Exercises week 35 - Markus Bjørklund

September 1, 2023

0.1 Exercise 1

0.1.1 Exercise 1.1

Assuming the vectors $\mathbf{a}$ and $\mathbf{b}$ are column vectors $n \times 1$ and $m \times 1$, the product only makes sense for a scalar result when $n = m$. So, we have

$$\alpha = \mathbf{b}^T \mathbf{a}, \tag{1}$$

where α is a scalar. From the lecture notes from week 35 (example 4) we can obtain the more general expression

$$\frac{\partial \alpha}{\partial \mathbf{z}} = \mathbf{a}^T \frac{\partial \mathbf{b}}{\partial \mathbf{z}} + \mathbf{b}^T \frac{\partial \mathbf{a}}{\partial \mathbf{z}} \tag{2}$$

In our case, $\mathbf{z}$ is equal to $\mathbf{a}$, so

$$\begin{align} \frac{\partial \alpha}{\partial \mathbf{a}} &= \mathbf{a}^T \frac{\partial \mathbf{b}}{\partial \mathbf{a}} + \mathbf{b}^T \frac{\partial \mathbf{a}}{\partial \mathbf{a}} \tag{3} \\ &= 0 + \mathbf{b}^T I \tag{4} \\ &= \mathbf{b}^T, \tag{5} \end{align}$$

where $\mathbf{I}$ is the $n \times n$ identity matrix for all vectors $\mathbf{a}$. I realise this is the transpose of what's given in the exercise text, but I hope this is a matter of numerator or denominator layout. I *think* the the result

$$\frac{\partial \left(\mathbf{b}^T \mathbf{a}\right)}{\partial \mathbf{a}} = \mathbf{b} \tag{6}$$

is true for denominator layout, while the formula given in the lecture notes

$$\frac{\partial \alpha}{\partial \mathbf{z}} = \mathbf{a}^T \frac{\partial \mathbf{b}}{\partial \mathbf{z}} + \mathbf{b}^T \frac{\partial \mathbf{a}}{\partial \mathbf{z}} \tag{7}$$

is true for numerator layout. For clarification, we can try another approach. The partial derivative of a scalar α by a vector component a_k is given by

$$\frac{\partial \alpha}{\partial a_k} = \sum_{i=0}^{n-1} \left(a_i \frac{\partial b_i}{\partial a_k} + b_i \frac{\partial a_i}{\partial a_k} \right) \tag{8}$$

$$= \sum_{i=0}^{n-1} b_i \frac{\partial a_i}{\partial a_k} \tag{9}$$

$$= \sum_{i=0}^{n-1} b_i \delta_{ik} \tag{10}$$

$$= b_k \tag{11}$$

Now the derivative of a scalar by a vector using numerator layout gives a row vector

$$\frac{\partial \alpha}{\partial \mathbf{a}} = \begin{bmatrix} \frac{\partial \alpha}{\partial a_0} & \frac{\partial \alpha}{\partial a_1} & \cdots & \frac{\partial \alpha}{\partial a_{n-1}} \end{bmatrix} = \begin{bmatrix} b_0 & b_1 & \cdots & b_{n-1} \end{bmatrix} = \mathbf{b}^T, \tag{12}$$

while for denominator layout we get a column vector

$$\frac{\partial \alpha}{\partial \mathbf{a}} = \begin{bmatrix} \frac{\partial \alpha}{\partial a_0} \\ \frac{\partial \alpha}{\partial a_1} \\ \vdots \\ \frac{\partial \alpha}{\partial a_{n-1}} \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n-1} \end{bmatrix} = \mathbf{b}, \tag{13}$$

0.1.2 Exercise 1.2

To avoid confusion with notation, I am going to use $\mathbf{x}$ instead of $\mathbf{a}$.

$$\alpha = \mathbf{x}^T \mathbf{A} \mathbf{x} = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} x_i a_{ij} x_j \tag{14}$$

