A Flexible Python Program to Assess Optimal Microgrid Resiliency, Autonomy and DER Integration Cost

Tianyi Ma
University of Colorado at Boulder
alex.ma@colorado.edu

Abstract

With various factors like the possibility of cyberattacks and more frequent extreme weather events, the grid is becoming more fragile and prone to high stress and blackouts. To reduce electricity costs and gain better power supply autonomy and resiliency, microgrids have become a desirable compliment to the central grid. Several works have showed clear financial and environmental advantages of microgrids using case studies. However, it remains difficult for microgrid designers to plan and adapt to varying location, renewable potential and microgrid DER (Distributed Energy Resources) settings in the real world. In this work, I created a fast Python program to optimize for DER settings, costs, autonomy and resiliency of any potential microgrid. First, I define a flexible input setup describing the renewable potential, renewable energy sources parameters and microgrid load profile. The program takes the input and as a first step returns an economically balanced DER setup with optimal autonomy. As a second step, the program combines the optimized DER setup and input to produce a 7-day profile of the microgrid resiliency. Ample visualizations are provided to make sense of the model decision and to assist in parameter adjusting and ultimate decision making. The results are demonstrated using the campus of Hong Kong University of Science and Technology.

1 Introduction

As the central grid strategy becomes more and more stressed because of extreme weather events, renewable energy penetration and cyberattacks, new grid strategies are urgently needed. One of the most promising and exciting directions is microgrid. Microgrid, according to NREL, is "a group of interconnected loads and distributed energy resources that acts as a single controllable entity with respect to the grid"(NREL 2024). The central grid's structure is fairly easy to understand. Microgrid instead has many different factors that can influence their costs and effectiveness, for example, which types and how much renewable energy to include, whether to include an energy storage system and what type and size and much more. Furthermore, these are easily influenced by local weather, economy, budget, priority and load demand.

It's well-proven by previous studies that renewable energy integration and battery storage helps the microgrid save money and become more green (Marqusee, Becker, and Ericson 2021). The resilience of microgrid is also an active area of study(Rehman et al. 2024). Given the plethora of mathematical optimization problem formations, solution algorithm and open-source solver implementations, a general purpose optimization program to help with real world microgrid design is yet to be realized. This study takes the first step by creating a flexible Python program that takes in the location, weather data, renewable energy unit parameters and load profile. In a two step process, the program returns the optimal grid DER setup with costs balanced by grid autonomy, and then a resiliency analysis over a period of 8 days.

2 Methods

This section provides the theoretical formulation of the mathematical optimization problem, including the definition, objective functions, constraints and the algorithm to assess resiliency. We start by defining the inputs.

2.1 Inputs

The formulation include three different DER components, namely solar, wind and battery storage. The following sections defines each of the components in sequence. A table for a complete list of definitions are detailed in the next page.

2.1.1 Solar

The final solar input is a 2-week long 10-minute interval time series of solar generation profile in kW10min. Here, the span and frequency of the time series is arbitrary, but should remain consistent throughout the input set so that the model is mathematically correct.

Aggregating all data, the 2-week solar interval series is calculated using the following equation at each time step t for $t \in \{1, ..., T\}$:

$$P^t_{solar} = \text{h} \times \text{w} \times DSR^t \times \text{efficiency} \times \text{performance ratio} \quad (1)$$

This program assumes the needed input data is well-prepared and provided at the time of calculation. This is also assumed for all other input types.

2.1.2 Wind

Like the solar generation profile, the final wind output should be a fixed-length interval time series in kW(interval). Here, we pick a length of 2 weeks and intervals of 10 minutes, the same as the solar profile.

The specific units can be different from the defined units. Here the paper choose the above units because they are either easy to understand (ft) or easy to find (m/sec for wind speed data). This work also strictly calculates the energy in kW(interval) instead of W(interval) or MW(interval), though will sometimes present the results in a different magnitude.

Aggregating all data, the 2-week wind interval series is calculated using the following equation at each time step t for $t \in \{1, ..., T\}$:

$$P^t_{wind} = 1/2 \times \rho_{air} \times A \times (v^t)^3 \quad (2)$$

Where A is the sweeping area associated with the unit blade length.

2.1.3 Battery

The microgrid DER resources may have battery storage. The battery capacity is a variable to be optimized in step 1 of the program, but is given as a constant in step 2 of the program. The battery power state and battery charge/discharge activities are optimization variables, and the cost is a constant to be specified by the user.

2.1.4 Load Demand And Power Balance

The load demand data P_{load} is a time series with a certain span and interval. Staying consistent with the above specifications, the length of the series is 2 weeks with 10-minute intervals. The decision and optimization variables for maintaining power balance are then listed after the definition of P_{load} in Table 1.

2.2 Problem Formulation

With all inputs defined and assumptions clarified, this subsection will proceed to describe the theoretical formulation of the problem, starting by defining the key metrics to optimize. It will then define the objective functions for autonomy and resiliency optimizations, then the problem constraints. Lastly, the section briefly discuss the programming package and solver used in this program.

2.2.1 Key Metric Definitions

The key metrics this program considers are costs, autonomy and resiliency. The cost is subject to optimization only in step 1 of the program, including the cost of solar, wind, battery and power from grid. Optionally, the user can choose to include cost of energy curtailed into consideration as well and compare the costs and trade-offs with not including cost of energy curtailed.

Definition "Autonomy": the minimal P_{grid} needed over a time period for a microgrid.

Definition "Resilience": the maximum duration a microgrid can maintain critical loads in island mode.

With these definitions of key metrics and inputs, the following subsections will formulate the optimization problems.

2.2.2 Costs and Autonomy Optimization

In step 1, the program optimizes for number of renewable energy units, size of battery storage capacity and microgrid reliance on grid power with costs as the objective function. Quantifying this optimization goal results in the following objective function:

$$a_{solar} = c_{solar\_panel} \times a_{solar\_panel} \quad (3)$$

$$a_{wind} = c_{wind\_turbine} \times a_{wind\_turbine} \quad (4)$$

$$a_{battery\_total} = a_{battery} \times E_{max} \quad (5)$$

Symbol	Definition	Unit
P_{solar}	2 week solar generation profile	$kW10min$
c_{solar_panel}	The number of units of the desired solar panel	A whole number variable
DSR	The direct solar radiation profile	$kW10min/m^2$
h	Height of a solar panel unit	m
w	Width of a solar panel unit	m
efficiency	Efficiency of a solar panel unit	$\%$
a_{solar_panel}	Cost of a solar panel unit	$
performance ratio	System performance ratio, default to 0.85	$\%$
P_{wind}	2 week wind generation profile	$kW10min$
v_{wind_speed}	2 week wind speed profile	m/sec
l_{blade}	The blade length of the wind turbine unit	ft
$a_{wind\_turbine}$	The cost of the wind turbine unit	$
ρ_{air}	The air density at the desired location	kg/m^3
$c_{wind\_turbine}$	The number of the desired wind turbine	A whole number variable
$P_{battery}$	The battery power for a 2-week span	$kW10min$
$P_{battery\_charge}$	The battery charge/discharge for a 2-week span	$kW10min$
E_{max}	The battery maximum capacity	$kW10min$
$a_{battery}$	The cost $(1kWh)$ of battery capacity	$
P_{load}	2 week load demand profile	$kW10min$
P_{grid}	Power from grid for a 2-week span	$kW10min$
$P_{energy\_curtailed}$	Power curtailed for a 2-week span	$kW10min$
P_{not_met}	Power not metfor a 2-week span	$kW10min$
$p_{critical\_load}$	Critical load percentage	$kW10min$
$a_{P_{not\_met}}$	Cost per unit power, a penalizing term	$
$a_{P_{grid}}$	Cost per unit power from grid	$\$/kW10min$
$a_{energy\_curtailed}$	Cost per unit power curtailed	$\$/kW10min$

Table 1: Variables and Constants Definitions

Summing the costs of solar, wind and battery storage, we get the estimated total cost of DER resources.

$$a_{total} = a_{solar} + a_{wind} + a_{battery\_total} \tag{6}$$

The total microgrid cost over a period of time in 10-minute intervals equals to the total cost of DER resources plus the cost of the minimal sum of power from the central grid:

$$\min_{P_{grid}} (a_{total} + a_{P_{grid}} \sum_{t=1}^{T} P^t_{grid}/6) \tag{7}$$

The problem also optimize the following decision variables: c_{solar_panel}, $c_{wind\_turbine}$, E_{max}, $P_{battery}$, $P_{battery\_charge}$, $P_{energy\_curtailed}$. These are listed here instead of under the minimization symbol to keep the equation clean. Notice the summation over P^t_{grid} is divided by 6 because the series has an interval of 10 minutes, and $10min/60min = 1/6$.

The following objective function is used if the user wants to take curtailed energy costs into consideration:

Cost of energy curtailed equals the unit cost multiplied by the sum of energy curtailed over 10-minute intervals divided by interval scalar 6.

$$a_{energy\_curtailed} = a_{P_{energy\_curtailed}} \sum_{t=1}^{T} P^t_{energy\_curtailed}/6 \tag{8}$$

The total cost of total energy used from grid is quantified same as before:

$$a_{grid} = a_{P_{grid}} \sum_{t=1}^{T} P^t_{grid}/6 \tag{9}$$

Summing the costs of DER resources, energy from grid and energy curtailed:

$$\min_{P_{grid}} (a_{total} + a_{grid} + a_{energy\_curtailed}) \quad (10)$$

This concludes our definition of objective function of minimizing DER costs and autonomy. The next subsection details the objective function and algorithm to assess microgrid resiliency. Constraints for the two problems are discussed together in the section to stay concise.

2.2.3 Resiliency Assessments

In step 2 of the program, the optimal DER setup is already calculated from step 1. At this step, the program will solve the optimization problem for resiliency iteratively where the island mode starting time increments in hours of 2 for the first week. At each starting time, the program calculates power not met by the microgrid power supply over the next 7 days. As a result, the last iteration will starts the end of the first week and looking ahead to the second week to obtain the amount of hours the microgrid can sustain critical load stepping into the second week. The resiliency at each starting hour is then plotted as a time series, and the duration histogram is plotted to visualize the resiliency distribution.

Empirically, battery storage is tricky to optimize given a microgrid with a rather large load demand. It very possible that the optimal battery storage is 0 from step 1. Thus, step 2 of the program will allow the user to input their desired battery storage capacity, which could be estimated based on budget, or an educated guess based on heuristics. Moreover, the user chosen battery storage size can also be plugged into step 1 again to assess the financial impact of enforcing a minimum battery storage capacity in step 1.

