PVLIB Python Design and Development

Will Holmgren

Post doc
Department of Atmospheric Sciences
University of Arizona

EPRI-Sandia PV Systems Symposium
May 10, 2016

Goals of this talk

- Introduce you to PVLIB Python
- Introduce you to modern Python package development practices
- Turn you into a PVLIB Python contributor
- Turn a few of you into PVLIB Python maintainers
- Answer your questions

Outline

What we will cover

- PVLIB Python structure, design philosophy
  - Functions
  - Classes
  - Modules
  - Package
- PVLIB Python development
  - Installation in an environment
  - git + GitHub
  - Documentation + ReadTheDocs
  - Testing + TravisCI
- How PVLIB Python is different from PVLIB Matlab

How we will cover it

- Slides with summaries, examples, and links
- Work through the entire process of proposing a new addition to PVLIB Python

What we will not cover

- Many modeling examples
- How to use Python

PVLIB Python Structure

PVLIB Python is a **package**.

All code should be organized into **modules** based on a topic.

The core code, the basic algorithms, should be contained in simple **functions**.

> Doing something simple? Something specific to a corner of the library? Stop here!

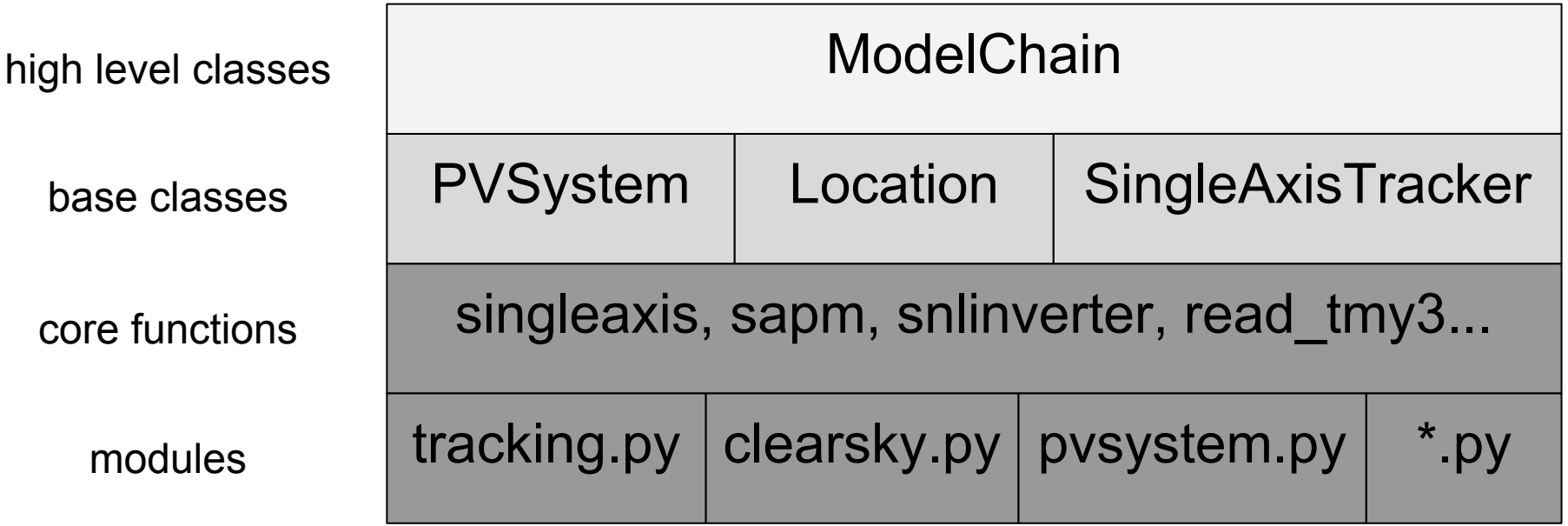
Additional functions can be written to make it easier to use basic functions.

Classes can provide additional structure, abstractions, don't-repeat-yourself.

> Like object oriented programming? Need structure? Use classes!

PVLIB Python Structure

simplified library structure



PVLIB Python modeling example

As a teaser, let’s model the annual energy yield for a system at a few locations...

http://pvlib-python.readthedocs.io/en/latest/package_overview.html#modeling-paradigms

PVLIB Python Structure: Modules

Code is organized into **modules:**

- clearsky.py
- irradiance.py
- pvsystem.py
- solarposition.py
- tmy.py
- and more…

https://github.com/pvlib/pvlib-python/tree/master/pvlib

http://pvlib-python.readthedocs.io/en/latest/modules.html

PVLIB Python Structure: Modules

modules contain **functions** and **classes**

pvsystem.py

- sapm
- snlinverter
- singlediode
- PVSystem
- ...

irradiance.py

- haydavies
- perez
- disc
- total_irrad
- ...

