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?

(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.

2 MLE of Multivariate Gaussian

In lecture, we discussed uses of the multivariate Gaussian distribution. We just assumed that we knew the parameters of the distribution (the mean vector μ and covariance matrix Σ). In practice, though, we will often want to estimate μ and Σ from data. (This will come up even beyond regression-type problems: for example, when we want to use Gaussian models for classification problems.) This problem asks you to derive the Maximum Likelihood Estimate for the mean and variance of a multivariate Gaussian distribution.

(a) Let X have a multivariate Gaussian distribution with mean $\mu \in \mathbb{R}^d$ and covariance matrix $\Sigma \in \mathbb{R}^{d\times d}$. **Write the log likelihood of drawing the n i.i.d. samples** $x_1, \ldots, x_n \in \mathbb{R}^d$ **from X given Σ and μ.**

Solution: First, we calculate the likelihood by separating out the product:

$$\begin{aligned} P(x_1, x_2, \ldots, x_n|\mu, \Sigma) &= P(x_1|\mu,\Sigma)P(x_2|\mu,\Sigma)\ldots P(x_n|\mu,\Sigma) \\ &= \prod_i \frac{1}{(2\pi)^{1/2}|\Sigma|^{1/2}} \exp\{-\frac{1}{2}(x_i-\mu)^T\Sigma^{-1}(x_i-\mu)\} \\ &= \frac{1}{(2\pi)^{n/2}|\Sigma|^{n/2}} \exp\{-\frac{1}{2}\sum_{i=1}^n (x_i-\mu)^T\Sigma^{-1}(x_i-\mu)\} \end{aligned} \tag{1}$$

Then we take the log:

$$\ln P(x_1, x_2, \ldots, x_n|\mu,\Sigma) = -\frac{n}{2}\log 2\pi - \frac{n}{2}\log|\Sigma| - \frac{1}{2}\sum_{i=1}^n (x_i-\mu)^T\Sigma^{-1}(x_i-\mu) \tag{2}$$

(b) **Find MLE of μ and Σ.** For taking derivatives with respect to matrices, you may use any formula in "The Matrix Cookbook" without proof. This is a reasonably involved problem part with lots of steps to get to the answer. We recommend students first do the one-dimensional case and then the two-dimensional case to warm up.

Solution: We find the MLE of the mean by differentiating with respect to μ, which requires using Equation 83 from The Matrix Cookbook.

Equation 83 tells us:

$$\frac{\partial}{\partial \mu}(x_i-\mu)^T\Sigma^{-1}(x_i-\mu) = (\Sigma^{-1}+\Sigma^{-T})(x_i-\mu)(-1) = -2\Sigma^{-1}(x_i-\mu).$$

Therefore,

$$\begin{aligned}
\frac{\partial}{\partial \mu} \ln P(\mu, \Sigma | x_1, x_2, \ldots, x_n) &= \frac{\partial}{\partial \mu} - \frac{1}{2} \sum_{i=1}^{n} (x_i - \mu)^T \Sigma^{-1} (x_i - \mu) \\
&= -\frac{1}{2} \sum_{i=1}^{n} \frac{\partial}{\partial \mu} (x_i - \mu)^T \Sigma^{-1} (x_i - \mu) \\
&= -\frac{1}{2} \sum_{i=1}^{n} -2\Sigma^{-1} (x_i - \mu) \\
&= \sum_{i=1}^{n} \Sigma^{-1} (x_i - \mu)
\end{aligned} \tag{3}$$

We set the expression equal to 0:

$$\sum_{i=1}^{n} \Sigma^{-1} (x_i - \mu) = 0 \implies \sum_{i=1}^{n} (x_i - \mu) = 0 \implies \sum_{i=1}^{n} x_i = n\mu \implies \mu = \frac{\sum_{i=1}^{n} x_i}{n}.$$

We find the MLE of the variance by differentiating with respect to Σ, which requires using Equations 57 and 61 from The Matrix Cookbook.

Equation 57 tells us:

$$\frac{\partial \log |\Sigma|}{\partial \Sigma} = \Sigma^{-T}$$

Equation 61 tells us:

$$\frac{\partial ((x_i - \mu)^T \Sigma^{-1} (x_i - \mu))}{\partial \Sigma} = -\Sigma^{-T} (x_i - \mu)(x_i - \mu)^T \Sigma^{-T}$$

Therefore,

$$\begin{aligned}
\frac{\partial}{\partial \Sigma} \ln P(\mu, \Sigma | x_1, x_2, \ldots, x_n) &= \frac{\partial}{\partial \Sigma} (-\frac{n}{2} \log |\Sigma| - \frac{1}{2} \sum_{i=1}^{n} (x_i - \mu)^T \Sigma^{-1} (x_i - \mu)) \\
&= -\frac{n}{2} \frac{\partial}{\partial \Sigma} (\log |\Sigma|) - \frac{1}{2} \sum_{i=1}^{n} \frac{\partial}{\partial \Sigma} ((x_i - \mu)^T \Sigma^{-1} (x_i - \mu)) \\
&= -\frac{n}{2} \Sigma^{-T} - \frac{1}{2} \sum_{i=1}^{n} -\Sigma^{-T} (x_i - \mu)(x_i - \mu)^T \Sigma^{-T}
\end{aligned} \tag{4}$$

We set the expression equal to 0. Remember that Σ is symmetric, so $\Sigma = \Sigma^T$:

$$\begin{aligned}
0 &= -\frac{n}{2} \Sigma^{-T} - \frac{1}{2} \sum_{i=1}^{n} -\Sigma^{-T} (x_i - \mu)(x_i - \mu)^T \Sigma^{-T} \\
\implies 0 &= n + \sum_{i=1}^{n} -(x_i - \mu)(x_i - \mu)^T \Sigma^{-1} \\
\implies \sum_{i=1}^{n} (x_i - \mu)(x_i - \mu)^T &= n\Sigma \\
\implies \frac{1}{n} \sum_{i=1}^{n} (x_i - \mu)(x_i - \mu)^T &= \Sigma
\end{aligned} \tag{5}$$

(c) Use the following code to sample from a two-dimensional Multivariate Gaussian and plot the samples:

```
import numpy as np
import matplotlib.pyplot as plt
mu = [15, 5]
sigma = [[20, 0], [0, 10]]
samples = np.random.multivariate_normal(mu, sigma, size=100)
plt.scatter(samples[:, 0], samples[:, 1])
plt.show()
```

Try the following three values of Σ:

$$\Sigma = [[20,0],[0,10]]; \Sigma = [[20,14],[14,10]]; \Sigma = [[20,-14],[-14,10]].$$

Calculate the mean and variance of these distributions from the samples (that is, implement part (b)). Report your results. Include your code in your write-up.

