pvlib 2023 update: pvlib-python, pvanalytics, twoaxistracking

pvlib-python
Cliff Hansen (Sandia)
Kevin Anderson (Sandia)
Will Holmgren (DNV)
Mark Mikofski (DNV)
Adam R. Jensen (DTU)
Anton Driesse (PV Performance Labs)

pvanalytics
Cliff Hansen (Sandia)
Will Vining (Sandia)
Kevin Anderson (Sandia)
Kirsten Perry (NREL)

twoaxistracking
Kevin Anderson (Sandia)
Adam R. Jensen (DTU)

PV Performance Modeling and Monitoring Workshop
Salt Lake City, May 9, 2023

PV Performance Labs

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Contents

1 What is pvlib?

2 What is pvlib python

3 Documentation and Tutorials

4 Community

5 Enhancements since PVPMC 2022

6 What's next?

7 Introductions to pvanalytics and twoaxistracking

What is pvlib?

A python ecosystem of compatible packages for PV systems modeling and analysis that are **community-driven**, **free**, **open-source**, and **well-documented**

pvlib-python

Library of functions for weather-to-power modeling

Customizable end-to-end PV system modeling (ModelChain)

Batteries-included data import library

pvanalytics

Library of functions for analysis of data from PV systems

Filtering and quality checks

Feature recognization: e.g., label inverter clipping

twoaxistracking

Simulate two-axis tracking solar collectors

Emphasis on self-shading

Find us at: https://github.com/pvlib

pvlib

What is pvlib python?

A python library for PV performance modeling

Modeling Toolbox

Stand-alone models for:

- Atmosphere
- Solar position
- Transposition
- Bifacial
- Temperature
- Clear-sky
- Snow
- Soiling
- Shading
- I-V curves
- Inverters
- IAM

...and more!

Weather-to-power workflow

Customizable end-to-end PV system modeling (ModelChain)

Scriptable and automatable by design

Data I/O

Batteries-included data import:

- TMY
- EPW
- NSRDB
- PVGIS
- CAMS
- ECMWF MACC
- SURFRAD
- SOLRAD
- MIDC
- BSRN
- UO SRML
- NOAA USCRN

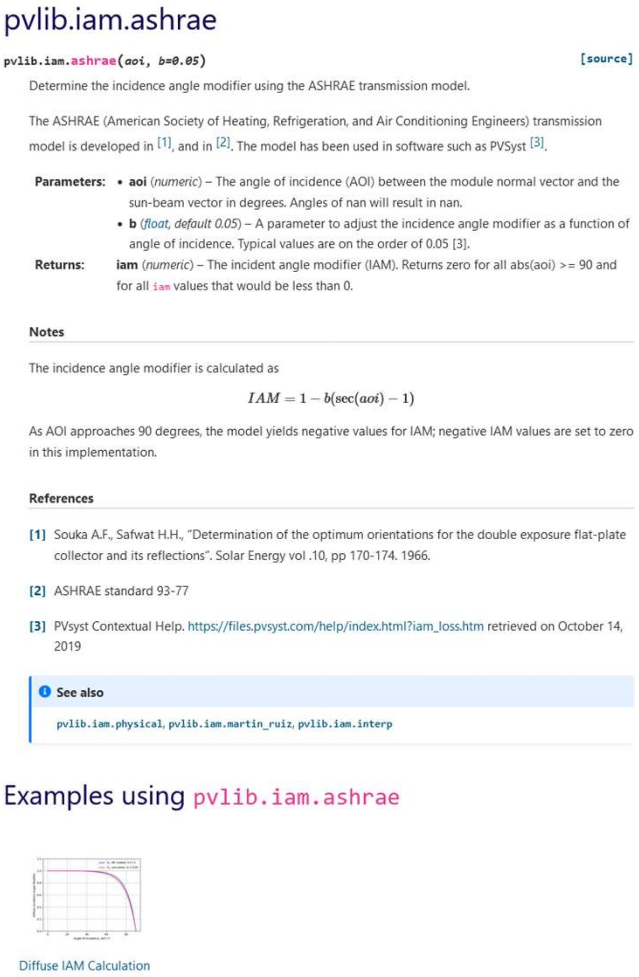
pvlib python Documentation: Model Descriptions

Each model function has a page with:

- Brief model description
- Inputs: description, data types, units
- Outputs: description, date types, units
- Published reference(s) for the model
- Links to other relevant functions
- Links to relevant gallery examples
- Other notes as needed

Several hundred model-level pages, all built automatically from in-code documentation

https://pvlib-python.readthedocs.io



pvlib.iam.ashrae

`pvlib.iam.ashrae(aoi, b=0.05)` [source]

Determine the incidence angle modifier using the ASHRAE transmission model.

