ln(M)$$

$$u^2 = \ln(M^2)$$

$$u = \sqrt{\ln(M^2)}$$

5. Suppose that the coupled random variables $(X,Y) \in \mathbb{R} \times \{0,1\}$ have joint distribution specified by

$$P(Y=0)=0.4, \quad X|Y=0 \sim \text{Normal}(-1,4), \quad X|Y=1 \sim \text{Normal}(1,4).$$

We will consider the following set of classifiers for predicting Y from an observation of X:

$$\mathcal{H} = \{h_\theta(x), \theta \in [-10,10]\}, \quad \text{where } h_\theta(x) = \begin{cases} 0, & x < \theta \\ 1, & x \geq \theta \end{cases}$$

In this case, because we have been told the distribution, we can compute the true risk for every $h_\theta \in \mathcal{H}$:

$$R(h_\theta) = P(Y=1)\int_{-\infty}^{\theta} f_X(x|Y=1)\,dx + P(Y=0)\int_{\theta}^{\infty} f_X(x|Y=0)\,dx$$

a) Write code that generates N (independent) realizations of (X,Y) then plots the empirical risk function $\hat{R}_N(h_\theta)$ overlaid on top of $R(h_\theta)$. Turn in plots of three realizations each for $N=10,100,1000$. These plots should have a horizontal axis indexed by $\theta \in [-10,10]$ (and this interval should be discretized to 1000 points).

b) Using Monte-Carlo simulation, estimate $E[|R(h_\theta)-\hat{R}_N(h_\theta)|]$ for the particular case of $\theta=0.45$ and $N=10,100,1000$. Here, the expectation is with respect to the draw of the data. For a fixed N, a single experiment consists of drawing $x_1,\dots,x_N$, computing $\hat{R}_N(h_{0.45})$, and then $|R(h_{0.45})-\hat{R}_N(h_{0.45})|$ (the quantity $R(h_{0.45})$ is deterministic). Run this experiment many times and average the results to get your estimate. Then repeat for the other values of N.

c) Using Monte-Carlo simulation, estimate

$$E\left[\max_{h_\theta \in \mathcal{H}} |R(h_\theta)-\hat{R}_N(h_\theta)|\right]$$

for $N=10,100,1000$. As above, the expectation is with respect to the random draw of the data $x_1,\dots,x_N$, so your simulation framework should be similar. The main difference is that every experiment produces a random function $\hat{R}_N(h_\theta)$ of θ that is compared against the deterministic function $R(h_\theta)$. You can compute the max by gridding the θ axis at sufficiently many points.

d) Using Monte Carlo simulation, estimate the average performance (generalization error) $E[R(\hat{h}_N)]$ of the empirical risk minimizer

$$\hat{h}_N = \arg\min_{h_\theta \in \mathcal{H}} \hat{R}_N(h_\theta),$$

for $N = 10, 100, 1000$. (You again need simulations as above to generate the $\hat{h}_N$ - given the minimizer, computing $R(\hat{h}_N)$ can be done with (2).) As before, $\hat{h}_N$ is a random classification rule (because of the randomness of the data), and so $R(\hat{h}_N)$ is a random number, even though $R(\cdot)$ is a deterministic function.
Compare your estimate of $E[R(\hat{h}_N)]$ to the risk of the Bayes classifier $R(h_{Bayes})$, whee as usual

$$h_{Bayes} = \arg\min_{h_\theta \in \mathcal{H}} R(h_\theta)$$

6. a) Compute the gradient (with respect to $w \in \mathbb{R}^p$) of
$$-\ell(w; x_n, y_n) = -y_n \log(\sigma(w^T \Psi(x_n))) - (1-y_n)\log(1-\sigma(w^T \Psi(x_n)))$$

$$\frac{\partial(-\ell(w;x_n y_n))}{\partial w} = \frac{\partial(-y_n \log(\sigma(w^T\Psi(x_n))) - (1-y_n)\log(1-\sigma(w^T\Psi(x_n))))}{\partial w}$$

$$= -y_n \frac{\partial(\log(\sigma(w^T\Psi(x_n))))}{\partial w} - (1-y_n)\frac{\partial(\log(1-\sigma(w^T\Psi(x_n))))}{\partial w}$$

$$= -y_n \frac{1}{\sigma(w^T\Psi(x_n))}\frac{\partial(\sigma(w^T\Psi(x_n)))}{\partial w} - (1-y_n)\frac{1}{1-\sigma(w^T\Psi(x_n))}\frac{\partial(1-\sigma(w^T\Psi(x_n)))}{\partial w}$$

$$= -y_n \frac{1}{\sigma(w^T\Psi(x_n))}\frac{\partial(\sigma(w^T\Psi(x_n)))}{\partial w} + (1-y_n)\frac{1}{1-\sigma(w^T\Psi(x_n))}\frac{\partial(\sigma(w^T\Psi(x_n)))}{\partial w}$$

$$\frac{\partial\sigma(w^T\Psi(x_n))}{\partial w} = \sigma(w^T\Psi(x_n))(1-\sigma(w^T\Psi(x_n)))\frac{\partial(w^T\Psi(x_n))}{\partial w} = \sigma(w^T\Psi(x_n))(1-\sigma(w^T\Psi(x_n)))\Psi(x_n)$$

$$= -y_n \cancel{\frac{1}{\sigma(w^T\Psi(x_n))}\sigma(w^T\Psi(x_n))}(1-\sigma(w^T\Psi(x_n)))\Psi(x_n) + (1-y_n)\cancel{\frac{1}{1-\sigma(w^T\Psi(x_n))}}\sigma(w^T\Psi(x_n))\cancel{(1-\sigma(w^T\Psi(x_n)))}\Psi(x_n)$$

$$= (-y_n(1-\sigma(w^T\Psi(x_n))) + (1-y_n)\sigma(w^T\Psi(x_n)))\Psi(x_n)$$
$$= (-y_n + \cancel{y_n\sigma(w^T\Psi(x_n))} + \sigma(w^T\Psi(x_n)) - \cancel{y_n\sigma(w^T\Psi(x_n))})\Psi(x_n)$$
$$\boxed{= (\sigma(w^T\Psi(x_n)) - y_n)\Psi(x_n)}$$

b) The file hw10p6data.mat contains a 2×1000 matrix X and a 1×1000 binary-valued vector Y. Interpret the columns of X as data points $x_n \in \mathbb{R}^2$ and the corresponding entry of Y as a class label $y_n \in \{0,1\}$. Implement gradient descent to fit a conditional probability function to the data. For the function space $\mathcal{F}$, use the space of all polynomials of degree 2, that is

$$\Psi(x) = \begin{bmatrix} x_1^2 \\ x_2^2 \\ x_1 x_2 \\ x_1 \\ x_2 \\ 1 \end{bmatrix}$$

Plot the resulting conditional probability function $p(x)$ and the corresponding classification regions. Turn in these plots along with your code.

