Project 1

Name: Yun Du

UIN :330006637

**The main idea of Program:**

The primary goal of the code is to determine the piecewise linear function's optimum cost given a set of data points.

The method finds the minimal cost of fitting a segment to a subset of the input points by iteratively computing the error of each feasible segment of the input points using linear regression. The cost includes both the segment mistake and the penalty for using too many segments.

Dynamic programming is used by the function ‘OptCost’ to determine the optimal cost and indices of the x list where the segments begin, allowing for a practical implementation. To reconstruct the piecewise linear regression, use the x list indexes where the segments begin.

**The pseudo code:**

**Part 1: compute the a, b, and error**

initialize N = length of x\_list

initialize err as N by N matrix of zeros

initialize x as array of elements in x\_list

initialize y as array of elements in y\_list

initialize xx as array of the squared values of x

initialize yy as array of the squared values of y

initialize tmp as array of the product of x and y

initialize y2 as N by N matrix of zeros

initialize xy as N by N matrix of zeros

initialize x2 as N by N matrix of zeros

initialize ysum as N by N matrix of zeros

initialize xsum as N by N matrix of zeros

for i from 0 to N-1:

for j from i to N-1:

y2[i, j] = y2[i, j-1] + yy[j]

x2[i, j] = x2[i, j-1] + xx[j]

xy[i, j] = xy[i, j-1] + tmp[j]

ysum[i, j] = ysum[i, j-1] + y[j]

xsum[i, j] = xsum[i, j-1] + x[j]

a = ((j-i+1) \* xy[i, j] - xsum[i, j] \* ysum[i, j]) / ((j-i+1) \* x2[i, j] - xsum[i, j] \* xsum[i, j])

b = (ysum[i, j] - a \* xsum[i, j]) / (j-i+1)

if i != j:

err[i, j] = y2[i, j] - 2\*a\*xy[i, j] + a\*a\*x2[i, j] - 2\*b\*ysum[i, j] + 2\*a\*b\*xsum[i, j] + (j-i+1) \* b\*b

**Part 2: compute optimal, return the optimal and its index, using Dynamic Programming**

Initialize OPT and inds as follows:

OPT = an array of length N filled with a very large number

OPT[0] = c

OPT[1] = c

inds = an array of N elements, where each element is a tuple (cost, indices)

inds[0] = (0, [])

inds[1] = (0, [])

Compute the error matrix err for all i, j pairs

For j in range(1, N):

Set temp = err[0, j] + c

Set inds[j] = (1, [j])

For i in range(2, j):

Set temp = err[i, j] + c + OPT[i-1]

If temp < OPT[j]:

Set OPT[j] = temp

Set inds[j] = (inds[i-1][0]+1, inds[i-1][1] + [j])

Return the last element of OPT and inds

**The proof of correctness:**

By using induction, we can demonstrate that the algorithm is valid. Let inds[j] be the indices of the subsequences that reach this optimal cost, and let OPT[j] be the optimal cost of the first j data points in the input sequence. We will demonstrate how the algorithm accurately computes OPT[j] and inds[j] for all j.

Basic case: The single data point is the only subsequence when j=1, and the best cost is c. (the cost parameter).

Suppose that the algorithm correctly calculates OPT[i] and inds[i] for every I j in the inductive step. We shall demonstrate how the algorithm appropriately computes OPT[j] and inds[j].

The procedure evaluates all subsequences that could finish at index j and computes OPT[j] by choosing the subsequence that minimizes the sum of the squared errors along with the cost parameter and the previous subsequence's optimal cost.

Let OPT[j][k] represent the optimal cost of the subsequence with k segments and an index of j. The procedure chooses the subsequence that minimizes the sum of the squared errors plus the cost parameter and the optimal cost of the preceding subsequence after considering all potential subsequences that terminate at index j-1 and have k-1 segments.

The minimum of OPT[j][k] for every k is the optimal cost of the subsequence with index j and k segments.

The method also determines the subsequence indices that result in the lowest cost. Let inds[j][k] represent the subsequence's indices, which reaches OPT[j][k], or the ideal cost.

The following argument is used by the method to accurately calculate OPT[j][k] and inds[j][k]:

The best subsequence must finish with a subsequence that ends at index I where I is between 1 and j-1, if the optimal subsequence that ends at index j has k segments. The best subsequence has k-1 segments and terminates at index i. The minimum of OPT[i][k-1] + err[i+1][j] + c, where err[i+1][j] is the sum of squared errors of a linear regression fit to the subsequence from i+1 to j, is the optimal cost of the subsequence that finishes at index j and includes k segments.

We can set inds[j][k] to be the indices of the subsequence that achieves the minimum in order to compute the indices of the subsequence that achieves the optimal cost OPT[j][k].

Therefore, the algorithm is correct.

**The Analysis of the time complexity:**

Time complexity : O(n^2)

Using stacked loops, the function calculates the part\_1 ‘error’ in O(n^2) time for all potential input point segments. Dynamic programming is then used to determine the best cost and indexes in part\_2, which takes O(n^2) time. So the total time complexity should be O(n^2).