Scientific coding example: `solar-data-tools` and `pv-system-profiler`

Alejandro Londono *Applied Energy Division*
SLAC National Accelerator Laboratory
Menlo Park, USA
londonoh@slac.stanford.edu

June 25, 2021

1 Longitude, Latitude, Tilt and Azimuth Estimation Example

This notebook presents an example of my coding skills by ilustrating the usage of the `ConfigurationEstimator` class. This class was developed to estimate latitude, longitude, tilt and azimuth, from photovoltaic solar collectors field data. NIST provides data from a collection of fully instrumented and documented arrays installed at their Maryland campus. These arrays have known latitude, longitude, tilt, and azimuth parameters. The data is publicly available here, and documentation of the systems is provided here. In preparation for this work, the available data was first downloaded. This data was originally provided in daily csv files covering a time span of 4 years. The power and irradiance columns from these 6937 individual csv files was extracted and combined to generate a single table spanning four years. The resulting `pickle` file is about 1.1GB in size, and is available from here. The source Jupyter notebook can be found here

All algorithms are available in gitHub at solar-data-tools and pv-system profiler. The example presented here illustrates the use of the pipeline and estimation algorithm using a local csv file containing the input power signals of a small fleet of rooftop photovoltaic systems from NIST. In production, the algorithm has been used to run estimations from an Amazon Web Services S3 bucket containing more than 4000 power signals from a vendor. I was responsible for the development of all code in `pv-system-profiler`, which includes the `ConfigurationEstimator` class as well as many other classes. I am also in charge of maintaining and improving solar-data tools. I am currently in the process of integrating `pv-system-profiler` to the `DataHandler` class of `solar-data-tools`. The goal is to be able to invoke the latitude, longitude, tilt, and azimuth estimation as a method on the `DataHandler` class.

The notebook below presents results of the `ConfigurationEstimator` class, which is aimed at future development of software for end users. I also created the `LongitudeStudy`, `LatitudeStudy` and `TiltAzimuthStudy` classes, which are intended for research and validation purposes. To give an example, the `LongitudeStudy` class was used to find the best performing algorithm combination from 192 different ways of estimating longitude.

Running fleet scale estimations for thousands of systems is computationally intensive. For this reason, I created an algorithm that runs parameter estimations in parallel using Amazon Web Services (AWS). This algorithm was created with an eye on future deployment as a software tool. To give an example, using the parallelization algorithm reduces estimation time for about 4000 power signals from 5 days to 3 hours using 20 AWS instances. The estimation of tilt and azimuth was particularly complex, since it required numerous sub-tasks and data handling. The tilt and azimuth study, and estimator classes are able to choose the unknowns, boundary conditions and initial values based on the provided inputs. This required involved use of a combination of lambda functions and dictionaries.

[1]:
```
%load_ext autoreload
%autoreload 2
```

[3]:
```
#Standard Imports
import pandas as pd
from geopy.distance import great_circle
from glob import glob
from time import time
# Solar Data Tools imports
from solardatatools import DataHandler
# pv-system-profiler imports
import sys
sys.path.append('..')
sys.path.append('./Documents/github/pv-system-profiler')
from pvsystemprofiler.estimator import ConfigurationEstimator
```

2 Configuration information from the published documentation:

[4]:
```
ground_system_configuration = {
    'longitude': -77.2141,
    'latitude': 39.1319,
    'tilt': 20,
    'azimuth': 0,
}
```

