Python Example

November 19, 2020

Global normal spectral irradiance in Albuquerque: a one-year open dataset for PV research

The Photovoltaic Systems Evaluation Laboratory (PSEL) at Sandia National Laboratories (SNL) in Albuquerque, NM has measured global normal spectral irradiance nearly continuously from August 2013 to April 2018. During this time other broadband irradiance measurements (global horizontal, direct normal, diffuse horizontal and global normal) and weather variables were also recorded. For this dataset PV Performance Labs (PVPL) has pulled together data from both sources to assemble a full calendar year spectral data set for use in photovoltaic research. It is composed of eight continuous segments of different durations taken from the two-year period September 2013 to August 2015.

The data set and creation process are described in further detail in the following report:

- A. Driesse and J. S. Stein "Global normal spectral irradiance in Albuquerque: a one-year open dataset for PV research," *Sandia Report No. SAND2020-12693, 2020.*

Please cite the above report if you use the data.

This Jupyter notebook demonstrates how to read and use the hdf5 data files using python. We hope that this dataset and the examples inspire you, and invite you to reach out to us with feedback and suggestions.

Notebook author: Anton Driesse anton.driesse@pvperformancelabs.com

Last modified: 2020-11-19

[1]:
```
import h5py
import numpy as np
import pandas as pd

import matplotlib
import matplotlib.pyplot as plt

# some preferences you will may want to change
matplotlib.style.use('classic')
matplotlib.rcParams['figure.facecolor'] = 'w'
matplotlib.rcParams['axes.grid'] = True

# %matplotlib notebook
```

[2]:
```
# make sure the three data file locations are correctly identified

WEATHER_FILE = 'data/weather.h5'
SPECTRA_FILE = 'data/spectra.h5'
AM15_FILE = 'data/am15.h5'

# and have a look what's in the first one

f = h5py.File(WEATHER_FILE, 'r')
f.visititems(print)
f.close()
```

```
cdef <HDF5 dataset "cdef": shape (14,), type "|V268">
data <HDF5 dataset "data": shape (54221,), type "|V76">
meta <HDF5 dataset "meta": shape (), type "|O">
```

[3]:
```
# there are three items: column definitions, the actual data and some general␣
↪meta data
```

[4]:
```
# the hdf5 file contents can be accessed like a dictionary
# '[()]' means: get everything

with h5py.File(WEATHER_FILE, 'r') as f:
    weather_meta_string = f['/meta'][()]

print('============================ Start of META ============================')
print(weather_meta_string)
print('============================= End of META =============================')
```

```
============================ Start of META ============================
title: Measured broadband irradiance and other weather parameters

description: Direct, diffuse, global horizontal and global normal irradiance as
well
  as temperature, humidity, windspeed and pressure. See citation for more
information.

project: 'Global normal spectral irradiance in Albuquerque: a one-year open
dataset
  for PV research'

version: 1.0

date: 2020-11-16

license: CC BY-SA 4.0

authors:
```

```
- Anton Driesse
- Joshua Stein

affiliations:
  Anton Driesse: PV Performance Labs Germany
  Joshua Stein: Sandia National Laboratories

contact:
  Anton Driesse: anton.driesse@pvperformancelabs.com
  Joshua Stein: jsstein@sandia.gov

urls:
- https://pvpmc.sandia.gov/
- https://datahub.duramat.org/project/about/spectral-irradiance-data-and-
resources

citation: 'A. Driesse and J.S. Stein, "Global normal spectral irradiance in
Albuquerque:
  a one-year open dataset for PV research" Sandia Report No. SAND2020-12693,
2020'

location:
  latitude: 35.0545
  longitude: -106.5401
  elevation: 1660
  city: Albuquerque
  state: New Mexico
  country: USA
  timezone: MST
  climatezone: BSk (Tropical and Subtropical Steppe Climate)

unicodetest: W/m²

============================== End of META ==============================
```

[5]:
```
# the general meta data is in YAML format, which is easily parsed into a nested␣
↪dictionary

import yaml

weather_meta = yaml.load(weather_meta_string, Loader=yaml.loader.SafeLoader)

print(weather_meta['project'])
print()
print(weather_meta['affiliations'])
```

```
Global normal spectral irradiance in Albuquerque: a one-year open dataset for PV
research
```

```
{'Anton Driesse': 'PV Performance Labs Germany', 'Joshua Stein': 'Sandia
National Laboratories'}
```

[6]:
```
# now check what's in the column definitions

with h5py.File(WEATHER_FILE, 'r') as f:
    weather_columns = f['/cdef'][()]

# the column definitions are retrieved as numpy structured arrays
# each element of those arrays is utf-8 encoded text, and must be decoded
# it is convenient to load them into a DataFrame before doing this

weather_columns = pd.DataFrame.from_records(weather_columns)
weather_columns = weather_columns.applymap(bytes.decode)
weather_columns.set_index('column', inplace=True)

print(weather_columns.index)
```

```
Index(['timestamp', 'source_year', 'ghi', 'dhi', 'gni', 'dni', 'dni_range',
       'temperature', 'humidity', 'windspeed', 'pressure', 'zenith', 'azimuth',
       'apparent_zenith'],
      dtype='object', name='column')
```

[7]:
```
print(weather_columns.columns)
```

```
Index(['dtype', 'is_index', 'description', 'description_details', 'units',
       'manufacturer', 'model', 'model_details', 'calibration_source',
       'calibration_details'],
      dtype='object')
```

[8]:
```
print(weather_columns[['units', 'manufacturer','model', 'model_details']])
```

```
                units  manufacturer   model           model_details
column
timestamp
source_year
ghi              W/m²  Kipp & Zonen    CM21            unventilated
dhi              W/m²        Eppley     PSP            unventilated
gni              W/m²        Eppley     PSP            unventilated
dni              W/m²  Kipp & Zonen    CHP1
dni_range        W/m²  Kipp & Zonen    CHP1
temperature        °C  Climatronics  100093     Accuracy: +/- 0.1°C
humidity            %  Climatronics  102273        Accuracy: +/- 1%
windspeed         m/s  Climatronics  102083  Accuracy: +/- 0.11 m/s
pressure           Pa  Climatronics  102270      Accuracy: +/- 0.1%
zenith              °
azimuth             °
apparent_zenith     °
```

