



<b>Stanford/</b> .....	1
└── .~lock.USArrests.csv# .....	1
└── ch12prob9.py .....	2
└── ch3prob9.py .....	5
└── ch3q14.py .....	8
└── dendogram.py .....	10
└── hw1graphs.py .....	12



# ~/.lock.USArrests.csv#

1 ,adam,ahlan,07.07.2023 02:20,file:///home/adam/.config/libreoffice/4;

## # ch12prob9.py

```
1 from numpy import *
2 import numpy as np
3 import matplotlib as mpl
4 import matplotlib.pyplot as plt
5 from scipy import stats
6 from scipy.interpolate import CubicSpline
7 from scipy.interpolate import splrep, BSpline
8
9
10 #np.genfromtxt("Auto.csv", delimiter=",")
11 #with open("Auto.csv", "r") as f:
12 #    print(f.read())
13
14 #data_a = loadtxt('Auto.csv',skiprows=1, usecols=(0,3), delimiter=',')
15 #print(data_a)
16
17 #CHAPTER 3 QUESTION 9
18
19 def data_loader(fname):
20     data_a = loadtxt(fname,skiprows=1, usecols=(0,1,2,3,4,5,6,7), delimiter=',')
21
22     #mpg          cylinders      displacement    horsepower      weight  acceleration   year   origin   name
23
24
25     #print(data_a.shape)
26     #print(data_a[0:,1].shape)
27     # We want it in the form of Y = XB
28     # Where Y is the response variable
29     # Where X is an array with size nx2 where n is the predictor variable and the other column is a 1
30     # B is the coefficients (slope and intercept) that we are trying to solve for     year_v = data_a[:,1]
31
32     n, m = data_a.shape
33     pred_a = data_a[:, 1:]
34
35     x_a = empty([n,pred_a.shape[1]+1], dtype=float64)
36     x_a[:, :-1] = pred_a
37     x_a[:, -1] = 1
38     name_l = ['mpg', 'cylinders', 'displacement', 'horsepower', 'weight', 'acceleration', 'year', 'origin']
39
40     #print(x_v.shape)
41
42     y_v = data_a[:,0]
43
44     return x_a, y_v, data_a, name_l
45
46 def scatter_matrix(data_a, name_l):
47     n, p = data_a.shape
48     fig, axs = plt.subplots(4, 7)
49     ax_l = list(axs.flat)
50     mpl.rcParams['figure.autolayout'] = True
51     font = {'family' : 'normal',
52            'weight' : 'bold',
53            'size' : 10}
54
55     #mpl.rc('font', **font)
56     for i in range(p):
57         for j in range(i+1,p):
58             print(i, j, name_l[i], name_l[j])
59             x_v = data_a[:,i]
60             y_v = data_a[:,j]
61             ax = ax_l.pop(0)
62             ax.scatter(x_v, y_v, s=2**2)
63             title = f'{name_l[i]} vs {name_l[j]}'
64             ax.set_title(title[:25])
65
66     plt.tight_layout()
67     fig.savefig('ch12_9.pdf')
68     plt.show()
69
70 def diag_plot(y_v, yfit_v):
71     res_v = y_v - yfit_v
72     i_v = yfit_v.argsort()
73     yfit1_v = yfit_v[i_v]
74     res1_v = res_v[i_v]
75     fig, ax = plt.subplots(figsize=(15, 15))
76     ax.scatter(yfit1_v, res1_v)
77     #spline = CubicSpline(yfit1_v, res1_v)
78     #res2_v = spline(yfit1_v)
79     tck = splrep(yfit1_v, res1_v, s=3500)
80     res2_v = BSpline(*tck)(yfit1_v)
81     #ax.plot(yfit1_v, res1_v)
