pvpumpingsystem

Release 1.0

Oct 26, 2020

Contents:

pvpumpingsystem is a package providing tools for modeling and sizing offgrid photovoltaic water pumping systems. It is specially designed for small to medium size systems, the type of pumping system typically used for isolated communities.

It can model the whole functioning of such pumping system on an hourly basis and eventually provide key financial and technical findings on a year. Conversely it can help choose some elements of the pumping station depending on output values wanted (like daily water consumption and acceptable risk of water shortage). Find more on the scope of the software in the section *Package Overview*.

The source code for pvpumpingsystem is hosted on GitHub: https://github.com/tylunel/pvpumpingsystem

The package was originally developped at T3E research group, in Ecole de Technologie Superieure, Montreal, Qc, Canada, by Tanguy Lunel.

The software is published under the open source license GPL-v3.

CHAPTER 1

Package Overview

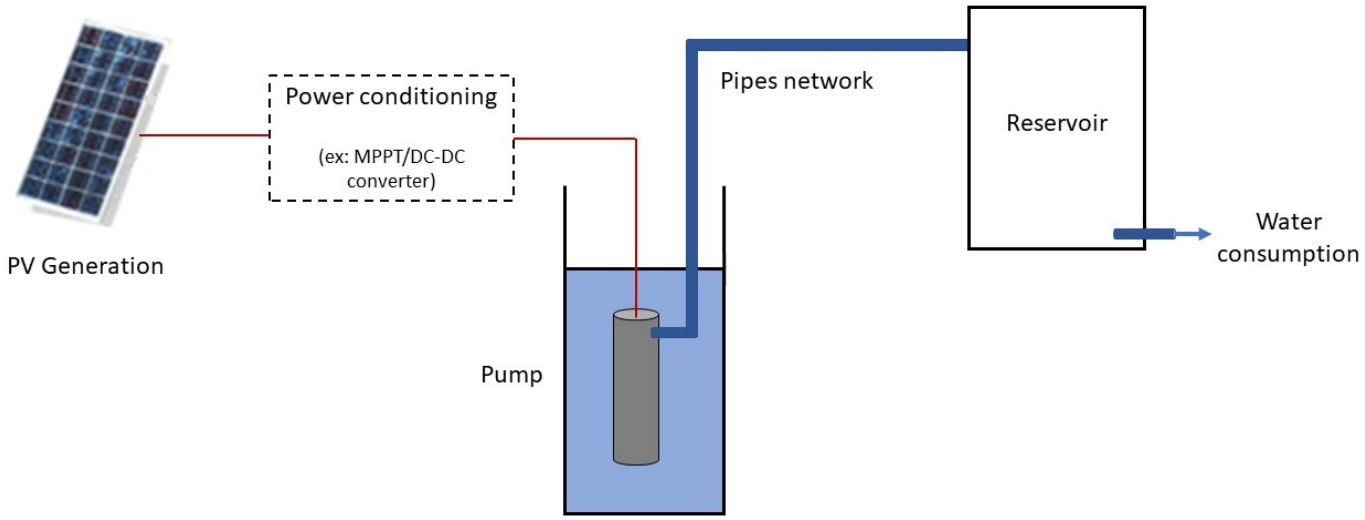
1.1 Introduction

1.1.1 Scope

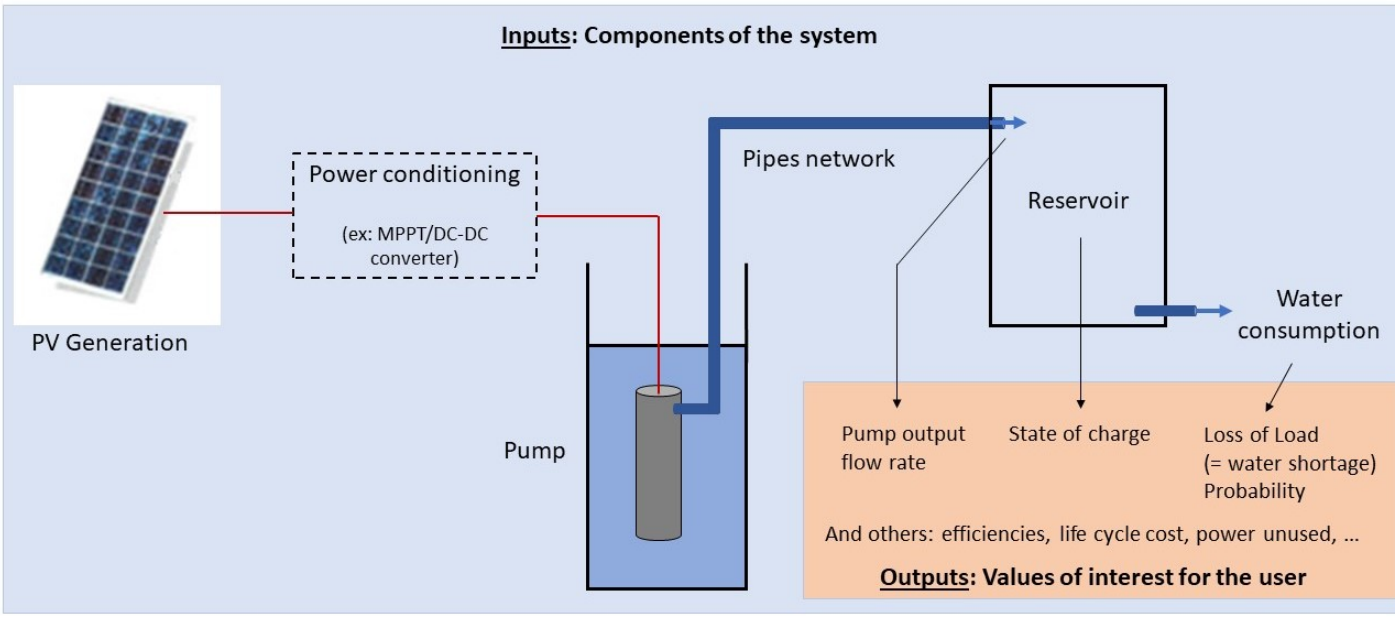
Pvpumpingsystem is an open source package providing various tools aimed at facilitating the modeling and sizing of photovoltaic powered water pumping systems.

This package helps users to model, test and validate different photovoltaic pumping systems before actually installing it in situ. In order to guide the designer in her/his choice, *pvpumpingsystem* provides both technical and financial information on the system. Even though the package is originally targeted at researchers and engineers, three practical examples are provided in order to help anyone to use pvpumpingsystem

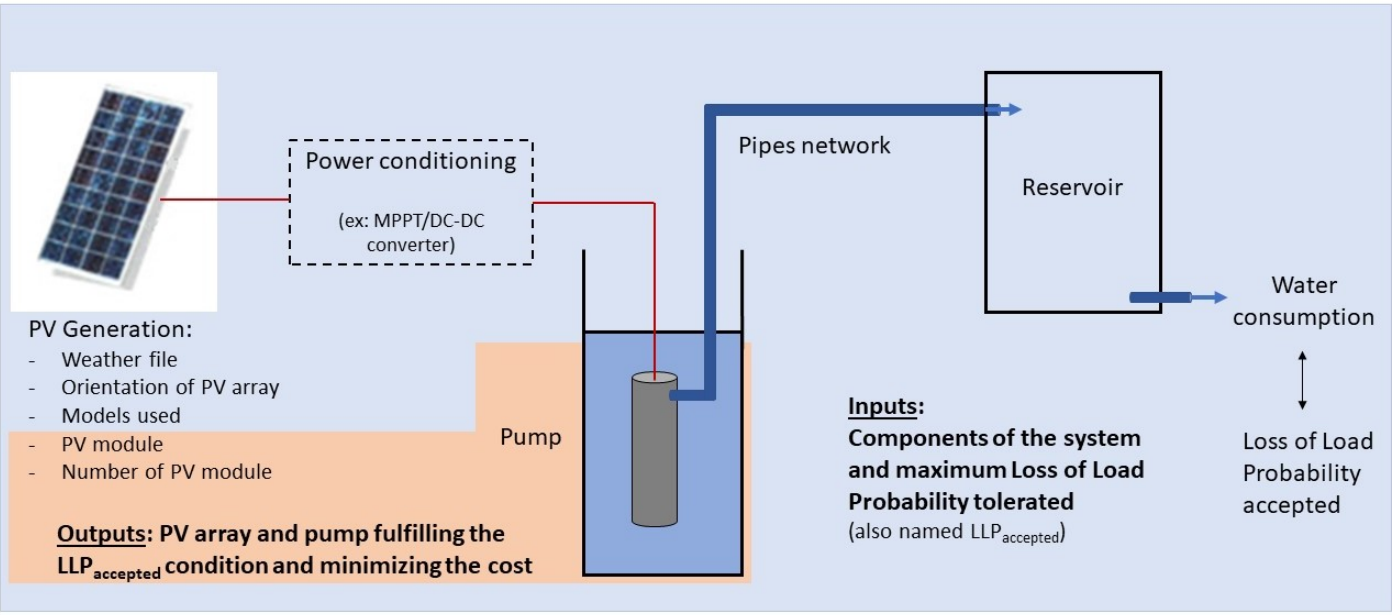
It models pumping systems minimally made of PV generator, DC motor-pump and pipes. Each component can be precisely defined by the user in such a way it corresponds closely to any actual system wanted. User can choose to add a MPPT/DC-DC converter to increase the energy yield of the PV array or to directly couple PV array and motor-pump. The software also allows to add water tank to mitigate the effect of intermittency.



The simulation eventually compute numerous outputs like hourly flow rates of a given pump, efficiencies of components, risk of water shortage and life cycle cost of the whole system.



Pvpumpingsystem also offers to automate the process of sizing. In this case, the user can provide a set of PV module, a set of motor-pumps and a water needs file, and the software looks for the cheapest assembly while making sure that it respects a minimum risk of water shortage.



Nevertheless, the number of sizing processes can be infinite, and this module is expected to significantly expand with time, welcoming new sizing process based on different selection criteria or algorithms. In particular, the reservoir size, the orientation of the PV array, the coupling strategy or even the diameter of pipes are inputs that could ultimately become outputs of the sizing process as well.

To better understand the possibilities of *pvpumpingsystem* and how it works, you are invited to consult the examples available in the form of Jupyter Notebook in *Examples* or the corresponding python files in `docs/examples`.

1.1.2 Code characteristics

Python is the programming language used in the software, and the code is structured within an object-oriented approach. Continuous integration services allow checking for lint in the code and to automatize the tests. Each class and function are documented in the docstring with reference to the literature when applicable.

In *pvpumpingsystem*, in order to increase the understandability of the code, the physical components of the PV pumping system corresponds to a class when possible, like for example the classes Pump(), MPPT(), PipeNetwork(), Reservoir() and PVGeneration(). Moreover, each of these classes are gathered into separate modules with appropriate names (*pump.py*, *mppt.py*, etc). The previous objects are then gathered in the class PVPumpSystem() which allows running partial or comprehensive modeling of the pumping system.