α is a scalar, given $m \times 1$ dimensions on $\mathbf{x}$ and $m \times m$ dimensions for matrix $\mathbf{A}$. Then, for component k, assuming $\mathbf{A}$ does not depend on $\mathbf{x}$, we have

$$\frac{\partial \alpha}{\partial x_k} = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_{ij} \frac{\partial}{\partial x_k} x_i x_j \tag{15}$$

$$= \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \underbrace{a_{ij} x_i \frac{\partial x_j}{\partial x_k}}_{\text{Only j=k survives}} + \underbrace{a_{ij} x_j \frac{\partial x_i}{\partial x_k}}_{\text{Only i=k survives}} \tag{16}$$

$$= \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_{ik} x_i + a_{kj} x_j \tag{17}$$

$$= \sum_{i=0}^{n-1} a_{ik} x_i + \sum_{j=0}^{n-1} a_{kj} x_j \tag{18}$$

Since there are no shared dummy (summation) indeces between the two terms, and they are equal in limits, we can simply rename to a single sum

$$\frac{\partial \alpha}{\partial x_k} = \sum_{l=0}^{n-1} a_{lk} x_l + a_{kl} x_l \tag{19}$$

$$= \sum_{l=0}^{n-1} x_l \left(a_{lk} + a_{kl}\right) \tag{20}$$

Which for all components of x_k we recognize as

$$\frac{\partial\left(\mathbf{x}^T \mathbf{A} \mathbf{x}\right)}{\partial \mathbf{x}} = \mathbf{x}^T \left(\mathbf{A} + \mathbf{A}^T\right) \tag{21}$$

0.1.3 Exercise 1.3

Let us first define a vector $\mathbf{u} = \mathbf{x} - \mathbf{A}\mathbf{s}$. Now, by the chain rule we get that

$$\frac{\partial \mathbf{u}^T \mathbf{u}}{\partial \mathbf{s}} = 2\mathbf{u}^T \frac{\partial \mathbf{u}}{\partial \mathbf{s}} \tag{22}$$

Now, we get that

$$\frac{\partial \mathbf{u}}{\partial \mathbf{s}} = -\mathbf{A}, \tag{23}$$

assuming that $\mathbf{x}$ and $\mathbf{A}$ are independent of $\mathbf{s}$. Inserting back, we get

$$\frac{\partial \mathbf{u}^T \mathbf{u}}{\partial \mathbf{s}} = -2\mathbf{u}^T A, \tag{24}$$

or expanding $\mathbf{u}$,

$$\frac{\partial \left(\mathbf{x} - \mathbf{A}\mathbf{s}\right)^T \left(\mathbf{x} - \mathbf{A}\mathbf{s}\right)}{\partial \mathbf{s}} = -2\left(\mathbf{x} - \mathbf{A}\mathbf{s}\right)^T \mathbf{A} \tag{25}$$

0.1.4 Exercise 1.4

We already established that

$$\frac{\partial \mathbf{u}}{\partial \mathbf{s}} = -\mathbf{A}, \tag{26}$$

so we have that

$$\frac{\partial \mathbf{u}^T}{\partial \mathbf{s}} = -\mathbf{A}^T. \tag{27}$$

Inserting this result gives us that

$$\frac{\partial^2 \left(\mathbf{x} - \mathbf{A}\mathbf{s}\right)^T \left(\mathbf{x} - \mathbf{A}\mathbf{s}\right)}{\partial \mathbf{s}^2} = 2\mathbf{A}^T\mathbf{A} \tag{28}$$

0.2 Exercise 2

0.2.1 Exercise 2.1

[1]:
```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(1)

x = np.random.rand(100)
noise_coeff1 = 0.1
y = 2.0 + 5*x**2 + noise_coeff1 * np.random.randn(100)

p = 3 #Number of predictors (polynomial order + 1 (the intercept))

X = np.zeros((len(x), p)) #Design matrix
X[:,0] = 1
X[:,1] = x
X[:,2] = x**2
```

