水下机器人的组装计划

摘要

近年，随着世界全球化加深，传统的供应关系已经不再适应于当今生产需要，其中原材料供应是企业供应链的主要环节。本文提出了原材料供应数学模型的基础假设，给出企业对供应过程中订购与运输方案的建议。

对于问题一，我们通过对供货商供货特征的定量分析，取得描述供应商重要程度的评价指标，建立了基于熵权逼近理想解方法的供应商排序模型。其中，我们主要从供应商的产业规模、供货能力、违约率、波动情况，来衡量此供应商对该企业正常生产的重要性。为了定量并客观地评价每个供应商的重要程度，我们在模型中使用了熵权，以有效避免多因素评价模型当中确定权重时的主观性。通过比较对象与理想解和负理想解的距离，得到出对象与理想解的相似度，最后排序优选确定最重要的 50 家供应商。

对于问题二，我们建立线性规划模型以对订购方案进行优化，并制定出未来 24 周原材料最经济的订购方案。其中，在供应商的二次优选中，我们注意到每个供应商仅有选择与否两种状态，故采用 0–1 规划取得满足供应需求的最少的 44 家供应商；考虑到原材料储备对保障企业生产的重要意义，制定订购方案前，我们首先应用 LSTM 模型预测选定的供应商的供应水平，确保企业供应链稳定与生产安全，进一步优化订购方案。根据得到的订购方案制定损耗最小的转运方案，最后通过计算机程序仿真，以模拟优化后方案的实施效果。

对于问题三，由于企业为压缩生产成本，尽量多地采购 A 类和尽量少地采购 C 类原材料。我们对问题二中线性规划模型进行扩展，减少生产企业对 C 类原材料供应商的依赖，并将 A 类原材料的采购优先度提高，构建线性规划模型，解出最优采购方案和转运方案。最后由计算机仿真体现方案的实施效果。

对于问题四，我们不再考虑因企业生产技术有限产生的产量上限，而通过优化订购方案、转运方案最大化供应链的供应能力。而通过分析供货数据我们发现，每周转运商的总转运能力远少于供给商的供给能力。因此，我们以对转运商转运方案的优化为主，通过对模型的不断反思与改进，得出了未来 24 周的订购和转运方案。

最后我们对模型进行了中肯的评价和适当的推广。

关键字： TOPSIS 法　熵权法　LSTM 模型　线性规划

近年来，随着社会经济的发展和科学技术的进步，管道越来越多的运用在我们的生活当中。管道运输是最实用、最经济的运输方式，所以管道运输在生活和生产中的使用也越来越广泛[1]。其中，自来水管道已经进入千家万户。当净水厂的出厂水经过供水管网，输送给用户时，其在供水管网中将会发生复杂的生物、物理、化学反应[2]。导致污垢在自来水管道壁上积累，影响自来水的品质和用途。自来水管道清理机器人是一种体型较小，使用机械臂以辅助完成管道清理任务的自动装置。相比与传统人工清理方式，机器人清理具有及时、高效的优点，因此倍受水务公司和住户的青睐。

一、 问题重述与分析

某工厂生产的 WPCR 装置由 3 个容器艇（用 A 表示）、4 个机器臂（用 B 表示）、5 个动力系统（用 C 表示）组装而成。而 A、B、C 由以下部件组成：

- 容器艇（A）由 6 个控制器（A1）、8 个划桨（A2）和 2 个感知器（A3）组成，组装需消耗 3 个工时；
- 机器臂（B）由 2 个力臂组件（B1）和 4 个遥感器（B2）组成，组装需消耗 5 个工时；
- 动力系统（C）由 8 个蓄电池（C1）、2 个微型发电机（C2）和 12 个发电螺旋（C3）组成，组装需消耗 5 个工时。

工厂在某一天生产组件产品时，都要付出一个与生产数量无关的固定成本，称为生产准备费用。而当一天结束时仍有某部件的库存，则须付出额外的库存费用。每次生产计划的计划期为一周，提供的最终产品为 WPCR，以满足订单需要，不可轻易缺货断供。

为了最大化经济效益，帮助生产工厂做出决策，本文建立了??? 模型。

1.1 问题一

题目要求生产周期开始时没有任何组件库存，周期结束后也不留下任何组件库存。在部件采购与 WPCR 组装无延迟的基础上，要求总成本最小。因此可直接以总成本为目标函数建立线性规划模型，在题目所给约束条件下求得最优解，以制定每周 7 天的生产计划。

1.2 问题二

题目要求在问题一模型的基础上，考虑组件入库延迟对模型和生产计划的影响。与问题一组件入库无延迟不同，问题二中组装产品所需的组件要提前一天入库才能参与生产。因此，本文在问题一模型约束条件中，添加了令工厂只得使用前一天的组件库存生产新组件的限制。通过求解新模型，得到记入组件入库延迟的最优 7 天的生产计划。

1.3 问题三

题目要求在问题二模型的基础上，考虑如何安排生产工厂停工检修，可令总成本最小。总共需设置 7 次停工检修，每次检修时间为 1 天，检修之后关键设备生产能力会略有上升。因此，本文在模型约束条件中，引入了表示当天 t 是否检修的 0–1 变量 μ_t，并根据题目调整关键设备产能。在上文基础上，建立了以总成本为目标函数的线性规划模型，以确定检修日安排的最佳方案。

1.4 问题四

题目中要求在供应商和转运商有限的情况下，评估企业每周产能的上限。本题取消生产企业每周产能上限的同时，也消除了原材料的库存问题。而通过分析供货数据不难发现，向企业输送的最大供给量主要取决于转运商的转运能力。本问题也转化为在转运商以最大能力运输原材料时，一家供应商每周供应的原材料尽量由一家转运商运输。因此，本文主要对转运商的转运方案进行优化，并据此给出未来 24 周的订购和转运方案。

最后我们对模型进行了中肯的评价和适当的推广。

二、 符号说明

表 **1**　文中符号所用说明

符号	说明
x_t^r	第 t 周，组件 r（包括 WPCR）的组装数量.
y_t^r	第 t 周，组件 r（包括 WPCR）的库存数量.
d_t	第 t 周，WPCR 的外部需求数量.
M_t^r	第 t 周，组件 r（仅包括 A、B 和 C）的生产总工时限制.
s^r	组件 r（包括 WPCR）的生产准备费用.
h^r	组件 r（包括 WPCR）的单件库存费用.
c^r	组件 r（仅包括 A、B 和 C）的单件工时消耗.

三、 工厂生产计划设计（问题一）

3.1 模型的建立

本题要求制定每周 7 天的生产计划，使总成本最小。工厂生产成本可分为两个部分：

1. 各部件生产的生产准备费用

   只与组件相关，与开工时间无关、任一天开工的生产准备费均相等，后文记为 s^r；

2. 各部件的存贮费用

   在每个时间阶段（本题将一周七天订为整个时间跨度 T，并等分为七个时间阶段 t）结尾，如果有产品 r 库存，则需支付相应的储存费用，单件产品 1 个时段的存贮费为 h_r。

另外，由于产品 A、B、C 的加工过程需占用关键设备，消耗的时间不可忽略，记为工时 $c^r(r=\text{A、B、C})$。

为了使总成本最小，本文以总成本为目标函数，建立线性规划模型。目标函数如下：

$$\min z=\sum_{t=1}^{T}\sum_{r=1}^{R}\left(s^r\omega_t^r+h^r y_t^r\right). \tag{1}$$

上式中，$h^r y_t^r$ 表示产品 r 库存的存贮费，其中，y_t^r 是该产品的存贮数量；$s^r\omega_t^r$ 表示部件 r 生产的生产准备费用。其中，ω_t^r 是为了表述在 t 时段是否生产组件 r，从而确定是否要支付生产准备费，而引用的 0–1 变量

$$\omega_t^r=\begin{cases}1, & x_t^r>0\\ 0, & x_t^r=0\end{cases},\quad t=1,2,\cdots,T, r=1,2,\cdots,R. \tag{2}$$

题目中还要求生产周期开始时没有任何组件库存，周期结束后也不留下任何组件库存，且部件采购与 WPCR 组装没有延迟，故应满足一下约束条件：

- 库存数量

  组件 r 在第 t 天的库存量，应以前一日的库存量 y_{t-1}^r 与当日组装数量 x_t^r 之和，再减去当日为生产组件 r' 而消耗的数量

  $$y_t^r=y_{t-1}^r+x_t^r-\eta_{r'}^r x_t^{r'},\quad t=1,2,\cdots,T, r=1,2,\cdots,R. \tag{3}$$

  上式中，$\eta_{r'}^r$ 为组装一个产品 r' 所需特定组件 r 的数量。

- WPCR 需求量

  WPCR 每日供应量应等于当日的 WPCR 外部需求数 d_t

  $$y_{t-1}^{\text{WPCR}}+x_t^{\text{WPCR}}-y_t^{\text{WPCR}}=d_t,\quad t=1,2,\cdots,T \tag{4}$$

