CS 189 Introduction to Machine Learning
Fall 2017 HW4

This homework is due **Friday, September 22 at 10pm.**

1 Getting Started

You may typeset your homework in latex or submit neatly handwritten and scanned solutions. Please make sure to start each question on a new page, as grading (with Gradescope) is much easier that way! Deliverables:

1. Submit a PDF of your writeup to assignment on Gradescope, "HW[n] Write-Up"
2. Submit your test set evaluation results, "HW[n] Test Set".

After you've submitted your homework, be sure to watch out for the self-grade form.

(a) Before you start your homework, write down your team. Who else did you work with on this homework? List names and email addresses. In case of course events, just describe the group. How did you work on this homework? Any comments about the homework?

None
Comments: lecture 6 notes should not be released too late

(b) Please copy the following statement and sign next to it:

I certify that all solutions are entirely in my words and that I have not looked at another student's solutions. I have credited all external sources in this write up.

I certify that all solutions are entirely in my words & that I have not looked at another student's solutions. I have credited all external sources in my write up

Problem # 2

a) X has multivariate Gaussian distribution with mean $\mu \in R^d$ and covariance matrix $\Sigma \in R^{d \times d}$

$$p(x \mid \mu, \Sigma) = \frac{1}{(2\pi)^{d/2} \|\Sigma\|^{1/2}} e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)}$$

Drawing n i.i.d sample $x_1, x_2, \ldots, x_n$

$$\prod_1^n p(x_i \mid \mu, \Sigma) = \prod_1^n \left(\frac{1}{(2\pi)^{d/2} \|\Sigma\|^{1/2}} \right)^n e^{-\frac{1}{2}(x_i-\mu)^T \Sigma^{-1} (x_i-\mu)}$$

log likelihood:

$$-n \log\left[(2\pi)^{d/2} \|\Sigma\|^{1/2}\right] - \frac{1}{2} \sum_1^n (x_i-\mu)^T \Sigma^{-1} (x_i-\mu)$$

b.) Take derivative wrt μ:

$$\frac{\partial(\,)}{\partial \mu} = -0 - \frac{\partial}{\partial \mu}\left[\frac{1}{2} \sum_1^n (\mu - x_i)^T \Sigma^{-1} (\mu - x_i)\right]$$

$\frac{\partial x^T A x}{\partial x} = x^T(A + A^T)$

$= 2x^T A$ if A sym

$\frac{\partial u^T v}{\partial x} = u^T \frac{\partial v}{\partial x} + v^T \frac{\partial u}{\partial x}$

$\frac{\partial Ax}{\partial x} = A \quad \frac{\partial x^T A}{\partial x} = A^T$

$$= -\sum_1^n (\mu - x_i)^T \Sigma^{-1}$$

$$= \left(n\mu - \sum_1^n x_i\right)^T \Sigma^{-1} = 0$$

$$\Rightarrow \quad \mu = \frac{1}{n} \sum_1^n x_i$$

Take derivative w.r.t Σ^{-1}

$$\frac{\partial}{\partial \Sigma^{-1}} \left[\frac{n}{2} \log\left((2\pi)^{-d/2} \|\Sigma\|^{-1}\right) - \frac{1}{2} \sum_1^n (x_i-\mu)^T \Sigma^{-1} (x_i-\mu) \right]$$

$\frac{\partial \ln \|X\|}{\partial X} = X^{-1}$

$$= \frac{n}{2} (\Sigma^{-1})^{-1} - \frac{1}{2} \sum_1^n (x_i-\mu)^T (x_i-\mu) = 0 \qquad \left(\frac{\partial a^T X b}{\partial X} = ab^T\right.$$

$$\Rightarrow \quad \Sigma = \frac{1}{n} \sum_1^n (x_i-\mu)^T (x_i-\mu) \qquad \left.\frac{\partial a^T X^T b}{\partial X} = ba^T\right)$$

(c) See appendix

Problem # 3

(a) $z \sim N(0,1) \Rightarrow P(z) = \frac{1}{\sqrt{2\pi}} e^{-\frac{z^2}{2}}$

$Y = w^T X + z \Rightarrow Y \sim N(w^T X, 1)$ given X & w

$\Rightarrow P(Y|X,w) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(Y - w^T X)^2}{2}}$

(b)

$$P(w|Y,X) = \frac{P(Y|w,X)\, P(w|X)}{\int_Y P(Y|w,X)\, P(w|X)}$$

$P(w) = \frac{1}{(2\pi)^{d/2} \|\Sigma\|^{1/2}} e^{-\frac{1}{2} w^T \Sigma^{-1} w}$

$P(w|X) = P(w)$

$= \text{const} \times e^{-\frac{(Y - w^T X)^2}{2} - \frac{1}{2} w^T \Sigma^{-1} w}$

$= \text{const} \times e^{-\frac{Y^2 - 2 Y w^T X + w^T X X^T w}{2} - \frac{1}{2} w^T \Sigma^{-1} w}$

$= \text{const} \times e^{-\frac{1}{2} w^T (\Sigma^{-1} + X X^T) w + Y w^T X}$

Given n training data points $(X_1, Y_1), \ldots, (X_n, Y_n)$

$P(w | Y_1, Y_2, \ldots, Y_n ; X_1, X_2, \ldots, X_n)$

$= P(w|Y_1, X_1)\, P(w|Y_2, X_2) \ldots P(w|Y_n, X_n)$

$= \text{const} \cdot e^{-\frac{1}{2} w^T (n\Sigma^{-1} + \sum_1^n X_i X_i^T) w + \sum_1^n Y_i X_i^T w}$

The mean of w is : $(n\Sigma^{-1} + \sum_1^n X_i X_i^T)^{-1} \sum_1^n X_i Y_i$

(c) BONUS : skip

(d) see appendix

Observation : when we have more training data point, the area of contours shrink

→ prediction more precise

(e) see appendix

Problem #4

(a) $\text{rank}(A + \hat{A}) = d$

We know that we have to find $\hat{A}$ and $\hat{\vec{y}}$ such that:

$$[A + \hat{A}, \vec{y} + \hat{\vec{y}}] \begin{bmatrix} \vec{w} \\ -1 \end{bmatrix} = 0$$

$$(A + \hat{A})\vec{w} - (\vec{y} + \hat{\vec{y}}) = 0$$

$$\Rightarrow \quad \vec{y} + \hat{\vec{y}} = (A + \hat{A})\vec{w}$$

The column $\vec{y} + \hat{\vec{y}}$ is linearly dependent on the columns of $A + \hat{A}$ $\Rightarrow$ adding column $\vec{y} + \hat{\vec{y}}$ to $(A + \hat{A})$ does not change the rank of $A + \hat{A}$, i.e

$$\text{rank}([A + \hat{A}, \vec{y} + \hat{\vec{y}}]) = \text{rank}(A + \hat{A}) = d$$

(6)

$$[A, \vec{y}] = U \Sigma V^T = \begin{bmatrix} U_{xx} & U_{xy} \\ U_{yx} & U_{yy} \end{bmatrix} \begin{bmatrix} \Sigma_{1,\dots,d} & \\ & \sigma \end{bmatrix} \begin{bmatrix} V_{xx} & V_{xy} \\ V_{yx} & V_{yy} \end{bmatrix}^T$$

$$[A + \hat{A}, \vec{y} + \hat{\vec{y}}] = \begin{bmatrix} U_{xx} & U_{xy} \\ U_{yx} & U_{yy} \end{bmatrix} \begin{bmatrix} \Sigma_{1,\dots,d} & \\ & 0 \end{bmatrix} \begin{bmatrix} V_{xx} & V_{xy} \\ V_{yx} & V_{yy} \end{bmatrix}^T$$

$$\Rightarrow [\hat{A}, \hat{\vec{y}}] = \begin{bmatrix} U_{xx} & U_{xy} \\ U_{yx} & U_{yy} \end{bmatrix} \begin{bmatrix} 0 & \\ & -\delta \end{bmatrix} \begin{bmatrix} V_{xx} & V_{xy} \\ V_{yx} & V_{yy} \end{bmatrix}^T$$

