

---

# Programming Assignment I

---

CS5691: PATTERN RECOGNITION AND MACHINE LEARNING

TEAM 17 CCS SECTION SPRING 2021

SEPTEMBER 23, 2022

AANAND KRISHNAN  
BE17B001

MANORANJAN J  
NA17B112

RENEETH KRISHNA MG  
BS17B025

# Contents

<b>I</b>	<b>Polynomial Curve Fitting</b>	<b>2</b>
I.1	<a href="#">Python Code</a> . . . . .	2
<b>II</b>	<b>Linear Model for Regression using Polynomial Basis Functions</b>	<b>4</b>
II.1	<a href="#">Python Code</a> . . . . .	4
<b>III</b>	<b>Linear Model for Regression using Gaussian Basis Functions</b>	<b>6</b>
III.1	<a href="#">Python Code</a> . . . . .	6
<b>IV</b>	<b>Direct links For Python Codes</b>	<b>16</b>

# I Polynomial Curve Fitting

## I.1 Python Code

---

```
1 import pandas as pd
2 import numpy as np
3 from mpl_toolkits.mplot3d import Axes3D
4 import matplotlib.pyplot as plt
5 import random
6 from matplotlib import cm
7
8 def process_dataset_1(degree, lamb, batch):
9
10     train = int(0.7*batch)    #70% of data
11     valid = int(0.2*batch)    #20 % of data
12     test = int(batch-(train+valid)) #10 % of data
13
14     #generating polynomial combinations
15     def polybasisfun(vec,deg):
16         def degcomb(vec,deg):
17             if deg == 0:
18                 return [1]
19             if deg == 1:
20                 return vec
21             if len(vec) == 1:
22                 return [vec[0]**deg]
23             u = []
24             for i in range(deg+1):
25                 u+=([(vec[0]**(i))*x for x in (degcomb(vec[1:],deg-i))])
26             return u
27         u = np.array([1])
28         for i in range(1,deg+1):
29             u=np.append(u,degcomb(vec,i))
30         return u
31
32     # Read data from csv file
33     f0 = pd.read_csv('datasets/function1.csv')
34     odata = f0.to_numpy()
35     np.random.shuffle(odata)
36
37     # split into train,validate,test
38     data = odata[:train,1:2]
39     validata = odata[-(valid):,1:2]
40     testdata = odata[-(valid+test):- (valid),1:2]
41     yd = odata[:train,2]
42     yv = odata[-(valid):,2]
43     yt = odata[-(valid+test):- (valid),2]
44
45     # formulate X matrix
46     X = np.apply_along_axis(polybasisfun,1,data,degree)
```

47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91

```
# calculate weights
w = ((np.linalg.inv((lamb*np.eye(np.shape(X)[1]))+((np.transpose(X))@X)))@((np.tr

# Get predictions
def output(v,deg,w):
    return (w@(np.transpose(polybasisfun(v,deg))))
yp = np.apply_along_axis(output,1,data,deg=degree,w=w)
yvp = np.apply_along_axis(output,1,validata,deg=degree,w=w)
ytp = np.apply_along_axis(output,1,testdata,deg=degree,w=w)

print('ERMS_training = %f'%(np.linalg.norm((yp-yd),2)/np.sqrt(train)))
print('ERMS_test = %f'%(np.linalg.norm((ytp-yt),2)/np.sqrt(test)))
print('ERMS_valid = %f'%(np.linalg.norm((yvp-yv),2)/np.sqrt(valid)))

x = np.linspace(min(data),max(data),50)
x = np.reshape(x,(x.shape[0],1))
y = np.apply_along_axis(lambda x:output(x,degree,w),1,x)

fig0 = plt.figure(figsize=(9,9))
plt.plot(x,y,color='red')
plt.scatter(data,yd,color='blue')
plt.title("degree = {} | lambda = {}".format(degree,lamb),fontsize = 20)
plt.xlabel('X-axis(x)')
plt.ylabel('Y-axis(y)')

return np.linalg.norm((yp-yd),2)/np.sqrt(train), np.linalg.norm((ytp-yt),2)/np.sq

##### Main Code #####

# # Call function to experiment dataset 1
Degree = [2,3,6,9]
lamb = [0,10e-2,10e-1,10,10e2]
batch = [10,200]

Erms = []
# Experiment with Degree
for l in lamb:
    erms = process_dataset_1(25, 1, 350)
    Erms.append(erms)
```

