

轻型货车性能计算与传动系统优化分析报告-陈柏宇

1. 前言

随着汽车工业的快速发展，车辆动力性与燃油经济性的平衡已成为工程设计的核心课题。本次作业基于车辆工程核心课程《汽车理论》的知识体系，以轻型货车为研究对象，通过理论推导、数值计算与编程实践，系统分析了车辆动力性能（驱动力、最高车速、爬坡度）、燃油经济性（等速油耗、循环工况油耗）及传动系统参数（主减速比、变速器档位）对车辆性能的综合影响。

实践意义与学习目标：

- 理论应用**：将课本中的动力传动模型、行驶阻力方程、燃油消耗模型转化为可计算的数学模型。
- 工程实践**：通过编程（Python）实现复杂方程的数值求解，培养解决实际工程问题的能力。
- 设计思维**：探索传动系统参数优化方法，理解参数间的耦合关系（如动力性 vs. 经济性）。
- 行业衔接**：模拟真实车辆开发流程，为未来参与新能源汽车动力系统设计奠定基础。

2. 作业核心要点与解题思路

2.1 初始设置

- 搭建 Python 编译环境（Python/VS Code 的安装）
- 使用库管理工具 pip 安装 scipy/matplotlib/pandas/numpy 等库
- 修改字体设置，使表格中文字能够被正确显示

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

Fence 1

2.2 题目 1.3：动力性能计算

核心问题：

- 驱动力-阻力平衡图绘制
- 最高车速、最大爬坡度及附着率计算
- 加速时间求解（图解积分法或数值积分）

入手方法：

- 数据预处理**：整理发动机外特性曲线（转矩-转速公式）、变速器传动比、车辆质量参数。
- 驱动力模型**：

$$F_t = \frac{T_q(n) \cdot i_g \cdot i_0 \cdot \eta_T}{r}$$

其中， $T_q(n)$ 为发动机扭矩， i_g 为变速器传动比， i_0 为主减速比， r 为车轮半径。

- 行驶阻力模型**：

$$F_r = G \cdot f + \frac{1}{2} \rho C_d A v^2$$

(滚动阻力 + 空气阻力)

4. **平衡点求解**：通过迭代法寻找驱动力与阻力相等的车速（最高车速）。

5. **爬坡度计算**：通过平衡方程 $F_t = G \sin \theta + G f \cos \theta$ 求解 θ_{\max} 。

6. **加速时间计算**：使用数值积分的方式求解。根据定义， $a = \frac{dv}{dt}$ ， $\frac{1}{a} = \frac{dt}{dv}$ ， $t = \int_{v_0}^{v_f} \frac{1}{a(v)} dv$ 。其中， v_0 为起始速度， v_f 为结束速度。因此，只要得出加速度与速度之间的关系式，就可以求出 0 到 70km/h 的加速时间。

7. **代码关键实现（Python 示例）**：

计算发动机牵引力：

```
1 def traction_force(n, ig, i0):
2     Tq = engine_torque(n) # 调用发动机扭矩公式
3     return Tq * ig * i0 * eta_T / r
4 def resistance_force(v):
5     v_mps = v / 3.6 # 单位转换
6     return G * f + 0.5 * rho * CdA * v_mps**2
```

Fence 2

计算 $\frac{1}{a}$ ：

```
1 def inverse_acceleration(v): #定义加速度倒数的函数
2     gear=2
3     ig = gear_ratios[gear-1]
4     n = (v / 3.6) * 60 * i0 * ig / (2 * np.pi * r)
5     while(n>n_max):
6         gear+=1
7         ig = gear_ratios[gear-1]
8         n = (v / 3.6) * 60 * i0 * ig / (2 * np.pi * r)
9     F_traction = traction_force(n, ig)
10    F_resistance = resistance_force(v)
11    F_net = F_traction - F_resistance
12    delta_m=cal_delta_m(gear)
13    a = F_net / delta_m
14    if a <= 0:
15        return np.inf
16    return 1/a
```

Fence 3

之后，使用 `scipy.integrate()` 函数进行数值积分

```
1 time, error = integrate.quad(inverse_acceleration, v_min_2, 70)
2 # 由于直接使用二档起步，忽略半离合时间，因此直接由二档最低时速（对应转速600rpm）起步
```

Fence 4

2.3 题目 2.7：燃油经济性分析

核心问题：

- 功率平衡图绘制
- 最高档与次高档等速百公里油耗曲线
- 六工况循环油耗计算

关键公式：

- 发动机功率需求:

$$P_e = \frac{(F_r + F_a + F_g) \cdot v}{\eta_T}$$

- 燃油消耗率插值: 根据发动机转速 n 和功率 P_e , 从负荷特性表中插值获取 b 。
- 百公里油耗计算:

$$Q_{100} = \frac{b \cdot P_e \cdot 100}{3600 \cdot \rho_{\text{fuel}} \cdot \frac{1}{v}}$$

$$b = B_0 + B_1 \cdot P_e + B_2 \cdot P_e^2 + B_3 \cdot P_e^3 + B_4 \cdot P_e^4$$

代码实现难点:

- 插值法:

```
1 def interpolate_coefficients(n):
2     # 根据转速n插值获取B0-B4系数
3     idx = np.searchsorted(n_values, n)
4     return w_low * coeffs_low + w_high * coeffs_high
```

Fence 5

- 计算燃油消耗率:

```
1 def calculate_fuel_rate(n, Pe):
2     # 计算燃油消耗率 (mL/s)
3     if Pe <= 0 or n < n_min:
4         return Qid
5     coeffs = interpolate_coefficients(n)
6     if coeffs is None:
7         return Qid
8     B0, B1, B2, B3, B4 = coeffs
9     b = B0 + B1*Pe + B2*Pe**2 + B3*Pe**3 + B4*Pe**4 # g/(kw·h)
10    return (b * Pe) / (3600 * fuel_density)
```

Fence 6

2.4 题目 3.1: 传动比优化分析

核心问题:

- 绘制不同主减速比 (i_0) 下的燃油经济性-加速时间曲线
- 分析 4 档与 5 档变速器对性能的影响

设计变量与目标函数:

- 自变量: $i_0 \in \{5.17, 5.43, 5.83, 6.17, 6.33\}$
- 目标函数:

- 加速时间 t_{accel} (0-70 km/h)
- 百公里油耗 Q_{100}

- 数学公式:

- 发动机输出扭矩:

$$T_q = -19.313 + 295.27 \cdot n - 165.44 \cdot n^2 + 40.874 \cdot n^3 - 3.8445 \cdot n^4$$

