sein_assgnment12

December 20, 2018

1 Assignment12

2 20142740

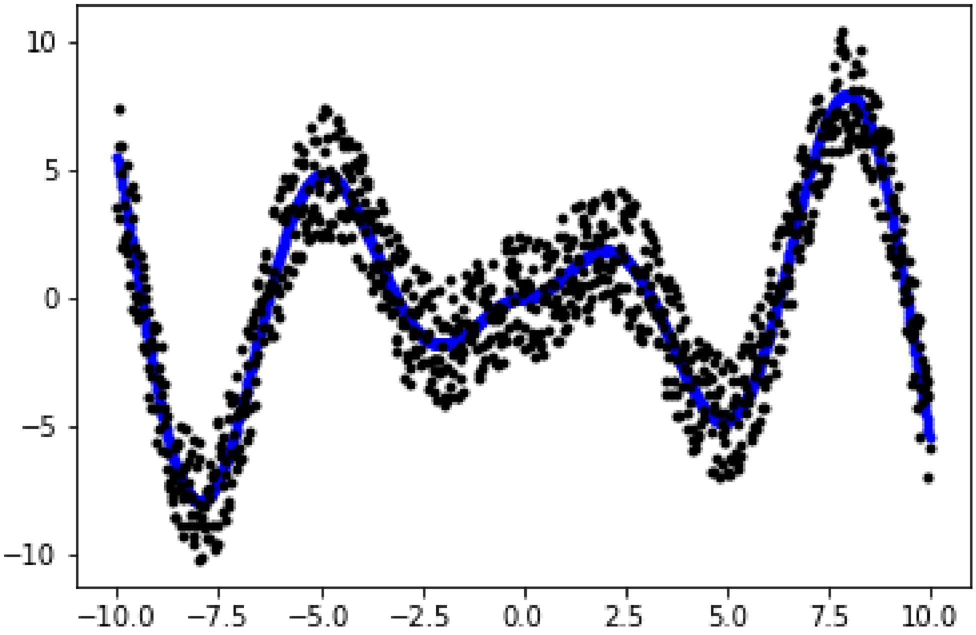
3 https://github.com/dkdvkd/assignment12

4 import packages

```
In [87]: import numpy as np
         import matplotlib.pyplot as plt
```

5 given data

```
In [88]: num     = 1001
         std     = 5

         # x  : x-coordinate data
         # y1 : (clean) y-coordinate data
         # y2 : (noisy) y-coordinate data

         def fun(x):
             # f = np.sin(x) * (1 / (1 + np.exp(-x)))
             f = np.abs(x) * np.sin(x)
             return f

         n       = np.random.rand(num)
         nn      = n - np.mean(n)
         x       = np.linspace(-10,10,num)
         y1      = fun(x)             # clean points
         y2      = y1 + nn * std      # noisy points

         plt.plot(x, y1, 'b.', x, y2, 'k.')
         plt.show()
```



6 least square, error function

```
In [89]: def Approximation(vecX,vecY):
             vecX = np.mat(vecX)
             vecY = np.mat(vecY).T
             xTx = vecX.T * vecX
             if(np.linalg.det(xTx) == 0.0): # if Singular function, return
                 print("singular matrix")
                 return
             weight = xTx.I * (vecX.T * vecY) # weight
             return weight

         def computeError(vec1, vec2):
             error = 0
             for i in range(0, len(vec2)):
                 error += np.sqrt((float(vec1[i]) - float(vec2[i]))**2)
             print("MSE: ", error)
             return np.sqrt(error)
```

7 lambda

```
In [62]: def getLambda(matrix, vec, lambda_):
             column = len(matrix.T)
             lamvec = np.ones((1,column), dtype=float)
             zvec = [0]
             matrixA = np.concatenate((matrix, lambda_*lamvec), axis=0)
             matrixY = np.concatenate((vec, zvec), axis=0)
             return matrixA, matrixY
```

8 Make p, A

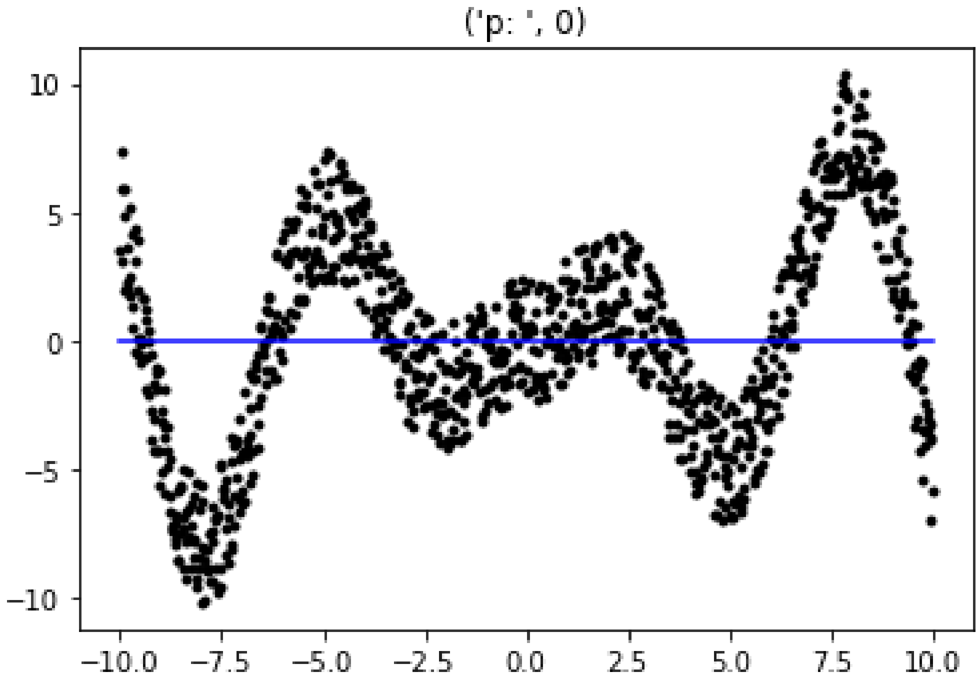
```
In [63]: def makeSigma(num):
             mu, sigma = 0, 1 # mean and standard deviation
             r = np.random.normal(mu, sigma, num)
             return r

         def func(n , lambda_, p):
             dimensions = [];
             dimenX = [];
             var = [];
             weight = [];
             y = [];
             y_ = [];

             for i in range(0, len(x)):
                 dimenX = []
                 for j in range(0, n+1):
                     dimenX.append(x[i]**j)
                 dimensions.append(dimenX)
             dimensions = np.mat(dimensions)
             matrixA, matrixY = getLambda(dimensions, y2, lambda_)
             weight = Approximation(matrixA, matrixY)
             print('weight:\n', weight)

             for j in range(0, n+1):
                 var = []
                 var = (x**j)*(float(weight[j]))
                 y_.append(var)
             y_ = np.mat(y_)
             y_ = y_.T

             for i in range(0, len(y_)):
                 sum_ = np.sum(y_[i])
                 y.append(float(sum_))

             return y
```

9 Training and show polynomial curves

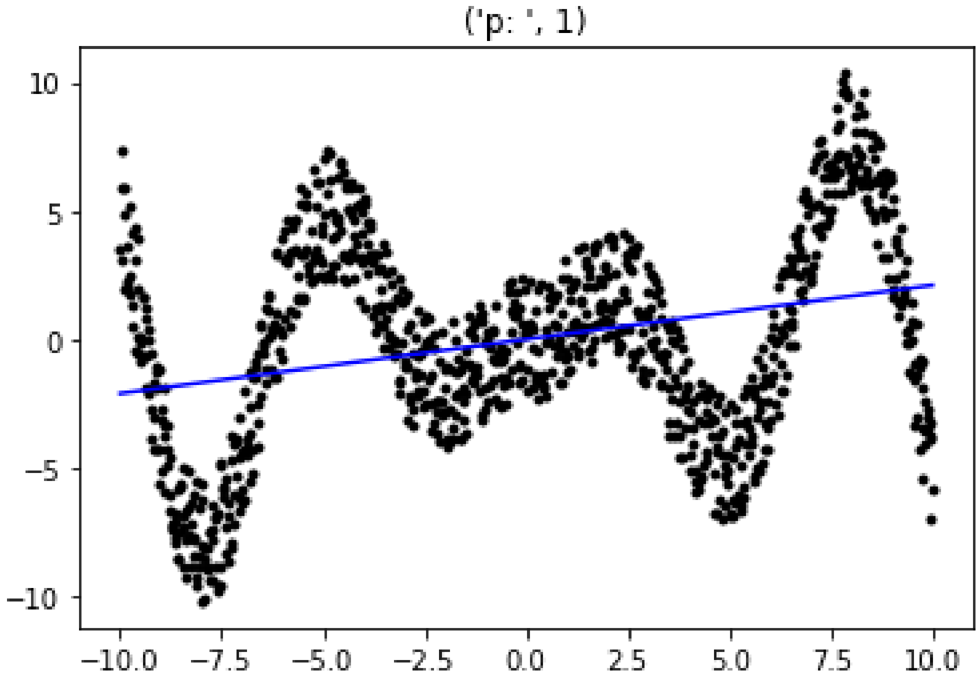
10 (lambda: 3)

```
In [98]: alpha=0.1

         def training(n, lambda_, p):
             error = []; y = []
             y = func(n, lambda_, p)
             error = computeError(y2, y)
             Ylabel = n,' dimension'

             plt.plot(x, y2, 'k.')
             plt.plot(x, y, 'b')
             title = ("p: ", n)
             plt.title(title)
             plt.show()

             return error


         # dimension is N
         r = makeSigma(15)
         error = []
         for n in range(0, 15):
             lambda_ = 3
             err = training(n, lambda_, r)
             error.append(err)
```

