An Iterative Solution

Here is the key equation again:

\[
\text{opt}(n+1) = 0
\]

\[
\text{opt}(j) = \max \{ \text{opt}(j+1), V_j + \text{opt}(p(j)) \}
\]

for each \( j \) between 1 and \( n \).

Our basic recursive algorithm was obtained by reading the equation as follows:

To compute \( \text{opt}(j) \), we need to compute \( \text{opt}(j+1) \) and \( \text{opt}(p(j)) \).

The iterative algorithm is obtained by reading the equation as follows:

Once we know \( \text{opt}(j+1) \) and \( \text{opt}(p(j)) \), we can immediately deduce \( \text{opt}(j) \).

The key is that both \( j+1 \) and \( p(j) \) are bigger than \( j \). So we compute

\( \text{opt}(n+1), \text{opt}(n), \ldots, \text{opt}(j+1), \text{opt}(j), \ldots, \text{opt}(1) \)
in that order. Thus when we wish to compute \( \text{opt}(i) \), both \( \text{opt}(j+1) \) and \( \text{opt}(p(j)) \) have already been computed. Here is the iterative algorithm.

\[
M[n+1] \leftarrow 0
\]

For \( j \leftarrow n \) down to 1

\[
M[i] \leftarrow \max \{ M[i+1], v_i + M[p(i)] \}
\]

A proof by induction that is based on the key equation can be given to establish the correctness of this algorithm (if needed, one is not completely sure of the correctness.)

The running time of the algo is clearly \( O(n) \).
If we want the optimal solution and not just its value, this can be easily recovered.

Note that $O_j$, the optimal solution for $C_{n+1}$ is better of two options:

1. $O_{j+1}$
2. $v_j$ plus an optimal solution for $C_i$.

Print $O_{\text{opt}}(i)$

if $j = n+1$ return.

if $v_j > v_j + M(C_{j+1})$

Print $O_{\text{opt}}(j+1)$

else

Print $(v_j)$

Print $O_{\text{opt}}(p(j))$
Segmented Least Squares

This problem is motivated by the issue of fitting multiple lines to a set of points. We assume a subroutine that given a set $Q$ of points finds a line $l$ that best fits $Q$. The quality of the fit is measured by an error term: if the error is 0, the fit is very good, and the larger it is, the worse is the fit.

Let's call this subroutine $\text{linefit}(Q)$. It returns $l(Q)$, the line that best fits $Q$, and $\text{error}(Q)$, an error term measuring the quality of the fit.

The input to the Segmented Least Squares problem is a sequence of points $P = \{p_1, p_2, \ldots, p_n\}$. The goal is to
partition the sequence into a set of "segments" so as to minimize a certain penalty function to be defined below. A segment is just a contiguous subsequence of the original sequence: it has the form \( \{p_i, \ldots, p_j\} \) for some \( i \leq j \).

The penalty of a partition is defined to be the sum of

(i) The number of segments in the partition times a given fixed multiplier \( c > 0 \).

(ii) for each segment of partition, sum euor of the segment as computed by lineFit().

Another way of describing the penalty of a partition is the sum, over each segment, of \( c + \text{euor} \) of that segment.
Some useful notation: let $E_{i,j}$ denote error $(\{p_i \ldots p_j\})$.

Consider an optimal partition for $\{p_i \ldots p_n\}$ and suppose that its last segment is $\{p_i \ldots p_j\}$. We can think of the optimal partition as being a partition $T$ at $\{p_i \ldots p_{i-1}\}$ followed by the segment $\{p_i \ldots p_j\}$.

Thus, penalty of optimal partition

$= \text{penalty of } T + C + E_{i,n}$.

Claim: $T$ must be optimal for $\{p_i \ldots p_{i-1}\}$.

The claim is easily proved by contradiction.

The claim says that if we knew $i$, we could compute an optimal partition for $\{p_i \ldots p_{i-1}\}$, add segment $\{p_i \ldots p_j\}$
and we would have an optimal partition for \( S_{P_i, \ldots, P_n} \).

We don't know \( i \), of course. But we can obtain an optimal partition for \( S_{P_i, \ldots, P_n} \) by taking the best of the following \( n \) options:

For each \( 1 \leq i \leq n \),

the optimal partition of \( S_{P_i, \ldots, P_n} \) followed by segment \( \{P_i, \ldots, P_n\} \).

Let \( \text{Opt}(j) \) denote the penalty of the optimal partition of \( S_{P_j} \).

Define \( \text{Opt}(0) = 0 \).

We have the following equation:

\[
\text{Opt}(n) = \min_{1 \leq i \leq n} (\text{Opt}(i-1) + C + c_{i,n}).
\]
In general, for any \( 1 \leq j \leq n \),

\[
\text{Opt}(i) = \min_{1 \leq i \leq j} (\text{Opt}(i-1) + C + e_{i,j}).
\]

Let us translate this into an iterative algorithm. Once we have \( \text{opt}(0), \ldots, \text{opt}(j-1) \), we can easily compute \( \text{opt}(i) \) using the above equation.

Array \( M[0..n] \)

\( M[0] \leftarrow 0 \)

For each \( 1 \leq i \leq j \leq n \),

compute \( e_{i,j} \) using \( \text{line}\_\text{fit} (p_{i}, \ldots, p_{j}) \).

For \( j \leftarrow 1 \) to \( n \) do

\[
M[j] \leftarrow \min_{1 \leq i \leq j-1} (M[i] + C + e_{i,j}).
\]

// That's a for loop where \( i \) ranges from \( 1 \) to \( j-1 \)