武漢理工大學

数学建模暑期培训论文

第 1 题

基于 xxxxxxxxx 模型

第 A010 组

姓名	方向
刘子川（组长）	编程
程宇	建模
祁成	写作

2020 年 7 月 21 日

摘要

控制高压油管的压力变化对减小燃油量偏差，提高发动机工作效率具有重要意义。本文建立了基于质量守恒定理的微分方程稳压模型，采用二分法、试探法以及自适应权重的蝙蝠算法对模型进行求解。//

针对问题一，建立基于质量守恒定律的燃油流动模型，考察单向阀开启时间对压力稳定性的影响。综合考虑压力与弹性模量、密度之间的关系，提出燃油压力-密度微分方程模型和燃油流动方程。本文采用改进的欧拉方法对燃油压力-密度微分方程求得数值解；利用二分法求解压力分布。综合考虑平均绝对偏差等反映压力稳定程度的统计量，求得直接稳定于 100MPa 的开启时长为 **0.2955ms**，在 2s、5s 内到达并稳定于 150MPa 时开启时长为 **0.7795ms**、**0.6734ms**，10s 到达并稳定于 150MPa 的开启时长存在多解。最后对求解结果进行灵敏度分析、误差分析。//

针对问题二，建立基于质量守恒定律的泵-管-嘴系统动态稳压模型，将燃油进入和喷出的过程动态化处理。考虑柱塞和针阀升程的动态变动，建立喷油嘴流量方程和质量守恒方程。为提高角速度求解精度，以凸轮转动角度为固定步长，转动时间变动步长，采用试探法粗略搜索与二分法精细搜索的方法求解，求得凸轮最优转动角速度 **0.0283rad/ms**（**转速 270.382 转/分钟**），并得到该角速度下高压油管的密度、压力周期性变化图。对求解结果进行误差分析与灵敏度分析，考察柱塞腔残余容积变动对高压油管压力稳态的影响。//

针对问题三，对于增加一个喷油嘴的情况，改变质量守恒方程并沿用问题二的模型调整供、喷油策略，得到最优凸轮转动角速度为 **0.0522rad/ms**（**498.726 转/分钟**）；对于既增加喷油嘴又增加减压阀的情况，建立基于自适应权重的蝙蝠算法的多变量优化模型，以凸轮转动角速度、减压阀开启时长和关闭时长为参数，平均绝对偏差 MAD 为目标，在泵-管-嘴系统动态稳压模型的基础上进行求解，得到最优参数：**角速度 0.0648 rad/ms**（**619.109 转/分钟**）、减压阀的开启时长 **2.4ms** 和减压阀的关闭时长 **97.6ms**。//

本文的优点为：1. 采用试探法粗略搜索与二分法精细搜索结合的方法，降低了问题的求解难度。2. 以凸轮转动角度为固定步长，对不同角速度按照不同精度的时间步长求解，大大提高了求解的精确度。3. 针对智能算法求解精度方面，采用改进的蝙蝠算法，使速度权重系数自适应调整，兼顾局部搜索与全局搜索能力。

关键词： 微分方程 微分方程 微分方程 微分方程

目录

一、 问题重述

1.1 问题背景

分析研究[1]。xxxxxxxxxxx[1]. 村通自来水工程是指在现有农村居民饮水安全工程的基础上，通过扩网、改造、联通、整合和新建等措施，把符合国家水质标准的自来水引接到行政村和有条件的自然村，形成具有高保证率和统一供水标准的农村供水网络，基本形成覆盖全县农村的供水安全保障体系，实现农村供水由点到面、由小型分散供水到适度集中供水、由解决水量及常规水质到水量、水质、水压达标等方面的提升，使广大农村居民长期受益，实现我县农村饮水“提质增效升级”的目的。

自来水管道铺设是搭建自来水系统的重要环节，合理的管道铺设方案可以大幅度节约成本。本问题要求在充分考虑市场因素后，研究用两种不同型号的管道铺设该村的自来水管道的方案，使得建设成本降低。由于不同类形的管道的成本不同，且在实际应用中自来水厂有功率限制，研究自来水管的铺设对于村通自来水工程有着重要意义。

1.2 问题概述

围绕相关附件和条件要求，研究两种型号的管道在各自来水厂间的铺设方案，依次提出以下问题:

问题一：设计从中心供水站 A 出发使得自来水管道的总里程最少的铺设方案，并求出该方案下 I 型管道和 II 型管道总里程数。

问题二：由于二型管道数量不足，设计自来水厂升级方案使得两个二级自来水厂升级为一级自来水厂，使得二级管道的使用量尽可能减小。

问题三：考虑自来水厂的功率限制，设计升级方案使得若干的二级自来水厂升级为一级，并求解该情况下的最小铺设总长度。

二、 模型假设

(1)

(2)

(3)

(4)

[1] xxxxxxxxxx.

