

航天飞行动力学

第三次作业 —— 飞行方案设计

一、题目

1. 导弹参数：

- * 导弹质量 $m_0 = 320kg$
- * 发动机推力 $P = 2000N$
- * 初始速度 $V_0 = 250m/s$
- * 初始位置 $x_0 = 0m$
- * 初始高度 $H_0 = 7000m$
- * 初始弹道倾角 $\theta = 0^\circ$
- * 初始俯仰角 $\varphi_0 = 0^\circ$
- * 初始攻角 $\alpha_0 = 0^\circ$
- * 初始俯仰角速度 $\dot{\varphi}_0 = 0rad/s$
- * 初始速度 $V_0 = 250m/s$
- * 参考长度 $S_{ref} = 0.45m^2$
- * 参考面积 $L_{ref} = 2.5m$
- * 升力系数 $C_y = 0.25\alpha + 0.05\delta_z$
- * 阻力系数 $C_x = 0.2 + 0.005\alpha^2$
- * 俯仰力矩系数 $m_z = -0.1\alpha + 0.024\delta_z$

2. 大气密度计算公式：

$$\begin{cases} \rho_0 = 1.2495 \text{ kg/m}^3 \\ T_0 = 288.15K \\ T = T_0 - 0.0065H \\ \rho = \rho_0 \left(\frac{T}{T_0} \right)^{4.25588} \end{cases} \quad (1)$$

3. 飞行方案：

(1) 当 $x < 9100m$ 时，采用瞬时平衡假设

$$\begin{cases} H^* = 2000 \times \cos(0.000314 \times 1.1 \times x) + 5000 \\ \delta_z = k_\varphi \times (H - H^*) + \dot{k}_\varphi \times (H - H^*) \\ \delta_z = k_\varphi(H - H^*) + \dot{k}_\varphi H \\ m_s = 0.0kg/s \end{cases} \quad (2)$$

(2) 当 $24000m > x > 9100m$ 时, 等高飞行方案, 采用瞬时平衡假设。

$$\begin{cases} H^* = 3050m \\ \delta_z = k_\varphi(H - H^*) + \dot{k}_\varphi H \\ \delta_z = k_\varphi(H - H^*) + \dot{k}_\varphi H \\ m_s = 0.46kg/s \end{cases} \quad (3)$$

(3) 当 $x > 24000m$ 且 $y > 0$, 目标位置为 $x_m = 30000m$, 采用比例导引法和瞬时平衡假设

$$\begin{cases} x_m = 30000m \\ m_z^\alpha \alpha + m_z^{\delta_z} \delta_z = 0 \\ m_s = 0.0kg/s \end{cases} \quad (4)$$

注: 舵偏角约束 $|\delta_z| \leq 30^\circ$

二、公式推导

1. $x < 24000m$ 的飞行方案:

基于“瞬时平衡”假设, 将包含 20 个方程的导弹运动方程组简化为铅垂平面内的质心运动方程组。

$$\begin{cases} m \frac{dV}{dt} = P \cos \alpha - X - mg \sin \theta \\ mV \frac{d\theta}{dt} = P \sin \alpha + Y - mg \cos \theta \\ \frac{dx}{dt} = V \cos \theta \\ \frac{dy}{dt} = V \sin \theta \\ \frac{dm}{dt} = -m_s \\ \alpha_b = -\frac{m_z^{\delta_z}}{m_z^\alpha} \delta_{zb} \\ \delta_z = k_\varphi(H - H^*) + \dot{k}_\varphi (\dot{H} - \dot{H}^*) \\ H^* = 2000 \times \cos(0.000314 \times 1.1 \times x) + 5000 \end{cases} \quad (5)$$

代入各物理量定义式：

$$\left\{ \begin{array}{l} \frac{dV}{dt} = \frac{P \cos \alpha - X}{m} - g \sin \theta \\ \frac{d\theta}{dt} = \frac{P \sin \alpha + Y}{m} - \frac{g \cos \theta}{V} \\ \frac{dx}{dt} = V \cos \theta \\ \frac{dy}{dt} = V \sin \theta \\ \frac{dm}{dt} = -m_s \\ \alpha_b = -\frac{m_z^{\delta_z}}{m_z^\alpha} \delta_{zb} \\ \delta_z = k_\varphi (H - H^*) + \dot{k}_\varphi (\dot{H} - \dot{H}^*) \\ H^* = 2000 \times \cos(0.000314 \times 1.1 \times x) + 5000 \\ Y = (0.25\alpha + 0.05\delta_z) \times \frac{1}{2}\rho V^2 \times S_{ref} \\ X = (0.2 + 0.005\alpha^2) \times \frac{1}{2}\rho V^2 \times S_{ref} \end{array} \right. \quad (6)$$

2. $x > 24000m$ 的飞行方案：

(1) 末段第一种计算方法：

$$\left\{ \begin{array}{l} r \frac{dq}{dt} = V_m \times \sin \eta - V_T \sin \eta_T \\ \tan q = \frac{y_T - y_m}{x_T - x_m} \\ \frac{d\theta^*}{dt} = k \frac{dq}{dt} \\ \theta^* - \theta_0 = k(q - q_0) \\ \theta_0, q_0? \\ \delta_z = k_\theta(\theta - \theta^*) + k_{\dot{\theta}}(\dot{\theta} - \dot{\theta}^*) \end{array} \right. \quad (7)$$

(2) 末段第二种计算方法：

只需要给出比例导引系数根据运动学方程

$$\left\{ \begin{array}{l} r \frac{dq}{dt} = V_m \times \sin \eta - V_T \sin \eta_T \\ \tan q = \frac{y_T - y_m}{x_T - x_m} \\ \frac{dq}{dt} = \frac{-V_m \sin(\theta - q)}{r} \end{array} \right. \quad (8)$$

由比例导引法 $\dot{\theta}^* = k\dot{q}$, 可得动力学方程第二式

$$mV_m \dot{\theta}^* = P \sin \alpha + Y - mg \cos \theta \Rightarrow mV_m k \dot{q} = P \sin \alpha + Y - mg \cos \theta \quad (9)$$

由于攻角较小, 进行线性化可得

$$mV_m k \dot{q} = P\alpha + Y^\alpha \alpha + Y^{\delta_z} \delta_z - mg \cos \theta \quad (10)$$

由于瞬时平衡 $m_z = 0$, 可得

$$-0.1\alpha + 0.024\delta_z = 0 \Rightarrow \delta_z = 0.1\alpha/0.024 \quad (11)$$