The objective function is simply the sum of power not met divided by the interval scaler weighted by the penalization for having power not met.

$$\min_{P_{not\_met}} (a_{P_{not\_met}} \sum_{t=1}^{T} P_{not\_met}/6) \quad (11)$$

Formally, the formal algorithm is detailed on the top right in Algorithm 1.

2.2.4 Constraints

As the last piece of the puzzle, this section describes the constraints for each of the two problems. First, DER costs and autonomy constraints is discussed.

Algorithm 1 Resiliency Calculation over 7 Days

```
1:  **Input:**
2:  **Horizon:** $H = 24 \times 7/2 = 84$
3:  **Start Times:** 1/1 00:00:00 + 02:00:00 × $i$ where $i \in \{1, 2, ..., H\}$
4:  **Number of Intervals:** $T = 24\times6\times7 = 1006$
5:  **Timesteps:** $t_{start}$ + 00:10:00 × $t$ where $t \in \{1, 2, ..., T\}$
6:  **Output:** The resiliency of the microgrid if central power is off at $t_{start}$ for Timesteps.
7:  $t_{durations}$
8:  **for** $t_{start}$ **do**
9:      Optimize for minimal power not met:
10:     $p_{load\_state} = 1 - P^t_{not\_met}/P^t_{load}$
11:     **if** at $t$, $p_{load\_state} \leq p_{critical\_load}$ **then**
12:         Add $t/6$ to $t_{durations}$
13:     **else** Add $T/6$ to $t_{durations}$
14:     **end if**
15: **end for**
16: **Return** $t_{durations}$
```

DER Costs and Autonomy First, we tackle the battery constraints, which is described below:

$$P^t_{battery} = P^{t-1}_{battery} + P^{t-1}_{battery\_charge} \quad (12)$$

This equation describes the battery charging/discharging activities. It specifies that the battery power state at time step t equals to the battery power state from the last time step t-1 plus the energy charged/discharged from the battery at time step t-1.

$$P^0_{battery} = 0, P^t_{battery} \leq E_{max}, P^t_{battery} \geq 0 \quad (13)$$

These constraints make sure that the battery obeys physics laws. The battery power should always be larger than 0 but below the battery capacity. Moreover, the battery storage starts empty.

Lastly, the maximum battery capacity has to be larger than 0.

$$E_{max} \geq 0 \quad (14)$$

Then the power balance equations and constraints are stated below:

$$P_{solar\_gen} = c_{solar\_panel} \cdot P_{solar} \quad (15)$$

$$P_{wind\_gen} = c_{wind\_turbine} \cdot P_{wind} \quad (16)$$

If these are constraints for step 2 of the program, then the number of renewable energy units are constants, or results from step 1 of the program.

$$P_{renewable} = P_{solar\_gen} + P_{wind\_gen} \tag{17}$$

The most important equation here is the power balance equation constraint:

$$P_{load} + P_{battery\_charge} + P_{energy\_curtailed} = P_{renewable} + P_{grid} \tag{18}$$

The left hand side is the power demand including load, battery charge/discharge and energy curtailed. Here, the battery charge can be negative which means the battery is discharging energy. On the right hand side is the power generation, or supply, which include the renewable generation and power from grid. As a reminder, here is the objective function of this problem:

$$\min_{P_{grid}}(a_{total} + a_{P_{grid}} \sum_{t=1}^{T} P_{grid}^{t}/6) \tag{19}$$

A few additional physical constraints for variables in this power balance equation are listed below:

For energy curtailments, we need it to be larger than 0 but lower than the renewable energy generation.

$$P_{energy\_curtailed}^{t} \geq 0, P_{energy\_curtailed}^{t} \leq P_{renewable}^{t} \tag{20}$$

For renewable energy generation in step 1, we need both to be above 0 at all times.

$$P_{solar\_gen}^{t} \geq 0, P_{wind\_gen}^{t} \geq 0 \tag{21}$$

Lastly, because energy cannot go back to the grid:

$$P_{grid}^{t} \geq 0 \tag{22}$$

Resiliency The constraints for resiliency are very similar to constraints for autonomy. Most importantly, the power balance equation becomes:

$$P_{load} + P_{battery\_charge} + P_{not\_met} + P_{energy\_curtailed} = P_{renewable} \tag{23}$$

As a reminder, the objective function for optimizing for resiliency is:

$$\min_{P_{not\_met}} (a_{P_{not\_met}} \sum_{t=1}^{T} P_{not\_met}/6) \tag{24}$$

Here, both the renewable generation profile and maximum battery capacity are already given and therefore fixed. Furthermore, power not met has the following constraints:

$$P_{not\_met} \geq 0, P_{not\_met} \leq P_{load} \tag{25}$$

2.2.5 Solver Description

Both of the optimization problems are linear convex problems that can be easily and quickly solved by solvers. This Python program used the package called 'cvxpy'(*cvxpy.org* n.d.) which will choose the best solver for the given data and problem definition from their built-in list of solvers. By setting 'verbose=True' the solver will print out the size of the problem, including the number of parameters and constraints. For a complete documentation of their solvers, refer to their online documentation(*Solver Features* n.d.).

This concludes the theoretical formation of the target optimization problems. The next section will apply the program to a specific example to showcase the results. It will first provide context for the chosen example, an university campus in HK, the input datasets and the output results. Finally, an analysis of the results is included.

3 Results

3.1 Example: HKUST

HKUST (Hong Kong University of Science and Technology) is located at Clear Water Bay in Hong Kong. It's nearby the sea, with a student population of 14.2k and a strong wind profile. The campus was chosen because university campuses are ideal places to set up a microgrid with a centralized administration and the resulted strong planning and coordination abilities. Moreover, Hong Kong, surrounded by water, is vulnerable to extreme weather events, making microgrid a realistically valuable alternative to the central grid. Lastly, HKUST has published peer-reviewed interval electricity meter data(Li et al. 2024), while the weather stations in Hong Kong have recent 1-minute and 10-minute data for solar radiation and wind speed(*Regional weather in Hong Kong – the latest 10-minute mean wind direction and wind speed and maximum gust — DATA.GOV.HK* n.d.; *Regional weather in Hong Kong – the latest 1-minute global solar radiation and direct solar radiation and diffuse radiation — DATA.GOV.HK* n.d.) that go back to decades. Especially the

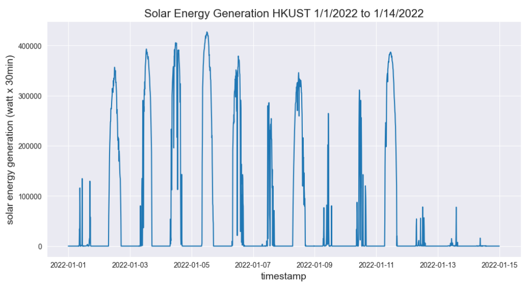


Figure 1: Solar Generation Profile

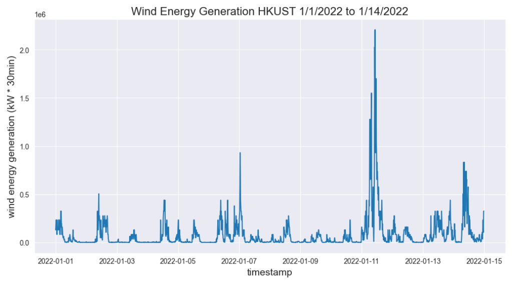


Figure 2: Wind Generation Profile

weather data is available to download for a specific time period. All of these properties make the HKUST campus an ideal example to study.

3.2 Input Datasets

3.2.1 Renewable Resources and Load Demand

The wind speed and solar radiation datasets are gathered by the government's automated weather stations located across HK. They're provided in 1-minute intervals for the solar radiation data and 10-minute intervals for the wind speed data with the associated station where the data was gathered. All data are hosted on data.gov.hk, the official government sites for providing their dataset.

This study examined all weater station locations and picked one station up a few miles north from the HKUST campus and is nearby the water. Because its proximity to the campus and the fact that it's also nearby water, its wind speed data should be close to the true speed at the campus.

Each interval has a corresponding csv file, and the program pre-process the data to concatenate the files and compile the radiation data to use 10-minute intervals instead of 1-minute intervals. The program then pick the data from 2022/1/1 00:00:00 to 2022/1/14 23:50:00. The reason for these ranges is because the load demand dataset provided by HKUST has a full profile at this time period, while missing some data points in other period.

3.2.2 Parameters

Here we pick example wind turbine and solar panel products existing in the real world market and use their specification as input parameters to calculate the renewable energy generation profiles. The prices are adjusted to Hong Kong dollars and their economy by estimating a conversion ratio based on their difference in GDP per capita. As of the time of writing, HK GDP per capita is around 50k USD, while the US GDP per capita is around 80k, so the conversion ratio is set to $50USD/80USD = 5/8$.

The solar panel chosen is sourced from the website ShopSolar(*Rich Solar MEGA 550 — 550 Watt Bifacial Solar Panel - ShopSolar.com* n.d.). The unit panel size is 89.7 x 44.6 x 1.2 inch. The panel efficiency is 21.3%. For a bundle if 8, the price is 3,599USD.

It's hard to find a community level wind turbine online. Here, we use an estimation from homeguide.com(*How Much Does a Wind Turbine Cost? (2024)* n.d.), where it costs 20k - 80k USD to purchase a wind turbine unit with 5 - 15kWh power range. Setting our wind turbine blade length to 32.5ft in diameter, it falls within range of the estimated power and we take the assumption that it costs around 66.6k USD, based on a real world example on eBay(*300KW Windkraftanlage Komplettanlage Kits 220V 380V 600V Windgenerator Windrad1P — eBay* n.d.).

Lastly, for the cost of unit battery storage we assume a typical price of 800USD = 6220 HKD for every kWh of capacity.

3.2.3 Renewable Generation Profile

After getting all datasets needed, the wind and solar energy generation profile per equipment unit is generated using the equations detailed in Section 2.1.1 and 2.1.2, respectively. To visualize what they look like, two time series graphs are provided above. Notice in Figure 1 the solar generation is heavily dependent on day light and fluctuates significantly. The wind profile also has a tall spike which reflects the versatility of the wind speeds at a location near water. These are all important aspects that greatly influence to the optimization results of the program.