牵引力计算:

$$T_w = T_q \cdot i_g \cdot i_0 \cdot \eta_t, \quad F = \frac{T_w}{r}$$

i_0	加速时间 (s)	油耗 (L/100km)
5.17	18.2	15.61
5.83	15.8	15.99
6.33	14.3	17.01

Table 1

- 优化结论：**：根据不同需求采用不同主减速比。对于加速有一定要求的工况，就选用更大的主减速比。反之对于燃油经济性有要求的，选择更小的主减速比。

3.3. 实践对专业能力与职业发展的帮助

3.1 专业能力提升

- 数学建模能力：**将车辆动力学问题转化为微分方程与优化问题。
- 编程实践：**掌握 Python 数值计算库（NumPy、SciPy）及数据可视化（Matplotlib）。
- 工程思维：**理解参数敏感性（如 i_0 对油耗与动力的矛盾影响）。

3.2 行业应用场景

- 传统燃油车优化：**通过调整传动比降低油耗（如商用车车队节能改造）。
- 新能源汽车设计：**电驱动系统参数匹配（电机特性曲线与减速比选择）。
- 智能驾驶算法：**基于动力模型的能量管理策略（如预测性巡航控制）。

3.3 对个人发展的启示

- 技术视野拓展：**认识到车辆设计是多目标权衡（性能、成本、法规）。
- 工具链熟悉：**Python/MATLAB 成为未来研发的核心工具（如 AVL Cruise 仿真）。
- 跨学科思维：**结合控制理论（如 PID 调速）、材料科学（轻量化）提升综合竞争力。

4. 完整代码

4.1 1.3 驱动力-阻力平衡图（Python）

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from scipy import integrate
4 #发动机参数
5 n_min = 600 #发动机最低转速
6 n_max = 4000 #发动机最高转速
7 n_max_power = 3800 # 最大功率转速，假设为3800转，需要从发动机特性确定。
8 n_max_torque = 2500 # 最大扭矩转速，假设为2500转，需要从发动机特性确定。
9 #车辆质量参数
10 m_load = 2000 #装载质量 (kg)
11 m_empty = 1800 #整车整备质量 (kg)
```

```

12 m_total = 3880 #总质量 (kg)
13 # 速度参数
14 v_max = 120 # 最大车速 (km/h)
15 v_min = 0 # 最小车速 (km/h)
16 v_min_2=5.14229 # 二档最低车速
17 # 车轮参数
18 r = 0.367 # 车轮半径 (m)
19 # 传动系统参数
20 eta_t = 0.85 # 传动系统机械效率
21 f = 0.013 # 滚动阻力系数
22 CdA = 2.77 # 空气阻力系数 × 迎风面积 (m^2)
23 i0 = 5.83 # 主减速器传动比
24 # 转动惯量
25 I_f = 0.218 # 飞轮转动惯量 (kg*m^2)
26 I_w1 = 1.798 # 前轮转动惯量 (kg*m^2)
27 I_w2 = 3.598 # 后轮转动惯量 (kg*m^2)
28 #变速箱传动比(4档)
29 ig_1_4 = 6.09
30 ig_2_4 = 3.09
31 ig_3_4 = 1.71
32 ig_4_4 = 1.00
33 #变速箱传动比(5档)
34 ig_1_5 = 5.56
35 ig_2_5 = 2.769
36 ig_3_5 = 1.644
37 ig_4_5 = 1.00
38 ig_5_5 = 0.793
39 #轴距 & 质心位置
40 L = 3.2 #轴距
41 a = 1.947 #质心至前轴距离 (满载)
42 h_g = 0.9 #质心高
43 #空气密度
44 rho = 1.225 # 空气密度 (kg/m^3) 在标准大气条件下
45 g = 9.81 #重力加速度
46 G = m_total * g #重力
47 # 考虑转动惯量的牛二
48 def cal_delta_m(gear):
49     return m_total + (2*I_w1 + 2*I_w2 + I_f * i0 **2 * gear_ratios[gear-1]
50     **2 )/(r**2)
51 # 定义函数：计算发动机转矩
52 def engine_torque(n):
53     """n: 发动机转速 (r/min)"""
54     n_scaled = n / 1000
55     Tq = -19.313 + 295.27 * n_scaled - 165.44 * n_scaled**2 + 40.874 *
56     n_scaled**3 - 3.8445 * n_scaled**4
57     return Tq
58 def traction_force(n, ig):
59     """
60     n: 发动机转速 (r/min)
61     ig: 变速器传动比
62     """
63     Tq = engine_torque(n)
64     Tw = Tq * ig * i0 * eta_t # 车轮转矩
65     F = Tw / r # 驱动力
66     return F

```

```

65 def resistance_force(v):
66     """v: 车速 (km/h)"""
67     v_mps = v / 3.6 # 将 km/h 转换为 m/s
68     Fr = G * f # 滚动阻力
69     Fa = 0.5 * rho * CdA * v_mps**2 # 空气阻力
70     F_total = Fr + Fa # 总阻力
71     return F_total
72 # 选择变速箱类型 (4 或 5)
73 gearbox_type = 5 # 选择 5 档变速箱
74 # 定义传动比列表
75 if gearbox_type == 4:
76     gear_ratios = [ig_1_4, ig_2_4, ig_3_4, ig_4_4]
77     num_gears = 4
78 elif gearbox_type == 5:
79     gear_ratios = [ig_1_5, ig_2_5, ig_3_5, ig_4_5, ig_5_5]
80     num_gears = 5
81 else:
82     raise ValueError("Invalid gearbox type. Choose 4 or 5.")
83 def calculate_max_speed_iterative(gear_ratios, n_max, r, i0, eta_t, f,
84 CdA, rho, G, v_min, v_max, resistance_force_values): # 计算最高车速
85     max_speed = 0
86     v_range = np.linspace(v_min, v_max, 100)
87     for ig in gear_ratios:
88         gear_max_speed = 0 # 当前档位的最高速度
89         for v in v_range:
90             # 计算发动机转速
91             n = (v / 3.6) * 60 * i0 * ig / (2 * np.pi * r)
92             # 检查转速是否超过最大转速
93             if n > n_max:
94                 gear_max_speed = (n_max / 60) * (2 * np.pi * r) * (3.6 /
95 (i0 * ig))
96                 break # 超过最大转速, 退出循环
97             # 计算驱动力和阻力
98             index = np.argmin(np.abs(v_range - v))
99             F_traction = traction_force(n, ig)
100             F_resistance = resistance_force_values[index] # 使用之前定义的函数
101             # 检查驱动力是否小于等于阻力
102             if F_traction <= F_resistance and v>80:
103                 gear_max_speed = v
104                 break # 找到平衡点, 退出循环
105         max_speed = max(max_speed, gear_max_speed)
106     return max_speed
107 def calculate_max_grade(gear_ratios, n_max_torque, r, i0, eta_t, f, G):
108     """计算最大爬坡度"""
109     ig_1 = gear_ratios[0]
110     Tq_max = engine_torque(n_max_torque)
111     F_traction_max = (Tq_max * ig_1 * i0 * eta_t) / r
112     theta_max=0
113     for theta in np.linspace(0, np.pi/2, 100):
114         if G * np.sin(theta)+G * f * np.cos(theta) > F_traction_max:
115             break
116         else:
117             theta_max=theta
118     max_grade = np.tan(theta_max) * 100

```