We try to write $[\hat{A}, \hat{\vec{y}}]$ in term of $[A, \vec{y}]$

$$[\hat{A}, \hat{\vec{y}}] = [A, \vec{y}] \times ?$$

$$? = [A, \vec{y}]^{-1} [\hat{A}, \hat{\vec{y}}]$$

$$= V^{-T} \Sigma^{-1} \cancel{U^{-1}} \overset{I}{\cancel{U}} \begin{bmatrix} 0 & \\ & -\delta \end{bmatrix} V^T$$

$$= \cancel{V^{-T}} \begin{bmatrix} \Sigma^{-1}_{1,\dots,d} & \\ & \sigma^{-1} \end{bmatrix} \begin{bmatrix} 0 & \\ & -\sigma \end{bmatrix} V^T$$

$$= \begin{bmatrix} V_{xx} & V_{xy} \\ V_{yx} & V_{yy} \end{bmatrix} \underbrace{\begin{bmatrix} 0 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} V_{xx}^T & V_{yx}^T \\ V_{xy}^T & V_{yy}^T \end{bmatrix}}_{\begin{bmatrix} 0 & 0 \\ -V_{xy}^T & -V_{yy}^T \end{bmatrix}} = -\begin{bmatrix} V_{xy} V_{xy}^T & V_{xy} V_{yy}^T \\ V_{yy} V_{xy}^T & V_{yy} V_{yy}^T \end{bmatrix}$$

p.T

We have

$$[\hat{A}, \hat{\vec{y}}] = -[A, \vec{y}] \begin{bmatrix} V_{xy} V_{xy}^T & V_{xy} V_{yy}^T \\ V_{yy} V_{xy}^T & V_{yy} V_{yy}^T \end{bmatrix}$$

we multiply both sides with $\begin{bmatrix} V_{xy} \\ V_{yy} \end{bmatrix}$

$$[\hat{A}, \hat{\vec{y}}] \begin{bmatrix} V_{xy} \\ V_{yy} \end{bmatrix} = -[A, \vec{y}] \begin{bmatrix} V_{xy} V_{xy}^T V_{xy} + V_{xy} V_{yy}^T V_{yy} \\ V_{yy} V_{xy}^T V_{xy} + V_{yy} V_{yy}^T V_{yy} \end{bmatrix}$$

we know that $V^T V = I \Rightarrow V_{xy}^T V_{xy} + V_{yy}^T V_{yy} = I$

$$\Rightarrow [\hat{A}, \hat{\vec{y}}] \begin{bmatrix} V_{xy} \\ V_{yy} \end{bmatrix} = -[A, \vec{y}] \begin{bmatrix} V_{xy} \\ V_{yy} \end{bmatrix}$$

$$\Rightarrow [A \perp \hat{A}, \hat{\vec{y}} + \vec{y}] \underbrace{\begin{bmatrix} V_{xy} \\ V_{yy} \end{bmatrix}}_{\vec{x}} = 0$$

solution $\vec{x} = \begin{bmatrix} V_{xy} \\ V_{yy} \end{bmatrix}$

Multiply $\vec{x}$ with $(-V_{yy}^{-1})$ to normalize $\vec{x}$ in the form $\begin{bmatrix} \vec{w} \\ -1 \end{bmatrix}$

$$\begin{bmatrix} -V_{xy} V_{yy}^{-1} \\ -1 \end{bmatrix} = \begin{bmatrix} \vec{w} \\ -1 \end{bmatrix}$$

$$\Rightarrow \vec{w} = -V_{xy} V_{yy}^{-1}$$

(c) See appendix

(d) See appendix

(e) Scale = $1/384$

Explanation:

$$\vec{f}(\vec{n}) = \sum_{i=1}^{9} \vec{\gamma}_i \, L_i(\vec{n})$$

$\vec{f}(\vec{n})$ x scale leads to $L_i(\vec{n})$ / scale and vice versa to keep $\vec{\gamma}_i$ constant.

Problem #5

$$Y = X_1 + X_2 + Z$$

$X_1, X_2 \sim N(0,5)$ $\quad Z \sim N(0,1)$

prove the theoretical error is

$$5(\hat{w}_1 - 1)^2 + 5(\hat{w}_2 - 1)^2 + 1$$

Solution:

True data $\quad Y = X_1 + X_2 + Z$

model $\quad \hat{Y} = \hat{w}_1 X_1 + \hat{w}_2 X_2$

Error: $\quad \hat{Y} - Y$

$$= X_1(\hat{w}_1 - 1) + X_2(\hat{w}_2 - 1) - Z$$

The expected error

$$E[(\hat{Y} - Y)^2]$$

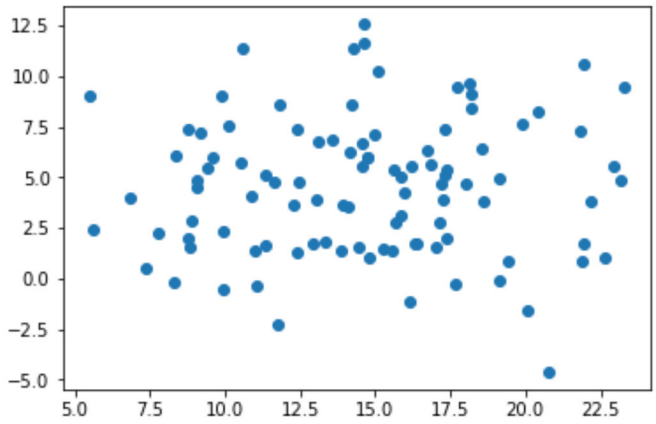
$$= E[X_1^2(\hat{w}_1 - 1)^2] + E[X_2^2(\hat{w}_2 - 1)^2] + E(Z^2)$$
$$+ \cancel{E[X_1 X_2(\hat{w}_1 - 1)(\hat{w}_2 - 1)]}^{0} +$$
$$+ \cancel{E[X_1(\hat{w}_1 - 1)Z]}^{0} + \cancel{E[X_2(\hat{w}_2 - 1)Z]}^{0}$$

$$= \cancel{E[X_1^2]}^{\sigma_{X_1}^2}(\hat{w}_1 - 1)^2 + \cancel{E[X_2^2]}^{\sigma_2^2}(\hat{w}_2 - 1)^2 + \cancel{E(Z^2)}^{\sigma_Z^2}$$

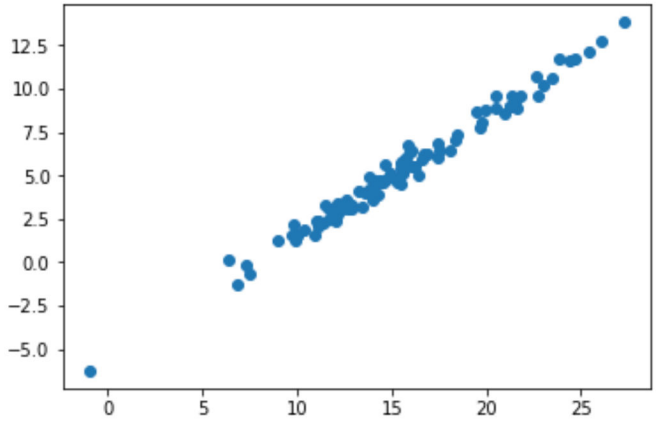
$$= 5(\hat{w}_1 - 1)^2 + 5(\hat{w}_2 - 1)^2 + 1$$

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.mlab import bivariate_normal