Optimal Grid Power and DER setting (1.38 million USD)
Average Power from grid: 413785.41 kWh (= 413.79 MWhr) Average Renewable Energy Generation: 840600.46 kWh (= 840.6 MWhr) Number of small solar panels: 119, costs: 34807.5 USD (= 34.81k USD) Number of wind turbine with 32.5ft blade length: 5, costs: 211250.0 (= 211.25k USD) Battery storage size: 0
45.61 % of total energy used is generated using solar and wind (curtailed energy is reasonably subtracted)
Cost Analysis
Two week P_{grid} costs: 1129634.17 USD (= 1.13 million USD) Two week P_{grid} without renewable costs: 2.08 million USD Reduced cost by 45.61 % looking only at the costs of energy production Overall the DER integration reduced cost by 33.76 %
Energy Curtailments
Average Power curtailed: 123413.13 kWh (= 123.41 MWhr) Two week energy curtailment costs, assuming $a_{curtailment} = \$1.87/kWh$, is 518335.13 USD (= 0.52 million USD)

Table 2: Results: DER Costs and Autonomy (Excluding Energy Curtailments)

Optimal Grid Power and DER setting (1.62 million USD)
Average Power from grid: 506807.69 kWh (= 506.81 MWhr) Average Renewable Energy Generation: 376436.04 kWh (= 376.44 MWhr) Number of small solar panels: 77, costs: 22522.5 USD (= 22.52k USD) Number of wind turbine with 32.5ft blade length: 2.0, costs: 84500.0 USD (= 84.5 k USD) Battery storage size: 0
33.38 % of total energy used is generated using solar and wind (curtailed energy is reasonably subtracted)
Cost Analysis
Two week P_{grid} costs: 1383585.0 USD (= 1.38 million USD) Two week P_{grid} without renewable costs: 2.08 million USD The microgrid reduced cost by 33.38 % Overall reduced cost by 22.03 %
Energy Curtailments
Average Power curtailed: 30627.59 kWh (= 30.63 MWhr) Two week $P_{energy\_curtailed}$ costs: 128635.89 USD (= 0.13 million USD)

Table 3: Results: DER Costs and Autonomy (Including Energy Curtailments)

3.3 Optimization Results

This section will provide optimization results for the above example setup at HKUST. For a general grasp of the scale of the problem, we have for each unit of solar panels an average daily generation of 1367.2kWh. For each unit of wind turbines the average daily generation is 135580kWh. The campus has a 760733.36kWh average daily load demand.

3.3.1 Costs and Autonomy

First we consider the base case of optimizing for DER resources costs and microgrid autonomy without including costs of curtailed energy. The problem includes 8067 variables and 24193 constraints. The optimal cost is 1.38 million USD including solar panel, wind turbine and battery storage installation and purchase costs with a daily load equivalent to 760.73 MGhr. A detailed breakdown is included in Table 2 on the next page.

Notice this is one of the possible situations where the battery costs are too much to accommodate to the local weather profile. Specifically, in the middle of second week the wind is particular strong and makes it difficult for small battery sizes to be useful. To provide some intuition, the battery size would have to exceed a certain amount, in this case possibly at least around a bit less than a hundred MWh to have an impact. This is too much investment for a price of \$800/kWh.

In the case including the cost of energy curtailed we include it as one of the optimization variable.

In this case the optimal cost is 1.62 million USD including DER setup costs. The detailed result breakdown is provided in Table 3 on the next page.

Analysis The first alternative includes more renewable energy resources, and is less reliant on the central grid. It's also more green than the second option and saves more money. For this particular example, it would be reasonable to choose the first option, if microgrid autonomy and environmental impact are prioritized over budget and efficiency. Notice that although the first option appears to cost less than the second option, it does not consider cost of curtailment, while the second option does. If we add the associated cost of curtailment to the first option's result, the second option is actually cheaper than the first option by 0.28 million USD. We can also compare energy efficiency trade-offs between the two cases. The costs are slightly higher in the second case, if we consider the original costs estimates, with around 75% less average energy curtailments. The energy efficiency equals to 85.32% for the base case where in the second case the energy efficiency equals to 92.9%, an increase of 7.6%. Each solution has its own advantages and disadvantages. For example, the extra energy could be traded in the future if given a such platform. Thus, the decision of which DER setting to adopt considering energy curtailments is up to the microgrid designer.

3.3.2 Resilience

The suggested battery size is 0 from step 1, but having a battery improves microgrid resiliency and the battery size is a manual parameter to be set for reasons discussed before. In this particular example we manually set the battery capacity to 50MWh which is appropriate with respect the size of the load. We then calculate the resiliency of the microgrid assuming it goes off grid starting in 2-hour incrementing timesteps. The resulting time series is visualized in Figure 3.

Analysis The graph shows that if the microgrid starts going on island mode during late nights, it is not able to stay operational for long. If the central grid goes offline during the day, it usually has a much easier time staying operational and can sustain for a relatively long period of time. One possible reason is that the solar energy generation is ample during the day but not at night. This relationship between renewable generation and grid resilience makes intuitive sense and is further evident by looking at the resilience spike at the trials starting from 2022/1/6 to 2022/1/7. Because the program looks 7 days ahead, the spike could be explained by the spike in wind production in the middle of the second week, referring to Figure 2. Overall, the resilience analysis showed that the microgrid resilience heavily relies upon its renewable energy resources.

The average resiliency considering all trial starting times is around 3.82 hours. However, if we exclude the trials that start at night, the average resiliency has a sharp increase to 6.93 hours, which is impressive given the scale of the load demand. Furthermore, the mean of island mode durations is satisfactory, from 3 hours to 7 hours, with a maximum of 22.67 hours surviving island mode. This has proved the capability and flexibility of the program to produce a desired outcome given a set of weather, location, renewable energy unit size and load profile parameters, with the goal of balancing costs, autonomy and resiliency and the tracking of energy efficiency.

4 Discussion

The usefulness of the program in this work to customize and optimize the design and planning of a potential microgrid is clearly evident. Furthermore, the program also produces detailed data-driven costs and effectiveness analysis regarding the microgrid autonomy and resiliency, where the resiliency is robustly tested using trials with 2-hour incrementing starting times over the course of a week with a 7-day horizon.

As a first step towards a general purpose program to assist microgrid design , there're several future directions that the author believes will be worthwhile pursuits. One obvious next step is to make this an API programming interface in the form of a Python open-source library that can be widely distributed. This will allow the designer to try out different renewable resources unit settings and tuning the battery storage parameter. It will allow the user to compare different results from historic runs and help with the decision making process. Moreover, this will also make it easier to develop a package in the long term with more functionalities. For example, one further development could be to connect it to a weather database so that the user doesn't have to provide the data. Lastly, more microgrid components could be defined mathematically and include in the coding

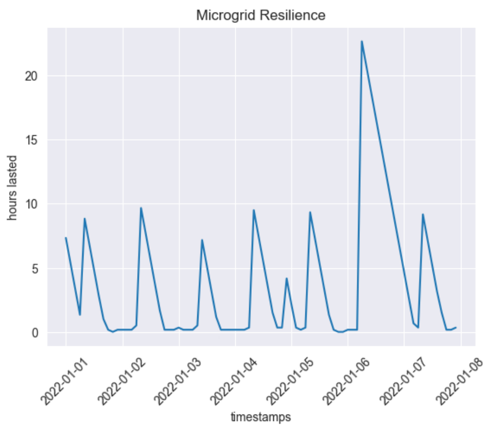


Figure 3: Microgrid Island Mode Duration Over The First Week

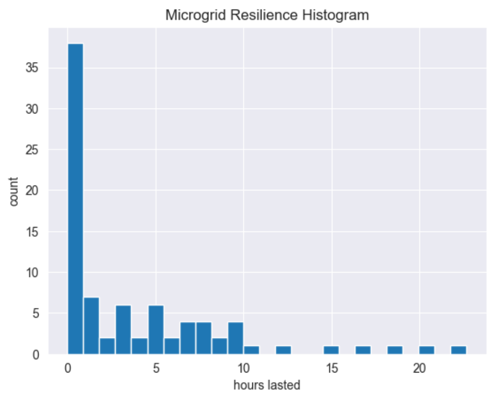


Figure 4: Microgrid Island Mode Duration Counts

package, for example, an EV network setup, or a different type of storage system or renewable energy source. These are all exciting future directions that the author encourages to explore.

References

300KW Windkraftanlage Komplettanlage Kits 220V 380V 600V Windgenerator Windrad1P — eBay (n.d.). https://www.ebay.com/itm/375523517271. [Accessed 13-12-2024].

cvxpy.org (n.d.). https://www.cvxpy.org/index.html. [Accessed 13-12-2024].

How Much Does a Wind Turbine Cost? (2024) (n.d.). https://homeguide.com/costs/wind-turbine-cost. [Accessed 13-12-2024].

Li, Mingchen et al. (2024). "A multi-year campus-level smart meter database". In: *Scientific Data* 11, p. 1284. ISSN: 2052-4463. DOI: https://doi.org/10.1038/s41597-024-04106-1.

Marqusee, Jeffrey, William Becker, and Sean Ericson (2021). "Resilience and economics of microgrids with PV, battery storage, and networked diesel generators". In: *Advances in Applied Energy* 3, p. 100049. ISSN: 2666-7924. DOI: https://doi.org/10.1016/j.adapen.2021.100049. URL: https://www.sciencedirect.com/science/article/pii/S266679242100041X.

NREL (2024). *Microgrid.* Accessed: 2024-12-12. URL: https://www.nrel.gov/grid/microgrids.html#:~:text=A%20microgrid%20is%20a%20group,grid%2Dconnected%20or%20island%20mode..

Regional weather in Hong Kong – the latest 1-minute global solar radiation and direct solar radiation and diffuse radiation — DATA.GOV.HK (n.d.). https://data.gov.hk/en-data/dataset/hk-hko-rss-latest-one-minute-solar-radiation-info. [Accessed 13-12-2024].

Regional weather in Hong Kong – the latest 10-minute mean wind direction and wind speed and maximum gust — DATA.GOV.HK (n.d.). https://data.gov.hk/en-data/dataset/hk-hko-rss-latest-ten-minute-wind-info. [Accessed 13-12-2024].

Rehman, Talha et al. (2024). "Optimal sizing of multi-energy microgrid with electric vehicle integration: Considering carbon emission and resilience load". In: *Energy Reports* 11, pp. 4192–4212. ISSN: 2352-4847. DOI: https://doi.org/10.1016/j.egyr.2024.04.001. URL: https://www.sciencedirect.com/science/article/pii/S2352484724002063.

Rich Solar MEGA 550 — 550 Watt Bifacial Solar Panel - ShopSolar.com (n.d.). https://shopsolarkits.com/products/rich-solar-mega-550-watt. [Accessed 13-12-2024].

