CS189

1. (a) By myself. consulting piazza.

(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.

Yao Cao

CS 189 HW2

2. (a) minimize $\|y - Xw\|_2^2$ (1)

subject to $\|w\|_2^2 \le \beta^2$

$$\|w\|_2^2 \le \beta^2 \Rightarrow \|w\|_2^2 - \beta^2 \le 0$$

(1) is equivalent to

$$\boxed{\min_{w} \max_{\lambda \geq 0} \|y - Xw\|_2^2 + \lambda(\|w\|_2^2 - \beta^2) = \min_{w} \max_{\lambda \geq 0} L(w, \lambda)}$$

① the argument is that if $\|w\|_2^2 > \beta^2$, to maximize $L(w, \lambda)$ is to set $\lambda \to \infty$, then the inner maximation will blow up and be ignored by outer minimization;

② if $\|w\|_2^2 \le \beta^2$, and $\lambda \geq 0$ then $\lambda(\|w\|_2^2 - \beta^2) \le 0$

$$\max_{\lambda \geq 0} \lambda(\|w\|_2^2 - \beta^2) = 0$$

$\Rightarrow \max_{\lambda \geq 0} L(w, \lambda) = \|y - Xw\|_2^2$, and the problem becomes

$$\min_{w} \|y - Xw\|_2^2$$

In practice if we choose a certain $\lambda > 0$ then the problem becomes:

$$\min_{w} \left(\|y - Xw\|_2^2 + \lambda(\|w\|_2^2 - \beta^2) \right)$$

$$= \min_{w} \left(\|y - Xw\|_2^2 + \lambda \|w\|_2^2 \right)$$

(b) increasing β is to relax the constraint on w, then it is easier to minimize $\|y - Xw\|_2^2$ closer to its unconstrained minimum, which is equivalent to a $\boxed{\text{smaller } \lambda}$, which means that we decrease our penalty on $\|w\|_2^2$ such that the minimization problem becomes more closer to $\min_w \|y - Xw\|_2^2$ and the minimization is closer to its unconstrained minimum.

(c) prediction without noise $w^T x$

prediction with noise $w^T(x+\varepsilon)$

change $|w^T(x+\varepsilon) - w^T x| = |w^T \varepsilon| \le \|w\|_2 \cdot \|\varepsilon\|_2$

so we want small $\|w\|_2$

(d) $\|y - Xw\|_2^2 + \lambda \|w\|_2^2$

$= (y - Xw)^T (y - Xw) + \lambda w^T w$

$= y^T y + w^T X^T X w - 2y^T X w + \lambda w^T w$

$= y^T y + w^T (X^T X + \lambda I) w - 2y^T X w$

$\nabla_w (y^T y + w^T (X^T X + \lambda I) w - 2y^T X w) = (X^T X + \lambda I + X^T X + \lambda I) w - 2X^T y$

$= 2(X^T X + \lambda I) w - 2X^T y = 0$

$\Rightarrow (X^T X + \lambda I) w = X^T y$

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

when $\lambda \to \infty \Rightarrow X^T X + \lambda I \to \infty \cdot I \Rightarrow (X^T X + \lambda I)^{-1} \to 0 \cdot I$

$\Rightarrow \hat{w} \to 0$

(e) X^TX has eigenvalues $\sigma_1^2, \cdots \sigma_d^2$ with eigenvectors $V_1, V_2, \cdots V_d$

$$X^TXV_i = \sigma_i^2 V_i \qquad (i = 1, \cdots, d)$$

$$\Rightarrow (X^TX + \lambda I)V_i = X^TXV_i + \lambda V_i$$

$$= \sigma_i^2 V_i + \lambda V_i$$

$$= (\sigma_i^2 + \lambda) V_i$$

so the eigenvalues for $X^TX + \lambda I$ is $\sigma_i^2 + \lambda$ $(i = 1, \cdots d)$

if one of the eigenvalues of X^TX, say σ_i^2 is close to 0, then $\det|X^TX| \to 0$ and X^TX is close to non-full rank situation, reverting such matrix will be numerically unstable;
if we add $\lambda > 0$ to σ_i^2, $\det|X^TX + \lambda I|$ won't get close to zero, thus $(X^TX + \lambda I)$ is solidly full rank, and will be easily inverted.

(f) the features for this data set seem to be at very different scale, I would expect the w_i to be at very different scale. In this case, I don't want shrink $\|w\|$ too much since that will make w_i more on similar scales and not able to reflect the huge difference in each feature. So I would choose $\lambda_2 = 0.5$

(g) we can decompose any w into $w_0 + X^T\alpha$ for $w_0 \in null(X)$, $\alpha \in \mathbb{R}^n$

① consider the situation without ridge regression

$$X^TXw = X^Ty$$
$$= X^TX(w_0 + X^T\alpha)$$
$$= X^TXw_0 + X^TXX^T\alpha$$
$$= X^TXX^T\alpha$$

$\Rightarrow XX^Ty = XX^TXX^T\alpha$

$\Rightarrow \alpha = (XX^TXX^T)^{-1}XX^Ty$

but there is no way we can solve w_0.

since there are infinite number of w_0, we have infinite number of solutions for w

② for ridge regression

$$w = (X^TX + \lambda I)^{-1}X^Ty$$

since $X^TX + \lambda I$ is always invertible

we only have one solution for w

(h) $(X^TX + \lambda I)w = X^Ty$

$= (X^TX + \lambda I)(w_0 + X^T\alpha)$

$= \lambda w_0 + X^TXX^T\alpha + \lambda X^T\alpha$

since X^Ty, $X^TXX^T\alpha$, $\lambda X^T\alpha \in col(X^T)$

but $\lambda w_0 \in null(X)$

so $X^Ty = X^TXX^T\alpha + \lambda X^T\alpha$ and $\lambda w_0 = 0$

$= (X^TXX^T + \lambda X^T)\alpha$

$\Rightarrow XX^Ty = (XX^TXX^T + \lambda XX^T)\alpha$

$\alpha = (XX^TXX^T + \lambda XX^T)^{-1}XX^Ty$

even if $\lambda \to 0$, $(XX^TXX^T + \lambda XX^T)$ is invertible.

then we have only one solution for α and only one solution for w_0 ($w_0 = 0$)

so we only have one solution for w.

