浙江大学第二十三届
大学生数学建模竞赛

2025 年 5 月 9 日—5 月 19 日

团队编号 8493

题　　目　　A √　　B

（在所选题目上打勾）

	参赛队员 1	参赛队员 2	参赛队员 3
姓名	袁明旭	杨佳昕	李一鸣
学号	3220105107	3220104688	3220104687
院(系)	控制科学与工程学院	控制科学与工程学院	控制科学与工程学院
专业	自动化（控制）	自动化（控制）	自动化（控制）
手机	15244634920	17300982876	17604215558
Email	3220105107@zju.edu.cn	3220104688@zju.edu.cn	3220104687@zju.edu.cn

浙江大学本科生院

浙江大学数学建模实践基地

分布式能源接入配电网的风险分析

摘要

随着“双碳”目标的推进，分布式能源（DG）在配电网中的使用率显著提升，但其随机性和波动性加剧了配电网的运行风险。配电网在故障或 DG 出力波动时，可能因失负荷或过负荷引发经济损失。本研究针对含 DG 的有源配电网，构建失负荷与过负荷风险量化模型，分析 DG 容量变化对系统风险的影响，为配电网规划与运行提供理论支持。

对于问题一，需要分别建立失负荷和过负荷的风险模型。失负荷风险模型基于蒙特卡洛模拟与网络拓扑重构技术。首先，根据元件故障率随机生成故障场景，若存在故障元件，则闭合联络开关重构拓扑。移除故障元件后，通过广度优先搜索（BFS）模拟功率分配过程。重新分配功率以转移负荷。失负荷危害度通过用户类型的差异化权重计算，结合故障概率与危害度函数，最终得到假设场景下失负荷发生概率为 42.76%，失负荷风险值为 216.34。

过负荷风险模型以 DG 出力波动为核心，采用潮流分析与功率转移策略。首先，DG 出力假设为正态分布，计算各馈线净功率，通过交流潮流分析计算线路电流。若线路发生过负荷，则通过迭代功率转移算法，将过剩功率经联络线转移至相邻馈线。过负荷危害度基于电流超过额定值 110%的超标量计算，结合故障概率与危害系数，最终量化过负荷风险。最终得到假设场景下过负荷的发生概率和风险值均为 0。

对于问题二，针对 62 节点有源配电网，采用问题一所建模型，通过蒙特卡洛模拟分析 DG 容量从初始值 I（300kW）以 0.3I 步长增至 3I 时的系统风险演变。结果显示：随着 DG 容量增加，失负荷发生概率与危害度呈下降趋势，概率从 42.63%下降至 41.42%，危害度从 217.35 下降至 164.09，而过负荷概率和危害度始终保持几乎为 0。系统总风险呈现下降趋势。这一结果为 DG 容量配置提供了理论依据。

关键词：分布式能源 失负荷 过负荷 蒙特卡洛 潮流分析 广度优先搜索

1. 问题重述

1.1 问题背景

随着我国双碳目标的推进，可再生分布式能源（DG）在配电网中的大规模应用不可避免，这对传统配电网的运行可靠性提出严峻挑战。配电网在运行过程中，可能发生失负荷或过负荷的问题：失负荷代表配电网因故障导致负荷供电中断；过负荷表示线路电流超过额定载流量的10%以上。DG接入配电网后，由于其发电出力的波动性与不确定性，对馈线的失负荷或过负荷会带来影响。因此，需要对DG 接入配电网进行风险分析，建立数学模型，计算配电网发生失负荷和过负荷的概率与危害程度，量化评估系统整体风险。

1.2 要解决的问题

本研究针对以下核心问题展开：

（1）建立分布式能源接入后配电网的失负荷风险与过负荷风险模型。其中，失负荷风险需结合联络线转供能力，量化故障后通过相邻馈线复电减少的负荷损失；过负荷风险需要考虑DG出力波动对馈线功率分布的影响，并约束其不得向上级电网倒送功率，仅允许通过联络线在馈线间调整功率；

（2）以62节点有源配电网为例，利用上一问题建立的风险模型，分析DG容量从初始值I以0.3I为步长增至3I时，系统总风险R_{sys}的演变规律。研究通过融合故障概率和危害度函数，为DG容量配置和配电网风险评估提供理论依据。

2. 问题分析

对于**问题一**：题目要求分别建立 DG 接入配电网后，配电系统的失负荷和过负荷风险的计算模型。由于配电网系统较为复杂，并且系统发生失负荷或过负荷是概率问题，因此可考虑采用蒙特卡洛算法，通过多次随机模拟电网状态，计算失负荷和过负荷的发生概率和危害程度。下面分析单次模拟过程中，两种故障计算模型的建立方式：

失负荷风险模型：首先建立原始的配电网拓扑网络，根据各分布式能源、用户、开关、配电线路的故障率，随机模拟配电网故障情况，移除网络中故障的节点或边，生成新的网络结构。若存在故障的节点或边，则打开馈线间的联络开关。随后，利用搜索算法，将供电节点（DG 和变电站出线开关）的功率向下级负载分配。若分配完成后，存在未能复电的负载，则记录失电负载的种类和失负荷量，计算危害程度。

过负荷风险模型：首先模拟分布式能源的波动情况，取固定均值，随机生成 DG 的出力。随后，从每条馈线的出线开关 CB 开始，搜索连接到这条馈线上的负载和 DG，计算馈线净功率。接下来，利用潮流分析公式，计算馈线上的电流，并与额定电流的 110%作比较，若超过，则打开相邻的联络开关转移部分功率。功率转移后，若馈线电流仍高于额定电流的 110%，则视为过负荷发生，最后计算馈线过负荷的危害程度。

对于**问题二**：题目要求针对有源配电网 62 节点系统图，DG 容量以 0.3I 为步长，分析其从初始容量 I 增加到 3I 的过程中，配电系统风险的演变情况。首先利用 python 的 NetworkX 库绘制题中所述 62 节点系统图，设置 DG 容量为 I，利用问题一的模型，通过蒙特卡洛模拟，计算失负荷与过负荷的危害程度，并相加得到系统总风险。随后，以 0.3I 为步长逐步增加 DG 的容量，重复上述过程，可以得到不同 DG 容量下R_{sys}的数据。最后，绘制 DG 容量与R_{sys}的关系曲线，分析其演变情况。

3. 模型假设

1. 每个 DG 和每个用户的故障率均为 0.5%，每个开关的故障率均为 0.2%，配电线路的故障率=线路长度（km）*0.002/km。每种类型（DG、用户、开关、线路）的故障是独立发生的，并且同一时间同一类型只发生一个故障；
2. 配电网不考虑无功功率和电压越限的影响，风险计算分析仅考虑有功功率和电流的影响；
3. 联络开关不考虑故障恢复的自愈系统对失负荷的影响，但是需考虑联络开关的负荷转移能力，设定所有联络开关的负荷转移能力均为 1.5MW；
4. DG 的出力遵循正态分布的规律，蒙特卡洛算法的单次模拟中根据正态分布的概率密度函数随机生成 DG 的出力。

4. 符号说明

符号	说明	单位
R_{sys}	系统总风险	/
P_{LL}	系统失负荷发生概率	/
P_{OL}	系统过负荷发生概率	/
C_{LL}	系统失负荷危害程度	/
C_{OL}	系统过负荷危害程度	/
ω_j	用户j失负荷的危害度系数	kW^{-1}
c_k	馈线k过负荷的危害度系数	A^{-1}
$I_{threshold}$	馈线的额定电流	A
λ_i	元件i的故障概率	/
D_j	节点j的负荷需求	kW
$T_{i,j}$	馈线i,j间联络开关的转移限制	kW

5. 模型建立与求解

5.1 问题一模型建立

5.1.1 建模准备

为量化分布式能源接入配电网后失负荷与过负荷的风险，本研究基于蒙特卡洛模拟和潮流分析的方式进行建模。对于问题一，模型以配电网的拓扑结构、负荷分布、DG 出力以及联络开关状态为输入，以系统风险值 R_{sys} 为输出，R_{sys} 的计算公式为

$$R_{sys} = P_{LL}C_{LL} + P_{OL}C_{OL}$$

其中 P_{LL} 和 C_{LL} 分别为失负荷的发生概率和危害程度，P_{OL} 和 C_{OL} 分别为过负荷的发生概率和危害程度。

为实现蒙特卡洛方法，首先需要计算故障模拟的有关概率，从而生成故障模拟的配电网络结构。根据假设，每类元件故障独立发生，并且同一时间同一类型只发生一个故障。因此，同一类型的元件之间，发生故障的事件是互斥事件，而不同类型元件之间的故障互不影响，所以有公式：

$$p_J = \Sigma_{i \in J} \lambda_i$$

其中，J 为元件类型，λ_i 为元件 i 故障的概率，p_J 为类型 J 的元件故障的总概率。根据上述公式，可分别获得网络中用户、开关、线路、DG 等元素的故障率，从而模拟配电网元件故障的情况，生成故障模拟的配电网络。接下来的失负荷与过负荷的计算模型，将基于这一故障模拟的配电网络建立。

