# V2A1_LinearRegression

December 17, 2018

## 1 Aufgabe 1: (10+9+14+12 = 45 Punkte)

Thema: Lineare "Least-Squares" Regression mit Regularisierung in Python
Gegeben seien Daten {(xn, tn) | n = 1, ..., N } welche ursprünglich von der Parabel f (x ) = w0+ w1x +w2x
2 mit w0 = 2, w1 = 1, w2 = 3 gesampelt wurden, aber nun mit Rauschen behaftet sind. Zu diesen Daten soll ein lineares Regressionsmodell y = wT(x ) mit polynomiellen Basisfunktionen bestimmt werden.

**a) Betrachten Sie das Programmgerüst V2A1_LinearRegression.py aus dem Praktikumsverzeichnis:** Erklären Sie kurz in eigenen Worten (jeweils 1-2 Sätze) wozu die Funktionen fun_true(.), generateDataSet(.), getDataError(.) und phi_polynomial(.) dienen. Versuchen Sie den Python-Code zu verstehen (muss nicht dokumentiert werden).

- fun_true(X): berechnet für jedes Element vom X das ensprechende y nach der Parabelfunktion y=3*xš-x+2

- generateDataset(N,xmin,xmax,sd_noise): erstellt eine N groSSe Liste von x Werten mit dazugehörigen zielwerten (y) die aber mit einem rauschen gemischt werden

- getDataError(Y,T): berechnet die Fehlerquadratsumme für T und Y
- phi_polynomial(x,deg=1): berechnet den Merkmalsvektor für x bis zum grad deg(standartmäSSig 1)

Von welcher Funktion sind die Original-Daten (xn, tn) gesampelt?

- fun_true(X) / t=3*xš-x+2

Wie lauten die Basisfunktionen j(x ) für j = 1, ...,deg des linearen Modells?

- = x^5 + x^4 + x^3 + x^2 + x + 1

Welche Rolle hat die Variable lmbda?

- lmbda ist der Regularisierungsparameter

Worin unterscheiden sich die Variablen X,T von X_test,T_test?

- X,T haben die gleichen Parameter wie X_test, T_test sind aber mit anderen Zufallswerten erstellt worden

Was stellen im Plot die grünen Kreuze/Punkte, grüne Kurve, rote Kurve dar?

- grüne Kreuze sind die Lerndaten
- grüne Punkte sind die Testdaten
- grüne Kurve ist die Ausgangsfunktion
- rote Kurve ist die von uns vorhergesagte Funktion

In [1]:
```python
# V2A1_LinearRegression.py
# Programmgeruest zu Versuch 2, Aufgabe 1
import numpy as np
import matplotlib.pyplot as plt

def fun_true(X):                              # compute 1-dim. parable function; X mus
    w2,w1,w0 = 3.0,-1.0,2.0                   # true parameters of parable y(x)=w0+w1*
    return w0+w1*X+w2*np.multiply(X,X)        # return function values (same size as X

def generateDataSet(N,xmin,xmax,sd_noise):    # generate data matrix X and target valu
    X=xmin+np.random.rand(N,1)*(xmax-xmin)    # get random x values uniformly in [xmin
    T=fun_true(X);                            # target values without noise
    if(sd_noise>0):
        T=T+np.random.normal(0,sd_noise,X.shape) # add noise
    return X,T

def getDataError(Y,T):       # compute data error (least squares) between prediction Y
    D=np.multiply(Y-T,Y-T); # squared differences between Y and T
    return 0.5*sum(sum(D)); #eine Summe zu viel?   E_D

def phi_polynomial(x,deg=1):     # compute polynomial basis function vector phi(x) for
    assert(np.shape(x)==(1,)), "currently only 1dim data supported"
    return np.array([x[0]**i for i in range(deg+1)]).T;
    # returns feature vector phi(x)=[1 x x**2 x**3 ... x**deg]

def predict(x,w):
    temp = np.array(sum([g*(x**i) for i,g in enumerate(w)]))
    return temp
```

In [2]:
```python
# (I) generate data
np.random.seed(10)                            # set seed of random generator (to be ab
N=10                                          # number of data samples
xmin,xmax=-5.0,5.0                            # x limits
sd_noise=10                                   # standard deviation of Guassian noise
X,T           = generateDataSet(N, xmin,xmax, sd_noise)             # generate trainin
X_test,T_test = generateDataSet(N, xmin,xmax, sd_noise)             # generate test da
print("X=",X, "T=",T)
```

```
X= [[ 2.71320643]
 [-4.79248051]
```

```
[ 1.33648235]
[ 2.48803883]
[-0.01492988]
[-2.75203354]
[-3.01937135]
[ 2.60530712]
[-3.30889163]
[-4.11660186]] T= [[24.02637686]
[76.78157398]
[ 6.06498717]
[16.33697066]
[ 6.34586048]
[39.50347318]
[22.71852474]
[30.04030926]
[40.44148448]
[61.40721056]]
```

In [3]:
```
# (II) generate linear least squares model for regression
lmbda=0                                                       # no regression
deg=5                                                         # degree of polynomi
N,D = np.shape(X)                                             # shape of data matr
N,K = np.shape(T)                                             # shape of target va
PHI = np.array([phi_polynomial(X[i],deg).T for i in range(N)])  # generate design ma
N,M = np.shape(PHI)                                           # shape of design ma
print("PHI=", PHI)
W_LSR = np.dot(np.linalg.pinv(PHI),T)
print("W_LSR=",W_LSR)
```

```
PHI= [[ 1.00000000e+00  2.71320643e+00  7.36148915e+00  1.99732397e+01
   5.41915225e+01  1.47032787e+02]
 [ 1.00000000e+00 -4.79248051e+00  2.29678694e+01 -1.10073066e+02
   5.27523025e+02 -2.52814381e+03]
 [ 1.00000000e+00  1.33648235e+00  1.78618507e+00  2.38720482e+00
   3.19045710e+00  4.26398961e+00]
 [ 1.00000000e+00  2.48803883e+00  6.19033720e+00  1.54017993e+01
   3.83202746e+01  9.53423310e+01]
 [ 1.00000000e+00 -1.49298770e-02  2.22901226e-04 -3.32788789e-06
   4.96849568e-08 -7.41790292e-10]
 [ 1.00000000e+00 -2.75203354e+00  7.57368863e+00 -2.08430452e+01
   5.73607595e+01 -1.57858734e+02]
 [ 1.00000000e+00 -3.01937135e+00  9.11660336e+00 -2.75264110e+01
   8.31124569e+01 -2.50947371e+02]
 [ 1.00000000e+00  2.60530712e+00  6.78762520e+00  1.76838483e+01
   4.60718559e+01  1.20031334e+02]
 [ 1.00000000e+00 -3.30889163e+00  1.09487638e+01 -3.62282731e+01
   1.19875430e+02 -3.96654807e+02]
```

```
 [ 1.00000000e+00 -4.11660186e+00  1.69464109e+01 -6.97616264e+01
   2.87180841e+02 -1.18220918e+03]]
W_LSR= [[ 7.06500519]
 [-3.86916089]
 [ 1.16066097]
 [ 0.27523414]
 [ 0.23060499]
 [ 0.02613605]]
```