- 逻辑变量

  为了表示在 t 时段是否生产组件 r，从而确定是否要支付生产准备费，引用 0–1 变量

ω_t^r，等于 1 表示生产，反之表示不生产

$$\omega_t^r = \begin{cases} 1, & x_t^r > 0 \\ 0, & x_t^r = 0 \end{cases}, \quad t = 1, 2, \cdots, T, r = 1, 2, \cdots, R. \tag{5}$$

这是一个分段函数，用组件 r 的组装数量 x_t^r 是否大于 0，来判断是否需要生产准备费。

- 生产总工时

  产品 A、B、C 的加工过程需占用关键设备，消耗的时间不可忽略，记为工时满足下式

$$\sum_{r=1}^{R} c^r x_t^r \leqslant M_t, \quad t = 1, 2, \cdots, T, \tag{6}$$

  各部件生产工时之和应小于总生产工时。

- 题中所给边界条件

  要求生产周期开始时没有任何组件库存，周期结束后也不留下任何组件库存，因此，令：

$$y_0^r = y_T^r = 0, \quad r = 1, 2, \cdots, R. \tag{7}$$

- 非负约束

  显然，组件 r（包括 WPCR）的组装数量与库存数量应为非负数

$$x_t^r, y_t^r \geqslant 0, \quad t = 1, 2, \cdots, T, r = 1, 2, \cdots, R. \tag{8}$$

由于式5不便直接用程序计算，故将式5、6合并为下式

$$\begin{gathered} c^r x_t^r = M_t^r \omega_t^r, \quad t = 1, 2, \cdots, T, r = 1, 2, \cdots, R; \\ x_t^r = x_t^r \omega_t^r, \quad t = 1, 2, \cdots, T, r = 1, 2, \cdots, R; \\ \textstyle\sum_{r=1}^{R} M_t^r \leqslant M_t; \\ M_t^r \leqslant M_t^r \omega_t^r, \quad t = 1, 2, \cdots, T, r = 1, 2, \cdots, R; \\ M_t^r \geqslant 0, \quad r = 1, 2, \cdots, R. \end{gathered} \tag{9}$$

不难发现，当 $\omega_t^r = 0$ 时，x_t^r 必然为 0, 反之亦然，此次变换为编程实现提供了便利。

3.2 模型的求解

题中要求制定每周 7 天的生产计划，故有 $T=7$。综上，建立的线性规划模型如下：

$$
\begin{aligned}
\min \quad & z=\sum_{t=1}^{T}\sum_{r=1}^{R}\left(s^r\omega_t^r+h^r y_t^r\right) \\
\text{s.t.} \quad & \textstyle\sum_{r=1}^{R} M_t^r \leqslant M_t; \\
& M_t^r \leqslant M_t^r\omega_t^r, && t=1,2,\cdots,T, r=1,2,\cdots,R; \\
& y_t^r = y_{t-1}^r + x_t^r - \eta_r^r x_t^{r'}, && t=1,2,\cdots,T, r=1,2,\cdots,R; \\
& \omega_t^r \in \{0,1\}, && t=1,2,\cdots,T, r=1,2,\cdots,R; \\
& x_t^r = x_t^r\omega_t^r, && t=1,2,\cdots,T, r=1,2,\cdots,R; \\
& c^r x_t^r = M_t^r\omega_t^r, && t=1,2,\cdots,T, r=1,2,\cdots,R; \\
& M_t^r \geqslant 0, && r=1,2,\cdots,R; \\
& y_0^r = y_T^r = 0, && r=1,2,\cdots,R; \\
& x_t^r, y_t^r \geqslant 0, && t=1,2,\cdots,T, r=1,2,\cdots,R; \\
& y_{t-1}^{\text{WPCR}} + x_t^{\text{WPCR}} - y_t^{\text{WPCR}} = d_t, t=1,2,\cdots,T;
\end{aligned}
. \tag{10}
$$

从一方面来说，本模型同时包含连续变量和整数变量，是混合整数规划。从另一方面，由于目标函数和约束条件对于决策变量而言都是线性的，所以本模型为线性规划。因此，本模型为混合线性规划模型，存在最优解且已有许多可靠的求解器以供使用。

3.3 生产方案展示

一般地，工厂为降低总成本有两种策略，分别是：开车型和仓储型。极端地，选择开车型的工厂将每个时段均进行生产以满足且仅满足所在时段需求，来尽可能减少库存费；而选择仓储型的工厂会把生产任务集中安排，来尽可能减少生产准备费，代价是会损失因产品积压导致的库存费。然而，由于关键设备总工时的限制，这两种策略不一定能不折不扣地实施（如：实际计算发现，周日的需求不能仅靠当日生产满足、也不存在任何一天能组装出满足一周需求的产品）。

故模型的最优解正是这两种策略达到平衡时的状态。

四、 工厂生产计划扩展（问题二）

现实情况中，新采购的组件并不能直接投入使用，而是应由供应商生产、交付、转运至库存中，再供生产取用。因此，在生产工厂发出采购订单后，订购的组件需经过一定延迟时间，才能供实际生产使用。

4.1 模型的建立与求解

本题要求在记入组件入库延迟的前提下，设计生产工厂每周 7 天的生产计划。与问题一组件入库无延迟不同，问题二中组装产品所需的组件要提前一天入库才能参与生产。

在此基础上，由于该工厂第一天（周一）开始时没有任何组件库存，而新采购的组件在第二天（周二）才能投入生产。因此工厂必须在前一天（上周周日），准备好了刚好满足周一需求的 WPCR 库存 (即 $y_0^{\text{WPCR}} = 39$)，以免缺货断供。

综上，本文在问题一模型基础上，添加以下约束条件：

$$\eta_{r'}^{r} x_t^{r'} \leqslant y_{t-1}^{r}, \quad t = 1, 2, \cdots, T, r = 1, 2, \cdots, R. \tag{11}$$

上式中，通过令工厂当天生产所需组件数量 $\eta_{r'}^{r} x_t^{r'}$，小于前一天库存组件 y_{t-1}^{r}，以限制工厂只得使用前一天的组件库存生产新组件。

将上式11加入问题一中线性规划模型式10，得：

$$\begin{array}{lll}
\min & z = \sum\limits_{t=1}^{T}\sum\limits_{r=1}^{R}\left(s^r\omega_t^r + h^r y_t^r\right) & \\
\text{s.t.} & \sum_{r=1}^{R} M_t^r \leqslant M_t; & \\
& M_t^r \leqslant M_t^r \omega_t^r, & t = 1, 2, \cdots, T, r = 1, 2, \cdots, R; \\
& y_t^r = y_{t-1}^r + x_t^r - \eta_r^r x_t^{r'}, & t = 1, 2, \cdots, T, r = 1, 2, \cdots, R; \\
& \omega_t^r \in \{0, 1\}, & t = 1, 2, \cdots, T, r = 1, 2, \cdots, R; \\
& x_t^r = x_t^r \omega_t^r, & t = 1, 2, \cdots, T, r = 1, 2, \cdots, R; \\
& c^r x_t^r = M_t^r \omega_t^r, & t = 1, 2, \cdots, T, r = 1, 2, \cdots, R; \\
& M_t^r \geqslant 0, & r = 1, 2, \cdots, R; \\
& y_0^r = y_T^r = 0, & r = 1, 2, \cdots, R; \\
& x_t^r, y_t^r \geqslant 0, & t = 1, 2, \cdots, T, r = 1, 2, \cdots, R; \\
& y_{t-1}^{\text{WPCR}} + x_t^{\text{WPCR}} - y_t^{\text{WPCR}} = d_t, & t = 1, 2, \cdots, T; \\
& \eta_{r'}^{r} x_t^{r'} \leqslant y_{t-1}^{r}, & t = 1, 2, \cdots, T, r = 1, 2, \cdots, R;
\end{array} \tag{12}$$

4.2 生产方案展示

五、 检修方案设计

为了保障生产安全、提高工厂生产条件，生产工厂常常有计划地对关键设备进行检修升级。对关键设备的检修势必会影响其正常生产，因此有必要针对工厂具体生产条件，合理设计检修安排，保证工厂安全、高效运转。

