solpy Documentation

Release 0.9

Nathan Charles

June 07, 2015

Contents

Solpy is a python library to model solar system power performance similar to PVWatts or NREL's System Advisor Model(SAM). This is primarily a research and analysis tool and there is no guarantee on the calculations.

CHAPTER 1

Quickstart

This is an example of modeling systems annual performance with no shade.

```
from solpy import pv
import json

jsonstring = """
{"system_name":"System Name",
    "zipcode":"17601",
    "tilt":34,
    "azimuth":180,
    "phase":1,
    "voltage":240,
    "array":[
        {"inverter":"SMA America: SB6000US 240V",
        "panel":"Mage Solar : USA Powertec Plus 250-6 MNCS",
        "series":14,
        "parallel":2}
        ]
    }"""

plant = pv.json_system(json.loads(jsonstring))
print plant.model()
#Year 1 Annual _output: 8395.12 kWh
```

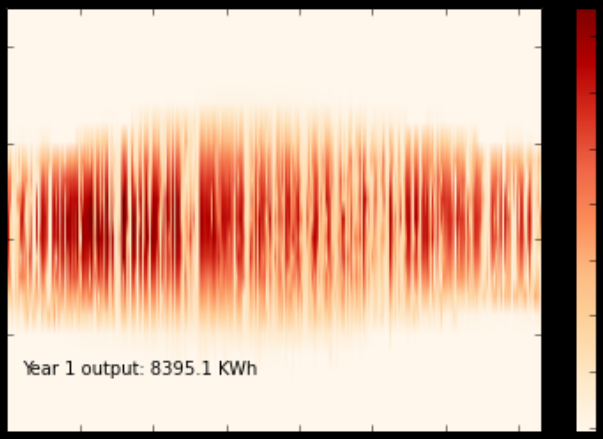
Annual Output can be plotted.

IPython

```
In [1]: from solpy import pv
   ...: import json
   ...:
   ...: jsonstring = """
   ...: {"system_name":"System Name",
   ...:     "zipcode":"17601",
   ...:     "tilt":34,
   ...:     "azimuth":180,
   ...:     "phase":1,
   ...:     "voltage":240,
   ...:     "array":[
   ...:         {"inverter":"SMA America: SB6000US 240V",
   ...:         "panel":"Mage Solar : USA Powertec Plus 250-6 MNCS",
   ...:         "series":14,
   ...:         "parallel":2}
   ...:         ]
   ...:     }"""
   ...:
   ...: plant = pv.json_system(json.loads(jsonstring))
   ...:
WARNING: forecast.io key not set.
Realtime weather data not availible.

In [2]: plant.model()
Out[2]: Year 1 Annual output: 8395.12 kWh

In [3]: plant.model().plot()
Out[3]:
```

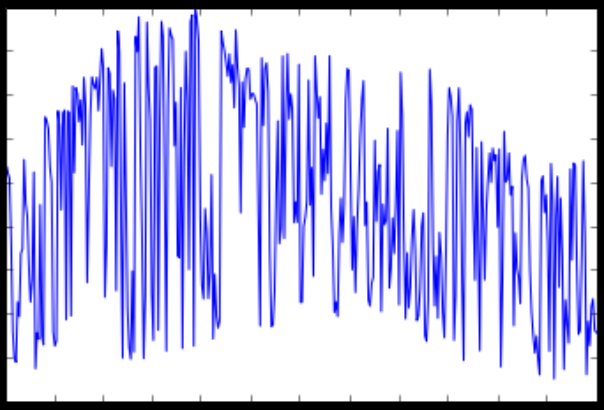


Annual output can also be charted.

