

# **Python Solar Energy Calculation Primer**

**Gianguido Piani**

## Python Solar Energy Calculation Primer

copyright by Gianguido Piani, 2017

contact information: <solarprimer@mailbox.org>



This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-sa/4.0/> or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

## Table of Contents

List of Program Code.....	5
List of Online Resources.....	6
Foreword.....	7
Main Reference Sites, Software.....	9
'solprimer' Package Download and Installation.....	11
1 The Solar Radiation Resource.....	17
1.1 Solar extraterrestrial spectrum.....	17
1.1.1 Main quantitative aspects of solar radiation.....	17
1.1.2 Planck's blackbody spectrum.....	21
1.2 Local and solar time.....	24
1.2.1 Time representation as Python class.....	24
1.2.2 Tracking solar time and position.....	25
1.2.3 UTC, local, and solar time.....	28
1.2.4 Equation of time.....	30
1.2.5 Conversion between local time and true solar time.....	32
1.2.6 Hour angle.....	35
1.2.7 Solar declination.....	36
1.2.8 Sunrise and sunset hour angle.....	39
1.3 Solar position in the sky.....	42
1.3.1 The celestial sphere, coordinate systems.....	42
1.3.2 Calculation of solar azimuth and elevation angle.....	43
1.3.3 Solar position graphical plots.....	46
1.3.4 High-precision algorithms for the solar position calculation.....	55
1.3.5 Error margin in solar time and position calculation.....	56
1.4 Solar radiation on the earth surface.....	58
1.4.1 Influence of the atmosphere on solar radiation.....	58
1.4.2 Global, direct, and diffuse irradiation.....	62
1.4.3 Clear-sky radiation.....	64
1.5 Flat panels with arbitrary orientation.....	67
1.5.1 Panel orientation and angle of incidence.....	67
1.5.2 Energy collected by a tilted surface.....	70
2 Solar and Climate Data Representation.....	73
2.1 Solar resource maps.....	73
2.2 The Typical Meteorological Year (TMY).....	73
2.2.1 General information about the TMY.....	73
2.2.2 TMY data format types.....	75
2.2.3 Time handling in TMY datasets.....	76
2.2.4 EPW format and conversion to TMY.....	83
2.2.5 TMY series generation via the EU PVGIS Website.....	87
2.2.6 Aggregated and statistical data from TMY datasets.....	90
2.3 TMY graphical data representation.....	101
2.3.1 Parameter plot.....	101
2.3.2 Barchart representation.....	111
2.3.3 Heat map.....	113
2.4 Solar energy collection over tilted surfaces.....	118
2.5 TMY data representation on the psychrometric chart.....	126
2.5.1 Moist air and the psychrometric chart.....	126
2.5.2 Basic psychrometric relations.....	127
2.5.3 Psychrometric chart representation.....	130
2.5.4 Ambient conditions and building thermal analysis.....	132

3 PV Technology.....	138
3.1 Solar modules.....	138
3.1.1 Basic technical parameters.....	138
3.1.2 Solar module testing.....	139
3.2 Solar component databases.....	142
3.2.1 Access to datasets in csv format via pandas dataframes.....	142
3.2.2 PV components datasets.....	149
3.2.3 CEC PV module database.....	151
3.2.4 Sandia PV module database.....	154
3.2.5 CEC Inverter dataset.....	156
4 PV system operation.....	157
4.1 PV solar generator without storage.....	157
4.2 Load analysis.....	164
4.3 PV solar generator with storage and load.....	169
Appendix. Online Sources for Solar Irradiation Data.....	172

## List of Program Code

Program_Code 1-1: script, plot solar extraterrestrial spectrum from tabulated data.....	19
Program_Code 1-2: function, AM0 radiation distribution profile.....	20
Program_Code 1-3: function, AM1.5 radiation distribution profile.....	20
Program_Code 1-4: function, Planck energy density profile.....	22
Program_Code 1-5: script, plot Planck's blackbody spectrum over AM0, AM1.5.....	23
Program_Code 1-6: script, Python datetime class examples.....	24
Program_Code 1-7: function, day of year.....	27
Program_Code 1-8: function, timezone offset.....	28
Program_Code 1-9: function, equation of time.....	31
Program_Code 1-10: function, true solar time from local time.....	33
Program_Code 1-11: function, local time from true solar time.....	34
Program_Code 1-12: function, represent decimal hours in HH:MM format.....	35
Program_Code 1-13: function, hour angle from time in decimal format.....	36
Program_Code 1-14: function, hour angle from time in datetime format.....	36
Program_Code 1-15: function, solar declination.....	38
Program_Code 1-16: function, sunrise and sunset hour interval.....	40
Program_Code 1-17: script, table with sunrise, sunset times at a given location.....	41
Program_Code 1-18: function, solar azimuth and elevation.....	45
Program_Code 1-19: script, Cartesian plot of solar azimuth, elevation for given latitude, day.....	46
Program_Code 1-20: script, Cartesian plot of solar time, elevation at latitude, day.....	48
Program_Code 1-21: script, Cartesian plot of solar azimuth, elevation at given latitude and different months of the year.....	50
Program_Code 1-22: script, solar path polar plot at arbitrary location and day.....	52
Program_Code 1-23: script, solar path polar plot at given latitude, different months of the year.....	53
Program_Code 1-24: script, compare azimuth, elevation angles with the NREL Solar Position Algorithm.....	55
Program_Code 1-25: function, clear-sky radiation at latitude, height.....	64
Program_Code 1-26: script, generate table and plot clear-sky radiation at latitude, height.....	66
Program_Code 1-27: function, angle of incidence between solar beam and an oriented flat surface.....	69
Program_Code 1-28: script, plot clear-sky radiation collected by a tilted surface.....	71
Program_Code 2-1: function, read CSV file, return TMY data as pandas dataframe.....	80
Program_Code 2-2: function, convert TYA file to CSV TMY3 format.....	81
Program_Code 2-3: script, convert TYA file to CSV TMY3 format.....	83
Program_Code 2-4: function, convert EPW file to CSV TMY3 format.....	84
Program_Code 2-5: script, convert EPW file to CSV TMY3 format.....	85
Program_Code 2-6: function, convert PVGIS file to CSV TMY3 format.....	88
Program_Code 2-7: script, convert PVGIS file to CSV TMY3 format.....	89
Program_Code 2-8: function, daily average of TMY data.....	90
Program_Code 2-9: function, daily total of TMY data.....	91
Program_Code 2-10: function, monthly total of TMY data.....	92
Program_Code 2-11: function, heating and cooling degree days from TMY.....	93
Program_Code 2-12: function, statistical values for TMY irradiation data.....	95
Program_Code 2-13: function, number of days with total GHI below a preset threshold.....	96
Program_Code 2-14: script, read CSV TMY file, show aggregated parameters.....	97
Program_Code 2-15: function, estimated noon time from GHI.....	100
Program_Code 2-16: function, plot hourly and daily data.....	101
Program_Code 2-17: script, plot selected variable from TMY file.....	101
Program_Code 2-18: function, plot hourly and daily data, multiple.....	104

Program_Code 2-19: script, plot multiple variables from TMY file.....	104
Program_Code 2-20: script, plot tmy data vs clear-sky radiation.....	106
Program_Code 2-21: script, plot noontime from GHI (equation of time).....	108
Program_Code 2-22: function, barchart for monthly data.....	111
Program_Code 2-23: script, monthly totals from TMY file, draw barchart.....	111
Program_Code 2-24: function, heatmap from TMY data.....	113
Program_Code 2-25: script, plot arbitrary TMY parameter as heat map.....	113
Program_Code 2-26: function, Plane-of-Array (POA) energy from TMY data.....	119
Program_Code 2-27: script, POA solar energy collection, graph.....	120
Program_Code 2-28: script, POA result verification.....	122
Program_Code 2-29: script, POA solar energy collection, barchart.....	123
Program_Code 2-30: script, POA solar energy collection, heatmap.....	124
Program_Code 2-31: script, TMY psychrometric chart data representation.....	130
Program_Code 3-1: function, read dataset in CSV format, return pandas dataframe.....	148
Program_Code 3-2: script, read dataset in CSV format, return pandas dataframe.....	149
Program_Code 3-3: script, read component database as CSV dataset, return index list....	150
Program_Code 3-4: script, read CEC PV module dataset as CSV, return module data.....	153
Program_Code 3-5: script, read CEC PV module dataset, add columns, sort data.....	153
Program_Code 3-6: script, read Sandia PV module dataset as CSV, return module data....	155
Program_Code 3-7: script, read CEC inverter dataset as CSV, return inverter data.....	156
Program_Code 4-1: script, PV solar generator simulation without storage.....	157
Program_Code 4-2: script, load profile generation.....	166
Program_Code 4-3: script, PV solar system simulation with load and storage.....	170

## List of Online Resources

Online_Resource 1: Online resources about Python.....	16
Online_Resource 1-1: Solar extraterrestrial radiation spectrum.....	18
Online_Resource 1-2: Tabulated solstices and equinoxes for the years 2000..2100.....	26
Online_Resource 1-3: Tabulated equation of time for the years 2000..2100.....	31
Online_Resource 1-4: Solar position calculator (NREL).....	57
Online_Resource 1-5: Solar position calculator (NOOA).....	57
Online_Resource 1-6: Solar position calculator (timeanddate.com).....	57
Online_Resource 1-7: Cartesian and polar solar elevation plot (Univ. of Oregon).....	57
Online_Resource 1-8: Solar power collected by tilted surfaces (pveducation).....	72
Online_Resource 2-1: TMY2 and TMY3 datasets.....	77
Online_Resource 2-2: EPW EnergyPlus TMY dataset.....	86
Online_Resource 2-3: TMY format overview.....	86
Online_Resource 2-4: EU PVGIS tool for TMY series generation.....	88
Online_Resource 2-5: Conversion of TMY2, TMY3, INTL Format to .csv via PVWatts.....	89
Online_Resource 2-6: ClimateConsultant.....	137
Online_Resource 2-7: DView utility to display tmy, epw data.....	137
Online_Resource 3-1: Repositories for PV module and inverter data.....	150
Online_Resource 4-1: ESMAP Global solar atlas.....	163
Online_Resource 4-2: EU PVGIS tool for PV generation simulation.....	163
Online_Resource 4-3: PVWatts® online calculator for electricity PV production estimation	163

## Foreword

This primer and the related set of Python scripts have been prepared for students at Bachelor and Master's level in engineering and environmental sciences, in particular in solar energy applications, as well as for practicing engineers. Goal of these tools is to add practical programming examples oriented to the solution of real-life problems to the knowledge presented by established textbooks. The most important calculations in solar energy applications are presented here in form of code, with the theory kept at a necessary minimum. Some more in-depth explanations are given only to those aspects, such as the use of the TMY datasets, which are not yet adequately covered in published textbooks. Because of its practical character, the text contains several references to online resources, data repositories, and other freely accessible software.

Thanks to mass production and standardization, today most engineers working with renewable energy sources do not design solar energy conversion devices, such as thermal collectors or PV modules, but focus their attention to their integration in complex systems of which the load is an essential part. In this primer the accent is therefore put on the use of solar energy through a chain of components under the variable inputs of renewable energy sources and the load demand.

All information sources referred to in the text are non-commercial and the presented examples can be fully replicated without the need of licensed software of any kind. The program modules have been developed in Python 3.5 on the Ubuntu operating system, as spreadsheet was used LibreOffice Calc. The choice of Python as programming language was almost (and fortunately!) obligated: it is complex but simple, flexible, free source, it has a wide developers' base, and several application packages are available for advanced functions, such as `numpy` and `pandas` for data processing and `matplotlib` for graphic presentation. Different from other advanced, "hard-coded" software tools, access to the Python code makes modifications and further developments possible with a limited effort.

The code of this primer is based on self-contained modules that give predictable results on a given set of input parameters. For each module

are indicated the input and output parameters, while the internal algorithms are explained briefly in this primer and more at length with comments in the code. Each function, such as, for example, the computation of solar declination as function of the day of the year, is dealt with only in one module. In this way any updating of equations or procedures can be carried out in only one place and serve for all the other modules. The code makes use of Python intrinsic data structures, such as the `datetime` class, and builds on the most important packages for data processing and presentation: `numpy`, `pandas`, `matplotlib`.

The original idea for writing this text was born in Autumn, 2015, when I was invited to hold a crash course in renewable energy applications, with a special accent on solar energy, at the Polytechnic University “Peter the Great” in St. Petersburg, Russia. With good and simple online tools, such as NREL’s PVWatts, it was possible to show the practical side of textbook theory, with simulations under different scenarios. What at that time was missing, however, was a way to look at the internals of the computational algorithms and their effect on the calculations’ outcome in a easily understandable way for the students.

Several tools are available to describe at detail and simulate the operation of photovoltaic generators. The most advanced program is probably the System Advisor Model (SAM) developed at NREL, while important online tools are the already mentioned PVWatts, together with ESMAP and EU PVGIS. They are used as reference in several parts of this primer. The Python package described here is not an alternative to them, but in first place a tool to understand better and without too many details the mathematical models behind solar energy simulations.

With this package and this text I hope to have filled a gap. In particular, my wish is that it will be useful for those students who want to understand how textbook equations are formulated in practical applications, those who need flexible tools to present data in reports and theses, as well as practicing engineers working on solar system applications.

Gi.P., July 2017



## Main Reference Sites, Software

Some reference sites on solar energy are referred to on several places of this primer. They provide for a wealth of high-quality information and instruction materials, in some cases with the added bonus of interactivity. Websites and online information sources are indicated as **Online\_Resource** in the text. The following ones are particularly important: anybody who wants to build a strong background in solar energy theory and applications should be well acquainted with them.

### PV Education

An information and educational site on solar energy and photovoltaic generation, with a wealth of good illustrations and practical examples, many of which supported by online calculators.

<http://pveducation.org>

### NREL

The National Renewable Energy Laboratory (NREL) is the United States' main laboratory for renewable energy and energy efficiency. NREL regularly produces publications with the right balance between theory and practice, which are both a complement in study and a useful reference for practicing engineers.

NREL maintains several important data repositories, which are referred to in several parts of this primer.

<http://www.nrel.gov/>

### NREL System Advisor Model (SAM)

SAM is a software package developed and published by NREL. It provides a performance and financial model for renewable energy applications. SAM offers models and covers in detail different renewable sources, from PV generation to thermal solar to CSP large-scale power plants to biomass and others. One important feature of SAM is the provision of comprehensive datasets for weather (Typical Meteorological

Year, TMY) and photovoltaic components, such as solar modules and inverters. Several references to those datasets are made in this primer and many examples have been double-checked with SAM.

SAM needs to be downloaded and installed. It is available for the most widespread operating systems including Linux/Ubuntu. Consistent with the use of Open Source software only, references in this text are to SAM for Linux, though differences with the other versions should be minimal, if any.

The use of SAM does not underlie any licenses, but registration is required.

<https://sam.nrel.gov/>

### **PVlib-Python**

PVlib-Python is a large set of functions in Python developed at the Sandia Laboratories for the simulation of photovoltaic energy systems performance. In particular, Pvlib-Python is oriented to the detailed modeling of the atmosphere and of solar cells and modules.

<http://pvlib-python.readthedocs.io>

[https://pvpmc.sandia.gov/applications/pv\\_lib-toolbox/](https://pvpmc.sandia.gov/applications/pv_lib-toolbox/)

### **Pysolar**

Pysolar is a set of precise calculation routines focusing on solar radiation and solar position.

<https://pysolar.readthedocs.io/en/latest/#>

## 'solprimer' Package Download and Installation

### Python

Python is a simple to use, real programming language with a wealth of application packages available online.

Python is interpreted, which makes program development easy as it is possible to test and experiment with the language. An interpreted language does not require any compilation and linking, which contributes to make development and testing easier and faster.

Python can be easily ported between different operating systems, in principle, without any code changes. Python is currently available on the Windows, Mac OS X, and Unix (Linux/Ubuntu) operating systems.

Last but not least, Python is free.

### The 'solprimer' package

`solprimer` simply means 'solar primer', that is, a set of Python functions and scripts to carry out the basic computations related to solar energy applications, in particular photovoltaic systems. In the design of the `solprimer` package the main focus was not put on the program code, but on the interaction among modules and the use of external data, such as the Typical Meteorological Year datasets.

`solprimer` consists in 36 Python scripts and 5 packages in a subfolder. Some scripts do not need any input data (for example, for the calculation of the solar position), others require input files. In the code and the documentation are provided all necessary indications where to find them.

The `solprimer` package has been developed in Python3.5 on Ubuntu and tested for compatibility with Python2.7. The code is 'pure Python', which means that it does not use any external code, either precompiled or written in other languages, and 'universal', that is, it runs on both Python2 and Python3. As such, it should work on any Python environment independently of the operating system. Four scripts do not work under Python2.7 due to missing support or implementation for some

functions, such as the `timezone` definition. With some jingling, also these packages should be brought to run under Python2.7.

`solprimer` requires four external packages: `numpy`, `pandas`, `xlrd`, and `matplotlib`. The packages `numpy` and `pandas` are for structured data processing,<sup>1</sup> `xlrd` is required to read Excel-type files, `matplotlib` is a versatile graphical presentation package.<sup>2</sup>

The `solprimer` scripts are self-contained and do not act on any global variable. In principle, they can be installed anywhere in a user directory system, provided that the external packages are reachable via system path definitions (in Ubuntu the `$PYTHONPATH` system variable).

If the external `numpy`, `pandas`, `xlrd`, and `matplotlib` packages need to be installed anew, it may be worthwhile to consider a virtual environment, described later in this introduction.<sup>3</sup> Other required packages, e.g. `math`, are already installed as part of the standard Python distribution.

### 'solprimer' installation

The `solprimer` package is contained in the Python wheel distribution file **`solprimer-1.0.1-py2.py3-none-any.whl`** (or other version number). The code is contained in clear text and can be copied into any user folder. The package is *\*not\** meant to be installed in the Python `site-packages` folder, as it will make its use much more complicated. **`pip`**, the standard installation utility for `PyPI`, however, will first try to put the package exactly there. This can be prevented, as described here. The instructions are given for Python on the Ubuntu system, but the basic steps should be the same, or similar, for other systems as well.

1. select a root user directory where to carry out the installation, e.g., the `$home` root directory. The `solprimer` directory with its subdirectories will be created there.
2. download/copy the package wheel distribution into the chosen directory. By right-clicking over the green **DOWNLOAD** button the file can be saved either in the `downloads` or in the defined installation folder.

---

1 `numpy` is documented at the website <http://www.numpy.org/>, `pandas` at <http://pandas.pydata.org/>

2 the `matplotlib` website is <https://matplotlib.org/>

3 package install reference <https://packaging.python.org/tutorials/installing-packages/>

3. open the terminal and move to the directory where the `.whl` file has been saved

4. give the command, in one line

```
$ pip install solprimer-1.0.1-py2.py3-none-any.whl  
--no-compile -t <target_directory>
```

<target\_directory> is a string for a directory that will be created for the installation

The `--no-compile` option avoids the creation of compiled `.pyc` modules that here would only be a nuisance and should be deleted

The `-t` switch indicates the name of the <target\_directory>, for example, `-t newfiles`

5. the `solprimer` main folder is created under <target\_directory>. Under `solprimer` are installed all scripts and subdirectories for the packages, data, and the documentation:

- **solprimer** contains the Python scripts
- **solprimer/solprim** contains the system packages with functions to be called from the scripts
- **solprimer/data-input** is the default directory for input data
- **solprimer/data-output** is the default directory for output data
- **solprimer/data-TMY** contains the 'Typical Meteorological Year' (TMY) source and processed datasets.
- **solprimer/docs** contains the documentation, including this text 'solprimer.pdf'

Online sources for data that is required as input for some of the scripts , in particular for the TMY files, are indicated in the documentation primer.

It might help to move the `solprimer` directory up one or two levels in the directory hierarchy. The subdirectories can also be changed provided, of course, that the relevant references are changed in the scripts.

An alternative way to install the files is just to extract them from the wheel, which essentially is a zipped file, into the target directory. The file directory structure is visible by opening the wheel.

## Script execution

The scripts are executed under the terminal by calling **PYTHON** from the directory where the main scripts are located (solprimer root directory):

```
~/solprimer$ python clearsky_profile.py
```

alternatively, each script can be opened with the IDLE editor and executed with **<F5>**.

## Virtual environment

A virtual environment is a protected directory area with its own set of packages, scripts, and data that does not interfere with the operation of other programs on a system.<sup>4</sup>

In order to run solprimer a virtual environment might be useful for keeping the installation packages separated from the rest of the operating system, without the need to install code in the common repository. This might be the case on multi-user systems.

A solprimer virtual environment setup requires the following steps:

1. Python3 needs to be default instead of Python2. Depending on the system, Python3 might either need to be installed anew or just defined as default with a command alias.
2. activate the virtual environment. This may require the installation of a dedicated program such as venv in the newest Python releases.
3. ensure that the packages numpy, pandas, xlrd, matplotlib are installed and accessible, if not, they should be installed in the virtual environment
4. the \$PYTHONPATH variable must point to the Python3 site-packages

The installation will be briefly described here with reference to Linux/Ubuntu. The procedure may differ for other operating systems.

---

4 see the reference description at <https://docs.python.org/3/tutorial/venv.html>

Under Ubuntu the current (default) Python version is displayed via the terminal command **python --version**. Due to the fact that Python3 in most cases is already pre-installed, if it is not the default version it is sufficient to define an alias, such as

```
~/user$ alias python=python3
```

This command can be inserted in a `.bash_aliases` file for automatic execution at login.

Under Python3 the virtual environment is created with the command<sup>5</sup>

```
~/user$ python3 -m venv new_directory
```

`new_directory` can be a local directory or include a full path from the computer root directory.

There might be problems in the execution of `venv`, either due to the permission level for creating a directory, or the routine might need to be updated, or the operating system require a different command/procedure. In this cases refer to the relevant documentation.

The standard installation utility for PyPI, **pip**, if used within an active virtual environment will install the packages there and not in a central, system-wide repository.

With **pip** the installation of the application packages is straightforward:

```
pip install numpy
```

```
pip install pandas
```

```
pip install xlrd
```

```
pip install matplotlib
```

After the package installation it must be verified that the environment variable `$PYTHONPATH` points to the directory that contains them. On the terminal this is done with the command **echo \$PYTHONPATH**:

```
:$HOME/new_directory/lib/python3.5/site-packages
```

If `$PYTHONPATH` does not point to the correct directory, it can be redefined with the following command (the required path name for the actual system must be inserted):

```
~/new_directory$ export PYTHONPATH = $PYTHONPATH:  
$HOME/new_directory/lib/python3.5/site-packages
```

---

5 'venv' command reference <https://docs.python.org/3/library/venv.html#module-venv>

The same command can be added to the `.profile` file in the user home directory for automatic assignment at login time.

