


中国研究生创新实践系列大赛

“华为杯”第十七届中国研究生数学建模竞赛

学　　校	华东师范大学
参赛队号	**20102690221**
队员姓名	**1.** 郑智杰
	2. 周嘉铭
	3. 裴的

中国研究生创新实践系列大赛
“华为杯”第十七届中国研究生数学建模竞赛

题　　目　飞行器质心平衡供油策略优化

摘　　要:

飞行器的油箱在执行任务的时候起到了供油的作用，同时可以通过油箱调度策略对飞行器的质心位置进行调整，使得飞行器可以在理想的质心位置上执行任务，以此获得更好的飞行性能和安全性能。本文将制定飞行器质心平衡供油策略拆解为多个子问题，对各个子问题使用序列二次规划、改进的模拟退火算法等逐步进行求解，最终取得全局较优解。

针对问题一，将油箱分为六种状态，使用积分的方法直接求解出各个时刻每个油箱的质心位置，而后将每个油箱看作质点，最后对六个油箱的质心位置和飞行器 (不载油) 的质心位置进行计算求得飞行器在各个时刻的质心变化曲线如图5-4所示，其中图5-4 (a) 表示的改质心曲线在 3 维坐标下的呈现。

针对问题二，将问题分解为两个子问题：**子问题一：**假设每个时刻的总供油量确定，那么需要求解在该供油量下的所有油箱的最优供油策略。**子问题二：**需要为每一时刻设计合理的供油量，假设每个时刻的供油策略都是较优解，那么该总供油策略的目标是公式12。针对子问题一：采用一种贪心的策略，对于单时刻下的供油策略采用序列二次规划来搜索较优解。针对子问题二：使用改进的模拟退火算法求出所有油箱在各个时刻下的供油策略。通过对两个子问题的求解后可以解出本问题的较优解，最终求得飞行器瞬时质心与理想质心距离的最大值为 **0.065365**m, 四个主油箱的总供油量为 **6441.574**kg。

针对问题三，分解出的子问题一二与问题二相一致，除此之外还可以分解出**子问题三：**通过指定初始总油量，使用基于序列二次规划的初始油量分布算法，得出初始油量分布。针对子问题三的求解，在问题二的单时刻序列二次规划供油策略搜索算法基础上加以改进便可搜寻出较优解。最终求得飞行器瞬时质心与理想质心距离的最大值为 **0.029063**m, 四个主油箱的总供油量为 **6805.217**kg。

针对问题四，在问题二的基础上去除了角度 $\theta = 0$ 的限制条件后，对子问题二的约束条件进行优化，使得模型在某一时刻的贪心程度更强。最后求解出飞行器瞬时质心与理想质心距离的最大值为 **0.030596**m, 四个主油箱的总供油量为 **7632.230**kg。

关键字： 序列二次规划　贪婪算法　模拟退火　质心

目录

1. 问题重述 4
1.1 问题背景 4
1.2 需要解决的问题 4
2. 模型假设 5
3. 符号说明 5
4. 问题的分析 6
4.1 问题一分析 6
4.2 问题二分析 6
4.3 问题三分析 7
4.4 问题四分析 7
5. 问题一：模型的建立与求解 7
5.1 模型建立 7
5.1.1 任务执行过程中飞行器质心变化模型 7
5.2 模型求解 8
5.2.1 燃油在油箱内不同形状的分类及其判别 8
5.2.2 各个油箱以及飞行器的质心位置计算 10
5.2.3 求解结果及分析 11
5.3 模型评估 11
5.3.1 算法的有效性和复杂度 11
6. 问题二：模型的建立与求解 12
6.1 模型建立 12
6.1.1 平直飞的趋向理想质心的油箱供油策略模型 12
6.2 模型的求解 13
6.2.1 子问题的提取与设计 13
6.2.2 单时刻序列二次规划供油策略搜索算法 13
6.2.3 基于改进的模拟退火的供油策略规划搜索算法 16
6.2.4 求解的结果及分析 17
6.3 模型的评估 20
6.3.1 算法的有效性和复杂度 20
7. 问题三：模型的建立与求解 20
7.1 模型建立 20
7.1.1 可变初始油量平直飞的趋向理想质心的供油策略模型 20
7.2 模型的求解 21
7.2.1 子问题的提取与设计 21
7.2.2 基于序列二次规划的初始油量分布优化算法 21
7.2.3 求解的结果及分析 22
7.3 模型的评估 23
7.3.1 算法的有效性和复杂度 23

8. 问题四：模型的建立与求解 25
8.1 模型建立 25
8.1.1 俯仰角可变的趋向理想质心的供油策略模型 25
8.2 模型的求解 25
8.2.1 子问题的提取与设计 25
8.2.2 求解的结果及分析 26
8.3 模型的评估 26
8.3.1 算法的有效性和复杂度 26
9. 模型的评价 29
9.1 模型的优点 29
9.2 模型的缺点 29
10. 参考文献 30
附录 A 程序代码 31
1.1 第一问代码 31
1.2 第二问代码 33
1.3 第三问代码 44
1.4 第四问代码 51

1. 问题重述

1.1 问题背景

飞行器在执行飞行任务的时候会携带多个油箱，通过调度不同的油箱进行联合供油可以满足不同的飞行任务要求和发动机工作需求。同时，飞行器内各个油箱的油量分布以及供油策略会导致飞行器质心的变化，进而影响到对飞行器的控制。所以油箱的供油策略的作用不仅仅是用于给发动机供油，同时还可以利用供油策略控制飞行器的重心。

为了更好的控制飞行器以及提高飞行器的飞行性能，就需要制定出各油箱的供油策略，并通过油箱向发动机或其他油箱供油的速度曲线来描述对应的供油策略。

根据上述问题，飞行器的质心平衡供油策略的优化就有下列的限制条件：

1. 每个油箱均为长方体且固定在飞行器内部。
2. 第 i 个油箱有对应的供油上限 $U_i(U_i > 0)$，并且每个油箱一次供油的持续时间不少于 60 秒。
3. 至多只能 2 个油箱同时向发动机供油，且至多只能有 3 个油箱同时供油。
4. 在执行飞行任务时，所有油箱的燃油总量至少需要满足发动机对于耗油量的需求，当某一时刻油箱的供油量大于计划的耗油量时，多余的燃油将被排出飞行器。
5. 飞行器的飞行姿态的变化仅考虑平直飞与俯仰的情况。
6. 惯性坐标系 $O-XYZ$：表示飞行器在地面上时，以不载油的飞行器的质心为原点 O，同时以飞行器的正前方为正向，重力方向的反方向为 Z 轴的正向，使用右手法则确定 Y 轴。
7. 飞行器坐标系 $O(t)-X(t)Y(t)Z(t)$：在 t 时刻，以不载油的飞行器的质心位置 $\overrightarrow{c_0}$ 为原点 $O(t)$，飞行器纵向中心轴为 $X(t)$ 轴，以飞行器前方为正向，$Y(t)$ 轴垂直于 $X(t)$ 轴所在的飞行器纵剖面，且 $O(t)-X(t)Y(t)$ 组成右手坐标系，通过右手法则确定 $Z(t)$ 轴。
8. 飞行器 t 时刻俯仰角 $\theta(t)$: 在飞行器坐标系中的 $X(t)$ 轴与惯性坐标系中的 $O-XY$ 水平面的夹角，$X(t)$ 轴正方向在重力方向分量在重力方向分量与重力方向相反时 $\theta(t)$ 为正。

1.2 需要解决的问题

所用的数据集的介绍：附件 1 为该飞行器的相关参数，附件 2-附件 5 给出了该类飞行器在执行某任务过程中飞行和控制的相关数据。因此需要解决的问题如下：

1. 在限制条件 1~8 的约束下，需要设计算法结合所给任务飞行器的油箱的供油速度及飞行器在飞行过程中的俯仰角变化数据, 建立数学模型，并依据此模型计算出飞行器的质心变化。因所给数据对应每秒记录一组数据，故需要求出每一时刻的质心变化及其曲线并对该算法进行有效性和复杂度的分析。
2. 问题二需要在飞行器始终保持平飞的任务中，制定满足附件 3 中条件 (1)~(6) 的 6 个油箱的供油策略，使得飞行器在给定的计划耗油速度下飞行器每一时刻的质心位置 $\overrightarrow{c_1}(t)$ 与理想质心位置 $\overrightarrow{c_2}(t)$ 的欧式距离的最大值达到最小。
3. 问题三则在问题二的基础上要求在任务结束的时候六个油箱剩余燃油总量至少有 $1m^3$, 同样也需要使得飞行器每一时刻的质心位置 $\overrightarrow{c_1}(t)$ 与理想质心位置 $\overrightarrow{c_2}(t)$ 的欧式距离的最大值达到最小。
4. 问题四是在实际任务规划过程中，考虑到飞行器还会有俯仰角的变化。所以在第二问的基础上加上俯仰角的变化，最终建模得出各个油箱的供油曲线以

及飞行器瞬时质心与飞行器 (不载油) 质心 $\overrightarrow{c_1}$ 的最大距离偏差和主油箱的总供油量。

2. 模型假设

1. 假设油箱内燃油不受惯性影响。
2. 假设燃油在管道中流动的时候不影响质心
3. 假设燃油密度均匀。
4. 油箱均为长方体且固定在飞行器内部。
5. 每个油箱一次供油的持续时间不少于 60 秒。
6. 主油箱 2、3、4、5 可直接向发动机供油，油箱 1 和油箱 6 作为备份油箱分别为油箱 2 和油箱 5 供油，不能直接向发动机供油。
7. 飞行器姿态的改变仅考虑平直飞与俯仰情况。

3. 符号说明

符号	意义
θ	飞行器内油箱的俯仰角（°）
$\theta(t)$	飞行器 t 时刻内油箱的俯仰角（°）
ρ_o	燃油的密度 (kg/m^3)
i	第 i 个油箱 $i = 1, 2, \cdots, 6$
t	飞行器执行飞行任务的某一时刻
a_i	第 i 个油箱内部的长 (m)
b_i	第 i 个油箱内部的宽 (m)
c_i	第 i 个油箱内部的高 (m)
$\overrightarrow{c_0}$	飞行器（不载油）质心
$\overrightarrow{o_i}$	第 i 个油箱的质心位置
$\overrightarrow{o_i}(t)$	飞行器 t 时刻第 i 个油箱的质心位置
$\overrightarrow{c_1}(t)$	飞行器 t 时刻的质心位置
$\overrightarrow{c_2}(t)$	飞行器 t 时刻的理想质心位置
d	飞行器理想质心位置与飞行器质心位置距离
$d(t)$	飞行器 t 时刻的理想质心位置与飞行器质心位置距离
$\overrightarrow{P_i}$	第 i 个油箱中心位置
M	飞行器（不载油）总重量 (kg)
$m_i(t)$	飞行器 t 时刻第 i 个油箱的燃油重量 (kg)
U_i	第 i 个油箱的供油速度上限 ($U_i > 0$)

$u(t)$	飞行器 t 时刻计划耗油速度 (kg/s)
$u_i(t)$	飞行器 t 时刻第 i 个油箱的输油量
u_a	飞行器初始指定的所有油箱总油量
s_i	第 i 个油箱是否供油
$s_i(t)$	t 时刻时第 i 个油箱是否供油
$s_i'(t)$	t 时刻时第 i 个油箱是否必须供油
cnt_i	t 时刻时第 i 个油箱已经持续供油的多少时间
$u_{left\_i}(t)$	t 时刻时第 i 个油箱剩余燃油的质量 (kg)
ε	所有时刻总供油量策略
S	油箱内燃油沿油箱坐标轴 Y' 方向的切面面积 (m^2)

4. 问题的分析

飞行器油箱内的燃油对飞行器质心影响很大，在整个飞行过程中燃油一直在消耗，故飞行器的质心也时刻在变化着，同时在飞行过程中飞行器其他的参数也是处于变化的状态，这就使得计算油箱的质心变得十分的复杂。飞行器在飞行的过程中有着不同的理想质心，当飞行器处在理想质心的时候可以使得飞行器的飞行性能、飞行品质、控制性能、安全性等达到一个良好的状态。

对于上述背景下直接求解飞行器的最优供油策略解空间过大，变量太多，容易导致时间复杂度、空间复杂度达到无法计算的程度，传统的优化算法如蚁群算法、模拟退火算法、遗传算法等都无法直接对问题进行优化，为了解决这些问题本文提出了局部搜索，全局迭代逐步优化的方法求解最优供油策略。

4.1 问题一分析

前提：题目中给出了某次任务中飞行器的 6 个油箱的供油速度及飞行器在任务期间的俯仰角的变化，同时根据飞行器参数可知油箱的中心位置、油箱的尺寸、各个油箱的初始油量以及燃料的密度。

条件：根据上述问题的背景以及限制条件 1~8 的要求，对附件 2 中的俯仰角变化求得飞行器的质心位置的变化。

目标：基于上述前提和假设求解出该飞行器在此次飞行任务中的每一时刻（每秒）在飞行器坐标系下的质心位置并做出质心变化曲线。

4.2 问题二分析

前提：同问题一给出了某次飞行器计划耗油速度数据以及飞行器的属性，但同时还给出了飞行坐标系下各个时刻的理想质心位置数据，且飞行器始终保持平飞 (俯仰角为 0) 的任务规划过程中。

条件：根据问题的背景，在限制条件 1~7 和所有假设的设定下对附件 3 所给的数据进行飞行器的 6 个油箱的供油策略规划。

目标：基于上述前提和假设通过调整各个油箱的供油策略使得飞行器的每个时刻质心位置 $\vec{c_1}(t)$ 与理想质心位置 $\vec{c_2}(t)$ 的欧式距离的最大值达到最小。需

要满足的目标为：

$$\min \max_t \|\overrightarrow{c_1}(t) - \overrightarrow{c_2}(t)\|_2$$

4.3 问题三分析

前提：同问题二给出了某次飞行器计划耗油速度数据，飞行坐标系下各个时刻的理想质心位置数据以及飞行器的属性。且飞行器同第二问始终保持平飞（俯仰角为 0）的任务规划过程中。但未给出飞行器各个油箱的初始油量。

条件：根据问题的背景，在限制条件 1 7，最终所有油箱剩余的油量的总和需大于 1 立方米的要求和所有假设的设定下对附件 4 所给出的数据，进行飞行器 6 个油箱的供油策略规划和各个油箱初始油量的设计。

目标：基于上述前提和假设通过调整各个油箱的供油策略使得飞行器的每个时刻质心位置与理想质心位置的欧式距离的最大值达到最小。需要满足的目标为：

$$\sum_{i=1}^{6} u_i(t_{end}) > \rho_o \tag{1}$$

$$\min \max_t \|\overrightarrow{c_1}(t) - \overrightarrow{c_2}(t)\|_2 \tag{2}$$

4.4 问题四分析

前提：同问题二一致，但飞行器可以有任意的俯仰角。

条件：根据问题的背景，在限制条件 1∼7 和所有假设的设定下同时为任意的俯仰角对所给的数据进行飞行器的 6 个油箱的供油策略规划。

目标：基于上述前提和假设通过调整各个油箱的供油策略使得飞行器的每个时刻质心位置 $\overrightarrow{c_1}(t)$ 与理想质心位置 $\overrightarrow{c_2}(t)$ 的欧式距离的最大值达到最小。需要满足的目标为：

$$\min \max_t \|\overrightarrow{c_1}(t) - \overrightarrow{c_2}(t)\|_2$$

5. 问题一：模型的建立与求解

5.1 模型建立

5.1.1 任务执行过程中飞行器质心变化模型

由假设 7 可知，飞行器姿态的改变仅考虑平直飞与俯仰情况，所以对于飞行器在时刻 t 的质心位置只与时刻 t 的各个油箱中剩余的燃油质量和俯仰角 $\theta(t)$ 有关，在研究单个油箱的时候，无论油箱内燃油质量如何变化，其 y 坐标都是保持不变的，所以在考虑单个油箱的时候只需计算 x、y 方向上的变化即可。

问题一求解飞行器各个时刻质心变化位置的流程图如图5-1所示。

对于单个油箱使用质心公式 (3) 可以计算出质心位置 $\overrightarrow{c_i}$。[6]

$$\begin{aligned} \overline{X} &= \frac{1}{S}\iint_D x dx dz \\ \overline{Z} &= \frac{1}{S}\iint_D z dx dz \end{aligned} \tag{3}$$