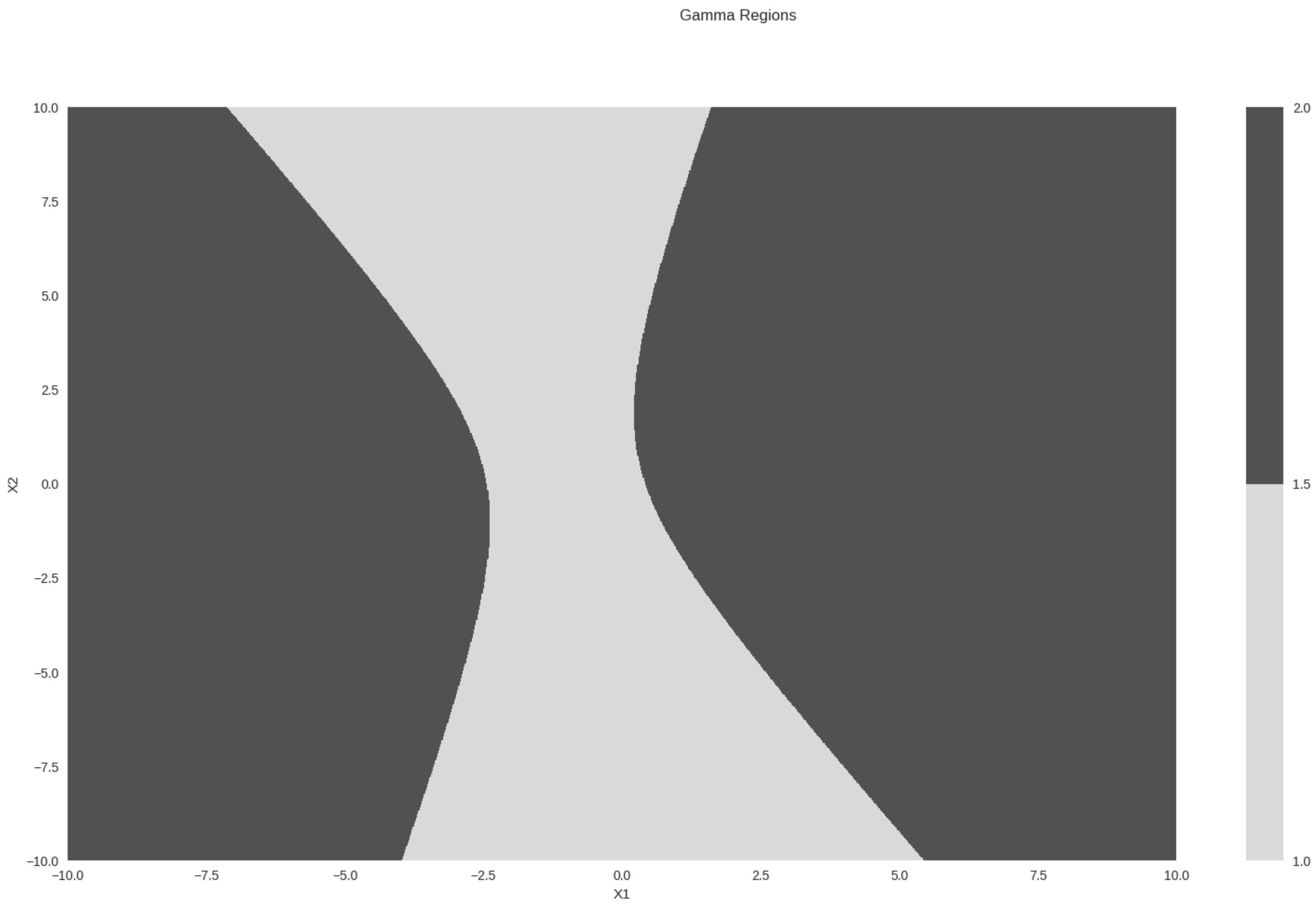
This week in lecture we covered classification. This is different from what the model fitting that we have been doing for regression problems. One way to classify is with the Bayes classifier which just chooses the class which maximizes the conditional probability. Another one that does not require the knowledge of the probability distribution is the nearest neighbor classifier which only uses the collected data to find the class for any given value in the domain.

We also talked about risk. Risk is the average performance for our classifier, computed as the expectation of our loss. Risk minimization occurs by minimizing a data driven (empirical) expectation of the loss with respect to a hypothesis (or our belief of what the classifier should be).

Finally we covered logistic regression which attempts to recover the conditional probability from the data and extrapolate it out for the classification problem. A function is choosen which gets passed through a logistic sigmoid to compute the conditional probability. By minimizing the loss with respect to parameters of our specific function structure, the function parameters, the function, and hence the conditional probability function can be recovered. This probability is then the one used to classifiy the data.

