solarenergy

Feb 05, 2023

Contents:

CHAPTER 1

solarenergy package

1.1 Submodules

1.1.1 solarenergy.solar_panels module

Functions for solar energy dealing with solar (PV) panels/modules.

class solarenergy.solar_panels.**SolarPanels**(*geo_lon: float = 0.0, geo_lat: float = 0.0, tz: str = None, az: float = 0.0, incl: float = 0.0, area: float = 0.0, p_max: float = 0.0, eff0: float = 0.0, deff_dt: float = 0.0, year: float = 2015, t_coef: float = -0.005, n_refr: float = 1.43, inv_model: str = None, inv_sn: str = None, name: str = None*)

Bases: object

Dataclass containing solar-panel parameters.

area = 0.0
Surface area of solar PV panels (m2; typically 1.6m2 per panel)

az = 0.0
'Azimuth' of the panel normal vector (rad; 0=S, π/2=W)

deff_dt = 0.0
degrades to 90% after 20 years)

Type Linear degradation of efficiency factor over time (yr^-1; -5e-3

eff0 = 0.0
Original efficiency of solar panels + inverter, at installation (0-1; e.g. 0.15 for 15%)

geo_lat = 0.0
Geographic latitude of the panels (rad; >0 for east of Greenwich)

geo_lon = 0.0
Geographic longitude of the panels (rad; >0 for northern hemisphere)

incl = 0.0
'Zenith angle' of the panel normal vector (rad; 0=horizontal, π/2=vertical)

inv_model = None
Model or type of the inverter

inv_sn = None
Serial number of the inverter

n_refr = 1.43
1.000293)

Type Refractive index of PV cover (typically 1.43; air

name = None
PV plant name

p_max = 0.0
Maximum electrical power of solar PV panels or inverter (kW)

t_coef = -0.005
PV temperature coefficient (/K; typically -0.005)

tz = None
Time zone where the solar panels sit (e.g. Europe/Berlin)

year = 2015
Installation year (e.g. 2015.25 for 2015-04-01)

solarenergy.solar_panels.**pv_cell_temperature**(*temp_a*, *glob_insol*, *v_wind*, *temp_c_noct=45*, *eta_c_noct=0.15*, *t_coef=-0.0045*, *glob_insol_noct=800*, *temp_a_noct=20*, *v_wind_noct=1*, *tau_alp=0.9*)

Estimate the PV-cell temperature as a function of ambient temperature, insolation and wind velocity.

Parameters

- **temp_a** (*float*) – Ambient temperature (°C or K - ensure all temperatures use the same unit).
- **glob_insol** (*float*) – Global projected insolation on PV cell (W/m2).
- **v_wind** (*float*) – Wind velocity (m/s).
- **temp_c_noct** (*float*) – Cell temperature for nominal operating cell temperature (NOCT; °C or K).
- **eta_c_noct** (*float*) – Cell/module efficiency for NOCT (-).
- **t_coef** (*float*) – Temperature coefficient for Pmpp and η_cell (/K).

- **glob_insol_noct** (*float*) – Projected global insolation on the PV module for NOCT (W/m2).
- **temp_a_noct** (*float*) – Ambient temperature for NOCT (°C or K).
- **v_wind_noct** (*float*) – Wind velocity for NOCT (m/s).
- **tau_alp** (*float*) – Tau alpha ($\tau\alpha$): the optical transmission-absorption coefficient for the PV module (0-1).

Returns PV cell temperature (°C or K).

Return type (float)

Note:

- This follows Duffie & Beckman (2013).
- NOCT stands for normal-operation cell temperature and concerns the specifications of the PV module for realistic conditions. If not available, use STC specs everywhere instead.
- Note that all temperatures should be either in °C or K; they should not be mixed.

`solarenergy.solar_panels.`**`pv_efficiency`**(*temp_c*, *eta_c_stc=0.15*, *t_coef=-0.0045*, *temp_c_stc=25*)

Compute the instantaneous PV effiency for a given cell temperature.

We assume that the efficiency varies linearly with temperature.

Parameters

- **temp_c** (*float*) – Temperature of the PV cell (°C or K).
- **eta_c_stc** (*float*) – Efficiency of PV (+inverter if desired) for standard test conditions (STC) (0-1).
- **t_coef** (*float*) – Temperature coefficient for P_mpp and η (/K; <0).
- **temp_c_stc** (*float*) – Temperature of the cells under Standard Test Conditions (°C or K, same as temp_c).