PVLIB Python Structure: Functions

All essential code is contained in basic **functions**.

Functions are simple. Functions are safe. Functions are great! (are → should be)

```
def snlinverter(inverter, v_dc, p_dc):
    '''
    Converts DC power and voltage to AC power using Sandia's
    Grid-Connected PV Inverter model.


def spa_python(time, latitude, longitude,
               altitude=0, pressure=101325, temperature=12, delta_t=None,
               atmos_refract=None, how='numpy', numthreads=4):
    """
    Calculate the solar position using a python implementation of the
    NREL SPA algorithm described in [1].
```

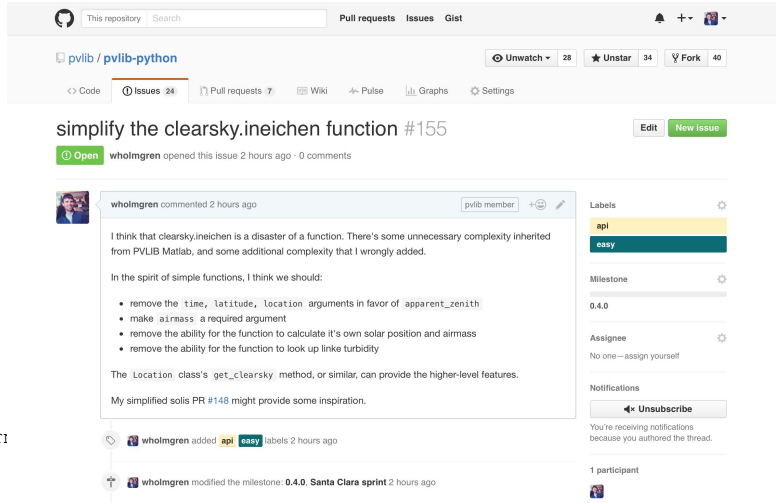
PVLIB Python Structure: Functions

```
def ineichen(time, latitude, longitude...)
    I0 = irradiance.extraradiation(time.dayofyear)

    if zenith_data is None:
        ephem_data = solarposition.get_solarposition(time,
                                                     latitude=latitude,
                                                     longitude=longitude,
                                                     altitude=altitude,
                                                     method=solarposition_method)
        time = ephem_data.index # fixes issue with time possibly not being tz-aware
        try:
            ApparentZenith = ephem_data['apparent_zenith']
        except KeyError:
            ApparentZenith = ephem_data['zenith']
            logger.warning('could not find apparent_zenith. using zenith')
    else:
        ApparentZenith = zenith_data

    if linke_turbidity is None:
        TL = lookup_linke_turbidity(time, latitude, longitude,
                                    interp_turbidity=interp_turbidity)
    else:
        TL = linke_turbidity

    # Get the absolute airmass assuming standard local pressure (per
    # alt2pres) using Kasten and Young's 1989 formula for airmass.

    if airmass_data is None:
        AMabsolute = atmosphere.absoluteairmass(airmass_relative=atmosphere.relativeairm
                                                pressure=atmosphere.alt2pres(altitude))
    else:
        AMabsolute = airmass_data
```

There's a lot going on in `ineichen` beyond simply implementing the model.

Maybe we should fix this!

https://github.com/pvlib/pvlib-python/issues/155



PVLIB Python Structure: A problem

We want to write simple, self-contained functions that are easy to understand.

We want to write functions that are easy to use.

“I want to know the airmass as a function of zenith angle.” - easy to use, easy to understand