5.1.2 失负荷模型建立

首先检测配电网中是否有故障发生，若有故障，闭合所有联络开关。将变电站出线开关 CB 视为容量为线路额定功率的电源，从 CB 和 DG 节点开始，利用广度优先搜索算法（BFS）依次搜索可访问的节点，并为各个节点分配功率，分配时考虑联络开关功率转移的限制，经过联络开关的功率最大值为 $T_{limit,i,j}$。在第 i 次

模拟中，搜索完成后，将未复电的节点集合记为L_i，失负荷量可计算为：

$$L_{LL,i}=\Sigma_{j\in L_i}D_j$$

其中$L_{LL,i}$为第i次模拟时的总失负荷量，D_j为节点j的负荷需求。失负荷概率计算为：

$$P_{LL}=\frac{1}{N}\Sigma_{i=1}^{N}I(L_{LL,i}>0)$$

其中N为模拟次数，$I()$为指示函数，当括号内表达式成立时取1，否则为0。危害程度C_{LL}取决于不同种类用户的权重，单次模拟的危害程度计算表达式为：

$$C_{LL,i}=\Sigma_{j\in L_i}\omega_j\cdot D_j$$

其中ω_j为节点j用户的危害度系数。若第i次模拟系统未出现失负荷问题，$C_{LL,i}=0$，否则，$C_{LL,i}$为正值。因此，单次模拟的系统失负荷风险表达式为

$$R_{LL,i}=P_{LL,i}C_{LL,i}=C_{LL,i}$$

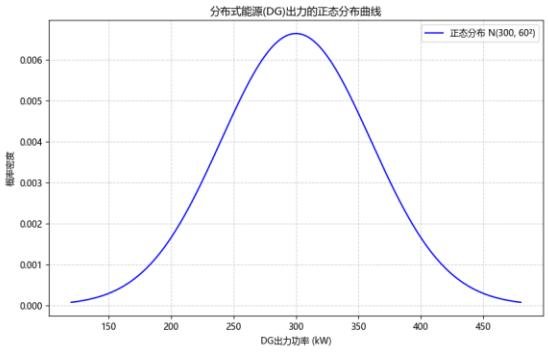
根据蒙特卡洛算法的原理，失负荷导致的系统风险值的最终表达式为

$$R_{LL}=\frac{1}{N}\Sigma_{i=1}^{N}R_{LL,i}=\frac{1}{N}\Sigma_{i=1}^{N}C_{LL,i}$$

5.1.3 过负荷模型建立

系统发生过负荷的主要原因在于DG出力的波动，因此为评估系统过负荷风险，此处设DG的出力为正态分布，取均值为I，方差为$0.2I$，满足表达式

$P_DG\sim N(\mu=I,\sigma^2=(0.2I)^2)$，DG出力的概率密度图像如下：



为进行系统的潮流分析。首先需要计算馈线净功率。馈线i净功率的计算表达式为：

$$P_{net,i} = max(0, P_{load,i} - P_{dg,i})$$

其中，$P_{net,i}$为馈线i的净功率，$P_{load,i}$为馈线i上的负载功率之和，$P_{dg,i}$为馈线i的DG出力之和。为防止向主网倒送功率，限制净功率的最小值为0。当净功率为0时，DG出力大于负载总功率，此时则需要进行功率转移。设馈线i与j之间联络开关功率转移的限额为$T_{limit,i,j}$，馈线j未达到过负荷条件下的可用容量C_j，功率转移量取联络开关功率转移限额、相邻馈线可转移容量和馈线i溢出功率中的的较小者

$$T_{i,j} = min(P_{dj,i} - P_{load,i}, C_j, T_{limit,i,j})$$

转移过后的净功率为：

$$P_{net,i,transferred} = P_{net,i} - T_{i,j}$$

下面利用交流潮流分析公式计算馈线i的电流，计算公式为

$$I_i = \frac{P_{net,i,transferred}}{\sqrt{3}Vcos\phi}$$

其中，V为馈线电压，$cos\phi$为功率因数角，由于题目仅考虑有功功率的影响，故此处取$cos\phi = 1$。额定载流量为$I_{threshold}$，当馈线电流超过额定载流量的110%，视为过负荷发生。

过负荷发生概率计算公式为：

$$P_{OL} = \frac{1}{N}\Sigma_{k=1}^{N} I(I_{k,i} > 1.1 \cdot I_{threshold})$$

其中$I_{k,i}$为第k次模拟中馈线i的电流。单次模拟下，过负荷发生的危害程度计算公式为

$$C_{OL,k} = \begin{cases} \Sigma_{i \in L_k} c_i (I_i - 1.1 \cdot I_{threshold}), I_i - 1.1 \cdot I_{threshold} > 0 \\ 0, else \end{cases}$$

其中c_i为馈线过负荷的危害度系数，L_k为发生过负荷的馈线组成的集合。第k次模拟中，系统风险的表达式为

$$R_{OL,k} = P_{OL,k}C_{OL,k} = C_{OL,k}$$

根据蒙特卡洛算法的原理，过负荷导致的系统风险值的最终表达式为

$$R_{OL} = \frac{1}{N}\Sigma_{k=1}^{N}R_{OL,k} = \frac{1}{N}\Sigma_{k=1}^{N}C_{OL,k}$$

5.1.4 系统总风险计算

根据公式$R_{sys} = P_{LL}C_{LL} + P_{OL}C_{OL} = R_{LL} + R_{OL}$，将失负荷与过负荷计算模型得到的风险值相加，可得到系统总风险R_{sys}.

5.2 问题一模型求解

以问题二的62节点配电网络图为例，进行失负荷、过负荷及系统总风险的计算。配电网网络结构建模方式参考“5.3 问题二模型建立”。配电网参数设置如下：

所有DG容量$P_{dg} = 300kW$，

所有联络开关功率转移限额为$\mathrm{T_{i,j}} = 1.5\mathrm{MW}$，

居民，商业，政府和机构，办公和建筑节点的失负荷危害度系数ω_i均设为$1.0kW^{-1}$，

所有馈线过负荷的危害度系数c_k均设为$1.0A^{-1}$。

应用问题一建立的模型，根据以上参数对配电网进行风险分析，可得到量化的输出结果。

5.3 问题二模型建立

问题二的求解需要应用问题一所建立的模型，而问题一的模型中最为关键的输入为配电网的网络结构。NetworkX是用于复杂网络（图结构）分析的Python

库，支持创建、操作和研究节点-边组成的网络模型。下面将使用 NetworkX 库将配电网抽象为无向图 $G=(V,E)$ 。

无向图 G 的节点 V 包含以下类型：

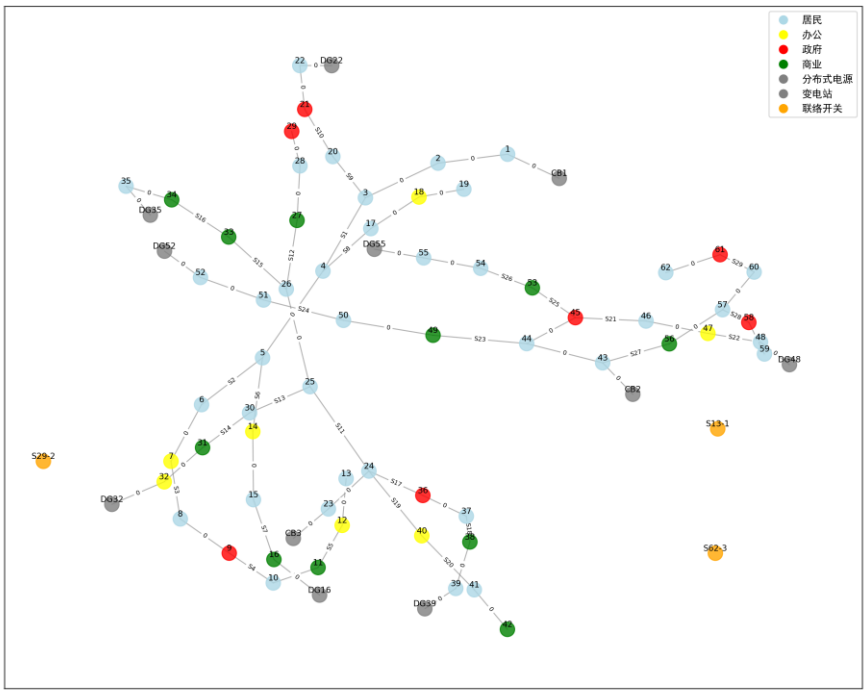
电源节点：与变电站直接相连的开关（CB1，CB2，CB3）与分布式能源 DG，每个节点配有功率属性 p 和分类属性 cat。