Solver Features (n.d.). https://www.cvxpy.org/tutorial/solvers/index.html. [Accessed 13-12-2024].

5 Appendix: Codebook

Import Libraries and Define Utility Function

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import cvxpy as cp
import os
import datetime as dt
from datetime import datetime, timedelta
from dateutil import parser
import math
import itertools
```

In [2]:
```
def create_10min_interval_data(start, end):
    timestamps = []
    current_time = start
    while current_time <= end:
        timestamps.append(current_time)
        current_time += timedelta(minutes=10)
    return timestamps
```

Data Preparation

Load Profile

In [3]:
```
load_df = pd.read_excel("data/load_raw.xlsx")
load_df.head()
```

Out[3]:

	time	number
0	2022-01-01 00:01:28	33050.3
1	2022-01-01 00:16:28	33051.2
2	2022-01-01 00:31:28	33052.2
3	2022-01-01 00:46:28	33053.0
4	2022-01-01 01:01:28	33053.8

Aggregate the series into 10min interval data

The original series is roughly in 15-minute intervals. The following code add up groups of 4 interval data and divide it by 6 to get an average profile over an hour in 10-minute intervals.

In [4]:
```
total_time_steps_2weeks = 2 * 7 * 24 * 4 + 1

load_series = []
for i in range(2 * 7 * 24):
    kWh = sum(load_df.loc[i*4:(i+1)*4].number.values)
    load_series.extend([kWh/6] * 6)

start, end = datetime(2022, 1, 1, 0, 0, 0), datetime(2022, 1, 14, 23, 50, 0)
timestamps = create_10min_interval_data(start, end)

load_cleaned = pd.DataFrame({"timestamps":timestamps, "P_load (kW * 10min)":load_series})
load_cleaned.head()
```

Out[4]:

	timestamps	P_load (kW * 10min)
0	2022-01-01 00:00:00	27543.416667
1	2022-01-01 00:10:00	27543.416667
2	2022-01-01 00:20:00	27543.416667
3	2022-01-01 00:30:00	27543.416667
4	2022-01-01 00:40:00	27543.416667

Save the data and calculate daily average load

In [5]:
```
load_cleaned.to_csv("data/load_cleaned.csv", index=False)
```

In [6]:
```
load_cleaned["P_load (kW * 10min)"].sum()/6/14
```

Out[6]:
```
760733.3595238095
```

Generate Unit Wind Profile

Formula:

- Input: Wind speed profile, blade length
- Equation: P_avail (kWh) = 1/2 * density of air (kg/m^3) * sweaping area (m^3) * v^3 (m/sec)

Wind speed profile from HK data source: https://data.gov.hk/en-data/dataset/hk-hko-rss-latest-ten-minute-wind-info

In [7]:
```
directory_path = "data/HK_wind_speed_profile"

files = sorted(os.listdir(directory_path))[1:]

for file in files[:3]:
    print(file)
```

```
20220101-0011-latest_10min_wind.csv
20220101-0019-latest_10min_wind.csv
20220101-0029-latest_10min_wind.csv
```

Compile files into one single series

In [8]:
```
total_time_steps_10min = 6 * 2 * 7 * 24

valid_files = files[:total_time_steps_10min]

curr_time = datetime(2022, 1, 1, 0, 0)

timestamps = []
wind_speeds = [] # km/hr

for file in valid_files:
    wind_speed_10min_df = pd.read_csv(f'{directory_path}/{file}')
    wind_speed_10min_HKUST = wind_speed_10min_df[wind_speed_10min_df['Automatic Weather Station'] == 'Sai Kung'
    wind_speed_flt = wind_speed_10min_HKUST['10-Minute Mean Speed(km/hour)'].values[0]
    if math.isnan(wind_speed_flt):
        wind_speed = None # fill in None value if there's no record at the time step
    else: wind_speed = int(wind_speed_flt)
    timestamps.append(curr_time)
    wind_speeds.append(wind_speed)
    curr_time += timedelta(minutes = 10)

wind_speed_df = pd.DataFrame({"timestamp":timestamps, "wind_speed":wind_speeds})
```

Dealing with Null data points through Interpolation

In [9]:
```
wind_speed_df.wind_speed.isnull().sum()
```

Out[9]:
```
5
```

In [10]:
```
none_idx = wind_speed_df.wind_speed[wind_speed_df.wind_speed.isnull()].index
none_idx
```

Out[10]:
```
Index([133, 219, 316, 867, 882], dtype='int64')
```

Got the index of None entries, and fill in value by interpolating.

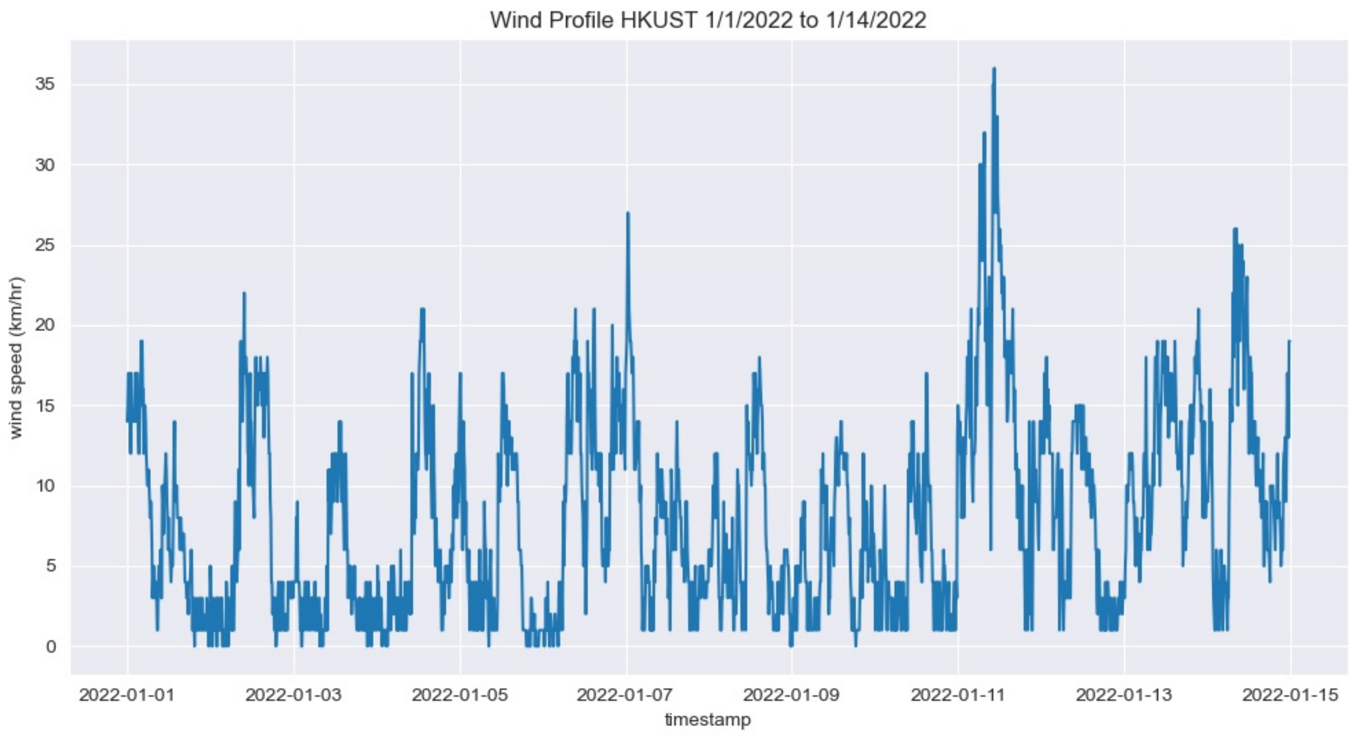
In [11]:
```
for idx in none_idx:
    wind_speed_df.loc[idx, 'wind_speed'] = (wind_speed_df.loc[idx-1, 'wind_speed'] + wind_speed_df.loc[idx+1, 'v

wind_speed_df.loc[none_idx]
```

Out[11]:

	timestamp	wind_speed
133	2022-01-01 22:10:00	1.5
219	2022-01-02 12:30:00	9.0
316	2022-01-03 04:40:00	2.0
867	2022-01-07 00:30:00	23.0
882	2022-01-07 03:00:00	11.5

Graph time series to see wind speed profile.

In [12]:
```
fig = plt.figure(figsize=(12, 6))
sns.set_style("darkgrid")
plt.plot(wind_speed_df.timestamp, wind_speed_df.wind_speed)
plt.xlabel("timestamp")
plt.ylabel("wind speed (km/hr)")
plt.title("Wind Profile HKUST 1/1/2022 to 1/14/2022")
plt.show()
```



Calculate the P_avail for one wind turbine with 32.5 ft blade length, a relatively small turbine for small communities.

In [13]:
```
radius = 65/2 # ft
air_density = 1.05093 # kg/m3; calculated using average elevation at HK (~1600 ft)
sweaping_area = (radius/3.281)**2 * math.pi # m^3

def convert_to_meter_per_sec(v):
    return v/3.6

def calculate_P_avail_wind(v):
    return 1/2 * air_density * sweaping_area * convert_to_meter_per_sec(v)**3