Solution: There is no "correct" answer for the sample mean and variance, since it depends on randomness, but hopefully you got values for the MLE mean and variance that were close to the true mean and variance. If they are very far off, this may indicate an error in your code. The following code works by computing the formula above directly (including the matrix multiplication):

```
sample_mean = np.average(samples, axis=0)
sample_variance = 0.01 * np.sum(np.matrix((samples[i]-sample_mean)).T
                                np.matrix((samples[i]-sample_mean))
                                for i in range(100))
```

3 Tikhonov Regularization and Weighted Least Squares

In this problem, we view Tikhonov regularization from a probabilistic standpoint.

In lecture, you have seen this worked out in one way. In an earlier homework, we introduced Tikhonov regularization as a generalization of ridge regression. In this problem, you are mainly going to use the computer to help deepen your understanding of how priors and thus the right regularization can help.

(a) Let $X \in \mathbb{R}^d$ be a d-dimensional random vector and $Y \in \mathbb{R}$ be a one-dimensional random variable. Assume a linear model between X and Y: $Y = w^T X + z$ where $z \in \mathbb{R}$, and $w \in \mathbb{R}^d$. Also assume that $z \sim N(0,1)$ is a Gaussian random variable. Assume $w \sim N(0,\Sigma)$ where Σ is a symmetric positive definite matrix and this random variable is independent of the observation noise. **What is the conditional distribution of Y given X and w?**

Solution: A Gaussian random variable plus a constant is a Gaussian random variable with the mean having that constant added to it while the random variable keeps the the same variance, so $Y|X,w \sim N(X^T w, 1)$. Concretely, we have

$$f(Y|X) = \frac{1}{\sqrt{2\pi}} \exp\{-\frac{1}{2}(Y - X^T w)^2\}, \tag{6}$$

the pdf of Gaussian distribution.

(b) (Tikhonov regularization) Given n training data points $\{(X_1,Y_1),(X_2,Y_2),\dots,(X_n,Y_n)\}$ generated according to the model of (X,Y) in the previous part (the w is common, but the observation noise varies across the different training points. The X_i are distinct and are arbitrary.), **derive the posterior distribution of w given the training data.** Note: w and $Y = (Y_1,Y_2,\dots,Y_n)$ are jointly Gaussian in this problem given the X_i.

Hint: (You may or may not find this useful) If the probability density function of a random variable is of the form

$$f(X) = C \cdot \exp\{-\frac{1}{2}X^T AX + b^T X\}, \tag{7}$$

where C is some constant to make $f(X)$ integrates to 1, then the mean of X is $A^{-1}b$. This can be used to help complete squares if you choose to go that way.

This case was derived in lecture, and you are free to replicate the derivation from lecture in your own language. You are also free to give a different derivation from first principles if you like.

Solution:

By Bayes' Theorem, we have

$$\begin{aligned} f(w|X_1,Y_1,\cdots,X_n,Y_n) &\propto [\prod_{i=1}^{n} f(Y_i|X_i,w)] f(w) \\ &\propto [\prod_{i=1}^{n} \exp\{-\frac{1}{2}(Y_i - X_i^T w)^2\}] \exp\{-\frac{w^T\Sigma^{-1}w}{2}\} \\ &\propto [\exp\{-\frac{1}{2}\sum_{i=1}^{n}(Y_i - X_i^T w)^2\}] \exp\{-\frac{w^T\Sigma^{-1}w}{2}\} \\ &\propto \exp\{-\frac{1}{2}[w^T(X^TX + \Sigma^{-1})w - 2(X^TY)^T w]\}, \end{aligned}$$

with $X \in \mathbb{R}^{n\times d}$ whose ith row is X_i^T and $Y \in \mathbb{R}^n$ whose ith entry is Y_i. In the above derivation, the first line follows from Bayes' Theorem and omitting the constant terms $f(X_1,Y_1,\dots,X_n,Y_n)$ in the denominator which don't depend on w. The second follows from writing out the pdf directly. The fourth line follows from expanding out the squared term.

Therefore the posterior of w given $X_1,Y_1,\cdots,X_n,Y_n$ is a multivariate Gaussian $N((X^TX + \Sigma^{-1})^{-1}X^TY, (X^TX + \Sigma^{-1})^{-1})$, which follows by using the following lemma.

Lemma: If the probability density function of a random variable is of the form

$$f(X) = C \cdot \exp\{-\frac{1}{2}X^T AX + b^T X\}, \tag{8}$$

where C is some constant to make $f(X)$ integrate to 1, and A is a symmetric positive definite matrix, then X is distributed as $N(A^{-1}b, A^{-1})$.

The proof of the lemma is as follows:

$$\begin{aligned} f(X) &= C \cdot \exp\{-\frac{1}{2}X^T AX + b^T X\}, \\ &= C_1 \cdot \exp\{-\frac{1}{2}(X-(A^{-1}b))^T A(X-(A^{-1}b)) - \frac{1}{2}b^T A^{-1} b\}, \\ &= C_2 \cdot \exp\{-\frac{1}{2}(X-(A^{-1}b))^T A(X-(A^{-1}b))\}, \end{aligned}$$

which is the density of a Gaussian random variable with mean $A^{-1}b$ and covariance matrix A^{-1}.

(c) **(BONUS but in-scope.)** How would you extend your result from the previous part to the case where the observation noise Z was no longer $N(0, I_n)$ but was instead distributed as $N(\mu_z, \Sigma_z)$ for some mean μ_z and some covariance Σ_z, but was independent of the parameter w. **Give an argument based on appropriately changing coordinates.**

Solution: By changing variables, our goal is to somehow reduce to the previous case — with white observation noises.

Define V such that $Z = \mu_z + \Sigma_z^{\frac{1}{2}} V$. Then $V \sim N(0, I)$ and $V = \Sigma_z^{-\frac{1}{2}}(Z - \mu_z)$. ($\Sigma_z^{1/2}$ is the matrix such that $\Sigma_z^{1/2}\Sigma_z^{T/2} = \Sigma_z$, which can be found by the eigenvalue decomposition of $\Sigma_z = U\Delta U^T$ and exploiting the spectral theorem's guarantee of orthonormal eigenvalues and all positive eigenvalues to form the diagonal matrix Δ. Taking $\Sigma_z^{\frac{1}{2}} = U\Delta^{\frac{1}{2}}$ gives us what we want.)

Then $\Sigma_z^{-\frac{1}{2}}(Y - \mu_z) = \Sigma_z^{-\frac{1}{2}} X w + V$ with $V \sim N(0, I)$. This makes the reasonable assumption that the Σ_z is invertible, since otherwise, there would be some dimension in which there is essentially no noise. So the problem reduces to the previous part by taking $\Sigma_z^{-\frac{1}{2}} X$ as X and $\Sigma_z^{-\frac{1}{2}}(Y - \mu_z)$ as Y, which yields the posterior of w given $X_1, Y_1, \ldots, X_n, Y_n$ as a multivariate Gaussian $N((X^T\Sigma_z^{-1}X + \Sigma^{-1})^{-1} X^T \Sigma_z^{-1}(Y - \mu_z), (X^T\Sigma_z^{-1}X + \Sigma^{-1})^{-1})$.