Now solving for beta according to

$$\beta = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y} \tag{29}$$

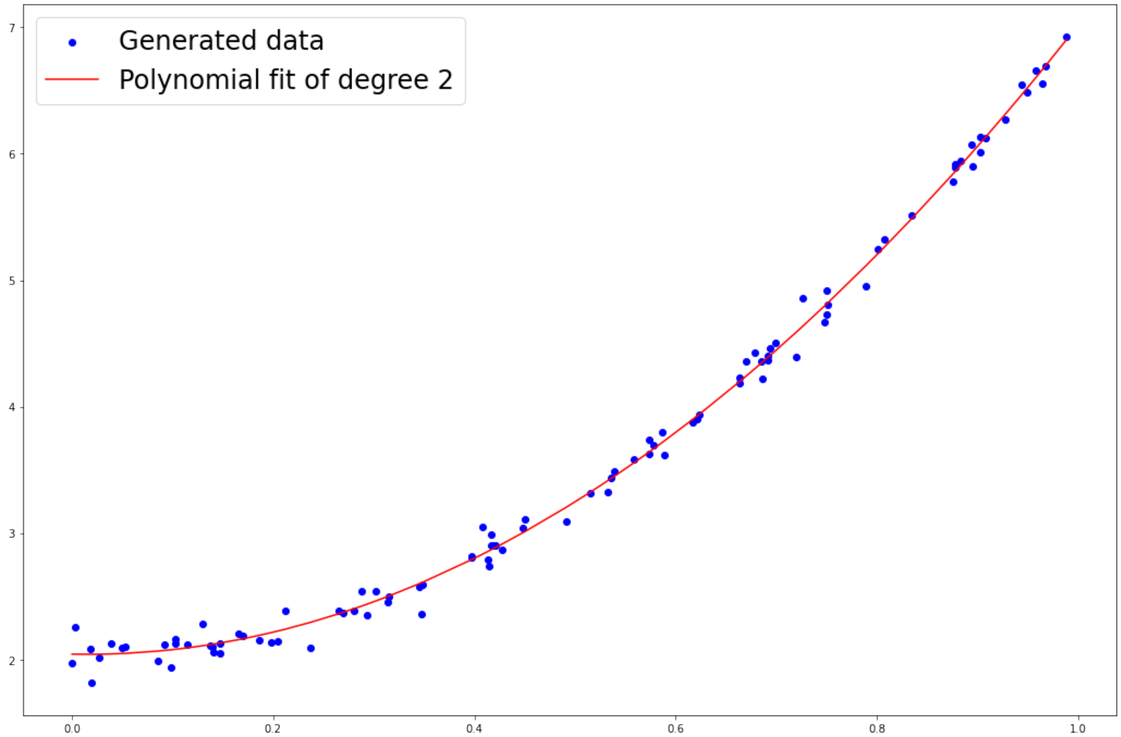
And obtaining the approximation for $\mathbf{y}$, $\tilde{\mathbf{y}}$, by

$$\tilde{\mathbf{y}} = \mathbf{X}\beta \tag{30}$$

[2]:
```
beta = (np.linalg.inv(X.T @ X) @ X.T ) @ y
y_tilde = X @ beta

print(beta)

fig = plt.figure(figsize=(18,12))
ax = fig.add_subplot(111)
ax.scatter(x,y, label="Generated data", color="blue")

ind = np.argsort(x) #Sorted indeces for plotting fit as a smooth line
ax.plot(x[ind],y_tilde[ind], label="Polynomial fit of degree {}".format(p-1),␣
 ↪color="red")
ax.legend(fontsize=24)
```

```
[ 2.04321701 -0.1571625   5.12868091]
```

