CS189 HW3

(a)

$$\theta^* = \arg\max_\theta \sum_{i=1}^{n} \log P(X_i \mid \theta) + n \log P(\theta)$$

$$= \arg\min_\theta \|X\theta - y\|_2^2 + \frac{\sigma^2}{\sigma_h^2}\|\theta - \mu_\theta\|_2^2$$

$$\nabla_\theta \left(\|X\theta - y\|_2^2 + \frac{\sigma^2}{\sigma_h^2}\|\theta - \mu_\theta\|_2^2\right)$$

$$= \nabla_\theta \left((X\theta - y)^T(X\theta - y) + \frac{\sigma^2}{\sigma_h^2}(\theta-\mu_\theta)^T(\theta-\mu_\theta)\right)$$

$$= \nabla_\theta \left((X\theta)^T X\theta - (X\theta)^T y - y^T(X\theta) + y^T y + \frac{\sigma^2}{\sigma_h^2}(\theta^T\theta - \theta^T\mu_\theta - \mu_\theta^T\theta + \mu_\theta^T\mu_\theta)\right)$$

$$= 2X^TX\theta - 2X^Ty + \frac{2\sigma^2}{\sigma_h^2}(\theta - \mu_\theta) = 0$$

$$X^TX\theta^* - X^Ty + \frac{\sigma^2}{\sigma_h^2}(\theta^* - \mu_\theta) = 0$$

$$\theta^* = \left(X^TX + \frac{\sigma^2}{\sigma_h^2}I\right)^{-1}\left(X^Ty + \frac{\sigma^2}{\sigma_h^2}\mu_\theta\right)$$

(b) The two figures for $\sigma_h = 1$ and $\sigma_h = 10$ are included in Appendix. The code is in problem1-starter.py from # Generating Data and Labels with Random Gaussian Noise to the end. After changing σ_h to 10, The range and variation of prior obviously increases. This is because σ_h is the standard deviation of prior distribution. σ_h increases means the variance of prior increases.

Code also includes the implement of MLE, MAP and prior.

(c) The two figures for $\mu_\theta = \begin{bmatrix}-5\\-5\end{bmatrix}$ and $\begin{bmatrix}3\\3\end{bmatrix}$ are included in Appendix. The code is in problem1-starter.py.

When change to $\begin{bmatrix}-5\\5\end{bmatrix}$, the center of prior plot moves from (0, 0) to (-5, 5). When change to $\begin{bmatrix}3\\3\end{bmatrix}$, the center of prior plot moves from (-5, 5) to (3, 3). θ_{MLE} doesn't change according to the plots with respect to prior

(d)

$$\theta_{MAP} = \arg\max_{\theta\in\mathbb{R}^d} \sum_{i=1}^{n} \log P(X_i \mid \theta) + \log P(\theta)$$

$$= \arg\max_{\theta\in\mathbb{R}^d} \sum_{i=1}^{n} \log\left(\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y_i - X_i\theta_i)^2}{2\sigma^2}}\right) + \log\left(\frac{1}{2\sigma_h} e^{-\frac{|\theta-\mu|_1}{\sigma_h}}\right)$$

$$= \arg\min_{\theta\in\mathbb{R}^d} \sum_{i=1}^{n} \frac{(y_i - X_i\theta_i)^2}{2\sigma^2} + \frac{|\theta-\mu|_1}{\sigma_h}$$

$$= \arg\min_{\theta\in\mathbb{R}^d} \sum_{i=1}^{n} (y_i - X_i\theta_i)^2 + \frac{2\sigma^2}{\sigma_h}|\theta-\mu|_1$$

$$= \arg\min_{\theta\in\mathbb{R}^d} \|y - X\theta\|^2 + \frac{2\sigma^2}{\sigma_h}|\theta-\mu|_1$$

(e) The two plots for $p(\theta_i) \sim L(0,1)$ and $p(\theta_i) \sim L(0, 0.0001)$
The code is in problem1_starter.py. With implement of Laplacian_prior and MAP_contour.
After change σ_h from 1 to 0.0001 the weight of prior in θ_{MAP} increases a lot.
This is because: According to the result in (d)

$$\theta_{MAP} = \underset{\theta \in R^d}{\arg\min} \|Y - X\theta\|^2 + 2\frac{\sigma^2}{\sigma_h^2}|\theta - \mu|,$$

This is the prior term in θ_{MAP}, its weight is $\frac{2\sigma^2}{\sigma_h^2}$. If I change σ_h from 1 to 0.0001, The weight changes to 10^8 times before.

(f) If prior of θ is $Unif(0,1)$ then $P(\theta) = 1$

$\because Y = X\theta + Z \quad Z \sim N(0,1) \therefore Y \sim N(X\theta, 1)$

$$\therefore \theta_{MAP} = \underset{\theta \in R^d}{\arg\min} \|Y - X\theta\|^2 + 2\sigma^2 \times \log P(\theta)$$

$$= \underset{\theta \in R^d}{\arg\min} \|Y - X\theta\|^2 + 2\sigma^2(\log 1)$$

$$= \underset{\theta \in R^d}{\arg\min} \|Y - X\theta\|^2 = \theta_{MLE}$$

So, now θ_{MAP} and θ_{MLE} are equivalent.