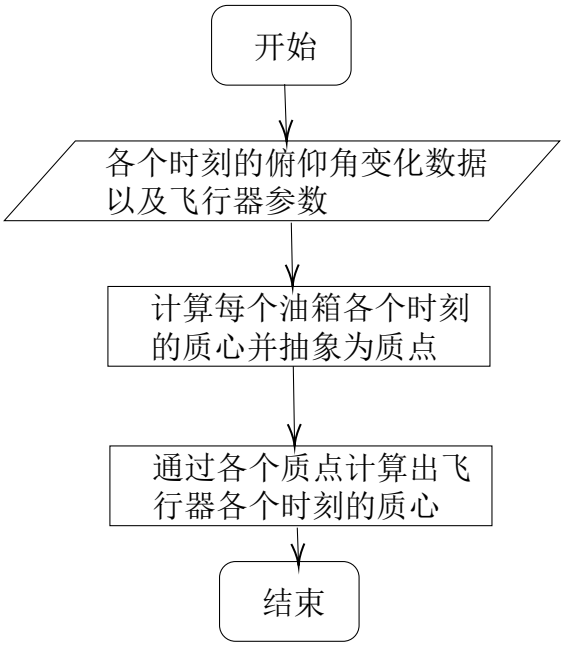


图 **5-1**　求解飞行器在执行任务的各个时刻质心位置流程图

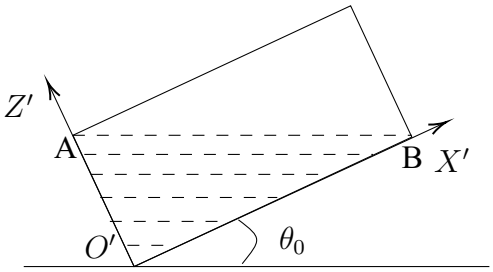


图 **5-2**　油箱倾斜临界状态

对于各个油箱里的燃料以及飞行器抽象成质心位置为 $\overrightarrow{c_i}$，质量为 m_i 的质点，最终根据公式 (4) 求得飞行器的质心位置。

$$\overrightarrow{C} = \frac{\sum_{i=1}^{n} m_i \overrightarrow{c_i}}{\sum_{i=1}^{n} m_i} \tag{4}$$

5.2 模型求解

5.2.1 燃油在油箱内不同形状的分类及其判别

为了求解整个飞行器的质心，需要先分析每个油箱的质心位置。因为飞行器 y 轴相对保持不变，所以从 y 轴对油箱进行切面如图5-2所示。

在研究油箱的时候重新建立油箱坐标系 $O'-X'Y'Z'$：以油箱水平高度最低的点为原点 O', 和 O' 相邻的两条边中靠近重力反方向的边作为 Z' 轴，另外一边为 X' 轴。

X' 轴与水平面的夹角记为 θ 角，其中图5-2表示临界状态，即油箱靠近 X'

的顶点 B 与靠近 Z' 的顶点 A 处在同一水平高度。记该状态下的夹角为 θ_0。

所以能将油箱的状态分为图5-3中的六种不同的状态 [1]，具体描述如表2所述。

表 **2**　油箱六种状态的描述

状态名称	θ 大小	液体高度
第一种状态	$\theta > \theta_0$	燃油上方水平面高度低于 A 点
第二种状态	$\theta > \theta_0$	燃油上方水平面高度高于 A 点且低于 B
第三种状态	$\theta > \theta_0$	燃油上方水平面高度高于 B 点
第四种状态	$\theta < \theta_0$	燃油上方水平面高度低于 B 点
第五种状态	$\theta < \theta_0$	燃油上方水平面高度高于 B 点且低于 A
第六种状态	$\theta < \theta_0$	燃油上方水平面高度高于 A 点

设燃油的切面面积为 S，油箱的长宽高分别为 a, b, c，所以对于各种状态的燃油切面的区域可以表示为：

第一种状态：

$$\begin{aligned} &0 < x' < \sqrt{2S\cot\theta} \\ &0 < z' < \left(\sqrt{2S\cot\theta} - x'\right)\tan\theta \end{aligned} \tag{5}$$

第二种状态：

$$0 < x' < \frac{1}{2}c\cot\theta + \frac{S}{c}$$

$$\begin{cases} 0 < z' < c & \text{当} x' \le \dfrac{S}{c} - \dfrac{c\cot\theta}{2} \\ 0 < z' < \frac{c}{2} + \frac{S}{c}\tan\theta - x'\tan\theta & \text{当} x > \dfrac{S}{c} - \dfrac{c\cot\theta}{2} \end{cases} \tag{6}$$

第三种状态：

$$0 < x' < a$$

$$\begin{cases} 0 < z' < c & \text{当} x' \le a - \sqrt{2\cot\theta\,(ac - S)} \\ 0 < z' < \left[a - \sqrt{2\cot\theta\,(ac - S)} - x'\right] & \text{当} x' > a - \sqrt{2\cot\theta\,(ac - S)} \end{cases} \tag{7}$$

第四种状态：

$$\begin{aligned} &0 < x' < \sqrt{2S\cot\theta} \\ &0 < z' < \left(\sqrt{2S\cot\theta} - x'\right)\tan\theta \end{aligned} \tag{8}$$

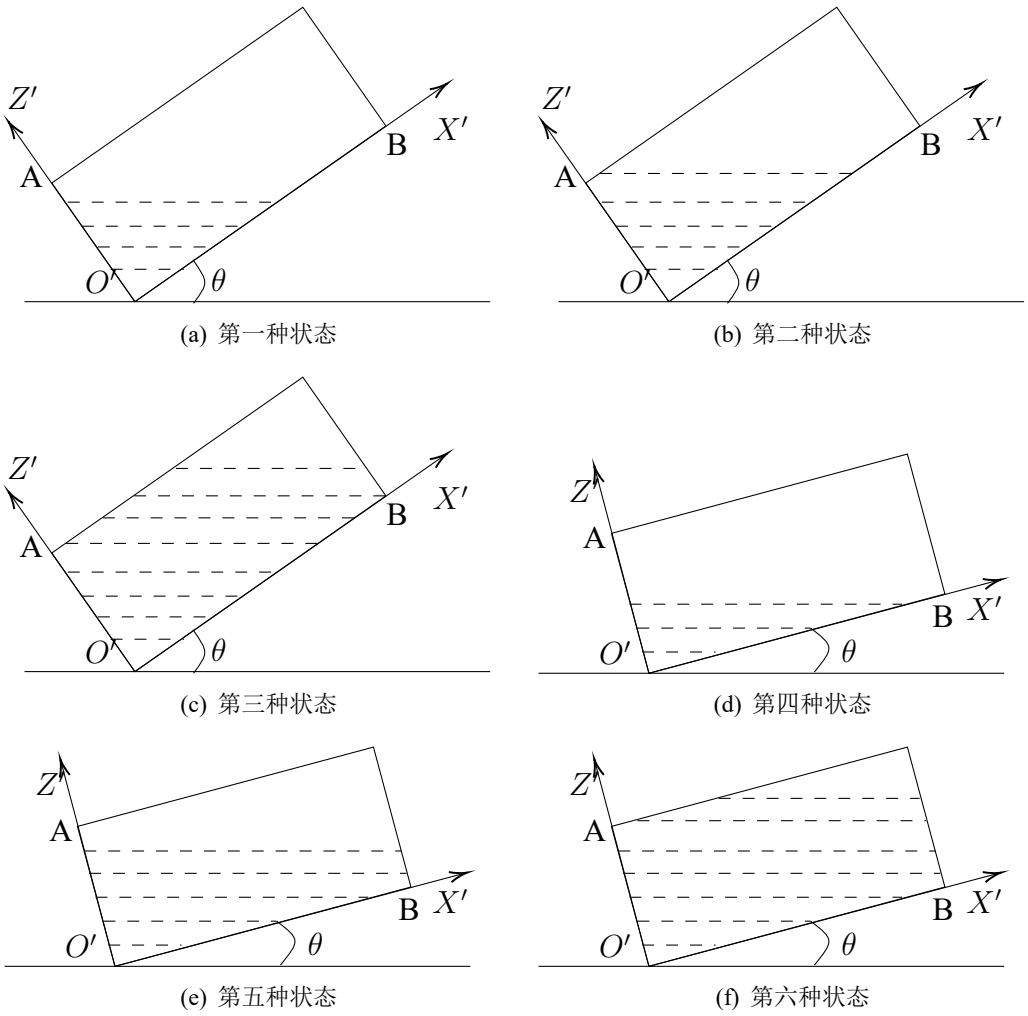
第五种状态：

$$\begin{aligned} &0 < x' < a \\ &0 < z' < \frac{1}{2}\text{atan}\theta - x'\tan\theta + \frac{S}{a} \end{aligned} \tag{9}$$

第六种状态：

$$0 < x' < a$$

$$\begin{cases} 0 < z' < c & \text{当} x' \le a - \sqrt{2\cot\theta\,(ac - S)} \\ 0 < z' < \left[a - \sqrt{2\cot\theta\,(ac - S)} - x'\right] & \text{当} x' > a - \sqrt{2\cot\theta\,(ac - S)} \end{cases} \tag{10}$$



(a) 第一种状态　(b) 第二种状态

(c) 第三种状态　(d) 第四种状态

(e) 第五种状态　(f) 第六种状态

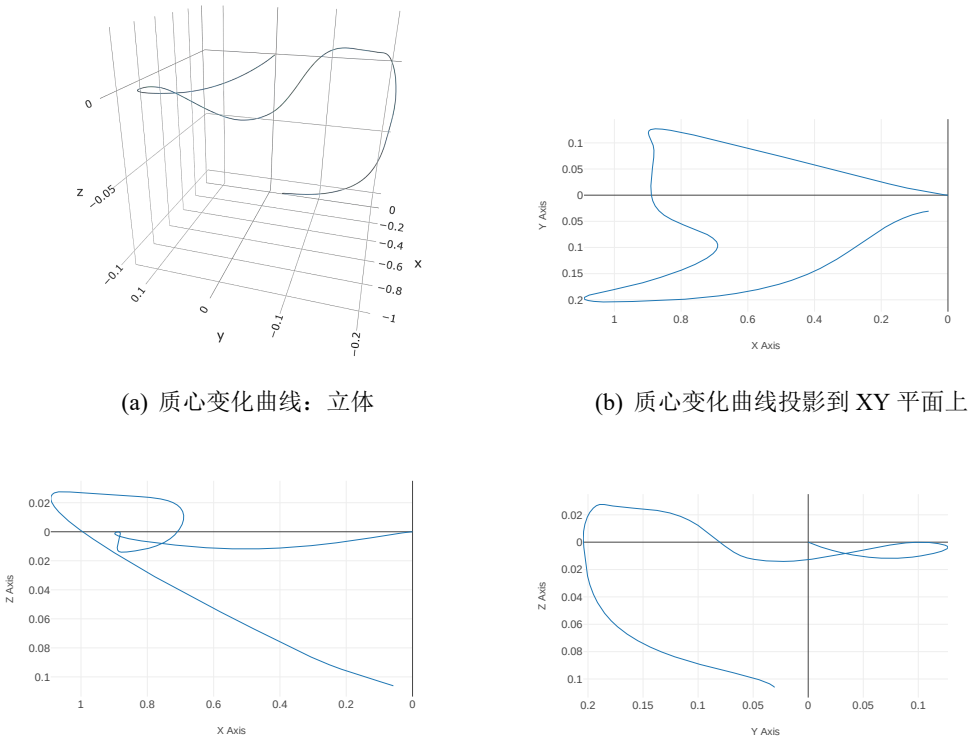
图 **5-3**　油箱倾斜时不同的六种状态

5.2.2 各个油箱以及飞行器的质心位置计算

利用质心公式 (3) 再结合上述油箱六种状态中燃油切面范围：(5)~(10)，同时将油箱坐标系转换为飞行器坐标系下，最后计算出各个油箱的质心 $\overrightarrow{o_i}(t)$，然后将所有油箱的质心抽象为质量为 $m_i(t)$ 的质点，将各个油箱的质心坐标和质量以及飞行器 (空油箱) 的质心位置和质量带入多个质点质心公式公式 (4) 可以得出飞行器某一时刻的质心位置 $\overrightarrow{c_1}(t)$

$$\overrightarrow{c_1}(t)=\frac{\sum_{i=1}^{6} m_i(t)\,\overrightarrow{o_i}(t)+M\overrightarrow{c_0}}{\sum_{i=1}^{6} m_i(t)+M} \tag{11}$$

5.2.3 求解结果及分析



(a) 质心变化曲线：立体

(b) 质心变化曲线投影到 XY 平面上

(c) 质心变化曲线投影到 XZ 平面上

(d) 质心变化曲线投影到 YZ 平面上

图 **5-4**　飞行器执行任务过程中的质心变化曲线

本题使用积分的方法直接求解出各个时刻每个油箱的质心位置，而后将每个油箱看作质点，最后对六个油箱的质心位置和飞行器 (不载油) 的质心位置进行计算求得飞行器在各个时刻的质心变化曲线如图 5-4所示，其中图5-4 (a) 表示的改质心曲线在 3 维坐标下的呈现，图5-4 (b) 为质心变化曲线投影到 XY 平面上, 图5-4 (c) 为质心变化曲线投影到 XZ 平面上，图5-4 (d) 为质心变化曲线投影到 YZ 平面上。

最终计算的结果放入到附件 6 结果表“第一问结果”中，第一问相关代码在附录 A 1 .1中。

5.3 模型评估

5.3.1 算法的有效性和复杂度

本问主要使用的多重积分求解出油箱在各种状态下的质心位置，然后将所有油箱以及飞行器 (不载油) 的质心共同计算求解出飞行器的质心位置, 所以本问的时间复杂度为 $O(n)$, 然而计算出该次飞行任务的质心变化需要使用计算多重

积分的代码，程序花费的时间主要是集中在多重积分计算上，运行完成所有的时刻质心位置总共花费时间为 70s。

在某次飞行任务中随着飞行器在飞行过程中的俯仰角 θ 的变化，油箱中的燃油水平面高度与油箱会处在不同状态，其中会有一个分界状态如图5-2所示。充分考虑了不同的情况下的油箱质心的计算后，最终得出一个合理有效的质心变化曲线。

6. 问题二：模型的建立与求解

6.1 模型建立

6.1.1 平直飞的趋向理想质心的油箱供油策略模型

优化目标：飞行器给定计划耗油速度数据与飞行器在飞行器坐标下的理想重新位置数据。在平直飞条件下制定使得飞行器每一时刻的质心位置 $\overrightarrow{c_1}(t)$ 与理想质心位置 $\overrightarrow{c_2}(t)$ 的欧式距离 $d(t)$ 的最大值达到最小。

目标函数：飞行器平直飞趋向理想质心的供油策略模型，即在飞行器保持平直飞的条件下通过调整油箱的供油使得飞行器的质心逼近理想质心，需要优化各个时刻每个油箱供油量。目标函数可以定义为：

$$\min \max_{t} \|\overrightarrow{c_1}(t) - \overrightarrow{c_2}(t)\|_2 \tag{12}$$

其中的 $\overrightarrow{c_1}$ 可以通过问题一的公式 (11) 求得。

决策变量：定义决策变量 s_i 表示第 i 个油箱是否供油：

$$s_i = \begin{cases} 0 & \text{该油箱不供油} \\ 1 & \text{该油箱可以供油} \end{cases} \tag{13}$$

同样对于飞行器执行飞行任务的中的时刻 t 的油箱是否供油的决策变量为 $s_i(t)$，在该时刻每个油箱的供油量可以表示为 $u_i(t)$.