用户节点：包括居民（res）、商业（com）、政府和机构（gov）、办公和建筑（off）四类用户，其功率与分类属性源自附件表格。

联络开关节点：包括 S13-1，S29-2，S62-3 三个联络开关，包含功率（p=0）与分类属性。默认情况下不与电路相连接，故障时可闭合以转移电荷。

边集合 E 表示配电线路，每条边具有长度、电阻、电抗以及是否具有开关（若有则为开关名称，否则为 0）属性。

最终输出的网络结构图如下：



得到 62 节点配电网的网络拓扑图后，可以将其作为问题一模型的输入，计算系统的总风险。

5.4 问题二模型求解

按照题目要求，将配电网网络拓扑图中节点 DG 的功率属性 p，从初始的 $I=300kW$ 以 $0.3I$ 为步长增加到 $3I$，运行问题一的模型进行求解。最终可得到 DG 不同容量情况下对应的系统风险数据，绘制折线图，可得到配电系统风险 R_{sys} 的演变情况。

6. 结果分析与检验

6.1 问题一

按照“5.2 问题一模型求解”的说明，进行 1000000 次蒙特卡洛模拟，输出结果如下：

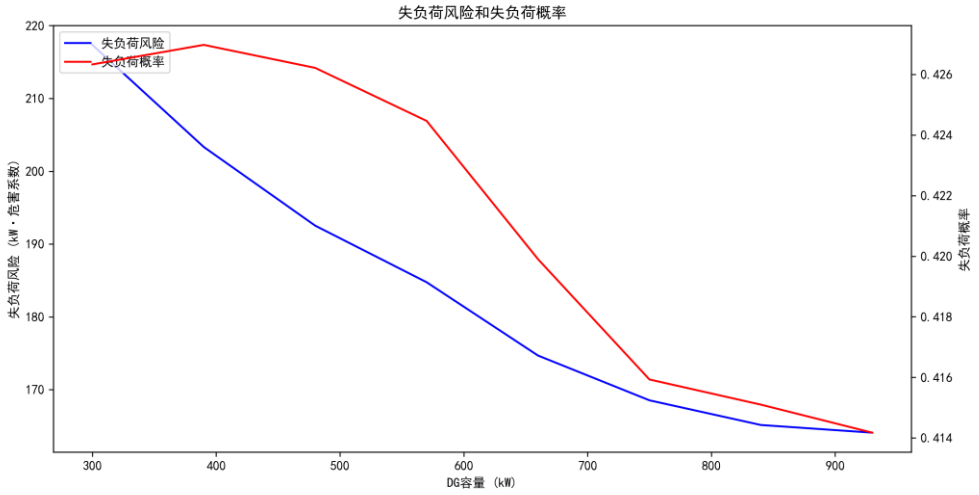
当前 DG 容量：300.0 kW
（进度条略）
系统失负荷风险：216.34 kW • 危害系数
系统过负荷风险：0.00 kW • 危害系数
失负荷概率：0.4276
过负荷概率：0.0000
系统风险：216.3370 kW • 危害系数

（其中“危害系数”为单位 kW^{-1}，下同）

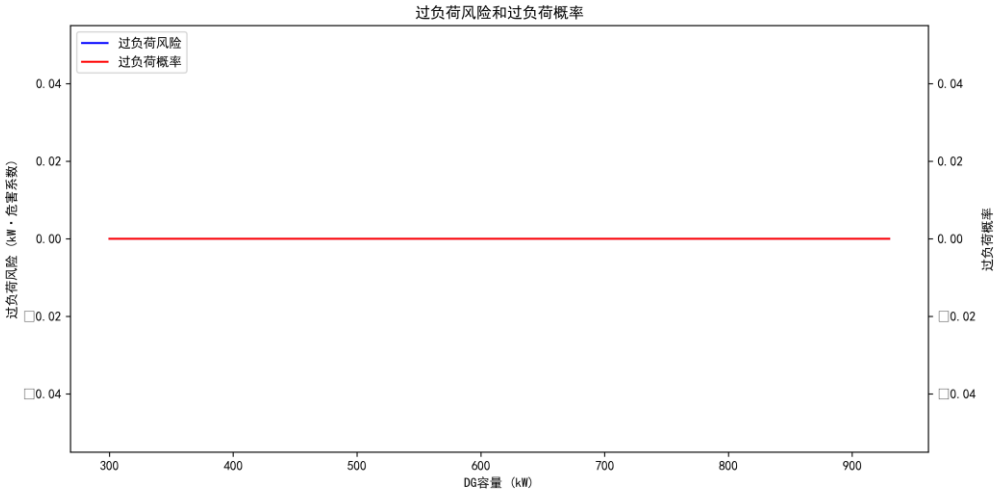
6.2 问题二

当蒙特卡洛模拟次数为 200000 次时，作失负荷风险、失负荷概率-DG 容量，过负荷风险、过负荷概率-DG 容量，以及系统总风险-DG 容量的图像：

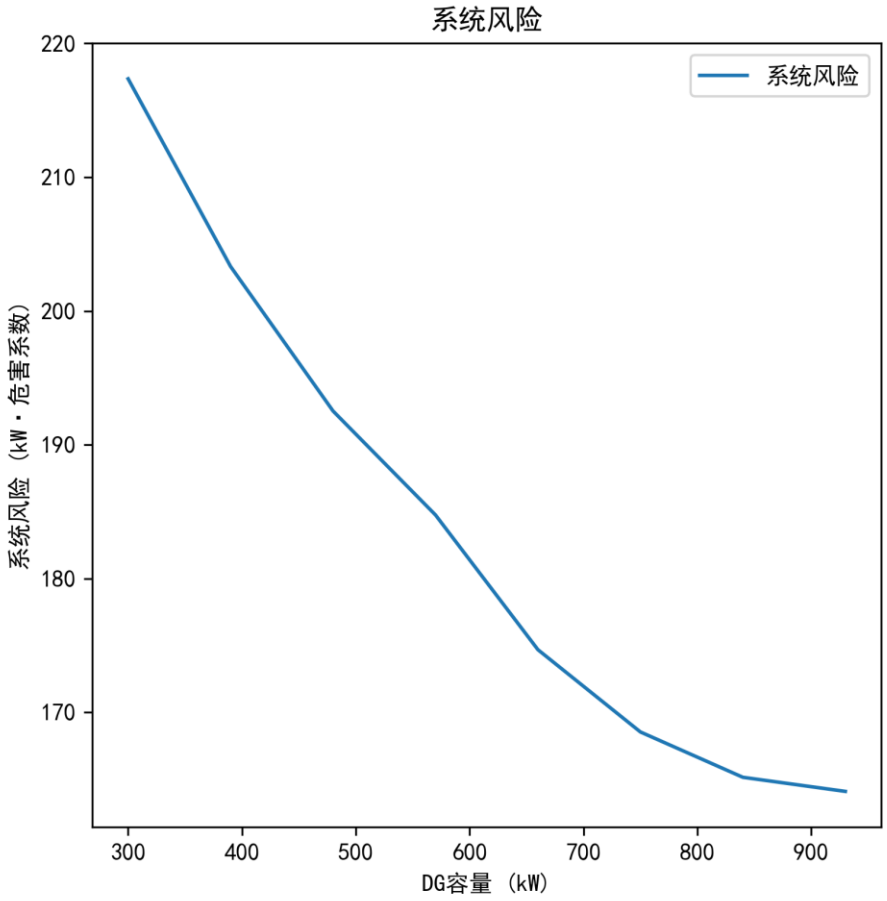
（1）失负荷风险、失负荷概率-DG 容量图像：



（2）过负荷风险、过负荷概率-DG 容量图像：



（3）系统风险-DG 容量图像：



代码运行结果：

```
当前 DG 容量: 300.0 kW
（进度条略）
系统失负荷风险: 217.35 kW·危害系数
系统过负荷风险: 0.00 kW·危害系数
失负荷概率: 0.4263
过负荷概率: 0.0000
系统风险：217.3532 kW·危害系数
当前 DG 容量: 390.0 kW
（进度条略）
系统失负荷风险: 203.33 kW·危害系数
系统过负荷风险: 0.00 kW·危害系数
失负荷概率: 0.4270
过负荷概率: 0.0000
系统风险：203.3252 kW·危害系数
当前 DG 容量: 480.0 kW
（进度条略）
系统失负荷风险: 192.52 kW·危害系数
系统过负荷风险: 0.00 kW·危害系数
失负荷概率: 0.4262
过负荷概率: 0.0000
系统风险：192.5223 kW·危害系数
当前 DG 容量: 570.0 kW
（进度条略）
系统失负荷风险: 184.75 kW·危害系数
系统过负荷风险: 0.00 kW·危害系数
失负荷概率: 0.4245
```

