

基于”层次分析法”和”灰色预测模型”的 低碳建筑研究

摘要

关键词

1 问题重述

1.1 问题背景

“双碳”即碳达峰与碳中和的简称，我国力争 2030 年前实现碳达峰，2060 年前实现碳中和。“双碳”战略倡导绿色、环保、低碳的生活方式。我国加快降低碳排放步伐，大力推进绿色低碳科技创新，以提高产业和经济的全球竞争力。低碳建筑是指在建筑材料与设备制造、施工建造和建筑物使用的整个生命周期内，减少化石能源的使用，提高能效，降低二氧化碳排放量。

1.2 目标任务

问题一：计算给定建筑通过空调调节温度的年碳排放量。

问题二：建立综合评价模型，找出易于量化的指标，评估居住建筑整个生命周期的碳排放。

问题三：基于问题二，考虑建筑生命周期三个阶段的碳排放问题，对江苏省 13 个地级市的居住建筑进行评价，验证模型的有效性。

问题四：建立碳排放预测模型，基于江苏省建筑全过程碳排放的历史数据，对 2023 年江苏省建筑全过程的碳排放量进行预测。

问题五：结合前面的讨论给出江苏省建筑碳减排的政策建议。

2 问题分析

2.1 问题一

2.2 问题二

2.3 问题三

2.4 问题四

2.5 问题五

经过以上分析，我们对江苏省建筑碳减排的政策建议如下

1. 建筑设计标准：加强建筑能效等级和节能措施的规定，提高被动设计的实现率。鼓励使用绿色环保材料，减少建筑材料的使用量。
2. 促进可再生能源利用：鼓励建筑采用太阳能、地源热泵等可再生能源设施，减少对传统能源的依赖，降低运行阶段的碳排放量。
3. 改善建筑使用效率：推广智能化节能控制系统，优化空调、照明、通风等系统的运行方式，减少不必要的能源消耗。
4. 推广低碳生活方式：鼓励居民采用低碳出行方式，如步行、骑行、乘坐公共交通工具等，减少个人碳排放量。
5. 废弃物处理：鼓励采用回收利用的废弃物处理方式，减少拆除阶段的碳排放量。
6. 建立碳排放权交易市场：通过建立碳排放权交易市场，引导企业和个人减少碳排放，实现碳排放量的交易和管理。
7. 优化政策环境：加强碳排放标准的规定和执行力度，鼓励和支持低碳技术研发和应用。引导金融机构加大对低碳建筑项目的投资和支持力度。

8. 把节约能源资源放在首位，实行全面节约战略，倡导推广绿色低碳的生产生活方式，大幅提高投入产出效率，持续降低单位产出能源资源消耗和碳排放，从源头和入口形成有效的碳排放控制阀门。
9. 统筹省内外能源资源，推广先进绿色低碳技术和经验。推动省内应对气候变化对外合作有序开展，加强交流合作，不断提高参与度和影响力。
10. 加强低碳社会建设宣传教育，提高全社会对碳达峰、碳中和的认知度和认可度。引导和支持各类市场主体适应低碳发展要求，提升低碳创新水平。以示范创建为载体，推广绿色低碳生产生活方式，扩大绿色低碳产品供给和消费，倡导形成简约适度、绿色低碳的生活方式。
11. 政府和市场共同发挥作用，科技、产业和制度创新协同并进，增强原始创新支撑能力，加快全面数字化，深化能源等相关领域改革，形成有效激励约束机制，构建绿色低碳创新体系。
12. 推进工业低碳工艺革新、数字化转型和绿色制造体系建设，加快重点领域对照标杆水平实施节能降碳技术改造，鼓励国有企业、骨干企业开展示范性改造。
13. 大力培育节能环保、资源循环利用、清洁能源等绿色低碳产业，聚焦集成电路、生物医药、人工智能等前沿领域，积极发展新一代信息技术、新材料、新能源汽车等战略性新兴产业。

3 模型假设

4 符号说明

5 模型的建立与求解

5.1 问题一的模型建立与求解

问题一要求计算通过空题调节温度产生的年碳排放量。我们需先求出空调制热和制冷的热量，借此通过 COP 和 EER 求出空调消耗的电量，最后转换成碳排放。其中 COP 和 EER 的定义分别为

$$COP = \frac{Q_{heat}}{W}, \quad EER = \frac{Q_{cold}}{W} \quad (5.1)$$

Q_{heat}/Q_{cold} 指的是单位时间内的制热/制冷量，单位为 J ，公式中 W 指的是单位为时间内空调消耗的功率，单位为 W

首先计算出建筑物各个月的能量需求量。设室内温度要维持的温度为 t_{in} ，室外温度为 t_{out} ，当月该地区平均温度为 t_{ave} ，方便起见，我们规定

$$t_{out} = t_{ave}$$
$$t_{in} = \begin{cases} 18^{\circ}C & t_{out} < 18^{\circ}C \\ t_{out} & t_{out} \in [18^{\circ}C, 26^{\circ}C] \\ 26^{\circ}C & t_{out} > 26^{\circ}C \end{cases}$$