# Question 2c

mu = [15, 5]
sigma = [[20, 0], [0, 10]]
samples = np.random.multivariate_normal(mu, sigma, size=100)
mu_estimate = np.mean(samples, axis=0).reshape(2,1)
sigma_estimate = 0
for i in range(100):
    deviation = samples[i, :].reshape(2,1) - mu_estimate
    sigma_estimate += np.matmul(deviation, deviation.T)
sigma_estimate /= 100
print("mu:\n", mu_estimate)
print("Sigma:\n", sigma_estimate)
plt.figure()
plt.scatter(samples[:, 0], samples[:, 1])
plt.show()

sigma = [[20, 14], [14, 10]]
samples = np.random.multivariate_normal(mu, sigma, size=100)
mu_estimate = np.mean(samples, axis=0).reshape(2,1)
sigma_estimate = 0
for i in range(100):
    deviation = samples[i, :].reshape(2,1) - mu_estimate
    sigma_estimate += np.matmul(deviation, deviation.T)
sigma_estimate /= 100
print("mu:\n", mu_estimate)
print("Sigma:\n", sigma_estimate)
plt.figure()
plt.scatter(samples[:, 0], samples[:, 1])
plt.show()

sigma = [[20, -14], [-14, 10]]
samples = np.random.multivariate_normal(mu, sigma, size=100)
mu_estimate = np.mean(samples, axis=0).reshape(2,1)
sigma_estimate = 0
for i in range(100):
    deviation = samples[i, :].reshape(2,1) - mu_estimate
    sigma_estimate += np.matmul(deviation, deviation.T)
sigma_estimate /= 100
print("mu:\n", mu_estimate)
print("Sigma:\n", sigma_estimate)
plt.figure()
plt.scatter(samples[:, 0], samples[:, 1])
plt.show()
```

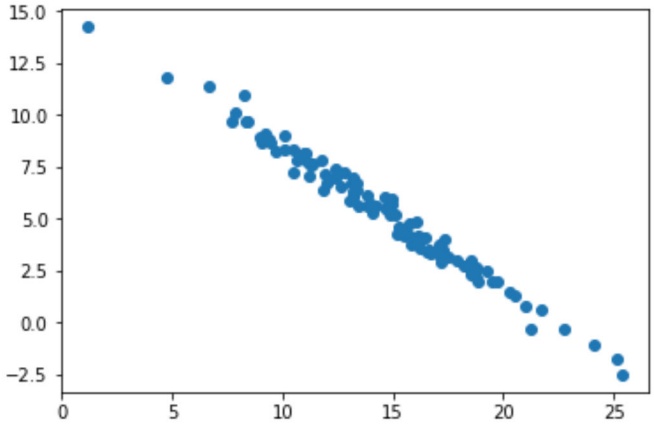
```
mu:
 [[ 14.62635369]
 [  4.53421849]]
Sigma:
 [[ 18.50848944   0.41721358]
 [  0.41721358  11.33299261]]
```



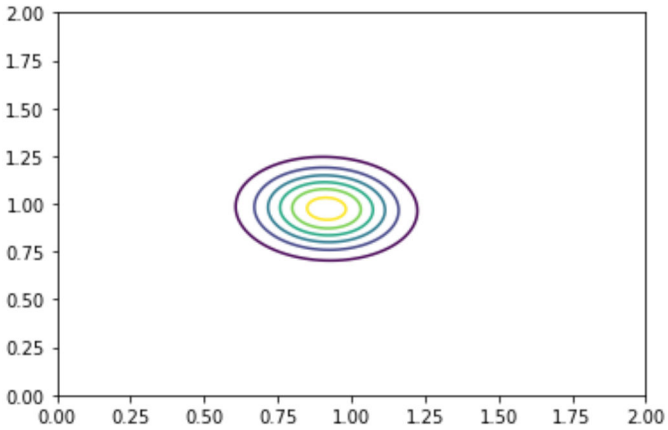

```
mu:
 [[ 15.35608833]
 [  5.22193569]]
Sigma:
 [[ 23.13563725  15.9046762 ]
 [ 15.9046762   11.12067857]]
```



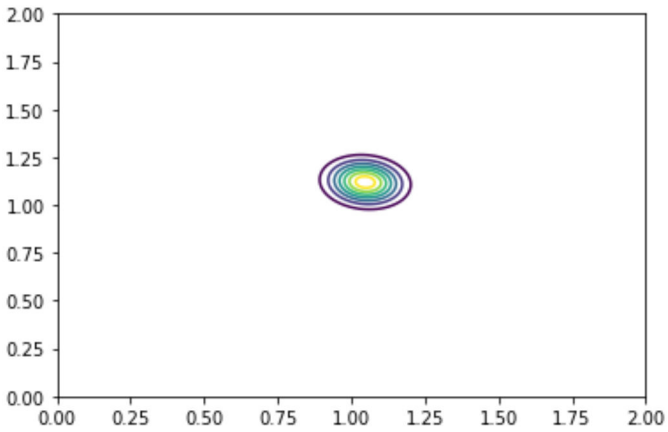

```
mu:
 [[ 14.54418001]
 [  5.3237607 ]]
Sigma:
 [[ 18.4894182  -12.85899727]
 [-12.85899727   9.11141132]]
```



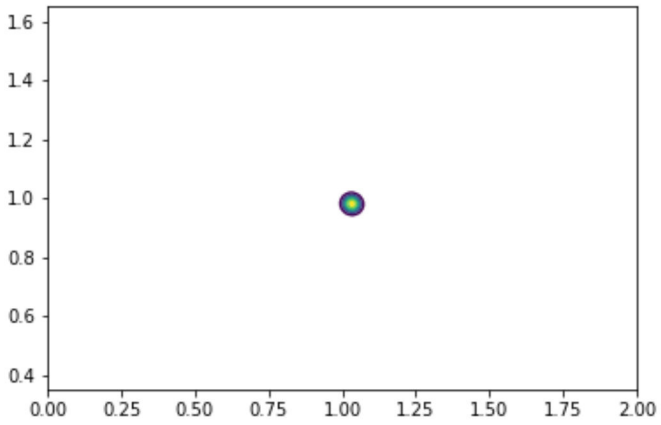
In [41]:
```
# Question 3d

mu_X = [0, 0]
sigma_X = [[5, 0], [0, 5]]
mu_w = [0, 0]
w_0 = np.linspace(0.5, 1.5, 100)
w_1 = np.linspace(0.5, 1.5, 100)
W0, W1 = np.meshgrid(w_0, w_1)

