CLOVER User Manual

Philip Sandwell PhD
Department of Physics
Imperial College London

prs09@imperial.ac.uk
philip.sandwell@gmail.com

Version 1

Abstract

CLOVER (Continuous Lifetime Optimisation of Variable Electricity Resources) is an open-source model for simulating and optimising community-scale electricity networks. Users can define the specifications and requirements of their project and investigate the technological, financial and environmental performance of systems under many different scenarios.

This manual provides a guide to setting up CLOVER, how to simulate and optimise electricity systems, and demonstrates some applications of the model outputs for use in research and practical projects.

Contents

1 Introduction

This guide provides a guide to using CLOVER (Continuous Lifetime Optimisation of Variable Electricity Resources) following the initial download. CLOVER is free and open-source for all to use, and the latest version of CLOVER is available on GitHub. Periodic updates to the code are also posted on Github to increase its functionality and fix bugs. This document provides examples of how to run the code, how the different modules and functions operate, and the kinds of outputs that can be obtained.

1.1 About CLOVER

CLOVER was developed at Imperial College London as a means of investigating how to support rural electrification strategies in developing countries. Under continuous development since 2015, CLOVER has been used for studies of electricity systems in Sub-Saharan Africa, South Asia and South America to explore the potential to provide reliable, affordable and sustainable power to rural and displaced communities.

CLOVER has the capabilities to model electricity systems of any size, from those serving individual households to large communities with diverse uses of energy and beyond, but has most commonly been used for village-scale minigrids serving hundreds of users. Its core functionality is to simulate and optimise systems supplied by any combination of solar, battery storage, diesel generation and a national grid connection to supply energy under specified performance parameters. CLOVER has been used to investigate technical case studies of specific systems, as well as broader analyses of the effects of rural electrification policies, for both academic and practitioner-focused audiences.

Some open-source examples of how CLOVER has been used include:

- Using solar and battery storage minigrids to support an unreliable grid network in providing electricity to domestic and commercial users in rural India (available here)
- Investigating how to support rural electrification policies in Rwanda and Nepal by increasing the utilisation of minigrid systems (available here)
- Exploring the opportunities for solar energy to offset diesel generation in refugee camps for camp operations and health services (available here)

CLOVER has also been used for investigations in several PhD theses (one example here) and Master's theses. The latter has included investigations into:

- Using minigrids to provide power to rural entrepreneurs in Rwanda
- Using solar power to provide electricity to health centres in remote areas of Kenya
- The opportunities for using health centres as anchor load clients in rural India
- Supporting educational electricity access at a women's education centre in Senegal
- The benefits of providing electricity for açaí processing in rural Brazil
- System design options for minigrids in refugee camps in Djibouti

1.1.1 What is CLOVER for?

CLOVER is a software tool for simulating and optimising community-scale electricity systems, typically minigrids to support rural electrification in developing countries. CLOVER allows users to model electricity demand and supply in locations and communities at an hourly resolution, for example allowing them to investigate how a specific electricity system might perform or to find the generation and storage capacity required to meet the needs of the community at the lowest cost. CLOVER can provide an insight into the technical performance, costs, and environmental impact of a system, and allow the user to evaluate many different scenarios to decide on the best way to provide sustainable, affordable and reliable electricity to the community.

1.1.2 What is CLOVER not for?

Fundamentally, CLOVER is an energy balance model which accounts for the generation and usage of electricity at an hourly resolution. The model is only as good as its data inputs and so the user should be aware of the many caveats that are attached to energy system modelling. CLOVER does not account for technical considerations such as power balancing in real systems, the compatibility of specific electronic components, or the many other practical considerations that would be relevant when designing the exact specifications of a system being deployed in the field. CLOVER can recommend the sizing, design and performance of a potential system, but the user should use this as a guide when using these results to inform real-life systems.

1.2 Getting CLOVER on your computer

1.2.1 Downloading CLOVER

Go to https://github.com/phil-sandwell/CLOVER to download the latest version of CLOVER by clicking the "Clone or download" button and selecting "Download ZIP". This will download a zipped folder containing all of the files you need to get started.

1.2.2 CLOVER file structure

The file structure from the download has two branches:

- a *Scripts* branch which contains Python files that the user runs and uses to generate outputs and perform simulations and optimisations,
- a *Locations* branch that describes individual locations and the specifics of a given scenario being investigated.

An example location, *Bahraich* in India, is included in the initial download for reference. New locations can be set up using the generic *New_Location* folder structure. It is recommended that the user makes a copy of the entire *New_Location* folder and renames it as their location to ensure that all of the necessary files are included.

1.2.3 CLOVER modules

CLOVER is designed to be modular, with each module or script performing a specific function (e.g. calculating the load of a community, getting the solar generation, or simulating the performance of a system). Each module can be run independently and some modules use (*import*) others to use as an input, for example the *Optimisation* module uses the *Energy_System*, which in turn uses the *Solar* module.

1.2.4 Running CLOVER

The recommended integrated development environment (IDE) for running CLOVER is Spyder although many others are available. This IDE is software which allows the user to view and edit scripts, run the code, run individual functions, and view the outputs easily. The easiest way to get Spyder is to download Anaconda which includes Spyder as a default package. CLOVER is written using Python 3 so the user should make sure that their Python environment uses this version; the packages that CLOVER uses are all included in the Anaconda distribution.

1.3 About this guide

This document is designed to guide a new user from the point of downloading CLOVER to being able to run their own simulations and optimisations. It contains worked examples of using the code and, owing to the format of the document, several pieces of code are included in this document which will not be necessary when running CLOVER using Spyder. These include viewing the input CSV files (which can be done in your file browser) and executing the code (which can be done by clicking the green arrow in Spyder) but are necessary to compile the code here. These will be highlighted as they come up.

Throughout this guide the **text in bold** highlights the steps you need to take to set up CLOVER, *text in italics* refers to the names of modules or other parts of the model structure, and `text in code` refers to variables or functions.

2 General setup

Once you have downloaded CLOVER you will need to set up the code to run on your machine. You may need to install Anaconda and/or Spyder (available here) in order to run the code if you have not used Python in the past.

After the initial download, each script will need to be updated to match your file path. This means that the modules that use other modules know where to look when accessing them, otherwise they will give an error. The scripts will also need to be updated to match the location you are investigating, although throughout this guide we will use Bahraich as the example location as all of the input data is provided in the CLOVER download.

2.1 Updating the file path and location

Each module (Python file, with a `.py` appendix) in the *Scripts* branch has a line in the initial definition function which sets the filepath of your CLOVER software and the location you are investigating. It looks something like this:

```
def __init__(self):
    self.location = 'Bahraich'
    self.CLOVER_filepath = '/***YOUR LOCAL FILE PATH***/CLOVER 4.0'
```

Our chosen location is Bahraich, so we do not need to update *self.location*, but we do need to input our file path. In my case this is saved in my documents (`Users/prs09`) in a folder called *CLOVER*, so the entire file path (`Users/prs09/CLOVER`) needs to be inserted here:

```
self.CLOVER_filepath = '/Users/prs09/CLOVER'
```

Some modules contain the filepath in several places so make sure to update all of them in each file. **Repeat this for all of the modules in the scripts folder so that they are all ready to use.**

2.2 Troubleshooting

If you have issues when updating the file paths or locations, check the following:

- Your file path contains a complete list of the folders where your CLOVER folder is saved
- You use the correct syntax for your operating system (e.g. "/" vs. "\" when stating the file path)
- You have not added or removed slashes from the end of the file path
- You have used the correct quotation marks, and they match (e.g. "double" or 'single' quotation marks, not 'grave accent')

2.3 Make a new location

Every location will require its own input files for the local generation, load demand and other factors. The easiest way to make a new location is to **copy the *New_Location* folder in the *Locations* folder and rename it as your chosen location**. This ensures that your folder structure is correctly set up, and maintains a copy of the generic folder in case you want to add a new location in the future.

In this guide we will use the default example location, *Bahraich*, which is a rural district in the state of Uttar Pradesh in northern India.

2.4 Get a *Renewables.ninja* API token

The *Solar* module gets solar generation data from another model developed at Imperial College London called *Renewables.ninja* which can provide renewables generation data for any location in the world at an hourly resolution and over several years. CLOVER automatically interfaces with the *Renewables.ninja* web interface but to do so you will need to **register for an account at https://www.renewables.ninja/register and use the API token in your version of CLOVER.** This is found in the "Profile" section of your *Renewables.ninja* account. More information about the API is available in the "Documentation" page on the website.

2.5 Establish your location

First you will need to provide details of the geographic location being investigated. These are contained in the *Location inputs* file in the *Location data* folder. You can edit these in the CSV file directly, but here we will import the data and print it to the screen to see the input data for Bahraich:

[1]:
```
import pandas as pd
location_inputs = pd.read_csv("/Users/prs09/Documents/CLOVER/Locations/Bahraich/
 ↪Location Data/Location inputs.csv")
location_inputs.head(len(location_inputs)-1)
```

[1]:

	Location	Bahraich	Location name
0	Country	India	Location country
1	Time difference	5.5	Time difference vs. UTC
2	Community size	100	Number of households in community
3	Community growth rate	0.01	Fractional growth rate per year
4	Years	20	years of simulation
5	Latitude	27.6	degrees of latitude (North +ve)
6	Longitude	81.6	degrees of longitude (East +ve)

Some of these variables should be self-explanatory: the location *Bahraich* is located in *India*. Others are less obvious: the time period under consideration has a maximum of 20 years (it can be less than this, but not more without modifying the code, so it is best to leave this as it is). Here we assume there are 100 households in the community with a household growth rate of 1% per year (0.01, expressed as a fraction). This is also where the *Renewables.ninja* API token should be copied

so that other parts CLOVER can use it later - as this is private I have not displayed mine in the table above.

Some are sensitive to positive or negative values, for example the time difference of India compared to UTC is +5:30 and so the input is 5.5, but countries west of UTC should use negative time differences (e.g. Honduras would be -6). Latitude and longitude are defined as North and East being positive and expressed as decimals; these are easily obtainable from Google Maps, for example.

3 Electricity generation

This section provides an overview of how to set up the electricity generation in CLOVER using its three modules solar, diesel and the national grid network. Further modules with new technologies could be added in the future to increase its functionality.

3.1 Solar

3.1.1 Preparation

The *Solar* module allows the user to get solar generation data for their location using the *Renewables.ninja* interface. Because this module relies on this external source of data it acts differently from other CLOVER modules, but ultimately returns the format of data that we expect from all of the generation files.

First, **complete the `PV generation inputs` CSV file in the *PV* folder**, which itself is located in the *Generation* folder. Let's take a look at the inputs:

[2]:
```
solar_inputs = pd.read_csv("/Users/prs09/Documents/CLOVER/Locations/Bahraich/
↪Generation/PV/PV generation inputs.csv",header=None)
solar_inputs.head(len(solar_inputs))
```

[2]:

	0	1	2
0	tilt	29	degrees above horizontal
1	azim	180	degrees from north
2	lifetime	20	years

These should all be straightforward: we assume our panels are south-facing (180° from North) and has a lifetime of 20 years, typical for a solar panel and used by CLOVER to account for module degradation. We also have stated that our panels will have a tilt (or elevation) of 29° above the horizontal, or make a 29° angle compared to flat ground (which would be 0°). This angle could (for example) be the tilt of a roof that the panels are assumed to be located on, or it could be the optimum angle that maximises total energy generation over the course of a year (which can be found using the *Renewables.ninja* web interface, or many other programmes).

3.1.2 Getting solar generation

Our goal is to get 20 years of solar generation data for our location at an hourly resolution. The data will be the output energy of one functional unit of nominal power of the PV for each hour of the year, assuming it is installed at the specified location and orientation. The functional unit of nominal power (also called the nameplate capacity) is the kilowatt-peak (kWp), defined to be instantaneous power output of a PV array under standard test conditions (1000 W/m^2). This value is most commonly reported when buying solar modules, for example a panel with a nominal power (or nameplate capacity) of 300 Wp (0.3 kWp) would provide 300 W of power when illuminated under standard test conditions, or have a higher output if the illumination were higher (and vice versa). The actual power received from a module of a given capacity depends on many factors,

for example the amount of illumination on the panel and the losses from a variety of sources, but *Renewables.ninja* calculates these automatically for us.

Using this functional unit makes it easy to scale up the generation capacity to the desired amount in a simulated system. First, check that you have already updated the location and filepaths of the *Solar* module and **run the script (using the green arrow in the Spyder console)**. Here we run it like so:

[3]:
```
import sys
sys.path.insert(0, '/Users/prs09/Documents/CLOVER/Scripts/Generation scripts/')
from Solar import Solar
```

The *Solar* module contains several nested functions which automatically provide a step-by-step process to send data to *Renewables.ninja* about the system parameters, account for the orientation of the system, and return ready-to-use data. We want to get 20 years of solar data but unfortunately the archives do not go back that far, so instead we will get 10 years of data and repeat it: this will allow us to keep the year-on-year variation whilst also covering our entire desired time period.

To do this we run a function which gets one year of solar generation and repeat the process until we have 10 years. In the Bahraich example we have data from 2007 to 2016, so let's replicate this. First we will generate the data for 2007 by running the following function:

[4]:
```
Solar().save_solar_output(gen_year = 2007)
```

This automatically saves a CSV file for us, and we can look at the first day's worth of entries:

[5]:
```
solar_generation_2007 = pd.read_csv("/Users/prs09/Documents/CLOVER/Locations/
 ↪Bahraich/Generation/PV/solar_generation_2007.csv",header=None)
solar_generation_2007.columns = ['Hour','kW']
print(solar_generation_2007[0:24])
```

```
    Hour     kW
0      0  0.000
1      1  0.000
2      2  0.000
3      3  0.000
4      4  0.000
5      5  0.000
6      6  0.000
7      7  0.009
8      8  0.184
9      9  0.419
10    10  0.591
11    11  0.687
12    12  0.733
13    13  0.697
14    14  0.594
15    15  0.417
16    16  0.192
```

```
17    17  0.017
18    18  0.000
19    19  0.000
20    20  0.000
21    21  0.000
22    22  0.000
23    23  0.000
```

As expected, we start getting solar generation from 6:00 (note that Python begins counting from 0, so the hours of the day run from 0 to 23) which peaks in the middle of the day and finishes by 17:00.

We then run the same function for the following nine years:

```
Solar().save_solar_output(gen_year = 2008)
Solar().save_solar_output(gen_year = 2009)
...
Solar().save_solar_output(gen_year = 2016)
```

Until we have 10 years of data. Now we can compile the data together using a function that stitches the data together in order, from the starting year of 2007, then repeats it to give the full 20 years of generation data:

[6]:
```
Solar().total_solar_output(start_year = 2007)
```

This file contains the total solar generation expected for the 1 kWp unit panel over the course of 20 years, which is what we aimed to achieve. This is the file that is imported in the *Energy_System* module and used to calculate the total PV generation in a system. **Use this process to generate solar data for your location.**

3.1.3 Extension and visualisation

For interest, let's see its cumulative generation over its lifetime, rounded to the nearest kWh:

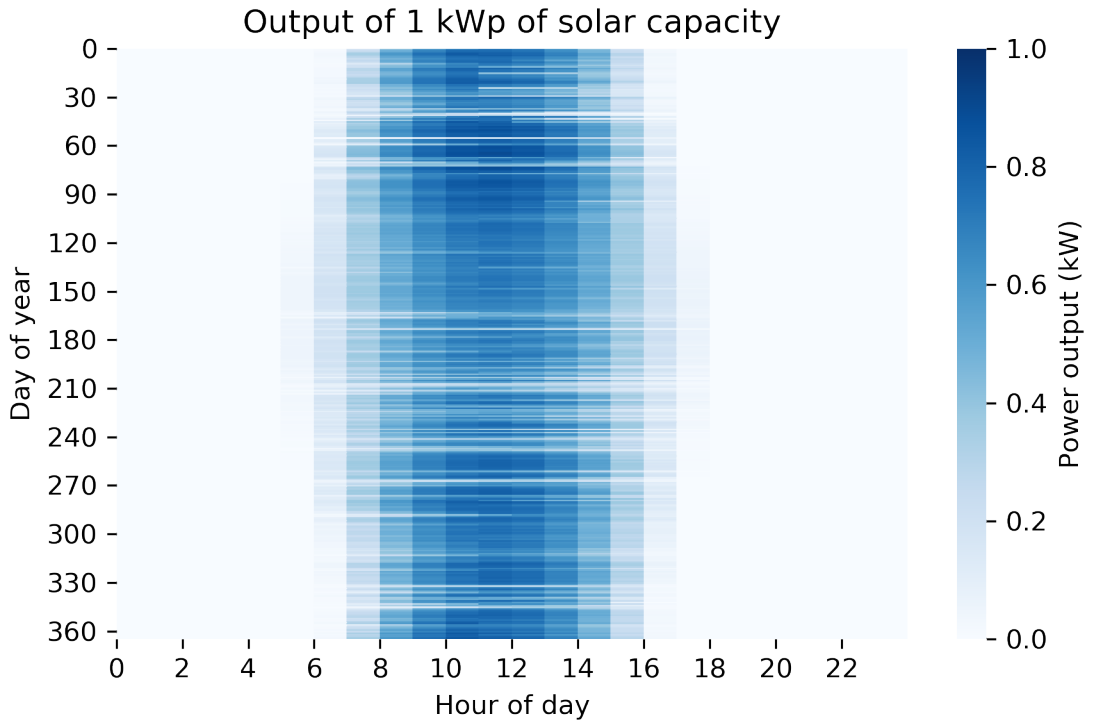
[7]:
```
import numpy as np
solar_generation_lifetime = pd.read_csv("/Users/prs09/Documents/CLOVER/Locations/
 ↪Bahraich/Generation/PV/solar_generation_20_years.csv")
total_generation = np.round(np.sum(solar_generation_lifetime['0.0']))
print('Cumulative generation: ' + str(total_generation)+' kWh')
print('Average generation: '+str(round(total_generation/(20*365)))+' kWh per␣
 ↪day')
```

```
Cumulative generation: 36655.0 kWh
Average generation: 5.0 kWh per day
```

This panel is expected to produce 36.7 MWh of energy over its lifetime, or around 5.0 kWh of energy per day - this is reasonable given the location of the panel in a relatively sunny location in India.

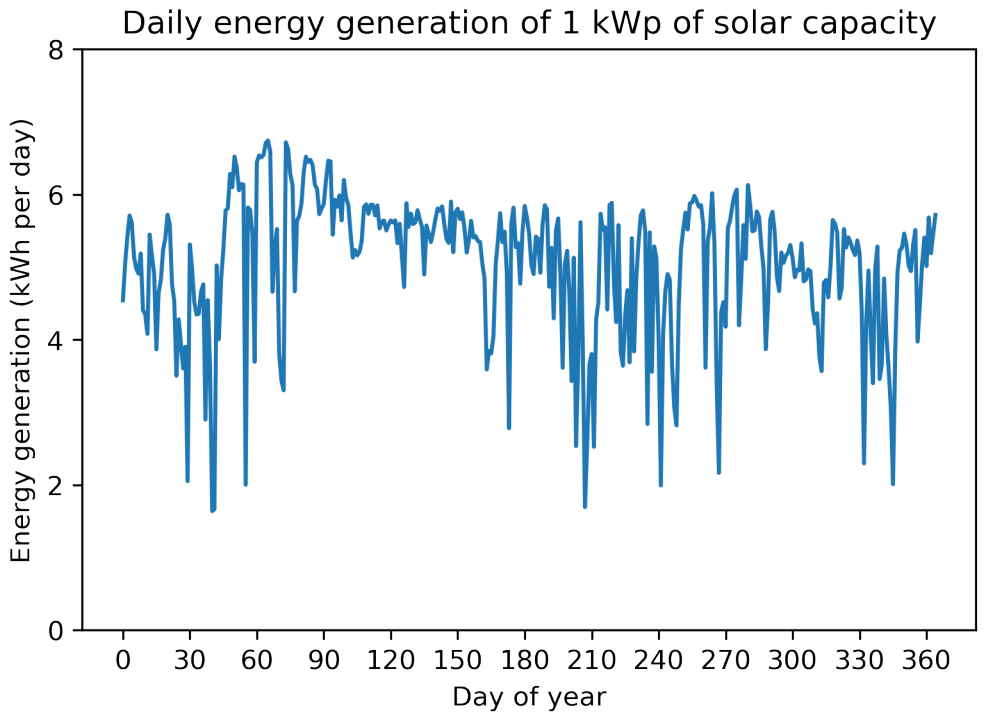
We can quickly visualise its generation over the course of the first year of its lifetime by taking the first 8760 hours (24 hours times 365 days) and plotting this as a heatmap:

[8]:
```
solar_gen_year = solar_generation_lifetime.iloc[0:8760]['0.0']
solar_gen_year = np.reshape(solar_gen_year.values,(365,24))

import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib as mpl

%matplotlib inline
mpl.rcParams['figure.dpi'] = 300

g = sns.heatmap(solar_gen_year,
                        vmin = 0.0, vmax = 1.0,
                        cmap = 'Blues', cbar_kws = {'label':'Power output (kW)'})
g.set(xticks = range(0,24,2), xticklabels = range(0,24,2),
      yticks = range(0,365,30), yticklabels = range(0,365,30),
      xlabel = 'Hour of day', ylabel = 'Day of year',
      title = 'Output of 1 kWp of solar capacity')
plt.xticks(rotation = 0)
plt.tight_layout()
plt.show()
```



As we might expect, the solar output varies throughout the year with longer periods of generation during the summer months. Some days have far less generation, potentially due to cloudy conditions. We can also see the total daily generation over the course of the year by taking the sum of the reshaped `solar_gen_year` object and plotting the result:

[9]:
```
solar_daily_sums = pd.DataFrame(np.sum(solar_gen_year,axis=1))
plt.plot(range(365),solar_daily_sums)
plt.xticks(range(0,365,30))
plt.yticks(range(0,9,2))
plt.xlabel('Day of year')
plt.ylabel('Energy generation (kWh per day)')
plt.title('Daily energy generation of 1 kWp of solar capacity')
plt.show()
```



3.1.4 Troubleshooting

The reason why we need to import the data from *Renewables.ninja* manually is because the API will flag multiple repeated requests to its servers and deny access. This makes it not possible to put the `Solar().save_solar_output(gen_year)` function into a for loop for convenience, as the code will be executed faster than that which the API will allow. This will also apply if you manually run the function too quickly, so if you receive an error message relating to the API then wait for around one minute and try again.

Periodically the *Renewables.ninja* changes its API settings which affects the way that the *Solar* module interacts with the website. This requires the CLOVER code to be updated, so if you identify this happening please email Philip Sandwell.

3.2 Grid

3.2.1 Preparation

The *Grid* module simulates the availability of the national grid network at the location, particularly when the grid is unreliable or has variable availability throughout the day. CLOVER assumes that when the grid is available it can provide an unlimited amount of power to satisfy the needs of the community for the entire hour in question or, if unavailable, no power can be drawn from it. The goal of the *Grid* module is to provide an hourly profile of whether the grid is available or not by using a user-specified availability profile (or several of them, if many are to be investigated).

First, **complete the `Grid inputs` CSV file** in the *Grid* folder, which itself is located in the *Generation* folder. Let's take a look at the inputs:

[10]:
```
grid_inputs = pd.read_csv("/Users/prs09/Documents/CLOVER/Locations/Bahraich/
 ↪Generation/Grid/Grid inputs.csv",header=0)
print(grid_inputs)
```

```
    Name  none  all  daytime  eight_hours  bahraich
0      0     0    1        0         0.33      0.57
1      1     0    1        0         0.33      0.61
2      2     0    1        0         0.33      0.54
3      3     0    1        0         0.33      0.50
4      4     0    1        0         0.33      0.48
5      5     0    1        0         0.33      0.48
6      6     0    1        0         0.33      0.46
7      7     0    1        0         0.33      0.34
8      8     0    1        1         0.33      0.25
9      9     0    1        1         0.33      0.30
10    10     0    1        1         0.33      0.35
11    11     0    1        1         0.33      0.35
12    12     0    1        1         0.33      0.33
13    13     0    1        1         0.33      0.29
14    14     0    1        1         0.33      0.32
15    15     0    1        1         0.33      0.35
16    16     0    1        1         0.33      0.35
17    17     0    1        1         0.33      0.32
18    18     0    1        0         0.33      0.39
19    19     0    1        0         0.33      0.14
20    20     0    1        0         0.33      0.18
21    21     0    1        0         0.33      0.46
22    22     0    1        0         0.33      0.47
23    23     0    1        0         0.33      0.51
```

This file describes five grid availability profiles, with each of the values corresponding to the probability that the grid will be available in the hour of the day specified on the left. Taking the sum of those values will give the average number of hours per day that the grid will be available. The profiles we have here are:

- `none` has no grid availability at all throughout the day, equivalent to not being connected to the grid
- `all` has full grid availability at all times item `daytime` has grid availability throughout the day (8:00 until 17:59) but never at night
- `eight_hours` will provide approximately eight hours of power, randomly available throughout the day
- `bahraich` is an example profile from data gathered from Bahraich district, where availability is higher in the early morning and late evening but lower during the day and early evening.

You can add further grid profiles by adding additional columns in the CSV file; they can have any name and values for grid availability must be in the range 0-1 as they represent probabilities. Save this file before moving on.

3.2.2 Getting grid availability profiles

Our goal is to get 20 years of grid availability data at an hourly resolution, represented as 0s (grid is unavailable) and 1s (grid is available), for each of the grid profiles listed in *Grid inputs*.

First, check that you have already updated the location and filepaths of the *Grid* module and **run the script (using the green arrow in the Spyder console)**. Here we run it like so:

[11]:
```
from Grid import Grid
```

The *Grid* module is quite straightforward as it uses a single function to take an input grid profile, repeats the probability profile over a 20 year period, uses binomial statistics to transsform them into a profile of 0s and 1s, and save this availability as a new output CSV file to use later. To do this for all of the grid profiles **run the following function** in your console:

```
Grid().get_lifetime_grid_status()
```

Once this function is completed your new grid profiles will appear in your *Grid* folder and be ready to use, with their file names corresponding to the name of the status given in the input file (e.g. `eight_hours_grid_status.csv`). Given the nature of the calculation, this function will take a while to complete. This is normal and fortunately this process needs to be completed only once: the same grid availability profiles can be used in each of your investigations for reproducibility.