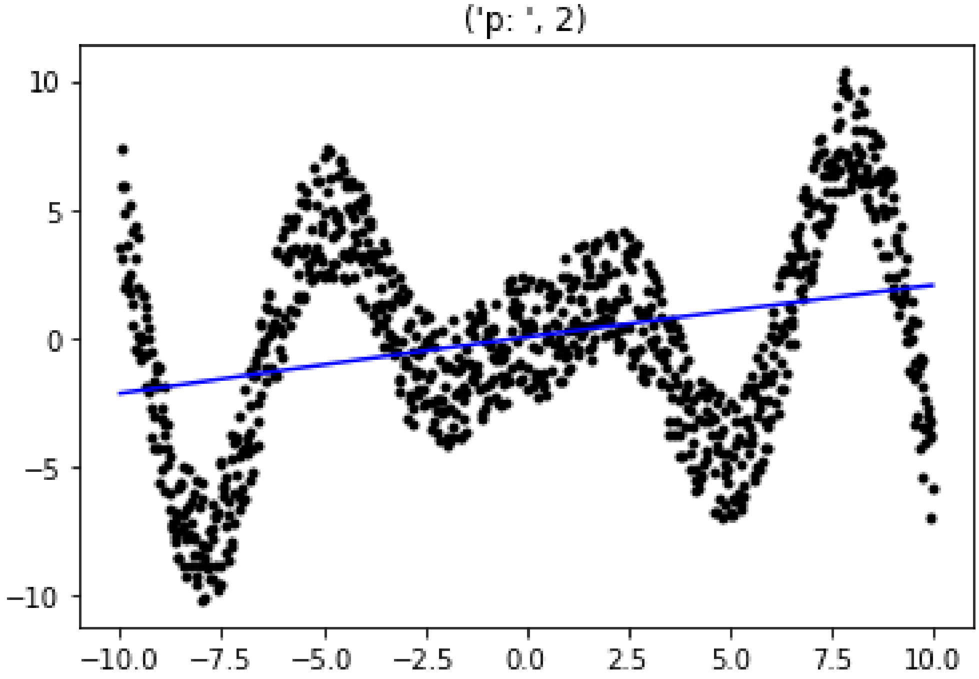
```
weight:
 [[-2.70850449e-16]]
MSE:  3205.76522356561
```



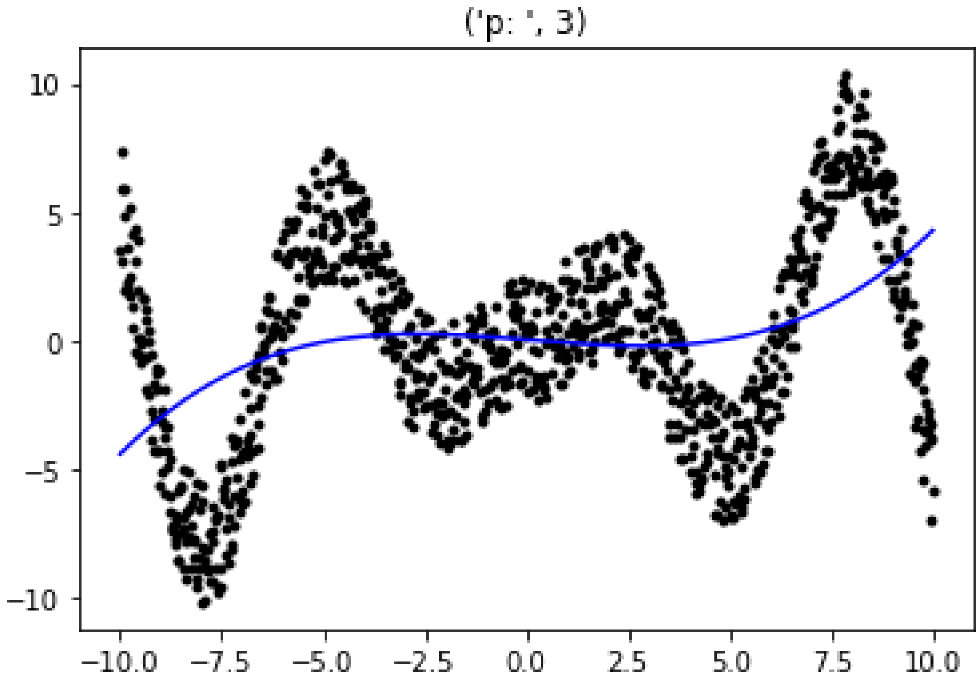
```
weight:
 [[-0.00187598]
 [ 0.21052704]]
MSE:  3133.987296050061
```



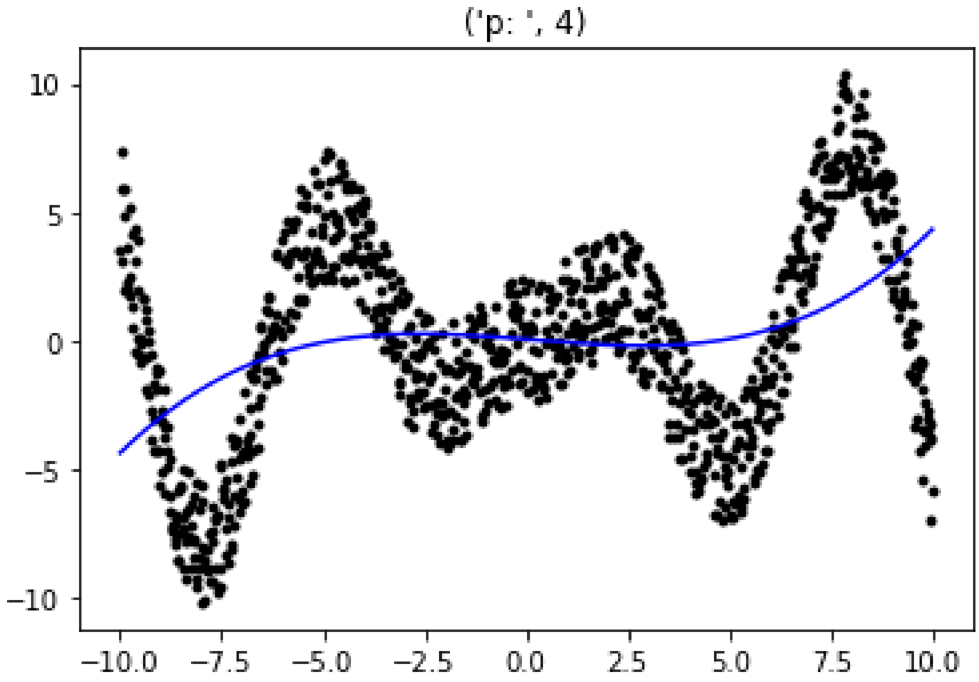
```
weight:
 [[ 0.02568001]
 [ 0.21051984]
 [-0.00083222]]
MSE:  3134.493874115038
```



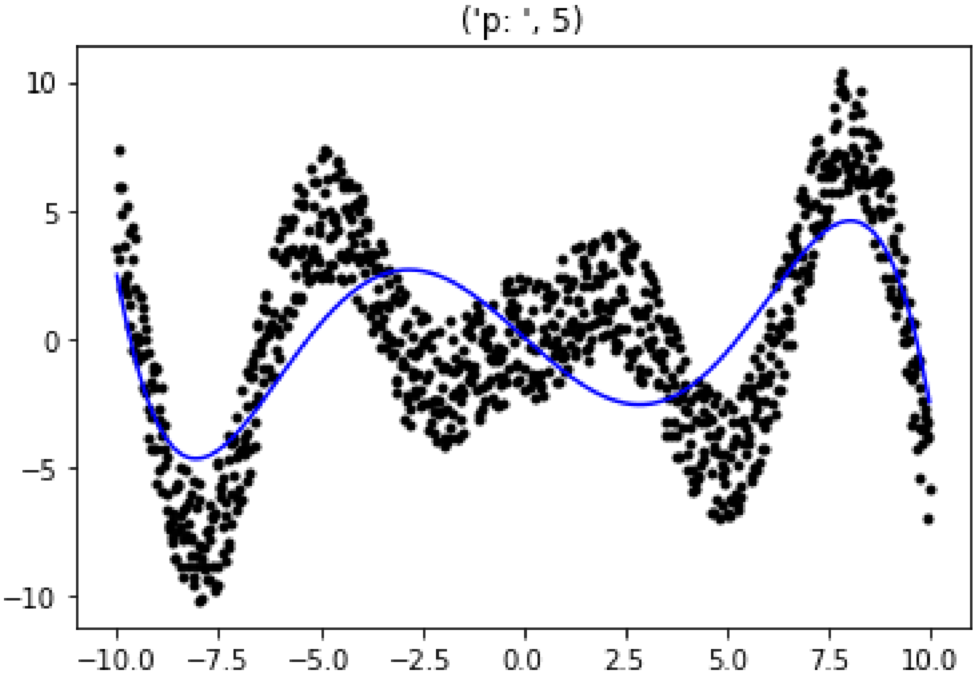
```
weight:
 [[ 0.03216591]
 [-0.12751761]
 [-0.00093864]
 [ 0.00562418]]
MSE:  3041.6160271947165
```