5.1 模型的建立与求解

题目要求制定生产工厂停工检修计划，并使总成本最小。为了表示当天 t 是否检修，本文引入了 0–1 变量 μ_t，等于 1 表示当天进行检修，等于 0 表示不进行检修

$$\mu_t \in \{0, 1\}. \tag{13}$$

总共设置 7 次停工检修，每次检修时间为 1 天，故有：

$$\sum_{t=1}^{210} \mu_t = 7. \tag{14}$$

生产工厂在检修日无法生产，且两次检修需彼此间隔 6 天及以上。

$$\sum_{t=i-5}^{i} \mu_t \leqslant 1, \quad i = 6, 7, \cdots, 210. \tag{15}$$

上式中，通过对任意 6 天内检修与否逻辑 μ_t 进行求和，以检验检修日间隔长度，保证检修日间隔符合题目要求。

检修之后，工厂关键设备生产效率会略有上升（工时限制放宽 10%），并以 2%/天的速率衰减到 0。因此，需对问题一中式9总工时限制：

$$\sum_{r=1}^{R} M_t^r \leqslant M_t. \tag{16}$$

进行修正，得：

$$\sum_{r=1}^{R} M_t^r \leqslant M_t \left(1 - \mu_t + 0.1\mu_{t-1} + 0.08\mu_{t-2} + 0.06\mu_{t-3} + 0.04\mu_{t-4} + 0.0\mu_{t-5}\right). \tag{17}$$

其中，M_t^r 表示第 t 周、组件 r 的生产总工时花费，M_t 表示计划中第 t 周生产总工时限制。

另外，本题不再要求生产周期开始时没有任何组件库存，周期结束后也不留下任何组件库存。因此，问题一中式7边界条件失效。

$$y_0^r = y_T^r = 0, \quad r = 1, 2, \cdots, R. \tag{18}$$

综上，得到线性模型如下：

$$
\begin{aligned}
\min \quad & z=\sum_{t=1}^{T}\sum_{r=1}^{R}\left(s^r\omega_t^r+h^r y_t^r\right) \\
\text{s.t.} \quad & \textstyle\sum_{r=1}^{R} M_t^r \leqslant M_t(1-\mu_t+0.1\mu_{t-1}+0.08\mu_{t-2}+0.06\mu_{t-3}+0.04\mu_{t-4}+0.0\mu_{t-5}); \\
& M_t^r \leqslant M_t^r\omega_t^r, && t=1,2,\cdots,T, r=1,2,\cdots,R; \\
& y_t^r = y_{t-1}^r + x_t^r - \eta_r^r x_t^{r'}, && t=1,2,\cdots,T, r=1,2,\cdots,R; \\
& \mu_t \in \{0,1\}; \\
& \textstyle\sum_{t=1}^{210}\mu_t = 7; \\
& \textstyle\sum_{t=i-5}^{i}\mu_t \leqslant 1, && i=6,7,\cdots,210; \\
& \omega_t^r \in \{0,1\}, && t=1,2,\cdots,T, r=1,2,\cdots,R; \\
& x_t^r = x_t^r\omega_t^r, && t=1,2,\cdots,T, r=1,2,\cdots,R; \\
& c^r x_t^r = M_t^r\omega_t^r, && t=1,2,\cdots,T, r=1,2,\cdots,R; \\
& M_t^r \geqslant 0, && r=1,2,\cdots,R; \\
& x_t^r, y_t^r \geqslant 0, && t=1,2,\cdots,T, r=1,2,\cdots,R; \\
& y_{t-1}^{\text{WPCR}} + x_t^{\text{WPCR}} - y_t^{\text{WPCR}} = d_t, && t=1,2,\cdots,T; \\
& \eta_{r'}^r x_t^{r'} \leqslant y_{t-1}^r, && t=1,2,\cdots,T, r=1,2,\cdots,R;
\end{aligned} \tag{19}
$$

5.2 检修方案展示

表 **2** 检修日期及总成本

检修日安排							总成本
第 1 次	第 2 次	第 3 次	第 4 次	第 5 次	第 6 次	第 7 次	5.3×10^6
1	37	44	51	65	86	210	

参考文献

[1] 刘进芬; 王志娜; 赵余婷; 刘斌; 智能管道清理机器人[J]. 物联网技术, 2021, 11(11): 96-97+100. DOI: 10.16667/j.issn.2095-1302.2021.11.028.

[2] 朱多彪; 李龙; 沈云; 基于升力法的贯流式水轮机叶片设计及可行性分析[J]. 水电能源科学, 2013, 31(07): 158-161.

