An Open Source Solar Power Forecasting Tool Using PVLIB Python

William F. Holmgren[1], Derek G. Groenendyk[2]

[1]Dept. of Atmospheric Sciences, Univ. of Arizona, Tucson, AZ; [2]Dept. of Hydrology, Univ. of Arizona, Tucson, AZ

Introduction to PVLIB

The PVLIB Toolbox is an open source MATLAB and Python library for photovoltaic modeling and analysis. PVLIB was originally developed at Sandia National Laboratories and has been expanded by contributions from members of the Photovoltaic Performance and Modeling Collaboration (PVPMC). The PVLIB source code is hosted on GitHub. PVLIB Python and PVLIB MATLAB are BSD 3-clause licensed and free for commercial use. We encourage all users to contribute to improving the library.

Adding forecasts to PVLIB Python

Solar power forecast methods continue to be developed at a rapid pace. We propose that both public and private solar power forecasters will benefit from standardized, open source, reference implementations of forecast methods that use publicly available data.

PVLIB and Python are natural choices for developing an open source tool that combines weather forecasts and PV models. Python is easy to read and write, portable across platforms, free and open source, and it has a large scientific computing community. Python has also been identified by Unidata as a key technology for geosciences.

We chose to use Unidata's Siphon library to easily and programmatically download geosciences data in Python. The Siphon library provides access to a Unidata THREDDS server that hosts forecasts from the Global Forecast System (GFS), North American Model (NAM), High Resolution Rapid Refresh (HRRR), Rapid Refresh (RAP), and National Digital Forecast Database (NDFD). Siphon and THREDDS simplify the process of obtaining a time series forecast for a point or subdomain of a forecast model.

Forecast module structure

We model forecasts using a base `ForecastModel` class and a series of subclasses corresponding to each of the supported weather models. `ForecastModel` defines the data retrieval and basic processing methods. Each subclass may redefine its own combination of the processing steps. The result is a consistent API for all weather models that makes analyzing the data easier and less error-prone. Users can easily create new classes and modify the existing classes.

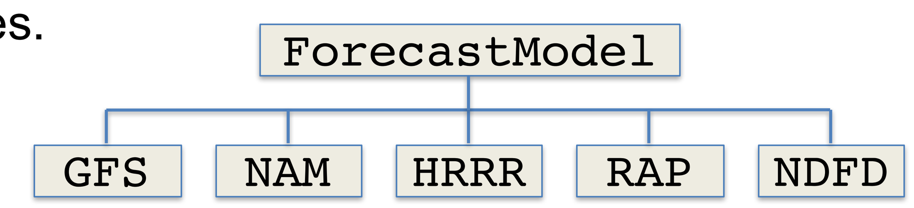

Fig. 1: Forecast model class inheritance diagram.

```
# simplified code
class ForecastModel():
    def get_data()
    def process_data()
    def get_processed_data()
    def uv_to_speed()
    def cloud_cover_to_irradiance()

class GFS(ForecastModel):
    def init():
        self.variables = {# GFS-specific name map}
    def process_data(data, cloud_cover='total_clouds', **kwargs):
        data = super(GFS, self).process_data(data)
        data['temperature'] = self.kelvin_to_celsius(data)
        data['wind_speed'] = self.uv_to_speed(data)
        data = self.cloud_cover_to_irradiance(data)
```

Accessing model data

Forecast data can be accessed using the `get_data` method of a forecast model object.

```
lat, lon, tz = 45.5, -122.7, 'Etc/GMT+8'  # Portland, OR
start = pd.Timestamp.now(tz=tz)
end = start + pd.Timedelta(days=7)
model = GFS()
raw_data = model.get_data(lat, lon, start, end)
```

	Downward_ Short-Wave_ Radia tion_ Flux_ surface_ Mixed_ intervals_ Average	Tempera ture_ surface	Total_ cloud_ cover_ boundary_ layer_ cloud_ Mixed_ intervals_ Average	Total_ cloud_ cover_ convective_ cloud	Total_ cloud_ cover_ entire_ a tmosphere_ Mixed_ intervals_ Average	Total_ cloud_ cover_ high_ cloud_ Mixed_ intervals_ Average	Total_ cloud_ cover_ low_ cloud_ Mixed_ intervals_ Average	Total_ cloud_ cover_ middle_ cloud_ Mixed_ intervals_ Average	Wind_ speed_ gust_ surface	u-comp onent_ of_ wind_ isobaric	v-comp onent_ of_ wind_ isobaric
2016-06-02 00:00:00-08:00	758.0	300.5	0.0	0.0	95.0	95.0	0.0	0.0	0.0	0.0	0.0
2016-06-02 03:00:00-08:00	280.0	293.0	0.0	0.0	94.0	94.0	0.0	0.0	0.0	0.0	0.0
2016-06-02 06:00:00-08:00	144.0	284.7	0.0	0.0	93.0	93.0	0.0	0.0	0.0	0.0	0.0
2016-06-02 09:00:00-08:00	0.0	286.6	0.0	0.0	98.0	0.0	98.0	0.0	0.0	0.0	0.0
2016-06-02 12:00:00-08:00	0.0	286.0	16.0	0.0	99.0	30.0	99.0	15.0	0.0	0.0	0.0

