SAND2016-5634C

PVLIB: Open Source Photovoltaic Performance Modeling Functions for Matlab and Python

IEEE PVSC 43
Portland, OR June, 2016

Joshua S. Stein, **William F. Holmgren**, Jessica Forbess, and Clifford W. Hansen

The University of Arizona

Sunshine Analytics

Sandia National Laboratories

PVPerformance MODELING COLLABORATIVE

Exceptional service in the national interest

U.S. Department of Energy

NNSA

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

A bit of History

- Matlab version started as an internal tool at Sandia in 2010-2011 developed to help standardize analyses across the PV group.
  - PVLIB Version 1.0 – May 2012 – 29 functions
  - PVLIB Version 1.1 – Jan 2013 – 38 functions
  - PVILB Version 1.2 – Dec 2014 – 44 functions
  - PVLIB Version 1.3 – Dec 2015 – 59 functions
- Python version was initially developed from 2013-2014 by Rob Andrews under contract from Sandia.
- 2015 Python PVLIB converted to Open Source GitHub project largely managed by Will Holmgren at University of Arizona.
- Download links available on PVPMC website:
  - https://pvpmc.sandia.gov click on Applications and Tools link

Two Versions

PVLIB Matlab

- Integrates with Matlab environment (help, search)
- Extensively tested by Sandia National Laboratories
- No extra toolboxes required

- Requires Matlab license ($$)
- Not fully integrated into GitHub (yet)
- Updates have been slow to be released.
- No formal way report/fix bugs except for email.

PVLIB Python

- Free
- Can be integrated with a huge ecosystem of Python libraries
- Comprehensive unit tests
- A real Python library, not just a wrapper with awkward syntax
- High-level features that do not (yet?) exist in PVLIB MATLAB
- A growing community on GitHub

- Getting started with Python, NumPy, SciPy can be challenging
- Not as many functions as PVLIB MATLAB

New Functions in Matlab V.1.3

- **pvl_FSspeccorr** – Spectral mismatch modifier function contributed by First Solar based on precipitable water.
- **pvl_calcPwat** – function to estimate precipitable water content
- **pvl_huld** – PV performance model of Huld et al., 2011
- **pvl_PVsyst_parameter_estimation** – function to estimate PVsyst module parameters from IV curves.
- **pvl_calcparams_PVsyst** – Calculates the five parameters for an IV curve using the PVsyst model.
- **pvl_desoto_parameter_estimation** – function to estimate Desoto module parameters from IV curves.
- **pvl_getISDdata** – Functions to access ground measured weather data from NOAA's Integrated Surface Data network

New Functionality in Python V0.2, V0.3

- **PVSystem** and **SingleAxisTracker** classes – abstractions that can help with standard modeling tasks.
- **Standardized variable names** throughout library
- **Conda** installation packages on the pvlib and conda-forge channels
- **Location.from_tmy** – create a Location object from a TMY file
- **lookup_linke_turbidity** – refactored out of ineichen, supports daily monthly → daily interpolation
- **Ported more functions from PVLIB MATLAB**

see documentation for full listing

pvlib-python.readthedocs.io/en/latest/whatsnew.html

Simple Usage Examples
Clear Sky Irradiance

MATLAB

pvl_clearsky_ineichen(Time, Location, varargin)

“Time” and “Location” are Matlab structures that contain vectors and scalars describing time steps and locations

Python

```
# function
ineichen(time, latitude, longitude, altitude=0, …)

# object oriented
portland = Location(45.5, -122.7, 15, 'Etc/GMT+8’)
portland.get_clearsky(time, method=‘ineichen’)
```

Next PVLIB Python version will add the Simplified Solis clear sky model

Script Example:

Calculate Spectral Correction vs. Air Mass and Plot Results

Matlab

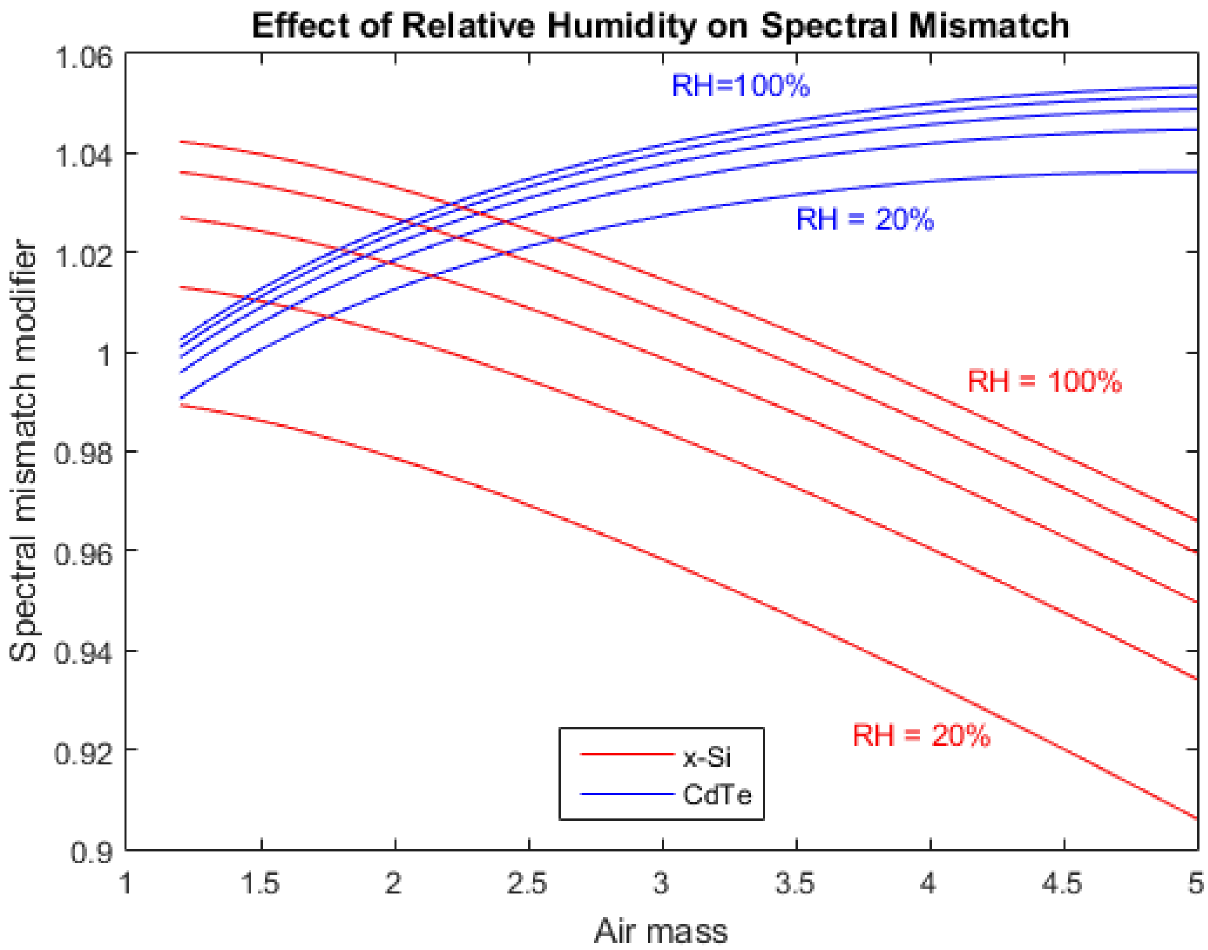
```
figure
AMa = 1.2:0.1:5;
for rh = 20:20:100
    Pwat = pvl_calcPwat(25,rh);
    MCdTe = pvl_FSspeccorr(Pwat, AMa,
                        'CdTe');
    MxSi = pvl_FSspeccorr(Pwat, AMa,
                        'xSi');
    plot(AMa,MCdTe,'r-')
    hold all
    plot(AMa,MxSi,'b-')
end

xlabel('Air mass')
ylabel('Spectral mismatch modifier')
title('Effect of Relative Humidity on Spectral
Mismatch')
legend('x-Si','CdTe','Location','South')
```

Python

```
airmass = np.linspace(1.2, 5)
rhs = np.linspace(20, 100, 5)
pws = gueymard94(25, rhs)
for pw in pws:
    cdte = first_solar_spectral_correction(
                pw, airmass, 'CdTe')
    xsi = first_solar_spectral_correction(
                pw, airmass, 'xSi')
    plt.plot(airmass, cdte, 'r-')
    plt.plot(airmass, xsi, 'b-')

plt.xlabel('Air mass')
plt.ylabel('Spectral mismatch modifier')
plt.title('Effect of Relative Humidity on Spectral
Mismatch')
plt.legend(['x-Si', 'CdTe'], loc='lower center')
```