A separate module *sizing.py* is dedicated to functions allowing to size these systems. These functions are globally numerical methods, relying on numerous simulations run according to an algorithm or to a factorial design. *sizing.py* module can be expanded a lot as many strategies can be imagined to size such a system.

Pvpumpingsystem relies on already existing packages for photovoltaic and fluid mechanics modeling, namely *pvlib-python* and *fluids*. *pvpumpingsystem*'s originality lies in the implementation of various motor-pump models for finite power sources and in the coupling of the distinct components models.

Pvpumpingsystem is released under a GPL-v3 license.

1.2 Databases accessible

The PV module database of the California Energy Commission (CEC) is made accessible through PVGeneration (being itself a wrapper of pvlib-python). As this database is nearly comprehensive (more than 22,000 modules)

and regularly updated, it was considered that having a function to define its own PV module was not relevant yet. Therefore, PV modules must be declared by giving the reference in the corresponding attribute in declaration of any PVGeneration instance.

Furthermore, the package also provide some pump and weather files in the folder `pvpumpingsystem/data`.

Concerning pump files, a template is provided in the folder in order to help anyone fill it in with the specification of the pump they want to model. A limited database coming from the company SunPumps is also accessible. Nevertheless, it does not mean that the developers particularly encourage their use, it rather reflects the difficulty to find other sources easily accessible online. Any addition to the database is warmly welcomed here.

The weather files consist in a very restricted list of .epw files coming from diverse climates and that users can exploit to learn and test the software. Similar files for many location around the world are available at EnergyPlus website, or can be constructed using PVGIS.

1.3 Getting support and contribute

If you need help, you think you have discovered a bug, or if you would like to edit *pvpumpingsystem*, then do not hesitate to open an issue on our GitHub issues page or on our GitHub pull request page.

1.4 Credits

The T3E research group would like to acknowledge Mr. Michel Trottier for his generous support, as well as the NSERC and the FRQNT for their grants and subsidies. We also acknowledges the contributions and fruitful discussions with Louis Lamarche and Sergio Gualteros that inspired and helped with the current work.

CHAPTER 2

Installation

Installing pvpumpingsystem can be done through different processes. Two of them are detailled here, mainly thought for newcomers. Experienced users can modify it to their liking.

> For people uncomfortable with package management, but who still plan on contributing or editing the code, follow the *Install pvpumpingsystem with Anaconda and Git* instructions to install pvpumpingsystem along with Anaconda and Git.
>
> For people only interested in the use of the package, follow the *Install pvpumpingsystem alone* instructions to install pvpumpingsystem alone.

Installing pvpumpingsystem is similar to installing most scientific python packages, so in case of trouble see the *References* section for further help.

Please see the *Compatibility* section for information on the optional packages that are needed for some pvpumpingsystem features.

2.1 Install pvpumpingsystem with Anaconda and Git

- Anaconda:

The Anaconda distribution is an open source distribution providing Python and others softwares and libraries useful for data science. Anaconda includes many of the libraries needed for pvpumpingsystem (Pandas, NumPy, SciPy, etc). Anaconda is especially recommended when using Windows.

Anaconda Python distribution is available at https://www.anaconda.com/download/.

See What is Anaconda? and the Anaconda Documentation for more information.

- Git:

Git is a version control system that widely help contribution and development for open source softwares. Git should be native on most of Linux distribution, but must be installed on Windows.

Git for Windows is available at https://gitforwindows.org/.

- pvpumpingsystem:

Once you have Anaconda and git installed, open a command line interface ('Anaconda Prompt' on Windows, terminal in Linux and macOS), change directory to the one you want to install pvpumpingsystem in, and type:

```
pip install -e git+https://github.com/tylunel/pvpumpingsystem#egg=pvpumpingsystem
```

- Test pvpumpingsystem:

To ensure *pvpumpingsystem* and its dependencies are properly installed, run the tests by going to the directory of pvpumpingsystem and by running pytest:

```
cd <relative/path/to/pvpumpingsystem/directory>
pytest
```

2.2 Install pvpumpingsystem alone

Note: Even if you decide not to use Anaconda or Git, you minimally need a Python version superior to 3.5, and to have pip and setuptools installed (installed by default with recent version of Python).

This second option simply uses pip:

```
pip install pvpumpingsystem
```

If you have troubles with the use of pip, here is the pip documentation to help you.

To ensure *pvpumpingsystem* and its dependencies are properly installed, you can consult the package information through pip:

```
pip show pvpumpingsystem
```

2.3 Compatibility

pvpumpingsystem is compatible with Python 3.5 and above.

Besides the libraries contained in Anaconda, *pvpumpingsystem* also requires:

- pvlib-python
- fluids
- numpy-financial

The full list of dependencies is detailled in setup.py.

2.4 References

Note: This section was adapted from the pvlib-python documentation. Thanks to them for this useful listing!

Here are a few recommended references for installing Python packages:

- Python Packaging Authority tutorial

- Conda User Guide

Here are a few recommended references for git and GitHub:

- The git documentation: detailed explanations, videos, more links, and cheat sheets. Go here first!
- Forking Projects
- Fork A Repo
- Cloning a repository

CHAPTER 3

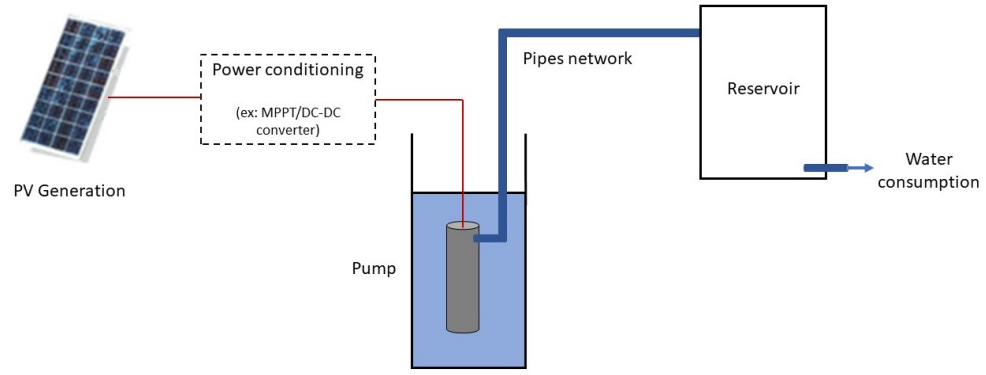
Getting started

To begin, a global view of how the code may be used is given here. Afterward, it is recommended to go through the two first examples provided below in order to get further step-by-step explanations.

3.1 General layout

The code make use of the possibilities provided by the object-oriented paradigm of python. In particular, the code tries to match physical components with objects (in the computer sense of the term) as much as possible.

Therefore, modeling a PV pumping system requires to start by defining each object, i.e each component represented in below diagram.



Once all components have been declared in objects, they are gathered in a parent object (an instance of class PVPumpSystem) where the type of coupling between PV array and pump is declared. Ultimately, the method `run_model()` launches the whole simulation. It is also the step where the financial parameters can be provided if a cost analysis is wanted.

Afterward, the results are contained in the object PVPumpSystem. At this point, it is useful to have an IDE like Spyder or Pycharm to explore the values internally contained. Otherwise, most of the results are actually stored in the attributes `flow`, `efficiency`, `water_stored`, `npv`, `llp`.

The examples below, in particular the jupyter notebook ones, provide further details on each step of a standard simulation.

3.2 Examples

Three examples of how the software can be used are in the folder `docs/examples`. The examples are provided under two forms, as Jupyter Notebook files or as Python files.

3.2.1 Jupyter Notebook

Following examples can be run locally with Jupyter Notebook, or by clicking on the corresponding icon in the upper right-hand corner of nbviewer pages, or by accessing through the binder build.

Simulation

The first two examples focus on simulation. These examples are important to understand because the modeling tools used here are the core of the software. These tools can be used later to get programs that fit a more particular use (for ex.: sizing process, parametric study, etc). For a given system, the examples show how to obtain the values of interest for the user (output flow rates, total water pumped in a year, loss of load probability (llp), net present value (npv), efficiencies and others):

Basic usage example

More advanced usage example

Once you went through these 2 examples, you are quite ready to dive into the code and adapt it to your needs.

Sizing

The third example shows how to use a sizing function written from the modeling tools presented in the two examples above. This function aims at optimizing the selection of the pump and the PV module, based on user requirements.

Sizing example

3.2.2 Python files

These examples are also available in the form of python files in order to freely adapt the code to your wishes. Directly check out in `docs/examples`.

CHAPTER 4

API reference

4.1 Classes

The different classes of *pvpumpingsystem*.

pvgeneration.PVGeneration(... [, ...])	Class representing the power generation through the photovoltaic system.
mppt.MPPT([efficiency, price, idname, ...])	Class defining a DC/DC converter with a MPPT controller.
pump.Pump(path[, ...])	Class representing a motor-pump.
pipenetwork.PipeNetwork(h_stat, l_tot, diam)	Class representing a simple hydraulic network.
reservoir.Reservoir([size, water_volume, ...])	Class defining a water tank with its main characteristics.
consumption.Consumption([flow_rate, ...])	The Consumption class defines a consumption schedule, typically through a year.
pvpumpsystem.PVPumpSystem(pvgeneration, ...)	Class defining a PV pumping system made of:

4.1.1 pvpumpingsystem.pvgeneration.PVGeneration

class pvpumpingsystem.pvgeneration.**PVGeneration**(*weather_data_and_metadata*, *pv_module_name*, *price_per_watt=nan*, *surface_tilt=0*, *surface_azimuth=180*, *albedo=0*, *modules_per_string=1*, *strings_in_parallel=1*, *racking_model='open_rack'*, *losses_parameters=None*, *surface_type=None*, *module_type='glass_polymer'*, *glass_params={'K': 4, 'L': 0.002, 'n': 1.526}*, *pv_database_name='cecmod'*, *orientation_strategy=None*, *clearsky_model='ineichen'*, *transposition_model='isotropic'*, *solar_position_method='nrel_numpy'*, *airmass_model='kastenyoung1989'*, *dc_model='desoto'*, *ac_model='pvwatts'*, *aoi_model='physical'*, *spectral_model='no_loss'*, *temperature_model='sapm'*, *losses_model='pvwatts'*, ***kwargs*)

Class representing the power generation through the photovoltaic system. It is a container of pvlib.ModelChain [1].

pv_module_name
The name of the PV module used. Should preferentially follow the form: '(company_name)_(reference_code)_(peak_power)'

Type str,

weather_data_and_metadata
Path to the weather file if it is .epw file, or the weather data itself otherwise. In the latter case, the dict must contains keys 'weather_data' and 'weather_metadata'. It should be created prior to the PVGeneration with the help of the corresponding pvlib function: (see https://pvlib-python.readthedocs.io/en/stable/api.html#io-tools). Possible weather file formats are numerous, including tmy2, tmy3, epw, and other more US related format. Note that Function 'get_pvgis_tmy' allows to get a tmy file according to the latitude and longitude of a location.

