CS 189: Introduction to Machine Learning

Fall 2017

•

Homework 4

•

Solutions by

Jinhong Du

3033483677

Question 1

(a)

Jinhong Du
jaydu@berkeley.edu

(b)

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.

Question 2

(a)

$$\because \quad X_i \overset{iid}{\sim} N(\mu, \Sigma)$$

$$\therefore \quad f_d(x_i) = \frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(x_i-\mu)^T\Sigma^{-1}(x_i-\mu)}$$

$$\therefore \quad f(x_1, \cdots, x_n|\Sigma, \mu) = \prod_{i=1}^{n} f_d(x_i)$$

$$= \frac{1}{(2\pi)^{\frac{nd}{2}}|\Sigma|^{\frac{n}{2}}} e^{-\frac{1}{2}\sum_{i=1}^{n}(x_i-\mu)^T\Sigma^{-1}(x_i-\mu)}$$

$$\therefore \quad \ln f(x_1, \cdots, x_n|\Sigma, \mu) = -\frac{1}{2}\sum_{i=1}^{n}(x_i-\mu)^T\Sigma^{-1}(x_i-\mu) - \frac{nd}{2}\ln(2\pi) - \frac{n}{2}\ln|\Sigma|$$

(b)

Formulas used are given below: $\forall\ x \in \mathrm{R}^d,\ A \in \mathrm{R}^{d\times d},\ A^T = A,$

$$\frac{\partial \ln|A|}{\partial A} = (A^T)^{-1} = A^{-1}$$
$$\frac{\partial x^T Ax}{\partial x} = 2Ax$$
$$\frac{\partial x^T Ax}{\partial A} = xx^T$$

Let

$$L(\mu, \Sigma) = f(x_1, \cdots, x_n|\Sigma, \mu)$$

and

$$\begin{cases} \dfrac{\partial \ln L(\mu,\Sigma)}{\partial \mu} = \sum\limits_{i=1}^{n}\Sigma^{-1}(x_i-\mu) = 0 \\ \dfrac{\partial \ln L(\mu,\Sigma)}{\partial \Sigma} = \frac{1}{2}\sum\limits_{i=1}^{n}(x_i-\mu)(x_i-\mu)^T - \frac{n}{2}\Sigma^{-1} = 0 \end{cases}$$

We get

$$\begin{cases} \hat{\mu} = \frac{1}{n}X_i = \overline{X} \\ \hat{\Sigma} = \frac{1}{n}\sum\limits_{i=1}^{n}(X_i-\overline{X})(X_i-\overline{X})^T = S^2 \end{cases}$$

i.e. $\hat{\mu}$ is the mean of random sample and $\hat{\Sigma}$ is the biased covariance matrix of random sample.

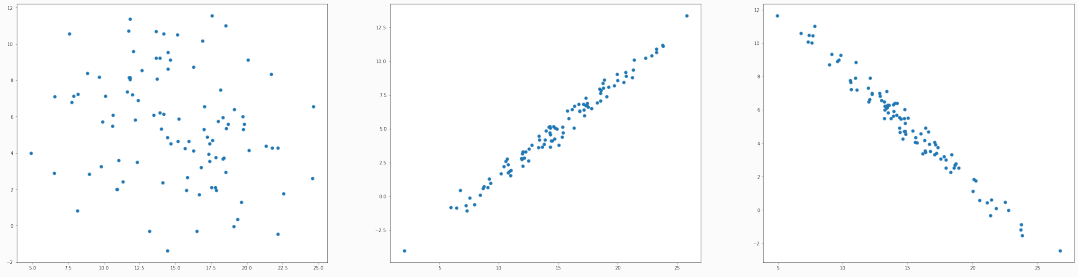
(c)

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
mu = [15, 5]
```

Solution (cont.)

```
        Sigma = [[[20,0],[0,10]],
            [[20,14],[14,10]],
            [[20,-14],[-14,10]]]
        plt.figure(figsize=(40,10))
        samples = [[],[],[]]
        sample_mu = np.zeros((3,2))
        sample_sigma = np.zeros((3,2,2))
        for i in range(len(Sigma)):
            plt.subplot(1,3,i+1)
            samples[i] = np.random.multivariate_normal(mu, Sigma[i],
              size=100)
            plt.scatter(samples[i][:, 0], samples[i][:, 1])
            sample_mu[i] = np.mean(samples[i],axis=0)
            sample_sigma[i] = np.array(samples[i]-sample_mu[i]).T.dot(
              np.array(samples[i]-sample_mu[i]))/len(samples[i])
            print('----------Sample %s----------'%(i+1))
            print('Sample mean:\n',sample_mu[i])
            print('Sample covariacne:\n',sample_sigma[i])
        plt.show()
```