约束条件： (1) 飞行器油箱 i 有个输送油量上限 U_i，在飞行器执行飞行任务的某一时刻 t 中

$$u_i(t) \leq U_i \quad (U_i > 0) \tag{14}$$

(2) 主油箱向飞行器供油的总量应该至少满足发动机对耗油的需要：

$$u_2(t) + u_3(t) + u_4(t) + u_5(t) \geq u(t) \tag{15}$$

(3) 飞行器最多有两个主油箱向发动机供油:

$$s_2 + s_3 + s_4 + s_5 \leq 2 \tag{16}$$

至多 3 个油箱可同时供油：

$$\sum_{i-1}^{6} s_i \leq 3 \tag{17}$$

(4) 飞行器在该任务中始终保持平直飞，故 $\theta = 0$。(5) 飞行器在某一时刻 t 中，油箱 i 累计输油时间记为 $cnt_i(t)$, 该时刻某油箱的是否必须供油用 $s_i'(t)$ 表示：

$$s_i'(t) = \begin{cases} 0 & cnt_i(t) \geq 60 \\ 1 & cnt_i(t) < 60 \end{cases} \tag{18}$$

(6) 在每个时刻油箱有个供油区间 $u_i(t) \in [0, min(U_i, u_{left\_i})]$，其中 u_{left_i} 表示油箱 i 剩余燃油的质量。

综上所述，所建立的的平直飞的趋向理想质心的供油策略模型可以表述为：

$$\begin{aligned} &\min \max_t \|\overrightarrow{c_1}(t) - \overrightarrow{c_2}(t)\|_2 \\ &s.t \begin{cases} u_i(t) \le U_i \quad (U_i > 0) \\ u_2(t) + u_3(t) + u_4(t) + u_5(t) \ge u(t) \\ u_i(t) \in [0, min(U_i, u_{left\_i})] \\ s_2 + s_3 + s_4 + s_5 \le 2 \\ \sum_{i-1}^{6} s_i \le 3 \\ s_i'(t) = \begin{cases} 0 & cnt_i(t) \ge 60 \\ 1 & cnt_i(t) < 60 \end{cases} \\ \theta = 0 \end{cases} \end{aligned} \tag{19}$$

6.2 模型的求解

6.2.1 子问题的提取与设计

对于本问题来说，直接对解空间求解，范围过大，效果必然不佳，需要将原始问题拆分为多个子问题，在依次对不同的子问题进去求解。

子问题一：假设每个时刻的总供油量确定，那么需要求解在该供油量下的所有油箱的最优供油策略。

子问题二：需要为每一时刻设计合理的供油量，假设每个时刻的供油策略都是较优解，那么该总供油策略的目标是公式12

针对子问题一：采用一种贪心的策略，对于单时刻下的供油策略采用序列二次规划来搜索较优解。贪心算法的步骤如表3

针对子问题二：使用改进的模拟退火算法求出所有油箱在各个时刻下的供油策略。

6.2.2 单时刻序列二次规划供油策略搜索算法

在本问中需要搜索目标包括飞行器每一时刻需要开启的油箱 $s_i(t)$ 以及该时刻各个油箱的输油量 $u_i(t)$，如果对于解空间直接搜索，范围过大，可能会导致无法得出理想的解。

为了对该模型进行求解，需要对模型进行和合理的简化：

- 将 60 秒设为一次供油的单位。所以可以将总时间 7200 秒切分为 120 个时间片段。为保证每个油箱至少 60s 处于输出状态，使油箱的供油量可以是一个极小值。
- 因为油箱 1 和油箱 6 的速度分别小于油箱 2 和油箱 5，设定一种油箱 1 向发动机供油的情况，油箱 1 向油箱 2 供油，同时油箱 2 向发动机供油，两个油箱的供油速度保持相同，因为油箱 1 的速度上限小于油箱 2，所以不会出现油箱 1 达到了上限速度，但油箱 2 不能以该速度向发动机供油的情况。
- 为了避免出现 4 个油箱同时供油的情况，去除了油箱 1 和油箱 6 同时供油的情况。同时因为油箱 1 和油箱 6 是通过油箱 2 和油箱 5 向发动机供油，所以去除了油箱 1 和油箱 2，油箱 5 和油箱 6 同时向发动机供油的情况。最后所有的供油组合如下面 12 种情况。(1, 3),(1, 4)，(1, 5), (2, 3), (2, 4), (2, 5), (2, 6), (3, 4),(3, 5),(3, 6), (4, 5), (4, 6)。

表 **3**　油箱六种状态的描述

基于序列二次规划的贪心算法步骤
Step1: 从时间片 1 开始进行枚举
Step2: 枚举设计好的供油油箱集合内的油箱组合
Step3: 进入时间片中的第一个时刻
Step4: 使用 SQP 找到当前时刻使得与理想质心欧式距离最短的供油策略
Step5: 进入下一时刻，若已遍历完当前时间片则跳至下一步，否则返回 step4
Step6: 更新当前时间片的最优油箱组合
Step7: 继续枚举下一油箱组合，若已遍历完则跳至下一步，否则返回 step3
Step8: 更新当前记录下最大的与理想质心的欧式距离
Step9: 进入下一时间片，若遍历完则算法结束，保存结果，返回记录下的最大欧式距离，否则返回 step2

将时间分为片段后，对于每一个片段，选择一种供油组合来进行供油。结合贪心的策略，从第一个时间片开始，枚举每一中供油组合，计算每种供油组合可以达到的欧氏距离的最小值。最后选择一个最小的欧氏距离的组合。同时更新最大的欧氏距离，更新总油量，并且记录下这一时间片的供油策略。对每一时刻进行上面所述的计算，直到最后一个时间片段，即可获得最大的欧式距离。所有时刻最小欧式距离的最大欧式距离，其流程图如图 6-1所示。

定义一次任务中的在时间 t 上的目标供油量为 ε 即一个供油策略。对于每个油箱的时间片来说可以输入序列二次规划供油策略搜索算法，记为该算法为

$$d_{\min} = \mathcal{L}(\varepsilon) \tag{20}$$

其中 $d_{\min}$ 为改时间序列下的飞行器的质心到理想质心的最大值的最小值，$\mathcal{L}(\varepsilon)$ 表示供油策略搜索算法，ε 为油箱总的各个时刻供油需求策略。对于一个完整的供油策略 ε 来说枚举每个时间片为了计算每一个时间片段的最小欧式距离，建立了一个目标函数:

$$min f_{a,b}(u_a, u_b) \tag{21}$$

函数 $f_{a,b}(u_a, u_b)$ 表示在使用两个油箱 a, b 后，油箱消耗了 u_a 和 u_b 的油量后的飞行器质心位置与理想质心的欧式距离。

其约束条件为：

$$\begin{aligned} &u_a + u_b = u_i(t) \\ &u_a < U_a \\ &u_b < U_b \end{aligned} \tag{22}$$

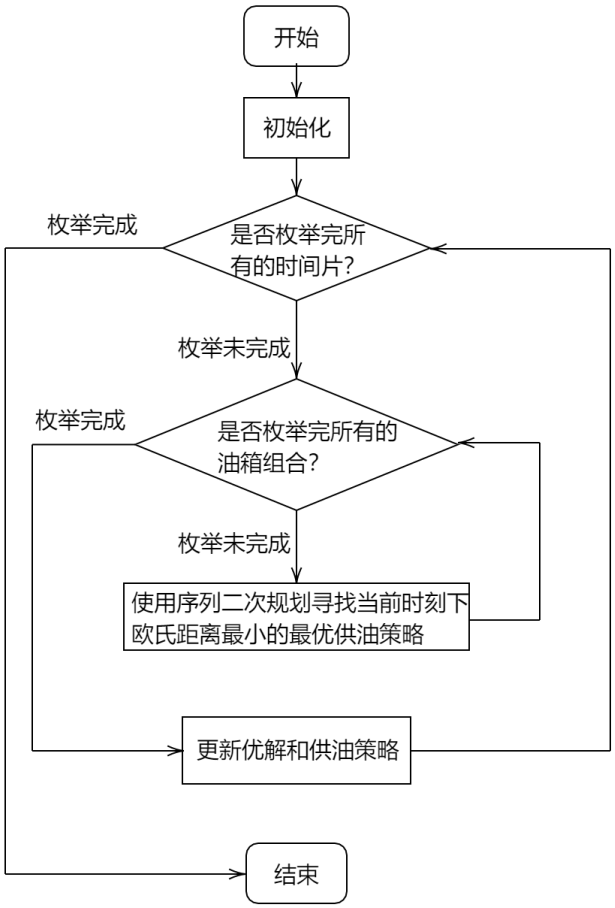


图 **6-1**　基于时间片的序列二次规划供油策略搜索算法流程图

综上，该模型表示为：

$$\begin{aligned} & minf_{a,b}(u_a, u_b) \\ & s.t \begin{cases} u_a + u_b = u_i(t) \\ u_a < U_a \\ u_b < U_b \end{cases} \end{aligned} \tag{23}$$

对于这个非线性规划的求解，使用了 SLSQP 的方法对求解函数的最小值进行优化 [2]。

SLSQP 是一种在步长算法中采用了 Han‐Powell 拟牛顿法，对 B 矩阵进行了 BFGS 更新，引入了 L1 检验函数等优化的一种序列最小二乘规划算法。通过 Lawson 和 Hanson 的 NNLS 非线性最小二乘法求解器的一个解决最小二乘规划算法的程序。

SQP（Sequential quadratic programming）是序列二次规划算法，是求解非线性优化问题 (NLP) 的经典算法。该方法的主要的缺点是合并了几个导数，这些导数可能需要在迭代到解之前进行解析工作，因此对于具有许多变量或约束的大型问题，SQP 变得相当麻烦。但是对于中小规划约束最优化问题的解决特别有效。

SQP 方法通过求解一系列优化子问题来获得最后的最优解，每个优化子问题在约束线性化的前提下优化目标的二次型模型。如果问题是无约束的，则该方法简化为寻找目标梯度为零的点的牛顿方法。如果问题只有等式约束，则该方法等价于将牛顿方法应用于问题的一阶最优性条件。

目标函数的临界点也将是拉格朗日函数的临界点，反之亦然。所以所有的约束要么等于 0. 要么无效。因此该算法通过牛顿迭代法来寻找拉格朗日函数的临界点。由于拉格朗日乘数是附加变量，因此迭代形成一个系统。通过使用二次算法求解的二次极小化子问题。[3][4]

对该系统中的不同方程进行分解，并将二阶项减半以匹配泰勒级数概念，即可得到一个极小子问题。这个问题是二次型的，因此必须用非线性方法来解决，这再次将解决非线性问题的需要引入到算法中，但是这个具有一个变量的可预测子问题比父问题更容易解决。

6.2.3 基于改进的模拟退火的供油策略规划搜索算法

通过序列二次规划供油策略搜索算法可以求得某个时刻下使得飞行器的质心和理想质心的最小距离的最优油箱供油策略，但是对于某次任务来说是由许多时刻组成，其目标供油量需要单独的搜索。

所以可以定义模拟退火的目标函数 [5]：

$$\min \mathcal{L}(\varepsilon) \tag{24}$$

约束条件受公式 (19) 平直飞的趋向理想质心的供油策略模型模型的约束条件制约。其中的 $\mathcal{L}(\varepsilon)$ 为公式 (20)。

模拟退火的流程如图6-2所示，未经改进的模拟退火对于本问求解的效果很一般，为了使得算法找到更好的解，需要对模拟退火进行优化。

模拟退火数据生成过程改进：对于单个时刻的供油来说有个供油需求 $\varepsilon(t)$ 的变化范围, 即。

$$u(t) < \varepsilon(t) < \min(U_i, u_a + u(t)) \tag{25}$$

其中 u_a 表示：单个时刻合理可接受的总发动机供油量波动范围，u_t 表示当前时刻的计划供油量

$$u_a = \frac{\sum_{i=1}^{6} m_i(1) - \sum_{n=1}^{t} m_i(n)}{t_a} \tag{26}$$

所有只要保证 $\varepsilon(t)$ 处在该区间范围内，该策略即为合理解。

对于每个时刻总的供油量 $\varepsilon(t)$ 根据该供油量所处的时间，随机生成不同的供油量：

$$\varepsilon(t) = u(t) + u_a\omega(1 - \frac{t}{t_a}) \tag{27}$$

其中 t_a 表示总时刻数，ω 表示 $0 \sim 1$ 的随机数。

对每个时刻都生成一个策略，最后获得到所有油箱总的供油需求策略 ε。

模拟退火随机扰动过程改进：针对该问题，通过多次运行模拟退火发现在飞机不同时刻供油的量对飞行器质心与理想质心的最小欧氏距离的影响是不同的，所以需要针对不同时刻的供油策略使用不同的概率进行扰动。优化后的供油策略概率 p 为：

$$p_2 = \frac{t}{t_a}(1 - \frac{T}{T_{\max}}) \tag{28}$$

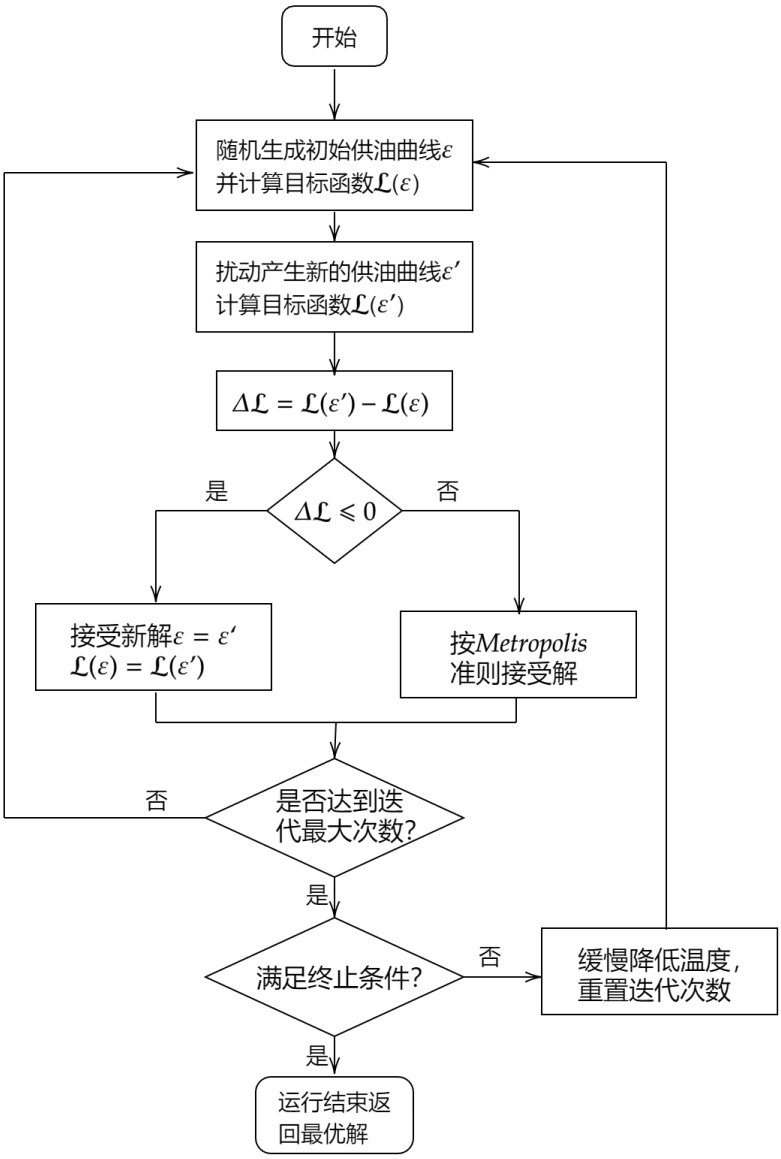


图 **6-2**　模拟退火流程图

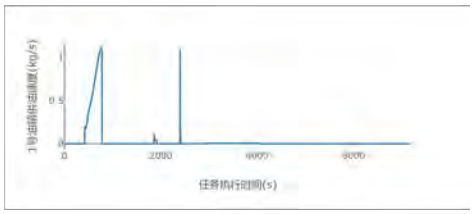
t 表示当前时刻，t_a 表示总时刻数，T 表示模拟退火的当前温度,$T_{\max}$ 则是模拟退火的初始温度，概率 p 表示当前时刻的策略可以扰动的概率。

在对模拟退火进行改进之后便可求解搜索飞行器的供油策略，达到收敛，完成对较优解的搜索。

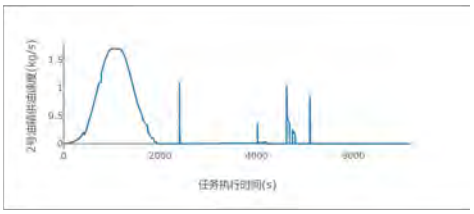
6.2.4 求解的结果及分析

通过对两个子问题的求解后可以解出本问题的较优解，最终求得飞行器瞬时质心与理想质心距离的最大值为 **0.06536568**m, 四个主油箱的总供油量为 **6441.574**kg，6 个油箱各自的供油速度曲线如图6-3所示，4 个主油箱的总供油速度曲线如图6-4所示。

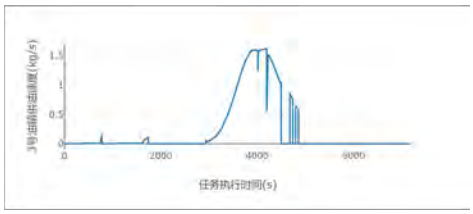
从结果中可以看出各个油箱耗油分布合理，飞行器瞬时质心与理想质心距离的最大值约为 6cm，总的质心偏移非常小，而在质心偏差小的基础上飞行器执