我们使用热传导方程计算用来需要制热/制冷的热量，其形式为

$$\Phi = \frac{\lambda \cdot A \cdot |\Delta T|}{\delta} \quad (5.2)$$

其中 Φ 表示传热速率， λ 为导热系数， A 为传热面积， ΔT 是室内外温度差，即 $t_{in} - t_{out}$ ， δ 表示材料厚度。

将建筑分成墙、门窗、房顶、地面四个部分，分别计算并累加即可得到需要制热/制冷的热量，设为 Q_{make} ，由 COP 和 EER 的定义可得到需电量

Q_{elec} 和热量 Q_{make} 的转化关系

$$Q_{elec} = \begin{cases} \frac{Q_{make}}{EER} & \Delta t < 0 \\ 0 & \Delta t = 0 \\ \frac{Q_{make}}{COP} & \Delta t > 0 \end{cases} \quad (5.3)$$

最后根据需电量与碳排放的换算关系 $m = Q_{elec} \cdot 0.28$ 求出每月碳排放后累加，即得到年度碳排放量。

5.2 问题二的模型建立与求解

5.2.1 建立层次结构模型

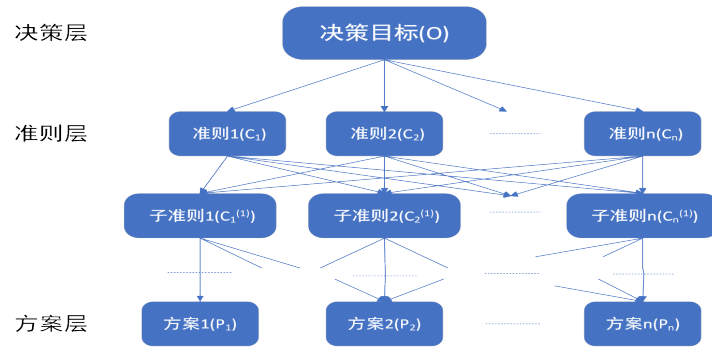


图 1: 层次分析法框架

准则层中准则因素之间相互独立。

我们选择的准则因素有：生活使用能耗、地区差异、周边产业、建造与拆除能耗、生产运输。

5.2.2 构建成对比较矩阵及归一化

1. 构建比较矩阵

构造比较矩阵是通过比较同一层次上的各因素对上-层相关因素的影响作用，而不是把所有因素放在一起比较，即将同一层的各因素进行两两对比。设某层有 n 个因素， $x = \{x_1, x_2 \dots x_n\}$ 要比较它们对上一层某一准则（或目标）的影响程度，确定在该层中相对于某一准则所

占的比重。上述比较是两两因素之间进行的比较，比较时常取 1 9 尺度。

| 尺度 | 含义 |
|------------|----------------------|
| 1 | 第 i 个因素与第 j 个因素影响相同 |
| 3 | 第 i 个因素与第 j 个因素影响稍强 |
| 5 | 第 i 个因素与第 j 个因素影响较强 |
| 7 | 第 i 个因素与第 j 个因素影响明显强 |
| 9 | 第 i 个因素与第 j 个因素影响极端强 |
| 2, 4, 6, 8 | 两相邻判断的中间值 |

用 a_{ij} 表示第 i 个因素相对于第 j 个因素的比较结果，则

$$a_{ij} = \frac{1}{a_{ji}}$$

$$A = (a_{ij})_{n \times n} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \quad (5.4)$$

A 则称为成对比较矩阵。

2. 归一化

对各城市的数据进行归一化处理

| 数据 \ 城市 | 苏州 | 南京 | 南通 | 无锡 | 常州 |
|---------|-------|-------|-------|-------|-------|
| 指标 | | | | | |
| 直接 | 48 | 51.5 | 27 | 57.2 | 29 |
| 间接 | 126.3 | 134.3 | 118.7 | 97.8 | 71.6 |
| 运营 | 4.8 | 4.9 | 3.5 | 2.7 | 2.8 |
| 归一化比例 | 0.226 | 0.242 | 0.127 | 0.269 | 0.126 |
| | 0.230 | 0.245 | 0.216 | 0.178 | 0.130 |
| | 0.257 | 0.262 | 0.187 | 0.144 | 0.150 |