The ASHRAE (American Society of Heating, Refrigeration, and Air Conditioning Engineers) transmission model is developed in [1], and in [2]. The model has been used in software such as PVSyst [3].

Parameters:
- **aoi** (*numeric*) – The angle of incidence (AOI) between the module normal vector and the sun-beam vector in degrees. Angles of nan will result in nan.
- **b** (*float, default 0.05*) – A parameter to adjust the incidence angle modifier as a function of angle of incidence. Typical values are on the order of 0.05 [3].

Returns: **iam** (*numeric*) – The incident angle modifier (IAM). Returns zero for all abs(aoi) >= 90 and for all `iam` values that would be less than 0.

Notes

The incidence angle modifier is calculated as

$$IAM = 1 - b(\sec(aoi) - 1)$$

As AOI approaches 90 degrees, the model yields negative values for IAM; negative IAM values are set to zero in this implementation.

References

[1] Souka A.F., Safwat H.H., "Determination of the optimum orientations for the double exposure flat-plate collector and its reflections". Solar Energy vol .10, pp 170-174. 1966.

[2] ASHRAE standard 93-77

[3] PVsyst Contextual Help. https://files.pvsyst.com/help/index.html?iam_loss.htm retrieved on October 14, 2019

See also

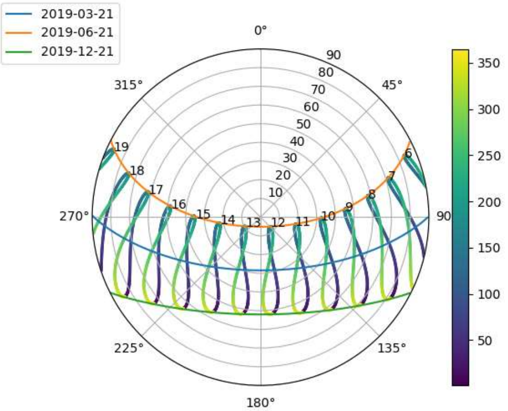
`pvlib.iam.physical`, `pvlib.iam.martin_ruiz`, `pvlib.iam.interp`

Examples using `pvlib.iam.ashrae`

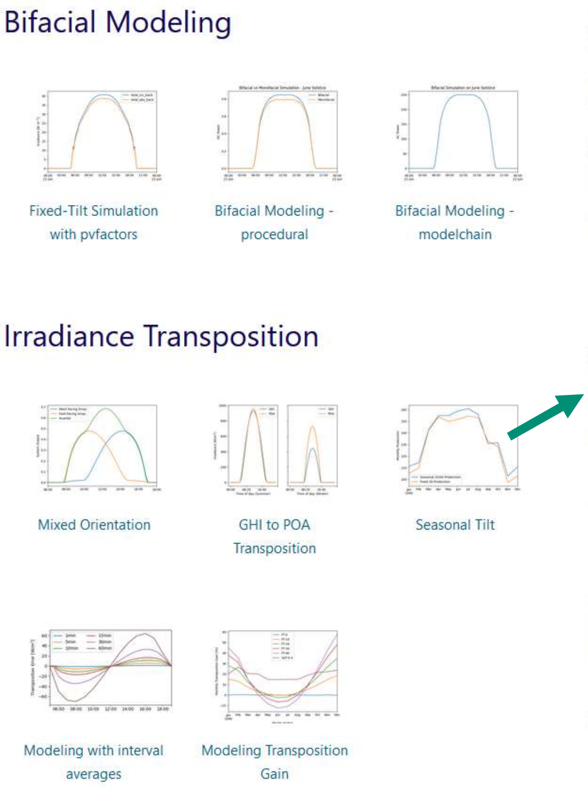
Diffuse IAM Calculation

pvlib python Documentation: Example Gallery

pvlib “cookbook” -- small self-contained scripts for various modeling tasks, intended as a starting point for your own code.