82     ax.plot(yfit1_v, res2_v)
```

```

83     plt.show()
84
85     def transform1_reg(x_a):
86         p_a = x_a[:,2:3]
87         p_a = (p_a)**2
88
89         return hstack((p_a,x_a))
90
91     def transform2_reg(x_a):
92         p_a = x_a[:,2:3]
93         p_a = (p_a)**0.5
94
95         return hstack((p_a,x_a))
96
97     def transform3_reg(x_a):
98         p_a = x_a[:,2:3]
99         p_a = log((p_a))
100
101         return hstack((p_a,x_a))
102
103     def transform4_reg(x_a):
104         p_a = x_a[:,2:4]
105         p_a = product(p_a,axis=1,keepdims=True)
106
107         return hstack((p_a,x_a))
108
109     def transform5_reg(x_a):
110         p_a = x_a[:,2:5]
111         p_a = product(p_a,axis=1,keepdims=True)
112
113         return hstack((p_a,x_a))
114
115
116     def interaction_study(x_a, y_v, name):
117         b_v = lin_regression(x_a,y_v)
118         yfit_v = fitted_func(x_a, b_v)
119         print(f'{name}: Coefficients {b_v=}')
120         print(f'{name}: r squared = ', r_square(y_v,yfit_v))
121
122
123
124
125     def lin_regression(x_a,y):
126         #y_v = X@B
127         b_v = linalg.pinv(x_a)@y
128         #print(b)
129         return b_v
130
131     def fitted_func(x_a,b_v):
132         yfit_v = x_a@b_v
133         # with np.printoptions(precision=2):
134         #     print(f'predicted mpg of cars {yfit_v=}')
135         return yfit_v
136
137     def r_square(y_v,yfit_v):
138         # This function is to find the r squared value
139         # This will be calculated by doing 1 - variance of (actual - predicted)/variance of actual
140         rsq = 1 - (var(y_v-yfit_v))/(var(y_v))
141
142         return rsq
143
144     def main ():
145         x_a, y_v, data_a, name_l = data_loader('Auto.csv')
146         cor_a = corrcoef(data_a, rowvar=False)
147         print(cor_a.shape)
148         with np.printoptions(precision=4):
149             print(cor_a)
150
151         #print(x_a,y_v)
152         #scatter_matrix(data_a, name_l)
153
154         #print(x_a,y_v)
155         b_v = lin_regression(x_a,y_v)
156         yfit_v = fitted_func(x_a, b_v)
157         i_v = abs(b_v).argsort()[::-1]
158         print(f'Coefficients {b_v=}')
159         print('Coefficients:', [(name, b) for name, b in zip(name_l, b_v)])
160         print("pred_influence",[name_l[i] for i in i_v])
161         print('r squared = ', r_square(y_v,yfit_v))
162
163         #diag_plot(y_v, yfit_v)
164         interaction_study(x_a, y_v, 'control')
165         interaction_study(transform1_reg(x_a),y_v, 'Horse Power Squared')
166         interaction_study(transform2_reg(x_a),y_v, 'Horse Power Square Root')
167         interaction_study(transform3_reg(x_a),y_v, 'Horse Power Log')
168         interaction_study(transform4_reg(x_a),y_v, 'Horse Power * Weight')
169         interaction_study(transform5_reg(x_a),y_v, 'Horse Power * Weight * Acceleration')

