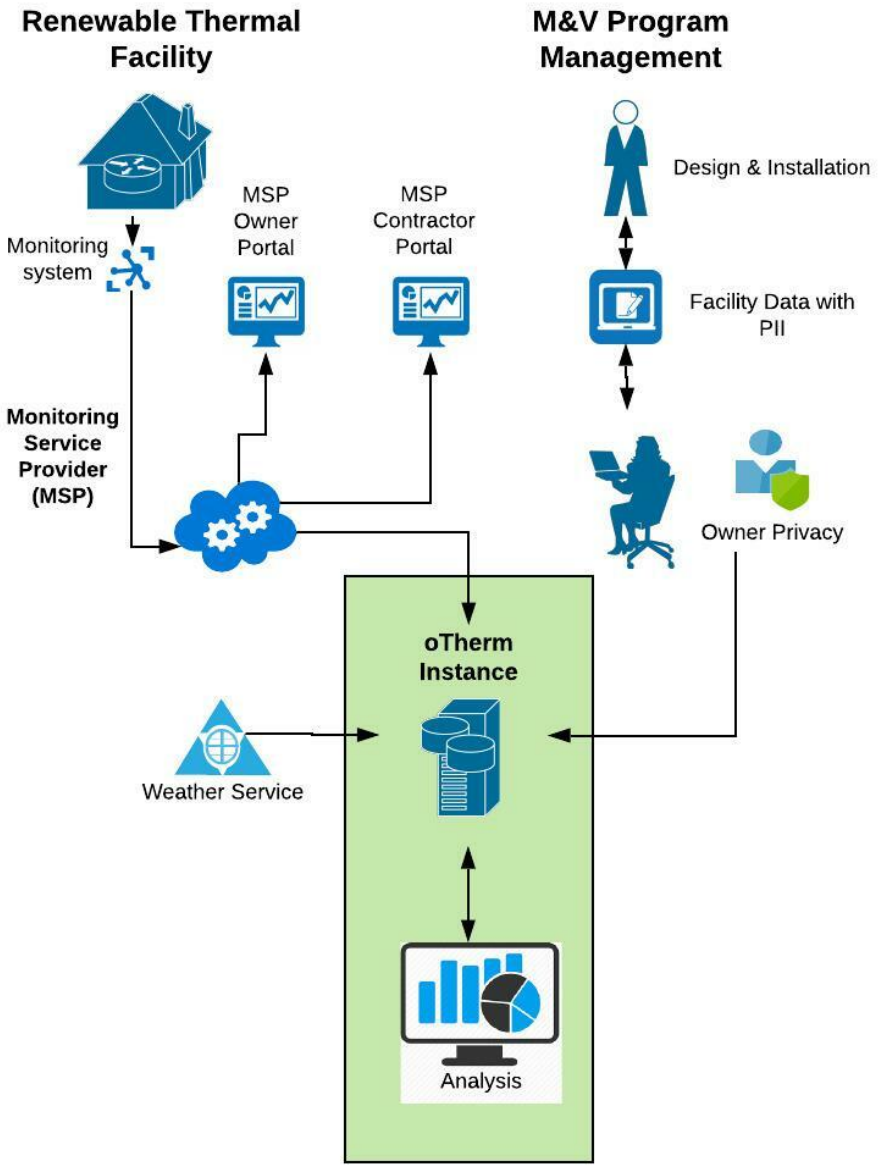
oTherm GSHP analysis

Matt Davis

Sep 22, 2022

CONTENTS:

otherm_gshp is a collection of Python scripts to analyze data from an **oTherm database instance** that use ground source heat pumps as the sole source of renewable thermal energy.



CHAPTER

ONE

OTHERM PROJECT

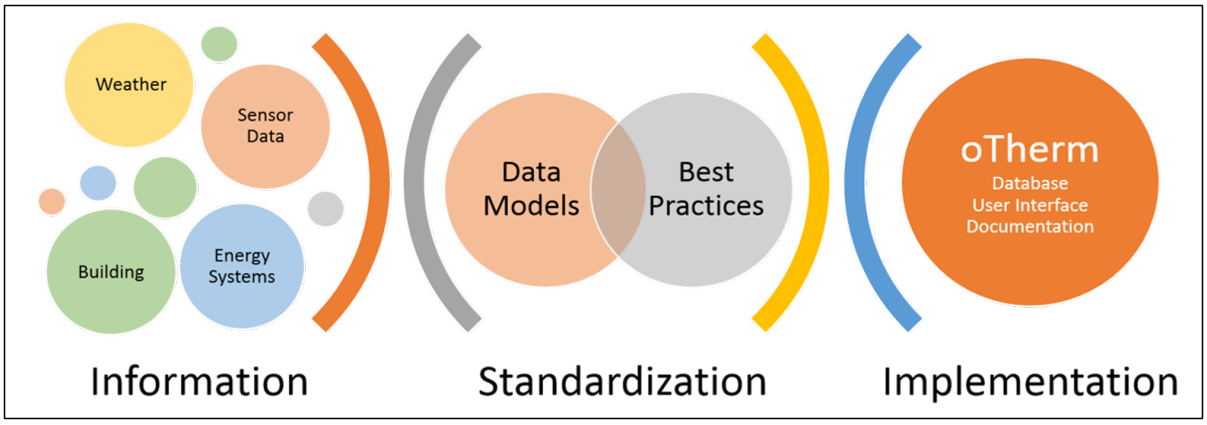
The *oTherm* project aims to standardize data collection methods for renewable thermal energy (RTE) systems, such as heat pumps (air-source and ground-source), solar thermal, and biomass systems. While the initial focus on the project has focused on ground source heat pump systems, the framework is extensible to other types of RTE systems.