代入, 可得

$$\alpha = \frac{mV_mk\dot{q} + mg \cos \theta}{P + Y^\alpha + Y^{\delta_z}(0.1/0.024)} \Rightarrow \frac{mV_mk\dot{q} + mg \cos \theta}{P + C_y^\alpha q S_{ref} + C_y^{\delta_z} q S_{ref}(0.1/0.024)} \quad (12)$$

最后得到弹道方程为

$$\left\{ \begin{array}{l} \frac{dV}{dt} = \frac{P \cos \alpha - X}{m} - g \sin \theta \\ \alpha = \frac{mV_k\dot{q} + mg \cos \theta}{P + C_y^\alpha q S_{ref} + C_y^{\delta_z} q S_{ref}(0.1/0.024)} \\ \frac{dx}{dt} = V \cos \theta \\ \frac{dy}{dt} = V \sin \theta \\ \dot{\theta}^* = k\dot{q} \\ \dot{\theta}^* = \dot{\theta} \\ \tan q = \frac{y_T - y_m}{x_T - x_m} \\ \frac{dq}{dt} = \frac{-V \sin(\theta - q)}{r} \\ \delta_z = 0.1\alpha/0.024 \end{array} \right. \quad (13)$$

补充约束条件

$$\left\{ \begin{array}{l} \frac{dV}{dt} = \frac{P \cos \alpha - X}{m} - g \sin \theta \\ \frac{d\theta}{dt} = \frac{-kV \sin(\theta - \arctan \frac{y_T - y_m}{x_T - x_m})}{r} \\ \frac{dx}{dt} = V \cos \theta \\ \frac{dy}{dt} = V \sin \theta \\ \frac{dm}{dt} = -m_s \\ \alpha = \frac{mV\dot{\theta} + mg \cos \theta}{P + C_y^\alpha q S_{ref} + C_y^{\delta_z} q S_{ref}(0.1/0.024)} \\ \alpha = -\frac{m_z^{\delta_z}}{m_z^\alpha} \delta_z \\ |\delta_z| \leq 30^\circ \end{array} \right. \quad (14)$$

三、仿真结果

三个系数的取值:

$$K_\varphi = -0.5$$

$$\dot{K}_\varphi = 0.6 * K_p hi$$

$$K_3 = 5$$

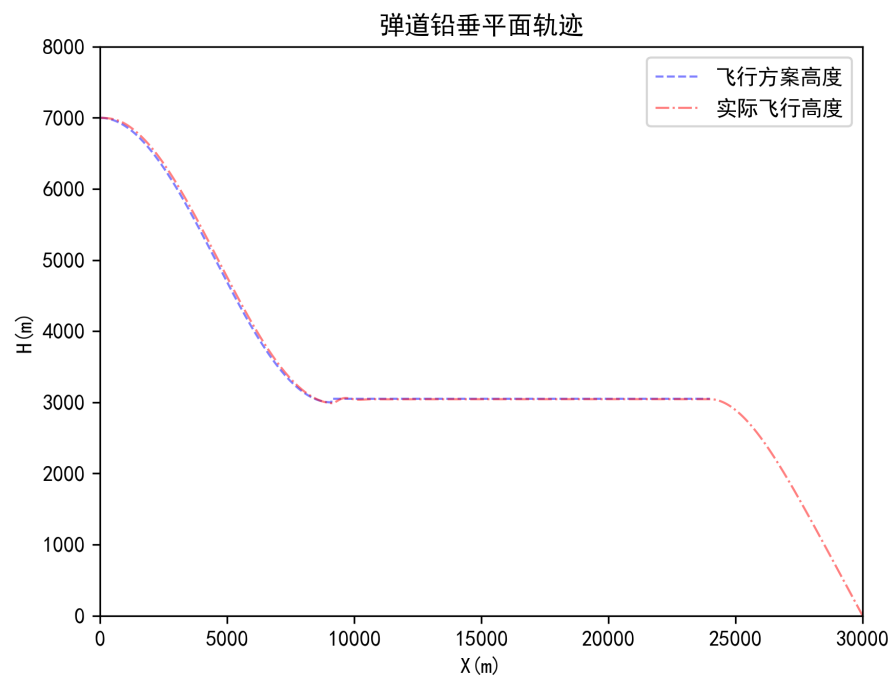


图 1 导弹飞行轨迹图

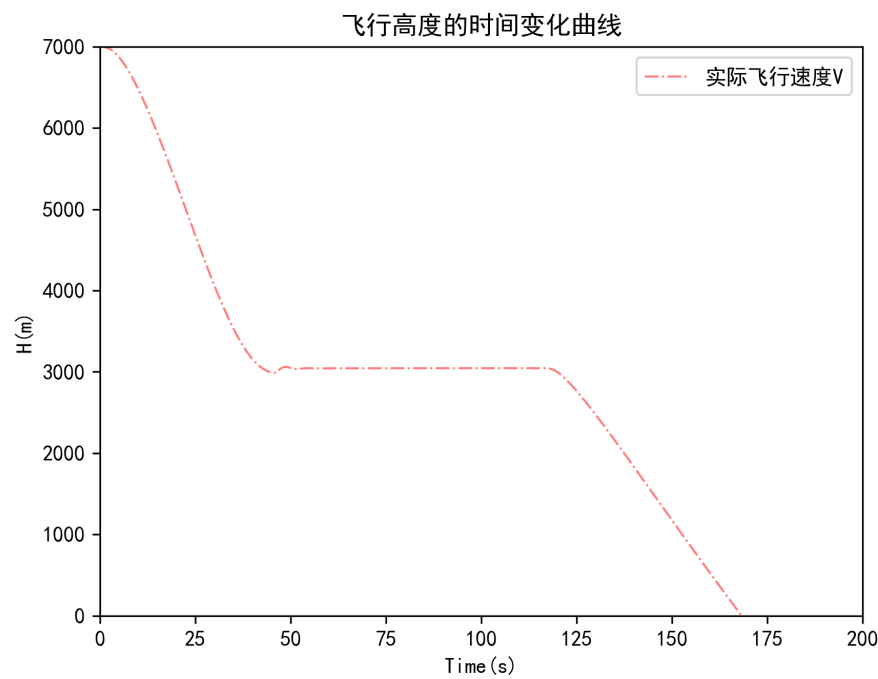


图 2 导弹飞行高度的时间曲线图

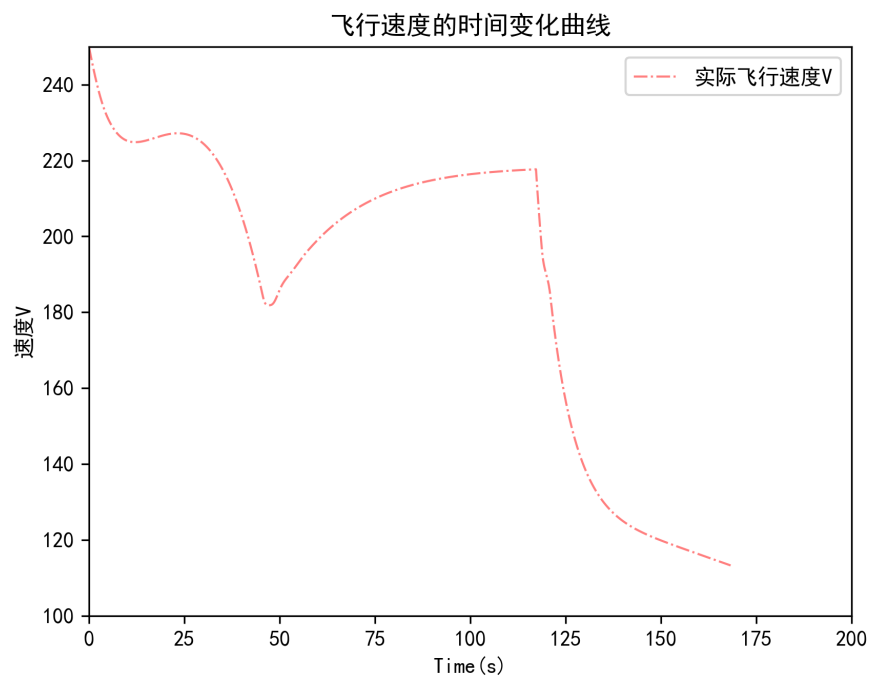


图 3 导弹飞行速度的时间曲线图

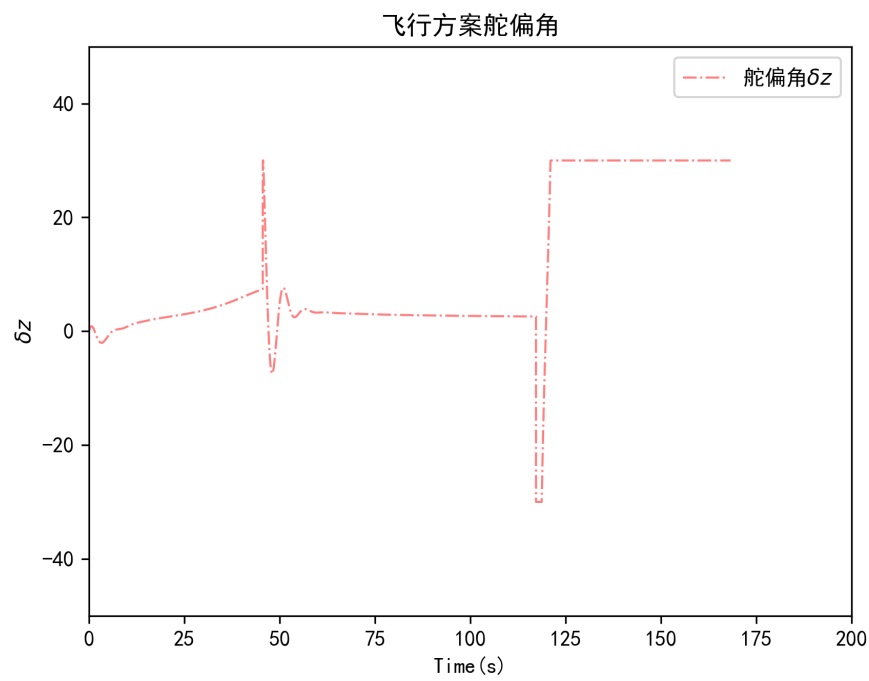


图 4 导弹飞行舵偏角的时间曲线图

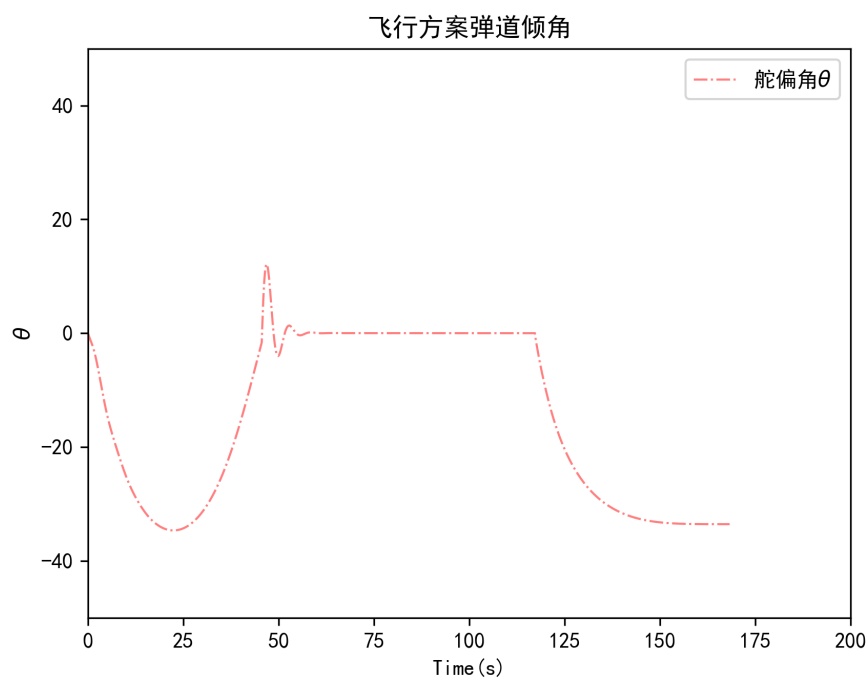
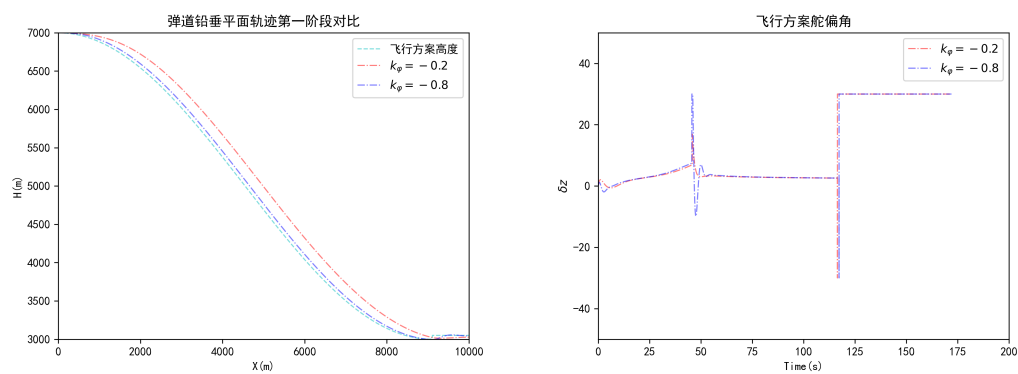