3.2.3 Extension and visualisation

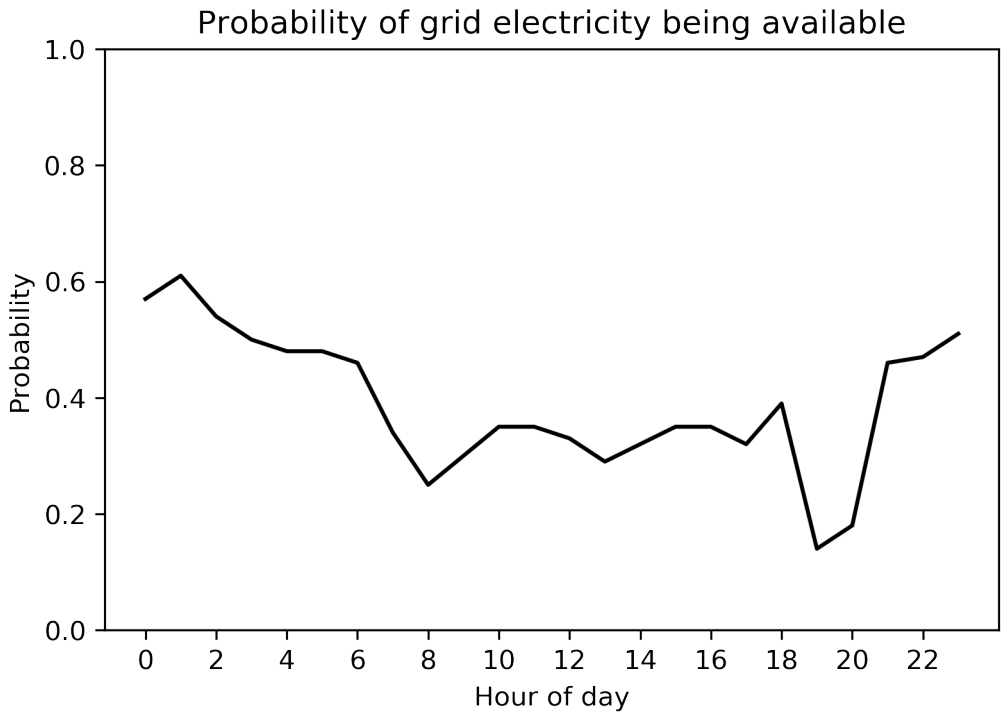
Let's take a closer look the `bahraich` grid profile by seeing how many hours per day it is available on average:

[12]:
```
bahraich_daily_hours = np.sum(grid_inputs['bahraich'],axis=0)
print(str(bahraich_daily_hours) + ' hours per day')
```

```
9.34 hours per day
```

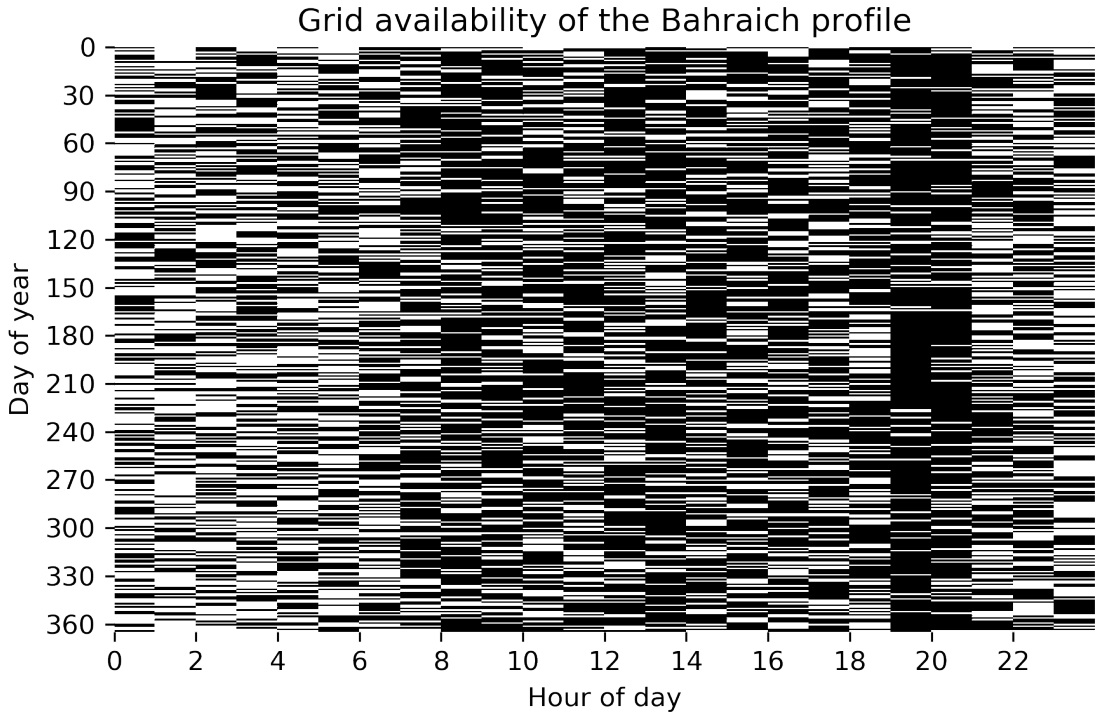
The `bahraich` profile gives an average of 9.34 hours of availability per day, relatively normal for that region of rural India but definitely has potential to be improved by installing a minigrid system. Now let's plot the grid availability throughout the day

[13]:
```
plt.plot(range(24),grid_inputs['bahraich'],color='k')
plt.xticks(range(0,24,2))
plt.yticks(np.arange(0,1.1,0.2))
plt.xlabel('Hour of day')
plt.ylabel('Probability')
plt.title('Probability of grid electricity being available')
plt.show()
```



As we saw before, the grid is most likely to be available in the early morning and least likely to be available in the evening. We can also investigate the `bahraich` grid availability profile that we made using `Grid().get_lifetime_grid_status()`, for example by viewing whether the grid is available or not over the first year. Here we plot it using a colour scheme of white meaning the grid is available, or black when it is not (i.e. a blackout):

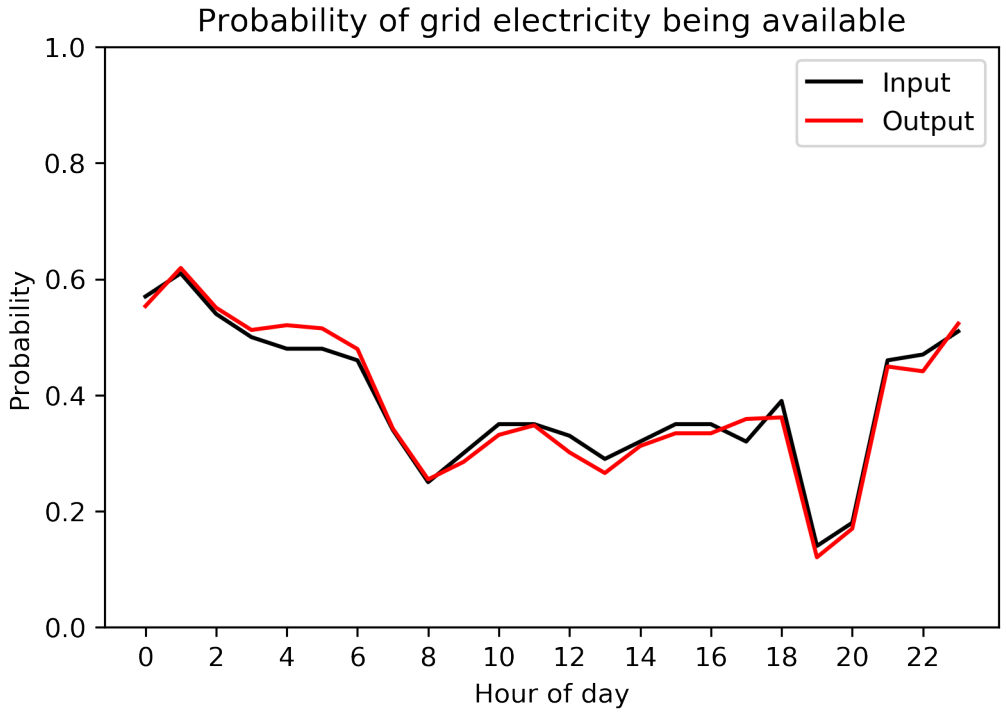
[14]:
```
bahraich_profile = pd.read_csv("/Users/prs09/Documents/CLOVER/Locations/Bahraich/
 ↪Generation/Grid/bahraich_grid_status.csv")
bahraich_profile = bahraich_profile.iloc[0:8760]['0']
bahraich_profile = np.reshape(bahraich_profile.values,(365,24))

g = sns.heatmap(bahraich_profile,
                vmin = 0.0, vmax = 1.0,
                cmap = 'Greys_r', cbar=False)
g.set(xticks = range(0,24,2), xticklabels = range(0,24,2),
      yticks = range(0,365,30), yticklabels = range(0,365,30),
      xlabel = 'Hour of day', ylabel = 'Day of year',
      title = 'Grid availability of the Bahraich profile')
plt.xticks(rotation = 0)
plt.tight_layout()
plt.show()
```



We can once again see the expected structure from the one-day availability input profile, but now with the randomness of day-to-day variations. There is greater availability in the mornings, but not always, and only on a very small number of days is power available between 19:00 and 21:00.

We can also compare the `bahraich` input profile to the synthesised availability profile to make sure they match up:

[15]:
```
plt.plot(range(24),grid_inputs['bahraich'],color='k',label='Input')
plt.plot(range(24),np.mean(bahraich_profile,axis=0),color='r',label='Output')
plt.legend()
plt.xticks(range(0,24,2))
plt.yticks(np.arange(0,1.1,0.2))
plt.xlabel('Hour of day')
plt.ylabel('Probability')
plt.title('Probability of grid electricity being available')
plt.show()
```



As we should expect, the output profile matches the input profile closely, but not exactly, owing to the random process that was used to generate it.

3.2.4 Troubleshooting

Rerunning the `Grid().get_lifetime_grid_status()` function will create new profiles: these will share the same statistics as a previously generated grid status but not be exactly the same. If you want to generate a new grid status but also want to keep ones you have already generated, it is recommended that you save your original files in another location and copy them back to the *Grid* folder once your new profile is complete.

CLOVER has the functionality to create grid profiles with variation throughout the day, but not throughout the year. If you want to include this, create an availability profile in a compatible format (20 years of hourly data, composed of 0s and 1s) using some other means, name it using

the `[name]_grid_status.csv` format, and copy this CSV file to the *Grid* folder. CLOVER should be able to use this CSV file as an input later, the same as any others that it generates.

3.3 Diesel

3.3.1 Preparation

The *Diesel* module is a passive script, meaning that the user needs to provide it with inputs but not actively run any of its functions (CLOVER does this automatically instead). In the current release of CLOVER diesel generation is treated as a backup source of power when the other sources are unable to provide electricity, filling in periods of blackouts after a simulation is complete to provide greater levels of reliability. This means it can be used as a backup source of power in a hybrid system (for example switching on automatically when renewable generation is not sufficient) but not as dispatchable generation coming on at user-specified times. This functionality will be included in the next update of CLOVER.

First, **complete the `Diesel inputs` CSV file** in the *Diesel* folder, which itself is located in the *Generation* folder. Let's take a look at the inputs:

[16]:
```
diesel_inputs = pd.read_csv("/Users/prs09/Documents/CLOVER/Locations/Bahraich/
 ↪Generation/Diesel/Diesel inputs.csv")
print(diesel_inputs)
```

```
    Diesel consumption   0.4    litres per kW capacity per hour
0  Diesel minimum load  0.35  Minimum capacity factor (0.0-1.0)
```

This input file contains just two variables but more parameters governing the usage of the diesel generator are included elsewhere. `Diesel consumption` refers to the hourly fuel consumption of the generator per kW of output, for example a generator providing 10 kW would use 4.0 litres of fuel per hour. CLOVER assumes that fuel consumption is constant per kW being supplied, although in real systems this may vary depending on the load factor.

`Diesel minimum load` is the lowest load factor that the generator is permitted to operate at (for example to avoid mechanical issues or degradation), expressed as a fraction. For example, a 5 kW generator would be forced to provide at least 1.75 kW (5.0 kW x 0.35) of power to ensure it runs above the minimum load factor even if the load were less than this, with the remaining energy being dumped.

3.3.2 Getting diesel generation

As mentioned above, as a passive script CLOVER uses the *Diesel* module automatically so the user does not need to run any of its functions. Nevertheless it is good practice to ensure that the script runs as expected, so check that you have already updated the location and filepaths of the *Diesel* module and **run the script (using the green arrow in the Spyder console)**. Here we run it like so:

[17]:
```
from Diesel import Diesel
```

Now the script is ready to be used by the rest of CLOVER.

4 Load profiles

CLOVER has the functionality to build load profiles from the bottom-up, summarised in the following steps:

1. Input the devices or appliances available in the community
2. Input the usage profile of each device throughout the day and year
3. Calculate the number of devices in use throughout the lifetime of the system
4. Calculate the daily utilisation profile of each device
5. Calculate the number of devices in use in every hour
6. Calculate the load profile of each device
7. Calculate the total load profile of the community.

This process is performed by the *Load* module, described in this section. The terms *device* and *appliance* are used interchangeably, meaning any piece of equipment that uses electricity, and specific meanings of other similar terms (for example *usage* and *utilisation*) will be made explicit where relevant.

4.1 Preparation

4.1.1 Input the devices in the community

CLOVER allows the user to include as many devices as desired in their load profile. These are input using the `Devices` input CSV in the *Load* folder for the given location, which can be edited directly. Let's look at the devices included for Bahraich:

[18]:
```
devices = pd.read_csv("/Users/prs09/Documents/CLOVER/Locations/Bahraich/Load/
↪Devices.csv")
devices.head(len(devices))
```

[18]:

	Device	Available	Power	Initial	Final	Innovation	Imitation	\
0	light	Y	3	2.00	4.00	0.04	0.50	
1	phone	Y	5	2.20	3.00	0.02	0.20	
2	radio	Y	10	0.28	0.35	0.01	0.20	
3	tv	Y	20	0.08	0.90	0.03	0.25	
4	laptop	Y	40	0.01	0.50	0.02	0.10	
5	fridge	Y	50	0.00	0.50	0.02	0.20	
6	fan	Y	10	0.10	2.00	0.04	0.30	
7	kerosene	Y	1	3.40	2.00	0.02	0.10	
8	mill	Y	750	0.01	0.10	0.02	0.20	
9	workshop	Y	350	0.05	0.05	0.02	0.20	
10	SME	Y	200	0.05	0.03	0.25	0.25	
11	streetlight	Y	25	0.20	0.20	0.00	0.00	

```
          Type
0     Domestic
1     Domestic
2     Domestic
3     Domestic
4     Domestic
5     Domestic
6     Domestic
7     Domestic
8   Commercial
9   Commercial
10  Commercial
11      Public
```

This CSV contains many variables, some of which are more obvious than others, summarised in the table below:

Variable	Explanation
`Device`	Unique name of the device
`Available`	Device is included (`Y`) or not (`N`) in the community load
`Power`	Device power consumption (W)
`Initial`	Average ownership of the device per household at the start of the time period
`Final`	Average ownership of the device per household at the end of the time period
`Innovation`	Parameter governing new households aquiring device
`Imitation`	Parameter governing households copying others
`Type`	Classification of the device as a `Domestic`, `Commercial` or `Public` device

Take the example of the device `light`, which represents an LED bulb. It is available in the load profile (`Available = Y`), has a power rating of 3 W, and classified as a `Domestic` load. At the start of the simulation period there is an average two LED bulbs per household (`Initial = 2.00`) which, over time, increases to four bulbs per household (`Final = 4.00`) by the end of the considered time period - which we defined to be to 20 years in Section 2.5.

The two remaining variables describe how quickly the average ownership per household increases: to generalise, `Innovation` describes how likely a household is to get a new appliance based on how much they like new things and `Imitation` describes how much a household is influenced by others already having the device. These are inputs into a model of how quickly devices diffuse throughout the community (described in more detail later) but, simply put, the larger these numbers the quicker households will acquire them. These should be treated almost as qualitative measures: the values for a more desirable appliance like a television should be higher than (for example) a radio. You can use later outputs from the *Load* module to check that your appliance

diffusion seems viable.

If you do not want to include any demand growth over time by keeping the number of appliances the same throughout the simulation, it is possible to turn off this feature simply by setting the values for `Initial` and `Final` to be the same, which will negate the `Innovation` and `Imitation` parameters (for example `streetlight` does this). Bear in mind that an increase in the number of households, defined when establishing the location, will result in an increase in the number of devices as the latter is calculated on the basis of the number of devices *per household*.

One of the devices, `kerosene`, is different from the others and is the only device which *must* remain present in the *Devices* file. CLOVER has the functionality to account for the usage of backup non-electric lighting devices, such as kerosene lamps, which are used during periods when electricity is unavailable. This is useful when investigating electricity systems with less than 100% reliability, for example. **This device is therefore necessary as an input for later functions and must be present here, and have corresponding utilisation profiles**, described later. If backup sources of non-electric lighting are not relevant to your investigation, set `Initial = 0` and `Final = 0` in the *Devices* file to not consider it whilst still allowing the other functions to operate as expected. If a different non-electric lighting source is relevant, such as candles, complete the input data for that source but the name must remain as `kerosene` in the *Devices* file.

With this in mind, complete the `Devices` input CSV for your investigation.

4.1.2 Input the device utilisation profiles

CLOVER first considers the service that appliances provide as a necessary step in understanding the electricity demand of the community, rather than jumping directly to the latter. This is because two devices may provide very similar services, and the times of using those devices may be very similar, but the electricity requirements could be very different: LED and incandescent light bulbs, for example. By considering the demand for the service as a first step it is possible to more easily consider issues such as the usage of efficient, low-power devices which are becoming more common.

The *utilisation* of a device is defined to be the probability that a device is being used, given that it is present in the community. Utilisation values can vary hourly (throughout the day) and monthly (throughout the year) and must be between 0 (the device is never used in a specific hour) to 1 (it is always used). Utilisation profiles for each device are found in the *Device utilisation* folder in the *Load* folder.

Every device listed in the `Devices` input CSV **must have a utilisation profile associated with it** for CLOVER to calculate the load demand correctly and be named `[device]_times`. Even if a device has `Available = N` in the *Devices* input CSV, CLOVER will still use the utilisation profile but not eventually include the load in the final total community demand. If you do not want to include a device at all it is much easier to delete its entry from the *Devices* input CSV.

Each utilisation profile is a 24 x 12 matrix of utilisation values, corresponding to the hour of the day for each month of the year. Let's take a look at the example of `light`:

[19]:
```
light_utilisation = pd.read_csv("/Users/prs09/Documents/CLOVER/Locations/
 ↪Bahraich/Load/Device utilisation/light_times.csv",header=None)
print(light_utilisation)
```

	0	1	2	3	4	5	6	7	8	9	10	11
0	0.39	0.39	0.39	0.39	0.39	0.39	0.39	0.39	0.39	0.39	0.39	0.39
1	0.39	0.39	0.39	0.39	0.39	0.39	0.39	0.39	0.39	0.39	0.39	0.39
2	0.39	0.39	0.39	0.39	0.39	0.39	0.39	0.39	0.39	0.39	0.39	0.39
3	0.39	0.39	0.39	0.39	0.39	0.39	0.39	0.39	0.39	0.39	0.39	0.39
4	0.39	0.39	0.39	0.39	0.48	0.56	0.46	0.39	0.39	0.39	0.39	0.39
5	0.39	0.39	0.47	0.48	0.23	0.12	0.28	0.45	0.53	0.57	0.40	0.39
6	0.58	0.48	0.22	0.00	0.00	0.00	0.00	0.00	0.00	0.05	0.37	0.53
7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
11	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
12	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
13	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
14	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
15	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
16	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
17	0.47	0.08	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.39	0.74	0.76
18	0.93	0.93	0.54	0.51	0.23	0.03	0.00	0.26	0.79	0.93	0.93	0.93
19	0.93	0.93	0.93	0.93	0.93	0.93	0.90	0.93	0.93	0.93	0.93	0.93
20	0.93	0.93	0.93	0.93	0.93	0.93	0.93	0.93	0.93	0.93	0.93	0.93
21	0.93	0.93	0.93	0.93	0.93	0.93	0.93	0.93	0.93	0.93	0.93	0.93
22	0.88	0.88	0.88	0.88	0.88	0.88	0.88	0.88	0.88	0.88	0.88	0.88
23	0.83	0.83	0.83	0.83	0.83	0.83	0.83	0.83	0.83	0.83	0.83	0.83

This device, representing an LED bulb from before, has a changing utilisation profile throughout the day: it is never used during the middle of the day (utilisation is 0.00), is very likely to be used in the evenings (up to 0.93), and some lights in the community are likely to be left on overnight (0.39). The utilisation of this device also changes throughout the year: in January (month 0) the utilisation at 18:00 is 0.93, but is 0.00 in July (month 6), owing to the changing times of sunset.

Making a representative utilisation profile will depend significantly on the specifics of your own investigation. These could come from primary data collection (indeed, the utilisation profile shown above did) or from your best estimate of what the demand for service is likely to be. Bear in mind that the probability that a device being used is an average over the entire community, so this should represent the utilisation of an "average" household without considering inter-household variations. To model device utilisation as the same throughout the year, use the same 24-value column for each of the 12 months.

For every device you have listed in the `Devices` input CSV, complete a corresponding utilisation profile matrix and ensure that is is called `[device]_times`.

4.2 Building load profiles

4.2.1 Calculating the number of devices in the community

The next step is to know how many devices there are in the community at any given time which, when combined with the utilisation profiles, will then allow us to calculate how many are in use. This step is relatively straightforward in that it takes information we have already input (into the `Devices` CSV file) and calculates everything for us. It also accounts for the growing number of devices in the community, if we chose to use that.

To calculate this automatically for all of the devices listed, **run the function**

```
Load().number_of_devices_daily()
```

in the console. This will take every device listed and calculate the number in the community on every day of our 20 year investigation period, and save them as a CSV file in the *Device ownership* folder in the *Load* folder with title format of `[device]_daily_ownership.csv`. Let's take a look at the `light` device again and look at the first and final number of devices in the community:

[20]:
```
light_ownership = pd.read_csv("/Users/prs09/Documents/CLOVER/Locations/Bahraich/
 ↪Load/Device ownership/light_daily_ownership.csv")
initial_light_ownership = int(light_ownership.iloc[0][1])
final_light_ownership = int(light_ownership.iloc[-1][1])
print('Initial light ownership = '+ str(initial_light_ownership))
print('Final light ownership = ' + str(final_light_ownership))
```

```
Initial light ownership = 200
Final light ownership = 487
```

The final `light` ownership perhaps looks higher than expected, but when we take into account how much the community has grown it makes sense:

[21]:
```
location_inputs = pd.read_csv("/Users/prs09/Documents/CLOVER/Locations/Bahraich/
 ↪Location Data/Location inputs.csv",index_col=0)
community_size = float(location_inputs.loc['Community size'][0])
growth_rate = float(location_inputs.loc['Community growth rate'][0])
years = float(location_inputs.loc['Years'][0])
final_community_size = int(community_size * (1.0 + growth_rate)**years)
print('Final community size: ' + str(final_community_size))

final_light_average_ownership = devices.iloc[0]['Final']
final_light_expected_ownership = int(final_community_size *␣
 ↪final_light_average_ownership)
print('Final expected light ownership: ' + str(final_light_expected_ownership))
```

```
Final community size: 122
Final expected light ownership: 488
```

The slight discrepancy in the two values likely comes from the diffusion model tending towards (but never quite reaching) the final value of average ownership and/or rounding errors.

4.2.2 Calculating the daily utilisation profiles

The utilisation profiles input for each device was defined by month but, if we assigned that utilisation profile for the whole month-long period, there would be a sharp transition between each month which would be unrealistic. To overcome this CLOVER interpolates between the monthly values to give a smooth transition between every day, meaning that there are no harsh boundaries where demand suddenly changes. This process is also performed automatically by CLOVER. **Run the following function in the console to calculate this:**

```
Load().get_device_daily_profile()
```

This function takes the (monthly) utilisation inputs and converts them to a daily utilisation input of size 365 x 24 and saves it in the *Device utilisation* folder in the *Load* folder using the title format `[device]_daily_times.csv`. CLOVER assumes that the values stated in the monthly utilisation profile apply to the middle day of each month when interpolating to the daily scale.

As an example, let's compare the monthly input values for January and February against the first week of daily values for the `light` device:

[22]:
```
jan_feb_inputs = light_utilisation[light_utilisation.columns[0:2]]
jan_feb_inputs.columns = ['Jan','Feb']
print('Input values for January and February:\n')
print(jan_feb_inputs)

light_daily_utilisation = pd.read_csv("/Users/prs09/Documents/CLOVER/Locations/
 ↪Bahraich/Load/Device utilisation/light_daily_times.csv",index_col=0)
light_daily_utilisation = round(light_daily_utilisation.iloc[0:7],3)
print('\nDaily varying values: \n')
print(light_daily_utilisation.T)
```

```
Input values for January and February:

     Jan   Feb
0   0.39  0.39
1   0.39  0.39
2   0.39  0.39
3   0.39  0.39
4   0.39  0.39
5   0.39  0.39
6   0.58  0.48
7   0.00  0.00
8   0.00  0.00
9   0.00  0.00
10  0.00  0.00
11  0.00  0.00
12  0.00  0.00
13  0.00  0.00
14  0.00  0.00
15  0.00  0.00
```

```
16  0.00  0.00
17  0.47  0.08
18  0.93  0.93
19  0.93  0.93
20  0.93  0.93
21  0.93  0.93
22  0.88  0.88
23  0.83  0.83

Daily varying values:

        0      1      2      3      4      5      6
0   0.390  0.390  0.390  0.390  0.390  0.390  0.390
1   0.390  0.390  0.390  0.390  0.390  0.390  0.390
2   0.390  0.390  0.390  0.390  0.390  0.390  0.390
3   0.390  0.390  0.390  0.390  0.390  0.390  0.390
4   0.390  0.390  0.390  0.390  0.390  0.390  0.390
5   0.390  0.390  0.390  0.390  0.390  0.390  0.390
6   0.555  0.557  0.559  0.560  0.562  0.564  0.566
7   0.000  0.000  0.000  0.000  0.000  0.000  0.000
8   0.000  0.000  0.000  0.000  0.000  0.000  0.000
9   0.000  0.000  0.000  0.000  0.000  0.000  0.000
10  0.000  0.000  0.000  0.000  0.000  0.000  0.000
11  0.000  0.000  0.000  0.000  0.000  0.000  0.000
12  0.000  0.000  0.000  0.000  0.000  0.000  0.000
13  0.000  0.000  0.000  0.000  0.000  0.000  0.000
14  0.000  0.000  0.000  0.000  0.000  0.000  0.000
15  0.000  0.000  0.000  0.000  0.000  0.000  0.000
16  0.000  0.000  0.000  0.000  0.000  0.000  0.000
17  0.615  0.605  0.594  0.584  0.574  0.563  0.553
18  0.930  0.930  0.930  0.930  0.930  0.930  0.930
19  0.930  0.930  0.930  0.930  0.930  0.930  0.930
20  0.930  0.930  0.930  0.930  0.930  0.930  0.930
21  0.930  0.930  0.930  0.930  0.930  0.930  0.930
22  0.880  0.880  0.880  0.880  0.880  0.880  0.880
23  0.830  0.830  0.830  0.830  0.830  0.830  0.830
```

Note that the daily utilisation CSV file has a slightly different structure (days are presented as rows) but we have transposed it here and rounded to three decimal places for comparability and readability. The hours where there is a difference between utilisation in January and February (hours 6 and 17) also vary in the daily profile, whilst the others remain the same as expected. As mentioned above, note that the utilisation input for January is not the same as the daily utilisation on day 0: the former is assumed to be for the middle day of the month in January, whereas the latter is halfway between the inputs for December and January. We can check this:

[23]:
```
december_utilisation_example = light_utilisation[light_utilisation.
 ↪columns[11]][6]
january_utilisation_example = light_utilisation[light_utilisation.columns[0]][6]
```

```
midpoint = round((december_utilisation_example + january_utilisation_example) /␣
↪2.0,3)

print('December utilisation at 6:00: ' + str(december_utilisation_example))
print('January utilisation at 6:00: ' + str(january_utilisation_example))
print('Midpoint utilisation between December and January: ' + str(midpoint))
print('Day 0 utilisation at 6:00: ' + str(light_daily_utilisation.iloc[0][6]))
```

```
December utilisation at 6:00: 0.53
January utilisation at 6:00: 0.58
Midpoint utilisation between December and January: 0.555
Day 0 utilisation at 6:00: 0.555
```

As expected, the two values match.

