Modeling And Simulation

TOPIC:

Agent-Based Simulation of Smart Grid Resilience: Creating agent-based models to simulate and analyze the resilience of smart grid systems.

Ritvik Gupta

BY: RITVIK GUPTA
SAP: 500106977
BATCH: B1(HONS.)(DATA-SCIENCE)

TO:
DR. SAURABH SHANU

Chapter 1: Concept Note

1.1 Introduction

The production, distribution, and use of electricity have all seen radical changes in recent years due to the incorporation of smart grid technology into our energy infrastructure. Smart grids provide previously unheard-of levels of efficiency and adaptability. Still, they also bring new difficulties, in maintaining the system's resilience to various stresses. This project note investigates the creation of a complex agent-based simulation model as a proactive means of boosting smart grid resilience.

1.2 Problem Statement

The increasing complexity of smart grid systems raises worries about their capacity to resist and recover from unexpected occurrences such as extreme weather, cyber-attacks, and equipment failures. A robust smart grid is critical for ensuring a steady and dependable energy supply. As a result, the need to thoroughly understand and strengthen the resilience of smart grid systems is a major problem that this study aims to solve.

1.3 Stakeholders

This initiative's success is dependent on key stakeholders' involvement. Utility firms are critical to the operation of smart grids, while government organizations offer regulatory frameworks and standards. The development of smart grid components is facilitated by technology suppliers, and the functionality and resilience of the grid are directly experienced by end users, both residential and industrial.

Ritvik Gupta

1.4 Goals and Objectives

The major goal is to create an agent-based simulation model that accurately recreates the behavior of various components in the smart grid. Specific aims include:

- Determine the impact of external disturbances on the smart grid.
- Evaluate the efficacy of various resilience tactics.
- Identifying system vulnerabilities and weaknesses to make targeted changes.

1.5 Technical Requirements

We intend to use certain tools/languages, assuring compatibility with the current smart grid infrastructure. Integration with relevant data sources will give real-time data inputs, improving model accuracy.

1.6 Risk and Mitigation Strategies

Potential hazards include data errors, model complexity, and stakeholder resistance. Mitigation strategies include:

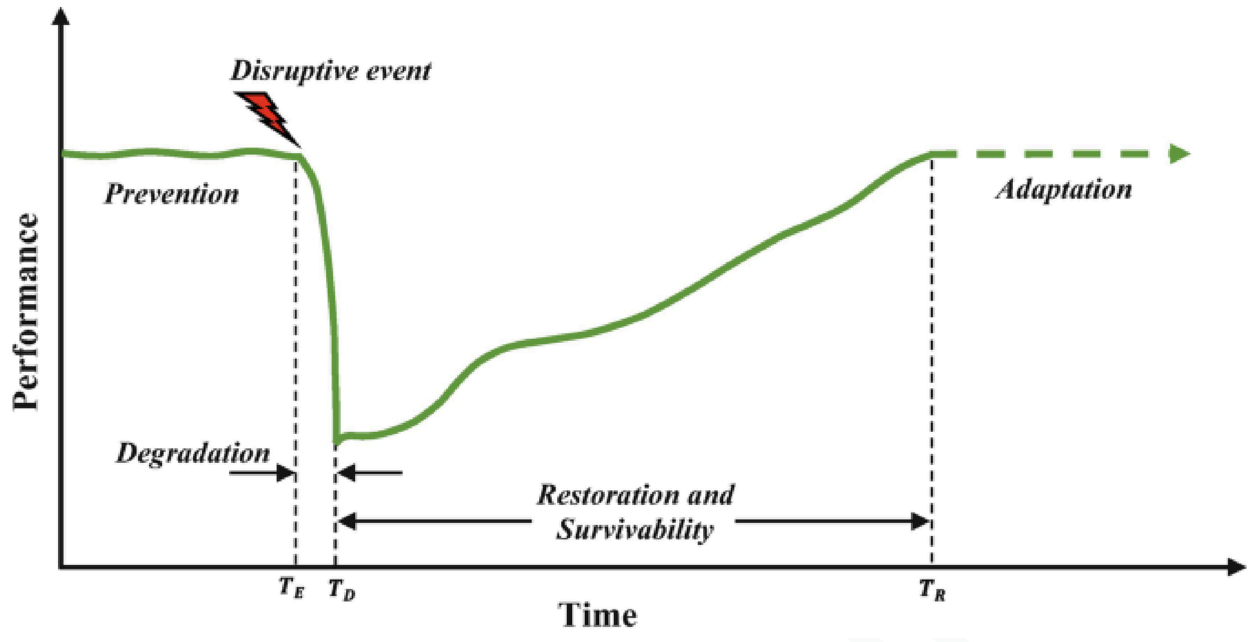
- Strict validation processes to correct data inconsistencies.
- An iterative model refining strategy for managing complexity.
- Proactive stakeholder involvement and communication to overcome any resistance.