(i) $\frac{1}{2}\|y - Xw\|_2^2 + \lambda\|\Gamma w\|_2^2$

$= \frac{1}{2}(y - Xw)^T(y - Xw) + \lambda(\Gamma w)^T(\Gamma w)$

$= \frac{1}{2}(y^Ty + w^TX^TXw - 2y^TXw) + \lambda w^T\Gamma^T\Gamma w$

$= \frac{1}{2}y^Ty + w^T(\frac{1}{2}X^TX + \lambda\Gamma^T\Gamma)w - y^TXw$

∇_w: $(\frac{1}{2}X^TX + \lambda\Gamma^T\Gamma + \frac{1}{2}X^TX + \lambda\Gamma^T\Gamma)w - X^Ty$

$= (X^TX + 2\lambda\Gamma^T\Gamma)w - X^Ty = 0$

$\Rightarrow \hat{w} = (X^TX + 2\lambda\Gamma^T\Gamma)^{-1}X^Ty$

3. (f) the highest power is D, and number of variates is L.

We can use D bars to represent power, and L stars to represent variates.

the number of bars put behind each star

$\Rightarrow$ variate corresponding to each star have a power of the number of bars put behind it

the number of bars put ahead of all stars

$\Rightarrow$ 1 to the power of number of bars, which is just one

To illustrate, we use $L=2$, $D=3$ as example:

$$\begin{array}{ll} \times\times & \quad ||| \end{array}$$

$$\begin{array}{lcl}
|||\times\times & \rightarrow & 1^3 x_1^0 x_2^0 \rightarrow 1 \\
||\times|\times & \rightarrow & 1^2 x_1^1 x_2^0 \rightarrow x_1 \\
||\times\times| & \rightarrow & 1^2 x_1^0 x_2^1 \rightarrow x_2 \\
|\times|\times| & \rightarrow & 1^1 x_1 x_2 \rightarrow x_1 x_2 \\
|\times||\times & \rightarrow & 1^1 x_1^2 x_2^0 \rightarrow x_1^2 \\
|\times\times|| & \rightarrow & 1^1 x_1^0 x_2^2 \rightarrow x_2^2 \\
\times|\times|| & \rightarrow & 1^0 x_1^1 x_2^2 \rightarrow x_1 x_2^2 \\
\times||\times| & \rightarrow & 1^0 x_1^2 x_2^1 \rightarrow x_1^2 x_2 \\
\times|||\times & \rightarrow & 1^0 x_1^3 x_2^0 \rightarrow x_1^3 \\
\times\times||| & \rightarrow & 1^0 x_1^0 x_2^3 \rightarrow x_2^3
\end{array}$$

then total number of possibility is the number of ways to arrange D bars and L stars, which is to choose D spots out of $(D+L)$ spots, which is $\binom{D+L}{D} = \binom{D+L}{L}$

$L \geq 2$ (multivariate)

(g) $D \geq 1$ for these data to be meaningful, then

the second column is $(x_{1,1}, x_{2,1}, x_{3,1}, \cdots x_{n,1})^T$

$= (\alpha_1, \alpha_2, \alpha_3 \cdots \alpha_n)^T$

the third column is $(x_{1,2}, x_{2,2}, \cdots x_{n,2})^T$

$= (\alpha_1, \alpha_2, \alpha_3 \cdots \alpha_n)^T$

thus F_L always has linearly dependent columns

This shows that we don't need the other $(L-1)$ variates, we only need to first one, so multvariate falls back to univariate.

In (e), it shows that we don't need two equal data points

These are both redundancies that we can remove from our model

(h) We can sample a $\binom{D+L}{L} \times L$ matrix, where each of the elements are randomly generated. This guarantees that $\vec{x}_i \neq \vec{x}_j$ and $x_{i,p} \neq x_{i,q}$ and then this matrix has full column rank. Then the F_L matrix generated from this matrix is also of full column rank.

4. (a) $(e^x)' = e^x$

$(e^x)^{(i)} = e^x$

1st: $e^x \approx e^{x_0} + e^{x_0}(x - x_0) = 1 + (x-0) = 1 + x$

2nd: $e^x \approx e^{x_0} + e^{x_0}(x - x_0) + \frac{1}{2}e^{x_0}(x-x_0)^2 = 1 + x + \frac{1}{2}x^2$

3rd: $e^x \approx 1 + x + \frac{1}{2}x^2 + \frac{1}{6}e^{x_0}(x-x_0)^3 = 1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3$

4th: $e^x \approx 1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4$

$(\sin x)^{(0)} = \sin x$

$(\sin x)^{(1)} = \cos x$

$(\sin x)^{(2)} = -\sin x$

$(\sin x)^{(3)} = -\cos x$

$(\sin x)^4 = \sin x$

1st: $\sin x \approx \sin x_0 + \cos x_0 (x - x_0) = x$

2nd: $\sin x \approx x + \frac{1}{2}(-\sin x_0)(x - x_0)^2 = x$

3rd: $\sin x \approx x + \frac{1}{6}(-\cos x_0)(x - x_0)^3 = x - \frac{1}{6}x^3$

4th: $\sin x \approx x - \frac{1}{6}x^3 + \frac{1}{24}\sin x_0 (x - x_0)^4 = x - \frac{1}{6}x^3$

