1. (a)

I did this homework with Luning Zhao.

We work on our homework seperately, but we discuss when meeting problems.

Homework is fine.

(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.

2.(a) $P(x_1, \cdots x_n) = P(x_1) \cdots P(x_n)$

$$= \prod_i^n \frac{1}{\sqrt{2\pi|\Sigma|}} \left(e^{\frac{-(x_i-\mu)^T \Sigma^{-1} (x_i-\mu)}{2}} \right)$$

$$\log P(x_1, \cdots x_n \mid \mu, \Sigma) = \sum_i - \frac{(x_i-\mu)^T \Sigma^{-1} (x_i-\mu)}{2} - \frac{n}{2}\log|\Sigma| + c$$

(b) $\cdot \dfrac{\partial \log P}{\partial \mu} = -\frac{1}{2} \sum_i \dfrac{-\Delta^T \Sigma^{-1} (x_i-\mu) + (x_i-\mu)^T \Sigma^{-1} \cdot -\Delta}{\Delta}$

$$= -\frac{1}{2} \sum_i \left(-(x_i-\mu)^T \Sigma^{-1} - (x_i-\mu)^T \Sigma^{-1} \right)$$

$$= \sum_i (x_i-\mu)^T \Sigma^{-1} = 0$$

$$\Rightarrow \sum_i (x_i-\mu) = 0 \quad \Rightarrow \mu = \frac{1}{n} \sum x_i$$

$\cdot \dfrac{\partial \log P}{\partial \Sigma} = -\frac{1}{2} \sum_i \Sigma^{-1} (x_i-\mu)(x_i-\mu)^T \Sigma^{-1} - \frac{n}{2} \frac{1}{|\Sigma|} |\Sigma| \Sigma^{-1}$

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

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

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

(c) $\Sigma_{true} = \begin{bmatrix} 20 & 0 \\ 0 & 10 \end{bmatrix} \Rightarrow \Sigma_{calc} = \begin{bmatrix} 19.1 & 0.3 \\ 0.3 & 8.2 \end{bmatrix}$

$\Sigma_{true} = \begin{bmatrix} 20 & 14 \\ 14 & 10 \end{bmatrix} \Rightarrow \Sigma_{calc} = \begin{bmatrix} 22.0 & 15.4 \\ 15.4 & 11.0 \end{bmatrix}$

$\Sigma_{true} = \begin{bmatrix} 20 & -14 \\ -14 & 10 \end{bmatrix} \Rightarrow \Sigma_{calc} = \begin{bmatrix} 20.0 & -14.0 \\ -14.0 & 10.0 \end{bmatrix}$

See code in next page.

2 (c)

```python
import numpy as np
import matplotlib.pyplot as plt

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

def calc_mu_sigma(mu_true, sigma_true):
    samples = np.random.multivariate_normal(mu_true, sigma_true, size=100)
    mu = np.mean(samples, axis=0)
    sigma = (samples - mu).T @ (samples - mu) / len(samples)
    return mu, sigma

sigma1 = [[20, 0], [0, 10]]
calc_mu_sigma(mu, sigma1)

sigma1 = [[20, 14], [14, 10]]
calc_mu_sigma(mu, sigma1)

sigma1 = [[20, -14], [-14, 10]]
calc_mu_sigma(mu, sigma1)
```

3. (a) $P(Y|X,w) = X^Tw + N(0,1)$

$= N(X^Tw, 1)$

$= \frac{1}{\sqrt{2\pi}} e^{-\frac{(Y-X^Tw)^2}{2}}$

(b) • likelihood $= P(\{x_i, y_i\}|w)$

$= \frac{1}{(\sqrt{2\pi})^n} \prod_i e^{-\frac{(y_i - x_i^Tw)^2}{2}}$

$= \frac{1}{(\sqrt{2\pi})^n} e^{-\frac{1}{2}\sum_i (y_i - x_i^Tw)^2}$

prior $= \frac{\exp(-\frac{1}{2} w^T \Sigma^{-1} w)}{\sqrt{(2\pi)^d |\Sigma|}}$

posterior $\propto$ likelihood $\times$ prior

$= \exp[-\frac{1}{2}((Y-Xw)^T (Y-Xw) + w^T\Sigma^{-1}w)]$

$= \exp[-\frac{1}{2}(Y^TY - Y^TXw - w^TX^TY + w^TX^TXw + w^T\Sigma^{-1}w)]$

$\propto \exp[-\frac{1}{2} w^T(X^TX + \Sigma^{-1})w + Y^TXw]$

• So we can know posterior is a multivariate gaussian distribution with mean of $(X^TX + \Sigma^{-1})^{-1}X^TY$, so the MAP estimate of w is $(X^TX + \Sigma^{-1})^{-1}X^TY$.

• This is a generalization of ridge regression because if we increase Σ, prior on w will have a wider range. If we think about ridge regression, increasing Σ means decreasing λ, so w will have a wider range too.

(c) $P(Y|X, w) = \frac{1}{\sqrt{2\pi|\Sigma_2|}} e^{-\frac{1}{2}(Y-Xw-\mu_2)^T \Sigma_2^{-1} (Y-Xw-\mu_2)}$

Replace $\Sigma_2^{-1} = U S^{-1} U^T$

$$P(Y, X|w) = \exp\left(-\frac{1}{2}(Y-Xw-\mu_2)^T U S^{-1} U^T (Y-Xw-\mu_2)\right)$$

$$= \exp\left[-\frac{1}{2}(Y-\mu_2-Xw)(S^{-\frac{1}{2}}U^T)^T (S^{-\frac{1}{2}}U^T)(Y-\mu_2-Xw)\right]$$

Make $\tilde{X} = S^{-\frac{1}{2}} U^T X \qquad \tilde{Y} = S^{-\frac{1}{2}} U^T (Y-\mu_2)$

likelihood $\propto \exp\left[-\frac{1}{2}(\tilde{Y}-\tilde{X}w)^T(\tilde{Y}-\tilde{X}w)\right]$

So posterior $\propto \exp\left[-\frac{1}{2} w^T (\tilde{X}^T\tilde{X} + \Sigma^{-1}) w + \tilde{Y}^T \tilde{X} w\right]$

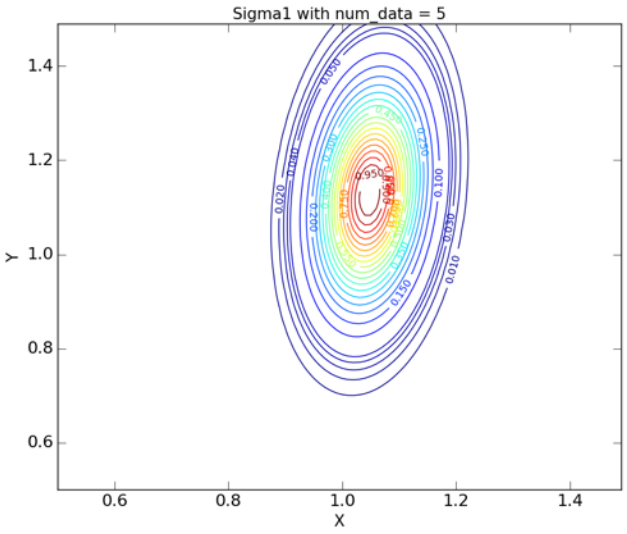
(d) see code and plots in next page.

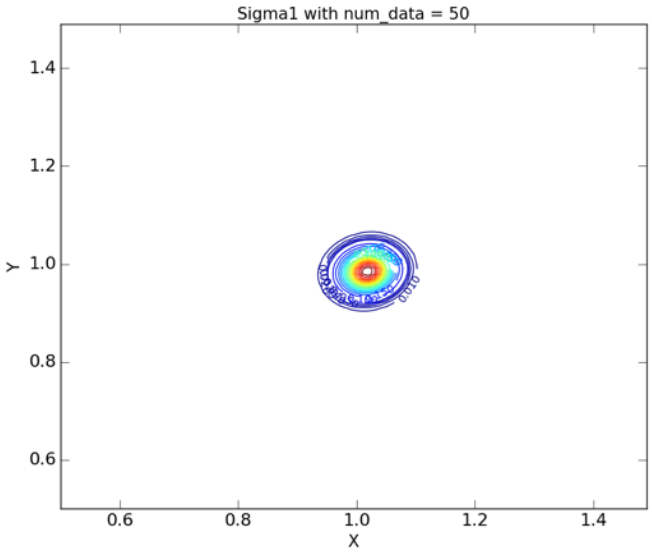
We can see that as the number of data points increases, the measured w shrinks to the true w in all priors.

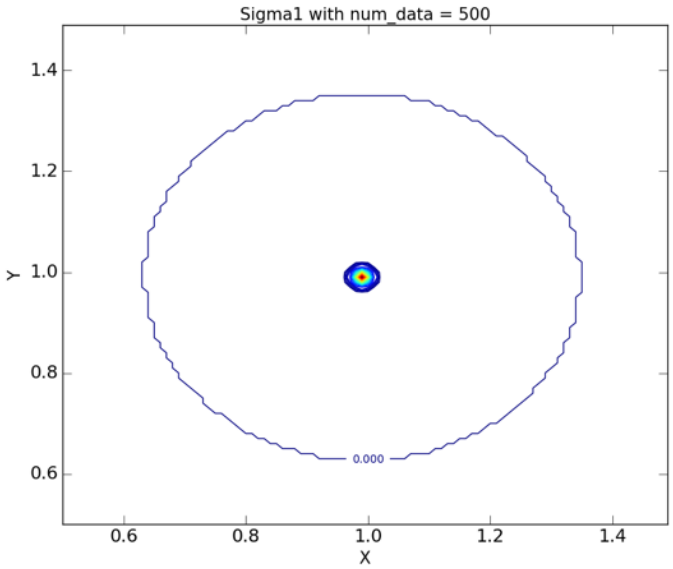
(e) see code and plots in next page.

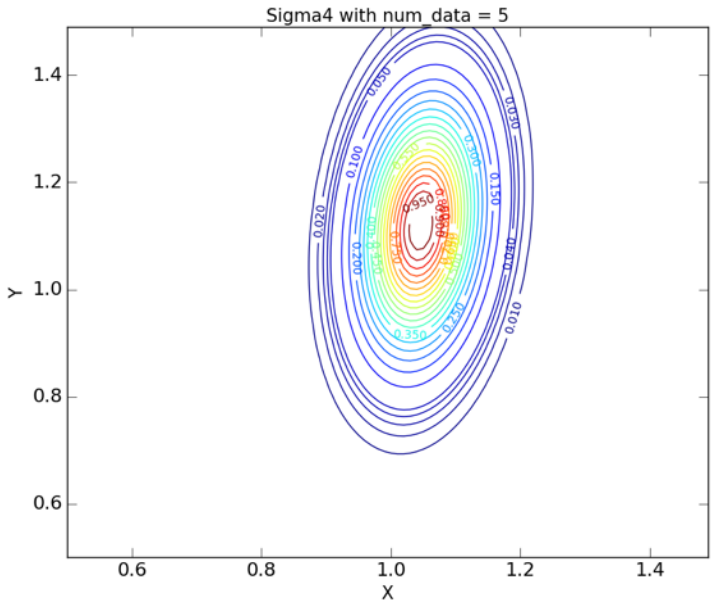
- When amount of training data increases, prior doesn't make a big difference, MSE ↓. because theoretical MSE $\propto \frac{1}{n}$, so MSE decrease with n, so MSE won't be affected by prior a lot
- "good" prior will make MSE smaller. sigma=3 is a good choice for our case.
- The influence of prior decrease with number of data points

