<u>SCORES: User Guide</u>

Constance Crozier ~ August 2020

This document contains documentation and example code designed to accompany the SCORES repository.

0 Getting Started

The following packages are necessary prerequisites: numpy, matplotlib, csv, copy, datetime, scipy.optimise, pyDOE, os, mpl_toolkits.basemap (for maps only).

The code in the repository does not contain the raw data necessary to run the model, but it does contain saved results from the UK which allow some analysis to be done. In order to get the full use of the model, or to study systems outside of the UK, hourly weather observations need to be added to the data folder.

**something about how to get the NASA data

1 Generation Models

Each generation model object describes a form of power generation, which has an associated hourly power output.

1.1 Classes

1.1.1 Base Class

GenerationModel(sites, year_min, year_max, months, fixed_cost, variable_cost, name, data_path, save_path)

Note that there are two options for initialising a generation mode: in select cases you can load a previous run of the model (providing it has been stored in the save path), otherwise you will need to run the model using raw weather data. For this reason data_path is technically an optional parameter, but if the simulation has not been previously stored, then it is required.

Parameter	Type	Description
sites	*Array-like*	*List of chosen site numbers, or the string 'all'*
year_min	*int*	*Lowest year to be included in the simulation*
year_max	*int*	*Highest year to be included in the simulation*
months	*Array-like*	*List of months to be included in the simulation (1-12)*

fixed_cost	*float*	*Cost incurred per MW-year of installation in GBP*
variable_cost	*int*	*Cost incurred per MWh of generation in GBP*
name	*str*	*Name of generator - used for graph plotting*
data_path	*str*	*Path to folder where the weather data is stored*
save_path	*str*	*Path to folder where model output will be stored if desired*

1.1.2 Offshore Wind

OffshoreWindModel(sites='all', year_min=2013, year_max=2019, months=list(range(1, 13)), fixed_cost=240000, variable_cost=3, tilt=5, air_density=1.23, rotor_diameter=190, rated_rotor_rpm=10, rated_wind_speed=11.5, v_cut_in=4, v_cut_out=30, n_turbine=None, turbine_size=10, data_path='', save_path='stored_model_runs/', save=True)

Parameter	**Type**	**Description**
tilt	*float*	*Blade tilt in degrees*
air_density	*float*	*Density of air in kg/m3*
rotor_diameter	*float*	*Rotor diameter in m*
rated_rotor_rpm	*float*	*Rated rotation speed in rpm*
rated_wind_speed	*float*	*Rated wind speed in m/s*
v_cut_in	*float*	*Cut in wind speed in m/s*
v_cut_out	*float*	*Cut out wind speed in m/s*
n_turbine	*Array-like*	*Relative number of turbines installed at each site, defaults to an even distribution across sites*
turbine_size	*float*	*Size of each turbine in MW*
save	*boo*	*Determines whether to save the results of the run*

1.1.4 Onshore Wind

OnshoreWindModel(sites='all', year_min=2013, year_max=2019, months=list(range(1, 13)), fixed_cost=120000, variable_cost=6, tilt=5, air_density=1.23, rotor_diameter=120, rated_rotor_rpm=13, rated_wind_speed=12.5, v_cut_in=3, v_cut_out=25, n_turbine=None, turbine_size=3.6, hub_height=90, data_path='', save_path='stored_model_runs/', save=True)

Parameter	Type	Description
tilt	*float*	*Blade tilt in degrees*
air_density	*float*	*Density of air in kg/m3*
rotor_diameter	*float*	*Rotor diameter in m*
rated_rotor_rpm	*float*	*Rated rotation speed in rpm*
rated_wind_speed	*float*	*Rated wind speed in m/s*
v_cut_in	*float*	*Cut in wind speed in m/s*
v_cut_out	*float*	*Cut out wind speed in m/s*
n_turbine	*Array-like*	*Relative number of turbines installed at each site, defaults to an even distribution across sites*
turbine_size	*float*	*Size of each turbine in MW*
hub_height	*float*	*Height of turbine hub - needed to adjust the wind speed data.*
save	*boo*	*Determines whether to save the results of the run*

1.1.4 Solar PV

SolarModel(sites='all', year_min=2013, year_max=2019, months=list(range(1, 13)), fixed_cost=42000, variable_cost=0, orient=0, tilt=22, efficiency=0.17, performance_ratio=0.85, plant_capacity=1, area_factor=5.84, data_path='', save_path='stored_model_runs/', save=True)