```
----------Sample 1------------
Sample mean:
 [ 15.03926032   5.464707  ]
Sample covariacne:
 [[ 17.65778843  -2.64622112]
 [ -2.64622112   9.0574125 ]]
----------Sample 2------------
Sample mean:
 [ 14.92167895   4.99235006]
Sample covariacne:
 [[ 21.18559615  14.91894818]
 [ 14.91894818  10.72089487]]
----------Sample 3------------
Sample mean:
 [ 15.00551355   5.02482564]
Sample covariacne:
 [[ 17.33778218 -12.05337665]
 [-12.05337665   8.60016821]]
```



Question 3

(a)

$\because$

$$Y = w^T X + z,$$

where $Y, z \in \mathbb{R},\ X, w \in \mathbb{R}^d$ and $z \sim N(0,1), w \sim (0, \Sigma)$, w and z are independent

$\therefore$

$$\begin{aligned} f_{Y|X,w}(y|x, w_0) &= f_z(y - w_0^T x) \\ &= \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(y - w_0^T x)^2} \end{aligned}$$

$\therefore \quad Y|X = x, w = w_0 \sim N(w_0^T x, 1)$

(b)

Let $Z = \begin{pmatrix} z_1 & z_2 & \cdots & z_n \end{pmatrix}^T,\ Y = \begin{pmatrix} Y_1 & Y_2 & \cdots & Y_n \end{pmatrix}^T,\ X = \begin{pmatrix} X_1^T & X_2^T & \cdots & X_n^T \end{pmatrix}^T,$
$Z \sim N(0, I_{n\times n}),\ w \sim N(0, \Sigma)$

$\because$

$$Y_i = X_i^T w + z_i$$

$\therefore$

$$Y = Xw + Z$$

$\because$

$$\begin{aligned} f_{Y|X,w}(y; x, w) &= \frac{1}{(2\pi)^{\frac{n}{2}}} e^{-\frac{1}{2}(y - xw)^T (y - xw)} \\ f_{w|X}(w; x) &= f_w(w) \\ &= \frac{1}{(2\pi)^{\frac{d}{2}} |\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2} w^T \Sigma^{-1} w} \end{aligned}$$

$\therefore$

$$f_{Y|X,w}(y; x, w) f_{w|X}(w; x) = \frac{1}{(2\pi)^{\frac{d+n}{2}} |\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}[(y - xw)^T (y - xw) + w^T \Sigma^{-1} w]}$$

$\therefore$

$$\begin{aligned} \int_w f_{Y|X,w}(y; x, w) f_{w|X}(w; x) \mathrm{d}w &= \int_w \frac{1}{(2\pi)^{\frac{d+n}{2}} |\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}[(y - xw)^T (y - xw) + w^T \Sigma^{-1} w]} \mathrm{d}w \\ &= \frac{1}{(2\pi)^{\frac{d+n}{2}} |\Sigma|^{\frac{1}{2}}} \int_w e^{-\frac{1}{2} w'^T (x^T x + \Sigma^{-1}) w' + \frac{1}{2} y^T x [(x^T x) + \Sigma^{-1}]^{-1} x^T y - \frac{1}{2} y^T y} \mathrm{d}w \\ &= \frac{|[(x^T x) + \Sigma^{-1}]^{-1}|^{\frac{1}{2}}}{(2\pi)^{\frac{1}{2}} |\Sigma|^{\frac{1}{2}}} e^{\frac{1}{2} y^T x [(x^T x) + \Sigma^{-1}]^{-1} x^T y - \frac{1}{2} y^T y} \end{aligned}$$

where $w' = w - (x^T x + \Sigma^{-1})^{-1} x^T y$

$\therefore$ given $X = x,\ Y = y,$

$$\begin{aligned} f_{W|X,Y}(w; x, y) &= \frac{f_{Y|X,w}(y; x, w) f_{w|X}(w; x)}{\int_w f_{Y|X,w}(y; x, w) f_{w|X}(w; x) \mathrm{d}w} \\ &= \frac{1}{(2\pi)^{\frac{d}{2}} |[(x^T x) + \Sigma^{-1}]^{-1}|^{\frac{1}{2}}} e^{-\frac{1}{2} w'^T (x^T x + \Sigma^{-1}) w'} \end{aligned}$$

Solution (cont.)

i.e.

$$W|X=x,Y=y\sim N((x^Tx+\Sigma^{-1})^{-1}x^Ty,(x^Tx+\Sigma^{-1})^{-1})$$

The posterior estimator of w is given by

$$w_{posterior}=(x^Tx+\Sigma^{-1})^{-1}x^Ty$$

(c)

My way:

Let $Z=\Sigma_z^{\frac{1}{2}}Z'+\mu_z$, then $Z'\sim N(0,1)$. Then $Y=Xw+Z$ implies that

$$\Sigma_z^{-\frac{1}{2}}(Y-Xw-\mu_z)=Z'$$

Let $y'=\Sigma_z^{-\frac{1}{2}}y-\Sigma_z^{-\frac{1}{2}}\mu_z$, $X'=\Sigma_z^{-1}X$, then $Z'=Y'-X'w$

Therefore from (b) we have

$$\begin{aligned}f_{W|X,Y}(w;x,y)&=\frac{1}{(2\pi)^{\frac{d}{2}}|(x'^Tx'+\Sigma^{-1})^{-1}|^{\frac{1}{2}}}e^{-\frac{1}{2}w''^T(x'^Tx'+\Sigma^{-1})w''}\\&=\frac{1}{(2\pi)^{\frac{d}{2}}|(x^T\Sigma_z^{-1}x+\Sigma^{-1})^{-1}|^{\frac{1}{2}}}e^{-\frac{1}{2}w''^T(x^T\Sigma_z^{-1}x+\Sigma^{-1})w''}\end{aligned}$$

where $w''=w-(x^T\Sigma_z^{-1}x+\Sigma^{-1})^{-1}x^T\Sigma_z^{-1}(y-\mu_z)$

i.e.

$$W|X=x,Y=y\sim N((x^T\Sigma_z^{-1}x+\Sigma^{-1})^{-1}x^T\Sigma_z^{-1}(y-\mu_z),(x^T\Sigma_z^{-1}x+\Sigma^{-1})^{-1})$$

Changing Coordinates:

We have

$$\begin{bmatrix}Y-\mu_z\\w\end{bmatrix}=\begin{bmatrix}\Sigma_z^{\frac{1}{2}}&X\Sigma^{\frac{1}{2}}\\0&\Sigma^{\frac{1}{2}}\end{bmatrix}\begin{bmatrix}u\\v\end{bmatrix}$$

where $u,v\overset{iid}{\sim}N(0,1)$. We wish

$$\begin{bmatrix}w\\Y-\mu_z\end{bmatrix}=\begin{bmatrix}A&B\\0&C\end{bmatrix}\begin{bmatrix}u'\\v'\end{bmatrix}$$

then we have $v'=D^{-1}Y$

$$w|X,Y'\sim N(BD^{-1}Y',AA^T)$$

$$Y'=Y-\mu_z$$

$\because$

$$\begin{aligned}E\left[\begin{bmatrix}w\\Y'\end{bmatrix}\begin{bmatrix}w^T&Y'^T\end{bmatrix}\right]&=\begin{bmatrix}AA^T+BB^T&BD^T\\DB^T&DD^T\end{bmatrix}\\&=\begin{bmatrix}\Sigma_w&\Sigma_{wY'}\\\Sigma_{Y'w}&\Sigma_{Y'}\end{bmatrix}\\E\left[\begin{bmatrix}Y'\\w\end{bmatrix}\begin{bmatrix}Y'^T&w^T\end{bmatrix}\right]&=\begin{bmatrix}\Sigma_z+X\Sigma X^T&X\Sigma\\\Sigma X^T&\Sigma\end{bmatrix}\\&=\begin{bmatrix}\Sigma_{Y'}&\Sigma_{wY'}\\\Sigma_{Y'w}&\Sigma_w\end{bmatrix}\end{aligned}$$

Solution (cont.)

$\therefore$

$$\begin{aligned} BD^{-1} &= BD^T D^{-T} D^{-1} \\ &= \Sigma_{wY'} \Sigma_{Y'}^{-1} \\ &= \Sigma X^T (\Sigma_z + X\Sigma X^T)^{-1} \\ AA^T &= AA^T + BB^T - (BD^T)(DD^T)^{-1} DB^T \\ &= \Sigma - \Sigma\Sigma_{wY'}\Sigma_{Y'}^{-1}\Sigma_{Y'w} \\ &= \Sigma - \Sigma X^T (\Sigma_z + X\Sigma X^T)^{-1} X\Sigma \end{aligned}$$

$\therefore$ from the Woodbury Matrix Identity, we have

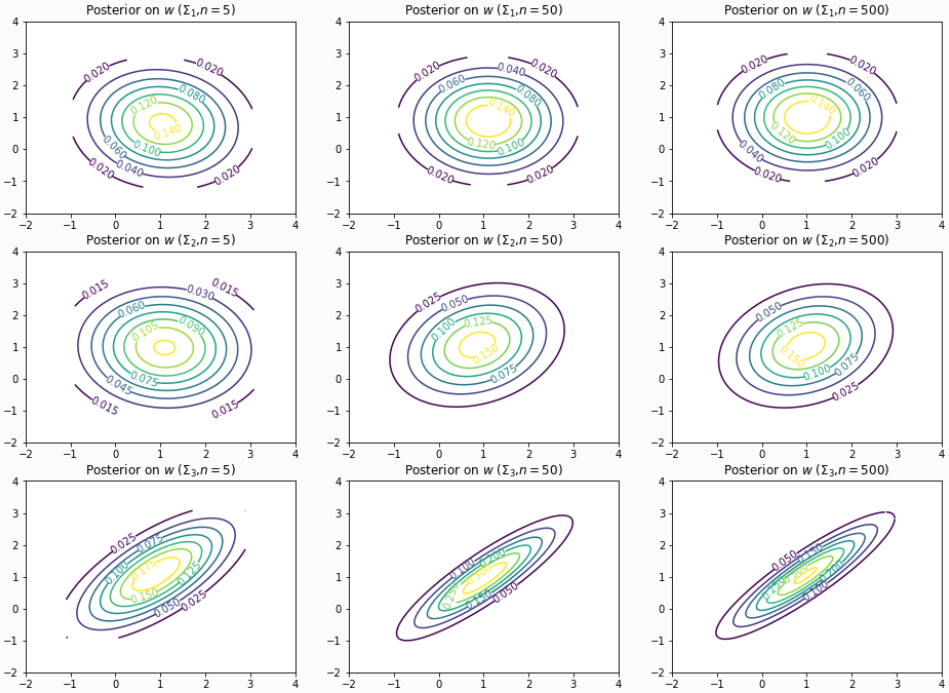
$$\begin{aligned} BD^{-1} &= (X^T\Sigma_z X + \Sigma^{-1})^{-1} X^T \Sigma_z^{-1} \\ AA^T &= (X^T\Sigma_z^{-1} X + \Sigma^{-1})^{-1} \end{aligned}$$

$\therefore$

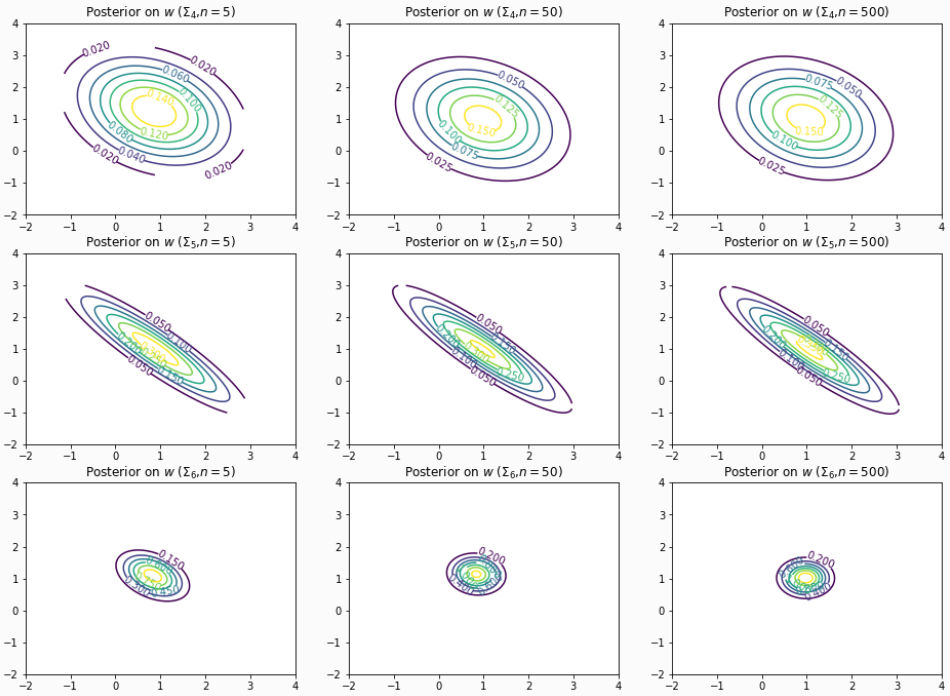
$$W|X = x, Y = y \sim N((x^T\Sigma_z^{-1}x + \Sigma^{-1})^{-1} x^T\Sigma_z^{-1}(y - \mu_z), (x^T\Sigma_z^{-1}x + \Sigma^{-1})^{-1})$$

(d)

We can see that when increse the data points, the posterior of w given X, Y seems to become stable. And high correlation between w_1 and w_2 given X, Y leads to a elliptical-shape distribution. Little varainces of w_1 and w_2 leads to a intensive conditional distribution.



Solution (cont.)



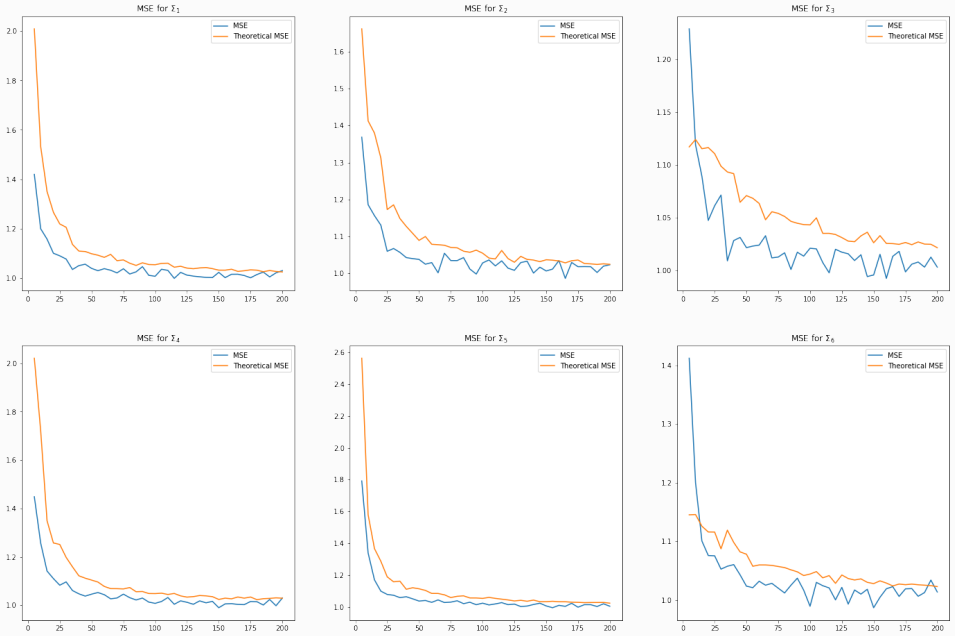