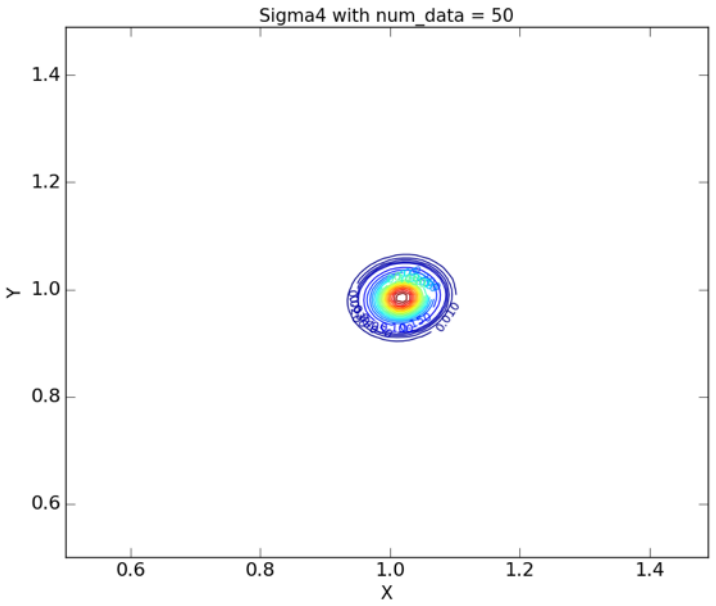
3 (d)

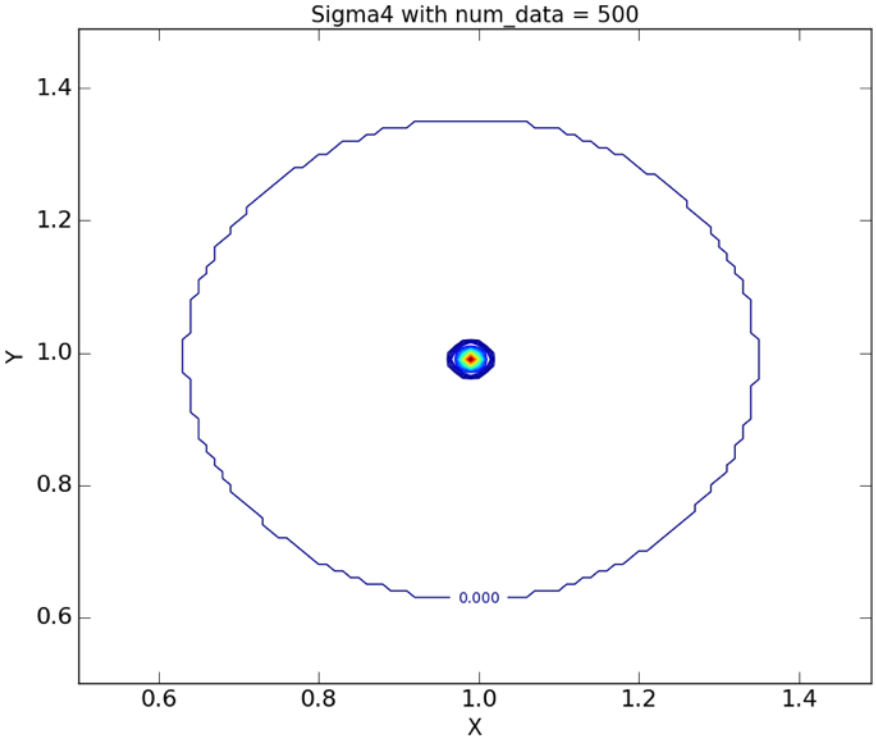

Sigma1 with num_data = 5
1.4
1.2
1.0
0.8
0.6
Y
0.6
0.8
1.0
1.2
1.4
X


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


Sigma1 with num_data = 500
1.4
1.2
1.0
0.8
0.6
Y
0.000
0.6
0.8
1.0
1.2
1.4
X


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

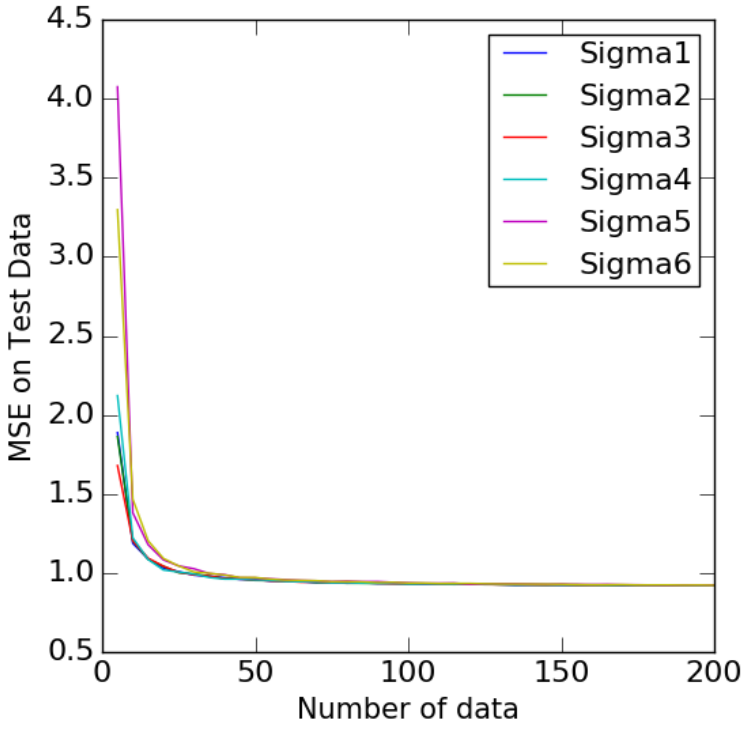

Sigma4 with num_data = 50
1.4
1.2
1.0
0.8
0.6
Y
0.6
0.8
1.0
1.2
1.4
X

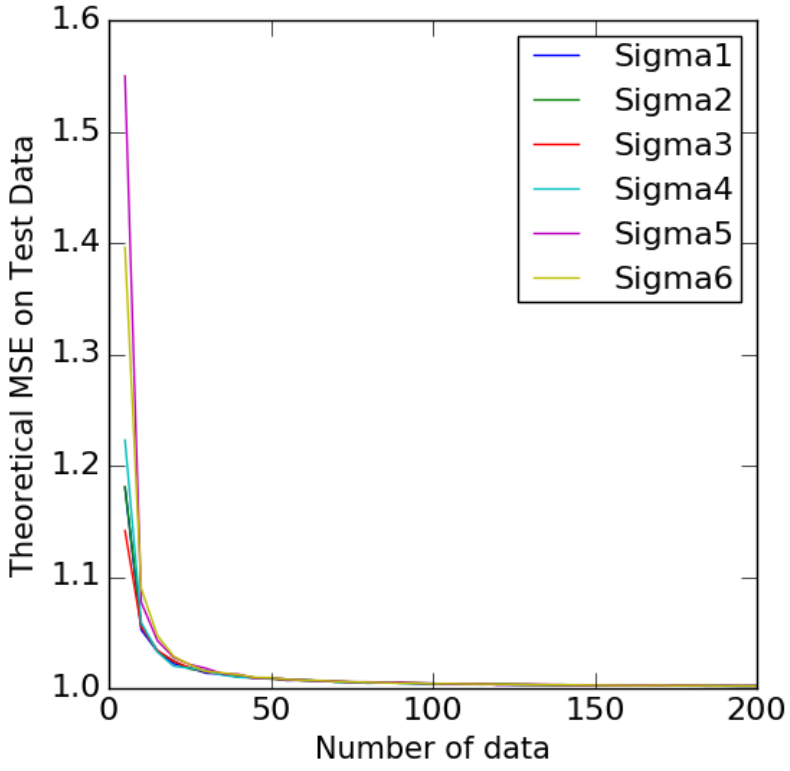

Sigma4 with num_data = 500
1.4
1.2
1.0
0.8
0.6
Y
0.000
0.6
0.8
1.0
1.2
1.4
X

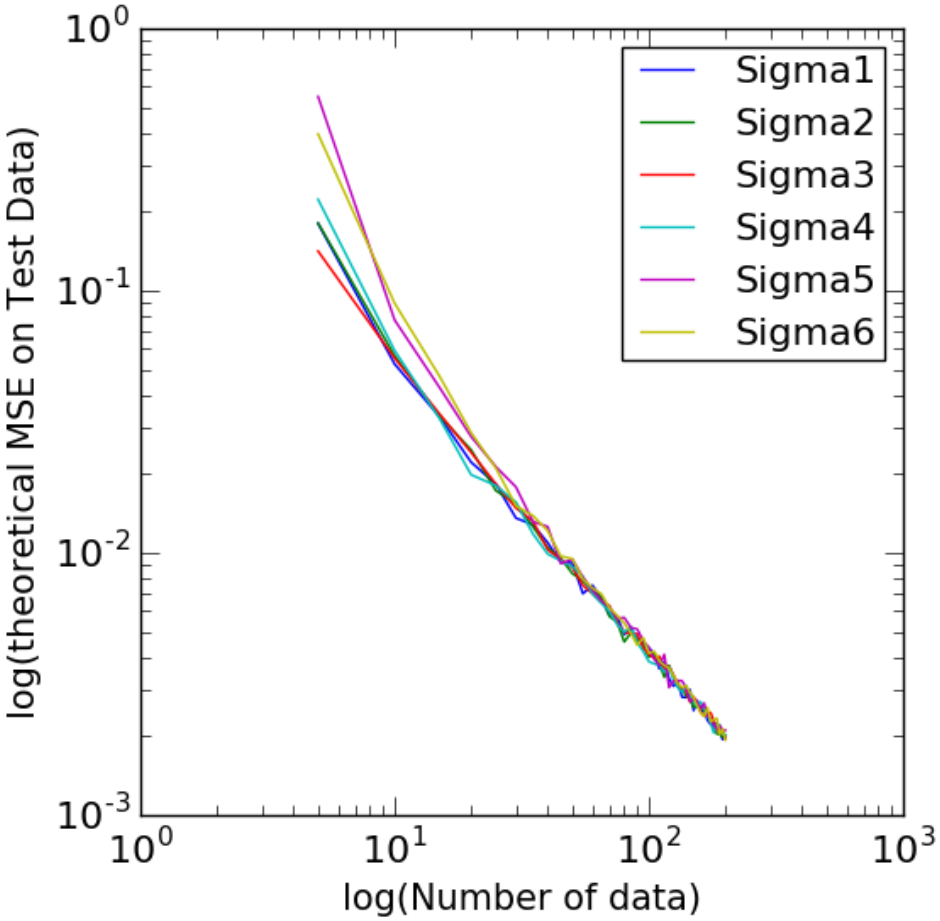
```
import matplotlib
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(0)

def generate_data(n):
    """
    This function generates data of size n.
    """
    #TODO implement this
    X = np.random.normal(loc=0, scale=5, size=(2,n))
    Z = np.random.normal(loc=0, scale=1, size=n)
    y = X[0] + X[1]+ Z
    return (X.T,y.T)

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

def compute_mean_var(X,y,Sigma):
    """
    This function computes the mean and variance of the posterior
    """
    #TODO implement this
    mux, muy = tikhonov_regression(X, y, Sigma)
    cov = (X.T @ X + np.linalg.inv(Sigma))
    sigmax = cov[0][0]
    sigmay = cov[1][1]
    sigmaxy = cov[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)# TODO compute the mean and covariance of posterior.

        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 = np.exp(-((X_grid-mux)**2*sigmax + (Y_grid-muy)**2*sigmay + 2*(X_grid-mux)*(Y_grid-muy)*sigmaxy)/2)
        # TODO Generate the function values of bivariate normal.

        # plot
        plt.figure(figsize=(10,10))
        CS = plt.contour(X_grid, Y_grid, Z/Z.max(),
                         levels = np.concatenate([np.arange(0,0.05,0.01),np.arange(0.05,1,0.05)]))
        plt.clabel(CS, inline=1, fontsize=10)
        plt.xlabel('X')
        plt.ylabel('Y')
        plt.title('Sigma'+ names[i] + ' with num_data = {}'.format(num_data))
        plt.savefig('Sigma'+ names[i] + '_num_data_{}.png'.format(num_data))
        plt.close()
```

