FYS-STK Exercises week 35

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

Exercise 2

First we create our random x, and a dependent y with noise drawn from a normal distribution with standard deviation 0.1 (code modified from the exercise text).

```python
np.random.seed(10)
x = np.random.rand(100,1)
noise = 0.1 * np.random.randn(100,1)
y = 2.0+5*x*x + noise
```

Then we create our design matrix, with intercept, and x up to the second power.

```python
X = np.zeros((100,3))
X[:,0] = 1
X[:,1] = x.T
X[:,2] = (x**2).T
```

Then we optimize the cost function, here mean squared error (MSE), to get our betas with the analytical expression

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

```python
# optimize MSE
xtx_inv = np.linalg.inv(X.T.dot(X))

Best = xtx_inv.dot(X.T).dot(y)
```

```
print(f"Beta vector: {Best.T}")
```

```
Beta vector: [[2.0146372  0.02474757 4.93762474]]
```

We then get our predicted $\tilde{y}$ using

$$\tilde{y} = X\beta,$$

```
ytilde = X.dot(Best)
```

and calculate MSE

$$MSE(y, \tilde{y}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2.$$

```
MSE = sum((y - ytilde)**2) / len(y)
print(f"MSE: {round(MSE.item(), 5)}")
```

```
MSE: 0.00971
```

The MSE is 0.00971, which is very close to the value 0.1 that we used for our simulation.

Finally, we plot the data and regression

```
sorted_index = np.argsort(x, axis=None)
#print(sorted_index)
xsort = x[sorted_index]
ysort = y[sorted_index]
ytsort = ytilde[sorted_index]

plt.plot(xsort, ysort, ".")
plt.plot(xsort, ytsort, "-", label = "Line of best fit")
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.show()
```

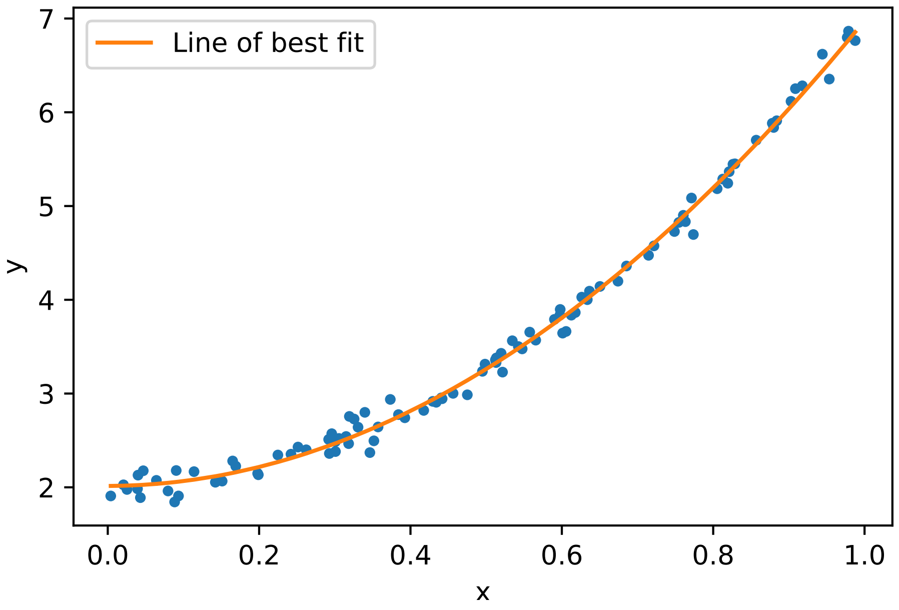

Figure 1: Linear regression for a model fit with polynomials up to the second degree.

Exercise 3

a)

For this exercise we also import `train_test_split` from `sklearn`.

```
from sklearn.model_selection import train_test_split
```

Create data (code from the exercise text)

```
np.random.seed()
n = 100
# Make data set.
x = np.linspace(-3, 3, n).reshape(-1, 1)
y = np.exp(-x**2) + 1.5 * np.exp(-(x-2)**2)+ np.random.normal(0, 0.1, x.shape)
```

Create a function to make a design matrix with an nth order polynomial

```
def design_poly_n(x, n):
    X = np.zeros((len(x), n))
    X[:,0] = 1
    for i in range(1, n):
        X[:,i] = (x**i).T

    return X
```

Then create design matrix up to a 5th order polynomial, and split in training and test data

```
X = design_poly_n(x, 5)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

b)

Calculate optimal beta and MSE

