UNIVERSIDAD
Panamericana

GREEN GRID SIMULATION

COM 139: SIMULATION & VISUALIZATION [1]

1 INTRODUCTION

The transition to renewable energy is one of the defining engineering challenges of our time. While solar power offers a limitless source of clean energy, it suffers from a critical flaw: **intermittency**. The sun does not always shine when we need power, and it often shines brightest when demand is lowest. This mismatch creates significant instability in our power grid, often visualized as the *"Duck Curve"*, where grid managers struggle to balance the massive influx of solar energy at noon with the sudden drop-off at sunset. To solve this, we cannot simply install more panels; we must design intelligent systems capable of storing, managing, and predicting energy flows within a smart home environment.

The fundamental challenge of residential renewable energy lies in the temporal misalignment between generation and demand, a friction exacerbated by the inherent stochasticity (randomness) of human behavior. While solar production follows a relatively predictable diurnal cycle dictated by physics and weather, household energy consumption is volatile and driven by immediate, often erratic human needs—such as the sudden spike of an electric kettle, the prolonged draw of an EV charger, or a thermostat adjustment. This unpredictability complicates the calculation of energy flow; a static model based on "average hourly usage" fails to capture the instantaneous power surges that define real-world usage. In the energy balance equation ($P_{grid} \approx P_{load} - P_{solar}$), the load is not a smooth curve but a jagged, noisy signal. Consequently, a system designed only for "average" days may unexpectedly drain its battery during a brief consumption spike or unnecessarily curtail (waste) solar energy because the storage logic failed to anticipate a drop in evening demand.

In real-world scenarios, this imbalance is currently managed through a combination of Energy Storage Systems (ESS) and intelligent Demand Response strategies. Traditionally, the utility grid acted as an infinite "battery" via net metering, absorbing excess solar and supplying deficits, but high renewable penetration is now destabilizing that infrastructure (e.g., voltage rise issues). To mitigate this, modern smart homes use local batteries to "time-shift" solar energy from noon to the evening peak. Crucially, to handle the unpredictability of user behavior, advanced Energy Management Systems (EMS) are moving beyond simple "charge/discharge" thresholds. They now employ predictive algorithms that forecast load spikes based on historical usage patterns, automatically deferring power-hungry tasks (like water heating) to align with solar availability, or pre-charging the battery from the grid if a high-consumption event is predicted during a cloudy forecast.

In this 3 part project that will span the whole semester, we will tackle this challenge by building a *"Green Grid"* simulation from the ground up. We will begin by constructing a digital twin of a solar-equipped home, using discrete-event simulation to model the physics of battery charging and household consumption. Once our virtual home is running, we will transition into the role of data analysts, using visualization techniques to diagnose inefficiencies, spot energy waste, and understand the "heartbeat" of our electrical system. This analysis will reveal the limitations of static mathematical models and the need for adaptability in real-world engineering.

Finally, we will evolve our simulation from a static system into an intelligent one. By integrating Machine Learning, we will teach our system to predict solar generation based on real-world data rather than idealized assumptions. This evolution **—from simulation to visualization to prediction—** mirrors

[1] *Engeneering Faculty, Universidad Panamericana, Guadalajara, México*

the real-world workflow of data scientists and systems engineers. By the end of this project, you will not just have learned three distinct technologies; you will have woven them together to solve a complex, interdisciplinary problem that is central to our sustainable future.