Parameter	Type	Description
orient	*float*	*Surface azimuth angle in degrees*
tilt	*float*	*Panel tilt in degrees*
efficiency	*float*	*Panel efficiency (0-1)*
performance_ratio	*float*	*Panel performance ratio - determined analytically (0-1)*
plant_capacity	*float*	*Installed capacity in MW*
area_factor	*float*	*Panel area per installed kW in m2/kW*
save	*boo*	*Determines whether to save the results of the run*

1.2 Functions

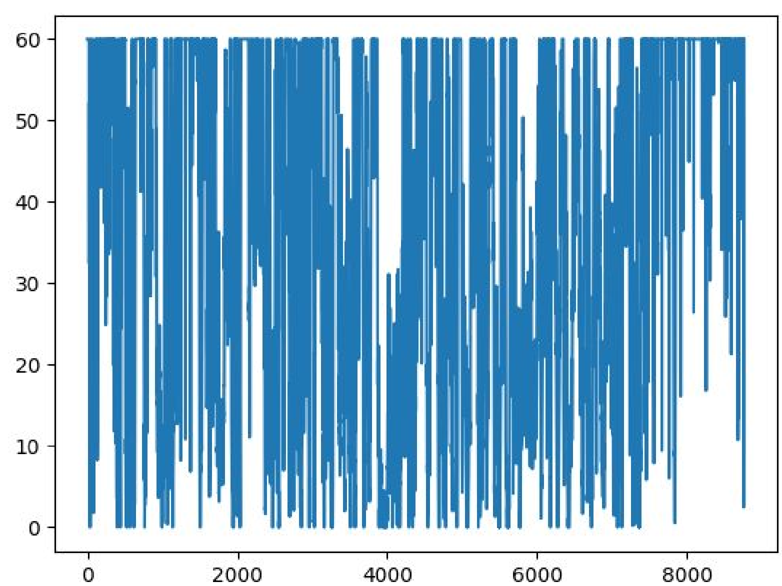
1.2.1 Running the model

run()

This will populate the object's power_out parameter, either by loading a result from save_path, or by running the relevant model using the data located in data_path. This is called during initialisation of an object if a stored model run is not available.

Example: The following plots the output of six 10 MW offshore wind turbines for 2015, one each at sites 1 and 2, and four at site 3.

```
from generation import OffshoreWindModel
import matplotlib.pyplot as plt

gen = OffshoreWindModel(sites=[1,2,3], year_min=2015, year_max=2015,
turbine_size=10, n_turbine=[1,1,4], data_path='data/offshore/')

plt.plot(gen.power_out)

plt.show()

[out]:
```



1.2.2 Load factor calculation

get_load_factor()

This will return the load factor in percent (0-100) of the predicted output power vector.

Example: The following calculates the aggregate load factor of solar stations uniformly distributed across the available locations between 2013-19.

```
from generation import SolarModel

gen = SolarModel(sites='all',year_min=2013, year_max=2019,
data_path='data/solar/')

print(gen.get_load_factor())

[out]:
      10.791703612581438
```

1.2.4 Scaling the amount of installed generation

scale_output(installed_capacity)

This will return an array of the hourly average output over a 24 hour period.

Parameter	Type	Description
installed_capacity	*float*	*Aggregated installed capacity in MW*

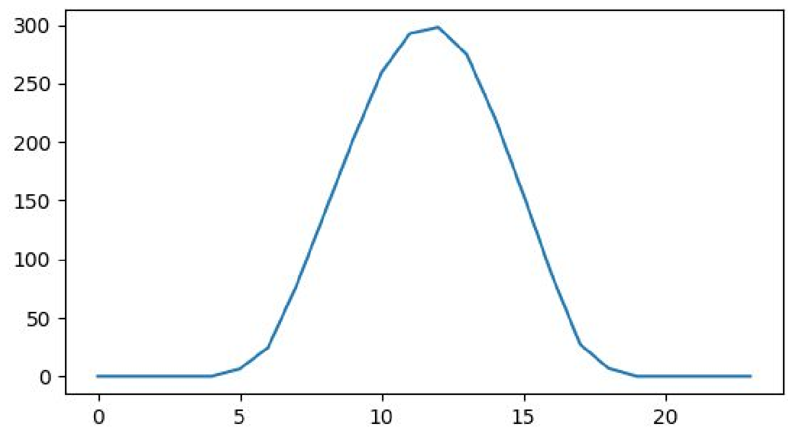
1.2.4 Getting the average daily output curve

get_dirunal_profile()

This will return an array containing the hourly average output over a 24 hour period.

