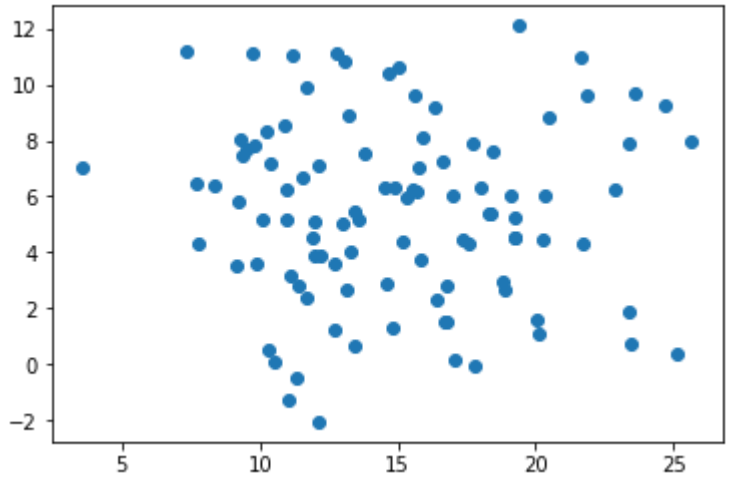
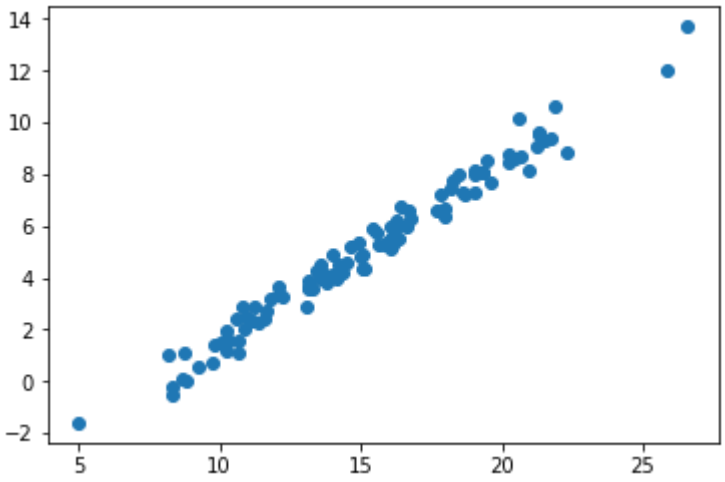
Question 2

In [13]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.widgets import Slider
np.random.seed(0)

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()
mean = np.average(samples, axis = 0)
covariance = np.sum(((sample - mean).reshape(2,1)).dot((np.transpose(sa
mple - mean)).reshape(1,2)) for sample in samples)/len(samples)


print("Mean_hat: " + str(mean))
print("Covariance_hat: " + str(covariance))
#print((np.transpose(sample - mean)).dot(sample - mean) for sample in s
amples)
```



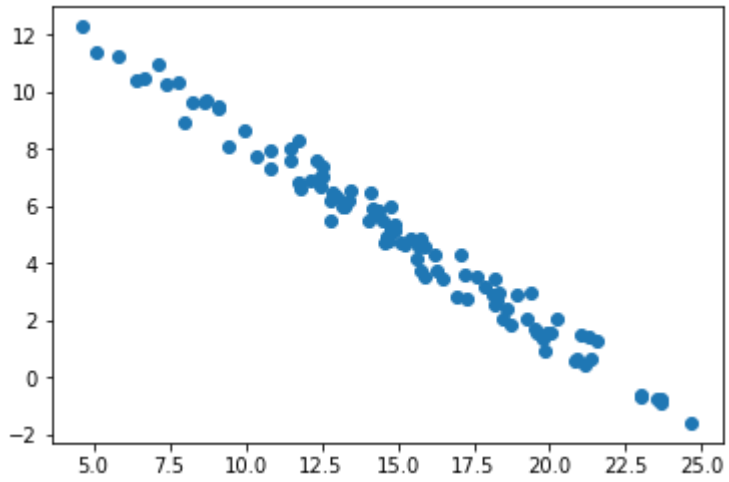

```
Mean_hat: [14.99571712  5.45150579]
Covariance_hat: [[20.88795883 -0.35242617]
 [-0.35242617 10.31880934]]
```

In [55]:
```
sigma = [[20, 14], [14, 10]]
samples = np.random.multivariate_normal(mu, sigma, size=100)
plt.scatter(samples[:, 0], samples[:, 1])
plt.show()
mean = np.average(samples, axis = 0)
covariance = np.sum(((sample - mean).reshape(2,1)).dot((np.transpose(sa
mple - mean)).reshape(1,2)) for sample in samples)/len(samples)
print("Mean_hat: " + str(mean))
print("Covariance_hat: " + str(covariance))
```




```
Mean_hat: [ 15.12304808   5.04815477]
Covariance_hat: [[ 17.78285704  12.3823083 ]
 [ 12.3823083    8.83176695]]
```

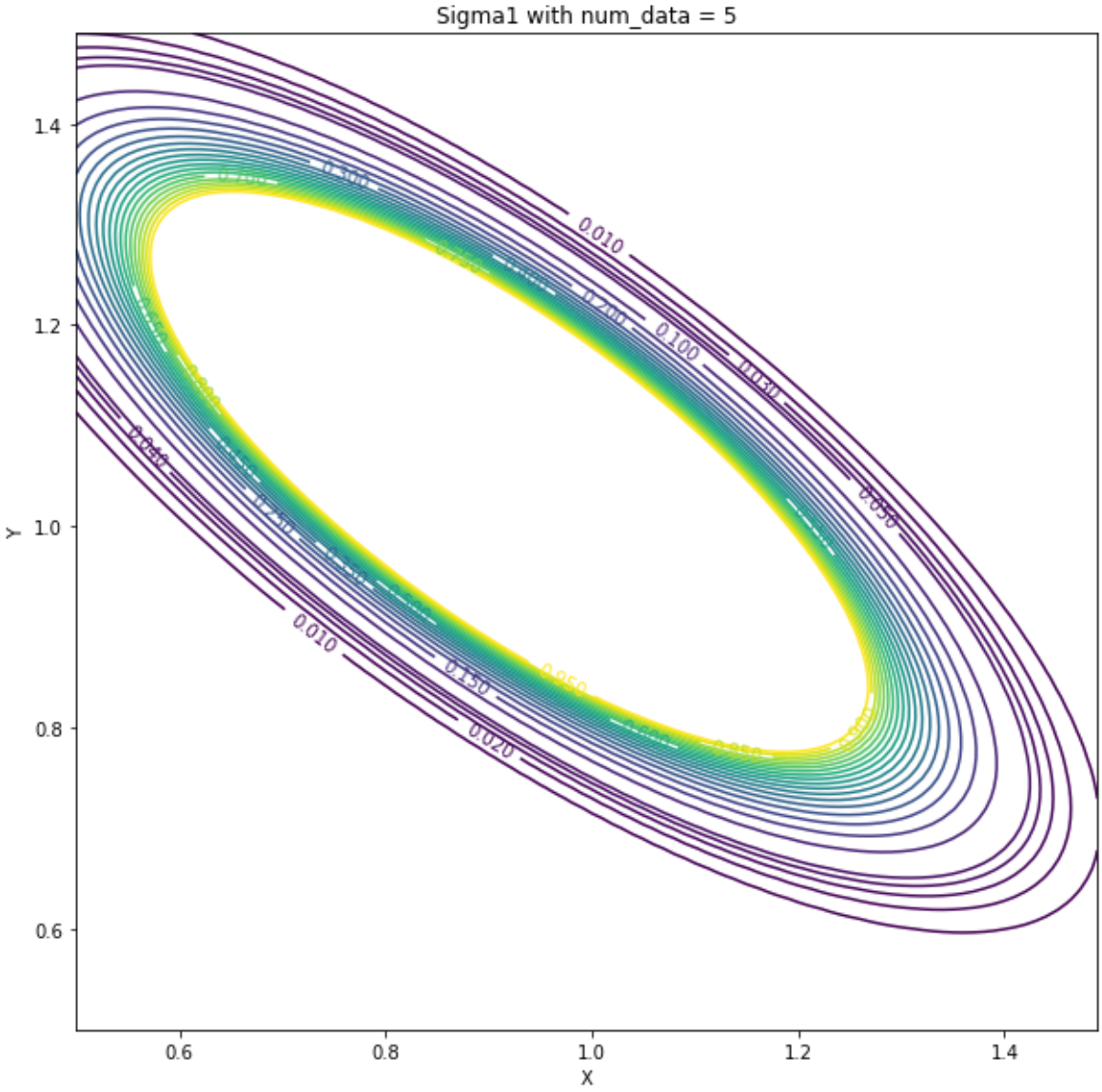
In [54]:
```
sigma = [[20, -14], [-14, 10]]
samples = np.random.multivariate_normal(mu, sigma, size=100)
plt.scatter(samples[:, 0], samples[:, 1])
plt.show()
mean = np.average(samples, axis = 0)
covariance = np.sum(((sample - mean).reshape(2,1)).dot((np.transpose(sa
mple - mean)).reshape(1,2)) for sample in samples)/len(samples)
print("Mean_hat: " + str(mean))
print("Covariance_hat: " + str(covariance))
```

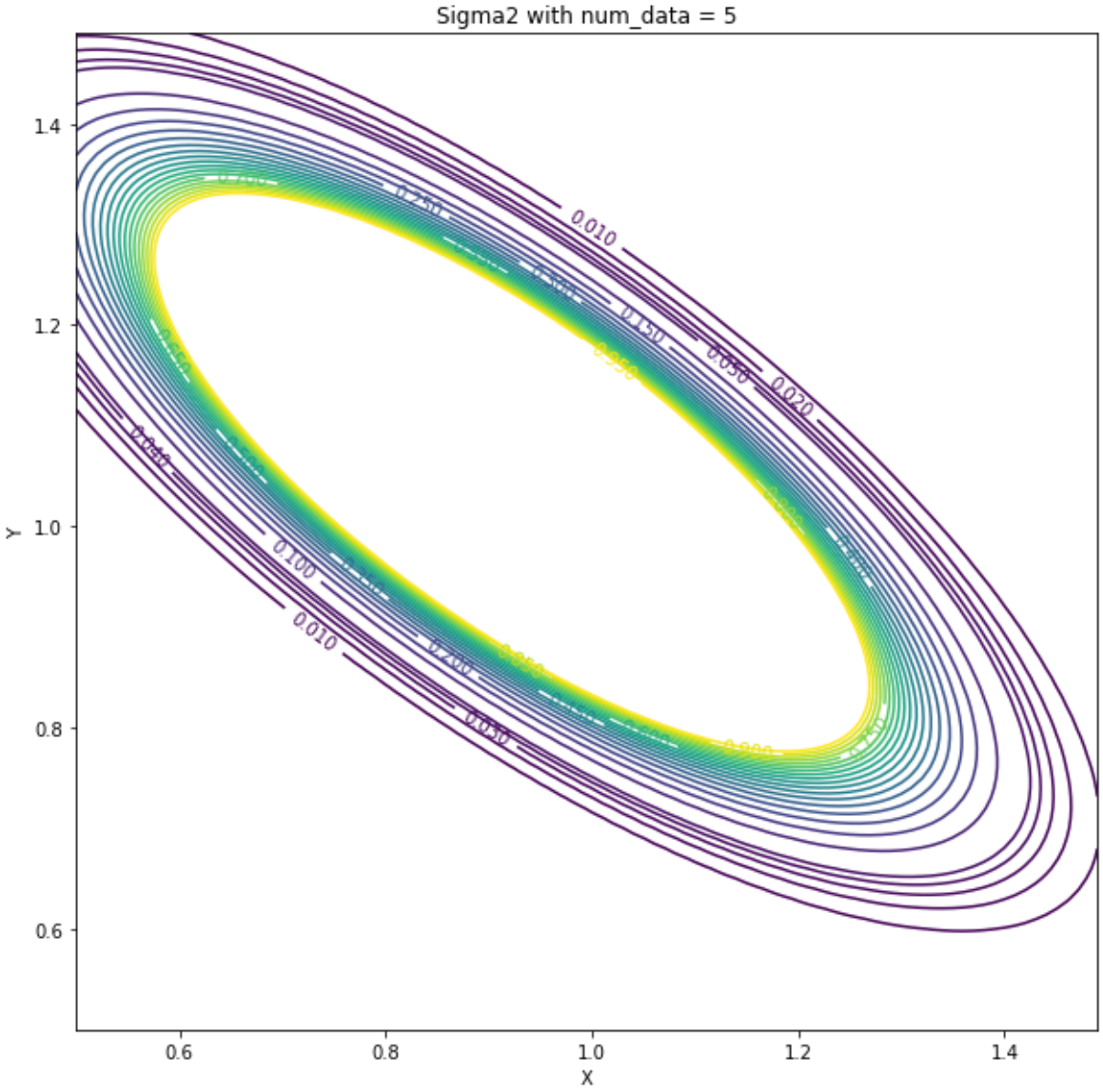


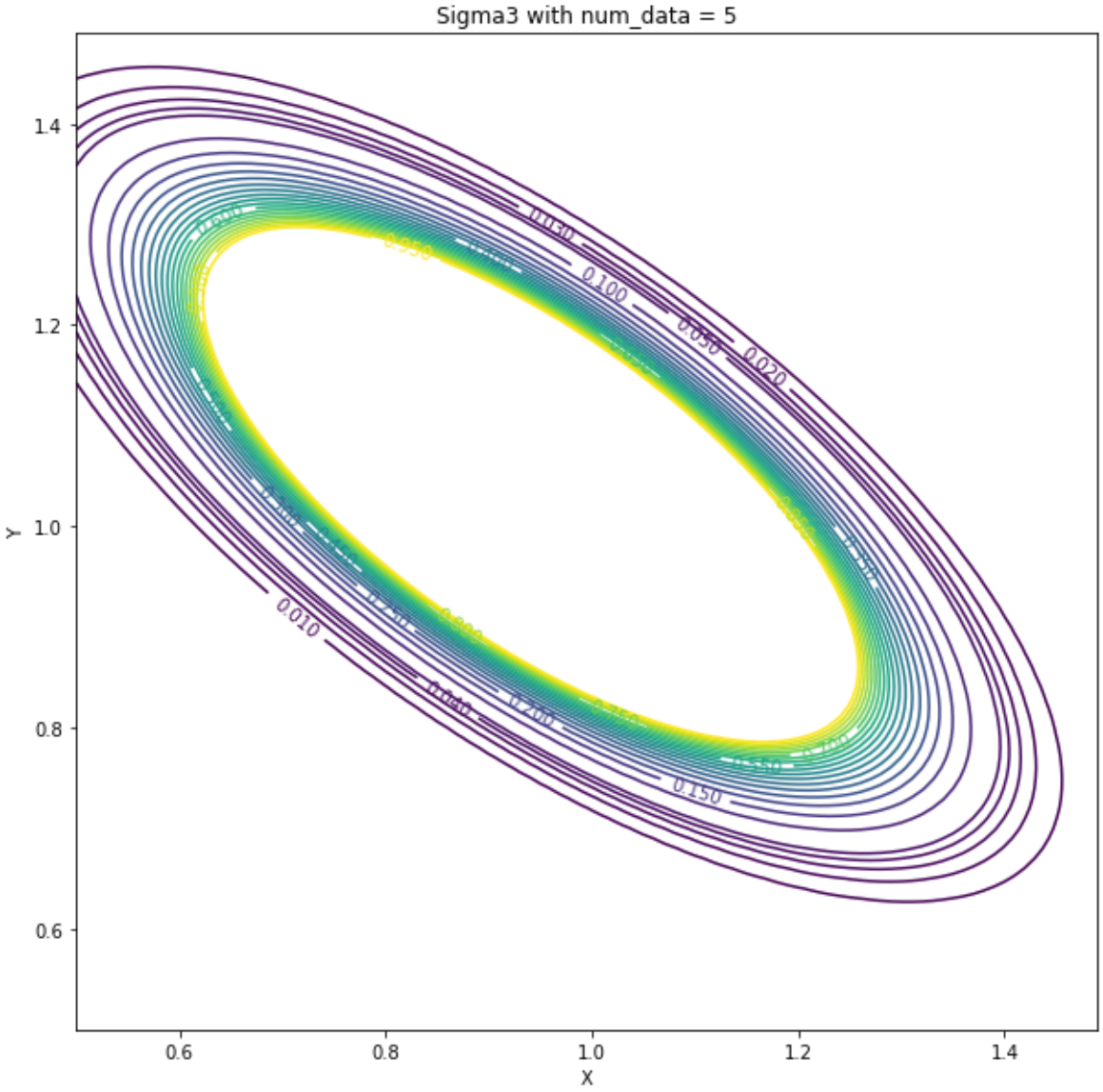

```
Mean_hat: [ 15.06341626   4.96594739]
Covariance_hat: [[ 21.46041612 -14.67064269]
 [-14.67064269  10.23285275]]
```

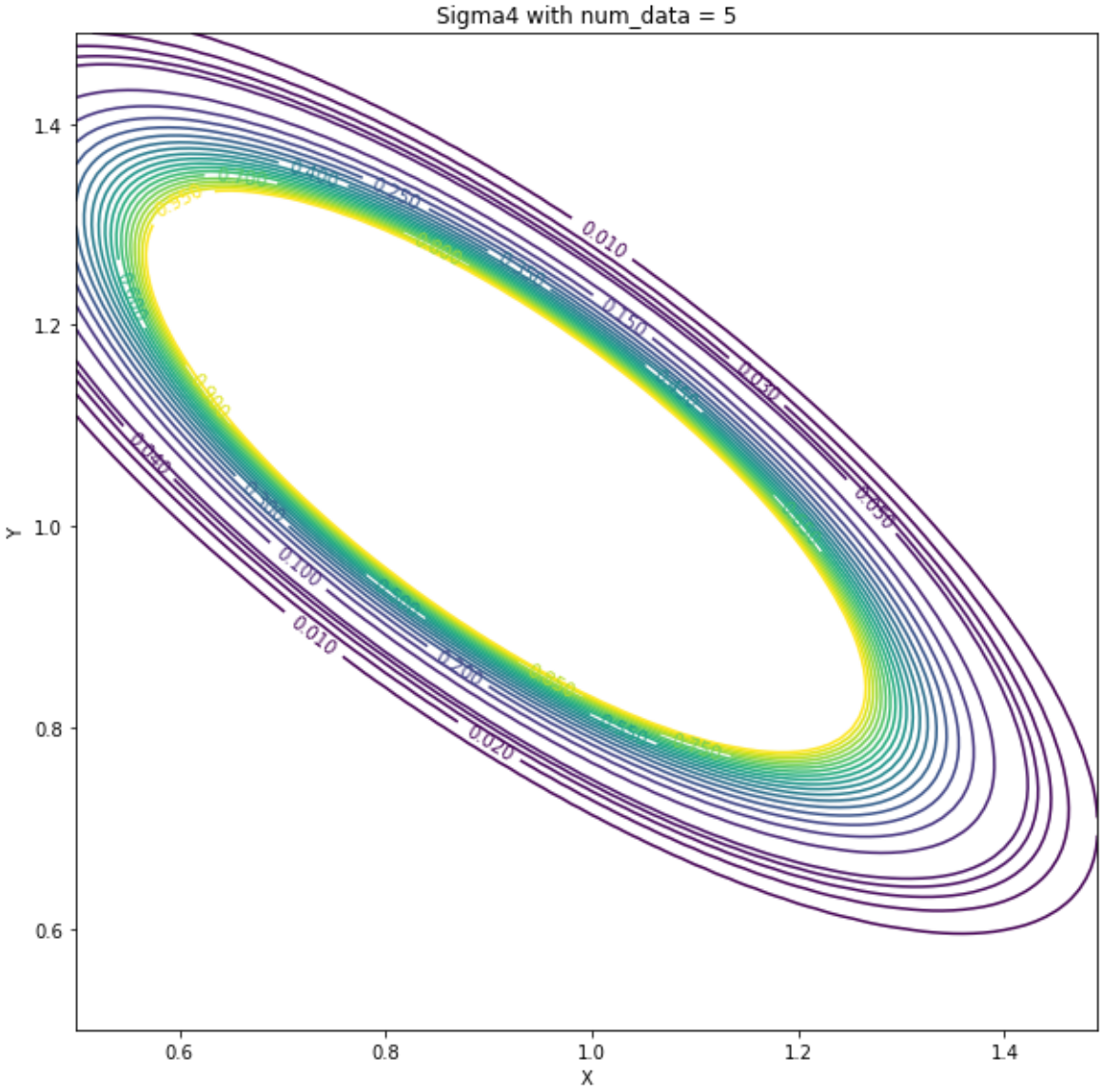
In [12]:
```
def generate_data(n):
    """
    This function generates data of size n.
    """