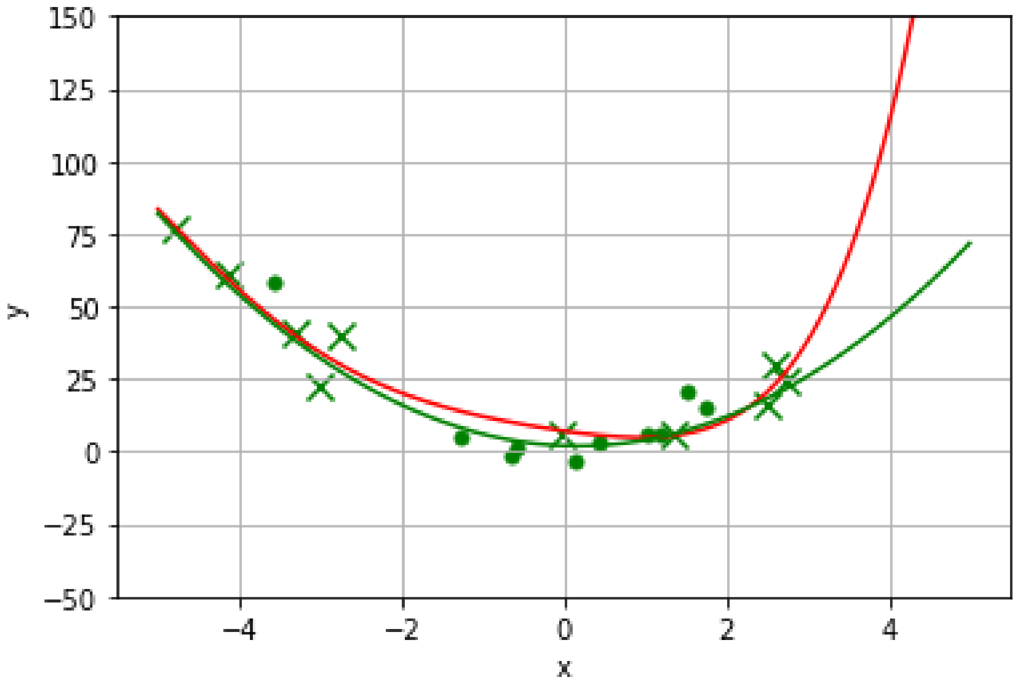
```
In [4]: # (III) make predictions for test data
        Y_test = np.array([predict(xt, W_LSR) for xt in X_test])
        Y_learn = np.array([predict(xt, W_LSR) for xt in X])
        print("Y_test=",Y_test)
        print("T_test=",T_test)
        print("learn data error = ", getDataError(Y_learn,T))
        print("test data error = ", getDataError(Y_test,T_test))
        print("W_LSR=",W_LSR)
        print("mean weight = ", np.mean(np.mean(np.abs(W_LSR))))
```

```
Y_test= [[ 5.65809158]
 [45.63529912]
 [13.77669052]
 [ 7.83896266]
 [ 9.67876039]
 [10.08485107]
 [ 5.07045944]
 [ 6.57739393]
 [ 6.18850946]
 [ 4.89231264]]
T_test= [[ 3.10905545]
 [57.97094574]
 [ 5.36688144]
 [15.48746047]
 [ 0.92351025]
 [-1.52698415]
 [ 6.31013154]
 [-2.84101855]
 [20.36655269]
 [ 6.00240429]]
learn data error =  151.66995610334058
test data error =  395.93589328574376
W_LSR= [[ 7.06500519]
 [-3.86916089]
 [ 1.16066097]
 [ 0.27523414]
 [ 0.23060499]
 [ 0.02613605]]
```

```
mean weight =  2.104467039777364
```

```
In [5]: # (IV) plot data
        ymin,ymax = -50.0,150.0                         # interval of y data
        x_=np.arange(xmin,xmax,0.01)                    # densely sampled x values
        Y_LSR = np.array([np.dot(W_LSR.T,np.array([phi_polynomial([x],deg)]).T)[0] for x in x_]
        # least squares prediction
        Y_true = fun_true(x_).flat

        fig = plt.figure()
        ax = fig.add_subplot(111)
        ax.scatter(X.flat,T.flat,c='g',marker='x',s=100)             # plot learning data poin
        ax.scatter(X_test.flat,T_test.flat,c='g',marker='.',s=100)   # plot test data points (g
        ax.plot(x_,Y_LSR.flat, c='r')         # plot LSR regression curve (red)
        ax.plot(x_,Y_true, c='g')             # plot true function curve (green)
        ax.set_xlabel('x')                    # label on x-axis
        ax.set_ylabel('y')                    # label on y-axis
        ax.grid()                             # draw a grid
        plt.ylim((ymin,ymax))                 # set y-limits
        plt.show()                            # show plot on screen
```



# V2A2_Regression

December 17, 2018

In [8]:
```python
#!/usr/bin/env python
# V2A2_Regression.py
# Programmgeruest zu Versuch 2, Aufgabe 2

import numpy as np
import scipy.spatial
from random import randint

# -------------------------------------------------------------------------------------
# base class for regressifiers
# -------------------------------------------------------------------------------------
class Regressifier:
    """
    Abstract base class for regressifiers
    Inherit from this class to implement a concrete regression algorithm
    """

    def fit(self,X,T):        # train/compute regression with lists of feature vectors
        """
        Train regressifier by training data X, T, should be overwritten by any derived
        :param X: Data matrix of size NxD, contains in each row a data vector of size
        :param T: Target vector matrix of size NxK, contains in each row a target vect
        :returns: -
        """
        pass

    def predict(self,x):      # predict a target vector given the data vector x
        """
        Implementation of the regression algorithm; should be overwritten by any deriv
        :param x: test data vector of size D
        :returns: predicted target vector
        """
        return None

    def crossvalidate(self,S,X,T,dist=lambda t: np.linalg.norm(t)):  # do a S-fold cro
        """
        Do a S-fold cross validation
```

```
:param S: Number of parts the data set is divided into
:param X: Data matrix (one data vector per row)
:param T: Matrix of target vectors; T[n] is target vector of X[n]
:param dist: a fuction dist(t) returning the length of vector t (default=Eukli
:returns (E_dist,sd_dist,E_min,E_max) : mean, standard deviation, minimum, and
:returns (Erel_dist,sdrel_dist,Erel_min,Erel_max) : mean, standard deviation, 
"""
X,T=np.array(X),np.array(T)                              # ensure array type
N=len(X)                                                 # N=number of data vectors
perm = np.random.permutation(N)                          # do a random permutation 
X1,T1=[X[i] for i in perm], [T[i] for i in perm]         # ... to get random partit
idxS = [range(i*N//S,(i+1)*N//S) for i in range(S)]      # divide data set into S p
E_dist,E_dist2,E_max,E_min=0,0,-1,-1                     # initialize first two mom
Erel_dist,Erel_dist2,Erel_max,Erel_min=0,0,-1,-1         # initialize first two mom
for idxTest in idxS:                                     # loop over all possible t
    # (i) generate training and testing data sets and train classifier
    idxLearn = [i for i in range(N) if i not in idxTest]
    if(S<=1): idxLearn=idxTest
    X_learn, T_learn = np.array([X1[i] for i in idxLearn]), np.array([T1[i] fo
    X_test , T_test  = np.array([X1[i] for i in idxTest ]), np.array([T1[i] fo
    self.fit(X_learn,T_learn)                            # train regressifier
    # (ii) test regressifier
    for i in range(len(X_test)):  # loop over all data vectors to be tested
        # (ii.a) regress for i-th test vector
        xn_test = X_test[i].T                                # data vector for test
        t_test = self.predict(xn_test)                       # predict target value
        # (ii.b) check for regression errors
        t_true = T_test[i].T                                 # true target value
        d=dist(t_test-t_true)                                # (Euklidean) distance
        dttrue=dist(t_true)                                  # length of t_true
        E_dist  = E_dist+d                                   # sum up distances (fo
        E_dist2 = E_dist2+d*d                                # sum up squared dista
        if(E_max<0)or(d>E_max): E_max=d                      # collect maximal erro
        if(E_min<0)or(d<E_min): E_min=d                      # collect minimal erro
        drel=d/dttrue
        Erel_dist  = Erel_dist+drel                          # sum up relative dist
        Erel_dist2 = Erel_dist2+(drel*drel)                  # sum up squared relat
        if(Erel_max<0)or(drel>Erel_max): Erel_max=drel       # collect maximal rela
        if(Erel_min<0)or(drel<Erel_min): Erel_min=drel       # collect minimal rela
E_dist      = E_dist/float(N)                                # estimate of first mo
E_dist2     = E_dist2/float(N)                               # estimate of second m
Var_dist    = E_dist2-E_dist*E_dist                          # variance of error
sd_dist     = np.sqrt(Var_dist)                              # standard deviation o
Erel_dist   = Erel_dist/float(N)                             # estimate of first mo
Erel_dist2  = Erel_dist2/float(N)                            # estimate of second m
Varrel_dist = Erel_dist2-Erel_dist*Erel_dist                 # variance of error
sdrel_dist  = np.sqrt(Varrel_dist)                           # standard deviation o
return (E_dist,sd_dist,E_min,E_max), (Erel_dist,sdrel_dist,Erel_min,Erel_max) 
```

```
                                                          # and maximum error (f
```

