研究生算法课课堂笔记

上课日期：2016年10月24日　　第(1)节课

组长学号及姓名：1601214440 李冈嶷
组员学号及姓名：1601214432 顾耀
组员学号及姓名：1601214430 高铮

内容简介：

本节课主要有以下三方面内容：

1、 分段最小二乘问题
2、 最长递减子序列
3、 求解子序列最大和

详细内容：

一、分段最小二乘问题(segmented least squares)：一维动规+遍历

导言：

在一条线段情况：平面上存在 n 个点 (x_1,y_1)，(x_2,y_2) ……，(x_n,y_n)。我们可以用一条线段进行拟合，使得其方差和$SSE=\sum(y_i-a\times x_i-b)^2$最小。可以利用高等数学的知识求解，计算出 a，b 值如下：

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

题目描述及分析：

现在，我们需要用多条线段去进行拟合，对于局部来说，计算方式和一条线段计算方式一致。但对于总体来说，一方面，如果拟合线过少，会导致精确度不足；另一方面，如果不加限制，不断添加线段数量，拟合的总误差肯定是不断下降的，而最终会出现所有的点都会被拟合，或者说过拟合。为了平衡精确性（accuracy）和简约性（parsimony），我们引入一个惩罚常数 c，这样整体的代价函数就变成：

$$f(x)=E+cL$$

其中 E 是选用线段总的 SSE，c 是惩罚常数，L 是线段数量。我们的目标是求出最小的$f(x)$。

思路：（点的下标从 1 开始）

$OPT(j)$：从 1 到第 j 点的最小的开销。

$e(i,j)$：从 i 到 j 点的 SSE（假设已经算出）

如果存在多条线段拟合的情况，那么最后一条线段肯定会从某个地方“断开”，不妨记断开的地方为 i，那么其总开销为$e(i,j)+c+OPT(i-1)$（最极端情况是$i=1$，这时令$OPT(0)=0$,我们就可以得出一条线段去拟合的情况）。现在我们要求出最小的总开销，我们就必须遍历所有可能断开的点，找出最小的开销，于是有：

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

这样我们只要返回$OPT(n)$就可以得到答案。

伪代码：

```
FOR j = 1 TO n
    FOR i = 1 TO j
        Compute the least squares e(i, j) for the segment pi, pi+1, …, pj.
M [0] ← 0.
    FOR j = 1 TO n
        M [j] ← min 1 ≤ i ≤ j { eij + c + M [ i - 1] }.
RETURN M[n].
```

e(i, j)的求解：

到此，我们已经得到了一个解决本题的方法，但是如果直接用伪代码红色部分蛮力计算生产**e(i, j)**的矩阵，复杂度为$O(n^3)$，存储**e(i, j)**空间复杂度为$O(n^2)$。正确的计算方法应该是不显式地存储矩阵。每次需要计算**e(i, j)**时，利用上一个已经求出的**e(i + 1, j)**来计算**e(i, j)**，具体如下：

第一步，设辅助变量，令

$S_x(i,j) = \sum_{n=i}^{j} x_n$

$S_y(i,j) = \sum_{n=i}^{j} y_n$

$S_{xy}(i,j) = \sum_{n=i}^{j} x_n y_n$

$S_{xx}(i,j) = \sum_{n=i}^{j} {x_n}^2$

$S_{yy}(i,j) = \sum_{n=i}^{j} {y_n}^2$

这些辅助变量，在一次（固定 j 的）遍历中，不需要保留所有元素，仅需利用前一组x个元素求得的和，求出x + 1个元素的和，例如：S_x（i + 1，j）$+ x_i$可以得到S_x（i，j），这样每一步更新复杂变量的开销为O(1)。

第二步，计算a（i，j），b（i，j）

由于辅助变量已经算出，可以利用辅助变量带入公式直接算出a（i，j），b（i，j）

时间复杂度为O(1)

第三部，计算e（i，j）

首先要对e(i, j)进行展开：

$$
\begin{aligned}
\mathrm{e(i,j)} &= \sum (y_i - a \times x_i - b)^2 \\
&= \sum\left[y_i^2 - 2\mathrm{a} \times x_i \times y_i - 2\mathrm{b} \times y_i + a^2 x_i^2 + 2\mathrm{ab} \times x_i + b^2\right] \\
&= \mathrm{S_{yy}} - 2\mathrm{a} \times \mathrm{S_{xy}} - 2\mathrm{b} \times \mathrm{S_y} + a^2 \times \mathrm{S_{xx}} + 2\mathrm{ab} \times \mathrm{S_x} + (\mathrm{j} - \mathrm{i} + 1) \times b^2
\end{aligned}
$$