Example: Plotting the average daily output of 800 MW of solar, evenly distributed across all sites, loading from a saved run of 2013-2019.

```
from generation import SolarModel
import matplotlib.pyplot as plt

gen = SolarModel()

gen.scale_output(800)

plt.plot(gen.get_diurnal_profile())

plt.show()

[out]:
```



2 Individual Storage Models

2.1 Classes

2.1.1 Base Class

StorageModel(eff_in, eff_out, self_dis, variable_cost, fixed_cost, max_c_rate, max_d_rate, name, capacity=1)

Parameter	Type	Description
eff_in	*float*	*Charging efficiency in % (0-100)*
eff_out	*float*	*Discharging efficiency in % (0-100)*
self_dis	*float*	*Rate of self discharge in %/month (0-100)*
fixed_cost	*float*	*Cost incurred per MWh-year of installed capacity in GBP*
variable_cost	*float*	*Cost incurred per MWh of storage throughput in GBP*
name	*str*	*Name of storage - used for graph plotting*
max_c_rate	*float*	*Maximum charging rate in %/hr (0-100)*
max_d_rate	*float*	*Maximum discharging rate in %/hr (0-100)*
capacity	*float*	*Installed storage capacity in MWh*

2.1.2 Li-Ion Battery Storage

BatteryStorageModel(eff_in=95, eff_out=95, self_dis=2, variable_cost=0, fixed_cost=16000, max_c_rate=100, max_d_rate=100, capacity=1)

2.1.3 Hydrogen Storage

HydrogenStorageModel(eff_in=67, eff_out=56, self_dis=0, variable_cost=42.5, fixed_cost=120, max_c_rate=0.032, max_d_rate=0.15, capacity=1)

2.1.4 Pumped Thermal Storage

ThermalStorageModel(eff_in=80, eff_out=47, self_dis=9.66,variable_cost=331.6, fixed_cost=773.5, max_c_rate=8.56, max_d_rate=6.82, capacity=1)

2.2 Functions

2.2.1 Running a charging simulation

charge_sim(surplus, t_res=1, return_output=False, start_up_time=0)

This simulates the opportunistic operation of the storage to remove the negative values in the input array 'surplus'. It returns the percentage of times when a negative surplus is avoided after operation of the storage and, if requested, the output vector after the storage has been used.

Parameter	Type	Description
surplus	*Array like*	*The generation net demand which is to be smoothed in MW*
t_res	*float*	*The time resolution in hours of the surplus provided*
return_output	*boo*	*Whether to also return the smoothed output vector*
start_up_time	*int*	*The number of first time intervals to be ignored in reliability calculation*

Example: get the reliability with which an 3.6 MW onshore wind turbine at site 20 with a 10 MWh battery can reach a minimum output of 1 MW.

```
from generation import OnshoreWindModel
from storage import BatteryStorageModel
import numpy as np

gen = OnshoreWindModel(turbine_size=3.6, year_min=2013, year_max=2013,
sites=[20], data_path='data/wind/')

wind_power = np.array(gen.power_out)
```

```
surplus = wind_power - np.array([1]*(365*24))

stor = BatteryStorageModel(capacity=10)

print(stor.charge_sim(surplus))

[out]:
      77.52283105022832
```

2.2.2 Analysing the storage throughput

analyse_usage()

After running a simulation this will return the energy put into storage, the energy extracted from storage, and the total energy curtailed (positive surplus that was not put into storage)

Example: analysis following the previous example.

```
print(stor.analyse_usage())

[out]:
      [721.7000558527664, 640.0424948515829, 8774.16639791325]
```

2.2.3 Sizing a storage system

size_storage(surplus, reliability, initial_capacity=0, req_res=1e3, t_res=1, max_capacity=1e8, start_up_time=0)

This uses bisection out the capacity of storage required to achieve the specified level of reliability. Note that if the max_capacity is too oversized, then the self-discharge rate can mean that increasing storage will actually decrease reliability, so it is important to use a maximum capacity that is not unrealistically large.

