Ausarbeitung Versuch 2 ILS Jan Holderied und Martin Goien

Aufgabe 1

a)

- Die Lambda Funktion nimmt das erste Element von x und Potenziert es mit n=3.
- Die Liste Phi für m = 5 enthält, 6 Einträge, die jeweils einer Funktion entsprechen: x[0]**0 bis x[0]**5
- Beim Aufrufen der Funktion get_phi_polyD1(m) wird die Funktion lambda x: np.array([phi_j(x) for phi_j in phi]) zurückgegeben. Beim Anwenden der zurückgegebenen Funktion auf verschiedene Werte von x, wird ein Numpy-Array zurückgegeben. Das Numpy-Array wird durch die m lambda Funktionen in der Liste phi berechnet. Für m=5 wird eine Liste aus von 6 lambda Funktionen zurückgegeben, die die Potenzen x[0]**0 bis x[0]**5 bilden.
- Bei der Erzeugung der lambda Funktionen wird n jeweils durch die funktion range() erzeugt. Um nun n innerhalb der Anonymen Funtion zu speichern, wird n dem Parameter n zugewießen, damit der wert von n beim späteren Aufruf vernwendet werden kann.
- Phi ist eine Liste von m+1 lambda Funktionen. Lambda x erzeugt ein Numpy-Array mit Werten, die mit Hilfe der m+1 lambda Funktionen aus Phi berechnet werden.
- phi_poly1D([2]) = [1 2 4 8 16 32]

b)

- n0, n1 sind die Exponenten der Polynome für die jeweiligen Merkmalsfunktionen, wobei n jeweils für den Grad 0 bis m steht.

```
n0:  0 n1:  0
n0:  0 n1:  1
n0:  1 n1:  0
n0:  0 n1:  2
n0:  1 n1:  1
n0:  2 n1:  0
n0:  0 n1:  3
n0:  1 n1:  2
n0:  2 n1:  1
n0:  3 n1:  0
n0:  0 n1:  4
n0:  1 n1:  3
n0:  2 n1:  2
n0:  3 n1:  1
n0:  4 n1:  0
```

- Es gibt $\binom{D+m-1}{m}$ viele Basisfunktionen vom Grad m.
- phi_poly2D([1,2])= [1 2 1 4 2 1 8 4 2 1 16 8 4 2 1]

Aufgabe 2

a)

- Es gibt folgende Klassen im Modul Regression:

    - DataScaler: Klasse um Datenvektoren zu standardisieren.
    - KNNRegressifier: Klasse für K-Nearest-Neighbor-Regressions Verfahren mit Hilfe von KD-Trees.
    - LSRRegressifier: Klasse für Least Squares Regression, Summe der Fehlerquadrate.
    - Regressifier: Abstrakte Basisklasse. Obige Regressionsklassen erben von dieser Klasse.
- Die Methoden der Basisklasse Regression sind folgende:
    - fit(): Die Methode fit Trainiert das Regressionsmodell mit den Trainigsdaten Matrix X und den Zieldaten Matrix T
    - predict(): Bekommt als Parameter einen neuen Datenvektor x, für den dann ein Zielvektor berechnet werden soll.
    - crossvalidate(): Macht eine Kreuzvalidierung. Parameter S bestimmt in wie viele Teile die Daten X unterteilt werden sollen. X ist die Trainingsdaten Matrix. T ist die Zeilwerte Matrix. Dist(t) berechnet die länge jedes einzelnen Vektors t, default ist die Euklidische Distanz.

b)

- Die Klasse LSRRegressifier erzeugt ein Objekt, dass ein Regressionsverfahren auf Basis der Summe der Fehlerquadrate implementiert.
- Die Parameter der Klasse LSRRegressidier:
    - lmbda: Ist ein Regulariserungs Parameter. Dieser verhindert zu große Gewichte, zwingt viele Komponenten auf nahe 0, reduziert effektive Parameterzahl.
    - phi: Basisfunktionen phi für lineare Regression, default ist ein Polynom mit Grad eins.
    - flagsSTD: Ist ein Flag welches bei einem Wert von größer 0 aussagt, dass die Daten Standartisiert sind.
    - eps: Maximal tolerierbarer Restwert, default ist $1*10^{-2}$
- Die Klasse DataScaler dient dafür die Datenvektoren zu standardisieren.
    - scale(): Methode standardisiert einen Vektor oder eine Matrix, auf Mittelwert 0 und Standartabweichung 1.
    - unscale(): Diese Methode rechnet für einen Vektor oder eine Datenmatrix die Standardisierung wieder zurück, auf die uhrsprüngliche Verteilung.
    - Dies ist wichtig um das Regressionsverfahren Numerisch Stabil zu halten.
    - maxZ: Variable steht für die Maximal zulässige größe der Datenmatrix X, damit diese noch gut konditioniert ist.
    - Z: Bei guter konditionierung wird Z zur null Matrix.