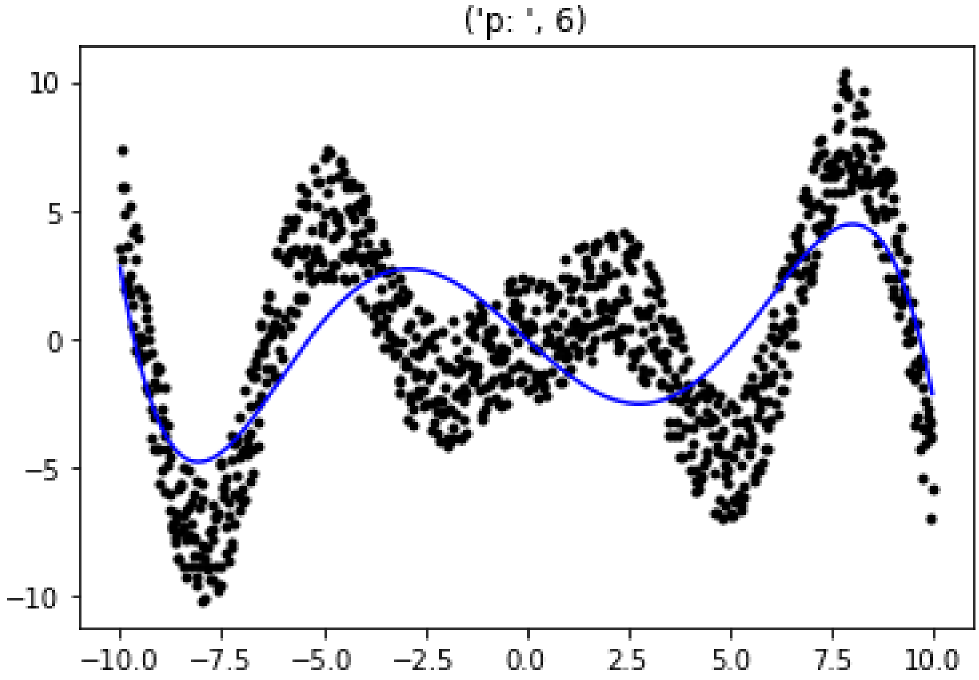
```
weight:
 [[ 4.64797126e-02]
 [-1.27538949e-01]
 [-2.38850853e-03]
 [ 5.62447682e-03]
 [ 1.69302489e-05]]
MSE:  3041.824953201113
```



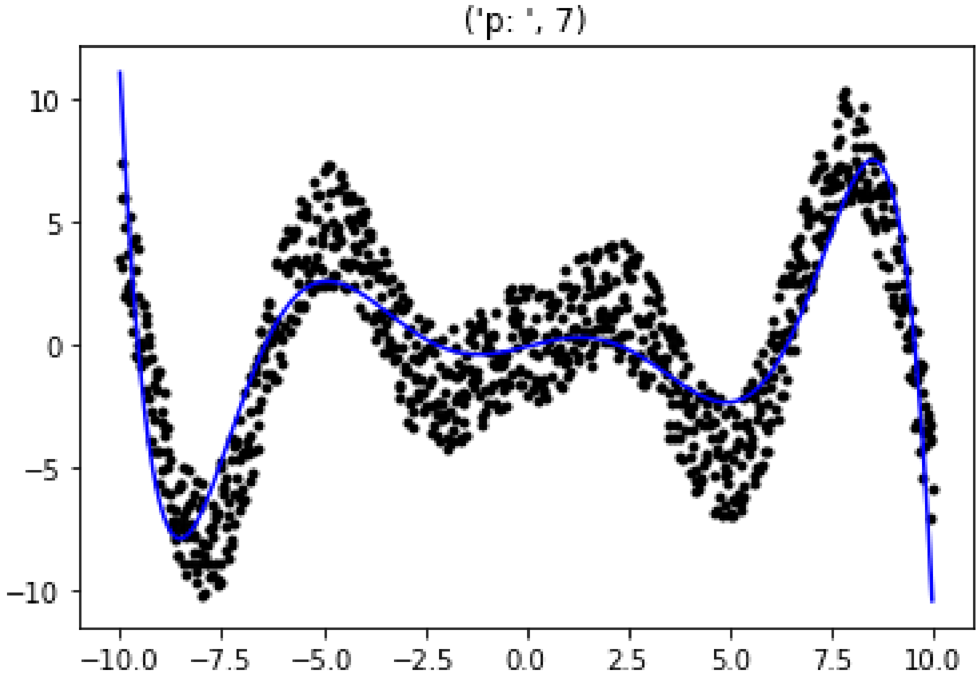
```
weight:
 [[ 8.25673232e-02]
 [-1.41860546e+00]
 [-4.00443743e-03]
 [ 6.58192386e-02]
 [ 3.11961440e-05]
 [-5.40923274e-04]]
MSE:  2614.960449815954
```



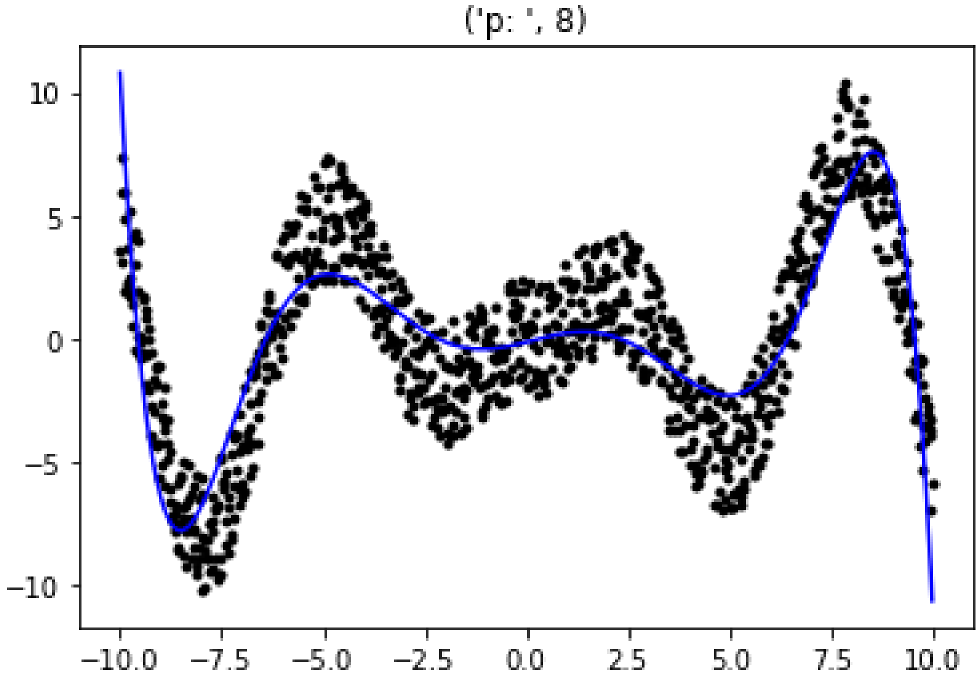
```
weight:
 [[-1.61276813e-02]
 [-1.41821848e+00]
 [ 1.70673637e-02]
 [ 6.58054328e-02]
 [-6.01922571e-04]
 [-5.40815439e-04]
 [ 4.64043991e-06]]
MSE:  2613.4603350041675
```