Type str or dict (containing pd.DataFrame and dict),

price_per_watt
Price per watt for the module referenced by pv_module_name [US dollars]

Type float, default is 2.5

surface_tilt
Angle the PV modules have with ground [°] Overwritten if orientation_strategy is not None.

Type float, default is 0

surface_azimuth
Azimuth of the PV array [°] Overwritten if orientation_strategy is not None.

Type float, default is 180 (oriented South)

albedo
Albedo of the soil around.

Type float, default 0

modules_per_string
Number of module put in a string.

Type integer, default is 1

strings_in_parallel
Number of PV module strings. Note that 'strings_in_parallel' is called 'strings_per_inverter' in pvlib.PVSystem. Name has been changed to simplify life of beginner user, but will complicate life of intermediate user.

Type integer, default is 1

racking_model
The type of racking for the PV array.s

Type str, default is 'open_rack'

system
A `PVSystem` object that represents the connected set of modules, inverters, etc. Uses the previous attributes.

Type pvlib.PVSystem

location
A `Location` object that represents the physical location at which to evaluate the model.

Type Location

orientation_strategy
The strategy for aligning the modules. If not None, overwrites the `surface_azimuth` and `surface_tilt` properties of the `system`. Allowed strategies include 'flat', 'south_at_latitude_tilt'. Ignored for SingleAxisTracker systems.

Type None or str, default None

clearsky_model
Passed to location.get_clearsky.

Type str, default 'ineichen'

transposition_model
Passed to system.get_irradiance.

Type str, default 'haydavies'

solar_position_method
Passed to location.get_solarposition.

Type str, default 'nrel_numpy'

airmass_model
Passed to location.get_airmass.

Type str, default 'kastenyoung1989'

dc_model
If None, the model will be inferred from the contents of system.module_parameters. Valid strings are 'desoto' and 'cec', unlike in pvlib.ModelChain because PVPS modeling needs a SDM.

Type None, str, or function, default None

ac_model
If None, the model will be inferred from the contents of system.inverter_parameters and system.module_parameters. Valid strings are 'snlinverter', 'adrinverter', 'pvwatts'. The ModelChain instance will be passed as the first argument to a user-defined function.

Type None, str, or function, default None

aoi_model
If None, the model will be inferred from the contents of system.module_parameters. Valid strings are 'physical', 'ashrae', 'sapm', 'martin_ruiz', 'no_loss'. The ModelChain instance will be passed as the first argument to a user-defined function.

Type None, str, or function, default None

spectral_model
If None, the model will be inferred from the contents of system.module_parameters. Valid strings are 'sapm', 'first_solar', 'no_loss'. The ModelChain instance will be passed as the first argument to a user-defined function. 'no_loss' is recommended if the user is not sure that the weather file contains complete enough information like for example 'precipitable_water'.

Type None, str, or function, default 'no_loss'

temperature_model
Valid strings are 'sapm' and 'pvsyst'. The ModelChain instance will be passed as the first argument to a user-defined function.

Type None, str or function, default None

losses_model
Valid strings are 'pvwatts', 'no_loss'. The ModelChain instance will be passed as the first argument to a user-defined function.

Type str or function, default 'no_loss'

name
Name of ModelChain instance.

Type None or str, default None

****kwargs**
Arbitrary keyword arguments. Included for compatibility, but not used.

Reference

----

[1] William F. Holmgren, Clifford W. Hansen, Mark A. Mikofski,

"pvlib python

Type a python package for modeling solar energy systems",

2018, Journal of Open Source Software

__init__(*weather_data_and_metadata*, *pv_module_name*, *price_per_watt=nan*, *surface_tilt=0*, *surface_azimuth=180*, *albedo=0*, *modules_per_string=1*, *strings_in_parallel=1*, *racking_model='open_rack'*, *losses_parameters=None*, *surface_type=None*, *module_type='glass_polymer'*, *glass_params={'K': 4, 'L': 0.002, 'n': 1.526}*, *pv_database_name='cecmod'*, *orientation_strategy=None*, *clearsky_model='ineichen'*, *transposition_model='isotropic'*, *solar_position_method='nrel_numpy'*, *airmass_model='kastenyoung1989'*, *dc_model='desoto'*, *ac_model='pvwatts'*, *aoi_model='physical'*, *spectral_model='no_loss'*, *temperature_model='sapm'*, *losses_model='pvwatts'*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(weather_data_and_metadata, ... [, ...])	Initialize self.
run_model()	Runs the modelchain of the PV generation.

Attributes

pv_module_name
weather_data_and_metadata

4.1.2 pvpumpingsystem.mppt.MPPT

class pvpumpingsystem.mppt.**MPPT**(*efficiency=0.96*, *price=nan*, *idname='default'*, *output_voltage_available=None*, *input_voltage_range=None*)

Class defining a DC/DC converter with a MPPT controller.

efficiency
Mean efficiency if float. Efficiency according to power if array.

Type float, default is 0.96

price
Price of the MPPT

Type float, default is 'nan'

idname
Name of the MPPT

Type str, default is 'default'

output_voltage_available
Correspond to the list of keys of 'input_voltage_range'

Type list, default is None

input_voltage_range
Input voltage range given as value (tuple) for each output voltage available given as key (float).

Type dict, default is None

__init__(*efficiency=0.96*, *price=nan*, *idname='default'*, *output_voltage_available=None*, *input_voltage_range=None*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__([efficiency, price, idname, ...])	Initialize self.

4.1.3 pvpumpingsystem.pump.Pump

class pvpumpingsystem.pump.**Pump**(*path*, *motor_electrical_architecture=None*, *idname=None*, *price=nan*, *controller=None*, *diameter_output=None*, *modeling_method='arab'*)

Class representing a motor-pump.

path
The path to the .txt file with the pump specifications.

Type str, default=''

motor_electrical_architecture
'permanent_magnet', 'series_excited', 'shunt_excited', 'separately_excited'.

Type str, default is None

modeling_method
name of the method used for modeling the pump.

Type str, default is 'arab'

idname
name of the pump

Type str, default is None

price
The price of the pump

Type numeric, default is None

controller
Name of controller

Type str, default is None

voltage_list
list of voltage (the keys of preceding dictionaries) [V]

Type None or list,

specs

Dataframe with columns of following numeric: 'voltage': voltage at pump input [V] 'current': current at pump input [A] 'power': electrical power at pump input [W]] 'tdh': total dynamic head in the pipes at output [m] 'flow': pump output flow rate [liter per minute]

Type None or pandas.DataFrame,

data_completeness
Provides some figures to assess the completeness of the data. (for more details, see pump.specs_completeness())

Type None or dict,

__init__ (*path*, *motor_electrical_architecture=None*, *idname=None*, *price=nan*, *controller=None*, *diameter_output=None*, *modeling_method='arab'*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(path[, ...])	Initialize self.
functIforVH()	Function computing the IV characteristics of the pump depending on head H.
functIforVH_Arab()	Function using Hadj Arab model for modeling I vs V of pump.

Continued on next page

Table 5 – continued from previous page

functIforVH_Kou()	Function using Kou model for modeling I vs V of pump.
functIforVH_theoretical()	Function using electrical architecture for modeling V vs I of pump.
functQforPH()	Function computing the output flow rate of the pump.
functQforPH_Arab()	Function using Hadj Arab model for output flow rate modeling.
functQforPH_Hamidat()	Function using Hamidat model for output flow rate modeling.
functQforPH_Kou()	Function using Kou model for output flow rate modeling.
functQforPH_theoretical()	Function using theoretical approach for output flow rate modeling.
functQforVH()	Function redirecting to functQforPH.
iv_curve_data(head[, nbpoint])	Function returning the data needed for plotting the IV curve at a given head.
`starting_characteristics`(tdh, ...)	To Develop: In order to start, the pump usually need a higher power input than the minimum power input in steady state operation.

Attributes

modeling_method

4.1.4 pvpumpingsystem.pipenetwork.PipeNetwork

class pvpumpingsystem.pipenetwork.**PipeNetwork**(*h_stat*, *l_tot*, *diam*, *roughness=0*, *material=None*, *fittings=None*, *optimism=None*)

Class representing a simple hydraulic network.

h_stat
static head [m]

Type float,

l_tot
total length of pipes (not necessarily horizontal) [m]

Type float,

diam
fixed pipe diameter for all the network (propose to correct with fluids.piping.nearest_pipe()?) [m]

Type float,

roughness
roughness of pipes [m]

Type float, default is 0

material
If given and roughness == 0, the roughness will be changed to the one of the material if the material is found in a database of roughnesses.

Type str, default is None

fittings
dictionnary of fittings, with angles as keys and number as values (check in fluids module how to define it)

Type dict, NOT IMPLEMENTED YET. default is None

optimism
For values of roughness coming from material, a minimum, maximum, and average value is normally given; if True, returns the minimum roughness; if False, the maximum roughness; if None, the average roughness.

Type boolean, default is None

__init__(*h_stat*, *l_tot*, *diam*, *roughness=0*, *material=None*, *fittings=None*, *optimism=None*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(h_stat, l_tot, diam[, roughness, ...])	Initialize self.
dynamichead(Qlpm[, T, verbose])	Calculates the dynamic head of the pipe network according to the flow given Q, and using the Darcy-Weisbach equation.

4.1.5 pvpumpingsystem.reservoir.Reservoir

class pvpumpingsystem.reservoir.**Reservoir**(*size=0*, *water_volume=0*, *price=0*, *material=None*)
Class defining a water tank with its main characteristics.

size
Volume of reservoir [L]. '0' means no reservoir is used

Type float, default is 0

water_volume
Volume of water in the reservoir [L]. 0 = empty

Type float, default is 0

material
Material of the reservoir

Type str, default is None

price
Price of the reservoir and of the pipes [USD] (to be separated ultimately)

Type float, default is 0

__init__(*size=0*, *water_volume=0*, *price=0*, *material=None*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__([size, water_volume, price, material])	Initialize self.
change_water_volume(quantity[, verbose])	Function for adding or removing water in the reservoir.

4.1.6 pvpumpingsystem.consumption.Consumption

class pvpumpingsystem.consumption.**Consumption**(*flow_rate=None*, *constant_flow=None*, *repeated_flow=None*, *length=8760*, *year=2005*, *safety_factor=1*)

The Consumption class defines a consumption schedule, typically through a year.

Parameters

- **flow_rate** (*pd.DataFrame*) – The consumption schedule in itself [L/min]
- **constant_flow** (*numeric*) – Parameter allowing to build consumption data with constant consumption through the flow_rates DataFrame.
- **repeated_flow** (*1D array-like*) – Parameter allowing to build consumption data with a repeated consumption through the time.