2. Probabilistic Model of Linear regression

(a) $\because Y = Xw_1 + w_0 + Z, \quad Z \sim N(0, 1)$

$\therefore Y | X = x \sim N(xw_1 + w_0, 1)$

$$P(Y|X) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(Y-(Xw_1+w_0))^2}{2}}$$

(b) Log likelihood function

$$\underset{w}{\arg\max} \sum_{i=1}^{n} \log\left(\frac{1}{\sqrt{2\pi}} e^{-\frac{(Y_i-(X_iw_1+w_0))^2}{2}}\right)$$

$$= \underset{w}{\arg\min} \sum_{i=1}^{n} (Y_i - (X_i w_1 + w_0))^2$$

$$= \underset{w}{\arg\min} \| Y - Xw_1 - w_0 * \mathbf{1} \|^2 \qquad \mathbf{1} \in R^n \quad \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$$

$$= \underset{w}{\arg\min} (Y - Xw_1 - w_0\mathbf{1})^T (Y - Xw_1 - w_0\mathbf{1})$$

$$= \underset{w}{\arg\min} (Y^T - w_1X^T - w_0\mathbf{1}^T)(Y - Xw_1 - w_0\mathbf{1})$$

$$= \underset{w}{\arg\min} (Y^TY - Y^TXw_1 - Y^Tw_0\mathbf{1} - w_1X^TY + w_1X^TXw_1 + w_1X^Tw_0\mathbf{1} - w_0\mathbf{1}^TY + w_0\mathbf{1}^TXw_1 + w_0\mathbf{1}^Tw_0\mathbf{1})$$

$$= \underset{w}{\arg\min} \sum_{i=1}^{n} Y_i^2 - 2Y^TXw_1 - 2w_0\sum_{i=1}^{n} Y_i + w_1^2X^TX + 2w_0w_1\sum_{i=1}^{n} X_i + nw_0^2$$

$$\begin{cases} \nabla w_1 = -2X^TY + 2X^TXw_1 + 2w_0\sum_{i=1}^{n} X_i \\ \nabla w_0 = -2\sum_{i=1}^{n} Y_i + 2w_1\sum_{i=1}^{n} X_i + 2nw_0 \end{cases}$$

$$\begin{cases} -2X^TY + 2X^TXw_1 + 2\sum_{i=1}^{n} X_i w_0 = 0 \\ -2\sum_{i=1}^{n} Y_i + 2w_1\sum_{i=1}^{n} X_i + 2nw_0 = 0 \end{cases}$$

$$\begin{cases} w_1 = \dfrac{(\sum_{i=1}^{n} X_i)(\sum_{i=1}^{n} Y_i) + n\sum_{i=1}^{n} X_iY_i}{n\sum_{i=1}^{n} X_i^2 - (\sum_{i=1}^{n} X_i)^2} \\ w_0 = \dfrac{(\sum_{i=1}^{n} Y_i)(\sum_{i=1}^{n} X_i^2) + (\sum_{i=1}^{n} X_iY_i)(\sum_{i=1}^{n} X_i)}{(\sum_{i=1}^{n} X_i)^2 - n\sum_{i=1}^{n} X_i^2} \end{cases}$$

(c) $\because Z \sim U[-0.5, 0.5] \quad Y = Xw + Z$

$\therefore Y | X \sim U[-0.5 + Xw, 0.5 + Xw]$

(d) $Y_1 = X_1 W + Z_1$ log likelihood function

$\vdots$ $\log P(Y=Y_1 \mid X=X_1) \times \cdots \times P(Y=Y_n \mid X=X_n)$

$Y_n = X_n W + Z_n$ $= \sum_{i=1}^{n} \log P(Y=Y_i \mid X=X_i)$

$\because$ when $-0.5 + X_i W \le Y_i \le 0.5 + X_i W$, $P(Y=Y_i \mid X=X_i) = 1$ otherwise it is 0

$\therefore$ To maximize log likelihood function, I need to make $-0.5 + X_i W \le Y_i \le 0.5 + X_i W$ hold for as many (X_i, Y_i) as possible

$\therefore -0.5 + X_i W \le Y_i \le 0.5 + X_i W \Rightarrow \frac{Y_i - 0.5}{X_i} \le W \le \frac{Y_i + 0.5}{X_i}$

$\therefore$ I think one appropriate estimate for W_{MLE} is : $\frac{1}{n} \sum_{i=1}^{n} \frac{Y_i}{X_i}$

(e) $Y_i = X_i W + Z_i$ $Z_i \sim N(0, 1)$

$Y_i \sim N(X_i W, 1)$ $W \sim N(0, \sigma^2)$

$\log P(W \mid Y, X) \propto \log P(Y \mid W, X) + \log P(W)$

$$= \sum_{i=1}^{n} \log P(Y_i \mid X_i W) + \log P(W)$$

$$\propto -\sum_{i=1}^{n} \frac{(Y_i - X_i W)^2}{2} - \frac{W^2}{2\sigma^2}$$

$$\propto -\frac{\sigma^2 \sum_{i=1}^{n} (Y_i - X_i W)^2 + W^2}{2\sigma^2}$$

$$\propto \frac{\left(\sum_{i=1}^{n} X_i^2 \sigma^2 + 1\right) W^2 - \sum_{i=1}^{n} 2\sigma^2 X_i Y_i W + \sum_{i=1}^{n} \sigma^2 Y_i^2}{2\sigma^2}$$

$$\propto \frac{\left(W - \frac{\sum_{i=1}^{n} \sigma^2 X_i Y_i}{\sum_{i=1}^{n} \sigma^2 X_i^2 + 1}\right)^2}{\frac{2\sigma^2}{\sum_{i=1}^{n} X_i^2 \sigma^2 + 1}}$$