c)

- Die Klasse KNNRegressifier berechnet eine Regression mit Hilfe des Fast K-Nearest Neighbors Modell. Es wird ein KD-Tree verwendet.
- Wozu dienen die Parameter:
    - K: gibt die Anzahl der K-Neighbors an
    - flagKLinReg: Wenn diese Variable > 0 ist, soll eine Lineare Least Squares Fehlerfunktionen auf die K-Neighbors angewandt werden. Bei = 0, soll einfach nur der Mittelwert der K-Neighbors Zielwerte Vektoren berechnet werden.
- Als erstes werden die Indexe der K-Nearest Neighbors berechnet und in einer Liste abgelegt. Wenn das Flag flagKLinReg gleich null ist, wird einfach der Mittelwert aus den Zielwerten berechnet. Wenn das Flag flagKLinReg größer null ist, wird aus den Daten ein Regressionsmodell erzeugt, mit Hilfe der Klasse

LSRRegressifier. Danach wird der Zielwert des Datum x mit der predict Methode der Klasse LSRRegressifier brechnet.

d)

- Im Modultest werden als ertses alle nötigen wie Daten T und X erzeugt. T wird hier mir einem noise versehen. Ebenfalls wird die Merkmalsfunktion ϕ mit Dimensions 1 und Grad 2 erzeugt. Danach wird ein Objekt von LSRRegressifier erzeugt und trainiert. Mit dem trainierten LSRRegressifier wird nun eine prediction mit dem neuen Datum [3.1415] berechnet. Das Modell wird anschließend kreuzvalidiert, um einen mean absolute und einen mean absolute perecentage error zu berechnen. Anschliesen wird ein Objekt des KNNRegressifier erzeugt und dieses trainiert. Hier wird ebenfall mit dem Datum [3.1415] eine prediction berechnet. Auch hier wird das Modell wieder kreuzvalidiert.
- Es werden folgende Gewichte gelernt: $[w_0, w_1, w_2]$ mit der Prädiktionsfunktion $y(x,w)=w_0x^0+w_1x^1+w_2x^2$
- MAE $\triangleq$ mean absolute error, MAPE $\triangleq$ mean absolute percentage error
    - MAE ist also der durchschnittliche Regressionsfehler der vom Modell gemacht wird
    - MAPE ist der durchschnittliche Regressionsfehler in Prozent der vom Modell gemacht wird
- Die Funktionen custom_range erzeugt eine Liste mit den Werten [0, 0.1, 0.2, ..., 1, 2, ..., 10, 20,..., 100, 200, ..., 1000] diese Werte werden alle einmal als Hyperparamter lambda und k gestezt und der MAE berechnet. Für K wird erst aber größer gleich eins getestet. Die Auswertung dieses Test ergab die Optimalen Hyperparamter: lambda für LSR: lambda= 1.0 , MAE= 0.6384710845985233 und Bestes K für KNN: K= 1 MAE= 0.43.

AUfgabe 3

a)

- Die N = 10 Trainingsdaten sind mit dem Abstand 0.11 zwischen 0 und 1 als Sinus verteilt. Dazu kommt ein Normalverteilter Noise der für jeden Datenpunkt dazu addiert wird. $f(x) = c*sin(2*\pi*f*X+\phi_0)+ \frac{1}{\sigma\sqrt{2\pi}}+e^{-0,5(\frac{x-\mu}{\sigma})^2}$
- $lambda = lambda_scale * (10^{lambda_log10})$
    - lambda_scale ist ein Skalierungsfaktor für den Regularisierungsparameter
    - lmbda_log10 ist der logaryhthmus des Regularisierungsparameter
    - Die Unterteilung dient dazu um eine Grobe und eine feine Skalierung/Einstellmöglichkeit zu haben. Mit Lambda_log10 kann ein logarithmisches vielfaches eingestellt werden und mit lambda_scale dieses dann skaliert bzw noch fein justiert werden.
- Die Unterschiedlichen Kurven stehen für folgendes:
    - Blau gestrichelt Kurve: Ist die wahre Funktionskurve
    - Rote Kurve: Ist die Least Squares Regression Modellkurve
    - Blaue Kreuze: Sind die Trainingsdaten Punkte
    - Gelben Kringel: Sind die Testdaten Punkte
- Slider:
    - Seed: Stellt den Seed für den Zufallszahlen Generator ein.
    - N: Stellt die Anzahl an Datenpunkte für Training und Testen ein.
    - sd_noise: Reguliert den Noise der auf die Daten addiert wird.
- **Experiment 1 N = 10**: Ohne Regularisierung kommt es zum Overfitting, da der Polynomgrad nahe der Anzahl den Datenpunkte ist.

