基于数据几何特征的期货价格交互作用规律研究

摘要

随着全球钢铁工业的不断发展, 黑色工业商品如煤炭铁矿石等期货交易市场不断扩大, 由于其价格多变, 吸引诸多国内外投资者进行买卖. 不同期货价格变化不尽相同又存在一定内在关联, 探究多期货价格变化的相关关系对发展经济与进行投资有着深远意义. 本文选取螺纹钢、铁矿石、不锈钢、热轧卷板、硅铁、焦煤、焦炭、锰硅、线材 9 种期货, 对其价格变动进行分析并建立了相关数学模型.

本文从期货交易所公开数据库中下载了目标种类期货的每个交易日价格数据, 对数据预处理后, 选取 9 种期货自 $2020-2022$ 三年间主力合约的日收盘价作为价格分析对象进行分析.

从机器学习的热门方向之一流形学习的观点来看, 在此问题中使用的数据集其实是高维空间中的一个低维流形. 由正则值原像定理, 一个性质较好的光滑函数可以确定一个子流形. 而多项式的零点而作为其中一个重要类别, 满足较好的性质, 易于计算. 在这个问题具体求解过程中将考虑使用一个二次多项式所确定的超曲面来拟合数据集, 并给出相应的投资策略.

针对问题一, 本文先对两两种类进行简单线性回归分析, 而后计算其 **Pearson 相关系数** 以初步判定其线性相关程度. 随后对各期货价格时间序列进行 **ADF 检验**, 各项时间序列一阶差分项通过检验, 得出结论各价格时间序列有长期稳定的均衡关系. 接下来对多变量进行**因子分析**, 通过因子检验得到多变量间有较好线性关系. 最后对于所有期货价格变量进行 **多元线性回归分析**, 计算得出相关系数为 $R^2 = 0.988$, 可以得出 9 种期货价格有较好的线性相关关系.

针对问题二, 本文使用多项式进行非线性关系分析, 首先在回归分析中引入价格二次项分析螺纹钢价格与其余期货价格一、二次项之间关系, 得到相关系数 $R^2 = 0.991$, 表明该种非线性关系较好贴合实际数据. 进一步联系线性代数与高维流形等相关数学知识对模型进行优化, 建立了形式简单的 **二次型非线性模型** $x^TAx = 1$, 运用仿射变换对期货价格进行 **标准化** 后, 带入标准化价格计算得出九阶对称阵 A 的具体取值.

针对问题三, 本问题中使用二次型非线性关系作为被讨论对象, 定量计算二次型模型 **预测值与真实值之间差值**, 以该差值来刻画关系准确程度, 并给出临界值 0.1 作为定量判据. 当差值不大于 0.1 时认为不同期货价格处于二次型关系窗口期, 反之则不处于窗口期. 进一步地, 对于二次曲面拟合较好或不好数据点渲染不同颜色分为两类, 使用 **数据可视化手段** 将高维空间中数据点投影到二维平面, 使用机器学习 **支持向量机** 分析二维平面两类数据点分布规律, 可给出大致判定窗口区判据. 而后采用流形学习中经典方法 **局部线性嵌入算 法** 对窗口期判断进行了进一步讨论.

针对问题四, 本问题仍沿用问题二中建立的 **二次型非线性模型**, 并假设标准化价格数据点在高维空间中大致沿二次型多项式 **梯度方向** 趋近曲面. 在给定步长下, 使用步长乘以价格变化向量以预测标准化价格向量的变化, 根据标准化方式, 进行逆向运算可以得到实际预测价格变化与变化比率. 依据该预测结果, 制定基本策略为买入价格变化增长率最大期货, 出售预测价格下跌率最大期货. 进一步具体给了不同投资策略以及投资回报率, 在本文所给策略中最高投资回报率 161.27%.

一、 问题重述

1.1 问题背景

自千禧年以来, 全球特别是亚洲地区钢铁产业发展迅速, 带动世界范围内矿石煤炭等消费量大幅度提高, 极大推动了黑色产业的发展. 自 2002 至 2011 年, 铁矿石消费年均增长率高达 8.49%, 在 2003 – 2007 年间, 年增长率更是超过 10%.

如铁矿石煤炭等褐色产业中大宗商品的价格随政策环境气候等变化迅速, 使其成为期货期权交易热门对象. 通过观察产品价格变化不难发现不同期货商品价格有着一定的相关性, 例如某产品与其主要原材料的价格大致呈正相关性. 比较不同产品的价格变动进而分析它们之间的关系, 对此建立模型并对于未来价格走向进行分析, 有助于了解市场且使投资者制定更为全面的投资计划.

期货指在某个未来时间点以约定价格买入或卖出某种资产（如股票、大宗商品、货币等）的合约. 例如, 某人买入一份黄金期货合约, 待到约定日期, 若黄金价格上涨, 他就可以以低于市场价的价格买入黄金并卖出期货合约；若价格下跌了, 他仍须按照合约以高于市场价的价格买入黄金. 黑色系期货是指以铁煤焦钢为代表的一类商品期货，这些期货在国内外市场都有较高的流动性和影响力，它们的价格会受供需、市场等等方面因素的影响. 同时, 黑色系期货之间也存在着紧密的上下游关联性，形成了一个相对完整的产业链.

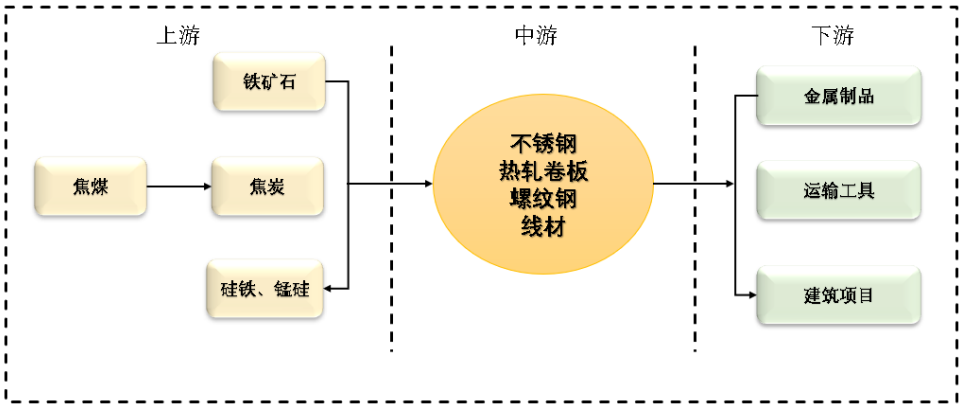

图 **1**　铁煤焦钢相关产业链划分

1.2 问题提出

为研究不同期货间价格变化关系, 特选取九种不同的工业期货产品, 分别为：螺纹钢、铁矿石、不锈钢、热轧卷板、硅铁、焦煤、焦炭、锰硅、线材. 各地交易网站提供了各商品若干年日线期货交易最大额、最小额,、开盘价、收盘价、持仓量等数据, 通过对各期货日线数据进行分析, 建立数学模型研究以下问题：

1. 分析计算每日各商品价格变动, 对于不同商品间价格建立线性关系, 并对于建立的线性关系进一步建模.
2. 除线性关系外, 考虑各产品之间可否建立非线性关系, 根据已有数据比较分析两种关系的优劣.
3. 各产品价格随时间气候等因素可能产生较大变动, 故建立的关系具有一定时效性, 尝试找出判据以判定关系拟合较好的时间段.
4. 通过分析出的关系, 试建立具体投资策略, 使得在已建立模型预测下投资回报率最优.

二、 问题分析

2.1 问题一分析

针对问题一, 题目要求从价格数据中找到不同品种商品间的线性关系, 可以使用统计学中的相关性分析. 对于不同商品日线价格建立线性回归方程, 用最小二乘法确定拟合方程的具体参数数值, 进而对回归方程计算相关系数来判断是否有较好线性相关性.

具体而言, 可以先考虑简单线性回归分析, 即只选取两种商品价格分别作为自变量与因变量. 选取所有的两两组合, 分别对每一种组合建立一元线性回归方程进行线性拟合, 而后计算每种组合的 Pearson 相关系数, 根据相关系数结果选取一定范围以判断不同商品日线价格间是否有较好的线性相关关系.

对于多个变量线性关系分析, 可先进行检验, 以判断多变量之间是否有较好的线性关系. 若通过该检验可进一步进行多元线性回归分析, 将某种商品价格作为因变量, 其余全部引入为自变量, 进行线性拟合并分析其线性相关性质. 若多元线性回归的相关系数接近 1, 则可以认为九种商品价格间有较好的线性性质. 同时可对价格时间序列进行简要平稳性检测, 以判定其在长期内变化是否均衡.

2.2 问题二分析

Weierstrass 逼近定理指出, 任何连续函数都可以用多项式函数列一致逼近. 这个定理在实际问题中有很广泛的应用，比如信号处理、图像处理、机器学习等领域. 针对问题二，我们可以借鉴该定理，对于一定时间范围内的非线性数据，可以考虑使用多项式进行拟合。为了找到更加精确的非线性关系，在本问题中可以使用多项式对各品种价格进行回归分析。

具体分析方法有如下两种：一种为将某种期货价格作为因变量, 其余商品价格幂次项作为自变量进行多元多项式拟合；另外一种为考虑所有期货价格幂次项之间相互关系, 可以选择常数项作为因变量, 找到一系列参数使得个价格幂次项相互组合后近似恒等于某常数, 在此方法下可以运用代数中二次型相关知识进行计算.

2.3 问题三分析

针对问题三, 关系窗口期是指实际数据与已建立模型中关系符合较好的时期, 即模型预测较为准确的时期. 在这一问题下将对第二问中二次型关系进行分析. 在此考虑使用预测值与实际值的差距来刻画关系的准确度, 可给出差额临界值, 若在某时间段内模型预测数据与该时间段实际数据相差小于临界值, 可以认为在此时间段内各期货价格处于已给出关系的窗口期.

进一步, 利用可视化技术进行图像展示. 可使用不同颜色区分处于与非处于窗口期两类数据点, 并使用可视化手段将高维空间数据点降为二维空间图像展示. 对于二维图像继续使用机器学习区分异类数据点分布规律, 根据分布规律可以给出窗口期大致判据.

2.4 问题四分析

针对问题四, 将沿用第二问中建立的二次型模型对问题进行分析. 若制定想制定回报率最高的投资策略, 则需要先对价格走向进行较为准确的预计. 对于采用的二次型模型, 定值二次型可表示为高维空间中光滑超曲面, 由于梯度代表二次型函数每点处函数值变化最快的方向, 可以假设在数据点大致沿梯度方向逼近模型给出的曲面, 在具体分析中使用二次型函数梯度对价格变化进行估计.

根据得到的预计价格变化, 为获得更高投资回报率, 则基本策略应为买入预测价格增长率最高品种, 售出预计价格下跌率最高品种. 对于具体策略可以进一步细化以求得相对最优策略.

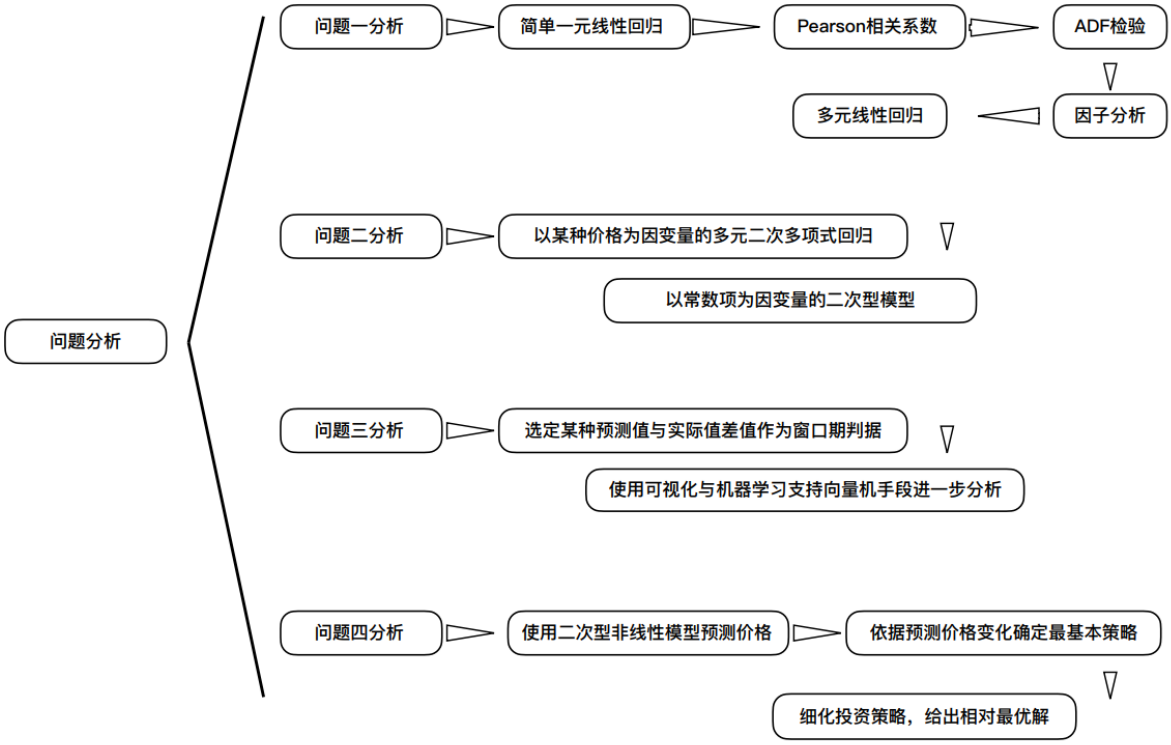

图 **2**　问题分析流程示意图

三、 基本假设与符号说明

- 只考虑价格之间以及价格随时间变化, 忽略其余因素对于文中提到商品价格的影响, 故在模型建立中不引入政策, 气候等因素.
- 由于各大交易所公开数据中并未列出每日日线数据, 而是选取每月部分天数给出数据, 在此假设以下使用的价格数据随时间连续变化且连续变化曲线大致可由散点图线性插值逼近.
- 由于不锈钢期货于 2019 年 9 月首次在全球范围内推出[1], 本文假设自 2020 年起不锈钢期货市场大致成熟稳定, 可以全面分析不锈钢期货与其他期货间相互关系.
- 在进行投资策略分析时假设期货交易时间限制足够长, 即在本文所分析的时间范围内各类期货可自由交易.
- 假定本文涉及所有期货交易忽略手续费并且均采用 1 倍杠杆, 交易市场持仓量高于每一次交易量.