(d) (Compare the effect of different priors) Do the following for $\Sigma = \Sigma_1, \Sigma_2, \Sigma_3, \Sigma_4, \Sigma_5, \Sigma_6$ respectively, where

$$\Sigma_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$
$$\Sigma_2 = \begin{bmatrix} 1 & 0.25 \\ 0.25 & 1 \end{bmatrix}$$
$$\Sigma_3 = \begin{bmatrix} 1 & 0.9 \\ 0.9 & 1 \end{bmatrix}$$
$$\Sigma_4 = \begin{bmatrix} 1 & -0.25 \\ -0.25 & 1 \end{bmatrix}$$
$$\Sigma_5 = \begin{bmatrix} 1 & -0.9 \\ -0.9 & 1 \end{bmatrix}$$
$$\Sigma_6 = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix}$$

In the above cases, the priors on the two parameters are independent with large variance, mildly positively correlated, strongly positively correlated, mildly negatively correlated, strongly negatively correlated, and then independent with small variances respectively.

Generate $5, 50$, and 500 data points from $Y = X_1 + X_2 + Z$ where $X_1, X_2 \sim N(0,5)$ and $Z \sim N(0,1)$ as training data. **Plot the contours of the posteriors on w for all 18 cases of assumed priors and number of data points. What do you observe?**

Hint: Use matplotlib.mlab.bivariate_normal to generate the value of pdf for bivariate normals.

Solution:

```
import matplotlib
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(0)

def generate_data(n):
    """
    This function generates data of size n.
    """
    X = np.random.randn(n,2)*np.sqrt(5)
    z = np.random.randn(n)
    y = np.sum(X, axis=1) + z #* 0.3
    return (X,y)

def tikhonov_regression(X,Y,Sigma):
    """
    This function computes w based on the formula of tikhonov_regression.
    """
    w = np.linalg.inv((X.T.dot(X)+np.linalg.inv(Sigma))).dot(X.T.dot(Y))
    return w

def compute_mean_var(X,y,Sigma):
    """
    This function computes the mean and variance of the posterior
    """
    mean = tikhonov_regression(X,y,Sigma)
    var = np.linalg.inv(X.T.dot(X)+np.linalg.inv(Sigma))
    mux,muy = mean
    sigmax = np.sqrt(var[0,0])
    sigmay = np.sqrt(var[-1,-1])
    sigmaxy = var[0,-1]
    return mux,muy,sigmax,sigmay,sigmaxy

Sigmas = [np.array([[1,0],[0,1]]),
          np.array([[1,0.25],[0.25,1]]),
          np.array([[1,0.9],[0.9,1]]),
          np.array([[1,-0.25],[-0.25,1]]),
          np.array([[1,-0.9],[-0.9,1]]),
        np.array([[0.1,0],[0,0.1]])]
names = [str(i) for i in range(1,6+1)]
for num_data in [5,50,500]:
    X,Y = generate_data(num_data)
    for i,Sigma in enumerate(Sigmas):
        # compute the mean and covariance of posterior.
        mux,muy,sigmax,sigmay,sigmaxy = compute_mean_var(X,Y,Sigma)

        x = np.arange(0.5, 1.5, 0.01)
        y = np.arange(0.5, 1.5, 0.01)
        X_grid, Y_grid = np.meshgrid(x, y)
        # Generate the function values of bivariate normal.
        Z = matplotlib.mlab.bivariate_normal(X_grid,Y_grid,
                                              sigmax, sigmay, mux,
                                              muy, sigmaxy)
        # plot
        plt.figure(figsize=(10,10))
        CS = plt.contour(X_grid, Y_grid, Z,
                          levels = np.concatenate([np.arange(0,0.05,0.01),np.arange
    (0.05,1,0.05)]))
        plt.clabel(CS, inline=1, fontsize=10)
        plt.xlabel('X')
        plt.ylabel('Y')
        plt.title('Sigma'+ names[i] + ' with num_data = {}'.format(num_data))
        plt.savefig('Sigma'+ names[i] + '_num_data_{}.png'.format(num_data))
        plt.close()
```

We can observe that a good prior matters, but enough data will wash away the effect of the prior.

(e) (Influence of Priors) Generate n training data samples from $Y = X_1 + X_2 + Z$ where $X_1, X_2 \sim N(0,5)$ and $Z \sim N(0,1)$ as before. Notice that the true parameters $w_1 = 1, w_2 = 1$ are moderately large and positively correlated with each other. We want to quantitatively understand how the effect of the prior influences the mean square error as we get more training data. This should corroborate the qualitative results you saw in the previous part.

In this case, we could directly compute the "test error" once we see the estimated parameters. Namely, if we estimate after learning that the model is $Y = \widehat{w}_1 X_1 + \widehat{w}_2 X_2 + Z$, the average test error should theoretically be $5(\widehat{w}_1 - 1)^2 + 5(\widehat{w}_2 - 1)^2 + 1$. (Make sure you understand why.)

However, it is good to also keep in mind what happens with a finite amount of test data. So, generate a test set of 100 data points from this model and use the test error on that to evaluate

