# Exercise Week 35

## Exercise 1: Analytical exercises

### a)

We know that $a^{\mathrm{T}} x$ is a dot product, and that it alternatively can be written as $a \cdot x$. We also have the following identity:

$$\frac{\partial\left(u^{\mathrm{T}} v\right)}{\partial w}=\frac{\partial(u \cdot v)}{\partial w}=u \cdot \frac{\partial v}{\partial w}+v \cdot \frac{\partial u}{\partial w}=u^{\mathrm{T}} \frac{\partial v}{\partial w}+v^{\mathrm{T}} \frac{\partial u}{\partial w}.$$

Furthermore, the derivative of $x$ with respect to itself is given by

$$\frac{\partial x}{\partial x}=\begin{bmatrix} \frac{\partial x_1}{\partial x_1} & \frac{\partial x_1}{\partial x_2} & \cdots & \frac{\partial x_1}{\partial x_N} \\ \frac{\partial x_2}{\partial x_1} & \frac{\partial x_2}{\partial x_2} & \cdots & \frac{\partial x_2}{\partial x_N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial x_N}{\partial x_1} & \frac{\partial x_N}{\partial x_2} & \cdots & \frac{\partial x_N}{\partial x_N} \end{bmatrix}=\begin{bmatrix} 1 & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 1 \end{bmatrix}=I_{N \times N},$$

while the derivative of $a$ with respect to $x$ gives us $0_{N \times N}$, assuming that none of the elements in $a$ are dependent on any of the elements in $x$. Thus, we get

$$\frac{\partial\left(a^{\mathrm{T}} x\right)}{\partial x}=a^{\mathrm{T}} \frac{\partial x}{\partial x}+x^{\mathrm{T}} \frac{\partial a}{\partial x}=a^{\mathrm{T}} I_{N \times N}+x^{\mathrm{T}} 0_{N \times N}=a^{\mathrm{T}}+0_{N \times 1}^{\mathrm{T}}=a^{\mathrm{T}},$$

just like we wanted to show.

Using the same identity as in the first line with $a$ as both $u$ and $w$, and $A a$ as $v$, along with the logic presented above we have

$$\frac{\partial\left(a^{\mathrm{T}} A a\right)}{\partial a}=a^{\mathrm{T}} \frac{\partial(A a)}{\partial a}+(A a)^{\mathrm{T}} \frac{\partial a}{\partial a}=a^{\mathrm{T}}\left(A \frac{\partial a}{\partial a}\right)+a^{\mathrm{T}} A^{\mathrm{T}} I_{N \times N}=a^{\mathrm{T}}\left(A I_{N \times N}+A^{\mathrm{T}} I_{N \times N}\right)=a^{\mathrm{T}}\left(A+A^{\mathrm{T}}\right),$$

where I have used that

$$\frac{\partial(A u)}{\partial v}=A \frac{\partial u}{\partial v},$$

as long as $A$ is not a function of $v$.

Lastly, we can split the partial derivative in the third expression into four seperate partial derivatives, i.e.

$$\frac{\partial (x-As)^{\mathrm{T}}(x-As)}{\partial s}=\frac{\partial \left(x^{\mathrm{T}}-s^{\mathrm{T}}A^{\mathrm{T}}\right)(x-As)}{\partial s}=\frac{\partial x^{\mathrm{T}}x}{\partial s}-\frac{\partial x^{\mathrm{T}}As}{\partial s}-\frac{\partial s^{\mathrm{T}}A^{\mathrm{T}}x}{\partial s}+\frac{\partial s^{\mathrm{T}}A^{\mathrm{T}}As}{\partial s}.$$