The instructions indicated above are very basic and cannot cover all possible situations that can arise. A good feeling for the machine and the operating system and the readiness to double-check every single step with online documentation (where a wealth of case descriptions and suggestions is available) are the best recipes for success.

### Online\_Resource 1: Online resources about Python

Official Website from the Python Software Foundation

<https://www.python.org/>

Python Tutorial

<http://docs.python.org/tutorial/index.html>

Library Reference

<http://docs.python.org/library/index.html>



# 1 The Solar Radiation Resource

## 1.1 Solar extraterrestrial spectrum

### 1.1.1 Main quantitative aspects of solar radiation

The solar radiation that reaches the outer atmospheric layers of the Earth has a particular spectral profile. Its intensity is highest between the wavelengths 0.3 and 4  $\mu\text{m}$ , i.e., between the ultraviolet and infrared electromagnetic radiation ranges, including visible light. About 7% of the of the extraterrestrial spectrum energy intensity is at ultraviolet wavelengths, 47% in the visible range and 46% infrared. The total power over all wavelengths is 1366  $\text{W}/\text{m}^2$ .

In solar energy applications two spectral profiles are particularly important, the extraterrestrial solar spectrum, called AM0 (from “Air Mass” 0), and the AM1.5 spectrum. One atmosphere, or Air Mass 1 (AM1), is defined as a column of air, ca. 100 km high, perpendicular to the Earth surface. When the sun is at or near the exact vertical of a point on Earth its radiation crosses, and is attenuated, by one atmosphere. This condition takes place only at geographical locations between the tropics and for a few days per year. For most of the times and the majority of locations the sun will have elevation angles  $<90^\circ$ , getting lower for locations further North. For the testing and certification of solar energy equipment has been defined and is used the solar spectrum AM1.5, equivalent to solar radiation after having crossed a mass of air corresponding to 1.5 atmospheres ( $\rightarrow$  Section 3.1.2).

Data sources for the solar radiation spectrum are reported in Online\_Resource 1-1. The tabulated spectrum AM0 and AM1.5 is shown via the Python script of Program\_Code 1-1.

### Online\_Resource 1-1: Solar extraterrestrial radiation spectrum

The tabulated extraterrestrial solar spectrum (ExtraTerrestrial Radiation, ETR) and the solar irradiation values across the atmosphere can be obtained from several online sources, they are also included in textbooks and reference handbooks on solar energy.

A spreadsheet with solar spectrum data is available at

<http://www.pveducation.org/pvcdrom/appendices/standard-solar-spectra>

[http://www.pveducation.org/sites/default/files/PVCDROM/Appendices/AM0AM1\\_5.xls](http://www.pveducation.org/sites/default/files/PVCDROM/Appendices/AM0AM1_5.xls)

The NREL Renewable Resource Data Center (RReDC) website presents an extensive collection of renewable energy resource data, including solar radiation data. Among other information it presents the current models for solar radiation outside the atmosphere and at ground level that are used for the definition of the AM0 and AM1.5 standards.

<http://rredc.nrel.gov/solar/spectra/>

<http://rredc.nrel.gov/solar/spectra/am0/>

The ASTM (originally “American Society for Testing and Materials”) is one of the most important worldwide organizations issuing standards. It has published widely accepted standards on the properties of solar radiation. ASTM G-173 is the Reference Solar Spectral Irradiance:

<http://rredc.nrel.gov/solar/spectra/am1.5/astmg173/astmg173.html>

ASTM E-490 is the Standard spectral irradiance for space applications at zero Air Mass (AM0)

ISO 9845-1:1992 is the international standard for solar spectral irradiance on the ground for air mass 1.5 (documentation provided on a commercial basis):

[http://www.iso.org/iso/catalogue\\_detail.htm?csnumber=17723](http://www.iso.org/iso/catalogue_detail.htm?csnumber=17723)

India has published the standard IS 12762 (Part 3) : 2013 as “Measurement Principles for Terrestrial Photovoltaic (PV) Solar Devices with Reference Spectral Irradiance Data”:

<https://law.resource.org/pub/in/bis/S05/is.12762.3.2013.pdf>

### Program\_Code 1-1: script, plot solar extraterrestrial spectrum from tabulated data

<b>script</b>	<b>spectrum_AM0AM15</b>
<b>description</b>	plot the solar extraterrestrial AM0, AM1.5 radiation profiles from tabulated data
<b>parameters</b>	<b>filename</b> : file name for spreadsheet file with solar spectral data including directory path, string
<b>returns</b>	graphical plot of solar spectral data

The solar spectrum data is read with the pandas function `read_excel` directly into a dataframe (for more information about pandas and dataframes see → Section 3.2.1). The graph is prepared and plotted with `matplotlib.pyplot`. An example of output is shown in Figure 1-1.

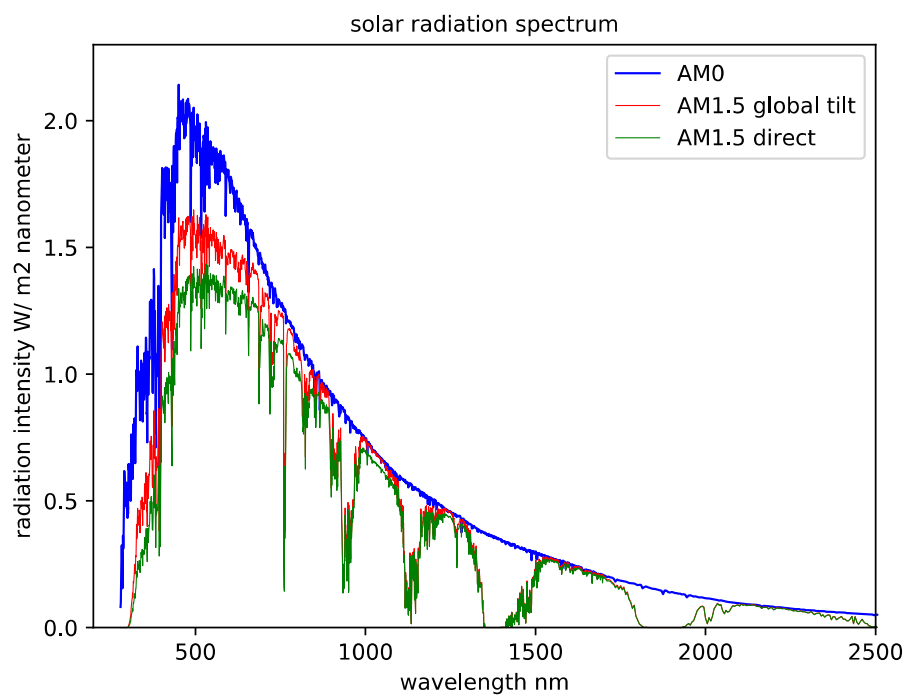


Figure 1-1: Solar radiation spectrum from spreadsheet

### Program\_Code 1-2: function, AM0 radiation distribution profile

<b>package</b>	<a href="#">solprim.solarradiation</a>
<b>function</b>	<a href="#">solar_spectrum_table_AM0</a>
<b>description</b>	simplified tabulated values for the solar extraterrestrial spectrum AM0
<b>parameters</b>	none
<b>returns</b>	tuple: - list: wavelengths in the range 0.3 – 4.0 micrometer, float - list: AM0 spectral density values in W/ m2/ micrometer, float

### Program\_Code 1-3: function, AM1.5 radiation distribution profile

<b>package</b>	<a href="#">solprim.solarradiation</a>
<b>function</b>	<a href="#">solar_spectrum_table_AM15</a>
<b>description</b>	simplified tabulated values for the solar extraterrestrial spectrum AM1.5
<b>parameters</b>	none
<b>returns</b>	tuple: - list: wavelengths in the range 0.3 – 4.0 micrometer, float - list: AM1.5 spectral density values in W/ m2/ micrometer, float

The functions [solar\\_spectrum\\_table\\_AM0](#) and [solar\\_spectrum\\_table\\_AM15](#) contain tables in form of a tuple of two lists of the same length. The same pointer will recover from one list a wavelength in micrometer and from the other the corresponding energy density of the solar spectrum in  $\text{W}\cdot\text{m}^{-2}\cdot\mu\text{m}^{-1}$ .

The output of the two functions is shown in Figure 1-2 together with the Planck's blackbody spectrum.

### 1.1.2 Planck's blackbody spectrum

Planck's blackbody law indicates the radiation spectral intensity  $B_\lambda$  emitted by a blackbody at a given temperature and at different wavelengths. It is measured as power emitted per unit area of the body, unit solid angle, and unit wavelength. Planck's formula has different aspects, depending on whether it is expressed as function of frequency or of wavelength. Following practice in work related to the solar spectrum, the law is expressed here as function of wavelength:

$$B_\lambda(\lambda, T) = \frac{2hc^2}{\lambda^5} \cdot \frac{1}{e^{\frac{hc}{\lambda k_B T}} - 1} \quad [\text{W sr}^{-1} \text{ m}^{-3}] \quad (1-1)$$

in Equation (1-1)  $\lambda$  is the wavelength of the emitted radiation component in meters,  $T$  the blackbody temperature in Kelvin,  $c$  the speed of light ( $=2.998 \cdot 10^8 \text{ m s}^{-1}$ ),  $k_B$  the Boltzmann constant ( $=1.381 \cdot 10^{-23} \text{ J K}^{-1}$ ) and  $h$  the Planck constant ( $=6.626 \cdot 10^{-34} \text{ J s}$ ).

The extraterrestrial solar spectrum is approximated quite closely by that of a blackbody at 5777 K. This value can be substituted for  $T$  in Eq. (1-1) and the resulting distribution calculated. However, if we just put in the parameters and make the calculation, the straight result differs from the measured solar radiation values by ten orders of magnitude. It therefore needs to be corrected as follows.

First of all, Planck's Equation (1-1) gives the energy density value per  $\text{m}^2$  radiative surface. It is then necessary to consider the surface of the sun as if it were a flat disk, the way it is seen from the earth, and multiply the Planck irradiation value by its area in  $\text{m}^2$ . The solar radius  $R$  is  $6.957 \cdot 10^8 \text{ m}$  and its area  $\pi R^2 = 1.52 \cdot 10^{18} \text{ m}^2$ . Secondly, energy radiates from the sun into all directions and Planck's relation refers to a solid angle, or steradian. On an ideal sphere with the sun at the center and radius  $R_{SE}$  equal to the sun-earth distance, a steradian has an area over the celestial sphere surface equal to  $R_{SE}^2$ . With an average sun-earth distance  $= 149 \cdot 10^9 \text{ m}$ ,  $R_{SE}^2$  is equal  $2.22 \cdot 10^{22} \text{ m}^2$ . Finally, the result needs to be scaled from meter to micrometer with the factor  $10^6$ .

From the coefficients indicated above, the conversion factor to relate the result of Planck's equation to the solar radiation value on the earth's outer atmosphere becomes

$$\frac{1.52 \cdot 10^{18}}{2.22 \cdot 10^{22} \cdot 10^6} \approx 0.685 \cdot 10^{-10}$$

that is, it covers ten orders of magnitude. The plot of Planck's function calculated with Equation (1-1) over the range 0.3 – 4.0  $\mu\text{m}$  is shown in Figure 1-2.

#### Program\_Code 1-4: function, Planck energy density profile

<b>package</b>	<b><code>solprim.solarradiation</code></b>
<b>function</b>	<b><code>planck_distribution</code></b>
<b>description</b>	generate the Planck distribution profile for blackbody radiation for a given temperature and in a given wavelength range
<b>parameters</b>	<b>temp</b> : temperature in degrees Kelvin (K), integer or float <b>wlen0</b> : minimum wavelength in $\mu\text{m}$ , float, default = 0.3 <b>wlen1</b> : maximum wavelength in $\mu\text{m}$ , float, default = 4.0 <b>steps</b> : number of simulation steps, integer, default = 120
<b>returns</b>	list with values of Planck's blackbody spectral distribution for temperature (K) between the min and max wavelengths for the given number of simulation steps

The function `planck_distribution` calculates the profile of the Planck's blackbody energy distribution between two wavelengths and for the desired number of steps.

### Program\_Code 1-5: script, plot Planck's blackbody spectrum over AM0, AM1.5

<b>script</b>	<b>spectrum_blackbody</b>
<b>description</b>	generate and display a graph of the Planck energy density profile for blackbody radiation and the AM0, AM1.5 profiles
<b>parameters</b>	<b>temp</b> : temperature in degrees Kelvin (K), integer or float <b>wlen0</b> : minimum wavelength in $\mu\text{m}$ , float, default = 0.3 <b>wlen1</b> : maximum wavelength in $\mu\text{m}$ , float, default = 4.0 <b>steps</b> : number of simulation steps, integer, default = 120
<b>returns</b>	plot of Planck's blackbody spectral distribution

The script `spectrum_blackbody` plots Planck's energy density profile calculated with Equation (1-1) over the range 0.3–4.0  $\mu\text{m}$  together with the standard AM0, AM1.5 spectra, as shown in Figure 1-2.

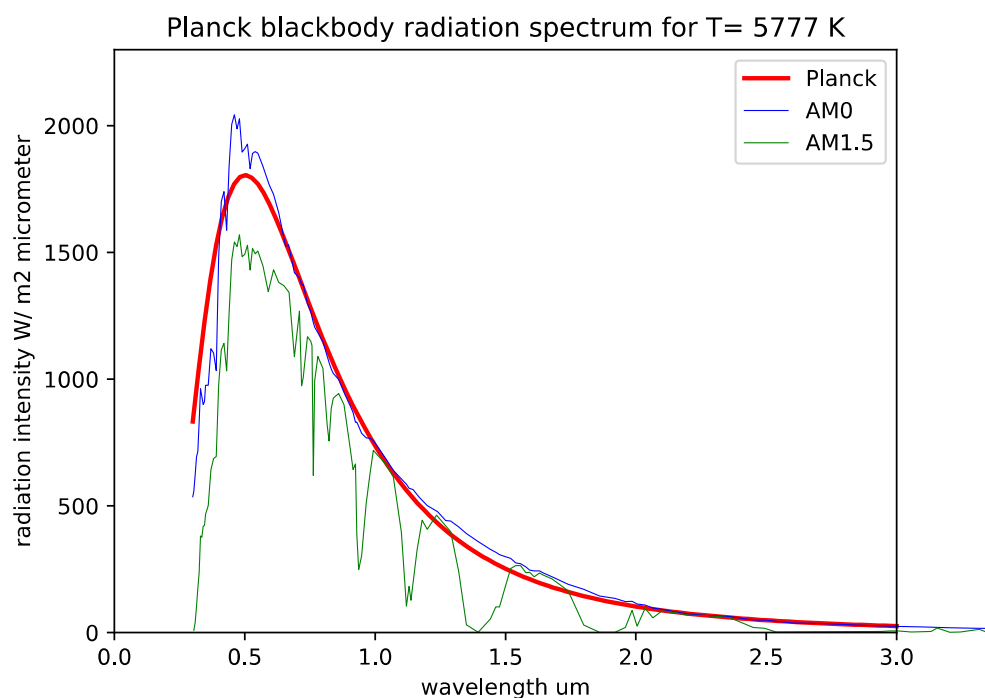


Figure 1-2: Planck's blackbody radiation spectrum compared to AM0, AM1.5

## 1.2 Local and solar time

### 1.2.1 Time representation as Python class

Time and date information can be represented in Python with a single variable by using the `datetime` class.<sup>6</sup> The availability of several packages and functions for operation on variables related to time simplifies coding. The following description is related to the `datetime` class implementation used in the code examples in this primer.

The main date and time variable in the code examples is *localdate*, for date and time or date only information, or *localtime*, if it refers only to time. *localdate* is a `datetime` object containing year, month, day, hour, minute, and timezone for a location, *localtime* contains only hours, minutes, and seconds. With the information in *localdate* and together with a location's longitude and latitude it is possible to perform all calculations related to the solar position in relation to any point on earth at any date and time. `datetime` objects will be used through all examples in order to maintain consistency, though in some cases different representations would lead to faster processing.

The use of the most important `datetime` objects is illustrated by the following script.

#### Program\_Code 1-6: script, Python datetime class examples

<b>script</b>	<b><a href="#">datetime_example</a></b>
<b>description</b>	show the operation of the main methods and functions for the Python <code>datetime</code> class for the current time read from the computer internal clock
<b>parameters</b>	none
<b>returns</b>	results of different <code>datetime</code> methods and functions

---

<sup>6</sup> The basic date and time types are described in §8.1 of the Python online documentation, <https://docs.python.org/3/library/datetime.html>



An example of output for `datetime_example` is

```
2017 7 4 20 2 -2.5
localdate 2017-07-04 20:02:00-02:30
localdate.time() 20:02:00
localdate.timetz() 20:02:00-02:30
localdate.astimezone() 2017-07-05 00:32:00+02:00
localdate.utcoffset() -1 day, 21:30:00
localdate.dst() None
localdate.tzname() UTC-02:30
localdate.tzinfo UTC-02:30

localdate.timetuple time.struct_time(tm_year=2017, tm_mon=7, tm_mday=4,
tm_hour=20, tm_min=2, tm_sec=0, tm_wday=1, tm_yday=185, tm_isdst=-1)

localdate.utctimetuple time.struct_time(tm_year=2017, tm_mon=7, tm_mday=4,
tm_hour=22, tm_min=32, tm_sec=0, tm_wday=1, tm_yday=185, tm_isdst=0)

localdate.timetuple().tm_yday 185

localdate.timetuple().tm_gmtoff None
```

The main date and time variable is called *localdate*. The initial values for year, month, day, hour, minute are read from the system clock, alternatively they can be input manually. These values together with the float value for the timezone are reported in the first line.

The *localdate* components cannot be assigned directly in the code. An object must be built with help of a **constructor**. The elements of an already existing object can be changed with the **replace** function.

`localtime.tzinfo` is shown as UTC-02:30, as it was input. The representation of `localtime.utcoffset()` is quite peculiar, as -1 day, 21:30:00, when what is meant is “-2.5”. Positive timezone values indicate locations East of the UTC meridian, negative values Western locations.

### 1.2.2 Tracking solar time and position

The mathematical modeling of the relative movement of earth and sun is highly complex, as this movement is affected by several factors. Fortunately, the most important aspects can be described precisely with simplified models, so that the number of the equations for the earth path around the sun, and, conversely, the position of the sun in the earth sky can be kept to few and manageable relations.

The main parameters describing the earth's orbit around the sun are shown in Table 1-1. Several important reference parameters, such as the exact timing of solstices and equinoxes are not fixed, but vary from year to year (Online\_Resource 1-2).

angle between the earth rotation axis and the plane of the orbit (ecliptic)	23.45° (*)
duration of the day with reference to the fixed stars	23h 56m 4s
duration of the day with reference to the sun	24h
duration of the year	365.25 days
aphelion (maximum distance earth-sun), around July 4 <sup>th</sup>	152.10×10 <sup>9</sup> m
perihelion (minimum distance earth-sun), around January 3 <sup>rd</sup>	147.10×10 <sup>9</sup> m
date of the spring equinox	March 20

*Table 1-1: Main orbital parameters of the Earth around the Sun*

In the following are presented the basic equations to model timekeeping and the calculation of the solar position in the sky at a specific date and time. The precision of the simplified calculations in comparison to the most complex ones is discussed in → Section 1.3.5.

### Online\_Resource 1-2: Tabulated solstices and equinoxes for the years 2000..2100

The exact dates for the solstices and the equinoxes (cardinal points) are not fixed but vary year after year. Tabulated values for solstices and equinoxes from 2000 to 2100 are published in

<http://www.astropixels.com/ephemeris/soleq2001.html>

(\*) In this text angles are indicated in degrees with decimals because this simplifies calculations. The declination angle is indicated in some texts as 23°26', i.e., in degrees and minutes, which is the same as 23.45°.

### Program\_Code 1-7: function, day of year

<b>package</b>	<b>solprim.solartimeposition</b>
<b>function</b>	<b>day_of_year</b>
<b>description</b>	from a date and time object in Python <code>datetime</code> format return the day of the year, with January 1 <sup>st</sup> = day 1
<b>parameters</b>	<b>localdate</b> : local date and time object in Python <code>datetime</code> format if time and timezone are included in the <code>datetime</code> object the day of year is calculated with reference to UTC, otherwise it is local
<b>returns</b>	day of year [1..365 or 366], integer

The function `day_of_year` returns the day of year [1..365] or [1..366] for a particular date as an integer, January, 1<sup>st</sup> is =1. In leap years, February 29<sup>th</sup> is day=60 and March 1<sup>st</sup> day=61. If the time and timezone information is contained in the `datetime` object, the day of year is returned with reference to UTC and not to local time.

Output examples:

At 23:59 of January 1<sup>st</sup> without consideration of the timezone the day of year is =1

```
>>> localtime 2017-01-01 23:59:00+00:00
>>> day of year = 1
```

At the same date and time and for a positive timezone the day of year is also =1. The location is East of Greenwich, i.e., UTC time is earlier than 23:59 and the day is still =1.

```
>>> localtime 2017-01-01 23:59:00+01:00
>>> day of year = 1
```

A negative timezone would indicate that the location is West of Greenwich, the UTC day of year is =2:

```
>>> localtime 2017-01-01 23:59:00-01:00
>>> day of year = 2
```

### Program\_Code 1-8: function, timezone offset

<b>package</b>	<a href="#">solprim.solartimeposition</a>
<b>function</b>	<a href="#">timeoffset_tz</a>
<b>description</b>	from a date and time object in Python datetime format return the timezone
<b>parameters</b>	<b>localdate</b> : local date and time object in Python datetime format
<b>returns</b>	timezone from the <i>utc_offset</i> attribute in a Python datetime object

The timezone can be read directly as attribute of a datetime object. However, the `datetime.utcoffset` method returns a `timedelta` in days and seconds. In case of a negative `utcoffset` value, this is given as `-1` day together with a positive amount of seconds. For example, `-1` hour is shown as `-1` day and `+82,800` seconds, their sum gives `-3600` seconds, i.e., `-1` hour. The [solprimer](#) function [timeoffset\\_tz](#) calculates and returns the timezone as a single, signed float value for hours and decimals.