3 Load in 4-year data file, derived from NIST csv files.

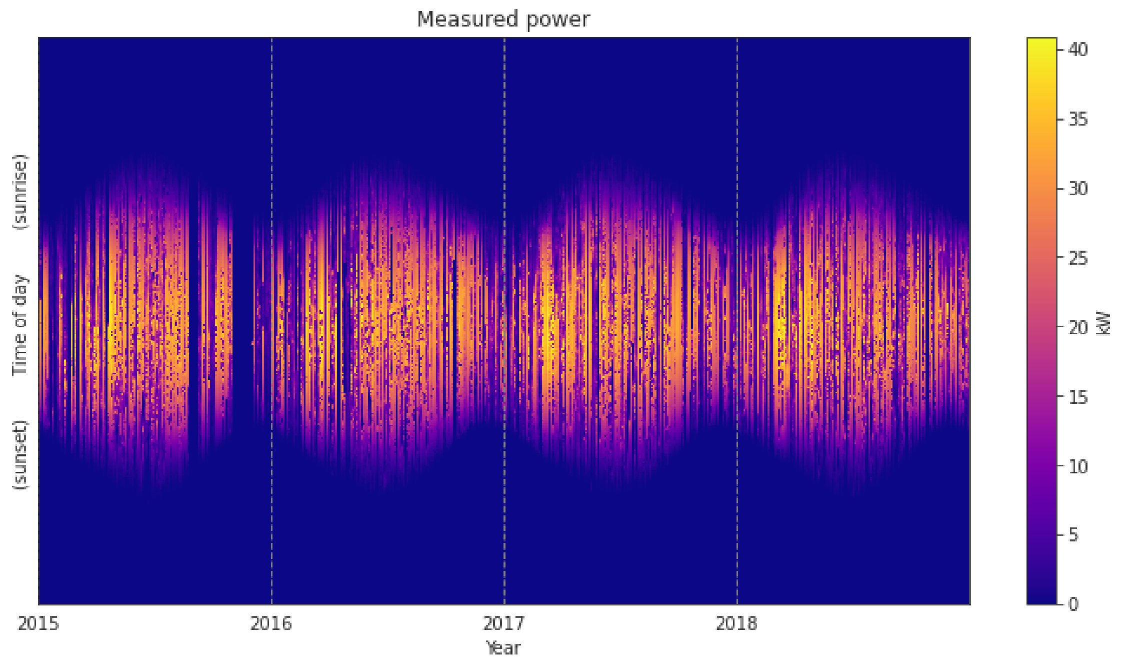
[5]:
```
try:
    df = pd.read_pickle('NIST_all_power_and_irradiance_with_current.pkl')
except FileNotFoundError:
    print('Please download example data file to working directory')
```

4 Utilize standard `solar-data-tools` preprocessing pipeline.

[6]:
```
dh = DataHandler(df)
dh.run_pipeline(power_col=('ground', 'ShuntPDC_kW_Avg_4'))
dh.report()
dh.plot_heatmap(matrix='filled');
```

```
total time: 37.93 seconds
--------------------------------
Breakdown
--------------------------------
Preprocessing              19.93s
Cleaning                   14.77s
Filtering/Summarizing      3.23s
    Data quality           0.76s
    Clear day detect       0.40s
    Clipping detect        0.33s
    Capacity change detect 1.74s

Length:                 4.00 years
Capacity estimate:      30.13 kW
Data sampling:          1.0 minute
Data quality score:     92.1%
Data clearness score:   15.7%
Inverter clipping:      False
Time shifts corrected: False
Time zone correction:   None
```



5 Initialize estimator class with the data handler instance and timezone information.

[7]:
```
est = ConfigurationEstimator(dh, gmt_offset=-5)
```

6 Estimate longitude using default keyboard arguments:

[8]:
```
est.estimate_longitude()
lon = est.longitude

m1 = 'Real Longitude:  {:.2f}\n'.format(ground_system_configuration['longitude'])
m1 += 'Estimated Longitude:  {:.2f}\n'.format(est.longitude)
m1 += 'error:       {:.2f}'.format(ground_system_configuration['longitude'] -␣
 ↪est.longitude)
print(m1)
```

```
Real Longitude:  -77.21
Estimated Longitude:  -77.74
error:       0.52
```

7 Estimate longitude using Haghdadi's published method of taking the median of daily estimates:

[9]:
```
est.estimate_longitude(estimator='calculated', eot_calculation=('da_rosa'),)
lon = est.longitude
m1 = 'real lon:  {:.2f}\n'.format(ground_system_configuration['longitude'])
m1 += 'est  lon:  {:.2f}\n'.format(est.longitude)
m1 += 'error:       {:.2f}'.format(ground_system_configuration['longitude'] -␣
 ↪est.longitude)
print(m1)
```

```
real lon:  -77.21
est  lon:  -77.74
error:       0.52
```

8 Estimate longitude using cost-function minimization method, with a Huber cost.

[10]:
```
est.estimate_longitude(estimator='fit_huber')

m1 = 'Real Longitude:  {:.2f}\n'.format(ground_system_configuration['longitude'])
m1 += 'Estimated Longitude:  {:.2f}\n'.format(est.longitude)
```

```
m1 += 'error:       {:.2f}'.format(ground_system_configuration['longitude'] -␣
 ↪est.longitude)
print(m1)
```

```
Real Longitude:  -77.21
Estimated Longitude:  -77.71
error:       0.49
```

9 Estimate latitude using default keyboard arguments

[11]:
```
est.estimate_latitude()

m1 = 'Real Latitude:  {:.2f}\n'.format(ground_system_configuration['latitude'])
m1 += 'Estimated  Latitude:  {:.2f}\n'.format(est.latitude)
m1 += 'error:       {:.2f}'.format(ground_system_configuration['latitude'] - est.
 ↪latitude)
print(m1)
```

```
Real Latitude:  39.13
Estimated  Latitude:  42.94
error:       -3.81
```

