**Agent Based Modelling Energy Simulator**

Python Instruction Manual

Authors: Karl Mason, Meysam Qadrdan

This document provides a comprehensive overview of the Agent Based Modelling Energy Simulator code base.

**Installation:**

The code is written in python 3. The specific version of python is python 3.6.7 running on a windows 10 machine. This can be downloaded from: <https://www.python.org/downloads/release/python-367/>

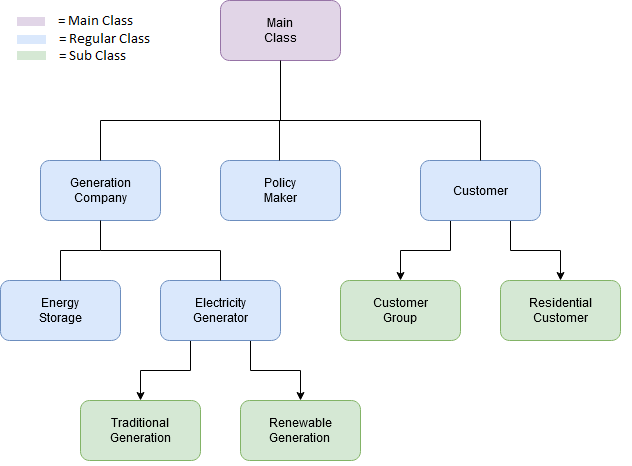
The code also uses some other packages, e.g. numpy, pandas, matplotlib, geopy. Once python 3 has been installed on the machine, simply type e.g. ‘pip install numpy’ into the windows command prompt to install these packages. Once these have been installed, the code should run.

**Structure:**

The code consists of a number of python files that perform various functions of the model. These python files include:

* **mainSim.py** This is the main class that runs the model. This class creates customers and generation companies. In this python script, the number of years the model is running for is also set. It loops through each year, in which the customer demand is met by renewables and nuclear, then battery storage and finally traditional generation technologies. At the end of each year, the policy maker class is called to update its parameters, e.g. CO2 price. These are given to the generation company who then decides to invest or new technologies or not.
* **generationCompany.py** This class represents a single generation company. It contains a list of plants (instances of renewableGenerator.py and traditionalGenerator.py) and also a list of energy storage. This class contains methods to get renewable generation, charge/discharge batteries, get traditional generation, make capacity investment decisions, bid in capacity auctions, record and update various other parameters (e.g. new CO2 price), and also to display graphs.
* **electricityGenerator.py** This class represents a plant. This is a super class of the renewableGenerator and traditionalGenerator classes. The purpose of this class is to contain information that is shared between these two sub classes. When creating plants in the model, use either of the two sub classes, **not** electricityGenerator.
* **renewableGenerator.py** This is a subclass of electricityGenerator. This class represents a renewable plant, e.g. solar, hydro, etc. This class reads in generation profile for the renewable type and scales the profile to the capacity of the plant. This class also calculates the various costs, revenue streams and economic metrics (NPV and ROI) for the plant.
* **traditionalGenerator.py** This class represents a non-renewable plant, e.g. CCGT, coal, and is a subclass of electricityGenerator. The main difference between this class and renewableGenerator in terms of functionality is how it generates electricity. This class receives a demand profile that it uses to determine the plants generation profile. This class also records costs, revenue, etc.
* **customer.py** This class represents a single customer. This customer reads in an electricity demand profile can scale it as desired. This class is also super class to the customerGroup class which is used throughout the simulator.
* **customerGroup.py** This class represents a group of customers and is a subclass of the customer class. This class reads in a demand profile and scales it to the number of customers it represents. It is also used to represent non-residential customers in the model. This class uses many of the methods implemented in its super class, customer.py.
* **policymaker.py** This class represents the policy maker that dictates the policy decisions in the simulator, e.g. CO2 price, holding capacity auctions, etc. All high level policy decisions are made in this class.
* **energyStorage.py** This class represents a battery. It contains methods to consume electricity in order to charge the battery, provide electricity by discharging the battery and also to maintain its charge capacity.
* **drawMap.py** This class contains the functionality to display electricity demand for various regions in GB on a map. This map uses the geopy package to convert region names into GPS coordinates and to then display these on the map. The idea is that this package will be useful if this code is to be extended to simulate countries other than GB in the future, with little extra effort. **\*\*Note:** The majority of the simulations simply use one aggregated customer to represent the demand for GB. Therefore, drawMap is not needed for these simulations.
* **drawMap\_noGeopy.py** This class is almost the same as drawMap.py with the exception that it does not use the geopy package. The geopy package requires an internet connection and will not work if this is not available. This class displays the same information on a map but the locations are hard coded instead.
* **Utils.py** This is a general use class that contains methods that are used throughout the rest of the simulator. This class saves time by preventing re-implementing the same method repeatedly in different places. This class contains functionality to read in files, display graphs, etc.

The figure below provides an illustration of the main classes in the model.