图 5 导弹飞行弹道倾角的时间曲线图

四、结果分析

1. k_φ 的影响:

图 6 K_φ 大小对导弹飞行弹道和舵偏角的影响

如图 6 所示, K_φ 是理想控制方程的放大系数, K_φ 绝对值越大, 导弹越快地恢复到预定的飞行方案。

2. \dot{K}_φ 的影响:

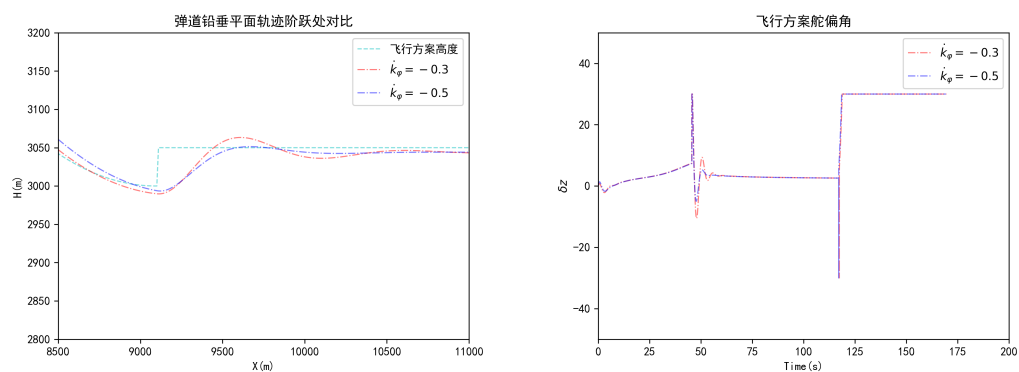


图 7 K_φ 大小对导弹飞行弹道和舵偏角的影响

如图 7 所示, K_φ 是用于减小超调量的放大系数, 起到阻尼作用, k_φ 绝对值越大, 导弹越快地稳定到预定的飞行方案。

3. k_3 的影响:

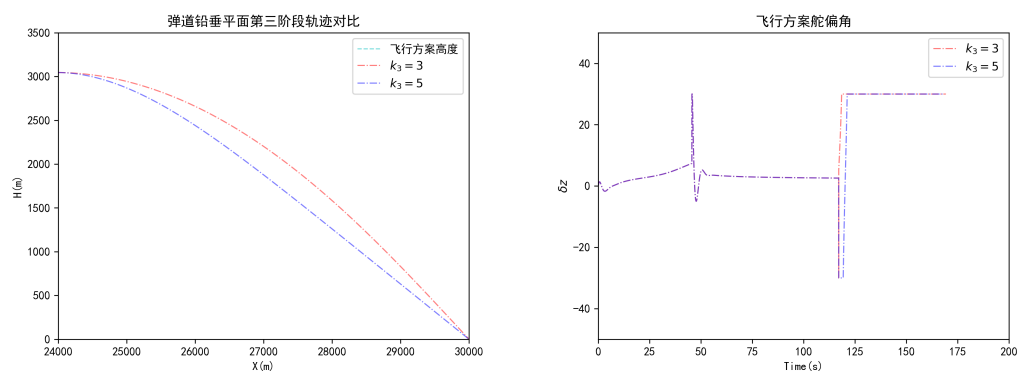


图 8 k_3 大小对导弹飞行弹道和舵偏角的影响

如图 8 所示, K_3 是比例导引法比例系数, K_3 越大, 后期弹道约平直。

注: 第三阶段的舵偏角由于使用显式的 Euler 法计算, 结果有一定的发散。

源代码 1: main.py

```
1  """
2      一级半运载火箭弹道计算
3  """
4
5
6  ##### 环境准备 #####
7  import numpy as np
8  from matplotlib import pyplot as plt
9  from scipy.interpolate import interp2d, interp1d, CloughTocher2DInterpolator
10
11  # 展示高清图
12  from matplotlib_inline import backend_inline
13  backend_inline.set_matplotlib_formats('svg')
14
15  plt.rcParams['font.sans-serif'] = ['SimHei']
16  plt.rcParams['axes.unicode_minus'] = False
17
18  # 忽略提示
19  import warnings
20  warnings.filterwarnings("ignore", category=DeprecationWarning)
21
22
23  ##### 定义数值 #####
24  # 导弹参数
25  S_ref = 19.6
26
27  ## 质量
28  # 构型：芯一级+2个大助推器+2个小助推器
29  m_0 = 645000 # 总质量
30  m0 = 6600 # 有效载荷质量
31
32  # 芯一级
33  m1_0 = 173000 # 芯一级起飞质量
34  m1_1 = 19500 # 芯一级结构质量
35  qm_1 = 334 # 芯一级秒流量
36
37  # 大助推器
38  m2_0 = 158000
39  m2_1 = 14500
40  qm_2 = 820
41
```

```
42 # 小助推器(单个助推器):
43 m3_0 = 72700
44 m3_1 = 7100
45 qm_3 = 410
46
47 # 整流罩
48 m4 = 4000
49
50 # 经纬度, 单位度
51 latitude_0 = 19.6 # 维度
52 longitude_0 = 111 # 经度
53
54 # 攻角参数
55 alpha_max = 0.65 #单位弧度
56 alpha_costant = 0.1 # 转弯段参数
57
58 # 时间
59 t1 = 15 # 垂直起飞段结束, 进入转弯段
60 t2 = 120 # 120s整流罩分离。
61 t3 = 160 # 160秒两小助推关机分离, 大助推外侧关机(推力减半, 秒流量减半)
62 t4 = 190 # 190秒两大助推内侧关机分离
63 t5 = 459.58 # 芯一级关机
64
65 # 目标轨道
66 Target_orbital_altitude = 500e3
67 Target_orbital_inclination = 60 #单位 度
68 #轨道高度偏差1km, 轨道倾角偏差0.5°
69
70 # 放大系数
71 K_phi = 1
72 K_alpha = 1
73
74 # 仿真时间步
75 timestep = 0.01
76 ##### 类的定义 #####
77 # 导弹状态定义
78 class statu():
79     __slot__=['Time', 'X', 'H', 'V', 'theta', 'mass', 'alpha', 'deltaz']
80
81     # 位置
82     # 速度
83     # 欧拉角
84     # 角加速度
```

```
85     # 舵偏角
86
87     # 初始化
88     def __init__(self, Time, X=0, H=0, V=0, theta=0, mass=0):
89         self.Time = Time
90         self.X = X
91         self.H = H
92         self.V = V
93         self.theta = theta
94         self.mass = mass
95         self.alpha = 0
96         self.deltaz = 0
97         self.dq = 0
98
99     # 显式Euler法, 给定飞行高度
100    def Euler(self, before, dmass):
101        self.Time = before.Time + timestep
102
103        self.X = before.X + before.V * np.cos(before.theta) * timestep
104        self.H = before.H + before.V * np.sin(before.theta) * timestep
105
106        self.deltaz = K_phi * (self.H - High_goal(self.X)) + K_phi_dot * (before.V * np
            .sin(before.theta) - High_goal_dot(self.X))
107
108        if self.deltaz > 30:
109            self.deltaz = 30
110        elif self.deltaz < -30:
111            self.deltaz = -30
112
113        self.alpha = 0.24 * self.deltaz
114
115        Y = (0.25 * self.alpha + 0.05 * self.deltaz) * 0.5 * air(self.H) * before.V *
            before.V * S_ref
116
117        X = (0.005 * self.alpha * self.alpha + 0.2) * 0.5 * air(self.H) * before.V *
            before.V * S_ref
118
119        self.mass = before.mass - dmass * timestep
120        if dmass == 0:
121            P = 0
122        else:
123            P = 2000
124
```

```

125     self.V = before.V + (P*np.cos(before.alpha*3.14159625/180) - X - self.mass
126         *9.8*np.sin(before.theta)) /self.mass*timestep
127
128     self.theta = before.theta + (P*np.sin(self.alpha*3.14159625/180) + Y - self.
129         mass*9.8*np.cos(before.theta)) /self.mass/self.V*timestep
130
131
132     # 比例导引法, 给定目标位置
133     def Euler2(self, before, Xm, Ym):
134         self.Time = before.Time + timestep
135
136         self.X = before.X + before.V * np.cos(before.theta) * timestep
137         self.H = before.H + before.V * np.sin(before.theta) * timestep
138         self.mass = before.mass
139
140         self.r = np.sqrt((self.X - Xm)*(self.X - Xm) + (self.H - Ym)*(self.H - Ym))
141
142         self.dq = - before.V * np.sin(before.theta - np.arctan(( self.H - Ym)/(self.X
143             - Xm)))/ self.r
144
145         self.theta = before.theta + K_q * self.dq * timestep
146
147         P = 0
148
149         self.alpha = (self.mass* before.V * K_q * self.dq + self.mass * 9.8 * np.cos(
150             self.theta))/(P + (0.25 + 0.05/0.24) * 0.5 * air(self.H) * before.V *
151             before.V * S_ref) /3.14159*180
152
153         self.deltaz = self.alpha / 0.24
154
155         if self.deltaz > 30:
156             self.deltaz = 30
157         if self.deltaz < -30:
158             self.deltaz = -30
159
160         self.alpha = 0.24 *self.deltaz
161
162         X = (0.005 * self.alpha * self.alpha + 0.2) * 0.5 * air(self.H) * before.V *
163             before.V * S_ref
164         self.V = before.V + (P*np.cos(before.alpha*3.14159625/180) - X - self.mass
165             *9.8*np.sin(before.theta)) /self.mass*timestep
166
167     def Euler3(self, before, Xm, Ym):
168         self.Time = before.Time + timestep

```

```

161
162     self.X = before.X + before.V * np.cos(before.theta) * timestep
163     self.H = before.H + before.V * np.sin(before.theta) * timestep
164     self.mass = before.mass
165
166     self.r = np.sqrt((self.X - Xm)*(self.X - Xm) + (self.H - Ym)*(self.H - Ym))
167
168     self.dtheta = - K_q * before.V * np.sin(before.theta - np.arctan(( self.H - Ym
169                                     ))/(self.X - Xm)))/ self.r
170
171     self.theta = before.theta + self.dtheta * timestep
172
173     P = 0
174
175     self.alpha = (self.mass* before.V * self.dtheta + self.mass * 9.8 * np.cos(
176         self.theta))/(P + (0.25 + 0.05/0.24) * 0.5 * air(self.H) * before.V *
177         before.V * S_ref) /3.14159*180
178
179     self.deltaz = self.alpha / 0.24
180
181     if self.deltaz > 30:
182         self.deltaz = 30
183     if self.deltaz < -30:
184         self.deltaz = -30
185
186     self.alpha = 0.24 * self.deltaz
187
188     X = (0.005 * self.alpha * self.alpha + 0.2) * 0.5 * air(self.H) * before.V *
189         before.V * S_ref
190     self.V = before.V + ((P*np.cos(self.alpha*3.14159/180) - X)/self.mass-9.8*np.
191         sin(self.theta)) *timestep
192
193
194
195
196
197
198 ##### 部分函数 #####
199 # 大气参数
200 def air (High):
201     rho0 =1.2495
202     T0 = 288.15
203     Temp = T0 - 0.0065*High
204     rho = rho0 * np.exp(4.25588*np.log(Temp / T0))
205     return rho