4.2.3 Calculating the number of devices in use

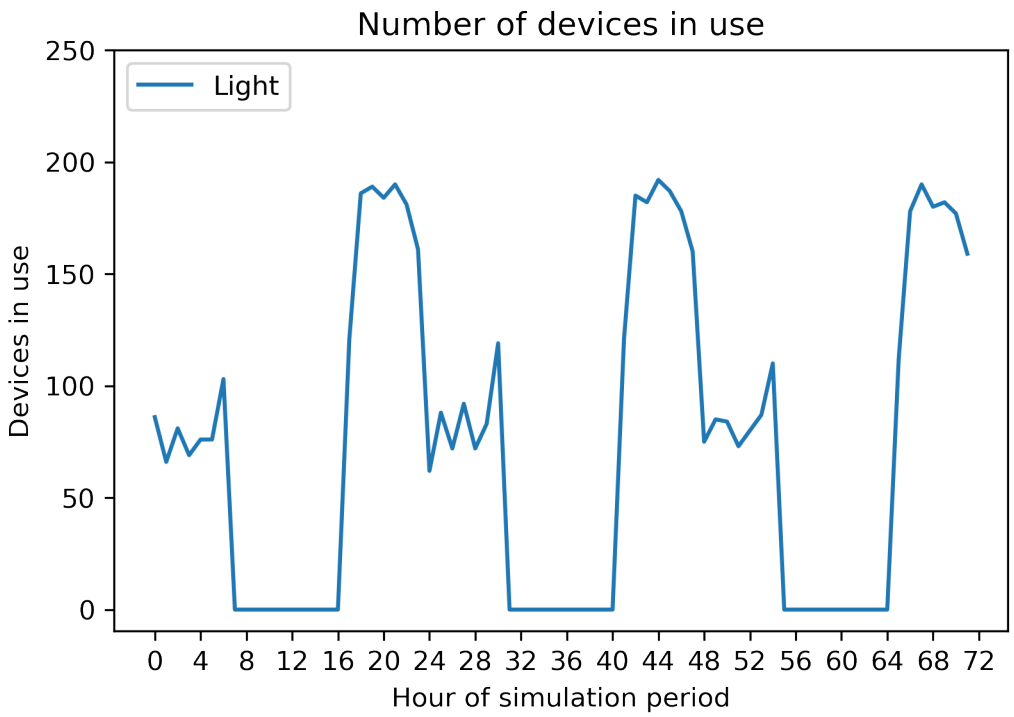
Now that we have both the number of devices in the community on a given day, and the daily utilisation profile of each device, we can calculate the number of devices that are in use at any given time. CLOVER does this automatically for each type of device by using a binomial random number generator: the number of trials is the number of devices, and the probability is the utilisation value for that hour. To do this, **run the following function in the console**:

```
Load().devices_in_use_hourly()
```

This function will take a while to complete as it runs through each type of device and randomly assigns the number in use over the full 20 year period, and needs to be performed once only. The outputs are saved as CSV files in the *Devices in use* folder in the *Load* folder, using the title format `[device]_in_use.csv`, so that they can be used as consistent inputs when used by CLOVER. Note that rerunning this function will create a new profile which will differ every time because of the random statistics used to generate it: if you need to rerun this function but want to keep your previous files, save a copy of them elsewhere otherwise they will be overwritten.

Let's consider the `light` device over the first three days of its usage:

[24]:
```
light_in_use = pd.read_csv("/Users/prs09/Documents/CLOVER/Locations/Bahraich/
↪Load/Devices in use/light_in_use.csv",index_col=0)
light_in_use = light_in_use[0:72]

plt.plot(range(72),light_in_use,label='Light')
plt.legend(loc='upper left')
plt.xticks(range(0,73,4))
plt.yticks(range(0,251,50))
plt.xlabel('Hour of simulation period')
plt.ylabel('Devices in use')
plt.title('Number of devices in use')
plt.show()
```



The number of `light` devices in use in a given hour changes throughout the day: as expected from the utilisation profile, some are in use overnight, none are in use during the day, and most are in use in the evenings. Comparing between days, the number of devices in use in a certain hour also varies.

4.2.4 Calculating the load profile of each device

Now that we know the number of each device in use at any given time, and the power rating of each from the *Device* input CSV, it is straightforward to get the load profile of each device. **Run the following function in the terminal:**

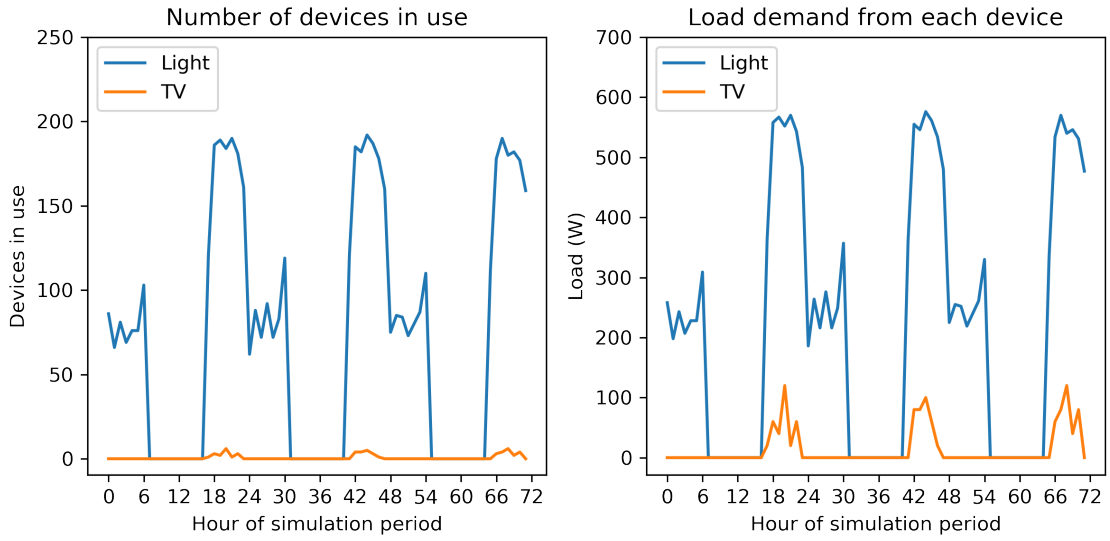
```
Load().device_load_hourly()
```

This function uses the `[device]_in_use.csv` file and the power rating to save a new CSV file in the *Device Load* folder in the *Load* folder using the title format `[device]_load.csv`.

Let's now compare two devices, `light` and `tv`, to see how the number in use and load demand vary over the first three days:

[25]:
```
tv_in_use = pd.read_csv("/Users/prs09/Documents/CLOVER/Locations/Bahraich/Load/
 ↪Devices in use/tv_in_use.csv",index_col=0)
tv_in_use = tv_in_use[0:72]

fig, ax = plt.subplots(1,2,figsize=(8,4))
```

```
ax[0].plot(range(72),light_in_use,label='Light')
ax[0].plot(range(72),tv_in_use,label='TV')
ax[0].legend(loc='upper left')
ax[0].set(xticks = (range(0,73,6)),yticks = range(0,251,50),
          xlabel = 'Hour of simulation period',
          ylabel = 'Devices in use',
          title = 'Number of devices in use')

light_load = pd.read_csv("/Users/prs09/Documents/CLOVER/Locations/Bahraich/Load/
 ↪Device load/light_load.csv",index_col=0)
light_load = light_load[0:72]
tv_load = pd.read_csv("/Users/prs09/Documents/CLOVER/Locations/Bahraich/Load/
 ↪Device load/tv_load.csv",index_col=0)
tv_load = tv_load[0:72]

ax[1].plot(range(72),light_load,label='Light')
ax[1].plot(range(72),tv_load,label='TV')
ax[1].legend(loc='upper left')
ax[1].set(xticks = (range(0,73,6)),yticks = range(0,701,100),
          xlabel = 'Hour of simulation period',
          ylabel = 'Load (W)',
          title = 'Load demand from each device')

plt.tight_layout()
plt.show()
```



Comparing the two devices, the usage of `tv` devices is limited to the evening periods only and very few are ever in use, owing to the low average ownership. When we compare the load demanded by each device, `tv` has a more significant effect, albeit still relatively small, as the relative power

rating is higher than that of `light`.

4.3 Calculating the total load demand of the community

Knowing the electricity demand of each appliance allows us to combine these to give the total electricity demand of the entire community over the duration of the investigation. Once again, CLOVER calculates this automatically when you **run the following function in the console:**

```
Load().total_load_hourly()
```

This function adds together all of the devices listed in the *Device* input file that have `Available = Y` and creates a new CSV in the *Device load* folder in the *Load* folder called `total_load.csv`. This also uses another of the input parameters from before, `Type`, to split the load into three categories: `Domestic` for households, `Commercial` for businesses and enterprises, and `Public` for community-level electricity uses.

With this, we now have the necessary load data to input into our simulations.

4.4 Troubleshooting

Most of the processes for generating load profile are automated but there are ample opportunities for simple mistakes when inputting the data which will cause errors. Solving the issues is normally simple but finding where they appear can be much harder, so if they come up:

- Check that you have used consistent spelling and capitalisation for your devices throughout as these are case sensitive, e.g. `radio` and `Radio` will be treated as two completely different devices
- Check that your file names correspond to the correct devices and are in the correct formats, e.g. `radio` has a utilisation profile named `radio_times`
- Ensure that your input variables are in the correct format, for example `Type` is either `Domestic`, `Commercial` or `Public`
- Ensure that your utilisation profiles are the correct size and format, and have the correct naming convention including the `.csv` suffix
- Ensure that your device power is input in Watts (W), not kilowatts (kW)
- Ensure that you have completed each of the steps in the correct order

4.5 Extension and visualisation

4.5.1 Using your own load profile

CLOVER gives users the functionality to construct their own load profiles from the bottom up, but sometimes you already have a load profile which you want to use directly without trying to recreate it using the *Load* module. This is straightforward to do, but your own load profile must be in the same format as the one generated by the *Load* module. To do this:

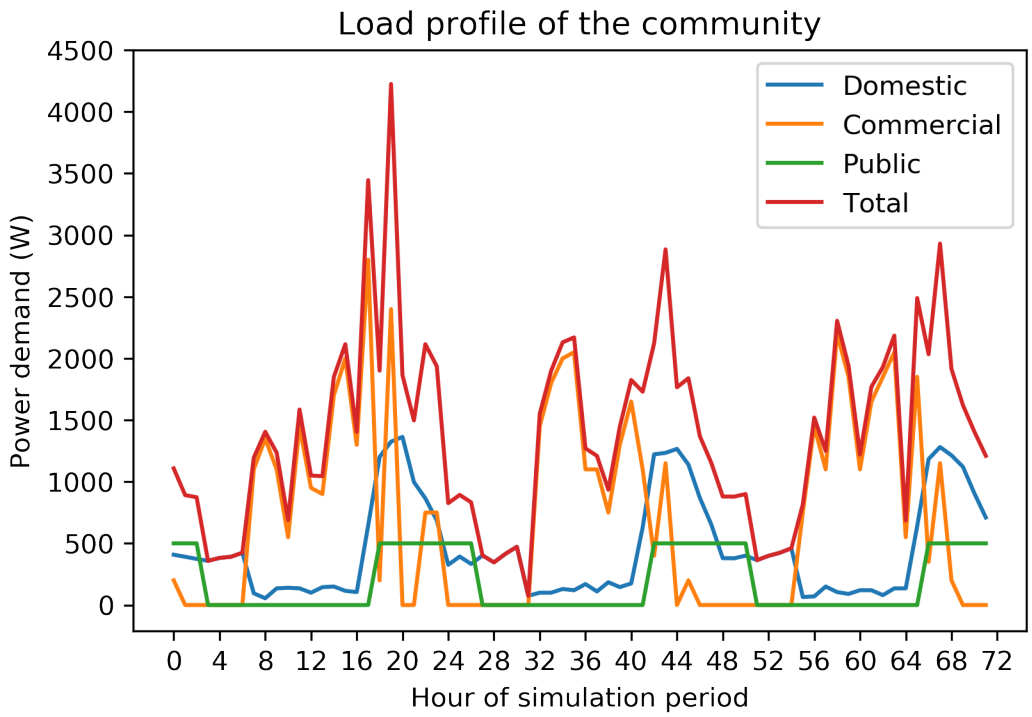
- Ensure that your load profile is in the correct format: at an hourly resolution for the duration of your simulation period, measured in Watts (not kilowatts), and with the `Domestic`, `Commercial` and `Public` headings
- Name your profile `total_load.csv` and copy it into the *Device load* folder in the *Load* folder of your location
- Run the function `Load().get_yearly_load_statistics('total_load.csv')` to get the yearly load statistics
- Copy `kerosene_load.csv` into the same folder (the values can be changed to zero, or the outputs ignored later, as necessary)

If you do not have `Domestic`, `Commercial` and `Public` loads, or they are combined into one category, it is most straightforward to include them all together in a single headed column (e.g. `Domestic`). The `kerosene_load.csv` needs to be included for compatibility with the later functionality, but the outputs relating to this can be ignored or the values in that profile can be set to zero for completeness. The `Load().get_yearly_load_statistics('total_load.csv')` is used later in the simulation process for sizing equipment (such as the inverters) so must be updated from the default case study values provided.

4.5.2 Comparing domestic, commercial and public electricity demand

Let's take a look at how the three demand types compare over the first three days of the simulation period:

[26]:
```
total_demand = pd.read_csv("/Users/prs09/Documents/CLOVER/Locations/Bahraich/
 ↪Load/Device load/total_load.csv",index_col=0)
total_demand = total_demand[0:72]

plt.plot(range(72),total_demand['Domestic'],label='Domestic')
plt.plot(range(72),total_demand['Commercial'],label='Commercial')
plt.plot(range(72),total_demand['Public'],label='Public')
plt.plot(range(72),np.sum(total_demand,axis=1),label='Total')
plt.legend(loc='upper right')
plt.xticks(range(0,73,4))
plt.yticks(range(0,4501,500))
plt.xlabel('Hour of simulation period')
plt.ylabel('Power demand (W)')
plt.title('Load profile of the community')
plt.show()
```



This allows us to compare the different types of demands in the community and how they vary throughout the day:

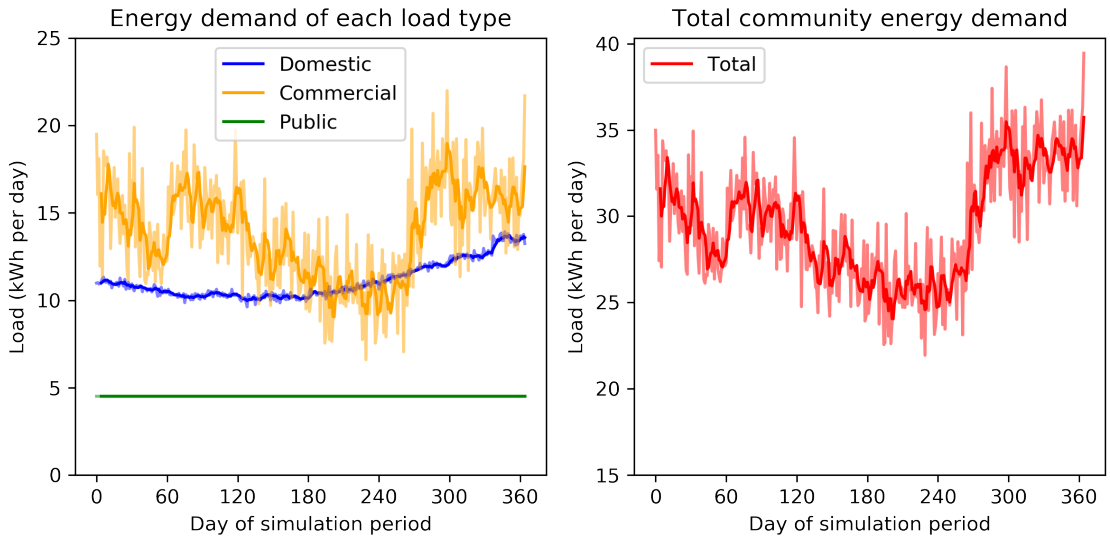
- The `Domestic` demand is highest in the evening due to lighting and entertainment (such as TV) loads
- The `Commercial` demand peaks in the day and has high variability caused by a smaller number of higher-power devices
- The `Public` demand, composed here as `streetlight` only, has a consistent load profile as it is modelled to operate on a timed basis

4.5.3 Varying demand throughout the year

We can also see how these demands vary over the year by reformatting the data into daily demands. For visibility let's look at a five-day rolling average, with the raw data plotted behind it:

[27]:
```
total_demand = pd.read_csv("/Users/prs09/Documents/CLOVER/Locations/Bahraich/
 ↪Load/Device load/total_load.csv",index_col=0)
total_demand = total_demand[0:8760]

domestic_demand = 0.001 * np.sum(np.reshape(total_demand['Domestic'].
 ↪values,(365,24)),axis=1)
```

```
commercial_demand = 0.001 * np.sum(np.reshape(total_demand['Commercial'].
 ↪values,(365,24)),axis=1)
public_demand = 0.001 * np.sum(np.reshape(total_demand['Public'].
 ↪values,(365,24)),axis=1)
total_demand = 0.001 * np.sum(np.reshape(np.sum(total_demand,axis=1).
 ↪values,(365,24)),axis=1)

fig, ax = plt.subplots(1,2,figsize=(8,4))

ax[0].plot(range(365),pd.DataFrame(domestic_demand).rolling(5).
 ↪mean(),label='Domestic',color='blue')
ax[0].plot(range(365),pd.DataFrame(commercial_demand).rolling(5).
 ↪mean(),label='Commercial',color='orange')
ax[0].plot(range(365),pd.DataFrame(public_demand).rolling(5).
 ↪mean(),label='Public',color='green')
ax[0].plot(range(365),domestic_demand,alpha=0.5,color='blue')
ax[0].plot(range(365),commercial_demand,alpha=0.5,color='orange')
ax[0].plot(range(365),public_demand,alpha=0.5,color='green')
ax[0].legend(loc='best')
ax[0].set(xticks = (range(0,366,60)),yticks = range(0,26,5),
          xlabel = 'Day of simulation period',
          ylabel = 'Load (kWh per day)',
          title = 'Energy demand of each load type')

ax[1].plot(range(365),pd.DataFrame(total_demand).rolling(5).
 ↪mean(),label='Total',color='red')
ax[1].plot(range(365),total_demand,alpha=0.5,color='red')
ax[1].legend(loc='best')
ax[1].set(xticks = (range(0,366,60)),yticks = range(15,41,5),
          xlabel = 'Day of simulation period',
          ylabel = 'Load (kWh per day)',
          title = 'Total community energy demand')

plt.tight_layout()
plt.show()
```



Note that the axes of the two subplots use the same scale but the axis of the right plot, showing the total community demand, is shifted upwards. `Commercial` demand, as before, displays the largest amount of variation both between days and over the course of the year, in this case because of varying agricultural demand. The `Domestic` demand has relatively little day-to-day variation but there is a noticeable decrease in the summer as lighting demand decreases. The growth in `Domestic` device ownership also results in the growth of the daily load demand (comparing January and December), almost exceeding the `Commercial` demand. Finally the `Public` demand remains the same throughout the year, as expected.

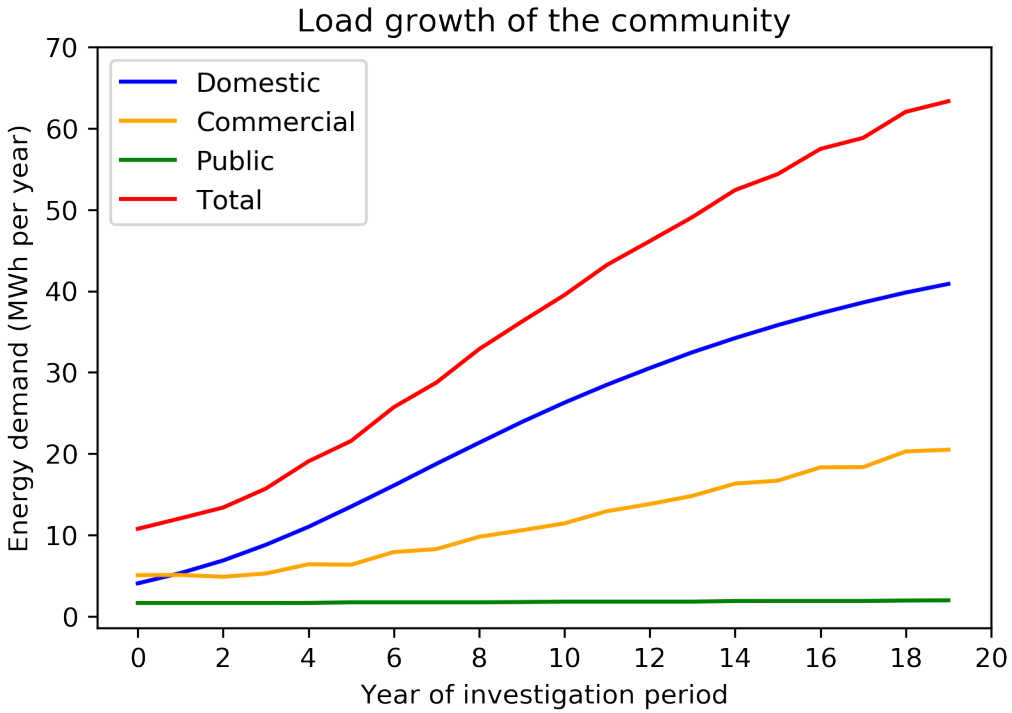
The total community demand is shown on the left. Its structure is mostly governed by the variation in the `Commercial` demand, but the effect of the `Domestic` demand can be seen when comparing the upwards shift in load comparing the start and end of this year-long profile.

4.5.4 Growing demand over the investigation period

We can see how load grows in the community across the investigation period, for example to replicate how appliance ownership might grow over time as a community develops economically. Let's reformat the load data into yearly totals and compare them across the 20-year period:

[28]:
```
total_demand = pd.read_csv("/Users/prs09/Documents/CLOVER/Locations/Bahraich/
 ↪Load/Device load/total_load.csv",index_col=0)

domestic_demand = 0.000001 * np.sum(np.reshape(total_demand['Domestic'].
 ↪values,(20,8760)),axis=1)
commercial_demand = 0.000001 * np.sum(np.reshape(total_demand['Commercial'].
 ↪values,(20,8760)),axis=1)
public_demand = 0.000001 * np.sum(np.reshape(total_demand['Public'].
 ↪values,(20,8760)),axis=1)
```

```
total_demand = 0.000001 * np.sum(np.reshape(np.sum(total_demand,axis=1).
 ↪values,(20,8760)),axis=1)

plt.plot(range(20),domestic_demand,label='Domestic',color='blue')
plt.plot(range(20),commercial_demand,label='Commercial',color='orange')
plt.plot(range(20),public_demand,label='Public',color='green')
plt.plot(range(20),total_demand,label='Total',color='red')

plt.legend(loc='upper left')
plt.xticks(range(0,21,2))
plt.yticks(range(0,71,10))
plt.xlabel('Year of investigation period')
plt.ylabel('Energy demand (MWh per year)')
plt.title('Load growth of the community')
plt.show()
```



We can see here that the `Domestic` and `Commercial` loads both increase over time, but the `Domestic` increases much faster than `Commercial` and therefore is the greatest contributor to the total load soon after the first year (as we saw in Section 4.5.3). The `Public` demand meanwhile remains the same throughout.

5 Energy system simulation

5.1 Preparation

Now that we have completed both the Section 3 and Section 4 inputs for our investigation we are almost ready to simulate an energy system. This will allow us to model the technical performance of an energy system of a given size and its ability to meet the load demanded by the community. At this stage we consider only the technical performance, rather than the financial or environmental considerations, which will come later in Section 6.

Before we can simulate a system we must first provide inputs for its technical performance and the conditions of the scenario under which we want it to operate.

5.1.1 Energy system inputs

The inputs for the technical performance of the system are included in the `Energy system inputs` file, which is located in the *Simulation* folder of your location folder.