Ritvik Gupta

Chapter 2: Literature Review:

Introduction:

Electrical distribution grids have significant challenges and opportunities as they transition to a decarbonized energy system and incorporate more renewable energy sources. To ensure grid stability, dependability, and efficiency, correct load and generation models must be constructed, as well as suitable demand response strategies implemented. Furthermore, the impact of extreme weather events on electrical system resilience needs to be carefully evaluated. This literature review will look at and evaluate research publications that address these key aspects of electrical grids in the context of smart grids and renewable energy integration.
The topic of this literature review is worth researching since it is critical in addressing the changing energy situation. With the increased use of renewable energy sources and the need for grid flexibility, precise load and generation models are essential for the appropriate design, operation, and management of electrical distribution networks. Furthermore, understanding and implementing demand response systems can improve grid stability, cost savings, and pollution reduction. Furthermore, extreme weather events, exacerbated by climate change, pose significant dangers to the resilience of electric systems, necessitating research, and strategies for improving their ability to withstand and recover from disruptions.
While great strides have been achieved in smart grids and renewable energy integration, significant research gaps and issues remain to be overcome. One such gap is the need for precise and probabilistic modeling of electrical household loads, which accounts for the inherent fluctuation and uncertainty in residential energy use patterns. There is also a need to explore and optimize demand response systems, taking into consideration price signals, user preferences, and grid constraints. Furthermore, the consequences of extreme weather events on electric system resilience, as well as mitigating strategies, must be studied further.



Defining essential terms and concepts:
a. Smart grids: Smart grids are modernized electrical grids that incorporate advanced technologies such as advanced metering infrastructure, real-time monitoring, and control systems, which enable two-way communication and the integration of renewable energy sources, demand response, and other innovative features.
b. Demand response: This involves altering power consumption patterns in response to price signals, grid circumstances, or other incentives. It allows customers to actively monitor their energy consumption while also ensuring grid stability and efficiency.
c. Agent-based modeling: Agent-based modeling is a computational modeling technique that models complex systems by simulating the activity and interactions of individual entities, known as agents, within them. It allows for the examination of emergent phenomena and the study of system dynamics.

This literature review focuses on four primary topics: modeling and implementation of probabilistic electrical household loads, demand response in smart grids, agent-based modeling of smart electricity grids, and the impact of extreme weather events on electric system resilience. These issues were chosen for their significance in ensuring the stability, efficiency, and resilience of electrical distribution grids in the context of renewable energy integration and climate change.

The concepts included in this literature review have been grouped based on their importance and interconnectedness. The examination begins by examining the modeling and application of probabilistic electrical household loads, as accurate load models are crucial for grid planning and operation. It then delves into the concept of demand response, discussing mathematical models, optimization approaches, and its role in improving grid dependability and efficiency. The next section explores the agent-based modeling of smart grids, stressing its applications and advantages in understanding the intricate linkages inside these systems. Finally, the study analyses the impact of extreme weather events on electric system resilience, as well as potential ways to improve system resilience.

This literature review comprises a selection of research publications that have contributed to our understanding of how to model electrical household loads, demand response, agent-based smart grid modeling, and the effects of extreme weather events on electric system resilience. The review focuses on the issues, techniques, and findings presented in these publications, which provide information about the current state of research in these disciplines. Its objective is to provide a thorough overview of the chosen topics while identifying research gaps and prospects.

Gupta

Review:

These research papers look at different aspects of electrical distribution networks, smart grids, and their resilience in the face of diverse threats. One of the topics raised in these studies is the decarbonization of the energy system, which comprises reducing greenhouse gas emissions and transitioning to cleaner and renewable energy sources. This transition is crucial to combating climate change and achieving environmental goals. The incorporation of renewable energy sources, such as solar and wind, into the power grid presents various challenges for electrical distribution networks. These issues include addressing the intermittent nature of renewable energy, ensuring grid stability and reliability, and making the optimum use of grid assets. The research papers stress the need to use accurate load and generation models to adequately assess and simulate these challenges.

To address these challenges, the studies suggest probabilistic bottom-up modeling of household load patterns. Unlike traditional systems that use standard load profiles, probabilistic profiles consider the randomness and diversity of individual family activity. This method gives a more realistic representation of energy consumption patterns and enables more extensive and up-to-date modeling of resident behavior. Understanding and forecasting future needs for electrical distribution networks is essential for effective grid planning, infrastructure development, and policy formulation.

Furthermore, the studies investigate the implications of probabilistic load profiles on asset utilization and node voltages in the distribution network. Accounting for unpredictability in-home load patterns, probabilistic modeling can provide insights into the potential stress on distribution grid assets and voltage levels. This information is incredibly important for improving asset management, identifying potential bottlenecks, and ensuring grid resilience.

