V2NR1

December 14, 2018

```
In [1]: # coding: utf-8

        # In[1]:


        # a).1
        # fun_true(X): fun_true(X) bekommt einen Spaltenvektor Ãbergeben.
            # Sie initialisiert die Parameter w0,w1, und w2 von y(x). Sie berechnet y(x) und g
        # generateDataSet(N, xmin, xmax, sd_noise): generateDataSet(.) berechnet den Datenvekt
            # Um y(x) zu berechnen, wird fun_true aufgerufen. Falls sd_noise positiv ist,
            # wird T mit zufÃďlligen Werten innerhalb der Standardabweichung kumuliert. genera
        # getDataError(Y,T): Berechnet die Abweichung zwischen der Prognose Y und den erzielte
            # DafÃr werden die Abweichungen der Matrizen multipliziert, sodass sie positiv wer
            # Die Ergebnisse werden aufsummiert und halbiert. Dies ergibt die Daten der kleins
        # phi_polynomial(x, deg=1): phi_polynomial(.) bekommt x und ein Grad uebergeben, der i
            # Sie prueft ob x noch ein Zeilen-Vektor ist. Sie gibt ein Array (Zeilenvektor) vo
            # Zum Schluss wird der Vektor transponiert, dass es ein Spaltenvektor ist.

        # a).2
        # Die Funktion fun_true(X): sampelt die Originaldaten (xn, tn), da aus den Werten y(x)
        # Siehe Aufgabenbeschreibung "die Werte wurden durch die Parabel f(x) = .. gesampelt."

        # a).3
        # phi(x) = [1, x, x**2, x**3, ..., x**deg], da Zufallswerte verwendet werden, kann das

        # phi1 = [[ 1.00000000e+00  2.71320643e+00  7.36148915e+00  1.99732397e+01 5.41915225e
        # phi2 = [ 1.00000000e+00 -4.79248051e+00  2.29678694e+01 -1.10073066e+02 5.27523025e+
        # phi3 = [ 1.00000000e+00  1.33648235e+00  1.78618507e+00  2.38720482e+00 3.19045710e+
        # phi4 = [ 1.00000000e+00  2.48803883e+00  6.19033720e+00  1.54017993e+01 3.83202746e+
        # phi5 = [ 1.00000000e+00 -1.49298770e-02  2.22901226e-04 -3.32788789e-06 4.96849568e-
        # phi6 = [ 1.00000000e+00 -2.75203354e+00  7.57368863e+00 -2.08430452e+01 5.73607595e+
        # phi7 = [ 1.00000000e+00 -3.01937135e+00  9.11660336e+00 -2.75264110e+01 8.31124569e+
        # phi8 = [ 1.00000000e+00  2.60530712e+00  6.78762520e+00  1.76838483e+01 4.60718559e+
        # phi9 = [ 1.00000000e+00 -3.30889163e+00  1.09487638e+01 -3.62282731e+01 1.19875430e+
        # phi10 = [ 1.00000000e+00 -4.11660186e+00  1.69464109e+01 -6.97616264e+01 2.87180841e

        # a).4
        # lambda regularisiert die Least-Squares. Da lambda im angegebenen Fall 0 ist fÃďllt d
```

```
#a).5
# X, T sind die Trainingsdaten, X_test, T_test sind die Testdaten.
# Die Testdaten werden vorbehalten wÃďhrend des Trainierens, somit kann durch mehrmali
# Testen verwendet werden. Im hier gezeigten Fall, werden beide Ãber eine Random-Funkt

T_test=
[[ 3.10905545]
 [57.97094574]
 [ 5.36688144]
 [15.48746047]
 [ 0.92351025]
 [-1.52698415]
 [ 6.31013154]
 [-2.84101855]
 [20.36655269]
 [ 6.00240429]]


T= [[24.02637686]
 [76.78157398]
 [ 6.06498717]
 [16.33697066]
 [ 6.34586048]
 [39.50347318]
 [22.71852474]
 [30.04030926]
 [40.44148448]
 [61.40721056]]
```

```
  File "<ipython-input-1-b413f7d180fb>", line 46
T_test=
       ^
SyntaxError: invalid syntax
```

In [2]:
```
# 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) bet
    D=np.multiply(Y-T,Y-T);                   # squared differences between Y and T
    return 0.5*sum(sum(D));                   # return least-squares data error functi

def phi_polynomial(x,deg=1):                            # compute polynomial basis fun
    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

# (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)

# (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.zeros((M,1))                                            # REPLACE THIS BY R
W_LSR = np.dot(np.linalg.inv(np.dot(np.dot(PHI.T,PHI),lmbda*np.ones(PHI.shape))),PHI.T


print ("W_LSR=",W_LSR)

# (III) make predictions for test data
Y_test = np.zeros((N,1))   # REPLACE THIS BY PROGNOSIS FOR TEST DATA X_test! (result s
Y_learn = np.zeros((N,1))  # REPLACE THIS BY PROGNOSIS FOR TEST DATA X_test! (result s
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))))
```

```
    # (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_]
    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 (
    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
```

```
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]]
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]]
```

```
    ---------------------------------------------------------------------------

    ValueError                                Traceback (most recent call last)

    <ipython-input-2-623d0885ce5f> in <module>()
     42
     43 #W_LSR = np.zeros((M,1))                                        # REPLACE THIS I
---> 44 W_LSR = np.dot(np.linalg.inv(np.dot(np.dot(PHI.T,PHI),lmbda*np.ones(PHI.shape))),PH
     45
     46


    ValueError: shapes (6,6) and (10,6) not aligned: 6 (dim 1) != 10 (dim 0)
```