附录 A　源程序

1.1　ELOL.py

```python
# %%
#引入序列长度
from dataclasses import dataclass
import numpy as np
import pandas as pd
import torch
from PyEMD import EMD
import matplotlib.pyplot as plt
from IPython.display import clear_output
import pickle
import os


#时间序列类
#传入数据，返回一个指定长度的
class DataSeq:
    def __init__(self, dataSet:np.array, step:int):
        self.data = dataSet
        self.step = step
        self.len = len(self.data) - self.step + 1

    def __getitem__(self,index,step=None):
        if step == None:
            step = self.step
        if isinstance(index,slice):
            return self.getkeys(index,step)
        return self.getkey(index,step)

    def getkey(self,index,step):
        data = self.data[index : index + step]
        assert len(data) == self.step, f'detaData out of index! length is {self.len} but index
            is {index}'
        return data

    def getkeys(self,indexSlice,step):
        start,stop = indexSlice.start, indexSlice.stop
        if start == None :
            start = 0
        if stop == None:
            stop = self.len
        else:
            stop = stop - 1
```

```python
        ls = []
        for index in range(start,stop+1):
            ls.append(self.getkey(index,step))
        datas = np.array(ls)
        return datas

    def __len__(self):
        return self.len

    def __str__(self):
        return str(self.data)



class DateData(torch.utils.data.Dataset):

    def __init__(self,dataSet:np.array,length = 256, imf_num = 4-1, pre_num=1,
        dataSetWindows=1000):
        super().__init__()

        self.rawData = dataSet.copy()
        self.rawLen = len(self.rawData)
        step = min(self.rawLen, dataSetWindows) - length
        self.step = step
        dataSet = dataSet[-dataSetWindows:]

        #获取 imf_num数
        self.X = DataEMD(dataSet[:-pre_num],length,step,imf_num)
        self.imf_num = min(self.X.imf_num - 1, imf_num)
        self.Data = DataEMD(dataSet,length,step,self.imf_num)

        self.len = len(self.Data) - 1
        self.params = [length , self.imf_num, pre_num, dataSetWindows]

    def __getitem__(self,index):
        if isinstance(index,slice):
            assert index.stop is None or len(self.X) >= index.stop, 'detaData out of index!'
        else:
            assert self.len >= index, f'detaData out of index! length is {self.len} but index is
                {index}'
        return self.Data[index].astype(np.float32), self.Data[index+1].astype(np.float32)

    def __len__(self):
        return self.len

    def update(self,newData):
        dataSet = np.concatenate([self.rawData,newData],axis=0)
```

```python
        self.__init__(dataSet,*self.params)

    def copy():
        pass

# %%
class DataEMD(DataSeq):
    def __init__(self, dataSet:np.array,length:int, step:int,imf_num=-1,emd = EMD()):
        self.rawData = dataSet
        data = emd(self.rawData,max_imf=imf_num).transpose(1,0)
        all_length,self.imf_num = data.shape
        super().__init__(DataSeq(data,length), step)
        self.emd = emd

    def update(self,data):
        pass


# %%
def re_EMD(data):
    batch, step, length, imf_num = data.shape
    ls = []
    for i in range(imf_num):
        chose = Chose_Y(i)
        temp_y = get_num(chose(data))
        ls.append(temp_y)
    d = np.concatenate(ls,axis=1)

    return d.sum(axis=1)


# %%
def get_num(Y):
    return Y.cpu().detach().numpy()

# %%
class Chose(torch.nn.Module):
    def __init__(self, imf):
        super().__init__()
        self.imf = imf

    def forward(self,X):
        #input = b * s * l * imf
        y = X[:,:,:,self.imf]
        O = y.transpose(1,0)
        return O
```

```python
class Chose_Y(torch.nn.Module):
    def __init__(self, imf,length=1):
        super().__init__()
        self.imf = imf
        self.length = length

    def forward(self,X):
        #input = b * s * l * imf
        b,s,l,imf = X.shape
        y = X[:,:,-self.length:,self.imf]
        O = y.reshape(b*s,-1)
        return O


class MyLstm_reg(torch.nn.Module):
    def __init__(self,length,hidden, layer=2,out_num = 1):
        super().__init__()
        self.LSTM = torch.nn.LSTM(length,hidden,num_layers=layer)
        self.state = None
        self.linear = torch.nn.Linear(hidden,out_num)

    def forward(self,X):

        y, self.states= self.LSTM(X)

        s,b,l = y.shape
        h = y.reshape(s*b,l)
        o = self.linear(h)
        return o


class add_net(torch.nn.Module):
    def __init__(self,axis = 2):
        super().__init__()
        self.axis = axis

    def forward(self,X):
        """
        X = b * s * imf * hidden
        """
        return X.sum(axis = self.axis)

# %%
class Trans:

    # 标准化类，默认使用正态标准化
    def __init__(self, trans_fn = None, re_trans_fn = None):
```

```python
        self.re_state = False
        if trans_fn == None:
            self.trans_fn = self._stand
            self.re_trans_fn = self._re_stand
        elif re_trans_fn != None:
            self.trans_fn = trans_fn
            self.re_trans_fn = re_trans_fn
        else:
            RuntimeError('没有传入恢复函数！')

    def _stand(self,data):
        if self.re_state == False:
            self.re_trans_params = [data.mean(),data.std()]
            self.re_state = True
        new_data = (data - self.re_trans_params[0]) / (self.re_trans_params[1])
        return new_data
    def _re_stand(self,data):
        temp_data = data * (self.re_trans_params[1])
        re_data = temp_data + self.re_trans_params[0]
        return re_data

    def _max_min(self, data):
        if self.re_state == False:
            self.re_trans_params = [data.min(),data.max()]
            self.re_state = True
        new_data = (data - self.re_trans_params[0]) / (self.re_trans_params[1] -
             self.re_trans_params[0])
        return new_data

    def _re_max_min(self,data):
        temp_data = data * (self.re_trans_params[1] - self.re_trans_params[0])
        re_data = temp_data + self.re_trans_params[0]
        return re_data

# %%
def grad_clipping(net, theta): #@save
    """裁剪梯度"""
    if isinstance(net, torch.nn.Module):
        params = [p for p in net.parameters() if p.requires_grad]
    else:
        params = net.params
    norm = torch.sqrt(sum(torch.sum((p.grad ** 2)) for p in params))
    if norm > theta:
        for param in params:
            param.grad[:] *= theta / norm

# %%
```

```python
def train_begin(net, data_iter, epoch, imf, device, lr=0.01,opim_fn = torch.optim.Adam,out_num
     = 1000, show_pic = True):
    # 初始化网络，在初次拟合训练时使用
    ls = []
    #迁移至GPU
    net.to(device)
    #初始 chose_Y 并选择 imf
    chose = Chose_Y(imf)
    opimter = opim_fn(net.parameters(),lr)

    loss = torch.nn.MSELoss()
    X,Y = next(iter(data_iter))

    trans = Trans()
    y = chose(Y)
    y_new = trans.trans_fn(y)
    out_time = torch.log(y_new.std() * y_new.abs().mean()/(500) + 1)

    for i in range(epoch):
        one_temp = []
        for X,Y in data_iter:
            X_new = trans.trans_fn(X)
            Y_new = trans.trans_fn(Y)
            yhat = net(X_new.to(device))
            y1 = chose(Y_new.to(device))
            l = loss(yhat[5:],y1[5:])
            if i> out_num and get_num(out_time) > get_num(l):
                return ls,trans
            opimter.zero_grad()
            l.backward()
            grad_clipping(net,1)
            opimter.step()
            one_temp.append(l)
            #ls.append(l)
        one_temp = torch.stack(one_temp,dim=0)
        ls.append(one_temp.max())

        if i % 20 == 0 and show_pic == True:
            clear_output(wait=True)
            print('out_time is loss less than',out_time,'and i is',i)
            plt.cla()
            temp = torch.stack(ls,dim=0)
            plt.plot(get_num(temp)[-100:])
            plt.show()

            plt.plot(get_num(y1)[:],'r')
            plt.plot(get_num(yhat)[:],'b',alpha=0.4)
```

```python
        plt.show()
    return ls,trans

# %%
def get_net(imf,length = 64 ,hidden=256):
    #返回一个网络
    net = torch.nn.Sequential(
    Chose(imf),
    MyLstm_reg(length,hidden=hidden)
    )
    return net

# %%
def load_nets_and_trans(nets_path_ls, trans_path,length=64,hidden=256):
    nets = []
    imf_num = len(nets_path_ls)
    with open(trans_path,'rb') as f:
        trans_ls = pickle.load(f)
    for i in range(imf_num):
            net = get_net(i,length=length,hidden=hidden)
            net.load_state_dict(torch.load(nets_path_ls[i]))
            nets.append(net)
    return nets, trans_ls

# %%
def train_all_net(data_iter, imf_num, device, lr = 0.0005, min_epoch = 800, max_epoch = 10000,
     root_path='net',net_suffix='_lstm.pkl',trans_suffix= 'trans_ls.info',
              opim_fn = torch.optim.Adam,show_pic = False, length=64, hidden=256):
    nets= []
    trans_ls = []
    for i in range(imf_num):
        temp_net = get_net(i,length, hidden)
        ls, trans_one = train_begin(temp_net, data_iter, max_epoch,
                                i,device, lr = lr, out_num = min_epoch,opim_fn=opim_fn,
                                show_pic = show_pic)
        trans_ls.append(trans_one)
        torch.save(temp_net.state_dict(),root_path+ f'\\{i}'+net_suffix)
        net = get_net(i,length, hidden)
        net.load_state_dict(torch.load(root_path+ f'\\{i}'+net_suffix))
        nets.append(net)
    dump = pickle.dumps(trans_ls)
    with open(root_path+ '\\'+trans_suffix,'wb') as f:
        f.write(dump)
    return nets, trans_ls


# %%
```

```python
def update_net(data_iter, nets, imf_num, device, lr = 0.0005, min_epoch = 800, max_epoch = 
    10000, root_path='net',net_suffix='_lstm.pkl',trans_suffix= 'trans_ls.info',
             opim_fn = torch.optim.Adam, show_pic = False, dump_local = False, length=64,
                 hidden=256):
    for net in nets:
        net.train()
    trans_ls = []
    for i in range(imf_num):
        ls, trans_one = train_begin(nets[i], data_iter, max_epoch, i,device,
                                lr = lr,out_num = min_epoch,opim_fn=opim_fn, show_pic= show_pic)
        trans_ls.append(trans_one)
        if dump_local == True:
            torch.save(nets[i].state_dict(),root_path+ f'\\{i}'+net_suffix)
            net = get_net(i, length, hidden)
            net.load_state_dict(torch.load(root_path+ f'\\{i}'+net_suffix))
            nets[i] = net

    if dump_local == True:
        dump = pickle.dumps(trans_ls)
        with open(root_path+ '\\'+trans_suffix,'wb') as f:
            f.write(dump)
    return nets, trans_ls

# %%
def predict_one(nets,trans_ls,data_iter):
    for net in nets:
        net.eval()
        net.to('cpu')
    X,Y = next(iter(data_iter))
    new_in = torch.cat([X,Y[0:1,-2:-1]],dim=1)
    imf_num = len(nets)

    pred_Y = torch.zeros_like(nets[0](new_in))

    for i in range(imf_num):
        trans_in_i = trans_ls[i].trans_fn(new_in)
        trans_Y = nets[i](trans_in_i)
        pred_Y += trans_ls[i].re_trans_fn(trans_Y)
    return pred_Y




# %%
class ELOL:
    """
    Emd LSTM OnLine Learning Module
    """
```

```python
def __init__(self,length,imf_num,hidden,rawData, device, pre_num=1, dataSetWindows=1000):

    self.length = length

    self.hidden = hidden
    self.data = DateData(rawData, length, imf_num-1, pre_num=pre_num, dataSetWindows =
        dataSetWindows)
    self.imf_num = self.data.imf_num
    self.data_iter = torch.utils.data.DataLoader(self.data, batch_size = 1)
    self.device = device

def init_nets(self, lr = 0.0005, min_epoch = 2000,
            max_epoch = 10000, root_path='net',
            net_suffix='_lstm.pkl',trans_suffix= 'trans_ls.info',
            opim_fn = torch.optim.Adam,show_pic = True):

    try:
        os.mkdir(root_path)
    except:
        print(f'文件夹 {root_path} 已经存在……开始训练网络')
    self.nets, self.trans_ls = train_all_net(self.data_iter,self.imf_num + 1, self.device,
                                    lr = lr, min_epoch = min_epoch,
                                    max_epoch = max_epoch, root_path = root_path,
                                    net_suffix = net_suffix,trans_suffix= trans_suffix,
                                    opim_fn = opim_fn,show_pic = show_pic,
                                        length=self.length, hidden = self.hidden)



def load_nets_and_trans(self, nets_path, trans_path):
        self.nets, self.trans_ls = load_nets_and_trans(nets_path,trans_path,
                                            length=self.length, hidden= self.hidden)



def update_data_and_net(self,data , lr = 0.005,
                min_epoch = 500, max_epoch = 10000,
                root_path='net',net_suffix='_lstm.pkl',
                trans_suffix= 'trans_ls.info',
                opim_fn = torch.optim.Adam,
                show_pic = False, dump_local = False):
    self.data.update(data)
    self.data_iter = torch.utils.data.DataLoader(self.data, batch_size = 1)
    self.nets, self.trans_ls = update_net(self.data_iter, self.nets, self.imf_num+1,
        self.device,
                                lr=lr, min_epoch= min_epoch, max_epoch=max_epoch,
                                root_path=root_path, net_suffix=net_suffix, trans_suffix=
                                    trans_suffix,
                                opim_fn=opim_fn, show_pic= show_pic,
```

```
                                                   dump_local=dump_local,length=self.length,
                                                   hidden=self.hidden)


    def predict(self):
        self.pred_Y = predict_one(self.nets,self.trans_ls,self.data_iter)
        return get_num(self.pred_Y[-1])
```