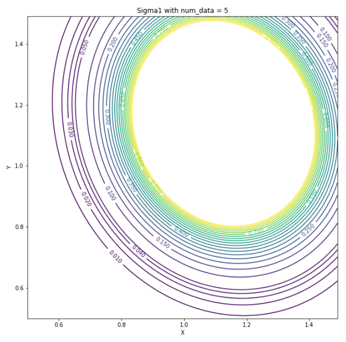


Figure 1: $n = 5$

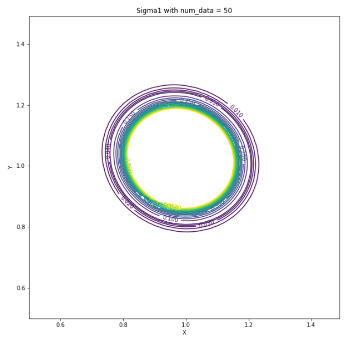


Figure 2: $n = 50$

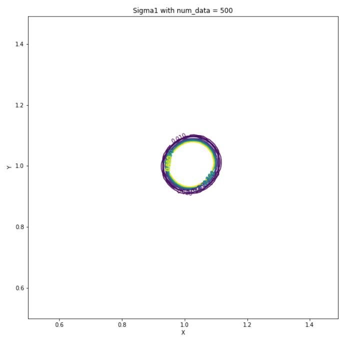


Figure 3: $n = 500$

Figure 4: Σ_1

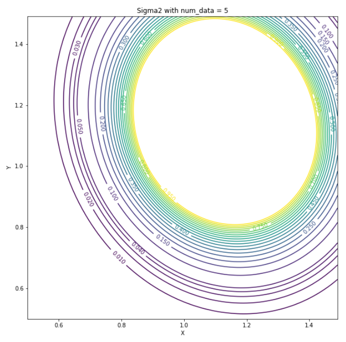


Figure 5: $n = 5$

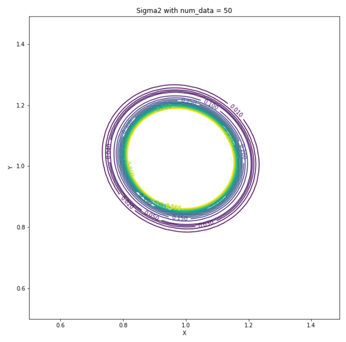


Figure 6: $n = 50$

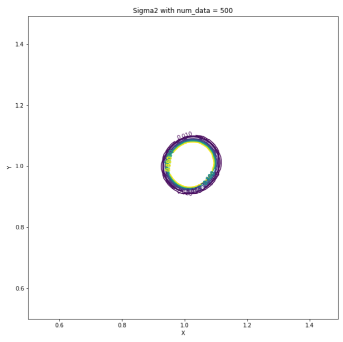


Figure 7: $n = 500$

Figure 8: Σ_2

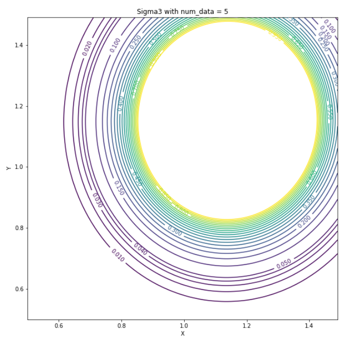


Figure 9: $n = 5$

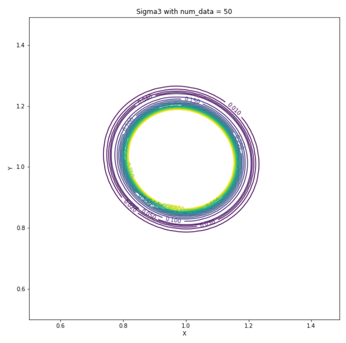


Figure 10: $n = 50$

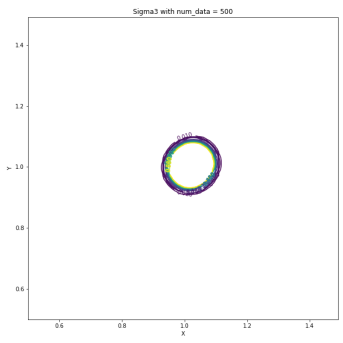


Figure 11: $n = 500$

Figure 12: Σ_3

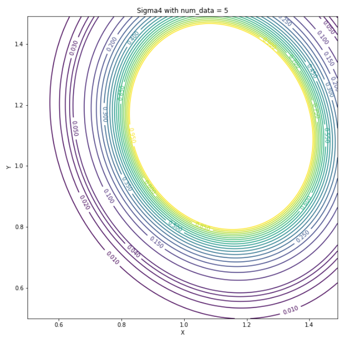


Figure 13: $n = 5$

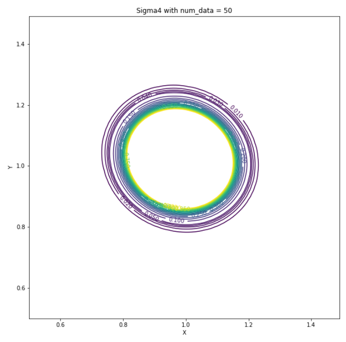


Figure 14: $n = 50$

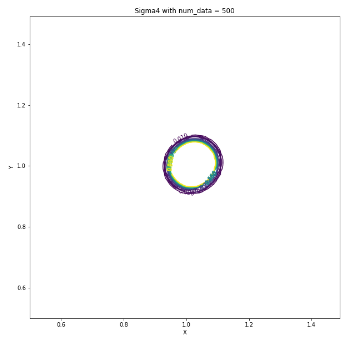


Figure 15: $n = 500$

Figure 16: Σ_4

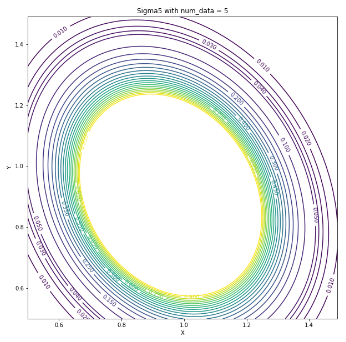


Figure 17: $n = 5$

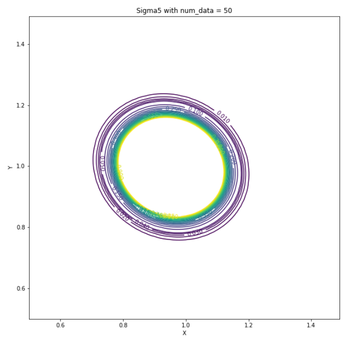


Figure 18: $n = 50$

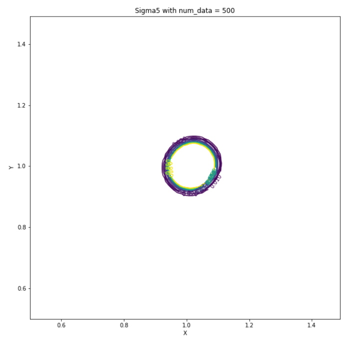


Figure 19: $n = 500$

Figure 20: Σ_5

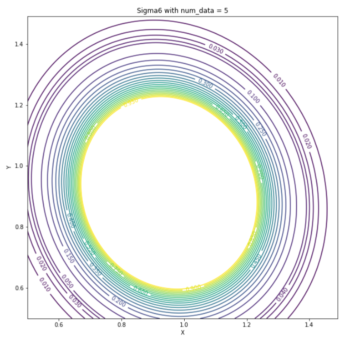


Figure 21: $n = 5$

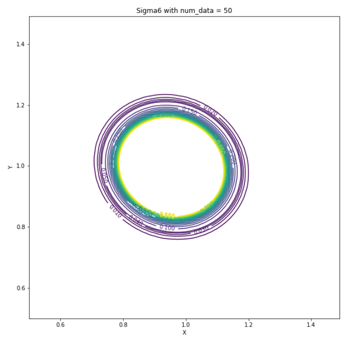


Figure 22: $n = 50$

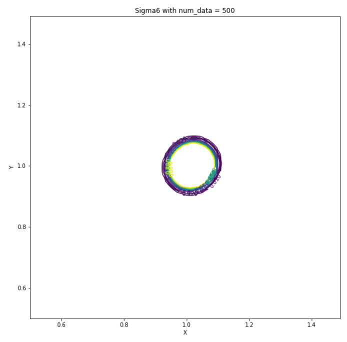


Figure 23: $n = 500$

Figure 24: Σ_6

what happens as we use more training data.