The study articles also stress the need to use detailed and updated data for modeling. They highlight the flaws of previous studies that used out-of-date or insufficient data, resulting in inaccurate behavior modeling and predictions. Access to current and detailed data allows for more accurate and reliable modeling of household behaviors and load profiles. This data-driven method produces more accurate assessments of grid issues and enables evidence-based decision-making.

Other research publications look into the role of demand response in smart grids. Demand response programs encourage customers to adjust their electricity consumption in response to grid conditions or price indications. These initiatives are critical for enhancing grid stability, increasing energy efficiency, and facilitating the integration of renewable energy sources. The papers look at various mathematical models, techniques, and optimization strategies used in demand response programs to achieve these objectives.

Finally, the study pieces shed light on the challenges that electrical distribution grids confront, such as decarbonization, extreme weather events, and the integration of distributed energy resources. They stress the importance of exact modeling methodologies, the necessity for robust and sustainable energy systems, and the role of probabilistic modeling and demand response in resolving these difficulties. These papers' results and recommendations can help create the strategies, legislation, and technological advances needed to ensure that future energy systems are dependable, efficient, and environmentally sustainable.

Nitik Gupta

Conclusion:

Finally, this literature study looked at the modeling and analysis of electrical home loads, demand response in smart grids, agent-based modeling of smart electricity grids, and how extreme weather events affect electric system resilience. The papers examined shed light on various aspects of these subjects, highlighting the need for trustworthy and up-to-date data, probabilistic modeling, advanced optimization methodologies, and robust system design in addressing the challenges faced by electrical distribution grids. Probabilistic modeling of electrical household demands improves grid design and management, while demand response programs lower peak loads, and shift loads, and preserve grid stability. Agent-based modeling provides insight into the dynamics of smart grids and allows for more informed decision-making. Improving the resilience of electric networks is crucial for ensuring constant power delivery in the face of harsh weather. Stakeholders and policymakers can make informed decisions to improve the efficiency, dependability, and resilience of future energy systems by considering the study's findings and recommendations.

To progress in the area, future research should address the scalability and computing limitations of agent-based modeling, study novel demand response methodologies, and create strategies to improve electric system resilience in the face of changing climate conditions. Furthermore, developing technologies such as artificial intelligence and machine learning can improve load and generation projections, enhance demand response approaches, and boost system resilience.

Finally, as the world transitions to a more sustainable energy system, the research and conclusions presented in these papers provide critical insights and guidance for the design, planning, and management of future electrical distribution networks. Stakeholders may contribute to a dependable, efficient, and sustainable energy future by incorporating probabilistic load models, implementing demand response programs, utilizing agent-based modeling tools, and enhancing system resilience.

Ritvik Gupta

References and Citations:

- · Afrin, Nadia, et al. “Voltage Support Strategy for PV Inverter to Enhance Dynamic Voltage Stability of Islanded Microgrid.” *International Journal of Electrical Power & Energy Systems*, vol. 121, Oct. 2020, p. 106059, https://doi.org/10.1016/j.ijepes.2020.106059. Accessed 25 Feb. 2024.
- · Albasrawi, Murtadha N., et al. “Analysis of Reliability and Resilience for Smart Grids.” *2014 IEEE 38th Annual Computer Software and Applications Conference*, July 2014, https://doi.org/10.1109/compsac.2014.75. Accessed 25 Feb. 2024.
- · Alimi, Oyeniyi Akeem, et al. “A Review of Machine Learning Approaches to Power System Security and Stability.” *IEEE Access*, vol. 8, 2020, pp. 113512–113531, https://doi.org/10.1109/access.2020.3003568. Accessed 25 Feb. 2024.

  Arab, Ali, et al. “Three Lines of Defense for Wildfire Risk Management in Electric Power Grids: A Review.” *IEEE Access*, vol. 9, 2021, pp. 61577–61593, https://doi.org/10.1109/access.2021.3074477. Accessed 25 Feb. 2024.
- · Chen, Chen, et al. “Modernizing Distribution System Restoration to Achieve Grid Resiliency against Extreme Weather Events: An Integrated Solution.” *Proceedings of the IEEE*, vol. 105, no. 7, July 2017, pp. 1267–1288, https://doi.org/10.1109/jproc.2017.2684780. Accessed 25 Feb. 2024.
- · Cobilean, Victor, et al. “A Review of Visualization Methods for Cyber-Physical Security: Smart Grid Case Study.” *IEEE Access*, vol. 11, 2023, pp. 59788–59803, https://doi.org/10.1109/access.2023.3286304. Accessed 25 Feb. 2024.
- Deng, Ruilong, et al. “A Survey on Demand Response in Smart Grids: Mathematical Models and Approaches.” *IEEE Transactions on Industrial Informatics*, vol. 11, no. 3, June 2015, pp. 570–582, https://doi.org/10.1109/tii.2015.2414719. Accessed 25 Feb. 2024.
- · Diahovchenko, Illia, et al. *Electric Power Systems Research*. Vol. 221, 2023, p. 109460, https://doi.org/10.1016/j.epsr.2023.109460. Accessed 25 Feb. 2024.
- · Hauer, Daniel, et al. *A Methodology for Resilient Control and Monitoring in Smart Grids*.
- · Manditereza, Patrick T., and Ramesh C. Bansal. “Protection of Microgrids Using Voltage-Based Power Differential and Sensitivity Analysis.” *International Journal of Electrical Power & Energy Systems*, vol. 118, June 2020, p. 105756, https://doi.org/10.1016/j.ijepes.2019.105756. Accessed 25 Feb. 2024.
- · Meregillano, Eireka Orlido, and Laurence L. Delina. “Stronger Typhoons, Weaker Electricity Systems? A Review of the Impacts of Extreme Weather Events on Coastal Communities and Strategies for Electric System Resilience.” *The Electricity Journal*, vol. 36, no. 9-10, Nov. 2023, p. 107339, https://doi.org/10.1016/j.tej.2023.107339. Accessed 25 Feb. 2024.
- · Momen, Hamidreza, et al. “Load Restoration and Energy Management of a Microgrid with Distributed Energy Resources and Electric Vehicles Participation under a Two-Stage Stochastic Framework.” *International Journal of Electrical Power & Energy Systems*, vol. 133, Dec. 2021, p. 107320, https://doi.org/10.1016/j.ijepes.2021.107320. Accessed 25 Feb. 2024.

