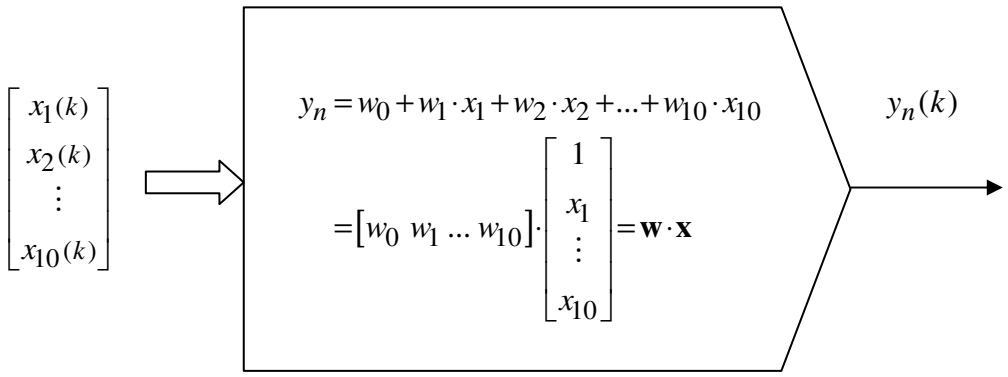
Data table

x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	yr
0.938948	0.243093	0.769404	0.110383	0.156845	0.070877	0.1133	0.468423	0.908125	-0.029232	1.363828
0.858192	0.185143	0.277638	0.446841	0.224112	0.110766	0.908541	0.209305	0.039953	0.481035	2.50465
0.913169	0.679671	0.171727	0.145886	0.477173	0.178838	-0.050086	0.026866	0.65981	0.64509	2.539099
0.956202	0.690921	0.554162	0.268243	0.383943	0.806728	0.978656	0.804882	0.673728	0.244051	2.699222
0.121536	0.314912	0.200431	0.093541	1.007575	0.771705	0.962618	0.916332	0.310904	0.395996	4.144885
0.217779	0.019734	0.881248	0.138186	0.501707	0.908214	0.679793	0.781976	0.392	0.316534	2.947219
0.120049	0.194485	0.499842	0.598249	0.527284	0.347768	0.067808	0.334744	0.667015	0.225257	1.458025
0.623391	0.549367	0.369276	0.775635	0.91393	-0.028653	0.333911	0.311915	0.385872	0.61114	4.821473
0.032171	0.236646	0.535352	0.103832	0.649027	0.633811	0.337705	0.51361	0.448676	0.795526	2.952753
0.294063	0.313644	0.731106	0.590844	0.789145	0.401911	0.74254	0.658666	1.103206	0.844122	3.103353
0.478286	0.151792	0.041067	0.761708	0.366838	0.314084	0.190706	0.877019	0.512109	0.124324	1.68367
0.631489	0.230648	0.715444	0.349493	0.614473	0.763165	0.109325	0.382724	0.156662	0.126445	2.899392
0.806412	0.145061	0.631143	0.178867	0.321745	0.433122	0.401611	0.933013	0.470253	0.119989	1.732479
0.875793	0.540398	0.720481	0.619636	0.284927	0.176698	0.121433	0.805642	1.023521	0.50093	2.424994
0.412426	0.187727	0.809584	0.630268	0.237916	0.584403	0.990486	0.423169	1.130321	0.712785	0.90443
0.076364	-0.030667	0.171511	1.098578	0.12352	0.340231	0.474652	0.105786	0.869858	0.980607	0.324889
0.229099	0.151545	0.879586	0.431847	0.676925	0.789163	0.06521	0.695888	0.154726	0.908567	3.243479
0.433329	0.416715	0.350979	0.569875	0.545078	0.312518	0.006975	0.375214	0.569668	0.595154	1.987566
0.556047	0.354277	0.925665	0.659707	0.303062	-0.019456	0.830501	0.579271	0.678314	0.545306	2.652519
1.106849	0.866208	0.508005	0.326271	0.358131	0.741534	0.36291	0.542272	0.403131	0.248913	1.936171
0.275372	0.653191	0.252415	0.108216	0.665622	0.866019	0.188994	0.636321	0.14304	0.056565	2.427732
0.054082	0.283505	0.737116	0.699752	0.166797	0.522295	0.569217	0.536383	0.26545	0.735396	1.345915
0.609995	0.489743	0.565919	0.237086	0.587457	0.616491	0.663779	0.109538	0.739676	0.604031	3.390541
0.800178	0.456974	0.951377	0.701188	1.025867	0.652106	0.699813	0.457901	0.300399	0.40337	4.221941
0.350688	0.458679	0.106048	0.058689	0.029924	0.256845	-0.058279	0.710634	0.567069	0.819259	1.83208
0.638661	0.172422	0.488099	0.199916	0.262988	0.155804	1.021443	0.086078	0.67862	0.624662	1.898901
0.884227	0.509434	0.607919	0.307157	0.406615	0.14473	0.48399	0.61505	0.656454	0.727122	3.366291
0.86772	0.001682	0.010741	0.504718	0.844167	0.612642	0.57629	0.212787	0.534508	-0.05615	2.891543
0.591463	0.986787	0.363205	0.633209	0.320309	0.063433	0.990416	0.273634	0.828862	0.362536	2.18687
0.83661	0.981698	0.708655	1.047123	0.016394	0.640411	0.486836	0.558144	0.638569	0.86762	2.324188
…	…	…	…	…	…	…	…	…	…	…

