基于模拟退火算法的列车节能运行优化策略

摘要

本文研究单列车节能优化决策问题。以 x 轴负半轴建立公里标坐标系，符合列车从左到右行驶习惯；统一数据单位和方向；进行数据预处理和线路加算坡度融合；给出**微分方程**的形式列车运行动力学模型，并对模型进行了检验和分析。

针对问题一：首先根据列车运行速度与运行时间和运行能耗的相关关系，建立**单质点模型**，使用**数值积分**的方法模拟列车运行过程，遍历每单位离散时间步长确定列车每时刻的位置、速度、加速度、牵引力、阻力，制动力与能耗信息，从而确定列车加速、匀速、制动的最佳时间分配。最后得到列车以最短时间到达站台 B、在最短时间上分别增加 10s、20s、50s、150s、300s 到达站台 B 的六组曲线。最短时间为 196.959s，程序运行的时间为 0.017s。

针对问题二：在单质点模型的基础上，基于指定路况和电机静动态特性上，通过建立最小化能耗函数和增加固定行驶时间约束、路段限速约束、坡度约束、电机静动态特性约束和储能装置能量平衡约束等约束条件，运用数值积分的方法编程模拟列车的运行过程，利用**模拟退火算法**，最终解得最佳速度曲线。

针对问题三：在单质点模型和模拟退火算法的基础上，设计优化方案考虑速度、牵引制动力、时间和能量消耗。加速阶段以最大牵引力加速至目标速度，维持阶段以目标速度匀速行驶，制动阶段在距离终点前减速以延迟到达，延迟阶段以低速行驶节能。具体数值可计算得出，绘制速度-距离、牵引制动力-距离、时间-距离和能量消耗-距离曲线展示优化效果。

关键字：模拟退火算法 数值积分 节能优化 单质点模型

一、 问题重述

1.1 问题背景

在过去的几十年里，随着我国城市交通电气化进程飞速推进，交通领域成为碳排放大户，其能源消费约占我国终端总能耗的约 10%。为应对气候变化，多方协同交通电动化与低碳城市融合发展，在保证乘客满意度的前提下，对能耗控制、提升城轨系统减碳能力提出了更高的要求。因空气阻力，摩擦，势能变化，位置限速等因素影响，在相同路线的行驶过程中，列车采用不同的行驶策略，通常会有不同的能量和时间消耗。因此，采用最优的行驶策略是减少能耗的关键方法之一。

1.2 问题重述

问题一：列车在水平轨道上从站台 A 运行至站台 B，在站台间距为 5144.7m，运行速度上限为 100km/h，列车质量为 176.3t，列车旋转部件惯性的旋转质量因数$\rho=1.08$，列车电机最大牵引力为 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$f=2.0895+0.0098v+0.006v^2$（该公式中速度单位为 m/是，阻力单位为 KN）的已知条件下，分析并建立列车运行过程的模型，编写程序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，并记录程序运行的时间，画出列车在最短时间内到达站台 B、在最短时间上分别增加 10s、20s、50s、150s、300s 到达站台 B 的曲线。

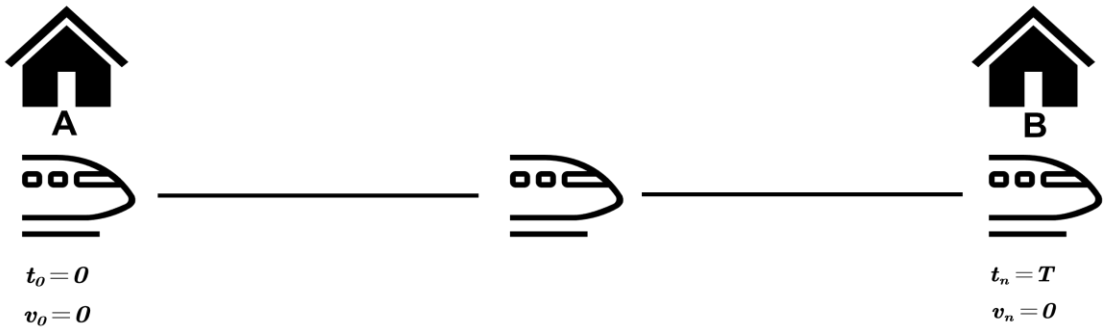

图 1 单列车运行过程

问题二：考虑附件 1,、附件 2 列车实际运行中不同限速，坡度等路况信息以及列车电机动静态特性，假定列车预期运行时间为 T，设计优化模型得到合理的速度轨迹，使行驶过程的能耗降低。根据优化模型，画出列车以最短时间到达站台 B、在最短行驶时间上分别增加 10s、20s、50s、150s、300s 到达站台 B 的曲线。

问题三：考虑列车实际运行中突发状况可能导致列车提前到达站台或延时到达站台，分析列车运行速度轨迹因新的到站时间而产生的变化。假定列车从站台 A 出发，原定于 320s 后到达站台 B，在列车运行至 2000m 位置处因突发事故影响需延迟 60s 到达站台 B 的已知条件下，探究并设计优化方案在保持列车节能运行的同时能快速地得到调整后的优化速度轨迹，作出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。

二、 问题分析

2.1 问题一的分析

问题一首先基于当行驶路程等其他条件一定时，列车运行能耗与列车最大运行速度呈正相关关系和区间运行时间和最大运行速度成负相关关系的前提下，建立单质点模型，通过调整站间的列车最大运行速度确定合理的区间运行时间。其次编写程序，使用数值积分的方法来模拟列车的运行过程，利用离散时间步长（例如 0.01 秒）逐步更新列车的速度和位置，直到到达目标位置。同时记录下每个时间步长的速度和位置，形成速度-距离曲线、牵引制动力-距离曲线、时间-距离曲线和能量消耗-距离曲线的数据。

2.2 问题二的分析

在问题一的基础上，需要利用附件一和附件二的参数值，求出指定路况和电机静动态参数下，得到最低能耗的速度轨迹。本题通过建立最小化能耗目标函数和增加列车在给定时间内到达站台 B、路段限速约束、坡度约束、电机动态特性约束、储能装置能量平衡约束等约束条件，利用问题一建立的数学模型，使用数值积分方法来模拟列车的运行过程，最终解得最佳速度轨迹。

2.3 问题三的分析

在问题一、二的基础上，依据之前建立的数学模型，考虑列车实际运行中突发状况可能导致列车提前到达站台或延时到达站台，分析列车运行速度轨迹因新的到站时间而产生的变化。在单质点模型和模拟退火算法的基础上，设计优化方案考虑速度、牵引制动力、时间和能量消耗。加速阶段以最大牵引力加速至目标速度，维持阶段以目标速度匀速行驶，制动阶段在距离终点前减速以延迟到达，延迟阶段以低速行驶节能。具体数值可计算得出，绘制速度-距离、牵引制动力-距离、时间-距离和能量消耗-距离曲线展示

优化效果。

三、 模型假设

1、假设列车运行过程中，由于列车运行坡道的长度远远大于列车的长度，将列车作为一个质点处理。

2、假设列车进站为车尾完全进站（车尾与车站坐标重合），列车出站为车头离开车站末端。

3、假设列车的牵引和制动力变化是不连续的，可以为允许范围内任意点力的值。

4、假设列车控制不存在牵引和制动同时使用的负载制动工况。

5、假设当坡道附加阻力，曲线附加阻力同时出现时，根据阻力值相等的原则，把列车通过曲线时所产生的附加阻力作为坡道阻力，加上线路实际坡度作为为加算坡度。

6、假设城市轨道交通中列车的行驶属于固定车站间隔、固定运行时长问题，允许设计方案行驶距离略小于给定里程（不超过 1m），允许设计方案行驶时间略小于给定时间（不超过 1s）。

