Developing Optimization Algorithms for Global Grid using Python based GBOML

Final Year Project report submitted to

Visvesvaraya National Institute of Technology, Nagpur

in partial fulfillment of the requirements for the award of the degree

Bachelor of Technology in Electrical and Electronics Engineering

By

Pranjali Kulkarni (BT20EEE087)

Tarun Gangadhar (BT20EEE107)

Under the guidance of

Dr. Sunil Bhat

Department of Electrical Engineering Visvesvaraya National Institute of TechnologyNagpur 440 010 (India)

May 2024

Developing Optimization Algorithms for Global Grid using Python based GBOML

Project report submitted to

Visvesvaraya National Institute of Technology, Nagpur

in partial fulfillment of the requirements for the award of the degree

Bachelor of

Technology in

Electrical and Electronics Engineering

By

Pranjali Kulkarni (BT20EEE087)

Tarun Gangadhar (BT20EEE107)

Under the guidance of

Dr. Sunil Bhat

Department of Electrical Engineering Visvesvaraya National Institute of TechnologyNagpur 440 010 (India)

May 2024

DEPARTMENT OF ELECTRICAL ENGINEERING

VISVESVARAYA NATIONAL INSTITUTE OF TECHNOLOGY

NAGPUR, MAHARASHTRA

2023-2024

Declaration

We, **Pranjali Kulkarni and Tarun Gangadhar** hereby declare that this project work titled "**Developing Optimisation Algorithms for Global Grid using Python based GBOML**" is carried out by us in the Department of Electrical Engineering of Visvesvaraya National Institute of Technology, Nagpur. This work is original and has not been submitted earlier whole or in part for the award of any degree/diploma at this or any other Institution/University.

Sr. No.	Enrollment No	Names	Signature
1	BT20EEE087	Pranjali Kulkarni	
2	BT20EEE107	Tarun Gangadhar	

Date: 26-04-2024

Certificate

This is to certify that the project titled ”**Developing Optimisation Algorithms for Global Grid using Python based GBOML**”, submitted by **Pranjali Kulkarni** and **Tarun Gangadhar** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology in Electrical and Electronics Engineering**,VNIT Nagpur. The work is comprehensive, complete and fit for final evaluation.

Dr. Sunil Bhat
Prof,Dept of Electrical
Engineering,VNIT, Nagpur

Dr. B. S. Umre
Head, Dept. of Electrical
EngineeringVNIT,
Nagpur

Date: 26-04-2024

ACKNOWLEDGMENTS

We extend our heartfelt gratitude to Dr. Sunil Bhat for his profound insights and invaluable guidance, which have significantly shaped this thesis. His unwavering support and mentorship have been instrumental throughout the journey of this project. We learnt a lot from his knowledge and experience.

We are deeply thankful to Dr. Arghya Mitra and Dr. Soumyabrata Das for their valuable suggestions and constructive feedback provided during the progress seminars. Their inputs have immensely enriched the quality of this work.

Special appreciation goes to Mr. Bimal Dora, a PhD scholar, for his gracious assistance and technical support, which have been indispensable in the execution of this project.

We would like to express our gratitude to all the esteemed faculties of the Electrical and Electronics Department for their continuous encouragement and support.

We are profoundly grateful to our parents for their unwavering encouragement and understanding throughout this journey.

Last but not the least, we thank the Almighty for giving us the strength and ability to face every challenge along this endeavor.

Pranjali Kulkarni
Tarun Gangadhar

ABSTRACT

Optimization algorithms play a pivotal role in modern society, addressing the ever-growing complexity of problems across various domains. In contemporary contexts, optimization algorithms serve as indispensable tools in numerous fields, ranging from engineering and finance to healthcare and logistics. These algorithms enable organizations to enhance efficiency, reduce costs, and improve decision-making processes by finding optimal solutions to complex problems.

Optimization algorithms find extensive applications in power systems, where they are used to optimize generation, transmission, and distribution of electrical energy. Specifically, for developing a global grid, optimization algorithms can be employed to efficiently allocate resources, plan interconnections, and manage energy flows across continents. By leveraging optimization techniques, such as linear programming and metaheuristic algorithms, planners can minimize costs, enhance reliability, and promote sustainability in the global energy network.

Python has emerged as a preferred programming language for optimization due to its versatility, ease of use, and extensive library support. Libraries such as NumPy, SciPy, and PyTorch provide robust implementations of optimization algorithms, facilitating rapid prototyping and deployment of solutions across various domains.

The Graph-Based Optimization Modeling Language (GBOML) is a modeling language for mathematical programming designed and implemented at the University of Liège, Belgium. GBOML enables the easy implementation of a broad class of structured mixed-integer linear programs typically found in applications ranging from energy system planning to supply chain management.

The Graph-Based Optimization Modeling Language (GBOML) offers a powerful framework for developing a global grid. Using GBOML, planners can model the intricate network of power generation, transmission lines, and interconnections between regions.

By formulating optimization problems in GBOML and Python, planners can explore various scenarios, consider uncertainties, and optimize the performance of the global grid under different operating conditions.

ABBREVIATIONS

GBOML	Graph Based Optimisation Model Language
AI	Artificial Intelligence
ML	Machine Learning
LP	Linear Programming
NLP	Nonlinear Programming
IP	Integer Programming
MILP	Mixed Integer Linear Programming
MINLP	Mixed Integer Nonlinear Programming
EMS	Energy Management Systems
DERMS	Distributed Energy Resource Management System
ESS	Energy Storage Systems
PV	Photovoltaic
HVDC	High Voltage Direct Current
TPE	Tree-structured Parzen Estimator
CMA-ES	Covariance Matrix Adaptation Evolution Strategy
ES6	ECMAScript 2015

List of Figures

List of Tables

INDEX

CHAPTER 1

INTRODUCTION

1.1 Overview

A global grid represents a vast interconnected network of electricity transmission lines spanning continents, enabling the seamless exchange of power between regions and countries. This concept aims to optimize the utilization of renewable energy resources, enhance energy security, and foster global economic cooperation. While regional and continental grids exist, the realization of a fully integrated global grid remains a vision for the future.

Currently, progress towards a global grid is evident in the establishment of interregional power interconnections, particularly in regions like Europe and Asia. Initiatives such as the European Union's Ten-Year Network Development Plans and the Belt and Road Initiative in Asia have catalysed investments in cross-border electricity infrastructure, aiming to bolster energy security and promote renewable energy integration.

However, numerous challenges impede the development of a global grid, including regulatory hurdles, differing technical standards, and geopolitical complexities. Overcoming these obstacles requires robust governance mechanisms and international cooperation frameworks.

Despite challenges, technological advancements and global policy initiatives provide promising pathways towards realizing the vision of a global grid. Innovations in transmission technology, smart grid solutions, and energy storage systems, combined with concerted international efforts, offer opportunities to address energy challenges and foster sustainable development on a global scale[1].

1.2 Motivation for studying Global Grid and GBOML

The global energy landscape stands at a pivotal juncture, marked by the urgent need to address pressing challenges such as the transition towards renewable energy, enhancing energy security, and mitigating climate change impacts. At the heart of these challenges lies the global grid, a vast interconnected network of electricity transmission lines spanning continents. Understanding the intricacies of the global grid, optimizing its operation, and leveraging tools like the Graph-Based Optimization Modeling Language (GBOML) are essential endeavors in tackling these multifaceted energy challenges.

Through a comprehensive examination of the following factors, we aim to underscore the significance of studying the global grid and its optimization in shaping a sustainable and resilient global energy future.

Addressing Global Energy Challenges:

The study of the global grid, its optimization, and the utilization of tools like GBOML are motivated by the urgent need to address pressing energy challenges on a global scale. These challenges include the transition towards renewable energy sources, enhancing energy security, and mitigating climate change impacts.

Efficient Utilization of Renewable Energy:

A comprehensive understanding of the global grid is crucial for maximizing the efficient utilization of renewable energy resources. By studying the global grid, researchers can identify optimal locations for renewable energy deployment, assess the feasibility of interconnecting renewable energy-rich regions with demand centers, and optimize energy flows to minimize curtailment and maximize utilization.

Enhancing Energy Security:

The development of a robust global grid infrastructure is essential for enhancing energy security by diversifying energy sources and reducing dependency on fossil fuels. By studying the global grid, researchers can identify vulnerabilities in the existing infrastructure, propose resilient design solutions, and develop contingency plans to mitigate potential disruptions in energy supply.

Mitigating Climate Chane Impacts:

The optimization of the global grid plays a crucial role in mitigating climate change impacts by facilitating the integration of renewable energy sources and reducing greenhouse gas emissions. Through advanced modeling and optimization techniques, researchers can design efficient energy systems that prioritize low-carbon energy sources, minimize transmission losses, and promote energy conservation.

Overcoming Regulatory and Technical Challenges:

The study of the global grid and its optimization involves addressing regulatory barriers, technical challenges, and investment constraints that hinder its development. By analyzing regulatory frameworks, researchers can identify policy levers to incentivize grid modernization and facilitate cross-border energy trade. Additionally, through the use of advanced modeling tools like GBOML, researchers can simulate various scenarios, optimize system configurations, and assess the economic viability of grid expansion projects.

Contributing to the Global Sustainability Goals:

Ultimately, studying the global grid and its optimization, alongside leveraging tools like GBOML, offers an opportunity to contribute to global sustainability goals. By developing innovative solutions to enhance the resilience, efficiency, and sustainability of the global energy infrastructure, researchers can play a pivotal role in advancing towards a more sustainable and interconnected energy future.

1.3 Aim and Objectives

The primary aims and objectives for our project in Phase I and Phase II are as follows:

1. Explore optimization principles in-depth to understand fundamental concepts and methodologies.
2. Gain proficiency in implementing diverse optimization algorithms using Python programming language.
3. Develop a comprehensive understanding of GBOML, delving into its features and intricacies for modelling optimization problems.
4. Acquire knowledge about microgrids, including their design, operation, and integration into existing grid systems.
5. Analyse the current grid scenario worldwide, examining challenges, trends, and initiatives towards developing a global grid.
6. Utilize Python and GBOML to construct an interconnected microgrid model aimed at achieving power balance and system optimization.
7. Extend the interconnected microgrid model to replicate and optimize a global grid scenario, leveraging advanced optimization techniques and scalable modelling approaches.

CHAPTER 2

OPTIMISATION ALGORITHMS

2.1 What is optimisation?

Optimization is a mathematical approach used to find the best possible solution to a problem among a set of feasible solutions. It is a fundamental concept that underpins decision-making processes across various disciplines, including engineering, economics, operations research, and computer science. At its core, optimization involves identifying the most efficient allocation of resources, maximizing benefits, or minimizing costs while adhering to constraints and objectives[2].

The process of optimization typically involves the following key components:

Objective Function:

This is the mathematical expression that quantifies the goal or objective of the optimization problem. It could be maximizing profit, minimizing cost, optimizing efficiency, or achieving any other desired outcome.

Decision Variables:

These are the variables that the optimizer can control or manipulate to achieve the desired objective. Decision variables represent the choices or actions available to the decision-maker.

Constraints:

Constraints are restrictions or limitations that must be satisfied for a solution to be considered feasible. Constraints can be related to physical limitations, resource availability, legal requirements, or any other factors that impose restrictions on the decision variables.

Types of Constraints:

1. **Equality Constraints**: These constraints require that the value of an expression be exactly equal to a specified value.
2. **Inequality Constraints**: These constraints specify that an expression must be either greater than or equal to, or less than or equal to, a certain value.

3. **Bound Constraints**: These constraints limit the range of values that decision variables can take. They are often used to ensure that variables stay within physically meaningful or legally required limits, such as non-negativity constraints.
4. **Integer Constraints**: These specify that some or all of the decision variables must take integer values. This is common in problems where the decision variables represent discrete choices or quantities, such as the number of items to manufacture or vehicles to dispatch.
5. **Binary Constraints**: A special case of integer constraints where the variables can only take values of 0 or 1. These are used in decision-making problems to indicate whether an option is chosen or not.
6. **Nonlinear Constraints**: These involve nonlinear relationships among the decision variables. They can be more challenging to handle because they may result in non-convex feasible regions, where local optimization methods might not find the global optimum.
7. **Logical Constraints**: These enforce logical conditions on the decision variables, often used in conjunction with binary variables to model logical implications, conjunctions, and disjunctions in decision-making.
8. **Dynamic Constraints**: Used in problems where the decision-making process is extended over time (dynamic systems), such as in the scheduling of operations or the management of inventory. These constraints ensure the consistency and feasibility of solutions across different time periods.

Feasible Region:

The feasible region is the set of all possible values of decision variables that satisfy the constraints. Solutions lying within this region are considered feasible solutions.

Optimal Solution: The optimal solution is the best possible solution to the optimization problem, according to the defined objective function. It represents the combination of decision variable values that maximizes or minimizes the objective function while satisfying all constraints.

Optimization problems can be classified based on various criteria, including the nature of the objective function (maximization or minimization), the type of decision variables (continuous or discrete), and the presence of constraints (constrained or unconstrained).

There are several approaches to solving optimization problems, ranging from analytical methods to numerical algorithms. Analytical methods involve finding closed-form solutions using mathematical techniques such as calculus and linear algebra. However, many real-world optimization problems are complex and cannot be solved analytically. In such cases, numerical optimization algorithms are employed to iteratively search for the optimal solution within the feasible region.

Common numerical optimization algorithms include gradient-based methods (such as gradient descent and Newton's method), evolutionary algorithms (such as genetic algorithms and particle swarm optimization), and metaheuristic techniques (such as simulated annealing and tabu search). These algorithms differ in their search strategies, convergence properties, and suitability for different types of optimization problems.

Overall, optimization is a powerful tool for decision-making and problem-solving in a wide range of applications. By formulating problems mathematically and employing appropriate optimization techniques, practitioners can find efficient solutions to complex real-world challenges, leading to improved outcomes and resource utilization.

Table 2.1: Different Optimisation Algorithms

Optimisation Algorithms	**Example**	**Application**
Gradient-Based Methods	Gradient descent is an iterative optimization algorithm used to minimize a differentiable objective function. It updates the parameters (decision variables) in the direction of the negative gradient of the objective function to reach the minimum.	Gradient descent is widely used in machine learning for training models, such as linear regression and neural networks.

Metaheuristic Algorithms	Genetic Algorithm: Genetic algorithms are inspired by the process of natural selection and evolution. They maintain a population of candidate solutions (chromosomes), iteratively selecting, recombining, and mutating them to generate new offspring. The fittest solutions survive and propagate to the next generation.	Genetic algorithms are used in optimization problems where traditional methods struggle, such as combinatorial optimization, scheduling, and design optimization.
Linear Programming Algorithms	The simplex method is an iterative algorithm used to solve linear programming (LP) problems. It starts at a feasible vertex of the feasible region and moves along the edges of the polytope to find the optimal vertex that minimizes or maximizes the objective function.	The simplex method is applied in various domains, including production planning, transportation logistics, and resource allocation.
Integer Programming Algorithms	Branch-and-bound is a systematic search algorithm used to solve mixed integer programming (MIP) problems. It decomposes the problem into smaller subproblems by branching on integer variables, exploring the solution space efficiently while pruning branches that cannot lead to better solutions than the current best.	Branch-and-bound is applied in scheduling problems, facility location, and network design where decisions involve discrete choices.
Dynamic Programming	The Bellman-Ford algorithm is a dynamic programming algorithm used to find the	The Bellman-Ford algorithm is applied in network

Algorithms	shortest paths from a single source vertex to all other vertices in a weighted graph with negative edge weights. It iteratively relaxes the edges in the graph to update the shortest path distances.	routing, distance-vector routing protocols, and network optimization.
Stochastic Optimization Algorithms	Simulated annealing is a probabilistic optimization algorithm inspired by the annealing process in metallurgy. It explores the solution space by probabilistically accepting worse solutions with a decreasing probability over time, allowing the algorithm to escape local optima and converge towards global optima.	Simulated annealing is used in combinatorial optimization, traveling salesman problems, and parameter optimization in machine learning.
Convex Optimization Algorithms	Interior-Point Method: Interior-point methods are optimization algorithms used to solve convex optimization problems efficiently. They iteratively move towards the interior of the feasible region, converging to the optimal solution by solving a sequence of barrier subproblems.	Interior-point methods are applied in linear programming, quadratic programming, and semidefinite programming problems in engineering, economics, and control theory.

2.2 Types of Optimisation Problems

Some of the most important optimisation problems and their significance is discussed below:

Linear Programming (LP):

LP models often involve problems with a single, well-defined objective function and linear constraints. They are characterized by convex feasible regions and efficiently solvable using algorithms such as the Simplex method or interior-point methods.

LP problems can be extended to handle uncertainty through techniques like robust optimization or stochastic programming, ensuring resilience against unforeseen variations in parameters.
Extensions of LP include network flow problems, such as the maximum flow or minimum cost flow problems, which have applications in transportation, telecommunications, and supply chain management [3].

Nonlinear Programming (NLP):
NLP problems encompass a wide range of mathematical optimization problems where the objective function or constraints are nonlinear. These problems often arise in engineering design, finance, and biology, among other fields.
Solving NLP problems typically requires specialized algorithms capable of handling non-convex functions and constraints, such as gradient-based methods, genetic algorithms, or interior-point methods for convex relaxation.
Nonlinear optimization techniques can be augmented with sensitivity analysis to assess the impact of parameter variations on the optimal solution, providing valuable insights into the robustness of the solution.

Integer Programming (IP):
IP problems are classified into pure integer programming (where all decision variables must be integers) and mixed integer programming (where some variables are integers and others are continuous).
Techniques for solving IP problems include branch-and-bound, branch-and-cut, and cutting-plane methods, which iteratively explore the solution space to identify optimal integer solutions.
IP formulations can be strengthened using techniques like integer cuts or Lagrangian relaxation, improving the quality of the solutions and reducing computational effort.

Mixed Integer Linear Programming (MILP):
MILP problems involve a combination of discrete (integer) and continuous decision variables, allowing for more flexible modelling of real-world optimization problems.
MILP formulations are widely used in diverse applications, including production planning, facility location, network design, and scheduling, where decisions involve both discrete

choices and continuous quantities.

Techniques for solving MILP problems include branch-and-bound, branch-and-cut, and integer programming relaxations, which decompose the problem into smaller subproblems and explore the solution space efficiently.

Extensions of MILP include mixed integer quadratic programming (MIQP), mixed integer quadratically constrained programming (MIQCP), and mixed integer bilevel programming (MIBLP), which further generalize the problem class to handle quadratic objectives or constraints.

Mixed Integer Nonlinear Programming (MINLP):

MINLP problems involve a combination of discrete (integer) and continuous decision variables, along with nonlinear objective functions or constraints, adding complexity compared to MILP.

MINLP formulations are used in various domains, including process optimization, chemical engineering, logistics planning, and finance, where decisions involve both discrete choices and nonlinear relationships.

Solving MINLP problems requires specialized algorithms capable of handling both discrete and continuous variables while addressing nonlinearities in the objective function or constraints.

Techniques for solving MINLP problems include branch-and-bound, outer approximation, and spatial branch-and-bound methods, which iteratively explore the solution space and refine the solution until an optimal solution is found.

MINLP problems often require a careful balance between computational efficiency and solution quality, as solving them can be computationally demanding due to the nonlinearity and combinatorial nature of the problem class.

Combinatorial Optimisation:

Combinatorial optimization problems involve finding the best arrangement or selection of discrete elements from a finite set of alternatives. They include problems like the traveling salesman problem, knapsack problem, and graph colouring problem.

Exact solution methods for combinatorial optimization problems include dynamic programming, branch-and-bound, and branch-and-cut algorithms, while metaheuristic techniques like genetic algorithms and simulated annealing provide efficient approximate solutions.

Combinatorial optimization problems often have applications in network design, telecommunications, computer science, and logistics, where discrete decision-making is prevalent.

Metaheuristic Optimisation:

Metaheuristic optimization algorithms are heuristic approaches that guide the search for optimal solutions through exploration and exploitation of the solution space.

These algorithms are characterized by their flexibility, robustness, and ability to handle complex, non-convex optimization landscapes, making them suitable for a wide range of optimization problems.

Common metaheuristic algorithms include genetic algorithms, particle swarm optimization, ant colony optimization, and simulated annealing, each offering unique strategies for exploration and exploitation of the search space.

Stochastic Optimisation:

Stochastic optimization deals with optimizing under uncertain conditions, where parameters or variables are subject to randomness or variability.

Techniques for solving stochastic optimization problems include stochastic gradient descent, Monte Carlo simulation, and scenario-based optimization, which account for uncertainty through probabilistic models or scenario analysis.

Stochastic optimization finds applications in finance (portfolio optimization), operations research (inventory management), and engineering (reliability-based design), where decision-making must account for uncertain future events or outcomes.

Dynamic Programming:

Dynamic programming is a method for solving optimization problems by breaking them down into simpler sub problems and solving them iteratively.

This approach is particularly useful for problems with overlapping substructures, such as shortest path problems, knapsack problems, and optimal control problems.

Dynamic programming algorithms, such as the Bellman-Ford algorithm, Floyd-Warshall algorithm, and the Viterbi algorithm, provide efficient solutions for problems with recursive or overlapping sub problems.

2.3 Advantages of Optimisation

Incorporating optimization techniques using optimization algorithms offers numerous benefits across various domains. Here's a detailed explanation of the advantages:

Efficiency and Cost Reduction:

Optimization algorithms enable organizations to streamline processes and allocate resources more efficiently, leading to cost savings and improved productivity.

By optimizing resource allocation, production scheduling, and logistics planning, companies can minimize operational costs while maximizing output and revenue.

Improved Decision Making:

Optimization techniques provide valuable insights into complex decision-making processes by identifying optimal solutions among a multitude of choices.

Decision-makers can use optimization models to evaluate different scenarios, assess trade-offs, and make informed decisions based on quantitative data rather than intuition or guesswork.

Resource Utilisation Optimisation:

Optimization algorithms help in optimizing the utilization of resources such as manpower, machinery, and raw materials.

By efficiently allocating resources based on demand forecasts, inventory levels, and production schedules, organizations can minimize waste and ensure optimal resource utilization.

Enhanced Performance and Productivity:

Optimization techniques optimize processes, workflows, and systems to enhance overall performance and productivity.

By identifying bottlenecks, inefficiencies, and constraints, organizations can redesign workflows and implement process improvements to boost productivity and output.

Competitive Advantage:

Organizations that leverage optimization algorithms gain a competitive edge by optimizing their operations, reducing costs, and delivering superior products or services to customers.

Optimization enables companies to respond quickly to changing market dynamics, customer demands, and competitive pressures, maintaining their competitive position in the market.

Risk Mitigation and Uncertainty Management:

Optimization models help in assessing and mitigating risks by identifying potential vulnerabilities, uncertainties, and worst-case scenarios.

Decision-makers can use optimization techniques to develop robust strategies, contingency plans, and risk management measures to mitigate the impact of unforeseen events and disruptions.

Sustainability and Environmental Impact Reduction:

Optimization algorithms support sustainable practices by optimizing resource utilization, minimizing waste, and reducing environmental impact.

Organizations can use optimization techniques to design eco-friendly supply chains, reduce carbon emissions, and adopt sustainable production practices, contributing to environmental conservation and sustainability goals.

Scalability and Adaptability:

Optimization models and algorithms are scalable and adaptable to various problem sizes, complexities, and domains.

Whether it's optimizing a small-scale production process or designing a large-scale logistics network, optimization techniques can be tailored to suit specific requirements and constraints, ensuring scalability and adaptability.

Innovation and Creativity Enhancement:

Optimization algorithms foster innovation and creativity by providing new perspectives, insights, and solutions to complex problems.

By exploring different optimization approaches, experimenting with alternative strategies, and challenging conventional thinking, organizations can drive innovation and creativity in their operations and processes.

Incorporating optimization techniques using optimization algorithms offers multifaceted benefits, including cost reduction, improved decision-making, resource optimization, enhanced performance, competitive advantage, risk mitigation, sustainability, scalability, adaptability, and innovation. By leveraging optimization algorithms effectively, organizations can achieve operational excellence, drive growth, and stay ahead in today's dynamic and competitive business environment.

2.4 Python in Optimisation

Python has emerged as a powerful language for optimization, offering a rich ecosystem of libraries and tools that facilitate the implementation and solving of various optimization problems. With its simple syntax, extensive libraries, and strong community support, Python has become a preferred choice for researchers, engineers, and data scientists working on optimization tasks. Python-based libraries such as SciPy, NumPy, CVXPY, PuLP, Pyomo, and Optuna provide a wide range of optimization algorithms, modeling languages, and tools for solving linear, nonlinear, convex, and combinatorial optimization problems efficiently. These libraries offer a combination of flexibility, scalability, and ease of use, making them suitable for a diverse range of applications in scientific computing, engineering design, finance, operations research, machine learning, and beyond. In this dynamic landscape, Python continues to play a pivotal role in advancing optimization techniques and enabling practitioners to tackle complex real-world challenges effectively. Several powerful libraries for optimization that provide a wide range of algorithms and tools to solve various optimization problems efficiently[4]. Here are some popular Python-based libraries for optimization:

Table 2.2: Popular Python-based libraries for optimization

Library	**Description**	**Features**	**Usage**
SciPy	SciPy is a comprehensive scientific computing library that includes modules for optimization, interpolation, integration, and much more.	SciPy's optimize module provides a variety of optimization algorithms, including nonlinear least-squares minimization, unconstrained and constrained minimization of multivariate scalar	It's widely used for scientific computing, engineering, data analysis, and machine learning applications.

		functions, global optimization, and linear programming.	
NumPy	NumPy is the fundamental package for scientific computing in Python, providing support for arrays, matrices, and mathematical functions.	NumPy's linear algebra module (numpy.linalg) includes functions for solving linear systems of equations, eigenvalue problems, and matrix factorizations, which are essential for many optimization algorithms.	While not primarily an optimization library, NumPy is a fundamental building block for optimization algorithms implemented in higher-level libraries.
CVXPY	CVXPY is a Python-embedded modeling language for convex optimization problems.	CVXPY allows users to formulate convex optimization problems in a natural syntax and solve them using a variety of solvers, including open-source solvers like ECOS and commercial solvers like Gurobi and MOSEK.	It's widely used in finance, engineering, control systems, and machine learning for solving convex optimization problems efficiently.

PuLP	PuLP is a linear programming modeling language in Python.	PuLP provides a high-level interface to optimization solvers, making it easy to formulate linear programming problems and solve them using a variety of open-source and commercial solvers.	It's commonly used in operations research, supply chain management, and logistics for modeling and solving linear programming problems.
Pyomo	Pyomo is a Python-based open-source optimization modeling language for linear, nonlinear, and mixed-integer optimization problems.	Pyomo allows users to define optimization models using a high-level algebraic syntax and solve them using a variety of optimization solvers, including open-source and commercial solvers.	It's widely used in research, academia, and industry for modeling and solving complex optimization problems efficiently.
Optuna	Optuna is an automatic hyperparameter optimization framework for machine learning algorithms.	Optuna provides a lightweight, flexible, and scalable framework for hyperparameter optimization, supporting various optimization algorithms, including TPE (Tree-structured	It's commonly used in machine learning and data science projects to optimize hyperparameters and improve model performance.

		Parzen Estimator) and CMA-ES (Covariance Matrix Adaptation Evolution Strategy).	

These libraries offer a rich set of tools and algorithms for solving optimization problems in Python, catering to a wide range of applications and use cases in scientific computing, engineering, finance, operations research, and machine learning.

