Model Documentation

Optimal PV Expansion Pathways: Monte Carlo Model

raul.hochuli@unibas.ch

February 12, 2025

Abstract

This short document the main functionality of the main parts if the model code and further explains certain parameter choices and settings, the reason behind them and how they originated.

Contents

1 Code Structure / Way of Working

OptimalPV_RH_directory_structure.txt - Notepad

File Edit Format View Help

```
** Directory structure for OptimalPV_RH **
 date: 2024-12-31 13:15:06.462548

├── auxiliary_functions.py
├── data_aggregation
│   ├── api_electricity_prices.py
│   ├── api_entsoe.py
│   ├── api_pvtarif.py
│   ├── default_settings.py
│   ├── extend_data.py
│   ├── get_elecpri_data.py
│   ├── preprepare_data.py
│   ├── split_data_geometry.py
│   └── sql_gwr.py
├── data_aggregation_MASTER.py
├── execution_RUNFILE.py
├── execution_scenarios.py
├── numpy-2.1.1-cp312-cp312-win_amd64.whl
├── plots
│   ├── map_municipality_to_DSO.py
│   ├── pv_coutns_power_timeseries.py
│   └── pv_houseclass_mapping.py
├── Predictions.py
├── pvalloc_initialization_MASTER.py
├── pvalloc_MCalgorithm_MASTER.py
├── pv_allocation
│   ├── alloc_algorithm.py
│   ├── alloc_sanitychecks.py
│   ├── default_settings.py
│   ├── initialization_large_functions.py
│   ├── initialization_small_functions.py
│   ├── inst_selection.py
│   └── pvprod_correction.py
├── solkat_month.csv
├── topo_subdf_0to399.csv
├── visualisations
│   ├── defaults_settings.py
│   ├── individual_visualisations.py
│   ├── plot_auxiliary_functions.py
│   ├── plot_ind_charac_omitted_gwr.py
│   ├── plot_ind_hist_NPV_freepartitions.py
│   ├── plot_ind_hist_pvcapaprod_sanitycheck.py
│   ├── plot_ind_hist_pvprod_deviation.py
│   ├── plot_ind_line_installedCap.py
│   ├── plot_ind_line_meteo_radiation.py
│   ├── plot_ind_line_productionHOY_per_node.py
│   ├── plot_ind_map_base.py
│   ├── plot_ind_map_node_connections.py
│   ├── plot_ind_map_topo_egid.py
│   └── plot_ind_var_summary_stats.py
└── visualization_MASTER.py
```

Figure 1: Code structure of OptimalPV model.

Figure 1 shows how the model code is built with all relevant files. The order is **"execution" files** > **"MASTER" files** > **subfiles**. Here is a quick summary of what the Master files are doing.

1.1 data_aggregation_MASTER

- After basic setup (including number of BFS municipalities extension if applicable), all local spatial data (solkat, solkat_month, pv_energy production, gwr) is split (data and geometries are separated to reduce storage space), and all necessary API sources are accessed to download the most recent version of data (PV tariffs by grid operators, mapping of relevant grid operator to BFS municipality). All data is then stored in a separate input folder, which is accessed by other data-agg scenarios, if **split_data-geometryAND_slow_api** is not activated. This code part is only run once on a large, inter-cantonal level, which certainly captures all relevant areas for all later stages.

- Get "fast" API data and transform it.

- Get all relevant building data from the GWR (own function because the SQL database is fast enough), get data from the separate input folder (spatial data), and time series data (demand[1] and meteo radiation that differ within one or multiple years).