Parameter	Type	Description
surplus	*Array like*	*The generation net demand which is to be smoothed in MW*
t_res	*float*	*The time resolution in hours of the surplus provided*
initial_capacity	*float*	*The smallest amount of storage to try, if this achieves greater than the stated reliability then an error is raised.*
req_res	*float*	*The precision to which the required storage needs to be achieved*
max_capacity	*float*	*The maximum amount of storage to try, if this is insufficient to achieve the required reliability np.inf will be returned.*
start_up_time	*int*	*The number of first time intervals to be ignored in reliability calculation*

Example: for the example above work out the amount of storage required to achieve 90% reliability

```
print(stor.size_storage(surplus, 90, max_storage=1000, req_res=1e-3))
```

```
[out]:
      37.12797164916992
```

2.2.4 Calculating the cost of the storage system

get_cost()

Following a charging simulation, this will return the cost in GBP per year of operating the storage system

Example: Calculate the cost of the storage in the 90% reliable system above

```
print(stor.get_cost())
```

```
[out]:
      594039.9169921875
```

3 Multiple Storage Models

3.1 Base Class

MultipleStorageAssets(assets, c_order=None, d_order=None)

Parameter	Type	Description
assets	*Array like*	*A list of StorageModel objects*
c_order	*Array like*	*The priority order for charging under 'ordered' strategy as reference to index in the assets list e.g. [0,1]. If none, assumed to be reverse of the order provided in assets*
d_order	*Array like*	*The priority order for discharging. If none, assumed to be in the order provided in assets list.*

3.2 Functions

3.2.1 Running a charging simulation

charge_sim(surplus,t_res=1, return_output=False, start_up_time=0, strategy='ordered', return_di_av=False)

This simulates the opportunistic operation of the multiple storage assets to remove the negative values in the input array 'surplus'. It returns the percentage of times when a negative surplus is avoided after operation of the storage and, if requested, the output vector after the storage has been used or the daily average charge and discharge profiles.

Parameter	Type	Description
surplus	*Array like*	*The generation net demand which is to be smoothed in MW*
t_res	*float*	*The time resolution in hours of the surplus provided*
return_output	*boo*	*Whether to also return the smoothed output vector*
start_up_time	*int*	*The number of first time intervals to be ignored in reliability calculation*
strategy	*str*	*The strategy to use to operate multiple storage assets. Options: 'ordered'*
return_di_av	*boo*	*Whether to return the daily average charge/discharging profiles of each asset*

Example: get the reliability with which an 3.6 MW onshore wind turbine at site 20 with a 1 MWh battery and 10 MWh of hydrogen can reach a minimum output of 1 MW.

```
from generation import OnshoreWindModel
from storage import BatteryStorageModel, HydrogenStorageModel,
MultipleStorageAssets
import numpy as np

gen = OnshoreWindModel(turbine_size=3.6, year_min=2013, year_max=2013,
sites=[20], data_path='data/wind/')

wind_power = np.array(gen.power_out)

surplus = wind_power - np.array([1]*(365*24))

stor = MultipleStorageAssets([BatteryStorageModel(capacity=1),
                               HydrogenStorageModel(capacity=10)])

print(stor.charge_sim(surplus))

[out]:
      67.51141552511416
```

3.2.2 Analysing the storage throughput

analyse_usage()

This will return one array per storage medium, containing the energy put into and taken out of each asset. The total energy curtailed will also be returned as float.

Example: analysis following the previous example.

```
print(stor.analyse_usage())

[out]:
      [[123.46064881468301, 16.659119543870013],
       [110.31644049533065, 6.033549637298847], 9362.25070352625]
```

3.2.3 Sizing a storage system

size_storage(surplus, reliability, initial_capacity=0, req_res=1e3, t_res=1, max_capacity=1e8, start_up_time=0)

This will size the total storage capacity required to reach a certain reliability, while keeping the relative sizes of the individual assets constant.

Parameter	Type	Description
surplus	*Array like*	*The generation net demand which is to be smoothed in MW*
t_res	*float*	*The time resolution in hours of the surplus provided*
initial_capacity	*float*	*The smallest amount of storage to try, if this achieves greater than the stated reliability then an error is raised.*
req_res	*float*	*The precision to which the required storage needs to be achieved*
max_capacity	*float*	*The maximum amount of storage to try, if this is insufficient to achieve the required reliability np.inf will be returned.*
start_up_time	*int*	*The number of first time intervals to be ignored in reliability calculation*

Example: for the example above work out the amount of storage required to achieve 90% reliability

```
print(stor.size_storage(surplus, 90, max_storage=1e5, req_res=1e-3))

[out]:
      370.10887498781085
```

3.2.4 Calculating the cost of the storage system

get_cost()

Following a charging simulation, this will return the cost in GBP per year of operating the storage system

Example: Calculate the cost of the storage in the 90% reliable system above

```
print(stor.get_cost())

[out]:
       589993.8299611415
```

4 Electricity System

This module combines a set of generation models with a multiple storage asset model to simulate a systems ability to meet an electricity demand profile.