2 THE DIGITAL TWIN SIMULATION

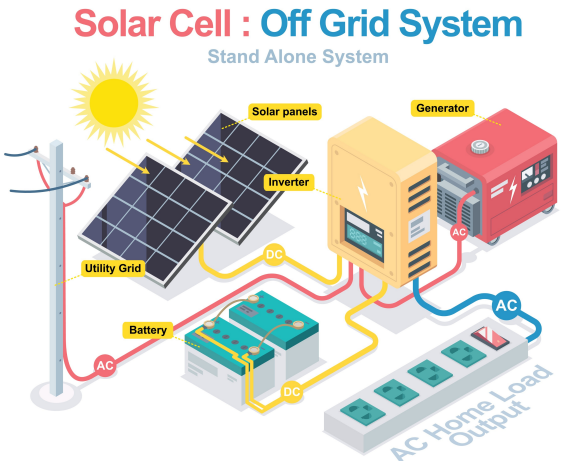
In this first leg of the project, you are tasked on designing and implementing a Digital Twin for a solar-equipped home. This digital twin is a dynamic, virtual replica of the physical home energy system—comprising the solar panels, battery storage, and household appliances. Unlike a static blueprint or a simple spreadsheet model, this digital twin "lives" alongside the system. It integrates the physics of the battery (state of charge, capacity limits), the environmental inputs (solar irradiance, temperature), and the stochastic nature of human behavior (flipping switches, running appliances). By creating this mirror image, we can run "what-if" scenarios safely in a virtual sandbox. For example, we can test if a 13.5 kWh battery is sufficient for a family of four during a cloudy week without actually buying the hardware, or simulate how the system behaves if the solar inverter fails for three days.

In advanced real-world applications, the digital twin serves as the critical bridge between physical reality and predictive intelligence. In our project, the digital twin becomes the testing ground for the machine learning models we develop in a later stage of the project. Instead of deploying an untested algorithm onto a real, expensive electrical grid, we feed our weather predictions into the digital twin to observe the consequences. If the twin predicts a battery blackout based on the forecast, the system can proactively adjust—simulating the real-world capability of a smart home to "pre-charge" from the grid before a storm hits. This transforms the simulation from a passive calculator into an active decision-support tool, mirroring how modern utilities use digital twins to manage the complexity of distributed renewable energy.

Consider the following aspects when designing the digital twin:

- **Battery Capacity & Depth of Discharge:** A battery is not a bottomless bucket. It has a maximum capacity (e.g., 13.5 kWh). Real batteries shouldn't be drained to 0% (it damages them). Should the model enforce a "floor" at 5%?
- **Round-Trip Efficiency:** Charging and discharging a battery incurs losses (typically 80-90% efficient). How will you model this loss in energy during charge/discharge cycles?
- **Inverter Clipping (Max Power Output):** Even if the sun is blazing and the panels could generate 10kW, the home's inverter might be rated for only 7kW. The excess is lost ("clipped"). How will you incorporate this limitation?
- **Temporal Resolution:** The simulation's time step (e.g., 1 minute, 5 minutes) affects accuracy and computational load. What resolution balances fidelity with performance? An Hourly tick (easy math) vs. 15-minute tick (industry standard). 15-minute intervals capture short spikes (like a microwave) that "average hourly" data smooths out and hides.
- **Load Volatility (The Human Factor):** Household energy consumption is erratic. People turn devices on/off unpredictably. How will you introduce randomness to simulate real human behavior? We need a "Base Load" (fridge, router) plus "Random Noise" plus "Scheduled Events" (evening cooking). A static flat line is bad design for a twin.
- **Solar Generation Profile:** Solar panels do not produce a constant output. Their generation varies with time of day, weather conditions, and season. How will you model this variability?
- **Priority Hierarchy:** How will you prioritize different energy demands? For example, should the battery be used to power essential appliances first, or should it be reserved for peak hours? Standard logic is usually: House Load > Charge Battery > Export to Grid.

- **Grid Constraints (Net Metering):** If the home is grid-tied, excess solar can be exported to the grid. However, there may be limits on how much can be sent back. Some grids prohibit export (Zero Export limitation). In that case, if the battery is full and the house is empty, solar generation must turn off. How will you model these constraints?
- **Fault Tolerance:** Real systems can fail (inverter faults, battery degradation). Will your twin simulate potential failures or maintenance events?