Ritvik Gupta

- · Ringler, Philipp, et al. "Agent-Based Modelling and Simulation of Smart Electricity Grids and Markets – a Literature Review." *Renewable and Sustainable Energy Reviews*, vol. 57, May 2016, pp. 205–215, https://doi.org/10.1016/j.rser.2015.12.169. Accessed 25 Feb. 2024.
- Roudbari, Alireza, et al. "Resilience-Oriented Operation of Smart Grids by Rescheduling of Energy Resources and Electric Vehicles Management during Extreme Weather Condition." *Sustainable Energy, Grids and Networks*, vol. 28, Dec. 2021, p. 100547, https://doi.org/10.1016/j.segan.2021.100547. Accessed 25 Feb. 2024.
- Vertgewall, Chris Martin, et al. "Modeling and Application of Probabilistic Electrical Household Loads in Distribution Grid Simulations." *2022 International Conference on Smart Energy Systems and Technologies (SEST)*, 5 Sept. 2022, https://doi.org/10.1109/sest53650.2022.9898438. Accessed 25 Feb. 2024.
- Wang, Hailong, et al. *Electric Power Systems Research*. Vol. 221, 2023, p. 109408, https://doi.org/10.1016/j.epsr.2023.109408. Accessed 25 Feb. 2024.

Ritvik Gupta

Chapter 3: Algorithm Designing

Ritvik's Smart Grid Resilience Simulator (Using Monte Carlo Simulation)

1. Input Parameters:

- **N:** Number of Monte Carlo iterations (integer)
- **T:** Number of simulation time steps (integer)
- **Grid_Model:** Structure containing component states (string), failure rates (float), and resilience metric function pointer (function pointer)
- **Demand_Model:** Function pointer representing the demand probability distribution function (function pointer)
- **Renewable_Gen_Model:** Function pointer representing the renewable generation probability distribution function (function pointer)
- **Resilience_Strategies:** Array of functions representing available resilience actions (list of function pointers)
- **System_Constraints:** Structure containing component state thresholds and imbalance threshold (dictionary)

2. Output:

- **Resilience_Metrics:** List of resilience metric values for each iteration (list of floats)

3. Declaration Section:

- **Δt:** Time step duration (float)
- **P_fail(i):** Pre-calculated failure probability for component i (list of floats)
- **D(n, t):** Demand level at time step t in iteration n (float)
- **G(n, t):** Renewable energy generation at time step t in iteration n (float)
- **O_i(n, t):** Binary variable indicating outage state of component i (list of lists of integers)
- **Imbalance(n, t):** Demand-supply imbalance at time step t in iteration n (list of floats)

- **C_i(n, t):** State of component i at time step t in iteration n (list of lists of strings)
- **Rn:** Resilience metric for iteration n (list of floats)

4. Procedure/Environment Section (Mathematics):

a. Initialization (t = 0):

```
1. $\Delta t$ = (Define time step duration)
2. For each component i in Grid_Model:
    ○ P_fail(i) = 1 - exp(-Grid_Model[i]["failure_rate"] * $\Delta t$)
3. For each iteration n (n = 1 to N):
    ○ $C_i$(n, 0) = [component_state for component_state in Grid_Model.keys()] (Initialize all component states)
    ○ For each time step t (t = 1 to T):
        ■ D(n, t) = Demand_Model(t) (Sample demand from the model)
        ■ G(n, t) = Renewable_Gen_Model(t) (Sample generation from the model)
        ■ $O_i$(n, t) = [0 for _ in Grid_Model.keys()] (Initialize outage states for all components)
```

