队伍编号	903900
题号	（D）

脱氧合金化过程的收得率预测与成本优化

摘 要

本文参考附件数据并查阅大量资料、数据，构建了相应的数学模型，对脱氧合金化过程中的合金收得率、合金收得率的影响因素、成本优化等问题进行了研究。

对于问题一，从附件 1 中读取大量的脱氧合金化过程的相关数据，通过转炉终点值和连铸正样值，计算出被钢水吸收的合金元素的重量，再根据合金成分说明与加入合金的具体数值，计算出某种元素加入钢水的总重量，从而得出这种元素的历史收得率。考虑到温度、连铸正样值、转炉终点值、加入合金料的种类及重量等因素，计算它们与收得率曲线的相关系数，分析出影响 C、Mn 收得率的主要因素。

对于问题二，我们考虑可能影响收得率的因素，提出线性模型进行收得率预测，在求解过程中，我们先使用梯度下降法，训练出一个准确度较高的模型，再使用模拟退火法进行优化，避免产生局部解，从而提高预测准确率，得到一个更准确的预测模型。

对于问题三，针对不同合金料的价格及熔炼效果，我们提出一个多元线性规划模型，在问题二的收得率预测模型基础上，尽可能地使成本降低，同时不违反钢类标准，为了求解该模型，我们引入了引入 G.B.Dantzig 提出的单纯形法（Simplex Method），在可行解空间内找到使得成本最低的一组解。

对于问题四，结合问题二完成的收得率预测模型，以及问题三针对合金化过程求解最优成本，以及相应配料方案的结果，总结炼钢厂在历史炼钢过程中存在的缺点，向炼钢厂领导提出合理建议，修改车间的合金化配料模式，从而提升炼钢厂的经营利润。

关键词：多变量梯度下降法，模拟退火法，多元线性规划，单纯形法

目录

一、问题重述

炼钢过程中的脱氧合金化是钢铁冶炼过程中的重要工艺环节，对于不同的钢种在熔炼结束时，需加入不同量、不同种类的合金，以使其所含合金元素达标，最终使得成品钢在某些物理性能上达到特定要求。随着钢铁行业中的高附加值钢种产量的不断提高，如何通过历史数据对脱氧合金化环节建立数学模型，在线预测并优化投入合金的种类与数量，在保证钢水质量的同时，最大程度地降低合金钢的生产成本，是各大钢铁产业提高竞争力所要解决的重要问题。

问题一：钢水脱氧合金化主要关注 C、Mn、S、P、Si 五种元素的含量，请根据附件 1 计算 C、Mn 两种元素的历史收得率，并分析影响其收得率的主要因素。

问题二：在问题 1 的基础上，构建数学模型，对 C、Mn 两种元素收得率进行预测，并进一步改进模型及算法，尽可能提高这两种元素收得率的预测准确率。

问题三：不同合金料的价格不同，其选择直接地影响钢水脱氧合金化的成本，请根据问题 2 中合金收得率的预测结果及附件 2，建立数学模型，实现钢水脱氧合金化成本优化计算，并给出合金化配料方案。

问题四：请根据你们的研究结果，给炼钢厂领导写一封建议信（一页以内）。

二、问题假设

1. 考虑脱氧合金化过程中导致的吹损、挥发等过程，以及化学反应，我们假设反应后的钢水损耗率设为 2%。
2. 不考虑合金料之间的相互作用。
3. 假设合金料在钢水中均匀分布，充分反应。

三、符号说明

符号	意义
Record	脱氧合金化的相关记录数据
IndexOfIron	钢号种类

Element	元素种类
IronWaterValue	铁水净重
NameOfMetal	合金种类名称
Temperature	钢水温度
ElementValueChange	合金化前后元素含量变化
MetalValue	合金化过程投入的合金原料量
OtherElements	合金化过程中非主要元素的变化量
alpha	梯度下降学习率
η	铁水损失率
Begin(i, type)	type 元素在第 i 个数据中的转炉终点
End(i, type)	type 元素在第 i 个数据中的连铸正样
putMetal(i, j)	第 j 个记录中，第 i 类合金的加入量
Percent(i, j)	第 i 种合金中，第 j 类元素的占比
GainRate	合金收得率
Correlation	相关性系数
Cost	合金料单价
W	合金料总成本

四、问题分析

4.1 问题一的分析

为了计算 C、Mn 收得率，我们首先根据附件 1 提供的信息，计算出被钢水吸收的合金元素的重量，以及合金料所提供的该元素的总的重量，得到附件 1 中每次反应过程的合金元素收得率。考虑温度、合金料成分和含量、脱氧剂等影响因素，我们计算出这些变量序列与收得率间的相关系数，分析原因并总结出影响收得率的主要因素。

4.2 问题二的分析

在问题一的基础上，我们根据收得率的影响因素，提出了线性回归模型来进行收得率的预测，根据梯度下降法来训练模型，并使用交叉验证的方法进行模型评估。为了改进模型，本文采用模拟

退火法，弥补了梯度下降法容易产生局部解的缺陷，进一步提高模型的准确度。

4.3 问题三的分析

考虑到合金料的成本问题，我们希望能提供一个方案，在满足钢类指标的同时，尽可能地降低成本，综合本题条件，我们提出了多元线性规划模型，以合金化整体投入成本作为规划目标函数，元素含量标准条件作为规划约束，实现合金化成本优化计算，并给出合金配料方案。

4.4 问题四的分析

结合前述问题的求解过程与实验结果，向炼钢厂领导提出合理意见。

五、问题一模型的建立与求解

5.1 数据预处理

为了计算 C、Mn 元素的历史得出率，我们需要从附件中读取的数据有：C、Mn 反应前后在钢水中的含量：转炉终点百分比含量与连铸正样百分比含量，以及加入合金料的量和各类合金料的成分含量。

STEP1: 读入数据——利用 python 从附件 1 和附件 2 中读取相关数据；

STEP2: 数据分类——根据钢号将数据进行划分；

STEP3: 去除异常——将数据中含缺省值 null 或为空的记录删除，只保留符合规范的数据；

5.2 计算历史收得率

合金收得率是指脱氧合金化过程中被钢水吸收的合金元素的重量与加入该元素的总重量之比。在第 i 个记录中，type 类元素在反应前的重量 Before_reflect_weight 和反应后的重量 After_reflect_weight 可以表示为：

$$Before\_reflect\_weight = Begin(i,element) * IronWaterValue$$

$$After\_reflect\_weight = End(i,element) * IronWaterValue * (1-\eta)$$

添加合金料中，type 类元素的总重量可以表示为：

$$Metal\_Contain\_Weight = \sum addMetal(i, metal) * Percent(metal, element)$$

根据收得率计算公式，第 i 个记录的 type 类元素的收得率为：

$$GainRate(i, element) = \frac{After\_reflect\_weight - Before\_reflect\_weight}{Metal\_contains\_weight}，element \in \{C, Mn\}$$

