**Question 1.** Getting Started

**Read through this page carefully.** You may typeset your homework in latex or submit neatly handwritten/scanned solutions. Please start each question on a new page. Delivera

1. Submit a PDF of your writeup, **with an appendix for your code**, to assignment on Gradescope, "¡¡title¿¿ Write-Up". If there are graphs, include those graphs in the
2. If there is code, submit all code needed to reproduce your results, "¡¡title¿¿ Code".
3. If there is a test set, submit your test set evaluation results, "¡¡title¿¿ Test Set".

After you've submitted your homework, watch out for the self-grade form.

(a) 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 work with Weiran Liu. Just do it. |

(b) Please copy the following statement and sign next to it. We just want to make it *extra* clear so that no one inadverdently cheats.

*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.*

| |
|---|
| 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. —Huanjie Sheng |

**Question 2.** Probabilistic Model of Linear Regression Both ordinary least squares and ridge regression have interpretations from a probabilistic standpoint. In particular, assuming a generative model for our data and a particular noise distribution, we will derive least squares and ridge regression as the maximum likelihood and maximum *a-posteriori* parameter estimates, respectively. This problem will walk you through a few steps to do that. (Along with some side digressions to make sure you get a better intuition for ML and MAP estimation.)

(a) Assume that $X$ and $Y$ are both one-dimensional random variables, i.e. $X, Y \in \mathbb{R}$. Assume an affine model between $X$ and $Y$: $Y = Xw_1 + w_0 + Z$, where $w_1, w_0 \in \mathbb{R}$, and $Z \sim N(0,1)$ is a standard normal (Gaussian) random variable. Assume $w_1, w_0$ are fixed parameters (i.e., they are not random). **What is the conditional distribution of $Y$ given $X$?**

The standard Gaussian distribution PDF is:

$$P(Z = z) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(z-\mu_0)^2}{2\sigma^2}}$$

Because $Z = Y - Xw_1 - w_0$ and $Z \sim N(0,1)$, we can substitute it into the above formula

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

(b) Given $n$ points of training data $\{(X_1, Y_1), (X_2, Y_2), \ldots, (X_n, Y_n)\}$ generated in an iid fashion by the probabilistic setting in the previous part, **derive the maximum likelihood estimator for $w_1, w_0$ from this training data.**

$$L(w_1, w_0|\{(X_1, Y_1), (X_2, Y_2), \ldots, (X_n, Y_n)\}) = (\frac{1}{\sqrt{2\pi}})^n e^{-\sum_{i=1}^n \frac{(Y_i - X_i w_1 - w_0)^2}{2}}$$

$$\ln(L(w_1, w_0|\{(X_1, Y_1), (X_2, Y_2), \ldots, (X_n, Y_n)\})) = -n\ln(\sqrt{2\pi}) - \frac{1}{2}\sum_{i=1}^{n}(Y_i - X_i w_1 - w_0)^2$$

Write it in a matrix format:

$$\ln(L(w_1, w_0|\{(X_1, Y_1), (X_2, Y_2), \ldots, (X_n, Y_n)\})) = -n\ln(\sqrt{2\pi}) - \frac{1}{2}\|\hat{Y} - \hat{X}\hat{w}\|_2^2$$

$$\hat{X} = \begin{bmatrix} X_1 & 1 \\ X_2 & 1 \\ \vdots & \vdots \\ X_n & 1 \end{bmatrix} \hat{Y} = \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{bmatrix} \hat{w} = \begin{bmatrix} w_1 \\ w_0 \end{bmatrix},$$

Therefore, we convert our problem into a quadratic optimization problem and we have derived the solution: $\hat{w} = (\hat{X}^\top \hat{X})^{-1} \hat{X}^\top \hat{Y}$. We write it out:

$$\hat{w} = \begin{bmatrix} w_1 \\ w_0 \end{bmatrix} = (\begin{bmatrix} X_1 & X_2 & \ldots & X_n \\ 1 & 1 & \ldots & 1 \end{bmatrix} \begin{bmatrix} X_1 & 1 \\ X_2 & 1 \\ \vdots & \vdots \\ X_n & 1 \end{bmatrix})^{-1} \begin{bmatrix} X_1 & X_2 & \ldots & X_n \\ 1 & 1 & \ldots & 1 \end{bmatrix} \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{bmatrix}$$

$$\hat{w} = \begin{bmatrix} w_1 \\ w_0 \end{bmatrix} = (\begin{bmatrix} \sum_{i=1}^n X_i^2 & \sum_{i=1}^n X_i \\ \sum_{i=1}^n X_i & n \end{bmatrix})^{-1} \begin{bmatrix} \sum_{i=1}^n X_i Y_i \\ \sum_{i=1}^n Y_i \end{bmatrix}$$

We denote:

$$E(X^2) = \frac{1}{n}\sum_{i=1}^n X_i^2 \quad E(X) = \frac{1}{n}\sum_{i=1}^n X_i \quad E(XY) = \frac{1}{n}\sum_{i=1}^n X_i Y_i \quad E(Y) = \frac{1}{n}\sum_{i=1}^n Y_i$$

$$Var(X) = E(X^2) - E(X)^2 \quad Var(Y) = E(Y^2) - E(Y)^2 \quad Cov(X,Y) = E(XY) - E(X)E(Y)$$

$$\hat{w} = \begin{bmatrix} w_1 \\ w_0 \end{bmatrix} = \frac{1}{n^2E(X^2) - n^2E(X)^2} \begin{bmatrix} n & -nE(X) \\ -nE(X) & nE(X^2) \end{bmatrix} \begin{bmatrix} nE(XY) \\ nE(Y) \end{bmatrix}$$

$$\hat{w} = \begin{bmatrix} w_1 \\ w_0 \end{bmatrix} = \frac{1}{Var(X)} \begin{bmatrix} E(XY) - E(X)E(Y) \\ -E(X)E(XY) + E(X^2)E(Y) \end{bmatrix}$$

$$\hat{w} = \begin{bmatrix} w_1 \\ w_0 \end{bmatrix} = \begin{bmatrix} \frac{Cov(X,Y)}{Var(X)} \\ \frac{E(X^2)E(Y) - E(X)E(XY)}{Var(X)} \end{bmatrix}$$

Therefore, we have:

$$w_1 = \frac{Cov(X,Y)}{Var(X)} \quad w_0 = \frac{E(X^2)E(Y) - E(X)E(XY)}{Var(X)}$$

(c) Now, consider a different generative model. Let $Y = Xw + Z$, where $Z \sim U[-0.5, 0.5]$ is a continuous random variable uniformly distributed between $-0.5$ and $0.5$. Again assume that $w$ is a fixed parameter. **What is the conditional distribution of $Y$ given $X$?**

For uniform distribution, we have simple PDF:

$$P(Z = z) = \begin{cases} 1 & |z| \leq 0.5 \\ 0 & otherwise \end{cases}$$

Because $Z = Y - Xw$, we get:

$$P(Y = y|X = x) = \begin{cases} 1 & |y - xw| \leq 0.5 \\ 0 & otherwise \end{cases}$$

This means the conditional distribution of Y given X is also a uniform distribution $Y|X \sim U[-0.5, 0.5]$.

(d) Given $n$ points of training data $\{(X_1, Y_1), (X_2, Y_2), \cdots, (X_n, Y_n)\}$ generated in an i.i.d. fashion in the setting of the part (c) **derive a maximum likelihood estimator of $w$.** Assume that $X_i > 0$ for all $i = 1, \ldots, n$. (Note that MLE for this case need not be unique; but you are required to report only one particular estimate.)

In this case, the values of PDF don't help us that much. However, the condition in the PDF is useful. So $\forall X_i, Y_i$ we have $|Y_i - X_i w| \leq 0.5$ and because we know $X_i > 0$ it can be rewritten as

$$\max(\frac{Y_i - 0.5}{X_i}) \leq w \leq \min(\frac{Y_i + 0.5}{X_i})$$

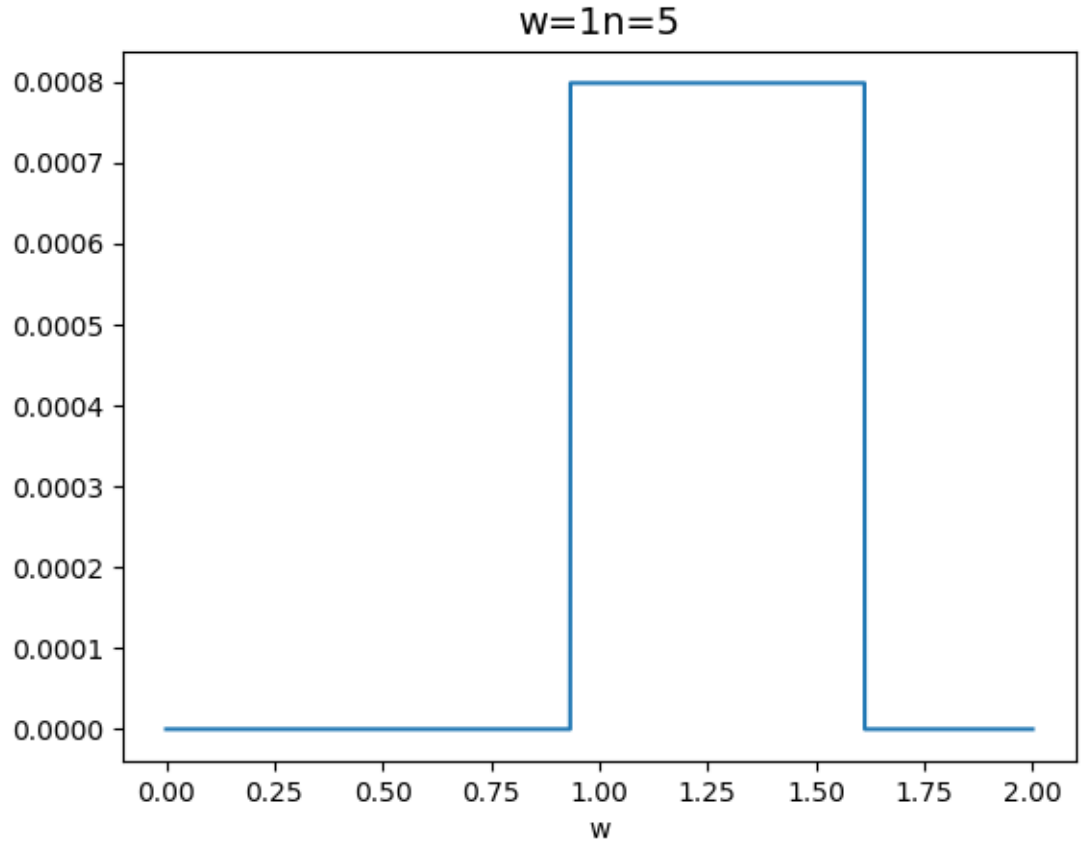
We have infinite number of estimators here so let's take the one in the middle.