2.5 Mathematical Optimisation

Following are some primary examples of mathematical optimisation problems along with their optimisation code:

A. EXAMPLE 1:

$$
\begin{aligned}
&Minimize \quad 80x + 95y \\
&\qquad\qquad s.t. \\
&10x + 15y >= 100 \\
&20x + 15y >= 160 \\
&x, y >= 0
\end{aligned}
$$

1. PYOMO:

```
!pip install pyomo
!pip install gurobipy
from pyomo.environ import *

model = ConcreteModel()
```

```
model.x = Var(within=NonNegativeReals)
model.y = Var(within=NonNegativeReals)

model.obj = Objective(expr=80 * model.x + 95 * model.y,
sense=minimize)

model.constr1 = Constraint(expr=10 * model.x + 15 * model.y
== 100)
model.constr2 = Constraint(expr=20 * model.x + 15 * model.y
== 160)

opt = SolverFactory('gurobi')
results = opt.solve(model)

model.display()
```

Output:

```
Model unknown

  Variables:
    x : Size=1, Index=None
        Key  : Lower : Value : Upper : Fixed : Stale :
Domain
        None :     0 :   6.0 :  None : False : False :
NonNegativeReals
    y : Size=1, Index=None
        Key  : Lower : Value              : Upper : Fixed :
Stale : Domain
        None :     0 : 2.6666666666666665 :  None : False :
False : NonNegativeReals
```

```
  Objectives:
    obj : Size=1, Index=None, Active=True
        Key  : Active : Value
        None :   True : 733.3333333333333

  Constraints:
    constr1 : Size=1
        Key  : Lower : Body  : Upper
        None : 100.0 : 100.0 : 100.0
    constr2 : Size=1
        Key  : Lower : Body  : Upper
        None : 160.0 : 160.0 : 160.0
```

2. PULP:

```
! pip install pulp
import pulp

# Create a linear programming problem
lp_problem = pulp.LpProblem("Optimization Problem",
pulp.LpMinimize)

# Define decision variables
x = pulp.LpVariable("x", lowBound=0)
y = pulp.LpVariable("y", lowBound=0)

# Objective function to minimize
lp_problem += 80 * x + 95 * y, "Objective"

# Constraints
lp_problem += 10 * x + 15 * y >= 100
lp_problem += 20 * x + 15 * y >= 160
```

```
# Solve the problem
lp_problem.solve()

# Print the results
print("Optimal Solution:")
print(f"x = {x.varValue}")
print(f"y = {y.varValue}")
print(f"Optimal Objective Value =
{pulp.value(lp_problem.objective)}")
```

Output:

```
Optimal Solution:
x = 6.0
y = 2.6666667
Optimal Objective Value = 733.3333365
/usr/local/lib/python3.10/dist-packages/pulp/pulp.py:1352:
UserWarning: Spaces are not permitted in the name.
Converted to '_'
  warnings.warn("Spaces are not permitted in the name.
Converted to '_'")
```

3. GUROBI

```
!pip install gurobipy

import random
import gurobipy as gp
from gurobipy import GRB

# Create a Gurobi model
model = gp.Model()
```

```
# Parameters
a = 80
b = 95

# Variables
x = model.addVar(lb=0, vtype=GRB.CONTINUOUS, name="x")
y = model.addVar(lb=0, vtype=GRB.CONTINUOUS, name="y")

# Constraints
model.addConstr(10 * x + 15 * y >= 100)
model.addConstr(20 * x + 15 * y >= 160)

# Objective
model.setObjective(a * x + b * y, GRB.MINIMIZE)

# Optimize the model
model.optimize()

# Print the results
if model.status == GRB.OPTIMAL:
    print("Optimal solution found:")
    print(f"x = {x.x}")
    print(f"y = {y.x}")
    print(f"Objective value = {model.objVal}")
else:
    print("No solution found")
```

Output:

```
Gurobi Optimizer version 10.0.2 build v10.0.2rc0 (linux64)
```

```
CPU model: Intel(R) Xeon(R) CPU @ 2.20GHz, instruction set
[SSE2|AVX|AVX2]
Thread count: 1 physical cores, 2 logical processors, using
up to 2 threads

Optimize a model with 2 rows, 2 columns and 4 nonzeros
Model fingerprint: 0x725cb80c
Coefficient statistics:
  Matrix range     [1e+01, 2e+01]
  Objective range  [8e+01, 1e+02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+02, 2e+02]
Presolve time: 0.01s
Presolved: 2 rows, 2 columns, 4 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.
Time
       0    0.0000000e+00   3.250000e+01   0.000000e+00
0s
       2    7.3333333e+02   0.000000e+00   0.000000e+00
0s

Solved in 2 iterations and 0.04 seconds (0.00 work units)
Optimal objective  7.333333333e+02
Optimal solution found:
x = 6.0
y = 2.6666666666666665
Objective value = 733.3333333333333
```

B. **EXAMPLE 2:**

$$\textit{Minimize } f(x) = (x_0 - 3)^2 + (x_1 - 4)^2$$

$$\textit{subject to}$$

$$x_0^2 + x_1^2 <= 10$$

A. **PYOMO:**

```
pip install pyomo
!pip install gurobipy
from pyomo.environ import *

# Create a concrete Pyomo model
model = ConcreteModel()

# Define decision variables
model.x0 = Var(within=Reals)
model.x1 = Var(within=Reals)

# Define the objective function to minimize
model.obj = Objective(expr=(model.x0 - 3)**2 + (model.x1 -
4)**2)

# Define the constraint
model.con = Constraint(expr=model.x0**2 + model.x1**2 <=
10)

# Create an instance of the solver (e.g., Ipopt)
solver = SolverFactory('gurobi')

# Solve the optimization problem
```

```
results = solver.solve(model)

# Print the results
print("Solver Status:", results.solver.status)
print("Objective Value:", value(model.obj))
print("x0 =", value(model.x0))
print("x1 =", value(model.x1))
```

Output:

```
Solver Status: ok
Objective Value: 3.3772238383040087
x0 = 1.8973665242749505
x1 = 2.5298220323666576
```

B. SCIPY:

```
pip install scipy

import numpy as np
from scipy.optimize import minimize

# Define the objective function to minimize
def objective_function(x):
    return (x[0] - 3)**2 + (x[1] - 4)**2

# Define the constraint
def constraint(x):
    return x[0]**2 + x[1]**2 - 10

# Initial guess
x0 = [0, 0]
```

```
# Define the bounds for the variables
bounds = ((None, None), (None, None))  # No specific bounds
for x0 and x1

# Define the constraint as a dictionary
constraint_dict = {'type': 'ineq', 'fun': constraint}

# Solve the optimization problem
result = minimize(objective_function, x0,
constraints=constraint_dict, bounds=bounds)

# Check if the optimization was successful
if result.success:
    print("Optimization successful!")
    print("Solver Status: " + result.message)
    print("Objective Value:", result.fun)
    print("x0 =", result.x[0])
    print("x1 =", result.x[1])
else:
    print("Optimization failed. Solver Status: " +
result.message)
```

Output:

```
Optimization successful!
Solver Status: Optimization terminated successfully
Objective Value: 25.0
x0 = 6.0
x1 = 8.0
```

2.6 Uses of Optimisation Techniques in Electrical Engineering

Optimization techniques play a critical role in electrical engineering, addressing various challenges and improving system efficiency, performance, and safety. Here’s how optimization is used across different domains within electrical engineering:

1. **Power System Operations and Planning**:
    - **Load Dispatch**: Optimization helps in economic load dispatch, ensuring that electricity generation and distribution are managed cost-effectively while meeting the demand.
    - **Capacity Expansion**: Techniques like linear programming and mixed-integer programming are used to plan the expansion of power systems, determining when and where to build new generation facilities or transmission lines.
    - **Renewable Energy Integration**: Optimization is used to manage the variability and intermittency of renewable energy sources, such as wind and solar, integrating them efficiently into the power grid.
2. **Energy Management Systems (EMS)**:
    - **Demand Response**: Optimization algorithms are used to manage and modify the demand for power through incentives and pricing mechanisms, balancing the load and reducing peak demand pressures on the grid.
    - **Energy Storage Management**: Techniques like dynamic programming are employed to optimize the charging and discharging cycles of energy storage systems, maximizing their efficiency and lifespan.
3. **Grid Modernization and Smart Grids**:
    - **Network Reconfiguration**: Optimization methods help in the dynamic reconfiguration of power distribution networks to minimize losses, improve reliability, and maintain voltage levels.
    - **Microgrid Management**: In microgrids, optimization is crucial for the optimal scheduling of distributed energy resources (DERs) and for operating microgrids in both grid-connected and islanded modes.
4. **Electric Vehicle (EV) Integration**:
    - **Charging Station Placement**: Optimization techniques determine optimal locations for EV charging stations to maximize coverage and minimize

costs.

- **Charging Schedules**: Algorithms are used to manage the charging of electric vehicles, especially in fleet operations, to reduce energy costs and peak load impact on the grid.

5. **Transmission and Distribution**:
    - **Network Design and Upgrades**: Optimization is used for designing new transmission and distribution networks and for upgrading existing infrastructure to improve performance and reduce costs.
    - **Fault Detection and Response**: Optimization algorithms can help in the rapid detection and isolation of faults in the network, enhancing the reliability and safety of the power supply.
6. **Signal Processing**:
    - **Filter Design**: Optimization is integral in the design of digital and analog filters to meet specific performance criteria such as minimal error, specific frequency response, or stability.
    - **Image and Sound Processing**: Techniques are applied to enhance the quality of images and sound by optimizing various parameters involved in compression, enhancement, and feature extraction.
7. **System and Component Design**:
    - **Electrical Machine Design**: Optimizing the design of motors, generators, and transformers to improve efficiency, reduce costs, and enhance performance.
    - **Circuit Optimization**: Used in the design and integration of circuits to achieve desired performance while minimizing power consumption and component count.

CHAPTER 3

GRAPH BASED OPTIMISATION MODELING LANGUAGE

3.1 Introduction to GBOML

Optimization using the Graph-Based Optimization Modeling Language (GBOML) represents a sophisticated approach to tackling complex mathematical programming problems. Developed and refined at the University of Liège, Belgium, GBOML stands as a testament to the intersection of mathematical theory and practical application. At its core, GBOML serves as a versatile modeling language tailored to address a broad spectrum of optimization challenges, ranging from energy system planning to supply chain management[5].

GBOML's strength lies in its ability to seamlessly handle structured mixed-integer linear programs, a common format encountered in real-world optimization scenarios. This versatility extends to its capacity to represent problems involving discrete-time dynamical systems over finite time horizons. A distinguishing feature of GBOML is its hierarchical hypergraph-based approach, allowing for the encoding of complex problem structures in a manner that is both intuitive and efficient.

By amalgamating elements of algebraic and object-oriented modeling languages, GBOML facilitates problem encoding and model reuse. Moreover, this fusion streamline model generation, exposes problem structure to specialized solvers, and simplifies post-processing tasks. The GBOML parser, implemented in Python, further enhances its accessibility and usability by converting GBOML input files into hierarchical graph data structures, representing optimization models in a format conducive to computational analysis.
The associated toolset offers a versatile array of interfaces, including a command-line

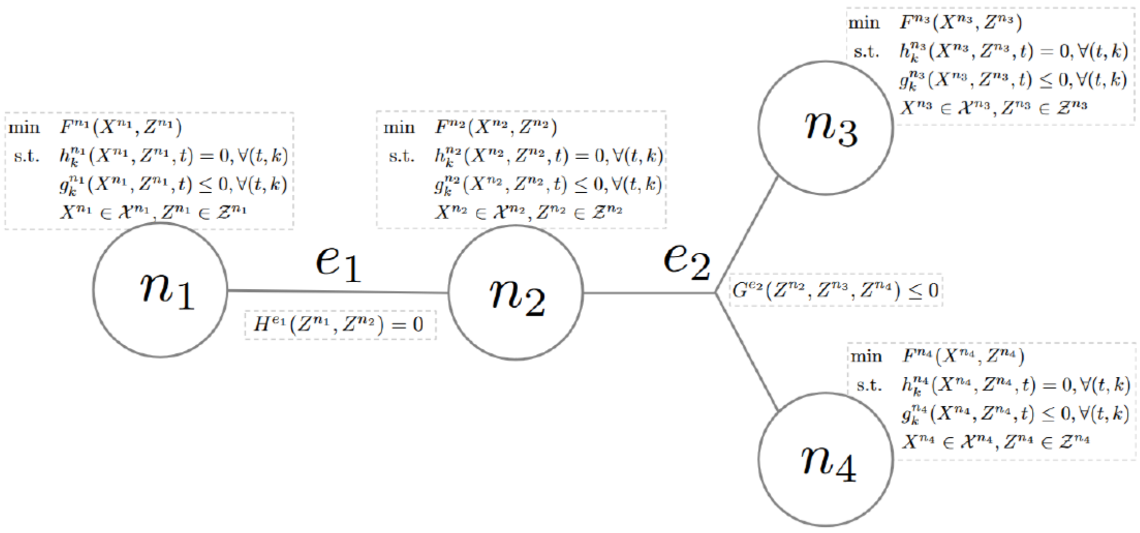
interface and a Python API, enabling users to construct and manipulate optimization models with ease. Moreover, GBOML seamlessly integrates with a diverse range of open-source and commercial solvers, including structure-exploiting ones, thereby harnessing the power of cutting-edge optimization algorithms to deliver robust and efficient solutions to complex optimization problems.

In essence, optimization using GBOML represents a sophisticated yet accessible approach to addressing real-world optimization challenges. Its combination of advanced modeling capabilities, streamlined workflow, and seamless integration with optimization solvers positions GBOML as a valuable tool for researchers, practitioners, and decision-makers seeking to optimize complex systems and processes across various domains.

3.2 Abstract GBOML Problem

The modeling language is particularly well-suited for representing problems involving the optimization of discrete time dynamical systems over a finite time horizon and exhibiting a natural block structure that may be encoded by a hierarchical hypergraph. A hypergraph abstraction is therefore employed to represent them. Nodes can therefore be viewed as hierarchical hypergraphs themselves representing optimization subproblems, while hyperedges express the
relationships between nodes. A global discretized time horizon and associated set of time periods common to all nodes are also defined. Each node is equipped with a set of so-called *internal* and *external* (or *coupling*) variables. A set of constraints may be also defined for each node, along with a local objective function representing its contribution to a system-wide objective. Finally, for each hyperedge, constraints involving the coupling variables of the nodes to which it is incident are defined in order to express the relationships between nodes. In the following paragraphs,
we formally define variables, constraints, objectives and formulate the abstract model that encapsulates the class of problems considered. For the sake of clarity, we do so for a model that can be represented by a hypergraph with a single *layer* (i.e., there is one level in the hierarchy).

$$
\begin{array}{ll}
\min & \sum_{n \in \mathcal{N}} F^n(X^n, Z^n) \\
\text{s.t.} & h_k^n(X^n, Z^n, t) = 0,\ \forall t \in \mathcal{T}_k^n,\ k = 1, \dots K^n,\ \forall n \in \mathcal{N} \\
& g_k^n(X^n, Z^n, t) \leq 0,\ \forall t \in \bar{\mathcal{T}}_k^n,\ k = 1, \dots \bar{K}^n,\ \forall n \in \mathcal{N} \\
& H^e(Z^e) = 0,\ \forall e \in \mathcal{E} \\
& G^e(Z^e) \leq 0,\ \forall e \in \mathcal{E} \\
& X^n \in \mathcal{X}^n, Z^n \in \mathcal{Z}^n,\ \forall n \in \mathcal{N}.
\end{array}
$$

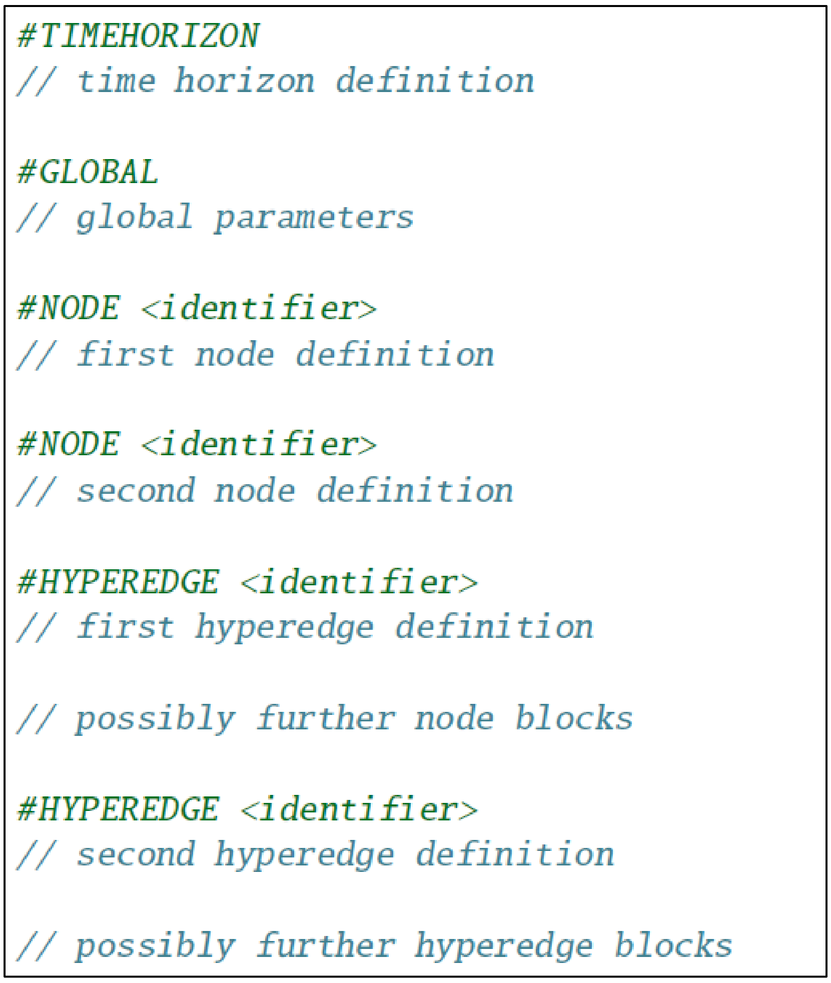


$\min\ F^{n_1}(X^{n_1}, Z^{n_1})$ s.t. $h_k^{n_1}(X^{n_1}, Z^{n_1}, t) = 0, \forall (t,k)$ $g_k^{n_1}(X^{n_1}, Z^{n_1}, t) \leq 0, \forall (t,k)$ $X^{n_1} \in \mathcal{X}^{n_1}, Z^{n_1} \in \mathcal{Z}^{n_1}$
$\min\ F^{n_2}(X^{n_2}, Z^{n_2})$ s.t. $h_k^{n_2}(X^{n_2}, Z^{n_2}, t) = 0, \forall (t,k)$ $g_k^{n_2}(X^{n_2}, Z^{n_2}, t) \leq 0, \forall (t,k)$ $X^{n_2} \in \mathcal{X}^{n_2}, Z^{n_2} \in \mathcal{Z}^{n_2}$
$\min\ F^{n_3}(X^{n_3}, Z^{n_3})$ s.t. $h_k^{n_3}(X^{n_3}, Z^{n_3}, t) = 0, \forall (t,k)$ $g_k^{n_3}(X^{n_3}, Z^{n_3}, t) \leq 0, \forall (t,k)$ $X^{n_3} \in \mathcal{X}^{n_3}, Z^{n_3} \in \mathcal{Z}^{n_3}$
n_3
n_1
e_1
n_2
e_2
$G^{e_2}(Z^{n_2}, Z^{n_3}, Z^{n_4}) \leq 0$
$H^{e_1}(Z^{n_1}, Z^{n_2}) = 0$
$\min\ F^{n_4}(X^{n_4}, Z^{n_4})$ s.t. $h_k^{n_4}(X^{n_4}, Z^{n_4}, t) = 0, \forall (t,k)$ $g_k^{n_4}(X^{n_4}, Z^{n_4}, t) \leq 0, \forall (t,k)$ $X^{n_4} \in \mathcal{X}^{n_4}, Z^{n_4} \in \mathcal{Z}^{n_4}$
n_4

Fig. 3.1 Schematic Representation of the abstract GBOML Problem

3.3 Block Definitions

To implement an instance of the abstract GBOML problem, the model must be encoded in an input file using the GBOML grammar. This input file is organized into blocks, each introduced by specific keywords: #TIMEHORIZON, #GLOBAL, #NODE, and #HYPEREDGE. The #TIMEHORIZON block provides information about the time horizon and must be the first block defined in the input file. Following that, global parameters are defined in the #GLOBAL block, which must come second. Nodes are defined in #NODE blocks, and hyperedges are defined in #HYPEREDGE blocks. Importantly, the order of appearance of #NODE and #HYPEREDGE blocks in the input file is irrelevant; hyperedge definitions may precede node definitions and vice versa. Therefore, the typical structure of an input file includes #TIMEHORIZON, #GLOBAL, #NODE, and #HYPEREDGE blocks, allowing for the systematic organization and encoding of the GBOML model.

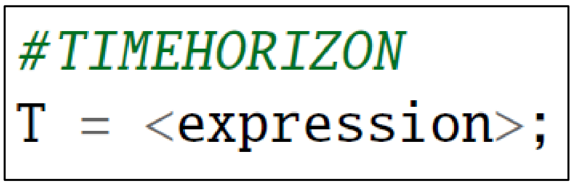


```
#TIMEHORIZON
// time horizon definition

#GLOBAL
// global parameters

#NODE <identifier>
// first node definition

#NODE <identifier>
// second node definition

#HYPEREDGE <identifier>
// first hyperedge definition

// possibly further node blocks

#HYPEREDGE <identifier>
// second hyperedge definition

// possibly further hyperedge blocks
```

Fig 3.2 Basic Structure of a GBOML Optimisation Code

Time Horizon

The time horizon, denoted as T, specifies the duration of the optimization horizon, representing the number of time periods under consideration. This definition is encapsulated within the #TIMEHORIZON block, which is typically the initial block in a GBOML file. The structure of this block is as follows:



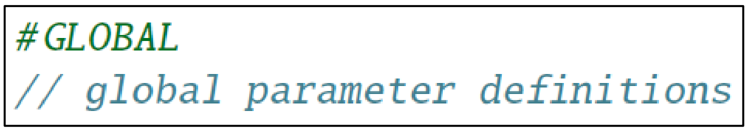
```
#TIMEHORIZON
T = <expression>;
```

Fig 3.3 Time Horizon Structure

Within the #TIMEHORIZON block, <expression> represents an algebraic expression intended to yield a positive integer value. Should <expression> yield a positive yet non-integral value, it will be automatically rounded to the nearest integer, accompanied by a warning. Expressions that fail to evaluate or result in a negative value are prohibited. Furthermore, as the #TIMEHORIZON block precedes all others in the input file and no parameters can be defined prior to it, <expression> cannot be dependent on any parameters.

Global Parameters

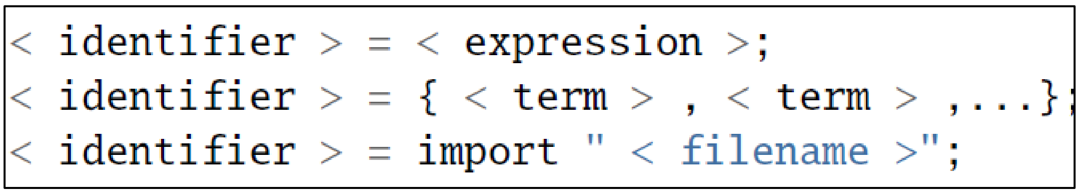
The non-mandatory #GLOBAL block contains the definitions of parameters that can be accessed anywhere in the model. This block is structured as follows:



```
#GLOBAL
// global parameter definitions
```

Fig 3.4 Global Parameters Structure

A parameter definition maps an identifier to a fixed value, which may be either a scalar or a vector. The identifier must be unique within a given #GLOBAL block and a value can be assigned to a parameter through one of the following three syntax rules:



```
< identifier > = < expression >;
< identifier > = { < term > , < term > ,...};
< identifier > = import " < filename >";
```

Fig 3.5 Identifier Structure

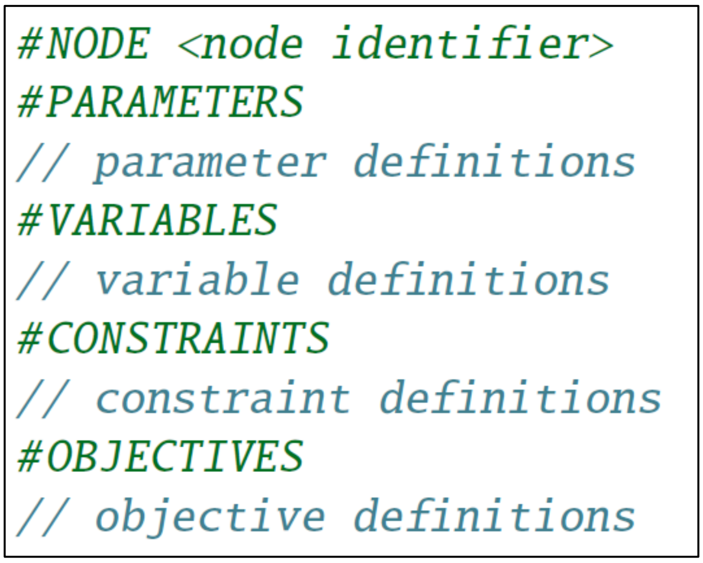
The process begins with defining a scalar parameter, adhering to the first rule. Within this definition, <expression> can be any scalar expression evaluating to a floating-point number. Notably, it may include references to previously defined global parameters. Upon successful validation of these rules, a parameter named <identifier> is instantiated and assigned the value of <expression>.

Subsequently, a vector parameter can be established directly, following the second rule. Here, each <term> within the definition represents a floating-point number, a previously defined scalar parameter, or an entry from a pre-existing vector parameter. This resultant vector parameter can then be accessed via indexing to retrieve individual entries. Alternatively, a vector parameter can be created by importing an input file, as outlined in the third rule. In this scenario, <filename> denotes the name of an input file formatted with delimiters such as comma, semicolon, space, or line feed. Unlike direct vector parameter definitions, the input file is restricted to containing only floating-point values and may not reference other parameters.

Nodes

In the hypergraph abstraction utilized in GBOML, nodes serve as representations of optimization subproblems. Consequently, each node possesses its distinct set of parameters and is endowed with a collection of variables, categorized into internal and external (or coupling) variables. Furthermore, for every node, a set of constraints can be delineated alongside a local objective function, reflecting its contribution to the overarching system-wide objective.

To delineate a #NODE block, it is imperative to assign a unique identifier. Subsequently, this block is subdivided into code blocks where parameters, variables, constraints, and objectives are defined. Each of these blocks is demarcated by one of the following keywords: #PARAMETERS, #VARIABLES, #CONSTRAINTS, and #OBJECTIVES. Thus, a typical #NODE block is structured as follows:



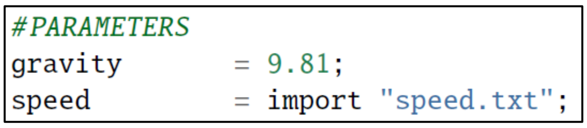
```
#NODE <node identifier>
#PARAMETERS
// parameter definitions
#VARIABLES
// variable definitions
#CONSTRAINTS
// constraint definitions
#OBJECTIVES
// objective definitions
```

Fig 3.6 Node Structure

Parameters

The parameters specified within a #NODE block adhere to the same guidelines as those outlined in the #GLOBAL block. Nevertheless, node parameters are confined to the respective node, and parameters defined in disparate nodes are inaccessible within this context.