    X = np.random.normal(0, 5, (n, 2))
    z = np.random.normal(0, 1, (n, 1))

    y = np.sum(X, axis = 1).reshape(n, 1) + z

    return (X,y)

def tikhonov_regression(X,Y,Sigma):
    """
    This function computes w based on the formula of tikhonov_regressio
n.
    """
    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
    """
    mux, muy = tikhonov_regression(X, y, Sigma)
    m = np.linalg.inv(X.T.dot(X) + np.linalg.inv(Sigma))
```

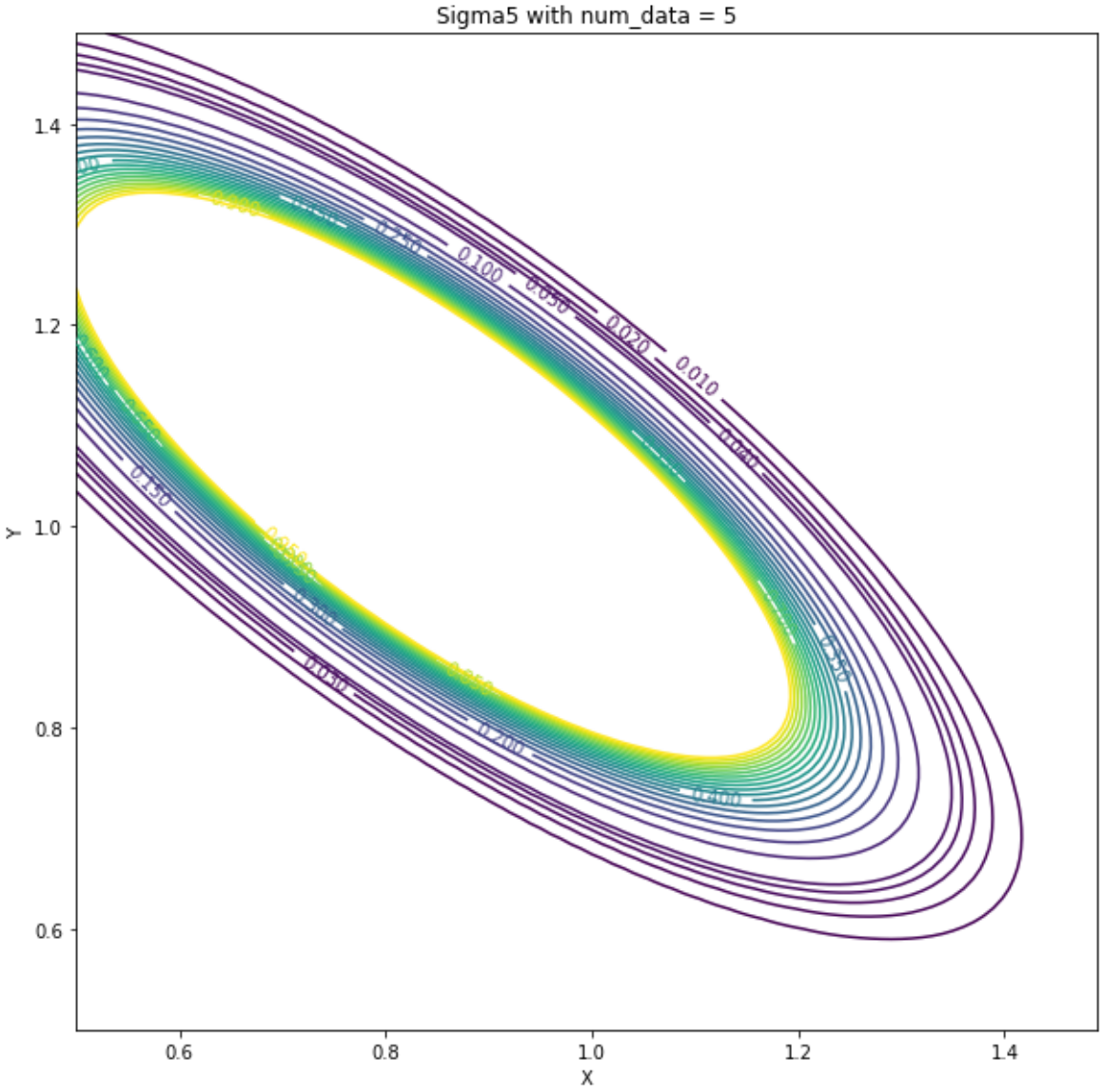
```
    sigmax, sigmay, sigmaxy = np.sqrt(m[0][0]), np.sqrt(m[1][1]), m[0][
1]

    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):

        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)

