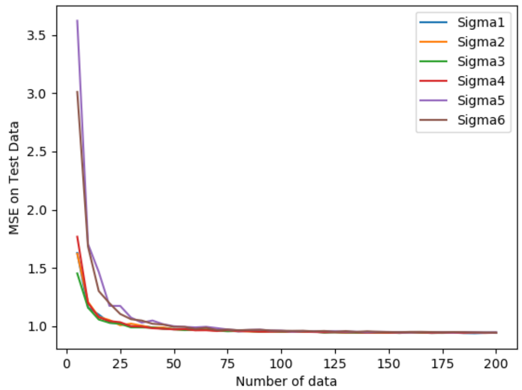
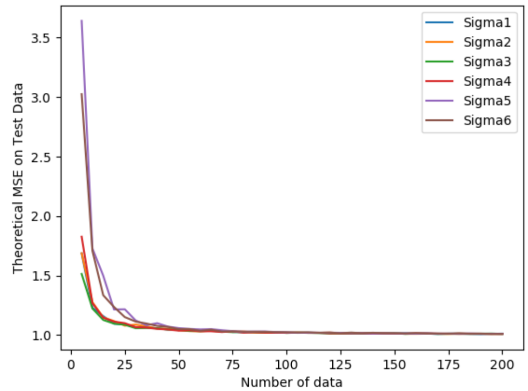
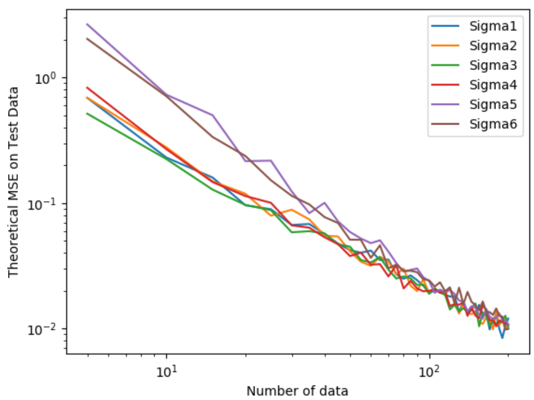
3e







1\ As the number of training data increases, we have smaller MSE.

2\ When number of training data is small, MSE depends on which prior we choose. Different prior ends up with different MSE.

3\ As the number of training data increases, MSE becomes less dependent on the prior we choose. This is because as the number of training data increases, the effect of prior on posterior distribution decreases, and information from the training data becomes dominant.

4\Here we don't have control on prior mean which is [0, 0], we only have control on Sigma, and the true is at [1, 1]. A good prior distribution should have a large density at point [1, 1]. Since the Guassian prior center is at [0, 0], we need a Sigma which makes the prior distribution centered at [0, 0] but points to [1, 1]. And such a sigma is strongly positively correlated(Sigma3).

Code:

```
import numpy as np
import matplotlib.pyplot as plt
from numpy.linalg import inv
np.random.seed(0)
w = [1.0,1.0]
n_test = 500
n_trains = np.arange(5,205,5)
n_trails = 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)]

def generate_data(n):
    """
    This function generates data of size n.
    """
    X = np.random.multivariate_normal([0, 0], [[5, 0], [0, 5]], size = n)
    z = np.random.normal(0, 1, size = n).reshape((n, 1))
    y = X @ np.array([[1], [1]]) + z
    return (X,y)

def compute_mean_var(X,y,Sigma):
    """
    This function computes the mean and variance of the posterior
    """
    pos_var = inv(X.T @ X + inv(Sigma))
    pos_mu = pos_var @ X.T @ y
    mux = pos_mu[0, 0]
    muy = pos_mu[1, 0]
    sigmax = sqrt(pos_var[0, 0])
    sigmay = sqrt(pos_var[1, 1])
    sigmaxy = pos_var[0, 1]
    return mux,muy,sigmax,sigmay,sigmaxy

def tikhonov_regression(X,y,Sigma):
    """
    This function computes w based on the formula of tikhonov_regression.
    """
```

```
    mux, muy, dummy, dummy, dummy = compute_mean_var(X,y,Sigma)
    return np.array([[mux], [muy]])

def compute_mse(X,Y, w):
    """
    This function computes MSE given data and estimated w.
    """
    n = X.shape[0]
    Y_hat = X @ w
    mse = (Y-Y_hat).T @ (Y-Y_hat)/n
    return mse

def compute_theoretical_mse(w):
    """
    This function computes theoretical MSE given estimated w.
    """
    theoretical_mse = 5*(w[0, 0]-1)**2 + 5*(w[1, 0]-1)**2 + 1
    return theoretical_mse

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

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

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

for seed in range(n_trails):
    np.random.seed(seed)
    for i,Sigma in enumerate(Sigmas):
        for j,n_train in enumerate(n_trains):
            #TODO implement the mses and theoretical_mses
            X_train, y_train = generate_data(n_train)
            w_hat = tikhonov_regression(X_train, y_train, Sigma)
            mses[i, j, seed] = compute_mse(X_test, y_test, w_hat)
            theoretical_mses[i, j, seed] = compute_theoretical_mse(w_hat)

# Plot
plt.figure()
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.legend()
plt.savefig('MSE.png')
```

```
plt.figure()
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('Theoretical MSE on Test Data')
plt.legend()
plt.savefig('theoretical_MSE.png')


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