To exemplify, consider the following valid #PARAMETERS block in GBOML:

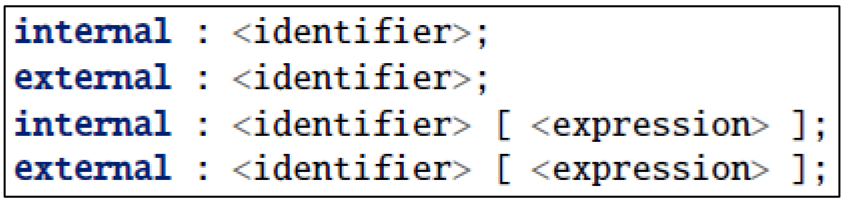


```
#PARAMETERS
gravity       = 9.81;
speed         = import "speed.txt";
```

Fig 3.7 Node's Parameters Structure

Variables

Variables are categorized using one of two keywords: internal or external. Internal variables are designated to model the internal state of a node, whereas external variables are intended to capture interactions between different nodes. In essence, the inter-node coupling is represented by imposing constraints on their external variables, a concept elaborated upon when introducing #HYPEREDGE blocks. Moreover, variables can denote either scalars or vectors. The syntax for declaring variables in GBOML is outlined as follows:

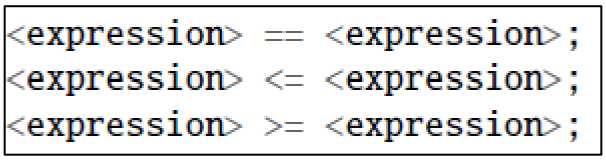


```
internal : <identifier>;
external : <identifier>;
internal : <identifier> [ <expression> ];
external : <identifier> [ <expression> ];
```

Fig 3.8 Node's Variables Structure

Constraints

The syntax rules for the definition of basic equality and inequality constraints are as follows:



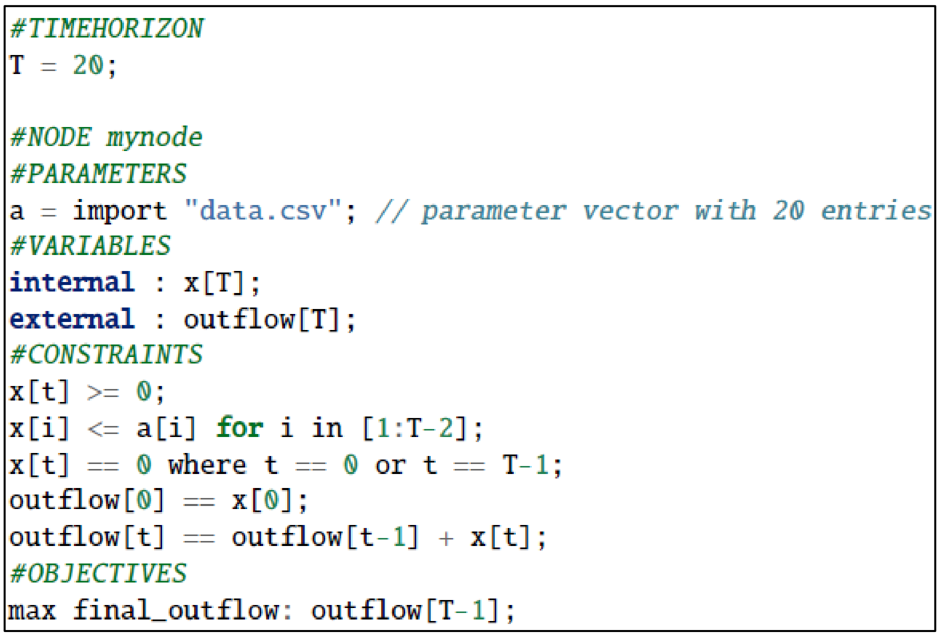
```
<expression> == <expression>;
<expression> <= <expression>;
<expression> >= <expression>;
```

Fig 3.9 Node's Constraints Structure

Within this context, both the left-hand side and the right-hand side of the constraints are formulated as general expressions, with the type of constraint determined by the comparison operator utilized. Additionally, in alignment with the principle that parameter and variable definitions are confined to individual nodes, constraints specified in a #NODE block should not make reference to quantities defined in alternative nodes.

```
<constraint identifier>: <constraint>;
```

The following is an example illustrating both expansion methods and making use of the keywords for and where in order to compactly write selectively-imposed constraints:



```
#TIMEHORIZON
T = 20;

#NODE mynode
#PARAMETERS
a = import "data.csv"; // parameter vector with 20 entries
#VARIABLES
internal : x[T];
external : outflow[T];
#CONSTRAINTS
x[t] >= 0;
x[i] <= a[i] for i in [1:T-2];
x[t] == 0 where t == 0 or t == T-1;
outflow[0] == x[0];
outflow[t] == outflow[t-1] + x[t];
#OBJECTIVES
max final_outflow: outflow[T-1];
```

Fig 3.10 Example illustrating expansion methods

Hyperedges

A hyperedge commonly links variables from distinct nodes through equality or inequality constraints, or both. Each hyperedge is delineated using a dedicated code block, which must commence with either the #HYPEREDGE keyword or the :#LINK keyword (both are interchangeable). Every hyperedge requires a unique <identifier>, ensuring that no two hyperedges, or a hyperedge and a node, share the same identifier. Additionally, a hyperedge may possess its parameters and constraints. Thus, valid hyperedge blocks adhere to the following structure:

```
#HYPEREDGE <identifier 1>
#PARAMETERS
// parameter definitions
#CONSTRAINTS
// constraint definitions

#LINK <identifier 2>
#PARAMETERS
// parameter definitions
#CONSTRAINTS
// constraint definitions
```

Fig 3.11 Hyperedge Structure

Parameters

Parameters defined in a #HYPEREDGE block follow the exact same rules as the ones defined in #NODE blocks.

Constraints

While affine constraints involving all variables declared in a #NODE block can be defined in the same block, constraints defined in #HYPEREDGE blocks couple external variables associated with any subset of nodes. The syntax for defining constraints is otherwise the same as the one used in #NODE blocks:

```
#CONSTRAINTS
 <expression> == <expression> <expansion range>;
 <expression> <= <expression> <expansion range>;
 <expression> >= <expression> <expansion range>;
```

Fig 3.12 Hyperedge's Constraints Structure

Similarly to the constraints defined in nodes, hyperedges can also be named by adding an identifier with colon before the constraint, as follows,

<constraint identifier>: <constraint>;

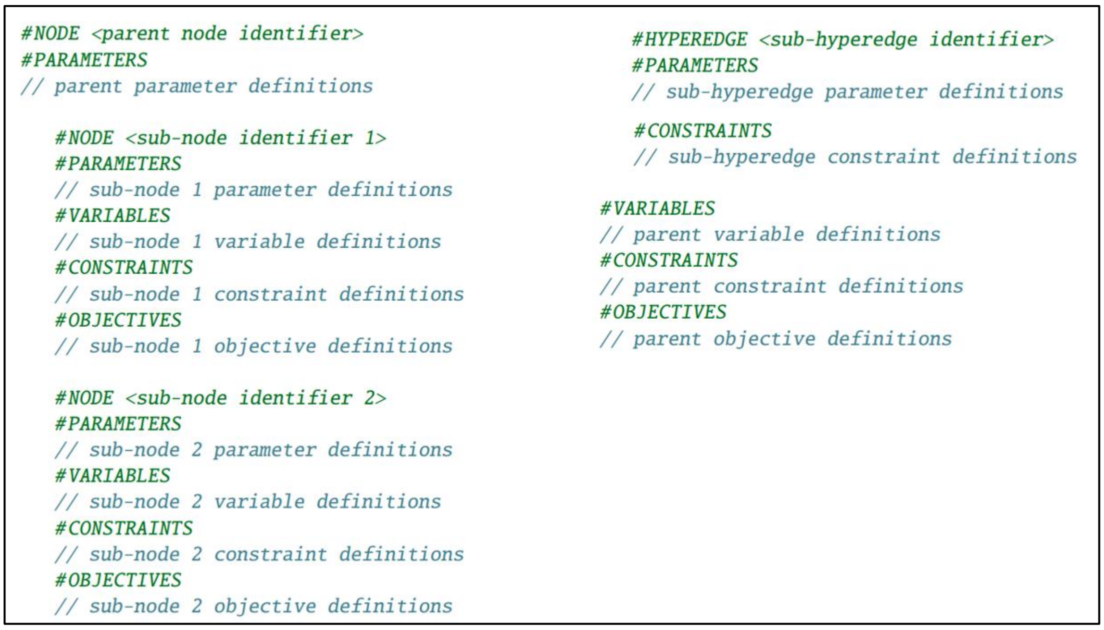
An Example including valid Hyperedge Blocks:

```
#TIMEHORIZON
// time horizon definition

#NODE node1
#VARIABLES
external : x;
external : inflow[T];
// further node content

#NODE node2
#VARIABLES
external : y;
external : outflow[T];
// further node content

#HYPEREDGE hyperedge1
#CONSTRAINTS
node1.inflow[t] == node2.outflow[t];

#NODE node3
#VARIABLES
external : z;
// further node content

#LINK hyperedge2
#PARAMETERS
weight = {1/3,2/3};
#CONSTRAINTS
node1.x <= weight[0]*node2.y + weight[1]*node3.z;
node2.y <= node3.z;
```

Fig 3.13 Optimisation Problem with valid Nodes and Hyperedge

3.4 Hierarchical Model

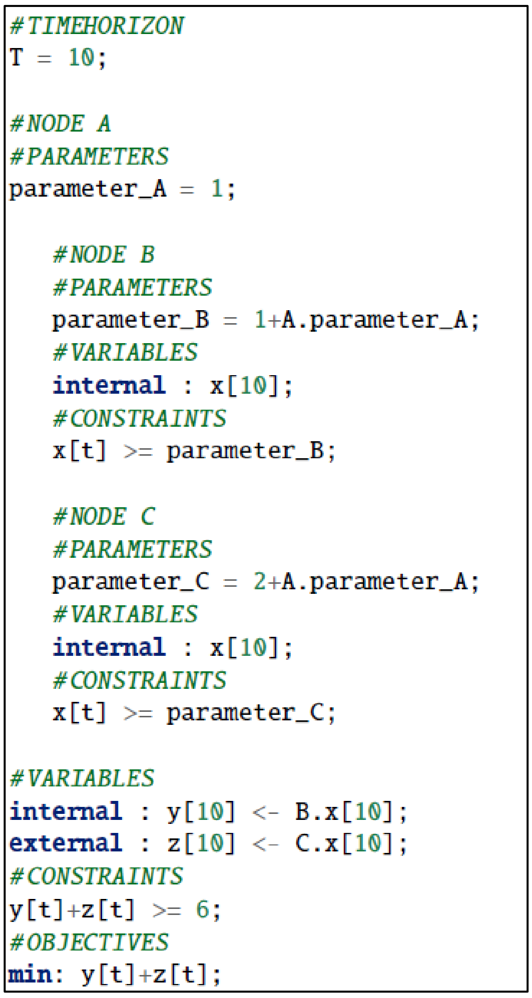
Within a hierarchical hypergraph, nodes can be constructed in a bottom-up manner, originating from sub-nodes connected by sub-hyperedges. Sub-nodes and sub-hyperedges are delineated between the #PARAMETERS and #VARIABLES blocks of a parent node. Consequently, a standard hierarchical block #NODE follows this structure:



```
#NODE <parent node identifier>
#PARAMETERS
// parent parameter definitions

   #NODE <sub-node identifier 1>
   #PARAMETERS
   // sub-node 1 parameter definitions
   #VARIABLES
   // sub-node 1 variable definitions
   #CONSTRAINTS
   // sub-node 1 constraint definitions
   #OBJECTIVES
   // sub-node 1 objective definitions

   #NODE <sub-node identifier 2>
   #PARAMETERS
   // sub-node 2 parameter definitions
   #VARIABLES
   // sub-node 2 variable definitions
   #CONSTRAINTS
   // sub-node 2 constraint definitions
   #OBJECTIVES
   // sub-node 2 objective definitions

   #HYPEREDGE <sub-hyperedge identifier>
   #PARAMETERS
   // sub-hyperedge parameter definitions

   #CONSTRAINTS
   // sub-hyperedge constraint definitions

#VARIABLES
// parent variable definitions
#CONSTRAINTS
// parent constraint definitions
#OBJECTIVES
// parent objective definitions
```

Fig 3.14 Hierarchical Model Structure

A valid Hierarchical Model is as shown below:



```
#TIMEHORIZON
T = 10;

#NODE A
#PARAMETERS
parameter_A = 1;

   #NODE B
   #PARAMETERS
   parameter_B = 1+A.parameter_A;
   #VARIABLES
   internal : x[10];
   #CONSTRAINTS
   x[t] >= parameter_B;

   #NODE C
   #PARAMETERS
   parameter_C = 2+A.parameter_A;
   #VARIABLES
   internal : x[10];
   #CONSTRAINTS
   x[t] >= parameter_C;

#VARIABLES
internal : y[10] <- B.x[10];
external : z[10] <- C.x[10];
#CONSTRAINTS
y[t]+z[t] >= 6;
#OBJECTIVES
min: y[t]+z[t];
```

Fig 3.15 Valid Hierarchical Model

A basic Mathematical Optimisation problem is solved using GBOML. The problem statement and code solution is as follows:

$$\begin{gathered} Minimize \quad 80x + 95y \\ s.t. \end{gathered}$$

$$10x + 15y >= 100$$

$$20x + 15y >= 160$$

$$x, y >= 0$$

GBOML Code:

1. Text File Code:

```
#TIMEHORIZON
T = 1;

#NODE A
#PARAMETERS
a=80;
b=95;
#VARIABLES
internal : x;
internal : y;
#CONSTRAINTS
x >= 0;
y >= 0;
10*x+15*y >= 100;
20*x+15*y >= 160;
#OBJECTIVES
    min : a*x+b*y;
```

2. Python Parser Code:

```
from gboml import GbomlGraph

gboml_model = GbomlGraph(24*365)
nodes, edges, _ = gboml_model.import_all_nodes_and_edges("path_to_GBOML_directory/
↪examples/microgrid/microgrid.txt")
gboml_model.add_nodes_in_model(*nodes)
gboml_model.add_hyperedges_in_model(*edges)
gboml_model.build_model()
solution = gboml_model.solve_cplex()
print(solution)
```

Fig 3.16 Python code to be entered in Python Environment

CHAPTER 4

MICRO GRID AND GLOBAL GRID

4.1 What is a Micro Grid?

A microgrid is a localized and self-contained electrical grid that operates independently or in conjunction with the main grid. It consists of a network of interconnected electricity sources, energy storage systems, loads, and control systems, all located within a defined geographic area. Microgrids can function autonomously, known as islanded mode, or can be connected to the main grid, known as grid-connected mode.

Microgrids are designed to provide reliable, resilient, and efficient electricity supply to local communities, institutions, or facilities. They often incorporate renewable energy sources such as solar photovoltaic panels, wind turbines, and biomass generators, along with conventional energy sources like diesel generators or natural gas turbines. Energy storage systems, such as batteries or flywheels, are also integral components of microgrids, helping to store excess energy for use during periods of high demand or low renewable energy availability[6].

One of the key features of microgrids is their ability to operate independently from the main grid during emergencies, such as power outages or natural disasters. In islanded mode, microgrids can continue to supply power to critical loads, such as hospitals, emergency services, and communication networks, ensuring continuity of essential services.

Microgrids also offer benefits such as increased energy security, reduced carbon emissions, and improved grid resilience. They enable local communities to generate and consume electricity locally, thereby reducing reliance on centralized power plants and long-distance transmission lines. Additionally, microgrids support the integration of renewable energy sources, contributing to the transition towards a more sustainable and decentralized energy system.

Overall, microgrids play a crucial role in enhancing energy reliability, resilience, and sustainability, making them an increasingly important component of modern energy infrastructure.

4.2 Components of a Micro Grid

Here's a detailed explanation of the components of a microgrid:

Energy Sources:

Renewable Energy Sources:

These include solar photovoltaic (PV) panels, wind turbines, biomass generators, and hydroelectric generators. They harness energy from natural resources and can provide clean, sustainable power to the microgrid.

Conventional Energy Resources:

In addition to renewables, microgrids may incorporate conventional energy sources like diesel generators or natural gas turbines for backup or supplementary power generation, especially during periods of low renewable energy availability.

Energy Storage Systems (ESS):

Battery Energy Storage:

Batteries store excess energy generated during times of low demand or high renewable energy production for use during periods of high demand or low renewable energy availability. They help in balancing supply and demand within the microgrid.

Flywheel Energy Storage:

Flywheels store kinetic energy in a rotating mass and can quickly discharge stored energy to stabilize the microgrid in response to sudden changes in load or generation.

Control Systems:

Microgrid Controller:

The microgrid controller manages and coordinates the operation of various components within the microgrid, including energy sources, energy storage systems, and loads. It optimizes energy flow, ensures system stability, and facilitates seamless transitions between grid-connected and islanded modes of operation.

Distributed Energy Resource Management System (DERMS):

DERMS monitors and controls distributed energy resources within the microgrid to optimize their performance, maximize efficiency, and maintain grid stability.

Loads:

Critical Loads:

These are essential loads that must remain powered at all times, such as hospitals, emergency services, and critical infrastructure.

Non-Critical Loads:

Non-critical loads, such as residential, commercial, and industrial loads, can be prioritized or shed during times of high demand or low generation to maintain system stability.

Interconnection Infrastructure:

Distribution Network:

The distribution network forms the physical infrastructure that interconnects the various components of the microgrid, including energy sources, energy storage systems, and loads.

Switchgear and Protection Systems:

Switchgear and protection systems ensure the safe and reliable operation of the microgrid by isolating faulty sections, controlling power flow, and protecting equipment from damage due to overloads or faults.

Monitoring and Communication Systems:

Sensors and Monitoring Devices:

These devices continuously monitor key parameters such as voltage, frequency, power flow, and energy production/consumption within the microgrid.

Communication Networks:

Communication networks enable real-time data exchange between different components of the microgrid and facilitate remote monitoring, control, and optimization of the system. Together, these components work in concert to create a resilient, flexible, and sustainable energy ecosystem that can operate autonomously or in coordination with the main grid, providing reliable power to local communities while enhancing energy security and resilience.

4.3 Global Grid

A global grid, also known as a global electricity grid or interconnected global grid, is a theoretical concept that proposes the creation of a worldwide network of interconnected electrical grids. Unlike traditional localized grids or regional interconnections, a global grid would span continents and oceans, facilitating the transmission of electricity across vast distances on a global scale.

The idea behind a global grid is to optimize the utilization of renewable energy resources by enabling the seamless exchange of electricity between regions with surplus renewable energy generation and regions with high energy demand. For example, solar energy could be harvested in sunny regions during the day and transmitted to other parts of the world experiencing nighttime or unfavorable weather conditions.

Key features of a global grid may include:

Intercontinental Transmission Lines:
High-voltage direct current (HVDC) transmission lines would be used to transmit electricity over long distances with minimal losses. These transmission lines would connect various regions and continents, forming the backbone of the global grid.

Energy Storage:
Energy storage systems, such as large-scale batteries or pumped hydroelectric storage, could be deployed strategically along the global grid to store surplus energy during times of low demand and release it when needed.

Grid Management and Control Systems:

Advanced control systems and grid management software would be implemented to monitor and optimize the flow of electricity across the global grid, ensuring stability, reliability, and efficient operation.

International Cooperation and Governance:
The development and operation of a global grid would require international cooperation and coordination among governments, utilities, and other stakeholders. Agreements and protocols would need to be established to govern the sharing of electricity, allocation of costs, and resolution of disputes.

While the concept of a global grid offers potential benefits such as enhanced energy security, increased renewable energy penetration, and mitigation of climate change, its realization faces significant technical, economic, and geopolitical challenges. These challenges include the high cost of infrastructure development, regulatory barriers, political considerations, and concerns about energy sovereignty and security.

Overall, while the idea of a global grid remains largely theoretical at present, ongoing advancements in technology, renewable energy deployment, and international cooperation may pave the way for its eventual implementation in the future.

4.4 Challenges for Global Grid System

The concept of a global grid, while promising in theory, faces numerous challenges that must be addressed for its successful implementation. Some of the key challenges include:

Technical Complexity:
Building and operating a global grid would require the development of complex infrastructure, including intercontinental transmission lines, high-voltage converters, and advanced control systems. Managing such a vast and interconnected network would pose significant technical challenges, including voltage control, grid stability, and synchronization of power systems across different regions.

Cost and Financing:

The construction of a global grid would involve substantial investment in infrastructure, including transmission lines, substations, and energy storage facilities. Financing such a large-scale project would require significant capital investment from governments, utilities, and private investors. The high upfront costs and long payback periods may deter investment, especially in regions with limited financial resources.

Regulatory and Policy Changes:

Establishing a regulatory framework and coordinating policies among different countries and regions would be essential for the operation of a global grid. Harmonizing regulatory standards, resolving legal disputes, and addressing issues related to tariffs, taxes, and energy trading would require international cooperation and coordination. Political considerations, including national sovereignty and security concerns, may also complicate regulatory efforts.

Geopolitical Considerations:

The development of a global grid would involve collaboration among countries with diverse political, economic, and social systems. Negotiating agreements, resolving conflicts of interest, and ensuring equitable distribution of benefits would require diplomatic skill and international cooperation. Geopolitical tensions, trade disputes, and geopolitical rivalries could hinder progress towards a global grid.

Intermittency and Variability of Renewable Energy:

A significant portion of electricity transmitted through a global grid would likely be generated from renewable energy sources such as solar and wind power. However, these sources are inherently variable and intermittent, posing challenges for grid stability and reliability. Integrating large amounts of renewable energy into the global grid would require advances in energy storage, grid management, and demand response technologies[7].

Environmental and Social Impacts:

The construction of intercontinental transmission lines and associated infrastructure could have environmental and social impacts, including habitat destruction, land use conflicts, and displacement of communities. Balancing the need for energy access and environmental sustainability would require careful planning, stakeholder engagement, and adherence to environmental regulations.

Addressing these challenges will require collaboration among governments, utilities, industry stakeholders, and civil society organizations. While the vision of a global grid holds immense potential for enhancing energy security, promoting renewable energy deployment, and mitigating climate change, overcoming these challenges will be essential for its realization.

4.5 Benefits of Global Grid System

The concept of a global grid, while presenting significant challenges, also offers numerous potential benefits that could address key energy and environmental challenges on a global scale. Some of the main benefits of a global grid include:

Enhances Energy Security:
A global grid would create a more interconnected and resilient energy infrastructure, reducing dependence on local or regional energy sources and increasing overall energy security. By diversifying energy supply sources and enabling energy sharing across regions, a global grid could help mitigate the impact of disruptions, such as natural disasters or geopolitical conflicts, on energy supplies.

Increased Renewable Energy Integration:
A global grid would facilitate the integration of renewable energy sources, such as solar, wind, and hydroelectric power, on a larger scale. By connecting regions with abundant renewable energy resources to regions with high energy demand, a global grid could optimize the use of renewable energy and reduce reliance on fossil fuels. This would help accelerate the transition to a low-carbon energy system and mitigate climate change.

Optimised Energy Efficiency:

A global grid could enable the more efficient use of energy resources by balancing supply and demand across different time zones and regions. Excess energy generated during periods of low demand or high renewable energy production could be transmitted to regions with higher demand, reducing the need for costly energy storage or curtailment of renewable energy generation.

Access to Clean and Affordable Energy:
By connecting remote and underserved regions to the global energy network, a global grid could expand access to clean and affordable energy. This could improve living standards, stimulate economic development, and reduce energy poverty in regions lacking access to reliable electricity infrastructure.

Economic Benefits:
The development and operation of a global grid would create significant economic opportunities, including job creation, investment in infrastructure, and growth of the renewable energy industry. By enabling cross-border energy trade and investment, a global grid could also promote economic cooperation and integration among countries and regions.

Geopolitical Stability:
A global grid could promote geopolitical stability by fostering energy cooperation and interdependence among countries and regions. By reducing energy dependency on a single country or region, a global grid could mitigate geopolitical tensions and conflicts related to energy resources.

Overall, while the realization of a global grid would require overcoming numerous technical, economic, and political challenges, the potential benefits in terms of energy security, renewable energy integration, economic development, and environmental sustainability make it an attractive proposition for policymakers, industry stakeholders, and civil society organizations alike.

4.6 Micro Grid Example

Problem Description

A grid-connected microgrid is a compact electric power system designed to ideally operate independently. It comprises interconnected electric generators, such as solar panels or fossil fuel generators, and loads, representing electricity consumers. Often, an electrical storage system is integrated into the microgrid to manage the balance between electricity production and consumption over time, while reducing reliance on the main distribution network. The schematic representation of the microgrid configuration is depicted in Figure 9.1.

This section focuses on investigating the process of sizing an electric microgrid similar to the one illustrated in Figure 9.1. The objective of the sizing problem is to determine the required capacity of solar panels and battery storage to minimize the overall cost of meeting specified electricity demand levels throughout the system's lifespan. Therefore, both initial investment and ongoing operational costs are considered. Additionally, the electricity consumption within the microgrid and the solar irradiation received by the panels are assumed to be known for a typical representative day.

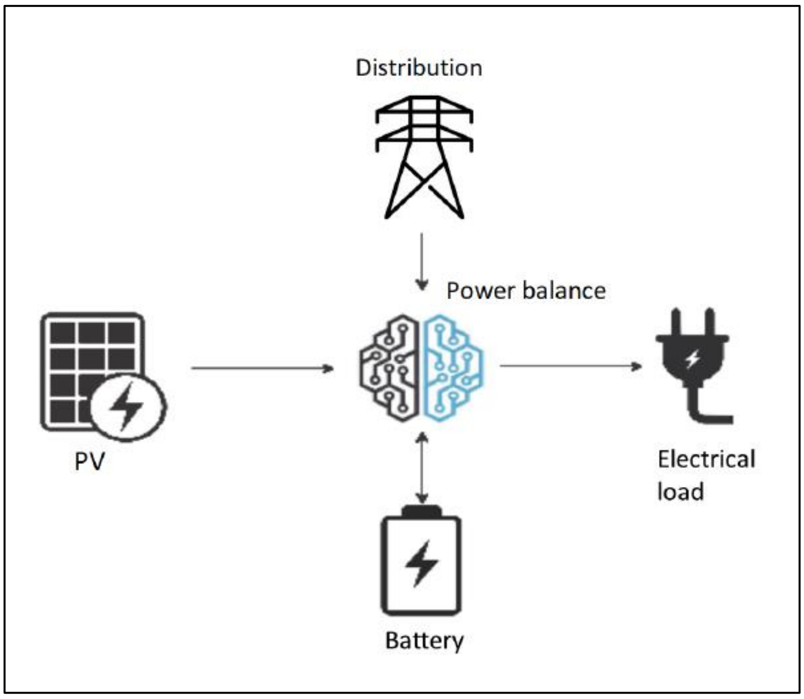
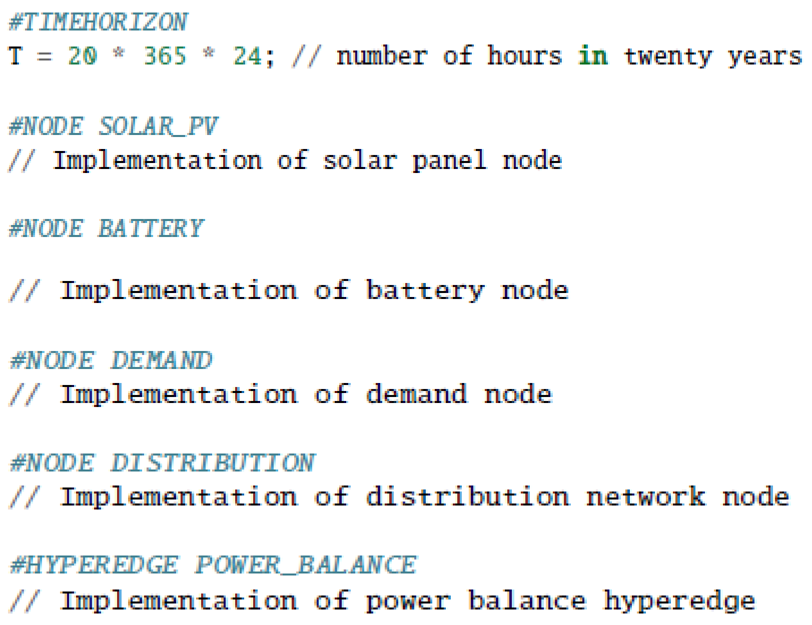


Fig 4.1 Micro Grid System Configuration

GBOML Implementation

The system comprises four nodes that emulate the behavior of various components within the microgrid. The initial node corresponds to solar panels, capturing their characteristics and output. The subsequent node delineates the dynamics and associated costs of a battery, factoring in the power flows involved in its charging and discharging processes. The third node models the load imposed by consumers within the microgrid. The final node represents the infrastructure of the electricity distribution network. The power equilibrium within the system is depicted through a hyperedge. Below is an outline of the optimization problem formulated in the language. Initially, the time horizon (T) is established, signifying the duration over the system's operational lifespan, which is assumed to span twenty years. Subsequently, the four nodes are instantiated, and thereafter, these nodes are interconnected via a hyperedge.