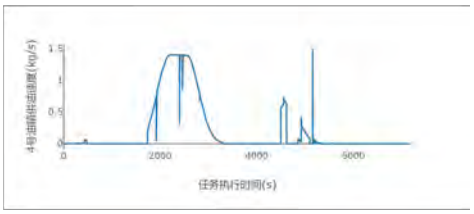
(a) 1 号油箱供油速度曲线



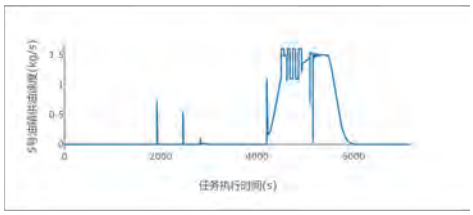
(b) 2 号油箱供油速度曲线



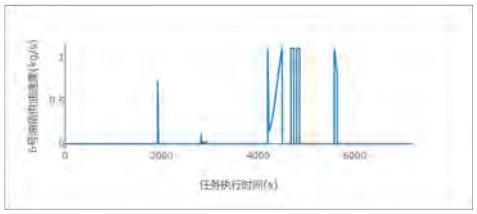
(c) 3 号油箱供油速度曲线



(d) 4 号油箱供油速度曲线



(e) 5 号油箱供油速度曲线



(f) 6 号油箱供油速度曲线

图 **6-3**　第二问六个油箱各自的供油速度曲线

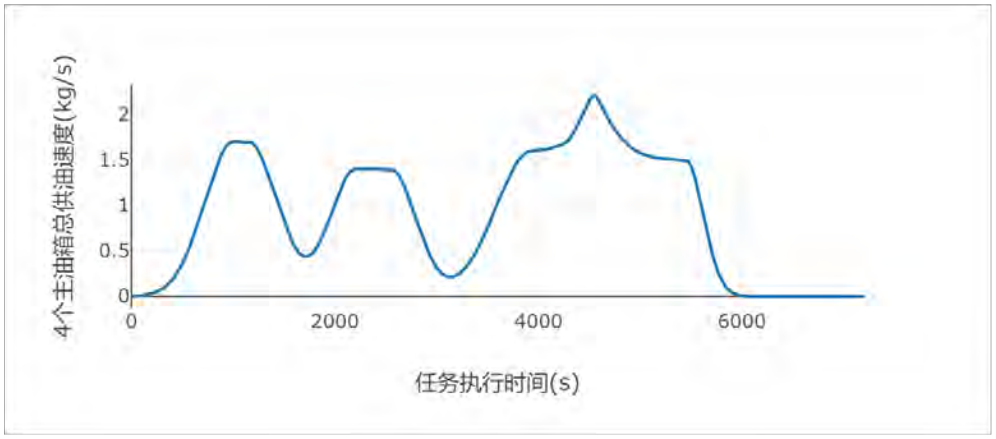
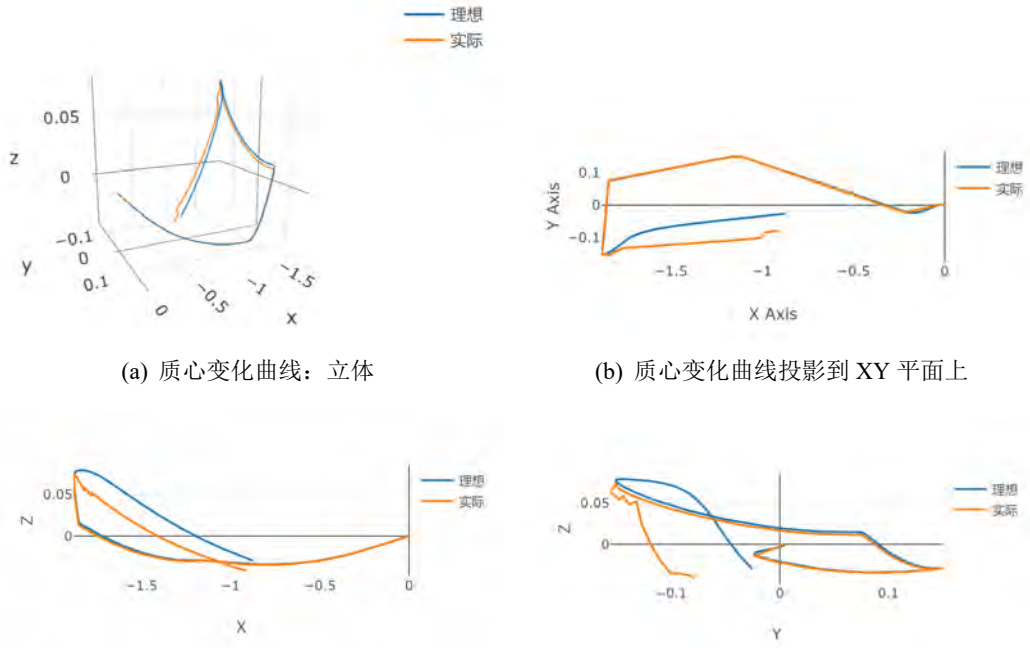


图 **6-4**　第二问 **4** 个主油箱的总供油曲线

行任务的开始时候携带了 7990kg 的燃料，任务结束的时候还剩余 1548.426kg 的燃料，使得燃料的损耗达到了较低水平。

观察实际质心与理想质心的变化曲线图6-5以及图 6-6，可以看到在曲线的末



(a) 质心变化曲线：立体　　(b) 质心变化曲线投影到 XY 平面上

(c) 质心变化曲线投影到 XZ 平面上　　(d) 质心变化曲线投影到 YZ 平面上

图 **6-5**　第二问理想与实际质心变化曲线

端上，理想质心与实际质心的欧式距离变大，观察题目所给数据可以发现，主要原因可能是在末尾一段时间上，发动机的计划供油量过大，导致各个油箱无法通过过多的供油来及时调整飞行器的质心位置从而导致的偏差。

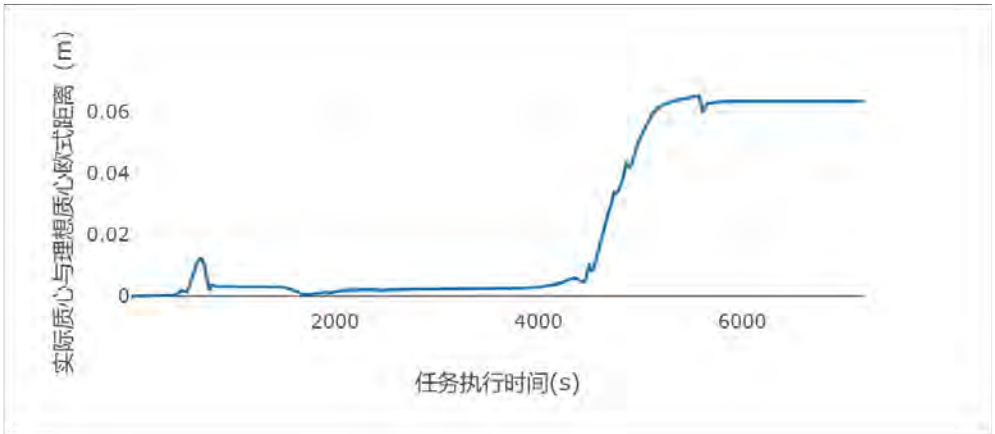


图 **6-6**　第二问实际质心与理想质心欧氏距离变化曲线

6.3 模型的评估

6.3.1 算法的有效性和复杂度

本文采用贪心算法逐一枚举时间长度为 60 的时间片，在每一个时间片上枚举设计好的供油油箱集合内的油箱，找出最优的当前供油油箱组合。其中，对应某一供油油箱组合在某一时间点上使得下一时刻欧式距离最小的供油量，使用 SQP 算法获得。

可见，时间片的规划使得本文的模型满足任意油箱供油时长至少为 60s，使用贪心的思路，在每一时刻寻找最优油箱组合使得本文的模型能够得到一个相对于全局的较优解。

到此，以上的讨论均以某一时刻供油量总和均为指定值。考虑在某一时刻与理想质点距离过远，供油总量将有可能增加，所以考虑使用模拟退火或遗传算法等启发式对供油总量在以上贪心算法中得到的结果进行优化。由于大部分约束条件已被化简或在 SQP 中已被处理完毕，剩余约束条件不多，所以考虑使用模拟退火对模型进行优化。

综上所述，本文的模型在一个约束条件的子集中对问题进行解答，且子集占了全集的大部分。模型最外层使用模拟退火进行指定时刻用油量的优化，内部的目标函数是一个对时间片进行枚举的贪心算法。通过观察实验结果图可以看出，模型具有一定的优势。

对于子问题一的空间复杂度为 $O(n^2)$，时间复杂度为 $O(n^2)$。

对于子问题二的空间复杂度为 $O(n)$，时间复杂度为 $O(n!)$。

7. 问题三：模型的建立与求解

7.1 模型建立

7.1.1 可变初始油量平直飞的趋向理想质心的供油策略模型

优化目标：在第二问的基础上，假定初始油量未定，同样在平直飞条件下制定使得飞行器每一时刻的质心位置 $\overrightarrow{c_1}(t)$ 与理想质心位置 $\overrightarrow{c_2}(t)$ 的欧式距离 $d(t)$ 的最大值达到最小。

目标函数：

与问题二类似目标函数可以定义为：

$$\min \max_t \|\overrightarrow{c_1}(t) - \overrightarrow{c_2}(t)\|_2 \tag{29}$$

决策变量：决策变量的条件与问题二一致。

约束条件：在问题二的约束条件下增加任务结束时，6 个油箱剩余燃料总量至少有 $1m^3$：

$$\sum_{i=1}^{6} u_i(t_{end}) > \rho_o \tag{30}$$

其中 t_{end} 表示最后时刻。

综上所示，最后的建立的模型可以表述为

$$\min \max_{t} \|\overrightarrow{c_1}(t) - \overrightarrow{c_2}(t)\|_2$$

$$s.t \begin{cases} u_i(t) \le U_i \quad (U_i > 0) \\ u_2(t) + u_3(t) + u_4(t) + u_5(t) \ge u(t) \\ u_i(t) \in [0, min(U_i, u_{left\_i})] \\ s_2 + s_3 + s_4 + s_5 \le 2 \\ \sum_{i-1}^{6} s_i \le 3 \\ s_i'(t) = \begin{cases} 0 & cnt_i(t) \ge 60 \\ 1 & cnt_i(t) < 60 \end{cases} \\ \theta = 0 \\ \sum_{i=1}^{6} u_i(t_{end}) > \rho_o \end{cases} \tag{31}$$

7.2 模型的求解

7.2.1 子问题的提取与设计

对于本问题来说，直接对解空间求解，范围过大，效果必然不佳，需要将原始问题拆分为多个子问题，在依次对不同的子问题进去求解。

子问题一：假设每个时刻的总供油量确定，那么需要求解在该供油量下的所有油箱的最优供油策略。

子问题二：需要为每一时刻设计合理的供油量，假设每个时刻的供油策略都是较优解，那么该总供油策略的目标是公式12

子问题三：通过指定初始总油量，使用基于序列二次规划的初始油量分布算法，得出初始油量分布。

针对子问题一、二的求解与第二问的求解方式一致。

针对子问题三的求解，在问题二的单时刻序列二次规划供油策略搜索算法基础上加以改进便可搜寻出较优解。

7.2.2 基于序列二次规划的初始油量分布优化算法

基于序列二次规划的初始油量分布优化算法其流程图如图 7-1所示。

建立目标函数

$$min f_{u_1,u_2,\ldots,u_6}(u_1, u_2, \ldots, u_6) \tag{32}$$

函数 $f_{u_1,u_2,\ldots,u_6}$ 表示返回当初始油量分布为 $u_1, u_2, \ldots, u_6$ 时，返回的飞行器质心位置与理想质心的欧式距离。

其约束条件为：

$$\begin{aligned} &\sum_{i=1}^{6} u_i = u_a \\ &u_i \in (0, a_i b_i c_i) \end{aligned} \tag{33}$$

综上，该模型表示为：

$$min f_{u_1,u_2,\ldots,u_6}(u_1, u_2, \ldots, u_6)$$

$$s.t \begin{cases} \sum_{i=1}^{6} u_i = u_a \\ u_i \in (0, a_i b_i c_i) \end{cases} \tag{34}$$

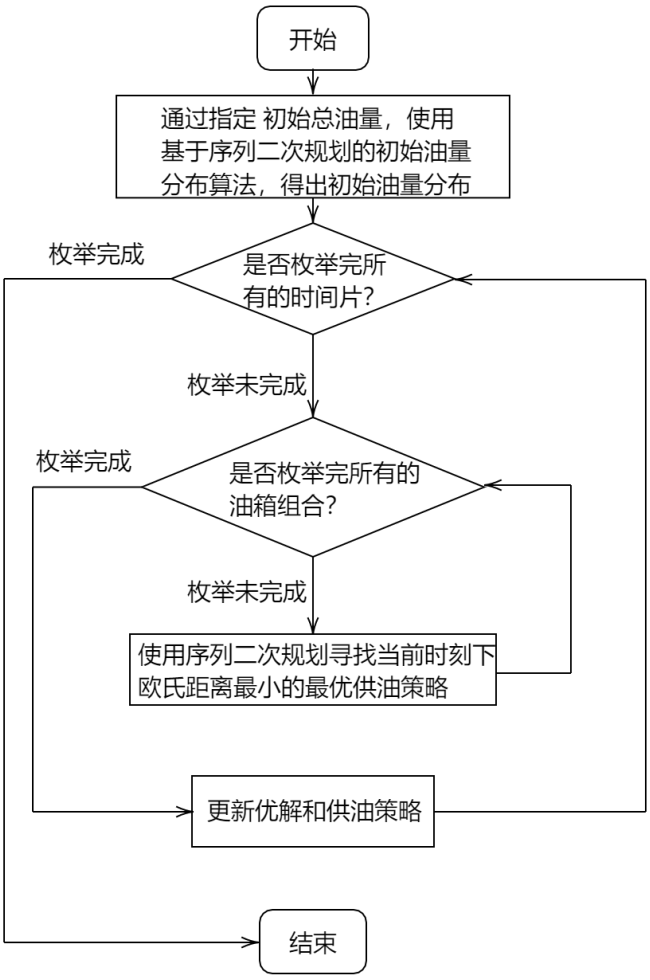


图 **7-1**　基于序列二次规划的初始油量分布优化算法流程图

对于这个非线性规划的求解，同样使用 SLSQP 的方法对求解函数的最小值进行优化。

7.2.3 求解的结果及分析

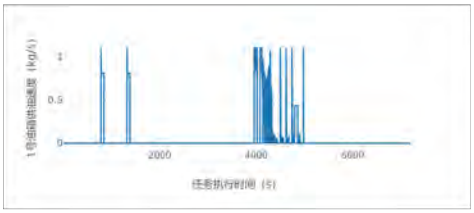
通过对子问题的求解后可以解出本问题的较优解，最终求得飞行器瞬时质心与理想质心距离的最大值为 **0.02906329**m, 四个主油箱的总供油量为 **6805.216557**kg，6 个油箱各自的供油速度曲线如图7-2所示，4 个主油箱的总供油速度曲线如图7-3所示。六个油箱的初始油量如表4所示：

理想与实际质心变化曲线图7-4 第三问实际质心与理想质心欧氏距离变化曲线如图7-5 所示。

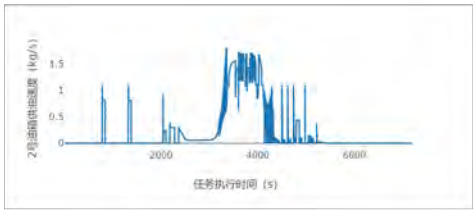
问题三的难点在如何分配初始的油量，初始油量分配的好坏会影响任务执行过程中最大的最小欧式距离。为此本文根据每个油箱的体积占比进行初始化的分配，并通过求解非线性规划来获得一个最好的初始油量的分配策略。最后通过问题二的求解方法，计算出任务执行过程中最大的最小欧式距离。

表 **4**　六个油箱的初始油量

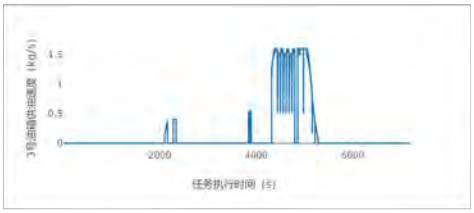
油箱编号	初始油量 (m^3)
1	0.40500000000000002665
2	1.66037795788950615083
3	1.69192095857332214237
4	2.10041270119428302721
5	2.65758838337119529527
6	0.69088784447051698745



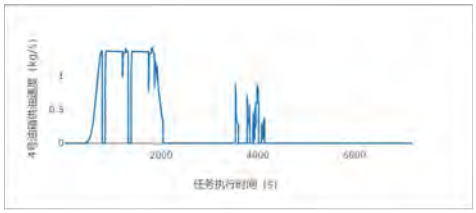
(a) 1 号油箱供油速度曲线



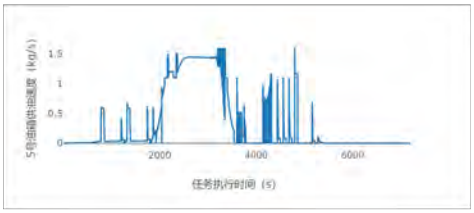
(b) 2 号油箱供油速度曲线



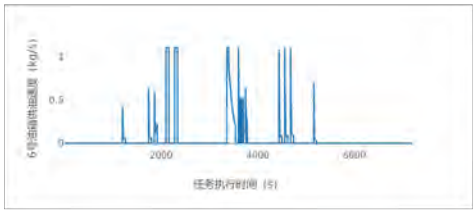
(c) 3 号油箱供油速度曲线



(d) 4 号油箱供油速度曲线



(e) 5 号油箱供油速度曲线



(f) 6 号油箱供油速度曲线

图 **7-2**　第三问六个油箱各自的供油速度曲线

7.3 模型的评估

7.3.1 算法的有效性和复杂度

在本题中，初始油量由模型给出，此时基于第二问的贪心算法有很大的优化空间。首先设计一个算法，给出指定总初始油量，使得油箱与初始理想质心最近的初始油量分布。此处算法使用 SQP 实现，之后使用得到的初始油量分布进行