- Ohne Regularisierung:

```
MAE_train=0.0002850648465006794, MAPE_train=0.0005980925603867552
MAE_test =8.863847595761266, MAPE_test =1.6283488145198937
```

- Lambda Optimiert LSR:

```
Results
Least Squares with lambda=0.022000000000000002
weights w=[ 0.41615209  1.07334467 -2.98109699 -1.56076258 -0.0971874   0.7015
  0.92574004  0.76997057  0.39631142 -0.08316296]
MAE_train=0.5077822112979301, MAPE_train=1.1809524820303248
MAE_test =0.40821262632168603, MAPE_test =2.8067948959680913
```

- K Optimiert KNN:

```
Results
KNN with K=4
MAE_train=0.49580798697995776, MAPE_train=0.8857806895226009
MAE_test =0.5087853967461358, MAPE_test =1.2580210269862508
```

- Für die Datenmenge N = 10 funktioniert die LSR Regression besser.

- **Experiment 2 N = 20**:
  - Lambda Optimiert LSR:

```
Results
Least Squares with lambda=0.0001
weights w=[ -0.22630759  11.27599658 -28.38149409   5.94104704  14.7892986
   7.53319342  -4.52129973 -11.57914952  -7.05834371  12.35803874]
MAE_train=0.45321529532083493, MAPE_train=1.4285867640024388
MAE_test =0.5447824449574326, MAPE_test =1.8103942090208005
```

  - K Optimiert KNN:

```
Results
KNN with K=3
MAE_train=0.5097435289813654, MAPE_train=1.6903754561991442
MAE_test =0.5552622810991148, MAPE_test =1.5273519139041531
```

- **Experiment 3 N = 500**:
  - Lambda Optimiert LSR:

```
Results
Least Squares with lambda=1.3000000000000001e-05
weights w=[ -0.07345939   6.95039179  -1.88930481 -49.06821556  53.61257575
   7.37125082 -28.40841545   1.70744909  33.91487979 -24.00816554]
MAE_train=0.4052938448578478, MAPE_train=4.264587387607232
MAE_test =0.37678488321780335, MAPE_test =2.9958138747219074
```

  - K Optimiert KNN:

```
Results
KNN with K=24
MAE_train=0.3981304863883478, MAPE_train=5.072502541604303
MAE_test =0.37580876425887605, MAPE_test =3.81642986990415
```

b)

- Es gibt drei Möglichkeiten die Trainingsdaten zu generieren:
  - 2-Dimensionale Sinusfunktion $f(x_1,x_2)=c*sin(2*\pi*f_1*x_1+f_2*x_2+\phi_0)+e^{-0,5(\frac{x-\mu}{\sigma})^2}$
  - 2-Dimensionale SI-Funktion $f(x_1,x_2)=c*si(2*\pi*fr+\phi_0)+e^{-0,5(\frac{x-\mu}{\sigma})^2}$
  - 2-Dimensionale Ebene $f(x_1,x_2)=c*x_1+d*x_2+e^{-0,5(\frac{x-\mu}{\sigma})^2}$
- Azimuth dreht den Plot um seine Z-Achse und Elevation dreht den Plot um seine Y-Achse.
- Da sich die Daten in einem 2-Dimensionalen Raum befinden, zum Beispiel auf einer Ebene, wird durch die Wurzel erzielt das in alle Richtungen gleiche viele Daten verteilt sind. Beispeil auf einer Ebene sind in beide Richtungen der Ebene gleich viele Datenpunkte vorhanden und es ist nicht mögliche eine Zahl am Slider einzustellen ohne eine gleichmäßige Verteilung.
- Neue Funktionen im Skript sind:
  - Slider für Azimuth
  - Slider für Elevation
  - Die Datenanzahl N wird nun durch $\sqart(N)$ eingestellt