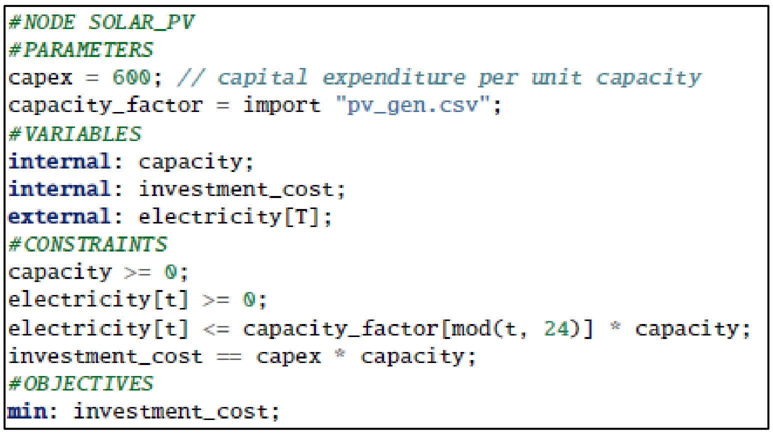


```
#TIMEHORIZON
T = 20 * 365 * 24; // number of hours in twenty years

#NODE SOLAR_PV
// Implementation of solar panel node

#NODE BATTERY

// Implementation of battery node

#NODE DEMAND
// Implementation of demand node

#NODE DISTRIBUTION
// Implementation of distribution network node

#HYPEREDGE POWER_BALANCE
// Implementation of power balance hyperedge
```

Fig 4.2 Skeleton of GBOML Implementation of Micro Grid Problem

Solar PV Node

This node incorporates two scalar internal variables: one for the capacity and another for the associated investment cost. A time-dependent external variable is utilized to model the power generated by the panels. Constraints are employed to specify the investment cost and the power output of the solar panels, as well as to enforce the non-negativity of optimization variables. Ultimately, the objective function seeks to minimize the investment cost.

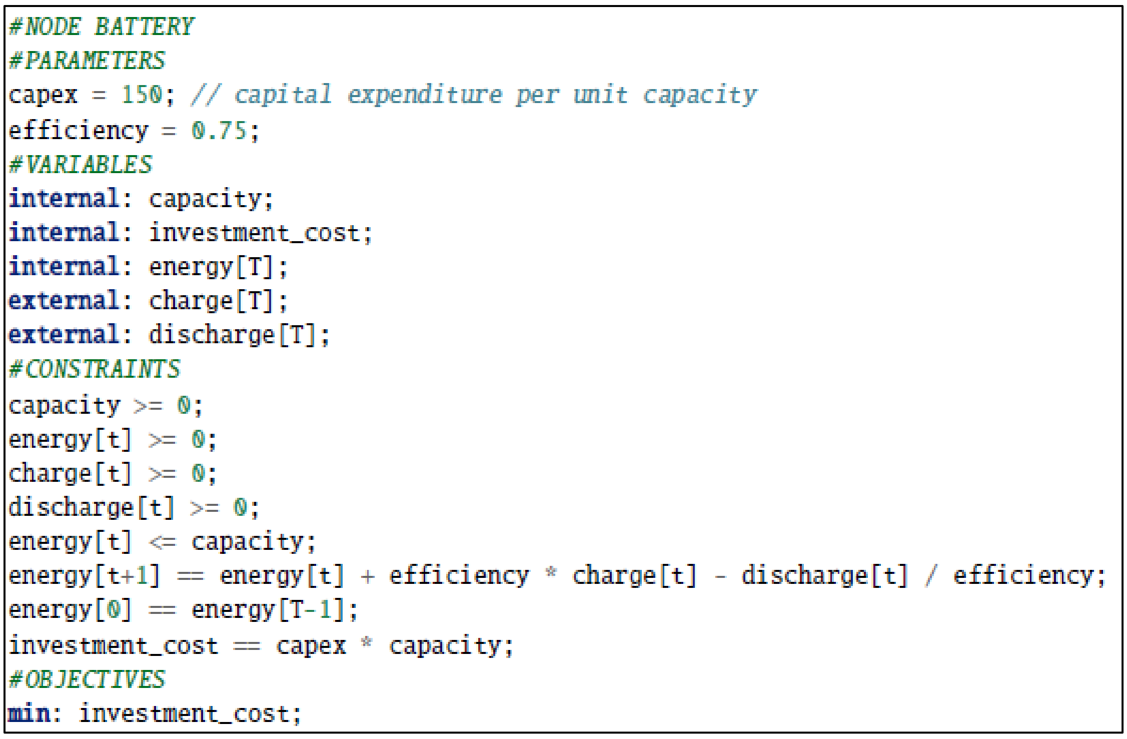


```
#NODE SOLAR_PV
#PARAMETERS
capex = 600; // capital expenditure per unit capacity
capacity_factor = import "pv_gen.csv";
#VARIABLES
internal: capacity;
internal: investment_cost;
external: electricity[T];
#CONSTRAINTS
capacity >= 0;
electricity[t] >= 0;
electricity[t] <= capacity_factor[mod(t, 24)] * capacity;
investment_cost == capex * capacity;
#OBJECTIVES
min: investment_cost;
```

Fig 4.3 Solar PV Node

Battery Node

A battery functions as an electrical storage device capable of retaining energy. The installed capacity, measured in watt-hours, signifies the maximum energy storage capacity of the battery. Similar to solar panels, the deployment of one unit of battery storage capacity incurs a capital expenditure measured in currency per watt-hour, resulting in the calculation of the investment cost. Energy can be either charged or discharged from the battery, with charging power and discharging power denoted accordingly in watts. The energy stored in the battery, also known as the state of charge, is limited by the installed capacity. Additionally, the state of charge is governed by the power flow in and out of the battery through the efficiency-dependent constraint. Furthermore, it is customary to ensure continuity in the energy stored within the battery across the time horizon. In addition to the two scalar internal variables representing installed capacity and investment cost, a time-dependent internal variable captures the energy stored in the battery. Moreover, the charging and discharging power flows are defined as time-dependent external variables of the battery node. Ultimately, the objective is to minimize the investment cost. The detailed implementation is presented below:

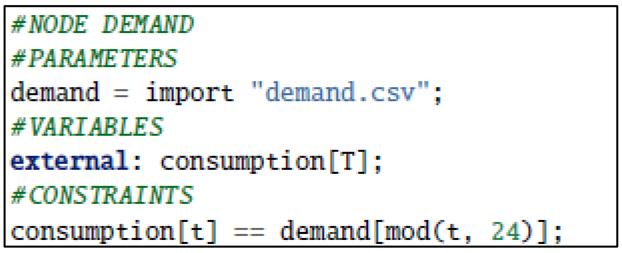


```
#NODE BATTERY
#PARAMETERS
capex = 150; // capital expenditure per unit capacity
efficiency = 0.75;
#VARIABLES
internal: capacity;
internal: investment_cost;
internal: energy[T];
external: charge[T];
external: discharge[T];
#CONSTRAINTS
capacity >= 0;
energy[t] >= 0;
charge[t] >= 0;
discharge[t] >= 0;
energy[t] <= capacity;
energy[t+1] == energy[t] + efficiency * charge[t] - discharge[t] / efficiency;
energy[0] == energy[T-1];
investment_cost == capex * capacity;
#OBJECTIVES
min: investment_cost;
```

Fig 4.4 Battery Node

Demand Node

In the demand node depicted below, the electrical consumption (measured in watts) is determined for each time, utilizing a time series supplied as a parameter. This time series furnishes the standard consumption pattern for the 24-hour duration of a representative day. No specific objective is necessitated for this computation.

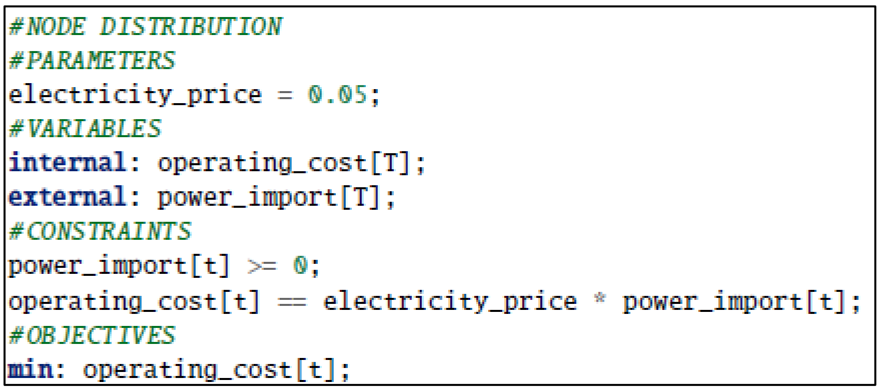


```
#NODE DEMAND
#PARAMETERS
demand = import "demand.csv";
#VARIABLES
external: consumption[T];
#CONSTRAINTS
consumption[t] == demand[mod(t, 24)];
```

Fig 4.5 Demand Node

Distribution Node

The distribution node characterizes the connectivity of the microgrid to the distribution network. It allows for the procurement of power (measured in watts) from the grid to address any potential power deficits within the microgrid at time. This purchased power incurs a marginal price (in currency per watt), resulting in an operating cost at each time. The objective of this node is to minimize the total operating cost over the system's lifespan, which is expressed as the summation of operating costs over all time periods. This node is implemented below, with the imported power represented as a time-dependent external variable. Additionally, the operating cost is calculated using a time-dependent internal variable, and the overall operating cost is minimized.



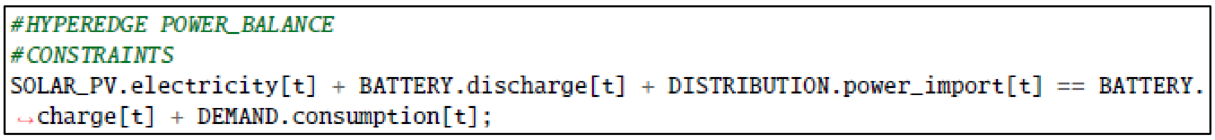
```
#NODE DISTRIBUTION
#PARAMETERS
electricity_price = 0.05;
#VARIABLES
internal: operating_cost[T];
external: power_import[T];
#CONSTRAINTS
power_import[t] >= 0;
operating_cost[t] == electricity_price * power_import[t];
#OBJECTIVES
min: operating_cost[t];
```

Fig 4.6 Distribution Node

Power Balance HyperEdge

Each node within the microgrid is linked through a hyperedge, which enforces an equality constraint representing the equilibrium between electricity generation and consumption within the microgrid. Consequently, the combined solar production, battery discharge power and purchased power from the distribution network must equal the sum of battery charging power and power consumed by loads and appliances.

This hyperedge can be instantiated as depicted below:



```
#HYPEREDGE POWER_BALANCE
#CONSTRAINTS
SOLAR_PV.electricity[t] + BATTERY.discharge[t] + DISTRIBUTION.power_import[t] == BATTERY.
↪charge[t] + DEMAND.consumption[t];
```

Fig 4.7 Power Balance Hyperedge

The complete model is obtained by substituting the code blocks of all nodes in the skeleton code introduced earlier. The model is then translated using the GBOMLcompiler and solved with Gurobi or any other solver.

```python
from gboml import GbomlGraph

gboml_model = GbomlGraph(24*365)
nodes, edges, _ = gboml_model.import_all_nodes_and_edges("path_to_GBOML_directory/
↪examples/microgrid/microgrid.txt")
gboml_model.add_nodes_in_model(*nodes)
gboml_model.add_hyperedges_in_model(*edges)
gboml_model.build_model()
solution = gboml_model.solve_cplex()
print(solution)
```

Fig 4.8 Python Parser Code for GBOML Micro Grid Problem

CHAPTER 5

INTERCONNECTION OF MICRO GRIDS

5.1 Using Randomly generated Data

5.1.1 Reference Data

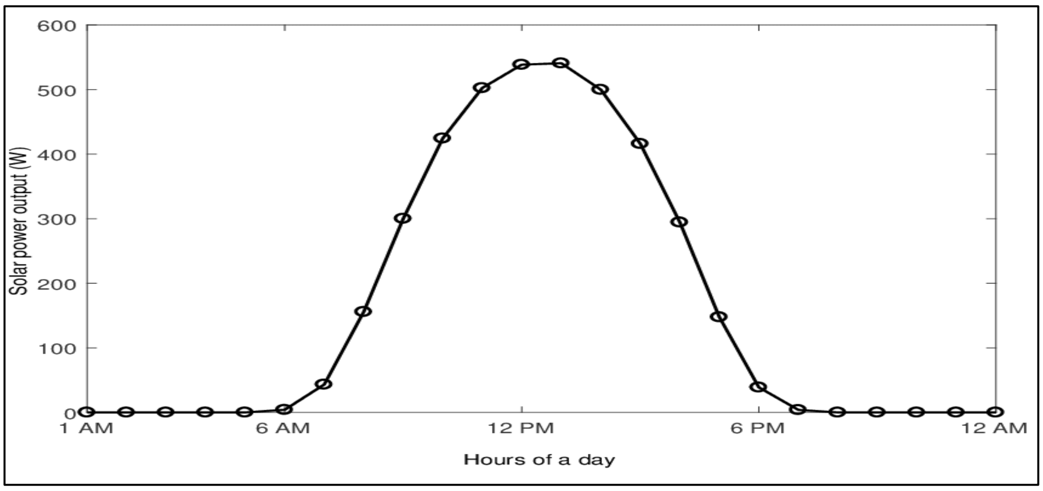


Fig 5.1 Average Generation per hour for a day per solar panel

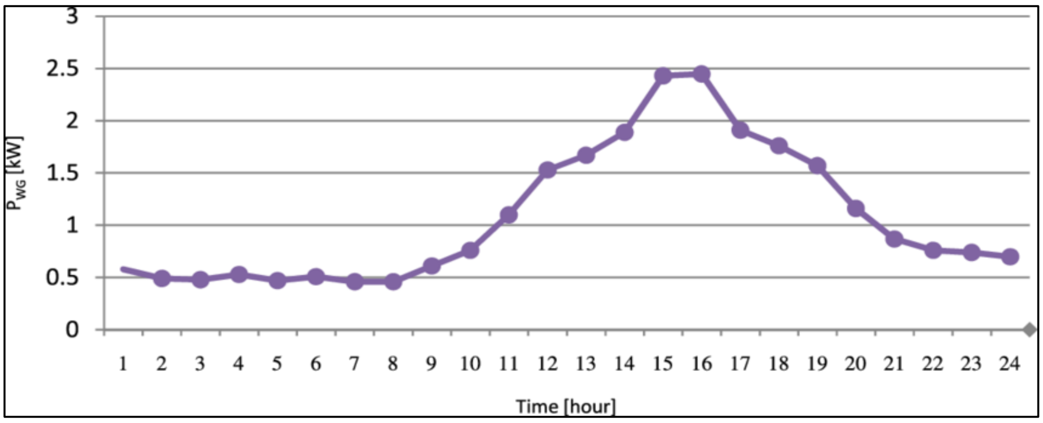


Fig 5.2 Average Generation per hour for a day per windmill

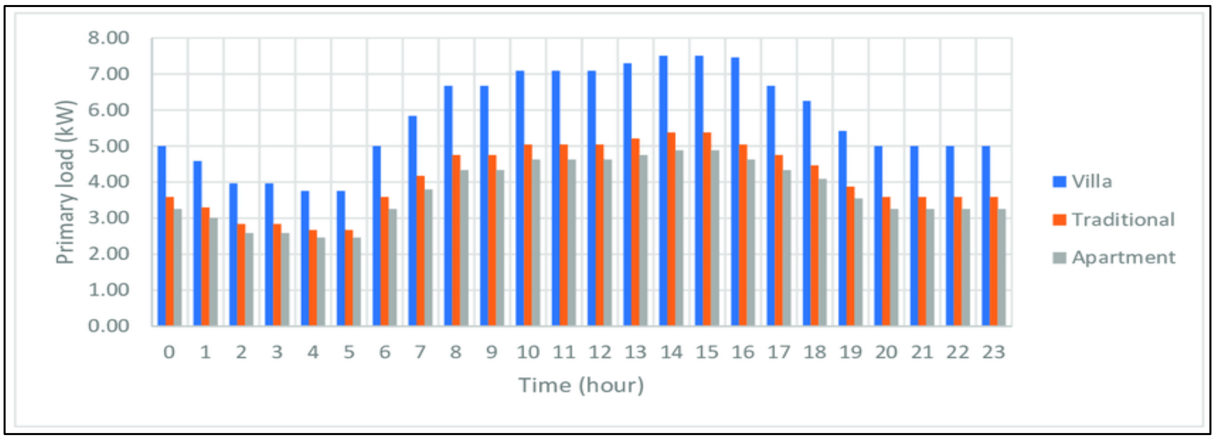


Fig 5.3 Average Load Requirement per hour per apartment for a day

Number of windmills: 100

Number of Solar Panels: 100

Number of Loads: 100

Battery Maximum Capacity: 100 kW[8]

Table 5.1 Time Zone, Generation and Load Assumption Values Table

TIME ZONE	GENERATION			LOAD (kW)
	SOLAR (kW)	WIND (kW)	TOTAL (kW)	
6AM-3PM	250-350	100-200	350-550	300-500
3PM-10PM	50-100	150-250	200-350	400-500
10PM-6AM	0	50-150	50-150	250-350

5.1.2 Test Cases

Test Case 1

Microgrid 1 (M1) is producing surplus energy while Microgrid 2 (M2) is deficit of energy and M1 is sufficient to clear M2 deficit.

Test Case 2

M1 is producing surplus energy while M2 is deficit of energy and M1 is not sufficient to clear M2 deficit, so the deficit is cleared by the grid.

Test Case 3

M2 is producing surplus energy while M1 is deficit of energy and M2 is sufficient to clear M1 deficit.

Test Case 4

M2 is producing surplus energy while M1 is deficit of energy and M2 is not sufficient to clear M1 deficit, so the deficit is cleared by the grid.

Test Case 5

Both M1 and M2 are deficit of energy. The Grid clears the deficit.

Test Case 6

Both M1 and M2 are Surplus.

5.2Code

```
import random
batt1 = 100
batt2 = 100
for time in range(1,25):
    print('Hour:',time)
    if (time>=6 and time<15):
    if (time>=6 and time<15):
    if (time>=6 and time<15):
    if (time>=6 and time<15):
        pvgen1 = random.randint(150,300)
        pvgen2 = random.randint(150,300)
        load1 = random.randint(300,400)
```

```
            load2 = random.randint(300,400)
        elif (time>=15 and time<22):
            pvgen1 = random.randint(150,300)
            pvgen2 = random.randint(150,300)
            load1 = random.randint(400,500)
            load2 = random.randint(400,500)
        elif (time>=22 or time<6):
            pvgen1 = random.randint(150,300)
            pvgen2 = random.randint(150,300)
            load1 = random.randint(250,300)
            load2 = random.randint(250,300)

        print("pvgen1:",pvgen1)
        print("pvgen2:",pvgen2)
        print("load1:",load1)
        print("load2:",load2)

        #Case1 M1 surplus M2 Deficit and M1 surplus is enough
        if(load1<(batt1+pvgen1) and load2>(batt2+pvgen2) and
    ((batt1+pvgen1-load1)-(load2-(batt2+pvgen2))>=0)):
            print('It is Case1: M1 surplus M2 Deficit and M1
    surplus is enough')
            print('Battery2,pvgen2 will completely get
    diminished and there is need of energy which is taken from
    M1')
            print('M1 surplus energy is enough alone no need of
    energy from Generation')
            if(pvgen1>=load1):
                print('Battery1 dosent get discharged since
    pvgen1 alone is enough to clear load1')
                if((pvgen1-load1)>=(load2-pvgen2-batt2)):
                    print('M2 is cleared since pvgen1 alone is
    sufficient here battery1 is no needed to discharge')
```

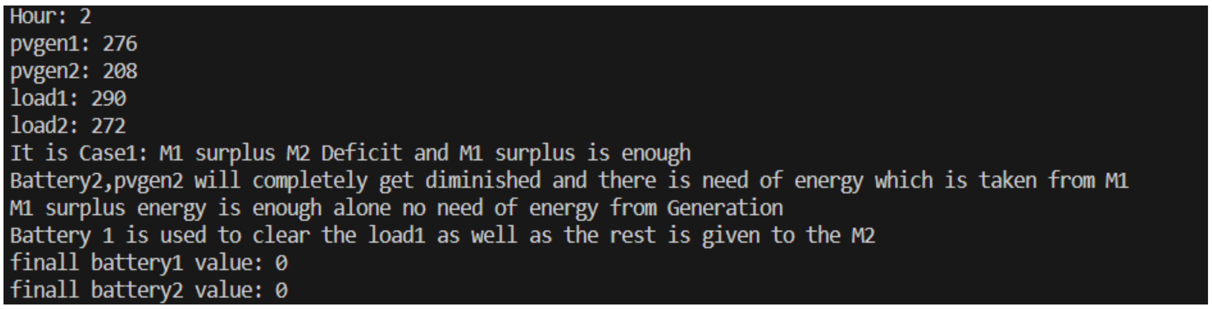
```
                if(batt1<100):
                    if(((pvgen1-load1)-(load2-pvgen2-
batt2))>(100-batt1)):
                        print('Excess Energy is getting
stored in battery after clearning M2')
                        batt1=100
                else:
                    print('Excess Energy is getting stored
in battery after clearning M2')
                    batt1=batt1+((pvgen1-load1)-(load2-
batt2-pvgen2))

            else:
                print('Whole Pvgen1 is used to provide
energy to M2 and rest is taken from battery')
                batt1=batt1-((load2-pvgen2-batt2)-(pvgen1-
load1))
        else:
            print('Battery 1 is used to clear the load1 as
well as the rest is given to the M2')
            batt1=batt1-((load1-pvgen1)+(load2-pvgen2-
batt2))
        batt2=0

    #Case2 M1 surplus M2 Deficit but M1 surplus is not
enough,G is needed
    elif(load1<(batt1+pvgen1) and load2>(batt2+pvgen2) and
((batt1+pvgen1-load1)-(load2-(batt2+pvgen2))<0)):
        print('It is Case2: M1 surplus M2 Deficit but M1
surplus is not enough,G is needed')
        print('Battery2,pvgen2 will completely get
diminished and there is need of energy which is taken from
M1')
```

```
        print('M1 surplus energy is not enough and energy
from Generation is needed')
        print('both batteries get discharge completelty')
        batt1=0
        batt2=0


    #Case3 M2 surplus M1 Deficit and M2 surplus is enough
    elif(load2<(batt2+pvgen2) and load1>(batt1+pvgen1) and
((batt2+pvgen2-load2)-(load1-(batt1+pvgen1))>=0)):
        print('It is Case3: M2 surplus M1 Deficit and M2
surplus is enough')
        print('Battery1,pvgen1 will completely get
diminished and there is need of energy which is taken from
M2')
        print('M2 surplus energy is enough alone no need of
energy from Generation')
        if(pvgen2>=load2):
            print('Battery2 dosent get discharged since
pvgen1 alone is enough to clear load2')
            if((pvgen2-load2)>=(load1-pvgen1-batt1)):
                print('M1 is cleared since pvgen1 alone is
sufficient here battery1 is no needed to discharge')
                if(batt2<100):
                    if(((pvgen2-load2)-(load1-pvgen1-
batt1))>(100-batt2)):
                        print('Excess Energy is getting
stored in battery after clearning M1')
                        batt2=100
                else:
                    print('Excess Energy is getting stored
in battery after clearning M1')
                    batt2=batt2+((pvgen2-load2)-(load1-
batt1-pvgen1))

            else:
                print('Whole Pvgen2 is used to provide 
energy to M1 and rest is taken from battery')
                batt2=batt2-((load1-pvgen1-batt1)-(pvgen2-
load2))
        else:
            print('Battery 2 is used to clear the load2 as 
well as the rest is given to the M1')
            batt2=batt2-((load2-pvgen2)+(load1-pvgen1-
batt1))
        batt1=0

    #Case4 M2 surplus M1 Deficit but M2 surplus is not 
enough,G is needed
    elif(load2<(batt2+pvgen2) and load1>(batt1+pvgen1) and 
((batt2+pvgen2-load2)-(load1-(batt1+pvgen1))<0)):
        print('It is Case4: M2 surplus M1 Deficit but M2 
surplus is not enough,G is needed')
        print('Battery1,pvgen1 will completely get 
diminished and there is need of energy which is taken from 
M2')
        print('M2 surplus energy is not enough and energy 
from Generation is needed')
        print('both batteries get discharge completelty')
        batt1=0
        batt2=0

    #case5 Both are deficit
    elif(load1>(batt1+pvgen1) and load2>(batt2+pvgen2)):
        print('It is Case5: Both are deficit')
        print('both batteries get discharged completely and
generaation station is used for power supply to load')
        batt1=0
        batt2=0

    #case6 both are surplus
    elif(load1<=(batt1+pvgen1) and load2<=(batt2+pvgen2)):
        print('It is Case6:  both are surplus')
        print('Here both are surplus so no energy transfer
between M1 and M2')
        if(pvgen1>=load1):
            print('pvgen1 alone is sufficient to diiminish
load demand so no battery discharge')
            if(batt1<100):
                print('excess energy produced from pvgen1
is used to charge battery1')
                if((pvgen1-load1)>=(100-batt1)):
                    batt1=100
                else:
                    batt1=batt1+(pvgen1-load1)
        else:
            print('battery1 is also used to satisfy load1')
            batt1=batt1-(load1-pvgen1)

        if(pvgen2>=load2):
            print('pvgen2 alone is sufficient to diiminish
load demand so no battery discharge')
            if(batt2<100):
                print('excess energy produced from pvgen2
is used to charge battery2')
                if((pvgen2-load2)>=(100-batt2)):
                    batt2=100
                else:
                    batt2=batt2+(pvgen2-load2)
```

```python
        else:
            print('battery2 is also used to satisfy load2')
            batt2=batt2-(load2-pvgen2)

    else:
        print('No case exists')

    print('finall battery1 value:',batt1)
    print('finall battery2 value:',batt2)
    print('------------------------------------------------
------------------------')
```