4.1 Classes

4.1.1 Base Class

ElectricitySystem(*gen_list, stor_list, demand, t_res=1, reliability=99, start_up_time=0, strategy='ordered'*)

Parameter	Type	Description
gen_list	*Array-like*	*List of generation model objects*
stor_list	*MultipleStorageAssets*	*MultipleStorageAssets object*
demand	*Array-like*	*Demand to be met in MW*
t_res	*float*	*Time resolution (hours)*
reliability	*float*	*Percentage of demand to be met (0-100)*
start_up_time	*int*	*Number of first time intervals to ignore in reliability*
strategy	*str*	*Storage operation strategy - 'ordered'*

4.1.2 GB electricity system

ElectricitySystemGB(*gen_list, stor_list, t_res=1, reliability=99, start_up_time=40*24*4, strategy='ordered', electrify_heat=False, evs=False, months=list(range(1,14)), year_min=2014, year_max=2019)*

Parameter	Type	Description

months	*Array-like*	*List of months to be included in the simulation (1-12)*
year_min	*int*	*Lowest year to be included in the simulation*
year_max	*int*	*Highest year to be included in the simulation*
electrify_heat	*boo*	*Whether to include electrified heating demand*
evs	*boo*	*Whether to include domestic EV charging*

4.2 Functions

4.2.1 System cost calculations

cost(x)

Returns the whole system cost in £bn /yr.

Parameter	Type	Description
$x[:n_{gen}]$	*Array-like*	*Installed capacity of each generator in GW (using order of gen_list)*
$x[n_{gen}:]$	*Array-like*	*Capacity of first n-1 storage assets relative to total installed. Must sum to less than 1. If only one storage asset then it will be empty.*

Example: The following returns the cost of running the GB system with existing demand using 60 GW each of offshore, onshore, and solar, alongside a 10/90 mix of batteries and hydrogen storage.

```
from generation import OffshoreWindModel, OnshoreWindModel, SolarModel
from storage import BatteryStorageModel, HydrogenStorageModel
from system import ElectricitySystemGB

gen = [OffshoreWindModel(), OnshoreWindModel(), SolarModel()]

stor = [BatteryStorageModel(), HydrogenStorageModel()]

es = ElectricitySystemGB(gen, stor, reliability=99)

print(es.cost([60,60,60,0.1]))

[out]:
      42.958280920024505
```

4.2.2 System operation analysis

analyse(x, filename='log/system_analysis.txt')

Runs a whole system simulation and writes a text file in the log with the system cost breakdown, the amounts and utilisation of storage installed, and total energy curtailment.

Parameter	Type	Description
$x[:n_{gen}]$	*Array-like*	*Installed capacity of each generator in GW (using order of gen_list)*
$x[n_{gen}:]$	*Array-like*	*Capacity of first n-1 storage assets relative to total installed. Must sum to less than 1. If only one storage asset then it will be empty.*
filename	*str*	*File path for analysis to be stored, should end in .txt*

Example: The following analyses the system from the previous example.

```
from generation import OffshoreWindModel, OnshoreWindModel, SolarModel
from storage import BatteryStorageModel, HydrogenStorageModel
from system import ElectricitySystemGB

gen = [OffshoreWindModel(), OnshoreWindModel(), SolarModel()]

stor = [BatteryStorageModel(), HydrogenStorageModel()]

es = ElectricitySystemGB(gen, stor, reliability=99)

es.analyse([60,60,60,0.1],filename='log/analysis.txt')

[out]:

System cost: £42.958280920024505 bn/yr

---------------------
INSTALLED GENERATION
---------------------

Offshore Wind: 60 GW
Onshore Wind: 60 GW
Solar: 60 GW

>>TOTAL: 180 GW

------------------
INSTALLED STORAGE
------------------

Li-Ion Battery: 0.4882112529278146 TWh
Hydrogen: 4.494901276450441 TWh
```

```
>>TOTAL: 4.882112529278146 TWh

--------------------
STORAGE UTILISATION
--------------------

>> Li-Ion Battery <<

1.426515280796148 TWh/yr in (grid side)
1.275665601444428 TWh/yr out (grid side)
4.458956654147789 cycles per year

>> Hydrogen <<

0.2718659270924416 TWh/yr in (grid side)
0.07082591178249478 TWh/yr out (grid side)
0.04619874541792025 cycles per year

-------------------
ENERGY UTILISATION
-------------------

Total Demand: 288.49464577846245 TWh/yr
Total Supply: 580.6900640898285 TWh/yr
Curtailment: 41.52144687548876 TWh/yr

---------------
COST BREAKDOWN
---------------

Offshore Wind: £15.44999218041244 bn/yr
Onshore Wind: £8.420025411401847 bn/yr
Solar: £2.52 bn/yr
Li-Ion Battery: £6.211480046845044 bn/yr
Hydrogen: £0.4568844814641887 bn/yr
```

