分布式能源配电网风险分析

2025 年 5 月 17 日

摘要

摘要

关键词： 关键词1，关键词2，关键词3

1 问题重述

1.1 问题背景

随着全球能源转型和我国"双碳"目标的深入推进，可再生分布式能源（如光伏、风电）在配电网中的渗透率显著提升。这种趋势不仅推动了清洁能源的利用，还对传统配电网的运行方式提出了新的挑战。传统配电网设计以集中式发电和单向潮流为主，难以适应分布式能源出力波动性和不确定性带来的复杂运行场景。分布式能源的接入改变了配电网的潮流分布，可能导致失负荷（因故障导致供电中断）或过负荷（线路电流超额定载流量10%以上），从而增加系统运行风险。此外，联经开关的存在为故障后功率转移提供了可能，但其操作复杂性进一步增加了风险评估的难度。在实际应用中，不同类型用户（如工业、商业、政府机构）对供电中断的敏感度差异显著，需综合考虑经济损失和社会影响来量化风险。现有研究多聚焦于分布式能源的接入技术或单一风险因素，缺乏对失负荷与过负荷风险的系统性建模与综合评估。本研究基于62节点有源配电网系统，结合分布式能源的运行特性，构建风险评估模型，为配电网的规划与优化提供理论支持。这种研究不仅在理论上丰富了配电网风险管理的知识体系，还在实践中为提升电网可靠性和经济性提供了科学指导，具有重要的现实意义。

1.2 问题提出

本研究旨在通过数学建模量化分布式能源接入配电网后引发的失负荷与过负荷风险，为配电网的稳定运行提供系统化解决方案。研究围绕以下核心任务展开：首先，需建立分布式能源接入配电网后失负荷与过负荷风险的数学模型，明确风险的概率分布及其危害程度，结合联经开关的功率转移机制优化风险计算。然后，基于62节点有源配电网系统，应用所建模型分析分布式能源接入对系统风险的动态影响，揭示风险演变的规律。这些任务通过理论建模、数据分析与仿真验证相结合，旨在为配电网运行提供科学的决策依据，提升其在高渗透率分布式能源场景下的可靠性和经济性。

2 问题分析

分布式能源接入配电网的风险评估涉及失负荷与过负荷的概率建模、风险演变分析、光伏容量影响评估及储能优化等多个子问题，呈现出高度的复杂性和综合性。问题的核心在于分布式能源出力的波动性与不确定性对配电网潮流分布的深刻影响，这不仅改变了传统配电网的运行模式，还引入了新的风险来源。失负荷风险源于故障导致的供电中断，其危害程度与用户类型和停电持续时间密切相关；过负荷风险则由线路电流超载引发，可能导致设备损坏或系统崩溃。联经开关的存在为功率转移提供了灵活性，但其操作需考虑网络拓扑的动态变化，增加了建模难度。

2.1 模型假设

为了简化模型分析，本文在建模过程中做出以下假设：

1. 假设分布式能源出力为固定值，且不随时间变化。
2. 假设配电网中所有节点的负荷均为已知值，且不随时间变化。
3. 假设配电网的拓扑结构在分析期间保持不变，且电压恒定。
4. 假设配电网中联络开关的容量足够大，能够承载所有可能的功率转移。
5. 假设配电系统故障发生独立，且故障率固定。

2.2 符号说明

符号	含义
R_{sys}	系统风险
P_{LL}	失负荷概率
C_{LL}	失负荷危害程度
P_{OL}	过负荷概率
C_{OL}	过负荷危害程度
$\mathcal{N}$	配电网中节点的集合
$\mathcal{L}$	配电网中线路的集合
$\mathcal{S}$	配电网中开关的集合
R_i	元件 i 的风险
P_i^{dg}	分布式发电节点 i 的出力功率
U	配电网的电压
I_{rated}	线路的额定电流

表 1: 符号说明

3 模型建立与求解

3.1 问题 1 建模与求解

3.1.1 问题 1 求解思路

问题 1 要求建立分布式能源接入配电网后失负荷与过负荷风险的数学模型，核心目标在于量化风险概率和危害程度，为进一步的分析奠定基础。分布式能源的出力波动与联络开关的功率转移特性是影响风险评估的关键因素。由于题目中指出，同一时刻同一类型只发生一个故障，因此可以独立分析电网中的各个节点，分别分析对应的风险。首先，针对失负荷风险，需要分析故障单元的故障率，结合网络拓扑和联络开关状态计算失负荷概率和风险期望。此外，考虑到不同用户类型的危害程度差异，还需要引入权重因子，量化危害程度。其次，针对过负荷风险，需要分析电路中电流的分布情况，结合联络开关的功率转移特性，判断是否超过额定载流量的 10%。最后，综合考虑失负荷与过负荷风险，建立综合风险评估模型。

技术路线上，我们选择使用 Python 作为主要工具，利用 pandas 和 numpy 分析负荷和出力数据，结合 networkx 库进行图论分析， matplotlib 库进行可视化。潜在挑战在于对于大规模节点计算的准确性，以及联络开关状态变化对风险评估的影响。

3.1.2 问题 1 建模

为了量化分布式能源接入配电网后失负荷与过负荷风险，我们首先需要建立数学模型来描述系统的运行状态和风险评估指标。模型以配电网的拓扑结构、负荷分布、分布式能源的出力以及联经开关的状态为输入，输出系统的风险值，风险计算公式题目中已经给出为：

$$R_{\text{sys}} = P_{\text{LL}} \cdot C_{\text{LL}} + P_{\text{OL}} \cdot C_{\text{OL}} \tag{1}$$