- Warscheinlichkeitsverteilung der Trainingsdaten jetzt mit Sin, SI, und Ebnen FUnktion
- Änderungen in der step() Funktion um nun die Modelle mit 2-D Daten zu Trainierung und auszuwerten

Plane mit N = 10

- Ohne Regularisierung:

```
Results
Least Squares with lambda=0.0
weights w=[ 1.51501800e-01  1.70538051e+00  1.18645386e+00 -5.99144034e-01
  7.69051883e-01 -7.71122736e-01 -1.63206643e+00  8.50229753e-01
  5.08870722e-01 -1.07448282e+00  4.77003691e-01 -1.23826318e+00
  7.90337250e-01  5.38140311e-02  5.93773597e-01  1.47942519e+00
 -8.03081601e-01 -4.68056472e-02 -5.28628736e-01 -8.76581920e-01
  1.36037380e+00 -2.20103972e-01  4.92385322e-01 -5.52758303e-02
  9.99595708e-02 -3.93431477e-01 -6.93939299e-02 -1.40719533e-01
 -5.24288704e-01  3.40214907e-01  3.87132178e-02  1.33147305e-01
  5.87599513e-02  9.37781571e-02  3.70654828e-01 -5.66928996e-01
  3.29438731e-02 -5.67586092e-02 -4.10733408e-03 -2.82808937e-02
  1.85368887e-02  9.29418817e-03  5.03760037e-02  6.87445692e-03
  9.63001003e-03  6.07698025e-02 -4.71888567e-02 -9.96693147e-03
 -4.26266966e-03  4.79487163e-03 -2.01457679e-02 -1.46956712e-02
 -3.06785492e-05 -4.70559361e-02  7.24708775e-02]
MAE_train=0.2592545325584011, MAPE_train=0.5672918006606058
MAE_test =1.6305448329345174, MAPE_test =1.842563182269817
```

- Lambda Optimiert LSR:

```
Results
Least Squares with lambda=7.7
weights w=[-0.0637523   0.55995464  0.49935671 -0.01763099  0.04358322  0.00258239
  0.25545514  0.10881488  0.11227796  0.25280945 -0.01498149 -0.08290636
  0.09515776  0.04969333  0.01873999  0.09684596  0.03432773 -0.04372512
 -0.07830265  0.00998314  0.10120895 -0.03626502  0.02121498  0.07241088
 -0.02649378 -0.11977342  0.0098904   0.01459101 -0.0574063   0.03876855
  0.0091204  -0.00481521 -0.00426407 -0.06196504  0.00436796 -0.07013629
  0.00980764  0.00128524 -0.01123447 -0.00712175 -0.00161598  0.01175873
  0.01944874 -0.00568622 -0.00462073  0.00557557 -0.01186755 -0.00293102
  0.00963803  0.00710652 -0.00764464 -0.00383921  0.01815369 -0.00199306
  0.00899143]
MAE_train=0.3371601088874742, MAPE_train=0.980145464678751
MAE_test =0.614161938359486, MAPE_test =1.1512585454788038
```

- K Optimiert KNN:

```
Results
KNN with K=10
MAE_train=0.40152151305833284, MAPE_train=1.2422856229860888
MAE_test =0.4404939423982262, MAPE_test =1.2021942799201597
```

- Bei N = 100 Daten funktioniert die KNN Regression besser als die LSR Regression.

Sin mit N = 10

- Lambda Optimiert LSR:

```
Results
Least Squares with lambda=1600.0
weights w=[-2.24343901e-03  3.67385527e-04 -1.33034495e-03 -3.00288075e-03
  4.50402138e-04 -4.45467977e-05  1.47950012e-03  1.69293954e-03
 -3.36631504e-03 -4.59455708e-03 -4.94363786e-03 -7.81749983e-04
  5.83312662e-05  6.08860861e-04  3.97810369e-04  3.91873236e-03
  4.68451345e-03 -2.06677570e-03  8.53519180e-03  5.36048956e-04
 -9.75266446e-03 -6.72195191e-03 -7.06091941e-04  6.55625530e-04
 -1.06826851e-03 -1.43759830e-03  6.96094904e-04  5.31815701e-04
  6.37810266e-03  7.09470601e-03  1.25093692e-04  1.87826327e-02
  3.49348585e-03  1.19077419e-02  6.88222117e-03 -1.45795654e-02
  1.76208876e-03  8.01013525e-04  3.10179201e-03 -1.77595220e-03
 -2.55378871e-03  8.14958147e-04  3.11550974e-04  3.24796048e-04
 -3.04216183e-04 -2.37358838e-03 -6.53041359e-03  5.67528765e-04
  1.41280830e-02  1.82744825e-03 -1.49239017e-02 -7.73884500e-03
 -2.31539137e-03  3.34553832e-03  3.95725864e-03]
MAE_train=0.5346504253417858, MAPE_train=66.82849725447448
MAE_test =0.6718382256163921, MAPE_test =56.711212474776055
```

