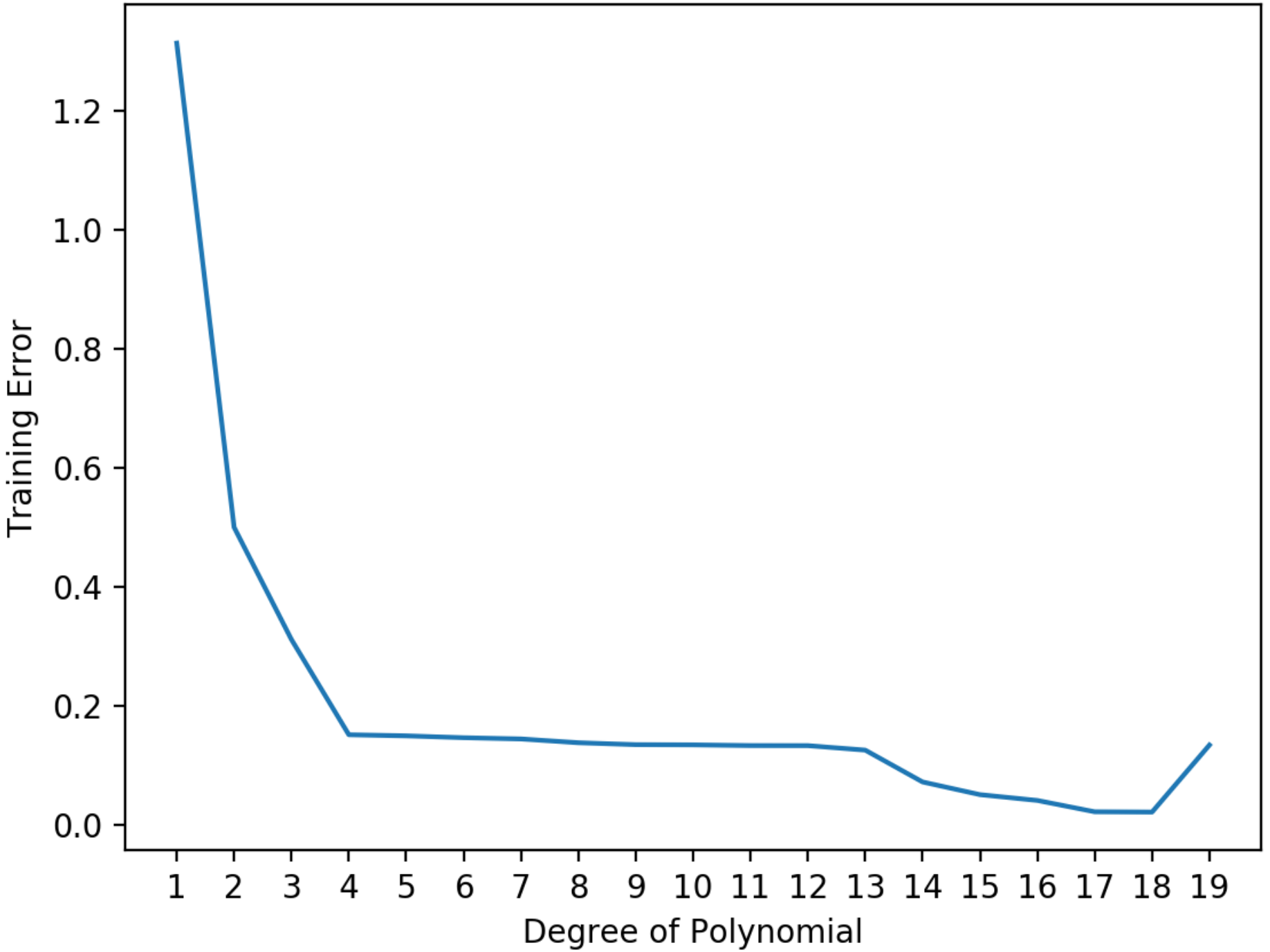
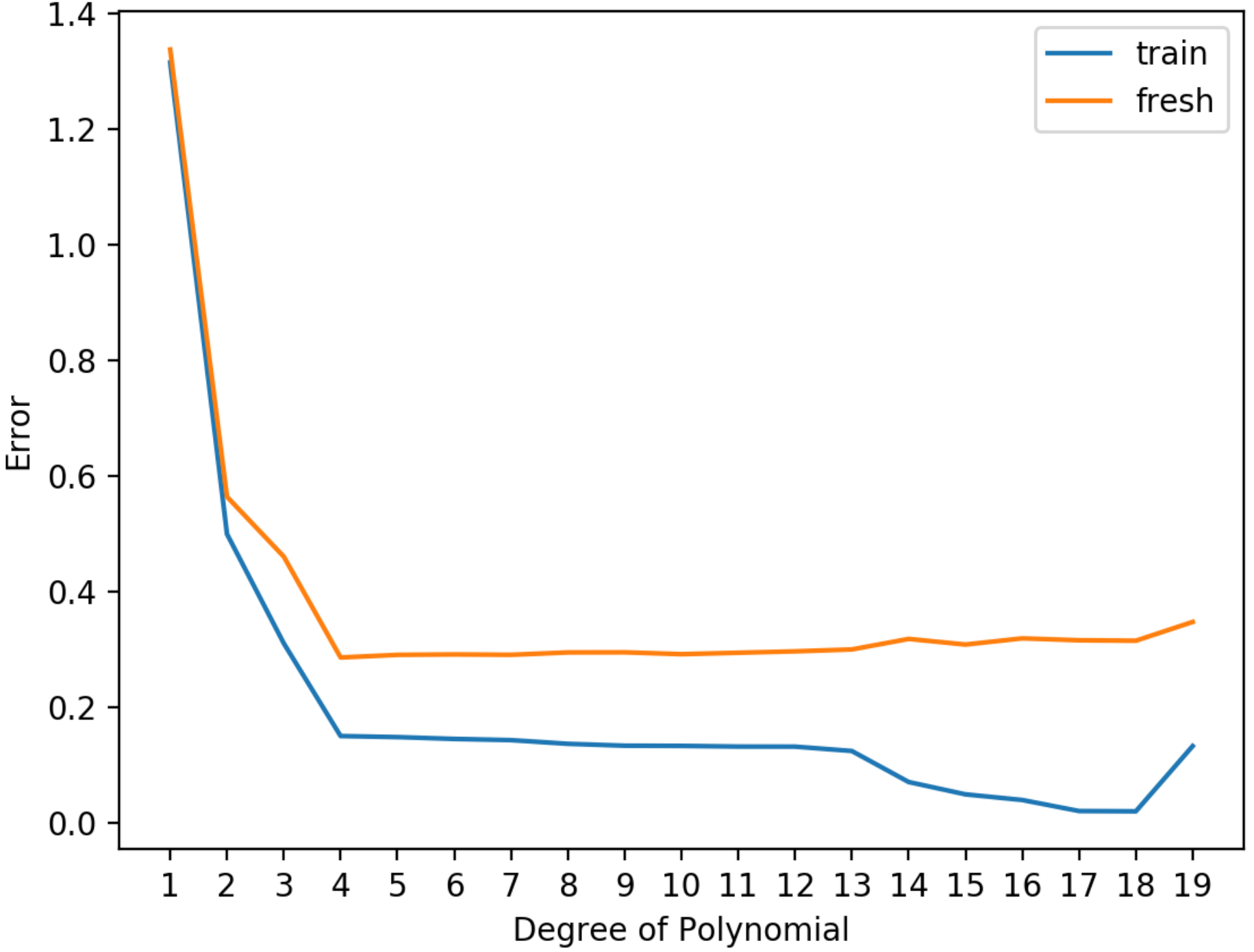

Training Error
1.2
1.0
0.8
0.6
0.4
0.2
0.0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
Degree of Polynomial


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()
```