Code not yet merged into PVLIB Python

Result



Note: RH labels were added later for clarity

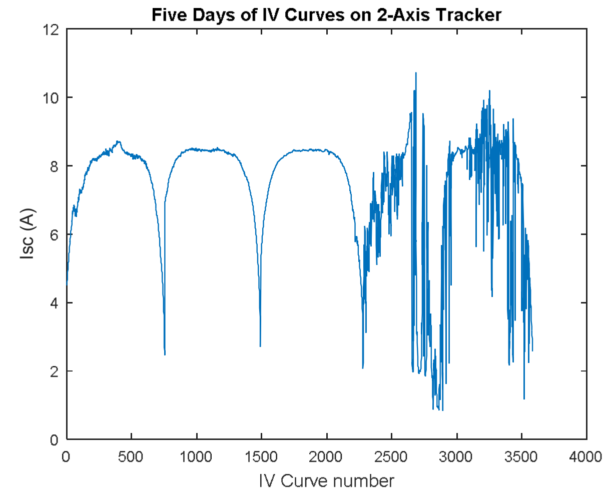
PVsyst PAN File Example

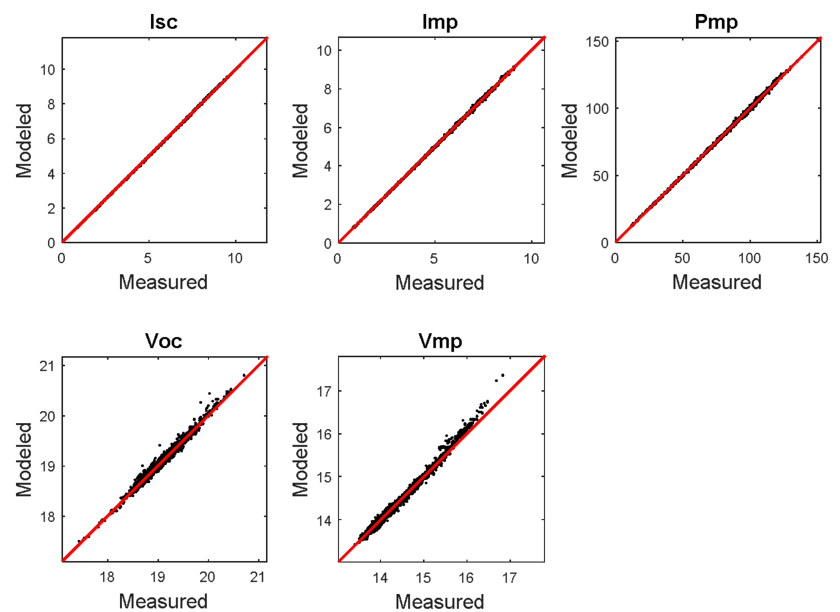
- 3,585 IV curves measured outdoors over 5 days on a 36 cell Mitsubishi c-Si module
- We use one function to estimate parameters:
  - pvl_PVsyst_parameter_estimation

  [PVsyst oflag] = pvl_PVsyst_parameter_estimation(IVCurves, Specs, Const, maxiter, eps1, graphic);
- And two functions to run the model
  - pvl_calcparams_PVsyst
  - pvl_singlediode

  [IL, Io, Rs, Rsh, nNsVth] = pvl_calcparams_PVsyst([IVCurves.Ee],[IVCurves.Tc],Specs.aIsc,PVsyst);

  Modeled = pvl_singlediode(IL, Io, Rs, Rsh, nNsVth);