```

117     return max_grade
118 def calculate_adhesion_rate(m_total, g, a, h_g, L, theta, F_traction_max):
119     """计算附着率"""
120     F_z = (m_total * g * (a * np.cos(theta) + h_g * np.sin(theta))) / L
121     mu = F_traction_max / F_z
122     return mu
123 # 创建图表
124 plt.figure(figsize=(18, 12)) # 调整图表大小，更大的图表，容纳更多子图
125 # 绘制驱动力-车速曲线
126 plt.subplot(3, 2, 1) # 创建一个子图，调整为 3 行 2 列
127 v_range = np.linspace(v_min, v_max, 100) # 使用统一的车速范围
128 for i in range(num_gears):
129     ig = gear_ratios[i]
130     # 生成发动机转速范围
131     n_range = np.linspace(n_min, n_max, 100)
132     # 计算车速 (km/h)
133     v_range_temp = n_range * r * 2 * np.pi / 60 / i0 / ig * 3.6
134     # 计算驱动力
135     traction_force_values = [traction_force(n, ig) for n in n_range]
136     plt.plot(v_range_temp, traction_force_values, label=f'Gear {i+1}')
137 plt.xlabel('Vehicle Speed (km/h)')
138 plt.ylabel('Traction Force (N)')
139 plt.title('Traction Force vs. Speed ({}, speed)'.format(gearbox_type))
140 plt.grid(True)
141 plt.legend() # 显示图例
142 plt.xlim(0, v_max) # 限制 x 轴范围
143 plt.ylim(0, 15000) # 限制 y 轴范围
144 # 绘制扭矩-转速曲线
145 plt.subplot(3, 2, 2) # 创建一个子图
146 n_range = np.linspace(n_min, n_max, 100) # 在最小转速和最大转速之间生成 100 个点
147 torque_values = [engine_torque(n) for n in n_range]
148 plt.plot(n_range, torque_values)
149 plt.xlabel('Engine Speed (r/min)')
150 plt.ylabel('Torque (N*m)')
151 plt.title('Engine Torque vs. Speed')
152 plt.grid(True)
153 # 绘制阻力-速度曲线
154 plt.subplot(3, 2, 3) # 创建一个子图
155 v_range = np.linspace(v_min, v_max, 100) # 使用统一的车速范围
156 resistance_force_values = [resistance_force(v) for v in v_range]
157 plt.plot(v_range, resistance_force_values)
158 plt.xlabel('Vehicle Speed (km/h)')
159 plt.ylabel('Resistance Force (N)') # 修改 y 轴标签
160 plt.title('Resistance Force vs. Speed') # 修改标题
161 plt.grid(True)
162 plt.xlim(0, v_max) # 限制 x 轴范围
163 # 绘制驱动力与阻力平衡图
164 plt.subplot(3, 2, 4) # 创建一个子图
165 v_range = np.linspace(v_min, v_max, 100) # 使用统一的车速范围
166 resistance_force_values = [resistance_force(v) for v in v_range] # 计算阻力
167 # 绘制每一档的驱动力曲线和阻力曲线
168 for i in range(num_gears):
169     ig = gear_ratios[i]

```

```

170     # 计算驱动力
171     n_range = np.linspace(n_min, n_max, 100)
172     v_range_temp = n_range * r * 2 * np.pi / 60 / i0 / ig * 3.6
173     traction_force_values = [traction_force(n, ig) for n in n_range]
174     # 截断驱动力曲线，只保留与阻力曲线车速范围重叠的部分
175     valid_indices = (v_range_temp >= v_min) & (v_range_temp <= v_max)
176     v_range_truncated = v_range_temp[valid_indices]
177     traction_force_values_truncated = np.array(traction_force_values)
178     [valid_indices]
179     plt.plot(v_range_truncated, traction_force_values_truncated,
180             label=f'Gear {i+1}') # 绘制驱动力曲线
181     plt.plot(v_range, resistance_force_values, label='Resistance',
182             color='black', linestyle='--') # 绘制阻力曲线，黑色虚线
183     plt.xlabel('Vehicle Speed (km/h)')
184     plt.ylabel('Force (N)')
185     plt.title('Traction & Resistance Force Balance')
186     plt.grid(True)
187     plt.xlim(0, v_max) # 限制 x 轴范围
188     plt.ylim(0, 15000)
189     plt.legend()
190     # 添加加速度倒数曲线图
191     plt.subplot(3, 2, 5)
192     v_range = np.linspace(v_min, 70, 200) # 速度范围从 v_min 到 70 km/h
193     # ig = gear_ratios[1] # 第二个传动比对应2档
194     def inverse_acceleration(v): #定义加速度倒数的函数
195         gear=2
196         ig = gear_ratios[gear-1]
197         n = (v / 3.6) * 60 * i0 * ig / (2 * np.pi * r)
198         while(n>n_max):
199             gear+=1
200             ig = gear_ratios[gear-1]
201             n = (v / 3.6) * 60 * i0 * ig / (2 * np.pi * r)
202             F_traction = traction_force(n, ig)
203             F_resistance = resistance_force(v)
204             F_net = F_traction - F_resistance
205             delta_m=cal_delta_m(gear)
206             a = F_net / delta_m
207             if a <= 0:
208                 return np.inf
209             return 1/a
210     acceleration_inverse_values = [inverse_acceleration(v) for v in v_range]
211     plt.plot(v_range, acceleration_inverse_values)
212     plt.xlabel('Vehicle Speed (km/h)')
213     plt.ylabel('Inverse of Acceleration (s^2/m)')
214     plt.title('Inverse of Acceleration vs. Speed (Gear 2&3&4)')
215     plt.grid(True)
216     plt.xlim(v_min, 70)
217     # 计算最大速度
218     v_range = np.linspace(v_min, v_max, 100)
219     resistance_force_values = [resistance_force(v) for v in v_range]
220     speed_interval = 0.1 # 速度间隔 (km/h)
221     max_speed_iterative = calculate_max_speed_iterative(gear_ratios, n_max, r,
222     i0, eta_t, f, CdA, rho, G, v_min, v_max, resistance_force_values)
223     print(f"\n迭代法计算最高车速: {max_speed_iterative:.2f} km/h")
224     # 计算最大爬坡度

```