Want to make cool plots like this one?
Check out the example gallery!



Bifacial Modeling

Fixed-Tilt Simulation with pvfactors

Bifacial Modeling - procedural

Bifacial Modeling - modelchain

Irradiance Transposition

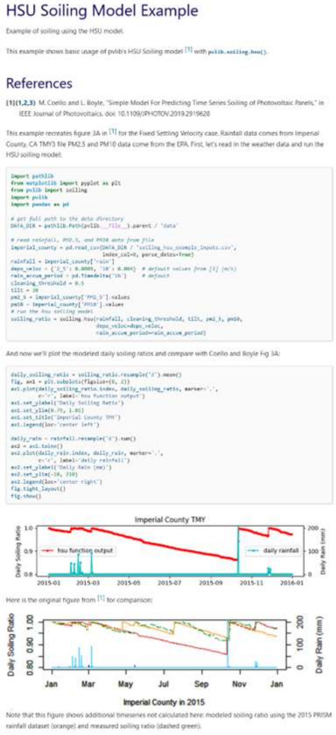
Mixed Orientation

GHI to POA Transposition

Seasonal Tilt

Modeling with interval averages

Modeling Transposition Gain



HSU Soiling Model Example

Example of soiling using the HSU model.

This example shows basic usage of pvlib's HSU Soiling model [1] with `pvlib.soiling.hsu()`.

References

[1](1,2,3) M. Coello and L. Boyle, "Simple Model For Predicting Time Series Soiling of Photovoltaic Panels," in IEEE Journal of Photovoltaics. doi: 10.1109/JPHOTOV.2019.2919628

This example recreates figure 3A in [1] for the Fixed Settling Velocity case. Rainfall data comes from Imperial County, CA TMY3 file PM2.5 and PM10 data come from the EPA. First, let's read in the weather data and run the HSU soiling model:

```
import pathlib
from matplotlib import pyplot as plt
from pvlib import soiling
import pvlib
import pandas as pd

# get full path to the data directory
DATA_DIR = pathlib.Path(pvlib.__file__).parent / 'data'

# read rainfall, PM2.5, and PM10 data from file
imperial_county = pd.read_csv(DATA_DIR / 'soiling_hsu_example_inputs.csv',
                              index_col=0, parse_dates=True)
rainfall = imperial_county['rain']
depo_veloc = {'2_5': 0.0009, '10': 0.004}  # default values from [1] (m/s)
rain_accum_period = pd.Timedelta('1h')     # default
cleaning_threshold = 0.5
tilt = 30
pm2_5 = imperial_county['PM2_5'].values
pm10 = imperial_county['PM10'].values
# run the hsu soiling model
soiling_ratio = soiling.hsu(rainfall, cleaning_threshold, tilt, pm2_5, pm10,
                            depo_veloc=depo_veloc,
                            rain_accum_period=rain_accum_period)
```

And now we'll plot the modeled daily soiling ratios and compare with Coello and Boyle Fig 3A:

```
daily_soiling_ratio = soiling_ratio.resample('d').mean()
fig, ax1 = plt.subplots(figsize=(8, 2))
ax1.plot(daily_soiling_ratio.index, daily_soiling_ratio, marker='.',
         c='r', label='hsu function output')
ax1.set_ylabel('Daily Soiling Ratio')
ax1.set_ylim(0.79, 1.01)
ax1.set_title('Imperial County TMY')
ax1.legend(loc='center left')

daily_rain = rainfall.resample('d').sum()
ax2 = ax1.twinx()
ax2.plot(daily_rain.index, daily_rain, marker='.',
         c='c', label='daily rainfall')
ax2.set_ylabel('Daily Rain (mm)')
ax2.set_ylim(-10, 210)
ax2.legend(loc='center right')
fig.tight_layout()
fig.show()
```

Here is the original figure from [1] for comparison:

Note that this figure shows additional timeseries not calculated here: modeled soiling ratio using the 2015 PRISM rainfall dataset (orange) and measured soiling ratio (dashed green).