Returns Efficiency of the solar cell.

Return type float

`solarenergy.solar_panels.`**`read_solar_panel_specs`**(*cfg_file='.solar_panels.cfg'*, *rel_to_home=True*, *to_rad=True*)

Read solar-panel specifications from a configuration file.

Parameters

- **cfg_file** (*str*) – Configuration file to read specs from (by default relative to home directory).
- **rel_to_home** (*bool*) – File path/name is relative to home directory.
- **to_rad** (*bool*) – Convert angles from degrees to radians (location, orientation).

Returns Dataclass of type SolarPanels containing the specifications.

Return type (*SolarPanels*)

1.1.2 solarenergy.radiation module

Functions for solar energy dealing with (solar) radiation.

References

- Marc van der Sluys, Celestial mechanics in a nutshell, https://cmians.sourceforge.io (2014-2022).

`solarenergy.radiation.`**`airmass`**(*sun_alt*, *return_value_below_horizon=False*)

Compute airmass as a function of Sun altitude.

Parameters

- **sun_alt** (*float*) – True altitude of the Sun (uncorrected for atmospheric refraction; rad), can be an array.
- **return_value_below_horizon** (*bool*) – Return a very large value when the Sun is below the horizon, larger when the Sun is lower. This can be useful for solvers. Default: False.

Returns Airmass at sea level (AM~1 if the Sun is in the zenith, AM~38 near the horizon).

Return type float

References

A.T. Young, "AIR-MASS AND REFRACTION," Applied Optics, vol. 33, pp. 1108-1110, Feb 1994.

`solarenergy.radiation.`**`clearsky_bird`**(*sun_alt*, *i_ext=1353*, *sun_dist=1*, *press=1013*, *uo=0.34*, *uw=1.42*, *ta5=0.2661*, *ta3=0.3538*, *ba=0.84*, *k1=0.1*, *rg=0.2*)

A simplified clear-sky model for direct and diffuse insolation on horizontal surfaces.

A.k.a. as "the Bird model".

This function is adapted from the libTheSky Fortran implementation (libthesky.sf.net).

See Bird & Hulstrom, A simplified clear-sky model for direct and diffuse insolation on horizontal surfaces, SERI/TR-642-761 (1981).

Note that the value of Taa does not agree with tabulated values from the paper, and hence neither do dependent values (except for AM~1). When I substitute their values for Taa, everything matches perfectly. Error in their formula, or (hopefully!) in their table?

Parameters

- **sun_alt** (*float*) – Sun altitude above the horizon (rad)
- **i_ext** (*float*) – Extraterrestrial radiation (at the top of the atmosphere; AM0; W/m^2 - optional, default: 1353 (1361.5))
- **sun_dist** (*float*) – Sun distance (AU - optional, default: 1)
- **press** (*float*) – Air pressure at the observer's site, corrected for altitude (hPa - optional, default: 1013)

- **uo** (*float*) – Ozone abundance in a vertical column (cm - optional, default: 0.34)
- **uw** (*float*) – Percipitable water-vapor abundance in a vertical column (cm - optional, default: 1.42)
- **ta5** (*float*) – Aerosol optical depth from surface in vertical path at 500 nm (optional, default: 0.2661)
- **ta3** (*float*) – Aerosol optical depth from surface in vertical path at 380 nm (optional, default: 0.3538)
- **ba** (*float*) – Aerosol forward-scattering ratio (optional, 0.82-0.86, default: 0.84)
- **k1** (*float*) – Aerosol-absorptance constant (optional, rural: 0.0933, urban: 0.385, default: 0.1)
- **rg** (*float*) – Ground albedo (optional, fraction - default: 0.2)

Returns

Tuple containing (i_tot, i_dir, i_dif, i_gr):

- i_tot (float): Total (global) insolation on a horizontal surface (GHI; W/m^2)
- i_dir (float): Direct (beam) insolation on a horizontal surface (BHI; W/m^2)
- i_dif (float): Diffuse insolation on a horizontal surface (DHI; W/m^2)
- i_gr (float): Ground-reflection insolation from a horizontal surface (W/m^2)

Return type tuple (float,float,float,float)

`solarenergy.radiation.`**`computeSunPos`**(*geo_lon*, *geo_lat*, *year*, *month*, *day*, *hour*, *minute=0*, *second=0*, *timezone='UTC'*, *debug=False*)

Obsolete version of sun_position_from_date_and_time(). Use that function instead!

