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) Since given X, $XW_1 + W_0$ is const, $z \sim N(0,1)$,

$$P(Y|X) = N(XW_1 + W_0, 1)$$

(b) $$\mathcal{L} = \prod_i P(Y_i|X_i) = \prod_i \frac{1}{\sqrt{2\pi}} e^{-\frac{(Y_i - X_iW_1 - W_0)^2}{2}}$$

$$\ln L = -\sum_i \frac{(Y_i - X_iW_1 - W_0)^2}{2} + C$$

$$\hat{W_1}, \hat{W_0} = \underset{W_1, W_0}{\operatorname{argmax}} (Y_i - X_iW_1 - W_0)^2$$

(c) Similar as (a), $P(Y|X) = XW + U[-0.5, 0.5]$

$$= U[-0.5 + XW,\ 0.5 + XW]$$

$$= \begin{cases} 1, & \text{for } -0.5 + XW \le y < 0.5 + XW \\ 0, & \text{otherwise} \end{cases}$$

(d) $$P(Y_i|X_i) = \begin{cases} 1, & \text{for } \frac{y_i - 0.5}{x_i} \le w \le \frac{y_i + 0.5}{x_i} \\ 0, & \text{otherwise.} \end{cases}$$

So $$\mathcal{L} = \prod_i P(Y_i|X_i) = \begin{cases} 1, & \max\left\{\frac{y_i - 0.5}{x_i}\right\} \le w \le \min\left\{\frac{y_i + 0.5}{x_i}\right\} \\ 0, & \text{otherwise} \end{cases}$$

(e) As n gets larger, the estimation of W becomes better.

(f) $$P(W|\{x_i, y_i\}) \propto \prod_i P(x_i, y_i|W) \cdot P(W)$$

$$= \prod_i \left(\frac{1}{\sqrt{2\pi}} e^{-\frac{(Y_i - X_iW)^2}{2}}\right) \cdot \frac{1}{\sqrt{2\pi}} e^{-\frac{W^2}{2\sigma^2}}$$

$$\ln P(W|\{x_i, y_i\}) = \sum_i \left(\frac{-(Y_i - X_iW)^2}{2}\right) - \frac{W^2}{2\sigma^2}$$

$$= -\frac{1}{2}\left(\sum_i (Y_i - X_iW)^2 + \frac{W^2}{\sigma^2}\right)$$

$$= -\frac{1}{2}\left[\left(\sum_i X_i^2 + \frac{1}{\sigma^2}\right)W^2 - 2\sum X_iY_iW + \sum Y_i^2\right]$$

$$= -\frac{1}{2}\left[\sqrt{\sum_i X_i^2 + \frac{1}{\sigma^2}}\, W - \sum X_iY_i \Big/ \sqrt{\sum X_i^2 + \frac{1}{\sigma^2}}\right]^2 + \sum Y_i^2 - \frac{(\sum X_iY_i)^2}{\sum X_i^2 + \frac{1}{\sigma^2}}$$

$$\ln P(w \mid \{x_i, y_i\}) = -\frac{1}{2}\left[\left(w - \frac{\sum x_i y_i}{\sum x_i^2 + \frac{1}{\sigma^2}}\right)^2 \cdot \left(\sum x_i^2 + \frac{1}{\sigma^2}\right)\right] + C$$

$$P(w \mid \{x_i, y_i\}) = C\, e^{-\frac{\left(w - \frac{\sum x_i y_i}{\sum x_i^2 + \frac{1}{\sigma^2}}\right)^2}{2\frac{1}{\left(\sum x_i^2 + \frac{1}{\sigma^2}\right)}}}$$

$$\hat{w} = \frac{\sum x_i y_i}{\sum x_i^2 + \frac{1}{\sigma^2}}$$

(g) $\quad y_i \sim N(w^T x_i, 1)$

$$P(y_i) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(y_i - w^T x_i)^2}{2}}$$

$$P(\{x_i, y_i\} \mid w) = \prod_i P(y_i \mid x_i)$$

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

$$\ln P(\{x_i, y_i\} \mid w) = \sum_i -\frac{1}{2}(y_i - w^T x_i)^2 + C$$

$$\max_w \ln P(\{x_i, y_i\} \mid w) \iff \min \{\|y_i - w^T x_i\|^2$$

(h) $\quad P(\{\vec{x}_i, y_i\} \mid \vec{w}) = \prod_i e^{-\frac{(y_i - \vec{w}^T \vec{x}_i)^2}{2}}$

$$P(\vec{w} \mid \{\vec{x}_i, y_i\}) = P(\{\vec{x}_i, y_i\} \mid \vec{w}) \cdot P(\vec{w})$$

$$= \prod_i e^{-\frac{(y_i - \vec{w}^T \vec{x}_i)^2}{2}} \cdot \prod_j e^{-\frac{w_j^2}{2\sigma^2}}$$

$$\ln P(\vec{w} \mid \{\vec{x}_i, y_i\}) = -\frac{1}{2}\left[\sum_i (y_i - \vec{w}^T \vec{x}_i)^2 + \sum_j \frac{w_j^2}{\sigma^2}\right]$$

$$\max_{\vec{w}} \ln P(\vec{w} \mid \{\vec{x}_i, y_i\}) \iff \min_{\vec{w}} \left(\|Y - Xw\|^2 + \frac{1}{\sigma^2}\|w\|^2\right)$$