3 (e)


Sigma1
Sigma2
Sigma3
Sigma4
Sigma5
Sigma6
MSE on Test Data
Number of data
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0
50
100
150
200


Sigma1
Sigma2
Sigma3
Sigma4
Sigma5
Sigma6
Theoretical MSE on Test Data
Number of data
1.6
1.5
1.4
1.3
1.2
1.1
1.0
0
50
100
150
200


Sigma1
Sigma2
Sigma3
Sigma4
Sigma5
Sigma6
log(theoretical MSE on Test Data)
log(Number of data)
10^0
10^{-1}
10^{-2}
10^{-3}
10^0
10^1
10^2
10^3

```python
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(0)
w = [1.0,1.0]
n_test = 100
n_trains = np.arange(5,205,5)
n_trails = 500

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.
    """
    #TODO implement this
    X = np.random.normal(loc=0, scale=5, size=(2,n))
    Z = np.random.normal(loc=0, scale=1, size=n)
    y = X[0] + X[1]+ Z
    return (X.T,y.T)

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

def compute_mse(X,Y, w):
    """
    This function computes MSE given data and estimated w.
    """
    #TODO implement this
    y_hat = X @ w
    mse = np.linalg.norm(Y - y_hat)**2/len(Y)
    return mse
```

```python
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)
            mses[i, j, seed] = compute_mse(X_test,y_test, w)
            theoretical_mses[i, j, seed] = compute_theoretical_mse(w)

# 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('log(Number of data)')
plt.ylabel('log(theoretical MSE on Test Data)')
plt.legend()
plt.savefig('log_theoretical_MSE.png')
```

4. (a) $\underset{\vec{w},\vec{r}}{\text{minimize}} \frac{1}{2}[\|\vec{r}\|^2 + \lambda\|\vec{w}\|^2]$ $\quad f(\vec{w},\vec{r}) =$

subject to $\vec{r} = X\vec{w} - \vec{y}$

Let $\mathcal{L}(\vec{w},\vec{r},\alpha)$

$= \frac{1}{2}[\|\vec{r}\|^2 + \lambda\|\vec{w}\|^2] + \alpha^T(\vec{r} - X\vec{w} + \vec{y})$

· Let $\theta(\alpha) = \max_{\alpha} \mathcal{L}(\vec{w},\vec{r},\alpha)$

if $\vec{r} = X\vec{w} - \vec{y}$, $\theta(\alpha) = f(\vec{w},\vec{r})$

if $\vec{r} \neq X\vec{w} - \vec{y}$, $\theta(\alpha) = \infty$.

So $\underset{\vec{w},\vec{r}}{\text{minimize}}\ f(\vec{w},\vec{r}) = \min_{\vec{w},\vec{r}} \max_{\alpha} \mathcal{L}(\vec{w},\vec{r},\alpha)$

(b) Because this is a linear problem,

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

To $\min_{\vec{w}\,\vec{r}} \mathcal{L}(\vec{w},\vec{r},\alpha) \Rightarrow \frac{\partial \mathcal{L}}{\partial \vec{w}} = \lambda\vec{w}^T - \alpha^T X = 0 \qquad \vec{w}^* = \frac{1}{\lambda} X^T\vec{\alpha}$.

$$\frac{\partial \mathcal{L}}{\partial \vec{r}} = \vec{r}^T + \alpha^T = 0 \qquad \vec{r}^* = -\vec{\alpha}$$

$$\mathcal{L}(\vec{w}^*,\vec{r}^*,\alpha) = \frac{1}{2}\vec{\alpha}^T\vec{\alpha} + \frac{1}{2\lambda}\vec{\alpha}^T X X^T\vec{\alpha} + \vec{\alpha}^T(-\vec{\alpha} - \frac{1}{\lambda} X X^T\vec{\alpha} + \vec{y})$$

$$= -\frac{1}{2}\alpha^T\alpha - \frac{1}{2\lambda}\alpha^T X X^T\alpha + \alpha^T y$$

$$\max L(\vec{w}^*,\vec{r}^*,\alpha) = \arg\max_{\alpha} -\frac{1}{2}[\alpha^T(XX^T + \lambda I)\alpha - \lambda\alpha^T y]$$

$$= \arg\min_{\alpha} [\tfrac{1}{2}\alpha^T(XX^T + \lambda I)\alpha - \lambda\alpha^T y]$$

(c) To $\min_{\alpha} g(\alpha) = \frac{1}{2}\alpha^T(k + \lambda I)\alpha - \lambda\alpha^T y$

$$\frac{\partial g(\alpha)}{\partial \alpha} = \alpha^T(k + \lambda I) - \lambda y^T = 0$$

$$\alpha = \lambda(k + \lambda I)^{-1} y$$

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

(a) $k(\vec{x}_i, \vec{x}_j) = (1 + \vec{x}_i^T \vec{x}_j)^2$

$= (1 + x_{i1}x_{j1} + x_{i2}x_{j2})^2$

$= 1 + 2x_{i1}x_{j1} + 2x_{i2}x_{j2} + 2x_{i1}x_{i2}x_{j1}x_{j2} + x_{i1}^2 x_{j1}^2 + x_{i2}^2 x_{j2}^2$

$= \begin{bmatrix} 1 & x_{i1}^2 & x_{i2}^2 & \sqrt{2}x_{i1} & \sqrt{2}x_{i2} & \sqrt{2}x_{i1}x_{i2} \end{bmatrix} \begin{bmatrix} 1 \\ x_{j1}^2 \\ x_{j2}^2 \\ \sqrt{2}x_{j1} \\ \sqrt{2}x_{j2} \\ \sqrt{2}x_{j1}x_{j2} \end{bmatrix}$

$\varphi(x) = \begin{bmatrix} 1 & x_1^2 & x_2^2 & \sqrt{2}x_1 & \sqrt{2}x_2 & \sqrt{2}x_1x_2 \end{bmatrix}$

$\Rightarrow k(\vec{x}_i, \vec{x}_j) = \langle \varphi(\vec{x}_i), \varphi(\vec{x}_j) \rangle$

Using kernal ridge regression. $\hat{y} = \phi(\vec{z})^T \vec{w}^* = \varphi(\vec{z})^T \cdot X^T (XX^T + \lambda I)^{-1} \vec{y}$.

using regularized Tikhonov ridge regression, $\hat{y} = \varphi(\vec{z})^T (X^T X + \lambda I)^{-1} X^T y$

$= \varphi(\vec{z})^T X^T (XX^T + \lambda I)^{-1} \vec{y}$.

so when $\Gamma^T \Gamma = \lambda I$, $\Gamma = \sqrt{\lambda} I$, kernal ridge regression is the same as regularized least square solution.

(c) Computing k term takes $O(n^2(l + \log p))$, and inverting takes $O(n^3)$ times X^T takes $O(n^3)$, times y takes $O(n^2)$, so computational complexity for kernal takes $O(n^2(l + \log p) + n^3)$.

Computing k using Tikhonov regression takes $O(d^2 n)$, inverting takes $O(d^3)$, $(d = \binom{l+p}{p})$

so total computation is $O(d^2 n + d^3)$

Actually doing prediction for kernal is $\underline{\langle \varphi(z), w^* \rangle} = O(l + \log p)$

for Tiknov is $O(d)$

5. (a) see plots in next page.

(b) see results and plots in next page.

(c) see error results in next page

From validation error, the best polynomial order is 13 when validation error = 0.
So we only want to high order polynomial when there are enough data points.

(d) see plots ~~on~~ next page.

(e) see error results and plots in next page.

When $\sigma \downarrow$, average squared loss $\downarrow$, it has similar effect of poly order.

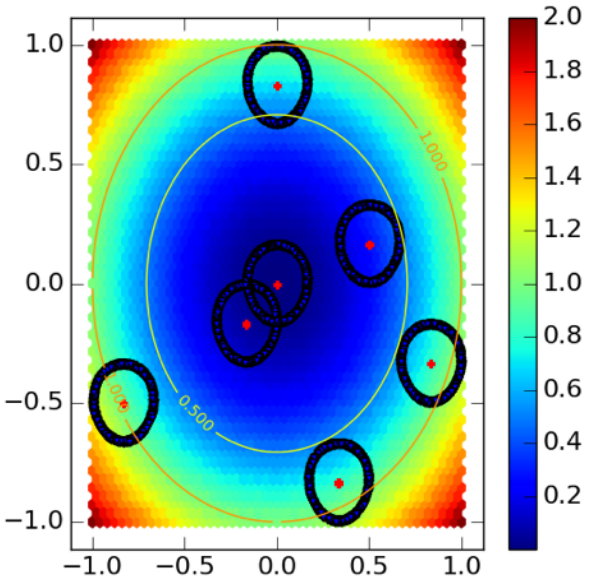
(f) Non-kernelized is more efficient.

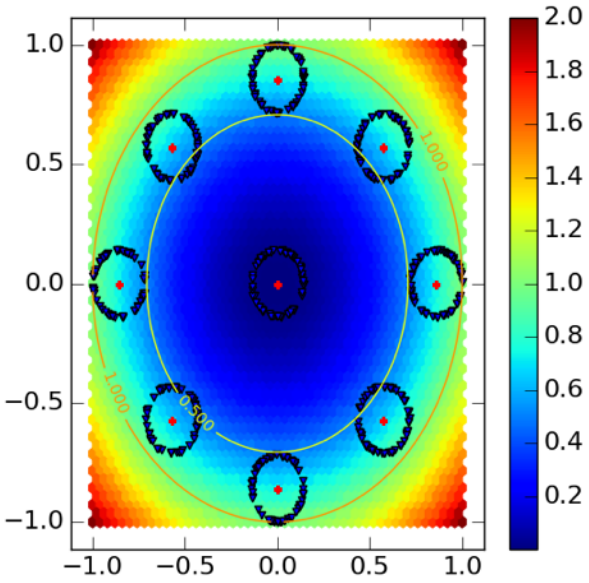
It's not possible to implement RBF in non-kernelized ridge regression, because it's an exponential formula, and it will expand to infinite polynomial terms.

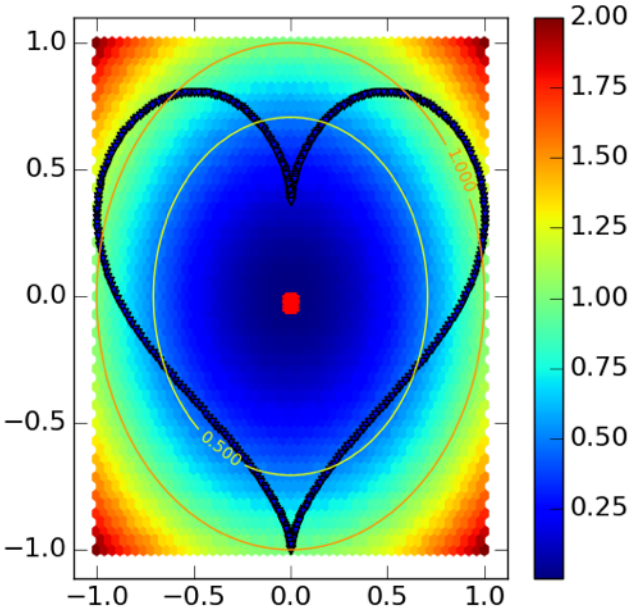
kernelized regression is prefered when there are more data in each X measurement and when polynomial order gets high.

(g) RBF and poly kernal is similar.

5. (a).