If we just plotted the average test-error (or the theoretical average test error) with respect to the amount of training data, we still have the randomness in the specific training data that was used. Consequently, it is worth replicating this experiment a few times (say 20 times) to get an average of averages. (It is also insightful to look at the spread.)

Plot the mean square error between $\hat{Y}_i$ and Y_i over the test data with respect to the size of training data n (increase n from 5 to 200 by 5). Include what the theoretical mean square error should be for those parameter estimates. Compare what happens for different priors as the amount of training data increases.

Recall the MSE is defined by

$$\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2. \tag{9}$$

Here $\hat{Y}_i = \widehat{w}^T X_i$ where $X_i \in \mathbb{R}^2$ and $\widehat{w}$ in your model is the solution to the least square problem with Tikhonov regularization given the training data.

You should observe that

- Enough data will wash away the effects of the prior.
- A good prior helps when there are not enough data points.

Solution:

```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(0)
w = [1.0,1.0]
n_test = 100
n_trains = np.arange(5,205,5) #np.arange(10,110,10)#np.arange(100,1100,100)
Sigmas = [np.array([[1,0],[0,1]]),
          np.array([[1,0.25],[0.25,1]]),
          np.array([[1,0.9],[0.9,1]]),
          np.array([[1,-0.25],[-0.25,1]]),
          np.array([[1,-0.9],[-0.9,1]]),
        np.array([[0.1,0],[0,0.1]])]
names = ['Sigma{}'.format(i+1) for i in range(6)]
# names = ['uncorrelated','positively correlated','negatively correlated']

def generate_data(n):
    """
    This function generates data of size n.
    """
    X = np.random.randn(n,2) * np.sqrt(5)
    z = np.random.randn(n)
    y = np.sum(X,axis=1) + z #* 0.3
    return (X,y)

def tikhonov_regression(X,Y,Sigma):
    """
    This function computes w based on the formula of tikhonov_regression.
    """
    w = np.linalg.inv((X.T.dot(X)+np.linalg.inv(Sigma))).dot(X.T.dot(Y))
    return w

def compute_mse(X,Y, w):
    """
    This function computes MSE given data and estimated w.
    """
    mse = np.mean((np.squeeze(X.dot(w)) - Y)**2)
    return mse

# Generate Test Data.
X_test , y_test = generate_data(n_test)

mses = np.zeros((len(Sigmas), len(n_trains), 10))


theoretical_mses = np.zeros((len(Sigmas), len(n_trains), 10))


for seed in range(10):
    np.random.seed(seed)
    for i,Sigma in enumerate(Sigmas):
        for j,n_train in enumerate(n_trains):
            X,y = generate_data(n_train) # Generate data.
            w = tikhonov_regression(X,y,Sigma) # Estimate w.
            mse = compute_mse(X_test , y_test , w) # Compute MSE.
            theoretical_mses[i,j,seed] = sum((w-1) ** 2) * 5 + 1
            mses[i,j,seed] = mse

plt.figure()
# Plot
for i,_ in enumerate(Sigmas):
    plt.plot(n_trains , np.mean(mses[i],axis = -1),label = names[i])
plt.xlabel('Number of data')
plt.ylabel('MSE on Test Data')
# plt.yscale('log')
plt.legend()
# plt.show()
plt.savefig('MSE.png')

plt.figure()
# Plot
for i,_ in enumerate(Sigmas):
    plt.plot(n_trains , np.mean(theoretical_mses[i],axis = -1),label = names[i])
plt.xlabel('Number of data')
plt.ylabel('MSE on Test Data')
# plt.yscale('log')
plt.legend()
# plt.show()
plt.savefig('theoretical_MSE.png')

# log Plot
plt.figure()
for i,_ in enumerate(Sigmas):
    plt.plot(n_trains , np.mean(mses[i]-1,axis = -1),label = names[i])
plt.xlabel('Number of data')
plt.ylabel('MSE on Test Data')
plt.xscale('log')
plt.yscale('log')
plt.legend()
```

Figure 25: Average (over runs) MSE on Test Data. We can see this dropping but it is hard to understand what exactly is going on.

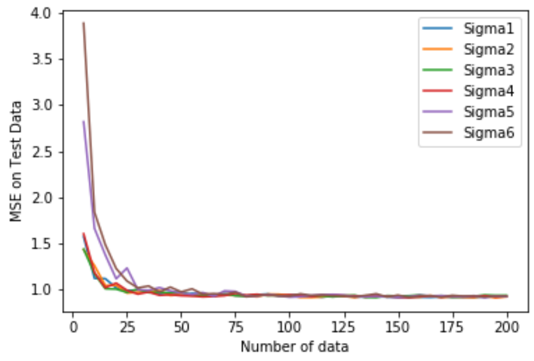
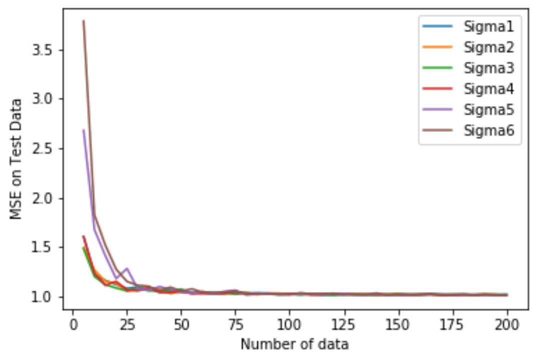


Figure 26: Average (over runs) Theoretical MSE. This reveals that there are still some wiggles — so the wiggles are also coming from variance in our estimator (given that we are only taking the average over a finite number of runs), not random peculiarities of our test data.

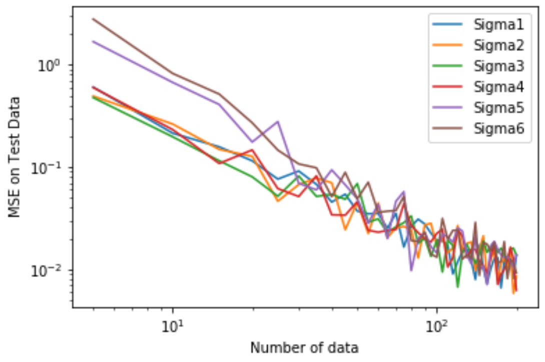


```
# plt.show()
plt.savefig('log_MSE.png')

plt.figure()
# Plot
for i,_ in enumerate(Sigmas):
    plt.plot(n_trains, np.mean(theoretical_mses[i]-1,axis = -1),label = names[i])
plt.xlabel('Number of data')
plt.ylabel('MSE on Test Data')
plt.xscale('log')
plt.yscale('log')
plt.legend()
# plt.show()
plt.savefig('log_theoretical_MSE.png')
```

We first plot the test MSE and theoretical MSE in the usual scale. We observe both of them decreases as n increases. But the lines are messed up with each other and indistinguishable. Then we use a log scale for both x and y axis. Interestingly, we observe a linear relationship under the log scale, which follows from the fact that MSE decreases as the reciprocal of n, the

Figure 27: Log log plot of average (over runs) theoretical MSE. By looking at a log-log plot, where both axes are on log scale, we can see more clearly the pattern — the quality of the prior impacts where we start with small amounts of training data, but the rate of convergence is given by a power-law.



number of training samples. And we can distinguish different priors better.