```
import matplotlib.mlab as mlab

Size = [5, 50, 500]
Sigma = [[[1, 0], [0, 1]],
         [[1, 0.25], [0.25, 1]],
         [[1, 0.9], [0.9, 1]],
         [[1, -0.25], [-0.25, 1]],
         [[1, -0.9], [-0.9, 1]],
         [[0.1, 0], [0, 0.1]]
        ]

plt.figure(figsize=(16,24))
for i in range(len(Sigma)):
  for j in range(len(Size)):
        plt.subplot(6,3,i*3+j+1)
        z = np.random.normal(size=Size[j])
        samples = np.random.multivariate_normal([5, 5], Sigma[0],
          size=Size[j])
        x1 = samples[:,0]
        x2 = samples[:,1]
        y = x1+x2+z
        mu = np.linalg.inv(samples.T.dot(samples)+
          np.linalg.inv(Sigma[i])).dot(samples.T).dot(y)
        sigma = np.linalg.inv(samples.T.dot(samples))+Sigma[i]
```

Solution (cont.)

```
        x_axis = np.arange(mu[0]-2.0, mu[0]+2.0, 0.01)
        y_axis = np.arange(mu[1]-2.0, mu[1]+2.0, 0.01)
        X, Y = np.meshgrid(x_axis, y_axis)
        w = mlab.bivariate_normal(X, Y, np.sqrt(sigma[0][0]),
          np.sqrt(sigma[1][1]), mu[0], mu[1], sigma[0][1])
        CS = plt.contour(X, Y, w)
        plt.clabel(CS, inline=1, fontsize=10)
        plt.xlim(-2,4)
        plt.ylim(-2,4)
        plt.title('Posterior on $w$ ($\Sigma_%i$, $n=%i$)'
          %(i+1,Size[j]))
plt.show()
```

(e)

As we can see when number of data increases, both MSE and theoretical MSE go down. And for bad priors like Σ_3, Σ_6(because we know that w_1 and w_2 are moderately and positive correlated), the actual MSE is bigger than theoretical MSE at first. For good priors like Σ_1 and Σ_2, the actual MSE is more smaller than the others at first. Actually, both actual MSE and theoretical MSE are influenced by the priors at fisrt.




```
import matplotlib.mlab as mlab

Size = [5, 50, 500]
Sigma = [[[1, 0], [0, 1]],
         [[1, 0.25], [0.25, 1]],
```

Solution (cont.)

```
        [[1, 0.9], [0.9, 1]],
        [[1, -0.25], [-0.25, 1]],
        [[1, -0.9], [-0.9, 1]],
        [[0.1, 0], [0, 0.1]]
       ]

Rmean = np.zeros((len(Sigma),40))
the_error = np.zeros((len(Sigma),40))
for i in range(len(Sigma)):
  for j in range(5,205,5):
      for _ in range(200):
          z = np.random.normal(size=j+100)
          samples = np.random.multivariate_normal([5, 5], Sigma[0],
            size=j+100)
          train = samples[:-100,:]
          test = samples[-100:,:]
          y = np.sum(train,axis=1)+z[:-100]
          y_test = np.sum(test,axis=1)+z[-100:]
          what = np.linalg.inv(train.T.dot(train)
            +np.linalg.inv(Sigma[i])).dot(train.T).dot(y)
          yhat = test.dot(what)
          Rmean[i][(j-5)//5] += np.sum((yhat-y_test)**2)/100
          the_error[i][(j-5)//5] += 5*(what[0]-1)**2
            +5*(what[1]-1)**2+1
      Rmean[i][(j-5)//5] /= 200
      the_error[i][(j-5)//5] /= 200

plt.figure(figsize=(24,16))
for i in range(len(Sigma)):
    plt.subplot(2,3,i+1)
    plt.plot(range(5,205,5),Rmean[i],label='MSE')
    plt.plot(range(5,205,5),the_error[i],label='Theoretical MSE')
    plt.title('MSE for $\Sigma_%i$'%(i+1))
    plt.legend()
plt.show()
```

Question 4

(a)

$\because$

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

$\therefore$

$$d \leqslant rank([A + \hat{A}, \vec{y} + \hat{\vec{y}}]) \leqslant d + 1$$

$\because$

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

i.e. $\vec{y} + \hat{\vec{y}} \in Range(A + \hat{A})$

$\therefore$

$$rank([A + \hat{A}, \vec{y} + \hat{\vec{y}}]) < d + 1$$

$\therefore$

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

(b)

Define SVD

$$\begin{bmatrix} A & \vec{y} \end{bmatrix} = U\Sigma V^T$$

From Eckart-Young-Mirsky Theorem, the closest lower-rank matrix in the Forbenius norm is

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

$\because$

$$\begin{bmatrix} A + \hat{A} & \vec{y} + \hat{\vec{y}} \end{bmatrix} = U \begin{bmatrix} \Sigma_{1\cdots d} & \vec{0} \\ \vec{0}^T & 0 \end{bmatrix} V^T$$

$$\begin{bmatrix} A & \vec{y} \end{bmatrix} = U \begin{bmatrix} \Sigma_{1\cdots d} & \vec{0} \\ \vec{0}^T & \Sigma_{d+1} \end{bmatrix} V^T$$

$\therefore$

$$\begin{aligned} \begin{bmatrix} \hat{A} & \hat{\vec{y}} \end{bmatrix} &= \begin{bmatrix} A + \hat{A} & \vec{y} + \hat{\vec{y}} \end{bmatrix} - \begin{bmatrix} A & \vec{y} \end{bmatrix} \\ &= -U \begin{bmatrix} \mathbf{0} & \vec{0} \\ \vec{0}^T & \Sigma_{d+1} \end{bmatrix} V^T \\ &= -\begin{bmatrix} U_1 & U_2 \end{bmatrix} \begin{bmatrix} \mathbf{0} & \vec{0} \\ \vec{0}^T & \Sigma_{d+1} \end{bmatrix} \begin{bmatrix} V_{11} & V_{12} \\ V_{21} & V_{22} \end{bmatrix}^T \\ &= -U_2\Sigma_{d+1} \begin{bmatrix} V_{12} \\ V_{22} \end{bmatrix}^T \end{aligned}$$

$\because$

$$U_2\Sigma_{d+1} = \begin{bmatrix} A & \vec{y} \end{bmatrix} \begin{bmatrix} V_{12} \\ V_{22} \end{bmatrix}$$

Solution (cont.)

$\therefore$

$$\begin{bmatrix}\hat{A} & \hat{\vec{y}}\end{bmatrix} = -\begin{bmatrix}A & \vec{y}\end{bmatrix}\begin{bmatrix}V_{12}\\V_{22}\end{bmatrix}\begin{bmatrix}V_{12}\\V_{22}\end{bmatrix}^T$$

$\because$ V is orthonromal

$\therefore$ $\begin{bmatrix}V_{12}\\V_{22}\end{bmatrix}^T\begin{bmatrix}V_{12}\\V_{22}\end{bmatrix} = 1$

$\therefore$

$$\begin{aligned}\begin{bmatrix}A+\hat{A} & \vec{y}+\hat{\vec{y}}\end{bmatrix}\begin{bmatrix}V_{12}\\V_{22}\end{bmatrix} &= \begin{bmatrix}A & \vec{y}\end{bmatrix}\begin{bmatrix}V_{12}\\V_{22}\end{bmatrix} + \begin{bmatrix}\hat{A} & \hat{\vec{y}}\end{bmatrix}\begin{bmatrix}V_{12}\\V_{22}\end{bmatrix}\\ &= \begin{bmatrix}A & \vec{y}\end{bmatrix}\begin{bmatrix}V_{12}\\V_{22}\end{bmatrix} - \begin{bmatrix}A & \vec{y}\end{bmatrix}\begin{bmatrix}V_{12}\\V_{22}\end{bmatrix}\begin{bmatrix}V_{12}\\V_{22}\end{bmatrix}^T\begin{bmatrix}V_{12}\\V_{22}\end{bmatrix}\\ &= 0\end{aligned}$$

$\therefore$

$$\begin{bmatrix}A+\hat{A} & \vec{y}+\hat{\vec{y}}\end{bmatrix}\begin{bmatrix}-V_{12}V_{22}^{-1}\\-V_{22}V_{22}^{-1}\end{bmatrix} = 0$$

$\therefore$

$$\begin{bmatrix}A+\hat{A} & \vec{y}+\hat{\vec{y}}\end{bmatrix}\begin{bmatrix}-V_{12}V_{22}^{-1}\\-I\end{bmatrix} = 0$$

$\therefore$ $\hat{w} = -V_{12}V_{22}^{-1}$

(c)

Let $A = \begin{bmatrix}L_1(\vec{n}) & L_2(\vec{n}) & \cdots & L_9(\vec{n})\end{bmatrix}$, $\vec{y} = \vec{f}(\vec{n})$, $\vec{w} = \begin{bmatrix}\vec{\gamma_1} & \vec{\gamma_2} & \cdots & \vec{\gamma_9}\end{bmatrix}$ then the problem becomes seeking for

$$\hat{\vec{w}} = \arg\min_{\vec{w}} \|A\vec{w} - \vec{y}\|^2$$

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.misc import imread,imsave

imFile = 'hw04-data/stpeters_probe_small.png'
compositeFile = 'hw04-data/tennis.png'
targetFile = 'hw04-data/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():
```