The goal of this digital twin is to create a realistic, dynamic model of a solar-equipped home that captures the complexities of energy flow, storage, and consumption. This twin will serve as the foundation for the subsequent visualization and machine learning components of the project, enabling you to analyze performance, identify inefficiencies, and ultimately design smarter energy management strategies.

2.1 Implementation considerations

- For storage capacity, assume a single standard home battery size (13.5 kWh Tesla Powerwall). But you can make this configurable so you can later explore different number and sizes for batteries.
- For solar generation, start with a simple sinusoidal curve peaking at noon to represent a clear day. Later, you can enhance this by incorporating real-world solar irradiance data or weather patterns. Set the peak solar generation capacity to 5 kW.

```
    sun_angle = time_of_day * (math.pi / 12)
    generation = SOLAR_PEAK * math.sin(sun_angle)

```

- For load profiles, you can create a base load representing essential appliances (fridge, lights) and then add random spikes to simulate discretionary usage (TV, oven). You can use probability distributions to model the likelihood of certain appliances being used at different times of the day. Start by assuming a 0.5 kW base load with random spikes up to 3 kW during peak hours (6-9 PM).
- Model the battery state of charge (SoC) as a percentage (0-100%) and update it based on the net energy flow (generation - load) at each time step. Ensure to account for round-trip efficiency losses during charging and discharging. Set the round-trip efficiency to 90%.
- The inverter should have a maximum output limit of 4 kW. If solar generation exceeds this limit, the excess energy is lost (clipped).

- Considering the grid has a max export limit of 20 kW, implement a configurable energy management strategy (**CHARGE_PRIORITY**) where you decide which is the priorities for energy use:
  - **LOAD_PRIORITY:** Power the house load first, charge the battery second, export any excess to the grid last.
  - **CHARGE_PRIORITY:** Charge the battery first, power the house load second, export any excess to the grid last.
  - **PRODUCE_PRIORITY:** Export all energy up to threshold first, charge the battery second, power the house load last.
  - If the battery is full and the house load is low, curtail solar generation (zero export)
- Consider implementing a simple control algorithm that decides when to charge/discharge the battery based on the current state of charge, solar generation, and load demand.
- Implement a **CLOUD_COVERAGE** parameter (0-1) that reduces solar generation proportionally. For example, a cloud coverage of 0.3 would reduce generation by 30%. This value should be affected in a weighted random manner to simulate changing weather conditions throughout the seasons. This are the probability factors based on seasons:
  - Spring: (0.1, 0.3, 0.4, 0.2)
  - Summer:(0.05, 0.15, 0.3, 0.5)
  - Fall: (0.2, 0.4, 0.3, 0.1)
  - Winter: (0.3, 0.4, 0.2, 0.1)
  - Where the values represent: (Clear, Partly Cloudy, Mostly Cloudy, Overcast)
  - A clear day would have a cloud coverage of (0.0 - 0.2), partly cloudy (0.2 - 0.6), mostly cloudy (0.6 - 0.8), and overcast (0.8 - 0.9).
  - You can use these weights to randomly select a cloud coverage level for each day.
- The energy inverter should have a random failure event that occurs on average once every 200 days (0.5%), lasting for a random duration between 4 to 72 hours. During this failure, solar generation is zero regardless of conditions.
- Model time in discrete steps of 60-minute intervals, but allow your system to be configurable for different time steps in order to achieve more granularity in data.
- Your simulation should run for at least a total of 30 simulated days (720 hours) to capture a full month of operation. The selection of the month/season could be configurable to observe seasonal effects on solar generation and load patterns. Yet, you can run shorter simulations (1 day, 7 days) for debugging purposes. On later stages we will need to run longer simulations (3 months, 6 months) up to a year.
- Log key metrics during the simulation, such as battery state of charge, solar generation, load demand, grid import/export, any unmet load, and any other relevant data. This data will be crucial for the visualization and analysis phase. Also consider logging events like inverter failures, battery charge/discharge cycles, and cloud coverage changes.

2.2 Expected output

Once your model is done, your system must be able to answer:

- What is the average state of charge of the battery over the month?
- How often does the battery reach full charge or empty state?