```

221 max_grade = calculate_max_grade(gear_ratios, n_max_torque, r, i0, eta_t,
222 f, G)
223 print(f"理论最大爬坡度: {max_grade:.2f} %")
224 # 爬坡度对应附着率检查
225 theta = np.arctan(max_grade / 100) # 坡度角
226 ig_1 = gear_ratios[0]
227 Tq_max = engine_torque(n_max_torque)
228 F_traction_max = (Tq_max * ig_1 * i0 * eta_t) / r
229 adhesion_rate = calculate_adhesion_rate(m_total, g, a, h_g, L, theta,
230 F_traction_max)
231 print(f"最大爬坡度对应附着率: {adhesion_rate:.2f}")
232 # 使用积分计算加速时间
233 time, error = integrate.quad(inverse_acceleration, v_min_2, 70)
234 print(f"\n用2档起步加速行驶至70km/h的加速时间 (积分): {time:.2f} s")
235 plt.tight_layout()
236 plt.show()

```

Fence 7

输出结果:

- 1 迭代法计算最高车速: 99.39 km/h
- 2 理论最大爬坡度: 34.61 %
- 3 最大爬坡度对应附着率: 0.51
- 4 用 2 档起步加速行驶至 70km/h 的加速时间 (积分): 95.10 s

Fence 8

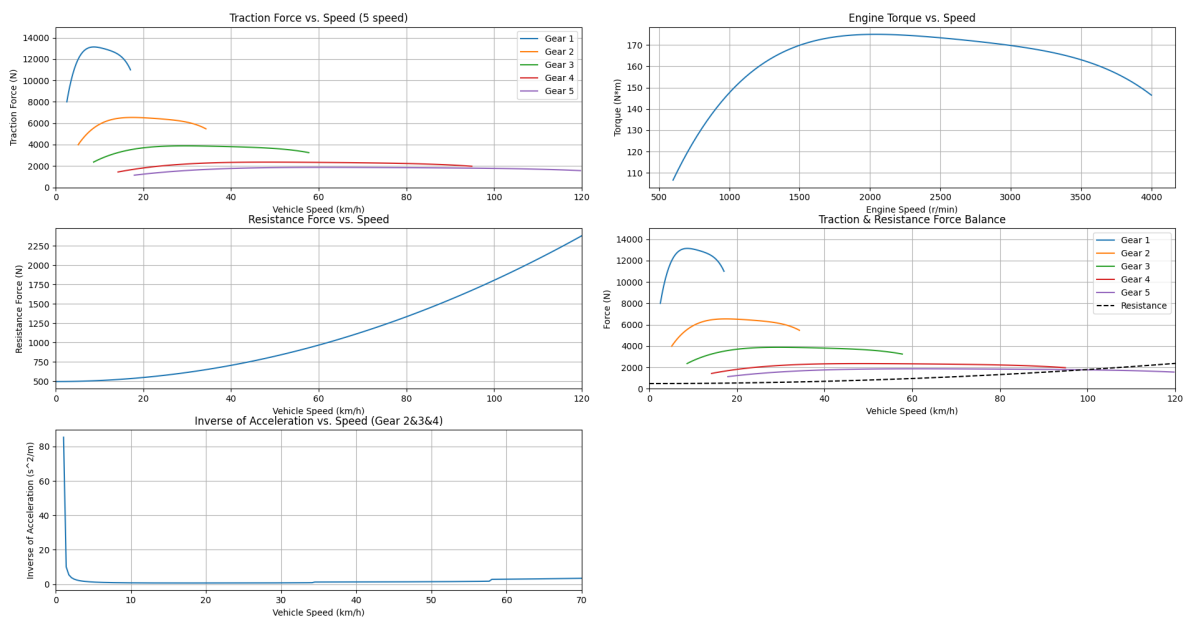


Figure 1

4.2 2.7 六工况循环油耗计算

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from scipy import integrate
4 import pandas as pd
5
6 plt.rcParams['font.sans-serif'] = ['SimHei']
7 plt.rcParams['axes.unicode_minus'] = False
8

```

```

9      #发动机参数
10     n_min = 600 #发动机最低转速
11     n_max = 4000 #发动机最高转速
12     n_max_power = 3800 # 最大功率转速，假设为3800转，需要从发动机特性确定。
13     n_max_torque = 2500 # 最大扭矩转速，假设为2500转，需要从发动机特性确定。
14
15     #车辆质量参数
16     m_load = 2000 #装载质量 (kg)
17     m_empty = 1800 #整车整备质量 (kg)
18     m_total = 3880 #总质量 (kg)
19
20     # 速度参数
21     v_max = 120 # 最大车速 (km/h)
22     v_min = 0 # 最小车速 (km/h)
23     v_min_2=5.14229 # 二档最低车速
24
25     # 车轮参数
26     r = 0.367 # 车轮半径 (m)
27
28     # 传动系统参数
29     eta_t = 0.85 # 传动系统机械效率
30     f = 0.013 # 滚动阻力系数
31     CdA = 2.77 # 空气阻力系数 × 迎风面积 (m^2)
32     i0 = 5.83 # 主减速器传动比
33     gear_ratios = [5.56, 2.769, 1.644, 1.00, 0.793] # 五档变速器传动比
34     num_gears=5 #档位数
35     gear_top = gear_ratios[-1] # 最高档传动比
36     gear_second = gear_ratios[-2] # 次高档传动比
37
38     # 转动惯量
39     I_f = 0.218 # 飞轮转动惯量 (kg*m^2)
40     I_w1 = 1.798 # 前轮转动惯量 (kg*m^2)
41     I_w2 = 3.598 # 后轮转动惯量 (kg*m^2)
42
43     #几何参数
44     L = 3.2 #轴距
45     a = 1.947 #质心至前轴距离（满载）
46     h_g = 0.9 #质心高
47
48     #其他参数
49     rho = 1.225 # 空气密度 (kg/m^3) 在标准大气条件下
50     Qid = 0.299 # 怠速油耗 (mL/s)
51     fuel_density = 0.74 # 燃油密度 (g/mL)
52     g = 9.81 #重力加速度
53     G = m_total * g #重力
54
55     # ===== 发动机特性参数
56     =====
57     engine_coeffs = pd.DataFrame({
58         'n': [815, 1207, 1614, 2012, 2603, 3006, 3403, 3804],
59         'B0': [1326.8, 1354.7, 1284.4, 1222.9, 1141.0, 1051.2, 1233.9,
1129.7],
        'B1':
        [-416.46, -303.98, -189.75, -121.59, -98.893, -73.714, -84.478, -45.291],