P_avail = [calculate_P_avail_wind(v) for v in wind_speed_df.wind_speed]
wind_speed_df['P_avail (kW * 10min)'] = P_avail

wind_speed_df.head()
```

Out[13]:

	timestamp	wind_speed	P_avail (kW * 10min)
0	2022-01-01 00:00:00	14.0	9526.306109
1	2022-01-01 00:10:00	15.0	11716.939912
2	2022-01-01 00:20:00	17.0	17056.392826
3	2022-01-01 00:30:00	17.0	17056.392826
4	2022-01-01 00:40:00	14.0	9526.306109

In [14]:
```
wind_speed_df.to_csv("data/wind_energy_generation.csv", index=False)
```

Calculate the expected average daily generation by this turbine.

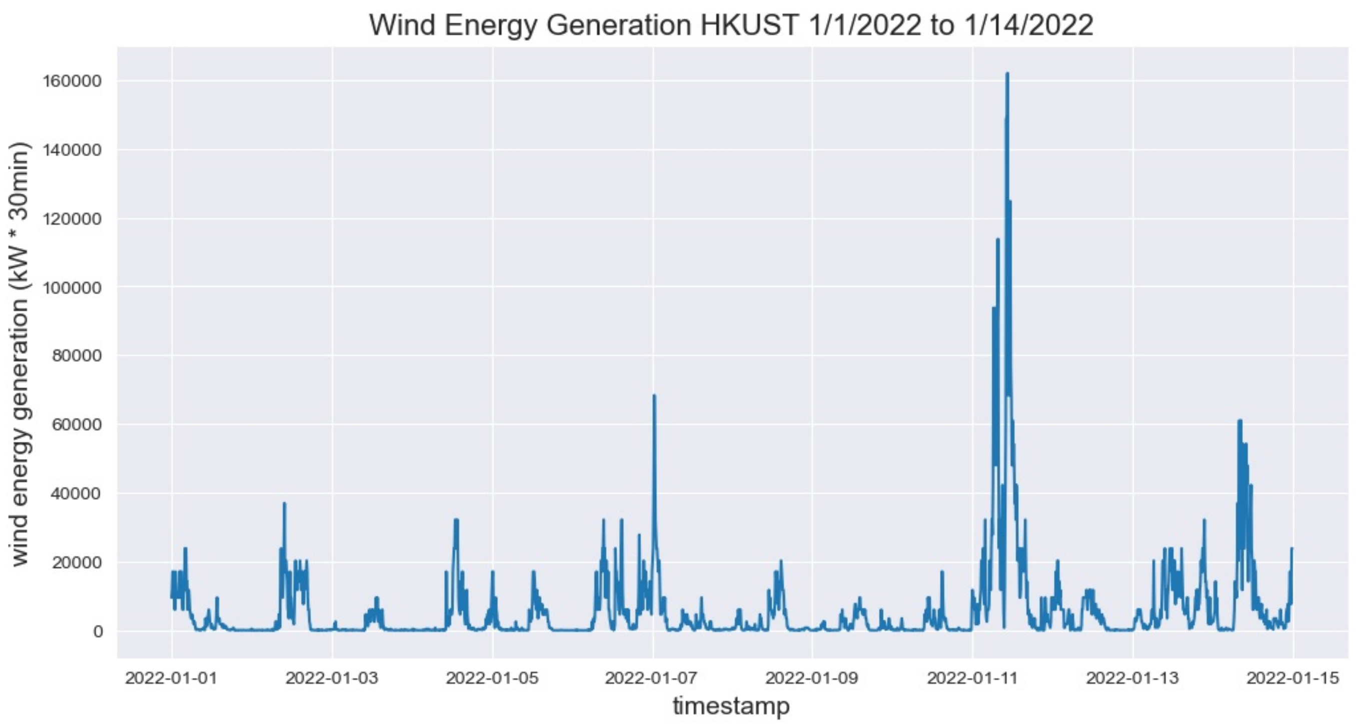
In [15]:
```
def daily_generation(P_avail_series):
    # divided by 6 to convert kW * 10min to kWh, divided by 14 to get daily average
    return P_avail_series.sum()/6/14

daily_generation(wind_speed_df['P_avail (kW * 10min)'])
```

Out[15]:
```
135580.58636918748
```

In [16]:
```
wind_speed_df.columns
```

```
Out[16]: Index(['timestamp', 'wind_speed', 'P_avail (kW * 10min)'], dtype='object')
```

```
In [17]: fig = plt.figure(figsize=(12, 6))
sns.set_style("darkgrid")
plt.plot(wind_speed_df.timestamp, wind_speed_df['P_avail (kW * 10min)'])
plt.xlabel("timestamp", fontsize=14)
plt.ylabel("wind energy generation (kW * 30min)", fontsize=14)
plt.title("Wind Energy Generation HKUST 1/1/2022 to 1/14/2022", fontsize=16)
plt.show()
```



Generate Unit Solar Profile

Formula:

- Input: solar radiation profile, panel area
- Equation: P_avail (kWh) = solar panel area (m^2) * solar radiation (kWh/m^2/hr) * solar panel efficiency (%) * performance ratio (0.75 - 0.9)

Data Source: https://data.gov.hk/en-data/dataset/hk-hko-rss-latest-one-minute-solar-radiation-info

*Here the solar radiation is direct solar radiation

```
In [18]: directory_path = "data/HK_solar_radiation_profile"

files = sorted(os.listdir(directory_path))[1:]

for file in files[:3]:
    print(file)
```

```
20220101-0011-latest_1min_solar.csv
20220101-0019-latest_1min_solar.csv
20220101-0029-latest_1min_solar.csv
```

```
In [19]: valid_files = files[:total_time_steps_10min]

curr_time = datetime(2022, 1, 1, 0, 0)

timestamps = []
direct_solar_radiation = [] # watt/m^2

for file in valid_files:
    solar_10min_df = pd.read_csv(f'{directory_path}/{file}')
    solar_10min_HKUST = solar_10min_df[solar_10min_df['Automatic Weather Station'] == 'King\'s Park']
    solar_flt = solar_10min_HKUST['Direct Solar Radiation(watt/square meter)'].values[0]
    if math.isnan(solar_flt):
        solar = None # fill in None value if there's no record at the time step
    else: solar = int(solar_flt)*10 # observation of radiation is last 1min. Mutiply by 10 for the sampling rate
    timestamps.append(curr_time)
    direct_solar_radiation.append(solar)
    curr_time += timedelta(minutes = 10)
```

```
solar_df = pd.DataFrame({"timestamp":timestamps, "direct_solar_radiation":direct_solar_radiation})
```

Fill in None values through interpolation

In [20]:
```
solar_df.direct_solar_radiation.isnull().sum()
```

Out[20]:
```
1
```

In [21]:
```
none_idx = solar_df.direct_solar_radiation[solar_df.direct_solar_radiation.isnull()].index
none_idx
```

Out[21]:
```
Index([994], dtype='int64')
```

In [22]:
```
idx = none_idx[0]
solar_df.loc[idx, 'direct_solar_radiation'] = (solar_df.loc[idx-1, 'direct_solar_radiation'] + solar_df.loc[idx
```

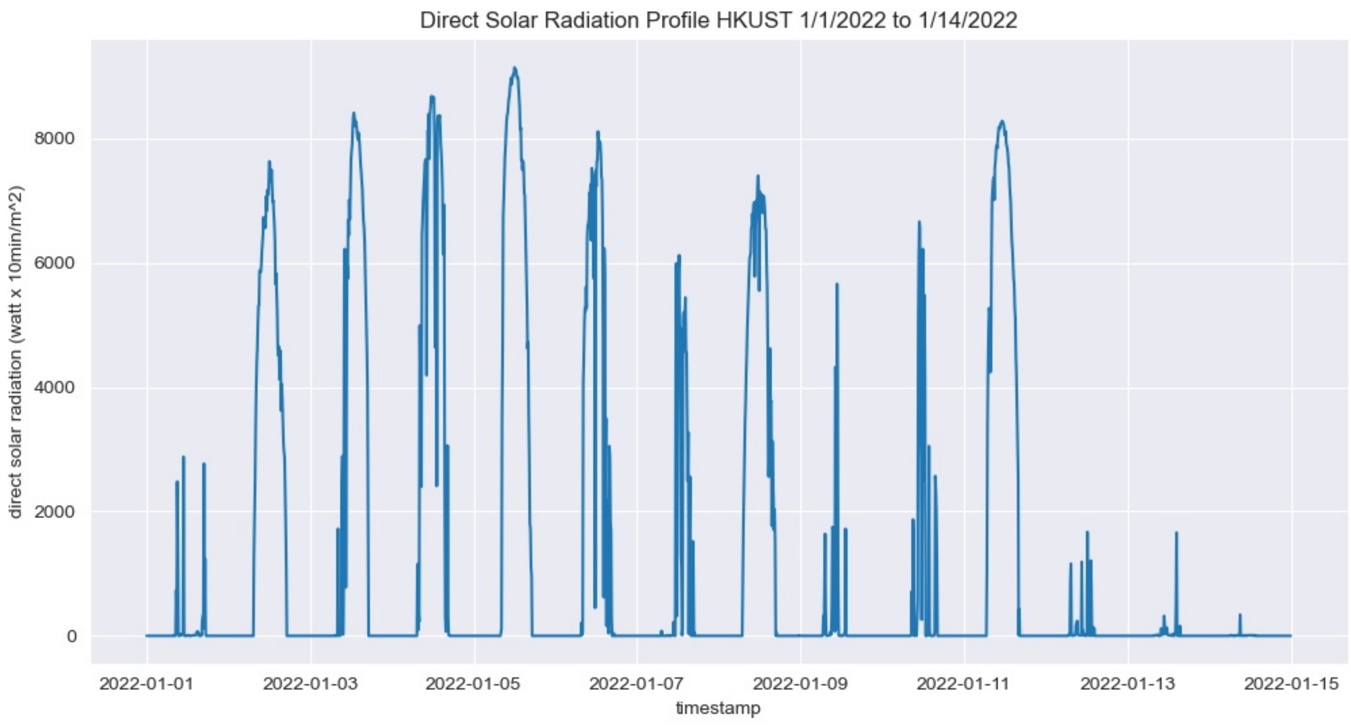
In [23]:
```
solar_df.loc[idx]
```

Out[23]:
```
timestamp                 2022-01-07 21:40:00
direct_solar_radiation                    0.0
Name: 994, dtype: object
```

In [24]:
```
solar_df.direct_solar_radiation.isnull().sum()
```

Out[24]:
```
0
```

In [25]:
```
fig = plt.figure(figsize=(12, 6))
sns.set_style("darkgrid")
plt.plot(solar_df.timestamp, solar_df.direct_solar_radiation)
plt.xlabel("timestamp")
plt.ylabel("direct solar radiation (watt x 10min/m^2)")
plt.title("Direct Solar Radiation Profile HKUST 1/1/2022 to 1/14/2022")
plt.show()
```



Here we assume we're installing this particular solar panel: https://shopsolarkits.com/products/rich-solar-mega-550-watt

According to their webpage, the specs of a solar panel is:

- Width x Height: 89.7 x 44.6 in
- Module Efficiency (%): 21.3%

Assuming a performance ratio of 0.85.

As a reminder, P_avail (kWh) = solar panel area (m^2) * solar radiation (kWh/m^2) * solar panel efficiency (%) * performance ratio (0.75 - 0.9)

Here we calculate watts instead of kW.

In [26]:
```
w = 89.7 # inch
h = 44.6 # inch
effi = 0.213
```

```
performance_ratio = 0.85

def solar_panel_area(w, h):
    return w * h/39.37**2

def calculate_P_avail_solar(radiation, panel_num=100):
    '''Calculate P_avail for panel_num number of solar panels. panel_num default to 100 panels.'''
    return solar_panel_area(w, h) * effi * performance_ratio * panel_num * radiation

