An Open Source Solar Power Forecasting Tool Using PVLIB Python

William F. Holmgren¹, Derek G. Groenendyk²
¹Dept. of Atmospheric Sciences, Univ. of Arizona, Tucson, AZ; ²Dept. of Hydrology, Univ. of Arizona, Tucson, AZ

Addition 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 forecast users 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.

```
# simplified code
class ForecastModel:
    def __init__(self):
        self.variables = [# GFS-specific name map
                      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):
        self.variables = [# GFS-specific name map

def process_data(data):
    cloud_cover = data['cloud_cover']
    total_clouds = data['total_clouds']
    cloud_cover = data['cloud_cover']
    total_clouds = data['total_clouds']
    processed_data = process_data(data)
    processed_data = process_data(data)
    processed_data = process_data(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, 'UTC/UTM8' # 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)

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.

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 Model1Chain object, and drive the simulation using downloaded and processed forecast data.

```
module = sandia_modules['Canadian_Solar_CS5P_220M__2009__']
inverter = ocv_inverters['SHM_AMERICA_SC630CP_US_315V_CEC_2012__']
system = SingleAxisTracker(module=module,
                          inverter_parameters=inverter, series_modules=15, parallel_modules=30)
lat, lon, tz = 45.5, -122.7, 'UTC/UTM8' # 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()
```

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.

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