```

```
170
171     raise SystemExit
172
173     #print(yhat_v)
174 if __name__ == '__main__':
175     main()
176
177
178 # zeros, full, empty, array, arange, indexing, transpose
179
180 #URLs used: https://docs.scipy.org/doc/scipy/tutorial/interpolate/smoothing\_splines.html
181 #
```

## # ch3prob9.py

```
1 from numpy import *
2 import numpy as np
3 import matplotlib as mpl
4 import matplotlib.pyplot as plt
5 from scipy import stats
6 from scipy.interpolate import CubicSpline
7 from scipy.interpolate import splrep, BSpline
8 import statsmodels.api as sm
9
10
11 #Adam Kainikara
12 #This code is for
13 #CHAPTER 3 QUESTION 9
14
15 def data_loader(fname):
16     data_a = loadtxt(fname, skiprows=1, usecols=(0,1,2,3,4,5,6,7), delimiter=',')
17
18     #mpg      cylinders      displacement      horsepower      weight      acceleration      year      origin      name
19
20
21     #print(data_a.shape)
22     #print(data_a[0:,1].shape)
23     # We want it in the form of  $Y = XB$ 
24     # Where Y is the response variable
25     # Where X is an array with size nx2 where n is the predictor variable and the other column is a 1
26     # B is the coefficients (slope and intercept) that we are trying to solve for     year_v = data_a[:,1]
27
28     n, m = data_a.shape
29     pred_a = data_a[:, 1:]
30
31     x_a = empty([n, pred_a.shape[1]+1], dtype=float64)
32     x_a[:, :-1] = pred_a
33     x_a[:, -1] = 1
34     name_l = ['mpg', 'cylinders', 'displacement', 'horsepower', 'weight', 'acceleration', 'year', 'origin']
35
36     #print(x_v.shape)
37
38     y_v = data_a[:, 0]
39
40     return x_a, y_v, data_a, name_l
41
42 def scatter_matrix(data_a, name_l):
43     n, p = data_a.shape
44     fig, axs = plt.subplots(4, 7)
45     ax_l = list(axs.flat)
46     mpl.rcParams['figure.autolayout'] = True
47     font = {'family' : 'normal',
48             'weight' : 'bold',
49             'size' : 10}
50
51     #mpl.rc('font', **font)
52     for i in range(p):
53         for j in range(i+1, p):
54             print(i, j, name_l[i], name_l[j])
55             x_v = data_a[:, i]
56             y_v = data_a[:, j]
57             ax = ax_l.pop(0)
58             ax.scatter(x_v, y_v, s=2**2)
59             title = f'{name_l[i]} vs {name_l[j]}'
60             ax.set_title(title[:25])
61
62     plt.tight_layout()
63     fig.savefig('ch12_9.pdf')
64     plt.show()
65
66 def diag_plot(y_v, yfit_v):
67     res_v = y_v - yfit_v
68     i_v = yfit_v.argsort()
69     yfit1_v = yfit_v[i_v]
70     res1_v = res_v[i_v]
71     fig, ax = plt.subplots(figsize=(15, 15))
72     ax.scatter(yfit1_v, res1_v)
73     #spline = CubicSpline(yfit1_v, res1_v)
74     #res2_v = spline(yfit1_v)
75     tck = splrep(yfit1_v, res1_v, s=3500)
76     res2_v = BSpline(*tck)(yfit1_v)
