程序化交易系统中线性回归模型及其优化算法①

杨光豹

(浙江广播电视大学 青田学院, 青田 323900)

摘 要: 在程序化交易中, 运用交易模型, 通过对历史数据的运算预测价格的未来趋势, 从而确定交易策略. 提出一种线性回归交易模型, 它从模型的原理分析、算法实现以及运行效率等方面进行了详细阐述, 从而得出一种能够满足实时处理大量历史数据, 时间复杂度最小化的交易模型的优化算法.

关键词: 程序化交易; 线性回归; 时间复杂度; 实时计算

Linear Regression and its Optimization Algorithm in Program-Trading System

YANG Guang-Bao

(School of Qingtian, Zhejiang Radio & Television University, Qingtian 323900, China)

Abstract: In program trading, trading strategy is determined by forecasting the tendency of prices in the future, based on calculating historical data according to the trading model. This paper proposes a linear regression trading model. It discusses the principle of the model, algorithm implementation and efficiency in running. Finally, an optimal algorithm for linear regression in Program-trading system is obtained which is competent in a real-time calculation for plenty of historical data. The algorithm's time complexity is least.

Key words: program trading; linear regression; time complexity; real-time calculation

随着计算机技术在现代金融领域的广泛应用, 金融业随即迎来了革命性的改变. 其中, 典型应用就是程序化交易[1]. 程序化交易是指依据统计学原理, 利用计算机对市场历史数据的运算获得一个交易模型, 并依据该模型对最新的行情进行实时计算, 生成交易决策并自动提交交易指令的交易方式. 程序化交易在国内的研究多数停留在概述与意义等理论上, 如: 文献[2], 文献[3]等, 它们只是从宏观面上阐述程序化交易的概念与现实意义等. 少数文章对程序化交易的模型及其实现进行研究, 如文献[4], 文献[5], 它们着重研究程序化交易中的模型设计, 很少考虑到模型实现时的运行效率与数据运算的实时性要求. 本文依据波浪理论与数理统计学中的线性回归理论, 对程序化交易模型进行设计与算法实现, 着重考虑决策运算的实时性, 并对运算效率与程序性能进行优化.

由于程序化交易需要对大量的历史数据进行运算, 并依此预测市场未来的趋势, 这时模型的算法显得极其重要. 目前国内流行的一些金融类的软件(如通达信, 同花顺等)对历史数据的指标运算仅仅是做到了"结果正确", 它们存在大量的重复计算, 性能很低. 比如在"通达信"交易软件中编写一个自定义指标, 用它统计主板所有股票的均线值之和, 然后将该指标显示在副窗口中, 结果软件反应速度大大减慢, 甚至经常出现"停止"反应的现象. 在"飞狐交易师"软件以及"同花顺"等金融类软件中同样出现这样的现象, 究其原因是这类软件的自定义指标在每次有新的价格到来时都要对所有历史数据进行重新计算以确定每一个历史数据所对应的指标值. 它会造成大量的重复计算, 从而使计算机对行情反应不及时. 国外一些先进的金融类软件(比如 Meta Trader 4 等)因为程序化交易的实时要求, 已经考虑到了这一问题, 对历史数据的重复计算已经做了粗粒度的优化, 避免了对已经计算过的历史数据进行重复计算, 它在指标加载时分为初始化阶段与实时计算阶段, 在初始化阶段软件要对所有数据进

① 收稿时间:2014-03-26;收到修改稿时间:2014-04-29

© 中国科学院软件研究所 http://www.c-s-a.org.cn

行计算, 但在初始化完成之后的实时计算阶段时, 每当有一新的行情价格到来时, 它只对最近新到的行情价格进行指标值计算, 而对历史数据的指标值则沿用原来的计算结果值, 从而减少了计算量. 但它的模型指标算法依然还存在可优化的, 大量的细粒度的重复计算. 本文将线性回归分析法[6]应用在程序化交易模型中, 并对线性回归线指标模型的算法进行优化, 做到了即在粗粒度上, 又在细粒度上避免各种重复计算, 大大减少了金融类软件在实时处理市场行情数据时的计算量, 使程序化交易的反应速度得以大幅度提高.