Solution (cont.)

```
    imFile = 'hw04-data/stpeters_probe_small.png'
    compositeFile = 'hw04-data/tennis.png'
    targetFile = 'hw04-data/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)
```

Solution (cont.)

```
57              vs.append(img[i,j])
58
59      return np.asarray(ns), np.asarray(vs)
60
61  # This function renders a diffuse sphere of radius r
62  # using the spherical harmonic coefficients given in
63  # the input coeff where coeff is a 9 x c matrix
64  # with c being the number of color channels
65  # The output is an 2r x 2r x c image of a diffuse sphere
66  # and the value of -1 on the image where there is no sphere
67  def renderSphere(r,coeff):
68
69      d = 2*r
70      img = -np.ones((d,d,3))
71      ns = []
72      ps = []
73
74      for i in range(d):
75          for j in range(d):
76
77              # Determine if the pixel is on the sphere
78              x = j - r
79              y = i - r
80              if x*x + y*y > r*r:
81                  continue
82
83              # Figure out the normal vector at the point
84              # We assume that the image is an orthographic projection
85              z = np.sqrt(r*r-x*x-y*y)
86              n = np.asarray([x,y,z])
87              n = n / np.sqrt(np.sum(np.square(n)))
88              ns.append(n)
89              ps.append((i,j))
90
91      ns = np.asarray(ns)
92      B = computeBasis(ns)
93      vs = B.dot(coeff)
94
95      for p,v in zip(ps,vs):
96          img[p[0],p[1]] = np.clip(v,0,255)
97
98      return img
99
```

Solution (cont.)

```
# relights the sphere in img, which is assumed to be a square image
# coeff is the matrix of spherical harmonic coefficients
def relightSphere(img, coeff):
  img = renderSphere(int(img.shape[0]/2),coeff)/255*img/255
  return img

# Copies the image of source onto target
# pixels with values of -1 in source will not be copied
def compositeImages(source, target):

  # Assumes that all pixels not equal to 0 should be copied
  out = target.copy()
  cx = int(target.shape[1]/2)
  cy = int(target.shape[0]/2)
  sx = cx - int(source.shape[1]/2)
  sy = cy - int(source.shape[0]/2)

  for i in range(source.shape[0]):
      for j in range(source.shape[1]):
          if np.sum(source[i,j]) >= 0:
              out[sy+i,sx+j] = source[i,j]

  return out

  # Fill in this function to compute the basis functions
  # This function is used in renderSphere()
  def computeBasis(ns):
      # Returns the first 9 spherical harmonic basis functions

      ##########################################################
      # Compute the first 9 basis functions
      ##########################################################
      B = np.ones((len(ns),9)) # This line is here just to fill space
      B[:,1] = ns[:,1]
      B[:,2] = ns[:,0]
      B[:,3] = ns[:,2]
      B[:,4] = np.multiply(ns[:,0],ns[:,1])
      B[:,5] = np.multiply(ns[:,1],ns[:,2])
      B[:,6] = 3*np.multiply(ns[:,2],ns[:,2])-1
      B[:,7] = np.multiply(ns[:,0],ns[:,2])
      B[:,8] = np.multiply(ns[:,0],ns[:,0])-np.multiply(ns[:,1],
        ns[:,1])

```

Solution (cont.)

```
143         return B
144
145     def regreession_multi(X,y,lamb=0):
146         n1, n2 = np.shape(X)
147         A = np.linalg.inv(X.T.dot(X)+lamb*np.identity(n2)
148         ).dot(X.T.dot(y))
149         yhat = X.dot(A)
150
151         Rmean =  np.linalg.norm(yhat-y)**2/n1
152
153         return {'A':A,'train_error':Rmean}
154
155     data,tennis,target = loadImages()
156     ns, vs = extractNormals(data)
157     B = computeBasis(ns)
158     # reduce the number of samples because computing the SVD on
159     # the entire data set takes too long
160     Bp = B[::50]
161     vsp = vs[::50]
162
163     data,tennis,target = loadImages()
164     ns, vs = extractNormals(data)
165     B = computeBasis(ns)
166
167     # reduce the number of samples because computing the SVD on
168     # the entire data set takes too long
169     Bp = B[::50]
170     vsp = vs[::50]
171
172     ########################################################
173     # Solve for the coefficients using least squares
174     # or total least squares here
175     coeff = regreession_multi(Bp,vsp)['A']
176
177     img = relightSphere(tennis,coeff)
178
179     output = compositeImages(img,target)
180
181     print('Coefficients:\n'+str(coeff))
182
183     plt.figure(1)
184     plt.imshow(output)
185     plt.show()
```

Solution (cont.)

```
186
187     imsave('hw04-data/output-ols.png',output)
```

```
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 ]]
```

(d)

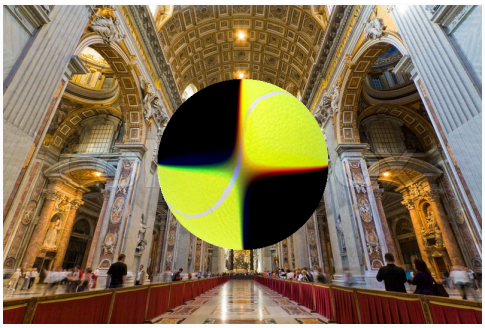
From 4(b) and (c), we have the solution to TLS is $-V_{12}V_{22}^{-1}$

```
1  A = np.concatenate((Bp,vsp),axis=1)
2  _,_,vt = np.linalg.svd(A)
3  v = vt.T
4  _,n = np.shape(vsp)
5
6  coeff = -v[:-3,-3:].dot(np.linalg.inv(v[-3:,-3:]))
7
8
9  img = relightSphere(tennis,coeff)
10
11 output = compositeImages(img,target)
12
13 print('Coefficients:\n'+str(coeff))
14
15 plt.figure(1)
16 plt.imshow(output)
17 plt.show()
18
```

Solution (cont.)

```
imsave('hw04-data/output-tls.png',output)
```

```
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]]
```



(e)