        Z = matplotlib.mlab.bivariate_normal(X_grid,Y_grid, sigmax, sig
may, 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_d
ata))
```

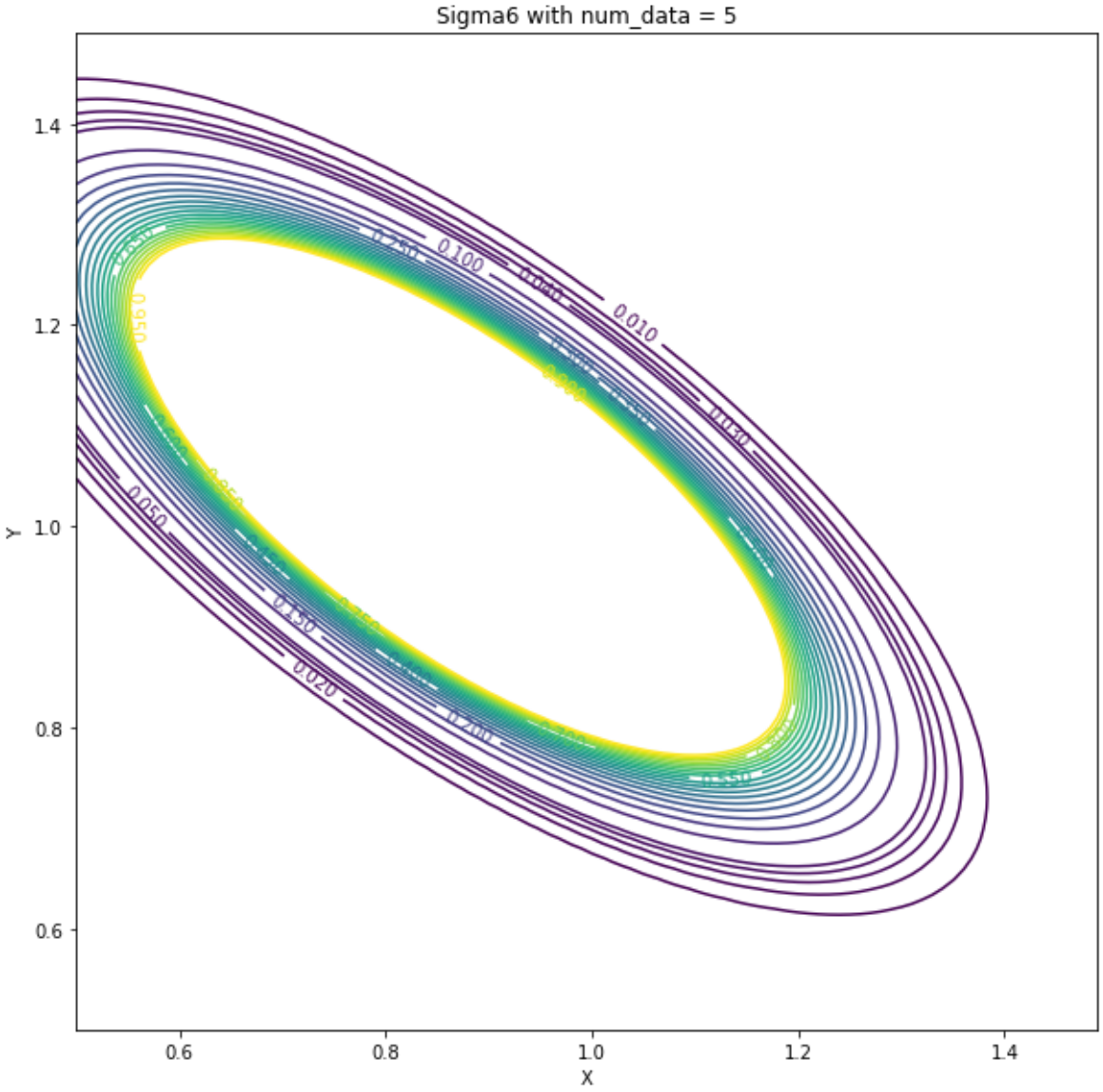

Sigma1 with num_data = 5
0.010
0.100
0.050
0.010
0.150
0.020
1.4
1.2
1.0
0.8
0.6
Y
0.6
0.8
1.0
1.2
1.4
X

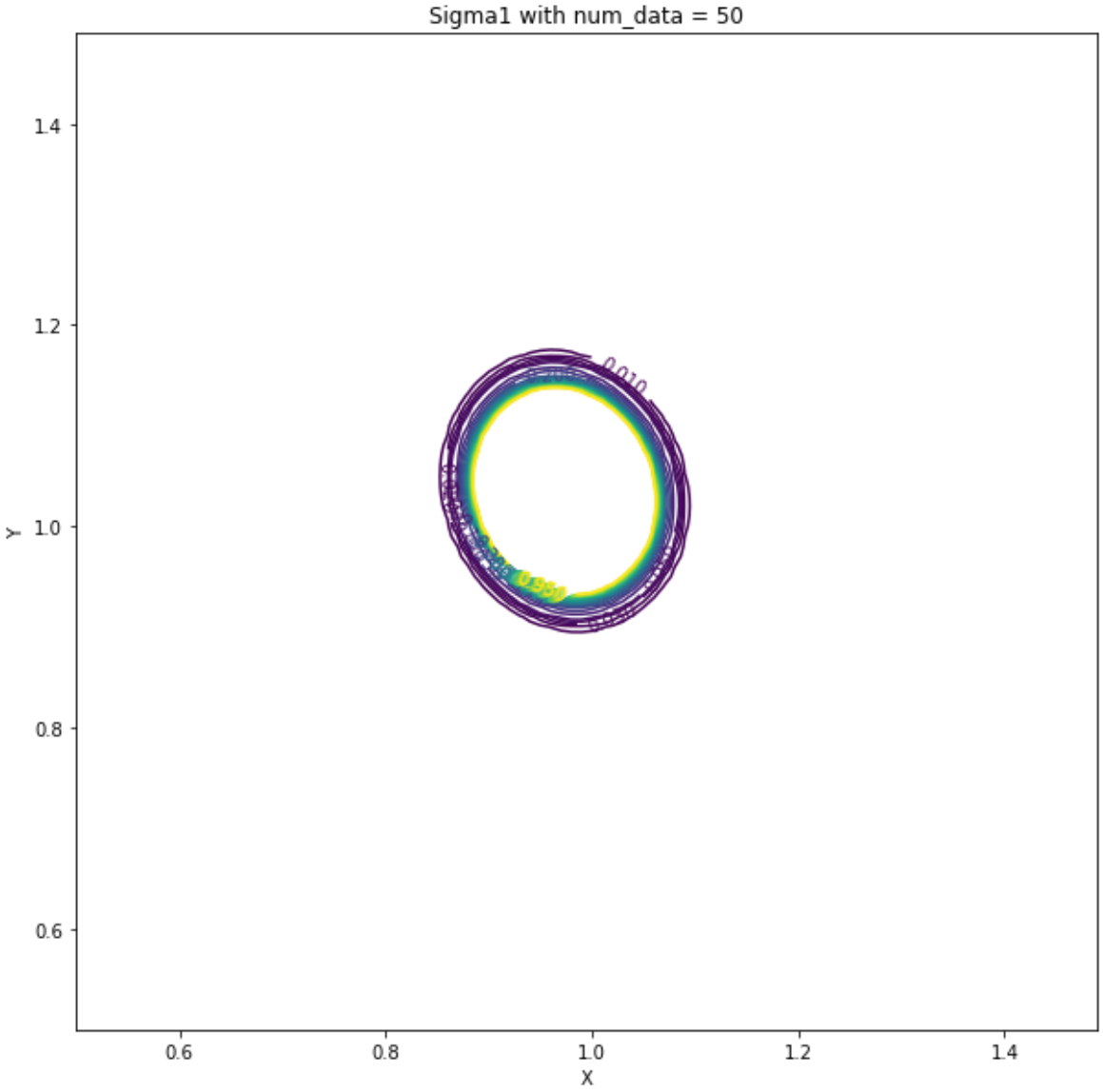

Sigma2 with num_data = 5
0.010
0.020
0.050
0.150
0.100
0.010
0.020
0.050
0.100
0.150
1.4
1.2
1.0
0.8
0.6
Y
0.6
0.8
1.0
1.2
1.4
X

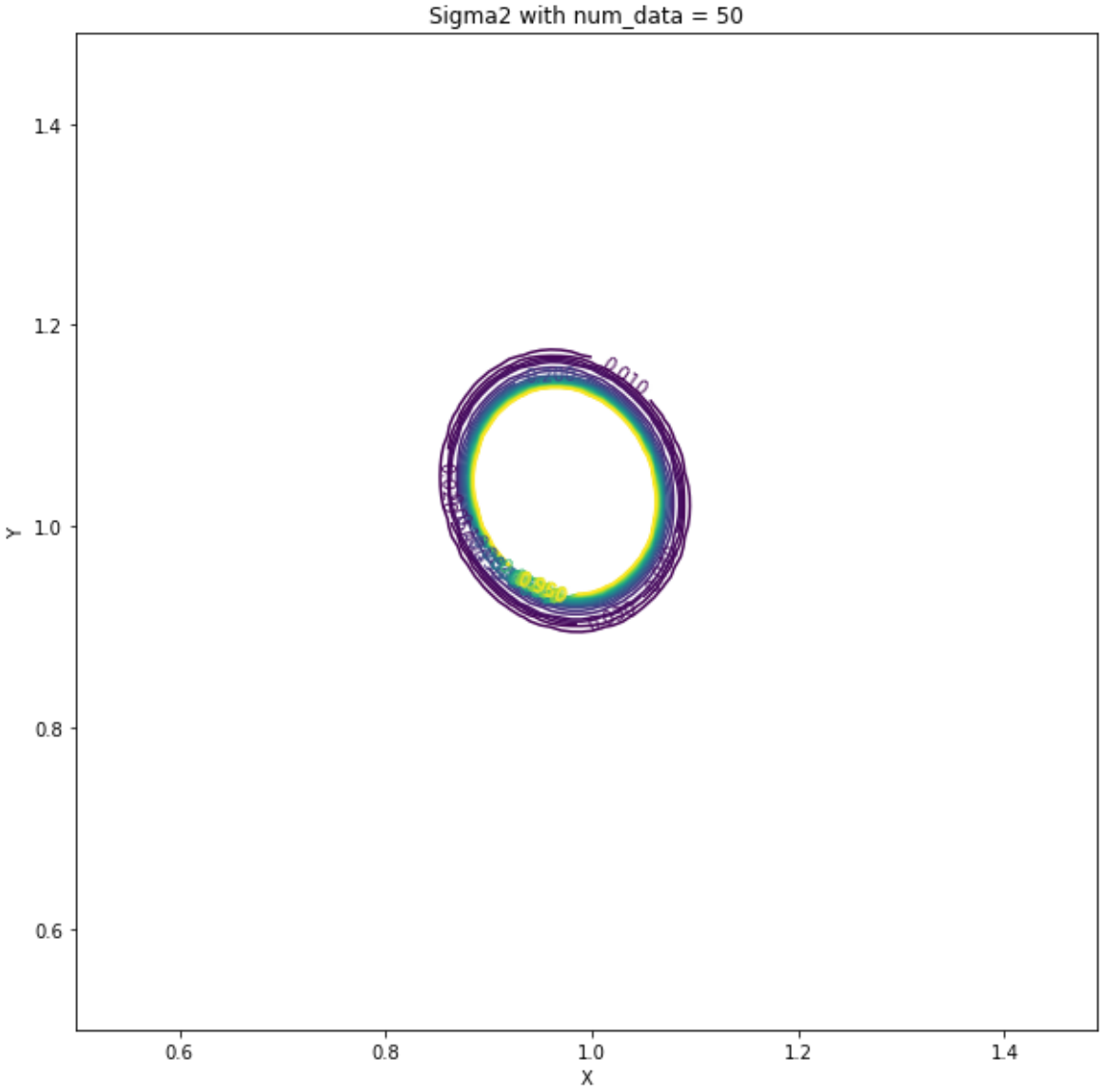

Sigma3 with num_data = 5
1.4
1.2
1.0
0.8
0.6
Y
0.6
0.8
1.0
1.2
1.4
X
0.150
0.100
0.050
0.020
0.010

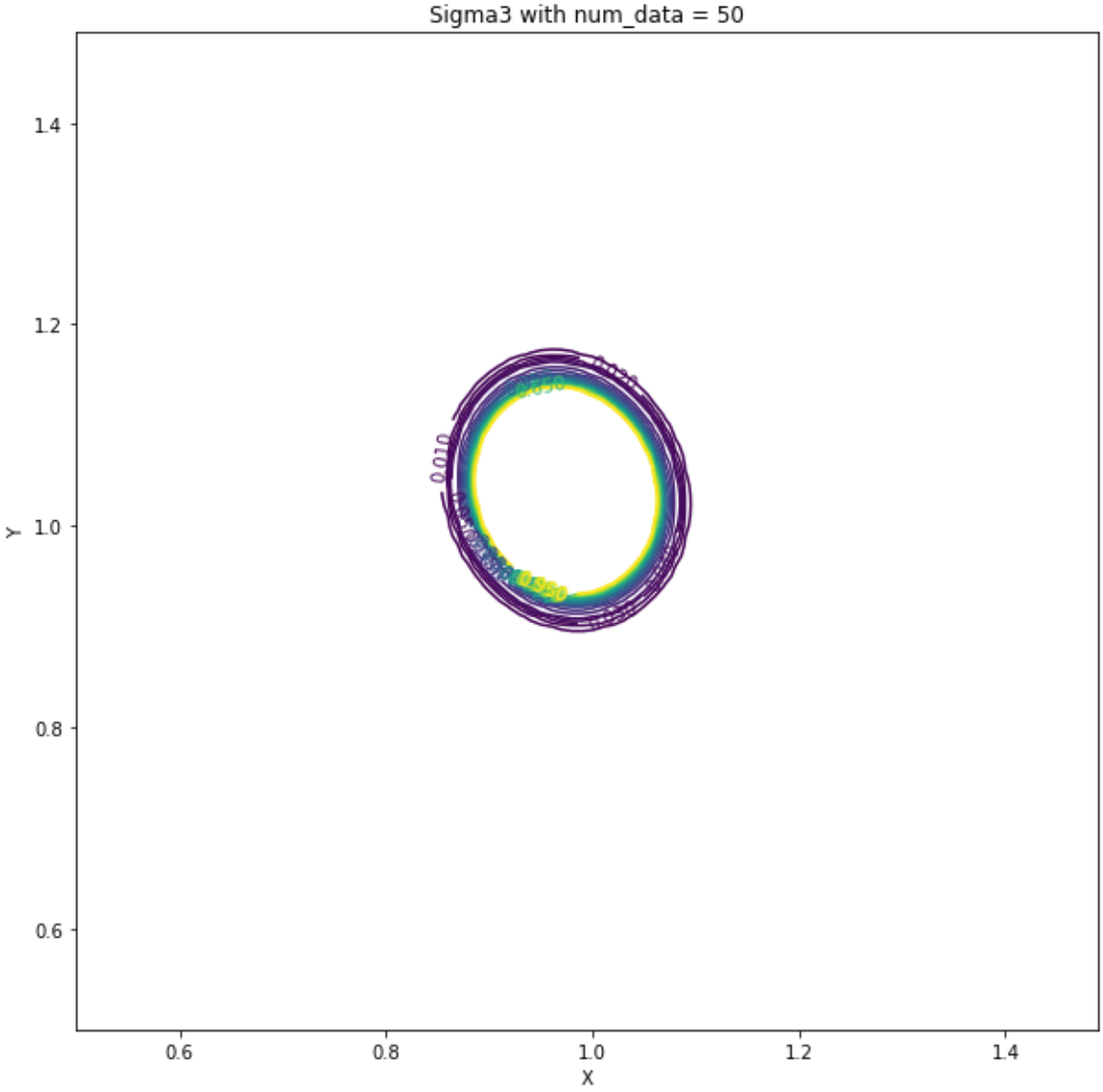

Sigma4 with num_data = 5
1.4
1.2
1.0
0.8
0.6
Y
0.6
0.8
1.0
1.2
1.4
X
0.010
0.050
0.100
0.020

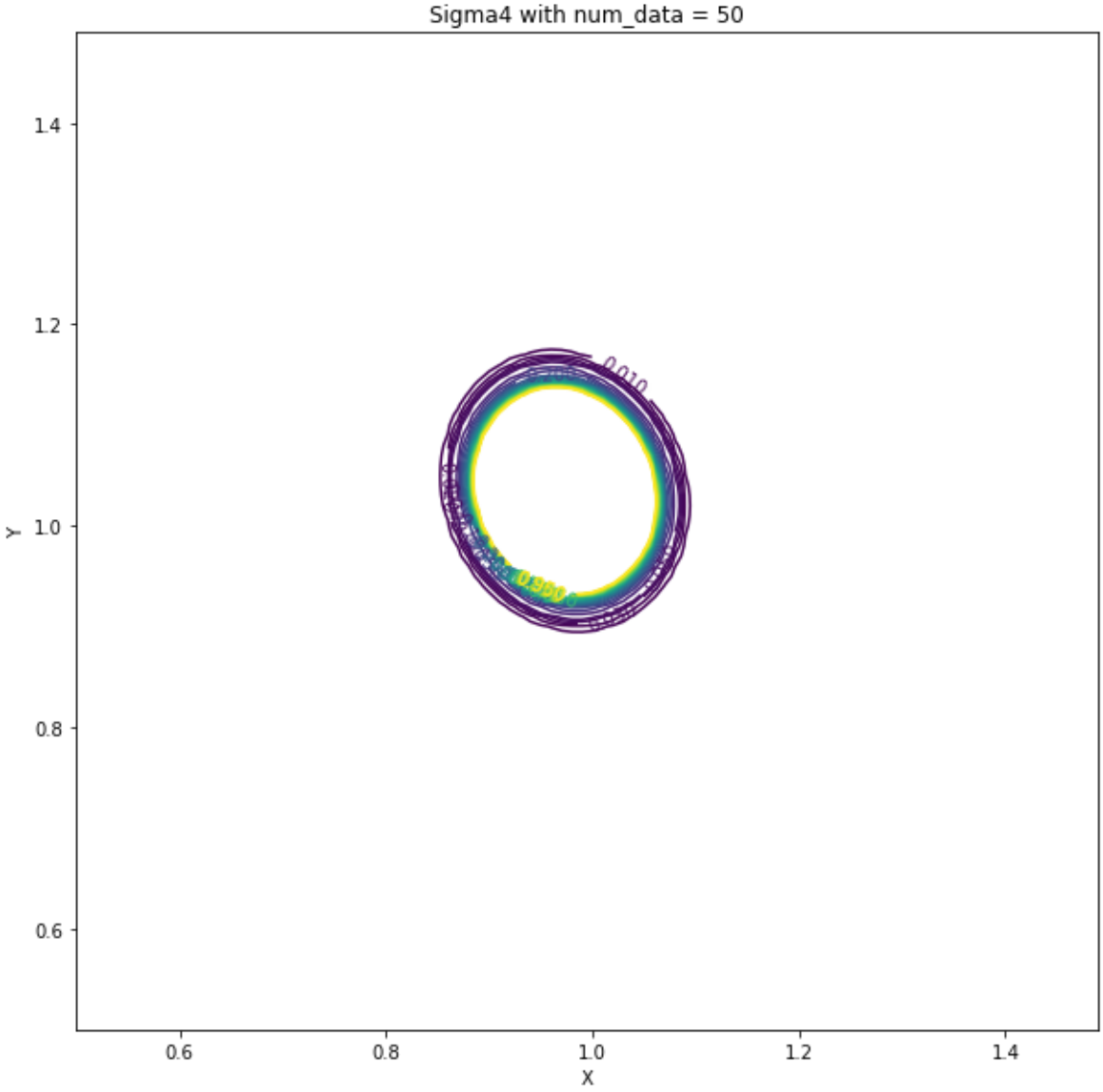

Sigma5 with num_data = 5
1.4
1.2
1.0
0.8
0.6
Y
0.6
0.8
1.0
1.2
1.4
X
0.100
0.050
0.020
0.010
0.150
0.200

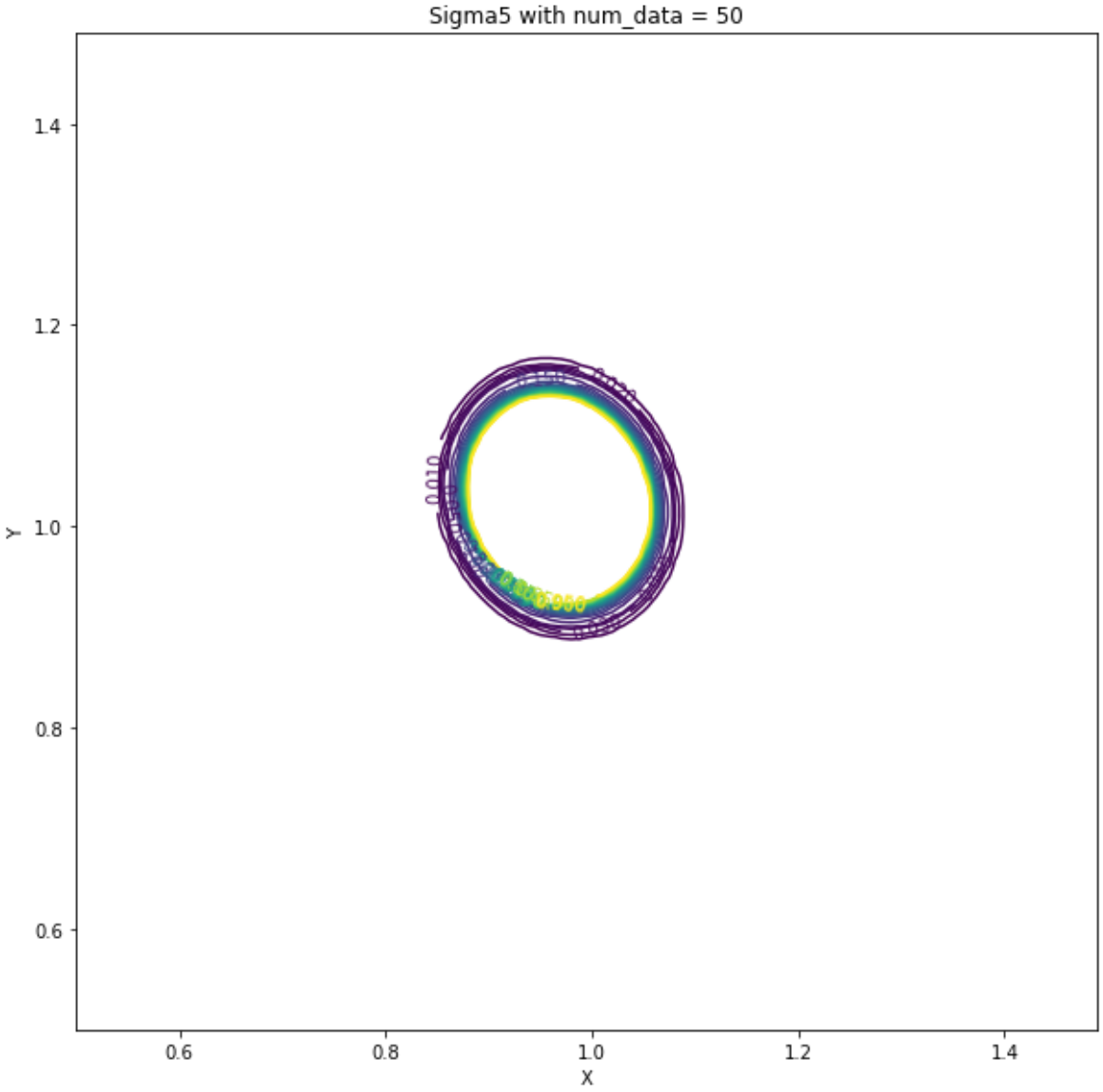

Sigma6 with num_data = 5
1.4
1.2
1.0
0.8
0.6
Y
0.6
0.8
1.0
1.2
1.4
X
0.010
0.050
0.100
0.150
0.200
0.250
0.300
0.350
0.400
0.450
0.500
0.550
0.560

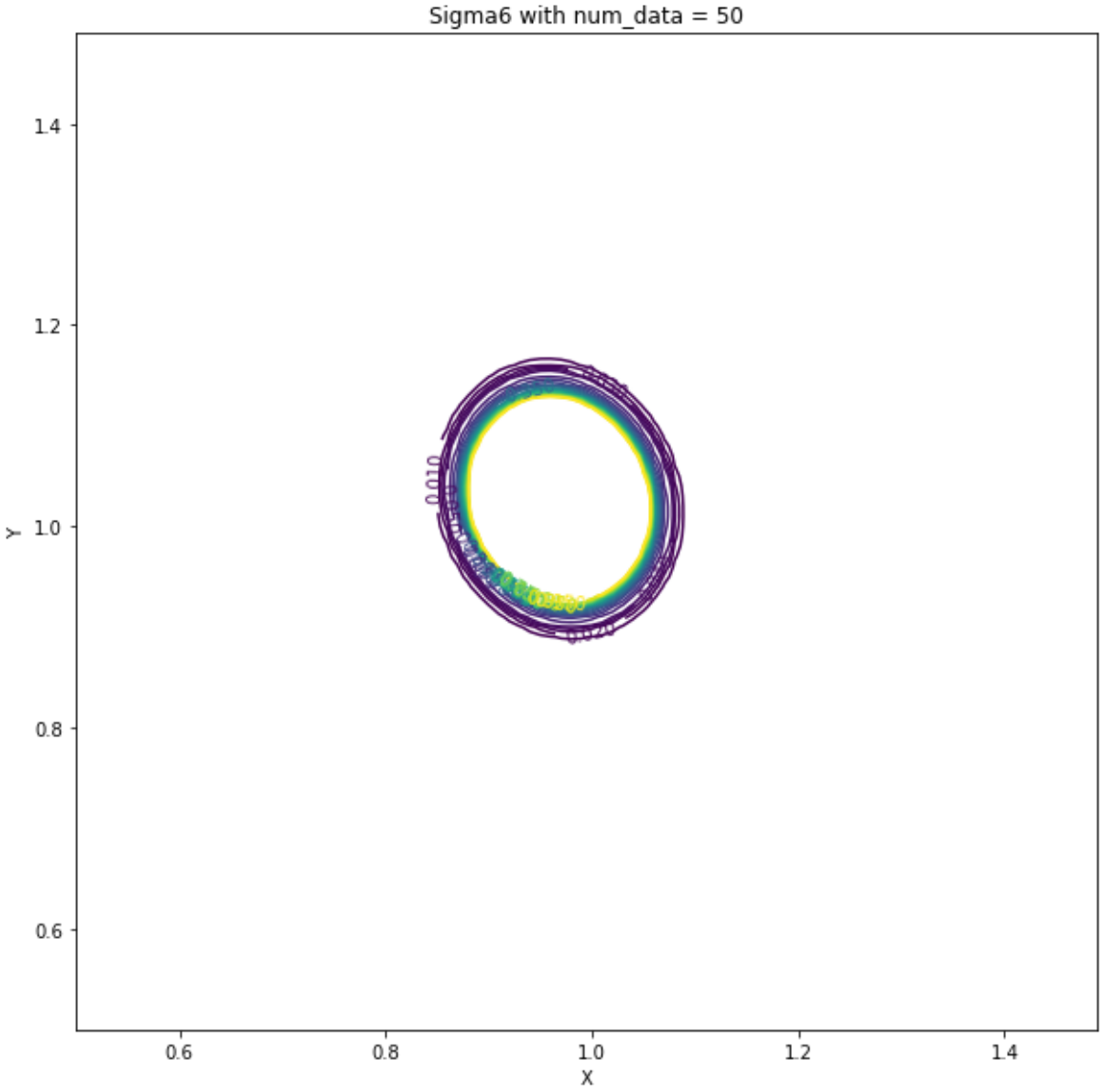

Sigma1 with num_data = 50
1.4
1.2
1.0
0.8
0.6
Y
0.6
0.8
1.0
1.2
1.4
X

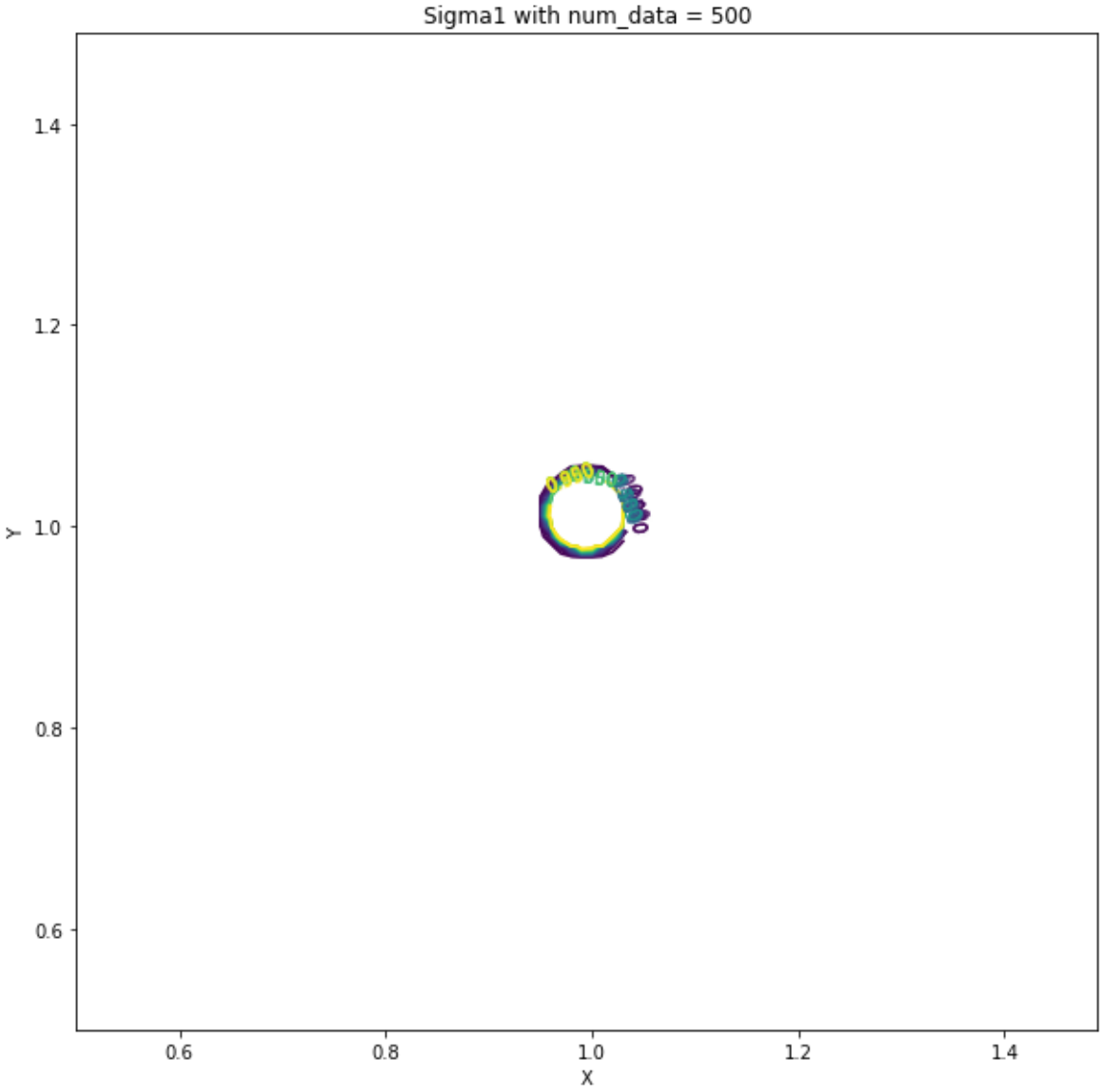

Sigma2 with num_data = 50
1.4
1.2
1.0
0.8
0.6
Y
0.6
0.8
1.0
1.2
1.4
X

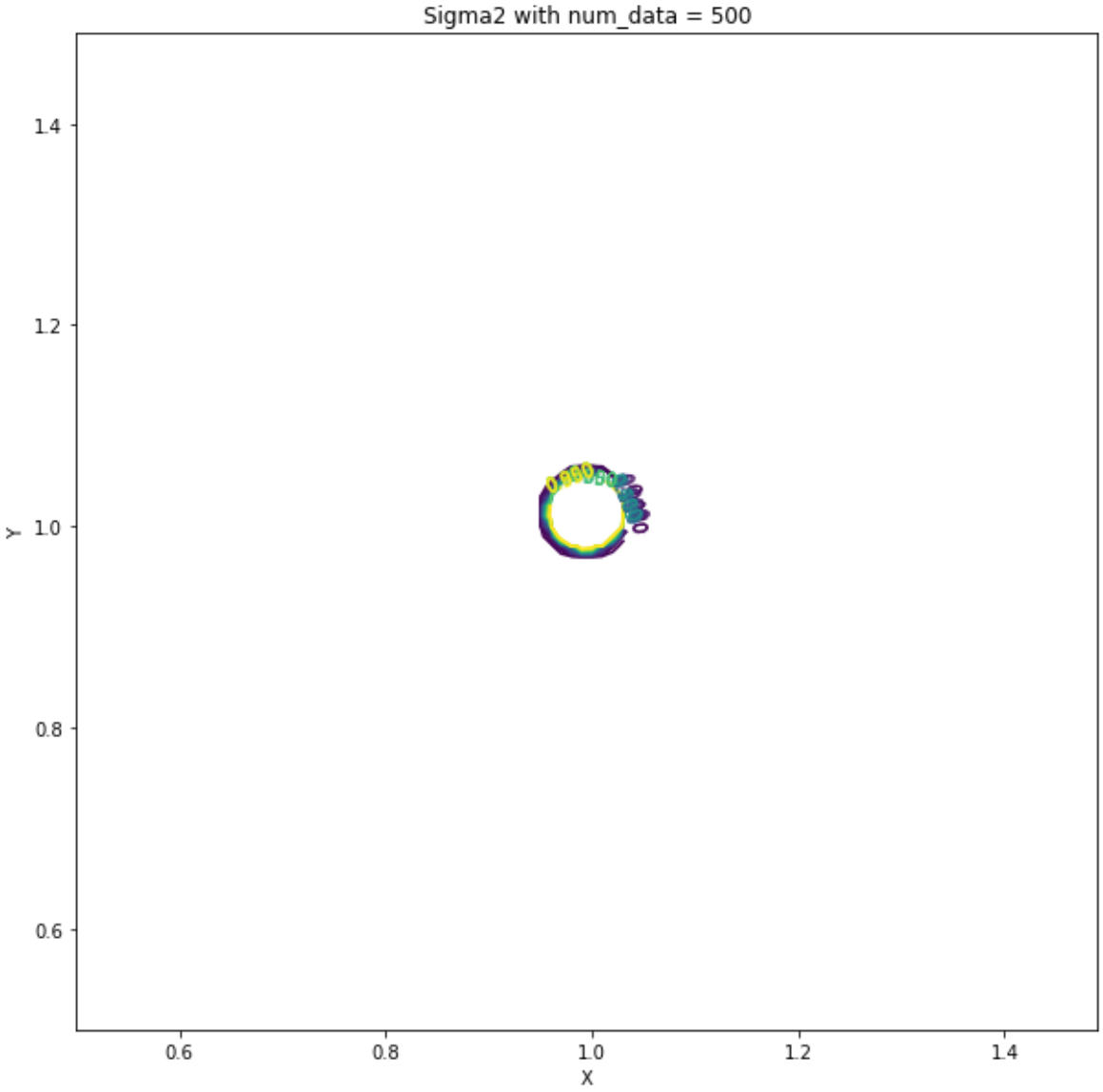

Sigma3 with num_data = 50
1.4
1.2
1.0
0.8
0.6
Y
0.6
0.8
1.0
1.2
1.4
X

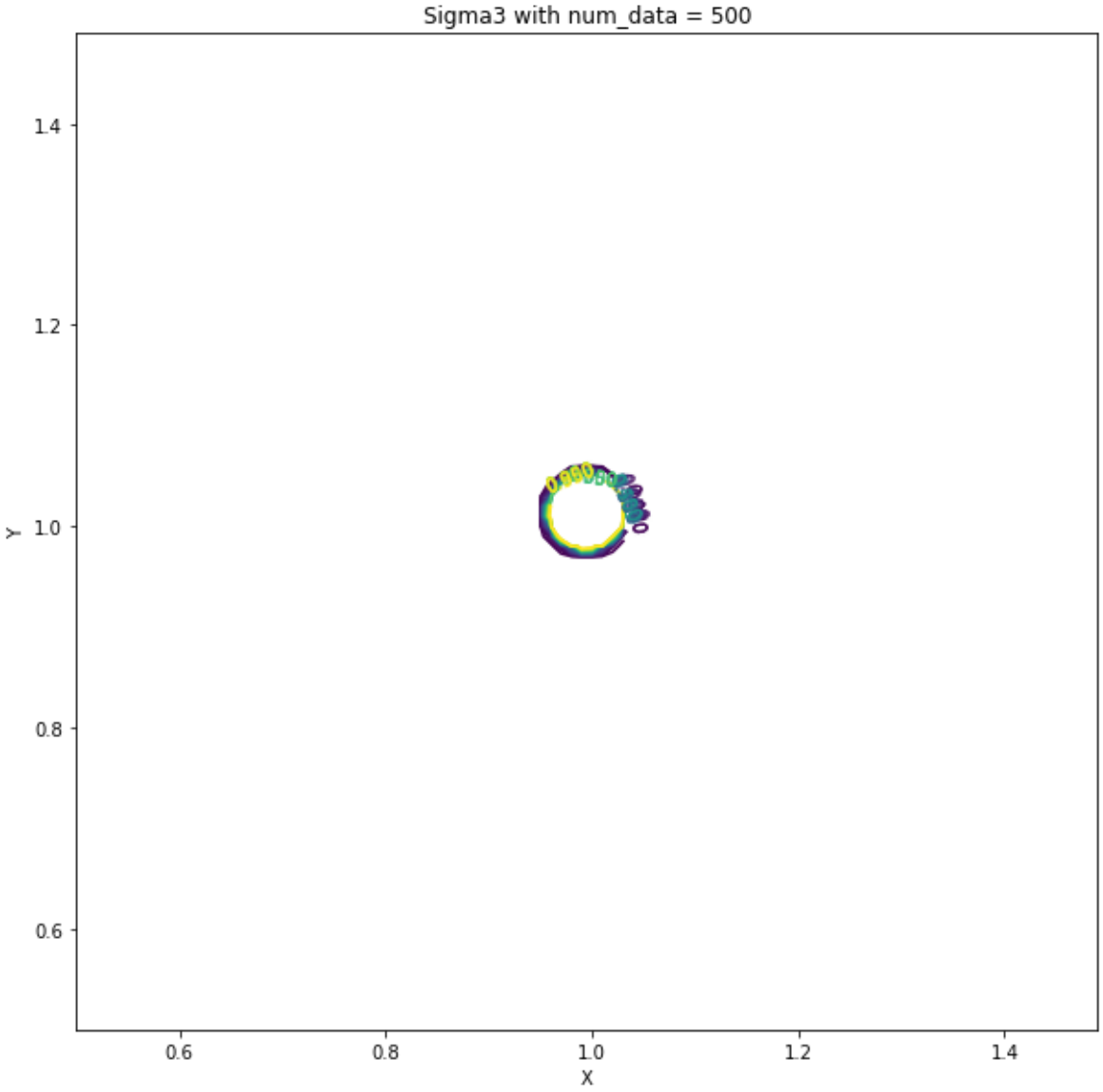

Sigma4 with num_data = 50
1.4
1.2
1.0
0.8
0.6
Y
0.6
0.8
1.0
1.2
1.4
X

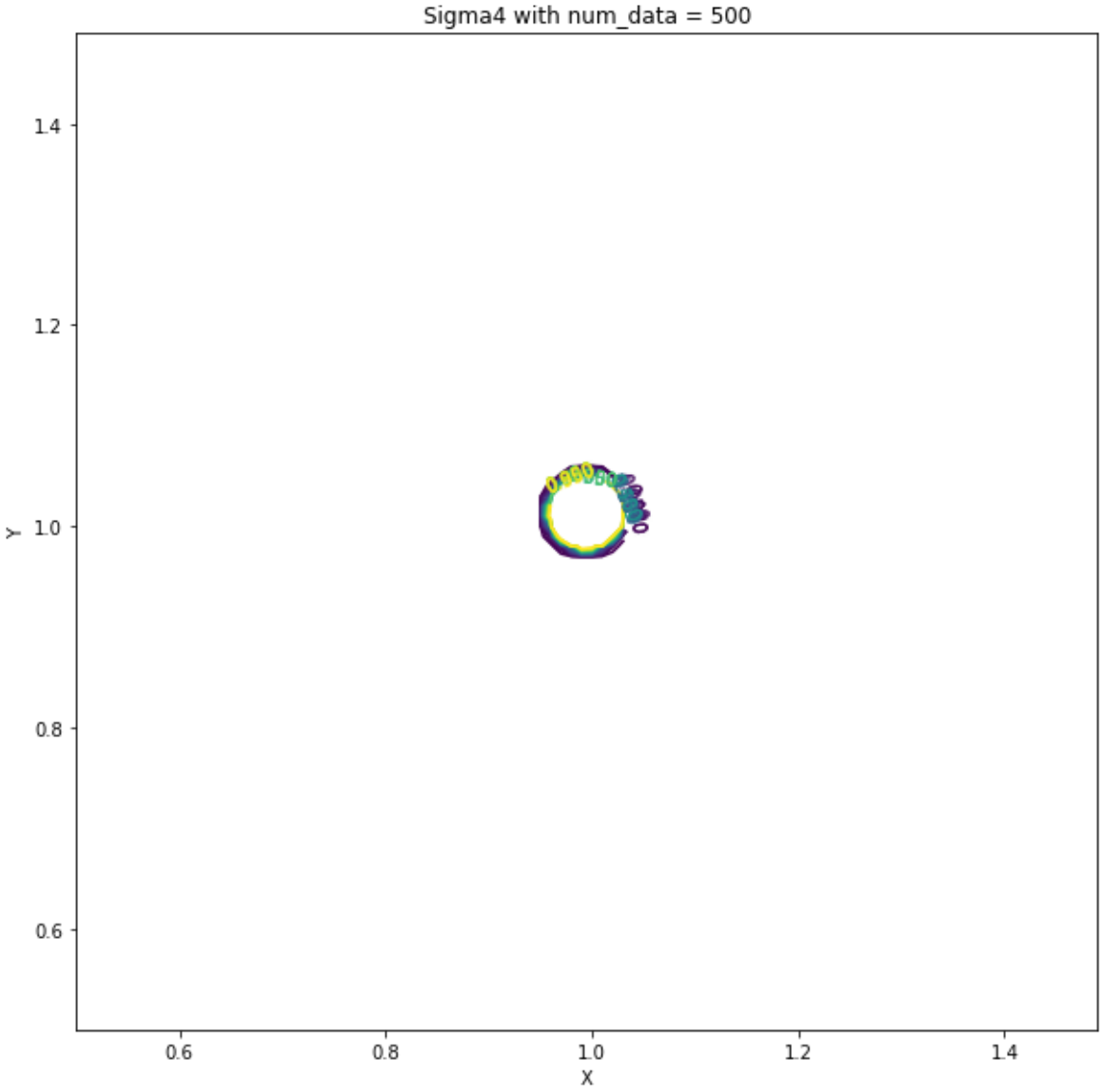

Sigma5 with num_data = 50
1.4
1.2
1.0
0.8
0.6
Y
0.6
0.8
1.0
1.2
1.4
X

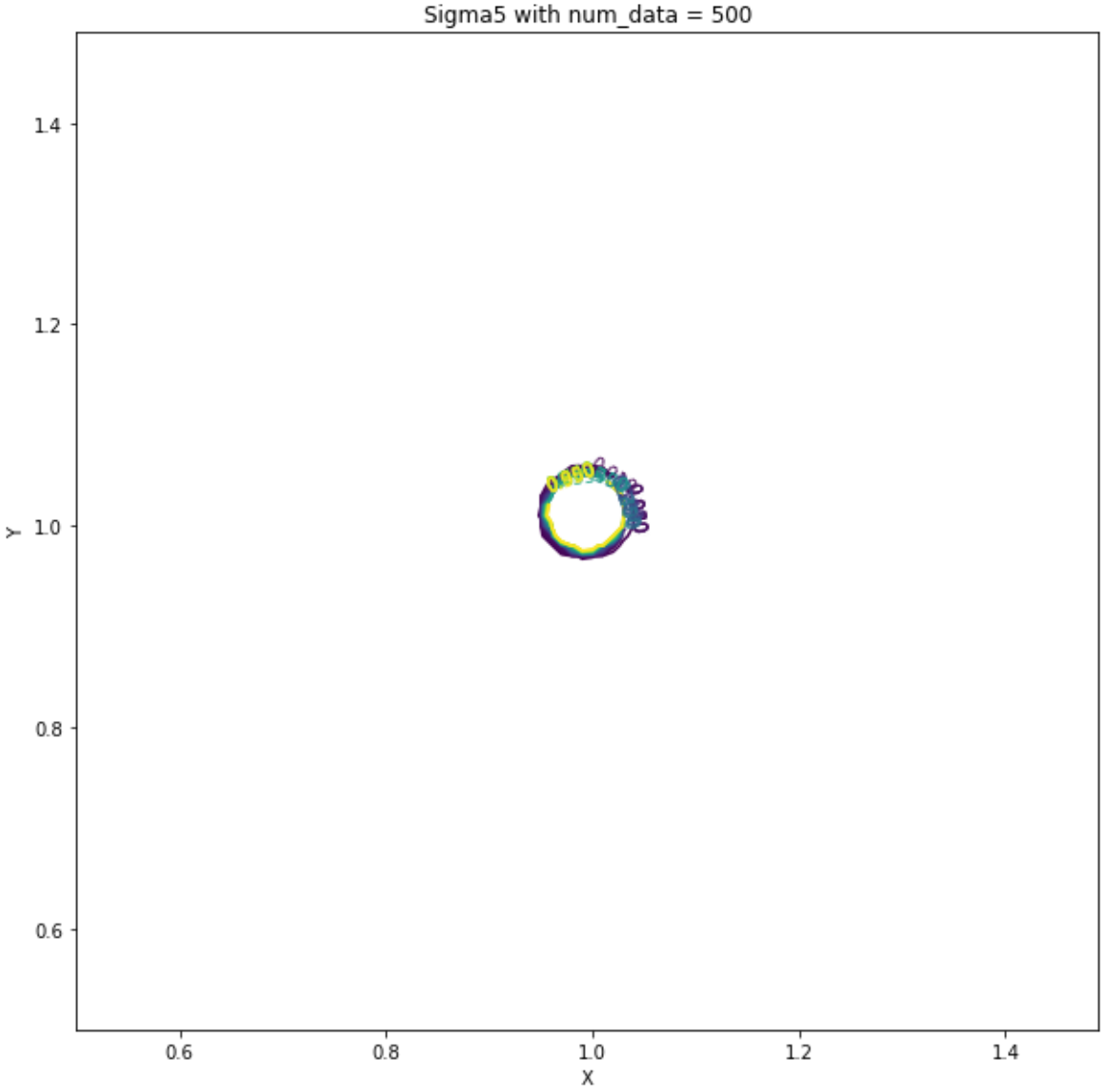

Sigma6 with num_data = 50
1.4
1.2
1.0
0.8
0.6
Y
0.6
0.8
1.0
1.2
1.4
X

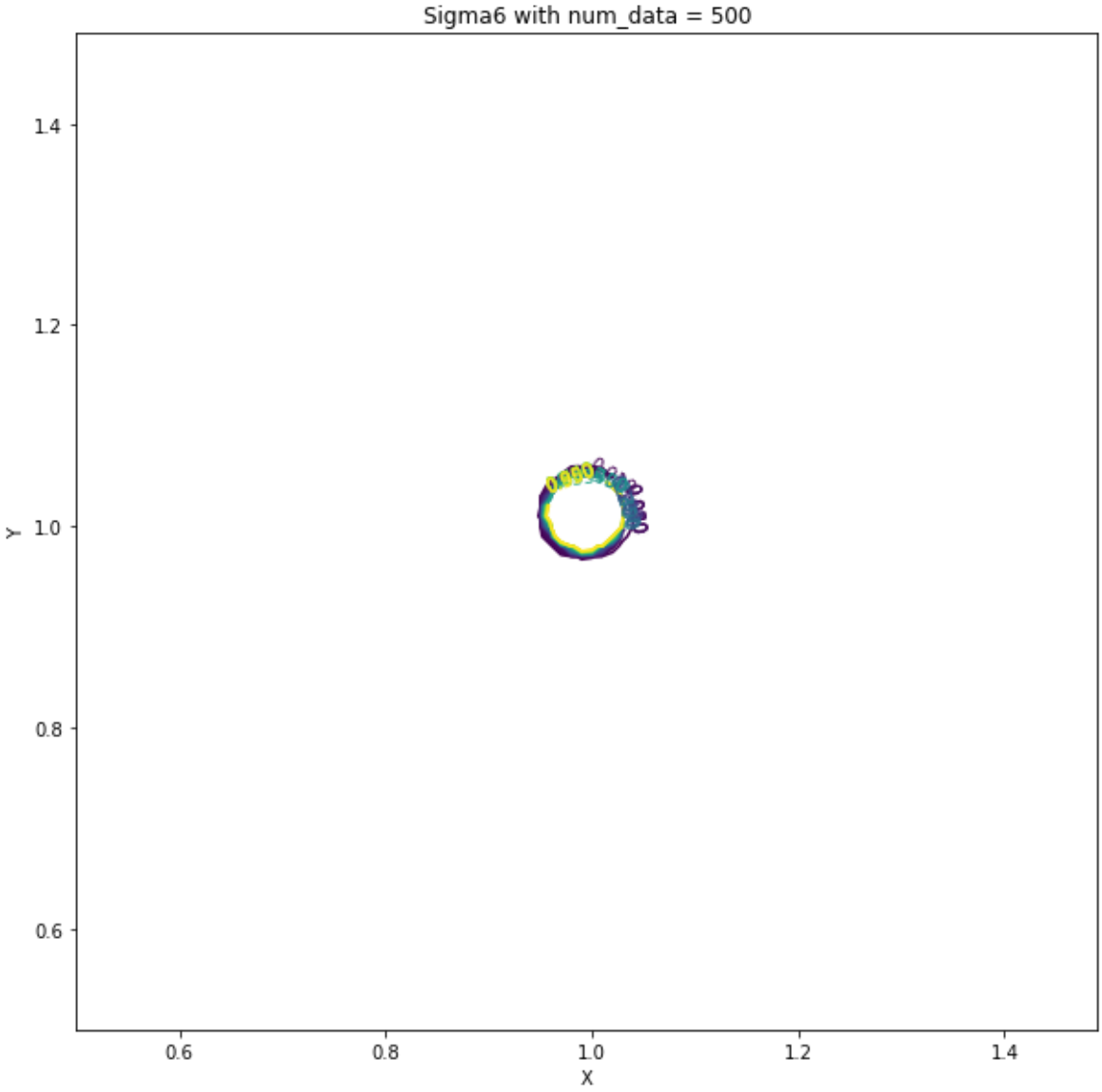

Sigma1 with num_data = 500
1.4
1.2
1.0
0.8
0.6
Y
0.6
0.8
1.0
1.2
1.4
X


Sigma2 with num_data = 500
1.4
1.2
1.0
0.8
0.6
Y
0.6
0.8
1.0
1.2
1.4
X


Sigma3 with num_data = 500
1.4
1.2
1.0
0.8
0.6
Y
0.6
0.8
1.0
1.2
1.4
X


Sigma4 with num_data = 500
1.4
1.2
1.0
0.8
0.6
Y
0.6
0.8
1.0
1.2
1.4
X


Sigma5 with num_data = 500
1.4
1.2
1.0
0.8
0.6
Y
0.6
0.8
1.0
1.2
1.4
X



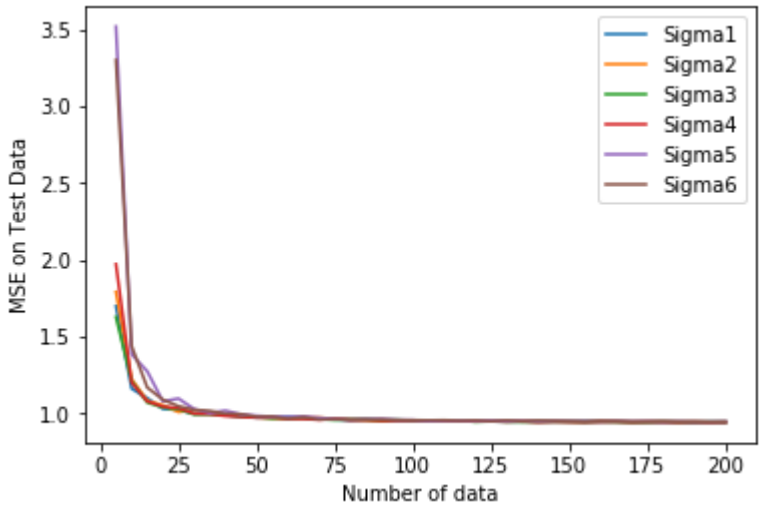
As the number of data points increases the covariance decreases. This can be observed from our outputs for each of the sigmas 1 through 6.

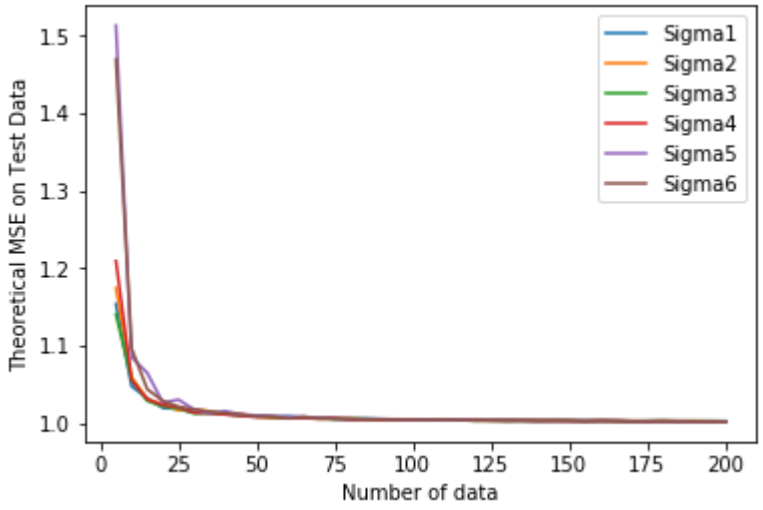
3. E

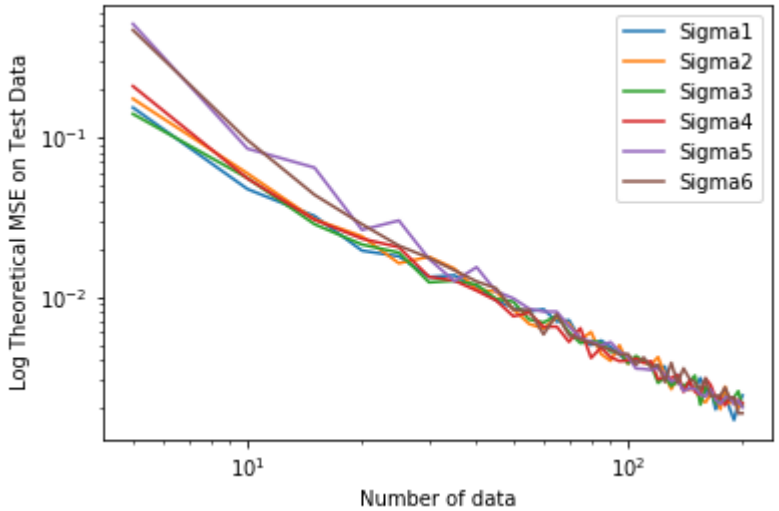
In [41]:
```
np.random.seed(0)
w = [1.0,1.0]
n_test = 500
n_trains = np.arange(5,205,5)
n_trails = 100