[2]: <matplotlib.legend.Legend at 0x29d5cad6cd0>

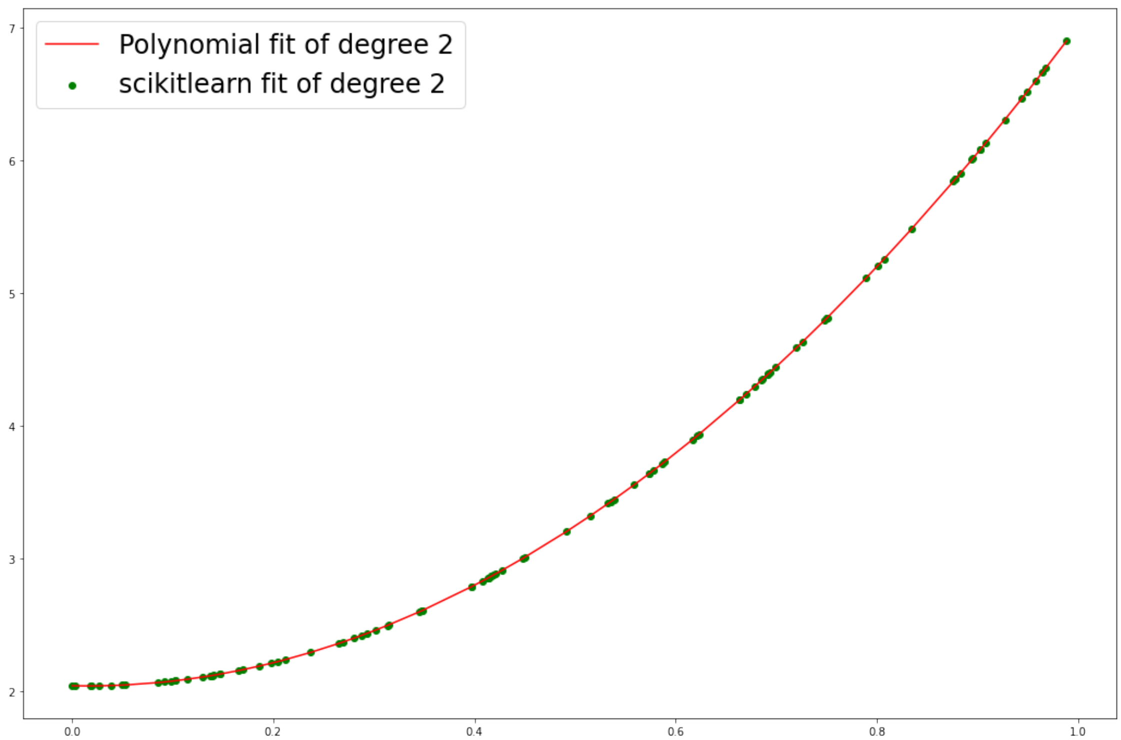


0.2.2 Exercise 2.2

[3]:
```
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures

#Scikit learn wants 2D inputs, (n,1) vectors instead of (n,)
x1 = x[:, np.newaxis]

#Polynomial fit
poly2 = PolynomialFeatures(degree=p-1)
X = poly2.fit_transform(x1)
linreg = LinearRegression()
linreg.fit(X,y)

y_predict = linreg.predict(X)


#Plotting
fig = plt.figure(figsize=(18,12))
ax = fig.add_subplot(111)
```

```
ax.plot(x[ind],y_tilde[ind], label="Polynomial fit of degree {}".format(p-1),␣
 ↪color="red")
ax.scatter(x,y_predict, label="scikitlearn fit of degree {}".format(p-1),␣
 ↪color='green')
ax.legend(fontsize=24)

tot_diff = np.sum(y_tilde[ind] - y_predict[ind])

print("Total (elementwise sum) difference between own fit and scikitlearn's: {:.
 ↪4e}".format(tot_diff))
```

```
Total (elementwise sum) difference between own fit and scikitlearn's: 1.0552e-12
```



Note: the exercise says scikitlearn does not include the intercept, but as far as I can see from the documentation, it has default value true (fit_interceptbool, default=True) in the "sklearn.linear_model.LinearRegression" class.

0.2.3 Exercise 2.3

[4]:
```
from sklearn.metrics import mean_squared_error, r2_score

mse = mean_squared_error(y, y_predict)
r2 = r2_score(y, y_predict)
```

```
print("Mean squared error: {:.4e}".format(mse))
print("R2 score: {:.4e}".format(r2))
```

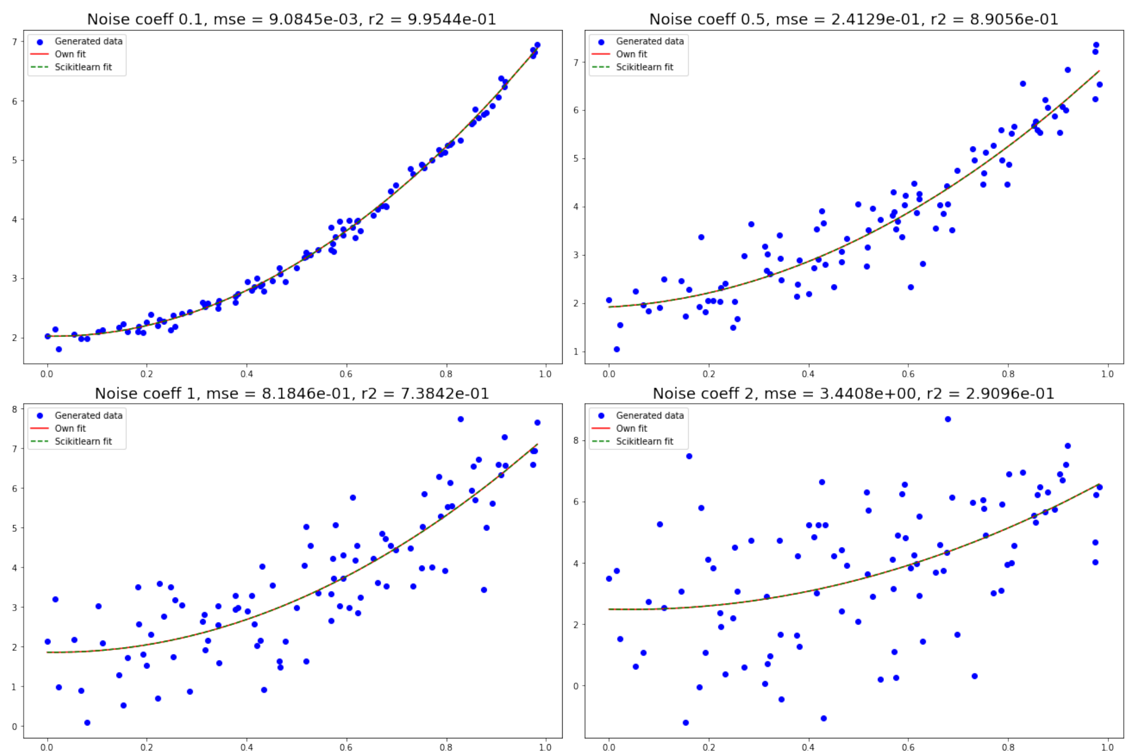
```
Mean squared error: 7.9026e-03
R2 score: 9.9641e-01
```