The difference in the range of values for the inputs and the outputs would create this difference in results when solving the total least squares problem because OLS doesn't have scale-invariance property.
It is because that our estimation comes from the assumed error term from true model is

$$U \begin{bmatrix} \mathbf{0} & \vec{0} \\ \vec{0}^T & \Sigma_{d+1} \end{bmatrix} V^T$$

when $y' = cy$, here y is the standarlized outputs and c is a positive constant, then the assume error term becomes

$$U' \begin{bmatrix} \mathbf{0} & \vec{0} \\ \vec{0}^T & \Sigma'_{d+1} \end{bmatrix} V'^T$$

because multiply only some column of $\begin{bmatrix} A & \vec{y} \end{bmatrix}$ with c may change the egenvalues as well as the egenvectors.

```
A = np.concatenate((Bp,vsp),axis=1)
_,_,vt = np.linalg.svd(A)
v = vt.T
_,n = np.shape(vsp)

coeff = -v[:-3,-3:].dot(np.linalg.inv(v[-3:,-3:]))

img = relightSphere(tennis,coeff*384)

output = compositeImages(img,target)
```

Solution (cont.)

```
11
12  print('Coefficients:\n'+str(coeff))
13
14  plt.figure(1)
15  plt.imshow(output)
16  plt.show()
17
18  imsave('hw04-data/output-tls-2.png',output)
```

```
Coefficients:
[[ 0.54526595  0.44019965  0.40460097]
 [-0.07882306 -0.05287614 -0.03959563]
 [-0.01498285 -0.01322608 -0.01245169]
 [-0.0027508   0.00120776  0.00310405]
 [-0.02058775 -0.02136242 -0.02096713]
 [ 0.14313155  0.13704163  0.13044962]
 [-0.01002394  0.00144957  0.00469367]
 [ 0.01907957  0.00997563  0.00279948]
 [-0.02840275 -0.01784431 -0.01530017]]
```

Question 5

Question What is the connection and difference among MLE, MAP and Bayes Estimate?

Solution

Maximum Likelihood Estimate (MLE)

$$\hat{\theta}_{MLE} = \arg\min_{\theta} P(X|\theta)$$

Maximum A Posterior Estimate (MAP)

$$\begin{aligned}\hat{\theta}_{MAP} &= \arg\min_{\theta} P(\theta|X) \\ &= \arg\min_{\theta} \frac{P(X|\theta)P(\theta)}{P(X)} \\ &= \arg\min_{\theta} (\log P(X|\theta) + \log P(\theta))\end{aligned}$$

consider the prior distribution of θ.

Bayes Estimate

$$P(\theta|X) = \frac{P(X\theta)P(\theta)}{\int_{\theta} P(X|\theta)P(\theta)\mathrm{d}\theta}$$

Here MLE and MAP are both point estimation, i.e., they only give an estimation of parameter θ. However, Bayes Estimate is generalized from MAP and it can give us a estiamted distribution of θ.

The difference between MLE and MAP is that: MLE supposes that θ is a unknown constant while MAP supposes that θ is a unknown random variable and its distribution needs to be estimated. In other words, MLE is a prior estimate method while MAP is a posterior estimate method. Or we can consider MLE as a special case of MAP, that means we suppose that $P(\theta) = 1$, i.e. $\theta = constant\ a.e.$.

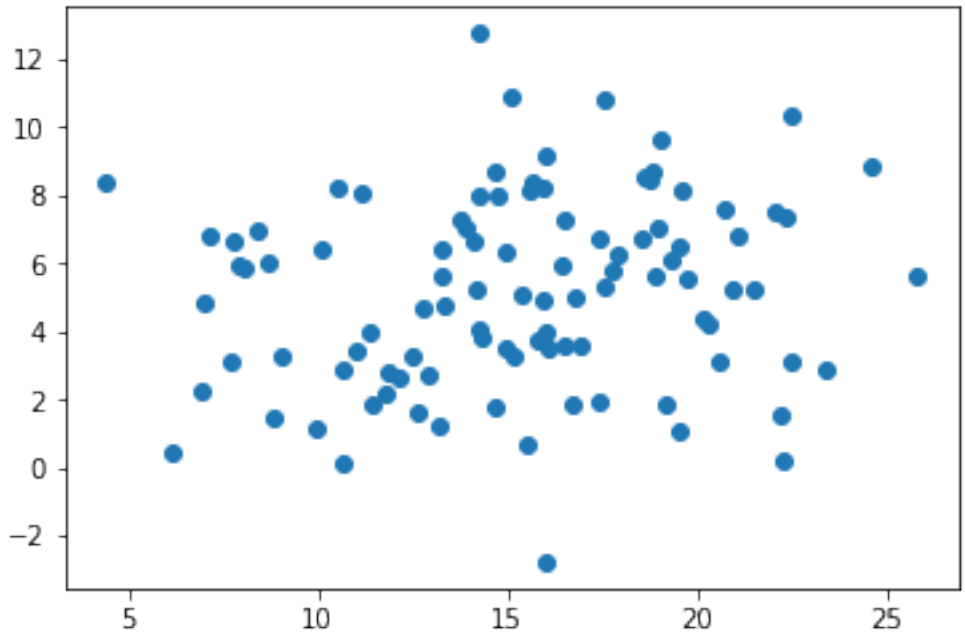
Therefore, I think the relationship among them should be : MLE $\subset$ MAP $\subset$ Bayes Estimate

HW4

September 22, 2017

1 Question 2

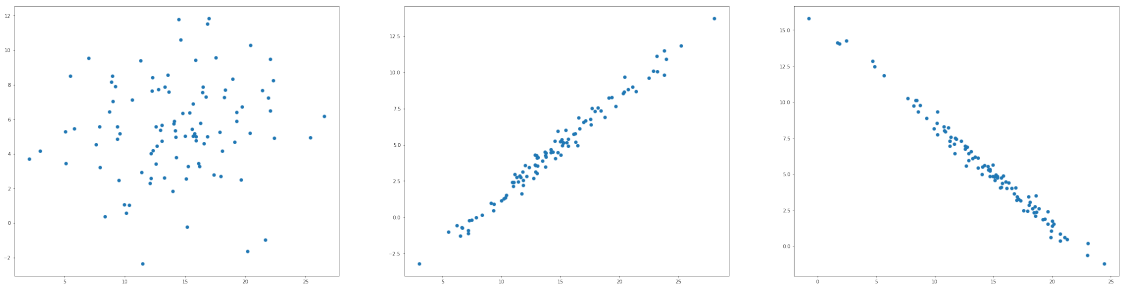
```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        %matplotlib inline
        mu = [15, 5]
        sigma = [[20, 0], [0, 10]]
        samples = np.random.multivariate_normal(mu, sigma, size=100)
        plt.scatter(samples[:, 0], samples[:, 1])
        plt.show()
```




```
In [2]: Sigma = [[[20,0],[0,10]],
                 [[20,14],[14,10]],
                 [[20,-14],[-14,10]]]
        plt.figure(figsize=(40,10))
```