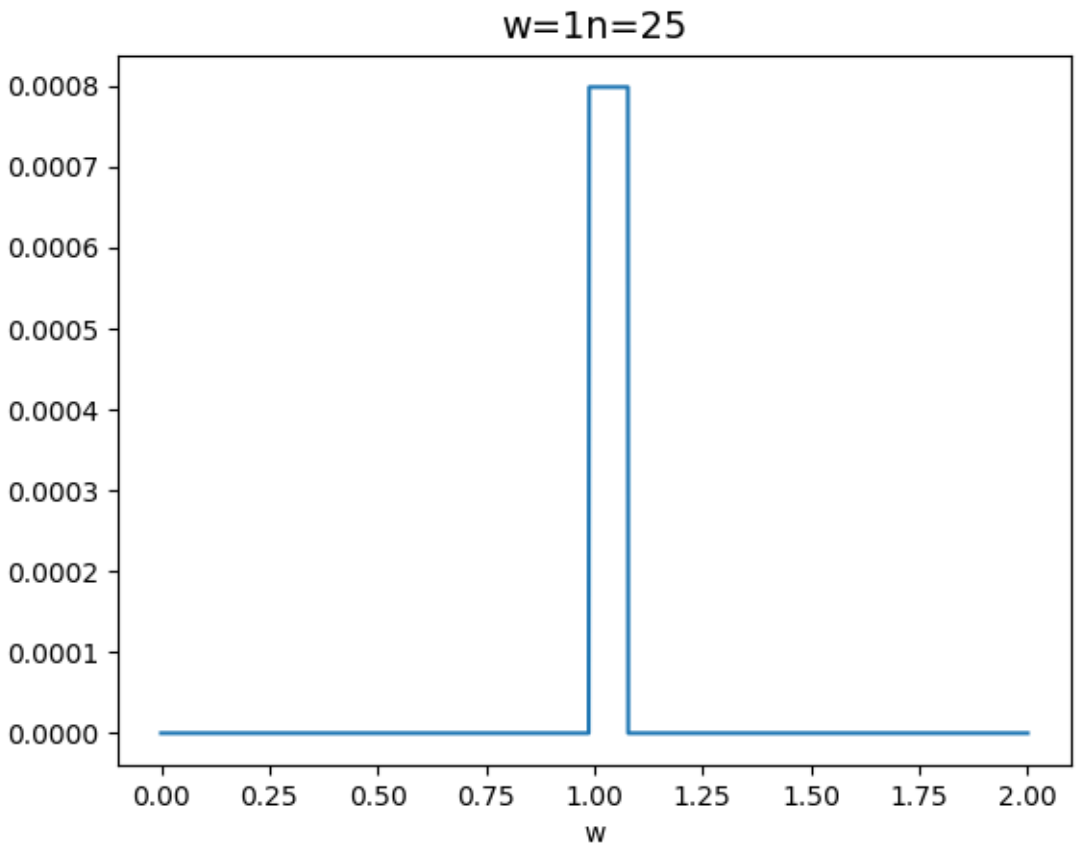
$$w = \frac{1}{2}(\max(\frac{Y_i - 0.5}{X_i}) + \min(\frac{Y_i + 0.5}{X_i}))$$

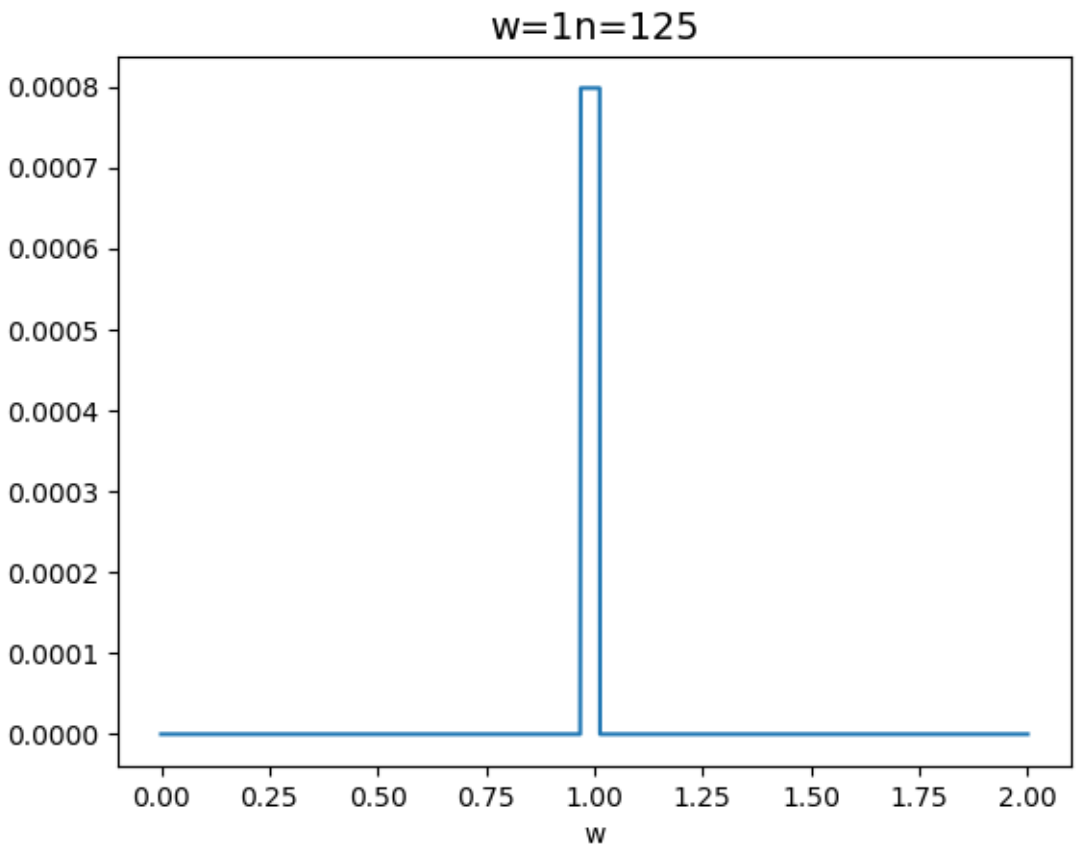
(e) Take the model $Y = Xw+Z$, where $Z \sim U[-0.5, 0.5]$. **Use a computer to simulate $n$ training samples $\{(X_1, Y_1), (X_2, Y_2), \cdots, (X_n, Y_n)\}$ and illustrate what the likelihood of the data looks like as a function of $w$ after $n = 5, 25, 125, 625$ training samples. Qualitatively describe what is happening as $n$ gets large.**

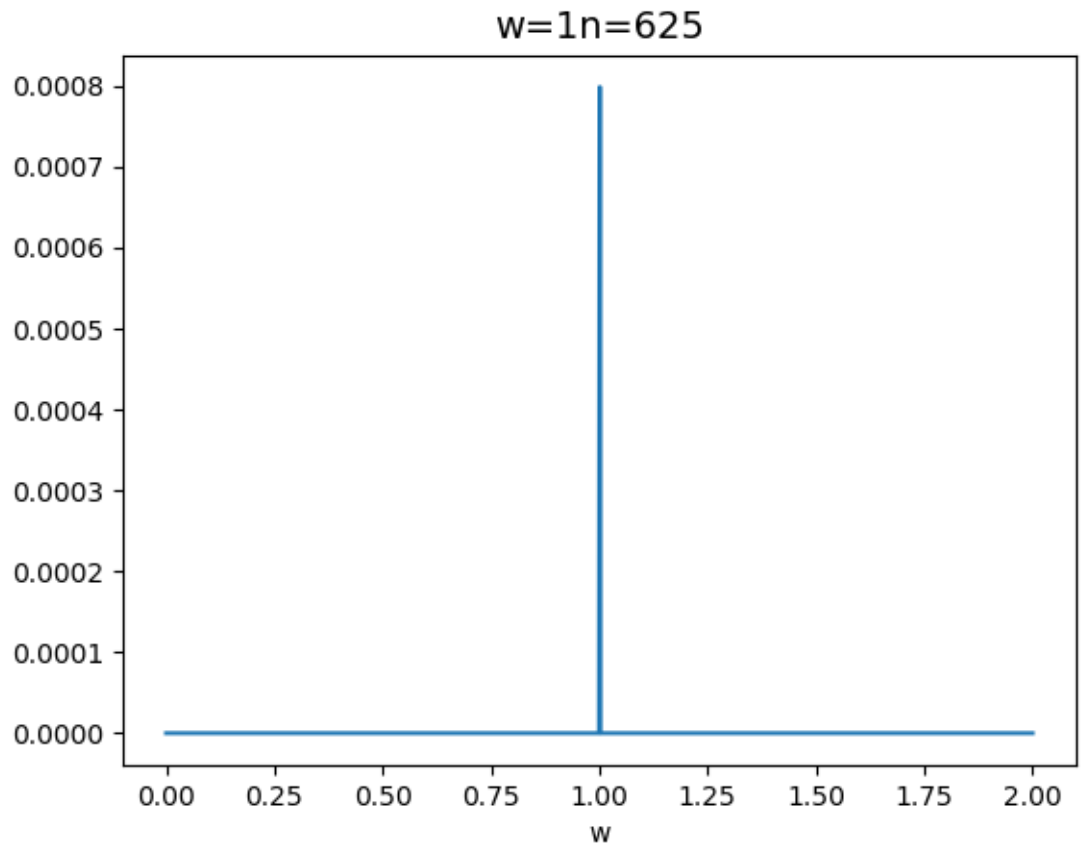
(You may use the starter code. Note that you have considerable design freedom in this problem part. You get to choose how you draw the $X_i$ as well as what true value $w$ you want to illustrate. You have total freedom in using additional python libraries for this problem part. No restrictions. )

Qualitatively, the larger the training sample is, the more accurate we can predict the true $w$ using the training data.




w=1n=25
0.0008
0.0007
0.0006
0.0005
0.0004
0.0003
0.0002
0.0001
0.0000
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
w


w=1n=125
0.0008
0.0007
0.0006
0.0005
0.0004
0.0003
0.0002
0.0001
0.0000
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
w




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

sample_size = [5, 25, 125, 625]
max_sample_size = sample_size[-1] * 2 # make sure we have straight lines
w = 1
np.random.seed(0)
count = 1
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.uniform(0, 1, n)
Z = np.random.uniform(-0.5, 0.5, n)
Y = X * w + Z
# make sure W covers the boundaries
W = np.hstack([np.linspace(0, np.max((Y - 0.5) / X), max_sample_size),
np.linspace(np.max((Y - 0.5) / X), np.min((Y + 0.5) / X), max_sample_size),
np.linspace(np.min((Y + 0.5) / X), 2, max_sample_size)])
N = len(W)
likelihood = np.ones(N) # likelihood as a function of w

for i1 in range(N):
if np.count_nonzero(abs(Y - X * W[i1]) <= 0.5) == n:
likelihood[i1] = 1
else:
likelihood[i1] = 0
# compute likelihood
likelihood /= sum(likelihood) # normalize the likelihood
```