四、 符号说明

符号	意义	单位
R	Davis 阻力	KN
F	牵引力	KN
B	制动力	KN
v	速度	m/s
M	列车质量	kg
t	时间	s
ρ	旋转质量因数	-
Fn	支持力	KN
G	重力	KN
g	重力加速度	m/s^2
η	牵引/再生制动效率	-

i	加速度	m/s^2

五、 模型的建立与求解

5.1 问题一模型建立与求解

5.1.1 模型的建立

对正在水平路面上行驶的列车进行受力分析，列车受重力、支持力、牵引力、制动力、阻力，制动力与牵引力不同时出现

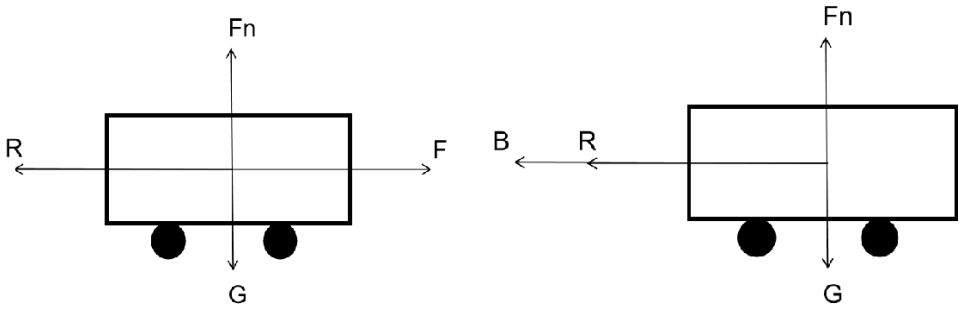

图 2 列车在水平面上受制动力时受力图

列车在牵引阶段做变加速运动，所受阻力与运行速度有关，根据已知条件，列车总阻力由两部分构成，其一为基本阻力，即：

$$R(v)=2.0895+0.0098v+0.006v^2 \tag{1}$$

其二为加速阻力：

$$f=M\rho\frac{dv}{dt} \tag{2}$$

由 Davis 阻力方程和牛顿第二定律可得，

$$M\rho\frac{dv}{dt}=F-R(v)-g(i) \tag{3}$$

$$\frac{dv}{dt}=a \tag{4}$$

$$g(i)=Mi \tag{5}$$

t 时刻对应的的路程为：

$$S_t = \int_t^{t+\Delta t} v dt \tag{6}$$

列车 t 时刻对应的瞬时速度为：

$$v_t = \int_t^{t+\Delta t} \frac{F - R - M\rho a}{m} dt \tag{7}$$

在列车运行过程中，由于列车运行轨道长度远远大于列车长度，故可将列车作为一个质点来处理，建立单质点模型。当行驶路程 S 等其他条件一定时，列车运行能耗与列车最大运行速度呈正相关关系和区间运行时间和最大运行速度成负相关关系，通过调整站间的列车最大运行速度确定合理的区间运行时间。使用数值积分的方法来模拟列车的运行过程，利用离散时间步长 dt（例如 0.01 秒）逐步更新列车的速度和位置，直到到达目标位置，确定列车加速、匀速、制动的最佳时间分配。同时记录下每个时间步长的速度和位置，形成速度-距离曲线、牵引制动力-距离曲线、时间-距离曲线和能量消耗-距离曲线的数据。

5.1.2 模型求解

运用 Python 语言在 vscode 环境下，用编程语言复现上述分析过程并求解画图，代码详见附录 1。运行程序后得到列车从站台 A 到达站台 B 的最短时间为 196.959s。列车以最短时间到达站台 B、在最短时间上分别增加 10s、20s、50s、150s、300s 到达站台 B 的曲线，如下图 3~10 所示。附录 1 程序运行时间为 0.017s。