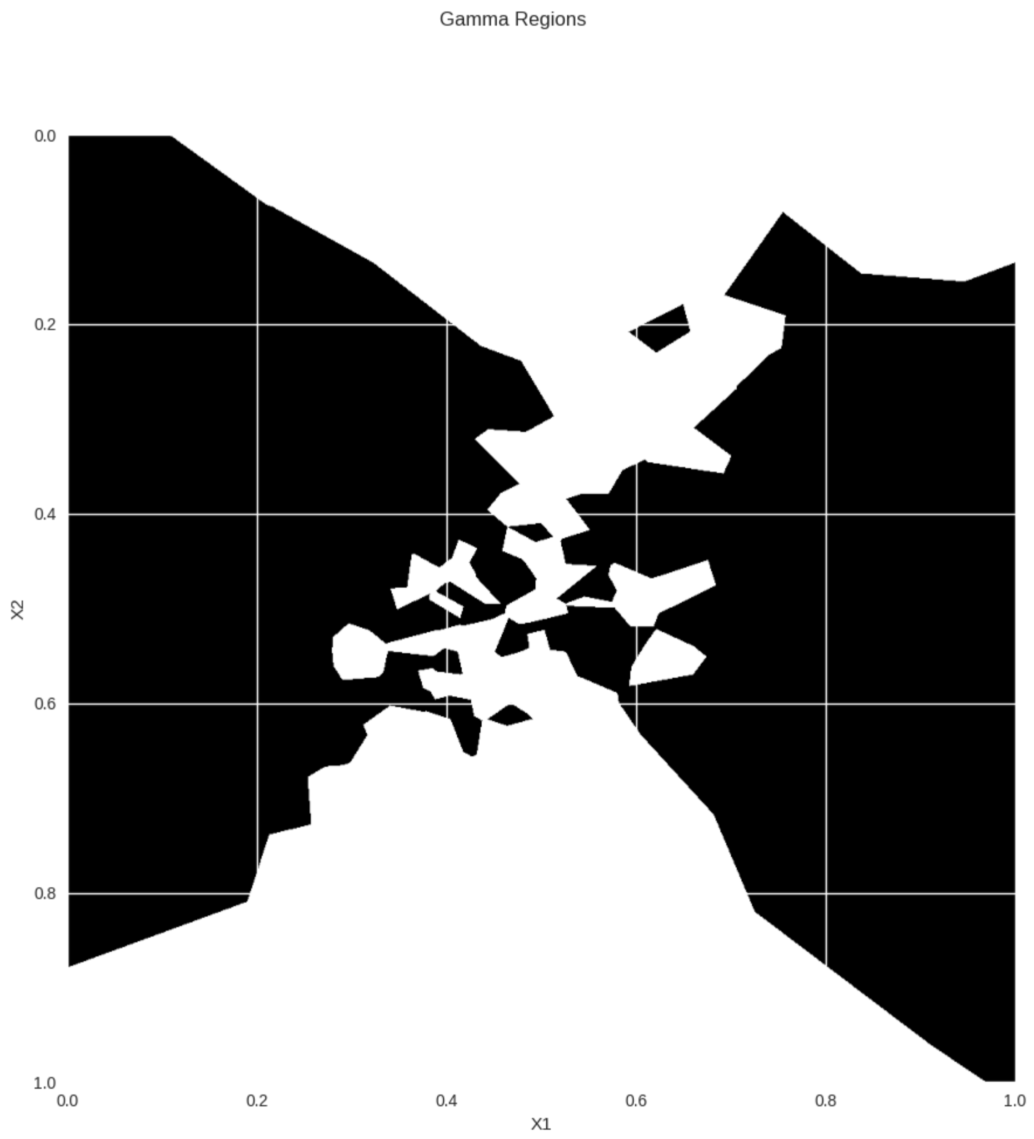
Classification is another problem that machine learning solves very well and is important for a variety of discrete model fitting tasks such as image labelling.

```python
"""Problem 2."""
import matplotlib as mpl
import matplotlib.pyplot as plt

import numpy as np

mpl.style.use('seaborn')


# pylint: disable=too-many-locals
def prob_2():
    """Prob 2."""
    print('Prob 2')

    f_y_1 = 1/2
    f_y_2 = 1/2

    mu1 = np.array([-1, 1])
    mu2 = np.array([1, 0])

    sigma1 = np.array([[3, -6], [-6, 24]])
    sigma2 = np.array([[16, -6], [-6, 8]])

    def gaussian_2d(x_val, mu_val, sigma):
        """Compute 2D Gaussian distribution."""
        return 1/(2 * np.pi * np.sqrt(np.linalg.det(sigma))) * \
            np.exp(-1/2 * np.transpose((x_val - mu_val)) @
                   np.linalg.inv(sigma) @ (x_val - mu_val))

    size = 1000
    x1_vec = np.linspace(-10, 10, size)
    x2_vec = np.linspace(-10, 10, size)

    x1_mat, x2_mat = np.meshgrid(x1_vec, x2_vec)
    gamma_mat = np.zeros_like(x1_mat)

    for x1_index in range(gamma_mat.shape[0]):
        for x2_index in range(gamma_mat.shape[1]):
            x1_val = x1_mat[x1_index, x2_index]
            x2_val = x2_mat[x1_index, x2_index]
            x_val = np.array([x1_val, x2_val])
            f_x_y_1 = gaussian_2d(x_val, mu1, sigma1) * f_y_1
            f_x_y_2 = gaussian_2d(x_val, mu2, sigma2) * f_y_2
            if f_x_y_1 > f_x_y_2:
                gamma_mat[x1_index, x2_index] = 1
            else:
                gamma_mat[x1_index, x2_index] = 2

    fig = plt.figure()
    fig.suptitle('Gamma Regions')
    axes = fig.add_subplot(111)

    csetf = axes.contourf(x1_mat, x2_mat, gamma_mat, levels=1)
    # axes.imshow(gamma_mat, origin='upper', extent=[0, 1, 1, 0])

    fig.colorbar(csetf)
    axes.set_xlabel('X1')
    axes.set_ylabel('X2')

    plt.show()


prob_2()
```