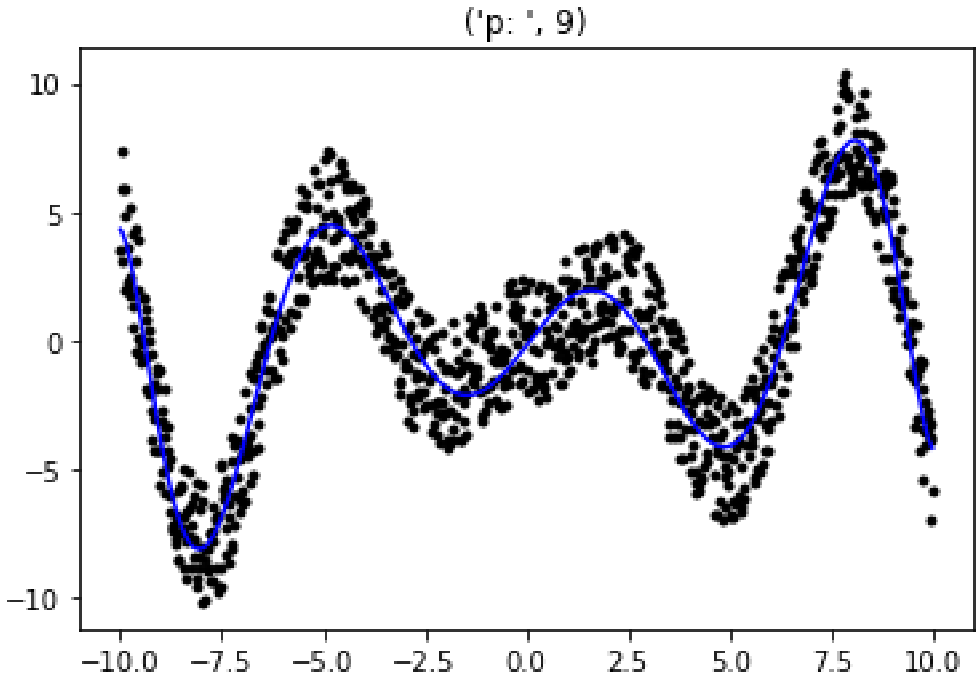
```
weight:
 [[-7.97623773e-02]
 [ 4.24066695e-01]
 [ 2.23468852e-02]
 [-1.00108187e-01]
 [-7.13775677e-04]
 [ 3.10611812e-03]
 [ 5.31761930e-06]
 [-2.25453645e-05]]
MSE:  1756.327317082739
```



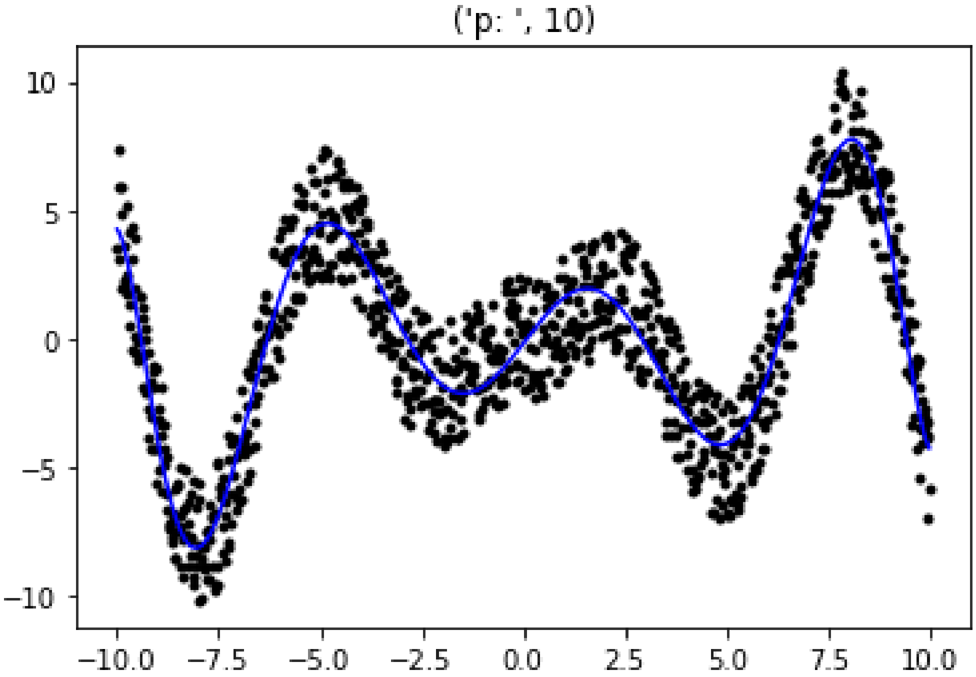
```
weight:
 [[-1.50507492e-01]
 [ 4.24563012e-01]
 [ 4.82611663e-02]
 [-1.00140454e-01]
 [-2.14126831e-03]
 [ 3.10671180e-03]
 [ 3.00474156e-05]
 [-2.25486409e-05]
 [-1.32321187e-07]]
MSE:  1757.5552925504026
```



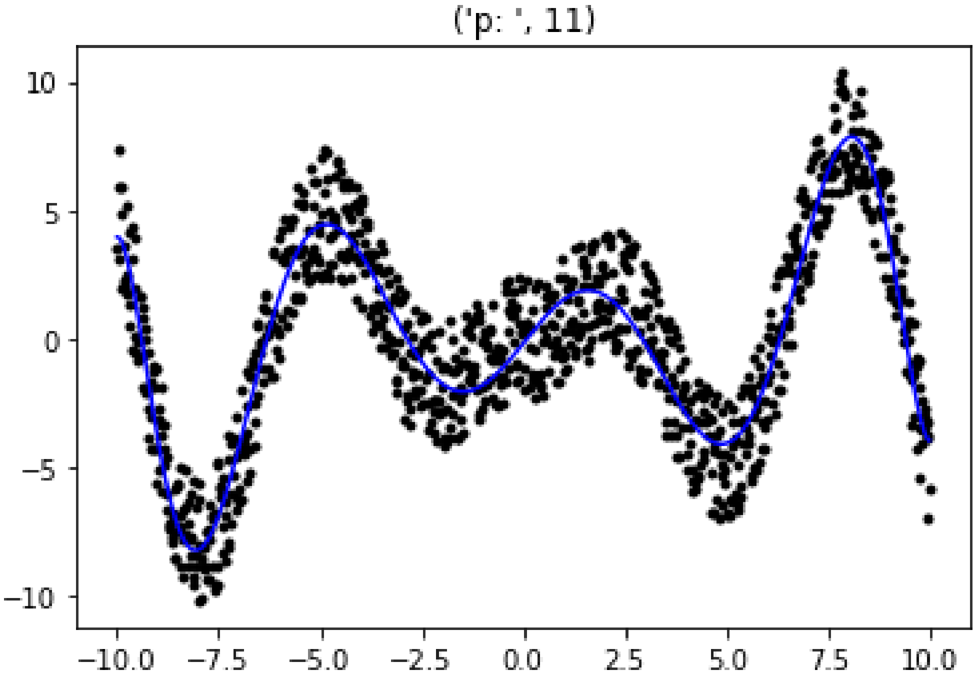
```
weight:
 [[-2.13787763e-01]
 [ 2.06454219e+00]
 [ 5.63306411e-02]
 [-3.41390540e-01]
 [-2.43578865e-03]
 [ 1.25164980e-02]
 [ 3.40932440e-05]
 [-1.56862457e-04]
 [-1.50932508e-07]
 [ 6.33464732e-07]]
MSE:  1302.6409222493905
```



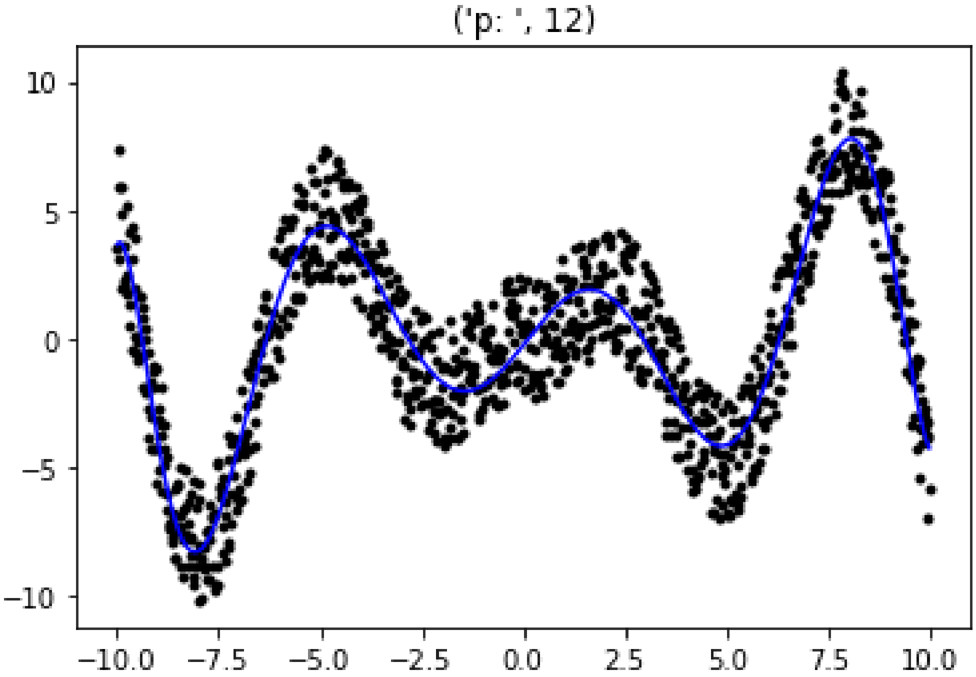
```
weight:
 [[-1.96941292e-01]
 [ 2.06438273e+00]
 [ 4.69237954e-02]
 [-3.41374334e-01]
 [-1.61971625e-03]
 [ 1.25159843e-02]
 [ 9.63061978e-06]
 [-1.56856049e-04]
 [ 1.45711717e-07]
 [ 6.33437260e-07]
 [-1.25050342e-09]]
MSE:  1302.4568004979596
```