The raw data can be processed into a standard format using the model's `process_data` method.

```
processed_data = model.process_data(raw_data)
```

In the GFS example shown here, `process_data` converts temperature from Kelvin to Celsius, calculates radiation components from total cloud cover using the Liu Jordan model, and calculates wind speed from the u and v wind components.

	temperature	wind_speed	ghi	dni	dhi	total_clouds	low_clouds	mid_clouds	high_clouds
2016-06-02 00:00:00-08:00	27.4	2.7	0.0	0.0	0.0	95.0	0.0	0.0	95.0
2016-06-02 03:00:00-08:00	19.9	4.0	0.0	0.0	0.0	94.0	0.0	0.0	94.0
2016-06-02 06:00:00-08:00	11.6	1.6	101.1	0.0	101.1	93.0	0.0	1.0	93.0
2016-06-02 09:00:00-08:00	13.5	3.2	293.6	3.8	290.9	98.0	98.0	3.0	5.0
2016-06-02 12:00:00-08:00	12.9	3.4	381.1	6.7	375.0	99.0	99.0	15.0	30.0
2016-06-02 15:00:00-08:00	13.2	2.7	306.4	0.0	306.4	100.0	100.0	96.0	59.0
2016-06-02 18:00:00-08:00	14.0	2.8	121.9	0.0	121.9	100.0	100.0	79.0	55.0
2016-06-02 21:00:00-08:00	16.5	3.1	0.0	0.0	0.0	99.0	98.0	0.0	68.0