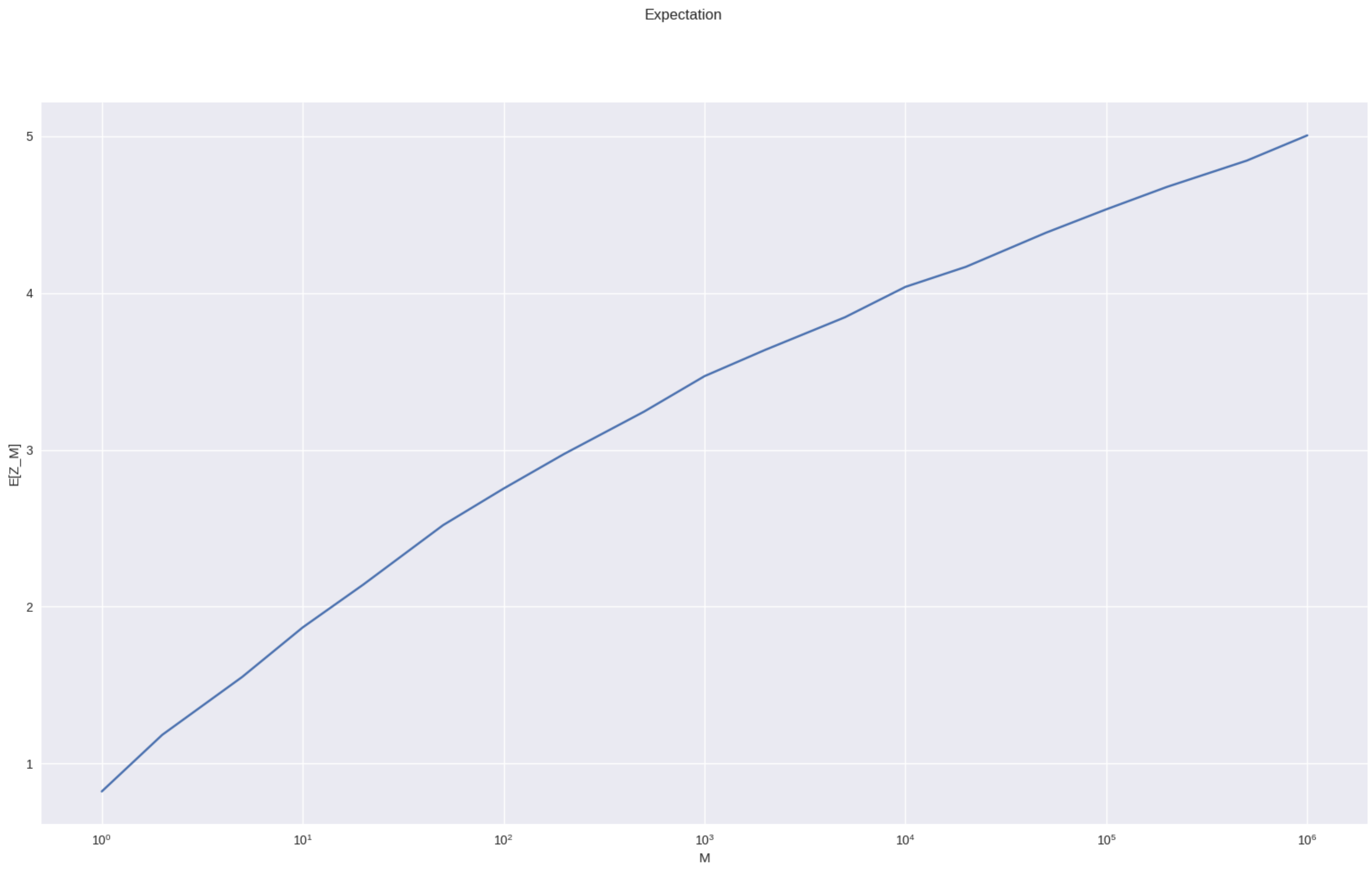

Gamma Regions
X2
X1
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
2.0
1.5
1.0

```python
"""Problem 3."""
import matplotlib as mpl
import matplotlib.pyplot as plt

import numpy as np

import scipy.io as sio

mpl.style.use('seaborn')

MAT_FILENAME = 'hw10p3data.mat'
data_samples = sio.loadmat(MAT_FILENAME)

X1_data = data_samples['X1']
X2_data = data_samples['X2']


# pylint: disable=too-many-locals
def part_a(x1_data, x2_data):
    """Part a."""
    print('Part a')

    min_dim_1 = min(min(x1_data[0]), min(x2_data[0]))
    max_dim_1 = max(max(x1_data[0]), max(x2_data[0]))

    min_dim_2 = min(min(x1_data[1]), min(x2_data[1]))
    max_dim_2 = max(max(x1_data[1]), max(x2_data[1]))

    x1_vec = np.linspace(min_dim_1, max_dim_1, 1000)
    x2_vec = np.linspace(min_dim_2, max_dim_2, 1000)

    x1_mat, x2_mat = np.meshgrid(x1_vec, x2_vec)
    gamma_mat = np.zeros_like(x1_mat)

    for x1_index in range(gamma_mat.shape[0]):
        for x2_index in range(gamma_mat.shape[1]):
            x1_val = x1_mat[x1_index, x2_index]
            x2_val = x2_mat[x1_index, x2_index]
            x_val = np.array([x1_val, x2_val])
            min_x1_data_distance = np.inf
            min_x2_data_distance = np.inf
            for x1_data_index in range(x1_data.shape[1]):
                x1_data_val = x1_data[:, x1_data_index]
                x1_data_distance = np.linalg.norm(x1_data_val - x_val)
                if x1_data_distance < min_x1_data_distance:
                    min_x1_data_distance = x1_data_distance
            for x2_data_index in range(x2_data.shape[1]):
                x2_data_val = x2_data[:, x2_data_index]
                x2_data_distance = np.linalg.norm(x2_data_val - x_val)
                if x2_data_distance < min_x2_data_distance:
                    min_x2_data_distance = x2_data_distance
            if min_x1_data_distance < min_x2_data_distance:
                gamma_mat[x1_index, x2_index] = 1
            else:
                gamma_mat[x1_index, x2_index] = 2

    fig = plt.figure()
    fig.suptitle('Gamma Regions')
    axes = fig.add_subplot(111)

    # csetf = axes.contourf(x1_mat, x2_mat, gamma_mat, levels=1)
    axes.imshow(gamma_mat, origin='upper', extent=[0, 1, 1, 0])

    # fig.colorbar(csetf)
```

```
    axes.set_xlabel('X1')
    axes.set_ylabel('X2')

    plt.show()


part_a(X1_data, X2_data)


def part_b():
    """Part b."""
    print('Part b')


part_b()
```


Gamma Regions
X2
0.0
0.2
0.4
0.6
0.8
1.0
X1
0.0
0.2
0.4
0.6
0.8
1.0

```python
"""Problem 4."""
import itertools

import matplotlib as mpl
import matplotlib.pyplot as plt

import numpy as np