Sigma = [[1, 0], [0, 1]]
Prior = bivariate_normal(W0, W1, Sigma[0][0], Sigma[1][1], 0, 0, Sigma[0][1])
Posterior = np.ones( Prior.shape, "float" )
plt.figure()
n = 5
X = np.random.multivariate_normal(mu_X, sigma_X, size=n)
Z = np.random.normal(0, 1, size=n)
Y = X[:, 0] + X[:, 1] + Z
for i in range(len(w_0)):
    for j in range(len(w_1)):
        w = np.array([W0[i,j], W1[i,j]]).reshape(2, 1)
        likelihood = 1
        for k in range(n):
            X_vector = X[k, :].reshape(2, 1)
            likelihood *= np.exp(-0.5 * (Y[k] - np.matmul(w.T, X_vector)) ** 2)
        Posterior[i, j] = likelihood * Prior[i, j]
plt.contour(W0, W1, Posterior)
plt.axis((0, 2, 0, 2))
plt.show()
```



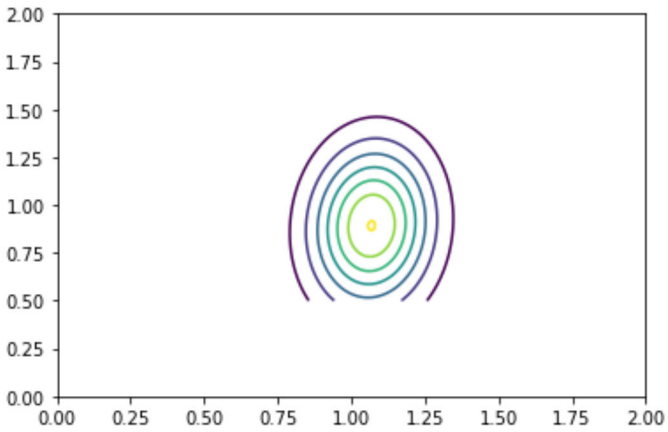
In [42]:
```
Sigma = [[1, 0], [0, 1]]
Prior = bivariate_normal(W0, W1, Sigma[0][0], Sigma[1][1], 0, 0, Sigma[0][1])
Posterior = np.ones( Prior.shape, "float" )
plt.figure()
n = 50
X = np.random.multivariate_normal(mu_X, sigma_X, size=n)
Z = np.random.normal(0, 1, size=n)
Y = X[:, 0] + X[:, 1] + Z
for i in range(len(w_0)):
    for j in range(len(w_1)):
        w = np.array([W0[i,j], W1[i,j]]).reshape(2, 1)
        likelihood = 1
        for k in range(n):
            X_vector = X[k, :].reshape(2, 1)
            likelihood *= np.exp(-0.5 * (Y[k] - np.matmul(w.T, X_vector)) ** 2)
        Posterior[i, j] = likelihood * Prior[i, j]
plt.contour(W0, W1, Posterior)
plt.axis((0, 2, 0, 2))
plt.show()
```



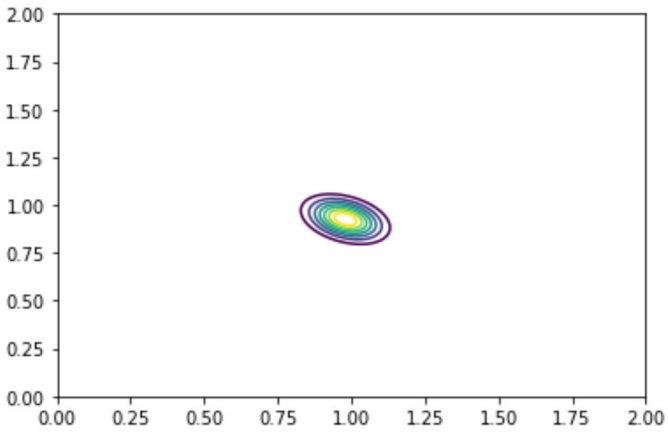
In [40]:
```
Sigma = [[1, 0], [0, 1]]
Prior = bivariate_normal(W0, W1, Sigma[0][0], Sigma[1][1], 0, 0, Sigma[0][1])
Posterior = np.ones( Prior.shape, "float" )
plt.figure()
n = 500
X = np.random.multivariate_normal(mu_X, sigma_X, size=n)
Z = np.random.normal(0, 1, size=n)
Y = X[:, 0] + X[:, 1] + Z
for i in range(len(w_0)):
    for j in range(len(w_1)):
        w = np.array([W0[i,j], W1[i,j]]).reshape(2, 1)
        likelihood = 1
        for k in range(n):
            X_vector = X[k, :].reshape(2, 1)
            likelihood *= np.exp(-0.5 * (Y[k] - np.matmul(w.T, X_vector)) ** 2)
        Posterior[i, j] = likelihood * Prior[i, j]
plt.contour(W0, W1, Posterior)
plt.axis((0, 2, 0, 2))
plt.show()
```



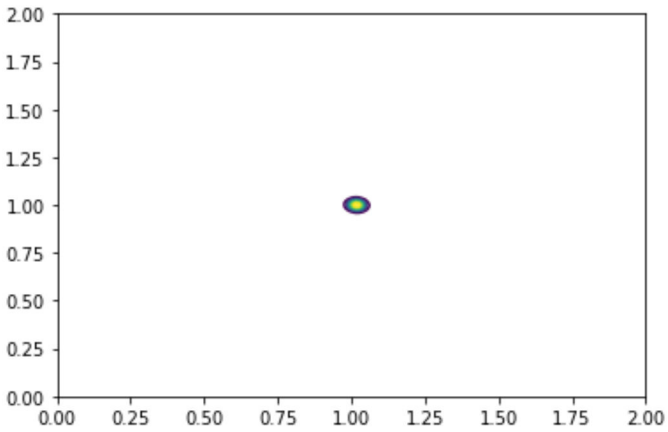
In [43]:
```
Sigma = [[1, 0.25], [0.25, 1]]
Prior = bivariate_normal(W0, W1, Sigma[0][0], Sigma[1][1], 0, 0, Sigma[0][1])
Posterior = np.ones( Prior.shape, "float" )
plt.figure()
n = 5
X = np.random.multivariate_normal(mu_X, sigma_X, size=n)
Z = np.random.normal(0, 1, size=n)
Y = X[:, 0] + X[:, 1] + Z
for i in range(len(w_0)):
    for j in range(len(w_1)):
        w = np.array([W0[i,j], W1[i,j]]).reshape(2, 1)
        likelihood = 1
        for k in range(n):
            X_vector = X[k, :].reshape(2, 1)
            likelihood *= np.exp(-0.5 * (Y[k] - np.matmul(w.T, X_vector)) ** 2)
        Posterior[i, j] = likelihood * Prior[i, j]
plt.contour(W0, W1, Posterior)
plt.axis((0, 2, 0, 2))
plt.show()
```



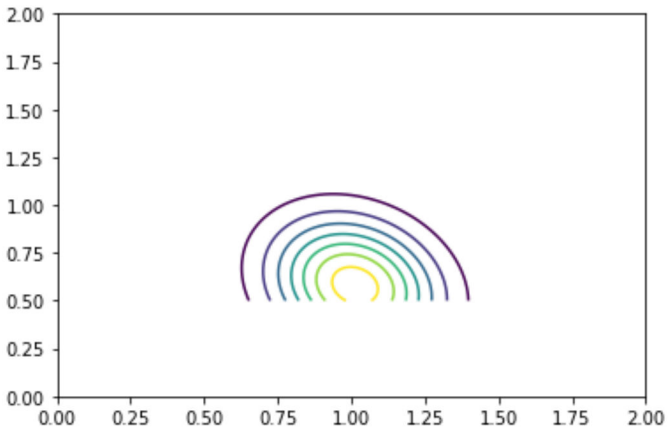
In [44]:
```
Sigma = [[1, 0.25], [0.25, 1]]
Prior = bivariate_normal(W0, W1, Sigma[0][0], Sigma[1][1], 0, 0, Sigma[0][1])
Posterior = np.ones( Prior.shape, "float" )
plt.figure()
n = 50
X = np.random.multivariate_normal(mu_X, sigma_X, size=n)
Z = np.random.normal(0, 1, size=n)
Y = X[:, 0] + X[:, 1] + Z
for i in range(len(w_0)):
    for j in range(len(w_1)):
        w = np.array([W0[i,j], W1[i,j]]).reshape(2, 1)