https://pvlib-python.readthedocs.io/en/stable/gallery/index.html

pvlib

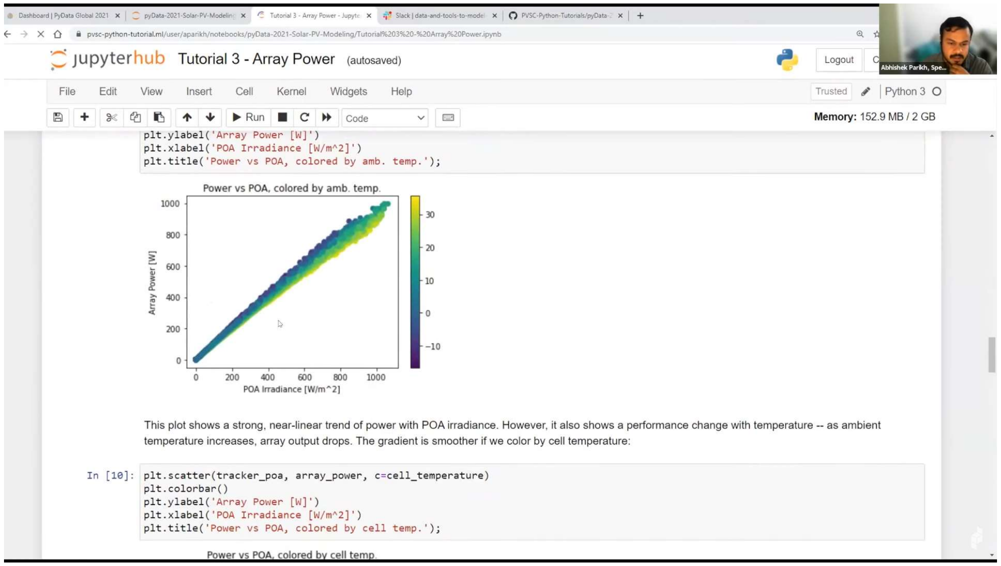
pvlib python: Tutorials

Interactive tutorials for:

- Modeling concepts
- Implementation in pvlib

The next one is here, tomorrow afternoon!
Led by Adam Jensen and Kevin Anderson

50th IEEE PVSC (11 June 2023), led by
Silvana Ovaitt and Mark Mikofski


Dashboard | PyData Global 2021
pyData-2021-Solar-PV-Modeling
Tutorial 3 - Array Power - Jupyte
Slack | data-and-tools-to-model
PVSC-Python-Tutorials/pyData-2
pvsc-python-tutorial.ml/user/aparikh/notebooks/pyData-2021-Solar-PV-Modeling/Tutorial%203%20-%20Array%20Power.ipynb
jupyterhub
Tutorial 3 - Array Power
(autosaved)
Logout
Abhishek Parikh, Spe...
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3
Run
Code
Memory: 152.9 MB / 2 GB
Power vs POA, colored by amb. temp.
Array Power [W]
POA Irradiance [W/m^2]
0
200
400
600
800
1000
30
20
10
0
−10
Power vs POA, colored by cell temp.

```
plt.ylabel('Array Power [W]')
plt.xlabel('POA Irradiance [W/m^2]')
plt.title('Power vs POA, colored by amb. temp.');
```

This plot shows a strong, near-linear trend of power with POA irradiance. However, it also shows a performance change with temperature -- as ambient temperature increases, array output drops. The gradient is smoother if we color by cell temperature:

```
In [10]: plt.scatter(tracker_poa, array_power, c=cell_temperature)
         plt.colorbar()
         plt.ylabel('Array Power [W]')
         plt.xlabel('POA Irradiance [W/m^2]')
         plt.title('Power vs POA, colored by cell temp.');
```

PVPMC 2022: https://github.com/PVSC-Python-Tutorials/PVPMC_2022

PVSC 2021: https://github.com/PVSC-Python-Tutorials/PVSC48-Python-Tutorial

PyData Global 2021
Youtube recording: https://www.youtube.com/watch?v=sweUakFg3I8
Source material: https://github.com/PVSC-Python-Tutorials/pyData-2021-Solar-PV-Modeling

pvlib python: Community Growth

Google Group (user discussion, announcements)