(b) $|f(x) - \phi_2(x)|$

$= \left| \frac{1}{6} e^{a(x)} x^3 \right| = \frac{1}{6} e^{a(x)} |x^3|$

when $x = 0$ $\quad |f(x) - \phi_2(x)| = 0$

when $x > 0$ $\quad |f(x) - \phi_2(x)| = \frac{1}{6} e^{a(x)} x^3 \Rightarrow$ monotonically increasing

when $x < 0$ $\quad |f(x) - \phi_2(x)| = -\frac{1}{6} e^{a(x)} x^3 \Rightarrow$ monotonically decreasing

so the sup is at two ends

$|f(3) - \phi_2(3)| = |e^3 - 1 - 3 - \frac{1}{2} \cdot 9| = 11.59$

$|f(-4) - \phi_2(-4)| = |e^{-4} - 1 + 4 - \frac{1}{2} \cdot 16| = 5.98$

so $\|f - \phi_2\|_\infty = 11.59$

- arbitrary m

$$|e^x - \phi_m(x)| = \left| \frac{1}{(m+1)!} e^{a(x)} x^{m+1} \right| \qquad a(x) \in [-4, 3], \quad x \in [-4, 3]$$

$$= \frac{1}{(m+1)!} e^{a(x)} |x|^{m+1}$$

$$= \frac{1}{\sqrt{2\pi(m+1)} \left(\frac{m+1}{e}\right)^{m+1}} e^{a(x)} |x|^{m+1}$$

$$= \frac{e^{a(x)}}{\sqrt{2\pi(m+1)}} \left(\frac{e|x|}{m+1}\right)^{m+1}$$

$$\leq \frac{e^3}{\sqrt{2\pi(m+1)}} \left(\frac{4e}{m+1}\right)^{m+1}$$

when $m \to \infty$, $\frac{1}{\sqrt{m+1}} \to 0$

$$\left(\frac{1}{m+1}\right)^{m+1} \to 0$$

$$|e^x - \phi_m(x)| \leq 0$$

- Outside of I, the approximation error becomes larger than the upper bound derived only for I

(c) $D > O(\log(1/\varepsilon))$

$$(D+1)\log\frac{1}{4e} + (D+\tfrac{1}{2})\log(D+1) > O(\log(1/\varepsilon)) = \log(1/\varepsilon)$$

$$\left(\frac{1}{4e}\right)^{D+1} \times (D+1)^{D+1/2} > 1/\varepsilon$$

$$(4e)^{D+1}\left(\frac{1}{D+1}\right)^{D+1/2} < \varepsilon$$

$$\left(\frac{4e}{D+1}\right)^{D+1}\frac{1}{\sqrt{D+1}} < \varepsilon$$

ignoring constants, we have proved that the upper bound of $|e^x - \phi_D(x)|$ is $< \varepsilon$

so for all $x \in I$, $|e^x - \phi_D(x)| \leq \varepsilon$

for $\sin x$, we have

$$|f(x) - \phi_D(x)| \leq \frac{1}{(D+1)!}\cdot|x|^{D+1} \leq \frac{1}{(D+1)!}4^{D+1} = \frac{1}{\sqrt{2\pi(D+1)}}\left(\frac{4e}{D+1}\right)^{D+1}$$

ignoring the constant, this upper bound error for $\sin x$ is the same as e^x, so the same conclusion can be drawn for $\sin x$

(d) $|f(x) - \phi_D(x)| \leq \frac{1}{(D+1)!}T\cdot|x|^{D+1}$ has the same form as these in (c) if ignoring the constant

so we can always find $D > O(\log(1/\varepsilon))$ to make the sup-norm less than ε

(e) $f(\vec{v}) = f(\vec{v}_0) + (\vec{v}-\vec{v}_0)^T \underbrace{[\nabla_{\vec{v}} f(\vec{v}_0)]}_{①} + \frac{1}{2!}(\vec{v}-\vec{v}_0)^T \underbrace{\nabla_{\vec{v}\vec{v}} f(\vec{v}_0)}_{②}(\vec{v}-\vec{v}_0)$

$$= f(x_0, y_0) + (x-x_0, y-y_0)\underbrace{\begin{pmatrix} f_x(x_0,y_0) \\ f_y(x_0,y_0) \end{pmatrix}}_{①} +$$

$$\frac{1}{2!}(x-x_0, y-y_0)\underbrace{\begin{pmatrix} f_{xx}(x_0,y_0) & f_{xy}(x_0,y_0) \\ f_{yx}(x_0,y_0) & f_{yy}(x_0,y_0) \end{pmatrix}}_{②}\begin{pmatrix} x-x_0 \\ y-y_0 \end{pmatrix}$$

$$= f(x_0,y_0) + f_x(x_0,y_0)(x-x_0) + f_y(x_0,y_0)(y-y_0) +$$

$$\frac{1}{2!}\Big[f_{xx}(x_0,y_0)(x-x_0)^2 + f_{xy}(x_0,y_0)(x-x_0)(y-y_0) +$$
$$f_{yx}(x_0,y_0)(x-x_0)(y-y_0) + f_{yy}(x_0,y_0)(y-y_0)^2\Big]$$