对于附件 1 中包含的历史熔炼数据，计算出每条记录的 C、Mn 元素的收得率，收得率曲线绘制如下：

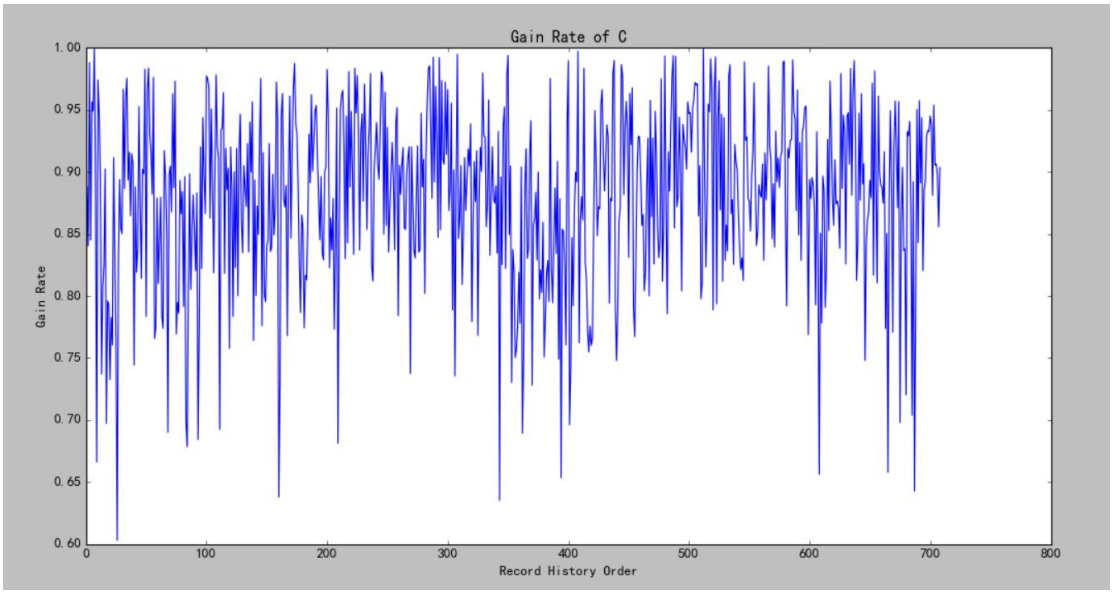

图 1. C 元素的历史收得率

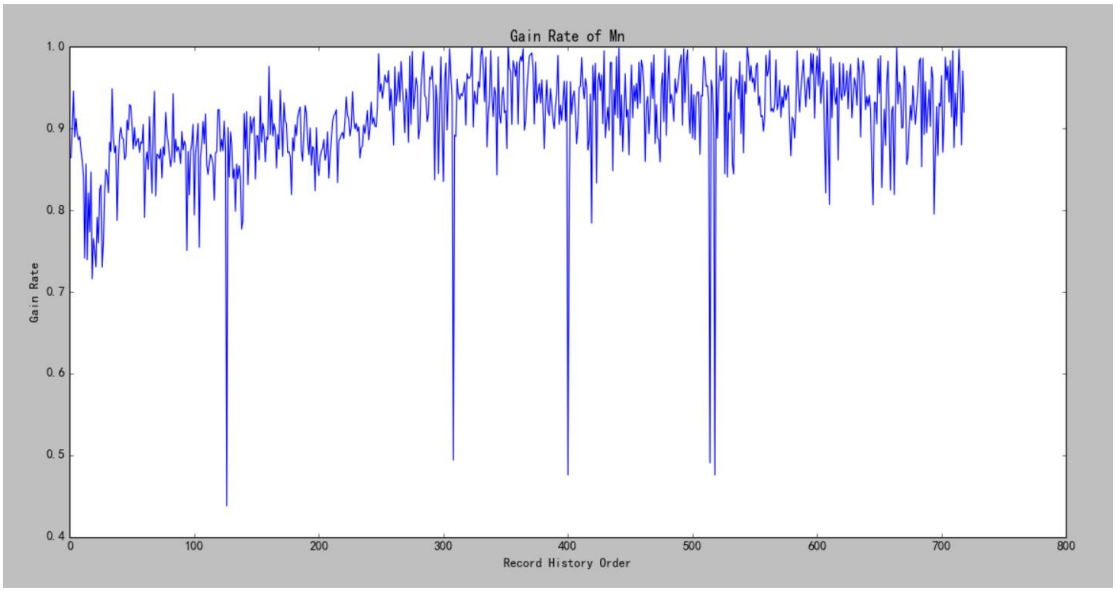

图 2. Mn 元素的历史收得率

计算 C、Mn 历史收得率平均值为:

The Mean History GainRate of C: 0.880013

The Mean History GainRate of Mn: 0.912330

5.3 相关度分析

考虑可能影响收得率的因素：包括转炉温度，反应前后铁水中某元素含量的变化值，合金料和脱氧剂的添加量等，分析它们与收得率 GainRate 之间的关系。

对每种影响因素，通过它们的变化曲线和收得率曲线，计算它们与收得率之间的协方差:

$$Cov（X，GainRate）=\frac{\sum_{i=1}^{n}（X_i-E（X））（GainRate_i-E（GainRate））}{n-1}$$

当 $Cov(X,GainRate)>0$，说明 X 和 GainRate 之间正相关；

当 $Cov(X,GainRate)<0$，说明 X 和 GainRate 之间负相关；

当 $Cov(X,GainRate)=0$，说明 X 和 GainRate 之间不相关；

为了得到主要的相关因素，我们还需要计算出这些影响因素与收得率之间的相关性系数 r，从而根据相关系数的大小，决定主要的影响因素：

$$r_{X,GainRate}=\frac{Cov(X,GainRate)}{s_X s_{GainRate}}$$

其中，

$$s_{GainRate}=\sqrt{\frac{\sum(GainRate_i-E(GainRate))^2}{n-1}}$$

$$s_X=\sqrt{\frac{\sum(X_i-E(X))^2}{n-1}}$$

结合附件一给出的数据段集合，此处我们对转炉温度，合金料和脱氧剂的添加量分别与对应元素的收得率进行相关性系数求解，所得因素中相关性系数排行前十的结果如下：

影响因素	与 C 元素收得率的相关性系数 r
钢水重量	+0.420648

转炉终点 C	-0.351624
Ceq_Val	+0.251727
硅铝钙	+0.213523
硅钙碳脱氧剂	-0.123324
Ala_Val	-0.115656
Als_Val	-0.115588
钒铁（FeV50-B）	+0.113319
锰硅合金	-0.108275
温度	+0.093333

影响因素	**与 Mn 元素收得率的相关性系数 r**
转炉终点 Mn	-0.484269
钢水重量	+0.278014
氮化钒铁	+0.212603
锰硅合金 FeMn68Si18	-0.143222
锰硅合金 FeMn64Si27	-0.137280
硅锰面	-0.134205
Ceq_Val	+0.119235
Cu_Val	+0.116352
V_Val	+0.113161
碳化硅	-0.099049

通过计算 C、Mn 元素收得率与列举出的多种因素的相关性系数，我们可以推出影响收得率因素的一般性结论：

1、脱氧合金化过程中，温度对收得率具有一定的影响，适宜温度可以提高反应的剧烈程度，使得合金化过程能充分进行；

2、合金料的投入决定了脱氧合金化过程的投入，直接影响收得率；

3、转炉温度和连铸正样决定了脱氧合金化过程的产量，直接影响收得率；

4、脱氧剂碳化硅能抑制氧化过程，防止合金化过程产生的金属物质被氧化而损耗，间接地

对收得率产生影响；

六、问题二模型的建立与求解

6.1 预测模型的分析

在问题一的基础上，我们考虑了影响收得率的主要因素：（1）温度、（2）转炉温度和连铸正样、（3）铁水重量、（4）合金料投入、（5）其余物质。 基于实验数据，提出了线性预测模型:

$$GainRate（i,element）= w0 * CostTemp + w1 * CostPotency + w2 * CostWater + w3 * CostMetal + w4 * CostOther + C$$

特别地：

$$CostTemp = Temperature（i）$$
$$CostMetal = \sum addMetal（i，element）* Percent（i，element）$$
$$CostPotency = End（i，element）- After（i，element）$$
$$CostWater = IronWaterValue（i）$$

6.2 模型求解

6.2.1 多元梯度下降法

为使预测结果准确率得以保障，我们结合梯度下降法，训练模型的权重参数，得到预测模型。通常地，一个函数的梯度方向是其增加最快的地方，而负梯度方向就是函数下降最快的方向。据此性质，我们可以通过沿负梯度方向寻找使得损失函数最小化的一组权重 $\{\theta\}$。

在训练过程中，我们以预测值 h 和真实值 y 之间的欧式距离作为损失函数 J：

$$J(\theta_0,\theta_1,...,\theta_k) = \sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2$$

梯度下降过程的具体形式如下：

$$\theta^{(\tau+1)} = \theta^{(\tau)} + aplha * \frac{1}{n} * \sum(y_i - \theta^{(\tau)^T} x_i)\ x_i$$

具体过程见附件程序，我们得到损失函数的收敛曲线，取得了一个较好的结果（见图 7），同时，C 元素和 Mn 元素的线性回归预测模型权重分布如下（见表 1，表 2）。

W0		0.000024
W1		0.123191
W2		0.033487
W3	W3_0	-0.011675
	W3_1	-0.000698
	W3_2	0.000000
	W3_3	-0.001519
	W3_4	0.000093
	W3_5	-0.000367
	W3_6	-0.003215
	W3_7	-0.002240
	W3_8	0.000000
	W3_9	-0.010388
	W3_10	0.000021
	W3_11	0.001327
	W3_12	-0.105913
	W3_13	-0.021608
	W3_14	-0.026586
	W3_15	-0.076590
W4	W4_0	0.000178
	W4_1	0.000527
	W4_2	-0.000805
	W4_3	-0.000332
	W4_4	0.000147
	W4_5	-0.000701
	W4_6	0.000281

	W4_7	0.000036
	W4_8	0.880013

表 1. C 元素收得率线性回归预测模型的权重分布

W0		0.000790
W1		0.042358
W2		0.030894
W3	W3_0	-0.000690
	W3_1	-0.005676
	W3_2	0.000000
	W3_3	-0.017868
	W3_4	-0.001692
	W3_5	-0.006008
	W3_6	-0.001003
	W3_7	-0.003099
	W3_8	0.000000
	W3_9	-0.037010
	W3_10	0.000007
	W3_11	-0.000019
	W3_12	-0.001790
	W3_13	-0.084666
	W3_14	-0.104058
	W3_15	-0.000027
W4	W4_0	-0.001599
	W4_1	-0.000419
	W4_2	0.001269
	W4_3	-0.001195
	W4_4	0.010252
	W4_5	0.000727
	W4_6	-0.001001

	W4_7	-0.000878
	W4_8	0.912330

表 2. Mn 元素收得率线性回归预测模型的权重分布

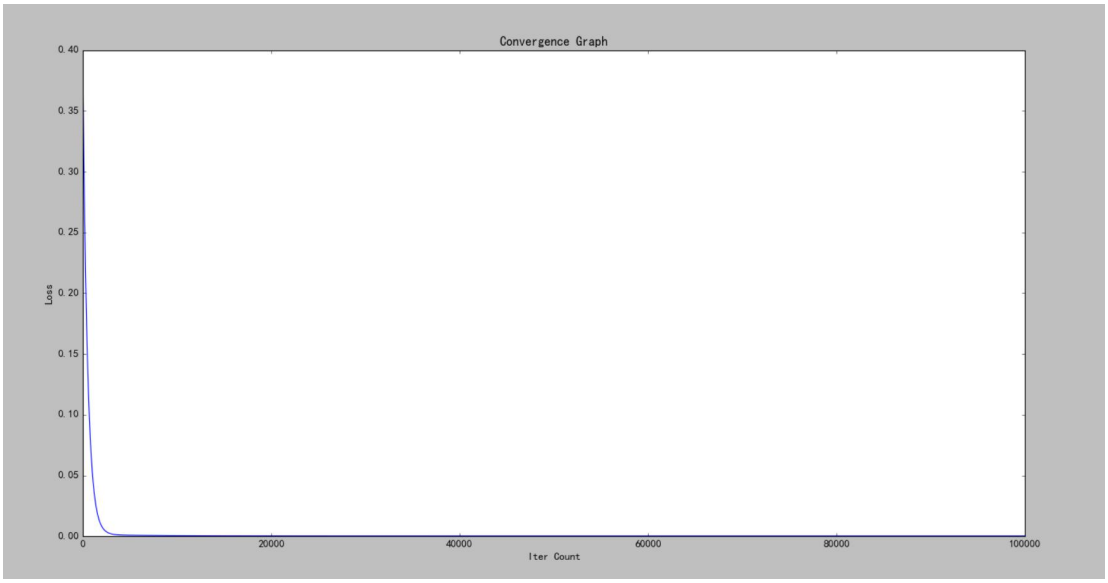

图 7. 损失函数的收敛曲线

总迭代步数为 100000 步，损失函数的大约在 2000 步左右就已经收敛完毕，说明了模型的准确性与梯度下降算法的预测可行性。

6.2.2 梯度下降模型验证

交叉验证(Cross Validation)是用于验证预测模型在广泛程度上的准确率的有效方法，通常的做法是：将原始数据中的大部分作为训练集，其余部分作为测试集，训练集输入将预测模型训练完毕，然后根据此模型在测试集上进行预测，最终获取其在测试集上的准确率。优点如下：