In [9]:
```
# -----------------------------------------------------------------------------------------
# DataScaler: scale data to standardize data distribution (for mean=0, standard deviat
# -----------------------------------------------------------------------------------------
class DataScaler:
    """
    Class for standardizing data vectors
    Some regression methods require standardizing of data before training to avoid num
    """

    def __init__(self,X):                # X is data matrix, where rows are data vector
        """
        Constructor: Set parameters (mean, std,...) to standardize data matrix X
        :param X: Data matrix of size NxD the standardization parameters (mean, std, .
        :returns: object of class DataScaler
        """
        self.meanX = np.mean(X,0)        # mean values for each feature column
        self.stdX  = np.std(X,0)         # standard deviation for each feature column
        if isinstance(self.stdX,(list,tuple,np.ndarray)):
            self.stdX[self.stdX==0]=1.0 # do not scale data with zero std (that is, co
        else:
            if(self.stdX==0): self.stdX=1.0   # in case stdX is a scalar
        self.stdXinv = 1.0/self.stdX     # inverse standard deviation


    def scale(self,x):                   # scales data vector x to mean=0 and std=1
        """
        scale data vector (or data matrix) x to mean=0 and s.d.=1
        :param x: data vector or data matrix
        :returns: scaled (standardized) data vector or data matrix
        """
        return np.multiply(x-self.meanX,self.stdXinv)

    def unscale(self,x):                 # unscale data vector x to original distributi
        """
        unscale data vector (or data matrix) x to original data ranges
        :param x: standardized data vector or data matrix
        :returns: unscaled data vector or data matrix
        """
        return np.multiply(x,self.stdX)+self.meanX

    def printState(self):
        """
        print standardization parameters (mean value, standard deviation (std), and in
        """
        print("mean=",self.meanX, " std=",self.stdX, " std_inv=",self.stdXinv)
```

In [10]:
```
from itertools import *
```

```
from functools import reduce
# ---------------------------------------------------------------------------------------
# function to compute polynomial basis functions
# ---------------------------------------------------------------------------------------
def phi_polynomial(x,deg=1):           # x should be list or np.array or 1xD matrix; 
    """
    polynomial basis function vector; may be used to transform a data vector x into a
    :param x: data vector to be transformed into a feature vector
    :param deg: degree of polynomial
    :returns phi: feature vector
    Example: phi_polynomial(x,3) returns for one-dimensional x the vector [1, x, x*x,
    """
    x=np.array(np.mat(x))[0]           # ensure that x is a 1D array (first row of x)
    D=len(x)
    #assert (D==1) or ((D>1) and (deg<=3)), "phi_polynomial(x,deg) not implemented fo
    if(D==1):
        phi = np.array([x[0]**i for i in range(deg+1)])
    else:
        phi = np.array(phi_helper(x.tolist(),deg))
        #phi = np.array([])
        #if(deg>=0):
        #    phi = np.concatenate((phi,[1]))      # include degree 0 terms
        #    if(deg>=1):
        #        phi = np.concatenate((phi,x))    # includes degree 1 terms
        #        if(deg>=2):
        #            for i in range(D):
        #                phi = np.concatenate(( phi, [x[i]*x[j] for j in range(i+1)] 
        #            if(deg>=3):
        #                for i in range(D):
        #                    for j in range(i+1):
        #                        phi = np.concatenate(( phi, [x[i]*x[j]*x[k] for k in
                    # EXTEND CODE HERE FOR deg>3!!!!
    return phi.T  # return basis function vector (=feature vector corresponding to da

def phi_helper(x, deg):
    if deg <= 0:
        return [1]
    else:
        ret = phi_helper(x, deg-1)
        ret += [reduce(lambda a,b:a*b,combi) for combi in combinations_with_replacemen
        return ret
```

In [11]:
```
# ---------------------------------------------------------------------------------------
# Least Squares (ML) linear regression with sum of squares Regularization,
# ---------------------------------------------------------------------------------------
class LSRRegressifier(Regressifier):
    """
    Class for Least Squares (or Maximum Likelihood) Linear Regressifier with sum of s
```

```
    """

    def __init__(self,lmbda=0,phi=lambda x: phi_polynomial(x,1),flagSTD=0,eps=0.000001
        """
        Constructor of class LSRegressifier
        :param lmbda: Regularization coefficient lambda
        :param phi: Basis-functions used by the linear model (default linear polynomia
        :param flagSTD: If >0 then standardize data X and target values T (to mean 0 a
        :param eps: maximal residual value to tolerate (instead of zero) for numerica
        :returns: -
        """
        self.lmbda=lmbda       # set regression parameter (default 0)
        self.phi=phi           # set basis functions used for linear regression (defa
        self.flagSTD=flagSTD;  # if flag >0 then data will be standardized, i.e., sca
        self.eps=eps;          # maximal residual value to tolerate (instead of zero)


    def fit(self,X,T,lmbda=None,phi=None,flagSTD=None): # train/compute LS regression
        """
        Train regressifier (see lecture manuscript, theorem 3.11, p33)
        :param X: Data matrix of size NxD, contains in each row a data vector of size
        :param T: Target vector matrix of size NxK, contains in each row a target vec
        :param lmbda: Regularization coefficient lambda
        :param phi: Basis-functions used by the linear model (default linear polynomia
        :param flagSTD: If >0 then standardize data X and target values T (to mean 0 a
        :returns: flagOK: if >0 then all is ok, otherwise matrix inversion was bad co
        """
        # (i) set parameters
        if lmbda==None: lmbda=self.lmbda       # reset regularization coefficient?
        if phi==None: phi=self.phi             # reset basis functions?
        if flagSTD==None: flagSTD=self.flagSTD # standardize data vectors?
        # (ii) scale data for mean=0 and s.d.=0 ?
        if flagSTD>0:                          # if yes, then...
            self.datascalerX=DataScaler(X)     # create datascaler for data matrix X
            self.datascalerT=DataScaler(T)     # create datascaler for target matrix
            X=self.datascalerX.scale(X)        # scale all features (=columns) of dat
            T=self.datascalerT.scale(T)        # ditto for target matrix T
        # (iii) compute weight matrix and check numerical condition
        flagOK,maxZ=1,0;                       # if <1 then matrix inversion is numer
        try:
            self.N,self.D = X.shape            # data matrix X has size N x D (N is n
            self.M = self.phi(self.D*[0]).size # get number of basis functions
            #self.K = T.shape[1]                # DELTE dummy code (just required for
            PHI = np.array([np.transpose(phi(x)) for x in X])                        #
            PHIT_PHI_lmbdaI = np.add(np.dot(PHI.T,PHI),np.dot(self.lmbda,np.identity(s
            #print(PHIT_PHI_lmbdaI)
            # REPLACE dummy code: compute PHI_T*PHI+lambda*I
            PHIT_PHI_lmbdaI_inv = np.linalg.inv(PHIT_PHI_lmbdaI)          # REPLACE du
```

```
            self.W_LSR = np.dot(np.dot(PHIT_PHI_lmbdaI_inv,PHI.T),T)# REPLACE dummy c
            # (iv) check numerical condition
            Z=np.dot(PHIT_PHI_lmbdaI,PHIT_PHI_lmbdaI_inv)-np.identity(self.M)  # REPL
            maxZ = np.max(Z)    # REPLACE dummy code: Compute maximum (absolute) comp
            assert maxZ<=self.eps              # maxZ should be <eps for good conditi
        except:
            flagOK=0;
            print("EXCEPTION DUE TO BAD CONDITION:flagOK=", flagOK, " maxZ=", maxZ, "
            raise
        return flagOK

    def predict(self,x,flagSTD=None):      # predict a target value given data vector
        """
        predicts the target value y(x) for a test vector x
        :param x: test data vector of size D
        :param flagSTD: If >0 then standardize data X and target values T (to mean 0
        :returns: predicted target vector y of size K
        """
        if flagSTD==None: flagSTD=self.flagSTD      # standardazion?
        if flagSTD>0: x=self.datascalerX.scale(x)   # if yes, then scale x before com
        phi_of_x = self.phi(x)                      # compute feature vector phi_of_x
        y=np.dot(self.W_LSR.T, phi_of_x)                    # REPLACE dummy code:  co
        if flagSTD>0: y=self.datascalerT.unscale(y) # scale prediction back to origin
        return y                  # return prediction y for data vector x
```

