
solsticepy Documentation

Release 0.1.9

Ye Wang, John Pye

Mar 05, 2023

Contents:

1	Create YAML input files for Solstice	3
2	Calculate sun position	5
3	Set up and run Solstice simulations	9
4	Process the results	11
5	Generate new heliostat field layouts	13
6	Preliminary calculation of heliostat field performance	15
7	Generate 3D views of a heliostat field	19
8	Find Solstice programs in Windows	21
9	Indices and tables	23
	Index	25

solsticepy provides a set of Python functions that make the task of setting up and running a Solstice ray-tracing simulation of a CSP system a little easier. At this stage it has been primarily used for simulations of central-tower CSP systems, even though Solstice itself is capable of simulating a wider range of system types.

Create YAML input files for Solstice

`solsticepy.gen_yaml` (*sun, hst_pos, hst_foc, hst_aims, hst_w, hst_h, rho_refl, slope_error, receiver, rec_param, rec_abs, outfile_yaml, outfile_recv, hemisphere='North', tower_h=0.01, tower_r=0.01, spectral=False, medium=0, one_heliostat=False*)

Generate the heliostat field and receiver YAML input files for Solstice ray-tracing simulation.

1. the sun

- *sun* (*Sun* object): parameters relating to the solar source

2. the field

- *hst_pos* (nx3 numpy array): heliostat positions (x, y, z) (first of the ‘field’ parameters)
- *hst_foc* (nx1 numpy array): heliostat focal length
- *hst_aims* (nx3 numpy array): heliostat aiming point (ax, ay, az)
- *hst_w* (float): heliostat mirror width (in x direction)
- *hst_h* (float): heliostat mirror height (in y direction)
- *hst_z* (float): heliostat center height (in z direction)
- *rho_refl* (float): reflector reflectivity
- *slope_error* (float): reflector surface slope error rms, radians
- *tower_h* (float): tower height (m)
- *tower_r* (float): tower radius (a cylindrical shape tower) (m)

3. the receiver

- *receiver* (str): 'flat', 'cylinder', or 'stl' or 'multi-aperture' (first of the ‘receiver’ parameters)
- *rec_abs* (float): receiver absorptivity

- *rec_param* (numpy array or str): each element contains the geometrical parameter of the corresponding receiver.

4. others

- *spectral* (bool): True - simulate the spectral dependent performance (first of the 'other' parameters)
- *medium* (float): if the atmosphere is surrounded by non-participant medium, medium=0; otherwise it is the extinction coefficient in m-1
- *one_heliosat* (boolean): if *True*, implements ray tracing from just one heliostat.

Returns: nothing (requested files are created and written)

Note that the parameters are in groups that relate to the *sun*, the *field* and the *receiver* then *others*.

Note also the type for *rec_param* should be as follows.

- if *receiver* == 'flat': np.array([width, height, grid_w, grid_h,, x, y, z, tilt angle (deg)])
- if *receiver* == 'cylinder': np.array([radius, height, grid_circ, grid_h, x, y, z, tilt angle (deg)])
- if *receiver* == 'stl': the directory of the stl file
- if *receiver* == 'multi-aperture': np.array([width, height, grid_w, grid_h,, x, y, z, tilt angle (deg), num_aperture, gamma (deg)])

Calculate sun position

class solsticepy.SunPosition

Calculate sun position according to a certain location, date and time. Reference: John A. Duffie and William A. Beckman. Solar Engineering of Thermal Processes, 4th edition.

Example

- Location: latitude = 37.44
- Date : 21 Jun
- Time : solar noon or two hours after sunrise

```
>>> from solsticepy.cal_sun import *
>>> sun=SunPosition()
>>> day=sun.days(21, 'Jun')
>>> latitude=37.44
>>> delta=sun.declination(day)
```

Solar noon

```
>>> omega=0.
>>> theta=sun.zenith(latitude, delta, omega)
>>> phi=sun.azimuth(latitude, theta, delta, omega)
>>> print(theta)
13.987953925483858
>>> print(phi)
0.0
```