- In **local_data_AND_spatial_mappings**, I change/extend the solkat data frame in an important manner: For a large number of buildings, the data frame contains a single shape covering the entire roof which however contains multiple buildings of the GWR-registery. The problem is, that this distorts the picture in two effects. 1st some residential roofs are much too large, giving unrealistic big installation capacity numbers in the model. 2nd the variable "STROMERTRAG" which serves as a sanity check to compare my modeled pv production to is also positively distorted. Figure 2 shows this in QGIS and the distribution of PV production within the sample by my model and the estimation by Sonnendach in STROMERTRAG. I solves this problem by selecting all GWR EGIDS within the identical Solkat EGID and add the same shape for each GWR-EGID within the s.join (the GWR points within the shape). Then the FLAECHE for all theses new entries in the solkat data

[1] Find some demand data, match houses to a demand archetype, and then attach that archetype and relevant demand time series to the building EGID.

frame are divided by the number of EGIDs within the shape selection. This is not optimal but the best alternative to drastically misleading roof sizes. Also the radiation for each roof partition is multiplied by the linear transformation given by the solkat_month data frame, giving a specific linear transformation (Fig 3, $a_{param} \cdot direct\ radiation + b_{param} \cdot diffuse\ radiation + c_{param}$)) for each roof partition of each month in the year. This solves the main issue within the solkat data frame and also includes a large share of additional GWR EGIDs in the sample which would have been otherwise excluded as they have not roof data and cannot be processed by the model. See Fig 4 to see how prediction accuracy improved before and after extending SOLKAT with missing EGIDS and dividing FLAECHE (large bulk of installations with 0 productions is also solved, DF_UID now only a unique identifier within an EGID subset, but therefore solkat_month can match the abc parameters correctly to the individual partitions).

- Extend the pre-prepped data with standalone information: Estimate a cost function stating the cost of an installation by kWp and define a reduction factor table for roofs that have a certain angle/tilt. Both are just data entered into the code file directly, which export a function or table to the pre-prep data folder.