77     #ax.plot(yfit1_v, res1_v)
78     ax.plot(yfit1_v, res2_v)
79
80     plt.show()
81
82 def transform1_reg(x_a):
83     p_a = x_a[:, 2:3]
```

```

83     p_a = (p_a)**2
84
85     return hstack((p_a,x_a))
86
87 def transform2_reg(x_a):
88     p_a = x_a[:,2:3]
89     p_a = (p_a)**0.5
90
91     return hstack((p_a,x_a))
92
93 def transform3_reg(x_a):
94     p_a = x_a[:,2:3]
95     p_a = log((p_a))
96
97     return hstack((p_a,x_a))
98
99 def transform4_reg(x_a):
100    p_a = x_a[:,2:4]
101    p_a = product(p_a,axis=1,keepdims=True)
102
103    return hstack((p_a,x_a))
104
105 def transform5_reg(x_a):
106    p_a = x_a[:,2:5]
107    p_a = product(p_a,axis=1,keepdims=True)
108
109    return hstack((p_a,x_a))
110
111
112 def interaction_study(x_a, y_v, name):
113     b_v = lin_regression(x_a,y_v,name=name)
114     yfit_v = fitted_func(x_a, b_v)
115     print(f'{name}: Coefficients {b_v=}')
116     print(f'{name}: r squared = ', r_square(y_v,yfit_v))
117
118
119
120
121 def lin_regression(x_a,y_v,name=''):
122     if name:
123         print(f'\n\n----- {name} -----')
124         model = sm.OLS(y_v, x_a)
125         results = model.fit()
126         print(results.summary())
127         #Using stats models to get p value even though I did my own regression
128         # y_v = X@B
129         b_v = linalg.pinv(x_a)@y_v
130         #print(b)
131         return b_v
132
133 def fitted_func(x_a,b_v):
134     yfit_v = x_a@b_v
135     # with np.printoptions(precision=2):
136     #     print(f'predicted mpg of cars {yfit_v=}')
137     return yfit_v
138
139 def r_square(y_v,yfit_v):
140     # This function is to find the r squared value
141     # This will be calculated by doing 1 - variance of (actual - predicated)/variance of actual
142     rsq = 1 - (var(y_v-yfit_v))/(var(y_v))
143
144     return rsq
145
146 def main ():
147     x_a, y_v, data_a, name_l = data_loader('Auto.csv')
148     cor_a = corrcoeff(data_a, rowvar=False)
149     print(cor_a.shape)
150     with np.printoptions(precision=4):
151         print(cor_a)
152
153     #print(x_a,y_v)
154     scatter_matrix(data_a, name_l)
155
156     #print(x_a,y_v)
157     b_v = lin_regression(x_a,y_v, name='Main Regression')
158     yfit_v = fitted_func(x_a, b_v)
159     i_v = abs(b_v).argsort()[::-1]
160     print(f'Coefficients {b_v=}')
161     print('Coefficients:', [(name, b) for name, b in zip(name_l, b_v)])
162     print("pred influence",[name_l[i] for i in i_v])
163     print('r squared = ', r_square(y_v,yfit_v))
164     diag_plot(y_v, yfit_v)
165
166     with np.printoptions(precision=2):
167         interaction_study(x_a, y_v, 'control')
168         interaction_study(transform1_reg(x_a),y_v, 'Horse Power Squared')
169         interaction_study(transform2_reg(x_a),y_v, 'Horse Power Square Root')