Two hours after sunrise

```
>>> daytime, sunrise=sun.solarhour(delta, latitude)
>>> omega=sunrise+15.*2. # solar noon
>>> theta=sun.zenith(latitude, delta, omega)
>>> phi=sun.azimuth(latitude, theta, delta, omega)
>>> print(theta)
```

(continues on next page)

(continued from previous page)

```
67.91774797592434
>>> print(phi)
-103.31434583806747
```

annual_angles (*latitude*, *casefolder=None*, *nd=5*, *nh=5*, *verbose=False*)

Generate a range of sun positions (azimuth-zenith angles and declination-solarhour angles) for annual optical lookup table simulation. Automatically detect the time when the sun is below the horizon (elevation<0), where a ray-tracing simulation is not required.

Arguments

- *latitude* (float): latitude of the location (deg)
- *casefolder* (str): directory to save the table and *case_list* in .csv files, or 'NOTSAVE' (by default) to not write the output to files
- *nd* (int): number of rows of the lookup table (points in the declination movement, suggest *nd*>=5)
- *nh* (int): number of columns of the lookup table (hours in a day, i.e. 24h)
- *verbose* (bool): write results to disk or not

Returns

- *AZI* (1 D numpy array): a list of azimuth angles (deg), counted from South towards to West
- *ZENITH* (1D numpy array): a list of zenith angles (deg)
- *table* (numpy array): declination (row) - solarhour (column) lookup table to be simulated
- *case_list* (numpy array): a flatten list of cases to be simulated, with the correspondence between declination-solarhour angles and azimuth-zenith angles for each case

Example

```
>>> from solsticepy.cal_sun import *
>>> sun=SunPosition()
>>> latitude=37.44
>>> casefolder='.'
>>> nd=5
>>> nh=9
>>> sun.annual_angles(latitude, casefolder, nd, nh)
```

A 5x9 lookup table will be generated to the current directory:

Lookup table			Solar hour angles (deg)								
			1	2	3	4	5	6	7	8	9
Declination angles (deg)			-180	-135	-90	-45	0	45	90	135	180
	1	-23.45	-	-	-	case 1	case 2	***1	-	-	-
	2	-11.73	-	-	-	case 3	case 4	***3	-	-	-
	3	0	-	-	case 5	case 6	case 7	***6	***5	-	-
	4	11.73	-	-	case 8	case 9	case 10	***9	***8	-	-
	5	23.45	-	-	case 11	case 12	case 13	***12	***11	-	-

- case n : this sun position is the n -th case to be simulated
- ***** n** : it is the symmetric case with the case n , no re-simulation is required
- - : the sun is below the horizon, no simulation is required

azimuth (*latitude, theta, delta, omega*)

Calculate azimuth angle at specified location and sun position, ref. J Duffie eq. 1.6.6

Arguments

- latitude (float): latitude angle (deg)
- delta (float): declination angle (deg)
- theta (float): zenith angle (deg)
- omega (float): solar hour angle (deg)

Return

- phi (float): azimuth angle (deg), counted from South towards to West

convert_AZEL_to_declination_hour (*theta, phi, latitude*)

Convert azimuth-elevation angle to declination-hour angle

Arguments

- theta (float): zenith angle (deg)
- phi (float): azimuth angle (deg), counted from South towards to West
- latitude (float): latitude latitude (deg)

Returns

- delta: declination angle (deg)
- omega: solar hour angle (deg)

convert_convention (*tool, azimuth, zenith*)

Return azimuth-elevation angle using the angle convention of Solstice or SolarTherm

Arguments

- tool (str): 'solstice' or 'solartherm'
- azimuth (float): azimuth angle (deg), counted from South towards to West
- zenith (float): zenith angle (deg)

Returns

- sol_azi (float): azimuth angle
- sol_ele (float): elevation angle

days (*dd, mm*)

Calculate the day-of-the-year for specified date and month, ref. J Duffie page 14, Table 1.6.1

Arguments

- dd (int): the day in the month
- mm (str): the month, e.g. 'Jan', 'Feb', 'Mar' etc

Return

- days (int): the-day-of-the year for the specified dd-mm, e.g. 1 Jan is day 1

declination (*days, form=None*)

