基于改进粒子群算法和惰行优化的列车运行策略规划

摘要

本文对节能列车运行控制优化策略进行研究，考虑列车牵引、巡航、惰行、制动四种工况，建立了基于时间最优的列车运行策略模型求取列车最短运行时间，建立了基于惰行优化的列车运行策略规划模型求取最低能耗的运行策略，建立分阶段的列车策略规划模型对运过程的突发事故情况及时作出运行策略调整。

针对问题一，基于列车运行模式，建立了**列车运行策略规划模型**。优先考虑牵引、巡航、制动三种工况。首先引入列车受力、有效质量、停车距离精度等约束建立目标规划模型，求得列车最短运行时间为 **200.68s**，程序运行时间为 **0.9s**. 然后通过降低最大运行速度实现运行时间的增长，以列车准时性误差指标最小为目标函数，，调整约束条件，利用引入**随机游走策略**的**粒子群优化算法**进行模型求解。

针对问题二，考虑路况信息以及电机的复杂动态过程，建立了**基于惰行优化的列车策略规划模型**。增加对惰行工况的考虑。首先基于问题一模型，引入反映工况运行安排的 0-1 变量，加入坡度变化、限速、电机牵引制动力约束，求得最短时间为 **209.6s**。然后以最短时间和增长后的时间作为列车计划运行时间，调整约束条件，以能耗最低为目标建立策略规划模型。通过数据分析得到惰行进入 86km/h 限速路段不会超速的最大速度为 **76.572km/h**，和粒子群优化算法求取最小能耗的列车运行优化方案。

针对问题三，考虑突发事故事故影响，建立了**分阶段的列车运行策略规划模型**。首先将运行过程以突发事故为节点划分为两个阶段。第一阶段考虑原计划运行时间和能耗最小求得 2000m 处的运行状态，并作为第二阶段的初始状态，经计算得状态时间为 **129.75s**。对问题二的基于惰行优化的列车运行策略规划模型引入初始状态约束，利用粒子群优化算法求解得到列车运行优化方案。

本文的亮点有：1. 本文采用两级分层优化策略，上层优化工况顺序和切换点，得到最短运行时间，从而将多目标问题转化为单目标问题，在下层优化策略中更好地降低能量消耗；2. 本文相较于一般的列车节能运行策略，引入惰行过程并且通过基于随机游走策略改进的 PSO 算法，重点对惰行时间以及惰行工况的切换点进行优化。

关键词：列车运行策略规划　随机游走策略　惰行优化　粒子群优化

目录

1 问题重述

1.1 问题背景

近年来，随着国内城市轨道交通系统的发展和客流量需求的增加，建设规模不断扩张，运营总能耗也随之增加且逐年增长，城轨系统节能运行已经成为运营企业关注的焦点问题之一。列车牵引能耗是构成城轨系统总能耗的主要目标，而列车牵引能耗主要用于列车运行。优化列车运行策略可以有效降低能耗，不仅有助于减少运营成本，而且可有效降低碳排放，具有显著的经济效益和社会效益。

图 1　问题背景图

1.2 问题提出

问题一：已知站台 A 至站台 B 的间距为 5144.7m，一辆列车的最快运行速度为 100km/h，质量为 176.3t，旋转部件惯性的旋转质量因数 ρ=1.08，最大牵引力为 310KN，最大制动力为 760KN，受到的阻力满足 Davis 阻力方程 f=2.0895+0.0098v+0.006v^2. 建立模型并编写程序，获取最短运行时间和在最短时间基础上分别增加 10s、20s、50s、150s、300s 的列车运行速度-距离、牵引制动力-距离、时间-距离曲线与能量消耗-距离的六组曲线。

问题二：附件一是列车从 XEQ 站到 SMKXY 站的路途中的坡度及限速的变化信息数据，附件二是电机的动态特性与参数并给出了静态电机牵引率与制动再生率的相关数据。基于附件一和附件二，设计能耗最低的可行的列车运行速度轨迹的优化方案，并获取最短运行时间和在最短时间基础上分别增加 10s、20s、50s、150s、300s 的列车运行速度-距离、牵引制动力-距离、时间-距离曲线与能量消耗-距离的六组曲线。

问题三：列车在起点站出发，计划运行 320s 到达终点，在行至 2000m 位置时，被

告知由于突发事故需要延迟 60s 到达终点，设计在保持列车节能公正常运行条件下最快进行调整的优化速度轨迹方案，并获取列车运行速度-距离、牵引制动力-距离、时间-距离曲线与能量消耗-距离的曲线。

2 模型假设

1. 假设列车为单质点模型，不考虑列车长度;
2. 假设列车耗能主要由电机牵引力做功耗能，不考虑列车中其他用电设施用电耗能;
3. 假设列车司机采取各种列车运行策略的反应时间可忽略不计;
4. 假设问题二坡度在每个路段之间保持不变.

3 符号说明

符号	说明
F_Q	列车电机的牵引力
F_Z	列车机械制动部件的制动力
f	列车运行过程中受到的阻力
m	列车有效质量
s	列车运行路程
v	列车运行速度
ρ	旋转质量因数
θ	坡度
T	列车计划运行时间
E	列车运行过程的能耗

注：表中未列出及重复的符号均以首次出现处为准，并且在计算时均转化为国际单位制进行计算。

4 问题一模型的建立及求解

4.1 问题一分析

问题一是一个优化问题，主要分为两小问，第一小问要求列车在站台 A、B 之间的运行时间最短，第二小问要求在最短运行时间上增长时间，求取列车驾驶策略。首先经过查找相关资料，了解了列车的运行模式，然后我们从列车运行工况、速度、受力等方面考虑来实现运行时间最小化及调整。

针对第一小问，本文为保证时间最短，仅考虑牵引、巡航、制动三个工况，依据题目要求、结合动力学方程、考虑停车精度引入相关约束，建立列车运行策略规划模型进行求解。

针对第二小问，首先通过分析，通过减小牵引力、增大制动力会增加运行风险，因此决定通过减小最大运行速度实现目标。基于第一小问模型，调整受力约束条件，为减小风险将牵引力与制动力都设为最大值，以列车准时性的误差指标最小化为目标建立优化模型。通过粒子群优化算法求解，并为了避免局部最优解，加入随机游走策略，对算法做出改进。

问题一思路图如下。

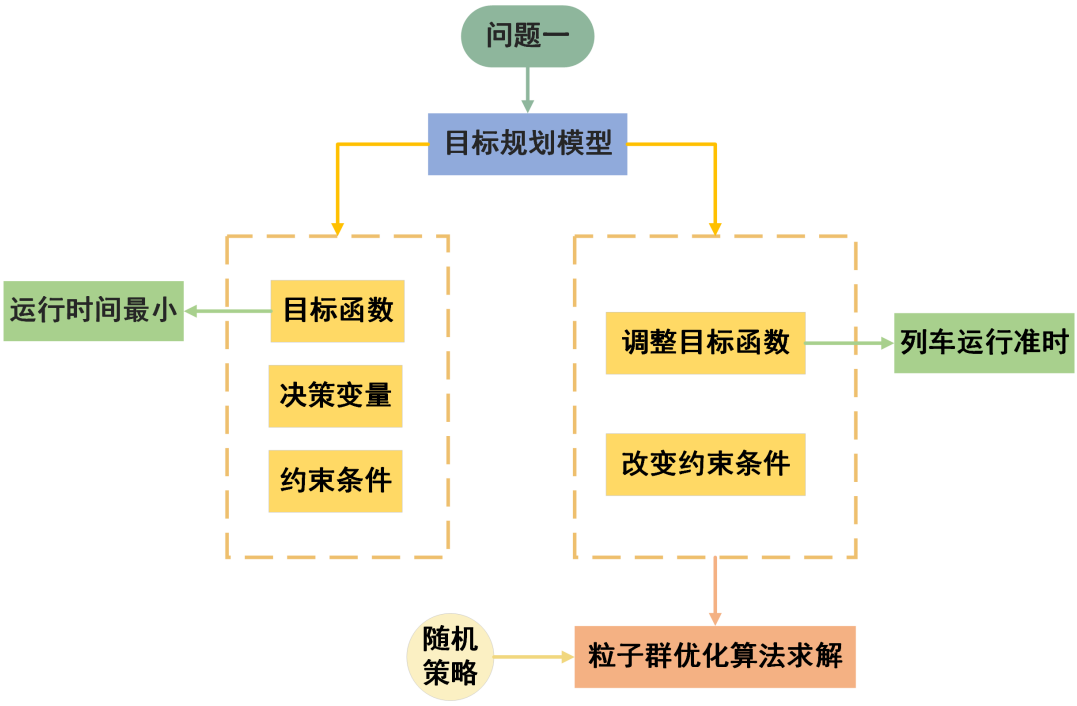


图 2　问题一思路图

4.2 列车运行模式

列车的运行工况可分为牵引、惰行、巡航、制动四类。在牵引的工况下，存在牵引力和阻力作用，牵引力克服阻力做功，目的使列车启动；在惰行的工况下，仅存在阻力作用，牵引力不做功；在巡航的工况下，同样是存在牵引力和阻力共同作用，并且牵引力的大小等于阻力的大小，列车所受合力为 0，目的使列车作匀速运动；在制动的工况下，存在制动力和阻力作用，目的使列车速度减为 0。本文假设在列车的运行过程中，仅牵引力做功会产生能耗，因此仅牵引工况和巡航工况会产生能量消耗。

列车的站间运行过程分为启动、中间、停车过程。本文为模型简洁，不考虑过于复杂的多工况组合模式。假设启动过程仅存在牵引工况，中间过程主要通过惰行、巡航、制动三个工况进行调速，停车过程仅通过制动工况进行减速。

综上分析，列车的运行模式如下图所示。

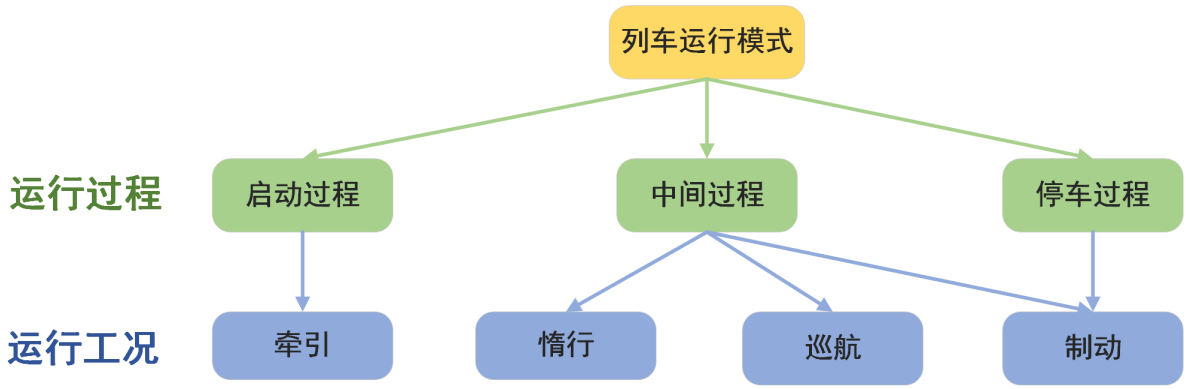


图 3　列车运行模式

4.3 基于时间最优的列车运行策略规划模型

第一小问要求列车从站台 A 出发到达站台 B 时间最短，为此本文建立目标规划模型，寻找列车最优驾驶策略。

4.3.1 决策变量

为方便研究，本文将列车看作均匀分布刚性质量带。为保证列车运行时间最短，应让列车的启动、中间、停车三个运行过程中都尽可能实现以最高速度匀速运动或最大加速度变速运动[?][?]。因此，在列车的运行过程中应主要优先考虑牵引、巡航、制动三个工况。本文用 t 记录列车的运行时间，运行过程的三个阶段具体情况如下：

$$t=\begin{cases}\text{启动牵动阶段}, 0 \le t < t_1;\\ \text{中间巡航阶段}, t_1 \le t < t_2;\\ \text{停车制动阶段}, t_2 \le t \le t_3.\end{cases} \tag{1}$$

其中 t_1、t_2、t_3 分别表示列车的启动阶段、中间阶段、停车阶段的结束时刻。

4.3.2 目标函数

本题目标是列车运行时间最短，即要求列车到达 B 站台完成停车操作的时刻最早。

$$\min \Phi = t_3. \tag{2}$$

其中 Φ 表示列车从 A 站台到达 B 站台的运行时间。

4.3.3 约束条件

1) 受力约束：在实际中，列车运行过程中受力情况较复杂，作用于列车的力的数量、大小、方向难以全部清楚估计。本文为简化列车运行规划模型，仅考虑影响列车速度的三种作用力：列车电机的牵引力 F_Q，机械制动部件的制动力 F_Z，列车受到的阻力 f。具体受力分析如下图示意。

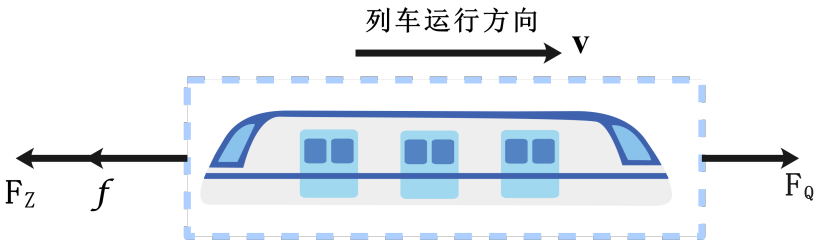


图 4　列车受力分析图

基于上述列车运行模式分析，不同运行阶段列车的受力情况不同，每个阶段列车所受合力情况如下

$$F(t) = \begin{cases} F_Q(t) - f(v), & 0 \le t < t_1 \\ 0, & t_1 \le t < t_2 \\ -F_Z(t) - f(v), & t_2 \le t \le t_3 \end{cases} \tag{3}$$

其中 $F_Q(t)$ 列车的牵引力大小随时间的变化情况；$F_Z(t)$ 列车的制动力大小随时间变化情况；$f(v)$ 列车受到的阻力大小随车速的变化情况，根据题目，两者函数关系满足 Davis 阻力方程即

$$f = 2.0895 + 0.0098v + 0.006v^2 \tag{4}$$

并且根据题目要求，本文规定了牵引力与和制动力的上限，约束如下：

$$\begin{cases} 0 \leqslant F_Q(t) \leqslant F_{Q\max} \\ 0 \leqslant F_Z(t) \leqslant F_{Z\max} \end{cases} \tag{5}$$

其中 $F_{Q\max}$ 表示列车电机的最大牵引力即 310KN；$F_{Z\max}$ 表示机械制动部件的最大制动力即 760KN。

2) 质量约束：考虑到列车存在旋转部件惯性的旋转质量因数[?]，假设题目中所给列车质量为空重质量，则列车有效质量为

$$m = m_l \times \rho \tag{6}$$

其中 m 表示列车的有效质量；m_l 表示列车的空重质量即 176.3t；ρ 表示列车的旋转部件惯性的旋转质量因数即 1.08.

3) 关系约束：根据动力学方程，路程与速度需满足如下等式约束：

$$\begin{cases} F(t) = m\frac{d^2s}{dt^2} \\ v(t) = \int_0^t \frac{d^2s}{dt^2}dt \end{cases} \tag{7}$$

4) 速度约束：根据题目的单列车运行过程可知列车初始与终止速度都为 0，故对此约束：

$$v(0) = v(t_3) = 0 \tag{8}$$

并且根据题目要求，本文规定了运行的速度上限，约束如下：

$$0 \le v(t) \le v_{\max} \tag{9}$$

其中 $v_{\max}$ 表示列车运行的速度上限即 100km/h.

5) 距离约束：列车运行需满足精准停靠在终点站台 B 的要求，但考虑到实际情况，停车的精度性可以在小界限内上下浮动，故对此约束：

$$|s(t_3) - s_{AB}| < \varepsilon \tag{10}$$

其中 s_{AB} 表示站点 A 和站点 B 之间的距离；ε 表示列车的停车精度，经查找资料，本文设置为 0.2m.

6) 能耗约束：在运行过程中牵引力做功会产生能量消耗，计算公式如下

$$E = \int_0^{s_{AB}} F_Q(s)\,ds = \int_0^{t_3} F_Q(t)\,v(t)\,dt \tag{11}$$

其中 E 表示能量消耗。

4.3.4 模型综述

根据以上分析，我们建立的基于站间最短运行时间的列车驾驶策略规划模型如下。

$$\min F_Q = t_3 \tag{12}$$

$$
s.t.\begin{cases}
F(t)=F_Q(t)-f(v), & if \quad 0\leqslant t<t_1,\\
F(t)=0, & if \quad t_1\leqslant t<t_2,\\
F(t)=-F_Z(t)-f(v), & if \quad t_2\leqslant t\leqslant t_3,\\
m=m_l\times\rho,\\
F(t)=m\frac{d^2s}{dt^2},\\
v(t)=\int_0^t\frac{d^2s}{dt^2}dt,\\
v(0)=v(t_3)=0,\\
|s(t_3)-s_{AB}|<\varepsilon,\\
E=\int_0^{t_3}F_Q\left(t\right)v\left(t\right)dt,\\
0\leqslant v(t)\leqslant v_{\max},\\
0\leqslant F_Q(t)\leqslant F_{Q\max},\\
0\leqslant F_Z(t)\leqslant F_{Z\max}.
\end{cases}
\tag{13}
$$

4.4 基于改进的粒子群算法的列车运行策略规划模型

4.4.1 调整约束条件

基于第一小问，第二小问要求在最短运行时间上增长时间到达 B 站台。由于站台 A、B 之间的路程固定，所以只能通过改变运行速度来实现运行时间的延长。本文初步从减小牵引力、增大制动力、降低最大运行速度三个方面考虑调整运行速度。

由于题目中规定列车制动力存在最大值上限，并且减小牵引力、增大制动力会造成列车加速度减小，列车运行存在在终点站台 B 速度未能减为 0 而无法正常停车的风险，在实际中应用会具有极大隐患。因此, 本文不予考虑减小牵引力、增大制动力两种实现途径。并且为规避风险，在后续求解过程中假设牵引力与制动力都为最大值。即

$$
\begin{cases}
F\left(t\right)=F_{Q\max}-f\left(v\right), & if \quad 0\leq t<t_1;\\
F\left(t\right)=0, & if \quad t_1\leq t<t_2;\\
F\left(t\right)=-F_{Z\max}-f\left(v\right), & if \quad t_2\leq t\leq t_3;
\end{cases}
\tag{14}
$$

综上分析，本文选取通过降低最大运行速度方式来实现列车运行时间的增加。改变速度上限约束。

$$
0\leq v\left(t\right)\leq v_i^{'}<v_{\max},\ \ i=1,2,...,5 \tag{15}
$$

其中 $v_i^{'}$ 表示第 i 次调整时间的运行过程中的速度上限。

4.4.2 改变目标函数

考虑现实情况，列车运行时间无法做到分秒不差，因此本文引入列车运行的准时性指标，用从起点站到终点站并停稳的运行时间与计划时间的差值来衡量。为保证列车尽

可能达到规定时间，要求列车准时性指标最小化，依据此建立目标函数如下

$$\min \delta = |t_3 - \tau_i|, \ \ i = 1, 2, ..., 5. \tag{16}$$

其中 τ_i 表示增长后的运行时间，分别为在最短时间基础上增加 10s、20s、50s、150s、300s；δ 表示列车准时性指标大小。

4.4.3 模型综述

根据以上分析并基于第一小问规划模型，我们针对每个的列车运行增长时间，建立的列车驾驶策略优化模型如下。

$$\min \delta = |t_3 - \tau_i|, \ \ i = 1, 2, ..., 5. \tag{17}$$

$$s.t. \begin{cases} F(t) = F_{Q\max} - f(v), & if \quad 0 \le t < t_1, \\ F(t) = 0, & if \quad t_1 \le t < t_2, \\ F(t) = -F_{Z\max} - f(v), & if \quad t_2 \le t \le t_3, \\ m = m_l \times \rho, \\ F(t) = m\frac{d^2s}{dt^2}, \\ v(t) = \int_0^t \frac{d^2s}{dt^2} dt, \\ Q = \int_0^{t_3} F_Q(t)\, v(t)\, dt, \\ v(0) = v(t_3) = 0, \\ |s(t_3) - s_{AB}| < \varepsilon, \\ 0 \le v(t) \le v_i' < v_{\max}. \end{cases} \tag{18}$$

4.5 引入随机游走策略的粒子群算法求解

基于上述规划模型，本文选取粒子群优化算法求解，并为了避免陷入局部最优解，在算法求解过程中引入随机游走策略[?]。

粒子群算法 (particle swarm optimization,PSO) 常用于优化问题求解。算法中每个粒子都代表问题的一个潜在最优解，具有适应度值、速度和位置三项指标特征。适应度值由适应度函数计算得到，其值的好坏决定了粒子的优劣；速度决定了粒子移动的方向和距离；位置表示粒子在解空间的位置。在求解过程中，粒子会不断随自身及其他粒子的移动经验即通过跟踪个体极值 Pbest 和群体极值 Gbest 更新个体位置，进行动态调整。从而实现个体在可解空间中的寻优。