三、 符号说明

符号	说明
P_n	20 个站点
P_n	20 个站点
P_n	20 个站点

注：表中未说明的符号以首次出现处为准

四、 问题一模型的建立与求解

4.1 问题描述与分析

问题一要求给出总里程最少的管道铺设方案。在村村通自来水工程的连通图 $G = (V(G), E(G))$ 中，每个供水站可以视作一个节点 $v \in V$，节点间的供水管道看作边 $e \in E$，那么由中心供水站到 I 级供水站、由 I 级供水站到 II 级供水站分别构成了生成树 $T_i = (V_i(T_i), E_i(T_i))$，其中 $V_i(G) = V_i(T_i)$，$E_i(T_i) \subset E_i(G), (i = 1, 2)$，且根据定理（这里引用一下!!!!!），$|E_i| = |V_i| - 1$。由于 I 级和 II 级供水站的本质区别是 II 级供水站不能与中心供水站直接相连，故本问题实质上是一个二阶最小生成树问题。我们以两节点间的欧式距离作为边的代价，应用改进的 Prim 算法（二阶的，想个名字!!!!!!）求解模型。其思维流程图如图 1 所示：

图 **1**　问题一思维流程图

4.2 模型的建立

同一般的最小生成树问题不同，问题一中的 II 级供水站生成树优化需待 I 级供水站的最小连通树生成后才能开始进行。因此，问题一的模型（取个屌点的名字?????）分为两部分，I 级供水站和 II 级供水站先后进行目标优化。本问题中，边 e 的代价是两节点 $v_i(x_i, y_i), v_j(x_j, y_j)$ 间的欧式距离, 可表示为：

$$cost(v_i, v_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}, \tag{1}$$

在此二阶最小生成树问题中，已知节点和边的关系为：$|E_i| = |V_i| - 1, (i = 1, 2)$，其中 $|E_i|$ 是边的数目，$|V_i|$ 是节点个数。把生成树节点对应的序列作为决策变量，以序列矩阵的形式表示，每一层的最优序列矩阵可以表示为

$$A_k = \begin{bmatrix} v_{11} & v_{12} \\ v_{21} & v_{22} \\ \dots & \dots \\ v_{|E_k|,1} & v_{|E_k|,2} \end{bmatrix}, k = 1, 2$$

其中，$v_{i1}, v_{i2} (1 \le i \le |E_k|)$ 分别代表边 e_i 的起始节点和终止节点。

分别将 I、II 管道总里程作为优化目标，可知总里程是所有边的代价之和，而代价可以表示成节点间的距离，最佳节点序列已经存放在最优序列矩阵中，这些节点距离可以通过节点序列取出。同时，总里程是关于 E_i 的函数。因此，在第一层和第二层分别对距离求和，可得目标函数最短路径为：

$$F(E) = F_1(E) + F_2(E), \tag{2}$$

$$F_k(E) = \sum_{i=1}^{|E_k|} cost(v_{i1}, v_{i2}), k = 1, 2 \tag{3}$$

此问题的约束条件为问题一中的 II 级供水站生成树优化不能先于 I 级供水站的最小连通树生成，对应的数学描述为

以从中心出发铺设的自来水管道总里程最少，结合约束条件，得到自来水管道最短路径：

$$minF(E) \tag{4}$$

$$s.t. \begin{cases} F(E) = F_1(E) + F_2(E) \\ F_k(E) = \sum_{i=1}^{|E_k|} cost(v_{i1}, v_{i2}), k = 1, 2 \\ cost(v_i, v_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \\ ????? \end{cases}$$

4.3 模型的求解

为求取管道最小里程和最优路径，我们需要求解每一层的最优序列矩阵。针对问题一，我们采用 Prim 算法，分别实现由中心供水站到 I 级供水站、由 I 级供水站到 II 级供水站的最小生成树，可以求得每一层的最优序列矩阵 A_k。