Timezones East of Greenwich have positive values, West of Greenwich negative values. For example, New York is in the timezone UTC-5.

### 1.2.3 UTC, local, and solar time

Three time references are used when tracking the solar position seen from the earth: universal time (UT), local time (LT), and the true solar time (TST). **Universal time**, also known as **UTC** (Coordinated Universal Time) or **GMT** (Greenwich Meridian Time), is the basis of legal timekeeping over the whole world and is independent of the location on earth. The name points to the historical choice of Greenwich, near London, as main reference for timekeeping and for determining the geographical longitude for all other locations on earth.

**Local time**, also known as **legal time** or **standard time**, is the time commonly used in any given country and location. Its minutes count is the same as the UTC minutes, while the hours depend on the timezone and whether daylight saving time (“summer time”) is in effect.

The **true solar time**, also called **apparent solar time**, is the local time defined by the solar position, with 12:00 noon as the instant when the sun is highest in the sky. In this Python primer UTC and local time are represented with a `datetime` object, the true solar time with a float variable.

While the difference between local time and UTC time is constant, the difference between local time and true solar time varies during the year (→ Section 1.2.4) Conversions between the two time systems are often necessary because in solar energy applications the solar position in the sky and the energy yield of flat panels are calculated much more easily with reference to the solar time than to local time.

Until the mid of the 19th century solar time and local time were in practice the same. Noon time was when the sun was highest in the sky, as indicated by the shadow of a pole or a sundial. Every city and village had its own local time. When more precise mechanical clocks became available at the beginning of the 19th century, it was clear that solar noon was not a regular time reference but could shift several minutes during the year, the difference is known as Equation of Time (→ 1.2.4). The solar time corrected by the Equation of Time is called **mean solar time**. Its difference to legal time depends only on the longitude of the location.

Another mismatch in time tracking became evident with increasing speed in transport and communication. The necessity for coordination of transport, in first place for railway schedules, made the use of local solar time increasingly inconvenient due to the necessity of following it at each location separately – every station had its own timezone! The issue was solved by defining 24 time zones, each 15° degrees longitude wide ( $\pm 7^{\circ}30'$  around the meridians 0°, 15°, 30°, ...). In each time zone the local time is equal to the mean solar time of its meridian. The time difference from each time zone to the next is one hour, while the minutes count is the same across most time zones. In this way, when it is 11h 23min in London, it is 19h 23min in Beijing, 21h 23min in Sidney, 6h 23min in New York, etc. For practical reasons in most cases the timezone borders follow political and administrative boundaries. In large countries or administrative entities, such as China, India, or the European Union, despite the easternmost and westernmost longitudes being much more than 15° apart, the same time reference holds for the whole area. In some countries the minute count is shifted by 30' or 15' compared to UTC.

### 1.2.4 Equation of time

The **Equation of Time** (also: **coefficient of time**) is a correction factor to account for the irregularity of the speed of the earth's motion around the sun, which leads to differences between solar time and local time. This deviation results from two major effects in the earth's movement, one with periodicity one year, the other half year. The first is due to the eccentricity of the earth orbit, i.e., that this is not perfectly circular but an ellipse, with the earth moving faster at the perihelion (around January 3rd) and slower at the aphelion (on or near July 4th). The second effect is due to the tilt of the earth's rotational axis with respect to the plane of its orbit, so that the sun seems to go faster than the clock at the summer and winter solstices and slower at the vernal (spring) and autumn equinoxes. The Equation of Time accounts for both these effects, it has the form<sup>7</sup>

$$ET \text{ (minutes)} = 9.87 \cdot \sin(2 \cdot B) - 7.53 \cos(B) - 1.5 \sin(B) \quad (1-2)$$

$$\text{with } B = 360 \cdot \frac{(d - 81)}{365} \text{ degrees}$$

Different Authors indicate different coefficients and different beginnings for the day count, though the equation is always built on two trigonometric functions with periodicity one year and half year. The Equation of Time can also be expressed in radians with the factor  $2\pi$  instead of 360.

In Eq. (1-2)  $d$  is the number of days since the beginning of the year, with  $d=1$  for January 1.  $B$  is the angular position of the earth along its orbit, moving at slightly less than  $1^\circ$  each day. This formulation of the Equation of Time is intrinsically incorrect, because its value must be continuous, while the day index  $d$  is discrete. However, the maximum difference from one day to the next is less than half minute, which is fully acceptable for most practical purposes.

There isn't any accepted convention concerning the sign of the Equation of Time, though at the beginning of the year solar time is beyond local time and a negative sign can be assumed. The apparent solar time can be ahead of the mean solar time by as much as 16 min 33 s, around Nov 3, or behind it by up to 14 min 6 s, around Feb 12. On or around Apr 15, Jun 13, Sep 1 and Dec 25 apparent and mean time are the same.

---

<sup>7</sup> Equation from D. Yogi Goswami, Principles of Solar Engineering, Third Edition, CRC Press, Taylor & Francis Group, 2015

### Program\_Code 1-9: function, equation of time

<b>package</b>	<b>solprim.solartimeposition</b>
<b>function</b>	<b>equation_of_time</b>
<b>description</b>	from the day of year return the correction factor given by the equation of time
<b>parameters</b>	<b>d</b> : day of year in decimal format, integer [1..365]. d=1 is January 1.
<b>returns</b>	equation of time correction at the given day in decimal minutes, float

This function returns the Equation of Time for the day of year.

The Equation of Time is computed with the relation<sup>8</sup>

$$ET \text{ (minutes)} = 9.85 \cdot \sin\left(\frac{4\pi(d-80)}{365.2422}\right) - 7.65 \sin\left(\frac{2\pi(d-3)}{365.2422}\right) \quad (1-3)$$

Similarly to Equation (1-2) , Eq. (1-3), from a different Author, has a component with one-year periodicity and one with two times per year.

The maximum difference of the equation of time from one day to the next is less than half minute. The equation of time is required only for the conversion between solar time and local time, for most practical purposes this error is not important.

#### Online\_Resource 1-3: Tabulated equation of time for the years 2000..2100

The tabulated Equation of Time for the years up to 2050 can be calculated with the script at the website <http://www.minasi.com/doesot.htm>

<sup>8</sup> Equation from C. Julian Chen, Physics of Solar Energy, John Wiley & Sons, Inc., 2011



### 1.2.5 Conversion between local time and true solar time

The calculations related to the solar position are based on the true (apparent) solar time, though in several situations it is necessary to know the relation to the actual local time, for example, to verify the operation of solar equipment at exact solar noon, which is not the same as when the clock indicates 12:00.

The relation between solar and local time is

$$T_{solar} = hr + \frac{min}{60} + \frac{lon}{15} - tz + EOT \quad (1-4)$$

The first two terms represent the conversion of the local time from hours:min to decimal hours. The third and the fourth terms relate the local time to the meridian (longitude) of the actual location. The fraction divisor = 15 reflects the one hour time difference for two meridians 15° apart.  $tz$  is the time zone of the location and it may include the daylight saving time shift. EOT is the correction factor from the equation of time.

In this equation the solar time is expressed in decimal hours, i.e., 7.5 hours is the same as 7h 30min. This notation makes it easier to calculate the trigonometric functions for the solar position where the solar time needs to be converted into an angle.

The timezones and the longitudes are conventionally considered as positive in direction East and negative in direction West. Moscow, at longitude 38°, is located in the timezone GMT+3, New York, at longitude -74°, is in GMT-5. In this way, the signs of Equation (1-4) are consistent.

An example shall illustrate the conversion procedure. Paris has longitude 2°21' East (decimal: 2.35) and is located in timezone GMT+1, with time reference meridian +15°. The solar time corresponding to the local time 14:13 on the 2<sup>nd</sup> of February is:

		hours with decimals
local time hours	14	14.00
local time min	13	0.22
longitude	2.35	0.16
tz (hours)	1.00	-1.00
EOT (min)	-13.77	-0.23
<b>solar time</b>		<b>13.15</b>



Solar time is 13.15 hours, or 13:09 in the common representation with 60 minutes per hour. The difference between local time and solar time is 1 hour 4 minutes, most of which is due to the timezone shift. Conversely, by definition, the sun reaches its highest position in the city sky at 12:00 noon solar time, i.e. at 13:04 local time. If it depended only on longitude and timezone the difference between solar and local time would remain the same during the year. Because of the effect of the Equation of Time and, where applicable, of daylight saving time, the difference between solar and local time depends also on the calendar day.

#### Program\_Code 1-10: function, true solar time from local time

<b>package</b>	<b><code>solprim.solartimeposition</code></b>
<b>function</b>	<b><code>true_solar_time</code></b>
<b>description</b>	calculate the true solar time from a date and time object in Python <code>datetime</code> format and a location identified by its longitude
<b>parameters</b>	<b><code>localtime</code></b> : local date and time object in Python <code>datetime</code> format including timezone <b><code>longitude</code></b> : longitude in decimal degrees (East is positive), float, default =0 (GMT meridian) If the timezone is not included in <code>datetime</code> and the longitude is not passed to the function, the <code>datetime</code> object is taken as local time
<b>returns</b>	true solar time in decimal hours (hours with decimal fraction)

The function `true_solar_time` accepts a `datetime` object and a longitude indication and returns the true solar time as hour with decimal fraction part.

Example of call to `true_solar_time` for the calculation example above

```
>>> localtime 2017-02-02 14:13:00+01:00
>>> true_solar_time = 13.15
```

### Program\_Code 1-11: function, local time from true solar time

<b>package</b>	<b><code>solprim.solartimeposition</code></b>
<b>function</b>	<b><code>local_time_from_solar</code></b>
<b>description</b>	from the true solar time as decimal hour with fraction, a reference localtime and a longitude return a <code>datetime</code> object in local or UTC time
<b>parameters</b>	<b><code>sun_time</code></b> : true solar time as hour with decimal fraction, float <b><code>localtime</code></b> : local date and time object in Python <code>datetime</code> format, time and timezone are included in the evaluation, though they may be omitted <b><code>longitude</code></b> : longitude in decimal degrees (East is positive), float, default =0 (GMT meridian)
<b>returns</b>	local time as Python <code>datetime</code> object

The function `local_time_from_solar` is the inverse of `true_solar_time`. It accepts a solar time hour with decimal fraction part, a reference `datetime` object with date and timezone indication, and a longitude value, and returns the local time as `datetime` object.

For a solar time near midnight it may happen that the calculated local time is one day earlier or later. This is not corrected in the calculated `datetime` object because the input solar time does not include a date representation. In practical terms, the conversion function from solar time to local time makes sense only for the sunlight hours, which in most cases are far from midnight anyway. The polar zones would be an exception, but there the use of solar energy is unrealistic because of the low solar elevation angles over the horizon.

Example of the function call

```
>>> true_solar_time = 13.15
>>> calculated local time = 2017-02-02 14:13:00+01:00
```

---

**Program\_Code 1-12: function, represent decimal hours in HH:MM format**


---

<b>package</b>	<b>solprim.solartimeposition</b>
<b>function</b>	<b>solar_time_HHMM</b>
<b>description</b>	convert a solar time expressed in decimal hours (hours with decimal fraction) to a string in hours and minutes
<b>parameters</b>	<b>sun_time</b> : true solar time as hour with decimal fraction, float
<b>returns</b>	string 'hours:minutes', hours [00..23] and minutes [00.59]

The function **solar\_time\_HHMM** converts an hour in decimal fraction format representing the true solar time to a hh:mm string with the hours in range [00..23] and minutes in range [00.59].

The solar time calculated in the earlier example has a value in hours and minutes

```
>>> true_solar_time = 13.15
>>> true_solar_time [HH:MM] = 13:09
```

### 1.2.6 Hour angle

The **solar hour angle**  $h_s$  is a practical indication of the time of the day considered as the difference between the current local solar time and solar noon expressed as an angle. It is used as indication of the time of the day in calculations related to the solar position in the sky. The hour angle is calculated from the true solar time with the relation

$$h_s = (T_{solar} - 12) \cdot 15^\circ \quad (1-5)$$

At solar noon, with the sun at its maximum elevation over the horizon, the hour angle is  $0.0^\circ$ . Each full hour difference to solar noon corresponds to  $15^\circ$ , four time minutes to one angular degree, the full 24-hour day to  $360^\circ$ . Conventionally, the time before solar noon is expressed with a negative value and after solar noon as positive. For example, at 9:30 local solar time the hour angle is  $-37.5^\circ$  ( $15^\circ$  per hour  $\times$  2.5 hours time difference to noon), at 14:30 solar time the hour angle is  $+37.5^\circ$ .

### Program\_Code 1-13: function, hour angle from time in decimal format

<b>package</b>	<b><a href="#">solprim.solartimeposition</a></b>
<b>function</b>	<b><a href="#">hour_angle_decimal</a></b>
<b>description</b>	from the solar time in decimal hour format return the hour angle
<b>parameters</b>	<b>sun_time</b> : true solar time as hour with decimal fraction, float <b>eot_corr</b> : Equation of Time correction in decimal minutes, float, default=0 <b>longitude</b> : longitude in decimal degrees (East is positive), float, default =0 <b>timezone</b> : timezone (East is positive), float, default=0
<b>returns</b>	hour angle [-180..+180] in degrees with decimal fraction, float

### Program\_Code 1-14: function, hour angle from time in datetime format

<b>package</b>	<b><a href="#">solprim.solartimeposition</a></b>
<b>function</b>	<b><a href="#">hour_angle_localtime</a></b>
<b>description</b>	from a time object representing the solar time in Python <code>datetime</code> format return the hour angle
<b>parameters</b>	<b>localtime</b> : local time object in Python <code>datetime</code> format
<b>returns</b>	hour angle [-180..+180] in degrees with decimal fraction, float

The functions of Program\_Code 1-13 and Program\_Code 1-14 calculate the hour angle from time expressed in decimal hour or as Python `datetime` object respectively. The code of both functions is very simple, they are used for convenience in calls from other scripts or functions. The time corrections in [hour\\_angle\\_decimal](#) bring improved precision in the calculation of the solar position as function of local time.

#### 1.2.7 Solar declination

In most locations on earth, with the exception of those around the equator, the total daily amount of solar radiation varies markedly during the year. This is the consequence of the tilt of the earth rotation axis to

the orbital plane around the sun (obliquity of the ecliptic). The reference parameter for this effect is the **solar declination** angle  $\delta_s$ , defined as the angle between the line that joins the centers of the earth and the sun and the equatorial plane, which is perpendicular to the earth rotational axis.  $\delta_s$  varies between  $-23.45^\circ$  on Dec 21 and  $+23.45^\circ$  on Jun 21 (winter and summer solstice).

At any day of the year the solar declination has the same numerical value as the latitude at which the sun is directly overhead (zenith position) at solar noon. The tropics of Cancer ( $23.45^\circ\text{N}$ ) and Capricorn ( $23.45^\circ\text{S}$ ) are the extreme latitudes where the sun is overhead at least once a year, which happens at the solstices. At latitudes higher than  $66.55^\circ\text{N}$  and lower than  $-66.55^\circ\text{S}$  at least once per year the sun does not rise above the horizon. These limit latitudes are called the Arctic and Antarctic circles, their absolute numerical value is equal  $90^\circ$  less the earth orbit tilt angle  $23.45^\circ$ . At the poles the sun stays above the horizon half the year (polar day) and remains below it for the other half of the year (polar night). Polar day and polar night begin and end at the spring and autumn equinoxes.

The solar declination  $\delta_s$  is approximately defined by the relation

$$\delta_s = 23.45^\circ \cdot \sin\left(\frac{360^\circ \cdot (d - 81)}{365}\right) \quad (1-6)$$

with  $d$  as the day number during the year with  $d = 1$  being January 1. The coefficient = 81 is used to relate the date to March 22 (approximate date for the spring equinox), when  $\delta_s = 0^\circ$ . For  $d < 81$  the factor 365 can be added to obtain a positive number for the date, though the result would remain the same because of the periodic nature of trigonometric functions. Via the fraction  $360^\circ/365$  the day number is scaled to the corresponding angle on the full orbital circle. Some textbooks present this equation as a cosine function of  $d+10$ , which yields the same result. The maximum error of Equation (1-6) is  $1.60^\circ$ .

A more precise equation for the computation of the solar declination  $\delta_s$  with a maximum error of  $\pm 1^\circ$  is

$$\delta_s = \sin^{-1}\left(\sin(23.45^\circ) \cdot \sin\left(\frac{360^\circ \cdot (d - 81)}{365}\right)\right) \quad (1-7)$$

For the calculation of the solar declination with Equation (1-6) and (1-7) hold the same considerations as for the Equation of Time, Eq. (1-2) and (1-3). These equations are inherently imprecise because the declination, the same as the correction of the Equation of Time, is a continuous function and not a stepwise parameter the way that the day number is. The day number could be passed as a fractional value, starting, e.g., from noon at the reference meridian and then adding correction values for the time of the day. However, even this continuous equation would remain an approximation, because the exact time of the winter and summer solstices varies from year to year. Moreover, the exact relation is not a simple trigonometric function. In real life and for most practical purposes these aspects can be disregarded. The maximum difference of the solar declination from one day to the next is  $0.4^\circ$  at the equinoxes, the error brought by the approximated equations (1-6) and (1-7) is therefore much lower than other factors that influence solar power generation, as discussed in → Section 1.3.5.

#### Program\_Code 1-15: function, solar declination

<b>package</b>	<b><code>solprim.solartimeposition</code></b>
<b>function</b>	<b><code>solar_declination</code></b>
<b>description</b>	from the day of year return the solar declination
<b>parameters</b>	<b>d</b> : day of year in decimal format, integer [1..365]. d=1 is January 1.
<b>returns</b>	solar declination at the given day in decimal degrees, float

The function `solar_declination` returns the solar declination for a given day of year according to the approximate relation<sup>9</sup>

$$\delta_s = 23.45^\circ \cdot \sin\left(\frac{2\pi \cdot (d - 80)}{365.2422}\right) + 0.01 \quad (1-8)$$

The term 0.01 has been introduced in order to avoid an output value  $\delta_s = 0$ , which would lead to numerical inconsistencies in other equations.

<sup>9</sup> Equation from C. Julian Chen, Physics of Solar Energy, John Wiley & Sons, Inc., 2011

### 1.2.8 Sunrise and sunset hour angle

From the solar declination  $\delta_s$  (which depends on the day of year) and the latitude  $\phi$  of a location can be determined the hour angle at which the sun is exactly at the horizon at sunrise or sunset. From the hour angle can be easily calculated the length of the solar day, which is twice as long.

The sunrise hour angle  $A_{SR}$  is given by the relation:

$$A_{SR} = -\arccos(\tan \phi \tan \delta_s) \quad (1-9)$$

The sunset hour angle  $A_{SS}$  has the same value of  $A_{SR}$ , with positive sign (the sunrise and sunset times are symmetric to the solar noon).

The arccosine function of Equation (1-9) can be calculated if its argument is in the range  $[-1..+1]$ . Values outside this range mean

$$\begin{aligned} A_{SR} &\geq 1 && \text{sun down 24h, polar night} \\ A_{SR} &\leq -1 && \text{sun up 24h, polar day} \end{aligned}$$

Argument values outside the range  $[-1..+1]$  are the result of high values of the latitude and the solar declination, that is, north of the Arctic circle or south of the Antarctic circle at times near the Summer solstice.

For values of  $A_{SR}$  in the range  $[-1..+1]$  the sunrise time is equal to 12 less the sunrise angle converted to decimal hours (Equation 1-5). Conversely, the sunset time is 12 plus the sunset angle converted to hours. The length of the solar day is the difference  $A_{SS} - A_{SR}$ , or twice the value of the hour angle.

In Equation (1-9) the sun is treated a geometrical point and not as a disk, as it in fact appears from the earth. In astronomy sunrise and sunset are defined as the times when at least some portion of the sun is visible on the horizon. The radius of the sun is  $16'$ , sunrise therefore occurs already at a solar elevation  $\alpha = -16'$ . In addition, because of atmospheric refraction at lower solar elevations, the sun appears to be over the horizon when it is in fact half degree below it. The apparent sunrise or sunset should therefore be calculated for a reference elevation  $\alpha = -50'$ . In this way the observed solar day becomes some minutes longer than the one calculated with Equation (1-9).

### Program\_Code 1-16: function, sunrise and sunset hour interval

<b>package</b>	<b>solprim.solartimeposition</b>
<b>function</b>	<b>sunrise_hour</b>
<b>description</b>	calculate the interval (hour angle) from solar noon to sunrise and sunset
<b>parameters</b>	<b>sol_decl</b> : solar declination in decimal degrees, float <b>latitude</b> : latitude in decimal degrees (North is positive), float
<b>turns</b>	tuple with - absolute value for sunrise, sunset angle in decimal degrees, float - sunrise time (solar mean time) in hours with decimal part, float - sunset time (solar mean time) in hours with decimal part, float - flag <b>sr_correct=True</b> , the equation argument is in correct range, a result was calculated. When <b>False</b> , the sunrise/sunset values are meaningless. - flag <b>sunup_flag=True</b> , no sunrise/sunset, sun up all the time (polar day). <b>sunup_flag=False</b> means polar night. The result is valid only when the flag <b>sr_correct=False</b> .

The **sunrise\_hour** function calculates the sunrise hour based on the geometrical considerations presented in this Section.

The arguments **sol\_decl** and **latitude** are transferred to the function in degrees and not in radians because this is the common format that they are represented in. In addition, this function is not called in tight loops and the conversion from degrees to radians needs to take place in any case, inside or outside the function. The function returns a tuple with the absolute value of the sunrise/sunset angle (the result of Equation 1-9), and the sunrise and sunset times in decimal hour format.

The flag **sr\_correct=True** indicates that the argument of Equation (1-9) is within the range  $[-1..+1]$  and the arccosine is evaluated correctly. The flag **sr\_correct=False** means that the sun path never crosses the horizon. In this case **sunup\_flag=True** indicates that the sun is above the horizon during the whole day (polar day), while **sunup\_flag=False** indicates that the sun is below the horizon all the time (polar night).



### Program\_Code 1-17: script, table with sunrise, sunset times at a given location

<b>script</b>	<b><a href="#">sunrise_sunset_time</a></b>
<b>description</b>	for a given latitude generate a table with sunrise and sunset times for all days of the year
<b>parameters</b>	<b>latitude</b> : latitude in decimal degrees (North is positive), float <b>longitude</b> : longitude in decimal degrees (East is positive), float <b>timezone</b> : timezone (East is positive), float
<b>returns</b>	spreadsheet file in .csv format with calendar day, solar declination, sunrise and sunset time as true solar time, sunrise and sunset time in local standard time. Times are returned in format %H:%M For the polar day and the polar night the indications 'Pol_day' and 'Pol_night' are returned instead of the sunrise and sunset times.