（其中 a，b 是第二步计算出来的a（i，j），b(i，j)）

其时间复杂度为O(1)，则整个e_{ij}矩阵的求解需要的时间复杂度为$O(n^2)$。

空间复杂度为O(1)。

总结：

如果利用蛮力计算e_{ij}，整个程序时间与空间复杂度都受限于e_{ij}的计算和存储，总时间复杂度为$O(n^3)$，空间复杂度为$O(n^2)$。如果改进了e_{ij}的计算，那么时间总的时间复杂度为$O(n^2)$，空间复杂度为O(n)。这样可以看出，一维动规+遍历的好处在于，虽然时间复杂度是$O(n^2)$，但是空间复杂度依旧为O(n)。

二、最长递减子序列

问题描述：

给定一个整数序列，求出该序列的最长严格递减子序列的长度，以及这种长度的子序列的个数。

数学表示与分析：

{a1, a2, a3, … , an}的最长递减子序列，可以通过一维动规的方法求解：

1. 定义f_j为以 j 结尾的最长严格递减子序列的长度，定义g_j为以 j 结尾且长度为f_j的子序列个数。

2. 初始值设置：$f_0 = 0$，$g_0 = 1$

递推公式： $f_j = 1 + max f_i$， $0 \le i < j$ and $a_i > a_j$

$g_j = \sum g_i$， $0 \le i < j$ and $a_i > a_j$ and $f_i + 1 = f_j$

返回值： $max f_j$，$1 \le j \le n$

$\sum g_k$， 下标 k 满足$f_k = max f_j$

时间复杂度：$\theta(n^2)$

空间复杂度：$\theta(n)$，不能被优化，因为这里的f_j依赖于之前所有的f_i（$i < j$）值。

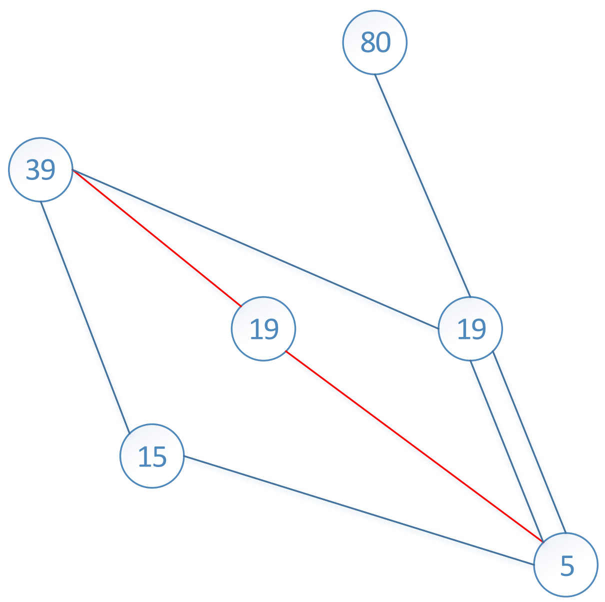
3 举例：

	+∞	39	15	19	80	19	5
f	0	1	2	2	1	2	3
g	1	1	1	1	1	2	3

上述序列在二维展开如下图所示，其中我们可以比较明显的看出来，有两段重复的序列，对于这种情况，应该做特殊处理，处理方法为：

将g_j数组进行从 n 到 0 的遍历，如果发现存在 data[i]=data[j] and f[i]=f[j] and i<j 的情况，则置 g[i]=0；如图中红色线段所示的那条子序列，满足上述情况。

因此，修正后$g_3 = 0$。所以，表格中的$g_6 = g_2 + g_3 + g_5 = 1 + 0 + 2 = 3$而不是 4。



三、求解子序列最大和

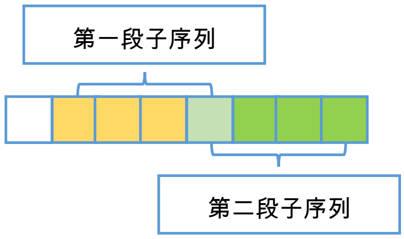
2-段子序列最大和：

背景条件

序列$a_1, a_2, \dots\dots a_n$中取$(a_{i1}, a_{j1}), (a_{i2}, a_{j2})$两段子序列，其中 $1 \le i_1 \le j_1 < i_2 \le j_2 \le n$，求两段子序列最大和。