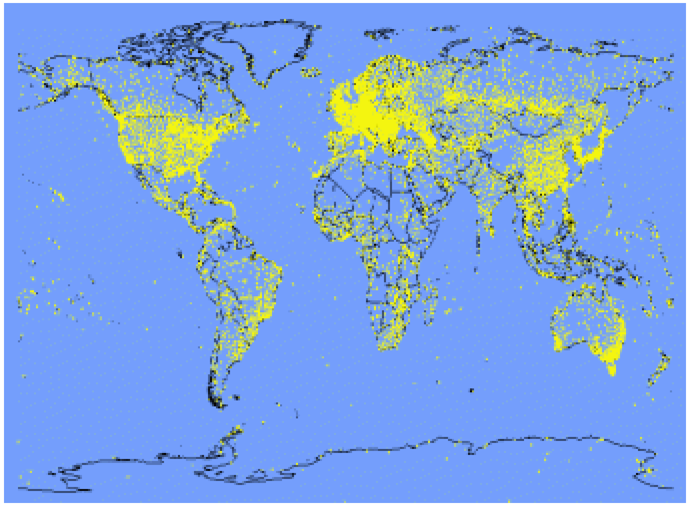




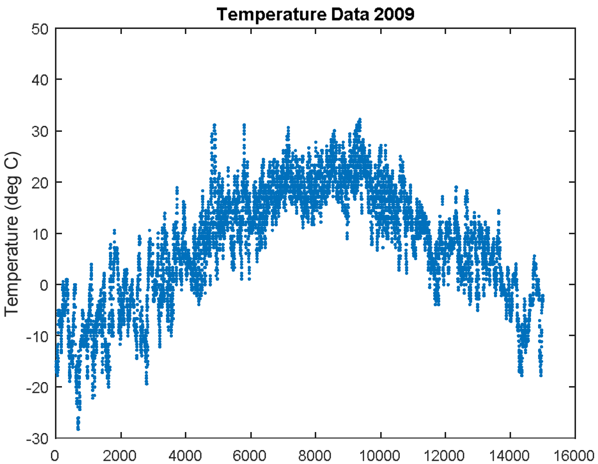
Finding Weather Data using PVLIB

- Two functions allow users to obtain measured weather data from NOAA's Integrated Surface Database (ISD)
  - pvl_getISDdata
  - pvl_readISH
- Example below shows how to retrieve data for a specified location and year. The functions will find the closest station.
- We chose Williston, Vermont and 2009. It found a station within a few kms.

```
fname = pvl_getISDdata(44.465,-73.105,2009,archive);

data = pvl_readISH([archive '\' fname]);
```



Map showing the locations of ISD stations



Plot of air temperature data near the point of interest from 2009 in Williston, VT

PVLIB Python

- There are two ways to run models using PVLIB python
  - Using functions and writing a script – Very explicit, easy to customize
  - Using classes – Many fewer lines of code, less easy to customize.

Functions

```
cs = pvlib.clearsky.ineichen(times, latitude, longitude, altitude=altitude)
solpos = pvlib.solarposition.get_solarposition(times, latitude, longitude)
dni_extra = pvlib.irradiance.extraradiation(times)
dni_extra = pd.Series(dni_extra, index=times)
airmass = pvlib.atmosphere.relativeairmass(solpos['apparent_zenith'])
pressure = pvlib.atmosphere.alt2pres(altitude)
am_abs = pvlib.atmosphere.absoluteairmass(airmass, pressure)
aoi = pvlib.irradiance.aoi(system['surface_tilt'], system['surface_azimuth'],
                           solpos['apparent_zenith'], solpos['azimuth'])
total_irrad = pvlib.irradiance.total_irrad(system['surface_tilt'],
                                           system['surface_azimuth'],
                                           solpos['apparent_zenith'],
                                           solpos['azimuth'],
                                           cs['dni'], cs['ghi'], cs['dhi'],
                                           dni_extra=dni_extra,
                                           model='haydavies')
temps = pvlib.pvsystem.sapm_celltemp(total_irrad['poa_global'],
                                     wind_speed, temp_air)
dc = pvlib.pvsystem.sapm(module, total_irrad['poa_direct'],
                         total_irrad['poa_diffuse'], temps['temp_cell'],
                         am_abs, aoi)
ac = pvlib.pvsystem.snlinverter(inverter, dc['v_mp'], dc['p_mp'])
```

Classes

```
system = PVSystem(module_parameters=module,
                  inverter_parameters=inverter)
location = Location(latitude, longitude, name=name, altitude=altitude)
mc = ModelChain(system, location,
                orientation_strategy='south_at_latitude_tilt')
mc.run_model(times)
annual_energy = mc.ac.sum()
```