1. 避免了全部数据当做训练集，最终正确率的计算依然建立在训练集的情况发生，更加全面地验证预测模型的泛化预测能力。

2. 验证过程简便，相对容易实现。

K 折交叉验证(K-Fold Cross Validation)将原始数据分为 K 组，每次选取其中的 K-1 组作为训练集，剩余部分作为测试集，循环 K 次，每一次迭代选取的训练集和测试集均不相同，输出每次的预测准确率，最终求取整体交叉验证的平均准确率。

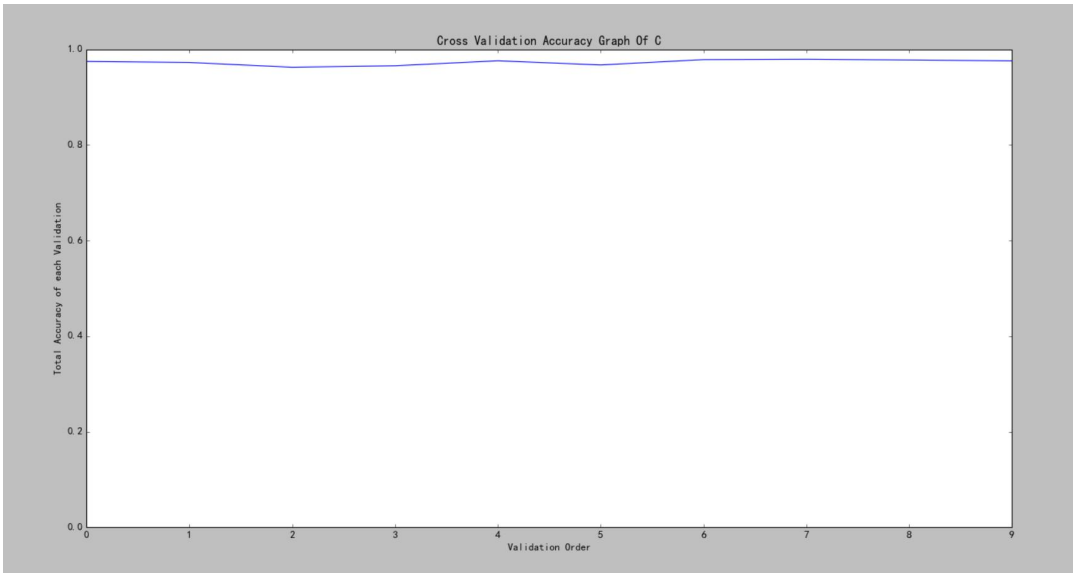

图 7. C 元素的 K-Fold 交叉验证预测准确率折线图，其中 K=10

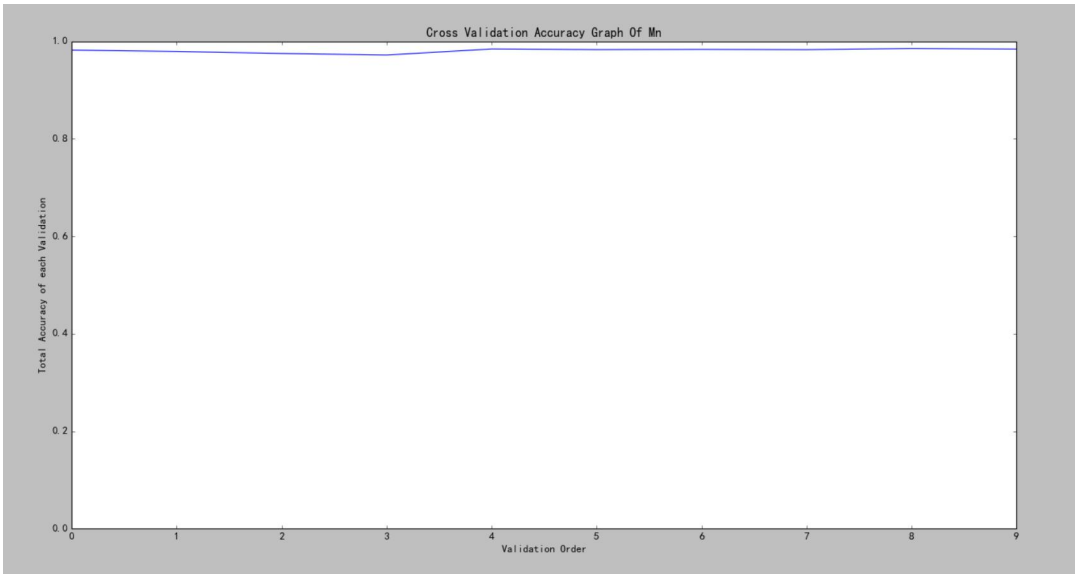

图 7. Mn 元素的 K-Fold 交叉验证预测准确率折线图，其中 K=10

图 7 和图 7 是 C、Mn 元素收得率在 K-Fold 交叉验证过程中的准确性折线图，在验证过程中，它们的精确度始终趋近于 1，客观上说明了预测模型的准确性。

```
The 0th K-Fold Prediction Accuracy Result: 0.974794
The 1th K-Fold Prediction Accuracy Result: 0.972653
The 2th K-Fold Prediction Accuracy Result: 0.962656
The 3th K-Fold Prediction Accuracy Result: 0.965747
The 4th K-Fold Prediction Accuracy Result: 0.976168
The 5th K-Fold Prediction Accuracy Result: 0.967432
The 6th K-Fold Prediction Accuracy Result: 0.978593
The 7th K-Fold Prediction Accuracy Result: 0.979370
The 8th K-Fold Prediction Accuracy Result: 0.977786
The 9th K-Fold Prediction Accuracy Result: 0.976028
Mean Accuracy of Cross Validation  -- C: 0.973123
```

```
The 0th K-Fold Prediction Accuracy Result: 0.982156
The 1th K-Fold Prediction Accuracy Result: 0.979278
The 2th K-Fold Prediction Accuracy Result: 0.975207
The 3th K-Fold Prediction Accuracy Result: 0.972034
The 4th K-Fold Prediction Accuracy Result: 0.984388
The 5th K-Fold Prediction Accuracy Result: 0.983147
The 6th K-Fold Prediction Accuracy Result: 0.983584
The 7th K-Fold Prediction Accuracy Result: 0.983048
The 8th K-Fold Prediction Accuracy Result: 0.985413
The 9th K-Fold Prediction Accuracy Result: 0.984182
Mean Accuracy of Cross Validation  -- Mn: 0.981244
```

6.3 模型的改进

6.3.1 模拟退火法

在之前的工作中，我们使用梯度下降法训练预测模型，取得了较好的结果，但是梯度下降法具有一个明显的缺陷：容易陷入到局部最优解中。为了避免这一问题，同时进一步提高算法的准确性，我们使用模拟退火法，进行算法的改进。

根据 Metropolis 准则，粒子在温度 T 时趋于平衡的概率为 exp(-ΔE/(kT))，其中 E 为温度 T 时的内能，ΔE 为变化量,k 为 Boltzmann 常数。模拟退火法引入了这一思想，我们设定接纳概率 P，当新解优于当前解时，P=1，必定接受这个解，否则，以 Metroplis 准则平衡概率接受这个解：

$$p=\begin{cases}1 & if\ \ E(new)<E(old)\\ \exp(-\dfrac{E(new)-E(old)}{T}) & if\ \ E(new)\geq E(old)\end{cases}$$

用固体退火模拟组合优化问题，将内能 E 模拟为目标函数值 f，温度 T 演化成控制参数 t，即得到解组合优化问题的模拟退火演算法：由初始解 i 和控制参数初值 t 开始，对当前解重复“产生新解→计算目标函数差→接受或丢弃”的迭代，并逐步衰减 t 值，算法终止时的当前解即为所得近似最优解，这是基于蒙特卡罗迭代求解法的一种启发式随机搜索过程，此处应用于优化线性回归拟合函数的权重设置，与梯度下降法结合，寻求全局最优的参数权重设置向量，具体做法为：