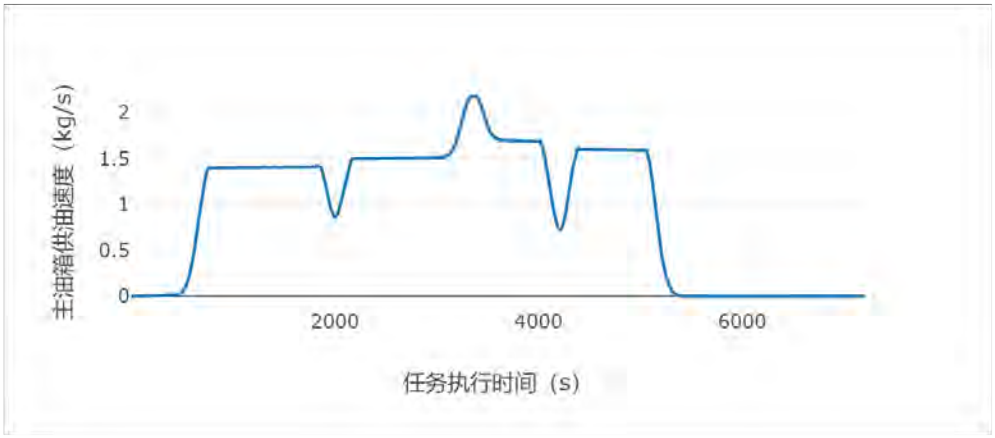
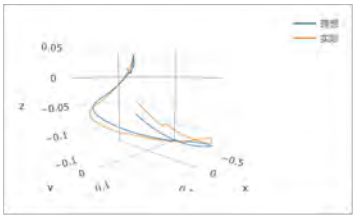
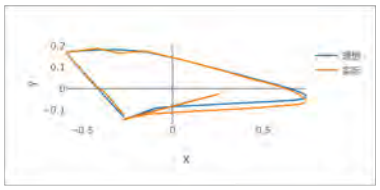


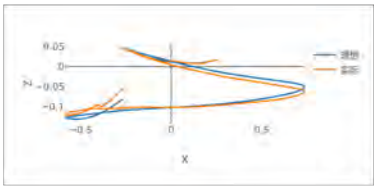
图 **7-3**　第三问 **4** 个主油箱的总供油曲线



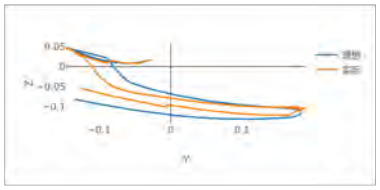
(a) 质心变化曲线：立体



(b) 质心变化曲线投影到 XY 平面上



(c) 质心变化曲线投影到 XZ 平面上



(d) 质心变化曲线投影到 YZ 平面上

图 **7-4**　第三问理想与实际质心变化曲线

第二问的贪心算法得到最小值。然后需要找出一个合适的总初始油量，使得刚才提到的算法能够得到最优解，于是考虑使用模拟退火法来对总初始油量进行优化。其中，目标函数为找到油量分布的 SQP 加上第二问的贪心算法。由于参数空间小，可以很快得到收敛，从而得到一个好的初始油量使得模型达到最优。由于第二问的算法能够得到一个较优解，从而算法具有一定的有效性。

对于子问题一的空间复杂度为 $O(n^2)$，时间复杂度为 $O(n^2)$。

对于子问题二的空间复杂度为 $O(n)$，时间复杂度为 $O(n!)$。

对于子问题三的空间复杂度为 $O(n^2)$，时间复杂度为 $O(n^2)$。

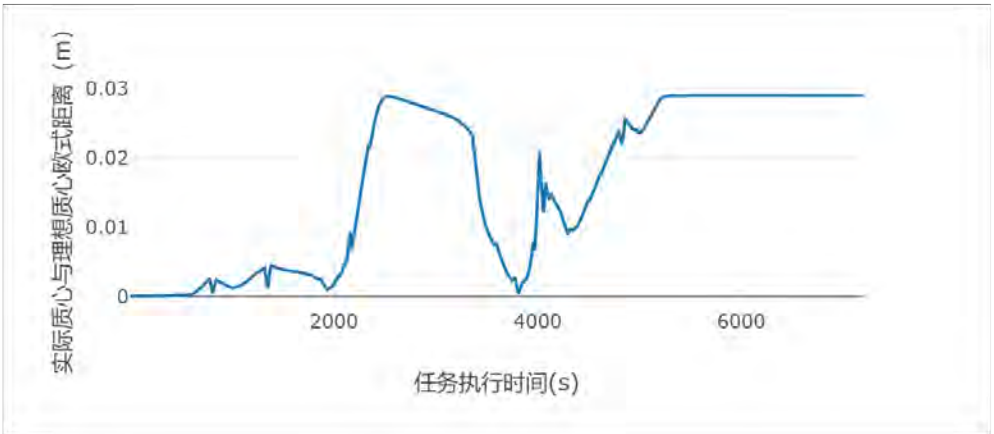


图 **7-5**　第三问实际质心与理想质心欧氏距离变化曲线

8. 问题四：模型的建立与求解

8.1 模型建立

8.1.1 俯仰角可变的趋向理想质心的供油策略模型

优化目标：同样在第二问的基础上，任意俯仰角条件下制定使得飞行器每一时刻的质心位置 $\overrightarrow{c_1}(t)$ 与理想质心位置 $\overrightarrow{c_2}(t)$ 的欧式距离 $d(t)$ 的最大值达到最小。

目标函数：

与问题二类似目标函数可以定义为：

$$\min \max_{t} \|\overrightarrow{c_1}(t) - \overrightarrow{c_2}(t)\|_2 \tag{35}$$

决策变量：决策变量的条件与问题二一致。

约束条件：与第二问的约束条件相比，少了角度 $\theta = 0$ 的限制条件。综上所示，最后的建立的模型可以表述为

$$\min \max_{t} \|\overrightarrow{c_1}(t) - \overrightarrow{c_2}(t)\|_2$$

$$s.t \begin{cases} u_i(t) \le U_i \quad (U_i > 0) \\ u_2(t) + u_3(t) + u_4(t) + u_5(t) \ge u(t) \\ u_i(t) \in [0, min(U_i, u_{left\_i})] \\ s_2 + s_3 + s_4 + s_5 \le 2 \\ \sum_{i-1}^{6} s_i \le 3 \\ s_i'(t) = \begin{cases} 0 & cnt_i(t) \ge 60 \\ 1 & cnt_i(t) < 60 \end{cases} \end{cases} \tag{36}$$

8.2 模型的求解

8.2.1 子问题的提取与设计

与第二问的子问题分解一样，可以分解出两个子模型

子问题一：假设每个时刻的总供油量确定，那么需要求解在该供油量下的所有油箱的最优供油策略。

子问题二：需要为每一时刻设计合理的供油量，假设每个时刻的供油策略都是较优解，那么该总供油策略的目标是公式12

针对子问题一二的求解方式与问题二的基本一致。在去除了角度 $\theta = 0$ 的限制条件后，对子问题二的约束条件进行优化，使得模型在某一时刻的贪心程度更强，如公式 (37) 所示。

$$\begin{aligned} & min f_{a,b}(u_a, u_b) \\ & s.t \begin{cases} u_a + u_b \geq u_i(t) \\ u_a < U_a \\ u_b < U_b \end{cases} \end{aligned} \tag{37}$$

因为理想质心始终在原点上，所以可以在每一时刻都始终以最贪心的方式获取较优解，而无需考虑理想质心的移动。

8.2.2 求解的结果及分析

通过对两个子问题的求解后可以解出本问题的较优解，最终求得飞行器瞬时质心与理想质心距离的最大值为 **0.03059667**m, 四个主油箱的总供油量为 **7632.230**kg，6 个油箱各自的供油速度曲线如图8-1所示，4 个主油箱的总供油速度曲线如图8-2所示。

理想与实际质心变化曲线图8-3 第四问实际质心与理想质心欧氏距离变化曲线如图8-4 所示。

图8-3中的曲线变化非常不平滑，这是因为质心变动的数量级很小，只在 2cm 的范围内进行波动，所以实际上的制定的策略较优。

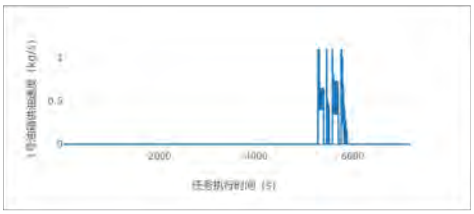
8.3 模型的评估

8.3.1 算法的有效性和复杂度

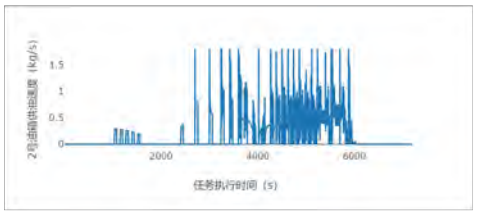
本问题时单目标规划问题，在本模型中，我们以供油曲线为主线，通过将时间分为长度为 60 的时间片，和设计供油组合来满足题目的限制条件，减少了模型设计时的繁琐，和算法设计上的复杂度。在本问题中，使用贪心策略求解整个任务执行过程中的最小的最大欧式距离，通过，求每一时刻实际质心与理想质心的欧式距离的最小值，获得整个任务执行过程中较优的一个最小的最大欧式距离。对于每一时刻的最小欧式距离，则通过求解一个非线性规划问题获得，使用 SQP 算法求解这个非线性规划问题获得该时刻的最小欧式距离。同时，在计算该时刻的质心时，使用问题一中处于不同角度和不同油量计算质心的方法。结果表明，SQP 算法在求解一个时刻时的最小欧式距离时具有很好的效果，同时使用全局的贪心策略求得的解也非常的优秀。

对于子问题一的空间复杂度为 $O(n^2)$，时间复杂度为 $O(n^4)$。

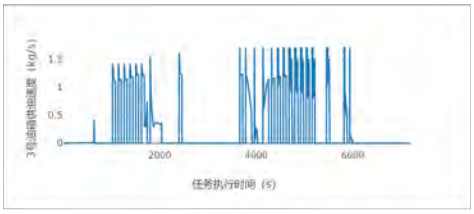
对于子问题二的空间复杂度为 $O(n)$，时间复杂度为 $O(n!)$。



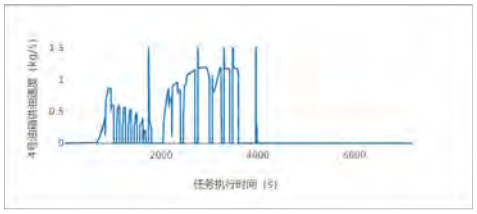
(a) 1 号油箱供油速度曲线



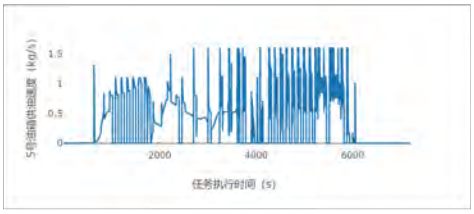
(b) 2 号油箱供油速度曲线



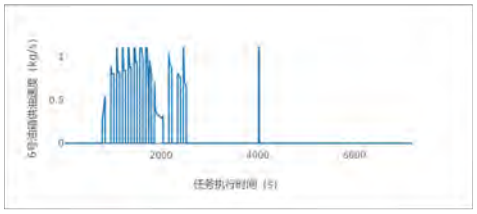
(c) 3 号油箱供油速度曲线



(d) 4 号油箱供油速度曲线



(e) 5 号油箱供油速度曲线



(f) 6 号油箱供油速度曲线

图 **8-1**　第四问六个油箱各自的供油速度曲线

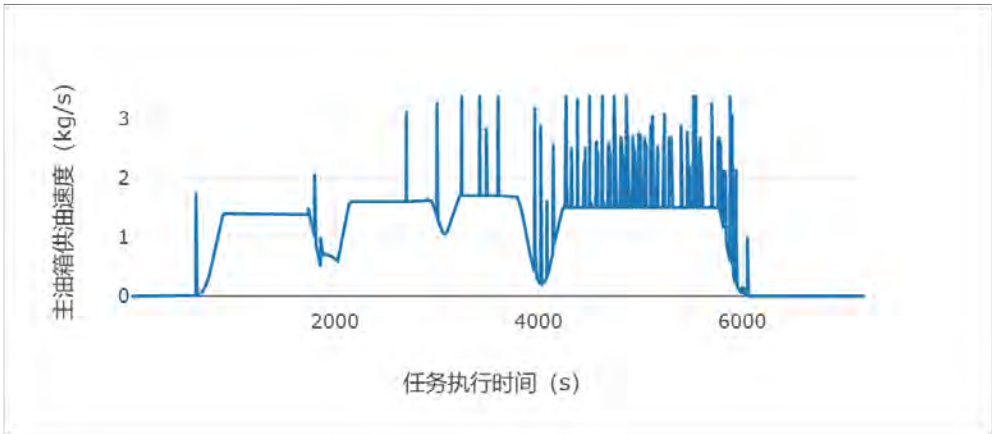
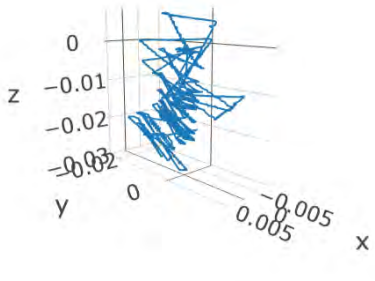
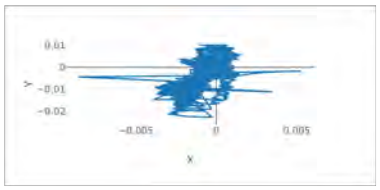


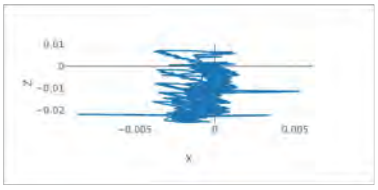
图 **8-2**　第四问 **4** 个主油箱的总供油曲线



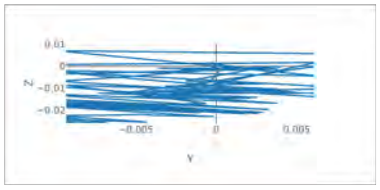
(a) 质心变化曲线：立体



(b) 质心变化曲线投影到 XY 平面上



(c) 质心变化曲线投影到 XZ 平面上



(d) 质心变化曲线投影到 YZ 平面上

图 **8-3**　第四问理想与实际质心变化曲线

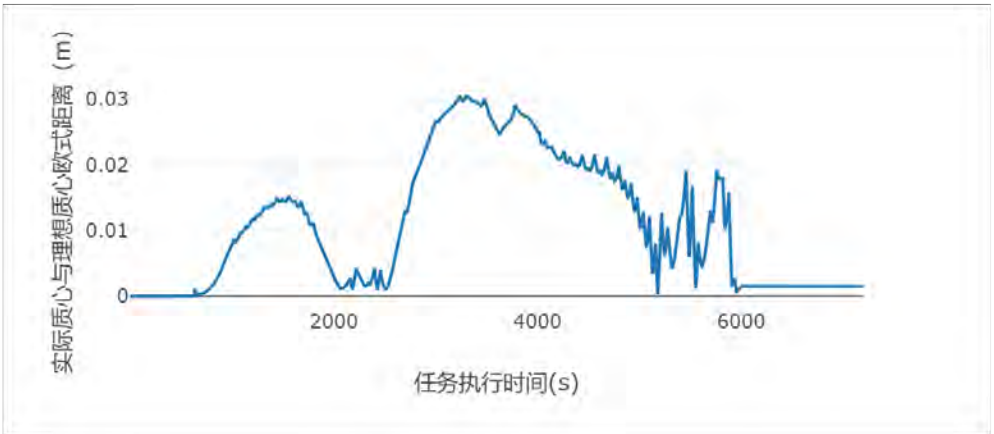


图 **8-4**　第四问实际质心与理想质心欧氏距离变化曲线

9. 模型的评价

9.1 模型的优点

1. 本文在正确分析了题意得基础上。建立了合理的，满足所有约束条件的，科学的数学模型，为求最小的所有时刻最大欧氏距离准备了条件。
2. 通过对时间进行分片的策略，巧妙的解决了持续供油 60 秒的约束条件，使算法的设计变得简单。
3. 通过对油箱供油的分析，设计了多种供油组合，很好的解决了油箱供油的限制，减小了之后算法的时间复杂度。
4. 本文的数学模型，以满足耗油量为优先，不会对油量造成浪费。
5. 本文的数学模型能获得较好的最小欧式距离，且算法时间复杂度较低。