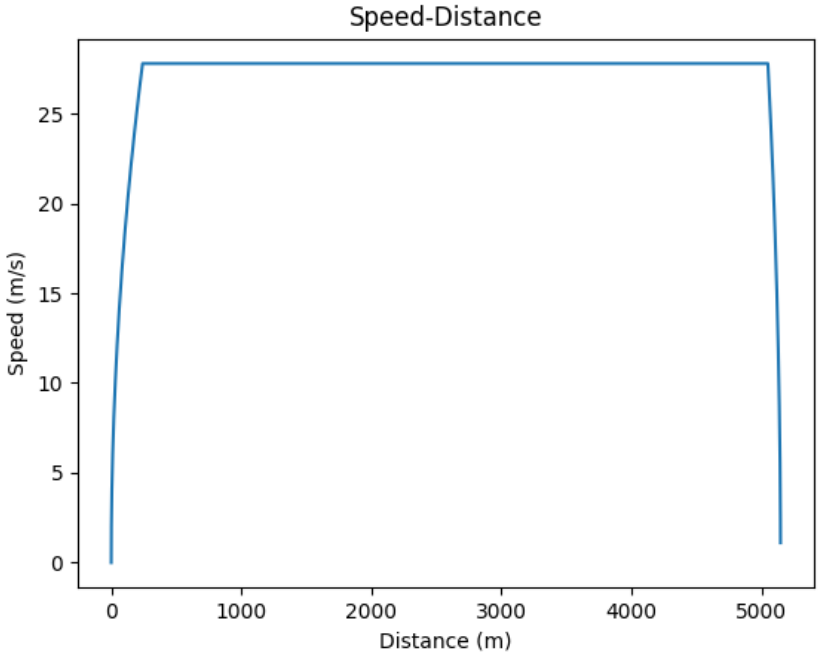

图 3 最短时间的速度-距离曲线

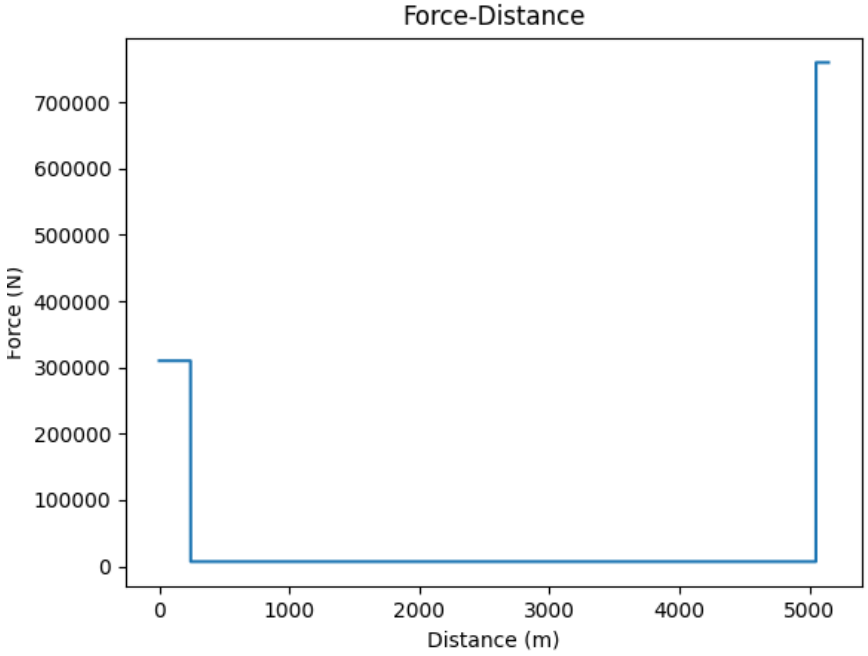

图 4 最短时间的牵引制动力-距离曲线

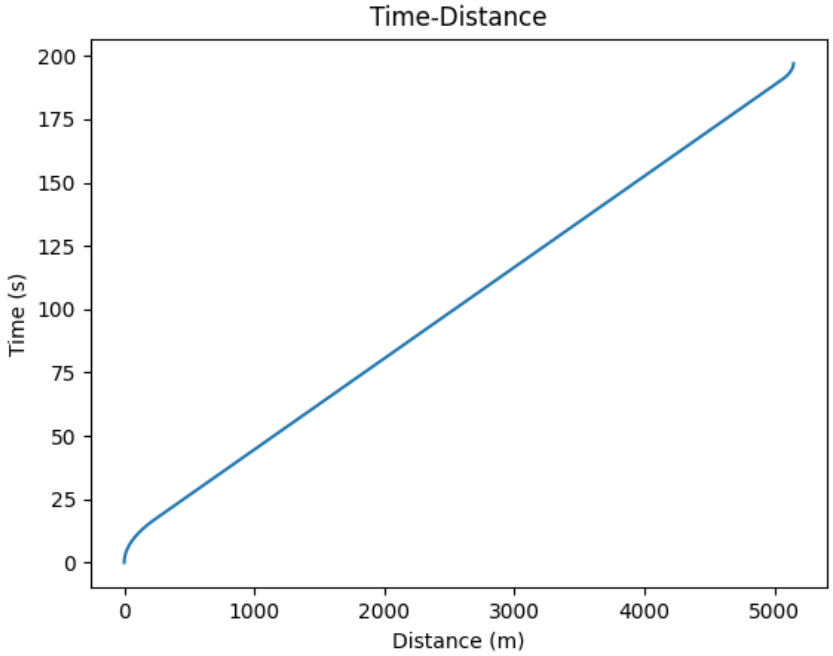

图 5 最短时间的时间-距离曲线

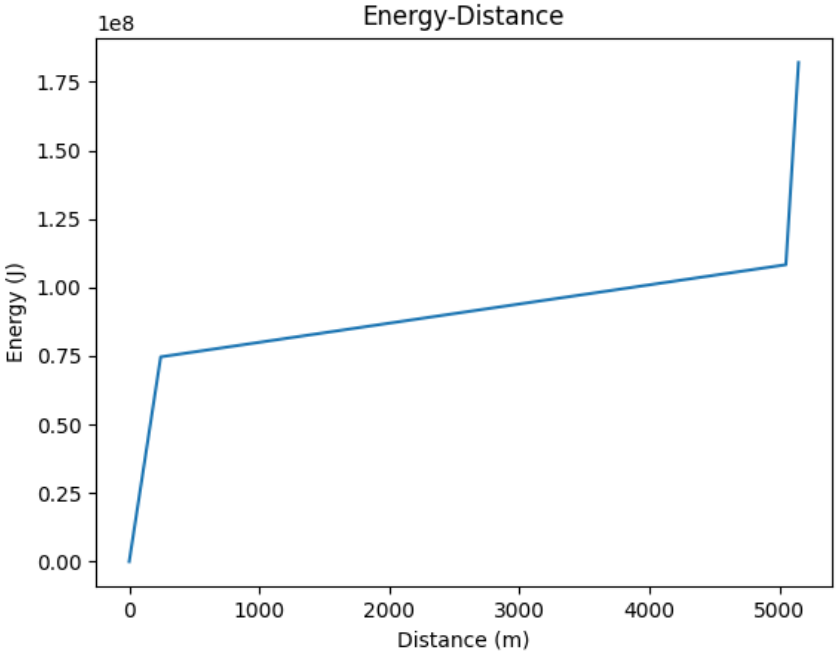

图 6 最短时间的能量消耗-距离曲线

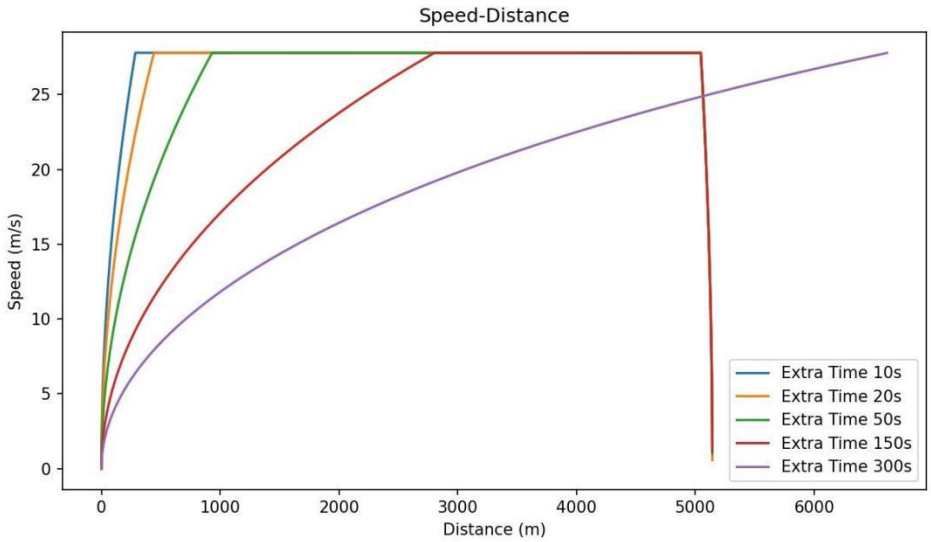

图 7 在最短时间上增加时

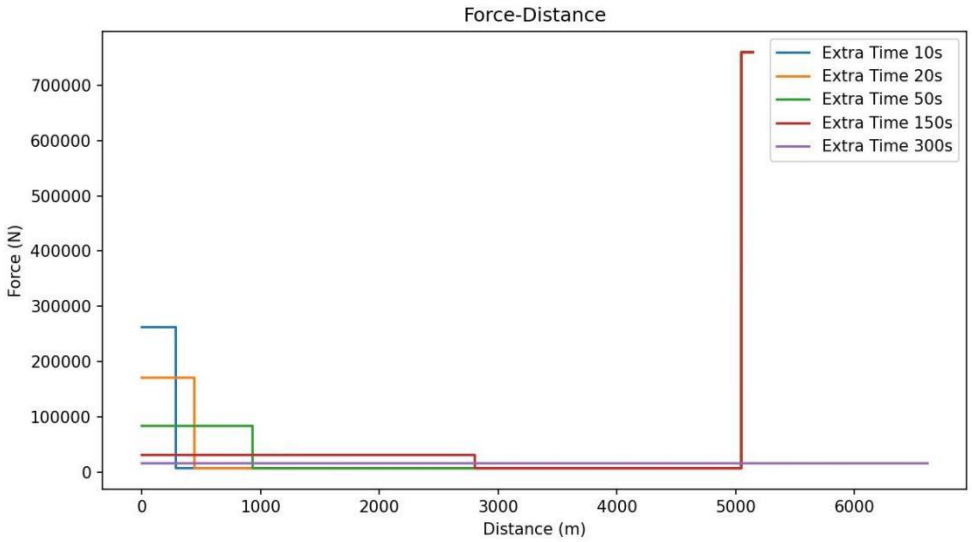

图 8 在最短时间上增加时间的牵引制动力-距离曲线

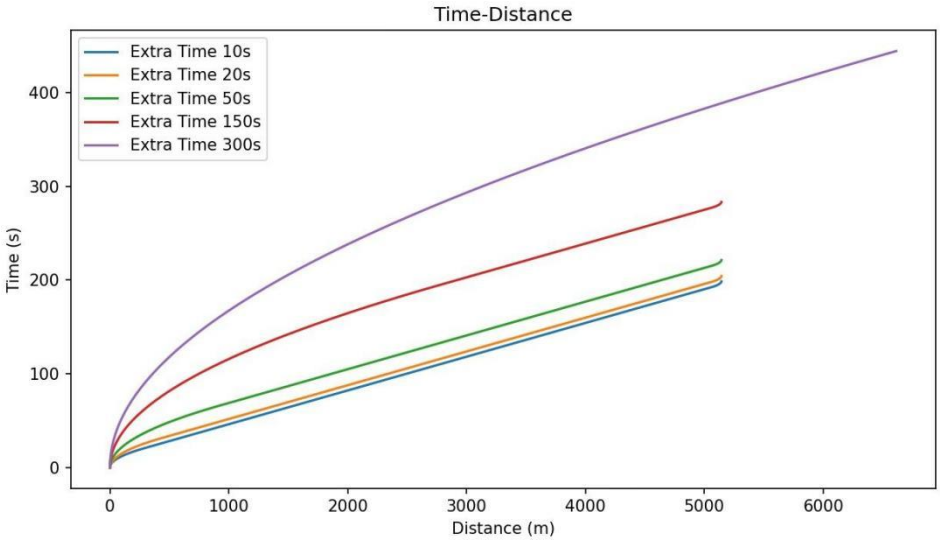

图 9 在最短时间上增加时间的时间-距离曲线

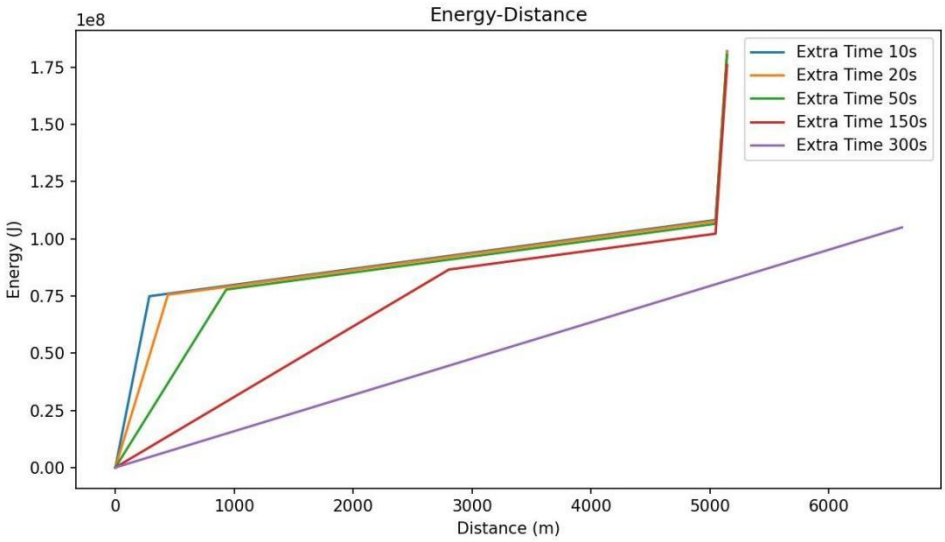

图 10 在最短时间上增加时间的能量消耗-距离曲线

5.2 问题二的模型建立与求解

5.2.1 模型的建立

列车节能操控控制策略分为全力牵引（Full power，FP），部分牵引（Partial power，PP），惰行（Coasting，C），部分制动（Partial braking，PB），全力制动（Full braking，FB）。全力牵引使用在临近陡上坡或限速上升处；部分牵引使用于普通坡道上，使列车保持恒定速度；惰行使用在临近陡下坡、限速下降的区域或其他调整运行时间的区域。部分制动使用在陡下坡处和列车运行速度接近限速时；全力制动使用在制动进站或限速下坡处。[1]