---

## II Linear Model for Regression using Polynomial Basis Functions

### II.1 Python Code

---

```
1 import pandas as pd
2 import numpy as np
3 from mpl_toolkits.mplot3d import Axes3D
4 import matplotlib.pyplot as plt
5 import random
6 from matplotlib import cm
7 #polynomial basis function regression
8 degree = 6
9 lamb = 0
10 batch = 500
11 train = int(0.7*batch) #70% of data
12 test = int(0.2*batch) #20 % of data
13 valid = batch-(train+test) #10 % of data
14
15
16 #generating polynomial combinations
17 def polybasisfun(vec,deg):
18     def degcomb(vec,deg):
19         if deg == 0:
20             return [1]
21         if deg == 1:
22             return vec
23         if len(vec) == 1:
24             return [vec[0]**deg]
25         u = []
26         for i in range(deg+1):
27             u+=([(vec[0]**(i))*x for x in (degcomb(vec[1:],deg-i))])
28         return u
29     u = np.array([1])
30     for i in range(1,deg+1):
31         u=np.append(u,degcomb(vec,i))
32     return u
33 f0 = pd.read_csv('function1_2d.csv')
34 odata = f0.to_numpy()
35 np.random.shuffle(odata)
36 data = odata[:train,1:3]
37 validdata = odata[-(valid):,1:3]
38 testdata = odata[-(valid+test):-(valid),1:3]
39 yv = odata[-(valid):,3]
40 yt = odata[-(valid+test):-(valid),3]
41 X = np.apply_along_axis(polybasisfun,1,data,degree)
42 yd = odata[:train,3]
43 w = ((np.linalg.inv((lamb*np.eye(np.shape(X)[1]))+((np.transpose(X))@X)))
44      @((np.transpose(X))@yd))
45
```

```

46 def output(v,deg,w):
47     return (w@(np.transpose(polybasisfun(v,deg))))
48
49 yp = np.apply_along_axis(output,1,data,deg=degree,w=w)
50 yvp = np.apply_along_axis(output,1,validata,deg=degree,w=w)
51 ytp = np.apply_along_axis(output,1,testdata,deg=degree,w=w)
52 print('ERMS_training = %f'%(np.linalg.norm((yp-yd),2)/np.sqrt(train)))
53 print('ERMS_test = %f'%(np.linalg.norm((ytp-yt),2)/np.sqrt(test)))
54 print('ERMS_valid = %f'%(np.linalg.norm((yvp-yv),2)/np.sqrt(valid)))
55 x = np.linspace(min(data[:,0]),max(data[:,0]),200)
56 y = np.linspace(min(data[:,1]),max(data[:,1]),200)
57 X,Y=np.meshgrid(x,y)
58 Z = (np.array([output([i,j],degree,w) for i,j in
59     zip(np.ravel(X),np.ravel(Y))])).reshape(Y.shape)
60 fig = plt.figure(figsize=(16,12))
61 ax = plt.axes(projection='3d')
62 ax.plot_surface(X,Y,Z,color='red', alpha=0.2,)
63 ax.scatter(data[:,0],data[:,1],yd,alpha=1,color='black',linewidths=1)
64 ax.set_xlabel('X-axis(x1)',fontsize = 15)
65 ax.set_ylabel('Y-axis(x2)',fontsize = 15)
66 ax.set_zlabel('Z-axis(y)',fontsize = 15)
67 ax.set_title("batch size = {} | degree = {} | lambda = {}".format(batch,degree,
68     lamb),fontsize = 20)
69 plt.show()
70
71 fig1 = plt.figure(figsize=(9,9))
72 plt.scatter(yd,yp,marker='x')
73 plt.title('training',fontsize=20)
74 plt.xlabel('target',fontsize=20)
75 plt.ylabel('predicted',fontsize=20)
76 fig2 = plt.figure(figsize=(9,9))
77 plt.scatter(yt,ytp,marker='x')
78 plt.title('test',fontsize=20)
79 plt.xlabel('target',fontsize=20)
80 plt.ylabel('predicted',fontsize=20)
81 fig3 = plt.figure(figsize=(9,9))
82 plt.scatter(yv,yvp,marker='x')
83 plt.title('validation',fontsize=20)
84 plt.xlabel('target',fontsize=20)
85 plt.ylabel('predicted',fontsize=20)