$\therefore P(W \mid Y, X) \sim N(\mu', \sigma'^2)$ $\therefore$ posterior mean

where $\mu' = \frac{\sum_{i=1}^{n} \sigma^2 X_i Y_i}{\sum_{i=1}^{n} \sigma^2 X_i^2 + 1}$ $\sigma'^2 = \frac{\sigma^2}{\sum_{i=1}^{n} \sigma^2 X_i^2 + 1}$ is $\mu' = \frac{\sum_{i=1}^{n} \sigma^2 X_i Y_i}{\sum_{i=1}^{n} \sigma^2 X_i^2 + 1}$

(f) $Y_i = W^T X_i + Z_i$ $Z_i \sim N(0, 1)$ $\therefore Y_i \sim N(W^T X_i, 1)$

log likelihood function

$$\sum_{i=1}^{n} \log \left[\frac{1}{\sqrt{2\pi}} e^{-\frac{(Y_i - W^T X_i)^2}{2}} \right]$$

$$W_{MLE} = \underset{W}{\arg\max} \sum_{i=1}^{n} \log \left[\frac{1}{\sqrt{2\pi}} e^{-\frac{(Y_i - W^T X_i)^2}{2}} \right]$$

$$= \underset{W}{\arg\min} \sum_{i=1}^{n} \frac{(Y_i - W^T X_i)^2}{2} + \underbrace{\frac{1}{2}\log 2\pi}_{\text{has no } W}$$

$$= \underset{W}{\arg\min} \sum_{i=1}^{n} (Y_i - W^T X_i)^2$$

$$= \underset{W}{\arg\min} \| Y - W^T X \|^2 = W_{LS}$$

$\therefore$ In this setting, the maximum likelihood estimator for w is the solution to a lease square problem

(g) $Y_i = W^T X_i + Z_i \quad Z_i \sim N(0,1) \quad W_i \sim N(0, \sigma^2)$

$p(w|x,Y) \propto P(X|W)\,P(W)$

$$P(W_i | X, Y) \propto \left(e^{-\frac{\sum_{i=1}^{n} (Y_i - W^T X_i)^2}{2} - \frac{W_i^2}{2\sigma^2}} \right)$$

I didn't manage to solve this problem.

3. Simple Bias - Variance Tradeoff

(a) ① $E\left(\frac{x_1+x_2+\dots+x_n}{n}-\mu\right) = \frac{1}{n}\times n\mu-\mu = 0$

② $E\left(\frac{x_1+x_2+\dots+x_n}{n+1}-\mu\right) = \frac{1}{n+1}\times n\mu-\mu = -\frac{\mu}{n+1}$

③ $E\left(\frac{x_1+x_2+\dots+x_n}{n+n_0}-\mu\right) = \frac{1}{n+n_0}\times n\mu-\mu = -\frac{n_0\mu}{n+n_0}$

④ $E(0-\mu) = -\mu$

(b) $X_1, X_2, \dots, X_n$ are i.i.d

① $Var(\hat{X}) = \frac{1}{n^2}\left[Var(X_1)+\dots+Var(X_n)\right] = \frac{1}{n^2}\times n\sigma^2 = \frac{\sigma^2}{n}$

② $Var(\hat{X}) = \frac{1}{(n+1)^2}\times n\sigma^2 = \frac{n\sigma^2}{(n+1)^2}$

③ $Var(\hat{X}) = \frac{1}{(n+n_0)^2}\times n\sigma^2 = \frac{n\sigma^2}{(n+n_0)^2}$

④ $Var(\hat{X}) = 0$

(c)
$$\begin{aligned} E[(\hat{X}-X')^2] &= Var(\hat{X}-X') + [E(\hat{X}-X')]^2 \\ &= Var(\hat{X}) + Var(X') + \{E[(\hat{X}-\mu)+(\mu-X')]\}^2 \\ &= Var(\hat{X}) + \sigma^2 + [E(\hat{X}-\mu)+E(\mu-X')]^2 \\ &= Var(\hat{X}) + \sigma^2 + [E(\hat{X}-\mu)]^2 \\ &= Var(\hat{X}) + \sigma^2 + [bias(\hat{X})]^2 \end{aligned}$$

$$\begin{aligned} E[(\hat{X}-\mu)^2] &= Var(\hat{X}-\mu) + [E(\hat{X}-\mu)]^2 \\ &= Var(\hat{X}) + [bias(\hat{X})]^2 \end{aligned}$$

Compare them, there is a σ^2 in $E[(\hat{X}-X')^2]$ but not in $E[(\hat{X}-\mu)^2]$

$\hat{X}$ is calculated based on training samples

so $\hat{X}$ will perform worse on fresh sample X', therefore the error will be larger.

(d) ① $= Var(\hat{X}) + bias^2(\hat{X}) = \frac{\sigma^2}{n} + 0^2 = \frac{\sigma^2}{n}$

② $= \frac{n}{(n+1)^2}\sigma^2 + \left(\frac{\mu}{n+1}\right)^2 = \frac{n\sigma^2+\mu^2}{(n+1)^2}$

③ $= \frac{n\sigma^2}{(n+n_0)^2} + \frac{n_0^2\mu^2}{(n+n_0)^2} = \frac{n\sigma^2+n_0^2\mu^2}{(n+n_0)^2}$

④ $= 0+\mu^2 = \mu^2$

(e) ① is when $n_0=0$ ② is when $n_0=1$ ④ is when $n_0=\infty$

(f) As n_0 increases, $|bias|$ increases (since bias ≤ 0 here, actually bias is decreasing), variance decreases.