```
plt.figure()
# plotting likelihood for different n
plt.plot(W, likelihood)
plt.xlabel('w', fontsize=10)
plt.title('w=' + str(w) + 'n=' + str(n), fontsize=14)
plt.show(block=False)
filename = 'Figure_2e' + str(count) + '.png'
savepath = os.path.join('.', filename)
plt.savefig(savepath)
count += 1
```

(f) (One-dimensional Ridge Regression) Now, let us return to the case of Gaussian noise. Given $n$ points of training data $\{(X_1, Y_1), (X_2, Y_2), \cdots, (X_n, Y_n)\}$ generated according to $Y_i = X_i W + Z_i$, where $Z_i \sim N(0, 1)$ are iid standard normal random variables. Assume $W \sim N(0, \sigma^2)$ is also a standard normal and is independent of both the $Z_i$'s and the $X_i$'s. **Use Bayes' Theorem to derive the posterior distribution of $W$ given the training data. What is the mean of the posterior distribution of $W$ given the data?**

Hint: Compute the posterior up-to proportionality and try to identify the distribution by completing the square.

$$P(W|X,Y) = \frac{P(X,Y|W)P(W)}{P(X,Y)} = \frac{1}{P(X,Y)} \frac{1}{(\sqrt{2\pi})^n} e^{-\sum_{i=1}^n \frac{(Y_i - X_i W)^2}{2}} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(W)^2}{2\sigma^2}}$$

To calculate the mean, we need to eyeball the above "PDF". We realize that the although it looks weird, it's still a Gaussian distribution's "PDF". We take take the integral of it and calculate it:

$$\int w P(W = w|X,Y) dw$$

But wait a minute, we haven't even normalize the above PDF. As is pointed out by the GSI during discussion section:

$$\int P(W = w|X,Y) dw = \text{Who know what}$$

We have do it in another way. Because $P(W = w|X,Y)$ is a Gaussian distribution's "PDF", we can also transform the above equation and get the mean from it directly. Here, we use the same notation:

$$E(X^2) = \frac{1}{n}\sum_{i=1}^n X_i^2 \quad E(X) = \frac{1}{n}\sum_{i=1}^n X_i \quad E(XY) = \frac{1}{n}\sum_{i=1}^n X_i Y_i \quad E(Y) = \frac{1}{n}\sum_{i=1}^n Y_i$$

$$P(W|X,Y) = \frac{1}{P(X,Y)} \frac{1}{(\sqrt{2\pi})^{n+1}\sigma} e^{-\frac{1}{2}((nE(x^2) + \frac{1}{\sigma^2})W^2 - 2nE(XY)W + nE(Y^2)}$$

From the last equation we can get the mean from exponent $\frac{nE(XY)}{nE(X^2)+\frac{1}{\sigma^2}}$.
(When I say "PDF", I mean a PDF like equation whose integral is not 1.)

(g) Consider $n$ training data points $\{(\vec{x}_1, Y_1), (\vec{x}_2, Y_2), \cdots, (\vec{x}_n, Y_n)\}$ generated according to $Y_i = \vec{w}^\top \vec{x}_i + Z_i$ where $Y_i \in \mathbb{R}, \vec{w}, \vec{x}_i \in \mathbb{R}^d$ with $\vec{w}$ fixed, and $Z_i \sim N(0,1)$ iid standard normal random variables. **Argue why the maximum likelihood estimator for $\vec{w}$ is the solution to a least squares problem.**

We write the log of maximum likelihood function in a matrix format like that in 2b:

$$\ln(L(\vec{w}|(X,Y)) = -n\ln(\sqrt{2\pi}) - \frac{1}{2}\|\hat{Y} - \hat{X}\hat{w}\|_2^2$$

$$\hat{X} = \begin{bmatrix} \vec{x}_1^\top \\ \vec{x}_2^\top \\ \vdots \\ \vec{x}_n^\top \end{bmatrix} \quad \hat{Y} = \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{bmatrix}$$

To get the maximum likelihood, we need to minimize $\|\hat{Y} - \hat{X}\hat{w}\|_2^2$, which is the same as a least square problem.

(h) (Multi-dimensional ridge regression) Consider the setup of the previous part: $Y_i = \vec{W}^\top \vec{x}_i + Z_i$, where $Y_i \in \mathbb{R}, \vec{W}, \vec{x}_i \in \mathbb{R}^d$, and $Z_i \sim N(0,1)$ iid standard normal random variables. Now we treat $\vec{W}$ as a random vector and assume a prior knowledge about its distribution. In particular, we use the prior information that the random variables $W_j$ are i.i.d. $\sim N(0, \sigma^2)$ for $j = 1, 2, \ldots, d$. **Derive the posterior distribution of $\vec{W}$ given all the $\vec{x}_i, Y_i$ pairs. What is the mean of the posterior distribution of the random vector $\vec{W}$?**

Hint: Use hints from part (f) and the following identities: For $\mathbb{X} = \begin{bmatrix} \vec{x}_1^\top \\ \vdots \\ \vec{x}_n^\top \end{bmatrix}$ and $\vec{Y} = \begin{bmatrix} Y_1 \\ \vdots \\ Y_n \end{bmatrix}$ we have $\mathbb{X}^\top\mathbb{X} = \sum_{i=1}^n \vec{x}_i\vec{x}_i^\top$ and $\mathbb{X}^T\vec{Y} = \sum_{i=1}^n \vec{x}_i Y_i$.

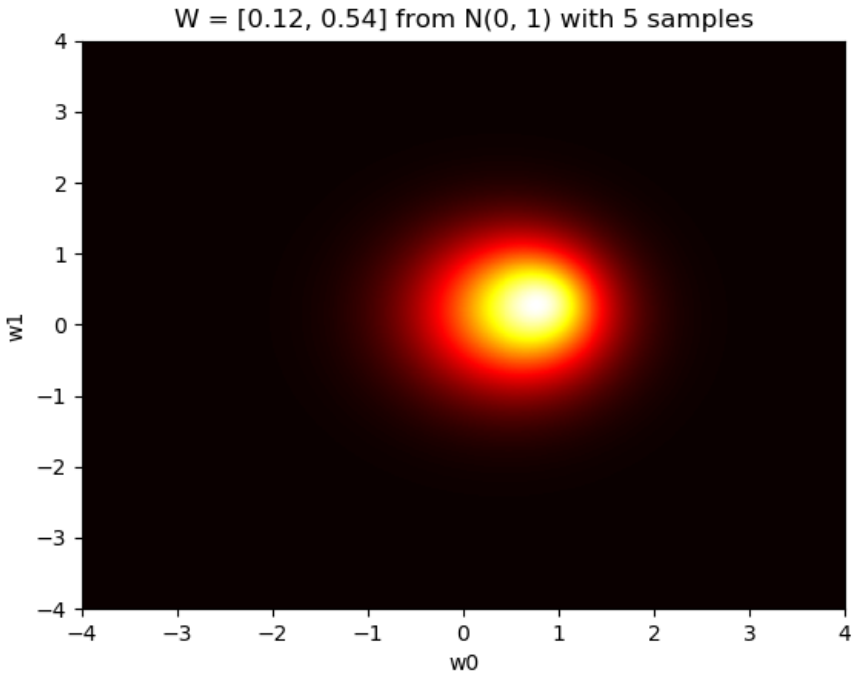
$$P(W|X,Y) = \frac{P(X,Y|W)P(W)}{P(X,Y)} = \frac{1}{P(X,Y)}\frac{1}{(\sqrt{2\pi})^n}e^{-\frac{\|\vec{Y}-\mathbb{X}\vec{W}\|_2^2}{2}}\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{\|W\|_2^2}{2\sigma^2}}$$

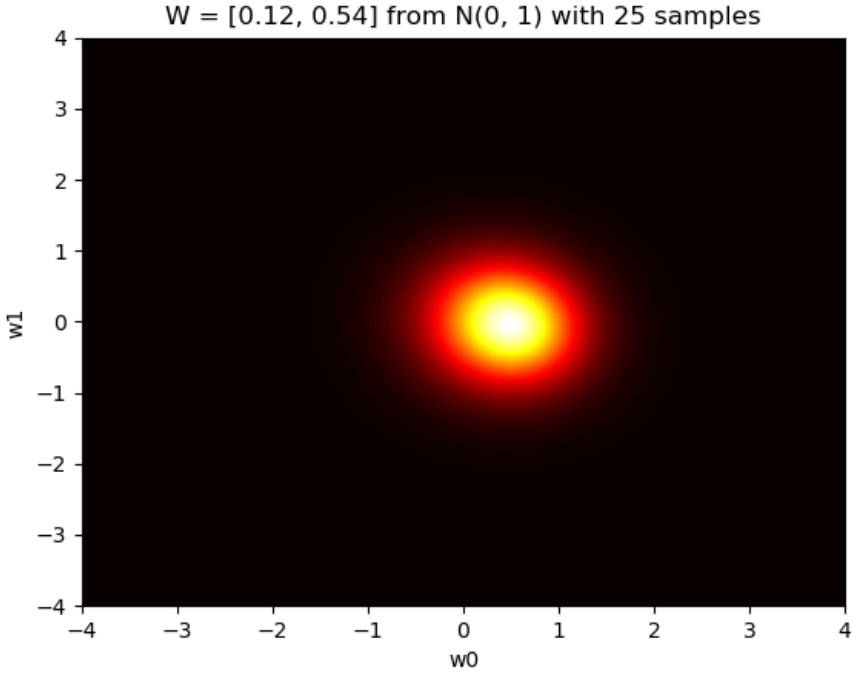
The mean of $\vec{W}$ is proportional to the solution to the optimization problem $\arg_W(\|\vec{Y} - \mathbb{X}\vec{W}\|_2^2 + \frac{1}{\sigma^2}\|\vec{W}\|_2^2)$ for the same reason as in 2f. Therefore, the mean is $(\mathbb{X}^\top\mathbb{X} + \frac{1}{\sigma^2}\mathbb{I})^{-1}\mathbb{X}^\top\vec{Y}$

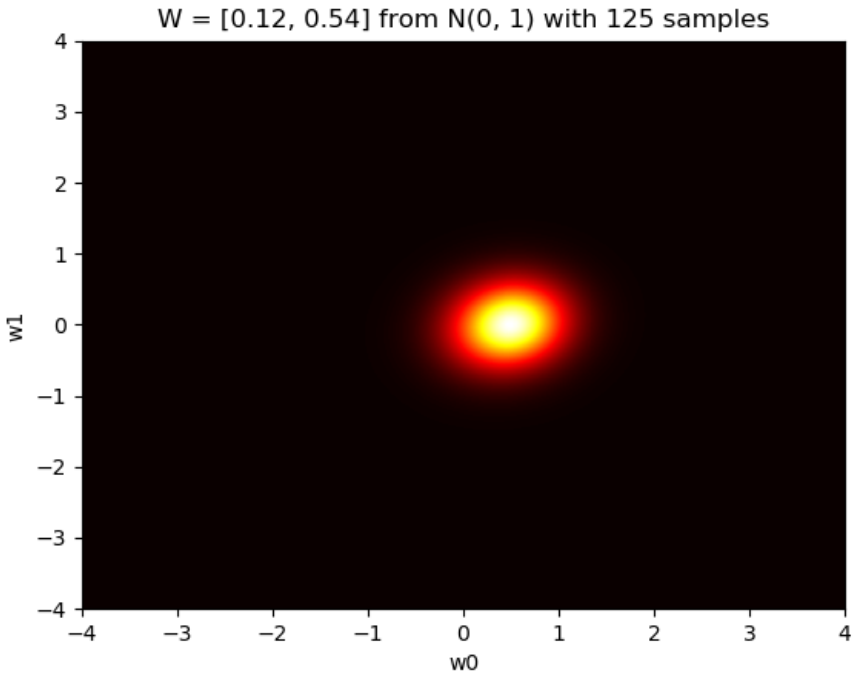
(i) Consider $d = 2$ and the setting of the previous part. **Use a computer to simulate and illustrate what the *a-posteriori* probability looks like for the $W$ model parameter space after $n = 5, 25, 125$ training samples for different values of $\sigma^2$.**

(Again, you may use the starter code. And like problem (e), there are no restrictions for using additional python libraries for this part as well.)

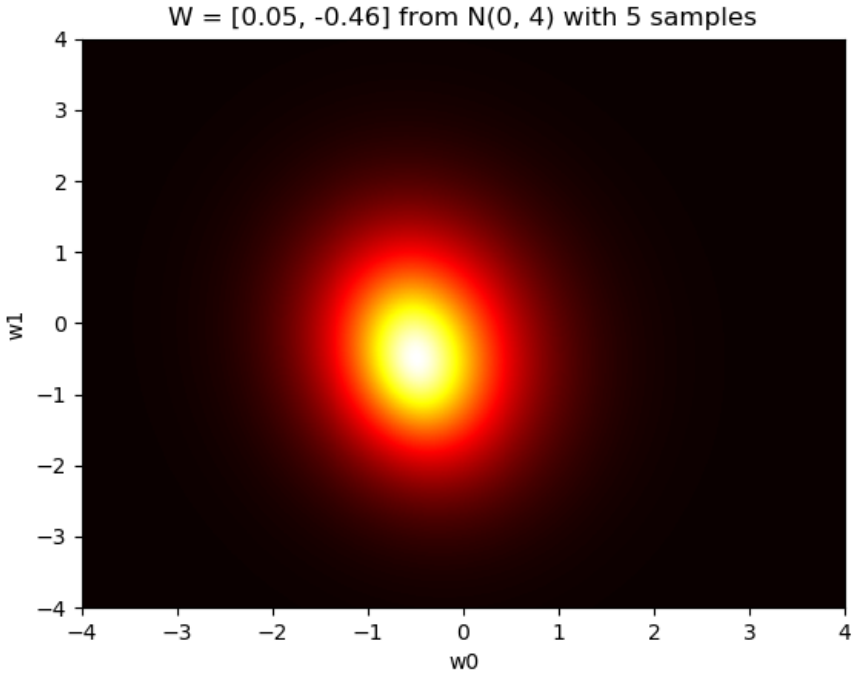
I used a seed of 0. $W = [0.11705680.53999046] \sim N(0,1)$:

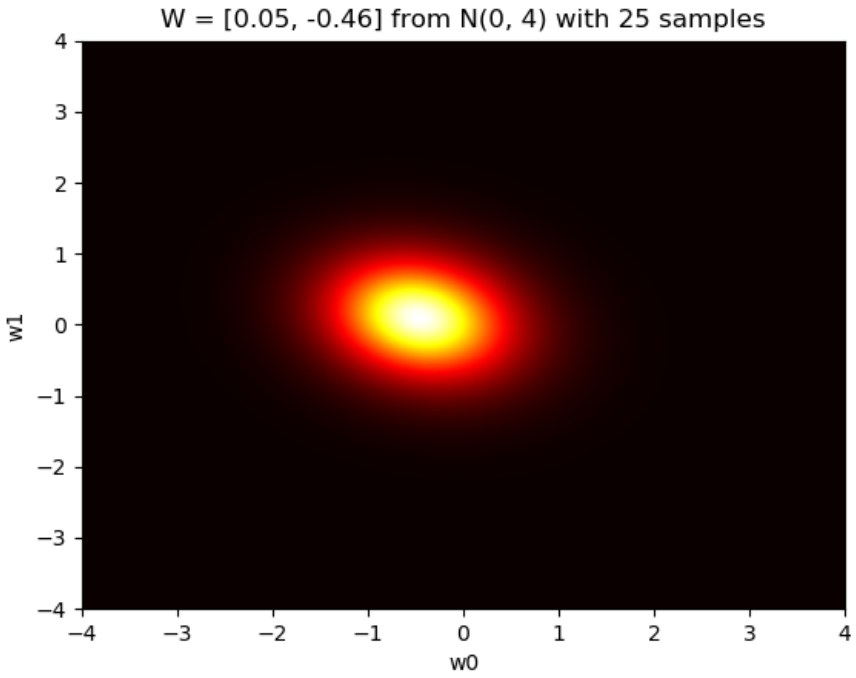

W = [0.12, 0.54] from N(0, 1) with 5 samples
w1
w0
4
3
2
1
0
−1
−2
−3
−4
−4
−3
−2
−1
0
1
2
3
4

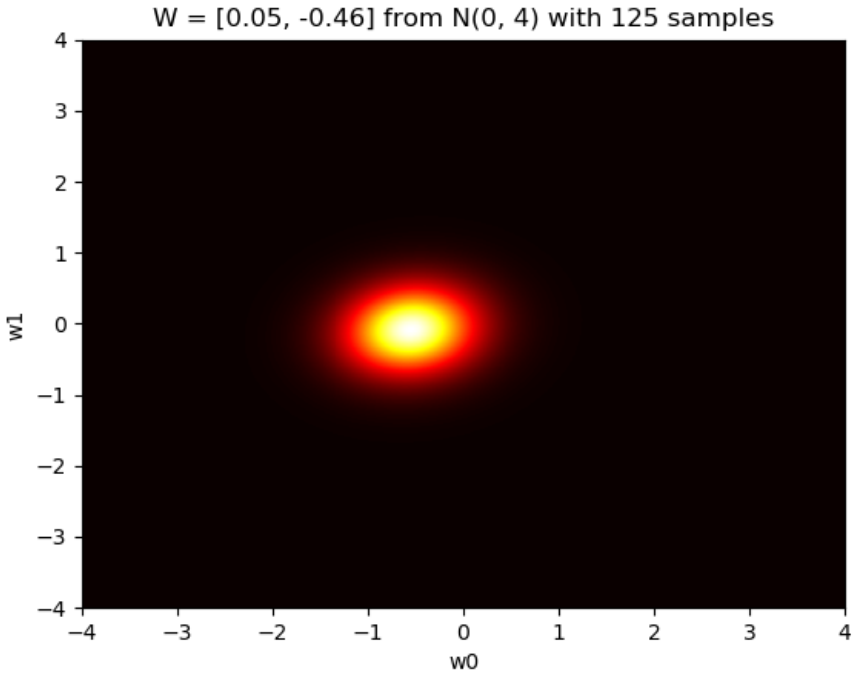

W = [0.12, 0.54] from N(0, 1) with 25 samples
w1
w0
4
3
2
1
0
−1
−2
−3
−4
−4
−3
−2
−1
0
1
2
3
4




$W = [1.06351333 - 1.49482314] \sim N(0, 4)$:






```
import numpy as np
import scipy.stats as scipystats
import matplotlib.pyplot as plt
import os

sample_size = [5, 25, 125]
Sigma = [1, 2]
WtrueAll = np.zeros((len(Sigma), 2))
Wmin = -4
Wmax = 4
count = 1
np.random.seed(0)
```

```
for iw in range(len(Sigma)):
Wtrue = scipystats.truncnorm(
Wmin / 2 / Sigma[iw], Wmax / 2 / Sigma[iw], loc=0, scale=Sigma[iw]).rvs(2)
WtrueAll[iw, :] = Wtrue
for k in range(len(sample_size)):
n = sample_size[k]
# generate data
# np.linspace, np.random.normal and np.random.uniform might be useful functions
X = np.random.normal(0, 1, (n, 2))
Z = np.random.normal(0, 1, n)
Y = X.dot(Wtrue) + Z

# compute likelihood
N = 1001
W0s = np.linspace(Wmin, Wmax, N)
W1s = np.linspace(Wmin, Wmax, N)

likelihood = np.ones([N, N]) # likelihood as a function of w_1 and w_0

for i1 in range(N):
w_0 = W0s[i1]
tempW0 = Y - X[:, 0] * w_0
for i2 in range(N):
w_1 = W1s[i2]
temp = tempW0 - X[:, 1] * w_1
# for i in range(n):
# compute the likelihood here
likelihood[i1, i2] = (1 / np.sqrt(2 * np.pi)) ** n \
* np.exp(- np.linalg.norm(temp) / 2) * \
(1 / np.sqrt(2 * np.pi) / Sigma[iw]) \
* np.exp(- (w_0 ** 2 + w_1 ** 2) / 2 / (Sigma[iw] ** 2))

# likelihood /= np.sum(likelihood)
# plotting the likelihood
plt.figure()
# for 2D likelihood using imshow
plt.imshow(likelihood, cmap='hot', aspect='auto', origin='lower', extent=[Wmin, Wmax, Wmin, Wmax])
plt.title('W = [%.2f, %.2f] from N(0, %d) with %d samples' % (Wtrue[0], Wtrue[1], Sigma[iw] ** 2, n))
plt.xlabel('w0')
plt.ylabel('w1')
plt.show(block=False)
filename = 'Figure_2i' + str(count) + '.png'
savepath = os.path.join('.', filename)
plt.savefig(savepath)
count += 1