5.2.3 层次单排序及一致性检验

1. 层次单排序

和积法（算术平均法）：取判断矩阵 n 个列向量归一化后的算术平均值，近似作为权重，即

$$W_i = \frac{1}{n} \sum_{j=1}^n \frac{a_{ij}}{\sum_{k=1}^n a_{kj}} (i = 1, 2, \dots, n) \quad (5.5)$$

求根法（几何平均法）：将比较矩阵的各列（或行）向量求几何平均后归一化，可近似作权重，即

$$W_i = \sum_{j=1}^n \frac{(\prod_{j=1}^n a_{ij})^{\frac{1}{n}}}{\sum_{k=1}^n (\prod_{j=1}^n a_{kj})^{\frac{1}{n}}} (i = 1, 2, \dots, n) \quad (5.6)$$

特征值法：求出矩阵的最大特征值以及其对应的特征向量，对求出的特征向量进行归一化即可得到我们的权重。

在我们的模型中综合采用了三种方法得到权重，降低了单一方法带来的不确定性，使数据结果更加可靠。

2. 一致性检验

通常情况下，由实际得到的判断矩阵不一定是一致的，即不一定满足

传递性和一致性实际中，也不必要求一致性绝对成立，但要求大体上是一致的，即不一致的程度应在容许的范围内主要考查以下指标：

(a) 一致性指标 CI

$$CI = \frac{\lambda_{\max} - n}{n - 1} \quad (5.7)$$

(b) 平均随机一致性指标 RI

为衡量 CI 的大小，引入随机一致性指标 RI ：

$$RI = \frac{CI_1 + CI_2 + \cdots + CI_n}{n} \quad (5.8)$$

其中，随机一致性指标 RI 和判断矩阵的阶数有关，一般情况下，矩阵阶数越大，则出现一致性随机偏离的可能性也越大，对于阶数小于 9，其对应关系如图：

| n | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|------|---|---|------|------|------|------|------|------|------|
| RI | 0 | 0 | 0.58 | 0.90 | 1.12 | 1.24 | 1.32 | 1.41 | 1.45 |

(c) 检验系数 CR

考虑到一致性的偏离有可能是由于随机原因造成的，因此在检验判断矩阵是否具有满意的一致性时，还需将 CI 和 RI 进行比较，得出检验系数 CR ，公式如下：

$$CR = \frac{CI}{RI} \quad (5.9)$$

一般地，如果 $CR \leq 0.1$ ，则认为该判断矩阵通过一致性检验， A_{\max} 对应的特征向量 W 可以作为排序的权重向量，此时

$$\lambda_{\max} \approx \sum_{i=1}^n \frac{(A \cdot W)_i}{nw_i} = \frac{1}{n} \sum_{i=1}^n \frac{\sum_{j=1}^n a_{ij}w_j}{w_i} \quad (5.10)$$

否则就不具有满意一致性，需要调整对比较矩阵。

5.2.4 计算组合权重及得分

得到最大特征值对应的特征向量

$$T = [t_1 \quad t_2 \quad \cdots \quad t_n]$$

计算得到权重向量

$$W = [w_1 \quad w_2 \quad \cdots \quad w_n]$$
$$w_i = \frac{t_i}{\sum_{i=1}^n t_i}$$

随后分别通过公式 (5.2) 和公式 (5.3) 求得两个权重向量，最后将三个权重向量取算术平均作为最终的权重向量。设指标评分矩阵为 P ，那么最后的得分矩阵 S 为：

$$S = P \cdot W$$

5.3 问题三的模型建立与求解

5.3.1 模型的建立

基于第二问所建模型，通过建筑生命周期的三个阶段确定代表性强的指标加入模型评价。我们选取的指标有：

建造阶段：建筑材料生产运输的碳排放系数、建造过程所产生的碳排放

运行阶段：生活使用能耗、气候原因、地区差异、周边产业

建造阶段：拆除过程的能耗

5.3.2 模型的验证

为了验证模型的有效性，我们将 2021 年江苏省 13 个地级市的居住建筑碳排放作为输入输入到所建立模型中进行综合评价，并将得到的结果排名与实际的排名进行比较，结果如图：