This script produces a table in .csv format with sunrise and sunset in solar apparent time and in standard local time for any location on earth identified by its geographical coordinates. The output file name contains references to latitude, longitude, and time zone in the following form:

sunrise\_sunset\_hour\_LAT55\_LON13\_UTC+01:00.csv

An example of the first output lines calculated for Lat=51.5, Lon= -3.2 (location of Cardiff, Wales) is

Date	Solar_decl	Sunrise_solar	Sunset_solar	Sunrise_local	Sunset_local
2017-01-01	-22.92	08:08	15:51	08:24	16:07
2017-01-02	-22.83	08:07	15:52	08:24	16:08
2017-01-03	-22.73	08:07	15:52	08:24	16:10

The length of the day tabulated with [sunrise\\_sunset\\_time](#) is shorter than the one reported officially.<sup>10</sup> The difference is mainly due to the considerations above about the position of the sun near the horizon and can be of a few minutes in the morning and in the evening. At this time the net solar energy contribution for power or heat generation is nil anyway.

<sup>10</sup> for sunrise and sunset times for any location on earth see, for example, the website [www.timeanddate.com/worldclock](http://www.timeanddate.com/worldclock)

## 1.3 Solar position in the sky

### 1.3.1 The celestial sphere, coordinate systems

The position of the sun in the sky seen from any location on earth at any date and time can be calculated quite easily with help of a few basic astronomic considerations. Several algorithms are available with different degrees of precision.

The position of any astronomic object, such as the sun, on the celestial sphere may be described by two commonly used coordinate systems: the horizontal coordinate system and the equatorial system. The horizontal coordinate system is generally more convenient, in particular for the description of the position of the sun relative to fixed flat-plane collectors.

In the **horizontal coordinate system** the earth is assumed to be fixed and the horizon of the observer represents the fundamental plane. A vertical line originating at the center of earth and proceeding through the observer into the sky points to the **zenith**  $Z$ , while the plane perpendicular to that line, or the corresponding great circle, is the **horizon**. The horizon divides the sphere into two hemispheres, an upper hemisphere visible to the observer and a lower hemisphere hidden under the horizon.

The motion of space objects, in our case – the sun, is considered to take place on the **celestial sphere**, a sphere with undefined radius surrounding the earth. On the celestial sphere the position of every object is defined by two angles, the **elevation** (also called **altitude** or **height**)  $\alpha$  of the object above the horizontal plane and the **azimuth**  $A$ , that is, the angle between the line of sight projected from the body on the horizontal plane and due North or South (Figure 1-3).

The elevation angle has range  $[-90^\circ..90^\circ]$ , though negative angles point to objects that are below the horizon and therefore are not visible. The azimuth angle  $A$  has range  $[0^\circ..360^\circ]$  and is usually counted in a clockwise fashion with  $A = 0^\circ$  meaning North and  $90^\circ$ ,  $180^\circ$ ,  $270^\circ$  East, South, and West respectively. In solar engineering practice due South is sometimes taken as reference  $0^\circ$  (in the Northern hemisphere) with angles increasing clockwise towards the West, so that  $-180^\circ$  and  $+180^\circ$  both correspond to direction North,  $-90^\circ$  is East and  $+90^\circ$  is West.

The **solar zenith angle**  $z$  is the angle between the solar elevation and the Zenith:

$$z = 90^\circ - \alpha \quad (1-10)$$

### 1.3.2 Calculation of solar azimuth and elevation angle

In the horizontal coordinate system the position of the sun on the sky dome is indicated by the azimuth and elevation angles  $A$  and  $\alpha$ . The solar elevation angle  $\alpha$  of the sun over the horizon at any given location and time is computed from the geographical latitude  $\phi$ , the solar declination  $\delta_s$  ( $\rightarrow$  1.2.7) and the hour angle referred to the apparent solar time  $h_s$  ( $\rightarrow$  1.2.6):

$$\sin \alpha = \sin \phi \cdot \sin \delta_s + \cos \phi \cdot \cos \delta_s \cdot \cos h_s \quad (1-11)$$

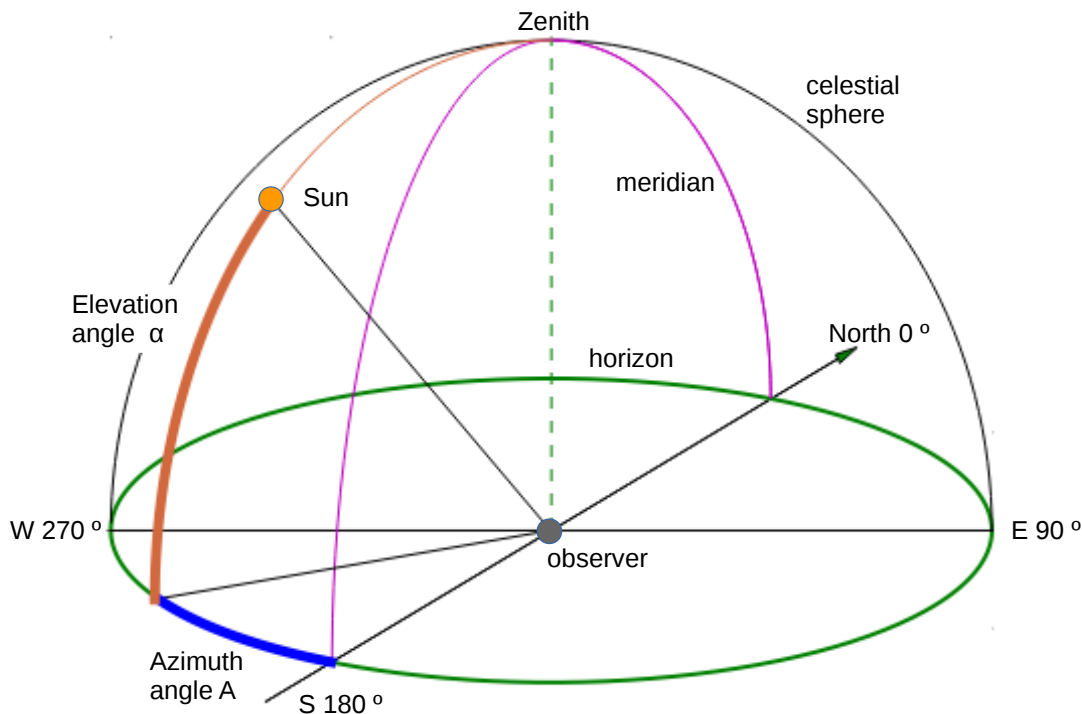


Figure 1-3: Coordinate system for the position of the Sun on the celestial sphere with indication of the angles for azimuth  $A$  and elevation  $\alpha$

At solar noon  $h_s = 0^\circ$ , the cosine term is  $=1$  and the function has a maximum ( $\varphi$  is constant and the approximated  $\delta_s$  is the same for the whole day). The solar elevation  $\alpha_s$  is then equal to  $90^\circ - |\varphi + \delta_s|$ .

The three parameters  $\varphi$ ,  $\delta_s$ , and  $h_s$  are unique for any location on earth on a particular day and time. A computed negative value of  $\alpha_s$ , which in Equation (1-11) could only depend on the  $h_s$  term, indicates that the sun is below the horizon, i.e., it is nighttime.

Equation (1-11) is related to Equation (1-9) for the sunrise hour angle  $A_{SR}$ . At sunrise and sunset the elevation of the sun on the horizon is exactly 0. By rearranging the terms of Equation (1-11) it results

$$\sin \phi \cdot \sin \delta_s + \cos \phi \cdot \cos \delta_s \cdot \cosh_s = 0$$

$$\cosh_s = -\frac{\sin \phi \cdot \sin \delta_s}{\cos \phi \cdot \cos \delta_s}$$

$$\cosh_s = -\tan \phi \cdot \tan \delta_s$$

which is the same as Equation (1-9) for  $A_{SR} = h_s$ .

The **solar azimuth angle**  $A$  is function of the location latitude  $\varphi$ , the solar declination  $\delta_s$ , the hour angle  $h_s$ , and the solar elevation  $\alpha$ :

$$\cos(A) = \frac{\sin \alpha \cdot \sin \phi - \sin \delta_s}{\cos \alpha \cdot \cos \phi} \quad (1-12)$$

with

$$\begin{aligned} \cos(A) &\geq 0 & 0^\circ \leq A \leq 90^\circ \\ \cos(A) &\leq 0 & 90^\circ \leq A \leq 180^\circ \end{aligned}$$

The azimuth equation can also be expressed as

$$\sin(A) = \frac{\cos \delta_s \cdot \sin h_s}{\cos \alpha} \quad (1-13)$$

The solar elevation  $\alpha$  depends on the three terms  $\varphi$ ,  $\delta_s$ , and  $h_s$ , so that also the solar azimuth ultimately depends on them. At a given latitude, day, and time solar azimuth and elevation are therefore uniquely determined.

The calculation of the solar azimuth angle with Equation (1-13) leads to a potential problem when the absolute value of  $A$  is greater than  $90^\circ$ . The arcsine function would return an angle  $<90^\circ$  since both  $\sin(A)$  and  $\sin(180^\circ - A)$  have the same value. A solution algorithm would first need

to consider the sunrise/sunset times and assess whether the sun is north or south of the East–West line and the true value of  $A$  lower or higher than  $90^\circ$ . With this information the value of  $A$ , if needed, can be corrected.

The function of Program\_Code 1-18 calculates the azimuth and the elevation coordinates of the sun for given input latitude, solar declination, and hour angle. Most functions described in this primer refer to it.

#### Program\_Code 1-18: function, solar azimuth and elevation

<b>package</b>	<b><code>solprim.solartimeposition</code></b>
<b>function</b>	<b><code>solar_azim_elev</code></b>
<b>description</b>	calculate the azimuth and the elevation coordinates of the sun for given latitude, solar declination, and hour angle
<b>parameters</b>	<b><code>sol_decl_rad</code></b> : solar declination in radians, float, default value =0 <b><code>hour_angle_rad</code></b> : hour angle from solar noon in radians (noon is zero, morning negative), float, default value =0 <b><code>latitude_rad</code></b> : latitude in radians (North is positive), float, default value =0
<b>returns</b>	tuple with - elevation angle in decimal degrees, float $[0^\circ..90^\circ]$ - azimuth angle in decimal degrees, float $[-180^\circ..+180^\circ]$ - elevation angle in radians, float $[0..\pi/2]$ - azimuth angle in radians, float $[-\pi..+\pi]$ - elevation angle sine, float $[0..1]$ - elevation angle cosine, float $[0..1]$ - azimuth angle cosine, float $[0..1]$

The function `solar_azim_elev` calculates the azimuth and elevation angles for a given latitude, solar declination, and hour angle.<sup>11</sup>

This function is used in tight loop iterations and must perform as efficiently as possible. Its values are obtained and passed to other iterative functions. For this reason its arguments are passed in radians and the result is returned in both decimal degrees and radians. While degrees are directly understandable by the user, e.g., in printouts,

<sup>11</sup> The used algorithm is based on the description by Muhammad Iqbal, An Introduction to Solar Radiation, Academic Press, 1983

radians are passed as input to other equations without repeated conversions between the two angular measures. For the same reason the output tuple includes also the sine and cosine of the elevation angle and the cosine of the azimuth angle, which are used in the [incidence\\_factor](#) function (→ Program\_Code 1-27) and thus do not need to be calculated again. The precision of the result is discussed in → Section 1.3.5.

### 1.3.3 Solar position graphical plots

The following Python scripts make use of the basic solar time and position functions to plot maps for practical applications.

Program\_Code 1-19: script, Cartesian plot of solar azimuth, elevation for given latitude, day

<b>script</b>	<a href="#">solpath_Cartesian_azim_elev</a>
<b>description</b>	for a given latitude, year, month, and day produce a Cartesian plot of the solar elevation vs azimuth
<b>parameters</b>	<b>year</b> : integer <b>month</b> : integer <b>day</b> : integer <b>latitude</b> : latitude in decimal degrees (North is positive), float <b>calc_step</b> : time step for calculation and plotting in minutes, integer
<b>returns</b>	plot for solar elevation vs azimuth at given latitude and date

The script [solpath\\_Cartesian\\_azim\\_elev](#) makes use of the function [solar\\_azim\\_elev](#) to calculate solar azimuth and elevation over the day and make a Cartesian plot. The required parameters are input manually at the beginning of the script. The time is considered to be true solar time and no corrections for longitude, Equation of Time are made. The maximum solar elevation is therefore always at 12 noon.

Two output examples are shown in Figure 1-4 and Figure 1-5.

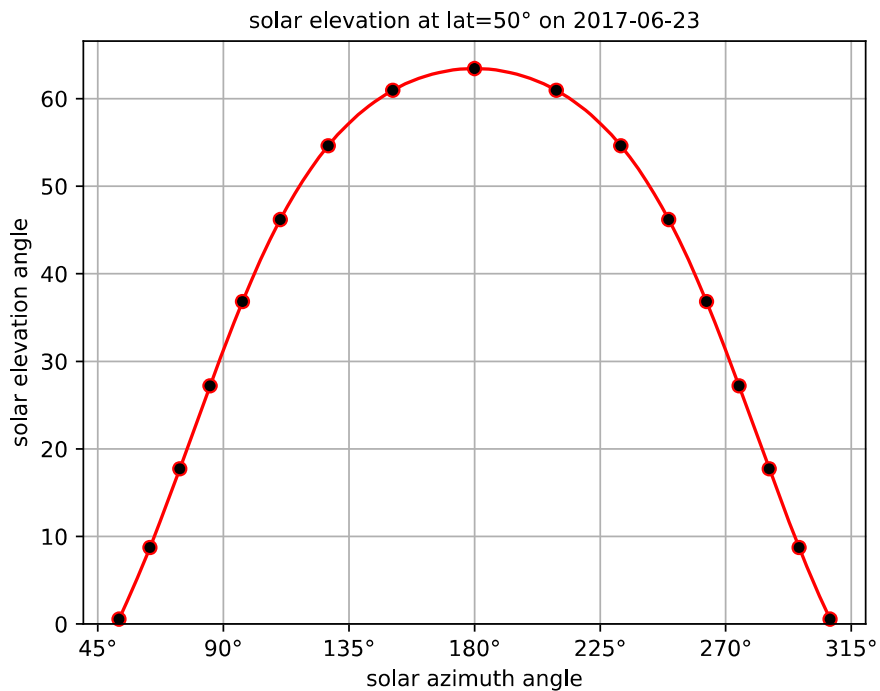


Figure 1-4: Solar elevation vs azimuth angle at Lat = 50°N at the time of the Summer solstice. At noon the sun reaches the highest position in the sky for the year.

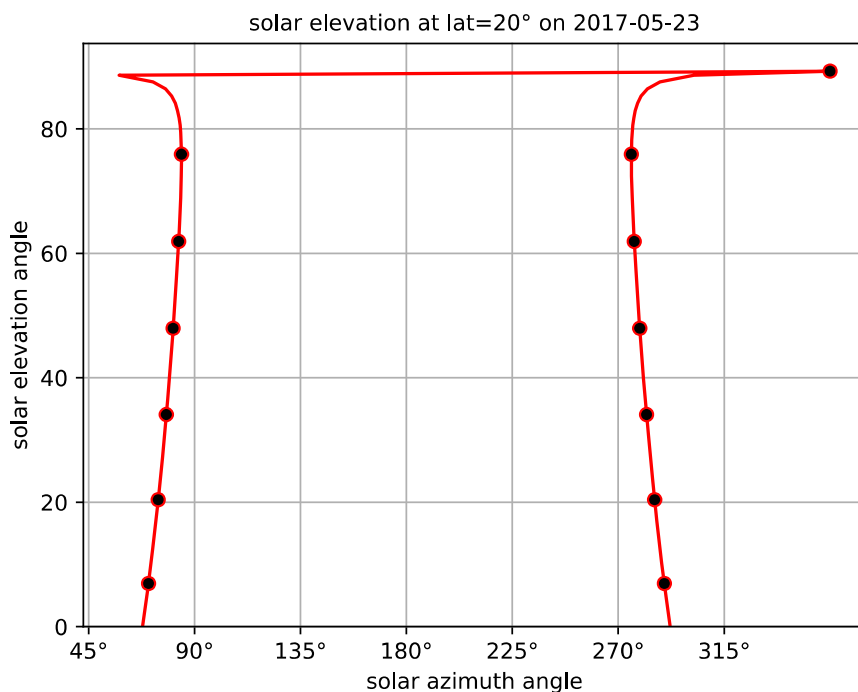
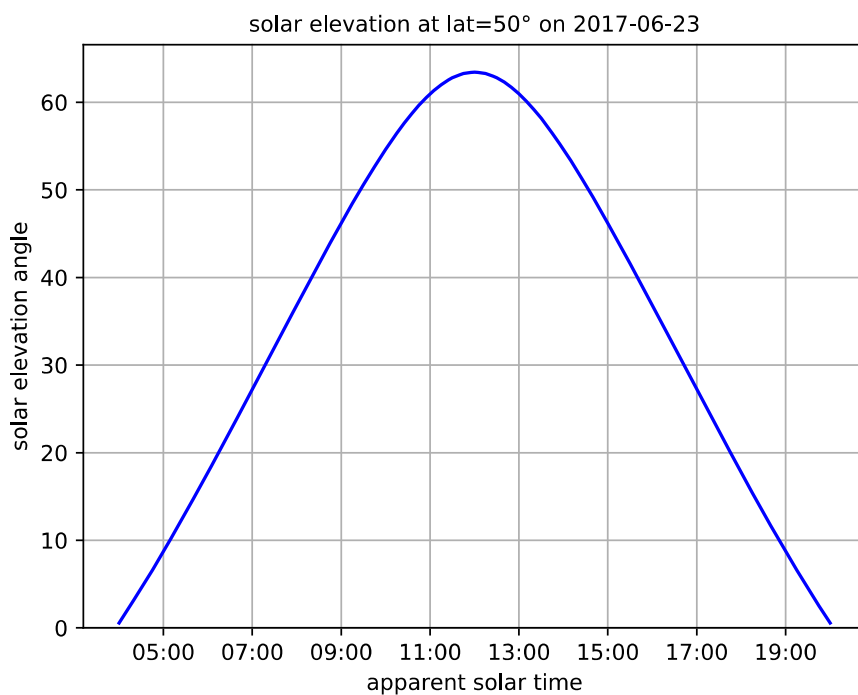


Figure 1-5: At Latitude = 20° in May the sun gets near the Zenith at noon and in a few minutes between 11am and 1pm it crosses several meridians in Northern direction, leading to this strange pattern. The point to the upper right is for 12noon.

### Program\_Code 1-20: script, Cartesian plot of solar time, elevation at latitude, day

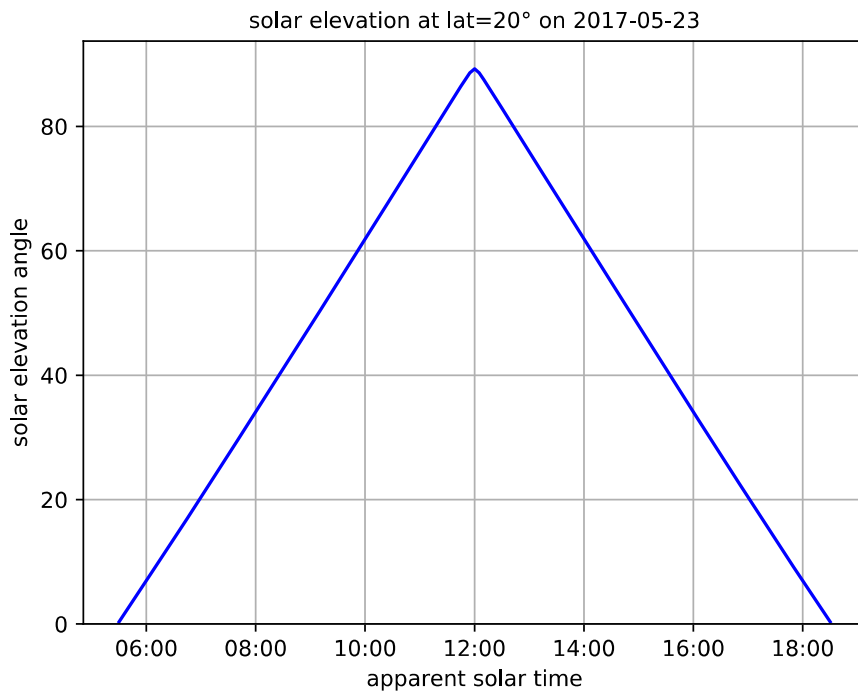
<b>script</b>	<a href="#">solpath_Cartesian_time_elev</a>
<b>description</b>	for a given latitude, year, month, and day produce a Cartesian plot of the solar elevation vs hour of the day
<b>parameters</b>	<b>year</b> : integer <b>month</b> : integer <b>day</b> : integer <b>latitude</b> : latitude in decimal degrees (North is positive), float <b>calc_step</b> : time step for calculation and plotting in minutes, integer
<b>returns</b>	plot for solar elevation vs hour of the day at given latitude and date

The scripts [solpath\\_Cartesian\\_azim\\_elev](#) and [solpath\\_Cartesian\\_time\\_elev](#) are similar, the second one plots the solar elevation as function of the time instead of the azimuth, Figure 1-6 and Figure 1-7.



*Figure 1-6: Solar elevation vs time at Latitude = 50°N at the time of the Summer solstice (the azimuth-elevation path is shown in Figure 1-4)*





*Figure 1-7: Solar elevation vs time at Latitude = 20°North in May. This curve is unambiguous, though the path is the same as the one shown in Figure 1-5.*

Program\_Code 1-21: script, Cartesian plot of solar azimuth, elevation at given latitude and different months of the year

<b>script</b>	<b><code>solpath_Cartesian_mult</code></b>
<b>description</b>	for a given latitude produce a Cartesian plot of the solar elevation vs azimuth for the same day in different months
<b>parameters</b>	<b>year</b> : integer <b>month_list</b> : months to be plotted, integer list <b>day_ref</b> : day to be plotted for each month, integer <b>latitude</b> : latitude in decimal degrees (North is positive), float <b>calc_step</b> : time step for calculation and plotting in minutes, integer
<b>returns</b>	plot for solar elevation vs azimuth at given latitude and chosen day for the selected months

The script `solpath_Cartesian_mult` allows the comparison of the solar path for different months of the year, at a selected day. The seasonal differences in the solar elevation are clearly visible (Figure 1-8 and Figure 1-9).

