

# An Open Source Solar Power Forecasting Tool Using PVLIB Python

William F. Holmgren<sup>1</sup>, Antonio T. Lorenzo<sup>2</sup>, and Derek G. Groenendyk<sup>1</sup>

<sup>1</sup>Dept. of Hydrology and Atmospheric Sciences, <sup>2</sup>College of Optical Sciences, 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 (PVMC). 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 users to contribute the library at [github.com/pvlib](https://github.com/pvlib) and ask questions on [stackoverflow.com](https://stackoverflow.com) using the `pvlib` tag.

## Adding forecasts to PVLIB Python

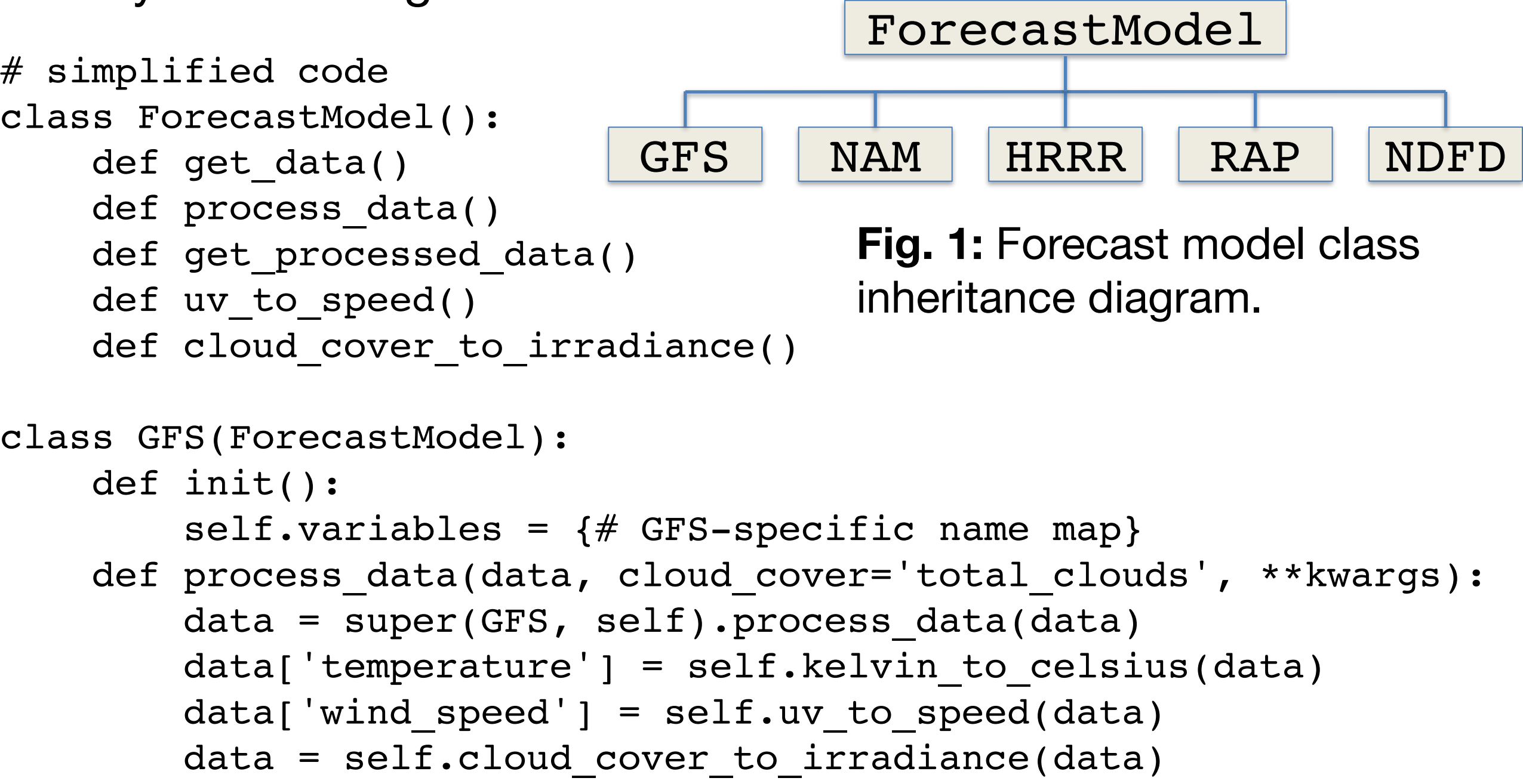
Solar power forecast methods continue to be developed at a rapid pace. We propose that both public and private solar power forecasters will benefit from standardized, open source, reference implementations of forecast methods that use publicly available data.