b. State Update:

```
■ For each component i:
    ■ If $C_i$(n, t-1) == "operational" and random number < P_fail(i):
        ■ $C_i$(n, t) = "failed"
        ■ $O_i$(n, t) = [1] + $O_i$(n, t-1) (Update outage state)
    ■ Else:
        ■ $C_i$(n, t) = $C_i$(n, t-1)
        ■ $O_i$(n, t) = [0] + $O_i$(n, t-1)
```

c.Demand-Supply Check:

```
■ Imbalance(n, t) = D(n, t) - G(n, t)
```

- If Imbalance(n, t) > System__Constraints["imbalance__threshold"]:
    - Apply chosen resilience strategy using Resilience__Strategies (specific function call depends on chosen strategy)

d. Resilience Metric Calculation:

- Rn = Grid__Model"metric__function": O_i(n, t, C_i(n, t))

5. Result Section (Actual Outcome):

- **Resilience__Metrics** = [Rn for Rn in Rn]

Ritvik Gupta

Chapter 4: Tools, Techniques And Computation

Tools:

Programming Language(C):

- C is a powerful and versatile language well-suited for system simulations due to its efficiency and direct memory access capabilities.

C-Libraires

stdio.h:

- Provides standard input/output functions like printf used for printing simulation results to the console. In this code, printf is used to display the final resilience metric.

- **stdlib.h:**
    - Offers general utility functions like:
        - rand for generating pseudo-random numbers. This is crucial for simulating component failures based on their failure rates.
        - malloc and free for dynamic memory allocation and deallocation. These are used to allocate memory for arrays holding time-series data and

Ritvik Gupta

component information, and then free that memory after use to prevent memory leaks.

- **math.h:**
  - Contains mathematical functions like sin and cos used in modeling demand and generation patterns with daily and weekly variations.
- **string.h:**
  - Provides string manipulation functions like strcmp used for comparing strings (e.g., checking component state).
-

Ritvik Gupta

Techniques:

Struct Definition:

- A custom GridComponent struct is defined to represent each component in the grid. It acts like a blueprint, storing essential information about a component:
  - component_state: A character array (char *) to hold the current state (operational or failed).
  - failure_rate: A float representing the probability of the component failing within a timestep.
  - rating: A float representing the capacity or power rating of the component.
  - outage_duration: A pointer to a float array. This array is dynamically allocated using malloc to store the outage duration for each timestep (initialized to 0 for operational state).

Ritvik Gupta

Function Pointers:

- Function pointers are used to define different resilience strategies (resilience_strategy_1, etc.). They store the memory addresses of the functions. This allows for flexibility in choosing and applying the desired strategy during the simulation. The actual implementation of the strategies is defined in separate functions.

Random Number Generation:

- The rand function from stdlib.h is used to generate pseudo-random numbers. These numbers are used to simulate random component failures based on their pre-calculated failure probabilities. By comparing the generated random number with the probability, the code determines if a component fails within a specific timestep.

Demand And Generation Modeling:

- The code defines functions like sample_demand and sample_renewable_generation to model the demand and renewable energy generation patterns. These functions incorporate several techniques:
    - Sine Functions (sin, cos): Used to introduce daily and weekly variations in demand and generation. The amplitude of the sine function controls the magnitude of the variation, while the phase shift adjusts the starting point of the cycle within the day or week.
    - Scaling Factors: The base demand/generation is multiplied by various factors to introduce additional variations:
        - Random factor: A random number between 0 and 1 is used to introduce random fluctuations around the base value.
        - Component-specific factors: These factors consider how individual components might affect the overall demand/generation profile. They are modeled using sine functions with unique phases for each component.

Ritik Gupta

Monte Carlo Simulation:

- Monte Carlo Simulation is a computational method utilized in the modeling and assessment of the behavior of complex systems by repeated random sampling. It is called such because it depends on randomness to produce numerical results, which is why it shares its name with the famous Monte Carlo Casino in Monaco which has been popular for its games of chance and randomness.
- Monte Carlo simulation uses random sampling to provide numerical results for systems characterized by uncertain parameters or inputs. It has widespread applications in finance, engineering, physics, and statistics where there are systems that involve randomness or uncertainty for analysis or optimization.
- The code employs a Monte Carlo simulation technique. This involves running the simulation multiple times (iterations) with slight variations in the random number generation to account for uncertainties. By performing multiple iterations, the code can capture the average behavior of the grid and analyze its overall resilience under various scenarios.

- Mathematics:

  - Exponential distribution: $P(x)=\lambda e^{-\lambda x}$

  - Uniform distribution: $P(x) = \frac{1}{b-a} \; for \; a \leq x \leq b$

  - Normal distribution: $P(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$

Pitvik Gupta

Resilience Metric Calculation:

- A function named calculate_resilience_metric is defined to quantify the impact of component failures. It iterates through each component and calculates the product of its rating and outage duration. This represents the total energy not served by that component due to failures during the entire simulation. The calculated values for all components are then summed to provide a single metric for the overall resilience of the grid during a specific iteration.