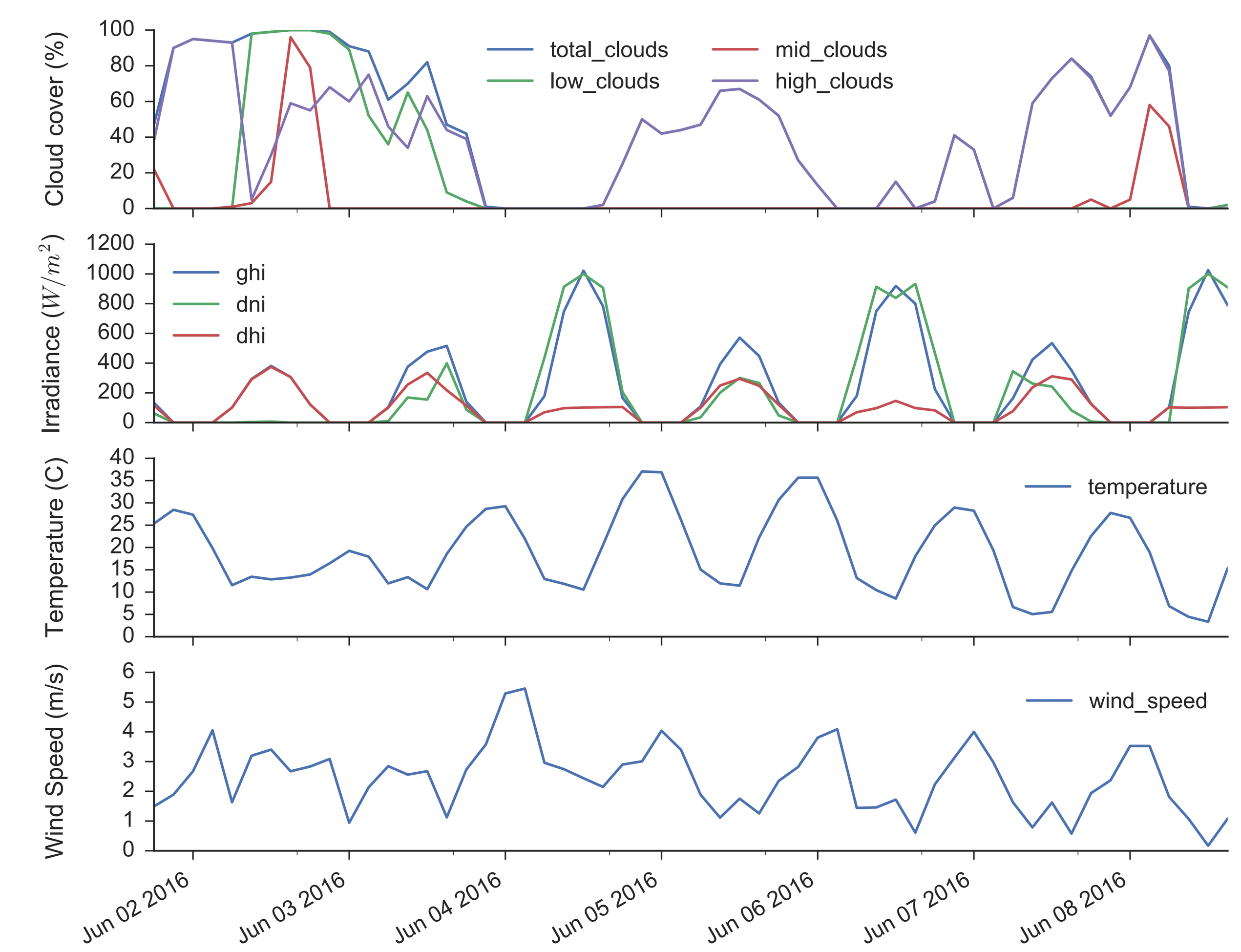

Fig. 2: Standardized PVLIB Python weather data for Portland, OR from the 2016-06-01 12Z GFS model run.

PV power forecasts

PVLIB Python provides standardized, yet extensible, classes for PV system modeling. Users can represent a system with a `PVSystem` or a `SingleAxisTracker` object, a simulation using a `ModelChain` object, and drive the simulation using downloaded and processed forecast data.

```
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, tz = 45.5, -122.7, 'Etc/GMT+8'  # Portland, OR
start = pd.Timestamp(datetime.date.today(), tz=tz)
end = start + pd.Timedelta(days=7) # 7 days from today
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=irradiance, weather=weather)
     mc.ac.plot()
```

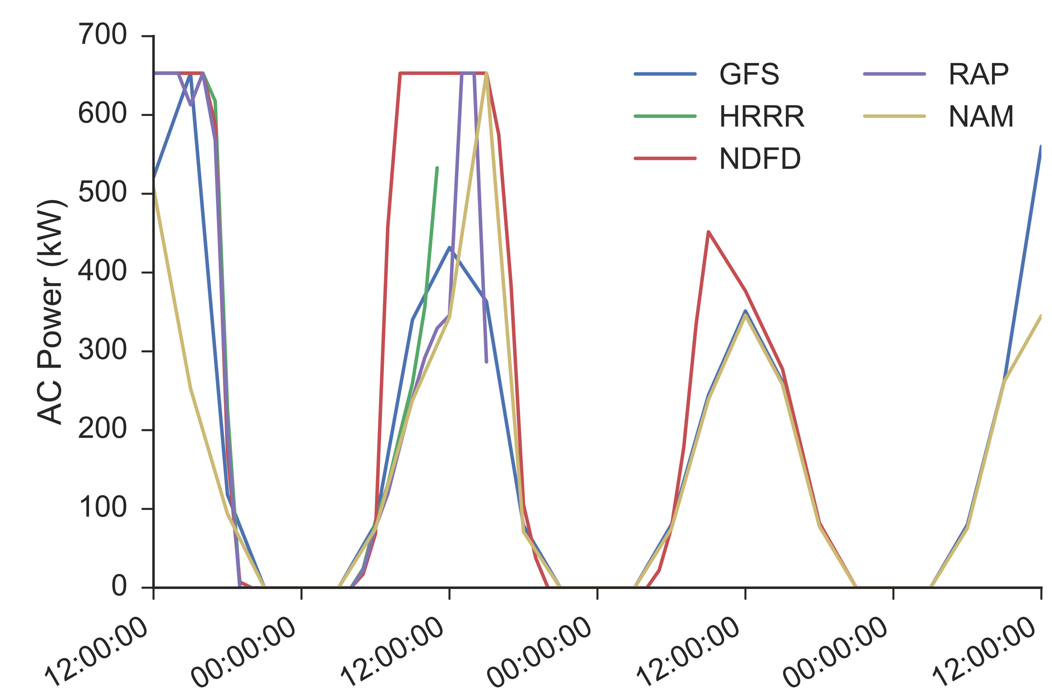

Fig. 3: PVLIB Python forecasts of AC power for a hypothetical single axis tracker in Portland, OR for 12:00 May 31 – 12:00 June 3, 2016. The power forecasts are derived from 5 different weather forecasts.

Acknowledgements

The authors gratefully acknowledge Sandia National Laboratories for the initial development of PVLIB MATLAB and PVLIB Python and the ongoing contributions of many others to the project. A list of PVLIB Python contributors may be found on the GitHub repository and in the online documentation. The authors also thank the developers and maintainers of the NOAA/NCEP/NWS forecasts. WFH thanks the Department of Energy (DOE) Office of Energy Efficiency and Renewable Energy (EERE) Postdoctoral Research Award for support.