1.2 header.py

```
from ELOL import MyLstm_reg,pd,torch,Trans,get_num,grad_clipping,clear_output,plt,np
import pickle

def pre_ABC(x,data_index = 2):
    if x['材料分类'] == "A":
        x[data_index:] = x[data_index:] / 0.6
    if x['材料分类'] == "B":
        x[data_index:] = x[data_index:] / 0.66
    if x['材料分类'] == "C":
        x[data_index:] = x[data_index:] / 0.72
    return x

def re_pre_ABC(x,data_index = 2):
    if x['材料分类'] == "A":
        x[data_index:] = x[data_index:] * 0.6
    if x['材料分类'] == "B":
        x[data_index:] = x[data_index:] * 0.66
    if x['材料分类'] == "C":
        x[data_index:] = x[data_index:] * 0.72
    return x

# %%
class DataSeq:
    def __init__(self, dataSet:np.array, step:int):
        self.data = dataSet
        self.step = step
        self.len = len(self.data) - self.step + 1

    def __getitem__(self,index,step=None):
        if step == None:
            step = self.step
        if isinstance(index,slice):
            return self.getkeys(index,step)
        return self.getkey(index,step)
```

```python
    def getkey(self,index,step):
        data = self.data[index : index + step]
        assert len(data) == self.step, f'detaData out of index! length is {self.len} but index
            is {index}'
        return data

    def getkeys(self,indexSlice,step):
        start,stop = indexSlice.start, indexSlice.stop
        if start == None :
            start = 0
        if stop == None:
            stop = self.len
        else:
            stop = stop - 1
        ls = []
        for index in range(start,stop+1):
            ls.append(self.getkey(index,step))
        datas = np.array(ls)
        return datas

    def __len__(self):
        return self.len

    def __str__(self):
        return str(self.data)

# %%
class PureData(torch.utils.data.Dataset):
    def __init__(self,X,Y):
        super().__init__()
        self.X = X
        self.Y = Y
        self.len = len(X)

    def __getitem__(self,index,step=None):
        return self.X[index],self.Y[index]
    def __len__(self):
        return self.len


# %%
def train_begin(net, data_iter, epoch, device, lr=0.01,opim_fn = torch.optim.Adam,out_num =
    1000, show_pic = True):
    # 初始化网络, 在初次拟合训练时使用
    ls = []
    #迁移至GPU
    net.to(device)
```

```
    #初始 chose_Y 并选择 imf
    opimter = opim_fn(net.parameters(),lr)

    loss = torch.nn.MSELoss()
    X,Y = next(iter(data_iter))

    trans = Trans()
    y_new = trans.trans_fn(Y)
    out_time = torch.log(y_new.std() * y_new.abs().mean()/(500) + 1)

    for i in range(epoch):
        one_temp = []
        for X,Y in data_iter:
            X_new = trans.trans_fn(X)
            Y_new = trans.trans_fn(Y)
            yhat = net(X_new.to(device))
            y1 = Y_new.to(device)
            l = loss(yhat[5:],trans_shape(y1)[5:])
            if i> out_num and get_num(out_time) > get_num(l):
                return ls,trans
            opimter.zero_grad()
            l.backward()
            grad_clipping(net,1)
            opimter.step()
            one_temp.append(l)
            #ls.append(l)
        one_temp = torch.stack(one_temp,dim=0)
        ls.append(one_temp.max())

        if i % 20 == 0:
            clear_output(wait=True)
            print('out_time is loss less than',out_time,'and i is',i)
            if show_pic == True:
                plt.cla()
                temp = torch.stack(ls,dim=0)
                plt.plot(get_num(temp)[-100:])
                plt.show()

                plt.plot(get_num(trans_shape(y1))[:],'r')
                plt.plot(get_num(yhat)[:],'b',alpha=0.4)
                plt.show()
    return ls,trans

# %%
def trans_shape(Y):
    b,s,l = Y.shape
    y = Y.reshape(b*s,-1)[:,-1].reshape(-1,1)
```

```python
    return y


# 使用选定的条目筛选项目
def filter_item(data, filter_list,key='供应商ID'):
    temp_data = data.copy()
    indexs = data[key]
    indexs.name=None
    temp_data.index = indexs
    output_data = temp_data.loc[filter_list,:]
    return output_data.reset_index(drop=True)

#传入一个数据框和总和，按顺序取到所有累加值，直至等于总和
def sort_and_sub(data,total):
    def cumsum_to_sub(temp_one):
        temp_index = temp_one[1:].cumsum() > temp_one['temp']
        temp_two = temp_one[1:].copy()
        temp_two.loc[temp_index] = 0
        idx_max = len(temp_two[temp_index==False])
        if idx_max < len(temp_two):
            temp_two.loc[idx_max] = temp_one['temp'] - temp_two.sum()
        return temp_two
    temp_data = data[:]
    temp_total = total[:]
    temp_total.index = temp_data.columns
    temp_total.name='temp'
    new_temp = pd.concat([pd.DataFrame(temp_total).T,temp_data],axis=0)
    temp_temp = new_temp.apply(cumsum_to_sub)
    return temp_temp

#删除Nan
def drop_nan(serise):
    temp_serise = serise.copy()
    nan_index = (pd.isna(temp_serise)!=True)
    return temp_serise.loc[nan_index]

def set_plt_size(long=12,high=8):
    plt.rcParams['figure.figsize'] = (long,high)



def predict_product(data,pred_step = 48, length = 48 , step = 24, batchSize = 12, hidden = 256
    , max_epoch = 1000, min_epoch = 150, device = 'cpu',show_pic = True):
    temp_data = np.array(data,dtype=np.float32)
    dataX = DataSeq(DataSeq(temp_data[:-1],length),step)
    dataY = DataSeq(DataSeq(temp_data[1:],length),step)
    data = PureData(dataX,dataY)
```

```
    data_iter = torch.utils.data.DataLoader(data,batchSize,shuffle=True)
    X,Y = next(iter(data_iter))

    net = MyLstm_reg(length, hidden)
    ls,trans = train_begin(net,data_iter,max_epoch,device,out_num=min_epoch,show_pic=show_pic)
    new_temp = torch.tensor(temp_data[-(length+step):])
    new_temp = trans.trans_fn(new_temp).numpy()

    for i in range(pred_step):
        dataX = DataSeq(DataSeq(new_temp[:-1],length),step)
        dataY = DataSeq(DataSeq(new_temp[1:],length),step)
        data = PureData(dataX,dataY)
        data_iter = torch.utils.data.DataLoader(data,batchSize)
        X,Y = next(iter(data_iter))
        X_new = Y
        yhat = net(X_new.to(device))
        new_temp = np.concatenate((new_temp,get_num(yhat[-1])))[-(length+step):]

    one = torch.tensor(new_temp)
    two = trans.re_trans_fn(one)
    return two,ls

#常数
gongying = "附件1 近5年402家供应商的相关数据.xlsx"
data_order = pd.read_excel(gongying,'企业的订货量')
data_supply = pd.read_excel(gongying,'供应商的供货量')

device = 'cuda:0'
```