根据相关研究成果，列车节能操作需遵循如下三个基本原则:第一：在列车牵引阶段，使用电机所能输出的最大牵引力牵引，有利于快速使列车获得较大的动能，同时进行能量回收，缩短运行所需的时间；第二：列车运行时尽量减少惰行的次数，一是不符合实际，多次惰行产生的加速度会使旅客感到不适，二是惰行次数越多证明牵引次数越多，而站间距离是固定的，牵引的时间越长能量的消耗越大；第三：在列车制动停车前的站间运行不采取制动措施，而是采用再生电机的制动；因为不必要的制动会带来巨大的能量损失。若没有突发情况，在一般线路上经过合理的操纵设计完全可以避免站间采取不必要的制动。

对单列车进行受力分析的过程中，由于牵引力、阻力都是与速度有关的函数，坡度梯度随列车所在的距离而变化，所以列车在全过程都在做变加速运动。在计算过程中，将列车的运动逐渐逼近匀变速运动，将规定的时间分为多个小的时间段，则认为列车在

各时间段内做匀变速运动，进而可以求出最节能的速度距离曲线和列车发动机所消耗的能量。

综合以上信息，使用模拟退火求出最优解： 在问题 2 中将时间离散化，则可将问题转化为带有约束条件的非线性整数规划问题，使用模拟退火求解答案。模拟退火是一种试图模拟退火的物理过程的方法。算法先以搜寻空间内一个任意点作起始:每一步先选择一个“邻居”,然后再计算从现有位置到达“邻居”的概率。不同于登山算法，由于只舍弃部分坏点,因此模拟退火得到的是全局最优解。

5.2.2 数据处理

由附件一可知，对 Distance=739.018m 和 2188.63m 两组数据进行处理，这两组数据的 Gradient 值出现异常,应取 0.679012 和 0.555556 的平均值进行计算,作为 739.018m-2188.63m 这段路程里 Gradient 的值。

根据附件二，分析可得到下表

参数	符号	城市轨道交通模型
静态列车质量	M	176.3t
旋转余量	p [%]	8
最大牵引力	F	310KN
最大再生制动力	B	260KN
列车阻力	R	$2.0895+0.0098v+0.006v^2$
牵引/再生制动效率	η	0.9/0.6

表 2 列车参数

5.2.3 模型求解

列车在坡面上行驶时，对列车进行受力分析，受重力、支持力、牵引力、阻力、支持力。

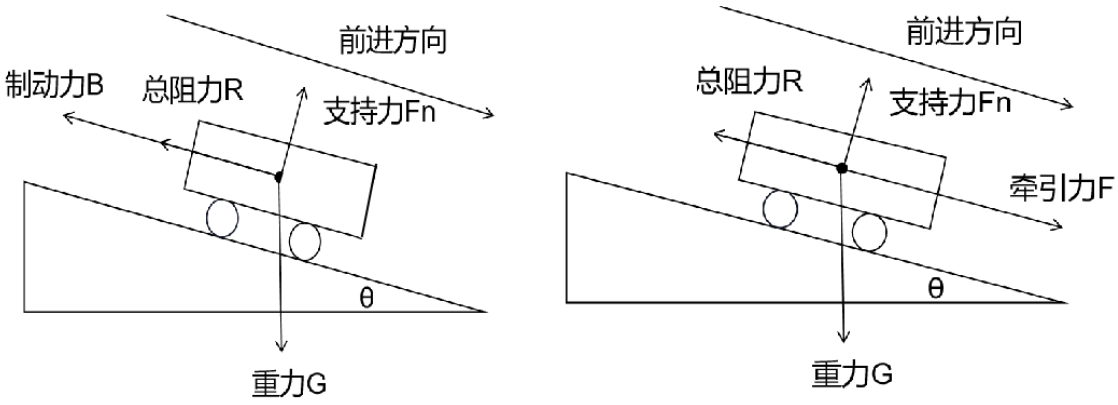

图 11 列车在坡面上受制动力或牵引力时受力图

由分析可知，牵引力为：

$$F=\begin{cases}310 & 0\leqslant v\leqslant 10\\ \dfrac{3100}{v} & v\geqslant 10\end{cases} \tag{8}$$

制动力为：

$$B=\begin{cases}260 & 0\leqslant v\leqslant 17\\ \dfrac{4420}{v} & v\geqslant 17\end{cases} \tag{9}$$

列车 t 时刻的瞬时速度为：

$$v_t=\int_t^{t+\Delta t}\frac{F+G\sin\theta-\rho ma-R}{m}dt \tag{10}$$

t 时刻对应路程为：公式（6）

目标函数为：

$$\min E_F=\int_{t_0}^{t_1}F(t)v(t)dt$$

约束条件为：

$$\begin{cases}0\leq V\leq v_{\max}\\ a\leq a_{\max}\\ 0\leq B\leq B_{\max}\\ 0\leq F\leq F_{\max}\\ \sum\limits_{t=0}^{T}s_t=L\end{cases}$$

由列车站间运行时间 t 和能耗 E 存在反比关系，可以认为当全力牵引段行驶路程的取值可以使列车运行时间取到最大时（$t_{\max}$），即满足耗能模型方程 时，列车耗能最小（$E_{\min}$）。现采用模拟退火算法（simulated annealing，SA）求解全力牵引段位移 s,当运行时间尽量长时，使得列车耗能取最小值。 T 为系统控制参数，t_0 为控制参数初值，控制参数衰减函数$T_{k+1}=\alpha T_k$，取α=0.99；s_0 为列车全力牵引段位移，0s 为位移初值，取极限加速到限速 27.78m/s（100km/h）列车行驶的距离；每一个 t 对应固定的 s 值和 E(t)值，通过上述分析，t 在可行域中的最大取值（s）得到的 E(t)值最小，为模型最优解。

表 3 列车极限加速/减速三段模型工况

列车运行段	工况	控制输入
牵引段（最大加速度）	全力牵引（FP）	$\mu_f=1$，$\mu_b=0$
惰行段	惰行（C）	$\mu_f=0$，$\mu_b=0$
制动段（最大减速度）	全力制动（FB）	$\mu_f=0$，$\mu_b=1$