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 compute_mse(X,Y, w): # Empirical
    """
```

```
    This function computes MSE given data and estimated w.
    """

    mse = np.sum(np.square(Y - X.dot(w)))/len(Y)
    return mse

def compute_theoretical_mse(w):
    """
    This function computes theoretical MSE given estimated w.
    """
    #TODO implement this
    theoretical_mse = 5*(w[0] - 1)**2 + 5*(w[1] - 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 = tikhonov_regression(x_train, y_train, Sigma)
            empirical_mse = compute_mse(X_test, y_test, w)
            theoretical_mse = compute_theoretical_mse(w)
            mses[i, j, seed] = empirical_mse
            theoretical_mses[i, j, seed] = theoretical_mse



# 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 = n
ames[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('Log Theoretical MSE on Test Data')
plt.legend()
plt.savefig('log_theoretical_MSE.png')
```







As the amount of training data increases our MSE, both theoretical and empirical, approach the same value. The goodness of our priors appears to be come less important as the amount of training data increases.

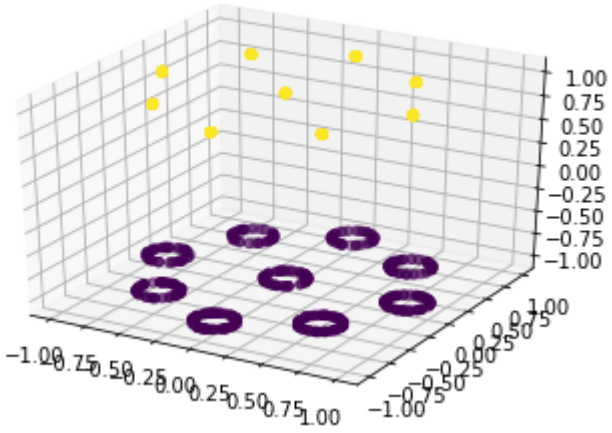
5 Kernel Ridge Regression

In [98]:
```
import matplotlib.pyplot as plt
from matplotlib import cm
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
from scipy.special import comb
import itertools

LAMBDA = 0.001

# choose the data you want to load
data = np.load('circle.npz')
# data = np.load('heart.npz')
# data = np.load('asymmetric.npz')

SPLIT = 0.8
X = data["x"]
y = data["y"]
X /= np.max(X)  # normalize the data

n_train = int(X.shape[0] * SPLIT)
X_train = X[:n_train:, :]
X_valid = X[n_train:, :]
y_train = y[:n_train]
y_valid = y[n_train:]

LAMBDA = 0.001


def lstsq(A, b, lambda_=0):
    return np.linalg.solve(A.T @ A + lambda_ * np.eye(A.shape[1]), A.T
@ b)


def heatmap(f, X, y, clip=5):
 # example: heatmap(lambda x, y: x * x + y * y)
 # clip: clip the function range to [-clip, clip] to generate a clean p
lot
 # set it to zero to disable this function
    xx = yy = np.linspace(np.min(X), np.max(X), 72)
    x0, y0 = np.meshgrid(xx, yy)
    x0, y0 = x0.ravel(), y0.ravel()
    z0 = f(x0, y0)

    if clip:
        z0[z0 > clip] = clip
        z0[z0 < -clip] = -clip
```

```
    plt.hexbin(x0, y0, C=z0, gridsize=50, cmap=cm.jet, bins=None)
    plt.colorbar()
    cs = plt.contour(xx, yy, z0.reshape(xx.size, yy.size), [-2, -1, -0.
5, 0, 0.5, 1,2], cmap=cm.jet)
    plt.clabel(cs, inline=1, fontsize=10)

    pos = y[:] == +1.0
    neg = y[:] == -1.0
    plt.scatter(X[pos, 0], X[pos, 1], c='red', marker='+')
    plt.scatter(X[neg, 0], X[neg, 1], c='blue', marker='v')
    plt.show()
```

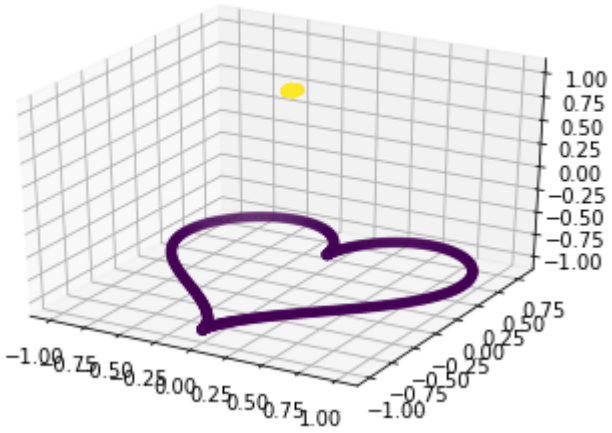
In [92]:
```
#5.A

# choose the data you want to load
data = np.load('circle.npz')
SPLIT = 0.8
Xc = data["x"]
yc = data["y"]
Xc /= np.max(Xc) # normalize the data

n_train_c = int(Xc.shape[0] * SPLIT)
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(Xc[:,0], Xc[:,1], yc, c=yc)
```

Out[92]: <mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x7fd1d166abe0>



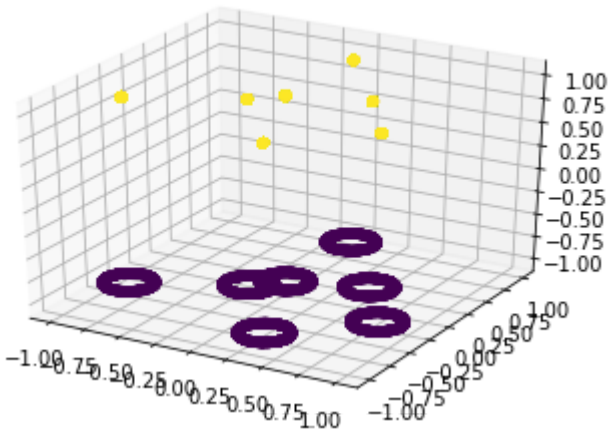
In [93]:
```
data = np.load('heart.npz')
SPLIT = 0.8
Xh = data["x"]
yh = data["y"]
Xh /= np.max(Xh) # normalize the data
n_train_h = int(Xh.shape[0] * SPLIT)
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(Xh[:,0], Xh[:,1], yh, c=yh)
```

Out[93]: <mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x7fd1cfdb4ac8>



In [94]:
```
data = np.load('asymmetric.npz')
SPLIT = 0.8
Xa = data["x"]
ya = data["y"]
Xa /= np.max(Xa) # normalize the data
n_train_a = int(Xa.shape[0] * SPLIT)
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(Xa[:,0], Xa[:,1], ya, c=ya)
```

Out[94]: <mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x7fd1d05cd470>



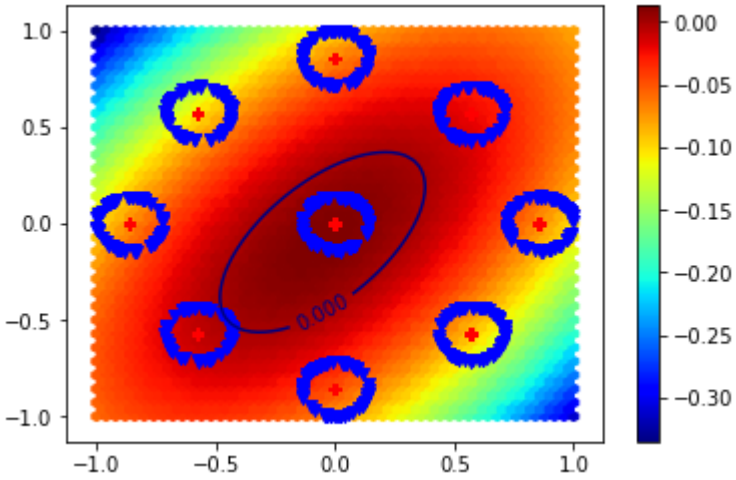
5.b

In [95]:
```
def partitions(n, b):
    masks = np.identity(b, dtype=int)
    for c in itertools.combinations_with_replacement(masks, n):
        yield sum(c)

def compute_multivariates(features, degree):
    if degree == 0:
        return [1]
    powers = partitions(degree, len(features))
    results = []
    for deg in powers:
        total = 1
        for i in range(len(deg)):
            total = total*(features[i]**deg[i])
        results.append(total)
    results.extend(compute_multivariates(features, degree-1))
    return np.array(results)

def lstsqL(A, b, lambda_):
    return np.linalg.solve((A.T @ A) + np.eye(len(A[0]))*lambda_, A.T @
 b)

l = 0.001


all_multivariates_Xc = np.array([compute_multivariates(x, 16) for x in
Xc])
all_multivariates_Xh = np.array([compute_multivariates(x, 16) for x in
Xh])
all_multivariates_Xa = np.array([compute_multivariates(x, 16) for x in
Xa]
                                )
Xc_train_mult = all_multivariates_Xc[:n_train_c:, :]
Xc_valid_mult = all_multivariates_Xc[n_train_c:, :]