本文在粒子群优化算法中引入随机游走策略可以通过为粒子增加随机性和多样性来增强搜索能力，希望在解空间中更全面地探索，尽可能避免陷入局部最优解。主要通过在速度更新和动态调整位置过程中添加随机扰动来实现。

在速度更新时，为每个粒子生成一个随机扰动项。随机扰动项可以由随机生成方向和幅度来表示，方向可以从概率分布中随机采样，幅度可以调节随机扰动的大小。更新公式为

$$v_i\left(t+1\right)=w\times v_i\left(t\right)+c_1\times r_1\left(Pbest_i-\chi_i\left(t\right)\right)+c_2\times r_2\left(Gbest-\chi_i\left(t\right)\right)+r_3\times d \quad (19)$$

其中 $v_i(t)$ 表示粒子 i 在时间步 t 的速度向量；$\chi_i(t)$ 表示粒子 i 在时间步 t 的位置向量；w 表示惯性权重，控制粒子的惯性效果;$c1$、$c2$ 表示加速系数，用于控制个体经验项和群体协作项的权重；d 表示随机扰动项，可以使粒子在解空间中获得随机性的移动；r_1、r_2、r_3 表示随机数。

在动态调整位置时，将随机扰动项与个体经验项、群体协作项相结合，用于更新粒子速度和位置，并且可以通过调节随机扰动项的权重来控制随机游走策略的影响程度。更新公式为

$$\chi_i(t+1)=\chi_i(t)+v_i(t+1) \quad (20)$$

其中 $\chi_i(t+1)$ 表示粒子 i 在时间步 $t+1$ 的位置向量；$v_i(t+1)$ 表示粒子 i 在时间步 $t+1$ 的速度向量。

综上介绍，引入随机游走策略的粒子群算法的实现步骤如下图所示

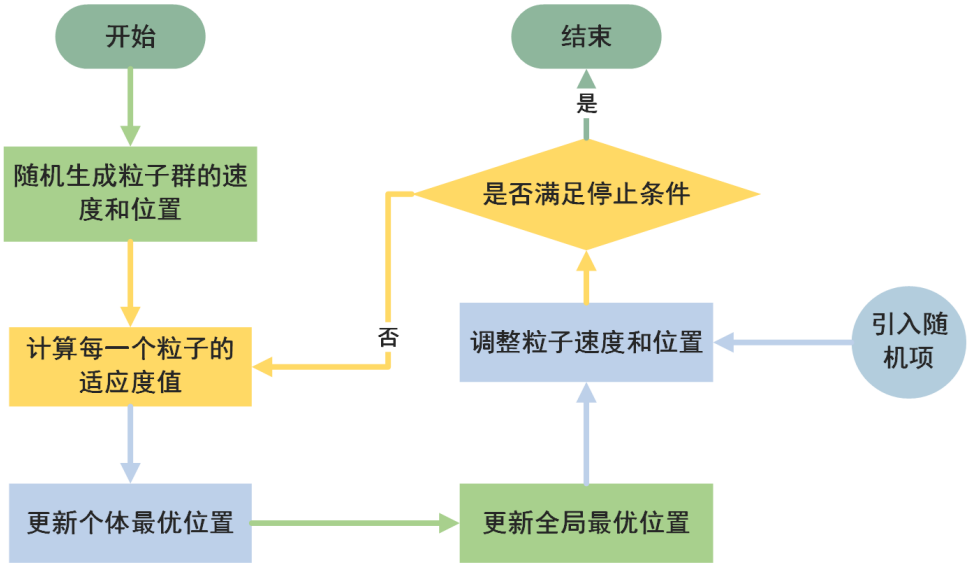


图 5　引入随机游走策略的粒子群算法的主要步骤

4.6 结果分析

本文在利用算法的求解过程中，将时间步长设置为 10^{-5}，程序总运行时间约为 **0.9s**，时间复杂度较小，简单高效。当增长运行时间时，准确性指标都小于 0.01s。经查阅相关资料，通常规定实际运行时间与计划时间的差值不能超过规定时间的 5%. 本文五种

增长时间情况均满足该条件，可认为均正点到达，程序运行得到的最优解精确度较高。利用改进的粒子群优化算法求解, 计算在最短时间基础上延长 10s 时间的 PSO 迭代收敛图如下所示，在迭代 36 次左右可以找到最优解。

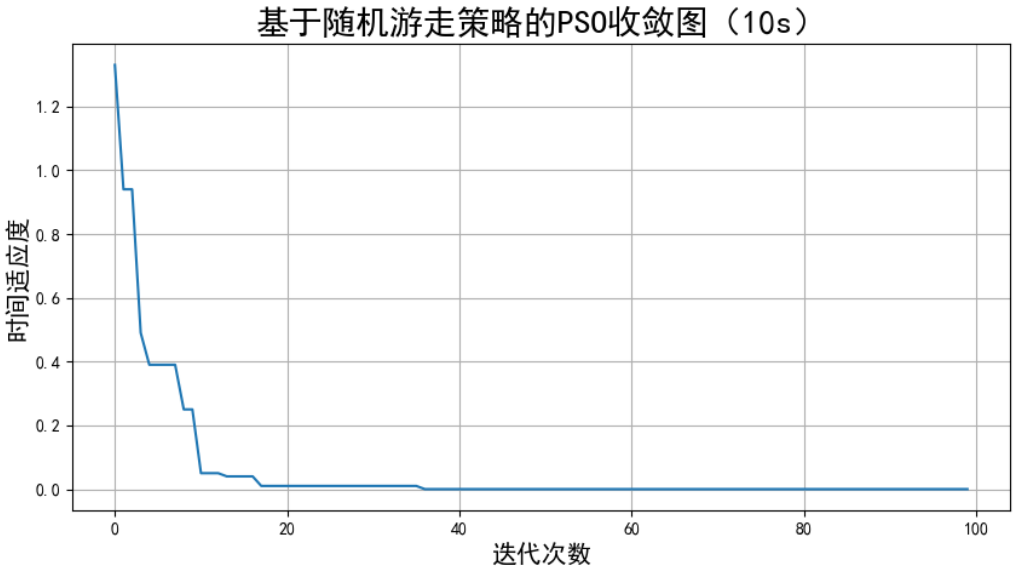


图 6　PSO 迭代收敛图（10s）

表 1　列车站间运行情况

运行时间	牵引 (s)	巡航 (s)	制动 (s)	总时间（s）	总能耗（KJ）	最大速度 (km/h)
最短时间	17.07	176.64	6.97	200.68	107792.16	100.01
增长 10s	14.97	187.98	15.35	209.06	93297.65	94.76
增长 20s	14.21	198.93	25.216	218.94	85249.78	89.95
增长 50s	12.31	231.69	55.306	249.03	66935.82	77.92
增长 150s	8.58	337.09	3.5	349.17	38374.17	54.31
增长 300s	5.93	490.77	2.42	499.12	24139.22	37.54

得到结果如上表所示。从站台 A 到站台 B 列车最短运行时间为 **200.68 s**，通过以最大牵引力启动，达到最大速度巡航、以最大制动力停车三个阶段实现。最终绘制的列车运行过程的速度-距离曲线、牵引制动力-距离曲线、时间-距离曲线与能量消耗-距离曲线如下所示。

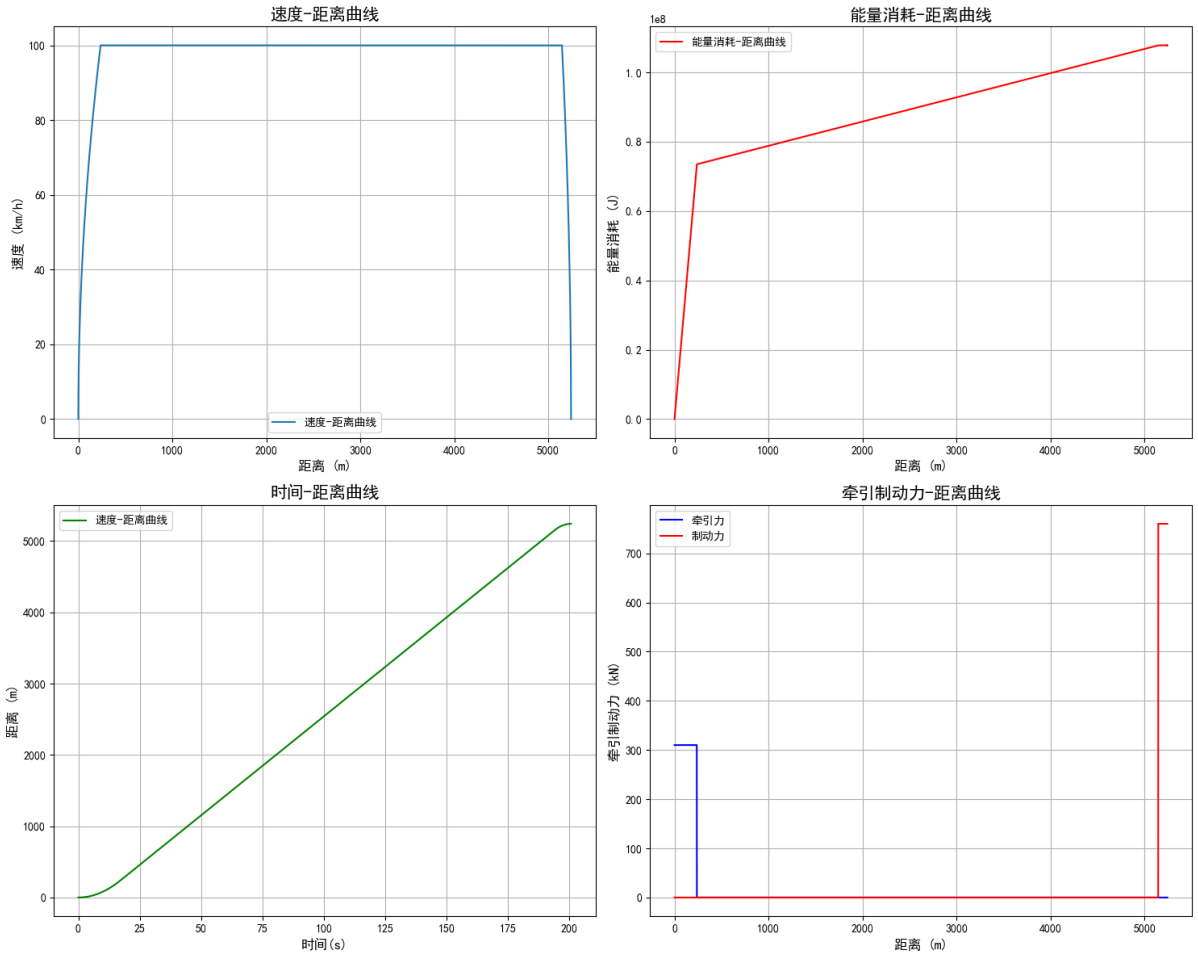


图 7　最短时间下列车运行曲线图

在最短运行时间上分别增加 10s、20s、50s、150s、300s 到达站台 B 的 5 组曲线图可见附录 A。由结果分析可知，在采取牵引-巡航-制动的工况模型时，虽然可以以更快的速度到达站台，但代价是增大了列车总能耗，这是实际情况下需要认真考虑的问题。

5 问题二模型的建立与求解

5.1 问题二分析

问题二仍是一个优化问题问题，相较于问题一，增加对站间的路况信息以及电机的复杂动态过程的考虑。本文将问题分为两部分进行求解。

第一小问，基于问题一的站间最短运行时间的目标规划模型，引入 0-1 决策变量来反映在列车运行过程中各工序的分配情况，调整受力约束、关系约束，增加坡度约束、工况约束、限速约束，以列车运行时间最短作为目标函数，建立基于惰行优化的列车运行策略规划模型进行求解。

第二小问，将最短时间与和增加 10s、20s、50s、150s、300s 的时间分别作为列车计

划运行时间，以能耗最小作为目标函数建立基于惰行优化的列车运行策略优化模型，并通过数据分析确定每一组的工况运行安排情况，利用粒子群优化算法求解得到可行的速度轨迹。

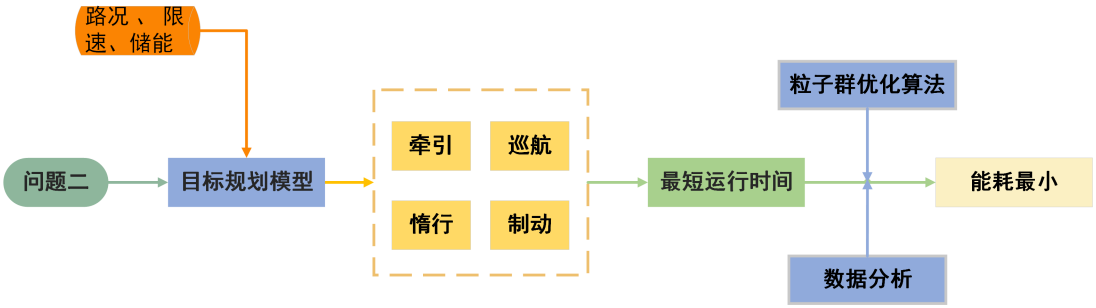


图 8　问题二思路图

5.2 基于时间最优的列车运行策略规划模型

基于问题一，问题二加入了对路段坡度、路段限速、电机动态性[?]、制动储能的考虑[?][?]，建立目标规划模型。

5.2.1 决策变量

欲实现时间最短，需要合理安排四个工况的运行[?]。对此，本文引入四个 0-1 变量。

$$x_i(t)=\begin{cases}0, & t\text{时刻列车未以工况}i\text{运行}\\ 1, & t\text{时刻列车正以工况}i\text{运行}\end{cases} \tag{21}$$

其中 i=1，2，3，4 分别代表牵引、巡航、惰行、制动工况；$x_i(t)$ 表示 t 时刻的工况运行的情况。

路段分为上坡路段与下坡路段，对此，本文引入变量 σ 表示路短的坡度情况。

$$\sigma(s)=sgn(\theta). \tag{22}$$

其中 θ 表示路程为 s 处的坡度。

5.2.2 目标函数

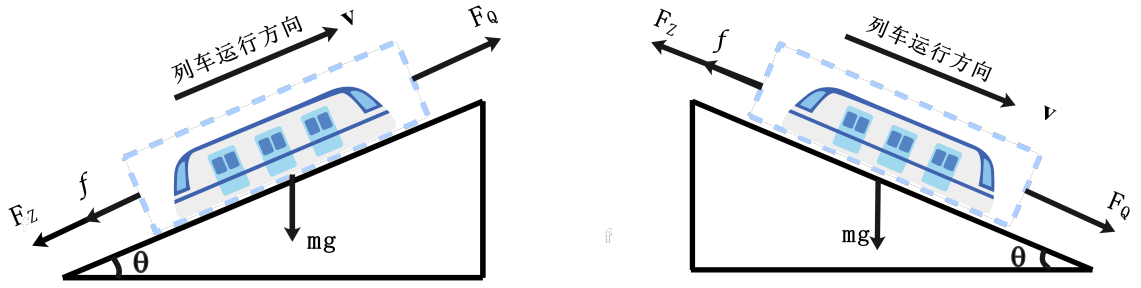
根据题目要求，以时间最短为目标函数。

$$\min T \tag{23}$$

其中 T 表示从站台 A 到站台 B 的列车计划运行时间。

5.2.3 约束条件

1) 受力约束：由于存在坡度，本文重新考虑列车的受力情况，影响列车运行的力有自身重力、运行时受到的阻力 f、牵引力 F_Q、制动力的作用 F_Z。上坡和下坡的受力分析如下图示意。



(a) 列车上坡受力分析图　　(b) 列车下坡受力分析图

图 9　列车受力分析图

列车牵引力

根据附件二信息，列车的牵引力由电机提供，电机的运行范围分为恒转矩区和弱磁区两个阶段，不同阶段的牵引力大小不同。计算公式如下

$$F_Q = \begin{cases} F_{Q1}, & 0 \le v \le \nu_Q^{tra} \\ F_{Q1} \times \frac{\nu_Q^{tra}}{v}, & v_Q^{tra} \le v \end{cases} \tag{24}$$

其中 F_{Q1} 表示电机牵引力即 310KN；v_Q^{tra} 表示制动的恒转矩与恒功率区切换速度点即 10m/s.

列车制动力

经查找相关资料，列车制动方式主要是由电机制动和空气制动两种相混合的方式。列车的动车既存在电机制动又存在空气制动，而其拖车只存在空气制动。并且在一般情况下，列车为了实现能量再利用会优先施加电机制动，当电机制动无法满足列车制动需求时，才会用动车提供空气制动。

为了简化模型，本文假设研究的列车不存在拖车。列车制动力分配规则：在最初制动过程中，仅存在牵引电机提供的制动力；当制动力达到 260KN 时，用空气制动补充；当达到械制动部件的最大制动力即 760KN 时，制动力达到上限不再增加。故对此约束：

$$F_Z = \begin{cases} F_{Z1} + F_{Z2}, & 0 \le v \le \nu_Z^{tra} \\ F_{Z1} \times \frac{\nu_{FZ}^{tra}}{v} + F_{Z2}, & v_Z^{tra} \le v \end{cases} \tag{25}$$

$$0 \leqslant F_Z(t) \leqslant F_{Z\max} \tag{26}$$

其中 F_{Z1} 表示再生电机制动力即 260KN；F_{Z2} 表示空气制动力，随着列车制动需求变化而变化；v_Z^{tra} 表示制动的恒转矩与恒功率区切换速度点即 17m/s.

2) 坡度约束：附件一还规定了每一个路段的坡度大小，考虑实际情况，本文假设路段间的坡度是均匀变化的。据此路段 $s_i \le s \le s_{i+1}$ 的坡度大小应满足如下等式约束：

$$\theta(s)-\theta(s_i)=\frac{\theta(s_{i+1})-\theta(s_i)}{s_{i+1}-s_i}(s-s_i) \quad i=0,1,...,19. \tag{27}$$

其中 s_i 至 s_{i+1} 表示起始站台到终点站台的第 i 个路段，$\theta(s)$ 表示路程为 s 出的坡度。

3) 关系约束：根据牛顿第二定律，各个工况下的加速度的计算公式为

上坡：

$$a(t)=\begin{cases}\frac{F_Q(v)-f(v)-mg\times\theta(s)}{m}, & x_1(t)=1\\ 0, & x_2(t)=1\\ \frac{-f(v)-mg\times\theta(s)}{m}, & x_3(t)=1\\ \frac{-F_Z(v)-f(v)-mg\times\theta(s)}{m}, & x_4(t)=1\end{cases} \tag{28}$$

下坡：

$$a(t)=\begin{cases}\frac{F_Q(v)-f(v)+mg\times\theta(s)}{m}, & x_1(t)=1\\ 0, & x_2(t)=1\\ \frac{-f(v)+mg\times\theta(s)}{m}, & x_3(t)=1\\ \frac{-F_Z(v)-f(v)+mg\times\theta(s)}{m}, & x_4(t)=1\end{cases} \tag{29}$$

其中 $a(t)$ 表示加速度随时间的变化情况；g 为重力加速度，本文取值为 $9.81m/s^2$；由于坡度值极小，可近似看作 $\sin\theta \approx \tan\theta \approx \theta$.