$$\ln P(\vec{w} \mid \{\vec{x}_i, y_i\}) = -\frac{1}{2}\left[\|Y - Xw\|^2 + \frac{1}{\sigma^2}\|w\|^2\right]$$

$$= -\frac{1}{2}\left[(Y - Xw)^T(Y - Xw) + \frac{1}{\sigma^2} w^T w\right]$$

$$= -\frac{1}{2}\left[Y^TY - Y^TXw - w^TX^TY + w^TX^TXw + \frac{1}{\sigma^2} w^Tw\right]$$

$$= -\frac{1}{2}\left[w^T(X^TX + \frac{1}{\sigma^2} I)w - 2Y^TXw + Y^TY\right]$$

To write $\ln P(\vec{w} \mid \{\vec{x}_i, y_i\})$ into Gaussian distribution,

$$\ln P(\vec{w} \mid \{\vec{x}_i, y_i\}) = -\frac{1}{2}\left[(w-\mu)^T\Sigma^{-1}(w-\mu)\right] + C$$

$$= -\frac{1}{2}\left[w^T\Sigma^{-1}w - \mu^T\Sigma^{-1}w - w^T\Sigma^{-1}\mu - \mu^T\Sigma^{-1}\mu\right] + C$$

$$= -\frac{1}{2}\left[w^T\Sigma^{-1}w - 2\mu^T\Sigma^{-1}w\right] + C$$

Compare with $-\frac{1}{2}\left[w^T(X^TX + \frac{1}{\sigma^2} I)w - 2Y^TXw\right] + C$.

We know: $\Sigma^{-1} = X^TX + \frac{1}{\sigma^2} I$.

$$\mu^T\Sigma^{-1} = Y^TX \Rightarrow \mu = (Y^TX\Sigma)^T$$

$$= \Sigma^TX^TY$$

$$= (X^TX + \frac{1}{\sigma^2} I)^{-1}X^TY.$$

(i) When n increases, w are constrained to a smaller region, meaning better estimation.

When σ increases, the $\|w\|^2$ has less weight, but w are ingeneral constrained better. But when σ is too large, when n is small, w is not very well constrained.

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

sample_size = [5,25,125,625]
plt.figure(figsize=[12, 10])
for k in range(4):
    n = sample_size[k]

    # generate data
    # np.linspace, np.random.normal and np.random.uniform might be useful functions
    X = np.linspace(1,100,n)
    Z = np.random.uniform(-0.5, 0.5, n)
    w_real = 3.2
    Y = X*w_real + Z

    W = np.linspace(3.18,3.22,10000)
    N = len(W)
    likelihood = np.ones(N) # likelihood as a function of w

    w_min = np.max((Y-0.5)/X)
    w_max = np.min((Y+0.5)/X)
    for i1 in range(N):
        # compute likelihood
        if W[i1]>w_min and W[i1]<w_max:
            likelihood[i1] = 1
        else:
            likelihood[i1] = 0


    likelihood /= sum(likelihood) # normalize the likelihood

    plt.figure(figsize=(15,10))
    # plotting likelihood for different n
    plt.plot(W, likelihood)
    plt.xlabel('w', fontsize=10)
    plt.title(['n=' + str(n)], fontsize=14)

plt.show()
```

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

sample_size = [5,25,125,625]
sigma = 100
print('For sigma=%f' %sigma)

w_real = [1,2]
print('w_real is :')
print(w_real)

for k in range(4):
    n = sample_size[k]

    # generate data
    # np.linspace, np.random.normal and np.random.uniform might be useful functions
    X = np.random.rand(n,2)
    Z = np.random.normal(0,1,n)
    Y = X @ w_real + Z


    # compute likelihood
    N = 1001
    W1s = np.linspace(0, 4,N)
    W2s = np.linspace(0, 4,N)
    likelihood = np.zeros([N,N]) # likelihood as a function of w_1 and w_0

    for i1 in range(N):
        w_1 = W1s[i1]
        for i2 in range(N):
            w_2 = W2s[i2]
            # compute the likelihood here
            likelihood[i1][i2] += (np.linalg.norm(Y - X @ [w_1, w_2]))**2
            likelihood[i1][i2] += (np.linalg.norm([w_1, w_2]))**2/sigma**2
            likelihood[i1][i2] /= -2

    # plotting the likelihood
    plt.figure()
    # for 2D likelihood using imshow
    plt.imshow(np.exp(likelihood.T), cmap='hot', aspect='auto',extent=[0,4,0,4])
    plt.xlabel('w0')
    plt.ylabel('w1')
    plt.show()
    plt.savefig('simga_%d_n_%d.png' %(sigma, n), format='png')
    print(n)
```

3. (a) 1. $E[\hat{x}-\mu] = E[\frac{\sum x_i}{n} - \mu] = E(\frac{\sum x_i}{n}) - \mu = \frac{1}{n} \cdot n\mu - \mu = 0$

2. $E[\hat{x}-\mu] = \frac{1}{n+1} E[\sum x_i] - \mu = \frac{n}{n+1}\mu - \mu = -\frac{1}{n+1}\mu$.

3. $E[\hat{x}-\mu] = \frac{1}{n+n_0} E[\sum x_i] - \mu = \frac{n}{n+n_0}\mu - \mu = -\frac{n_0}{n+n_0}\mu$

4. $E[\hat{x}-\mu] = 0 - \mu = -\mu$.