注：牵引段$\mu_f=1$和制动段$\mu_b=1$为控制输入

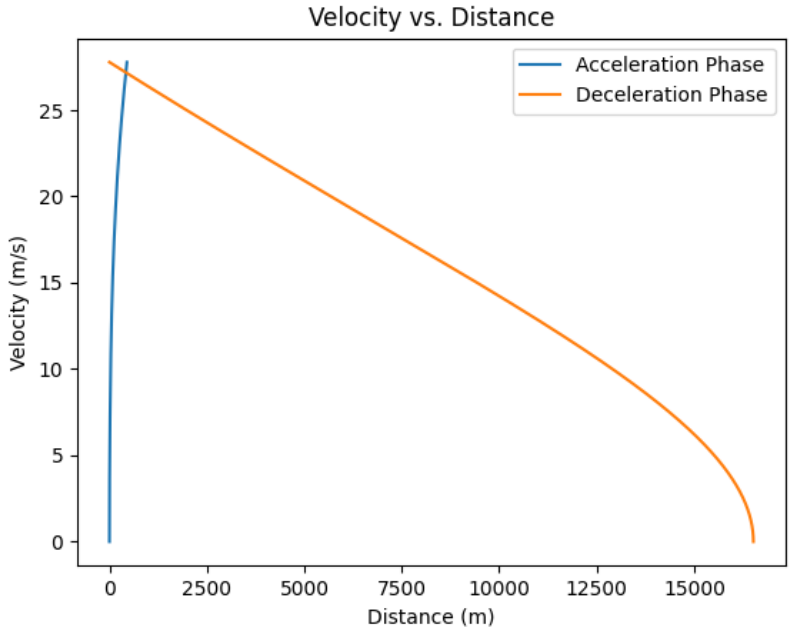

图 12 速度-距离曲线

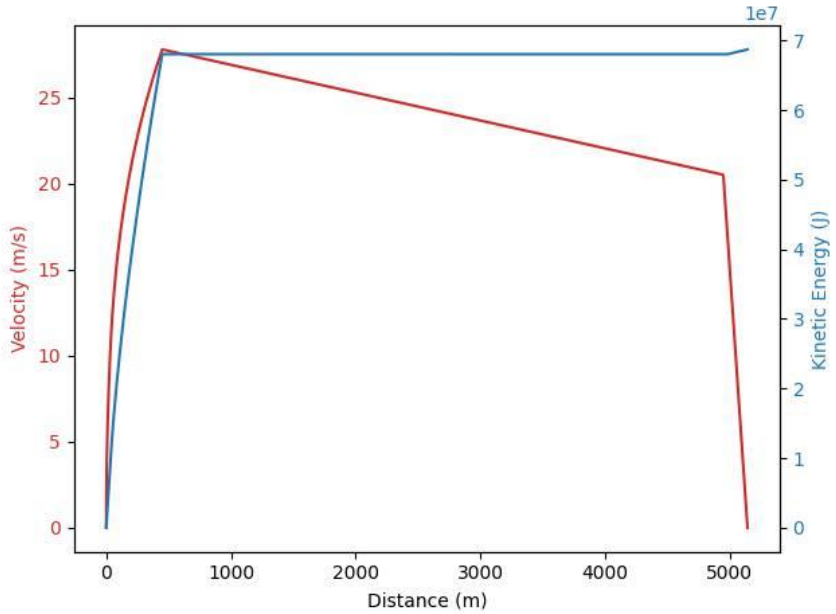

图 13 速度-动能曲线

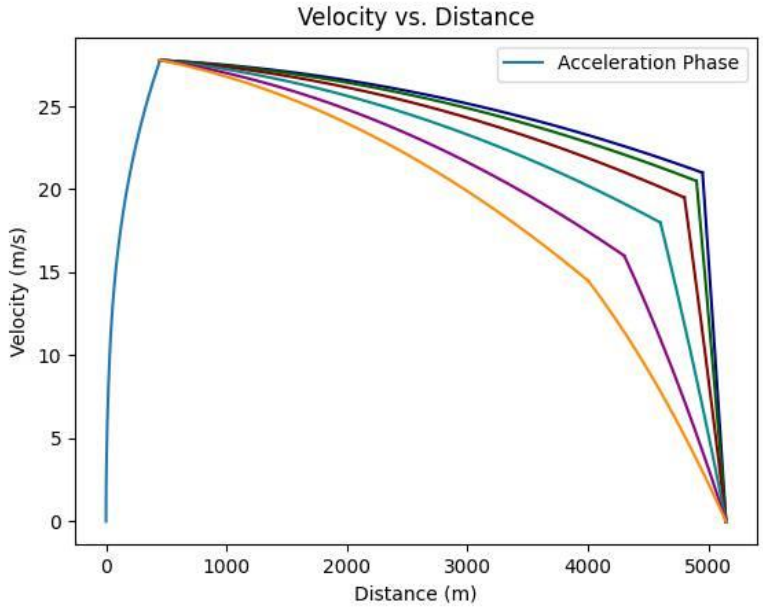

图 14 速度-路程曲线

5.3 问题三的模型建立与求解

5.3.1 模型的建立

为了设计优化方案，需要考虑列车的速度、牵引制动力、时间和能量消耗等因素。

1. 加速阶段：列车从起点开始以最大牵引力加速，直到达到调整后的目标速度。
2. 维持阶段：列车以调整后的目标速度匀速行驶，直到接近事故地点。
3. 制动阶段：在距离事故地点一定距离时，列车开始减速，以适应延迟到达终点的时间。
4. 延迟阶段：列车以较低的速度行驶，以延迟到达终点并保持节能。

以下是具体过程：

1. 加速阶段：
    - 初始速度：0 m/s
    - 加速度：根据列车的性能参数确定
    - 加速时间：根据加速度和目标速度计算得出
    - 距离：根据加速度和加速时间计算得出
2. 维持阶段：
    - 速度：调整后的目标速度
    - 时间：320s - 加速时间
3. 制动阶段：
    - 减速度：根据列车的性能参数确定
    - 减速时间：60s
    - 距离：根据减速度和减速时间计算得出
4. 延迟阶段：
    - 速度：较低的速度，以适应延迟到达终点并保持节能
    - 时间：60s

接着，使用问题二里模拟退火算法优化列车的速度轨迹时，需要明确定义目标函数和状态空间，以及确定温度和退火率等参数。具体过程如下：

1. 定义状态空间：
    - 状态：表示列车在每个时间步的速度值。
    - 邻域操作：根据当前状态生成新的状态，例如微调当前速度值。
2. 定义目标函数：
    - 目标函数：结合列车的速度、时间和能量消耗等指标，计算一个综合评估值。可以考虑最小化总行驶时间和能量消耗的加权和。
3. 初始化和参数设置：
    - 初始状态：可以根据初始的速度-距离曲线估计出一个初始状态。
    - 温度和退火率：设置初始温度和退火率，控制搜索空间的探索和收敛速度。
4. 模拟退火算法优化：

- 初始化当前状态为初始状态。
- 迭代直到满足停止条件（例如达到最大迭代次数或收敛到一个可接受的解）：
- 在当前状态基础上，生成邻域状态。
- 计算目标函数值的变化（例如评估新状态和当前状态的目标函数值差异）。
- 根据一定的概率接受或拒绝邻域状态作为新的当前状态（例如根据 Metropolis 准则）。
- 更新温度和退火率。
- 返回最优的状态作为优化后的速度轨迹。

5.3.2 模型求解

参照问题二列出目标函数与约束条件：

目标函数为：

$$\min E_F = \int_{t_0}^{t_1} F(t)v(t)dt$$

约束条件为：

$$\begin{cases} 0 \le V \le v_{\max} \\ a \le a_{\max} \\ 0 \le B \le B_{\max} \\ 0 \le F \le F_{\max} \\ \sum_{t=0}^{T} s_t = L \end{cases}$$

编写程序，套用问题二使用模拟退火算法的大致框架，所得到的结果图如下。

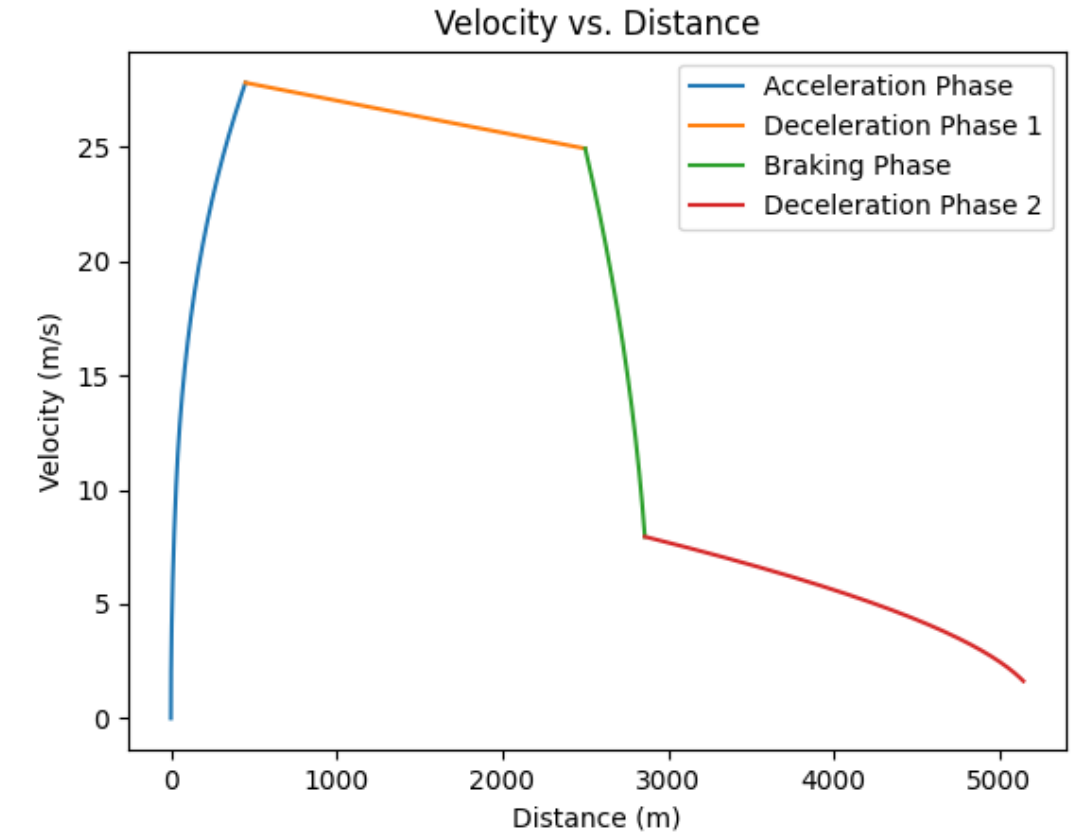
Velocity vs. Distance
Acceleration Phase
Deceleration Phase 1
Braking Phase
Deceleration Phase 2
Velocity (m/s)
Distance (m)
0
5
10
15
20
25
0
1000
2000
3000
4000
5000