PVLIB Python: forecasts

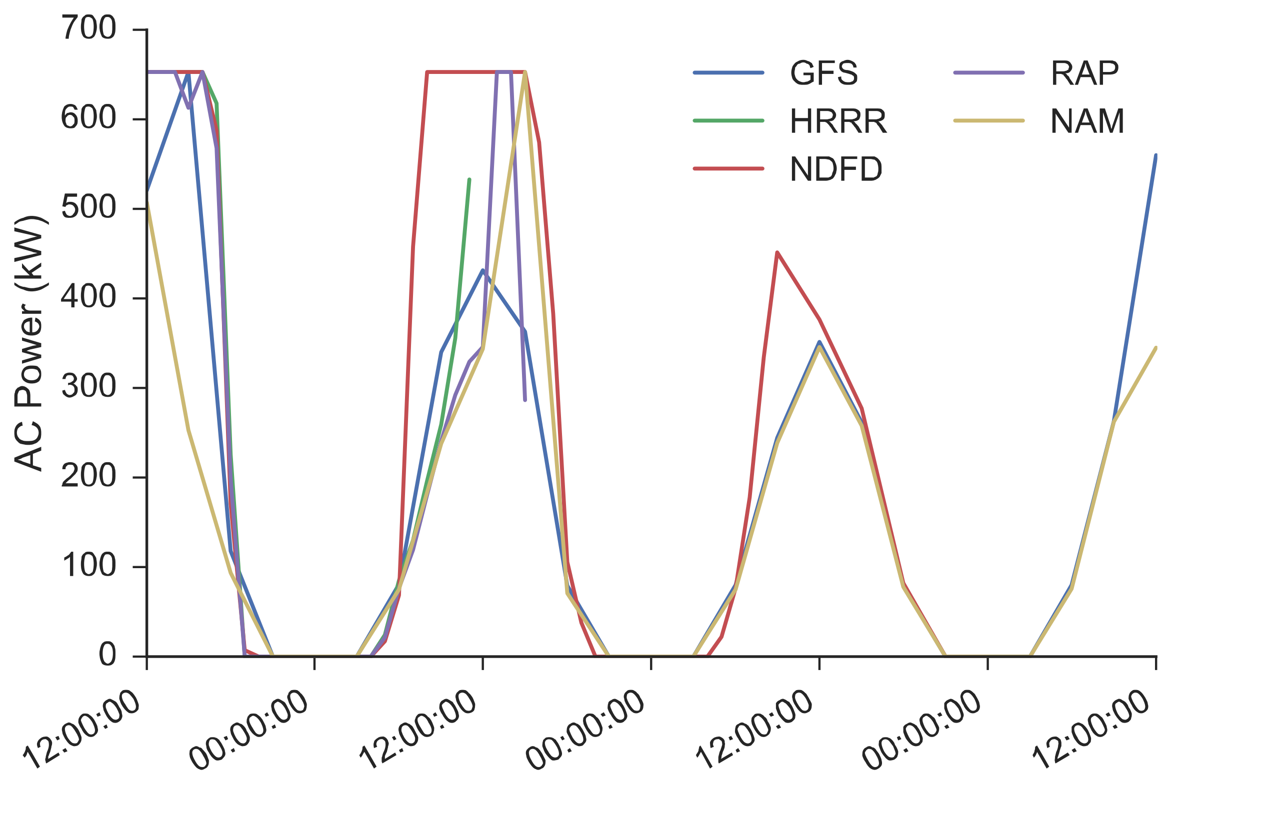
```
module = sandia_modules['Canadian_Solar_CS5P_220M___2009_']
inverter = cec_inverters['SMA_America__SC630CP_US_315V__CEC_2012_']

system = SingleAxisTracker(module_parameters=module,
   inverter_parameters=inverter, series_modules=15, parallel_modules=300)

lat, lon = 45.5, -122.7  # Portland, OR

start = pd.Timestamp.now()
end = start + pd.Timedelta(days=7)

for fx_class in [GFS, NAM, HRRR, RAP, NDFD]:
   fx_model = fx_class()
   fx_data = fx_model.get_processed_data(lat, lon, start, end)
   irradiance = fx_data[['ghi', 'dni', 'dhi']]
   weather = fx_data[['wind_speed', 'temp_air']]
   mc = ModelChain(system, fx_model.location)
   mc.run_model(fx_data.index, irradiance, weather)
   mc.ac.plot()
```

*Code not yet merged into PVLIB Python

PVLIB Python: forecasts

```
module = sandia_modules['Canadian_Solar_CS5P_220M___2009_']
inverter = cec_inverters['SMA_America__SC630CP_US_315V__CEC_2012_']

system = SingleAxisTracker(module  parameters=module,
    inverter_parameters=inverte

lat, lon = 45.5, -122.7  # Portla

start = pd.Timestamp.now()
end = start + pd.Timedelta(day

for fx_class in [GFS, NAM, HR
    fx_model = fx_class()
    fx_data = fx_model.get_proc
    irradiance = fx_data[['ghi', 'd
    weather = fx_data[['wind_sp
    mc = ModelChain(system, f
    mc.run_model(fx_data.inde
    mc.ac.plot()
```