In [12]:
```
# ------------------------------------------------------------------------------------
# KNN regression
# ------------------------------------------------------------------------------------
class KNNRegressifier(Regressifier):
    """
    Class for fast K-Nearest-Neighbor-Regression using KD-trees
    """

    def __init__(self,K,flagKLinReg=0):
        """
        Constructor of class KNNRegressifier
        :param K: number of nearest neighbors that are used to compute prediction
        :flagKLinReg: if >0 then the do a linear (least squares) regression on the th
                      otherwise just take the mean of the K nearest neighbors target
        :returns: -
        """
        self.K = K                                 # K is number of nearest-neighbors
        self.X, self.T = [],[]                     # initially no data is stored
        self.flagKLinReg=flagKLinReg               # if flag is set then do a linear

    def fit(self,X,T): # train/compute regression with lists of data vectors X and ta
        """
        Train regressifier by stroing X and T and by creating a KD-Tree based on X
```

```
        :param X: Data matrix of size NxD, contains in each row a data vector of size
        :param T: Target vector matrix of size NxK, contains in each row a target vec
        :returns: -
        """
        self.X, self.T = np.array(X),np.array(T)   # just store feature vectors X and
        self.N, self.D = self.X.shape              # store data number N and dimensio
        self.kdtree = scipy.spatial.KDTree(self.X) # do an indexing of the feature ve

    def predict(self,x,K=None,flagKLinReg=None):   # predict a target value given dat
        """
        predicts the target value y(x) for a test vector x
        :param x: test data vector of size D
        :param K: number of nearest neighbors that are used to compute prediction
        :flagKLinReg: if >0 then the do a linear (least squares) regression on the th
                      otherwise just take the mean of the K nearest neighbors target
        :returns: predicted target vector of size K
        """
        if(K==None): K=self.K                      # do a K-NN search...
        if(flagKLinReg==None): flagKLinReg=self.flagKLinReg # if flag >0 then do a re
        nn = self.kdtree.query(x,K)                # get indexes of K nearest neighbo
        if K==1: idxNN=[nn[1]]                     # cast nearest neighbor indexes nn
        else: idxNN=nn[1]
        t_out=0
        if(self.flagKLinReg==0):
            # just take mean value of KNNs
            t_out=np.mean([self.T[i] for i in idxNN])
        else:
            # do a linear regression of the KNNs
            lsr=LSRRegressifier(lmbda=0.0001,phi=lambda x:phi_polynomial(x,1),flagSTD=
            lsr.fit(self.X[idxNN],self.T[idxNN])
            t_out=lsr.predict(x)
        return t_out
```

In [13]:
```
# *******************************************************
# __main___
# Module test
# *******************************************************

if __name__ == '__main__':
    print("\n-----------------------------------------")
    print("Example: 1D-linear regression problem")
    print("-----------------------------------------")
    # (i) generate data
    N=100
    w0,w1=4,2                  # parameters of line
    X=np.zeros((N,1))          # x data: allocate Nx1 matrix as numpy ndarray
    X[:,0]=np.arange(0,50.0,50.0/N)  # equidistant sampling of the interval [0,50)
    T=np.zeros((N,1))          # target values: allocate Nx1 matrix as numpy ndarray
```

```
sd_noise = 1.0             # noise power (=standard deviation)
T=T+w1*X+w0 + np.random.normal(0,sd_noise,T.shape)  # generate noisy target value
par_lambda = 0             # regularization parameter
print("X=",X)
print("T=",T)

# (ii) define basis functions (phi should return list of basis functions; x shoul
deg=2;                                    # degree of polynomial
phi=lambda x: phi_polynomial(x,1)         # define phi by polynomial basis-functions u
print("phi(4)=", phi([4]))                 # print basis function vector [1, x, x*x ..
print("phi([1,2])=", phi([1,2]))           # print basis function vector for two-dim.

# (iii) compute LSR regression
print("\n-----------------------------------------")
print("Do a Least-Squares-Regression")
print("-----------------------------------------")
lmbda=0;
lsr = LSRRegressifier(lmbda,phi)
lsr.fit(X,T)
print("lsr.W_LSR=",lsr.W_LSR)            # weight vector (should be approximately
x=np.array([3.1415]).T
print("prediction of x=",x,"is y=",lsr.predict(x))

# do S-fold crossvalidation
S=3
err_abs,err_rel = lsr.crossvalidate(S,X,T)
print("LSRRegression cross-validation: absolute errors (E,sd,min,max)=",
      err_abs, "  relative errors (E,sd,min,max)=", err_rel)

# (iv) compute KNN-regression
print("\n-----------------------------------------")
print("Do a KNN-Regression")
print("-----------------------------------------")
K=2;
knnr = KNNRegressifier(K)
knnr.fit(X,T)
print("prediction of x=",x,"is y=",knnr.predict(x))

# do S-fold crossvalidation
err_abs,err_rel = knnr.crossvalidate(S,X,T)
print("KNNRegression cross-validation: absolute errors (E,sd,min,max)=",
      err_abs, "  relative errors (E,sd,min,max)=", err_rel)
```

```
-----------------------------------------
Example: 1D-linear regression problem
-----------------------------------------
X= [[ 0. ]
```

```
[ 0.5]
[ 1. ]
[ 1.5]
[ 2. ]
[ 2.5]
[ 3. ]
[ 3.5]
[ 4. ]
[ 4.5]
[ 5. ]
[ 5.5]
[ 6. ]
[ 6.5]
[ 7. ]
[ 7.5]
[ 8. ]
[ 8.5]
[ 9. ]
[ 9.5]
[10. ]
[10.5]
[11. ]
[11.5]
[12. ]
[12.5]
[13. ]
[13.5]
[14. ]
[14.5]
[15. ]
[15.5]
[16. ]
[16.5]
[17. ]
[17.5]
[18. ]
[18.5]
[19. ]
[19.5]
[20. ]
[20.5]
[21. ]
[21.5]
[22. ]
[22.5]
[23. ]
[23.5]
[24. ]
```

```
[24.5]
[25. ]
[25.5]
[26. ]
[26.5]
[27. ]
[27.5]
[28. ]
[28.5]
[29. ]
[29.5]
[30. ]
[30.5]
[31. ]
[31.5]
[32. ]
[32.5]
[33. ]
[33.5]
[34. ]
[34.5]
[35. ]
[35.5]
[36. ]
[36.5]
[37. ]
[37.5]
[38. ]
[38.5]
[39. ]
[39.5]
[40. ]
[40.5]
[41. ]
[41.5]
[42. ]
[42.5]
[43. ]
[43.5]
[44. ]
[44.5]
[45. ]
[45.5]
[46. ]
[46.5]
[47. ]
[47.5]
[48. ]
```

```
 [48.5]
 [49. ]
 [49.5]]
T= [[  6.83201747]
 [  4.08753737]
 [  6.66932205]
 [  7.77531959]
 [  8.29691136]
 [  8.0218027 ]
 [  9.13060012]
 [ 10.32787321]
 [  9.86455198]
 [ 13.78764613]
 [ 14.66120483]
 [ 13.27876243]
 [ 17.10096036]
 [ 17.69254971]
 [ 20.60392182]
 [ 19.0507613 ]
 [ 20.54543181]
 [ 19.78929123]
 [ 20.91112176]
 [ 21.56557997]
 [ 22.99757597]
 [ 26.23070818]
 [ 26.84318957]
 [ 27.48143519]
 [ 27.3222863 ]
 [ 28.09751753]
 [ 29.38404981]
 [ 31.3053832 ]
 [ 32.94912426]
 [ 33.48066847]
 [ 31.87618172]
 [ 36.55110537]
 [ 36.67673421]
 [ 37.38079867]
 [ 37.00558168]
 [ 37.85882417]
 [ 40.35743309]
 [ 39.53798979]
 [ 41.12924118]
 [ 43.95232222]
 [ 44.38233112]
 [ 45.29789303]
 [ 45.54222387]
 [ 47.05345409]
 [ 47.07882018]
```

```
[ 50.38219467]
[ 49.58466477]
[ 49.36192899]
[ 50.95947156]
[ 53.00053505]
[ 53.25900426]
[ 55.96974707]
[ 55.82858277]
[ 55.23260012]
[ 57.77691964]
[ 58.05220333]
[ 60.59824475]
[ 60.35286317]
[ 62.55073264]
[ 63.90530768]
[ 63.38698507]
[ 66.29524218]
[ 64.90806921]
[ 67.21906604]
[ 68.13540154]
[ 69.51499327]
[ 67.91592015]
[ 72.65167049]
[ 72.71331263]
[ 75.48551458]
[ 72.9123328 ]
[ 74.92762616]
[ 75.43926589]
[ 76.42930322]
[ 77.72172722]
[ 80.28641515]
[ 79.3840556 ]
[ 83.22774385]
[ 81.74324371]
[ 81.72859136]
[ 84.1200039 ]
[ 86.01590555]
[ 86.78027845]
[ 85.99492472]
[ 87.46847964]
[ 89.59251584]
[ 88.72671359]
[ 90.90614838]
[ 92.84699912]
[ 93.63308904]
[ 94.45192742]
[ 94.43275285]
[ 95.18242137]
```

```
 [ 95.37177015]
 [ 97.80576852]
 [ 99.01510883]
 [ 98.79215104]
 [ 99.36836094]
 [102.67741711]
 [103.86062374]]
phi(4)= [1 4]
phi([1,2])= [1 1 2]

-----------------------------------------
Do a Least-Squares-Regression
-----------------------------------------
lsr.W_LSR= [[4.00338443]
 [1.99729795]]
prediction of x= [3.1415] is y= [10.27789596]
LSRRegression cross-validation: absolute errors (E,sd,min,max)= (0.9010614762127104, 0.6375557

-----------------------------------------
Do a KNN-Regression
-----------------------------------------
prediction of x= [3.1415] is y= 9.729236660642488
KNNRegression cross-validation: absolute errors (E,sd,min,max)= (1.1576012228411576, 0.8334887
```