print(WtrueAll)
```

**Question 3.** Simple Bias-Variance Tradeoff

Consider a random variable $X$, which has unknown mean $\mu$ and unknown variance $\sigma^2$. Given $n$ iid realizations of training samples $X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n$ from the random variable, we wish to estimate the mean of $X$. We will call our estimate of $X$ the random variable $\hat{X}$, which has mean $\hat{\mu}$. There are a few ways we can estimate $\mu$ given the realizations of the $n$ samples:

1. Average the $n$ samples: $\frac{x_1+x_2+\ldots+x_n}{n}$.
2. Average the $n$ samples and one sample of 0: $\frac{x_1+x_2+\ldots+x_n}{n+1}$.
3. Average the $n$ samples and $n_0$ samples of 0: $\frac{x_1+x_2+\ldots+x_n}{n+n_0}$.
4. Ignore the samples: just return 0.

In the parts of this question, we will measure the *bias* and *variance* of each of our estimators. The *bias* is defined as

$$E[\hat{X} - \mu]$$

and the *variance* is defined as

$$\text{Var}[\hat{X}].$$

(a) **What is the bias of each of the four estimators above?**

1. $E[\hat{X} - \mu] = E[\hat{X}] - \mu = \frac{1}{n}(E[X_1] + E[X_2] + \cdots + E[X_n]) - \mu = \frac{n\mu}{n} - \mu = 0$
2. $E[\hat{X} - \mu] = E[\hat{X}] - \mu = \frac{1}{n+1}(E[X_1] + E[X_2] + \cdots + E[X_n]) - \mu = \frac{n\mu}{n+1} - \mu = -\frac{\mu}{n+1}$
3. $E[\hat{X} - \mu] = E[\hat{X}] - \mu = \frac{1}{n+n_0}(E[X_1] + E[X_2] + \cdots + E[X_n]) - \mu = \frac{n\mu}{n+n_0} - \mu = -\frac{n_0\mu}{n+n_0}$
4. $E[\hat{X} - \mu] = E[\hat{X}] - \mu = 0 - \mu = -\mu$

(b) **What is the variance of each of the four estimators above?**

1.
$$\begin{aligned} & Var[\hat{X}] = E[\hat{X}^2] - E[\hat{X}]^2 \\ = \; & E[\hat{X}^2] - \mu^2 \\ = \; & \tfrac{1}{n^2}(E[X_1^2] + E[X_2^2] + \cdots + E[X_n^2] + 2E[X_1X_2] + 2E[X_1X_3] + \cdots + 2E[X_{n-1}X_n]) - \mu^2 \\ = \; & \tfrac{1}{n^2}(n(\sigma^2 + \mu^2) + n(n-1)\mu^2) - \mu^2 \\ = \; & \tfrac{1}{n}\sigma^2 \end{aligned}$$
2. $Var[\hat{X}] = \frac{1}{(n+1)^2}(n(\sigma^2 + \mu^2) + n(n-1)\mu^2) - \frac{n^2}{(n+1)^2}\mu^2 = \frac{n}{(n+1)^2}\sigma^2$
3. $Var[\hat{X}] = \frac{1}{(n+n_0)^2}(n(\sigma^2 + \mu^2) + n(n-1)\mu^2) - \frac{n^2}{(n+n_0)^2}\mu^2 = \frac{n}{(n+n_0)^2}\sigma^2$
4. $Var[\hat{X}] = 0 - 0 = 0$

(c) Suppose we have constructed an estimator $\hat{X}$ from some samples of $X$. We now want to know how well $\hat{X}$ estimates a fresh (new) sample of $X$. Denote this fresh sample by $X'$. Note that $X'$ is an

i.i.d. copy of the random variable $X$. **Derive a general expression for the expected squared error $E[(\hat{X} - X')^2]$ in terms of $\sigma^2$ and the bias and variance of the estimator $\hat{X}$. Similarly, derive an expression for the expected squared error $E[(\hat{X} - \mu)^2]$. Compare the two expressions and comment on the differences between them, if any.**

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

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

The difference is that the first error directly depends on $\sigma$ but the second doesn't. $E[(\hat{X} - X')^2] - E[(\hat{X} - \mu)^2] = \sigma^2$. The first error is no less than the second.

(d) It is a common mistake to assume that an unbiased estimator is always "best." Let's explore this a bit further. **Compute the expected squared error for each of the estimators above.**

1. $E[(\hat{X} - \mu)^2] = \frac{1}{n}\sigma^2 + \mu^2 - 2\mu^2 + \mu^2 = \frac{1}{n}\sigma^2$
2. $E[(\hat{X} - \mu)^2] = \frac{n}{(n+1)^2}\sigma^2 + \frac{n^2}{(n+1)^2}\mu^2 - \frac{2n^2+2n}{(n+1)^2}\mu^2 + \frac{n^2+2n+1}{(n+1)^2}\mu^2 = \frac{n}{(n+1)^2}\sigma^2 + \frac{1}{(n+1)^2}\mu^2$
3. $E[(\hat{X} - \mu)^2] = \frac{n}{(n+n_0)^2}\sigma^2 + \frac{n^2}{(n+n_0)^2}\mu^2 - \frac{2n^2+2nn_0}{(n+n_0)^2}\mu^2 + \frac{n^2+2nn_0+n_0^2}{(n+n_0)^2}\mu^2 = \frac{n}{(n+n_0)^2}\sigma^2 + \frac{n_0^2}{(n+n_0)^2}\mu^2$
4. $E[(\hat{X} - \mu)^2] = 0 + 0 - 0 + \mu^2 = \mu^2$

(e) **Demonstrate that the four estimators are each just special cases of the third estimator, but with different instantiations of the hyperparameter $n_0$.**

1. $n_0 = 0$
2. $n_0 = 1$
4. $n_0 \to \infty$

$$\lim_{n_0 \to \infty} \frac{x_1 + x_2 + \cdots + x_n}{n + n_0} = 0$$

(f) **What happens to bias as $n_0$ increases? What happens to variance as $n_0$ increases?**

Bias increases as $n_0$ increases.
Variance decreases as $n_0$ increases.

(g) Say that $n_0 = \alpha n$. **Find the setting for $\alpha$ that would minimize the expected total error, assuming you secretly knew $\mu$ and $\sigma$.** Your answer will depend on $\sigma$, $\mu$, and $n$.

$$\begin{aligned} & E[(\hat{X} - \mu)^2] \\ = & \frac{n}{(n+n_0)^2}\sigma^2 + \frac{n_0^2}{(n+n_0)^2}\mu^2 \\ = & \frac{1}{n(1+\alpha)^2}\sigma^2 + \frac{\alpha^2}{(1+\alpha)^2}\mu^2 \end{aligned}$$

We take the derivative and find the minima: $2(1+\alpha)(\mu^2\alpha - \frac{\sigma^2}{n}) = 0$. Therefore $\alpha = \frac{\sigma^2}{n\mu^2}$

(h) For this part, let's assume that we had some reason to believe that $\mu$ *should be small* (close to 0) and $\sigma$ *should be large*. In this case, **what happens to the expression in the previous part?**

$\alpha$ will get really large (approach infinity) preferring estimator 4 to estimator 1.

(i) In the previous part, we assumed there was reason to believe that $\mu$ *should be small*. Now let's assume that we have reason to believe that $\mu$ is not necessarily small, but *should be close to some fixed value* $\mu_0$. **In terms of $X$ and $\mu_0$, how can we define a new random variable $X'$ such that $X'$ is expected to have a small mean? Compute the mean and variance of this new random variable.**

As is pointed out on Piazza, small means close to zero.
Thus, we can define $X' = X - \mu_0$.

$$E[X'] = E[X] - \mu_0$$
$$Var[X'] = Var[X] - 0 = \sigma^2$$

(j) Draw a connection between $\alpha$ in this problem and the regularization parameter $\lambda$ in the ridge-regression version of least-squares. **What does this problem suggest about choosing a regularization coefficient and handling our data-sets so that regularization is most effective**? This is an open-ended question, so do not get too hung up on it.

$\alpha$ balances between the bias and the variance of the estimator. If the sample mean is really big, then $\alpha$ should be small to prioritize the bias; if the sample variance is really big, then $\alpha$ should be big to prioritize the variance. $\lambda$ plays a similar role in ridge regression.

It says in the question that I should not get too hung up on it, so I will let it go assuming this is enough to get credit.

**Question 4.** Estimation and approximation in linear regression

In typical applications, we are dealing with data generated by an *unknown* function (with some noise), and our goal is to estimate this function. So far we used linear and polynomial regressions. In this problem we will explore the quality of polynomial regressions when the true function is not polynomial.

Suppose we are given a full column rank feature matrix $\mathbb{X} \in \mathbb{R}^{n \times d}$ and an observation vector $\vec{y} \in \mathbb{R}^n$. Let the vector $\vec{y}$ represent the noisy measurement of a true signal $\vec{y}^*$:

$$\vec{y} = \vec{y}^* + \vec{z}, \tag{1}$$

with $\vec{z} \in \mathbb{R}^n$ representing the random noise in the observation $\vec{y}$, where $z_j \sim \mathcal{N}(0, \sigma^2)$ are i.i.d. We define the vectors $\vec{w}^*$ and $\hat{\vec{w}}$ as follows:

$$\vec{w}^* = \arg\min_{\vec{w}} \|\vec{y}^* - \mathbb{X}\vec{w}\|_2^2 \qquad \text{and} \qquad \hat{\vec{w}} = \arg\min_{\vec{w}} \|\vec{y} - \mathbb{X}\vec{w}\|_2^2.$$

Observe that for a given true signal $\vec{y}^*$ the vector $\vec{w}^*$ is fixed, but the vector $\hat{\vec{w}}$ is a random variable since it is a function of the random noise $\vec{z}$. Note that the vector $\mathbb{X}\vec{w}^*$ is the best linear fit of the true signal $\vec{y}^*$ in the column space of $\mathbb{X}$. Similarly, the vector $\mathbb{X}\hat{\vec{w}}$ is the best linear fit of the observed noisy signal $\vec{y}$ in the column space of $\mathbb{X}$.

After obtaining $\hat{\vec{w}}$, we would like to bound the error $\|\mathbb{X}\hat{\vec{w}} - \vec{y}^*\|_2^2$, which is the *prediction error* incurred based on the specific $n$ training samples. In this problem we will see how to get a good estimate of this prediction error. When using polynomial features, we will also learn how to decide the degree of the polynomial when trying to fit a noisy set of observations from a smooth function.

**Remark**: You can use the closed form solution for OLS and results from Discussion 3 for all parts of this problem. For parts (a)-(c), assume that the feature matrix $\mathbb{X}$ and the true signal vector $\vec{y}^*$ are fixed (and not random). Furthermore, in all parts the expectation is taken over the randomness in the noise vector $\vec{z}$.

(a) **Show that** $\mathbb{E}[\hat{\vec{w}}] = \vec{w}^*$ and use this fact to **show that**

$$\mathbb{E}\left[\|\vec{y}^* - \mathbb{X}\hat{\vec{w}}\|_2^2\right] = \|\vec{y}^* - \mathbb{E}[\mathbb{X}\hat{\vec{w}}]\|_2^2 + \mathbb{E}\left[\|\mathbb{X}\hat{\vec{w}} - \mathbb{E}[\mathbb{X}\hat{\vec{w}}\|_2^2\right].$$

Note that the above decomposition of the squared error corresponds to the sum of bias-squared and the variance of our estimator $\mathbb{X}\hat{\vec{w}}$.

$$\begin{aligned}
& \mathbb{E}[\hat{\vec{w}}] \\
= \; & \mathbb{E}[(\mathbb{X}^\top\mathbb{X})^{-1}\mathbb{X}^\top\vec{y}] \\
= \; & \mathbb{E}[(\mathbb{X}^\top\mathbb{X})^{-1}\mathbb{X}^\top(\vec{y}^* + \vec{z})] \\
= \; & \mathbb{E}[(\mathbb{X}^\top\mathbb{X})^{-1}\mathbb{X}^\top\vec{y}^*] + \mathbb{E}[(\mathbb{X}^\top\mathbb{X})^{-1}\mathbb{X}^\top\vec{z}] \\
= \; & \vec{w}^* + 0 \\
= \; & \vec{w}^*
\end{aligned}$$

$$
\begin{aligned}
& \mathbb{E}\left[\|\vec{y}^* - \mathbb{X}\hat{\vec{w}}\|_2^2\right] \\
= \; & \mathbb{E}\left[(\vec{y}^* - \mathbb{X}\hat{\vec{w}})^\top(\vec{y}^* - \mathbb{X}\hat{\vec{w}})\right] \\
= \; & \mathbb{E}\left[\vec{y^*}^\top\vec{y^*} - 2\vec{y^*}^\top\mathbb{X}\hat{\vec{w}} + \hat{\vec{w}}^\top\mathbb{X}^\top\mathbb{X}\hat{\vec{w}}\right] \\
= \; & \vec{y^*}^\top\vec{y^*} - 2\vec{y^*}^\top\mathbb{E}[\mathbb{X}\hat{\vec{w}}] + \mathbb{E}\left[\hat{\vec{w}}^\top\mathbb{X}^\top\mathbb{X}\hat{\vec{w}}\right] \\
= \; & \vec{y^*}^\top\vec{y^*} - 2\vec{y^*}^\top\mathbb{E}[\mathbb{X}\hat{\vec{w}}] + \mathbb{E}[\mathbb{X}\hat{\vec{w}}]^2 - \mathbb{E}[\mathbb{X}\hat{\vec{w}}]^2 + \mathbb{E}\left[\hat{\vec{w}}^\top\mathbb{X}^\top\mathbb{X}\hat{\vec{w}}\right] \\
= \; & \|\vec{y}^* - \mathbb{E}[\mathbb{X}\hat{\vec{w}}]\|_2^2 + \mathrm{Var}[\mathbb{X}\hat{\vec{w}}] \\
= \; & \|\vec{y}^* - \mathbb{E}[\mathbb{X}\hat{\vec{w}}]\|_2^2 + \mathbb{E}\left[\|\mathbb{X}\hat{\vec{w}} - \mathbb{E}[\mathbb{X}\hat{\vec{w}}]\|_2^2\right].
\end{aligned}
$$

(b) Recall that if $\vec{v} \sim \mathcal{N}(\vec{0}, \Sigma)$, where $\Sigma \in \mathbb{R}^{(d\times d)}$ is the covariance matrix, then for any matrix $A \in \mathbb{R}^{k\times d}$, we have $\mathbb{A}\vec{v} \sim \mathcal{N}(\vec{0}, \mathbb{A}\Sigma\mathbb{A}^T)$. Use this fact to **show that** the distribution of the vector $\hat{\vec{w}}$ is given by

$$\hat{\vec{w}} \sim \mathcal{N}(\vec{w}^*, \sigma^2(\mathbb{X}^\top\mathbb{X})^{-1}).$$

In our case we have $\vec{z} \sim \mathcal{N}(0, \sigma^2\mathbb{I})$, and $\hat{\vec{w}} = (\mathbb{X}^\top\mathbb{X})^{-1}\mathbb{X}^\top(\vec{z} + \vec{y}^*) = (\mathbb{X}^\top\mathbb{X})^{-1}\mathbb{X}^\top\vec{z} + \vec{w}^*$. We can see that $\vec{w}$ is a transformed Gaussian distribution with a shifted mean $\vec{w}^*$. Therefore, we just need to use the rule above and compute the variance:

$$
\begin{aligned}
& \Sigma[\vec{w}] \\
= \; & \sigma^2(\mathbb{X}^\top\mathbb{X})^{-1}\mathbb{X}^\top((\mathbb{X}^\top\mathbb{X})^{-1}\mathbb{X}^\top)^\top \\
= \; & \sigma^2(\mathbb{X}^\top\mathbb{X})^{-1}\mathbb{X}^\top\mathbb{X}(\mathbb{X}^\top\mathbb{X})^{-1} \\
= \; & \sigma^2(\mathbb{X}^\top\mathbb{X})^{-1}
\end{aligned}
$$

Therefore, $\hat{\vec{w}} \sim \mathcal{N}(\vec{w}^*, \sigma^2(\mathbb{X}^\top\mathbb{X})^{-1})$.

(c) Use part (b) to **show that**

$$\frac{1}{n}\mathbb{E}\left[\|\mathbb{X}\hat{\vec{w}} - \mathbb{X}\vec{w}^*\|_2^2\right] = \sigma^2\frac{d}{n}.$$

Hint: The trace trick: $\mathrm{trace}(AB) = \mathrm{trace}(BA)$, might be useful.

We need to compute the variance of $\mathbb{X}\vec{w}$ first:

$$
\begin{aligned}
& \Sigma[\mathbb{X}\hat{\vec{w}}] \\
= \; & \sigma^2\mathbb{X}(\mathbb{X}^\top\mathbb{X})^{-1}\mathbb{X}^\top
\end{aligned}
$$

$$
\begin{aligned}
& \tfrac{1}{n}\mathbb{E}\left[\|\mathbb{X}\hat{\vec{w}} - \mathbb{X}\vec{w}^*\|_2^2\right] \\
= \; & \tfrac{1}{n}\mathbb{E}\left[\|\mathbb{X}\hat{\vec{w}} - \mathbb{E}[\mathbb{X}\hat{\vec{w}}]\|_2^2\right] \\
= \; & \tfrac{1}{n}\sum_{i=1}^{d}\mathbb{E}\left[(\mathbb{X}_i\hat{\vec{w}} - \mathbb{E}[\mathbb{X}_i\hat{\vec{w}}])^2\right] \\
= \; & \tfrac{1}{n}\sum_{i=1}^{d}\mathrm{Var}\left[\mathbb{X}_i\hat{\vec{w}}\right] \\
= \; & \tfrac{1}{n}tr(\Sigma[\mathbb{X}\hat{\vec{w}}]) \\
= \; & \tfrac{1}{n}tr(\sigma^2\mathbb{X}(\mathbb{X}^\top\mathbb{X})^{-1}\mathbb{X}^\top) \\
= \; & \tfrac{1}{n}tr(\sigma^2\mathbb{X}(\mathbb{X}^\top\mathbb{X}^\top\mathbb{X})^{-1}) \\
= \; & \tfrac{1}{n}tr(\sigma^2\mathbb{I}) \\
= \; & \tfrac{1}{n}\sigma^2 d
\end{aligned}
$$

We notice that when we compute the variance of the norm, it's just the trace of the covariance matrix.

(d) Assume the underlying model is a noisy linear model with scalar samples $\{\alpha_i, y_i\}_{i=1}^n$, i.e. $y_i = w_1\alpha_i + w_0 + z_i$. We construct matrix $\mathbb{X}$ by using $D+1$ polynomial features $\vec{P}_D(\alpha_i) = [1, \alpha_i, \ldots, \alpha_i^D]^\top$ of the *distinct* sampling points $\{\alpha_i\}_{i=1}^n$. For any $D \geq 1$, compare with model (1) and **compute $\vec{w}^*$ for this case. Also compute the bias ($\|\vec{y}^* - \mathbb{X}\vec{w}^*\|_2$) for this case.** Using the previous parts of this problem, **compute the number of samples $n$ required to ensure that the average expected prediction squared error is bounded by $\epsilon$?** Your answer should be expressed as a function of $D$, $\sigma^2$, and $\epsilon$.

**Conclude that as we increase model complexity, we require a proportionally larger number of samples for accurate prediction.**

I don't understand what the question means so I put both answers here:

$$\vec{w}^* = (\mathbb{X}^\top\mathbb{X})^{-1}\mathbb{X}^\top\vec{y}^*$$

$$\vec{w}^* = \begin{bmatrix} w_0 \\ w_1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Because we know the true model, so we can predict the true values without any bias. However, I will put both answers just in case I understand the question wrong:

$$
\begin{aligned}
& \|\vec{y}^* - \mathbb{X}\vec{w}^*\|_2 \\
= \; & \sqrt{\|\vec{y}^* - \mathbb{X}\vec{w}^*\|_2^2} \\
= \; & \sqrt{(\vec{y}^*)^\top\vec{y}^* - 2(\vec{y}^*)^\top\mathbb{X}\vec{w}^* + (\vec{w}^*)^\top\mathbb{X}^\top\mathbb{X}\vec{w}^*} \\
= \; & \sqrt{(\vec{y}^*)^\top\vec{y}^* - (\vec{y}^*)^\top\mathbb{X}(\mathbb{X}^\top\mathbb{X})^{-1}\mathbb{X}^\top\vec{y}^*}
\end{aligned}
$$

$$\|\vec{y}^* - \mathbb{X}\vec{w}^*\|_2 = 0$$

We have shown that $\frac{1}{n}\mathbb{E}\left[\|\mathbb{X}\hat{\vec{w}} - \mathbb{X}\vec{w}^*\|_2^2\right] = \sigma^2\frac{d}{n}$. for $X \in \mathbb{R}^{n\times d}$. In this question, we just need to replace $d$ by $D+1$ and get:

$$\frac{1}{n}\mathbb{E}\left[\|\mathbb{X}\hat{\vec{w}} - \mathbb{X}\vec{w}^*\|_2^2\right] = \sigma^2\frac{D+1}{n}$$

Use the formula in 4a we have:

$$\frac{1}{n}\mathbb{E}\left[\|\vec{y}^* - \mathbb{X}\hat{\vec{w}}\|_2^2\right] = \sigma^2\frac{D+1}{n}$$

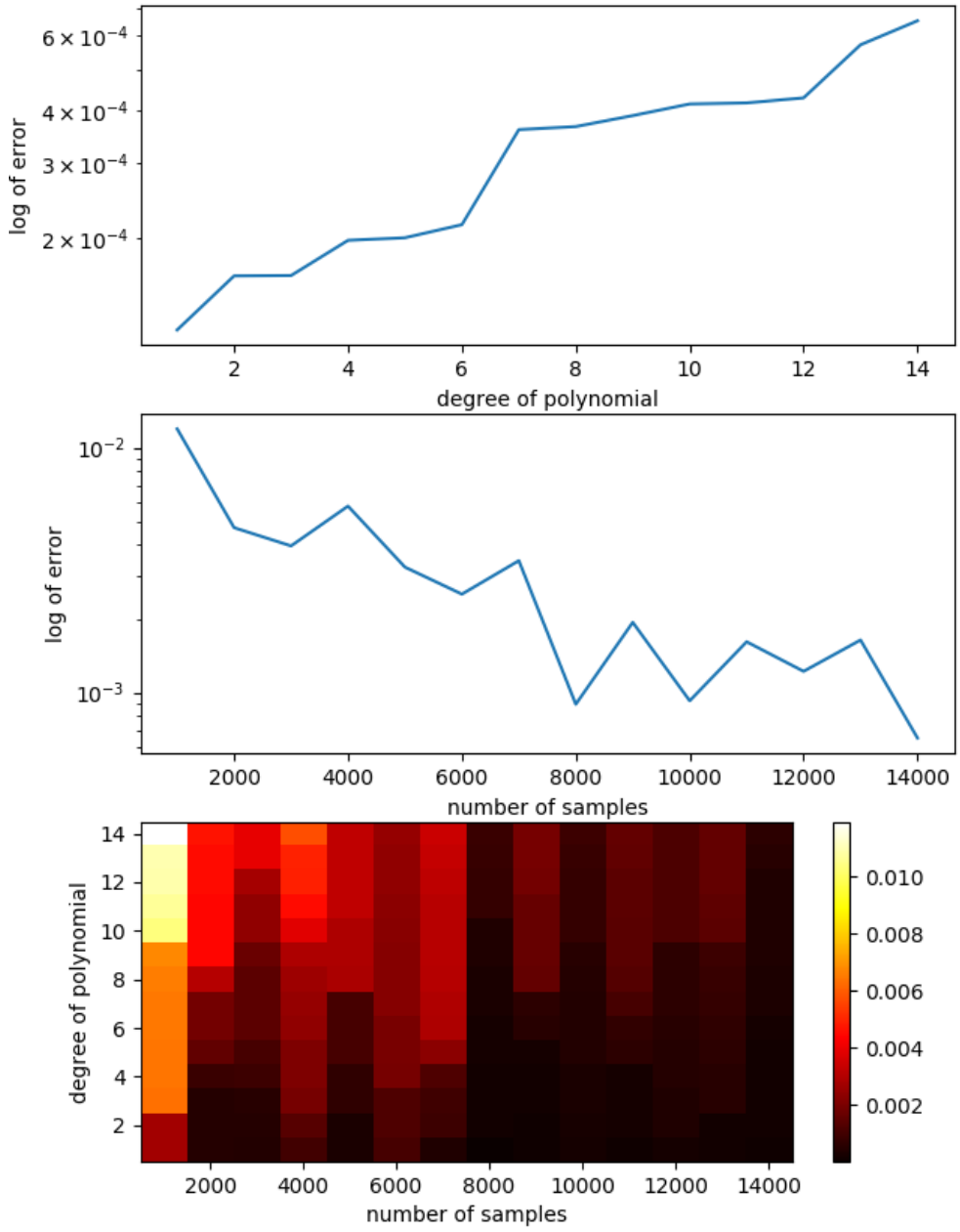
Therefore, we want $\sigma^2\frac{D+1}{n} \leq \epsilon$. It's easy to get $n \geq \frac{\sigma^2(D+1)}{\epsilon}$ Based on the formula above, we can conclude that as we increase model complexity (polynomial degree $D$), we require a proportionally larger number of samples for accurate prediction as a linear function of $D$.

(e) Simulate the problem from part (d) for yourself. Set $w_1 = 1$, $w_0 = 1$, and sample $n$ points $\{\alpha_i\}_{i=1}^n$ uniformly from the interval $[-1, 1]$. Generate $y_i = w_1\alpha_i + w_0 + z_i$ with $z_i$ representing standard Gaussian noise.

**Fit a $D$ degree polynomial to this data and show how the average error $\frac{1}{n}\|\mathbb{X}\hat{\vec{w}} - \vec{y}^*\|_2^2$ scales as a function of both $D$ and $n$.**

You may show separate plots for the two scalings. It may also be helpful to average over multiple realizations of the noise (or to plot point clouds) so that you obtain smooth curves.

(For this part, the libraries numpy.random and numpy.polyfit might be useful. You are free to use any and all python libraries.)




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

# assign problem parameters
w1 = 1
w0 = 1
sample_size_increment = 1000
sample_sizes = np.arange(sample_size_increment, 15000, sample_size_increment)
degrees = np.arange(1, 15)
error = np.zeros((len(degrees), len(sample_sizes)))
```

```
for iN in range(len(sample_sizes)):
# generate data
# np.random might be useful
N = sample_sizes[iN]
alpha = np.random.uniform(-1, 1, N)
Z = np.random.normal(0, 1, N)
Y_true = alpha * w1 + w0
Y = Y_true + Z
# fit data with different models
# np.polyfit and np.polyval might be useful
for ideg in range(len(degrees)):
D = degrees[ideg]
w = np.polyfit(alpha, Y, D)
y_predicted = np.polyval(w, alpha)
error[ideg, iN] = np.mean((y_predicted - Y_true) ** 2)

# plotting figures
# sample code

plt.figure(figsize=(7, 10))
plt.subplot(311)
plt.semilogy(degrees, error[:, -1])
plt.xlabel('degree of polynomial')
plt.ylabel('log of error')

plt.subplot(312)
plt.semilogy(sample_sizes, error[-1, :])
plt.xlabel('number of samples')
plt.ylabel('log of error')

plt.subplot(313)
plt.imshow(error, cmap='hot', aspect='auto', interpolation='none', origin='lower',
extent=[sample_sizes[0] - sample_size_increment / 2, sample_sizes[-1] + sample_size_increment / 2,
degrees[0] - 0.5, degrees[-1] + 0.5])
plt.colorbar()
plt.xlabel('number of samples')
plt.ylabel('degree of polynomial')

plt.show()
```