*Code not yet merged into PVLIB F

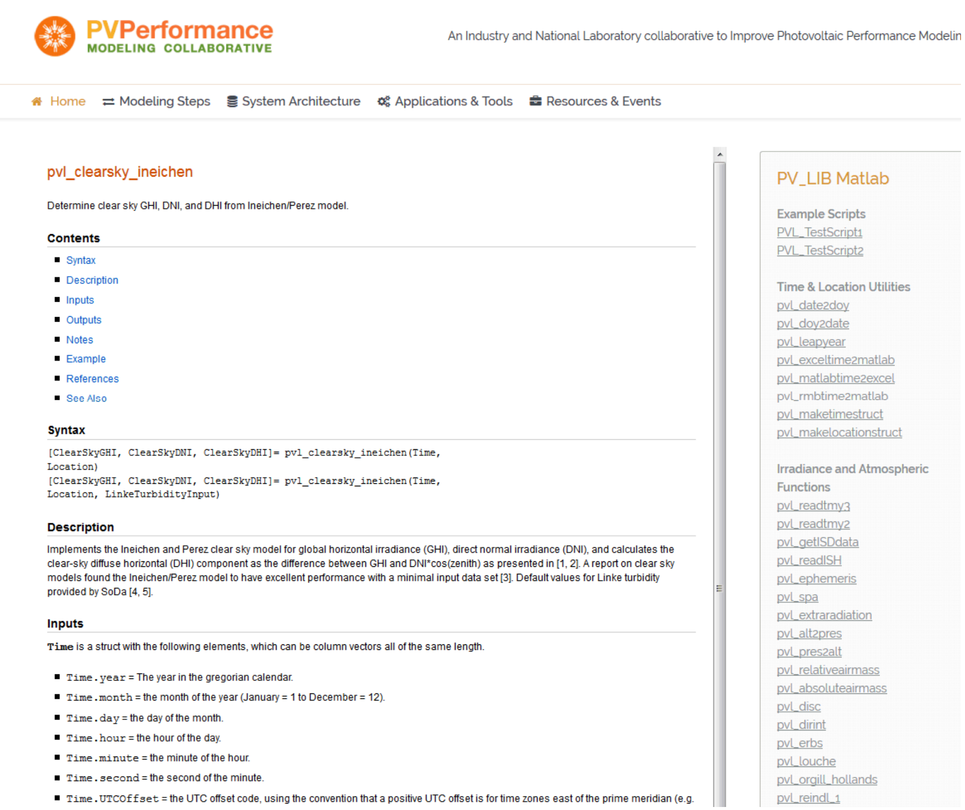


PVLIB Documentation

Sandia National Laboratories

Matlab

pvpmc.sandia.gov/applications/pv_lib-toolbox/



PVPerformance MODELING COLLABORATIVE

An Industry and National Laboratory collaborative to Improve Photovoltaic Performance Modelin

Home | Modeling Steps | System Architecture | Applications & Tools | Resources & Events

pvl_clearsky_ineichen

Determine clear sky GHI, DNI, and DHI from Ineichen/Perez model.

Contents

- Syntax
- Description
- Inputs
- Outputs
- Notes
- Example
- References
- See Also

Syntax

```
[ClearSkyGHI, ClearSkyDNI, ClearSkyDHI]= pvl_clearsky_ineichen(Time,
Location)
[ClearSkyGHI, ClearSkyDNI, ClearSkyDHI]= pvl_clearsky_ineichen(Time,
Location, LinkeTurbidityInput)
```

Description

Implements the Ineichen and Perez clear sky model for global horizontal irradiance (GHI), direct normal irradiance (DNI), and calculates the clear-sky diffuse horizontal (DHI) component as the difference between GHI and DNI*cos(zenith) as presented in [1, 2]. A report on clear sky models found the Ineichen/Perez model to have excellent performance with a minimal input data set [3]. Default values for Linke turbidity provided by SoDa [4, 5].

Inputs

`Time` is a struct with the following elements, which can be column vectors all of the same length.

- `Time.year` = The year in the gregorian calendar.
- `Time.month` = the month of the year (January = 1 to December = 12).
- `Time.day` = the day of the month.
- `Time.hour` = the hour of the day.
- `Time.minute` = the minute of the hour.
- `Time.second` = the second of the minute.
- `Time.UTCOffset` = the UTC offset code, using the convention that a positive UTC offset is for time zones east of the prime meridian (e.g.

