Ex4.3

In [ ]:
```
# Imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [ ]:
```
dataset = pd.read_csv('ex4_3_train.csv')
dataset2 = pd.read_csv('ex4_3_val.csv')
dataset3 = pd.read_csv('ex4_3_test.csv')
# spilt(x,y)
x_train = dataset.iloc[:,[0]].values
y_train =  dataset.iloc[:,[1]].values
x_val = dataset2.iloc[:,[0]].values
x_test = dataset3.iloc[:,[0]].values
y_val =  dataset2.iloc[:,[1]].values
y_test =  dataset3.iloc[:,[1]].values
x = x_train
x
```

Out[ ]:
```
array([[0.67617],
       [0.1308 ],
       [0.89631],
       [0.84971],
       [0.08061],
       [0.16107],
       [0.20518],
       [0.24813],
       [0.03688],
       [0.79355],
       [0.59199],
       [0.66897],
       [0.72757],
       [0.13881],
       [0.09378],
       [0.28845],
       [0.5713 ],
       [0.78853],
       [0.43217],
       [0.96281],
       [0.25144],
       [0.65929],
       [0.2101 ],
       [0.74713],
       [0.16187],
       [0.70124],
       [0.64351],
       [0.04   ],
       [0.48249],
       [0.13987],
       [0.20824],
       [0.96712],
       [0.77346],
       [0.20269],
       [0.02245],
       [0.64714],
       [0.41949],
       [0.36231],
       [0.72005],
       [0.60114],
       [0.89005],
       [0.0108 ],
       [0.77568],
       [0.91047],
       [0.86028],
       [0.9105 ],
       [0.64219],
       [0.35195],
       [0.62083],
       [0.38099],
       [0.9922 ],
       [0.12921],
       [0.19444],
       [0.84807],
       [0.89209],
       [0.83117],
       [0.68007],
       [0.09324],
       [0.05761]])
```

plot

In [ ]:
```
#  Plotting Functions
```

```
def evaluate_fits(order_list, mse_list):

  fig, ax = plt.subplots()
  ax.bar(order_list, mse_list)
  ax.set(title='Comparing Polynomial Fits', xlabel='Polynomial order', ylabel='MSE')

def plot_fitted_polynomials(x, y, theta_hat):
  x_grid = np.linspace(x.min() - .5, x.max() + .5)

  plt.figure()

  for order in range(0, max_order + 1):
    X_design = make_design_matrix(x_grid, order)
    plt.plot(x_grid, X_design @ theta_hat[order]);

  plt.ylabel('y')
  plt.xlabel('x')
  plt.plot(x, y, 'C0.');
  plt.legend([f'order {o}' for o in range(max_order + 1)], loc=1)
  plt.title('polynomial fits')
  plt.show()
```

$$\mathbf{X} = \left[\mathbf{1}, \mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^k\right],$$

1.make a design matrix

In [ ]:
```
def make_design_matrix(x, order):
 # Atleast shape (n x 1) can matrix work
  if x.ndim == 1:
    x = x[:, None]

  #if x has more than one feature
  # x^0 here
  design_matrix = np.ones((x.shape[0], 1))

  # Loop
  for degree in range(1, order + 1):
      design_matrix = np.hstack((design_matrix, x**degree))

  return design_matrix
```

Bsp

In [ ]:
```
order = 20
X_design = make_design_matrix(x, order)