- $f(\vec{v}) = e^x y^2$

$f_x = e^x y^2 \qquad f_{xx} = e^x y^2 \qquad f_{xy} = f_{yx} = 2e^x y$

$f_y = 2e^x y \qquad f_{yy} = 2e^x$

$$e^x y^2 = e^{x_0} y_0^2 + (\vec{v}-\vec{v}_0)^T \begin{pmatrix} e^x y^2 \\ 2e^x y \end{pmatrix} + \frac{1}{2!}(\vec{v}-\vec{v}_0)^T \begin{pmatrix} e^x y^2 & 2e^x y \\ 2e^x y^2 & 2e^x \end{pmatrix}(\vec{v}-\vec{v}_0)$$

(f) suppose $\vec{v} = \begin{pmatrix} x \\ y \end{pmatrix}$

$$g(t) = f(\vec{v}(t)) = f(t\vec{v}) = f(tx, ty) = e^{tx}(ty)^2$$

$$g'(t) = e^{tx} \cdot x\,(ty)^2 + e^{tx} \cdot 2ty^2$$

$$= e^{tx} y^2 (xt^2 + 2t) \qquad g'(t=0) = 0$$

$$g''(t) = e^{tx} \cdot x y^2 (xt^2 + 2t) + e^{tx} \cdot y^2 (2tx + 2) \qquad g''(t=0) = 2y^2$$

$$g(t) \approx g(0) + g'(0)t + \tfrac{1}{2} g''(0)(t)^2$$

$$= \tfrac{1}{2} \cdot 2y^2 t^2 = y^2 t^2$$

5. (a) make a matrix $X \in \mathbb{R}^{n\times(D+1)}$, two vectors $\vec{y} \in \mathbb{R}^{n\times 1}$, $\vec{w} \in \mathbb{R}^{(D+1)\times 1}$

$$X = \begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^D \\ 1 & x_2 & x_2^2 & \cdots & x_2^D \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^D \end{pmatrix} \qquad \vec{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} \qquad \vec{w} = \begin{pmatrix} w_0 \\ \vdots \\ w_D \end{pmatrix}$$

then the problem becomes

$$\arg\min_{\vec{w}} \| X\vec{w} - y \|_2^2$$

(c) the larger D, the smaller training error.
larger D means larger degree of freedom in fitting, thus fits better.

If $D = n$, then $X \in \mathbb{R}^{n\times(n+1)}$ $\quad$ $\text{rank}(X) = n$.

$\text{rank}(X^TX) = \text{rank}(X^T) = n$. $\quad$ $X^TX \in \mathbb{R}^{(n+1)\times(n+1)}$

$X^TX\vec{w} = X^T\vec{y}$

(X^TX) is not full rank and cannot be inverted.

(d) for the same D, generally test error are larger than training error. (Training error → bias, test error → bias + variance)
Training error generally decreases as D gets larger (bias ↓)
test error first decreases and then increases as D gets larger (as D ↑, bias ↓, variance ↑)

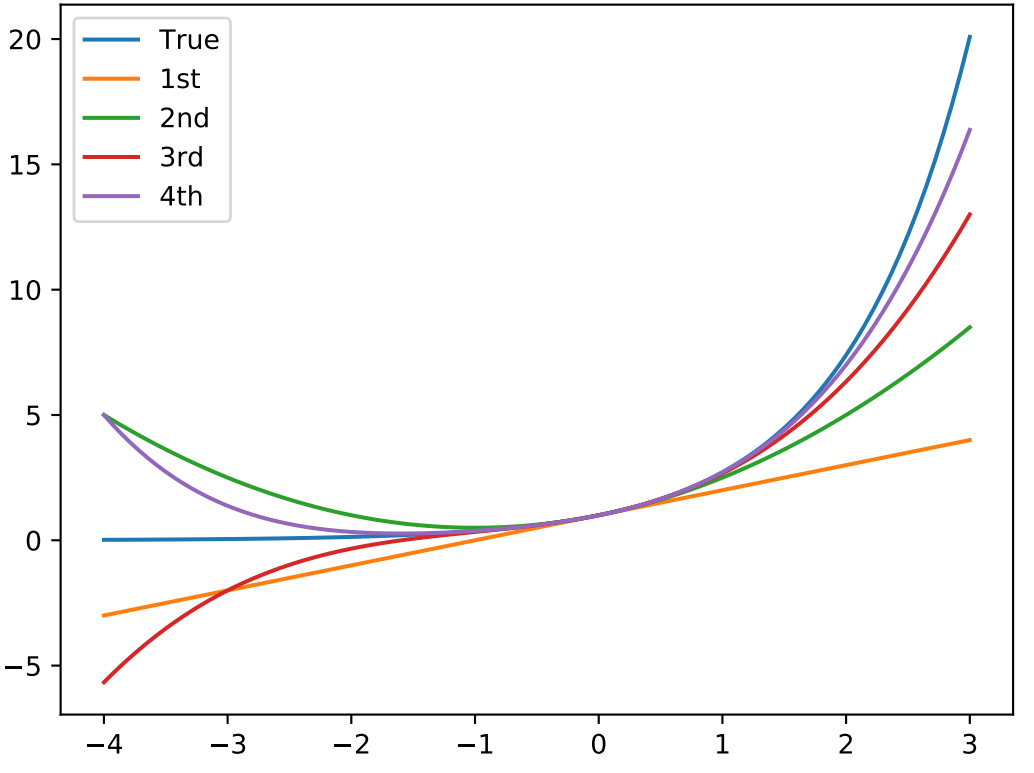
(e) we should choose D of the smallest test error since test error reflects true error, then $D = 4$, this is reasonable since number of training data is $n = 20$.