PV_LIB Matlab

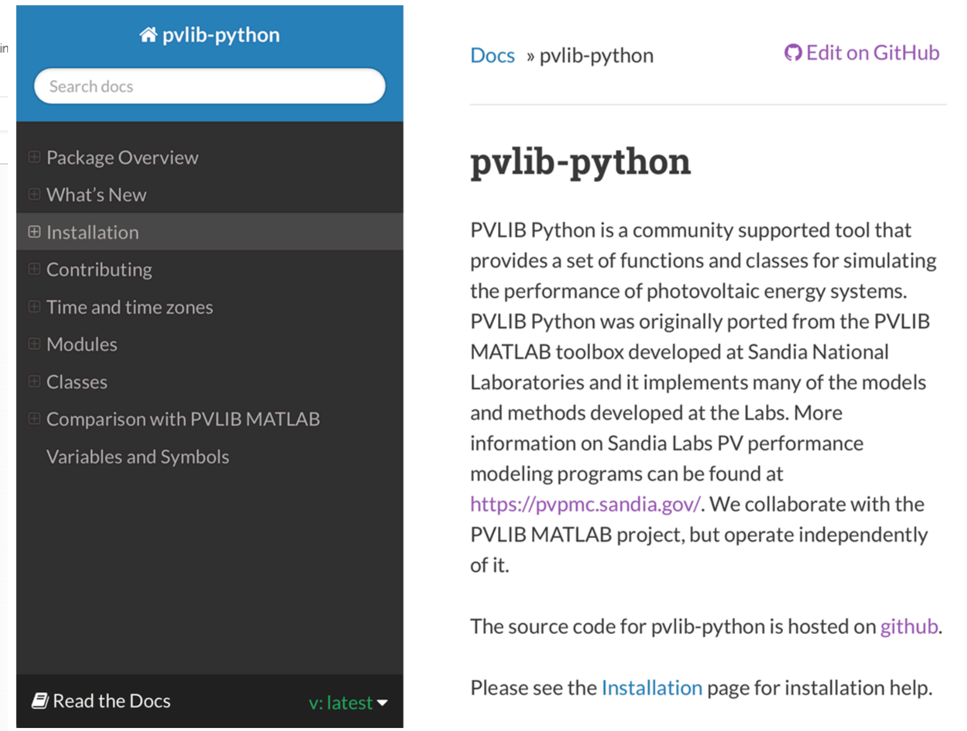
Example Scripts
PVL_TestScript1
PVL_TestScript2

Time & Location Utilities
pvl_date2doy
pvl_doy2date
pvl_leapyear
pvl_exceltime2matlab
pvl_matlabtime2excel
pvl_rmbtime2matlab
pvl_maketimestruct
pvl_makelocationstruct

Irradiance and Atmospheric Functions
pvl_readtmy3
pvl_readtmy2
pvl_getISDdata
pvl_readISH
pvl_ephemeris
pvl_spa
pvl_extraradiation
pvl_alt2pres
pvl_pres2alt
pvl_relativeairmass
pvl_absoluteairmass
pvl_disc
pvl_dirint
pvl_erbs
pvl_louche
pvl_orgill_hollands
pvl_reindl_1

Python

pvlib-python.readthedocs.io



pvlib-python

Search docs

- Package Overview
- What's New
- Installation
- Contributing
- Time and time zones
- Modules
- Classes
- Comparison with PVLIB MATLAB
- Variables and Symbols

Read the Docs v: latest

Docs » pvlib-python | Edit on GitHub

pvlib-python

PVLIB Python is a community supported tool that provides a set of functions and classes for simulating the performance of photovoltaic energy systems. PVLIB Python was originally ported from the PVLIB MATLAB toolbox developed at Sandia National Laboratories and it implements many of the models and methods developed at the Labs. More information on Sandia Labs PV performance modeling programs can be found at https://pvpmc.sandia.gov/. We collaborate with the PVLIB MATLAB project, but operate independently of it.

The source code for pvlib-python is hosted on github.