*Figure1: Model Class Structure*

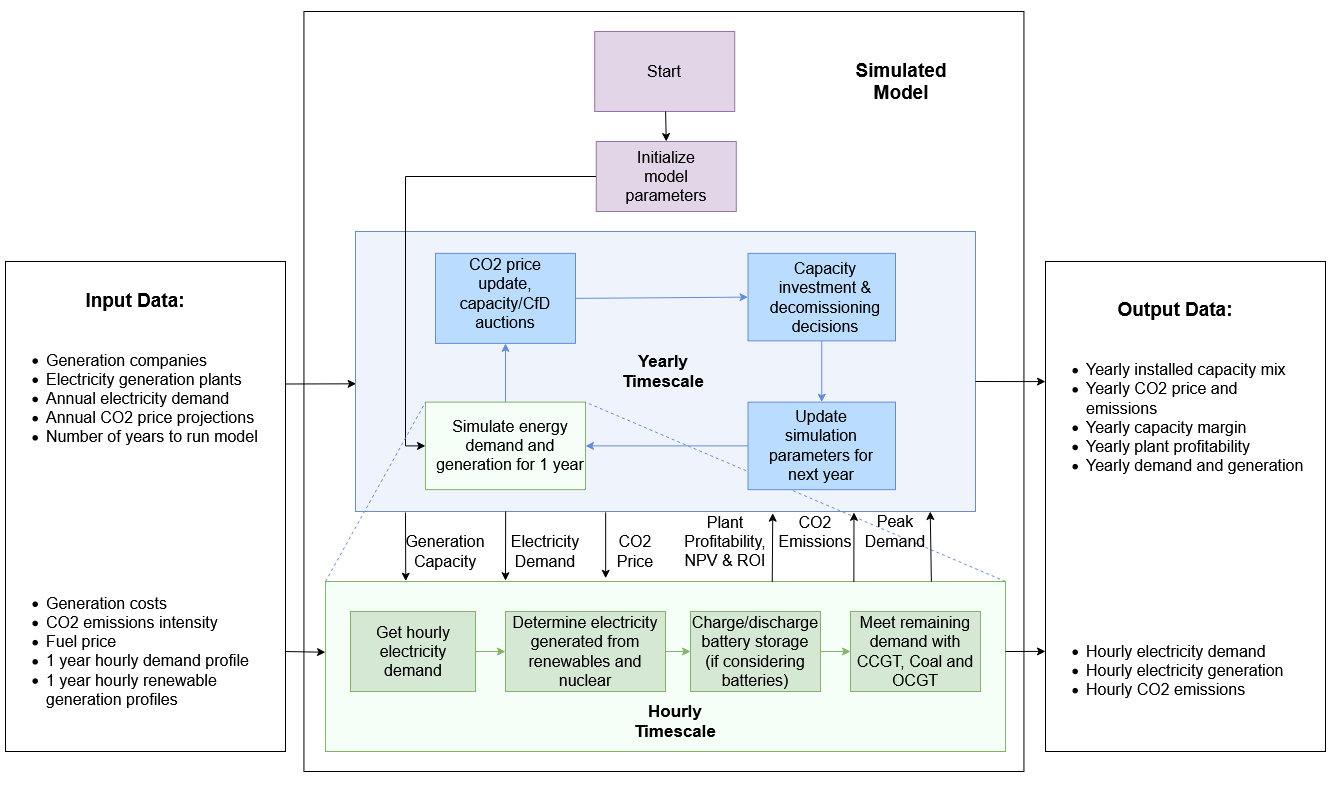
**Files:**

The code reads in a number of data sets. These are saved within various folders in the codebase. Each of the renewable generation profiles are saved in their respective folders, e.g. hydro generation is saved in the ‘Hydro’ folder. The load profiles are saved in the ‘Load’ folder. There are a number of datasets relating to yearly economic data, e.g. yearly wholesale electricity price and population growth rates. These are also contained in folders within the software package, i.e. ‘WholesaleEnergyPrices’ and ‘PopulationGrowthRates’ respectively. The ‘Images’ folder contains map images for the drawMap.py file. The ‘Results’ folder contains any file outputs of the simulator. The plant ownership data that corresponds to the companies that own each of the individual plant is contained within the folder ‘OtherDocuments’. This folder also contains other raw data files obtained from BEIS and other sources.

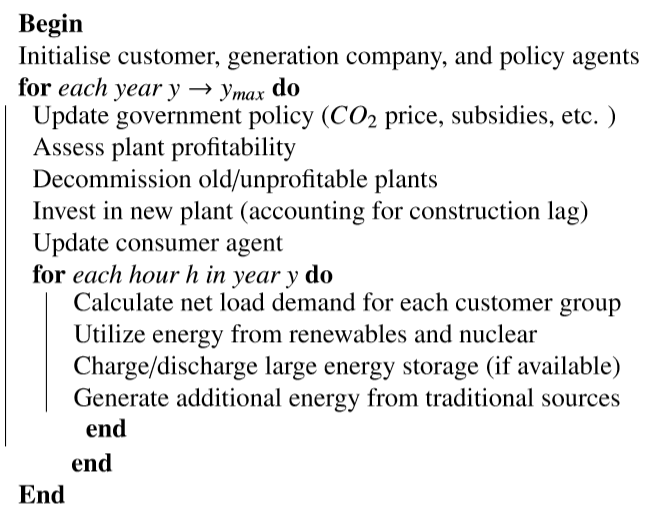
The Datasets.xlsx file contains all of the data sets needed to run the model. However, running the model by reading data directly from the .xlsx file is **significantly** slower than reading data from the smaller individual .txt and .csv files.

**Functionality:**

The overall functionality of the code base is outlined in Figure 2 below and in the pseudocode below. The model consists of an outer loop that loops over each year and an inner loop that loops through each hour in the year. High level policy and investment decisions are made in the yearly loop while operational aspects are considered at an hourly level.



*Figure 2: Simulator Flowchart*



*Algorithm 1: Simulator Pseudocode*