Let's look at the inputs included for the Bahraich case study:

[29]:
```
energy_system_inputs = pd.read_csv("/Users/prs09/Documents/CLOVER/Locations/
 ↪Bahraich/Simulation/Energy system inputs.csv",header=None)
energy_system_inputs.head(len(energy_system_inputs))
```

[29]:

	0	1	2
0	Battery maximum charge	0.900	State of charge (0.0-1.0)
1	Battery minimum charge	0.400	State of charge (0.0-1.0)
2	Battery leakage	0.004	Fractional leakage per hour
3	Battery conversion in	0.950	Conversion efficiency (0.0-1.0)
4	Battery conversion out	0.950	Conversion efficiency (0.0-1.0)
5	Battery cycle lifetime	1500.000	
6	Battery lifetime loss	0.400	
7	Battery C rate discharging	0.330	
8	Battery C rate charging	0.330	
9	Transmission efficiency DC	0.950	
10	Transmission efficiency AC	0.950	
11	DC to AC conversion	0.950	
12	DC to DC conversion	0.950	
13	AC to DC conversion	0.800	
14	AC to AC conversion	0.980	

```
5   Expected number of cycles over lifetime
6   Fractional loss over lifetime (0.0-1.0)
7                            Discharge rate
8                               Charge rate
9     Efficiency of DC distribution network
10    Efficiency of AC distribution network
11          Conversion efficiency (0.0-1.0)
12          Conversion efficiency (0.0-1.0)
13          Conversion efficiency (0.0-1.0)
14          Conversion efficiency (0.0-1.0)
```

These variables control how the electricity system performs, in particular the performance of the battery storage and the conversion efficiencies in the system. The table below describes in more detail what each one means:

Variable	Explanation
`Battery maximum charge`	Maximum permitted state of charge of the battery
`Battery minimum charge`	Minimum permitted state of charge of the battery
`Battery leakage`	Fraction of the energy stored in the battery lost per hour
`Battery conversion in`	Conversion efficiency of energy entering the battery
`Battery conversion out`	Conversion efficiency of energy leaving the battery
`Battery cycle lifetime`	Number of charging cycles expected from the battery over its lifetime
`Battery C rate discharging`	C-rate of the battery whilst providing energy
`Battery C rate charging`	C-rate of the battery whilst receiving energy
`Transmission efficiency DC`	Transmission efficiency of a DC distribution network
`Transmission efficiency AC`	Transmission efficiency of an AC distribution network
`DC to AC conversion`	Conversion efficiency from DC power to an AC distribution network
`DC to DC conversion`	Conversion efficiency from DC power to an DC distribution network
`AC to DC conversion`	Conversion efficiency from AC power to an DC distribution network
`AC to AC conversion`	Conversion efficiency from AC power to an AC distribution network

The variables `Battery maximum charge` and `Battery minimum charge` refer to the maximum and minimum permitted states of charge of the battery: in this case the battery is allowed to cycle between 90% and 40% of its total capacity, resulting in a depth of discharge (DOD) of 50%, and

meaning that 50% of the total installed battery capacity is actually usable by the system. `Battery leakage` is the fraction of energy that leaks out of the battery every hour, in this case `0.004` or 0.4% of the energy presently stored in it per hour. `Battery conversion in` and `Battery conversion out` are the conversion efficiencies of energy being supplied to and from the battery respectively; when multiplied together these give the battery round-trip efficiency.

`Battery cycle lifetime` refers to the number of charging and discharging cycles that the battery can be expected to perform over its lifetime, with the lifetime defined to be over when the battery has degraded by `Battery lifetime loss`; in this case `Battery lifetime loss = 0.2` (as is typical for this definition) and so the lifetime is over when the battery provides just 80% of its original capacity. The battery degradation is calculated by multiplying the lifetime loss by the energy throughput of the battery (at a given point in time) and then dividing by the expected cumulative energy throughput over the lifetime of the battery (the cycle lifetime multiplied by the depth of discharge and total capacity). This simplified method does not account for the effects of temperature or reduced cycling which may affect the lifetime of a battery in practice.

Finally for the battery parameters, `Battery C rate discharging` and `Battery C rate charging` are the C-rates for discharging and charging the batteries, measured as the maximum permitted fraction of the battery capacity that can be stored or supplied in one hour. These battery parameters can be taken from a datasheet provided by a battery manufacturer or used as indicative values in more general investigations. They are also agnostic to the type of battery technology being investigated, for example lead acid or lithium ion batteries. Some of these parameters will be dependent on one another: for example, a given battery being used with a higher DOD will likely have a lower cycle lifetime. These relationships are often available on battery datasheets (for example as performance curves) but need to be input manually and individually here. Similarly, higher C-rates will also likely result in lower cycle lifetimes.

Let's take a look at some of the variables:

[30]:
```
max_charge = energy_system_inputs.iloc[0][1]
min_charge = energy_system_inputs.iloc[1][1]
DOD = max_charge - min_charge
print('Maximum state of charge: ' + str((int(100 * max_charge))) + '%')
print('Minimum state of charge: ' + str((int(100 * min_charge))) + '%')
print('Depth of discharge: ' + str(int(100 * DOD)) + '%\n')

battery_conversion_in = energy_system_inputs.iloc[3][1]
battery_conversion_out = energy_system_inputs.iloc[4][1]
round_trip_efficiency = battery_conversion_in * battery_conversion_out
print('Battery input efficiency: ' + str((int(100 * battery_conversion_in))) +␣
 ↪'%')
print('Battery output efficiency: ' + str((int(100 * battery_conversion_out))) +␣
 ↪'%')
print('Round trip efficiency: ' + str(int(100 * round_trip_efficiency)) + '%')
```

```
Maximum state of charge: 90%
Minimum state of charge: 40%
Depth of discharge: 50%
```

```
Battery input efficiency: 95%
Battery output efficiency: 95%
Round trip efficiency: 90%
```

The next two variables, `Transmission efficiency AC` and `Transmission efficiency DC`, describe the efficiency of the power distribution network being used to transmit power from the generation and storage source to the consumers. This can be AC (alternating current, generally better for high-power applications and long-range transmission) or DC (direct current, generally better for low-power applications and short-range transmission). Only one of these will be used at a time but both should be completed, for example using a dummy value if only one is ever to be investigated. Finally, `DC to AC conversion` (for example) gives the conversion efficiency of DC power sources, such as solar or batteries, to an AC distribution network. These are the efficiencies of inverters, rectifiers and voltage converters that would be used in the system; as before, these should all be included for completeness but dummy values (or the defaults) could be used if only one distribution network is being considered.

Complete the `Energy system inputs` CSV with the technical performance parameters for your investigation.

5.1.2 Scenario inputs

The inputs which describe the situation we are investigating are provided in the *Scenario inputs* CSV file in the *Scenario* folder of your location folder. These describe parameters such as the types of technologies that are being used in the system and the loads that are being met. Let's take a look at the default inputs for Bahraich:

[31]:
```
scenario_inputs = pd.read_csv("/Users/prs09/Documents/CLOVER/Locations/Bahraich/
 ↪Scenario/Scenario inputs.csv",header=None)
scenario_inputs.head(len(scenario_inputs))
```

[31]:
```
                              0         1  \
0                            PV         Y
1                       Battery         Y
2                Diesel backup         Y
3      Diesel backup threshold       0.1
4                          Grid         Y
5                     Grid type  bahraich
6   Prioritise self generation         Y
7                      Domestic         Y
8                    Commercial         Y
9                        Public         Y
10        Distribution network        DC

                                      2
0                                 (Y/N)
1                                 (Y/N)
2                                 (Y/N)
3   Maximum acceptible blackouts (0.0-1.0)
```

```
4                                    (Y/N)
5                               Grid profile
6                                    (Y/N)
7                                    (Y/N)
8                                    (Y/N)
9                                    (Y/N)
10          DC or AC distribution network
```

Many of these may be straightforward, but the table below describes them explicitly.

Variable	Explanation
`PV`	Whether solar PV is available (`Y`) or not (`N`)
`Battery`	Whether battery storage is available (`Y`) or not (`N`)
`Diesel backup`	Whether a diesel generator backup is available (`Y`) or not (`N`)
`Diesel backup threshold`	The blackout threshold which the diesel generator is used to achieve
`Grid`	Whether the national grid is available (`Y`) or not (`N`)
`PV`	Whether solar PV is available (`Y`) or not (`N`)
`Prioritise self generation`	Whether to prioritise local generation (`Y`) or energy from the grid (`N`)
`Domestic`	Whether `Domestic` loads are included in the load profile (`Y`) or not (`N`)
`Commercial`	Whether `Commercial` loads are included in the load profile (`Y`) or not (`N`)
`Public`	Whether `Public` loads are included in the load profile (`Y`) or not (`N`)
`Distribution network`	Whether an `AC` or `DC` distribtion network is used to transmit electricity

The three of the variables, `PV`, `Battery` and `Grid`, are present for future-proofing and have no effect at present: solar and battery storage must be considered in simulations for now, although (as we will see in Section 5.2) they can have capacities of zero which mean they are not actually included. Similarly `Grid type` describes the grid availability profile to be used from the *Grid* module, which can be similarly switched off by selecting a profile with no availability (e.g. `Grid type = none`).

The `Diesel backup` variable is active and controls whether a diesel generator can be used to supply additional power during times of blackouts. Periods of blackouts will be described in more detail later, but for now the `blackouts` parameter can be described as the fraction of time that insufficient energy is available in the system to meet the loads. If a system of a specified solar and storage capacity, operating with a given grid availability, has a `blackouts` parameter greater than `Diesel backup threshold` then the diesel generator is used retroactively to top up hours where blackouts occur, up to the point at which the system `blackouts` and `Diesel backup threshold` are equal. For example, if a system had `blackouts = 0.17` and (as in default values) `Diesel backup threshold = 0.1`, then the diesel generator would be used to supply power in 7% (`0.07`) of the

hours to make `blackouts = 0.10` after its implementation.

`Prioritise self generation` describes whether the system will use its own locally-generated energy from solar first before drawing power from the grid if available and then storage (`Y`), or whether it will take power from the grid first if available and then from solar and then storage (`N`). In either case, it may be that either locally generated or grid power is unavailable and therefore this should be thought of as a prioritisation of sources rather than a backup. In both cases the diesel backup is considered after this prioritisation occurs.

`Domestic`, `Commercial` and `Public` refer to whether these demand types are to be included in the load profile used in the investigation. Finally the `Distribution network` defines whether an `AC` or `DC` transmission network is used to distribute electricity from the sources to the loads, which will affect the conversion efficiencies used as inputs in the previous section.

Complete the `Scenario inputs` CSV with the details of the situation of your investigation.

5.2 Performing a simulation of an energy system

5.2.1 Inputs

We are now able to perform a simulation of an energy system using the *Energy_System* module. This relies on all of the information we have input and generated previously in Section 3 and Section 4, and the earlier parts of this section. This will let us investigate the technological performance of a system with a specified solar and battery capacity, operating under the conditions we defined earlier.

To perform a simulation we must first **run the *Energy_System* script (using the green arrow in the Spyder console)**, which we do here using the following:

[32]:
```
sys.path.insert(0, '/Users/prs09/Documents/CLOVER/Scripts/Simulation scripts/')
from Energy_System import Energy_System
```

To simulate an energy system we need to specify four further parameters; these are taken as inputs for convenience when investigating many system sizes, for example during optimisations which we will explore further later. These are:

Variable	Explanation
`start_year`	Starting year of the simulation
`end_year`	End year of the simualation
`PV_size`	Installed solar capacity in kWp
`storage_size`	Installed battery storage capacity in kWh

These tell the function running the simulation both the time period to consider and the capacity of the system that is being investigated. The parameters `start_year` and `end_year` are defined by the first day of their respective years and (as with the rest of Python) start from `0`. For example, running a simulation for only the first year of a 20-year timeline would require `start_year = 0` and `end_year = 1`, i.e. running from the first day of Year 0 up to (but not including) the first day of Year 1. These inputs **must** be integers.

The parameters `PV_size` and `storage_size` refer to the installed capacities of the solar and battery storage components in their functional units of kWp and kWh respectively. These inputs can be any number, including decimals (for example as a multiple of a given solar panel size) and zero if they are not to be included, as we saw earlier in Section 5.1.2.

The `Energy_System().simulation(...)` function includes default values for each of these parameters which are used if the user does not specify any of their input values. In our example, we will simulate a system over the first year of its lifetime (`start_year = 0, end_year = 1`) and with `PV_size = 5` kWp and `storage_size = 20` kWh.

5.2.2 Running a simulation

To run a simulation we **run the following function in the console** with our choice of input variables, saving the output as a variable called `example_simulation` so we can look at the outputs in more detail:

[33]:
```
example_simulation = Energy_System().simulation(start_year=0, end_year=1,␣
↪PV_size=5, storage_size=20)
```

```
Time taken for simulation: 0.45 seconds per year
```

When we run this function we get an output, which we called `example_simulation`, which is composed of two further outputs: one describing the technical performance of the system, and one describing the input parameters that we gave to the function. If we did not save the output of this function as a variable then the results would have been printed to the screen, but not available to use later. We also get an estimate of the time taken to perform each year of the simulation: running the entire function likely looking much longer than this, but this value can be useful in identifying potential errors if the value is much higher for some simulations rather than others.

When this function is called it automatically takes into account all of the earlier input data and operating conditions to simulate the system over the defined time period - making the function itself very straightforward to use.

5.2.3 Simulation outputs

The important parts of this function are its two outputs which tell us how the system has performed over the simulated time period. Let's take a look at the first component by defining a new variable called `example_simulation_performance`:

[34]:
```
example_simulation_performance = example_simulation[0]
example_simulation_performance.head(24).round(3)
```

[34]:

	Load energy (kWh)	Total energy used (kWh)	Unmet energy (kWh)	Blackouts	\
0	1.166	1.166	0.0	0.0	
1	0.938	0.938	0.0	0.0	
2	0.920	0.968	0.0	0.0	
3	0.377	0.377	0.0	0.0	

4	0.402	0.423	0.0	0.0
5	0.412	0.412	0.0	0.0
6	0.446	0.470	0.0	0.0
7	1.258	1.322	0.0	0.0
8	1.479	1.513	0.0	0.0
9	1.300	1.300	0.0	0.0
10	0.726	0.726	0.0	0.0
11	1.668	1.668	0.0	0.0
12	1.105	1.105	0.0	0.0
13	1.100	1.100	0.0	0.0
14	1.947	1.947	0.0	0.0
15	2.226	2.244	0.0	0.0
16	1.479	1.511	0.0	0.0
17	3.626	3.626	0.0	0.0
18	2.001	2.106	0.0	0.0
19	4.447	4.447	0.0	0.0
20	1.961	2.064	0.0	0.0
21	1.576	1.576	0.0	0.0
22	2.226	2.226	0.0	0.0
23	2.038	2.038	0.0	0.0

	Renewables energy used (kWh)	Storage energy supplied (kWh)	\
0	0.000	1.166	
1	0.000	0.000	
2	0.000	0.968	
3	0.000	0.000	
4	0.000	0.423	
5	0.000	0.000	
6	0.000	0.470	
7	0.041	1.281	
8	0.830	0.683	
9	1.300	0.000	
10	0.726	0.000	
11	1.668	0.000	
12	1.105	0.000	
13	1.100	0.000	
14	1.947	0.000	
15	1.882	0.363	
16	0.866	0.645	
17	0.077	0.000	
18	0.000	2.106	
19	0.000	3.473	
20	0.000	2.064	
21	0.000	0.000	
22	0.000	0.898	
23	0.000	0.000	

	Grid energy (kWh)	Diesel energy (kWh)	Diesel times	\
0	0.000	0.000	0.0	
1	0.938	0.000	0.0	
2	0.000	0.000	0.0	
3	0.377	0.000	0.0	
4	0.000	0.000	0.0	
5	0.412	0.000	0.0	
6	0.000	0.000	0.0	
7	0.000	0.000	0.0	
8	0.000	0.000	0.0	
9	0.000	0.000	0.0	
10	0.000	0.000	0.0	
11	0.000	0.000	0.0	
12	0.000	0.000	0.0	
13	0.000	0.000	0.0	
14	0.000	0.000	0.0	
15	0.000	0.000	0.0	
16	0.000	0.000	0.0	
17	3.550	0.000	0.0	
18	0.000	0.000	0.0	
19	0.000	0.974	1.0	
20	0.000	0.000	0.0	
21	1.576	0.000	0.0	
22	0.000	1.329	1.0	
23	0.000	2.038	1.0	

	Diesel fuel usage (l)	Storage profile (kWh)	\
0	0.000	-1.166	
1	0.000	0.000	
2	0.000	-0.920	
3	0.000	0.000	
4	0.000	-0.402	
5	0.000	0.000	
6	0.000	-0.446	
7	0.000	-1.217	
8	0.000	-0.649	
9	0.000	0.591	
10	0.000	1.941	
11	0.000	1.432	
12	0.000	2.202	
13	0.000	2.045	
14	0.000	0.733	
15	0.000	-0.345	
16	0.000	-0.613	
17	0.000	0.000	
18	0.000	-2.001	
19	0.560	-4.447	

20	0.000	-1.961
21	0.000	0.000
22	0.560	-2.226
23	0.815	-2.038

	Renewables energy supplied (kWh)	Hourly storage (kWh)	\
0	0.000	16.834	
1	0.000	16.766	
2	0.000	15.731	
3	0.000	15.668	
4	0.000	15.182	
5	0.000	15.121	
6	0.000	14.591	
7	0.041	13.251	
8	0.830	12.515	
9	1.891	13.027	
10	2.667	14.818	
11	3.100	16.119	
12	3.308	17.999	
13	3.145	17.999	
14	2.680	17.999	
15	1.882	17.564	
16	0.866	16.849	
17	0.077	16.782	
18	0.000	14.608	
19	0.000	11.076	
20	0.000	8.968	
21	0.000	8.932	
22	0.000	7.999	
23	0.000	7.998	

	Dumped energy (kWh)	Battery health	Households	Kerosene lamps	\
0	0.000	1.0	100	0.0	
1	0.000	1.0	100	0.0	
2	0.000	1.0	100	0.0	
3	0.000	1.0	100	0.0	
4	0.000	1.0	100	0.0	
5	0.000	1.0	100	0.0	
6	0.000	1.0	100	0.0	
7	0.000	1.0	100	0.0	
8	0.000	1.0	100	0.0	
9	0.000	1.0	100	0.0	
10	0.000	1.0	100	0.0	
11	0.000	1.0	100	0.0	
12	0.148	1.0	100	0.0	
13	1.871	1.0	100	0.0	
14	0.624	1.0	100	0.0	

15	0.000	1.0	100	0.0
16	0.000	1.0	100	0.0
17	0.000	1.0	100	0.0
18	0.000	1.0	100	0.0
19	0.000	1.0	100	0.0
20	0.000	1.0	100	0.0
21	0.000	1.0	100	0.0
22	0.000	1.0	100	0.0
23	0.000	1.0	100	0.0

	Kerosene mitigation
0	75.0
1	103.0
2	81.0
3	91.0
4	74.0
5	73.0
6	70.0
7	0.0
8	0.0
9	0.0
10	0.0
11	0.0
12	0.0
13	0.0
14	0.0
15	0.0
16	0.0
17	122.0
18	333.0
19	338.0
20	336.0
21	277.0
22	132.0
23	108.0

This component gives the performance of the system at an hourly resolution, with the first 24 hours of the simulation shown here and rounded to three decimal places for convenience. They are defined in the table below:

Variable	Explanation
`Load energy (kWh)`	Load energy demanded by the community
`Total energy used (kWh)`	Total energy used by the community
`Unmet energy (kWh)`	Energy that would have been needed to meet energy demand
`Blackouts`	Whether there was a blackout period (`1`) or not (`0`)

Variable	Explanation
`Renewables energy used (kWh)`	Renewable energy used directly by the community
`Storage energy supplied (kWh)`	Energy supplied by battery storage
`Grid energy (kWh)`	Energy supplied by the grid network
`Diesel energy (kWh)`	Energy supplied by the diesel generator
`Diesel times`	Whether the diesel generator was on (1) or off (0)
`Diesel fuel usage (l)`	Litres of diesel fuel used
`Storage profile (kWh)`	Dummy profile of energy into (+) or out of (-) the battery
`Renewables energy supplied (kWh)`	Total renewable energy generation supplied to the system
`Hourly storage (kWh)`	Total energy currently stored in the battery
`Dumped energy (kWh)`	Energy dumped due to overgeneration when storage is full
`Battery health`	Measure of the relative health of the battery
`Households`	Number of households currently in the community
`Kerosene lamps`	Number of kerosene lamps used
`Kerosene mitigation`	Number of kerosene lamps mitigated through power availability

The majority of these variables describe the energy flows within the system, the sources that they come from and the amount of load energy that is being met. Others describe a binary characteristic of whether or not an hour experiences a blackout (defined as any shortfall in service availability during that hour) or if a diesel generator is being used, and others (such as the number of households, kerosene usage and mitigation, and storage profile) are used either in the computation of this function or later functions that rely on this output.

Let's now take a look at the other output of `Energy_System().simulation(...)`, which we will define as a new variable called `example_system_description`:

[35]:
```
example_simulation_description = example_simulation[1]
example_simulation_description.head()
```

[35]:
```
                Start year  End year  Initial PV size  Initial storage size  \
System details         0.0       1.0              5.0                  20.0

                Final PV size  Final storage size  Diesel capacity
System details           4.95           19.229097                4
```

This variable provides details of the system that was simulated, including several of the input variables such as the time period being investigated and the solar and storage capacities we used. It also describes three new variables: `Final PV size` and `Final storage size` describe the relative capacities of the solar and battery components at the end of the simulation period after accounting for degradation, and `Diesel capacity` is the minimum diesel generator capacity (in

kW) necessary to supply power as a backup system.

The outputs of this variable are primarily used as inputs for later functions, particularly those that deal with optimisation as it is necessary to know the status of an earlier system when considering periodic improvements over time.

5.2.4 Saving simulation results and opening saved files

Saving simulation outputs as variables allows us to explore them in more detail but, once the session is closed, these variables are deleted and the data is lost - meaning that the same simulation would need to be performed again in order to investigate the same scenario. As the simulation function relies on data previously stored elsewhere, as long as the input conditions are unchanged then the same result will be generated, but this is not convenient in the long term.

CLOVER provides a function to save the output of simulations as CSV files, storing the data much more conveniently. To save an output (`simulation_name`) we need to have first stored it as a variable, and choose a filename (`filename`) to store it (note that the `filename` variable in this function must be a string). In our case `simulation_name = example_simulation_performance`, and we choose `file_name = 'my_saved_simulation'`. **To save the simulation results we run the function**:

```
Energy_System().save_simulation(simulation_name = example_simulation_performance,
filename = 'my_saved_simulation')
```

This function creates a new CSV file in the *Saved simulations* folder in the *Simulation* folder in your location folder titled `my_saved_simulation.csv`. If the `filename` variable is left blank, the title of the CSV will default to the time when the save operation was performed. **Be aware that running this function with a filename that already exists will overwrite the existing file**. Notice as well that we used `example_simulation_performance` as the variable to be saved, rather than the two-component output `example_simulation`: performing this function with the latter will result in an error.

To open a saved file, we use the name of the CSV file to open the correct result, for example:

```
opened_simulation = Energy_System().open_simulation(filename =
'my_saved_simulation')
```

This will open the `my_saved_simulation.csv` file and record the data as a new variable, `opened_simulation`, which will be in the same format as the original saved variable `example_simulation_performance`.

5.3 Troubleshooting

Most of the *Energy System* functionality is contained within the `Energy_System().simulation(...)` function and so potential issues are most likely to come either from how the module gathers data from other parts of CLOVER:

- Ensure that the `self.location` variable is correct in all of the modules that *Energy_System* imports

- Check that your `Scenario inputs` CSV is completed with the scenario you want to investigate, and any changes are saved in the CSV file before running another simulation
- Ensure that you use the correct `filename` when saving and opening previous simulations
- When running simulations, remember to save the output of `Energy_System().simulation(...)` as a variable

5.4 Extension and visualisation

5.4.1 Exploring the performance of the system

We can use the `example_simulation_performance` variable to investigate the performance of the system. Some variables make more sense to look at their average over the simulation period:

[36]:
```
example_simulation_performance_averages =␣
 ↪example_simulation_performance[['Blackouts','Diesel times']].mean().round(3)
print(example_simulation_performance_averages)
```

```
Blackouts       0.100
Diesel times    0.104
dtype: float64
```

Here we can see that the average for `Blackouts` is `0.100` meaning that power is unavailable for 10% of the time, or equivalently the system has power 90% of the time. We could have expected this from our earlier condition in the `Scenario inputs` CSV which set `Diesel backup threshold = 0.1`, forcing the diesel backup generator to be used to provide this level of reliability. In this case the average of `Diesel times` is `0.104`, meaning that the generator is switched on for 10.4% of the time in order to provide the desired level of reliability.

Other variables make more sense to look at their sum, so here we look at their performance over the year but then presented as a daily average:

[37]:
```
example_simulation_performance_sums = example_simulation_performance[[
    'Total energy used (kWh)','Unmet energy (kWh)','Renewables energy used␣
 ↪(kWh)','Storage energy supplied (kWh)',
    'Grid energy (kWh)','Diesel energy (kWh)','Renewables energy supplied␣
 ↪(kWh)','Dumped energy (kWh)'
    ]].sum()/365.0
print(example_simulation_performance_sums.round(3))
```

```
Total energy used (kWh)              30.465
Unmet energy (kWh)                    0.909
Renewables energy used (kWh)         11.940
Storage energy supplied (kWh)         7.920
Grid energy (kWh)                     7.239
Diesel energy (kWh)                   3.366
Renewables energy supplied (kWh)     22.656
```

```
Dumped energy (kWh)                    1.468
dtype: float64
```

These values show the average daily energy supply and usage in the system. Here we see that 30.5 kWh per day are consumed by the community, with a further 0.9 kWh going unmet on average. The supply is composed of renewable energy from our solar capacity directly (11.9 kWh) and from the battery storage (7.9 kWh), with the grid (7.2 kWh) and the backup diesel generator (3.4 kWh) also supplying energy. Our solar capacity generates an average of 45.3 kWh per day: 13.1 kWh is used directly, then the rest is stored in the batteries, an average of 1.5 kWh per day is dumped when the batteries are already full, and the remainder is lost owing to the transmission and conversion efficiencies in the system.

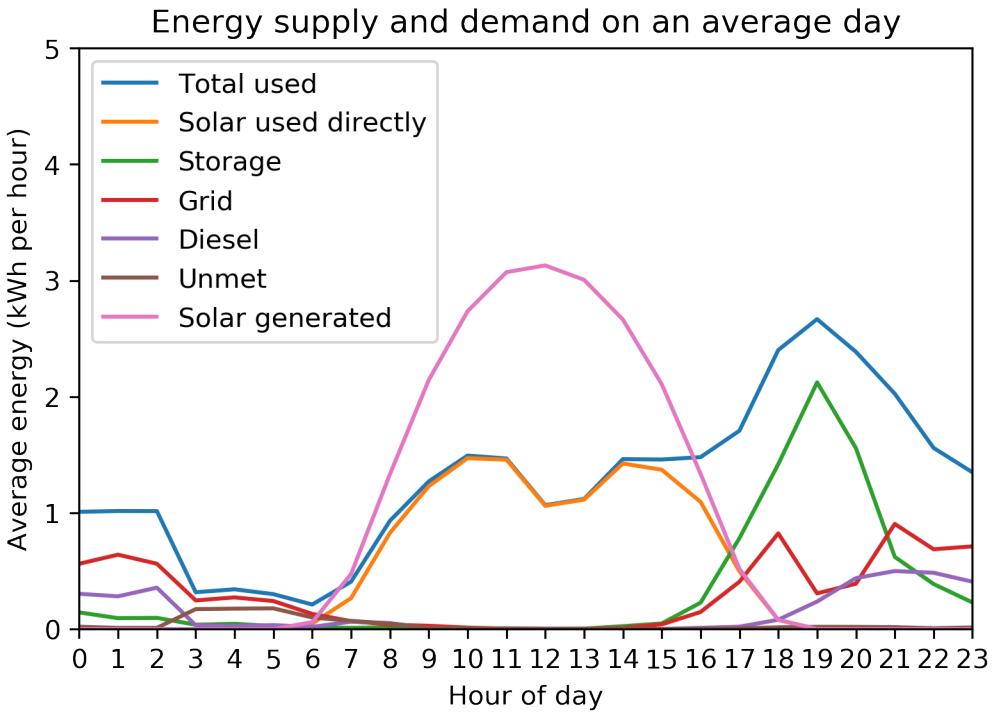
Adding up the renewables energy used, storage energy, grid energy and diesel energy gives us the total energy used, and when we also add the unmet energy this gives us the amount of energy required to meet the load demanded by the community. This combined value is slightly higher than the value for load energy from the *Load* module because the former accounts for the losses in the system needed to entirely satisfy the latter.