```
xtx_inv = np.linalg.inv(X_train.T.dot(X_train))
Best = xtx_inv.dot(X_train.T).dot(y_train)
print("Beta:", Best.T)
```

```
Beta: [[ 0.8626045   0.44595956 -0.01713606 -0.03636564 -0.00540025]]
```

```
ytilde_train = X_train.dot(Best)
MSE = sum((y_train - ytilde_train)**2) / len(y_train)
print("MSE", MSE)
```

```
MSE [0.02859804]
```

Compare to MSE when applying betas from training data to test data

```
ytilde_test = X_test.dot(Best)
MSE = sum((y_test - ytilde_test)**2) / len(y_test)
print("MSE", MSE)
```

```
MSE [0.02913716]
```

MSE is a bit higher for the test data.

c)

```
np.random.seed(3489)
n = 100
# Make data set.
x = np.linspace(-3, 3, n).reshape(-1, 1)
y = np.exp(-x**2) + 1.5 * np.exp(-(x-2)**2)+ np.random.normal(0, 0.1, x.shape)

maxpoly = 15
poly = np.arange(maxpoly) + 1
MSE_train_out = np.zeros(maxpoly)
MSE_test_out = np.zeros(maxpoly)

for i in poly:
    X = design_poly_n(x, i)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
    xtx_inv = np.linalg.inv(X_train.T.dot(X_train))
    Best = xtx_inv.dot(X_train.T).dot(y_train)

    ytilde_train = X_train.dot(Best)
    MSE_train_out[i-1] = sum((y_train - ytilde_train)**2) / len(y_train)

    ytilde_test = X_test.dot(Best)
    MSE_test_out[i-1] = sum((y_test - ytilde_test)**2) / len(y_test)
```

Then plot the MSEs to compare test and train set. Passing on to R for plotting.

```
# R code
# passing onto R for plotting
library(reticulate)
library(tidyverse)

polydeg <- py$poly
MSEtest <- py$MSE_test_out
MSEtrain <- py$MSE_train_out

df <- data.frame(
    poly = polydeg,
    test = MSEtest,
    train = MSEtrain
) %>%
    pivot_longer(c(test,train), names_to = "data_type", values_to = "MSE")

ggplot(df, aes(poly, MSE, col = data_type)) +
    geom_line() +
    theme_bw() +
    labs(
        x = "Polynomial degree",
        color = "Data type"
    )
```

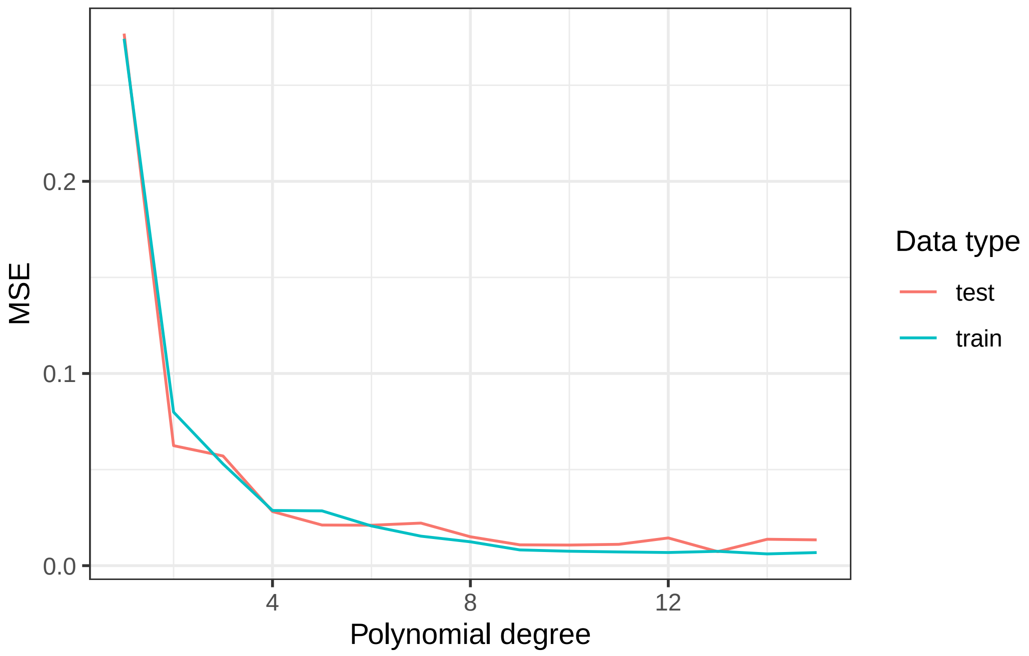

Figure 2: MSE as function of highest polynomial degree included in the model for training (80%) and test (20%) data. n = 100

For 15 degree polynomials the the test and training are still pretty similar. MSE for the training data is decreasing for each polynomial degree, while for the test data it fluctuates a bit more, the minimum seemingly at 13 degrees.

Try a higher-degree polynomial