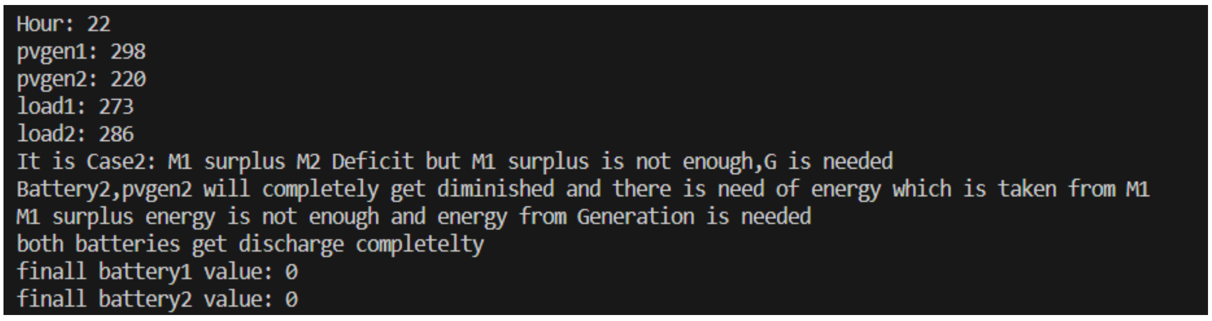
5.3 Output

Test Case 1



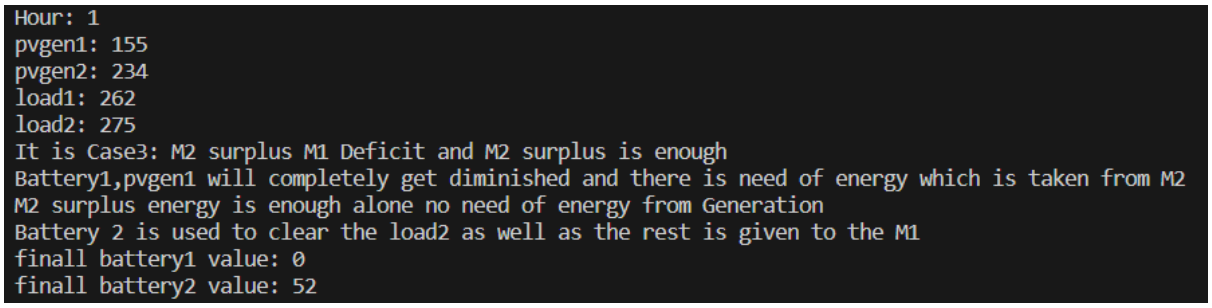
```
Hour: 2
pvgen1: 276
pvgen2: 208
load1: 290
load2: 272
It is Case1: M1 surplus M2 Deficit and M1 surplus is enough
Battery2,pvgen2 will completely get diminished and there is need of energy which is taken from M1
M1 surplus energy is enough alone no need of energy from Generation
Battery 1 is used to clear the load1 as well as the rest is given to the M2
finall battery1 value: 0
finall battery2 value: 0
```

Test Case 2

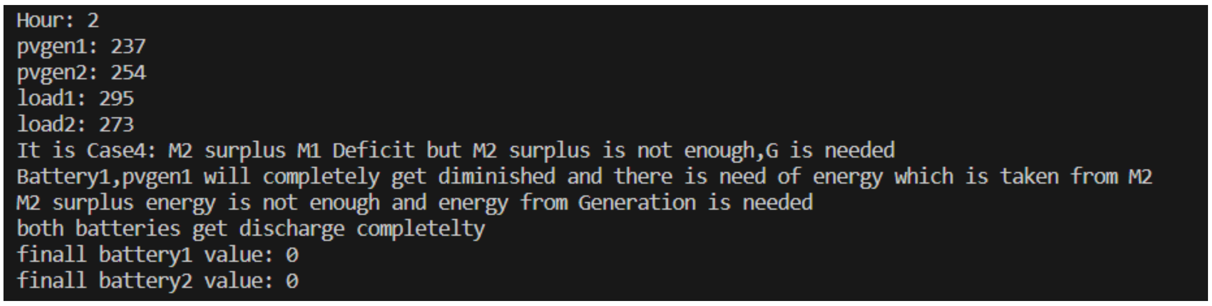


```
Hour: 22
pvgen1: 298
pvgen2: 220
load1: 273
load2: 286
It is Case2: M1 surplus M2 Deficit but M1 surplus is not enough,G is needed
Battery2,pvgen2 will completely get diminished and there is need of energy which is taken from M1
M1 surplus energy is not enough and energy from Generation is needed
both batteries get discharge completelty
finall battery1 value: 0
finall battery2 value: 0
```

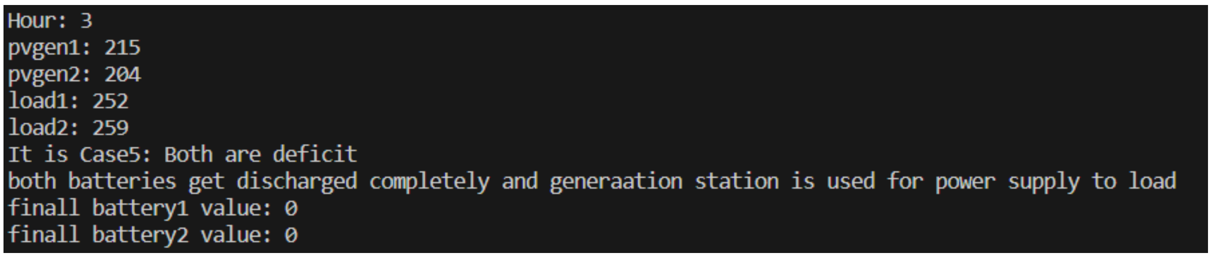
Test Case 3


```
Hour: 1
pvgen1: 155
pvgen2: 234
load1: 262
load2: 275
It is Case3: M2 surplus M1 Deficit and M2 surplus is enough
Battery1,pvgen1 will completely get diminished and there is need of energy which is taken from M2
M2 surplus energy is enough alone no need of energy from Generation
Battery 2 is used to clear the load2 as well as the rest is given to the M1
finall battery1 value: 0
finall battery2 value: 52
```

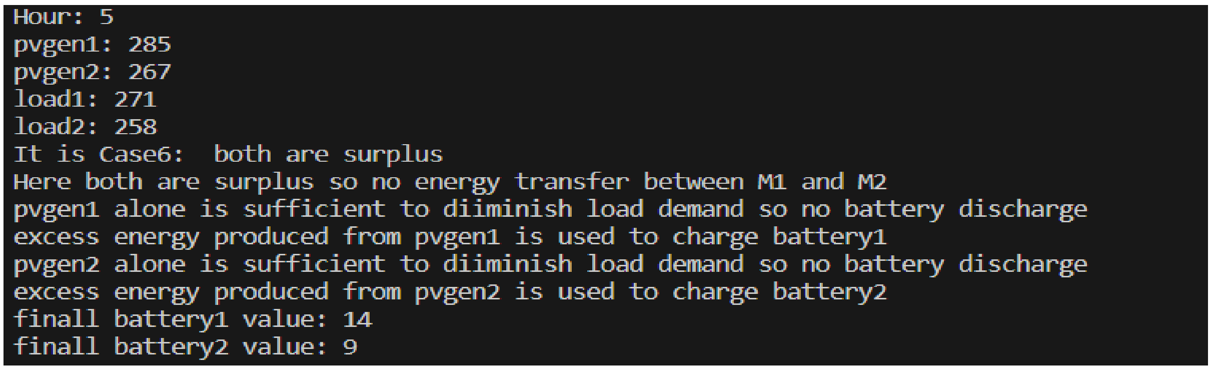
Test Case 4


```
Hour: 2
pvgen1: 237
pvgen2: 254
load1: 295
load2: 273
It is Case4: M2 surplus M1 Deficit but M2 surplus is not enough,G is needed
Battery1,pvgen1 will completely get diminished and there is need of energy which is taken from M2
M2 surplus energy is not enough and energy from Generation is needed
both batteries get discharge completelty
finall battery1 value: 0
finall battery2 value: 0
```

Test Case 5


```
Hour: 3
pvgen1: 215
pvgen2: 204
load1: 252
load2: 259
It is Case5: Both are deficit
both batteries get discharged completely and generaation station is used for power supply to load
finall battery1 value: 0
finall battery2 value: 0
```

Test Case 6


```
Hour: 5
pvgen1: 285
pvgen2: 267
load1: 271
load2: 258
It is Case6:  both are surplus
Here both are surplus so no energy transfer between M1 and M2
pvgen1 alone is sufficient to diiminish load demand so no battery discharge
excess energy produced from pvgen1 is used to charge battery1
pvgen2 alone is sufficient to diiminish load demand so no battery discharge
excess energy produced from pvgen2 is used to charge battery2
finall battery1 value: 14
finall battery2 value: 9
```

5.4 Bornholm Micro Grid Data

Data

We assumed two identical Micro Grids situated close to each other for simplicity.

The total values of PV Generation, the Wind Generation and the Load Profile was obtained[9].

The csv file of data are mentioned in the references.

Micro Grid Specifications and Assumptions:

- Area of Bornholm: 538 sq.km.
- We assumed 1/1000th area of Bornholm i.e. 0.538 sq,km.
- This is approximately 133 acres i.e. about 60% of our campus area.
- So the values of generation and load which are in MW are assumed to be in KW.
- The battery capacity is assumed to be 30 kWh.

Algorithm Specifications:

- The PV generation, wind generation, battery and the load demand values are used to design this algorithm using Python.
- Real time values from Bornholm island in Denmark are used to analyse the interconnection.
- The algorithm is developed for interconnection of two Microgrids prioritizing the critical loads in the area that is covered by the Microgrids.
- The critical load is assumed to be 20% of the total load in the area under observation.

This algorithm deals with four test cases explained below.

Test Cases

Test Case 1

Both Microgrid 1 and Microgrid 2 are surplus in nature.

Test Case 2

Both Microgrid 1 and Microgrid 2 are deficit in nature.

Test Case 3

Microgrid 1 is surplus and Microgrid 2 is deficit in nature.

Test Case 4

Microgrid 1 is deficit and Microgrid 2 is surplus in nature.

Code

```
import csv
from datetime import datetime

# Function to read CSV file and return data as a dictionary
def read_csv(filename):
    data = {}
    with open(filename, 'r') as file:
        reader = csv.reader(file)
        for row in reader:
            data[row[0]] = {'load': float(row[1]), 'pvgen':
float(row[2]), 'windgen': float(row[3])}
    return data

# Function to parse the date in different formats
def parse_date(date_str):
    parts = date_str.split('/')
    if len(parts) == 3:
        # Check if the input is in mm/dd/yyyy format
        try:
            date_obj = datetime.strptime(date_str,
"%m/%d/%Y")
        except ValueError:
            try:
                # Check if the input is in m/d/y format
                date_obj = datetime.strptime(date_str,
"%m/%d/%y")
            except ValueError:
                raise ValueError("Incorrect date format,
please use mm/dd/yyyy or m/d/y")
    else:
        raise ValueError("Incorrect date format, please use
mm/dd/yyyy or m/d/y")
    return date_obj.strftime("%m/%d/%Y")

# Function to calculate generation for each hour
def calculate_generation(grid1_data, grid2_data, date):
    # Initialize bat1 with a value of 25 kW
    bat1 = 30
    bat2= 0

    for hour in range(24):
        hour_str = f"{date} {hour:02d}:00"
        hour_str_no_zero = f"{date} {hour}:00"
        gen1 = 0  # Initialize gen1, gen2, load1, load2
here
        gen2 = 0
        load1 = 0
        load2 = 0
        if hour_str in grid1_data and hour_str in
grid2_data:
            gen1 = grid1_data[hour_str]['pvgen'] +
grid1_data[hour_str]['windgen']
            gen2 = grid2_data[hour_str]['pvgen'] +
grid2_data[hour_str]['windgen']
            load1 = grid1_data[hour_str]['load']
            load2 = grid2_data[hour_str]['load']

            # Perform your checks or calculations here
based on gen1, gen2, load1, and load2
            print(f"For {hour_str_no_zero}:")
            print(f"Gen1: {gen1}, Gen2: {gen2}, Load1:
{load1}, Load2: {load2}")

        print( )
        #Case1 if both grids are in surplus energy
        if( gen1>=load1 and gen2>=load2 ):
                print(" it is the Case1 where both grids
are in surplus energy ")
                print(" generation in microgrid1 will alone
clear the load1 ")
                print(" generation in microgrid2 will alone
clear the load2 ")
                gen1 = gen1 - load1
                gen2 = gen2 - load2
                if( gen1>0 and gen1 <=(30-bat1) and
bat1!=30 ):
                    print( f"left out gen1 power which is
{gen1} is used to charge the battery1" )
                    bat1 = bat1 + gen1
                elif( gen1>0 and gen1>(30-bat1) and
bat1!=30 ):
                    print( f"part of the left out gen1
power which is {30-bat1} is used to charge the battery1")
                    print( f"the rest of the gen1 power
which is {gen1-(30-bat1)} is transfered to the common
grid")
                    bat1=30
                elif(bat1==30 and gen1>0):
                    print( f"the gen1 power i.e {gen1} will
be transferred to the grid")
                print(f"the final value of battery1 at end
of the hour {hour} is {bat1}")

                if( gen2>0 and gen2 <=(30-bat2) and
bat2!=30):
                    print( f"left out gen2 power which is
{gen2} is used to charge the battery2" )
                    bat2 = bat2 + gen2
                elif( gen2>0 and gen2>(30-bat2) and
bat2!=30 ):
                    print( f"part of the left out gen2
power which is {30-bat2} is used to charge the battery2")
                    print( f"the rest of the gen2 power
which is {gen2-(30-bat2)} is transfered to the common
grid")
                    bat2=30
                elif(bat2==30 and gen2>0):
                    print( f"the gen2 power i.e {gen2} will
be transferred to the grid")
                print(f"the final value of battery2 at end
of the hour {hour} is {bat2}")

        #Case2 is both the grids are in deficit of energy
        elif ( gen1<load1 and gen2<load2 ):
                print("It is the Case2 where both the grids
are in deficit of energy")
                if( gen1<(0.2*load1)):
                    print(f" part of the priorityload1 
which is {gen1} will be cleared by gen1")
                    if(bat1==0):
                        print(f"left out part of 
priorityload1 which is {(0.2*load1)-gen1} and the non-
prioritylaod1({0.8*load1}) are shed")
                    elif(bat1<((0.2*load1)-gen1)):
                        print(f"battery1 will completely 
discharge by providing power of {((0.2*load1)-gen1)} to 
satisfy part of priorityload1")
                        print(f"rest of the priorityload1 
which is {((0.2*load1)-gen1)-bat1} will be shed and the 
non-priorityload1 power({0.8*load1}) also will be shed")
                        bat1=0
                    elif(bat1==((0.2*load1)-gen1)):
                        print(f"battery1 will completely 
discharge by providing power of {((0.2*load1)-gen1)} to 
satisfy priorityload1")
                        print(f"the non-priorityload1 
power({0.8*load1}) is supplied by the common grid")
                        bat1=0
                    elif(bat1>((0.2*load1)-gen1)):
                        print(f"part of the bat1 power 
which is {(0.2*load1)-gen1} will be used to satisfy 
priorityload1")
                        print(f"the non-prioritylaod1 which 
is {0.8*load1} will be shed")
                        bat1= bat1-((0.2*load1)-gen1)
                elif( gen1==(0.2*load1)):
                    print(f" gen1 will be completely used 
to satisfy the priorityload1 which is {0.2*load1}")
                    print(f"the non-priorityload1 which is 
{0.8*load1} is shed")
```

```
                elif( gen1>(0.2*load1)):
                    print(f"part of the gen1 which is 
{0.2*load1} will be used to clear priorityload1")
                    print(f"the rest of the part of 
gen1({gen1-(0.2*load1)}) is used to clear part of the non-
priorityload1")
                    print(f"the left part of non-
priorityload1({(0.8*load1)-(gen1-(0.2*load1))}) is shed")
                print(f"the final value of battery1 at end 
of the hour {hour} is {bat1}")

                if( gen2<(0.2*load2)):
                    print(f" part of the priorityload2 
which is {gen2} will be cleared by gen2")
                    if(bat2==0):
                        print(f"left out part of 
priorityload2 which is {(0.2*load2)-gen2} and the non-
prioritylaod2({0.8*load2}) is shed")
                    elif(bat2<((0.2*load2)-gen2)):
                        print(f"battery2 will completely 
discharge by providing power of {((0.2*load2)-gen2)} to 
satisfy part of priorityload2")
                        print(f"rest of the priorityload2 
which is {((0.2*load2)-gen2)-bat2} will be shed and the 
non-priorityload2 power({0.8*load2}) also will be shed")
                        bat2=0
                    elif(bat2==((0.2*load2)-gen2)):
                        print(f"battery2 will completely 
discharge by providing power of {((0.2*load2)-gen2)} to 
satisfy priorityload2")
                        print(f"the non-priorityload2 
power({0.8*load2}) is shed")
                        bat2=0
                    elif(bat2>((0.2*load2)-gen2)):
                        print(f"part of the bat2 power
which is {(0.2*load2)-gen2} will be used to satisfy
priorityload2")
                        print(f"the non-priorityload2 which
is {0.8*load2} will be shed")
                        bat2= bat2-((0.2*load2)-gen2)
                elif( gen2==(0.2*load2)):
                    print(f" gen2 will be completely used
to satisfy the priorityload2 which is {0.2*load2}")
                    print(f"the non-priorityload2 which is
{0.8*load2} is shed")
                elif( gen2>(0.2*load2)):
                    print(f"part of the gen2 which is
{0.2*load2} will be used to clear priorityload2")
                    print(f"the rest of the part of
gen2({gen2-(0.2*load2)}) is used to clear part of the non-
priorityload2")
                    print(f"the left part of non-
priorityload2({(0.8*load2)-(gen2-(0.2*load2))}) is shed")
                print(f"the final value of battery2 at end
of the hour {hour} is {bat2}")


        #Case3 Microgrid1 is surplus but microgrid2 is
deficit
        elif( gen1>load1):
                print(" it is the Case3 where Microgrid1 is
surplus but microgrid2 is deficit ")
                if((gen2>0.2*load2 and gen2<load2)
or(gen2<0.2*load2 and bat2==0) ):
                    print(f"the excess power in the
microgrid1({gen1-load1}) after satisfying the load1 will be
transmitted to microgrid2")
                    if((gen1-load1)<(load2-gen2)):
                        print(f"since excess power in
grid1({gen1-load1}) is less than the requirement in
grid2({load2-gen2}), part of left out of demand of grid 2
is satisfied by excess of grid1({gen1-load1})and the rest
of the part left i.e ({(load2-gen2)-(gen1-load1)}is shed))
")
                    elif((gen1-load1)==(load2-gen2)):
                        print(f"excess power in
grid1({gen1-load1}) is enough to satify the part of left
out of demand of grid 2 and is cleared. ")
                    elif((gen1-load1)>(load2-gen2)):
                        print(f" part of excess power in
grid1({load2-gen2}) is enough to satify the part of left
out of demand of grid 2 and is cleared. ")
                        if(bat1==30):
                            print(f"the left out part of
excess of gen1({(gen1-load1)-(load2-gen2)}) is transfered
to the commen grid")
                        elif(bat1>0 and bat1<30):
                            if(((gen1-load1)-(load2-
gen2))<=(30-bat1)):
                                print(f"the left out part
of excess of gen1({(gen1-load1)-(load2-gen2)}) is used to
store bat1 ")
                            elif(((gen1-load1)-(load2-
gen2))>(30-bat1)):
                                print(f"part of the left
out part of excess of gen1({30-bat1}) is used to store bat1
and the rest part({(((gen1-load1)-(load2-gen2))-(30-
bat1))})  is transfered to the common grid")
                elif((gen2<0.2*load2) and
((gen2+bat2)>0.2*load2)):
                    print(f"part of bat2({0.2*load2 -
gen2}) will be used to clear the priotrityload2")
                    if((gen1-load1)<(load2-gen2)):
                        print(f"since excess power in
grid1({gen1-load1}) is less than the requirement in
grid2({load2-gen2}), part of left out of demand of grid 2
is satisfied by excess of grid1({gen1-load1})and the rest
of the part left i.e ({(load2-gen2)-(gen1-load1)}is shed))
")
                    elif((gen1-load1)==(load2-gen2)):
                        print(f"excess power in
grid1({gen1-load1}) is enough to satify the part of left
out of demand of grid 2 and is cleared. ")
                    elif((gen1-load1)>(load2-gen2)):
                        print(f" part of excess power in
grid1({load2-gen2}) is enough to satify the part of left
out of demand of grid 2 and is cleared. ")
                        if(bat1==30):
                            print(f"the left out part of
excess of gen1({(gen1-load1)-(load2-gen2)}) is transfered
to the commen grid")
                        elif(bat1>0 and bat1<30):
                            if(((gen1-load1)-(load2-
gen2))<=(30-bat1)):
                                print(f"the left out part
of excess of gen1({(gen1-load1)-(load2-gen2)}) is used to
store bat1 ")
                            elif(((gen1-load1)-(load2-
gen2))>(30-bat1)):
                                print(f"part of the left
out part of excess of gen1({30-bat1}) is used to store bat1
and the rest part({(((gen1-load1)-(load2-gen2))-(30-
bat1))})  is transfered to the common grid")
                elif((gen2<0.2*load2) and
((gen2+bat2)==0.2*load2)):
                    print(f" bat2({0.2*load2 - gen2}) will
be used to clear the priotrityload2")
                    if((gen1-load1)<(0.8*load2)):
                        print(f"since excess power in
grid1({gen1-load1}) is less than non-
prioritylload2({0.8*load2}), part of non-prioritylload2 is
satisfied by excess of grid1({gen1-load1})and the rest of
the part left i.e ({(0.8*load2)-(gen1-load1)}is shed)) ")
                    elif((gen1-load1)==(0.8*load2)):
                        print(f"excess power in
grid1({gen1-load1}) is enough to satify the non-
prioritylload2 and is cleared. ")
                    elif((gen1-load1)>(0.8*load2)):
                        print(f" part of excess power in
grid1({0.8*load2}) is enough to satify the non-
prioritylload2 and is cleared. ")
                        if(bat1==30):
                            print(f"the left out part of
excess of gen1({(gen1-load1)-(0.8*load2)}) is transfered to
the commen grid")
                        elif(bat1>0 and bat1<30):
                            if(((gen1-load1)-
(0.8*load2))<=(30-bat1)):
                                print(f"the left out part
of excess of gen1({(gen1-load1)-(0.8*load2)}) is used to
store bat1 ")
                            elif(((gen1-load1)-
(0.8*load2))>(30-bat1)):
                                print(f"part of the left
out part of excess of gen1({30-bat1}) is used to store bat1
```

```
and the rest part({(((gen1-load1)-(0.8*load2))-(30-
bat1))})  is transfered to the common grid")
                elif((gen2<0.2*load2) and
((gen2+bat2)<0.2*load2)):
                    print(f" bat2({bat2}) will be used to
clear the part of priotrityload2")
                    if((gen1-load1)<((0.2*load2-
(gen2+bat2))+(0.8*load2))):
                        print(f"since excess power in
grid1({gen1-load1}) is less than the requirement in
grid2({(0.2*load2-(gen2+bat2))+(0.8*load2)}), part of left
out of demand of grid 2 is satisfied by excess of
grid1({gen1-load1})and the rest of the part left i.e
({((0.2*load2-(gen2+bat2))+(0.8*load2))-(gen1-load1)}is
shed)) ")
                    elif((gen1-load1)==((0.2*load2-
(gen2+bat2))+(0.8*load2))):
                        print(f"excess power in
grid1({gen1-load1}) is enough to satify the part of left
out of demand of grid 2 and is cleared. ")
                    elif((gen1-load1)>((0.2*load2-
(gen2+bat2))+(0.8*load2))):
                        print(f" part of excess power in
grid1({((0.2*load2-(gen2+bat2))+(0.8*load2))}) is enough to
satify the part of left out of demand of grid 2 and is
cleared. ")
                        if(bat1==30):
                            print(f"the left out part of
excess of gen1({(gen1-load1)-(((0.2*load2-
(gen2+bat2))+(0.8*load2)))}) is transfered to the common
grid")
                        elif(bat1>0 and bat1<30):
                            if(((gen1-load1)-(((0.2*load2-
(gen2+bat2))+(0.8*load2))))<=(30-bat1)):
                                print(f"the left out part
of excess of gen1({(gen1-load1)-(((0.2*load2-
(gen2+bat2))+(0.8*load2)))}) is used to store bat1 ")
                            elif(((gen1-load1)-((0.2*load2-
(gen2+bat2))+(0.8*load2)))>(30-bat1)):
                                print(f"part of the left
out part of excess of gen1({30-bat1}) is used to store bat1
and the rest part({(((gen1-load1)-(((0.2*load2-
(gen2+bat2))+(0.8*load2))))-(30-bat1))})  is transfered to
the common grid")
                print(f"the final value of battery1 at end
of the hour {hour} is {bat1}")
                print(f"the final value of battery2 at end
of the hour {hour} is {bat2}")


        #Case4 Microgrid2 is surplus but microgrid1 is
deficit
        elif( gen2>load2):
                print(" it is the Case4 where Microgrid2 is
surplus but microgrid1 is deficit ")
                if((gen1>0.2*load1 and gen1<load1)
or(gen1<0.2*load1 and bat1==0) ):
                    print(f"the excess power in the
microgrid2({gen2-load2}) after satisfying the load2 will be
transmitted to microgrid1")
                    if((gen2-load2)<(load1-gen1)):
                        print(f"since excess power in
grid2({gen2-load2}) is less than the requirement in
grid1({load1-gen1}), part of left out of demand of grid1 is
satisfied by excess of grid2({gen2-load2})and the rest of
the part left i.e ({(load1-gen1)-(gen2-load2)}is shed)) ")
```

```
                    if((gen2-load2)==(load1-gen1)):
                        print(f"excess power in
grid2({gen2-load2}) is enough to satify the part of left
out of demand of grid1 and is cleared. ")
                    if((gen2-load2)>(load1-gen1)):
                        print(f" part of excess power in
grid2({load1-gen1}) is enough to satify the part of left
out of demand of grid1 and is cleared. ")
                        if(bat2==30):
                            print(f"the left out part of
excess of gen2({(gen2-load2)-(load1-gen1)}) is transfered
to the common grid")
                        if(bat2>0 and bat2<30):
                            if(((gen2-load2)-(load1-
gen1))<=(30-bat2)):
                                print(f"the left out part
of excess of gen2({(gen2-load2)-(load1-gen1)}) is used to
store bat2 ")
                            if(((gen2-load2)-(load1-
gen1))>(30-bat2)):
                                print(f"part of the left
out part of excess of gen2({30-bat2}) is used to store bat2
and the rest part({(((gen2-load2)-(load1-gen1))-(30-
bat2))})  is transfered to the common grid")
                if((gen1<0.2*load1) and
((gen1+bat1)>0.2*load1)):
                    print(f"part of bat1({0.2*load1 -
gen1}) will be used to clear the priotrityload1")
                    if((gen2-load2)<(load1-gen1)):
                        print(f"since excess power in
grid2({gen2-load2}) is less than the requirement in
grid1({load1-gen1}), part of left out of demand of grid 1
is satisfied by excess of grid2({gen2-load2})and the rest
of the part left i.e ({(load1-gen1)-(gen2-load2)}is shed))
")
                    if((gen2-load2)==(load1-gen1)):
                        print(f"excess power in
grid2({gen2-load2}) is enough to satify the part of left
out of demand of grid 1 and is cleared. ")
                    if((gen2-load2)>(load1-gen1)):
                        print(f" part of excess power in
grid2({load1-gen1}) is enough to satify the part of left
out of demand of grid 1 and is cleared. ")
                        if(bat2==30):
                            print(f"the left out part of
excess of gen2({(gen2-load2)-(load1-gen1)}) is transfered
to the commen grid")
                        if(bat2>0 and bat2<30):
                            if(((gen2-load2)-(load1-
gen1))<=(30-bat2)):
                                print(f"the left out part
of excess of gen2({(gen2-load2)-(load1-gen1)}) is used to
store bat2 ")
                            if(((gen2-load2)-(load1-
gen1))>(30-bat2)):
                                print(f"part of the left
out part of excess of gen2({30-bat2}) is used to store bat2
and the rest part({(((gen2-load2)-(load1-gen1))-(30-
bat2))})  is transfered to the common grid")
                if((gen1<0.2*load1) and
((gen1+bat1)==0.2*load1)):
                    print(f" bat1({0.2*load1 - gen1}) will
be used to clear the priotrityload1")
                    if((gen2-load2)<(0.8*load1)):
                        print(f"since excess power in
grid2({gen2-load2}) is less than non-
```