Some of the challenges in the monitoring and verification (M&V) of RTE systems include:

- necessity for continuous data (approximately 1-minute resolution) of system energy flows, both thermal and electric power
- need to continously monitor the temperature of the renewable thermal source
- need to contextualize observations with respect to the equipment, the building envelope and outdoor weather conditions
- documentation of sensor accuracies for the quantification of uncertainty, which can be significant in RTE systems

These challenges often limit the number of facilities that can be included in an M&V program, most often ranging from 3 - 24, and make it exceeding difficult to combine data from multiple studies.

The oTherm project establishes a standarized data framework and software necessary to efficiently collect and analyze data within a given M&V program. With the standardization of the data models, data from separate M&V program can be easily included into cross-program analyses.



The initial work has been funded by the Renewable Thermal Alliance and the New York State Energy Research & Development Authority. Continued work has been supported by the US Department of Energy, Office of State Energy Programs. The collaborative research has been led by the University of New Hampshire in coordination with the Yale School for the Environment.

CHAPTER

TWO

MODULES

The modules support the back-end interactions with an oTherm instance. They are provided as examples only.

CHAPTER

THREE

ANALYSIS

This set of analysis modules uses the oTherm APIs to access oTherm data. See the `API Documentation <https://otherm.org/api_documentation> for more details. The more complex oTherm data is organized into dataclasses that use that are ready for analysis. For example,

```
@dataclass
class ThermalLoad:
    uuid: str
    name: str
    description: Optional[str]
    conditioned_area: float
    heating_design_load: float
    cooling_design_load: float
    heating_design_oat: float
    cooling_design_oat: float
```

3.1 analysis package

The analysis packages provided provide some examples of the potential uses of oTherm GSHP data. These are provided as is and have the following limitations:

- Each site has a single heat pump. The analyses presented can, in the future, be extended to multiple pieces of equipment.

3.1.1 Modules

3.1.2 analysis.daily_summaries module

Calculation of daily performance metrics for a given piece of equipment (e.g. a heat pump). This function is also embedded in oTherm instance and can be updated daily as a cron job. Included here to document methods and allow for customization.

Daily metrics include, each integrated for one day:

- Heating and cooling degree days, relative to 65F base temperature
- Heat pump run time (hours)
- Heat pump energy usage (kWh)
- Auxiliary heat energy usage (kWh)

- Thermal energy generated by heat pump under heating and cooling modes (MBtu)
- Thermal energy exchanged with the ground (MBtu)
- Average outdoor air temperature (F)
- Source minimum and maximum temperatures (F)
- Number of monitoring records for day

Note: Modifying this script will not change calculations in oTherm instance. See oTherm instance administrator to update codebase.

---

analysis.daily_summaries.**create_daily_summaries**(*data*, *heatpump_threshold_watts*)

Parameters

- **data** (*pandas.DataFrame*) – Heat pump operating data from oTherm db
- **heatpump_threshold_watts** (*float*) – Threshold to determine if heat pump is on or off.

Returns

- *pandas.DataFrame*
- *The returned DataFrame contains daily summary metrics described above*

3.1.3 analysis.load_summary module

analysis.load_summary.**load_summary_graph**(*site*, *thermal_load*, *ds*)

Parameters

- **ds** (*DataFrame*) – Pandas ddataframe containing daily summaries
- **site** (*dict*) – Dataclass object containing site information

3.1.4 analysis.ewt_violins module

Creates histograms of the heat pump entering water temperature averaged on hourly intervals using the Seaborn library. When multiple site names are provided, histograms are plotted along the x axis and labeled with the site name.