过负荷概率: 0.0000

系统风险：184.7521 kW·危害系数

当前 DG 容量: 660.0 kW

（进度条略）

系统失负荷风险: 174.68 kW·危害系数

系统过负荷风险: 0.00 kW·危害系数

失负荷概率: 0.4199

过负荷概率: 0.0000

系统风险：174.6790 kW·危害系数

当前 DG 容量: 750.0 kW

（进度条略）

系统失负荷风险: 168.53 kW·危害系数

系统过负荷风险: 0.00 kW·危害系数

失负荷概率: 0.4159

过负荷概率: 0.0000

系统风险：168.5324 kW·危害系数

当前 DG 容量: 840.0 kW

（进度条略）

系统失负荷风险: 165.14 kW·危害系数

系统过负荷风险: 0.00 kW·危害系数

失负荷概率: 0.4151

过负荷概率: 0.0000

系统风险：165.1443 kW·危害系数

当前 DG 容量: 930.0 kW

（进度条略）

系统失负荷风险: 164.09 kW·危害系数

系统过负荷风险: 0.00 kW·危害系数

失负荷概率: 0.4142

过负荷概率: 0.0000

系统风险：164.0903 kW·危害系数

输出结果表明，在 DG 容量从 300kW 增加到 900kW 的过程中，系统的失负荷的风险与概率均有所下降，过负荷的风险与概率始终保持几乎为 0，进而导致系统风险下降。此外，由于系统发生过负荷的概率几乎为 0，系统风险由失负荷主导。

在实际应用的条件下，系统风险演变情况可能并不与 DG 容量成线性关系，本研究讨论的情况基于 DG 出力为正态分布的假设，而实际情况的 DG 出力常常波动较大，且不成规律，因此系统风险变化规律与 DG 容量可能呈现与本研究不同的结果，需要结合具体情况分析。

7. 模型评价与改进

7.1 模型优点

- 基于蒙特卡洛模拟的概率分析，避免了引入神经网络而带来的复杂计算，同时代码模块化设计，便于理解，算法简单；
- 模型考虑了多场景下的风险因素，贴合实际需求；
- 可视化 NetworkX 输电网络，便于调试与直观理解。

7.2 模型缺点

- 模型假设“同一时间同一类型只发生一个故障”，但实际配电网可能出现多重故障，模型假设与实际情况存在差异；
- 模型未考虑联络开关闭合或断开的动作时间、保护装置配合等实际操作约束；
- 蒙特卡洛模拟次数较高，耗时较长，不能进行实时风险评估。

7.3 模型改进

根据上述优缺点分析，我们提出下列可能的模型改进方向：

7.3.1 基于图神经网络的拓扑特征与故障风险分析

针对分布式能源接入后配电网拓扑结构的复杂化，可以采用图神经网络（GNN）构建拓扑 - 风险关联模型。GNN 以图信号处理理论为基础，将 62 节点有源配电网抽象为包含节点（变压器、母线、分布式电源、负荷节点等）和边（馈线、联络线）的图结构。在模型训练过程中，利用消息传递机制，节点通过聚合邻接节点和边的电气参数（如线路阻抗、节点注入功率）、拓扑连接关系等特征，迭代更新自身状态。以联络开关 S13 - 1、S29 - 2、S62 - 3 为例，其开合状态变化引发的拓扑结构改变，可通过节点嵌入向量动态表征。经过多层图卷积操作，

GNN 能够捕捉故障在不同馈线间的传播路径特性，结合故障单元故障率（分布式能源 0.5%、开关 0.2%、线路长度 ×0.002/km），构建基于历史故障数据训练的传播概率矩阵。该矩阵用于预测特定故障发生时，失负荷风险在各节点和线路的扩散轨迹，进而通过节点风险评分函数，量化计算各区域失负荷概率 PLL，为失负荷风险模型中联络线转供策略的制定提供数据支持。

7.3.2 深度强化学习驱动的负荷转移策略优化

为实现系统拓扑变化时经济损失最小化的目标，可以基于深度强化学习（DRL）设计动态负荷转移策略。DRL 框架以马尔可夫决策过程（MDP）为基础，将配电网运行状态（节点电压、线路负载率、联络开关状态、分布式能源出力等）定义为状态空间，联络开关的开合操作作为动作空间，以失负荷经济损失、过负荷惩罚成本等构建奖励函数。在 62 节点系统中，智能体通过深度神经网络（如深度 Q 网络）感知馈线负载情况，当检测到某馈线（如馈线 1）因故障出现失负荷时，智能体根据历史经验与探索机制，尝试调整联络开关 S13-1，将部分负荷转供至相邻馈线。通过时序差分学习算法，以最小化系统风险 $R_{sys} = P_{LL}C_{LL} + P_{OL}C_{OL}$ 为优化目标，迭代更新网络参数。训练过程中，智能体在分布式能源容量从 I 到 3I（步长 0.3I）的不同工况下探索最优策略，形成适应动态变化的负荷转移方案，有效降低失负荷风险和过负荷风险，满足不向上级电网倒送功率的约束条件。

7.3.3 不确定性量化与随机建模方法

针对分布式能源出力的波动性，可以采用不确定性量化（UQ）技术，基于随机偏微分方程（SPDE）构建 DG 出力随机模型。以接入系统的 8 个分布式能源为例，将设备故障概率作为随机变量，利用伊藤微积分构建能量转换随机微分方程。通过有限元法和蒙特卡洛模拟对 SPDE 进行离散求解，生成大量不同容量（I－3I）下的 DG 出力随机样本。采用核密度估计方法，刻画不同设备状态下 DG 出力的概率分布特性，并将其融入过负荷风险模型。在计算过负荷概率 POL 时，通过机会约束规划，确保在 95% 置信水平下，馈线有功功率不向上级变电站倒送，同时在相邻馈线间合理调节功率，实现过负荷风险的有效控制。

8. 参考文献

[1] XU Wei,许威,ZHANG Zhonghui,张忠会,HE Lezhang,何乐彰.基于多因素元件停运模型的电网失负荷计算方法[C].//江西省电机工程学会.2014 年江西省电机工程学会年会论文集.2014:351-354.