mpl.style.use('seaborn')


def compute_expectation(size):
    """Compute expectation."""
    num_simulations = 500
    xm_rv = np.random.randn(size, num_simulations)
    zm_rv = np.max(np.abs(xm_rv), axis=0)
    expectation = np.mean(zm_rv)
    return expectation


def part_a():
    """Part a."""
    print('Part a')

    size_list = [[1 * 10**i, 2 * 10**i, 5 * 10**i] for i in range(0, 6)]
    # flatten list
    size_list = list(itertools.chain(*size_list))
    size_list.append(1 * 10**6)

    expectation_list = [compute_expectation(size) for size in size_list]

    fig = plt.figure()
    fig.suptitle('Expectation')
    axes = fig.add_subplot(111)

    axes.semilogx(size_list, expectation_list)

    axes.set_xlabel('M')
    axes.set_ylabel('E[Z_M]')

    plt.show()


part_a()
```


Expectation
E[Z_M]
5
4
3
2
1
10^0
10^1
10^2
10^3
10^4
10^5
10^6
M

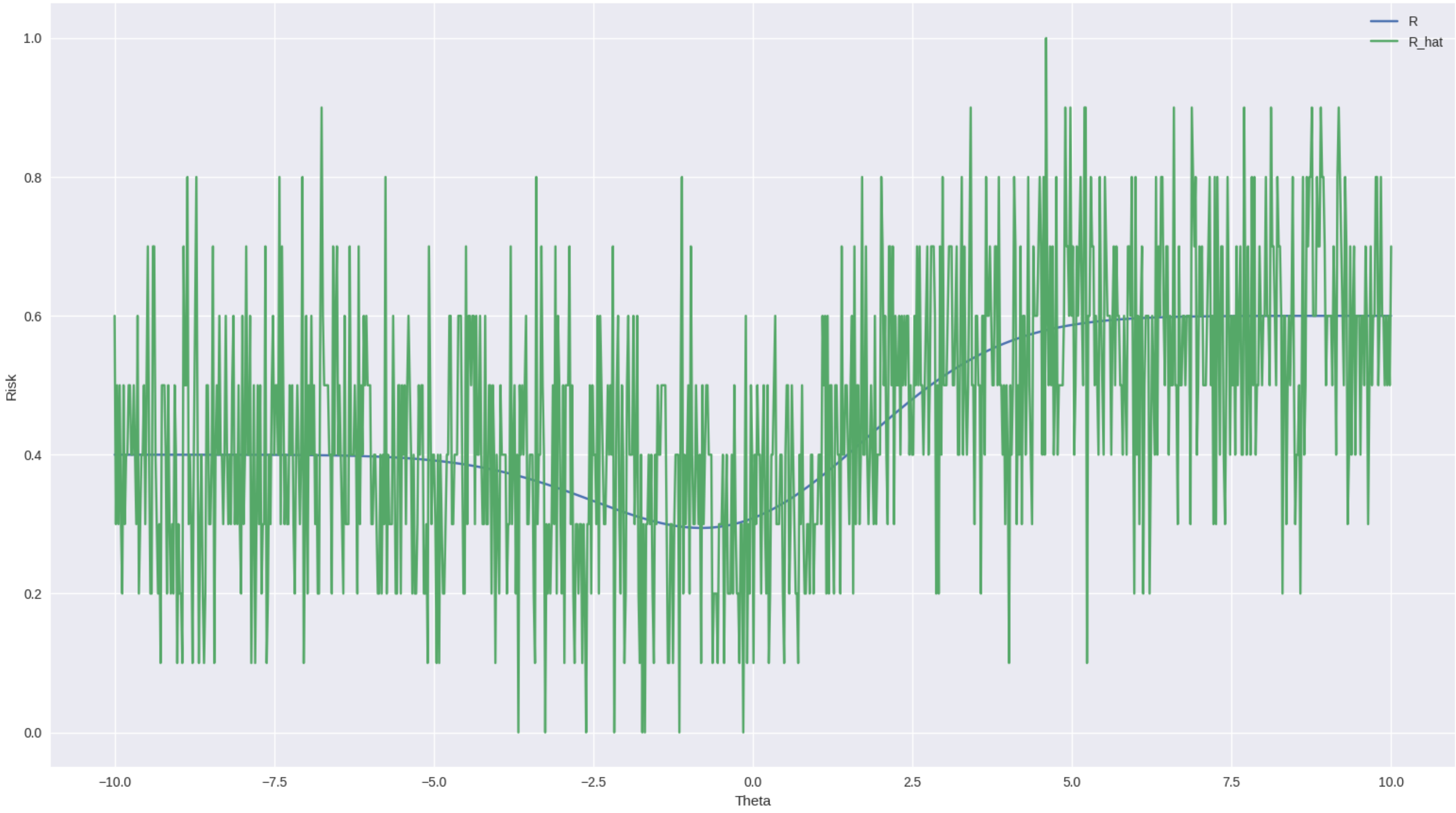
```python
"""Problem 5."""
import operator

import matplotlib as mpl
import matplotlib.pyplot as plt

import numpy as np

from scipy import stats as sst

mpl.style.use('seaborn')

F_Y_0 = 0.4
MU_X_Y_0 = -1
VAR_X_Y_0 = 4
MU_X_Y_1 = 1
VAR_X_Y_1 = 4


# pylint: disable=too-many-arguments
def compute_risk(theta, f_y_0, mu_x_y_0, var_x_y_0, mu_x_y_1, var_x_y_1):
    """Compute risk."""
    f_y_1 = 1 - f_y_0
    std_x_y_0 = np.sqrt(var_x_y_0)
    std_x_y_1 = np.sqrt(var_x_y_1)
    risk = f_y_1 * sst.norm.cdf(theta, mu_x_y_1, std_x_y_1) \
        + f_y_0 * (1 - sst.norm.cdf(theta, mu_x_y_0, std_x_y_0))

    return risk


# pylint: disable=too-many-locals
def compute_empirical_risk(theta, realization_num, f_y_0, mu_x_y_0, var_x_y_0,
                           mu_x_y_1, var_x_y_1):
    """Compute empirical risk."""
    std_x_y_0 = np.sqrt(var_x_y_0)
    std_x_y_1 = np.sqrt(var_x_y_1)

    uniform_realizations = np.random.rand(realization_num)

    y_realizations = np.zeros_like(uniform_realizations)
    y_realizations[uniform_realizations > f_y_0] = 1

    x_realizations = np.zeros_like(y_realizations)
    for idx, y_realization in enumerate(y_realizations):
        if y_realization == 0:
            x_realizations[idx] = np.random.normal(mu_x_y_0, std_x_y_0)
        else:
            x_realizations[idx] = np.random.normal(mu_x_y_1, std_x_y_1)

    def h_theta(x_realizations, theta):
        h_vec = np.zeros_like(x_realizations)
        h_vec[x_realizations >= theta] = 1
        return h_vec

    loss = (h_theta(x_realizations, theta) - y_realizations) ** 2
    empirical_risk = np.mean(loss)
    return empirical_risk


def part_a(f_y_0, mu_x_y_0, var_x_y_0, mu_x_y_1, var_x_y_1):
    """Part a."""
    print('Part a')
```