Xh_train_mult = all_multivariates_Xh[:n_train_h:, :]
Xh_valid_mult = all_multivariates_Xh[n_train_h:, :]

Xa_train_mult = all_multivariates_Xa[:n_train_a:, :]
Xa_valid_mult = all_multivariates_Xa[n_train_a:, :]

yc_train = yc[:n_train_c]
yc_valid = yc[n_train_c:]

yh_train = yh[:n_train_h]
yh_valid = yh[n_train_h:]

ya_train = ya[:n_train_a]
ya_valid = ya[n_train_a:]


Xc_train_reg = Xc[:n_train_c:, :]
```

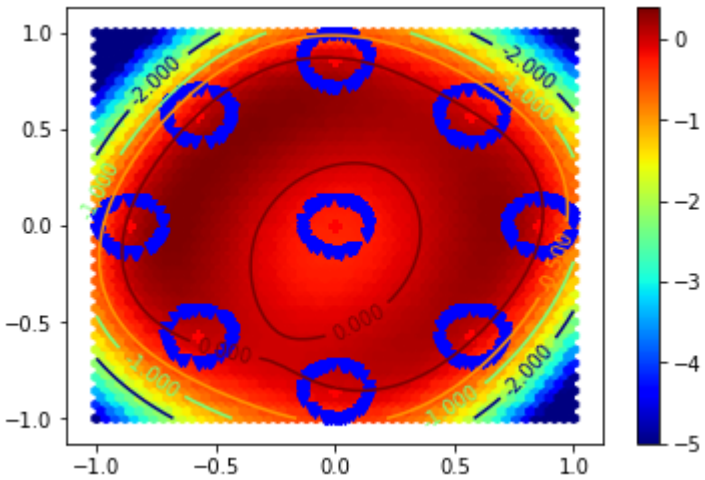
```
Xc_valid_reg = Xc[n_train_c:, :]

Xh_train_reg = Xh[:n_train_h:, :]
Xh_valid_reg = Xh[n_train_h:, :]

Xa_train_reg = Xa[:n_train_a:, :]
Xa_valid_reg = Xa[n_train_a:, :]
```

In [99]:
```
def f(x, y, p):
    data = np.array([x, y]).T
    return np.array([compute_multivariates(x_, p) for x_ in data])

for p in range(1, 17):

    start_of_features = comb(2 + p, 2, exact=True)

    Xc_train = Xc_train_mult[:, -start_of_features:]
    Xc_valid = Xc_valid_mult[:, -start_of_features:]

    w = lstsqL(Xc_train, yc_train, l)

    err_train = np.sum(np.square((yc_train - Xc_train.dot(w)))) / len(y
c_train)
    err_test = np.sum(np.square((yc_valid - Xc_valid.dot(w)))) / len(yc
_valid)

    print('p:', p, ', err_train:', err_train, ', err_test:', err_test)

    if p%2 == 0:

        heatmap(lambda x, y: f(x, y, p).dot(w), Xc, yc)
```

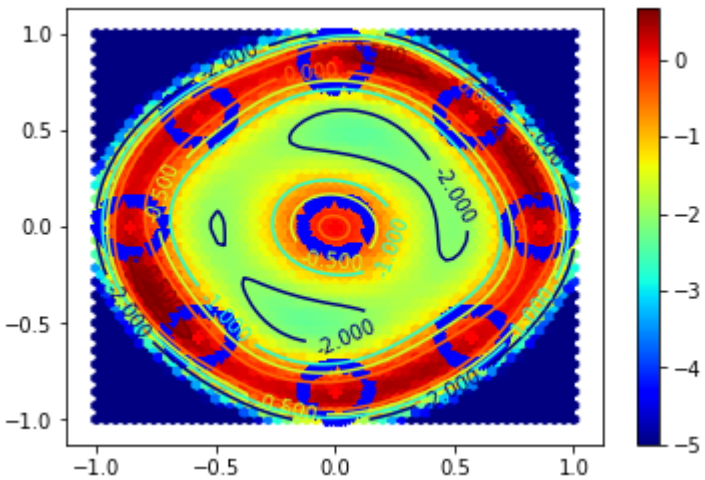
```
p: 1 , err_train: 0.9970876133159098 , err_test: 0.9975785898043925
p: 2 , err_train: 0.9955368301850909 , err_test: 1.001056114431042
```



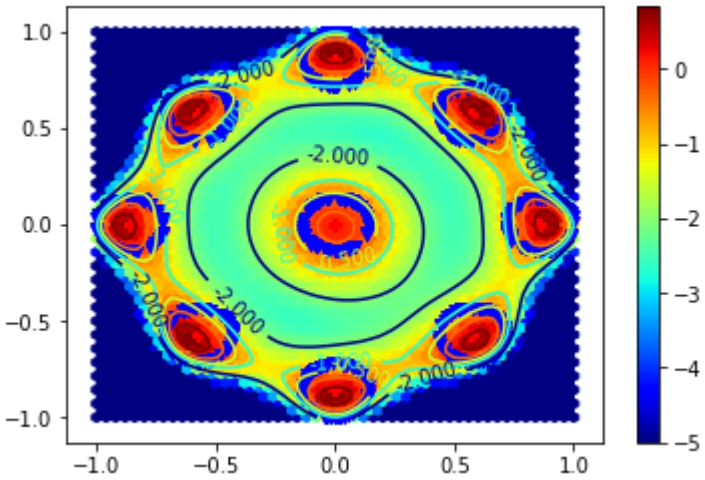

```
p: 3 , err_train: 0.9926989379391203 , err_test: 1.0193501224786765
p: 4 , err_train: 0.9430114480896957 , err_test: 0.9979140428312968
```



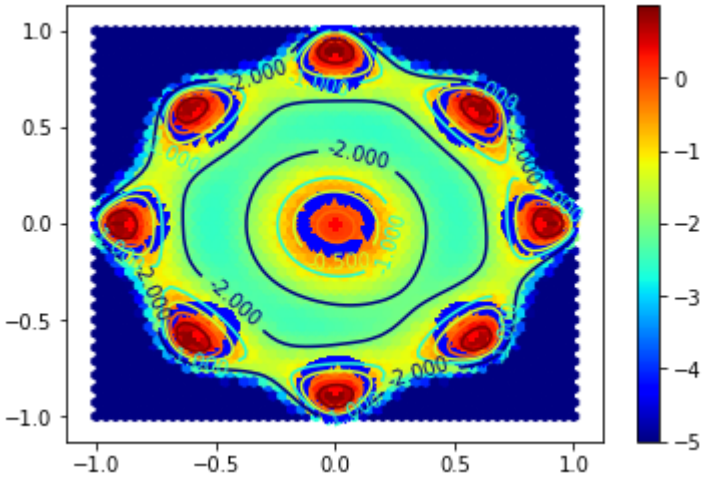

```
p: 5 , err_train: 0.9355498484866808 , err_test: 1.0285965870845089
p: 6 , err_train: 0.5471551370040443 , err_test: 0.5856875932674246
```



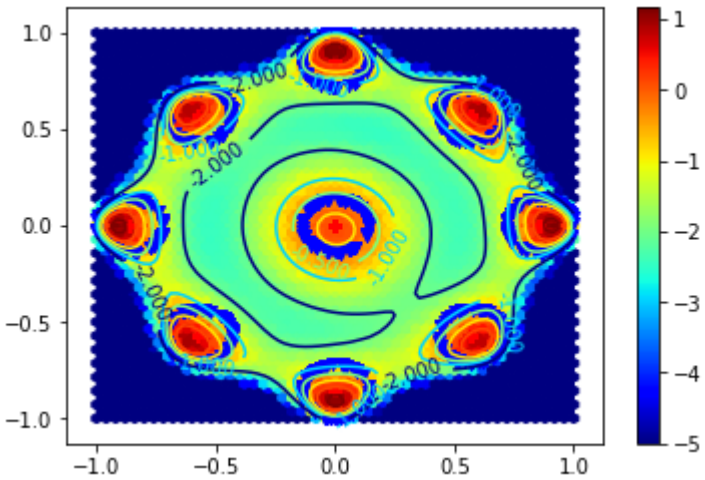

```
p: 7 , err_train: 0.5450153735010657 , err_test: 0.5820071075131588
p: 8 , err_train: 0.23019031845941818 , err_test: 0.2499900445424043
```



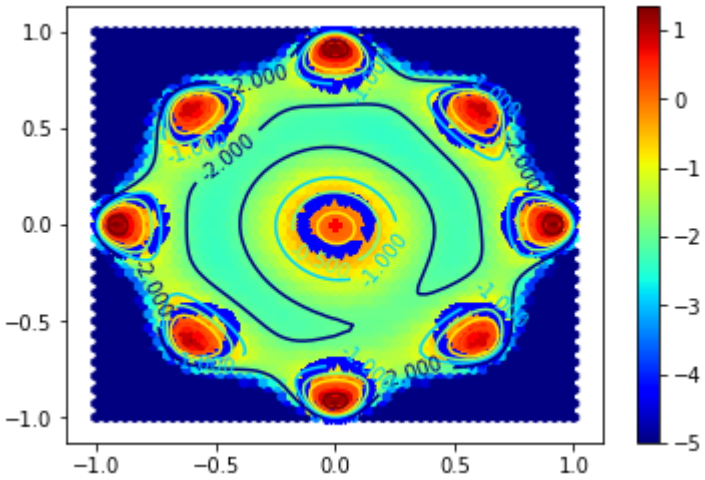

```
p: 9 , err_train: 0.22976015143770662 , err_test: 0.25113476414101443
p: 10 , err_train: 0.17427302934821742 , err_test: 0.19299805978605986
```



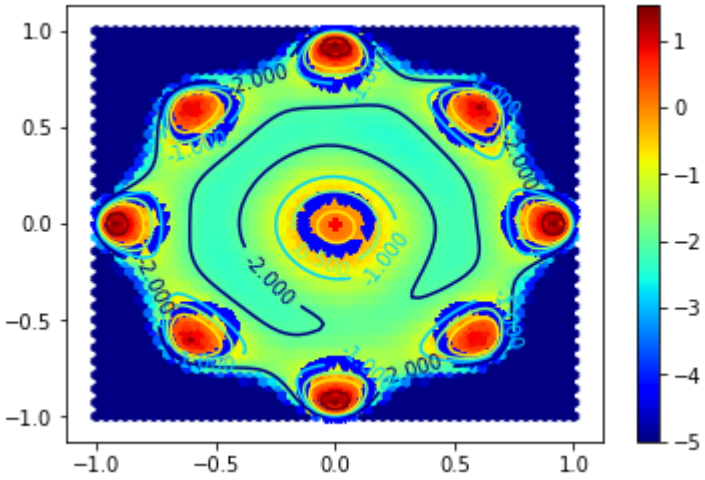

```
p: 11 , err_train: 0.17407314778499966 , err_test: 0.19329666070563614
p: 12 , err_train: 0.1567228043261502 , err_test: 0.17533526394090573
```




```
p: 13 , err_train: 0.15667106189811006 , err_test: 0.17548995194148045
p: 14 , err_train: 0.14578655477947844 , err_test: 0.16474477919287348
```

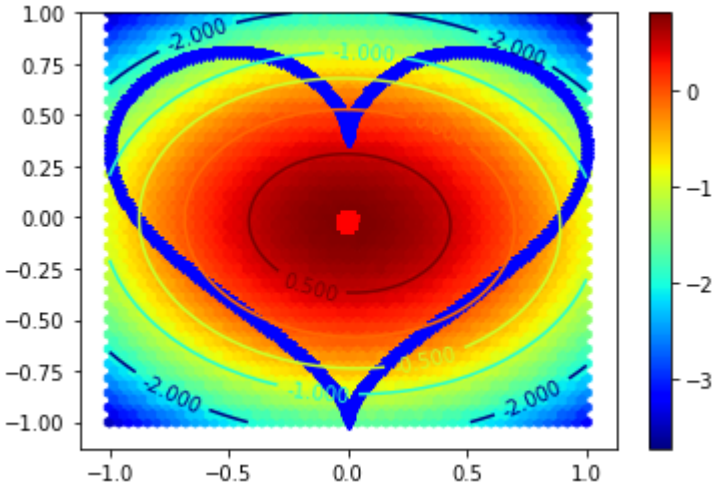



```
p: 15 , err_train: 0.14578368356266688 , err_test: 0.16481623671627252
p: 16 , err_train: 0.13919818279558657 , err_test: 0.15895198216320244
```

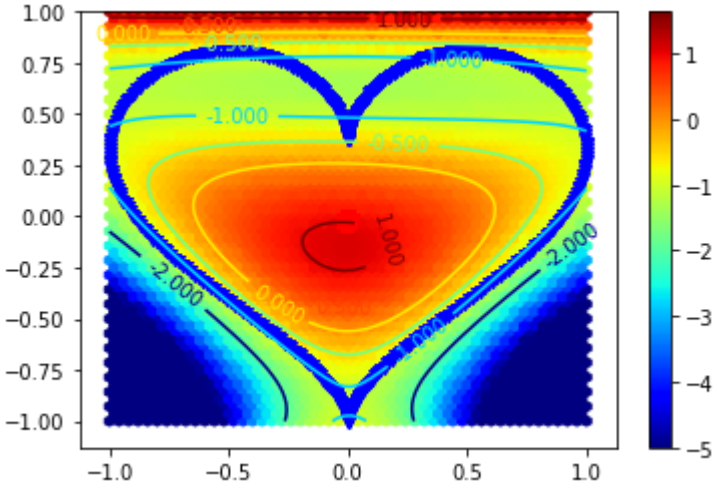


In [100]:
```
for p in range(1, 17):

    start_of_features = comb(2 + p, 2, exact=True)

    Xh_train = Xh_train_mult[:, -start_of_features:]
    Xh_valid = Xh_valid_mult[:, -start_of_features:]

    w = lstsqL(Xh_train, yh_train, l)

    err_train = np.sum(np.square((yh_train - Xh_train.dot(w)))) / len(y
h_train)
    err_test = np.sum(np.square((yh_valid - Xh_valid.dot(w)))) / len(yh
_valid)

    print('p:', p, ', err_train:', err_train, ', err_test:', err_test)

    if p%2 == 0:

        heatmap(lambda x, y: f(x, y, p).dot(w), Xh, yh)
```

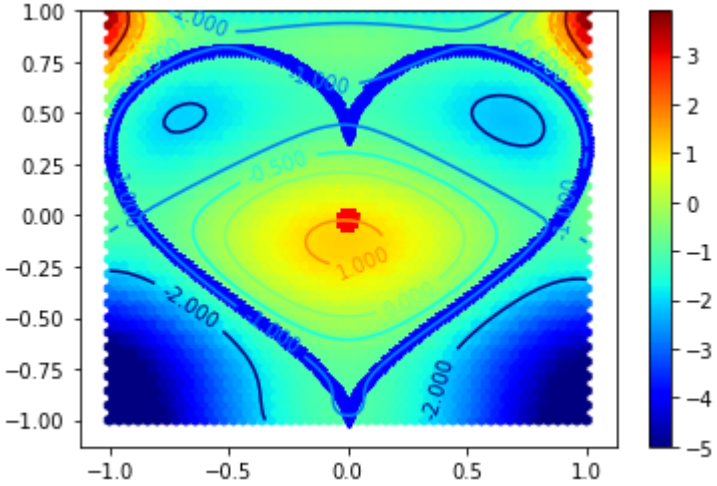
```
p: 1 , err_train: 0.9626430338740402 , err_test: 0.9599523272662586
p: 2 , err_train: 0.23671821257487874 , err_test: 0.18983714832915385
```



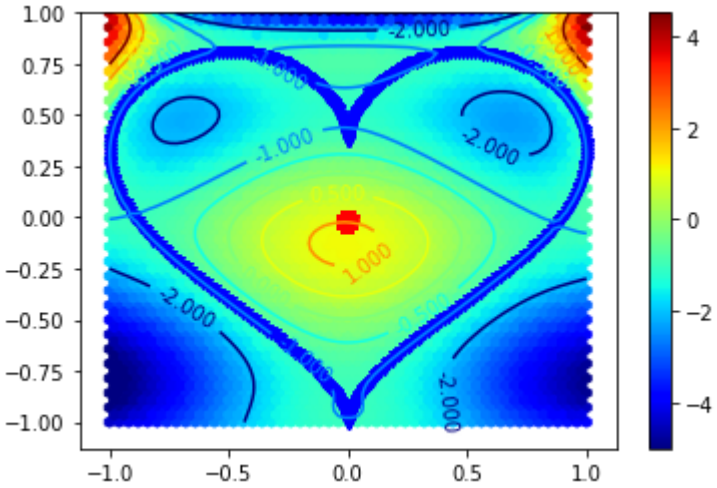

```
p: 3 , err_train: 0.11548077719357827 , err_test: 0.09080121300292604
p: 4 , err_train: 0.012169269479436938 , err_test: 0.009122615062870267
```



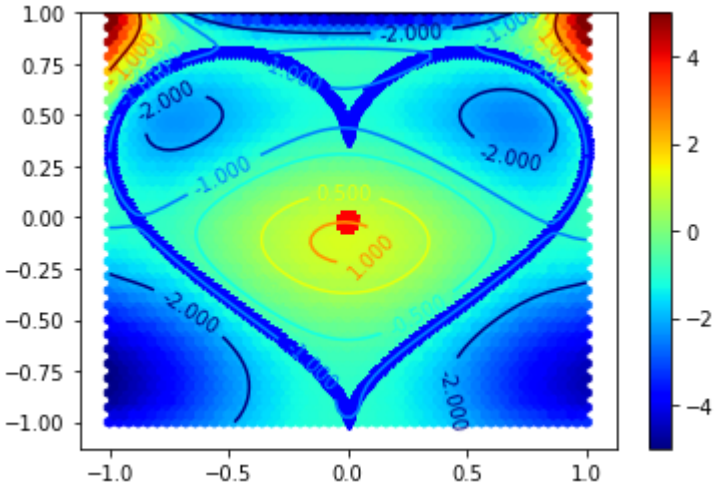

```
p: 5 , err_train: 0.005160374530276253 , err_test: 0.004102409581422235
p: 6 , err_train: 0.0026296280347262897 , err_test: 0.001857989038532558
```



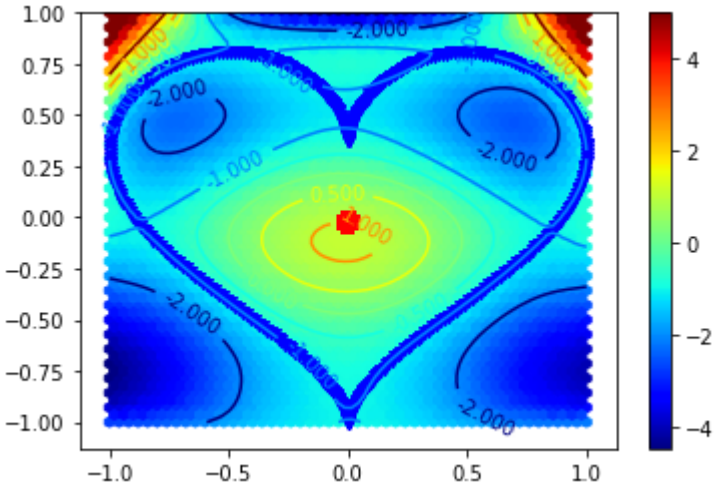
```
p: 7 , err_train: 0.0023779646776404613 , err_test: 0.0016441536755293
557
p: 8 , err_train: 0.00235380320456882 , err_test: 0.001639654274965594
5
```



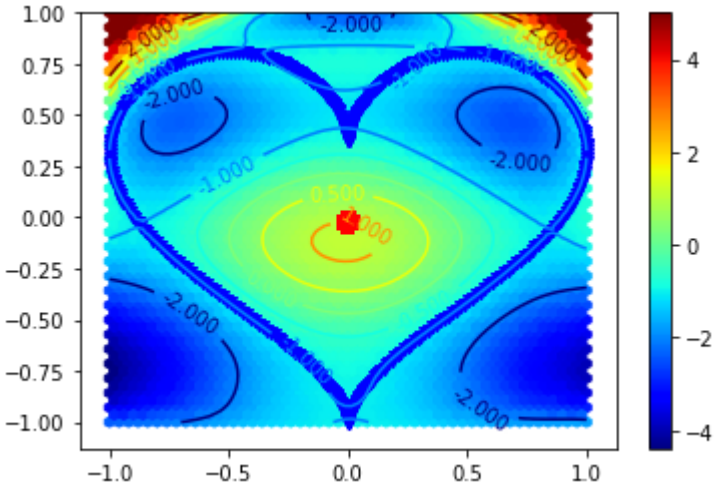

```
p: 9 , err_train: 0.0023210708791157014 , err_test: 0.0016087263829643
495
p: 10 , err_train: 0.0021933463362215143 , err_test: 0.001500351093887
8238
```



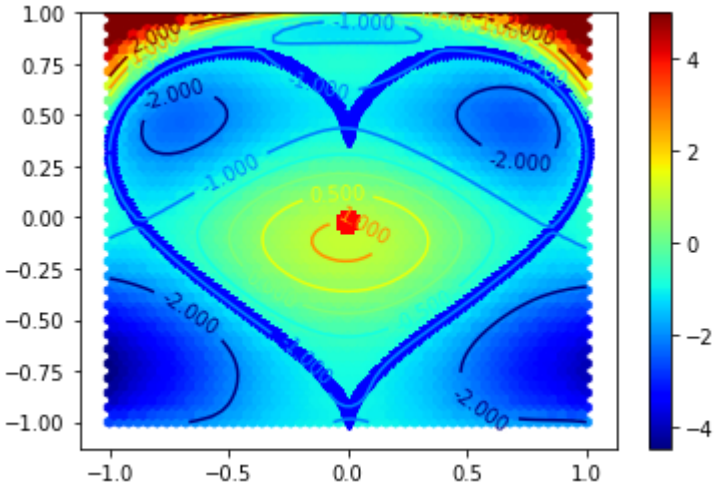

```
p: 11 , err_train: 0.002183604986854652 , err_test: 0.0014878771101232
168
p: 12 , err_train: 0.0020904468867542674 , err_test: 0.001413644316893
0805
```