```

```
199 # 飞行方案
200 def High_goal(X):
201     if X <= 9100:
202         return 2000 * np.cos(0.000314 * 1.1 * X) + 5000
203     elif X <= 24000:
204         return 3050
205     else:
206         return 0
207
208 # 飞行方案的时间导数
209 def High_goal_dot(X):
210     if X <= 9100:
211         return -2000 * 0.000314 * np.sin(0.000314 * 1.1 * X)
212     elif X <= 24000:
213         return 0
214     else:
215         return 0
216
217 # 读入数据
218 def read_file(file_path):
219     # 读取文档中的数据
220     with open(file_path, 'r', encoding='UTF-8') as f:
221         lines = f.readlines()
222
223     # 提取关键数据
224     data = []
225     for line in lines:
226         items = line.strip().split()
227         if items:
228             data.append(items)
229     return data
230
231 def alpha_cornering( time ):
232     b = np.exp(alpha_constant*(t1-time))
233     return 4* alpha_max * b *(b-1)
234
235
236 ##### 插值函数 #####
237 # cx插值数据预处理
238 data = read_file('data/cx.txt')
239 cx_high_numbers = np.array(data[1], dtype= float)
240 cx_mach_numbers = np.array(data[3], dtype= float)
241 cx_values = [row for row in np.array(data[5:], dtype= float)]
```

```
242 # kind参数决定了插值的类型, 'linear'代表线性插值
243 cx_interp_func = interp2d(cx_high_numbers, cx_mach_numbers, cx_values, kind='linear')
244 # print(cx_interp_func(0.1,0.1),cn_alpha_interp_func(0.3))
245
246 # cx插值数据预处理
247 data = read_file('data/cn_alpha.txt')
248 cn_alpha_mach = np.array(data[1], dtype= float)
249 cn_alpha_values = [row for row in np.array(data[3:], dtype= float)]
250 # 允许外插
251 cn_alpha_interp_func = interp1d(cn_alpha_mach, cn_alpha_values, kind='linear',
    fill_value='extrapolate')
252
253 ## 芯一级推力P_1 插值预处理
254 data = read_file('data/P_1.txt')
255 P_1_high = np.array(data[1], dtype= float)
256 P_1_values = [row for row in np.array(data[3:], dtype= float)]
257
258 P_1_interp_func = interp1d(P_1_high, P_1_values, kind='linear', fill_value='
    extrapolate')
259
260 ## 两大助推器总推力P_2 插值预处理
261 data = read_file('data/P_2.txt')
262 P_2_high = np.array(data[1], dtype= float)
263 P_2_values = [row for row in np.array(data[3:], dtype= float)]
264
265 P_2_interp_func = interp1d(P_2_high, P_2_values, kind='linear', fill_value='
    extrapolate')
266 print(P_2_interp_func(20))
267
268 ## 两小助推器总推力P_3 插值预处理
269 data = read_file('data/P_1.txt')
270 P_3_high = np.array(data[1], dtype= float)
271 P_3_values = [row for row in np.array(data[3:], dtype= float)]
272
273 P_3_interp_func = interp1d(P_3_high, P_3_values, kind='linear', fill_value='
    extrapolate')
274 print(P_3_interp_func(20))
275
276 ##### 计算初始化 #####
277 # 发射方位角
278 A_0 =
279
280 # 飞行初始状态
```

```
281 statu_n = [statu(0, 0, 7000, 250, 0, 320)]
282 statu_n[0].alpha = 0
283 statu_n[0].deltaz = 0
284
285 X_goal = np.arange(0,24000,10)
286 H_goal = [High_goal(i) for i in X_goal]
287 plt.plot(X_goal,H_goal, 'b--', alpha=0.5, linewidth=1, label='飞行方案高度')
288
289 # 方位角
290 ##### 计算开始 #####
291 # 第一阶段
292 while statu_n[-1].Time < 15:
293     statu_n.append(statu(statu_n[-1].Time + timestep))
294     statu_n[-1].Euler(statu_n[-2],0)
295
296
297 ##### 绘图可视化 #####
298 # 绘图
299 X_data = [n.X for n in statu_n]
300 H_data = [n.H for n in statu_n]
301 plt.plot(X_data,H_data, 'r-.', alpha=0.5, linewidth=1, label='实际飞行高度')
302 plt.title("弹道铅垂平面轨迹")
303 plt.legend() #显示上面的label
304 plt.xlabel('X(m)') #x_label
305 plt.ylabel('H(m)')#y_label
306 plt.ylim(0,8000)
307 plt.xlim(0,30000) #仅设置y轴坐标范围
308 plt.savefig('img/飞行轨迹.png', dpi=300)
309 plt.clf()
310
311 T_data = [n.Time for n in statu_n]
312 deltaz_data = [n.deltaz for n in statu_n]
313 plt.plot(T_data,deltaz_data, 'r-.', alpha=0.5, linewidth=1, label='舵偏角$\delta z$')
314 plt.title("飞行方案舵偏角")
315 plt.legend() #显示上面的label
316 plt.xlabel('Time(s)') #x_label
317 plt.ylabel('$\delta z$')#y_label
318 plt.ylim(-50,50)
319 plt.xlim(0,200)
320 plt.savefig('img/飞行舵偏角.png', dpi=300)
321 plt.clf()
322
323 plt.plot(T_data,H_data, 'r-.', alpha=0.5, linewidth=1, label='实际飞行速度V')
```



```
324 plt.title("飞行高度的时间变化曲线")
325 plt.legend() #显示上面的label
326 plt.xlabel('Time(s)') #x_label
327 plt.ylabel('H(m)') #y_label
328 plt.ylim(0,7000)
329 plt.xlim(0,200)
330 plt.savefig('img/飞行高度.png', dpi=300)
331 plt.clf()
332
333 V_data = [n.V for n in statu_n]
334 plt.plot(T_data,V_data, 'r-.', alpha=0.5, linewidth=1, label='实际飞行速度V')
335 plt.title("飞行速度的时间变化曲线")
336 plt.legend() #显示上面的label
337 plt.xlabel('Time(s)') #x_label
338 plt.ylabel('速度V') #y_label
339 plt.ylim(100,250)
340 plt.xlim(0,200)
341 plt.savefig('img/飞行速度.png', dpi=300)
342 plt.clf()
343
344 theta_data = [n.theta*180/3.14159 for n in statu_n]
345 plt.plot(T_data,theta_data, 'r-.', alpha=0.5, linewidth=1, label=r'舵偏角$\theta$')
346 plt.title("飞行方案弹道倾角")
347 plt.legend() #显示上面的label
348 plt.xlabel('Time(s)') #x_label
349 plt.ylabel(r'$\theta$') #y_label
350 plt.ylim(-50,50)
351 plt.xlim(0,200)
352 plt.savefig('img/飞行弹道倾角.png', dpi=300)
353 plt.clf()
```