1.0
0.5
0.0
−0.5
−1.0
−1.0 −0.5 0.0 0.5 1.0
1.000
0.500
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2

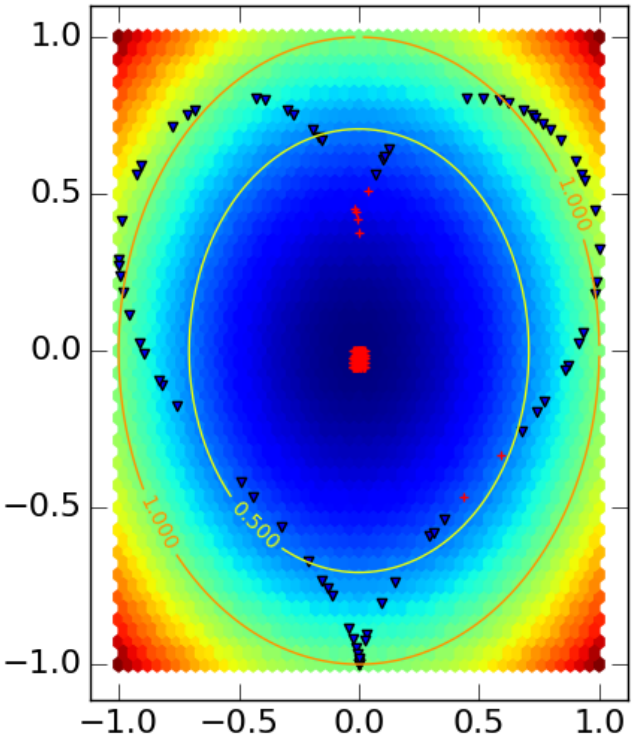

1.0
0.5
0.0
−0.5
−1.0
−1.0 −0.5 0.0 0.5 1.0
1.000
0.500
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2


1.0
0.5
0.0
−0.5
−1.0
−1.0 −0.5 0.0 0.5 1.0
1.000
0.500
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25

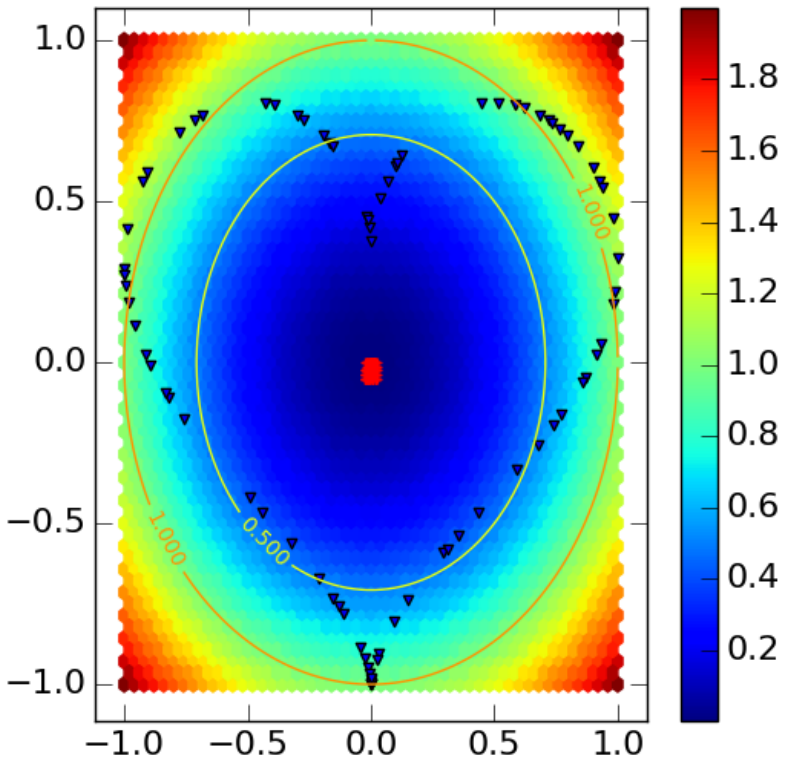
5. (b)

```
***  heart  ***
order     training_error      valid_error
    1   1.265255e+00   1.213930e+00
    2   2.739726e-01   1.393035e-01
    3   6.973848e-02   1.990050e-02
    4   0.000000e+00   0.000000e+00
    5   0.000000e+00   0.000000e+00
    6   0.000000e+00   0.000000e+00
    7   0.000000e+00   0.000000e+00
    8   0.000000e+00   0.000000e+00
    9   0.000000e+00   0.000000e+00
   10   0.000000e+00   0.000000e+00
   11   0.000000e+00   0.000000e+00
   12   0.000000e+00   0.000000e+00
   13   0.000000e+00   0.000000e+00
   14   0.000000e+00   0.000000e+00
   15   0.000000e+00   0.000000e+00
```