P_avail = [calculate_P_avail_solar(radiation) for radiation in solar_df.direct_solar_radiation]
solar_df['P_avail(watt * 10min)'] = P_avail

solar_panel_area(w, h), solar_df.loc[6*16]
```

Out[26]:
```
(2.58105032339097,
 timestamp                 2022-01-01 16:00:00
 direct_solar_radiation                   30.0
 P_avail(watt * 10min)             1401.897483
 Name: 96, dtype: object)
```

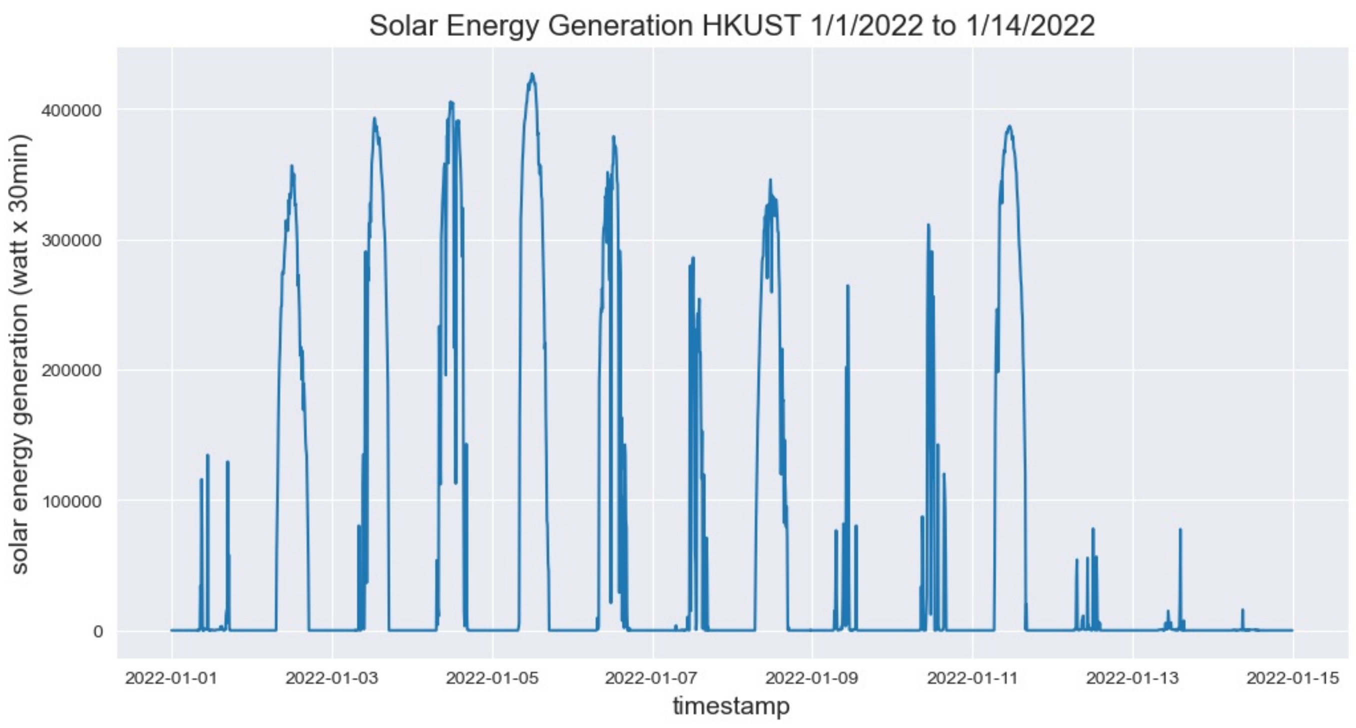
Calculate daily generation (in WattsHr)

In [27]:
```
def daily_generation(P_avail_series):
    '''Calculate average daily solar power generation.
    Sum over 2 weeks of generation profile, divided by 6 to convert W * 10min to Wh, then divided by 14 to get 
    return P_avail_series.sum()/6/14

daily_generation(solar_df['P_avail(watt * 10min)'])
```

Out[27]:
```
1367206.083527098
```

In [28]:
```
solar_df.to_csv("data/solar_energy_generation.csv", index=False)
```

In [29]:
```
fig = plt.figure(figsize=(12, 6))
sns.set_style("darkgrid")

x = solar_df.timestamp.iloc[::3].values
y = solar_df['P_avail(watt * 10min)'].iloc[::3].values

plt.plot(solar_df.timestamp, solar_df['P_avail(watt * 10min)'])
plt.xlabel("timestamp", fontsize=14)
plt.ylabel("solar energy generation (watt x 30min)", fontsize=14)
plt.title("Solar Energy Generation HKUST 1/1/2022 to 1/14/2022", fontsize=16)
plt.show()
```



Load and DER Constants and Variables

In [30]:
```
load_df = pd.read_csv("data/load_cleaned.csv")
load_df.head()
```

Out[30]:

	timestamps	P_load (kW * 10min)
0	2022-01-01 00:00:00	27543.416667
1	2022-01-01 00:10:00	27543.416667
2	2022-01-01 00:20:00	27543.416667
3	2022-01-01 00:30:00	27543.416667
4	2022-01-01 00:40:00	27543.416667

In [31]:
```
load_df.tail()
```

Out[31]:

	timestamps	P_load (kW * 10min)
2011	2022-01-14 23:10:00	32485.116667
2012	2022-01-14 23:20:00	32485.116667
2013	2022-01-14 23:30:00	32485.116667
2014	2022-01-14 23:40:00	32485.116667
2015	2022-01-14 23:50:00	32485.116667

In [32]:
```
P_load = list(load_df["P_load (kW * 10min)"].values)
```

In [33]:
```
solar_df = pd.read_csv("data/solar_energy_generation.csv")
wind_df = pd.read_csv("data/wind_energy_generation.csv")
solar_df.head()
```

Out[33]:

	timestamp	direct_solar_radiation	P_avail(watt * 10min)
0	2022-01-01 00:00:00	0.0	0.0
1	2022-01-01 00:10:00	0.0	0.0
2	2022-01-01 00:20:00	0.0	0.0
3	2022-01-01 00:30:00	0.0	0.0
4	2022-01-01 00:40:00	0.0	0.0

In [34]:
```
P_solar = list(solar_df["P_avail(watt * 10min)"].values)
P_solar = [p/1000 for p in P_solar]
```

In [35]:
```
wind_df.head()
```

Out[35]:

	timestamp	wind_speed	P_avail (kW * 10min)
0	2022-01-01 00:00:00	14.0	9526.306109
1	2022-01-01 00:10:00	15.0	11716.939912
2	2022-01-01 00:20:00	17.0	17056.392826
3	2022-01-01 00:30:00	17.0	17056.392826
4	2022-01-01 00:40:00	14.0	9526.306109

In [36]:
```
P_wind = list(wind_df["P_avail (kW * 10min)"].values)
```

In [37]:
```
num_intervals = 24 * 14 * 6 # biweekly 10-min intervals
num_intervals
```

Out[37]:
```
2016
```

In [38]:
```
P_grid = cp.Variable(num_intervals)
P_battery_max = cp.Variable()
P_battery = cp.Variable(num_intervals)
P_EV = cp.Variable(num_intervals)
P_battery_charge = cp.Variable(num_intervals)
```

In [39]:
```
P_grid.shape
```

Out[39]:
```
(2016,)
```

In [40]:
```
a_P_grid = 1.5 # HK$/kWh
a_P_not_met = 1000 # HK$/kWh (simulating the need for a diesel generator including initial investment and genera
a_wind_turbine = 520000 * 5/8 # HK$ adjusted by per capita GDP ratio
a_solar_panel = 30000 * 5/8
a_battery = 6220 * 5/8
a_energy_curtailed = 0.12 / 0.13 * 5/8 # 0.12 USD / 0.13 (ratio USD to HK$) adjusted by per capita GDP ratio
```

General Algorithm

The optimization is formulated as a 2-step problem where battery size and autonomy are optimized first concurrently, and then resilience is optimized based on the battery size obtained.

In [41]:
```
print(P_load[:5]) # sanity check
```

```
[27543.416666666668, 27543.416666666668, 27543.416666666668, 27543.416666666668, 27543.416666666668]
```

Solving for Autonomy: Constraints

In [42]:
```
battery_balance_RHS = P_battery[:num_intervals-1] + P_battery_charge[:num_intervals-1]
battery_constraints = [P_battery[0] == P_battery_max/2, P_battery[1:num_intervals] == battery_balance_RHS,
                       P_battery <= P_battery_max, P_battery >= 0, P_battery_max >= 0,
                      P_battery_charge <= P_battery_max, -P_battery_max <= P_battery_charge]

c1 = cp.Variable(integer=True)
c2 = cp.Variable(integer=True)
P_energy_curtailed = cp.Variable(num_intervals)
renewable_gen = c1*P_solar[:num_intervals] + c2*P_wind[:num_intervals]
wind_gen = c2*P_wind[:num_intervals]
solar_gen = c1*P_solar[:num_intervals]
renewable_gen = solar_gen + wind_gen
power_gen = renewable_gen + P_grid
power_balance_constraints = [P_load[:num_intervals] + P_battery_charge + P_energy_curtailed - power_gen == 0, 0
                             0 <= wind_gen, 0 <= solar_gen, 0 <= power_gen, P_energy_curtailed >= 0, P_energy_cu
```

In [43]:
```
constraints = itertools.chain(battery_constraints, power_balance_constraints)

# convert a_P_grid to HK$/kWh by dividing it by 6

# Switch between the two objective function to estimate costs and autonomy with or without considering energy cu
# obj_fc = cp.Minimize(a_battery*P_battery_max + a_solar_panel * c1 * 12/100 + a_wind_turbine * c2 + a_P_grid*cp
obj_fc = cp.Minimize(a_battery*P_battery_max + a_solar_panel * c1 * 12/100 + a_wind_turbine * c2 + a_P_grid*cp.s
problem = cp.Problem(obj_fc, constraints)
problem.solve(verbose=True)
```