print(X_design[0:20, 0:20])
```

```
[[1.00000000e+00 6.76170000e-01 4.57205869e-01 3.09148892e-01
  2.09037207e-01 1.41344688e-01 9.55730377e-02 6.46236209e-02
  4.36965537e-02 2.95462987e-02 1.99783208e-02 1.35087412e-02
  9.13420553e-03 6.17627575e-03 4.17621237e-03 2.82382952e-03
  1.90938881e-03 1.29107143e-03 8.72983769e-04 5.90285435e-04]
 [1.00000000e+00 1.30800000e-01 1.71086400e-02 2.23781011e-03
  2.92705563e-04 3.82858876e-05 5.00779410e-06 6.55019468e-07
  8.56765464e-08 1.12064923e-08 1.46580919e-09 1.91727842e-10
  2.50780017e-11 3.28020263e-12 4.29050503e-13 5.61198058e-14
  7.34047060e-15 9.60133555e-16 1.25585469e-16 1.64265793e-17]
 [1.00000000e+00 8.96310000e-01 8.03371616e-01 7.20070013e-01
  6.45405954e-01 5.78483810e-01 5.18500824e-01 4.64737474e-01
  4.16548845e-01 3.73356895e-01 3.34643519e-01 2.99944332e-01
  2.68843104e-01 2.40966763e-01 2.15980919e-01 1.93585858e-01
  1.73512940e-01 1.55521383e-01 1.39395371e-01 1.24941465e-01]
 [1.00000000e+00 8.49710000e-01 7.22007084e-01 6.13496639e-01
  5.21294229e-01 4.42948920e-01 3.76378127e-01 3.19812258e-01
  2.71747674e-01 2.30906716e-01 1.96203745e-01 1.66716285e-01
  1.41660494e-01 1.20370339e-01 1.02279880e-01 8.69082371e-02
  7.38467982e-02 6.27483629e-02 5.33179114e-02 4.53047625e-02]
 [1.00000000e+00 8.06100000e-02 6.49797210e-03 5.23801531e-04
  4.22236414e-05 3.40364773e-06 2.74368044e-07 2.21168080e-08
  1.78283589e-09 1.43714401e-10 1.15848179e-11 9.33852171e-13
  7.52778235e-14 6.06814535e-15 4.89153197e-16 3.94306392e-17
  3.17850383e-18 2.56219193e-19 2.06538292e-20 1.66490517e-21]
 [1.00000000e+00 1.61070000e-01 2.59435449e-02 4.17872678e-03
  6.73067522e-04 1.08410986e-04 1.74617575e-05 2.81256528e-06
  4.53019889e-07 7.29679135e-08 1.17529418e-08 1.89304634e-09
  3.04912974e-10 4.91123328e-11 7.91052344e-12 1.27414801e-12
  2.05227020e-13 3.30559161e-14 5.32431641e-15 8.57587644e-16]
 [1.00000000e+00 2.05180000e-01 4.20988324e-02 8.63783843e-03
  1.77231169e-03 3.63642912e-04 7.46122528e-05 1.53089420e-05
  3.14108872e-06 6.44488585e-07 1.32236168e-07 2.71322169e-08
  5.56698826e-09 1.14223465e-09 2.34363706e-10 4.80867452e-11
  9.86643838e-12 2.02439583e-12 4.15365536e-13 8.52247006e-14]
 [1.00000000e+00 2.48130000e-01 6.15684969e-02 1.52769911e-02
  3.79067981e-03 9.40581381e-04 2.33386458e-04 5.79101819e-05
  1.43692534e-05 3.56544285e-06 8.84693335e-07 2.19518957e-07
  5.44692389e-08 1.35154522e-08 3.35358916e-09 8.32126079e-10
  2.06475444e-10 5.12327519e-11 1.27123827e-11 3.15432353e-12]
 [1.00000000e+00 3.68800000e-02 1.36013440e-03 5.01617567e-05
  1.84996559e-06 6.82267308e-08 2.51620183e-09 9.27975236e-11
  3.42237267e-12 1.26217104e-13 4.65488680e-15 1.71672225e-16
  6.33127166e-18 2.33497299e-19 8.61138038e-21 3.17587709e-22
  1.17126347e-23 4.31961967e-25 1.59307574e-26 5.87526331e-28]
 [1.00000000e+00 7.93550000e-01 6.29721602e-01 4.99715578e-01
  3.96549297e-01 3.14681694e-01 2.49715659e-01 1.98161861e-01
  1.57251345e-01 1.24786805e-01 9.90245688e-02 7.85809465e-02
  6.23579101e-02 4.94841196e-02 3.92681231e-02 3.11612191e-02
  2.47279854e-02 1.96228928e-02 1.55717466e-02 1.23569595e-02]
 [1.00000000e+00 5.91990000e-01 3.50452160e-01 2.07464174e-01
  1.22816717e-01 7.27062680e-02 4.30413836e-02 2.54800687e-02
  1.50839459e-02 8.92954511e-03 5.28620141e-03 3.12937837e-03
  1.85256070e-03 1.09669741e-03 6.49233900e-04 3.84339976e-04
  2.27525423e-04 1.34692775e-04 7.97367758e-05 4.72033739e-05]
 [1.00000000e+00 6.68970000e-01 4.47520861e-01 2.99378030e-01
  2.00274921e-01 1.33977914e-01 8.96272050e-02 5.99579114e-02
  4.01100440e-02 2.68324161e-02 1.79500814e-02 1.20080660e-02
  8.03303588e-03 5.37386001e-03 3.59495113e-03 2.40491446e-03
  1.60881563e-03 1.07624939e-03 7.19978554e-04 4.81644053e-04]
 [1.00000000e+00 7.27570000e-01 5.29358105e-01 3.85145076e-01
  2.80220003e-01 2.03879668e-01 1.48336730e-01 1.07925355e-01
  7.85232502e-02 5.71311612e-02 4.15669189e-02 3.02428432e-02
  2.20037854e-02 1.60092942e-02 1.16478822e-02 8.47464962e-03
```