[9]:
```
# the column definitions give a good indication of what to expect in the data␣
↪section
# and provide lots of useful information for using the data
```

[10]:
```
# the data themselves are also retrieved as numpy structured arrays
# but most of the columns are numeric and ready to use

with h5py.File(WEATHER_FILE, 'r') as f:
    weather_struct = f['/data'][()]

print(weather_struct.dtype)
```

```
[('timestamp', 'S20'), ('source_year', '<i8'), ('ghi', '<f4'), ('dhi', '<f4'),
('gni', '<f4'), ('dni', '<f4'), ('dni_range', '<f4'), ('temperature', '<f4'),
('humidity', '<f4'), ('windspeed', '<f4'), ('pressure', '<f4'), ('azimuth',
'<f4'), ('zenith', '<f4'), ('apparent_zenith', '<f4')]
```

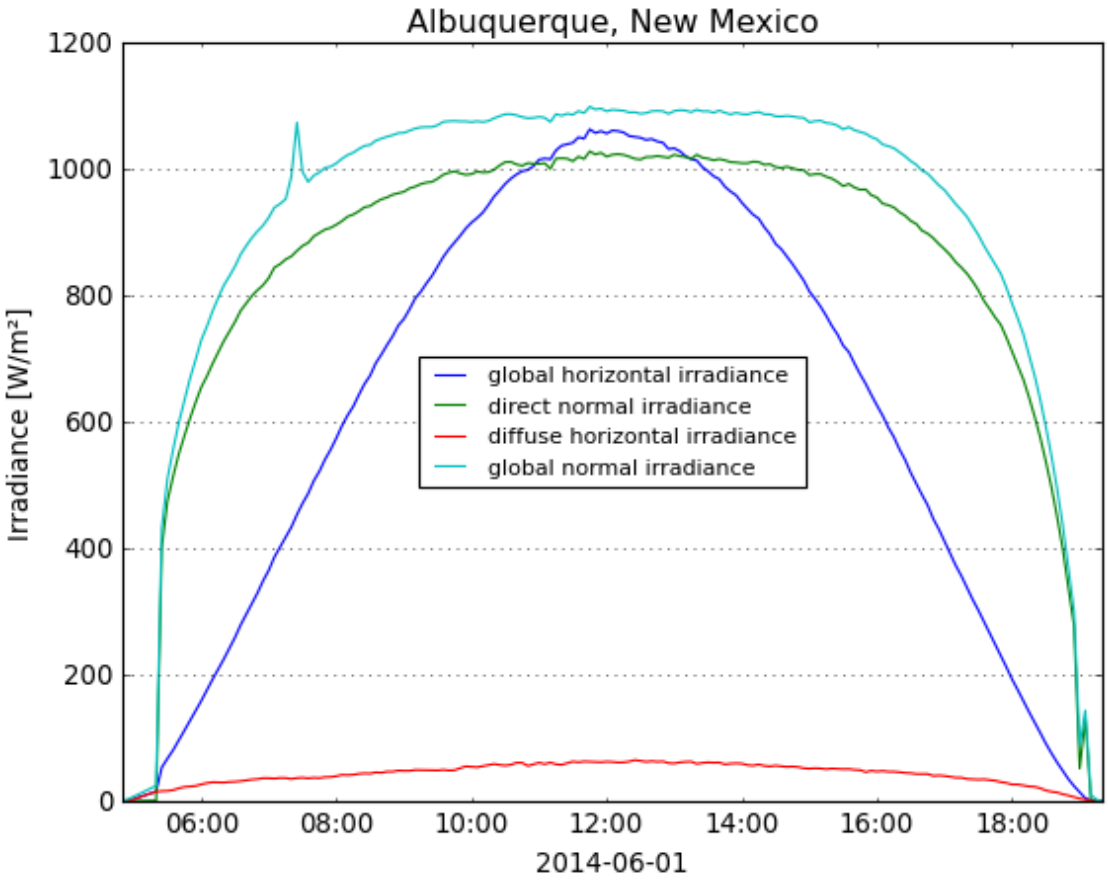
[11]:
```
# structured arrays are fast but don't have as many features as pandas␣
↪DataFrames

print('Maximum GHI:', np.nanmax(weather_struct['ghi']))
```

```
Maximum GHI: 1377.4062
```

[12]:
```
# converting to pandas DataFrames is easy though
# only the timestamps are stored as ISO formatted text
# these must be decoded and converted to datetime
# and then designated as index for the DataFrame

weather = pd.DataFrame.from_records(weather_struct)
weather.timestamp = pd.to_datetime(weather['timestamp'].str.decode('ascii'))
weather.set_index('timestamp', inplace=True)

print(weather.index)
```