```

---

## III Linear Model for Regression using Gaussian Basis Functions

### III.1 Python Code

---

```
1  # -*- coding: utf-8 -*-
2
3
4  import numpy as np
5  import pandas as pd
6  import matplotlib.pyplot as plt
7
8  ##### Dataset 3 #####
9
10 def process_dataset_3(D, var, r):
11
12     """
13     D: No of gauss functions for fit
14     var: variance for fit
15     r: regularisation term
16     """
17     # Read txt file
18
19     f = open('datasets/2_music.txt', 'r')
20
21     length = 0
22     for line in f:
23
24         d = [float(i) for i in line.split(',')]
25
26         if length == 0:
27             data = np.array(d)
28             data = data[np.newaxis,:]
29         else:
30             d = np.array(d)
31             d = d[np.newaxis,:]
32             data = np.append(data,d,axis=0)
33         length = length+1
34
35     f.close()
36
37     data = pd.DataFrame(data)
38
39
40     # shuffle dataset
41     data = data.sample(frac=1)
42
43     # get dependent and independent variable
44     data = np.array(data)
45
46
```

```

47 X = data[:,0:-2]
48 y = data[:,-2:]
49
50 # get unique values of y
51 y_unique = np.unique(y,axis=0)
52
53 # length of data for fit
54 train_len = int(np.shape(X)[0]*0.7)
55 val_len = int(np.shape(X)[0]*0.2)
56 test_len = int(np.shape(X)[0]) - val_len - train_len
57
58 # test train split
59 X_train = X[0:train_len]
60 X_test = X[train_len:train_len+test_len]
61 X_val = X[train_len+test_len:train_len+test_len+val_len]
62
63 y_train = y[0:train_len]
64 y_test = y[train_len:train_len+test_len]
65 y_val = y[train_len+test_len:train_len+test_len+val_len]
66
67 # K-means clustering
68 loop = 1
69 prev_zni = np.zeros((train_len,D-1))
70
71 while(loop>0):
72     zni = np.zeros((train_len,D-1))
73
74     if loop == 1:
75         # randomly choose k points
76         random_index = np.random.randint(0,train_len,D-1)
77
78         # Initialize MUi
79         MUi = X_train[random_index,:]
80
81         # Determine points belonging to clusters
82         for j in range(0,train_len):
83             i = np.argmin(np.linalg.norm(X_train[j,:]-MUi,axis=1),axis=0)
84             zni[j,i] = 1
85
86         # Determine number of datapoints in the clusters
87         Ni = np.sum(zni,axis=0)
88
89         # Update MUi
90         for j in range(0,D-1):
91             if Ni[j] == 0:
92                 continue
93
94             z = zni[:,j]
95             MUi[j,:] = (np.sum(X_train*z[:,np.newaxis],axis=0))/Ni[j]
96