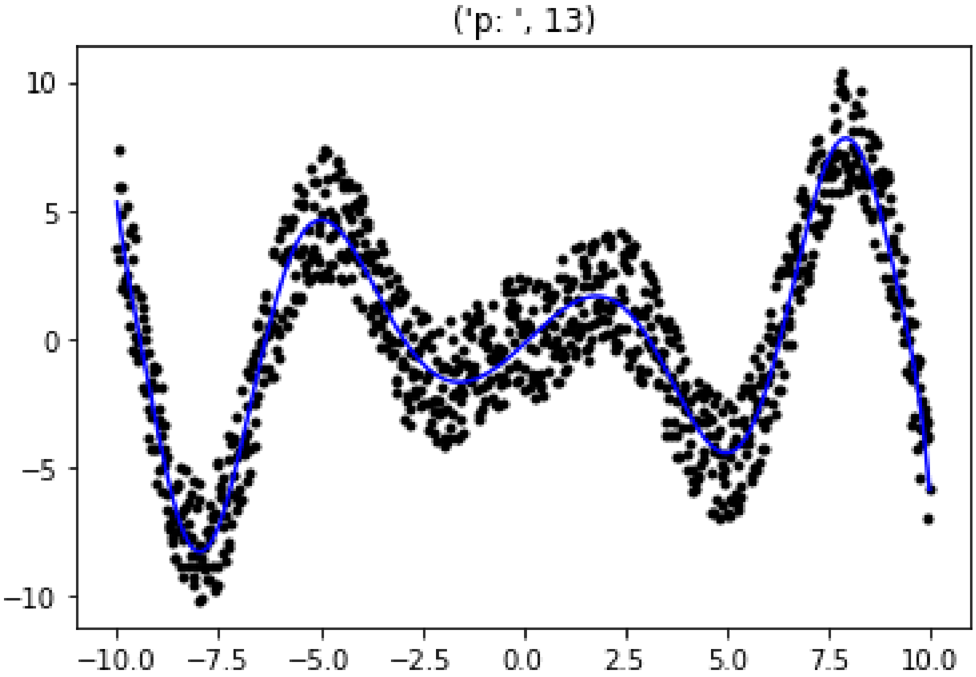
```
weight:
 [[-1.93602122e-01]
 [ 1.98080671e+00]
 [ 4.63455294e-02]
 [-3.23157790e-01]
 [-1.58868141e-03]
 [ 1.14215365e-02]
 [ 8.92594979e-06]
 [-1.30281074e-04]
 [ 1.52820319e-07]
 [ 3.53158095e-07]
 [-1.27676845e-09]
 [ 1.06888284e-09]]
MSE:  1301.052009174007
```



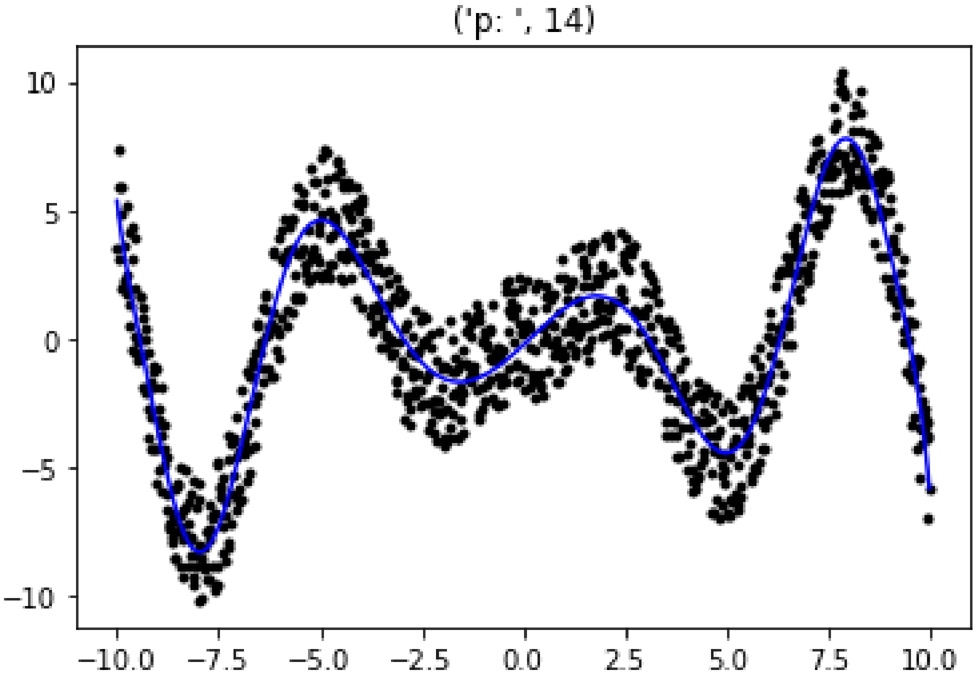
```
weight:
 [[-2.64139292e-01]
 [ 1.98147440e+00]
 [ 1.01925129e-01]
 [-3.23254668e-01]
 [-8.53618036e-03]
 [ 1.14261508e-02]
 [ 3.23496507e-04]
 [-1.30377206e-04]
 [-6.24083016e-06]
 [ 3.54066772e-07]
 [ 5.82940455e-08]
 [ 1.06569225e-09]
 [-2.07220036e-10]]
MSE:  1300.0790240774631
```



```
weight:
 [[-2.44437073e-01]
 [ 1.47293757e+00]
 [ 9.77483342e-02]
 [-1.69266905e-01]
 [-8.24209139e-03]
 [-1.69767207e-03]
 [ 3.14053051e-04]
 [ 3.44837456e-04]
 [-6.08876807e-06]
 [-7.95891520e-06]
 [ 5.70981836e-08]
 [ 7.05307801e-08]
 [-2.03565210e-10]
 [-2.22382106e-10]]
MSE:  1279.6182758507362
```

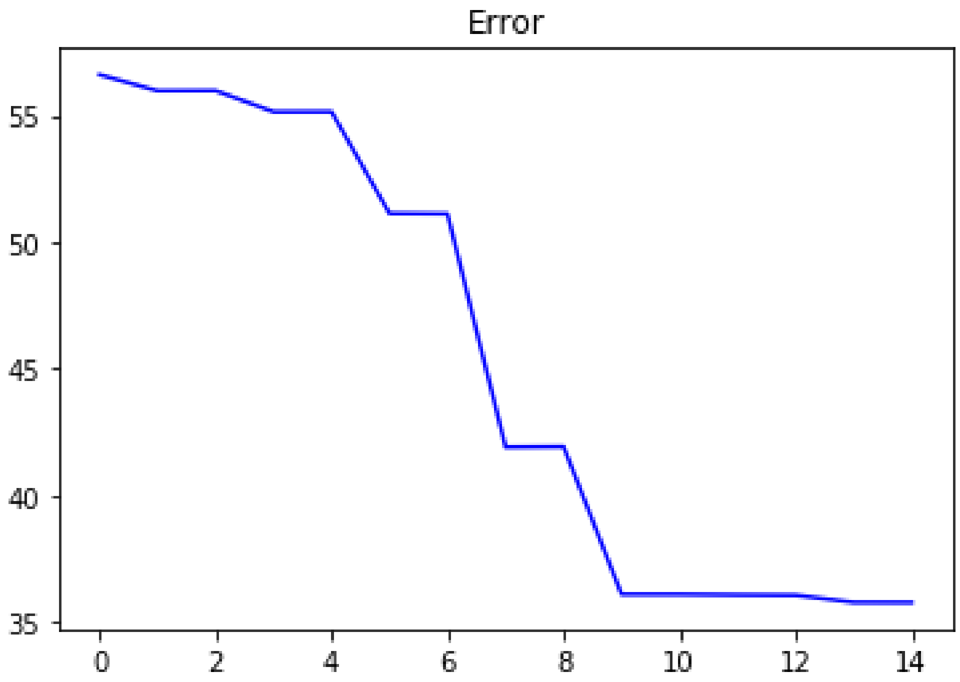


```
weight:
 [[-2.58717334e-01]
 [ 1.47301151e+00]
 [ 1.12787629e-01]
 [-1.69281311e-01]
 [-1.07956993e-02]
 [-1.69672037e-03]
 [ 4.75510327e-04]
 [ 3.44808374e-04]
 [-1.09237137e-05]
 [-7.95846505e-06]
 [ 1.31095748e-07]
 [ 7.05273523e-08]
 [-7.63092314e-10]
 [-2.22371907e-10]
 [ 1.65696953e-12]]
MSE:  1279.3783387353249
```