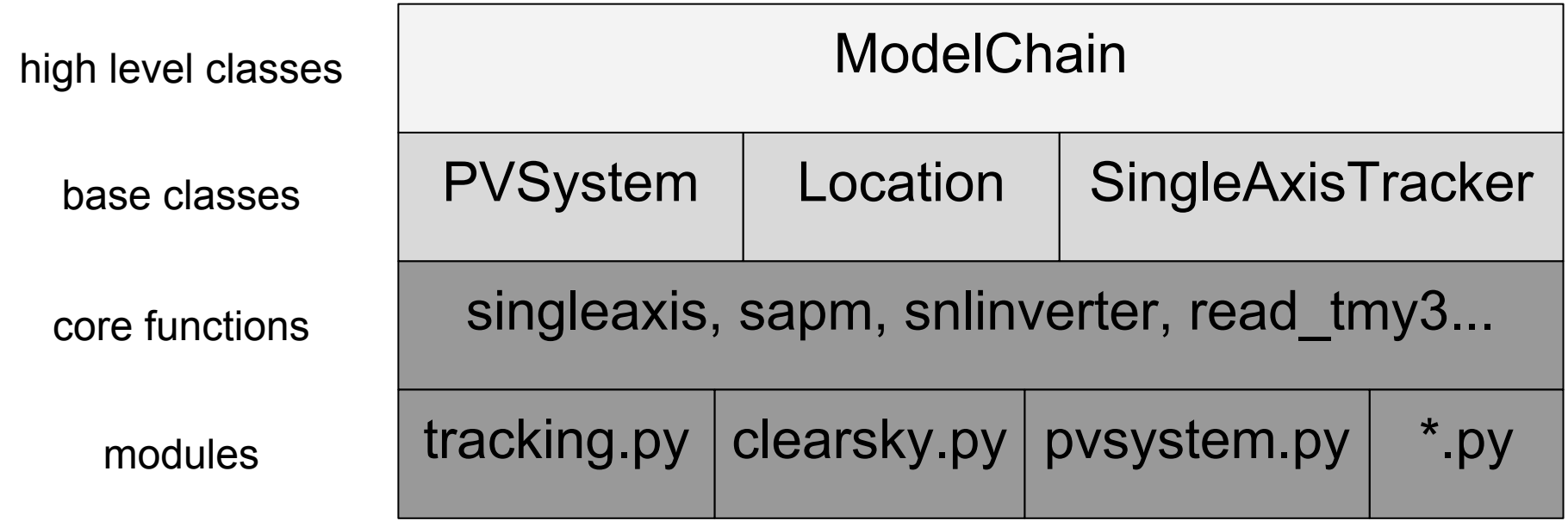
“I want to know the airmass as a function of the date and time for my point on the Earth.” - easy to use, fairly easy to understand

“I want a function that lets me do either of the above.” - harder to use, harder to understand.

“Oh, and I want to be able to choose among many different airmass models” - :(

PVLIB Python Structure

Maybe we can use classes to provide broader functionality that is still easy to understand.



PVLIB Python Structure: Classes

Classes and objects can provide convenient abstractions.

```
class PVSystem(object):
    """
    The PVSystem class defines a standard set of PV system attributes
    and modeling functions. This class describes the collection and
    interactions of PV system components rather than an installed system
    on the ground. It is typically used in combination with
    :py:class:`~pvlib.location.Location` and
    :py:class:`~pvlib.modelchain.ModelChain`
    objects.
```

```
class Location(object):
    """
    Location objects are convenient containers for latitude, longitude,
    timezone, and altitude data associated with a particular
    geographic location. You can also assign a name to a location object.
```

pvlib-python/issues/17
pvlib-python/pull/93

Replaces and expands on PVLIB MATLAB's location struct

https://github.com/Sandia-Labs/PVLIB_Python/pull/26

PVLIB Python uses classes to separate intrinsic and extrinsic data.

PVLIB Python Structure: PVSystem class

```
class PVSystem(object):
    """
    The PVSystem class defines a standard set of PV system attributes
    and modeling functions. This class describes the collection and
    interactions of PV system components rather than an installed system
    on the ground.

    def __init__(self,
                 surface_tilt=0, surface_azimuth=180,
                 albedo=None, surface_type=None,
                 module=None, module_parameters=None,
                 series_modules=None, parallel_modules=None,
                 inverter=None, inverter_parameters=None,
                 racking_model='open_rack_cell_glassback',
                 **kwargs):

    def get_irradiance(self, solar_zenith, solar_azimuth, dni, ghi, dhi,
                       dni_extra=None, airmass=None, model='haydavies',
                       **kwargs):

    def sapm(self, poa_direct, poa_diffuse,
             temp_cell, airmass_absolute, aoi, **kwargs):
```

many more methods not shown here

pvlib-python/issues/17
pvlib-python/pull/93

object constructor accepts intrinsic data.

data assigned as attributes

methods accept extrinsic data

PVLIB Python Structure: PVSystem class

```
class PVSystem(object):
```

```
    def snlinverter(self, v_dc, p_dc):
        """Uses :func:`snlinverter` to calculate AC power based on
        ``self.inverter_parameters`` and the input parameters.

        Parameters
        ----------
        See pvsystem.snlinverter for details

        Returns
        -------
        See pvsystem.snlinverter for details
        """
        return snlinverter(self.inverter_parameters, v_dc, p_dc)
```

`PVSystem.snlinverter` **method** is a wrapper around the `snlinverter` **function**.