源代码 2: comparis.py

```
1  """
2  弹道计算程序
3  分析放大系数的影响
4  """
5
6  import numpy as np
7  from matplotlib import pyplot as plt
8
9  # 展示高清图
10 from matplotlib_inline import backend_inline
11 backend_inline.set_matplotlib_formats('svg')
12
13 plt.rcParams['font.sans-serif'] = ['SimHei']
14 plt.rcParams['axes.unicode_minus'] = False
15
16 # 导弹参数
17 S_ref = 0.45
18 L_ref = 2.5
19
20 # 仿真时间步
21 timestep = 0.01
22
23 # 导弹状态定义
24 class statu():
25     __slot__=['Time', 'X', 'H', 'V', 'theta', 'mass', 'alpha', 'deltaz']
26
27     # 位置
28     # 速度
29     # 欧拉角
30     # 角加速度
31     # 舵偏角
32
33     # 初始化
34     def __init__(self, Time, X=0, H=0, V=0, theta=0, mass=0):
35         self.Time = Time
36         self.X = X
37         self.H = H
38         self.V = V
39         self.theta = theta
40         self.mass = mass
41         self.alpha = 0
```

```
42     self.deltaz = 0
43     self.q = 0
44
45     # 显式Euler法, 给定飞行高度
46     def Euler(self, before, dmass, K_phi, K_phi_dot):
47         self.Time = before.Time + timestep
48
49         self.X = before.X + before.V * np.cos(before.theta) * timestep
50         self.H = before.H + before.V * np.sin(before.theta) * timestep
51
52         self.deltaz = K_phi * (self.H - High_goal(self.X)) + K_phi_dot * (before.V * np
53             .sin(before.theta) - High_goal_dot(self.X))
54
55         if self.deltaz > 30:
56             self.deltaz = 30
57         elif self.deltaz < -30:
58             self.deltaz = -30
59
60         self.alpha = 0.24 * self.deltaz
61
62         Y = (0.25 * self.alpha + 0.05 * self.deltaz) * 0.5 * air(self.H) * before.V *
63             before.V * S_ref
64
65         X = (0.005 * self.alpha * self.alpha + 0.2) * 0.5 * air(self.H) * before.V *
66             before.V * S_ref
67
68         self.mass = before.mass - dmass * timestep
69         if dmass == 0:
70             P = 0
71         else:
72             P = 2000
73
74         self.V = before.V + (P * np.cos(before.alpha * 3.14159625 / 180) - X - self.mass
75             * 9.8 * np.sin(before.theta)) / self.mass * timestep
76         self.theta = before.theta + (P * np.sin(self.alpha * 3.14159625 / 180) + Y - self.
77             mass * 9.8 * np.cos(before.theta)) / self.mass / self.V * timestep
78
79     # 比例导引法, 给定目标位置
80     def Euler2(self, before, Xm, Ym, K_q):
81         self.Time = before.Time + timestep
82
83         self.X = before.X + before.V * np.cos(before.theta) * timestep
84         self.H = before.H + before.V * np.sin(before.theta) * timestep
```

```
80         self.mass = before.mass
81
82         self.r = np.sqrt((self.X - Xm)*(self.X - Xm) + (self.H - Ym)*(self.H - Ym))
83
84         self.dq = - before.V * np.sin(before.theta - np.arctan(( self.H - Ym)/(self.X
            - Xm)))/ self.r
85
86         self.theta = before.theta + K_q * self.dq * timestep
87
88         P = 0
89
90
91         self.alpha = (self.mass* before.V * K_q * self.dq + self.mass * 9.8 * np.cos(
            self.theta))/(P + (0.25 + 0.05/0.24) * 0.5 * air(self.H) * before.V *
            before.V * S_ref) /3.14159*180
92
93         self.deltaz = self.alpha / 0.24
94
95         if self.deltaz > 30:
96             self.deltaz = 30
97         if self.deltaz < -30:
98             self.deltaz = -30
99
100        self.alpha = 0.24 *self.deltaz
101
102        X = (0.005 * self.alpha * self.alpha + 0.2) * 0.5 * air(self.H) * before.V *
            before.V * S_ref
103        self.V = before.V + (P*np.cos(before.alpha*3.14159625/180) - X - self.mass
            *9.8*np.sin(before.theta)) /self.mass*timestep
104
105        # 大气参数
106        def air (High):
107            rho0 =1.2495
108            T0 = 288.15
109            Temp = T0 - 0.0065*High
110            rho = rho0 * np.exp(4.25588*np.log(Temp / T0))
111            return rho
112
113        # 飞行方案
114        def High_goal(X):
115            if X <= 9100:
116                return 2000 * np.cos(0.000314 * 1.1 * X) + 5000
117            elif X <= 24000:
```

```
118         return 3050
119     else:
120         return 0
121
122 # 飞行方案的时间导数
123 def High_goal_dot(X):
124     if X <= 9100:
125         return -2000 * 0.000314 * np.sin(0.000314 * 1.1 * X)
126     elif X <= 24000:
127         return 0
128     else:
129         return 0
130
131 def calculate(K_phi ,K_phi_dot, K_q):
132
133     # 飞行初始状态
134     statu_n = [statu(0, 0, 7000, 250, 0, 320)]
135     statu_n[0].alpha = 0
136     statu_n[0].deltaz = 0
137
138     # 第一阶段
139     while statu_n[-1].X < 9100:
140         statu_n.append(statu(statu_n[-1].Time + timestep))
141         statu_n[-1].Euler(statu_n[-2],0,K_phi ,K_phi_dot )
142
143     # 第二阶段