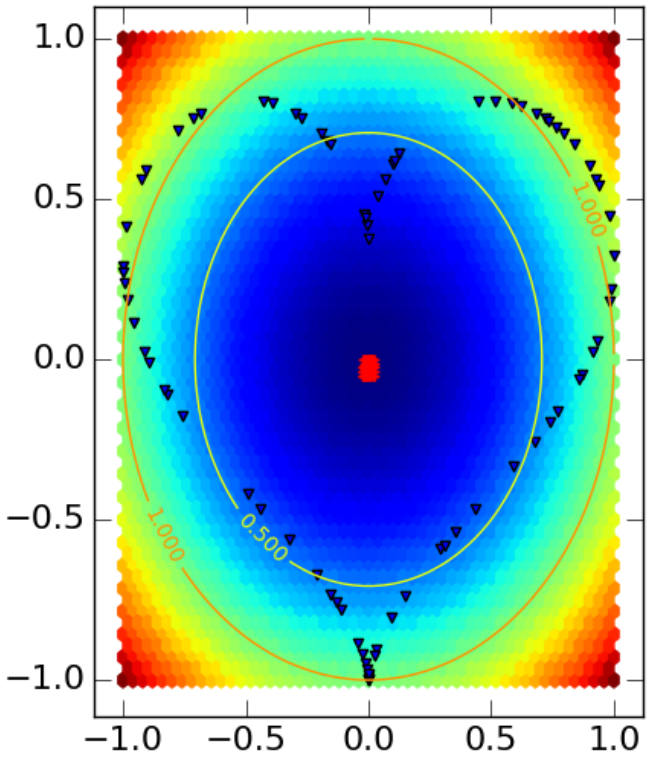
order 2



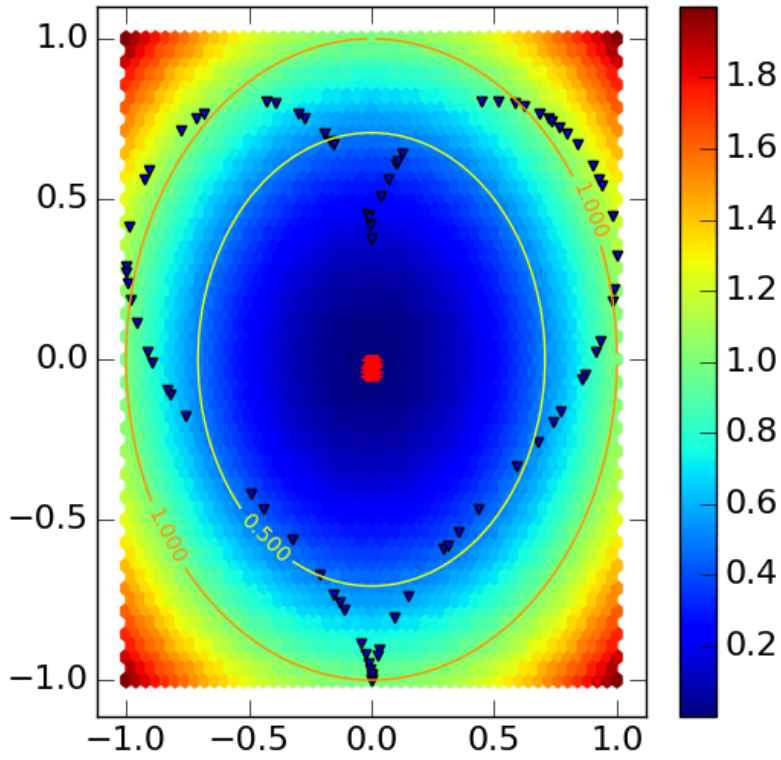
order 4



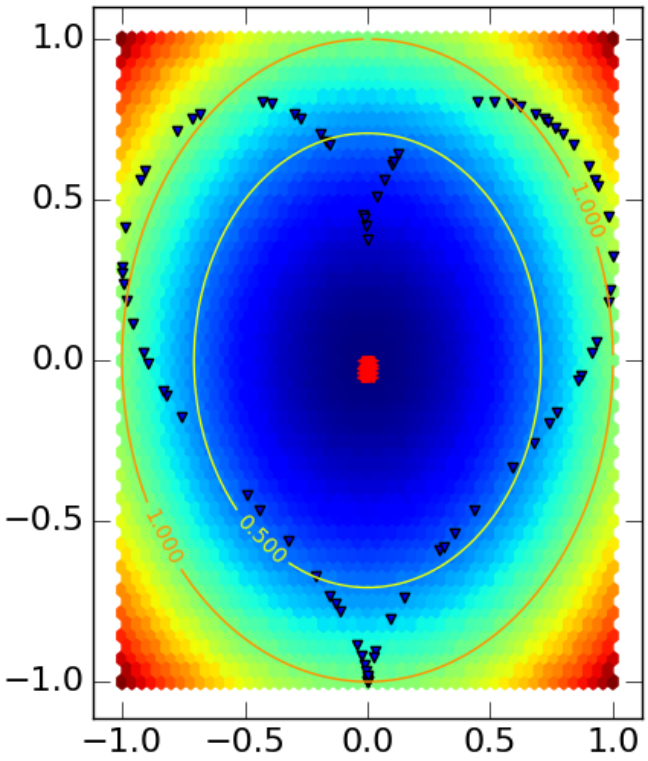
order 6



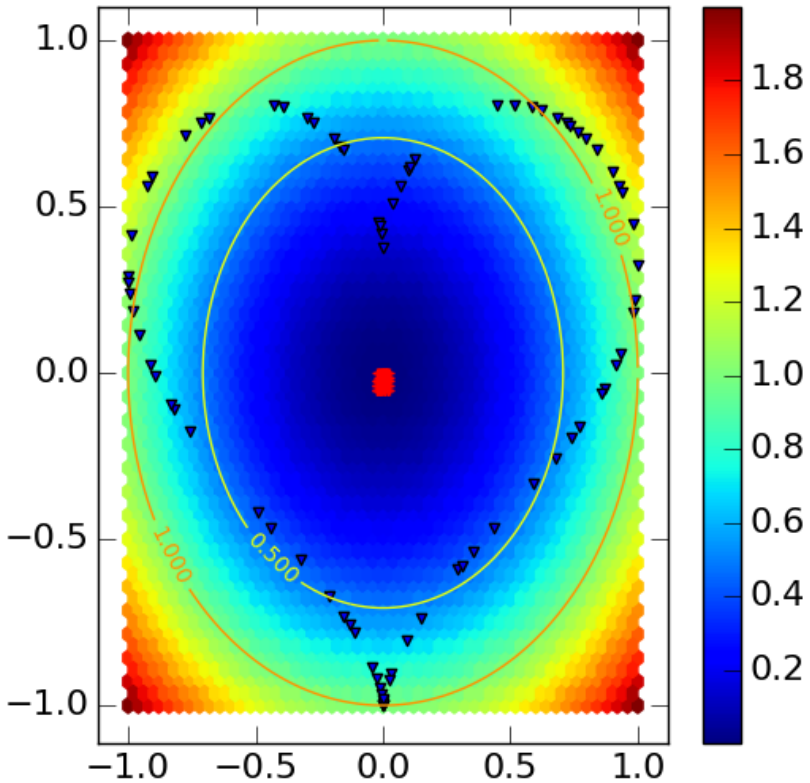
order 8



order 10



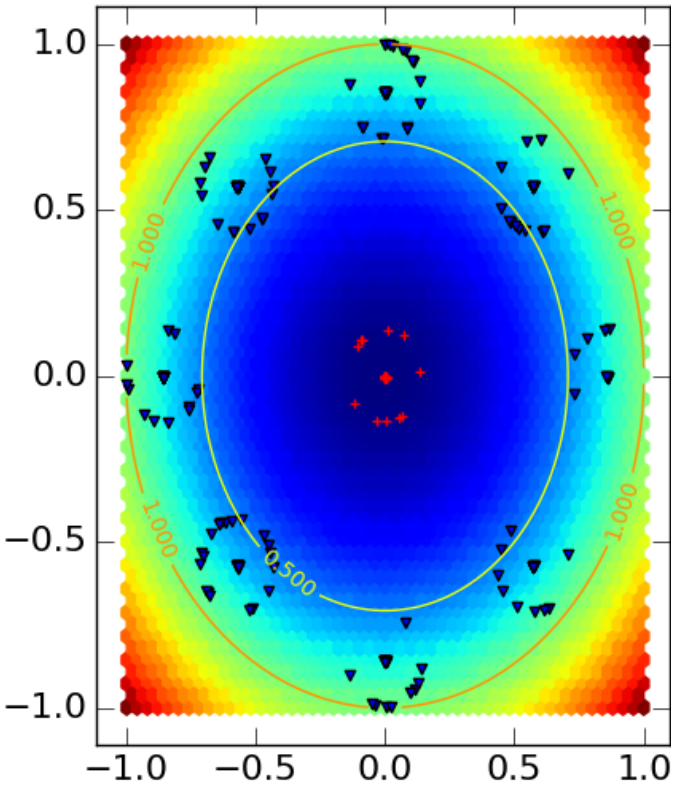
order 12



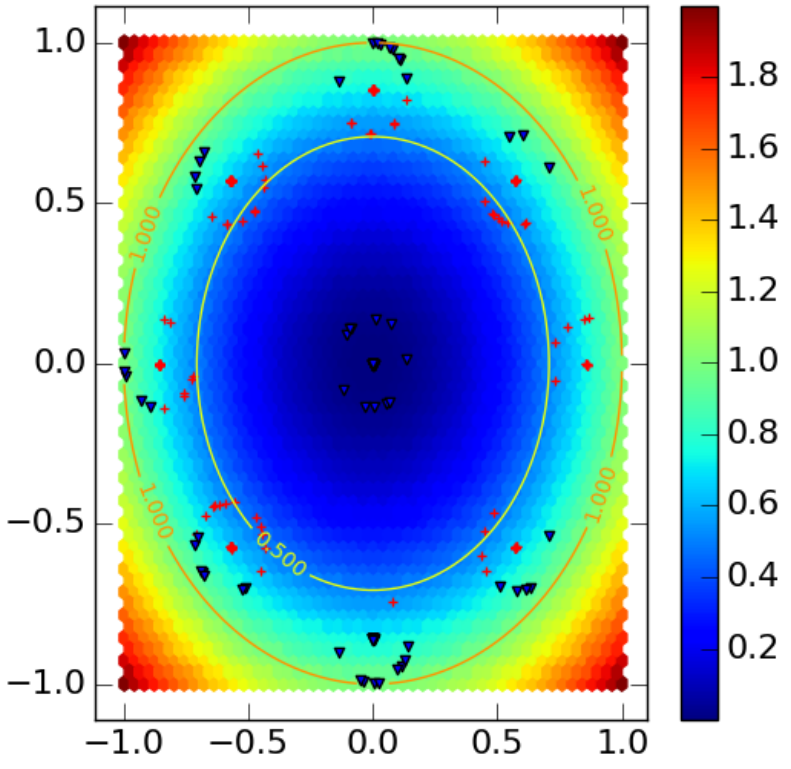
*** circle ***

order	training_error	valid_error
1	1.892683e+00	1.902913e+00
2	1.941463e+00	1.864078e+00
3	1.912195e+00	2.058252e+00
4	1.341463e+00	1.572816e+00
5	1.234146e+00	1.456311e+00
6	5.609756e-01	6.407767e-01
7	5.658537e-01	6.407767e-01
8	2.439024e-02	1.941748e-02
9	2.926829e-02	1.941748e-02
10	4.878049e-03	1.941748e-02
11	9.756098e-03	1.941748e-02
12	4.878049e-03	0.000000e+00
13	4.878049e-03	0.000000e+00
14	0.000000e+00	0.000000e+00
15	0.000000e+00	0.000000e+00