```

```

60     'B2': [72.379, 36.657, 14.524, 7.0035, 4.4763, 2.8593, 2.9788,
0.71113],
61     'B3':
[-5.8629, -2.0553, -0.51184, -0.18517, -0.091077, -0.05138, -0.047449, -0.0007521
5],
62     'B4':
[0.17768, 0.043072, 0.0068164, 0.0018555, 0.00068906, 0.00035032, 0.00028230, -0.
000038568]
63 })
64
65 # ===== 核心计算函数 =====
66 def interpolate_coefficients(n):
67     """根据转速n插值获取B0-B4系数"""
68     n_values = engine_coeffs['n'].values
69     if n < n_values[0] or n > n_values[-1]:
70         return None
71     idx = np.searchsorted(n_values, n)
72     idx = max(1, min(idx, len(n_values)-1))
73     n_low, n_high = n_values[idx-1], n_values[idx]
74     w_low = (n_high - n) / (n_high - n_low)
75     w_high = (n - n_low) / (n_high - n_low)
76     coeffs = w_low * engine_coeffs.iloc[idx-1, 1:] + w_high *
engine_coeffs.iloc[idx, 1:]
77     return coeffs.values
78
79 # 考虑转动惯量的牛二
80 def cal_delta_m(gear):
81     return m_total + (2*I_w1 + 2*I_w2 + I_f * i0 **2 * gear_ratios[gear-1]
**2 )/(r**2)
82
83 def engine_torque(n):
84     """n: 发动机转速 (r/min)"""
85     n_scaled = n / 1000
86     Tq = -19.313 + 295.27 * n_scaled - 165.44 * n_scaled**2 + 40.874 *
n_scaled**3 - 3.8445 * n_scaled**4
87     return Tq
88
89 def traction_force(n, ig):
90     """
91     n: 发动机转速 (r/min)
92     ig: 变速器传动比
93     """
94     Tq = engine_torque(n)
95     Tw = Tq * ig * i0 * eta_t # 车轮转矩
96     F = Tw / r # 驱动力
97     return F
98
99 def resistance_force(v):
100     """v: 车速 (km/h)"""
101     v_mps = v / 3.6 # 将 km/h 转换为 m/s
102     Fr = G * f # 滚动阻力
103     Fa = 0.5 * rho * CdA * v_mps**2 # 空气阻力
104     F_total = Fr + Fa # 总阻力
105     return F_total
106

```

```

107 #速度(km/h)计算发动机转速 (r/min)
108 def cal_rpm(v_kmh, gear):
109     v = v_kmh / 3.6 # km/h → m/s
110     wheel_rps = v / (2 * np.pi * r) # 车轮转/秒
111     return wheel_rps * gear_ratios[gear-1] * i0 * 60 # 发动机转速
112
113 def cal_vkmh(n, gear): #计算车速
114     n_scaled = n / (i0 * gear_ratios[gear-1])
115     v = (2 * np.pi * r) * n_scaled / 60
116     v_kmh=v*3.6
117     return v_kmh
118
119 def engine_torque(n):
120     """n: 发动机转速 (r/min)"""
121     n_scaled = n / 1000
122     Tq = -19.313 + 295.27 * n_scaled - 165.44 * n_scaled**2 + 40.874 *
n_scaled**3 - 3.8445 * n_scaled**4
123     return Tq
124
125 def cal_resist_power(v_kmh, acceleration, slope_angle, gear): #计算发动机功率
需求 (kw)
126     v = v_kmh / 3.6 # m/s
127     F_roll = G * f * np.cos(slope_angle)
128     F_air = 0.5 * rho * CdA * v**2 # 修改后的空气阻力计算
129     F_accel = cal_delta_m(gear) * acceleration
130     F_gradient = G * np.sin(slope_angle)
131     F_total = F_roll + F_air + F_accel + F_gradient
132     return (F_total * v) / (eta_t * 1000)
133
134 def calculate_fuel_rate(n, Pe):
135     # 计算燃油消耗率 (mL/s)
136     if Pe <= 0 or n < n_min:
137         return Qid
138     coeffs = interpolate_coefficients(n)
139     if coeffs is None:
140         return Qid
141     B0, B1, B2, B3, B4 = coeffs
142     b = B0 + B1*Pe + B2*Pe**2 + B3*Pe**3 + B4*Pe**4 # g/(kw·h)
143     return (b * Pe) / (3600 * fuel_density)
144
145 def cal_fuel_time(t, v_start, accel): #建立在6种工况中油耗(mL/s)与时间的函数
146     v=v_start+accel*t
147     n=cal_rpm(v, gear=3)
148     Pe=cal_resist_power(v, accel, 0, gear=3)
149     fuel_rate=calculate_fuel_rate(n, Pe) #油耗(mL/s)
150     return fuel_rate/1000 # 转换为L/s
151
152 # ===== 六工况油耗计算
=====
153 six_conditions = [
154     {'type': '匀速', 'duration': 7.2, 'v_start': 25, 'v_end': 25,
'accel': 0},
155     {'type': '匀加速', 'duration': 16.7, 'v_start': 25, 'v_end': 40,
'accel': 0.25},

```

```

156     {'type': '匀速', 'duration':22.5, 'v_start':40, 'v_end':40,
157     'accel':0},
158     {'type': '匀加速', 'duration':14.0, 'v_start':40, 'v_end':50,
159     'accel':0.20},
160     {'type': '匀速', 'duration':18.0, 'v_start':50, 'v_end':50,
161     'accel':0},
162     {'type': '匀减速', 'duration':19.3, 'v_start':50, 'v_end':25,
163     'accel':-0.36},
164 ]
165
166 def simulate_six_cycles():
167     total_fuel=0 # 总油耗 (mL)
168     for cond in six_conditions:
169         duration = cond['duration']
170         accel = cond['accel']
171         v_start = cond['v_start']
172         v_end = cond['v_end']
173         fuel=integrate.quad(cal_fuel_time, 0, duration, args=
174         (v_start,accel))[0]
175         total_fuel+=fuel
176     return total_fuel
177
178 # ===== 功率平衡图 =====
179 def plot_power_balance():
180     plt.figure(figsize=(12, 8))
181     v_range = np.linspace(0, 120, 1000) # 调整车速范围
182     n_range = np.linspace(n_min, n_max, 1000) # 发动机转速范围
183     colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd']
184
185     # 各档位驱动力功率
186     for idx in range(num_gears):
187         ig=gear_ratios[idx] # 当前档位传动比
188         v_range_temp = cal_vkmh(n_range, idx+1) # 计算车速范围
189         Pe =
190         [traction_force(n,gear_ratios[idx])*cal_vkmh(n,idx+1)/3.6/1000 for n in
191         n_range] # 计算功率
192         plt.plot(v_range_temp, Pe, label=f'{idx+1}档 (ig={ig:.2f})',
193         linewidth=2)
194
195     # 行驶阻力功率
196     resist_power = [(0.5*rho*CdA*(v/3.6)**3 + G*f*(v/3.6))/1000 for v in
197     v_range]
198     plt.plot(v_range, resist_power, label='行驶阻力', linewidth=3,
199     linestyle='--', color='#2c3e50')
200     plt.title('货车功率平衡图', fontsize=16)
201     plt.xlabel('车速 (km/h)', fontsize=12)
202     plt.ylabel('功率 (kw)', fontsize=12)
203     plt.grid(True, alpha=0.3)
204     plt.legend()
205     plt.xlim(0, 120)
206     plt.ylim(0, 80)
207     plt.show()
208
209 # ===== 等速油耗曲线 =====
210 def plot_constant_speed_consumption():