- What is the total energy generated by the solar panels over the month?
- What is the total energy consumed by the household over the month?
- How much energy is imported from/exported to the grid over the month?
- How many times did the inverter fail, and what was the total downtime?
- What is the average cloud coverage during the month?
- What is the peak load demand observed during the month?
- How often was there unmet load (when demand exceeded supply)?
- What is the efficiency of the battery system (considering round-trip losses)?
- How does the energy management strategy (LOAD_PRIORITY vs. CHARGE_PRIORITY vs. PRODUCE_PRIORITY) affect overall system performance?
- If I receive 0.9 cents per kWh exported to the grid, and it costs me 0.75 cents per kWh consumed from the grid, which energy management strategy is most cost-effective?
- What is the impact of different cloud coverage levels on solar generation and battery usage?
- How does the system perform under different seasonal conditions (e.g., summer vs. winter)?
- What is the average duration of inverter failures, and how does this impact overall energy availability?

Also, it should be able to show at least, but not limited to:

- Average production for a month
- Average consumption for a month
- Battery state of charge over time
- Grid import/export over time
- Cloud coverage over time
- Inverter status over time
- Load demand over time
- Unmet load events over time
- Comparison of different energy management strategies

3 THE TASK

Your task is build the Green Grid digital twin simulation as described above. You should use Python with SimPy. You are free to use any additional libraries as needed (e.g., NumPy, Pandas for data handling). Make sure to structure your code well, with clear functions/classes for different components (battery, solar panels, load, grid interaction). This is only the first leg of the whole project, thus, the success of your later stages relies on how well you implement the simulation. Think that in the next phase of the project, you will be integrating this simulation with a web-based dashboard, thus consider producing as much information as you can think in a format that could easily be consumed by a web interface. Remember to document your code and provide instructions on how to run the simulation.

This is a team project, thus, you should work in teams of 2-3 people. Make sure to divide the work evenly among team members and communicate effectively to ensure a cohesive final product. The least you should deliver is a working simulation that meets the requirements outlined above, along with a brief report summarizing your design choices, challenges faced, and potential improvements for future iterations.

The system should be able to be configurable to allow different scenarios to be tested. At minimum, the following parameters should be adjustable:

- Battery capacity (kWh)
- Solar panel capacity (kW)
- Inverter max output (kW)
- Load profile characteristics (base load, peak load, variability)
- Cloud coverage patterns
- Energy management strategy (LOAD_PRIORITY, CHARGE_PRIORITY, PRODUCE_PRIORITY)
- Simulation duration (days)
- Time step (minutes)
- Grid export limit (kW)
- Round-trip efficiency (%)
- Inverter failure frequency and duration
- Season/month for simulation
- Cost of energy imported/exported (cents per kWh)
- Any other parameters you deem relevant for testing different scenarios.

3.1 What to deliver

- Link to the repository where your code is placed
- A README file with clear instructions on how to set up and run the simulation, including any dependencies or libraries required.
- The complete source code for the digital twin simulation, well-documented and organized.
- A brief report (2-4 pages) summarizing your design choices, challenges faced, and potential improvements for future iterations. The report should also include answers to the expected output questions listed above.
- A development log with the day-to-day progress of the project with clear indication of the task that each of the team members performed.
- A personal reflection by each team member on the work done for the project, this should include ideas from the inception of the simulation to decisions taken to account for the future integration.

3.2 How is it going to be graded

- 20% - Report
- 50% - Code running
- 15% - Parametrized code
- 5% - Expected output produced
- 5% - Personal reflections
- 5% - Documentation and coding standards

REFERENCES

[1] Brown, Kevin (Retrieved 10:24, May 09, 2024), *Simulating and Visualizing Real-Life Events in Python with SimPy*, towardsdatascience.com,
https://towardsdatascience.com/simulating-real-life-events-in-python-with-simpy-619ffcdbf81f