9.2 模型的缺点

1. 本文为了简化模型，对时间进行分片，使得供油时间不是非常的灵活。
2. 为了减少算法的时间复杂度，减少了几种油箱的组合。可能会导致不能得到最优的结果。

10. 参考文献

[1] 金先龙，张淑敏，非满载液罐汽车坡道行驶时液体质心位置的计算及分析[J]，专用汽车，04 期：第 13-17 页，1990。

[2] Dieter Kraft, D，A software package for sequential quadratic programming，Germany：Wiss. Berichtswesen d. DFVLR，88-28，1998。

[3] Nocedal, J. and Wright, S. Numerical Optimization, 2nd. ed., Ch. 18. Springer, 2006。

[4] You, Fengqi. Lecture Notes, Chemical Engineering 345 Optimization. Northwestern University, 2015。

[5] 杨若黎, 顾基发，一种高效的模拟退火全局优化算法 [J]. 系统工程理论与实践，17(5): 30-36，1997。

[6] 李超群，刘智慧，张玉洁，质心公式推导及其在求解积分中的应用 [J],《数学学习与研究》01 期：69-70 , 2014。

附录 A 程序代码

1.1 第一问代码

```
import numpy as np
from scipy.integrate import dblquad


def get_relative_D(theta, s, a, c):
    theta = np.radians(theta)
    a_h = a * np.sin(theta)
    c_h = c * np.cos(theta)

    if theta == 0.0:
        return 0.0, a, lambda x: 0.0, lambda x, val=s / a: val
    if a_h >= c_h:
        s_c = c * c / np.tan(theta) / 2.0
        s_a = a * c - s_c
        if s <= s_c:

            x_max = np.sqrt(2.0 * s / np.tan(theta))
            gfun = lambda x: 0.0

            def hfun(x, x_max=x_max, tan_theta=np.tan(theta)):
                return (x_max - x) * tan_theta

            return 0.0, x_max, gfun, hfun
        elif s <= s_a:
            x_max = c / np.tan(theta) + (s - s_c) / c
            gfun = lambda x: 0.0

            def hfun(x, x_max=x_max, theta=theta, c=c, t=(s -
                s_c) / c):
                return c if x <= t else (x_max - x) *
                    np.tan(theta)

            return 0.0, x_max, gfun, hfun
        else:
            x_max = a
            lft = np.sqrt(2 * (a * c - s) / np.tan(theta))
            gfun = lambda x: 0.0

            def hfun(x, x_ex=a - lft + c / np.tan(theta),
                theta=theta, c=c, t=a - lft):
                return c if x <= t else (x_ex - x) * np.tan(theta)

            return 0.0, x_max, gfun, hfun
    else:
        s_a = a * a * np.tan(theta) / 2.0
        s_c = a * c - s_a
        if s <= s_a:
            x_max = np.sqrt(2.0 * s / np.tan(theta))
            gfun = lambda x: 0.0
```

```
            def hfun(x, x_max=x_max, tan_theta=np.tan(theta)):
                return (x_max - x) * tan_theta

            return 0.0, x_max, gfun, hfun
        elif s <= s_c:
            x_max = a
            gfun = lambda x: 0.0
            c_u = a * np.tan(theta) + (s - s_a) / a

            def hfun(x, x_ex=c_u / np.tan(theta), theta=theta):
                return (x_ex - x) * np.tan(theta)

            return 0.0, x_max, gfun, hfun
        else:
            x_max = a
            lft = np.sqrt(2 * (a * c - s) / np.tan(theta))
            gfun = lambda x: 0.0

            def hfun(x, x_ex=a - lft + c / np.tan(theta),
                theta=theta, c=c, t=a - lft):
                return c if x <= t else (x_ex - x) * np.tan(theta)

            return 0.0, x_max, gfun, hfun


def get_relative_xz(theta, s, a, c):
    if theta == 0.0:
        return 0.0, s / a / 2.0 - c / 2.0
    relative_D = get_relative_D(np.abs(theta), s, a, c)
    x = dblquad(lambda z, x: x, *relative_D)[0]/s
    z = dblquad(lambda z, x: z, *relative_D)[0]/s
    if theta > 0:
        return x - a / 2.0, z - c / 2.0
    else:
        return a / 2.0 - x, z - c / 2.0


def get_real_xyz(theta, v, x, y, z, a, b, c):
    rt_x, rt_z = get_relative_xz(theta, v / b, a, c)
    return x + rt_x, y, z + rt_z


def centroid(env, tanks, bird_theta, used_fuel):
    theta = bird_theta['theta']
    x = env['bird_x'] * env['bird_m']
    y = env['bird_y'] * env['bird_m']
    z = env['bird_z'] * env['bird_m']
    m_sum = env['bird_m']
    for tank in tanks:
        left = tank['first'] - used_fuel[str(int(tank['id']))]
            / env['fuel_density']
        m = left * env['fuel_density']
        x_t, y_t, z_t = get_real_xyz(theta,
                                     left,
```

```
                                tank['x'],
                                tank['y'],
                                tank['z'],
                                tank['a'],
                                tank['b'],
                                tank['c'])
        x += x_t * m
        y += y_t * m
        z += z_t * m
        m_sum += m
    return x / m_sum, y / m_sum, z / m_sum


def txt_to_table(path):
    table = []
    with open(path, 'r') as fp:
        attr = fp.readline().split();
        for line in fp.readlines():
            table.append(dict(zip(attr, map(float,
                line.split()))))
    return table


if __name__ == "__main__":
    tanks = txt_to_table('tank.txt')
    env = txt_to_table('env.txt')[0]
    used_fuels = txt_to_table('used_fuel.txt')
    bird_thetas = txt_to_table('bird_theta.txt')
    for i, x in enumerate(used_fuels):
        used_fuels[i]['2'] -= used_fuels[i]['1']
        used_fuels[i]['5'] -= used_fuels[i]['6']
        if i > 0:
            id = 1
            while str(id) in x:
                used_fuels[i][str(id)] += used_fuels[i -
                    1][str(id)]
                id += 1
    print(used_fuels[-1])
    with open('Problem1.txt', 'w') as fp:
        fp.write('time\tx\ty\tz\n')
        for i, (used_fuel, bird_theta) in
            enumerate(zip(used_fuels, bird_thetas)):
            t = i + 1
            print(t)
            fp.write(str(t) + ' ')
            fp.write('%f %f %f\n' % centroid(env, tanks,
                bird_theta, used_fuel))
```