```



```

97     # compare previous and current Zni
98     comp = prev_zni - zni
99
100    prev_zni = zni
101    loop = loop+1
102
103    # if no change break the loop
104    if np.max(comp) == 0 and np.min(comp) == 0:
105        break
106
107    # Assign Mui
108    mu = MUi
109
110    # gaussian function
111    phii = lambda x,mu : np.e**(-(np.linalg.norm(x-mu))/var**2)
112
113    # Formulate phi matrix for training data
114    phin = np.zeros((train_len,D))
115
116    for i in range(0,train_len):
117        for j in range(0,D):
118            if j==0:
119                phin[i,j] = 1
120                #phin[i,j] = phii(X_train[i,:],mu[j,:])
121            else:
122                phin[i,j] = phii(X_train[i,:],mu[j,:])
123
124    # weights with quadratic regularisation
125    W = np.linalg.inv(phin.T@phin + r*np.identity(D))@phin.T@y_train
126
127    # weights with tikhanov regularisation
128    # phin_bar = np.zeros((D,D))
129    # for i in range(0,D):
130    #     for j in range(0,D):
131    #         phin_bar[i,j] = np.exp(np.linalg.norm(mu[i,:]-mu[j,:])/var**2)
132
133    # W = np.linalg.inv(phin.T@phin + r*phin_bar)@phin.T@y_train
134
135    #####
136
137    # Test on training dataset
138    y_pred = np.zeros((train_len,2))
139
140    # Formulate phi matrix for training data
141    phin = np.zeros((train_len,D))
142
143    for i in range(0,train_len):
144        for j in range(0,D):
145            if j==0:
146                phin[i,j] = 1

```

```

147         # phin[i,j] = phii(X_train[i,:],mu[j,:])
148     else:
149         phin[i,j] = phii(X_train[i,:],mu[j])
150
151     for i in range(0,train_len):
152         pred = (phin[i,:])@W
153         y_pred[i,:] = pred
154
155         # term1 = np.deg2rad((pred[0] - y_unique[:,0])/2)
156         # term2 = np.deg2rad((pred[1] - y_unique[:,1])/2)
157         # a = (np.sin(term1)**2) + # np.cos(np.deg2rad(pred[0]))*
158         #      np.cos(np.deg2rad(y_unique[:,0]))*(np.sin(term2# )**2)
159         # dist = 2*6373*np.arctan2(np.sqrt(a),np.sqrt(1-a))
160
161         # index = np.argmin(dist)
162         # y_pred[i,:] = y_unique[index,:]
163
164     Erms_train1 = ((np.linalg.norm(y_pred[:,0]-y_train[:,0])))/np.sqrt(train_len)
165     Erms_train2 = ((np.linalg.norm(y_pred[:,1]-y_train[:,1])))/np.sqrt(train_len)
166
167     # # scatter plot
168     # fig1 = plt.figure(figsize=(9,9))
169     # plt.scatter(y_train[:,0],y_pred[:,0],marker='x')
170     # plt.scatter(y_train[:,1],y_pred[:,1],marker='x')
171     # plt.xlabel('target',fontsize=20)
172     # plt.ylabel('predicted',fontsize=20)
173     # plt.title("Gaussian Basis Function (For Dataset 3 of # best performing
174     #           model with training data) | degree = {} | # lambda = {}".format(D,
175     #           fontsize = 20)
176     # plt.legend(["y0", "y1"])
177     #####
178
179     # Test on validation dataset
180     y_pred = np.zeros((val_len,2))
181
182     # Formulate phi matrix for validation data
183     phin = np.zeros((val_len,D))
184
185     for i in range(0,val_len):
186         for j in range(0,D):
187             if j==0:
188                 phin[i,j] = 1
189                 # phin[i,j] = phii(X_val[i,:],mu[j,:])
190             else:
191                 phin[i,j] = phii(X_val[i,:],mu[j])
192
193     for i in range(0,val_len):
194         pred = (phin[i,:])@W
195         y_pred[i,:] = pred
196