符号	符号说明
β_i	第 i 个品种的回归系数
p_i	第 i 个品种的实际价格
p_i^*	第 i 种品种的标准化价格
A	二次型模型系数矩阵
Δ	预测值和实际值的差值
$\nabla(x^TAx)$	二次型函数每一点对应的梯度向量
λ_0	预测模型的基准步长
λ	预测模型的变化步长
$p_{\max}^{(i)}$	三年内第 i 种品种实际价格最大值
$p_{\min}^{(i)}$	三年内第 i 种品种实际价格最小值

四、 数据预处理与数据分析

4.1 数据预处理

螺纹钢、热轧卷板、不锈钢、线材日线数据取自上海期货交易所；铁矿石、焦煤、焦炭日线数据取自大连期货交易所；锰硅、硅铁日线数据取自郑州商品交易所. 期货市场每一个期货在不同的时期会有对应的月份合约, 比如 rb2106, 代表的是螺纹钢月份合约的最后期限是 2019 年 6 月.

根据交易所公开数据, 对数据进行如下提取：首先, 同一商品不同时限合约间日交易量差别较大, 部分商品不同时限交易合约交易量有着较为明显的数量级差异, 在以下分析与建模过程中, 选取主力合约, 即日交易量最大且拥有最大持仓量的合约进行分析, 其能较好地代表该期货商品的交易状况；第二, 由于日线最高价最低价等波动较大, 在此选取主力合约的日收盘价作为价格分析对象, 以确保价格有着较好的连续性；第三, 由于不锈钢期货在 2019 年首次上市, 故我们选取 2020 − 2022 三年数据以便可以对所有期货价格进行综合分析.

对下载的数据集进行了整理后, 由此得到了日期-期货-价格的对应关系表. 其中删除了所有的节假日, 并且笔者注意到有些日期存在数据缺失的现象, 为了之后的分析的有效性, 笔者使用线性插值补足缺失的数据.

4.2 价格趋势初步分析

经过数据预处理后, 选取 2020 − 2022 三年内各类商品主力合约日线收盘价 (共 728 组样本数据), 其随时间变化如下图所示：

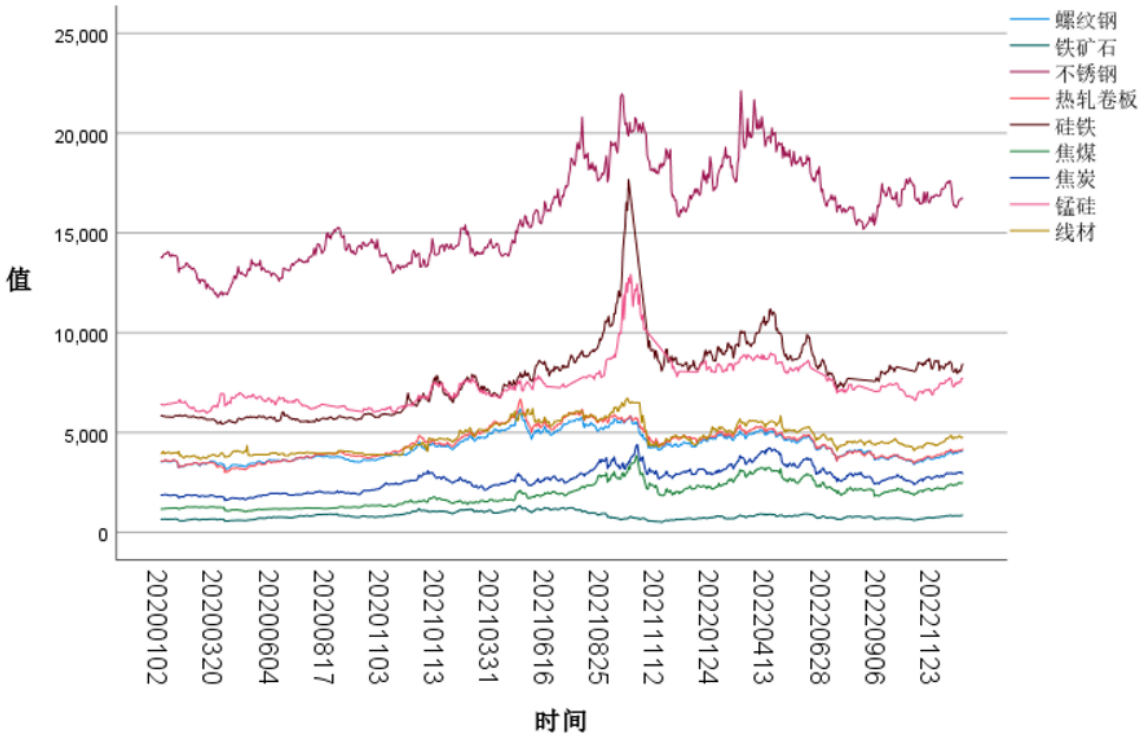

图 **3**　**9** 种期货收盘价随时间变化折线图

通过收盘价的趋势变化, 我们可以初步得出以下结论：

1. 除了铁矿石, 其他黑色系商品期货的价格相关性都比较好, 价格出现最高点的时间节点相近.
2. 螺纹钢、热轧卷板、焦煤、焦炭和线材价格走势相似度很高, 锰硅和硅铁价格走势相似度很高,
3. 硅铁和不锈钢的价格起伏较大, 铁矿石的价格起伏则小得多.

4.3 相关性分析

为确定价格间线性关系, 首先建立简单线性回归模型：

在九种商品中选取两种商品 A, B 并且选取一段时间（共计 n 个交易日）其每日收盘价分别为 x_i, y_i. 对其日收盘价进行线性回归分析, 建立线性回归方程 $\hat{y} = \alpha x + \beta$. 满足约束条件

$$\bar{y} = \alpha\bar{x} + \beta, \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i, \bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{1}$$

并使用最小二乘法：

$$\sum_{i=1}^{n} (\alpha x_i + \beta - y_i)^2_{min} \tag{2}$$

可以求解参数 α, β 具体数值. 对于此类线性回归, 可以通过计算相关系数来判断二者变量间的线性相关程度. 本模型中选取 $Pearson$ 相关系数来刻画变量间相关性. $Pearson$ 相关系数定义为两个变量的协方差除以其标准差的乘积, 具体计算如下：

$$\rho_{xy} = \frac{cov(x, y)}{\sigma_x \sigma_y} = \frac{E[(x - \mu_x)(y - \mu_y)]}{\sigma_x \sigma_y} \tag{3}$$

对于有限离散样本的相关性计算, 可用 r 表示其相关系数, 计算方法如下：

$$r=\frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i-\bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i-\bar{y})^2}} \tag{4}$$

当 $|r|$ 越接近 1 时, 两变量的线性相关程度越高；当 $|r|$ 越接近 0 时, 即两变量相关程度越小. 对于九种不同期货, 对其进行两两组合, 选择其中一种的价格作为因变量, 另一作为自变量, 计算得到的不同期货间 $Pearson$ 线性相关系数. 我们采用 $SPSS\ Statistics$ 软件对九种期货价格两两之间计算 $Pearson$ 系数, 得到结果如下图所示：

		螺纹钢	铁矿石	不锈钢	热轧卷板	硅铁	焦煤	焦炭	锰硅	线材
螺纹钢	皮尔逊相关性	1	.588**	.641**	.989**	.704**	.653**	.686**	.638**	.951**
	显著性（双尾）		<.001	<.001	.000	<.001	<.001	<.001	<.001	.000
	个案数	728	728	728	728	728	728	728	728	728
铁矿石	皮尔逊相关性	.588**	1	-.010	.618**	.024	.017	.139**	-.097**	.484**
	显著性（双尾）	<.001		.782	<.001	.521	.643	<.001	.009	<.001
	个案数	728	728	728	728	728	728	728	728	728
不锈钢	皮尔逊相关性	.641**	-.010	1	.612**	.871**	.920**	.869**	.813**	.713**
	显著性（双尾）	<.001	.782		<.001	<.001	<.001	<.001	<.001	<.001
	个案数	728	728	728	728	728	728	728	728	728
热轧卷板	皮尔逊相关性	.989**	.618**	.612**	1	.663**	.615**	.661**	.594**	.928**
	显著性（双尾）	.000	<.001	<.001		<.001	<.001	<.001	<.001	.000
	个案数	728	728	728	728	728	728	728	728	728
硅铁	皮尔逊相关性	.704**	.024	.871**	.663**	1	.904**	.849**	.935**	.799**
	显著性（双尾）	<.001	.521	<.001	<.001		<.001	<.001	.000	<.001
	个案数	728	728	728	728	728	728	728	728	728
焦煤	皮尔逊相关性	.653**	.017	.920**	.615**	.904**	1	.960**	.827**	.734**
	显著性（双尾）	<.001	.643	<.001	<.001	<.001		.000	<.001	<.001
	个案数	728	728	728	728	728	728	728	728	728
焦炭	皮尔逊相关性	.686**	.139**	.869**	.661**	.849**	.960**	1	.781**	.717**
	显著性（双尾）	<.001	<.001	<.001	<.001	<.001	.000		<.001	<.001
	个案数	728	728	728	728	728	728	728	728	728
锰硅	皮尔逊相关性	.638**	-.097**	.813**	.594**	.935**	.827**	.781**	1	.710**
	显著性（双尾）	<.001	.009	<.001	<.001	.000	<.001	<.001		<.001
	个案数	728	728	728	728	728	728	728	728	728
线材	皮尔逊相关性	.951**	.484**	.713**	.928**	.799**	.734**	.717**	.710**	1
	显著性（双尾）	.000	<.001	<.001	.000	<.001	<.001	<.001	<.001	
	个案数	728	728	728	728	728	728	728	728	728

**. 在 0.01 级别（双尾），相关性显著。

图 **4**　**9** 种期货两两之间 $Pearson$ 系数

一般认为：$|r|\geq 0.8$ 时两变量具有强相关性；$0.8>|r|\geq 0.5$ 时两变量中度相关；$0.5>|r|\geq 0.3$ 时两变量弱相关；若 $0.3>|r|$, 则可以认为两变量几乎无相关性；当 $r=0$ 时, 认为两变量为完全独立变量. 由上表中数据可以得到：螺纹钢价格分别于热轧卷板价格线性相关性最强, 相关系数高达 0.989；铁矿石价格与不锈钢价格相关性最弱, 相关系数仅为 -0.010, 可认为二者几乎为完全独立变量.

4.4 平稳性分析

为了更好的判断变量之间的相互关系, 除了相关性分析, 时间序列平稳性分析也是非常重要的部分, 这是用来分析变量之间是否存在长期的均衡关系的重要指标, 例如有的变量之间可能具有高度的相关性但是时间序列平稳性很差. 在此我们从日线数据出发对螺纹钢、铁矿石、不锈钢、热轧卷板、硅铁、焦煤、焦炭、锰硅、线材这 9 种期货之间进行 ADF 检验. ADF 检验目的在于检测时间序列是否存在单位根, 若时间序列存在

单位根, 则长期来看它是不平稳的, 将一直受到扰动项影响无法回归长期均值, 若时间序列不存在单位根, 则可认为其是平稳的.

使用 $SPSSPRO$ 网站单位根检验功能对以上 9 种期货价格进行 ADF 分析, 最终得到数据如下表所示：

品种	差分阶数	t-value	p-value	AIC	1%	5%	10%
螺纹钢	0	-1.728	0.417	8170.67	-3.439	-2.866	-2.569
	1	-27.464	0.000***	8162.201	-3.439	-2.866	-2.569
	2	-10.54	0.000***	8184.204	-3.44	-2.866	-2.569
铁矿石	0	-2.108	0.241	6614.456	-3.439	-2.866	-2.569
	1	-11.116	0.000***	6608.738	-3.439	-2.866	-2.569
	2	-10.439	0.000***	6645.363	-3.44	-2.866	-2.569
不锈钢	0	-1.683	0.44	10172.001	-3.439	-2.866	-2.569
	1	-17.727	0.000***	10159.669	-3.439	-2.866	-2.569
	2	-11.086	0.000***	10182.019	-3.44	-2.866	-2.569
热轧卷板	0	-1.539	0.514	8170.223	-3.439	-2.866	-2.569
	1	-26.377	0.000***	8161.532	-3.439	-2.866	-2.569
	2	-10.715	0.000***	8183.656	-3.44	-2.866	-2.569
硅铁	0	-4.155	0.001***	11902.873	-3.44	-2.866	-2.569
	1	-9.36	0.000***	11877.471	-3.44	-2.866	-2.569
	2	-10.562	0.000***	11929.549	-3.44	-2.866	-2.569
焦煤	0	-1.654	0.455	8045.68	-3.439	-2.866	-2.569
	1	-27.416	0.000***	8037.489	-3.439	-2.866	-2.569
	2	-10.128	0.000***	8063.873	-3.44	-2.866	-2.569
焦炭	0	-2	0.286	8229.177	-3.439	-2.866	-2.569
	1	-7.038	0.000***	8221.737	-3.44	-2.866	-2.569
	2	-10.058	0.000***	8254.206	-3.44	-2.866	-2.569
锰硅	0	-4.128	0.001***	12047.958	-3.44	-2.866	-2.569
	1	-10.343	0.000***	12046.252	-3.44	-2.866	-2.569
	2	-11.561	0.000***	12091.441	-3.44	-2.866	-2.569
线材	0	-1.963	0.303	8498.523	-3.44	-2.866	-2.569
	1	-7.243	0.000***	8489.42	-3.44	-2.866	-2.569
	2	-11.108	0.000***	8517.005	-3.44	-2.866	-2.569

注：***、**、*分别代表1%、5%、10%的显著性水平

图 **5**　**9** 种期货的 ADF 检验表

图表数据解读如下：$1\%, 5\%, 10\%$ 分别对应置信 $99\%, 95\%, 90\%$ 的临界值, 若计算得出的 $t-value$ 不大于 1% 下临界值, 则说明在 99% 置信下该期货价格时间序列是平稳的. 根据上表中数据, 可得到在 99% 或 95% 置信下仅有锰硅, 硅铁价格变化为平稳的时间序列. 若对各期货价格时间序列做一阶差分后分析, 所有期货的 $t-value$ 统计值均小于 1% 的临界值, 说明它们在 99% 置信下均不存在单位根, 可以得出结论：它们的价格序列具有长期稳定的均衡关系. 这为后面问题 4 的投资组合策略提供了理论依据.