- K Optimiert KNN:

```
Results
KNN with K=3
MAE_train=0.38072878528170206, MAPE_train=254.59713228326763
MAE_test =0.4832733215998089, MAPE_test =3.2505410626753757
```

SI mit N = 10

- Lambda Optimiert LSR:

```
Results
Least Squares with lambda=1400.0
weights w=[-2.49641775e-03  3.25294452e-04 -1.41068331e-03 -3.30191789e-03
  5.16716593e-04 -1.64919961e-05  1.52059113e-03  1.75633049e-03
 -3.96338244e-03 -5.07391925e-03 -5.40485675e-03 -8.91516796e-04
  1.26986238e-04  7.05090832e-04  4.93232517e-04  4.16877908e-03
  4.95572034e-03 -2.50470030e-03  9.20276527e-03  3.66319042e-04
 -1.08102699e-02 -7.33124821e-03 -8.11429571e-04  8.00892514e-04
 -1.18835208e-03 -1.56420280e-03  7.96727638e-04  6.51717990e-04
  6.86623923e-03  7.58166878e-03 -6.38505367e-05  2.03422760e-02
  3.86336912e-03  1.27982375e-02  7.40556950e-03 -1.61690891e-02
  1.94093759e-03  8.63148817e-04  3.18262109e-03 -1.88058836e-03
 -2.79262520e-03  9.35025837e-04  4.76418739e-04  2.78674707e-04
 -3.48316322e-04 -2.50494069e-03 -6.88754862e-03  5.00514341e-04
  1.46785358e-02  2.13220270e-03 -1.58379831e-02 -8.05563856e-03
 -2.39919154e-03  3.28655589e-03  4.41829080e-03]
MAE_train=0.531313204306722, MAPE_train=52.84849840190854
MAE_test =0.6730042516133683, MAPE_test =50.29358603178038
```

- K Optimiert KNN:

```
Results
KNN with K=3
MAE_train=0.380728785281702, MAPE_train=254.59713228326763
MAE_test =0.4832733215998089, MAPE_test =3.2505410626753757
```

Plane mit N = 30

- Lambda Optimiert LSR:

```
Results
Least Squares with lambda=2.0
weights w=[ 3.08342779e-03  8.21696863e-01  7.90527023e-01  1.00644492e-01
  2.48986213e-01 -2.01980767e-01  2.66863777e-01  3.01312403e-01
  1.97570411e-01  3.17090171e-01 -8.40565486e-02 -5.66531755e-02
  2.11755477e-02 -1.99709046e-01  1.83481811e-01 -1.46954872e-01
 -9.26510084e-02 -1.16522934e-01 -2.81456546e-01 -9.26909509e-02
 -1.72482452e-01  2.15343176e-02 -5.32299737e-02  3.27344588e-02
  7.29114632e-02 -4.56435114e-02  5.33177369e-02 -5.65838322e-02
  3.12915309e-02  6.60640307e-05  4.83508948e-02  7.22541627e-02
  1.94919893e-02  6.89453807e-02  1.21542012e-02  4.08716146e-02
 -1.86341891e-03  1.39366090e-02 -2.13579468e-03 -9.16532147e-03
 -5.30095735e-03 -3.10053156e-03  9.35608251e-03 -7.00930903e-03
  6.14517641e-03 -1.34339786e-03  1.52030238e-03 -1.13360457e-02
 -3.22006010e-03  6.75294457e-03 -1.21438479e-02 -6.59346117e-03
 -2.62497177e-03  2.65496957e-04 -3.73012091e-03]
MAE_train=0.37675571215513903, MAPE_train=1.0066654778923172
MAE_test =0.4100642348774386, MAPE_test =2.3167673386688654
```

- K Optimiert KNN:

```
Results
KNN with K=32
MAE_train=0.39282613797368976, MAPE_train=2.4231440166853497
MAE_test =0.4082421352362284, MAPE_test =1.3231480666791038
```

- Bei N = 100 Daten funktioniert die KNN Regression besser als die LSR Regression.