```

```

197 Erms_val1 = ((np.linalg.norm(y_pred[:,0]-y_val[:,0])))/np.sqrt(val_len)
198 Erms_val2 = ((np.linalg.norm(y_pred[:,1]-y_val[:,1])))/np.sqrt(val_len)
199
200
201 # scatter plot for validation data
202 # fig1 = plt.figure(figsize=(9,9))
203 # plt.scatter(y_val,y_pred,marker='x')
204 # plt.xlabel('target',fontsize=20)
205 # plt.ylabel('predicted',fontsize=20)
206 # plt.title("Gaussian Basis Function (For Dataset 2 of best performing model
207 #           with validation data) | degree = {} | lambda # = {}".format(D,r),
208 #           fontsize = 20)
209
210 #####
211
212 # Test on test dataset
213 y_pred = np.zeros((test_len,2))
214
215 # Formulate phi matrix for test data
216 phin = np.zeros((test_len,D))
217
218 for i in range(0,test_len):
219     for j in range(0,D):
220         if j==0:
221             phin[i,j] = 1
222             #phin[i,j] = phii(X_test[i,:],mu[j,:])
223         else:
224             phin[i,j] = phii(X_test[i,:],mu[j])
225
226 for i in range(0,test_len):
227     pred = (phin[i,:])@W
228     y_pred[i,:] = pred
229
230 Erms_test1 = ((np.linalg.norm(y_pred[:,0]-y_test[:,0])))/np.sqrt(test_len)
231 Erms_test2 = ((np.linalg.norm(y_pred[:,1]-y_test[:,1])))/np.sqrt(test_len)
232
233 # scatter plot for test data
234 fig1 = plt.figure(figsize=(9,9))
235 plt.scatter(y_test[:,0],y_pred[:,0],marker='x')
236 plt.scatter(y_test[:,1],y_pred[:,1],marker='x')
237 plt.xlabel('target',fontsize=20)
238 plt.ylabel('predicted',fontsize=20)
239 plt.title("Gaussian Basis Function (For Dataset 3 of best performing model
240         with test data) | degree = {} | lambda = {}".format(D,r),
241         fontsize = 20)
242 plt.legend(["y0","y1"])
243 #####
244 return Erms_train1, Erms_train2, Erms_val1, Erms_val2, Erms_test1, Erms_test2
245
246 ##### Dataset 2 #####

```

```

247
248 def process_dataset_2(D, var, r):
249
250     """
251     D: No of gauss functions for fit
252     var: variance for fit
253     r: regularisation term
254     """
255
256     # Read csv file
257     data = pd.read_csv('datasets/function1_2d.csv')
258
259     # shuffle dataset
260     # data = data.sample(frac=1)
261
262     # get dependent and independent variable
263     X = data[['x1', 'x2']]
264     y = data['y']
265
266     # convert into numpy
267     X = np.array(X)
268     y = np.array(y)
269
270     # length of data for fit
271     train_len = int(np.shape(X)[0]*0.7)
272     test_len = int(np.shape(X)[0]*0.2)
273     val_len = int(np.shape(X)[0]*0.1)
274
275     # test train split
276     X_train = X[0:train_len]
277     X_test = X[train_len:train_len+test_len]
278     X_val = X[train_len+test_len:train_len+test_len+val_len]
279
280     y_train = y[0:train_len]
281     y_test = y[train_len:train_len+test_len]
282     y_val = y[train_len+test_len: train_len+test_len+val_len]
283
284     # K-means clustering
285     loop = 1
286     prev_zni = np.zeros((train_len,D-1))
287
288     while(loop>0):
289         zn_i = np.zeros((train_len,D-1))
290
291         if loop == 1:
292             # randomly choose k points
293             random_index = np.random.randint(0,train_len,D-1)
294
295             # Initialize MUi
296             MUi = X_train[random_index,:]