```

```

201 plt.figure(figsize=(10, 6)) # 首先创建 figure
202
203 # 计算四档
204 v_range_4 = np.linspace(20, 90, 1000)
205 fuel_consumption_4 = []
206 for v in v_range_4:
207     n = cal_rpm(v, 4)
208     Pe = cal_resist_power(v, 0, 0, 4)
209     fuel_rate = calculate_fuel_rate(n, Pe)
210     # 正确的百公里油耗计算方法
211     fuel_ml_per_km = (fuel_rate * 3600) * 100 / v
212     fuel_consumption_4.append(fuel_ml_per_km / 1000) # L/100km
213 plt.plot(v_range_4, fuel_consumption_4, label=f'4档 (ig=
{gear_ratios[4-1]:.2f})')
214
215 # 计算五档
216 v_range_5 = np.linspace(20, 120, 1000)
217 fuel_consumption_5 = []
218 for v in v_range_5:
219     n = cal_rpm(v, 5)
220     Pe = cal_resist_power(v, 0, 0, 5)
221     fuel_rate = calculate_fuel_rate(n, Pe)
222     # 正确的百公里油耗计算方法
223     fuel_ml_per_km = (fuel_rate * 3600) * 100 / v
224     fuel_consumption_5.append(fuel_ml_per_km / 1000) # L/100km
225 plt.plot(v_range_5, fuel_consumption_5, label=f'5档 (ig=
{gear_ratios[5-1]:.2f})')
226
227 plt.xlabel('车速 (km/h)')
228 plt.ylabel('油耗 (L/100km)')
229 plt.title('等速百公里油耗曲线')
230 plt.legend()
231 plt.grid()
232 plt.show()
233
234
235 # ===== 主函数 =====
236 if __name__ == '__main__':
237     # 计算六工况油耗
238     total_fuel_100km = simulate_six_cycles() / 1.075 * 100 # 将总油耗转换为百公里
油耗
239     print(f"六工况循环百公里油耗: {total_fuel_100km:.2f} L/100km")
240     # 绘制功率平衡图
241     plot_power_balance()
242     # 绘制等速油耗曲线
243     plot_constant_speed_consumption()

```

Fence 9

输出结果:

1 | 六工况百公里油耗: 17.01L/100km

Fence 10

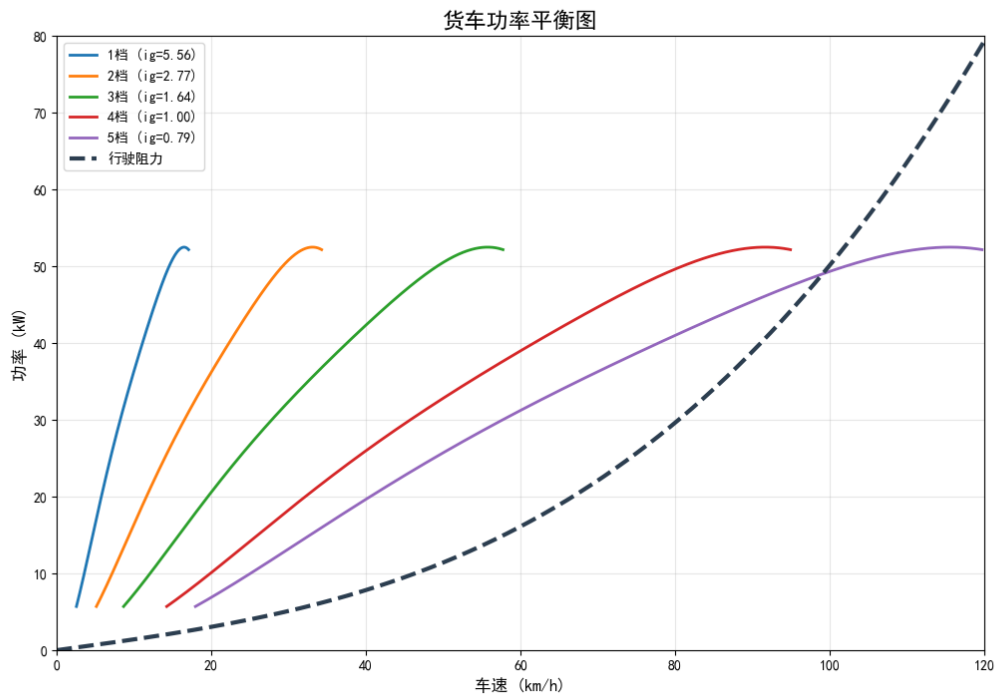


Figure 2

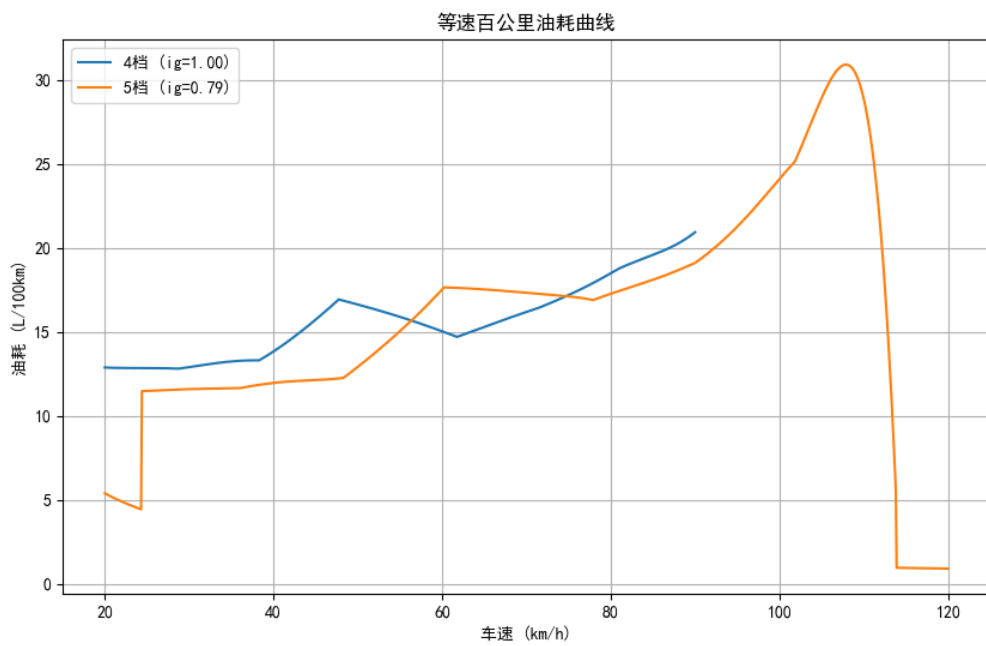


Figure 3

4.3 3.1 不同主减速比的燃油经济性-加速时间计算

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from scipy import integrate
4 import pandas as pd
5
6 plt.rcParams['font.sans-serif'] = ['SimHei']
7 plt.rcParams['axes.unicode_minus'] = False

```