Computation:

Arrays:

- The code heavily relies on arrays to store various time-series data for each iteration and timestep. Here are some key arrays used:
    - D: Stores the demand level for each timestep in each iteration. This is a 1D array of size N_ITERATIONS * N_TIMESTEPS.
    - G: Stores the renewable energy generation for each timestep in each iteration (same size as D).

Ritvik Gupta

- Oi: A 2D array of size N_ITERATIONS * N_TIMESTEPS x N_COMPONENTS. It holds the outage state (0 for operational, 1 for failed) of each component at each timestep within an iteration.
- Imbalance: A 2D array of size N_ITERATIONS * N_TIMESTEPS x 1. It stores the demand-supply imbalance (difference between demand and generation) for each timestep within an iteration.
- Ci: A 3D array of size N_ITERATIONS * N_TIMESTEPS x N_COMPONENTS x 16 (string length for component state). It holds the character arrays representing the state (operational/failed) of each component at each timestep within an iteration.

Looping:

- Nested loops are extensively used to iterate over different iterations and timesteps within each iteration. These loops drive the core simulation process:
    - The outer loop iterates over N_ITERATIONS, representing the number of Monte Carlo simulations.
    - The inner loop iterates over N_TIMESTEPS within each iteration, simulating the behavior of the grid for each timestep.

Ritik Gupta

Conditional Statements:

Conditional statements (if, else) play a crucial role in various aspects of the simulation:

- Component Failure: During the state update step, an if statement checks if the generated random number is less than the calculated failure probability for a component. If true, the component's state is changed to "failed" and its outage duration is initiated.

- Resilience Strategy Application: When the imbalance exceeds the imbalance_threshold, an if statement triggers the application of the chosen resilience strategy using function pointers.
-

Mathematical Operations:

- A variety of mathematical operations are used throughout the code:
- Basic Arithmetic: Addition, subtraction, multiplication, and division are used for calculations like:
  - Updating component outage durations.
  - Applying scaling factors to demand and generation.
  - Calculating demand-supply imbalance.
- Exponential Function: Used in the failure probability calculation (1 - exp(-Grid_Model[i].failure_rate * delta_t)) to model the probability of a component failing within a timestep based on its failure rate.
- Summation: The calculate_resilience_metric function uses summation to calculate the total energy not served by all components during an iteration.

Ritik Gupta

Chapter-5: Code Implementation in C

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

#define N_COMPONENTS 5
#define N_ITERATIONS 100
#define N_TIMESTEPS 24 * 4

// Defining Grid Model Structure
typedef struct
{
   char *component_state;
   float failure_rate;
   float rating;
   float *outage_duration;
} GridComponent;
```

Ritvik Gupta

```
// Function to sample demand from the demand model
float sample_demand(int timestep, int iteration, GridComponent
*components)
{
   float base_demand = 50.0; // Base demand
   float demand_factor_daily = 1.0 + 0.1 * sin(2 * M_PI * timestep /
(24.0 * 2.0) + M_PI / 4.0);         // Daily demand pattern
   float demand_factor_weekly = 1.0 + 0.1 * sin(2 * M_PI * iteration /
(N_ITERATIONS * 7.0) + M_PI / 4.0); // Weekly demand pattern
   float demand_factor_random = 1.0 + 0.05 * (float)rand() /
RAND_MAX;                              // Random demand variation
   float demand = base_demand * demand_factor_daily *
demand_factor_weekly * demand_factor_random;        // Calculate
demand
   for (int i = 0; i < N_COMPONENTS; i++)
// Apply component-specific demand factors
   {
      demand *= (1.0 + 0.02 * components[i].rating * sin(2 * M_PI *
timestep / 24.0 + 2 * M_PI * i / N_COMPONENTS));
   }
   return demand;
}

// Function to sample renewable generation from the renewable
generation model
```

Ritvik Gupta

```
float sample_renewable_generation(int timestep, GridComponent
*components)

{


   float base_generation = 50.0;
// Base generation

   float generation_factor_daily = 1.0 + 0.1 * sin(2 * M_PI * timestep /
24.0 + M_PI / 4.0);                    // Daily generation pattern

   float generation_factor_weekly = 1.0 + 0.1 * sin(2 * M_PI * timestep /
(24.0 * 7.0) + M_PI / 4.0);              // Weekly generation pattern

   float generation_factor_random = 1.0 + 0.05 * (float)rand() /
RAND_MAX;                                // Random generation variation

   float generation = base_generation * generation_factor_daily *
generation_factor_weekly * generation_factor_random; // Calculate
generation

   for (int i = 0; i < N_COMPONENTS; i++)
// Apply component-specific generation factors

   {

      generation += components[i].rating * 0.1 * sin(2 * M_PI * timestep /
24.0 + 2 * M_PI * i / N_COMPONENTS + M_PI / 4.0);