```



```

170     interaction_study(transform3_reg(x_a),y_v, 'Horse Power Log')
171     interaction_study(transform4_reg(x_a),y_v, 'Horse Power * Weight')
172     interaction_study(transform4_reg(x_a),y_v, 'Horse Power * Weight * Acceleration')
173
174     raise SystemExit
175
176     #print(yhat_v)
177 if __name__ == '__main__':
178     main()
179
180
181 # zeros, full, empty, array, arange, indexing, transpose
182
183 #URLs used: https://docs.scipy.org/doc/scipy/tutorial/interpolate/smoothing\_splines.html
184 #

```

## # ch3q14.py

```
1 from numpy import *
2 import numpy as np
3 import matplotlib as mpl
4 import matplotlib.pyplot as plt
5 from scipy import stats
6 from scipy.interpolate import CubicSpline
7 from scipy.interpolate import splrep, BSpline
8 import statsmodels.api as sm
9
10
11 #Adam Kainikara
12 #This code is for
13 #CHAPTER 3 QUESTION 14
14
15 def data_loader():
16     random.seed(1)
17     x1_v = random.uniform(size=100)
18     x2_v = 0.5*x1_v + random.normal(size=100)/10
19     y_v = 2+2*x1_v+0.3*x2_v+random.normal(size=100)
20     x_a = empty((100,3), dtype=float64)
21     x_a[:,0] = 1
22     x_a[:,1] = x1_v
23     x_a[:,2] = x2_v
24     return x_a, y_v
25
26 def scatter_plot(x1_v,x2_v):
27     plt.scatter(x1_v,x2_v)
28     plt.show()
29
30 def rand_var_study(x_a, y_v, name):
31     b_v = lin_regression(x_a,y_v,name=name)
32     yfit_v = fitted_func(x_a, b_v)
33     print(f'{name}: Coefficients {b_v=}')
34     print(f'{name}: r squared = ', r_square(y_v,yfit_v))
35
36 def transform1_x1_only(x_a):
37     #X1 and constant only
38     return hstack((x_a[:,0:1],x_a[:,1:2]))
39
40
41 def transform1_x2_only(x_a):
42     #X2 and constant only
43     return hstack((x_a[:,0:1],x_a[:,2:]))
44
45 def transform3_reg(x_a):
46     p_a = x_a[:,2:3]
47     p_a = log((p_a))
48
49     return hstack((p_a,x_a))
50
51 def transform4_reg(x_a):
52     p_a = x_a[:,2:4]
53     p_a = product(p_a,axis=1,keepdims=True)
54
55     return hstack((p_a,x_a))
56
57 def transform5_reg(x_a):
58     p_a = x_a[:,2:5]
59     p_a = product(p_a,axis=1,keepdims=True)
60
61     return hstack((p_a,x_a))
62
63
64 def interaction_study(x_a, y_v, name):
65     b_v = lin_regression(x_a,y_v, name=name)
66     yfit_v = fitted_func(x_a, b_v)
67     print(f'{name}: Coefficients {b_v=}')
68     print(f'{name}: r squared = ', r_square(y_v,yfit_v))
69
70
71
72
73 def lin_regression(x_a,y_v,name=''):
74     if name:
75         print(f'\n\n----- {name} -----')
76         model = sm.OLS(y_v, x_a)
77         results = model.fit()
78         print(results.summary())
79     #Using stats models to get p value even though I did my own regression
```

```

80     # y_v = X@B
81     b_v = linalg.pinv(x_a)@y_v
82     #print(b)
83     return b_v
84
85 def fitted_func(x_a,b_v):
86     yfit_v = x_a@b_v
87     # with np.printoptions(precision=2):
88     #     print(f'predicted mpg of cars {yfit_v=}')
89     return yfit_v
90
91 def r_square(y_v,yfit_v):
92     # This function is to find the r squared value
93     # This will be calculated by doing 1 - variance of (actual - predicted)/variance of actual
94     rsq = 1 - (var(y_v-yfit_v))/(var(y_v))
95
96     return rsq
97
98 def main ():
99     x_a, y_v = data_loader()
100     b_v = lin_regression(x_a,y_v,name='Main Regression')
101     print('beta values', b_v)
102     scatter_plot(x_a[:,1],x_a[:,2])
103     cor_a = corcoef(x_a[:,1:], rowvar=False)
104     print(cor_a.shape)
105     with np.printoptions(precision=4):
106         print(cor_a)
107     rand_var_study(x_a,y_v,'Control')
108     rand_var_study(transform1_x1_only(x_a),y_v,'X1 Only')
109     rand_var_study(transform1_x2_only(x_a),y_v,'X2 Only')
110
111     x1_a = vstack((x_a,array([[1,0.1,0.8]])))
112     y1_v = hstack((y_v,array([6])))
113     rand_var_study(x_a,y_v,'New Control')
114     rand_var_study(transform1_x1_only(x1_a),y1_v,'New X1 Only')
115     rand_var_study(transform1_x2_only(x1_a),y1_v,'New X2 Only')
116
117     raise SystemExit
118
119     #print(x_a,y_v)
120     #scatter_matrix(data_a, name_l)
121
122     #print(x_a,y_v)
123     yfit_v = fitted_func(x_a, b_v)
124     i_v = abs(b_v).argsort()[::-1]
125     print(f'Coefficients {b_v=}')
126     print('Coefficients:', [(name, b) for name, b in zip(name_l, b_v)])
127     print("pred influence", [name_l[i] for i in i_v])
128     print('r squared = ', r_square(y_v,yfit_v))
129
130     #diag_plot(y_v, yfit_v)
131     interaction_study(x_a, y_v, 'control')
132     interaction_study(transform1_reg(x_a),y_v, 'Horse Power Squared')
133     interaction_study(transform2_reg(x_a),y_v, 'Horse Power Square Root')
134     interaction_study(transform3_reg(x_a),y_v, 'Horse Power Log')
135     interaction_study(transform4_reg(x_a),y_v, 'Horse Power * Weight')
136     interaction_study(transform4_reg(x_a),y_v, 'Horse Power * Weight * Acceleration')
137
138
139     #print(yhat_v)
140 if __name__ == '__main__':
141     main()
142
143
144 # zeros, full, empty, array, arange, indexing, transpose
145
146 #URLs used: https://docs.scipy.org/doc/scipy/tutorial/interpolate/smoothing_splines.html
147 #