其中 P_{LL} 和 P_{OL} 分别表示失负荷和过负荷的概率，C_{LL} 和 C_{OL} 分别表示失负荷和过负荷的危害程度。

在模型中，由于假设 5，我们可以将不同节点的故障风险独立分析。因此，我们选择枚举所有可能的故障单元，结合网络拓扑和联络开关转移功率，计算故障后的失负荷损失。具体而言，我们可以针对故障单元的类型来讨论对应的风险评估。

当故障单元为负载时，我们认为失负荷只是因为变电站到负荷的线路故障导致，因此负载失负荷后可以从同网络的其他 dg 节点接收。同时特殊考虑 dg 节点，认为它的失负荷损失为其出力功率。权重因子的设置根据不同用户类型的敏感度进行调整，我们使用的具体数值如表 2 所示。 形

用户类型	权重因子
工业用户	1.0
政府机构	0.8
商业用户	0.5
居民	0.2

表 2: 不同用户类型的权重因子

式化地说，假设故障单元为 i ，其对应的负载功率或出力为 P_i ，同网络的剩余容量为 P_{remain} ，权重因子为 w_i ，则失负荷损失为：

$$L_{\text{node},i} = \begin{cases} (P_i - P_{\text{remain}}) \cdot w_i & \text{if } i \text{ is a load} \\ P_i^{dg} & \text{if } i \text{ is a dg node} \end{cases} \tag{2}$$

当故障单元为线路时，需要考虑其影响到的节点以及联络开关的状态。首先，我们将每条馈线抽象为一棵以变电站为根节点的树，树的每个节点都是一个负载。那么，线路断开意味着这棵树被分为两棵树，其中一棵树的根节点为变电站，另一棵树的根节点为故障单元。这种情况下，考虑联络开关的影响，若被影响的子树可以通过联络开关转移功率，由于前述假设 4 ，则可以假定相连馈线的所有多余功率都可以被转移到故障子树。另外，还要考虑故障子树中的 dg 节点的出力。为了简化分析，此时我们计算被影响子树的平均权重，以此作为这棵子树的权重，乘以其失负荷的功率作为此时的损失。形式化地说，假设故障边的编号为 i ，故障子树的节点集合为 $\mathcal{N}$ ，对应的负载功率的和为 $P_{\mathcal{N}}$ ， dg 节点的出力功率的和为 $P_{\mathcal{N}}^{dg}$ ，从其他馈线转移的功率为 P_{transfer} ，则失负荷损失为：

$$L_{\text{line},i} = \max\left(0, \frac{\sum_{k\in\mathcal{N}} w_k}{|\mathcal{N}|} \cdot (P_{\mathcal{N}} - P_{\text{transfer}} - P_{\mathcal{N}}^{dg})\right) \tag{3}$$

当故障单元为开关时，为了简化处理，我们认为它不会影响到任何节点的功率转移，即：

$$\forall i \in \mathcal{S}, \quad L_{\text{switch},i} = 0 \tag{4}$$

综上所述，我们可以将失负荷风险 $P_{\text{LL}} \cdot C_{\text{LL}}$ 表示为：

$$P_{\text{LL}} \cdot C_{\text{LL}} = \sum_{i\in\mathcal{N}} R_i \cdot L_{\text{node},i} + \sum_{j\in\mathcal{L}} R_j \cdot L_{\text{line},j} + \sum_{k\in\mathcal{S}} R_k \cdot L_{\text{switch},k} \tag{5}$$

其中 $\mathcal{N}$、$\mathcal{L}$ 和 $\mathcal{S}$ 分别表示故障单元为负载、线路和开关的集合。

对于过负荷风险，我们需要计算每条馈线的电流情况，判断是否超过额定载流量的 10%。这里我们假定馈线的电流为三相交流电，那么有功率与电压的关系为：

$$I = \frac{P}{\sqrt{3} \cdot U} \tag{6}$$

其中 I 为电流，P 为功率，U 为电压。

根据假设 1 和 2，我们可以认为负载功率和出力功率是固定的。又由假设 3，所以可以认为馈线电流与馈线总净功率成线性关系，因此只需计算馈线的总净功率。

首先计算馈线 i 的总净功率 $P_{\text{net},i}$：

$$P_{\text{net},i} = \sum_{j \in \mathcal{N}_i} P_j - \sum_{k \in \mathcal{DG}_i} P_k^{\text{dg}} \tag{7}$$

其中 $\mathcal{N}_i$ 和 $\mathcal{DG}_i$ 分别表示馈线 i 上的负载节点和分布式发电节点集合。当 $P_{\text{net},i} > 0$ 时，表示馈线 i 上的负载大于分布式发电节点的出力，此时我们需要计算电流 I_i：

$$I_i = \frac{P_{\text{net},i}}{\sqrt{3} \cdot U} \tag{8}$$

否则，则考虑联络开关的功率转移。假设联络开关能够转移的最大功率为 $P_{\text{transfer},i}$，则此时的电流 I_i 为：

$$I_i = \frac{|P_{\text{net},i}| - P_{\text{transfer},i}}{\sqrt{3} \cdot U} \tag{9}$$

然而，由于实际上的负载和出力功率是有波动的，为了简化分析，我们认为实际上的电流成正态分布，期望值为 I_i，标准差为 $\sigma_i = 0.1 \cdot I_i$。同时，由于电流本身即使没有过载也会导致线路老化等损失，因此我们给出一个分段线性的损失函数如下：

$$L_{\text{current},i}(I) = \begin{cases} 0.2\sqrt{3}U \frac{I}{I_{\text{rated}} \cdot 1.1} \cdot \bar{w}_i & \text{if } I_i < 1.1 \cdot I_{\text{rated}} \\ 2\sqrt{3}U(I_i - 1.1 \cdot I_{\text{rated}} + 0.1) \cdot \bar{w}_i & \text{if } I_i \geq 1.1 \cdot I_{\text{rated}} \end{cases} \tag{10}$$