When multiple sites are plotted, the *seaborn.violinplot* parameters are set to prodcue histograms that are equal width with the area of each mode scaled to the relative number of hours in heating or cooling.

analysis.ewt_violins.**determine_mode**(*row*)

Parameters
row – a row in a pandas DataFrame

Returns
new column in the DataFrame, that identifies when heat pump is heating or cooling

analysis.ewt_violins.**ewt_violins**(*site_names*, *start_date*, *end_date*, *db*)

Parameters

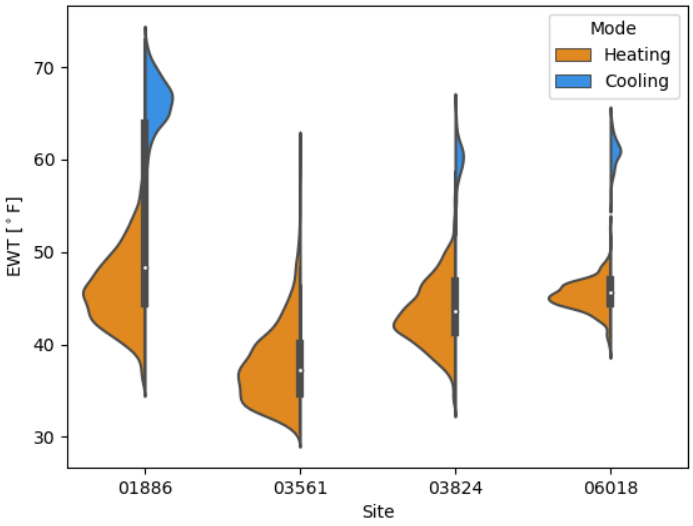
- **site_names** (*list*) – A list of site names to include in analyis. Each site will have it's own violin plot
- **start_date** (*str*) – Start date of analysis in format 'YYYY-MM-DD'

- **end_date** (*str*) – End date of analysis in format 'YYYY-MM-DD'
- **timezone** (*str*) – Timezone of installation
- **db** (*str*) – oTherm database to use for analysis

Returns
The image is written to a file in the ../temp_files directory

Return type
image file

Example output:



3.1.5 analysis.geoexchange_proxies module

3.1.6 analysis.hp_mfr_plots module

3.1.7 analysis.kwh_per_sf module

One particularly helpful analysis that can be accomplished with very simple monitoring equipment is the energy usage as a function of conditioned area and outdoor air temperature. One application of this analysis offers an opportunity to compare the efficiency of different technologies, such as air-source and ground-source heat pumps over a wide range of outdoor weather conditions (e.g., Ueno and Loomis, 2015).

This module creates a scatterplot of energy use intensity as a function of average daily outdoor air temperature.

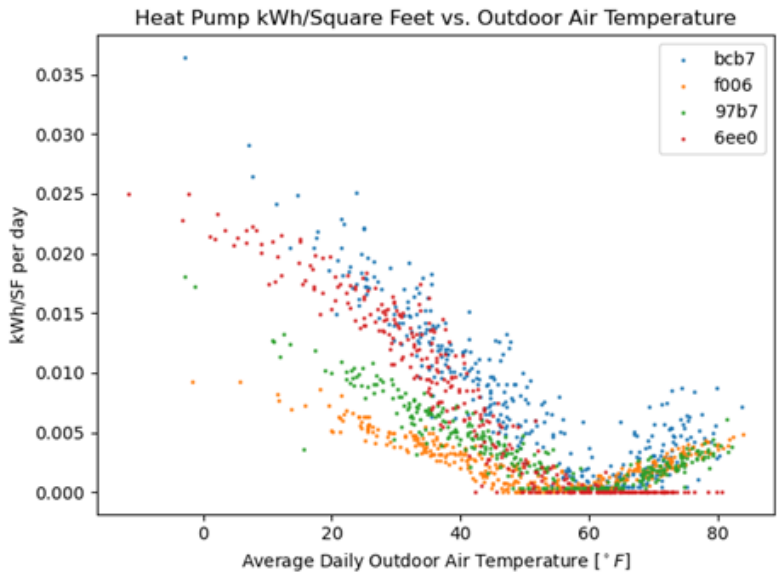
`analysis.kwh_per_sf.`**kwh_vs_oat**(*site_names*, *start_date*, *end_date*, *db*)

Parameters

- **site_names** (*list*) – List of site names, as strings
- **start_date** (*str*) – Beginning date of request, such as *'2015-01-01'*
- **end_date** (*str*) – End date of request
- **symbol_colors** (*dict*) – Dictionary of colors for graph symbols with site name as keys

- **db** (*str*) – The name of the database to pull operating data from

Example output:



3.1.8 analysis.load_factor module

Because adoption of GSHP systems will often replace fossil-fuel fired systems and represent more energy intensive appliances in a home, electric utilities are also interested in the load factors for typical residential GSHP systems and the month-to-month variation in load factors over the course of a year. For the purposes here, the load factor is defined as the ratio of the electricity consumed over a period of time, such as one month, to the consumption that would have occurred if the peak demand operated over the entire month. The load factor ranges from 0 to 1, with higher values representing more uniform and predictable demand.

$$LoadFactor = \frac{kWh_{used\ in\ period}}{kW_{peak} \cdot (hours\ in\ period)}$$

Note: runs correctly with limited oTherm data, still needs full testing

analysis.load_factor.**generate_csv**(*data*, *site_name*)

Parameters

- **data** (*pandas.DataFrame*) – Heat pump operating data with datetime index
- **site_name** (*str*) – Name of the site to analyze. At present, assumes a single heat pump at each site.