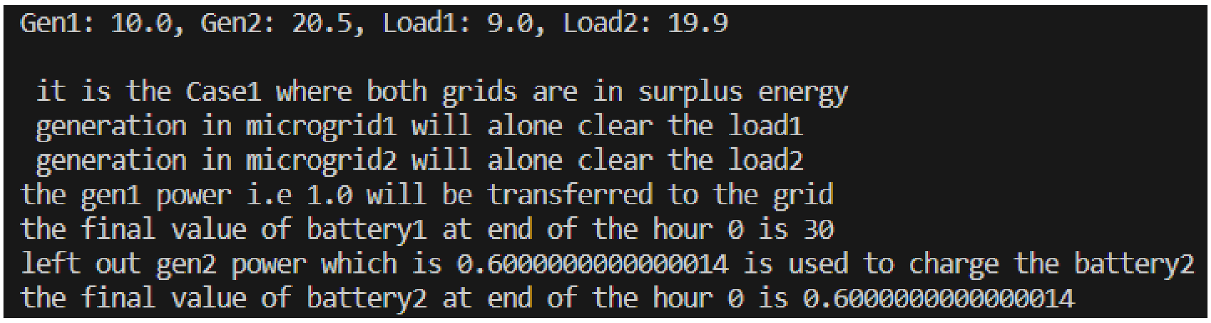
```
priorityload1({0.8*load1}), part of non-priorityload1 is
satisfied by excess of grid2({gen2-load2})and the rest of
the part left i.e ({(0.8*load1)-(gen2-load2)}is shed)) ")
                    if((gen2-load2)==(0.8*load1)):
                        print(f"excess power in
grid2({gen2-load2}) is enough to satify the non-
priorityload1 and is cleared. ")
                    if((gen2-load2)>(0.8*load1)):
                        print(f" part of excess power in
grid2({0.8*load1}) is enough to satify the non-
priorityload1 and is cleared. ")
                        if(bat2==30):
                            print(f"the left out part of
excess of gen2({(gen2-load2)-(0.8*load1)}) is transfered to
the common grid")
                        if(bat2>0 and bat2<30):
                            if(((gen2-load2)-
(0.8*load1))<=(30-bat2)):
                                print(f"the left out part
of excess of gen2({(gen2-load2)-(0.8*load1)}) is used to
store bat2 ")
                            if(((gen2-load2)-
(0.8*load1))>(30-bat2)):
                                print(f"part of the left
out part of excess of gen2({30-bat2}) is used to store bat2
and the rest part({(((gen2-load2)-(0.8*load1))-(30-
bat2))})  is transfered to the common grid")
                if((gen1<0.2*load1) and
((gen1+bat1)<0.2*load1)):
                    print(f" bat1({bat1}) will be used to
clear the part of priotrityload1")
                    if((gen2-load2)<((0.2*load1-
(gen1+bat1))+(0.8*load1))):
```

```
                        print(f"since excess power in
grid2({gen2-load2}) is less than the requirement in
grid1({(0.2*load1-(gen1+bat1))+(0.8*load1)}), part of left
out of demand of grid 1 is satisfied by excess of
grid2({gen2-load2})and the rest of the part left i.e
({((0.2*load1-(gen1+bat1))+(0.8*load1))-(gen2-load2)}is
shed)) ")
                    if((gen2-load2)==((0.2*load1-
(gen1+bat1))+(0.8*load1))):
                        print(f"excess power in
grid2({gen2-load2}) is enough to satify the part of left
out of demand of grid 1 and is cleared. ")
                    if((gen2-load2)>((0.2*load1-
(gen1+bat1))+(0.8*load1))):
                        print(f" part of excess power in
grid2({((0.2*load1-(gen1+bat1))+(0.8*load1))}) is enough to
satify the part of left out of demand of grid 1 and is
cleared. ")
                        if(bat2==30):
                            print(f"the left out part of
excess of gen2({(gen2-load2)-(((0.2*load1-
(gen1+bat1))+(0.8*load1)))}) is transfered to the common
grid")
                        if(bat2>0 and bat2<30):
                            if(((gen2-load2)-(((0.2*load1-
(gen1+bat1))+(0.8*load1))))<=(30-bat2)):
                                print(f"the left out part
of excess of gen2({(gen2-load2)-(((0.2*load1-
(gen1+bat1))+(0.8*load1)))}) is used to store bat2 ")
                            if(((gen2-load2)-((0.2*load1-
(gen1+bat1))+(0.8*load1)))>(30-bat2)):
                                print(f"part of the left
out part of excess of gen2({30-bat2}) is used to store bat2
```

```
and the rest part({(((gen2-load2)-(((0.2*load1-
(gen1+bat1))+(0.8*load1))))-(30-bat2))})  is transfered to
the common grid")
                print(f"the final value of battery1 at end
of the hour {hour} is {bat1}")
                print(f"the final value of battery2 at end
of the hour {hour} is {bat2}")
        print("-------------------------------------------
-----------------------------------------------------------
--------------------------------------------------")

# Main function
def main():
    date = input("Enter the date (format: mm/dd/yyyy or
m/d/y): ")
    date = parse_date(date)
    file_path1 = 'grid1.csv'  # Update the file path
    file_path2 = 'grid2.csv'  # Update the file path
    grid1_data = read_csv(file_path1)
    grid2_data = read_csv(file_path2)
    calculate_generation(grid1_data, grid2_data, date)

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

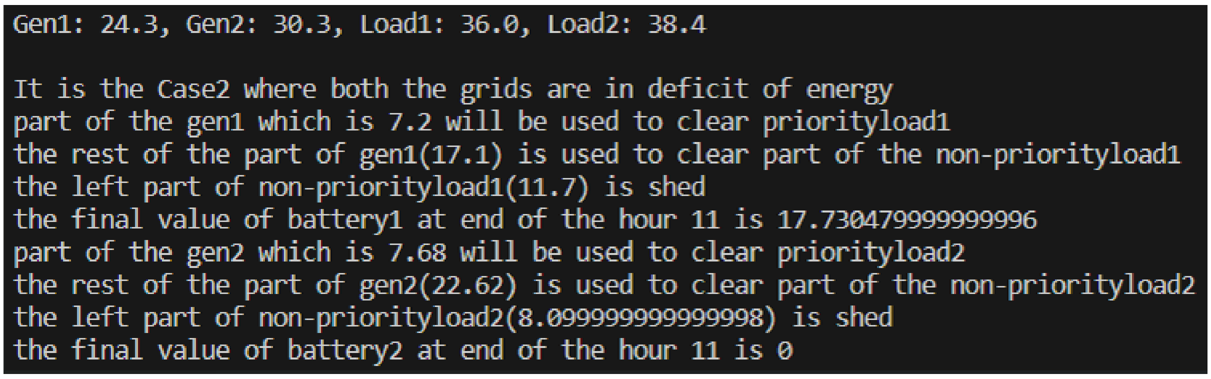
Output

Test Case 1



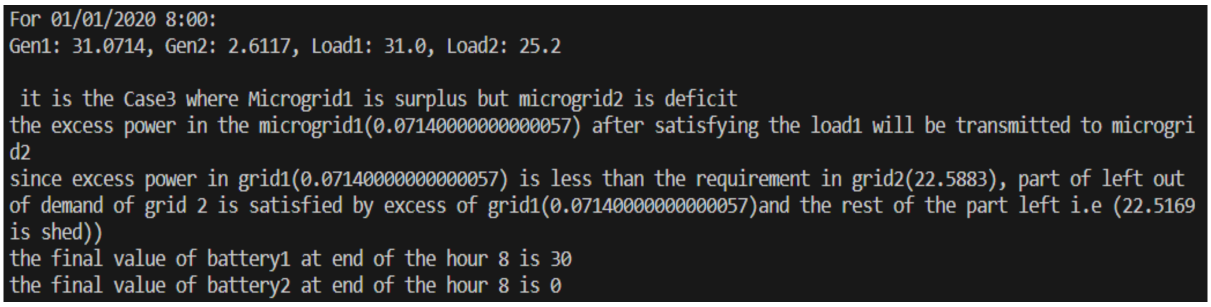
```
Gen1: 10.0, Gen2: 20.5, Load1: 9.0, Load2: 19.9

 it is the Case1 where both grids are in surplus energy
 generation in microgrid1 will alone clear the load1
 generation in microgrid2 will alone clear the load2
the gen1 power i.e 1.0 will be transferred to the grid
the final value of battery1 at end of the hour 0 is 30
left out gen2 power which is 0.6000000000000014 is used to charge the battery2
the final value of battery2 at end of the hour 0 is 0.6000000000000014
```

Test Case 2



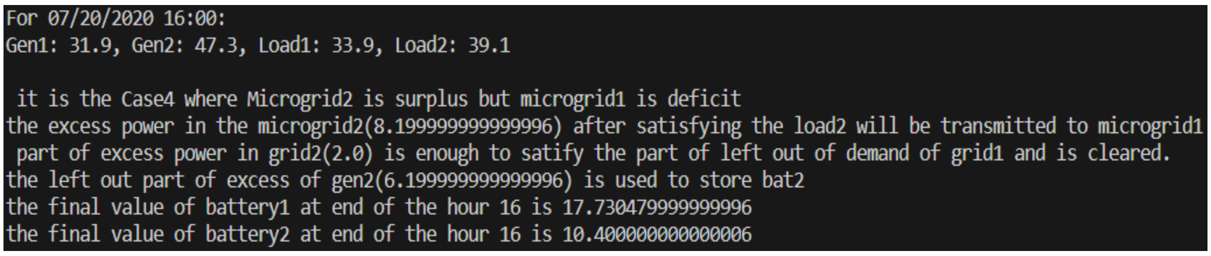
```
Gen1: 24.3, Gen2: 30.3, Load1: 36.0, Load2: 38.4

It is the Case2 where both the grids are in deficit of energy
part of the gen1 which is 7.2 will be used to clear prioritiload1
the rest of the part of gen1(17.1) is used to clear part of the non-priorityload1
the left part of non-priorityload1(11.7) is shed
the final value of battery1 at end of the hour 11 is 17.730479999999996
part of the gen2 which is 7.68 will be used to clear priorityload2
the rest of the part of gen2(22.62) is used to clear part of the non-priorityload2
the left part of non-priorityload2(8.099999999999998) is shed
the final value of battery2 at end of the hour 11 is 0
```

Test Case 3



```
For 01/01/2020 8:00:
Gen1: 31.0714, Gen2: 2.6117, Load1: 31.0, Load2: 25.2

 it is the Case3 where Microgrid1 is surplus but microgrid2 is deficit
the excess power in the microgrid1(0.07140000000000057) after satisfying the load1 will be transmitted to microgri
d2
since excess power in grid1(0.07140000000000057) is less than the requirement in grid2(22.5883), part of left out
of demand of grid 2 is satisfied by excess of grid1(0.07140000000000057)and the rest of the part left i.e (22.5169
is shed))
the final value of battery1 at end of the hour 8 is 30
the final value of battery2 at end of the hour 8 is 0
```

Test Case 4



```
For 07/20/2020 16:00:
Gen1: 31.9, Gen2: 47.3, Load1: 33.9, Load2: 39.1

 it is the Case4 where Microgrid2 is surplus but microgrid1 is deficit
the excess power in the microgrid2(8.199999999999996) after satisfying the load2 will be transmitted to microgrid1
 part of excess power in grid2(2.0) is enough to satify the part of left out of demand of grid1 and is cleared.
the left out part of excess of gen2(6.199999999999996) is used to store bat2
the final value of battery1 at end of the hour 16 is 17.730479999999996
the final value of battery2 at end of the hour 16 is 10.400000000000006
```

5.5 Interconnection of two Macro Grids

- Microgrid 1 is situated in United Kingdom.
- Microgrid 2 is situated in Netherlands.

- The distance between United Kingdom and Germany is 1023 km.
- Microgrids are connected through HVDC cables for power transfer.
- Transmission losses for HVDC lines are 3.5% for 1000km.

Data

- The Solar Generation, Wind Generation and Load Profile datasets were obtained from Kaggle Website.
  The links to which are shared in the references.
- The cost and investment values were obtained from the International Renewable Energy Agency (IRENA) website.

Table 5.2 Difference between Previous and Current Model

Previous Model	**Current Model**
• Single Microgrid (Bornholm) with different timestamps was used for the interconnection. • Transmission losses were assumed to be negligible. • Cost optimization was not performed for either grids. • Graphical representation was expected.	• Two different Microgrids (UK and Germany) are used. • Transmission losses are considered for 1. Overhead HVDC Transmission Line 2. Undersea HVDC Cable • Cost optimization is performed for both the grids. • Graphical Representation of Load, Solar Generation, Wind Generation, Battery, Losses and Power transfer was achieved.

Test Cases

- The Test Cases are Same as mentioned above in 5.4

Code

```
import csv
import matplotlib.pyplot as plt

# Function to read data from CSV file and filter by date
def read_data_by_date(csv_file, target_date):
    data = []
    with open(csv_file, 'r') as file:
        reader = csv.reader(file)
        headers = next(reader)
        for row in reader:
            if row[0].startswith(target_date):
                data.append(row)
    return headers, data

# Function to process data for a specific date
def process_data_for_date(headers, data):
    bat1 = 200
    bat2= 200
    # Lists to store data for plotting
    time_list = []
    M1_W_list = []
    M2_W_list = []
    M1_S_list = []
    M2_S_list = []
    load1_list = []
    load2_list = []
    bat1_list = []
    bat2_list = []
    M1PowerTransfered_list = []  # List to store time and
excess power
    M2PowerTransfered_list = []
    HvdcLoss_list = []
    UnderSeaLoss_list = []

    for row in data:
        print("Time:", row[0].split()[1])
        time = row[0].split()[1]
        M1_W, M1_S, load1, M2_W, M2_S, load2 = map(float,
row[1:])
        print(f"M1_W({M1_W}) M1_S({M1_S}) L1({load1})
M2_W({M2_W}) M2_S({M2_S}) L2({load2}) ")
        gen1 = M1_W+M1_S
        gen2 = M2_W+M2_S
        M1_PowTrs=0
        M2_PowTrs=0
        HVDCLoss = 0
        UndSeaLoss = 0
        # Append data to lists for plotting
        # Append data to lists for plotting
        time_list.append(time)
        M1_W_list.append(M1_W)
        M1_S_list.append(M1_S)
        M2_W_list.append(M2_W)
        M2_S_list.append(M2_S)
        load1_list.append(load1)
        load2_list.append(load2)
        bat1_list.append(bat1)
        bat2_list.append(bat2)

        #Case1 if both grids are in surplus energy
        if( gen1>=load1 and gen2>=load2 ):
                print(" it is the Case1 where both grids
are in surplus energy ")
                print(" generation in microgrid1 will alone
clear the load1 ")
                print(" generation in microgrid2 will alone
clear the load2 ")
                gen1 = gen1 - load1
                gen2 = gen2 - load2
                if( gen1>0 and gen1 <=(200-bat1) and
bat1!=200 ):
                    print( f"left out gen1 power which is
{gen1} is used to charge the battery1" )
                    bat1 = bat1 + gen1
                elif( gen1>0 and gen1>(200-bat1) and
bat1!=200 ):
                    print( f"part of the left out gen1
power which is {200-bat1} is used to charge the battery1")
                    print( f"the rest of the gen1 power
which is {gen1-(200-bat1)} is transfered to the common
grid")
                    bat1=200
                elif(bat1==200 and gen1>0):
                    print( f"the gen1 power i.e {gen1} will
be transferred to the grid")
                print(f"the final value of battery1 at the
time {row[0].split()[1]} is {bat1}")

                if( gen2>0 and gen2 <=(200-bat2) and
bat2!=200):
                    print( f"left out gen2 power which is
{gen2} is used to charge the battery2" )
                    bat2 = bat2 + gen2
                elif( gen2>0 and gen2>(200-bat2) and
bat2!=200 ):
                    print( f"part of the left out gen2
power which is {200-bat2} is used to charge the battery2")
```

```
                    print( f"the rest of the gen2 power 
which is {gen2-(200-bat2)} is transfered to the common 
grid")
                    bat2=200
                elif(bat2==200 and gen2>0):
                    print( f"the gen2 power i.e {gen2} will 
be transferred to the grid")
                print(f"the final value of battery2 at at 
the time {row[0].split()[1]} is {bat2}")

        #Case2 is both the grids are in deficit of energy
        elif ( gen1<load1 and gen2<load2 ):
                print("It is the Case2 where both the grids 
are in deficit of energy")
                if( gen1<(0.2*load1)):
                    print(f" part of the priorityload1 
which is {gen1} will be cleared by gen1")
                    if(bat1==0):
                        print(f"left out part of 
priorityload1 which is {(0.2*load1)-gen1} and the non-
prioritylaod1({0.8*load1}) are shed")
                    elif(bat1<((0.2*load1)-gen1)):
                        print(f"battery1 will completely 
discharge by providing power of {((0.2*load1)-gen1)} to 
satisfy part of priorityload1")
                        print(f"rest of the priorityload1 
which is {((0.2*load1)-gen1)-bat1} will be shed and the 
non-priorityload1 power({0.8*load1}) also will be shed")
                        bat1=0
                    elif(bat1==((0.2*load1)-gen1)):
                        print(f"battery1 will completely 
discharge by providing power of {((0.2*load1)-gen1)} to 
satisfy priorityload1")
```

```
                        print(f"the non-priorityload1
power({0.8*load1}) is supplied by the common grid")
                        bat1=0
                    elif(bat1>((0.2*load1)-gen1)):
                        print(f"part of the bat1 power
which is {(0.2*load1)-gen1} will be used to satisfy
priorityload1")
                        print(f"the non-prioritylaod1 which
is {0.8*load1} will be shed")
                        bat1= bat1-((0.2*load1)-gen1)
                elif( gen1==(0.2*load1)):
                    print(f" gen1 will be completely used
to satisfy the priorityload1 which is {0.2*load1}")
                    print(f"the non-priorityload1 which is
{0.8*load1} is shed")
                elif( gen1>(0.2*load1)):
                    print(f"part of the gen1 which is
{0.2*load1} will be used to clear priorityload1")
                    print(f"the rest of the part of
gen1({gen1-(0.2*load1)}) is used to clear part of the non-
priorityload1")
                    print(f"the left part of non-
priorityload1({(0.8*load1)-(gen1-(0.2*load1))}) is shed")
                print(f"the final value of battery1 at the
time {row[0].split()[1]} is {bat1}")

                if( gen2<(0.2*load2)):
                    print(f" part of the priorityload2
which is {gen2} will be cleared by gen2")
                    if(bat2==0):
                        print(f"left out part of
priorityload2 which is {(0.2*load2)-gen2} and the non-
prioritylaod2({0.8*load2}) is shed")
```

```
                    elif(bat2<((0.2*load2)-gen2)):
                        print(f"battery2 will completely
discharge by providing power of {((0.2*load2)-gen2)} to
satisfy part of priorityload2")
                        print(f"rest of the priorityload2
which is {((0.2*load2)-gen2)-bat2} will be shed and the
non-priorityload2 power({0.8*load2}) also will be shed")
                        bat2=0
                    elif(bat2==((0.2*load2)-gen2)):
                        print(f"battery2 will completely
discharge by providing power of {((0.2*load2)-gen2)} to
satisfy priorityload2")
                        print(f"the non-priorityload2
power({0.8*load2}) is shed")
                        bat2=0
                    elif(bat2>((0.2*load2)-gen2)):
                        print(f"part of the bat2 power
which is {(0.2*load2)-gen2} will be used to satisfy
priorityload2")
                        print(f"the non-priorityload2 which
is {0.8*load2} will be shed")
                        bat2= bat2-((0.2*load2)-gen2)
                elif( gen2==(0.2*load2)):
                    print(f" gen2 will be completely used
to satisfy the priorityload2 which is {0.2*load2}")
                    print(f"the non-priorityload2 which is
{0.8*load2} is shed")
                elif( gen2>(0.2*load2)):
                    print(f"part of the gen2 which is
{0.2*load2} will be used to clear priorityload2")
                    print(f"the rest of the part of
gen2({gen2-(0.2*load2)}) is used to clear part of the non-
priorityload2")
```

```
                    print(f"the left part of non-
priorityload2({(0.8*load2)-(gen2-(0.2*load2))}) is shed")
                print(f"the final value of battery2 at the
time {row[0].split()[1]} is {bat2}")


        #Case3 Microgrid1 is surplus but microgrid2 is
deficit
        elif(gen1>load1 and gen2<load2):
                print(" it is the Case3 where Microgrid1 is
surplus but microgrid2 is deficit ")

                if((gen2>0.2*load2) or(gen2<0.2*load2 and
bat2==0) ):
                    print(f"the excess power in the
microgrid1({(gen1-load1)}) after satisfying the load1 will
be transmitted to microgrid2")
                    if(((gen1-load1)*0.965)<(load2-gen2)):
                        print(f"since excess power in
grid1({(gen1-load1)*0.965}) is less than the requirement in
grid2({load2-gen2}), part of left out")
                        print(f"of demand of grid 2 is
satisfied by excess of grid1({(gen1-load1)*0.965})and the
rest of the part left i.e ({(load2-gen2)-((gen1-
load1)*0.965)}is shed)) ")
                        print(f"HVDC Transmission Power
Loss is ({(gen1-load1)*0.035})")
                        print(f"UnderseaCable Power Loss is
({(gen1-load1)*0.050})")
                        HVDCLoss = (gen1-load1)*0.035
                        UndSeaLoss = (gen1-load1)*0.050
                        M1_PowTrs= (-((gen1-load1)))
                        M2_PowTrs= (((gen1-load1)*0.965))
                    elif(((gen1-load1)*0.965)==(load2-
gen2)):
                        print(f"excess power in
grid1({(gen1-load1)*0.965}) is enough to satify the part of
left out of demand of grid 2 and is cleared. ")
                        print(f"HVDC Transmission Power
Loss is ({(gen1-load1)*0.035})")
                        print(f"UnderseaCable Power Loss is
({(gen1-load1)*0.050})")
                        HVDCLoss = (gen1-load1)*0.035
                        UndSeaLoss = (gen1-load1)*0.050
                        M1_PowTrs= (-((gen1-load1)))
                        M2_PowTrs= (((gen1-load1)*0.965))

                    elif(((gen1-load1))>((load2-
gen2)*1.03626)):
                        print(f" part of excess power in
grid1({(load2-gen2)*1.03626}) is enough to satify the part
of left out of demand of grid 2 and is cleared. ")
                        print(f"HVDC Transmission Power
Loss is ({((load2-gen2)*1.03626)*0.035})")
                        print(f"UnderSeaCable Power Loss is
({((load2-gen2)*1.03626)*0.050})")
                        HVDCLoss = ((load2-
gen2)*1.03626)*0.035
                        UndSeaLoss = ((load2-
gen2)*1.03626)*0.050
                        M1_PowTrs= (-((load2-
gen2)*1.03626))
                        M2_PowTrs= (((load2-gen2)))
                        if(bat1==200):
                            print(f"the left out part of
excess of gen1({(gen1-load1)-((load2-gen2)*1.03626)}) is
transfered to the commen grid")
                        elif(bat1>0 and bat1<200):
                            if(((gen1-load1)-((load2-
gen2)*1.03626))<=(200-bat1)):
                                print(f"the left out part
of excess of gen1({(gen1-load1)-((load2-gen2)*1.03626)}) is
used to store bat1 ")
                            elif(((gen1-load1)-((load2-
gen2)*1.03626))>(200-bat1)):
                                print(f"part of the left
out part of excess of gen1({200-bat1}) is used to store
bat1 and the rest part({(((gen1-load1)-((load2-
gen2)*1.03626))-(200-bat1))})  is transfered to the common
grid")

                elif((gen2<0.2*load2) and
((gen2+bat2)>0.2*load2)):
                    print(f"part of bat2({0.2*load2 -
gen2}) will be used to clear the priotrityload2")
                    if((gen1-load1)<(load2-gen2)):
                        print(f"since excess power in
grid1({gen1-load1}) is less than the requirement in
grid2({load2-gen2}), part of left out of demand of grid 2
is satisfied by excess of grid1({(gen1-load1)*0.965})and
the rest of the part left i.e ({(load2-gen2)-((gen1-
load1)*0.965)}is shed)) ")
                        M1_PowTrs= (-((gen1-load1)))
                        M2_PowTrs= (((gen1-load1)*0.965))
                        print(f"HVDC Transmission Power
Loss is ({(gen1-load1)*0.035})")
                        print(f"UnderseaCable Power Loss is
({(gen1-load1)*0.050})")
```

```
                        HVDCLoss = (gen1-load1)*0.035
                        UndSeaLoss = (gen1-load1)*0.050

                    elif(((gen1-load1)*0.965)==(load2-
gen2)):
                        print(f"excess power in
grid1({(gen1-load1)*0.965}) is enough to satify the part of
left out of demand of grid 2 and is cleared. ")
                        M1_PowTrs= (-((gen1-load1)))
                        M2_PowTrs= (((gen1-load1)*0.965))
                        print(f"HVDC Transmission Power
Loss is ({(gen1-load1)*0.035})")
                        print(f"UnderseaCable Power Loss is
({(gen1-load1)*0.050})")
                        HVDCLoss = (gen1-load1)*0.035
                        UndSeaLoss = (gen1-load1)*0.050

                    elif((gen1-load1)>((load2-
gen2)*1.03626)):
                        print(f" part of excess power in
grid1({((load2-gen2)*1.03626)}) is enough to satify the
part of left out of demand of grid 2 and is cleared. ")
                        M1_PowTrs= (-((load2-
gen2)*1.03626))
                        M2_PowTrs= (((load2-gen2)))
                        print(f"HVDC Transmission Power
Loss is ({((load2-gen2)*1.03626)*0.035})")
                        print(f"UnderSeaCable Power Loss is
({((load2-gen2)*1.03626)*0.050})")
                        HVDCLoss = ((load2-
gen2)*1.03626)*0.035
                        UndSeaLoss = ((load2-
gen2)*1.03626)*0.050
```

```
                        if(bat1==200):
                            print(f"the left out part of
excess of gen1({(gen1-load1)-(((load2-gen2)*1.03626))}) is
transfered to the commen grid")
                        elif(bat1>0 and bat1<200):
                            if(((gen1-load1)-(((load2-
gen2)*1.03626)))<=(200-bat1)):
                                print(f"the left out part
of excess of gen1({(gen1-load1)-(((load2-gen2)*1.03626))})
is used to store bat1 ")
                            elif(((gen1-load1)-(((load2-
gen2)*1.03626)))>(200-bat1)):
                                print(f"part of the left
out part of excess of gen1({200-bat1}) is used to store
bat1 and the rest part({(((gen1-load1)-(((load2-
gen2)*1.03626)))-(200-bat1))})  is transfered to the common
grid")


                elif((gen2<0.2*load2) and
((gen2+bat2)==0.2*load2)):
                    print(f" bat2({0.2*load2 - gen2}) will
be used to clear the priotrityload2")

                    if(((gen1-load1)*0.965)<(0.8*load2)):
                        print(f"since excess power in
grid1({(gen1-load1)*0.965}) is less than non-
priorityload2({0.8*load2}), part of non-priorityload2 is
satisfied by excess of grid1({(gen1-load1)*0.965})and the
rest of the part left i.e ({(0.8*load2)-((gen1-
load1)*0.965)}is shed)) ")
                        print(f"HVDC Transmission Power
Loss is ({(gen1-load1)*0.035})")
                                print(f"UnderSeaCable Power Loss is
({(gen1-load1)*0.050})")
                                HVDCLoss = (gen1-load1)*0.035
                                UndSeaLoss = (gen1-load1)*0.050
                                M1_PowTrs= (-((gen1-load1)))
                                M2_PowTrs= (((gen1-load1)*0.965))