11 show error

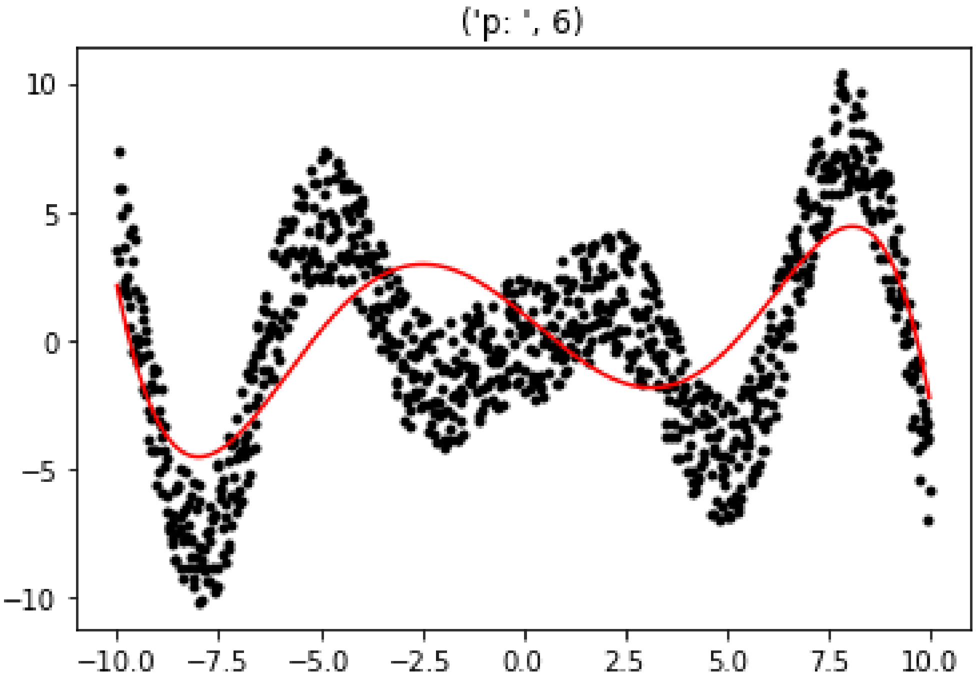
```
In [99]: x_axis = np.arange(0, 15, 1)
         plt.plot(x_axis, error, 'b')
         plt.title('Error')
         plt.show()
```



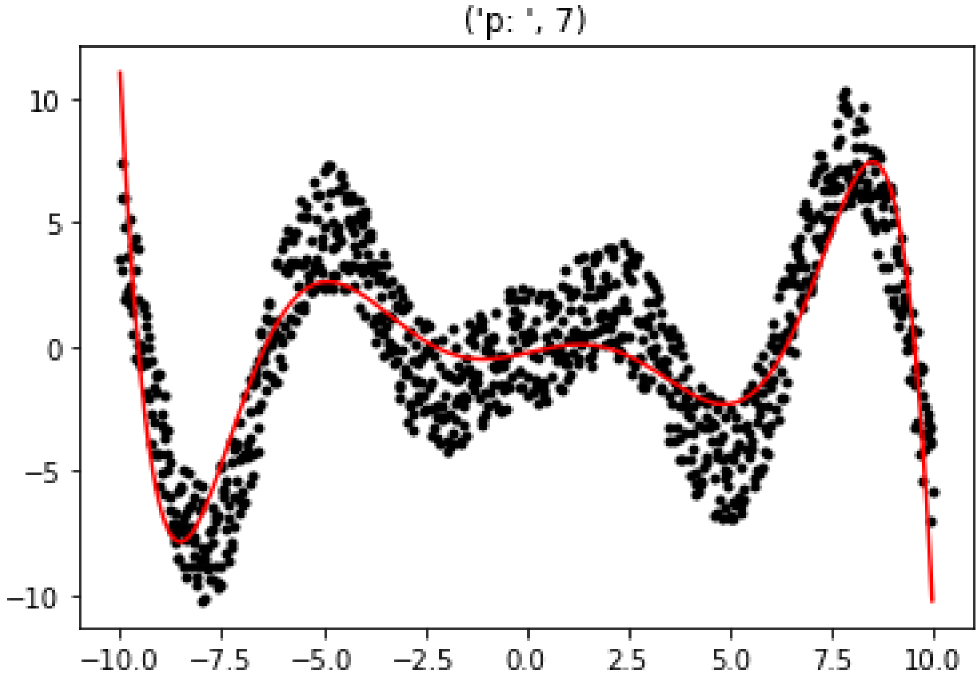
12 change lambda to 30

```
In [102]: error = []

          def training_r(n, lambda_, p):
              error = []; y = []
              y = func(n, lambda_, p)
              error = computeError(y2, y)
              Ylabel = n,' dimension'

              plt.plot(x, y2, 'k.')
              plt.plot(x, y, 'r')
              title = ("p: ", n)
              plt.title(title)
              plt.show()

              return error

          for n in range(6, 15):
              lambda_ = 30
              err = training_r(n, lambda_, r)
              error.append(err)
```

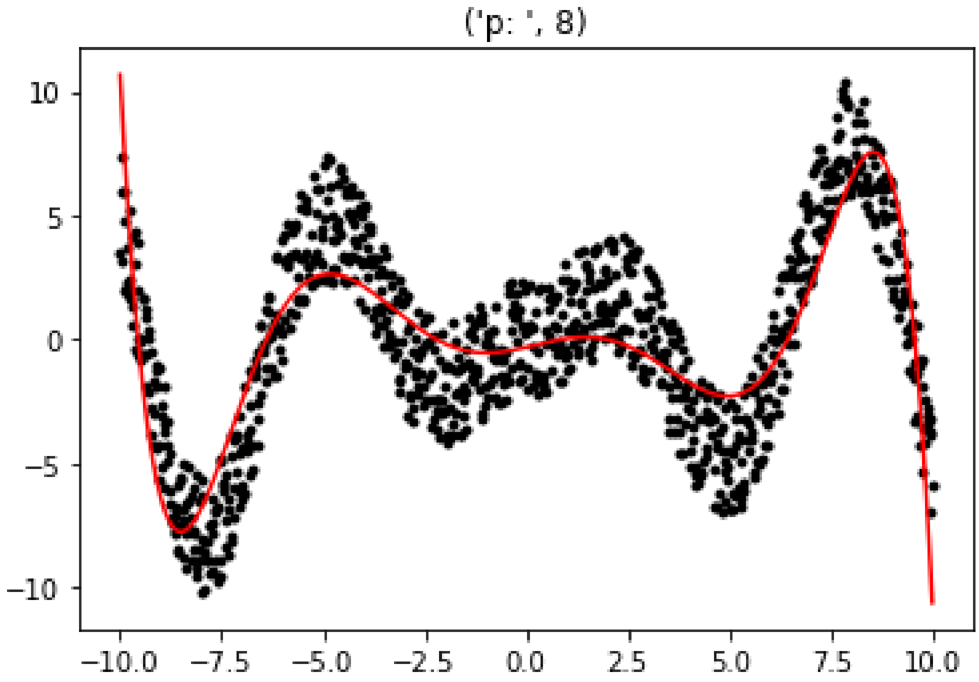
```
weight:
 [[ 1.01732870e+00]
 [-1.28736506e+00]
 [-6.86744647e-02]
 [ 6.11370213e-02]
 [ 1.21462272e-03]
 [-5.04351271e-04]
 [-6.35726110e-06]]
MSE:  2649.954671888378
```