Prim 算法的伪代码如下。

Algorithm 1: Procedure of Apriori

Input: item data base: D
minimum Support threshold: Sup_{min}
minimum Confidence threshold: $Conf_{min}$
Output: frequent item sets F
Initialize
iteration $t \leftarrow 1$
The candidate FIS:$C_t = \varnothing$
The length of FIS:$length = 1$
for *i=1 to sizeof(D)* **do**
  I_i=D(i)
  n=sizeof(I_i)
  for *j=1 to n* **do**
    if $I_i(j) \notin C_t$ **then**
      $C_t = C_t \cup I_i(j)$
    end
  end
end
$F_t = \{f|f \in C_t, Sup(f) > Sup_{min}\}$
while $F \neq \varnothing$ **do**
  t=t+1
  length=length+1
  $C_t \leftarrow$ all candidate of FIS in F_{t-1}
  $F_t = \{f|f \in C_t, (Sup(f) > Sup_{min}) \bigcap (Comf(f) > Conf_{min})\}$
end
return F_{t-1}

结果在这里放一些。。。。。。

4.4 实验结果及分析

1. 灵敏度分析；（2. 对比分析）；3. 算法收敛性分析；4. 算法时间复杂度分析。。。

五、 问题二模型的建立与求解

5.1 问题描述与分析

问题二要求升级两个 II 级供水站为 I 级供水站，使得 II 级管道里程数最少。

其中，第一层更新后的决策变量仍是最优节点序列，用最优序列矩阵表示为：

$$A_1' = \begin{bmatrix} v_{11} & v_{12} \\ v_{21} & v_{22} \\ \ldots & \ldots \\ v_{|E_1|,1} & v_{|E_1|,2} \\ v_{|E_1'|,1} & v_{|E_1'|,2} \end{bmatrix}, |E_1'| = |E_1| + 1 \tag{5}$$

其中，A' 是更新后的最优序列矩阵。$|E_1'|$,$|E_1|$ 分别是更新后和更新前的边集合的模，即边数。升级两个 II 级供水站为 I 级后，增加一条边，故满足 $|E_1'| = |E_1| + 1$。$v_{i1}, v_{i2}(1 \leq i \leq |E_k|)$ 分别代表边 e_i 的起始节点和终止节点。