1 线性回归模型分析

在金融市场中, 一段时间内金融品价格(股票、期货、外汇等)往往总是围绕着一条线性回归趋势线在两侧波动并沿这趋势线方向发展, 当价格在波动过程中偏离趋势线距离大了就会向趋势线靠拢. 如图 1 所示. 波浪理论[7]认为金融品价格走势都是波浪式发展的, 无论处于什么阶段, 它在一定时段内都是沿各自回归趋势线两侧波动, 并沿趋势线方向发展.

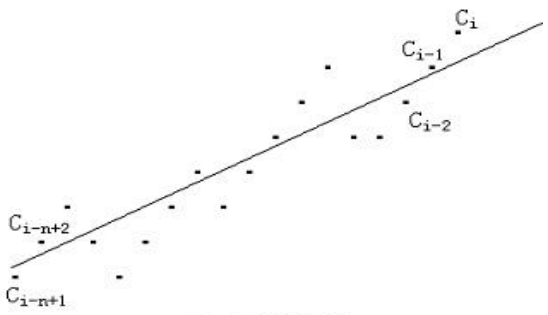

图 1 回归趋势图

趋势线的数学模型:

设该趋势线的直线方程为 y=k*x+b, 其中 k 为直线的斜率, 也称斜率回归系数, 它反应价格上涨或下跌的力度, b 为直线的截距, 也称常数项回归系数. 我们将坐标原点(0,0)设定在当前周期, 所以 b 也是当前回归线的中值, 是当前价格波动的中枢位置. 图中 C_i C_{i-1} C_{i-2} ……C_{i-n+2} C_{i-n+1} 为 n 个时间周期内的各时间点的价格, 其中 Ci 为当前周期的价格, 历史价格所在的周期均为负方向. 依据线性回归理论可得以下两个结论: 1, 各个周期的实际价格与趋势线的差的平方和必定取最小值. 2, 各个周期的价格与趋势线的差的总和必定为 0. 由结论 1 得:

$$Si=\sum_{j=0}^{n-1}\{c_{i-j}-[k_i*(-j)+b_i]\}^2=\sum_{j=0}^{n-1}(c_{i-j}-b_i+j*k_i)^2$$

$$=\sum_{j=0}^{n-1}[j^2*k_i^2+2*j*(c_{i-j}-b_i)*k_i+(c_{i-j}-b_i)^2]$$

$$=k_i^2*\sum_{j=0}^{n-1}j^2+k_i*\sum_{j=0}^{n-1}[2*j*(c_{i-j}-b_i)]+\sum_{j=0}^{n-1}(c_{i-j}-b_i)^2$$

两边对k_i取导数可得

$$k_i=\frac{-\sum_{j=0}^{n-1}[2*j*(c_{i-j}-b_i)]}{2*\sum_{j=0}^{n-1}j^2}=\frac{\frac{n*(n-1)}{2}*b_i-\sum_{j=0}^{n-1}j*c_{i-j}}{(n-1)*n*(2*n-1)/6}$$

$$\sum_{j=0}^{n-1}\{c_{i-j}-[k_i*(-j)+b_i]\}=\sum_{j=0}^{n-1}c_{i-j}+k_i*\sum_{j=0}^{n-1}j-\sum_{j=0}^{n-1}b_i=0$$

由结论2可得

$$\sum_{j=0}^{n-1}c_{i-j}+\frac{k_i*n*(n-1)}{2}-n*b_i=0$$

$$b_i=(\sum_{j=0}^{n-1}c_{i-j}+\frac{k_i*n*(n-1)}{2})*\frac{1}{n}=\frac{1}{n}*\sum_{j=0}^{n-1}c_{i-j}+\frac{k_i*(n-1)}{2}$$

$$k_i=(\frac{n-1}{2}*\sum_{j=0}^{n-1}c_{i-j}-\sum_{j=0}^{n-1}j*c_{i-j})/[\frac{(n-1)*n*(2*n-1)}{6}-\frac{n*(n-1)^2}{4}]$$

将b_i代入k_i解得

至此, 回归趋势线的回归系数k指标与b指标都已经确定, 接下来就可以利用计算机程序算法进行实现.

2 线性回归模型的算法.