根据动力学方程，路程与速度满足如下等式约束：

$$\begin{cases} v(t)=\int_0^t a(t)\,dt \\ s(t)=\int_0^t v(t)\,dt \end{cases} \tag{30}$$

4) 工况约束：为保证模型的简洁性，假设每一运行时刻只能存在一种运行工况，不存在多种工况同时运行的情况，故对此约束：

$$\sum_{i=1}^{4} x_i(t)=1 \tag{31}$$

并且，由于列车电机在转换工况时需要一定时间，而且为保证列车的平稳性与舒适度，应该避免频繁地进行工况转换，遵循运行工况保持原则，即当列车转换到一个新的工况时，应在该工况下维持一段时间。

$$x_i(t)-x_i(t+\xi)=0, \forall\xi\in[0,\Delta t] \tag{32}$$

其中 Δt 表示一个工况的最短维持时间；ξ 表示在时间段 $[0,\Delta t]$ 的任意时刻。

在运行最开始时刻列车必须借助牵引工序来启动，借助制动工序来停车，因此需满足如下等式约束

$$x_1(0)=1 \quad x_4(T)=1 \tag{33}$$

5) 限速约束：附件一规定了每一个路段的最大运行速度, 故对此约束：

$$v\left(t\right) \leq v^{'}\left(s\right) \tag{34}$$

其中 $v^{'}\left(s\right)$ 表示路程 s 处的限速大小。

5.2.4 模型综述

根据以上分析，建立考虑坡度与限速一最短运行时间为目标函数的规划模型如下：

$$\min T \tag{35}$$

$$s.t.\begin{cases} F_Q = F_{Q1}, & if \quad 0 \leq v \leq \nu_Q^{tra} \\ F_Q = F_{Q1} \times \frac{\nu_Q^{tra}}{v}, & if \quad v_Q^{tra} \leq v \\ F_Z = F_{Z1} + F_{Z2}, & if \quad 0 \leq v \leq \nu_Z^{tra} \\ F_Z = F_{Z1} \times \frac{\nu_{FZ}^{tra}}{v} + F_{Z2}, & if \quad v_Z^{tra} \leq v \\ 0 \leqslant F_Z\left(t\right) \leqslant F_{Z\max} & \\ a\left(t\right) = \frac{T(v) - f(v) - mg \times \theta(s) \times \sigma(s)}{m}, & if \quad x_1\left(t\right) = 1 \\ a\left(t\right) = 0, & if \quad x_2\left(t\right) = 1 \\ a\left(t\right) = \frac{-f(v) - mg \times \theta(s) \times \sigma(s)}{m}, & if \quad x_3\left(t\right) = 1 \\ a\left(t\right) = \frac{-B(v) - f(v) - mg \times \theta(s) \times \sigma(s)}{m}, & if \quad x_4\left(t\right) = 1 \\ \theta\left(s\right) - \theta\left(s_i\right) = \frac{\theta(s_{i+1}) - \theta(s_i)}{s_{i+1} - s_i}\left(s - s_i\right) \quad i = 0, 1, ..., 19, & \\ \sum_{i=1}^{4} x_i\left(t\right) = 1, & \\ v\left(t\right) = \int_0^t a\left(t\right) dt, & \\ s\left(t\right) = \int_0^t v\left(t\right) dt, & \\ x_i(t) - x_i(t + \xi) = 0, \forall \xi \in [0, \Delta t], & \\ v\left(t\right) \leq v^{'}\left(s\right), & \\ x_1(0) = 1, x_4(T) = 1, x_i(t) = \{0, 1\}. & \end{cases} \tag{36}$$

5.2.5 求解及结果分析

基于上述模型，利用引入随机游走策略的粒子群优化算法求解得到最短运行时间为 **209.6**s, 列车各阶段的工况运行情况如下。

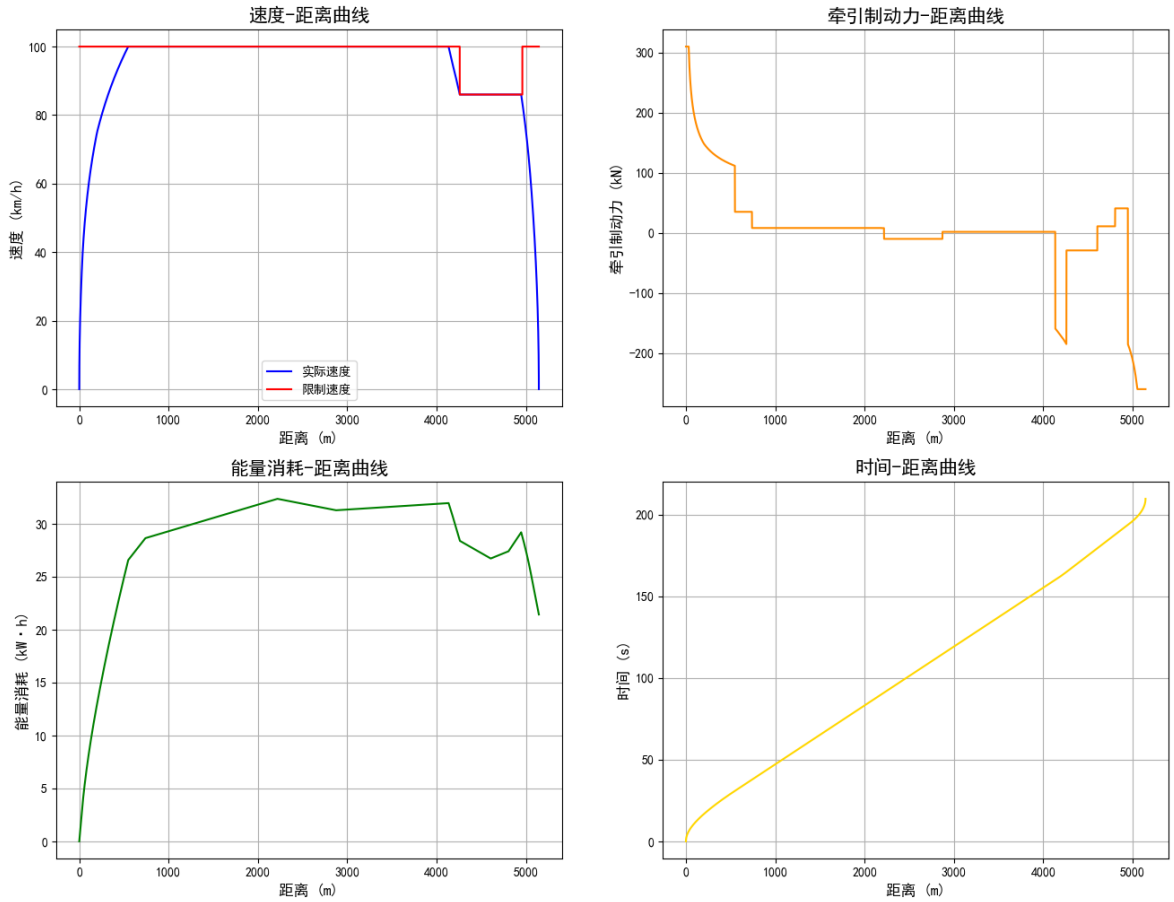


图 10　最短时间下列车运行曲线图

本题的限速组合是低—高—低型限速顺序组合方式。为了保证运行时间最短，未运行惰行工况，并从动力学考虑，保证列车变速时加速度最大化、匀速时速度最大化。从上图中也可以看出列车以牵引—巡航—制动—巡航—制动进行工况分配安排。在行程起点采用最大牵引力进行列车启动，以保证能够在最短时间内达到最大速度即 100m/s；当列车行驶至低限速阶段起点时，最大运行速度要求由 100m/s 变为 86m/s，列车为降低速度进行制动；当速度降至 86m/s 时为保证时间最短再次进行巡航；在接近终点站，采用最大制动力进行停车。

5.3 基于惰行优化的列车运行策略规划模型

给定列车计划运行时间，求取能耗最小的运行策略，计划运行时间为最短时间及在最短运行时间上分别增加 10s、20s、50s、150s、300s 即 209s、219s、229s、259s 、359s 、509s. 根据计划的时间运行，要求能耗最小化，为此本文建立基于惰行优化的列车运行策略规划模型[?]，求得可行的速度轨迹。

5.3.1 增加约束条件

能耗约束：考虑到电机牵引制动过程具有转换效率，且列车制动时会将一定比例的能量储存至储能装置中。因此列车的能耗为牵引力做功与制动存储的能量的差值[?]。据此作如下等式约束：

$$E(t)=\frac{\int_0^t[x_1(t)+x_2(t)]\times F_Q(v)\times v(t)\,dt}{\eta_Q}-\eta_Z\times\int_0^t x_4(t)\times F_{Z1}(v)\times v(t)\,dt \quad (37)$$

其中 η_Q 表示电机牵引效率因子即 0.9，η_Z 表示电机再生制动效率因子即 0.6.

时间约束：与问题一第二小问类似，增加列车准时性衡量指标：

$$|T_i-T_i^{'}|<\varpi,\ \ i=1,2,...,5. \quad (38)$$

其中 T_i 表示列车计划运行时间，分别为在最短时间基础上增加 10s、20s、50s、150s、300s 后的时间；$T_i^{'}$ 表示列车实际运行时间；ϖ 表示允许的时间误差项。

5.3.2 改变目标函数

根据题目列车在运行过程中能耗越低越好，本文以在规定时间内运行条件下能耗最小为目标函数。

$$\min E(T) \quad (39)$$

其中 $E(T)$ 表示列车的总能量损耗。

5.3.3 模型求解

由于在模型的求解过程存在大量随机不确定因素，本文综合数据分析和粒子群优化算法两种方式进行求解。首先通过数据分析根据列车计划运行时间确定合适的工况运行安排，然后利用粒子群求解在该安排和时间约束条件下能耗最小的运行策略。

数据分析：

启动过程和停车过程的牵引和制动工况已确定。只需确定中间过程的工况安排即可确定列车的运行情况。经分析，列车运行的中间过程应为惰行、巡航、制动三个工况的组合模式。

由于惰行工况并不产生能量消耗，为了节能，在满足附件二限速条件下，在列车运行过程中应尽可能维持惰行工况，并减小牵引工况运行时间。

由于路段存在限速条件，在到达新的限速段和终点时，应实时判断是否最大制动力情况下能将速度降到目标值。

在中间运行过程中，上坡路段仅可能做减速运动或匀速运动，速度不可能增加。因此，在保证初始速度满足限速的条件下，上坡阶段不会存在超速的风险。而下坡路段情

况不同，列车会进行加速。在高限速下坡路段经计算不会出现超速情况。但在进入低限速阶段时，初始路段为下坡，很可能在惰行工况运行下出现超速情况。

通过如下方程可求解得到对在低限速阶段不需要巡航的初始最大速度为**76.572km/h**.

$$\begin{cases} a\left(t\right)=g\theta\left(s\right)-\frac{f(v)}{m} \\ v_{\max}^{'}-v_{s}=\int_{t_1}^{t_2}a\left(t\right)dt \\ s_2-s_1=\int_{t_1}^{t_2}v\left(t\right)dt \end{cases} \tag{40}$$

其中 $v_{\max}^{'}$ 表示低限速阶段的限速大小即 86km/s；s_1 表示低限速阶段的下坡起点即 4259.1m；s_2 表示低型限速阶段的下坡终点即 4604.65m；v_s 表示在低限速阶段不需要巡航的初始最大速度。

当 v_0 >76.572km/h 时, 说明列车在坡度影响下，由于下坡导致的加速超过限速，需要制动巡航并储能，若之后再出现上坡减速，再切换至惰行。

当 v_0 ≤76.572km/h 时, 说明列车在低限速路段里不可能发生超速现象，因此可以直接惰行跳过巡航。

粒子群优化算法求解：

基于上述数据分析确定的运行工况安排，采用粒子群优化算法求解每个工况的时间以确定最优运行策略，针对本题设置适应度函数如下

$$\begin{cases} y_1=-1+e^{\frac{E_{\min}-E}{E_{\min}}} \\ y_2=\frac{10}{1+(Time-T)} \\ y_3=\begin{cases} 1,\varepsilon<10 \\ \frac{101}{e^2+1},\varepsilon\geq10 \end{cases} \\ y_4=\begin{cases} 1,v\leq v_{\max} \\ \frac{1}{1+(v-v_{\max})},v>v_{\max} \end{cases} \end{cases} \tag{41}$$

其中 y 表示根据模型设定的目标函数；$Time$ 表示列车当前的实际运行时间；ε 表示停车精度，$E_{\min}$ 表示当前记录的最小能耗，E 表示当前能耗。v_{max} 表示最大运行速度。

综合题目和相关研究，为各函数设置权重，得到适应度函数如下：

$$fitness=20y_1+100y_2+10y_3+10y_4. \tag{42}$$

其中 $fitness$ 表示适应度函数。

5.3.4 结果分析

根据模型求解，为了使节能最大化，在满足约束条件的情况下，尽可能增加惰行工况的持续时间，并针对 86km/h 限速路段的下坡路段的超速情况采取制动巡航储能措施。最后得到的最短时间为 **209.6s**，在其基础上分别增加 10s、20s、50s、150s、300s 到达站台所计算得到的信息如下表所示。

运行时间	最低能耗 (KJ)	最高行驶速度 (km/h)	总时间 (s)
最短时间	77181	100	209.6
增加 10s	60389	95	219.81
增加 20s	54857	96.25	229.38
增加 50s	43277	86.71	259.51
增加 150s	27185	70	359.54
增加 300s	20145	59.14	509.93

在最短时间基础上延长 10s、50s 到达站台的列车运行过程速度-距离曲线、牵引制动力-距离曲线、时间-距离曲线与能量消耗-距离曲线如下所示，其余延长时间的详细结果见附件 B。

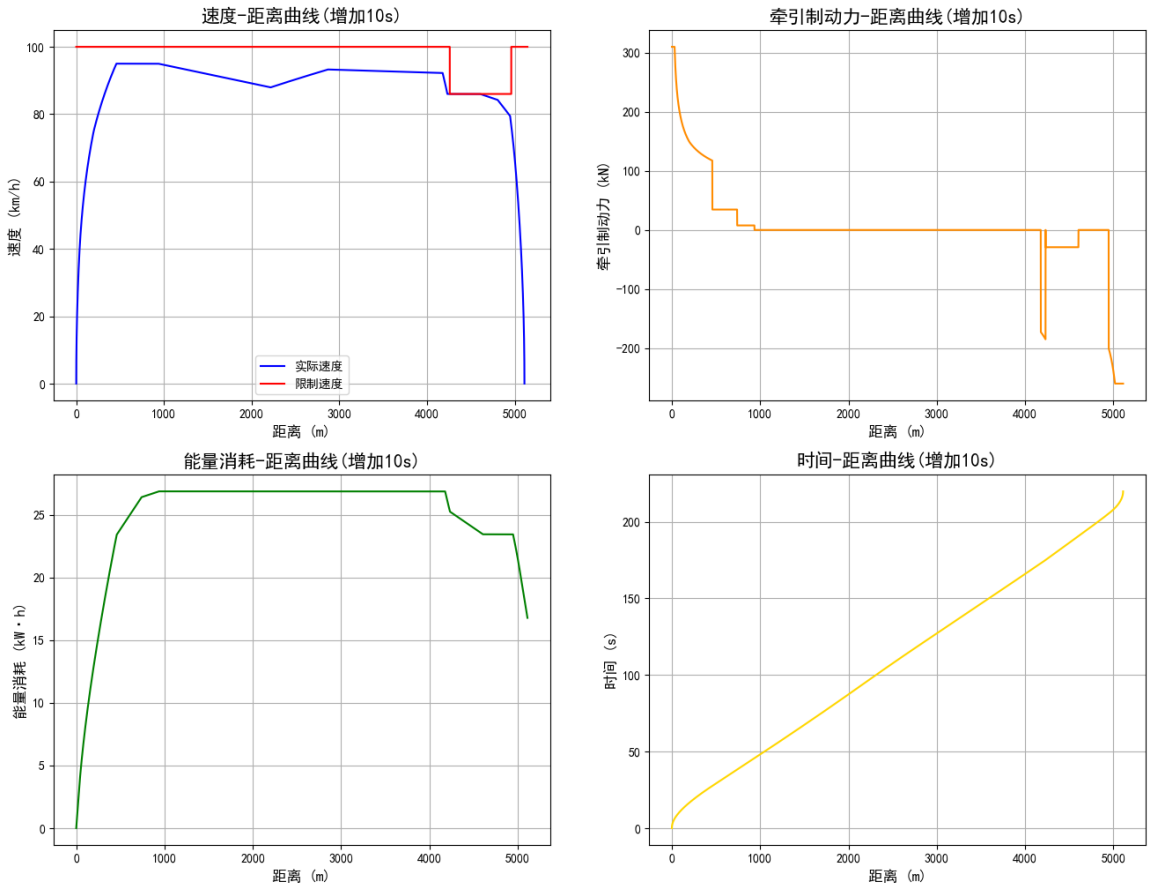


图 11　列车运行曲线图-10s

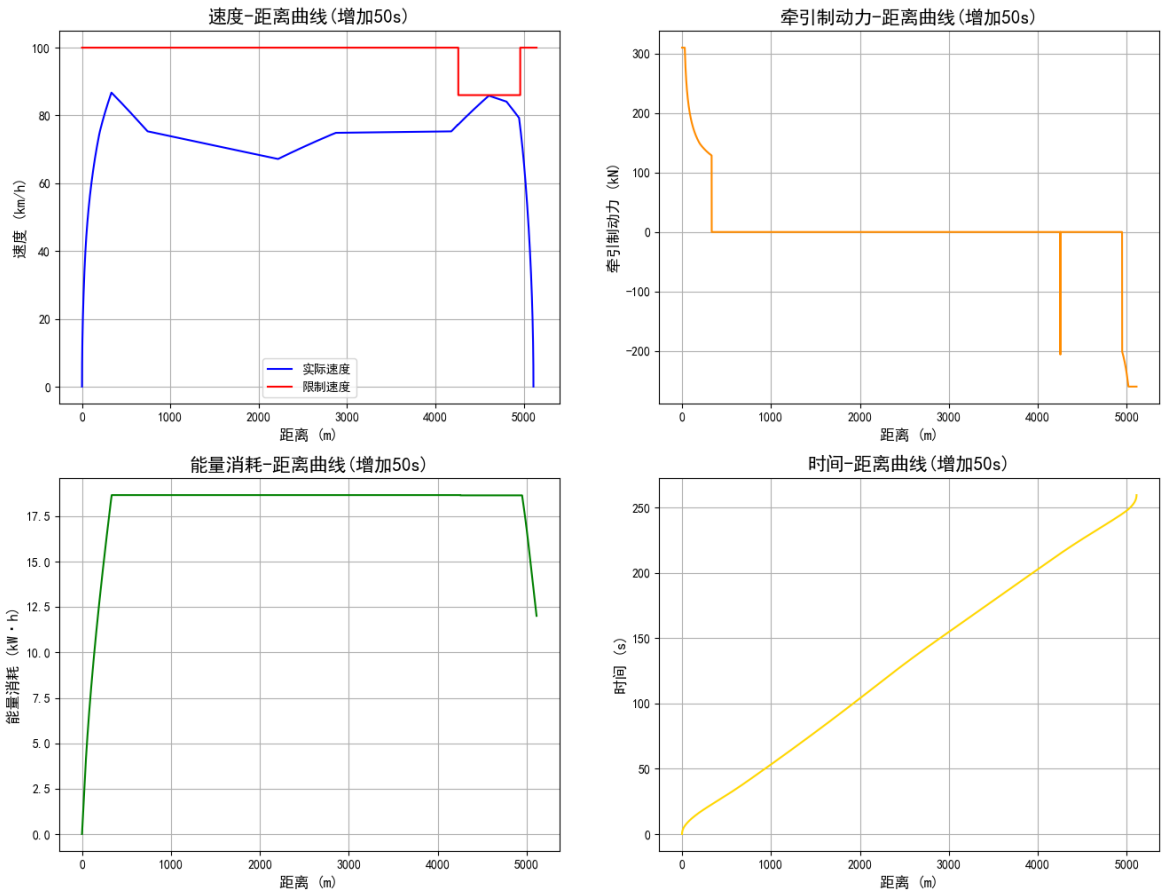


图 12　列车运行曲线图-50s

延长 10s 的情况下，列车经历了牵引-惰行-制动-巡航-制动 5 个阶段；延长 50s 的情况下，列车经历了牵引-惰行-巡航-惰行-制动 5 个阶段，可见由于惰行导致在 86km/h 限速路段的下坡路段会发生超速现象，因此采用制动巡航的方式运行。由结果可以发现，随着约束时间的延长，列车所需要牵引达到的最高行驶速度减小，产生的最终能耗也会减少，可见追求节能和即时两个理想目标往往要进行权衡。

6 问题三模型的建立与求解

6.1 问题三分析

问题三由于突发事故将列车运行分为两个阶段，第一阶段列车按照原计划 320s 运行考虑能耗最小，求得在 2000m 的运行状态，作为第二阶段的初始状态，基于问题二的运行策略优化模型，增加初始状态约束，调整目标函数，建立目标规划模型进行求解。

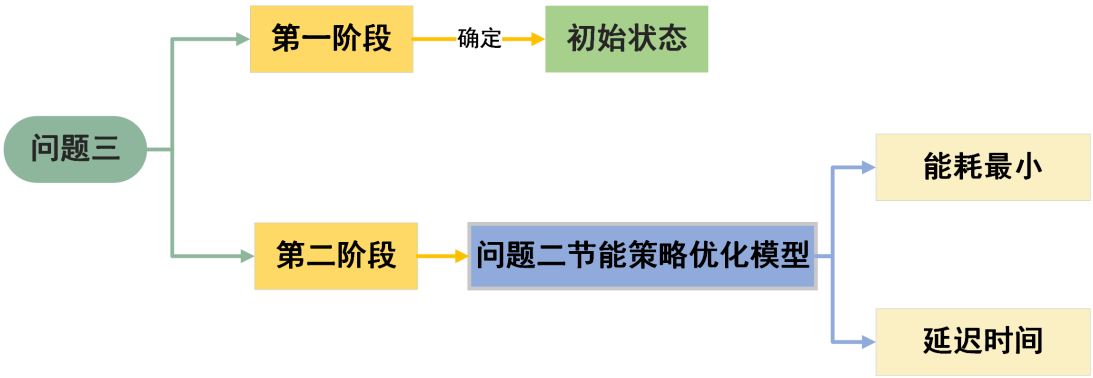


图 13　问题三思路图

6.2 分阶段的列车运行策略规划模型

将列车运行分为两个阶段[?]。在突发事故发生前，假设列车的运行方案是在计划时间为 320s 的条件，以最低能耗为目标的规划模型的最优策略作为参考。在突发事故发生后，列车运行将以当前在 2000m 的运行状态作为初始状态，建立在到达时间为 380s 的条件下，以最低能耗为目标的单目标规划模型，求取最优驾驶策略，作为调整后的优化速度轨迹。