```
p: 13 , err_train: 0.002070218387077949 , err_test: 0.0013951985148570
45
p: 14 , err_train: 0.002036091103609981 , err_test: 0.0013707220535593
664
```

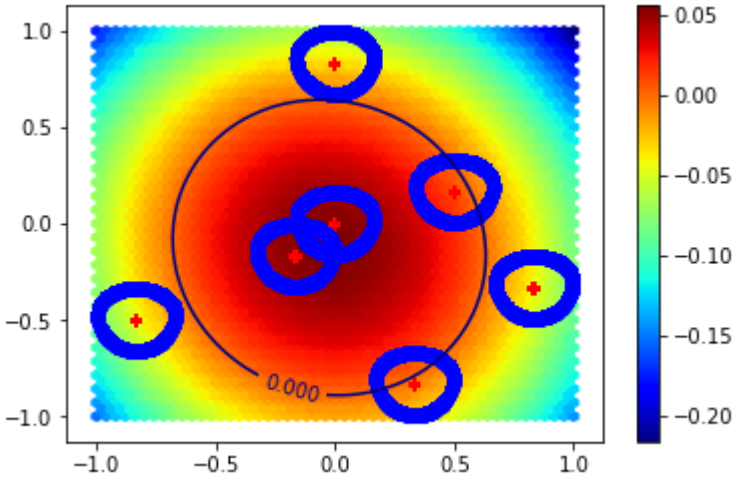



```
p: 15 , err_train: 0.0020051950660451073 , err_test: 0.001344044974020
4047
p: 16 , err_train: 0.0019982969435813607 , err_test: 0.001340150863952
8704
```

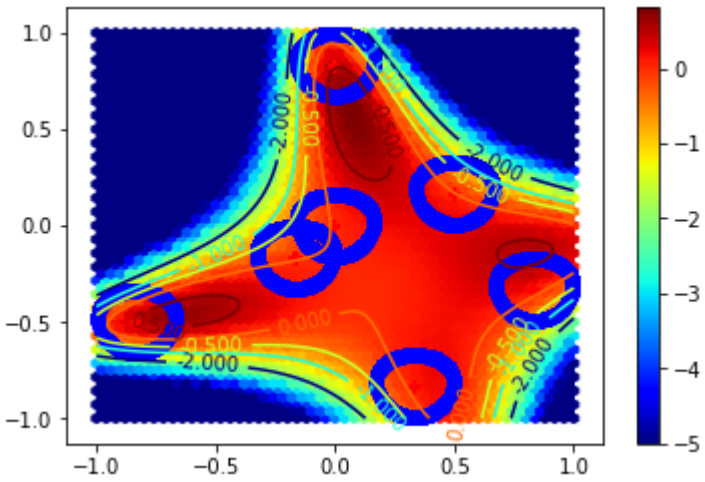


In [101]:
```
for p in range(1, 17):

    start_of_features = comb(2 + p, 2, exact=True)

    Xa_train = Xa_train_mult[:, -start_of_features:]
    Xa_valid = Xa_valid_mult[:, -start_of_features:]

    w = lstsqL(Xa_train, ya_train, l)

    err_train = np.sum(np.square((ya_train - Xa_train.dot(w)))) / len(y
a_train)
    err_test = np.sum(np.square((ya_valid - Xa_valid.dot(w)))) / len(ya
_valid)

    print('p:', p, ', err_train:', err_train, ', err_test:', err_test)

    if p%2 == 0:

        heatmap(lambda x, y: f(x, y, p).dot(w), Xa, ya)
```

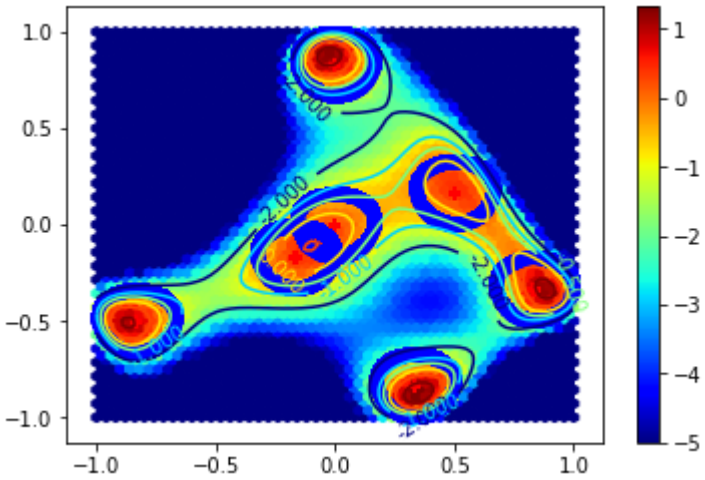
```
p: 1 , err_train: 0.9999889473314001 , err_test: 1.0001938547280624
p: 2 , err_train: 0.9982596085896175 , err_test: 1.0001762139936028
```



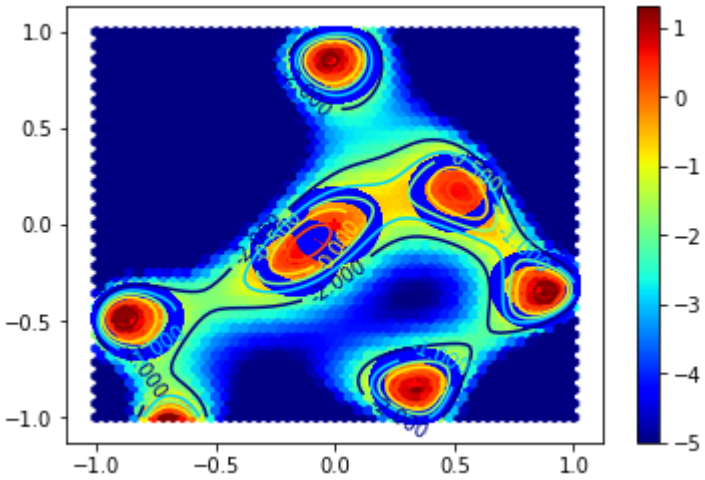

```
p: 3 , err_train: 0.9915649431277649 , err_test: 0.9913884251589582
p: 4 , err_train: 0.8286921505835065 , err_test: 0.8223688775072794
```



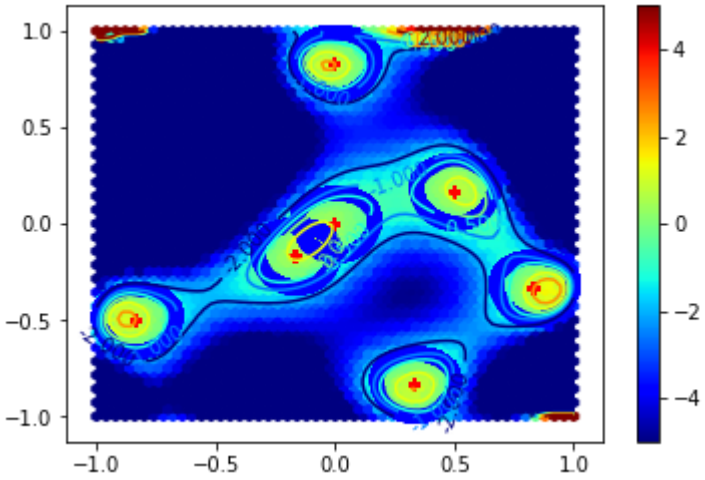

```
p: 5 , err_train: 0.758986873678395 , err_test: 0.7488111180155654
p: 6 , err_train: 0.26403992226598055 , err_test: 0.24239777871236923
```



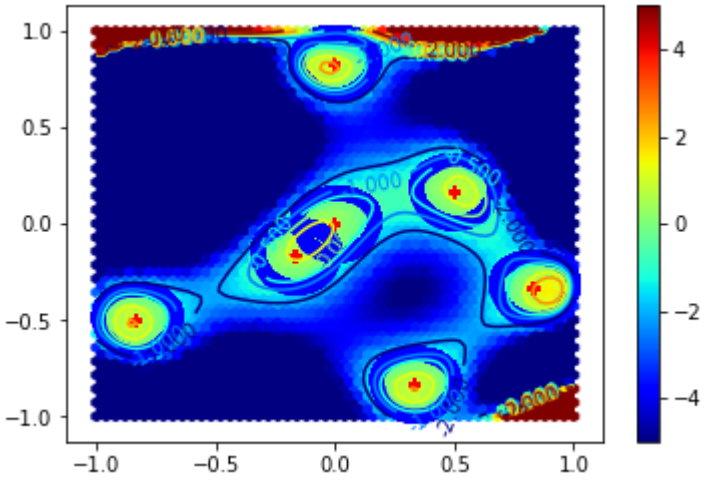

```
p: 7 , err_train: 0.22219460538584873 , err_test: 0.201139320362268
p: 8 , err_train: 0.17985325230664329 , err_test: 0.1583471579755343
```



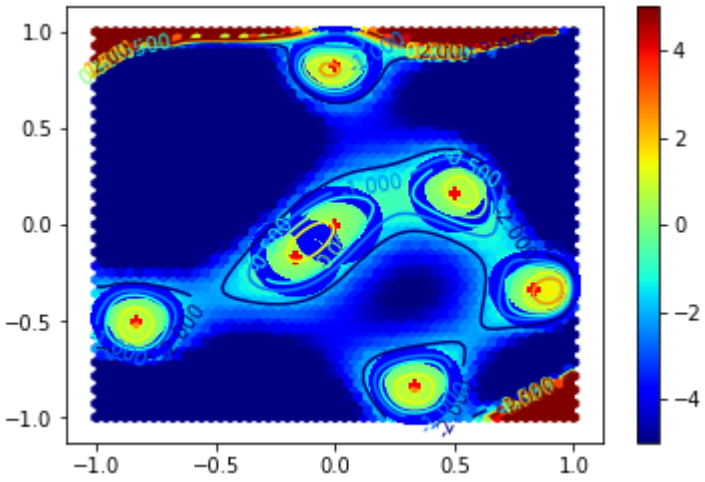
```
p: 9 , err_train: 0.16358087733792306 , err_test: 0.14199340087958245
p: 10 , err_train: 0.15797733433153385 , err_test: 0.13662301478558983
```



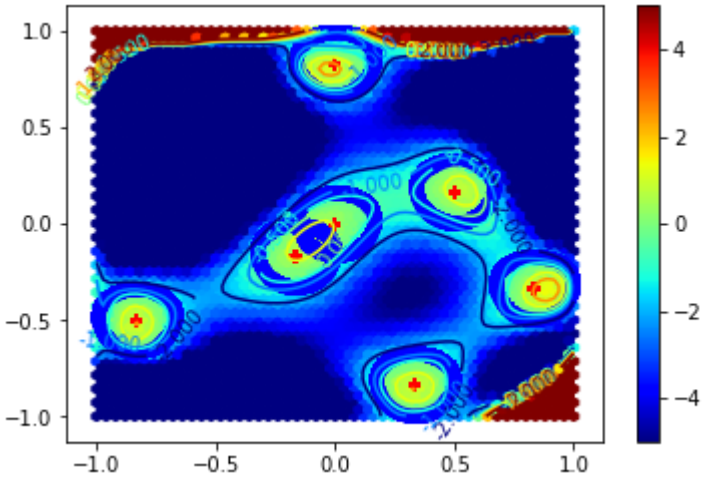
```
p: 11 , err_train: 0.15256240975000976 , err_test: 0.13119849462707694
p: 12 , err_train: 0.15173648517132773 , err_test: 0.13051903373523482
```



```
p: 13 , err_train: 0.15066193745292503 , err_test: 0.12951624491360123
p: 14 , err_train: 0.14994973815018323 , err_test: 0.1288266215998676
```




```
p: 15 , err_train: 0.14929491859425115 , err_test: 0.1281737583889233
p: 16 , err_train: 0.14813412684437402 , err_test: 0.12694373236875683
```

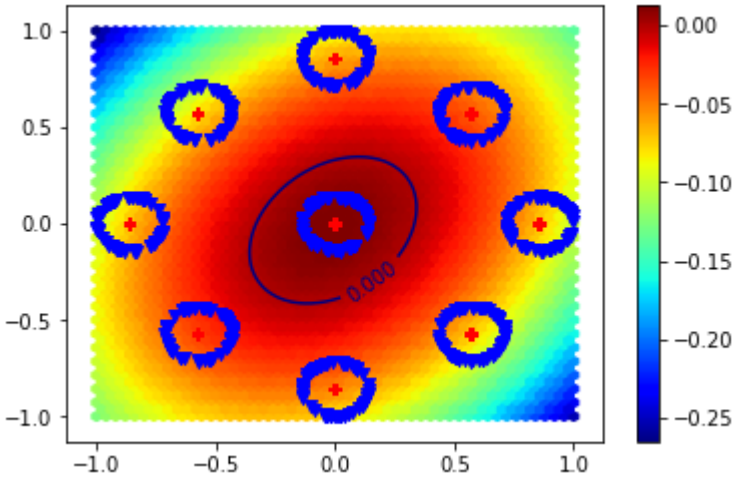


5.C

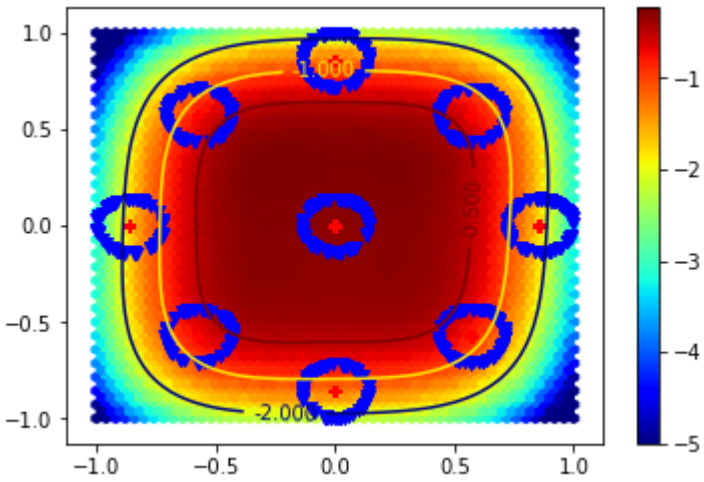
In [102]:
```
def kernelRidge_c(A, b, lambda_, d):
    K = np.power(Xc_train_reg @ Xc_train_reg.T + 1, d)
    return A.T @ (np.linalg.inv(K+lambda_*np.eye(len(K[0])))) @ (b)
```

In [107]:
```
for p in range(1, 17):

    start_of_features = comb(2 + p, 2, exact=True)

    Xc_train = Xc_train_mult[:, -start_of_features:]
    Xc_valid = Xc_valid_mult[:, -start_of_features:]

    w = kernelRidge_c(Xc_train, yc_train, l, p)

    err_train = np.sum(np.square((yc_train - Xc_train.dot(w)))) / len(y
c_train)
    err_test = np.sum(np.square((yc_valid - Xc_valid.dot(w)))) / len(yc
_valid)

    print('p:', p, ', err_train:', err_train, ', err_test:', err_test)

    if p%2 == 0:

        heatmap(lambda x, y: f(x, y, p).dot(w), Xc, yc)
```

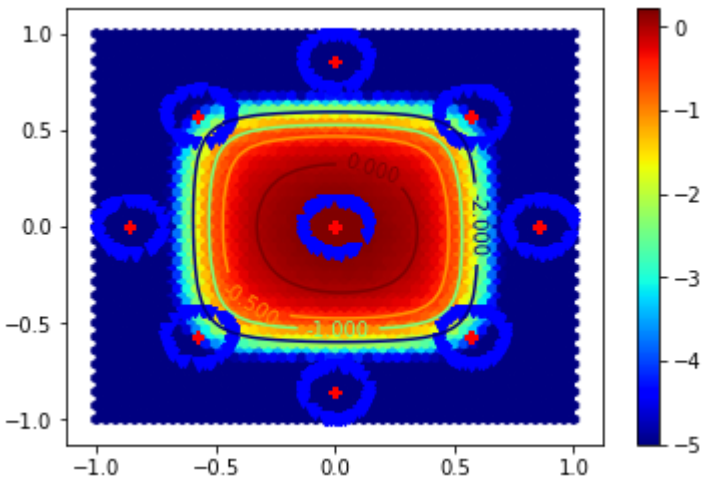
```
p: 1 , err_train: 0.9970876133159098 , err_test: 0.9975785898043206
p: 2 , err_train: 0.9957465938672967 , err_test: 0.998022133298275
```



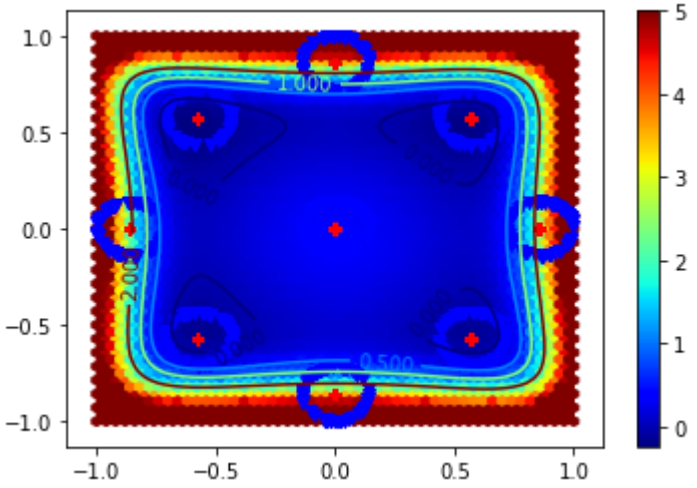

```
p: 3 , err_train: 1.0070456675383619 , err_test: 1.0128167986560404
p: 4 , err_train: 2.373414963646662 , err_test: 2.2262222194285117
```



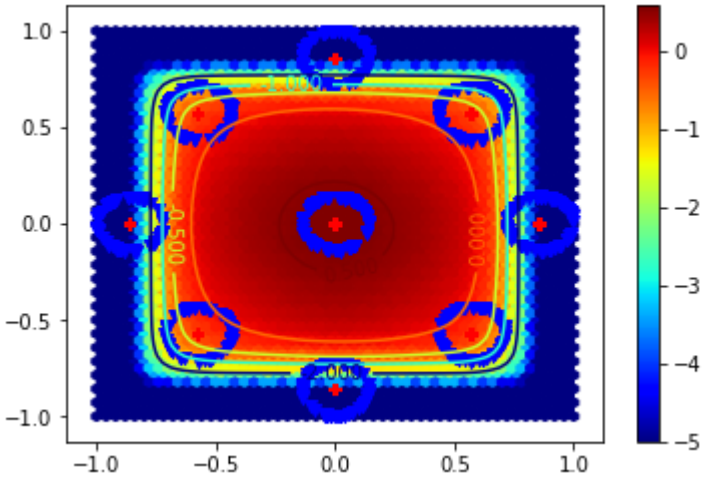

```
p: 5 , err_train: 1.2764811130520257 , err_test: 1.2002557067146848
p: 6 , err_train: 231.57516025765509 , err_test: 231.06178640304765
```



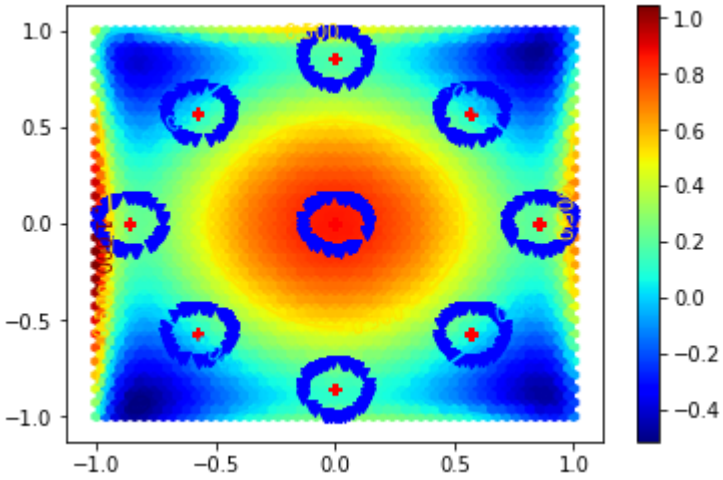

```
p: 7 , err_train: 5.396790132080263 , err_test: 5.811965294781059
p: 8 , err_train: 11.540489627112922 , err_test: 13.785094896529127
```



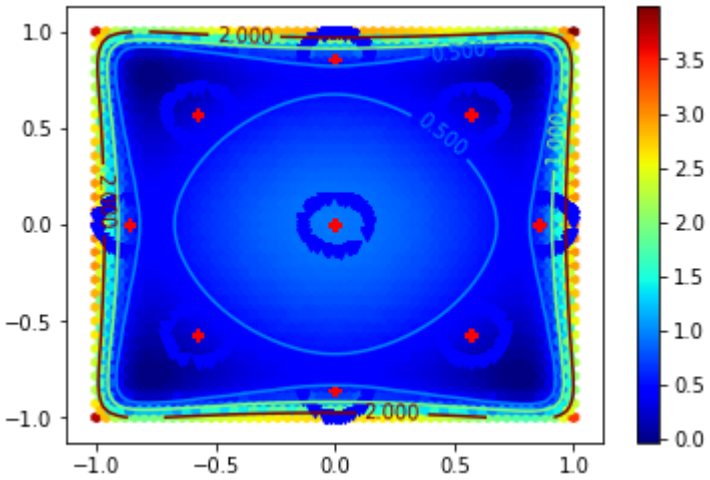
```
p: 9 , err_train: 2.380101773804355 , err_test: 2.702271112360945
p: 10 , err_train: 48.68635879920936 , err_test: 51.25328440581242
```



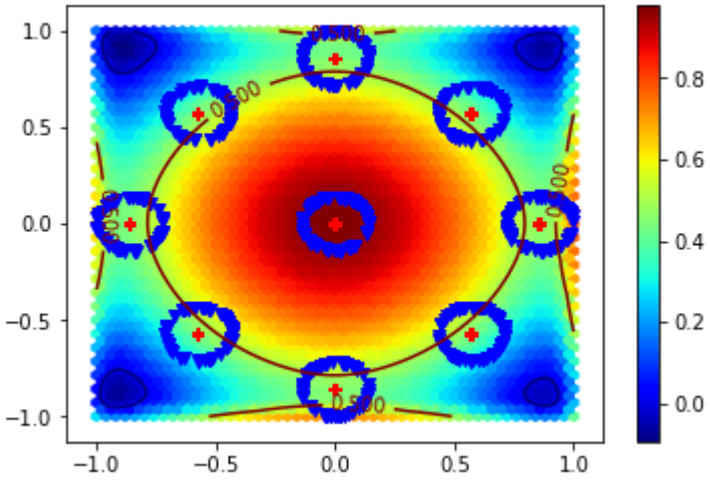
```
p: 11 , err_train: 1.3759202672101454 , err_test: 1.3510390683471785
p: 12 , err_train: 1.205432222642685 , err_test: 1.1822988117166844
```