Returns

Produces a csv file with the following columns:

month_and_year	year and month of analysis
Compressor (kWh)	total energy consumption of compressor
Auxiliary (kWh)	total energy consumption of compressor
Total Load (kWh)	sum of Compressor and Auxiliary energy consumption
Compressor Peak (hourly)	peak hourly electric power (kW) for compressor
Auxiliary Peak (hourly)	peak hourly electric power (kW) for auxiliary
Total Peak (hourly)	peak hourly electric power (kW) over period
Load Factor (Total)	calculated load factor for month
intervals	number of 1-minute interval data over month
completeness	number of intervals divided by minutes in month

Return type
csv file

3.1.9 analysis.spf_with_uncertainty module

The seasonal performance factor (SPF) is a metric used to evaluate the performance of installed heat pumps. SPF values are sometimes separated into monthly values or values binned on ranges of entering water temperatures.

In heating mode, the SPF is calculated similarly to the COP. The difference is that COP values are determined under laboratory conditions while the SPF values are calculated using real-world operational data. Further, while COPs are measured with laboratory-grade equipment, calculation of SPFs may use estimated or proxy values in lieu of measured values, depending on the availability and quality of data. The heating SPF is calculated as the ratio of the heating or cooling provided and the electricity used to generate the heating or cooling:

$$SPF = \frac{\text{Heating or Cooling Provided [kWh]}}{\text{Electricity Used [kWh]}}$$

When calculating SPF values, it is important to note the boundaries of the analysis. Spitler and Gehlin (2020) build upon the SEPEMO boundaries defined by Nordman and others (2012) to delineate a set of nested boundaries the include successively more components of the system.

Uncertainty Analysis One of the primary challenges in analyzing SPF values and comparing them between systems or with laboratory-rate COP values is the uncertainty associated with measurements used to calculate the SPF values. All measurements have some degree of associated uncertainty, but field measurements used to calculate SPF values generally are obtained with lower quality sensors than those used the laboratory to calculate COP values. As a result, they have a larger uncertainty due to sensor bias. Most studies that report measured performance (COP or SPF) do not quantify uncertainty (e.g., Puttagunta et al., 2010; Huelman et al., 2016) even though it can be significant.

Uncertainty due to sensor bias can be absolute or fractional. Absolute uncertainty has the same units as the value being measured. Fractional uncertainty is a fraction of the measured value. While the sensor bias for a given sensor will be constant, the impact on the uncertainty of the calculated SPF depends on the measured value, which changes in time. This is of particular concern with the uncertainty of a measure of temperature difference.

Calculating the SPF of GSHP systems relies on quantifying the geoexchange (thermal energy exchanged with the subsurface) and the electricity used by the GSHP system. Quantifying the geoexchange requires taking the product of density and specific heat capacity of the heat transfer fluid, the mass flow rate of the heat transfer fluid, and the temperature change of the heat transfer fluid across the heat pump. The uncertainties in the density and specific heat capacity values are very small relative to the other uncertainties and are typically ignored (Spitler et al., in prep). The temperature change of the heat transfer fluid has a constant absolute uncertainty, meaning that the true temperature change is within a fixed number of degrees from the measured value. Electricity usage measurements can have a fractional or absolute uncertainty, depending on the measurement method.

Because the uncertainty of geoexchange and the electrical consumption of the GSHP system (EQ and Ew, respectively) can change depending on the actual conditions, the uncertainty must be calculated separately for each timestep in the

period of interest. Following Taylor (1997), the fractional uncertainty for thermal energy exchanged with the subsurface and the electrical consumption (eQ and eW, respectively) can then be calculated as:

$$e_{Q,n} = \frac{\sum_{i=1}^{n} E_{Q,n}}{\sum_{i=1}^{n} Q_i}$$
$$e_{W,n} = \frac{\sum_{i=1}^{n} E_{W,n}}{\sum_{i=1}^{n} W_i}$$