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 forecast 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. A script is also available for downloading the relevant point data from NOMADS, creating a time series, and exporting it to csv (see [github.com/wholmgren/get\\_nomads](https://github.com/wholmgren/get_nomads)).

## 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 data easier and less error-prone. Users can easily create new classes and modify the existing classes.



## Accessing and processing weather 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, Radiation, Flux, surface, Mixed, intervals, Average	Temperature, surface	Total, cloud, cover, boundary, layer, cloud, Mixed, intervals, Average	Total, cloud, cover, convective, cloud	Total, cloud, cover, entire, a, tmosphere, cloud, 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	756.0	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

The raw data can be processed into a standard format (see right) 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 irradiance components from total cloud cover, and calculates wind speed from the u and v wind components. The `process_data` method is slightly different for each weather model.

The most critical component of the process is converting cloud cover into irradiance. By default, the following equation from Larson, Nonnenmacher, and Coimbra (2015) is used:

$$ghi = (offset + (1 - offset) * (1 - cloud\_cover)) * ghi\_clear$$

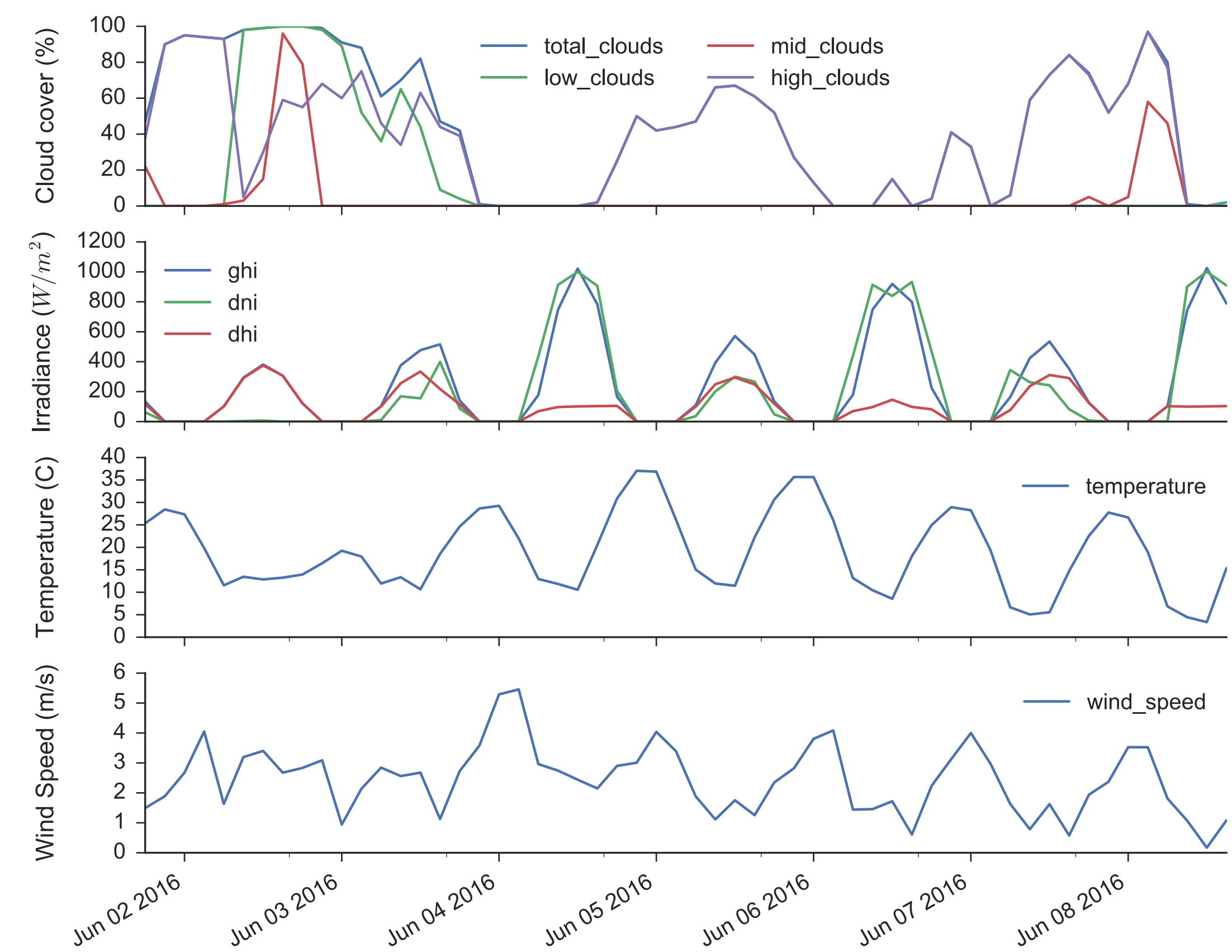
where `offset=0.35`, `cloud_cover` is the total cloud cover, and `ghi_clear` is determined by PVLIB’s climatological clear sky model. Forecasters may experiment with different parameters. The DISC model is then used to calculate DNI and DHI.