图 15 速度-距离曲线

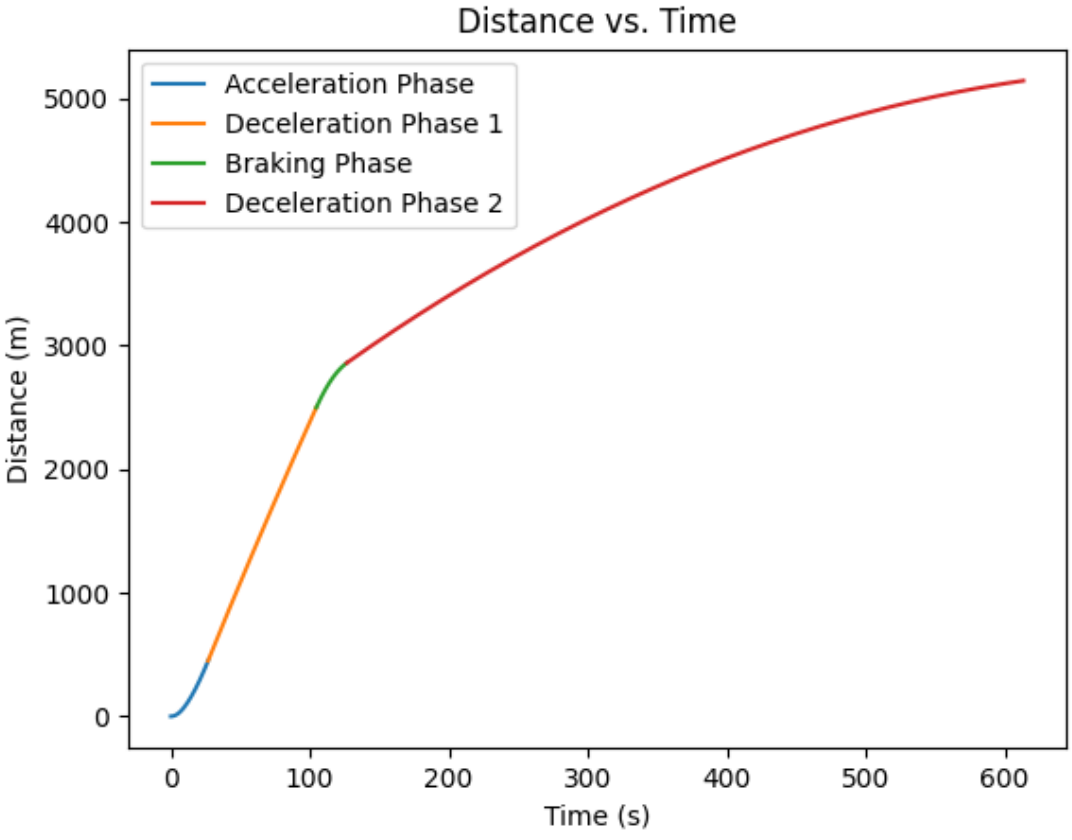
Distance vs. Time
Acceleration Phase
Deceleration Phase 1
Braking Phase
Deceleration Phase 2
Distance (m)
Time (s)
0
1000
2000
3000
4000
5000
0
100
200
300
400
500
600

图 16 距离-时间曲线

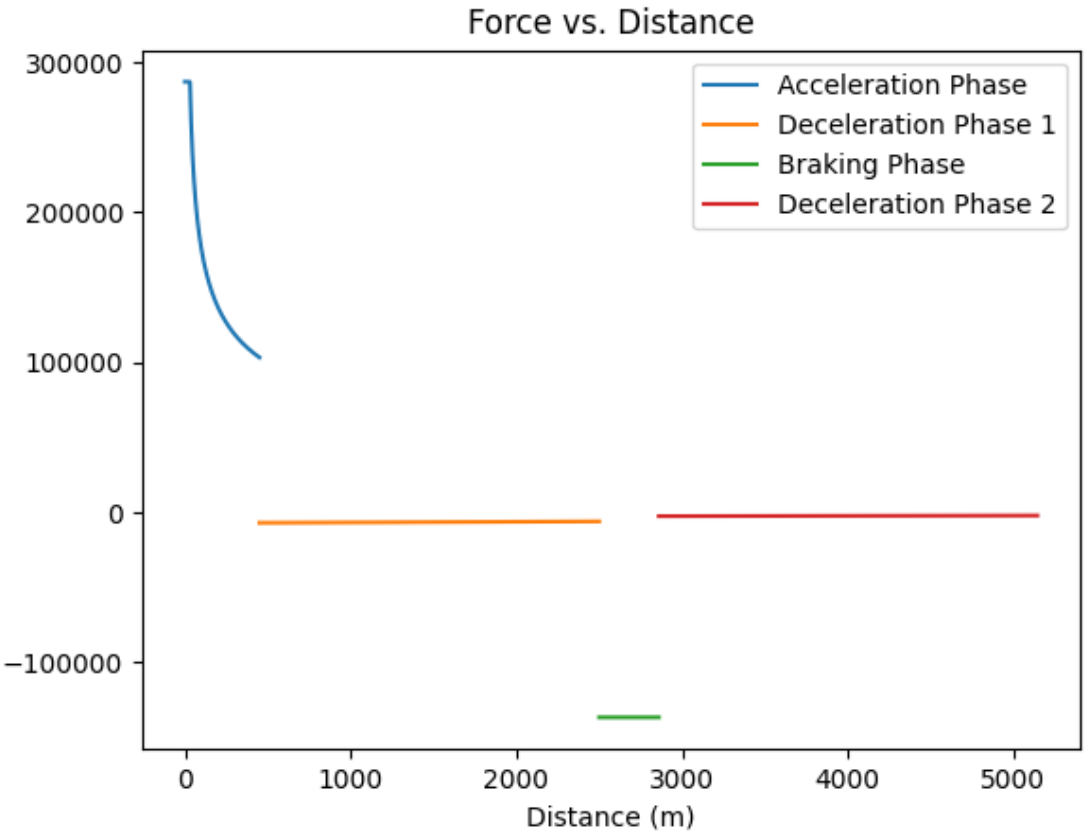

图 17 力-距离曲线

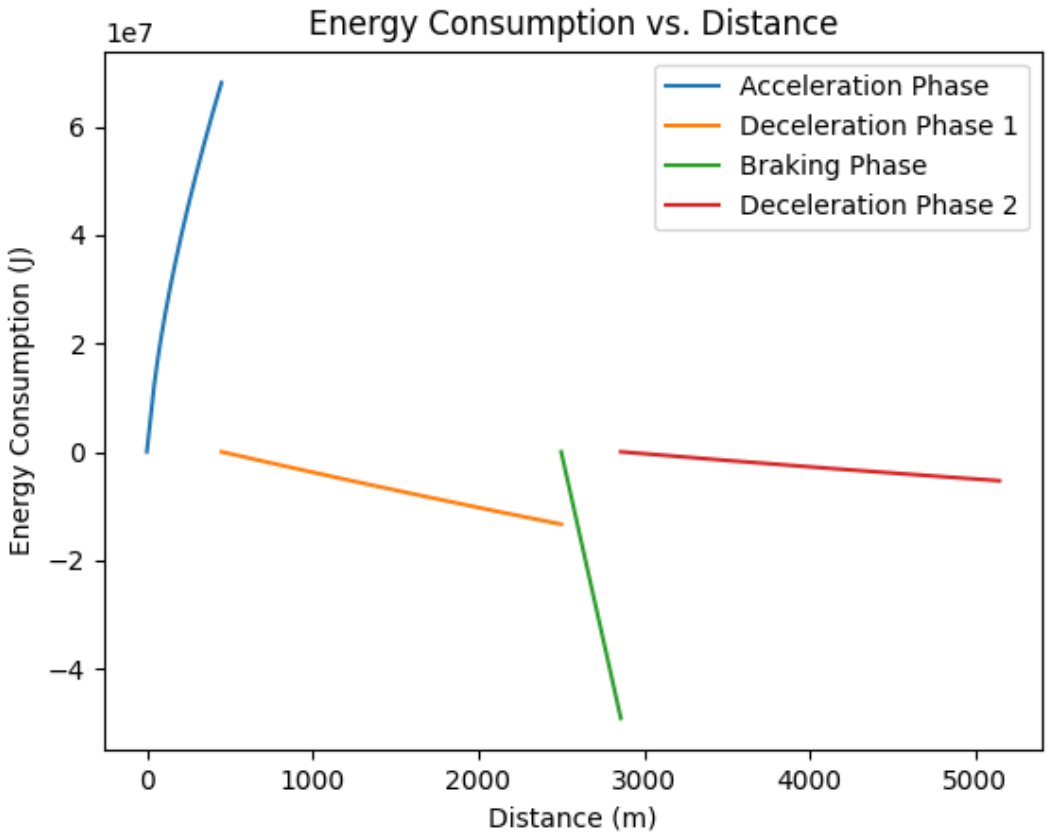

图 18 能耗-距离曲线

六、 模型的评价和优化

6.1.1 模型的优点

一般情况下，当求解问题的约束条件较多，范围规模较大时，直接采用优化算法求解，常常难以找到最优解，甚至出现算法不收敛的情况。因而，结合模拟方法，在优化之前在可行域内先找到合理解，然后在进行优化成为求解复杂有效途径，再运用模拟退火从而达到目的。

6.1.2 模型的缺点

调整方案模型使用的假定前提存在一定的严苛性，使得调整的适用范围被局限。

七、 参考文献

[1] 刘炜,李群湛,郭蕾等.基于多种群遗传算法的城轨列车节能运行优化研究[J].系统仿真学报,2010,22(04):921-925.DOI:10.16182/j.cnki.joss.2010.04.012.