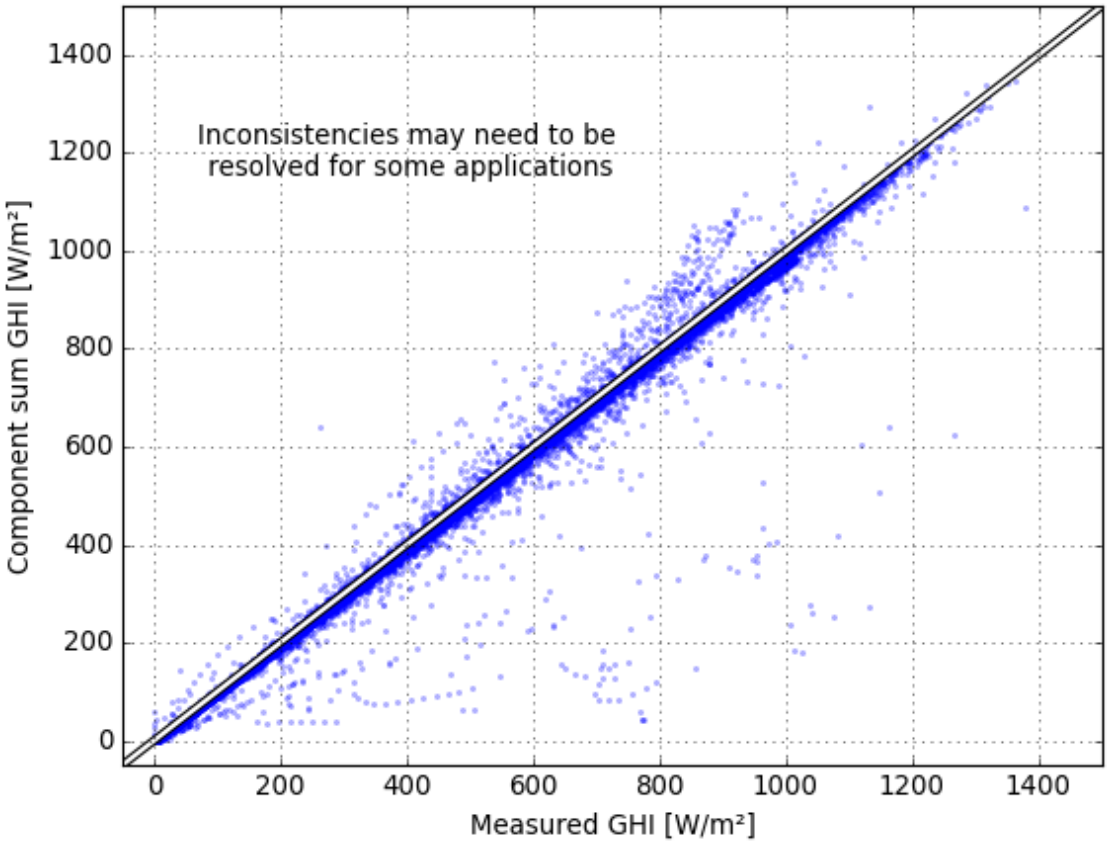
```
DatetimeIndex(['2014-01-01 07:10:00', '2014-01-01 07:15:00',
               '2014-01-01 07:20:00', '2014-01-01 07:25:00',
               '2014-01-01 07:30:00', '2014-01-01 07:35:00',
               '2014-01-01 07:40:00', '2014-01-01 07:45:00',
               '2014-01-01 07:50:00', '2014-01-01 07:55:00',
               …
               '2014-12-31 16:25:00', '2014-12-31 16:30:00',
               '2014-12-31 16:35:00', '2014-12-31 16:40:00',
               '2014-12-31 16:45:00', '2014-12-31 16:50:00',
               '2014-12-31 16:55:00', '2014-12-31 17:00:00',
               '2014-12-31 17:05:00', '2014-12-31 17:10:00'],
              dtype='datetime64[ns]', name='timestamp', length=54221, freq=None)
```

[13]:
```
# now it's familiar territory for pandas users

nice_day = '2014-06-01'
irradiance = ['ghi','dni','dhi','gni']

weather.loc[nice_day, irradiance].plot()
plt.xlabel(nice_day)

# pick up info from the cdef and meta for the plot
plt.ylabel('Irradiance [%s]' % weather_columns.units.ghi);
plt.legend([weather_columns.description[c] for c in irradiance],
           loc='best', fontsize='small')
plt.title('%s, %s' % (weather_meta['location']['city'],
                      weather_meta['location']['state']));
```



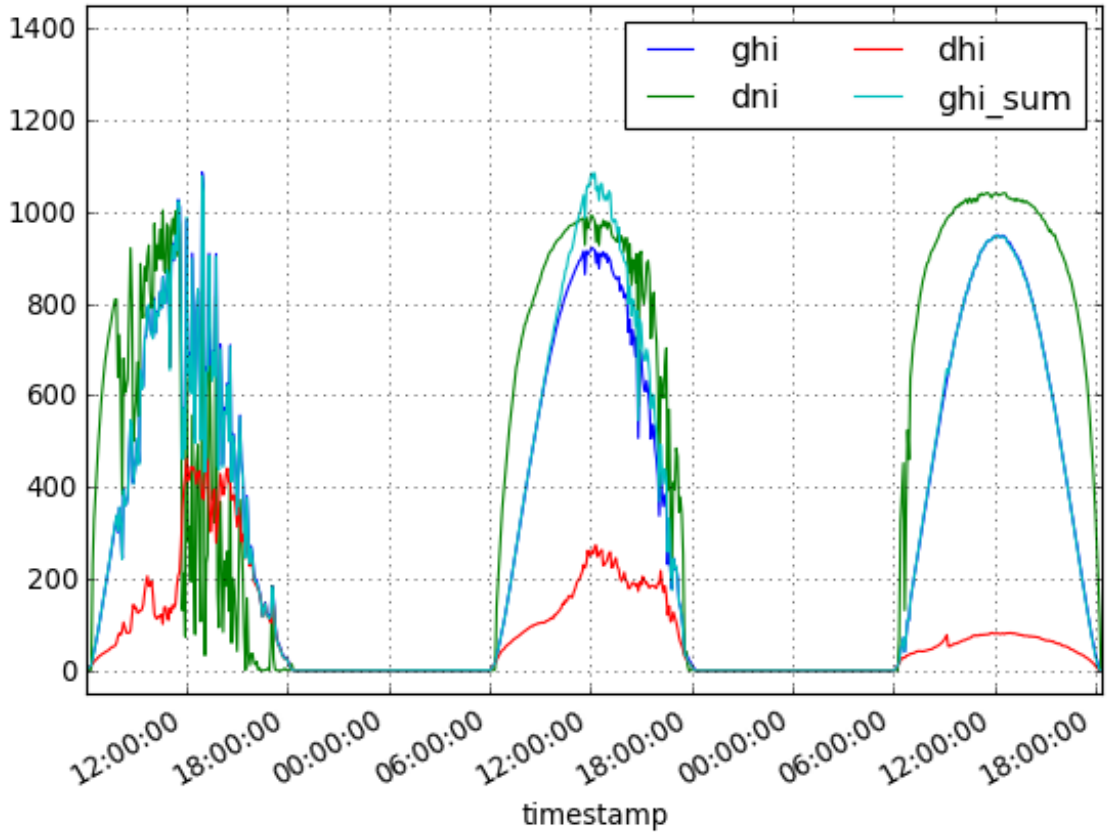
[14]:
```
# check the consistency of the irradiance meassurements

weather['ghi_sum'] = weather.eval('dni * cos(zenith/57.29578) + dhi')
```

```
plt.figure()
plt.plot('ghi', 'ghi_sum', '.b', data=weather, alpha=0.3, ms=4)
plt.plot([-50,1500],[-50,1500],'k-', lw=4)
plt.plot([-50,1500],[-50,1500],'w-', lw=2)
plt.xlim(-50, 1500)
plt.ylim(-50, 1500)
plt.xlabel('Measured GHI [W/m²]')
plt.ylabel('Component sum GHI [W/m²]')

plt.text(400, 1200, 'Inconsistencies may need to be\n resolved for some␣
 ↪applications',
         ha='center', va='center');
```