1. 每一轮梯度下降迭代时，保留损失函数的前一轮历史解，计算当前迭代产出的损失函数解，以模拟退火规则对新产生的参数权重向量采取接收或淘汰的行为：

$J(\theta_{i+1})<=J(\theta_i)$，以概率 $p=1$ 接收新解，该新解优于历史解

$J(\theta_{i+1})>J(\theta)$，以概率 $p=(-\dfrac{delta\_E}{T})$ 接收新解，该新解差于历史解

2. 通过模拟退火算法优化梯度下降模型的权重参数，有利于模型设置全局最优的权重向量，从

根本上提高预测准确率。

6.3.2 模型的求解

改进之后，损失函数变化曲线如图 7 所示，相对于改进前的模型，收敛步数有所增加，这是在启发式探索最优解过程中付出的代价。

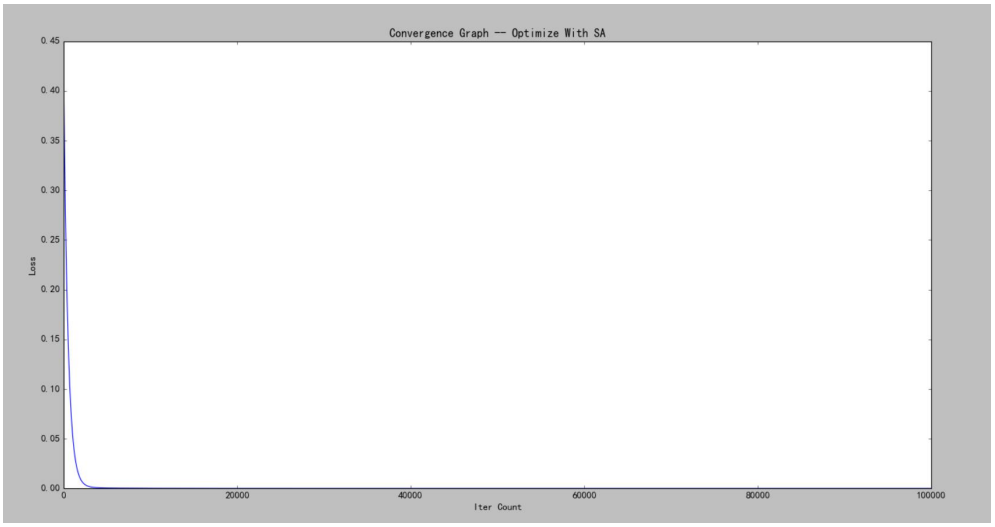

图 8. 采用模拟退火法优化梯度下降模型，所得收敛曲线

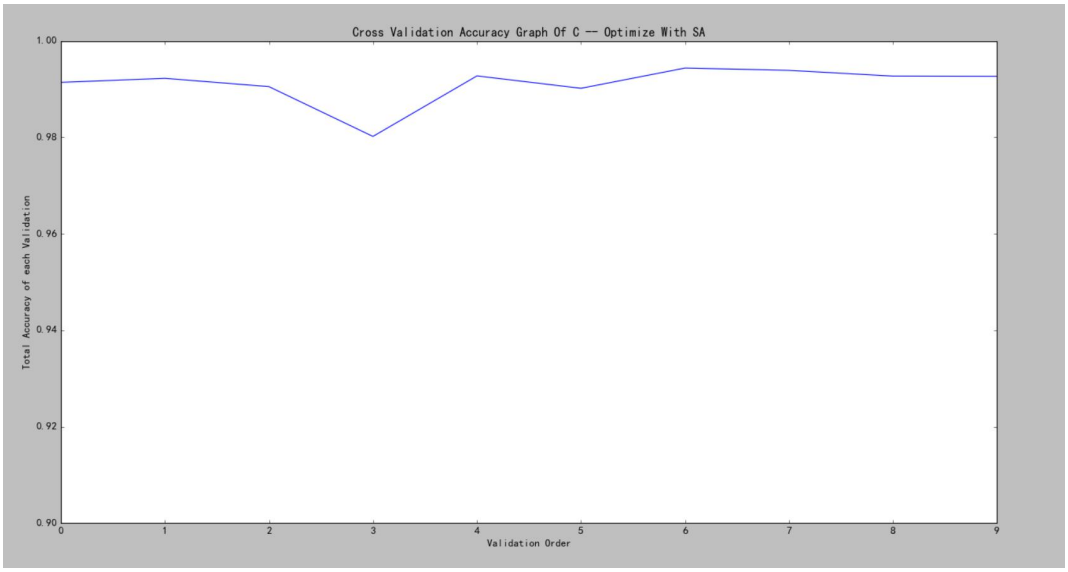

图 9. 采用模拟退火法优化梯度下降模型，C 元素收得率预测准确率曲线

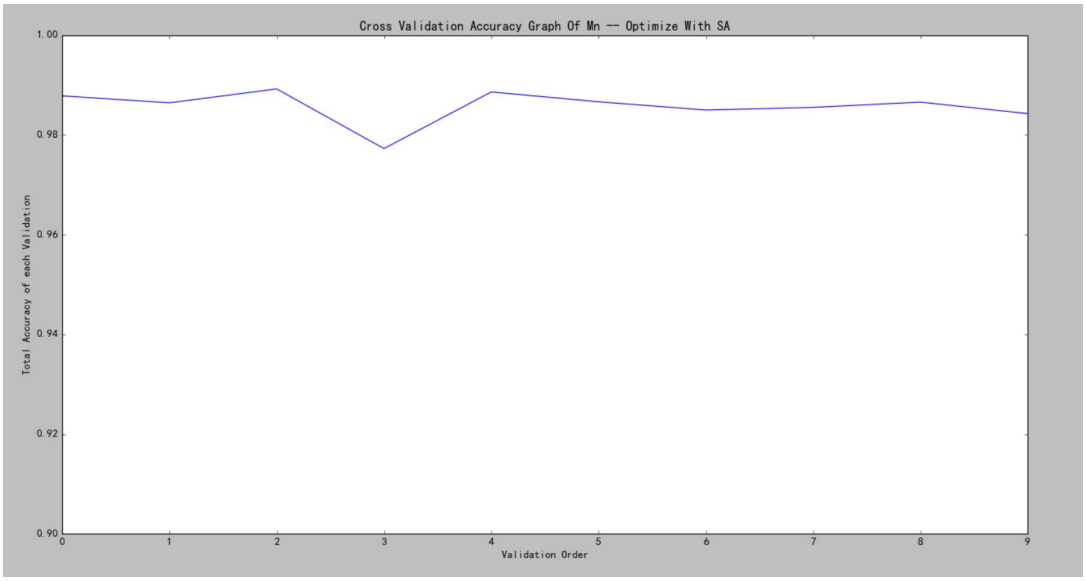

图 10. 采用模拟退火法优化梯度下降模型，Mn 元素收得率预测准确率曲线

在使用模拟退火法之后，模型的准确度平均可达 98.57%，相较之前的模型提高了 2 个百分点，说明了优化方法的有效性。

```
The 0th K-Fold Prediction Accuracy Result -- Optimize With SA: 0.991421
The 1th K-Fold Prediction Accuracy Result -- Optimize With SA: 0.992258
The 2th K-Fold Prediction Accuracy Result -- Optimize With SA: 0.990539
The 3th K-Fold Prediction Accuracy Result -- Optimize With SA: 0.980209
The 4th K-Fold Prediction Accuracy Result -- Optimize With SA: 0.992760
The 5th K-Fold Prediction Accuracy Result -- Optimize With SA: 0.990191
The 6th K-Fold Prediction Accuracy Result -- Optimize With SA: 0.994388
The 7th K-Fold Prediction Accuracy Result -- Optimize With SA: 0.993912
The 8th K-Fold Prediction Accuracy Result -- Optimize With SA: 0.992717
The 9th K-Fold Prediction Accuracy Result -- Optimize With SA: 0.992672
```

```
The 0th K-Fold Prediction Accuracy Result -- Optimize With SA: 0.987819
The 1th K-Fold Prediction Accuracy Result -- Optimize With SA: 0.986426
The 2th K-Fold Prediction Accuracy Result -- Optimize With SA: 0.989213
The 3th K-Fold Prediction Accuracy Result -- Optimize With SA: 0.977253
The 4th K-Fold Prediction Accuracy Result -- Optimize With SA: 0.988604
The 5th K-Fold Prediction Accuracy Result -- Optimize With SA: 0.986628
The 6th K-Fold Prediction Accuracy Result -- Optimize With SA: 0.984980
The 7th K-Fold Prediction Accuracy Result -- Optimize With SA: 0.985509
The 8th K-Fold Prediction Accuracy Result -- Optimize With SA: 0.986564
The 9th K-Fold Prediction Accuracy Result -- Optimize With SA: 0.984255
Mean Accuracy of Cross Validation  -- Mn: 0.985725
```

七、问题三模型的建立与求解

7.1 成本优化模型的建立

在钢水脱氧合金化过程中，投入合金料的种类、数量的选择，既决定了合金元素的收得率，又决定了脱氧合金化过程的成本，我们基于问题二中的收得率预测模型，提出了成本优化模型，给出

具体的合金配料方案。

不难得出：

$$W = \arg\min \sum_i Cost（i）\times putMetal(i),\ i \in NameOfMetal$$

其中，W 表示最小化成本，$Cost（i）$表示第 i 种合金料的单价，$putMetal（i）$表示第 i 种合金料的投放量。

为了达到所炼钢材的合金元素含量要求[3]，应满足以下约束条件：

$$\begin{cases} \min(C) \le Begin（i，C）+ GainRate（C）\times \sum_i putMetal（i）\times percent（i，C）\le \max(C) \\ \min(P) \le Begin（i，P）+ GainRate（P）\times \sum_i putMetal（i）\times percent（i，P）\le \max(P) \\ \min(S) \le Begin（i，S）+ GainRate（S）\times \sum_i putMetal（i）\times percent（i，S）\le \max(S) \\ \min(Mn) \le Begin（i，Mn）+ GainRate（Mn）\times \sum_i putMetal（i）\times percent（i，Mn）\le \max(Mn) \\ \min(Si) \le Begin（i，Si）+ GainRate（Si）\times \sum_i putMetal（i）\times percent（i，Si）\le \max(Si) \end{cases}$$