## PV power forecasts from multiple forecast models

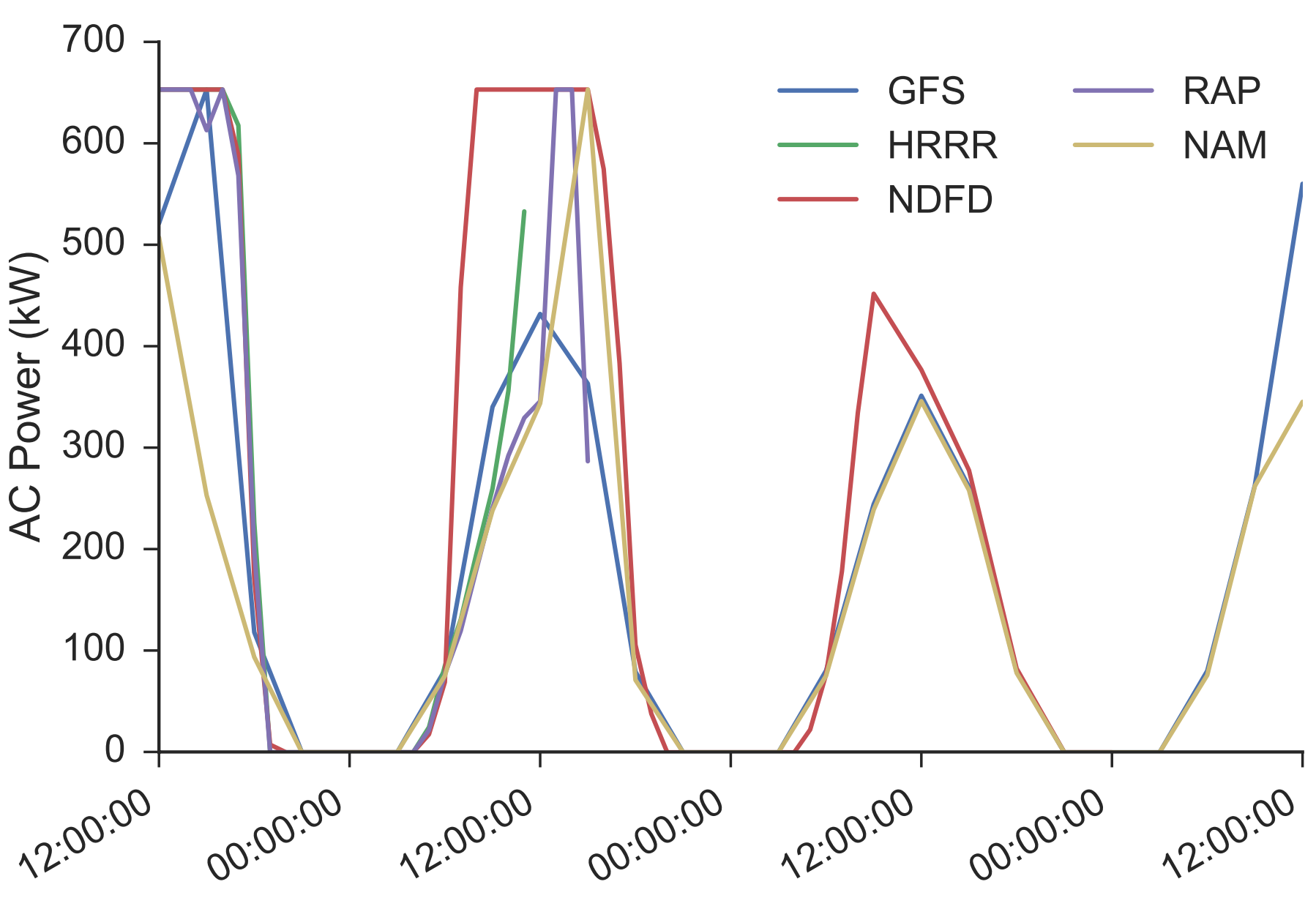
PVLIB Python provides standardized, yet extensible, classes for PV system modeling. Users can represent a PV system with a `PVSystem` or a `SingleAxisTracker` object, a simulation using a `ModelChain` object, and drive the simulation using downloaded and processed forecast data. The example below uses the Sandia Array Performance Model to forecast the power generation of one inverter of a utility-scale power plant. A PV forecast is created for each of the weather models supported in PVLIB.

```
module = sandia_modules['Canadian_Solar_CS5P_220M_2009_']
inverter = cec_inverters['SMA_America_SC630CP_US_315V_CEC_2012_']
system = SingleAxisTracker(module_parameters=module,
    module_parameters=module, inverter_parameters=inverter,
    modules_per_string=15, strings_per_inverter=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)
    mc = ModelChain(system, fx_model.location)
    mc.run_model(fx_data.index, weather=fx_data)
    mc.ac.plot()
```