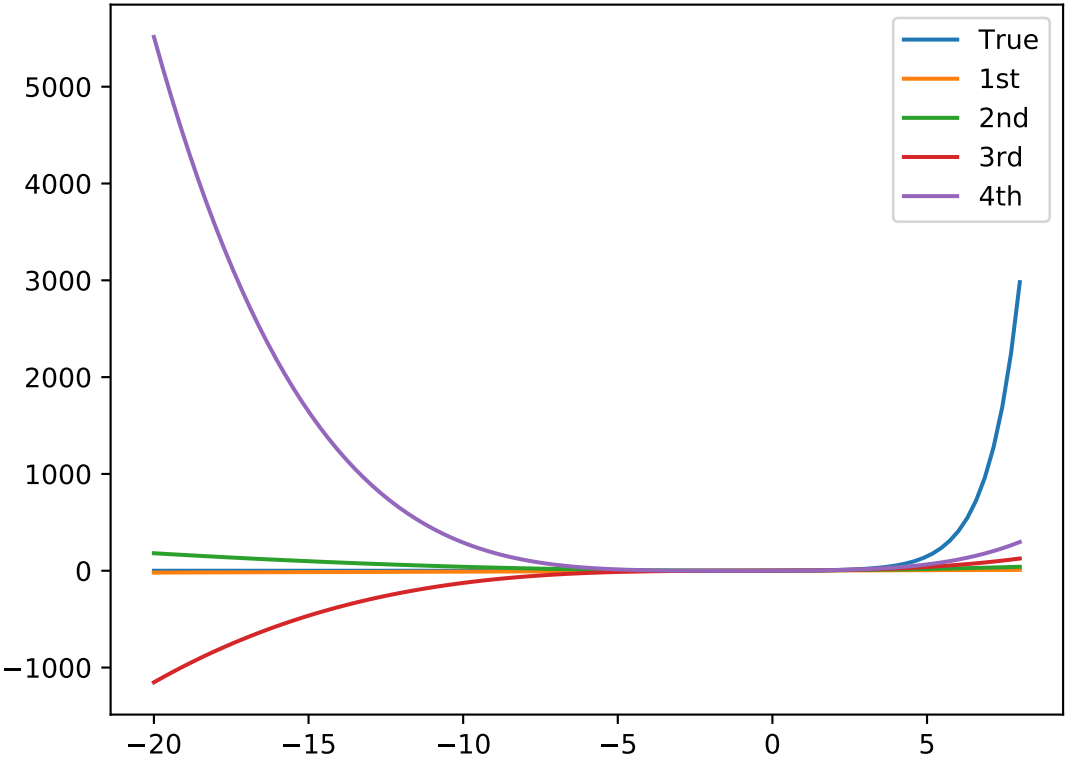
5(f/g)

```
Average train error:
[[ 0.00088367201  0.00088368315  0.00088368317  0.00088367864]
 [ 0.00088307579  0.00088309747  0.00088309963  0.0008831116 ]
 [ 0.00088306822  0.00088307784  0.00088307157  0.00088310166]
 [ 0.00088307738  0.00088305417  0.00088308247  0.00088309341]
 [ 0.00088305615  0.00088307639  0.00088310855  0.00088311197]
 [ 0.0008830751   0.00088306685  0.00088311072  0.00088308146]]

Average valid error:
[[ 0.00153046675  0.00153062035  0.00153067924  0.0015306089 ]
 [ 0.00152964227  0.00152983217  0.00152967147  0.00152986294]
 [ 0.00152969318  0.00152984999  0.00152979887  0.00152994792]
 [ 0.00152986363  0.00152919045  0.00153002294  0.00152981639]
 [ 0.0015297551   0.00152961641  0.00152989859  0.00152979744]
 [ 0.00152995831  0.00152983616  0.0015298718   0.00152932371]]
```

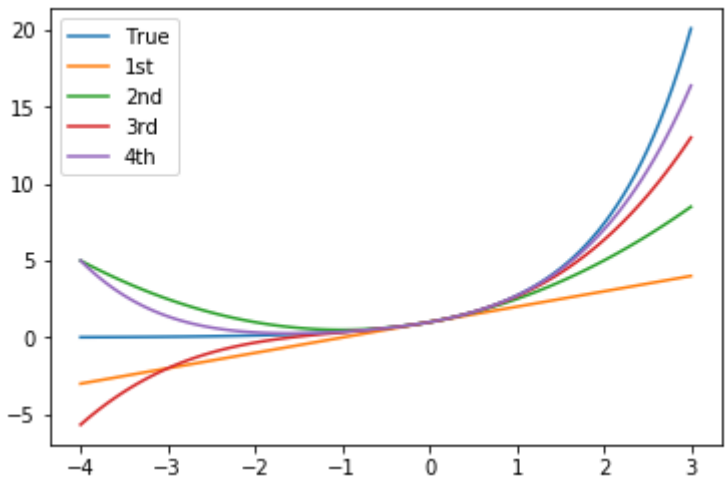
the best D for Lambda 0.1 is D=4, with the smallest validation error 0.0015291904541523491

the best D is 4 and the best Lamdba is 0.1, with the smallest validation error 0.0015291904541523491