(b) 1. $Var[\hat{x}] = Var[\frac{x_1+\cdots x_n}{n}] = \frac{1}{n^2}(Var[x_1] + \cdots + Var[x_n]) = \frac{n\cdot\sigma^2}{n^2} = \frac{\sigma^2}{n}$

2. $Var[\hat{x}] = \frac{n\sigma^2}{(n+1)^2} = \frac{n}{(n+1)^2}\sigma^2$

3. $Var[\hat{x}] = \frac{n\sigma^2}{(n+n_0)^2} = \frac{n}{(n+n_0)^2}\sigma^2$

4. $Var[\hat{x}] = 0$

(c) $E[(\hat{x}-x')^2] = Var[\hat{x}-x'] + (E[\hat{x}-x'])^2$

$= Var[\hat{x}] + Var[x'] + (E[\hat{x}] - E[x'])^2$

$= Var[\hat{x}] + \sigma^2 + (E[\hat{x}] - \mu)^2$

$= Var[\hat{x}] + E[\hat{x}-\mu]^2 + \sigma^2$

$E[(\hat{x}-\mu)^2] = Var[\hat{x}-\mu] + (E[\hat{x}-\mu])^2$

$= Var[\hat{x}] - 0 + (E[\hat{x}] - \mu)^2$

$= Var[\hat{x}] + E[\hat{x}-\mu]^2$

$\therefore E[(\hat{x}-x')^2] = E[(\hat{x}-\mu)^2] + \sigma^2$, because x' have variance σ^2, so this will introduce another variance of σ^2.

(d) 1. $E[(\hat{x}-x')^2] = \frac{\sigma^2}{n} + 0^2 + \sigma^2 = \frac{n+1}{n}\sigma^2$

$E[(\hat{x}-\mu)^2] = \frac{\sigma^2}{n} + 0^2 = \frac{\sigma^2}{n}$

2. $E[(\hat{x}-x')^2] = \frac{n}{(n+1)^2}\sigma^2 + \frac{\mu^2}{(n+1)^2} + \sigma^2 = \frac{n+(n+1)^2}{(n+1)^2}\sigma^2 + \frac{1}{(n+1)^2}\mu^2$

$E[(\hat{x}-\mu)^2] = \frac{n}{(n+1)^2}\sigma^2 + \frac{1}{(n+1)^2}\mu^2$

3. $E[(\hat{x}-x')^2] = \frac{n}{(n+n_0)^2}\sigma^2 + \frac{n_0^2}{(n+n_0)^2}\mu^2 + \sigma^2 = \frac{n+(n+n_0)^2}{(n+n_0)^2}\sigma^2 + \frac{n_0^2}{(n+n_0)^2}\mu^2$

$E[(\hat{x}-\mu)^2] = \frac{n}{(n+n_0)^2}\sigma^2 + \frac{n_0^2}{(n+n_0)^2}\mu^2$

4. $E[(\hat{x}-x')^2] = 0 + \mu^2 + \sigma^2 = \mu^2 + \sigma^2$

$E[(\hat{x}-\mu)^2] = \mu^2$

(e) 1. $n_0 = 0 \Rightarrow E[(\hat{x}-x')^2] = \frac{n+n^2}{n^2}\sigma^2 = \frac{1+n}{n}\sigma^2$

$E[(\hat{x}-\mu)^2] = \frac{n}{n^2}\sigma^2 = \frac{1}{n}\sigma^2$

2. $n_0 = 1 \Rightarrow E[(\hat{x}-x')^2] = \frac{n+(n+1)^2}{(n+1)^2}\sigma^2 + \frac{1}{(n+1)^2}\mu^2$

$E[(\hat{x}-\mu)^2] = \frac{n}{(n+1)^2}\sigma^2 + \frac{1}{(n+1)^2}\mu^2$

4. $n_0 \to \infty \Rightarrow E[(\hat{x}-x')^2] = \sigma^2 + \mu^2$

$E[(\hat{x}-\mu)^2] = \mu^2$

(f) bias $E[\hat{x}-\mu] = -\frac{n_0}{n+n_0}\mu$, when $n_0 \uparrow$, $(1-\frac{n}{n+n_0})\mu \uparrow$, so bias $\uparrow$.

Variance $\mathrm{Var}[\hat{x}] = \frac{n}{(n+n_0)^2}\sigma^2$, when $n_0 \uparrow$, $\frac{n}{(n+n_0)^2}\sigma^2 \downarrow$, so variance $\downarrow$.

(g) $E[(\hat{x}-\mu)^2] = \mathrm{Var}[\hat{x}] + E[\hat{x}-\mu]^2$

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

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

$\frac{\partial E[(\hat{x}-\mu)^2]}{\partial \alpha} = \frac{\sigma^2}{n}\cdot(-2)\cdot(1+\alpha)^{-3} + \frac{2\alpha(1+\alpha)^2 - \alpha^2\cdot 2(1+\alpha)}{(1+\alpha)^4}\mu^2 = 0 \Rightarrow \alpha = \frac{\sigma^2}{n\mu^2}$

(h) If $\mu \ll 0$, $\sigma > 0$, then $\alpha = \frac{\sigma^2}{n\mu^2} \rightarrow$ very large

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

$E[X'] = E[X] - E[\mu_0] = \mu - \mu_0$

$Var[X'] = Var[X] + Var[\mu_0] = \sigma^2$

(j) When $\alpha \uparrow$, bias $\uparrow$, variance $\downarrow$, so larger α will constrain model to a smaller region, therefore creates a smaller variance, but larger α will also make larger bias. From this, we can see α has the same effect of λ, larger α means larger λ.