4.2.3 Visualisation of daily load profiles

get_dirunal_profile(gen_cap, stor_cap)

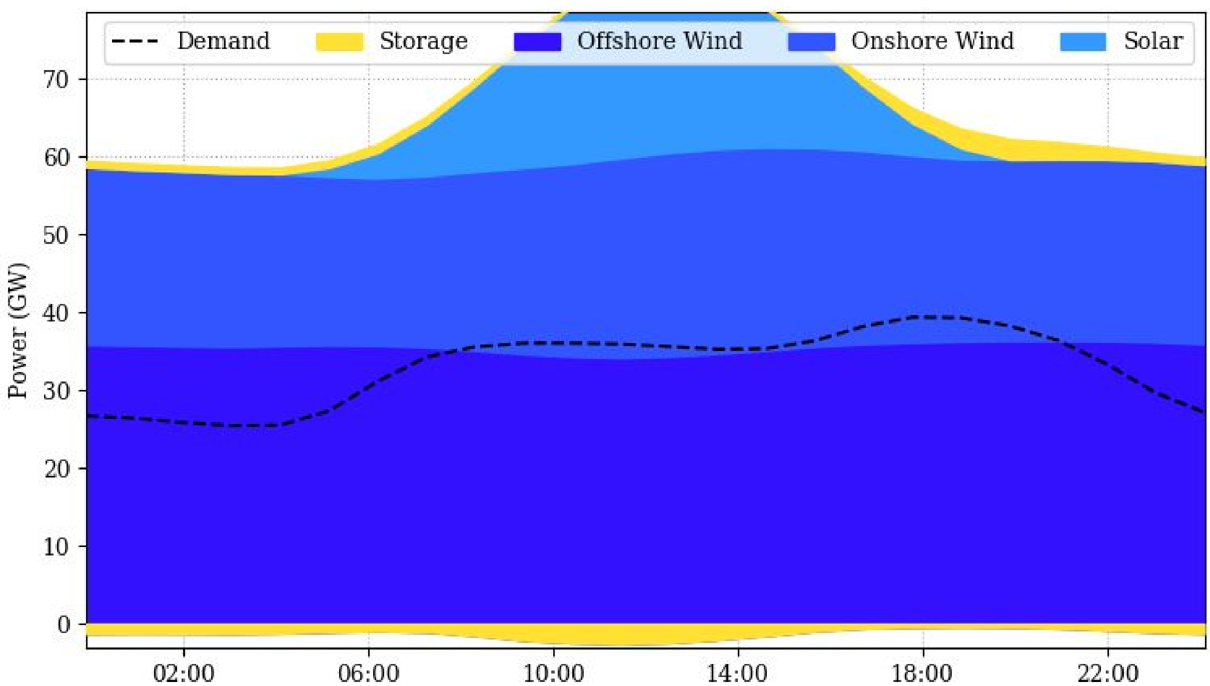
Plots the average daily supply and demand breakdown, alongside the average daily usage profile for each storage asset.