基于问题二的预测模型和投入合金料的决策序列$\{putMetal\}$，得到 C、Mn、P、S、Si 各个元素的收得率 GainRate，应用于约束条件，使得成本优化模型转化为求解W 最小的多元线性规划问题。

7.2 模型的求解

为了求解该多元线性规划问题，本文引入 G.B.Dantzig 提出的 Simplex Method[1]，该方法是一种顶点迭代算法，即从一个顶点出发，沿凸多面体的棱迭代到另一个顶点，使得目标函数值下降。由于顶点个数的有限性，可以证明经过有限次迭代之后一定能求得最优解或证明无解，而几何空间中的一个点可以表示为多元线性规划问题中的一组可行解，具体步骤如下：

STEP1：求出一个初始解$x_0 = \{putMetal\}$，以及对于的目标函数值W_0；

STEP2：在可行解的几何空间中，取出x_0所在的棱上的点，将这些点的目标函数值c_i和W_0进行比较，得到检验数$r_i = c_i - W_0$，若$r_i \ge 0$恒成立，那么x_0为最优解，转入 STEP4，否则，进入 STEP3；

STEP3：x_0不是最优解说明方案仍有改进的余地，在x_0的坐标中选取一个非基变量，将它转变为基变量，进行换基计算，求得一个新的可行解x_0，转入 STEP2；

STEP4：得到x_0为最优解；

7.3 模型的评估

我们结合附件 1 中的历史数据，读取输入模型，获取炼钢炉号为 7A06878-7A06620 的炼钢过程所需最优成本的计算，结果如下：

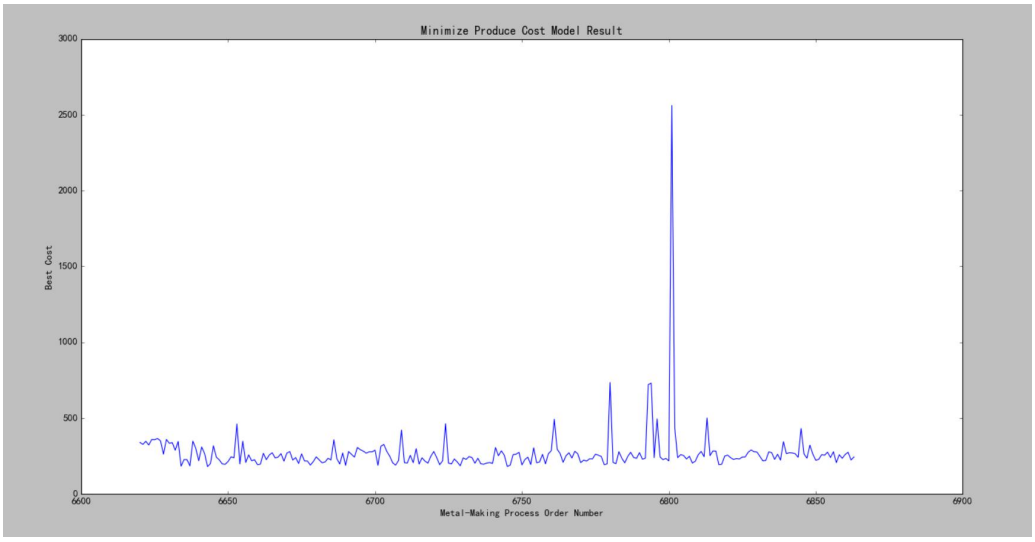

图 11. 多元线性规划成本优化模型，针对历史合金化数据计算的最优成本

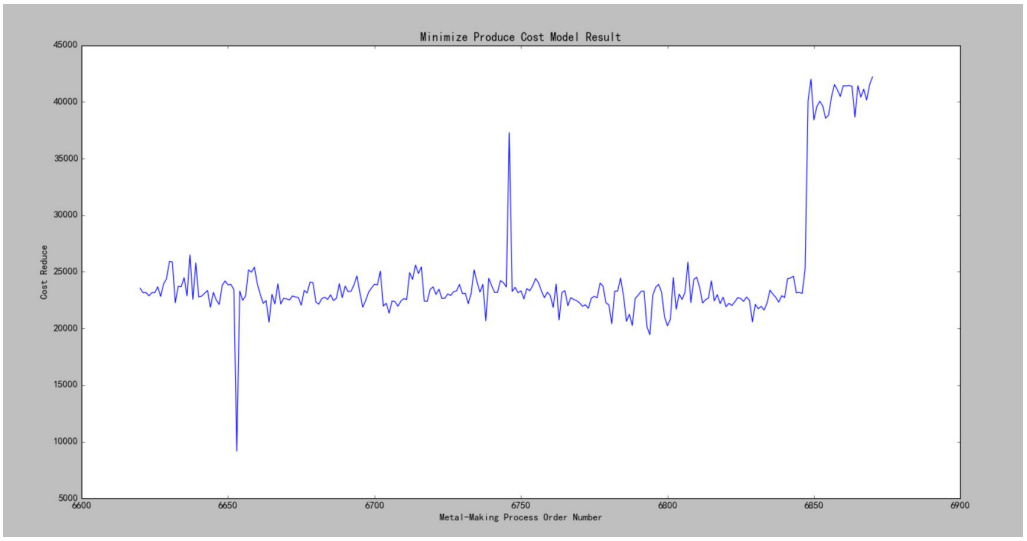

图 12. 模型的最优成本，相对于历史成本的节省金额曲线

此外，我们还将成本优化模型所得的最优成本结果，以及配料方案，具体数据放置于提交附件**成本优化模型结果.xlsx 中，**可自行查看模型优化结果。此处列举历史数据中，编号排序前十的合金化成本最优的配料方案(保留两位小数)，以及其对应的最优成本：

炉号	氮化钒铁	低铝硅铁	钒氮合金	钒铁(FeV50-A)	钒铁(FeV50-B)	硅铝钙	硅铝合金FeA130	硅铝锰合金球	硅锰面	硅铁	硅铁FeSi75-B	石油焦增碳剂	锰硅合金FeMn64	锰硅合金FeMn68	碳化硅(55%)	硅钙碳脱氧剂	优化后的成本(元)

							Si25						Si27	Si18			
6878	0	0.58	0	0.66	0.66	0.72	2.35	0.03	1.09	1.53	1.53	1.25	0.94	0.94	0.23	0.77	340
6877	0	0.59	0	0.63	0.63	0.63	2.29	0.03	1.10	1.55	1.55	1.25	0.94	0.94	0.24	0.77	327
6876	0	0.57	0	0.68	0.68	0.79	2.38	0.03	1.09	1.51	1.51	1.24	0.94	0.94	0.23	0.77	347
6875	0	0.58	0	0.62	0.62	0.64	2.25	0.03	1.10	1.56	1.56	1.25	0.94	0.94	0.24	0.77	323
6874	0	0.56	0	0.71	0.71	0.85	2.45	0.03	1.08	1.49	1.49	1.25	0.94	0.94	0.22	0.77	359
6873	0	0.56	0	0.70	0.70	0.86	2.45	0.03	1.08	1.49	1.49	1.24	0.94	0.94	0.22	0.77	358
6872	0	0.57	0	0.72	0.72	0.86	2.50	0.03	1.08	1.48	1.48	1.25	0.94	0.94	0.22	0.77	366
6871	0	0.57	0	0.68	0.68	0.78	2.41	0.03	1.09	1.51	1.51	1.25	0.94	0.94	0.23	0.77	350
6870	0	0.64	0	0.48	0.48	0.22	1.94	0.03	1.12	1.67	1.67	1.25	0.95	0.95	0.27	0.77	262
6869	0	0.56	0	0.71	0.71	0.85	2.45	0.03	1.08	1.49	1.49	1.25	0.94	0.94	0.22	0.77	360

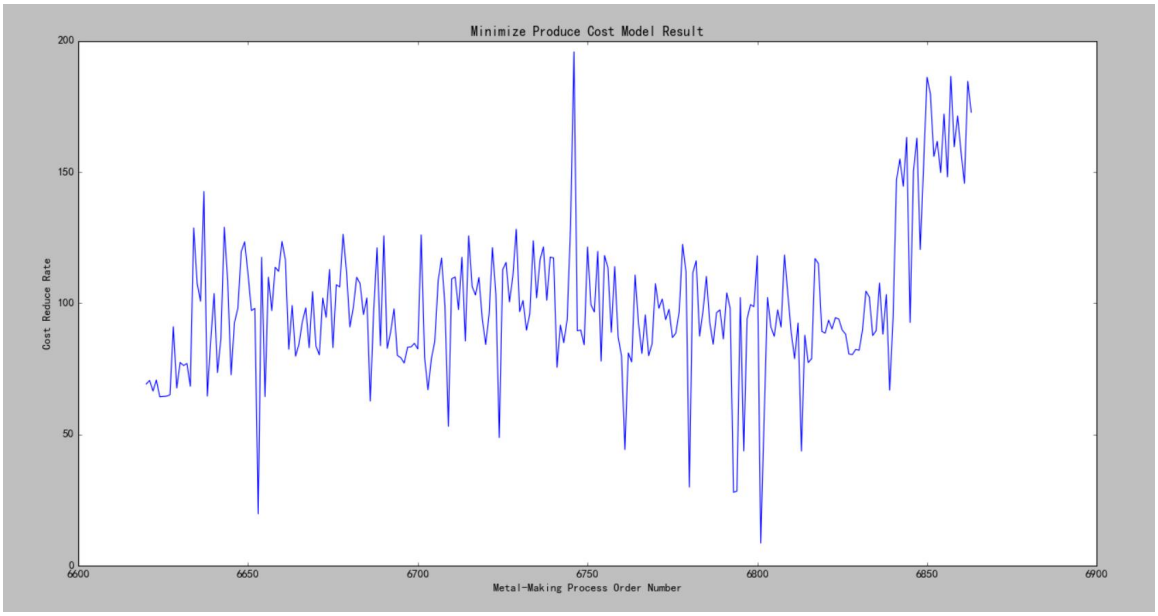

图 13. 多元线性规划成本优化模型，成本优化率曲线

结合成本优化模型的输出结果，我们可以获取从历史数据成本到模型优化成本之间的优化率，即成本整体的降低比率，最终可以得到，模型的平均成本优化率为：99.7713×10^{2} %