```

```

297
298     # Determine points belonging to clusters
299     for j in range(0,train_len):
300         i = np.argmin(np.linalg.norm(X_train[j,:]-MUj,axis=1),axis=0)
301         zni[j,i] = 1
302
303     # Determine number of datapoints in the clusters
304     Ni = np.sum(zni,axis=0)
305
306     # Update MUj
307     for j in range(0,D-1):
308         if Ni[j] == 0:
309             continue
310
311         z = zni[:,j]
312         MUj[j,:] = (np.sum(X_train*z[:,np.newaxis],axis=0))/Ni[j]
313
314     # compare previous and current Zni
315     comp = prev_zni - zni
316
317     prev_zni = zni
318     loop = loop+1
319
320     # if no change break the loop
321     if np.max(comp) == 0 and np.min(comp) == 0:
322         break
323
324     # Assign Mui
325     mu = MUj
326
327     # gaussian function
328     phii = lambda x,mu : np.e**(-(np.linalg.norm(x-mu))/var**2)
329
330     # Formulate phi matrix for training data
331     phin = np.zeros((train_len,D))
332
333     for i in range(0,train_len):
334         for j in range(0,D):
335             if j==0:
336                 phin[i,j] = 1
337             else:
338                 phin[i,j] = phii(X_train[i,:],mu[j-1,:])
339
340
341     # weights with quadratic regularisation
342     W = np.linalg.inv(phin.T@phin + r*np.identity(D))@phin.T@y_train
343
344     #####
345
346     # Test on training dataset

```

```

347 y_pred = np.zeros((train_len,1))
348
349 # Formulate phi matrix for training data
350 phin = np.zeros((train_len,D))
351
352 for i in range(0,train_len):
353     for j in range(0,D):
354         if j==0:
355             phin[i,j] = 1
356         else:
357             phin[i,j] = phii(X_train[i,:],mu[j-1])
358
359 for i in range(0,train_len):
360     y_pred[i,:] = (phin[i,:])@W
361
362 Erms_train = ((np.linalg.norm(y_pred-y_train[:,np.newaxis])))/
363               np.sqrt(train_len)
364
365
366 ## scatter plot for training data
367 # fig1 = plt.figure(figsize=(9,9))
368 # plt.scatter(y_train,y_pred,marker='x')
369 # plt.xlabel('target',fontsize=20)
370 # plt.ylabel('predicted',fontsize=20)
371 # plt.title("Gaussian Basis Function (For Dataset 2 of best performing model
372 #           with training data) | degree = {} | lambda = # {}".format(D,r),
373 #           fontsize = 20)
374
375
376 #####
377
378 # Test on validation dataset
379 y_pred = np.zeros((val_len,1))
380
381 # Formulate phi matrix for validation data
382 phin = np.zeros((val_len,D))
383
384 for i in range(0,val_len):
385     for j in range(0,D):
386         if j==0:
387             phin[i,j] = 1
388         else:
389             phin[i,j] = phii(X_val[i,:],mu[j-1])
390
391 for i in range(0,val_len):
392     y_pred[i,:] = (phin[i,:])@W
393
394 Erms_val = ((np.linalg.norm(y_pred-y_val[:,np.newaxis])))/np.sqrt(val_len)
395
396 # scatter plot for validation data