1.3 main.py

```
# %%
from header import *
import pulp
from random import choice

# %%
df_pred = pd.read_csv('pred.csv').drop(columns='Unnamed: 0')

# %%
#获取分类数据
data = pd.read_csv('temp.csv')
index = data[data['选中'] == 1]['Unnamed: 0'].values

df_class = filter_item(data_order.loc[:,['供应商ID','材料分类']],index)
```

```python
new_df_pred = pd.concat([df_class['材料分类'],df_pred],axis=1)

# %%
new_df_pred.apply(re_pre_ABC,axis=1).sum(axis=0)

# %%
#new_df_pred = new_df_pred.apply(re_pre_ABC,axis=1)

# %%
new_df_pred = new_df_pred.iloc[:,:2+24]

# %%
#预测的最大供货量的和
pred_sum = new_df_pred.iloc[:,2:].sum(axis=0)
#去除供应商id和类别后预测的最大供应量
pred_pure = new_df_pred.iloc[:,2:]
#大于2.84的索引
low_index = (pred_sum -2.84e4 < 0)
#小于2.84的索引
big_index = (pred_sum -2.84e4 > 0)

# %%
Storage_cost = 0.1
gross_profit = 1
# 开始准备线性规划数据

big_num_index = pred_sum.index.to_list()

myProblem = pulp.LpProblem('订购数量规划',sense=pulp.LpMaximize)

D_w = pred_sum.copy()
D_w.loc[big_index] = 0
D_w.loc[low_index] = 2.84e4 - D_w[low_index]
D_w = D_w.tolist()

d_w_constraint = pred_sum.copy()
d_w_constraint.loc[low_index] = 0
d_w_constraint.loc[big_index] = d_w_constraint[big_index] - 2.84e4
d_w_constraint = d_w_constraint.tolist()

d_var = pulp.LpVariable.dicts(name='d_w',indices=big_num_index,lowBound=0)

u = pulp.LpVariable.dicts(name='u_w',indices=big_num_index,lowBound=0)
for i in range(len(big_num_index)):
   u[str(i)] = 2.84e4 + pulp.lpSum([d_var[str(j)] for j in range(i+1)]) - sum(D_w[:i+1])
```

```
#定义目标
myProblem += gross_profit * pulp.lpSum(d_var) - Storage_cost * pulp.lpSum(u)

#定义约束

for i in range(len(big_num_index)):
    myProblem += (d_var[str(i)] <= d_w_constraint[i])

for i in range(len(big_num_index)):
    myProblem += (u[str(i)] >= 2.84e4*2)

myProblem += pulp.lpSum(d_var) <= sum(D_w) + 2.84e4

#solve
myProblem.solve()

# %%
#检查输出
ls = []
for v in u.values():
    print(v.name, "=", v.value())
    ls.append(v.value())
print(sum(ls))

for v in d_var.values():
    print(v.name, "=", v.varValue)

# %%
ls = []
for v in d_var.values():
    #print(v.name, "=", v.varValue)
    ls.append(v.value())
#预测的供货量
pred_order = pd.Series(ls)
#预测的最大供货量的和的复制
pred_order_temp = pred_sum.copy()
#经过加上2.84e4的遮蔽运算，得到的是每周预计供货量
pred_order_temp.loc[big_index] = 2.84e4
pred_order = pd.Series(pred_order_temp.values + pred_order.values)

# %%
pred_order

# %%
#倒序排列后的预测最大供应量数据，包含供应商ID和类别
temp = new_df_pred.iloc[::-1,:].reset_index(drop=True)
temp_pure = temp.iloc[:,2:]
```

```
out = pd.concat([temp.iloc[:,:2],sort_and_sub(temp_pure,pred_order)],axis=1)
out.to_csv('订购方案.csv',index=None)



# %%
#获取分数
index = data[data['选中'] == 1]['Unnamed: 0'].values
Fraction = pd.read_csv('总表.csv').drop(columns='Unnamed: 0')
Fraction = filter_item(Fraction,index)
Fraction = Fraction['综合排分'].values

# %%
pd.read_csv('总表.csv').drop(columns='Unnamed: 0')

# %%

#获取偏差
index = data[data['选中'] == 1]['Unnamed: 0'].values

data_order = filter_item(data_order, index)
data_supply = filter_item(data_supply, index)

pure_race_data = ((data_supply.iloc[:,2:] - data_order.iloc[:,2:]) / data_order.iloc[:,2:])
pure_race_data = pure_race_data.fillna(0)

race_mean = pure_race_data.sum(axis=1) / (data_order.iloc[:,2:] !=0).sum(axis=1)
#取偏差绝对值
race_mean_abs = race_mean

pred_order_index = pred_order[(pred_order != 0)].index.to_list()
pred_order_pure = pred_pure.loc[:,[str(i) for i in pred_order_index]]
pred_order_lp = pred_order[pred_order_index]

# %%
# 开始准备线性规划数据

def get_week_order(week):

    pred_order_pure_one_week = pred_order_pure[str(week)].tolist()

    pred_order_lp_one_week = pred_order_lp[week].tolist()

    myProblem1 = pulp.LpProblem('订购数量分配规划',sense=pulp.LpMaximize)

    z_var =
        pulp.LpVariable.dicts(name='z_w_'+str(week)+'_n',indices=range(len(pred_order_pure_one_week)),lowBound=0
```

```python
    #定义目标
    myProblem1 += pulp.lpSum([Fraction[i] * z_var[i] for i in
        range(len(pred_order_pure_one_week))])

    #定义约束

    #myProblem1 += pulp.lpSum([z_var[i]*(1/(1+race_mean[i])) for i in
        range(len(pred_order_pure_one_week))]) == 2.84e4 + pred_order_lp_one_week
    myProblem1 += pulp.lpSum([z_var[i]*((1+race_mean[i])) for i in
        range(len(pred_order_pure_one_week))]) ==pred_order_lp_one_week

    #print(pred_order_lp_one_week)

    for i in range(len(pred_order_pure_one_week)):
        myProblem1 += z_var[i]*(1/(1+race_mean[i])) <= pred_order_pure_one_week[i]

    myProblem1.solve()

    ls = []
    i=0
    for v in z_var.values():
        #print(v.name, "=", v.varValue,'pred_order = ',pred_order_pure_one_week[i],'race
            =',race_mean[i])
        ls.append(v.value())
        i += 1

    #print(pulp.lpSum([z_var[i]*((1+race_mean[i])) for i in
        range(len(pred_order_pure_one_week))]).value() )

    out = pd.Series(ls,name=f'W{week}')

    return out

# %%
out = []
for i in range(len(big_num_index)):
    if i in pred_order_index:
        out.append(get_week_order(i))
    else:
        out.append(pred_pure[str(i)].rename(f'W{i}'))

# %%
out_order = pd.DataFrame(out).T
out_order.index = data[data['选中'] == 1]['Unnamed: 0'].values

# %%
```

```python
out_order.values.sum()

# %%
(out_order.values*(race_mean.values+1).reshape(44,-1)).sum()

# %%
# 添加分类数据，方便还原
out_order.insert(0,"材料分类",new_df_pred["材料分类"].values)
out_order

# %%
#temp = (out_order.values*(race_mean.values+1).reshape(44,-1))
#pd.DataFrame(temp)

# %%
# 先还原，后导出
out_order.apply(lambda x:re_pre_ABC(x,data_index=1),axis=1).to_csv('order_24_week.csv')

# %%
#获取比率new_pred_pure_race_data
index = data[data['选中'] == 1]['Unnamed: 0'].values

data_order = filter_item(data_order, index)
data_supply = filter_item(data_supply, index)

pred_pure_race_data = (data_supply.iloc[:,2:] - data_order.iloc[:,2:]) / data_order.iloc[:,2:]

new_pred_pure_race_data = pred_pure_race_data.copy()

#储存到 dict中，方便调用
supplyer, week = new_pred_pure_race_data.shape
race_dict = {}
for i in range(supplyer):
   race_dict[i] = drop_nan(new_pred_pure_race_data.loc[i] + 1).tolist()

#设置函数方便抽取随机数

def get_all_race_random():
   ls = []
   for i in race_dict.values():
      ls.append(choice(i))
   return ls

# %%
race_mean.values+1

# %%
```

```python
def set_plt_size(long=12,high=8):
    plt.rcParams['figure.figsize'] = (long,high)

# %%
2**16

# %%
pd.DataFrame(out_order.values * (race_mean.values+1).reshape(44,-1)).sum(axis=0).cumsum() +
    2.84e4 - pd.Series([x*2.84e4 for x in range(1,24+1)])

# %%
pd.DataFrame(out_order.values * (race_mean.values+1).reshape(44,-1)).sum(axis=0).cumsum() +
    2.84e4

# %%
out_order.iloc[:,1:].values * (race_mean.values+1).reshape(44,-1)

# %%
def 动态库存量(df):
    sum_temp = df.sum(axis=0).cumsum()+2.84e4
    temp = pd.Series([x*2.84e4 for x in range(1,len(sum_temp) +1)])
    return sum_temp - temp
动态库存量(pd.DataFrame(out_order.iloc[:,1:].values * (race_mean.values+1).reshape(44,-1)))

# %%
动态库存量(pd.DataFrame(out_order.iloc[:,1:].values *
    (race_mean.values+1).reshape(44,-1))).values

# %%


# %%
# 绘制预测供货量与实际供货量的仿真
sum_ls = []
set_plt_size()
for ssjds in range(32):
    weeks = out_order.columns
    simulation = []
    for week in weeks:
        race_week = get_all_race_random()
        simulation.append(out_order[week] * race_week)

    out_simulation = pd.DataFrame(simulation).T

    #out_simulation.sum(axis=0).plot(alpha=0.4,color='gray',linewidth=0.05)
    out_simulation.sum(axis=0).plot(alpha=0.1,color='gray',linewidth=0.5)
```

```python
    sum_ls.append(out_simulation.sum().sum()/24)
pd.DataFrame(out_order.values *
     (race_mean.values+1).reshape(44,-1)).sum().plot(color='r',linewidth=2)
#plt.savefig('fig/预测仿真.png',dpi=320)

# %%
max(12,1,122)

# %%
def 动态库存量(df):
    sum_temp = df.sum(axis=0)
    length = len(sum_temp)
    temp = 2.84e4
    ls = []
    for i in range(length):
        temp = max((temp + sum_temp[i] - 2.84e4) , 0)
        ls.append(temp)
    return pd.Series(ls)
sum_ls = []
set_plt_size()
for ssjds in range(3200):
    weeks = out_order.columns
    simulation = []
    for week in weeks:
        race_week = get_all_race_random()
        simulation.append(out_order[week] * race_week)

    out_simulation = pd.DataFrame(simulation).T

    #out_simulation.sum(axis=0).plot(alpha=0.4,color='gray',linewidth=0.05)
    #动态库存量(out_simulation).plot(alpha=0.1,color='gray',linewidth=0.5)
    动态库存量(out_simulation).plot(alpha=0.4,color='gray',linewidth=0.05)

    sum_ls.append(out_simulation.sum().sum()/24)
动态库存量(pd.DataFrame(out_order.values *
     (race_mean.values+1).reshape(44,-1))).plot(color='r',linewidth=2)
plt.plot([0 for i in range(24)],color='blue',linewidth=1)
plt.savefig('fig/预测库存量仿真.png',dpi=320)

# %%
动态库存量(pd.DataFrame(out_order.values * (race_mean.values+1).reshape(44,-1)))

# %%
[0 for i in range(24)]

# %%
sum_ls = []
```

```python
set_plt_size()
for ssjds in range(320):
    weeks = out_order.columns
    simulation = []
    for week in weeks:
        race_week = get_all_race_random()
        simulation.append(out_order[week] * race_week)

    out_simulation = pd.DataFrame(simulation).T

    #out_simulation.sum(axis=0).plot(alpha=0.4,color='gray',linewidth=0.05)
    out_simulation.sum(axis=0).plot(alpha=0.1,color='gray',linewidth=0.5)

    sum_ls.append(out_simulation.sum().sum()/24)
pd.DataFrame(out_order.values *
     (race_mean.values+1).reshape(44,-1)).sum().plot(color='r',linewidth=2)
plt.savefig('fig/预测仿真.png',dpi=320)

# %%
(pd.DataFrame(out_order.values * (race_mean.values+1).reshape(44,-1)).sum().sum())/24

# %%
(sum(sum_ls))/320
```