附录

附完整代码:

```
import networkx as nx
import pandas as pd
import matplotlib.pyplot as plt
from typing import Dict, List, Tuple, Optional
import numpy as np
from collections import deque
import datetime
import pandapower as pp
# import math
# import ipython.display import set_matplotlib_formats
plt.rcParams["font.family"] = ["SimHei"]
# netwokx 网络建立初始化
# 输出：networkx 网络
def initialization(p_dg):
    # 读取 excel 数据
    nodes = pd.read_excel('data.xlsx', sheet_name='1', header=0)
    edges = pd.read_excel('data.xlsx', sheet_name='2', header=0)
    # networkx 构建电网
    G = nx.Graph()
    # 录入 excel 中的 62 名用户，属性：p(功率)，cat(分类)
    for idx, row in nodes.iterrows():
        node_id = row['No.']
        p = row['有功 P/kW']
        cat = row['分类']
        attrs = {'p': p, 'cat': cat}
        G.add_node(node_id, **attrs)
    # 录入 excel 中的配电线路，属性：length(长度)，resistance(电
阻)，reactance(电抗)，switch
    # switch 用来判断是否有开关，若为 Si 则是开关名，若为 0 则是没有开关
    for idx, row in edges.iterrows():
        start = row['起点']
        end = row['终点']
        length = row['长度/km']
        resistance = row['电阻/Ω']
        reactance = row['电抗/Ω']
        switch = row['开关']
        G.add_edge(start, end,
                   length=length,
```

```
                resistance=resistance,
                reactance=reactance,
                switch=switch)
    # 添加DG，并关联到已有网络，属性：p，cat
    G.add_node('DG16', cat='dg', p=p_dg)
    G.add_node('DG22', cat='dg', p=p_dg)
    G.add_node('DG32', cat='dg', p=p_dg)
    G.add_node('DG35', cat='dg', p=p_dg)
    G.add_node('DG39', cat='dg', p=p_dg)
    G.add_node('DG48', cat='dg', p=p_dg)
    G.add_node('DG52', cat='dg', p=p_dg)
    G.add_node('DG55', cat='dg', p=p_dg)
    G.add_edge('DG16', 16, length=0, resistance=0, reactance=0, switch=
0)
    G.add_edge('DG22', 22, length=0, resistance=0, reactance=0, switch=
0)
    G.add_edge('DG32', 32, length=0, resistance=0, reactance=0, switch=
0)
    G.add_edge('DG35', 35, length=0, resistance=0, reactance=0, switch=
0)
    G.add_edge('DG39', 39, length=0, resistance=0, reactance=0, switch=
0)
    G.add_edge('DG48', 48, length=0, resistance=0, reactance=0, switch=
0)
    G.add_edge('DG52', 52, length=0, resistance=0, reactance=0, switch=
0)
    G.add_edge('DG55', 55, length=0, resistance=0, reactance=0, switch=
0)
    #添加CB，并关联到已有网络，属性：p，cat
    G.add_node('CB1', cat='cb', p=2200)
    G.add_node('CB2', cat='cb', p=2200)
    G.add_node('CB3', cat='cb', p=2200)
    G.add_edge('CB1', 1, length=0, resistance=0, reactance=0, switch=0)
    G.add_edge('CB2', 43, length=0, resistance=0, reactance=0, switch=0
)
    G.add_edge('CB3', 23, length=0, resistance=0, reactance=0, switch=0
)
    #添加跨线开关，但不关联到已有网络，属性：cat，p
    G.add_node('S13-1', cat='link', p=0)
    G.add_node('S29-2', cat='link', p=0)
    G.add_node('S62-3', cat='link', p=0)
```

```
    # G.add_edge('S13-
1', 13, length=0, resistance=0, reactance=0, switch=0)
    # G.add_edge('S13-
1', 43, length=0, resistance=0, reactance=0, switch=0)
    # G.add_edge('S29-
2', 19, length=0, resistance=0, reactance=0, switch=0)
    # G.add_edge('S29-
2', 29, length=0, resistance=0, reactance=0, switch=0)
    # G.add_edge('S62-
3', 23, length=0, resistance=0, reactance=0, switch=0)
    # G.add_edge('S62-
3', 62, length=0, resistance=0, reactance=0, switch=0)
    return G
# 绘制电网图函数
# 输入：networkx 网络
# 输出：plt.show()输出网络图，同时保存一张网络图 graph.png 到本地
def draw(G: nx.Graph):
    plt.figure(figsize=(15, 12))
    pos = nx.spring_layout(G, k=0.2, iterations=50, seed=42)
    node_colors = []
    # node_sizes = []
    legend_handles = []
    legend_labels = []
    cat_colors = {
        'res': 'lightblue',
        'com': 'green',
        'gov': 'red',
        'off': 'yellow',
        'link': 'orange',
        'default': 'gray'
    }
    cat_mapping = {
        'res': '居民',
        'com': '商业',
        'gov': '政府',
        'off': '办公',
        'link': '联络开关',
        'dg': '分布式电源',
        'cb': '变电站',
        'unknown': '未知'
    }
    '''
```

```
    for node in G.nodes:
        node_cat = G.nodes[node].get('cat', 'unknown')
        # print(f"节点 {node}: 分类={node_cat}\n")
        if node_cat == 'res':
            node_colors.append('lightblue')
        elif node_cat == 'com':
            node_colors.append('green')
        elif node_cat == 'gov':
            node_colors.append('red')
        elif node_cat == 'off':
            node_colors.append('yellow')
        elif node_cat == 'link':
            node_colors.append('orange')
        else:
            node_colors.append('gray')
        '''

    for node in G.nodes:
        node_cat = G.nodes[node].get('cat', 'unknown')
        color = cat_colors.get(node_cat, cat_colors['default'])
        node_colors.append(color)
        chinese_cat = cat_mapping.get(node_cat, '未知')
        # 创建图例
        # if node_cat not in legend_labels:
        if chinese_cat not in legend_labels:
            handle = plt.Line2D([0], [0], marker='o', color='w', marker
facecolor=color, markersize=10)
            legend_handles.append(handle)
            legend_labels.append(chinese_cat)
    # 绘制节点和配电线
    nx.draw_networkx_nodes(G, pos, node_color=node_colors, node_size=20
0, alpha=0.8)
    nx.draw_networkx_edges(G, pos, width=1.0, alpha=0.6, edge_color='gr
ay')
    # 节点注释
    # 简单注释
    node_labels = {node: f"{node}\n" for node in G.nodes}
    # 完整注释
    # node_labels = {node: f"{node}\ncat: {G.nodes[node].get('cat', 'un
known')}\np: {G.nodes[node].get('p', 'unknown')}kW" for node in G.nodes
}
    nx.draw_networkx_labels(G, pos, labels=node_labels, font_size=8)
```

```
    # 配电线注释
    # 简单注释
    edge_labels = {(u, v): f"{G[u][v].get('switch', 'unknown')}" for u,
 v in G.edges}
    # 完整注释
    # edge_labels = {(u, v): f"len: {G[u][v].get('length', 'unknown')}k
m\nR: {G[u][v].get('resistance', 'unknown')}Ω\nX: {G[u][v].get('reactan
ce', 'unknown')}Ω\nswitch: {G[u][v].get('switch', 'unknown')}" for u, v
 in G.edges}
    nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_
size=6)
    # 添加图例
    plt.legend(legend_handles, legend_labels, loc='upper right')
    plt.savefig('graph.png', dpi=300)
    # set_matplotlib_formats('retina')
    plt.axis('off')
    plt.tight_layout()
    plt.show()
# nx.draw(G)
# 初始化元件故障率
# 输入：networkx 网络
# 输出：所有节点故障率，所有边故障率，所有节点负荷功率，以及一个拥有所有开关的
开关列表
def failure_rate(G: nx.Graph) -> Tuple[Dict, Dict, Dict, Dict]:
    node_failure_rates = {}
    for node, data in G.nodes(data=True):
        if data['cat'] in ['res', 'com', 'gov', 'off']:
            node_failure_rates[node] = 0.005
        elif data['cat'] == 'dg':
            node_failure_rates[node] = 0.005
        elif data['cat'] == 'cb':
            node_failure_rates[node] = 0.002
        elif data['cat'] == 'link':
            node_failure_rates[node] = 0.002

    edge_failure_rates = {}
    for u, v, data in G.edges(data=True):
        length = data.get('length', 1)
        edge_failure_rates[(u, v)] = 0.002 * length
        if 'switch' in data and data['switch'] != 0:
            edge_failure_rates[(u, v)] += 0.002
```

```
    # 节点负荷功率
    node_loads = {node: data['p'] for node, data in G.nodes(data=True)
                  if data['cat'] in ['res', 'com', 'gov', 'off']}

    # 开关列表，包括普通的开关和联络开关，格式为：switches['S1'] = (1, 2)
    switches = {}