__init__(*flow_rate=None*, *constant_flow=None*, *repeated_flow=None*, *length=8760*, *year=2005*, *safety_factor=1*)

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__([flow_rate, constant_flow, ...])	Initialize self.

4.1.7 pvpumpingsystem.pvpumpsystem.PVPumpSystem

class pvpumpingsystem.pvpumpsystem.**PVPumpSystem**(*pvgeneration*, *motorpump*, *coupling='mppt'*, *motorpump_model=None*, *mppt=None*, *pipes=None*, *reservoir=None*, *consumption=None*, *idname=None*)

Class defining a PV pumping system made of:

pvgeneration
Note that the weather file used here should ideally not smooth the extreme conditions (avoid TMY or IWEC for example). The pvgeneration.modelchain.dc_model must be a Single Diode model if the system is directly-coupled

Type pvpumpingsystem.PVGeneration,

motorpump
The pump used in the system.

Type pvpumpingsystem.Pump

coupling
represents the type of coupling between pv generator and pump. Can be 'mppt' or 'direct'

Type str,

motorpump_model
The modeling method used to model the motorpump. Can be: 'kou', 'arab', 'hamidat' or 'theoretical'. Overwrite the motorpump.modeling_method attribute if not None.

Type str, default None

mppt
Maximum power point tracker of the system.

Type pvpumpingsystem.MPPT

pipes

Type pvpumpingsystem.PipeNetwork

reservoir

Type pvpumpingsystem.Reservoir

consumption

Type pvpumpingsystem.Consumption

llp
Loss of Load Probability, i.e. Water shortage probability. It is None until computed by run_model(), and then it ranges between 0 and 1.

Type None or float,

initial_investment
Cost of the system at the installation [USD]. It is None until computed by run_model()

Type None or float,

__init__(*pvgeneration*, *motorpump*, *coupling='mppt'*, *motorpump_model=None*, *mppt=None*, *pipes=None*, *reservoir=None*, *consumption=None*, *idname=None*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(pvgeneration, motorpump[, ...])	Initialize self.
calc_efficiency()	Computes the efficiencies between PV array output and motorpump output, between irradiance and PV output, and global efficiency.
calc_flow([friction, atol, stop])	Computes the flow at the output of the PVPS, and assigns the value to the attribute 'flow'.
calc_reservoir([starting_soc])	Computes the water volume in the reservoir and extra or lacking water compared to the consumption at any time step.
define_motorpump_model(model)	
operating_point([plot, nb_pts, stop])	Finds the IV operating point(s) of the PV array and the pump (load).
run_model([friction, starting_soc])	Comprehensive modeling of the PVPS.

4.2 Functions and methods

4.2.1 Pump modeling

The core of the software's originality lies in the implementation of different motor-pump models and in their coupling with the PV generator.

pump.Pump.iv_curve_data(head[, nbpoint])	Function returning the data needed for plotting the IV curve at a given head.

Continued on next page

Table 11 – continued from previous page

pump.Pump.functIforVH()	Function computing the IV characteristics of the pump depending on head H.
pump.Pump.functIforVH_Arab()	Function using Hadj Arab model for modeling I vs V of pump.
pump.Pump.functIforVH_Kou()	Function using Kou model for modeling I vs V of pump.
pump.Pump.functIforVH_theoretical()	Function using electrical architecture for modeling V vs I of pump.
pump.Pump.functQforVH()	Function redirecting to functQforPH.
pump.Pump.functQforPH()	Function computing the output flow rate of the pump.
pump.Pump.functQforPH_Hamidat()	Function using Hamidat model for output flow rate modeling.
pump.Pump.functQforPH_Arab()	Function using Hadj Arab model for output flow rate modeling.
pump.Pump.functQforPH_Kou()	Function using Kou model for output flow rate modeling.
pump.Pump.functQforPH_theoretical()	Function using theoretical approach for output flow rate modeling.
pump.get_data_pump(path)	Loads the pump data from the .txt file designated by the path.
pump.specs_completeness(specs, ...)	Evaluates the data completeness of a motor-pump.
pump._curves_coeffs_Arab06(specs, ...)	Compute curve-fitting coefficient with method of Hadj Arab [1] and Djoudi Gherbi [2].
pump._curves_coeffs_Kou98(specs, ...)	Compute curve-fitting coefficient with method of Kou [1].
pump._curves_coeffs_Hamidat08(specs, ...)	Compute curve-fitting coefficient with method of Hamidat [1].
pump._curves_coeffs_theoretical(specs, ...)	Compute curve-fitting coefficient following theoretical analysis of motor architecture.
pump._curves_coeffs_theoretical_variable_cst_eff(specs, ...)	Compute curve-fitting coefficient following theoretical analysis of motor architecture.
pump._curves_coeffs_theoretical_constant_cst_eff(specs, ...)	Compute curve-fitting coefficient following theoretical analysis of motor architecture.
pump._curves_coeffs_theoretical_basic(specs, ...)	Compute curve-fitting coefficient following theoretical analysis of motor architecture.
pump._domain_V_H(specs, data_completeness)	Function giving the range of voltage and head in which the pump will work.
pump._domain_P_H(specs, data_completeness)	Function giving the range of power and head in which the pump will work.
pump._extrapolate_pow_eff_with_cst_efficiency(specs)	Adapt/complete specifications of a limite pump datasheet.
pump.plot_Q_vs_P_H_3d(pump)	Print the graph of Q [L/min] vs tdh [m] and P [W] in 3 dimensions.
pump.plot_I_vs_V_H_3d(pump)	Print the graph of I [A] vs tdh [m] and V [V] in 3 dimensions.
pump.plot_Q_vs_V_H_2d(pump)	Print the graph of Q [L/min] vs tdh [m] for each voltage available.

pvpumpingsystem.pump.Pump.iv_curve_data

Pump.**iv_curve_data**(*head*, *nbpoint=40*)

Function returning the data needed for plotting the IV curve at a given head.

Parameters

- **head** (*float*) – Total dynamic head at pump output [m]
- **nbpoint** (*integer, default 40*) – Number of data point wanted

Returns

with following couples keys-values: I: list of current [A] V: list of voltage [V]

Return type dict

pvpumpingsystem.pump.Pump.functIforVH

Pump.**functIforVH**()
Function computing the IV characteristics of the pump depending on head H.

Returns

- **Function giving I according to voltage V and head H for the pump:** I = f1(V, H)
- **Domains of validity for V and H. Can be functions, so as the** range of one depends on the other, or fixed ranges.

Return type tuple

pvpumpingsystem.pump.Pump.functIforVH_Arab

Pump.**functIforVH_Arab**()
Function using Hadj Arab model for modeling I vs V of pump.

Check out *_curves_coeffs_Arab06()* for more details.

pvpumpingsystem.pump.Pump.functIforVH_Kou

Pump.**functIforVH_Kou**()
Function using Kou model for modeling I vs V of pump.

Check out *_curves_coeffs_Kou98()* for more details.

pvpumpingsystem.pump.Pump.functIforVH_theoretical

Pump.**functIforVH_theoretical**()
Function using electrical architecture for modeling V vs I of pump.

Check out *_curves_coeffs_theoretical()* for more details.

pvpumpingsystem.pump.Pump.functQforVH

Pump.**functQforVH**()
Function redirecting to functQforPH. It first computes P with functIforVH(), and then reinjects it into functQforPH().

pvpumpingsystem.pump.Pump.functQforPH

Pump.**functQforPH**()
Function computing the output flow rate of the pump.

Returns

- **the function giving Q according to power P and head H** for the pump: Q = f2(P, H)
- **the domains of validity for P and H. Can be functions, so as the** range of one depends on the other, or fixed ranges.

Return type tuple

pvpumpingsystem.pump.Pump.functQforPH_Hamidat

Pump.**functQforPH_Hamidat**()
Function using Hamidat model for output flow rate modeling.

Check out *_curves_coeffs_Hamidat08()* for more details.

pvpumpingsystem.pump.Pump.functQforPH_Arab

Pump.**functQforPH_Arab**()
Function using Hadj Arab model for output flow rate modeling.

Check out *_curves_coeffs_Arab06()* for more details.

pvpumpingsystem.pump.Pump.functQforPH_Kou

Pump.**functQforPH_Kou**()
Function using Kou model for output flow rate modeling.

Check out *_curves_coeffs_Kou98()* for more details.

pvpumpingsystem.pump.Pump.functQforPH_theoretical

Pump.**functQforPH_theoretical**()
Function using theoretical approach for output flow rate modeling.

Check out *_curves_coeffs_theoretical()* for more details.

pvpumpingsystem.pump.get_data_pump

pvpumpingsystem.pump.**get_data_pump**(*path*)
Loads the pump data from the .txt file designated by the path. This .txt files contains the specifications of the datasheets, and must follow the style of the template: (~/pvpumpingsystem/data/pump_files/0_template_for_pump_specs.txt)

Parameters **path** (*str*) – path to the file of the pump data

Returns A pandas.DataFrame containing the specifications (voltage, flow, current, tdh, power) and a dict with the metadata of the pump.

Return type tuple

pvpumpingsystem.pump.specs_completeness

pvpumpingsystem.pump.**specs_completeness**(*specs*, *motor_electrical_architecture*)

Evaluates the data completeness of a motor-pump.

Parameters

- **specs** (*pandas.DataFrame*) – Dataframe with specifications of motor-pump
- **motor_electrical_architecture** (*str*) – Can be 'permanent_magnet', 'series_excited', 'shunt_excited', 'separately_excited'.

Returns

- **voltage_number: float** number of voltage for which data are given
- **data_number: float** number of points for which lpm, current, voltage and head are given
- **head_number: float** number of head for which other data are given
- **lpm_min: float** Ratio between min flow_rate given and maximum. Should be ideally 0.
- **head_min:float** Ratio between min head given and maximum. Should be ideally 0.
- **elec_archi: boolean** A valid electrical architecture for the motor is given

Return type dict

pvpumpingsystem.pump._curves_coeffs_Arab06

pvpumpingsystem.pump.**_curves_coeffs_Arab06**(*specs*, *data_completeness*)

Compute curve-fitting coefficient with method of Hadj Arab [1] and Djoudi Gherbi [2].

It uses a 3rd order polynomial to model Q(P) and a 1st order polynomial to model I(V). Each corresponding coefficient depends on TDH through a 3rd order polynomial.

Parameters **specs** (*pd.DataFrame*) – DataFrame with specs.