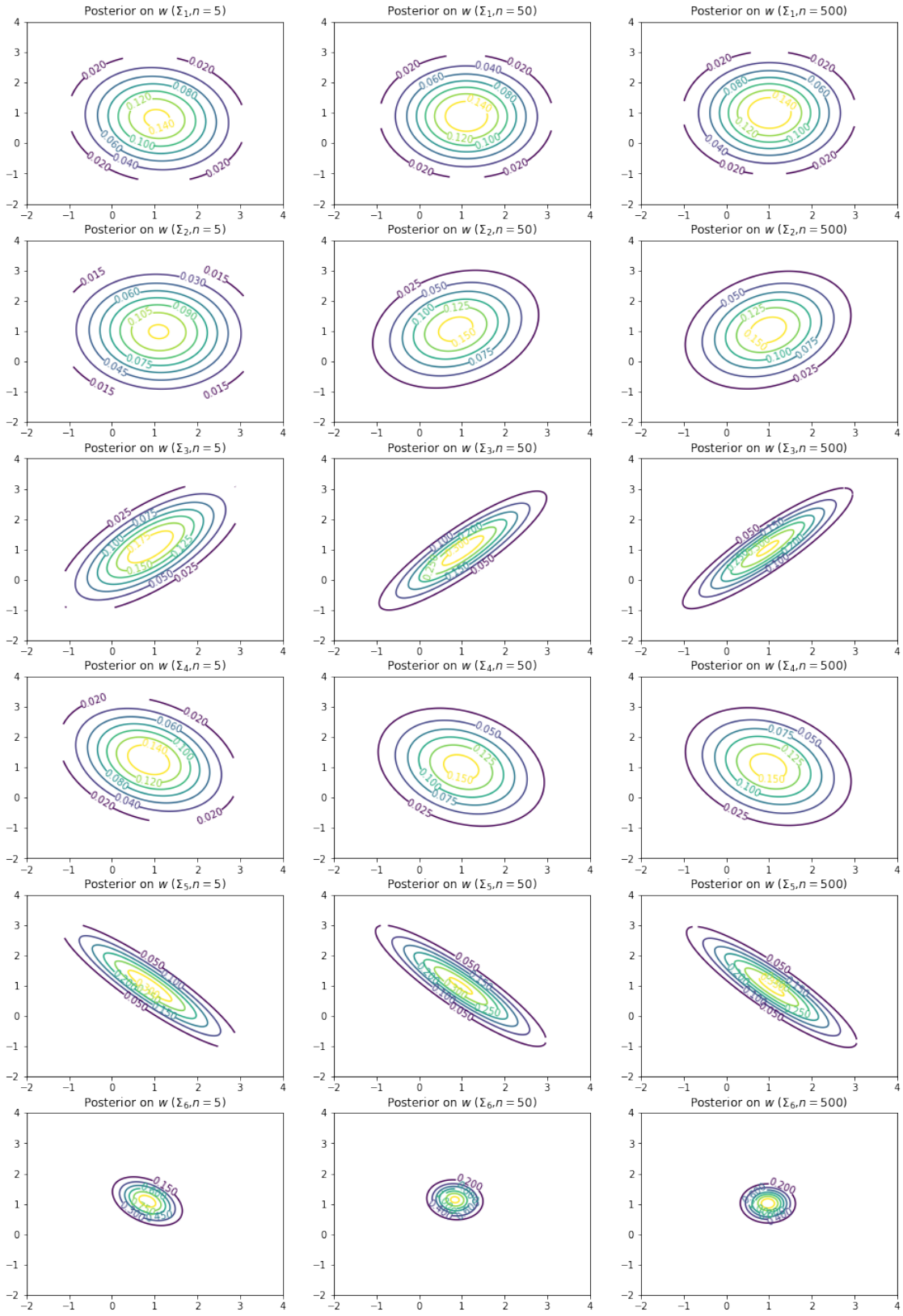
```
        samples = [[],[],[]]
        sample_mu = np.zeros((3,2))
        sample_sigma = np.zeros((3,2,2))
        for i in range(len(Sigma)):
            plt.subplot(1,3,i+1)
            samples[i] = np.random.multivariate_normal(mu, Sigma[i], size=100)
            plt.scatter(samples[i][:, 0], samples[i][:, 1])
            sample_mu[i] = np.mean(samples[i],axis=0)
            sample_sigma[i] = np.array(samples[i]-sample_mu[i]).T.dot(np.array(samples[i]-sample
            print('----------Sample %s------------'%(i+1))
            print('Sample mean:\n',sample_mu[i])
            print('Sample covariacne:\n',sample_sigma[i])
        plt.show()
```

```
----------Sample 1------------
Sample mean:
 [ 14.28787044   5.47782083]
Sample covariacne:
 [[ 22.95292513   1.65803728]
 [  1.65803728   7.85644204]]
----------Sample 2------------
Sample mean:
 [ 14.46595395   4.5781421 ]
Sample covariacne:
 [[ 23.6030086   16.11194783]
 [ 16.11194783  11.20843844]]
----------Sample 3------------
Sample mean:
 [ 14.31014025   5.47617638]
Sample covariacne:
 [[ 23.45000531 -16.58822107]
 [-16.58822107  11.88901912]]
```



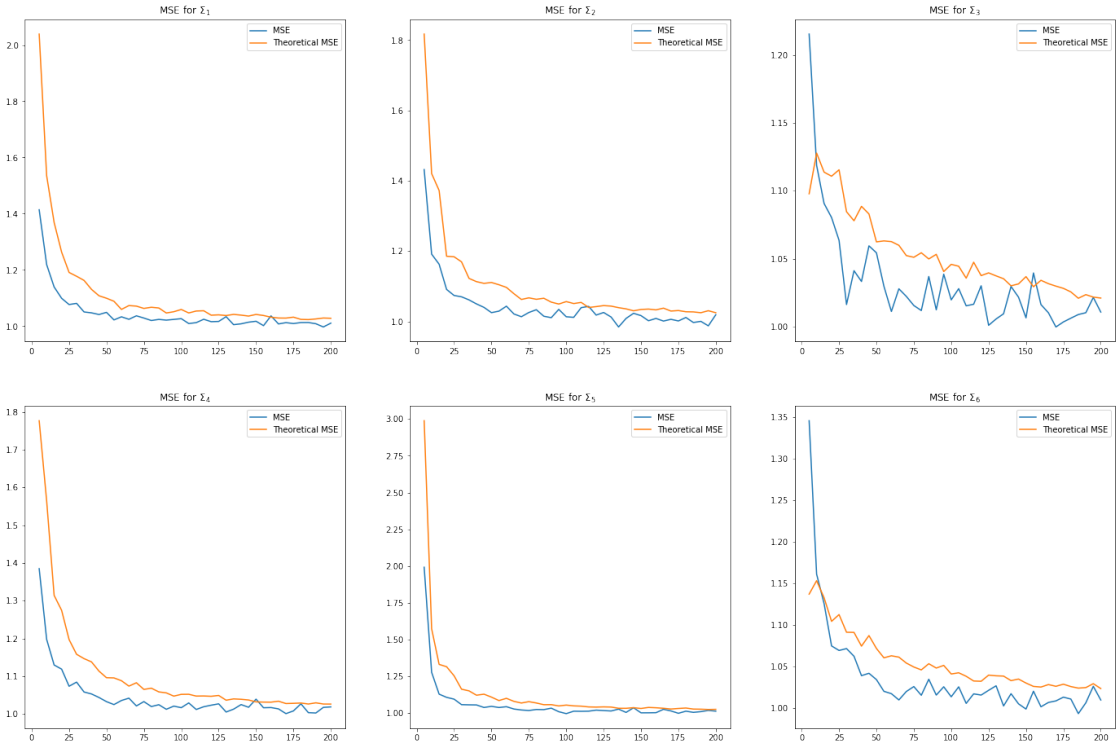
2 Question 3

In [126]:
```
import matplotlib.mlab as mlab

Size = [5, 50, 500]
Sigma = [[[1, 0], [0, 1]],
         [[1, 0.25], [0.25, 1]],
         [[1, 0.9], [0.9, 1]],
         [[1, -0.25], [-0.25, 1]],
         [[1, -0.9], [-0.9, 1]],
         [[0.1, 0], [0, 0.1]]
        ]

plt.figure(figsize=(16,24))
for i in range(len(Sigma)):
    for j in range(len(Size)):
        plt.subplot(6,3,i*3+j+1)
        z = np.random.normal(size=Size[j])
        samples = np.random.multivariate_normal([5, 5], Sigma[0], size=Size[j])
        x1 = samples[:,0]
        x2 = samples[:,1]
        y = x1+x2+z
        mu = np.linalg.inv(samples.T.dot(samples)+np.linalg.inv(Sigma[i])).dot(samples
        sigma = np.linalg.inv(samples.T.dot(samples))+Sigma[i]
        x_axis = np.arange(mu[0]-2.0, mu[0]+2.0, 0.01)
        y_axis = np.arange(mu[1]-2.0, mu[1]+2.0, 0.01)
        X, Y = np.meshgrid(x_axis, y_axis)
        w = mlab.bivariate_normal(X, Y, np.sqrt(sigma[0][0]), np.sqrt(sigma[1][1]), mu
        CS = plt.contour(X, Y, w)
        plt.clabel(CS, inline=1, fontsize=10)
        plt.xlim(-2,4)
        plt.ylim(-2,4)
        plt.title('Posterior on $w$ ($\Sigma_%i$,$n=%i$)'%(i+1,Size[j]))
plt.show()
```


Posterior on w (Σ_1,$n = 5$)
Posterior on w (Σ_1,$n = 50$)
Posterior on w (Σ_1,$n = 500$)
Posterior on w (Σ_2,$n = 5$)
Posterior on w (Σ_2,$n = 50$)
Posterior on w (Σ_2,$n = 500$)
Posterior on w (Σ_3,$n = 5$)
Posterior on w (Σ_3,$n = 50$)
Posterior on w (Σ_3,$n = 500$)
Posterior on w (Σ_4,$n = 5$)
Posterior on w (Σ_4,$n = 50$)
Posterior on w (Σ_4,$n = 500$)
Posterior on w (Σ_5,$n = 5$)
Posterior on w (Σ_5,$n = 50$)
Posterior on w (Σ_5,$n = 500$)
Posterior on w (Σ_6,$n = 5$)
Posterior on w (Σ_6,$n = 50$)
Posterior on w (Σ_6,$n = 500$)

In [138]:
```
import matplotlib.mlab as mlab

Size = [5, 50, 500]
Sigma = [[[1, 0], [0, 1]],
         [[1, 0.25], [0.25, 1]],
         [[1, 0.9], [0.9, 1]],
         [[1, -0.25], [-0.25, 1]],
         [[1, -0.9], [-0.9, 1]],
         [[0.1, 0], [0, 0.1]]
        ]


Rmean = np.zeros((len(Sigma),40))
the_error = np.zeros((len(Sigma),40))
for i in range(len(Sigma)):
    for j in range(5,205,5):
        for _ in range(200):
            z = np.random.normal(size=j+100)
            samples = np.random.multivariate_normal([5, 5], Sigma[0], size=j+100)
            train = samples[:-100,:]
            test = samples[-100:,:]
            y = np.sum(train,axis=1)+z[:-100]
            y_test = np.sum(test,axis=1)+z[-100:]
            what = np.linalg.inv(train.T.dot(train)+np.linalg.inv(Sigma[i])).dot(train
            yhat = test.dot(what)
            Rmean[i][(j-5)//5] += np.sum((yhat-y_test)**2)/100
            the_error[i][(j-5)//5] += 5*(what[0]-1)**2+5*(what[1]-1)**2+1
        Rmean[i][(j-5)//5] /= 200
        the_error[i][(j-5)//5] /= 200
```