设定金融品价格序列的历史周期总数为 m, 价格数组序列从最早周期到当前周期依次为: price[0]、price[1].……price[m-1]. 回归趋势线统计的周期数为 n, 则 m 必须大于 n, 回归线指标从第 n 个周期 price[n-1] 处开始往后计算. 算法利用 MQL5 编程语言, 在 MT5(Meta Trader 5)交易系统中进行调试运行

(1)算法一(传统循环算法).

按传统循环算法, 需要设置两层嵌套循环进行计算, 这样以来, 计算所有 k 指标需要的循环次数是: 2*(m-n+1)*n+ (m-n+1). 计算 b 指标循环次数是: m-n+1. 总的计算次数是: 2*(m-n+1)*(n+1), 代码如图 2 所示.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

```
for(int i=n-1;i<rates_total;i++){
     double temp1=0,temp2=0;
     for(int j=0;j<n;j++){
       temp1=temp1+price[i-j];
       temp2=temp2+j*price[i-j];
      }
     kBuffer[i]=((n-1)*temp1/2-temp2)/temp;
     bBuffer[i]= temp1/n+kBuffer[i]*(n-1)/2;
  }
```

图 2　算法一

代码含意说明:

temp 存储$\frac{(n-1)*n*(2*n-1)}{6}-\frac{n*(n-1)^2}{4}$值.

temp1,temp2 分别存储, $\sum_{j=0}^{n-1} c_{i-j}$, $\sum_{j=0}^{n-1} j*c_{i-j}$

kBuffer[]存储指标 Ki 的值, bBuffer[]存储 bi 值.

该算法时间复杂度为 O(m*n), 它存在一个严重缺点: 大量的重复计算. 在当前周期一个新的价格到来后, 它将会对所有历史周期的回归线指标都重新计算一次. 这样就可能出现程序化交易系统由于来不及计算只能将最新的价格放到缓冲中等待, 延误了最佳的交易决策时机. 严重时将在市场价格频繁变动时程序被堵塞而停止反应甚至系统死机.

(2)算法二(粗粒度优化算法).

在 MT4(Meta Trader 4)金融交易平台[8]中普遍采用的一种粗粒度的优化方法是: 对已经计算过的历史周期指标不进行计算, 因为它已经成为了历史就不会再发生改变(需要时只从缓冲中取来便可). 它只对最新的可能改变的指标进行计算. 代码如图 3 所示:

```
int since;
if(prev_calculated>0)since=prev_calculated-1;
  else since=n-1;
for(int i=since;i<rates_total;i++){
  double temp1=0,temp2=0;
  for(int j=0;j<n;j++){
    temp1=temp1+price[i-j];
    temp2=temp2+j*price[i-j];
   }
   kBuffer[i]=((n-1)*temp1/2-temp2)/temp;
   bBuffer[i]= temp1/n+kBuffer[i]*(n-1)/2;
  }
```

图 3　算法二

代码含意说明:

prev_calculated 存储已计算过的历史数据计数器,

since 表示最近一次计算过的历史数据位置

temp,temp1,temp2,kBuffer[],bBuffer[]同上.

它在加载一个指标时首先是初始化指标, 在初始化过程, 由于所有的历史周期指标都还没计算过, 这一过程要计算所有历史数据所对应的指标, 程序循环标志 since 从 0 开始. 它需用双层嵌套循环来完成, 与“算法一”计算次数也相同. 当完成初始化后, 每当有新的价格到来时, 它开始对行情指标进行实时计算, 这时 prev_calculated 值已经记录了所有已经计算过的历史数据的数量, since 将从最近一次历史数量位置开始计算. 这一过程只对当前周期最新的价格进行指标计算, 这时计算只在循环内部进行一次, 不再有多次外循环, 所以总的指标循环次数是: 2*(n+1). 比原来的计算量大大地减少了. 该算法时间复杂度在指标初始化时为 O(m*n), 初始化之后的实时计算时为 O(n). 它的优点是在实时计算时避免了大量的对历史数据的重复计算, 使算法复杂度从 O(m*n)减到 O(n), 从而大大缩短程序在实时计算时的响应时间, 提高反应速度.

(3)算法三(最佳优化算法).