    # 添加普通开关
    for u, v, data in G.edges(data=True):
        if 'switch' in data and data['switch'] != 0:
            switches[data['switch']] = (u, v)
    # 添加联络开关
    for node, data in G.nodes(data=True):
        if data['cat'] == 'link':
            neighbors = list(G.neighbors(node))
            if len(neighbors) >= 2:
                switches[node] = (neighbors[0], neighbors[1])
            # for neighbor in G.neighbors(node):
            #     if G.has_edge(node, neighbor):
            #         switches[node] = (node, neighbor)
            #         break

    return node_failure_rates, edge_failure_rates, node_loads, switches
# 潮流分析，用于分析电网中各节点的供电状态
# 输入：networkx 网络，所有节点负荷功率，联络开关的最大运载功率
# 输出：一个字典，用来表示节点通电状态，格式：{节点 ID：(是否通电，分配的功率)}
def power_analysis(G: nx.Graph, node_loads: Dict, limit: float) -> Dict
:
    # G_copy = G.copy()
    node_status = {node: (False, 0) for node in G.nodes()}
    # 获取所有电源的功率
    power_sources = {}
    for node, data in G.nodes(data=True):
        if data.get('cat') == 'cb' or data.get('cat') == 'dg':
            power_sources[node] = data.get('p', 0)
    # BFS，从电源节点开始对所有用户节点进行功率分配
    for src, power in power_sources.items():
        visited = set()
        queue = deque([(src, power)])
        node_status[src] = (True, power)
        visited.add(src)
        while queue:
```

```
            cur_node, remain_power = queue.popleft()
            for neighbor in G.neighbors(cur_node):
                if neighbor in visited:
                    continue

                # 如果是用户节点，就分配功率
                if G.nodes[neighbor].get('cat') in ['res', 'com', 'gov'
, 'off']:
                    require_power = node_loads.get(neighbor, 0)
                    allocate_power = min(require_power, remain_power)
                    is_supplied = allocate_power >= require_power
                    node_status[neighbor] = (is_supplied, allocate_powe
r)
                    new_remain = remain_power - allocate_power
                # 如果是联络开关，直接分配功率(p=0)，但联络开关能通过的功率
不超过其最大运载功率限制
                elif G.nodes[neighbor].get('cat') == 'link':
                    allocate_power = 0
                    new_remain = min(remain_power, limit)
                    node_status[neighbor] = (True, allocate_power)
                # 如果不是用户节点，直接跳过
                else:
                    allocate_power = 0
                    new_remain = remain_power

                # 继续按顺序分配功率
                if new_remain > 0:
                    visited.add(neighbor)
                    queue.append((neighbor, new_remain))
    return node_status
# 模拟单次故障
# 输入：networkx 网络，所有节点故障率，所有边故障率，所有节点负荷功率，开关列
表，联络开关的最大运载功率
# 输出：经处理的networkx 网络，是否有故障(0 or 1)，失负荷量，各节点供电状态的
数组
def single_failure(G: nx.Graph,
                   node_failure_rates: Dict,
                   edge_failure_rates: Dict,
                   node_loads: Dict,
                   switches: Dict,
                   limit: float) -> Tuple[nx.Graph, List, float, Dict]:
    G_copy = G.copy()
```

```
    # 移除联络开关相关的边
    contact_edges = [
        ('S13-1', 13), ('S13-1', 43),
        ('S29-2', 19), ('S29-2', 29),
        ('S62-3', 23), ('S62-3', 62)
    ]
    for u, v in contact_edges:
        if G_copy.has_edge(u, v):
            G_copy.remove_edge(u, v)
    # 初始化四类故障元件
    user = [node for node in node_failure_rates if G.nodes[node]['cat']
 in ['res', 'com', 'gov', 'off']]
    dg = [node for node in node_failure_rates if G.nodes[node]['cat'] =
= 'dg']
    switch = list(switches.keys())
    edges = list(edge_failure_rates.keys())
    failed_user = None
    failed_dg = None
    failed_switch = None
    failed_edge = None
    is_failed = [] # 判断是否有故障发生
    '''
    # 随机用户故障
    if user and np.random.rand() < sum(node_failure_rates[node] for nod
e in user):
        user_probs = [node_failure_rates[node] for node in user]
        total_user_rate = sum(user_probs)
        user_probs = [p / total_user_rate for p in user_probs]
        failed_user = np.random.choice(user, p=user_probs)
        is_failed.append(failed_user)
    # 随机 DG 故障
    if dg and np.random.rand() < sum(node_failure_rates[node] for node
in dg):
        dg_probs = [node_failure_rates[node] for node in dg]
        total_dg_rate = sum(dg_probs)
        dg_probs = [p / total_dg_rate for p in dg_probs]
        failed_dg = np.random.choice(dg, p=dg_probs)
        is_failed.append(failed_dg)

    # 随机开关故障
    if switch and np.random.rand() < sum(node_failure_rates.get(switch,
 0) for switch in switch):
```

```
        switch_probs = [node_failure_rates.get(switch, 0) for switch in
switch]
        total_switch_rate = sum(switch_probs)
        switch_probs = [p / total_switch_rate for p in switch_probs]
        failed_switch = np.random.choice(switch, p=switch_probs)
        is_failed.append(failed_switch)

    # 随机线路故障
    if edges and np.random.rand() < sum(edge_failure_rates.values()):
        edge_probs = [edge_failure_rates[edge] for edge in edges]
        total_edge_rate = sum(edge_probs)
        edge_probs = [p / total_edge_rate for p in edge_probs]
        edge_copy = np.arange(len(edges))
        selected_index = np.random.choice(edge_copy, p=edge_probs)
        # failed_edge = np.random.choice(edges, p=edge_probs)
        # is_failed.append(failed_edge)
        failed_edge = edges[selected_index]
        is_failed.append(failed_edge)
    '''
    # 随机用户故障
    if user:
        user_probs = [node_failure_rates[node] for node in user]
        total_user_rate = sum(user_probs)
        if total_user_rate > 0:
            user_probs = [p / total_user_rate for p in user_probs]
            if np.random.rand() < total_user_rate:
                failed_user = np.random.choice(user, p=user_probs)
                is_failed.append(failed_user)
    # 随机 DG 故障
    if dg:
        dg_probs = [node_failure_rates[node] for node in dg]
        total_dg_rate = sum(dg_probs)
        if total_dg_rate > 0:
            dg_probs = [p / total_dg_rate for p in dg_probs]
            if np.random.rand() < total_dg_rate:
                failed_dg = np.random.choice(dg, p=dg_probs)
                is_failed.append(failed_dg)
    # 随机开关故障
    if switch:
        switch_probs = [node_failure_rates.get(switch, 0) for switch in
switch]
        total_switch_rate = sum(switch_probs)
```

```
        if total_switch_rate > 0:
            switch_probs = [p / total_switch_rate for p in switch_probs
]
            if np.random.rand() < total_switch_rate:
                failed_switch = np.random.choice(switch, p=switch_probs
)
                is_failed.append(failed_switch)
    # 随机线路故障
    if edges:
        edge_probs = [edge_failure_rates[edge] for edge in edges]
        total_edge_rate = sum(edge_probs)
        if total_edge_rate > 0:
            edge_probs = [p / total_edge_rate for p in edge_probs]
            if np.random.rand() < total_edge_rate:
                edge_copy = np.arange(len(edges))
                selected_index = np.random.choice(edge_copy, p=edge_pro
bs)
                failed_edge = edges[selected_index]
                is_failed.append(failed_edge)

    # 应用故障
    # 如果用户故障，移除故障用户节点
    if failed_user is not None:
        G_copy.remove_node(failed_user)

    # 如果DG故障，移除故障DG节点
    if failed_dg is not None:
        G_copy.remove_node(failed_dg)
    # 如果开关故障，移除开关对应的线路
    if failed_switch is not None:
        if failed_switch in switches:
            u, v = switches[failed_switch]
            if G_copy.has_edge(u, v):
                G_copy.remove_edge(u, v)

    # 如果线路故障，移除故障线路
    if failed_edge is not None:
        u, v = failed_edge
        if G_copy.has_edge(u, v):
            G_copy.remove_edge(u, v)

    #print(is_failed)
```

```
    # 如果有故障发生
    if is_failed:
        # for link_switch in [node for node, data in G.nodes(data=True)
 if data['cat'] == 'link']:
        #     neighbors = list(G.neighbors(link_switch))
        #     if len(neighbors) >= 2:
                # u, v = neighbors[0], neighbors[1]
                # if not G_copy.has_edge(u, v):
                    # G_copy.add_edge(u, v, switch=0, length=0, resista
nce=0, reactance=0)
        # 把联络开关连接到电路里
        G_copy.add_edge('S13-
1', 13, length=0, resistance=0, reactance=0, switch=0)
        G_copy.add_edge('S13-
1', 43, length=0, resistance=0, reactance=0, switch=0)
        G_copy.add_edge('S29-
2', 19, length=0, resistance=0, reactance=0, switch=0)
        G_copy.add_edge('S29-
2', 29, length=0, resistance=0, reactance=0, switch=0)
        G_copy.add_edge('S62-
3', 23, length=0, resistance=0, reactance=0, switch=0)
        G_copy.add_edge('S62-
3', 62, length=0, resistance=0, reactance=0, switch=0)
            # 如果此时联通开关是断开的，就闭合联通开关
            # if (G_copy.has_edge(link_switch, u) and
            #         G_copy.has_edge(link_switch, v) and
            #         G_copy[link_switch][u]['switch'] == 0 and
            #         G_copy[link_switch][v]['switch'] == 0):
            #
            #         G_copy[link_switch][u]['switch'] = 1
            #         G_copy[link_switch][v]['switch'] = 1
        # 通过功率计算各节点供电状态
        node_status = power_analysis(G_copy, node_loads, limit)
        # 计算失负荷量
        lost_load = sum(node_loads[node] for node in node_loads
                        if node in node_status and not node_status[node
][0])
        return G_copy, is_failed, lost_load, node_status
    # 如果没有故障发生
    else:
        node_status = power_analysis(G_copy, node_loads, limit)
        lost_load = sum(node_loads[node] for node in node_loads
```

```
                        if node in node_status and not node_status[node
][0])
        return G_copy, [], lost_load, node_status

def get_feeder_nodes(G: nx.Graph, cb_node: str) -> set:
    """
    获取指定 CB 的供电区域（不包含联络开关节点）
    - G: 当前网络拓扑
    - cb_node: 起始 CB 节点（CB1/CB2/CB3）
    - 返回值: {供电区域节点}
    """
    visited = set()
    queue = deque([cb_node])
    visited.add(cb_node)

    while queue:
        node = queue.popleft()
        for neighbor in G.neighbors(node):
            if neighbor not in visited:
                # 排除联络开关节点
                if G.nodes[neighbor].get('cat') != 'link':
                    visited.add(neighbor)
                    queue.append(neighbor)
    return visited
def check_overload(G: nx.Graph, node_loads: Dict, ol_hazard_num: float,
contact_limit: float = 1.5e3) -> float:
    """
    检查馈线是否过载，并通过联络开关转移功率
    - G: 当前网络拓扑
    - node_loads: 节点负荷功率字典
    - contact_limit: 联络开关最大转供功率（默认 1.5MW）
    - 返回值: 过载危害值
    """
    overload_hazard = 0.0
    V_line = 10.0  # kV
    I_threshold = 242.0  # A:过负荷阈值
    threshold = I_threshold * V_line * np.sqrt(3)  # kW:过载阈值对应的功
率
    # 获取各 CB 的供电区域
    # feeder1_nodes = get_feeder_nodes(G, 'CB1')
    # feeder2_nodes = get_feeder_nodes(G, 'CB2')
    # feeder3_nodes = get_feeder_nodes(G, 'CB3')
```

```
    # print(f"CB1 节点：{feeder1_nodes}"
    #       f"\nCB2 节点：{feeder2_nodes}"
    #       f"\nCB3 节点：{feeder3_nodes}")
    # 各馈线的净功率 = 负荷总和 - DG 总出力
    total_power = {
        'CB1': 0.0,
        'CB2': 0.0,
        'CB3': 0.0
    }
    excess = {}
    capacity = {}
    for cb in ['CB1', 'CB2', 'CB3']:
        feeder_nodes = get_feeder_nodes(G, cb) # 获取各 CB 的供电区域
        load_sum = sum(node_loads.get(node, 0) for node in feeder_nodes
) # 计算各 CB 的总负荷
        dg_sum = 0.0
        for dg_node in ['DG16', 'DG22', 'DG32', 'DG35', 'DG39', 'DG48',
 'DG52', 'DG55']:
            if dg_node in feeder_nodes:
                dg_sum += G.nodes[dg_node]['p'] # 各 CB 的 DG 出力
        total_power[cb] = max(0,load_sum - dg_sum)  # 净功率 = 负荷总
和 - DG 出力 同时考虑功率不能倒送
        # 处理 DG 过剩功率转移
        # 过剩功率 = DG 出力 - 负荷（若 DG 出力 > 负荷）
        excess[cb] = max(0.0, dg_sum - load_sum)
        # 剩余容量 = 馈线最大允许功率 - 净功率
        capacity[cb] = threshold - total_power[cb]
    # 迭代处理过载，直到无法转移或所有馈线不超载
    changed = True
    while changed:
        changed = False
        # DG 出力过多时，多余的出力向其它馈线转移
        for from_cb in ['CB1', 'CB2', 'CB3']:
            if excess[from_cb] > 0:
                # 找到相邻馈线，按剩余容量排序（优先转移给剩余容量大的）
                if from_cb == 'CB1':
                    adjacent = ['CB2', 'CB3']
                elif from_cb == 'CB2':
                    adjacent = ['CB1', 'CB3']
                else:
                    adjacent = ['CB1', 'CB2']
                # 按剩余容量从高到低排序
```

```
                sorted_adj = sorted(adjacent, key=lambda x: capacity[x]
, reverse=True)
                for to_cb in sorted_adj:
                    transferable = min(excess[from_cb], capacity[to_cb]
, contact_limit)
                    if transferable > 0:
                        # 执行转移
                        excess[from_cb] -= transferable
                        capacity[to_cb] -= transferable
                        # 更新净功率
                        total_power[from_cb] += transferable  # 出方净功
率增加（因转移出去）
                        total_power[to_cb] -= transferable    # 接收方净
功率减少（因接收功率）
                        changed = True
                        break  # 转移后跳出循环，继续处理下一个馈线

    # 步骤2：处理过载（与之前逻辑相同）
    for cb in ['CB1', 'CB2', 'CB3']:
        if total_power[cb] > threshold:
            overload_amount = total_power[cb] - threshold
            overload_hazard += overload_amount * ol_hazard_num
    return overload_hazard
# 蒙特卡洛模拟，进行失负荷计算
# 输入：networkx 网络，所有节点故障率，所有边故障率，所有节点负荷功率，开关列
表，不同类型用户的危害系数，联络开关的最大运载功率，模拟次数
# 输出：平均失负荷风险值，平均失负荷功率值，失负荷的节点列表
def montecarlo(
        G: nx.Graph,
        node_failure_rates: Dict,
        edge_failure_rates: Dict,
        node_loads: Dict,
        switches: Dict,
        hazard_num: Dict,
        ol_hazard_num: float,
        limit: float,
        simulation_times: int = 10000) -> Tuple[float, float, List[floa
t],float,float,float]:
    total_hazard = 0.0
    total_lost_load = 0.0
    hazard_list = []
    total_overload_hazard = 0.0 # 过负荷风险
```

```
    lostload_times = 0  # 失负荷次数
    overload_times = 0  # 过负荷次数
    for i in range(simulation_times):
        # 生成DG的随机出力（正态分布）
        G_temp = G.copy()
        for dg_node in ['DG16', 'DG22', 'DG32', 'DG35', 'DG39', 'DG48',
 'DG52', 'DG55']:
            mean = G_temp.nodes[dg_node]['p']
            std = mean * 0.2    # 波动标准差设为额定容量的20%
            perturbed_p = np.random.normal(mean, std) # 设置DG出力为正态
分布
            perturbed_p = max(0, perturbed_p)  # 确保功率不为负
            G_temp.nodes[dg_node]['p'] = perturbed_p

        # 模拟单次故障
        G_copy, is_failed, lost_load, node_status = single_failure(
            G_temp, node_failure_rates, edge_failure_rates, node_loads,
 switches, limit)