Three observations:

- Enough data will wash away the effect of prior.
- A good prior helps when there are not enough data.
- How to plot is IMPORTANT for scientific observation.

4 Total Least Squares

Recall that in the least squares problem, we want to solve for $\vec{w}$ in $\min_{\vec{w}} ||A\vec{w}-\vec{y}||$. We measure the error as the difference between $A\vec{w}$ and $\vec{y}$, which can be viewed as adding an error term $\hat{\vec{y}}$ such that the equation $A\vec{w}=\vec{y}+\hat{\vec{y}}$ has a solution:

$$\min ||\hat{\vec{y}}||_F, \text{ subject to } A\vec{w}=\vec{y}+\hat{\vec{y}} \tag{10}$$

Although this optimization formulation allows for errors in the measurements of $\vec{y}$, it does not allow for errors in the feature matrix A that is measured from the data. In this problem, we will explore a method called *total least squares* that allows for both error in the matrix A and the vector $\vec{y}$, represented by $\hat{A}$ and $\hat{\vec{y}}$, respectively. Specifically, the total least squares problem is to find the solution for $\vec{w}$ in the following minimization problem:

$$\min ||[\hat{A},\hat{\vec{y}}]||_F, \text{ subject to } (A+\hat{A})\vec{w}=\vec{y}+\hat{\vec{y}} \tag{11}$$

where the matrix $[\hat{A},\hat{y}]$ is the concatenation of the columns of $\hat{A}$ with the column vector $\vec{y}$. Intuitively, this equation is finding the smallest perturbation to the matrix of data points A and the outputs $\vec{y}$ such that the linear model can be solved exactly. This minimization problem can be rewritten as

$$\min_{\hat{A},\hat{\vec{y}}} [A+\hat{A},\vec{y}+\hat{\vec{y}}] \begin{bmatrix} \vec{w} \\ -1 \end{bmatrix} = \vec{0} \tag{12}$$

(a) Let the matrix $A \in \mathbb{R}^{n\times d}$ and $\vec{y} \in \mathbb{R}^n$. Assuming that $n > d$ and $\text{rank}(A+\hat{A}) = d$, **explain why rank**$([A+\hat{A},\vec{y}+\hat{\vec{y}}]) = d$**.**

Solution:

Given that the constraint in the minimization problem from Equation (11) is satisfied, the last column of the matrix $[A+\hat{A},\vec{y}+\hat{\vec{y}}]$. which is $\vec{y}+\hat{\vec{v}}$ can be expressed as a linear combination of the remaining columns in the matrix. Therefore, the rank of the matrix is the rank of the remaining columns, which we previously assumed was d.

(b) We know that the matrix $[A+\hat{A},\vec{y}+\hat{\vec{y}}]$ is degenerate (has less than full rank). Therefore, by the Eckart-Young-Mirsky Theorem which tells us what the closest lower-rank matrix in the Frobenius norm is, the matrix $[A+\hat{A},\vec{y}+\hat{\vec{y}}]$ that minimizes the problem is given by

$$[A,\vec{y}] = U\Sigma V^\top$$
$$[A+\hat{A},\vec{y}+\hat{\vec{y}}] = U \begin{bmatrix} \Sigma_{1,\dots,d} & \\ & 0 \end{bmatrix} V^\top$$

where $\Sigma_{1,..,d}$ is the diagonal matrix of the d largest singular values of $[A,\vec{y}]$. **Using this information, find a nonzero solution to $[A+\hat{A},\vec{y}+\hat{\vec{y}}]\vec{x} = \vec{0}$ and thus solve for $\vec{w}$ in Equation (12).**

HINT: the last column of the product $[A,\vec{y}]V$ will be useful

Solution:

Let's start by expressing the SVD of $[A,\vec{y}]$ in terms of submatrices and vectors:

$$[A,\vec{y}] = \begin{bmatrix} U_{xx} & \vec{u}_{xy} \\ \vec{u}_{xy}^\top & u_{yy} \end{bmatrix} \begin{bmatrix} \Sigma_{1,\dots,m} & \\ & \sigma_{m+1} \end{bmatrix} \begin{bmatrix} V_{xx} & \vec{v}_{xy} \\ \vec{v}_{xy}^\top & v_{yy} \end{bmatrix}^\top$$

and

$$[A,\vec{y}] + [\hat{A},\hat{\vec{y}}] = \begin{bmatrix} U_{xx} & \vec{u}_{xy} \\ \vec{u}_{xy}^\top & u_{yy} \end{bmatrix} \begin{bmatrix} \Sigma_{1,\dots,m} & \\ & 0 \end{bmatrix} \begin{bmatrix} V_{xx} & \vec{v}_{xy} \\ \vec{v}_{xy}^\top & v_{yy} \end{bmatrix}^\top$$

We can subtract the second equation from the first to solve for $[\hat{A},\hat{\vec{y}}]$:

$$
\begin{aligned}
[A,\vec{y}] - [A,\vec{y}] - [\hat{A},\hat{\vec{y}}] &= \begin{bmatrix} U_{xx} & \vec{u}_{xy} \\ \vec{u}_{xy}^\top & u_{yy} \end{bmatrix} \begin{bmatrix} \Sigma_{1,\dots,m} & \\ & \sigma_{m+1} \end{bmatrix} \begin{bmatrix} V_{xx} & \vec{v}_{xy} \\ \vec{v}_{xy}^\top & v_{yy} \end{bmatrix}^\top - \begin{bmatrix} U_{xx} & \vec{u}_{xy} \\ \vec{u}_{xy}^\top & u_{yy} \end{bmatrix} \begin{bmatrix} \Sigma_{1,\dots,m} & \\ & 0 \end{bmatrix} \begin{bmatrix} V_{xx} & \vec{v}_{xy} \\ \vec{v}_{xy}^\top & v_{yy} \end{bmatrix}^\top \\
-[\hat{A},\hat{\vec{y}}] &= \begin{bmatrix} U_{xx} & \vec{u}_{xy} \\ \vec{u}_{xy}^\top & u_{yy} \end{bmatrix} \left(\begin{bmatrix} \Sigma_{1,\dots,m} & \\ & \sigma_{m+1} \end{bmatrix} - \begin{bmatrix} \Sigma_{1,\dots,m} & \\ & 0 \end{bmatrix} \right) \begin{bmatrix} V_{xx} & \vec{v}_{xy} \\ \vec{v}_{xy}^\top & v_{yy} \end{bmatrix}^\top \\
[\hat{A},\hat{\vec{y}}] &= -\begin{bmatrix} U_{xx} & \vec{u}_{xy} \\ \vec{u}_{xy}^\top & u_{yy} \end{bmatrix} \begin{bmatrix} 0 & \\ & \sigma_{m+1} \end{bmatrix} \begin{bmatrix} V_{xx} & \vec{v}_{xy} \\ \vec{v}_{xy}^\top & v_{yy} \end{bmatrix}^\top \\
&= -\begin{bmatrix} \vec{u}_{xy} \\ u_{yy} \end{bmatrix} \sigma_{m+1} \begin{bmatrix} \vec{v}_{xy} \\ v_{yy} \end{bmatrix}^\top
\end{aligned}
$$