2 variable arguments

1 constant system argument, 2 variable arguments

PVLIB Python Structure: Location class

```
class Location(object):
    """
    Location objects are convenient containers for latitude, longitude,
    timezone, and altitude data associated with a particular
    geographic location. You can also assign a name to a location object.

    def __init__(self, latitude, longitude, tz='UTC', altitude=0,
                 name=None, **kwargs):

    @classmethod
    def from_tmy(cls, tmy_metadata, tmy_data=None, **kwargs):
        Create an object based on a metadata
        dictionary from tmy2 or tmy3 data readers.

    def get_solarposition(self, times, pressure=None, temperature=12,
                          **kwargs):
        Uses the :py:func:`solarposition.get_solarposition` function
        to calculate the solar zenith, azimuth, etc. at this location.

    def get_clearsky(self, times, model='ineichen', **kwargs):
        Calculate the clear sky estimates of GHI, DNI, and/or DHI
        at this location.

    def get_airmass(self, times=None, solar_position=None,
                    model='kastenyoung1989'):
        Calculate the relative and absolute airmass.
```

Replaces and expands on PVLIB MATLAB's location struct

Construct a `Location` object from a TMY file

Automatically propagates altitude

1 variable argument, fills in the intrinsic data needed by the `ineichen` function

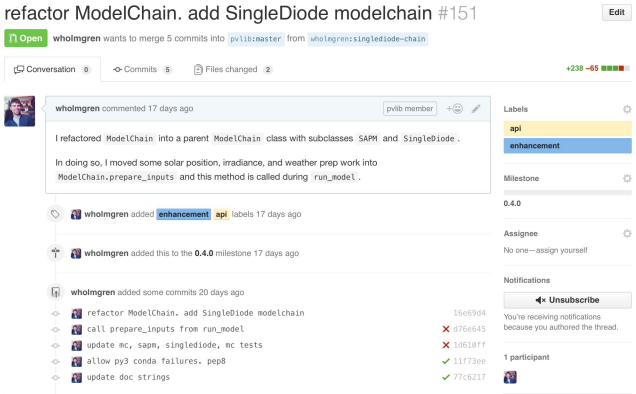
Calculates both relative and absolute airmass

PVLIB Python Structure: ModelChain class

The ModelChain is a new, experimental, high-level class

```
class ModelChain(object):
    """
    An experimental class that represents all of the modeling steps
    necessary for calculating power or energy for a PV system at a given
    location.

    def __init__(self, system, location,
                 orientation_strategy='south_at_latitude_tilt',
                 clearsky_model='ineichen',
                 transposition_model='haydavies',
                 solar_position_method='nrel_numpy',
                 airmass_model='kastenyoung1989',
                 **kwargs):

    def run_model(self, times, irradiance=None, weather=None):
```

pvlib-python/pull/151

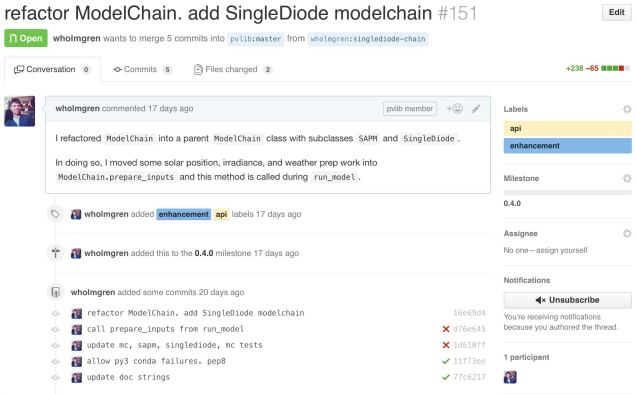


PVLIB Python Structure: ModelChain class

The ModelChain is a new, experimental, high-level class

```
class ModelChain(object):

    def run_model(self, times, irradiance=None, weather=None):

        # ...lots of stuff above here…

        self.temps = self.system.sapm_celltemp(self.total_irrad['poa_global'],
                                               self.weather['wind_speed'],
                                               self.weather['temp_air'])

        self.aoi = self.system.get_aoi(self.solar_position['apparent_zenith'],
                                       self.solar_position['azimuth'])

        self.dc = self.system.sapm(self.total_irrad['poa_direct'],
                                   self.total_irrad['poa_diffuse'],
                                   self.temps['temp_cell'],
                                   self.airmass['airmass_absolute'],
                                   self.aoi)

        self.ac = self.system.snlinverter(self.dc['v_mp'], self.dc['p_mp'])

        return self
```

[pvlib-python/pull/151]



PVLIB Python Structure: LocalizedPVSystem class

The LocalizedPVSystem is a new, experimental, high-level class that inherits all methods from PVSystem and Location.

```
class LocalizedPVSystem(PVSystem, Location):
    """
    The LocalizedPVSystem class defines a standard set of installed PV
    system attributes and modeling functions. This class combines the
    attributes and methods of the PVSystem and Location classes.

    See the :py:class:`PVSystem` class for an object model that
    describes an unlocalized PV system.
    """
    def __init__(self, pvsystem=None, location=None, **kwargs):
```

Useful? More trouble than it's worth?