`solarenergy.radiation.`**`cosAngleSunPanels`**(*sp_az*, *sp_incl*, *sun_az*, *sun_alt*)

Obsolete version of cos_angle_sun_panels(). Use that function instead!

`solarenergy.radiation.`**`cos_angle_sun_panels`**(*sp_az*, *sp_incl*, *sun_az*, *sun_alt*)

Compute the cosine of the angle between the orientation vector of the solar panels and the position vector of the Sun.

This is the cosine of the angle under which the direct sunlight hits the solar panels. Multiply it with the DNI to obtain the direct insolation on the panels. See Celestial mechanics in a nutshell, Sect. 4.3: Insolation on an inclined surface (http://CMiaNS.sf.net).

Parameters

- **sp_az** (*float*) – Azimuth in which the solar panels are facing (rad; e.g. north or south = 0, same as sun_az).
- **sp_incl** (*float*) – Inclination ('zenith angle') of the solar panels w.r.t. the horizontal (rad).
- **sun_az** (*float*) – Azimuth of the Sun (rad; e.g. north or south = 0, same as sp_az).
- **sun_alt** (*float*) – Altitude of the Sun (rad).

Returns

The cosine between the normal vector of the solar panels and the position vector of the Sun (rad). Note that this value is zero (indicating radiation from behind the panels) or positive.

Return type float

`solarenergy.radiation.`**`diffuseRad_from_globalRad_sunshine`**`(`*glob_horiz*, *sun_frac*, *sun_alt*, *i_ext=1361.5*`)`

Obsolete version of diffuse_radiation_from_global_radiation_and_sunshine(). Use that function instead!

`solarenergy.radiation.`**`diffuse_radiation_from_global_radiation_and_sunshine`**`(`*glob_ho sun_fra sun_alt i_ext=1*

Compute the diffuse horizontal radiation from the global horizontal radiation, the fraction of sunshine and the Sun altitude.

Parameters

- **glob_horiz** (*float*) – Global horizontal radiation (W/m2).
- **sun_frac** (*float*) – Fraction of sunshine (e.g. fraction of cloud cover) (-; 0-1).
- **sun_alt** (*float*) – Sun altitude above the horizon (rad).
- **i_ext** (*float*) – Extraterrestrial radiation (W/m2). Defaults to the solar constant.

Returns

Tuple containing (dif_horiz, beam_horiz, beam_norm):

- dif_horiz (float): Diffuse horizontal radiation = DHI (W/m2).
- beam_horiz (float): Beam (direct) horizontal radiation = BHI (W/m2).
- beam_norm (float): Beam (direct) normal radiation = BNI = DNI (W/m2).

Return type tuple (float,float,float)

`solarenergy.radiation.`**`diffuse_radiation_projection_Perez87`**`(`*doy*, *alt*, *surf_incl*, *theta*, *beam_norm*, *dif_horiz*`)`

Obsolete version of diffuse_radiation_projection_perez87(). Use that function instead!

`solarenergy.radiation.`**`diffuse_radiation_projection_perez87`**`(`*doy*, *sun_alt*, *surf_incl*, *theta*, *beam_norm*, *dif_horiz*, *return_components=False*`)`

Compute diffuse radiation on an inclined surface using the 1987 Perez model

This function is adapted from the libTheSky Fortran implementation (libthesky.sf.net).

See Perez et al. Solar Energy Vol. 39, Nr. 3, p. 221 (1987) - references to equations and tables are to this paper. Most equations can be found in the Nomenclature section at the end of the paper (p.230). I use a and c here, not b and d.

Parameters

- **doy** (*int*) – Day of year (Nday)
- **sun_alt** (*float*) – Altitude of the Sun (radians, may be an array)
- **surf_incl** (*float*) – Surface inclination wrt horizontal (radians) - 0 = horizontal, pi/2 = vertical
- **theta** (*float*) – Angle between surface normal vector and Sun position vector (radians, may be an array)
- **beam_norm** (*float*) – Beam (direct) normal radiation = DNI (W/m2; in the direction of the Sun, may be an array)
- **dif_horiz** (*float*) – Diffuse radiation on a horizontal surface = DHI (W/m2, may be an array)
- **return_components** (*bool*) – Return isotropic, circumsolar and horizon parts separately (i.e., four return values).