From the hint, we then look at the last column of the product $[A,\vec{y}]V$, which is given by

$$
[A,\vec{y}] \begin{bmatrix} \vec{v}_{xy} \\ v_{yy} \end{bmatrix} = \begin{bmatrix} \vec{u}_{xy} \\ u_{yy} \end{bmatrix} \sigma_{m+1}
$$

and we can substitute this into the equation above to get

$$
[\hat{A},\hat{\vec{y}}] = -[A,\vec{y}] \begin{bmatrix} \vec{v}_{xy} \\ v_{yy} \end{bmatrix} \begin{bmatrix} \vec{v}_{xy} \\ v_{yy} \end{bmatrix}^\top
$$

and thus

$$
\begin{aligned}
[A,\vec{y}] + [\hat{A},\hat{\vec{y}}] &= [A,\vec{y}] - [A,\vec{y}] \begin{bmatrix} \vec{v}_{xy} \\ v_{yy} \end{bmatrix} \begin{bmatrix} \vec{v}_{xy} \\ v_{yy} \end{bmatrix}^\top \\
[A+\hat{A},\vec{y}+\hat{\vec{y}}] \begin{bmatrix} \vec{v}_{xy} \\ v_{yy} \end{bmatrix} &= [A,\vec{y}] \begin{bmatrix} \vec{v}_{xy} \\ v_{yy} \end{bmatrix} - [A,\vec{y}] \begin{bmatrix} \vec{v}_{xy} \\ v_{yy} \end{bmatrix} \\
&= \vec{0}
\end{aligned}
$$

which makes $\begin{bmatrix} \vec{v}_{xy} \\ v_{yy} \end{bmatrix}$ a nonzero solution to the equation, and we have $\vec{w} = -\vec{v}_{xy} v_{yy}^{-1}$ from Equation (12).

(c) In this problem, we will use total least squares to approximately learn the lighting in a photograph, which we can then use to paste new objects into the image while still maintaining the realism of the image. You will be estimating the lighting coefficients for the interior of St. Peter's Basillica, and you will then use these coefficients to change the lighting of an image of a tennis ball so that it can be pasted into the image. In Figure 28, we show the result of pasting the tennis ball in the image without adjusting the lighting on the ball. The ball looks too bright for the scene and does not look like it would fit in with other objects in the image.

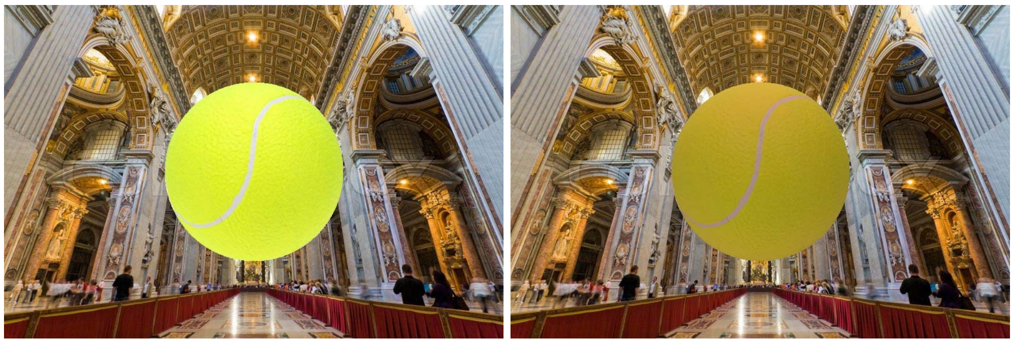


Figure 28: Tennis ball pasted on top of image of St. Peter's Basillica without lighting adjustment (left) and with lighting adjustment (right)

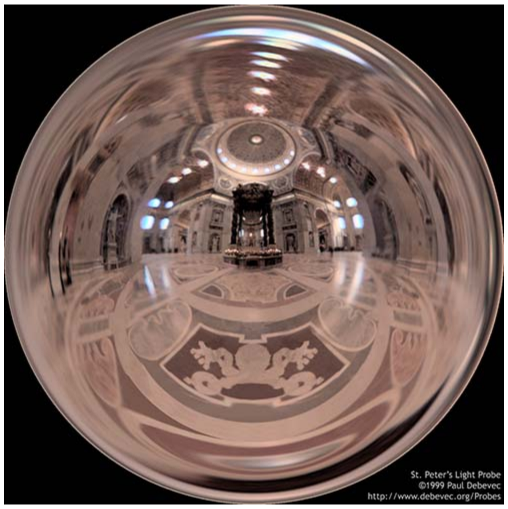


Figure 29: Image of a spherical mirror inside of St. Peter's Basillica

To start, we will represent environment lighting as a spherical function $\vec{f}(\vec{n})$ where $\vec{n}$ is a 3 dimensional unit vector ($||\vec{n}||_2 = 1$), and $\vec{f}$ outputs a 3 dimensional color vector, one component for red, green, and blue light intensities. Because $\vec{f}(\vec{n})$ is a spherical function, the input must correspond to a point on a sphere. In this case, we are using normalized vectors in 3 dimensions to represent a point on a sphere. The function $\vec{f}(\vec{n})$ represents the total incoming light from the direction $\vec{n}$ in the scene.

To convincingly add an object to an image, we need to need to apply the lighting from the environment onto the added object. In this case, we'll be adding a tennis ball to the scene. Because the tennis ball is what's considered a diffuse surface, the color of a point on the ball can be computed as a weighted sum of all of the incoming light at that point that has a positive dot product with the direction $\vec{n}$. This weighted sum can be approximated by blurring the image used to estimate the lighting environment. However, we skip this blurring step and instead approximate the spherical function $\vec{f}(\vec{n})$ in terms of the first 9 spherical harmonic basis functions, where spherical harmonics are like the Fourier transform but for spherical functions. Because we are using only 9 of the basis functions, most of the information in the image cannot be captured, which will effectively blur the image.

The first 9 unormalized basis functions are given by:

$$
\begin{aligned}
L_{0,0} &= 1\\
L_{1,-1} &= y\\
L_{1,1} &= x\\
L_{1,0} &= z\\
L_{2,-2} &= x*y\\
L_{2,-1} &= y*z\\
L_{2,0} &= 3*z^2-1\\
L_{2,1} &= x*z\\
L_{2,2} &= x^2-y^2
\end{aligned}
$$

where $\vec{n} = [x, y, z]^\top$. The lighting function can then be approximated as

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

where $L_i(\vec{n})$ is the ith basis function from the list above.