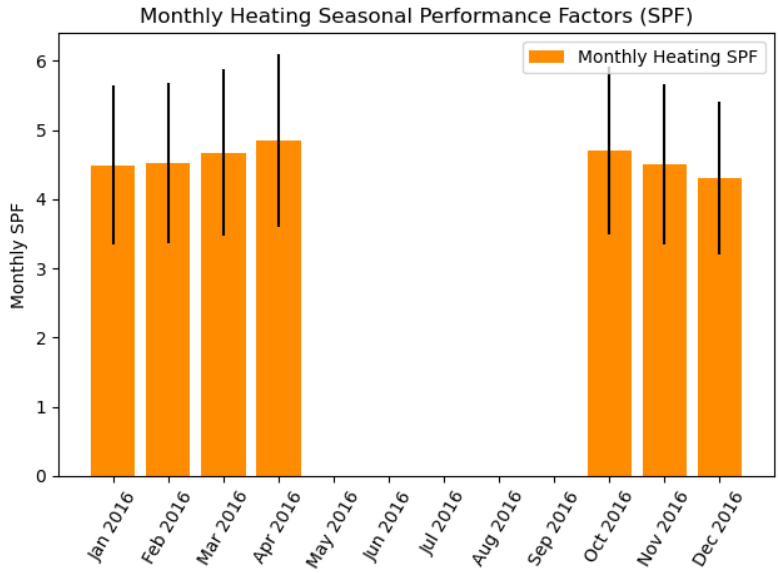
Where Qi and Wi are the measured values of the geoexchange and electrical consumption, respectively. The quantities are summed over n time intervals, typically each 1-minute in duration.

The fractional uncertainty of the SPF value can then be obtained by adding the fractional uncertainties of the thermal energy exchanged with the subsurface and the electrical consumption of the GSHP system in quadrature:

$$e_{SPF,n} = \sqrt{(e_{Q,n}^2 + e_{W,n}^2)}$$

While this description of uncertainty analysis focuses on SPF calculations, as they involve multiple types of measurements, uncertainty analysis should also be performed when calculating and reporting other key performance indicators.

Example output:



@author: Ryan Chase, University of New Hampshire

`analysis.spf_with_uncertainty.`**`lag_temps`**(*initial_data*)

Lag temperature measurements by one value. Necessary for on-pipe measurements. Assumes that operating data is at minute-resolution and thermal response of temperature sensors is approximately one minute.

Parameters

`initial_data` (*`pd.DataFrame`*) – Data as initially pulled from database.

Returns

Dataframe with lagged temperature measurements. Additional column is as follows:

DeltaT	circulating fluid temperature change (as *float*)

Return type

pd.DataFrame

analysis.spf_with_uncertainty.**to_kilowatts**(*data*, *derate*, *power_fac*)

Unit conversion and electricity usage/heat rate adjustments.

Converts heat flow from btu/hr to kW and electricity from W to kW. Scales heat flow and electricity consumption values.

Parameters

- **data** (*pd.DataFrame*) – Heat pump operational data.
- **derate** (*float*) – Scales heat flow values.
- **power_fac** (*float*) – Scales electricity usage.

Returns

Dataframe with unit conversion and necessary scaling applied to electricity usage and heatflow. Additional columns are as follows:

q	heat flow in kW with scaling (as *float*)
kw_used	electricity usage in kW with scaling (as *float*)

Return type
pd.DataFrame

analysis.spf_with_uncertainty.**error_heat_from_ground**(*mrl_hr*, *E_deltaT*, *e_v*, *data*)

Calculates error associated with heat transfer from ground.

Converts heat flow from btu/hr to kW and electricity from W to kW. Scales heat flow and electricity consumption values.

Parameters

- **mrl_hr** (*float*) – Max record length in hours.
- **E_deltaT** (*float*) – Absolute uncertainty in temperature change of circulating fluid.
- **e_v** (*float*) – Flow rate fractional error.
- **data** (*pd.Dateframe*) – Heatpump operational data.

Returns

Dataframe with additional columns associated with error in heat exchange rates. Additional columns are as follows:

e_deltaT	fractional uncertainty deltaT (as *float*)
e_q	fractional uncertainty heat transfer rate (as *float*)
E_q	absolute uncertainty heat transfer rate (as *float*)
tvalue	date time (as *datetime64[ns, UTC]*)
timedelta	time since last timestep (as *timedelta64[ns]*)
elapsed_hours	hours since last timestep (as *float*)
E_Q	absolute uncertainity in kWh to/from ground (as *float*)

Return type
pd.DataFrame

analysis.spf_with_uncertainty.**elec_error_single_elec_measurement**(*e_e*, *error_data*)

Calculates absolute electrical error.

For datasets that include a single electricity consumption value. Calculates absolute electrical error for each timestep in kWh.

Parameters

- **e_e** (*float*) – Fractional uncertainty of electricity usage.
- **error_data** (*pd.Dateframe*) – Heatpump operational data.

Returns

Dataframe with additional column for electricity usage uncertainty. Additional column is as follows:

E_W	absolute electrical uncertainity in kWh (as *float*)

Return type
pd.DataFrame

analysis.spf_with_uncertainty.**heat_calcs_single_elec_measurement**(*error_data*, *pump_power*)

Calculates heat flow and electricity usage during heating periods.

For datasets that include a single electricity consumption value. Considers electricity consumption of single stage circulating pump that will not contribute useful heat to building.