```

```

397     # fig1 = plt.figure(figsize=(9,9))
398     # plt.scatter(y_val,y_pred,marker='x')
399     # plt.xlabel('target',fontsize=20)
400     # plt.ylabel('predicted',fontsize=20)
401     # plt.title("Gaussian Basis Function (For Dataset 2 of best performing model
402     #           with validation data) | degree = {} | lambda # = {}".format(D,r),
403     #           fontsize = 20)
404
405     #####
406
407     # Test on test dataset
408     y_pred = np.zeros((test_len,1))
409
410     # Formulate phi matrix for test data
411     phin = np.zeros((test_len,D))
412
413     for i in range(0,test_len):
414         for j in range(0,D):
415             if j==0:
416                 phin[i,j] = 1
417             else:
418                 phin[i,j] = phii(X_test[i,:],mu[j-1])
419
420     for i in range(0,test_len):
421         y_pred[i,:] = (phin[i,:])@W
422
423     Erms_test = ((np.linalg.norm(y_pred-y_test[:,np.newaxis])))/np.sqrt(test_len)
424
425     # # scatter plot for validation data
426     # fig1 = plt.figure(figsize=(9,9))
427     # plt.scatter(y_test,y_pred,marker='x')
428     # plt.xlabel('target',fontsize=20)
429     # plt.ylabel('predicted',fontsize=20)
430     # plt.title("Gaussian Basis Function (For Dataset 2 of best performing
431     #           model with test data) | degree = {} | lambda # = {}".format(D,r),
432     #           fontsize = 20)
433
434     #####
435     return Erms_train, Erms_val, Erms_test
436
437
438     ##### Main Code #####
439
440     # Call function to experiment dataset 2/3
441     D = [2,3,6,9,20,33,40,100]
442     var = [1,3,5,10,50,100]
443     reg = [0,10e-5,10e-4,10e-3,10e-2,10e-1,10,10e2,10e3,10e4,10e5]
444
445     Erms = []
446     # Experiment with variance

```

```

447 for v in var:
448     erms = process_dataset_3(100, v, 0)
449     Erms.append(erms)
450
451 # Plot table
452 fig = plt.figure(figsize=(9,9))
453 row_labels=["var = 1", "var = 3", "var = 5", "var = 10", "var = 50", "var = 100"]
454 column_labels=["Erms_train", "Erms_val", "Erms_test"]
455 column_labels=["Erms_train_y0", "Erms_train_y1", "Erms_val0", "Erms_val1",
456                "Erms_test0", "Erms_test1"]
457 plt.axis('tight')
458 plt.axis('off')
459 plt.table(cellText=Erms, rowLabels=row_labels, colLabels=column_labels, loc="center")
460
461 Erms = []
462 # Experiment with Dimensions
463 for d in D:
464     erms = process_dataset_3(d, 3, 0)
465     Erms.append(erms)
466
467 # Plot table
468 fig = plt.figure(figsize=(9,9))
469 row_labels=["D = 2", "D = 3", "D = 6", "D = 9", "D = 20", "D = 30",
470            "D = 40", "D = 100"]
471 column_labels=["Erms_train", "Erms_val", "Erms_test"]
472 column_labels=["Erms_train_y0", "Erms_train_y1", "Erms_val0", "Erms_val1",
473                "Erms_test0", "Erms_test1"]
474 plt.axis('tight')
475 plt.axis('off')
476 plt.table(cellText=Erms, rowLabels=row_labels, colLabels=column_labels,
477            loc="center")
478
479 Erms = []
480 # Experiment with regularisation
481 for r in reg:
482     erms = process_dataset_3(100, 3, r)
483     Erms.append(erms)
484
485 # Plot table
486 fig = plt.figure(figsize=(9,9))
487 row_labels=["lambda = 0", "lambda = 10e-5", "lambda = 10e-4", "lambda = 10e-3",
488            "lambda = 10e-2", "lambda = 10e-1", "lambda = 10", "lambda = 10e2",
489            "lambda = 10e3", "lambda = 10e4", "lambda = 10e5"]
490 column_labels=["Erms_train", "Erms_val", "Erms_test"]
491 column_labels=["Erms_train_y0", "Erms_train_y1", "Erms_val0", "Erms_val1",
492                "Erms_test0", "Erms_test1"]
493 plt.axis('tight')
494 plt.axis('off')
495 plt.table(cellText=Erms, rowLabels=row_labels, colLabels=column_labels,
496            loc="center")

```



---

## IV Direct links For Python Codes

- [Polynomial Curve Fitting](#)
- [Linear Model for Regression using Polynomial Basis Functions](#)
- [Linear Model for Regression using Gaussian Basis Functions](#)