Since $x^{\mathrm{T}}x$ is independent of $s$, the first term is zero. The second and third terms are both equal to $x^{\mathrm{T}}A$, since $s^{\mathrm{T}}A^{\mathrm{T}}x=x^{\mathrm{T}}As$ is a scalar. Finally, the last term can be rewritten as

$$\frac{\partial s^{\mathrm{T}}A^{\mathrm{T}}As}{\partial s}=A^{\mathrm{T}}A\frac{\partial s^{\mathrm{T}}s}{\partial s}=A^{\mathrm{T}}A\frac{\partial s^{2}}{\partial s}=2A^{\mathrm{T}}As=2s^{\mathrm{T}}A^{\mathrm{T}}A,$$

hence

$$\frac{\partial (x-As)^{\mathrm{T}}(x-As)}{\partial s}=-x^{\mathrm{T}}A-x^{\mathrm{T}}A+2s^{\mathrm{T}}A^{\mathrm{T}}A=-2\left(x^{\mathrm{T}}-s^{\mathrm{T}}A^{\mathrm{T}}\right)A=-2(x-As)^{\mathrm{T}}A.$$

just like we wanted to show. To find the second derivative with respect to the vector $s$ we can use that $x^{\mathrm{T}}A$ and $A^{\mathrm{T}}A$ are both independent of $s$, hence

$$\frac{\partial^{2}(x-As)^{\mathrm{T}}(x-As)}{\partial s\,\partial s^{\mathrm{T}}}=-2\frac{\partial \left(x^{\mathrm{T}}-s^{\mathrm{T}}A^{\mathrm{T}}\right)A}{\partial s^{\mathrm{T}}}=-2\frac{\partial \left(x^{\mathrm{T}}A\right)}{\partial s^{\mathrm{T}}}+2\frac{\partial \left(s^{\mathrm{T}}A^{\mathrm{T}}A\right)}{\partial s^{\mathrm{T}}}=2A^{\mathrm{T}}A.$$

Since $A$ acts as the design matrix $X$ when $x$ is replaced with the outputs $y$ and the vector $s$ with the parameter vector $\beta$, we see that the double derivative of the mean squared error (which is proportional to $(y-X\beta)^{\mathrm{T}}(y-X\beta)$) indeed is proportional to the Hessian matrix $H=X^{\mathrm{T}}X$.

## b)

As we know:

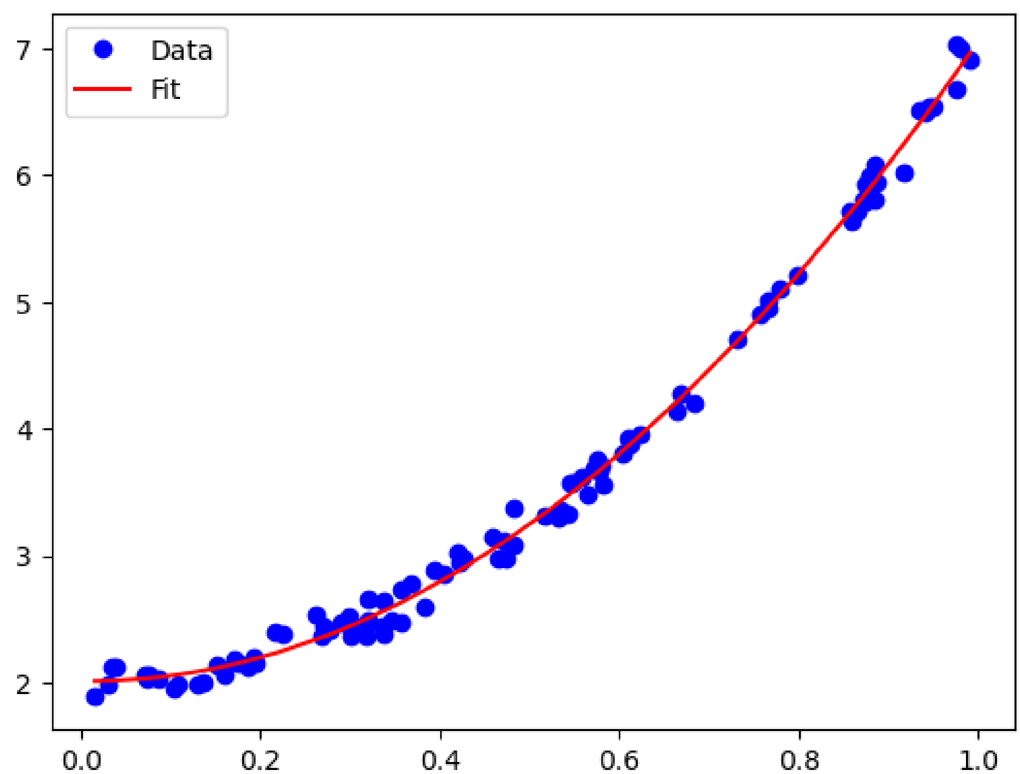
$$\mathrm{a}^{T}\mathrm{A}=\mathrm{z}^{T}$$

