Objectives

- Dynamic Programming
  - Segmented Least Squares

Summary: Properties of Problems for DP

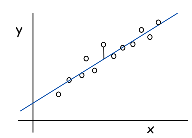
- Polynomial number of subproblems
- Solution to original problem can be easily computed from solutions to subproblems
- Natural ordering of subproblems, easy to compute recurrence

SEGMENTED LEAST SQUARES

Least Squares

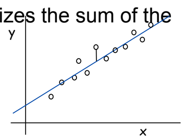
- Foundational problem in statistic and numerical analysis
- Given *n* points in the plane: (x_1, y_1), (x_2, y_2) , . . . , (x_n, y_n)
- Find a line $y = ax + b$ that minimizes the sum of the squared error
  - "line of best fit"

Sum of squared error $SSE = \sum_{i=1}^{n} (y_i - ax_i - b)^2$



Least Squares

- Foundational problem in statistic and numerical analysis
- Given *n* points in the plane: (x_1, y_1), (x_2, y_2) , . . . , (x_n, y_n)
- Find a line $y = ax + b$ that minimizes the sum of the squared error
  - "line of best fit"



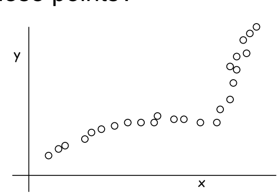
Sum of squared error $SSE = \sum_{i=1}^{n} (y_i - ax_i - b)^2$

- Closed form solution. Calculus $\Rightarrow$ min error is achieved when

$$a = \frac{n \sum_i x_i y_i - (\sum_i x_i)(\sum_i y_i)}{n \sum_i x_i^2 - (\sum_i x_i)^2}, \quad b = \frac{\sum_i y_i - a \sum_i x_i}{n}$$

Least Squares

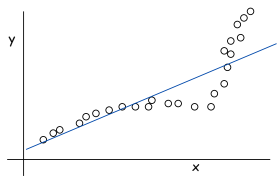
- What happens to the error if we try to fit one line to these points?



- What pattern does it seem like these points have?

Least Squares

- What happens to the error if we try to fit one line to these points?
  - Large error

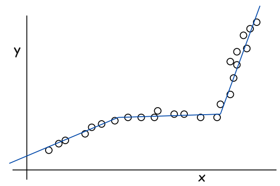


- Pattern: More like 3 lines

Segmented Least Squares

- Points lie roughly on a **sequence** of line segments
- Given *n* points in the plane $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ with $x_1 < x_2 < \ldots < x_n$, find a sequence of lines that **minimizes f(x)**

If I want the ***best*** fit, how many lines should I use?



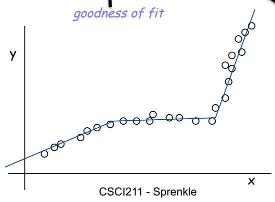
Segmented Least Squares

- Points lie roughly on a **sequence** of line segments
- Given *n* points in the plane $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ with $x_1 < x_2 < \ldots < x_n$, find a sequence of lines that **minimizes f(x)**

What's a reasonable choice for *f(x)* to balance *accuracy* and *parsimony*?

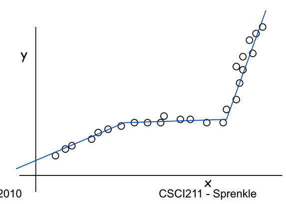
goodness of fit

number of lines



Segmented Least Squares

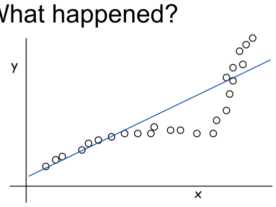
- Points lie roughly on a **sequence** of several line segments.
- Given *n* points in the plane $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ with $x_1 < x_2 < \ldots < x_n$, find a sequence of lines that minimizes:
  - *E*: sum of the sums of the squared errors in each segment
  - *L*: the number of lines
- **Tradeoff function**: $E + c\,L$, for some constant $c > 0$.



How should we define an optimal solution? …

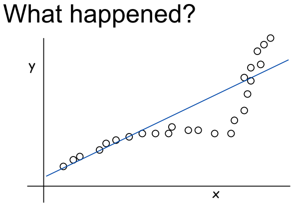
Segmented Least Squares

- What made it seem like the points were in 3 lines? What happened?