1.4 work1.py

```python
# %%
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# %%
def get_bool(data,bool):
    return data[bool]
def sort_and_plot(data):
    return data.sort_values().reset_index(drop=True).plot()

# %%
#归一化方案
def Normal(x):
    return (x-x.mean()) / x.std()
def Min_Max(x):
    return (x-x.min()) / (x.max() - x.min())

# %%
```

```python
def plot_and_save(data:pd.DataFrame, path:str, size=(24.0, 16.0)):
    plt.rcParams['figure.figsize'] = size
    sort_and_plot(data)
    plt.savefig(path)
    plt.cla()

# %%
gongying = "附件1 近5年402家供应商的相关数据.xlsx"
转运商 = "附件2 近5年8家转运商的相关数据.xlsx"

# %%
data_order = pd.read_excel(gongying,'企业的订货量')
data_supply = pd.read_excel(gongying,'供应商的供货量')

# %%
test_data = data_order.copy()

# %%
data_item_num = test_data.iloc[:,2:]

# %%
test_data['订货次数']=(data_item_num>0).sum(axis=1)
test_data['订货总量'] = data_item_num.sum(axis=1)
test_data['供货总量'] = data_supply.iloc[:,2:].sum(axis=1)

# %%
data_sub = data_item_num-data_supply.iloc[:,2:]

# %%
test_data['平均供货偏差'] = (((data_sub / data_item_num).abs().fillna(0)).sum(axis=1) /
     test_data['订货次数'])
test_data['单次最大供应量'] = data_supply.iloc[:,2:].max(axis=1)

# %%
test_data.iloc[:,-5:]

# %%
targets = test_data.columns[-5:]

# %%
for target in targets:
    plot_and_save(test_data[target],'fig/'+target+'.png',size=(12.0,8))

# %%
data_change = test_data.iloc[:,-5:]

# %%
```

```
for i in [ '订货总量', '供货总量', '单次最大供应量']:
    data_change[i] = np.log(data_change[i].values)

# %%
data_change = data_change.apply(Min_Max)

# %%
data_change

# %%
data_temp = np.asarray(data_change[['订货次数', '订货总量', '供货总量', '平均供货偏差',
     '单次最大供应量']])
#计算熵值
k = -1/np.log(402)
data_log= data_temp*np.log(data_temp)
data_log = pd.DataFrame(data_log)
data_log=data_log.fillna(0)
data_log=data_log.values
ls=[]
#计算变异指数
for i in range(5):
    e_j=k*data_log.sum(axis=0)[i]
    ls.append(e_j)
temp_list =[]
for i in ls:
    temp_list.append(1-i)
#计算权重
ls=[]
#删除错误定义
#del(sum)
for i in temp_list:
    ls.append(i/sum(temp_list))


# %%
print(ls,targets)
```