- ~~600+~~ 700+ members
- https://groups.google.com/g/pvlib-python

GitHub (code development)

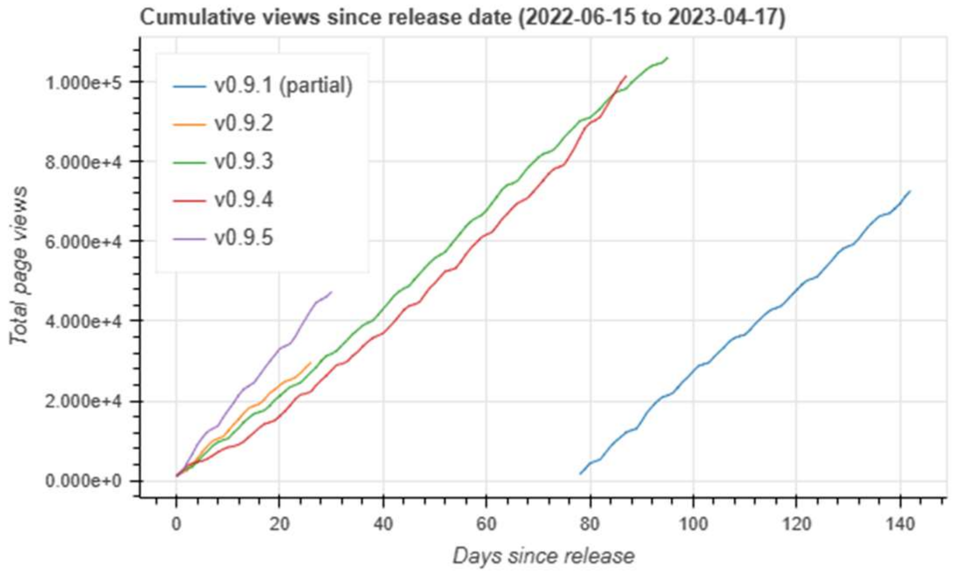
- Code contributions from ~~80+~~ 90+ people
- https://github.com/pvlib/pvlib-python

Citations

- 300+ since 2022
- Influence outside of PV modeling, e.g.,

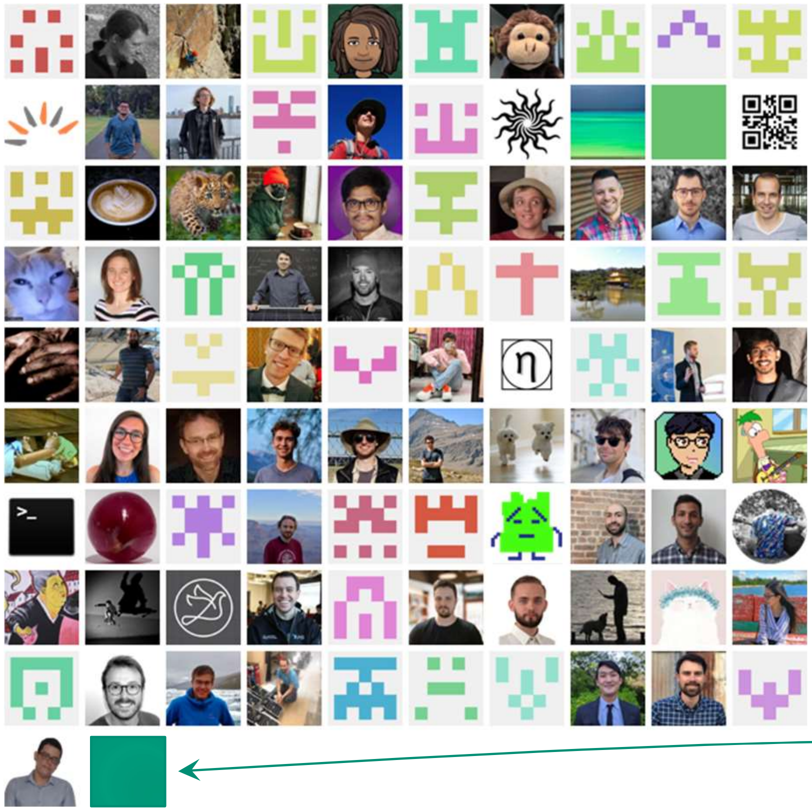
J. Rowland et al., *Scale-dependent influence of permafrost on riverbank erosion rates*. ESS Open Archive. February 09, 2023.