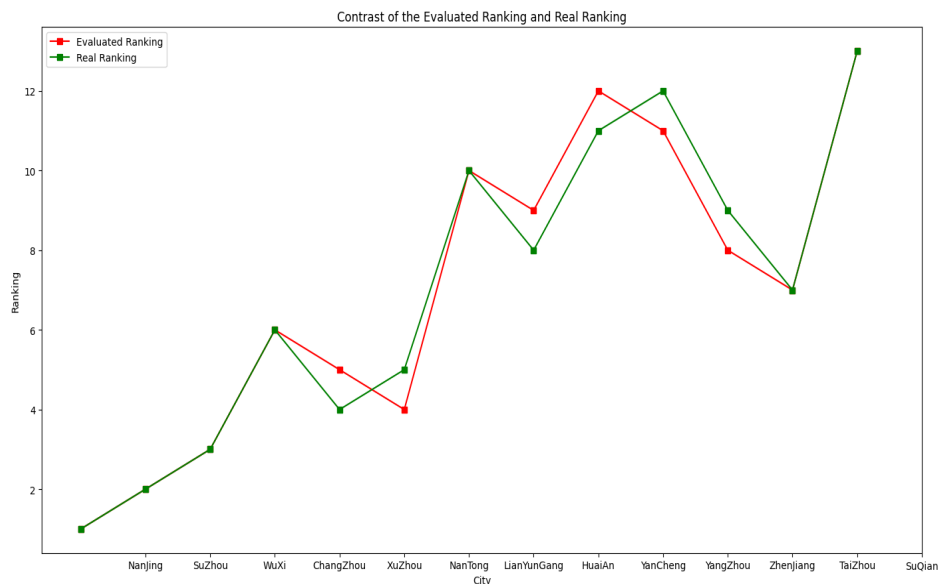


图 2: CONTRAST

通过比较来看，评价模型的评价结果与实际结果吻合得较好，很好的验证了所建立模型的有效性。

5.4 问题四的模型建立与求解

5.4.1 模型的分析

本题是分析预测 2023 年江苏省建筑全过程的碳排放量。由于要求基于江苏省建筑全过程碳排放的历史数据建立预测模型，用于分析江苏省未来建筑全过程的碳排放量，因此我们采用信息不完全、不充分的预测系统——灰色预测，建立灰色预测模型 GM (1,1) 模型，基于历史时期数据去预测未来时期数据。

5.4.2 模型的建立

1. 级比检验

级比检验用于判断数据序列进行模型构建的适用性，如果通过了该检

验，则可以使用灰色预测。计算公式为：

$$\lambda(k) = \frac{x^{(0)}(k-1)}{x^{(0)}(k)}, k = 2, 3, \dots, n \quad (5.11)$$

如果 $\lambda(k)$ 在区间

$$(e^{-\frac{2}{n+1}}, e^{\frac{2}{n+2}})$$

内，说明可用 GM (1,1) 模型，其中 $x^{(0)}(t)$ 是 年份-数据序列。

如果 $\lambda(k)$ 在该区间外，则可尝试通过平移变换，将每个数据都加上某一常数 c ，再看 $\lambda(k)$ 是否在区间内，最后模型结果再减去 c 即可。如果无论如何进行平移变换 $\lambda(k)$ 始终在该区间外，则说明该问题不适合用 GM (1, 1)。

而经过计算，我们的数据可以通过级比检验。

2. 构造累加生成序列

由于历史数据量少且无明显规律，难以预测，我们使用累加生成法得到一个新的序列 $x^{(1)}(t)$ ，并尝试用数学方法拟合新序列进而预测。累加生成序列公式为：

$$x^{(1)}(k) = \sum_{i=1}^k x^{(0)}(i) \quad (5.12)$$

3. 构建一阶常微分方程

通常情况下，累加生成序列的图像可用指数函数图像拟合，而一阶常微分方程的通解形式恰是指数函数，因此我们构建一阶常微分方程，通过求解这个方程得到预测函数。

该一阶常微分方程的形式为：

$$\frac{dx^{(1)}}{dt} + ax^{(1)} = u \quad (5.13)$$

其中 a, u 为未知参数。

5.4.3 模型的求解

1. 方程式改写

由于数据是离散而非连续的，首先将 $\frac{dx^{(1)}}{dt}$ 改写成 $\frac{\Delta x^{(1)}}{\Delta t}$ 。而由于 t 的单

位是年份,因此始终有 $\Delta t = (t+1)-t = 1$,因此我们得到新形式的方程

$$x^{(0)}(t) + ax^{(1)}(t) = u$$

移项得到:

$$x^{(0)}(t) = -ax^{(1)}(t) + u \quad (5.14)$$

得到该方程后, 注意到可以通过最小二乘法求得最佳的参数 a 和 u 。

2. 最小二乘法求解最佳参数

方程 (5.14) 的矩阵形式为 $Y = BU$, 具体形式为:

$$\begin{bmatrix} x^{(0)}(2) \\ x^{(0)}(3) \\ \vdots \\ x^{(0)}(N) \end{bmatrix} = \begin{bmatrix} -\frac{1}{2}[x^{(1)}(2) + x^{(1)}(1)] & 1 \\ -\frac{1}{2}[x^{(1)}(3) + x^{(1)}(2)] & 1 \\ \vdots & 1 \\ -\frac{1}{2}[x^{(1)}(N) + x^{(1)}(N-1)] & 1 \end{bmatrix} \begin{bmatrix} a \\ u \end{bmatrix} \quad (5.15)$$