144     while statu_n[-1].X <= 24000:
145         statu_n.append(statu(statu_n[-1].Time + timestep))
146         statu_n[-1].Euler(statu_n[-2],0.46,K_phi ,K_phi_dot)
147
148
149     # 第三阶段
150     while statu_n[-1].X <= 30000 and statu_n[-1].H > 0:
151         statu_n.append(statu(statu_n[-1].Time + timestep))
152         statu_n[-1].Euler2(statu_n[-2],30000,0,K_q)
153
154     return statu_n
155
156 # 飞行方案
157 X_goal = np.arange(0,24000,10)
158 H_goal = [High_goal(i) for i in X_goal]
159
160 # 放大系数
```

```
161 statu_1=calculate(-0.2,-0.5,2)
162 statu_2=calculate(-0.8,-0.5,2)
163
164 # 绘图
165
166 ## 第一个放大系数
167 X_data_1 = [n.X for n in statu_1]
168 H_data_1 = [n.H for n in statu_1]
169 X_data_2 = [n.X for n in statu_2]
170 H_data_2 = [n.H for n in statu_2]
171
172 plt.plot(X_goal,H_goal, 'c--', alpha=0.5, linewidth=1, label='飞行方案高度')
173
174 plt.plot(X_data_1,H_data_1, 'r-.', alpha=0.5, linewidth=1, label=r'$k\_varphi=-0.2$')
175 plt.plot(X_data_2,H_data_2, 'b-.', alpha=0.5, linewidth=1, label=r'$k\_varphi=-0.8$')
176
177 plt.title("弹道铅垂平面轨迹第一阶段对比")
178 plt.legend() #显示上面的label
179 plt.xlabel('X(m)') #x_label
180 plt.ylabel('H(m)')#y_label
181 plt.ylim(3000,7000)
182 plt.xlim(0,10000) #仅设置y轴坐标范围
183 plt.savefig('img/飞行轨迹2.png', dpi=300)
184 plt.clf()
185
186
187 T_data_1 = [n.Time for n in statu_1]
188 T_data_2 = [n.Time for n in statu_2]
189 deltaz_data_1 = [n.deltaz for n in statu_1]
190 deltaz_data_2 = [n.deltaz for n in statu_2]
191
192 plt.plot(T_data_1,deltaz_data_1, 'r-.', alpha=0.5, linewidth=1, label=r'$k\_varphi$
    ==-0.2$')
193 plt.plot(T_data_2,deltaz_data_2, 'b-.', alpha=0.5, linewidth=1, label=r'$k\_varphi$
    ==-0.8$')
194
195 plt.title("飞行方案舵偏角")
196 plt.legend() #显示上面的label
197 plt.xlabel('Time(s)') #x_label
198 plt.ylabel('$\delta z$')#y_label
199 plt.ylim(-50,50)
200 plt.xlim(0,200)
201 plt.savefig('img/飞行舵偏角2.png', dpi=300)
```

```
202 plt.clf()
203
204 ## 第二个放大系数
205 statu_3 = calculate(-0.6,-0.3,3)
206 statu_4 = calculate(-0.6,-0.5,3)
207
208 X_data_3 = [n.X for n in statu_3]
209 H_data_3 = [n.H for n in statu_3]
210 X_data_4 = [n.X for n in statu_4]
211 H_data_4 = [n.H for n in statu_4]
212
213 plt.plot(X_goal,H_goal, 'c--', alpha=0.5, linewidth=1, label='飞行方案高度')
214
215 plt.plot(X_data_3,H_data_3, 'r-.', alpha=0.5, linewidth=1, label=r'$\dot{k}_\varphi$
    ==-0.3$')
216 plt.plot(X_data_4,H_data_4, 'b-.', alpha=0.5, linewidth=1, label=r'$\dot{k}_\varphi$
    ==-0.5$')
217
218 plt.title("弹道铅垂平面轨迹阶跃处对比")
219 plt.legend() #显示上面的label
220 plt.xlabel('X(m)') #x_label
221 plt.ylabel('H(m)')#y_label
222 plt.ylim(2800,3200)
223 plt.xlim(8500,11000) #仅设置y轴坐标范围
224 plt.savefig('img/飞行轨迹3.png', dpi=300)
225 plt.clf()
226
227 T_data_3 = [n.Time for n in statu_3]
228 T_data_4 = [n.Time for n in statu_4]
229 deltaz_data_3 = [n.deltaz for n in statu_3]
230 deltaz_data_4 = [n.deltaz for n in statu_4]
231
232 plt.plot(T_data_3,deltaz_data_3, 'r-.', alpha=0.5, linewidth=1, label=r'$\dot{k}_\varphi$
    ==-0.3$')
233 plt.plot(T_data_4,deltaz_data_4, 'b-.', alpha=0.5, linewidth=1, label=r'$\dot{k}_\varphi$
    ==-0.5$')
234
235 plt.title("飞行方案舵偏角")
236 plt.legend() #显示上面的label
237 plt.xlabel('Time(s)') #x_label
238 plt.ylabel('$\delta z$')#y_label
239 plt.ylim(-50,50)
240 plt.xlim(0,200)
```

```
241 plt.savefig('img/飞行舵偏角3.png', dpi=300)
242 plt.clf()
243
244
245 ## 第三个放大系数
246 statu_5 = calculate(-0.6, -0.5, 3)
247 statu_6 = calculate(-0.6, -0.5, 6)
248
249 X_data_5 = [n.X for n in statu_5]
250 H_data_5 = [n.H for n in statu_5]
251 X_data_6 = [n.X for n in statu_6]
252 H_data_6 = [n.H for n in statu_6]
253
254 plt.plot(X_goal, H_goal, 'c--', alpha=0.5, linewidth=1, label='飞行方案高度')
255
256 plt.plot(X_data_5, H_data_5, 'r-.', alpha=0.5, linewidth=1, label=r'$k_3=3$')
257 plt.plot(X_data_6, H_data_6, 'b-.', alpha=0.5, linewidth=1, label=r'$k_3=5$')
258
259
260 plt.title("弹道铅垂平面第三阶段轨迹对比")
261 plt.legend() #显示上面的label
262 plt.xlabel('X(m)') #x_label
263 plt.ylabel('H(m)') #y_label
264 plt.ylim(0, 3500)
265 plt.xlim(24000, 30000) #仅设置y轴坐标范围
266 plt.savefig('img/飞行轨迹4.png', dpi=300)
267 plt.clf()
268
269 T_data_5 = [n.Time for n in statu_5]
270 T_data_6 = [n.Time for n in statu_6]
271 deltaz_data_5 = [n.deltaz for n in statu_5]
272 deltaz_data_6 = [n.deltaz for n in statu_6]
273
274 plt.plot(T_data_5, deltaz_data_5, 'r-.', alpha=0.5, linewidth=1, label=r'$k_3=3$')
275 plt.plot(T_data_6, deltaz_data_6, 'b-.', alpha=0.5, linewidth=1, label=r'$k_3=5$')
276 plt.title("飞行方案舵偏角")
277 plt.legend() #显示上面的label
278 plt.xlabel('Time(s)') #x_label
279 plt.ylabel('$\delta z$') #y_label
280 plt.ylim(-50, 50)
281 plt.xlim(0, 200)
282 plt.savefig('img/飞行舵偏角4.png', dpi=300)
283 plt.clf()
```