```

## # dendogram.py

```
1 from numpy import *
2 import numpy as np
3 import matplotlib as mpl
4 import matplotlib.pyplot as plt
5 from scipy.interpolate import splrep, BSpline
6 from scipy.cluster.hierarchy import complete, fcluster
7 from scipy.cluster import hierarchy
8 from scipy.spatial.distance import pdist
9
10 #Chapter 12 question 9
11
12
13
14
15
16 def data_loader(fname):
17     x_a = loadtxt(fname, skiprows=1, usecols=(1,2,3,4), delimiter=',')
18
19
20     state_v = loadtxt(fname, skiprows=1, usecols=(0), delimiter=',', dtype=str)
21
22
23
24     print(state_v)
25     #print(data_a[0:,1].shape)
26     # We want it in the form of Y = XB
27     # Where Y is the response variable
28     # Where X is an array with size nx2 where n is the predictor variable and the other column is a 1
29     # B is the coefficients (slope and intercept) that we are trying to solve for     year_v = data_a[:,1]
30
31
32
33     return x_a, state_v
34
35 def dendo_construct(x_a, state_v):
36     n = state_v.shape[0]
37
38     def label_func(id):
39         if id < n:
40             return state_v[id]
41         return f'{id}'
42
43     d_a = pdist(x_a)
44     print(d_a.shape)
45     print(d_a)
46     z_a = complete(d_a)
47     print(z_a.shape)
48     print(z_a)
49     plt.figure()
50     dn = hierarchy.dendrogram(z_a, leaf_label_func=label_func)
51     plt.show()
52
53 def z_score(x_a):
54     m_v = x_a.mean(axis=0)
55     std_v = x_a.std(axis=0)
56     z_a = (x_a - m_v)/std_v
57     return z_a
58
59 def scatter_matrix(data_a, name_l):
60     n, p = data_a.shape
61     fig, axs = plt.subplots(4, 7)
62     ax_l = list(axs.flat)
63     mpl.rcParams['figure.autolayout'] = True
64     font = {'family' : 'normal',
65            'weight' : 'bold',
66            'size' : 10}
67
68     #mpl.rc('font', **font)
69     for i in range(p):
70         for j in range(i+1,p):
71             print(i, j, name_l[i], name_l[j])
72             x_v = data_a[:,i]
73             y_v = data_a[:,j]
74             ax = ax_l.pop(0)
75             ax.scatter(x_v, y_v, s=2**2)
76             title = f'{name_l[i]} vs {name_l[j]}'
77             ax.set_title(title[:25])
78     plt.tight_layout()
79     fig.savefig('ch12_9.pdf')
80     plt.show()
81
82 def main():
83     x_a, state_v = data_loader('USArrests.csv')
84     dendo_construct(x_a, state_v)
85     dendo_construct(z_score(x_a), state_v)
86     raise SystemExit
```

```
87     raise SystemExit
88
89     #print(yhat_v)
90 if __name__ == '__main__':
91     main()
92
93
94 # zeros, full, empty, array, arange, indexing, transpose
95
96 #URLs used: https://docs.scipy.org/doc/scipy/tutorial/interpolate/smoothing\_splines.html
97 #https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.dendrogram.html#scipy.cluster.hierarchy.dendrogram
```