1.2 第二问代码

```
import numpy as np
from scipy.integrate import dblquad
from scipy import optimize
```

```
from copy import deepcopy


def get_relative_D(theta, s, a, c):
    theta = np.radians(theta)
    a_h = a * np.sin(theta)
    c_h = c * np.cos(theta)

    if theta == 0.0:
        return 0.0, a, lambda x: 0.0, lambda x, val=s / a: val
    if a_h >= c_h:
        s_c = c * c / np.tan(theta) / 2.0
        s_a = a * c - s_c
        if s <= s_c:

            x_max = np.sqrt(2.0 * s / np.tan(theta))
            gfun = lambda x: 0.0

            def hfun(x, x_max=x_max, tan_theta=np.tan(theta)):
                return (x_max - x) * tan_theta

            return 0.0, x_max, gfun, hfun
        elif s <= s_a:
            x_max = c / np.tan(theta) + (s - s_c) / c
            gfun = lambda x: 0.0

            def hfun(x, x_max=x_max, theta=theta, c=c, t=(s -
                s_c) / c):
                return c if x <= t else (x_max - x) *
                    np.tan(theta)

            return 0.0, x_max, gfun, hfun
        else:
            x_max = a
            lft = np.sqrt(2 * (a * c - s) / np.tan(theta))
            gfun = lambda x: 0.0

            def hfun(x, x_ex=a - lft + c / np.tan(theta),
                theta=theta, c=c, t=a - lft):
                return c if x <= t else (x_ex - x) * np.tan(theta)

            return 0.0, x_max, gfun, hfun
    else:
        s_a = a * a * np.tan(theta) / 2.0
        s_c = a * c - s_a
        if s <= s_a:
            x_max = np.sqrt(2.0 * s / np.tan(theta))
            gfun = lambda x: 0.0

            def hfun(x, x_max=x_max, tan_theta=np.tan(theta)):
                return (x_max - x) * tan_theta

            return 0.0, x_max, gfun, hfun
        elif s <= s_c:
```

```
            x_max = a
            gfun = lambda x: 0.0
            c_u = a * np.tan(theta) + (s - s_a) / a

            def hfun(x, x_ex=c_u / np.tan(theta), theta=theta):
               return (x_ex - x) * np.tan(theta)

            return 0.0, x_max, gfun, hfun
         else:
            x_max = a
            lft = np.sqrt(2 * (a * c - s) / np.tan(theta))
            gfun = lambda x: 0.0

            def hfun(x, x_ex=a - lft + c / np.tan(theta),
               theta=theta, c=c, t=a - lft):
               return c if x <= t else (x_ex - x) * np.tan(theta)

            return 0.0, x_max, gfun, hfun


def get_relative_xz(theta, s, a, c):
   if theta == 0.0:
      return 0.0, s / a / 2.0 - c / 2.0
   relative_D = get_relative_D(np.abs(theta), s, a, c)
   x = dblquad(lambda z, x: x, *relative_D)[0] / s
   z = dblquad(lambda z, x: z, *relative_D)[0] / s
   if theta > 0:
      return x - a / 2.0, z - c / 2.0
   else:
      return a / 2.0 - x, z - c / 2.0


def get_real_xyz(theta, v, x, y, z, a, b, c):
   rt_x, rt_z = get_relative_xz(theta, v / b, a, c)
   return x + rt_x, y, z + rt_z


def centroid(env, tanks, bird_theta, used_fuel_v):
   theta = bird_theta['theta']
   x = env['bird_x'] * env['bird_m']
   y = env['bird_y'] * env['bird_m']
   z = env['bird_z'] * env['bird_m']
   m_sum = env['bird_m']
   for tank in tanks:
      left = tank['first'] - used_fuel_v[str(int(tank['id']))]
      m = left * env['fuel_density']
      x_t, y_t, z_t = get_real_xyz(theta,
                                   left,
                                   tank['x'],
                                   tank['y'],
                                   tank['z'],
                                   tank['a'],
                                   tank['b'],
                                   tank['c'])
```

```
            x += x_t * m
            y += y_t * m
            z += z_t * m
            m_sum += m
        return x / m_sum, y / m_sum, z / m_sum


    def txt_to_table(path):
        table = []
        with open(path, 'r') as fp:
            attr = fp.readline().split();
            for line in fp.readlines():
                table.append(dict(zip(attr, map(float,
                    line.split()))))
        return table


    def find_best_change(env, tanks, bird_theta, used_fuel,
        use_tanks, target, min_fuel):
        def distance(change):
            for i, use_tank in enumerate(use_tanks):
                used_fuel[use_tank] += change[i]
            x, y, z = centroid(env, tanks, bird_theta, used_fuel)
            for i, use_tank in enumerate(use_tanks):
                used_fuel[use_tank] -= change[i]
            return sum(((x - target[0]) ** 2, (y - target[1]) ** 2,
                (z - target[2]) ** 2))

        tanks_bounds = [(1e-15, max(1e-15,min(tanks[int(use_tank)
            - 1]['first'] - used_fuel[use_tank],
                           tanks[int(use_tank) - 1]['upper']))) for
                               use_tank in
                    use_tanks]

        cons = (
            {'type': 'eq', 'fun': lambda x, min_fuel=min_fuel:
                np.sum(x) - min_fuel}
            # {'type': 'ineq', 'fun': lambda x, min_fuel=min_fuel:
                min_fuel - np.sum(x) + 0.0001}
        )

        distance_min = optimize.minimize(distance,
            np.zeros(len(use_tanks)), bounds=tanks_bounds,
            constraints=cons,
                                    method='SLSQP')

        if distance_min.success:
            return distance_min.fun, tuple(distance_min.x)
        else:
            return -1, use_tanks


    use_tanks_list = [
        ('1', '3'),
```

```
        ('1', '4'),
        ('1', '5'),
        ('2', '3'),
        ('2', '4'),
        ('2', '5'),
        ('2', '6'),
        ('3', '4'),
        ('3', '5'),
        ('3', '6'),
        ('4', '5'),
        ('4', '6')
    ]

    if __name__ == "__main__":
        env = txt_to_table('env.txt')[0]
        tanks = txt_to_table('tank.txt')
        bird_theta = {'theta': 0.0}
        min_fuels = txt_to_table('min_fuel.txt')
        for tank in tanks:
            tank['upper'] /= env['fuel_density']
        for min_fuel in min_fuels:
            min_fuel['min_fuel'] /= env['fuel_density']
        image_xyzs = txt_to_table('image_xyz.txt')

        sum_min_fuels = np.sum([min_fuel['min_fuel'] for min_fuel
            in min_fuels])
        first_all_tanks = np.sum([tank['first'] for tank in tanks])
        AA = (first_all_tanks - sum_min_fuels) / 7200.0

        used_fuel_v = {'1': 0, '2': 0, '3': 0, '4': 0, '5': 0,
            '6': 0}
        ans_use_fuel_per_time = []
        ans_xyz_per_time = []
        ans_dis_per_time = []
        ans_max = 0
        for k in range(120):
            kt = k * 60
            now_max = 9999999999
            fuel_v_wanna_use = None
            fuel_v_wanna_use_per_time_t = None
            xyz_wanna_use_per_time_t = None
            dis_wanna_use_per_time_t = None
            for use_tanks in use_tanks_list:
                t_used_fuel_per_time = []
                t_xyz_per_time = []
                t_dis_per_time = []
                t_used_fuel_v = deepcopy(used_fuel_v)
                t_max = 0
                for i in range(60):
                    time = kt + i
                    min_fuel = min_fuels[time]['min_fuel']
                    uppers = np.sum([tanks[int(use_tank) -
                        1]['upper'] for use_tank in use_tanks])
                    lefts = np.sum([tanks[int(use_tank) - 1]['first']
```

```
                - t_used_fuel_v[use_tank] for use_tank in
                use_tanks])
            if min_fuel > uppers or min_fuel > lefts:
                t_max = -1
                break

            dis, change = find_best_change(env, tanks,
                bird_theta, t_used_fuel_v, use_tanks,
                                    [image_xyzs[time][c] for c
                                        in 'xyz'], min_fuel +
                                        1e-8)

            t_dict = {'1': 0.0, '2': 0.0, '3': 0.0, '4': 0.0,
                '5': 0.0, '6': 0.0}
            if dis == -1:
                t_max = -1
                break
            for i, tank in enumerate(use_tanks):
                if tank == '1':
                    t_dict['2'] += change[i] *
                        env['fuel_density']
                if tank == '6':
                    t_dict['5'] += change[i] *
                        env['fuel_density']
                t_dict[tank] += change[i] * env['fuel_density']
            x,y,z =
                centroid(env,tanks,bird_theta,t_used_fuel_v)
            t_used_fuel_per_time.append(t_dict)
            t_dis_per_time.append(np.sqrt(dis))
            t_xyz_per_time.append({'x':x,'y':y,'z':z})
            t_max = max(t_max, dis)
            for i, use_tank in enumerate(use_tanks):
                t_used_fuel_v[use_tank] += change[i]
        if t_max != -1 and t_max < now_max:
            now_max = t_max
            fuel_v_wanna_use = t_used_fuel_v
            fuel_v_wanna_use_per_time_t = t_used_fuel_per_time
            xyz_wanna_use_per_time_t = t_xyz_per_time
            dis_wanna_use_per_time_t = t_dis_per_time
    if fuel_v_wanna_use:
        used_fuel_v = fuel_v_wanna_use
        ans_use_fuel_per_time.extend(fuel_v_wanna_use_per_time_t)
        ans_xyz_per_time.extend(xyz_wanna_use_per_time_t)
        ans_dis_per_time.extend(dis_wanna_use_per_time_t)

    else:
        print("no solution")
        exit(0)
    ans_max = max(ans_max, now_max)
    print(k, np.sqrt(ans_max))
with open('use_fuel_per_time.txt', 'w') as fp:
    fp.write('ans_max:%.20f\n' % (np.sqrt(ans_max)))
    fp.write('time\t1\t2\t3\t4\t5\t6\n')
    for i, per_time in enumerate(ans_use_fuel_per_time):
```

```
            fp.write('%d\t%.20f\t%.20f\t%.20f\t%.20f\t%.20f\t%.20f\n'
               % (
               i + 1, per_time['1'], per_time['2'],
                  per_time['3'], per_time['4'], per_time['5'],
                  per_time['6']))
      with open('dis_per_time.txt', 'w') as fp:
         fp.write('time\tdis\n')
         for i, per_time in enumerate(ans_dis_per_time):
            fp.write('%d\t%.20f\n' % (i+1,per_time))
      with open('xyz_per_time.txt', 'w') as fp:
         fp.write('time\tx\ty\tz\n')
         for i, per_time in enumerate(ans_xyz_per_time):
            fp.write('%d\t%.20f\t%.20f\t%.20f\n' %
               (i+1,per_time['x'],per_time['y'],per_time['z']))
```

```
   from sko.SA import SA
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from scipy.integrate import dblquad
from scipy import optimize
from copy import deepcopy


def get_relative_D(theta, s, a, c):
   theta = np.radians(theta)
   a_h = a * np.sin(theta)
   c_h = c * np.cos(theta)

   if theta == 0.0:
      return 0.0, a, lambda x: 0.0, lambda x, val=s / a: val
   if a_h >= c_h:
      s_c = c * c / np.tan(theta) / 2.0
      s_a = a * c - s_c
      if s <= s_c:

         x_max = np.sqrt(2.0 * s / np.tan(theta))
         gfun = lambda x: 0.0

         def hfun(x, x_max=x_max, tan_theta=np.tan(theta)):
            return (x_max - x) * tan_theta

         return 0.0, x_max, gfun, hfun
      elif s <= s_a:
         x_max = c / np.tan(theta) + (s - s_c) / c
         gfun = lambda x: 0.0

         def hfun(x, x_max=x_max, theta=theta, c=c, t=(s - s_c)
            / c):
            return c if x <= t else (x_max - x) * np.tan(theta)

         return 0.0, x_max, gfun, hfun
      else:
```

```
            x_max = a
            lft = np.sqrt(2 * (a * c - s) / np.tan(theta))
            gfun = lambda x: 0.0

            def hfun(x, x_ex=a - lft + c / np.tan(theta),
                theta=theta, c=c, t=a - lft):
                return c if x <= t else (x_ex - x) * np.tan(theta)

            return 0.0, x_max, gfun, hfun
        else:
            s_a = a * a * np.tan(theta) / 2.0
            s_c = a * c - s_a
            if s <= s_a:
                x_max = np.sqrt(2.0 * s / np.tan(theta))
                gfun = lambda x: 0.0

                def hfun(x, x_max=x_max, tan_theta=np.tan(theta)):
                    return (x_max - x) * tan_theta

                return 0.0, x_max, gfun, hfun
            elif s <= s_c:
                x_max = a
                gfun = lambda x: 0.0
                c_u = a * np.tan(theta) + (s - s_a) / a

                def hfun(x, x_ex=c_u / np.tan(theta), theta=theta):
                    return (x_ex - x) * np.tan(theta)

                return 0.0, x_max, gfun, hfun
            else:
                x_max = a
                lft = np.sqrt(2 * (a * c - s) / np.tan(theta))
                gfun = lambda x: 0.0

                def hfun(x, x_ex=a - lft + c / np.tan(theta),
                    theta=theta, c=c, t=a - lft):
                    return c if x <= t else (x_ex - x) * np.tan(theta)

                return 0.0, x_max, gfun, hfun


    def get_relative_xz(theta, s, a, c):
        if theta == 0.0:
            return 0.0, s / a / 2.0 - c / 2.0
        relative_D = get_relative_D(np.abs(theta), s, a, c)
        x = dblquad(lambda z, x: x, *relative_D)[0] / s
        z = dblquad(lambda z, x: z, *relative_D)[0] / s
        if theta > 0:
            return x - a / 2.0, z - c / 2.0
        else:
            return a / 2.0 - x, z - c / 2.0


    def get_real_xyz(theta, v, x, y, z, a, b, c):
```

```
   rt_x, rt_z = get_relative_xz(theta, v / b, a, c)
   return x + rt_x, y, z + rt_z


def centroid(env, tanks, bird_theta, used_fuel_v):
   theta = bird_theta['theta']
   x = env['bird_x'] * env['bird_m']
   y = env['bird_y'] * env['bird_m']
   z = env['bird_z'] * env['bird_m']
   m_sum = env['bird_m']
   for tank in tanks:
      left = tank['first'] - used_fuel_v[str(int(tank['id']))]
      m = left * env['fuel_density']
      x_t, y_t, z_t = get_real_xyz(theta,
                                   left,
                                   tank['x'],
                                   tank['y'],
                                   tank['z'],
                                   tank['a'],
                                   tank['b'],
                                   tank['c'])
      x += x_t * m
      y += y_t * m
      z += z_t * m
      m_sum += m
   return x / m_sum, y / m_sum, z / m_sum


def txt_to_table(path):
   table = []
   with open(path, 'r') as fp:
      attr = fp.readline().split();
      for line in fp.readlines():
         table.append(dict(zip(attr, map(float, line.split()))))
   return table


def find_best_change(env, tanks, bird_theta, used_fuel,
   use_tanks, target, min_fuel):
   def distance(change):
      for i, use_tank in enumerate(use_tanks):
         used_fuel[use_tank] += change[i]
      x, y, z = centroid(env, tanks, bird_theta, used_fuel)
      for i, use_tank in enumerate(use_tanks):
         used_fuel[use_tank] -= change[i]
      return sum(((x - target[0]) ** 2, (y - target[1]) ** 2, (z
         - target[2]) ** 2))

   tanks_bounds = [(0, max(0,min(tanks[int(use_tank) -
      1]['first'] - used_fuel[use_tank],
                    tanks[int(use_tank) - 1]['upper']))) for
                       use_tank in
               use_tanks]
```

```
    cons = (
        {'type': 'eq', 'fun': lambda x, min_fuel=min_fuel:
            np.sum(x) - min_fuel}
        # {'type': 'ineq', 'fun': lambda x, min_fuel=min_fuel:
            min_fuel - np.sum(x) + 0.0001}
    )

    distance_min = optimize.minimize(distance,
        np.zeros(len(use_tanks)), bounds=tanks_bounds,
        constraints=cons)

    # print(distance_min)
    if distance_min.success:
        return distance_min.fun, tuple(distance_min.x)
    else:
        return -1, use_tanks


use_tanks_list = [
    ('1', '3'),
    ('1', '4'),
    ('1', '5'),
    ('2', '3'),
    ('2', '4'),
    ('2', '5'),
    ('2', '6'),
    ('3', '4'),
    ('3', '5'),
    ('3', '6'),
    ('4', '5'),
    ('4', '6')
]


class SA2(SA):
    # def get_new_x(self, x):
    #     u = np.random.uniform(-1, 1, size=self.n_dims)
    #     x_new = x + 20 * np.sign(u) * self.T * ((1 + 1.0 /
        self.T) ** np.abs(u) - 1.0)
    #     return x_new
    def get_new_x(self, x):
        # x_new = np.array(x, dtype=bool)
        # idx_t = np.random.randint(120) # 随机改变一个时间
        # idx_i = idx_list[np.random.randint(12)]
        # x_new[idx_t, :] = False
        # x_new[idx_t, idx_i] = True
        return generate(x, self.T / self.T_max+0.01)


p = np.array([2.4, 2.2, 2.9, 3.6, 3.4, 4.1, 2.3, 4.0, 4.7, 2.9,
    4.5, 2.7])
p /= sum(p)
```

```
def generate(old_strategy=None, cx=1):
   global AA, min_fuels
   global p
   if old_strategy is not None:
      for i in range(7200):
         if np.random.randn() < cx*(i/7200):
            old_strategy[i] = min_fuels[i]['min_fuel'] + AA *
               np.random.rand()*(1-i/7200)
      return old_strategy
   else:
      strategy = np.zeros((7200,), dtype=float)
      for i in range(7200):
         strategy[i] = min_fuels[i]['min_fuel'] + AA *
            np.random.rand()
      return strategy

CNT=0

def demo_func(x):
   global env, tanks, bird_theta, image_xyzs,CNT
   used_fuel_v = {'1': 0, '2': 0, '3': 0, '4': 0, '5': 0, '6': 0}
   ans_use_fuel_per_time = []
   ans_max = 0
   for k in range(120):
      kt = k * 60
      now_max = 9999999999
      fuel_v_wanna_use = None
      fuel_v_wanna_use_per_time_t = None
      for use_tanks in use_tanks_list:
         t_used_fuel_per_time = []
         t_used_fuel_v = deepcopy(used_fuel_v)
         t_max = 0
         for i in range(60):
            time = kt + i
            dis, change = find_best_change(env, tanks,
               bird_theta, t_used_fuel_v, use_tanks,
                                    [image_xyzs[time][c] for c in
                                       'xyz'], x[time])
            t_dict = {'1': 0.0, '2': 0.0, '3': 0.0, '4': 0.0,
               '5': 0.0, '6': 0.0}
            if dis == -1:
               t_max = -1
               break
            for i, tank in enumerate(use_tanks):
               if tank == '1':
                  t_dict['2'] += change[i] * env['fuel_density']
               if tank == '6':
                  t_dict['5'] += change[i] * env['fuel_density']
               t_dict[tank] += change[i] * env['fuel_density']
            t_used_fuel_per_time.append(t_dict)
            t_max = max(t_max, dis)
            for i, use_tank in enumerate(use_tanks):
               t_used_fuel_v[use_tank] += change[i]
         if t_max != -1 and t_max < now_max:
```

```
            now_max = t_max
            fuel_v_wanna_use = t_used_fuel_v
            fuel_v_wanna_use_per_time_t = t_used_fuel_per_time
        if fuel_v_wanna_use:
            used_fuel_v = fuel_v_wanna_use
            ans_use_fuel_per_time.extend(fuel_v_wanna_use_per_time_t)
        else:
            print("no solution")
            return 999999
        ans_max = max(ans_max, now_max)
        print(k, np.sqrt(ans_max))
    with open('use_fuel_per_time_sa_%d.txt'%(CNT), 'w') as fp:
        fp.write('ans_max:%f\n'%(np.sqrt(ans_max)))
        fp.write('time\t1\t2\t3\t4\t5\t6\n')
        for i, per_time in enumerate(ans_use_fuel_per_time):
            fp.write('%d\t%.20f\t%.20f\t%.20f\t%.20f\t%.20f\t%.20f\n'
                % (
            i + 1, per_time['1'], per_time['2'], per_time['3'],
                per_time['4'], per_time['5'], per_time['6']))
        CNT+=1
    return np.sqrt(ans_max)*1000


if __name__ == '__main__':
    env = txt_to_table('env.txt')[0]
    tanks = txt_to_table('tank.txt')
    bird_theta = {'theta': 0.0}
    min_fuels = txt_to_table('min_fuel.txt')
    for tank in tanks:
        tank['upper'] /= env['fuel_density']
    for min_fuel in min_fuels:
        min_fuel['min_fuel'] /= env['fuel_density']
    sum_min_fuels = np.sum([min_fuel['min_fuel'] for min_fuel in
        min_fuels])
    first_all_tanks = np.sum([tank['first'] for tank in tanks])
    AA = (first_all_tanks - sum_min_fuels) / 7200
    image_xyzs = txt_to_table('image_xyz.txt')

    x0 = generate()
    sa = SA2(func=demo_func, x0=x0, T_max=999, T_min=1e-9,
        L=3000, max_stay_counter=150)
    best_x, best_y = sa.run()
    print('best_x:', best_x[10, :], 'best_y', best_y)
    plt.plot(pd.DataFrame(sa.best_y_history).cummin(axis=0))
    plt.show()
```