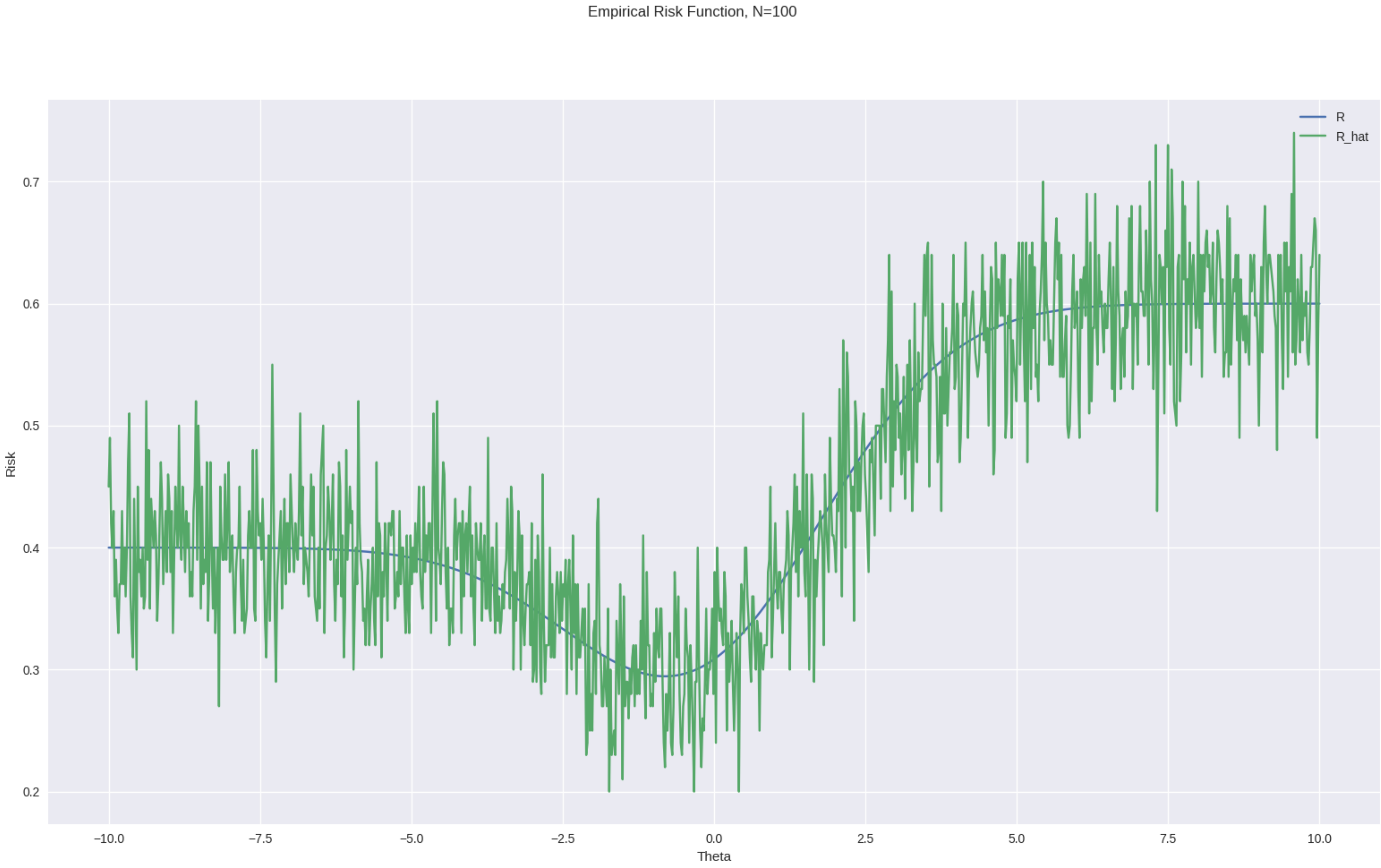
```
    realization_num_list = [10, 100, 1000]
    theta_list = np.linspace(-10, 10, 1000).tolist()

    risk_list = [compute_risk(theta, f_y_0, mu_x_y_0, var_x_y_0, mu_x_y_1,
                              var_x_y_1)
                 for theta in theta_list]

    for realization_num in realization_num_list:
        fig = plt.figure()
        fig.suptitle('Empirical Risk Function, N=' + str(realization_num))
        axes = fig.add_subplot(111)

        empirical_risk_list = [compute_empirical_risk(theta, realization_num,
                                                      f_y_0, mu_x_y_0,
                                                      var_x_y_0, mu_x_y_1,
                                                      var_x_y_1)
                               for theta in theta_list]

        axes.plot(theta_list, risk_list, label='R')
        axes.plot(theta_list, empirical_risk_list, label='R_hat')

        axes.set_xlabel('Theta')
        axes.set_ylabel('Risk')
        axes.legend()

        plt.show()


part_a(F_Y_0, MU_X_Y_0, VAR_X_Y_0, MU_X_Y_1, VAR_X_Y_1)


def part_b(f_y_0, mu_x_y_0, var_x_y_0, mu_x_y_1, var_x_y_1):
    """Part b."""
    print('Part b')

    theta = 0.45
    realization_num_list = [10, 100, 1000]
    num_simulations = 1000

    risk = compute_risk(theta, f_y_0, mu_x_y_0, var_x_y_0, mu_x_y_1,
                        var_x_y_1)

    for realization_num in realization_num_list:

        empirical_risk_list = [compute_empirical_risk(theta, realization_num,
                                                      f_y_0, mu_x_y_0,
                                                      var_x_y_0, mu_x_y_1,
                                                      var_x_y_1)
                               for item in range(num_simulations)]

        risk_error_list = np.abs(risk - empirical_risk_list)
        risk_error_expectation = np.mean(risk_error_list)

        print('Risk Error Expectation, N=' + str(realization_num) + ' : ' +
              str(risk_error_expectation))


part_b(F_Y_0, MU_X_Y_0, VAR_X_Y_0, MU_X_Y_1, VAR_X_Y_1)


def part_c(f_y_0, mu_x_y_0, var_x_y_0, mu_x_y_1, var_x_y_1):
    """Part c."""
    print('Part c')
```

```
    realization_num_list = [10, 100, 1000]
    num_simulations = 100

    theta_list = np.linspace(-10, 10, 100).tolist()
    risk_list = [compute_risk(theta, f_y_0, mu_x_y_0, var_x_y_0, mu_x_y_1,
                              var_x_y_1)
                 for theta in theta_list]
    risk_vec = np.array(risk_list)

    for realization_num in realization_num_list:

        empirical_risk_list_list = [[compute_empirical_risk(theta,
                                                            realization_num,
                                                            f_y_0, mu_x_y_0,
                                                            var_x_y_0,
                                                            mu_x_y_1,
                                                            var_x_y_1)
                                     for theta in theta_list]
                                    for item in range(num_simulations)]

        empirical_risk_mat = np.array(empirical_risk_list_list)

        risk_error_vec = np.abs(risk_vec - empirical_risk_mat)
        max_risk_error_vec = np.max(risk_error_vec, axis=1)
        max_risk_error_expectation = np.mean(max_risk_error_vec)

        print('Max Risk Error Expectation, N=' + str(realization_num) + ' : ' +
              str(max_risk_error_expectation))


part_c(F_Y_0, MU_X_Y_0, VAR_X_Y_0, MU_X_Y_1, VAR_X_Y_1)


def part_d(f_y_0, mu_x_y_0, var_x_y_0, mu_x_y_1, var_x_y_1):
    """Part d."""
    print('Part d')

    realization_num_list = [10, 100, 1000]
    num_simulations = 100

    theta_list = np.linspace(-10, 10, 100).tolist()
    risk_list = [compute_risk(theta, f_y_0, mu_x_y_0, var_x_y_0, mu_x_y_1,
                              var_x_y_1)
                 for theta in theta_list]
    risk_vec = np.array(risk_list)

    for realization_num in realization_num_list:

        empirical_risk_list_list = [[compute_empirical_risk(theta,
                                                            realization_num,
                                                            f_y_0, mu_x_y_0,
                                                            var_x_y_0,
                                                            mu_x_y_1,
                                                            var_x_y_1)
                                     for theta in theta_list]
                                    for item in range(num_simulations)]

        empirical_risk_mat = np.array(empirical_risk_list_list)

        empirical_risk_argmin_vec = np.argmin(empirical_risk_mat, axis=1)
        theta_min_list = [theta_list[theta_min_idx]
                          for theta_min_idx in empirical_risk_argmin_vec]

        risk_performance_list = [compute_risk(theta_min, f_y_0, mu_x_y_0,
```