五、 问题一模型建立与求解

5.1 多变量相关矩阵

为分析多个变量间是否存在较好线性关系, 首先使用因子分析进行初步检验. 具体检验过程如下：上分析中我们计算了 $Pearson$ 相关系数, 可进一步由 $R_{ij}^2 = \rho_{x_i y_j}^2$ 得到不同样本间的相关矩阵

$$R=\begin{pmatrix} R_{11}^2 & R_{12}^2 & \cdots & R_{1n}^2 \\ R_{21}^2 & R_{22}^2 & \cdots & R_{2n}^2 \\ \vdots & \vdots & \ddots & \vdots \\ R_{m1}^2 & R_{m2}^2 & \cdots & R_{mn}^2 \end{pmatrix}=\left(R_{ij}^2\right) \tag{5}$$

根据该矩阵进而对多变量进行因子分析, 结果在同一因子内的期货价格有较好的线性相关性质.

5.2 KMO 值计算

采用因子旋转方法使因子更有意义, 根据 KMO 公式：

$$KMO=\frac{\sum\sum_{i\neq j} r_{ij}^2}{\sum\sum_{i\neq j} r_{ij}^2 \sum\sum_{i\neq j} \alpha_{ij\bullet 1,2\cdots k}^2} \tag{6}$$

其中 r_{ij} 和 $\alpha_{ij\bullet 1,2\cdots k}^2$ 是一些相关系数[2], 计算得到 KMO 值为 0.819, 0.819 在 0.8 以上, 这说明数据非常适合因子分析; 另一方面, “公因子方差” 均大于 0.8, 即所有变量的共同度都在 80% 以上, 说明所提取的因子对各变量的解释能力很好; 根据碎石图, 我们发现螺纹钢和铁矿石在曲线中处在较陡的位置, 说明这两个是最佳的因子.

KMO 和巴特利特检验

KMO 取样适切性量数。		.819
巴特利特球形度检验	近似卡方	12908.664
	自由度	36
	显著性	.000

公因子方差

	初始	提取
螺纹钢	1.000	.965
铁矿石	1.000	.911
不锈钢	1.000	.886
热轧卷板	1.000	.956
硅铁	1.000	.934
焦煤	1.000	.930
焦炭	1.000	.860
锰硅	1.000	.870
线材	1.000	.935

提取方法：主成分分析法。

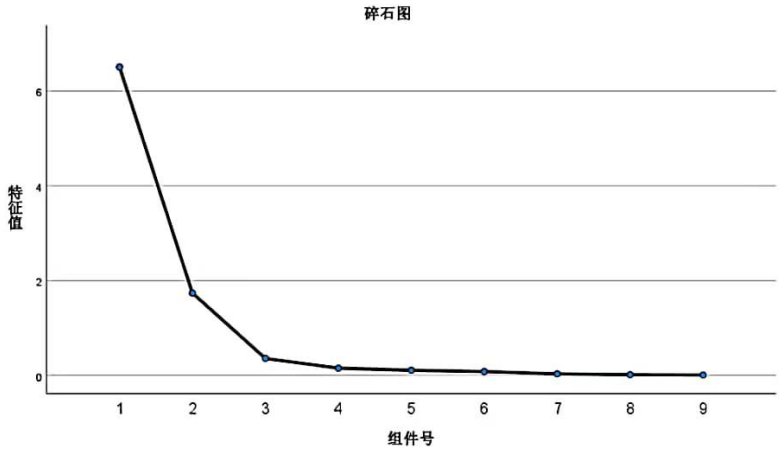

图 **6**　因子分析相关结果图

5.3 多元线性回归

九种期货价格数据均通过了因子分析, 接下来可以对多价格变量建立多元线性回归模型, 具体建模步骤如下：对于 9 种期货价格 $p_1, p_2, \ldots, p_9$, 取其中螺纹钢价格 p_1 作为因变量, 其余期货价格作为自变量, 建立多元线性回归方程

$$y=\hat{p_1}=\beta_1 p_2+\beta_2 p_3+\cdots+\beta_8 x_9+\varepsilon=\sum_{i=1}^{8}\beta_i p_{i+1}+\varepsilon \tag{7}$$

对于取 n 天数据可以得到方程组

$$\begin{cases} y_1=\beta_1 p_{21}+\beta_2 p_{31}+\cdots+\beta_8 p_{91}+\varepsilon \\ y_2=\beta_1 p_{22}+\beta_2 p_{32}+\cdots+\beta_8 p_{92}+\varepsilon \\ \cdots \\ y_n=\beta_1 p_{2n}+\beta_2 p_{3n}+\cdots+\beta_8 p_{9n}+\varepsilon \end{cases} \tag{8}$$

与一元简单回归类似, 多元回归对于各系数 β_i 的求解, 仍对选取的价格数据使用最小二乘法计算:

$$\sum_{m=1}^{n}(\hat{p_{1m}}-p_{1m})^2_{\min}=\sum_{m=1}^{n}(p_{1m}-\sum_{i=1}^{8}\beta_i x_{i+1m}-\varepsilon)^2_{\min} \tag{9}$$

使用 $SPSS\ Statistics$ 对上述模型进行数值求解, 得到结果如下所示：

模型	R	R 方	调整后 R 方	标准估算的错误	更改统计				
					R 方变化量	F 变化量	自由度 1	自由度 2	显著性 F 变化量
1	.994[a]	.988	.988	76.629	.988	7180.509	8	719	.000

期货种类	变量符号	回归系数	标准化数值	标准误差	p-value
铁矿石	ρ_2	β_1	0.028	0.035	0.002
不锈钢	ρ_3	β_2	-0.034	0.003	0.003
热轧卷板	ρ_4	β_3	0.761	0.013	<0.001
硅铁	ρ_5	β_4	-0.074	0.006	<0.001
焦煤	ρ_6	β_5	0.071	0.03	0.008
焦炭	ρ_7	β_6	-0.021	0.023	0.31
锰硅	ρ_8	β_7	0.094	0.008	<0.001
线材	ρ_9	β_8	0.211	0.017	<0.001

图 **7**　问题一回归分析结果

给出的多元线性回归相关系数 $R^2=0.988$, 故此多元线性回归方程与已有数据吻合度较好, 可以认为螺纹钢价格 p_1 与其余八种期货价格 $p_2,\ldots,p_9$ 间有较好的线性相关性质, 各价格数据较好满足以下方程：（系数 β_i 由上表中给出）

$$p_1=\sum_{i=1}^{8}\beta_i p_{i+1}+\varepsilon \tag{10}$$

由上述模型可以得到九种期货价格之间有较好的线性性质.

六、 问题二模型建立与求解

在本问中对于非线性关系求解, 选择使用多项式进行拟合. 根据第一问建立的多元线性模型, 可以先对其进行简要修改得到多元多项式非线性模型：

6.1 变量选取与多项式系数确定

首先为保持与第一问一致性, 在此问建模中仍使用螺纹钢价格 p_1 作为因变量, 其余八种期货价格作为自变量. 其次, 由于自变量数量较多, 每次添加幂次项相当于新引入若干个自变量, 则相应的需要增加若干参数. 若引入过于高阶项需要计算参数过多, 容易造成过度拟合. 故在初步模型中仅引入另外八种期货价格的平方项 $p_2^2,\ldots,p_8^2$ 进行非线性拟合. 建立如下二次非线性多元多项式模型：

$$p_1=\varepsilon+\sum_{i=1}^{8}\beta_i' p_{i+1}+\sum_{i=1}^{8}\beta_{i+8}p_{i+1}^2 \tag{11}$$

使用 $SPSS\ Statistics$ 进行多元回归分析, 得到结果如下图所示：

模型	R	R 方	调整后 R 方	标准估算的错误	更改统计					
					R 方变化量	F 变化量	自由度 1	自由度 2	显著性 F 变化量	德宾-沃森
1	.996[a]	.991	.991	65.106	.991	4991.360	16	711	.000	.348

变量种类	变量符号	回归系数	标准化数值	标准误差	p-value
铁矿石	ρ_2	β_1'	0.129	0.233	0.031
不锈钢	ρ_3	β_2'	-0.313	0.022	<0.001
热轧卷板	ρ_4	β_3'	0.322	0.078	<0.001
硅铁	ρ_5	β_4'	-0.725	0.024	<0.001
焦煤	ρ_6	β_5'	0.373	0.102	<0.001
焦炭	ρ_7	β_6'	-0.073	0.101	0.42
锰硅	ρ_8	β_7'	0.909	0.046	<0.001
线材	ρ_9	β_8'	0.109	0.112	0.367
铁矿石	ρ_2^2	β_9	-0.074	0	0.198
不锈钢	ρ_3^2	β_{10}	0.316	0	<0.001
热轧卷板	ρ_4^2	β_{11}	0.448	0	<0.001
硅铁	ρ_5^2	β_{12}	0.641	0	<0.001
焦煤	ρ_6^2	β_{13}	-0.325	0	<0.001
焦炭	ρ_7^2	β_{14}	0.115	0	0.203
锰硅	ρ_8^2	β_{15}	-0.82	0	<0.001
线材	ρ_9^2	β_{16}	0.041	0	0.728

图 **8**　问题二回归分析结果

引入平方项的多项式回归分析相关系数 $R^2 = 0.991$, 大于第一问中的多元线性回归相关系数 0.988, 从而较线性回归更为精确, 各期货价格很好地满足上述非线性方程. 其中 $\beta_i', \beta_{i+8}, i = 1, 2, ..., 8$, 即为表中数据.

6.2 对模型的进一步修改

为使得模型更加精确, 对于二次项还可引入交叉项 $p_i p_j, \forall i \neq j$ 进行拟合, 首先构造价格列向量 $p = (p_2, \ldots, p_9)^T \in \mathbb{R}^8$ 与系数行向量 $\beta' = (\beta_1', \ldots, \beta_8') \in \mathbb{R}_8$ 模型方程可改写为：

$$p_1 = \varepsilon + \beta' p + p^T B p \tag{12}$$

$B \in M_8(\mathbb{R})$ 是二次型对应的对称矩阵.

进一步地, 还可考虑螺纹钢价格 p_1 的二阶项可能带来的扰动, 可以将常数项作为因变量, 价格项作为自变量. 此时价格向量变为九维列向量 $x = (p_1, \ldots, p_9)^T \in \mathbb{R}^9$, 同理二次型矩阵与一次系数向量相应改变为九阶对称阵与九维行向量, 可以得到：

$$\begin{aligned} c =& x^T A x + \hat{\beta} x \\ A \in M_9(\mathbb{R})\, A^T =& A, \hat{\beta} \in \mathbb{R}_9 \end{aligned} \tag{13}$$

为使得模型有更简洁且易于求解的形式, 同时利于后续问题的计算, 在此舍去价格一次项, 将模型方程改写为：

$$c = x^T A x \tag{14}$$

6.3 对二次型模型的标准化

首先可以对于上述方程进行归一化, 即等式左右同时除以 c, 得到

$$\begin{aligned} 1 = x^T \hat{A} x \quad \hat{A} = \frac{1}{c} A \\ \hat{A} \in M_9(\mathbb{R}) \quad \hat{A}^T = \hat{A} \end{aligned} \tag{15}$$

根据矩阵对称性得到 $\hat{A}$ 有 45 个参数需要求解, 由最小二乘法确定参数, 给出确定的表示矩阵 $\hat{A}$ 后, $x^T \hat{A} x = 1$ 为 $\mathbb{R}^9$ 空间中一张确定的二次光滑超曲面, 每天所有种类期货价格状态对九维空间中的一个点.

由于价格数据变化较大, 且不同期货间价格差异大, 对于矩阵求解前可以先对价格列向量进行标准化. 即笔者希望对于数据进行某种可逆变换, 从而让所有价格向量中各元素均处于 $[0, 1]$ 区间内. 为达到此目的, 可以先选定一种期货价格 p_i 全部工作日数据, 选取其中价格最大值 $p_{\max}^{(i)}$, 与所有价格最小值 $p_{\min}^{(i)}$, 对于该种期货任意日期价格可以用如下公式进行标准化：

$$p_i^* = \frac{p_i - p_{\min}^{(i)}}{p_{\max}^{(i)} - p_{\min}^{(i)}} \tag{16}$$

对于每一种期货价格序列进行如上标准化, 可以得到每日标准化价格向量 x^*. 使用标准化后的数据进行二次型拟合, 可以计算得到 $1 = x^{*T} A x^*$ 中对称矩阵 A 各元素具体数值. 在接下来的计算中, 未经说明均使用 x 表示标准化后的价格向量, 那么得到最终二次型模型为：

$$x^T A x = 1 \tag{17}$$

使用 $Python$ 编程计算后只保留三位小数, 得到矩阵 A 如下：

$$A = \begin{bmatrix} 50.028 & -2.889 & 3.499 & -56.848 & -30.731 & -20.807 & 22.004 & 19.339 & 4.146 \\ -2.889 & 8.165 & 2.114 & -5.343 & -15.069 & 12.637 & -9.302 & 15.634 & -1.227 \\ 3.499 & 2.114 & 6.031 & -9.009 & -3.592 & -10.349 & 7.657 & 1.327 & 2.089 \\ -56.848 & -5.343 & -9.009 & 76.068 & 42.638 & 13.446 & -12.753 & -30.889 & -6.620 \\ -30.731 & -15.069 & -3.592 & 42.638 & 36.190 & 2.502 & -1.833 & -37.380 & 1.443 \\ -20.807 & 12.637 & -10.349 & 13.446 & 2.502 & 18.032 & -14.095 & 19.915 & -8.155 \\ 22.004 & -9.302 & 7.657 & -12.753 & -1.833 & -14.095 & 14.515 & -17.124 & 0.155 \\ 19.339 & 15.634 & 1.327 & -30.889 & -37.380 & 19.915 & -17.124 & 33.503 & 2.197 \\ 4.146 & -1.227 & 2.089 & -6.620 & 1.443 & -8.155 & 0.155 & 2.197 & 4.813 \end{bmatrix} \tag{18}$$

七、 问题三模型建立与求解

本问要求找出具体判据以判断价格关系是否处于窗口期, 在本问中我们将使用第二问中建立的二次型非线性模型作为不同期货价格间关系, 进而对二次型关系进行是否处于窗口期的判定. 第二问中二次型方程为：$1 = x^T A x$

7.1 判据选取

最直观且简单的判定依据即预测价格与实际价格的差距, 我们希望此种差距尽量小. 但由于使用非线性关系模型, 根据二次型逆向求解标准化价格向量步骤繁杂且解不具有唯一性, 故我们沿用考虑预测值与实际值差距这一主体思想, 但是对于数值选取可以做如下变动：

可以将已有数据的每日各期货价格标准化向量 x_i 带入二次型计算 $c_i = x_i^T A x_i$, 在此问题中使用预测值与实际值差值, 即

$$\Delta = 1 - c_i \tag{19}$$

来刻画模型精确性, 即当 $|\Delta|$ 较小接近 0 可以认为价格处于关系窗口期；若 $|\Delta|$ 较大则认为关系失效, 并未处于此种关系窗口期. 不难看出 Δ 与 0 的差距相较预测价格实际价格间差额与 0 差距间有很强的一致性, 且在非线性模型下, 该判据在算法简单性上远优于价格差额判据.

