# SMA 算法概述

李子煜

2025 年 4 月 4 日

## 1　如何构建窗口并拿到窗口内的价格

当我们在向交易所订阅 K 线的时候，我们可以轻松的通过交易所返回的 dataframe 获取连续的收盘价来计算 SMA，当订阅更新的时候，新时刻的 SMA 也被计算出来，但题目当中的数据是不连续的，所以我们第一步就是要” 补全” 回溯窗口 (W 个周期内) 的数据。

在代码中，时间点以及价格是一次一次生成的，这模拟了交易所向我们推送行情流的过程，所以当价格被生成后，首先要做的就是构造回溯窗口，算法如下：(本算法以 1 秒钟作为最小变动时间单位，交易时间为 A 股交易时间)

**Algorithm 1** 窗口构建算法

用户输入回溯窗口的大小 $W$
初始化变量 $i \leftarrow 0$ 作为目前窗口大小
**while** $i < W$ **do**
    时间回退 1 秒并判断回退后的时间是否在交易时间段
    **if** 若是交易时间 **then**
        把该时间加入窗口
        $i \leftarrow i + 1$
    **else**
        说明回退后的时间进入了闭市时间段，则需要进一步回退，若是 12:59 则回退至 11:30, 若是 9:29 则回退至 15:00
        把时间加入窗口
        $i \leftarrow i + 1$
    **end if**
**end while**

注意我们默认当数据点 (行情) 产生的时候一定是交易时段内的，所以回退 1 秒若进入了非交易时段只可能是 12:59 或是 9:29。(若前一天是周中则回退一天，若不是则回退到上周五)

接下来注意到题目中的假设，其假定在两条数据流之间股票价格一直为第一条数据当中的价格，所以我们可以根据这个逻辑来拿到窗口中不是数据流的那些点的数据，只要看他的时间被夹在哪两个数据流的点之间就好了。

**Algorithm 2** 窗口内获得价格算法

```
for 时间点 in 回溯窗口 do
    if 如果这个时间点本身就是数据流的一个时间点 then
        直接把数据流的价格赋值给这个时间点的价格
    else
        判断在已有的数据流当中，这个时间点位于哪两条数据之间
        if 如果能找到这样两条数据 then
            把时间上更早的那个数据流的价格赋值给这个时间点
        else
            找不到这样两条数据夹住这个时间点，说明这个时间点已经比最早的那个数据流还早了，就当作是没有数据，直接跳出循环就好了(反正再早也没数据了)
            跳出循环
        end if
    end if
end for
```

经过以上两个算法，我们可以得到任意一个数据流上的点的回溯窗口，以及窗口内所有点的价格，这些价格要么是从题目假设得到的前一个数据流点的价格，要么超出了最早的数据，导致这个回溯窗口实际上长度达不到我们的要求，直接被截断了。

## 2 计算 SMA

如果真的这样先得到窗口再计算是不行的，因为窗口本身的内存就是 O(Window) 了，超出了题目的要求，所以我们应该是回溯的时候就维护两个变量，一个计算回溯窗口内的总价格，一个计算有效数据的个数，最后两个做商就是 SMA 了

**Algorithm 3** 计算 SMA

用户输入回溯窗口的大小 $W$
初始化变量 $i \leftarrow 0$ 作为目前窗口大小
初始化变量 $totalprice \leftarrow 0$ 作为总价格
**while** $i < W$ **do**
    时间回退 1 秒并判断回退后的时间是否在交易时间段
    **if** 若是交易时间 and 该时间点的价格在两个数据流点之间 **then**
        把该时间加入窗口
        $i \leftarrow i + 1$
        $totalprice \leftarrow totalprice+$ 时间点的价格
    **else if** 如果时间点已经早于了最早的数据流时间 **then**
        说明已经回溯到数据开始了，直接跳出循环
    **else** 时间点没遭遇最早的数据流时间，但是处于闭市时间段
        说明回退后的时间进入了闭市时间段，则需要进一步回退，若是 12:59 则回退至 11:30, 若是 9:29 则回退至 15:00
        把时间加入窗口
        $i \leftarrow i + 1$
        $totalprice \leftarrow totalprice+$ 时间点的价格
    **end if**
**end while**
$SMA = totalprice/i$
如果完整的回溯了整个回溯窗口，那么就是正常的 SMA，如果回溯到一半发现已经到最早的数据流了，那么就是用窗口的一部分算得的 SMA

这样一来，每个数据流上的点都能通过这些算法算出 SMA，而两个数据流上的点之间的那些点 (这些点的价格都是继承了第一个数据流点的价格)，我们不建立窗口，也不计算 SMA。

这个算法的核心思想在于，在我们建立回溯窗口的时候，其实已经排除掉了所有非交易时段，而且人为的按照题目的提示，把两个数据流上的点之间的价格” 补齐” 了，这使得我们可以直接通过传统的 K 线 SMA 算法来完成 SMA 的计算。

# 3 回溯窗口空间复杂度分析

由于我们实际上并没有把整个窗口作为一个 list 保存在内存当中等待回溯完成后销毁，而只是维护了两个变量来对这个回溯进行变相的" 求和"，这里的空间复杂度其实只有 O(1)。

# 4 num_bin 的内存限制

在数据流不断产生数据的过程当中，慢慢的产生的数据会堆积从而超过 num_bin，我采取的方式是桶的非等距采样，主观上认为数据流产生的远期数据重要性不如近期的，所以保留更多的近期数据 (这只在数据超过了 num_bin 的内存限制的时候才执行这个优化，不然不执行)

原理如下: 由于回溯窗口的大小是固定的，所以远期的数据基本上不会被回溯到 (回溯到的意思是需要这个数据取补两个数据流点中间缺失的数据)，这样一来的话把他们舍掉也不会对 SMA 的精度有太大影响了。但是不能砍的太多，因为如果这样的话，有很多补充的点可能被夹在同一对数据流点之间，补充点的价格都被这对数据流点的第一个点的价格赋值了，显然这是不好的。

# 5 调试的打印

代码中有调试的部分，当计算 SMA 的时候会打印 output.csv 来显示 SMA 的明细，这只是一个调试信息而已，关掉了也能算 SMA，实际并不占用内存。调试信息显示的其实就是从每个数据流点出发计算的回溯窗口。

# 6 代码的运行

这部分请详见 README.md