其中 I_{rated} 为馈线的额定电流，$\bar{w}_i$ 为馈线 i 的平均权重。从而计算期望损失：

$$P_{\text{OL}} \cdot C_{\text{OL}} = \sum_i E[L_{\text{current},i}] = \sum_i \int_{-\infty}^{+\infty} L_{\text{current},i}(I) \cdot N(I_i,\ 0.1I_i) dI \tag{11}$$

其中 $N(I_i,\ 0.1I_i)$ 为正态分布函数。

综上所述，可以根据公式 1，5 和 11 计算系统的风险值 R_{sys}。

3.1.3 问题 1 求解

基于所建立的数学模型，我们使用 Python 对题目给出的 62 节点有源配电网系统进行建模与求解。我们首先使用 networkx 库构建配电网的拓扑结构，利用 pandas 和 numpy 库处理负荷和出力数据。然后，针对每个节点的故障单元，计算其失负荷和过负荷风险。最后，综合所有节点的风险值，得到系统的总风险值 R_{sys}。

首先有网络拓扑图，如图 1 所示。

接下来通过 Python 代码实现上述模型的求解过程，计算出系统的失负荷和过负荷风险。分析结果表明，系统风险主要来自于失负荷风险，占总风险的 95.51%，而过负荷风险仅占 4.49%。总的风险值为 36.82 ，总风险相对较高。

3.2 问题 2 建模与求解

3.2.1 问题 2 求解思路

问题 2 要求分析分布式能源接入对系统风险的动态影响，揭示风险演变的规律。核心目标是通过仿真分析分布式能源接入容量、位置等因素对系统风险的影响，量化风险演变的趋势。我们可以通过改变分布式能源的接入容量，观察系统风险的变化。技术路线依然使用 Python 进行仿真分析，结合 matplotlib 库进行可视化。

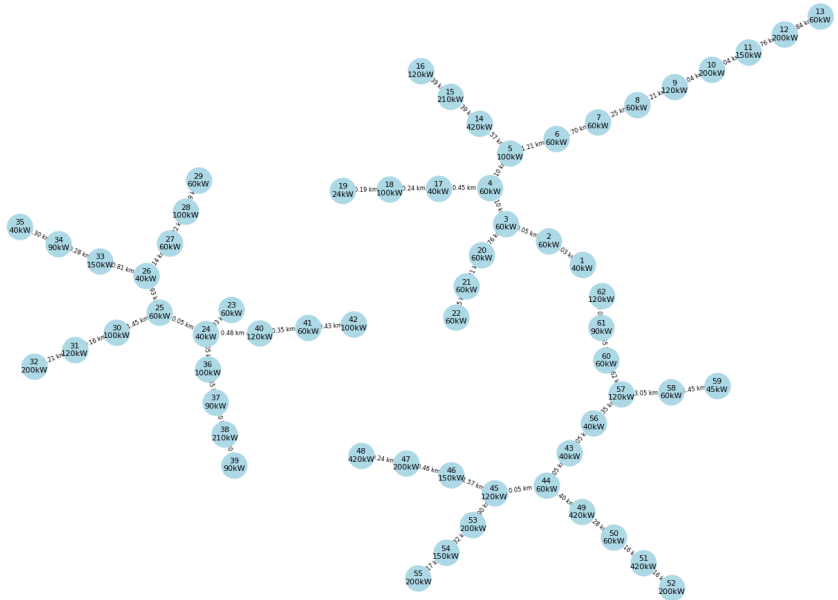

图 1: 62节点有源配电网拓扑结构

3.2.2 问题 2 建模

为了衡量每个馈线承担的风险，我们为每个馈线定义风险占比。假设整张图的边集和点集是 $\mathcal{E}$ 和 $\mathcal{N}$ ，馈线 i 的边集与点集分别为 $\mathcal{E}_i$ 和 $\mathcal{N}_i$，则馈线 i 的风险占比 $R_{\text{ratio},i}$ 定义为：

$$R_{\text{ratio},i} = \frac{\sum_{j\in\mathcal{N}_i} R_j + \sum_{k\in\mathcal{E}_i} R_k}{\sum_{j\in\mathcal{N}} R_j + \sum_{k\in\mathcal{E}} R_k} \tag{12}$$

从而可以计算出每个馈线的风险占比。

然后，我们可以通过改变分布式能源的接入容量，观察系统风险的变化。

3.2.3 问题 2 求解

基于所建立的数学模型，我们使用 Python 对题目给出的 62 节点有源配电网系统进行建模与求解。

首先使用 numpy 库生成分布式能源的接入容量数据，假设分布式能源的接入容量系数为 I 到 $3I$ ，步长为 $0.3I$ ，其中 I 为 dg 节点的出力功率。然后，针对每个节点的故障单元，计算其失负荷和过负荷风险。计算结果表 3 所示。

容量因子	总DG容量(kW)	失负荷风险	过负荷风险	系统总风险
1.00	2400.00	35.17	1.65	36.82
1.30	3120.00	31.57	1.41	32.97
1.60	3840.00	28.15	1.16	29.31
1.90	4560.00	26.86	0.91	27.77
2.20	5280.00	29.33	1.37	30.70
2.50	6000.00	32.53	1.22	33.75
2.80	6720.00	36.09	1.96	38.05
3.10	7440.00	39.68	1.90	41.58