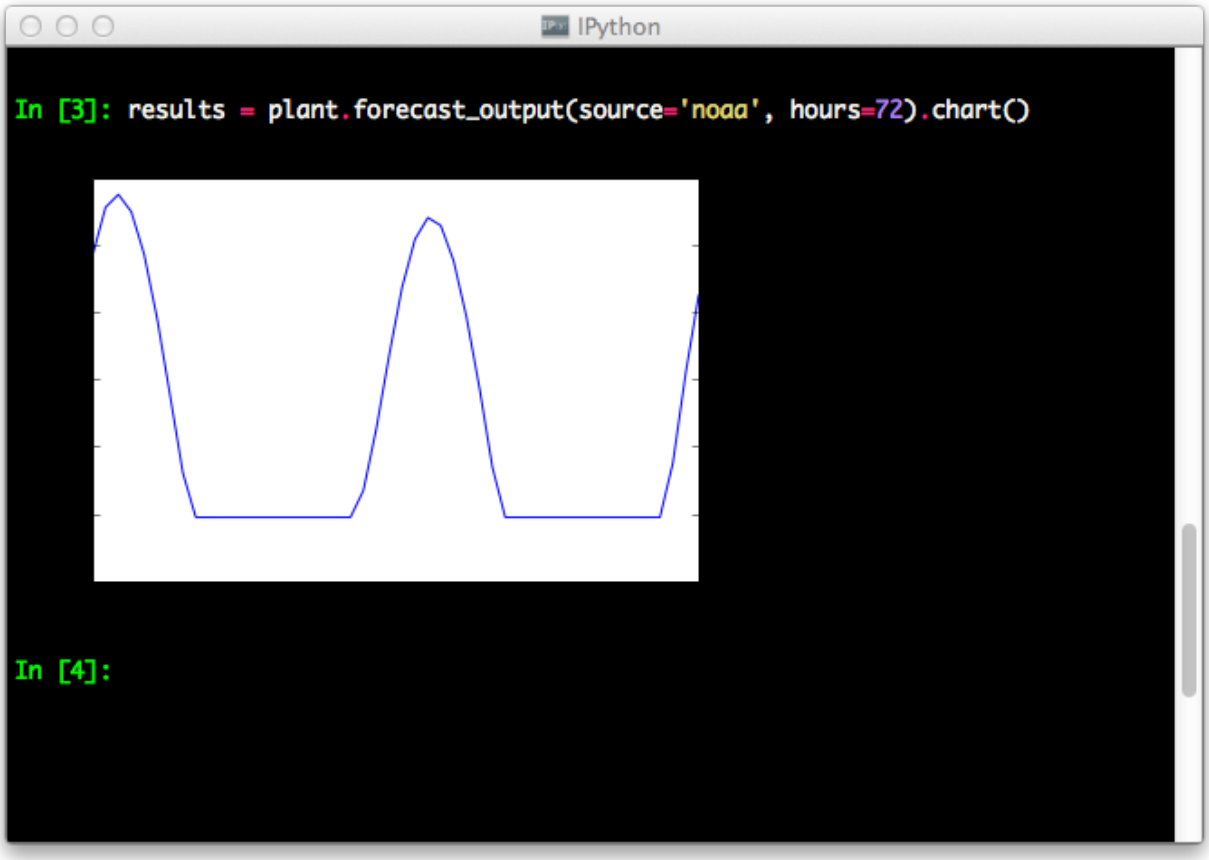
IPython

```
In [1]: from solpy import pv
   ...: import json
   ...:
   ...: jsonstring = """
   ...: {"system_name":"System Name",
   ...:     "zipcode":"17601",
   ...:     "tilt":34,
   ...:     "azimuth":180,
   ...:     "phase":1,
   ...:     "voltage":240,
   ...:     "array":[
   ...:         {"inverter":"SMA America: SB6000US 240V",
   ...:         "panel":"Mage Solar : USA Powertec Plus 250-6 MNCS",
   ...:         "series":14,
   ...:         "parallel":2}
   ...:         ]
   ...:     }"""
   ...:
   ...: plant = pv.json_system(json.loads(jsonstring))
   ...:
WARNING: forecast.io key not set.
Realtime weather data not availible.

In [2]: plant.model().chart()
Out[2]:
```