我们要最小二乘法要求解的目标就是 $(Y - BU)^T(Y - BU)$ 去最小时的 U 。求解 U 的估计值的方法为

$$\hat{U} = [\hat{a}, \hat{u}] = (B^T B)^{-1} B^T Y$$

借此求得参数 a, u 的最佳近似值 \hat{a}, \hat{u} 后, 即可求解微分方程得到拟合函数。

3. 求解微分方程

通过方才求解到的 \hat{a}, \hat{u} , 代入方程 (5.14), 求得方程的解:

$$\hat{x}^{(1)}(k+1) = (x^{(0)}(1) - \frac{\hat{u}}{\hat{a}})e^{-\hat{a}k} + \frac{\hat{u}}{\hat{a}}, \quad k = 0, 1, \dots \quad (5.16)$$

当 k 小于已有数据量是, 求得的为拟合值, 大于时则是预测值。需要注意的是求得的为 $\hat{x}^{(1)}$, 是累加生成序列, 真正要求的原始序列 $\hat{x}^{(0)}$ 需要再通过差分法求得, 即:

$$\hat{x}^{(0)}(k) = \begin{cases} \hat{x}^{(1)}(1) & k = 1 \\ \hat{x}^{(1)}(k) - \hat{x}^{(1)}(k-1) & k > 1 \end{cases} \quad (5.17)$$

5.4.4 模型的检验

最后需要对我们的模型进行检验，判断拟合值是否能较好地逼近实际值。常用的检验方法有：级比偏差检验、残差检验、后验差比检验等。我们选择的方法是后验差比检验，通过后验差比值 C 的大小来判断模型的预测精度，其计算公式为：

$$C = \frac{\sigma_1}{\sigma_2} \quad (5.18)$$

其中 σ_1, σ_2 分别表示历史数据方差和残差方差。

若 C 值小于 0.35，则可说明模型精度高；若小于 0.5，则说明模型精度合格；若小于 0.65，则说明模型精度基本合格；若大于 0.65，则说明模型精度不合格。

模型拟合结果与预测和历史记录对比图如下：

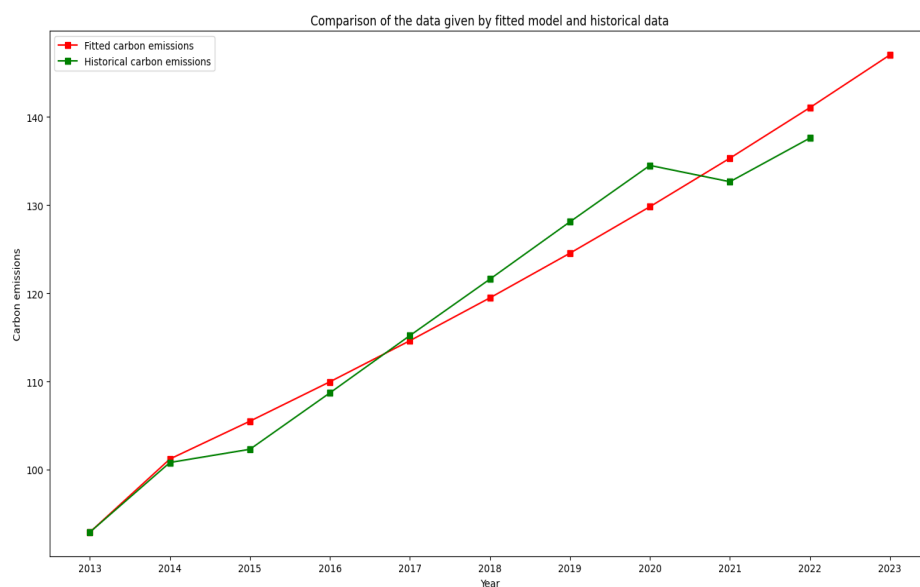


图 3: COMPARISON

6 结果检验与误差分析

7 模型评价

8 模型推广与改进

8.1 预测模型的改进

8.1.1 方程式修正

考虑到原方程中含有 $\frac{\Delta x^{(1)}}{\Delta t}$, 因此将 $x^{(1)}(t)$ 修正为均值生成序列 $z^{(1)}(t)$, 计算公式为

$$z^{(1)}(t) = \frac{x^{(1)}(t)}{2} + \frac{x^{(1)}(t-1)}{2} + \cdots, \quad t = 2, \dots, n$$