$\alpha = \frac{\sigma^2}{n\mu^2}$, when we have large sample, (large n), we need smaller λ.

When our sample has large variance, we want higher λ.

When our sample has large μ, we want smaller λ.

4. (a) · $w^* = (X^TX)^{-1}X^Ty^*$

$\hat{w} = (X^TX)^{-1}X^Ty$

$E[\hat{w}] = (X^TX)^{-1}X^T E[y]$

$= (X^TX)^{-1}X^T E[y^* + z]$

$= (X^TX)^{-1}X^T (E[y^*] + E[z])$ ($E[z] \to 0$)

$= (X^TX)^{-1}X^Ty^*$

$= E[w^*]$

$E[w^*] = (X^TX)^{-1}X^Ty^*$

· $E[\|y^* - X\hat{w}\|^2] = E[(y^* - X\hat{w})^T(y^* - X\hat{w})]$

$= E[\|y\|^2 - 2y^{*T}X\hat{w} + \|X\hat{w}\|^2]$

$= \|y\|^2 - 2y^{*T}E[X\hat{w}] + E[\|X\hat{w}\|^2]$

$\|y^* - E[X\hat{w}]\|^2 = (y^* - E[X\hat{w}])^T(y^* - E[X\hat{w}])$

$= \|y\|^2 - 2y^{*T}E[X\hat{w}] + \|E[X\hat{w}]\|^2$

$E[\|X\hat{w} - E[X\hat{w}]\|^2] = E[(X\hat{w} - E[X\hat{w}])^T(X\hat{w} - E[X\hat{w}])]$

$= E[\|X\hat{w}\|^2 - 2\hat{w}^TX^TE[X\hat{w}] + \|E[X\hat{w}]\|^2)$

$= E[\|X\hat{w}\|^2] - 2E[(X\hat{w})^T]E[X\hat{w}] + \|E[X\hat{w}]\|^2$

$= E[\|X\hat{w}\|^2] - 2E[X\hat{w}]^TE[X\hat{w}] + \|E[X\hat{w}]\|^2$

$= E[\|X\hat{w}\|^2] - 2\|E[X\hat{w}]\|^2 + \|E[X\hat{w}]\|^2$

$= E[\|X\hat{w}\|^2] - \|E[X\hat{w}]\|^2$

$\Rightarrow E[\|y^* - X\hat{w}\|^2] = \|y^* - E[X\hat{w}]\|^2 + E[\|X\hat{w} - E[X\hat{w}]\|^2]$

(b) $\hat{w} = w^* + (X^TX)^{-1}X^T z$

$z_i \sim N(0, \sigma^2) \Rightarrow z_i \propto e^{-\frac{z_i^2}{2\sigma^2}} \Rightarrow z \propto e^{-\frac{\sum z_i^2}{2\sigma^2}} \propto e^{-\frac{z^Tz}{2\sigma^2}}$

$\Rightarrow z \sim N(0, \sigma^2 I)$

$(X^TX)^{-1}X^Tz \sim N(0, (X^TX)^{-1}X^T \sigma^2 X (X^TX)^{-1})$

$= N(0, \sigma^2 (X^TX)^{-1} \cancel{X^TX (X^TX)^{-1}})$

$= N(0, \sigma^2 (X^TX)^{-1})$

$\Rightarrow \hat{w} \sim N(w^*, \sigma^2 (X^TX)^{-1})$

(c) $\frac{1}{n} E[\|X\hat{w} - Xw^*\|^2] = \frac{1}{n} E[\|X\hat{w} - E[X\hat{w}]\|^2] = \frac{1}{n} Var[X\hat{w}]$

$\hat{w} \sim N(w^*, \sigma^2 (X^TX)^{-1}) \Rightarrow X\hat{w} \sim N(Xw^*, \sigma^2 X (X^TX)^{-1} X^T)$

$Var[X\hat{w}] = tr(\sigma^2 X (X^TX)^{-1} X^T)$

$= \sigma^2 tr((X^TX)^{-1} X^TX)$

$= \sigma^2 \cdot d$

$\Rightarrow \frac{1}{n} E[\|X\hat{w} - Xw^*\|^2] = \frac{1}{n} Var[X\hat{w}] = \frac{\sigma^2 d}{n}$

(d) $\cdot w^* = \underset{w}{\operatorname{argmin}} \|y^* - P_D w\|$

$\Rightarrow w^* = [w_0, w_1, \cdots 0]^T$

$\cdot y^* = Xw^* = 0 \Rightarrow$ bias $\|y^* - Xw^*\|^2 = 0$

$\cdot$ From (c) we know, $\frac{1}{n} E[\|X\hat{w} - y^*\|^2] = \frac{\sigma^2 d}{n} \leq \varepsilon \Rightarrow n \geq \sigma^2 d \varepsilon$

So when we increase model complexity, $d\uparrow$, $n\uparrow$, so we require a larger number of samples.

(e) see code in attachment.

From the plot we can see: the average error $\propto D$, $\propto \frac{1}{\text{number of samples}}$.