```
===============================================================================
                                     CVXPY
                                     v1.6.0
===============================================================================
(CVXPY) Dec 13 02:57:19 AM: Your problem has 8067 variables, 24193 constraints, and 0 parameters.
(CVXPY) Dec 13 02:57:19 AM: It is compliant with the following grammars: DCP, DQCP
(CVXPY) Dec 13 02:57:19 AM: (If you need to solve this problem multiple times, but with different data, consider
using parameters.)
(CVXPY) Dec 13 02:57:19 AM: CVXPY will first compile your problem; then, it will invoke a numerical solver to ob
tain a solution.
(CVXPY) Dec 13 02:57:19 AM: Your problem is compiled with the CPP canonicalization backend.
-------------------------------------------------------------------------------
                                  Compilation
-------------------------------------------------------------------------------
(CVXPY) Dec 13 02:57:19 AM: Compiling problem (target solver=SCIPY).
(CVXPY) Dec 13 02:57:19 AM: Reduction chain: Dcp2Cone -> CvxAttr2Constr -> ConeMatrixStuffing -> SCIPY
(CVXPY) Dec 13 02:57:19 AM: Applying reduction Dcp2Cone
(CVXPY) Dec 13 02:57:19 AM: Applying reduction CvxAttr2Constr
(CVXPY) Dec 13 02:57:19 AM: Applying reduction ConeMatrixStuffing
(CVXPY) Dec 13 02:57:19 AM: Applying reduction SCIPY
(CVXPY) Dec 13 02:57:19 AM: Finished problem compilation (took 1.386e-02 seconds).
-------------------------------------------------------------------------------
                                Numerical solver
-------------------------------------------------------------------------------
(CVXPY) Dec 13 02:57:19 AM: Invoking solver SCIPY  to obtain a solution.
Solver terminated with message: Optimization terminated successfully. (HiGHS Status 7: Optimal)
-------------------------------------------------------------------------------
                                    Summary
-------------------------------------------------------------------------------
(CVXPY) Dec 13 02:57:19 AM: Problem status: optimal
(CVXPY) Dec 13 02:57:19 AM: Optimal value: 1.058e+07
(CVXPY) Dec 13 02:57:19 AM: Compilation took 1.386e-02 seconds
(CVXPY) Dec 13 02:57:19 AM: Solver (including time spent in interface) took 6.722e-01 seconds
```

Out[43]:
```
10582243.648011215
```

In [44]:
```
import math
import statistics

battery = a_battery*P_battery_max * 0.13
solar = a_solar_panel * c1.value * 12/100 * 0.13
wind = a_wind_turbine * c2.value * 0.13
grid = a_P_grid*sum(P_grid.value)/6 * 0.13
```

```
curtailed = a_energy_curtailed * sum(P_energy_curtailed.value)/6 * 0.13
total = problem.value*0.13

print(f"Result break down (excluding minimizing curtailment): \n\n{round(total/1000/1000, 2)} million USD inclu
print(f"Average Power from grid: {round(sum(P_grid.value)/6/14, 2)} kWh (= {round(sum(P_grid.value)/6/14/1000,
print(f"Average Renewable Energy Generation: {round(sum(renewable_gen.value)/6/14,2)} kWh (= {round(sum(renewab
print(f"Number of small solar panels: {round(float(c1.value))}, costs: {round(solar, 2)} USD (= {round(solar/10
print(f"Number of wind turbine with 32.5ft blade length: {c2.value}, costs: {round(wind, 2)} USD (= {round(wind,
print(f"Battery storage size: {round(float(P_battery_max.value))}")

renewable_percentage = round((sum(renewable_gen.value) - sum(P_energy_curtailed.value))/sum(P_load)*100, 2)
print(f"\n{renewable_percentage} % of total energy used is generated using solar and wind. (Curtailed energy wa

P_load_cost = a_P_grid*sum(P_load)*0.13/6
print(f"Two week P_grid costs: {round(grid, 2)} USD (= {round(grid/1000/1000, 2)} million USD)")
print(f"Two week P_grid without renewable costs: {round(P_load_cost/1000/1000, 2)} million USD, microgrid reduc

print(f"Average Power curtailed: {round(statistics.mean(P_energy_curtailed.value)*6, 2)} kWh (= {round(statisti
print(f"Two week P_energy_curtailed costs: {round(curtailed, 2)} USD (= {round(curtailed/1000/1000, 2)} million
```

```
Result break down (excluding minimizing curtailment):

1.38 million USD including setup costs with expectation of on average 760733.36 kWh daily load, equivalent to 76
0.73 MGhr (including solar/wind installation & purchase costs and curtailed power cost)
Average Power from grid: 413785.41 kWh (= 413.79 MWhr)
Average Renewable Energy Generation: 840600.46 kWh (= 840.6 MWhr)
Number of small solar panels: 119, costs: 34807.5 USD (= 34.81k USD)
Number of wind turbine with 32.5ft blade length: 5.0, costs: 211250.0 USD (= 211.25 k USD)
Battery storage size: 0

45.61 % of total energy used is generated using solar and wind. (Curtailed energy was subtracted)

Two week P_grid costs: 1129634.17 USD (= 1.13 million USD)
Two week P_grid without renewable costs: 2.08 million USD, microgrid reduced cost by 45.61 %, overall reduced co
st by 33.76 % (taking into account solar and wind installation & purchase, and cost of energy curtailed)

Average Power curtailed: 123413.13 kWh (= 123.41 MWhr)
Two week P_energy_curtailed costs: 518335.13 USD (= 0.52 million USD)
```

In [45]:
```
P_battery_max.value
```

Out[45]:
```
array(-0.)
```

Solving Resilience (P_notmet)

In [46]:
```
c1.value, c2.value
```

Out[46]:
```
(array(119.), array(5.))
```

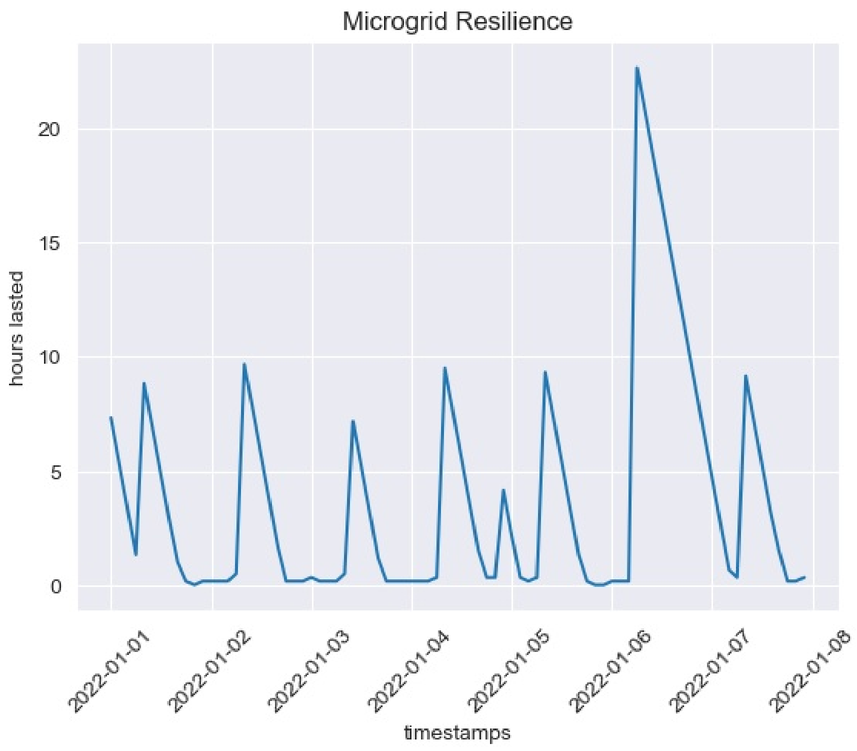
In [47]:
```
total_i = 12 * 7 # 7 days every 2 hours

hours_interval = 2 * 6 # number of 10 minute intervals in 2 hours
interval_start_idx = 0
num_intervals_ahead = 7 * 24 * 6 # 7 days ahead in 10 minute interval

# Maximizing resiliency for island mode trails in increments of 2 hours for 7 days
# Record the total lasting hours = resiliency for every trial and starting timestamps
total_lasting_hours = []
timestamps = []
for i in list(range(total_i)):
    if i == 0: timestamps.append(datetime(2022, 1, 1, 0, 0, 0))
    else: timestamps.append(timestamps[-1] + timedelta(minutes=hours_interval*10))
    interval_start = i * hours_interval
    interval_end = interval_start + num_intervals_ahead
    P_battery_max_resi = 50000
    P_battery_resi = cp.Variable(num_intervals_ahead)
    P_battery_charge_resi = cp.Variable(num_intervals_ahead)
    battery_balance_RHS_resi = P_battery_resi[:num_intervals_ahead-1] + P_battery_charge_resi[:num_intervals_ah
    battery_constraints_resi = [P_battery_resi[0] == P_battery_max_resi,
                                P_battery_resi[1:num_intervals_ahead] == battery_balance_RHS_resi,
                                P_battery_resi <= P_battery_max_resi, P_battery_resi >= 0]

    c1_resi = c1.value
    c2_resi = c2.value
    P_notmet = cp.Variable(num_intervals_ahead)
    P_energy_curtailed_resi = cp.Variable(num_intervals_ahead)
    renewable_gen_resi = [s*c1_resi + w*c2_resi for s, w in zip(P_solar[interval_start:interval_end], P_wind[in
    power_balance_constraints_resi = [P_load[interval_start:interval_end] - P_notmet + P_battery_charge_resi -
                                      0 <= P_notmet, P_notmet <= P_load[interval_start:interval_end],
                                     P_energy_curtailed_resi >= 0, P_energy_curtailed_resi <= renewable_gen_res
    constraints_resilience = itertools.chain(battery_constraints_resi, power_balance_constraints_resi)

    obj_fc_setup = a_battery * P_battery_max_resi + a_solar_panel * c1_resi * 5/100 + a_wind_turbine * c2_resi
    obj_fc_resilience = cp.Minimize(obj_fc_setup + a_P_not_met*cp.sum(P_notmet)/6)
    problem_resilience = cp.Problem(obj_fc_resilience, constraints_resilience)
    problem_resilience.solve()

    critical_load_p = 0.15 # critical load percentgae

    # Calculate the load not met percentage for the 7 days ahead.
    # If at a timestep t, the load not met percentage goes over 1 - critical load percentage,
    # Then the critical loads are not met, and the microgrid fails to function at time t
    # And t/6 would be the duration where the microgrid survivied island mode = the resiliency at starting time
    load_percentage = np.array([round(p_notmet/p_load*100, 2) for p_notmet, p_load in zip(P_notmet.value, P_loa
    indices = np.where(load_percentage/100 > 1 - critical_load_p)[0]
    if len(indices) == 0: indices = [num_intervals_ahead-1]
    print(f"Lasted: {(indices[0]+1)/6} hrs. Starting time: {timestamps[-1]}")
    total_lasting_hours.append(indices[0]/6)
```