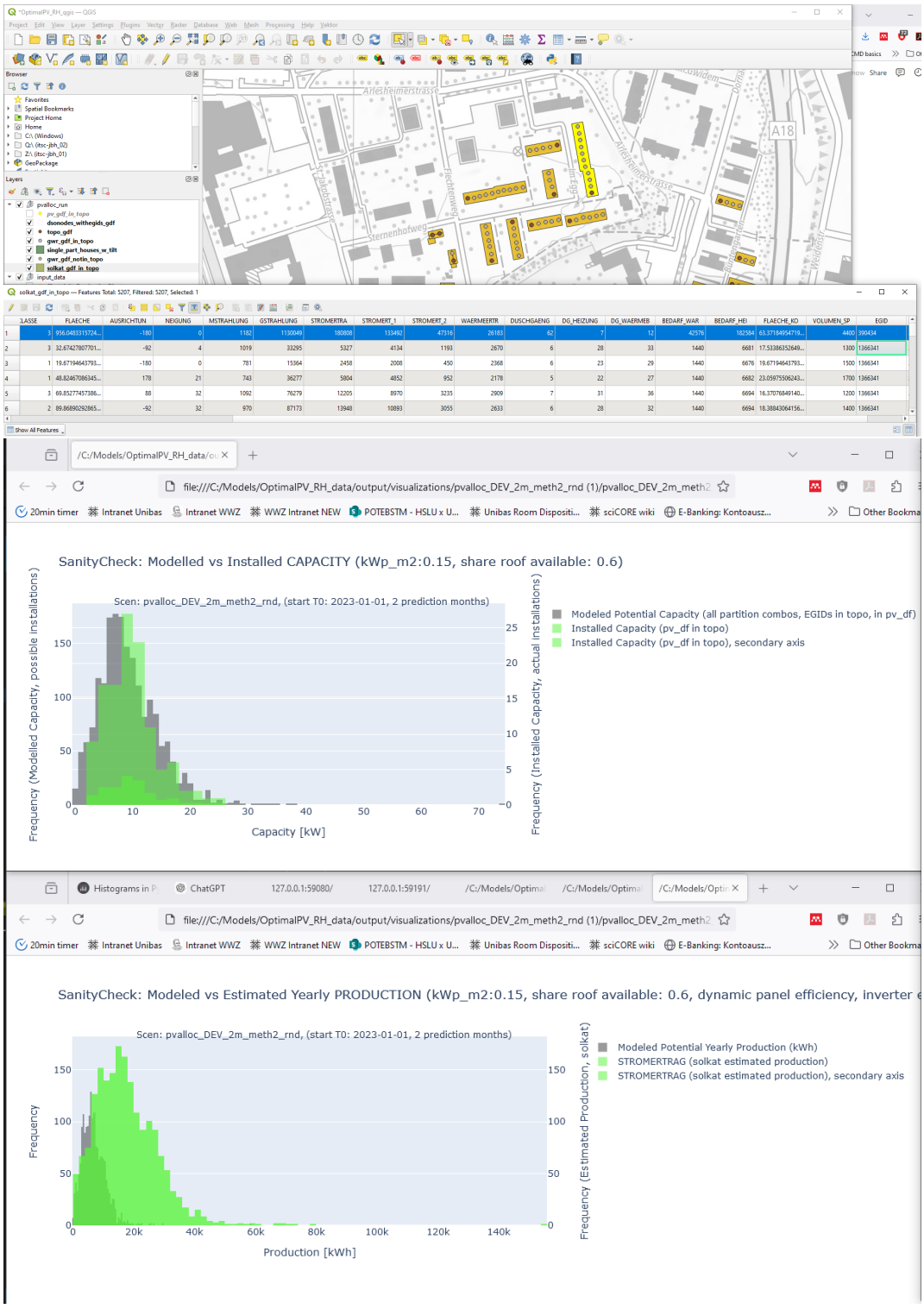

Figure 2: GWR SOLKAT EGID Missmatch

Name	Alias	Typ	Domäne	Kardinalität	Erläuterung
A_PARAM	Parameter a	Float		1	Mit den Parametern a, b und c lässt sich für den jeweiligen Monat aus den Direktstrahlung (DIR) und Diffusstrahlung (DIF) auf eine horizontale Fläche die Strahlung auf die Dachfläche über die Linearkombination (a*DIR + b*DIF +c) annähern.
B_PARAM	Parameter b	Float		1	Siehe A_PARAM
C_PARAM	Parameter c	Float		1	Siehe A_PARAM

Figure 3: SOLKAT_month description of abc parameters

Figure 4: PV production histogram for all EGIDs in samlpe, before and after extending the missing EGIDS within the solkat DF

1.2 pvalloc_initialization_MASTER

- After basic setup, I add standalone data to the preprep_data folder: add hours of the weather year and add the grid nodes from DSO operator. These steps should / could be added to the data_aggreation_MASTER, but were initially added here and left for simplicity.

- **initial.import_prepre_AND_create_topology**: This imports all the non-time-series data and transforms it into a usable format. For EGIDs that are in the sample but not in the EGID-DSO grid node mapping, I attach the EGID to the closest DSO node. Next, I create the topology, building a directory with all relevant EGIDs from the GWR as keys and all relevant information as values: **pv_df** (TF_switch for a pv installation or not, the source of the instillation (real-world data based or assigned by algorithm, installation or roof unique identifier number, date of installation, power of installations), **solkat** (all solkat-partitions attached to the EGID including FLAECHE, STROMERTRAg, AUSRICHTUNG, NEIGUNG; the iteration/list of all possible combinations of roof-partitions per EGID is only added later, after the economics components of each partition area already determined), **netflex data**[2] (aggregated 300 consumers to 4 main demand archetypes: high/low demand by a certain proxy of GWR, with/out heat pump by GWR heating assets, then match the archetype to EGID and store demand profile separately for later access), **pvtarif** (the tariff for a kWh of PV production fed into the grid; mean of two values if multiple operators), **elecpri** (the electricity price set by grid operators for the houses within the municipalities, separated into energy costs, grid costs, aid fees, taxes and fixed costs).

- **initial.import_ts_data**: imports all the time series data that is specific for a selected year (e.g. a certain weather year) and creates time-dependent data frames (e.g. grid premium per hour of year (HOY)).