PVLIB Python Installation: Python

Many ways to get Python, many ways to get PVLIB Python.

Easiest way is the Anaconda Python distribution + **conda** package manager.

Anaconda comes with a bunch of numerical and scientific libraries preinstalled.

www.continuum.io

PVLIB Python Installation Guide: http://pvlib-python.readthedocs.io/en/latest/installation.html

PVLIB Python Installation: Python

How you install PVLIB Python depends on how you want to use PVLIB Python.

Do you want to use the pvlib-python as-is, or do you want to be able to edit the source code?

If you want to use pvlib-python *as-is*, follow the simple Install standard release instructions.

If you want to be able to *edit the source code*, follow the Install as an editable library instructions.

Installing pvlib-python is similar to installing most scientific python packages, so see the References section for further help.

PVLIB Python Installation Guide: http://pvlib-python.readthedocs.io/en/latest/installation.html

PVLIB Python Installation: PVLIB Python

“I don’t care about modifying the source code, I only want to use PVLIB Python to solve my problem.”

With the Anaconda Python distribution...

```
conda install pvlib -c pvlib
```

With any other Python distribution...

```
pip install pvlib
```

Don’t use sudo!!!

PVLIB Python Installation Guide: http://pvlib-python.readthedocs.io/en/latest/installation.html

PVLIB Python Installation: PVLIB Python

“I want to play with the source code and maybe even contribute to the library.”

Installing pvlib-python as an editable library involves 3 steps:

1. Obtain the source code → GitHub
2. Set up a virtual environment → conda
3. Install the source code → pip

PVLIB Python Installation Guide: http://pvlib-python.readthedocs.io/en/latest/installation.html

3 steps to git proficiency

1. Read the git documentation!

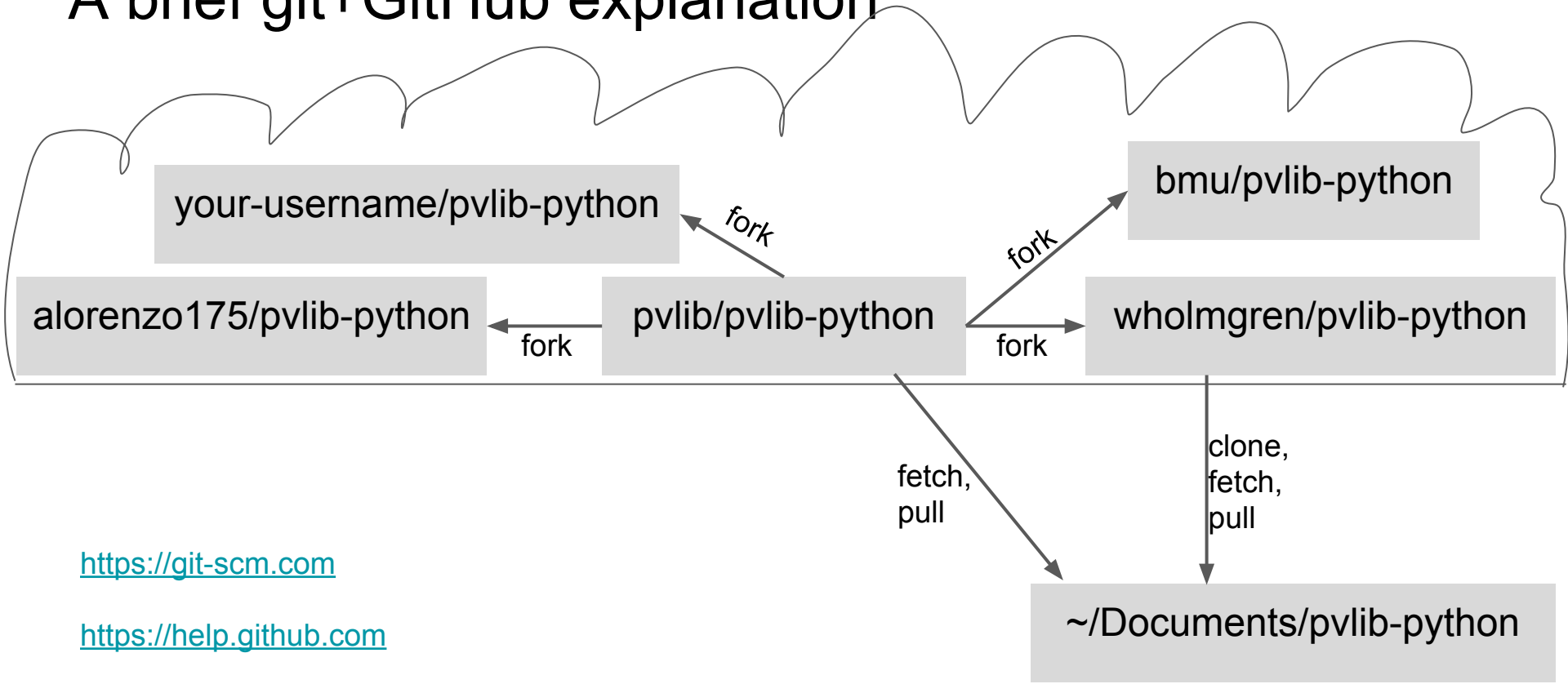
https://git-scm.com and https://help.github.com