10 Estimate latitude using the optimized method to calculate daylight hours

[12]:
```
est.estimate_latitude(daylight_method='optimized_estimates')
m1 = 'real lat:  {:.2f}\n'.format(ground_system_configuration['latitude'])
m1 += 'est  lat:  {:.2f}\n'.format(est.latitude)
m1 += 'error:       {:.2f}'.format(ground_system_configuration['latitude'] - est.
 ↪latitude)
print(m1)
```

```
real lat:  39.13
est  lat:  42.94
error:       -3.81
```

11 Estimate latitude using sunrise-sunset method to calculate daylight hours

[13]:
```
est.estimate_latitude(daylight_method='sunrise-sunset', data_matrix='filled',␣
 ↪daytime_threshold=0.01)
m1 = 'real lat:  {:.2f}\n'.format(ground_system_configuration['latitude'])
m1 += 'est  lat:  {:.2f}\n'.format(est.latitude)
```

```
m1 += 'error:       {:.2f}'.format(ground_system_configuration['latitude'] - est.
 ↪latitude)
print(m1)
```

```
real lat:  39.13
est  lat:  37.17
error:       1.96
```

12 Estimate error in miles for Longitude and Latitude Calculation

[14]:
```
est = ConfigurationEstimator(dh, gmt_offset=-5)
est.estimate_longitude()
est.estimate_latitude()

lon = est.longitude
lat = est.latitude
ground_coord = (ground_system_configuration['latitude'],␣
 ↪ground_system_configuration['longitude'])
estimated_coord = (lat, lon)
error_dist = great_circle(estimated_coord, ground_coord).miles
m1 = 'Longitude error (Degrees):       {:.2f}\n'.
 ↪format(ground_system_configuration['longitude'] - est.longitude)
m1 += 'Latitude error (Degrees):       {:.2f}\n'.
 ↪format(ground_system_configuration['latitude'] - est.latitude)
m1 += 'error (miles):       {:.2f}'.format(error_dist)
print(m1)
```

```
Longitude error (Degrees):       0.52
Latitude error (Degrees):       -3.81
error (miles):       264.48
```

13 Estimate tilt and azimuth using the AC power signal

[15]:
```
est = ConfigurationEstimator(dh, gmt_offset=-5)
est.
 ↪estimate_orientation(lon_precalculate=ground_system_configuration['longitude']
                        ,␣
 ↪lat_precalculate=ground_system_configuration['latitude'],
                        tilt_precalculate=None,  azimuth_precalculate=None)
m1 = 'Real tilt:  {:.2f}\n'.format(ground_system_configuration['tilt'])
m1 += 'Estimated Tilt:  {:.2f}\n'.format(est.tilt)
m1 += 'error:       {:.2f}\n'.format(ground_system_configuration['tilt'] - est.
 ↪tilt)
m1 += 'Real azimuth:  {:.2f}\n'.format(ground_system_configuration['azimuth'])
m1 += 'Estimated azimuth:  {:.2f}\n'.format(est.azimuth)
```

```
m1 += 'error:       {:.2f}'.format(ground_system_configuration['azimuth'] - est.
 ↪azimuth)
print(m1)
```

```
Real tilt:  20.00
Estimated Tilt:  34.37
error:       -14.37
Real azimuth:  0.00
Estimated azimuth:  -2.67
error:       2.67
```

14 Estimate tilt and azimuth using the DC current signal

[16]:
```
dh = DataHandler(df)
dh.run_pipeline(power_col= ('ground', 'ShuntCurrent_A_Avg_4'), verbose=False)
```

[17]:
```
est = ConfigurationEstimator(dh, gmt_offset=-5)
est.
 ↪estimate_orientation(lon_precalculate=ground_system_configuration['longitude']
                         ,␣
 ↪lat_precalculate=ground_system_configuration['latitude'],
                         tilt_precalculate=None,  azimuth_precalculate=None)
m1 = 'Real tilt:  {:.2f}\n'.format(ground_system_configuration['tilt'])
m1 += 'Estimated Tilt:  {:.2f}\n'.format(est.tilt)
m1 += 'error:       {:.2f}\n'.format(ground_system_configuration['tilt'] - est.
 ↪tilt)
m1 += 'Real azimuth:  {:.2f}\n'.format(ground_system_configuration['azimuth'])
m1 += 'Estimated azimuth:  {:.2f}\n'.format(est.azimuth)
m1 += 'error:       {:.2f}'.format(ground_system_configuration['azimuth'] - est.
 ↪azimuth)
print(m1)
```

```
Real tilt:  20.00
Estimated Tilt:  21.61
error:       -1.61
Real azimuth:  0.00
Estimated azimuth:  -1.37
error:       1.37
```

[ ]:
```

```