        likelihood = 1
        for k in range(n):
            X_vector = X[k, :].reshape(2, 1)
            likelihood *= np.exp(-0.5 * (Y[k] - np.matmul(w.T, X_vector)) ** 2)
        Posterior[i, j] = likelihood * Prior[i, j]
plt.contour(W0, W1, Posterior)
plt.axis((0, 2, 0, 2))
plt.show()
```



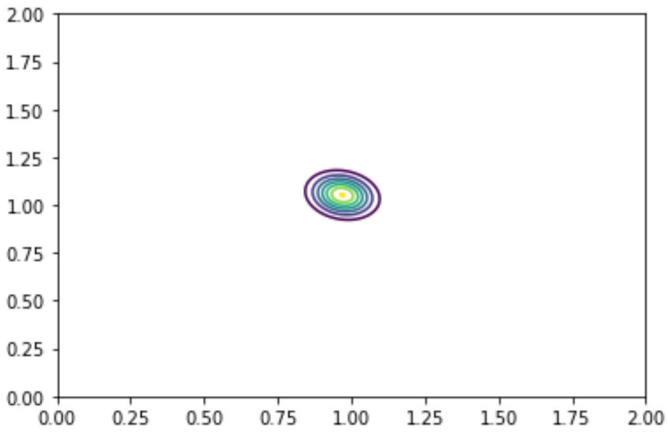
In [45]:
```
Sigma = [[1, 0.25], [0.25, 1]]
Prior = bivariate_normal(W0, W1, Sigma[0][0], Sigma[1][1], 0, 0, Sigma[0][1])
Posterior = np.ones( Prior.shape, "float" )
plt.figure()
n = 500
X = np.random.multivariate_normal(mu_X, sigma_X, size=n)
Z = np.random.normal(0, 1, size=n)
Y = X[:, 0] + X[:, 1] + Z
for i in range(len(w_0)):
    for j in range(len(w_1)):
        w = np.array([W0[i,j], W1[i,j]]).reshape(2, 1)
        likelihood = 1
        for k in range(n):
            X_vector = X[k, :].reshape(2, 1)
            likelihood *= np.exp(-0.5 * (Y[k] - np.matmul(w.T, X_vector)) ** 2)
        Posterior[i, j] = likelihood * Prior[i, j]
plt.contour(W0, W1, Posterior)
plt.axis((0, 2, 0, 2))
plt.show()
```



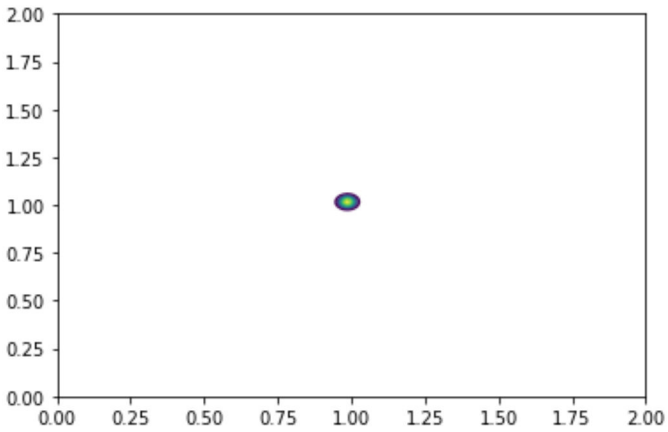
In [46]:
```
Sigma = [[1, 0.9], [0.9, 1]]
Prior = bivariate_normal(W0, W1, Sigma[0][0], Sigma[1][1], 0, 0, Sigma[0][1])
Posterior = np.ones( Prior.shape, "float" )
plt.figure()
n = 5
X = np.random.multivariate_normal(mu_X, sigma_X, size=n)
Z = np.random.normal(0, 1, size=n)
Y = X[:, 0] + X[:, 1] + Z
for i in range(len(w_0)):
    for j in range(len(w_1)):
        w = np.array([W0[i,j], W1[i,j]]).reshape(2, 1)
        likelihood = 1
        for k in range(n):
            X_vector = X[k, :].reshape(2, 1)
            likelihood *= np.exp(-0.5 * (Y[k] - np.matmul(w.T, X_vector)) ** 2)
        Posterior[i, j] = likelihood * Prior[i, j]
plt.contour(W0, W1, Posterior)
plt.axis((0, 2, 0, 2))
plt.show()
```



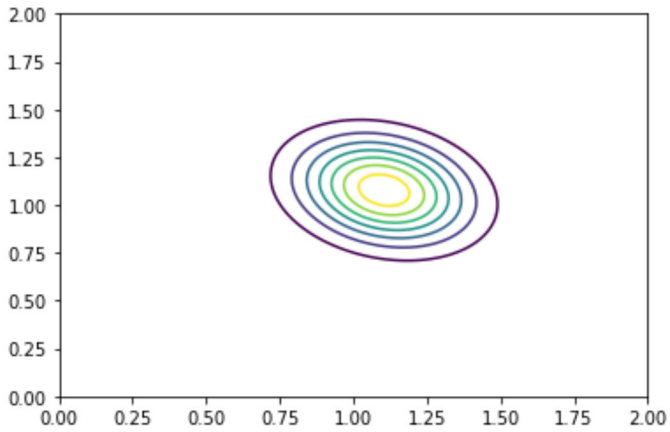
In [47]:
```
Sigma = [[1, 0.9], [0.9, 1]]
Prior = bivariate_normal(W0, W1, Sigma[0][0], Sigma[1][1], 0, 0, Sigma[0][1])
Posterior = np.ones( Prior.shape, "float" )
plt.figure()
n = 50
X = np.random.multivariate_normal(mu_X, sigma_X, size=n)
Z = np.random.normal(0, 1, size=n)
Y = X[:, 0] + X[:, 1] + Z
for i in range(len(w_0)):
    for j in range(len(w_1)):
        w = np.array([W0[i,j], W1[i,j]]).reshape(2, 1)
        likelihood = 1
        for k in range(n):
            X_vector = X[k, :].reshape(2, 1)
            likelihood *= np.exp(-0.5 * (Y[k] - np.matmul(w.T, X_vector)) ** 2)
        Posterior[i, j] = likelihood * Prior[i, j]
plt.contour(W0, W1, Posterior)
plt.axis((0, 2, 0, 2))
plt.show()
```



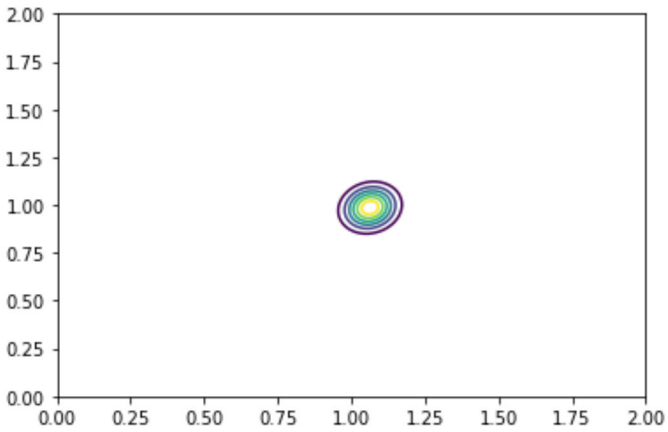
In [48]:
```
Sigma = [[1, 0.9], [0.9, 1]]
Prior = bivariate_normal(W0, W1, Sigma[0][0], Sigma[1][1], 0, 0, Sigma[0][1])
Posterior = np.ones( Prior.shape, "float" )
plt.figure()
n = 500
X = np.random.multivariate_normal(mu_X, sigma_X, size=n)
Z = np.random.normal(0, 1, size=n)
Y = X[:, 0] + X[:, 1] + Z
for i in range(len(w_0)):
    for j in range(len(w_1)):
        w = np.array([W0[i,j], W1[i,j]]).reshape(2, 1)
        likelihood = 1
        for k in range(n):
            X_vector = X[k, :].reshape(2, 1)
            likelihood *= np.exp(-0.5 * (Y[k] - np.matmul(w.T, X_vector)) ** 2)
        Posterior[i, j] = likelihood * Prior[i, j]
plt.contour(W0, W1, Posterior)
plt.axis((0, 2, 0, 2))
plt.show()
```



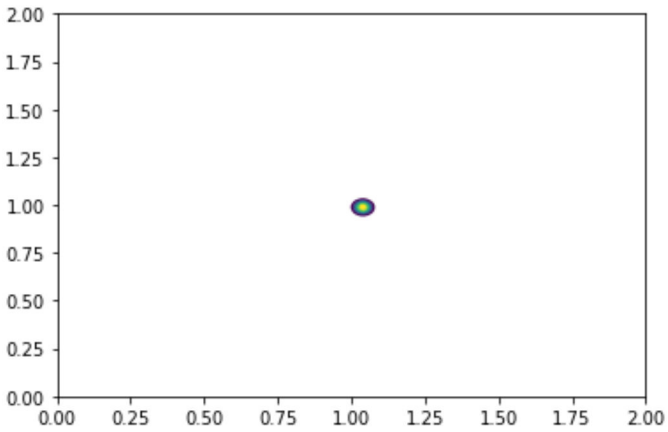
In [49]:
```