Segmented Least Squares

- What made it seem like the points were in 3 lines? What happened?



- Looking for *change* in linear approximation
  - Where to partition points into line segments

Recall: Properties of Problems for DP

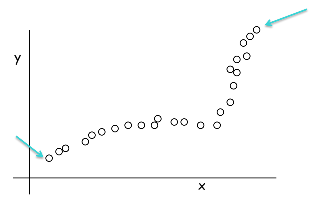
- Polynomial number of subproblems
- Solution to original problem can be easily computed from solutions to subproblems
- Natural ordering of subproblems, easy to compute recurrence

We need to:
- Figure out how to break the problem into subproblems
- Figure out how to compute solution from subproblems
- Define the recurrence relation between the problems

Toward a Solution

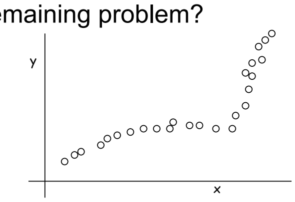
- Consider just the first or last point

What do we know about those points? their segments? cost of a segment?



Toward a Solution

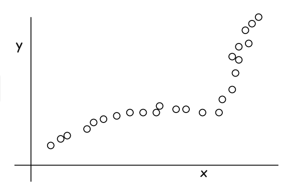
- p_n can only belong to one segment
  - Segment: $p_i, \dots, p_n$
  - Cost: c (cost for segment) + error of segment
- What is the remaining problem?



Toward a Solution

- p_n can only belong to one segment
  - Segment: $p_i, \dots, p_n$
  - Cost: c (cost for segment) + error of segment
- What is the remaining problem?
  - Solve for $p_1, \dots, p_{i-1}$

Goal: Formulate as a recurrence



Dynamic Programming: Multiway Choice

- Notation.
  - **OPT(j)** = minimum cost for points $p_1, p_{i+1}, \dots, p_j$.
  - **e(i, j)** = minimum sum of squares for points $p_i, p_{i+1}, \dots, p_j$.
- How do we compute OPT(j)?
  - Last problem: binary decision (include job or not)
  - This time: multiway decision
    - Which option do we choose?

Dynamic Programming: Multiway Choice

- Notation.
  - **OPT(j)** = minimum cost for points $p_1, p_{i+1}, \dots, p_j$.
  - **e(i, j)** = minimum sum of squares for points $p_i, p_{i+1}, \dots, p_j$.
- To compute OPT(j):
  - Last segment contains points $p_i, p_{i+1}, \dots, p_j$ for some i
  - Cost = e(i, j) + c + OPT(i-1).

$$OPT(j) = \begin{cases} 0 & \text{if } j = 0 \\ \min\limits_{1 \le i \le j} \{ e(i,j) + c + OPT(i-1) \} & \text{otherwise} \end{cases}$$

Segmented Least Squares: Algorithm

```
INPUT: n, p1,…,pN , c

Segmented-Least-Squares()
   M[0] = 0
   e[0][0] = 0
   for j = 1 to n
      for i = 1 to j
         e[i][j] = least square error for the
                      segment pi, …, pj

   for j = 1 to n
      M[j] = min 1 ≤ i ≤ j (e[i][j] + c + M[i-1])

   return M[n]
```

Costs?

Segmented Least Squares: Algorithm Analysis

```
INPUT: n, p1,…,pN , c

Segmented-Least-Squares()
   M[0] = 0
   e[0][0] = 0
   for j = 1 to n
      for i = 1 to j
         e[i][j] = least square error for the
                      segment pi,…, pj

   for j = 1 to n
      M[j] = min 1 ≤ i ≤ j (e[i][j] + c + M[i-1])

   return M[n]
```

can be improved to $O(n^2)$ by pre-computing various statistics

$O(n^3)$

$O(n^2)$

- Bottleneck: computing e(i, j) for $O(n^2)$ pairs, O(n) per pair using previous formula

How Do We Find the *Solution*?

Post-Processing: Finding the Solution

```
FindSegments(j):
   if j = 0:
      output nothing
   else:
      Find an i that minimizes ei,j + c + M[i-1]
      Output the segment {pi, …, pj}
      FindSegments(i-1)
```

Cost? O(n)