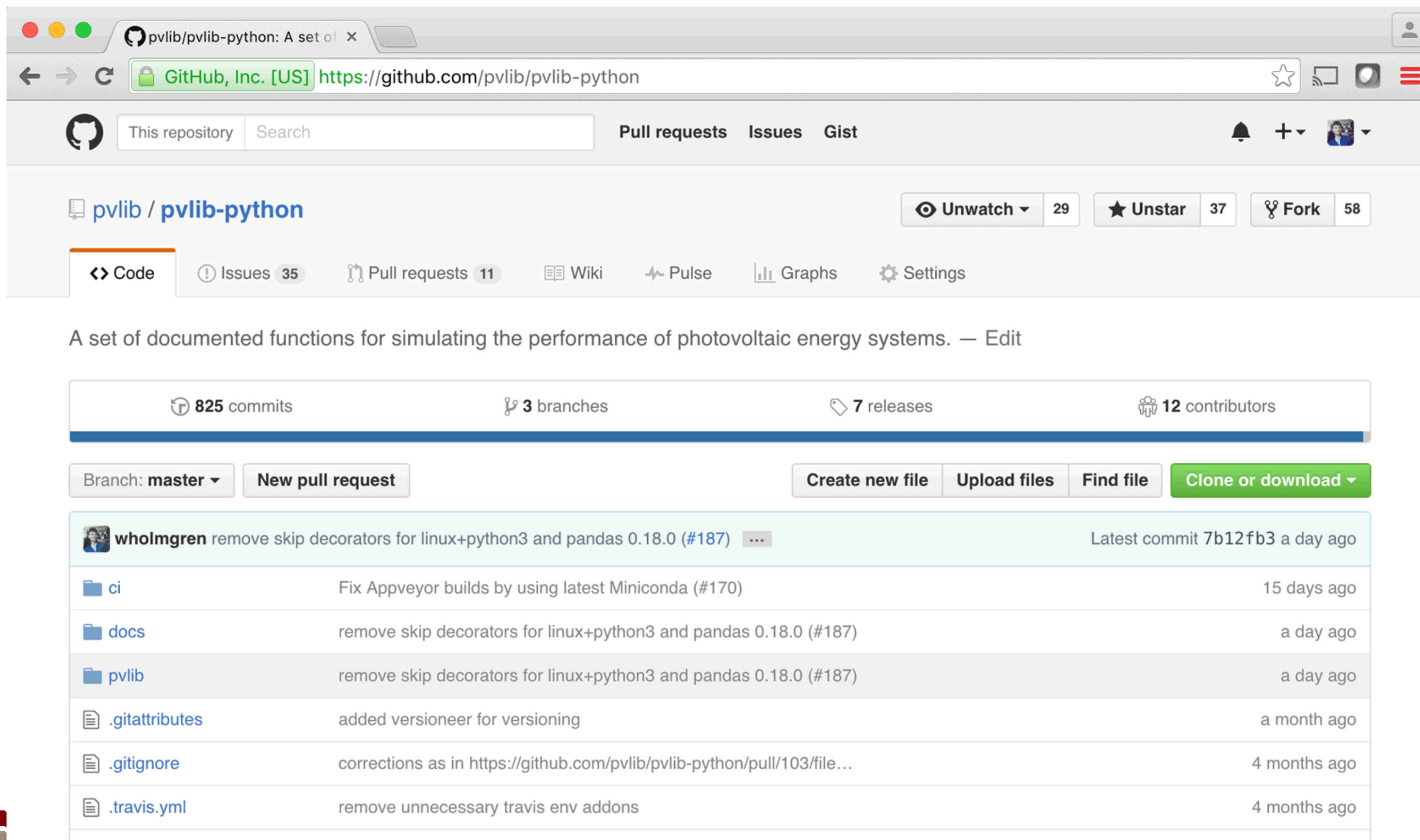
Please see the Installation page for installation help.

PVLIB on GitHub

github.com/pvlib/pvlib-python

github.com/sandialabs/MATLAB_PV_LIB

Please go here!



Challenges for PVLIB going forward

- Maintain sufficient overlap between the MATLAB and Python libraries
- Recruit new developers and maintainers
- High quality open source software takes time, therefore money
- Users must contribute their knowledge and expertise back to the community
  - Code is great, but discussion, documentation equally important!
- Industry must allow its developers spend time to improve the library.
  - Thanks First Solar, SunPower, SolarCity employees – need more!
- Funding agencies must appreciate the value, PVLIB community must communicate the value.
  - Value includes both technical merit and broader impacts
  - Difficult to quantify value

Acknowledgements

- Sandia National Labs for starting PVLIB
- DOE EERE Postdoctoral Fellowship for supporting Will
- All of the PVLIB users that have contributed code or user feedback
  - Special thanks to PVLIB Python contributors Tony Lorenzo, Anton Driesse, Bjorn Mueller, Rob Andrews
- Tucson Electric Power, Arizona Public Service, Public Service Company of New Mexico for providing U. Arizona with system data for PV modeling

Thank You

William Holmgren
holmgren@email.arizona.edu

Joshua S. Stein
jsstein@sandia.gov