(g) $n_0 = \alpha n$, $Var(\hat{X}) = \frac{n\sigma^2}{(n+\alpha n)^2}$ $bias(\hat{X}) = -\frac{\alpha n\mu}{n+\alpha n}$

expected total error $= \dfrac{n\sigma^2}{(n+\alpha n)^2} + \dfrac{(\alpha n\mu)^2}{(n+\alpha n)^2}$

$= \dfrac{n\sigma^2 + \alpha^2 n^2\mu^2}{(n+\alpha n)^2}$

$\dfrac{\partial}{\partial\alpha}$ expected total error $= \dfrac{n^2\mu^2 \times 2\alpha(n+\alpha n)^2 - (n\sigma^2+\alpha^2n^2\mu^2)\times 2(n+\alpha n)\times n}{(n+\alpha n)^4}$

$= 0$

$2n^2\mu^2\alpha(n+\alpha n)^2 = 2n(n+\alpha n)(n\sigma^2+\alpha^2n^2\mu^2)$

$n\mu^2(n+\alpha n)\alpha = n\sigma^2 + \alpha^2n^2\mu^2$

$n^2\mu^2\alpha + n^2\mu^2\alpha^2 = n^2\mu^2\alpha^2 + n\sigma^2$

$n^2\mu^2\alpha = n\sigma^2$

$\alpha = \dfrac{n\sigma^2}{n^2\mu^2} = \dfrac{\sigma^2}{n\mu^2}$

(h) When σ is large and μ is small (close to 0)

$\alpha = \dfrac{\sigma^2}{n\mu^2} \to \infty$ (very large)

(i) $X' = X - \mu_0$

$E(X') = E(X-\mu_0) = \mu - \mu_0$

$Var(X') = Var(X-\mu_0) = Var(X) = \sigma^2$

(j) In ridge regression, as λ increases, model bias $\uparrow$, model variance $\downarrow$.

This is very similar to α's influence of $\hat{x}$'s bias and variance.

So in ridge regression, we can use cross validation to select the λ value with the smallest validation error.

4 Robotic Learning of Controls from Demonstrations and Images

(a) The 0th 10th and 20th images in the training set are plotted in the appendix

Their corresponding control vectors are shown in appendix.

The code is in robotic_ridge_code_starter.py under section # 4(a).

(b) The code to do this is under section # 4(b)

When I attempt to do this, it will raise Singular matrix error

This is because $\det(X^TX) = 0$, X^TX is not invertible

(c) For each λ in [0.1, 1.0, 10.0, 100.0, 1000.0] the result is shown in appendix

The code is under section # 4(c)

(d) The result is shown in appendix

The code is under section # 4(d)

(e) The result is shown in appendix

The code is under section # 4(e)

By increasing λ value, the bias will increase because $\lambda \uparrow$ means the model complexity will decrease, therefore bias will increase

By increasing λ value, the variance will decrease because $\lambda \uparrow$, model complexity $\downarrow$, therefore varience $\downarrow$.

(f) The result is shown in appendix

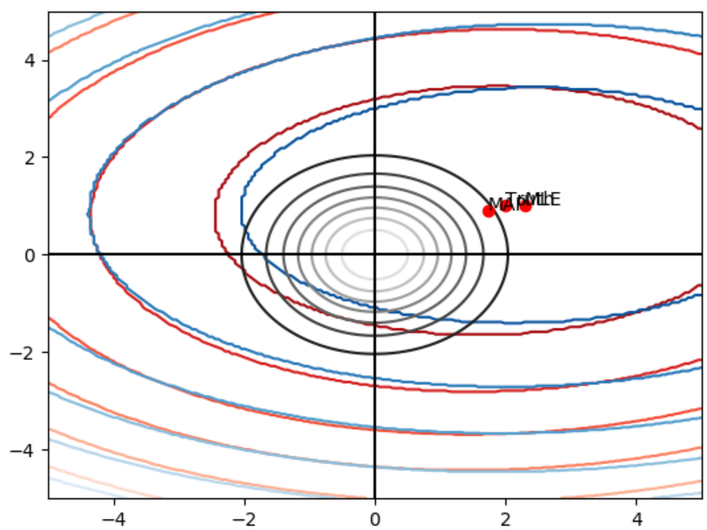
The code is under section # 4(f)

Appendix

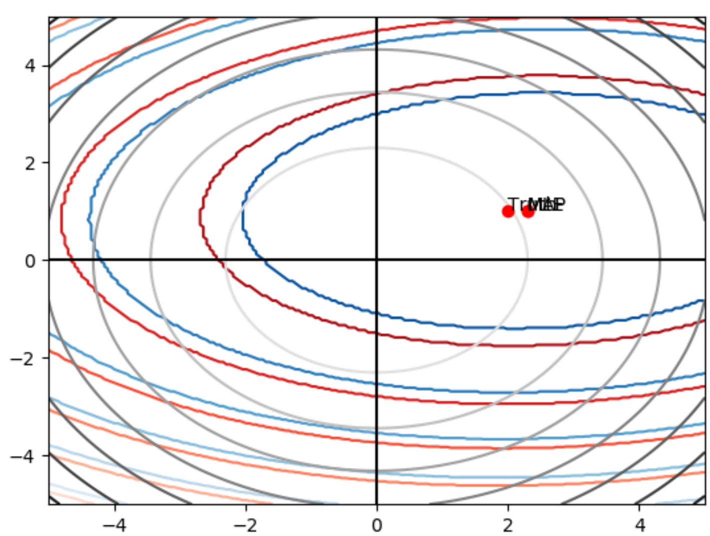
1.