6.2.1 确定初始运行状态

借助问题二的基于能耗最小的站间节能策略优化模型，将 T 设置为 320s 求取得到最优节能策略。依据此，获取列车行至 2000m 位置时的速度、牵引力大小、阻力大小等运行状态参数。并将其作为下一阶段的初始运行状态。

6.2.2 目标规划模型建立

初始状态约束：基于问题二的节能策略优化模型，增加对列车初始状态的考虑，初始状态处于列车运行的中间过程阶段。

$$\begin{cases} v\left(t\right)=v_0+\int_{t_0}^{t}a\left(t\right)dt \\ s\left(t\right)=s_0+\int_{t_0}^{t}v\left(t\right)dt \end{cases} \tag{43}$$

其中 v_0 表示初始状态的速度；s_0 表示初始路程；t_0 表示列车行至 2000m 位置的时间。

目标函数：

根据题目要求优化方案在保持列车节能运行下，即需保证第二阶段的能耗最小：

$$\min E = \frac{\int_{t_0}^{T} [x_1(t) + x_2(t)] \times F_Q(v) \times v(t)\, dt}{\eta_Q} - \eta_Z \times \int_{t_0}^{T} x_4(t) \times F_{Z1}(v) \times v(t)\, dt \quad (44)$$

6.3 求解与结果分析

第一阶段的结束状态如下表，在第二阶段以此作为初始运行状态。

时间 (s)	牵引制动力 (KN)	基本阻力 (KN)	坡度阻力 (KN)	速度 (km/h)	加速度 (m/s)
129.75	0	3.34	1.17	47.26	-0.026

基于列车在 2000m 处的运行状态，建立新的单目标规划模型，得到列车运行过程的速度-距离曲线、牵引制动力-距离曲线、时间-距离曲线与能量消耗-距离曲线如下图。

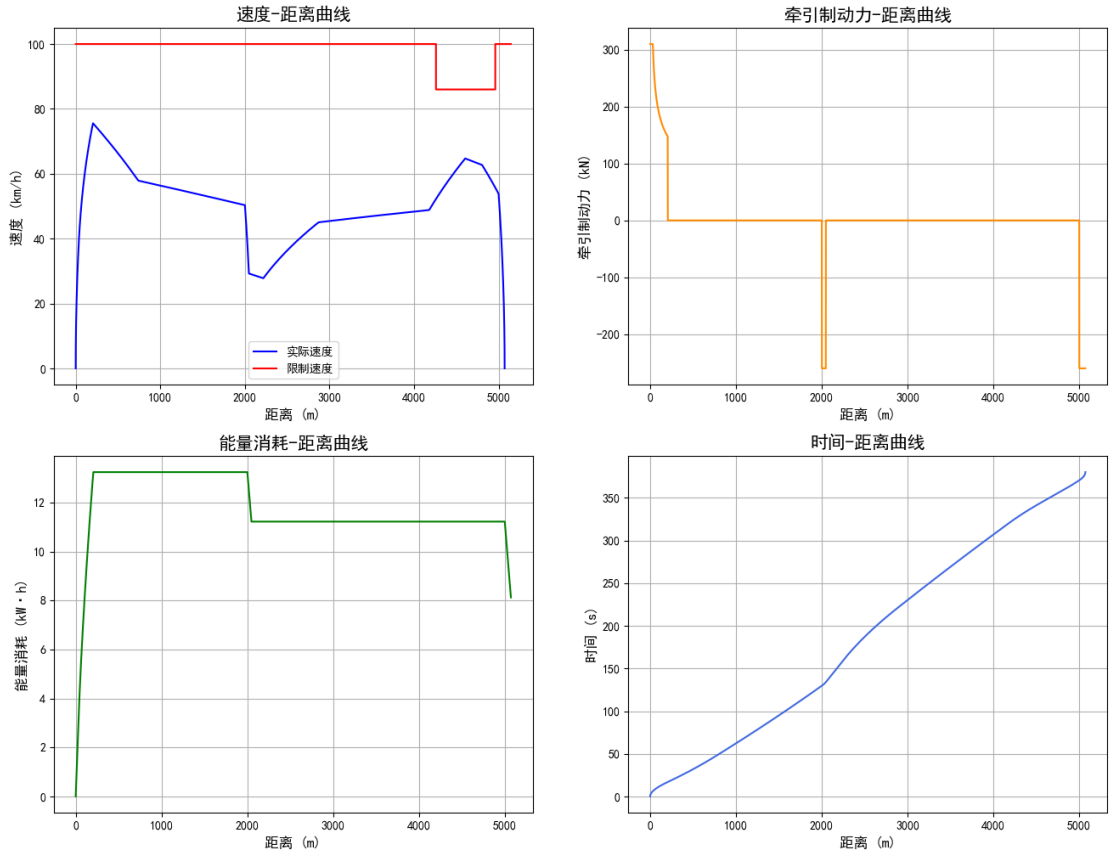


图 14　列车运行曲线图-突发事件

得到的阻力-距离曲线图如下所示。

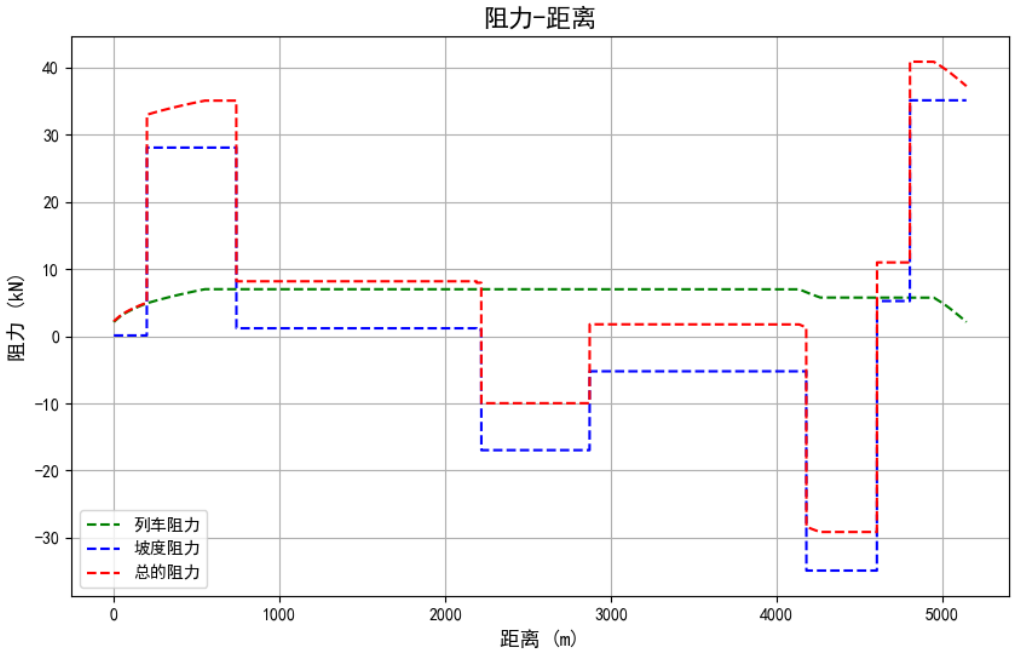


图 15　阻力-距离曲线图

列车到站的最终能耗为 **29250 kJ**，最高行驶速度为 **75.546875 km/h**，列车行驶时间为 **380.27 s**，与只是采取简单的牵引-巡航-制动工况模型节省了约 **10728kJ** 的能耗，可见采用惰行优化的列车运行策略模型可以有效地在相同到达时间的条件下提高节能效率。根据结果可以发现，采用不同的工况策略可能会对最终的列车能耗造成很大的影响，因此列车司机在面对突发状况时，需要随机应变，在规定时间内采取合适的策略节省列车运行能耗。

7 模型总结与评价

7.1 模型优点

1) 问题一建立基于改进的 PSO 的时间优化策略规划模型，引入随机游走策略改进 PSO 算法，更有利于求解全局最优解，提高算法效率。

2) 问题二建立惰行优化运行策略规划模型，引入列车惰行过程，在保证运行时间的情况下，进一步减少列车运行过程中的能量消耗。

3) 问题三建立分段惰行列车优化运行策略规划模型，考虑实际路况的不确定因素，分别对突发事件发生前后采用不同的优化策略，从而得到运行策略的最优解。

7.2 模型缺点

1) 问题二模型没有考虑运行过程乘客的舒适度与满意度，后续可以对模型添加舒适度的相关参数，如舒适加速度，并设置约束条件，进一步完善模型。

2) 由于列车相关参数不完整，导致模型仍存在不足，后续可以通过完善参数，对模型添加约束条件，进一步优化模型，使模型结果更准确可靠。

7.3 模型推广

本文的模型建立基于改进的 PSO 算法的惰行优化运行策略模型，综合考虑运行时间与列车耗能问题，并针对路况做出分段规划，后续可以结合更多实际情况与列车的相关参数进行完善和推广，得到更贴近真实生活的结果，助力城市交通电气化进程快速推进，更好地挖掘城轨系统减碳节能的潜力。

参考文献

[1] 李晶, 荀径, 尹晓宏, 等. 北京市轨道交通列车运行节能控制方案研究与应用 [J]. 铁道运输与经济, 2022,44(06):136-141.

[2] 李华柏陈春棉. 基于遗传算法的高速动车组列车惰行节能控制策略研究 [J]. 电机与控制应用, 2022, 49(7):112-117.

[3] 张友兵, 王建敏, 张国振等. 高速铁路列车制动曲线计算精确度与效率分析 [J]. 铁道科学与工程学报,2022,19(01):10-18.

[4] 杨彦强, 刘海东, 麻存瑞等. 列车节能运行目标速度控制优化研究 [J]. 交通运输系统工程与信息,2019,19(01):138-144.

[5] 邹文骏. 苏州轨道交通列车再生制动能量吸收装置应用情况分析 [J]. 城市轨道交通研究, 2022,25(05):21-26.

[6] 缪鹃, 王介源, 戴炎林. 一种基于离散区间工况选择的列车节能运行优化方法 [J]. 中国铁道科学,2023,44(02):211-220.

[7] T. Zhang and Z. Wang, ”The Approximation of the Train Resistance Based on Improved PSO-RBF,” 2020 Chinese Control And Decision Conference (CCDC), Hefei, China, 2020: 1726-1731

[8] Ying C S, Chow A H F, Wang Y H, et al. Adaptive metro service schedule and train composition with a proximal policy optimization approach based on deep reinforcement learning[J]. IEEE Transactions on Intelligent Transportation Systems, 2021, 23(7): 6895-6906.

[9] Rao Y, Sun P, Wang Q, et al. Optimal running time supplement for the energy□efficient train control considering the section running time constraint[J]. IET Intelligent Transport Systems, 2022, 16(5): 661-674.

[10] He D , Zhang L , Guo S ,et al.Energy-efficient Train Trajectory Optimization Based on Improved Differential Evolution Algorithm and Multi-particle Model[J].Journal of Cleaner Production, 2021, 304(2):127163