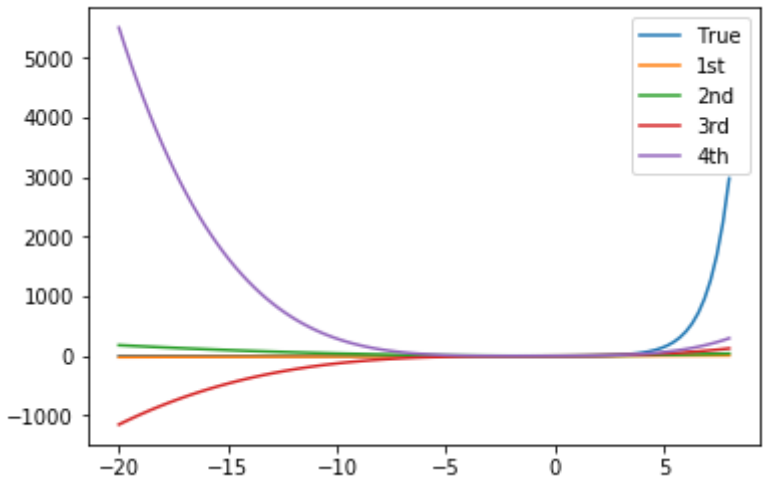

True
1st
2nd
3rd
4th
20
15
10
5
0
−5
−4
−3
−2
−1
0
1
2
3

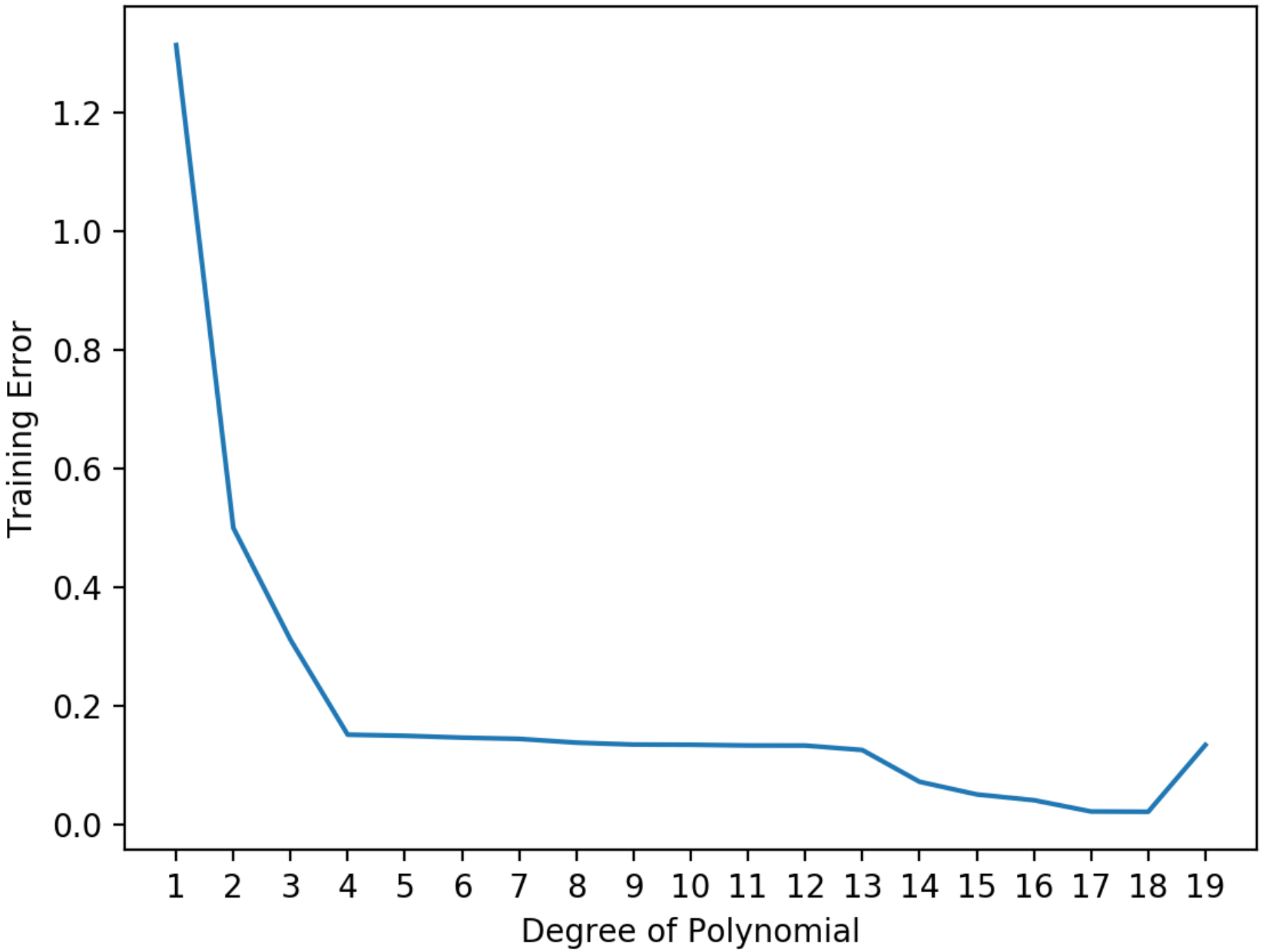

True
1st
2nd
3rd
4th
5000
4000
3000
2000
1000
0
−1000
−20
−15
−10
−5
0
5

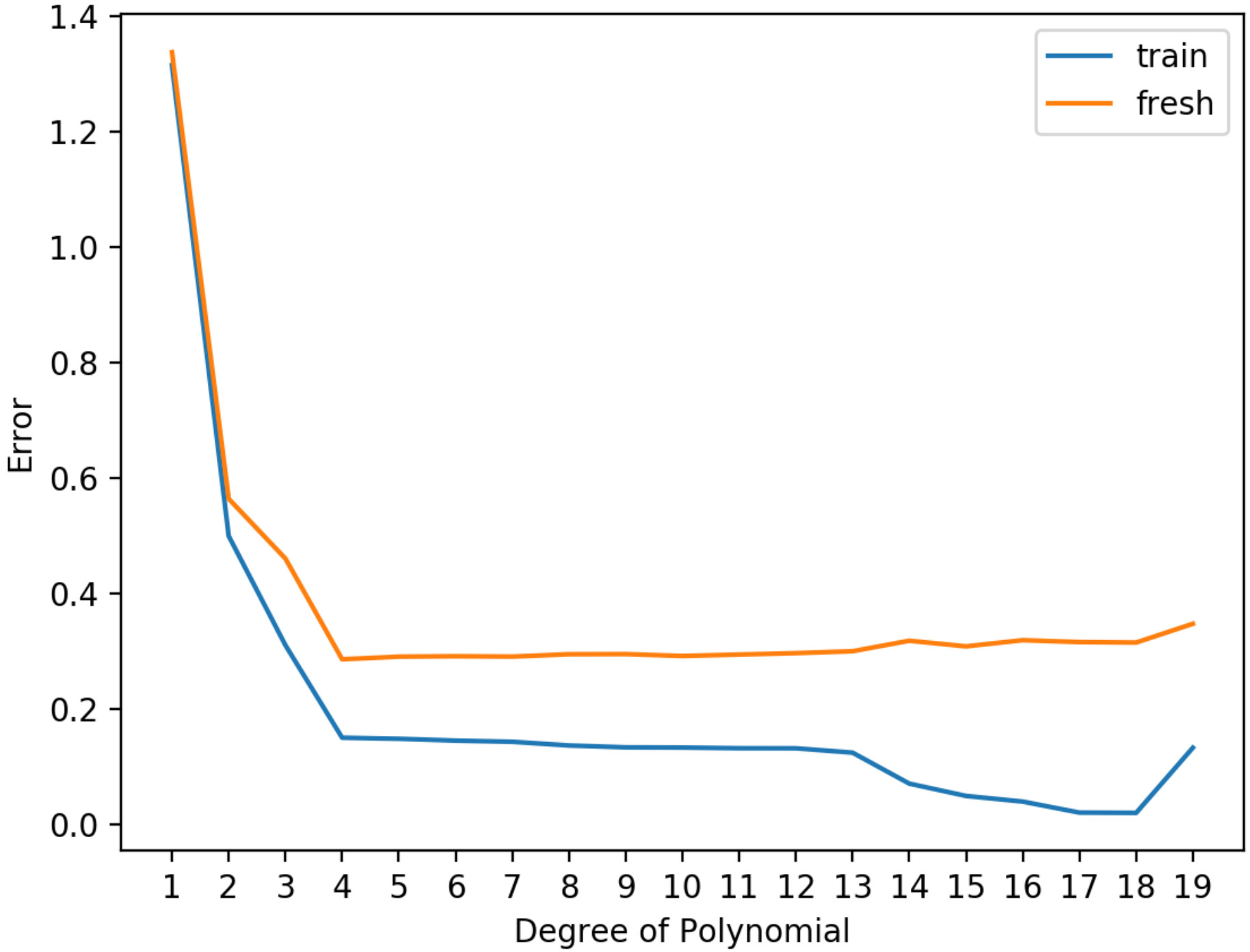
In [29]:
```
import numpy as np
import matplotlib.pyplot as plt


t1 = lambda x : 1+x
t2 = lambda x: 1+x + 1.0/2*x**2
t3 = lambda x : 1+x +1.0/2*x**2+1.0/6*x**3
t4 = lambda x : 1+x +1.0/2*x**2+1.0/6*x**3+1.0/24*x**4

x = np.linspace(-4, 3, 100)
plt.plot(x, np.exp(x), label = 'True')
plt.plot(x, t1(x), label = '1st')
plt.plot(x, t2(x), label = '2nd')
plt.plot(x, t3(x), label = '3rd')
plt.plot(x, t4(x), label = '4th')
plt.legend()
plt.savefig('1.pdf', dpi=100, format='pdf')
plt.show()
```



In [30]:
```
x = np.linspace(-20, 8, 100)
plt.plot(x, np.exp(x), label = 'True')
plt.plot(x, t1(x), label = '1st')
plt.plot(x, t2(x), label = '2nd')
plt.plot(x, t3(x), label = '3rd')
plt.plot(x, t4(x), label = '4th')
plt.legend()
plt.savefig('2.pdf', dpi=100, format='pdf')
plt.show()
```




Training Error
1.2
1.0
0.8
0.6
0.4
0.2
0.0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
Degree of Polynomial