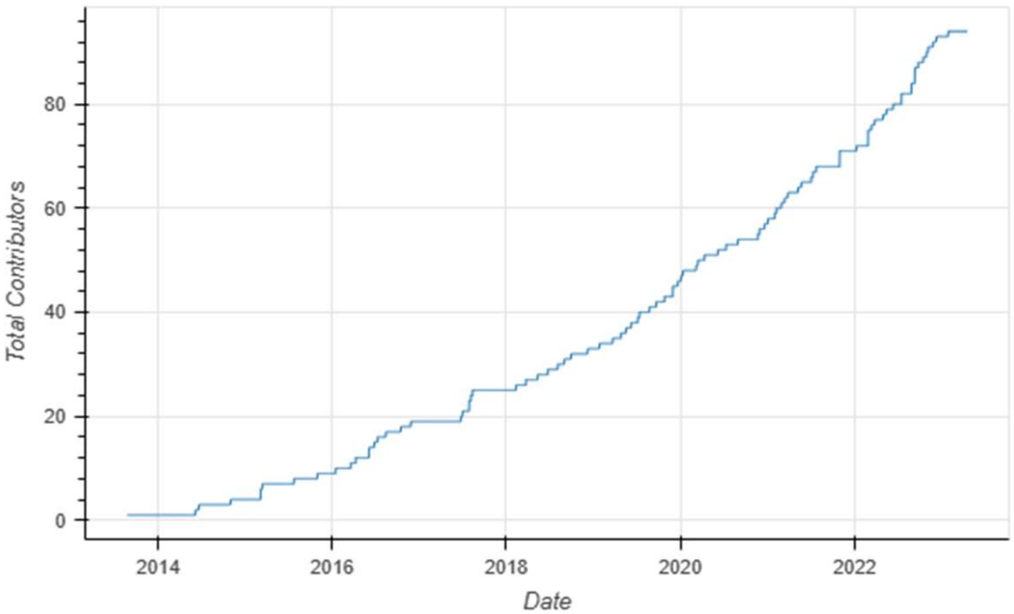
10k page views / month



pvlib python documentation page views

pvlib python: GitHub Contributors





*Not all contributions are code!

This software is made possible by contributions from people like you. <u>You can help!</u>

https://pvlib-python.readthedocs.io/en/stable/contributing.html

pvlib python Enhancements (v0.9.3 – v0.9.5)

pvlib.irradiance

- Boland sky diffuse model

pvlib.iam

- schlick
- schlick_diffuse

pvlib.spectrum

- Spectral mismatch calculations (integration over spectral range)

pvlib.snow

- Townsend model (corrected in v0.9.5)

pvlib.temperature

- Coefficient translator (e.g., between Faiman and SAPM)
- faiman_rad (adds radiative loss term)

pvlib.pvarray

- pvefficiency_adr (and fit_pvefficiency_adr)

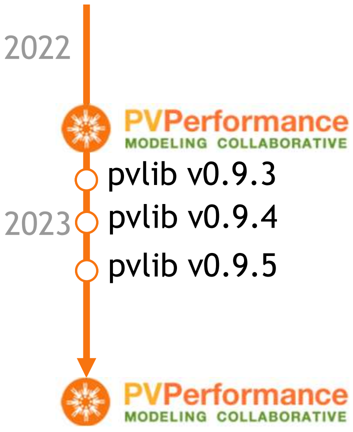
***pvarray** will eventually contain DC power models (now in pvlib.pvsystem)

pvlib.ivtools

- astm_e1036 (extracts Voc etc. from data per ASTM standard)

pvlib.bifacial

- Can specify isotropic or Hay-Davies sky diffuse models
- Can vectorize infinite_sheds for faster calculation (but uses more memory)



2022

PVPerformance MODELING COLLABORATIVE

pvlib v0.9.3

2023 pvlib v0.9.4

pvlib v0.9.5

PVPerformance MODELING COLLABORATIVE

Full details: https://pvlib-python.readthedocs.io/en/stable/whatsnew.html

Pvlib python: What’s next?