表 3: 分布式能源容量对系统风险的影响

同时，我们使用 matplotlib 库绘制系统风险随分布式能源接入容量变化的曲线图，如图 2 所示。

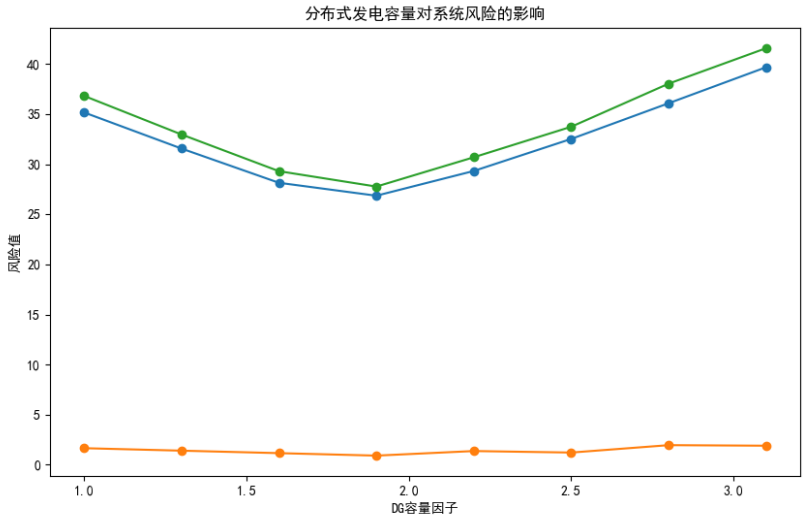

图 2: 分布式能源接入容量对系统风险的影响

从图中可以看出，系统风险随着分布式能源接入容量的增加而呈现出先下降后上升的趋势。在接入容量较小时，系统风险随着接入容量的增加而下降，说明分布式能源的接入能够有效降低系统风险。然而，当接入容量达到一定程度后，系统风险又开始上升，这可能是由于 dg 节点本身的出力波动性和不确定性导致的。

同时，还有分馈线的风险占比随分布式能源接入容量变化的曲线图，如图 3 所示。

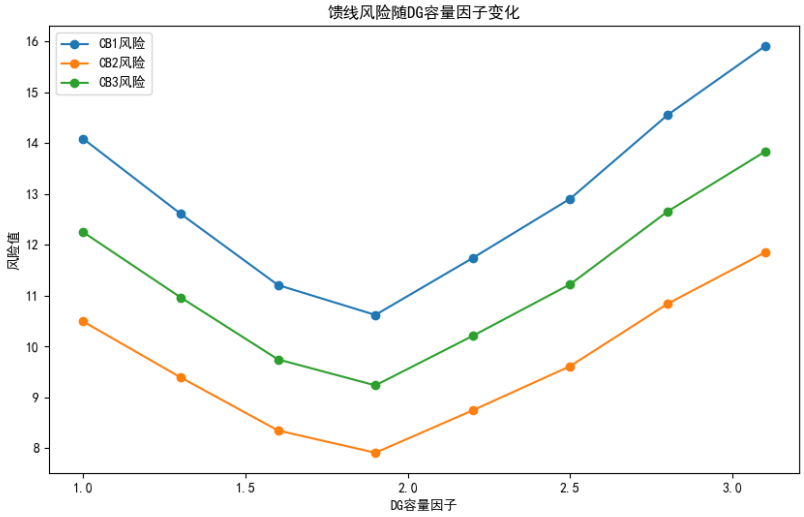

图 3: 馈线风险占比随分布式能源接入容量变化

经过以上分析，可以得出：当容量因子为 1.9 时，系统的风险最低，降低到了约 27.77。

4 模型评价与推广

4.1 模型的优点

本研究的模型具有以下优点：

1. 综合考虑了失负荷与过负荷风险，能够全面评估分布式能源接入对配电网的影响。
2. 采用了图论分析方法，能够有效处理大规模配电网的拓扑结构。
3. 模型本身较为简单，易于实现，计算效率高。
4. 结合了数据分析与仿真验证，能够提供可靠的风险评估结果。
5. 为配电网的规划与优化提供了理论支持，具有重要的现实意义。

4.2 模型的不足

本研究的模型也存在一些不足之处：

1. 假设分布式能源出力为固定值，未考虑其波动性和不确定性对风险评估的影响。
2. 模型中未考虑配电网的动态特性，可能导致风险评估结果的偏差。
3. 模型中使用的权重因子和损失函数参数缺乏实证依据，可能影响模型的准确性。
4. 模型的推广性有限，可能不适用于所有类型的配电网。
5. 模型结构过于简单，未考虑其他可能影响风险的因素，如天气、负荷变化等。

4.3 模型的推广和优化

针对模型的不足之处，我们可以考虑以下优化和推广方向：

1. 引入分布式能源出力的波动性和不确定性，采用随机建模方法进行风险评估。
2. 考虑配电网的动态特性，建立动态风险评估模型。
3. 基于实证数据优化权重因子和损失函数参数，提高模型的准确性。
4. 扩展模型的适用范围，考虑不同类型配电网的特点。
5. 引入其他影响因素，如天气、负荷变化等，建立更为复杂的风险评估模型。