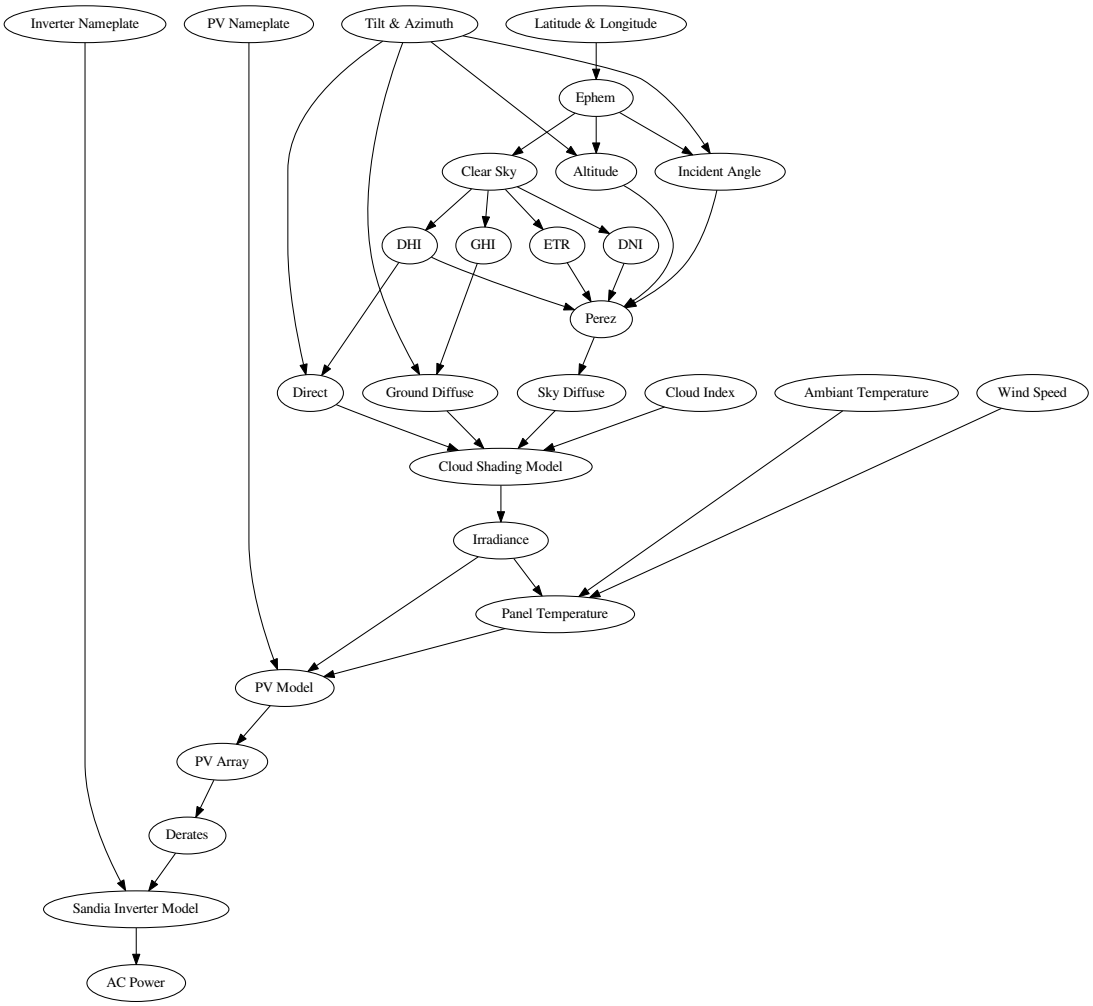
It is also possible to do some basic forecasting using NOAA GFS or forecast.io


IPython
In [3]: results = plant.forecast_output(source='noaa', hours=72).chart()
In [4]:

CHAPTER 2

Theory and model interaction

Modeling the solar contribution on a structure requires a strong an understanding the solar resource. This is a synthesis of many components starting with external flux (ETR), ephemeris of the sun, atmospheric conditions and obstructions and albedo.



This graph shows the interaction of models. Models can be connected to interact in different ways. For example shading obstructions can be applied after irradiance has been calculated. This is the method some tools in the solar industry such as the Solar Pathfinder use to simplify the process. However it can be applied earlier to better estimate diffuse and reflected components.

2.1 Ephemeris

The ephemeris, or position of the sun, can be calculated at a time using algorithms such as the Sun Position Algorithim (SPA), solpy uses the pyephem package currently.

2.2 Clear Sky Models

Irradiance calculations generally involves some level of modeling since historic weather data is generally not available at the correct orientation for the site. A synthesis of irradiance starts starts with clear sky models. A value for external flux (ETR) is calculated and adjusted for atmospheric conditions such as airmass. The most common clear sky models are from Liu and Jordan, and Bird. These models calculate the parameters of Direct Normal Irradiance (DNI), Global Horizontal Irradiance (GHI) and Diffuse Horizontal Irradiance (DHI).