附录 A　问题一结果

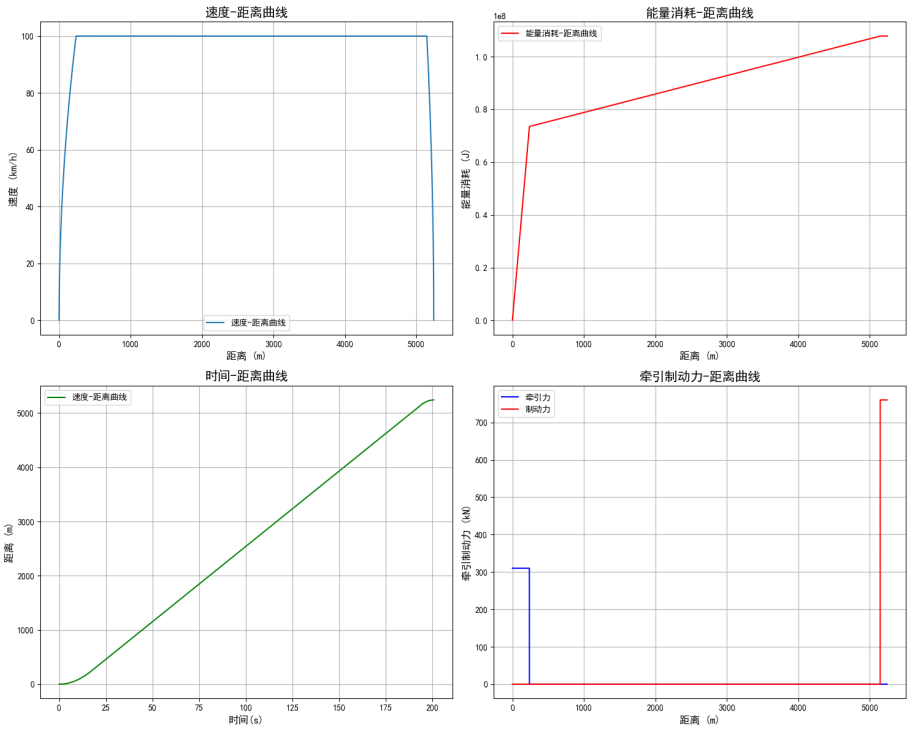


图 16　列车运行曲线图-最短时间

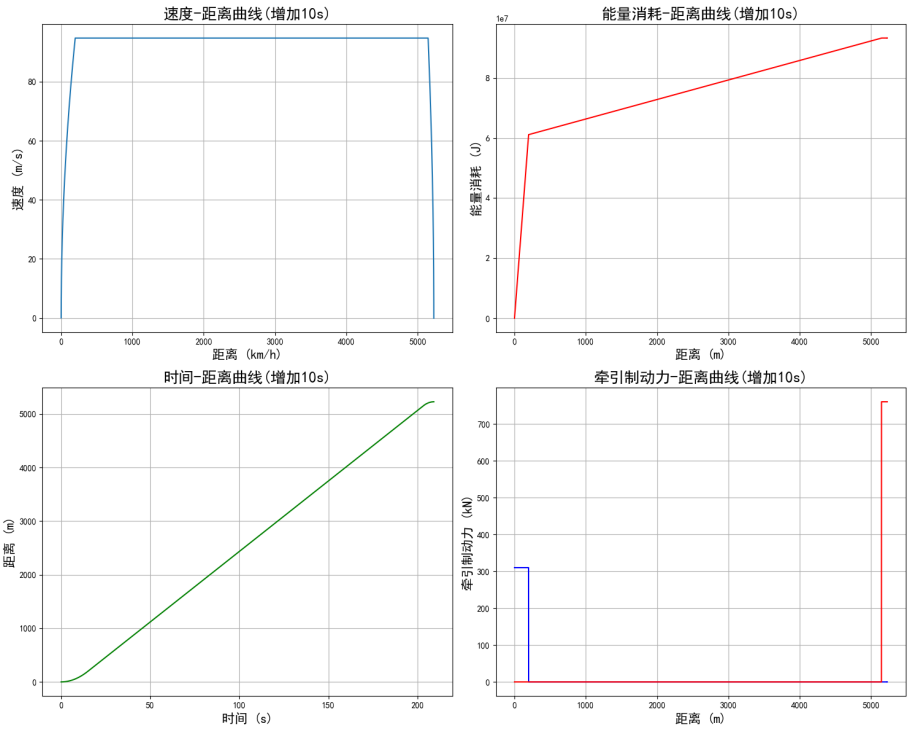


图 17　列车运行曲线图-10s

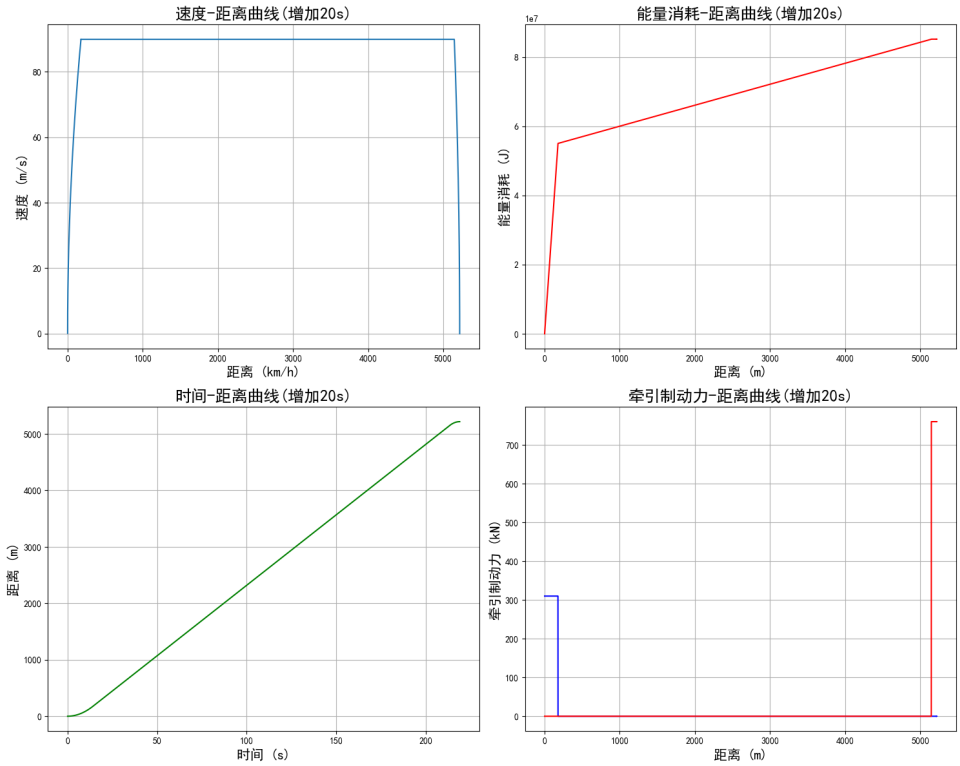


图 18　列车运行曲线图-20s

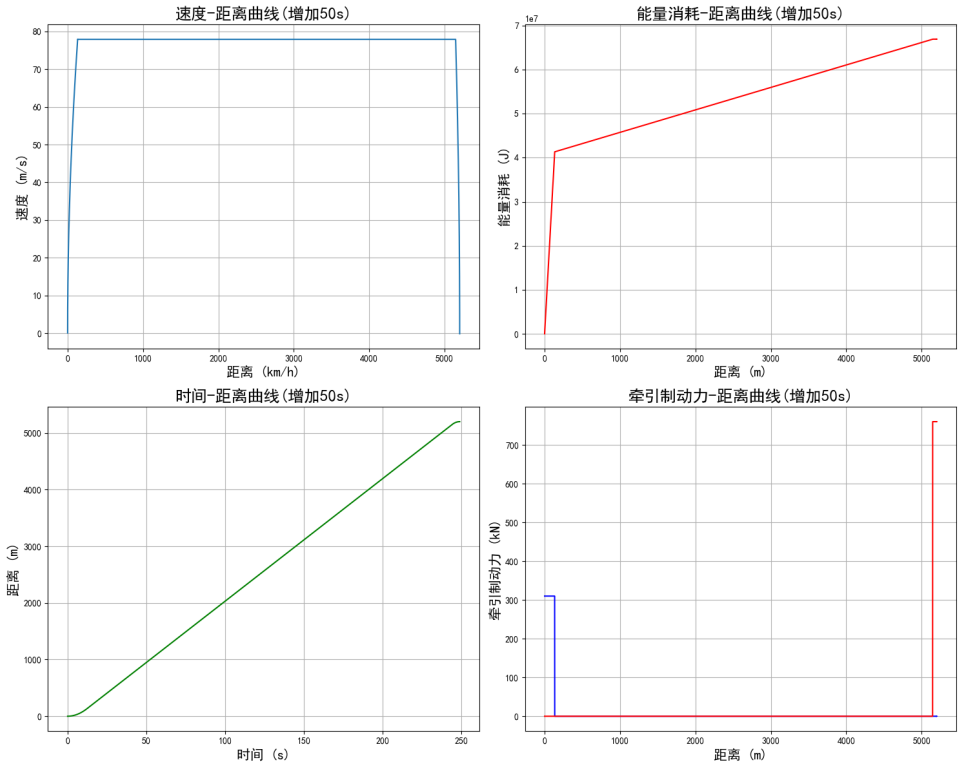


图 19　列车运行曲线图-50s

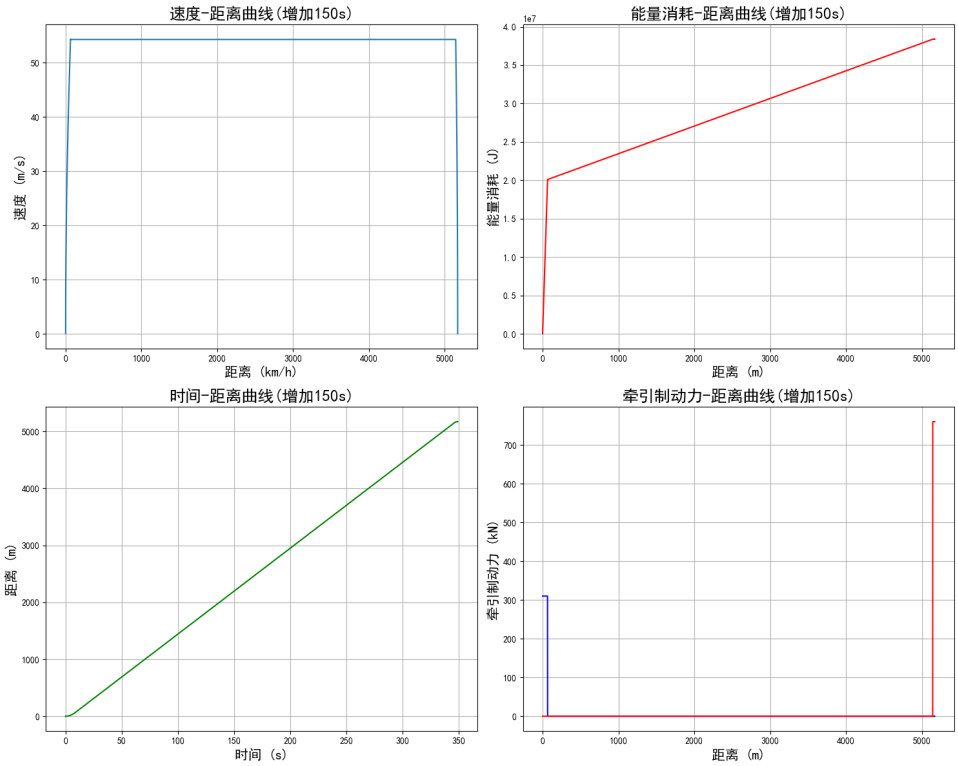


图 20　列车运行曲线图-150s

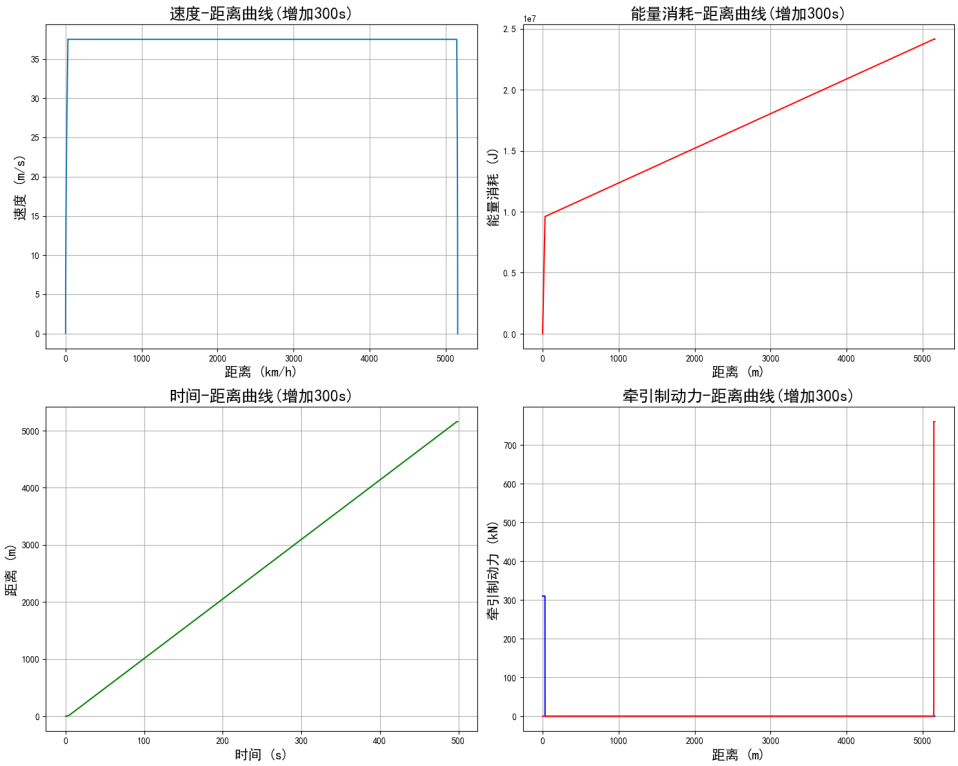


图 21　列车运行曲线图-300s

附录 B　问题二结果

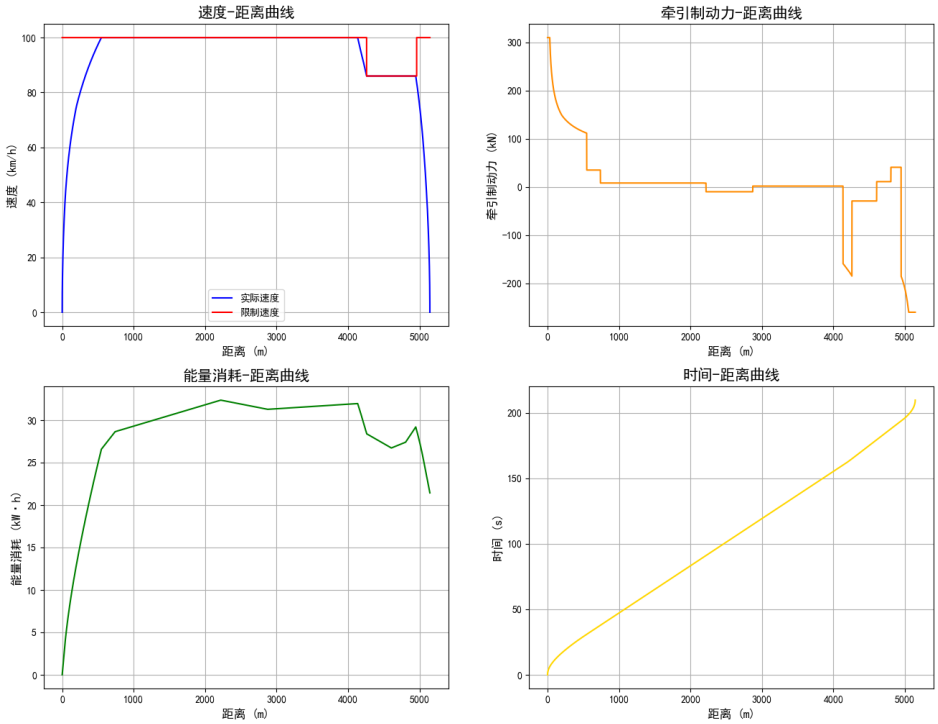


图 22　列车运行曲线图-最短时间

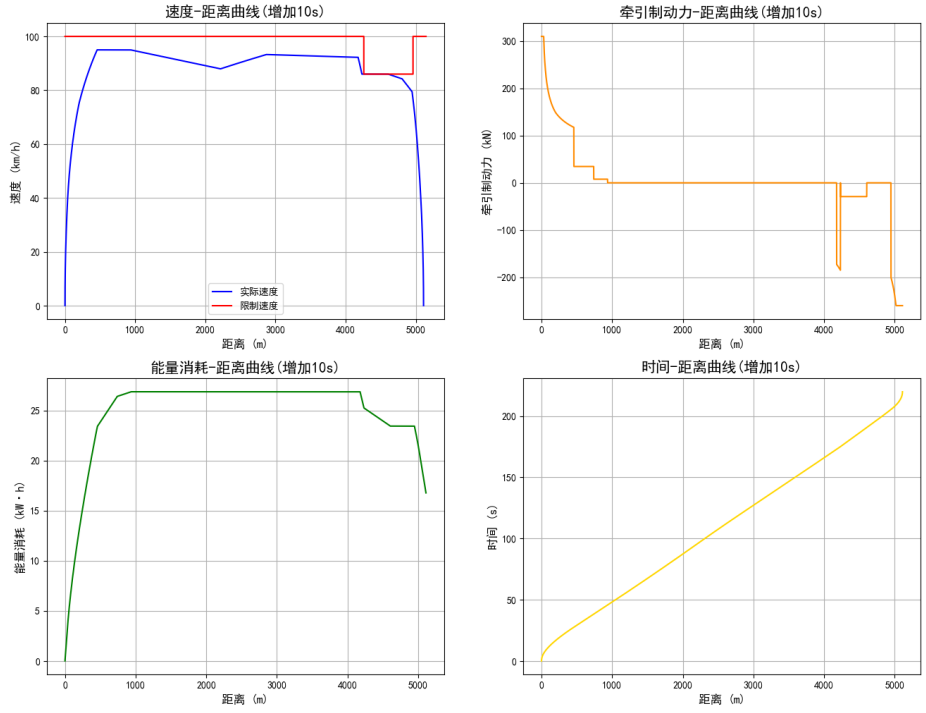


图 23　列车运行曲线图-10s

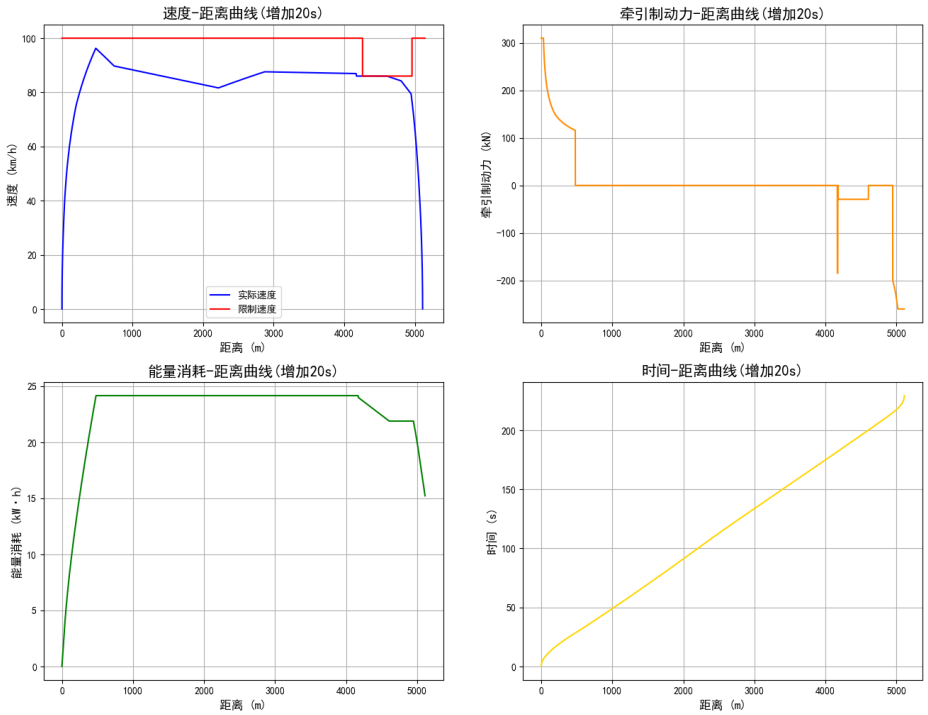


图 24　列车运行曲线图-20s

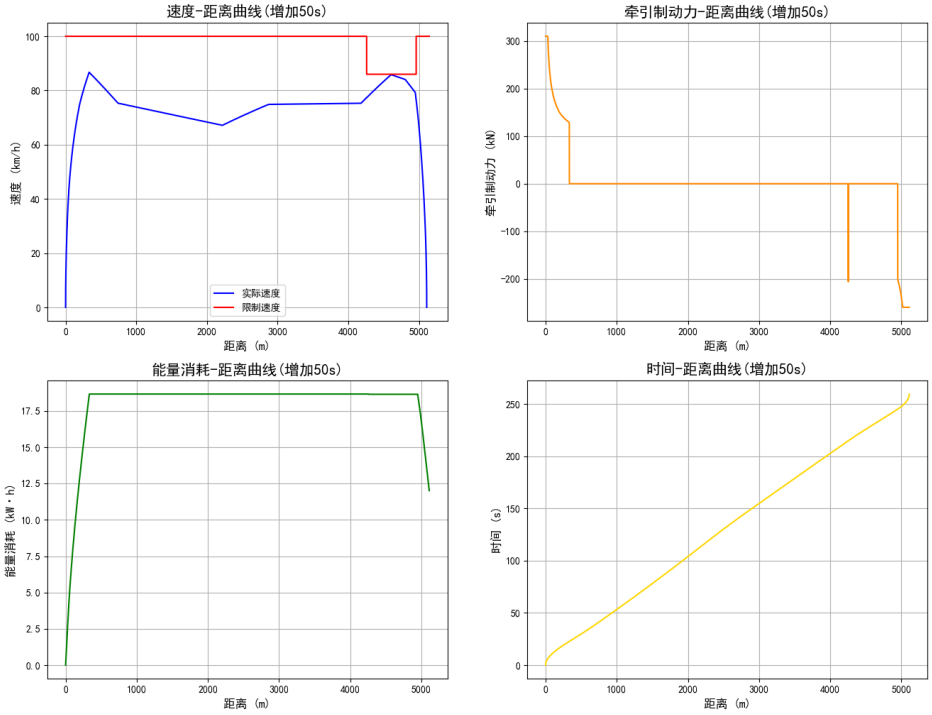


图 25　列车运行曲线图-50s

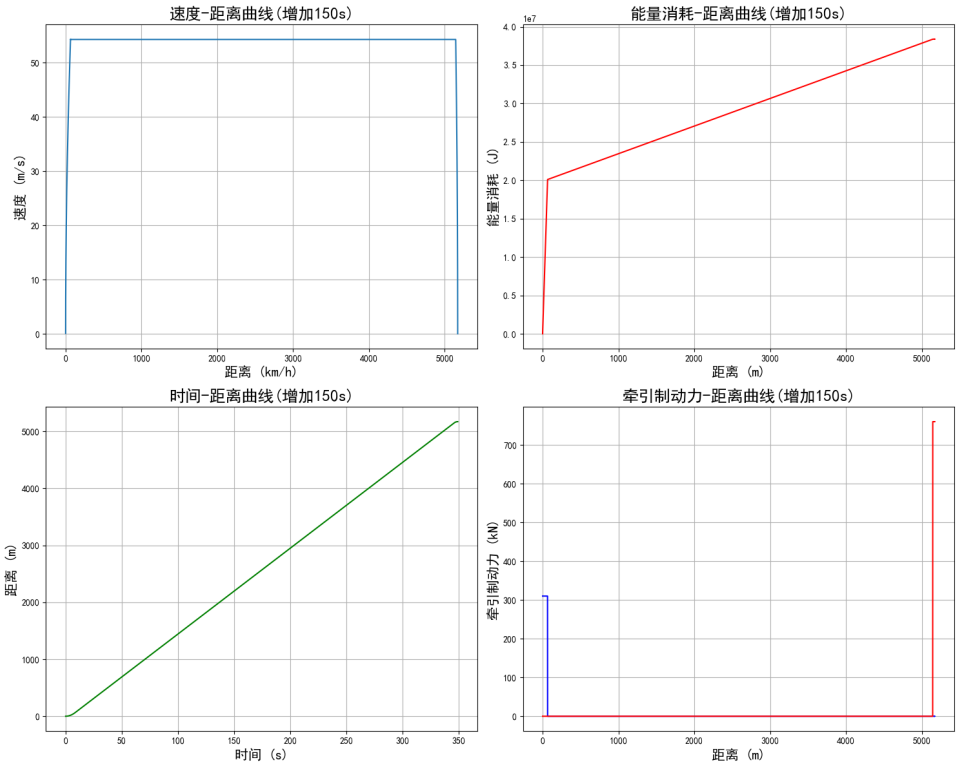


图 26　列车运行曲线图-150s

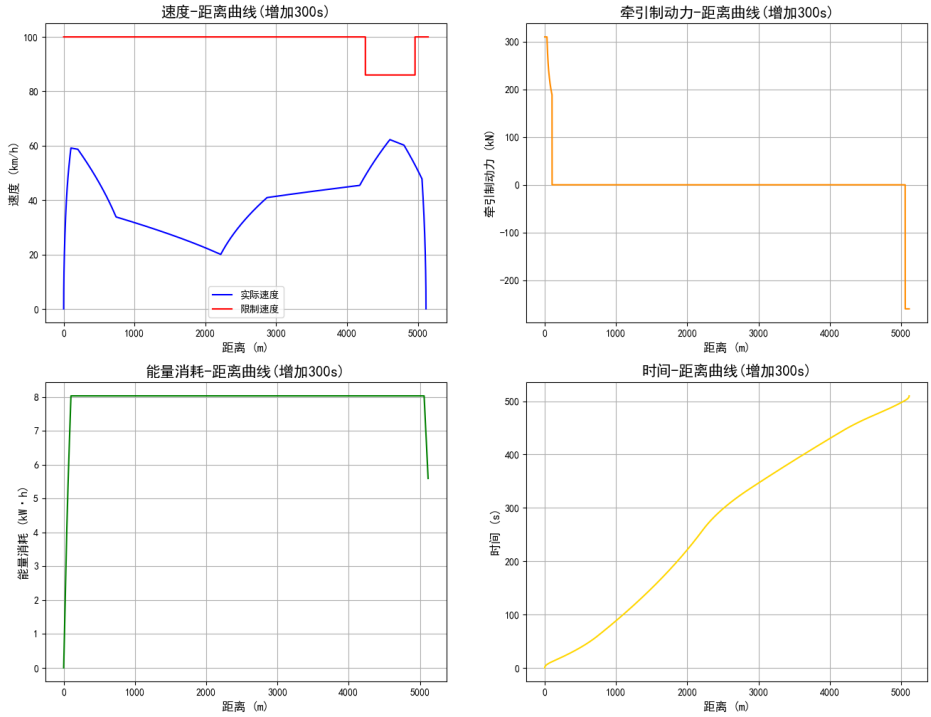


图 27　列车运行曲线图-300s

附录 C　问题一 Python 代码

3.1　求最短时间曲线

```
import time as pytime
plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中文标签
plt.rcParams['axes.unicode_minus']=False #用来正常显示负号

# Given parameters
M = 176300*1.08 # Mass of the train in kg
v_cruise = 27.78 # Cruising speed of the train in m/s
FQ_max = 310000 # Maximum traction force in N
B_max = 760000 # Maximum braking force in N
g = 10 # Acceleration due to gravity in m/s^2
L = 5144.7
# Time step for the simulation
dt = 0.001

# Defining the force function f(v) based on velocity
def f(v):
    # Given f(v) = 2.0895 + 0.0098v + 0.006v^2 and considering the units (N = 1000 KN)
    return (2.0895 + 0.0098 * v + 0.006 * v**2) * 1000

# Starting the timer
start_time = pytime.time()

# Resetting the lists to store values for plotting
times = []
positions = []
velocities = []
energies = []
traction_forces = []
brake_forces = []

# Resetting initial parameters for the simulation
time = 0
position = 0
velocity = 0
energy_consumption = 0
print(position)
# Traction phase
traction_start_time = time
while velocity < v_cruise:
    force_traction = FQ_max
    acceleration = force_traction / M
    velocity += acceleration * dt
```

```
    position += velocity * dt
    energy_consumption += force_traction * velocity * dt
    time += dt
    times.append(time)
    positions.append(position)
    velocities.append(velocity)
    energies.append(energy_consumption)
    traction_forces.append(force_traction / 1000) # Convert to kN
    brake_forces.append(0) # No brake force in this phase
traction_end_time = time
print(position)
# Cruising phase - Adjusting the time for cruising to ensure total distance is 5144.7m
remaining_distance = L - position
time_cruise = remaining_distance / v_cruise - dt # Subtracting dt for accurate braking
    transition
cruise_start_time = time
for _ in range(int(time_cruise/dt)):
    position += v_cruise * dt
    energy_consumption += f(v=v_cruise) * v_cruise * dt
#     velocity -= f(v=velocity)/M * dt
#     position += velocity * dt
    time += dt
    times.append(time)
    positions.append(position)
#     velocities.append(v_cruise)
    velocities.append(velocity)
    energies.append(energy_consumption)
    traction_forces.append(0) # No traction force in cruising phase
    brake_forces.append(0) # No brake force in cruising phase
cruise_end_time = time
print(position)
# Braking phase
brake_start_time = time
while velocity > 0:
    force_brake = B_max
    acceleration = -force_brake / M
    velocity += acceleration * dt
    position += velocity * dt
    energy_consumption += force_brake * velocity * dt
    time += dt
    times.append(time)
    positions.append(position)
    velocities.append(velocity)
    energies.append(energy_consumption)
    traction_forces.append(0) # No traction force in braking phase
    brake_forces.append(force_brake / 1000) # Convert to kN
brake_end_time = time
```

```
print(position)
# Ending the timer
end_time = pytime.time()

# Plotting the curves
plt.figure(figsize=(15, 12))

# Speed - Position
plt.subplot(2, 2, 1)
plt.plot(positions, velocities, label="速度-距离曲线")
plt.xlabel("距离 (m)",fontsize=12)
plt.ylabel("速度 (m/s)",fontsize=12)
plt.title("速度-距离曲线",fontsize=15)
plt.grid(True)
plt.legend()

# Energy Consumption - Position
plt.subplot(2, 2, 2)
plt.plot(positions, energies, label="能量消耗-距离曲线", color="red")
plt.xlabel("距离 (m)",fontsize=12)
plt.ylabel("能量消耗 (J)",fontsize=12)
plt.title("能量消耗-距离曲线",fontsize=15)
plt.grid(True)
plt.legend()

# Time - Position
plt.subplot(2, 2, 3)
plt.plot(times, positions, label="速度-距离曲线", color="green")
plt.xlabel("时间(s)",fontsize=12)
plt.ylabel("距离 (m)",fontsize=12)
plt.title("速度-距离曲线",fontsize=15)
plt.grid(True)
plt.legend()

# Traction and Brake forces vs. Position
plt.subplot(2, 2, 4)
plt.plot(positions, traction_forces, label="牵引力", color="blue")
plt.plot(positions, brake_forces, label="制动力", color="red")
plt.xlabel("距离 (m)",fontsize=12)
plt.ylabel("牵引制动力 (kN)",fontsize=12)
plt.title("牵引制动力-距离曲线",fontsize=15)
plt.grid(True)
plt.legend()

plt.tight_layout()
plt.savefig("E:/Desktop/曲线图.png")
plt.show()
```

```
# Outputting the required metrics
output = {
    "Program Execution Time (s)": end_time - start_time,
    "Traction Phase Time (s)": traction_end_time - traction_start_time,
    "Cruising Phase Time (s)": cruise_end_time - cruise_start_time,
    "Braking Phase Time (s)": brake_end_time - brake_start_time,
    "Total Travel Time (s)": brake_end_time,
    "Total Energy Consumption (J)": energies[-1]
}

output
```