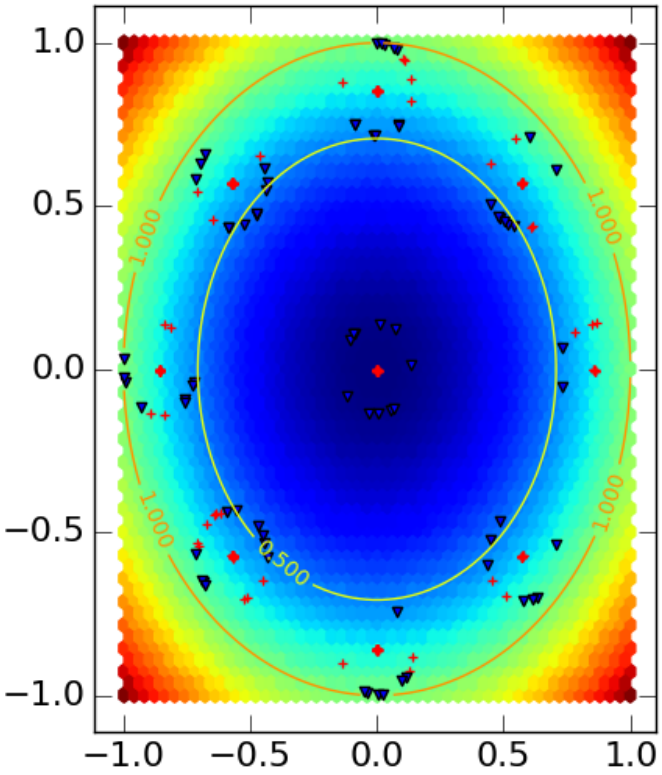
order 2



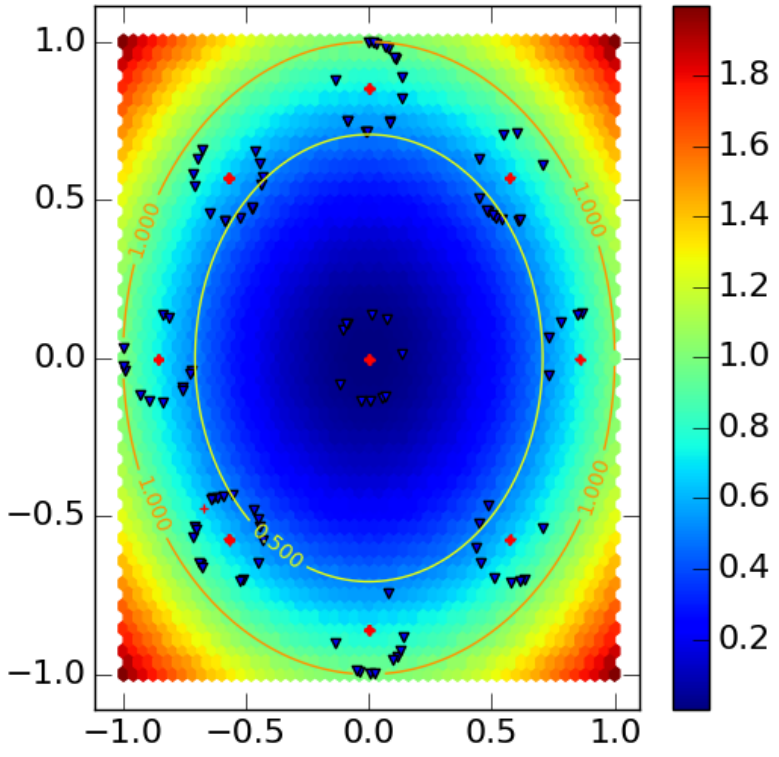
order 4



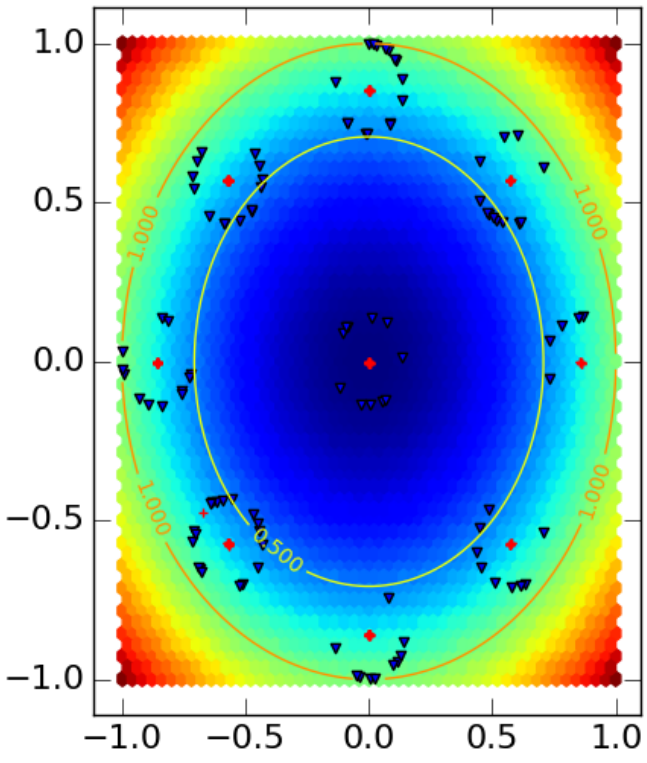
order 6



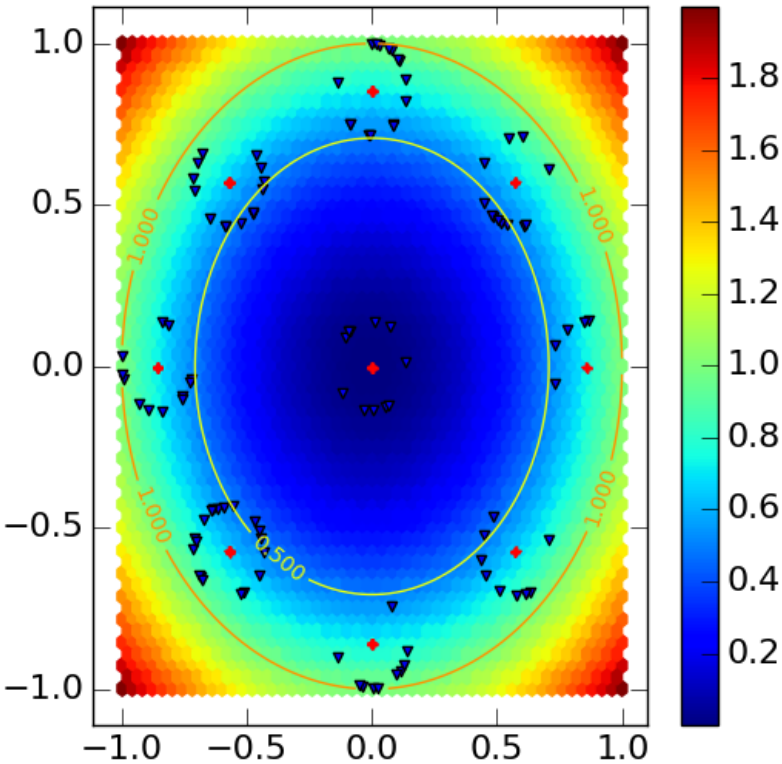
order 8



order 10



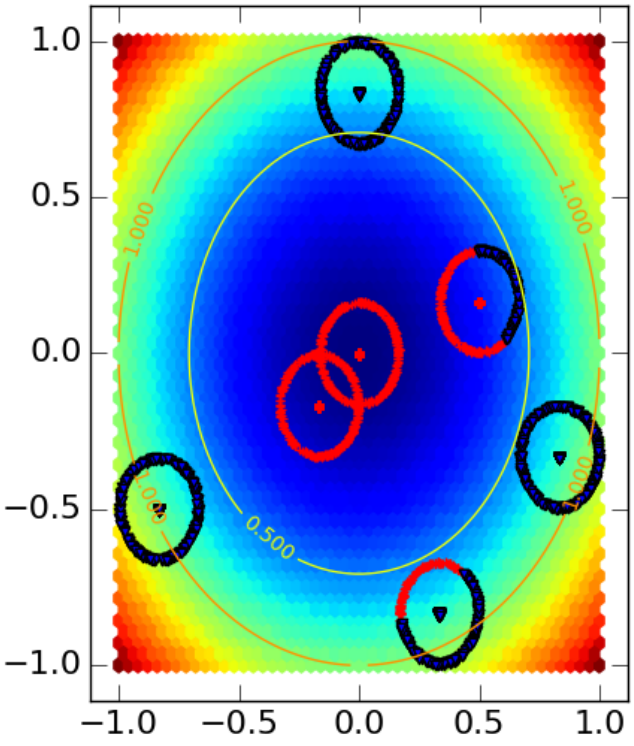
order 12



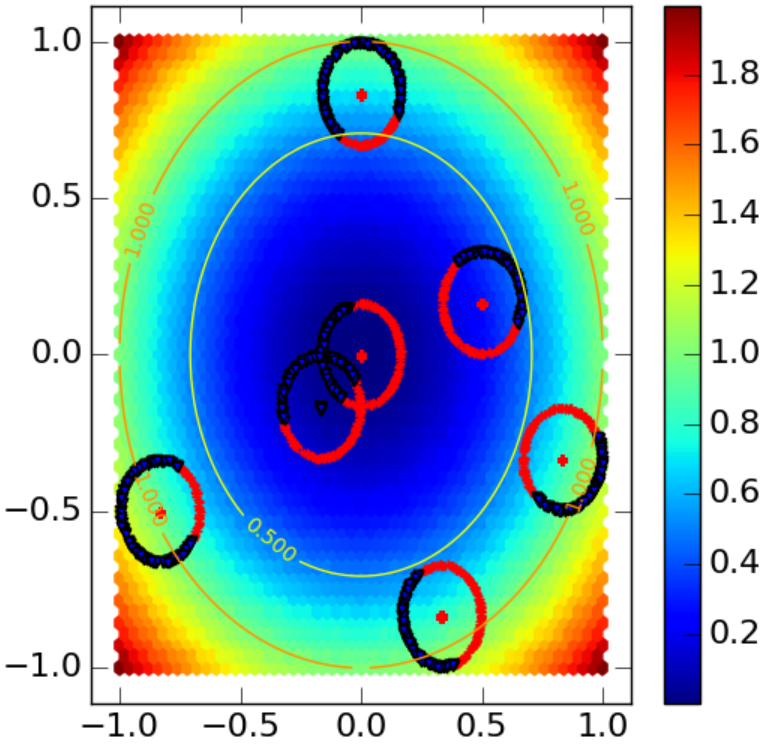
*** asymmetric ***

order	training_error	valid_error
1	1.990976e+00	2.055081e+00
2	1.982190e+00	2.025641e+00
3	2.437426e+00	2.468186e+00
4	1.252909e+00	1.209877e+00
5	9.515555e-01	9.335233e-01
6	2.218000e-01	1.728395e-01
7	1.885538e-01	1.547958e-01
8	1.517454e-01	1.111111e-01
9	1.405842e-01	1.006648e-01
10	1.363097e-01	1.006648e-01
11	1.325101e-01	9.591643e-02
12	1.332225e-01	9.686610e-02
13	1.332225e-01	9.591643e-02
14	1.322726e-01	9.591643e-02
15	1.315602e-01	9.306743e-02

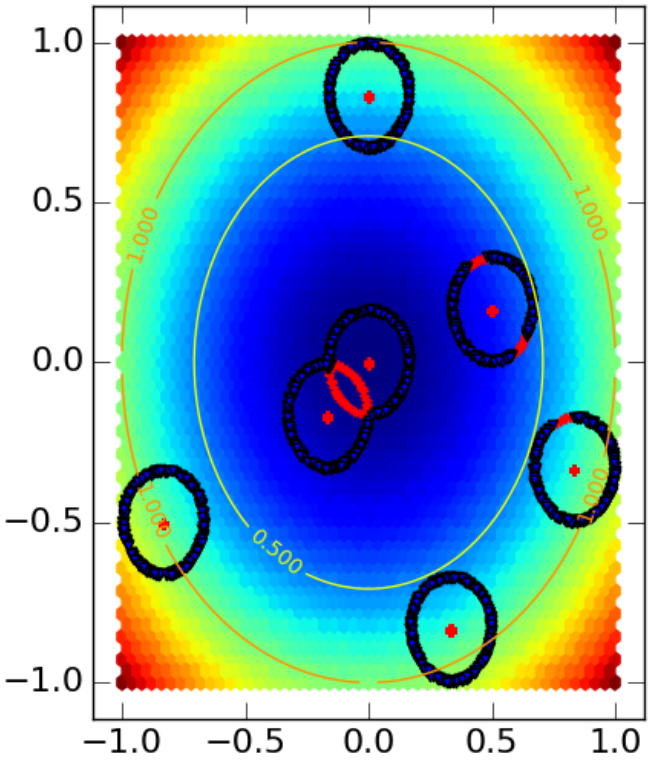
order 2



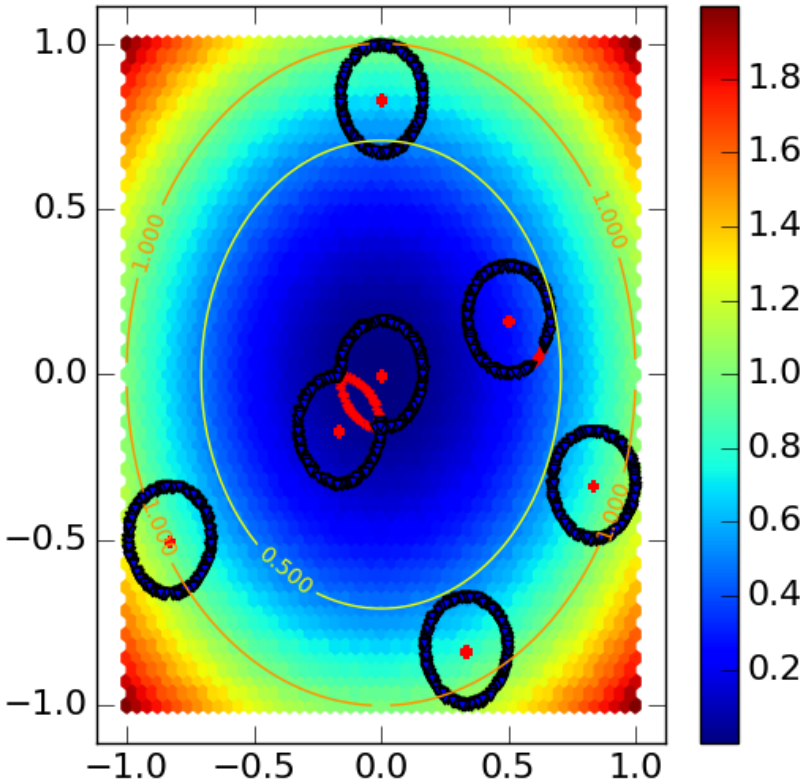
order 4



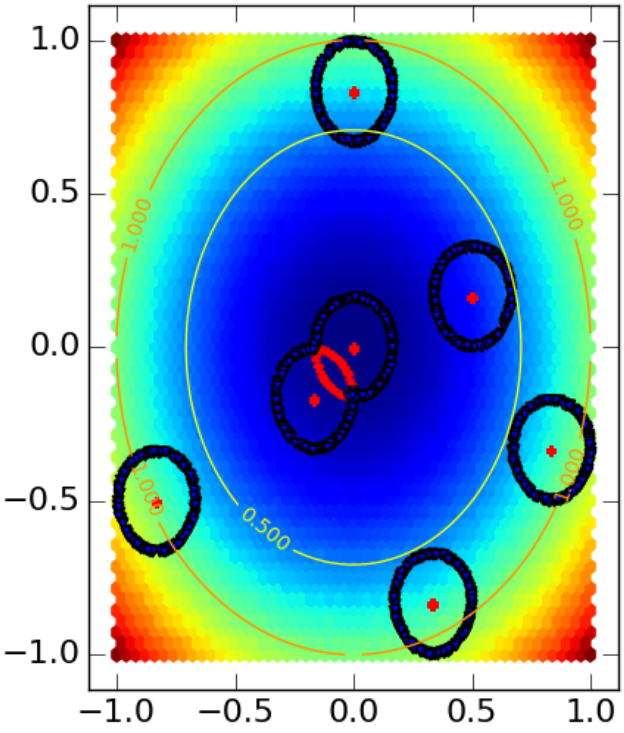
order 6



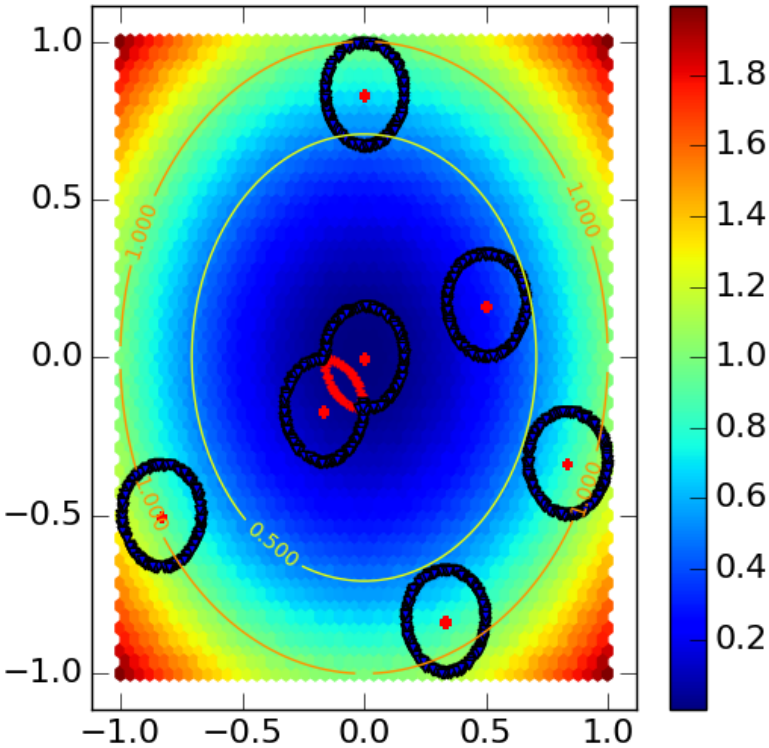
order 8



order 10



order 12



5.(c)

*** heart ***

order	training_error	valid_error
1	1.265255e+00	1.213930e+00
2	2.739726e-01	1.393035e-01
3	6.973848e-02	1.990050e-02
4	0.000000e+00	0.000000e+00
5	0.000000e+00	0.000000e+00
6	0.000000e+00	0.000000e+00
7	0.000000e+00	0.000000e+00
8	0.000000e+00	0.000000e+00
9	0.000000e+00	0.000000e+00
10	0.000000e+00	0.000000e+00
11	0.000000e+00	0.000000e+00
12	0.000000e+00	0.000000e+00
13	0.000000e+00	0.000000e+00
14	0.000000e+00	0.000000e+00
15	0.000000e+00	0.000000e+00