[15]:
```
# inconsistencies are not too hard to spot because separate
# trackers are used for beam and diffuse measurements

before = '2014-03-22'
after = '2014-03-24'
irradiance = ['ghi','dni','dhi', 'ghi_sum']
```

```
weather.loc[before:after, irradiance].plot()
plt.ylim(-50, 1450)
plt.legend(loc='upper right', ncol=2);
```



[16]:
```
# now read in the spectra and convert to a pandas DataFrame

with h5py.File(SPECTRA_FILE, 'r') as f:
    spectra = f['/data'][()]

spectra = pd.DataFrame.from_records(spectra)
spectra.timestamp = pd.to_datetime(spectra['timestamp'].str.decode('ascii'))
spectra.set_index('timestamp', inplace=True)

print(spectra.index)
```

```
DatetimeIndex(['2014-01-01 07:10:00', '2014-01-01 07:15:00',
               '2014-01-01 07:20:00', '2014-01-01 07:25:00',
               '2014-01-01 07:30:00', '2014-01-01 07:35:00',
               '2014-01-01 07:40:00', '2014-01-01 07:45:00',
               '2014-01-01 07:50:00', '2014-01-01 07:55:00',
               …
```

```
               '2014-12-31 16:25:00', '2014-12-31 16:30:00',
               '2014-12-31 16:35:00', '2014-12-31 16:40:00',
               '2014-12-31 16:45:00', '2014-12-31 16:50:00',
               '2014-12-31 16:55:00', '2014-12-31 17:00:00',
               '2014-12-31 17:05:00', '2014-12-31 17:10:00'],
              dtype='datetime64[ns]', name='timestamp', length=54221, freq=None)
```

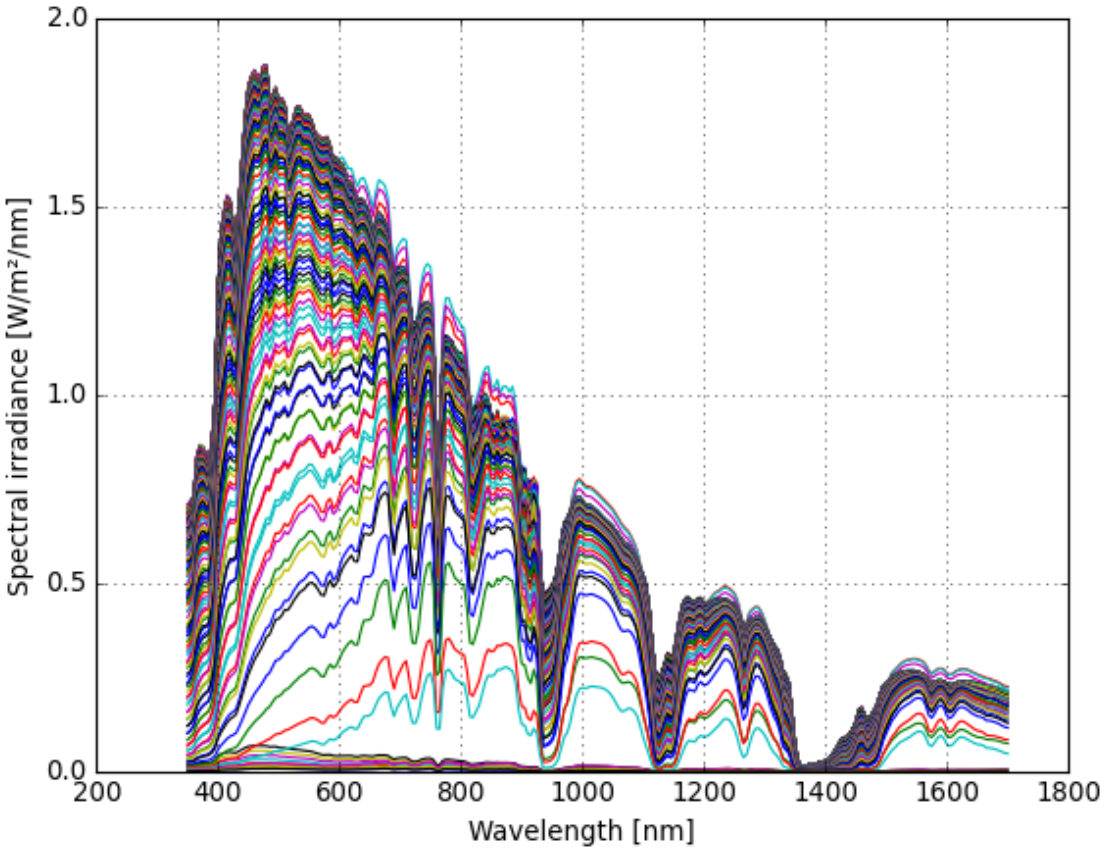
[17]:
```
print(spectra.columns)
```

```
Index(['350.0', '355.0', '360.0', '365.0', '370.0', '375.0', '380.0', '385.0',
       '390.0', '395.0',
       …
       '1655.0', '1660.0', '1665.0', '1670.0', '1675.0', '1680.0', '1685.0',
       '1690.0', '1695.0', '1700.0'],
      dtype='object', length=271)
```