(b)

$\sigma_h = 1$

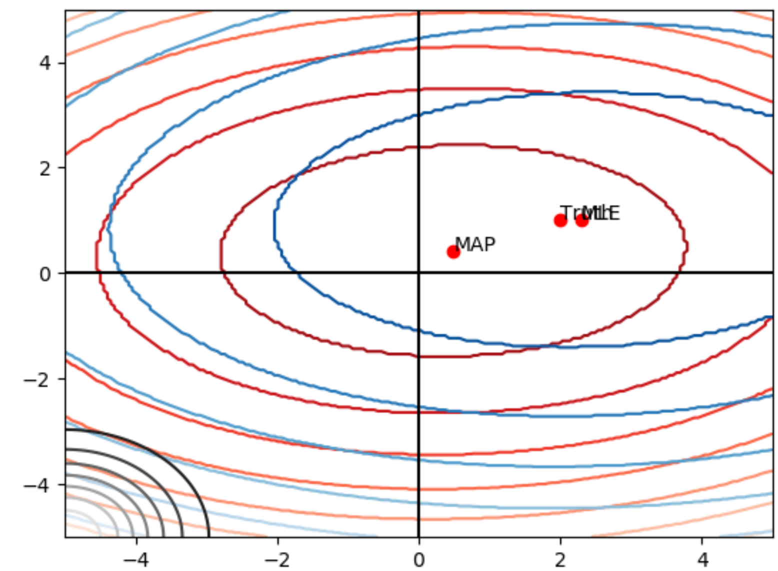


$\sigma_h = 10$

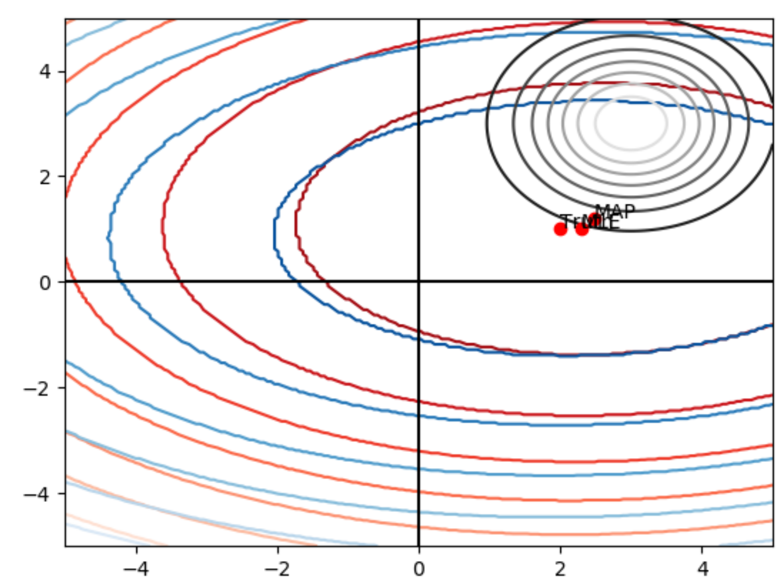


(c)

$\mu_\theta = \begin{pmatrix}-5\\-5\end{pmatrix}$:

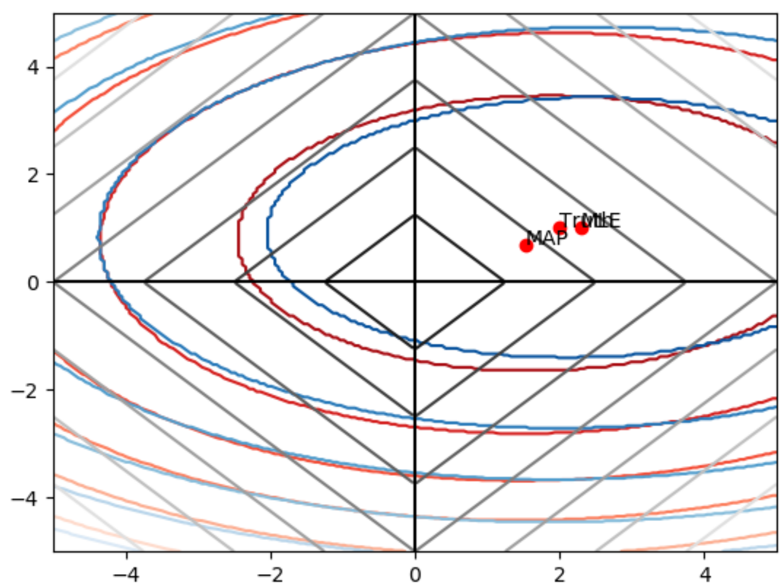


$\mu_\theta = \begin{pmatrix}3\\3\end{pmatrix}$

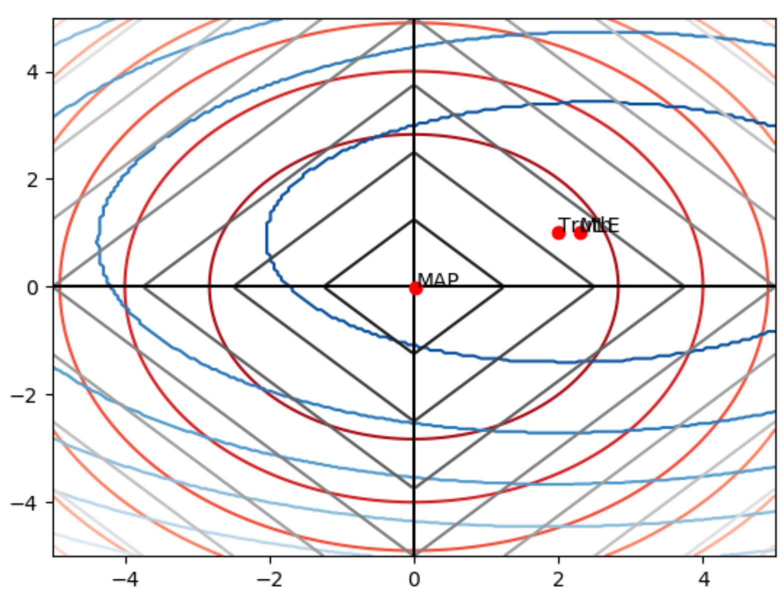


(e)

$P(\theta_i) \sim L(0, 1)$



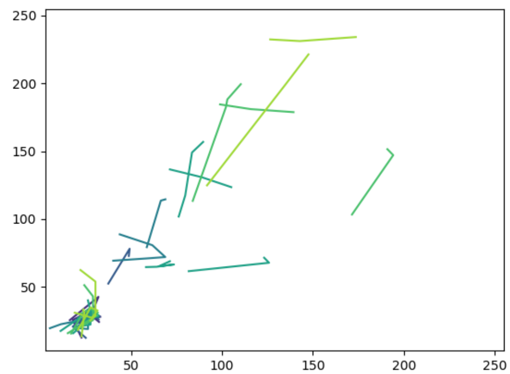
$P(\theta_i) \sim L(0, 0.0001)$



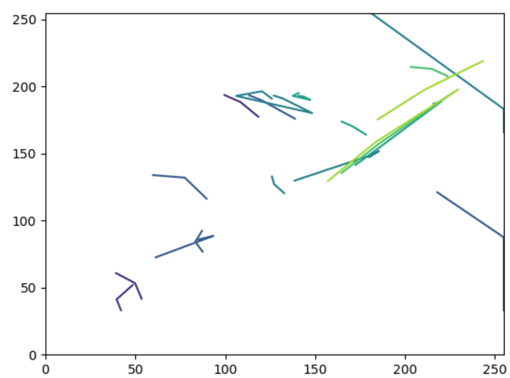
4.

(a)

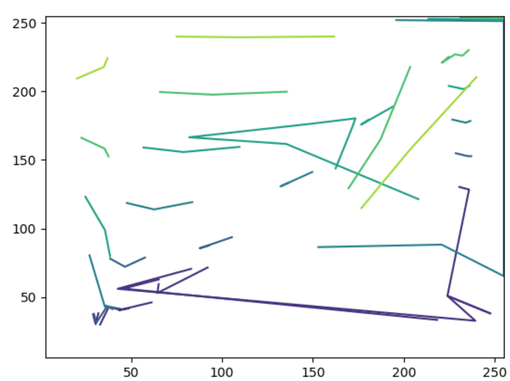
0th image



10th image



20th image



Their corresponding control vectors are:

```
0th image control vector: [ 0. -1.  0.]
10th image control vector: [-1.         -0.45111084 -1.        ]
20th image control vector: [0.         0.         0.37368774]
```

(c)

```
Average squared Euclidien distance for different lambda on training set: [1.256702068937529e-15,
1.25669241758656e-13, 1.2565943984111686e-11, 1.2556154166406199e-09, 1.2459339631725425e-07]
```

(d)

```
Average squared Euclidien distance for different lambda on training set after scale: [3.255747498
915746e-07, 2.910512290768579e-05, 0.0015903814573038663, 0.034773122042375766, 0.254402961467970
3]
```

(e)

```
Average squared Euclidien distance for different lambda on test set: [2.0792664172815383e-16, 2.0
7926090800701e-14, 2.0792067442486828e-12, 2.0786652769260328e-10, 2.073269975904201e-08]
Average squared Euclidien distance for different lambda on test set after scale: [5.4858500599334
95e-08, 5.2638881496879074e-06, 0.0003806734444981973, 0.011336004792241953, 0.13159252242709102]
```

(f)

```
Condition number of training data without standardization: 52711693.12866252
Condition number of training data with standardization: 444.7259317110044
Condition number of test data without standardization: 28927142.279349737
Condition number of test data with standardization: 39339.65804431199
```

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

'''
Hyperparameter Configurations
'''
# Prior weights are assumed to be i.i.d, so they all have the same prior_std
prior_mean = [0, 0]
prior_std = 0.0001
noise_mean = 0.0
noise_std = 1.0
# Can either by Gaussian or Laplacian Prior
options = ["Gaussian", "Laplacian"]
option = options[1]

# Plot and Randomness Configurations
fig, ax = plt.subplots()
ax.axhline(y=0, color='k')
ax.axvline(x=0, color='k')
lower_range = -5
upper_range = 5
# DO NOT Change the Seed and Num Points for Easy and Consistent Grading
np.random.seed(7)
num_data_points = 10