                           elif(((gen1-
load1)*0.965)==(0.8*load2)):
                                print(f"excess power in
grid1({(gen1-load1)*0.965}) is enough to satisfy the non-
priorityload2 and is cleared. ")
                                print(f"HVDC Transmission Power
Loss is ({(gen1-load1)*0.035})")
                                print(f"UnderSeaCable Power Loss is
({(gen1-load1)*0.050})")
                                HVDCLoss = (gen1-load1)*0.035
                                UndSeaLoss = (gen1-load1)*0.050
                                M1_PowTrs= (-((gen1-load1)))
                                M2_PowTrs= (((gen1-load1)*0.965))

                           elif((gen1-
load1)>((0.8*load2)*1.03626)):
                                print(f" part of excess power in
grid1({(0.8*load2)*1.03626}) is enough to satisfy the non-
priorityload2 and is cleared. ")
                                M1_PowTrs= (-((0.8*load2)*1.03626))
                                M2_PowTrs= ((((0.8*load2)))
                                print(f"HVDC Transmission Power
Loss is ({((0.8*load2)*1.03626)*0.035})")
                                print(f"UnderSeaCable Power Loss is
({((0.8*load2)*1.03626)*0.050})")
```

```
                                HVDCLoss =
((0.8*load2)*1.03626)*0.035
                                UndSeaLoss =
((0.8*load2)*1.03626)*0.050

                                if(bat1==200):
                                    print(f"the left out part of
excess of gen1({(gen1-load1)-((0.8*load2)*1.03626)}) is
transfered to the commen grid")
                                elif(bat1>0 and bat1<200):
                                    if(((gen1-load1)-
((0.8*load2)*1.03626))<=(200-bat1)):
                                        print(f"the left out part
of excess of gen1({(gen1-load1)-((0.8*load2)*1.03626)}) is
used to store bat1 ")
                                    elif(((gen1-load1)-
((0.8*load2)*1.03626))>(200-bat1)):
                                        print(f"part of the left
out part of excess of gen1({200-bat1}) is used to store
bat1 and the rest part({(((gen1-load1)-
((0.8*load2)*1.03626))-(200-bat1))})  is transfered to the
common grid")

                        elif((gen2<0.2*load2) and
((gen2+bat2)<0.2*load2)):
                            print(f" bat2({bat2}) will be used to
clear the part of priotrityload2")
                            if((gen1-load1)*0.965)<((0.2*load2-
(gen2+bat2))+(0.8*load2)):
                                print(f"since excess power in
grid1({(gen1-load1)*0.965}) is less than the requirement in
grid2({(0.2*load2-(gen2+bat2))+(0.8*load2)}), part of left
out of demand of grid 2 is satisfied by excess of
grid1({(gen1-load1)*0.965})and the rest of the part left
i.e ({((0.2*load2-(gen2+bat2))+(0.8*load2))-((gen1-
load1)*0.965)}is shed))  ")
                                print(f"HVDC Transmission Power
Loss is ({(gen1-load1)*0.035})")
                                print(f"UnderSeaCable Power Loss is
({(gen1-load1)*0.050})")
                                HVDCLoss =
((0.8*load2)*1.03626)*0.035
                                UndSeaLoss =
((0.8*load2)*1.03626)*0.050
                                M1_PowTrs= (-((gen1-load1)))
                                M2_PowTrs= (((gen1-load1)*0.965))

                            elif(((gen1-load1)*0.965)==((0.2*load2-
(gen2+bat2))+(0.8*load2))):
                                print(f"excess power in
grid1({(gen1-load1)*0.9651}) is enough to satify the part
of left out of demand of grid 2 and is cleared. ")
                                print(f"HVDC Transmission Power
Loss is ({(gen1-load1)*0.035})")
                                print(f"UnderSeaCable Power Loss is
({(gen1-load1)*0.530})")
                                HVDCLoss =
((0.8*load2)*1.03626)*0.035
                                UndSeaLoss =
((0.8*load2)*1.03626)*0.050
                                M1_PowTrs= (-((gen1-load1)))
                                M2_PowTrs= (((gen1-load1)*0.965))

                            elif((gen1-load1)>(((0.2*load2-
(gen2+bat2))+(0.8*load2))*1.03626)):
                                print(f" part of excess power in
grid1({((0.2*load2-(gen2+bat2))+(0.8*load2))*1.03626}) is
enough to satify the part of left out of demand of grid 2
and is cleared. ")
                        M1_PowTrs= (-(((0.2*load2-
(gen2+bat2))+(0.8*load2))*1.03626))
                        M2_PowTrs= ((((0.2*load2-
(gen2+bat2))+(0.8*load2))))
                        print(f"HVDC Transmission Power
Loss is ({((0.2*load2-
(gen2+bat2))+(0.8*load2))*1.03626*0.035})")
                        print(f"UnderSeaCable Power Loss is
({((0.2*load2-(gen2+bat2))+(0.8*load2))*1.03626*0.050})")
                        HVDCLoss = ((0.2*load2-
(gen2+bat2))+(0.8*load2))*1.03626*0.035
                        UndSeaLoss = ((0.2*load2-
(gen2+bat2))+(0.8*load2))*1.03626*0.050
                        if(bat1==200):
                            print(f"the left out part of
excess of gen1({(gen1-load1)-(((0.2*load2-
(gen2+bat2))+(0.8*load2))*1.03626)}) is transfered to the
common grid")
                        elif(bat1>0 and bat1<200):
                            if(((gen1-load1)-(((0.2*load2-
(gen2+bat2))+(0.8*load2))*1.03626))<=(200-bat1)):
                                print(f"the left out part
of excess of gen1({(gen1-load1)-(((0.2*load2-
(gen2+bat2))+(0.8*load2))*1.03626)}) is used to store bat1
")
                            elif(((gen1-load1)-
(((0.2*load2-(gen2+bat2))+(0.8*load2))*1.03626))>(200-
bat1)):
                                print(f"part of the left
out part of excess of gen1({200-bat1}) is used to store
```

```
bat1 and the rest part({(((gen1-load1)-((((0.2*load2-
(gen2+bat2))+(0.8*load2))*1.03626)))-(200-bat1))})  is
transfered to the common grid")
                print(f"the final value of battery1 at the
time {row[0].split()[1]} is {bat1}")
                print(f"the final value of battery2 at the
time {row[0].split()[1]} is {bat2}")


        #Case4 Microgrid2 is surplus but microgrid1 is
deficit
        elif(gen2>load2 and gen1<load1):
                print(" it is the Case4 where Microgrid2 is
surplus but microgrid1 is deficit ")

                if((gen1>0.2*load1) or(gen1<0.2*load1 and
bat1==0) ):
                    print(f"the excess power in the
microgrid2({(gen2-load2)}) after satisfying the load2 will
be transmitted to microgrid1")
                    if(((gen2-load2)*0.965)<(load1-gen1)):
                        print(f"since excess power in
grid2({(gen2-load2)*0.965}) is less than the requirement in
grid1({load1-gen1}), part of left out of demand of grid 1
is satisfied by excess of grid2({(gen2-load2)*0.965})and
the rest of the part left i.e ({(load1-gen1)-((gen2-
load2)*0.965)}is shed)) ")
                        print(f"HVDC Transmission Power
Loss is ({(gen2-load2)*0.035})")
                        print(f"UnderSeaCable Power Loss is
({(gen2-load2)*0.050})")
                        HVDCLoss = (gen2-load2)*0.035
                        UndSeaLoss = (gen2-load2)*0.050
                        M2_PowTrs= (-((gen2-load2)))
                        M1_PowTrs= (((gen2-load2)*0.965))

                    elif(((gen2-load2)*0.965)==(load1-
gen1)):
                        print(f"excess power in
grid2({(gen2-load2)*0.965}) is enough to satify the part of
left out of demand of grid 1 and is cleared. ")
                        print(f"HVDC Transmission Power
Loss is ({(gen2-load2)*0.035})")
                        print(f"UnderSeaCable Power Loss is
({(gen2-load2)*0.050})")
                        HVDCLoss = (gen2-load2)*0.035
                        UndSeaLoss = (gen2-load2)*0.050
                        M2_PowTrs= (-((gen2-load2)))
                        M1_PowTrs= (((gen2-load2)*0.965))

                    elif(((gen2-load2))>((load1-
gen1)*1.03626)):
                        print(f" part of excess power in
grid2({(load1-gen1)*1.03626}) is enough to satify the part
of left out of demand of grid 1 and is cleared. ")
                        print(f"HVDC Transmission Power
Loss is ({((load1-gen1)*1.03626)*0.035})")
                        print(f"UnderSeaCable Power Loss is
({((load1-gen1)*1.03626)*0.050})")
                        HVDCLoss = ((load1-
gen1)*1.03626)*0.035
                        UndSeaLoss = ((load1-
gen1)*1.03626)*0.050
                        M2_PowTrs= (-((load1-
gen1)*1.03626))
                        M1_PowTrs= (((load1-gen1)))
```

```
                          if(bat2==200):
                              print(f"the left out part of
excess of gen2({(gen2-load2)-((load1-gen1)*1.03626)}) is
transfered to the commen grid")
                          elif(bat2>0 and bat2<200):
                              if(((gen2-load2)-((load1-
gen1)*1.03626))<=(200-bat2)):
                                  print(f"the left out part
of excess of gen2({(gen2-load2)-((load1-gen1)*1.03626)}) is
used to store bat2 ")
                              elif(((gen2-load2)-((load1-
gen1)*1.03626))>(200-bat2)):
                                  print(f"part of the left
out part of excess of gen2({200-bat2}) is used to store
bat2 and the rest part({(((gen2-load2)-((load1-
gen1)*1.03626))-(200-bat2))})  is transfered to the common
grid")

                  elif((gen1<0.2*load1) and
((gen1+bat1)>0.2*load1)):
                      print(f"part of bat1({0.2*load1 -
gen1}) will be used to clear the priotrityload1")
                      if((gen2-load2)<(load1-gen1)):
                          print(f"since excess power in
grid2({gen2-load2}) is less than the requirement in
grid1({load1-gen1}), part of left out of demand of grid 1
is satisfied by excess of grid2({(gen2-load2)*0.965})and
the rest of the part left i.e ({(load1-gen1)-((gen2-
load2)*0.965)}is shed)) ")
                          M2_PowTrs= (-((gen2-load2)))
                          M1_PowTrs= (((gen2-load2)*0.965))
                          print(f"HVDC Transmission Power
Loss is ({(gen2-load2)*0.035})")
```

```
                                print(f"UnderSeaCable Power Loss is
({(gen2-load2)*0.050})")
                                HVDCLoss = (gen2-load2)*0.035
                                UndSeaLoss = (gen2-load2)*0.050

                            elif(((gen2-load2)*0.965)==(load1-
gen1)):
                                print(f"excess power in
grid2({(gen2-load2)*0.965}) is enough to satisfy the part of
left out of demand of grid 1 and is cleared. ")
                                M2_PowTrs= (-((gen2-load2)))
                                M1_PowTrs= (((gen2-load2)*0.965))
                                print(f"HVDC Transmission Power
Loss is ({(gen2-load2)*0.035})")
                                print(f"UnderSeaCable Power Loss is
({(gen2-load2)*0.050})")
                                HVDCLoss = (gen2-load2)*0.035
                                UndSeaLoss = (gen2-load2)*0.050

                            elif((gen2-load2)>((load1-
gen1)*1.03626)):
                                print(f" part of excess power in
grid2({((load1-gen1)*1.03626)}) is enough to satify the
part of left out of demand of grid 1 and is cleared. ")
                                M2_PowTrs= (-((load1-
gen1)*1.03626))
                                M1_PowTrs= (((load1-gen1)))
                                print(f"HVDC Transmission Power
Loss is ({((load1-gen1)*1.03626)*0.035})")
                                print(f"UnderSeaCable Power Loss is
({((load1-gen1)*1.03626)*0.050})")
                                HVDCLoss = ((load1-
gen1)*1.03626)*0.035
```

```
                        UndSeaLoss = ((load1-
gen1)*1.03626)*0.050

                        if(bat2==200):
                            print(f"the left out part of
excess of gen2({(gen2-load2)-(((load1-gen1)*1.03626))}) is
transfered to the commen grid")
                        elif(bat2>0 and bat2<200):
                            if(((gen2-load2)-(((load1-
gen1)*1.03626)))<=(200-bat2)):
                                print(f"the left out part
of excess of gen2({(gen2-load2)-(((load1-gen1)*1.03626))})
is used to store bat2 ")
                            elif(((gen2-load2)-(((load1-
gen1)*1.03626)))>(200-bat2)):
                                print(f"part of the left
out part of excess of gen2({200-bat2}) is used to store
bat2 and the rest part({(((gen2-load2)-(((load1-
gen1)*1.03626)))-(200-bat2))})  is transfered to the common
grid")


                elif((gen1<0.2*load1) and
((gen1+bat1)==0.2*load1)):
                    print(f" bat1({0.2*load1 - gen1}) will
be used to clear the priotrityload1")

                    if(((gen2-load2)*0.965)<(0.8*load1)):
                        print(f"since excess power in
grid1({(gen2-load2)*0.965}) is less than non-
priorityload1({0.8*load1}), part of non-priorityload1 is
satisfied by excess of grid2({(gen2-load2)*0.965})and the
rest of the part left i.e ({(0.8*load1)-((gen2-
load2)*0.965)}is shed)) ")
                              print(f"HVDC Transmission Power
Loss is ({(gen2-load2)*0.035})")
                              print(f"UnderSeaCable Power Loss is
({(gen2-load2)*0.050})")
                              HVDCLoss = (gen2-load2)*0.035
                              UndSeaLoss = (gen2-load2)*0.050
                              M2_PowTrs= (-((gen2-load2)))
                              M1_PowTrs= (((gen2-load2)*0.965))

                         elif(((gen2-
load2)*0.965)==(0.8*load1)):
                              print(f"excess power in
grid2({(gen2-load2)*0.965}) is enough to satify the non-
priorityload1 and is cleared. ")
                              print(f"HVDC Transmission Power
Loss is ({(gen2-load2)*0.035})")
                              print(f"UnderSeaCable Power Loss is
({(gen2-load2)*0.050})")
                              HVDCLoss = (gen2-load2)*0.035
                              UndSeaLoss = (gen2-load2)*0.050
                              M2_PowTrs= (-((gen2-load2)))
                              M1_PowTrs= (((gen2-load2)*0.965))

                         elif((gen2-
load2)>((0.8*load1)*1.03626)):
                              print(f" part of excess power in
grid2({(0.8*load1)*1.03626}) is enough to satify the non-
priorityload1 and is cleared. ")
                              M2_PowTrs= (-((0.8*load1)*1.03626))
                              M1_PowTrs= (((0.8*load1)))
                              print(f"HVDC Transmission Power
Loss is ({((0.8*load1)*1.03626)*0.035})")
```

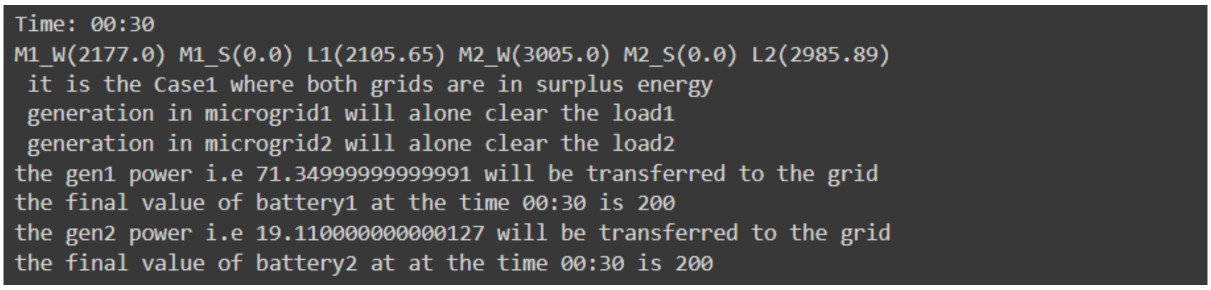
```
                        print(f"UnderSeaCable Power Loss is
({((0.8*load1)*1.03626)*0.050})")
                        HVDCLoss =
((0.8*load1)*1.03626)*0.035
                        UndSeaLoss =
((0.8*load1)*1.03626)*0.050
                        if(bat2==200):
                            print(f"the left out part of
excess of gen2({(gen2-load2)-((0.8*load1)*1.03626)}) is
transfered to the commen grid")
                        elif(bat2>0 and bat2<200):
                            if(((gen2-load2)-
((0.8*load1)*1.03626))<=(200-bat2)):
                                print(f"the left out part
of excess of gen2({(gen2-load2)-((0.8*load1)*1.03626)}) is
used to store bat2 ")
                            elif(((gen2-load2)-
((0.8*load1)*1.03626))>(200-bat2)):
                                print(f"part of the left
out part of excess of gen2({200-bat2}) is used to store
bat2 and the rest part({(((gen2-load2)-
((0.8*load1)*1.03626))-(200-bat2))})  is transfered to the
common grid")

                elif((gen1<0.2*load1) and
((gen1+bat1)<0.2*load1)):
                    print(f" bat1({bat1}) will be used to
clear the part of priotritylaod1")
                    if((gen2-load2)*0.965)<((0.2*load1-
(gen1+bat1))+(0.8*load1)):
                        print(f"since excess power in
grid2({(gen2-load2)*0.965}) is less than the requirement in
grid1({(0.2*load1-(gen1+bat1))+(0.8*load1)}), part of left
out of demand of grid 1 is satisfied by excess of
grid2({(gen2-load2)*0.965})and the rest of the part left
i.e ({((0.2*load1-(gen1+bat1))+(0.8*load1))-((gen2-
load2)*0.965)}is shed)) ")
                        print(f"HVDC Transmission Power
Loss is ({(gen2-load2)*0.035})")
                        print(f"UnderSeaCable Power Loss is
({(gen2-load2)*0.050})")
                        HVDCLoss = (gen2-load2)*0.035
                        UndSeaLoss = (gen2-load2)*0.050
                        M2_PowTrs= (-((gen2-load2)))
                        M1_PowTrs= (((gen2-load2)*0.965))

                    elif(((gen2-load2)*0.965)==((0.2*load1-
(gen1+bat1))+(0.8*load1))):
                        print(f"excess power in
grid1({(gen2-load2)*0.9651}) is enough to satify the part
of left out of demand of grid 1 and is cleared. ")
                        print(f"HVDC Transmission Power
Loss is ({(gen2-load2)*0.035})")
                        print(f"UnderSeaCable Power Loss is
({(gen2-load2)*0.050})")
                        HVDCLoss = (gen2-load2)*0.035
                        UndSeaLoss = (gen2-load2)*0.050
                        M2_PowTrs= (-((gen2-load2)))
                        M1_PowTrs= (((gen2-load2)*0.965))

                    elif((gen2-load2)>(((0.2*load1-
(gen1+bat1))+(0.8*load1))*1.03626)):
                        print(f" part of excess power in
grid2({((0.2*load1-(gen1+bat1))+(0.8*load1))*1.03626}) is
enough to satify the part of left out of demand of grid 1
and is cleared. ")
```

```
                        M2_PowTrs= (-((((0.2*load1-
(gen1+bat1))+(0.8*load1))*1.03626))
                        M1_PowTrs= ((((0.2*load1-
(gen1+bat1))+(0.8*load1))))
                        print(f"HVDC Transmission Power
Loss is ({((0.2*load1-
(gen1+bat1))+(0.8*load1))*1.03626*0.035})")
                        print(f"UnderSeaCable Power Loss is
({((0.2*load1-(gen1+bat1))+(0.8*load1))*1.03626*0.050})")
                        HVDCLoss = ((0.2*load1-
(gen1+bat1))+(0.8*load1))*1.03626*0.035
                        UndSeaLoss = ((0.2*load1-
(gen1+bat1))+(0.8*load1))*1.03626*0.050
                        if(bat1==200):
                            print(f"the left out part of
excess of gen2({(gen2-load2)-(((0.2*load1-
(gen1+bat1))+(0.8*load1))*1.03626)}) is transfered to the
common grid")
                        elif(bat2>0 and bat2<200):
                            if(((gen2-load2)-(((0.2*load1-
(gen1+bat1))+(0.8*load1))*1.03626))<=(200-bat2)):
                                print(f"the left out part
of excess of gen2({(gen2-load2)-(((0.2*load1-
(gen1+bat1))+(0.8*load1))*1.03626)}) is used to store bat2
")
                            elif(((gen2-load2)-
(((0.2*load1-(gen1+bat1))+(0.8*load1))*1.03626))>(200-
bat2)):
                                print(f"part of the left
out part of excess of gen2({200-bat2}) is used to store
bat2 and the rest part({(((gen2-load2)-((((0.2*load1-
(gen1+bat1))+(0.8*load1))*1.03626)))-(200-bat2))})  is
transfered to the common grid")
```

```
                print(f"the final value of battery1 at the 
time {row[0].split()[1]} is {bat1}")
                print(f"the final value of battery2 at the 
time {row[0].split()[1]} is {bat2}")
        M1PowerTransfered_list.append(M1_PowTrs)
        M2PowerTransfered_list.append(M2_PowTrs)
        HvdcLoss_list.append(HVDCLoss)
        UnderSeaLoss_list.append(UndSeaLoss)
        print("-------------------------------------------
-----------------------------------------------------------
--------------------------------------------------")

    # Plotting
    plt.figure(figsize=(30, 12))

    # Load vs Time
    plt.plot(time_list, load1_list, label='Load 1')
    plt.plot(time_list, load2_list, label='Load 2')
    plt.xlabel('Time')
    plt.ylabel('Load')
    plt.title('Load vs Time')
    plt.legend()
    plt.gca().set_aspect('auto', adjustable='box')
    plt.show()

    plt.figure(figsize=(30, 12))
    # M1_W and M2_W vs Time
    plt.plot(time_list, M1_W_list, label='M1_W')
    plt.plot(time_list, M2_W_list, label='M2_W')
    plt.xlabel('Time')
    plt.ylabel('Power')
    plt.title('M1_W and M2_W vs Time')
    plt.legend()
    plt.gca().set_aspect('auto', adjustable='box')
    plt.show()

    plt.figure(figsize=(30, 12))
    # M1_S and M2_S vs Time
    plt.plot(time_list, M1_S_list, label='M1_S')
    plt.plot(time_list, M2_S_list, label='M2_S')
    plt.xlabel('Time')
    plt.ylabel('Power')
    plt.title('M1_S and M2_S vs Time')
    plt.legend()
    plt.gca().set_aspect('auto', adjustable='box')
    plt.show()

    plt.figure(figsize=(30, 12))
    # Battery 1 and Battery 2 vs Time
    plt.plot(time_list, bat1_list, label='Battery 1')
    plt.plot(time_list, bat2_list, label='Battery 2')
    plt.xlabel('Time')
    plt.ylabel('Battery Capacity')
    plt.title('Battery 1 and Battery 2 vs Time')
    plt.legend()
    plt.gca().set_aspect('auto', adjustable='box')
    plt.show()

    plt.figure(figsize=(30, 12))
    # M1 Pow Trs and M2 Pow Trs vs Time
    plt.plot(time_list,  M1PowerTransfered_list,
label='M1')
    plt.plot(time_list,  M2PowerTransfered_list,
label='M2')
    plt.xlabel('Time')
    plt.ylabel('Battery Capacity')
    plt.title('M1_PowTrs and M2_PowTrs vs Time')
    plt.legend()
    plt.gca().set_aspect('auto', adjustable='box')
    plt.show()

    plt.figure(figsize=(30, 12))
    # LOSSES
    plt.plot(time_list,  HvdcLoss_list, label='HVDC Loss')
    plt.plot(time_list,  UnderSeaLoss_list,
label='UnderSeaCable Loss')
    plt.xlabel('Time')
    plt.ylabel('Losses')
    plt.title('Losses vs Time')
    plt.legend()
    plt.gca().set_aspect('auto', adjustable='box')
    plt.show()

# Main code
if __name__ == "__main__":
    target_date = input("Enter the date (format:
DD/MM/YYYY): ")
    csv_file = "f5.csv"
    headers, data = read_data_by_date(csv_file,
target_date)
    if data:
        process_data_for_date(headers, data)
    else:
        print("No data found for the given date.")
```

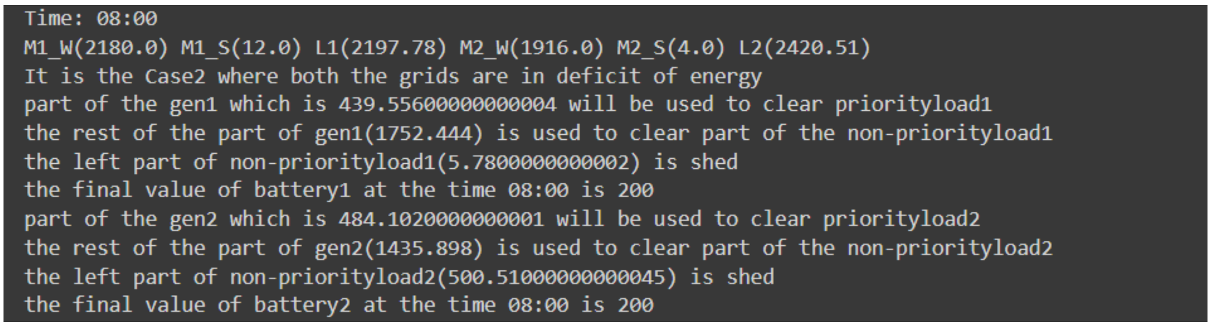
Output

Test Case 1



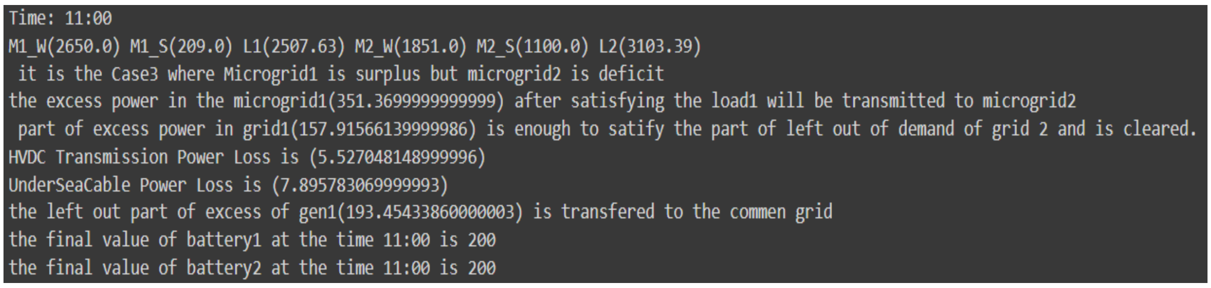
```
Time: 00:30
M1_W(2177.0) M1_S(0.0) L1(2105.65) M2_W(3005.0) M2_S(0.0) L2(2985.89)
 it is the Case1 where both grids are in surplus energy
 generation in microgrid1 will alone clear the load1
 generation in microgrid2 will alone clear the load2
the gen1 power i.e 71.34999999999991 will be transferred to the grid
the final value of battery1 at the time 00:30 is 200
the gen2 power i.e 19.110000000000127 will be transferred to the grid
the final value of battery2 at at the time 00:30 is 200
```

Test Case 2



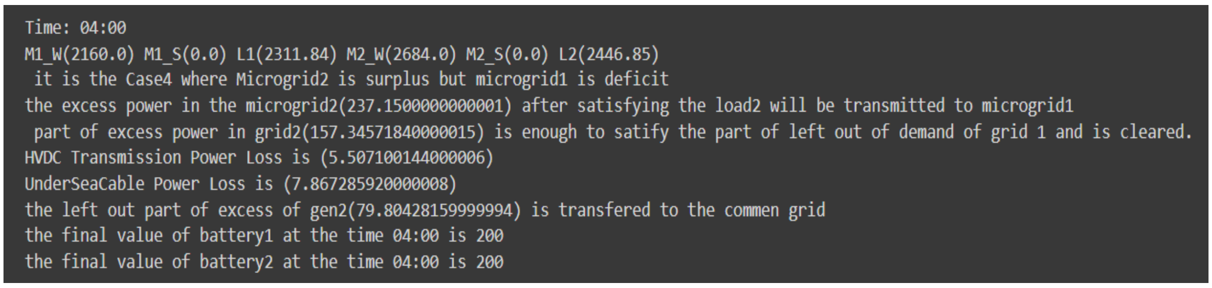
```
 Time: 08:00
 M1_W(2180.0) M1_S(12.0) L1(2197.78) M2_W(1916.0) M2_S(4.0) L2(2420.51)
 It is the Case2 where both the grids are in deficit of energy
 part of the gen1 which is 439.55600000000004 will be used to clear prioritload1
 the rest of the part of gen1(1752.444) is used to clear part of the non-priorityload1
 the left part of non-priorityload1(5.7800000000002) is shed
 the final value of battery1 at the time 08:00 is 200
 part of the gen2 which is 484.1020000000001 will be used to clear priorityload2
 the rest of the part of gen2(1435.898) is used to clear part of the non-priorityload2
 the left part of non-priorityload2(500.51000000000045) is shed
 the final value of battery2 at the time 08:00 is 200
```

Test Case 3