八、 附录

附录 1

介绍：问题一代码

```
# 列车参数
mass = 176.3e3  # kg
p = 1.08
max_force = 310e3  # N
max_brake_force = 760e3  # N
distance = 5144.7  # m
max_speed = 100 / 3.6  # m/s

# Davis 阻力方程参数
a = 2.0895
b = 0.0098
c = 0.006

# 初始化
dt = 0.01  # 时间步长，秒

def compute_distance(speed, force, max_speed, direction='accelerate'):
    distance = 0.0
    while speed < max_speed if direction == 'accelerate' else speed > 0:
        resistance = (a + b*speed + c*speed**2) * 1e3  # N
        acceleration = (force - resistance if direction == 'accelerate' else resistance - force)
/ (mass * p)
        speed += acceleration * dt
        distance += speed * dt
    return distance
```

```
# 估计加速和减速阶段的距离
distance_accelerate = compute_distance(0.0, max_force, max_speed, direction='accelerate')
distance_decelerate = compute_distance(max_speed, max_brake_force, 0.0, direction='decelerate')

# 二分查找法找到最佳的匀速行驶距离
low = 0.0
high = distance - distance_accelerate - distance_decelerate
while high - low > 1e-6:
    mid = (low + high) / 2
    distance_decelerate = compute_distance(max_speed, max_brake_force, 0.0, direction='decelerate')
    if distance_accelerate + mid + distance_decelerate < distance:
        low = mid
    else:
        high = mid

distance_cruise = mid
print(f'加速距离: {distance_accelerate} m, 匀速距离: {distance_cruise} m, 减速距离: {distance_decelerate} m')

def simulate(mass, p, max_force, max_brake_force, distance, max_speed, dt, distance_accelerate, distance_cruise, distance_decelerate):
    position = 0.0
    speed = 0.0
    time = 0.0
    energy = 0.0
    positions = [position]
    speeds = [speed]
    times = [time]
    energies = [energy]
    forces = [max_force]

    # 加速阶段
    while position < distance_accelerate:
        resistance = (a + b*speed + c*speed**2) * 1e3
        acceleration = (max_force - resistance) / (mass * p)
        speed += acceleration * dt
        position += speed * dt
        time += dt
        energy += max_force * speed * dt
        positions.append(position)
```

```
            speeds.append(speed)
            times.append(time)
            energies.append(energy)
            forces.append(max_force)

        # 匀速阶段
        while position < distance_accelerate + distance_cruise:
            resistance = (a + b*speed + c*speed**2) * 1e3
            position += speed * dt
            time += dt
            energy += resistance * speed * dt
            positions.append(position)
            speeds.append(speed)
            times.append(time)
            energies.append(energy)
            forces.append(resistance)

        # 减速阶段
        while position < distance:
            resistance = (a + b*speed + c*speed**2) * 1e3
            acceleration = (resistance - max_brake_force) / (mass * p)
            speed += acceleration * dt
            position += speed * dt
            time += dt
            energy += max_brake_force * speed * dt
            positions.append(position)
            speeds.append(speed)
            times.append(time)
            energies.append(energy)
            forces.append(max_brake_force)

        return positions, speeds, times, energies, forces

# 新的加速时间，秒
extra_times = [10, 20, 50, 150, 300]

plt.figure(figsize=(15,10))

for extra_time in extra_times:
        # 计算新的力
        new_force = mass * max_speed / (distance_accelerate / max_speed + extra_time)
        # 计算新的距离
```

```
    new_distance_accelerate = compute_distance(0.0, new_force, max_speed, direction='accelerate')
    new_distance_cruise = distance - new_distance_accelerate - distance_decelerate
    # 进行模拟
    positions, speeds, times, energies, forces = simulate(mass, p, new_force, max_brake_force, distance, max_speed, dt, new_distance_accelerate, new_distance_cruise, distance_decelerate)

    # 画出曲线
    plt.subplot(221)
    plt.plot(positions, speeds, label=f'Extra Time {extra_time}s')
    plt.xlabel('Distance (m)')
    plt.ylabel('Speed (m/s)')
    plt.title('Speed-Distance')

    plt.subplot(222)
    plt.plot(positions, forces, label=f'Extra Time {extra_time}s')
    plt.xlabel('Distance (m)')
    plt.ylabel('Force (N)')
    plt.title('Force-Distance')

    plt.subplot(223)
    plt.plot(positions, times, label=f'Extra Time {extra_time}s')
    plt.xlabel('Distance (m)')
    plt.ylabel('Time (s)')
    plt.title('Time-Distance')

    plt.subplot(224)
    plt.plot(positions, energies, label=f'Extra Time {extra_time}s')
    plt.xlabel('Distance (m)')
    plt.ylabel('Energy (J)')
    plt.title('Energy-Distance')

# 添加图例
plt.subplot(221)
plt.legend()

plt.subplot(222)
plt.legend()

plt.subplot(223)
plt.legend()
```

```
plt.subplot(224)
plt.legend()

plt.tight_layout()
plt.show()



import numpy as np
import matplotlib.pyplot as plt

# Parameters
dt = 0.1
mass = 176.3e3    # kg
p = 1.08
max_force = 0.9 * 310e3    # N
max_brake_force = 0.6 * 760e3    # N
distance = 5144.7    # m

#  额外力与距离范围
extra_forces = [
        (0, 198.966, 106.6506256),
        (198.967, 739.018, 28045.47218),
        (739.018, 2188.63, 1173.155922),
        (2188.63, 2870.8, -16956.62313),
        (2870.8, 4178.29, -5225.853996),
        (4178.29, 4259, -34867.67489),
        (4259, 4604.65, 5225.853996),
        (4604.65, 4803.62, 35080.74697),
        (4803.62, 4960, 35080.74697),
        (4960, 5144.7, 35080.74697),
]

# Function to calculate energy and time
def calculate_energy_time(x1, x2, x3):
        x4 = 5144.7
        position = 0
        speed = 1
        time = 0
        energy = 0
        while position < distance:
```

```
            fr = 0
            f2 = 0
            f1 = 0
            if 0 <= position <= 4529 or 4960 <= position <= distance:
                max_speed = 100 / 3.6
            else:
                max_speed = 86 / 3.6    # m/s

            for lower, upper, extra_force in extra_forces:
                if lower <= position < upper:
                    f1 = extra_force
                    break

            acceleration = (f1 + fr + f2) / (p * mass)
            speed += acceleration * dt
            speed = min(speed, max_speed)    # Ensure speed is not more than max_speed
            if position >= x2:
                speed = min(speed, 86 / 3.6)
            position += speed * dt
            time += dt
            if 0 <= position <= x2:
                energy += f1 * speed * dt
        return energy, time

# Simulated Annealing
def simulated_annealing():
        x = np.sort(np.random.rand(3) * distance)    # x1, x2, x3
        T = 10.0    # initial temperature
        T_min = 0.1    # minimum temperature
        cooling_rate = 0.99    # cooling rate
        while T > T_min:
            new_x = x + (np.random.rand(3) - 0.5) * T
            new_x.sort()
            new_x = np.clip(new_x, 0, distance)
            old_energy, old_time = calculate_energy_time(*x)
            new_energy, new_time = calculate_energy_time(*new_x)
            if (new_energy < old_energy) or (np.exp((old_energy - new_energy) / T) >
np.random.rand()):
                x = new_x
            T = T * cooling_rate
        return x
```

```
# Run the simulated annealing
optimized_points = simulated_annealing()
print("Optimized points:", optimized_points)

energy, time = calculate_energy_time(*optimized_points)
print("Energy:", energy)
print("Time:", time)









import numpy as np
import matplotlib.pyplot as plt

# Parameters
dt = 0.1
mass = 176.3e3    # kg
p = 1.08
max_force = 0.9 * 310e3    # N
max_brake_force = 0.6 * 760e3    # N
distance = 5144.7    # m

# Function to calculate energy and time
def calculate_energy_time(x1, x2, x3, return_data=False):
        x4 = 5144.7
        position = 0
        speed = 1
        time = 0
        energy = 0

        # Data for plotting
        positions = []
        speeds = []
        forces = []
        times = []
        energies = []

        while position < distance:
                fr = 0
```

```
            f2 = 0
            if 0 <= position <= 4529 or 4960 <= position <= distance:
                max_speed = 100 / 3.6
            else:
                max_speed = 86 / 3.6    # m/s
            if position <= x1:
                if 0 <= speed <= 10:
                    f1 = 0.9 * 310e3
                else:
                    f1 = 0.9 * 310e3 / speed
            elif x1 <= position < x2:
                fr = -2.0895 * 1e3 - 0.0098 * speed * 1e3 - 0.006 * 1e3 * speed ** 2
                f2 = -200e3
                f1 = -fr - f2
            elif x2 <= position < x3:
                f1 = 0
            elif x3 <= position < x4:
                f1 = -0.6 * 260e3
            acceleration = (f1 + fr + f2) / (p * mass)
            speed += acceleration * dt
            speed = min(speed, max_speed)    # Ensure speed is not more than max_speed
            if position >= x2:
                speed = min(speed, 86 / 3.6)
            position += speed * dt
            time += dt
            if 0 <= position <= x2:
                energy += f1 * speed * dt

            # Save data for plotting
            if return_data:
                positions.append(position)
                speeds.append(speed)
                forces.append(f1)
                times.append(time)
                energies.append(energy)

    if return_data:
        return positions, speeds, forces, times, energies
    else:
        return energy, time

def plot_curves(x1, x2, x3):
```

```
    positions, speeds, forces, times, energies = calculate_energy_time(x1, x2, x3, return_data=True)

    # Speed vs. distance
    plt.plot(positions, speeds)
    plt.xlabel('Position (m)')
    plt.ylabel('Speed (m/s)')
    plt.title('Speed vs. Distance')
    plt.show()

    # Force vs. distance
    plt.plot(positions, forces)
    plt.xlabel('Position (m)')
    plt.ylabel('Force (N)')
    plt.title('Force vs. Distance')
    plt.show()

    # Time vs. distance
    plt.plot(positions, times)
    plt.xlabel('Position (m)')
    plt.ylabel('Time (s)')
    plt.title('Time vs. Distance')
    plt.show()

    # Energy vs. distance
    plt.plot(positions, energies)
    plt.xlabel('Position (m)')
    plt.ylabel('Energy (J)')
    plt.title('Energy vs. Distance')
    plt.show()
```