'''
Point Estimate Functions
'''
def MLE_Point_Est(X, Y):
    '''
    Input: X - nx2; Y - nx1
    Return: w - 2x1
    Returns the OLS Solution
    '''
    return np.linalg.solve(X.T@X, X.T@Y)

def Gaussian_MAP_Point_Est(X, Y, noise_std, prior_std, prior_mean):
    '''
    Input: X - nx2; Y - nx1, Prior and Noise Staistics (All scalars)
    Return: w - 2x1
    Returns the Ridge Regression Solution
    '''
    lambda_ = (noise_std/prior_std)**2
    return np.linalg.solve(X.T@X+lambda_*np.identity(X.shape[1]),
        X.T@Y+lambda_*np.array(prior_mean).reshape((2,1)))

def LASSO_Point_Est(w1, w2, map_contour):
    '''
    This part of the code is filled out. Since in most cases there is no
     closed form for the LASSO equation, we directly find the center
     through a linear search.
    DON'T DO THIS.
    '''
    min_candidates = np.where(map_contour == np.amin(map_contour))
```

```
    processed_points = list(zip(min_candidates[0], min_candidates[1]))

    x_ind = processed_points[0][0]
    y_ind = processed_points[0][1]
    return np.array([w1[x_ind][y_ind], w2[x_ind][y_ind]]).reshape((-1,1))

'''
Functions for Contour Plotting
'''

def Gaussian_Prior(w1, w2, prior_mean):
    '''
    Returns the Gaussian Prior or L2 norm squared for weights
    '''
    return
     (1/(2*np.pi*(prior_std**2)))*np.exp(-0.5*((((w1-prior_mean
     [0])/prior_std)**2)+(((w2-prior_mean[1])/prior_std)**2)))

def Laplacian_Prior(w1, w2, prior_mean):
    '''
    Returns the Laplacian Prior for weights
    '''
    return
     ((np.abs(w1-prior_mean[0])+np.abs(w2-prior_mean
     [1]))/prior_std)*(2*(noise_std**2))

def Gaussian_MLE_Contour(w1,w2, X, Y):
    '''
    Returns the MLE estimate value
    '''
    x = np.array([0]*200)
    y = np.array([0]*200)
    Z, z = np.meshgrid(x,y)
    for i in range(200):
        for j in range(200):
            Z[i][j] = np.sum(((Y-X@(np.array([w1[i][j],
                w2[i][j]]).reshape((2,1))))**2)/(2*(noise_std**2))) +
                 num_data_points*np.log(np.sqrt(2*np.pi)*noise_std)
    return Z

def MAP_Contour(w1, w2, X, Y, noise_std, prior_mean, prior_std,
 option="Gaussian"):
    '''
        Hint 1: Use the methods above to compute the MAP. Ideally one line
         of code.
        Hint 2: Use two for loops to go through the (x,y) coordinates for
         w1, w2
    '''
    x = np.array([0]*200)
    y = np.array([0]*200)
    Z, z = np.meshgrid(x,y)
    for i in range(200):
        for j in range(200):
            if option == "Laplacian":
```

```
                Z[i][j] = np.linalg.norm(Y-X@(np.array([w1[i][j],
                 w2[i]
                 [j]])
                 .reshape((2,1))))**2+(2*(noise_std**2)/prior_std)*((np
                 .abs(w1[i][j]-prior_mean[0])+np.abs(w2[i][j]-prior_mean
                 [1])))
            Z[i][j] = np.sum(((Y-X@(np.array([w1[i][j],
             w2[i][j]]).reshape((2,1))))**2)/(2*(noise_std**2))) +
             (np.array([w1[i][j],
                w2[i][j]]) - np.array(prior_mean))@((np.array([w1[i][j],
                 w2[i][j]]) -
                 np.array(prior_mean)).reshape(2,1))/(2*(prior_std**2))
    return Z

'''
Plotting Functions
'''
def plot_point(w, txt):
    plt.plot(w[0][0], w[1][0], 'ro')
    ax.annotate(txt, (w[0][0], w[1][0]))

# Generating Data and Labels with Random Gaussian Noise
w = np.array([[2.0], [1.0]])
Xf= np.random.uniform(-1, 1, num_data_points)
X1 = np.array([1.0]*num_data_points)
X = np.array([Xf, X1]).T
Y = X.dot(w).reshape((-1, 1)) + np.random.normal(noise_mean, noise_std,
 num_data_points).reshape((-1,1))

w1 = np.linspace(lower_range, upper_range, 200)
w2= np.linspace(lower_range, upper_range, 200)
w1, w2 = np.meshgrid(w1, w2)

prior = Gaussian_Prior(w1, w2, prior_mean) if option=="Gaussian" else
 Laplacian_Prior(w1,w2, prior_mean)
mle_contour = Gaussian_MLE_Contour(w1,w2, X, Y)
map_contour = MAP_Contour(w1, w2, X, Y, noise_std, prior_mean, prior_std,
 option)

w_mle = MLE_Point_Est(X,Y)
w_map = Gaussian_MAP_Point_Est(X,Y, noise_std, prior_std, prior_mean) if
 option=="Gaussian" else LASSO_Point_Est(w1, w2, map_contour)

plot_point(w, "Truth")
plot_point(w_mle, "MLE")
plot_point(w_map, "MAP")