Model parameter tools

- Module IV model parameter translator (e.g., CEC model ↔ Pvsyst model)

Documentation revisions

- Rewrite/reorg the docs to follow an intentional strategy instead of the current ad-hoc “pile of info”

Fill in some modeling gaps

- Transformer losses, shading losses, inverter operations off unity power factor

What else? What would you like to contribute? Come to the pvlib user discussion tomorrow, 3pm

What is pvanalytics?

- Workflow-independent library of base functions
- Fully compatible with pvlib-python
- Launched Feb 2020, v0.1.3 Dec 2022
- 6 contributors, 23 forks, 69 stars

Quality control

- Plausibility of irradiance and weather measurements
- Identification of missing, interpolated, or stale data
- Outlier detection
- Identification of timestamp problems such as daylight savings shifts

Feature identification

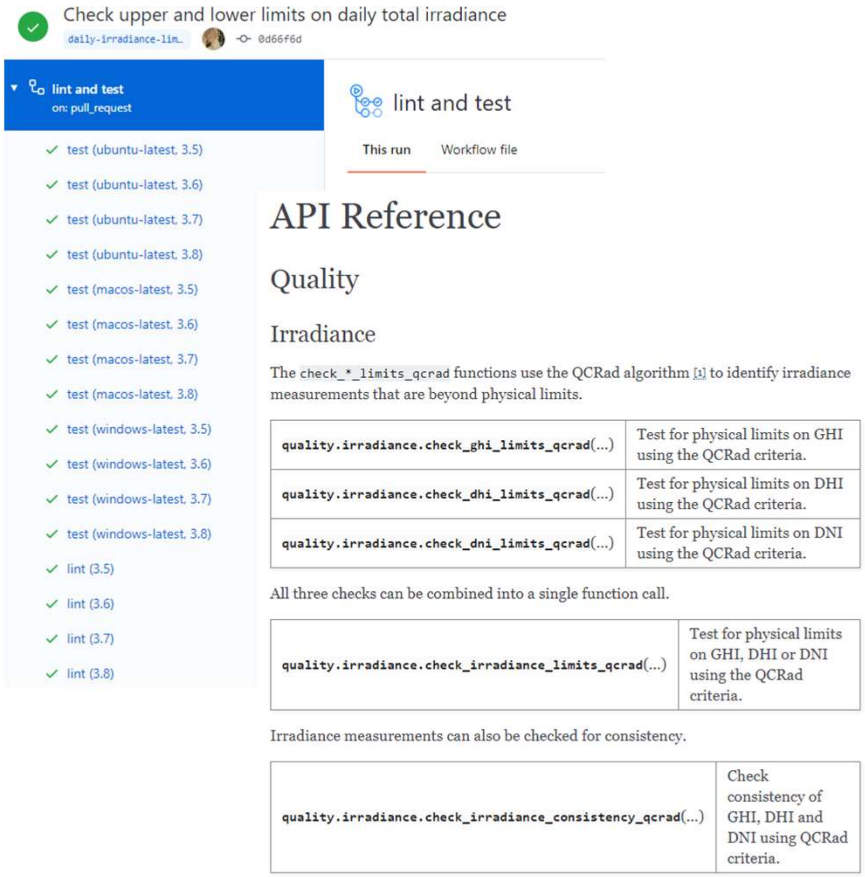
- Inverter clipping
- Clear-sky periods
- Day/night detection from power or irradiance

Identification of system properties

- Tilt and azimuth from power data
- Differentiation between fixed and tracking PV systems

Metrics

- NREL weather corrected performance ratio



Check upper and lower limits on daily total irradiance

✓ test (ubuntu-latest, 3.5)
✓ test (ubuntu-latest, 3.6)
✓ test (ubuntu-latest, 3.7)
✓ test (ubuntu-latest, 3.8)
✓ test (macos-latest, 3.5)
✓ test (macos-latest, 3.6)
✓ test (macos-latest, 3.7)
✓ test (macos-latest, 3.8)
✓ test (windows-latest, 3.5)
✓ test (windows-latest, 3.6)
✓ test (windows-latest, 3.7)
✓ test (windows-latest, 3.8)
✓ lint (3.5)
✓ lint (3.6)
✓ lint (3.7)
✓ lint (3.8)

API Reference

Quality

Irradiance

The `check_*_limits_qcrad` functions use the QCRad algorithm [1] to identify irradiance measurements that are beyond physical limits.