```
Time: 11:00
M1_W(2650.0) M1_S(209.0) L1(2507.63) M2_W(1851.0) M2_S(1100.0) L2(3103.39)
 it is the Case3 where Microgrid1 is surplus but microgrid2 is deficit
the excess power in the microgrid1(351.3699999999999) after satisfying the load1 will be transmitted to microgrid2
 part of excess power in grid1(157.91566139999986) is enough to satify the part of left out of demand of grid 2 and is cleared.
HVDC Transmission Power Loss is (5.527048148999996)
UnderSeaCable Power Loss is (7.895783069999993)
the left out part of excess of gen1(193.45433860000003) is transfered to the commen grid
the final value of battery1 at the time 11:00 is 200
the final value of battery2 at the time 11:00 is 200
```

Test Case 4



```
 Time: 04:00
 M1_W(2160.0) M1_S(0.0) L1(2311.84) M2_W(2684.0) M2_S(0.0) L2(2446.85)
  it is the Case4 where Microgrid2 is surplus but microgrid1 is deficit
 the excess power in the microgrid2(237.1500000000001) after satisfying the load2 will be transmitted to microgrid1
  part of excess power in grid2(157.34571840000015) is enough to satify the part of left out of demand of grid 1 and is cleared.
 HVDC Transmission Power Loss is (5.507100144000006)
 UnderSeaCable Power Loss is (7.867285920000008)
 the left out part of excess of gen2(79.80428159999994) is transfered to the commen grid
 the final value of battery1 at the time 04:00 is 200
 the final value of battery2 at the time 04:00 is 200
```

Plots

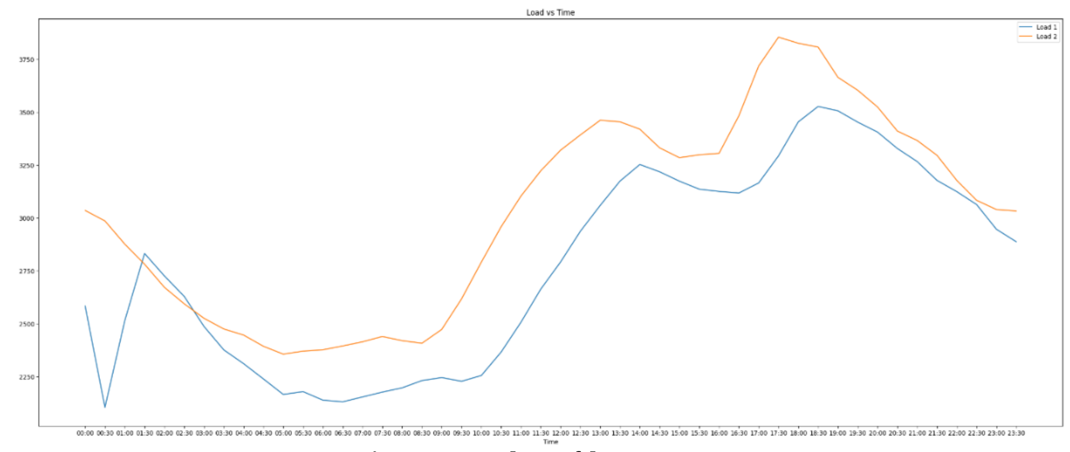


Fig 5.4:Load Profile vs. Time

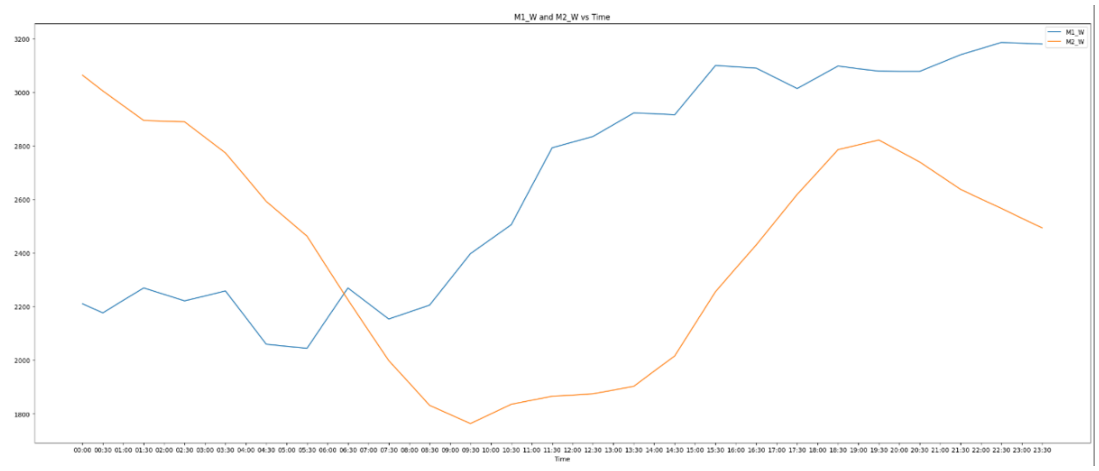


Fig 5.5: Wind Generation vs. Time

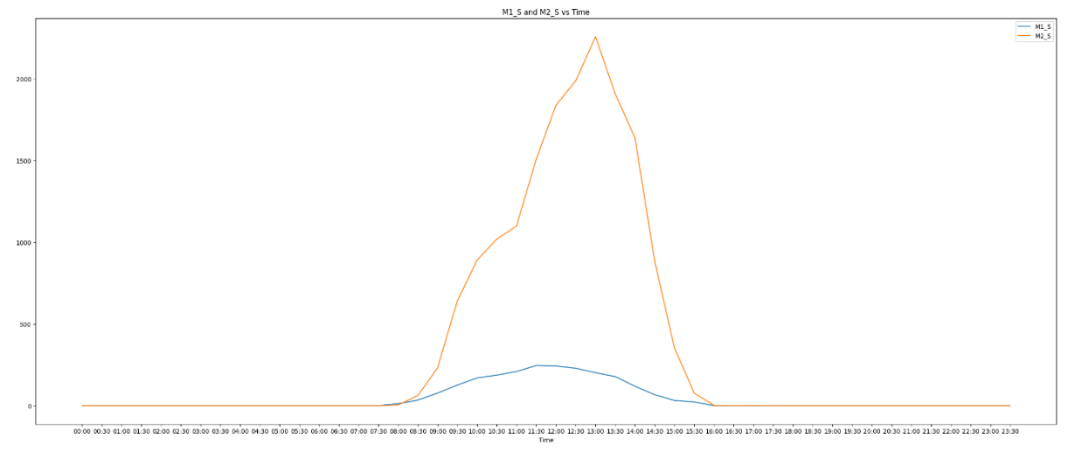


Fig 5.6: Solar Generation vs. Time

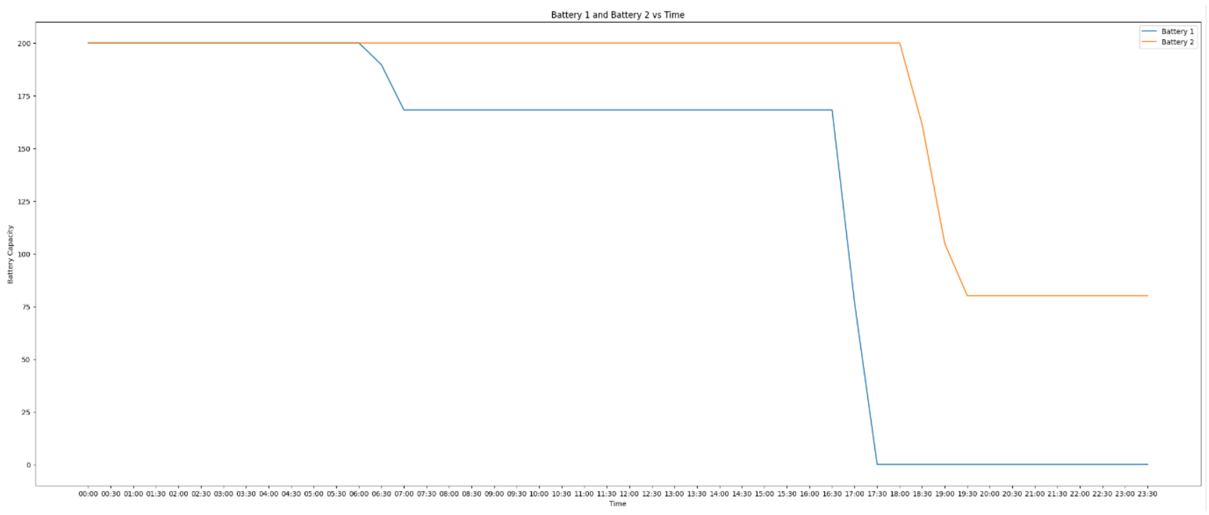


Fig 5.7: Battery Status vs. Time

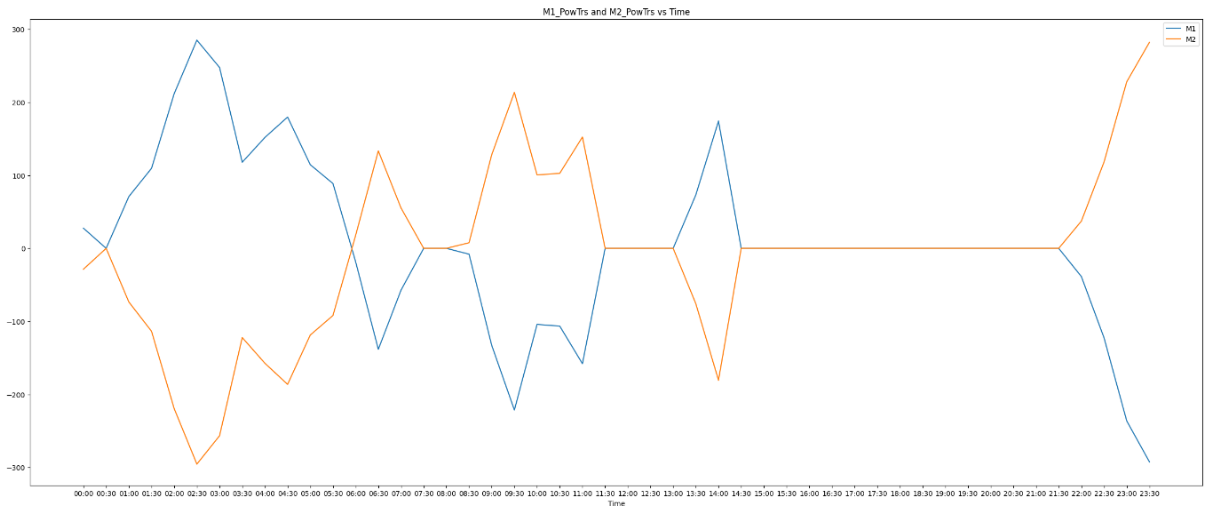


Fig 5.8: M1 <-> M2 Power Transfer vs. Time

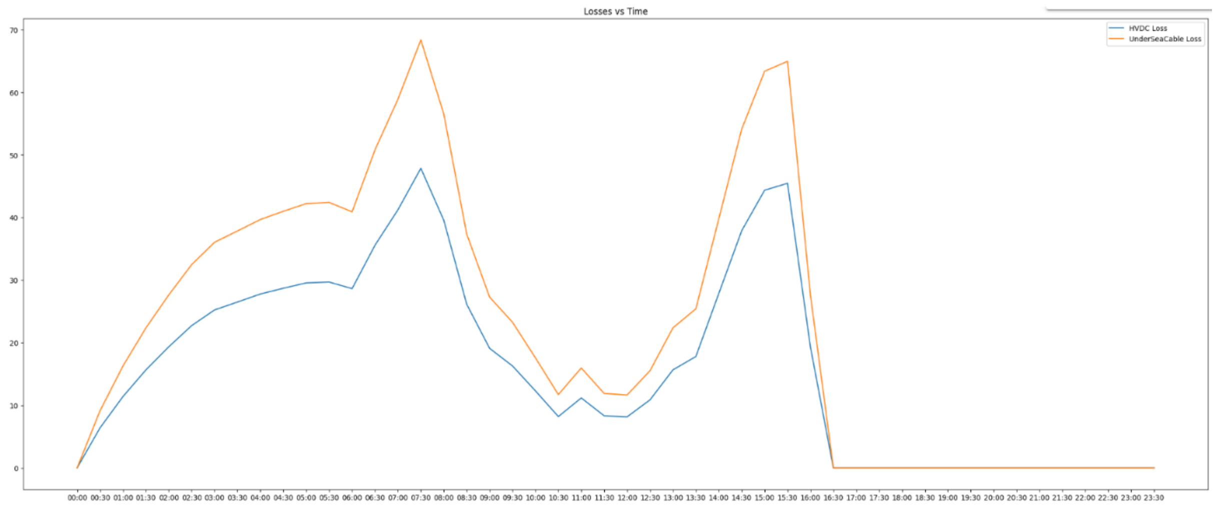


Fig 5.9: Losses vs. Time

CHAPTER 6

ONLINE SIMULATOR

6.1 Overview

The development of an online simulator for analyzing microgrid data represents a significant advancement in the accessibility and usability of energy systems simulation tools. This chapter details the creation of a web-based application designed to simulate and analyze the performance of microgrids based on renewable energy sources across different European countries. The simulator provides users with insights into wind and solar energy generation as well as load demands for selected countries.

6.2 Technologies Used

The online simulator is developed using a combination of modern web technologies that are well-suited for creating interactive and dynamic web applications. Here's an overview of the key technologies used:

1. React: A JavaScript library for building user interfaces, react allows for the development of complex interactive web applications with efficient updates and rendering of components. Reacts component-based architecture makes it ideal for this type of application, enabling modular and maintainable code.
2. JavaScript (ES6+): The primary programming language used; JavaScript enables the creation of dynamic web content. ES6 (ECMAScript 2015) introduces several features that make the code more robust and easier to read, such as arrow functions, classes, template literals, destructuring, and promises.
3. CSS: Used for styling the web application, CSS enhances the user interface by providing a pleasant visual experience, ensuring that the application is not only functional but also aesthetically appealing.
4. HTML5: The standard markup language for creating web pages. HTML5 elements are used to structure the web application and integrate media content efficiently.

6.3 Application Architecture

The simulator is structured around React components, with each component handling specific parts of the application's UI and logic:

- App Component: Serves as the root component that encapsulates all other sub-components. It manages the global state of the application, including selected countries, date inputs, and simulation data.
- File Upload Component: Manages the uploading of Excel files and parsing them using the XLSX library. It extracts necessary data for simulation.
- Date Selection Component: Allows users to select a specific date for which the simulation should run.
- Country Selection Component: Enables users to select up to two countries for which the simulation will compare data.
- Simulation Output Component: Displays the results of the simulation, including time-series data for wind and solar generation and load demands.

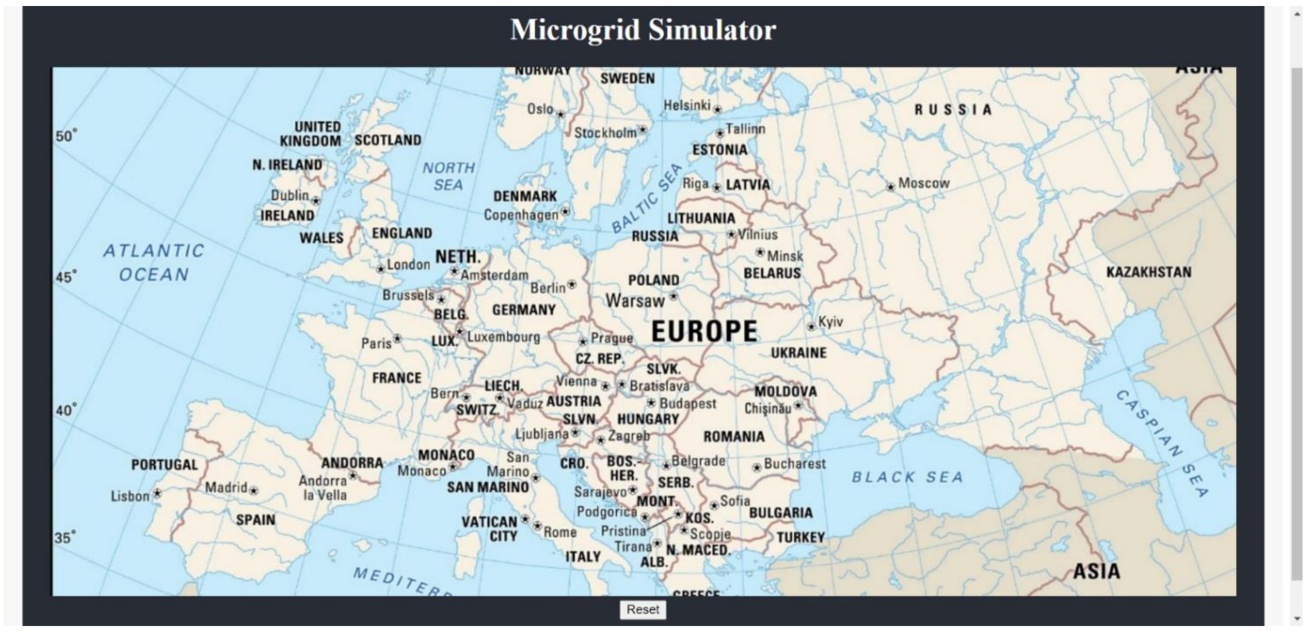


Fig 6.1: Microgrid Simulator Interface

6.4 Workflow

1. Initialization: When users first visit the simulator, they are presented with an interface to upload an Excel file containing the data.
2. Data Input: Users upload an Excel file, and the application parses it to extract necessary data points.
3. Configuration: Users select the date and the countries they want to simulate. The application supports selecting two countries for a comparative analysis.
4. Simulation: Upon confirmation of the selections, the application filters the data for the selected date and countries and calculates the required outputs.
5. Results Display: The results are displayed in a formatted output, showing the generation and load data at half-hour intervals throughout the selected day.
6. Reset and Re-configuration: Users can reset the simulation at any point to change the input file, date, or country selections and rerun the simulation with different parameters.

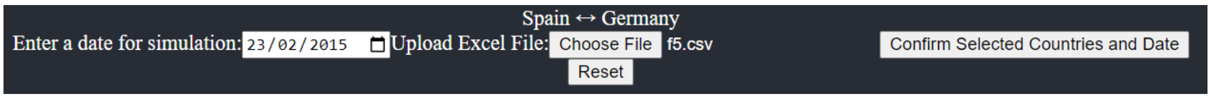

Spain ↔ Germany
Enter a date for simulation: 23/02/2015
Upload Excel File: f5.csv

Fig 6.2: Features of Website

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 Conclusion

Throughout the project, a range of optimization challenges pertinent to global and microgrid systems were addressed, showcasing the capabilities of Python and GBOML in modeling and solving complex problems. The study effectively demonstrated how advanced optimization techniques can lead to significant improvements in energy distribution and management, crucial for achieving sustainability and resilience in power systems worldwide.

The development of a microgrid simulator as part of this project further emphasizes the practical implications of the research, providing valuable tools for energy system analysis and planning. The simulator serves as an innovative platform for testing and visualizing the dynamics of microgrids under various operational scenarios, enhancing understanding and facilitating the design of more efficient and robust energy solutions.

7.2 Future Scope

Future work could expand on the foundational research conducted in this thesis by exploring additional optimization algorithms, enhancing the functionality of the microgrid simulator, and applying the developed models to real-world datasets and scenarios. Also, interconnection of diverse energy grids across the world, aiming to achieve a fully integrated global grid that optimizes renewable energy utilization and enhances global energy security.

Appendix A

Advanced Topics in Optimization for Global Grid Management

A.1. Advanced Techniques in Optimization

The development of optimization algorithms for global grid management involves several sophisticated techniques that are critical for handling the complexities associated with large-scale energy systems. This section delves into some of these advanced techniques, their applications, and benefits.

Stochastic Optimization:

Stochastic optimization deals with optimization problems under uncertainty. For global grid management, this means optimizing grid operations in the presence of uncertain renewable energy outputs, load demands, and system failures. Techniques such as scenario-based planning and stochastic dynamic programming help in making more resilient grid management decisions.

Robust Optimization:

Robust optimization provides solutions that are feasible under a range of possible uncertain parameters. This is particularly important for global grid management where parameters such as demand forecasts, generation capacity, and market prices are not known precisely in advance. Robust optimization helps in designing grid systems that can perform well under a wide range of possible future conditions.

Distributed Optimization:

Distributed optimization algorithms decompose large problems into smaller sub-problems that can be solved by local processors or agents. In the context of a global grid, this approach allows different regions or countries to solve parts of the optimization problem locally while coordinating to achieve a global objective, thus enhancing computational efficiency and scalability.

A.2. Computational Intelligence in Grid Management

The integration of computational intelligence techniques like machine learning and artificial intelligence with traditional optimization approaches has opened new avenues for enhancing grid management.

Machine Learning for Predictive Analytics:

Machine learning models can predict grid loads, renewable energy outputs, and potential system failures with high accuracy. These predictions are crucial for the real-time optimization of grid operations, enabling proactive rather than reactive management.

AI for System Diagnostics and Anomaly Detection:

AI techniques are employed to monitor grid operations continuously, detect anomalies, and diagnose system issues before they lead to failures. This proactive approach helps in maintaining system reliability and preventing costly downtimes

.

A.3. Real-World Applications and Case Studies

European Super Grid:

The European Super Grid is a proposed electrical power transmission network in Europe aimed to connect the energy resources of EU countries. Optimization techniques in such a grid involve managing cross-border energy flows efficiently, minimizing transmission losses, and balancing energy supply and demand across different regions.

China’s Global Energy Interconnection (GEI):

GEI is an ambitious plan by China to develop a worldwide energy network that utilizes renewable energy as a major source. Optimization in this context involves scheduling and dispatching power in a way that maximizes the use of renewables while ensuring global energy supply security.

A.4. Challenges and Future Directions

The path to implementing advanced optimization techniques in global grid management is fraught with challenges including technological barriers, the need for international collaboration and standardization, cybersecurity concerns, and economic and political issues. Future research directions may focus on developing more adaptive, resilient, and secure optimization frameworks that can handle the increasing complexity and interdependence of global energy systems.

This appendix provides a detailed exploration into the sophisticated areas of optimization within the context of global grid management, presenting a comprehensive view that builds upon the concepts discussed in the main body of the report. The information herein is intended to offer a deeper understanding of the advanced strategies and real-world applications that shape modern energy systems on a global scale.

Annexure A:

Supplementary Materials for Optimization Algorithms in Global Grid Development

A1: Extended Dataset Descriptions

Dataset A: Solar Panel Output Data

- **Description**: Hourly generation data from solar panels installed across various geographic locations. Used to model the variability in power generation due to changing weather conditions.
- **Source**: Simulated based on historical weather data.
- **Format**: CSV file with columns for Date, Time, Location, and Power Output.

Dataset B: Wind Turbine Output Data

- **Description**: Minute-by-minute power output data from wind turbines, reflecting the impact of wind speed fluctuations.
- **Source**: Derived from operational data of wind farms.
- **Format**: Excel spreadsheet with Date, Time, Turbine ID, and Power Output.

A2: Technical Documentation

GBOML Framework Usage Guide

- **Content**: Detailed documentation on how to use the Graph-Based Optimization Modeling Language (GBOML) framework for setting up and solving optimization problems specific to energy grids.
- **Format**: PDF with step-by-step instructions, example code snippets, and troubleshooting tips.

References

[1]. Chatzivasileiadis, S., Ernst, D. and Andersson, G., 2013. The global grid. *Renewable Energy*, *57*, pp.372-383.

[2]. Polak, E. ed., 2012. *Optimization: algorithms and consistent approximations* (Vol. 124). Springer Science & Business Media.

[3]. Jordehi, A.R., 2015. Optimisation of electric distribution systems: A review. Renewable and Sustainable Energy Reviews, 51, pp.1088-1100.

[4]. Bynum, M.L., Hackebeil, G.A., Hart, W.E., Laird, C.D., Nicholson, B.L., Siirola, J.D., Watson, J.P. and Woodruff, D.L., 2021. *Pyomo-optimization modeling in python* (Vol. 67). Berlin/Heidelberg, Germany: Springer.

[5]. Miftari, B., Berger, M., Djelassi, H. and Ernst, D., 2022. GBOML: graph-based optimization modeling language. *Journal of Open Source Software*, *7*(72), p.4158

[6]. Olivares, D.E., Mehrizi-Sani, A., Etemadi, A.H., Cañizares, C.A., Iravani, R., Kazerani, M., Hajimiragha, A.H., Gomis-Bellmunt, O., Saeedifard, M., Palma-Behnke, R. and Jiménez-Estévez, G.A., 2014. Trends in microgrid control. *IEEE Transactions on smart grid*, *5*(4), pp.1905-1919.

[7]. Che, L., Zhang, X., Shahidehpour, M., Alabdulwahab, A. and Abusorrah, A., 2015. Optimal interconnection planning of community microgrids with renewable energy sources. *IEEE Transactions on Smart Grid*, *8*(3), pp.1054-1063.

[8]. Khalid, A., Stevenson, A. and Sarwat, A.I., 2021. Overview of technical specifications for grid-connected microgrid battery energy storage systems. *IEEE Access*, *9*, pp.163554-163593.

[9]. Gabderakhmanova, T., Engelhardt, J., Zepter, J.M., Sørensen, T.M., Boesgaard, K., Ipsen, H.H. and Marinelli, M., 2020, September. Demonstrations of DC microgrid and virtual power plant technologies on the Danish Island of Bornholm. In *2020 55th International Universities Power Engineering Conference (UPEC)* (pp. 1-6). IEEE.