八、问题四的建议与方案

尊敬的炼钢厂领导：

您好，通过使用炼钢厂的历史数据进行数学模型的建立，我们得以成功的求解出钢水合金化过程的收得率预测模型，以及进一步地完成了合金化成本的优化模型，在此首先需要感谢您的数据支持。

此外，在使用您提供的历史数据进行建模的过程中，我们发现，炼钢厂在进行合金化过程时，对于硅锰合金的投入量往往超过一吨，而且使用的频率很高，同比于其余合金材料，硅锰合金虽然在 Mn 元素含量上占据高值，但是单价是比其余提供等量 Mn 元素的合金要高的，而且硅锰合金对于 C 元素，S 元素等其余主要元素的提供量占比较小，应考虑适当地降低硅锰合金的大规模采用情况。

上述的问题其实不仅体现在某种元素的使用规模上，其实根本原因是贵厂在炼钢过程中，并非采用自动控制的脱氧合金化原料选取模型，而是按照不同的固定收得率或经验值计算各种合金的加入量，难以实现不同炉次合金配料的自动优化和成本控制。

针对此根本问题，我们向您提出以下建议：

1. 采用梯度下降构建元素收得率预测模型，加以模拟退火法优化模型预测的准确率（即论文问题二构建的预测模型），实现不同炉次合金化前的不同收得率准确预测，适当地求解当前合金化过程每种主要元素所需的准确含量。

2. 利用多元变量线性规划模型，结合收得率预测模型的结果，合理地分配每种原料的添加重量，制定不同的配料方案，从而实现钢筋各种化学元素含量确实符合国家标准，同时求得总体合金化成本的最小值，切实降低炼钢过程的开销。