Returns

Diffuse irradiation on the inclined surface (W/m2) (may be an array) - if return_components=False

tuple: Diffuse irradiation + components as (float1, float2, float3, float4), may be arrays - if return_components=True:

- float1: Total diffuse irradiation on the inclined surface (W/m2)
- float2: Isotropic diffuse irradiation on the inclined surface (W/m2)
- float3: Circumsolar diffuse irradiation on the inclined surface (W/m2)
- float4: Horizon-band diffuse irradiation on the inclined surface (W/m2)

Return type float

`solarenergy.radiation.`**`extinctionFactor`**(*airmass*, *return_value_below_horizon=False*)

Obsolete version of extinction_factor(). Use that function instead!

`solarenergy.radiation.`**`extinction_factor`**(*airmass*, *return_value_below_horizon=False*)

Compute the atmospheric extinction factor for sunlight from the air mass.

Parameters

- **airmass** (*float*) – Airmass at sea level (AM~1 if the Sun is in the zenith, AM~38 near the horizon), can be an array.
- **return_value_below_horizon** (*bool*) – Return a very large value when the Sun is below the horizon, larger when the Sun is lower. This can be useful for solvers. Default: False.

Returns

The extinciton factor for sunlight in the atmosphere. Divide the extraterrestrial (AM0) radiation (or, if unknown, the solar constant) by this number to obtain the DNI.

Return type float

`solarenergy.radiation.`**`reflectance_transmittance`**(*ang_i*, *n_ref1*, *n_ref2*, *comp_transmittance=False*, *comp_polarised=False*)

Compute the reflectance and transmittance as function of incidence angle and refactive indices.

Compute the reflectance for the transition from a medium with refractive index n_ref1 to one with n_ref2, under an incidence angle ang_i. Optionally, compute the transmittance, and the polarised components. The media are assumed to be non-magnetic.

Parameters

- **ang_i** (*float*) – Angle of incidence (rad).
- **n_ref1** (*float*) – Refractive index of initial medium (-).
- **n_ref2** (*float*) – Refractive index of second medium (-).
- **comp_transmittance** (*bool*) – Compute and return the transmittance and its angle.
- **comp_polarised** (*bool*) – Compute and return the polarised reflectances and transmittances.

Returns

Tuple consisting of one or more values, depending on the input parameters:

- comp_transmittance=False, comp_polarised=False: (r_unp);
- comp_transmittance=False, comp_polarised=True: (r_unp, r_prp,r_par);
- comp_transmittance=True, comp_polarised=False: (r_unp, t_unp, ang_t);
- **comp_transmittance=True, comp_polarised=True: (r_unp, t_unp, ang_t, r_prp,r_par,** t_prp,t_par);

With the following variables: r_unp (float): Unpolarised reflectance (-);

t_unp (float): Unpolarised transmittance (-); ang_t (float): Angle of transmittance (rad);

r_prp (float): Perpendicular polarised reflectance (-); r_par (float): Parallel polarised reflectance (-); t_prp (float): Perpendicular polarised transmittance (-); t_par (float): Parallel polarised transmittance (-).

Return type (tuple)

See:

- libSUFR, optics.f90: libsufr.sf.net
- Hecht, Optics, 3rd Ed. (1998), p.113ff

- https://en.wikipedia.org/wiki/Fresnel_equations#Power_or_intensity_equations

`solarenergy.radiation.`**`solar_power_from_clear_sky`**(*sp*, *dat*, *warn=True*)

Model to compute the electrical power for a given solar-panel system and Sun position(s) for a clear sky.

Parameters

- **sp** (*se.SolarPanels*) – struct containing solar-panel data, including the elements: - sp.az (float): azimuth (rad; same as df['sun_az'] (default S=0, W=pi/2)); - sp.incl (float): inclunation (rad; horizontal=0, vertical=pi/2); - sp.eff0 (float): original PV+inverter efficiency at determined T (e.g. 20°C) (fraction; 0-1); - sp.year (int): year of installation (defaults to 2015); - sp.deff_dt (float): change in PV efficiency (dη/dt; year^-1, <0; defaults to 0); - sp.t_coef (float): PV temperature coefficient (K^-1; defaults to 0); - sp.n_refr (float): panel refractive index (>1; defaults to 1.43); - sp.area (float): PV area (m^2); - sp.p_max (float): maximum power of solar-panel system (W).
- **dat** (*pd.DataFrame*) –