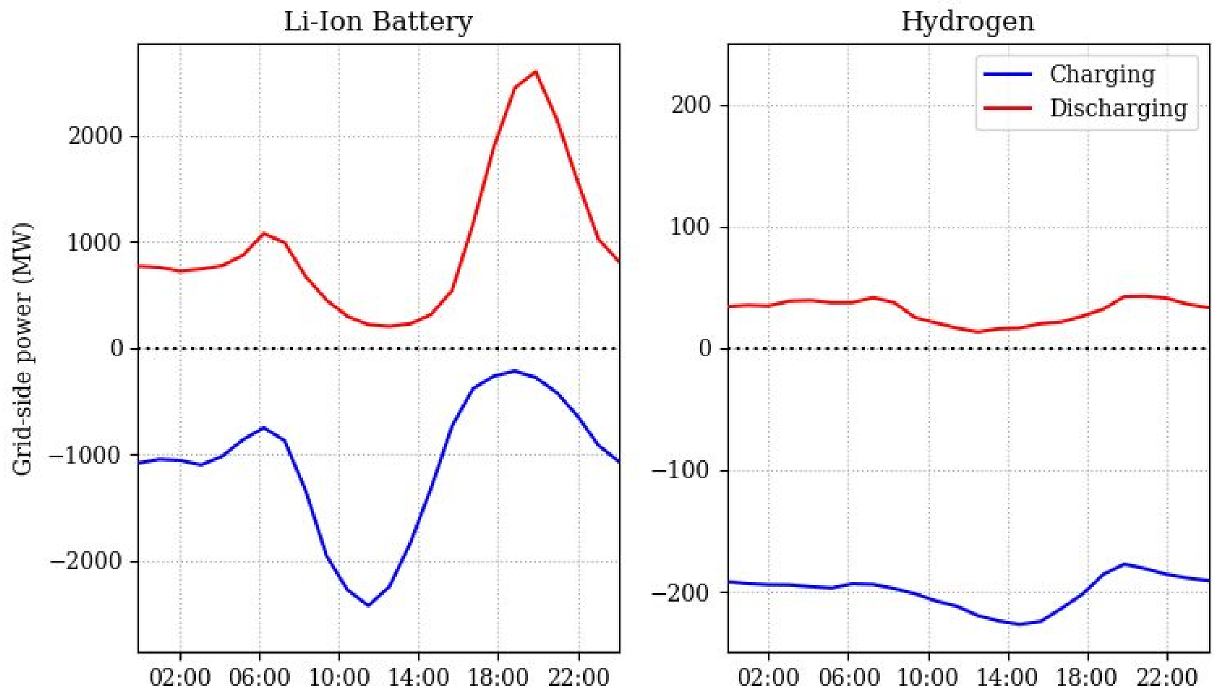
Parameter	Type	Description

gen_cap	*Array-like*	*Installed capacity of each generator in GW (using order of gen_list)*
stor_cap	*Array-like*	*Capacity of first n-1 storage assets relative to total installed. Must sum to less than 1. If only one storage asset then it will be empty.*

Example: The following plots the usage profile of the system from the previous example.

```
from generation import OffshoreWindModel, OnshoreWindModel, SolarModel
from storage import BatteryStorageModel, HydrogenStorageModel
from system import ElectricitySystemGB

gen = [OffshoreWindModel(), OnshoreWindModel(), SolarModel()]

stor = [BatteryStorageModel(), HydrogenStorageModel()]

es = ElectricitySystemGB(gen, stor, reliability=99)

es.get_diurnal_profile([60,60,60],[0.1])

[out]:
```





5 Load factor estimator

This module uses the results from the generation models to estimate the load factor of a generator at a specific location, by interpolating the available data points.

5.1 Base Class

LoadFactorEstimator(gen_type, data_loc=None)

Parameter	Type	Description
gen_type	*str*	*Code to determine the type of generator. 'w' for onshore wind, 'osw' for offshore wind, and 's' for solar.*
data_loc	*str*	*Location of folder containing the raw weather data*

5.2 Functions

5.2.1 Determine load factors at all available sites

calculate_load_factors()

This will estimate the load factor of a generator at each of the sites provided, and store the results in the stored_model_runs folder.

5.2.2 Estimate the load factor at a particular point

estimate(lat, lon, max_dist=1, num_pts=3)

This will estimate the load factor of a generator at a specified location, by interpolating the estimated load factors at the site locations. A weighted average of the closest n points will be performed, providing the points are within the specified maximum distance.

Parameter	Type	Description
lat	*float*	*Location latitude*
lon	*float*	*Location longitude*
max_dist	*float*	*The largest straight line distance in degrees that a point will be used for interpolation*
num_pts	*int*	*The number of closest points that will be used for the weighted average*

Example: Calculate the load factor of a solar farm in Greenwich Park

```
from maps import LoadFactorEstimator

lfe = LoadFactorEstimator('s')

print(lfe.estimate(51.48,0.00))

[out]:
      12.933988121729667
```

6 Load factor maps

This module uses the results from the load factor estimation to plot maps showing the geographic variation in predicted load factor.