`quality.irradiance.check_ghi_limits_qcrad(...)`	Test for physical limits on GHI using the QCRad criteria.
`quality.irradiance.check_dhi_limits_qcrad(...)`	Test for physical limits on DHI using the QCRad criteria.
`quality.irradiance.check_dni_limits_qcrad(...)`	Test for physical limits on DNI using the QCRad criteria.

All three checks can be combined into a single function call.

`quality.irradiance.check_irradiance_limits_qcrad(...)`	Test for physical limits on GHI, DHI or DNI using the QCRad criteria.

Irradiance measurements can also be checked for consistency.

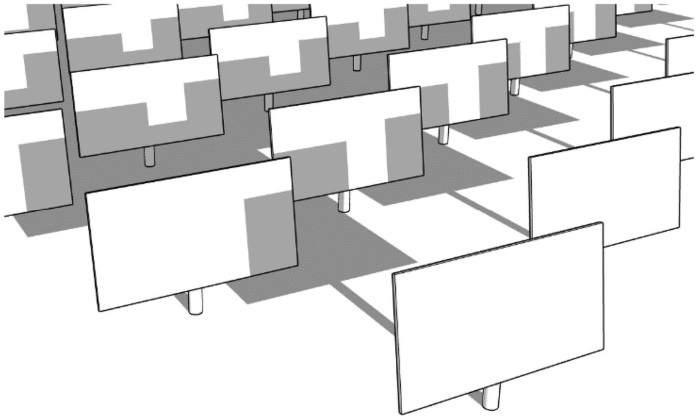
`quality.irradiance.check_irradiance_consistency_qcrad(...)`	Check consistency of GHI, DHI and DNI using QCRad criteria.

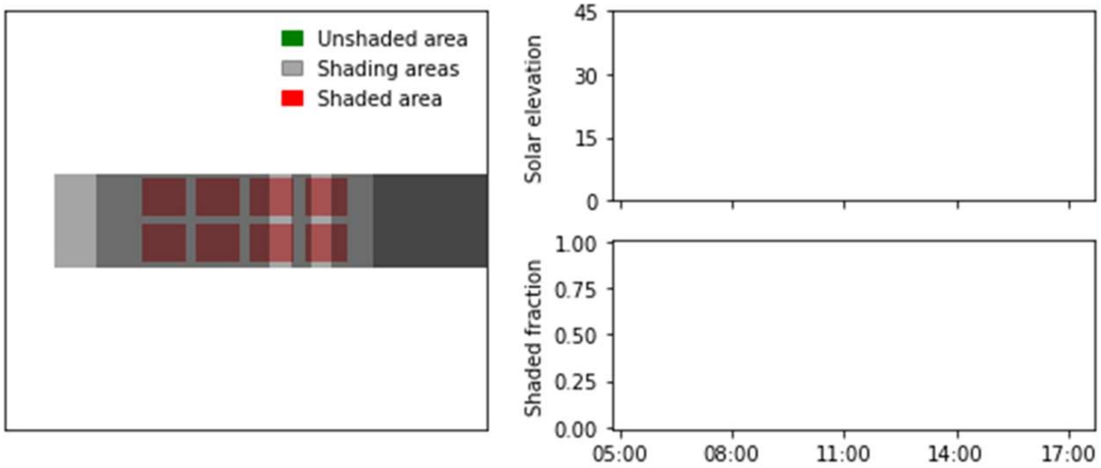
What is pvlib/twoaxistracking?

Shading of two-axis trackers

- Fully customizable field layouts
- Arbitrary rectangular panel shape
- Differentiation between active and frame area
- Extensive documentation, validated against literature

Validated against literature!





Thank You

www.github.com/pvlib/pvlib-python
https://pvlib-python.readthedocs.io

www.github.com/pvlib/pvanalytics
https://pvanalytics.readthedocs.io

SAND 2023-04324C

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology &Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.