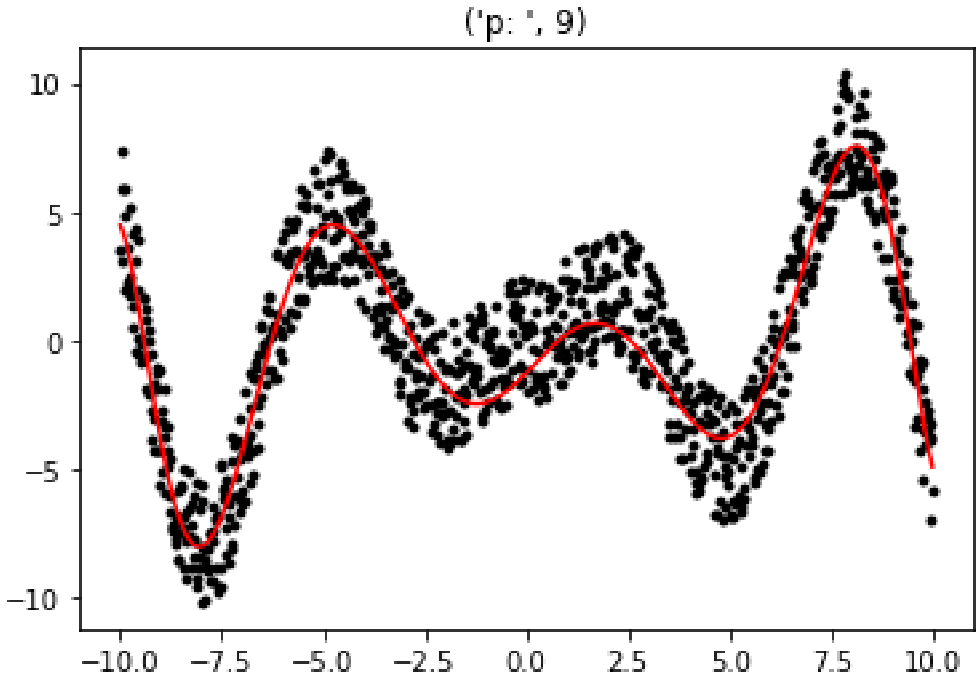
```
weight:
 [[-2.65639157e-01]
 [ 3.72704621e-01]
 [ 3.77683535e-02]
 [-9.67690710e-02]
 [-1.04049831e-03]
 [ 3.04468056e-03]
 [ 7.29565851e-06]
 [-2.22063101e-05]]
MSE:  1758.219343332191
```



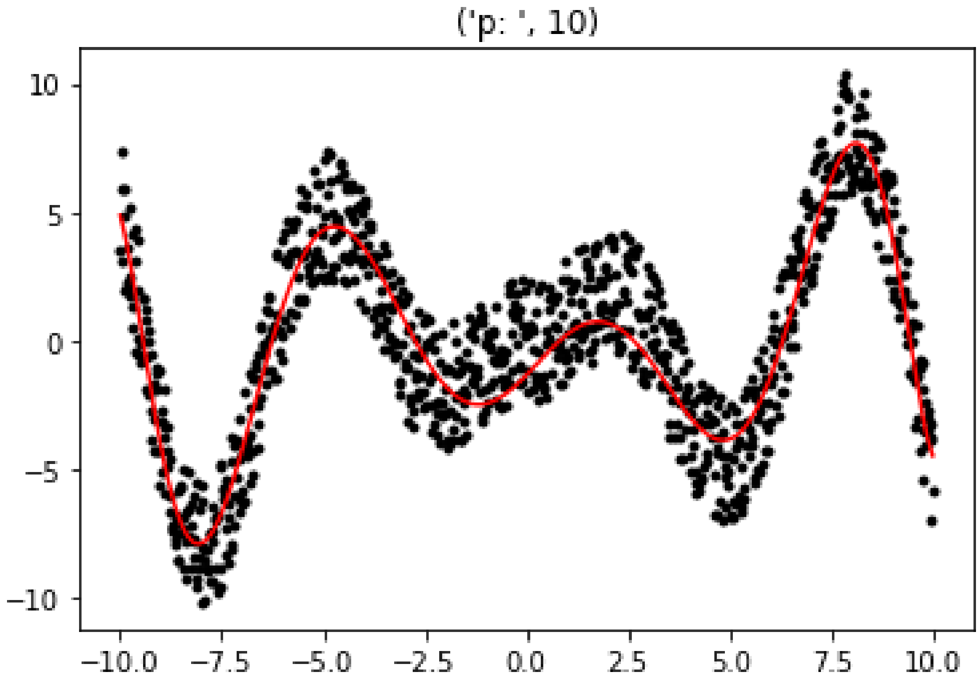
```
weight:
 [[-3.19432977e-01]
 [ 3.85411198e-01]
 [ 6.98024765e-02]
 [-9.75951424e-02]
 [-2.92748479e-03]
 [ 3.05987974e-03]
 [ 4.08476778e-05]
 [-2.22901895e-05]
 [-1.82003755e-07]]
MSE:  1759.2621554312122
```



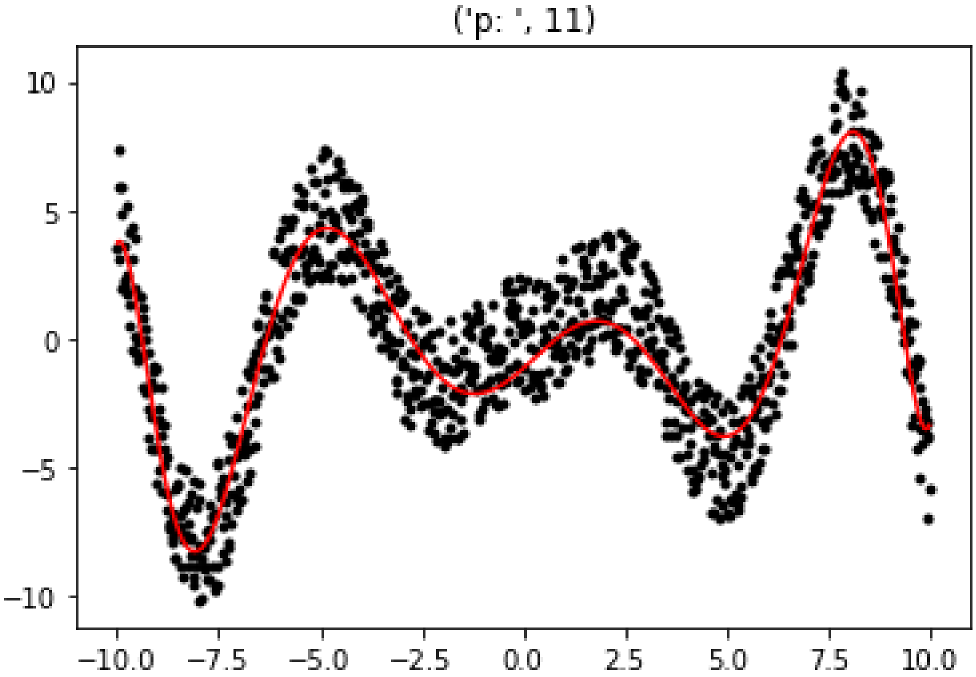
```
weight:
 [[-1.27442221e+00]
 [ 1.61758915e+00]
 [ 1.91582321e-01]
 [-2.95968263e-01]
 [-7.37221543e-03]
 [ 1.10765160e-02]
 [ 1.01904979e-04]
 [-1.38901461e-04]
 [-4.62875038e-07]
 [ 5.56464066e-07]]
MSE:  1354.830055889791
```



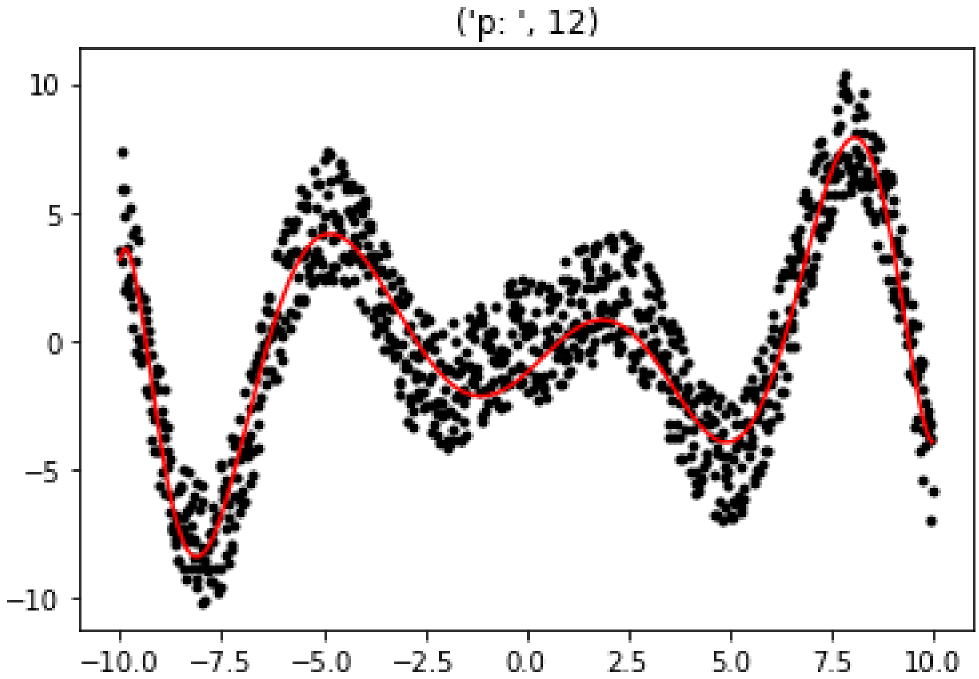
```
weight:
 [[-1.34538759e+00]
 [ 1.63286018e+00]
 [ 2.45807753e-01]
 [-2.97520204e-01]
 [-1.22935785e-02]
 [ 1.11257158e-02]
 [ 2.51988908e-04]
 [-1.39515134e-04]
 [-2.29916142e-06]
 [ 5.59094945e-07]
 [ 7.78287271e-09]]
MSE:  1354.1277675210397
```