```

8
9 # 发动机参数
10 n_min = 600 # 发动机最低转速
11 n_max = 4000 # 发动机最高转速
12 n_max_power = 3800 # 最大功率转速，假设为3800转，需要从发动机特性确定。
13 n_max_torque = 2500 # 最大扭矩转速，假设为2500转，需要从发动机特性确定。
14
15 # 车辆质量参数
16 m_load = 2000 # 装载质量 (kg)
17 m_empty = 1800 # 整车整备质量 (kg)
18 m_total = 3880 # 总质量 (kg)
19
20 # 速度参数
21 v_max = 85 # 最大车速 (km/h)
22 v_min = 0 # 最小车速 (km/h)
23 v_min_2 = 10 # 二档最低车速
24
25 # 车轮参数
26 r = 0.367 # 车轮半径 (m)
27
28 # 传动系统参数
29 eta_t = 0.85 # 传动系统机械效率
30 f = 0.013 # 滚动阻力系数
31 CdA = 2.77 # 空气阻力系数 × 迎风面积 (m^2)
32
33 # 转动惯量
34 I_f = 0.218 # 飞轮转动惯量 (kg*m^2)
35 I_w1 = 1.798 # 前轮转动惯量 (kg*m^2)
36 I_w2 = 3.598 # 后轮转动惯量 (kg*m^2)
37
38 # 几何参数
39 L = 3.2 # 轴距
40 a = 1.947 # 质心至前轴距离 (满载)
41 h_g = 0.9 # 质心高
42
43 # 其他参数
44 rho = 1.225 # 空气密度 (kg/m^3) 在标准大气条件下
45 Qid = 0.299 # 怠速油耗 (mL/s)
46 fuel_density = 0.74 # 燃油密度 (g/mL)
47 g = 9.81 # 重力加速度
48 G = m_total * g # 重力
49
50 # 变速箱传动比(5档)
51 ig_1_5 = 5.56
52 ig_2_5 = 2.769
53 ig_3_5 = 1.644
54 ig_4_5 = 1.00
55 ig_5_5 = 0.793
56 gear_ratios = [ig_1_5, ig_2_5, ig_3_5, ig_4_5, ig_5_5] # 五档变速器传动比
57 num_gears = 5 # 档位数
58 gear_top = gear_ratios[-1] # 最高档传动比
59 gear_second = gear_ratios[-2] # 次高档传动比
60
61

```



```

62 # ===== 发动机特性参数
63 =====
64 engine_coeffs = pd.DataFrame({
65     'n': [815, 1207, 1614, 2012, 2603, 3006, 3403, 3804],
66     'B0': [1326.8, 1354.7, 1284.4, 1222.9, 1141.0, 1051.2, 1233.9,
1129.7],
67     'B1': [-416.46, -303.98, -189.75, -121.59, -98.893, -73.714, -84.478,
-45.291],
68     'B2': [72.379, 36.657, 14.524, 7.0035, 4.4763, 2.8593, 2.9788,
0.71113],
69     'B3': [-5.8629, -2.0553, -0.51184, -0.18517, -0.091077, -0.05138,
-0.047449, -0.00075215],
70     'B4': [0.17768, 0.043072, 0.0068164, 0.0018555, 0.00068906,
0.00035032, 0.00028230, -0.000038568]
71 })
72
73 # ===== 核心计算函数 =====
74 def interpolate_coefficients(n):
75     """根据转速n插值获取B0-B4系数"""
76     n_values = engine_coeffs['n'].values
77     if n < n_values[0] or n > n_values[-1]:
78         return None
79     idx = np.searchsorted(n_values, n)
80     idx = max(1, min(idx, len(n_values) - 1))
81     n_low, n_high = n_values[idx - 1], n_values[idx]
82     w_low = (n_high - n) / (n_high - n_low)
83     w_high = (n - n_low) / (n_high - n_low)
84     coeffs = w_low * engine_coeffs.iloc[idx - 1, 1:] + w_high *
engine_coeffs.iloc[idx, 1:]
85     return coeffs.values
86
87
88 # 考虑转动惯量的牛二
89 def cal_delta_m(i0, gear):
90     return m_total + (2 * I_w1 + 2 * I_w2 + I_f * i0 ** 2 *
gear_ratios[gear - 1] ** 2) / (r ** 2)
91
92
93 # 速度(km/h)计算发动机转速 (r/min)
94 def cal_rpm(i0, v_kmh, gear):
95     v = v_kmh / 3.6 # km/h → m/s
96     wheel_rps = v / (2 * np.pi * r) # 车轮转/秒
97     return wheel_rps * gear_ratios[gear - 1] * i0 * 60 # 发动机转速
98
99
100 def cal_vkmh(i0, n, gear): # 计算车速
101     n_scaled = n / (i0 * gear_ratios[gear - 1])
102     v = (2 * np.pi * r) * n_scaled / 60
103     v_kmh = v * 3.6
104     return v_kmh
105
106
107 def engine_torque(n):
108     """n: 发动机转速 (r/min)"""

```

```

109     n_scaled = n / 1000
110     Tq = -19.313 + 295.27 * n_scaled - 165.44 * n_scaled ** 2 + 40.874 *
n_scaled ** 3 - 3.8445 * n_scaled ** 4
111     return Tq
112
113
114 def calculate_power(i0, v_kmh, acceleration, slope_angle, gear):
115     """计算发动机功率需求 (kw)"""
116     v = v_kmh / 3.6 # m/s
117     F_roll = G * f * np.cos(slope_angle)
118     F_air = 0.5 * rho * CdA * v ** 2 # 修改后的空气阻力计算
119     F_accel = cal_delta_m(i0, gear) * acceleration
120     F_gradient = G * np.sin(slope_angle)
121     F_total = F_roll + F_air + F_accel + F_gradient
122     return (F_total * v) / (eta_t * 1000)
123
124
125 def calculate_fuel_rate(n, Pe):
126     # 计算燃油消耗率 (mL/s)
127     if Pe <= 0 or n < n_min:
128         return Qid
129     coeffs = interpolate_coefficients(n)
130     if coeffs is None:
131         return Qid
132     B0, B1, B2, B3, B4 = coeffs
133     b = B0 + B1 * Pe + B2 * Pe ** 2 + B3 * Pe ** 3 + B4 * Pe ** 4 #
g/(kw·h)
134     return (b * Pe) / (3600 * fuel_density)
135
136
137 def cal_fuel_time(i0, t, v_start, accel): # 建立在6种工况中油耗(mL/s)与时间的
函数
138     v = v_start + accel * t
139     n = cal_rpm(i0, v, gear=3)
140     Pe = calculate_power(i0, v, accel, 0, gear=3)
141     fuel_rate = calculate_fuel_rate(n, Pe) # 油耗(mL/s)
142     return fuel_rate / 1000 # 转换为L/s
143
144
145 # ===== 六工况油耗计算
=====
146 six_conditions = [
147     {'type': '匀速', 'duration': 7.2, 'v_start': 25, 'v_end': 25, 'accel':
0},
148     {'type': '匀加速', 'duration': 16.7, 'v_start': 25, 'v_end': 40,
'accel': 0.25},
149     {'type': '匀速', 'duration': 22.5, 'v_start': 40, 'v_end': 40, 'accel':
0},
150     {'type': '匀加速', 'duration': 14.0, 'v_start': 40, 'v_end': 50,
'accel': 0.20},
151     {'type': '匀速', 'duration': 18.0, 'v_start': 50, 'v_end': 50, 'accel':
0},
152     {'type': '匀减速', 'duration': 19.3, 'v_start': 50, 'v_end': 25,
'accel': -0.36},
153 ]

```