# Choose which contours to plot [prior, mle, or map]
plt.contour(w1, w2, map_contour, 7, cmap='Reds_r')
plt.contour(w1, w2, mle_contour, 7, cmap='Blues_r')
plt.contour(w1, w2, prior, 7, cmap='gray')

plt.show()
```

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

class HW3_Sol(object):

    def __init__(self):
        pass

    def load_data(self):
        self.x_train = pickle.load(open('x_train.p','rb'),
         encoding='latin1')
        self.y_train = pickle.load(open('y_train.p','rb'),
         encoding='latin1')
        self.x_test = pickle.load(open('x_test.p','rb'), encoding='latin1')
        self.y_test = pickle.load(open('y_test.p','rb'), encoding='latin1')

if __name__ == '__main__':

    hw3_sol = HW3_Sol()

    hw3_sol.load_data()

    # Your solution goes here
    #4(a)
    hw3_sol.x_train.astype(float)
    hw3_sol.y_train.astype(float)
    hw3_sol.x_test.astype(float)
    hw3_sol.y_test.astype(float)
    #Only choose one to plot each time
    #plot 0th image
    plt.contour(hw3_sol.x_train[0][0], hw3_sol.x_train[0][1],
     hw3_sol.x_train[0][2])
    #plot 10th image
    plt.contour(hw3_sol.x_train[10][0], hw3_sol.x_train[10][1],
     hw3_sol.x_train[10][2])
    #plot 20th image
    plt.contour(hw3_sol.x_train[20][0], hw3_sol.x_train[20][1],
     hw3_sol.x_train[20][2])
    print('0th image control vector:', hw3_sol.y_train[0])
    print('10th image control vector:', hw3_sol.y_train[10])
    print('20th image control vector:', hw3_sol.y_train[20])
    plt.show()

    #4(b)
    n = hw3_sol.x_train.shape[0]
    X = np.zeros((n,2700))
    for i in range(n):
        X[i] = hw3_sol.x_train[i].flatten()
    U = hw3_sol.y_train.reshape((n, 3))
    #PI = np.linalg.solve(X.T@X, X.T@U)

    #4(c)
    LAMBDA = [0.1, 1.0, 10.0, 100.0, 1000.0]
    errors = []
```

```
for i in range(5):
    PI = np.linalg.solve(X.T@X+LAMBDA[i]*np.identity(X.shape[1]), X.T@U)
    residual = X@PI - U
    error = np.zeros((n,1))
    for j in range(n):
        error[j] = residual[j][0]**2 + residual[j][1]**2 +
         residual[j][2]**2
    error = np.mean(error)
    errors += [error]
print('Average squared Euclidien distance for different lambda on
 training set:', errors)

#4(d)
scaled_errors = []
scaled_X = X*2/255-1
for i in range(5):
    PI_scaled =
     np.linalg.solve(scaled_X.T@scaled_X+LAMBDA
     [i]*np.identity(scaled_X.shape[1]), scaled_X.T@U)
    residual_scaled = scaled_X@PI_scaled - U
    scaled_error = np.zeros((n,1))
    for j in range(n):
        scaled_error[j] = residual_scaled[j][0]**2 +
         residual_scaled[j][1]**2 + residual_scaled[j][2]**2
    scaled_error = np.mean(scaled_error)
    scaled_errors += [scaled_error]
print('Average squared Euclidien distance for different lambda on
 training set after scale:', scaled_errors)

#4(e)
n_test = hw3_sol.x_test.shape[0]
X_test = np.zeros((n_test,2700))
for i in range(n_test):
    X_test[i] = hw3_sol.x_test[i].flatten()
U_test = hw3_sol.y_test.reshape((n_test, 3))
X_test_scaled = X_test*2/255-1
errors_test = []
scaled_errors_test = []
for i in range(5):
    PI =
     np.linalg.solve(X_test.T@X_test+LAMBDA[i]*np.identity(X_test.shape
     [1]), X_test.T@U_test)
    PI_scaled =
     np.linalg.solve(X_test_scaled.T@X_test_scaled+LAMBDA
     [i]*np.identity(X_test_scaled.shape[1]), X_test_scaled.T@U_test)
    residual_test = X_test@PI - U_test
    residual_scaled_test = X_test_scaled@PI_scaled - U_test
    error_test = np.zeros((n_test, 1))
    scaled_error_test = np.zeros((n_test, 1))
    for j in range(n_test):
        error_test[j] = residual_test[j][0]**2 + residual_test[j][1]**2
         + residual_test[j][2]**2
        scaled_error_test[j] = residual_scaled_test[j][0]**2 +
         residual_scaled_test[j][1]**2 + residual_scaled_test[j][2]**2
    error_test = np.mean(error_test)
```

```
    scaled_error_test = np.mean(scaled_error_test)
    errors_test += [error_test]
    scaled_errors_test += [scaled_error_test]
print('Average squared Euclidien distance for different lambda on test
 set:', errors_test)
print('Average squared Euclidien distance for different lambda on test
 set after scale:', scaled_errors_test)

#4(f)
u_train, s_train, v_train = np.linalg.svd(X.T@X +
 100*np.identity(X.shape[1]))
u_train_scaled, s_train_scaled, v_train_scaled =
 np.linalg.svd(scaled_X.T@scaled_X + 100*np.identity(scaled_X.shape[1]))
u_test, s_test, v_test = np.linalg.svd(X_test.T@X_test +
 100*np.identity(X_test.shape[1]))
u_test_scaled, s_test_scaled, v_test_scaled =
 np.linalg.svd(X_test_scaled.T@X_test_scaled +
 np.identity(X_test_scaled.shape[1]))
k_train = np.amax(s_train) / np.amin(s_train_scaled)
k_train_scaled = np.amax(s_train_scaled) / np.amin(s_train_scaled)
k_test = np.amax(s_test) / np.amin(s_test)
k_test_scaled = np.amax(s_test_scaled) / np.amin(s_test_scaled)
print('Condition number of training data without standardization:',
 k_train)
print('Condition number of training data with standardization:',
 k_train_scaled)
print('Condition number of test data without standardization:', k_test)
print('Condition number of test data with standardization:',
 k_test_scaled)
```