2. Struggle with it!

https://git-scm.com and https://help.github.com

3. Read it again!

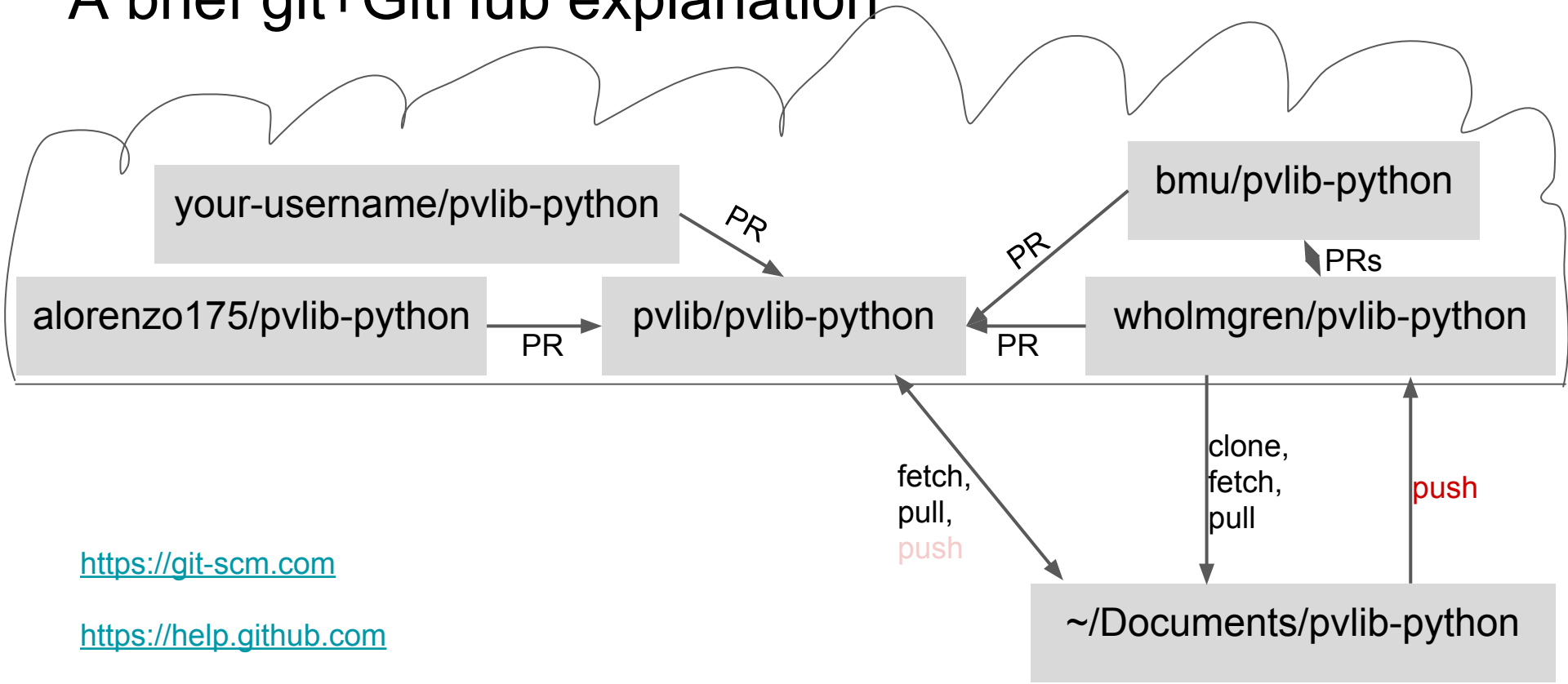
A brief git+GitHub explanation



https://git-scm.com

https://help.github.com

A brief git+GitHub explanation



https://git-scm.com

https://help.github.com

Getting PVLIB Python with git and GitHub

Instructions on how to use GitHub Desktop from our installation guide...