5.4.2 Electricity usage on an average day

We can use these outputs to visualise the energy flows in the system on an average day:

[38]:
```
total_used = np.mean(np.reshape(example_simulation_performance['Total energy␣
 ↪used (kWh)'].values,(365,24)),axis=0)
renewable_energy = np.mean(np.reshape(example_simulation_performance['Renewables␣
 ↪energy used (kWh)'].values,(365,24)),axis=0)
storage_energy = np.mean(np.reshape(example_simulation_performance['Storage␣
 ↪energy supplied (kWh)'].values,(365,24)),axis=0)
grid_energy = np.mean(np.reshape(example_simulation_performance['Grid energy␣
 ↪(kWh)'].values,(365,24)),axis=0)
diesel_energy = np.mean(np.reshape(example_simulation_performance['Diesel energy␣
 ↪(kWh)'].values,(365,24)),axis=0)
unmet_energy = np.mean(np.reshape(example_simulation_performance['Unmet energy␣
 ↪(kWh)'].values,(365,24)),axis=0)
renewables_supplied = np.mean(np.
 ↪reshape(example_simulation_performance['Renewables energy supplied (kWh)'].
 ↪values,(365,24)),axis=0)

plt.plot(total_used, label = 'Total used')
plt.plot(renewable_energy, label = 'Solar used directly')
plt.plot(storage_energy, label = 'Storage')
plt.plot(grid_energy, label = 'Grid')
plt.plot(diesel_energy, label = 'Diesel')
plt.plot(unmet_energy, label = 'Unmet')
plt.plot(renewables_supplied, label = 'Solar generated')
plt.legend()
plt.xlim(0,23)
```

```
plt.ylim(0,5)
plt.xticks(range(0,24,1))
plt.yticks(range(0,6,1))
plt.xlabel('Hour of day')
plt.ylabel('Average energy (kWh per hour)')
plt.title('Energy supply and demand on an average day')
plt.show()
```

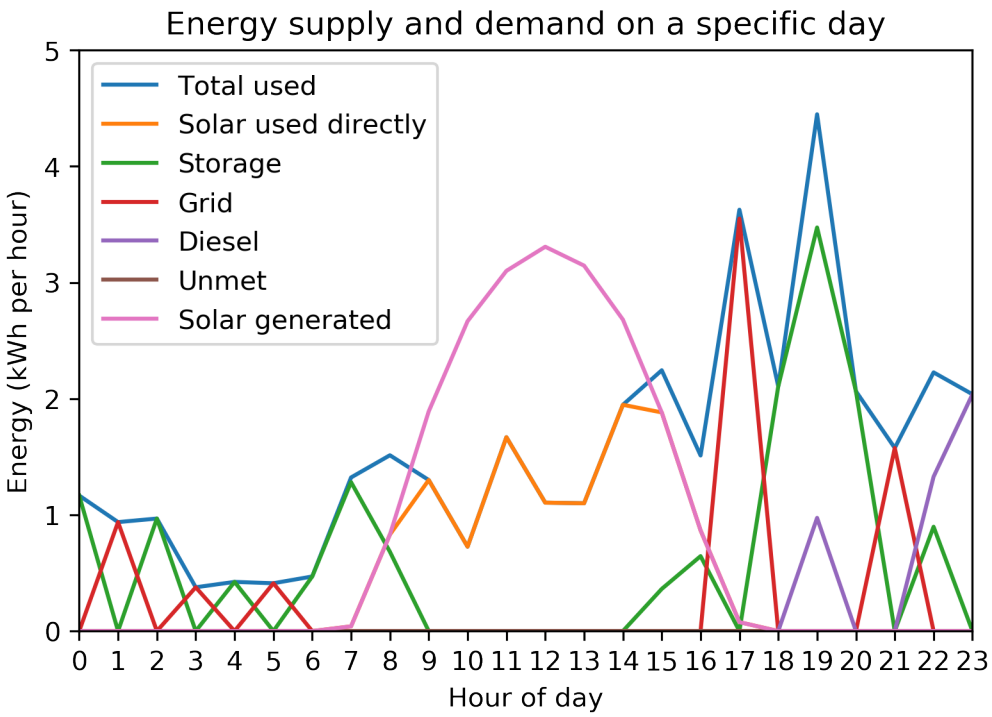


Here we can see that, on average, the solar generation in this system exceeds the demand during the middle of the day, resulting in the total energy used by the community being almost entirely satisfied by solar energy. In the evening, when the load demanded by the community increases and the solar generation decreases, the energy is instead supplied by a combination of battery storage, the grid network, and the diesel generator which continue to be used throughout the night.

It is important to note that the values shown here are for an "average" day and likely not reflective of any single day. Aside from the variations in solar generation and the load demanded, it is far more likely that in any given hour only one or two energy sources would be used at a time. By considering all of the days in the simulation we have artificially smoothed the data to present the averages.

Let's instead look at the first day of data for the simulation period:

[39]:
```
total_used = example_simulation_performance.iloc[0:24]['Total energy used (kWh)']
renewable_energy = example_simulation_performance.iloc[0:24]['Renewables energy␣
 ↪used (kWh)']
storage_energy = example_simulation_performance.iloc[0:24]['Storage energy␣
 ↪supplied (kWh)']
grid_energy = example_simulation_performance.iloc[0:24]['Grid energy (kWh)']
diesel_energy = example_simulation_performance.iloc[0:24]['Diesel energy (kWh)']
unmet_energy = example_simulation_performance.iloc[0:24]['Unmet energy (kWh)']
renewables_supplied = example_simulation_performance.iloc[0:24]['Renewables␣
 ↪energy supplied (kWh)']

plt.plot(total_used, label = 'Total used')
plt.plot(renewable_energy, label = 'Solar used directly')
plt.plot(storage_energy, label = 'Storage')
plt.plot(grid_energy, label = 'Grid')
plt.plot(diesel_energy, label = 'Diesel')
plt.plot(unmet_energy, label = 'Unmet')
plt.plot(renewables_supplied, label = 'Solar generated')
plt.legend()
plt.xlim(0,23)
plt.ylim(0,5)
plt.xticks(range(0,24,1))
plt.yticks(range(0,6,1))
plt.xlabel('Hour of day')
plt.ylabel('Energy (kWh per hour)')
plt.title('Energy supply and demand on a specific day')
plt.show()
```



As we can see, the data is much spikier as it displays the variation between consecutive hours, rather than smoother averages. We can see how once again solar meets most of the demand during the day but during the evening and night the energy is supplied by either battery storage or, if available, the national grid - whose sporadic availability results in a seemingly peaked supply profile.

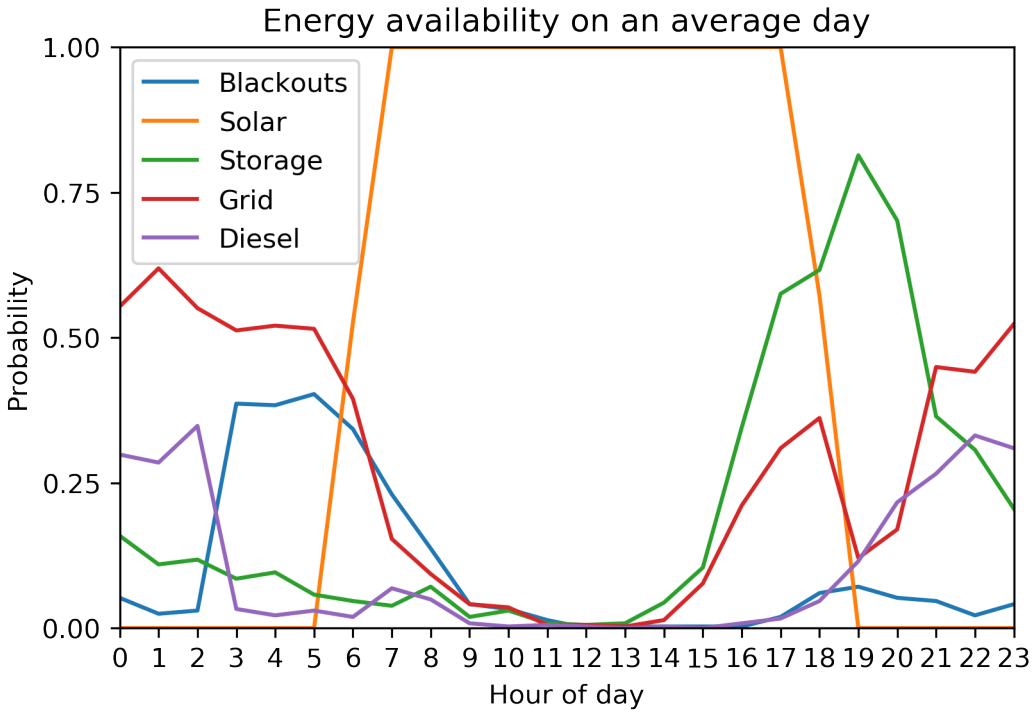
5.4.3 Electricity availability

We can also use the outputs of `example_simulation_performance` to investigate the availability of different electricity services and the times at which different energy sources are used, including the overall measure of service availability recorded in the `Blackouts` variable.

Let's once again consider an "average" day to visualise the availability:

[40]:
```
blackouts = np.mean(np.reshape(example_simulation_performance['Blackouts'].
 ↪values,(365,24)),axis=0)
solar_usage = np.mean(np.reshape(example_simulation_performance['Renewables␣
 ↪energy used (kWh)'].values>0,(365,24)),axis=0)
diesel_times = np.mean(np.reshape(example_simulation_performance['Diesel times'].
 ↪values,(365,24)),axis=0)
grid_usage = np.mean(np.reshape(example_simulation_performance['Grid energy␣
 ↪(kWh)'].values>0,(365,24)),axis=0)
```

```
storage_usage = np.mean(np.reshape(example_simulation_performance['Storage␣
 ↪energy supplied (kWh)'].values>0,(365,24)),axis=0)

plt.plot(blackouts, label = 'Blackouts')
plt.plot(solar_usage, label = 'Solar')
plt.plot(storage_usage, label = 'Storage')
plt.plot(grid_usage, label = 'Grid')
plt.plot(diesel_times, label = 'Diesel')
plt.legend()
plt.xlim(0,23)
plt.ylim(0,1)
plt.xticks(range(0,24,1))
plt.yticks(np.arange(0,1.1,0.25))
plt.xlabel('Hour of day')
plt.ylabel('Probability')
plt.title('Energy availability on an average day')
plt.show()
```



Here we can see that the blackout periods are not consistent throughout the day: although the average is 10% of the time they are much more frequent in the early morning, likely because the battery storage is depleted and the sun has not yet risen. As expected solar energy is always available (at least somewhat) during the day and never at night. Battery storage is used most of the time in the evening and more rarely throughout the night, when grid power is more commonly used. The diesel generator is also sometimes used during the evening and early hours of the

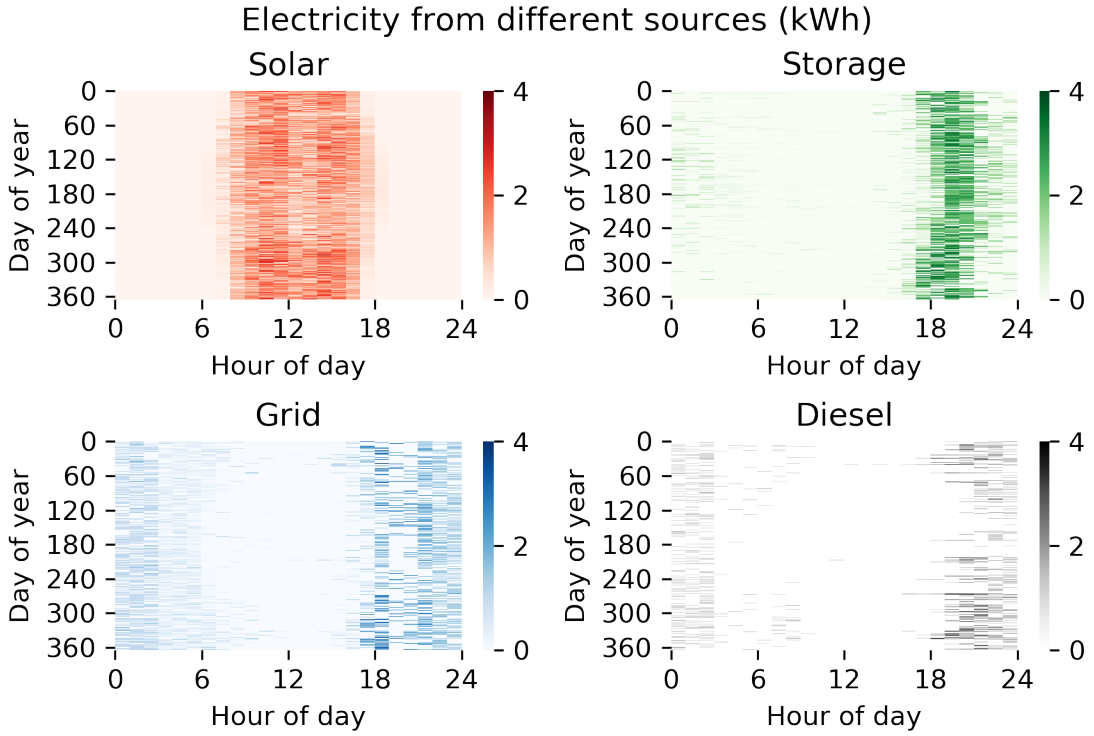
morning, but rarely throughout the night.

5.4.4 Visualising seasonality

CLOVER allows us to investigate both the magnitude and timings of electricity supply and demand in the system at several different timescales. Given that the demand profiles and resource availability change throughout the year, it can be useful to visualise the variation at an annual timescale to identify any seasonal variation:

[41]:
```
total_used = np.reshape(example_simulation_performance['Total energy used␣
 ↪(kWh)'].values,(365,24))
renewable_energy = np.reshape(example_simulation_performance['Renewables energy␣
 ↪used (kWh)'].values,(365,24))
storage_energy = np.reshape(example_simulation_performance['Storage energy␣
 ↪supplied (kWh)'].values,(365,24))
grid_energy = np.reshape(example_simulation_performance['Grid energy (kWh)'].
 ↪values,(365,24))
diesel_energy = np.reshape(example_simulation_performance['Diesel energy (kWh)'].
 ↪values,(365,24))
unmet_energy = np.reshape(example_simulation_performance['Unmet energy (kWh)'].
 ↪values,(365,24))
renewables_supplied = np.reshape(example_simulation_performance['Renewables␣
 ↪energy supplied (kWh)'].values,(365,24))

fig,([ax1,ax2],[ax3,ax4]) = plt.subplots(2,2)#,sharex=True, sharey=True)
g1 = sns.heatmap(renewable_energy,
                 vmin = 0.0, vmax = 4.0,
                 cmap = 'Reds', cbar=True, ax = ax1)
ax1.set(xticks = range(0,25,6), xticklabels = range(0,25,6),
      yticks = range(0,365,60), yticklabels = range(0,365,60),
      xlabel = 'Hour of day', ylabel = 'Day of year',
      title = 'Solar')
g2 = sns.heatmap(storage_energy,
                 vmin = 0.0, vmax = 4.0,
                 cmap = 'Greens', cbar=True, ax = ax2)
ax2.set(xticks = range(0,25,6), xticklabels = range(0,25,6),
      yticks = range(0,365,60), yticklabels = range(0,365,60),
      xlabel = 'Hour of day', ylabel = 'Day of year',
      title = 'Storage')
g3 = sns.heatmap(grid_energy,
                 vmin = 0.0, vmax = 4.0,
                 cmap = 'Blues', cbar=True, ax = ax3)
ax3.set(xticks = range(0,25,6), xticklabels = range(0,25,6),
      yticks = range(0,365,60), yticklabels = range(0,365,60),
      xlabel = 'Hour of day', ylabel = 'Day of year',
      title = 'Grid')
```

```
g4 = sns.heatmap(diesel_energy,
                 vmin = 0.0, vmax = 4.0,
                 cmap = 'Greys', cbar=True, ax = ax4)
ax4.set(xticks = range(0,25,6), xticklabels = range(0,25,6),
      yticks = range(0,365,60), yticklabels = range(0,365,60),
      xlabel = 'Hour of day', ylabel = 'Day of year',
      title = 'Diesel')
plt.tight_layout()
fig.suptitle('Electricity from different sources (kWh)')
fig.subplots_adjust(top=0.87)
plt.xticks(rotation = 0)
plt.show()
```



From this we can see that, as expected, solar energy provides most of the energy during the daytime but has periods of reduced supply in the third quarter of the year. Battery storage provides electricity mainly during the evening just after the sun goes down, with the time of sunset visibly later in the day during the middle of the year, whilst the grid provides power both during the evening and early hours of the morning. The diesel generator has a more seasonally varying profile: it is used less often in the summer months, when solar generation is higher and lasts for longer during the day, pushing back the times when storage is required and reducing the need for diesel generation.

6 Energy system optimisation

6.1 Preparation

Now that we can simulate an energy system (Section 5) we can combine this information with the costs and impact over its lifetime, and those of many other system configurations, in order to select the optimum for our chosen application and criteria. This will allow us to choose the most suitable combination of generation and storage technologies to meet our needs most cost effectively, for example.

Before we can do this we need to provide more information about the the impacts of the installed system, and the conditions that we define to be the optimum for our scenario.

6.1.1 Finance inputs

Financial information is often the key decision metric for designing and implementing a system: renewable energy systems can provide a number of co-benefits, but often ultimately need to make financial sense in order to be selected for deployment. This could be in terms of providing electricity at a given price, for example lower than the incumbent or some alternative option, or ensuring that the total cost of a system does not exceed a certain budget.

The inputs for the financial impact of the system are included in the `Finance inputs` file, which is located in the *Impact* folder of your location folder. Let's take a look at the inputs for the Bahraich case study:

[42]:
```
finance_inputs = pd.read_csv("/Users/prs09/Documents/CLOVER/Locations/Bahraich/
↪Impact/Finance inputs.csv",header=None)
finance_inputs.head(len(finance_inputs))
```

[42]:

	0	1	2
0	Discount rate	0.100	fraction
1	PV cost	500.000	$/kWp
2	PV O&M	5.000	$/kWp p.a.
3	PV cost decrease	5.000	% p.a.
4	Storage cost	400.000	$/kWh
5	Storage O&M	10.000	$/kWh p.a.
6	Storage cost decrease	5.000	% p.a.
7	Diesel generator cost	200.000	$/kW
8	Diesel generator cost decrease	0.000	% p.a.
9	Diesel fuel cost	0.900	$/litre
10	Diesel fuel cost decrease	-1.000	% p.a.
11	Diesel O&M	20.000	$/kW p.a.
12	BOS cost	200.000	$/kW
13	BOS cost decrease	2.000	% p.a.
14	PV installation cost	100.000	$/kW
15	PV installation cost decrease	0.000	% p.a.
16	Diesel installation cost	50.000	$/kW

```
17  Diesel installation cost decrease     0.000       % p.a.
18                      Connection cost   100.000  $/household
19                        Kerosene cost     0.008       $/hour
20                            Grid cost     0.010        $/kWh
21                  Grid extension cost  5000.000         $/km
22             Grid infrastructure cost  2000.000            $
23                        Inverter cost   200.000         $/kW
24              Inverter cost decrease     2.000       % p.a.
25                   Inverter lifetime     4.000        years
26              Inverter size increment     1.000           kW
27                           Misc. costs     0.000         $/kW
28                           General O&M   500.000       $ p.a.
```

These variables describe the costs of the various elements of the energy system and will be dependent on the specifics of your location; although this is true for all of the input files, the costs are likely to vary significantly between locations and can also have a relatively large impact on the results of your optimisation. These data can be difficult to assign specific values (for example if different suppliers have different costs for a given component, or if lower costs are available for purchasing larger quantities) so the general ethos should be to use a value reflective of what is available for your location. Take care to notice the units of each variable as using an input in the wrong units would affect the costs significantly. The table below describes in more detail what each variable means:

Variable	Explanation
`Discount rate`	The discount rate or cost of finance, expressed as a fraction
`PV cost`	Cost of PV modules in $/kWp
`PV O&M`	The annual cost of maintaining PV modules in $/kWp
`PV cost decrease`	The annual cost decrease of PV modules in %
`Storage cost`	The cost of storage in $/kWh
`Storage O&M`	The annual cost of maintaining storage in $/kWh
`Storage cost decrease`	The annual cost decrease of storage in %
`Diesel generator cost`	Cost of a diesel generator in $/kW
`Diesel generator cost decrease`	The annual cost decrease of a diesel generator in %
`Diesel fuel cost`	Cost of diesel fuel in $/litre
`Diesel fuel cost decrease`	The annual cost decrease of diesel fuel in %
`Diesel O&M`	The annual cost of maintaining the diesel generator in $/kWh
`BOS cost`	Cost of balance of systems (BOS) components for PV in $/kWp
`BOS cost decrease`	The annual cost decrease of BOS components in %
`PV installation cost`	Cost of installing PV modules in $/kWp

Variable	Explanation
`PV installation cost decrease`	The annual cost decrease of installing PV modules in %
`Diesel installation cost`	Cost of installing diesel generators in $/kW
`Diesel installation cost decrease`	The annual cost decrease of installing diesel generators in %
`Connection cost`	The cost of connecting a household to the system in $
`Kerosene cost`	The cost of using a kerosene lamp for one hour in $
`Grid cost`	The cost of grid electricity in $/kWh
`Grid extension cost`	The cost of extending the grid by 1 km in $
`Grid infrastructure cost`	The cost of transformers (etc.) to connect the system to the grid in $
`Inverter cost`	Cost of an inverter in $/kW
`Inverter cost decrease`	The annual cost decrease of an inverter in %
`Inverter lifetime`	The lifetime of an inverter in years
`Inverter size increment`	The variety of available sizes of inverters in kW
`Misc. costs`	General miscellaneous capacity-dependent costs for the system in $/kW
`General O&M`	General miscellaneous annual costs for the system in $ per year

The first variable, `Discount rate`, describes the cost of financing used when considering the value of money over time. This is input here as a fraction, `0.1`, corresponding to a discount rate of 10%. The cost of financing can vary significantly between countries and projects depending on many factors, such as the risk associated with the project, and can affect the cost effectiveness of different technologies: broadly speaking, a high discount rate will discourage large initial investment (for example in solar and storage capacity) and favour repeated expenditure over time (for example on diesel fuel).

The costs of solar generation, diesel and storage technologies are treated similarly. Each has a cost associated with purchasing the equipment and maintaining it dependent on the total capacity installed, and an annual cost decrease representing how costs of technology can change over time (we define these variables as a decrease, so a positive value represents costs falling over time whilst a negative value represents them increasing). The diesel generator has additional variables associated with the cost of fuel, solar generation has those for balance of system components such as frames and wiring, and both have costs of initially installing the generation capacity; these are all treated in a similar way.

Two variables relate directly to households in the system. The first is `Connection cost`, which represents the cost of connecting a household to the system; this could include wiring, electricity meters, installation costs, or any others related to providing a household with a connection. The second is `Kerosene cost`, which is the cost that a household incurs for using one kerosene lamp for one hour. This would mainly be comprised of the cost of kerosene fuel, but could also include a contribution to the cost of the lamp itself although this will likely be negligible. This variable is

used to calculate the spending on kerosene by the community when electricity is unavailable.

The cost of electricity from the grid used by the system is assigned in `Grid cost`. Additional costs associated with the national grid are `Grid extension cost`, which represents the cost of extending the network to the community being investigated if it is not currently present there, and `Grid infrastructure cost`, which is the cost of the transformers and other equipment used to convert power from the grid for use in the local distribution network. At present `Grid extension cost` is not used in the financial calculations, but could be used in the future to calculate the breakeven distance at which an off-grid system is more cost effective than extending the national network.

Variables about the inverter used in the system are also included here, with the cost and cost decrease acting similarly to those for generation and storage capacities. In addition the lifetime of the inverter, in years, is included to govern the points in the simulation at which the inverter must be replaced and a new one is purchased; this is included here as depending on the length of a simulation period and its point in the overall lifetime of the system it may necessitate several, or no, replacements. The `Inverter size increment` variable describes the capacity of inverters that are available to be used: for example if this variable is set to 3 kW then the system can used an inverter with a 3 kW, 6 kW, or 9 kW (and so on) capacity, with the inverter being oversized as necessary.

Finally, `Misc. costs` and `General O&M` can be used to include any additional miscellaneous costs that are not captured in the other variables and that are dependent on either the capacity of the system or are annually recurring, respectively.

If you are not using certain technologies in your investigation then it is not necessary to provide values for all of the variables included here. For example, if you are evaluating a solar and battery storage system operating far from the national grid network, you do not need to input values relating to diesel generators, fuel, or the cost of electricity from the national grid. In this case it is best to leave the default values in place or set them to zero, rather than delete them, to ensure that you do not introduce any issues in the way that CLOVER reads the CSV file.

These variables are designed to enumerate the financial inputs into separate and easy-to-update categories, but some can be combined if necessary if their units allows. For example, if a supplier is offering that a complete solar system can be installed for a given price, this can be captured in `PV cost` on its own rather than also trying to assign values to `BOS cost` and `PV installation cost` (which here could be set to zero as they are included in `PV cost`). Likewise if the system is maintained by a single operator, and is not dependent on the capacity that has been installed, it may make more sense to include their salary in `General O&M` and set `PV O&M` (etc.) to be zero; this method would not consider other capacity-dependent costs however, such as minor replacement parts.

Complete the `Finance inputs` CSV file with the financial information for your investigation.

6.1.2 Environmental inputs

CLOVER allows users to analyse the environmental impact of their systems to explore the potential benefits of low-carbon energy technologies. At present these are considered using the greenhouse gas (GHG) emissions of the various technologies both in terms of embedded GHGs from their manufacture and the impact over their lifetimes, for example through the carbon intensity of

the electricity they provide. These can be compared to alternatives, such as the carbon intensity of the national grid network, to advocate for cleaner sources of power.