即方程改写为

$$x^{(0)}(t) = -az^{(1)}(t) + u$$

9 参考文献

10 附录

附录 A 问题一

Listing 1: Question1

```
1 import matplotlib.pyplot as plt
2
3 monthdays = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
4 temperatures = [-1, 2, 6, 12, 22, 28, 31, 32, 26, 23, 15, 2]
5
6 COP = 3.5
7 EER = 2.7
8
9 length = 4
10 width = 3
11 height = 3
12
13 roof_thickness = 0.3
14 roof_thermal = 0.2
15 roof_square = length * width
16 windowAndDoor_thickness = 0.1
17 windowAndDoor_thermal = 1.6
18 windowAndDoor_square = 5
19 wall_thickness = 0.3
20 wall_thermal = 0.3
21 wall_square = (length + width) * height * 2 - windowAndDoor_square
22 ground_thickness = 1.0
23 ground_thermal = 0.25
24 ground_square = length * width
25
26
27 def get_delta_t(tout):
28     if tout < 18:
29         return 18 - tout
30     elif tout > 26:
31         return 26 - tout
32     else:
33         return 0
34
35
36 def get_qmake(k, a, t, d):
```

```

37     return (k * a * t) / d
38
39
40 def get_day_qmake(dt):
41     if dt == 0:
42         return 0.0
43
44     day_qmake = \
45         get_qmake(wall_thermal, wall_square, dt, wall_thickness) + \
46         get_qmake(windowAndDoor_thermal, windowAndDoor_square, dt,
47                     windowAndDoor_thickness) + \
48         get_qmake(roof_thermal, roof_square, dt, roof_thickness) + \
49         get_qmake(ground_thermal, ground_square, dt, ground_thickness)
50     return day_qmake * 86400
51
52 def get_month_qmake(month):
53     return get_day_qmake( abs(get_delta_t(temperatures[month]))) * monthdays[month]
54
55
56 def get_qelec(qmake, dt):
57     if dt < 0:
58         return qmake / EER
59     elif dt == 0:
60         return 0
61     else:
62         return qmake / COP
63
64
65 def get_carbon_emission(qelec):
66     return (qelec / 3600000) * 0.28
67
68
69 carbonEmissions = []
70 for i in range(0, 12):
71     qelec = get_qelec(get_month_qmake(i), get_delta_t(temperatures[i]))
72     carbonEmissions.append(get_carbon_emission(qelec))
73
74 print("Sum of carbon emissions: " + str( sum(carbonEmissions)) + "kg")
75
76 plt.figure(figsize=(10, 6))
77 plt.bar( range(1, len(carbonEmissions) + 1), carbonEmissions, fc='g')

```



```

78 plt.title("Carbon Emissions")
79 plt.xlabel("Month")
80 plt.ylabel("kg")
81
82 plt.show()

```

附录 B 问题二

Listing 2: Question2

```

1  import numpy as np
2  import matplotlib.pyplot as plt
3
4
5  cities = ["SuZhou", "NanJing", "NanTong", "WuXi", "ChangZhou"]
6
7  discriminantMatrix = np.array([[1, 1/5, 1/5], [5, 1, 1/2], [5, 2, 1]])
8  eigens = np.linalg.eig(discriminantMatrix)
9  maxEigenvalue = np. max(eigens[0])
10 rowAndColumn = np.argwhere(eigens[0] == maxEigenvalue)
11 maxEigenvector = eigens[1][::-1, rowAndColumn[0]]
12
13 RI_LIST = [0, 0.001, 0.58, 0.9, 1.12, 1.24, 1.32, 1.41, 1.45, 1.49, 1.52, 1.54, 1.56,
14            1.58, 1.59]
15 dim = discriminantMatrix.shape[0]
16 RI = RI_LIST[dim]
17 CI = (maxEigenvalue - dim) / (dim - 1)
18 CR = CI / RI
19 print("CR: " + str(CR))
20 print("Passed consistency test") if CR < 0.1 else print("Failed to passed consistency
    test")
21
22 Scores = np.array([[48.0, 126.3, 4.8],
23                   [51.5, 134.3, 4.9],
24                   [27, 118.7, 3.5],
25                   [57.2, 97.8, 2.7],
26                   [29, 71.6, 2.8]])
27
28 sums = np. sum(Scores, axis=0)
29 weightedScores = Scores / sums.reshape(1, -1)

```

```

30 # Eigenvalue average
31 eigenvectorWeight = maxEigenvector / sum(maxEigenvector)
32
33 # Arithmetic average
34 arithmeticSums = np.sum(discriminantMatrix, axis=0)
35 normalizedMatrix = discriminantMatrix / arithmeticSums.reshape(1, -1)
36 arithmeticWeight = np.sum(normalizedMatrix, axis=1)
37 arithmeticWeight /= dim
38 arithmeticWeight = arithmeticWeight[-1::-1]
39
40 # Geometric average
41 prodVector = np.prod(discriminantMatrix, axis=1)
42 prodVector = np.power(prodVector, 1/dim)
43 prodSums = np.sum(prodVector, axis=0)
44 geometricWeight = prodVector / prodSums
45 geometricWeight = geometricWeight[-1::-1]
46
47 weight = (arithmeticWeight + geometricWeight) / 2
48
49 finalScores = np.dot(weightedScores, weight)
50 for i in range(len(finalScores)):
51     print('City {:}, Scores {:}'.format(cities[i], finalScores[i]))
52
53 plt.figure(figsize=(10, 6))
54 x = [1, 2, 3, 4, 5]
55 x_label = cities
56 plt.bar(x, finalScores, fc='g')
57 plt.title("Scores given by evaluation model")
58 plt.xticks(x, x_label)
59 plt.xlabel("City")
60 plt.ylabel("Evaluation score")
61
62 plt.show()