   }

   return generation;

}


// Function pointers for resilience strategies
```

Ritvik Gupta

```
void resilience_strategy_1(GridComponent *components)
{
  // Simulating by  activating a backup system (reducing imbalance by a
percentage)
  printf("Activating backup system (reducing imbalance)\n");
  float imbalance_reduction = 0.2; // Reduce imbalance by 20%
  for (int i = 0; i < N_COMPONENTS; i++)
  {
    if (components[i].outage_duration[0] > 0)
    {
      components[i].outage_duration[0] -= 1;
      if (components[i].outage_duration[0] == 0)
      {
        components[i].component_state = "operational";
      }
    }
  }
  float imbalance_before = 0.0;
  for (int i = 0; i < N_COMPONENTS; i++)
  {
    imbalance_before += components[i].rating *
(components[i].failure_rate * (components[i].component_state[0] ==
'f' ? 1.0 : 0.0));
```

Ritvik Gupta

```
  }
  float imbalance_after = 0.0;
  for (int i = 0; i < N_COMPONENTS; i++)
  {
    imbalance_after += components[i].rating *
(components[i].failure_rate * (components[i].component_state[0] ==
'f' ? 1.0 : 0.0));
  }
  imbalance_after -= imbalance_before * imbalance_reduction;
  for (int i = 0; i < N_COMPONENTS; i++)
  {
    if (components[i].failure_rate > 0 && imbalance_after > 0)
    {
      components[i].outage_duration[0] += 1;
      components[i].component_state = "failed";
      imbalance_after -= components[i].rating *
components[i].failure_rate;
    }
  }
}

void resilience_strategy_2(GridComponent *components)
{
```

Ritvik Gupta

```
  // Simulating load shedding (reducing demand to match supply)
  printf("Implementing load shedding (matching supply and
demand)\n");
  float imbalance = 0.0;
  for (int i = 0; i < N_COMPONENTS; i++)
  {
    imbalance += components[i].rating * (components[i].failure_rate *
(components[i].component_state[0] == 'f' ? 1.0 : 0.0));
  }
  if (imbalance > 0)
  {
    float demand_reduction = imbalance / (N_COMPONENTS * 0.9);
    for (int i = 0; i < N_COMPONENTS; i++)
    {
      if (components[i].failure_rate > 0)
      {
        components[i].rating *= 1.0 - demand_reduction /
(components[i].failure_rate * 0.1);
      }
    }
  }
}
```

Ritvik Gupta

```
void resilience_strategy_3(GridComponent *components)
{
  // Simulating demand response (shifting demand to match supply)
  printf("Implementing demand response (shifting demand to match supply)\n");
  float imbalance = 0.0;
  for (int i = 0; i < N_COMPONENTS; i++)
  {
    imbalance += components[i].rating * (components[i].failure_rate * (components[i].component_state[0] == 'f' ? 1.0 : 0.0));
  }
  if (imbalance > 0)
  {
    float demand_shift = imbalance / (N_COMPONENTS * 0.9);
    for (int i = 0; i < N_COMPONENTS; i++)
    {
      if (components[i].failure_rate > 0)
      {
        components[i].rating *= 1.0 + demand_shift / (components[i].failure_rate * 0.1);
      }
    }
  }
```

Ritvik Gupta

```
}

// Function to calculate resilience metric
float calculate_resilience_metric(GridComponent *components)
{
   float energy_not_served = 0.0;
   for (int i = 0; i < N_COMPONENTS; i++)
   {
      energy_not_served += components[i].rating *
components[i].outage_duration[0];
   }
   return energy_not_served;
}

int main()
{
   // Input Parameters
   int N; // Number of Monte Carlo iterations
   int T; // Number of simulation time steps

   // Defining Grid_Model as an array of GridComponent structures
   GridComponent Grid_Model[N_COMPONENTS];
```

Ritvik Gupta

```
  printf("Enter details for each component:\n");
  for (int i = 0; i < N_COMPONENTS; i++)
  {
    printf("Enter the values of failure rate and rating for component %d
\n", i + 1);
    Grid_Model[i].component_state = "operational";
    printf("Enter failure rate: ");
    scanf("%f", &Grid_Model[i].failure_rate);
    printf("Enter rating: ");
    scanf("%f", &Grid_Model[i].rating);
    Grid_Model[i].outage_duration = (float *)calloc(1, sizeof(float));
  }

  // Defining System_Constraints as a dictionary structure

  struct SystemConstraints
  {
    float imbalance_threshold;
    float state_threshlod;
  };

  struct SystemConstraints System_Constraints;
```

Ritvik Gupta

```
  System_Constraints.imbalance_threshold = 0.05;


  float imbalance_threshold =
System_Constraints.imbalance_threshold;