```
 6.16590082e-03 4.48612446e-03 3.26396957e-03 2.37476634e-03]
[1.00000000e+00 1.38810000e-01 1.92682161e-02 2.67462108e-03
 3.71264152e-04 5.15351769e-05 7.15359790e-06 9.92990925e-07
 1.37837070e-07 1.91331637e-08 2.65587446e-09 3.68661933e-10
 5.11739630e-11 7.10345780e-12 9.86030977e-13 1.36870960e-13
 1.89990580e-14 2.63725923e-15 3.66077954e-16 5.08152808e-17]
[1.00000000e+00 9.37800000e-02 8.79468840e-03 8.24765878e-04
 7.73465441e-05 7.25355890e-06 6.80238754e-07 6.37927903e-08
 5.98248788e-09 5.61037713e-10 5.26141167e-11 4.93415187e-12
 4.62724762e-13 4.33943282e-14 4.06952010e-15 3.81639595e-16
 3.57901612e-17 3.35640132e-18 3.14763316e-19 2.95185037e-20]
[1.00000000e+00 2.88450000e-01 8.32034025e-02 2.40000215e-02
 6.92280619e-03 1.99688344e-03 5.76001030e-04 1.66147497e-04
 4.79252455e-05 1.38240371e-05 3.98754349e-06 1.15020692e-06
 3.31777186e-07 9.57011293e-08 2.76049908e-08 7.96265958e-09
 2.29682916e-09 6.62520370e-10 1.91104001e-10 5.51239490e-11]
[1.00000000e+00 5.71300000e-01 3.26383690e-01 1.86463002e-01
 1.06526313e-01 6.08584827e-02 3.47684512e-02 1.98632161e-02
 1.13478554e-02 6.48302978e-03 3.70375491e-03 2.11595518e-03
 1.20884520e-03 6.90613260e-04 3.94547356e-04 2.25404904e-04
 1.28773822e-04 7.35684844e-05 4.20296751e-05 2.40115534e-05]
[1.00000000e+00 7.88530000e-01 6.21779561e-01 4.90291837e-01
 3.86609822e-01 3.04853443e-01 2.40386086e-01 1.89551640e-01
 1.49467155e-01 1.17859336e-01 9.29356218e-02 7.32825259e-02
 5.77854701e-02 4.55655768e-02 3.59298243e-02 2.83317443e-02
 2.23404303e-02 1.76160995e-02 1.38908230e-02 1.09533306e-02]
[1.00000000e+00 4.32170000e-01 1.86770909e-01 8.07167837e-02
 3.48833724e-02 1.50755471e-02 6.51519917e-03 2.81567363e-03
 1.21684967e-03 5.25885922e-04 2.27272119e-04 9.82201917e-05
 4.24478202e-05 1.83446745e-05 7.92801797e-06 3.42625152e-06
 1.48072312e-06 6.39924111e-07 2.76556003e-07 1.19519208e-07]
[1.00000000e+00 9.62810000e-01 9.27003096e-01 8.92527851e-01
 8.59334740e-01 8.27376081e-01 7.96605965e-01 7.66980189e-01
 7.38456196e-01 7.10993010e-01 6.84551180e-01 6.59092721e-01
 6.34581063e-01 6.10980993e-01 5.88258610e-01 5.66381272e-01
 5.45317553e-01 5.25037193e-01 5.05511060e-01 4.86711104e-01]]
```

This is for least squares

In [ ]:
```
X = x
def ordinary_least_squares(X, y):