```

附录 C 问题三

Listing 3: Question3

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3

```

```

4
5 cities = ["NanJing", "SuZhou", "WuXi", "ChangZhou", "XuZhou", "NanTong", "LianYunGang"
6         ,
7           "HuaiAn", "YanCheng", "YangZhou", "ZhenJiang", "TaiZhou", "SuQian"]
8
9 discriminantMatrix = np.array([[1, 1/5, 1/5], [5, 1, 1/2], [5, 2, 1]])
10 eigens = np.linalg.eig(discriminantMatrix)
11 maxEigenvalue = np. max(eigens[0])
12 rowAndColumn = np.argwhere(eigens[0] == maxEigenvalue)
13 maxEigenvector = eigens[1][:,-1, rowAndColumn[0]]
14
15 RI_LIST = [0, 0.001, 0.58, 0.9, 1.12, 1.24, 1.32, 1.41, 1.45, 1.49, 1.52, 1.54, 1.56,
16            1.58, 1.59]
17 dim = discriminantMatrix.shape[0]
18 RI = RI_LIST[dim]
19 CI = (maxEigenvalue - dim) / (dim - 1)
20 CR = CI / RI
21 print("CR: " + str(CR))
22 print("Passed consistency test") if CR < 0.1 else print("Failed to passed consistency
23 test")
24
25 Scores = np.array([[51.5, 134.3, 4.9],
26                    [48.6, 126.3, 4.8],
27                    [57.2, 97.8, 2.7],
28                    [29, 71.6, 2.8],
29                    [30.1, 79.6, 2.91],
30                    [27, 118.7, 3.5],
31                    [12.4, 36.4, 1.32],
32                    [13.5, 32.5, 1.34],
33                    [16.7, 45.2, 1.72],
34                    [20.4, 52.3, 2.01],
35                    [14.01, 36.1, 1.401],
36                    [14.2, 37.3, 1.412],
37                    [9.27, 25.3, 0.965]])
38
39 sums = np. sum(Scores, axis=0)
40 weightedScores = Scores / sums.reshape(1, -1)
41
42 # Eigenvalue average
43 eigenvectorWeight = maxEigenvector / sum(maxEigenvector)
44
45 # Arithmetic average

```

```

43 arithmeticSums = np. sum(discriminantMatrix, axis=0)
44 normalizedMatrix = discriminantMatrix / arithmeticSums.reshape(1, -1)
45 arithmeticWeight = np. sum(normalizedMatrix, axis=1)
46 arithmeticWeight /= dim
47 arithmeticWeight = arithmeticWeight[-1::-1]
48
49 # Geometric average
50 prodVector = np.prod(discriminantMatrix, axis=1)
51 prodVector = np.power(prodVector, 1/dim)
52 prodSums = np. sum(prodVector, axis=0)
53 geometricWeight = prodVector / prodSums
54 geometricWeight = geometricWeight[-1::-1]
55
56 weight = (arithmeticWeight + geometricWeight) / 2
57
58 finalScores = np.dot(weightedScores, weight)
59 for i in range( len(finalScores)):
60     print('City {:}, Scores {:}'. format(cities[i], finalScores[i]))
61
62 sortedScores = sorted(
63     enumerate(finalScores), key=lambda finalScores:finalScores[1], reverse=True)
64 finalRanking = [finalScores[0] + 1 for finalScores in sortedScores]
65 realRanking = [1, 2, 3, 6, 4, 5, 10, 8, 11, 12, 9, 7, 13]
66
67 plt.figure(figsize=(20, 12))
68 x = range(1, 14)
69 x_label = cities
70 plt.plot(x_label, finalRanking, 's-', color='r', label="Evaluated Ranking")
71 plt.plot(x_label, realRanking, 's-', color='g', label="Real Ranking")
72
73 plt.title("Contrast of the Evaluated Ranking and Real Ranking")
74 plt.xticks(x, x_label)
75 plt.xlabel("City")
76 plt.ylabel("Ranking")
77 plt.legend(loc="best")
78
79 plt.show()

```

附录 D 问题四

Listing 4: Question4