	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. PVLIB was used to download forecast data from the Unidata THREDDs server, rename fields, calculate wind speed, and derive irradiance from cloud cover.



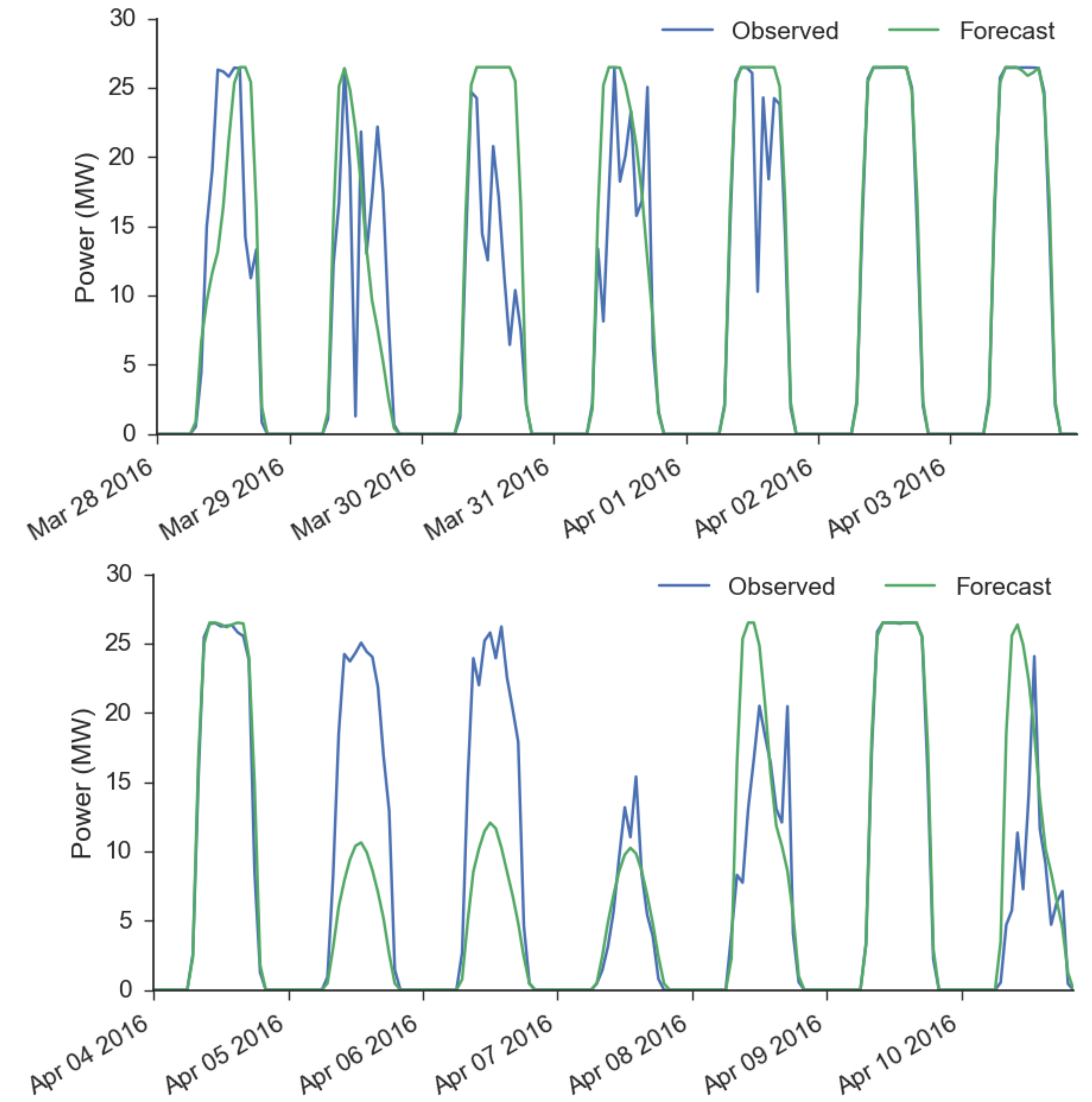
**Fig. 3:** PVLIB Python forecasts of AC power for a hypothetical single axis tracker in Portland, OR for 12:00 May 31 – 12:00 June 3, 2016. The power forecasts are derived from 5 different weather forecasts.