参加 MathorCup 的学生

2019.4.14

九、参考文献

[1] 展丙军. 单纯形法的改进及其应用 [A]. 黑龙江 : 大庆师范学院，数学系，2006. 1-4

[2] 龚伟. 转炉冶炼脱氧和合金化 [A]. 辽宁 : 东北大学，钢铁冶金专业，2001. 2-5

[3] GB/T 1499.2. 《钢筋混凝土用钢 第二部分：热轧带肋钢筋》. 1-7

附 录

1. 问题一的 Python 代码实现

```
'''
Introduction: Solution for Question One
'''
def plotGainRate(arr, title):
    plt.plot(range(1, len(arr)+1), arr)

    plt.title(title)
    plt.xlabel('Record History Order')
    plt.ylabel('Gain Rate')

    plt.show()

def plotFactor(x_arr, y_arr, title):
    x = range(len(x_arr))

    plt.plot(x, y_arr)
    plt.xticks(x, x_arr, rotation=20)

    plt.title(title)
    plt.xlabel('相关因素')
    plt.ylabel('相关性系数')

    plt.show()

def GetFactorsSeq(flag=0):
    Records = []
    OtherElements = []

    if flag == 0:
        length = len(Gain_Rate_C)
        Records = Records_C
        OtherElements = OtherElements_C
    elif flag == 1:
        length = len(Gain_Rate_Mn)
        Records = Records_Mn
        OtherElements = OtherElements_Mn
    elif flag == 2:
        length = len(Gain_Rate_S)
        Records = Records_S
        OtherElements = OtherElements_S
    elif flag == 3:
        length = len(Gain_Rate_P)
```

```
        Records = Records_P
        OtherElements = OtherElements_P
    elif flag == 4:
        length = len(Gain_Rate_Si)
        Records = Records_Si
        OtherElements = OtherElements_Si

    res = []

    # Influence Used Material Content
    Materials = []

    for ele in Records:
        mat = []
        for index in range(13, 17):
            mat.append(ele[index])
        for index in range(18, 29):
            mat.append(ele[index])
        Materials.append(mat)

    Materials = np.transpose(Materials)

    index = 0
    for ele in Records:
        temp = []
        temp.append(ele[1])
        temp.append(ele[flag + 8] - ele[flag + 2])
        temp.append(ele[7])
        temp.append(ele[len(ele) - 1])

        for loc in range(len(M_Orders)):
            temp.append(Materials[loc][index])

        for loc in range(len(OtherElements)):
            temp.append(OtherElements[loc][index])

        res.append(temp)
        index += 1

    return res

def GetResultSeq(flag=0):
    if flag == 0:
        return Gain_Rate_C
    elif flag == 1:
        return Gain_Rate_Mn
```

```
    elif flag == 2:
        return Gain_Rate_S
    elif flag == 3:
        return Gain_Rate_P
    elif flag == 4:
        return Gain_Rate_Si
```

```
def GetGainRateDiv():
    return Gain_Rate_Div
```

```
def Calculate_Gain_Rate(record, index1, index2, order):
    originC = record[index1]
    resultC = record[index2]
    totalWeight = record[7]
    # Check whether number
    if originC == '' or resultC == '':
        return -1
```

```
    improveC = 0.0
    materialC = 0.0
    if resultC > originC:
        improveC = float(resultC * totalWeight * (0.98) - originC * totalWeight)
    # Calculate used materials
    for loc in range(len(Cons)+1):
        if record[loc+13] > 0:
            if loc <= 4:
                materialC += record[loc+13] * Cons[loc][order]
            else :
                materialC += record[loc+13] * Cons[loc-1][order]
```

```
    if materialC == 0:
        return 0
    else:
        return improveC / materialC
```

```
def Calculate_CorrCoef(Records, OtherElements, Gain_Rate, target_index, name, M_Orders):
    Corr = []
    # Influence Improve Content
    Temperature = []
    IronWaterWeight = []
    OutOxygen = []
    Origin = []
```

```
    for ele in Records:
        Origin.append(ele[target_index])
```

```
        Temperature.append(ele[1])
        IronWaterWeight.append(ele[7])
        OutOxygen.append(ele[29])

    Corr_T = np.corrcoef(Temperature[:len(Gain_Rate)], Gain_Rate)
    Corr_O = np.corrcoef(Origin[:len(Gain_Rate)], Gain_Rate)
    Corr_I = np.corrcoef(IronWaterWeight[:len(Gain_Rate)], Gain_Rate)
    Corr_Ox = np.corrcoef(OutOxygen[:len(Gain_Rate)], Gain_Rate)

    Corr.append(Corr_T[0][1])
    Corr.append(Corr_O[0][1])
    Corr.append(Corr_I[0][1])
    Corr.append(Corr_Ox[0][1])

    # Influence Used Material Content
    Materials = []

    for ele in Records:
        mat = []
        for index in range(13, 17):
            mat.append(ele[index])
        for index in range(18, 29):
            mat.append(ele[index])
        Materials.append(mat)

    Materials = np.transpose(Materials)

    for index in range(len(M_Orders)):
        Corr_Temp = np.corrcoef(Materials[index][:len(Gain_Rate)], Gain_Rate)

        if np.isnan(Corr_Temp[0][1]):
            Corr.append(0)
        else:
            Corr.append(Corr_Temp[0][1])

    List_Names = np.hstack((['温度', '转炉终点' + name, '钢水重量', '硅钙碳脱氧剂'], M_Orders))
    OE = np.array(OtherElements)
    OE_Res = []

    for ele in OE:
        Corr_Temp = np.corrcoef(ele[:len(Gain_Rate)], Gain_Rate)
        OE_Res.append(Corr_Temp[0][1])
        Corr.append(Corr_Temp[0][1])

    OE_Names = ['Ceq_Val', 'Cr', 'Ni_Val', 'Cu_Val', 'V_Val', 'Alt_Val', 'Als_Val', 'Mo_Val']
```

```
    x = range(len(OE_Names))

    List_Names = np.hstack([List_Names, OE_Names])
    for index in range(len(Corr)):
        temp = Corr[index]
        Corr[index] = [temp, List_Names[index]]

    Corr.sort(key=Corr_Cmp, reverse=True)


def Corr_Cmp(a):
    return abs(a[0])

# Read Origin History Data
import matplotlib.pyplot as plt
import numpy as np
import xlrd

plt.rcParams['font.sans-serif']=['SimHei']
plt.rcParams['axes.unicode_minus']=False

Names = [[], [], [], [], []]
Records_C = []
Records_Mn = []
Records_S = []
Records_P = []
Records_Si = []

Cons = []

data1 = xlrd.open_workbook(r'../data/D-1.xlsx')
data2 = xlrd.open_workbook(r'../data/D-2.xlsx')

sheet = data1.sheet_by_name('Sheet5')
sheet1 = data2.sheet_by_name('Sheet1')

OtherElements_C = [[], [], [], [], [], [], [], []]
OtherElements_Mn = [[], [], [], [], [], [], [], []]
OtherElements_S = [[], [], [], [], [], [], [], []]
OtherElements_P = [[], [], [], [], [], [], [], []]
OtherElements_Si = [[], [], [], [], [], [], [], []]

# Read Iron Elements construction
for index in range(1, 17):
```

```
    cons = []
    for loc in range(1, 11):
        if sheet1.cell_value(index, loc) == '':
            cons.append(0.0)
        else:
            cons.append(float(sheet1.cell_value(index, loc)))
    Cons.append(cons)
```

```
# Read History Records
for index in range(1, 1717):
    record = []
    ss = (sheet.cell_value(index, 2)).strip()
    record.append(ss)
```

```
    for loc in range(3, 15):
        ss = sheet.cell_value(index, loc)
        if ss == '':
            record.append(0.0)
        else:
            record.append(sheet.cell_value(index, loc))
    for loc in range(28, 45):
        record.append(sheet.cell_value(index, loc))
```

```
    # Judge whether the gain rate exceed 1, otherwise delete the record
    grc = Calculate_Gain_Rate(record, 2, 8, 0)
    grm = Calculate_Gain_Rate(record, 3, 9, 1)
    grs = Calculate_Gain_Rate(record, 4, 10, 2)
    grp = Calculate_Gain_Rate(record, 5, 11, 3)
    grsi = Calculate_Gain_Rate(record, 6, 12, 4)
```

```
    if grc < 1.0 and grc > 0.0:
        Records_C.append(record)
        Names[0].append((sheet.cell_value(index, 2)).strip())
        for loc in range(15, 19):
            temp = sheet.cell_value(index, loc)
            if temp != '' and temp != 'NULL':
                OtherElements_C[loc-15].append(float(sheet.cell_value(index, loc)))
        for loc in range(20, 24):
            temp = sheet.cell_value(index, loc)
            if temp != '' and temp != 'NULL':
                OtherElements_C[loc-20+4].append(float(sheet.cell_value(index, loc)))
```

```
    if grm < 1.0 and grm > 0.0:
        Records_Mn.append(record)
        Names[1].append((sheet.cell_value(index, 2)).strip())
```

```
        for loc in range(15, 19):
            temp = sheet.cell_value(index, loc)
            if temp != '' and temp != 'NULL':
                OtherElements_Mn[loc-15].append(float(sheet.cell_value(index, loc)))
        for loc in range(20, 24):
            temp = sheet.cell_value(index, loc)
            if temp != '' and temp != 'NULL':
                OtherElements_Mn[loc-20+4].append(float(sheet.cell_value(index, loc)))
```

```
    if grs < 1.0 and grs > 0.0:
        Records_S.append(record)
        Names[2].append((sheet.cell_value(index, 2)).strip())
        for loc in range(15, 19):
            temp = sheet.cell_value(index, loc)
            if temp != '' and temp != 'NULL':
                OtherElements_S[loc-15].append(float(sheet.cell_value(index, loc)))
        for loc in range(20, 24):
            temp = sheet.cell_value(index, loc)
            if temp != '' and temp != 'NULL':
                OtherElements_S[loc-20+4].append(float(sheet.cell_value(index, loc)))
```

```
    if grp < 1.0 and grp > 0.0:
        Records_P.append(record)
        Names[3].append((sheet.cell_value(index, 2)).strip())
        for loc in range(15, 19):
            temp = sheet.cell_value(index, loc)
            if temp != '' and temp != 'NULL':
                OtherElements_P[loc-15].append(float(sheet.cell_value(index, loc)))
        for loc in range(20, 24):
            temp = sheet.cell_value(index, loc)
            if temp != '' and temp != 'NULL':
                OtherElements_P[loc-20+4].append(float(sheet.cell_value(index, loc)))
```

```
    if grsi < 1.0 and grsi > 0.0:
        Records_Si.append(record)
        Names[4].append((sheet.cell_value(index, 2)).strip())
        for loc in range(15, 19):
            temp = sheet.cell_value(index, loc)
            if temp != '' and temp != 'NULL':
                OtherElements_Si[loc-15].append(float(sheet.cell_value(index, loc)))
        for loc in range(20, 24):
            temp = sheet.cell_value(index, loc)
            if temp != '' and temp != 'NULL':
                OtherElements_Si[loc-20+4].append(float(sheet.cell_value(index, loc)))
```

```
Gain_Rate_C = []
```

```
Gain_Rate_Mn = []
Gain_Rate_S = []
Gain_Rate_P = []
Gain_Rate_Si = []

IronWaterLoss = 0.02

# Calculate Gain Rate of C
for index in range(len(Records_C)):
    grc = Calculate_Gain_Rate(Records_C[index], 2, 8, 0)
    if grc == -1:
        continue

    Gain_Rate_C.append(grc)
# Calculate Gain Rate of Mn
for index in range(len(Records_Mn)):
    grm = Calculate_Gain_Rate(Records_Mn[index], 3, 9, 1)
    if grm == -1:
        continue

    Gain_Rate_Mn.append(grm)

# Calculate Gain Rate of S
for ele in Records_S:
    grs = Calculate_Gain_Rate(ele, 4, 10, 2)
    if grs == -1:
        continue
    Gain_Rate_S.append(grs)


# Calculate Gain Rate of P
for ele in Records_P:
    grp = Calculate_Gain_Rate(ele, 5, 11, 3)
    if grp == -1:
        continue
    Gain_Rate_P.append(grp)

# Calculate Gain Rate of Si
for ele in Records_Si:
    grsi = Calculate_Gain_Rate(ele, 6, 12, 4)
    if grsi == -1:
        continue
    Gain_Rate_Si.append(grsi)
# Accord to Iron Name, divide the gain rate result
Gain_Rate_Div_Sample = [[], [], [], [], [], [], [], [], []]
Gain_Rate_Div = []
```

```
for order in range(5):
    Gain_Rate_Div.append(Gain_Rate_Div_Sample)
```

```
for order in range(5):
    Gain_Rate = []
    if order == 0:
        Gain_Rate = Gain_Rate_C
    elif order == 1:
        Gain_Rate = Gain_Rate_Mn
    elif order == 2:
        Gain_Rate = Gain_Rate_S
    elif order == 3:
        Gain_Rate = Gain_Rate_P
    else:
        Gain_Rate = Gain_Rate_Si
```

```
    for index in range(len(Names[order])):
        if Names[order][index] == 'HRB500B':
            Gain_Rate_Div[order][0].append(Gain_Rate[index])
        if Names[order][index] == '20MnKB':
            Gain_Rate_Div[order][1].append(Gain_Rate[index])
        if Names[order][index] == 'HRB400D':
            Gain_Rate_Div[order][2].append(Gain_Rate[index])
        if Names[order][index] == 'HRB400B':
            Gain_Rate_Div[order][3].append(Gain_Rate[index])
        if Names[order][index] == 'HRB500D':
            Gain_Rate_Div[order][4].append(Gain_Rate[index])
        if Names[order][index] == 'Q235A':
            Gain_Rate_Div[order][5].append(Gain_Rate[index])
        if Names[order][index] == 'Q235':
            Gain_Rate_Div[order][6].append(Gain_Rate[index])
        if Names[order][index] == '20MnKA':
            Gain_Rate_Div[order][7].append(Gain_Rate[index])
        if Names[order][index] == 'Q345B':
            Gain_Rate_Div[order][8].append(Gain_Rate[index])
'''
Basically select influence factors:
* Influence Improve Content
    1. Temperature
    2. Origin Percent of the Element
    3. Iron-Water weight
    4. Out-Oxygen content
* Influence Used Material Content
    1. Materials' content having target Element
'''
```

```
M_Orders = ['氮化钒铁', '低铝硅铁', '钒氮合金', '钒铁(FeV50-A)', '钒铁(FeV50-B)', '硅铝钙',
            '硅铝合金', '硅铝锰合金球', '硅锰面', '硅铁', '硅铁 FeSi75-B', '石油焦增碳剂',
            '锰硅合金 FeMn64Si27', '锰硅合金 FeMn68Si18', '碳化硅']
```