 # Compute theta_hat using OLS
  theta_hat = np.linalg.inv(X.T @ X) @ X.T @ y

  return theta_hat


theta_hat = ordinary_least_squares(X, y)
print(theta_hat)
```

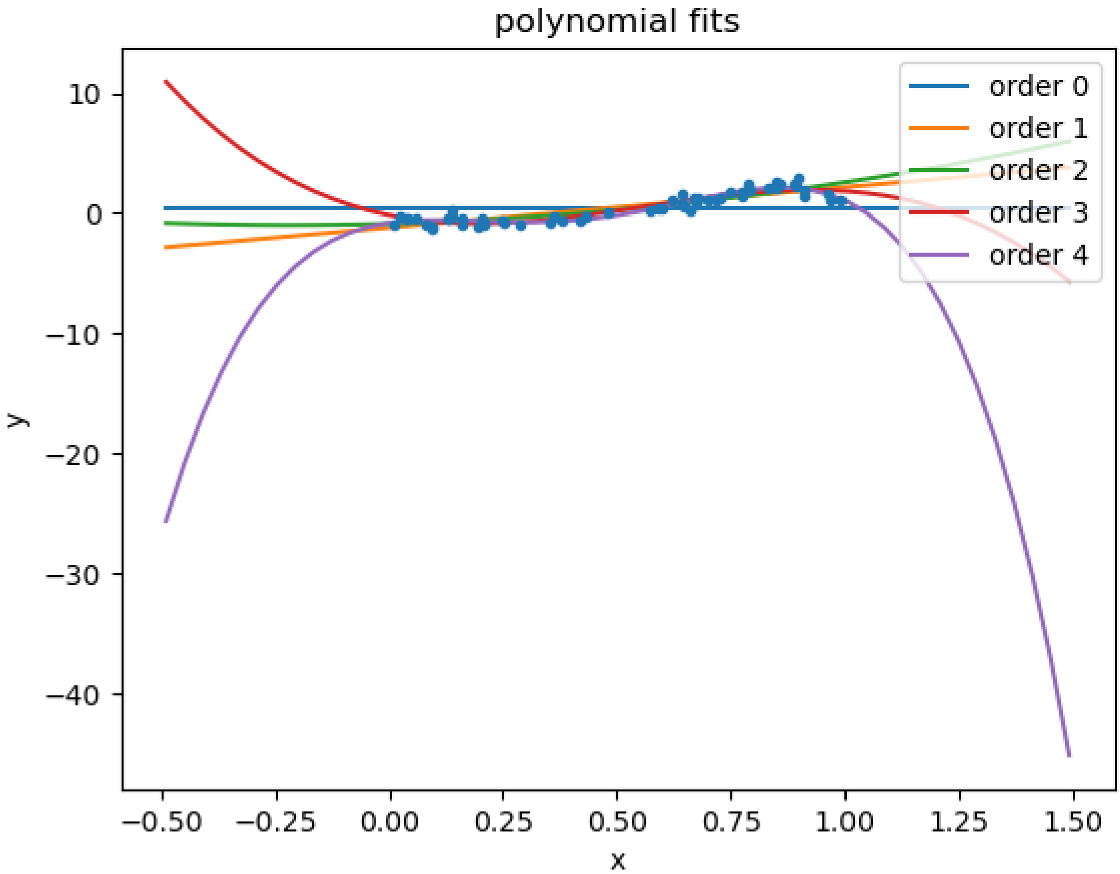
```
[[1.61851916]]
```

2.MLE estimator

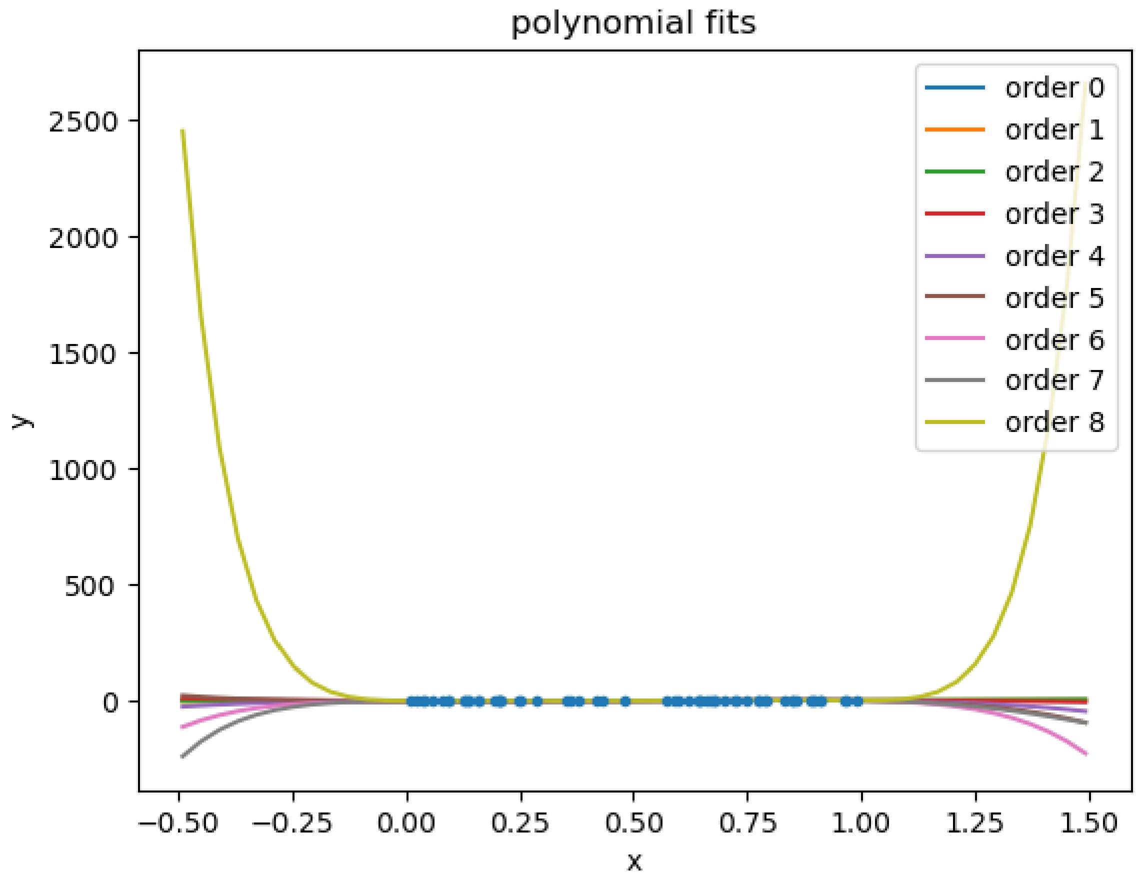
In [ ]:
```
def solve_poly_reg(x, y, max_order):