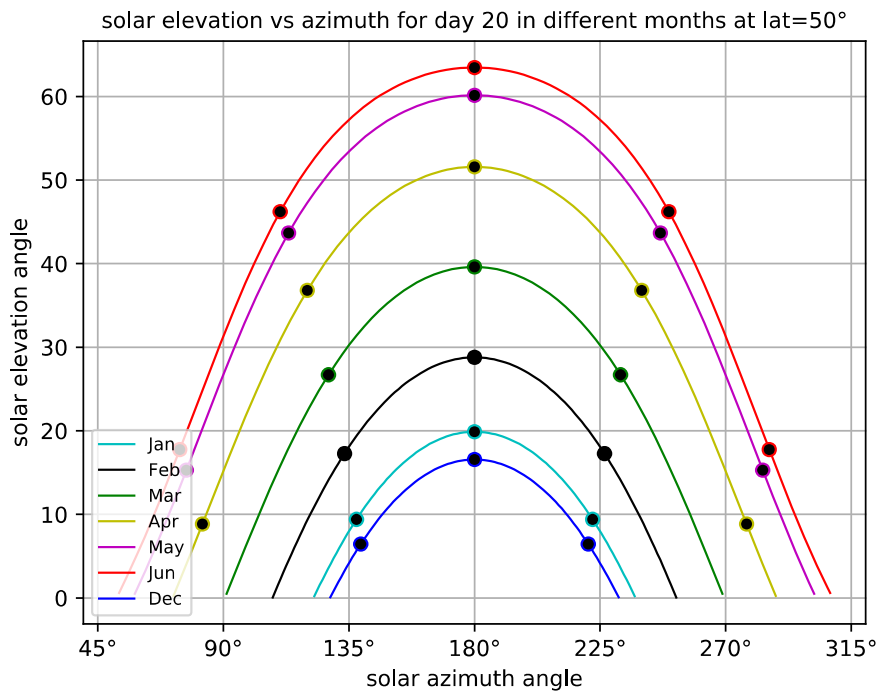


Figure 1-8: Solar azimuth-elevation plot for Lat = 50°N on day 20 in different months of the year

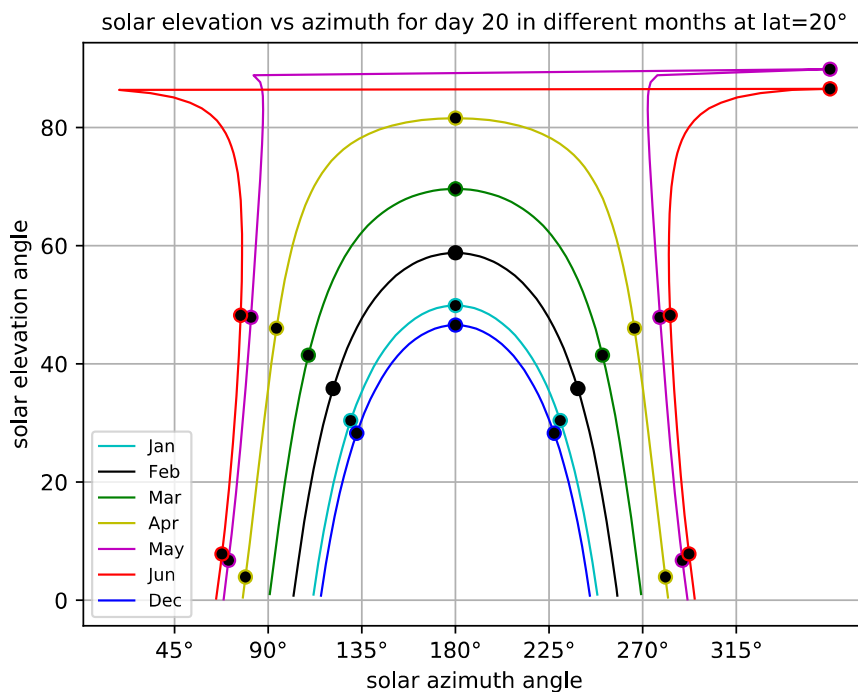
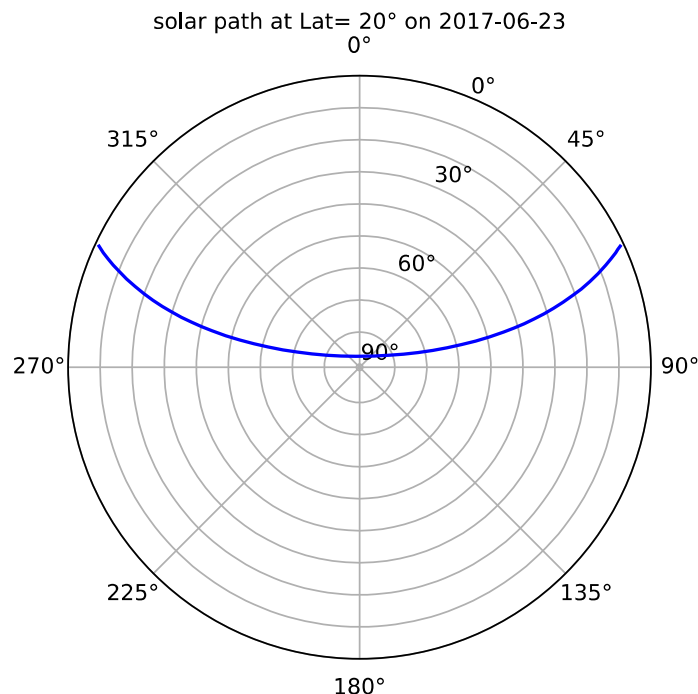


Figure 1-9: Solar azimuth-elevation plot for Lat = 20°N on day 20 in different months of the year. The strange pattern for elevations near the Zenith appears for May and June.

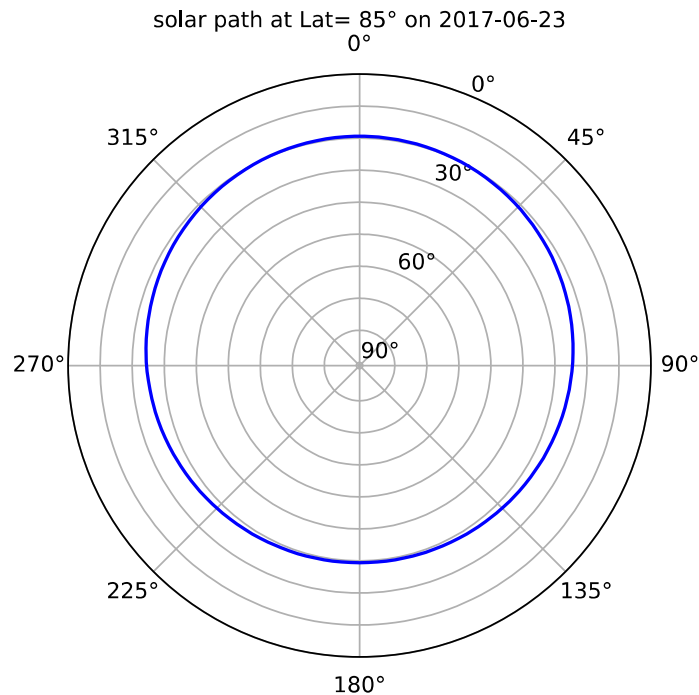
### Program\_Code 1-22: script, solar path polar plot at arbitrary location and day

<b>script</b>	<b><code>solpath_polar</code></b>
<b>description</b>	for a given latitude, year, month, and day produce a polar plot of solar elevation vs azimuth
<b>parameters</b>	<b>year</b> : integer <b>month</b> : integer <b>day</b> : integer <b>latitude</b> : latitude in decimal degrees (North is positive), float <b>calc_step</b> : time step for calculation and plotting in minutes, integer
<b>returns</b>	polar plot for solar elevation vs azimuth at given latitude and given day

The polar representation provides a better feeling than a Cartesian plot for the azimuth on a circle around the observer and not only on an axis. The function `solpath_polar` produces a polar plot for the a chosen latitude and day of year (Figure 1-10 and Figure 1-11).



*Figure 1-10: At Lat =20°N on June 23 the sun is all the time North of the East-West line (90° – 270°). On May 20 the solar path would reach elevation = 90° as shown in the Cartesian multiplot of Figure 1-9.*



*Figure 1-11: At Latitude = 85°N at the Summer solstice the sun neither rises or sets. It rotates around the observer between elevation angles ~20° and ~30°*

Program\_Code 1-23: script, solar path polar plot at given latitude, different months of the year

<b>script</b>	<a href="#"><code>solpath_polar_mult</code></a>
<b>description</b>	for a given latitude, year, month, and day produce a polar plot of the solar elevation vs azimuth
<b>parameters</b>	<b>year</b> : integer <b>month_list</b> : months to be plotted, integer list <b>day_ref</b> : day to be plotted for each month, integer <b>latitude</b> : latitude in decimal degrees (North is positive), float <b>calc_step</b> : time step for calculation and plotting in minutes, integer
<b>returns</b>	polar plot for solar elevation vs azimuth at given latitude and selected day for different months

The script [`solpath\_polar\_mult`](#) makes a polar plot of the azimuth-elevation solar paths on the same day for several months in the year, as shown in Figure 1-12 and Figure 1-13.

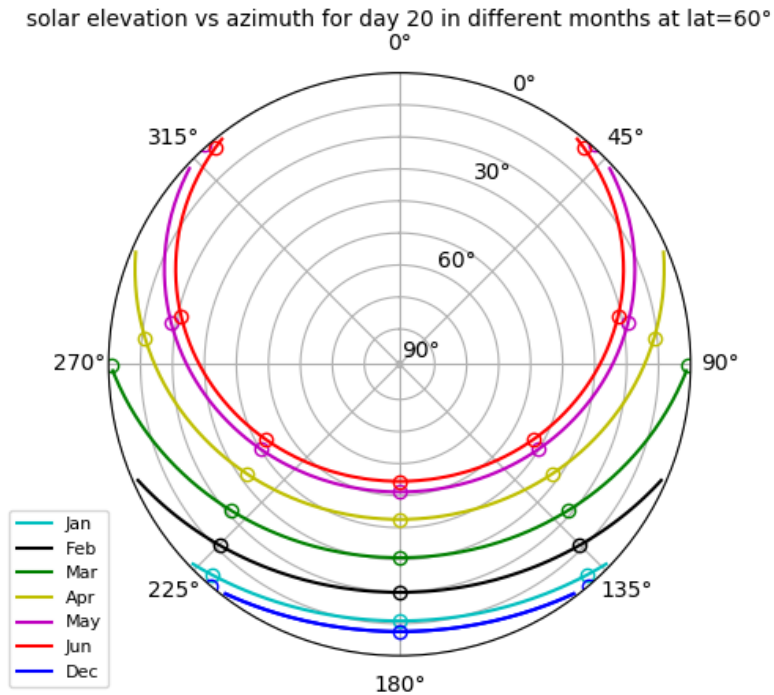


Figure 1-12: At Latitude = 60°N in Midsummer the sun rises early and sets late and rotates around the observer. In Winter the sun is low, the day only few hours long.

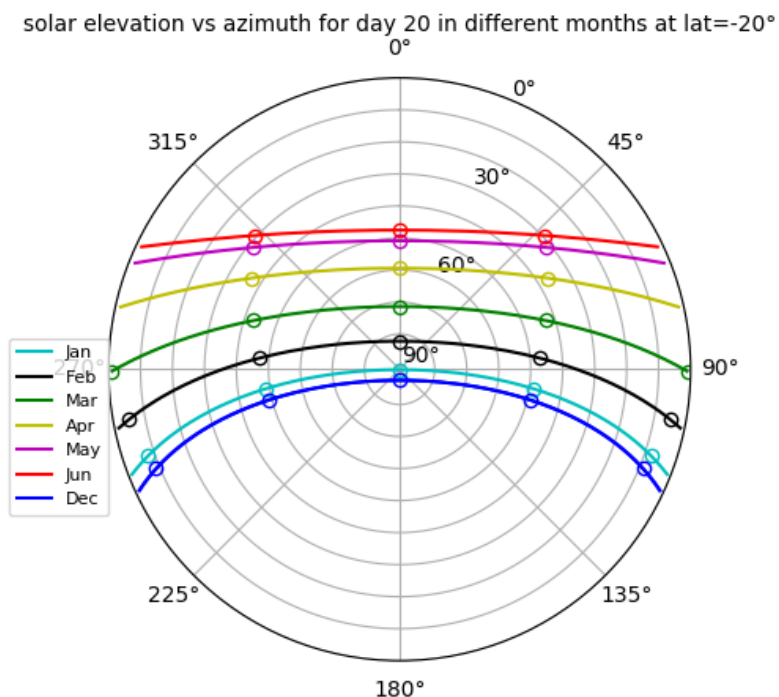


Figure 1-13: Near the equator, at Lat= 20°S, the sun is high in the sky all the time. In January and in November at noon the sun reaches the Zenith.

### 1.3.4 High-precision algorithms for the solar position calculation

The solar position calculations illustrated so far are based on simplified assumptions. Their result differs to some extent from the exact solar position calculated by more precise algorithms. The suitability of a solution depends on the purpose for the calculations. In some cases simplified calculations are fully acceptable, in others more precise results are required.

A widespread high-precision algorithm for the calculation of the solar position at any day and time is the NREL Solar Position Algorithm.<sup>12</sup> This algorithm is implemented in the Python **pvlib** package<sup>13</sup>, is available online (Online\_Resource 1-4) and its code is public.<sup>14</sup>

The following script (Program\_Code 1-24) produces a table with azimuth, elevation, and Equation of Time generated by the function **get\_solarposition** in the **pvlib** package (which must be installed in the system) and with the function **solar\_azim\_elev**.

Program\_Code 1-24: script, compare azimuth, elevation angles with the NREL Solar Position Algorithm

<b>script</b>	<b>test_solpos_pvlib</b>
<b>description</b>	at latitude, longitude calculate the solar position with the NREL algorithm and the <b>solprimer</b> package for 8760 hours in the year
<b>parameters</b>	<b>latitude</b> : latitude in decimal degrees (North is positive), float <b>longitude</b> : longitude in decimal degrees (East is positive), float
<b>returns</b>	table in <b>.csv</b> format with 8760 hourly tabulated values for Equation of Time, solar azimuth and elevation, calculated with the NREL Solar Position algorithm and with the <b>solar_azim_elev</b> function

The output of the **test\_solpos\_pvlib** script is a table in **.csv** format that can be opened and examined with a spreadsheet (→ Section 1.3.5).

12 NREL, Solar Position Algorithm for Solar Radiation Applications, by Ibrahim Reda and Afshin Andreas, NREL/TP-560-34302, Revised January 2008

<http://www.nrel.gov/docs/fy08osti/34302.pdf>

13 pvlib is the NREL library of solar PV functions for Python

<http://pvlib-python.readthedocs.io>

14 Code available at <http://rredc.nrel.gov/solar/codesandalgorithms/spa/>

### 1.3.5 Error margin in solar time and position calculation

The sun, as seen from the earth, has an angle of  $0.5^\circ$  and takes about 2 minutes to move  $1^\circ$  arc in the sky. For solar energy applications any calculation with a time error in the order of 1-2 minutes is acceptable, also for equipment that needs to track the solar position, such as measuring instruments or parabolic collectors. In the case of flat-plate collectors errors of a few minutes or few degrees in the exact solar position do not have a significant influence on the overall result of the energy totals for the day or longer time period.

The script of Program\_Code 1-24 generates a table with hourly solar azimuth and elevation values over a full year as calculated by the `solar_azim_elev` function and the NREL Solar Position Algorithm implemented in the `pvlib` package. For example, in a test for latitude= $37^\circ\text{N}$  the results by the simple algorithms of this primer's package match quite well those of the more sophisticated algorithms, with a maximum difference in azimuth and elevation over the year  $< 1.3^\circ$ . The largest errors in azimuth take place at latitudes near the equator during the mid-day hours, when solar elevation is very high and the sun crosses several longitude degrees in few minutes.

On the other hand, the `pvlib` function on a modern laptop computer needs more than 60 seconds to process the 8760 hours in the year, while this primer's `solar_azim_elev` thanks to its simpler algorithms does the same calculation in ca. 0.065 seconds, i.e., it is about 1000 times faster.

The choice of a particular calculation routine is dictated by several aspects. The strongest influence on solar energy availability on earth is given by climate and weather. Weather, in particular, cannot be forecast with high precision and can only be described with statistical means provided that long data series are available (Typical Meteorological Year, → Section 2.2). Minor errors in the calculation of the solar position have a much smaller influence on the total simulated yield of a solar array than the natural variation of weather. The solar position with respect to the earth for solar energy applications in many cases can therefore be adequately described with a few parameters and simple geometrical models. For higher precision relevant models should be used.

Tests on the compared outcome for solar radiation collected on tilted flat plates are presented in → Section 4.1, Table 4-1.



**Online\_Resource 1-4: Solar position calculator (NREL)**

The National Renewable Energy Laboratory (NREL) of the US department of energy maintains a very precise online calculator, based on the NREL Solar Position Algorithm:

<https://www.nrel.gov/midc/solpos/spa.html>

This calculator can generate data series.

**Online\_Resource 1-5: Solar position calculator (NOOA)**

The US National Oceanic & Atmospheric Administration (NOOA) maintains a solar position calculator

<https://www.esrl.noaa.gov/gmd/grad/solcalc/>

**Online\_Resource 1-6: Solar position calculator (timeanddate.com)**

An intuitive and clearly designed website with solar and moon data is found at <https://www.timeanddate.com/astronomy/>

**Online\_Resource 1-7: Cartesian and polar solar elevation plot (Univ. of Oregon)**

The Solar Radiation Monitoring Laboratory at Univ. of Oregon maintains an online facility for the generation of solar plots with flexible parameters. Solar path diagrams in Cartesian coordinates can be generated at

<http://solardat.uoregon.edu/SunChartProgram.php>

The website for the generation of polar sun charts is

<http://solardat.uoregon.edu/PolarSunChartProgram.php>

Different solar radiation resources are found at

<http://solardat.uoregon.edu/SolarRadiationBasics.html#Fig3>

## 1.4 Solar radiation on the earth surface

### 1.4.1 Influence of the atmosphere on solar radiation

The position of the sun as seen at any time and from any location on the earth can be calculated with help of basic geometrical and physical considerations without any random aspects: at a certain location and given time the sun is at a certain point in the sky. On the other hand, this is not sufficient to assess the amount of solar energy that can be collected on the earth surface. For this it is necessary to consider the influence of the atmosphere and of the weather. These can be described analytically only to a limited extent. In particular, the effect of weather is dealt with in practice with help of statistical data from long-term observations.

The solar radiation measured on the earth surface is lower than that outside the atmosphere. The reduction is due to the effects of reflection, absorption, and scattering in the air. In particular, the reduction of the solar radiation in the atmosphere is mainly caused by water vapor (H<sub>2</sub>O), ozone (O<sub>3</sub>), oxygen (O<sub>2</sub>), and carbon dioxide (CO<sub>2</sub>). The filtering effect of these gases is strongly selective and encompasses only some parts of the solar spectrum. This is clearly illustrated in Figure 1-2, where the solar radiation density around the wavelengths 1200, 1400, and 1800-1900 nanometer has been notably reduced after crossing the atmosphere.

The intensity of the solar radiation reaching the earth surface depends on the elevation angle of the sun. For low elevation angles the direct solar radiation needs to cross a larger section of the atmosphere to reach any point on the ground, so the attenuation factor is larger. The length of the solar radiation path through the atmosphere compared to the shortest path is indicated as **air mass** (*AM*). *AM*1, i.e., one air mass, indicates a vertical path from sea level to the outer atmospheric layer at standard barometric pressure and the sun directly overhead. *AM*=0 or *AM*0 refers to the solar radiation outside the atmosphere.

A simplified, and widely reported, relation links the air mass factor *AM* to the solar elevation  $\alpha$ :

$$AM = \frac{1}{\sin \alpha} \quad (1-14)$$

This equation does not hold for very low values of the solar elevation  $\alpha$  and cannot be computed for  $\alpha \approx 0$ .

The following relation provides a better fit, as it takes in consideration the curvature of the earth<sup>15</sup>

$$AM = \frac{1}{\sin \alpha + 0.50572(6.07995 + \alpha)^{-1.6364}} \quad (1-15)$$

The plots for AM obtained with Equation (1-14) and (1-15) are shown in Figure 1-14. The results are similar, though the exponential relation is better at very small elevation angles. Conventionally, AM is taken as =38 for  $\alpha \approx 0$ .

From the plot in Figure 1-14 it appears how for elevation angles  $\alpha < 5^\circ$  the AM factor increases fast and solar radiation is strongly attenuated. Under overcast very little solar radiation, if any, is converted to power.

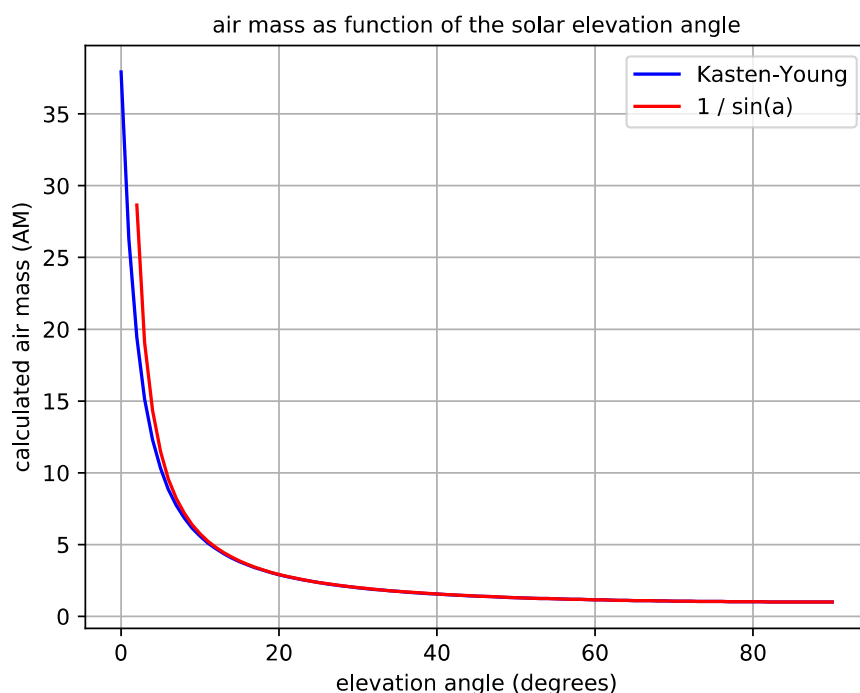


Figure 1-14: Clear-sky airmass coefficient as function of elevation calculated with Equation (1-14) and (1-15)

15 Equation by F. Kasten and A. T. Young quoted in D. Yogi Goswami, Principles of Solar Engineering, Third Edition, CRC Press, Taylor & Francis Group, 2015, Chap.2.5 "Estimation of Terrestrial Solar Radiation" and in Solar Energy, Delft University of Technology, 2014, [https://courses.edx.org/c4x/DelftX/ET.3034TU/asset/solar\\_energy\\_v1.1.pdf](https://courses.edx.org/c4x/DelftX/ET.3034TU/asset/solar_energy_v1.1.pdf)

From the plots of the solar elevation as function of day and time it can be quickly assessed how long the sun is high enough in the sky to provide useful power under the right weather conditions.

