how to use the hydrogen cost-calculation tool

This document aims to guide you through the installation and setup of Python and a working IDE, in this case PyCharm, in order to use the model. The guideline is aimed towards users who have never used Python before. Important note: this tutorial was put together for windows 10. So if you're using another operating system, be aware that you will have to download certain files according to your machines software specifications. If you are a seasoned Python user you might want to skip to the package dependencies section or to the end of this document, where the model parameters are discussed.

The model

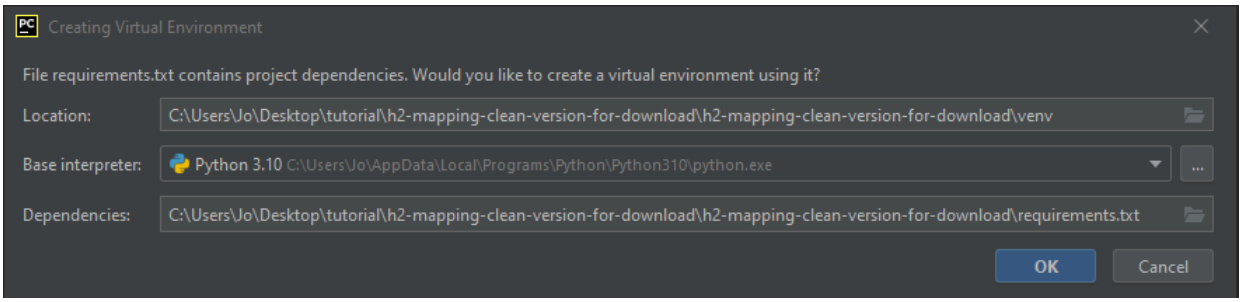
The hydrogen cost-calculation model consists of two parts. Both of which try to calculate the cost of green-hydrogen in the future. The final cost is made up from the production and the transport cost. The "base"-model is relatively static, where you can change parameters once for every run. Depending on the year in which the hydrogen would be needed, the model takes different underlying market and technology data into account. The monte-carlo-simulation based on this model covers the uncertainty inherent in future forecasts much better. By allowing the underlying production cost related data to range between extreme estimations, the simulation can compute a distribution of hydrogen production costs. To cover the whole globe, a network of 5970 points was constructed where renewable energy production from solar or wind would be feasible. Connecting these locations to their solar and wind energy potential, establishes a starting point for the calculation of the hydrogen production costs. In regards to the transport cost there are a lot of different factors. There are three modes of transportation available, which are truck, ship and pipeline. Furthermore the hydrogen itself can be transported in different states. It could be stored as-is in gaseous phase, liquified, converted to Ammonia or to Toluene as an example for liquid organic hydrogen carriers (LOHC). Of course conversion and reconversion come with an added cost but they are advantageous at longer distances because of their storage characteristics. According to the geographic location of the hydrogen-demanding facility the program will find the point with the cheapest combination of production and transport cost.

Setting up Python and PyCharm

Firstly you need to download and install the python programming language. You can find the newest release on https://www.python.org/. While going through the installation choose the option "Add Python 3.10 to PATH". Secondly choose and install an editor, in this guide we will use PyCharm. It is an IDE distributed by Jetbrains (www.jetbrains.com/pycharm/). The free "Community" edition is sufficient for our purposes. If not done already, download H2-mapping repository from github. Then open the "mapping-h2-main" -folder as a project in PyCharm.

Installing packages

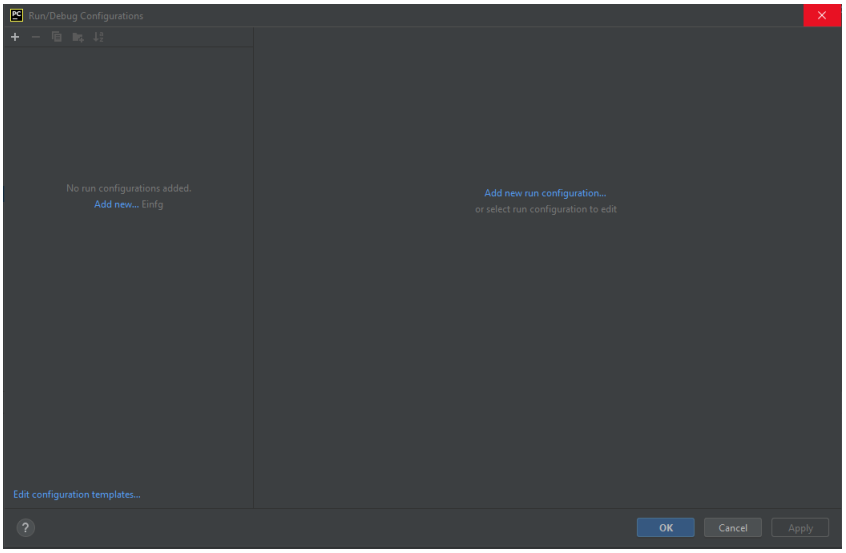
PyCharm should automatically detect the requirements.txt file, which allows us to conveniently install all required packages:



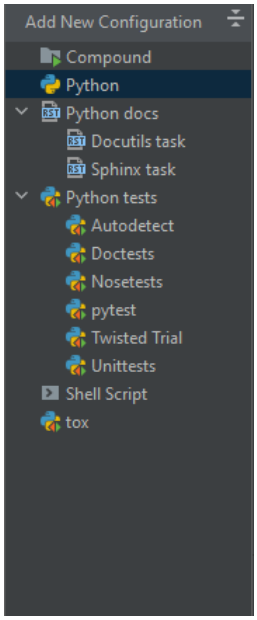
If that's not the case, install the packages through requirements.txt.
You can do this by putting `pip install -r C:\Path\to\requirements.txt` into the Terminal. Note that you need to fill in the path to the reuirements.txt file according to your folder structure. If the manual installation via requirements.txt doesn't work either or when a package is missing, type `pip install package_name` (pandas, geopy, requests, scipy, GDAL, fiona, networkx, haversine, shapely, geopandas, plotly...) into the Terminal. Some packages can not be downloaded from within PyCharm and have to be manually downloaded e.g. fiona and GDAL (wheels for a Windows 10 are delivered within the h2-mapping-directory). Use `pip install path/to/.whl` to install the package from a wheel-file. Now you will need to modify a few areas of the code in order to make it run on your own device. Certain paths in the code have to be set according to your own pc. Namely within the `geo_path.py` and `mc_geo_path.py` files. The `sys.path.append()` statements (lines 11 and 12) in `geo_path.py` need to be modified according to your computer's folder structure. The same needs to be done in `mc_geo_path.py` for the lines 8 and 9.

Interpreter configuration

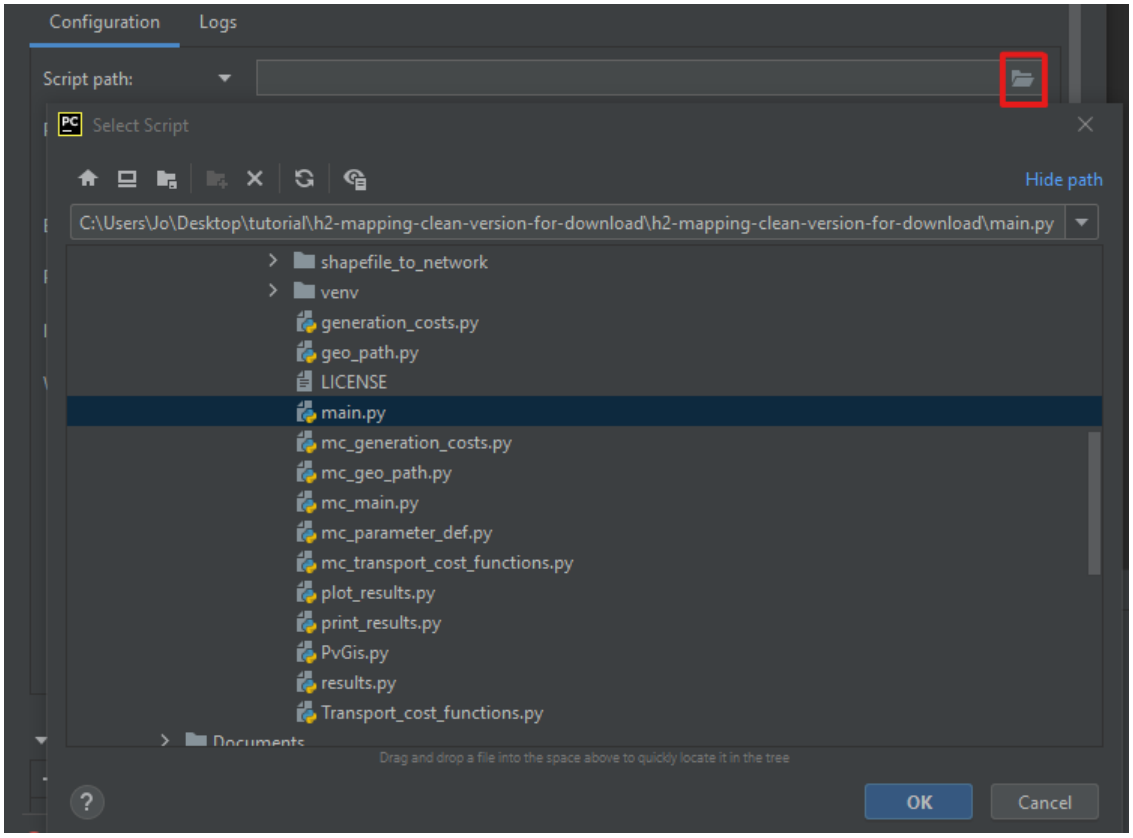
Before running `main.py / mc_main.py` you have to configure a python interpreter. At the time of writing this guide the standard interpreter would be called Python 3.10 and comes preinstalled with the python programming language. You can set the interpreter by clicking on `Add Configuration...` in the top right of the Window. Then the following window should pop up.