  # Create a dictionary with polynomial order as keys,
  # and np array of theta_hat (weights) as the values
  theta_hats = {}

  # Loop over polynomial orders from 0 through max_order
  for order in range(max_order + 1):
```

```
    # Create design matrix
    X_design = make_design_matrix(x, order)

    # Fit polynomial model
    this_theta = ordinary_least_squares(X_design, y)

    theta_hats[order] = this_theta

  return theta_hats
```

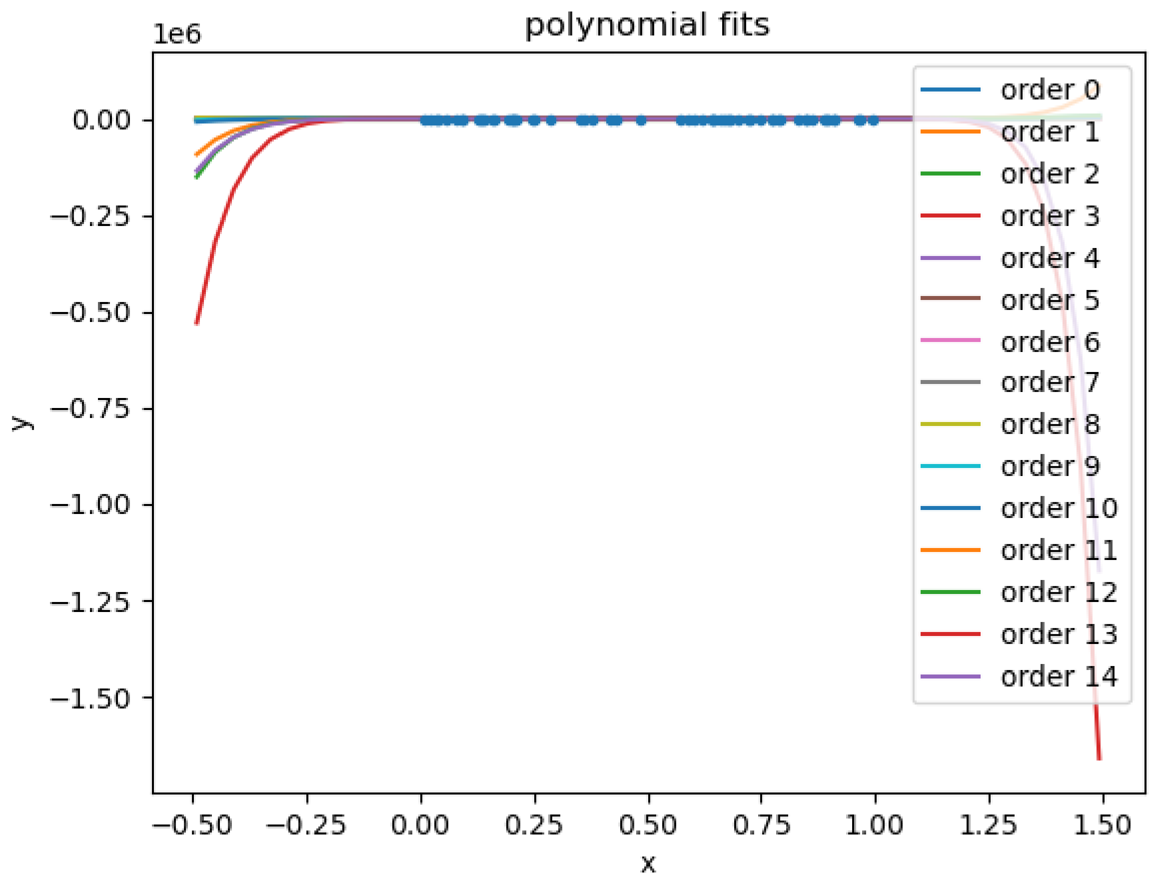
In [ ]:
```
max_order = 4
theta_hats = solve_poly_reg(x, y, max_order)