The function of incoming light $\vec{f}(\vec{n})$ can be measured by photographing a spherical mirror placed in the scene of interest. In this case, we provide you with an image of the sphere as seen in Figure 29. In the code provided, there is a function extractNormals(img) that will extract the training pairs $(\vec{n}_i, \vec{f}(\vec{n}_i))$ from the image. An example using this function is in the code.

Use the spherical harmonic basis functions to create a 9 dimensional feature vector for each sample. Use this to formulate an ordinary least squares problem and solve for the unknown coefficients $\vec{\gamma}_i$. Report the estimated values for $\vec{\gamma}_i$ and include a visualization of the approximation using the provided code. The code provided will load the images, extracts the training data, relights the tennis ball with incorrect coefficients, and saves the results. Your task is to compute the basis functions and solve the least squares problems to provide the code with the correct coefficients. To run the starter code, you will need to use Python with `numpy` and `scipy`. Because the resulting data set is large, we reduce it in the code by taking every 50th entry in the data. This is done for you in the starter code, but you can try using the entire data set or reduce it by a different amount.

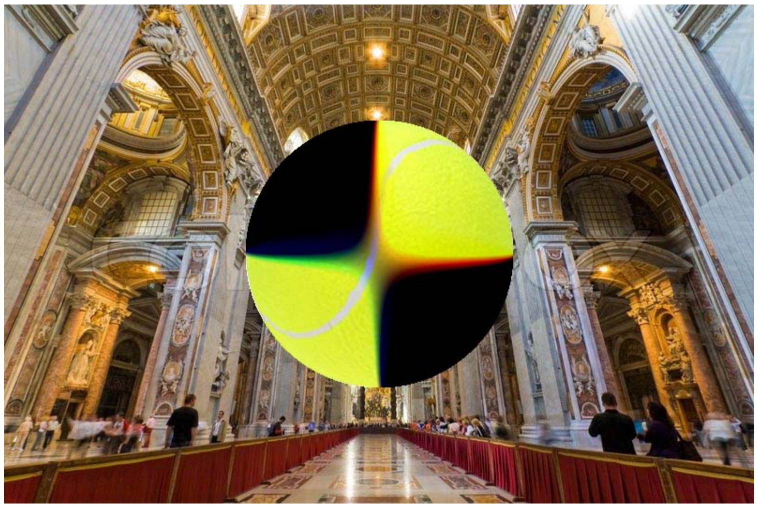
Solution:

```
[ 202.31845431   162.41956802   149.07075034]
[ -27.66555164   -17.88905339   -12.92356688]
[  -1.08629293     0.42947012     1.15475569]
[  -5.15203925    -4.51375871    -4.24262639]
[  -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 ]
```

(d) When we estimate the direction $\vec{n}$ to compute our training data $(\vec{n}_i, \vec{f}(\vec{n}_i))$, we make some approximations about how the light is captured on the image. We also assume that the spherical mirror is a perfect sphere, but in reality, there will always be small imperfections. Thus, our measurement for $\vec{n}$ contains some error, which makes this an ideal problem to apply total least squares. **Solve this problem with total least squares by allowing perturbations in the matrix of basis functions. Report the estimated values for $\vec{\gamma}_i$ and include a visualization of the approximation.** The output image will be visibly wrong, and we'll explore how to fix this problem in the next part. Your implementation may only use the SVD and the matrix inverse functions from the linear algebra library in numpy.

Solution:

```
[   2.13318421e+02    1.70780299e+02    1.57126297e+02]
[  -3.23046362e+01   -2.02975310e+01   -1.45516114e+01]
[  -4.89811386e+00   -3.37684058e+00   -1.14207091e+00]
[  -4.31689131e+00   -3.80778081e+00   -4.83616306e+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]
```



(e) In the previous part, you should have noticed that the visualization is drastically different than the one generated using least squares. This difference is caused by the difference in scale between the inputs and the outputs. The inputs are all unit vectors, and the outputs are 3 dimensional vectors where each value is in $[0,384]$. Typically, values in an image will be in $[0,255]$, but the original image had a much larger range. We compressed the range to a smaller scale using tone mapping, but the effect of the compression is that relatively bright areas of the image become less bright. As a compromise, we scaled the image colors down to a maximum color value of 384 instead of 255. **Explain why the difference in the range of values for the inputs $L_i(\vec{n})$ and the outputs $\vec{f}(\vec{n}_i)$ would create this difference in results when solving the total least squares problem. Propose a value by which to scale the outputs $\vec{f}(\vec{n}_i)$ such that the values of the inputs and outputs are roughly on the same scale. Solve this scaled total least squares problem, report the computed spherical harmonic coefficients and provide a rendering of the relit sphere.**

Solution:

Recall: total least squares assumes that the noise is the same in all directions. When we just have some data, it can be hard to think about what the noise model should be. The default is to take a "significant figures" mentality and assume that everything should be on the same scale, assuming that noise is roughly proportional to the size of the inputs. Because most of the basis functions lie within $[-1,1]$, we want to scale the image pixel values so that they lie in a similar range. For these results, we scaled the values by $1/384$, but any reasonable value that scales the pixel values to a similar range is acceptable.

```
[ 209.41539449   169.06782937   155.39642541]
[ -30.28100667   -20.3163958    -15.21596685]
[  -1.05621451     0.46391495     1.19212042]
[  -5.7563859     -5.08161145    -4.78411908]
[  -7.95607504    -8.25078526    -8.0969764 ]
[  55.29299419    52.93568994    50.39069818]
[  -3.84934062     0.5565465      1.80231959]
[   7.35375998     3.85567243     1.0984583 ]
[ -10.91282516    -6.85792251    -5.88064457]
```

5 Your Own Question

Write your own question, and provide a thorough solution.

Writing your own problems is a very important way to really learn material. The famous "Bloom's Taxonomy" that lists the levels of learning is: Remember, Understand, Apply, Analyze, Evaluate, and Create. Using what you know to create is the top-level. We rarely ask you any HW questions about the lowest level of straight-up remembering, expecting you to be able to do that yourself. (e.g. make yourself flashcards) But we don't want the same to be true about the highest level.

As a practical matter, having some practice at trying to create problems helps you study for exams much better than simply counting on solving existing practice problems. This is because thinking about how to create an interesting problem forces you to really look at the material from the perspective of those who are going to create the exams.

Besides, this is fun. If you want to make a boring problem, go ahead. That is your prerogative. But it is more fun to really engage with the material, discover something interesting, and then come up with a problem that walks others down a journey that lets them share your discovery. You don't have to achieve this every week. But unless you try every week, it probably won't happen ever.