        # 如果用户没有供电(node_status=False)，计算危害度C，并将危害度累加
起来
        hazard = 0.0
        for node in node_loads:
            if node in node_status and not node_status[node][0]:
                user_cat = G.nodes[node]['cat']
                hazard += node_loads[node] * hazard_num[user_cat] # C=节
点功率*危害度系数

        if hazard > 0:  # 如果发生失负荷，失负荷次数加1
            lostload_times += 1

        total_hazard += hazard
        total_lost_load += lost_load
        hazard_list.append(hazard)
        overload_hazard = check_overload(G_copy, node_loads, ol_hazard_
num ,contact_limit=1.5e6)
        if overload_hazard > 0:  # 如果发生过载，过载次数加1
            overload_times += 1
        total_overload_hazard += overload_hazard
        # if (i+1) % 1000 == 0:
        #    print(f"完成{i+1}/{simulation_times}次模拟")
```

```
            stride(i + 1, simulation_times)

        # 计算平均失负荷风险，计算平均失负荷量
        avg_hazard = total_hazard / simulation_times
        avg_lost_load = total_lost_load / simulation_times
        # 计算平均过负荷风险
        avg_overload_hazard = total_overload_hazard / simulation_times
        # 计算失负荷和过负荷频率，并用频率趋近概率
        lostload_prob = lostload_times / simulation_times
        overload_prob = overload_times / simulation_times
        #print(overload_times)
        return avg_hazard, avg_lost_load, hazard_list, avg_overload_hazard,
lostload_prob, overload_prob
# 一个用来打印进度条的函数
# 输入：当前循环次数，总循环次数
# 输出：一个进度条
def stride(iter, total, length=100):
    # percent = "{:.1f}".format(100 * (iter / float(total)))
    filled_length = int(length * iter // total)
    bar = '#' * filled_length + ' ' * (length - filled_length)
    print(f'\r进度: |{bar}| {iter}/{total}', end='', flush=True)
    if iter == total:
        print()
def simulate_and_draw(init_p_dg, step, max_p_dg, simulation_times, limi
t, hazard_num, ol_hazard_num):
    dg_capacity = np.arange(init_p_dg, max_p_dg + step, step)
    risk_list = []
    lostload_prob_list = []
    overload_prob_list = []
    overload_risk_list = []
    r_sys_list = []
    for p_dg in dg_capacity:
        print(f"当前 DG 容量: {p_dg} kW")
        G = initialization(p_dg)
        # draw(G)
        node_failure_rates, edge_failure_rates, node_loads, switches =
failure_rate(G)
        # print(node_failure_rates)
        # print(edge_failure_rates)
        # print(node_loads)
        # print(switches)
```

```
        # G_copy, is_failed, lost_load, node_status = single_failure(G,
 node_failure_rates, edge_failure_rates, node_loads, switches)
        # print(G_copy)
        # print(is_failed)
        # print(lost_load)
        # print(node_status)
        risk, avg_lost, hazard_list,overload_risk,lostload_prob,overloa
d_prob = montecarlo(
        G, node_failure_rates, edge_failure_rates, node_loads, switches
, hazard_num, ol_hazard_num, simulation_times=simulation_times, limit=l
imit)
        r_sys = risk + overload_risk
        print(f"系统失负荷风险: {risk:.2f} kW·危害系数")
        print(f"系统过负荷风险: {overload_risk:.2f} kW·危害系数")
        # draw(G_copy)
        print(f"失负荷概率: {lostload_prob:.4f}")
        print(f"过负荷概率: {overload_prob:.4f}")
        print(f"系统风险: {r_sys:.4f} kW·危害系数")
        risk_list.append(risk)
        lostload_prob_list.append(lostload_prob)
        overload_risk_list.append(overload_risk)
        overload_prob_list.append(overload_prob)
        r_sys_list.append(r_sys)
    # _, axes = plt.subplots(3, 1, figsize=(12, 6))

    # 绘制失负荷风险和失负荷概率图
    fig1, ax1 = plt.subplots(figsize=(12, 6))
    ax1.set_xlabel('DG 容量 (kW)')
    ax1.set_ylabel('失负荷风险 (kW·危害系数)', color='black')
    line1 = ax1.plot(dg_capacity, risk_list, label='失负荷风险
', color='blue')
    ax1.tick_params(axis='y', labelcolor='black')
    ax2 = ax1.twinx()
    ax2.set_ylabel('失负荷概率', color='black')
    line2 = ax2.plot(dg_capacity, lostload_prob_list, label='失负荷概率
', color='red')
    ax2.tick_params(axis='y', labelcolor='black')
    lines = line1 + line2
    labels = [l.get_label() for l in lines]
    ax1.legend(lines, labels, loc='upper left')
    ax1.set_title('失负荷风险和失负荷概率')
    plt.savefig('lostload.png', dpi=300)
```

```
    plt.show()
    # 绘制过负荷风险和过负荷概率图
    fig2, ax3 = plt.subplots(figsize=(12, 6))
    ax3.set_xlabel('DG 容量 (kW)')
    ax3.set_ylabel('过负荷风险 (kW·危害系数)', color='black')
    line3 = ax3.plot(dg_capacity, overload_risk_list, label='过负荷风险
', color='blue')
    ax3.tick_params(axis='y', labelcolor='black')
    ax4 = ax3.twinx()
    ax4.set_ylabel('过负荷概率', color='black')
    line4 = ax4.plot(dg_capacity, overload_prob_list, label='过负荷概率
', color='red')
    ax4.tick_params(axis='y', labelcolor='black')
    lines = line3 + line4
    labels = [l.get_label() for l in lines]
    ax3.legend(lines, labels, loc='upper left')
    ax3.set_title('过负荷风险和过负荷概率')
    plt.savefig('overload.png', dpi=300)
    plt.show()
    # 绘制系统风险图
    plt.figure(figsize=(6, 6))
    plt.plot(dg_capacity, r_sys_list, label='系统风险')
    plt.xlabel('DG 容量 (kW)')
    plt.ylabel('系统风险 (kW·危害系数)')
    plt.title('系统风险')
    plt.legend()
    plt.savefig('rsys.png', dpi=300)
    plt.show()

    '''
    # 失负荷风险和失负荷概率
    ax1 = axes[0]
    ax1_r = ax1.twinx()
    ax1.plot(dg_capacity, risk_list, label='失负荷风险 (kW·危害系
数)', color='blue')
    ax1_r.plot(dg_capacity, lostload_prob_list, label='失负荷概率
', color='red')
    ax1.set_xlabel('DG 容量 (kW)')
    ax1.set_ylabel('失负荷风险 (kW·危害系数)')
    ax1_r.set_ylabel('失负荷概率')
    ax1.set_title('失负荷风险和失负荷概率')
    lines, labels = ax1.get_legend_handles_labels()
```

```
    lines2, labels2 = ax1_r.get_legend_handles_labels()
    ax1_r.legend(lines + lines2, labels + labels2, loc='upper right')
    # 过负荷风险和过负荷概率
    ax2 = axes[1]
    ax2_r = ax2.twinx()
    ax2.plot(dg_capacity, overload_risk_list, label='过负荷风险 (kW·危害系数)', color='blue')
    ax2_r.plot(dg_capacity, overload_prob_list, label='过负荷概率', color='red')
    ax2.set_xlabel('DG 容量 (kW)')
    ax2.set_ylabel('过负荷风险 (kW·危害系数)')
    ax2_r.set_ylabel('过负荷概率')
    ax2.set_title('过负荷风险和过负荷概率')
    lines, labels = ax2.get_legend_handles_labels()
    lines2, labels2 = ax2_r.get_legend_handles_labels()
    ax2_r.legend(lines + lines2, labels + labels2, loc='upper right')
    # 系统风险
    ax3 = axes[2]
    ax3.plot(dg_capacity, r_sys_list, label='系统风险 (kW·危害系数)', color='blue')
    ax3.set_xlabel('DG 容量 (kW)')
    ax3.set_ylabel('系统风险 (kW·危害系数)')
    ax3.set_title('系统风险')
    ax3.legend(loc='upper right')
    plt.subplots_adjust(hspace=0.5)
    plt.tight_layout()
    plt.show()
    '''
    return risk_list, overload_risk_list, lostload_prob_list, overload_prob_list, r_sys_list
if __name__ == '__main__':
    # 危害度系数初始化
    # 失负荷危害度系数
    hazard_num = {
        'res': 1.0,
        'com': 1.0,
        'gov': 1.0,
        'off': 1.0
    }
    ol_hazard_num = 1.0 # 过负荷危害系数，此处不区分故障节点，因为过负荷是整条线路的故障
    simulation_times = 1000000 # 模拟次数
```

```
    limit = 1500 # 联络开关的最大负载能力(kW)
    p_dg = 300 # 每个DG的输出功率(kW)
    # starttime = datetime.datetime.now()

    G = initialization(p_dg)
    draw(G)
    risk, overload_risk, lostload_prob, overload_prob, r_sys = simulate
_and_draw(p_dg, 0.3*p_dg, 3*p_dg, simulation_times, limit, hazard_num,
ol_hazard_num)
    # print(f"设定模拟次数{simulation_times}次，联络开关最大负载能力
{limit} kW，初始DG功率{p_dg} kW：")
    # endtime = datetime.datetime.now()
    # print(f"运行时间: {endtime-starttime}")
```