Returns Coefficients resulting from linear regression under keys 'coeffs_f1' and 'coeffs_f2', and statistical figures on goodness of fit (keys: 'rmse_f1', 'nrmse_f1', 'r_squared_f1', 'adjusted_r_squared_f1', 'rmse_f2', 'nrmse_f2', 'r_squared_f2', 'adjusted_r_squared_f2')

Return type dict

References

[1] Hadj Arab A., Benghanem M. & Chenlo F., "Motor-pump system modelization", 2006, Renewable Energy

[2] Djoudi Gherbi, Hadj Arab A., Salhi H., "Improvement and validation of PV motor-pump model for PV pumping system performance analysis", 2017, Solar Energy

pvpumpingsystem.pump._curves_coeffs_Kou98

pvpumpingsystem.pump.**_curves_coeffs_Kou98**(*specs*, *data_completeness*)

Compute curve-fitting coefficient with method of Kou [1].

It uses a 3rd order multivariate polynomial with cross terms to model V(I, TDH) and Q(V, TDH) from the data.

Parameters **specs** (*pd.DataFrame*) – DataFrame with specs.

Returns Coefficients resulting from linear regression under keys 'coeffs_f1' and 'coeffs_f2', and statistical figures on goodness of fit (keys: 'rmse_f1', 'nrmse_f1', 'r_squared_f1', 'adjusted_r_squared_f1', 'rmse_f2', 'nrmse_f2', 'r_squared_f2', 'adjusted_r_squared_f2')

Return type dict

References

[1] Kou Q, Klein S.A. & Beckman W.A., "A method for estimating the long-term performance of direct-coupled PV pumping systems", 1998, Solar Energy

pvpumpingsystem.pump._curves_coeffs_Hamidat08

pvpumpingsystem.pump.**_curves_coeffs_Hamidat08**(*specs*, *data_completeness*)

Compute curve-fitting coefficient with method of Hamidat [1]. It uses a 3rd order polynomial to model P(Q) = a + b*Q + c*Q^2 + d*Q^3 and each corresponding coefficient depends on TDH through a 3rd order polynomial as well. This function needs to be reversed numerically to be used as Q(P).

Parameters **specs** (*pd.DataFrame*) – DataFrame with specs.

Returns Coefficients resulting from linear regression under key 'coeffs_f2', and statistical figures on goodness of fit (keys: 'rmse_f2', 'nrmse_f2', 'r_squared_f2', 'adjusted_r_squared_f2')

Return type dict

References

[1] Hamidat A., Benyoucef B., Mathematic models of photovoltaic motor-pump systems, 2008, Renewable Energy

pvpumpingsystem.pump._curves_coeffs_theoretical

pvpumpingsystem.pump.**_curves_coeffs_theoretical**(*specs*, *data_completeness*, *elec_archi*, *force_model='flexible'*)

Compute curve-fitting coefficient following theoretical analysis of motor architecture.

This kind of approach is used in [1], [2].

Nevertheless, following function takes some liberties with the model of function f2 described in the mentionned papers, in order not to rely on K_p and K_t that are assumed to be unavailable in pump datasheet.

It uses a equation of the form V = R_a*i + beta(H)*np.sqrt(i) to model V(I, TDH) and an equation of the form Q = (a + b*H) * (c + d*P) to model Q(P, TDH) from the data.

Parameters **specs** (*pd.DataFrame*) – DataFrame with specs.

Returns Coefficients resulting from linear regression under keys 'coeffs_f1' and 'coeffs_f2', and statistical figures on goodness of fit (keys: 'rmse_f1', 'nrmse_f1', 'r_squared_f1', 'adjusted_r_squared_f1', 'rmse_f2', 'nrmse_f2', 'r_squared_f2', 'adjusted_r_squared_f2')

Return type dict

References

[1] Mokkedem & al, 2011, 'Performance of a directly-coupled PV water pumping system', Energy Conversion and Management

[2] Khatib & Elmenreich, 2016, 'Modeling of Photovoltaic Systems Using MATLAB', Wiley

[3] Martiré & al, 2008, "A simplified but accurate prevision method for along the sun PV pumping systems"

pvpumpingsystem.pump._curves_coeffs_theoretical_variable_efficiency

pvpumpingsystem.pump.**_curves_coeffs_theoretical_variable_efficiency**(*specs*, *data_completeness*, *elec_archi*)

Compute curve-fitting coefficient following theoretical analysis of motor architecture.

This kind of approach is used in [1], [2].

Nevertheless, following function takes some liberties with the model of function f2 described in the mentionned papers, in order not to rely on K_p and K_t that are assumed to be unavailable in pump datasheet.

It uses a equation of the form V = R_a*i + beta(H)*np.sqrt(i) to model V(I, TDH) and an equation of the form Q = (a + b*H) * (c + d*P) to model Q(P, TDH) from the data.

Parameters **specs** (*pd.DataFrame*) – DataFrame with specs.

Returns Coefficients resulting from linear regression under keys 'coeffs_f1' and 'coeffs_f2', and statistical figures on goodness of fit (keys: 'rmse_f1', 'nrmse_f1', 'r_squared_f1', 'adjusted_r_squared_f1', 'rmse_f2', 'nrmse_f2', 'r_squared_f2', 'adjusted_r_squared_f2')

Return type dict

References

[1] Mokkedem & al, 2011, 'Performance of a directly-coupled PV water pumping system', Energy Conversion and Management

[2] Khatib & Elmenreich, 2016, 'Modeling of Photovoltaic Systems Using MATLAB', Wiley

pvpumpingsystem.pump._curves_coeffs_theoretical_constant_efficiency

pvpumpingsystem.pump.**_curves_coeffs_theoretical_constant_efficiency**(*specs*, *data_completeness*, *elec_archi*, *force_model='flexible'*)

Compute curve-fitting coefficient following theoretical analysis of motor architecture.

This kind of approach is used in [1], [2].

Nevertheless, following function takes some liberties with the model of function f2 described in the mentionned papers, in order not to rely on K_p and K_t that are here assumed to be unavailable in pump datasheet.

It uses a equation of the form V = R_a*i + beta(H)*np.sqrt(i) to model V(I, TDH) and an equation of the form Q = (a + b*H) * (c + d*P) to model Q(P, TDH) from the data.

Parameters **specs** (*pd.DataFrame*) – DataFrame with specs.

Returns Coefficients resulting from linear regression under keys 'coeffs_f1' and 'coeffs_f2', and statistical figures on goodness of fit (keys: 'rmse_f1', 'nrmse_f1', 'r_squared_f1', 'adjusted_r_squared_f1', 'rmse_f2', 'nrmse_f2', 'r_squared_f2', 'adjusted_r_squared_f2')

Return type dict

References

[1] Mokkedem & al, 2011, 'Performance of a directly-coupled PV water pumping system', Energy Conversion and Management

[2] Khatib & Elmenreich, 2016, 'Modeling of Photovoltaic Systems Using MATLAB', Wiley

[3] Martiré & al, 2008, "A simplified but accurate prevision method for along the sun PV pumping systems"

pvpumpingsystem.pump._curves_coeffs_theoretical_basic

pvpumpingsystem.pump.**_curves_coeffs_theoretical_basic**(*specs*, *data_completeness*, *elec_archi*)

Compute curve-fitting coefficient following theoretical analysis of motor architecture.

Very basic model only to use with MPPT and assuming a constant efficiency.

It uses an equation of the form Q = gamma*P/TDH to model Q(P, TDH) from the data.

Parameters **specs** (*pd.DataFrame,*) – DataFrame with specs.

Returns Coefficients resulting from linear regression under key 'coeffs_f2', and statistical figures on goodness of fit (keys: 'rmse_f2', 'nrmse_f2', 'r_squared_f2', 'adjusted_r_squared_f2')

Return type dict

References

[1] Mokkedem & al, 2011, 'Performance of a directly-coupled PV water pumping system', Energy Conversion and Management

[2] Khatib & Elmenreich, 2016, 'Modeling of Photovoltaic Systems Using MATLAB', Wiley

pvpumpingsystem.pump._domain_V_H

pvpumpingsystem.pump.**_domain_V_H**(*specs*, *data_completeness*)

Function giving the range of voltage and head in which the pump will work.

Parameters

- **specs** (*pandas.DataFrame,*) – Specifications typically coming from Pump.specs
- **data_completeness** (*dict,*) – Typically comes from specs_completeness() function.

Returns Two lists, the domains on voltage V [V] and on head [m]

Return type tuple

pvpumpingsystem.pump._domain_P_H

pvpumpingsystem.pump.**_domain_P_H**(*specs*, *data_completeness*)

Function giving the range of power and head in which the pump will work.

Parameters

- **specs** (*pandas.DataFrame,*) – Specifications typically coming from Pump.specs
- **data_completeness** (*dict,*) – Typically comes from specs_completeness() function.

Returns Two lists, the domains on power P [W] and on head [m]

Return type tuple

pvpumpingsystem.pump._extrapolate_pow_eff_with_cst_efficiency

pvpumpingsystem.pump.**_extrapolate_pow_eff_with_cst_efficiency**(*specs*, *efficiency_coeff=1*)

Adapt/complete specifications of a limite pump datasheet. Used in '__init__()'

Works on the assumption that the available (I, V, Q, TDH) point is the rated operating point, and that the efficiency is constant then (oversimplification!). In order to mitigate this last assumption, a coeff can be used to consider the mean efficiency as a ratio of the rated efficiency.

Parameters

- **specs** (*pandas.DataFrame*) – Attribute specs of Pump().
- **efficiency_coeff** (*float, in range [0, 1]*) – The ratio between the mean efficiency and the rated efficiency -> mean_efficiency = efficiency_coeff * rated_efficiency

Returns Attribute specs of Pump().

Return type pandas.DataFrame

pvpumpingsystem.pump.plot_Q_vs_P_H_3d

pvpumpingsystem.pump.**plot_Q_vs_P_H_3d**(*pump*)

Print the graph of Q [L/min] vs tdh [m] and P [W] in 3 dimensions.

Returns **Graph Q (H, P)**

Return type matplotlib.figure

pvpumpingsystem.pump.plot_I_vs_V_H_3d

pvpumpingsystem.pump.**plot_I_vs_V_H_3d**(*pump*)

Print the graph of I [A] vs tdh [m] and V [V] in 3 dimensions.

Returns **Graph I (V, H)**

Return type matplotlib.figure

pvpumpingsystem.pump.plot_Q_vs_V_H_2d

pvpumpingsystem.pump.**plot_Q_vs_V_H_2d**(*pump*)
Print the graph of Q [L/min] vs tdh [m] for each voltage available.

Returns **Graph Q (H, V)**

Return type matplotlib.figure

4.2.2 Other components modeling

reservoir.Reservoir.change_water_volume(quantity)	Function for adding or removing water in the reservoir.
consumption.adapt_to_flow_pumped(...)	Method for shrinking the consumption flow_rate attribute at the same size than the corresponding pumped flow rate data.
pipenetwork.PipeNetwork.dynamichead(Qlpm[, ...])	Calculates the dynamic head of the pipe network according to the flow given Q, and using the Darcy-Weisbach equation.
pvgeneration.PVGeneration.run_model()	Runs the modelchain of the PV generation.

pvpumpingsystem.reservoir.Reservoir.change_water_volume

Reservoir.**change_water_volume**(*quantity*, *verbose=False*)
Function for adding or removing water in the reservoir.