(f) $E[\|y^* - X\hat{w}\|^2] = \|y^* - Xw^*\|^2 + E[\|X\hat{w} - E[X\hat{w}]\|^2]$

(1) bias $= \|y^* - Xw^*\|^2$

$= \sum_i \left(e^{a_i} - \left(1 + a_i + \frac{a_i^2}{2} + \cdots + \frac{a_i^D}{D!}\right)\right)^2$

$\leq \sum_i \left(\frac{e^3 \cdot 4^{D+1}}{(D+1)!}\right)^2$

$= n\left(\frac{e^3 \cdot 4^{D+1}}{(D+1)!}\right)^2$ so when $D\uparrow$, bias $\downarrow$.

(2) variance $= \sigma^2 \cdot D$ so when $D\uparrow$, variance $\uparrow$

$\Rightarrow$ prediction error $= n\frac{e^3 \cdot 4^{D+1}}{(D+1)!} + \sigma^2 D \approx \frac{n \cdot e^3 \cdot 4^{D+1} \cdot e^{D+1}}{\sqrt{2\pi(D+1)}\,(D+1)^{D+1}} + \sigma^2 D$

$= \frac{ne^3}{\sqrt{2\pi(D+1)}}\left(\frac{4e}{D+1}\right)^{D+1} + \sigma^2 D.$

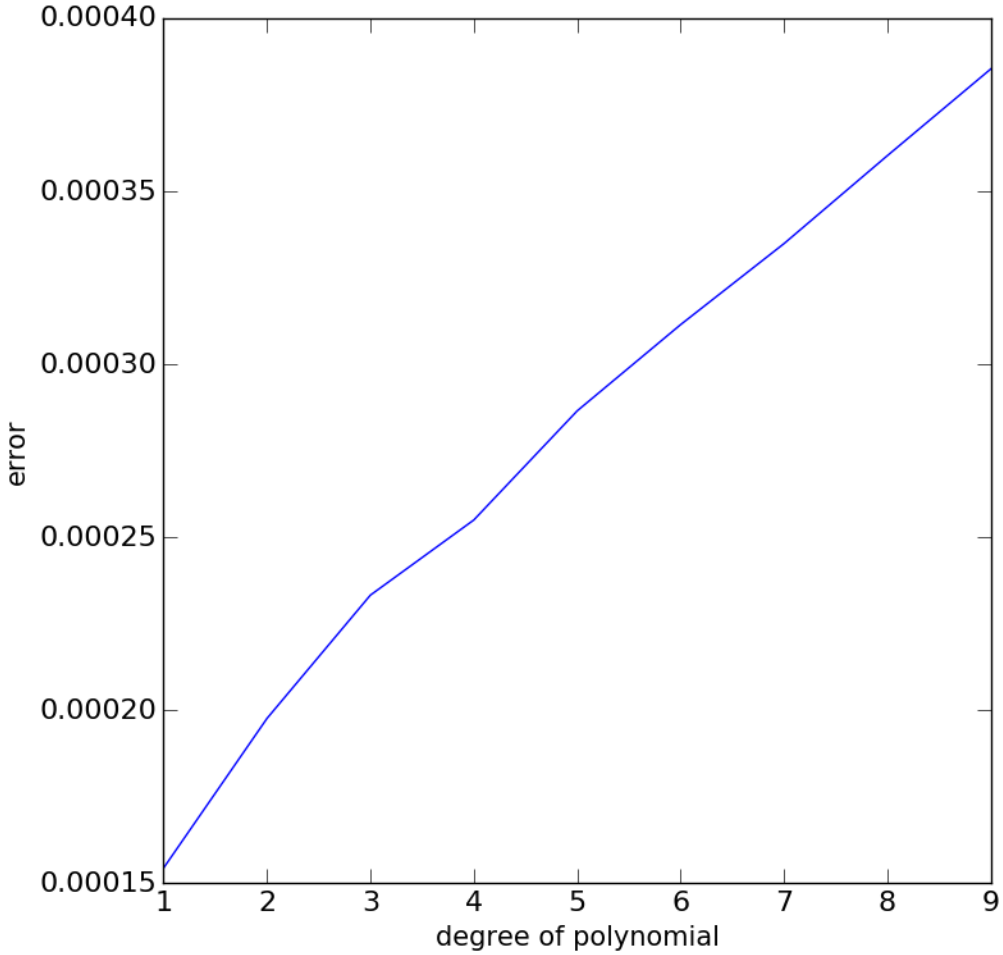
(g) The average error still $\propto \frac{1}{\text{number of samples}}$.