# Visualize
plot_fitted_polynomials(x, y, theta_hats)
```



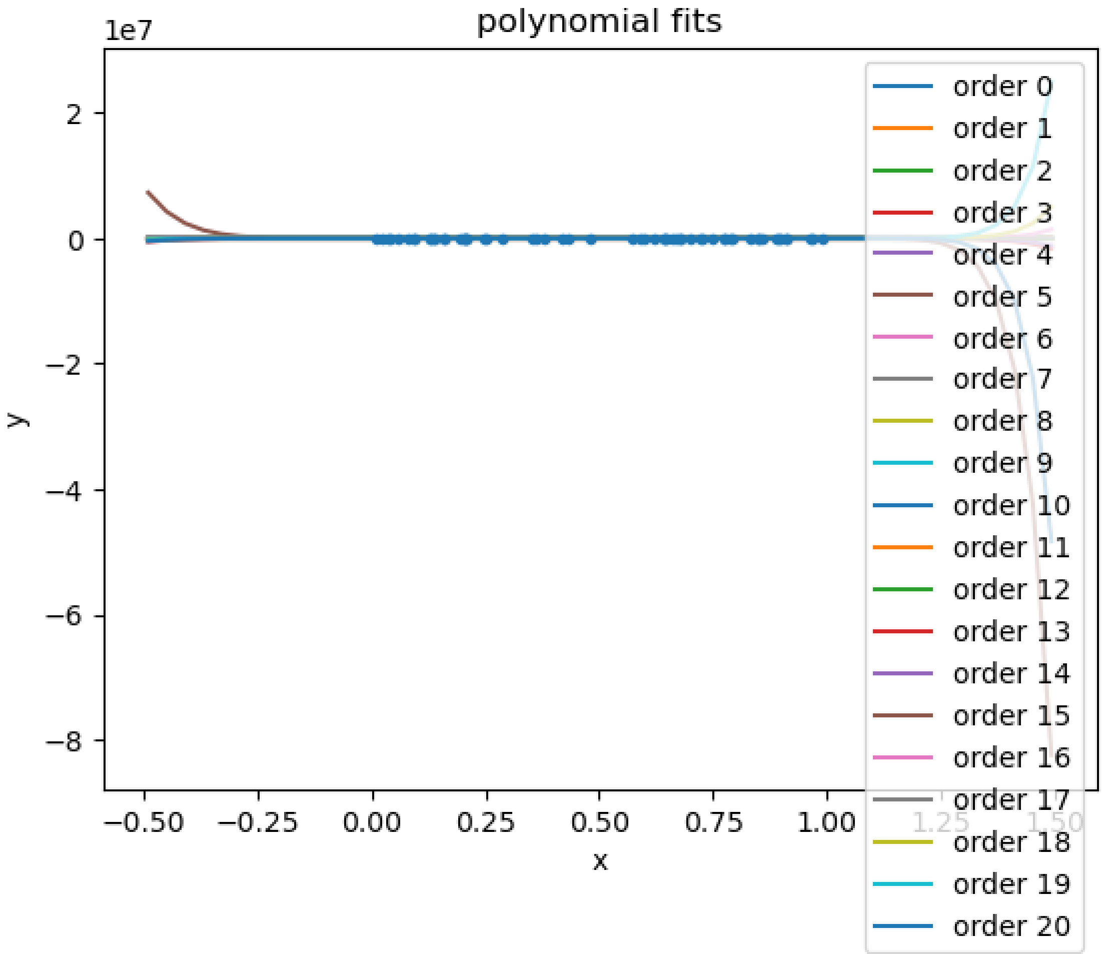
In [ ]:
```
max_order = 8
theta_hats = solve_poly_reg(x, y, max_order)

# Visualize
plot_fitted_polynomials(x, y, theta_hats)
```



In [ ]:
```
max_order = 14
theta_hats = solve_poly_reg(x, y, max_order)

# Visualize
plot_fitted_polynomials(x, y, theta_hats)
```



In [ ]:
```
max_order = 20
theta_hats = solve_poly_reg(x, y, max_order)