# Exercise 2: Making your own data and exploring scikit-learn

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split

n = 100
x = np.random.rand(n,1)
idx = np.argsort(x, axis=0).flatten()
x = x[idx] # Always sort sooner rather than later
y = 2.0+5*x*x+0.1*np.random.randn(n,1)
```

## 1. My Code

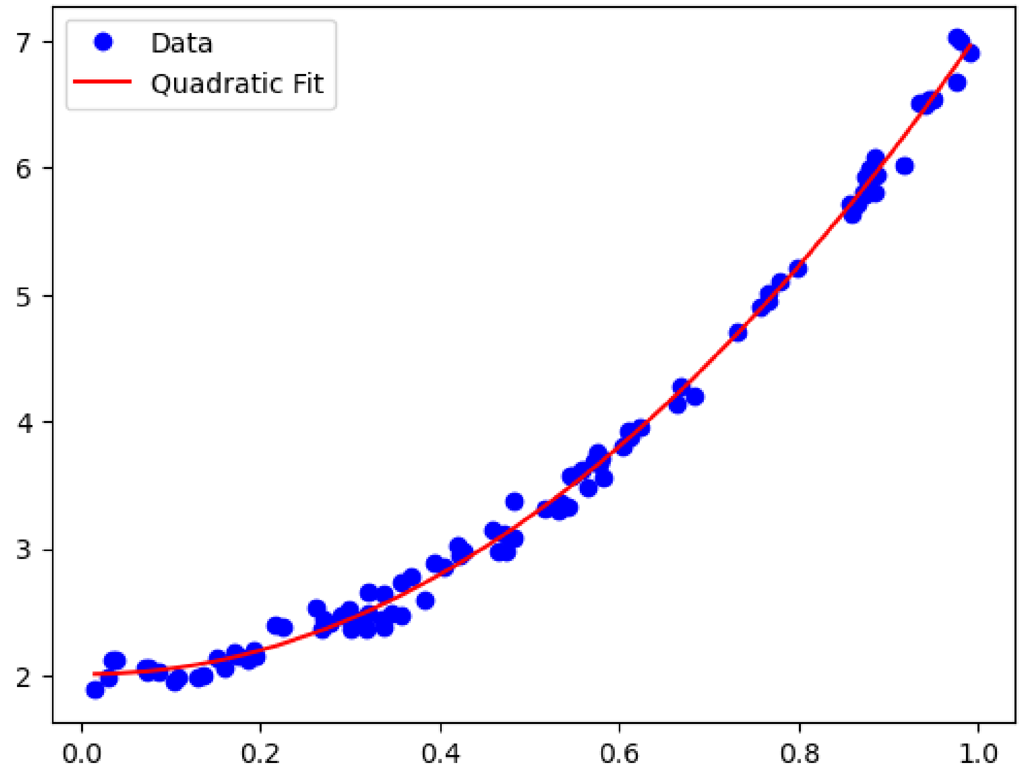
```
# Design matrix
X = np.ones((n,3))
X[:,1] = x.flatten()
X[:,2] = x.flatten()**2

# Coefficients and model
β = np.linalg.inv(X.T @ X) @ X.T @ y
y_pred = X @ β

plt.plot(x,y, "bo", label="Data")
plt.plot(x, y_pred, "r", label="Fit")
plt.legend()
plt.show()
```




## 2. Scikit-learn