> Follow these steps to obtain the library using git/GitHub:
>
> 1. **Download** the GitHub Desktop application.
> 2. **Fork** the pvlib-python project by clicking on the "Fork" button on the upper right corner of the pvlib-python GitHub page.
> 3. **Clone** your fork to your computer using the GitHub Desktop application by clicking on the *Clone to Desktop* button on your fork's homepage. This button is circled in the image below. Remember the system path that you clone the library to.

A lot of people like GitHub Desktop. I prefer to use the command line.

PVLIB Python Installation Guide: http://pvlib-python.readthedocs.io/en/latest/installation.html

Getting PVLIB Python with git and GitHub

GitHub Desktop Preferences panel has an “Install Command Line Tools” button.

After forking the main pvlib-python GitHub repository...

```
# with https...
$ git clone https://github.com/wholmgren/pvlib-python.git

# with ssh...
$ git clone git@github.com:wholmgren/pvlib-python.git
```

I use an ssh key with GitHub. No need to enter passwords!

PVLIB Python Installation Guide: http://pvlib-python.readthedocs.io/en/latest/installation.html

Add a git remote for the main repo

Teaches your git checkout about another source of data.

```
$ git remote add pvlib git@github.com:pvlib/pvlib-python.
git

$ git remote -v
origin   git@github.com:wholmgren/pvlib-python.git (fetch)
origin   git@github.com:wholmgren/pvlib-python.git (push)
pvlib    git@github.com:pvlib/pvlib-python.git (fetch)
pvlib    git@github.com:pvlib/pvlib-python.git (push)

# now you can do one or all of
$ git fetch pvlib
$ git rebase pvlib/master
$ git merge pvlib/master
$ git pull pvlib master
```

Freedom through git verbs!

PVLIB Python Installation Guide: http://pvlib-python.readthedocs.io/en/latest/installation.html

Create a conda environment for pvlib

Environments make it possible to use multiple versions and package configurations on top of one Python installation.

Useful for users, essential for developers.

```
$ conda create --name pvlibdev python=3.5 pandas scipy

$ source activate pvlibdev

$ conda list

# optional
$ conda install jupyter ipython matplotlib seaborn nose flake8
```

PVLIB Python Installation Guide: http://pvlib-python.readthedocs.io/en/latest/installation.html

Install the source code

Installing in “development mode” creates an alias/symlink/shortcut from your local folder to your environment’s Python package listing.

```
$ pip install -e pvlib-python

# alternatively
$ cd pvlib-python
$ python setup.py develop
```

Google “python site packages” and “pip development install” for more.

PVLIB Python Installation Guide: http://pvlib-python.readthedocs.io/en/latest/installation.html

Contributing: Fix a problem or add a new feature

1. Find or make a new issue on GitHub.
2. Make a new git branch. `$ git branch changes; $ git checkout changes`
3. Make changes.
4. Test changes. `$ nose`
5. Make and test documentation `$ cd docs/sphinx; make html`
6. Commit changes. `$ git commit -a -m 'my changes'`
7. Push changes. `$ git push`
8. Make a Pull Request.

We're going to walk through this by porting pvl_erbs.m to Python.

PVLIB Python Contributing Guide: http://pvlib-python.readthedocs.io/en/latest/contributing.html

Contributing: Make/find an issue on GitHub

Admitting there is a problem is the first step!

Consider using the milestones and labels.

Issues labeled easy are good for beginners.

Making a new issue is not required, but is often a good idea.

Please don't be afraid to make an issue!

Please comment on existing issues, too.

PVLIB Python Contributing Guide: http://pvlib-python.readthedocs.io/en/latest/contributing.html

Contributing: Make a new git branch

```
$ git branch changes

$ git checkout changes

# or in one step...

$ git checkout -b changes
```

Branches are not required, but usually a good idea.

Branches use almost no disk space (very different from CVS/SVN)

PVLIB Python Contributing Guide: http://pvlib-python.readthedocs.io/en/latest/contributing.html

Contributing: Make the changes

Porting pvl_erbs.m to PVLIB Python...

PVLIB Python Contributing Guide: http://pvlib-python.readthedocs.io/en/latest/contributing.html

Contributing: Test the changes

PVLIB Python has a lot of tests. Not enough, but a lot.

We will not accept code that is not rigorously tested.

90% of the code is executed by the test suite.

Most of the remaining 10% is difficult to automatically test (e.g. dialog boxes, versioning).

Two kinds of tests:

1. **Does it crash?** Usually easy to test.
2. **Does it give the right result?** Usually annoying, but important.