```

154
155
156 def simulate_six_cycles(i0):
157     total_fuel = 0 # 总油耗 (L)
158     for cond in six_conditions:
159         duration = cond['duration']
160         accel = cond['accel']
161         v_start = cond['v_start']
162         v_end = cond['v_end']
163         fuel = integrate.quad(lambda t: cal_fuel_time(i0, t, v_start,
164             accel), 0, duration)[0]
165         total_fuel += fuel
166     return total_fuel
167
168 def inverse_acceleration(i0, v): # 定义加速度倒数的函数
169     gear = 1
170     ig = gear_ratios[gear - 1]
171     n = (v / 3.6) * 60 * i0 * ig / (2 * np.pi * r)
172     while n > n_max:
173         gear += 1
174         if gear > num_gears: # 如果超过最高档位，则返回无穷大，表示无法加速
175             return np.inf
176         ig = gear_ratios[gear - 1]
177         n = (v / 3.6) * 60 * i0 * ig / (2 * np.pi * r)
178     Tq = engine_torque(n)
179     Tw = Tq * ig * i0 * eta_t # 车轮转矩
180     F_traction = Tw / r # 驱动力
181     v_mps = v / 3.6 # 将 km/h 转换为 m/s
182     Fr = G * f # 滚动阻力
183     Fa = 0.5 * rho * CdA * v_mps ** 2 # 空气阻力
184     F_resistance = Fr + Fa # 总阻力
185     F_net = F_traction - F_resistance
186     delta_m = cal_delta_m(i0, gear)
187     a = F_net / delta_m
188     if a <= 0:
189         return np.inf
190     return 1 / a
191
192
193 def calculate_acceleration_time(i0):
194     time, error = integrate.quad(lambda v: inverse_acceleration(i0, v),
195         v_min_2, v_max) # 修改积分上限到100
196     return time
197
198 if __name__ == '__main__':
199     i0_values = [5.17, 5.43, 5.83, 6.17, 6.33]
200     acceleration_times = []
201     fuel_economies = []
202
203     for i0 in i0_values:
204         # 计算零百加速时间
205         acceleration_time = calculate_acceleration_time(i0)
206         acceleration_times.append(acceleration_time)

```

```

207         # 计算EPA循环工况燃油经济性（这里使用六工况循环作为简化）
208         total_fuel_100km = simulate_six_cycles(i0) / 1.075 * 100 # 将总油
耗转换为百公里油耗
209         fuel_economies.append(total_fuel_100km)
210         print(f"主减速比: {i0:.2f}, 六工况循环百公里油耗:
{total_fuel_100km:.2f} L/100km, 0-100km/h加速时间: {acceleration_time:.2f}
s")
211
212         # 绘制燃油经济性-加速时间曲线
213         plt.figure(figsize=(10, 6))
214         plt.plot(fuel_economies, acceleration_times, marker='o')
215         plt.xlabel('EPA循环工况燃油经济性 (L/100km)')
216         plt.ylabel('10-85km/h加速时间 (s)') # 修改为100km/h
217         plt.title('燃油经济性-加速时间曲线')
218         plt.grid(True)
219
220         # 添加数据点标签
221         for i, i0 in enumerate(i0_values):
222             plt.annotate(f'i0={i0:.2f}', (fuel_economies[i],
acceleration_times[i]), textcoords="offset points", xytext=(5, 5),
ha='center')
223
224         plt.show()
225

```

Fence 11

输出结果:

```

1  主减速比: 5.17, 六工况循环百公里油耗: 15.61 L/100km, 0-100km/h加速时间: 167.68 s
2  主减速比: 5.43, 六工况循环百公里油耗: 15.99 L/100km, 0-100km/h加速时间: 160.90 s
3  主减速比: 5.83, 六工况循环百公里油耗: 17.01 L/100km, 0-100km/h加速时间: 151.92 s
4  主减速比: 6.17, 六工况循环百公里油耗: 17.96 L/100km, 0-100km/h加速时间: 145.93 s
5  主减速比: 6.33, 六工况循环百公里油耗: 18.37 L/100km, 0-100km/h加速时间: 143.71 s

```

Fence 12

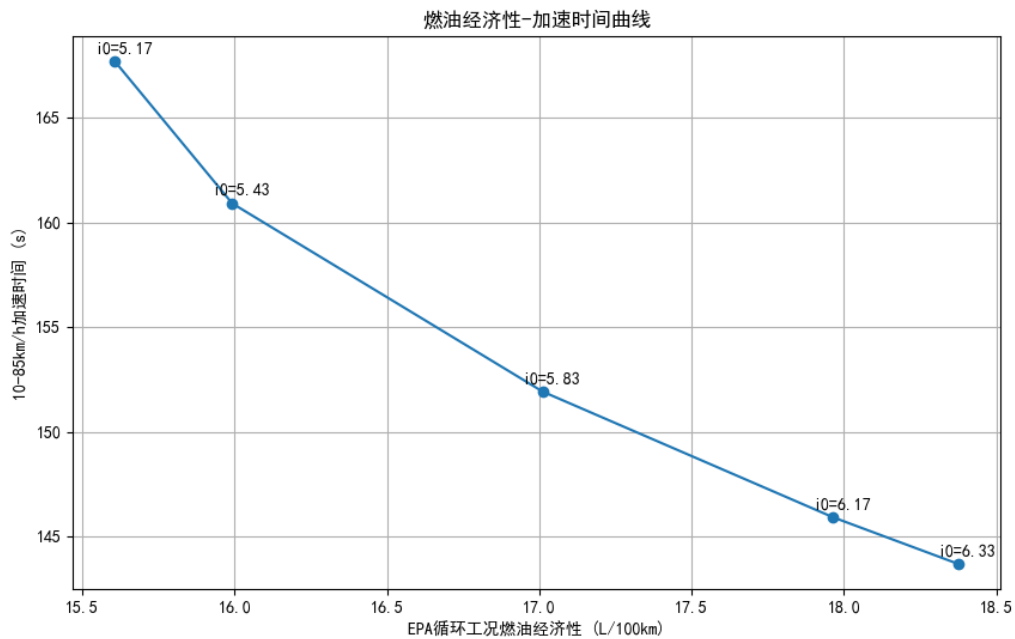


Figure 4