Parameters **quantity** (*float*) – amount of water too add or remove (in liters)

Returns (water_volume, extra (+) or lacking water(-))

Return type tuple

pvpumpingsystem.consumption.adapt_to_flow_pumped

pvpumpingsystem.consumption.**adapt_to_flow_pumped**(*Q_consumption*, *Q_pumped*)
Method for shrinking the consumption flow_rate attribute at the same size than the corresponding pumped flow rate data.

Parameters

- **Q_consumption** (*pd.DataFrame,*) – Dataframe with pandas timestamp as index. Typically comes from PVPumpSystem.consumption.flow_rate
- **Q_pumped** (*pd.DataFrame,*) – Dataframe with pandas timestamp as index. Typically comes from PVPumpSystem.flow.Qlpm

Returns Consumption data modified.

Return type pandas.DataFrame

pvpumpingsystem.pipenetwork.PipeNetwork.dynamichead

PipeNetwork.**dynamichead**(*Qlpm*, *T=20*, *verbose=False*)
Calculates the dynamic head of the pipe network according to the flow given Q, and using the Darcy-Weisbach equation.

Parameters

- **Q** (*float,*) – water flow in liter per minute [lpm]
- **T** (*float,*) – water temperature [°C]
- **verbose** (*boolean,*) – allows printing of Re numbers of the computing

Returns dynamic head [m]

Return type float

pvpumpingsystem.pvgeneration.PVGeneration.run_model

PVGeneration.**run_model**()

Runs the modelchain of the PV generation.

See pvlib.modelchain.run_model() for more details.

4.2.3 Global modeling

pvpumpsystem.PVPumpSystem.define_motorpump_model(model)	
pvpumpsystem.PVPumpSystem.operating_point([...])	Finds the IV operating point(s) of the PV array and the pump (load).
pvpumpsystem.PVPumpSystem.calc_flow([...])	Computes the flow at the output of the PVPS, and assigns the value to the attribute 'flow'.
pvpumpsystem.PVPumpSystem.calc_efficiency()	Computes the efficiencies between PV array output and motorpump output, between irradiance and PV output, and global efficiency.
pvpumpsystem.PVPumpSystem.calc_reservoir([...])	Computes the water volume in the reservoir and extra or lacking water compared to the consumption at any time step.
pvpumpsystem.PVPumpSystem.run_model([...])	Comprehensive modeling of the PVPS.
pvpumpsystem.function_i_from_v(V, I_L, I_o, ...)	Deprecated: 'function_i_from_v' deprecated.
pvpumpsystem.operating_point(params, ...[, ...])	Finds the IV operating point(s) between PV array and load (motor-pump).
pvpumpsystem.calc_flow_directly_coupled(...)	Computes input electrical characteristics, total dynamic head, and flow at pump output.
pvpumpsystem.calc_flow_mppt_coupled(...[, ...])	Computes input electrical characteristics, total dynamic head, and flow at pump output.
pvpumpsystem.calc_efficiency(df, irradiance, ...)	Computes the efficiencies between PV array output and motorpump output, between irradiance and PV output, and global efficiency.

pvpumpingsystem.pvpumpsystem.PVPumpSystem.define_motorpump_model

PVPumpSystem.**define_motorpump_model**(*model*)

pvpumpingsystem.pvpumpsystem.PVPumpSystem.operating_point

PVPumpSystem.**operating_point**(*plot=False*, *nb_pts=50*, *stop=8760*)

Finds the IV operating point(s) of the PV array and the pump (load).

cf pvpumpsystem.operating_point for more details

Parameters

- **plot** (*Boolean*) – Allows or not the printing of IV curves of PV system and of the load.
- **nb_pts** (*numeric*) – number of points on graph
- **stop** (*numeric*) – number of data on which the computation is run

Returns

- *pandas.DataFrame* – Current ('I') and voltage ('V') at the operating point between load and pv array.
- *Note / Issues*
- ————-
- *Takes ~10sec to compute 8760 iterations*

pvpumpingsystem.pvpumpsystem.PVPumpSystem.calc_flow

PVPumpSystem.**calc_flow**(*friction=False*, *atol=0.1*, *stop=8760*, ***kwargs*)

Computes the flow at the output of the PVPS, and assigns the value to the attribute 'flow'.

cf *calc_flow_directly_coupled()* and *calc_flow_mppt_coupled()* for more details.

Parameters

- **friction** (*boolean, default is False*) – Decide if the system considers the friction head due to the flow rate of water pump or not. Often can be put to False if the pipes are well sized, because negligible in relation to the static head. When turned to True, it approximately multiplies by 3 the computation time (if atol kept to default value).
- **atol** (*numeric*) – absolute tolerance on the uncertainty of the flow in L/min
- **stop** (*numeric*) – number of data on which the computation is run

Returns

Return type None

Notes

Takes ~20 sec for computing 8760 iterations with mppt coupling and atol=0.1 lpm

Takes ~60 sec for computing 8760 iterations with direct coupling and atol=0.1 lpm

pvpumpingsystem.pvpumpsystem.PVPumpSystem.calc_efficiency

PVPumpSystem.**calc_efficiency**()

Computes the efficiencies between PV array output and motorpump output, between irradiance and PV output, and global efficiency. Assigns the resulting data to the attribute 'efficiency'.

cf *calc_efficiency()* for more details

Returns

Return type None

pvpumpingsystem.pvpumpsystem.PVPumpSystem.calc_reservoir

PVPumpSystem.**calc_reservoir**(*starting_soc='morning'*)

Computes the water volume in the reservoir and extra or lacking water compared to the consumption at any time step. Assigns the resulting data to the attribute 'flow'.

cf *calc_reservoir()* for more details

Parameters **starting_soc** (*str or float, default is 'morning'*) – State of Charge of the reservoir at the beginning of the simulation [%]. Available strings are 'empty' (no water in reservoir), 'morning' (enough water for one morning consumption) and 'full'.

Returns

Return type None

pvpumpingsystem.pvpumpsystem.PVPumpSystem.run_model

PVPumpSystem.**run_model**(*friction=False*, *starting_soc='morning'*, ***kwargs*)

Comprehensive modeling of the PVPS. Computes Loss of Power Supply (LLP) and stores it as an attribute. Re-run eveything even if already computed before.

Parameters

- **friction** (*boolean, default is False*) – Decide if the friction head is taken into account in the computation. Turning it to True multiply by three the calculation time.
- ****kwargs** – Keyword arguments that apply to the financial analysis. kwargs are transfered to fin.net_present_value() function.

pvpumpingsystem.pvpumpsystem.function_i_from_v

pvpumpingsystem.pvpumpsystem.**function_i_from_v**(*V*, *I_L*, *I_o*, *R_s*, *R_sh*, *nNsVth*, *M_s=1*, *M_p=1*)

Deprecated: 'function_i_from_v' deprecated. Use pvlib.pvsystem.i_from_v instead

Function I=f(V) coming from equation of Single Diode Model with parameters adapted to the irradiance and temperature.

The adaptation of the 5 parameters from module parameters to array parameters is made according to [1].

Parameters

- **V** (*numeric*) – Voltage at which the corresponding current is to be calculated in volt.
- **I_L** (*numeric*) – The light-generated current (or photocurrent) in amperes.
- **I_o** (*numeric*) – The dark or diode reverse saturation current in amperes.
- **nNsVth** (*numeric*) – The product of the usual diode ideality factor (n, unitless), number of cells in series (Ns), and cell thermal voltage at reference conditions, in units of V.
- **R_sh** (*numeric*) – The shunt resistance in ohms.
- **R_s** (*numeric*) – The series resistance in ohms.

- **M_s** (*numeric*) – The number of module in series in the whole pv system. (= modules_per_strings)
- **M_p** (*numeric*) – The number of module in parallel in the whole pv system. (= strings_per_inverter)

Returns Output current of the whole pv source, in A.

Return type numeric

Notes

According to the speed of the computations, it seems that the complexity of this function is cubic O(n^3), and therefore it takes too much time to compute this way for long vectors (around 45min for 8760 elements).

Different from pvsystem.i_from_v because it includes M_s and M_p, so it gives the corresponding current at the output of the array, not only the module.

References

[1] Petrone & al (2017), "Photovoltaic Sources Modeling", Wiley, p.5. URL: http://doi.wiley.com/10.1002/9781118755877

pvpumpingsystem.pvpumpsystem.operating_point

pvpumpingsystem.pvpumpsystem.**operating_point**(*params, modules_per_string, strings_per_inverter, load_fctIfromVH=None, load_interval_V=[-inf, inf], pv_interval_V=[-inf, inf], tdh=0*)

Finds the IV operating point(s) between PV array and load (motor-pump).

Parameters

- **params** (*pandas.Dataframe*) – Dataframe containing the 5 diode parameters. Typically comes from PVGeneration.ModelChain.diode_params
- **modules_per_string** (*numeric*) – Number of modules in series in a string
- **strings_per_inverter** (*numeric*) – Number of strings in parallel
- **load_fctIfromVH** (*function*) – The function I=f(V, H) of the load directly coupled with the array.
- **tdh** (*numeric*) – Total dynamic head

Returns Current ('I') and voltage ('V') at the operating point between load and pv array. I and V are float. It is 0 when there is no irradiance, and np.nan when pv array and load don't match.

Return type pandas.DataFrame

Notes

Takes ~10sec for computing 8760 iterations

pvpumpingsystem.pvpumpsystem.calc_flow_directly_coupled

pvpumpingsystem.pvpumpsystem.**calc_flow_directly_coupled**(*pvgeneration*, *motorpump*, *pipes*, *friction=False*, *atol=0.1*, *stop=8760*, ***kwargs*)

Computes input electrical characteristics, total dynamic head, and flow at pump output.

Parameters

- **pvgeneration** (*pvpumpingsystem.pvgeneration.PVGeneration object*) – The PV generator object
- **motorpump** (*pump.Pump object*) – Pump associated with the PV generator
- **pipes** (*pipenetwork.PipeNetwork object*) – Hydraulic network linked to the pump
- **friction** (*boolean, default is False*) – Decide if the system takes into account the friction head due to the flow rate of water pump (friction = True) or if the system just considers the static head of the system (friction = False). Often can be put to False if the pipes are well sized.
- **atol** (*numeric*) – absolute tolerance on the uncertainty of the flow in l/min. Used if friction = True.
- **stop** (*numeric*) – number of data on which the computation is run

Returns

df –

pd.Dataframe with following attributes: 'I': Current in A at operating point 'V': Voltage in V at operating point 'Qlpm': Flow rate of water in L/minute 'P': Input power to the pump in W 'P_unused': Power unused (because too low or too high) 'tdh': Total dynamic head in m

Return type pandas.DataFrame,

Notes

Takes ~15 sec for computing 8760 iterations with atol=0.1lpm

pvpumpingsystem.pvpumpsystem.calc_flow_mppt_coupled

pvpumpingsystem.pvpumpsystem.**calc_flow_mppt_coupled**(*pvgeneration*, *motorpump*, *pipes*, *mppt*, *friction=False*, *atol=0.1*, *stop=8760*, ***kwargs*)

Computes input electrical characteristics, total dynamic head, and flow at pump output.