Calculate the solar declination angle for a specified day-of-the-year, ref. J Duffie page 13, declination angle: $\delta = 23.45 \cdot \sin(360 \cdot (284 + \text{day}) / 365)$

Arguments

- day (int): day of the year, i.e from 1 to 365
- form (str): 'detail' or simple' model

Return

- delta (float): declination angle (deg)

solarhour (*delta, latitude*)

Calculate day length and sunrise hour-angle, ref. J Duffie page 17

Arguments

- delta (float): declination angle (deg)
- latitude (float): latitude angle (deg)

Returns

- hour (float): length of the daylight hour (h)
- sunrise (float): the solar hour angle at sunrise (deg)

zenith (*latitude, delta, omega*)

Calculate the zenith angle, ref. J Duffie eq.1.6.5

Arguments

- latitude (float): latitude angle at the location (deg)
- delta (float): declination angle (deg)
- omega (float): solar hour angle (deg)

Return

- theta (float): the zenith angle (deg)

Set up and run Solstice simulations

class `solsticepy.Master` (*casedir*='', *nproc*=None)

Set up the Solstice simulation, i.e. establishing the case folder, calling the Solstice program and post-processing the results

Argument

- *casedir* (str): the case directory
- ***nproc* (int): number of processors, e.g. *nproc*=1 will run in serial mode, *nproc*=4 will run with 4 processors in parallel** *nproc*=None will run with any number of processors that are available

in_case (*folder*, *fn*)

Joining a file name with the case directory

Argument

- *fn* (str): file name

Return

- a joining directory of the file in the case directory

run (*azimuth*, *elevation*, *num_rays*, *rho_mirror*, *dni*, *folder*, *gen_vtk*=False, *printresult*=False, *verbose*=False, *system*='crs')

Run an optical simulation (one sun position) using Solstice

- *azimuth* (float): the azimuth angle of the ray-tracing simulation in Solstice, counted from East towards to North
- *elevation* (float): the elevation angle of the ray-tracing simulation in Solstice
- *num_rays* (int): number of rays to be cast in the ray-tracing simulation
- *rho_mirror* (float): reflectivity of mirrors, required for results post-processing
- *dni* (float): the direct normal irradiance (W/m²), required to obtain performance of individual heliostat

- *gen_vtk* (boolean): if True, generate .vtk files for rendering in Paraview
- *system* (str): 'crs' for a central receiver system, or 'dish' for a parabolic dish system

Returns: no return value (results files are created and written)

run_annual (*nd, nh, latitude, num_rays, num_hst, rho_mirror, dni, gen_vtk=False, verbose=False*)

Run a list of optical simulations to obtain annual performance (lookup table) using Solstice

Arguments

- *nd* (int): number of rows in the lookup table (discretisation of the declination angle)
- *nh* (int): number of columns in the lookup table (discretisation of the solar hour angle)
- *latitude* (float): the latitude angle of the plan location (deg)
- *num_rays* (int): number of rays to be cast in the ray-tracing simulation
- *num_hst* (int): number of heliostats
- **nproc (int): number of processors, e.g. nproc=1 will run in serial mode,**
 - nproc=4 will run with 4 processors in parallel** nproc=None will run with any number of processors that are available
 - *rho_mirror* (float): reflectivity of mirrors, required for results post-processing
 - *dni* (float): the direct normal irradiance (W/m²), required to obtain performance of individual heliostat
 - *gen_vtk* (bool): True - perform postprocessing for visualisation of each individual ray-tracing scene (each sun position), False - no postprocessing for visualisation

Return

- *table* (numpy array), the annual optical efficiency lookup table
- *ANNUAL* (numpy array), the annual output of each heliostat

Process the results

`solsticepy.process_raw_results` (*rawfile*, *savedir*, *rho_mirror*, *dni*, *verbose=False*)

Process the raw Solstice *simul* output into readable CSV files for central receiver systems

Arguments

- *rawfile* (str): the directory of the *simul* file that generated by Solstice
- *savedir* (str): the directory for saving the organised results
- *rho_mirror* (float): mirror reflectivity (needed for reporting energy sums)
- *dni* (float): the direct normal irradiance (W/m²), required to obtain performance of individual heliostat
- *verbose* (bool), write results to disk or not