得到二次曲面拟合的时间-误差绝对值图像如下:（此处用交易日顺序编号大致指代时间）

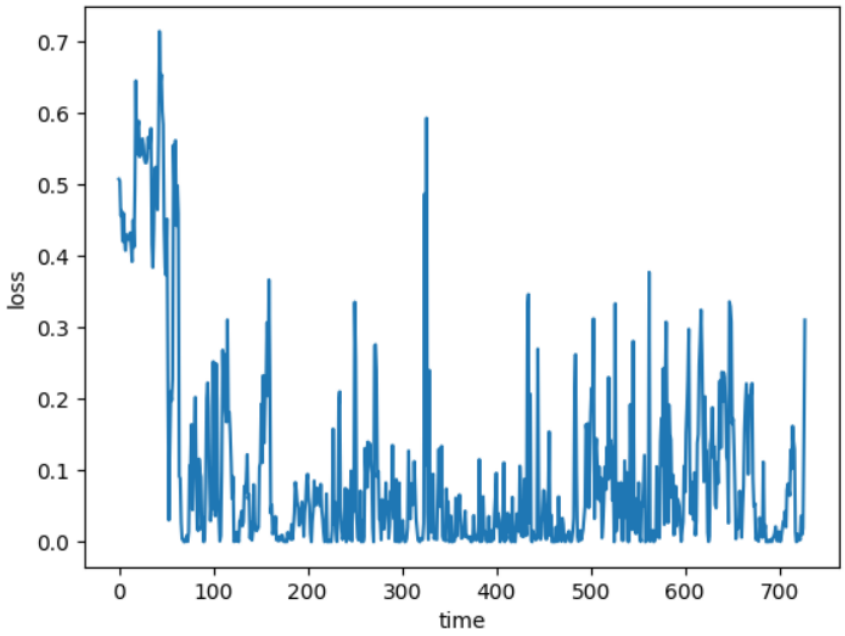

图 **9**　二次曲面拟合的时间-误差绝对值图像

可以看到, 大多数时段内误差值都处于不断波动之中, 但在有些时段内误差值较低且稳定. 这就是我们想要寻找的" 窗口期".

根据以上图像, 我们选取 $|\Delta|$ 临界值为 0.1 作为是否处于窗口期判据. 当 $|\Delta|$ 高于 0.1 时认为该关系不处于窗口期；当 $|\Delta|$ 不高于 0.1 时认为该关系处于窗口期.

7.2 可视化与投影

为进一步找到窗口期相关判据, 可以使用可视化手段进行分析, 具体操作步骤如下：将九维空间中计算得出 $|\Delta|$ 高于 0.1 的标准化数据点渲染为红色；计算得到 $|\Delta|$ 不高于 0.1 的标准化数据点渲染为绿色. 使用两种颜色对处于关系窗口期和不处于窗口区两类标准化数据点进行可视化区分. 而后选取某两期货标准化数据作为两维坐标轴, 使用数据可视化手段将数据点从高维空间投影到所选二维空间, 且在投影过程中保持数据点颜色. 例如所选标准化价格 p_1^*, p_2^* 为二维图像两坐标轴, 则可做如下投影进行变换得到二维图像：

$$\begin{aligned} f: \quad & \mathbb{R}^9 \longrightarrow \mathbb{R}^2 \\ & (p_1^*, \ldots, p_9^*) \longmapsto (p_1^*, p_2^*) \end{aligned} \tag{20}$$

7.3 机器学习支持向量机分析

支持向量机能够有效区分二维空间里面的数据点, 并且能在加入新的数据点之后比较容易地判断它属于哪一类. 对于给定的训练集 $T = \{(\boldsymbol{a}_1, c_1), (\boldsymbol{a}_2, c_2), \cdots, (\boldsymbol{a}_N, c_N)\}$, 其中, $\boldsymbol{a}_i \in \Omega \subset \mathbb{R}^n, \Omega$ 称为输入空间, 输入空间中的每一个点 $a_i = (a_{i1}, a_{i2}, \cdots, a_{in})^T$ 由

n 种期货的价格组成; $c_i \in \{-1, 1\}, i = 1, 2, \cdots, N$; 当该 a_i 对应的 $\Delta < 0.1$ 时, 设置 c_i 为 1, 否则设为 -1. 不妨假设 $c_j = 1, j = 1, 2, \cdots, N_1, c_j = -1, j = N_1 + 1, \cdots, N$, 即 1 类有 N_1 个训练样本点, -1 类有 $N - N_1$ 个训练样本点. 寻找 $\mathbb{R}^n$ 上的一个实值函数 $g(\boldsymbol{x})$, 以便用分类函数 $f(x) = \text{sgn}(g(x))$, 推断任意一个模式 $x = (x_1, x_2, \cdots, x_n)^T \in \mathbb{R}^n$ 相对应的输出 $f(x)$ 值的问题为分类问题.

因为空间 $\mathbb{R}^n$ 中超平面都可以写为 $\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x} + b = 0$ 的形式, 参数 $(\boldsymbol{\omega}, b)$ 乘以任意一个非零常数后得到的是同一个超平面, 定义满足条件

$$\begin{cases} c_i\left(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{a}_i + b\right) \geqslant 0, \quad i = 1, 2, \cdots, N, \\ \min_{i=1,2,\cdots,N}\left|\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{a}_i + b\right| = 1 \end{cases} \tag{21}$$

的超平面为训练集 T 的规范超平面.

对于线性可分的情况来说, 只有满足 $\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{a}_i + b = \pm 1$ 成立的 a_i 向量在建立分类超平面的时候起到了作用, 这种向量具有稀疏性, 即通常只占样本集很小的一部分, 它们之间的间隔为 $\frac{2}{\|\boldsymbol{\omega}\|}$.

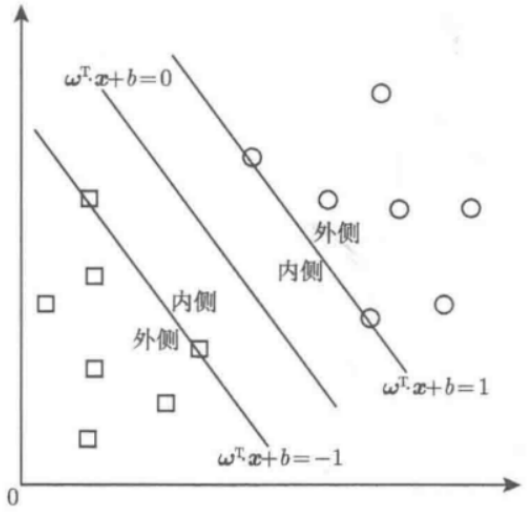

图 **10**　线性可分支持向量分类机[10]

为了寻找最优的超平面, 希望能够最大化 $\frac{2}{\|\boldsymbol{\omega}\|}$, 考虑分类边界 $\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{a}_i + b = \pm 1$, 将问题转化为如下二次规划问题：

$$\begin{array}{ll} \min & \frac{1}{2}\|\boldsymbol{\omega}\|^2, \\ \text{s.t.} & c_i\left(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{a}_i + b\right) \geqslant 1, \quad i = 1, 2 \cdots, N. \end{array} \tag{22}$$

7.4 结果图像展示与分析

部分两两期货组合的二维化视图结果如下所示：

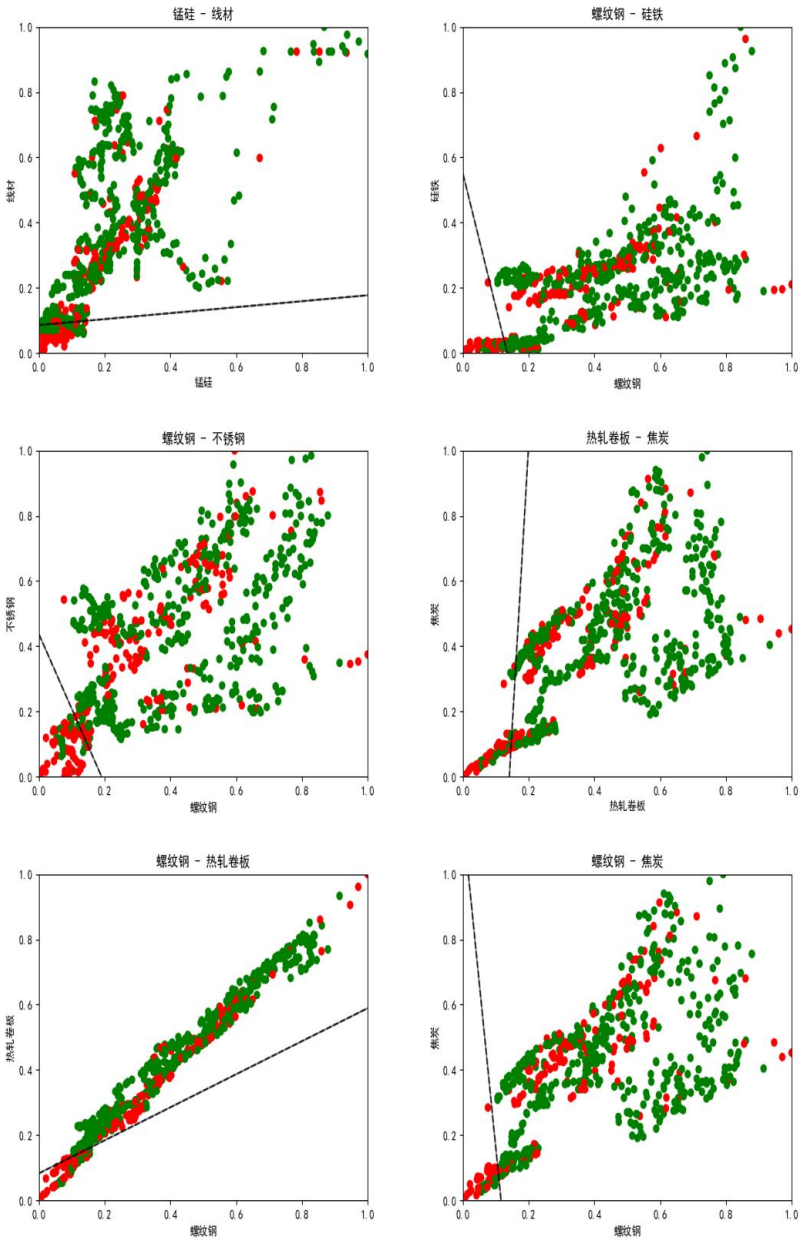

图 **11**　部分两两品种组合的二维化视图

图中黑色直线即为支持向量机对于不同颜色点分布区域进行的大致分割, 不难观察得到直线某侧主要分布红色数据点, 另一侧主要分布绿色数据点, 此分割较好的分离了处于窗口期与非处于窗口期的两类数据点. 可以选取表征较为明显的二维视图进行具体分析：

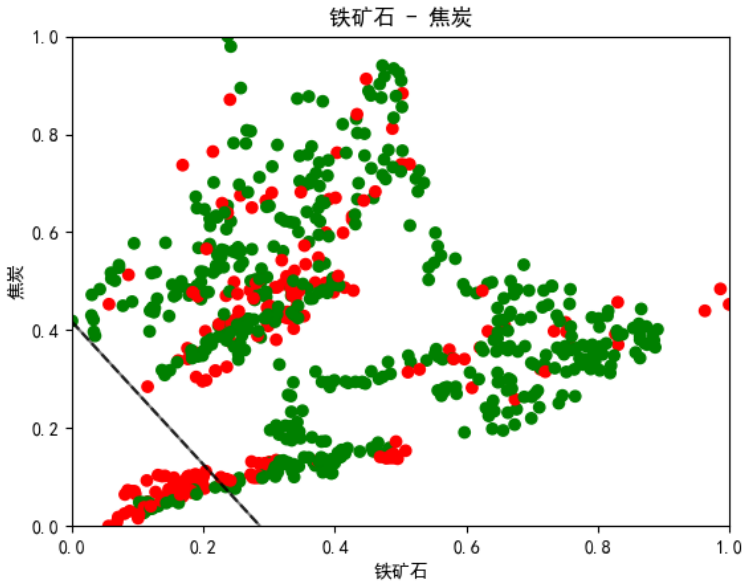

图 **12**　铁矿石-焦炭的二维化视图

在以铁矿石与焦炭标准化价格为坐标轴的对应图表中, 支持向量机辨认出了左下角是红色聚集区域, 也就是二次曲面拟合效果较差的区域. 根据需求我们可以对该分类结果继续细分, 例如若采取较保守的投资策略, 我们可以只取右半部分绿色占主导的区域. 这样, 我们就得到了判断窗口期的一个大致条件: 铁矿石标准化价格 > 0.5 且焦炭标准化价格 < 0.5.

7.5 进一步讨论

7.2 – 7.4 中都是对坐标轴平面的标准投影. 虽然易于分析, 但由于将一些仅在两个维度上相近, 在其余维度上实则相距较远的数据点混杂在一起, 导致分离效果有限. 一种改进方向是使用之前主成分分析的结果对特定的超平面进行投影来改进.

另一种改进方向是进行非线性的降维. 这里我们使用流形学习中常用的局部线性嵌入算法[3], 目的是利用流形局部同胚于欧式空间的特点, 通过保持局部线性关系将流形铺开到低维空间中.