Parameters

- **error_data** (*pd.Dateframe*) – Heatpump operational data.
- **pump_power** (*float*) – Electricity consumption of single stage circulating pump.

Returns

Dataframe with additional columns associated with heatflow and electricity consumption. Additional columns are as follows:

electricity_kWh	electricity kWh used in heating mode (as *float*)
hfg	heat from ground in kWh (as *float*)
pump_power	pump electricity usage in Kw (as *float*)
heat_provided	heat provided to building in kWh (as *float*)

Return type
pd.DataFrame

analysis.spf_with_uncertainty.**total_heat_sum_error**(*spf_heat_data*)

Total values for heating and overall heating spf.

Uses heating operational data. Calculate total heat extracted from ground and absolute error. Also calculates spf and fractional error.

Parameters
spf_heat_data (*pd.Dateframe*) – Heatpump operational data during heating.

Returns

- **total_ground_heat** (*float*) – Total heat from ground in kWh.
- **total_gh_error** (*float*) – Absolute error of total heat from ground in kWh.

- **total_heat_spf** (*float*) – Heating spf.
- **ah_e_spf** (*float*) – Heating fractional uncertainity of SPF.

analysis.spf_with_uncertainty.**monthly_ground_heat**(*spf_heat_data*, *percent_max*)

Calculates monthly heat flow and spf values for plotting.

Resamples data to monthly values. Determines which months have signifigant heating loads to be plotted. Calculates monthly spf values and absolute uncertainty for plotting.

Parameters

- **spf_heat_data** (*pd.Dateframe*) – Heatpump operational data during heating.
- **percent_max** (*float*) – Multipled by highest heating month to determine minimum kWh to plot.

Returns

Dataframe with additional columns associated with monthly heatflow and spf. Additional columns are as follows:

year	year of each timestep (as *str*)
month_and_year	year and month of each timestep (as *str*)
monthly_heating_spf	heating spf value for month (as *float*)
fhe	heating fractional uncertainity (as *float*)
fee	electric fractional uncertainity (as *float*)
e_spf	fractional uncertainity heating SPF (as *float*)
E_spf	absolute uncertainity heating SPF (as *float*)

Return type
pd.DataFrame

3.1.10 analysis.time_of_day_usage module

There is a growing interest in quantifying hourly demand profiles for building heat and cooling to manage generation assets and explore models for demand-response programs (e.g., National Academies, 2021). While heat pump usage patterns tend to vary with season – with winters having higher demand in morning and summer a higher demand in the afternoon – specific usage patterns depend on preferences of building occupants and individual usage patterns. Quantifying patterns of usage across a large number of heat pumps in a given regions will help to inform utilities in forecasting weather-dependent generation patterns and identify opportunities for demand response measures.

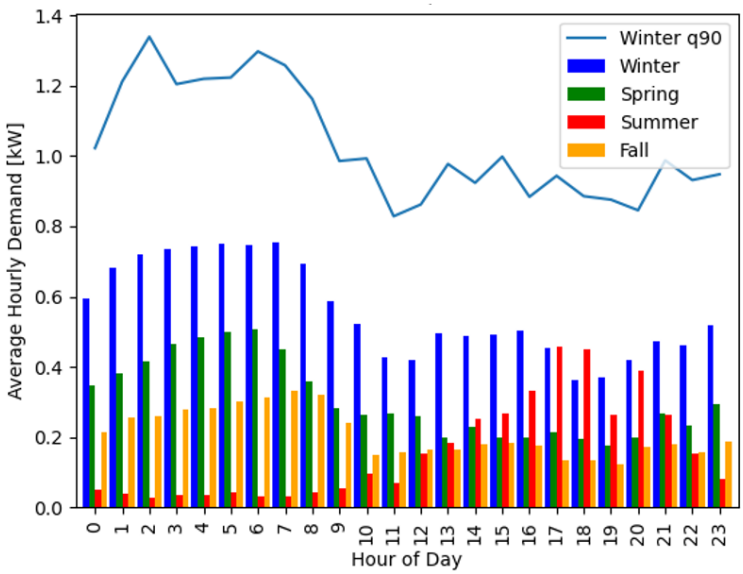
This module calculates the aggregate statistics for kW on an hourly, time-of-day basis Currently calculates mean and 90th quantile for each hour and creates matplotlib plot. It is currently set up for a single year but should be extendable to multiple years.

analysis.time_of_day_usage.**hourly_daily_stats**(*site*, *hp_data*)

Parameters

- **site** –
- **hp_data** –

Example output:


Winter q90
Winter
Spring
Summer
Fall
Average Hourly Demand [kW]
Hour of Day
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23

CHAPTER

FOUR

DB_TOOLS

A collection of scripts that enable uploading and downloading of data from an oTherm instance

4.1 db_tools package

4.1.1 Modules

There are two essential database scripts. The first is to write a set of influxDB line protocol text files that can be uploaded to the oTherm database, and the second is a set of functions that use API requests to retrieve and store data into local *pandas.DataFrame* and *dataclass* objects.