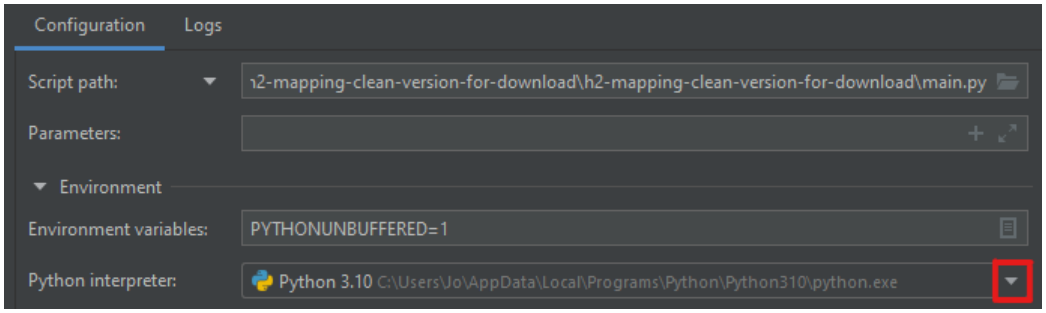
Now you click on `add new...` and select Python.



Afterwards you select the .py file you wish to run (in this case either “main” or “mc_main”) as the script path.



Finally under Python interpreter select Python 3.10 and exit the window by clicking on "Ok".



Running the model

Now you should be able to run the code by clicking on the green "Play" button in the top right corner of the PyCharm environment. You can change parameters directly in `main.py / mc_main.py` by modifying the values shown in the following graphic:

```
31  # Define parameters for the main model
32  end_tuple = (25.28893697992723, 51.565533460963046)   # [lat, long]
33  h2_demand = 50  # [kt/yr]
34  year = 2020
35  centralised = True
36  pipeline = True
37  max_pipeline_dist = 2000
```

The `end_tuple` variable takes the geographical coordinates of the site where the Hydrogen is needed. The annual hydrogen demand is given in `h2_demand` (kilotons per year). The `year` variable can be set to the following years 2020/2030/2040/2050. The `centralised` variable is either True or False, depending on whether the centralised re-conversion of transport-intermediates (Ammonia, LOHC, lq. H2) is desired or omitted. The same goes for the `pipeline` variable which either includes or excludes the usage of pipelines in the model. Lastly with the help of `max_pipeline_dist` you can put a restriction on the maximum length of a pipeline in kilometers, given that it is used in the model. A summary of your results is shown in the console. Additionally The whole calculation results are stored in the "Results" folder. You will find the latest run under "final_df". The file is organized by rows and columns. The rows are simply the 5970 different locations while the columns contain all the calculation steps from solar potential, over capital expenditures to total cost of hydrogen per kilogram.

In the Monte-Carlo-simulation there are two more parameters that you can set.

```
52  # Define parameters for the main model
53  end_tuple = [(23.0550, 113.4242), (29.6084, -95.0527)]  # [lat, long]
54  h2_demand = [100]  # [kt/yr]
55  year = [2030]
56  centralised = True
57  pipeline = True
58  max_pipeline_dist = 10000
59  iterations = 1000
60  elec_type = ['alkaline']
```

Firstly you can set the number of `iterations` and secondly the electrolyzer technology by changing the `elec_type` parameter. The three technologies available are "alcaline" for alkaline electrolysis, "pem" for polymer electrolyte membrame electrolysis and "soe" for solid oxide electrolysis. Also note that it's possible to make the calculations for multiple locations at once by giving a list of coordinates. When all packages are installed and all

files are in the correct folders `main.py / mc_main.py` should run and iterate. To run the main files, right-click on `main.py / mc_main.py` an click on `run 'main' or run 'mc_main'` respectively. Remember that a python interpreter has to be set before you can run the program. The Computing time depends on whether a new location is entered (meaning the shortest path algorithm has to run) or how many mc simulation iterations are desired. Of course the components of your PC also play a role. Like in the main model you will find your results in the "Results" folder. All Monte-Carlo-Simulation runs are stored in the "mc" subfolder and for every run there will be an extra folder containing 4 csv-files. They are named according to their content and are organised by iterations as rows (e.g. 1000 iterations = 1000 rows) and locations in the columns.