2.3 Cloud Shading

Having estimated the clear sky component, clouds must be accounted for. There has been empirical correlation between and cloud cloud cover C and bright sunshine σ.

$$\kappa = aC + bC^2$$

$$\sigma = 1 - \kappa$$

The coefficients a and b are determined using least squares regression fit for the observed values of cloud shade. The complement of σ is often called cloud shade and can be used to adjust the clear sky components.

2.4 Typical Meteorological Year (TMY)

The alternative to synthetic irradiance is using Typical Meteorological Year (TMY) data sets. These data sets are chosen based on historical weather data measured at various Airport weather stations. Monthly data is cherry picked to create annual datasets representational of typical weather for a location. TMY3 is the current version and gives measured GHI, DNI, DHI, ETR, cloud index as well as various other ambient conditions.

2.5 Beam

These general irradiance components need to be adapted to the local site and orientation. The beam component of radiation is based on DNI and is adjusted for the angle of incidence. Angle of incidence is calculated where Σ is tilt of collector and ϕ_c is azimuth of collector.

$$\cos\theta_I = \cos\alpha\cos(\phi_s - \phi_c)\sin\Sigma + \sin\alpha\cos\Sigma$$

The beam component is then calculated.

$$I_{beam} = I_{DNI}\cos\theta_I$$

2.6 Ground Reflected

The ground reflected component of irradiance is largely a function of albedo (ρ) and incident angle. It is well characterized by the function:

$$I_{reflected} = \rho(I_{beam} + I_{diffuse})(\frac{1 - \cos\Sigma}{2})$$

2.7 Diffuse Sky models

Calculating the diffuse sky irradiation component is the area with the least consensus. Noorian, et al. compare 12 different models and that is not an exhaustive list. In general they fall into two categories: Isotropic and non-isotropic. Isotropic models assume that the diffuse radiation is uniform across the sky. Liu & Jordan developed a commonly used isotropic diffuse model:

$$I_{diffuse} = DHI \cdot \frac{1 + \cos\theta_c}{2}$$

Anisotropic models are much more complicated. The Perez 90 model is often used but is much more complicated and was developed around computer simulation. The basic form is seen in the following equation where the coefficients are developed from empirical data.

$$0.5[1 - F_1'](1 + \cos\theta_c) + F_1'\frac{a}{b} + F_2'\sin\theta_c$$

2.8 Total Irradiance on an inclined plane

$$I_{total} = I_{beam} + I_{diffuse} + I_{reflected}$$

2.9 PV Module Temperature

NREL proposes a 3 parameter model for PV module temperature.

$$T_{module}(^\circ C) = 0.943 \cdot T_{ambient} + 0.028 \cdot I_{total} - 1.528 \cdot WindSpeed + 4.3$$

2.10 CEC PV module model

The single diode model is the most common module model.

http://pvpmc.org/modeling-steps/module-iv-curve/diode-equivalent-circuit-models/ http://pvpmc.org/pv-lib/functions-by-catagory/pvl_calcparams_desoto/

Solpy currently doesn't use this by default because of performance issues, rather it uses a simpler formuation.

$$Power = \frac{I_{total}}{1000} \cdot I_{mpp} \cdot (V_{mpp} - tk_{Vmp} \cdot (25 - T_{module}))$$

2.11 Sandia Inverter Model

Inverter power is calculated using the Sandia Inverter model described in King, David L, Sigifredo Gonzalez, Gary M Galbraith, and William E Boyson. 2007. "Performance Model for Grid-Connected Photovoltaic Inverters."

$$P_{ac} = (P_{aco}/(A - B)) - C \cdot (A - B) \cdot (P_{dc} - B) + C \cdot (P_{dc} - B)^2$$

where:

$$A = P_{dco} \cdot (1 + C_1(V_{dc} - V_{dco}))$$
$$B = P_{so} \cdot (1 + C_2(V_{dc} - V_{dco}))$$
$$C = C_o \cdot (1 + C_3(V_{dc} - V_{dco}))$$