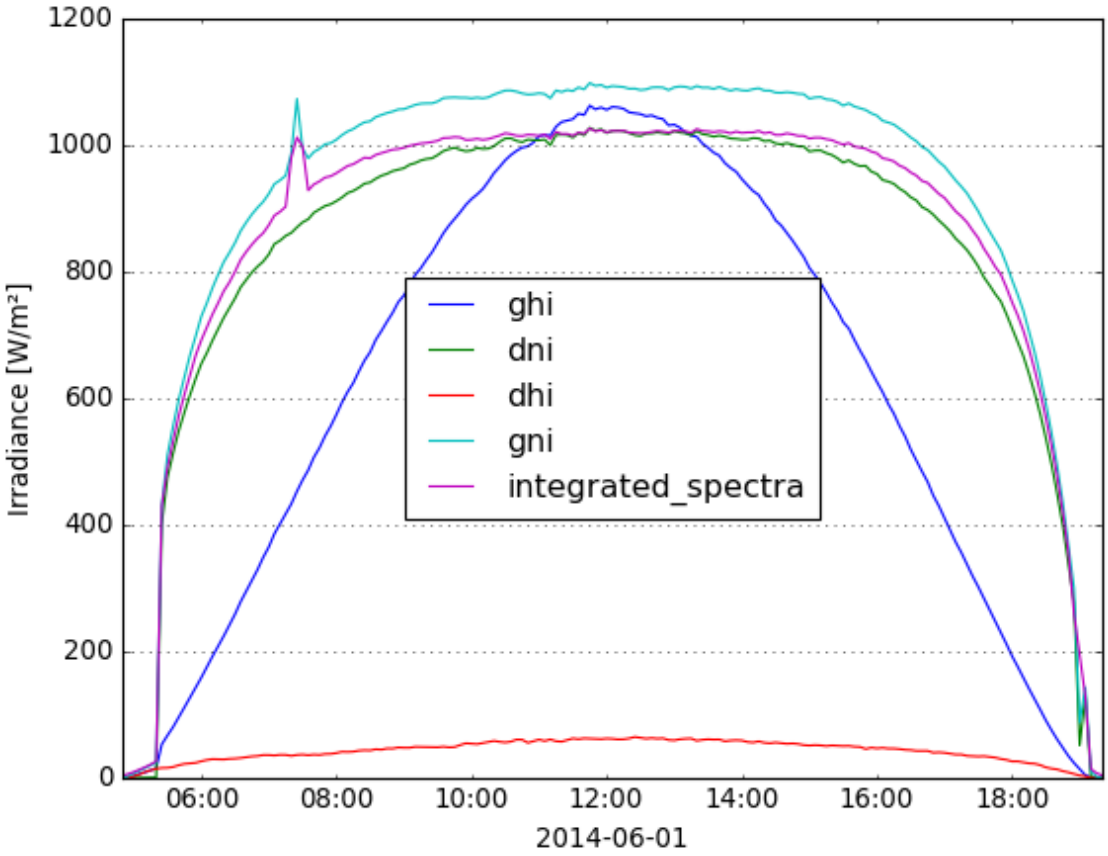
[18]:
```
# the spectral integration will work better later if we change these column␣
↪labels to numbers

spectra.columns = spectra.columns.astype(float)
print(spectra.columns)
```

```
Float64Index([ 350.0,  355.0,  360.0,  365.0,  370.0,  375.0,  380.0,  385.0,
               390.0,  395.0,
              …
              1655.0, 1660.0, 1665.0, 1670.0, 1675.0, 1680.0, 1685.0, 1690.0,
              1695.0, 1700.0],
             dtype='float64', length=271)
```

[19]:
```
# plot some spectra

plt.figure()
plt.plot(spectra[nice_day].T)
plt.xlabel('Wavelength [nm]')
plt.ylabel('Spectral irradiance [W/m²/nm]');
```



[20]:
```
# integrate the spectra and compare to broadband values

wavelengths = spectra.columns.values
weather['integrated_spectra'] = np.trapz(spectra, x=wavelengths, axis=1)

irradiance = ['ghi','dni','dhi','gni', 'integrated_spectra']

weather.loc[nice_day, irradiance].plot()
plt.xlabel(nice_day)
plt.ylabel('Irradiance [W/m²]')
plt.legend(loc='best');
```

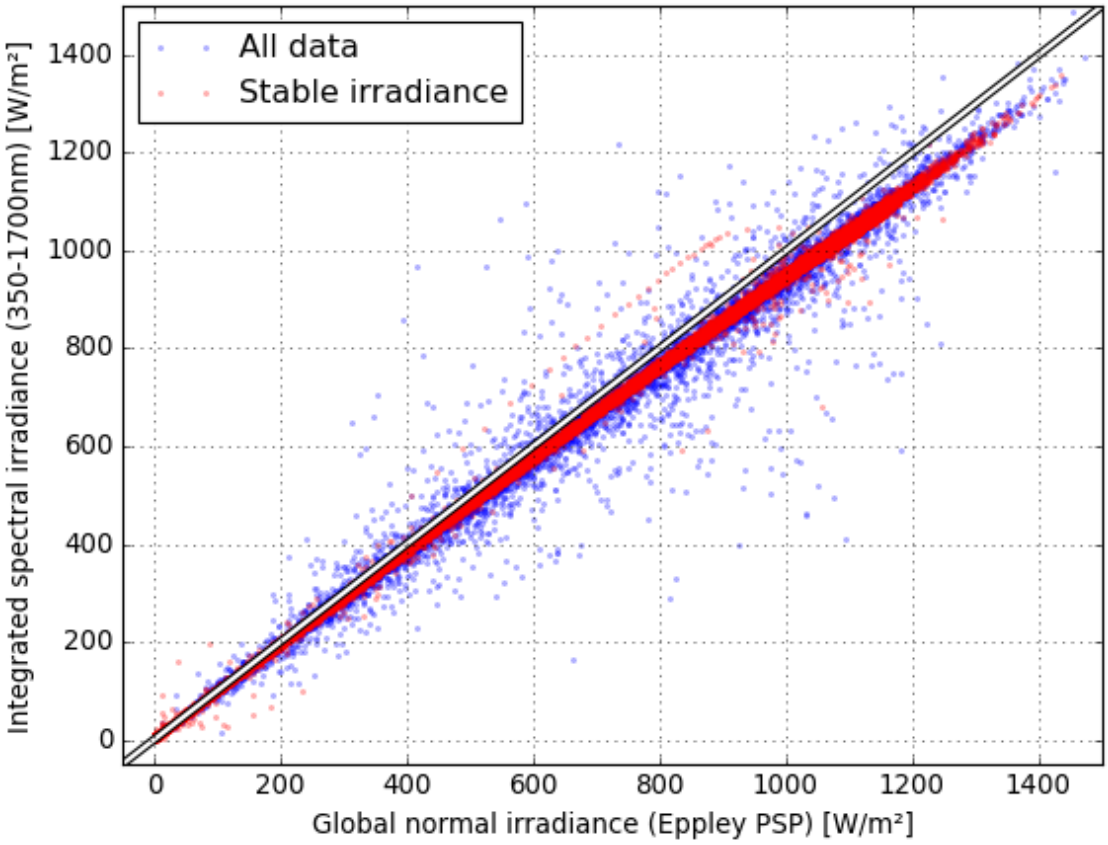


[21]:
```
# now compare the whole year in a scatter plot
# here it is useful to check the 'dni_range' values to see if the conditions␣
 ↪were stable

# what is that? you ask...

print(weather_columns.description_details.dni_range)
```

```
difference between minimum and maximum dni in a centered 15-second interval
```