The inputs for the environmental impact of the system are included in the `GHG inputs` file, which is located in the *Impact* folder of your location folder. Let's take a look at the inputs for the Bahraich case study:

[43]:
```
GHG_inputs = pd.read_csv("/Users/prs09/Documents/CLOVER/Locations/Bahraich/
 ↪Impact/GHG inputs.csv",header=None)
GHG_inputs.head(len(GHG_inputs))
```

[43]:

	0	1	2
0	PV GHGs	3000.000	kgCO2/kWp
1	PV O&M GHGs	5.000	kgCO2/kWp p.a.
2	PV GHG decrease	5.000	% p.a.
3	Storage GHGs	110.000	kgCO2/kWh
4	Storage O&M GHGs	5.000	kgCO2/kWh p.a.
5	Storage GHG decrease	5.000	% p.a.
6	Diesel generator GHGs	2000.000	kgCO2/kW
7	Diesel generator GHG decrease	0.000	% p.a.
8	Diesel fuel GHGs	2.000	kgCO2/litre
9	Diesel O&M GHGs	10.000	kgCO2/kW p.a.
10	BOS GHGs	200.000	kgCO2/kW
11	BOS GHG decrease	2.000	% p.a.
12	PV installation GHGs	50.000	kgCO2/kW
13	PV installation GHG decrease	0.000	% p.a.
14	Diesel installation GHGs	50.000	kgCO2/kW
15	Diesel installation GHG decrease	0.000	% p.a.
16	Connection GHGs	10.000	kgCO2/household
17	Kerosene GHGs	0.055	kgCO2/hour
18	Grid GHGs (initial)	0.800	kgCO2/kWh
19	Grid GHGs (final)	0.400	kgCO2/kWh
20	Grid extension GHGs	290000.000	kgCO2/km
21	Grid infrastructure GHGs	1200000.000	kgCO2
22	Inverter GHGs	75.000	kgCO2/kW
23	Inverter GHG decrease	2.000	% p.a.
24	Inverter lifetime	4.000	years
25	Inverter size increment	1.000	kW
26	Misc. GHGs	0.000	kgCO2/kW
27	General O&M GHGs	200.000	kgCO2 p.a.

Similarly to the financial inputs, the environmental inputs consider the initial impact of installing the technologies (the embedded GHG emissions from their manufacture) and the impact of maintaining them, as well as the potential for technologies to decrease their impact over time as manufacturing becomes more efficient, for example. These data are typically much more difficult to identify values for as the environmental impact is rarely considered in such detail, if at all, as a secondary metric to the financial impact. As a result it may be better to either use the default values provided or set them to zero to disregard them depending on the nature of your investigation;

either may be appropriate, as long as the decision is acknowledged and justified where necessary. The table below describes in more detail what each variable means:

Variable	Explanation
PV GHGs	GHGs of PV modules in kgCO2/kWp
PV O&M GHGs	The annual cost of maintaining PV modules in kgCO2/kWp
PV GHGs decrease	The annual GHG decrease of PV modules in %
Storage GHGs	The GHGs of storage in kgCO2/kWh
Storage O&M GHGs	The annual GHGs of maintaining storage in kgCO2/kWh
Storage GHGs decrease	The annual GHG decrease of storage in %
Diesel generator GHGs	GHGs of a diesel generator in kgCO2/kW
Diesel generator GHG decrease	The annual GHG decrease of a diesel generator in %
Diesel fuel GHGs	GHGs of diesel fuel in kgCO2/litre
Diesel O&M GHGs	The annual GHGs of maintaining the diesel generator in kgCO2/kWh
BOS GHGs	GHGs of balance of systems (BOS) components for PV in kgCO2/kWp
BOS GHGs decrease	The annual GHG decrease of BOS components in %
PV installation GHGs	GHGs of installing PV modules in kgCO2/kWp
PV installation GHGs decrease	The annual GHG decrease of installing PV modules in %
Diesel installation GHGs	GHGs of installing diesel generators in kgCO2/kW
Diesel installation GHGs decrease	The annual GHGs decrease of installing diesel generators in %
Connection GHGs	The GHGs of connecting a household to the system in kgCO2
Kerosene GHGs	The GHGs of using a kerosene lamp for one hour in kgCO2
Grid GHGs (initial)	The GHGs of grid electricity at the start of the time period in kgCO2/kWh
Grid GHGs (final)	The GHGs of grid electricity at the end of the time period in kgCO2/kWh
Grid extension GHGs	The GHGs of extending the grid by 1 km in kgCO2
Grid infrastructure GHGs	The GHGs of transformers (etc.) to connect the system to the grid in kgCO2
Inverter GHGs	GHGs of an inverter in kgCO2/kW
Inverter GHGs decrease	The annual GHG decrease of an inverter in %
Inverter lifetime	The lifetime of an inverter in years
Inverter size increment	The variety of available sizes of inverters in kW

Variable	Explanation
`Misc. GHGs`	General miscellaneous capacity-dependent GHGs for the system in kgCO2/kW
`General O&M GHGs`	General miscellaneous annual GHGs for the system in kgCO2 per year

Almost all of the variables in the above table are environmental analogues to those in the financial inputs and therefore their descriptions will not be repeated here. The exceptions to this are `Grid GHGs (initial)` and `Grid GHGs (final)`, which describe the emissions intensity of the grid network at the start and end of the considered lifetime of the system respectively. These allow the user to take into account how the electricity grid might be decarbonised over time in line with national policy objectives, which would have a subsequent impact on the GHGs of a system using grid electricity throughout its lifetime.

In general many of the technologies have relatively carbon-intensive manufacturing processes, such as processing silicon for solar panels and smelting metals for balance of systems components and wiring, whilst diesel fuel has notoriously high emissions from its usage. Emissions associated with operation and maintenance could come from the maintenance itself (for example replacement parts) or other considerations, such as the GHGs of a worker travelling to the site; in practice, however, these O&M emissions are usually dwarfed by the embedded emissions of equipment and those from diesel fuel and the national grid. Emissions from transporting equipment are not explicitly included but can be implicitly included by adding them to the appropriate variables, for example setting `PV GHGs` to a value including both the emissions from manufacturing a panel and from shipping it to the installation site (both in terms of capacity, here kgCO2/kWp).

Complete the `GHG inputs` CSV file with the financial information for your investigation.

6.1.3 Optimisation inputs

The optimisation process in CLOVER is mostly automatic but in order for it to work we need to state the conditions under which it should operate. CLOVER performs a large number of system simulations, each with different combinations of generation and storage capacity sizes, and appraises them based on their technical, financial and environmental performances; depending on our interests, any of them might be considered "the best". For this reason we need to provide CLOVER with some details about what we consider to be the "optimum" system using two variables that we define:

- The *threshold criterion*, which determines whether a simulated system meets the standards required to be considered as a potential optimum system, and
- The *optimisation criterion*, which determines which of the potential systems is selected to be the best.

Simply put, the threshold criterion decides whether a system meets our stated needs and, as many systems are likely to be able to meet our needs, the optimisation criterion selects the one that performs the best according to its financial or environmental impact. As an example, consider that we want to design a system which provides electricity 95% of the time or more: clearly many systems will be able to do that, some much larger than would be affordable, so we decide that

the one which provides the lowest cost of electricity would be best. Here we would define the threshold criterion to be `Blackouts` (from Section 5.2.3 and set to a value of `0.05`, i.e. a maximum of 5% of the time can experience a loss of supply), whilst the optimisation criterion would be the levelised cost of used electricity in $/kWh (LCUE, explained further later) with the lowest being the best. CLOVER would then identify a system which meets those two criteria, and give us a range of other impact metrics as well.

We also need to provide information about scenario we are investigating: optimising over different time periods will potentially provide different results as resource generation, load demands, technological performance and degradation all change over time. CLOVER uses a step-by-step optimisation process which divides the total investigation lifetime into shorter time periods; this allows us to replicate the process of designing a system to meet some future needs, then revisiting the system after a few years to upgrade it as necessary to meet a growing demand or maintain its performance. This also allows us to consider that additional capacity in the context of what has already been installed, adding only what is necessary rather than overhauling the existing equipment.

The inputs for the optimisation parameters are included in the `Optimisation inputs` file, which is located in the *Optimisation* folder of your location folder. Let's take a look at the inputs for the Bahraich case study:

[44]:
```
optimisation_inputs = pd.read_csv("/Users/prs09/Documents/CLOVER/Locations/
↪Bahraich/Optimisation/Optimisation inputs.csv",header=None)
optimisation_inputs.head(len(optimisation_inputs))
```

[44]:

	0	1	\
0	Scenario length	12	
1	Iteration length	4	
2	PV size (min)	0	
3	PV size (max)	1	
4	PV size (step)	5	
5	PV size (increase)	0	
6	Storage size (min)	0	
7	Storage size (max)	1	
8	Storage size (step)	5	
9	Storage size (increase)	0	
10	Threshold criterion	Blackouts	
11	Threshold value	0.05	
12	Optimisation criterion	LCUE ($/kWh)	

	2	3
0	years	NaN
1	years	NaN
2	kWp	NaN
3	kWp	NaN
4	kWp	NaN
5	kWp	Set to 0 to ignore
6	kWh	NaN

```
7                                        kWh                  NaN
8                                        kWh                  NaN
9                                        kWh   Set to 0 to ignore
10            Name of column from appraisal                  NaN
11  Max/min value permitted (see guidance)                   NaN
12            Name of column from appraisal                  NaN
```

Some of the variables included in the `Optimisation inputs` CSV file are not used by the current optimisation function. These were left over from previous processes which have since been improved to increase the speed and efficiency of the overall optimisation. These previous functions (and their input variables) are *deprecated*, meaning that they are not used by the standard processes in the model but are still present in dormant sections of the code and can be used if specifically desired by the user. For clarity, these functions are not covered in this document but their variables are still included in the `Optimisation inputs` CSV file for completeness.

The table below describes in more detail what each variable means:

Variable	Explanation
`Scenario length`	Total length of the investigation period in years
`Iteration length`	Length of each step-by-step time period in years
`PV size (min)`	Minimum size of PV capacity to be considered in kWp
`PV size (max)`	*Deprecated*
`PV size (step)`	Optimisation resolution for PV size in kWp
`PV size (increase)`	*Deprecated*
`Storage size (min)`	Minimum size of storage capacity to be considered in kWh
`Storage size (max)`	*Deprecated*
`Storage size (step)`	Optimisation resolution for storage size in kWh
`Storage size (increase)`	*Deprecated*
`Threshold criterion`	Criterion for identifying sufficient systems
`Threshold value`	Value required for a system to be considered sufficient
`Optimisation criterion`	Criterion for identifying optimum system

Two variables control the length of the investigation period and that of each of the sub-periods, with the total `Scenario length` comprised of several `Iteration length`. For example, setting `Scenario length = 12` and `Iteration length = 4` would mean that there would be three distinct periods: the first considering the first four years, the second considering the next four years (including the performance of the system that was already installed for the period before), and the third considering the last set of four years (again including the periods before). `Scenario length` must be a multiple of `Iteration length` for the optimisation process to work, and `Scenario length` can be any length up to the total investigation lifetime we defined in Section 2.5 to be 20 years.

The optimisation process in CLOVER first considers a system with the smallest generation and storage capacity and gradually increases it until it finds one that meets the threshold criterion. The first system it considers at the start of the investigation period, with the smallest installed capacity, is defined by `PV size (min)` and `Storage size (min)`. These variables identify the starting point of the optimisation which, for many, would be no installed capacity at all - meaning `PV size (min) = 0` and `Storage size (min) = 0`. If there is already a system in place then set these variables to the capacity of that system. When CLOVER moves on to the second (or later) iterations during the optimisation process it automatically considers the capacity that was installed during the previous time period.

The optimisation function requires the user to define the resolution of the investigation, defined through the variables `PV size (step)` and `Storage size (step)` respectively: once CLOVER has investigated a system it then analyses one with a larger capacity, with the increase in capacity of the next system defined by these two variables. For example, with `PV size (step) = 1` CLOVER would consider systems with `PV size = 0, 1, 2, 3, ...` whereas `PV size (step) = 5` would consider `PV size = 0, 5, 10, 15, ...`, with the same logic being true for `Storage size (step)`. Both variables can be set to different values, and can be non-integers. For an investigation of a theoretical system then choosing a round number for the step size might be more convenient, but for a real investigation it could be better to use the real available capacity of the equipment being installed. For example, if you know that the system will be build using solar panels with a capacity of 350 Wp, you could choose to use `PV size (step) = 0.35`.

`Threshold criterion` is the name of the variable being used to decide whether a system provides sufficient performance to be considered as a potential optimum system, as described earlier, with `Threshold value` being the value it must meet. The choices for `Threshold criterion` are described later in Section 6.2.4 and CLOVER knows automatically whether the `Threshold value` should be considered as a maximum or minimum value: for example, if `Threshold criterion = Blackouts` then CLOVER interprets `Threshold value = 0.05` as systems are permitted to have blackout periods for a maximum of 5% of the time. `Optimisation criterion`, meanwhile, is the variable being used to select the optimum system and CLOVER similarly knows whether this should be interpreted as a maximum or minimum: for example, `Optimisation criterion = LCUE ($/kWh)` would be used to find the lowest cost of electricity. Take care that the inputs for `Threshold criterion` and `Optimisation criterion` are spelled correctly, otherwise the optimisation process will not work.

6.1.4 Considerations

CLOVER can take a number of variables as either the threshold or optimisation criteria and these should be chosen to best reflect the context of the investigation and its goals. Typically the threshold criterion relates to the technological performance of the system on the basis that, in order for a system to be viable, it needs to be able to provide a minimum level of service to the community which is (almost always) based on its core technical functionality, rather than economic or environmental factors. The level of service availability, either in terms of the time energy is (un)available or the proportion of demanded that should be met, are the most commonly used threshold criteria. Selecting an appropriate threshold value will vary depending on the situation, for example a system for basic domestic applications could permit blackouts 5-10% of the time whilst one for a hospital may need 0% system downtime.

The optimisation criterion, meanwhile, is typically a measure of the system impact either in financial or environmental terms. Most commonly this is the levelised cost of used electricity (LCUE, measured in $/kWh), which considers the cost of power over the lifetime of the system and is normally most relevant to system designers, but could also be other financial metrics such as the total system cost which may be more relevant to donor-led projects. Environmental analogues of these metrics, such as the emissions intensity of electricity or the cumulative GHG emissions, may be more relevant for projects driven by climate change objectives.

Whilst a single criterion must be selected for use in the optimisation process, as we will see in Section 6.2.4 the optimisation process gives us all of the available outputs and so we can investigate all of them for an optimum system configuration. Using different optimisation criteria may yield different results but often the optimum systems will be relatively similar: for a given level of service availability, it is likely that the capacity of a system optimised for lowest cost of electricity will be similar to one optimised for lowest cumulative cost, but may not be identical.

Choosing the resolution of an optimisation is a trade-off between precision and computation time. Taking the above example of 1 kWp or 5 kWp resolutions, the former method would be more likely to suggest a system closer to the true optimum: for example, if the true optimum system (if it were possible to know it exactly) has `PV size = 7` then the former method would be able to find it, but the latter method would suggest either `PV size = 5` or `PV size = 10`, depending on how the other aspects of system influence this. Even if the true optimum system is actually `PV size = 7.2`, the former method still gets closer.

The importance of a precise answer will depend on the requirements of your investigation, the size of the system, your available processing power and your patience, so there is no exact answer for the "best" step sizes to use. In general, it is good practice to run an initial optimisation with a low resolution (e.g. `PV size = 10`) to get an idea of the order of magnitude of the system: if the optimum output for this trial run is `PV size = 20`, it would be appropriate to re-run the optimisation with a higher resolution (e.g. step sizes of 1 kWp or 5 kWp) to gain greater precision. If the output is `PV size = 300`, it might be better to increase the step size (e.g. to 20 kWp) to reduce computational time.

The optimisation process CLOVER uses provides the optimum system for the iteration period under consideration, which is not necessarily the same as one for the entire lifetime of the system. This is to replicate real-life system design and the timeframes of community energy projects, where plans are built around time horizons of a few years owing to practical constraints such as funding, the limits of demand prediction, or the likelihood of altering the business model. This also matches a reasonable timeframe of making major upgrades to the system, for example returning to the site every few years to perform capacity upgrades to meet a growing demand. As such CLOVER identifies the optimum system for the next short-term time window, providing the system that best meets the requirements for the next stage of design, before moving on to the next stage in time. This allows users to know the optimum system for the immediate future and what subsequent system upgrades might look like to maintain that optimum performance. As will all predictions the further into the future you consider the greater uncertainty there is, but remember that you can always return in a few years' time to re-run your optimisation when it's time to upgrade your system.

As a result the of the precision of your system, the CLOVER optimisation process might suggest different pathways of capacity sizing over the lifetime of the system. In the example above, if a system requires `PV size = 7.2` in its first period and `PV size = 12.1` in its second then using a

`PV size (step)` of 1 kWp would probably give `PV size = 7` and `PV size = 12`, but using a step size of 10 kWp might give `PV size = 10` for both periods. The latter case might have evaluated the relative costs and benefits of increasing the capacity to `PV size = 20` and found that (for example) increasing the storage capacity instead was more worthwhile. The precision we chose has therefore locked the two optimisations into different pathways, one with a larger PV capacity and one with a larger storage capacity; both will provide the same level of service (as the threshold criteria are the same in both cases) but the optimisation impacts such as the costs will be different. In some cases this effect can be exacerbated over several time periods, but in general the most effective way to mitigate this is through using higher precision in your optimisations to stay as close as possible to the true optimum system.

Complete the `Optimisation inputs` CSV file with the optimisation information for your investigation.

6.2 Performing an optimisation of an energy system

6.2.1 Preparation

We are now able to perform an optimisation of an energy system using the Optimisation module. This relies on all of the information we have input and generated previously in the Section 5, and the earlier parts of this section. This will let us identify the optimum system, find its solar and battery capacity, and get information about its technological performance and its financial and environmental impacts.

To perform an optimisation we must first **run the Optimisation script (using the green arrow in the Spyder console)**, which we do here using the following:

[45]:
```
sys.path.insert(0, '/Users/prs09/Documents/CLOVER/Scripts/Optimisation scripts/')
from Optimisation import Optimisation
```

The optimisation function we will use does not take any arguments from the console as they are all included in the previous CSV files. We can override them in the function but this is generally for specific applications which are not relevant here.

6.2.2 Running an optimisation

To run an optimisation we **run the following function in the console**, saving the output as a variable called `example_optimisation` so we can look at the outputs in more detail:

```
example_optimisation = Optimisation().multiple_optimisation_step()
```

This will run the optimisation function, which automatically simulates a variety of systems with different capacity sizes until it finds the optimum configuration. **The optimisation process can take a long time to complete**, ranging from minutes to hours depending on your scenario and available computing resources (hence the advice to begin at a relatively low resolution to get an idea of the expected system size). If you need to cancel the optimisation function before it completes, click on the console and press `Ctrl + C` on your keyboard.

Whilst the optimisation function runs it outputs some information to the screen to update on its progress. An example is below:

```
Step 1 of 3

Time taken for simulation: 0.16 seconds per year

Current system: Blackouts = 0.1, Target = 0.05
```

In this case, the function is considering the first four-year period in the twelve-year lifetime (`Step 1 of 3`). As before when running a simulation, it prints information about the speed at which the simulations are being performed. Finally it presents some information about the performance of the system it just simulated in terms of the chosen threshold criterion (`Blackouts = 0.1`) and the desired target set by the `Optimisation inputs` CSV file (`Target = 0.05`).

In this case, we can notice that all of the systems have `Blackouts = 0.1` or lower, even for the very first system that is simulated which has `PV_size = 0` and `storage_size = 0`. This is because the `Diesel backup = Y` and `Diesel backup threshold = 0.1` (from Section 5.1.2) which means that the diesel generator is activated to provide `Blackouts = 0.1` at the very least. Because we set the threshold criterion higher than this, the system is unable to use diesel alone to meet the requirements thereby forcing it to install some renewable capacity to reach `Blackouts = 0.05`. We could have set `Diesel backup threshold = 0.05` to potentially allow the use of diesel generation to meet our target, but in our case this results in a diesel-dominated system which doesn't provide a good example!

A few other statements are shown in the console that update you on what the optimisation process is doing. `Using single line optimisation` means that CLOVER is scanning along a fixed capacity of either PV or storage whilst varying the other, whilst `Increasing storage size` and `Increasing PV size` describe the steps CLOVER is taking to identify the optimum system by checking larger configurations. These are for information only so there is no need to do anything with this information.

Finally, when the optimisation is complete, you receive the message:

```
Time taken for optimisation: 10:15 minutes
```

This lets you know how long the entire optimisation process took (in this case just over 10 minutes). This is for information only but you can use it to judge how to amend your resolution for your next optimisation, or to know approximately how long you will need to wait next time you optimise the system.

Opening the `example_optimisation` variable allows us to view the results of the optimisation function, which are presented in a table. Each row corresponds to an iteration period (in our case, three) with the columns describing the results for each stage of the optimisation. There is a large number of results and so we will discuss them in more detail in Section 6.2.4.

6.2.3 Saving optimisation results and opening saved files

Similarly to the reasons for saving simulation results described earlier in Section 5.2.4 it is important to save the optimisation results for future reference. It is especially important because of the time it takes to run optimisations, which can be much longer than any single simulation.

CLOVER provides a function to save the output of optimisations as CSV files, storing the data much more conveniently. To save an output (`optimisation_name`) we need to have first stored it as a variable, and choose a filename (`filename`) to store it (note that the `filename` variable in this function must be a string). In our case `optimisation_name = example_optimisation`, and we choose `file_name = 'my_saved_optimisation'`. **To save the optimisation results we run the function**:

```
Optimisation().save_optimisation(optimisation_name = example_optimisation,
        filename = 'my_saved_optimisation')
```

This function creates a new CSV file in the *Saved optimisations* folder in the *Optimisation* folder in your location folder titled `my_saved_optimisation.csv`. If the filename variable is left blank, the title of the CSV defaults to the time the save operation was performed. Be aware that running this function with a filename that already exists will overwrite the existing file.

To open a saved file, we use the name of the CSV file to open the correct result, for example:

```
opened_optimisation = Optimisation().open_optimisation(filename = 'my_saved_optimisation')
```

This will open the `my_saved_optimisation.csv` file and record the data as a new variable, `opened_optimisation`, which will be in the same format as the original saved variable `example_optimisation`.

6.2.4 Optimisation results

Our optimisation function gives us a large number of results in its output variable. Let's open the example saved optimisation file, located in the *Bahraich* folder:

[46]:
```
opened_optimisation = Optimisation().open_optimisation(filename =␣
↪'my_saved_optimisation')
opened_optimisation.T.head(len(opened_optimisation.T)).round(3)
```

[46]:

	0	0	0
Start year	0.000	4.000	8.000
End year	4.000	8.000	12.000
Initial PV size	10.000	20.000	30.000
Initial storage size	40.000	70.000	100.000
Final PV size	9.600	19.200	28.800
Final storage size	34.915	60.884	86.162
Diesel capacity	0.000	0.000	0.000
Cumulative cost ($)	29618.740	46665.340	58062.030
Cumulative system cost ($)	28863.620	45517.130	56608.180
Cumulative GHGs (kgCO2eq)	55702.070	110820.050	169535.870
Cumulative system GHGs (kgCO2eq)	48801.380	98672.910	151484.150
Cumulative energy (kWh)	53963.453	153124.929	310774.471
Cumulative discounted energy (kWh)	44290.408	99759.179	160367.390
LCUE ($/kWh)	0.652	0.456	0.353
Emissions intensity (gCO2/kWh)	904.341	644.395	487.441
Blackouts	0.043	0.039	0.049

Unmet energy fraction	0.029	0.026	0.030
Renewables fraction	0.764	0.767	0.768
Total energy (kWh)	53963.453	99161.476	157649.542
Total load energy (kWh)	54635.778	100110.598	159832.375
Unmet energy (kWh)	1610.022	2633.543	4726.531
Renewable energy (kWh)	22179.611	41880.569	69248.362
Storage energy (kWh)	19067.076	34183.638	51891.289
Grid energy (kWh)	12716.766	23097.270	36509.891
Diesel energy (kWh)	0.000	0.000	0.000
Discounted energy (kWh)	44290.408	55468.771	60608.211
Kerosene displacement	0.954	0.965	0.960
Diesel fuel usage (l)	0.000	0.000	0.000
Total cost ($)	29618.740	17046.600	11396.690
Total system cost ($)	28863.620	16653.510	11091.050
New equipment cost ($)	25200.000	13308.900	8218.550
New connection cost ($)	400.000	262.440	172.190
O&M cost ($)	3159.960	2953.460	2560.360
Diesel cost ($)	0.000	0.000	0.000
Grid cost ($)	103.660	128.720	139.960
Kerosene cost ($)	755.130	393.090	305.640
Kerosene cost mitigated ($)	17295.780	11875.070	7921.960
Total GHGs (kgCO2eq)	55702.070	55117.980	58715.820
Total system GHGs (kgCO2eq)	48801.380	49871.530	52811.240
New equipment GHGs (kgCO2eq)	37350.000	31617.150	27556.470
New connection GHGs (kgCO2eq)	40.000	40.000	40.000
O&M GHGs (kgCO2eq)	1800.000	2600.000	3400.000
Diesel GHGs (kgCO2eq)	0.000	0.000	0.000
Grid GHGs (kgCO2eq)	9611.380	15614.390	21814.770
Kerosene GHGs (kgCO2eq)	6900.680	5246.450	5904.580
Kerosene GHGs mitigated (kgCO2eq)	142468.860	143334.180	139938.040

Note that here we have displayed the outputs vertically (rather than horizontally as in the default) so that they are easier to see. These results are automatically generated as a result of the optimisation process but it is also possible to calculate them for a specific case study system, described later in Section 6.4.5.