3.2 随机游走 PSO

```
import matplotlib.pyplot as plt

# Given parameters
M = 176300 # Mass of the train in kg
v_cruise = 27.78 # Cruising speed of the train in m/s
FQ_max = 310000 # Maximum traction force in N
B_max = 760000 # Maximum braking force in N
g = 9.81 # Acceleration due to gravity in m/s^2
L = 5144.7 # Total distance

# Defining the force function f(v) based on velocity
def f(v):
    return (2.0895 + 0.0098 * v + 0.006 * v**2) * 1000

def simulate_with_specific_cruise_speed(cruise_speed):
    dt = 0.01
    time = 0
    position = 0
    velocity = 0
    times = []
    positions = []
    velocities = []

    # Traction phase
    while velocity < cruise_speed:
        force_traction = min(FQ_max, M * f(v=velocity) + M * g)
        acceleration = force_traction / M
        velocity += acceleration * dt
        position += velocity * dt
        time += dt
```

```
        times.append(time)
        positions.append(position)
        velocities.append(velocity)

    # Cruising phase
    while position < (L - (B_max * velocity**2 / (2 * M * g))):
        force_traction = f(v=velocity)
        velocity = cruise_speed
        position += velocity * dt
        time += dt
        times.append(time)
        positions.append(position)
        velocities.append(velocity)

    # Braking phase
    while velocity > 0:
        force_brake = min(B_max, M * f(v=velocity) + M * g)
        acceleration = -force_brake / M
        velocity += acceleration * dt
        position += velocity * dt
        time += dt
        times.append(time)
        positions.append(position)
        velocities.append(velocity)

    return {
        "times": times,
        "positions": positions,
        "velocities": velocities
    }

# Particle definition for PSO
class Particle:
    def __init__(self):
        self.position = np.random.uniform(0, v_cruise)
        self.velocity = np.random.uniform(-0.5, 0.5)
        self.best_position = self.position
        self.best_score = float('inf')

# Optimized PSO function
def PSO_optimized(target_extra_time, n_particles=10, n_iterations=100, w=0.5, c1=1.5, c2=1.5,
    random_walk_factor=0.1):
    particles = [Particle() for _ in range(n_particles)]
    g_best_position = np.random.uniform(0, v_cruise)
    g_best_score = float('inf')
    best_scores = []
```

```
    for _ in range(n_iterations):
        for particle in particles:
            fitness = abs(simulate_with_specific_cruise_speed(particle.position)["times"][-1] -
                (simulate_with_specific_cruise_speed(v_cruise)["times"][-1] + target_extra_time))
            if fitness < particle.best_score:
                particle.best_score = fitness
                particle.best_position = particle.position

            if fitness < g_best_score:
                g_best_score = fitness
                g_best_position = particle.position

        best_scores.append(g_best_score)

        for particle in particles:
            inertia = w * particle.velocity
            personal_attraction = c1 * np.random.random() * (particle.best_position -
                particle.position)
            global_attraction = c2 * np.random.random() * (g_best_position - particle.position)
            random_walk = random_walk_factor * np.random.uniform(-1, 1)
            particle.velocity = inertia + personal_attraction + global_attraction + random_walk
            particle.position += particle.velocity
            particle.position = np.clip(particle.position, 0, v_cruise)

    return g_best_position, best_scores

# Run the optimized PSO for 10s extended time and plot the results
adjusted_speed, best_scores = PSO_optimized(10)

plt.figure(figsize=(10, 5))
plt.plot(best_scores)
plt.xlabel("迭代次数",fontsize=15)
plt.ylabel("时间适应度",fontsize=15)
plt.title("基于随机游走策略的PSO收敛图（10s）",fontsize=20)
plt.grid(True)
plt.savefig("E:/Desktop/基于随机游走策略的PSO收敛图（10s）.png")
plt.show()

print("Adjusted cruising speed for 10s extended time:", adjusted_speed, "m/s")
```

3.3 延长时间曲线

```
# Given parameters
M = 176300 # Mass of the train in kg
```

```
v_cruise = 27.78 # Cruising speed of the train in m/s
FQ_max = 310000 # Maximum traction force in N
B_max = 760000 # Maximum braking force in N
g = 9.81 # Acceleration due to gravity in m/s^2
L = 5144.7 # Total distance in m
dt = 0.01 # Time step for the simulation

# Force function based on velocity
def f(v):
    return (2.0895 + 0.0098 * v + 0.006 * v**2) * 1000

# Simulation function with specific cruising speed
def simulate_with_specific_cruise_speed(adjusted_v_cruise):
    # Starting the timer
    start_time = pytime.time()

    time = 0
    position = 0
    velocity = 0
    energy_consumption = 0
    times = []
    positions = []
    velocities = []
    energies = []
    traction_forces = []
    brake_forces = []

    # Traction phase
    traction_start_time = time
    while velocity < adjusted_v_cruise:
        force_traction = min(FQ_max, M * f(v=velocity) + M * g)
        acceleration = force_traction / M
        velocity += acceleration * dt
        position += velocity * dt
        energy_consumption += force_traction * velocity * dt
        time += dt
        times.append(time)
        positions.append(position)
        velocities.append(velocity)
        energies.append(energy_consumption)
        traction_forces.append(force_traction / 1000)
        brake_forces.append(0)
    traction_end_time = time

    # Cruising phase
    cruise_start_time = time
    while position < L - (L - position) * (B_max / (M * g + B_max)):
```

```
    position += adjusted_v_cruise * dt
    energy_consumption += f(v=adjusted_v_cruise) * adjusted_v_cruise * dt
    time += dt
    times.append(time)
    positions.append(position)
    velocities.append(adjusted_v_cruise)
    energies.append(energy_consumption)
    traction_forces.append(0)
    brake_forces.append(0)
cruise_end_time = time

# Braking phase
braking_start_time = time
while velocity > 0:
    force_brake = min(B_max, M * f(v=velocity) + M * g)
    acceleration = -force_brake / M
    velocity += acceleration * dt
    position += velocity * dt
    energy_consumption += force_brake * velocity * dt
    time += dt
    times.append(time)
    positions.append(position)
    velocities.append(velocity)
    energies.append(energy_consumption)
    traction_forces.append(0)
    brake_forces.append(force_brake / 1000)
braking_end_time = time

# Ending the timer
end_time = pytime.time()

# Outputting the required metrics
output = {
    "Program Execution Time (s)": end_time - start_time,
    "Traction Phase Time (s)": traction_end_time - traction_start_time,
    "Cruising Phase Time (s)": cruise_end_time - cruise_start_time,
    "Braking Phase Time (s)": braking_end_time - brake_start_time,
    "Total Travel Time (s)": braking_end_time,
    "Total Energy Consumption (J)": energies[-1]
}

print(output)

return {
    "times": times,
    "positions": positions,
    "velocities": velocities,
```

```
        "energies": energies,
        "traction_forces": traction_forces,
        "brake_forces": brake_forces
    }

# Adjusted cruise speeds based on the required added times
adjusted_cruise_speeds = {
    10: 26.32,
    20: 24.97,
    50: 21.63,
    150: 15.07,
    300: 10.42
}

# Simulating and plotting for each added time
for added_time, v_cruise_adjusted in adjusted_cruise_speeds.items():
    result = simulate_with_specific_cruise_speed(v_cruise_adjusted)
    plt.figure(figsize=(15, 12))

    # Speed - Position
    plt.subplot(2, 2, 1)
    plt.plot(result["positions"], result["velocities"])
    plt.xlabel("距离 (m)",fontsize=15)
    plt.ylabel("速度 (m/s)",fontsize=15)
    plt.title(f"速度-距离曲线(增加{added_time}s)",fontsize=18)
    plt.grid(True)

    # Energy Consumption - Position
    plt.subplot(2, 2, 2)
    plt.plot(result["positions"], result["energies"], color="red")
    plt.xlabel("距离 (m)",fontsize=15)
    plt.ylabel("能量消耗 (J)",fontsize=15)
    plt.title(f"能量消耗-距离曲线(增加{added_time}s)",fontsize=18)
    plt.grid(True)

    # Time - Position
    plt.subplot(2, 2, 3)
    plt.plot(result["times"], result["positions"], color="green")
    plt.xlabel("时间 (s)",fontsize=15)
    plt.ylabel("距离 (m)",fontsize=15)
    plt.title(f"速度-距离曲线(增加{added_time}s)",fontsize=18)
    plt.grid(True)

    # Traction and Brake forces vs. Position
    plt.subplot(2, 2, 4)
    plt.plot(result["positions"], result["traction_forces"], color="blue")
    plt.plot(result["positions"], result["brake_forces"], color="red")
```

```
    plt.xlabel("距离 (m)",fontsize=15)
    plt.ylabel("牵引制动力 (kN)",fontsize=15)
    plt.title(f"牵引制动力-距离曲线(增加{added_time}s)",fontsize=18)
    plt.grid(True)

    plt.tight_layout()
    plt.savefig(f"E:/Desktop/problem1_{added_time}s.png")
    plt.show()
```

附录 D　问题二 Matlab 代码

```
function fitness = CacFitNess(Energy,Time,MissError,overSpeed,Jerk)
%UNTITLED 计算适应度函数
%
global DESINTIME EMAX;
y=zeros(1,5);
 %能耗
if  Energy>EMAX
    y(1)= 0;
else
    y(1)=-1+exp((EMAX-Energy)/EMAX);
end
%时间
y(2)= 10/(1+(Time-DESINTIME)^2);

%超限速
if overSpeed>0
    y(4)=1/(1+overSpeed);
else
    y(4)=1;
end
%停车误差
if abs(MissError)<10
    y(5)=1;
else
    y(5)=101/(MissError^2+1);
end
% fitness = 10*y(1)+100*y(2)+y(3)+10*y(4)+20*y(5);
fitness = 20*y(1)+100*y(2)+10*y(4)+10*y(5);
end
clc
clear;
close all;
tic
%参数初始化
```

```
arginitial();
%限速和坡度约束处理，并且保存处理结果
CacBrakeSpeedLimit();
[Emax,Tmin]=CacMinTime();

%多目标粒子群算法参数
options.PopulationSize = 100; %种群大小
options.MaxGenerations = 100; %算法最大迭代次数
options.Continue = 0 ; %是否继续原来的优化，如果要继续则要传入种群options.Pop和速度参数options.Veo
for num=1:1
%正式开始优化
[ExREP,ExChar,REPNum,gridFit,Population,Velocity,enterCountList,repNumList,avgPersonalFitnessList,GobalFitNessL
% for i=1:REPNum
%     [flag,Energy,Time,MissError] = CalcEJT(ExREP(i,:),2);
%     disp(['运行能耗：',num2str(Energy),' 运行时间：',num2str(Time),'
    停车误差：',num2str(MissError)])
% end
[flag,Energy,Time,MissError] = CalcEJT(ExREP(1,:),1); %展示优化结果
str=['Result',num2str(num),'.mat'];
save(str,'ExREP','ExChar','enterCountList','repNumList');
str=['运行次数:',num2str(num)];
disp(str);

end
toc


function [res,Population,Velocity]=PSO_d(options)
%UNTITLED3 粒子群优化算法的主函数
%
global SWITCHNUM TRAVDIS;
global setNum omiga C1 C2 MaxFlyVeo;
%粒子群算法参数
setNum=options.PopulationSize; %种群数量
iteraNum=options.MaxGenerations; %迭代次数
omiga=1.1;%惯性因子
C1=1.5;   %个体加速常数
C2=2;   %社会加速常数
MaxFlyVeo=1.5*TRAVDIS/SWITCHNUM; %最大飞行速度
%产生初始种群
if options.Continue == 1
    Population=options.Pop;
    Velocity=options.Velo;
else
    Population=CreateInitPopulation(setNum,SWITCHNUM); %生成种群
    Velocity=zeros(setNum,SWITCHNUM);
end
```

```
socityMaxFitPos=zeros(1,SWITCHNUM);
indiviMaxFitPos=zeros(setNum,SWITCHNUM);
maxFitNessForIndivi=zeros(setNum,1);
maxFitNessForSocity=0;
maxSocityFitNessInIter=[];
indiviFitNess=zeros(setNum,1);
setnum=setNum;
swnum=SWITCHNUM;
OldPopulation=Population;
OldVelocity=Velocity;
for iter=1:iteraNum
   %对个体进行评估
   for i=1:setnum
       [flag,Energy,Time,MissError,overSpeed,swp,jerk]=CalcEJT(Population(i,:),0);
       Population(i,:)=swp;
       %测试个体,不符合要求就产生新个体
       if flag~= 0
          Population(i,:)=OldPopulation(i,:);
          Velocity(i,:)=-1*Velocity(i,:); %飞出边界则反向
           %再次测试
           [flag,Energy,Time,MissError,overSpeed,swp,jerk]=CalcEJT(Population(i,:),0);
           Population(i,:)=swp;
           while flag~= 0
              Population(i,:)=CreateInitPopulation(1,SWITCHNUM);
              [flag,Energy,Time,MissError,overSpeed,swp,jerk]=CalcEJT(Population(i,:),0);
              Population(i,:)=swp;
              disp('PSOmain 44:重新产生个体');
           end
       end
       indiviFitNess(i)=CacFitNess(Energy,Time,MissError,overSpeed,jerk);
   end
   for i=1:setNum
       if indiviFitNess(i)>maxFitNessForIndivi(i)
              %计算个体最大适应度和最适应位置
              maxFitNessForIndivi(i)=indiviFitNess(i);
              indiviMaxFitPos(i,:)=Population(i,:);
              if indiviFitNess(i)>maxFitNessForSocity
                 %更新群体最大适应度和最适应位置
                 maxFitNessForSocity=indiviFitNess(i);
                 socityMaxFitPos=Population(i,:);
              end
       end
   end
    maxSocityFitNessInIter=[maxSocityFitNessInIter,maxFitNessForSocity];
    %保存旧的种群
    OldPopulation=Population;
    OldVelocity=Velocity;
```

```
    %产生新的种群
    [Population,Velocity]=PSOIterate(Population,Velocity,socityMaxFitPos,indiviMaxFitPos);
    Population=sort(Population,2);%重新排序一下X
    %修改惯性因子
    omiga=omiga*(1-iter/iteraNum*0.6);
    disp(iter);
end
    res=socityMaxFitPos;
    save('Result.mat','maxSocityFitNessInIter','socityMaxFitPos');
    figure('Name','适应度变化');
    plot(maxSocityFitNessInIter);
end

function [E,t]=CacMinTime()
%计算最小运行时间,结果作为制动限速曲线的参考
%返回值：能耗和时间
global TRAINWGH STARTPOINT ENDPOINT EMAX TMSTEPLEN Tmin Tmax;
dt=TMSTEPLEN;
dv=0.001; %允许速度误差
M=TRAINWGH;
%添加第一个点
sCurve=[STARTPOINT];
vCurve=[0];
ECurve=[0];
S=STARTPOINT;
v=0.01;
t=0;
E=0;
Force=[0];
Jerk=0;
while(S<ENDPOINT&&v>0)
    vLimit = sqrt(2 * (ENDPOINT - S) * 0.8);
    vLimit = min(SpeedLimitBrake(S+v*dt) - 0.03,vLimit);%采样误差，要加v*dt;
    if v<vLimit-dv
        %在限速曲线误差之下
        Fa=TrateForce(v);
        acc=(Fa-AntiForce(v,S))/M;
        v=v+acc*dt;
        E=E+Fa*(vCurve(length(vCurve))+v)/2*dt;
        Force=[Force,Fa];
        Jerk=Jerk+abs(acc);
    elseif v>vLimit-dv&&v<vLimit-0.03
        %在限速曲线误差范围内,尽量保持匀速
        Fanti=AntiForce(v,S);
        if(TrateForce(v)<Fanti)
            %上坡，阻力大于牵引力
            Fa=TrateForce(v);
```

```
            acc=(Fa-Fanti)/M;
            E=E+Fa*(vCurve(length(vCurve))+v)/2*dt;
            Force=[Force,Fa];
            Jerk=Jerk+abs(acc);
        elseif(Fanti>0)
            %上坡，可以保持匀速
            acc=0;%如果牵引力能够将列车保持在巡航状态
            E=E+abs(Fanti)*(vCurve(length(vCurve))+v)/2*dt;
            Force=[Force,abs(Fanti)];
        elseif(abs(Fanti)<BrakeForce(v))
            %下坡，可以保持匀速
            acc=0;
            Force=[Force,0];
        elseif(abs(Fanti)>BrakeForce(v))
            %下坡，不能够制动保持匀速
            acc=(-1*BrakeForce(v)-Fanti)/M;
            Force=[Force,0];
            Jerk=Jerk+abs(acc);
        end
        v=v+acc*dt;
    elseif(v>=vLimit-0.02)
        %接触到制动曲线，进行制动
%         acc=(-1*BrakeForce(v)-AntiForce(v,S))/M;
        acc=(-1*BrakeForce(v)-AntiForce(v,S))/M;
        v=v+acc*dt;
        %E=E+BrakeForce(v)*(vCurve(length(vCurve))+v)/2*dt;
        Force=[Force,0];
        Jerk=Jerk+abs(acc);
    end
    vCurve=[vCurve,v];
    S=S+(vCurve(length(vCurve)-1)+v)/2*dt;
    sCurve=[sCurve,S];
    ECurve=[ECurve,E];
    t=t+dt;
end
EMAX=E;
Tmin=t;
Tmax=1.5*Tmin;
h=figure('Name','考虑制动最小运行时间');%打开新窗口
plotSpeedLimit();
plotRoadGrad();
hold on;
vCurve = vCurve * 3.6 ; %转换成km/h
plot(sCurve,vCurve,'Marker','o');
xlabel('行驶距离(m)');
ylabel('行驶速度(km/h)');
end
```