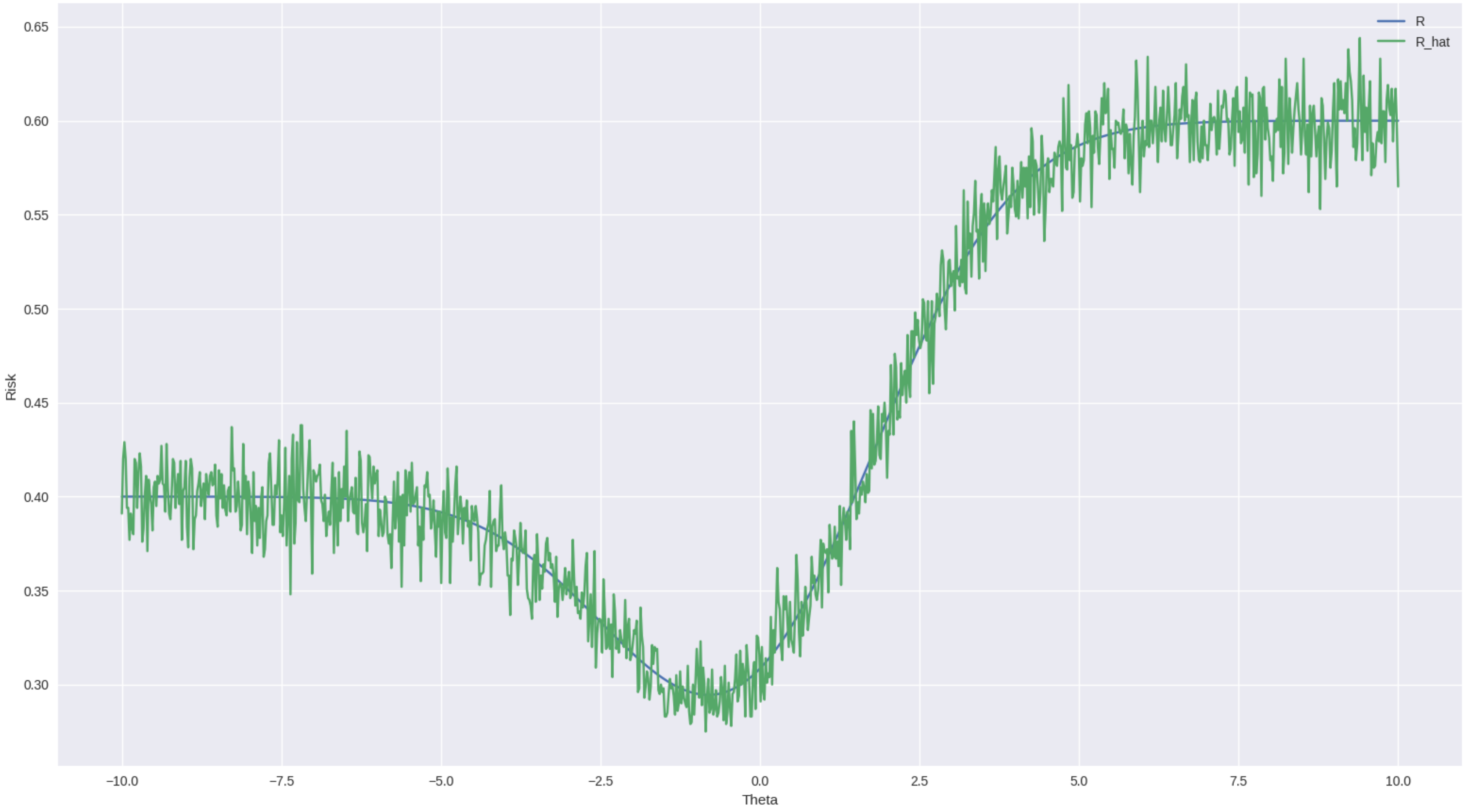
```
                                              var_x_y_0, mu_x_y_1, var_x_y_1)
                                 for theta_min in theta_min_list]

        generalization_error = np.mean(risk_performance_list)
        print('Generialization Error, N=' + str(realization_num) + ' : ' +
              str(generalization_error))

    bayes_risk = np.min(risk_vec)
    print('Bayes Risk : ' + str(bayes_risk))


part_d(F_Y_0, MU_X_Y_0, VAR_X_Y_0, MU_X_Y_1, VAR_X_Y_1)
```