4.1.2 db_tools.influx_lp_writer module

4.1.3 db_tools.otherm_db_reader module

A collection of functions that use oTherm APIs to retrieve data from an oTherm instance. The typical application is to first retrieve the *site* data. Then, using the *site* dataclass object, retrieve information about the:

- *weather_station*,
- *thermal_load*,
- *monitoring_system*, and
- *heat_pump_data*.

The tools also contain scripts for:

- Retrieving the specifications for any oTherm monitoring system by the name of the monitoring system, and
- Retrieving heat pump peformance data from a local SQLite database (*note*, the SQLite database is not part of the oTherm database.

Note: The names and types of data elements used in the analyses differ from the oTherm data model specification.

The *dataclass* objects use for analysis are constructed from json objects returned from the oTherm database. However, because the *dataclass* objects represent a single instance, the data elements are reorganized into a simpler representation than the original json response.

Example

The input typically consists of a site_name and start and end dates. The functions can be called from analyses modules. For example

```
site_name = 'GES649'
start_date = '2015-01-01'
end_date = '2021-01-01'

#Get site information
site = get_site_info(site_name)

#Get equipment information and dataframe of heat pump operating data
equipment, hp_data = get_equipment_data(site.id, start_date, end_date, site.timezone)

#Get monitoring system information and measurement specifications
equip_monitoring_system = get_equipment_monitoring_system(equipment.id)

#Get weather data for station
wx_data = get_weather_data(site.weather_station.nws_id, site.timezone, start_date, end_
↪date)

#Get thermal source specifications
source_specs = get_source_specs(site)
```

db_tools.otherm_db_reader.**get_site_info**(*site_name*, *db*)

get site info docstring

Parameters

site_name (*str*) – name of oTherm site

Returns

The **site** object consists is a nested dataclass object

```
@dataclass
class Site:
    id: int
    name: str
    city: str
    state: str
    timezone: str
    thermal_load: ThermalLoad
    weather_station: WeatherStation
```

To access data elements, use the dot syntax. For example, the Weather Station ID, is accessed by

```
>>> site.weather_station
'KPSM'
```

db_tools.otherm_db_reader.**get_thermal_load**(*site*, *db*)

Dataclass object with equipment specifications ::

@dataclass class ThermalLoad:

uuid: str name: str description: Optional[str] conditioned_area: float heating_design_load: float cooling_design_load: float heating_design_oat: float cooling_design_oat: float

To access data elements, use the dot syntax. For example, the Weather Station ID, is accessed by

db_tools.otherm_db_reader.**get_equipment**(*site_id*, *db*)

Uses 'request' method to read equipment table for a specific site

Parameters

site_id (*int*) – The site_id in the PostgreSQL database. Can be obtained from *site.id*

Returns

Equipment dataclass contains equipment information in the following fields

```
@dataclass
```

class Equipment:

id: int uuid: str model: str description: Optional[str] no_flowmeter_flowrate: float type: int site: int manufacturer: int

db_tools.otherm_db_reader.**get_equipment_data**(*site_id*, *start_date*, *end_date*, *timezone*, *db*)

Uses 'request' method to reads heat pump operating data from otherm influx database and returns a pandas dataframe. The data DataFrame returned includes all records for the equipment at a site. At present, the script is limited to a single piece of equipment at a site.

Parameters

- **site_id** (*int*) – The site_id in the PostgreSQL database. Can be obtained from *site.id*
- **start_date** (*str*) – start date (e.g. 2018-1-1)
- **end_date** (*str*) – end date (e.g. 2018-12-31)
- **timezone** (*str*) – (e.g. 'US/Eastern')

Returns

Equipment dataclass contains equipment information in the following fields:

```
@dataclass
class Equipment:
    id: int
    uuid: str
    model: str
    description: Optional[str]
    no_flowmeter_flowrate: float
    type: int
    site: int
    manufacturer: int
```

pandas.DataFrame containing heat pump operating data over the specified time range. The DataFrame contains all fields stored for the piece of equipment in the influxDB database.

Note: The index of the *DataFrame* is set to the `time` field and localized according the `site.timezone` attribute

---

db_tools.otherm_db_reader.**get_equipment_monitoring_system**(*equip_id*)

Retrieves the equipment monitoring system and specifications

Parameters
uuid (*str*) – *uuid* of thermal equipment

Returns

Dataclass object with equipment monitoring system specifications

```
@dataclass
class MonitoringSysInfo:
    id: int
    name: Optional[str]
    description: Optional[str]
    specs: list

@dataclass
class EquipmentMonitor:
    id: int
    start_date: str
    end_date: Optional[str]
    equip_id: int
    monitoring_system_spec: int
    info: MonitoringSysInfo
```

To access data elements, use the dot syntax. For example, the *list* containing the monitoring system specifications can be accessed by

```
>>> monitoring_system.info.specs
`[{'measurement_spec': {'name': 'HPP VA W 8% EP', 'description': 'Heat pump power,␣
↪volt-amps, electrical panel', ...`
```

The monitoring system specifications is a list of measurements performed by the monitoring system, each measurement has its own set of specifications. See oTherm documentation for more details.