5 附录

5.1 代码实现

5.1.1 风险模型代码

```python
import numpy as np
from scipy.stats import norm
from scipy.integrate import quad
import pandas as pd
import matplotlib.pyplot as plt
import networkx as nx
from typing import List, Dict, Tuple, Optional, Literal, Union


class RiskModel:
    def __init__(
        self,
        nodes_data: pd.DataFrame,
        lines_data: pd.DataFrame,
        substation_data: Dict[str, int] = {},
        tie_switches: Dict[str, Tuple[int, int]] = {},
        dg_data: List[Dict[Literal["node", "capacity"], Union[int, float]]] = [],
        node_type: Dict[
            int, Literal["Residential", "Commercial", "Governmental",
            ↪  "Industrial"]
        ] = {},
        feeder_capacity: float = 2200,
        feeder_current_limit: float = 220,
        voltage: float = 10,
        dg_failure_rate: float = 0.005,
        load_failure_rate: float = 0.005,
        switch_failure_rate: float = 0.002,
```

```
        line_failure_rate_per_km: float = 0.002,
        **kwargs,
    ):
        """
        初始化风险模型

        :param nodes_data: 包含节点信息的 DataFrame
        :param lines_data: 包含线路拓扑信息的 DataFrame
        :param substation_data: 字典，包含变电站及其对应的节点，作为每个馈线的根节点
        :param tie_switches: 字典，包含联络开关及其对应的节点对
        :param dg_data: 分布式发电机数据，包含节点和容量
        :param feeder_capacity: 每个馈线的容量
        :param feeder_current_limit: 每个馈线的电流限制
        :param voltage: 系统电压，单位为 kV
        :param dg_failure_rate: 分布式发电机的故障率
        :param load_failure_rate: 负荷的故障率
        :param switch_failure_rate: 开关的故障率
        :param line_failure_rate_per_km: 每公里线路的故障率
        """
        self.node_weight = {
            "Residential": 0.2,
            "Commercial": 0.5,
            "Governmental": 0.8,
            "Industrial": 1.0,
        }
        self.node_type = node_type

        self.nodes_data = nodes_data
        self.lines_data = lines_data

        self.node_loads: Dict[int, float] = {}
        for i, row in nodes_data.iterrows():
            self.node_loads[i + 1] = row["有功P/kW"]

        self.dg_data = dg_data

        # 系统参数
        self.feeder_capacity = feeder_capacity
        self.feeder_current_limit = feeder_current_limit
        self.voltage = voltage

        # 故障率
        self.dg_failure_rate = dg_failure_rate
```

```
        self.load_failure_rate = load_failure_rate
        self.switch_failure_rate = switch_failure_rate
        self.line_failure_rate_per_km = line_failure_rate_per_km

        # 系统拓扑信息
        self.substation_data = substation_data
        self.tie_switches = tie_switches

        # 建图
        self.network = self._build_network()

        # 预处理馈线信息
        self.feeder_regions: Dict[str, List[int]] = {}
        self.feeder_table: Dict[int, str] = {}
        for substation, node in substation_data.items():
            dfs_result = sorted(list(nx.dfs_postorder_nodes(self.network, node)))
            self.feeder_regions[substation] = dfs_result
            for node in dfs_result:
                self.feeder_table[node] = substation

    def _build_network(self) -> nx.Graph:
        """
        构建网络拓扑图

        :return: 网络拓扑图
        """
        network = nx.Graph()
        for i in range(1, len(self.nodes_data) + 1):
            network.add_node(
                i,
                load=self.nodes_data.loc[i - 1, "有功P/kW"],
                is_dg=i in [dg["node"] for dg in self.dg_data],
            )

        # 添加边
        for _, row in self.lines_data.iterrows():
            start = int(row["起点"])
            end = int(row["终点"])
            length = float(row["长度/km"])
            resistance = float(row["电阻/Ω"])
            reactance = float(row["电抗/Ω"])

            network.add_edge(
```

```
                start,
                end,
                length=length,
                resistance=resistance,
                reactance=reactance,
                failure_rate=length * self.line_failure_rate_per_km,
            )

        return network

    def _calculate_remaining_capacity(self, feeder: str) -> float:
        """
        计算馈线剩余容量

        :param feeder: 馈线名称
        :return: 剩余容量
        """
        nodes = self.feeder_regions[feeder]
        total_load = sum(self.node_loads[node] for node in nodes)

        # 计算分布式发电机的贡献
        dg_contribution = sum(
            dg["capacity"] for dg in self.dg_data if dg["node"] in nodes
        )

        net_load = max(0, total_load - dg_contribution)

        return self.feeder_capacity - net_load

    def _calculate_node_failure_load_loss(self, node: int, is_dg: bool) -> float:
        """
        计算节点故障导致的负荷损失

        :param node: 节点编号
        :return: 负荷损失
        """
        # 如果是 dg 直接相连的节点，则认为本身不会发生故障，发生故障一定是 dg 故障
        if is_dg:
            return next(
                (dg["capacity"] for dg in self.dg_data if dg["node"] == node), 0
            )

        # 否则考虑节点故障导致的损失
```

```
156             loss = max(
157                 self.node_loads[node]
158                 - self._calculate_remaining_capacity(self.feeder_table[node]),
159                 0,
160             )
161 
162             return loss * self.node_weight[self.node_type[node]]
163 
164         def _calculate_line_failure_load_loss(self, line: Tuple[int, int]) -> float:
165             """
166             计算线路故障导致的负荷损失
167 
168             :param line: 线路两端节点的元组
169             :return: 负荷损失
170             """
171             if self.feeder_table[line[0]] != self.feeder_table[line[1]]:
172                 return 0
173 
174             feeder = self.feeder_table[line[0]]
175 
176             temp_network = self.network.copy()
177             temp_network.remove_edge(*line)
178 
179             # 计算断开后，馈线中哪些节点无法到达根节点
180             affected_nodes = set()
181             root_node = self.substation_data[feeder]
182             for node in self.feeder_regions[feeder]:
183                 if node != root_node and not nx.has_path(temp_network, root_node,
                    ↪  node):
184                     affected_nodes.add(node)
185 
186             # 计算负荷损失
187             total_loss = sum(self.node_loads[node] for node in affected_nodes)
188 
189             # 减去断开后的子图中分布式发电机的贡献
190             for dg in self.dg_data:
191                 if dg["node"] in affected_nodes:
192                     total_loss -= min(dg["capacity"], total_loss)
193 
194             # 计算联络线从其他馈线中转移的负荷
195             for _, (node1, node2) in self.tie_switches.items():
196                 other_feeder = None
197                 if node1 in affected_nodes and node2 not in affected_nodes:
```

```
                other_feeder = self.feeder_table[node2]
            elif node2 in affected_nodes and node1 not in affected_nodes:
                other_feeder = self.feeder_table[node1]

            if other_feeder and other_feeder != feeder:
                other_remaining_capacity = self._calculate_remaining_capacity(
                    other_feeder
                )
                total_loss -= min(other_remaining_capacity, total_loss)

        # 计算平均权重，简化分析
        mean_weight = sum(
            self.node_weight[self.node_type[node]] for node in affected_nodes
        ) / len(affected_nodes)

        return total_loss * mean_weight

    def _calculate_switch_failure_load_loss(self, switch: Tuple[int, int]) ->
    ↪  float:
        """
        计算开关故障导致的负荷损失

        :param switch: 开关两端节点的元组
        :return: 负荷损失
        """
        # 简化处理，认为开关故障不导致负荷损失
        return 0

    def calculate_load_loss_risk(self) -> float:
        """
        计算负荷损失风险

        :return: 负荷损失风险
        """
        total_risk = 0.0

        # 分三部分计算风险
        # 然后累加到总风险中

        for node, data in self.network.nodes(data=True):
            if data["is_dg"]:
                failure_rate = self.dg_failure_rate
            else:
```

```
                failure_rate = self.load_failure_rate

            load_loss = self._calculate_node_failure_load_loss(node,
            ↪  data["is_dg"])

            total_risk += load_loss * failure_rate

        for u, v, data in self.network.edges(data=True):
            line_failure_rate = data["failure_rate"]
            load_loss = self._calculate_line_failure_load_loss((u, v))
            total_risk += load_loss * line_failure_rate

        for switch in self.tie_switches.values():
            failure_rate = self.switch_failure_rate
            load_loss = self._calculate_switch_failure_load_loss(switch)
            total_risk += load_loss * failure_rate

        return total_risk

    def calculate_overload_risk(self) -> float:
        """
        计算过载风险

        :return: 过载风险
        """
        # TODO: 分用户类型计算损失
        # TODO: 计算原件故障导致线路断路无法均衡负载导致的过负荷风险
        total_risk = 0.0
        for feeder in self.feeder_regions.keys():
            feeder_risk = self._calculate_feeder_overload_risk(feeder)
            total_risk += feeder_risk

        return max(0.1, total_risk)

    def _calculate_feeder_current(self, feeder: str) -> float:
        """
        计算馈线电流

        :param feeder: 馈线名称
        :return: 馈线电流
        """
        nodes = self.feeder_regions[feeder]