0.2.4 Discussion

See the discussion below the comparison

[5]:
```
def diff_noise(x, noise_coeff1):
    #x = np.random.rand(100)
    #noise_coeff1 = 0.1
    y_new = 2.0 + 5*x**2 + noise_coeff1 * np.random.randn(100)

    p = 3 #Number of predictors (polynomial order + 1 (the intercept))

    X = np.zeros((len(x), p)) #Design matrix
    X[:,0] = 1
    X[:,1] = x
    X[:,2] = x**2

    beta_new = (np.linalg.inv(X.T @ X) @ X.T ) @ y_new
    y_tilde = X @ beta_new


    #Scikit learn wants 2D inputs, (n,1) vectors instead of (n,)
    x1 = x[:, np.newaxis]

    #Polynomial fit
    poly2 = PolynomialFeatures(degree=p-1)
    X = poly2.fit_transform(x1)
    linreg = LinearRegression()
    linreg.fit(X,y_new)

    y_predict = linreg.predict(X)
    ind = np.argsort(x) #Sorted indeces for plotting fit as a smooth line

    return x[ind], y_tilde[ind], y_predict[ind], y_new[ind]

# Compare different noise coefficients
noise_list = [0.1, 0.5, 1, 2]
x = np.random.rand(100)
n_cols = 2
n_rows = 2
fig, axs = plt.subplots(ncols=n_cols, nrows=n_rows, figsize=(18, 12),
                        layout="constrained")
np.random.seed(10)
```

```python
counter = 0
for col in range(n_cols):
    for row in range(n_rows):
        noise = noise_list[counter]
        x, y_tilde, y_predict, y_new = diff_noise(x,noise)

        mse = mean_squared_error(y_new, y_predict)
        r2 = r2_score(y_new, y_predict)

        axs[col,row].scatter(x,y_new, label="Generated data", color="blue")
        axs[col,row].plot(x,y_tilde, label="Own fit".format(noise), color="red")
        axs[col,row].plot(x,y_predict, label="Scikitlearn fit".format(noise), color='green', linestyle="dashed")
        axs[col,row].set_title("Noise coeff {}, mse = {:.4e}, r2 = {:.4e}".format(noise, mse, r2), fontsize=18)
        axs[col,row].legend()
        counter += 1
```



The results behave pretty much as expected, with the fit yielding higher error and lower prediction score R^2 with increasing noise. In our original case with noise coefficient 0.1, we achieve an R^2-score of 0.99557, which is very close to the perfect score of 1, meaning the model is confident it will predict future samples.