The list can be search for individual measurements specifications with `utilities.get_measurement_specs`

`db_tools.otherm_db_reader.`**get_weather_data**(*nws_id*, *timezone*, *start_date*, *end_date*)

Parameters

- **nws_id** (*str*) – National Weather Station 4 character station identifier
- **timezone** (*str*) – Timezone of site, such as *'US/Eastern'*
- **start_date** (*str*) – Beginning date of request, such as *'2015-01-01'*
- **end_date** (*str*) – End date of request

Returns

- ***pandas.DataFrame***
  - *The returned DataFrame contains weather station data over the specified time range and contains all fields stored for the weather station.*

Note: The index of the *DataFrame* is set to the `time` field and localized according the `site.timezone` attribute

---

`db_tools.otherm_db_reader.`**get_source_specs**(*site*)

Retrieves the source specifications.

Parameters
site (*str*) – site name

Returns

Dataclass object with source specifications

```
@dataclass
class SourceSpec:
    site: str
    site_id: int
    source_name: str
    source_type: str
    description: str
    freeze_protection: Optional[float]
    grout_type: Optional[str]
    formation_conductivity: Optional[float]
    formation_type: Optional[str]
    grout_conductivity: Optional[float]
    antifreeze: Optional[str]
    pipe_dimension_ratio: Optional[str]
    n_pipes_in_circuit: Optional[int]
    n_circuits: Optional[int]
    total_pipe_length: Optional[float]
```

To access data elements, use the dot syntax.

Note: While the oTherm data model supports multiple types of sources, this db_reader tool only supports the vertical loop spec at present.

---

db_tools.otherm_db_reader.**get_mfr_data**(*parameters*)

db_tools.otherm_db_reader.**get_monitoring_system**(*name*)

Similar to *get_equipment_monitoring_system()* but returns monitoring_system attributes for a given monitoring system by name rather than equipment being monitored. This function requires the exact name of the monitoring system, as specified in the oTherm database

Parameters
name (*str*) – The name of the monitoring system

Returns
All specifications of a monitoring system in the oTherm database. Refer to oTherm documentation for detais.

Return type
dict

For more explanation of the parameters and return values, see *get_equipment_monitoring_system()*

4.1.4 db_tools.csv_to_yaml module

db_tools.csv_to_yaml.**output_yaml**(*equipment_model*, *site_model*, *thermal_load_model*)

CHAPTER

FIVE

OTHERM DATABASE FIELDS

	Data Element	Units	Name	Type
	Timestamp	UTC	Datetime	time
1	GSHP compressor	W	heatpump_power	electric power
2	Auxiliary back up heat (Electric)	W	heatpump_aux	electric power
3	Entering water temperature	^{0}F	source_supplytemp	temperature
4	Leaving water temperature	^{0}F	source_returntemp	temperature
5	Source temperature difference	^{0}F	source_tempdiff	temperature difference
6	Source fluid pump power	W	sourcefluid_pump_power	electric power
7	Source fluid flow rate	gpm	sourcefluid_flowrate	fluid flow rate
8	Load circulating pumps/fan	W	load_pumpsfans	electric power
9	Load supply temperature	^{0}F	load_supplytemp	temperature
10	Load return temperature	^{0}F	load_returntemp	temperature
11	Load temperature difference	^{0}F	load_tempdiff	temperature difference
12	Thermostat set point	^{0}F	tstat_set	equipment state
13	Thermostat temp	^{0}F	tstat_temp	equipment state
14	Percent full load	%	compressor_stage	equipment state
15	Heat meter load	BTU	loadside_heat	thermal energy
16	Heat meter ground loop	BTU	sourceside_heat	thermal energy

CHAPTER

SIX

CREDITS AND DISCLAIMERS

6.1 About the oTherm Project

The oTherm project was initiated as part of discussions in the Renewable Thermal Alliance (RTA) and initiated with an RTA Innovation Grant to the University of New Hampshire (UNH) to begin work on data dictionaries. Subsequently, UNH partnered with the New York State Research and Development Authority (NYSERDA) and the Yale Center for Business and the Environment to secure a grant from the U.S. Department of Energy, Office of State Energy Programs(DE-EE0008619) to build a functional prototype. The UNH Interoperability Lab has been instrumental in the development work throughout the process.

6.2 Disclaimer

These materials were prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights.

Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

d

INDEX

O

T