1.3 第三问代码

```
    import numpy as np
from scipy.integrate import dblquad
from scipy import optimize
from copy import deepcopy
```

```
def get_relative_D(theta, s, a, c):
   theta = np.radians(theta)
   a_h = a * np.sin(theta)
   c_h = c * np.cos(theta)

   if theta == 0.0:
      return 0.0, a, lambda x: 0.0, lambda x, val=s / a: val
   if a_h >= c_h:
      s_c = c * c / np.tan(theta) / 2.0
      s_a = a * c - s_c
      if s <= s_c:

         x_max = np.sqrt(2.0 * s / np.tan(theta))
         gfun = lambda x: 0.0

         def hfun(x, x_max=x_max, tan_theta=np.tan(theta)):
            return (x_max - x) * tan_theta

         return 0.0, x_max, gfun, hfun
      elif s <= s_a:
         x_max = c / np.tan(theta) + (s - s_c) / c
         gfun = lambda x: 0.0

         def hfun(x, x_max=x_max, theta=theta, c=c, t=(s - s_c)
            / c):
            return c if x <= t else (x_max - x) * np.tan(theta)

         return 0.0, x_max, gfun, hfun
      else:
         x_max = a
         lft = np.sqrt(2 * (a * c - s) / np.tan(theta))
         gfun = lambda x: 0.0

         def hfun(x, x_ex=a - lft + c / np.tan(theta),
            theta=theta, c=c, t=a - lft):
            return c if x <= t else (x_ex - x) * np.tan(theta)

         return 0.0, x_max, gfun, hfun
   else:
      s_a = a * a * np.tan(theta) / 2.0
      s_c = a * c - s_a
      if s <= s_a:
         x_max = np.sqrt(2.0 * s / np.tan(theta))
         gfun = lambda x: 0.0

         def hfun(x, x_max=x_max, tan_theta=np.tan(theta)):
            return (x_max - x) * tan_theta

         return 0.0, x_max, gfun, hfun
      elif s <= s_c:
         x_max = a
         gfun = lambda x: 0.0
         c_u = a * np.tan(theta) + (s - s_a) / a
```

```
            def hfun(x, x_ex=c_u / np.tan(theta), theta=theta):
                return (x_ex - x) * np.tan(theta)

            return 0.0, x_max, gfun, hfun
        else:
            x_max = a
            lft = np.sqrt(2 * (a * c - s) / np.tan(theta))
            gfun = lambda x: 0.0

            def hfun(x, x_ex=a - lft + c / np.tan(theta),
                theta=theta, c=c, t=a - lft):
                return c if x <= t else (x_ex - x) * np.tan(theta)

            return 0.0, x_max, gfun, hfun


def get_relative_xz(theta, s, a, c):
    if theta == 0.0:
        return 0.0, s / a / 2.0 - c / 2.0
    relative_D = get_relative_D(np.abs(theta), s, a, c)
    x = dblquad(lambda z, x: x, *relative_D)[0] / s
    z = dblquad(lambda z, x: z, *relative_D)[0] / s
    if theta > 0:
        return x - a / 2.0, z - c / 2.0
    else:
        return a / 2.0 - x, z - c / 2.0


def get_real_xyz(theta, v, x, y, z, a, b, c):
    rt_x, rt_z = get_relative_xz(theta, v / b, a, c)
    return x + rt_x, y, z + rt_z


def centroid(env, tanks, bird_theta, used_fuel_v):
    theta = bird_theta['theta']
    x = env['bird_x'] * env['bird_m']
    y = env['bird_y'] * env['bird_m']
    z = env['bird_z'] * env['bird_m']
    m_sum = env['bird_m']
    for tank in tanks:
        left = tank['first'] - used_fuel_v[str(int(tank['id']))]
        m = left * env['fuel_density']
        x_t, y_t, z_t = get_real_xyz(theta,
                                     left,
                                     tank['x'],
                                     tank['y'],
                                     tank['z'],
                                     tank['a'],
                                     tank['b'],
                                     tank['c'])
        x += x_t * m
        y += y_t * m
        z += z_t * m
```

```
            m_sum += m
        return x / m_sum, y / m_sum, z / m_sum


    def txt_to_table(path):
        table = []
        with open(path, 'r') as fp:
            attr = fp.readline().split();
            for line in fp.readlines():
                table.append(dict(zip(attr, map(float, line.split()))))
        return table


    def find_best_change(env, tanks, bird_theta, used_fuel,
        use_tanks, target, min_fuel):
        def distance(change):
            for i, use_tank in enumerate(use_tanks):
                used_fuel[use_tank] += change[i]
            x, y, z = centroid(env, tanks, bird_theta, used_fuel)
            for i, use_tank in enumerate(use_tanks):
                used_fuel[use_tank] -= change[i]
            return sum(((x - target[0]) ** 2, (y - target[1]) ** 2, (z
                - target[2]) ** 2))

        tanks_bounds = [(1e-15, max(1e-15,min(tanks[int(use_tank) -
            1]['first'] - used_fuel[use_tank],
                           tanks[int(use_tank) - 1]['upper']-1e-7)))
                              for use_tank in
                      use_tanks]

        cons = (
            {'type': 'eq', 'fun': lambda x, min_fuel=min_fuel:
                np.sum(x) - min_fuel}
            # {'type': 'ineq', 'fun': lambda x, min_fuel=min_fuel:
                min_fuel - np.sum(x) + 0.0001}
        )

        distance_min = optimize.minimize(distance,
            np.zeros(len(use_tanks)), bounds=tanks_bounds,
            constraints=cons)

        # print(distance_min)
        if distance_min.success:
            return distance_min.fun, tuple(distance_min.x)
        else:
            return -1, use_tanks


    use_tanks_list = [
        ('1', '3'),
        ('1', '4'),
        ('1', '5'),
        ('2', '3'),
        ('2', '4'),
```

```
    ('2', '5'),
    ('2', '6'),
    ('3', '4'),
    ('3', '5'),
    ('3', '6'),
    ('4', '5'),
    ('4', '6')
]


def find_best_tank_first(env, tanks, bird_theta, use_tanks,
    target, sum_fuel_t0):
    global L_all_tanks

    def distance(tank_firsts):
        used_fuel = dict()
        for i, use_tank in enumerate(use_tanks):
            used_fuel[use_tank] = -tank_firsts[i]
        x, y, z = centroid(env, tanks, bird_theta, used_fuel)
        return sum(((x - target[0]) ** 2, (y - target[1]) ** 2, (z
            - target[2]) ** 2))

    tanks_bounds = [(0, tanks[int(use_tank) - 1]['a'] *
        tanks[int(use_tank) - 1]['b'] * tanks[int(use_tank) -
        1]['c'])
                for use_tank in
                use_tanks]

    init_tanks = [
        sum_fuel_t0 * tanks[int(use_tank) - 1]['a'] *
            tanks[int(use_tank) - 1]['b'] * tanks[int(use_tank) -
            1][
            'c'] / L_all_tanks
        for use_tank in
        use_tanks]

    cons = (
        {'type': 'eq', 'fun': lambda x, sum_fuel_t0=sum_fuel_t0:
            np.sum(x) - sum_fuel_t0}
        # {'type': 'ineq', 'fun': lambda x, min_fuel=min_fuel:
            min_fuel - np.sum(x) + 0.0001}
    )

    distance_min = optimize.minimize(distance, init_tanks,
        bounds=tanks_bounds, constraints=cons)

    # print(distance_min)
    if distance_min.success:
        return distance_min.fun, tuple(distance_min.x)
    else:
        return -1, use_tanks


if __name__ == "__main__":
```

```
env = txt_to_table('env.txt')[0]
tanks = txt_to_table('tank.txt')
bird_theta = {'theta': 0.0}
min_fuels = txt_to_table('min_fuel.txt')
for tank in tanks:
   tank['upper'] /= env['fuel_density']
for min_fuel in min_fuels:
   min_fuel['min_fuel'] /= env['fuel_density']
image_xyzs = txt_to_table('image_xyz.txt')

sum_min_fuels = np.sum([min_fuel['min_fuel'] for min_fuel in
   min_fuels])
first_all_tanks = np.sum([tank['first'] for tank in tanks])
L_all_tanks = np.sum(
   [tanks[int(use_tank) - 1]['a'] * tanks[int(use_tank) -
      1]['b'] * tanks[int(use_tank) - 1]['c'] for use_tank in
    '123456'])
print(L_all_tanks - sum_min_fuels)
AA = (first_all_tanks - sum_min_fuels) / 7200.0

first_dis, tank_fisrts = find_best_tank_first(env, tanks,
   bird_theta, list('123456'),
                                    [image_xyzs[0][c] for c in
                                       'xyz'], sum_min_fuels +
                                       1.2 + 1e-4)
if first_dis == -1:
   print('error_sum_fuel')
   exit()
print(tank_fisrts)

with open('tank_first.txt', 'w') as fp:
   fp.write('1\t2\t3\t4\t5\t6\n')
   fp.write('%.20f\t%.20f\t%.20f\t%.20f\t%.20f\t%.20f\n' %
      tuple(tank_fisrts))

for i, tank in enumerate(tanks):
   tank['first'] = tank_fisrts[i]

used_fuel_v = {'1': 0, '2': 0, '3': 0, '4': 0, '5': 0, '6': 0}
ans_use_fuel_per_time = []
ans_xyz_per_time = []
ans_dis_per_time = []
ans_max = first_dis
for k in range(120):
   kt = k * 60
   now_max = 9999999999
   fuel_v_wanna_use = None
   fuel_v_wanna_use_per_time_t = None
   xyz_wanna_use_per_time_t = None
   dis_wanna_use_per_time_t = None
   for use_tanks in use_tanks_list:
      t_used_fuel_per_time = []
      t_xyz_per_time = []
      t_dis_per_time = []
```

```
        t_used_fuel_v = deepcopy(used_fuel_v)
        t_max = 0
        for i in range(60):
            time = kt + i
            min_fuel = min_fuels[time]['min_fuel']
            uppers = np.sum([tanks[int(use_tank) - 1]['upper']
                for use_tank in use_tanks])

            lefts = np.sum([tanks[int(use_tank) - 1]['first'] -
                t_used_fuel_v[use_tank] for use_tank in
                use_tanks])

            if min_fuel > uppers or min_fuel > lefts:
                t_max = -1
                break

            dis, change = find_best_change(env, tanks,
                bird_theta, t_used_fuel_v, use_tanks,
                                    [image_xyzs[time][c] for c in
                                        'xyz'], min_fuel + 1e-8)

            t_dict = {'1': 0.0, '2': 0.0, '3': 0.0, '4': 0.0,
                '5': 0.0, '6': 0.0}
            if dis == -1:
                t_max = -1
                break
            for i, tank in enumerate(use_tanks):
                if tank == '1':
                    t_dict['2'] += change[i] * env['fuel_density']
                if tank == '6':
                    t_dict['5'] += change[i] * env['fuel_density']
                t_dict[tank] += change[i] * env['fuel_density']
            x, y, z = centroid(env, tanks, bird_theta,
                t_used_fuel_v)
            t_used_fuel_per_time.append(t_dict)
            t_dis_per_time.append(np.sqrt(dis))
            t_xyz_per_time.append({'x': x, 'y': y, 'z': z})
            t_max = max(t_max, dis)
            for i, use_tank in enumerate(use_tanks):
                t_used_fuel_v[use_tank] += change[i]
        if t_max != -1 and t_max < now_max:
            now_max = t_max
            fuel_v_wanna_use = t_used_fuel_v
            fuel_v_wanna_use_per_time_t = t_used_fuel_per_time
            xyz_wanna_use_per_time_t = t_xyz_per_time
            dis_wanna_use_per_time_t = t_dis_per_time
    if fuel_v_wanna_use:
        used_fuel_v = fuel_v_wanna_use
        ans_use_fuel_per_time.extend(fuel_v_wanna_use_per_time_t)
        ans_xyz_per_time.extend(xyz_wanna_use_per_time_t)
        ans_dis_per_time.extend(dis_wanna_use_per_time_t)
    else:
        print("no solution")
        exit(0)
```

```
        ans_max = max(ans_max, now_max)
        print(k, np.sqrt(ans_max))

    with open('use_fuel_per_time.txt', 'w') as fp:
        fp.write('ans_max:%.20f\n' % (np.sqrt(ans_max)))
        fp.write('time\t1\t2\t3\t4\t5\t6\n')
        for i, per_time in enumerate(ans_use_fuel_per_time):
            fp.write('%d\t%.20f\t%.20f\t%.20f\t%.20f\t%.20f\t%.20f\n'
                % (
                i + 1, per_time['1'], per_time['2'], per_time['3'],
                    per_time['4'], per_time['5'], per_time['6']))
    with open('dis_per_time.txt', 'w') as fp:
        fp.write('time\tdis\n')
        for i, per_time in enumerate(ans_dis_per_time):
            fp.write('%d\t%.20f\n' % (i+1,per_time))
    with open('xyz_per_time.txt', 'w') as fp:
        fp.write('time\tx\ty\tz\n')
        for i, per_time in enumerate(ans_xyz_per_time):
            fp.write('%d\t%.20f\t%.20f\t%.20f\n' %
                (i+1,per_time['x'],per_time['y'],per_time['z']))
```

1.4 第四问代码

```
    import numpy as np
from scipy.integrate import dblquad
from scipy import optimize
from copy import deepcopy


def get_relative_D(theta, s, a, c):
    if s < 0.0:
        return 0, 0, 0.0, 0.0, 0.0
    theta = np.radians(theta)
    a_h = a * np.sin(theta)
    c_h = c * np.cos(theta)

    if theta == 0.0:
        return 0.0, a, lambda x: 0.0, lambda x, val=s / a: val
    if a_h >= c_h:
        s_c = c * c / np.tan(theta) / 2.0
        s_a = a * c - s_c
        if s <= s_c:

            x_max = np.sqrt(2.0 * s / np.tan(theta))
            gfun = lambda x: 0.0

            def hfun(x, x_max=x_max, tan_theta=np.tan(theta)):
                return (x_max - x) * tan_theta

            return 0.0, x_max, gfun, hfun
        elif s <= s_a:
            x_max = c / np.tan(theta) + (s - s_c) / c
            gfun = lambda x: 0.0
```

```
            def hfun(x, x_max=x_max, theta=theta, c=c, t=(s - s_c)
                / c):
                return c if x <= t else (x_max - x) * np.tan(theta)

            return 0.0, x_max, gfun, hfun
        else:
            x_max = a
            lft = np.sqrt(2 * (a * c - s) / np.tan(theta))
            gfun = lambda x: 0.0

            def hfun(x, x_ex=a - lft + c / np.tan(theta),
                theta=theta, c=c, t=a - lft):
                return c if x <= t else (x_ex - x) * np.tan(theta)

            return 0.0, x_max, gfun, hfun
    else:
        s_a = a * a * np.tan(theta) / 2.0
        s_c = a * c - s_a
        if s <= s_a:
            x_max = np.sqrt(2.0 * s / np.tan(theta))
            gfun = lambda x: 0.0

            def hfun(x, x_max=x_max, tan_theta=np.tan(theta)):
                return (x_max - x) * tan_theta

            return 0.0, x_max, gfun, hfun
        elif s <= s_c:
            x_max = a
            gfun = lambda x: 0.0
            c_u = a * np.tan(theta) + (s - s_a) / a

            def hfun(x, x_ex=c_u / np.tan(theta), theta=theta):
                return (x_ex - x) * np.tan(theta)

            return 0.0, x_max, gfun, hfun
        else:
            x_max = a
            lft = np.sqrt(2 * (a * c - s) / np.tan(theta))
            gfun = lambda x: 0.0

            def hfun(x, x_ex=a - lft + c / np.tan(theta),
                theta=theta, c=c, t=a - lft):
                return c if x <= t else (x_ex - x) * np.tan(theta)

            return 0.0, x_max, gfun, hfun


def get_relative_xz(theta, s, a, c):
    if s <= 0.0:
        return 0.0, 0.0
    if theta == 0.0:
        return 0.0, s / a / 2.0 - c / 2.0
    relative_D = get_relative_D(np.abs(theta), s, a, c)
```

```
    x = dblquad(lambda z, x: x, *relative_D)[0] / s
    z = dblquad(lambda z, x: z, *relative_D)[0] / s
    if theta > 0:
        return x - a / 2.0, z - c / 2.0
    else:
        return a / 2.0 - x, z - c / 2.0


def get_real_xyz(theta, v, x, y, z, a, b, c):
    rt_x, rt_z = get_relative_xz(theta, v / b, a, c)
    return x + rt_x, y, z + rt_z


def centroid(env, tanks, bird_theta, used_fuel_v):
    theta = bird_theta['theta']
    x = env['bird_x'] * env['bird_m']
    y = env['bird_y'] * env['bird_m']
    z = env['bird_z'] * env['bird_m']
    m_sum = env['bird_m']
    for tank in tanks:
        left = tank['first'] - used_fuel_v[str(int(tank['id']))]
        m = left * env['fuel_density']
        x_t, y_t, z_t = get_real_xyz(theta,
                                     left,
                                     tank['x'],
                                     tank['y'],
                                     tank['z'],
                                     tank['a'],
                                     tank['b'],
                                     tank['c'])
        x += x_t * m
        y += y_t * m
        z += z_t * m
        m_sum += m
    return x / m_sum, y / m_sum, z / m_sum


def txt_to_table(path):
    table = []
    with open(path, 'r') as fp:
        attr = fp.readline().split();
        for line in fp.readlines():
            table.append(dict(zip(attr, map(float, line.split()))))
    return table


def find_best_change(env, tanks, bird_theta, used_fuel,
    use_tanks, target, min_fuel):
    def distance(change):
        for i, use_tank in enumerate(use_tanks):
            used_fuel[use_tank] += change[i]
        x, y, z = centroid(env, tanks, bird_theta, used_fuel)
        for i, use_tank in enumerate(use_tanks):
            used_fuel[use_tank] -= change[i]
```

```
        return sum(((x - target[0]) ** 2, (y - target[1]) ** 2, (z
            - target[2]) ** 2))

    tanks_bounds = [(1e-15, max(1e-15, min(tanks[int(use_tank) -
        1]['first'] - used_fuel[use_tank],
                               tanks[int(use_tank) -
                                  1]['upper'])-1e-10)) for use_tank
                                  in
                use_tanks]

    cons = (
        {'type': 'ineq', 'fun': lambda x, min_fuel=min_fuel:
            np.sum(x) - min_fuel}
        # {'type': 'ineq', 'fun': lambda x, min_fuel=min_fuel:
            min_fuel - np.sum(x) + 0.0001}
    )

    distance_min = optimize.minimize(distance,
        np.zeros(len(use_tanks)), bounds=tanks_bounds,
        constraints=cons)

    # print(distance_min)
    if distance_min.success:
        return distance_min.fun, tuple(distance_min.x)
    else:
        return -1, use_tanks


use_tanks_list = [
    ('1', '3'),
    ('1', '4'),
    ('1', '5'),
    ('2', '3'),
    ('2', '4'),
    ('2', '5'),
    ('2', '6'),
    ('3', '4'),
    ('3', '5'),
    ('3', '6'),
    ('4', '5'),
    ('4', '6')
]

if __name__ == "__main__":
    env = txt_to_table('env.txt')[0]
    tanks = txt_to_table('tank.txt')
    bird_thetas = txt_to_table('bird_theta.txt')
    # bird_theta = {'theta':0.0}
    min_fuels = txt_to_table('min_fuel.txt')
    for tank in tanks:
        tank['upper'] /= env['fuel_density']
    for min_fuel in min_fuels:
        min_fuel['min_fuel'] /= env['fuel_density']
    image_xyzs = txt_to_table('image_xyz.txt')
```

```
for image_xyz in image_xyzs:
    image_xyz['x'] = 0.0
    image_xyz['y'] = 0.0
    image_xyz['z'] = 0.0
ans_use_fuel_vs = []
used_fuel_v = {'1': 0, '2': 0, '3': 0, '4': 0, '5': 0, '6': 0}
ans_use_fuel_per_time = []
ans_max = 0
for k in range(120):
    kt = k * 60
    now_max = 9999999999
    fuel_v_wanna_use = None
    fuel_v_wanna_use_per_time_t = None
    for use_tanks in use_tanks_list:
        t_used_fuel_per_time = []
        t_used_fuel_v = deepcopy(used_fuel_v)
        t_max = 0
        for i in range(60):
            time = kt + i
            dis, change = find_best_change(env, tanks,
                bird_thetas[time], t_used_fuel_v, use_tanks,
                                    [image_xyzs[time][c] for c in
                                        'xyz'],
                                    min_fuels[time]['min_fuel']
                                    + 1e-8)
            t_dict = {'1': 0.0, '2': 0.0, '3': 0.0, '4': 0.0,
                '5': 0.0, '6': 0.0}
            if dis == -1:
                t_max = -1
                break
            for i, tank in enumerate(use_tanks):
                if tank == '1':
                    t_dict['2'] += change[i] * env['fuel_density']
                if tank == '6':
                    t_dict['5'] += change[i] * env['fuel_density']
                t_dict[tank] += change[i] * env['fuel_density']
            t_used_fuel_per_time.append(t_dict)
            t_max = max(t_max, dis)
            for i, use_tank in enumerate(use_tanks):
                t_used_fuel_v[use_tank] += change[i]
        if t_max != -1 and t_max < now_max:
            now_max = t_max
            fuel_v_wanna_use = t_used_fuel_v
            fuel_v_wanna_use_per_time_t = t_used_fuel_per_time
    if fuel_v_wanna_use:
        used_fuel_v = fuel_v_wanna_use
        ans_use_fuel_per_time.extend(fuel_v_wanna_use_per_time_t)
    else:
        print("no solution")
        exit(0)
    ans_max = max(ans_max, now_max)
    print(k, np.sqrt(ans_max))
with open('use_fuel_per_time.txt', 'w') as fp:
    fp.write('ans_max:%.20f\n' % (np.sqrt(ans_max)))
```

```
        fp.write('time\t1\t2\t3\t4\t5\t6\n')
        for i, per_time in enumerate(ans_use_fuel_per_time):
            fp.write('%d\t%.20f\t%.20f\t%.20f\t%.20f\t%.20f\t%.20f\n'
                % (
                i + 1, per_time['1'], per_time['2'], per_time['3'],
                    per_time['4'], per_time['5'], per_time['6']))
```