```
poly2 = PolynomialFeatures(degree=2).fit(x, y)
X = poly2.fit_transform(x)

# Penta line fit (?)
plf5 = LinearRegression().fit(X, y)
y_pred = plf5.predict(X)

plt.plot(x, y, 'bo', label='Data')
```

```
plt.plot(x, y_pred, "r", label="Quadratic Fit")
plt.legend()
plt.show()
```




## 3. $MSE$ & $R^2$

**Mean Squared Errror** $(\chi^2)$: A function which measures the average square error between our prediction and the data. With more datapins $n$, we can have more confidence in our model. This should be minimized.

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

$R^2$: A function which measures how well our model can predict new values. The max score is 1, and the worst score is $-\infty$. A score of 0 means the same y-value for all predictions.

$$R^2(\vec{y}, \vec{\tilde{y}}) = 1 - \frac{\sum_{i=1}^{n} (y_i - \tilde{y}_i)^2}{\sum_{i=1}^{n} (y_i - \acute{y})^2}$$

```
MSE = mean_squared_error(y, y_pred)
R2 = r2_score(y, y_pred)
print(f'{MSE = :.2e}, {R2 = :.2%}')
```

```
MSE = 8.66e-03, R2 = 99.63%
```

# Exercise 3: Split data in test and training data

## a) Manual 5-deg polynomial design matrix and split

```
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)

# Manual design matrix
X = np.ones((n, 6))
X[:,1] = x.ravel()
X[:,2] = x.ravel()**2
X[:,3] = x.ravel()**3
X[:,4] = x.ravel()**4
X[:,5] = x.ravel()**5
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)
β = np.linalg.inv(X.T @ X) @ X.T @ y
```

## b) Predictions and MSE Calculation

```
y_tilde = X_train @ β
y_pred = X_test @ β

MSE_train = mean_squared_error(y_train, y_tilde)
MSE_test = mean_squared_error(y_test, y_pred)

print(f'{MSE_train = :.2e}')
print(f'{MSE_test = :.2e}')

MSE_train = 2.30e-02
MSE_test = 3.17e-02
```

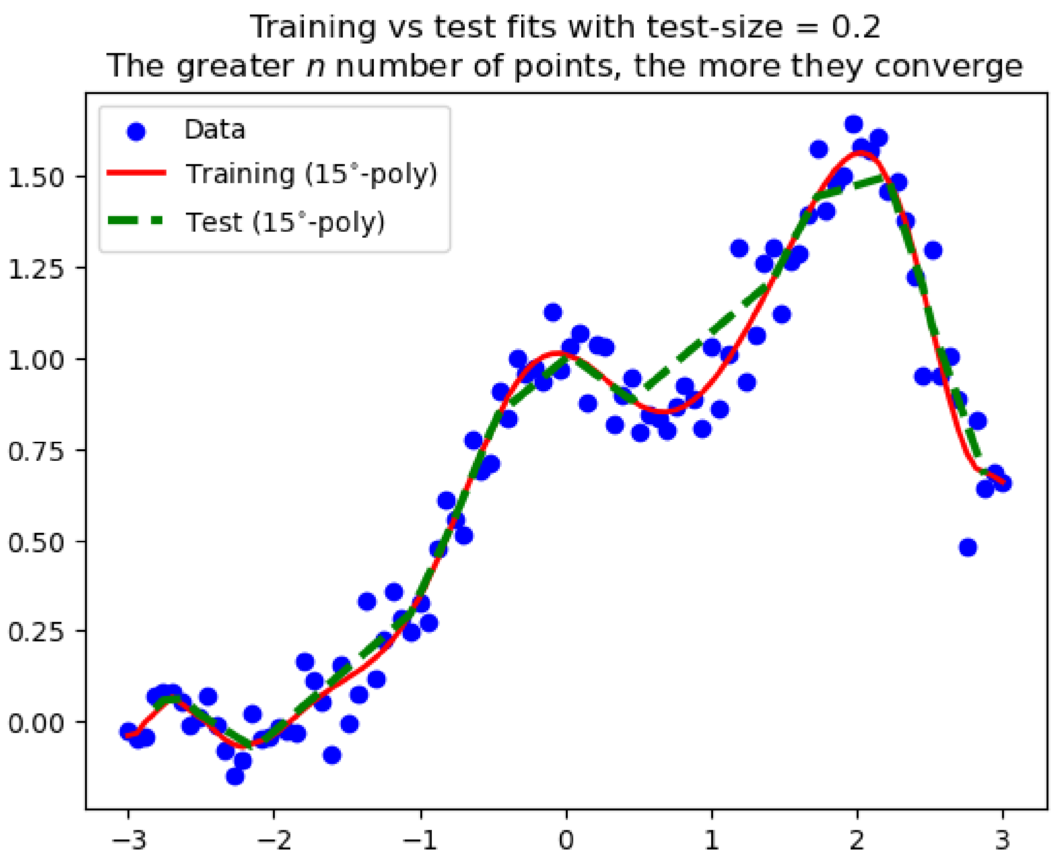
## c) Scikit-learn 15-deg Polynomial Design Matrix and Split