## 0.1 a) Versuchen Sie zunächst den Aufbau des Moduls V2A2_Regression.py zu verstehen:

Betrachten Sie den Aufbau des Moduls durch Eingabe von pydoc V2A2_Regressifier.
Welche Klassen gehören zu dem Modul und welchen Zweck haben sie jeweils?

- DataScaler: Standatisiert Daten
- Regressifier: Abstrakte Klasse für Regressierer
- KNNRegressifier: ermöglicht Regression mithilfe von Fast-KNN
- LSRRegressifier: ermöglicht Regression mithilfe von Fehlerquadratsummen

Betrachten Sie nun die Basis-Klasse Regressifier im Quelltext: Wozu dienen jeweils die Methoden fit(self,X,T), predict(self,x) und crossvalidate(self,S,X,T) ?

- fit(self,X,T): Verlangt nach der implimentierung einer Methode die den Regressierer mit X und T trainiert.
- predict(self,x): Verlangt nach der implimentierung einer Methode die auf x eine Regression anwendet und eine Vorhersage zurückliefert
- crossvalidate(self,S,X,T): Macht S-fache Kreuzvaliedierung mit den Daten,Labeln (X,T) und liefert als Ergebniss Informationen über die relativen und absoluten Fehlerwerte

Worin unterscheidet sich crossvalidate(.) von der entsprechenden Methode für Klassifikation (siehe vorigen Versuch)?

- hier kann man noch angeben mit welcher Längenfunktion gearbeitet werden soll, und es werden andere Fehlerstatistiken zurückgegeben

b) Betrachten Sie nun die Funktion phi_polynomial(x,deg):
Was berechnet die Funktion? Welches Ergebnis liefert phi_polynomial([3],5)? Welches Ergebnis liefert phi_polynomial([3,5],2)?

```
In [14]: phi_polynomial([3],5)

Out[14]: array([  1,   3,   9,  27,  81, 243])

In [15]: phi_polynomial([3,5],2)

Out[15]: array([ 1,  3,  5,  9, 15, 25])
```

- phi_polynomial(x,deg): berechnet die Basisfunktion für die Werte x mit dem grad deg

Geben Sie eine allgemeine Formel an für das Ergebnis von phi_polynomial([x1,x2],2)?

- [1, x1, x2, (x1)š, x1*x2, (x2)š]

Wozu braucht man diese Funktion im Zusammenhang mit Regression?

- Das Kreuzprodukt aus dieser Funktion und dem Zielwertevektor geben die Prognose bei der Regression

Bis zu welchem Polynomgrad kann die Funktion Basisfunktionen berechnen? Erweitern Sie die Funktion mindestens bis Grad 5.

- momentan bis grad 3, siehe Verbesserung im code

```
In [16]: phi_polynomial([3,5],5)

Out[16]: array([   1,    3,    5,    9,   15,   25,   27,   45,   75,  125,   81,
                 135,  225,  375,  625,  243,  405,  675, 1125, 1875, 3125])
```

c) Betrachten Sie die Klasse LSRRegressifier:
Welche Art von Regressions-Modell berechnet diese Klasse?

- Fehlerquadratsummenregression mit Regularisierung

Wozu dienen jeweils die Parameter lmbda, phi, flagSTD und eps ?

- lmbda: Regularisierungsparameter
- phi: Basisfunktion
- flagSTD: gibt an ob die Daten Standatisiert werden sollen
- eps: höchster erlaubter Fehlerrestwert

Welche Rolle spielt hier die Klasse DataScaler? In welchen Methoden und zu welchem Zweck werden die Daten ggf. umskaliert? Welches Problem kann auftreten wenn man dies nicht tut? Wozu braucht man die Variablen Z und maxZ in der Methode fit(.)?

- DataScaler wird in fit() und predict() verwendet um Daten zu Standartisieren
- nicht scalierte Daten werde können kleinste abweichungen in Gewichten zu groSSen und/oder unterschiedlichen Änderungen führen.
- Z ist die Matrix mit den Fehlerwerten die durch Invertierung entstehen.
- maxZ ist der gröSSte Wert aus Z und wird verwendet um zu bestimmen ob der Fehlerwert annehmbar ist

Vervollständigen Sie die Methoden fit(self,X,T,...) und predict(self,x,...) (vgl. vorige Aufgabe).