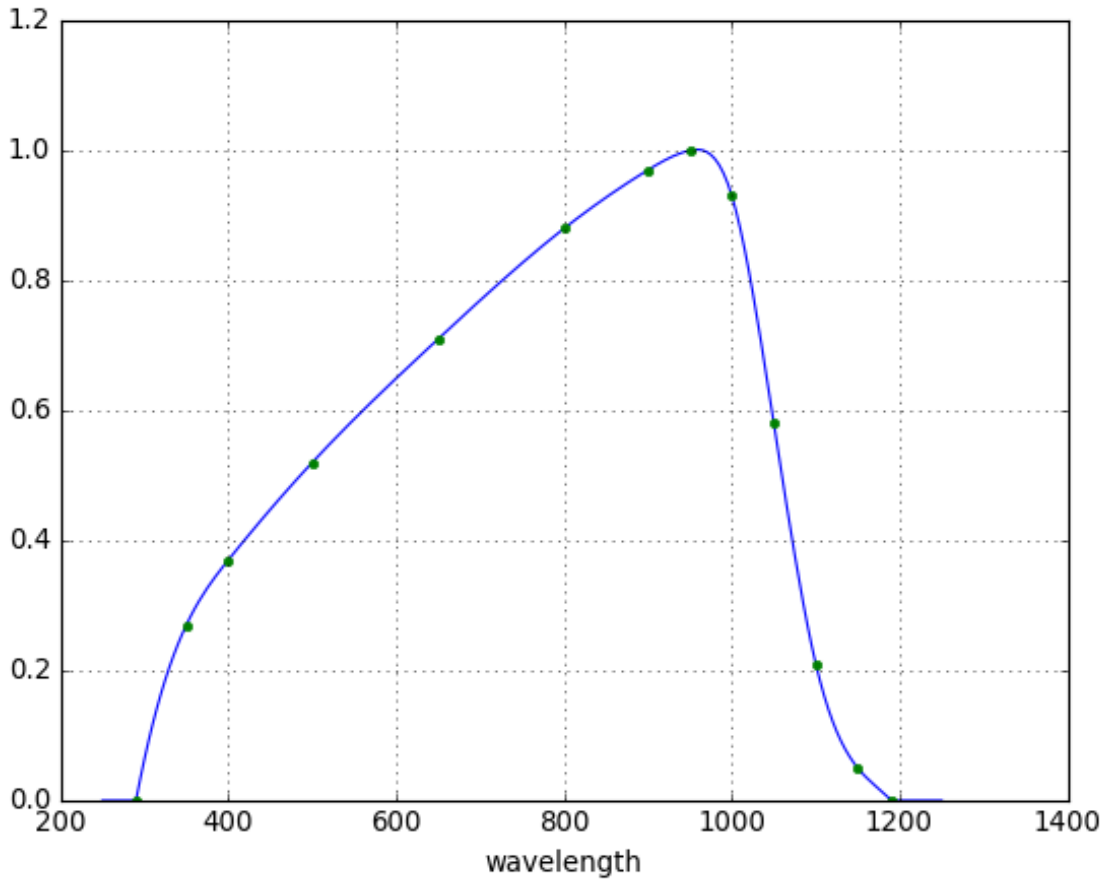
[22]:
```
# ok, that could be useful

stable = weather.dni_range < 10

plt.figure()
plt.plot('gni', 'integrated_spectra', '.b', data=weather[~stable], alpha=.3,␣
 ↪ms=4)
plt.plot('gni', 'integrated_spectra', '.r', data=weather[stable], alpha=.3,␣
 ↪ms=4)
plt.plot([-50,1500],[-50,1500],'k-', lw=4)
```

```
plt.plot([-50,1500],[-50,1500],'w-', lw=2)
plt.xlim(-50, 1500)
plt.ylim(-50, 1500)
plt.xlabel('Global normal irradiance (Eppley PSP) [W/m²]')
plt.ylabel('Integrated spectral irradiance (350-1700nm) [W/m²]')
plt.legend(['All data', 'Stable irradiance'], loc='best');
```