The intensity of direct solar irradiance at ground level for AM1 is about 70% of the extraterrestrial radiation value. For elevation angles  $\alpha < 90^\circ$  and assuming a direct proportionality between airmass and attenuation the relation is

$$B_n = B_0 \cdot 0.7 \cdot \frac{1}{AM}$$

that is, taking the simplified Equation (1-14) for the airmass as function of the elevation angle  $\alpha$ :

$$B_n = B_0 \cdot 0.7 \cdot \sin\alpha \quad (1-16)$$

An empirical relation provides a better estimation of the direct solar irradiance on the ground for an air mass coefficient AM, including the effect of the location altitude expressed in [km]:<sup>16</sup>

$$B_n = B_0 \cdot \left( (1 - 0.14 \cdot h) 0.7^{(AM^{0.678})} + 0.14 \cdot h \right) \quad (1-17)$$

In Figure 1-15 the beam radiation as a fraction of the extraterrestrial radiation ETR is presented in relation to the airmass factor as it results from Equations (1-16) and (1-17) and in Figure 1-16 as function of the elevation angle. The exponential function returns a stronger beam radiation component at low angles in comparison to the  $\sin(\alpha)$  profile and provides a better fit of the observed data.

---

16 E. G. Laue, "The measurement of solar spectral irradiance at different terrestrial elevations," Solar Energy, vol. 13, pp. 43-50, 1970 quoted in Solar Energy, Delft University of Technology, 2014

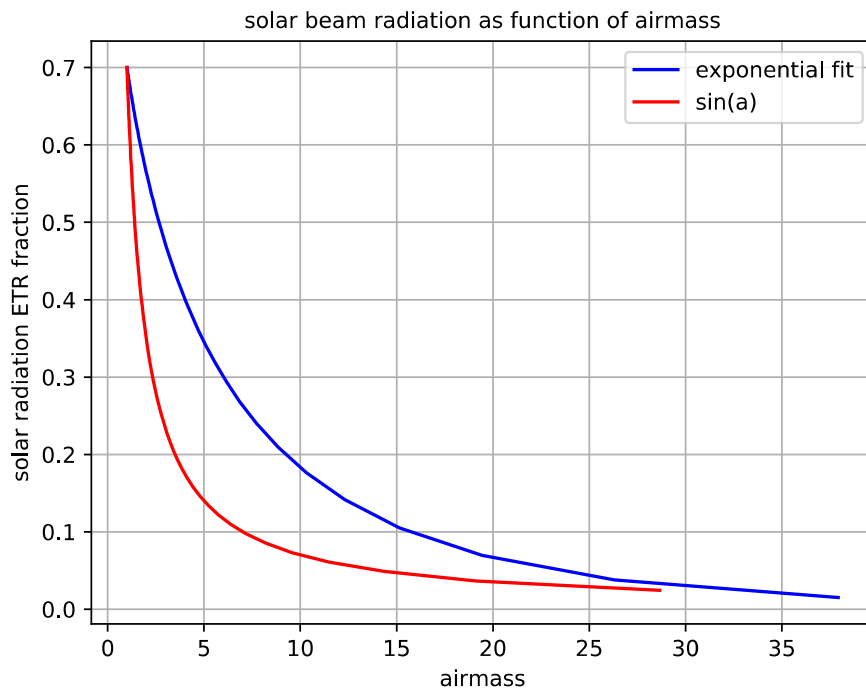


Figure 1-15: Beam radiation as a fraction of the extraterrestrial radiation ETR in relation to the airmass factor

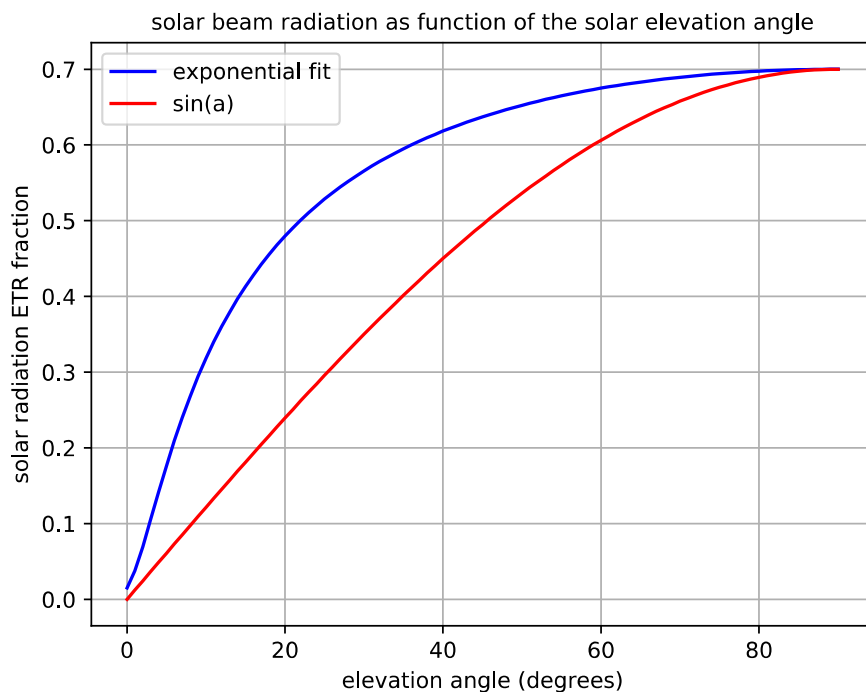


Figure 1-16: Solar beam radiation as function of the solar elevation angle. The curve of Equation (1-17) indicates a stronger beam radiation component compared with the  $\sin(\alpha)$  function, which indicates a strong attenuation.

### 1.4.2 Global, direct, and diffuse irradiation

The total solar radiation reaching the earth surface has two components: a **direct (beam)** component of the solar rays after crossing the atmosphere and a **diffuse** component caused by scattering effects in the sky. In order to calculate the amount of solar energy collected by flat surfaces it is necessary to know the intensity of these components separately. Several physical models describe the relation between direct and diffuse solar radiation, though these apply only for clear weather conditions.

In practical work it is necessary to refer to measured values for the intensity of the different solar radiation components. These are recorded at meteorological stations for thousands of locations worldwide and a large amount of datasets is openly available (→ Section 2.2).

The measured components in  $[\text{W} \cdot \text{m}^{-2}]$  are defined as follows:

- **Direct horizontal radiation** ( $B_h$ ): Shortwave radiation (0.2–3.0  $\mu\text{m}$ ) sourced from a solid angle with  $5^\circ$  aperture centered around the sun's disk falling on a horizontal surface on the ground.
- **Direct normal (beam) radiation** ( $B_n$ ): Shortwave radiation (0.2–3.0  $\mu\text{m}$ ) sourced from a solid angle with  $5^\circ$  aperture centered around the solar disk and falling on a surface normal to the direction of the radiation.
- **Diffuse horizontal radiation** ( $D_h$ ): Shortwave isotropic radiation (0.2–3.0  $\mu\text{m}$ ) from the whole open sky less the direct solar radiation component  $B_h$ .
- **Total isotropic radiation** (= from all directions) indicated as  $G_h$ .

The parameters  $B_h$ ,  $B_n$ ,  $D_h$ , and  $G_h$  are mutually related. The total (global) horizontal radiation is the sum of the direct horizontal and the diffuse horizontal radiation components:

$$G_h = B_h + D_h \quad (1-18)$$

the direct horizontal and the direct normal radiation are related by

$$B_h = B_n \cdot \sin \alpha \quad (1-19)$$

with  $\alpha$  the sun elevation angle above the horizon.

Meteorological stations collect and record solar radiation intensity parameters on a daily or hourly basis. These parameters are indicated as

- **GHI** (Global Horizontal Irradiation): the sum of the direct and diffuse components of solar radiation collected over an horizontal unit surface [ $1 \text{ m}^2$ ] since the last recording instance
- **DNI** (Direct Normal Irradiation, also Beam Irradiation): the solar radiation component over a plane normal to the direction of the sun
- **DHI** (Diffuse Horizontal Irradiation): the solar radiation component scattered by the atmosphere over an horizontal surface.

In other words, GHI, DNI, and DHI are the integral values over a regular period of time  $T$  (hour, day, month, or year) of the instant solar radiation values  $G_h$ ,  $B_n$ , and  $D_h$ :

$$[GHI, DNI, DHI] = \int_0^T [G_h, B_n, D_h] \cdot dt \quad (1-20)$$

GHI, DNI, and DHI are measured and recorded over an horizontal and a normal plane to the direction of the solar radiation. In practice, their values are recorded on a hourly basis and then added up to form daily, monthly and yearly totals. The long-term averages of these parameters are reported as tabulated values in reference textbooks and on solar resource maps. The GHI parameter over the year or per day is the principal reference to indicate the availability of solar energy resource at any location.

A note about terminology. **Solar irradiance** is the measure of the instant **power** from the sun over a unit surface, typically reported in SI units as Watt per square meter,  $\text{W} \cdot \text{m}^{-2}$ . **Solar insolation** is a measure of the **energy** from the sun on a unit surface and over a longer time period, it is reported in Watt-hour per square meter and the time period, e.g.,  $[\text{W} \cdot \text{m}^{-2} \cdot \text{day}^{-1}]$ . However, the use across literature isn't yet fully consistent.

### 1.4.3 Clear-sky radiation

Starting with the solar extraterrestrial radiation and with the considerations of Sections 1.4.1 and 1.4.2 it is possible to define a simple clear-sky direct and diffuse radiation profile. This will be useful as reference for comparison with real data, to understand the relative effect of the clean atmosphere and of weather on solar radiation at a given location. Some examples will be shown in → Section 2.3.1, in particular Program\_Code 2-20.

The function of Program\_Code 1-25 generates hourly lists of direct, diffuse, and global clear-sky solar radiation as function of latitude in the TMY (Typical Meteorological Year) format, described in → Section 2.2. This ensures their compatibility with recorded solar insolation and weather data.

#### Program\_Code 1-25: function, clear-sky radiation at latitude, height

<b>package</b>	<b><code>solprim.solartimeposition</code></b>
<b>function</b>	<b><code>clearsky_radiation</code></b>
<b>description</b>	calculate the direct, diffuse, and global radiation components of solar energy for all hours in the year under clear-sky conditions
<b>parameters</b>	<b>latitude</b> : latitude in decimal degrees (North is positive), float, default =0 <b>height</b> : height over sea level in m, float, default = 0
<b>returns</b>	tuple with - list with generated clear-sky global horizontal radiation on hourly basis in W/m2, integer [0..8759] - list with generated clear-sky direct normal radiation on hourly basis in W/m2, integer [0..8759] - list with generated clear-sky diffuse horizontal radiation on hourly basis in W/m2, integer [0..8759]

The function `clearsky_radiation` generates hourly values of GHI, DNI, DHI as they would be measured at any point on earth defined by its latitude and under the assumption of perfect clear-sky conditions. The DNI, DHI values must therefore be near those recorded in the TMY for any location under very clear days.



For each day of the year the function first calculates the extraterrestrial radiation (ETR), which, due to the ellipticity of the earth orbit, varies around the solar constant by  $\pm 3\%$  depending on the day of the year. Further are considered the daylight hours of each day. The solar elevation angle is calculated with geometrical relations ( $\rightarrow$  Section 1.3.2). The airmass and the attenuation factor are then calculated as a function of the elevation angle from Equation (1-15) and (1-17) respectively, see  $\rightarrow$  Section 1.4.1.

At negative latitude values, i.e., for locations South of the equator, the clearsky radiation profile has a minimum in June and a maximum in December, following the seasons in the Southern hemisphere.

For the estimation of the intensity of diffuse radiation as a function of the beam radiation under clear-sky conditions there isn't yet any widely accepted relation. An approximate and widely used assumption is to take the intensity of diffuse radiation to be 10% of the beam radiation.

In order to facilitate the study of the `clearsky_radiation` function several intermediate variables are also tabulated and passed to the calling function. For example, the script `clearsky_profile` (Program\_Code 1-26) plots detailed tabulated data on hourly basis and dhi/ dni/ dhi for every day. An example of tabulated hourly output is the following.

HRA	azim (deg)	elev (deg)	ETR [W/m <sup>2</sup> ]	air mass (aam)	factor	DNI [W/m <sup>2</sup> ]	GHI [W/m <sup>2</sup> ]	DHI [W/m <sup>2</sup> ]
-180	0	-83	0	0	0	0	0	0
-165	65.98	-74.88	0	0	0	0	0	0
-150	82.55	-62.34	0	0	0	0	0	0
-135	91.14	-49.38	0	0	0	0	0	0
-120	97.73	-36.44	0	0	0	0	0	0
-105	103.85	-23.68	0	0	0	0	0	0
-90	116.85	0.05	76.7032	37.1975	0.0009	1	0	0
-75	121.17	6.3	1413.478	8.4789	0.2199	310	65	31
-60	130.27	16.85	1413.478	3.4138	0.4412	623	243	62
-45	141.46	25.92	1413.478	2.2788	0.5367	758	407	75
-30	155.24	32.76	1413.478	1.8437	0.5833	824	528	82
-15	171.4	36.51	1413.478	1.6776	0.6031	852	592	85
0	188.6	36.51	1413.478	1.6776	0.6031	852	592	85
15	204.76	32.76	1413.478	1.8437	0.5833	824	528	82
30	218.54	25.92	1413.478	2.2788	0.5367	758	407	75
45	229.73	16.85	1413.478	3.4138	0.4412	623	243	62

### Program\_Code 1-26: script, generate table and plot clear-sky radiation at latitude, height

<b>script</b>	<b>clearsky_profile</b>
<b>description</b>	calculate the direct, diffuse, and global radiation components of solar energy for all hours in the year under clear-sky conditions
<b>parameters</b>	<b>latitude</b> : latitude in decimal degrees (North is positive), float, default =0 <b>height</b> : height over sea level in m, float, default = 0
<b>returns</b>	<ul style="list-style-type: none"> <li>- .csv table with ghi, dni, dhi and other parameters on a hourly basis [0..8759]</li> <li>- .csv table with daily totals for ghi, dni, dhi [0..364]</li> <li>- plot of daily totals for ghi, dni, dhi over the year</li> </ul>

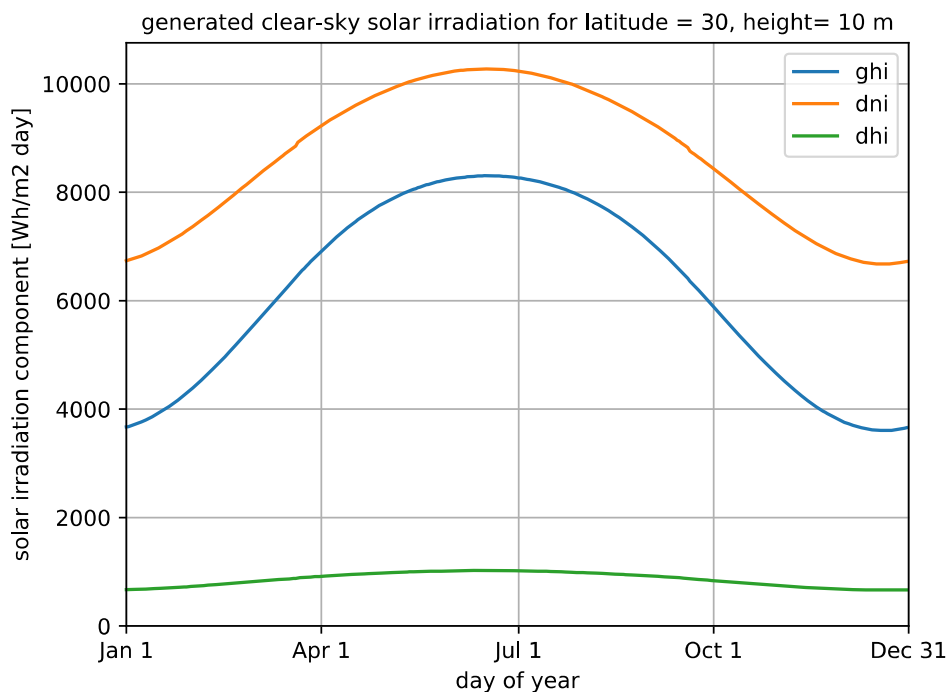


Figure 1-17: Generated clear-sky solar irradiation for Lat = 30°N. The DNI profile is not entirely smooth because of some approximation errors during the first and the last hour of some days.

## 1.5 Flat panels with arbitrary orientation

### 1.5.1 Panel orientation and angle of incidence

The most widely used systems for the capture of solar energy are flat panels – solar modules for electric power generation or thermal panels for heating a fluid, such as water. The useful power amount of a solar beam striking a plane surface is its normal component with respect to plane. That is, the power that can be transferred to the surface depends on the beam's **angle of incidence**  $\theta$  with the normal direction from the plane. The maximum power transfer is reached when the incident beam is perpendicular to the surface, that is, the angle between beam and normal direction is zero. For angles different from zero, the share of useful power collected on the plane is

$$h(\theta) = h_0 \cdot \cos\theta \quad (1-21)$$

with  $h_0$  the power of the beam (over a normal surface) and  $\theta$  the angle between the beam and the vector normal to the plane. In the case of the horizontal plane finding  $\theta$  is immediate: for a solar elevation angle  $\alpha$ , the angle of incidence  $\theta$  of the solar beam over the plane is given by

$$\theta = 90^\circ - \alpha$$

this is the same as the Zenith angle of Equation 1-10, p.43.

A generic surface for the collection of solar radiation has a **tilt**  $\beta$  over the ground ( $\beta=0^\circ$  is flat,  $\beta=90^\circ$  is vertical). The **orientation** of the surface is indicated by a **azimuth angle**  $Az$ . In a way similar as for the solar direction, the azimuth angle for South orientation can be indicated as either  $=0^\circ$  or  $=180^\circ$ . In this text, in analogy with geographical coordinates and the azimuth angle of the sun,  $Az=180^\circ$  means orientation due South.

The angle of incidence  $\theta$  of the solar beam on a flat surface is the angle between two vectors, the solar elevation pointing from the ground to the sun (vector  $s$ ), and the normal to the surface (vector  $n$ ). The angle of incidence is obtained from the scalar product of the two vectors:

$$\vec{s} \cdot \vec{n} = |s| \cdot |n| \cdot \cos\theta \quad (1-22)$$

This relation can be transformed in the following expression in terms of the orientation angles

$$\cos\theta = (\cos\alpha \cdot \sin\beta \cdot \cos(Az - A) + \sin\alpha \cdot \cos\beta) \quad (1-23)$$

where  $A$  and  $\alpha$  are the solar azimuth and elevation,  $A_z$  and  $\beta$  the flat surface orientation and tilt.  $\theta$  must be in the range  $[0^\circ..90^\circ]$ , i.e., it is defined only for positive values of the cosine function.

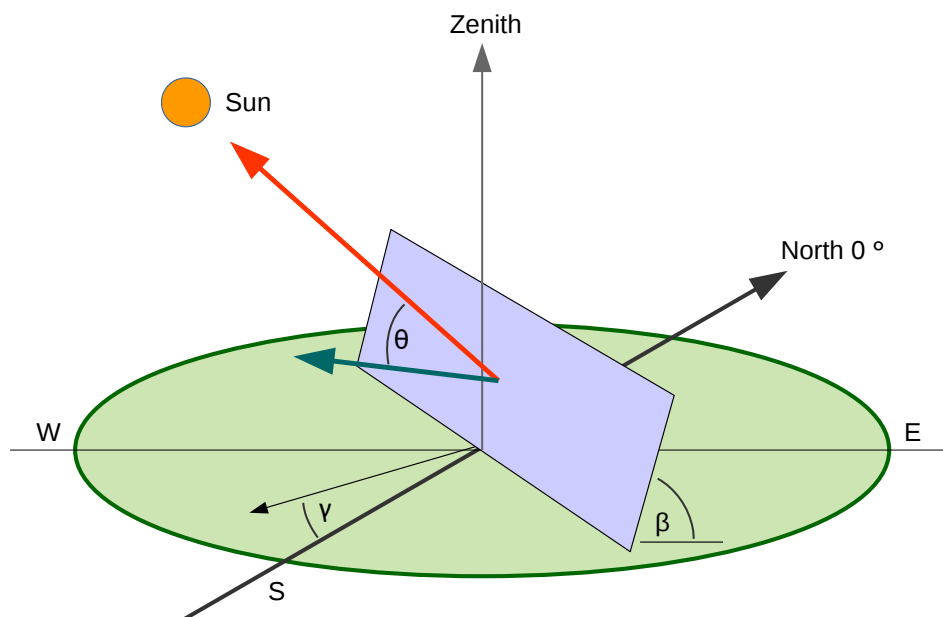
According to Equation (1-23) when the solar azimuth is the same as the panel orientation  $A_z=A$ , their difference is 0 and the cosine term becomes =1. The relation

$$\cos \alpha \cdot \sin \beta + \sin \alpha \cdot \cos \beta$$

is the same as

$$\sin(\alpha + \beta)$$

From this relation it follows that the angle of incidence  $\theta$  is equal  $90^\circ$  less the sum of the solar elevation  $\alpha$  and the flat surface tilt  $\beta$ .