Returns

- *efficiency_total* (float): the total optical efficiency
- *performance_hst* (numpy array): the breakdown of losses of each individual heliostat
- The simulation results are created and written in the *savedir*

`solsticepy.get_breakdown` (*casedir*)

Postprocess the .csv output files (heliostats-raw.csv, before trimming), to obtain the breakdown of total energy losses of the designed field (after trimming) for central receiver systems

Argument

- *casedir* (str): the directory of the case that contains the folder of *sunpos_1*, *sunpos_2*, ..., and all the other case-related details
- *verbose* (bool), write results to disk or not

Outputs

- output file: *OELT_Solstice_breakdown.motab*, it contains the annual lookup tables of each breakdown of energy

- output files: result-formatted-designed.csv file in each sunpos folder, each of them is a list of the breakdown of energy at this sun position

`solsticepy.process_raw_results_dish(rawfile, savedir, rho_mirror, dni, verbose=False)`

Process the raw Solstice *simul* output into readable CSV files for dish systems

Arguments

- `rawfile` (str): the directory of the *simul* file that generated by Solstice
- `savedir` (str): the directory for saving the organised results
- `rho_mirror` (float): mirror reflectivity (needed for reporting energy sums)
- `dni` (float): the direct normal irradiance (W/m²), required to obtain performance of individual heliostat

Returns

- `efficiency_total` (float): the total optical efficiency
- The simulation results are created and written in the *savedir*

Generate new heliostat field layouts

`solsticepy.radial_stagger` (*latitude, num_hst, width, height, hst_z, towerheight, R1, fb, dsep=0.0, field='polar', num_aperture=0, gamma=0.0, rec_w=0.0, rec_z=[], savedir='.', verbose=False, plot=False, plt_aiming=None*)

Generate a radial-stagger heliostat field, ref. Collado and Guallar, 2012, Campo: Generation of regular heliostat field.

Arguments

- `latitude` (float): latitude of the field location (deg)
- `num_hst` (int) : number of heliostats
- `width` (float) : mirror width (m)
- `height` (float) : mirror height (m)
- `hst_z` (float) : the vertical location of each heliostat (m)
- `towerheight` (float): tower height (m)
- `R1` (float) : distance from the first row to the bottom of the tower, i.e. (0, 0, 0)
- `fb` (float) : the field layout growing factor, in (0, 1)
- `dsep` (float) : separation distance (m)
- `field` (str) : 'polar-half' or 'surround-half' or 'polar' or 'surround' field or 'multi-aperture', the 'half' option is for simulation a symmetric field
- `num_aperture`(int): number of apertures, for a multi-aperture configuration
- `gamma` (float) : the anangular range of the multi-aperture configuration (deg)
- `rec_z` (list) : a list of the elevation heights of the apertures
- `savedir` (str) : directory of saving the `pos_and_aiming.csv`
- `verbose`(bool) : write results to disk or not
- `plot` (bool) : True - plot the layout by Matplotlib

Returns

- pos_and_aiming (nx7 numpy array): position, focal length and aiming point of each generated heliostat
- a pos_and_aiming.csv file is created and written to the savedir

Example

```
>>> from solsticepy.cal_layout import *
>>> latitude=34.
>>> hst_width=13.
>>> hst_height=10.
>>> hst_z=5.
>>> target_area=75715.
>>> num_hst=target_area/hst_width/hst_height*2 # create a twice large_
↳field, so that the inefficient heliostat can be trimmed-off after_
↳optical simulations
>>> tower_height=110.
>>> R1=40. # the distance from the first row to the bottom of the tower
>>> fb=0.6
>>> casefolder='.' # replace it as the directory of your case folder
>>> plot=False # if you want to plot the field layout, set it as True
>>> pos_and_aim=radial_stagger(latitude, num_hst, hst_width, hst_height,
↳hst_z, tower_height, R1, fb, savedir=casefolder,plot=plot)
```

An nx7 array is returned, and the pos_and_aim.csv file is saved in the local directory

Preliminary calculation of heliostat field performance

class solsticepy.**FieldPF**(receiver_norm=array([0, 1, 0]))

Preliminary calculation of heliostat field performance.