  // Declaration
  float delta_t;    // Time step duration
  float *P_fail;    // Pre-calculated failure probability for each
component
  float *D;         // Demand level at each time step in each iteration
  float *G;         // Renewable energy generation at each time step in each
iteration
  int **Oi;         // Outage state of each component at each time step in
each iteration
  float **Imbalance; // Demand-supply imbalance at each time step in
each iteration
  char ***Ci;       // State of each component at each time step in each
iteration
  float *Rn;        // Resilience metric for each iteration


  // Initialization Section
  // Initialization for Grid_Model components
  delta_t = 1.0; // value for time step duration
  for (int i = 0; i < N_COMPONENTS; i++)
```

Ritvik Gupta

```
  {
    P_fail = (float *)malloc(sizeof(float) * sizeof(Grid_Model) /
sizeof(Grid_Model[0]));
    P_fail[i] = 1 - exp(-Grid_Model[i].failure_rate * delta_t);
  }
  // Initialization for other variables
  D = (float *)calloc(N_ITERATIONS * N_TIMESTEPS, sizeof(float));
  G = (float *)calloc(N_ITERATIONS * N_TIMESTEPS, sizeof(float));
  Oi = (int **)calloc(N_ITERATIONS * N_TIMESTEPS, sizeof(int *) *
N_COMPONENTS);
  for (int i = 0; i < N_ITERATIONS * N_TIMESTEPS; i++)
  {
    Oi[i] = (int *)calloc(N_COMPONENTS, sizeof(int));
  }
  Imbalance = (float **)calloc(N_ITERATIONS * N_TIMESTEPS,
sizeof(float *));
  for (int i = 0; i < N_ITERATIONS * N_TIMESTEPS; i++)
  {
    Imbalance[i] = (float *)calloc(1, sizeof(float));
  }
  Ci = (char ***)calloc(N_ITERATIONS * N_TIMESTEPS, sizeof(char
**) * N_COMPONENTS);
  for (int i = 0; i < N_ITERATIONS * N_TIMESTEPS; i++)
  {
```

Ritvik Gupta

```
    Ci[i] = (char **)calloc(N_COMPONENTS, sizeof(char *));
    for (int j = 0; j < N_COMPONENTS; j++)
    {
      Ci[i][j] = (char *)calloc(16, sizeof(char));
      strcpy(Ci[i][j], "operational");
    }
  }
  Rn = (float *)calloc(N_ITERATIONS, sizeof(float));



  // Initialization (t = 0)
  for (int n = 0; n < N_ITERATIONS; n++)
  {
    for (int t = 0; t < N_TIMESTEPS; t++)
    {
      D[n * N_TIMESTEPS + t] = sample_demand(t, n, Grid_Model);
      G[n * N_TIMESTEPS + t] = sample_renewable_generation(t,
Grid_Model);
    }
  }
  // State Update
  for (int n = 0; n < N_ITERATIONS; n++)
```

Ritvik Gupta

```
  {
    for (int t = 1; t < N_TIMESTEPS; t++)
    {
      for (int i = 0; i < N_COMPONENTS; i++)
      {
        if (strcmp(Ci[n * N_TIMESTEPS + t - 1][i], "operational") == 0
&& ((float)rand() / RAND_MAX) < P_fail[i])
        {
          strcpy(Ci[n * N_TIMESTEPS + t][i], "failed");
          Oi[n * N_TIMESTEPS + t][i] = 1;
          Grid_Model[i].outage_duration[0] = 1;
        }
        else
        {
          strcpy(Ci[n * N_TIMESTEPS + t][i], Ci[n * N_TIMESTEPS + t -
1][i]);
          Oi[n * N_TIMESTEPS + t][i] = 0;
        }
      }
    }
  }

  // Demand-Supply Check
```

Ritvik Gupta

```
  void (*Resilience_Strategies[])() = {&resilience_strategy_1,
&resilience_strategy_2, &resilience_strategy_3};

  for (int n = 0; n < N_ITERATIONS; n++)

  {

    for (int t = 0; t < N_TIMESTEPS; t++)

    {

      Imbalance[n * N_TIMESTEPS + t][0] = D[n * N_TIMESTEPS + t] -
G[n * N_TIMESTEPS + t];

      if (Imbalance[n * N_TIMESTEPS + t][0] > imbalance_threshold)

      {

        // Applying resilience strategy

        // Resilience_Strategies[1](Grid_Model);

        //Resilience_Strategies[0](Grid_Model);

        Resilience_Strategies[2](Grid_Model);

      }

    }

  }


  // Resilience Metric Calculation

  for (int n = 0; n < N_ITERATIONS; n++)

  {

    for (int i = 0; i < N_COMPONENTS; i++)

    {
```

Ritvik Gupta

```
            Rn[n] += calculate_resilience_metric(Grid_Model);
        }
    }
    // Result
    printf("Simulation Results:\n\n");
    printf("Resilience Metric: %f\n", Rn[0] / N_ITERATIONS);
    return 0;
}
```

Ritvik Gupta