- siehe code

d) Betrachten Sie die Klasse KNNRegressifier:
Welche Art von Regressions-Modell berechnet diese Klasse?

- fast K-Nearest-Neighbor-Regression

Wozu dienen jeweils die Parameter K und flagKLinReg?

- K: Anzahl der NN die verwendet werden um eine Vorhersage zu treffen
- flagKLinReg: gibt an ob eine Regression verwendet werden soll oder nur der Mittelwert aus den NN genommen wird

Beschreiben Sie kurz in eigenen Worten (2-3 Sätze) auf welche Weise die Prädiktion y(x) berechnet wird.

- Es werden die KNNs für x ermittelt. Damit wird ein LSRRegressifier trainiert, der dann nach einer prediction gefragt wird.

e) Betrachten Sie abschlieSSend den Modultest:
Beschreiben Sie kurz was im Modultest passiert.

- es wird ein Linearer Datensatz erstellt
- die Funktion phi wird neu definiert und auf Grad 2 festgesetzt
- LSR Regression wird durchgeführt und Kreuzvalidiert
- KNN Regression wird durchgeführt und Kreuzvalidiert

Welche Gewichte W werden gelernt? Wie lautet also die gelernte Prädiktionsfunktion? Welche Funktion sollte sich idealerweise (für N  ) ergeben?
w0 = [3.78705442]
w1 = [2.01036841]
y = 3.787 + 2.010*x
y = 4 + 2*x für N
Welche Ergebnisse liefert die Kreuzvalidierung? Was bedeuten die Werte?

- (E,sd,min,max)= (0.8335277318740052, 0.6423712068147653, 0.030450286307427632, 3.059745154084382)
- E ist der mittlere Fehlerwert
- sd ist die Standartabweichung der Fehler
- min ist der kleinste Fehlerwert
- max ist der gröSSte Fehlerwert

Vergleichen und Bewerten Sie die Ergebnisse von Least Squares Regression gegenüber der KNN-Regression (nach Optimierung der Hyper-Parameter , K, ...).

LSR: absolute errors (E,sd,min,max)= (0.7210491003592423, 0.5070165487566235, 0.0066007843635347285, 2.780939336314603)
KNN: absolute errors (E,sd,min,max)= (1.1743092886650586, 0.9170722237516987, 0.0038023455924474092, 4.2882465799825695)
LSR: relative errors (E,sd,min,max)= (0.02240702968734219, 0.028216719179408916, 0.0001525998169486947, 0.14635559861181296)
KNN: relative errors (E,sd,min,max)= (0.05354319692710508, 0.13202983166413382, 5.0979643788569236e-05, 1.0603171407681475)

- LSR scheint in fast allen Bereichen wesentlich bessere Ergebnisse als KNNR zu liefern.

# V2A3_regression_airfoilnoise

December 17, 2018

In [3]:
```
#!/usr/bin/env python
# V2A3_regression_airfoilnoise.py
# Programmgeruest zu Versuch 2, Aufgabe 3
# to log outputs start with: python V2A3_regression_airfoilnoise.py >V2A3_regression_a

import numpy as np
import pandas as pd

from V2A2_Regression import *


# ***** MAIN PROGRAM ********
# (I) Hyper-Parameters
S=3;                  # S-fold cross-validation
lmbda=1;              # regularization parameter (lambda>0 avoids also singularities)
K=13;                  # K for K-Nearest Neighbors
flagKLinReg = 1;      # if flag==1 and K>=D then do a linear regression of the KNNs to ma
deg=5;                # degree of basis function polynomials
flagSTD=1;            # if >0 then standardize data before training (i.e., scale X to mea
N_pred=5;             # number of predictions on the training set for testing
x_test_1 = [1250,11,0.2,69.2,0.0051];   # REPLACE dummy code: define test vector 1
x_test_2 = [1305,8,0.1,57.7,0.0048];   # REPLACE dummy code: define test vector 2
```