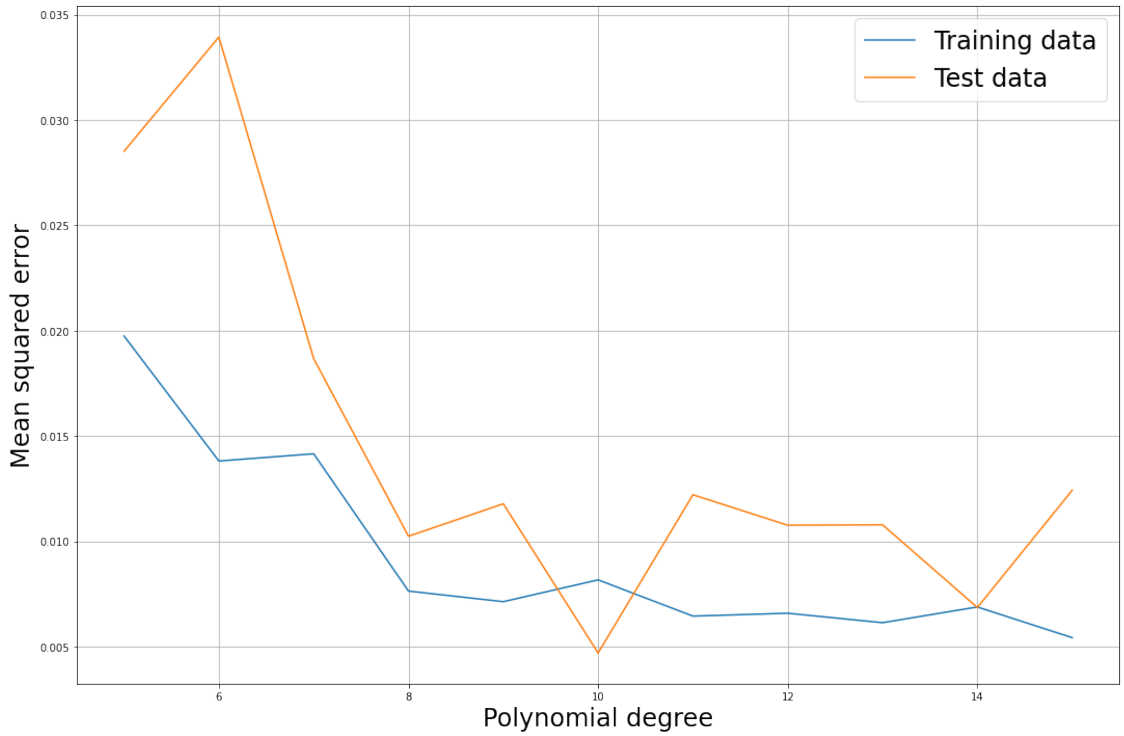
0.3 Exercise 3

Note that a), b) and c) is done in one go here, as this made the most sense for me. Dealing with the specific case of a fifth order polynomial in a) and b) is just running the function with p=6.

[6]:
```
from sklearn.model_selection import train_test_split
np.random.seed(1)
n = 100

noise_coeff2 = 1
x = np.linspace(-3, 3, n)
y = np.exp(-x**2) + 1.5 * np.exp(-(x-2)**2) + noise_coeff2*np.random.normal(0, 0.
↪1, x.shape)

def poly_train_func(x,y,p):
    "Input p is polynomial order + 1"
    X = np.zeros((len(x), p))
    for i in range(p):
        X[:,i] = x**i

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)

    beta = np.linalg.inv(X_train.T @ X_train) @ X_train.T @ y_train

    y_tilde = X_train @ beta
    y_predict = X_test @ beta

    mse_train = mean_squared_error(y_train, y_tilde)
    mse_test = mean_squared_error(y_test, y_predict)

    return mse_train, mse_test

deg_list = []
mse_train_list = []
mse_test_list = []

for deg in range(6,17): #5. order to 15. order polynomial
    mse_train, mse_test = poly_train_func(x,y,deg)

    deg_list.append(deg-1)
    mse_train_list.append(mse_train)
    mse_test_list.append(mse_test)
```

[7]:
```
#Plotting
fig = plt.figure(figsize=(18,12))
ax = fig.add_subplot(111)

ax.plot(deg_list, mse_train_list, label="Training data")
```

```python
ax.plot(deg_list, mse_test_list, label="Test data")
ax.set_xlabel("Polynomial degree", fontsize=24)
ax.set_ylabel("Mean squared error", fontsize=24)
ax.legend(fontsize=24)
ax.grid()
```



I found that the lowest MSE was given by a polynomial of order 10. Although the sample size is a bit low here, we can see the trends of over- and underfitting. The prediction error in the training data will generally decrease as you increase the complexity of the fit, but similtaneously increase the bias. This is a case of overfitting, which is why the test data trends upwards with increased complexity beyond a certain point. I.E. your model is so finely tuned to the training data, it becomes worse at predicting new examples it has not trained on.