附录 E　问题二 Python 代码

5.1　最短时间

```
   import math
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
import time

L = 5144.7 # 总距离（两站点间距）
Vmax = 100 / 3.6 # 最大行驶速度（单位为 m/s）
M = 176.3 # 由于力的单位都是 kN，此处质量的单位也应是 T，这样 F/M 时才能得到单位为 m/s^2 的加速度
r = 1.08 # 质量旋转换算因数
rM = r * M # 等效质量
Fmax = 310
Bmax = 760 # 牵引制动力 F 的范围是 -760  F  310
t = 0.01 # 时间步长
g = 9.80665
f = lambda v: 2.0895 + 0.0098 * v + 0.006 * v**2 # 阻力随速度变化的函数
# 所有力的单位都是 KN

# 电机动态特性
TrackionMAX = lambda v: 310 if 0 <= v <= 10 else 310 * 10 / v
BrakingMAX = lambda v: 260 if 0 <= v <= 17 else 260 * 17 / v
TractionPowerCoefficient = 1 / 0.9
RegenerativeCoefficient = 0.6

# 限速路段
SpeedLimitStart = 4259.1
SpeedLimitEnd = 4960
SpeedLimit = 86 / 3.6


# 坡度
def grad(x: float) -> float:
    # 坡道阻力 Fi = Gsin   Gtan  = Gi（i为坡度）
    # 坡度常用百分号或千分号为单位
    if x <= 198.966:
        i = 0.0617284

    elif 198.966 < x <= 739.018:
        i = 16.2346
```

```
    elif 739.018 < x < 2188.63:
        i = 0.679012

    elif 2188.63 <= x < 2217.05:
        i = 0.555556

    elif 2217.05 <= x < 2870.8:
        i = -9.81481

    elif 2870.8 <= x < 4178.29:
        i = -3.02469

    elif 4178.29 <= x <= 4604.65:
        i = -20.1852

    elif 4604.65 < x <= 4803.62:
        i = 3.02469

    elif 4803.62 < x:
        i = 20.3086

    return i / 1000 # 本题中坡度使用千分号作为单位


Fi = lambda x: M * g * grad(x) # 坡度力


# 计算平均值工具函数
def avg(*vals):
    n = 0
    sum = 0
    for val in vals:
        sum += val
        n += 1
    return sum / n


def showStage():
    print("Stage:", stage)


# 计算减速过程需要的用时和距离（考虑阻力随速度变化，传入当前速度、目标速度和制动力）
def decelerateDistanceTime(
    currentSpeed: float, targetSpeed: float, brakeForcePlus: float = Bmax
):
    timeConsumption = 0
```

```
    v = [currentSpeed]
    x = [0]
    while v[-1] > targetSpeed:
        v.append(v[-1] - (f(v[-1]) + brakeForcePlus) / M * t)
        x.append(x[-1] + avg(v[-2], v[-1]) * t)
        timeConsumption += t
    distance = x[-1]
    # print("timeConsumption:", timeConsumption)
    # print("V:", v)
    # print("X:", x)
    return distance, timeConsumption


# 计算加速过程需要的用时和距离（考虑阻力随速度变化，传入当前速度、目标速度和牵引力）
def accelerateDistanceTime(
    currentSpeed: float, targetSpeed: float, trackForcePlus: float = Fmax
):
    timeConsumption = 0
    v = [currentSpeed]
    x = [0]
    while v[-1] < targetSpeed:
        v.append(v[-1] + (trackForcePlus - f(v[-1])) / rM * t)
        x.append(x[-1] + avg(v[-2], v[-1]) * t)
        timeConsumption += t
    distance = x[-1]
    # print("timeConsumption:", timeConsumption)
    # print("V:", v)
    # print("X:", x)
    return distance, timeConsumption

def getConstanSpeed(expectedTravelTime):
    lowSpeed = 100 / 3.6 * 0.2
    highSpeed = 100 / 3.6
    totalTravelTime = float("inf")

    while abs(totalTravelTime - expectedTravelTime) >= 0.01:
        constantSpeed = avg(lowSpeed, highSpeed)

        distanceA, timeA = accelerateDistanceTime(0, constantSpeed)
        distanceC, timeC = decelerateDistanceTime(constantSpeed, 0)
        distanceB = L - distanceA - distanceC
        timeB = distanceB / constantSpeed
        totalTravelTime = timeA + timeB + timeC

        # 列车速度太快了，早于预期时间到达
        if totalTravelTime < expectedTravelTime:
            highSpeed = constantSpeed
```

```
        # 列车速度太慢了，晚于预期时间到达
        else:
            lowSpeed = constantSpeed

    print(constantSpeed, totalTravelTime)
    return constantSpeed


def getLatestBrakingX(lX, rX, endX, startSpeed, endSpeed):
    actualX = 0

    while abs(actualX - endX) >= 0.01:
        initX = avg(lX, rX)
        x = [initX]
        v = [startSpeed]
        T = 0

        while v[-1] > endSpeed:
            v.append(v[-1] - (BrakingMAX(v[-1]) + f(v[-1]) + Fi(x[-1])) / rM * t)
            x.append(x[-1] + avg(v[-2], v[-1]) * t)
            T += t
        actualX = x[-1]
        # print(initX, actualX, T)

        if actualX < endX:
            lX = initX
        else:
            rX = initX

    print(initX, actualX, T)
    return initX


# getLatestBrakingX(lX=3000, rX=4259, endX=4259, startSpeed=100 / 3.6, endSpeed=86 / 3.6)
# getLatestBrakingX(lX=4000, rX=5144.7, endX=5144.7, startSpeed=86 / 3.6, endSpeed=0)

# 开始计算问题。Vmax是列车最大行驶速度。
Vmax = 100/3.6
# Vmax = getConstanSpeed(shortestArrivalTime + 10)
# Vmax = getConstanSpeed(shortestArrivalTime + 20)
# Vmax = getConstanSpeed(shortestArrivalTime + 50)
# Vmax = getConstanSpeed(shortestArrivalTime + 150)
# Vmax = getConstanSpeed(shortestArrivalTime + 300)

brakePoint1 = getLatestBrakingX(
    lX=3000, rX=4259, endX=4259, startSpeed=100 / 3.6, endSpeed=86 / 3.6
)
```

```
brakePoint2 = getLatestBrakingX(
    lX=4000, rX=5144.7, endX=5144.7, startSpeed=86 / 3.6, endSpeed=0
)

V_Lst = [0] # 速度
X_Lst = [0] # 行驶距离
F_Lst = [0] # 牵引制动力
f_Lst = [0] # 基本阻力
Fi_Lst = [0] # 坡度阻力
a_Lst = [0] # 加速度
W_Lst = [0] # 能耗

# 初始化条件
F_Lst[0] = TrackionMAX(0)
f_Lst[0] = f(V_Lst[0])
Fi_Lst[0] = Fi(0)
a_Lst[0] = (F_Lst[0] - f_Lst[0]) / rM
ACTargetSpeed = Vmax
DETargetSpeed = 0

# 一共有四个阶段 (stage)
# AC (Accelerate) - 加速
# CO (Constant) - 匀速
# DE (Decelerate) - 减速
# AR (Arrival) - 到达
stage = "AC"
print("Stage:", stage)

# 对程序开始计时
startTime = time.time()

# 未到达时迭代计算
while stage != "AR":
    if len(X_Lst) > 100000: # 防止程序出错无限循环 (正常运行时不会用到, 方便出错调试而已)
        break

    # 上个微元的状态, 提高代码可读性。
    a = a_Lst[-1]
    F0 = F_Lst[-1]
    v0 = V_Lst[-1]
    x0 = X_Lst[-1]

    # 开始计算本微元。先算本微元速度。
    # 从 AC 和 DE 状态离开转成 CO 和 AR 状态会自动判断
    # 出于列车寿命的维护考虑, AC 和 DE 阶段不紧邻
    if stage == "AC":
        if v0 + a * t < ACTargetSpeed:
```

```
            v = v0 + a * t
        elif v0 + a * t >= ACTargetSpeed:
            v = ACTargetSpeed
            stage = "CO"
            showStage()

    if stage == "DE":
        if v0 + a * t > DETargetSpeed:
            v = v0 + a * t
        elif v0 + a * t <= DETargetSpeed:
            v = DETargetSpeed
            stage = "CO" if DETargetSpeed != 0 else "AR"
            showStage()


    dx = avg(v0, v) * t # 上微元到本微元行驶的距离
    x = x0 + dx # 本微元距离原点的距离
    _f = f(v) # 摩擦力
    _Fi = Fi(x) # 本微元的坡度力

    # 切换列车状态（阶段）
    if brakePoint1 - Vmax * t < x < brakePoint1 + Vmax * t and stage == "CO":
        stage = "DE"
        print("x:", x)
        showStage()
        DETargetSpeed = 86 / 3.6

    if brakePoint2 - 86 / 3.6 * t < x and stage == "CO":
        stage = "DE"
        print("x:", x)
        showStage()
        DETargetSpeed = 0

    # 根据不同的阶段决定列车的牵引制动力
    if stage == "AC": # 加速时
        F = TrackionMAX(v) # 最大牵引力
        a = (F - _f - _Fi) / rM
    elif stage == "CO": # 匀速时
        F = _f + _Fi # 静力平衡状态
        a = 0
        print(a)
    elif stage == "DE": # 减速时
        F = -BrakingMAX(v) # 最大制动力
        a = (F - _f - _Fi) / rM

    # 能耗计算
    dW = F0 * dx * 1 / 0.9 if F0 > 0 else F0 * dx * 0.6
```

```
    # 将本微元的各项数据记录下来
    V_Lst.append(v)
    X_Lst.append(x) # 目前已经行驶的距离
    F_Lst.append(F) # 牵引制动力
    f_Lst.append(_f) # 基本阻力
    Fi_Lst.append(_Fi)
    a_Lst.append(a)
    W_Lst.append(W_Lst[-1] + dW) # 目前已经消耗的总能量

# 程序运行完毕，结束计时
endTime = time.time()
t_Lst = [_ * t for _ in range(0, len(X_Lst))]

# 展示结果
print(f"能耗: {int(W_Lst[-1])/3600} kW·h ({int(W_Lst[-1])} kJ) ")
print(f"列车最高行驶速度: {Vmax*3.6} km/h")
print(f"程序运行时间: {endTime-startTime} s")
print(f"列车行驶时间: {(len(X_Lst) - 1)*t} s")
# 由于列表太长，为方便调试，每10s的微元进行一次罗列
print("V_Lst:", V_Lst[::1000])
print("X_Lst:", X_Lst[::1000])
print("F_Lst:", F_Lst[::1000])
print("f_Lst:", f_Lst[::1000])
print("Fi_Lst:", Fi_Lst[::1000])
print("a_Lst:", a_Lst[::1000])
print("W_Lst:", W_Lst[::1000])

# Matplotlib 画图
fig = plt.figure(figsize=(14, 9))
ax1 = fig.add_subplot(2, 2, 1)
ax2 = fig.add_subplot(2, 2, 2)
ax3 = fig.add_subplot(2, 2, 3)
ax4 = fig.add_subplot(2, 2, 4)

V_Lst = [v * 3.6 for v in V_Lst]
ax1.plot(X_Lst, V_Lst)
ax1.set_ylabel("Speed (km/h)")
ax1.set_xlabel("Distance (m)")

ax2.plot(X_Lst, F_Lst)
ax2.set_ylabel("Tractive / Braking Force (kN)")
ax2.set_xlabel("Distance (m)")

ax3.plot(X_Lst, t_Lst)
ax3.set_ylabel("Time (s)")
ax3.set_xlabel("Distance (m)")
```

```
W_Lst = [w / 3600 for w in W_Lst]
ax4.plot(X_Lst, W_Lst)
ax4.set_ylabel("Power Consumption (kW · h)")
ax4.set_xlabel("Distance (m)")

plt.show()

totalf_Lst=[f_Lst[i] + Fi_Lst[i] for i in range(len(f_Lst))]
plt.plot(X_Lst,f_Lst,color='green',label='Train Resistance',linestyle='--')
plt.plot(X_Lst,Fi_Lst,color='blue',label='Grade Resistance',linestyle='--')
plt.plot(X_Lst,totalf_Lst,color='red',label='Total Resistance')
plt.legend()
plt.ylabel("Resistance (kN)")
plt.xlabel("Distance (m)")
plt.show()
```

5.2 惰行优化

```
import math
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
import time
L = 5144.7 # 总距离（两站点间距）
Vmax = 100 / 3.6 # 最大行驶速度（单位为 m/s）
M = 176.3 # 由于力的单位都是 kN，此处质量的单位也应是 T，这样 F/M 时才能得到单位为 m/s^2 的加速度
r = 1.08 # 质量旋转换算因数
rM = r * M # 等效质量
Fmax = 310
Bmax = 760 # 牵引制动力 F 的范围是 -760  F  310
t = 0.01 # 时间步长
g = 9.80665
f = lambda v: 2.0895 + 0.0098 * v + 0.006 * v**2 # 阻力随速度变化的函数
# 所有力的单位都是 KN

# 电机动态特性
TrackionMAX = lambda v: 310 if 0 <= v <= 10 else 310 * 10 / v
BrakingMAX = lambda v: 260 if 0 <= v <= 17 else 260 * 17 / v
TractionPowerCoefficient = 1 / 0.9
RegenerativeCoefficient = 0.6

# 限速路段
SpeedLimitStart = 4259.1
SpeedLimitEnd = 4960
```

```
SpeedLimit = 86 / 3.6

#限速下坡路段
LengthLimit=4604.65
# 坡度
def grad(x: float) -> float:
    # 坡道阻力 Fi = Gsin    Gtan  = Gi (i为坡度)
    # 坡度常用百分号或千分号为单位
    if x <= 198.966:
        i = 0.0617284

    elif 198.966 < x <= 739.018:
        i = 16.2346

    elif 739.018 < x < 2188.63:
        i = 0.679012

    elif 2188.63 <= x < 2217.05:
        i = 0.555556

    elif 2217.05 <= x < 2870.8:
        i = -9.81481

    elif 2870.8 <= x < 4178.29:
        i = -3.02469

    elif 4178.29 <= x <= 4604.65:
        i = -20.1852

    elif 4604.65 < x <= 4803.62:
        i = 3.02469

    elif 4803.62 < x:
        i = 20.3086

    return i / 1000 # 本题中坡度使用千分号作为单位


Fi = lambda x: M * g * grad(x) # 坡度力

# 计算平均值工具函数
def avg(*vals):
    n = 0
    sum = 0
    for val in vals:
        sum += val
        n += 1
```

```
    return sum / n


def showStage():
    print("Stage:", stage)


# 计算减速过程需要的用时和距离（考虑阻力随速度变化，传入当前速度、目标速度和制动力）
def decelerateDistanceTime(
    currentSpeed: float, targetSpeed: float, brakeForcePlus: float = Bmax
):
    timeConsumption = 0
    v = [currentSpeed]
    x = [0]
    while v[-1] > targetSpeed:
        v.append(v[-1] - (f(v[-1]) + brakeForcePlus) / M * t)
        x.append(x[-1] + avg(v[-2], v[-1]) * t)
        timeConsumption += t
    distance = x[-1]
    # print("timeConsumption:", timeConsumption)
    # print("V:", v)
    # print("X:", x)
    return distance, timeConsumption


# 计算加速过程需要的用时和距离（考虑阻力随速度变化，传入当前速度、目标速度和牵引力）
def accelerateDistanceTime(
    currentSpeed: float, targetSpeed: float, trackForcePlus: float = Fmax
):
    timeConsumption = 0
    v = [currentSpeed]
    x = [0]
    while v[-1] < targetSpeed:
        v.append(v[-1] + (trackForcePlus - f(v[-1])) / rM * t)
        x.append(x[-1] + avg(v[-2], v[-1]) * t)
        timeConsumption += t
    distance = x[-1]
    # print("timeConsumption:", timeConsumption)
    # print("V:", v)
    # print("X:", x)
    return distance, timeConsumption

def getConstanSpeed(expectedTravelTime):
    lowSpeed = 100 / 3.6 * 0.2
    highSpeed = 100 / 3.6
    totalTravelTime = float("inf")
```

```
    while abs(totalTravelTime - expectedTravelTime) >= 0.01:
        constantSpeed = avg(lowSpeed, highSpeed)

        # 要对 accelerateDistanceTime 和 decelerateDistanceTime 传力进去
        distanceA, timeA = accelerateDistanceTime(0, constantSpeed)
        distanceC, timeC = decelerateDistanceTime(constantSpeed, 0)
        distanceB = L - distanceA - distanceC
        timeB = distanceB / constantSpeed
        totalTravelTime = timeA + timeB + timeC

        # 列车速度太快了，早于预期时间到达
        if totalTravelTime < expectedTravelTime:
            highSpeed = constantSpeed
        # 列车速度太慢了，晚于预期时间到达
        else:
            lowSpeed = constantSpeed

    return constantSpeed


def getLatestBrakingX(lX, rX, endX, startSpeed, endSpeed, brakeForceFactor=1):
    actualX = 0

    while abs(actualX - endX) >= 0.01:
        initX = avg(lX, rX)
        x = [initX]
        v = [startSpeed]
        T = 0

        while v[-1] > endSpeed:
            v.append(
                v[-1]
                - (BrakingMAX(v[-1]) * brakeForceFactor + f(v[-1]) + Fi(x[-1])) / rM * t
            )
            x.append(x[-1] + avg(v[-2], v[-1]) * t)
            T += t
        actualX = x[-1]
        # print(initX, actualX, T)

        if actualX < endX:
            lX = initX
        else:
            rX = initX

    # print(initX, actualX, T)
    return initX
```

```
# getLatestBrakingX(lX=3000, rX=4259, endX=4259, startSpeed=100 / 3.6, endSpeed=86 / 3.6)
# getLatestBrakingX(lX=4000, rX=5144.7, endX=5144.7, startSpeed=86 / 3.6, endSpeed=0)

# 开始计算问题。Vmax是列车最大行驶速度。

lVmax = 100 / 3.6 * 0.2
rVmax = 100 / 3.6
shortestArrivalTime = 209.6
expectedArrivalTime = shortestArrivalTime + 10
expectedArrivalTime = 260
actualArrivalTime = float("inf")

trackForceFactor = 1
brakeForceFactor = 1

# 对程序开始计时
startTime = time.time()

while (
    abs(actualArrivalTime - expectedArrivalTime) >= expectedArrivalTime * 0.001
): # 误差不大于等于千分之一，即总行驶时长为 500 秒时，误差不大于等于 0.5 秒
    Vmax = avg(lVmax, rVmax)

    brakePointSpeedLimit = (
        getLatestBrakingX(
            lX=0,
            rX=SpeedLimitStart,
            endX=SpeedLimitStart,
            startSpeed=Vmax,
            endSpeed=SpeedLimit,
            brakeForceFactor=brakeForceFactor,
        )
        if Vmax > SpeedLimit
        else -1
    )

    brakePointArrival = getLatestBrakingX(
        lX=0,
        rX=L,
        endX=L,
        startSpeed=Vmax if Vmax <= SpeedLimit else SpeedLimit,
        endSpeed=0,
        brakeForceFactor=brakeForceFactor,
    )

    V_Lst = [0] # 速度
```

```
X_Lst = [0] # 行驶距离
F_Lst = [0] # 牵引制动力
f_Lst = [0] # 基本阻力
Fi_Lst = [0] # 坡度阻力
a_Lst = [0] # 加速度
W_Lst = [0] # 能耗

# 初始化条件
F_Lst[0] = TrackionMAX(0)
f_Lst[0] = f(V_Lst[0])
Fi_Lst[0] = Fi(0)
a_Lst[0] = (F_Lst[0] - f_Lst[0]) / rM
ACTargetSpeed = Vmax
DETargetSpeed = 0

# 一共有四个阶段 (stage)
# AC (Accelerate) - 加速
# CO (Constant) - 匀速
# DE (Decelerate) - 减速
# AR (Arrival) - 到达
stage = "AC"
# print("Stage:", stage)

# 未到达时迭代计算
while stage != "AR":
    if len(X_Lst) > 100000: # 防止程序出错无限循环 (正常运行时不会用到, 方便出错调试而已)
        break

    # 上个微元的状态, 提高代码可读性。
    a = a_Lst[-1]
    F0 = F_Lst[-1]
    v0 = V_Lst[-1]
    x0 = X_Lst[-1]

    # 开始计算本微元。先算本微元速度。
    # 从 AC 和 DE 状态离开转成 CO 和 AR 状态会自动判断
    # 出于列车寿命的维护考虑, AC 和 DE 阶段不紧邻
    if stage == "AC":
        if v0 + a * t < ACTargetSpeed:
            v = v0 + a * t
        elif v0 + a * t >= ACTargetSpeed:
            v = ACTargetSpeed
            stage = "CR"
            # showStage()

    if stage == "DE":
        if v0 + a * t > DETargetSpeed:
```

```
            v = v0 + a * t
        elif v0 + a * t <= DETargetSpeed:
            v = DETargetSpeed
            stage = "CR" if DETargetSpeed != 0 else "AR"
            # showStage()
    if stage =="CR":
       v = v0 +a*t

    dx = avg(v0, v) * t # 上微元到本微元行驶的距离
    x = x0 + dx # 本微元距离原点的距离
    _f = f(v) # 摩擦力
    _Fi = Fi(x) # 本微元的坡度力

    # 切换列车状态（阶段）
    # 进入 AC 或 DE 状态需要条件判断设置
    # 这两个条件判断得改，现在差了十几米远
    if (
        brakePointSpeedLimit - v * t < x < brakePointSpeedLimit + v * t
        and stage == "CR"
    ):
        stage = "DE"
        # print("x:", x)
        # showStage()
        DETargetSpeed = SpeedLimit

    if brakePointArrival - v * t < x and (stage == "CR" or stage =="CO"):
        stage = "DE"
        # print("x:", x)
        # showStage()
        DETargetSpeed = 0

    if SpeedLimit-a*t<v<SpeedLimit+a*t and stage=="CR" and x>brakePointSpeedLimit:
       stage="CO"
    if LengthLimit -v*t<x<LengthLimit+v*t and stage =="CO":
       stage="CR"
    # 根据不同的阶段决定列车的牵引制动力
    if stage == "AC": # 加速时
        F = TrackionMAX(v) * trackForceFactor # 最大牵引力
        a = (F - _f - _Fi) / rM
    elif stage == "CR": # 惰行时
         # F = _f + _Fi # 静力平衡状态
        F = 0
        a = -(_f + _Fi)/rM
    elif stage == "CO": # 匀速时
        F = _f + _Fi # 静力平衡状态
        a = 0
    elif stage == "DE": # 减速时
```

```
            F = -BrakingMAX(v) * brakeForceFactor # 最大制动力
            a = (F - _f - _Fi) / rM

        # 能耗计算
        dW = F0 * dx * 1 / 0.9 if F0 > 0 else F0 * dx * 0.6
        # 将本微元的各项数据记录下来
        V_Lst.append(v)
        X_Lst.append(x) # 目前已经行驶的距离
        F_Lst.append(F) # 牵引制动力
        f_Lst.append(_f) # 基本阻力
        Fi_Lst.append(_Fi)
        a_Lst.append(a)
        W_Lst.append(W_Lst[-1] + dW) # 目前已经消耗的总能量

    actualArrivalTime = (len(X_Lst) - 1) * t

    if actualArrivalTime < expectedArrivalTime:
        rVmax = Vmax
    else:
        lVmax = Vmax


# 程序运行完毕，结束计时
endTime = time.time()
t_Lst = [_ * t for _ in range(0, len(X_Lst))]

# 展示结果
print(f"能耗: {int(W_Lst[-1])/3600} kW · h ({int(W_Lst[-1])} kJ) ")
print(f"列车最高行驶速度: {Vmax*3.6} km/h")
print(f"程序运行时间: {endTime-startTime} s")
print(f"列车行驶时间: {(len(X_Lst) - 1)*t} s")
# 由于列表太长，为方便调试，每10s的微元进行一次罗列
print("V_Lst:", V_Lst[::1000])
print("X_Lst:", X_Lst[::1000])
print("F_Lst:", F_Lst[::1000])
print("f_Lst:", f_Lst[::1000])
print("Fi_Lst:", Fi_Lst[::1000])
print("a_Lst:", a_Lst[::1000])
print("W_Lst:", W_Lst[::1000])

# Matplotlib 画图
fig = plt.figure(figsize=(14, 15))
axSpeed = fig.add_subplot(3, 2, 1)
axTrackBrakeForce = fig.add_subplot(3, 2, 2)
axResistance = fig.add_subplot(3, 2, 3)
axTime = fig.add_subplot(3, 2, 4)
axPower = fig.add_subplot(3, 2, 5)
```

```
x_values = [0, 4259, 4259.1, 4960, 4960.1, 5144.7]
y_values = [100, 100, 86, 86, 100, 100]
V_Lst = [v * 3.6 for v in V_Lst]
axSpeed.plot(X_Lst, V_Lst,color="blue")
axSpeed.plot(x_values,y_values,color="red")
axSpeed.set_ylabel("Speed (km/h)")
axSpeed.set_xlabel("Distance (m)")

axTrackBrakeForce.plot(X_Lst, F_Lst,color="green")
axTrackBrakeForce.set_ylabel("Tractive / Braking Force (kN)")
axTrackBrakeForce.set_xlabel("Distance (m)")

axTime.plot(X_Lst, t_Lst,color="gold")
axTime.set_ylabel("Time (s)")
axTime.set_xlabel("Distance (m)")

W_Lst = [w / 3600 for w in W_Lst]
axPower.plot(X_Lst, W_Lst,color="pink")
axPower.set_ylabel("Power Consumption (kW · h)")
axPower.set_xlabel("Distance (m)")

totalf_Lst = [f_Lst[i] + Fi_Lst[i] for i in range(len(f_Lst))]
axResistance.plot(X_Lst, f_Lst, color="green", label="Train Resistance", linestyle="--")
axResistance.plot(X_Lst, Fi_Lst, color="blue", label="Grade Resistance", linestyle="--")
axResistance.plot(X_Lst, totalf_Lst, color="red", label="Total Resistance")
axResistance.set_ylabel("Resistance (kN)")
axResistance.set_xlabel("Distance (m)")
axResistance.legend()
```