CHAPTER 3

modules

PV array classes

class solpy.modules.**Array**(*module*, *shape*)
rewrite of pvArray

dec()
decrease channel with most panels

dump()
dump to dict

i_mpp()
total mppt circuit current

i_sc()
total short circuit current

inc()
increase channel with least panels

maxlength(*maxl*)
max length of string. needs to be set before running

mcount()
module count

minlength(*minl*)
min length of string. needs to be set before running

output(*insolation*, *t_ambient=25*)
total dc power output

v_dc(*t_cell=25*)
todo:not sure what this should return

v_max(*ashrae_min*)
max voltage

v_min(*ashrae2p*, *t_adder=30*)
min voltage under load

class solpy.modules.**Module**(*model*)
generic module class uses JSON defintion

i_dc(*t_cell=25*)
Current adjusted for temperature

mppt_max(*irradiance*, *t_cell*)
find mppt_max conditions

mppt_search(*irradiance*, *t_cell*, *v_low*, *v_mid*, *v_hi*)
golden rect search

output(*insolation*, *t_cell=25*, *simple=True*)
Watts DC output

single_diode(*irradiance*, *t_cell*, *v_diode*)
single diode model with CEC coefficients

v_dc(*t_cell=25*)
Voltage of module at cell tempture

v_max(*ashrae_min*)
Max Voltage at minimum temperature

v_min(*ashrae2p*, *t_adder=30*)
Minimum voltage of module under load

vi_output(*insolation*, *t_cell=25*, *simple=True*)
Watts DC output

class solpy.modules.**Mppt**(*module*, *series*, *parallel=1*)
structure to aggregate panels into an array)

dec()
decrease number of panels in channel

dump()
dump to dict

i_mpp()
mppt current

i_sc()
short circuit current

inc()
increase number of panels in channel

output(*insolation*, *t_ambient=25*)
watts output of channel

v_dc(*t_cell=25*)
channel voltage at temperature

v_max(*ashrae_min*)
max channel voltage at temperature

v_min(*ashrae2p*, *t_adder=30*)
min channel voltage under load

solpy.modules.**manufacturers**()
return list of panel manufacturers

solpy.modules.**model_search**(*parms*)
search for a module model

solpy.modules.**models**(*manufacturer=None*)
returns list of available panel models

CHAPTER 4

inverters

Inverter class and related functions methods

class solpy.inverters.**Inverter**(*model, array=None, orientation=[(180, 0)]*)
Sandia Inverter Model

Inverter power is calculated using the Sandia Inverter model.

This is described in King, David L, Sigifredo Gonzalez, Gary M Galbraith, and William E Boyson. 2007.

"Performance Model for Grid-Connected Photovoltaic Inverters."

$$P_{ac} = (P_{aco}/(A-B)) - C \cdot (A-B) \cdot (P_{dc}-B) + C \cdot (P_{dc}-B)^2$$

where:

$$A = P_{dco} \cdot (1 + C_1(V_{dc} - V_{dco}))$$
$$B = P_{so} \cdot (1 + C_2(V_{dc} - V_{dco}))$$
$$C = C_o \cdot (1 + C_3(V_{dc} - V_{dco}))$$

dump()
dump to dict

i_ac(*insolation*, *v_ac*)
ac current

p_ac(*insolation*, *t_cell=25*)
AC power in Watts

ratio()
AC/DC ratio

solpy.inverters.**manufacturers**()
get list of manufacturers

solpy.inverters.**models**(*manufacturer=None*)
returns list of available inverter models

CHAPTER 5

pv

Photovoltaic System Performance Monitoring

class solpy.pv.**ResultSet**
system moduling results

chart()
plots chart of values

dump()
returns list of python datetime timestamps and values in Watts

dumps()
returns list of unix timestamps and values in Watts

plot()
plots heatmap of values

summary()
prints summary

class solpy.pv.**System**(*shape*)
PV System

describe()
describe system

dump()
dump to dict

forecast_output(*daylightSavings=False*, *source=None*, *hours=24*)
forecast output of system

min_row_space(*delta*, *rise_hour=9*, *set_hour=15*)
Row Space Function

min_setback(*delta*, *rise_hour=9*, *set_hour=15*)
East West _setback

model(*model_name='p9'*, *single_thread=False*)
model pv system performance

now(*timestamp=None*, *weather_data=None*, *model='STC'*)
Preditive power output

p_ac(*ins*, *t_cell=25*)
ac power output

p_dc(*ins*, *t_cell=25*)
dc power output

set_zipcode(*zipcode*, *station_class=3*)
update zipcode

solstice(*hour*)
position on winter soltice (Dec 21)

virr(*p_ac*, *timestamp=None*, *weather_data=None*)
calculate virtual irradiation

solpy.pv.**json_system**(*json_description*)
Load a system from a json description

solpy.pv.**load_system**(*filename*)
Load a system from a json file

CHAPTER 6

design

Parametric Design tools

solpy.design.**celery_worker_status**()
get celery worker status

solpy.design.**combinations**(*_a*, *_b*)
generate combinations

solpy.design.**design**(*reqs*, *ranking=None*)
Design a PV system based upon various ranking algorithms.