2. 问题二的 Python 代码实现

```
'''
Introduction: Solution for Question Two
'''
import xlwt
import math
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d
from matplotlib import style
from PartOne import *

# Feature Scaling
def featureNormalize(X):
    X_norm = X;
    mu = np.zeros((1,X.shape[1]))
    sigma = np.zeros((1,X.shape[1]))
    for i in range(X.shape[1]):
        mu[0,i] = np.mean(X[:,i]) # Mean
        sigma[0,i] = np.std(X[:,i]) # Std
```

```
    X_norm = (X - mu) / sigma
    return X_norm,mu,sigma
# Cost Computing
def computeCost(X, y, theta):
    m = y.shape[0]
```

```
    C = X.dot(theta) - y
    J2 = (C.T.dot(C))/ (2*m)
    return J2
# Gradient Descent
def gradientDescent(X, y, theta, alpha, num_iters):
    m = y.shape[0]
    T = 1000
    factor = 0.01

    # Store History Loss
    J_history = np.zeros((num_iters, 1))
    for iter in range(num_iters):
        target = np.dot(X, theta)
```

```
        loss = target - y
        gradient = np.dot(X.T, loss) / m
        # Here use Simulated Annealing to Optimize Gradient Descent
        pre_theta = theta

        theta = theta - alpha * gradient
        J_history[iter] = computeCost(X, y, theta)

        # Judge whether the loss always moves in best directions
        if J_history[iter] > J_history[iter - 1]:
            # Accept the wrose answer in probability
            delta_E = abs(J_history[iter] - J_history[iter - 1])
            random_num = np.random.rand(1)[0]
            if math.exp(delta_E / T) <= random_num:
                theta = pre_theta

        T -= factor * T

    return J_history,theta

# Result Prediction
def predict(data, theta):
    X = np.array(data)

    the = []
    for ele in theta:
        the.append(ele[0])

    return X.dot(the)

# Do Origin Data Train And Test
iterations = 100000
alpha = 0.001
length = 27

xVars = np.array(GetFactorsSeq(), np.float64)
yVars = np.array(GetResultSeq(), np.float64)

print(xVars[0])

x = xVars.reshape((-1, length))
y = yVars.reshape((-1, 1))

m = y.shape[0]

x, mu, sigma = featureNormalize(x)
```

```
for ele in x:
    for loc in range(length):
        if np.isnan(ele[loc]):
            ele[loc] = 0
```

```
X = np.hstack([x,np.ones((x.shape[0], 1))])
```

```
theta = np.zeros((length+1, 1))
```

```
J_history,theta = gradientDescent(X, y, theta, alpha, iterations)
```

```
print('Best Weights Combination Found By Gradient Descent: ')
```

```
for index in range(len(theta)):
    if index <= 2:
        print('w%d of Mn Prediction Model : %f' % (index, theta[index]))
    elif index >= 3 and index <= 18:
        print('w3_%d of Mn Prediction Model : %f'% (index-3, theta[index]))
    else:
        print('w4_%d of Mn Prediction Model : %f' % (index-19, theta[index]))
```

```
# Prediction Accuracy Check
predicts = []
loss = []
accuracy = []
```

```
for ele in X:
    predicts.append(predict(ele, theta))
```

```
for index in range(len(predicts)):
    lossRate = abs(predicts[index] - y[index]) / max(y[index], predicts[index])
    if lossRate == 1:
        print(predicts[index], y[index])
    loss.append(lossRate)
    accuracy.append(1-lossRate)
```

```
# print(accuracy)
```

```
Mean = 0
for ele in accuracy:
    if ele[0] >= 0 and ele[0] <= 1.0:
        Mean += ele[0]
```

```
Mean /= len(accuracy)
```

```
# Do Cross Validation, Dividing into ten parts
part_length = int(len(X) / 10)

Means = []

for index in range(10):
    testX = X[index * part_length : (index+1) * part_length]
    testY = y[index * part_length : (index+1) * part_length]
    trainX_front = X[0 : index * part_length]
    trainX_end = X[(index+1) * part_length : ]
    trainX = np.vstack((trainX_front, trainX_end))
    trainY = np.vstack((y[0 : index * part_length], y[(index+1) * part_length : ]))

    theta = np.zeros((length+1, 1))

    J_history, theta = gradientDescent(trainX, trainY, theta, alpha, iterations)

    predicts = []
    loss = []
    accuracy = []

    for ele in testX:
        predicts.append(predict(ele, theta))

    for i in range(len(predicts)):
        lossRate = abs(predicts[i] - testY[i]) / max(testY[i], predicts[i])

        loss.append(lossRate)
        accuracy.append(1-lossRate)

    Mean = 0
    for ele in accuracy:
        if ele[0] >= 0 and ele[0] <= 1.0:
            Mean += ele[0]

    Mean /= len(accuracy)
    print('The %dth K-Fold Prediction Accuracy Result -- Optimize With SA: %f' % (index, Mean))
    Means.append(Mean)

Ms = np.array(Means)
print("Mean Accuracy of Cross Validation -- Mn: %f" % Ms.mean())
```

3. 问题三的 Python 代码实现

```
'''
Introduction: Solution for Question Three
'''
from PartOne import *
from scipy import optimize
import matplotlib.pyplot as plt
import numpy as np
import xlrd
import xlwt

Iron_Names = ['HRB500B', '20MnKB', 'HRB400D', 'HRB400B', 'HRB500D', 'Q235A', 'Q235', '20MnKA', 'Q345B']

def ReadIronConstraint():
    res = []

    data3 = xlrd.open_workbook(r'../data/钢筋化学元素含量国家标准.xlsx')
    sheet = data3.sheet_by_name('Sheet1')

    for index in range(9):
        element_range = []
        temp = []
        for k in range(5):
            element_range.append(sheet.cell_value(index+2, k+2))
        for ele in element_range:
            arr = ele.split('-')
            temp.append([float(arr[0]) * 0.01, float(arr[1]) * 0.01])

        res.append(temp)

    return res

def GetOriginElementDiv():
    res = [[], [], [], [], [], [], [], [], []]
    data1 = xlrd.open_workbook(r'../data/D-1.xlsx')
    sheet = data1.sheet_by_name('Sheet5')
    for index in range(1, 1717):
        iron_name = (sheet.cell_value(index, 2)).strip()
        iid = (sheet.cell_value(index, 0)).strip()
        iid = iid[3:]
        Origin = []
        Origin.append(int(iid))

        for i in range(5):
            temp = sheet.cell_value(index, i + 4)
```

```
            if temp == '':
                return res
            Origin.append(float(temp))
        Origin.append(int(sheet.cell_value(index, 9)))
        for i in range(28, 32):
            temp = sheet.cell_value(index, i)
            Origin.append(float(temp))
        for i in range(33, 45):
            temp = sheet.cell_value(index, i)
            Origin.append(float(temp))
        target_id = -1
        if iron_name == 'HRB500B':
            target_id = 0
        elif iron_name == '20MnKB':
            target_id = 1
        elif iron_name == 'HRB400D':
            target_id = 2
        elif iron_name == 'HRB400B':
            target_id = 3
        elif iron_name == 'HRB500D':
            target_id = 4
        elif iron_name == 'Q235A':
            target_id = 5
        elif iron_name == 'Q235':
            target_id = 6
        elif iron_name == '20MnKA':
            target_id = 7
        else:
            target_id = 8
        res[target_id].append(Origin)
    return res
```

```
def GetPriceAndContent():
    data2 = xlrd.open_workbook(r'../data/D-2.xlsx')
    sheet = data2.sheet_by_name('Sheet1')
    price = []
    Contents = []
    for index in range(1, 17):
        price.append(int(sheet.cell_value(index, 10)) / 1000)
```

```
    for index in range(1, 17):
        temp = []
        for i in range(1, 6):
            content = sheet.cell_value(index, i)
            if content == '':
                temp.append(0.0)
```

```
            else:
                temp.append(float(content))
        Contents.append(temp)
    return price, Contents
```

```
def Cmp(a):
    return a[0]

if __name__ == '__main__':
    Constraint = ReadIronConstraint()
    # According to different Iron Type, Read Predict Gain_Rate and History Origin Elements Value
    Gain_Rate_Div = GetGainRateDiv()
    Origin_Element_Div = GetOriginElementDiv()
    # We need for different Iron Types, the five elements should input how much
    Need_Element_Value = [[], [], [], [], [], [], [], [], []]
    for index in range(9):
        ele = Origin_Element_Div[index]
        cons = Constraint[index]

        j = 0
        for el in ele:
            Need = []
            Need.append(el[0])
            for i in range(1, 6):
                need = 0.0
                origin_ele = el[i]
                if origin_ele >= cons[i-1][0]:
                    need = 0.0
                else:
                    need = cons[i-1][0] - origin_ele
                # Divide to the Gain Rate
                target_gain_rate = Gain_Rate_Div[i-1][index][j]
                Need.append(need / target_gain_rate * el[6])
            j += 1

            Need_Element_Value[index].append(Need)
```

```
    # Read in Metal Source price and elements content inside each metal
    price, Contents = GetPriceAndContent()
    # Target Minize Function
    Min_Target_Func = price
    Min_Target_Func = np.array(Min_Target_Func)
    Contents = np.array(Contents)
    Contents = Contents.T
    '''
    Define Linear Planning Eqation Set:
```

```
        1. All used metals must fit elements needed for the problem
        2. All used metals should cost the least money
    '''
    Result_Price = []
    Result_X = []
    Old_Price = []
    for index in range(9):
        temp = Need_Element_Value[index]
        if len(temp) == 0:
            continue
```

```
        for ele in temp:
            Conditions = []
            for i in range(1, 6):
                Conditions.append(ele[i])
            Conditions = np.array(Conditions)
```

```
            # Perform Linear Optimize
            res = optimize.linprog(c=Min_Target_Func, A_eq=Contents, b_eq=Conditions,
method='interior-point',
bounds=((0,None),(0,None),(0,None),(0,None),(0,None),(0,None),(0,None),(0,None),(0,None),(0,No
ne),(0,None),(0,None),(0,None),(0,None),(0,None)))

            Result_X.append([ele[0], res.x])
            Result_Price.append([ele[0], res.fun])
    for index in range(9):
        ele = Origin_Element_Div[index]
        for el in ele:
            temp = el[7:]
            Old_Price.append([el[0], np.array(temp).dot(Min_Target_Func)])
    Result_Price.sort(key=Cmp, reverse=True)
    Result_X.sort(key=Cmp, reverse=True)
    Old_Price.sort(key=Cmp, reverse=True)
    Reduce_Arr = []
    Result_Arr = []
    for index in range(len(Result_Price)):
        if Result_Price[index][1] > 1:
            Result_Arr.append(Result_Price[index][1])
        res = (Old_Price[index][1] - Result_Price[index][1]) / Result_Price[index][1]
        if Result_Price[index][1] > 1:
            Reduce_Arr.append(res)
```