Sin mit N = 30

- Lambda Optimiert LSR:

```
Results
Least Squares with lambda=0.002
weights w=[ 2.07598073e-02  1.04369139e+00  2.20419648e+00  1.66329541e-01
  3.76123316e-01 -3.74094641e-01  4.74495252e-02 -2.06475030e+00
 -5.18953038e+00 -2.87374150e+00 -1.65024850e-01 -1.34976920e-01
  4.71927749e-02 -3.13244643e-01  3.64487800e-01 -2.82599663e-01
  4.19393170e-02  1.37681738e+00  2.96599133e+00  3.24810474e+00
  7.48747766e-01  5.11059499e-02 -3.54082814e-02  3.65736827e-02
  1.05651236e-01 -6.52548251e-02  8.72570599e-02 -1.20411660e-01
  8.19626520e-02  4.04644462e-02  2.95707645e-03 -2.07581255e-01
 -7.76201306e-01 -9.28443501e-01 -6.00277182e-01  9.08304025e-03
 -5.27028569e-03  1.25390909e-02 -3.18507825e-03 -1.20167944e-02
 -4.63880695e-03 -6.82987209e-03  1.21738579e-02 -1.03546318e-02
  1.34098163e-02 -6.80442911e-03 -2.21788080e-03 -1.35343496e-02
 -5.85020002e-03  2.55332724e-02  4.59669644e-02  9.26826597e-02
  8.06453179e-02  3.00987065e-02 -1.28942528e-02]
MAE_train=0.37655031473143935, MAPE_train=2.1306156572439923
MAE_test =0.4157835737253886, MAPE_test =5.675535460866786
```

- K Optimiert KNN:

```
Results
KNN with K=12
MAE_train=0.377512310532976, MAPE_train=2.8587101159757498
MAE_test =0.4278391832767592, MAPE_test =2.7310565037796515
```

SI mit N = 30

- Lambda Optimiert LSR:

```
Results
Least Squares with lambda=100000.0
weights w=[ 1.25139334e-06  1.26242987e-05 -3.17979380e-05 -1.55231111e-05
 -2.25539358e-05 -1.02128536e-04  6.51840536e-05 -4.95675075e-06
 -8.79613030e-06 -6.31423632e-05 -3.09106191e-05 -1.63480096e-04
  9.62566489e-05 -6.75648234e-05 -1.71709479e-04  1.66377896e-04
 -3.82322723e-05  1.38668988e-05  6.50703320e-06 -7.25326054e-05
 -1.29816498e-04 -4.59603087e-05 -2.82910876e-04  2.32507957e-04
 -7.91391765e-05  6.48764941e-05 -1.54809376e-04 -2.39440818e-04
  3.58233967e-04 -8.16600920e-05 -4.84152621e-05 -4.62717227e-05
  1.11692339e-04  1.58577520e-04 -2.50212474e-04 -1.84713252e-04
  1.98863047e-04  3.89326387e-04  3.94434795e-04  5.25339026e-04
 -7.02153563e-04  2.76297340e-04 -7.91757366e-05 -5.73060795e-04
  4.47374630e-04  4.39532757e-04  9.92842696e-06 -9.30383106e-04
 -2.84017309e-04  5.77997808e-04 -2.71358956e-04  5.30005296e-04
  8.24998848e-04 -7.73481241e-04  8.18110012e-06]
MAE_train=0.3975395021559066, MAPE_train=11890.576448524698
MAE_test =0.4136656948661201, MAPE_test =10199.132222077176
```

- K Optimiert KNN:

```
Results
KNN with K=50
MAE_train=0.3959200583377023, MAPE_train=27.582812093954935
MAE_test =0.40379400794010906, MAPE_test =24.703090318224948
```

- Durch Optimieren des Polynomgrades bis Grad = 15 wurde keien nennenswerte verbesserung erzielt.

c)