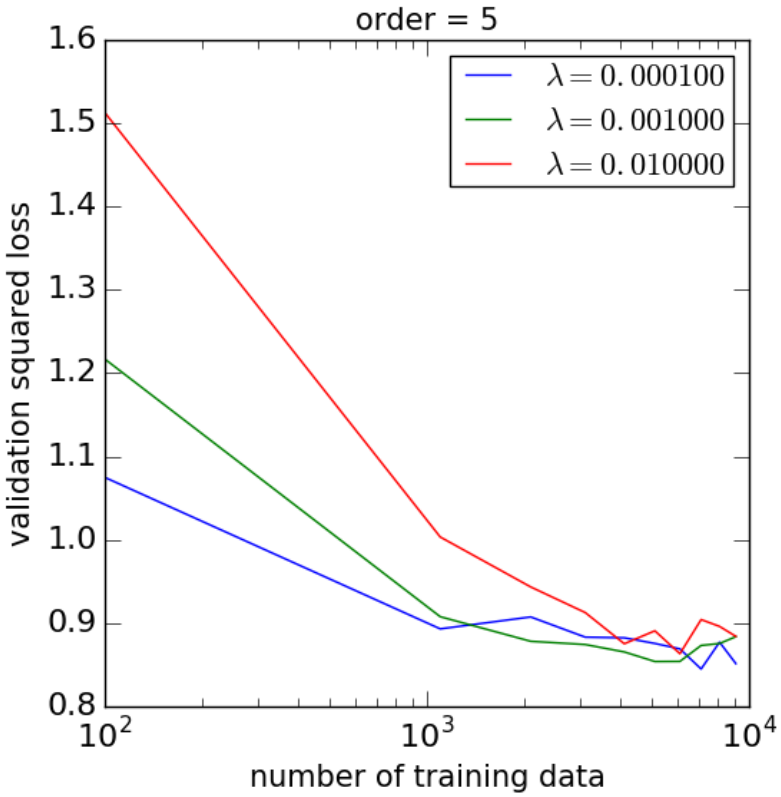
*** circle ***

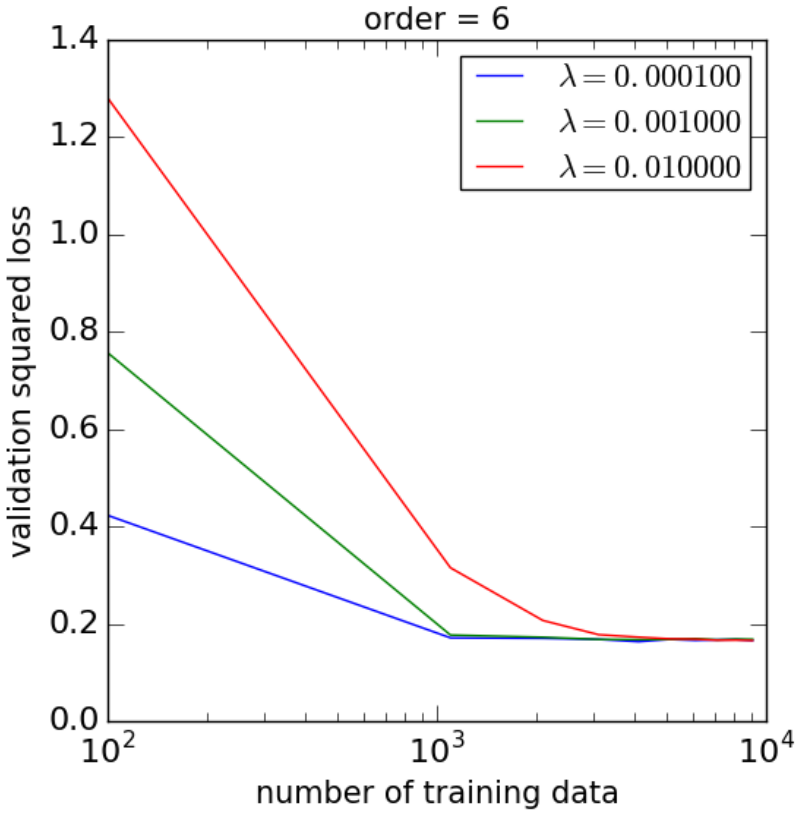
order	training_error	valid_error
1	1.892683e+00	1.902913e+00
2	1.941463e+00	1.864078e+00
3	1.912195e+00	2.058252e+00
4	1.341463e+00	1.572816e+00
5	1.234146e+00	1.475728e+00
6	5.609756e-01	6.213592e-01
7	5.609756e-01	6.796117e-01
8	0.000000e+00	0.000000e+00
9	0.000000e+00	0.000000e+00
10	0.000000e+00	0.000000e+00
11	0.000000e+00	0.000000e+00
12	0.000000e+00	0.000000e+00
13	0.000000e+00	0.000000e+00
14	0.000000e+00	0.000000e+00
15	0.000000e+00	0.000000e+00

*** circle_0.15_training ***

order	training_error	valid_error
1	1.620915e+00	1.796105e+00
2	1.333333e+00	1.649485e+00
3	1.699346e+00	2.286369e+00
4	1.411765e+00	1.947308e+00
5	1.202614e+00	1.901489e+00
6	5.490196e-01	8.659794e-01
7	5.228758e-01	7.101947e-01
8	0.000000e+00	1.786942e-01
9	0.000000e+00	1.924399e-01
10	0.000000e+00	1.970218e-01
11	0.000000e+00	1.970218e-01
12	0.000000e+00	1.603666e-01
13	0.000000e+00	0.000000e+00
14	0.000000e+00	1.374570e-02
15	0.000000e+00	1.832761e-02
16	0.000000e+00	1.191294e-01
17	0.000000e+00	1.282932e-01
18	0.000000e+00	1.512027e-01
19	0.000000e+00	1.649485e-01
20	0.000000e+00	1.695304e-01
21	0.000000e+00	2.107675e-01
22	0.000000e+00	2.611684e-01
23	0.000000e+00	3.436426e-01

5 (d).

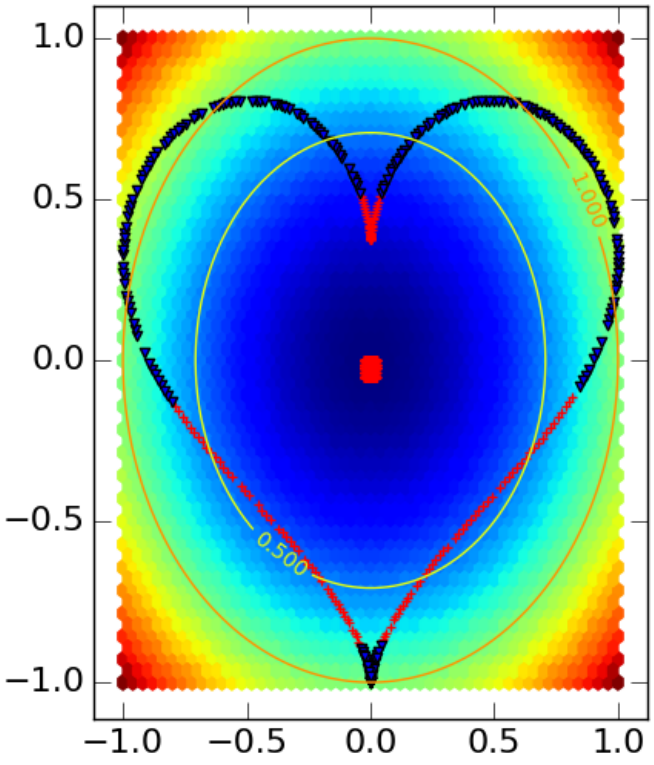

order = 5
validation squared loss
number of training data
$\lambda = 0.000100$
$\lambda = 0.001000$
$\lambda = 0.010000$
1.6
1.5
1.4
1.3
1.2
1.1
1.0
0.9
0.8
10^2
10^3
10^4


order = 6
validation squared loss
number of training data
$\lambda = 0.000100$
$\lambda = 0.001000$
$\lambda = 0.010000$
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
10^2
10^3
10^4

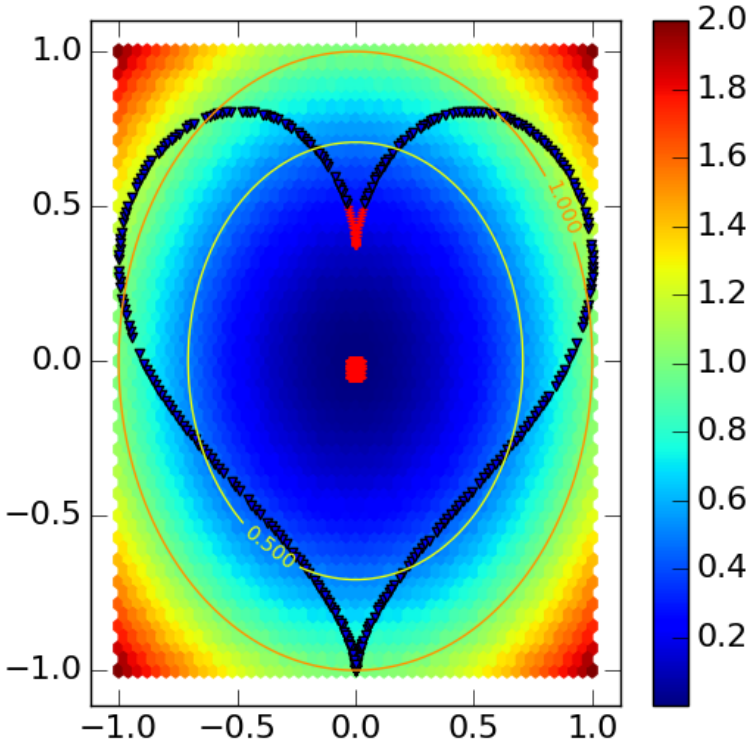
5 (e).

sigma	valid_error
10.00	6.697892e-01
3.00	1.967213e-01
1.00	0.000000e+00
0.30	0.000000e+00
0.10	0.000000e+00
0.03	0.000000e+00