```
poly15 = PolynomialFeatures(degree=15).fit(x, y)
poly15 = PolynomialFeatures(degree=15)
X = poly15.fit_transform(x)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)


# Sorting columns as values are picked at random
idx_train = np.argsort(X_train[:,1])
```

```python
idx_test = np.argsort(X_test[:,1])
X_test = X_test[idx_test]
X_train = X_train[idx_train]

β = np.linalg.inv(X.T @ X) @ X.T @ y

y_tilde = X_train @ β
y_pred = X_test @ β

MSE_train = mean_squared_error(y_train, y_tilde)
MSE_test = mean_squared_error(y_test, y_pred)

print(F'{MSE_train = :.2e}')
print(F'{MSE_test = :.2e}')
```

```
MSE_train = 4.74e-01
MSE_test = 7.64e-01
```

```python
plt.scatter(x, y, c="b", label="Data")

plt.plot(X_train[:,1], y_tilde, c="r", label="Training ($15^{\circ}$-
poly)", linewidth=2)
plt.plot(X_test[:,1], y_pred, c="green", label="Test ($15^{\circ}$-
poly)", linewidth=3, linestyle='--')
plt.legend()
plt.title(f'Training vs test fits with test-size = 0.2\nThe greater
$n$ number of points, the more they converge')
plt.show()
```



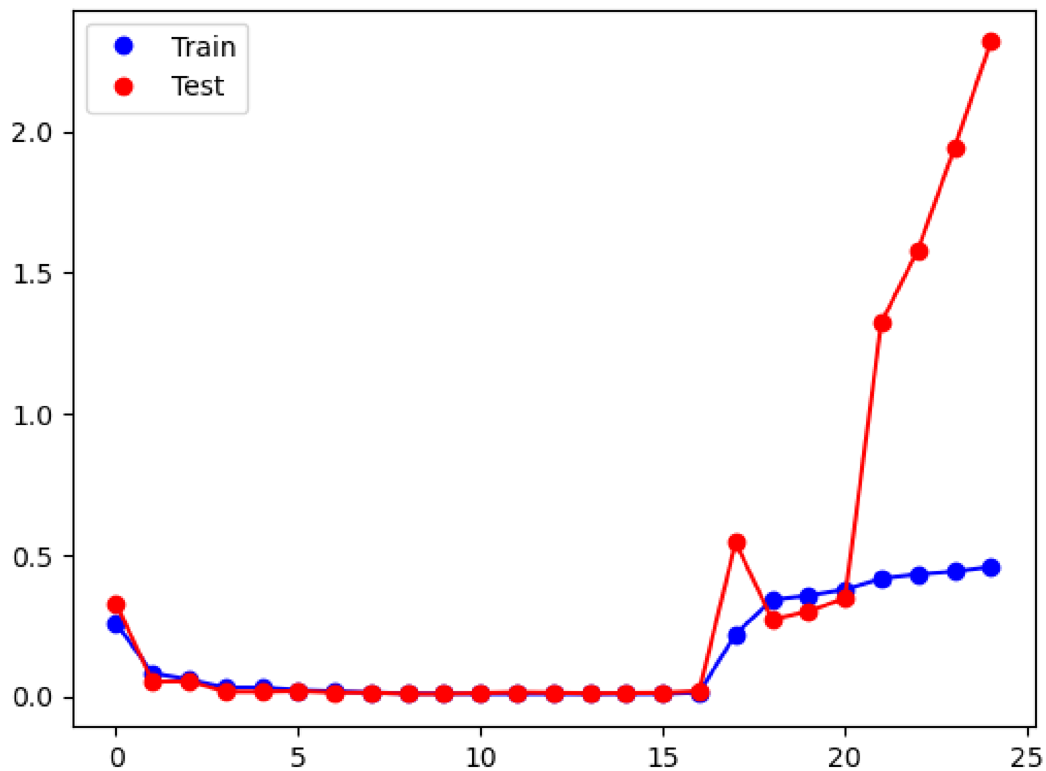

## Testing Different Degrees

1. Try