```

```
282         total_load = sum(self.node_loads[node] for node in nodes)
283         dg_capacity = sum(dg["capacity"] for dg in self.dg_data if dg["node"] in
            ↪  nodes)
284
285         net_load = max(0, total_load - dg_capacity)  # 当前网络净负载
286         excess_load = 0  # 额外负载
287
288         if net_load > 0:
289             current = net_load / (np.sqrt(3) * self.voltage)  # 考虑三相电流
290         else:
291             # DG的贡献大于负荷，考虑相邻馈线的负荷转移
292             excess_load = dg_capacity - total_load
293
294             for _, (node1, node2) in self.tie_switches.items():
295                 if node1 in nodes or node2 in nodes:
296                     other_feeder = (
297                         self.feeder_table[node2]
298                         if node1 in nodes
299                         else self.feeder_table[node1]
300                     )
301                     other_remaining_capacity = self._calculate_remaining_capacity(
302                         other_feeder
303                     )
304
305                     transferable_load = min(excess_load, other_remaining_capacity)
306                     excess_load -= transferable_load
307
308             net_load = total_load + max(0, excess_load)
309             current = net_load / (np.sqrt(3) * self.voltage)
310
311         return current
312
313     def _calculate_feeder_overload_risk(self, feeder: str) -> float:
314         """
315         计算馈线过载风险
316
317         :param feeder: 馈线名称
318         :return: 馈线过载风险
319         """
320         total_risk = 0.0
321         current = self._calculate_feeder_current(feeder)
322
323         # TODO: 认为馈线电流为正态分布
```

```
            mu = current
            sigma = 0.1 * current  # 假设标准差为平均值的10%
            mean_weight = sum(
                self.node_weight[self.node_type[node]]
                for node in self.feeder_regions[feeder]
            ) / len(self.feeder_regions[feeder])

            def overload_consequence(x: float) -> float:
                if x > self.feeder_current_limit * 1.1:
                    return (
                        2
                        * np.sqrt(3)
                        * self.voltage
                        * (x - self.feeder_current_limit * 1.1 + 0.1)
                        * mean_weight
                    )
                else:
                    return (
                        0.2
                        * np.sqrt(3)
                        * self.voltage
                        * (x / (self.feeder_current_limit * 1.1))
                        * mean_weight
                    )

            total_risk = quad(
                lambda x: overload_consequence(x) * norm.pdf(x, mu, sigma),
                0,
                np.inf,
            )[0]

            return total_risk

        def calculate_system_risk(self) -> Tuple[float, float, float]:
            """
            计算系统风险

            :return: 系统风险、负荷损失风险、过载风险
            """
            load_loss_risk = self.calculate_load_loss_risk()
            overload_risk = self.calculate_overload_risk()

            # 计算系统风险
```

```
367             system_risk = load_loss_risk + overload_risk
368
369             return system_risk, load_loss_risk, overload_risk
370
371         def draw_network(self, path: Optional[str] = None) -> None:
372             """
373             绘制网络拓扑图
374
375             :param path: 保存路径，默认为 None，表示直接显示图形
376             """
377             # TODO: 分类染色节点
378
379             node_labels = {
380                 node: f"{node}\n{data['load']}kW"
381                 for node, data in self.network.nodes(data=True)
382             }
383             edge_labels = {
384                 (u, v): f"{data['length']:.2f} km"
385                 for u, v, data in self.network.edges(data=True)
386             }
387
388             for u, v in self.network.edges():
389                 self.network.edges[u, v]["weight"] = 1 / len(edge_labels[(u, v)])
390
391             plt.figure(figsize=(14, 10))
392
393             pos = nx.kamada_kawai_layout(self.network, weight="weight", scale=10)
394             nx.draw(
395                 self.network,
396                 pos,
397                 node_size=700,
398                 node_color="lightblue",
399             )
400
401             nx.draw_networkx_labels(
402                 self.network,
403                 pos,
404                 labels=node_labels,
405                 font_size=8,
406             )
407             nx.draw_networkx_edge_labels(
408                 self.network,
409                 pos,
```

```
            edge_labels=edge_labels,
            font_size=6,
        )

        plt.title("Network Topology")
        plt.axis("off")

        if path:
            plt.savefig(path)
        else:
            plt.show()


def get_default_model() -> RiskModel:
    node_data = pd.read_excel("appendix.xlsx", sheet_name=0)
    line_data = pd.read_excel("appendix.xlsx", sheet_name=1)

    dg_data = [
        {"node": 13, "capacity": 300},
        {"node": 18, "capacity": 300},
        {"node": 22, "capacity": 300},
        {"node": 29, "capacity": 300},
        {"node": 32, "capacity": 300},
        {"node": 39, "capacity": 300},
        {"node": 48, "capacity": 300},
        {"node": 59, "capacity": 300},
    ]

    node_type = {
        1: "Residential",
        2: "Residential",
        3: "Residential",
        4: "Residential",
        5: "Residential",
        6: "Residential",
        7: "Industrial",
        8: "Residential",
        9: "Governmental",
        10: "Residential",
        11: "Commercial",
        12: "Industrial",
        13: "Residential",
        14: "Industrial",
```

```
        15: "Residential",
        16: "Commercial",
        17: "Residential",
        18: "Industrial",
        19: "Residential",
        20: "Residential",
        21: "Governmental",
        22: "Residential",
        23: "Residential",
        24: "Residential",
        25: "Residential",
        26: "Residential",
        27: "Commercial",
        28: "Residential",
        29: "Governmental",
        30: "Residential",
        31: "Commercial",
        32: "Industrial",
        33: "Commercial",
        34: "Commercial",
        35: "Residential",
        36: "Governmental",
        37: "Residential",
        38: "Commercial",
        39: "Residential",
        40: "Industrial",
        41: "Residential",
        42: "Commercial",
        43: "Residential",
        44: "Residential",
        45: "Governmental",
        46: "Residential",
        47: "Industrial",
        48: "Residential",
        49: "Commercial",
        50: "Residential",
        51: "Residential",
        52: "Residential",
        53: "Commercial",
        54: "Residential",
        55: "Residential",
        56: "Commercial",
        57: "Residential",
```

```
            58: "Governmental",
            59: "Residential",
            60: "Residential",
            61: "Governmental",
            62: "Residential",
        }

        tie_switches = {
            "S13-1": (13, 23),
            "S29-2": (29, 43),
            "S62-3": (62, 1),
        }

        substation_switches = {
            "CB1": 1,
            "CB2": 23,
            "CB3": 43,
        }

        model = RiskModel(
            nodes_data=node_data,
            lines_data=line_data,
            substation_data=substation_switches,
            tie_switches=tie_switches,
            dg_data=dg_data,
            node_type=node_type,
        )

        return model
```