Ntrain of training data

Ntest of testing data



In python: yn[k]=sum(w*x)

k … sample index (discrete time, row in the data table)

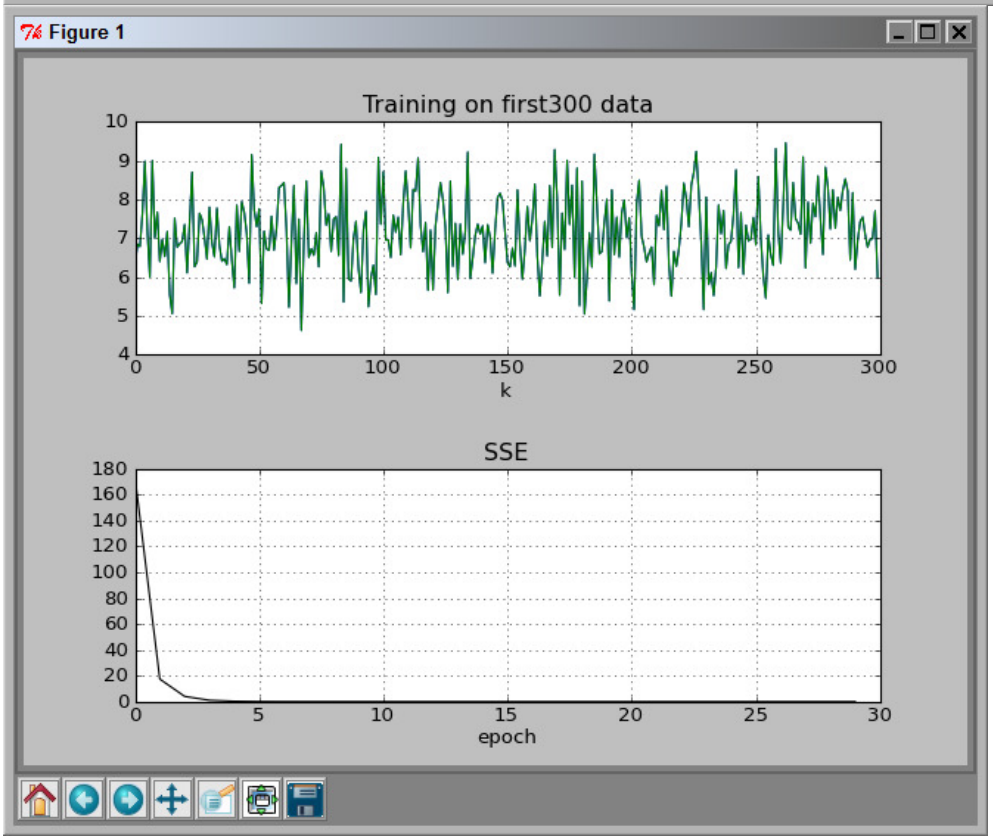
GD_Stat_Lin_10inputs_Adaptation_and_Testing.py - D:\TEACHING\SBS\2011_2012\LS\Lectures\Python\devel\GD_Stat_Lin_in_Matrix_Train_Test\GD_Stat_Lin_10in...