  Pandas DataFrame containing solar-panel and weather data, including the columns:

  - df['dtm'] (datetime): Date and time;
  - df['sun_az'] (float): Sun azimuth (rad; same as sp.az (default S=0, W=pi/2));
  - df['sun_alt'] (float): Sun altitude (rad);
  - df['sun_dist'] (float): Sun distance (AU; defaults to 1);
  - df['press'] (float): air pressure (mbar; defaults to 1010);
  - df['temp'] (float): ambient air temperature (°C; defaults to 15);
  - df['ws'] (float): wind speed (m/s; defaults to 3).

  More columns with intermediate results will be added during the calculation. The final result will be added as a column named 'Pclrsky', as well as returned as an array.
- **warn** (*bool*) – Warn if a parameter or variable is missing and the default value is used (defaults to True).

Returns

Array containing predicted electrical power of solar panels for a clear sky (kW). Note that this result is ALSO added to the input DataFrame, as a column named 'Pclrsky'.

Return type float

`solarenergy.radiation.`**`sun_position_from_date_and_time`**`(`*geo_lon, geo_lat, year, month, day, hour, minute=0, second=0, timezone='UTC', debug=False*`)`

Compute the Sun local position (azimuth, altitude and distance) for the given geographical location and date and time (ymd, hms) using SolTrack.

Parameters

- **geo_lon** (*float*) – Geographic longitude to compute the Sun position for (rad).
- **geo_lat** (*float*) – Geographic latitude to compute the Sun position for (rad).
- **year** (*int*) – Year (CE) to compute the Sun position for.
- **month** (*int*) – Month to compute the Sun position for.
- **day** (*int*) – Day of month to compute the Sun position for.
- **hour** (*int*) – Hour of day to compute the Sun position for (local time!).
- **minute** (*int*) – Minute to compute the Sun position for (optional; default = 0).
- **second** (*int*) – Second to compute the Sun position for (optional; default = 0).
- **timezone** (*str*) – Timezone for which date and time are provided (optional; default = 'UTC').
- **debug** (*bool*) – Switch to write detailed output to stdout (optional; default = False).

Returns

Tuple containing (azimuth, altitude, distance):

- azimuth (float): Azimuth of the Sun (rad; south = 0 on the northern hemisphere).
- altitude (float): Altitude of the Sun (rad).
- distance (float): Distance Sun-Earth (AU).

Return type tuple (float,float,float)

`solarenergy.radiation.`**`sun_position_from_datetime`**`(`*geo_lon, geo_lat, date_time, utc=False, debug=False*`)`

Compute the Sun local position (azimuth, altitude and distance) for the given geographical location and a Python datetime using SolTrack.

Parameters

- **geo_lon** (*float*) – Geographic longitude to compute the Sun position for (rad).
- **geo_lat** (*float*) – Geographic latitude to compute the Sun position for (rad).
- **date_time** (*datetime*) – Date and time to compute the Sun position for (timezone aware and/or UTC, i.e. must be UTC if timezone naive. CHECK: must be UTC if a list or (numpy) array?).

- **utc** (*bool*) – Specify that the input is in UTC. This can speed up calls with a single datetime (as opposed to arrays). Defaults to False.
- **debug** (*bool*) – Switch to write detailed output to stdout (optional; default = False).

Returns

Tuple containing (arrays of) (azimuth, altitude, distance):

- azimuth (float): Azimuth of the Sun (rad; south = 0 on the northern hemisphere).
- altitude (float): Altitude of the Sun (rad).
- distance (float): Distance Sun-Earth (AU).

Return type tuple (float,float,float)

1.2 Module contents

SolarEnergy module

SolarEnergy contains a Python module to do simple modelling in the field of solar energy. The code is being developed by Marc van der Sluys of the department of Astrophysics at the Radboud University Nijmegen, the Netherlands and the department of Sustainable energy of the HAN University of Applied Sciences in Arnhem, the Netherlands. SolarEnergy can be used under the conditions of the EUPL 1.2 licence. These pages contain the API documentation. For more information on the Python package, licence, source code and data files, see the SolarEnergy GitHub page.

The SolarEnergy code is based on the libTheSky Fortran library. Information on the theory behind this code can be found in the document Celestial mechanics in a nutshell.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

s

Index

T

Y