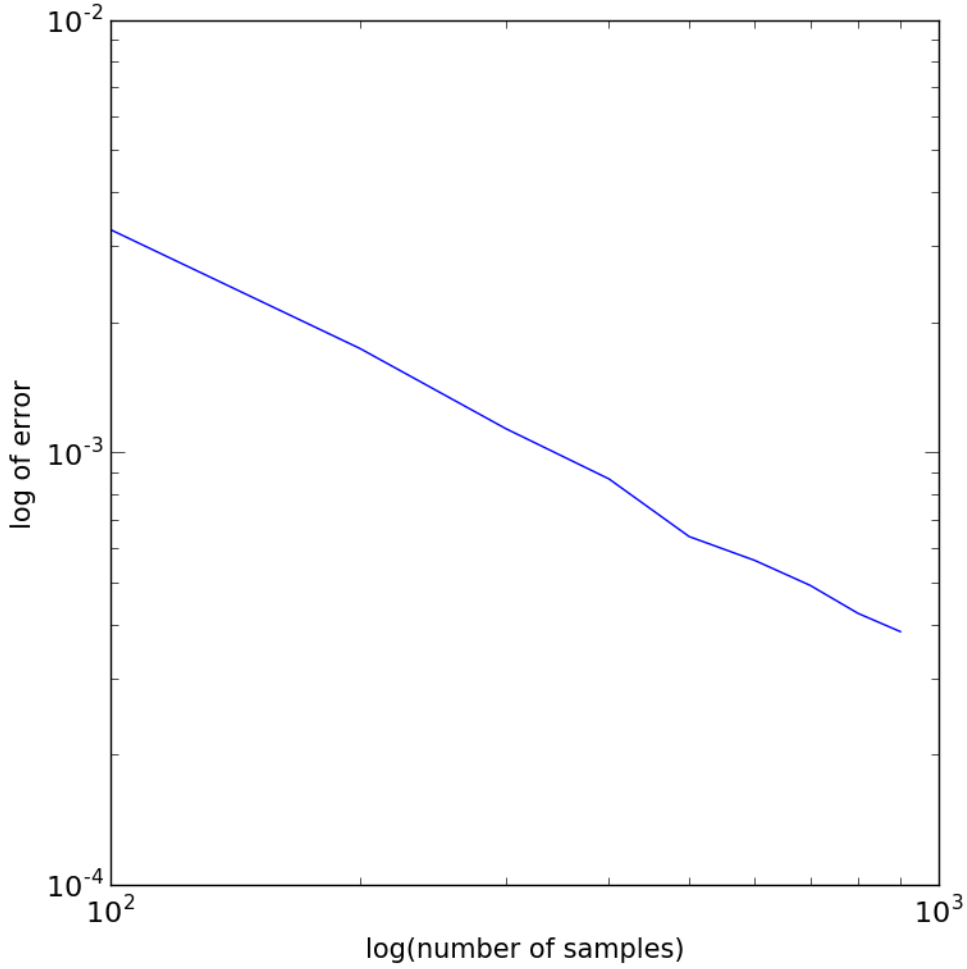
But when $D\uparrow$, average error first decrease, then increase.

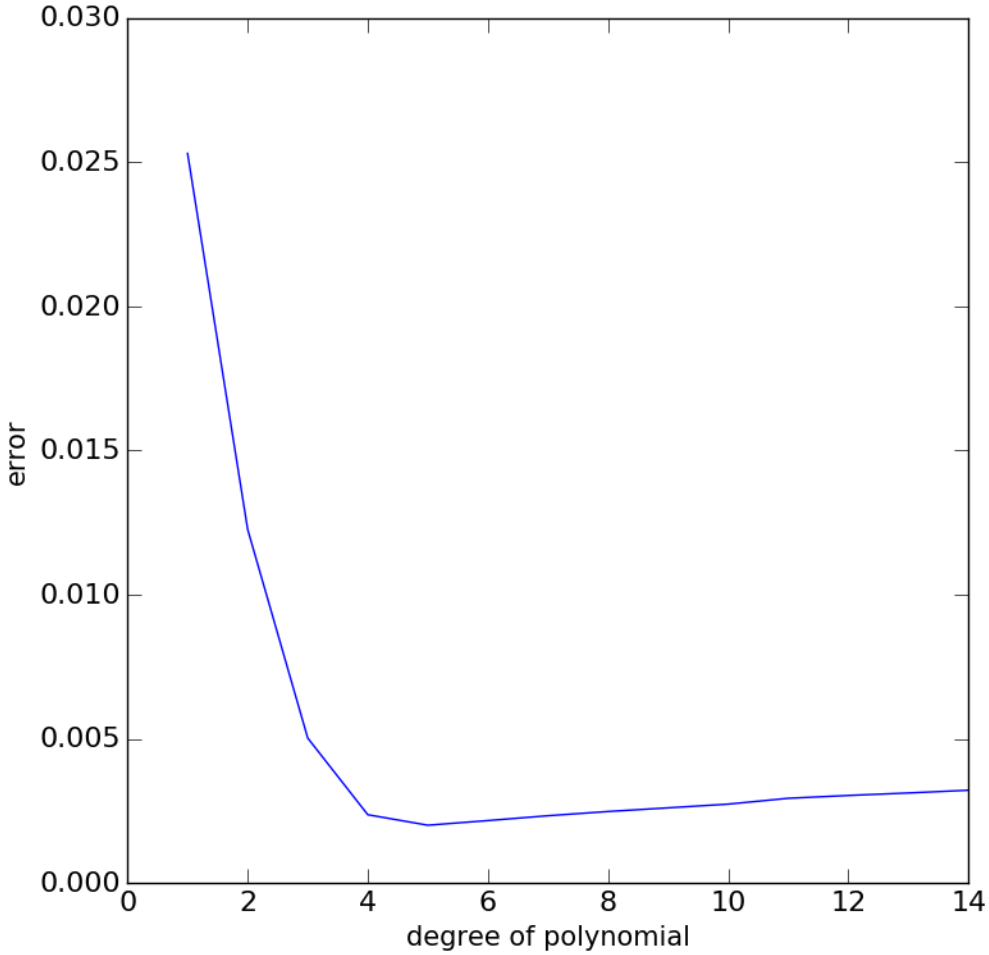
(h) The difference between (e) and (g) is for e, bias $=0$, variance is the only term in error, so when $D\uparrow$, variance $\uparrow$, error $\uparrow$. For (g), bias $\neq 0$, variance still increase with D, but bias $\downarrow$ with D. So in total, error $\downarrow$ first and then $\uparrow$.