In [139]:
```
plt.figure(figsize=(24,16))
for i in range(len(Sigma)):
    plt.subplot(2,3,i+1)
    plt.plot(range(5,205,5),Rmean[i],label='MSE')
    plt.plot(range(5,205,5),the_error[i],label='Theoretical MSE')
    plt.title('MSE for $\Sigma_%i$'%(i+1))
    plt.legend()
plt.show()
```



3 Question 4

```
In [21]: import numpy as np
         import matplotlib.pyplot as plt
         from scipy.misc import imread,imsave

         imFile = 'hw04-data/stpeters_probe_small.png'
         compositeFile = 'hw04-data/tennis.png'
         targetFile = 'hw04-data/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 = 'hw04-data/stpeters_probe_small.png'
             compositeFile = 'hw04-data/tennis.png'
             targetFile = 'hw04-data/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
# the input coeff where coeff is a 9 x c matrix
# with c being the number of color channels
# The output is an 2r x 2r x c image of a diffuse sphere
# and the value of -1 on the image where there is no sphere
def renderSphere(r,coeff):

    d = 2*r
```

```
    img = -np.ones((d,d,3))
    ns = []
    ps = []

    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:
                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)))
            ns.append(n)
            ps.append((i,j))

    ns = np.asarray(ns)
    B = computeBasis(ns)
    vs = B.dot(coeff)

    for p,v in zip(ps,vs):
        img[p[0],p[1]] = np.clip(v,0,255)

    return img

# relights the sphere in img, which is assumed to be a square image
# coeff is the matrix of spherical harmonic coefficients
def relightSphere(img, coeff):
    img = renderSphere(int(img.shape[0]/2),coeff)/255*img/255
    return img

# Copies the image of source onto target
# pixels with values of -1 in source will not be copied
def compositeImages(source, target):

    # Assumes that all pixels not equal to 0 should be copied
    out = target.copy()
    cx = int(target.shape[1]/2)
    cy = int(target.shape[0]/2)
    sx = cx - int(source.shape[1]/2)
    sy = cy - int(source.shape[0]/2)

    for i in range(source.shape[0]):
```

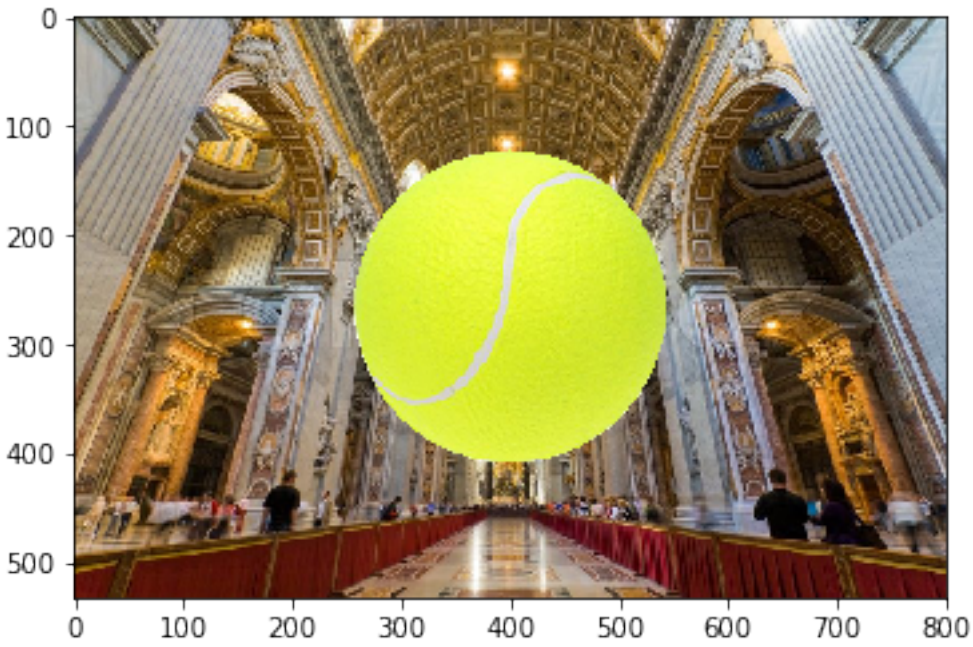
```
                for j in range(source.shape[1]):
                    if np.sum(source[i,j]) >= 0:
                        out[sy+i,sx+j] = source[i,j]

        return out
```

In [22]:
```
# Fill in this function to compute the basis functions
# This function is used in renderSphere()
def computeBasis(ns):
    # Returns the first 9 spherical harmonic basis functions

    ################################################
    # Compute the first 9 basis functions
    ################################################
    B = np.ones((len(ns),9)) # This line is here just to fill space
    B[:,1] = ns[:,1]
    B[:,2] = ns[:,0]
    B[:,3] = ns[:,2]
    B[:,4] = np.multiply(ns[:,0],ns[:,1])
    B[:,5] = np.multiply(ns[:,1],ns[:,2])
    B[:,6] = 3*np.multiply(ns[:,2],ns[:,2])-1
    B[:,7] = np.multiply(ns[:,0],ns[:,2])
    B[:,8] = np.multiply(ns[:,0],ns[:,0])-np.multiply(ns[:,1],ns[:,1])

    return B
```

In [23]:
```
data,tennis,target = loadImages()
ns, vs = extractNormals(data)
B = computeBasis(ns)
# reduce the number of samples because computing the SVD on
# the entire data set takes too long
Bp = B[::50]
vsp = vs[::50]

################################################
# Solve for the coefficients using least squares
# or total least squares here
################################################
coeff = np.zeros((9,3))
coeff[0,:] = 255

img = relightSphere(tennis,coeff)

output = compositeImages(img,target)

print('Coefficients:\n'+str(coeff))

plt.figure(1)
```

```
        plt.imshow(output)
        plt.show()

        imsave('hw04-data/output.png',output)
```

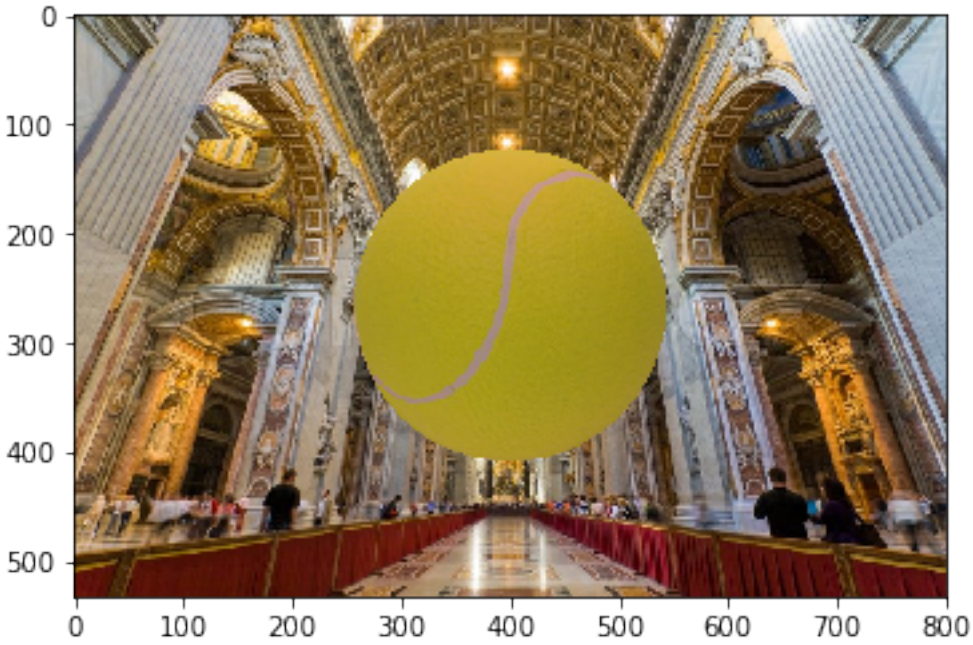
```
Coefficients:
[[ 255.  255.  255.]
 [   0.    0.    0.]
 [   0.    0.    0.]
 [   0.    0.    0.]
 [   0.    0.    0.]
 [   0.    0.    0.]
 [   0.    0.    0.]
 [   0.    0.    0.]
 [   0.    0.    0.]]
```




```
In [24]: def regreession_multi(X,y,lamb=0):
             n1, n2 = np.shape(X)
             A = np.linalg.inv(X.T.dot(X)+lamb*np.identity(n2)).dot(X.T.dot(y))
             yhat = X.dot(A)

             Rmean =  np.linalg.norm(yhat-y)**2/n1

             return {'A':A,'train_error':Rmean}
```

```
In [25]: data,tennis,target = loadImages()
         ns, vs = extractNormals(data)
         B = computeBasis(ns)

         # reduce the number of samples because computing the SVD on
         # the entire data set takes too long
         Bp = B[::50]
         vsp = vs[::50]

         ################################################
         # Solve for the coefficients using least squares
         # or total least squares here
         coeff = regreession_multi(Bp,vsp)['A']

         img = relightSphere(tennis,coeff)

         output = compositeImages(img,target)

         print('Coefficients:\n'+str(coeff))

         plt.figure(1)
         plt.imshow(output)
         plt.show()

         imsave('hw04-data/output-ols.png',output)
```