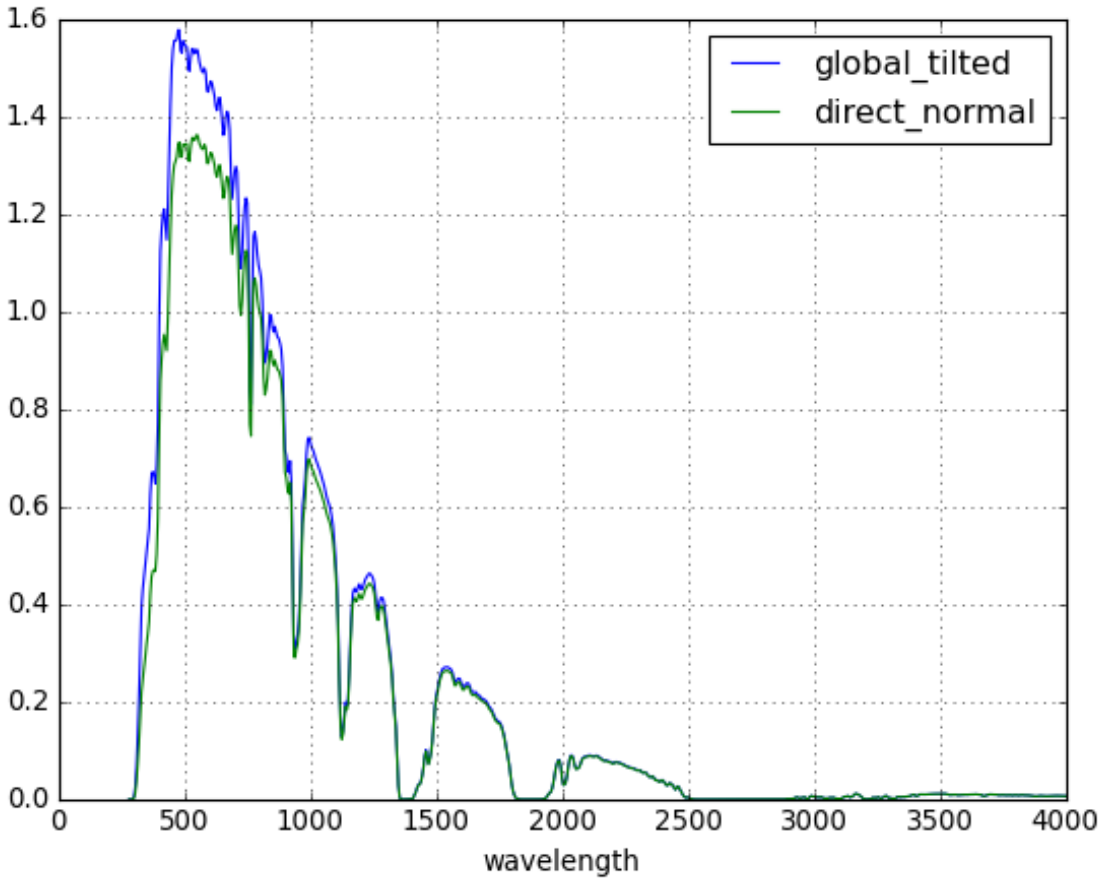


[23]:
```
# the last objective is to calculate spectral mismatch
# here is some generic spectral response data for crystalline silicon

SR_DATA = np.array([[ 290,    0.00],
                    [ 350,    0.27],
                    [ 400,    0.37],
                    [ 500,    0.52],
                    [ 650,    0.71],
                    [ 800,    0.88],
                    [ 900,    0.97],
                    [ 950,    1.00],
                    [1000,    0.93],
                    [1050,    0.58],
```

```
                    [1100,    0.21],
                    [1150,    0.05],
                    [1190,    0.00]]).transpose()

sr = pd.Series(index=SR_DATA[0], data=SR_DATA[1])
sr.name = 'spectral_response'
sr.index.name = 'wavelength'

# use cubic interpolation to obtain values at 5nm intervals
# note: the above points were carefully chosen to achieve
# a smooth interpolated result and avoid undulations

sr = sr.reindex(np.linspace(250.0, 1250.0, 201))
sr.interpolate('cubic', inplace=True, limit_direction='both', fill_value=0.0)

plt.figure()
sr.plot()
plt.plot(SR_DATA[0], SR_DATA[1], '.', ms=8);
```



[24]:
```
# and of course we need a reference spectrum, which is also provided
# note that it has constant wavelength intertervals of 5 nm
# in order to be consistent with the measured spectra

with h5py.File(AM15_FILE, 'r') as f:
    ref_spectra = f['/data'][()]

ref_spectra = pd.DataFrame.from_records(ref_spectra)
ref_spectra.set_index('wavelength', inplace=True)

am15g = ref_spectra['global_tilted']

ref_spectra.plot();
```



[25]:
```
#
```

Some notes about spectral mismatch

Spectral mismatch is really the ratio of two ratios:

$$smm = \frac{\text{usable fraction of measured spectra}}{\text{usable fraction of reference spectrum}}$$

Typically smm is presented as the product of two ratios which obfuscates the meaning.

Two problems that frequently arise are:

1. Wavelength intervals of spectra or spectral response do not match
   - this usually means you have to interpolate to matching wavelengths
   - to make things easier this data set is standardized on 5 nm intervals
2. Wavelength ranges of spectra do not match
   - this means you have to truncate one or extend the other by modeling
   - both options will introduce some errors
   - proceeding with unequal ranges would usually produce bigger errors

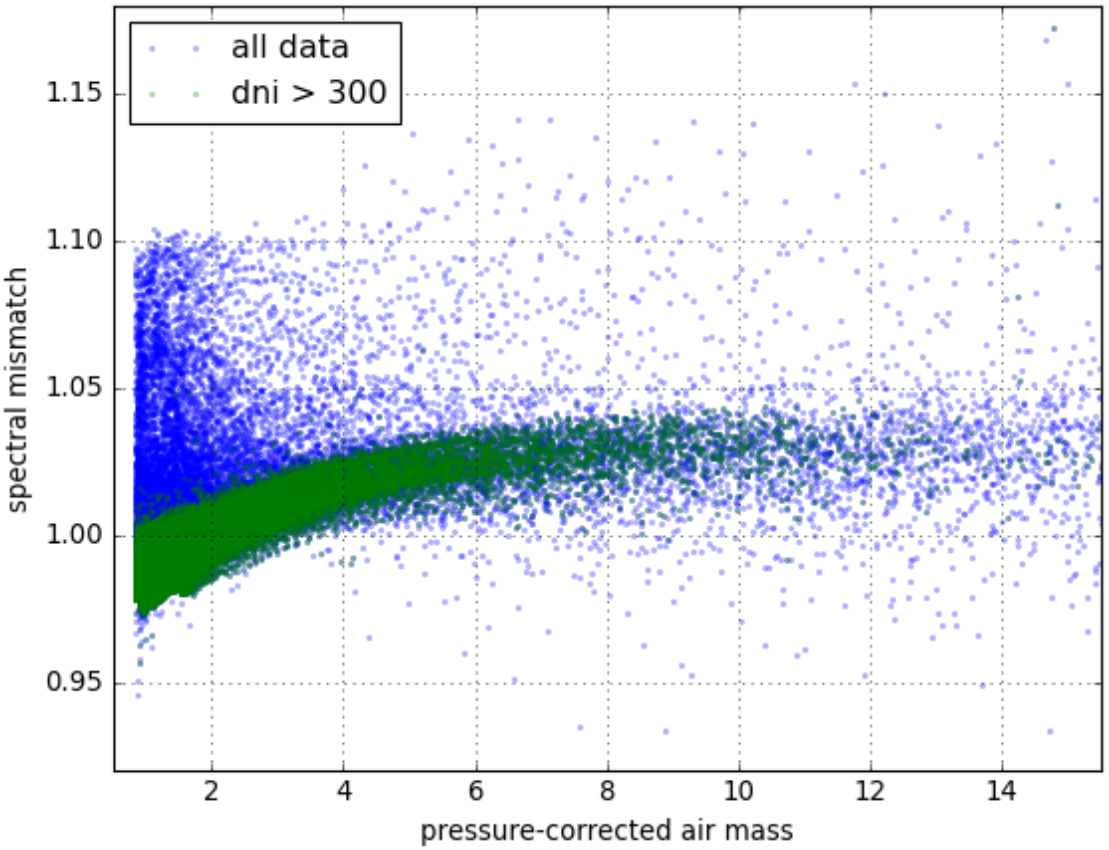
[26]:
```
# here are two helper functions for integration of spectral irradiance

# single spectra or sr will be pandas.Series, with wavelength as the index
# time series of spectra will be pandas.DataFrames, with time as the index
#   and wavelengths as column labels

def integrate_one(s):
    return np.trapz(s.fillna(0.0).values, s.index, axis=0)

def integrate_many(df):
    return np.trapz(df.fillna(0.0).values, df.columns, axis=1)

print('The integral of the global tilted reference spectrum is %.2f W/m².'
      % integrate_one(am15g))
```