目标函数为 II 级管道的总里程

$$F_2(E) = \sum_{i=1}^{|E_2|'} cost(v_{i1}, v_{i2}), \tag{6}$$

其思维流程图如图 2 所示：

图 **2**　问题二思维流程图

5.2 模型的建立

5.3 模型的求解

5.4 实验结果及分析

结果如下表**??**所示：

表 **1**　**xxxxxxxxxxxxxxxxxxxxxx**

xxxxxxx	xxxxxxx
xxxxxxx	909.80
xxxxxxx	852.60

由表1可知

其各个小车的运输细节图下图所示：

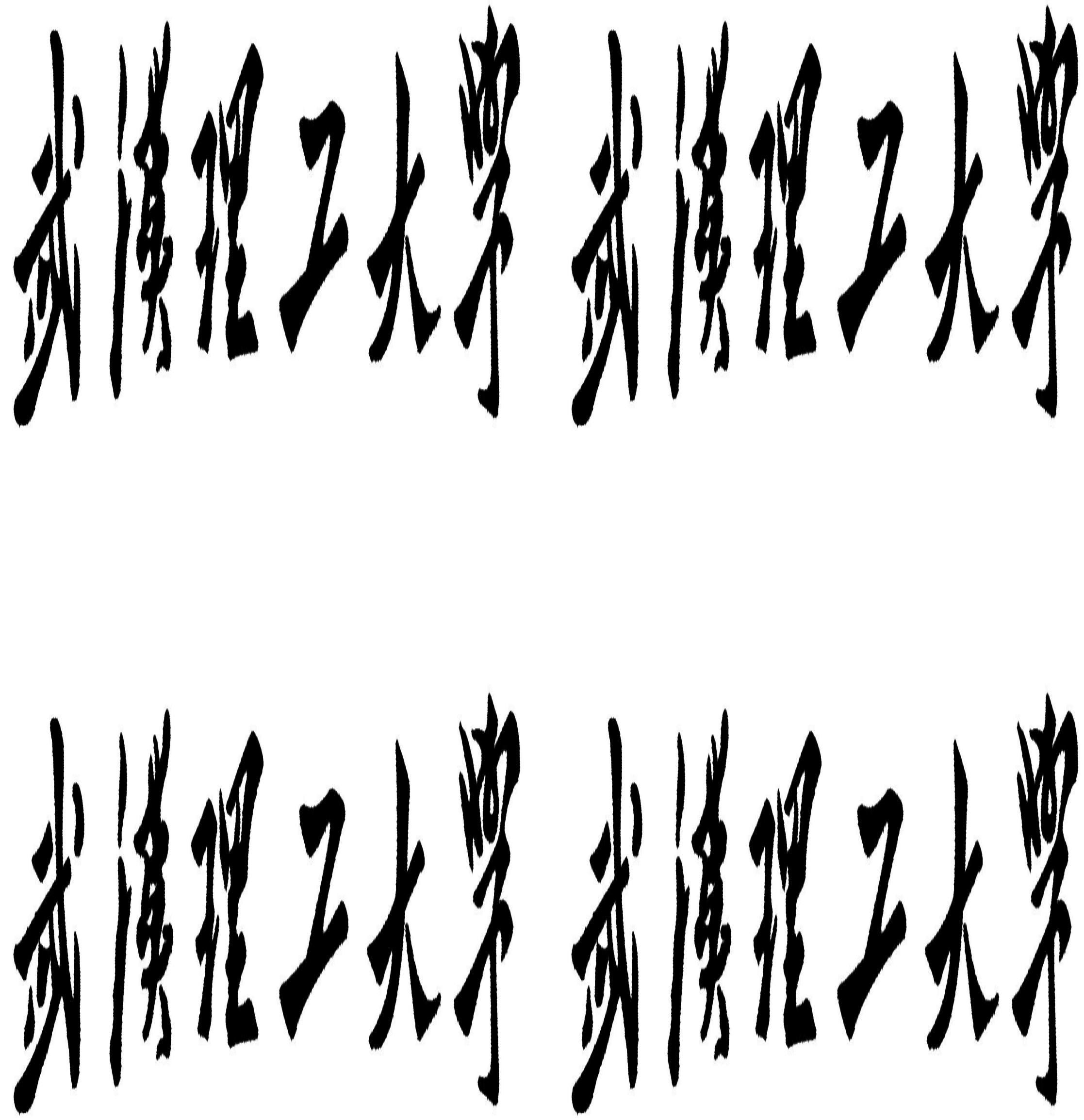

图 3　xxxxxxxxxxxxxxxxxxxxxxxxxxx

六、 问题三模型的建立与求解

6.1 结果分析

七、 灵敏度分析

八、 模型的评价

8.1 模型的优点

(1)

(2)

8.2 模型的缺点

8.3 模型改进

参考文献

[1] 张斯嘉, 郭建胜, 钟夫, 等. 基于蝙蝠算法的多目标战备物资调运决策优化 [J]. 火力与指挥控制, 2016, 41(1): 58-61.

附录 A 问题一、二代码及其可视化

Graph 类实现最小生成树算法

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import copy
import networkx as nx
from tqdm.notebook import tqdm

class Graph(object):
def __init__(self, Matrix, add_edge=None):
self.Matrix = Matrix
self.nodenum = len(self.Matrix)
self.edgenum = self.get_edgenum()
self._weight_ = np.zeros((self.nodenum, self.nodenum))
self.add_edge = add_edge

def get_edgenum(self):
count = 0
for i in range(self.nodenum):
for j in range(i):
if self.Matrix[i][j] > 0 and self.Matrix[i][j] < 9999:
count += 1
return count

def plot_matrix(self, pos=None, figsize=(15,15), title="Pipeline ONE"):
plt.figure(figsize=(12,9))
self._get_edge()
G_nx = nx.Graph()
G_nx2 = nx.Graph()
if self.add_edge!=None:
for i in range(self.nodenum):
for j in range(self.nodenum):
if self._weight_[i, j]!=0 and i<13 and j<13:
G_nx.add_edge(i, j)
if self._weight_[i, j]!=0 and i>0 and j>0:
G_nx2.add_edge(i, j)
else:
for i in range(self.nodenum):
for j in range(self.nodenum):
if self._weight_[i, j]!= 0:
G_nx.add_edge(i, j)
if self.add_edge!=None:
nx.draw_networkx(G_nx, pos[:len(self.add_edge)+1], alpha=0.85)
```

```
nx.draw_networkx(G_nx2,pos,alpha=0.6,with_labels=False,node_color='slateblue',
node_shape=".", node_size=100, style='dashed')
else:
nx.draw_networkx(G_nx, pos, alpha=0.85)
GG = nx.Graph()
GG.add_node(0)
nx.draw_networkx(GG, {0:pos[0]}, node_color='r',node_shape='*', node_size=1200)
plt.title(title)
plt.show() # display

def _get_edge(self):
edge = self.prim()
for k in edge:
self._weight_[k[0],k[1]] = self.Matrix[k[0],k[1]]
return self._weight_