Parameters

- **pvgeneration** (pvpumpingsystem.pvgeneration.PVGeneration) – The PV generator.
- **motorpump** (*pump.Pump object*) – Pump associated with the PV generator
- **pipes** (*pipenetwork.PipeNetwork object*) – Hydraulic network linked to the pump
- **mppt** (*mppt.MPPT object,*) – The maximum power point tracker of the system.

- **friction** (*boolean, default is False*) – Decide if the system takes into account the friction head due to the flow rate of water pump (friction = True) or if the system just considers the static head of the system (friction = False). Often can be put to False if the pipes are well sized.
- **atol** (*numeric*) – absolute tolerance on the uncertainty of the flow in l/min. Used if friction=True.
- **stop** (*numeric*) – number of data on which the computation is run

Returns

df –

pd.Dataframe with following attributes: 'Qlpm': Flow rate of water in L/minute 'P': Input power to the pump in W 'P_unused': Power unused (because too low or too high) 'tdh': Total dynamic head in m

Return type pandas.DataFrame

Notes

Takes ~15 sec for computing 8760 iterations with atol=0.1lpm

pvpumpingsystem.pvpumpsystem.calc_efficiency

pvpumpingsystem.pvpumpsystem.**calc_efficiency**(*df*, *irradiance*, *pv_area*)

Computes the efficiencies between PV array output and motorpump output, between irradiance and PV output, and global efficiency.

Parameters

- **df** (*pd.DataFrame*) –

  Dataframe containing at least: electric power 'P' flow-rate 'Qlpm' total dynamic head 'tdh'

- **irradiance** (*pd.DataFrame*) – Dataframe containing irradiance on PV
- **pv_area** (*numeric*) – Suface of PV collectors

Returns Dataframe with efficiencies

Return type pandas.DataFrame

4.2.4 Sizing tools

sizing.shrink_weather_representative(...[, ...])	Create a new weather_data object representing the range of weather that can be found in the weather_data given.
sizing.shrink_weather_worst_month(weather_data)	Create a new weather_data object with only the worst month of the weather_data given, according to the global horizontal irradiance (ghi) data.
sizing.subset_respecting_llp_direct(...[, ...])	Function returning the configurations of PV modules and pump that will minimize the net present value of the system and will insure the Loss of Load Probability (llp) is inferior to the one given.

Continued on next page

Table 14 – continued from previous page

sizing.size_nb_pv_direct(pvps_fixture, ...)	Function sizing the PV generator (i.e.
sizing.subset_respecting_llp_mppt(... [, ...])	Function returning the configurations of PV modules and pump that will minimize the net present value of the system and will ensure the Loss of Load Probability (llp) is inferior to the one given.
sizing.size_nb_pv_mppt(pvps_fixture, ...)	Function sizing the PV generator (i.e.
sizing.sizing_minimize_npv(pv_database, ...)	Function returning the configuration of PV modules and pump that minimizes the net present value (NPV) of the system and ensures that the Loss of Load Probability (llp) is inferior to the 'llp_accepted'.

pvpumpingsystem.sizing.shrink_weather_representative

pvpumpingsystem.sizing.**shrink_weather_representative**(*weather_data*, *nb_elt=48*)

Create a new weather_data object representing the range of weather that can be found in the weather_data given. It allows to reduce the number of lines in the weather file from 8760 (if full year and hourly data) to 'nb_elt' lines, and eventually to greatly reduce the computation time.

Parameters

- **weather_data** (*pandas.DataFrame*) – The hourly data on irradiance, temperature, and others meteorological parameters. Typically comes from pvlib.epw.read_epw() or pvlib.tmy.read.tmy().
- **nb_elt** (*integer, default 48*) – Number of line to keep in the weather_data file.

Returns Weather data with (nb_elt) lines

Return type pandas.DataFrame

pvpumpingsystem.sizing.shrink_weather_worst_month

pvpumpingsystem.sizing.**shrink_weather_worst_month**(*weather_data*)

Create a new weather_data object with only the worst month of the weather_data given, according to the global horizontal irradiance (ghi) data.

Parameters **weather_data** (*pandas.DataFrame*) – The hourly data on irradiance, temperature, and others meteorological parameters. Typically comes from pvlib.epw.read_epw() or pvlib.tmy.read.tmy().

Returns Weather data with (nb_elt) lines

Return type pandas.DataFrame

pvpumpingsystem.sizing.subset_respecting_llp_direct

pvpumpingsystem.sizing.**subset_respecting_llp_direct**(*pv_database*, *pump_database*, *weather_data*, *weather_metadata*, *pvps_fixture*, *llp_accepted=0.01*, *M_s_guess=None*, *M_p_guess=None*, ***kwargs*)

Function returning the configurations of PV modules and pump that will minimize the net present value of the system and will insure the Loss of Load Probability (llp) is inferior to the one given.

Parameters

- **pv_database** (*list of strings,*) – List of pv module names to try. If name is not eact, it will search a pv module database to find the best match.
- **pump_database** (*list of pvpumpingssytem.Pump objects*) – List of motor-pump to try.
- **weather_data** (*pd.DataFrame*) – Weather file of the location. Typically comes from pvlib.iotools.epw.read_epw()
- **weather_metadata** (*dict*) – Weather file metadata of the location. Typically comes from pvlib.iotools.epw.read_epw()
- **pvps_fixture** (*pvpumpingsystem.PVPumpSystem object*) – The PV pumping system to size.
- **llp_accepted** (*float, default is 0.01*) – Maximum Loss of Load Probability that can be accepted. Between 0 and 1
- **M_S_guess** (*integer, default is None*) – Estimated number of modules in series in the PV array. Will be sized by the function.

Returns All configurations tested respecting the LLP.

Return type pandas.Dataframe

pvpumpingsystem.sizing.size_nb_pv_direct

pvpumpingsystem.sizing.**size_nb_pv_direct**(*pvps_fixture*, *llp_accepted*, *M_s_min*, *M_s_max*, *M_p_min*, *M_p_max*, *M_s_guess=None*, *M_p_guess=None*, ***kwargs*)

Function sizing the PV generator (i.e. the number of PV modules) for a specified llp_accepted.

Returns Number of modules in series and number of strings in parallel.

Return type tuple

pvpumpingsystem.sizing.subset_respecting_llp_mppt

pvpumpingsystem.sizing.**subset_respecting_llp_mppt**(*pv_database*, *pump_database*, *weather_data*, *weather_metadata*, *pvps_fixture*, *llp_accepted=0.01*, *M_s_guess=None*, ***kwargs*)

Function returning the configurations of PV modules and pump that will minimize the net present value of the system and will ensure the Loss of Load Probability (llp) is inferior to the one given.

Parameters

- **pv_database** (*list of strings,*) – List of pv module names to try. If name is not eact, it will search a pv module database to find the best match.
- **pump_database** (*list of pvpumpingssytem.Pump objects*) – List of motor-pump to try.
- **weather_data** (*pd.DataFrame*) – Weather file of the location. Typically comes from pvlib.iotools.epw.read_epw()
- **weather_metadata** (*dict*) – Weather file metadata of the location. Typically comes from pvlib.iotools.epw.read_epw()
- **pvps_fixture** (*pvpumpingsystem.PVPumpSystem object*) – The PV pumping system to size.

- **llp_accepted** (*float, default is 0.01*) – Maximum Loss of Load Probability that can be accepted. Between 0 and 1
- **M_S_guess** (*integer, default is None*) – Estimated number of modules in series in the PV array. Will be sized by the function.

Returns All configurations tested respecting the LLP.

Return type pandas.Dataframe,

pvpumpingsystem.sizing.size_nb_pv_mppt

pvpumpingsystem.sizing.**size_nb_pv_mppt**(*pvps_fixture*, *llp_accepted*, *M_s_guess*, ***kwargs*)

Function sizing the PV generator (i.e. the number of PV modules) for a specified llp_accepted. Here 'M_s' represents the total number of PV module (because M_p = 1).

Returns Number of PV modules in the array, regardless of how they are arranged.

Return type float

pvpumpingsystem.sizing.sizing_minimize_npv

pvpumpingsystem.sizing.**sizing_minimize_npv**(*pv_database*, *pump_database*, *weather_data*, *weather_metadata*, *pvps_fixture*, *llp_accepted=0.01*, *M_s_guess=None*, *M_p_guess=None*, ***kwargs*)

Function returning the configuration of PV modules and pump that minimizes the net present value (NPV) of the system and ensures that the Loss of Load Probability (llp) is inferior to the 'llp_accepted'.

It proceeds by sizing the number of PV module needed to respect 'llp_accepted' for each combination of pump and pv module. If the combination does not allow to respect 'llp_accepted' in any case, it is discarded. Then the combination with the lowest NPV is returned as the solution (first element of the tuple returned). All combinations details are also returned (second element of the tuple returned).

Parameters

- **pv_database** (*list of strings,*) – List of pv module names to try. If name is not eact, it will search a pv module database to find the best match.
- **pump_database** (*list of pvpumpingssytem.Pump objects*) – List of motor-pump to try.
- **weather_data** (*pd.DataFrame*) – Weather data of the location. Typically comes from pvlib.iotools.epw.read_epw()
- **weather_metadata** (*dict*) – Weather file metadata of the location. Typically comes from pvlib.iotools.epw.read_epw()
- **pvps_fixture** (*pvpumpingsystem.PVPumpSystem object*) – The PV pumping system to size.
- **llp_accepted** (*float, between 0 and 1*) – Maximum Loss of Load Probability that can be accepted.
- **M_S_guess** (*integer,*) – Estimated number of modules in series in the PV array. Will be sized by the function.
- ****kwargs** (*dict,*) – Keyword arguments internally given to py:func:*PVPumpSystem.run_model()*. Made for giving the financial parameters of the project.

Returns First element is a pandas.DataFrame containing the configuration that minimizes the net present value (NPV) of the system. This first element can contain more than one configuration if multiple configurations have the exact same NPV which also turns to be the minimum. Second element is a pandas.DataFrame containing all configurations tested respecting the LLP.