对每个样本 ‘x_i‘ , 我们用其临近的几个样本进行线性重构.

$$x_i \approx \sum_j w_{ij} x_j, \sum_j w_{ij} = 1 \tag{23}$$

最优化以下函数

$$\varepsilon(W) = \sum_i \|x_i - \sum_j w_{ij} x_j\|^2 \tag{24}$$

得到重构系数 ‘$w_(ij)$‘ 后, 将数据集嵌入到二维平面上, 要保持该重构关系, 于是就成为最优化如下函数的问题:

$$\xi(Y) = \sum_i \|y_i - \sum_j w_{ij} y_j\|^2 \tag{25}$$

具体的计算过程可见[4]

效果如下图. 期货价格数据作为时间序列具有时间维度上的连续性, 但同时也有间断点. 同样以 Δ 的大小为基准进行染色, 颜色越深代表二次曲面拟合误差越大. 图中颜色较浅且间断点较少的区域, 可以被认为是窗口期.

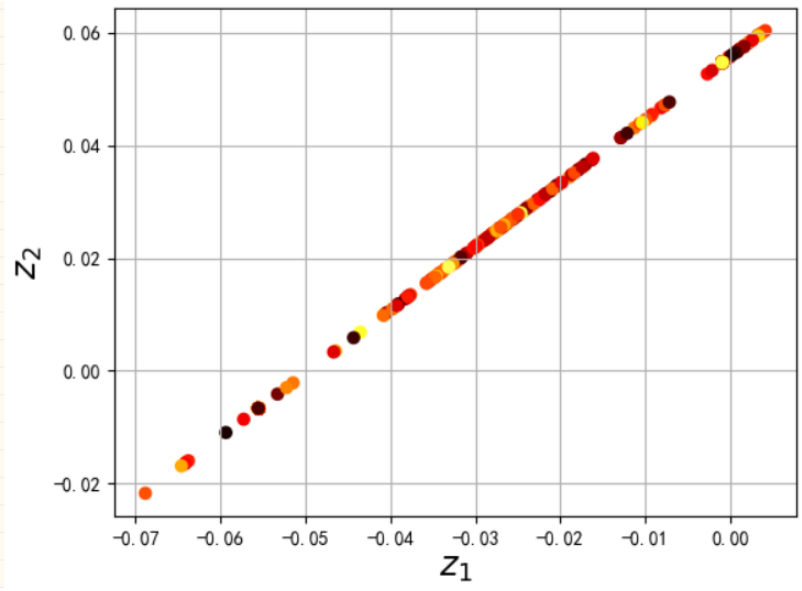

图 **13**　九种期货价格 **LLE** 平面嵌入图

八、 问题四模型建立与求解

第四问要求根据已建立模型对未来各期货价格进行预测, 根据预测价格确定投资策略以获得尽量高的投资回报率. 在本问中使用第二问所建立的二次型非线性模型对期货价格进行预测.

8.1 进一步假设与预测方法

根据对问题四的分析, 现提出如下假设：当标准化数据点与二次超曲面比较靠近时, 该数据点有按照对应二次型多项式 $f(x) = x^T Ax$ 梯度方向接近曲面的趋势. 而越接近曲面, 接近的速率就越小.

对于第二问中已经提出的二次型模型 $x^T Ax = 1$, 可以计算对应二次型 $f(x) = x^T Ax$ 的梯度

$$\nabla(x^T Ax) = (A + A^T)x = 2Ax \tag{26}$$

对于某一天的标准化价格向量 $x_0 \in \mathbb{R}^n$, 使用上述公式计算梯度 $\nabla(x_0^T Ax_0)$, 按照第三问中方法计算该点处 Δ, 根据假设认为在此时价格变化方向为 $\Delta \cdot \nabla(x_0^T Ax_0)$. 单位化价格变化方向后得到单位向量 $v = \frac{\Delta \cdot \nabla(x_0^T Ax_0)}{|\Delta \cdot \nabla(x_0^T Ax_0)|}$. 对于给定的变化步长 λ , 预测标准化价格为 $x_0 + \lambda v$, 单步对应变化量为 λv. 若想进行多步预测, 则按照单步预测方法进行迭代, 具体做法如下：若需要对某天标准化价格向量 x_0 进行 k 步预测, 首先按照单步预测得到 $x_1 = x_0 + \lambda v$, 而后计算 $\nabla(x_1^T Ax_1)$ 对 x_1 继续进行单步预测, 如上对单步预测算法迭代 k 次即得到预测标准化价格向量 x_k.

其中对于变化步长的取值, 可以先给出一个基准步长 λ_0 . 由于策略采取日线交易, 故希望单步步长预测为对下一交易日价格预测, 可以采取如下算法给出 λ_0 具体数值：对于所选取的 $n = 728$ 个交易日数据, 首先计算每日标准价化格向量 $x_i, 1 \leq i \leq n$, 而后计

算每相邻两日标准价格向量间距离 $|x_i - x_{i+1}|, 1 \leq i \leq n-1$. 将 λ_0 取为所有距离平均值 $\lambda_0 = \frac{\sum_{i=1}^{n-1} |x_i - x_{i+1}|}{n-1}$, 计算得到 λ_0 具体取值为 0.01.

对于步长确定还可以有进一步细化：设 $\lambda = \alpha(\Delta)\lambda_0$, 其中步长缩放系数 $\alpha(\Delta)$ 是一个关于 Δ 的函数. 我们这里取为 $\frac{1+|\Delta|}{1+\frac{1}{4}|\Delta|}$. 这样可以使步长始终处于区间 $(1,4)$ 的合理范围内, 同时 Δ 的增大对步长有正贡献.

需要注意的是, 以上计算均使用标准化后价格向量, 若想要得到预期变化后的实际价格 x_e, 需要根据我们使用的标准化方法对标准化价格向量进行如下逆向运算：

$$x_e = \text{diag}\left\{p_{\text{max}}^{(1)} - p_{\text{min}}^{(1)}, \ldots, p_{\text{max}}^{(9)} - p_{\text{min}}^{(9)}\right\}(x + \lambda v) + (p_{\text{min}}^{(1)}, \ldots, p_{\text{min}}^{(9)})^T \tag{27}$$

x_e 即为实际非标准化的预测价格向量. 将 x_e 减去真实价格 x_{original} 可以得到预测实际价格变化：

$$\begin{aligned} x_{\text{original}} &= \text{diag}\left\{p_{\text{max}}^{(1)} - p_{\text{min}}^{(1)}, \ldots, p_{\text{max}}^{(9)} - p_{\text{min}}^{(9)}\right\} x + (p_{\text{min}}^{(1)}, \ldots, p_{\text{min}}^{(9)})^T \\ x_{\text{change}} &= x_e - x_{\text{original}} = \text{diag}\left\{p_{\text{max}}^{(1)} - p_{\text{min}}^{(1)}, \ldots, p_{\text{max}}^{(9)} - p_{\text{min}}^{(9)}\right\} \lambda v \end{aligned} \tag{28}$$

根据预测价格变化 x_{change} 可进一步确定投资策略.

8.2 交易策略制定

对于一定的投资金额, 为获得最大投资回报率, 基于得到的预测价格变化 x_{change}, 再由此计算出价格变化比率 r_{change} .

$$\begin{aligned} x_{\text{change}} &= (p_{\text{change}}^{(1)}, \ldots, p_{\text{change}}^{(9)}) \\ r_{\text{change}}^{(i)} &= \frac{p_{\text{change}}^{(i)} - p_i}{p_i} \\ r_{\text{change}} &= (r_{\text{change}}^{(1)}, \ldots, r_{\text{change}}^{(9)}) \end{aligned} \tag{29}$$

为获得最大投资回报率, 可以制定如下基本交易策略：取向量 r_{change} 分量正向最大值对应期货买入, 即对于预测价格增长率最大期货买入；取该向量中分量负向最大值对应期货卖出, 即对于预测价格下跌率最大期货卖出. 根据以上基本交易方式, 继续给出以下假设：第一, 起始资金为 1000w；第二, 每次交易至多买卖一种期货；第三, 交易忽略手续费并且均采用 1 倍杠杆的情况. 下制定了三种交易策略. 对于每种策略, 使用 Python 程序将其在实际数据集上的盈利情况进行了模拟. （代码详见附录）

8.2.1 第一种交易策略

1. 限定每次交易的数量, 即每日买卖各种期货分别不超过 10 手 (1 手 10 吨)；
2. 投资者拥有的全部资金均可用于交易；
3. 对于预测步数与交易间隔时间不做要求；
4. 总交易时长为三年.

下面给出不同预测步数与交易间隔时间下三年总投资回报率图：

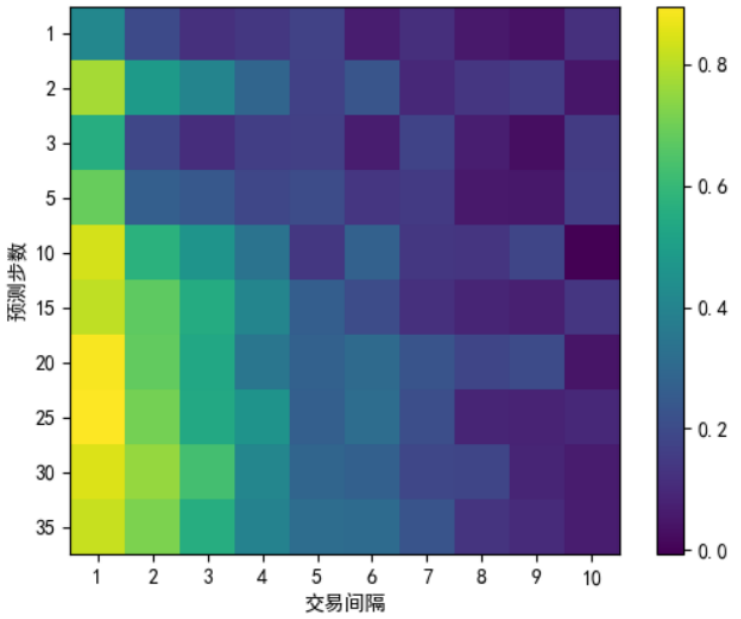

图 **14**　预测步数-交易间隔对三年总投资回报率的影响 **(**固定买入手数**)**

图像中方块不同颜色代表了不同的投资回报率, 可以看到预测步数的增加导致预测效果的提升, 从而促进盈利. 图表中预测不是为 25 步交易间隔为 1 天时三年总投资回报率最高, 为 89.6%.

8.2.2 第二种交易策略

1. 固定每次交易金额为 70 w, 即每次均按照 70 w 价格买入或售出某种期货；
2. 全部资金均可用于交易；
3. 对于预测步数与交易间隔时间不做要求；
4. 总交易时长为三年.

不同预测步数与交易间隔时间下三年总投资回报率如下图所示：

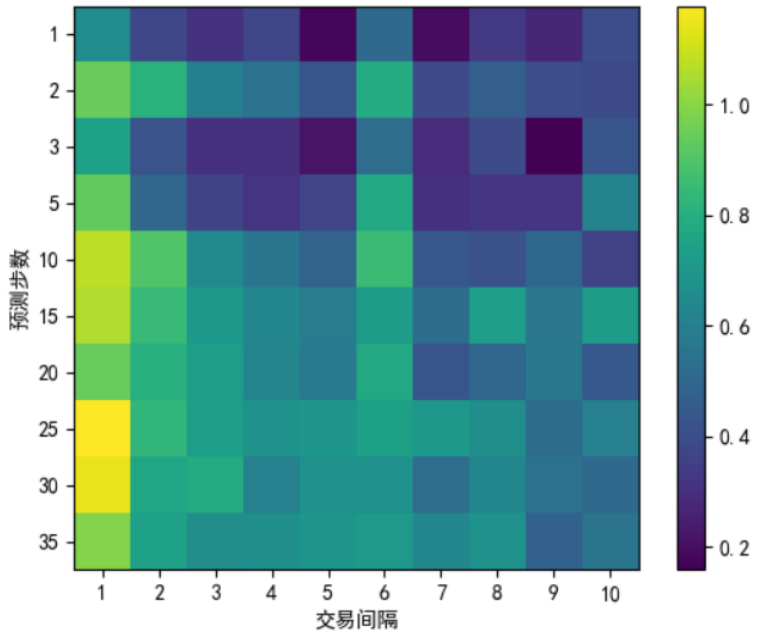

图 **15**　预测步数-交易间隔对三年总投资回报率的影响 **(**固定买入金额**)**

观察该图表可大致得出结论：在所给范围内, 交易间隔较长且预测步数较长时投资回报率较高. 图表中交易间隔为 1 天, 预测步长为 25 时三年总投资回报率最高, 为 117.8%.