Many of these variables have similar names, so here are some pointers on interpreting them:

- *Cumulative* means that the variable is a running total since the start of the system lifetime (i.e. the current period and all previous periods)
- *New* means the variable includes only the current period
- *Cost* values are all discounted costs, meaning they take into account the discount rate, and are presented in terms of $ at the start of the lifetime
- *System* means the electricity system (PV and storage capacity, O&M, diesel, grid, household connections)
- *Total* refers to the costs from all sources (i.e. the *system* and kerosene expenditure) a each time

period

- *Energy* is either the amount of energy or (when specified) the discounted energy which takes into account the discount rate

These identifiers (or the names themselves) should provide enough description to work out what each variable means. Now let's consider a few of the most relevant ones that you're most likely to use in more detail:

Variable	Explanation
`Start / End year`	The first and last years in each considered period
`Initial PV / storage size`	Installed capacity of PV/storage at the start of the period
`Final PV / storage size`	Equivalent capacity of PV/storage at the end of the period, after degradation
`Diesel capacity`	Installed capacity of diesel generator required for backup (kW)
`LCUE ($/kWh)`	Levelised cost of used electricity up to the end of the simulation period ($/kWh)
`Emissions intensity (gCO2/kWh)`	Emissions intensity up to the end of the simulation period (gCO2/kWh)
`Blackouts`	Proportion of time where blackouts occurred for each iteration period
`Unmet energy fraction`	Proportion unmet energy for each iteration period
`Renewables fraction`	Proportion energy from PV and storage for each iteration period
`Kerosene displacement`	Proportion kerosene mitigated during each iteration period
`Cumulative cost ($)`	Running total of cumulative costs over the system lifetime in $
`Cumulative GHGs (kgCO2eq)`	Running total of cumulative GHGs over the system lifetime in kgCO2eq

The first variables are useful in characterising the iteration period (`Start year` and `End year`) and knowing the capacity of the system that was installed (`Initial PV size` and `Initial storage size`), possibly the most important variables in terms of describing the system itself. The `Final PV size` and `Final storage size` describe the state of the equipment at the end of the iteration period, incorporating the effects of degradation. Note that for the next iteration period, CLOVER may choose to "refresh" the installed capacity either by returning it to the same initial size from the previous iteration or increasing it, or leave it "as is" and not add any further capacity. In the latter case the final size from the previous iteration would be the initial size of the next; this is relatively common for systems without growing load profiles. Finally `Diesel capacity` describes the size of the diesel generator installed to provide power as a backup.

The remaining variables in the table are useful to use as either threshold or optimisation criteria for designing your system: many of the variables in `example_optimisation` could be used for this

purpose, but these are the most commonly used and most likely to be relevant for your investigation. Bear in mind that **the input for** `Threshold criterion` **or** `Optimisation criterion` **must match the variable name exactly** otherwise the function will fail. For example, `LCUE ($/kWh)` would work but `LCUE` would not.

Two important variables relate to the financial and environmental impact of the system at each point in its lifetime: the levelised cost of used electricity (LCUE, $/kWh) and emissions intensity (gCO2/kWh). These are convenient metrics of capturing the performance of the system per unit of electricity and so are readily comparable to other systems for the same (or other) applications. The LCUE is the most common optimisation criterion as it conveniently captures core financial information which is readily comparable to other situations. LCUE considers only the energy *used* by the community, explicitly defined as the useful electricity actually consumed rather than (for example) all of the electricity that was generated, some of which was likely dumped if demand was already met. Furthermore this definition accounts for blackout periods when electricity was demanded, but unmet.

For each iteration step these variables provide information about the LCUE and emissions intensity by the end of that step: for example, the LCUE over the first time period is `$0.65/kWh` but falls to `$0.35/kWh` by the end of the optimisation period over the whole lifetime. This is because the impacts of deploying the equipment occur at the start of each period but their benefits, in terms of the electricity they provide, are received over the total remaining lifetime of the system including subsequent iteration periods. The assumption here is that a community energy project would need to purchase equipment outright before it can be deployed, reflecting common practice, but would not capture more nuanced financial plans which would require further development outside of CLOVER.

The next set of variables describe the performance of the system during each iteration period. These can be used as optimisation criteria but are also very useful as threshold criteria and each period is considered independently to ensure the minimum performance threshold is met throughout. If this were not the case, if the first period performed very well then the second could perform below the expectation of the threshold, if the average over the entire period still satisfied the defined threshold value. As before `Blackouts` and `Unmet energy fraction` describe the service unavailability in terms of the proportion of time electricity is not available and the proportion of energy demand that is not satisfied by the system, and when used as a threshold criterion are defined as the maximum allowable value for a system to be sufficient. `Renewables fraction` describes the proportion of energy that was supplied by either solar or storage (i.e. not from the grid or diesel generation), whilst `Kerosene displacement` describes the proportion of kerosene usage that was mitigated by the availability of electricity. For the latter two variables, these are considered a minimum allowable value when used as a threshold criterion.

Finally, `Cumulative cost ($)` and `Cumulative GHGs (kgCO2eq)` are running totals of the cumulative incurred costs or GHGs up to the point of each iteration period. These are the best values for capturing the entire impact of the scenario over its lifetime, including impacts from all sources including kerosene usage. For the impacts of the electricity system alone, `Cumulative system cost ($)` and `Cumulative system GHGs (kgCO2eq)` can instead be used (if kerosene usage was not already disregarded earlier). In this case, the total lifetime impact of the system (and residual kerosene usage) was that it cost around $58,000 and emitted around 170 tCO2 to supply power to the community of 100 households for 12 years.

You now should have everything you need to investigate your scenarios, perform optimisations

and design electricity systems. Good luck!

6.3 Troubleshooting

Most of the *Optimisation* functionality is contained within the `Optimisation().multiple_optimisation_step()` function and so potential issues are most likely to come either from how the module gathers data from other parts of CLOVER or from the input CSV files.

- Ensure that the `self.location` variable is correct in all of the modules that `Optimisation` imports
- Check that your `Finance inputs`, `GHG inputs` and `Optimisation inputs`, CSVs are completed with the scenario you want to investigate, and any changes are saved in the CSV file before running another simulation
- Check that your `Scenario inputs` CSV is completed with the scenario you want to investigate and the *Energy_System* module is working correctly
- Ensure that you use the correct filename when saving and opening previous simulations
- When running optimisations, remember to save the output of `Optimisation().multiple_optimisation_step()` as a variable

6.4 Extension and visualisation

6.4.1 Changing parameter optimisation

Investigating the design of a system under a given threshold criterion allows us to identify the optimum configuration under those conditions, but in many cases the target value of the threshold criterion might be negotiable. A common example is the tradeoff between the reliability of the system and the cost of electricity: with all other scenario conditions being the same, a more reliable system will need greater generation and storage capacities which will increase the investment required and so increase the cost of electricity. It is often interesting, therefore, to investigate a range of values for the threshold criterion to see what effect it has on the resulting optimisation criterion, and from those options select a system that best suits our needs.

We can demonstrate the above example by considering a system with the `Optimisation criterion` as `LCUE ($/kWh)` and the `Threshold criterion` as `Blackouts` as before, but with a range of `Threshold value`. Whereas before we set this to be `0.05`, we now want to optimise using several options ranging from `0.25` to `0.05` in steps of `0.05`. All of the other inputs in the `Optimisation inputs` CSV file are the same as Section 6.1.3 but we have set `Diesel backup = N` in the `Sceneario inputs` CSV file to avoid the diesel generator backup improving the reliability, in order to better demonstrate our goal here.

We can use the below function to automatically perform several optimisations in a row and save the results to CSV files:

```
Optimisation().changing_parameter_optimisation(parameter = 'Threshold value',
    parameter_values = [0.25,0.20,0.15,0.10,0.05],
```

```
    results_folder_name = 'Changing parameter example')
```

This function takes more variables than the usual `Optimisation().multiple_optimisation_step()` function as it repeatedly uses that function, with changing parameters, to perform several optimisations. Be aware that, by doing this, the `Optimisation().changing_parameter_optimisation(...)` function takes several times longer to finish and, when investigating systems with the strictest threshold values, can take much more time than previous optimisations you might have performed.

We define which parameter we want to change each time using the `parameter` input, which can correspond to any valiable in the `Optimisation inputs` CSV file. Here we want to investigate the effect of using different values for the threshold criterion, so we set `parameter = 'Threshold value'`. Next we need to specify the values we want to investigate which we input as a `list`-type variable; using the information above, we therefore input `parameter_values = [0.25,0.20,0.15,0.10,0.05]`. The list of `parameter_values` can be as long or short as desired. This will run the optimisation function sequentially for each value (`Threshold value = 0.25`, `Threshold value = 0.20`, ..., `Threshold value = 0.05`) by automatically rewriting the `Optimisation inputs` CSV file: be aware that the final rewrite will leave the final `Threshold value` in the file, so you may need to remember to amend this if you use the usual `Optimisation().multiple_optimisation_step()` again.

Finally, the results of each optimisation are saved in a folder with the name `results_folder_name` in the *Saved optimisations* folder in your *Optimisation* folder, which itself is in the overall file structure for your location. Here we use `results_folder_name = 'Changing parameter example'` to specify the place where the outputs should be saved. The folder must be manually created beforehand in order to save the files there, otherwise they will be saved in a new folder with the date and time the function was performed.

Whilst running, the function will save the outputs of each optimisation as a CSV with the name `Threshold value = 0.25.csv` (and so on) to allow you to investigate each system individually afterwards. Once the entire function is complete, it outputs a summary file of the final results for each optimised system in the file `Threshold value lifetime summary of results.csv` (the name of this file will change if you used a different `parameter`). This is composed of the final row of each optimisation so bear in mind that some of the data refers to a single iteration period when interpreting the results and refer to Section 6.2.4 for guidance. If you need to cancel the function for any reason, the output CSVs that have already been saved will still be available but this final CSV will not, as it is compiled at the end of the process (but can be manually compiled if necessary).

Using `parameter = 'Threshold value'` is the most common usage of this function to set the process running self-sufficiently, allowing you to leave it to work away in the background without intervention and return results when they are ready. It is also possible to use it for other types of investigations, but be sure to think carefully how these would be interpreted by the function. For example, if you wanted to investigate the effect on the system of either using a threshold value of `0.05` for either the `Blackouts` or `Unmet energy fraction`, you could use `parameter = 'Threshold criterion'` and `parameter_values = ['Blackouts', 'Unmet energy fraction']`. Or if you wanted to find an optimum systems with the lowest cost of electricity or lowest emissions intensity, you could use `parameter = 'Optimisation criterion'` and `parameter_values = ['LCUE ($/kWh)', 'Emissions intensity (gCO2/kWh)']`. Owing to the variety of potential combinations they have not all been tested so unexpected bugs may occur, in which case please

get in touch.

6.4.2 Interpreting optimisation results

The meaning and implications of your optimisation results will depend very strongly on your scenario and the goals of your investigation: CLOVER can provide results but the value is in interpreting them and applying them in the context of your project. With that in mind, we can use the example in Section 6.4.1 to demonstrate some of the techniques that could be useful.

As stated earlier, the goal of this investigation was to see what effect different levels of reliability (from `Blackouts = 0.25` to `Blackouts = 0.05`) would have on the optimum system design, defined to be the one with the lowest LCUE. Let's open the results we generated previously:

[47]:
```
optimisation_example = Optimisation().open_optimisation(filename = '/Changing␣
↪parameter example/Threshold value lifetime summary of results')
optimisation_example.T.head(len(optimisation_example.T)).round(3)
```

[47]:

	0	1	2	\
Parameter value	0.250	0.200	0.150	
Start year	0.000	0.000	0.000	
End year	12.000	12.000	12.000	
Step length	4.000	4.000	4.000	
Optimisation length	12.000	12.000	12.000	
Maximum PV size	20.000	20.000	25.000	
Maximum storage size	55.000	65.000	70.000	
Maximum diesel capacity	0.000	0.000	0.000	
LCUE ($/kWh)	0.268	0.282	0.301	
Emissions intensity (gCO2/kWh)	444.411	437.315	460.682	
Blackouts	0.226	0.177	0.127	
Unmet fraction	0.161	0.123	0.089	
Renewables fraction	0.471	0.460	0.449	
Storage fraction	0.248	0.275	0.299	
Diesel fraction	0.000	0.000	0.000	
Grid fraction	0.281	0.265	0.252	
Total energy (kWh)	266649.494	279106.252	290770.084	
Total load energy (kWh)	314578.751	314578.751	314578.751	
Unmet energy (kWh)	50503.688	38815.038	27883.654	
Renewable energy (kWh)	125602.499	128275.629	130486.554	
Storage energy (kWh)	66216.550	76869.370	87028.926	
Grid energy (kWh)	74830.445	73961.253	73254.605	
Diesel energy (kWh)	0.000	0.000	0.000	
Discounted energy (kWh)	138250.028	144964.140	150964.432	
Total cost ($)	49206.500	48696.650	50056.010	
Total system cost ($)	37102.430	40824.860	45366.490	
New equipment cost ($)	29446.250	32795.160	36829.580	
New connection cost ($)	834.630	834.630	834.630	
O&M cost ($)	6437.500	6815.780	7325.660	

Diesel cost ($)	0.000	0.000	0.000
Grid cost ($)	384.060	379.300	376.630
Kerosene cost ($)	12104.060	7871.810	4689.520
Kerosene cost mitigated ($)	26442.590	30674.860	33857.150
Kerosene displacement	0.678	0.782	0.865
Diesel fuel usage (l)	0.000	0.000	0.000
Total GHGs (kgCO2eq)	261382.140	218776.770	193499.040
Total system GHGs (kgCO2eq)	118501.880	122057.280	133952.620
O&M GHGs (kgCO2eq)	5500.000	5900.000	6400.000
Diesel GHGs (kgCO2eq)	0.000	0.000	0.000
Grid GHGs (kgCO2eq)	48633.790	48049.120	47629.300
Kerosene GHGs (kgCO2eq)	142880.270	96719.480	59546.420
Kerosene GHGs mitigated (kgCO2eq)	300912.540	347073.330	384246.390

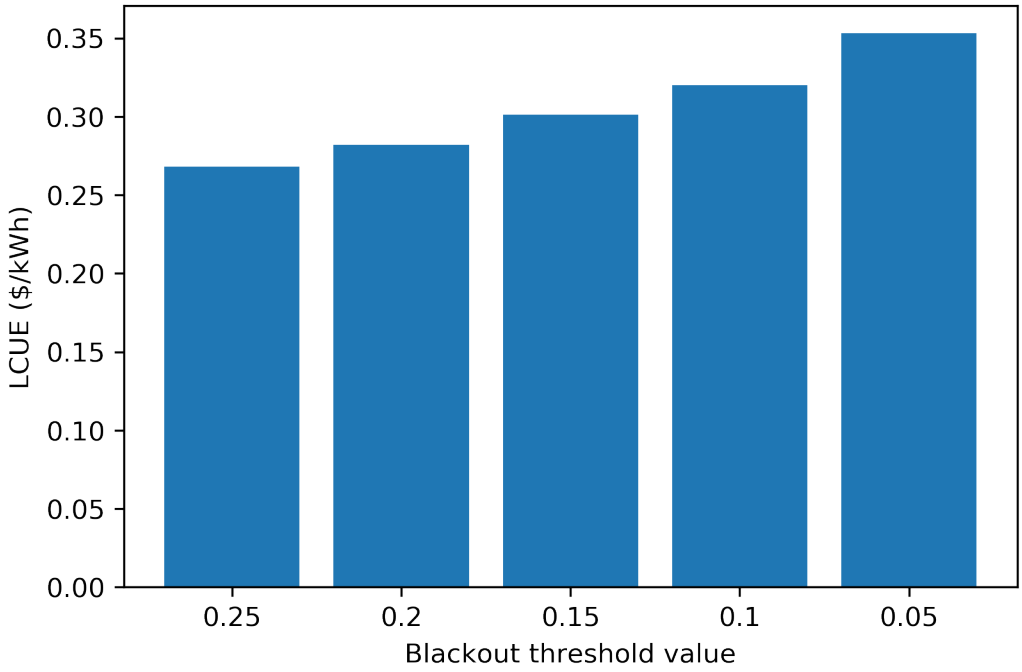
	3	4
Parameter value	0.100	0.050
Start year	0.000	0.000
End year	12.000	12.000
Step length	4.000	4.000
Optimisation length	12.000	12.000
Maximum PV size	25.000	30.000
Maximum storage size	85.000	100.000
Maximum diesel capacity	0.000	0.000
LCUE ($/kWh)	0.320	0.353
Emissions intensity (gCO2/kWh)	452.121	487.441
Blackouts	0.085	0.044
Unmet fraction	0.059	0.029
Renewables fraction	0.434	0.429
Storage fraction	0.323	0.338
Diesel fraction	0.000	0.000
Grid fraction	0.244	0.233
Total energy (kWh)	300815.451	310774.471
Total load energy (kWh)	314578.751	314578.751
Unmet energy (kWh)	18448.425	8970.096
Renewable energy (kWh)	130486.554	133308.542
Storage energy (kWh)	97074.293	105142.003
Grid energy (kWh)	73254.605	72323.927
Diesel energy (kWh)	0.000	0.000
Discounted energy (kWh)	155792.252	160367.390
Total cost ($)	52631.400	58062.030
Total system cost ($)	49799.110	56608.180
New equipment cost ($)	40635.940	46727.450
New connection cost ($)	834.630	834.630
O&M cost ($)	7951.920	8673.780
Diesel cost ($)	0.000	0.000
Grid cost ($)	376.630	372.340
Kerosene cost ($)	2832.290	1453.860

Kerosene cost mitigated ($)	35714.370	37092.810
Kerosene displacement	0.918	0.960
Diesel fuel usage (l)	0.000	0.000
Total GHGs (kgCO2eq)	172452.620	169535.870
Total system GHGs (kgCO2eq)	136005.050	151484.150
O&M GHGs (kgCO2eq)	7000.000	7800.000
Diesel GHGs (kgCO2eq)	0.000	0.000
Grid GHGs (kgCO2eq)	47629.300	47040.540
Kerosene GHGs (kgCO2eq)	36447.560	18051.710
Kerosene GHGs mitigated (kgCO2eq)	407345.240	425741.080

Once again we have displayed the outputs vertically to make them easier to see. The results in this table are characterised from left to right in terms of the `Parameter value`, here our range from 25% to 5% blackouts. An important note here is that *Total* now refers to the lifetime cumulative cost, rather than the final iteration as it was before. The fraction of energy from each source considers only the energy used in the system (i.e. after the `Unmet fraction` has been taken out), and `Renewables fraction` refers to energy from renewable generation used directly, with `Storage fraction` accounted for separately. The differences in naming conventions come from different versions of the model which have been updated and modified over time, so bear this in mind and refer to the documentation when interpreting the outputs.

Let's look at how the different `parameter_values` we used in the function impact the LCUE graphically:

[48]:
```
plt.bar(range(5),optimisation_example['LCUE ($/
 ↪kWh)'],tick_label=optimisation_example['Parameter value'])
plt.xlabel('Blackout threshold value')
plt.ylabel('LCUE ($/kWh)')
plt.show()
```



As expected the value of LCUE, our chosen optimisation criterion, increases for more reliable systems: if the community are willing to pay more for electricity then the most reliable system may be appropriate, but if not then perhaps a less reliable but more affordable system would be better. It is worth noting here that whilst we set `Blackouts` to be the threshold criterion, the actual levels of blackouts in each system was always lower:

[49]:
```
blackouts = optimisation_example[['Parameter value','Blackouts']]
blackouts.head().round(3)
```

[49]:

	Parameter value	Blackouts
0	0.25	0.226
1	0.20	0.177
2	0.15	0.127
3	0.10	0.085
4	0.05	0.044

This is because the actual value of `Blackouts` must always be equal to or lower than the threshold value, but this is worth noting when considering the difference between the criteria we set for a system and its actual performance.

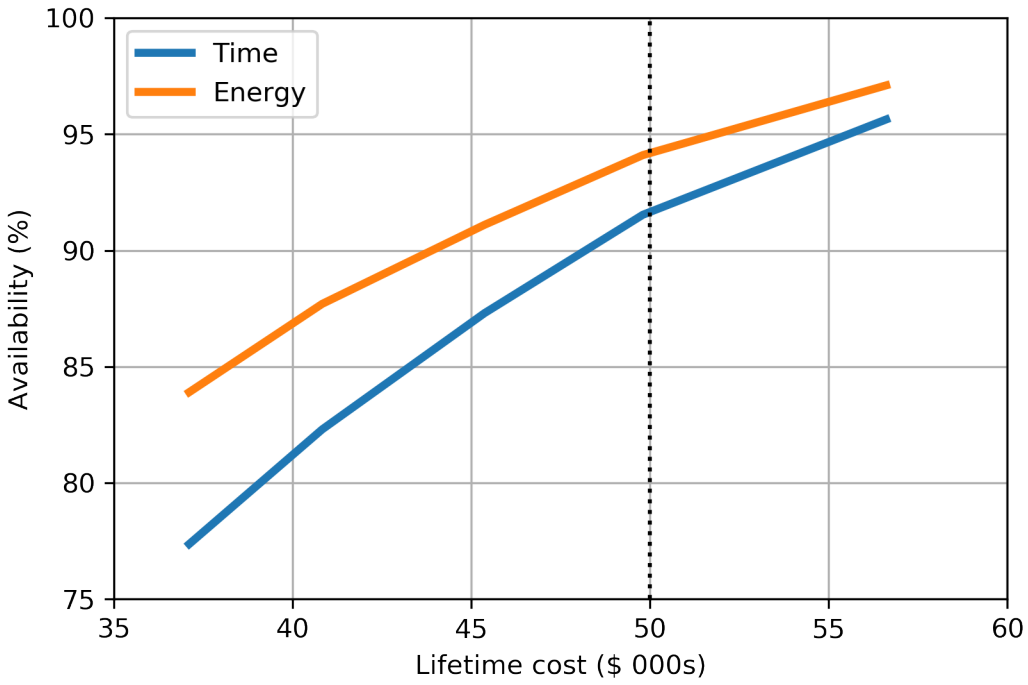
After seeing these results it is also good to evaluate the resolution of the optimisations that we used. Our `Optimisation inputs` CSV used `PV size (step)` as 5 kWp: given that the installed capacities (here shown in the `Maximum PV size` variable) are in the range 20-30 kWp, this step size is likely too large to give us a good resolution and the should probably have been smaller to avoid unnecessary oversizing. The `Storage size (step)` value of 5 kWh was probably a good choice, however, as the storage sizes are all much larger than the step size. If we were looking into

more detail into a given optimisation, or a new set of optimisations, it would be worthwhile re-running the function with a smaller `PV size (step)` variable. This would also likely find systems with the value for `Blackouts` closer to the `parameter_value`. If we were designing a system for implementation, it may also be a good idea to run a final optimisation using a smaller step size for both variables, possibly using the actual capacities of the PV panels and batteries that might be used, to get the most precise answer available.

6.4.3 Incorporating further constraints

Perhaps in addition to wanting to supply electricity at the lowest LCUE, a project developer also has a maximum budget for the lifetime of the project (say $50,000). This could either be a funding constraint or a comparison to some other kind of system, for example using diesel only (rather than solar, storage and the grid as considered here). This time let's plot the results in terms of the lifetime cost of the system against the availability of electricity, both in terms of time (`Blackouts`) and also show the availability of energy demanded (using `Unmet fraction`) for some additional insight:

[50]:
```
plt.plot(optimisation_example['Total system cost ($)']/1000,
         (1-optimisation_example['Blackouts'])*100,
        linewidth=3, label = 'Time')
plt.plot(optimisation_example['Total system cost ($)']/1000,
         (1-optimisation_example['Unmet fraction'])*100,
        linewidth=3, label = 'Energy')
plt.plot([50,50],[75,100], color = 'k', linestyle = ':')
plt.ylabel('Availability (%)')
plt.ylim(75,100)
plt.xlabel('Lifetime cost ($ 000s)')
plt.xlim(35,60)
plt.grid(which='both')
plt.legend()
plt.show()
```



In all cases the availability of energy is greater than the availability in terms of time, suggesting that when blackouts occur they are either at times with lower demand or power is available for at least some of those hours when blackouts occur. It also suggests that using `Blackouts` as the threshold criterion is more strict and will result in a more conservative estimate of reliability.

Here we can see that the maximum budget of $50,000 would correspond to a system with around power available around 92% of the time, and would supply 94% of the energy demanded by the community. This is just an estimate, however, and further optimisations around this point would inform the actual system design and capacities necessary to achieve that performance. Conveniently for us, however, the system found for `Parameter value = 0.10` matches these performances quite closely which hints at the next steps for future optimisations: for example, using `parameter_values = [0.12, 0.11, 0.10, 0.09, 0.08]` to hone in on the best system.

6.4.4 Considering environmental performance

In this case the we designed the systems to meet a minimum level of system performance (`Blackouts`) at the lowest LCUE, and also added a constraint of the total available budget of the project. Many developers also want to know the environmental impact of the projects, both in terms of the emissions intensity of the power provided and the total GHG emissions.

Let's first look at the emissions intensity:

[51]:
```
emissions_intensity = optimisation_example[['Parameter value','Emissions␣
↪intensity (gCO2/kWh)']]
emissions_intensity.head().round(3)
```

[51]:

	Parameter value	Emissions intensity (gCO2/kWh)
0	0.25	444.411
1	0.20	437.315
2	0.15	460.682
3	0.10	452.121
4	0.05	487.441

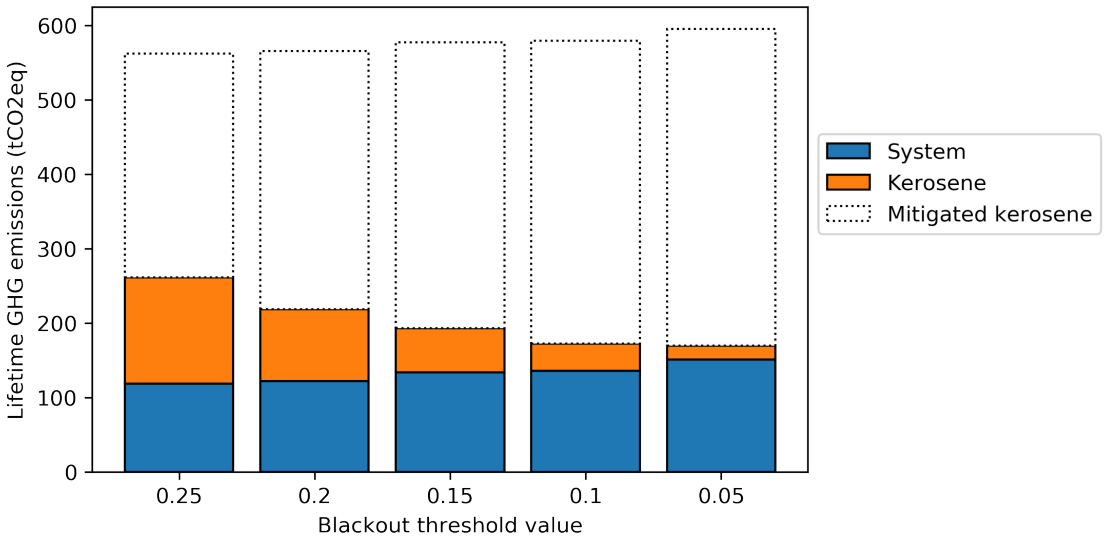
It seems that there is not an obvious linearity here, but there are several factors at play. The systems use approximately the same amount of high-carbon grid electricity, but the higher-reliability systems have a greater share of energy coming from renewable sources which will decrease the average emissions intensity. At higher reliabilities, however, there is a diminishing return on investment for the equipment as more equipment (and hence more embedded emissions) are necessary for the same incremental gains. Let's compare to the emissions intensity of the grid:

[52]:
```
print(GHG_inputs.loc[18:19])
```

```
                       0    1          2
18  Grid GHGs (initial)  0.8  kgCO2/kWh
19    Grid GHGs (final)  0.4  kgCO2/kWh
```

Over the lifetime of the system the average intensity of the grid is around 600 gCO2/kWh, but for all of our systems it is between 430-490 gCO2/kWh, so overall these systems perform quite favourably - especially bearing in mind that some of the 23-28% of their energy comes from the grid (`Grid fraction`). Diesel-only systems can have emissions intensities around 1000 gCO2/kWh, so these systems seem like a good option. Let's also consider the total GHG emissions from each source:

[53]:
```
plt.bar(range(5),optimisation_example['Total system GHGs (kgCO2eq)'],
       edgecolor='k',label='System')
plt.bar(range(5),optimisation_example['Kerosene GHGs (kgCO2eq)'],
        bottom = optimisation_example['Total system GHGs (kgCO2eq)'],
       edgecolor='k',label='Kerosene')
plt.bar(range(5),optimisation_example['Kerosene GHGs mitigated (kgCO2eq)'],
        bottom = optimisation_example['Total system GHGs␣
 ↪(kgCO2eq)']+optimisation_example['Kerosene GHGs (kgCO2eq)'],
       facecolor='w',edgecolor='k',linestyle = ':',label='Mitigated kerosene')
plt.legend(bbox_to_anchor=(1.0, 0.75))
plt.xticks(range(5),optimisation_example['Parameter value'])
plt.xlabel('Blackout threshold value')
plt.yticks(range(0,600001,100000),range(0,601,100))
plt.ylabel('Lifetime GHG emissions (tCO2eq)')
plt.show()
```



The GHGs associated with the system increase as reliability increases, which as discussed earlier is inherent in larger systems with greater equipment requirements. In this case, however, the impact of kerosene usage has a significant effect: kerosene for lighting when electricity is unavailable is very common in our case study location of Bahraich, and by achieving higher levels of reliability our systems can offset larger amounts of kerosene usage in the evenings. Less reliable systems, however, provide less availability in the evening periods and so kerosene usage remains relatively prevalent; this is also captured in the `Kerosene displacement` variable.