- Optimierung mit LSR Regression:

```
 3.09799978e-02  3.48737004e-02  8.15542845e-03  3.46280417e-02
 1.01558287e-01  2.25004308e-02  3.27882824e-02 -5.13703986e-02
-3.64794903e-03  4.30026128e-03 -1.39639702e-02 -2.64356969e-02
-4.88931903e-02  1.79784791e-02  5.42610434e-02 -6.48977987e-03
 6.62817376e-02  4.98093166e-02 -3.64697574e-02  6.43201165e-03
 3.91453322e-02  2.15818866e-02  3.44286509e-02  3.04336178e-02
-1.82571966e-02  6.94588199e-02  8.25470265e-03  3.24147830e-02
-2.60326471e-02  3.17315800e-02 -1.62751691e-02 -3.05532031e-02
 4.76815677e-03 -3.57984709e-02 -3.45555588e-02  2.18955344e-03
-3.40301688e-03 -3.20034197e-02 -8.02083122e-02 -8.48529738e-03
 4.94803330e-02 -2.52578899e-02 -2.02431899e-02 -7.49531856e-02
 6.35986527e-03  9.02037463e-02 -1.62957673e-01 -4.22398971e-02
-1.23619286e-01 -1.83261864e-02 -9.11681102e-03 -2.55811412e-02
-1.00979790e-02 -1.63994048e-02 -7.64593069e-03 -3.11427703e-02
 5.03603264e-02  4.66126305e-02  4.76470916e-02 -4.97015660e-03
 1.32520429e-01  5.56340938e-02 -1.27687022e-02  3.47619055e-03
-2.44173926e-02 -3.00095173e-02 -8.94457550e-02  1.06628486e-02
-4.26538832e-02 -5.71490458e-02 -4.05612535e-02  1.43689644e-02
-8.35219735e-02  4.24504831e-02 -1.05057297e-02 -2.53033673e-02
-8.70950785e-02 -5.95859684e-02 -6.70305944e-02 -1.27552137e-02
 3.30556490e-03  1.71744756e-02 -5.69422301e-02  9.03159354e-03
-7.54320496e-03 -2.36333325e-03 -4.26364836e-02  3.82074258e-02
 5.17458367e-04 -7.81237423e-02 -2.59477944e-02  1.78148516e-02
 9.75463138e-03  6.78504570e-02  1.32832342e-02 -1.19868686e-01
-4.08979041e-02  5.13933825e-02  2.52896345e-02 -1.37745228e-01
-1.11030425e-02 -5.30147682e-02 -1.00836447e-01  1.87233734e-02
-6.44519745e-02 -1.47806295e-02  4.86205823e-02  1.64564904e-02
-5.77647121e-03 -1.06671648e-02 -5.64027593e-02  6.54193678e-02
 1.81814134e-02 -5.81409164e-02 -6.62165360e-02 -6.39244453e-02
-1.34215029e-02 -1.38390521e-01 -6.79757069e-02  1.31612463e-02
 9.84460305e-03  5.54680870e-03  7.02660987e-02  5.15521030e-02
 3.67476043e-02 -8.92078165e-03  1.04012421e-01  1.40634287e-01
-1.15348071e-01  1.39893675e-01 -2.56990014e-02 -3.78164037e-03
 8.03402943e-02  3.25301660e-02 -9.91892126e-02  1.89428289e-01
-7.41933664e-03 -2.56399894e-02  6.93163214e-02  1.40055995e-02
-8.17977156e-02 -7.67661445e-02 -1.18082934e-01  4.12319239e-04
-2.00802272e-02 -2.84412036e-02 -6.25562331e-02 -5.39930610e-03
 2.04398559e-02 -1.65339924e-01  2.05467627e-02 -3.71719911e-02
 1.29767793e-01  1.97498571e-02 -9.61162062e-02 -1.78726925e-02
 6.82108509e-02 -1.06809556e-01  7.53202421e-02  1.40537348e-01
-1.57779118e-02  9.84027725e-02 -4.80632935e-02 -2.70940067e-02
-4.35562537e-02  4.79173665e-02 -3.88921983e-02  1.50515428e-01
 6.52460277e-02 -6.13321760e-04  1.02334579e-01  1.10606201e-01
-8.60299818e-02 -4.22587453e-02  1.03768918e-02  6.51570820e-02
 1.57890028e-02 -5.22489909e-02  1.80684811e-02  1.37462820e-01
-1.62289947e-01 -8.56316556e-03  3.43829040e-02  2.14459787e-02
-4.89663093e-02 -1.27919524e-01  1.20295559e-01  2.58522388e-02
-1.11751915e-01  9.44338073e-03  1.19549675e-02 -9.21055397e-02
-1.17451572e-01 -7.21435330e-02 -1.11919460e-02  4.78723346e-02
 3.27141006e-02 -7.64186793e-03  1.87288402e-01 -2.99275851e-03
-1.83868407e-02 -2.01855366e-01 -9.91707582e-03 -1.48807823e-02
-9.95514940e-02 -2.26607081e-02 -1.45193081e-03 -1.23887125e-03
-1.22152755e-02 -9.42353224e-03]
number of basisfunctions M= 462
IV.1) Some predictions on the training data:
Prediction for X[ 1250 ]= [8.00000e+03 0.00000e+00 1.01600e-01 7.13000e+01 1.21072e-03]  is y= [125.81360032] , whereas true value is T[ 1250 ]= 125.465
Prediction for X[ 215 ]= [4.00000e+02 0.00000e+00 2.28600e-01 3.96000e+01 2.53511e-03]  is y= [124.0743118] , whereas true value is T[ 215 ]= 123.565
Prediction for X[ 313 ]= [1.60000e+04 4.00000e+00 2.28600e-01 7.13000e+01 4.00603e-03]  is y= [112.55068253] , whereas true value is T[ 313 ]= 112.768
Prediction for X[ 285 ]= [4.00000e+02 2.00000e+00 2.28600e-01 3.17000e+01 3.72371e-03]  is y= [126.03226081] , whereas true value is T[ 285 ]= 123.417
Prediction for X[ 1122 ]= [2.50000e+03 1.27000e+01 2.54000e-02 7.13000e+01 1.21808e-02]  is y= [135.3537284] , whereas true value is T[ 1122 ]= 134.058
IV.2) Some predictions for new test vectors:
Prediction for x_test_1 is y= [130.04401162]
Prediction for x_test_2 is y= [131.75557132]
IV.3) S= 3 fold Cross Validation:
MAE= 1.6101356751117588 MAPE= 0.013011600533743123
```

```
from polynomial_basis_functions import *
from Regression import *

# ***** MAIN PROGRAM ********
# (I) Hyper-Parameters
# (I.a) Hyper-Parameters for evaluation
seed=42           # define seed for random number generator
modeltype='lsr'   # define which model to evaluate (either 'lsr' or 'knn')
S=3               # do S-fold cross-validation
N_pred=5;         # number of predictions on the training set for testing
x_test_1 = [1250,11,0.2,69.2,0.0051];   # test vector 1
x_test_2 = [1305,8,0.1,57.7,0.0048];    # test vector 2
# (I.b) Hyper-Parameters for linear regression
deg=6             # degree of basis function polynomial phi(x)
lmbda=3           # regularization parameter (lambda>0 avoids also singularities)      You, 1 second ago • Uncommitted changes
flagSTD=1         # if >0 then standardize data before training (i.e., scale X to mean value 0 and standard deviation 1)
eps=0.05          # parameter to recognize badly conditioned matrixes
```