8.2.3 第三种交易策略

在期货市场中, 若仓位过重甚至是满仓交易, 即大部分资金用于购买期货, 账户可用金过少, 则可能导致无法应对波动率风险, 甚至是在震荡行情中引发爆仓, 这显然与期望得到回报率最大相悖. 为避免上述情况发生, 在 1 倍杠杆前提下提出如下交易策略：

1. 交易总时长仍为三年；
2. 每天均可进行交易, 且可选取不同预测步数对价格进行预测；
3. 第一天交易使用本金中一半资金买入向量 r_{change} 分量正向最大值对应的期货；
4. 第二天交易前先对前日买入期货价格进行预测, 若预测结果表明价格将下降则清仓; 若不然则计算当日对应的预测价格变化率向量 r_{change} , 判断该向量分量正向最大值对应的期货是否与前日买入期货相同, 如果相同则保持仓位不动, 如果不相同则选择清仓之后再用一半资金买入当日预测价格变化率 r_{change} 分量正向最大值对应的期货；
5. 重复上述步骤 4;

由于每次均使用总资金一半进行期货购买, 另外一半资金为账户可用资金. 在 1 倍杠杆前提下可以有效应对可能被强制平仓的风险. 最后获得的总收益与预测步数关系如下表所示,

表 **1**　预测步数与总收益

预测步数	1	2	3	5	10
总收益	82.99%	77.85%	71.87%	88.80	112.87%
预测步数	15	20	25	30	35
总收益	119.60%	152.30%	138.49%	161.27%	138.45%

在给出预测步数中, 预测步数为 30 时三年总投资回报率最高为 161.27%, 将每次买入/卖出用红点标记, 可以绘制出总资产随时间变化图像：

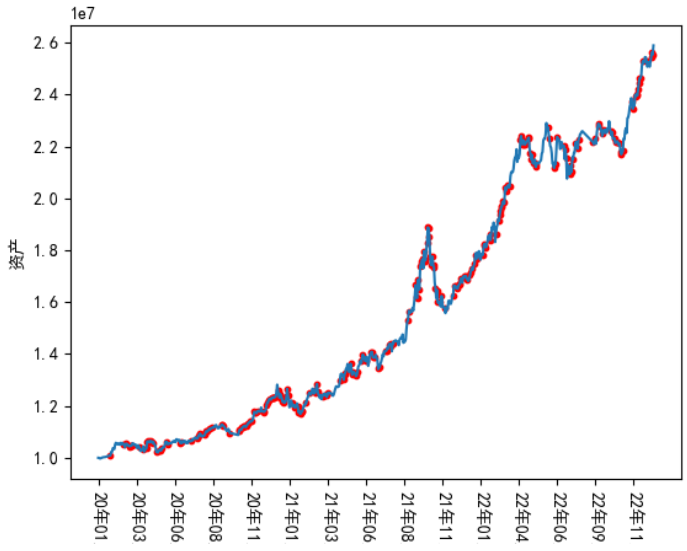

图 **16**　预测步数为 **30** 时总资产随时间变化图像

比较已提出的三种投资策略, 可知采用第三种投资组合策略得到的收益相较于前两种策略更高,, 同时选用范围 $1-35$ 步数对价格进行预测, 最终得到的投资回报率均维持在较高水平, 且该种策略很好的规避了强制平仓风险, 对于现实期货交易有较好的借鉴意义.

接下来可以进行进一步分析交易具体情况：以螺纹钢为例, 下图用可视化手段在螺纹钢价格—时间图上展示其在第三种策略中的交易情况. 其中红点代表在该交易日买入, 绿点表示卖出. 不难看出在螺纹钢交易过程中, 模型预测成功让投资者多次短线获利. 说明二次型非线性模型与沿梯度逼近的假设在实际投资问题分析中具有一定的实用性.

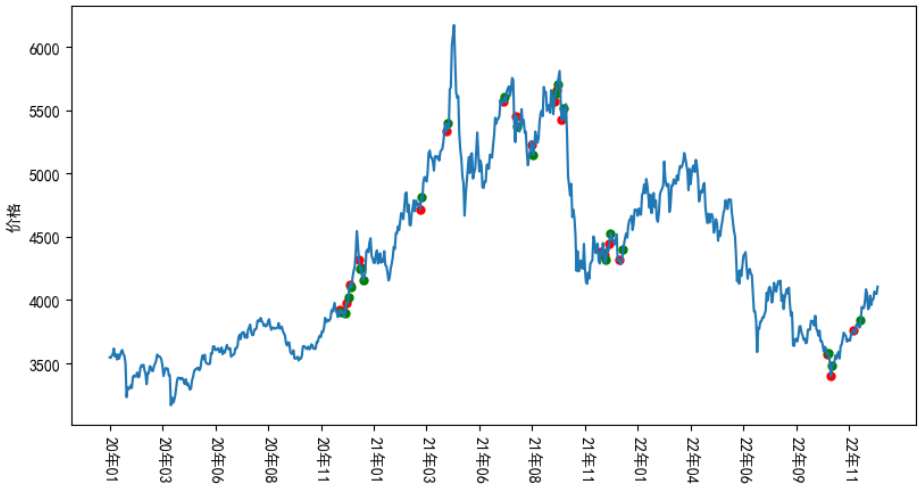

图 **17**　交易过程中螺纹钢买入卖出节点图

九、 模型评价与改进方向

9.1 模型亮点

从几何的观点构建模型, 本文

1. 在问题一数据预处理中进行了时间序列平稳性分析, 为后续模型提供了时间平稳性证明, 使得建模更加严谨.
2. 在问题二求解过程中开创性的构造了二次型非线性模型, 将数据问题与线性代数知识相结合, 并且得到了非常简洁的方程形式, 减少后续问题分析的复杂程度.
3. 在问题三建模过程中采取了可视化与机器学习手段, 使用二维图像形式具象展示了处于窗口期与非窗口期数据点分布情况.
4. 在问题四中使用二次型函数梯度对于未来价格进行预测对于有较高准确性, 提出的投资策略均有较高回报率, 说明二次型非线性模型可解释性很强, 对实际操盘有着一定指导意义.

9.2 模型的进一步改进

对于问题 2, 在求出矩阵 A 之后, 我们可以通过对其进行奇异值分解, 通过舍弃较小的奇异值来达到进一步降维的效果. 也就是说, 该模型提供了一种方便的动态调节参数量大小/模型精确性的方式. 这可以用于缓解过拟合和控制计算成本.

对于问题 3, 为了找出一个判据来进一步确定这 9 种期货之间相互关系存在的窗口期, 除了采用支持向量机的方法, 我们还可以对 728 组数据进行滚动回归, 对于第 t 日数据以 n 作为窗口期, 在 $[t-n,t]$ 的窗口中进行数据回归, 这是金融学经常使用的一种方

法[5]. 通过使用过去一段时间内的数据来估计回归系数, 并观察其变化情况. 如果回归系数在某个时间段内比较稳定, 那么可以认为这个时间段内的关系是有效的；如果回归系数在某个时间段内发生较大波动或变化趋势, 那么可以认为这个时间段内的关系是失效或变化的. 这样, 我们就可以根据回归系数的稳定性来判断关系的时效性, 以此确定关系存在的窗口期.

对于问题 4, 我们没有令时间作为一个单独的维度, 缺失了数据点在时间上相距较近/较远的信息. 更本质地, 如果把曲面看成是动力系统的相图中的一部分, 我们假设它是吸引的. 但这对分析本身就落在曲面上的数据点, 或者与曲面较接近的点, 就没有多大帮助了. 具体表现在变化步长策略中, 当 Δ 较小时, 我们的模型给出的策略往往是极少买进. 加入时间信息最直接的方式是将时间作为一个新的维度. 但由于时间的严格递增性, 在实际预测应用中导致当前状态点偏离已知数据点, 从而导致拟合效果不佳. 此外, 对于以多项式为基础的模型来说, 难以捕捉时间序列的周期性, 对时间距离较远的数据点不能顾及, 并且容易产生龙格现象[6]. 因此在该问题上不宜采取这种方式. 为了将时间信息加入到模型的训练数据集中, 也可以采取滚动窗口回归提取数据点随时间变化趋势的方法, 将每个数据点处窗口拟合的直线参数作为数据点的坐标, 扩展数据集到 18 维. 在这个新的空间中依然采用二次曲面进行拟合. 这样既保留了时间信息, 又避免了以上缺陷. 但是由于笔者仅使用了近三年的日线数据, 经整理后有 728 个数据点, 而 18 维空间中的二次曲面有 171 个参数, 过拟合不可避免, 难以对模型的有效性合理评估. 笔者猜想该方法适用于更大一些的时间序列数据集.

在整一个问题求解过程中, 均只考虑了简单情况, 即所有期货交易忽略手续费并且均采用 1 倍杠杆, 后续可加入手续费因素, 不仅可以考虑其他倍率的杠杆, 还可以考虑不同的期货之间不同的手续费和杠杆倍率来进一步扩大模型的适用性. 另外, 还可以对钢铁产业链条的上下游关系进行分析来进一步扩大模型的准确性.

参考文献

[1] https://finance.sina.com.cn/money/future/fmnews/2019-09-25/doc-iicezueu8213630.shtml

[2] Cureton, E.E., & D'Agostino, R.B. (1993). Factor Analysis: An Applied Approach (1st ed.). Psychology Press: 389-391.

[3] 马瑞, 王家廙, 宋亦旭. 基于局部线性嵌入 (LLE) 非线性降维的多流形学习 [J]. 清华大学学报：自然科学版,2008(4):582-585.

[4] Geron Aurelien, Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems (2nd edition), 东南大学出版社, 2020.05: 304-308.

[5] 陈晨. 中国黑色系商品期货跨种类套利研究 [D]. 上海财经大学, 2020. DOI: 10.27296/d.cnki. gshcu.2020.002015.

[6] Epperson, James. On the Runge example. Amer. Math. Monthly. 1987, **94**: 329–341 [2018-04-27]. doi:10.2307/2323093.

[7] 敖艺璇. 商品期货跨品种套利策略设计 [D]. 电子科技大学, 2022. DOI:10.27005/d.cnki.gdzku.2022.004744.

[8] 张静文. 商品期货跨品种套利策略的实证研究——以铁矿石、焦炭为例 [J]. 现代商业, 2020,No.581(28):90-92.DOI:10.14097/j.cnki.5392/2020.28.035.

[9] 周亮. 基于协整的期货跨品种套利研究——以黑色系期货为例 [J]. 价格理论与实践,2017, No.394(04):112-115.DOI:10.19851/j.cnki.cn11-1010/f.2017.04.029.

[10] 司守奎，孙玺菁主编. Python 数学实验与建模 [M]. 北京：科学出版社, 2020.03: 522.