Here we can see that systems with increasing reliability lower total GHG emissions over their lifetimes, but this diminishes between the systems with `Parameter value = 0.1` and `0.05`. Given the budgetary constraints discussed in Section 6.4.3, let's assume that the system with `Parameter value = 0.1` fits our environmental needs by significantly reducing GHGs from existing kerosene use whilst also staying within an appropriate budget for the project and therefore select this system for deployment.

6.4.5 Returning to system performance

Now that we've selected the system with `Parameter value = 0.1` as the one to be installed we should investigate its technical performance to make sure that it will meet the needs of the community. Whilst this has already been assessed during the optimisation process, this will allow us to dive deeper into its operation to identify any potentially unexpected results.

To do this we first use a function we mentioned earlier to open the performance of the system from its CSV file:

[54]:
```
chosen_system = Optimisation().open_optimisation(filename = '/Changing parameter␣
↪example/Threshold value = 0.10')
```

Note that here we must also specify the folder the file is stored in (`Changing parameter optimisation`) as the function looks in the `Saved optimisations` folder by default. Next we can

use a function in the Energy_System module to run a simulation for this chosen system, automatically accounting for the equipment upgrades:

[55]:
```
chosen_system_simulation = Energy_System().lifetime_simulation(chosen_system)
chosen_system_simulation.head(24).round(3)
```

```
Time taken for simulation: 0.15 seconds per year

Time taken for simulation: 0.17 seconds per year

Time taken for simulation: 0.14 seconds per year
```

[55]:

	Load energy (kWh)	Total energy used (kWh)	Unmet energy (kWh)	Blackouts	\
0	1.166	1.166	0.0	0.0	
1	0.938	0.938	0.0	0.0	
2	0.920	0.968	0.0	0.0	
3	0.377	0.377	0.0	0.0	
4	0.402	0.423	0.0	0.0	
5	0.412	0.412	0.0	0.0	
6	0.446	0.470	0.0	0.0	
7	1.258	1.320	0.0	0.0	
8	1.479	1.479	0.0	0.0	
9	1.300	1.300	0.0	0.0	
10	0.726	0.726	0.0	0.0	
11	1.668	1.668	0.0	0.0	
12	1.105	1.105	0.0	0.0	
13	1.100	1.100	0.0	0.0	
14	1.947	1.947	0.0	0.0	
15	2.226	2.226	0.0	0.0	
16	1.479	1.479	0.0	0.0	
17	3.626	3.626	0.0	0.0	
18	2.001	2.106	0.0	0.0	
19	4.447	4.681	0.0	0.0	
20	1.961	2.064	0.0	0.0	
21	1.576	1.576	0.0	0.0	
22	2.226	2.343	0.0	0.0	
23	2.038	2.145	0.0	0.0	

	Renewables energy used (kWh)	Storage energy supplied (kWh)	\
0	0.000	1.166	
1	0.000	0.000	
2	0.000	0.968	
3	0.000	0.000	
4	0.000	0.423	
5	0.000	0.000	
6	0.000	0.470	
7	0.081	1.239	

8	1.479	0.000
9	1.300	0.000
10	0.726	0.000
11	1.668	0.000
12	1.105	0.000
13	1.100	0.000
14	1.947	0.000
15	2.226	0.000
16	1.479	0.000
17	0.153	0.000
18	0.000	2.106
19	0.000	4.681
20	0.000	2.064
21	0.000	0.000
22	0.000	2.343
23	0.000	2.145

	Grid energy (kWh)	Diesel energy (kWh)	Diesel times	\
0	0.000	0.0	0.0	
1	0.938	0.0	0.0	
2	0.000	0.0	0.0	
3	0.377	0.0	0.0	
4	0.000	0.0	0.0	
5	0.412	0.0	0.0	
6	0.000	0.0	0.0	
7	0.000	0.0	0.0	
8	0.000	0.0	0.0	
9	0.000	0.0	0.0	
10	0.000	0.0	0.0	
11	0.000	0.0	0.0	
12	0.000	0.0	0.0	
13	0.000	0.0	0.0	
14	0.000	0.0	0.0	
15	0.000	0.0	0.0	
16	0.000	0.0	0.0	
17	3.473	0.0	0.0	
18	0.000	0.0	0.0	
19	0.000	0.0	0.0	
20	0.000	0.0	0.0	
21	1.576	0.0	0.0	
22	0.000	0.0	0.0	
23	0.000	0.0	0.0	

	Diesel fuel usage (l)	Storage profile (kWh)	\
0	0.0	-1.166	
1	0.0	0.000	
2	0.0	-0.920	

3	0.0	0.000
4	0.0	-0.402
5	0.0	0.000
6	0.0	-0.446
7	0.0	-1.177
8	0.0	0.182
9	0.0	2.481
10	0.0	4.607
11	0.0	4.532
12	0.0	5.510
13	0.0	5.190
14	0.0	3.413
15	0.0	1.537
16	0.0	0.254
17	0.0	0.000
18	0.0	-2.001
19	0.0	-4.447
20	0.0	-1.961
21	0.0	0.000
22	0.0	-2.226
23	0.0	-2.038

	Renewables energy supplied (kWh)	Hourly storage (kWh)	\
0	0.000	30.334	
1	0.000	30.212	
2	0.000	29.123	
3	0.000	29.007	
4	0.000	28.467	
5	0.000	28.353	
6	0.000	27.770	
7	0.081	26.421	
8	1.661	26.487	
9	3.781	28.739	
10	5.334	31.499	
11	6.200	31.499	
12	6.615	31.499	
13	6.290	31.499	
14	5.361	31.499	
15	3.763	31.499	
16	1.733	31.499	
17	0.153	31.373	
18	0.000	29.141	
19	0.000	24.343	
20	0.000	22.181	
21	0.000	22.093	
22	0.000	19.661	
23	0.000	17.437	

	Dumped energy (kWh)	Battery health	Households	Kerosene lamps	Kerosene mitigation
0	0.000	1.0	100	0.0	75.0
1	0.000	1.0	100	0.0	103.0
2	0.000	1.0	100	0.0	81.0
3	0.000	1.0	100	0.0	91.0
4	0.000	1.0	100	0.0	74.0
5	0.000	1.0	100	0.0	73.0
6	0.000	1.0	100	0.0	70.0
7	0.000	1.0	100	0.0	0.0
8	0.000	1.0	100	0.0	0.0
9	0.000	1.0	100	0.0	0.0
10	1.502	1.0	100	0.0	0.0
11	4.179	1.0	100	0.0	0.0
12	5.108	1.0	100	0.0	0.0
13	4.805	1.0	100	0.0	0.0
14	3.117	1.0	100	0.0	0.0
15	1.334	1.0	100	0.0	0.0
16	0.115	1.0	100	0.0	0.0
17	0.000	1.0	100	0.0	122.0
18	0.000	1.0	100	0.0	333.0
19	0.000	1.0	100	0.0	
20	0.000	1.0	100	0.0	
21	0.000	1.0	100	0.0	
22	0.000	1.0	100	0.0	
23	0.000	1.0	100	0.0	

```
19                  338.0
20                  336.0
21                  277.0
22                  132.0
23                  108.0
```

Here we have once again displayed the first day of data from this simulation output, similarly to Section 5.2.3 and notice that the output of `Energy_System().lifetime_simulation(...)` gives a single output of the system performance, rather than the additional second output describing the makeup of that system (as we already know it from the input we used).

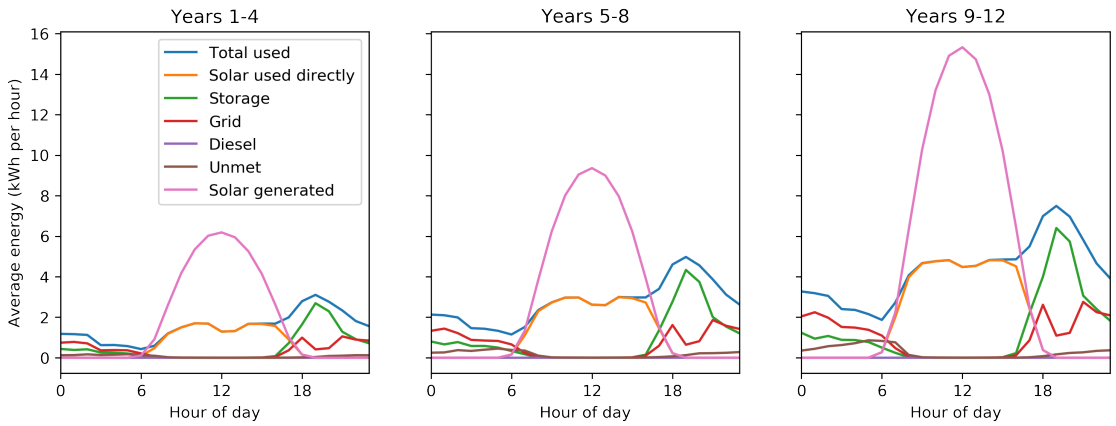
As an aside, it is also possible to perform the reverse process that we have just used: we can generate the technical, financial and environmental outputs of a given simulation using the function `Optimisation().system_appraisal(chosen_system_simulation)`. This function takes the (two-component) variable `chosen_system_simulation` as an input, and returns the same format of appraisal results as is generated by the standard optimisation processes. This could be useful, for example, when assessing the performance of a specific case study system which was not generated through the optimisation process.

Let's look at the energy use on an average day for each of the three iteration periods, plotted at the same scale:

[56]:
```
fig,([ax1,ax2,ax3]) = plt.subplots(1,3,sharex=True, sharey=True,figsize=(12,4))

#Iteration period 1
iteration_length = 365 * 4
start_hour = 0 * (iteration_length * 24)
end_hour = 1 * (iteration_length * 24) - 1
iteration_performance = chosen_system_simulation.loc[start_hour:end_hour]
total_used = np.mean(np.reshape(iteration_performance['Total energy used (kWh)'].
 ↪values,(iteration_length,24)),axis=0)
renewable_energy = np.mean(np.reshape(iteration_performance['Renewables energy␣
 ↪used (kWh)'].values,(iteration_length,24)),axis=0)
storage_energy = np.mean(np.reshape(iteration_performance['Storage energy␣
 ↪supplied (kWh)'].values,(iteration_length,24)),axis=0)
grid_energy = np.mean(np.reshape(iteration_performance['Grid energy (kWh)'].
 ↪values,(iteration_length,24)),axis=0)
diesel_energy = np.mean(np.reshape(iteration_performance['Diesel energy (kWh)'].
 ↪values,(iteration_length,24)),axis=0)
unmet_energy = np.mean(np.reshape(iteration_performance['Unmet energy (kWh)'].
 ↪values,(iteration_length,24)),axis=0)
renewables_supplied = np.mean(np.reshape(iteration_performance['Renewables␣
 ↪energy supplied (kWh)'].values,(iteration_length,24)),axis=0)

ax1.plot(total_used, label = 'Total used')
ax1.plot(renewable_energy, label = 'Solar used directly')
ax1.plot(storage_energy, label = 'Storage')
ax1.plot(grid_energy, label = 'Grid')
```

```
ax1.plot(diesel_energy, label = 'Diesel')
ax1.plot(unmet_energy, label = 'Unmet')
ax1.plot(renewables_supplied, label = 'Solar generated')
ax1.set(xticks = range(0,24,6),
        xlim=(0,23),
        xlabel = 'Hour of day',
        ylabel = 'Average energy (kWh per hour)',
        title = 'Years 1-4')
ax1.legend()

#Iteration period 2
iteration_length = 365 * 4
start_hour = 1 * (iteration_length * 24)
end_hour = 2 * (iteration_length * 24) - 1
iteration_performance = chosen_system_simulation.loc[start_hour:end_hour]
total_used = np.mean(np.reshape(iteration_performance['Total energy used (kWh)'].
 ↪values,(iteration_length,24)),axis=0)
renewable_energy = np.mean(np.reshape(iteration_performance['Renewables energy␣
 ↪used (kWh)'].values,(iteration_length,24)),axis=0)
storage_energy = np.mean(np.reshape(iteration_performance['Storage energy␣
 ↪supplied (kWh)'].values,(iteration_length,24)),axis=0)
grid_energy = np.mean(np.reshape(iteration_performance['Grid energy (kWh)'].
 ↪values,(iteration_length,24)),axis=0)
diesel_energy = np.mean(np.reshape(iteration_performance['Diesel energy (kWh)'].
 ↪values,(iteration_length,24)),axis=0)
unmet_energy = np.mean(np.reshape(iteration_performance['Unmet energy (kWh)'].
 ↪values,(iteration_length,24)),axis=0)
renewables_supplied = np.mean(np.reshape(iteration_performance['Renewables␣
 ↪energy supplied (kWh)'].values,(iteration_length,24)),axis=0)

ax2.plot(total_used, label = 'Total used')
ax2.plot(renewable_energy, label = 'Solar used directly')
ax2.plot(storage_energy, label = 'Storage')
ax2.plot(grid_energy, label = 'Grid')
ax2.plot(diesel_energy, label = 'Diesel')
ax2.plot(unmet_energy, label = 'Unmet')
ax2.plot(renewables_supplied, label = 'Solar generated')
ax2.set(xticks = range(0,24,6),
        xlim=(0,23),
        xlabel = 'Hour of day',
        title = 'Years 5-8')

#Iteration period 3
iteration_length = 365 * 4
start_hour = 2 * (iteration_length * 24)
end_hour = 3 * (iteration_length * 24) - 1
```

```
iteration_performance = chosen_system_simulation.loc[start_hour:end_hour]
total_used = np.mean(np.reshape(iteration_performance['Total energy used (kWh)'].
 ↪values,(iteration_length,24)),axis=0)
renewable_energy = np.mean(np.reshape(iteration_performance['Renewables energy␣
 ↪used (kWh)'].values,(iteration_length,24)),axis=0)
storage_energy = np.mean(np.reshape(iteration_performance['Storage energy␣
 ↪supplied (kWh)'].values,(iteration_length,24)),axis=0)
grid_energy = np.mean(np.reshape(iteration_performance['Grid energy (kWh)'].
 ↪values,(iteration_length,24)),axis=0)
diesel_energy = np.mean(np.reshape(iteration_performance['Diesel energy (kWh)'].
 ↪values,(iteration_length,24)),axis=0)
unmet_energy = np.mean(np.reshape(iteration_performance['Unmet energy (kWh)'].
 ↪values,(iteration_length,24)),axis=0)
renewables_supplied = np.mean(np.reshape(iteration_performance['Renewables␣
 ↪energy supplied (kWh)'].values,(iteration_length,24)),axis=0)

ax3.plot(total_used, label = 'Total used')
ax3.plot(renewable_energy, label = 'Solar used directly')
ax3.plot(storage_energy, label = 'Storage')
ax3.plot(grid_energy, label = 'Grid')
ax3.plot(diesel_energy, label = 'Diesel')
ax3.plot(unmet_energy, label = 'Unmet')
ax3.plot(renewables_supplied, label = 'Solar generated')
ax3.set(xticks = range(0,24,6),
        xlim=(0,23),
        xlabel = 'Hour of day',
        title = 'Years 9-12')

plt.show()
```

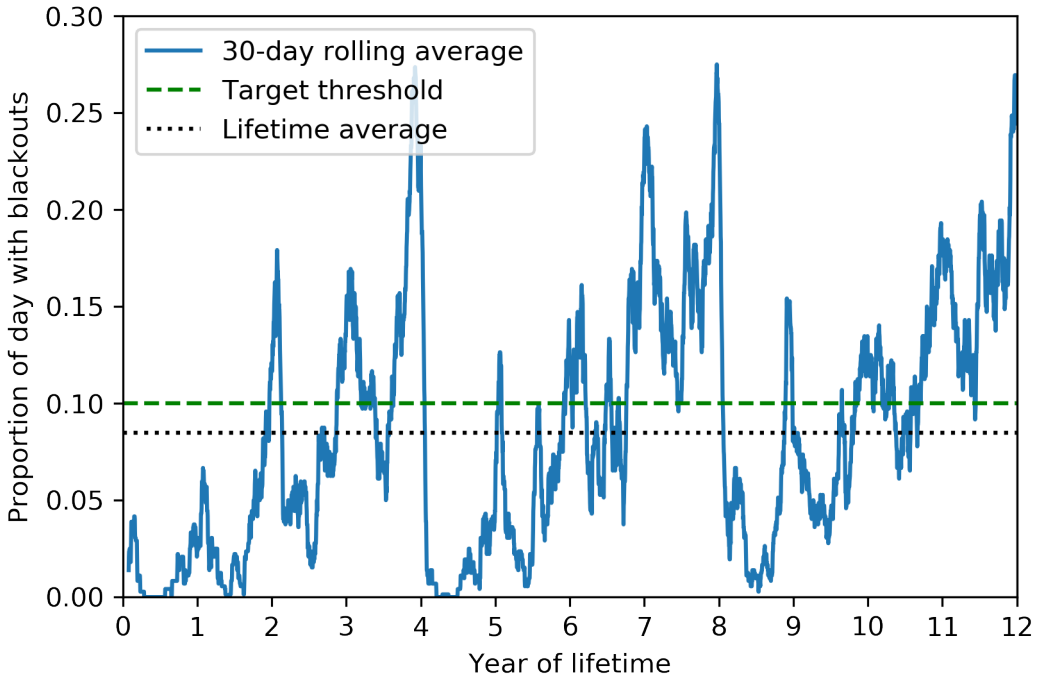


We can observe here the growing energy generation and usage across the three iteration periods and some minor variations in the relative usage of each source of power at different points dur-

ing the day, but in general the performance seems quite similar across the three iteration periods over the lifetime of the system. If we had further issues to consider, for example if the load profile changed in shape over time as well as increasing, we might need to investigate the different iteration periods in more detail.

Now let's look at the availability of energy, in terms of the average proportion of blackouts per day, over the lifetime of the system:

[57]:
```
blackouts = pd.DataFrame(chosen_system_simulation['Blackouts'])
daily_blackout_mean = np.mean(np.reshape(np.mean(blackouts[0:365*24*12],axis=1).
 ↪values,(365*12,24)),axis=1)
lifetime_mean = np.mean(daily_blackout_mean)
plt.figsize=(12,4)

plt.plot(range(365*12),pd.DataFrame(daily_blackout_mean).rolling(30).mean(),␣
 ↪label = '30-day rolling average')
plt.plot([0,len(daily_blackout_mean)],[0.1,0.1],
         color = 'g', linestyle = '--', label = 'Target threshold')

plt.plot([0,len(daily_blackout_mean)],[lifetime_mean,lifetime_mean],
         color = 'k', linestyle = ':', label = 'Lifetime average')

plt.ylim(0,0.3)
plt.xlim(0,365*12)
plt.xticks(range(0,len(daily_blackout_mean)+1,365),range(13))
plt.legend()
plt.xlabel('Year of lifetime')
plt.ylabel('Proportion of day with blackouts')
plt.show()
```



To remove some of the inherent variability between days we present the data in terms of a 30-day rolling average. As we can see, the availability of power varies quite significantly over the lifetime and each iteration period: as the load grows, the ability of the system to maintain the desired threshold decreases and so blackouts increase around halfway through each iteration period. There is also a seasonal effect, where blackouts occur more frequently in the winter months. Let's look at the average blackouts for each year of the lifetime:

[58]:
```
blackouts = pd.DataFrame(chosen_system_simulation['Blackouts'])
print('Year | Mean | Standard deviation')
for year in range(12):
    start_hour = year * 365 * 24
    end_hour = (year+1) * 365 * 24 - 1
    yearly_blackouts = blackouts.loc[start_hour:end_hour]
    print('Year '+str(year+1)+': '+str(np.mean(yearly_blackouts)['Blackouts'].
 ↪round(2))
          +str(' +/- ')+str(np.std(yearly_blackouts)['Blackouts'].round(2)))
print('Lifetime average: '+str(np.mean(blackouts)['Blackouts'].round(2))
      +str(' +/- ')+str(np.std(blackouts)['Blackouts'].round(2)))
```

```
Year | Mean | Standard deviation
Year 1: 0.01 +/- 0.11
Year 2: 0.04 +/- 0.19
Year 3: 0.07 +/- 0.26
Year 4: 0.14 +/- 0.35
Year 5: 0.01 +/- 0.12
```

```
Year 6: 0.06 +/- 0.24
Year 7: 0.11 +/- 0.32
Year 8: 0.17 +/- 0.37
Year 9: 0.05 +/- 0.21
Year 10: 0.07 +/- 0.26
Year 11: 0.12 +/- 0.32
Year 12: 0.17 +/- 0.37
Lifetime average: 0.08 +/- 0.28
```

The first two years in each iteration period (1-2, 5-6, 9-10) perform better than expected, but the final years (4, 8, 12) all have far more blackouts than our threshold value (`0.10`) or the lifetime average. These later years also have larger standard deviations, which means that some days have more variable performances, also visible in the figure above.

This could pose a problem for system operators: although customers will receive an adequate service on average, during the winter months or the later years they might not pay their monthly bills (for example) if their service has not met the levels promised to them. This will affect the financial returns of the project, as well as potentially influence community perception of the system.

If these periods of reduced availability will be an issue we have a few options to improve the system design. The first would be to re-optimise the system using a more restrictive threshold criterion: this will likely lower both the mean for each iteration period and the lifetime average, but the same changing profile will probably still be present (albeit exceeding the initial threshold of `Blackouts = 0.10` less frequently). Another option would be to shorten the lifetime period, allowing the system to be resized more frequently to better account for the growing load profile and have a more consistent incidence of blackouts. Finally in practice it may be pragmatic to install the system as designed here but monitor its performance: if the load profile grows over time as expected the system could be revisited earlier than expected to maintain the desired performance, or if not then new equipment could be installed later.

6.4.6 Practical considerations

CLOVER is designed for users to model community electricity systems, simulate their performance, and identify optimum configurations based on the inputs they choose. The simulation and optimisation functions, and CLOVER in general, will provide results which can inform system design for practical deployment but should always be treated with appropriate caution. The confidence you have in your input data, for example, can be used as a good judge of the confidence you have in the results of using CLOVER.

This is not to say that all of the modelling is purely theoretical: CLOVER should be used as a tool to help inform research or practical system deployment in the context of many other activities such as community engagement, political priorities, business model development and technical equipment specifications. These practical considerations are sometimes impossible to quantify and so should be dealt with separately to ensure they are properly accounted for as part of a project, and should inform (and be informed by) modelling in CLOVER wherever possible.

The greatest value that CLOVER can bring to a project is its ability to (relatively) quickly and easily explore many different scenarios and provide results on the design and performance of systems that can be explored in detail. Questions around providing different levels of service

to a community, using different types of energy technologies, and under different load growth scenarios and grid availabilities can all be answered using CLOVER, with the results considering more than just the cost of electricity but many other performance, financial and environmental metrics to best understand the implications of a given system.

Once CLOVER has been used to identify a system which would be a good candidate for deployment then the considerations around the equipment to be installed, the topography of the distribution network, the customers to be connected, the payment structures, the overall business model and many other practical points can be brought in to interrogate the appropriateness of the proposed system. These are beyond the scope and capabilities of CLOVER and will rely on the expertise of the user or their team to best identify how to implement a system which can provide reliable, affordable and sustainable electricity to the communities they aim to serve.

7 CLOVER development

7.1 Improvements to CLOVER

CLOVER is under continuous development to improve its functionality and capabilities. These will be added to the CLOVER repository on GitHub once they have been finalised and tested, as well as any existing or new bugs that are identified in the code. Check back in at the repository to see any new developments to the code and bug fixes, recorded as commits and in the update register, to ensure your version of CLOVER is up to date.

7.2 Acknowledgements and support for CLOVER

CLOVER has been under development since 2015 and has received a support from a variety of funding sources. We would like to gratefully acknowledge the support of the Grantham Institute - Climate Change and the Environment for contributions to PhD scholarships, Climate-KIC for research allowances, the Engineering and Physical Sciences Research Council (EP/R511547/1 and EP/R030235/1), the ClimateWorks Foundation, and Research England GCRF QR Funding.

Philip Sandwell would like to thank Hamish Beath, Javier Baranda Alonso, and Scot Wheeler, under the supervision of Ned Ekins-Daukes and Jenny Nelson, for their contributions to developing the CLOVER code and functionality over the years. In addition he would like to thank Oytun Babacan, Sheridan Few, Paloma Ortega Arriaga and Anaïs Matthey-Junod for their comments and suggestions to improve this user manual.

7.3 Get involved

If you would like to be involved with the development of CLOVER, have suggestions for improvements to its functionality, or identify (or solve) any bugs, please get in touch or submit pull requests on GitHub. CLOVER is entirely open-source and so users are encouraged to use the code, make modifications, and develop improvements to suit their unique purposes under the share-and-share-alike conditions of the license. We are always happy to hear about how CLOVER has been used!