1. cosine factors: sun - heliostat
2. another cosine factors: receiver - heliostat

Note the angle conventions in this program:

- azimuth: the solar azimuth angle, from South to West
- zenith: the solar zenith angle, 0 from vertical

Example

```
>>> from solsticepy.cal_field import *
>>> pos_and_aiming=np.loadtxt('./pos_and_aiming.csv', delimiter=',',
↳skiprows=2) #load the field layout file (refer to cal_layout.py)
>>> pos=pos_and_aiming[:, :3]
>>> aim=pos_and_aiming[:, 4:]
>>> azimuth=np.r_[0.]
>>> zenith=np.r_[12.]
>>> field=FieldPF(np.r_[0,1,0])
>>> sun_vec=field.get_solar_vector(azimuth, zenith)
>>> norms=field.get_normals(towerheight=70., hstpos=pos, sun_vec=sun_vec)
>>> COORD, TRI, ele, nc=field.mesh_heliostat_field(width=10., height=8.,
↳normals=norms, hstpos=pos)
>>> cos=field.get_cosine(hst_norms=norms, sun_vec=sun_vec)
>>> savedir='./field.vtk'
>>> COS=np.repeat(cos, ele)
>>> DATA={'cos':COS}
>>> NORMS=np.repeat(norms, ele, axis=0)
>>> gen_vtk(savedir, COORD.T, TRI, NORMS, True, DATA)
```

The field.vtk file is saved in the local directory and can be open in ParaView to visualise the cosine factor of each individual heliostat. See more details in the 'gen_vtk.py' in the next section.

Argument

- **receiver_normal** (numpy array): A numpy 3-vector with unit normal direction of the receiver aperture (default: [0,1,0])

get_cosine (*hst_norms, sun_vec*)

Calculate the cosine factor between the sun and the heliostats

Arguments

- **hst_norms** (numpy array): normal vectors of the heliostats
- **sun_vec** (numpy array): solar vector

Return

- **cos_factor** (numpy array): the cosine factor between the sun and the heliostats

get_normals (*towerheight, hstpos, sun_vec*)

Calculate the normal vectors of each heliostat

Arguments

- **towerheight** (float): tower height
- **hstpos** (nx3 numpy array): heliostat positions
- **sun_vec** (numpy array): the solar vector

Return

- **hst_norms** (numpy array): normal vectors of each heliostat

get_rec_view (*towerheight, hstpos*)

Check the visibility of each heliostat from the view of the receiver

Arguments

- **towerheight** (float): tower height
- **hstpos** (numpy array): position of each heliostat

Return

- **vis_idx** (numpy array): the indices of the heliostats that can be seen by the receiver

get_solar_vector (*azimuth, zenith*)

Calculate the solar vector using azimuth and zenith angles.

Arguments

- **azimuth** (float): the sun's azimuth (deg), from South increasing towards to the West
- **zenith** (float): angle created between the solar vector and the Z axis (deg)

Return

- **sun_vec** (numpy array): a 3-component 1D array with the solar vector

mesh_heliostat (*width, height*)

The local coordinate of the triangular mesh of a heliostat

Arguments

- **width** (float): width of the heliostat
- **height** (float): height of the heliostat

Returns

- `coords` (numpy array): coordinates of the vertices
- `tri` (numpy array): the indices of the triangular mesh

mesh_heliostat_field (*width, height, normals, hstpos*)

Generate the necessary elements to create the VTK file to view the heliostat layout

Arguments

- `width` (float): width of the heliostat
- `height` (float): height of the heliostat
- `normals` (numpy array): normal vectors of the heliostats
- `hstpos` (numpy array): the positions of the heliostats

Returns

- `COORD` (numpy array): the coordinates of the indices of the heliostat field
- `TRI` (numpy array): the indices of the triangular mesh of the heliostat field
- `ele` (int): number of element of the triangular mesh
- `nc` (int): number of indices

plot_cosine (*savename*)