虽然“算法二”已经在实时计算时从粗粒度上对重复计算进行了优化, 避免了大量的历史数据指标值的重复计算. 但它在 n 值比较大时依然存在大量的细粒度的重复计算, 更何况在初始化时也没有进行任何优化措施, 初始化指标过程极慢. 为此本文提出一种最佳优化算法(算法三). 它在程序中无论是粗粒度的重复计算, 还是细粒度的重复计算都要避免, 计算量比“算法二”要少 n 个数量级. 其原理如下:

根据线性回归趋势线模型可得:

$$k_i=(\frac{n-1}{2}*\sum_{j=0}^{n-1}c_{i-j}-\sum_{j=0}^{n-1}j*c_{i-j})/[\sum_{j=0}^{n-1}j^2-\frac{n*(n-1)^2}{4}]$$

$$k_{i-1}=(\frac{n-1}{2}*\sum_{j=0}^{n-1}c_{i-1-j}-\sum_{j=0}^{n-1}j*c_{i-1-j})/[\sum_{j=0}^{n-1}j^2-\frac{n*(n-1)^2}{4}]$$

$$k_i-k_{i-1}=[\frac{n-1}{2}*(\sum_{j=0}^{n-1}c_{i-j}-\sum_{j=0}^{n-1}c_{i-1-j})-(\sum_{j=0}^{n-1}j*c_{i-j}-\sum_{j=0}^{n-1}j*c_{i-1-j})]$$
$$/[\sum_{j=0}^{n-1}j^2-\frac{n*(n-1)^2}{4}]$$

$$k_i=k_{i-1}+[\frac{n-1}{2}*(c_i-c_{i-n})-(\sum_{j=0}^{n-1}c_{i-1-j}-n*c_{i-n})]$$
$$/[\frac{(n-1)*n*(2*n-1)}{6}-\frac{n*(n-1)^2}{4}]$$

其代码如图 4 所示.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

```
int since;
if(prev_calculated>0)since=prev_calculated-1;
  else since=n-1;
for(int i=since;i<rates_total;i++){
  if(i==n-1){
     double temp1=0,temp2=0;
     for(int j=0;j<n;j++){
       temp1=temp1+price[i-j];
       temp2=temp2+j*price[i-j];
      }
     tempBuffer[i]=temp1;
     kBuffer[i]=((n-1)*temp1/2-temp2)/temp;
     bBuffer[i]= temp1/n+kBuffer[i]*(n-1)/2;
  }
  else{
     tempBuffer[i]= tempBuffer[i-1]+price[i]-price[i-n];
     kBuffer[i]=kBuffer[i-1]+((n-1)*(price[i]-price[i-n])/2
               -(tempBuffer[i-1]-n*price[i-n]))/temp;
     bBuffer[i]= tempBuffer[i]/n+kBuffer[i]*(n-1)/2;
  }
  }
```

图 4 算法三

代码含意说明:

tempBuffer[]存储上式中 $\sum_{j=0}^{n-1} c_{i-1-j}$ 的值

since,temp,temp1,temp2,kBuffer[],bBuffer[]同上.

程序在第一次计算 tempBuffer[i]时需要进入内循环计算, 一旦有了第一个值后, 接下来它就只需要通过前一值与变动的价格值来一次性计算得到, 无需进入内循环. ki, 第一次计算出来之后, 下一次计算就可以通过 ki-1 与 tempBuffer[i-1], 以及相关价格值直接计算得到, 不用再通过循环来计算了. 该算法通过增加一个数组变量 tempBuffer[]临时存放中间计算结果, 从而使程序的时间复杂度降低, 将计算次数从原来的 2*(m-n+1)*(n+1)减少到 3*(m-n)+(n+3).

“算法三”是一种最佳优算法, 它在指标初始化时的时间复杂度是 O(m), 在初始化之后, 当一个新的价格到达进行实时计算时, 它的时间复杂度为 O(1). 它相比“算法二”在时间复杂度上降低了 n 个数量级, 从而能够大大提高程序在初始化指标与实时计算时的性能.

3 线性回归模型的算法效率测试.