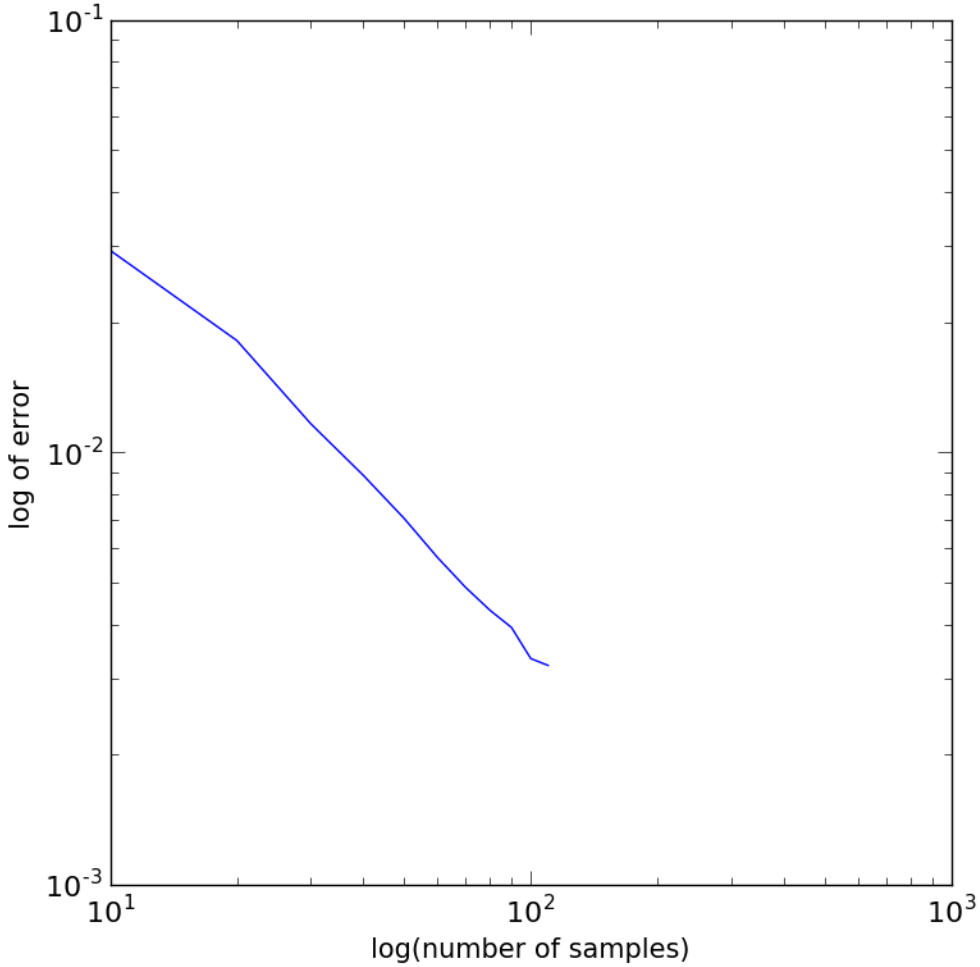
```python
import numpy as np
import matplotlib.pyplot as plt
import pdb

def que_e():
    degs = np.arange(1,10)
    step = 100
    ns = np.arange(100,1000,step)

    # assign problem parameters
    w1 = 1
    w0 =1


    errors = []
    for runs in range(50):
        for i in range(len(ns)):
            error = np.zeros((len(ns), len(degs)))
            n = ns[i]
            # generate data
            # np.random might be useful

            # generate alpha
            alpha = np.linspace(-1,1,n)

            # generate z
            sigma = 1
            z = np.random.normal(loc=0, scale=sigma, size=n)

            # generate y
            y_real = alpha*w1 + w0
            y = y_real + z

            for j in range(len(degs)):
                deg = degs[j]
                # fit data with different models
                # np.polyfit and np.polyval might be useful
                w = np.polyfit(alpha,y,deg)
                error[i][j] = np.linalg.norm(np.polyval(w, alpha) - y_real)/n
            errors.append(error)

    errors = np.mean(errors, axis=0)

    # plotting figures
    # sample code

    plt.figure(figsize=(20,10))
    plt.subplot(121)
    plt.plot(degs,errors[-1])
    plt.xlabel('degree of polynomial')
    plt.ylabel('error')
    plt.subplot(122)
    plt.loglog(ns, errors[:,-1])
    plt.xlabel('log(number of samples)')
    plt.ylabel('log of error')
    plt.tight_layout()
    plt.show()
    plt.savefig('4_e.png', format='png')
    plt.close()

def que_g():
    degs = np.arange(1,15)
    step = 10
    ns = np.arange(10,120,step)

    errors = []
    for runs in range(50):
        for i in range(len(ns)):
            error = np.zeros((len(ns), len(degs)))
            n = ns[i]
            # generate data
            # np.random might be useful
```

```
            # generate alpha
            alpha = np.linspace(-4,3,n)

            # generate z
            sigma = 1
            z = np.random.normal(loc=0, scale=sigma, size=n)

            # generate y
            y_real = np.exp(alpha)
            y = y_real + z

            for j in range(len(degs)):
                deg = degs[j]
                # fit data with different models
                # np.polyfit and np.polyval might be useful
                w = np.polyfit(alpha,y,deg)
                error[i][j] = np.linalg.norm(np.polyval(w, alpha) - y_real)/n
            errors.append(error)

    errors = np.mean(errors, axis=0)

    # plotting figures
    # sample code

    plt.figure(figsize=(20,10))
    plt.subplot(121)
    plt.plot(degs,errors[-1])
    plt.xlabel('degree of polynomial')
    plt.ylabel('error')
    plt.subplot(122)
    plt.loglog(ns, errors[:,-1])
    plt.xlabel('log(number of samples)')
    plt.ylabel('log of error')
    plt.tight_layout()
    plt.show()
    plt.savefig('4_g.png', format='png')
    plt.close()
```