Sigma = [[1, -0.25], [-0.25, 1]]
Prior = bivariate_normal(W0, W1, Sigma[0][0], Sigma[1][1], 0, 0, Sigma[0][1])
Posterior = np.ones( Prior.shape, "float" )
plt.figure()
n = 5
X = np.random.multivariate_normal(mu_X, sigma_X, size=n)
Z = np.random.normal(0, 1, size=n)
Y = X[:, 0] + X[:, 1] + Z
for i in range(len(w_0)):
    for j in range(len(w_1)):
        w = np.array([W0[i,j], W1[i,j]]).reshape(2, 1)
        likelihood = 1
        for k in range(n):
            X_vector = X[k, :].reshape(2, 1)
            likelihood *= np.exp(-0.5 * (Y[k] - np.matmul(w.T, X_vector)) ** 2)
        Posterior[i, j] = likelihood * Prior[i, j]
plt.contour(W0, W1, Posterior)
plt.axis((0, 2, 0, 2))
plt.show()
```



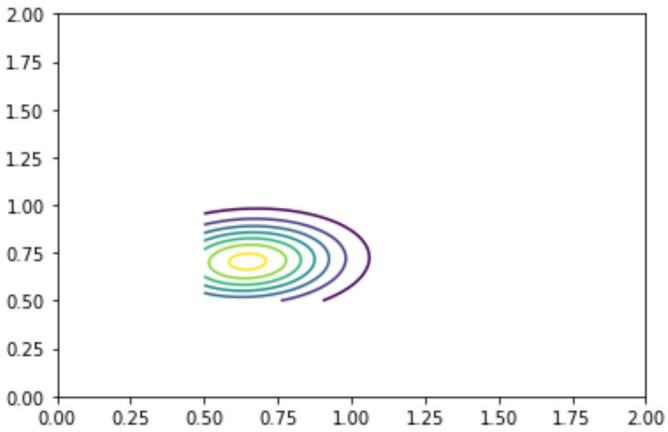
In [50]:
```
Sigma = [[1, -0.25], [-0.25, 1]]
Prior = bivariate_normal(W0, W1, Sigma[0][0], Sigma[1][1], 0, 0, Sigma[0][1])
Posterior = np.ones( Prior.shape, "float" )
plt.figure()
n = 50
X = np.random.multivariate_normal(mu_X, sigma_X, size=n)
Z = np.random.normal(0, 1, size=n)
Y = X[:, 0] + X[:, 1] + Z
for i in range(len(w_0)):
    for j in range(len(w_1)):
        w = np.array([W0[i,j], W1[i,j]]).reshape(2, 1)
        likelihood = 1
        for k in range(n):
            X_vector = X[k, :].reshape(2, 1)
            likelihood *= np.exp(-0.5 * (Y[k] - np.matmul(w.T, X_vector)) ** 2)
        Posterior[i, j] = likelihood * Prior[i, j]
plt.contour(W0, W1, Posterior)
plt.axis((0, 2, 0, 2))
plt.show()
```



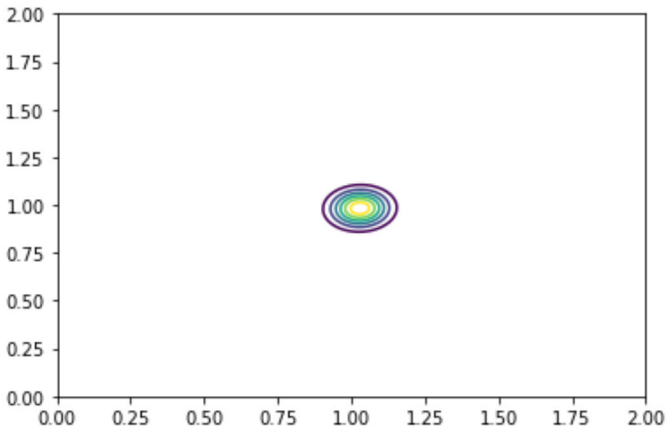
In [51]:
```
Sigma = [[1, -0.25], [-0.25, 1]]
Prior = bivariate_normal(W0, W1, Sigma[0][0], Sigma[1][1], 0, 0, Sigma[0][1])
Posterior = np.ones( Prior.shape, "float" )
plt.figure()
n = 500
X = np.random.multivariate_normal(mu_X, sigma_X, size=n)
Z = np.random.normal(0, 1, size=n)
Y = X[:, 0] + X[:, 1] + Z
for i in range(len(w_0)):
    for j in range(len(w_1)):
        w = np.array([W0[i,j], W1[i,j]]).reshape(2, 1)
        likelihood = 1
        for k in range(n):
            X_vector = X[k, :].reshape(2, 1)
            likelihood *= np.exp(-0.5 * (Y[k] - np.matmul(w.T, X_vector)) ** 2)
        Posterior[i, j] = likelihood * Prior[i, j]
plt.contour(W0, W1, Posterior)
plt.axis((0, 2, 0, 2))
plt.show()
```



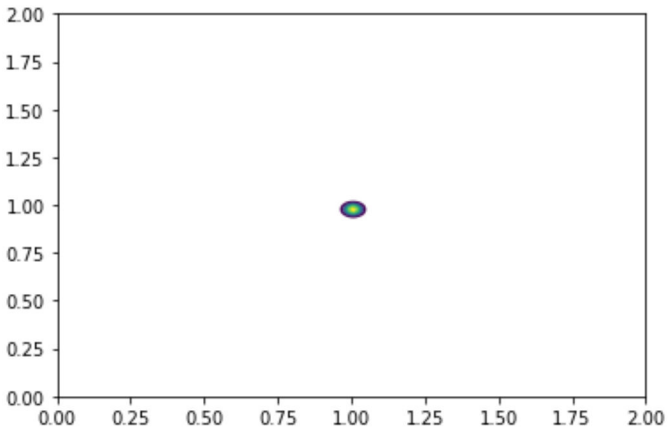
In [52]:
```
Sigma = [[1, -0.9], [-0.9, 1]]
Prior = bivariate_normal(W0, W1, Sigma[0][0], Sigma[1][1], 0, 0, Sigma[0][1])
Posterior = np.ones( Prior.shape, "float" )
plt.figure()
n = 5
X = np.random.multivariate_normal(mu_X, sigma_X, size=n)
Z = np.random.normal(0, 1, size=n)
Y = X[:, 0] + X[:, 1] + Z
for i in range(len(w_0)):
    for j in range(len(w_1)):
        w = np.array([W0[i,j], W1[i,j]]).reshape(2, 1)
        likelihood = 1
        for k in range(n):
            X_vector = X[k, :].reshape(2, 1)
            likelihood *= np.exp(-0.5 * (Y[k] - np.matmul(w.T, X_vector)) ** 2)
        Posterior[i, j] = likelihood * Prior[i, j]
plt.contour(W0, W1, Posterior)
plt.axis((0, 2, 0, 2))
plt.show()
```



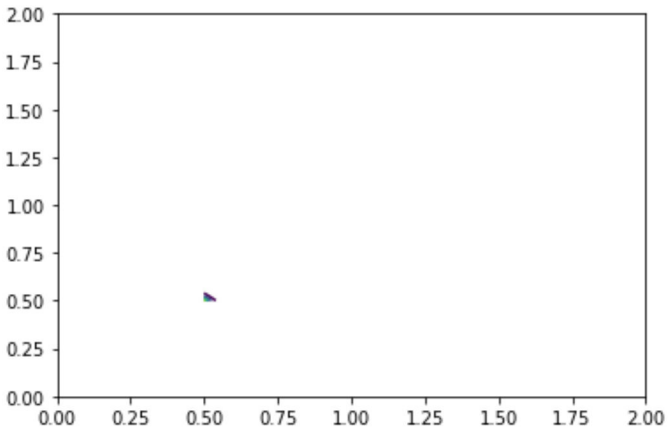
In [53]:
```
Sigma = [[1, -0.9], [-0.9, 1]]
Prior = bivariate_normal(W0, W1, Sigma[0][0], Sigma[1][1], 0, 0, Sigma[0][1])
Posterior = np.ones( Prior.shape, "float" )
plt.figure()
n = 50
X = np.random.multivariate_normal(mu_X, sigma_X, size=n)
Z = np.random.normal(0, 1, size=n)
Y = X[:, 0] + X[:, 1] + Z
for i in range(len(w_0)):
    for j in range(len(w_1)):