在 MT5 交易平台中我们选择模型的历史数据分别为 H1、M15、M5 三种历史数据(m 值分别为: 92282 368310 1097576). 回归统计周期 n 分别设定为 10, 100, 1000, 1000000. 对三种算法在初始化时与实时计算时分别进行测试, 测试的内容是在以上设定情况下单独运行三种算法获得模型指标结果所需要的时间(毫秒). 在算法的开始与结束处分别写上语句 start=GetTickCount();end=GetTickCount();然后输出它们的差就是算法的运算时间. 其结果如图 5,图 6 所示. 在测试结果表中, 运行时间为 0 表示在毫秒级以下就完成运算了, 所以可以忽略它的运行时间, 认为做到了实时运算, 而∞表示程序运行时间过大, 短时间无法完成运算.

m	92282			368310			1097576			
n	10	100	1000	10	100	1000	10	100	1000	1000000
算法一	15	125	1328	78	531	4969	203	1703	14609	∞
算法二	32	187	1985	109	750	7359	296	2281	22140	∞
算法三	0	0	0	31	31	31	110	109	110	31

图 5 初始化过程测试结果

m	92282			368310			1097576			
n	10	100	1000	10	100	1000	10	100	1000	1000000
算法一	16	141	1250	63	516	5125	204	1547	15234	∞
算法二	0	0	0	0	0	0	0	0	0	16
算法三	0	0	0	0	0	0	0	0	0	0

图 6 实时运算过程测试结果

在初始化过程中, “算法一”与“算法二”由于时间复杂度一样, 所以运算所需时间在同一个数量级别上, 相差不大. 运算所需时间主要受 m*n 的值决定. 而“算法三”性能明显比前两种算法要优越得多, 它不受 n 值的影响. 尤其是在 m*n 值很大时, “算法一”与“算法二”简直无法容忍. 可是“算法三”却能轻松完成.

在初始化之后的实时运算中, “算法一”明显要差, 它根本无法用在程序化交易的实时运算中, 而“算法二”基本上都能在毫秒级以下完成运算(特例情况除外), 完全能胜任程序化交易要求的实时运算. 为了区别“算法二”与“算法三”的性能, 特另加一组条件(m=1097576,n=1000000)进行测试, 结果表明“算法二”不能胜任, 而“算法三”依然轻松进行实时计算. 因为“算法二”的时间复杂度是 O(n), 它会随着 n 值的增大而增大, 而“算法三”的时间复杂度是 O(1), 它不受 n 值影响. 明显“算法三”是最优的. 总之, 经过测试三种算法中“算法三”无论是在初始化过程还是在实时计算过程都是最优的算法, 它能大大缩短程序化交易指标计算的时间.

4 结语

本文通过对线性回归线的数学模型分析, 在 Meta Trader 5 系统中设计出一种最优算法的线性回归模型并用 MQL5 编程语言实现, 它以最大程度减少循环运

© 中国科学院软件研究所 http://www.c-s-a.org.cn

算中的重复计算, 从而缩短程序化交易的模型运算所需的时间, 极大提高程序的反应速度. 使软件在程序化交易中, 在海量历史数据面前照样做到实时运算, 实时决策, 快速下单.

参考文献

1 熊熊,张博洋,张永杰,付琳惠.程序化交易系统的检测与优化体系.科学决策,2013,8:1–15.

2 陈梦根.算法交易的兴起及最新研究进展.证券市场导报,2013,9:11–16.

3 李凤雨.高频交易对证券市场的影响及监管对策.上海金融,2012,9:48–52.

4 张戈,程棵,陆凤彬,汪寿阳.基于 Copula 函数的程序化交易策略.系统工程理论与实践,2011,4:599–605.

5 包思,郑伟安,周瑜.基于 MACD 的平稳技术指标在高频交易中的应用.华东师范大学学报(自然科学版),2013,9:152–160.

6 任丽丽,陆秋君.基于模糊线性回归的电子商务交易额预测.统计与决策,2013,3:31–33.

7 李媛媛.基于波浪理论的证券市场投资周期性研究.商业时代,2013,5:59–60.

8 Blackledge J, Murphy K. Forex trading using meta trader 4 with the fractal market hypothesis. Dublin Institute of Technology. 2011

www.c-s-a.org.cn

© 中国科学院软件研究所 http://www.c-s-a.org.cn