※$j_1 < i_2$，不取等原因：

如果存在$j_1 = i_2$，则可能会发生如下情况，即两段首尾元素重合。



解题过程：

△复习：单段子序列最大和求解过程

令 f(j)表示以元素a_j结尾的子序列和，F(j)表示前 j 个元素组成序列中子序列最大和。

从前向后计算并保存F(j)，$\mathrm{F(j)} = \max\begin{cases} f(j) \\ F(j-1) \end{cases}$

最后求解g(j), 令g(j)从元素a_j开始，向前遍历所有可能的切分点，记录最大子序列和。

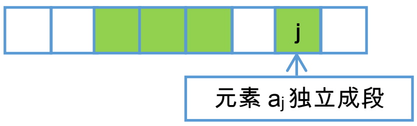
下面讨论 2-段子序列最大和问题求解过程

令：$f_1(j)$表示 1 段子序列最大和，以元素a_j结尾

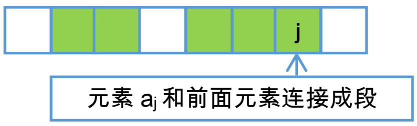
$f_2(j)$表示 2 段子序列最大和，第二段子序列以元素a_j结尾

求解F(j)，需要考虑两种情况：

1．当a_j元素独立作为最后一段子序列时，除元素a_j外，在其之前应仅有一段最优解，所以最大和应为$a_j + \mathrm{F(j-1)}$，如下图



2．当a_j元素和其前面若干元素整体组成后一段子序列时，除元素a_j外，在其之前仍应该有 2 段最优解，所以最大和应为$a_j + f_2(\mathrm{j}-1)$，如下图



※允许出现相邻两段首尾相连情况，如下图

即求解目标为：

$$f_2(\mathrm{j}) = \max\begin{cases} a_j + F(j-1) \\ a_j + f_2(j-1) \end{cases}$$

初始条件

在计算两段子序列最大和时，需要至少两个元素参与，所以$f_2(0), f_2(1)$均不存在。

初始条件$f_2(2) = a_1 + a_2$表示在只有两个元素时，将这两个元素全部选择，即为最大和。若题目中禁止返回负值，则求出最大和若小于零，应直接返回零值。

返回值

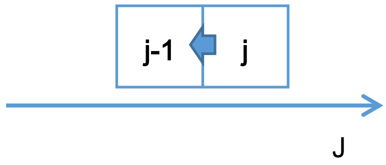
$$f_2(j), 2 \le j \le n$$

复杂度分析

时间复杂度O(n) ：对整个数组进行两次扫描。

空间复杂度O(n)

空间复杂度一般从依赖图中判断，本问题在求解每个元素的时依赖于前一个元素。



k-段子序列最大和：

下面将 2-段和问题推广至求解 k-段子序列最大和。

令f(j, k)表示从前 j 个元素中选取 k 段并且最后一段必须以第 j 个元素结尾时的最优解，F(j, k)表示前 j 个元素取 k 段的最大和（无论以哪个元素结尾）。同求解 2 段子序列一样，将根据元素a_j是否独立构成子序列，分两种情况讨论。第一种情况，当元素a_j单独成为一段子序列时，则在元素a_j前应有$j-1$段子序列。第二种情况，元素a_j和之前若干元素共同组成最后一段（第 k 段）子序列，则如果除去元素a_j，仍存在 k 段子序列，但其最后一个元素必须为a_{j-1}。

即：

$$\mathrm{f}(\mathrm{j},\mathrm{k}) = \max\begin{cases} \mathrm{F}(\mathrm{j}-1，\mathrm{k}-1) + a_j, & \text{当}a_j\text{元素独立成段} \\ \mathrm{f}(\mathrm{j}-1，\mathrm{k}) + a_j, & \text{当}a_j\text{元素和之前元素连接成段} \end{cases}$$

初始条件

1．序列中应包含至少 k 个元素

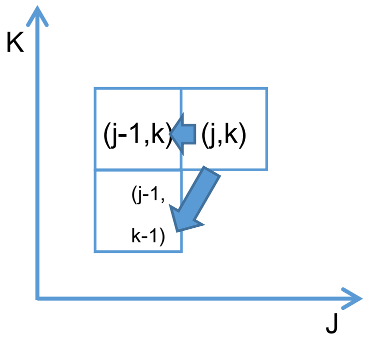
2．$\mathrm{f}(\mathrm{k},\mathrm{k}) = \sum_{i=1}^{k} a_i$

3．$\mathrm{f}(\mathrm{j},0) = 0$ 即认为从 j 个元素中取 0 段子序列最大和是 0；

依赖关系分析

存在依赖图如右图所示：

所以在求解时，对 K 从下向上计算，对 j 从左向右计算



复杂度分析

相当于对数组进行 k 次循环，

所以时间复杂度为 O(kn)

具体计算

1.计算F(j)

随着不断计算f（j）

F(j)中记录f（j）的running max，即目前为止max（f(j)）

f不需要保存，只记录F即可

eg:

	1	2	3	……
f(j)	7	3	13	……
F(j)	7	7	13	……

2.计算F(j,k)

求F(j,k)伪代码

```
f(j,0)=F(j,0)=0,k=0;
FOR k=1 TO K
    FOR j=1 TO n
        f(j,k)=a_j + max( F(j-1,k-1), f(j-1,k))
```

在计算F(j,k)过程中可以使用滚动数组优化，优化原则为：

当前的k计算出的值，只对下一个k+1时刻起作用。