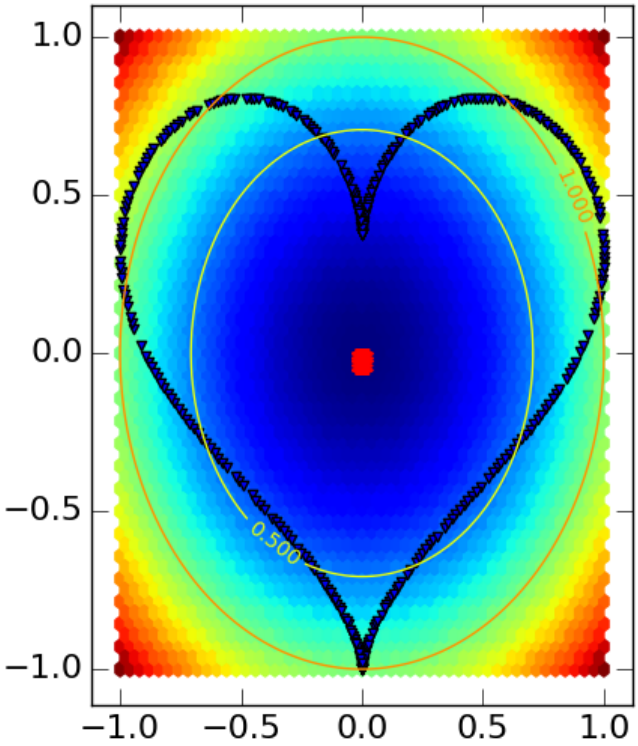
sigma 10



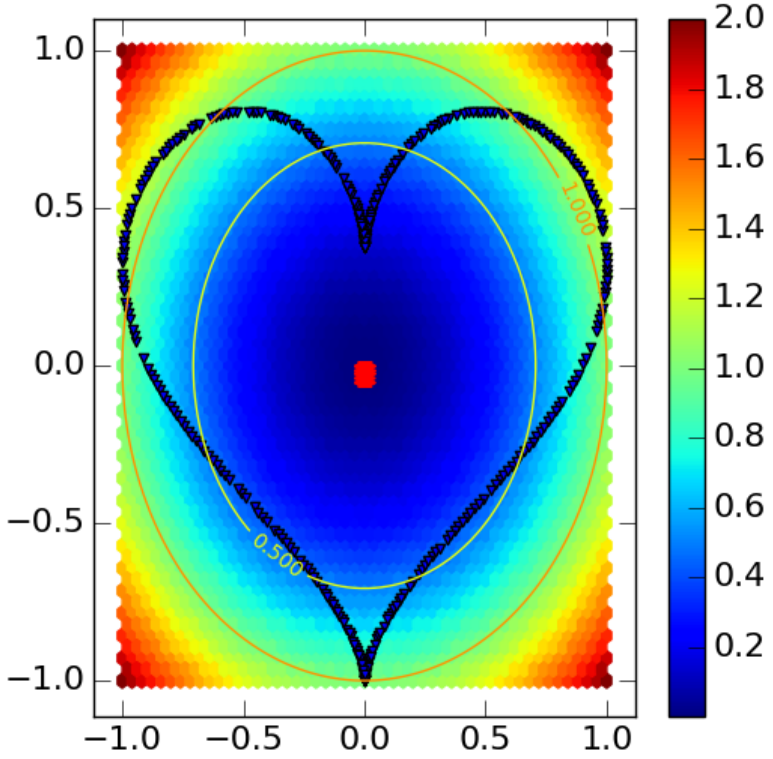
sigma 3



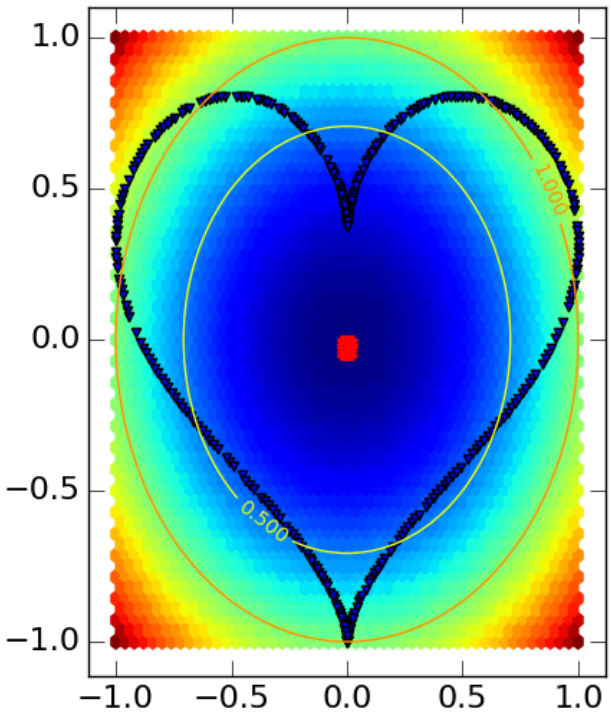
sigma 1



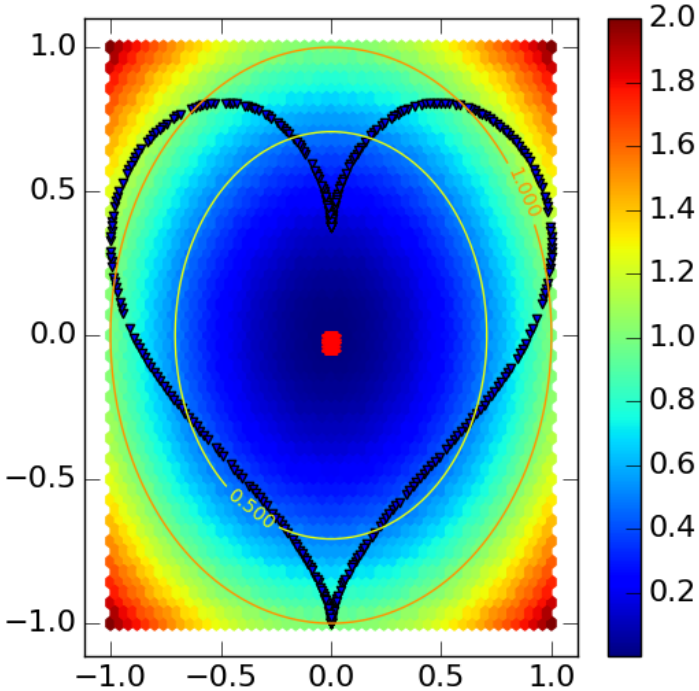
sigma 0.3



sigma 0.1



sigma 0.03



```
import matplotlib.pyplot as plt
from matplotlib import cm
import numpy as np

# choose the data you want to load
#data = np.load('circle.npz')
#pic = 'circle_0.15_training'
data = np.load('heart.npz')
pic = 'heart_noclip'
#data = np.load('asymmetric.npz')
#pic = 'asymmetric'

SPLIT = 0.15
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:]
n_valid = len(y_valid)

LAMBDA = 0.001

orders = np.arange(1,24)

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, p, pic, clip=5):
    # example: heatmap(lambda x, y: x * x + y * y)
    # clip: clip the function range to [-clip, clip] to generate a clean plot
    #   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.clf()
    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()
    #plt.savefig('pre_kernal_%s_order%d.png' %(pic,p), format='png')
    plt.savefig('pre_kernal_%s_sigma%.2f.png' %(pic,p), format='png')
```

```
def main():
    # loop through different orders


    ########   poly kernal #########
    #print('***  %s  *** '  %pic)
    #print('order     training_error      valid_error')
    #for p in orders:
    #    # derive kernal matrix
    #    K = (X_train @ X_train.T + 1)**p

    #    # calculate predicted position for training data
    #    temp =  np.linalg.inv((K+LAMBDA*np.identity(n_train))) @ y_train
    #    y_pre_train = [((z @ X_train.T+1)**p) @ temp for z in X_train]
    #    y_pre_train = np.sign(y_pre_train)

    #    # calculate predicted position for valid data
    #    y_pre_valid = [((z @ X_train.T+1)**p) @ temp for z in X_valid]
    #    y_pre_valid = np.sign(y_pre_valid)

    #    train_err = np.sum((y_pre_train - y_train)**2)/n_train
    #    valid_err = np.sum((y_pre_valid - y_valid)**2)/n_valid

    #    print('%5d   %e   %e' %(p, train_err, valid_err))
    #
    #    # example usage of heatmap
    #    heatmap(lambda x, y: x * x + y * y, X_valid, y_pre_valid, p, pic)

    ######   RBF kernal #########
    print('sigma          valid_error')
    sigmas = [10,3,1,0.3,0.1,0.03]
    for sigma in sigmas:
        # derive kernal matrix
        K = np.zeros((n_train, n_train))
        for i in range(n_train):
            for j in range(n_train):
                v = X_train[i] - X_train[j]
                K[i][j] = np.exp(-(v @ v.T)/(2*sigma**2))
        # calculate predicted position for valid data
        temp = np.zeros((n_valid, n_train))
        for i in range(n_valid):
            for j in range(n_train):
                v = X_valid[i] - X_train[j]
                temp[i][j] = np.exp(-(v @ v.T)/(2*sigma**2))


        y_pre_valid =  temp @ np.linalg.inv((K+LAMBDA*np.identity(n_train))) @ y_train
        y_pre_valid = np.sign(y_pre_valid)

        valid_err = np.sum((y_pre_valid - y_valid)**2)/n_valid

        print('%.2f    %e' %(sigma, valid_err))

        # example usage of heatmap
        heatmap(lambda x, y: x * x + y * y, X_valid, y_pre_valid, sigma, pic)
```

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import scipy.io as spio
from matplotlib import cm
import pdb

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


def multiPoly(D):
    order = []
    total_order = 0
    for i in np.arange(0,D+1):
        for j in np.arange(0,D+1):
            if i+j < D+1:
                order.append([i, j])
    return np.array(order)

def fit(X_train, y_train, X_valid, y_valid, D, LAMBDA):
    n_train = len(y_train)
    n_valid = len(y_valid)

    x1_train,x2_train = X_train.T
    x1_valid,x2_valid = X_valid.T

    # YOUR CODE TO COMPUTE THE AVERAGE ERROR PER SAMPLE
    # construct X matrix
    orders = multiPoly(D)
    X_D = [x1_train**order[0] * x2_train**order[1]  for order in orders]
    X_train_D = np.vstack(X_D).T
    X_D = [x1_valid**order[0] * x2_valid**order[1]  for order in orders]
    X_valid_D = np.vstack(X_D).T

   # use MLE to
   w = np.linalg.inv(X_train_D.T @ X_train_D + LAMBDA  * np.identity(len(orders))) @ X_train_D.T @ y_train.T
   y_pre_train = np.sign(X_train_D @ w)
   y_pre_valid = np.sign(X_valid_D @ w)
   train_err = np.sum((y_pre_train - y_train)**2)/n_train
   valid_err = np.sum((y_pre_valid - y_valid)**2)/n_valid

   #heatmap(lambda x, y: x * x + y * y, X_valid, y_pre_valid, D, pic)
   return(train_err, valid_err)

def heatmap(f, X, y, p, pic, clip=5):
   # example: heatmap(lambda x, y: x * x + y * y)
   # clip: clip the function range to [-clip, clip] to generate a clean plot
   #   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.clf()
   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()
   plt.savefig('pre_non_kernal_%s_order%d.png' %(pic,p), format='png')
```

```
def main():
    np.set_printoptions(precision=11)
    data = np.load('asymmetric.npz')
    pic = 'asymmetric'
    SPLIT = 0.8
    X = data["x"]
    y = data["y"]
    X /= np.max(X)  # normalize the data
    n_train = int(X.shape[0] * SPLIT)

    #poly_orders = np.arange(1,16)
    #LAMBDA = 0.001
    LAMBDA = np.array([0.0001,0.001, 0.01])
    poly_orders = np.array([5,6])
    #n_data = (np.arange(0.1,1,0.05)*n_train).astype('int')
    n_data = np.arange(100,10000,1000)
    n_circle = 100


    for i,p in enumerate(poly_orders):
        #print('***  %s  *** '  %pic)
        #print('order     training_error      valid_error')
        plt.figure()
        for j,l in enumerate(LAMBDA):
            train_errors = np.zeros((len(n_data),n_circle))
            valid_errors = np.zeros((len(n_data),n_circle))

            for k,n in enumerate(n_data):

                # sample training data
                n_train = int(X.shape[0] * SPLIT)
                X_train = X[:n_train:, :]
                y_train = y[:n_train]

                X_valid = X[n_train:, :]
                y_valid = y[n_train:]

                for h in range(n_circle):
                    idx = np.random.choice(np.arange(n_train), n)
                    train_errors[k][h], valid_errors[k][h] = fit(X_train[idx], y_train[idx], X_valid, y_valid, p, l)
```

```
            mean_train_error = np.mean(train_errors, axis=1)
            mean_valid_error = np.mean(valid_errors, axis=1)
            plt.plot(n_data, mean_valid_error, label=r'$\lambda=%f$' %l)
        plt.legend()
        plt.xlabel('number of training data')
        plt.ylabel('validation squared loss')
        plt.title('order = %d' %p)
        plt.gca().set_xscale('log')
        plt.savefig('order_%d_valid_err_vs_num_train.png' %p, format='png')
        plt.close()
```

6. Q: If A and B are nxn matrix, under what condition does $e^{A+B} = e^A e^B$?

A: $e^{A+B} = I + (A+B) + \frac{1}{2!}(A+B)^2 + \cdots$

$e^A = I + A + \frac{1}{2}A^2 + \cdots$

$e^B = I + B + \frac{1}{2}B^2$

$e^A e^B = I + A + B + AB + \frac{1}{2}A^2 + \frac{1}{2}B^2$

$e^{A+B} = I + A + B + \frac{1}{2}A^2 + \frac{1}{2}B^2 + \frac{1}{2}AB + \frac{1}{2}BA$

$\Rightarrow AB = \frac{1}{2}AB + \frac{1}{2}BA$

$\frac{1}{2}AB = \frac{1}{2}BA$

$\Rightarrow AB = BA.$