- **initial.define_construction_capacity**: define the construction capacity for each future month, dependent on a certain set of past months by looking at how much was kWp was built during a couple of past years. This number is then increased by a linear growth rate, for each following year and split up equally with a larger share going to summer months because that's when more construction is occurring than during winter months.

[2] demand curves for residential households used from a former research project with Ali Darudi

- **aloc.calc_economics_in_topo_df**:
  - Run through the created topology directory (topo_egid.json), extract all the relevant criteria for each building and build a tabular data frame of it (topo_df).
  - The topo data frame is already quite large, containing a row for each roof partition of each house, it is now getting even larger because I need to specify the production for each hour, hence the data frame gets 8760 times larger. This is not manageable for regular computers, so I partition the computation steps below for individual chunks of the data set, determined by the "topo_subdf_partitioner" that determines how many EGIDs are in each sub-data frame.
  - I merge the radiation hours of the weather year to the subdf (diffuse/direct radiation and temperature). Dependent on the settings, I then compute the actual radiation per partition either with a flat linear combination (e.g. $factor_{direct\ rad} \cdot direct\ radiation + factor_{diffuse\ rad} diffuse\ radiation$) or the month-dependent abc linear transformation, specified for each partition for each month ($a_{param,month} \cdot direct\ radiation + b_{param,month} \cdot diffuse\ radiation + c_{param,month}$)) which is supposed to also consider the shading of other buildings. This gives the radiation potential for each partition in W/m^2 used for calculating PV production for a roof directly.
  - I next add a column determining the relative radiation, by dividing the radiation for each partition by the mean of the top radiation hours (mean of the 20 hours in the weather year with the highest radiation). Depending on the settings, the mean top hours are taken for the general 20 top hours during the weather year or the top 20 roof-specific hours for each individual partition. Depending on the abc-transformation, these might be different hours, with different top-radiation values (leading to higher relative radiation). This gives another method of computing the PV production, using kWp values from existing and modeled installations.
  - Next, I add the panel efficiency factor to each roof by the hour of the year, either as a flat constant or a flat constant that is further reduced during determined hot hours within the weather year. There is also supposed to be a panel efficiency factor calculated per hour of the year, depending on the temperature, extracted from the meteo data. (Check again where that is in the code, or if it has not been built/coded yet).

- Using the stored demand profiles and the demand archetype assigned to the house previously, I estimate a load curve for each house for a given hour of the year to allow for self-consumption. Swiss Store data would be a more sophisticated source for demand profiles[3]
- Next, I compute the electricity production per house for each hour of the year using different methods:
  - **method2.1** uses the $radiation\ (W/m^2) \cdot inverter\ inefficiency \cdot share\ of\ roof\ available \cdot panel\ efficiency \cdot FLAECHE\ (m^2) \cdot angle\ tilt\ reduction factor$ (method2.2 omits the angle tilt reduction factor[4]).
  - **method3.1** uses the $relative\ max\ radiation\ (0/1) \cdot kWp/m^2\ assumption \cdot inverter\ inefficiency \cdot share\ of\ roof\ available \cdot panel\ efficiency \cdot FLAECHE\ (m^2) \cdot angle\ tilt\ reduction factor$ (method3.2 omits the angle tilt reduction factor[5]).
- Each sub-data frame is then exported to a parquet file, so I can loop through all subdfs at a later stage if I ever want to make hourly specific calculations per house.

SANITY CHECK:

- The sanity check plays a major role in the code to ensure that simply my technological model calculations are somewhat reasonably close to reality. It proceeds to model run the building of PV installations in the model for a given number of months, using the exact same functions as the later Monte Carlo iterations but storing the results in a different folder.
- **algo.update_gridprem**: This function loops through the topo dictionary and extracts all EGIDs with an installation (built or modeled). It then loops through all the topo_time_subdfs (all HOY for all partitions) and calculates how much the installation produced, how much was self-consumed and how much went into the grid. This data frame is then later aggregated at the grid node level, determining the total feedin per node. If a certain node

[3] Suggested by Hector Ramirez, but showing no (real) variation as there is only a single and multi-family household as an archetype profile. The rest are just level shifts depending on a more sophisticated matching of household characteristics dependent on the building registry. Idea: I could derive a generic variation profile that would add a bandwidth of random deviation from these two profiles for given hours in the year. This would give a more sophisticated data source but still allow for randomness in the hours when electricity is consumed.