# Visualize
plot_fitted_polynomials(x, y, theta_hats)
```

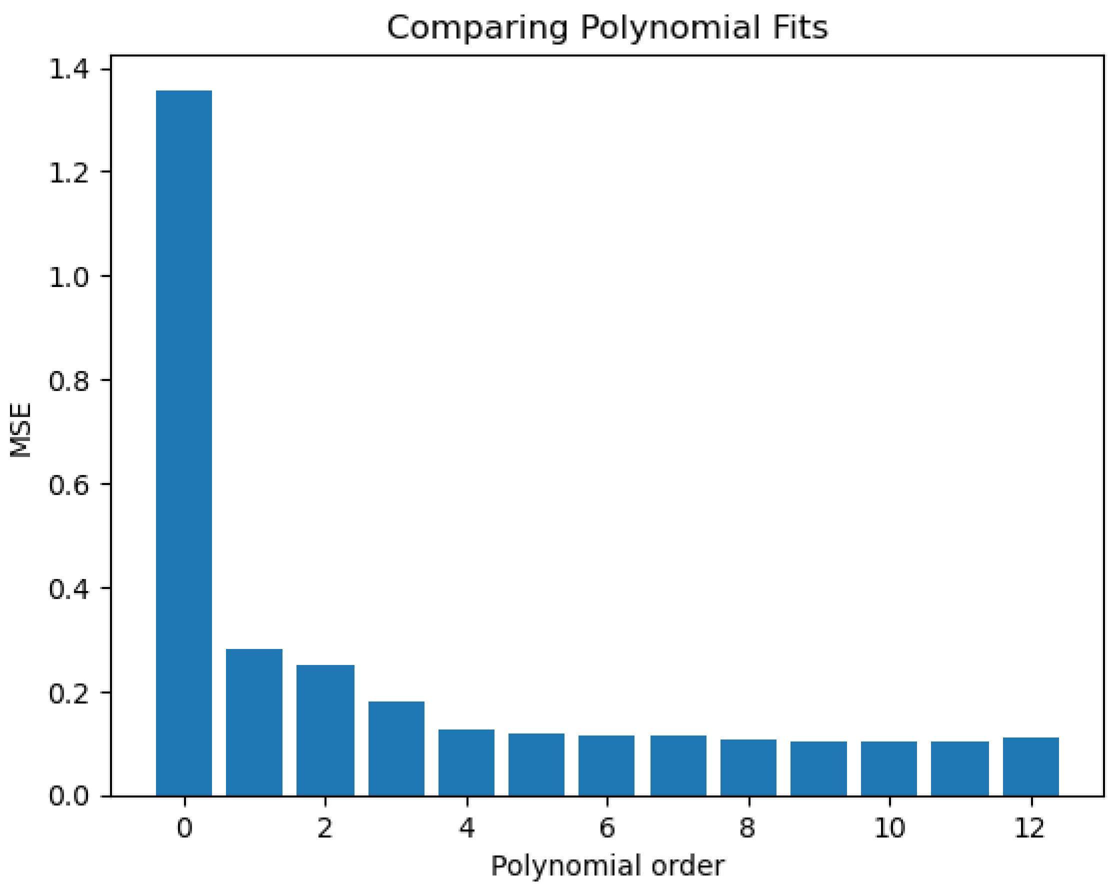


3.MSE

In [ ]:
```
max_order = 12
```

In [ ]:
```
mse_list = []
order_list = list(range(max_order + 1))

for order in order_list:

  X_design = make_design_matrix(x, order)