```
1 import numpy as np
2 import math
3 import matplotlib.pyplot as plt
4
5 historicalData = [92.9, 100.8, 102.3, 108.7, 115.2, 121.6, 128.1, 134.5, 132.65,
6                  137.6]
7 years = len(historicalData)
8 Data0 = np.array(historicalData)
9
10 e = math.e
11
12 # 级比检验
13 def comparison_test():
14     left_border = math.exp(float(-2) / (years + 1))
15     right_border = math.exp(float(2) / (years + 2))
16     for k in range(2, years):
17         test_value = float(Data0[k - 1]) / Data0[k]
18         if test_value < left_border or test_value > right_border:
19             print(k, test_value)
20             print(left_border, right_border)
21             return False
22     return True
23
24
25 def get_prefix_sum_sequence():
26     history_data_prefix_sum = [sum(historicalData[0: i + 1]) for i in range(years)]
27     return np.array(history_data_prefix_sum)
28
29
30 def get_y_and_b(data1):
31     y = np.zeros([years - 1, 1])
32     b = np.zeros([years - 1, 2])
33     for i in range(0, years - 1):
34         y[i][0] = historicalData[i + 1]
35         b[i][0] = -0.5 * (data1[i] + data1[i + 1])
36         b[i][1] = 1
37     return y, b
38
39
40 def calc_a_and_u(y, b):
```

```

41     _u = np.linalg.inv(b.T.dot(b)).dot(b.T).dot(y)
42     return _u[0][0], _u[1][0]
43
44
45 def get_fitted_data1(_a, _u):
46     fitted_data1 = np.zeros(years + 1)
47     fitted_data1[0] = Data0[0]
48     for i in range(1, years + 1):
49         fitted_data1[i] = (Data0[0] - _u / _a) * math.exp(-_a * i) + _u / _a
50     return fitted_data1
51
52
53 def get_diff_ave(fitted_data1):
54     diff_ave = 0.0
55     for i in range(0, years):
56         diff_ave += (Data0[i] - fitted_data1[i])
57     return diff_ave / years
58
59
60 def get_diff_variance(fitted_data1):
61     diff_ave = get_diff_ave(fitted_data1)
62     _diff_var = 0.0
63     for i in range(0, years):
64         _diff_var += (Data0[i] - fitted_data1[i] - diff_ave) ** 2
65     return _diff_var / years
66
67
68 def get_histo_ave():
69     return sum(Data0) / years
70
71
72 def get_histo_variance():
73     histo_ave = get_histo_ave()
74     _histo_var = 0.0
75     for i in range(0, years):
76         _histo_var += (Data0[i] - histo_ave) ** 2
77     return _histo_var / years
78
79
80 def get_fitted_data0(fitted_data1):
81     fitted_data0 = np.zeros(years + 1)
82     fitted_data0[0] = fitted_data1[0]

```

```

83     for i in range(1, years + 1):
84         fitted_data0[i] = fitted_data1[i] - fitted_data1[i - 1]
85     return fitted_data0
86
87
88 def test_model_accuracy(_histo_var, _diff_var):
89     _c = _histo_var / _diff_var
90     if _c < 0.35:
91         print("High model accuracy!")
92     elif _c < 0.5:
93         print("Qualified model accuracy!")
94     elif _c < 0.65:
95         print("Basically qualified model accuracy!")
96     else:
97         print("Unqualified model accuracy!")
98         return False
99     return True
100
101
102 if __name__ == '__main__':
103     if not comparison_test():
104         print("Failed to pass comparison test!")
105         exit(0)
106
107     print("Passed comparison test!")
108
109     Data1 = get_prefix_sum_sequence()
110     Y, B = get_y_and_b(Data1)
111     a, u = calc_a_and_u(Y, B)
112     fittedData1 = get_fitted_data1(a, u)
113     histo_var = get_histo_variance()
114     diff_var = get_diff_variance(fittedData1)
115     fittedData0 = get_fitted_data0(fittedData1)
116
117     if not test_model_accuracy(histo_var, diff_var):
118         print("Using Grey Prediction model is not suitable!")
119         exit(0)
120
121     print("Predict: " + str(fittedData0[-1]))
122
123     plt.figure(figsize=(20, 12))
124     x0_label = range(2013, 2024)

```

```
125     x1_label = range(2013, 2023)
126     plt.plot(x0_label, fittedData0, 's-', color='r', label="Fitted carbon emissions")
127     plt.plot(x1_label, Data0, 's-', color='g', label="Historical carbon emissions")
128
129     plt.title("Comparison of the data given by fitted model and historical data")
130     plt.xticks(x0_label)
131     plt.xlabel("Year")
132     plt.ylabel("Carbon emissions")
133     plt.legend(loc="best")
134
135     plt.show()
```