*Figure 1-18: Coordinate system for an oriented flat surface:  $\beta$  is the tilt angle of the collecting surface from ground,  $\gamma$  is the azimuth angle of the normal to the surface with respect to due South,  $\theta$  is the solar beam angle over the surface.  $\cos(\theta)$  gives the share (0..1) of the solar beam radiation that is actually collected by the surface.*

### Program\_Code 1-27: function, angle of incidence between solar beam and an oriented flat surface

<b>package</b>	<b><code>solprim.solartimeposition</code></b>
<b>function</b>	<b><code>incidence_factor</code></b>
<b>description</b>	calculate the cosine of the angle of incidence $\theta$ between the solar beam direction and the normal to an oriented flat panel
<b>parameters</b>	<b><code>sol_elev_sin</code></b> : sin of solar elevation angle, float [0..1] <b><code>sol_elev_cos</code></b> : cos of solar elevation angle, float [0..1] <b><code>srf_tilt_sin</code></b> : sin of flat surface tilt angle, float [0..1] <b><code>srf_tilt_cos</code></b> : cos of flat surface tilt angle, float [0..1] <b><code>sol_azim_rad</code></b> : solar azimuth angle in rad, float $[-\pi..+\pi]$ <b><code>srf_azim_rad</code></b> : flat surface azimuth angle in rad, float $[-\pi..+\pi]$
<b>returns</b>	cosine of the angle of incidence $\theta$ , float [0..1]

The function `incidence_factor` for the calculation of the angle of incidence of a solar beam over a flat surface is accessed repeatedly to track the solar path over days, months, and years. In order to operate without unnecessary delays it accepts pre-calculated sine and cosine values and returns the cosine of the angle of incidence. The sine and cosine values for the solar elevation angles are returned by the function `solar_azim_elev` that uses them as intermediate calculation steps. The value returned by the function `incidence_factor`, a cosine, can be multiplied directly by the intensity of solar beam radiation to obtain the power collected on an oriented flat surface.

The practical application of the `incidence_factor` function is shown in Section 2.4, for the calculation of solar energy collected by flat panels over long periods of time.

### 1.5.2 Energy collected by a tilted surface

The energy collected on a flat panel with given orientation (azimuth) and tilt is known as **plane of array** (POA). The POA energy yield is the sum of the direct and diffuse component of the incoming solar radiation. Their contribution is calculated in different ways.

The energy obtained from the direct component of solar radiation is given by the product of the beam intensity with the cosine of the angle of incidence:

$$h(\theta) = h_0 \cdot \cos \theta \quad (1-21)$$

with  $h_0$  equal either the direct radiation  $B_n$  or DNI over a time interval.

The cosine of the incidence angle  $\theta$  is obtained from Equation (1-23). The relation holds only for  $\cos(\theta) \geq 0$ , that is, for incidence angles  $\geq 0^\circ$  and  $\leq 90^\circ$ .

The calculation of the diffuse radiation collected by a tilted surface follows a different approach, for which knowledge of the solar position is not necessary. It is usually assumed that the radiation intensity from each direction in the sky dome is the same (isotropic). A tilted surface “sees” only the section of sky in front of it. The share of the global diffuse radiation in an isotropic sky over a flat surface with tilt angle  $\beta$  is expressed by the relation:

$$D_\beta = D_h \cdot \frac{1}{2} \cdot (1 + \cos \beta) \quad (1-24)$$

some textbooks report a simplified form (the factor is 180 with the angle  $\beta$  expressed in degrees or  $\pi$  for radians):

$$D_\beta = D_h \cdot \left( \frac{180 - \beta}{180} \right) \quad (1-25)$$

Other models assume that the diffuse sky radiation is not the same from all directions (anisotropic), for example, that the intensity is higher from the equator direction or near the sun. These models are predictably more difficult to use, as they need to consider several parameters and conditions and don't guarantee that they can be applied in all situations. For simplified calculations, Equation (1-24) is adequate.<sup>17</sup>

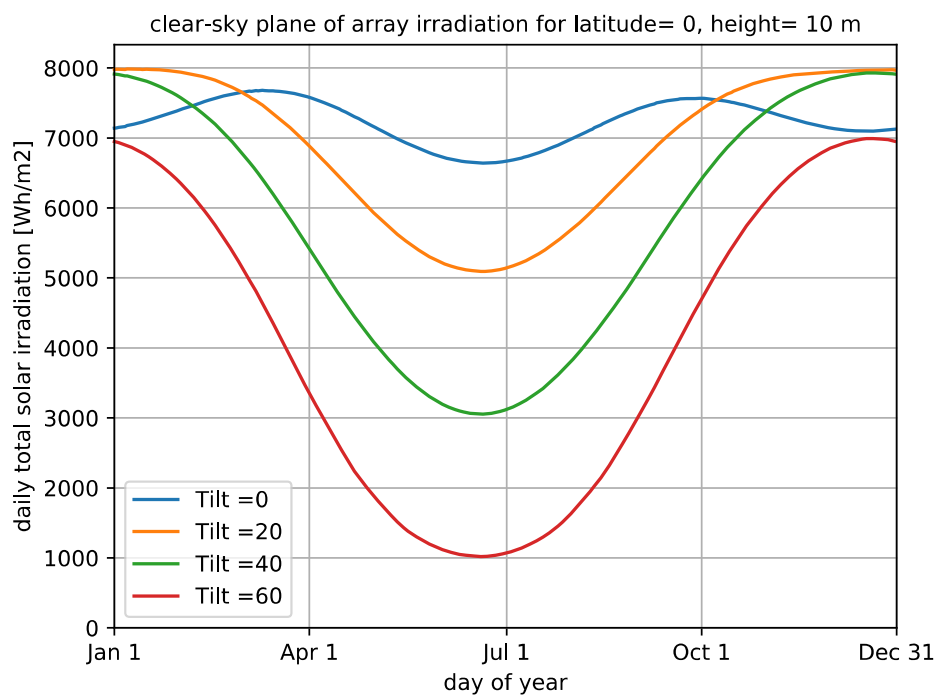
---

17 Different models are reported in Abbas Mousavi Maleki, Seyed, H. Hizam, and Chandima Gomes, “Estimation of Hourly, Daily and Monthly Global Solar Radiation on Inclined Surfaces: Models Re-Visited”, *Energies* 2017, 10, 134; doi:10.3390/en10010134

### Program\_Code 1-28: script, plot clear-sky radiation collected by a tilted surface

<b>script</b>	<b>clearsky_poa</b>
<b>description</b>	for a given latitude plot the clear-sky radiation collected by a surface with given orientation and tilt (plane-of-array) in the course of the year. The plot is produced for different tilt angles.
<b>parameters</b>	<b>latitude</b> : latitude in decimal degrees (North is positive), float <b>height</b> : height over sea level, meter, integer/float <b>plane_azim</b> : oriented surface azimuth angle, degrees [0°..360°], float <b>plane_tilt_list</b> : list of oriented surface tilt angles, deg [0°..90°], float
<b>returns</b>	plot of the plane of array irradiation from clear-sky radiation for different values of the tilt angle over the year

The script **clearsky\_poa** calculates hourly the energy yield on a tilted flat surface from the components of the clear-sky beam and diffuse radiation from the **clearsky\_radiation** function. For a given latitude the function shown the yield at different tilt angles (Figure 1-19, Figure 1-20).



*Figure 1-19: Clear-sky plane-of-array energy yield at the equator. The best result over the year is obtained with an horizontal plane.*

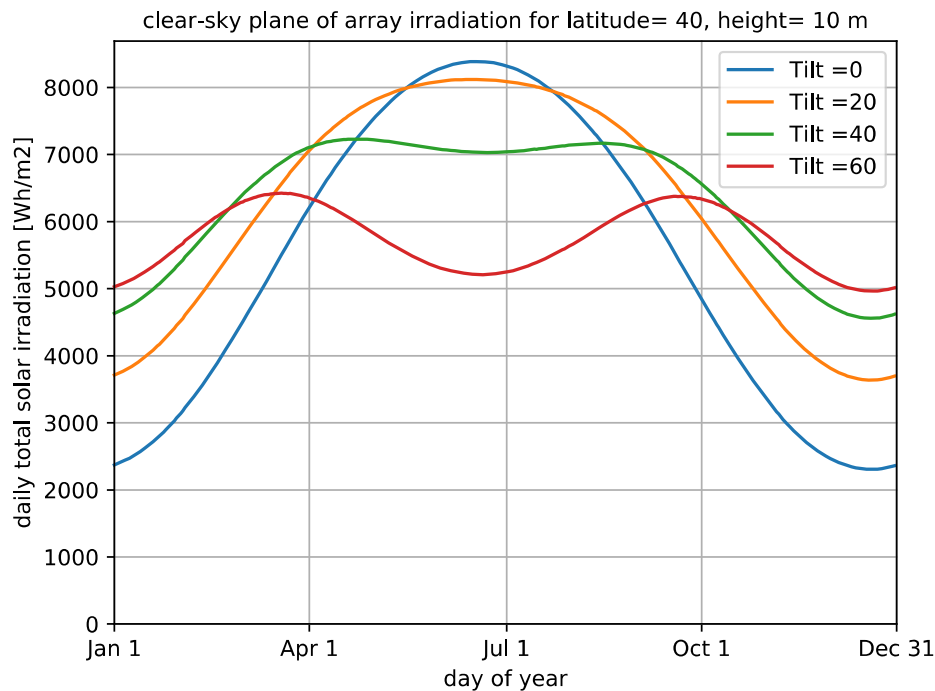


Figure 1-20: Clear-sky plane-of-array energy yield at latitude =40°N. Flat-surface tilt angles between 40° and 60° give the most regular result over the year.

### Online\_Resource 1-8: Solar power collected by tilted surfaces (pveducation)

The effect of the latitude and the tilt angle of a collecting surface are shown graphically and intuitively with an interactive tool at

<http://pveducation.org/pvcdrom/properties-sunlight/solar-radiation-tilted-surface>



## 2 Solar and Climate Data Representation

### 2.1 Solar resource maps

Weather plays the most important role on the yield of solar collectors. Its influence cannot be calculated analytically, it can only be estimated from observations made over several years. Essential weather-related information extracted from large datasets is presented in an easily understandable way, such as tables or maps.

Due to the increasing interest for renewable energy applications, in the last years a large number of maps has been made available for several regions and countries worldwide. The most common parameter reported on solar resource maps is the Global Horizontal Irradiation (GHI), but some maps report also the Direct Normal Irradiation / Beam Irradiation (DNI) and the Diffuse Horizontal Irradiation (DHI). The maps usually refer to the whole year and a widespread unit of measure for the cumulated irradiation values is  $[\text{kWh}\cdot\text{m}^{-2}]$ , either as day average or year total. Other units and time intervals are also found.

A list of map sources for solar data is reported in the Appendix. In particular, high-quality maps are provided by the World Bank ESMAP facility, by the US research institute NREL and by the European company Solargis.

### 2.2 The Typical Meteorological Year (TMY)

#### 2.2.1 General information about the TMY

Cumulative data about solar energy, such as that reported in tables or maps, is a simple and compact indicator about daily, monthly or yearly averages, but it does not reflect the natural changes in the availability of the natural resource. Parameters such as average GHI and DNI are sufficient for a rough dimensioning of a basic solar system and a first assessment of its economic feasibility, but they are of limited utility for the analysis of the system operation over time. On the other hand, understanding the system behavior in course of time is necessary for the correct dimensioning of storage facilities and the requirements for backup solutions.

The analysis of system time behavior becomes possible with the use of a dataset for the most important ambient variables (GHI, DNI, temperature, humidity) over the full year. This dataset is called the **Typical Meteorological Year (TMY)** and has the aspect of a spreadsheet table with the single values reported as columns on a hourly basis (shorter time definition, 30 or 15 minutes, are also used in some cases, though they are still uncommon). This brings to a total of 24 hours x 365 days = 8760 hours, leap years are not considered. A TMY has therefore 8760 data rows over a dozen or more columns. Some rows at the beginning of the file may contain location metadata, such as location name, country, geographical coordinates, height, etc.

Simulations based on the TMY are easily carried out with personal computers or laptops and are becoming increasingly popular because of all the additional information they provide compared to the simpler cumulative or average indicators. TMY files have a typical size of few hundreds kB up to approx. 1.8 MB. TMY data is available for a large number of locations worldwide. Some datasets are available on the Internet for free, others on a commercial basis.

A TMY dataset is produced for any location from meteorological recordings over ten to twenty years. From these are selected twelve single months, each best representing the same month over the full recorded data, in particular with regard to GHI, DNI, and the dry bulb temperature. The annual and monthly average values of these parameters are selected in such way to be consistent with the long-term averages for the chosen dataset. Data for months with untypical, extreme high or low values for some parameter is discarded. The TMY is then formed by linking the data for the selected months, smoothing it over some hours at the limit from one month to the next.

TMY data is used, among others, to assess the performance and economic viability for solar energy use, for both PV and thermal generation, as well as in building simulation for the analysis of heating and cooling under varying ambient conditions. Due to the fact that TMY datasets represent long-term representative data rather than extreme situations, taken alone they are not suited for designing systems to meet worst-case conditions.

## 2.2.2 TMY data format types

### TMY2

TMY2 is the collection of data from 229 locations (all in the USA) between 1961 and 1990. Its format was defined at a time when computer memory space was a major issue, so that its contents are coded in order keep the file size small. TMY2 files in the original coding can be read with a text editor, though the data is not understandable. Today TMY2 datasets are of limited importance and have only legacy value. However, TMY2 has established a methodology that, after several enhancements and improvements, is still in use.

### TMY3

The Typical Meteorological Year No.3 (TMY3) has been developed for 1020 locations in the USA. The dataset was derived from the 1961-1990 and 1991-2005 National Solar Radiation Data Base (NSRDB) archives. TMY3 data makes use of the comma separated value (CSV) format, which is simple to read and edit, for example via spreadsheet applications.

TMY3 refers actually to two different sets of data. In the earlier format single files were approx. 450 kB, in the updated format (also referred to as "TYA") about 1.7–1.8 MB.

The field order in the older format TMY3 is:

Row#1, header for the location metadata:

**Source, Location ID, City, State, Country, Latitude, Longitude, Time Zone, Elevation**

Row#2, location metadata

Row#3, header for the TMY data

**Year, Month, Day, Hour, GHI, DNI, DHI, Tdry, Tdew, RH, Pres, Wspd, Wdir, Albedo**

Rows#4..8763 contain the bulk of data, on hourly basis for the whole year (a 365-day year contains 8760 hours).

In a text editor the CSV file appears as

```
Source,Location ID,City,State,Country,Latitude,Longitude,TimeZone,Elevation
TMY3,725030,New York Laguardia Arpt,NY,USA,40.783000,-73.883000,-5,3
Year,Month,Day,Hour,GHI,DNI,DHI,Tdry,Tdew,RH,Pres,Wspd,Wdir,Albedo
1991,1,1,0,0,0,0,-2.2,-12.8,40,1035,4.6,330,0.16
1991,1,1,1,0,0,0,-2.2,-12.8,40,1035,3.1,340,0.16
1991,1,1,2,0,0,0,-2.2,-12.8,40,1036,3.6,330,0.16
1991,1,1,3,0,0,0,-2.8,-12.8,42,1036,2.6,10,0.16
1991,1,1,4,0,0,0,-2.8,-12.2,44,1036,2.1,50,0.16
[...]
2003,12,31,22,0,0,0,6.1,-3.9,47,1018,7.2,240,0.17
2003,12,31,23,0,0,0,6.1,-3.9,47,1018,6.7,250,0.17
```

The year of the first column is the one from which the month has been selected, in most cases it is not relevant for computations.

The tables in the updated TMY3 format are about 60 columns wide and contain a large number of weather data, including information on the operation and the recording quality of the data sensors. In fact, most data has been overtaken from the earlier files. Missing data is reported as such, while some other parameters have been simulated on the basis of wide-scale inputs, for example, satellite observations.

Due to the limited size of the files, its simplicity in practical use and the possibility to access and edit data via Python functions, the pandas package as well as spreadsheet programs, the functions and scripts developed for this primer make use of the “previous” TMY3 format. Data conversion scripts are presented to read data from new TMY formats and store it in as following the “previous” TMY3 format.

### 2.2.3 Time handling in TMY datasets

Data in the TMY datasets is reported on an hourly basis. For data that can be integrated over time, such as, for example, the solar irradiation values, is recorded the total amount during the 60-minute period ending at the timestamp. For variables that only have instant values, such as temperatures or the wind speed, is recorded the actual value at the instant of the timestamp.

In the TMY2 and the “previous” TMY3 format year, month, day, and hour are reported each in its own column. The hour indicated as “0” represents the time between 24:00:00 and 0:59:59, hour “1” is the time between 01:00:00 and 01:59:59 etc.

In the updated TMY format the date is contained in one single field with MM/DD/YYYY representation. The first hour is indicated as 01:00, the second 02:00 etc. with the following aspect

```
01/01/2002,01:00  
01/01/2002,02:00  
01/01/2002,03:00  
01/01/2002,04:00  
01/01/2002,05:00
```

Hour “0” in the older TMY3 format is therefore the same as “01:00” in the new TMY.

### Online\_Resource 2-1: TMY2 and TMY3 datasets

The TMY2 dataset was the first collection of TMY resource data for 239 stations, all in the USA. Due to the memory limitations of the time, the files are saved in a compact form which needs to be decoded. The original full dataset is still available on the Internet, though today it only has legacy value:

[http://rredc.nrel.gov/solar/old\\_data/nsrdb/1961-1990/tmy2/](http://rredc.nrel.gov/solar/old_data/nsrdb/1961-1990/tmy2/)

National Solar Radiation Data Base User's Manual (1961-1990):

<http://rredc.nrel.gov/solar/pubs/NSRDB/>

TMY2 User's manual (pdf):

<http://rredc.nrel.gov/solar/pubs/tmy2/>

The National Solar Radiation Data Base TMY3 dataset contains TMY3 files both in the updated format (each file is ~ 1.8 MB) and the “previous” format (each file ~ 450kB). It is located at

[http://rredc.nrel.gov/solar/old\\_data/nsrdb/1991-2005/tmy3/](http://rredc.nrel.gov/solar/old_data/nsrdb/1991-2005/tmy3/)

The TMY3 Users Manual is posted under

<http://www.nrel.gov/docs/fy08osti/43156.pdf>

The TMY3 files are in `.csv` format and can be easily opened with a spreadsheet program such as LibreOffice Calc. In the interface window shall be selected Data Format “UTF-8”. The choice of “UTF-16” may lead to conversion problems, with the content ending up looking like Chinese characters.

In other spreadsheet programs it may be necessary to turn off all automatic and default data interpretation, lest the table content is converted to something quite different from TMY data. Without control of the conversion process a temperature of 1.2 degrees, for example, might be understood as January, 2<sup>nd</sup>. LibreOffice Calc gives the user full control by default and is less likely to present this type of problems.

### SAM Formats TMY2, TMY3, INTL

The comprehensive renewable energy simulation package System Advisor Model (SAM) by NREL includes a wide selection of TMY data – TMY2, TMY3, and INTL (these are shown as part of the name). In addition, SAM can read files in the new TMY and EPW formats. The formats TMY2 and TMY3 are for USA locations, INTL is used for all other (international) locations. In the SAM version of 2017-01-17 are included files for 1259 USA locations, 55 Canadian, 305 from other countries. All files are stored as `.csv` and can be opened and edited with spreadsheet programs.

The basic content is the same for all SAM data files, TMY2, TMY3 or INTL, but some of their columns are different. For example, the SAM TMY2 files include the following fields

**Year, Month, Day, Hour, GHI, DNI, DHI, Tdry, Tdew, RH, Pres, Wspd, Wdir, Snow Depth**

The SAM TMY3 files contain the following datafields (an example of the content was shown in → Section 2.2.2).

**Year, Month, Day, Hour, GHI, DNI, DHI, Tdry, Twet, RH, Pres, Wspd, Wdir, Alb**

In the “previous” TMY3 format the variable *Twet* (wet bulb temperature) is shown instead of *Tdew* (dew point), though the two temperatures are related and any of them can be calculated from the other and the dry temperature. *Albedo* is reported instead of *Snow Depth*, though real

measurements of both snow depth and albedo are seldom. These fields are either marked as not valid, given default values, or filled with 0s.

The files in the SAM solar resource database marked as “INTL” also have a somewhat different data structure, with the fields

**Year, Month, Day, Hour, Beam, Diffuse, Tdry, Tdew, Pres, Wdir, Wspd, Aod, Pwp, Alb**

The INTL TMY format of the SAM package does not report values for the global horizontal irradiation *GHI* neither the relative humidity *RH*, though the latter is in relation to *Tdry* and *Twet*. In the SAM INTL format are included the two parameters *Aod* (aerosol optical depth [0..1]) and *Pwp* (precipitable water). However, the same as for albedo and snow depth, the fields are there but the related data is not. The files of INTL type also differ in the header of Row#0 and Row#1, with the “source” information placed at the end of the record instead of the beginning.

The function `readtmy_csv` (Program\_Code 2-1) reads .csv files in the SAM TMY3 “previous” format. It would be quite easy to adapt this function in order to read also INTL TMY files. However, this is hardly worth the effort because the data for the same locations is available as EPW files (→ Section 2.2.4), which are easier to access, already contain the GHI information, and most of all are available for a far larger number of locations. Conversion functions are provided to extract and save the data of interest from the updated TMY3 files (Program\_Code 2-2), from EPW files (Program\_Code 2-3), and from the updated PVGIS TMY profiles (Program\_Code 2-6).

### Program\_Code 2-1: function, read CSV file, return TMY data as pandas dataframe

<b>package</b>	<a href="#">solprim.tmyutility</a>
<b>function</b>	<a href="#">tmy_readcsv</a>
<b>description</b>	read a TMY2 or TMY3 dataset in <code>.csv</code> format, return data as pandas dataframe
<b>parameters</b>	<b>filename</b> : TMY ( <code>.csv</code> ) dataset file name with directory path, string
<b>returns</b>	tuple with 1. location metadata as Python dictionary structure 2. pandas dataframe indexed for the hour of the year [0..8759] and the parameters in columns. The name of the columns are read from the original TMY file.

The utility package [solprim.tmyutility](#) contains several functions for reading and processing files with TMY hourly data as well as for their representation in graphical format.

The function [tmy\\_readcsv](#) reads a TMY dataset in `.csv` format and returns a pandas dataframe indexed for the hour of the year [0..8759] and the parameters in columns. The use of pandas data structures makes further processing by other functions much easier.