Empirical Risk Function, N=10




Empirical Risk Function, N=100
R
R_hat
Risk
Theta
0.2
0.3
0.4
0.5
0.6
0.7
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

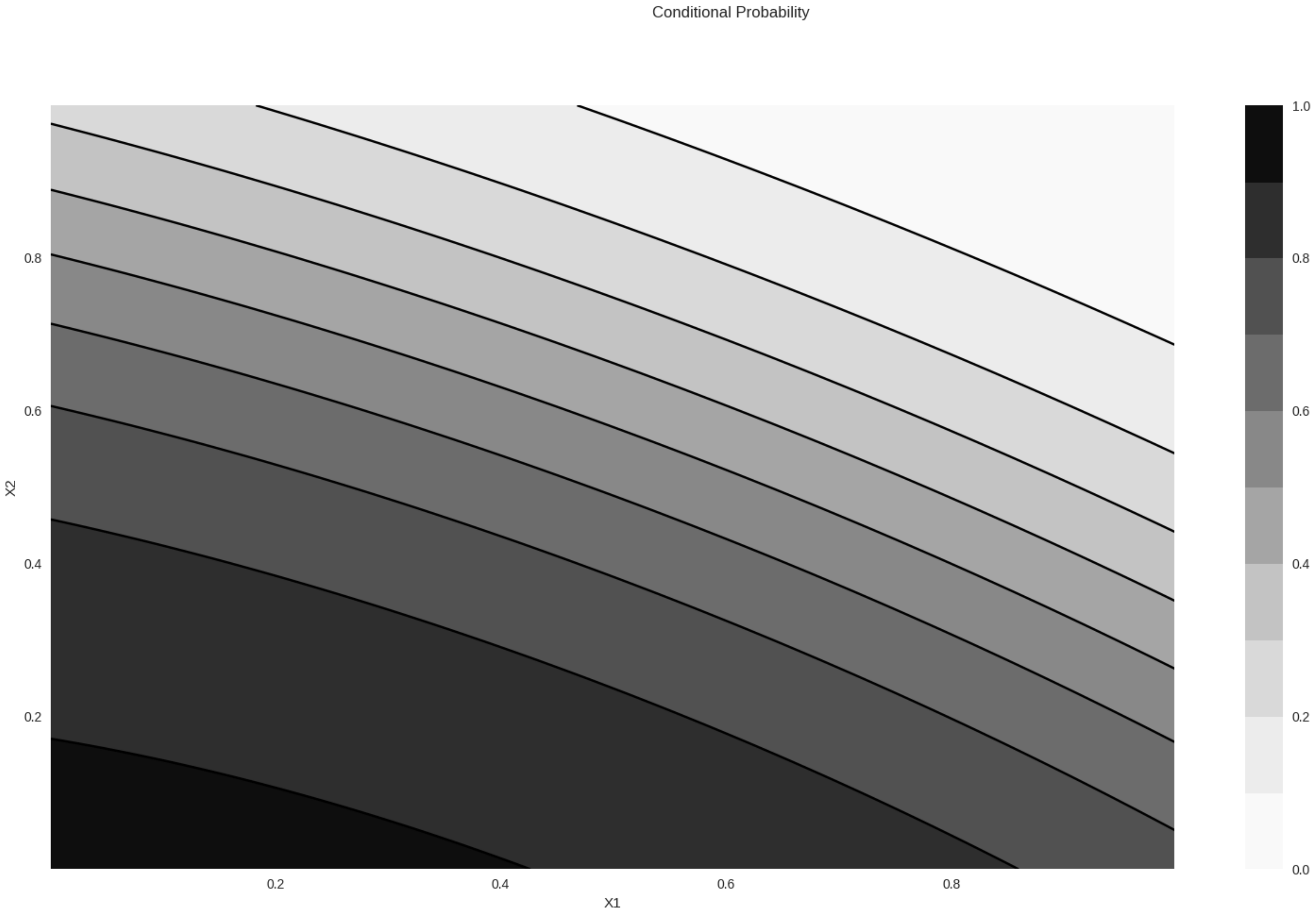
Empirical Risk Function, N=1000

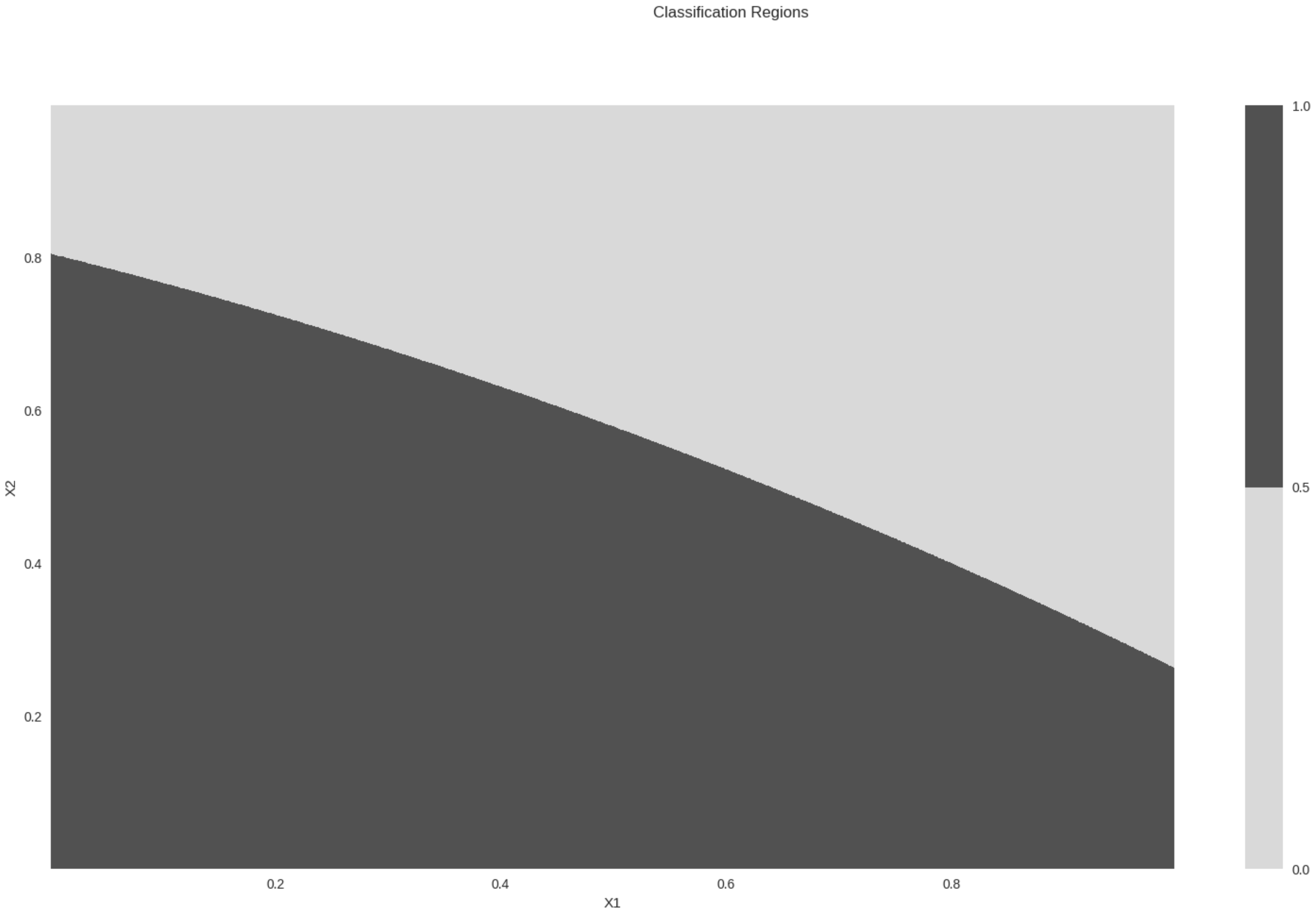


```
Part a
Part b
Risk Error Expectation, N=10 : 0.11988326886803308
Risk Error Expectation, N=100 : 0.03585461946650348
Risk Error Expectation, N=1000 : 0.01202029444845518
Part c
Max Risk Error Expectation, N=10 : 0.40976314321600227
Max Risk Error Expectation, N=100 : 0.1357988272030948
Max Risk Error Expectation, N=1000 : 0.04217571003299191
Part d
Generialization Error, N=10 : 0.36983408130167544
Generialization Error, N=100 : 0.30848712418967816
Generialization Error, N=1000 : 0.2983588735032951
Bayes Risk : 0.29469128466453054
```

```python
"""Problem 6."""
import matplotlib as mpl
import matplotlib.pyplot as plt

import numpy as np

import scipy.io as sio
import scipy.special as ssp

mpl.style.use('seaborn')

MAT_FILENAME = 'hw10p6data.mat'
data_samples = sio.loadmat(MAT_FILENAME)

X_data = data_samples['X']
Y_data = data_samples['Y']


def phi(xn_data):
    """Compute phi."""
    return np.array([xn_data[0]**2, xn_data[1]**2, xn_data[0]*xn_data[1],
                     xn_data[0], xn_data[1], 1])


def compute_loss_grad(weights, x_data, y_data):
    """Compute derivative of negative cross entropy loss."""
    loss_grad = 0
    for idx, _ in enumerate(x_data):
        xn_data = x_data[:, idx]
        yn_data = y_data[0, idx]
        phi_xn = phi(xn_data)

        loss_grad += (ssp.expit(weights @ phi_xn) - yn_data) * phi_xn

    return loss_grad


def compute_cond_prob_func(x_data, y_data):
    """Compute the probability function, parameterized by weights."""
    weights = np.array([0, 0, 0, 0, 0, 0])

    weight_norm_delta_tol = 1/1000
    step_size = 0.01

    norm_diff = np.inf
    while norm_diff > weight_norm_delta_tol:
        prev_norm = np.linalg.norm(weights)

        weights = weights - step_size \
            * compute_loss_grad(weights, x_data, y_data)

        curr_norm = np.linalg.norm(weights)
        norm_diff = np.abs(curr_norm - prev_norm)
    return weights


def compute_cond_prob_mat(weights, x1_mat, x2_mat):
    """Compute conditional probability."""
    cond_prob_mat = np.zeros_like(x1_mat)
    for x1_index in range(cond_prob_mat.shape[0]):
        for x2_index in range(cond_prob_mat.shape[1]):
            x1_val = x1_mat[x1_index, x2_index]
            x2_val = x2_mat[x1_index, x2_index]
            x_val = np.array([x1_val, x2_val])
```

```
            cond_prob_mat[x1_index, x2_index] = ssp.expit(weights @ phi(x_val))

    return cond_prob_mat


# pylint: disable=too-many-locals
def part_b(x_data, y_data):
    """Part b."""
    print('Part b')
    y_data = y_data.astype(float)

    min_dim_1 = min(x_data[0])
    max_dim_1 = max(x_data[0])

    min_dim_2 = min(x_data[1])
    max_dim_2 = max(x_data[1])

    size = 1000
    x1_vec = np.linspace(min_dim_1, max_dim_1, size)
    x2_vec = np.linspace(min_dim_2, max_dim_2, size)

    x1_mat, x2_mat = np.meshgrid(x1_vec, x2_vec)

    weights = compute_cond_prob_func(x_data, y_data)

    cond_prob_mat = compute_cond_prob_mat(weights, x1_mat, x2_mat)

    # Plot conditional probability function p(x)
    fig = plt.figure()
    fig.suptitle('Conditional Probability')
    axes = fig.add_subplot(111)

    csetf = axes.contourf(x1_mat, x2_mat, cond_prob_mat, levels=10)
    axes.contour(x1_mat, x2_mat, cond_prob_mat, csetf.levels, colors='k')

    fig.colorbar(csetf, ax=axes)
    axes.set_xlabel('X1')
    axes.set_ylabel('X2')

    plt.show()

    # Compute classification regions
    # > 50%, y = 1; <= 50%, y = 0
    class_mat = np.zeros_like(x1_mat)
    one_mask = cond_prob_mat > 0.5
    class_mat[one_mask] = 1

    # Plot classification regions
    fig = plt.figure()
    fig.suptitle('Classification Regions')
    axes = fig.add_subplot(111)

    csetf = axes.contourf(x1_mat, x2_mat, class_mat, levels=1)

    fig.colorbar(csetf, ax=axes)
    axes.set_xlabel('X1')
    axes.set_ylabel('X2')

    plt.show()


part_b(X_data, Y_data)
```


Conditional Probability
X1
X2
0.2
0.4
0.6
0.8
0.0
1.0



Classification Regions
0.8
0.6
0.4
0.2
X2
0.2
0.4
0.6
0.8
X1
1.0
0.5
0.0