## # hw1graphs.py

```
1 from numpy import *
2 import matplotlib.pyplot as plt
3 from matplotlib import patheffects
4 from statistics import linear_regression
5 from scipy.interpolate import CubicSpline
6
7
8 #THIS CODE IS FOR THE 5 THINGS GRAPH
9
10
11 fig, ax = plt.subplots(figsize=(15, 15))
12
13 #x x^2 constant
14 def powxy(x, y):
15     return [x**2, x, 1, y]
16 def powxy2(x, y):
17     return [x**-3, x**-2, x**-1, 1, y]
18 def startdata():
19     pass
20     data_a = array([
21         powxy(0,4),
22         #powxy(11,90),
23         powxy(15,170),
24         powxy(20,400)
25     ])
26     # yvalue_start = array([
27     #     powxy[4]
28     # ])
29     X_a = data_a[:, :-1]
30     Y_a = data_a[:, -1]
31     print(X_a.shape)
32     print(Y_a.shape)
33     Xinv_a = linalg.pinv(X_a)
34     coefs = Xinv_a @ Y_a
35     print(coefs)
36
37     x_v = linspace(0.5, 20, 400)
38     y_v = coefs[0] * (x_v ** 2) + coefs[1] * (x_v ** 1) + coefs[2]
39
40     #y_v = coefs[0] * (x_v ** 3 + coefs[1] * (x_v ** 2) + coefs[2] * (x_v ) + coefs[3]
41     print(y_v)
42     fig[0, 0].plot(x_v, y_v, label='var')
43     ax.legend()
44
45     plt.show()
46
47
48
49 def startdata_cubic():
50
51     data_a = array([
52         (0,4),
53         (5,20),
54         (15,140),
55         (20,400)
56     ])
57     # yvalue_start = array([
58     #     powxy[4]
59     # ])
60     X_v = data_a[:, 0]
61     Y_v = data_a[:, 1]
62     spline = CubicSpline(X_v, Y_v)
63
64
65     x1_v = linspace(0.5, 20, 400)
66     y1_v = spline(x1_v)
67
68     #y_v = coefs[0] * (x_v ** 3 + coefs[1] * (x_v ** 2) + coefs[2] * (x_v ) + coefs[3]
69     print(y1_v)
70     ax.plot(x1_v, y1_v, label='variance')
71     ax.legend()
72
73
74 def bias_cubic():
75
76     data_a = array([
77         (0,400),
78         (5,140),
79         (15,20),
```

```

80         (20,4)
81     ])
82     # yvalue_start = array([
83     #     powxy[4]
84     # ])
85     X_v = data_a[:,0]
86     Y_v = data_a[:,1]
87     spline = CubicSpline(X_v, Y_v)
88
89
90     x1_v = linspace(0.5,20,400)
91     y1_v = spline(x1_v)
92
93     #y_v = coefs[0] * (x_v **3 + coefs[1] * (x_v ** 2) + coefs[2] * (x_v ) + coefs[3]
94     print(y1_v)
95     ax.plot(x1_v, y1_v, label='squared bias')
96
97     ax.legend()
98
99 def trainererror_cubic():
100
101     data_a = array([
102         (0,340),
103         (5,210),
104         (10,110),
105         (15,60),
106         (20,15)
107     ])
108     # yvalue_start = array([
109     #     powxy[4]
110     # ])
111     X_v = data_a[:,0]
112     Y_v = data_a[:,1]
113     spline = CubicSpline(X_v, Y_v)
114
115
116     x1_v = linspace(0.5,20,400)
117     y1_v = spline(x1_v)
118
119     #y_v = coefs[0] * (x_v **3 + coefs[1] * (x_v ** 2) + coefs[2] * (x_v ) + coefs[3]
120     print(y1_v)
121     ax.plot(x1_v, y1_v, label='training error')
122
123     ax.legend()
124
125 def testerror_cubic():
126
127     data_a = array([
128         (0,350),
129         (5,250),
130         (10,140),
131         (15,210),
132         (20,340)
133     ])
134     # yvalue_start = array([
135     #     powxy[4]
136     # ])
137     X_v = data_a[:,0]
138     Y_v = data_a[:,1]
139     spline = CubicSpline(X_v, Y_v)
140
141
142     x1_v = linspace(0.5,20,400)
143     y1_v = spline(x1_v)
144
145     #y_v = coefs[0] * (x_v **3 + coefs[1] * (x_v ** 2) + coefs[2] * (x_v ) + coefs[3]
146     print(y1_v)
147     ax.plot(x1_v, y1_v, label='test error')
148
149     ax.legend()
150
151 def irrerror():
152     ax.axhline(y=200, xmin = 0.05, xmax = 0.95, label='irreducible error')
153     ax.legend()
154 def bias():
155     data_a = array([
156         powxy2(2,25),
157         powxy2(3,16),
158         powxy2(4,9),
159         powxy2(5,4)
160     ])
161     # yvalue_start = array([
162     #     powxy[4]

```

```

163     # ])
164     X_a = data_a[:, :-1]
165     Y_a = data_a[:, -1]
166     print(X_a.shape)
167     print(Y_a.shape)
168     Xinv_a = linalg.pinv(X_a)
169     coefs = Xinv_a @ Y_a
170     print(coefs)
171
172     x_v = linspace(1, 20, 400)
173     y_v = coefs[0] * (x_v ** 3) + coefs[1] * (x_v ** 2) + coefs[2] * (x_v ) + coefs[3]
174     print(y_v)
175     ax.plot(x_v, y_v, label='bias')
176     ax.legend()
177
178     plt.show()
179     def main():
180         plt.xlabel("flexibility")
181         plt.ylabel("value")
182         startdata_cubic()
183         bias_cubic()
184         trainerror_cubic()
185         testerror_cubic()
186         irrerror()
187         plt.show()
188         #startdata()
189         #bias()
190     if __name__ == '__main__':
191         main()
192
193     #y_v = (coefs[0] * x_v) + ((coefs[1]) * 2) + coefs[3]
194
195     #https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.axhline.html

```