1.5 linear.jl

```
### A Pluto.jl notebook ###
# v0.19.9

using Markdown
using InteractiveUtils
```

```
# This Pluto notebook uses @bind for interactivity. When running this notebook outside of
    Pluto, the following 'mock version' of @bind gives bound variables a default value
    (instead of an error).
macro bind(def, element)
    quote
        local iv = try
            Base.loaded_modules[Base.PkgId(Base.UUID("6e696c72-6542-2067-7265-42206c756150"),
            "AbstractPlutoDingetjes")].Bonds.initial_value catch; b -> missing; end
        local el = $(esc(element))
        global $(esc(def)) = Core.applicable(Base.get, el) ? Base.get(el) : iv(el)
        el
    end
end

#    5928cfb0-1923-11ed-3ace-2bfad8956c7c
using JuMP,Gurobi,DataFrames,CSV,Parsers,Tables,PlutoUI

#    274a1543-f320-4705-8e98-55d6a3fbea74
function Base.filter((title,name)::Tuple, df::DataFrame)
  return filter(title=>x -> x == name, df)
end

#    dda52e10-674b-47ab-b765-aa4acf90780a
function Base.filter((title,name)::Tuple, df::DataFrame,week::Integer)
  return filter(title=>x -> x == name, df)[:,"W"*string(week-1)]
end

#    eb6d766a-bc18-4f21-9945-d2bb387a5e6d
function Base.filter((title,name)::Tuple, df::DataFrame,week::String)
  return filter(title=>x -> x == name, df)[:,week]
end

#    d2e2fc85-3b17-4fce-af9b-5f1d38109691
function Base.sum(l::Vector{Vector{AffExpr}})
  temp = zero(l[1][1])
  for i in l
    temp += sum(i)
  end
  return temp
end

#    243d047b-920d-451c-946e-1fcd45f227b4
function num2weekStr(num)
  return "W"*string(num-1)
end

#    2fd96ad3-0367-49ea-846e-07f6722f82ff
```

```
@bind week Slider(1:24)

#    36ca59d9-2891-43af-bdbb-0a2a4eee5853
md"""
 目前在计算的周: $(week)
"""

#    9cea8136-3a62-4776-ba94-0f1aa0eaaf42
weekStr = num2weekStr(week)

#    500d4a9c-0879-4738-bf22-ad726d90fe9d
@bind limit Slider(1:5)

#    7ec11938-e35a-478c-9af3-5b8a8e0108b6
md"""
目前的转运商允许量: $(limit)
"""

#    4669b138-a94b-43f2-a4c2-0a803f1ca3d0
@bind Transshipment_capacity Slider(6000:500:22000)

#    b1530cee-90e3-483d-86e8-cc54e7c24338
md"""
目前的转运能力: $(Transshipment_capacity)
"""

#    a9ed9185-630c-4c6e-aac5-155900de98f3
md"""
## 导入数据
"""

#    4c41b978-fdd4-4b03-b621-f0753b727eb5
begin
  df_order_24_week = CSV.read("order_24_week (2).csv",DataFrame)
  df_Average_loss_of_forwarders = CSV.read("转运商平均损耗.csv",DataFrame)
  forwarders_id = df_Average_loss_of_forwarders[:,:转运商ID]
end

#    ef66f393-4a3b-4d0a-8d20-dd521b5529cd
filter((:材料分类,"C"),df_order_24_week,weekStr)

#    ccb7017d-eaaf-451e-96f0-76b729ba799c
md"""
# 线性方程求解函数
"""

#    7f084915-1fb5-46aa-bc15-e1d73ed5960e
```

```
function solve_chose_forwarders(week::String,
                    df_order::DataFrame,
                    df_Average_loss::DataFrame,
                    rate_list::Vector,
                    limit::Int64,
                    Transshipment_capacity::Int64)
  #每个材料类别的损耗量
  function sum_model(u ,
               len_of_u::Int64,
               num_of_forwarder::Int64,
               Average_loss::Vector{Float64},
               order::Vector{Float64})
    sum_i = sum([[u [i,j] * Average_loss[j] * order[i] for i in 1:len_of_u] for j in
        1:num_of_forwarder])
    return sum_i
  end

  # 全部损耗量
  function sum_model(U::Vector,
               length_list::Vector,
               num_of_forwarder::Int64,
               Average_loss::Vector{Float64},
               order_list::Vector{Vector{Float64}},
               rate_list::Vector)
    temp=0
    for (u ,len_of_u,order,rate) in zip(U,length_list,order_list,rate_list)
      temp += rate* sum_model(u ,
               len_of_u,
               num_of_forwarder,
               Average_loss,
               order)
    end
    return temp
  end

  #每一个供应商的转运量
  function sum_forwarders(index_of_forwarders::Int64,
                 U::Vector,
                 length_list::Vector,
                 Average_loss::Vector{Float64},
                 order_list::Vector{Vector{Float64}})
    temp = 0
    j = index_of_forwarders
    for (u ,len_of_u,order) in zip(U,length_list,order_list)
      temp += sum([u [i,j] * order[i] for i in 1:len_of_u])
    end
    return temp
```

```
end

# 01约束
function sum_bin(u ,len::Int64,model)
   for i in 1:len
      @constraint(model, sum(u [i,:])<=limit)
   end
end

#运货量等于订货量
function sum_bin(u ,len::Int64,
            model,
            order::Vector{Float64},
            num_of_forwarder::Int64)
   for i in 1:len
      @constraint(model, sum([u [i,j] * order[i] for j in 1:num_of_forwarder]) == order[i])
   end
end

#定义模型
model = Model(Gurobi.Optimizer)

#定义变量
num_of_forwarder=length(df_Average_loss_of_forwarders[:,1])
begin
   length_A = length(filter((:材料分类,"A"),df_order,week))
   length_B = length(filter((:材料分类,"B"),df_order,week))
   length_C = length(filter((:材料分类,"C"),df_order,week))
   @variable(model, u [i = 1:length_A, j= 1:num_of_forwarder],Bin)
   @variable(model, u [i = 1:length_B, j= 1:num_of_forwarder],Bin)
   @variable(model, u [i = 1:length_C, j= 1:num_of_forwarder],Bin)
end

#定义目标函数
@objective(model, Min,
      sum_model([u ,u ,u ],
         [length_A,length_B,length_C],
         num_of_forwarder,
         df_Average_loss[:,:平均损耗],
         [filter((:材料分类,"A"),df_order,week),
            filter((:材料分类,"B"),df_order,week),
            filter((:材料分类,"C"),df_order,week)],
         rate_list))

#定义约束
#01约束
for (u ,len_of_u) in zip([u ,u ,u ],[length_A,length_B,length_C])
```

```
    sum_bin(u ,len_of_u,model)
  end
  #供应商的转运量小于6000
  for j in 1:num_of_forwarder
    @constraint(model,
    sum_forwarders(j,[u ,u ,u ],[15,14,15],
            df_Average_loss[:,:平均损耗],
            [filter((:材料分类,"A"),df_order,week),
            filter((:材料分类,"B"),df_order,week),
            filter((:材料分类,"C"),df_order,week)])
    <=Transshipment_capacity)
  end
  #运货量等于订货量
  for (u ,len_of_u,order) in zip([u ,u ,u ],
                [length_A,length_B,length_C],
                [filter((:材料分类,"A"),df_order,week),
                  filter((:材料分类,"B"),df_order,week),
                  filter((:材料分类,"C"),df_order,week)])
    sum_bin(u ,len_of_u,model,order,num_of_forwarder)
  end
  optimize!(model)
  return model,(u ,u ,u )
end

#    407fe9e3-c524-4b51-a0dd-0229e398f535
function solve_chose_forwarders(week::String,
                  df_order::DataFrame,
                  df_Average_loss::DataFrame)
  solve_chose_forwarders(weekStr,df_order,df_Average_loss,[1.2,1.1,1,1,1],limit,Transshipment_capacity)
end

#    3737fa99-4b4d-46f3-bdd4-106e589aa847
(model,var) = solve_chose_forwarders(weekStr,df_order_24_week,df_Average_loss_of_forwarders)

#    febc35df-f986-43e6-a98a-6a8107ef7dd9
termination_status(model)

#    1f51645c-fa66-45d0-a2a3-846554264525
termination_status(model)

#    14a0e20c-b79d-4d0f-aed9-e31753f2ddfa
md"""
# 把解转化为坐标
"""

#    02bd0f1c-4c41-4a3a-8928-d578cccec15a
# 解矩阵转化为数字
```

```
function mat2sym(mat::Matrix)

  function var2sym(sym_list::Vector,
           var::Vector)
    temp = missing
    for (i,sym) in zip(var,sym_list)
      if i == 1.0
        temp = sym
        break
      end
    end
    return temp
  end

  function mat2vec(mat::Matrix)
    x,_ = size(mat)
    return [mat[i,:] for i in 1:x]
  end

  return mat |> x->value.(x) |> mat2vec |> x->(x-> var2sym(forwarders_id,x)).(x)
end

#    9851ee35-f131-48e7-b87e-1c03d27b5263
# 解转化为坐标数字元组
function solve2location(供应商ID表_list::Vector,
            solve_list::Tuple,
            data_list::Vector)
  #辅助函数
  function delete(str::AbstractString,
          del::String)
    return replace(str,del=>"")
  end

  function sym2num(str::AbstractString)
    return str |> x->delete(x,"T") |> x->delete(x,"S") |> x->Parsers.parse(Int64,x)
  end

  function sym2num(str::Missing)
    return -1
  end

  function mat2location(供应商ID表::Vector,
              solve::Matrix,
              data::Vector)
    x = 供应商ID表 |> x->sym2num.(x)
    y = solve |> mat2sym |> x->sym2num.(x)
    return zip(x,y,data)
```

```
  end

  temp = []
  for (供应商ID表,solve,data) in zip(供应商ID表_list,solve_list,data_list)
    output = mat2location(供应商ID表,solve,data)
    for i in output
      temp = cat(temp,i,dims=1)
    end
  end
  return temp
end

#    a083a5da-0536-4bd9-9c5f-2ec66678d7f2
begin
  供应商ID表_list = [filter((:材料分类,i),df_order_24_week)[:,:Column1] for i in ["A","B","C"]]
  周订购数据_list = [filter((:材料分类,i),df_order_24_week)[:,week+2] for i in ["A","B","C"]]
  location_list = solve2location(供应商ID表_list, var, 周订购数据_list)
end

#    393930a9-26f6-4a25-9648-97b90d21eb51
md"""
# 写入表格，准备复制
"""

#    c9dee99e-9a55-4092-b2e4-e48dc1dcccba
#temp = DataFrame(fill!(Matrix{Float64}(undef, 402, 8),-114.514), :auto)

#    c7764ec3-6fc2-455a-8fdf-b1a319d4c0f6
for i in location_list
  (y,x,data) = i
  if x == -1
    continue
  end
  temp[y,x] =data
end

#    a9df1cd9-bfe1-461d-bac3-a91589c2163f
CSV.write("temp_location-$(weekStr).csv",temp)

#    7e218375-87e3-44a7-afa3-d5f4052375fc
#read("temp_location-$(weekStr).csv",String) |> x->replace(x,"-114.514"=>"") |> x->
    write("temp_location-$(weekStr).csv",x)

#    606088ae-d999-42ea-aa98-308412ac779d
function write_table(location_list::Vector,weekStr::String)
  temp = DataFrame(fill!(Matrix{Float64}(undef, 402, 8),-114.514), :auto)
  for i in location_list
```

```
    (y,x,data) = i
    if x == -1
      continue
    end
    temp[y,x] =data
  end
  path = joinpath("location","temp_location-$(weekStr).csv")
  CSV.write(path,temp)
  read(path,String) |> x->replace(x,"-114.514"=>"") |> x-> write(path,x)
end

#    2927eb71-9b9e-4026-8857-3331e715fc39
joinpath("location","temp_location-$(weekStr).csv")

#    618e342c-f035-4337-8705-91c10eaffbfd
write_table(location_list,weekStr)
```