```
p: 13 , err_train: 1.2425505543860438 , err_test: 1.2430008483714767
p: 14 , err_train: 1.9582860975830128 , err_test: 2.012558241739467
```

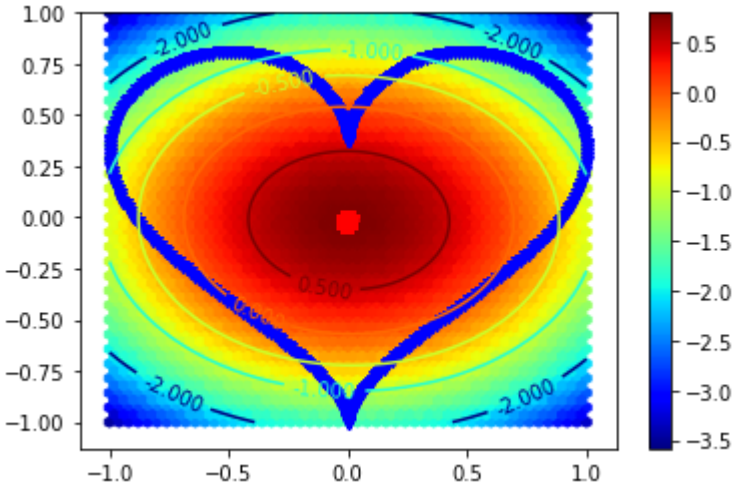



```
p: 15 , err_train: 1.3887290658312998 , err_test: 1.3741528531911804
p: 16 , err_train: 1.3851995004667337 , err_test: 1.3726369675387606
```

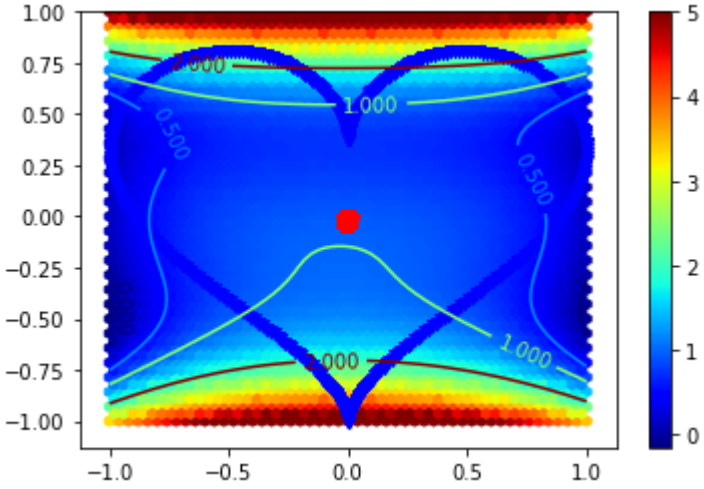


In [111]:
```
def kernelRidge_h(A, b, lambda_, d):
    k = np.power(Xh_train_reg @ Xh_train_reg.T + 1, d)
    return A.T @ (np.linalg.inv(k+lambda_*np.eye(len(k[0])))) @ (b)

for p in range(1, 17):

    start_of_features = comb(2 + p, 2, exact=True)

    Xh_train = Xh_train_mult[:, -start_of_features:]
    Xh_valid = Xh_valid_mult[:, -start_of_features:]

    w = kernelRidge_h(Xh_train, yh_train, l, p)

    err_train = np.sum(np.square((yh_train - Xh_train.dot(w)))) / len(y
h_train)
    err_test = np.sum(np.square((yh_valid - Xh_valid.dot(w)))) / len(yh
_valid)

    print('p:', p, ', err_train:', err_train, ', err_test:', err_test)

    if p%2 == 0:

        heatmap(lambda x, y: f(x, y, p).dot(w), Xh, yh)
```

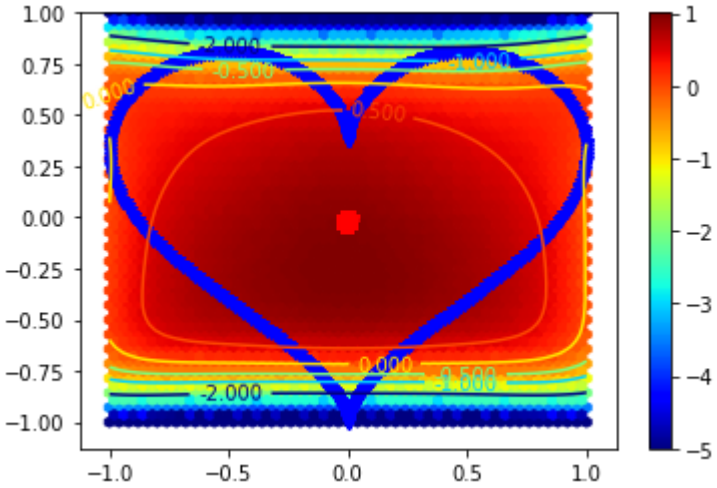
```
p: 1 , err_train: 0.9626430338740402 , err_test: 0.9599523272666317
p: 2 , err_train: 0.2376736220887096 , err_test: 0.1865502999186147
```



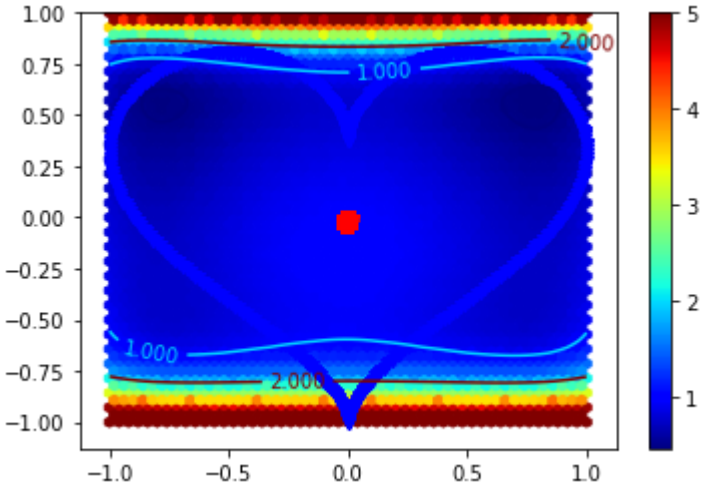

```
p: 3 , err_train: 0.7250102102869832 , err_test: 0.6634741710343797
p: 4 , err_train: 3.9097183852141058 , err_test: 4.084778213112436
```



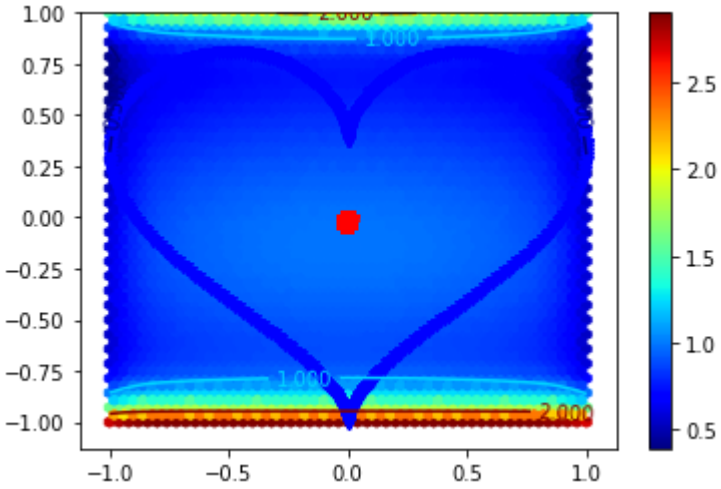

```
p: 5 , err_train: 3.0908456577432255 , err_test: 3.438031044033822
p: 6 , err_train: 1.37233670007729 , err_test: 1.553082580961232
```



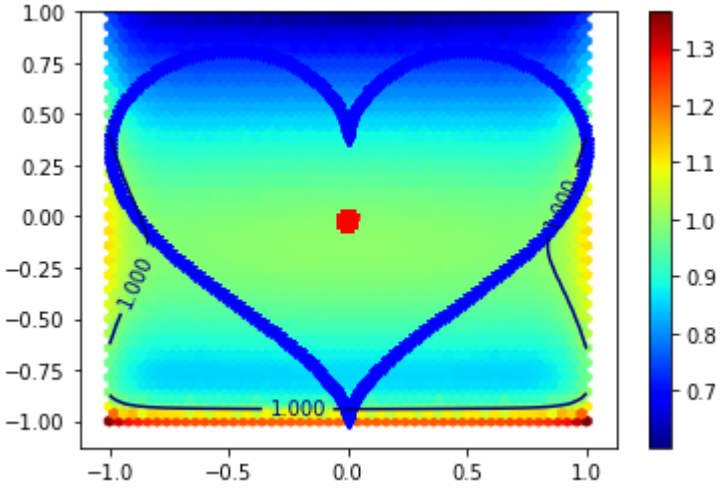

```
p: 7 , err_train: 0.8772791292992967 , err_test: 0.7840117239528607
p: 8 , err_train: 4.044458061975079 , err_test: 4.625989198343586
```



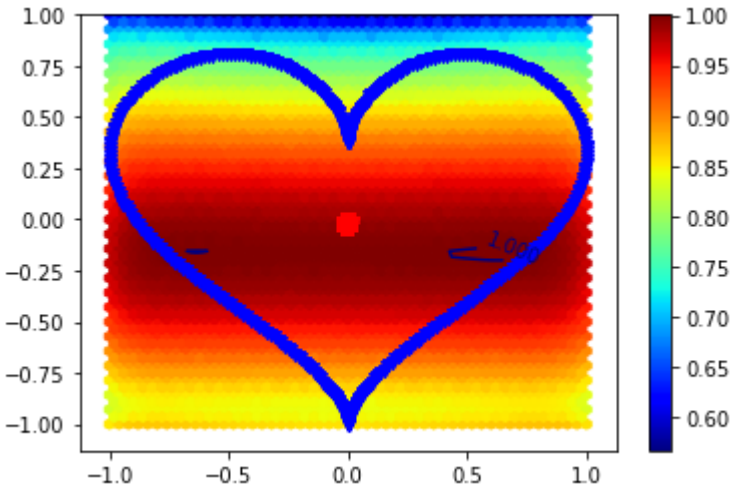

```
p: 9 , err_train: 1.9235953957379857 , err_test: 2.01866017079569
p: 10 , err_train: 1.7107716987949613 , err_test: 1.7436472966039305
```



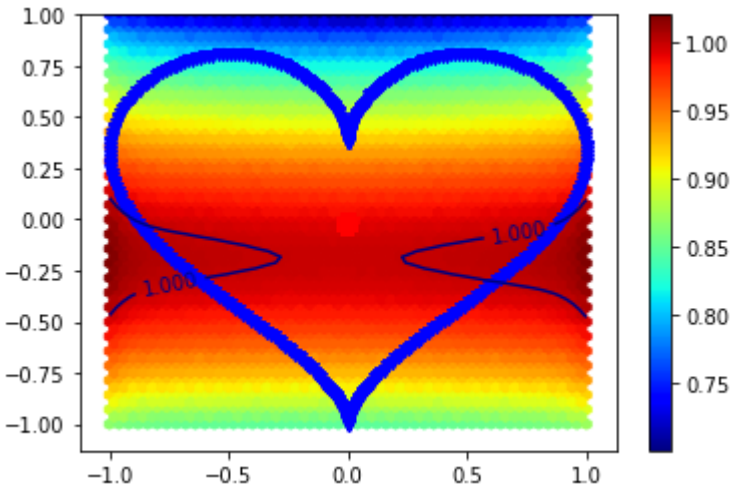

```
p: 11 , err_train: 1.5639951606060305 , err_test: 1.5658121688494582
p: 12 , err_train: 1.5121127877597855 , err_test: 1.4915163352137515
```




```
p: 13 , err_train: 1.4907585388432802 , err_test: 1.459938329415604
p: 14 , err_train: 1.479766132815939 , err_test: 1.4394183687898097
```



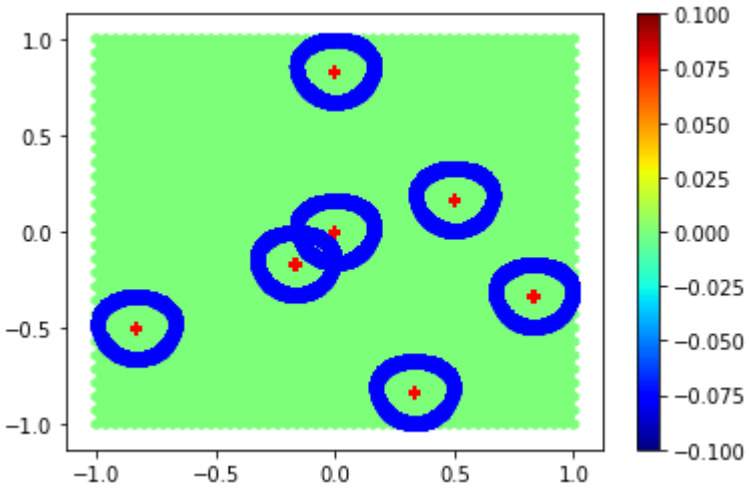

```
p: 15 , err_train: 1.50373411734295 , err_test: 1.4628055208990944
p: 16 , err_train: 1.5274283375858524 , err_test: 1.4857520907710446
```



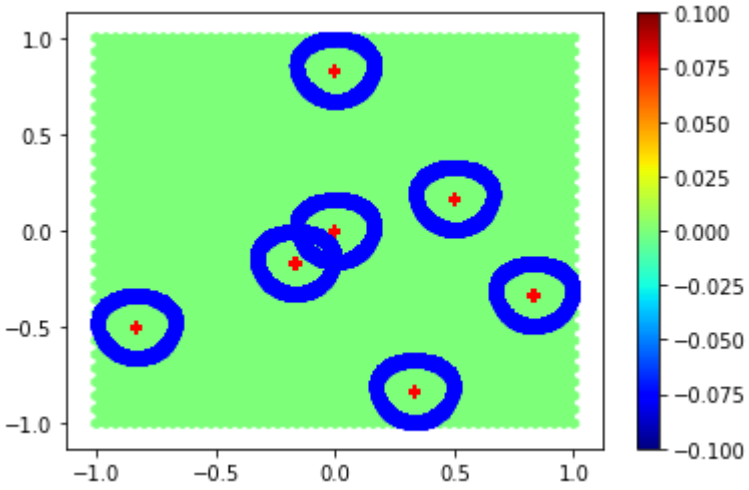
In [112]:
```
def kernelRidge_a(A, b, lambda_, d):
    k = np.power(Xa_train_reg @ Xa_train_reg.T + 1, d)
    return A.T @ (np.linalg.inv(k+lambda_*np.eye(len(k[0])))) @ (b)

for p in range(1, 17):

    start_of_features = comb(2 + p, 2, exact=True)

    Xa_train = Xa_train_mult[:, -start_of_features:]
    Xa_valid = Xa_valid_mult[:, -start_of_features:]

    w = kernelRidge_a(Xa_train, ya_train, l, p)

    err_train = np.sum(np.square((ya_train - Xa_train.dot(w)))) / len(y
a_train)
    err_test = np.sum(np.square((ya_valid - Xa_valid.dot(w)))) / len(ya
_valid)

    print('p:', p, ', err_train:', err_train, ', err_test:', err_test)

    if p%2 == 0:

        heatmap(lambda x, y: f(x, y, p).dot(w), Xa, ya)
```

```
p: 1 , err_train: 1.0 , err_test: 1.0
p: 2 , err_train: 1.0 , err_test: 1.0
```

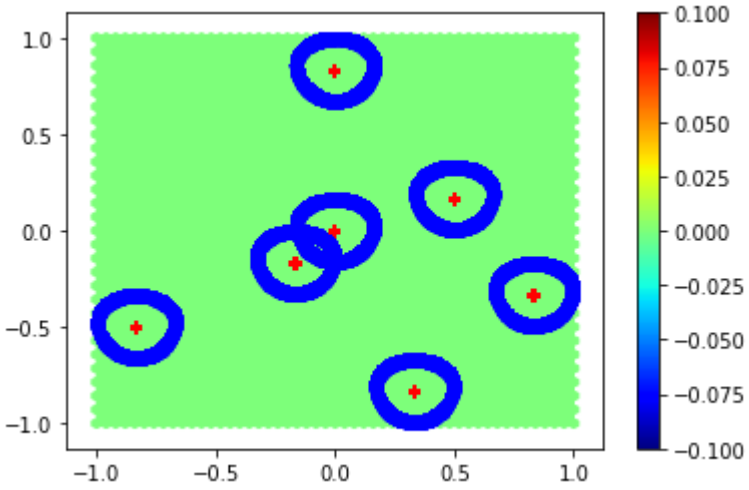
```
/usr/lib/python3.6/site-packages/matplotlib/contour.py:1180: UserWarni
ng: No contour levels were found within the data range.
  warnings.warn("No contour levels were found"
```



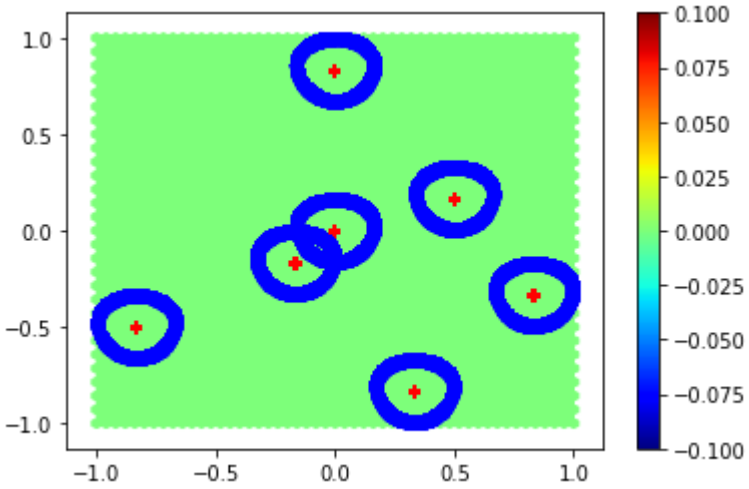

```
p: 3 , err_train: 1.0 , err_test: 1.0
p: 4 , err_train: 1.0 , err_test: 1.0
```




```
p: 5 , err_train: 1.0 , err_test: 1.0
p: 6 , err_train: 1.0 , err_test: 1.0
```




```
p: 7 , err_train: 1.0 , err_test: 1.0
p: 8 , err_train: 1.0 , err_test: 1.0
```




```
p: 9 , err_train: 1.0 , err_test: 1.0

---------------------------------------------------------------------------
-----
KeyboardInterrupt                         Traceback (most recent call
 last)
<ipython-input-112-55da21c36bca> in <module>()
     10     Xa_valid = Xa_valid_mult[:, -start_of_features:]
     11
---> 12     w = kernelRidge_a(Xa_train, ya_train, l, p)
     13
     14     err_train = np.sum(np.square((ya_train - Xa_train.dot(w)))
) / len(ya_train)

<ipython-input-112-55da21c36bca> in kernelRidge_a(A, b, lambda_, d)
      1 def kernelRidge_a(A, b, lambda_, d):
----> 2     k = np.power(Xa_train_reg @ Xa_train_reg.T + 1, d)
      3     return A.T @ (np.linalg.inv(k+lambda_*np.eye(len(k[0]))))
@ (b)
      4
      5 for p in range(1, 17):

KeyboardInterrupt:
```

In [115]:
```
n_train_c_new = int(Xc.shape[0] * 0.85)
Xc_train_reg_n = Xc[:n_train_c_new:, :]
Xc_valid_reg_n = Xc[n_train_c_new:, :]

def kernelRidge_c1(A, b, lambda_, d):
    K = np.power(Xc_train_reg_n @ Xc_train_reg_n.T + 1, d)
    return A.T @ (np.linalg.inv(K+lambda_*np.eye(len(K[0])))) @ (b)
```

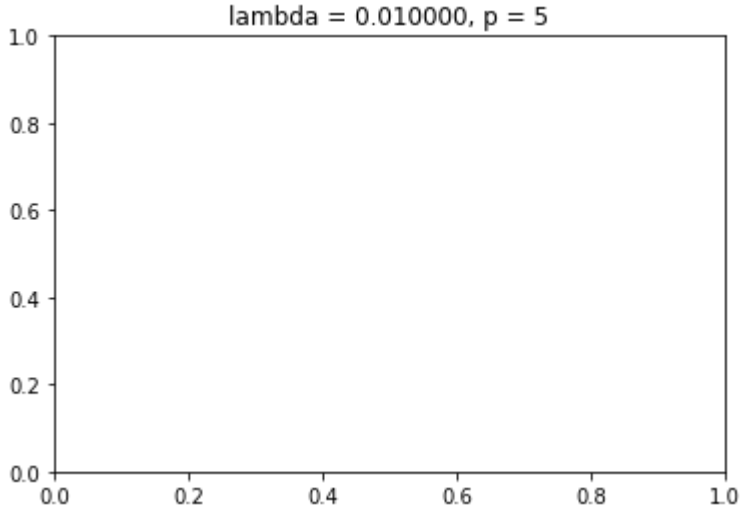
In [116]:
```
Xc_train_mult_1 = all_multivariates_Xc[:n_train_c_new:, :]
Xc_valid_mult_1 = all_multivariates_Xc[n_train_c_new:, :]

yc_train_1 = yc[:n_train_c_new]
yc_valid_1 = yc[n_train_c_new:]
for p in range(1, 24):

    start_of_features = comb(2 + p, 2, exact=True)

    Xc_train = Xc_train_mult_1[:, -start_of_features:]
    Xc_valid = Xc_valid_mult_1[:, -start_of_features:]

    w = kernelRidge_c1(Xc_train, yc_train_1, l, p)

    err_train = np.sum(np.square((yc_train_1 - Xc_train.dot(w)))) / len
(yc_train)
    err_test = np.sum(np.square((yc_valid_1 - Xc_valid.dot(w)))) / len(
yc_valid)

    print('p:', p, ', err_train:', err_train, ', err_test:', err_test)
```

```
p: 1 , err_train: 1.0588512249641662 , err_test: 0.7526770266620376
p: 2 , err_train: 1.0580111232053249 , err_test: 0.7518484636022634
p: 3 , err_train: 1.0646380445004886 , err_test: 0.7547367569791548
p: 4 , err_train: 2.4515270181784263 , err_test: 1.7105972221422674
p: 5 , err_train: 1.3551296120596572 , err_test: 0.8854668480762531
p: 6 , err_train: 243.82658664770233 , err_test: 167.11795707266853
p: 7 , err_train: 5.645093182546054 , err_test: 4.042940231037674
p: 8 , err_train: 9.283822832746875 , err_test: 7.5962194004169525
p: 9 , err_train: 2.2207355834901246 , err_test: 1.7223230957807836
p: 10 , err_train: 56.64877480108909 , err_test: 40.415303590043116
p: 11 , err_train: 1.4267561392814845 , err_test: 0.96437255089493
p: 12 , err_train: 1.3268382440228845 , err_test: 0.9176510400713306
p: 13 , err_train: 1.3361150636800645 , err_test: 0.9224610340304717
p: 14 , err_train: 2.135787987312577 , err_test: 1.4984393278632955
p: 15 , err_train: 1.48181988521133 , err_test: 0.9979163028658273
p: 16 , err_train: 1.4838681661855595 , err_test: 0.9910661995258027
p: 17 , err_train: 1.478433519719714 , err_test: 0.9820736219596226
p: 18 , err_train: 1.526769911828564 , err_test: 1.011341869698491
p: 19 , err_train: 1.5791228326241984 , err_test: 1.0436867061796808
p: 20 , err_train: 1.6261550202503383 , err_test: 1.0727024348827376
p: 21 , err_train: 1.6693443558337224 , err_test: 1.099441931250384
p: 22 , err_train: 1.7097560550214135 , err_test: 1.1245735166814692
p: 23 , err_train: 1.74707747292698 , err_test: 1.1478504878219045
```

In [120]:
```
Xa_train_mult = all_multivariates_Xa[:n_train_a:, :]
Xa_valid_mult = all_multivariates_Xa[n_train_a:, :]
ya_train = ya[:n_train_a]
ya_valid = ya[n_train_a:]

n_trains = np.arange(10, n_train_a, 1000)
for p in (5, 6):

    start_of_features = comb(2 + p, 2, exact=True)
    Xa_valid = Xa_valid_mult[:, -start_of_features:]

    for l in (0.0001,0.001,0.01):
        err_valid = []

        for n_train in n_trains:

            # 100 trials
            err_test = 0
            for seed in range(100):

                np.random.seed(seed)
                samples = np.random.randint(0, n_train_a, n_train)
                Xa_train = Xa_train_mult[samples, -start_of_features:]

                #err_test = 0
                w = lstsqL(Xa_train, ya_train[samples], l)

                err_test += np.sum(np.square((ya_valid - Xa_valid.dot(w
)))) / len(ya_valid)


                err_valid.append(err_test/100)

    plt.title("lambda = %f, p = %d" % (l, p))
    plt.plot(n_trains, err_valid)
    plt.show()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call
 last)
<ipython-input-120-1b3dd7bae6b4> in <module>()
     31
     32     plt.title("lambda = %f, p = %d" % (l, p))
---> 33     plt.plot(n_trains, err_valid)
     34     plt.show()

/usr/lib/python3.6/site-packages/matplotlib/pyplot.py in plot(*args, *
*kwargs)
   3259                      mplDeprecation)
   3260     try:
-> 3261         ret = ax.plot(*args, **kwargs)
   3262     finally:
   3263         ax._hold = washold

/usr/lib/python3.6/site-packages/matplotlib/__init__.py in inner(ax, *
args, **kwargs)
   1715                     warnings.warn(msg % (label_namer, func.__n
ame__),
   1716                                   RuntimeWarning, stacklevel=
2)
-> 1717             return func(ax, *args, **kwargs)
   1718         pre_doc = inner.__doc__
   1719         if pre_doc is None:

/usr/lib/python3.6/site-packages/matplotlib/axes/_axes.py in plot(sel
f, *args, **kwargs)
   1370         kwargs = cbook.normalize_kwargs(kwargs, _alias_map)
   1371
-> 1372         for line in self._get_lines(*args, **kwargs):
   1373             self.add_line(line)
   1374             lines.append(line)

/usr/lib/python3.6/site-packages/matplotlib/axes/_base.py in _grab_nex
t_args(self, *args, **kwargs)
    402                 this += args[0],
    403                 args = args[1:]
--> 404             for seg in self._plot_args(this, kwargs):
    405                 yield seg
    406

/usr/lib/python3.6/site-packages/matplotlib/axes/_base.py in _plot_arg
s(self, tup, kwargs)
    382             x, y = index_of(tup[-1])
    383
--> 384         x, y = self._xy_from_xy(x, y)
    385
    386         if self.command == 'plot':

/usr/lib/python3.6/site-packages/matplotlib/axes/_base.py in _xy_from_
xy(self, x, y)
    241         if x.shape[0] != y.shape[0]:
    242             raise ValueError("x and y must have same first dim
ension, but "
```

```
--> 243                                 "have shapes {} and {}".format(x.
shape, y.shape))
    244         if x.ndim > 2 or y.ndim > 2:
    245             raise ValueError("x and y can be no greater than 2
-D, but have "

ValueError: x and y must have same first dimension, but have shapes (1
7,) and (1700,)
```



5. E

5. F

Polynomial ridge regression with the implementation of the kernel trick is more efficient. If the amount of training data we have is significantly less than the hnumber of features we have than we would opt to implement kernel. Otherwise, the non-kernalized version of polynomial ridge regression is more efficient.

5. G

HW04 - CS 189

1.
Who else did you work with on this homework? In case of course events, just describe the group. How did you work on this homework? Any comments about the homework?

I worked on this homework with Ehimare Okoyomon, Prashanth Ganeth, and Daniel Mockaitis. We worked by getting together throughout the week and communicating on facebook.

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}

Nicholas Lorio, 26089160

Question 4

a. We can rewrite the first expression f(w, r) as the lagrangian, L(w, r, alpha). The optimal solution for min f(w, r) is always greater than or equal to min L(w, r, alpha) due to weak duality theorem. Thus lagrangian gives a lower bound on min f(w, r). If the constraint that the min problem is subject to is satisfied than our problem is just like normal, if its not then we want the penalty to become infinity such that the minimization of its is infeasible. That being said we want to maximize our lower bound, L(w, r, alpha) which would give us f(w, r). Taking min on both sides gives us the second expression which we are trying to show as equivalent.

Question 6.

How are Kernel Hilbert spaces useful in machine learning?

Reproducing kernel Hilbert space, RKHS, is defined as a space where if two functions are close norm than they are also pointwise close i.e the functions given the same value x are likewise close in the space. We take advantage of this in ML.

"An RKHS is associated with a kernel that reproduces every function in the space in the sense that for any x in the set on which the functions are defined, "evaluation at x" can be performed by taking an inner product with a function determined by the kernel. Such a reproducing kernel exists if and only if every evaluation functional is continuous."
https://en.wikipedia.org/wiki/Reproducing_kernel_Hilbert_space

The kernel we have come to know through our understanding and implementation of the kernel trick for ML ridge regression is defined by K(x, y) = inner product(phi(x), phi(y)). The representation stated implies that these elements of RKHS are inner products of the elements of

the features space and can be seen as hyperplanes, subspaces whose dimensions are one less than those of their ambient space. The ambient space is the original features matrix. The properties of Kernel Hilbert Spaces allow us to perform the Kernel trick we used throughout this HW which vastly increases the efficiency of our modeling.

Sources

https://cs.stanford.edu/people/davidknowles/lagrangian_duality.pdf
IEOR 160 Notes

HW04

2.a. $f(x_1 \dots x_d) = \dfrac{\exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right)}{\sqrt{(2\pi)^d |\Sigma|}}$

$$lik(\mu, \Sigma) = \prod_{i=1}^{n} (2\pi)^{-d/2} |\Sigma|^{-1/2} \exp\left\{-\frac{1}{2}(x_i-\mu)^T \Sigma^{-1}(x_i-\mu)\right\}$$

$$-\log(lik(\mu,\Sigma)) = \sum_{i=1}^{n} +\frac{d}{2}\log(2\pi) + \frac{1}{2}\log(|\Sigma|) + \frac{1}{2}(x_i-\mu)^T\Sigma^{-1}(x_i-\mu)$$

$= L(\mu, \Sigma)$

2.b. * 2.4 (84) ML cookbook

$$\arg\min_{\mu,\Sigma} L(\mu,\Sigma) \begin{cases} \partial/\partial\mu = \sum_{i=1}^{n} 0 + 0 + \frac{1}{2}\left(-2(-1)\Sigma^{-1}(x_i-\mu)\right) \\ \quad = \sum_{i=1}^{n} 2\cdot\Sigma^{-1}(x_i-\mu) \\ \partial/\partial\Sigma = \Sigma \frac{1}{2}(\Sigma^{-1})^T + \frac{1}{2}\left(-\Sigma^{-1}\right)^T (x_i-\mu)(x_i-\mu)^T\Sigma^{-1} \end{cases}$$

* (57) * 61, [redacted]

① $\sum_{i=1}^{n} \Sigma^{-1}(x_i-\mu) = 0$ $\quad -(\Sigma^T)^{-1}$ $\quad x_i^T - \mu^T$

② $\frac{1}{2}\sum_{i=1}^{n}(\Sigma^{-1})^T - \Sigma^{-1T}(x_i-\mu)(x_i-\mu)^T\Sigma^{-1T} = 0$

① $\sum_{i=1}^{n}\Sigma^{-1}x_i - \Sigma^{-1}\mu = 0$, $\Sigma^{-1}\sum x_i = \Sigma^{-1}\sum\mu$

$$\boxed{\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} x_i} \qquad \hat{\mu} = \frac{1}{n}\sum x_i$$

② $\frac{n}{2}(\Sigma^{-1})^T - \frac{1}{2}\sum(\Sigma^{-1})^T(x_i x_i^T - x_i\mu^T - \mu x_i^T + \mu\mu^T)(\Sigma^{-1})^T = 0$

(2)

$$n(\Sigma^{-1})^T = \sum_{i=1}^{n} (\Sigma^{-1})^T (x_i x_i^T - x_i \mu^T - \mu x_i^T + \mu\mu^T)(\Sigma^{-1})^T$$

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

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

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

$$n\Sigma^T = \Sigma^{-1} \cdot \sum_{i=1}^{n} (x_i - \mu)(x_i - \mu)^T$$

$$\boxed{\hat{\Sigma} = \frac{1}{n} \sum_{i=1}^{n} (x_i - \mu)(x_i - \mu)^T}$$

$z = y - x^T w$

3. a. $x \in \mathbb{R}^d \quad y \in \mathbb{R}$, $Y = x^T w + z$

$z \sim N(0,1)$

$w \in \mathbb{R}^d \sim N(0, \Sigma)$ Σ symmetric, PSD

w is independent of z

$$\mathbb{P}(Y \mid x_1 \dots x_d, w_1 \dots w_d) = P(Y,$$

$$\mathbb{P}(Y = x^T w + z \mid \{x_i, w_i\}_{i=1\dots d}) = \mathbb{P}(\underline{z = y - x^T w} \mid \{x_i, w_i\}_{i=1\dots d})$$

$$= \mathbb{P}(z = y - \underbrace{x^T w}_{\text{scalar}} \mid \underbrace{\{x_i, w_i\}_{i=1\dots d}}_{\text{prior } \Sigma})$$

$$= \frac{1}{\sqrt{2\pi \cdot 1}} \exp\left\{-\frac{1}{2}(y - x^T w)^2\right\}$$

3.b. n strong Data pts. (x_i, Y_i) $i = 1, \dots, n$ i.i.d

Y_i given drawn according $\{x_i\}_{i=1}^n$

$\mathbb{P}(w \mid \text{strong Data})$

$$= \frac{\mathbb{P}(\text{Data} \mid w)\, \mathbb{P}(w)}{\mathbb{P}(\text{Data})}$$

MAP $\propto \mathbb{P}(\text{Data} \mid w)\, \mathbb{P}(w) = \mathbb{P}(w)\, \mathbb{P}(\text{Data} \mid w)$

$$\arg\max_w \prod_{i=1}^n \frac{1}{(2\pi)^{d/2}|\Sigma|} \exp\left\{-\frac{1}{2} w_i^T \Sigma^{-1} w_i\right\} \cdot \frac{1}{\sqrt{2\pi}} \exp\left\{-\frac{1}{2}(y_i - x_i w)^2\right\}$$

$$f(v) = C \cdot \exp\left\{ -\tfrac{1}{2} v^T A v + b^T v \right\} \quad \hat{v} = A^{-1} b$$

3 $$\arg\max \prod_{i=1}^{n} \underbrace{(2\pi)^{-d/2}(2\pi)^{-1/2}}_{\text{constant}} |\Sigma|^{-1/2} \cdot \exp\left\{ -\tfrac{1}{2}(y_i - x_i^T w)^2 + w_i^T \Sigma^{-1} w_i \right\}$$

$\int f(v)$ integrates to 1 for C then mean is $A^{-1}b$ for v.

$$\exp\left\{ -\tfrac{1}{2}\left[y_i^2 - y_i x_i^T w - x_i^T w y_i + (x_i^T w)^2 \right] - \tfrac{1}{2} w_i^T \Sigma^{-1} w_i \right]\}$$

$$\underbrace{\exp(-1/2\, y_i^2)}_{\text{constant}} \exp\left\{ -\tfrac{1}{2}\left(-2 y_i x_i^T w + w^T x_i^T x_i w \right) - \tfrac{1}{2} w^T \Sigma^{-1} w \right\}$$

$$C \cdot \prod_{i=1}^{n} \exp\left\{ \underbrace{(x_i y_i)^T}_{b} w - \tfrac{1}{2} w^T \underbrace{(x_i^T x_i + \Sigma^{-1})}_{A} w \right\} = G$$

Note $\sum x_i^T x_i = X^T X$ $\quad \therefore A = X^T X + \Sigma^{-1}$

$\sum x_i y_i = X^T y$ $\quad b = X^T y$

$$\mu, \mathbb{E}(G) = (X^T X + \Sigma^{-1})^{-1} X^T y$$

$$\hat{w} = (X^T X + \Sigma^{-1})^{-1} X^T y$$

* according to the hint given by the problem

- tikhonov generally refers to ridge regression from probability perspective:

Ridge reg. introduces prior on w assuming w_i are iid

tikhonov introduces prior on w where the ws are jointly gaussian, (generated from a gaussian perspective)

RR: $\hat{w} = (x^T x + \frac{1}{\sigma^2} I)^{-1} x^T y$ Tik: $\hat{w} = (x^T x + \Sigma^{-1})^{-1} x^T y$

3.b. (cont)

$$\propto \exp\left\{(X^T Y)^T w - \frac{1}{2} w^T (X^T X + \Sigma_w^{-1}) w\right\}$$

complete the square $ax^2 + bx + c \Rightarrow a(x+d)^2 + e$

$d = b/2a \qquad e = c - b^2/4a$

let us define $M = (X^T X + \Sigma_w^{-1})$

$$p(w \mid x_1, y_1, \ldots, x_n, y_n) \propto \exp\left\{-\frac{1}{2}\left(2(X^T y)^T w - w^T M w\right)\right\}$$

$$\propto \exp\left\{-\frac{1}{2}\left[(w - M^{-1}(X^T y))^T M (w - M^{-1}(X^T y))\right] + \text{constants}\right\}$$

- this matches the form of the JG posterior for w, JG,

$$f_Z(z) = C \cdot \exp\left\{-\frac{1}{2}(z-\mu)^T \Sigma^{-1} (z-\mu)\right\}$$

thus $\mu = M^{-1}(X^T y)$, as written earlier from the mat

$$\Sigma^{-1} = M \quad , \quad M^{-1} = \Sigma = (X^T X + \Sigma_w^{-1})^{-1}$$

$$y = Xw + z$$

3.c.

$z_i \sim \mathcal{N}(\mu_z, \Sigma_z)$ independent of w

• Posterior dist of w by appropriately choosing coordinates. Σ_z invertible Hint:

Let v be $\mathcal{N}(0, I)$ standard Gaussian $v \sim \mathcal{N}(0, I)$

Gaussian property

$$z = \Sigma_z^{1/2} v + \mu_z \quad \text{"reparameterization"}$$

$$y = Xw + \Sigma_z^{1/2} v + \underbrace{\mu_z}_{\text{constant}}$$

$\Sigma^{-1/2}$ left mult

$$\Sigma^{-1/2} y = \Sigma^{-1/2} Xw + v + \Sigma^{-1/2} \mu_z$$

$$\Sigma^{-1/2}(y - \mu_z) = \Sigma^{-1/2} Xw + v$$

$$\tilde{y} = \Sigma^{-1/2}(y - \mu_z)$$
$$\tilde{X} = \Sigma^{-1/2} X$$

$$\tilde{y} = \tilde{X} w + v \quad \therefore \text{ we can use from our past results that}$$

$$\hat{w} = (\tilde{X}^T \tilde{X} + \Sigma_w^{-1})^{-1} \tilde{X}^T \tilde{y}$$

$$= \left[(\Sigma_z^{-1/2} X)^T (\Sigma_z^{-1/2} X) + \Sigma_w^{-1} \right]^{-1} (\Sigma_z^{-1/2} X)^T (\Sigma_z^{-1/2} (y - \mu_z))$$

$$\left[X^T \Sigma^{-1/2} \Sigma^{-1/2} X + \Sigma_w^{-1} \right]^{-1} (X^T \Sigma^{-1/2} \Sigma^{-1/2} (y - \mu_z))$$

Σ_z^{-1} Σ_z^{-1}

4.a.

$$\min_{w,r} \frac{1}{2}\left[\|r\|_2^2 + \lambda \|w\|_2^2 \right] \quad s.t \quad r = Xw - y$$
$$r - Xw + y = 0$$

$\rightsquigarrow$ turn back to the lagrangian

$$\min_{w,r} \left[L(w,r,\alpha) \right] = \min_{w,r} \underbrace{\frac{1}{2}\left[\|r\|_2^2 + \lambda \|w\|_2^2 \right]}_{F(w,r)} + \underbrace{\alpha (r - Xw + y)}_{h(w,r)}$$

Note that • $L(w,r,0) = F(w,r)$

• if constraint $h(w,r) = 0$ is satisfied then the best we can do is set $\alpha = 0$. (i.e the smallest penalty)

• gives penalty to the constraint being violated

• if constraint $h(w,r) \neq 0$ Not satisfied then we can make $L(w,r,\alpha)$ infinite & thus not a n optimal solution to our Min optimization problem. we do this by taking max w.r.t α.

"the min of ∞ is infeasible" thus

$$\max_{w,r,\alpha} L(w,r,\alpha) = F(w,r)$$

& :.

$$\min_{w,r} \max_{\alpha} L(w,r,\alpha) = \min_{w,r} F(w,r)$$

$F(w,r) \geq L(w,r,\alpha)$ optimal
gives LB on $F(w,r)$
so to get a tight LB we
Max $L(w,r,\alpha)$

Max = - Min , Duality ?

4.b.

$$\min_{w,r} \max_{\alpha} L(w,r,\alpha) = \max_{\alpha} \min_{w,r} L(w,r,\alpha)$$

$$\max_{\alpha} \min_{w,r} L(w,r,\alpha) = \arg\min_{\alpha} \left[\frac{1}{2}\alpha^T(k+\lambda I)\alpha - \lambda\alpha^T y \right]$$

w/ $k = xx^T \in \mathbb{R}^{n\times n}$

Lagrangian dual $\sim d(w,r,\alpha) =$

$$\max_{\alpha} \min_{w,r} \left[\frac{1}{2}\|r\|_2^2 + \frac{\lambda}{2}\|w\|_2^2 + \alpha^T(r - xw + y) \right]$$

$$\max_{\alpha} \min_{w,r} \left[\frac{1}{2} r^T r + \frac{\lambda}{2} w^T w + \alpha^T r - \alpha x w + \alpha^T y \right]$$

$$\max_{\alpha} \left[\begin{array}{ll} \delta/\delta w = \lambda w - x^T\alpha = 0 & w^* = \frac{1}{\lambda}x^T\alpha \\ \delta/\delta r = r + \alpha = 0 & r^* = -\alpha \end{array} \right.$$

$$\arg\max_{\alpha} \left[\frac{1}{2}\|-\alpha\|_2^2 + \frac{\lambda}{2}\left\|\frac{1}{\lambda}x^T\alpha\right\|_2^2 + \alpha^T\left(+\alpha + x\frac{1}{\lambda}x^T\alpha + y\right) \right]$$

$$\arg\min_{\alpha} \left[\frac{1}{2}\alpha^T\alpha + \frac{\lambda}{2}\left(\frac{1}{\lambda}x^T\alpha\right)^T\left(\frac{1}{\lambda}x^T\alpha\right) + \alpha^T\alpha + \frac{1}{\lambda}\alpha^T\lambda x^T\alpha + \alpha^T y \right]$$

$$\frac{1}{2}\alpha^T\alpha + \frac{1}{2\lambda}\alpha^T x x^T\alpha + \alpha^T y \quad \text{Mult all by } \lambda$$

$$\frac{1}{2}\lambda\alpha^T\alpha + \frac{1}{2}\alpha^T x x^T\alpha + \lambda\alpha^T y$$

$$= \arg\min_{\alpha} \left[\frac{1}{2}\alpha^T(k+\lambda I)\alpha + \lambda\alpha^T y \right]$$

9.c. $\min_{\alpha} \delta/\delta x$ (from b)

$K = XX^T$ symmetric

$(K + \lambda I)$ symmetric

$= \frac{1}{2}\left[(K+\lambda I) + (K+\lambda I)^T \right] \alpha - \lambda y = 0$

$(K+\lambda I)\alpha = \lambda y \qquad \alpha^* = \lambda (K+\lambda I)^{-1} y$

$\therefore w^* = \frac{1}{\lambda} X^T \alpha^* = \frac{1}{\lambda} X^T \lambda (K+\lambda I)^{-1} y$

$w^* = X^T (K+\lambda I)^{-1} y$

4.D 9.D. Likewise $\Rightarrow w^* = (\Phi^T \Phi + \lambda I)^{-1} \Phi^T y$ (eqn 1)

&

kernel ridge regression $w^* = \Phi^T (K+\lambda I)^{-1} y$ (eqn 2)

w/ $K = \Phi \Phi^T$

$\therefore$ given a new point x.

$\hat{y}_1 \qquad\qquad \hat{y}_2$

$\Phi(x)^T w_1 \qquad\qquad \Phi(x^T) w_2$

$\Phi_1^T (\Phi^T \Phi + \lambda I)^{-1} \Phi^T y_1 = \Phi_2^T \Phi^T (\Phi \Phi^T + \lambda I)^{-1} y_2$

Using hint we can see how the equivalent is

$\Phi_1^T (\Phi^T \Phi + \lambda I)^{-1} \Phi^T y = \Phi_2^T (\Phi^T \Phi + \lambda I)^{-1} \Phi^T y$

4.e.

- computational complexity

	Tikhonov Directly	Generalized version
computing	$\Phi^T\Phi$ $O(d^2n)$	$O(n^3 + \log(p))$
inverting	$O(d^3)$	Inverting $O(n^3)$
total	$O(d^3 + d^2n)$	tot $O(n^3 + n^2(l + \log(p))$
	$\hat{w}_1$	$\hat{w}_2$

Actually using the derived $\hat{w}$ from either method should take the same computational timework

~~$y_1 = (\Phi^T\Phi)\hat{w}$~~ $\boxed{y = X\hat{w}}$

$y_1 = X\hat{w}_1$