```
seed = 42

np.random.seed(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)

MSE_test = []
MSE_train = []

p_max = 25

for i in range(p_max):
    poly_i = PolynomialFeatures(degree=i)
    X = poly_i.fit_transform(x)

    # I get different answers depending on which way I create the
design matrix. I ended up using the built in version.
```

```python
    # X = np.ones((len(x), p_max))
    # for j in range(i):
    #     X[:, j] = x.ravel()**j


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

    β = np.linalg.pinv(X_train.T @ X_train) @ X_train.T @ y_train

    y_tilde = X_train @ β
    y_pred = X_test @ β

    MSE_train.append(mean_squared_error(y_train, y_tilde))
    MSE_test.append(mean_squared_error(y_test, y_pred))

print(MSE_test)
plt.plot(range(p_max), MSE_train, "b")
plt.plot(range(p_max), MSE_train, "bo", label="Train")

plt.plot(range(p_max), MSE_test, "r")
plt.plot(range(p_max), MSE_test, "ro", label="Test")
plt.legend()
plt.show()
```

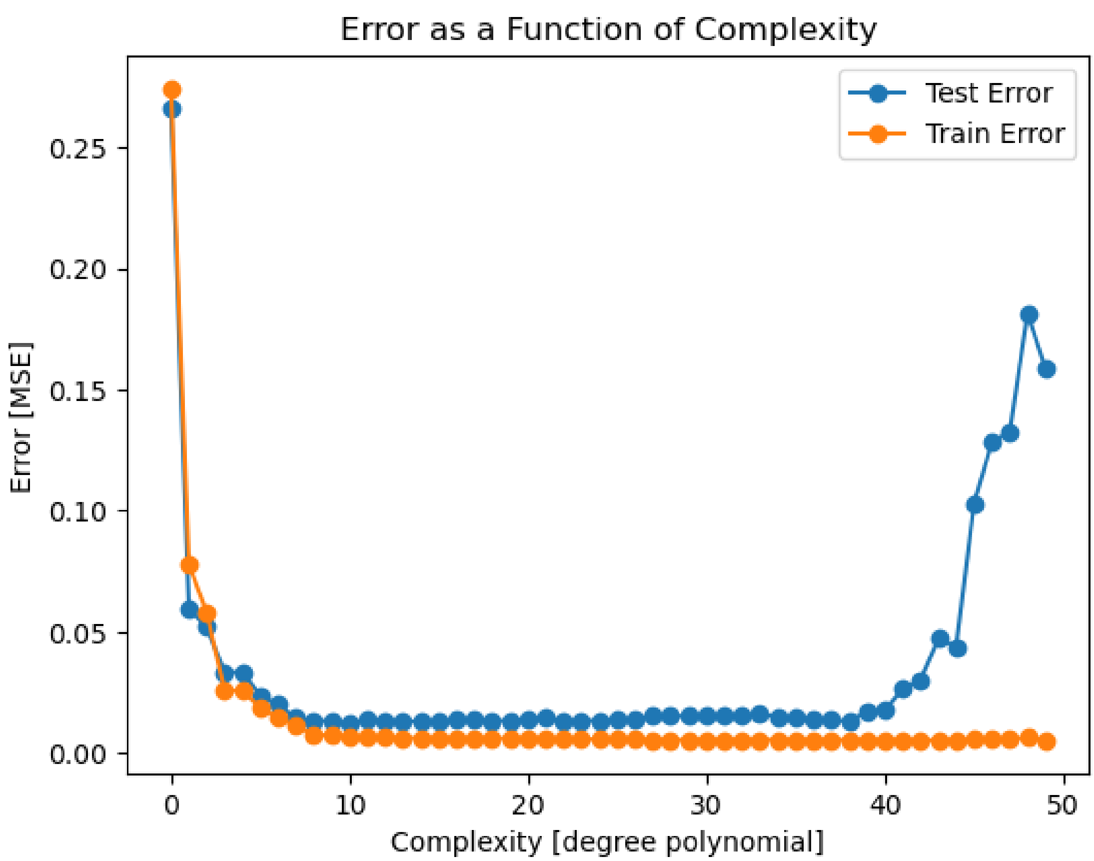
```
[0.3287667392934547, 0.05144824718205926, 0.05268728865540793,
0.015199719448453252, 0.014980695935185112, 0.01765614496691658,
0.011897777364679566, 0.011914110225868382, 0.007409878712775829,
0.007463300256676656, 0.010386745172822428, 0.012974607998867562,
0.011488594258683073, 0.009117237986602977, 0.010327990252747637,
0.011141631693318528, 0.01732309613518318, 0.5441733937006774,
0.2712992034132279, 0.2998650673534187, 0.3467158675116523,
1.324440469351247, 1.579646870277024, 1.9405105803869145,
2.3176452283198676]
```




2. Try: Most successful

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler


seed = 42
p_max = 50

np.random.seed(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)