train
fresh
Error
Degree of Polynomial
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

```
##HW2-5-(b)

#!/usr/bin/env python3


import matplotlib.pyplot as plt
import numpy as np
import scipy.io as spio


# There is numpy.linalg.lstsq, whicn you should use outside of this classs
def lstsq(A, b):
    return np.linalg.solve(A.T @ A, A.T @ b)

def get_X(x_train, D):
    n = x_train.shape[0]
    X = np.ones((n, 1))
    for d in range(1, D+1):
        X = np.hstack((X, x_train.reshape((n, 1))**d))
    return X

def main():
    data = spio.loadmat('1D_poly.mat', squeeze_me=True)
    x_train = np.array(data['x_train'])
    y_train = np.array(data['y_train']).T

    n = 20  # max degree
    err = np.zeros(n - 1)

    for D in range(1, n):
        X = get_X(x_train, D)
        w = lstsq(X, y_train)
        err[D-1] = np.linalg.norm(np.dot(X, w)-y_train)/n

    plt.plot(range(1, n), err)
    plt.xlabel('Degree of Polynomial')
    plt.ylabel('Training Error')
    plt.xticks(np.arange(1, n, 1.0))
    plt.show()


if __name__ == "__main__":
    main()
#!/usr/bin/env python3


##HW2-5-(d)

import matplotlib.pyplot as plt
```

```
import numpy as np
import scipy.io as spio


# There is numpy.linalg.lstsq, whicn you should use outside of this classs
def lstsq(A, b):
    return np.linalg.solve(A.T @ A, A.T @ b)



def get_X(x_train, D):
    n = x_train.shape[0]
    X = np.ones((n, 1))
    for d in range(1, D+1):
        X = np.hstack((X, x_train.reshape((n, 1))**d))
    return X

def main():
    data = spio.loadmat('1D_poly.mat', squeeze_me=True)
    x_train = np.array(data['x_train'])
    y_train = np.array(data['y_train']).T
    y_fresh = np.array(data['y_fresh']).T

    n = 20  # max degree
    err_train = np.zeros(n - 1)
    err_fresh = np.zeros(n - 1)

    for D in range(1, n):
        X = get_X(x_train, D)
        w = lstsq(X, y_train)
        err_train[D-1] = np.linalg.norm(np.dot(X, w)-y_train)/n
        err_fresh[D-1] = np.linalg.norm(np.dot(X, w)-y_fresh)/n

    plt.figure()
    plt.plot(range(1, n), err_train, label='train')
    plt.plot(range(1, n), err_fresh, label='fresh')
    plt.legend()
    plt.xlabel('Degree of Polynomial')
    plt.ylabel('Error')
    plt.xticks(np.arange(1, n, 1.0))
    #plt.ylim([0.25, 0.34])
    plt.show()


if __name__ == "__main__":
    main()

# coding: utf-8
```

```
# In[22]:

##HW2-5-(fg)

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import scipy.io as spio

def all_perms(elements):   ##from stack overflow
    if len(elements) <=1:
        yield elements
    else:
        for perm in all_perms(elements[1:]):
            for i in range(len(elements)):
                # nb elements[0:1] works in both string and list contexts
                yield perm[:i] + elements[0:1] + perm[i:]

def all_combination(total, D):
    elements = [i for i in range(total)]
    comb_set = []
    for perm in all_perms(elements):
        comb = np.array(perm) < D
        c_str = ''
        for i in comb:
            if i:
                c_str += '1'
            else:
                c_str += '0'
        if c_str not in comb_set:
            comb_set.append(c_str)
    return comb_set

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

def get_exps(c_str, l):
    exps = np.zeros(l)
    zeros = 0
    for i in c_str:
        if i == '0':
            zeros += 1
        if i == '1' and zeros >= 1:
            exps[zeros-1] += 1
    return exps

def get_X(x_train, D):
    n = x_train.shape[0]
```

```
    l = x_train.shape[1]
    total = D+l
    comb_set = all_combination(total, D)
    X = None
    for j, c_str in enumerate(comb_set):
        exps = get_exps(c_str, l)
        feature = (x_train[:, 0]**exps[0] * x_train[:, 1]**exps[1] *                x_train[:, 2]**exps[2] *
x_train[:, 3]**exps[3] * x_train[:, 4]**exps[4]).reshape((n, 1))
        if X == None:
            X = feature
        else:
            X = np.hstack((X, feature))
    return X


# In[23]:


data = spio.loadmat('polynomial_regression_samples.mat', squeeze_me=True)
data_x = data['x']
data_y = data['y']

Kc = 4  # 4-fold cross validation
KD = 6  # max D = 6
LAMBDA = [0.05, 0.1, 0.15, 0.2]

def get_4_fold(X, y):
    n = X.shape[0]
    index = np.arange(n)
    np.random.shuffle(index)
    index_4 = [index[:n/4], index[n/4:n/2], index[n/2:3*n/4], index[3*n/4:n]]
    train_X = []
    test_X = []
    train_y = []
    test_y = []
    for i in range(4):
        test_index = index_4[i]
        train_index = np.hstack((index_4[(i+1)%4], index_4[(i+2)%4], index_4[(i+3)%4]))
        yield X[train_index], y[train_index], X[test_index], y[test_index]



# In[24]:


def fit(D, lambda_):
    X = get_X(data_x, D)
    err_train_list = []
    err_test_list = []
```

```
    print('4-fold cv: so trained 4 times here:')
    for train_X, train_y, test_X, test_y in get_4_fold(X, data_y):
        print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)
        w = ridge(train_X, train_y, lambda_)
        err_train = np.linalg.norm(np.dot(train_X, w) - train_y)/train_y.shape[0]
        err_test = np.linalg.norm(np.dot(test_X, w) - test_y)/test_y.shape[0]
        err_train_list.append(err_train)
        err_test_list.append(err_test)
    err_train_ave = sum(err_train_list) / float(len(err_train_list))
    err_test_ave = sum(err_test_list) / float(len(err_test_list))
    return err_train_ave, err_test_ave

def best_index(a):
    i,j = np.unravel_index(a.argmin(), a.shape)
    return i, j

def main():
    np.set_printoptions(precision=11)
    Etrain = np.zeros((KD, len(LAMBDA)))
    Evalid = np.zeros((KD, len(LAMBDA)))

    best_D = None
    best_lambda = None

    for D in range(1, KD+1):
        for i in range(len(LAMBDA)):
            print('this is for D = {} and lambda = {}'.format(D, LAMBDA[i]))
            Etrain[D-1, i], Evalid[D-1, i] = fit(D, LAMBDA[i])

    print('Average train error:', Etrain, sep='\n')
    print('Average valid error:', Evalid, sep='\n')

    i, j = best_index(Evalid)
    print('the best D is {} and the best Lamdba is {}, with the smallest validation error {}'.format(
i+1, LAMBDA[j], Evalid[i, j]))

if __name__ == "__main__":
    main()
```