附录 A　核心投资模拟程序

```
import numpy as np
import quadricSurface
from matplotlib import pyplot as plt

plt.rcParams['font.sans-serif'] = ['Simhei'] #显示中文
plt.rcParams['axes.unicode_minus'] = False #显示负号

class Market:
    def __init__(self, dates, prices, day=0):
        self.day = day
        self.dates = dates
        self.prices = prices
        self.futures = ["螺纹钢", "铁矿石", "不锈钢", "热轧卷板", "硅铁", "焦煤", "焦炭", "锰硅",
            "线材"]

    def get_price(self, future):
        return self.prices[future][self.day]

    def get_all_prices(self):
        return np.array([self.get_price(_) for _ in range(9)])

    def next_day(self, days=1):
        self.day += days
        if self.day >= len(self.dates):
            raise Exception("day number exceeded")

    def measure_property(self, player):
        return sum([self.get_price(_) * player.positions[_] for _ in range(len(self.futures))])
            + player.money

class Player:
    def __init__(self, money=1e7, positions=None):
        self.money = money
        self.history = []
        if positions:
            self.positions = positions
        else:
            self.positions = np.zeros(9)
        self.future_buy_points_x = [[], [], [], [], [], [], [], [], []]
        self.future_sell_points_x = [[], [], [], [], [], [], [], [], []]

    def buy_amount(self, future, amount, market: Market):
        if amount < 0:
            raise Exception(f"Negative amount! Trying to buy {amount} of
                {market.futures[future]}.")
        need_pay = market.get_price(future) * amount
        if self.money < need_pay:
            raise Exception(f"Not enough money! {self.money} < {need_pay}.")
        else:
            self.money -= need_pay
            self.positions[future] += amount
        self.history.append(f"buy {amount} units of {market.futures[future]}")
        self.future_buy_points_x[future].append(market.day)

    def buy_money(self, future, money, market: Market):
        if money < 0:
            raise Exception(f"Negative money! Trying to buy {money} yuan of
                {market.futures[future]}.")
        elif money > self.money:
            raise Exception(f"Not enough money! Trying to buy {money} yuan of
                {market.futures[future]}. But only have {self.money} yuan.")
```

```
        want_amount = money // market.get_price(future)
        actually_pay = want_amount * market.get_price(future)
        self.money -= actually_pay
        self.positions[future] += want_amount
        self.history.append(f"buy {want_amount} units of {market.futures[future]}")
        self.future_buy_points_x[future].append(market.day)

    def sell_money(self, future, money, market: Market):
        if money < 0:
            raise Exception(f"Negative money! Trying to sell {money} yuan of
                {market.futures[future]}.")
        want_amount = money // market.get_price(future)
        actually_sell = want_amount * market.get_price(future)
        if want_amount > self.positions[future]:
            raise Exception(f"Not enough amount! Trying to sell {actually_sell} yuan of
                {market.futures[future]} (amount: {want_amount}). But only have
                {self.positions[future]} of it.")
        self.money += actually_sell
        self.positions[future] -= want_amount
        self.history.append(f"sell {want_amount} units of {market.futures[future]}")
        self.future_sell_points_x[future].append(market.day)

    def sell_amount(self, future, amount, market: Market):
        if amount < 0:
            raise Exception(f"Negative amount! Trying to sell {amount} of
                {market.futures[future]}.")
        elif amount > self.positions[future]:
            raise Exception(f"Not enough position! Trying to sell {amount} of
                {market.futures[future]}. But only have {self.positions[future]}.")
        want_money = market.get_price(future) * amount
        self.money += want_money
        self.positions[future] -= amount
        self.history.append(f"sell {amount} units of {market.futures[future]}")
        self.future_sell_points_x[future].append(market.day)

    def log_position(self, market: Market):
        return f"持仓: {[str(market.futures[_]) + ': ' + str(self.positions[_]) for _ in
            range(len(self.positions))]}"

    def sell_out(self, market: Market):
        for (future_index, position) in enumerate(self.positions):
            if position:
                self.sell_amount(future_index, position, market)

def predict_price(market: Market, step_number, step_length=0.01):
    price_max = np.array([np.max(market.prices[_]) for _ in range(len(market.prices))])
    price_min = np.array([np.min(market.prices[_]) for _ in range(len(market.prices))])
    price_scaling = price_max - price_min

    current_price = market.get_all_prices()
    new_price = current_price

    for step in range(step_number):

        grad = quadricSurface.grad(quadricSurface.result, (new_price - price_min) /
            price_scaling)
        difference_with_C = 1 - quadricSurface.my_func(quadricSurface.result, (new_price -
            price_min) / price_scaling)

        direction = grad * difference_with_C
        direction = direction / np.linalg.norm(direction)

        actual_step_length = (1 + abs(difference_with_C)) / (4 + abs(difference_with_C)) *
            step_length * 4
```

```
        # print(actual_step_length) # debug

        standardized_price_difference = direction * actual_step_length
        new_price += price_scaling * standardized_price_difference

    return new_price

def run_with_strategy(initial_money=10000000, predict_step_number=1, predict_step_length=0.01,
    trade_scale=100, trade_interval=1, trade_money=0.):
    future_market = Market(quadricSurface.date_list, quadricSurface.future)
    player = Player(money=initial_money)

    strategy = ["scale", "money"][int(trade_money > 0)]

    # main loop
    while future_market.day < len(future_market.dates) - trade_interval:
        current_price = future_market.get_all_prices()
        predicted_price = predict_price(future_market, predict_step_number, predict_step_length)
        price_difference = predicted_price - current_price
        price_difference_ratio = price_difference / current_price

        max_profit_future = np.argmax(price_difference_ratio)

        ratio_of_owned = price_difference_ratio[player.positions > 0]
        if ratio_of_owned.size > 0:
            min_profit = np.min(ratio_of_owned)
            min_profit_future = list(price_difference_ratio).index(min_profit)
        else:
            min_profit_future = 0

        if price_difference[max_profit_future] > 0:
            if strategy == "scale":
                if player.money - future_market.get_price(max_profit_future) * trade_scale >= 0:
                    player.buy_amount(max_profit_future, trade_scale, future_market)
            elif strategy == "money":
                player.buy_money(max_profit_future, min(trade_money, player.money), future_market)
        if price_difference[min_profit_future] < 0:
            if strategy == "scale":
                if player.positions[min_profit_future] >= trade_scale:
                    player.sell_amount(min_profit_future, trade_scale, future_market)
            elif strategy == "money":
                player.sell_money(min_profit_future, min(trade_money,
                    player.positions[min_profit_future] *
                    future_market.get_price(min_profit_future)), future_market)

        future_market.next_day(trade_interval)

    # report final status
    # print(player.money, player.positions)
    # print(future_market.measure_property(player))
    return (future_market.measure_property(player) - initial_money) / initial_money

def greed_trade(display_graph=True, predict_step_length=10):
    future_market = Market(quadricSurface.date_list, quadricSurface.future) # 市场日期和市场期货
    initial_money = 10000000
    player = Player(money=initial_money)

    price_difference = predict_price(future_market, 1) - future_market.get_all_prices()
    price_difference_ratio = price_difference / future_market.get_all_prices()

    max_profit_future = np.argmax(price_difference_ratio)
    min_profit_future = np.argmin(price_difference_ratio)
    player.buy_money(max_profit_future, initial_money / 2, future_market)
```

```
property_history = []
buy_points_x = []
buy_points_y = []
sell_points_x = []
sell_points_y = []

while future_market.day < len(future_market.dates) - 1:
    max_1 = max_profit_future
    price_difference = predict_price(future_market, predict_step_length) -
        future_market.get_all_prices()
    price_difference_ratio = price_difference / future_market.get_all_prices()

    max_profit_future = np.argmax(price_difference_ratio)
    min_profit_future = np.argmin(price_difference_ratio)
    if future_market.get_price(max_profit_future) >= 0:
        if max_profit_future == max_1:
            ...
        else:
            player.sell_out(future_market)
            player.buy_money(max_profit_future, player.money / 2, future_market)
            buy_points_x.append(future_market.day)
    else:
        player.sell_out(future_market)

        sell_points_x.append(future_market.day)

    print(player.history)
    player.history.clear()
    print(player.log_position(future_market))
    # print(future_market.measure_property(player))
    property_history.append(future_market.measure_property(player))
    if buy_points_x and buy_points_x[-1] == future_market.day:
        buy_points_y.append(property_history[-1])
    if sell_points_x and sell_points_x[-1] == future_market.day:
        sell_points_y.append(property_history[-1])
    future_market.next_day()

m = (future_market.measure_property(player) - initial_money) / initial_money

print('总收益: {:.2%}'.format(m))

if display_graph:
    original_date = quadricSurface.date_list
    formatted_date = [f"{str(original_date[_])[2:4]}年{str(original_date[_])[4:6]}月" for _
        in range(len(original_date))]
    fig, ax = plt.subplots()

    plt.xlabel(f'时间')
    plt.ylabel(f'资产')

    plt.xticks(range(0, len(property_history), 50), np.array(formatted_date)[::50],
        rotation=-90)

    watch_future = 1
    plt.plot(future_market.prices[watch_future])
    plt.scatter(player.future_buy_points_x[watch_future],
        [future_market.prices[watch_future][_] for _ in
        player.future_buy_points_x[watch_future]], color='red', s=25)
    plt.scatter(player.future_sell_points_x[watch_future],
        [future_market.prices[watch_future][_] for _ in
        player.future_sell_points_x[watch_future]], color='green', s=25)
    # plt.plot(property_history)
    # plt.scatter(buy_points_x, buy_points_y, color='red', s=10)
    # plt.scatter(sell_points_x, sell_points_y, color='green', s=25)
```

```
        plt.show()

    return m

if __name__ == "__main__":
    # # strategy 1, 2 heat map
    # profits = np.zeros((10, 10))
    #
    # for i in range(10):
    #     for j in range(10):
    #         predict_step_number = [1, 2, 3, 5, 10, 15, 20, 25, 30, 35][i]
    #         trade_interval = j + 1
    #         profit_rate = run_with_strategy(initial_money=10000000,
    #                                         predict_step_number=predict_step_number,
    #                                         predict_step_length=0.01,
    #                                         trade_scale=100,
    #                                         trade_interval=trade_interval,
    #                                         # trade_money=700000
    #                                         )
    #         profits[i][j] = round(profit_rate, 3)
    #
    # print(profits.tolist())
    # print(profits)

    # # Custom strategy loop
    # future_market = Market(quadricSurface.date_list, quadricSurface.future)
    # player = Player(money=10000000)
    #
    # while future_market.day < len(future_market.dates) - 1:
    #     player.buy_amount(0, 1, future_market)
    #     player.sell_out(future_market)
    #
    #     print(player.history)
    #     player.history.clear()
    #     print(player.log_position(future_market))
    #     future_market.next_day()
    #
    # print(future_market.measure_property(player))

    # greed_trade
    # profits = []
    #
    # for i in [1, 2, 3, 5, 10, 15, 20, 25, 30, 35]:
    #     profit = greed_trade(display_graph=False, predict_step_length=i)
    #     profits.append(profit)
    #
    # print(profits)

    #graph
    for i in [30]:
        profit = greed_trade(display_graph=True, predict_step_length=i)
```

附录 B　数据预处理程序

2.1　读取表格

```
import pandas as pd
import numpy as np

pd.options.mode.chained_assignment = None
```

```
# construct the final form
key_list = list('012345678')
real_dates = [date.strftime('%Y%m%d') for date in pd.date_range(start='2020-01-01',
    end='2022-12-31')]
final_dict = {'date': real_dates}
final_dict.update({key: np.zeros(len(real_dates)) for key in key_list})
final_form = pd.DataFrame(final_dict)
# final_form = pd.read_excel("test.xlsx")

# name is among 'rb', 'ss', 'hc', 'wr' - 0, 2, 3, 8
def read_0238form(file_path, year, month, name, number_name):

    if int(year) < 2021 or year == '2021' and month == '04':
        form = pd.read_excel(file_path, skiprows=2)
        all_contracts = list(form['合约'].values)
        all_prices = list(form['收盘价'].values)
        all_dates = list(form['日期'].values)
        all_amounts = list(form['成交量'].values)
    else:
        form = pd.read_excel(file_path, skiprows=3)
        # print(form)
        all_contracts = list(form['Contract'].values)
        all_prices = list(form['Close'].values)
        all_dates = list(form['Date'].values)
        all_amounts = list(form['Volume'].values)

    # 补全合约列
    prev_string = None
    for i in range(len(all_contracts)):
        if isinstance(all_contracts[i], str):
            prev_string = all_contracts[i]
        elif isinstance(all_contracts[i], float) and prev_string is not None:
            all_contracts[i] = prev_string

    # 只保留对应名字的合约
    valid_indexes = [i for i in range(len(all_contracts)) if all_contracts[i][:2] == name]
    all_contracts = [all_contracts[_] for _ in valid_indexes]
    all_prices = [all_prices[_] for _ in valid_indexes]
    all_dates = [all_dates[_] for _ in valid_indexes]
    all_amounts = [all_amounts[_] for _ in valid_indexes]

    date_prefix = f"{year}{month}"

    # 找到主力合约
    for day in range(1, 32):
        day = str(day)
        if len(day) == 1:
            day = '0' + day
        # print(day)
        indexes = [_ for _ in range(len(all_dates)) if str(int(float(str(all_dates[_]))))[-2:]
            == day]
        # print(indexes)
        if indexes:
            max_index = all_amounts.index(max([all_amounts[_] for _ in indexes]))
            if max_index:
                break
            else:
                continue
        else:
            continue

    # try:
    #     assert max_index
```

```
    # except Exception as e:
    #     print(e)
    #     print(indexes)
    #     print(all_dates)

    # print(all_amounts[max_index])

    # 从主力合约开始向下找到所有有效日期(寻找31格)
    for day in range(1, 32): # 遍历所有日期
        day = str(day)
        if len(day) == 1:
            day = '0' + day
        date_str = f"{date_prefix}{day}"
        for row in range(max_index, min(max_index + 31, len(all_dates))):
            if date_str == str(int(float(str(all_dates[row])))):
                # 将这一天这一单的收盘价放入总表
                # print("added")
                date_index = list(final_form['date']).index(date_str)
                final_form[number_name][date_index] = all_prices[row]
                break


# read_0238form("data/rawData/2019/0238/所内合约行情报表2019.1.xls", '2019', '01', 'rb', '0')
# print(final_form)
# final_form.to_csv("test.csv")
# exit()

def read_156form(file_path, year, number_name):
    form = pd.read_excel(file_path)
    for month in ['01', '02', '03', '04', '05', '06', '07', '08', '09', '10', '11', '12']:

        all_contracts = list(form['合约'].values)
        all_prices = list(form['收盘价'].values)
        all_dates = list(form['日期'].values)
        all_amounts = list(form['成交量'].values)

        # 找到主力合约
        for day in range(1, 32):
            day = str(day)
            if len(day) == 1:
                day = '0' + day
            # print(day)
            indexes = [_ for _ in range(len(all_dates)) if str(all_dates[_])[-4:] == month + day]
            # print(indexes)
            if indexes:
                max_index = all_amounts.index(max([all_amounts[_] for _ in indexes]))
                if max_index:
                    break
                else:
                    continue

        date_prefix = f"{year}{month}"
        # print(max_index)

        # 从主力合约开始向下找到所有有效日期(寻找31格)
        for day in range(1, 32): # 遍历所有日期
            day = str(day)
            if len(day) == 1:
                day = '0' + day
            date_str = f"{date_prefix}{day}"
            for row in range(max_index, min(max_index + 31, len(all_dates))):
                # print("yes")
                if date_str == str(all_dates[row]):
                    # 将这一天这一单的收盘价放入总表
```

```
                date_index = list(final_form['date']).index(date_str)
                # print(date_index)
                final_form[number_name][date_index] = all_prices[row]
                break

        print(f"Future: {number_name}\nYear: {year}\nMonth: {month}")


# read_156form("data/rawData/2019/1.xlsx", '2019', '1')
# print(final_form)
# final_form.to_csv("test.csv")
# exit()

def read_47form(file_path, year, number_name):
    form = pd.read_excel(file_path, skiprows=1)
    for month in ['01', '02', '03', '04', '05', '06', '07', '08', '09', '10', '11', '12']:

        all_contracts = list(form['品种代码'].values) if int(year) < 2021 else
            list(form['合约代码'].values)
        all_prices = list(form['今收盘'].values)
        all_dates = list(form['交易日期'].values)
        all_amounts = list(form['成交量(手)'].values)

        # 找到主力合约
        for day in range(1, 32):
            day = str(day)
            if len(day) == 1:
                day = '0' + day
            # print(day)
            indexes = [_ for _ in range(len(all_dates)) if str(all_dates[_])[-5:] ==
                f"{month}-{day}"]
            # print(indexes)
            if indexes:
                max_index = all_amounts.index(max([all_amounts[_] for _ in indexes]))
                if max_index:
                    break
                else:
                    continue

        max_contract = all_contracts[max_index]

        # 只留下主力合约者
        valid_indexes = [_ for _ in range(len(all_contracts)) if str(all_dates[_])[:7] ==
            f"{year}-{month}" and all_contracts[_] == max_contract]
        all_contracts = [all_contracts[_] for _ in valid_indexes]
        all_prices = [all_prices[_] for _ in valid_indexes]
        all_dates = [all_dates[_] for _ in valid_indexes]
        all_amounts = [all_amounts[_] for _ in valid_indexes]


        for _ in range(len(all_contracts)):
            date_index = list(final_form['date']).index(all_dates[_].replace("-", ""))
            final_form[number_name][date_index] = all_prices[_]

        print(f"Future: {number_name}\nYear: {year}\nMonth: {month}")


for future in [('hc', 3), ('rb', 0), ('wr', 8), ('ss', 2)]:
    for year in ['2020', '2021', '2022']:
        for month in ['01', '02', '03', '04', '05', '06', '07', '08', '09', '10', '11', '12']:
            read_0238form(f"data/rawData/{year}/0238/所内合约行情报表{year}.{int(month) if
                int(year) != 2021 else month}.xls", year, month, future[0], str(future[1]))
            print(f"Future: {future}\nYear: {year}\nMonth: {month}\n")
```

```
# read_47form("data/rawData/2019/4.xls", '2019', '4')
# read_47form("data/rawData/2019/7.xls", '2019', '7')
read_47form("data/rawData/2020/4.xls", '2020', '4')
read_47form("data/rawData/2020/7.xls", '2020', '7')
read_47form("data/rawData/2021/4.xls", '2021', '4')
read_47form("data/rawData/2021/7.xls", '2021', '7')
read_47form("data/rawData/2022/4.xls", '2022', '4')
read_47form("data/rawData/2022/7.xls", '2022', '7')

# read_156form("data/rawData/2019/1.xlsx", '2019', '1')
# read_156form("data/rawData/2019/5.xlsx", '2019', '5')
# read_156form("data/rawData/2019/6.xlsx", '2019', '6')
read_156form("data/rawData/2020/1.xlsx", '2020', '1')
read_156form("data/rawData/2020/5.xlsx", '2020', '5')
read_156form("data/rawData/2020/6.xlsx", '2020', '6')
read_156form("data/rawData/2021/1.xlsx", '2021', '1')
read_156form("data/rawData/2021/5.xlsx", '2021', '5')
read_156form("data/rawData/2021/6.xlsx", '2021', '6')
read_156form("data/rawData/2022/1.xlsx", '2022', '1')
read_156form("data/rawData/2022/5.xlsx", '2022', '5')
read_156form("data/rawData/2022/6.xlsx", '2022', '6')

final_form.to_excel("test.xlsx")
exit()
```