```
np.random.seed(3489)
n = 100
```

```
# Make data set.
x = np.linspace(-3, 3, n).reshape(-1, 1)
y = np.exp(-x**2) + 1.5 * np.exp(-(x-2)**2)+ np.random.normal(0, 0.1, x.shape)

maxpoly = 20
poly = np.arange(maxpoly) + 1
MSE_train_out = np.zeros(maxpoly)
MSE_test_out = np.zeros(maxpoly)

for i in poly:
    X = design_poly_n(x, i)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
    xtx_inv = np.linalg.inv(X_train.T.dot(X_train))
    Best = xtx_inv.dot(X_train.T).dot(y_train)

    ytilde_train = X_train.dot(Best)
    MSE_train_out[i-1] = sum((y_train - ytilde_train)**2) / len(y_train)

    ytilde_test = X_test.dot(Best)
    MSE_test_out[i-1] = sum((y_test - ytilde_test)**2) / len(y_test)
```

```
# R code
# passing onto R for plotting

polydeg <- py$poly
MSEtest <- py$MSE_test_out
MSEtrain <- py$MSE_train_out

df <- data.frame(
    poly = polydeg,
    test = MSEtest,
    train = MSEtrain
) %>%
    pivot_longer(c(test,train), names_to = "data_type", values_to = "MSE")

ggplot(df, aes(poly, MSE, col = data_type)) +
    geom_line() +
    theme_bw() +
    labs(
        x = "Polynomial degree",
        color = "Data type"
    )
```

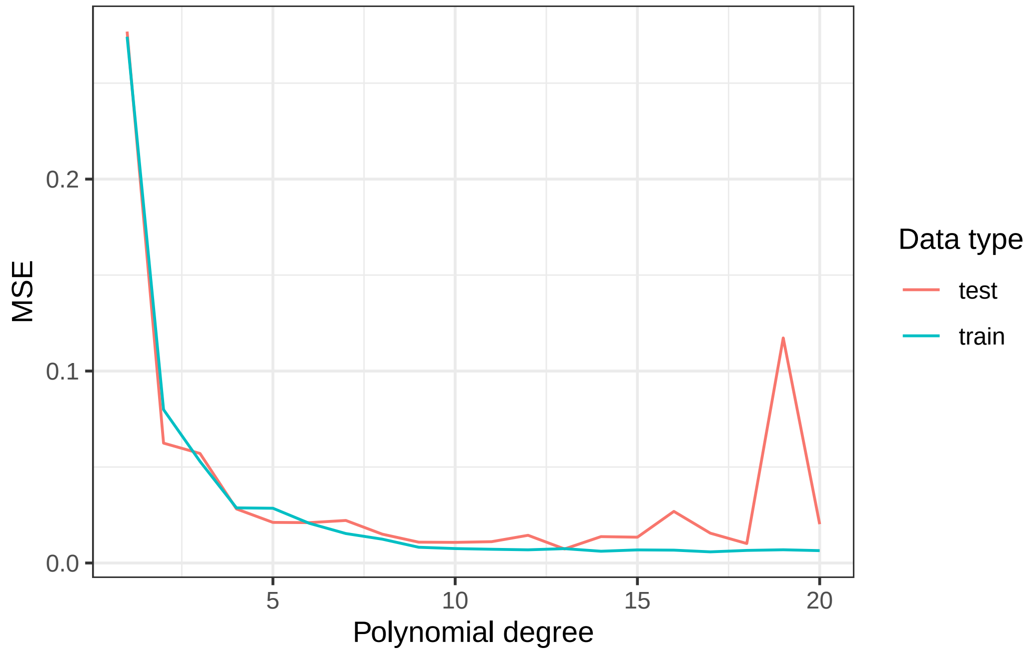

Figure 3: MSE as function of highest polynomial degree included in the model for training (80%) and test (20%) data. Same data as the previous figure, but for a higher polynomial degree. n = 100

The MSE starts to really deviate when polynomial degree goes above around 15. The best trade-off to avoid overfitting is probably around 10-15 degrees.