Return type tuple

4.2.5 Ancillary functions

function_models.correlation_stats(funct_mod, ...)	Compute statistical figures to assess quality of curve fitting.
function_models.compound_polynomial_1_2(...)	Model of a compound polynomial function made of a global equation of first order on x, for which each coefficient follows a second order equation on y.
function_models.compound_polynomial_1_3(...)	Model of a compound polynomial function made of a global equation of first order on x, for which each coefficient follows a third order equation on y.
function_models.compound_polynomial_2_2(...)	Model of a compound polynomial function made of a global equation of second order on x, for which each coefficient follows a third order equation on y.
function_models.compound_polynomial_2_3(...)	Model of a compound polynomial function made of a global equation of second order on x, for which each coefficient follows a third order equation on y.
function_models.compound_polynomial_3_3(...)	Model of a compound polynomial function made of a global equation of third order on x, for which each coefficient follows a third order equation on y.
function_models.polynomial_multivar_3_3_4(...)	Model of a multivariate polynomial function of third order on x and y, and with 1 interaction term.
function_models.polynomial_multivar_3_3_1(...)	Model of a multivariate polynomial function of third order on x and y, and with 1 interaction term.
function_models.polynomial_multivar_2_2_1(...)	Model of a multivariate polynomial function of second order on x and y, and with 1 interaction term.
function_models.polynomial_multivar_2_2_0(...)	Model of a multivariate polynomial function of second order on x and y, and with no interaction term.
function_models.polynomial_multivar_1_1_0(...)	Model of a multivariate polynomial function of first order on x and y, and with no interaction term.
function_models.polynomial_multivar_0_1_0(...)	Model of a multivariate polynomial function of first order on y (actually not really multivariate so).
function_models.polynomial_5(x, y_intercept, ...)	Model of a polynomial function of fifth order.
function_models.polynomial_4(x, y_intercept, ...)	Model of a polynomial function of fourth order.
function_models.polynomial_3(x, y_intercept, ...)	Model of a polynomial function of third order.
function_models.polynomial_2(x, y_intercept, ...)	Model of a polynomial function of second order.
function_models.polynomial_1(x, y_intercept, a)	Model of a polynomial function of first order, i.e.
function_models.polynomial_divided_2_1(x, a, ...)	Model of a polynomial function of second order divided by x.
waterproperties.water_prop(name, T)	Function giving water property requested.

Continued on next page

Table 15 – continued from previous page

finance.initial_investment(pvps[, ...])	Function computing the initial investment cost.
finance.net_present_value(pvps[, ...])	Function computing the net present value of a PVPS

pvpumpingsystem.function_models.correlation_stats

pvpumpingsystem.function_models.**correlation_stats**(*funct_mod*, *params*, *data_input*, *data_to_fit*)

Compute statistical figures to assess quality of curve fitting. In particular 'root mean square error', 'normalized root mean square error', 'r_squared', 'adjusted_r_squared' and size of data sample 'nb_data' are computed.

Returns Keys are: -'rmse' -'nrmse' -'r_squared' -'adjusted_r_squared' -'nb_data'

Return type dict

pvpumpingsystem.function_models.compound_polynomial_1_2

pvpumpingsystem.function_models.**compound_polynomial_1_2**(*input_val*, *a1*, *a2*, *a3*, *b1*, *b2*, *b3*)

Model of a compound polynomial function made of a global equation of first order on x, for which each coefficient follows a second order equation on y.

pvpumpingsystem.function_models.compound_polynomial_1_3

pvpumpingsystem.function_models.**compound_polynomial_1_3**(*input_val*, *a1*, *a2*, *a3*, *a4*, *b1*, *b2*, *b3*, *b4*)

Model of a compound polynomial function made of a global equation of first order on x, for which each coefficient follows a third order equation on y.

pvpumpingsystem.function_models.compound_polynomial_2_2

pvpumpingsystem.function_models.**compound_polynomial_2_2**(*input_val*, *a1*, *a2*, *a3*, *b1*, *b2*, *b3*, *c1*, *c2*, *c3*)

Model of a compound polynomial function made of a global equation of second order on x, for which each coefficient follows a third order equation on y.

pvpumpingsystem.function_models.compound_polynomial_2_3

pvpumpingsystem.function_models.**compound_polynomial_2_3**(*input_val*, *a1*, *a2*, *a3*, *a4*, *b1*, *b2*, *b3*, *b4*, *c1*, *c2*, *c3*, *c4*)

Model of a compound polynomial function made of a global equation of second order on x, for which each coefficient follows a third order equation on y.

pvpumpingsystem.function_models.compound_polynomial_3_3

pvpumpingsystem.function_models.**compound_polynomial_3_3**(*input_val*, *a1*, *a2*, *a3*, *a4*, *b1*, *b2*, *b3*, *b4*, *c1*, *c2*, *c3*, *c4*, *d1*, *d2*, *d3*, *d4*)

Model of a compound polynomial function made of a global equation of third order on x, for which each coefficient follows a third order equation on y.

pvpumpingsystem.function_models.polynomial_multivar_3_3_4

pvpumpingsystem.function_models.**polynomial_multivar_3_3_4**(*input_val*, *y_intercept*, *a1*, *a2*, *a3*, *b1*, *b2*, *b3*, *c1*, *c2*, *c3*, *c4*)

Model of a multivariate polynomial function of third order on x and y, and with 1 interaction term.

pvpumpingsystem.function_models.polynomial_multivar_3_3_1

pvpumpingsystem.function_models.**polynomial_multivar_3_3_1**(*input_val*, *y_intercept*, *a1*, *a2*, *a3*, *b1*, *b2*, *b3*, *c1*)

Model of a multivariate polynomial function of third order on x and y, and with 1 interaction term.

pvpumpingsystem.function_models.polynomial_multivar_2_2_1

pvpumpingsystem.function_models.**polynomial_multivar_2_2_1**(*input_val*, *y_intercept*, *a1*, *a2*, *b1*, *b2*, *c1*)

Model of a multivariate polynomial function of second order on x and y, and with 1 interaction term.

pvpumpingsystem.function_models.polynomial_multivar_2_2_0

pvpumpingsystem.function_models.**polynomial_multivar_2_2_0**(*input_val*, *y_intercept*, *a1*, *a2*, *b1*, *b2*)

Model of a multivariate polynomial function of second order on x and y, and with no interaction term.

pvpumpingsystem.function_models.polynomial_multivar_1_1_0

pvpumpingsystem.function_models.**polynomial_multivar_1_1_0**(*input_val*, *y_intercept*, *a1*, *b1*)

Model of a multivariate polynomial function of first order on x and y, and with no interaction term.

pvpumpingsystem.function_models.polynomial_multivar_0_1_0

pvpumpingsystem.function_models.**polynomial_multivar_0_1_0**(*input_val*, *y_intercept*, *b1*)

Model of a multivariate polynomial function of first order on y (actually not really multivariate so).

pvpumpingsystem.function_models.polynomial_5

pvpumpingsystem.function_models.**polynomial_5**(*x*, *y_intercept*, *a*, *b*, *c*, *d*, *e*)

Model of a polynomial function of fifth order.

pvpumpingsystem.function_models.polynomial_4

pvpumpingsystem.function_models.**polynomial_4**(*x*, *y_intercept*, *a*, *b*, *c*, *d*)

Model of a polynomial function of fourth order.

pvpumpingsystem.function_models.polynomial_3

pvpumpingsystem.function_models.**polynomial_3**(*x*, *y_intercept*, *a*, *b*, *c*)
Model of a polynomial function of third order.

pvpumpingsystem.function_models.polynomial_2

pvpumpingsystem.function_models.**polynomial_2**(*x*, *y_intercept*, *a*, *b*)
Model of a polynomial function of second order.

pvpumpingsystem.function_models.polynomial_1

pvpumpingsystem.function_models.**polynomial_1**(*x*, *y_intercept*, *a*)
Model of a polynomial function of first order, i.e. a linear function.

pvpumpingsystem.function_models.polynomial_divided_2_1

pvpumpingsystem.function_models.**polynomial_divided_2_1**(*x*, *a*, *b*, *c*)
Model of a polynomial function of second order divided by x.

pvpumpingsystem.waterproperties.water_prop

pvpumpingsystem.waterproperties.**water_prop**(*name*, *T*)
Function giving water property requested.

Parameters

- **name** (*str*) – Options are: 'temp', 'pres', 'vf': fluid specific volume [m3/kg], 'rhof': fluid density [kg/m3] - reverse of vf, 'vg', 'hfg', 'Cpf', 'Cpg', 'muf': dynamic viscosity, 'mug', 'nug': cinematic viscosity of vapor (gas), 'nuf': cinematic viscosity of fluid water (uses 'muf' and 'rhof') 'kf', 'kg', 'Prf', 'Prg', 'st', 'betaf'
- **T** (*float,*) – Temperature for which the property is searched N.B.: if name == 'temp', T must be replaced by pression [in bar]

Returns Value of physical quantity requested through parameter 'name'.

Return type float

pvpumpingsystem.finance.initial_investment

pvpumpingsystem.finance.**initial_investment**(*pvps*, *labour_price_coefficient=0.2*, ***kwargs*)
Function computing the initial investment cost.

Parameters

- **pvps** (*pvpumpingsystem.PVPumpSystem,*) – The Photovoltaic pumping system whose cost is to be analyzed.
- **labour_price_coefficient** (*float, default is 0.2*) – Ratio of the price of labour and secondary costs (wires, racks (can be expensive!), transport of materials, etc) on initial investment. It is considered at 0.2 in Gualteros (2017), but is more around 0.40 in Tarpuy(Peru) case.

Returns Initial investment for the whole pumping system.

Return type float

pvpumpingsystem.finance.net_present_value

pvpumpingsystem.finance.**net_present_value**(*pvps*, *discount_rate=0.02*, *labour_price_coefficient=0.2*, *opex=0*, *lifespan_pv=30*, *lifespan_mppt=14*, *lifespan_pump=12*)

Function computing the net present value of a PVPS

Parameters

- **pvps** (*pvpumpingsystem.PVPumpSystem,*) – The photovoltaic pumping system to evaluate.
- **discount_rate** (*float, default is 0.02*) – Dicsount rate.
- **labour_price_coefficient** (*float, default is 0.2*) –

  Ratio of the price of labour on capital cost. Example: If labour_price_coefficient = 0.2 (20%), it is considered that a 1000 USD PV array will cost 200 USD more to be installed on site.
- **opex** (*float, default is 0*) – Yearly operational expenditure of the pvps.
- **lifespan_pv** (*float, default is 30*) – Lifespan of the photovoltaic modules in years. It is also considered as the lifespan of the whole system.
- **lifespan_mppt** (*float, default is 14*) – Lifespan of the mppt in years.
- **lifespan_pump** (*float, default is 12*) – Lifespan of the pump in years.

Returns The net present value of the PVPS

Return type float

CHAPTER 5

Citing pvpumpingsystem

——still to come—— Paper currently under review at Journal of Open Source Software.

Index

Symbols

A

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

V

W