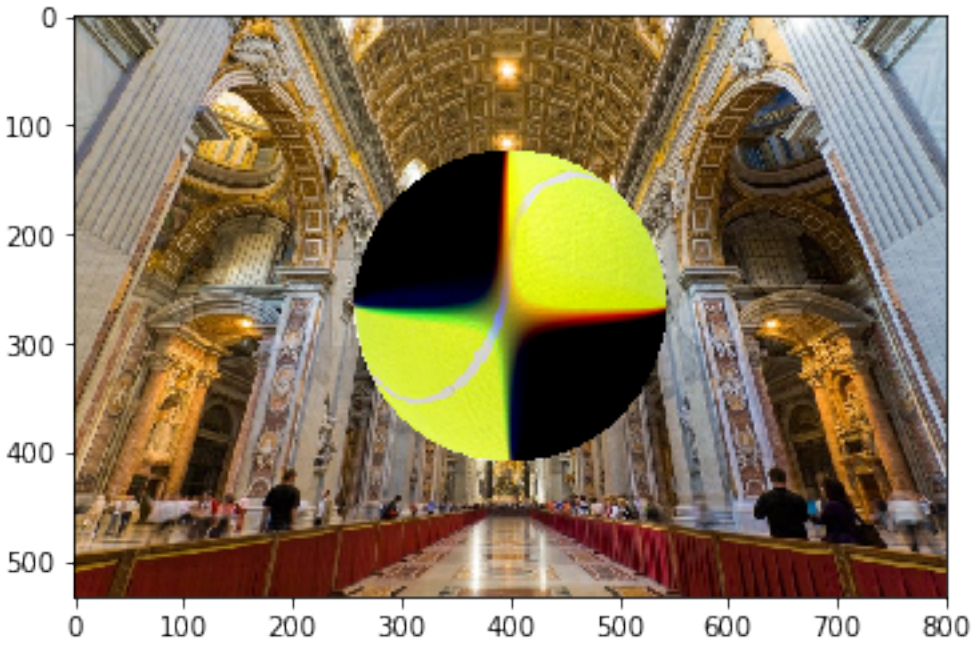
```
Coefficients:
[[ 202.31845431  162.41956802  149.07075034]
 [ -27.66555164  -17.88905339  -12.92356688]
 [  -5.15203925   -4.51375871   -4.24262639]
 [  -1.08629293    0.42947012    1.15475569]
 [  -3.14053107   -3.70269907   -3.74382934]
 [  23.67671768   23.15698002   21.94638397]
 [  -3.82167171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



In [31]:
```
data,tennis,target = loadImages()
ns, vs = extractNormals(data)
B = computeBasis(ns)

# reduce the number of samples because computing the SVD on
# the entire data set takes too long
Bp = B[::50]
vsp = vs[::50]

##################################################
# Solve for the coefficients using least squares
# or total least squares here
A = np.concatenate((Bp,vsp),axis=1)
u1,e1,vt1 = np.linalg.svd(A)
v1 = vt1.T
_,n = np.shape(vsp)

coeff = -v1[:-3,-3:].dot(np.linalg.inv(v1[-3:,-3:]))


img = relightSphere(tennis,coeff)

output = compositeImages(img,target)

print('Coefficients:\n'+str(coeff))
```

```
        plt.figure(1)
        plt.imshow(output)
        plt.show()

        imsave('hw04-data/output-tls.png',output)
```

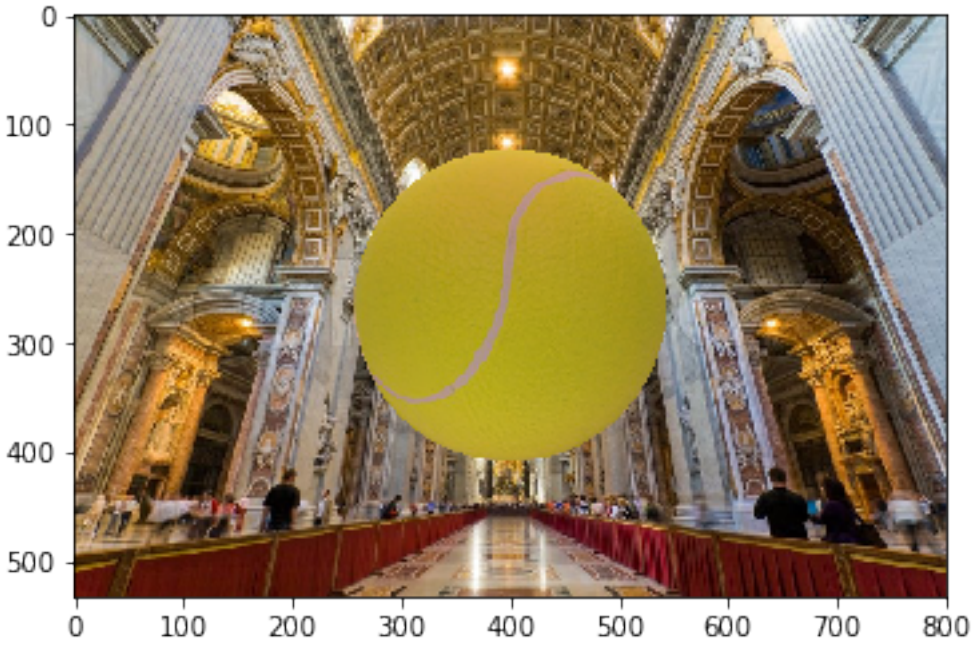
```
Coefficients:
[[  2.13318421e+02   1.70780299e+02   1.57126297e+02]
 [ -3.23046362e+01  -2.02975310e+01  -1.45516114e+01]
 [ -4.31689131e+00  -3.80778081e+00  -4.83616306e+00]
 [ -4.89811386e+00  -3.37684058e+00  -1.14207091e+00]
 [ -7.05901066e+03  -7.39934207e+03  -4.26448732e+03]
 [ -3.05378224e+02  -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]]
```



In [30]:
```
        data,tennis,target = loadImages()
        ns, vs = extractNormals(data)
        B = computeBasis(ns)

        # reduce the number of samples because computing the SVD on
        # the entire data set takes too long
        Bp = B[::50]
        vsp = vs[::50]/384
```

```
        ################################################
        # Solve for the coefficients using least squares
        # or total least squares here
        A = np.concatenate((Bp,vsp),axis=1)
        u2,e2,vt2 = np.linalg.svd(A)
        v2 = vt2.T
        _,n = np.shape(vsp)

        coeff = -v2[:-3,-3:].dot(np.linalg.inv(v2[-3:,-3:]))

        img = relightSphere(tennis,coeff*384)

        output = compositeImages(img,target)

        print('Coefficients:\n'+str(coeff))

        plt.figure(1)
        plt.imshow(output)
        plt.show()

        imsave('hw04-data/output-tls-2.png',output)
```

```
Coefficients:
[[ 0.54526595  0.44019965  0.40460097]
 [-0.07882306 -0.05287614 -0.03959563]
 [-0.01498285 -0.01322608 -0.01245169]
 [-0.0027508   0.00120776  0.00310405]
 [-0.02058775 -0.02136242 -0.02096713]
 [ 0.14313155  0.13704163  0.13044962]
 [-0.01002394  0.00144957  0.00469367]
 [ 0.01907957  0.00997563  0.00279948]
 [-0.02840275 -0.01784431 -0.01530017]]
```




```
In [28]: e1

Out[28]: array([  1.97242074e+04,   1.13371638e+03,   3.15377927e+02,
                  5.47019961e+01,   3.61875111e+01,   3.61365506e+01,
                  3.57013616e+01,   3.23244599e+01,   1.64545121e+01,
                  1.61751017e+01,   1.61259393e+01,   1.51314453e+01])

In [29]: e2

Out[29]: array([ 80.01823499,  56.06983748,  36.25734868,  36.16417259,
                 36.13202541,  32.3669552 ,  16.26908789,  16.17674571,
                 16.15123095,  12.12656088,   2.51359918,   0.7490711 ])

In [34]: v1[-3:,-3:]

Out[34]: array([[-0.00046915, -0.00101783,  0.01065118],
                [ 0.00075001,  0.00063401, -0.01297574],
                [-0.00029218,  0.00061354,  0.00489062]])

In [35]: v2[-3:,-3:]

Out[35]: array([[-0.43046285, -0.71538562,  0.33722025],
                [-0.45160273,  0.10363068, -0.81227462],
                [-0.44193905,  0.68716557,  0.47552476]])

In [ ]:
```