        w = np.array([W0[i,j], W1[i,j]]).reshape(2, 1)
        likelihood = 1
        for k in range(n):
            X_vector = X[k, :].reshape(2, 1)
            likelihood *= np.exp(-0.5 * (Y[k] - np.matmul(w.T, X_vector)) ** 2)
        Posterior[i, j] = likelihood * Prior[i, j]
plt.contour(W0, W1, Posterior)
plt.axis((0, 2, 0, 2))
plt.show()
```



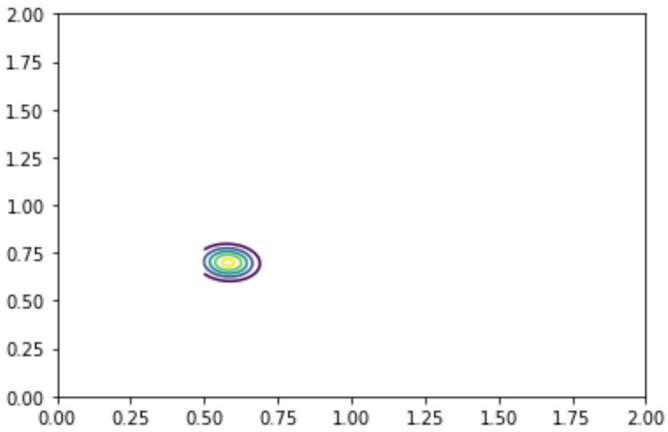
In [54]:
```
Sigma = [[1, -0.9], [-0.9, 1]]
Prior = bivariate_normal(W0, W1, Sigma[0][0], Sigma[1][1], 0, 0, Sigma[0][1])
Posterior = np.ones( Prior.shape, "float" )
plt.figure()
n = 500
X = np.random.multivariate_normal(mu_X, sigma_X, size=n)
Z = np.random.normal(0, 1, size=n)
Y = X[:, 0] + X[:, 1] + Z
for i in range(len(w_0)):
    for j in range(len(w_1)):
        w = np.array([W0[i,j], W1[i,j]]).reshape(2, 1)
        likelihood = 1
        for k in range(n):
            X_vector = X[k, :].reshape(2, 1)
            likelihood *= np.exp(-0.5 * (Y[k] - np.matmul(w.T, X_vector)) ** 2)
        Posterior[i, j] = likelihood * Prior[i, j]
plt.contour(W0, W1, Posterior)
plt.axis((0, 2, 0, 2))
plt.show()
```



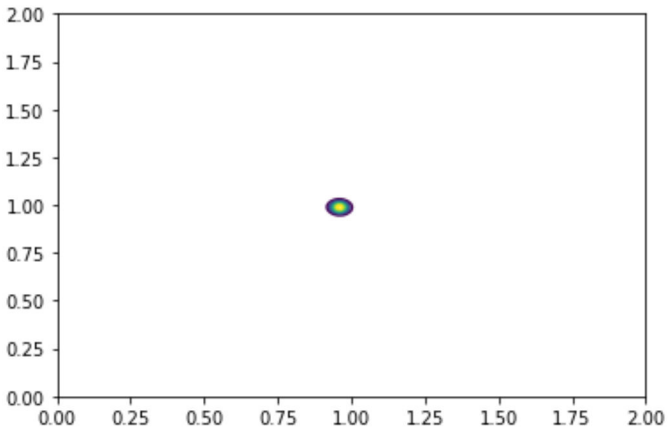
In [61]:
```
Sigma = [[0.1, 0], [0, 0.1]]
Prior = bivariate_normal(W0, W1, Sigma[0][0], Sigma[1][1], 0, 0, Sigma[0][1])
Posterior = np.ones( Prior.shape, "float" )
plt.figure()
n = 5
X = np.random.multivariate_normal(mu_X, sigma_X, size=n)
Z = np.random.normal(0, 1, size=n)
Y = X[:, 0] + X[:, 1] + Z
for i in range(len(w_0)):
    for j in range(len(w_1)):
        w = np.array([W0[i,j], W1[i,j]]).reshape(2, 1)
        likelihood = 1
        for k in range(n):
            X_vector = X[k, :].reshape(2, 1)
            likelihood *= np.exp(-0.5 * (Y[k] - np.matmul(w.T, X_vector)) ** 2)
        Posterior[i, j] = likelihood * Prior[i, j]
plt.contour(W0, W1, Posterior)
plt.axis((0, 2, 0, 2))
plt.show()
```



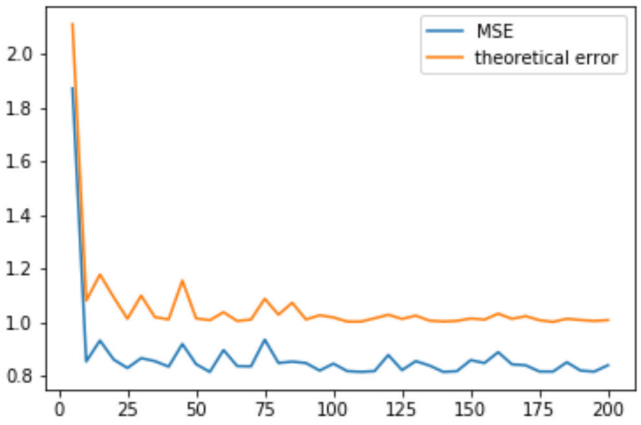
In [60]:
```
Sigma = [[0.1, 0], [0, 0.1]]
Prior = bivariate_normal(W0, W1, Sigma[0][0], Sigma[1][1], 0, 0, Sigma[0][1])
Posterior = np.ones( Prior.shape, "float" )
plt.figure()
n = 50
X = np.random.multivariate_normal(mu_X, sigma_X, size=n)
Z = np.random.normal(0, 1, size=n)
Y = X[:, 0] + X[:, 1] + Z
for i in range(len(w_0)):
    for j in range(len(w_1)):
        w = np.array([W0[i,j], W1[i,j]]).reshape(2, 1)
        likelihood = 1
        for k in range(n):
            X_vector = X[k, :].reshape(2, 1)
            likelihood *= np.exp(-0.5 * (Y[k] - np.matmul(w.T, X_vector)) ** 2)
        Posterior[i, j] = likelihood * Prior[i, j]
plt.contour(W0, W1, Posterior)
plt.axis((0, 2, 0, 2))
plt.show()
```



In [59]:
```
Sigma = [[0.1, 0], [0, 0.1]]
Prior = bivariate_normal(W0, W1, Sigma[0][0], Sigma[1][1], 0, 0, Sigma[0][1])
Posterior = np.ones( Prior.shape, "float" )
plt.figure()
n = 500
X = np.random.multivariate_normal(mu_X, sigma_X, size=n)
Z = np.random.normal(0, 1, size=n)
Y = X[:, 0] + X[:, 1] + Z
for i in range(len(w_0)):
    for j in range(len(w_1)):
        w = np.array([W0[i,j], W1[i,j]]).reshape(2, 1)
        likelihood = 1
        for k in range(n):
            X_vector = X[k, :].reshape(2, 1)
            likelihood *= np.exp(-0.5 * (Y[k] - np.matmul(w.T, X_vector)) ** 2)
        Posterior[i, j] = likelihood * Prior[i, j]
plt.contour(W0, W1, Posterior)
plt.axis((0, 2, 0, 2))
plt.show()
```

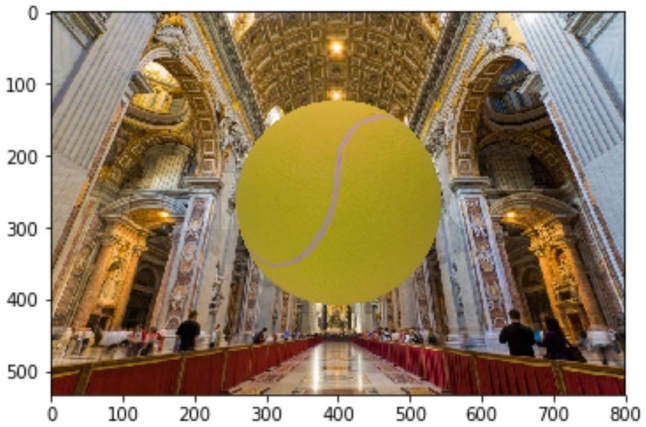


In [79]:
```
# Question 3e