- Optimierung mit KNN Regression:

```
#### KNN regression with K= 1 , flagKLinReg= 1  ####
IV.1) Some predictions on the training data:
Prediction for X[ 680 ]= [5.00000e+02 9.90000e+00 1.52400e-01 3.17000e+01 2.52785e-02]  is y= [127.179] , whereas true value is T[ 680 ]= 127.179
Prediction for X[ 293 ]= [2.50000e+03 2.00000e+00 2.28600e-01 3.17000e+01 3.72371e-03]  is y= [121.407] , whereas true value is T[ 293 ]= 121.407
Prediction for X[ 158 ]= [5.00000e+02 4.00000e+00 3.04800e-01 7.13000e+01 4.97773e-03]  is y= [130.715] , whereas true value is T[ 158 ]= 130.715
Prediction for X[ 1341 ]= [6.30000e+03 3.30000e+00 1.01600e-01 3.17000e+01 2.51435e-03]  is y= [120.702] , whereas true value is T[ 1341 ]= 120.702
Prediction for X[ 258 ]= [5.00000e+02 2.00000e+00 2.28600e-01 5.55000e+01 3.13525e-03]  is y= [124.692] , whereas true value is T[ 258 ]= 124.692
IV.2) Some predictions for new test vectors:
Prediction for x_test_1 is y= [136.191]
Prediction for x_test_2 is y= [132.252]
IV.3) S= 3 fold Cross Validation:
MAE= 1.981358189081225 MAPE= 0.016112775478578 65
```

```
# (I.c) Hyper-Parameters for KNN regression
K=1               # K for K-Nearest Neighbors
flagKLinReg = 1   # if flag==1 and K>=D then do a linear regression of the KNNs to make prediction
lr_deg=5          # degree of basis function polynomials for KNN-regression
lr_lambda=5       # regularization parameter (lambda>0 avoids also singularities)      You, 1 second ago • Uncommitted changes
lr_flagSTD=1      # if >0 then standardize data before training (i.e., scale X to mean value 0 and standard deviation 1)
lr_eps=0.05       # parameter to recognize badly conditioned matrixes
```

Die Ergebnisse der KNN Regression waren besser.