test_error = []
train_error = []
X_train, X_test, y_train, y_test = train_test_split(x, y,
test_size=0.2)
scaler = StandardScaler()
scaler.fit(X_train)
```

```
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

for degree in range(p_max):
    model = make_pipeline(PolynomialFeatures(degree=degree),
LinearRegression(fit_intercept=False))
    clf = model.fit(X_train_scaled,y_train)
    y_fit = clf.predict(X_train_scaled)
    y_pred = clf.predict(X_test_scaled)
    test_error.append(mean_squared_error(y_test, y_pred))
    train_error.append(mean_squared_error(y_train, y_fit))

polydegree = [_ for _ in range(p_max)]
plt.plot(polydegree, test_error, "-o", label='Test Error')
plt.plot(polydegree, train_error, "-o", label='Train Error')
plt.xlabel('Complexity [degree polynomial]')
plt.ylabel('Error [MSE]')
plt.title('Error as a Function of Complexity')
plt.legend()
plt.show()
```



## Conclusion

- The greater the number $n$ of data points, the more accurate our model will be. With a high $n$, the test and training data was more similar.
- The MSE was consistently lower for the 5th degree polynomial by an order of magnitude, as $\mathrm{MRE}_5 \approx 10^{-2}$ and $\mathrm{MRE}_{15} \approx 10^{-1}$. This might shows that the 15th degree polynomial is overfitting the data.
- At the most successful attempt, we see a convergence of the training data and test data from the start, but with minimal error at around $8^{\circ}$-polynomial. We see divergence at around $40^{\circ}$-polynomial. We could therefore keep the model quite simple
- Different methods of finding the design matrix yielded different results. This is something to keep in mind when working with data.