附录 F　问题三 Python 代码

```
import math
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
import time
plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中文标签
plt.rcParams['axes.unicode_minus']=False #用来正常显示负号

#
    假设一辆列车在水平轨道上运行，从站台A运行至站台B，其间距为5144.7m，运行的速度上限为100km/h，列车质量为176.3t，列
    =
    1.08，列车电机的最大牵引力为310KN，机械制动部件的最大制动力为760KN。列车受到的阻力满足Davis阻力方程f
    = 2.0895 + 0.0098v + 0.006v^2，该公式中的速度单位为m/s，阻力单位为KN。
```

```
L = 5144.7 # 总距离（两站点间距）
Vmax = 100 / 3.6 # 最大行驶速度（单位为 m/s）
M = 176.3 # 由于力的单位都是 kN，此处质量的单位也应是 T，这样 F/M 时才能得到单位为 m/s^2 的加速度
r = 1.08 # 质量旋转换算因数
rM = r * M # 等效质量
Fmax = 310
Bmax = 760 # 牵引制动力 F 的范围是 -760  F  310
t = 0.01 # 时间步长
g = 9.80665
f = lambda v: 2.0895 + 0.0098 * v + 0.006 * v**2 # 阻力随速度变化的函数
# 所有力的单位都是 KN

# 电机动态特性
TrackionMAX = lambda v: 310 if 0 <= v <= 10 else 310 * 10 / v
BrakingMAX = lambda v: 260 if 0 <= v <= 17 else 260 * 17 / v
TractionPowerCoefficient = 1 / 0.9
RegenerativeCoefficient = 0.6

# 限速路段
SpeedLimitStart = 4259.1
SpeedLimitEnd = 4960
SpeedLimit = 86 / 3.6
coastingPosition=935.2
LengthLimit=4604.65

# 坡度
def grad(x: float) -> float:
    # 坡道阻力 Fi = Gsin  Gtan  = Gi（i为坡度）
    # 坡度常用百分号或千分号为单位
    if x <= 198.966:
        i = 0.0617284

    elif 198.966 < x <= 739.018:
        i = 16.2346

    elif 739.018 < x < 2188.63:
        i = 0.679012

    elif 2188.63 <= x < 2217.05:
        i = 0.555556

    elif 2217.05 <= x < 2870.8:
        i = -9.81481

    elif 2870.8 <= x < 4178.29:
        i = -3.02469
```

```
    elif 4178.29 <= x <= 4604.65:
        i = -20.1852

    elif 4604.65 < x <= 4803.62:
        i = 3.02469

    elif 4803.62 < x:
        i = 20.3086

    return i / 1000 # 本题中坡度使用千分号作为单位


Fi = lambda x: M * g * grad(x) # 坡度力

# 程序运行计时（弃用）
# def timeit(func):
#     def i(*arg, **kwargs):
#         startTime = time.time()
#         res = func(*arg, **kwargs)
#         endTime = time.time()
#         print(f"Running Time: {endTime-startTime} s")
#         return res

#     return i


# 计算平均值工具函数
def avg(*vals):
    n = 0
    sum = 0
    for val in vals:
        sum += val
        n += 1
    return sum / n


def showStage():
    print("Stage:", stage)


# 计算减速过程需要的用时和距离（考虑阻力随速度变化，传入当前速度、目标速度和制动力）
def decelerateDistanceTime(
    currentSpeed: float, targetSpeed: float, brakeForcePlus: float = Bmax
):
    timeConsumption = 0
    v = [currentSpeed]
```

```
    x = [0]
    while v[-1] > targetSpeed:
        v.append(v[-1] - (f(v[-1]) + brakeForcePlus) / M * t)
        x.append(x[-1] + avg(v[-2], v[-1]) * t)
        timeConsumption += t
    distance = x[-1]
    # print("timeConsumption:", timeConsumption)
    # print("V:", v)
    # print("X:", x)
    return distance, timeConsumption


# 计算加速过程需要的用时和距离（考虑阻力随速度变化，传入当前速度、目标速度和牵引力）
def accelerateDistanceTime(
    currentSpeed: float, targetSpeed: float, trackForcePlus: float = Fmax
):
    timeConsumption = 0
    v = [currentSpeed]
    x = [0]
    while v[-1] < targetSpeed:
        v.append(v[-1] + (trackForcePlus - f(v[-1])) / rM * t)
        x.append(x[-1] + avg(v[-2], v[-1]) * t)
        timeConsumption += t
    distance = x[-1]
    # print("timeConsumption:", timeConsumption)
    # print("V:", v)
    # print("X:", x)
    return distance, timeConsumption


#
    由于题目要求考虑延迟到达的情况，可以通过三种手段减缓抵达时间，1）降低加速时的牵引力；2）降低匀速时的行驶速度；3）
def getConstanSpeed(expectedTravelTime):
    lowSpeed = 100 / 3.6 * 0.2
    highSpeed = 100 / 3.6
    totalTravelTime = float("inf")

    while abs(totalTravelTime - expectedTravelTime) >= 0.01:
        constantSpeed = avg(lowSpeed, highSpeed)

        # 这个地方需要改！！
        # 要对 accelerateDistanceTime 和 decelerateDistanceTime 传力进去
        distanceA, timeA = accelerateDistanceTime(0, constantSpeed)
        distanceC, timeC = decelerateDistanceTime(constantSpeed, 0)
        distanceB = L - distanceA - distanceC
        timeB = distanceB / constantSpeed
        totalTravelTime = timeA + timeB + timeC
```

```
        # 列车速度太快了，早于预期时间到达
        if totalTravelTime < expectedTravelTime:
            highSpeed = constantSpeed
        # 列车速度太慢了，晚于预期时间到达
        else:
            lowSpeed = constantSpeed

    # print(constantSpeed, totalTravelTime)
    return constantSpeed


def getLatestBrakingX(lX, rX, endX, startSpeed, endSpeed, brakeForceFactor=1):
    actualX = 0

    while abs(actualX - endX) >= 0.01:
        initX = avg(lX, rX)
        x = [initX]
        v = [startSpeed]
        T = 0

        while v[-1] > endSpeed:
            v.append(
                v[-1]
                - (BrakingMAX(v[-1]) * brakeForceFactor + f(v[-1]) + Fi(x[-1])) / rM * t
            )
            x.append(x[-1] + avg(v[-2], v[-1]) * t)
            T += t
        actualX = x[-1]
        # print(initX, actualX, T)

        if actualX < endX:
            lX = initX
        else:
            rX = initX

    # print(initX, actualX, T)
    return initX

# 开始计算问题。Vmax是列车最大行驶速度。

lVmax = 100 / 3.6 * 0.2
rVmax = 100 / 3.6
# shortestArrivalTime = 209.6
# expectedArrivalTime = shortestArrivalTime + 10
expectedArrivalTime = 320
delayTime = 60
```

```
delayInformX = 2000
actualArrivalTime = float("inf")

trackForceFactor = 1
brakeForceFactor = 1

# 对程序开始计时
startTime = time.time()

while (
    abs(actualArrivalTime - expectedArrivalTime) >= expectedArrivalTime * 0.001
): # 误差不大于等于千分之一，即总行驶时长为 500 秒时，误差不大于等于 0.5 秒
    Vmax = avg(lVmax, rVmax)

    brakePointSpeedLimit = (
        getLatestBrakingX(
            lX=0,
            rX=SpeedLimitStart,
            endX=SpeedLimitStart,
            startSpeed=Vmax,
            endSpeed=SpeedLimit,
            brakeForceFactor=brakeForceFactor,
        )
        if Vmax > SpeedLimit
        else -1
    )

    brakePointArrival = getLatestBrakingX(
        lX=0,
        rX=L,
        endX=L,
        startSpeed=Vmax if Vmax <= SpeedLimit else SpeedLimit,
        endSpeed=0,
        brakeForceFactor=brakeForceFactor,
    )

    V_Lst = [0] # 速度
    X_Lst = [0] # 行驶距离
    F_Lst = [0] # 牵引制动力
    f_Lst = [0] # 基本阻力
    Fi_Lst = [0] # 坡度阻力
    a_Lst = [0] # 加速度
    W_Lst = [0] # 能耗

    # 初始化条件
    F_Lst[0] = TrackionMAX(0)
    f_Lst[0] = f(V_Lst[0])
```

```
Fi_Lst[0] = Fi(0)
a_Lst[0] = (F_Lst[0] - f_Lst[0]) / rM
ACTargetSpeed = Vmax
DETargetSpeed = 0

# 一共有四个阶段 (stage)
# AC (Accelerate) - 加速
# CO (Constant) - 匀速
# DE (Decelerate) - 减速
# AR (Arrival) - 到达
stage = "AC"
# print("Stage:", stage)

# 未到达时迭代计算
while stage != "AR":
    if len(X_Lst) > 100000: # 防止程序出错无限循环 (正常运行时不会用到, 方便出错调试而已)
        break

    # 上个微元的状态, 提高代码可读性。
    a = a_Lst[-1]
    F0 = F_Lst[-1]
    v0 = V_Lst[-1]
    x0 = X_Lst[-1]

    # 开始计算本微元。先算本微元速度。
    # 从 AC 和 DE 状态离开转成 CO 和 AR 状态会自动判断
    # 出于列车寿命的维护考虑, AC 和 DE 阶段不紧邻
    if stage == "AC":
        if v0 + a * t < ACTargetSpeed:
            v = v0 + a * t
        elif v0 + a * t >= ACTargetSpeed:
            v = ACTargetSpeed
            stage = "CR"
            # showStage()

    if stage == "DE":
        if v0 + a * t > DETargetSpeed:
            v = v0 + a * t
        elif v0 + a * t <= DETargetSpeed:
            v = DETargetSpeed
            stage = "CR" if DETargetSpeed != 0 else "AR"
            # showStage()
    if stage =="CR":
       v = v0 +a*t

    dx = avg(v0, v) * t # 上微元到本微元行驶的距离
    x = x0 + dx # 本微元距离原点的距离
```

```
    _f = f(v) # 摩擦力
    _Fi = Fi(x) # 本微元的坡度力

    # 切换列车状态（阶段）
    # 进入 AC 或 DE 状态需要条件判断设置
    # 这两个条件判断得改，现在差了十几米远
    if (
        brakePointSpeedLimit - v * t < x < brakePointSpeedLimit + v * t
        and stage == "CR"
    ):
        stage = "DE"
        # print("x:", x)
        # showStage()
        DETargetSpeed = SpeedLimit

    if brakePointArrival - v * t < x and (stage == "CR" or stage =="CO"):
        stage = "DE"
        # print("x:", x)
        # showStage()
        DETargetSpeed = 0

    if SpeedLimit-a*t<v<SpeedLimit+a*t and stage=="CR" and x>brakePointSpeedLimit:
       stage="CO"
    # 根据不同的阶段决定列车的牵引制动力
    if stage == "AC": # 加速时
        F = TrackionMAX(v) * trackForceFactor # 最大牵引力
        a = (F - _f - _Fi) / rM
    elif stage == "CR": # 惰行时
         # F = _f + _Fi # 静力平衡状态
        F = 0
        a = -(_f + _Fi)/rM
    elif stage == "CO": # 匀速时
        F = _f + _Fi # 静力平衡状态
        a = 0
    elif stage == "DE": # 减速时
        F = -BrakingMAX(v) * brakeForceFactor # 最大制动力
        a = (F - _f - _Fi) / rM

    # 能耗计算
    dW = F0 * dx * 1 / 0.9 if F0 > 0 else F0 * dx * 0.6

    # 将本微元的各项数据记录下来
    V_Lst.append(v)
    X_Lst.append(x) # 目前已经行驶的距离
    F_Lst.append(F) # 牵引制动力
    f_Lst.append(_f) # 基本阻力
    Fi_Lst.append(_Fi)
```

```
        a_Lst.append(a)
        W_Lst.append(W_Lst[-1] + dW) # 目前已经消耗的总能量

    actualArrivalTime = (len(X_Lst) - 1) * t

    if actualArrivalTime < expectedArrivalTime:
        rVmax = Vmax
    else:
        lVmax = Vmax


for idx, x in enumerate(X_Lst):
    if x > delayInformX:
        delayInformIdx = idx
        break

V_LstInit = V_Lst[:delayInformIdx]
X_LstInit = X_Lst[:delayInformIdx]
F_LstInit = F_Lst[:delayInformIdx]
f_LstInit = f_Lst[:delayInformIdx]
Fi_LstInit = Fi_Lst[:delayInformIdx]
a_LstInit = a_Lst[:delayInformIdx]
W_LstInit = W_Lst[:delayInformIdx]


lConstV = Vmax * 0.2
rConstV = Vmax
expectedArrivalTime += delayTime
actualArrivalTime = float("inf")


while (
    abs(actualArrivalTime - expectedArrivalTime) >= expectedArrivalTime * 0.001
): # 误差不大于等于千分之一，即总行驶时长为 500 秒时，误差不大于等于 0.5 秒
    ConstV = avg(lConstV, rConstV)

    brakePointArrival = getLatestBrakingX(
        lX=0,
        rX=L,
        endX=L,
        startSpeed=Vmax if Vmax <= SpeedLimit else SpeedLimit,
        endSpeed=0,
        brakeForceFactor=brakeForceFactor,
    )

    V_Lst = V_LstInit.copy()
    X_Lst = X_LstInit.copy()
```

```
F_Lst = F_LstInit.copy()
f_Lst = f_LstInit.copy()
Fi_Lst = Fi_LstInit.copy()
a_Lst = a_LstInit.copy()
W_Lst = W_LstInit.copy()

ACTargetSpeed = Vmax
DETargetSpeed = 0

stage = "DE"
DETargetSpeed = ConstV

# 未到达时迭代计算
while stage != "AR":
    if len(X_Lst) > 100000: # 防止程序出错无限循环（正常运行时不会用到，方便出错调试而已）
        break

    # 上个微元的状态，提高代码可读性。
    a = a_Lst[-1]
    F0 = F_Lst[-1]
    v0 = V_Lst[-1]
    x0 = X_Lst[-1]

    # 开始计算本微元。先算本微元速度。
    # 从 AC 和 DE 状态离开转成 CO 和 AR 状态会自动判断
    # 出于列车寿命的维护考虑，AC 和 DE 阶段不紧邻
    if stage == "AC":
        if v0 + a * t < ACTargetSpeed:
            v = v0 + a * t
        elif v0 + a * t >= ACTargetSpeed:
            v = ACTargetSpeed
            stage = "CO"
            # showStage()

    if stage == "DE":
        if v0 + a * t > DETargetSpeed:
            v = v0 + a * t
        elif v0 + a * t <= DETargetSpeed:
            v = DETargetSpeed
            stage = "CR" if DETargetSpeed != 0 else "AR"
            # showStage()
    if stage =="CR":
       v = v0 +a*t

    dx = avg(v0, v) * t # 上微元到本微元行驶的距离
    x = x0 + dx # 本微元距离原点的距离
    _f = f(v) # 摩擦力
```

```
    _Fi = Fi(x) # 本微元的坡度力

    # 切换列车状态（阶段）
    # 进入 AC 或 DE 状态需要条件判断设置
    # 这两个条件判断得改，现在差了十几米远
    if (
        brakePointSpeedLimit - v * t < x < brakePointSpeedLimit + v * t
        and stage == "CR"
    ):
        stage = "DE"
        # print("x:", x)
        # showStage()
        DETargetSpeed = SpeedLimit

    if brakePointArrival - v * t < x and (stage == "CR" or stage =="CO"):
        stage = "DE"
        # print("x:", x)
        # showStage()
        DETargetSpeed = 0

    if coastingPosition-v*t<x<coastingPosition+v*t and stage=="CO":
        stage="CR"

    if SpeedLimit-a*t<v<SpeedLimit+a*t and stage=="CR" and x>brakePointSpeedLimit:
       stage="CO"

    if LengthLimit -v*t<x<LengthLimit+v*t and stage =="CO":
       stage="CR"

    # 根据不同的阶段决定列车的牵引制动力
    if stage == "AC": # 加速时
        F = TrackionMAX(v) * trackForceFactor # 最大牵引力
        a = (F - _f - _Fi) / rM
    elif stage == "CR": # 惰行时
         # F = _f + _Fi # 静力平衡状态
        F = 0
        a = -(_f + _Fi)/rM
    elif stage == "CO": # 匀速时
        F = _f + _Fi # 静力平衡状态
        a = 0
    elif stage == "DE": # 减速时
        F = -BrakingMAX(v) * brakeForceFactor # 最大制动力
        a = (F - _f - _Fi) / rM

    # 能耗计算
    dW = F0 * dx * 1 / 0.9 if F0 > 0 else F0 * dx * 0.6
```

```
        # 将本微元的各项数据记录下来
        V_Lst.append(v)
        X_Lst.append(x) # 目前已经行驶的距离
        F_Lst.append(F) # 牵引制动力
        f_Lst.append(_f) # 基本阻力
        Fi_Lst.append(_Fi)
        a_Lst.append(a)
        W_Lst.append(W_Lst[-1] + dW) # 目前已经消耗的总能量

    actualArrivalTime = (len(X_Lst) - 1) * t

    if actualArrivalTime < expectedArrivalTime:
        rConstV = ConstV
    else:
        lConstV = ConstV


# 程序运行完毕，结束计时
endTime = time.time()
t_Lst = [_ * t for _ in range(0, len(X_Lst))]
```