(f) Assume that the underlying model is the noisy exponential function with scalar samples $\{\alpha_i, y_i\}_{i=1}^n$ where $y_i = e^{\alpha_i} + z_i$ with *distinct* sampling points $\{\alpha_i\}_{i=1}^n$, in the interval $[-4, 3]$ and i.i.d. Gaussian noise $z_i \sim \mathcal{N}(0, 1)$. We again construct matrix $\mathbb{X}$ by using $D + 1$ polynomial features $[1, \alpha_i, \ldots, \alpha_i^D]^\top$ and use linear regression to fit the observations. Recall, the definitions of the bias and variance of the OLS estimator from part (a) and **show that for a fixed $n$, as the degree $D$ of the polynomial increases: (1) the bias decreases and (2) the variance increases**. Use the values you derived for bias and variance and **show that for the prediction error, the optimal choice of $D$ is given by** $\mathcal{O}(\log n / \log \log n)$.

Hint: You can directly use previous parts of this problem, Discussion 3 and Problem 4 of HW 2. You may assume that $n$ and $D$ are large for approximation purposes.

(1) Recall taylor series of the exponential function at $x = 0$:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots + f^{m+1}(a(x))\frac{x^{m+1}}{(m+1)!}$$

Because we know that the sample points are in the interval $[-4, 3]$, the derivate of $e^x$ is bounded by $[e^{-4}, e^3]$. Therefore, we can get the upper bound of the bias for a polynomial degree of D:

$$|\vec{y}_i^* - \mathbb{X}_i\hat{\vec{w}}^*| = f^{D+1}(a(x))\frac{x^{D+1}}{(D+1)!} \leq e^3\frac{x^{D+1}}{(D+1)!} \approx e^3\frac{4^{D+1}e^{D+1}}{\sqrt{2\pi(D+1)}(D+1)^{D+1}}$$

We have shown that the above formula of bias decreases as $D$ increases. (2) We have derived the variance above:

$$\frac{1}{n}\mathbb{E}\left[\|\mathbb{X}\hat{\vec{w}} - \mathbb{X}\vec{w}^*\|_2^2\right] = \sigma^2\frac{D+1}{n} = \frac{D+1}{n}$$

It's obvious that the variance increases when $D$ increases. (3) We want to minimize the prediction error

$$\min\frac{1}{n}\mathbb{E}\left[\|\vec{y}^* - \mathbb{X}\hat{\vec{w}}\|_2^2\right] = \frac{1}{n}\mathbb{E}\left[\|\mathbb{X}\hat{\vec{w}} - \mathbb{X}\vec{w}^*\|_2^2\right] + \frac{1}{n}\|\vec{y}^* - \mathbb{X}\hat{\vec{w}}^*\|_2^2$$

$$\min\frac{1}{n}\mathbb{E}\left[\|\vec{y}^* - \mathbb{X}\hat{\vec{w}}\|_2^2\right] = \mathbb{E}\left[\|\mathbb{X}\hat{\vec{w}} - \mathbb{X}\vec{w}^*\|_2^2\right] + \frac{1}{n}n|\vec{y}_i^* - \mathbb{X}_i\hat{\vec{w}}^*|^2$$

$$\begin{aligned}
& \nabla\tfrac{1}{n}\mathbb{E}\left[\|\vec{y}^* - \mathbb{X}\hat{\vec{w}}\|_2^2\right] \\
= \; & \nabla\tfrac{1}{n}\mathbb{E}\left[\|\mathbb{X}\hat{\vec{w}} - \mathbb{X}\vec{w}^*\|_2^2\right] + \nabla\tfrac{1}{n}\|\vec{y}^* - \mathbb{X}\hat{\vec{w}}^*\|_2^2 \\
= \; & \nabla\tfrac{D+1}{n} + \nabla(e^3\tfrac{4^{D+1}e^{D+1}}{\sqrt{2\pi(D+1)}(D+1)^{D+1}})^2 \\
= \; & \tfrac{1}{n} + \nabla\tfrac{e^6}{2\pi}\tfrac{(16e^2)^{D+1}}{(D+1)^{2D+3}}
\end{aligned}$$

We let $A = \frac{e^6}{2\pi}$ and $t = 16e^2$, $B = \ln(T) - 2$:

$$\begin{aligned}
& \nabla\tfrac{1}{n}\mathbb{E}\left[\|\vec{y}^* - \mathbb{X}\hat{\vec{w}}\|_2^2\right] \\
= \; & \tfrac{1}{n} + A\nabla\tfrac{T^{D+1}}{(D+1)^{2D+3}} \\
= \; & \tfrac{1}{n} + A\tfrac{\ln(T)T^{D+1}(D+1)^{2D+3} - T^{D+1}(D+1)^{2D+3}(2\ln(D+1) + \frac{2D+3}{D+1})}{(D+1)^{4D+6}} \\
= \; & \tfrac{1}{n} + A(\ln(T) - (2\ln(D+1) + 2 + \tfrac{1}{D+1}))\tfrac{T^{D+1}}{(D+1)^{2D+3}} \\
= \; & \tfrac{1}{n} + AB - A(2\ln(D+1) + \tfrac{1}{D+1}))\tfrac{T^{D+1}}{(D+1)^{2D+3}}
\end{aligned}$$