Args: reqs (dict): JSON object of design constriants. Shading is an optional constraint.

ranking (list): algorithms that define valuation of parts. The default rankings are knapsack and efficient.

Returns: list of systems

For example:

```
    >>> reqs = {"system_name":"HAPPY CUSTOMER",
        "address":"15013 Denver W Pkwy, Golden, CO",
        "zipcode":"80401",
        "phase":1,
        "voltage":240,
        "service":200,
        "tilt":25,
        "azimuth":180,
        "notes":"reqs",
        "inverter options":["SMA America: SB5000TL-US-22 (240V) 240V",
            "SMA America: SB7000TL-US-12 (240V) 240V",
            "SMA America: SB8000TL-US-12 (240V) 240V",
            "SMA America: SB9000TL-US-12 (240V) 240V",
            "SMA America: SB6000US-11 240V"],
        "panel options":["Axitec : AC-250P-156-60S *"],
        "space":[[10,5]],
        "desired size":25000}
    >>> design(reqs, ranking=[efficient])
    [{'DCnominal': 23100,
    'address': '15013 Denver W Pkwy, Golden, CO',
    'algorithm': 'efficient',
    'array': [{'inverter': u'SMA America: SB5000TL-US-22 (240V) 240V',
        'panel': 'Axitec : AC-250P-156-60S *',
        'quantity': 1,
        'shape': [{'parallel': 1, 'series': 12},
            {'parallel': 1, 'series': 11}]},
```

```
    {'inverter': u'SMA America: SB5000TL-US-22 (240V) 240V',
        'panel': 'Axitec : AC-250P-156-60S *',
        'quantity': 1,
        'shape': [{'parallel': 1, 'series': 12},
            {'parallel': 1, 'series': 11}]},
    {'inverter': u'SMA America: SB5000TL-US-22 (240V) 240V',
        'panel': 'Axitec : AC-250P-156-60S *',
        'quantity': 1,
        'shape': [{'parallel': 1, 'series': 12},
            {'parallel': 1, 'series': 11}]},
    {'inverter': u'SMA America: SB5000TL-US-22 (240V) 240V',
        'panel': 'Axitec : AC-250P-156-60S *',
        'quantity': 1,
        'shape': [{'parallel': 1, 'series': 12},
            {'parallel': 1, 'series': 11}]}],
    'azimuth': 180,
    'notes': 'symetric design of most efficient combination',
    'phase': 1,
    'system_name': 'HAPPY CUSTOMER',
    'tilt': 25,
    'voltage': 240,
    'yearone': 35746.2,
    'zipcode': '80401'}]
```

solpy.design.**efficient**(*items*, *maxweight*)
symetric design of the most efficeint inverter panel combo

solpy.design.**fill**(*inverter*, *zipcode*, *ac_dc_ratio=1.2*, *mount='Roof'*, *station_class=1*, *v_max=600*, *bipolar=True*)
deprecated use generate_options

solpy.design.**generate_options**(*inverter_name*, *module_name*, *zipcode*, *ac_dc_ratio=1.2*, *mount='Roof'*, *station_class=1*, *v_max=600*, *bipolar=True*)
String sizing: find all valid configurations for a location

solpy.design.**knapsack**(*item_set*, *maxweight*)
knapsack problem weight is system DC size and value is annual output this could be expanded with different constraints for different rankings

solpy.design.**performance_model_plant**(*json_def*)
model performance of a system

solpy.design.**performance_model_set**(*clist*)
wrapper for distributed performance modelling

solpy.design.**str_format**(*inverter*)
format as str: '9769.5W : 13S x 3P : ratio 1.22 : 314.0 - 552.0 V'

solpy.design.**tools_fill**(*inverter*, *zipcode*, *ac_dc_ratio=1.2*, *mount='Roof'*, *station_class=1*, *v_max=600*, *bipolar=True*)
deprecated legacy function

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

S

R

S

T

V