[4] A multiplicative reduction factor between 1 and 0 depending on the angle and tilt of the roof partition

[5] A multiplicative reduction factor between 1 and 0 depending on the angle and tilt of the roof partition

threshold limit is exceeded, the premium for feeding in that hour of the year at that node is raised to the corresponding CHF value.

- **algo.update_npv_df**: This function loops through all topo_time_subdfs, extracting pvprod_kW, demand_kW, pv_tarif_Rp_kWh, elecpri_Rp_kWh, prem_Rp_kWh to calculate the economic expenditure (feeding · feedin premium for a given hour + demand · electricity price) and the economic revenue (feeding · PV feeding tariff from grid operator + self consumption of PV production · electricity price). The topo_time_subdf is then aggregated to the partition level (income per partition) and **here now all possible rooftop partition** combinations are created. Next, the net present value is computed for each combination depending on the income and installation cost[6]. The data frame featuring all combinations and NPVs is exported as npv _df for the most current version and per iteration month to see changes over time.

- **select.select_AND_adjust_topology**: This function selects the must current npv_df and selects a row from it depending on the selection method. The NPV-probability-weighted method standardizes the NPVs for all installations by the maximum NPV within the sample. Next a random number between 0 and 1 is selected. The installation with the closest value to the random number is selected and installed. Other selection methods either just pick random installations or the installation featuring the highest NPV. The EGID for the selected roof partition combination is selected in the topo_egid and all the relevant keys are updated (installation capacity, installation id, installation TF switch, date of installation etc). The selected row of npv_df is stored to the pred_inst_df a data frame (stored after each monthly iteration to observe change in selection) and the topo_egid dictionary is overwritten to represent the most recent version of the sample topology.

- Within the iteration loop for assigning future PV installations, the "unit of observation" is months. The **initial.define_construction_capacity** function assigned the construction capacity for each month within a year. After each installation, the total amount of "built installations" is constantly updated upwards, just until the construction capacity for the month is exceeded. Then the while loop enters into the next month. This loop continues

[6]Cost: FLAECHE · kWpeak assumption gives the total kWp value for the installation which is fed into a cost assumption function interpolated form values in the documentation of solar rechner Schweiz. Because the majority of installations is below 60 kWp, I also restrict the cost interpolation to that kWp range from solar rechner schweiz

over the months, until the total capacity for the year is met, which again then causes the year loop to enter the next year. With each transition into the next month / year, the construction capacity is reset again. This is a straightforward way of "spreading out the installations" in a realistic manner over time, see Fig 5. The problem is the mechanism, defining when a monthly/yearly construction capacity is met. Even though the plot looks realistic, longer model runs often have **no installation within the last months of the year**. This is because each small exceedance of construction capacity during the months throughout the year accumulates to a larger amount, causing the construction capacity for the entire year to be triggered prematurely. Simply, lowering the capacity for some months does not really work, because does not affect the additional installation causing the loop to enter the next month (it will still be built, because e.g. 0.001 kWp capacity is left within the month. Maybe it might be worth a try to assign more capacity to the last months of the year, not just the "construction-friendly" summer months.

- **sanity.sanity_check_summary_byEGID**: For sanity checks, this function goes through a some EGIDs[7] and lists all key indicators of the EGID in a excel (e.g. most settings from the default settings, the number of i partitions, total roof area, tilt, angle, feedin from EGID, self consumption etc) to have a quick fact sheet when comparing individual houses e.g. to calculations of online PV investment calculators.