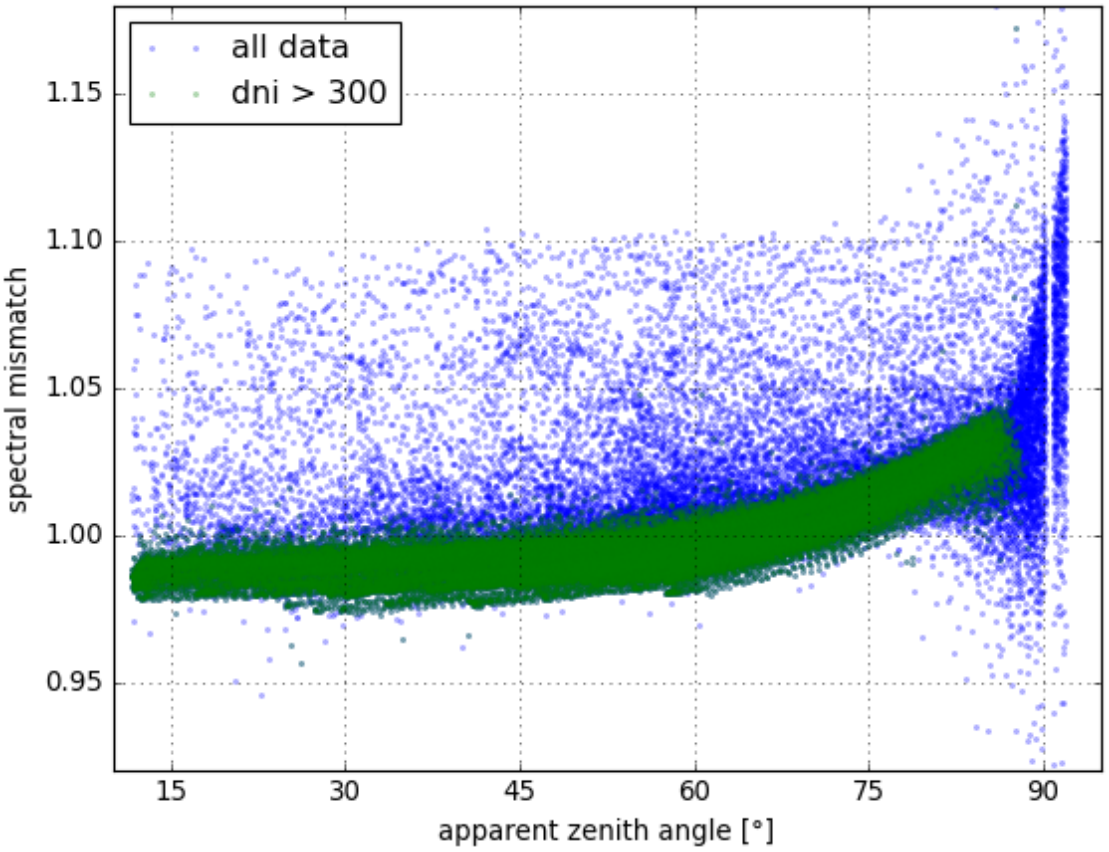
```
The integral of the global tilted reference spectrum is 1000.37 W/m².
```

[27]:
```
# truncate the reference spectrum to match the measured spectra
am15g_t = am15g.reindex(spectra.columns)

# now calculate the usable fractions...
uf_measured  = integrate_many(spectra * sr) / integrate_many(spectra)
uf_reference = integrate_one (am15g_t * sr) / integrate_one (am15g_t)

# ...and their quotient, which is the spectral mismatch
weather['smm'] = uf_measured / uf_reference
```

[28]:
```
# plot mismatch against air mass and zenith angle

from pvlib.atmosphere import get_relative_airmass, get_absolute_airmass

weather['am_abs'] = get_absolute_airmass(get_relative_airmass(weather.
 ↪apparent_zenith),
```

```
                                          weather.pressure)

# identify some clear times
clear = 'dni > 300'

plt.figure()
plt.plot('am_abs', 'smm', '.', data=weather, alpha=.3, ms=4, label='all data')
plt.plot('am_abs', 'smm', '.', data=weather.query(clear), alpha=.3, ms=4,␣
 ↪label=clear)
plt.xlim(0.5, 15.5)
plt.ylim(0.92, 1.18)
plt.xlabel('pressure-corrected air mass')
plt.ylabel('spectral mismatch')
plt.legend(loc='best')

plt.figure()
plt.plot('apparent_zenith', 'smm', '.', data=weather, alpha=.3, ms=4,␣
 ↪label='all data')
plt.plot('apparent_zenith', 'smm', '.', data=weather.query(clear), alpha=.3,␣
 ↪ms=4, label=clear)
plt.xticks([15, 30, 45, 60, 75, 90])
plt.xlim(10, 95)
plt.ylim(0.92, 1.18)
plt.xlabel('apparent zenith angle [°]')
plt.ylabel('spectral mismatch')
plt.legend(loc='best');
```


all data
dni > 300
1.15
1.10
1.05
1.00
0.95
spectral mismatch
2
4
6
8
10
12
14
pressure-corrected air mass


all data
dni > 300
spectral mismatch
1.15
1.10
1.05
1.00
0.95
15
30
45
60
75
90
apparent zenith angle [°]

The end