File Edit Format Run Options Windows Help

```
## Another ideal data and ideal model example where linear data are estimated by linear model
## no noise in data (very idealistic case)

from numpy import *
from numpy.random import rand, randn
from matplotlib.pyplot import *

### save data file to your working folder from
### http://www.fs.cvut.cz/~bukovsky/SBS/en/data/Linear_Static_data_Xy.txt

### importing data,

data= loadtxt('Linear_Static_data_Xy.txt')  #  y=f(x1,x2,...,x10), first 10 columns are x1 .. x10

### we are going to use linear model
### yn=w0+w1*x1+w2*x2+..+w10*x10

Ntrain=300;   # length of training data

yr=data[:,-1] # data

#=============================================================================
### Training by gradient descent adaptation (sample-by-sample)
#---------setup------
w=rand(11)    # 11 weights
mu=0.1        # learning rate
epochs=30     # how many times to adapt over all data, for training
#--------init--------
x=zeros((11)) # input vector to a model x=[1 x1(k) x2(k) ... x10(k)]
yn=zeros(Ntrain)   # model output
e=zeros(Ntrain)    # error e=yr-yn
SSE=zeros(epochs) # sum of square errors
yr=data[0:Ntrain,-1] # first N rows of last column  ... targets to which we train the model

for epoch in range(epochs):
    for k in range (0,Ntrain):
        # building x
        x[0]=1
        x[1:11]=data[k,0:10]
        yn[k]=sum(w*x) # sum(w*x)=w0+w1*x1+w2*x2+..+w10*x10
        e[k]=yr[k]-yn[k]
```

```
        w=w+mu*e[k]*x

    SSE[epoch]=sum(e*e)
    print(SSE[epoch])

figure()
subplot(211)
plot(yr),title('Training on first' + str(Ntrain) + ' data')
plot(yn,'g'),xlabel('k'),grid()

subplot(212)
plot (SSE,'k'), xlabel('epoch'),title('SSE'),grid()

#=============================================================================
### Testing ... calculating model ouptuts also for the remaining (non-training) data
#---------setup------
N=data.shape[0]  # for all data N=1012
#--------init--------
yn=zeros(N)
e=zeros(N)
for k in range (N): # for all data (training + testing)
    x[0]=1
    x[1:11]=data[k,0:10]
    yn[k]=sum(w*x)
    e[k]=data[k,-1]-yn[k]

figure()
subplot(211)
plot(data[:,-1]),title('Testing, first ' + str(Ntrain)+' samples were also training data')
plot(yn,'g')
plot(data[0:Ntrain,-1],'*')
xlabel('k'),grid()

subplot(212)
plot(e,'r'),title('error')
xlabel('k'),grid()
show()
```

Ln: 50 Col: 31



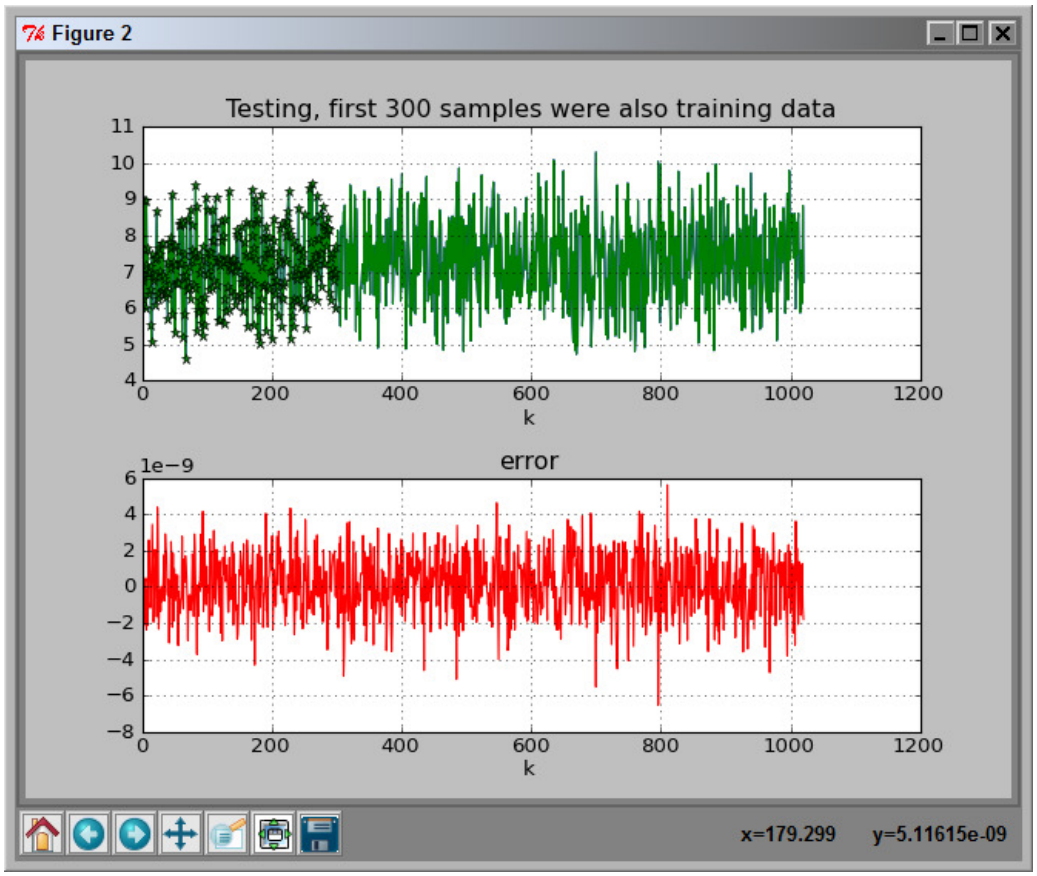


Figure 2
Testing, first 300 samples were also training data
11
10
9
8
7
6
5
4
0
200
400
600
800
1000
1200
k
error
1e−9
6
4
2
0
−2
−4
−6
−8
0
200
400
600
800
1000
1200
k
x=179.299
y=5.11615e-09