## Forecast verification

We compared PVLIB Python forecasts to observations for 3 months of data for a 25 MW single axis tracker near Tucson, AZ. We chose to use a simple PV model (NREL’s PVWATTS) in which the only required parameters are DC nameplate capacity, temperature coefficient, and maximum AC capacity. We resampled 3 hour data from the GFS model to 5 minutes, applied the PVLIB GFS processing functions, the PVLIB PV modeling tools, and finally compared 1 hour average forecasts and data.

```
location = Location(latitude=32.2, longitude=-110.9, altitude=700)
system = SingleAxisTracker(
    module_parameters={'pdc0': 32.5, 'gamma_pdc': -0.0035})
system.peak_ac_power = 26.5
mc = ModelChain(system, location, orientation_strategy=None,
    dc_model='pvwatts', ac_model='pvwatts',
    aoi_model='physical', spectral_model='no_loss',
    temp_model='sapm', losses_model='no_loss')
```

```
fx_model = GFS()
for fx_file in nomads_files:
    fx_data = pd.read_csv(fx_file)
    fx_data = fx_data.resample('5min').interpolate()
    fx_data = fx_model.process_data(fx_data)
    mc.run_model(fx_data.index, weather=fx_data)
    ac = mc.ac.clip_upper(system.peak_ac_power)
    ac = ac.resample('1h', label='right').mean()
```



**Fig. 4:** Observations and forecasts for a single axis tracker in Tucson, AZ.

	Hours 03-24	Hours 24-48	Hours 48-72
<b>MBE (MW)</b>	-0.4	-0.1	-0.1
<b>MAE (MW)</b>	2.5	2.4	3.0

The mean bias error and mean absolute error of the GFS+PVLIB forecasts for daylight hours are shown above. The forecasts based only on total cloud cover, temperature, and wind speed demonstrate periods of success and failure. Still, the simplicity and standardization of the forecast generation may make the tool valuable for benchmarking more sophisticated forecast models.

## Acknowledgements

We gratefully acknowledge the Electric Power Research Institute, Tucson Electric Power, and Arizona Public Service for funding. WFH thanks the Department of Energy (DOE) Office of Energy Efficiency and Renewable Energy (EERE) Postdoctoral Research Award for support. We thank Sandia National Laboratories for the initial development of PVLIB and the ongoing contributions of many others to the project. A list of PVLIB Python contributors may be found online. We also thank the developers and maintainers of the NOAA forecasts, the Unidata THREDDs service, and the Unidata Siphon project.