Plot the cosine factors of heliostats in Matplotlib

Argument

- `savename` (str): the directory to save the figure, with suffix, e.g. `'png'` or `'jpg'`

Return

- No return value (a figure is written in the specified path)

plot_select (*savename, tilt*)

Plot the heliostats that can be seen by the receiver in Matplotlib

Argument

- `savename` (str): the directory to save the figure, with suffix, e.g. `'png'` or `'jpg'`
- `savename` (float): receiver tilted angle

Return

- No return value (a figure is written in the specified path)

Generate 3D views of a heliostat field

`solsticepy.gen_vtk` (*savedir, points, indices, norms, colormap=True, DATA=None*)

Generate 3D views of a heliostat field with triangular mesh in the VTK format that can be visualised in ParaView software

Arguments

- `savedir` (str): directory to save the VTK file
- `points` (nx3 numpy array): the vertices of the objects, each column of the array is X, Y, Z coordinates respectively
- `indices` (nx3 numpy array): the indices of the triangular mesh
- `norms` (nx3 numpy array): the normal vectors of the triangular mesh
- `colormap` (bool): True - show the data of each heliostat (e.g. cosine factor), False - not show the data results, but only show the geometry of the heliostats
- `DATA` (dic): key is 'cosine' or 'atm' or others performance parameter to be visualised

Return

- No return value (a VTK file is created and written in the *savedir*)

Find Solstice programs in Windows

`solsticepy.find_solstice_root` (*version_required=None, verbose=0*)

Locate the place where Solstice files are installed on Windows

Arguments

- `version_required` (None or str): if not None, then enforce this specified version of Solstice (eg '0.9.0')
- `verbose` (bool): if True, output file locations to stderr

Return

- `dirn` (str): the directory where Solstice is installed in the Windows system

`solsticepy.find_prog` (*name, version_required=None*)

Find the path to any required Solstice executable program

Arguments

- `name` (str): stem-name (eg 'solpp') of the program (eg 'solpp.exe') required
- `version_required` (None or str): if not None, then enforce this specified version of Solstice (e.g. '0.9.0')

Return

- `path` (str): path of the required Solstice executable program

CHAPTER 9

Indices and tables

- `genindex`
- `search`

A

`annual_angles()` (*solsticepy.SunPosition method*), 6
`azimuth()` (*solsticepy.SunPosition method*), 7

C

`convert_AZEL_to_declination_hour()` (*solsticepy.SunPosition method*), 7
`convert_convention()` (*solsticepy.SunPosition method*), 7

D

`days()` (*solsticepy.SunPosition method*), 7
`declination()` (*solsticepy.SunPosition method*), 7

F

`FieldPF` (*class in solsticepy*), 15
`find_prog()` (*in module solsticepy*), 21
`find_solstice_root()` (*in module solsticepy*), 21

G

`gen_vtk()` (*in module solsticepy*), 19
`gen_yaml()` (*in module solsticepy*), 3
`get_breakdown()` (*in module solsticepy*), 11
`get_cosine()` (*solsticepy.FieldPF method*), 16
`get_normals()` (*solsticepy.FieldPF method*), 16
`get_rec_view()` (*solsticepy.FieldPF method*), 16
`get_solar_vector()` (*solsticepy.FieldPF method*), 16

I

`in_case()` (*solsticepy.Master method*), 9

M

`Master` (*class in solsticepy*), 9
`mesh_heliostat()` (*solsticepy.FieldPF method*), 16
`mesh_heliostat_field()` (*solsticepy.FieldPF method*), 17

P

`plot_cosine()` (*solsticepy.FieldPF method*), 17
`plot_select()` (*solsticepy.FieldPF method*), 17
`process_raw_results()` (*in module solsticepy*), 11
`process_raw_results_dish()` (*in module solsticepy*), 12

R

`radial_stagger()` (*in module solsticepy*), 13
`run()` (*solsticepy.Master method*), 9
`run_annual()` (*solsticepy.Master method*), 10

S

`solarhour()` (*solsticepy.SunPosition method*), 8
`SunPosition` (*class in solsticepy*), 5

Z

`zenith()` (*solsticepy.SunPosition method*), 8