Therefore, we have the equality:

$$\frac{1}{n} = A(2\ln(D+1) + \frac{1}{D+1}) - B)\frac{T^{D+1}}{(D+1)^{2D+3}}$$

It's converted to the equation we did in HW02:

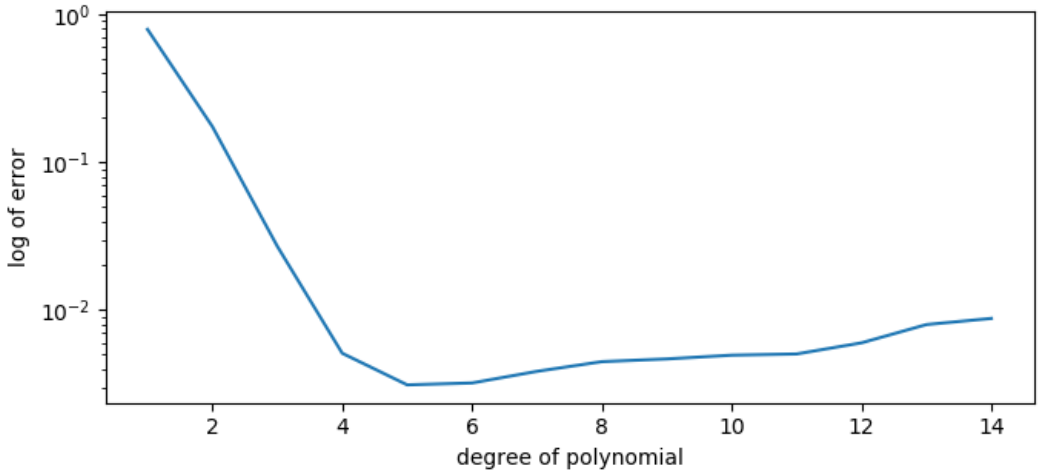
$$\frac{1}{n} \approx \frac{1}{(D+1)^{2D+3}} \Rightarrow \ln n \approx 2(D+1)\ln(D+1)$$

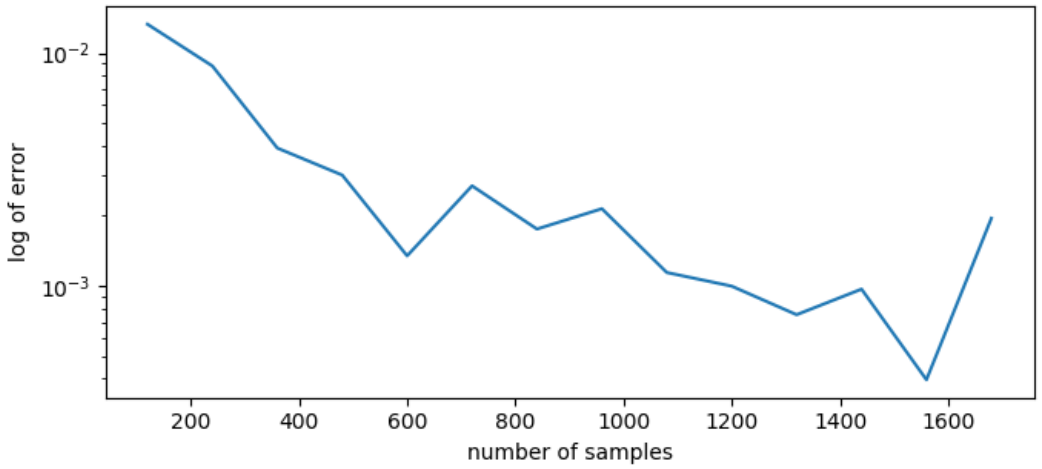
Because we only aim for an approximation, the scaling factor 2 is not that important. Now, let's substitute our answer into the equation above we get:

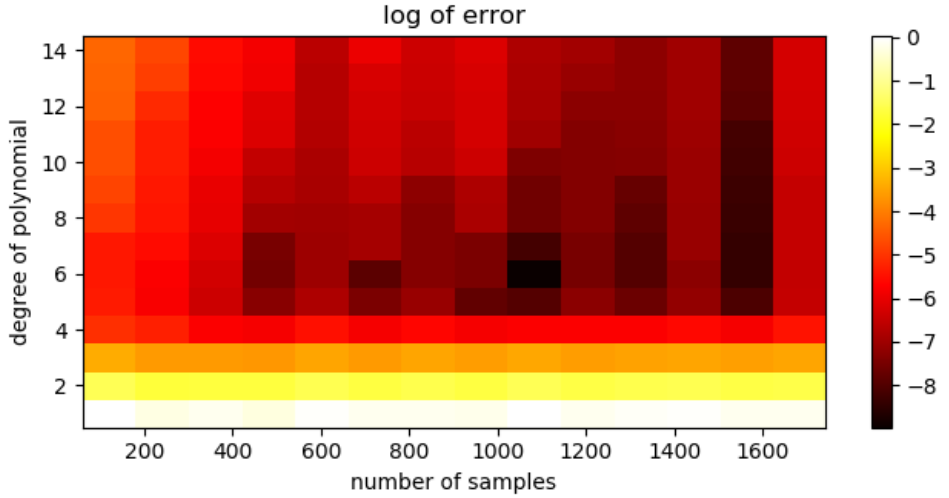
$$\frac{\ln(n)}{\ln\ln(n)}(\ln\ln(n) - \ln\ln\ln(n)) \approx \ln n$$

Therefore, the optimal D is given by $\mathcal{O}(\log n/\log\log n)$.

(g) Simulate the problem in part (f) yourself. Sample $n$ points $\{\alpha_i\}_{i=1}^n$ uniformly from the interval $[-4, 3]$. Generate $y_i = e^{\alpha_i} + z_i$ with $z_i$ representing standard Gaussian noise. **Fit a $D$ degree polynomial to this data and show how the average error $\frac{1}{n}\|\mathbb{X}\hat{\vec{w}} - \vec{y}^*\|_2^2$ scales as a function of both $D$ and $n$.** You may show separate plots for the two scalings. For scaling with $D$, choose $n = 120$. It may also be helpful to average over multiple realizations of the noise (or to plot point clouds) so that you obtain smooth curves. (For this part, the libraries numpy.random and numpy.polyfit might be useful and you are free to use any and all python libraries.)




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

# assign problem parameters
replicates = 10
sample_size_increment = 120
nBox = 15
sample_sizes = np.arange(sample_size_increment, sample_size_increment * nBox, sample_size_increment)
degrees = np.arange(1, nBox)
error = np.zeros((len(degrees), len(sample_sizes)))
```

```
for iRep in range(replicates):
for iN in range(len(sample_sizes)):
# generate data
# np.random might be useful
N = sample_sizes[iN]
alpha = np.random.uniform(-4, 3, N)
Z = np.random.normal(0, 1, N)
Y_true = np.exp(alpha)
Y = Y_true + Z
# fit data with different models
# np.polyfit and np.polyval might be useful
for ideg in range(len(degrees)):
D = degrees[ideg]
w = np.polyfit(alpha, Y, D)
y_predicted = np.polyval(w, alpha)
error[ideg, iN] += np.mean((y_predicted - Y_true) ** 2)

error /= replicates

# plotting figures
# sample code

plt.figure(figsize=(7, 10))
plt.subplot(311)
plt.semilogy(degrees, error[:, 1])
plt.xlabel('degree of polynomial')
plt.ylabel('log of error')

plt.subplot(312)
plt.semilogy(sample_sizes, error[-1, :])
plt.xlabel('number of samples')
plt.ylabel('log of error')

plt.subplot(313)
plt.title('log of error')
plt.imshow(np.log(error), cmap='hot', aspect='auto', interpolation='none', origin='lower',
extent=[sample_sizes[0] - sample_size_increment / 2, sample_sizes[-1] + sample_size_increment / 2,
degrees[0] - 0.5, degrees[-1] + 0.5])
plt.colorbar()
plt.xlabel('number of samples')
plt.ylabel('degree of polynomial')

plt.tight_layout()
plt.show()
```

(h) Comment on the differences in plots obtained in part (e) and part (g).

The underline model in part(e) is a polynomial with degree of one. Therefore, we can have zero (or close to zero) bias when our predictors has a polynomial degree greater than one. Further increase the polynomial degree will only increase the variance. In part(g) the underline more is an exponential function which has infinite polynomial degree. Therefore, we will have a bias variance trade-off. Although increasing degree will result in smaller bias, it will also increase the variance. We can calculate a optimal polynomial degree which gives the smallest expected square error.

**Question 5.** Robotic Learning of Controls from Demonstrations and Images

Huey, a home robot, is learning to retrieve objects from a cupboard, as shown in Fig. 1. The goal is to push obstacle objects out of the way to expose a goal object. Huey's robot trainer, Anne, provides demonstrations via tele-operation. When tele-operating the robot, Anne can look at the images captured by the robot and provide controls to Huey remotely.

During a demonstration, Huey records the RGB images of the scene for each timestep, $x_0, x_1, ..., x_n$, where $x_i \in \mathbb{R}^{30\times30\times3}$ and the controls for his body, $u_0, u_1, \ldots, u_n$, where $u_i \in \mathbb{R}^3$. The controls correspond to making small changes in the 3D pose (i.e. translation and rotation) of his body. Examples of the data are shown in the figure.

Under an assumption (sometimes called the Markovian assumption) that all that matters for the current control is the current image, Huey can try to learn a linear *policy* $\pi$ (where $\pi \in \mathbb{R}^{2700\times3}$) which linearly maps image states to controls (i.e. $\pi^\top x = u$). We will now explore how Huey can recover this policy using linear regression. Note please use **numpy** and **numpy.linalg** to complete this assignment.

Figure 1: A) Huey trying to retrieve a mustard bottle. An example RGB image of the workspace taken from his head mounted camera is shown in the orange box. The angle of the view gives Huey and eye-in-hand perspective of the cupboard he is reaching into. B) A scatter plot of the 3D control vectors, or $u$ labels. Notice that each coordinate of the label lies within the range of $[-1, 1]$ for the change in position. Example images, or states $x$, are shown for some of the corresponding control points. The correspondence is indicated by the blue lines.

(a) To get familiar with the structure of the data, **please visualize the 0th, 10th and 20th images in the training dataset. Also find out what's their corresponding control vectors.**

0th image

10th image

20th image

The 0th control vector is: $\begin{bmatrix} 0. & -1. & 0. \end{bmatrix}$

The 10th control vector is: $\begin{bmatrix} -1. & -0.45111084 & -1. \end{bmatrix}$

The 20th control vector is: $\begin{bmatrix}0. & 0. & 0.37368774\end{bmatrix}$

(b) Load the $n$ training examples from x_train.p and compose the matrix $X$, where $X \in \mathbb{R}^{n \times 2700}$. Note, you will need to flatten the images to reduce them to a single vector. The flattened image vector will be denoted by $\bar{x}$ (where $\bar{x} \in \mathbb{R}^{2700 \times 1}$). Next, load the $n$ examples from y_train.p and compose the matrix $U$, where $U \in \mathbb{R}^{n \times 3}$. Try to perform ordinary least squares to solve:

$$\min_{\pi} \|X\pi - U\|_2^F$$

to learn the *policy* $\pi \in \mathbb{R}^{2700 \times 3}$. **Report what happens as you attempt to do this and explain why.**

Nothing happens. But I get really tiny control vectors with a warning. I guess I'm supposed to get a singular matrix and thus not invertible. However, I use np.linalg.lstsq so somehow this problem doesn't emerge. I think it's because lstsq returns the unique solution using ridge regression when $\lambda \to 0$.