附录 2

介绍：问题二模拟退火代码

```
import numpy as np
import matplotlib.pyplot as plt

# Parameters
dt = 0.1
mass = 176.3e3    # kg
p = 1.08
max_force = 0.9 * 310e3    # N
```

```
max_brake_force = 0.6 * 760e3    # N
distance = 5144.7    # m

#  额外力与距离范围
extra_forces = [
        (0, 198.966, 106.6506256),
        (198.967, 739.018, 28045.47218),
        (739.018, 2188.63, 1173.155922),
        (2188.63, 2870.8, -16956.62313),
        (2870.8, 4178.29, -5225.853996),
        (4178.29, 4259, -34867.67489),
        (4259, 4604.65, 5225.853996),
        (4604.65, 4803.62, 35080.74697),
        (4803.62, 4960, 35080.74697),
        (4960, 5144.7, 35080.74697),
]

# Function to calculate energy and time
def calculate_energy_time(x1, x2, x3):
        x4 = 5144.7
        position = 0
        speed = 1
        time = 0
        energy = 0
        while position < distance:
                fr = 0
                f2 = 0
                f1 = 0
                if 0 <= position <= 4529 or 4960 <= position <= distance:
                        max_speed = 100 / 3.6
                else:
                        max_speed = 86 / 3.6    # m/s

                for lower, upper, extra_force in extra_forces:
                        if lower <= position < upper:
                                f1 = extra_force
                                break

                acceleration = (f1 + fr + f2) / (p * mass)
                speed += acceleration * dt
                speed = min(speed, max_speed)    # Ensure speed is not more than max_speed
                if position >= x2:
```

```
                speed = min(speed, 86 / 3.6)
            position += speed * dt
            time += dt
            if 0 <= position <= x2:
                energy += f1 * speed * dt
    return energy, time

# Simulated Annealing
def simulated_annealing():
    x = np.sort(np.random.rand(3) * distance)    # x1, x2, x3
    T = 10.0    # initial temperature
    T_min = 0.1    # minimum temperature
    cooling_rate = 0.99    # cooling rate
    while T > T_min:
        new_x = x + (np.random.rand(3) - 0.5) * T
        new_x.sort()
        new_x = np.clip(new_x, 0, distance)
        old_energy, old_time = calculate_energy_time(*x)
        new_energy, new_time = calculate_energy_time(*new_x)
        if (new_energy < old_energy) or (np.exp((old_energy - new_energy) / T) >
np.random.rand()):
            x = new_x
        T = T * cooling_rate
    return x

# Run the simulated annealing
optimized_points = simulated_annealing()
print("Optimized points:", optimized_points)

energy, time = calculate_energy_time(*optimized_points)
print("Energy:", energy)
print("Time:", time)
```

附录 3

介绍：问题三代码

```
  import numpy as np
import matplotlib.pyplot as plt

# Parameters
dt = 0.1
mass = 176.3e3  # kg
```

```
p = 1.08
max_force = 0.9 * 310e3  # N
max_brake_force = 0.6 * 760e3  # N
distance = 5144.7  # m

# Function to calculate energy and time
def calculate_energy_time(x1, x2, x3, return_data=False):
    x4 = 5144.7
    position = 0
    speed = 1
    time = 0
    energy = 0

    # Data for plotting
    positions = []
    speeds = []
    forces = []
    times = []
    energies = []

    while position < distance:
        fr = 0
      f2 = 0
        if 0 <= position <= 4529 or 4960 <= position <= distance:
            max_speed = 100 / 3.6
        else:
            max_speed = 86 / 3.6  # m/s
        if position <= x1:
            if 0 <= speed <= 10:
                f1 = 0.9 * 310e3
            else:
                f1 = 0.9 * 310e3 / speed
        elif x1 <= position < x2:
            fr = -2.0895 * 1e3 - 0.0098 * speed * 1e3 - 0.006 * 1e3 * speed ** 2
            f2 = -200e3
            f1 = -fr - f2
        elif x2 <= position < x3:
            f1 = 0
        elif x3 <= position < x4:
            f1 = -0.6 * 260e3
        acceleration = (f1 + fr + f2) / (p * mass)
        speed += acceleration * dt
```

```
        speed = min(speed, max_speed)  # Ensure speed is not more than max_speed
        if position >= x2:
                speed = min(speed, 86 / 3.6)
            position += speed * dt
            time += dt
            if 0 <= position <= x2:
                energy += f1 * speed * dt

        # Save data for plotting
            if return_data:
                positions.append(position)
                speeds.append(speed)
                forces.append(f1)
                times.append(time)
                energies.append(energy)

    if return_data:
            return positions, speeds, forces, times, energies
        else:
            return energy, time

def plot_curves(x1, x2, x3):
        positions, speeds, forces, times, energies = calculate_energy_time(x1, x2, x3, return_d
ata=True)

    # Speed vs. distance
        plt.plot(positions, speeds)
        plt.xlabel(&apos;Position (m)&apos;)
        plt.ylabel(&apos;Speed (m/s)&apos;)
        plt.title(&apos;Speed vs. Distance&apos;)
        plt.show()

    # Force vs. distance
        plt.plot(positions, forces)
        plt.xlabel(&apos;Position (m)&apos;)
        plt.ylabel(&apos;Force (N)&apos;)
        plt.title(&apos;Force vs. Distance&apos;)
        plt.show()

    # Time vs. distance
        plt.plot(positions, times)
        plt.xlabel(&apos;Position (m)&apos;)
```

```
    plt.ylabel(&apos;Time (s)&apos;)
        plt.title(&apos;Time vs. Distance&apos;)
        plt.show()

    # Energy vs. distance
        plt.plot(positions, energies)
        plt.xlabel(&apos;Position (m)&apos;)
        plt.ylabel(&apos;Energy (J)&apos;)
        plt.title(&apos;Energy vs. Distance&apos;)
        plt.show()


# Plot the curves
plot_curves(x1, x2, x3)
```