6.1 Classes

6.1.1 Base class

LoadFactorMap(load_factor_estimator, lat_min, lat_max, lon_min, lon_max, lat_num, lon_num, quality, is_land)

Parameter	Type	Description
load_factor_estimator	*LoadFactorEstimator*	*Object to fill in the estimates at each point*
lat_min	*float*	*Minimum latitude to show on map*
lon_min	*float*	*Minimum longitude to show on map*
lat_max	*float*	*Maximum latitude to show on map*
lon_max	*float*	*Maximum longitude to show on map*
quality	*str*	*'h' for high resolution, 'l' for low resolution*
is_land	*boo*	*Whether the shaded area is on land or off land*

6.1.2 Offshore wind map

OffshoreWindMap(lat_min=48.2, lat_max=61.2, lon_min=-10.0, lon_max=4.0, lat_num=400, lon_num=300, quality='h', data_loc=None)

Parameter	Type	Description
lat_num	*int*	*Number of x points on the shaded mesh*
lon_num	*int*	*Number of y points on the shaded mesh*
data_loc	*str*	*Path to weather data - required if load factors have not previously been saved*

6.1.3 Onshore wind map

OnshoreWindMap(lat_min=49.9, lat_max=59.0, lon_min=-7.5, lon_max=2.0, lat_num=400, lon_num=300, quality='h', turbine_size=3.6, data_loc=None)

Parameter	Type	Description
lat_num	*int*	*Number of x points on the shaded mesh*
lon_num	*int*	*Number of y points on the shaded mesh*
turbine_size	*float*	*Rated capacity of individual turbine in MW*
data_loc	*str*	*Path to weather data - required if load factors have not previously been saved*

6.1.4 Solar map

SolarMap(lat_min=49.9, lat_max=59.0, lon_min=-7.5, lon_max=2.0, lat_num=400, lon_num=300, quality='h', turbine_size=3.6, data_loc=None)

Parameter	Type	Description
lat_num	*int*	*Number of x points on the shaded mesh*
lon_num	*int*	*Number of y points on the shaded mesh*
data_loc	*str*	*Path to weather data - required if load factors have not previously been saved*

6.2 Functions

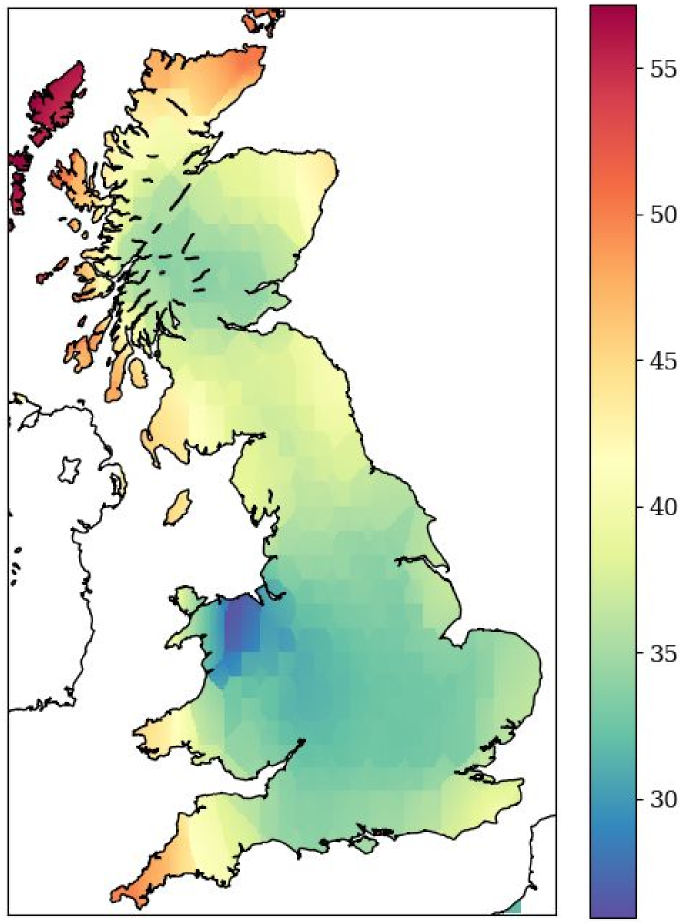
6.2.1 Draw a map

draw_map(show=True, savepath='', cmap=None, vmax=None, vmin=None)

Parameter	Type	Description
show	*boo*	*Whether to show the result*
savepath	*str*	*If desired, location to save the result*
cmap	*matplotlib.cm*	*Color map to use for shading*
vmax	*float*	*Value to cap the colour map at (default is set to the max value)*
vmin	*float*	*Minimum value to cap colour map at (default is the min value)*

Example: Plot a map of onshore wind load factor

```
from maps import OnshoreWindMap

mp = OnshoreWindMap()

mp.draw_map()

[out]:
```


55
50
45
40
35
30