```
FutureWarning: ‘rcond‘ parameter will change to the default of machine precision times ‘‘max(M, N)‘‘ where
    M and N are the input matrix dimensions.
To use the future default and silence this warning we advise to pass ‘rcond=None‘, to keep using the old,
    explicitly pass ‘rcond=-1‘.
pi = np.linalg.lstsq(x_flatten, y_data)[0]
```

$$\begin{bmatrix}[2.60143328e-05 & 6.11372426e-05 & -2.60356071e-05] \\ [3.66994526e-05 & 5.99107123e-05 & -8.31351361e-06] \\ [3.59142889e-05 & 4.87766484e-05 & -3.62659142e-06] \\ & \dots & \\ [8.35502600e-06 & 8.60957100e-05 & 4.31982236e-05] \\ [-1.12220104e-07 & 7.70357948e-05 & 3.92972131e-05] \\ [-6.06919961e-07 & 6.57158733e-05 & 4.78024777e-05]]\end{bmatrix}$$

(c) Now try to perform ridge regression:

$$\min_{\pi} ||X\pi - U||_2^2 + \lambda||\pi||_2^2$$

on the dataset for regularization values $\lambda = \{0.1, 1.0, 10, 100, 1000\}$. Measure the average squared Euclidean distance for the accuracy of the policy on the training data:

$$\frac{1}{n}\sum_{i=0}^{n-1} ||\bar{x}_i^T\pi - u_i||_2^2$$

**Report the training error results for each value of $\lambda$.**

Training error for lambda=0.1 is 1.2567022361125987e-15
Training error for lambda=1.0 is 1.2566924120917558e-13
Training error for lambda=10.0 is 1.2565944002606628e-11
Training error for lambda=100.0 is 1.2555154166320118e-09
Training error for lambda=1000.0 is 1.2459339631806616e-07

(d) Next, we are going to try standardizing the states. For each pixel value in each data point, $x$, perform the following operation:

$$x \mapsto \frac{x}{255} \times 2 - 1.$$

Since we know the maximum pixel value is 255, this rescales the data to be between $[-1, 1]$ . **Repeat the previous part and report the average squared training error for each value of $\lambda$.**

Training error for lambda=0.1 is 3.255747498917072e-07
Training error for lambda=1.0 is 2.9105122907685884e-05
Training error for lambda=10.0 is 0.001590381457303861
Training error for lambda=100.0 is 0.03477312204237575
Training error for lambda=1000.0 is 0.2544029614679703

(e) Evaluate both *policies* (i.e. with and without standardization on the new validation data x_test.p and y_test.p for the different values of $\lambda$. **Report the average squared Euclidean loss** and **qualitatively explain how changing the values of $\lambda$ affects the performance in terms of bias and variance**.

Errors without standardization:
Training error for lambda=0.1 is 0.7740209335126198
Training error for lambda=1.0 is 0.7740206766754301
Training error for lambda=10.0 is 0.7740172564335654
Training error for lambda=100.0 is 0.7739831459262324
Training error for lambda=1000.0 is 0.7736442562269468
Errors with standardization:
Training error for lambda=0.1 is 0.868077068694251
Training error for lambda=1.0 is 0.8621029329745038
Training error for lambda=10.0 is 0.827507615937965
Training error for lambda=100.0 is 0.7246530853296966
Training error for lambda=1000.0 is 0.7250142005123811

As we discussed in the previous questions, when $\lambda$ increases, bias decreases but variance increase. It's so-called bias-variance trade-off.

After standardization, small numbers force $w$ to be large to fit all training data well. This makes $\lambda$ functional and penalize those large weights to avoid over fitting. Without $\lambda$, the bias on test data will be big.

On the other hand, the failure of regularization without standardization can be understood using the condition number in part(e). The condition number without standardization is huge so if we add a relatively small $\lambda$ to those eigenvalues, it won't affect the condition number that much. This means the inverse of the matrix is still dominant by eigenvectors with small eigenvalues. This is usually bad in practice because those eigenvectors might be good to predict data in training set but lack ability to adapt to new test set data.

Basically, not like OLS, ridge regression is not scale free. A more detailed explanation can be found here. (`https://stats.stackexchange.com/questions/111017/question-about-standardizing-in-ridge-regression`).

(f) To better understand how standardizing improved the loss function, we are going to evaluate the *condition number* $\kappa$ of the optimization, which is defined as

$$\kappa = \frac{\sigma_{\max}(X^T X + \lambda I)}{\sigma_{\min}(X^T X + \lambda I)}$$

or the ratio of the maximum singular value to the minimum singular value of the relevant matrix. Roughly speaking, the condition number of the optimization process measures how stable the solution will be when some error exists in the observations. More precisely, given a linear system $Ax = b$, the condition number of the matrix $A$ is the maximum ratio of the relative error in the solution $x$ to the relative error of $b$.

For the regularization value of $\lambda = 100$, **report the condition number with the standardization technique applied and without**.

Condition number without standardization: 52711702.820233904
Condition number with standardization: 444.72593171196445

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


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')

self.x_train_float_flatten = np.asarray(self.x_train, dtype=float).reshape(
(self.x_train.shape[0], int(np.round(self.x_train.size / self.x_train.shape[0]))))
self.y_train_float = np.asarray(self.y_train, dtype=float)
self.x_test_float_flatten = np.asarray(self.x_test, dtype=float).reshape(
(self.x_test.shape[0], int(np.round(self.x_train.size / self.x_train.shape[0]))))
self.y_test_float = np.asarray(self.y_test, dtype=float)
```

```
def plot_image(self):
plt.figure(figsize=(7, 10))
plt.subplot(311)
plt.title('0th image')
plt.imshow(self.x_train[0, :, :, :], aspect='equal', interpolation='none')
plt.axis('off')

plt.subplot(312)
plt.title('10th image')
plt.imshow(self.x_train[10, :, :, :], aspect='equal', interpolation='none')
plt.axis('off')

plt.subplot(313)
plt.title('20th image')
plt.imshow(self.x_train[20, :, :, :], aspect='equal', interpolation='none')
plt.axis('off')

plt.tight_layout()
plt.show()

def print_control(self):
print("The 0th control vector is: ", self.y_train[0, :])
print("The 10th control vector is: ", self.y_train[10, :])
print("The 20th control vector is: ", self.y_train[20, :])

def try_ols(self):
pi = np.linalg.lstsq(self.x_train_float_flatten, self.y_train_float)[0]
print(pi)

def ridge(self, A, b, lambda_):
# Make sure data are centralized
# return np.linalg.inv(A.T.dot(A) + lambda_ * np.eye(A.shape[1])).dot(A.T.dot(b))
return np.linalg.solve(A.T.dot(A) + lambda_ * np.eye(A.shape[1]), A.T.dot(b))

def get_euclidean_error(self, X, w, y):
return np.sum((X.dot(w) - y) ** 2) / y.shape[0]

def get_singular_values(self, A):
_, eig, _ = np.linalg.svd(A)
return eig

LAMBDAs = [0.1, 1.0, 10, 100, 1000]

def try_ridge_5c(self):
errors = np.zeros(len(self.LAMBDAs))
x_train = self.x_train_float_flatten
y_train = self.y_train_float
for i in range(len(self.LAMBDAs)):
pi = self.ridge(x_train, y_train, self.LAMBDAs[i])
errors[i] = self.get_euclidean_error(x_train, pi, y_train)
print("Training error for lambda=%.1f is %s" % (self.LAMBDAs[i], errors[i]))

def try_ridge_5d(self):
errors = np.zeros(len(self.LAMBDAs))
x_train_normalized = self.x_train_float_flatten / 255 * 2 - 1
y_train = self.y_train_float
for i in range(len(self.LAMBDAs)):
pi = self.ridge(x_train_normalized, y_train, self.LAMBDAs[i])
errors[i] = self.get_euclidean_error(x_train_normalized, pi, y_train)
print("Training error for lambda=%.1f is %s" % (self.LAMBDAs[i], errors[i]))

def test_ridge_5e(self):
```

```
errors = np.zeros(len(self.LAMBDAs))
errors_xnormalized = np.zeros(len(self.LAMBDAs))
x_train = self.x_train_float_flatten
x_train_normalized = x_train / 255 * 2 - 1
x_test = self.x_test_float_flatten
x_test_normalized = x_test / 255 * 2 - 1
y_train = self.y_train_float
y_test = self.y_test_float
for i in range(len(self.LAMBDAs)):
pi = self.ridge(x_train, y_train, self.LAMBDAs[i])
pi_normalized = self.ridge(x_train_normalized, y_train, self.LAMBDAs[i])
errors[i] = self.get_euclidean_error(x_test, pi, y_test)
errors_xnormalized[i] = self.get_euclidean_error(x_test_normalized, pi_normalized, y_test)

print("Errors without standardization: ")
for i in range(len(self.LAMBDAs)):
print("Training error for lambda=%.1f is %s" % (self.LAMBDAs[i], errors[i]))

print("Errors with standardization: ")
for i in range(len(self.LAMBDAs)):
print("Training error for lambda=%.1f is %s" % (self.LAMBDAs[i], errors_xnormalized[i]))

def get_condition_number(self):
x_train = self.x_train_float_flatten
x_train_normalized = x_train / 255 * 2 - 1
lmbda = 100
n_feature = x_train.shape[1]
result = self.get_singular_values(x_train.T.dot(x_train) + lmbda * np.eye(n_feature))
print("Condition number without standardization: ", result[0] / result[-1])
result = self.get_singular_values(x_train_normalized.T.dot(x_train_normalized) + lmbda * np.eye(n_feature))
print("Condition number with standardization: ", result[0] / result[-1])


if __name__ == '__main__':
hw3_sol = HW3_Sol()

hw3_sol.load_data()

# Your solution goes here

# plot 0th, 10th, 20th image
hw3_sol.plot_image()
hw3_sol.print_control()

# try OLS
print("---------5b---------")
hw3_sol.try_ols()

# try Ridge
print("---------5c---------")
hw3_sol.try_ridge_5c()

# try Ridge after X normalization
print("---------5d---------")
hw3_sol.try_ridge_5d()

# try Ridge on test data
print("---------5e---------")
hw3_sol.test_ridge_5e()

# get condition number
print("---------5f---------")
hw3_sol.get_condition_number()
```

**Question 6.** Your Own Question

**Write your own question, and provide a thorough solution.**

Writing your own problems is a very important way to really learn the material. The famous "Bloom's Taxonomy" that lists the levels of learning is: Remember, Understand, Apply, Analyze, Evaluate, and Create. Using what you know to create is the top-level. We rarely ask you any HW questions about the lowest level of straight-up remembering, expecting you to be able to do that yourself. (e.g. make yourself flashcards) But we don't want the same to be true about the highest level.

As a practical matter, having some practice at trying to create problems helps you study for exams much better than simply counting on solving existing practice problems. This is because thinking about how to create an interesting problem forces you to really look at the material from the perspective of those who are going to create the exams.

Besides, this is fun. If you want to make a boring problem, go ahead. That is your prerogative. But it is more fun to really engage with the material, discover something interesting, and then come up with a problem that walks others down a journey that lets them share your discovery. You don't have to achieve this every week. But unless you try every week, it probably won't happen ever.

In Q5(b), nothing happens if I use numpy.linalg.lstsq. However, the matrix is supposed to be non-invertible. What happens? To answer this question, I have to get the source code of numpy.linalg.lstsq (`https://github.com/numpy/numpy/blob/v1.13.0/numpy/linalg/linalg.py#L1813-L1981`) I found the following lines:

```
lapack_routine = lapack_lite.dgelsd
...
results = lapack_routine(m, n, n_rhs, a, m, bstar, ldb, s, rcond,
0, work, lwork, iwork, 0)
...
x = array(transpose(bstar)[:n,:], dtype=result_t, copy=True)
...
return wrap(x), wrap(resids), results['rank'], st
```

It looks like numpy.linalg.lstsq uses lapack_lite.dgelsd to solve the least square problem. How does lapack_lite.dgelsd work then? Unfortunately, I cannot find the source code. However, I do find something equally useful (`http://www.netlib.org/lapack/explore-html/df/dc5/group__variants_g_ecomputational_ga3766ea903391b5cf9008132f7440ec7b.html`) It tells me that lapac_lite.dgelsd is basically a function calculating QR decomposition.

Now it's time for Wikipedia (`https://en.wikipedia.org/wiki/QR_decomposition`).

In least square problem, we eventually convert the problem into solving a linear problem $Ax = b$. If we find the QR fractorization of $A^\top = QR$, then we can make the problem easier. A solution to the linear problem is given by

$$x = \begin{bmatrix} (R_1^\top)^{-1} b \\ 0 \end{bmatrix} \quad R = \begin{bmatrix} R_1 \\ 0 \end{bmatrix} \quad \text{Q is an orthogonal matrix}$$

Let's make sure this is correct:

$$Ax = (QR)^\top x = \begin{bmatrix} R_1^\top & 0 \end{bmatrix} Q^\top Q \begin{bmatrix} (R_1^\top)^{-1} b \\ 0 \end{bmatrix} = \begin{bmatrix} R_1^\top & 0 \end{bmatrix} \begin{bmatrix} (R_1^\top)^{-1} b \\ 0 \end{bmatrix} = b$$

It's not exactly $Ax = b$ because the orignal linear problem might have infinite number of solutions if $A$ is singular. However, $R_1^\top$ is a triangular matrix so we can always get a inverse which is also a triangular matrix.

Is this the same as SVD solution? By doing a Google search, I'm able to find an answer (`https://www.math.uci.edu/~chenlong/RNLA/LSQRSVD.pdf`). Basically, QR is more stable but SVD is

faster. However, both give an unique solution so nothing will happen even if our feature matrix is rank-deficient.