mu_X = [0, 0]
sigma_X = [[5, 0], [0, 5]]
num = np.linspace(5, 200, 40, endpoint=True, dtype="int")
errors = np.zeros( num.shape, "float" )
errors_theor = np.zeros( num.shape, "float" )

X_test = np.random.multivariate_normal(mu_X, sigma_X, size=100)
Z_test = np.random.normal(0, 1, size=100)
Y_test = X_test[:, 0] + X_test[:, 1] + Z_test

for i in range(len(num)):
    X = np.random.multivariate_normal(mu_X, sigma_X, size=num[i])
    Z = np.random.normal(0, 1, size=num[i])
    Y = X[:, 0] + X[:, 1] + Z
    coeff = np.matmul( np.linalg.inv( np.matmul(X.T, X) ), np.matmul(X.T, Y) )
    errors_theor[i] = 5 * (coeff[0] - 1) ** 2 + 5 * (coeff[1] - 1) ** 2 + 1
    e = 0
    for j in range(100):
        e += (Y_test[j] - np.dot(X_test[j, :], coeff)) ** 2
    errors[i] = e / 100
plt.figure()
plt.plot(num, errors, label="MSE")
# plt.show()
# plt.figure()
plt.plot(num, errors_theor, label="theoretical error")
plt.legend()
plt.show()
```

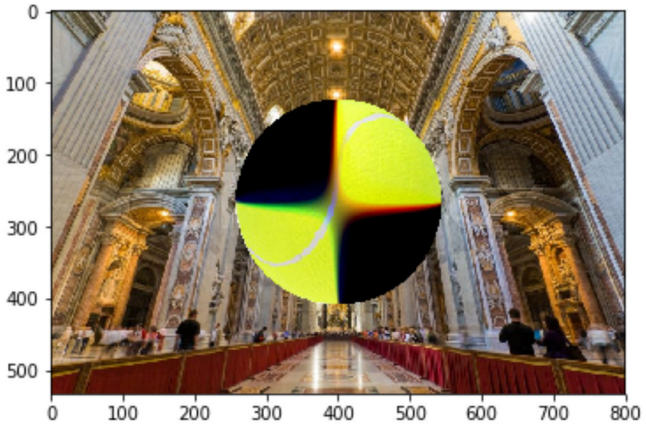


In [4]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.misc import imread,imsave

imFile = 'stpeters_probe_small.png'
compositeFile = 'tennis.png'
targetFile = 'interior.jpg'

# This loads and returns all of the images needed for the problem
# data - the image of the spherical mirror
# tennis - the image of the tennis ball that we will relight
# target - the image that we will paste the tennis ball onto
def loadImages():
    imFile = 'stpeters_probe_small.png'
    compositeFile = 'tennis.png'
    targetFile = 'interior.jpg'

    data = imread(imFile).astype('float')*1.5
    tennis = imread(compositeFile).astype('float')
    target = imread(targetFile).astype('float')/255

    return data, tennis, target


# This function takes as input a square image of size m x m x c
# where c is the number of color channels in the image.  We
# assume that the image contains a scphere and that the edges
# of the sphere touch the edge of the image.
# The output is a tuple (ns, vs) where ns is an n x 3 matrix
# where each row is a unit vector of the direction of incoming light
# vs is an n x c vector where the ith row corresponds with the
# image intensity of incoming light from the corresponding row in ns
def extractNormals(img):

    # Assumes the image is square
    d = img.shape[0]
    r = d / 2
    ns = []
    vs = []
    for i in range(d):
        for j in range(d):

            # Determine if the pixel is on the sphere
            x = j - r
            y = i - r
            if x*x + y*y > r*r-100:
                continue

            # Figure out the normal vector at the point
            # We assume that the image is an orthographic projection
            z = np.sqrt(r*r-x*x-y*y)
            n = np.asarray([x,y,z])
            n = n / np.sqrt(np.sum(np.square(n)))
            view = np.asarray([0,0,-1])
            n = 2*n*(np.sum(n*view))-view
            ns.append(n)
            vs.append(img[i,j])

    return np.asarray(ns), np.asarray(vs)

# This function renders a diffuse sphere of radius r
# using the spherical harmonic coefficients given in
```

```
Question 5c
Coefficients:
[[ 202.31845431  162.41956802  149.07075034]
 [ -27.66555164  -17.88905339  -12.92356688]
 [  -5.15203925   -4.51375871   -4.24262639]
 [  -1.08629293    0.42947012    1.15475569]
 [  -3.14053107   -3.70269907   -3.74382934]
 [  23.67671768   23.15698002   21.94638397]
 [  -3.82167171    0.57606634    1.81637483]
 [   4.7346737     1.4677692    -1.12253649]
 [  -9.72739616   -5.75691108   -4.8395598 ]]
```




```
Question 5d
Coefficients:
[[  2.13318421e+02   1.70780299e+02   1.57126297e+02]
 [ -3.23046362e+01  -2.02975310e+01  -1.45516114e+01]
 [ -4.31689131e+00  -3.80778081e+00  -4.83616306e+00]
 [ -4.89811386e+00  -3.37684058e+00  -1.14207091e+00]
 [ -7.05901066e+03  -7.39934207e+03  -4.26448732e+03]
 [ -3.05378224e+02  -1.56329401e+02   3.50285345e+02]
 [ -9.76079364e+00  -5.33182216e+00  -1.55699782e+00]
 [  7.30792588e+02   3.52130316e+02  -6.11683200e+02]
 [ -9.08887079e+00  -3.84309477e+00  -4.16456437e+00]]
```



```
Question 5e
Coefficients:
[[ 209.38212459  169.03666402  155.36677288]
 [ -30.26805402  -20.30443706  -15.20472049]
 [  -5.753416     -5.07881542   -4.78144904]
 [  -1.05630713    0.46377951    1.19195587]
 [  -7.90569522   -8.20316831   -8.05137623]
 [  54.96251667   52.62398401   50.09265545]
 [  -3.8491927     0.55663535    1.80236903]
 [   7.32655583    3.83064183    1.07500107]
 [ -10.90665749   -6.8522162    -5.87526417]]
```