In [4]:
```
# (II) Load data
fname='./AirfoilSelfNoise/airfoil_self_noise.xls'
airfoil_data = pd.read_excel(fname,0); # load data as pandas data frame
T = airfoil_data.values[:,5]           # target values = noise load (= column 5 of data
X = airfoil_data.values[:,:5]          # feature vectors (= column 0-4 of data table)
N,D=X.shape                            # size and dimensionality of data set
idx_perm = np.random.permutation(N)    # get random permutation for selection of test
print("Data set ",fname," has size N=", N, " and dimensionality D=",D)
print("X=",X)
print("T=",T)
print("x_test_1=",x_test_1)
print("x_test_2=",x_test_2)
print("number of basis functions M=", len(phi_polynomial(X[1],deg)))
```

```
Data set  ./AirfoilSelfNoise/airfoil_self_noise.xls  has size N= 1502  and dimensionality D= 5
```

```
X= [[1.00000e+03 0.00000e+00 3.04800e-01 7.13000e+01 2.66337e-03]
 [1.25000e+03 0.00000e+00 3.04800e-01 7.13000e+01 2.66337e-03]
 [1.60000e+03 0.00000e+00 3.04800e-01 7.13000e+01 2.66337e-03]
 ...
 [4.00000e+03 1.56000e+01 1.01600e-01 3.96000e+01 5.28487e-02]
 [5.00000e+03 1.56000e+01 1.01600e-01 3.96000e+01 5.28487e-02]
 [6.30000e+03 1.56000e+01 1.01600e-01 3.96000e+01 5.28487e-02]]
T= [125.201 125.951 127.591 ... 106.604 106.224 104.204]
x_test_1= [1250, 11, 0.2, 69.2, 0.0051]
x_test_2= [1305, 8, 0.1, 57.7, 0.0048]
number of basis functions M= 252
```

```
In [5]: # (III) Do least-squares regression with regularization
        print("\n#### Least Squares Regression with regularization lambda=", lmbda, " ####")
        lsr = LSRRegressifier(lmbda=lmbda,phi=lambda x: phi_polynomial(x,deg),flagSTD=flagSTD)
        lsr.fit(X,T)
        print("lsr.W_LSR=",lsr.W_LSR)
        print("III.1) Some predictions on the training data:")
        for i in range(N_pred):
            n=idx_perm[i]
            print("Prediction for X[",n,"]=",X[n]," is y=",lsr.predict(X[n]),", whereas true va
        print("III.2) Some predicitions for new test vectors:")
        print("Prediction for x_test_1 is y=", lsr.predict(x_test_1))    # REPLACE dummy code:
        print("Prediction for x_test_2 is y=", lsr.predict(x_test_2))    # REPLACE dummy code:
        print("III.3) S=",S,"fold Cross Validation:")
        err_abs,err_rel = lsr.crossvalidate(S,X,T)                      # REPLACE dummy code: do c
        print("absolute errors (E,sd,min,max)=", err_abs, "\nrelative errors (E,sd,min,max)=",
```

```
#### Least Squares Regression with regularization lambda= 1  ####
lsr.W_LSR= [-3.94646520e-01 -1.80155527e+00 -3.96662974e-01 -8.64281233e-01
  3.26438812e-01 -6.21770966e-01  6.48905815e-01 -7.71134930e-01
  2.95779376e-01 -8.67132337e-02  3.61560493e-01 -2.81989365e-02
 -3.64074945e-02  1.43998983e-01  4.77503076e-02 -3.63380016e-02
 -9.73944497e-02 -3.71505203e-02 -1.53013579e-01 -2.47759041e-01
  1.29242314e-01  3.13172340e-01  1.50772922e-01  8.40072758e-01
 -8.44083280e-02  1.32532606e+00  2.52907215e-01  2.08504016e-01
 -2.03801432e-01  5.74149325e-01  3.06551563e-01 -2.34531355e-01
  4.66271349e-01 -2.43145107e-01  1.06213081e-02  2.84851888e-01
 -8.48806231e-02 -2.00727798e-01 -6.93365225e-02 -5.88853578e-02
  7.29728355e-02 -1.36528243e-01 -2.26202702e-01 -1.28035094e-01
 -2.24863325e-01 -1.13338956e-01  4.19411521e-01 -1.68848150e-02
  9.78093460e-02 -9.27714500e-02 -1.60258656e-02 -3.22486764e-01
  1.01134491e-01  7.97108235e-03 -1.81313744e-01 -3.69616798e-01
 -2.47146762e-01  2.02432625e-01 -4.38061153e-01  3.64617492e-02
 -7.59308984e-01  1.95668256e-01  6.70564131e-01 -1.47283279e-01
 -2.77532541e-01 -3.08022282e-01  1.19842567e-02 -7.66976903e-02
```

```
  2.18511621e-02  1.72857687e-01 -6.25981195e-01  1.36043453e-01
  6.68313146e-02 -1.65130385e-02 -5.46617990e-02  5.48724195e-02
 -5.19208304e-02 -7.63188408e-02  3.78906795e-02  1.94056398e-01
 -2.78668584e-01 -7.05574660e-01  8.01753250e-05 -1.97576222e-01
  5.53327040e-02 -2.03772994e-02 -1.47505902e-01  5.91246025e-02
 -1.79737621e-02 -7.12236378e-02 -2.34113997e-01 -4.01567558e-02
  2.90612583e-02 -7.81051692e-02  2.06254670e-05  5.56183281e-02
  4.80864131e-03  2.42168243e-02 -1.15039749e-01  1.75335472e-02
  2.94441955e-02 -2.98387467e-01  2.24608964e-03 -5.67996318e-02
  3.60431186e-03  3.50473096e-02 -9.11562735e-02 -2.58145605e-02
  3.76595743e-02  1.50183200e-01 -3.75077570e-02  1.27138401e-02
  1.16133919e-01 -2.18208523e-01  1.41690489e-01  1.29547055e-01
  1.02847461e-01  7.46744531e-03 -5.50688491e-02 -1.09489486e-01
  4.33124764e-02  3.52862834e-02  1.02885915e-01  1.58937930e-01
  3.22368270e-02  1.33278437e-01  1.54200823e-02  1.03812366e-02
  2.68478094e-02 -4.69455614e-03 -1.03924197e-01  3.63370895e-01
  1.95941884e-01  8.55311399e-03 -6.75202423e-01  6.17886875e-02
 -5.57216536e-03 -6.85068836e-01  9.73106644e-04  8.93303864e-04
  3.32687011e-01  1.03800671e-01  2.21151780e-01 -1.24168329e-01
  5.08255859e-02 -6.41651342e-02 -3.57687689e-02  1.84616236e-01
 -5.38713351e-02  2.16353496e-01 -4.49026953e-02  5.07394826e-02
 -1.58277599e-03 -3.52975947e-02 -2.32615544e-02  1.11961868e-01
 -3.17638241e-01  3.66642352e-03  1.01689005e-01 -9.14031375e-02
  1.07835193e-01 -5.52562467e-02 -1.69661381e-01  1.65517660e-02
 -1.62822814e-01 -3.68756103e-01  8.04196574e-02 -8.18975854e-02
 -1.12357085e-01  3.69551139e-02 -5.07037615e-02 -1.69293973e-01
  1.78704521e-01  2.25341988e-01 -8.85750241e-02  5.14764795e-02
  1.97391087e-01  5.44295165e-02  2.08205031e-01  8.63666478e-03
  2.50397522e-01  2.12918833e-01  3.01972410e-02 -4.05213862e-03
 -1.89041169e-02 -1.59662947e-01  2.21658174e-01  2.82466967e-02
  8.76627247e-02  4.79010683e-02  9.35769244e-02  9.70424870e-02
 -2.11928518e-02 -2.45208682e-02 -2.29960413e-02 -1.97186002e-02
  1.40300710e-01  2.80593592e-01  1.01183417e-01  7.86300171e-03
 -1.36568507e-01  1.93395520e-01  1.49290958e-02  9.97525775e-02
 -5.93850011e-02 -3.88941947e-02 -2.24188695e-01  1.14703293e-01
 -8.61448667e-02 -3.94278300e-02 -3.86775078e-02 -1.38123752e-01
  3.28920405e-02 -1.46146334e-01 -8.05652042e-02 -9.70461716e-02
  2.15697923e-01  3.68525314e-02  2.31497755e-01 -2.69752137e-02
  1.55073486e-02  1.84839607e-01  8.53004635e-03  5.65295263e-02
 -2.07214477e-01 -2.92021485e-02  4.13387816e-03  3.20040301e-02
 -6.48052853e-02 -9.35343805e-02 -1.63259658e-02 -1.15971880e-01
 -5.33499384e-02 -1.74711442e-01 -7.62523429e-02 -3.15365640e-03
 -3.13599999e-02  1.47577802e-02  8.98079738e-02  5.76990605e-02
  3.07022562e-02  6.10550828e-02  4.30451202e-02  2.50332155e-02
 -1.23965738e-01 -1.06664213e-01 -3.90944224e-02  3.18752489e-02
 -5.83388127e-02  6.63287979e-03  8.53339681e-02  2.11106178e-02]
III.1) Some predictions on the training data:
Prediction for X[ 642 ]= [5.00000e+03 9.90000e+00 1.52400e-01 7.13000e+01 1.93001e-02]  is y=
```

```
Prediction for X[ 6 ]= [4.00000e+03 0.00000e+00 3.04800e-01 7.13000e+01 2.66337e-03]  is y= 12
Prediction for X[ 1359 ]= [4.00000e+02 6.70000e+00 1.01600e-01 3.96000e+01 5.78076e-03]  is y=
Prediction for X[ 990 ]= [1.00000e+04 0.00000e+00 2.54000e-02 3.96000e+01 4.28464e-04]  is y= 
Prediction for X[ 953 ]= [8.0000e+02 1.9700e+01 5.0800e-02 3.9600e+01 3.6484e-02]  is y= 123.9
III.2) Some predicitions for new test vectors:
Prediction for x_test_1 is y= 130.45406757371268
Prediction for x_test_2 is y= 133.1060885540002
III.3) S= 3 fold Cross Validation:
absolute errors (E,sd,min,max)= (2.1022709165494167, 2.63711268032694, 0.000984334197639214, 4
relative errors (E,sd,min,max)= (0.016946838227214888, 0.021770475968140576, 8.455680284846053
```

```
In [ ]: lmbdaRange = [1,2,5,10,30,60,100]
        degRange = list(range(1,8))
        LSRErrors = []

        for lmb in lmbdaRange:
            for d in degRange:
                # (III) Do least-squares regression with regularization
                print("\n#### Least Squares Regression with regularization lambda=", lmb, " de
                lsr = LSRRegressifier(lmbda=lmb,phi=lambda x: phi_polynomial(x,d),flagSTD=flag
                lsr.fit(X,T)
                print("lsr.W_LSR=",lsr.W_LSR)    # REPLACE dummy code: print weight vector for
                #print("III.1) Some predictions on the training data:")
                #for i in range(N_pred):
                    #n=idx_perm[i]
                    #print("Prediction for X[",n,"]=",X[n]," is y=",lsr.predict(X[n]),", where
                #print("III.2) Some predicitions for new test vectors:")
                #print("Prediction for x_test_1 is y=", lsr.predict(x_test_1))    # REPLACE du
                #print("Prediction for x_test_2 is y=", lsr.predict(x_test_2))    # REPLACE du
                print("III.3) S=",S,"fold Cross Validation:")
                err_abs,err_rel = lsr.crossvalidate(S,X,T)                   # REPLACE dummy co
                LSRErrors.append((lmb, d, err_abs, err_rel))
                print("absolute errors (E,sd,min,max)=", err_abs, "\nrelative errors (E,sd,min
```

```
In [7]: # (IV) Do KNN regression
        print("\n#### KNN regression with flagKLinReg=", flagKLinReg, " ####")
        knnr = KNNRegressifier(K,flagKLinReg)                                   # REPLACE dumm
        knnr.fit(X,T)
        print("IV.1) Some predictions on the training data:")
        for i in range(N_pred):
            n=idx_perm[i]
            print("Prediction for X[",n,"]=",X[n]," is y=",knnr.predict(X[n]),", whereas true 
        print("IV.2) Some predicitions for new test vectors:")
        print("Prediction for x_test_1 is y=", knnr.predict(x_test_1))    # REPLACE dummy code
        print("Prediction for x_test_2 is y=", knnr.predict(x_test_2))    # REPLACE dummy code
        print("IV.3) S=",S,"fold Cross Validation:")
        err_abs,err_rel = knnr.crossvalidate(S,X,T)                   # REPLACE dummy code: do
        print("absolute errors (E,sd,min,max)=", err_abs, "\nrelative errors (E,sd,min,max)=",
```

```
#### KNN regression with flagKLinReg= 1  ####
IV.1) Some predictions on the training data:
Prediction for X[ 642 ]= [5.00000e+03 9.90000e+00 1.52400e-01 7.13000e+01 1.93001e-02]  is y= 1
Prediction for X[ 6 ]= [4.00000e+03 0.00000e+00 3.04800e-01 7.13000e+01 2.66337e-03]  is y= 12
Prediction for X[ 1359 ]= [4.00000e+02 6.70000e+00 1.01600e-01 3.96000e+01 5.78076e-03]  is y=
Prediction for X[ 990 ]= [1.00000e+04 0.00000e+00 2.54000e-02 3.96000e+01 4.28464e-04]  is y= 1
Prediction for X[ 953 ]= [8.0000e+02 1.9700e+01 5.0800e-02 3.9600e+01 3.6484e-02]  is y= 127.1
IV.2) Some predicitions for new test vectors:
Prediction for x_test_1 is y= 126.56192024990972
Prediction for x_test_2 is y= 132.35085577388358
IV.3) S= 3 fold Cross Validation:
absolute errors (E,sd,min,max)= (3.1014230510756793, 3.1881851528245053, 0.0020327675492239905
relative errors (E,sd,min,max)= (0.02490752287142648, 0.02601855286718329, 1.5901618890310798e
```

```
In [ ]: flagRange = [0,1]
        kRange = list(range(1,15))
        KNNErrors = []

        for f in flagRange:
            for k in kRange:
                # (IV) Do KNN regression
                print("\n#### KNN regression with flagKLinReg=", f," K=",k, " ####")
                knnr = KNNRegressifier(k,f)                                    # REPLACE dummy
                knnr.fit(X,T)
                #print("IV.1) Some predictions on the training data:")
                #for i in range(N_pred):
                 #    n=idx_perm[i]
                 #    print("Prediction for X[",n,"]=",X[n]," is y=",knnr.predict(X[n]),", wher
                #print("IV.2) Some predicitions for new test vectors:")
                #print("Prediction for x_test_1 is y=", knnr.predict(x_test_1))     # REPLACE d
                #print("Prediction for x_test_2 is y=", knnr.predict(x_test_2))     # REPLACE d
                print("IV.3) S=",S,"fold Cross Validation:")
                err_abs,err_rel = knnr.crossvalidate(S,X,T)                    # REPLACE dummy
                KNNErrors.append((f, k, err_abs, err_rel))
                print("absolute errors (E,sd,min,max)=", err_abs, "\nrelative errors (E,sd,min
```