- **sanity.create_gdf_export_of_topology**: For further sanity checks, this function exports all the spatial data into geo-files to then be analysed in QGIS. This way, I can check which solkat partitions are within the sample, plot the entire topology or subselect certain spatial data types (e.g. all single-roof buildings with a flat roof). This function is necessary because the large spatial input files are too large for QGIS to handle efficiently.

- **sanity.check_multiple_xtf_ids_per_EGID**: Similarly to the one before, this function loops through the spatial data points within the sample and checks whether existing PV installations are assigned multiple times to the same EGID, causing double / overcounting.

[7] The list is set by the default settings and can also be extended by some EGIDs that have a modeled installation assigned to them through the model iterations

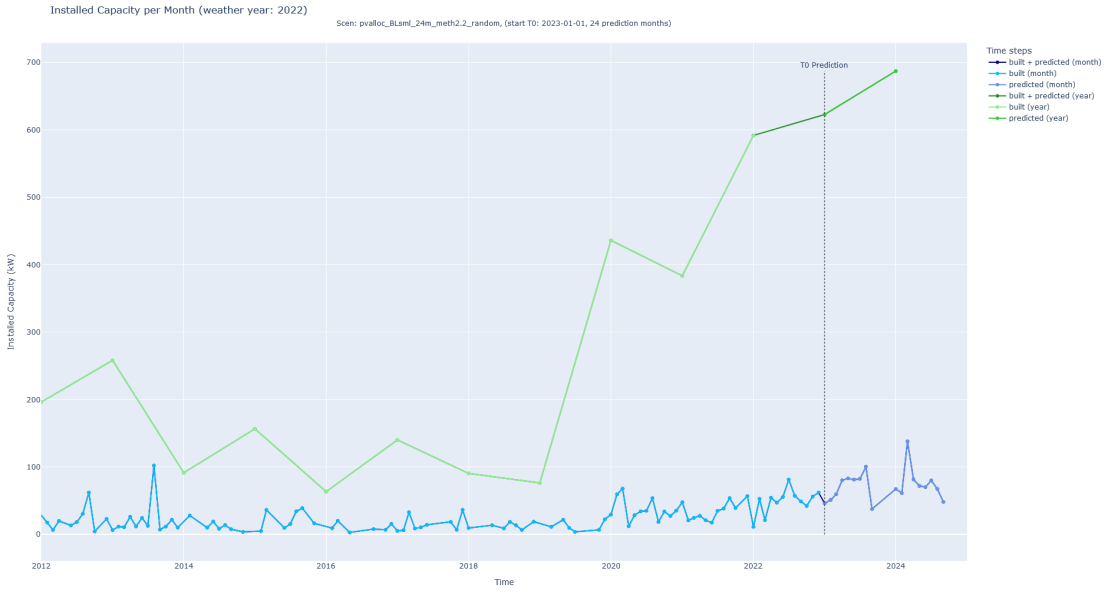

Figure 5: A line graph showing the modeled installed capacity within the sample over time in months and years

1.3 PV Allocation MC ALGORITHM Master

- **pvalloc_initialization_MASTER** is really the main file to starting the allocation of PV installation process. It is split from the Monte Carlo iterations, mainly to save time and clarity because the MC iteration is basically the same as the SANTIY CHECK part (also using exactly the same functions to reduce error potential), but just for more repetitions.

- The pvalloc_MCalgorith_MASTER file picks up the scenario settings from the json file, ensuring the same settings truly stay with the scenario throughout all the model steps. It searches for the scenario (now by name, **no longer just the "pvalloc_run" folder**), creates a subdirectory for each MC iteration and stores all the relevant data files there for a model run.

- Next, the functions **algo.update_gridprem**, **algo.update_npv_df** and **select.select_AND_adjust_topo** are again applied within a loop over each month of the year. The **main difference** is that time range for the model runs are now much longer, resembling actual longterm predictions and not just a single year for sanity checks.

2 Setting Descriptions

2.1 Data Aggregation Settings

2.2 PV Alloc INITIALIZATION Settings

3 Most Current Version & Persistant Flaws