0.00040
0.00035
0.00030
0.00025
0.00020
0.00015
error
1
2
3
4
5
6
7
8
9
degree of polynomial


10^{-2}
10^{-3}
10^{-4}
log of error
10^{2}
10^{3}
log(number of samples)


0.030
0.025
0.020
0.015
0.010
0.005
0.000
error
0
2
4
6
8
10
12
14
degree of polynomial


10^{-1}
10^{-2}
10^{-3}
log of error
10^{1}
10^{2}
10^{3}
log(number of samples)

5. (b) It reports X^tX is a singular matrix.

Because we have 2700 X3 parameters, but only 91X3 equations, so this problem is underdetermined.

(c) larger λ means π is more constrained, so it will cause smaller variance but larger bias.

6. Q: $A \in R^{m \times n}$ $(m \geq n)$, and A has a full rank.

Prove that: $\begin{bmatrix} I & A \\ A^t & 0 \end{bmatrix} \begin{bmatrix} r \\ x \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}$ has a solution X where

X minimizes $\|AX - b\|^2$.

A: $\begin{cases} r + AX = b & ① \\ A^T r = 0 & ② \end{cases}$

A^T ① $\Leftrightarrow$ $A^T r + A^T A X = A^T b$

$A^T r = 0$

$\Rightarrow$ $A^T A X = A^T b$

$X = (A^T A)^{-1} A^T b$

so X minimizes $\|AX - b\|^2$.

```
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').astype
(float)
        self.y_train = pickle.load(open('y_train.p','rb'), encoding='latin1').astype
(float)
        self.x_test = pickle.load(open('x_test.p','rb'), encoding='latin1').astype(f
loat)
        self.y_test = pickle.load(open('y_test.p','rb'), encoding='latin1').astype(f
loat)


if __name__ == '__main__':

    hw3_sol = HW3_Sol()

    hw3_sol.load_data()

    # Your solution goes here

    ###### a ########
    # visulize data
    image0 = hw3_sol.x_train[0]
    control0 = hw3_sol.y_train[0]
    plt.imshow(image0)
    image10 = hw3_sol.x_train[10]
    control10 = hw3_sol.y_train[10]
    plt.imshow(image10)
    image20 = hw3_sol.x_train[20]
    control20 = hw3_sol.y_train[20]
    plt.imshow(image20)

    ###### b ########
    n = len(hw3_sol.x_train)
    X = hw3_sol.x_train.reshape(n, 2700)
    U = hw3_sol.y_train
    #pi = np.linalg.inv(X.T @ X ) @ X.T @ U

    ###### c ########
    lambs = np.array([0.1,  1, 10, 100, 1000])
    train_error = []
    pi_c = []
    for lamb in lambs:
        pi = np.linalg.inv(X.T @ X + lamb * np.identity(2700)) @ X.T @ U
        pi_c.append(pi)
        train_error.append(np.linalg.norm(X @ pi - U)**2/n)
    print('avg training error without standardization')
    print(train_error)

    ###### d ########
    Xs = X/255.*2 -1
    train_error = []
    pi_d = []
    for lamb in lambs:
        pi = np.linalg.inv(Xs.T @ Xs + lamb * np.identity(2700)) @ Xs.T @ U
        pi_d.append(pi)
        train_error.append(np.linalg.norm(Xs @ pi - U)**2/n)
    print('avg training error with standardization')
    print(train_error)

    ###### e ########
    n_test = len(hw3_sol.x_test)
    X_test = hw3_sol.x_test.reshape(n_test, 2700)
    y_test = hw3_sol.y_test
    print('avg test error without standardization')
```

```
    test_error = []
    for i in range(len(lambs)):
        pi = pi_c[i]
        test_error.append(np.linalg.norm(X_test @ pi - y_test)**2/n_test)
    print(test_error)

    print('avg test error with standardization')
    test_error = []
    Xs_test = X_test/255.*2 -1
    for i in range(len(lambs)):
        pi = pi_d[i]
        test_error.append(np.linalg.norm(Xs_test @ pi - y_test)**2/n_test)
    print(test_error)

    ###### e ########
    lamb = 100
    sv = np.linalg.cond(X.T @ X + lamb * np.identity(2700))
    print('condition number without standardization')
    print(sv)
    svs = np.linalg.cond(Xs.T @ Xs + lamb * np.identity(2700))
    print('condition number with standardization')
    print(svs)
```