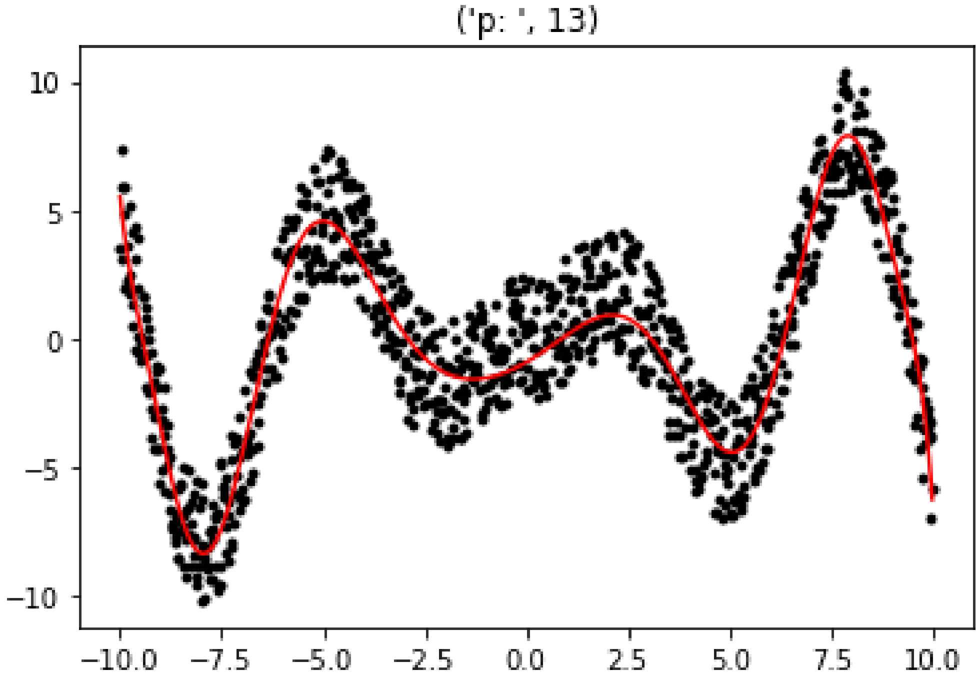
```
weight:
 [[-1.16572429e+00]
 [ 1.39237590e+00]
 [ 2.14694283e-01]
 [-2.37779650e-01]
 [-1.06237561e-02]
 [ 7.35496035e-03]
 [ 2.14074301e-04]
 [-4.55600618e-05]
 [-1.91668469e-06]
 [-4.47659457e-07]
 [ 6.36968849e-09]
 [ 3.88075552e-09]]
MSE:  1342.3359469744191
```



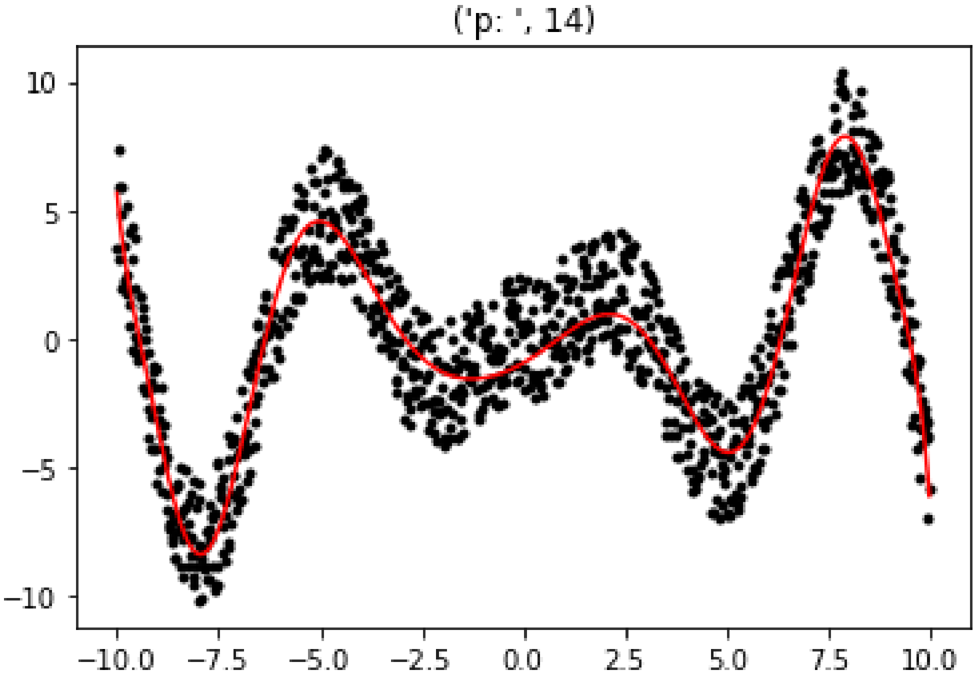
```
weight:
 [[-1.27445560e+00]
 [ 1.40914463e+00]
 [ 3.16108379e-01]
 [-2.40212703e-01]
 [-2.36168730e-02]
 [ 7.47084706e-03]
 [ 8.07750981e-04]
 [-4.79743875e-05]
 [-1.40384807e-05]
 [-4.24838266e-07]
 [ 1.19617122e-07]
 [ 3.80062456e-09]
 [-3.94637337e-10]]
MSE:  1339.0599372074544
```



```
weight:
 [[-9.27115212e-01]
 [ 9.02232909e-01]
 [ 2.42473527e-01]
 [-5.80686469e-02]
 [-1.84322260e-02]
 [-9.04354456e-03]
 [ 6.41267351e-04]
 [ 5.69316341e-04]
 [-1.13576977e-05]
 [-1.14334670e-05]
 [ 9.85346365e-08]
 [ 9.69888764e-08]
 [-3.30204455e-10]
 [-3.01107166e-10]]
MSE:  1301.4463698590037
```

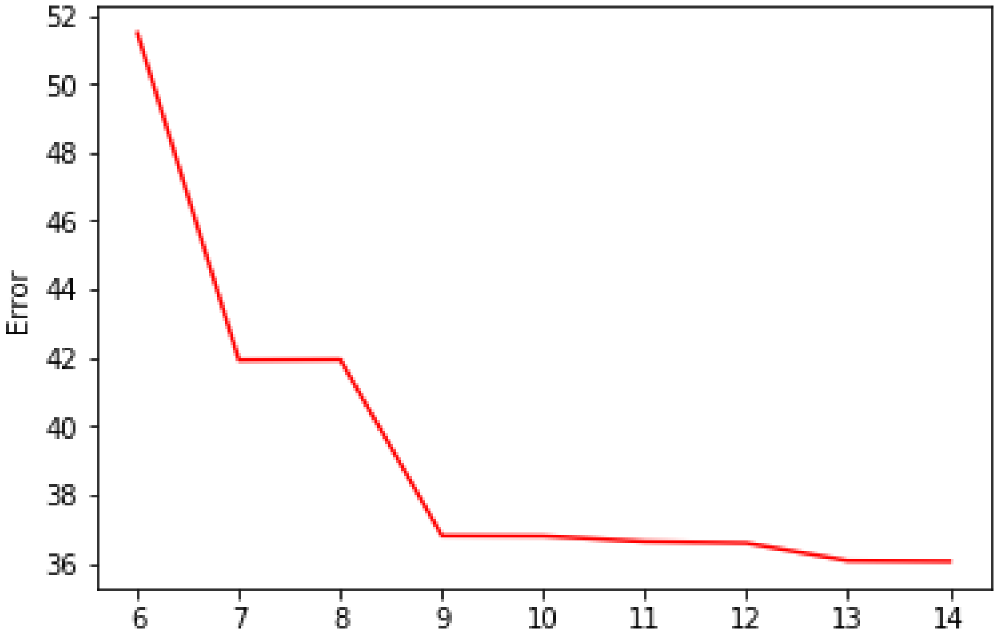


```
weight:
 [[-9.55794219e-01]
 [ 9.04077928e-01]
 [ 2.74373443e-01]
 [-5.84281369e-02]
 [-2.38911900e-02]
 [-9.01979628e-03]
 [ 9.87358699e-04]
 [ 5.68590629e-04]
 [-2.17350163e-05]
 [-1.14222342e-05]
 [ 2.57472573e-07]
 [ 9.69033407e-08]
 [-1.53255342e-09]
 [-3.00852658e-10]
 [ 3.56171404e-12]]
MSE:  1300.22514266042
```



13 show error

```
In [103]: x_axis = np.arange(6, 15, 1)
          plt.plot(x_axis, error, 'r')
          ylab = 'Error'
          plt.ylabel(ylab)
          plt.show()
```


Error
52
50
48
46
44
42
40
38
36
6
7
8
9
10
11
12
13
14