2.2 过滤掉节假日

```
import pandas as pd
import numpy as np

form = pd.read_excel("test.xlsx", index_col=0)

date_list = list(form['date'].values)
future_data = [list(form[str(_)].values) for _ in range(9)]

remove_indexes = []

for (i, column) in enumerate(future_data):
    for (j, item) in enumerate(column):
        form[str(i)][j] = float(str(item).replace(',', ''))
    form[str(i)].astype(float)

for row_index in range(len(date_list)):
    # print([future_data[i][row_index] for i in range(9)])
    if np.sum([float(str(future_data[i][row_index]).replace(',', '')) for i in range(9)]) == 0:
        remove_indexes.append(row_index)

form = form.drop(remove_indexes, axis=0)

form.to_excel("filtered.xlsx")
```

2.3 对缺失值插值

```
import pandas as pd

form = pd.read_excel("filtered.xlsx", index_col=0)

date_list = list(form['date'].values)
```

```
future_data = [list(form[str(_)].values) for _ in range(9)]

def interpolate_time_series(data):
    """Linearly interpolate missing values in a time series data list."""
    for i in range(len(data)):
        if data[i] == 0:
            # Find the next non-zero value
            j = i + 1
            while j < len(data) and data[j] == 0:
                j += 1
            if j == len(data):
                # No non-zero value found, so cannot interpolate
                break
            # Linearly interpolate between the two non-zero values
            x1, y1 = i, data[i-1]
            x2, y2 = j, data[j]
            slope = (y2 - y1) / (x2 - x1 + 1)
            intercept = y1 - slope * x1
            for k in range(i, j):
                data[k] = slope * (k+1) + intercept
    return data

for (i, column) in enumerate(future_data):
    form[str(i)] = interpolate_time_series(column)

form.to_excel("interpolated.xlsx")
```

附录 C　问题二二次型模型矩阵求解程序

3.1　制作原始数据平方列

```
import pandas as pd
import numpy as np

form = pd.read_excel("interpolated.xlsx", index_col=0)

date_list = list(form['date'].values)
future_data = [form[str(_)].values for _ in range(9)]
squared_data = [column * column for column in future_data]



for _ in range(9):
    form[f'{_} squared'] = squared_data[_]

form.to_csv("nonLinear.xlsx")
```

3.2　计算矩阵系数

```
import numpy as np
from scipy.optimize import minimize
import pandas as pd
from matplotlib import pyplot as plt

form = pd.read_excel("interpolated.xlsx", index_col=0)
future = [form[str(_)].values for _ in range(9)]

standardized_form = pd.read_excel("standardized.xlsx", index_col=0)
```

```
date_list = list(standardized_form['date'].values)
standardized_future = [standardized_form[str(_)].values for _ in range(9)]
# for i in range(9):
#     max_value = np.max(future_data[i])
#     min_value = np.min(future_data[i])
#     form[str(i)] = (form[str(i)] - min_value) / (max_value - min_value)
# form.to_excel("standardized.xlsx")

paras0 = np.ones(45)

# 45 length list to matrix
def to_matrixA(a):
    return np.array([
        [a[0], a[1], a[2], a[3], a[4], a[5], a[6], a[7], a[8]],
        [a[1], a[9], a[10], a[11], a[12], a[13], a[14], a[15], a[16]],
        [a[2], a[10], a[17], a[18], a[19], a[20], a[21], a[22], a[23]],
        [a[3], a[11], a[18], a[24], a[25], a[26], a[27], a[28], a[29]],
        [a[4], a[12], a[19], a[25], a[30], a[31], a[32], a[33], a[34]],
        [a[5], a[13], a[20], a[26], a[31], a[35], a[36], a[37], a[38]],
        [a[6], a[14], a[21], a[27], a[32], a[36], a[39], a[40], a[41]],
        [a[7], a[15], a[22], a[28], a[33], a[37], a[40], a[42], a[43]],
        [a[8], a[16], a[23], a[29], a[34], a[38], a[41], a[43], a[44]]])

# x is an array of length 9
def my_func(params, x: np.ndarray):
    z_pred = np.matmul(x.T, to_matrixA(params))
    z_pred = np.matmul(z_pred, x)
    return z_pred

def grad(params, x: np.ndarray):
    return np.matmul(x.T, to_matrixA(params))

xs = np.array([[standardized_future[i][_] for i in range(9)] for _ in range(len(date_list))])

# Define your cost function
def my_cost(params, xs, zs):
    z_preds = np.array([my_func(params, _) for _ in xs])
    return np.sum((z_preds - zs)**2)

# res = minimize(my_cost, paras0, args=(xs, np.ones(len(date_list))))

# Print the optimal value of parameter a
# print(res.x)

result = [50.02810444, -2.88874897, 3.49922088, -56.84832229, -30.7314123,
 -20.8074924, 22.00408563, 19.33944626, 4.14644399, 8.16506352,
   2.11352252, -5.3434172, -15.06910263, 12.63667188, -9.30170597,
  15.6336573, -1.2266852, 6.03054014, -9.0089566, -3.59248482,
 -10.34928899, 7.65706778, 1.32693748, 2.08916196, 76.06796073,
 42.63822509, 13.44637666, -12.75342873, -30.88852845, -6.61955562,
 36.19035499 , 2.50174033, -1.83327526, -37.37986317, 1.44348714,
18.03212637, -14.09458582, 19.91511781, -8.15505718, 14.51482134 ,
-17.12362909, 0.1553415, 33.50251549, 2.19701824, 4.81280929]
# print(to_matrixA(result))
# loss = [abs(my_func(result, row)-1)**2 for row in xs]
# form['loss'] = loss
# form.to_excel('loss.xlsx')
# plt.plot(loss)
# plt.xlabel('time')
# plt.ylabel('loss')
# plt.title('time-loss')
# plt.show()
```

附录 D　问题三可视化与机器学习程序

4.1　热力图绘制

```
from matplotlib import pyplot as plt
import numpy as np

plt.rcParams['font.sans-serif'] = ['Simhei'] #显示中文
plt.rcParams['axes.unicode_minus'] = False #显示负号

data = np.array([[0.192, 0.373, 0.228, 0.232, 0.197, 0.168, 0.133, 0.1, 0.075, 0.182],
 [0.447, 0.474, 0.404, 0.292, 0.276, 0.315, 0.231, 0.201, 0.186, 0.16],
 [0.197, 0.314, 0.215, 0.176, 0.223, 0.206, 0.208, 0.08, 0.092, 0.195],
 [0.227, 0.352, 0.212, 0.206, 0.264, 0.128, 0.212, 0.068, 0.043, 0.155],
 [0.526, 0.471, 0.418, 0.31, 0.292, 0.347, 0.233, 0.175, 0.189, 0.188],
 [0.497, 0.497, 0.442, 0.464, 0.309, 0.294, 0.235, 0.156, 0.146, 0.198],
 [0.481, 0.475, 0.473, 0.38, 0.299, 0.331, 0.271, 0.243, 0.231, 0.225],
 [0.58, 0.647, 0.48, 0.432, 0.366, 0.328, 0.323, 0.22, 0.211, 0.191],
 [0.452, 0.496, 0.558, 0.436, 0.341, 0.351, 0.296, 0.241, 0.237, 0.201],
 [0.569, 0.608, 0.51, 0.503, 0.335, 0.329, 0.316, 0.313, 0.219, 0.166]]
)

data2 = np.array([[0.275, 0.252, 0.335, 0.457, 0.348, 0.525, 0.29, 0.287, 0.38, 0.481],
    [0.532, 0.446, 0.394, 0.516, 0.57, 0.657, 0.377, 0.488, 0.377, 0.51], [0.328, 0.255,
    0.206, 0.238, 0.424, 0.565, 0.315, 0.266, 0.299, 0.505], [0.489, 0.302, 0.265, 0.297,
    0.455, 0.744, 0.389, 0.255, 0.478, 0.555], [0.618, 0.444, 0.509, 0.489, 0.591, 0.831,
    0.481, 0.504, 0.51, 0.527], [0.754, 0.676, 0.636, 0.596, 0.542, 0.726, 0.5, 0.772, 0.638,
    0.721], [0.526, 0.351, 0.576, 0.538, 0.512, 0.615, 0.489, 0.594, 0.527, 0.567], [0.722,
    0.709, 0.616, 0.639, 0.622, 0.663, 0.726, 0.773, 0.593, 0.725], [0.456, 0.431, 0.605,
    0.542, 0.497, 0.654, 0.516, 0.803, 0.648, 0.684], [0.426, 0.607, 0.564, 0.624, 0.6, 0.549,
    0.681, 0.81, 0.529, 0.729]])

fig, ax = plt.subplots()

# Create the heatmap
plt.imshow(data2)

# Add a colorbar
plt.colorbar()

plt.yticks(np.arange(0, 10), [1, 2, 3, 5, 10, 15, 20, 25, 30, 35])
plt.xticks(np.arange(0, 10), np.arange(1, 11))

plt.xlabel(f'交易间隔')
plt.ylabel(f'预测步数')
plt.title(f'预测步数-交易间隔对盈利的影响(固定买入份额)')

# Show the plot
plt.show()


# plt.clf()

# fig, ax = plt.subplots()

# table = ax.table(cellText=data, loc='center',
#                  rowLabels=np.arange(10),
#                  colLabels=np.arange(10))
# table.get_celld()[(0, 0)].get_text().set_text("Top Left Corner")
# table.auto_set_font_size(False)
# table.set_fontsize(14)

# ax.axis('off')
```

```
# plt.show()
```

4.2 图片拼合

```
from PIL import Image

grid_size = (1100, 1650)
grid_image = Image.new('RGB', grid_size)

for (_, path) in enumerate(['78', '02', '03', '04', '36', '06']):
    img = Image.open(f"figs/{path}.png").resize((550, 550))
    grid_image.paste(img, ((_ // 3) * 550, (_ % 3) * 550))


grid_image.save('grid.jpg')
```

附录 E 支持向量机

```
import numpy as np
from scipy.optimize import minimize
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.svm import SVC # "Support vector classifier"

form = pd.read_excel("loss.xlsx", index_col=0)
date_list = list(form['date'].values)
future_data = [form[str(_)].values for _ in range(9)]
loss_list = form['loss'].values
loss_color = ['green' if loss < 0.1 else 'red' for loss in loss_list]

plt.rcParams['font.sans-serif'] = ['Simhei'] #显示中文
plt.rcParams['axes.unicode_minus'] = False #显示负号

def plot_svc_decision_function(model, ax=None, plot_support=True):
    """Plot the decision function for a 2D SVC"""
    if ax is None:
        ax = plt.gca()
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()

    # create grid to evaluate model
    x = np.linspace(xlim[0], xlim[1], 30)
    y = np.linspace(ylim[0], ylim[1], 30)
    Y, X = np.meshgrid(y, x)
    xy = np.vstack([X.ravel(), Y.ravel()]).T
    P = model.decision_function(xy).reshape(X.shape)

    # plot decision boundary and margins
    ax.contour(X, Y, P, colors='k',
               levels=[-1, 0, 1], alpha=0.5,
               linestyles=['--', '-', '--'])

    # plot support vectors
    if plot_support:
        ax.scatter(model.support_vectors_[:, 0],
                   model.support_vectors_[:, 1],
                   s=300, linewidth=1, facecolors='none')
```

```
    ax.set_xlim(xlim)
    ax.set_ylim(ylim)

name_list = ["螺纹钢", "铁矿石", "不锈钢", "热轧卷板", "硅铁", "焦煤", "焦炭", "锰硅", "线材"]

def plot_and_save(index1, index2):
    model = SVC(kernel='linear', C=1E10)
    model.fit(np.array([(future_data[index1][_], future_data[index2][_]) for _ in
        range(len(date_list))]), loss_color)

    plot_svc_decision_function(model)
    plt.scatter(future_data[index1], future_data[index2], c=loss_color)
    plt.xlabel(f'{name_list[index1]}')
    plt.ylabel(f'{name_list[index2]}')
    plt.title(f'{name_list[index1]} - {name_list[index2]}')
    plt.xlim(0, 1)
    plt.ylim(0, 1)
    plt.savefig(f"figs/{index1}{index2}.png")

if __name__ == "__main__":
    A = [0, 1, 2, 3, 4, 5, 6, 7, 8]
    B = []
    for i in range(len(A)):
        for j in range(i + 1, len(A)):
            B.append((A[i], A[j]))

    # print(B)

    for tu in B:
        plot_and_save(tu[0], tu[1])
        plt.clf()
```

5.1 局部线性嵌入算法

```
import matplotlib as mpl
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import proj3d
import pandas as pd
import numpy as np
# import paintClassify

plt.rcParams['font.sans-serif'] = ['Simhei'] #显示中文
plt.rcParams['axes.unicode_minus'] = False #显示负号

#Generate mainfold data set
from sklearn.datasets import make_swiss_roll

#Generate data
# X, t = make_swiss_roll(n_samples=1000, noise=0.2, random_state=41)

form = pd.read_excel("loss.xlsx", index_col=0)
date_list = list(form['date'].values)
future_data = [form[str(_)].values for _ in range(9)]
loss_list = np.sqrt(np.sqrt(form['loss'].values))
# loss_color = ['green' if loss < 0.1 else 'red' for loss in loss_list]
xs = np.array([[future_data[i][_] for i in range(9)] for _ in range(len(date_list))])

from sklearn.manifold import LocallyLinearEmbedding

lle = LocallyLinearEmbedding(n_components=2, n_neighbors=10, random_state=42)
X_reduced = lle.fit_transform(xs)
plt.title("九种期货价格 LLE 平面嵌入图", fontsize=14)
```

```
# model = paintClassify.SVC(kernel='linear', C=1E10)
# model.fit(X_reduced, loss_color)
# paintClassify.plot_svc_decision_function(model)
# print(X_reduced)
# exit()
plt.scatter(X_reduced[:, 0], X_reduced[:, 1], c=loss_list, cmap=plt.cm.hot)
plt.xlabel("$z_1$", fontsize=18)
plt.ylabel("$z_2$", fontsize=18)
# plt.axis([-0.065, 0.055, -0.1, 0.12])
plt.grid(True)

# save_fig("lle_unrolling_plot")
plt.show()
```