a) Vervollständigen Sie das Programmgerüst V2A3_regression_airfoilnoise.py um eine Least-Squares-Regression auf den Daten zu berechnen. Optimieren Sie die HyperParameter um bei einer S = 3-fachen Kreuzvalidierung möglichst kleine Fehlerwerte zu erhalten.

Welche Bedeutung haben jeweils die Hyper-Parameter lmbda, deg, flagSTD?

- lmbda: Regularisierungs parameter
- deg: Ist der Grad er polynomiellen Basisfunktion
- flagSTD: gibt an ob die Daten standartisiert werden sollen

Was passiert ohne Skalierung der Daten (flagSTD=0) bei höheren Polynomgraden (achten Sie auf die Werte von maxZ)?

- EXCEPTION DUE TO BAD CONDITION:flagOK= 0 maxZ= 195590361182.93994 eps= 1e-06
- der Fehlerwert beim invertieren explodiert ohne die Skalierung

Geben Sie Ihre optimalen Hyper-Parameter sowie die resultierenden Fehler-Werte an.

- lambda=1, deg=5, flagSTD=1
- absolute errors (E,sd,min,max)= (2.2241976966993438, 3.6504292455503355, 0.0023069096170331704, 83.63604473632992)
- relative errors (E,sd,min,max)= (0.018030635646137383, 0.03132168189811296, 1.746150761488692e-05, 0.762838110293237)

Welche Prognosen ergibt Ihr Modell für die neuen Datenvektoren x_test_1=[1250,11,0.2,69.2,0.0051] bzw. x_test_2=[1305,8,0.1,57.7,0.0048] ?

- Prediction for x_test_1 is y= 130.45406757371268
- Prediction for x_test_2 is y= 133.1060885540002

Welchen Polynomgrad und wieviele Basisfunktionen verwendet Ihr Modell?

- Polynomgrad ist 5
- mit 6 Basisfunktionen

b) Vervollständigen Sie das Programmgerüst V2A3_regression_airfoilnoise.py um eine KNN-Regression auf den Daten zu berechnen. Optimieren Sie die Hyper-Parameter um bei einer S = 3-fachen Kreuzvalidierung möglichst kleine Fehlerwerte zu erhalten.
Welche Bedeutung haben jeweils die Hyper-Parameter K und flagKLinReg?

- K: anzahl der NN die zum Ergebniss beitragen
- flagKLinReg: gibt an ob die KNN noch in einen LSRegressifier gegeben werden oder ob nur der Durchschnitt der KNN berechnet wird.

Geben Sie Ihre optimalen Hyper-Parameter sowie die resultierenden Fehler-Werte an.

- flagKLinReg=1, K=13
- absolute errors (E,sd,min,max)= (3.0858901947930324, 3.0296772825440974, 0.006016325705218151, 32.13059666400561)
- relative errors (E,sd,min,max)= (0.02482629194750422, 0.024838560765987942, 4.864074982591945e-05, 0.2851617187841634)

Welche Prognosen ergibt Ihr Modell für die neuen Datenvektoren x_test_1=[1250,11,0.2,69.2,0.0051] bzw. x_test_2=[1305,8,0.1,57.7,0.0048] ?

- Prediction for x_test_1 is y= 126.56192024990972
- Prediction for x_test_2 is y= 132.35085577388358

c) Vergleichen Sie die beiden Modelle. Welches liefert die besseren Ergebnisse?
LSR: absolute errors (E,sd,min,max)= (2.2241976966993438, 3.6504292455503355, 0.0023069096170331704, 83.63604473632992)
KNN: absolute errors (E,sd,min,max)= (3.0858901947930324, 3.0296772825440974,

0.006016325705218151, 32.13059666400561)
absolute differences: (0.861692498, 0.620751963, 0.003709416, 51.505448072)
LSR: relative errors (E,sd,min,max)= (0.018030635646137383, 0.03132168189811296, 1.746150761488692e-05, 0.762838110293237)
KNN: relative errors (E,sd,min,max)= (0.02482629194750422, 0.024838560765987942, 4.864074982591945e-05, 0.2851617187841634) relative differences: (0.006795656, 0.006483121, 0.000031179, 0.477676392

- KNN scheint die bessereren Ergebnisse zu liefern, verwendet intern aber auch LSR