  # Get prediction for the polynomial regression model of this order
  y_hat = X_design @ theta_hats[order]

  # Compute the residuals
  residuals = y - y_hat

  # Compute the MSE
  mse = np.mean(residuals ** 2)

  mse_list.append(mse)
```

In [ ]:
```
# Visualize MSE of fits
evaluate_fits(order_list, mse_list)
```



1. Sei find the best order 10

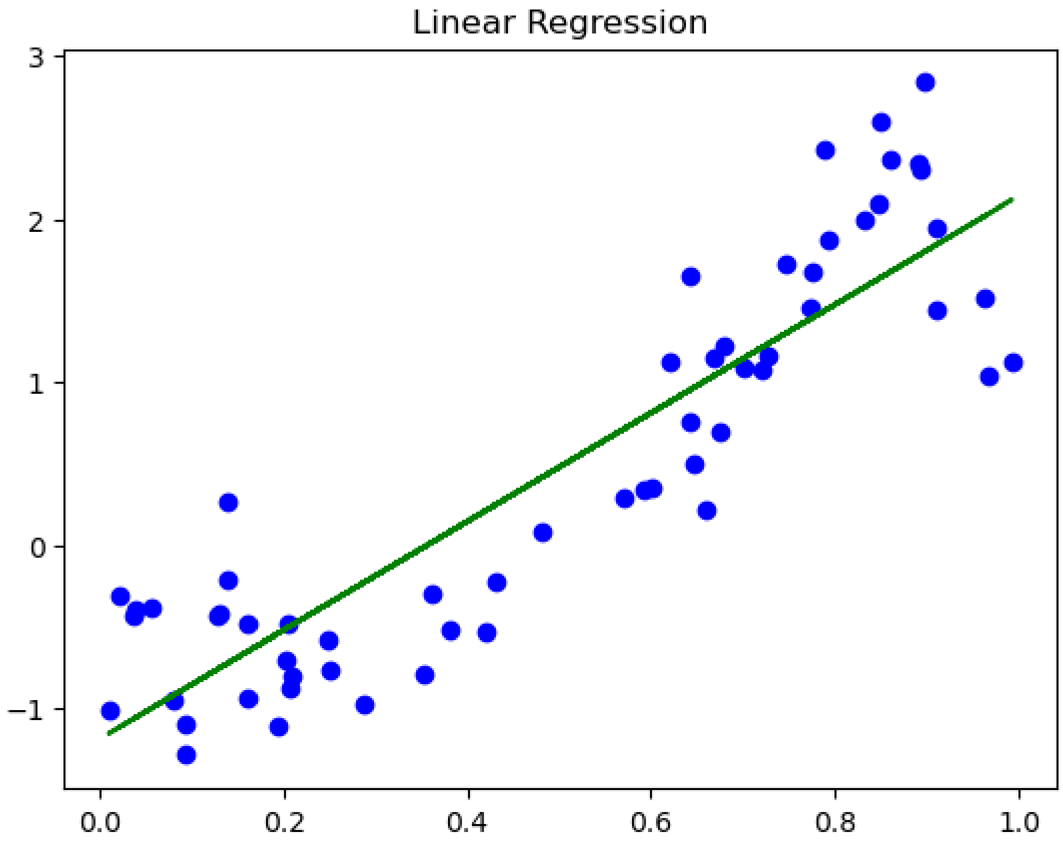
In [ ]:
```
# Fitting Linear Regression to the dataset
from sklearn.linear_model import LinearRegression
lin = LinearRegression()

lin.fit(X, y)
```

Out[ ]:
```
LinearRegression()
```

In [ ]:
```
# Fitting Polynomial Regression to the dataset
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree = 1)
X_poly = poly.fit_transform(X)

poly.fit(X_poly, y)
lin2 = LinearRegression()
lin2.fit(X_poly, y)
```

Out[ ]:
```
LinearRegression()
```

In [ ]:
```
# Visualising the Polynomial Regression results
plt.scatter(X, y, color = 'blue')

plt.plot(X, lin2.predict(poly.fit_transform(X)), color = 'green')
plt.title('Linear Regression')
plt.show()
```



5.Plot the training and test errors against the degree of the polynomial. A paper-pencil plot on squared paper is fine. What do you observe?

You can see whether the model is overfitted by looking at the relationship between the loss values of the training and validation sets with the change of epoch, and if it is, you can stop the training in time and then adjust the model structure and hyperparameters according to the situation, which saves time greatly.

The results obtained after the test set are more accurate

6.Implement the estimator wRIDGE.

In [ ]:

```
from sklearn.linear_model import Ridge
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error


def ridge_demo():
     #4 岭
    estimator = Ridge(alpha=1)  # 1
    # estimator = RidgeCV(alphas=(0.1, 1, 10))  # 2
    estimator.fit(x_train, y_train)

    y_predict = estimator.predict(x_test)
    print("predict为:\n", y_predict)
    print("coef:\n", estimator.coef_)
    print("intercept:\n", estimator.intercept_)

    # .2
    # 均方误差
    error = mean_squared_error(y_test, y_predict)
    print("error:\n", error)
```

```python
if __name__ == '__main__':
    ridge_demo()
```

```
predict为:
 [[ 0.08611446]
 [-0.61103771]
 [ 1.01428889]
 [-0.67416242]
 [-0.38271066]
 [-0.28471841]
 [ 0.40139757]
 [ 1.08890372]
 [ 1.13026814]
 [ 1.11656511]
 [-0.30415231]
 [ 0.99474151]
 [ 1.06470355]
 [ 1.36750358]
 [ 0.64680376]
 [-0.58468137]
 [ 0.22297451]
 [ 1.4754823 ]
 [ 0.41969665]
 [-0.59106477]
 [ 1.57165883]
 [ 0.77651441]
 [ 0.91873652]
 [-0.83454175]
 [ 0.03955821]
 [ 0.22195317]
 [ 0.9643849 ]
 [ 0.66042168]
 [ 1.79422664]]
coef:
 [[2.83706576]]
intercept:
 [-0.94098846]
error:
 0.23403165723376682
```

7.Find a good combination of d and   that gives you a small validation error. Is the test error smaller than the test error of the optimal solution from (d)?

I think they are both similar in this question

Because there are no errors or omissions in the data there is no need to add