def prim(self, first_node = 0):
# 存储已选顶点，初始化时可随机选择一个起点
select = [first_node]
# 存储未选顶点
candidate = list(range(0, self.nodenum))
candidate.remove(first_node)
if self.add_edge!=None:
node = []
for i in self.add_edge:
if i[0] not in node:
node.append(i[0])
if i[1] not in node:
node.append(i[1])
for i in node:
select.append(i)
if i in candidate:
candidate.remove(i)
# 存储每次搜索到的最小生成树的边
edge = []+self.add_edge if self.add_edge!=None else []

def min_edge(select, candidate, graph):
min_weight = np.inf
v, u = 0, 0
for i in select:
for j in candidate:
if min_weight > graph[i][j]:
min_weight = graph[i][j]
v, u = i, j
return v, u

num = len(self.add_edge)+1 if self.add_edge!=None else 1
```

```
for i in range(num, self.nodenum):
v, u = min_edge(select, candidate, self.Matrix)
edge.append([v, u])
select.append(u)
candidate.remove(u)
return edge
```

问题一代码实现及可视化

```
def distance(x1,y1,x2,y2):
return np.sqrt((x1-x2)**2+(y1-y2)**2)

def fix(x):
if x.startswith('A'):
return 0
return 1 if x.startswith('V') else 2

def get_xy(i,j=0):
pos = []  # 元组中的两个数字是第i（从0开始计数）个点的坐标
for k in range(j, i):
pos.append((data['X坐标'].loc[k], data['Y坐标'].loc[k]))
return pos

weight_array = np.zeros((181,181))
data = pd.read_excel('/content/drive/My Drive/competitions/CMCM/demo1/data.xlsx')
data['类型'] = data['类型'].apply(lambda x:fix(x))

# 初始化权重矩阵
for i in tqdm(range(181)):
for j in range(181):
point_i = data[data['序号']==i]
point_j = data[data['序号']==j]
weight_array[i][j] = distance(point_i['X坐标'].values,
point_i['Y坐标'].values,
point_j['X坐标'].values,
point_j['Y坐标'].values)
if (i==0 and j>12) or (j==0 and i>12):
weight_array[i][j]=0
weight_array[weight_array==0] = 10000

weight_array_A = weight_array[:13,:13]
G_A = Graph(weight_array_A)
pos_A = get_xy(G_A.nodenum)
edge_A = G_A.prim(first_node=0)
G_A.plot_matrix(pos_A)
```

```
G = Graph(weight_array, edge_A)
print('节点数据为%d, 边数为%d\n'%(G.nodenum, G.edgenum))
pos = get_xy(G.nodenum)
edge = G.prim()
G.plot_matrix(pos, title="Pipeline TWO")

sum = 0
for p in edge:
i,j=p[0],p[1]
sum = sum+weight_array[i][j]
sum
```

问题二代码实现及可视化

```
_max = (0,0,0)
_max2 = (0,0,0)
for ed in edge:
i,j = ed[0],ed[1]
if _max[0] < weight_array[i][j] and not (i<13 and j<13):
_max = (weight_array[i][j], i, j)
for ed in edge:
i,j = ed[0],ed[1]
if i!=126 and j!=125:
if _max2[0] < weight_array[i][j] and not (i<13 and j<13):
_max2 = (weight_array[i][j], i, j)

_max, _max2

_weight_ = G._get_edge()
plt.figure(figsize=(12,9))
G_nx = nx.Graph()
G_nx2 = nx.Graph()
G_nx3 = nx.Graph()
G_nx4 = nx.Graph()
GG = nx.Graph()
for i in range(G.nodenum):
for j in range(G.nodenum):
if _weight_[i, j]!=0 and i<13 and j<13:
G_nx.add_edge(i, j)
if _weight_[i, j]!=0 and i>0 and j>0:
G_nx2.add_edge(i, j)
# G_nx3.add_edge(126, 125)
# G_nx4.add_edge(88, 89)
G_nx3.add_node(125)
```

```
G_nx4.add_node(89)

nx.draw_networkx(G_nx3, {125:pos[125],},node_color='r',
node_size=300, node_shape='.',with_labels=False, style='dashed')
nx.draw_networkx(G_nx4, {89:pos[89],},node_color='r',
node_size=300, node_shape='.',with_labels=False, style='dashed')

nx.draw_networkx(G_nx, pos[:len(G.add_edge)+1], alpha=0.85)
nx.draw_networkx(G_nx2,pos,alpha=0.6,with_labels=False,node_color='slateblue',
node_shape=".", node_size=100, style='dashed')
GG = nx.Graph()
GG.add_node(0)
nx.draw_networkx(GG, {0:pos[0]}, node_color='r',node_shape='*', node_size=1200)

plt.title("Upgrade Two Secondary Pipes")
plt.show()
```

附录 B 问题三代码及其可视化