```
Lasted: 7.5 hrs. Starting time: 2022-01-01 00:00:00
Lasted: 5.5 hrs. Starting time: 2022-01-01 02:00:00
Lasted: 3.5 hrs. Starting time: 2022-01-01 04:00:00
Lasted: 1.5 hrs. Starting time: 2022-01-01 06:00:00
Lasted: 9.0 hrs. Starting time: 2022-01-01 08:00:00
Lasted: 7.0 hrs. Starting time: 2022-01-01 10:00:00
Lasted: 5.0 hrs. Starting time: 2022-01-01 12:00:00
Lasted: 3.0 hrs. Starting time: 2022-01-01 14:00:00
Lasted: 1.1666666666666667 hrs. Starting time: 2022-01-01 16:00:00
Lasted: 0.3333333333333333 hrs. Starting time: 2022-01-01 18:00:00
Lasted: 0.16666666666666666 hrs. Starting time: 2022-01-01 20:00:00
Lasted: 0.3333333333333333 hrs. Starting time: 2022-01-01 22:00:00
Lasted: 0.3333333333333333 hrs. Starting time: 2022-01-02 00:00:00
Lasted: 0.3333333333333333 hrs. Starting time: 2022-01-02 02:00:00
Lasted: 0.3333333333333333 hrs. Starting time: 2022-01-02 04:00:00
Lasted: 0.6666666666666666 hrs. Starting time: 2022-01-02 06:00:00
Lasted: 9.833333333333334 hrs. Starting time: 2022-01-02 08:00:00
Lasted: 7.833333333333333 hrs. Starting time: 2022-01-02 10:00:00
Lasted: 5.833333333333333 hrs. Starting time: 2022-01-02 12:00:00
Lasted: 3.8333333333333335 hrs. Starting time: 2022-01-02 14:00:00
Lasted: 1.8333333333333333 hrs. Starting time: 2022-01-02 16:00:00
Lasted: 0.3333333333333333 hrs. Starting time: 2022-01-02 18:00:00
Lasted: 0.3333333333333333 hrs. Starting time: 2022-01-02 20:00:00
Lasted: 0.3333333333333333 hrs. Starting time: 2022-01-02 22:00:00
Lasted: 0.5 hrs. Starting time: 2022-01-03 00:00:00
Lasted: 0.3333333333333333 hrs. Starting time: 2022-01-03 02:00:00
Lasted: 0.3333333333333333 hrs. Starting time: 2022-01-03 04:00:00
Lasted: 0.3333333333333333 hrs. Starting time: 2022-01-03 06:00:00
Lasted: 0.6666666666666666 hrs. Starting time: 2022-01-03 08:00:00
Lasted: 7.333333333333333 hrs. Starting time: 2022-01-03 10:00:00
Lasted: 5.333333333333333 hrs. Starting time: 2022-01-03 12:00:00
Lasted: 3.3333333333333335 hrs. Starting time: 2022-01-03 14:00:00
Lasted: 1.3333333333333333 hrs. Starting time: 2022-01-03 16:00:00
Lasted: 0.3333333333333333 hrs. Starting time: 2022-01-03 18:00:00
Lasted: 0.3333333333333333 hrs. Starting time: 2022-01-03 20:00:00
Lasted: 0.3333333333333333 hrs. Starting time: 2022-01-03 22:00:00
Lasted: 0.3333333333333333 hrs. Starting time: 2022-01-04 00:00:00
Lasted: 0.3333333333333333 hrs. Starting time: 2022-01-04 02:00:00
Lasted: 0.3333333333333333 hrs. Starting time: 2022-01-04 04:00:00
Lasted: 0.5 hrs. Starting time: 2022-01-04 06:00:00
Lasted: 9.666666666666666 hrs. Starting time: 2022-01-04 08:00:00
Lasted: 7.666666666666667 hrs. Starting time: 2022-01-04 10:00:00
Lasted: 5.666666666666667 hrs. Starting time: 2022-01-04 12:00:00
Lasted: 3.6666666666666665 hrs. Starting time: 2022-01-04 14:00:00
Lasted: 1.6666666666666667 hrs. Starting time: 2022-01-04 16:00:00
Lasted: 0.5 hrs. Starting time: 2022-01-04 18:00:00
Lasted: 0.5 hrs. Starting time: 2022-01-04 20:00:00
Lasted: 4.333333333333333 hrs. Starting time: 2022-01-04 22:00:00
Lasted: 2.3333333333333335 hrs. Starting time: 2022-01-05 00:00:00
Lasted: 0.5 hrs. Starting time: 2022-01-05 02:00:00
Lasted: 0.3333333333333333 hrs. Starting time: 2022-01-05 04:00:00
Lasted: 0.5 hrs. Starting time: 2022-01-05 06:00:00
Lasted: 9.5 hrs. Starting time: 2022-01-05 08:00:00
Lasted: 7.5 hrs. Starting time: 2022-01-05 10:00:00
Lasted: 5.5 hrs. Starting time: 2022-01-05 12:00:00
Lasted: 3.5 hrs. Starting time: 2022-01-05 14:00:00
Lasted: 1.5 hrs. Starting time: 2022-01-05 16:00:00
Lasted: 0.3333333333333333 hrs. Starting time: 2022-01-05 18:00:00
Lasted: 0.16666666666666666 hrs. Starting time: 2022-01-05 20:00:00
Lasted: 0.16666666666666666 hrs. Starting time: 2022-01-05 22:00:00
```

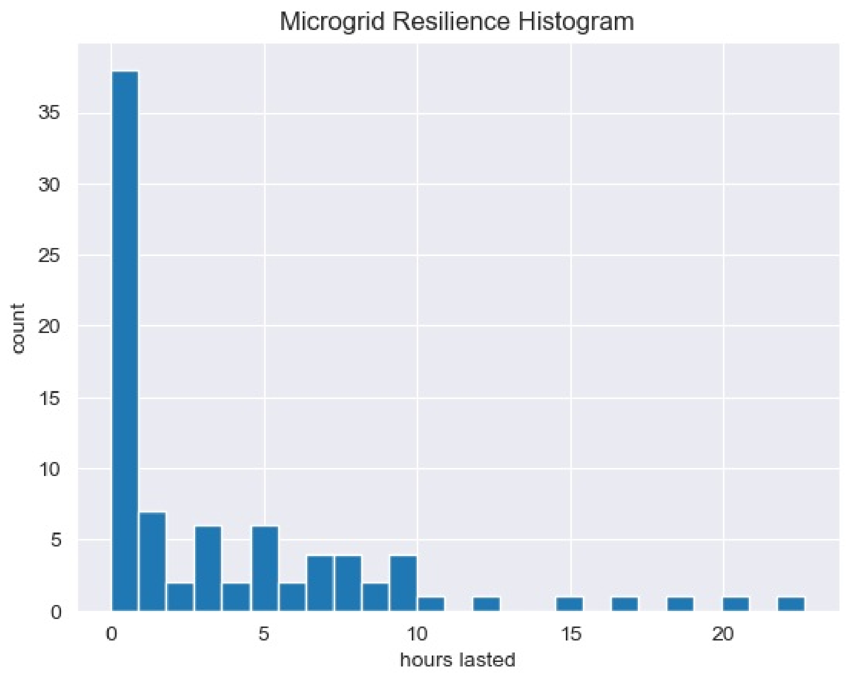
```
Lasted: 0.3333333333333333 hrs. Starting time: 2022-01-06 00:00:00
Lasted: 0.3333333333333333 hrs. Starting time: 2022-01-06 02:00:00
Lasted: 0.3333333333333333 hrs. Starting time: 2022-01-06 04:00:00
Lasted: 22.833333333333332 hrs. Starting time: 2022-01-06 06:00:00
Lasted: 20.833333333333332 hrs. Starting time: 2022-01-06 08:00:00
Lasted: 18.833333333333332 hrs. Starting time: 2022-01-06 10:00:00
Lasted: 16.833333333333332 hrs. Starting time: 2022-01-06 12:00:00
Lasted: 14.833333333333334 hrs. Starting time: 2022-01-06 14:00:00
Lasted: 12.833333333333334 hrs. Starting time: 2022-01-06 16:00:00
Lasted: 10.833333333333334 hrs. Starting time: 2022-01-06 18:00:00
Lasted: 8.833333333333334 hrs. Starting time: 2022-01-06 20:00:00
Lasted: 6.833333333333333 hrs. Starting time: 2022-01-06 22:00:00
Lasted: 4.833333333333333 hrs. Starting time: 2022-01-07 00:00:00
Lasted: 2.8333333333333335 hrs. Starting time: 2022-01-07 02:00:00
Lasted: 0.8333333333333334 hrs. Starting time: 2022-01-07 04:00:00
Lasted: 0.5 hrs. Starting time: 2022-01-07 06:00:00
Lasted: 9.333333333333334 hrs. Starting time: 2022-01-07 08:00:00
Lasted: 7.333333333333333 hrs. Starting time: 2022-01-07 10:00:00
Lasted: 5.333333333333333 hrs. Starting time: 2022-01-07 12:00:00
Lasted: 3.3333333333333335 hrs. Starting time: 2022-01-07 14:00:00
Lasted: 1.6666666666666667 hrs. Starting time: 2022-01-07 16:00:00
Lasted: 0.3333333333333333 hrs. Starting time: 2022-01-07 18:00:00
Lasted: 0.3333333333333333 hrs. Starting time: 2022-01-07 20:00:00
Lasted: 0.5 hrs. Starting time: 2022-01-07 22:00:00
```

Visualizing Resiliency Profile and Histogram

In [48]:
```python
plt.plot(timestamps, total_lasting_hours)
plt.xlabel('timestamps')
plt.ylabel('hours lasted')
plt.title('Microgrid Resilience')
plt.xticks(rotation=45)
plt.show()
```



In [49]:
```python
plt.hist(total_lasting_hours, bins=25)
plt.xlabel('hours lasted')
plt.ylabel('count')
plt.title('Microgrid Resilience Histogram')
plt.show()
```



Average, median and maximum Resiliency

```
In [50]: np.array(total_lasting_hours).mean(), statistics.median(total_lasting_hours) , max(total_lasting_hours)
```

```
Out[50]: (3.8214285714285716, 1.4166666666666665, 22.666666666666668)
```

Average Resiliency at daylight

```
In [51]: hrs = pd.Series(total_lasting_hours)
         hrs[hrs>1].mean()
```

```
Out[51]: 6.925925925925927
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js