代码 1: 风险模型代码

```
from typing import IO, Optional
from risk_model import get_default_model

def analyze_system_risk(
    topology_graph: Optional[str] = None, log_out: Optional[IO[str]] = None
) -> None:
    """
    分析系统风险
    """
    model = get_default_model()
    system_risk, load_loss_risk, overload_risk = model.calculate_system_risk()
    risk_level = "高" if system_risk > 30 else "中" if system_risk > 15 else "低"
```

```
    print(f"====== 系统风险分析 ======", file=log_out)
    print(f"系统风险: {system_risk}", file=log_out)
    print(f"失负荷风险: {load_loss_risk}", file=log_out)
    print(f"过负荷风险: {overload_risk}", file=log_out)
    print(f"====== 风险占比 ======", file=log_out)
    print(f"失负荷风险占比: {load_loss_risk / system_risk:.2%}", file=log_out)
    print(f"过负荷风险占比: {overload_risk / system_risk:.2%}", file=log_out)
    print(f"风险等级: {risk_level}", file=log_out)
    print(f"====== 网络拓扑图 ======", file=log_out)
    print(f"输出到: {topology_graph}", file=log_out)
    model.draw_network(topology_graph)


if __name__ == "__main__":
    analyze_system_risk()
```

代码 2: 问题 1 风险模型代码

5.1.2 分布式能源容量趋势代码

```
import numpy as np
import matplotlib.pyplot as plt
from typing import Optional, IO, Tuple
from risk_model import get_default_model


def analyze_dg_capacity(
    path: Optional[str] = None,
    output: Optional[IO[str]] = None,
    capacity_range: Tuple[float, float, float] = (1.0, 3.1, 0.3),
) -> None:
    """
    分析分布式发电容量对系统风险的影响

    :param path: 可选的路径参数，默认为None
    """
    model = get_default_model()

    dg_capacity_factors = np.arange(*capacity_range)

    results = {
        "capacity_factor": [],
        "total_dg_capacity": [],
        "load_loss_risk": [],
```

```
        "overload_risk": [],
        "system_risk": [],
    }

    base_dg_data = model.dg_data.copy()

    print("====== 分布式发电容量对系统风险的影响 ======", file=output)

    for factor in dg_capacity_factors:
        print(f"计算容量因子: {factor:.2f}", file=output)

        # 更新DG容量
        model.dg_data = list(
            map(
                lambda dg: {
                    "node": dg["node"],
                    "capacity": dg["capacity"] * factor,
                },
                base_dg_data,
            )
        )

        # 计算系统风险
        system_risk, load_loss_risk, overload_risk = model.calculate_system_risk()

        # 记录结果
        results["capacity_factor"].append(factor)
        results["total_dg_capacity"].append(sum(dg["capacity"] for dg in
        ↪  model.dg_data))
        results["load_loss_risk"].append(load_loss_risk)
        results["overload_risk"].append(overload_risk)
        results["system_risk"].append(system_risk)

        print(f"  失负荷风险: {load_loss_risk:.2f}", file=output)
        print(f"  过负荷风险: {overload_risk:.2f}", file=output)
        print(f"  系统总风险: {system_risk:.2f}", file=output)

    # 绘制图像
    plt.rcParams["font.sans-serif"] = ["SimHei"]  # 设置中文字体
    plt.rcParams["axes.unicode_minus"] = False  # 显示负号
    plt.figure(figsize=(10, 6))
    plt.plot(
        results["capacity_factor"],
```

```
        results["load_loss_risk"],
        label="失负荷风险",
        marker="o",
    )
    plt.plot(
        results["capacity_factor"],
        results["overload_risk"],
        label="过负荷风险",
        marker="o",
    )
    plt.plot(
        results["capacity_factor"],
        results["system_risk"],
        label="系统总风险",
        marker="o",
    )
    plt.title("分布式发电容量对系统风险的影响")
    plt.xlabel("DG容量因子")
    plt.ylabel("风险值")

    if path:
        plt.savefig(path)

    print("保存图像到:", path, file=output)

    print("====== 馈线风险分析 ======", file=output)

    total_risk = (
        model.network.number_of_nodes() * model.load_failure_rate
        + model.network.size("failure_rate")
    )
    risk_ratios = [
        (
            model.network.subgraph(nodes).number_of_nodes() *
            ↪  model.load_failure_rate
            + model.network.subgraph(nodes).size("failure_rate")
        )
        / total_risk
        for _, nodes in model.feeder_regions.items()
    ]
    feeder_results = {
        feeder_name: [risk * ratio for risk in results["system_risk"]]
        for feeder_name, ratio in zip(model.feeder_regions.keys(), risk_ratios)
```

```
    }

    for idx, feeder_name in enumerate(model.feeder_regions.keys()):
        print(f"馈线 {feeder_name} 风险占比: {risk_ratios[idx]:.2%}", file=output)

    # 绘制馈线风险图
    plt.figure(figsize=(10, 6))
    for feeder_name, risks in feeder_results.items():
        plt.plot(
            results["capacity_factor"],
            risks,
            label=f"{feeder_name}风险",
            marker="o",
        )
    plt.title("馈线风险随DG容量因子变化")
    plt.xlabel("DG容量因子")
    plt.ylabel("风险值")
    plt.legend()
    if path:
        plt.savefig(path.replace(".png", "_feeder.png"))
    print(f"保存馈线风险图像到保存馈线风险图像到: {path.replace(".png",
    ↪  "_feeder.png")}", file=output)

    print(f"====== 分析结论 ======", file=output)
    system_risks = zip(
        dg_capacity_factors,
        results["system_risk"],
        results["load_loss_risk"],
        results["overload_risk"],
    )
    min_risk = min(system_risks, key=lambda x: x[1])
    print(f"最小系统风险发生在容量因子: {min_risk[0]:.2f}", file=output)
    print(f"最小系统风险: {min_risk[1]:.2f}", file=output)
    print(f"  对应失负荷风险: {min_risk[2]:.2f}", file=output)
    print(f"  对应过负荷风险: {min_risk[3]:.2f}", file=output)

    if not path:
        plt.show()


if __name__ == "__main__":
    analyze_dg_capacity()
```

代码 3: 分布式能源容量趋势代码

5.1.3 其余工具代码

```python
from Q1_risk_model import analyze_system_risk
from Q2_dg_capacity_trending import analyze_dg_capacity
import os

if __name__ == "__main__":
    os.makedirs("problem1", exist_ok=True)
    os.makedirs("problem2", exist_ok=True)

    topology_graph = "problem1/topology_graph.png"
    system_risk_log = "problem1/system_risk_log.txt"
    analyze_system_risk(
        topology_graph=topology_graph,
        log_out=open(system_risk_log, "w", encoding="utf-8"),
    )
    print(f"系统风险分析结果已保存到: {system_risk_log}")

    trending_graph = "problem2/dg_capacity_trending.png"
    dg_capacity_log = "problem2/dg_capacity_log.txt"
    analyze_dg_capacity(
        path=trending_graph,
        output=open(dg_capacity_log, "w", encoding="utf-8"),
    )
    print(f"分布式发电容量对系统风险的影响分析结果已保存到: {dg_capacity_log}")
```

代码 4: 运行所有代码