PVLIB Python Contributing Guide: http://pvlib-python.readthedocs.io/en/latest/contributing.html

Contributing: Test the changes

from test_tracking.py

Simple case of a single axis tracker oriented North-South.

```
def test_solar_noon():
    apparent_zenith = pd.Series([10])
    apparent_azimuth = pd.Series([180])
    tracker_data = tracking.singleaxis(apparent_zenith, apparent_azimuth,
                                       axis_tilt=0, axis_azimuth=0,
                                       max_angle=90, backtrack=True,
                                       gcr=2.0/7.0)

    expect = pd.DataFrame({'aoi': 10, 'surface_azimuth': 90,
                           'surface_tilt': 0, 'tracker_theta': 0},
                           index=[0], dtype=np.float64)

    assert_frame_equal(expect, tracker_data)
```

PVLIB Python Contributing Guide: http://pvlib-python.readthedocs.io/en/latest/contributing.html

Contributing: Test the changes

Run the tests locally using the Nose package

```
# execute all modules and functions that contain *test*
$ nose

# only run some tests, print test names
$ nose pvlib/test/test_clearsky -v
```

Later, we'll see how TravisCI automatically runs the tests for all Pull Requests.

PVLIB Python Contributing Guide: http://pvlib-python.readthedocs.io/en/latest/contributing.html

Contributing: Make and test documentation

Only needed if you've made extensive changes to the documentation.

```
$ conda install sphinx sphinx_rtd_theme
$ cd docs/sphinx
$ make html
```

Documentation will be created in `docs/sphinx/build/html`

Improving the documentation is a great way to contribute!

Later, we'll see how we can use readthedocs to build and host the documentation.

PVLIB Python Contributing Guide: http://pvlib-python.readthedocs.io/en/latest/contributing.html

Contributing: Commit the changes

Tell git what what you've done

```
$ git status

$ git add file1.py file2.py

$ git commit -m 'my changes'

$ git log

# print stackoverflow's nice git tree
$ git lg
```

PVLIB Python Contributing Guide: http://pvlib-python.readthedocs.io/en/latest/contributing.html

Contributing: Push changes to GitHub

Push your changes to your GitHub repository

```
$ git push

# will not work if you're working on a new branch
# copy/paste the command that it prints, e.g.

$ git push --set-upstream changes
```

A copy of the new code now lives on the GitHub servers.

PVLIB Python Contributing Guide: http://pvlib-python.readthedocs.io/en/latest/contributing.html

Contributing: make a Pull Request

Pull requests are a mechanism for proposing changes to a repository.

You can make a Pull Request even if your code isn't yet complete, like asking for comments on a draft.

Maybe you think you're a bad programmer that writes crappy code and you don't want be embarrassed.

I *know* that I'm a bad programmer that writes crappy code and I've been worried about being embarrassed too!

Propose the code, and we'll make it better together.

PVLIB Python Contributing Guide: http://pvlib-python.readthedocs.io/en/latest/contributing.html

Contributing: readthedocs

The readthedocs service will automatically build your documentation if you authorize it.

Example...

PVLIB Python Contributing Guide: http://pvlib-python.readthedocs.io/en/latest/contributing.html

Contributing: TravisCI

The TravisCI service will automatically test your Pull Request.

Tests against multiple versions of Python and the minimum versions of the required dependencies.

It will also test your branches if you authorize it to.

Appveyor is a similar service, but it runs on Windows instead of Linux.

Example…

PVLIB Python Contributing Guide: http://pvlib-python.readthedocs.io/en/latest/contributing.html

Concluding thoughts/questions

- What's missing from PVLIB Python?
- What's in PVLIB Python that shouldn't be?
- Is PVLIB Python trying to be too many things to too many people?
- Should PVLIB Python continue to depend heavily on Pandas?
- Can you help make PVLIB Python better?
- Is there anything preventing you from contributing to PVLIB Python?

This is your library, not mine. If you don't like something, change it and share it!

PV modeling is far too broad for one person to write a great library.

Open source enables reproducible science and engineering.

Thanks to

The Sandia PVLIB team

The PVLIB community

WH thanks the DOE EERE Postdoctoral Fellowship Program

U Arizona team thanks Tucson Electric Power, Arizona Public Service, the SVERI utilities, and the UA Renewable Energy Network https://sveri.uaren.org/

EPRI and Southern Company Services for funding the forecasting tools.

backup, unused, discarded slides...

PVLIB Python Structure: Functions

Some functions choose among several other functions

```
def total_irrad(surface_tilt, surface_azimuth,
                apparent_zenith, azimuth,
                dni, ghi, dhi, dni_extra=None, airmass=None,
                albedo=.25, surface_type=None,
                model='isotropic',
                model_perez='allsitescomposite1990', **kwargs):
    '''
    Determine diffuse irradiance from the sky on a
    tilted surface.


def get_solarposition(time, latitude, longitude,
                      altitude=None, pressure=None,
                      method='nrel_numpy',
                      temperature=12, **kwargs):
    """
    A convenience wrapper for the solar position calculators.
```