Additional information on the operation of the [tmy\\_readcsv](#) function is found in the comments to the code. pandas functionality is briefly dealt with in → Section 3.2.1.



### Program\_Code 2-2: function, convert TYA file to CSV TMY3 format

<b>package</b>	<b>solprim.tmyconvert</b>
<b>function</b>	<b>convert_tya_to_csv</b>
<b>description</b>	read a TMY dataset in <code>.csv</code> file with updated TMY3 format (TYA), save relevant data in a <code>.csv</code> file in “previous” TMY3 format
<b>parameters</b>	<b>filename</b> : TYA ( <code>.csv</code> ) dataset file name with directory path, string
<b>returns</b>	<code>.csv</code> file with 8760 hourly values for the columns in the original dataset including GHI, DNI, DHI, Tdry, RH. Most of the original TYA columns are discarded.

The function `convert_tya_to_csv` reads a `.csv` file in updated TMY3 format (TYA) and prepares a file in “previous” TMY3 format with relevant content for solar energy calculations.

TYA files are stored as `.csv` and can be opened with a text editor or a spreadsheet. TYA datafiles contain location metadata only in row#1 and without a separate description header. The format is, for example:

```
725847,"SOUTH LAKE TAHOE",CA,-8.0,38.900,-120.000,1909
```

In order, these fields are

```
station ID, location name, state, time zone, latitude,
longitude, elevation over sea level (m)
```

Row#2 contains the header for the main data

```
Date (MM/DD/YYYY),Time (HH:MM),ETR (W/m^2),ETRN (W/m^2),GHI (W/m^2),GHI
source,GHI uncert (%),DNI (W/m^2),DNI source,DNI uncert (%),DHI (W/m^2),DHI
source,DHI uncert (%),GH illum (lx),GH illum source,Global illum uncert
(%),DN illum (lx),DN illum source,DN illum uncert (%),DH illum (lx),DH
illum source,DH illum uncert (%),Zenith lum (cd/m^2),Zenith lum
source,Zenith lum uncert (%),TotCld (tenths),TotCld source,TotCld uncert
(code),OpqCld (tenths),OpqCld source,OpqCld uncert (code),Dry-bulb (C),Dry-
bulb source,Dry-bulb uncert (code),Dew-point (C),Dew-point source,Dew-point
uncert (code),RHum (%),RHum source,RHum uncert (code),Pressure
(mbar),Pressure source,Pressure uncert (code),Wdir (degrees),Wdir
source,Wdir uncert (code),Wspd (m/s),Wspd source,Wspd uncert (code),Hvis
(m),Hvis source,Hvis uncert (code),CeilHgt (m),CeilHgt source,CeilHgt
uncert (code),Pwat (cm),Pwat source,Pwat uncert (code),AOD (unitless),AOD
source,AOD uncert (code),Alb (unitless),Alb source,Alb uncert
(code),Lprecip depth (mm),Lprecip quantity (hr),Lprecip source,Lprecip
uncert (code),PresWth (METAR code),PresWth source,PresWth uncert (code)
```

The bulk of data is stored in row# [3..8762]. Only few of the parameters in TYA files are relevant for solar energy calculations.

The operation of the `convert_tya_to_csv` function is straightforward. At the beginning the location metadata is read from row#1. With the location metadata is built a Python dictionary, the country code is filled with 'USA' (updated TMY3 files are available only for the United States) and the data source with the field 'TMY3\_new(TYA)'. This metadata is then stored in the header of the target file as comma-separated values with the TMY3 data structure. The location metadata is also returned to the calling function at the end of the function.

The bulk of the data is read in a pandas dataframe and the columns of interest are selected. The data column names are the same as for the “previous” TMY3 files. With the new columns is built a new dataframe and saved as .csv together with the location metadata. The generated file receives the same name as the input TYA file with the extension '\_CONVERT' and datatype .csv.

Compared to older TMY3 files, the TYA files contain information for ETR (Extraterrestrial radiation) and ETRN (Extraterrestrial radiation normal), both in  $[W \cdot m^{-2}]$ . These are the calculated values of the solar radiation at the outer layer of the atmosphere, at the angle of incidence determined by latitude, day of year, and time of day, and normal (this value does not change during the day). These parameters may be useful for comparison with the solar radiation measured the earth surface and are therefore included in the generated TMY file.

### Program\_Code 2-3: script, convert TYA file to CSV TMY3 format

<b>script</b>	<b>convert_tya_csv</b>
<b>description</b>	read a TMY updated dataset (TYA) in <code>.csv</code> format, save relevant data in a <code>.csv</code> file in “previous” TMY3 format
<b>parameters</b>	<b>filename</b> : TYA ( <code>.csv</code> ) dataset file name with directory path, string
<b>returns</b>	<code>.csv</code> file in older TMY3 format with same data as the original TYA file for the main columns, in addition extraterrestrial radiation ETR, ETRN

The script `convert_tya_csv` calls the function `convert_tya_to_csv` to read a file in TYA format and prepare a TMY3 file with relevant data.

The output file is stored in the same directory where the input file is located.

#### 2.2.4 EPW format and conversion to TMY

For international users outside the USA, the TMY2 and TMY3 datasets are of limited value. Fortunately, there is another major repository, also managed by an US government institution, with a wide selection of international TMY data. The **EnergyPlus** building performance simulation software developed by the USA Department of Energy offers access to an open database with datasets for approx. 2100 locations: 1042 locations in the USA, 71 locations in Canada, and more than 1000 locations in hundred other countries worldwide.

EnergyPlus uses the proprietary data format `.epw` (from “Energy Plus Weather”). The data storage format is `.csv`, so that `.epw` files can be opened and read with text editors and spreadsheet programs. The content is distributed over some dozens columns with data for the 8760 hours of a 365-day year, including the main parameters for solar radiation, temperature, and relative humidity. Part of the data is coded and not immediately recognizable, while the main data of importance for solar energy use is in clear and can be easily processed. `.epw` datafiles have an 8-rows header with different location and aggregated data followed by 8760 rows [8..8767] with TMY data content. The time count

is the same as in the TMY2 and older TMY3 formats, with single columns for year, month, day, and hour. The hour count is 1-24, whereby each hour contains accumulated values for the following 60 minutes. Data for hour=0 in the older TMY3 format is thus the same as for hour=1 in EPW.

#### Program\_Code 2-4: function, convert EPW file to CSV TMY3 format

<b>package</b>	<b><code>solprim.tmyconvert</code></b>
<b>function</b>	<b><code>convert_epw_to_csv</code></b>
<b>description</b>	read a TMY dataset in EPW (.csv) format, save relevant data in a .csv file in TMY3 format
<b>parameters</b>	<b>filename</b> : EPW (.csv) dataset file name with directory path, string
<b>returns</b>	.csv file with 8760 hourly values for the older format columns including GHI, DNI, DHI, Tdry, RH. Most of the original .epw columns are discarded.

The function `convert_epw_to_csv` reads a file in .epw format and builds a file in the older TMY3 format with the most relevant data for solar energy calculations. The generated file has the same name as the input EPW file with the extension '\_CONVERT' and datatype .csv. At the beginning is read the location metadata from row#0, which has the following form:

```
LOCATION,Beijing,Beijing,CHN,CSWD,545110,39.80,116.47,8.0,31.3
```

In order, these fields are

```
'LOCATION', location name, state/region, country code, data  
source, station ID, latitude, longitude, time zone, elevation  
over sea level (m)
```

With the location metadata is built a Python dictionary in which the index strings are put in relation to their values. This metadata is then stored in the header of the output file as comma-separated values with the TMY3 data structure. At the end the location metadata is returned to the calling function.

Rows #1..7 of the input file contain parameters related to building design that are not relevant for solar energy applications.<sup>18</sup> These rows are skipped over.

Rows #8..8767 contain bulk data in several dozens columns. Each row is read sequentially, the data is split into single parameter values and the relevant ones are appended to the respective lists. The hour indication is decreased by 1 to match the [0..23] convention of TMY files. The rest of the data is the same in the EPW and the TMY files, so that any data for hour 'h' in EPW the same as for hour 'h-1' in TMY.

The ready columns are built into a pandas `dataframe` and its content is appended with the pandas '`to_csv`' method to the `.csv` output file.

#### Program\_Code 2-5: script, convert EPW file to CSV TMY3 format

<b>script</b>	<code>convert_epw_csv</code>
<b>description</b>	read a TMY file in EPW format, prepare a <code>.csv</code> file in TMY3 "previous" format, save relevant data
<b>parameters</b>	<b>filename</b> : EPW ( <code>.csv</code> ) dataset file name with directory path, string
<b>returns</b>	TMY3 file in <code>.csv</code> format with same data as the original <code>.epw</code> file for the columns related to solar energy applications (most <code>.epw</code> data is discarded as non-relevant)

The script `convert_epw_csv` calls the function `convert_epw_to_csv` to read an `.epw` file and store relevant TMY data in a `.csv` file with TMY3 "previous" data format. The output file is stored in the same directory of the input file.

<sup>18</sup> For more detailed information on the EPW data format refer to the documentation [https://energyplus.net/sites/default/files/pdfs\\_v8.3.0/AuxiliaryPrograms.pdf](https://energyplus.net/sites/default/files/pdfs_v8.3.0/AuxiliaryPrograms.pdf)

### Online\_Resource 2-2: EPW EnergyPlus TMY dataset

TMY data in .epw (EnergyPlus) format for more than two thousands locations worldwide, half in the USA, half international, is accessible on the EnergyPlus website <https://energyplus.net/weather>

The data sources are national and international meteorological organizations, listed under <https://energyplus.net/weather/sources>

A detailed description of the EPW data format is given in

[https://energyplus.net/sites/default/files/pdfs\\_v8.3.0/AuxiliaryPrograms.pdf](https://energyplus.net/sites/default/files/pdfs_v8.3.0/AuxiliaryPrograms.pdf)

This information, however, is very detailed and far more than what is strictly required to understand the applications and exercises described in this primer.

### Online\_Resource 2-3: TMY format overview

An overview of different TMY formats is contained in

[https://www.nrel.gov/analysis/sam/help/html-php/index.html?weather\\_format.htm](https://www.nrel.gov/analysis/sam/help/html-php/index.html?weather_format.htm)

### 2.2.5 TMY series generation via the EU PVGIS Website

EU PVGIS is a European Union tool in many respects similar to PVWatts by NREL. Its website provides information about solar radiation and photovoltaic system performance. PVGIS can be used to calculate the energy that can be obtained from different types of PV systems at most locations in the world (a performance comparison of PVWatts and PVGIS is presented in → Section 4.1).

PVGIS includes a tool to generate TMY from data series at selected locations, with the input obtained from satellite data and weather recordings at meteorological stations on the ground. Locations are selected in PVGIS by their geographical coordinates and not by the indication of a geographical name (Online\_Resource 2-4).

A file generated and downloaded by PVGIS has the name, for example

**tmy\_era\_44.53\_11.3\_2005\_2014.csv**

The numerical indications are the coordinates of the location, while 2005 and 2014 refer to the underlying dataset years, from 2005 to 2014. A location in the Southern hemisphere has, appropriately, minus signs in the numerical coordinates, as in the case of Cape Town, South Africa:

**tmy\_era\_-33.969\_18.597\_2005\_2014.csv**

A TMY file generated by PVGIS contains metadata only for latitude, longitude and height over sea level. It lists separately the reference years from which the single months have been selected. Date and time are given as compact parameter in a single field, while the data content is more or less the same as for the TMY3 files in the “previous” format.

Further, a PVGIS TMY file contains the date and time reference only for UTC and independently on the actual, selected geographical location. This may lead to inconsistencies with reference to the solar position, so that in some cases the time reference needs to be corrected.

More information about the PVGIS format is given in the conversion function described in Program\_Code 2-6.

### Online\_Resource 2-4: EU PVGIS tool for TMY series generation

EU PVGIS (Photovoltaic Geographical Information System) is a map-based data repository for solar radiation and a tool to simulate the performance of PV solar generation systems. The beta version is online at

<http://re.jrc.ec.europa.eu/PVGIS5-beta.html>

EU PVGIS includes a tool to generate a TMY from data series at locations identified by their geographical coordinates

[http://re.jrc.ec.europa.eu/pvg\\_tools/en/tools.html#TMY](http://re.jrc.ec.europa.eu/pvg_tools/en/tools.html#TMY)

The old version of EU PVGIS, with a much more limited functionality (and a not particularly friendly user interface) is accessible at

<http://re.jrc.ec.europa.eu/pvgis/apps4/pvest.php>

### Program\_Code 2-6: function, convert PVGIS file to CSV TMY3 format

<b>package</b>	<b><code>solprim.tmyconvert</code></b>
<b>function</b>	<b><code>convert_pvgis_to_csv</code></b>
<b>description</b>	read a TMY dataset in PVGIS <code>.csv</code> format, save relevant data in a <code>.csv</code> file in TMY3 format
<b>parameters</b>	<b>filename</b> : PVGIS ( <code>.csv</code> ) dataset file name with directory path, string
<b>returns</b>	<code>.csv</code> file with 8760 hourly values in the “previous” format with columns for GHI, DNI, DHI, Tdry, RH. Some of the original PVGIS columns are discarded.

The conversion steps from a TMY datafile generated by PVGIS to a TMY3 file are described at detail in the function `convert_pvgis_to_csv`.

TMY files generated by PVGIS do not contain any information about the location name, only geographical coordinates. The original PVGIS files should be renamed before their conversion because the function takes the file name as location name.

The data strings from the original input file may contain the character sequence '+AC0', it is unclear why. The function deletes that pattern when it is found in the input data.



### Program\_Code 2-7: script, convert PVGIS file to CSV TMY3 format

<b>script</b>	<b>convert_pvgis_csv</b>
<b>description</b>	read a TMY file in PVGIS .csv format, prepare a .csv file in TMY3 format, save relevant data
<b>parameters</b>	<b>filename</b> : PVGIS (.csv) dataset file name with directory path, string
<b>returns</b>	TMY3 dataset ("previous" format) as .csv file with the same data as the original PVGIS file

The script `convert_pvgis_csv` calls the function `convert_pvgis_to_csv` to read a PVGIS file and store TMY data in a .csv file with TMY3 data format. The output file is stored in the same directory of the input file.

### Online\_Resource 2-5: Conversion of TMY2, TMY3, INTL Format to .csv via PVWatts

The online program PVWatts for the simple simulation of PV generation systems can also be used to prepare TMY files. PVWatts has access to a wide TMY reference database for worldwide locations. In order to produce a TMY dataset it is necessary to define a PV conversion system, for this the default values are sufficient, and select the result download in "hourly" format. This is a table with 8760 values for beam irradiance [ $\text{W}\cdot\text{m}^{-2}$ ], diffuse irradiance [ $\text{W}\cdot\text{m}^{-2}$ ], ambient temperature ( $^{\circ}\text{C}$ ), and wind speed [ $\text{m}\cdot\text{s}^{-1}$ ] for the selected location. In addition, the generated file contains values for the Plane-of-Array irradiance [ $\text{W}\cdot\text{m}^{-2}$ ], that is, the energy collected on a flat plane with given azimuth and tilt, the cell temperature ( $^{\circ}\text{C}$ ), the dc array output [W], and the ac system output [W].

The generated table is easily opened in a spreadsheet for visualization and further processing. By inserting location metadata in the TMY3 format described in → Section 2.2.2 it is possible to produce a TMY file that can be read with the function of Program\_Code 2-1.

PVWatts is found at <http://pvwatts.nrel.gov/pvwatts.php>

### 2.2.6 Aggregated and statistical data from TMY datasets

With the TMY datasets stored in a standard and clearly defined format, in this case the “previous” TMY3 format, it is possible to run several simple statistical and analysis functions. These are described in the following, they provide some understanding of the location and the local availability of solar energy. The script of Program\_Code 2-14 makes use of all these functions to present in a compact form the potential for the use of solar energy at the selected location.

Program\_Code 2-8: function, daily average of TMY data

<b>package</b>	<b>solprim.tmyutility</b>
<b>function</b>	<b>tmy_daily_average</b>
<b>description</b>	from a list with 8760 hourly values calculate the daily averages, return a list with 365 values
<b>parameters</b>	<b>datalist</b> : list or dataframe column with TMY hourly values [0..8759], integer or float
<b>returns</b>	list object with 365 daily items, daily average values for <b>datalist</b> content, integer or float

From a TMY list with 8760 hourly values (one pandas column) the function returns a list with 365 values, each the daily average of the selected parameter. This function is useful as a filter, e.g., to show the yearly profile of data such as GHI or temperature while canceling out short-term variations.

The function **tmy\_daily\_average** is used, for example, in the preparation of graphs to cancel out short-time variations and, in the case of solar irradiation the values for the night hours are canceled away, thus de-cluttering the graphical output.

**Program\_Code 2-9: function, daily total of TMY data**

<b>package</b>	<b><code>solprim.tmyutility</code></b>
<b>function</b>	<b><code>tmy_daily_total</code></b>
<b>description</b>	from a list with 8760 hourly values for cumulative parameters calculate the daily totals, return a list with 365 values
<b>parameters</b>	<b><code>datalist</code></b> : list or <code>dataframe</code> column with TMY hourly values [0..8759], integer or float
<b>returns</b>	list object with 365 items, daily total values for <b><code>datalist</code></b> content, integer or float

From a TMY list with 8760 hourly values (one pandas column) the function `tmy_daily_total` returns a list with 365 values, each the daily total of the selected parameter.

This function very similar to `tmy_daily_average` but it is treated separately because averages can be computed on any type of numeric values, while totals make sense only for cumulative values such as, e.g., energy, solar irradiation, or rainfall, but not for, e.g., pressure or temperature.

### Program\_Code 2-10: function, monthly total of TMY data

<b>package</b>	<b>solprim.tmyutility</b>
<b>function</b>	<b>tmy_monthly_total</b>
<b>description</b>	from a TMY dataset of cumulative values (ghi, dni, dhi, but also ,e.g., rainfall) calculate the monthly and the year totals and the monthly and yearly averages
<b>parameters</b>	<b>datalist</b> : list or dataframe column with TMY hourly values [0..8759], integer or float
<b>returns</b>	tuple with <ul style="list-style-type: none"> <li>- list of total values per calendar month [0..11], float</li> <li>- list of average values per calendar month [0..11], float</li> <li>- total for the year, float</li> <li>- daily average for the year, float</li> </ul>

The function **tmy\_monthly\_total** operates on any cumulative data in TMY format, such as solar irradiation or rainfall. The function returns a tuple with further lists for totals and averages on a monthly basis, the total of the input value and its daily average for the year.

In the function are considered the different lengths of the months during the year, so that, for example, total and average values for January are computed on the basis of 31 days.

An application example of **tmy\_monthly\_total** is shown in Program\_Code 2-14.

### Program\_Code 2-11: function, heating and cooling degree days from TMY

<b>package</b>	<b>solprim.tmyutility</b>
<b>function</b>	<b>tmy_hdd_cdd</b>
<b>description</b>	from a Tdry column from a pandas TMY dataframe and reference temperatures (ambient threshold and indoor reference temperature) calculate the heating and the cooling degree days
<b>parameters</b>	<b>t_dry</b> : 'Tdry' list or dataframe column with hourly values [0..8759], integer or float <b>hdd_threshold</b> : threshold temperature for the heating degree days, float, default = 13 <b>hdd_ref</b> : reference indoor temperature for heating, float, default = 18 <b>cdd_threshold</b> : threshold temperature for the cooling degree days, float, default = 26 <b>cdd_ref</b> : reference indoor temperature for cooling, float, default = 24
<b>returns</b>	tuple with - heating degree-days, integer - cooling degree-days, integer

The heating degree days parameter (HDD) is a compact measure of both the intensity of the heating demand, which depends on the outdoor temperature over the year, as well as the duration of the heating season.

HDD is calculated from the daily average temperatures over the year and two reference temperatures, a threshold temperature  $T_{threshold}$ , usually taken as 13° Celsius, and an indoor temperature  $T_{ref}$ , with typical value 18° Celsius. Only those days with an average temperature  $T_{avg}$  less or equal  $T_{threshold}$  are considered in the HDD calculation:

$$T_{avg} \leq T_{threshold, heat} \quad (2-1)$$

The underlying assumption is that in cold days energy must be provided to buildings to raise their indoor temperature at least to the  $T_{ref}$  value. In days with a low  $T_{avg}$  the difference between  $T_{avg}$  and the indoor reference temperature  $T_{ref}$  is added up to the year HDD total:

$$HDD = \sum_{day=1}^{365} |T_{indoor ref, heat} - T_{avg}(day)| \text{ for } T_{avg}(day) \leq T_{threshold, heat} \quad (2-2)$$

The HDD calculated in this way provides a good indication of the energy demand for heating over the whole year. HDD values calculated in degrees Celsius for cities in the temperate zone, such as Southern France, Spain, or Italy, lie around 2000 HDD, in the much colder Stockholm at about 4000 HDD.

The cooling degree days indicator (CDD) is obtained in a similar way, by considering only those days with an average temperature above a certain threshold and an indoor reference temperature for cooling. The equation for CDD is then

$$CDD = \sum_{day=1}^{365} |T_{avg}(day) - T_{indoor\ ref,\ cool}| \text{ for } T_{avg}(day) \geq T_{threshold,\ cool} \quad (2-3)$$

A typical threshold temperature for inclusion in the CDD sum is 26°, an indoor cooling temperature 24°. These values may be defined otherwise depending on the relative humidity, which requires additional cooling to provide the necessary comfort.

The function `tmy_hdd_cdd` returns a tuple with the two values for HDD and CDD for the input `t_dry` dataset and the threshold and reference values. An example of its use is shown in Program\_Code 2-14.

### Program\_Code 2-12: function, statistical values for TMY irradiation data

<b>package</b>	<b><code>solprim.tmyutility</code></b>
<b>function</b>	<b><code>tmy_irradiation_stats</code></b>
<b>description</b>	from a TMY list dataset of solar irradiation values (ghi/ dni/ dhi) return key aggregated parameters
<b>parameters</b>	<b><code>sol_irr</code></b> : list or dataframe column with GHI/ DNI/ DHI hourly values [0..8759], integer or float
<b>returns</b>	tuple with <ul style="list-style-type: none"> <li>- irrads max on hourly basis (Wh), integer</li> <li>- irrads max for the day (Wh), integer</li> <li>- day number for irrads max (day 1 is January, 1), integer</li> <li>- irrads year sum (kWh), integer</li> <li>- irrads day average (kWh), float</li> </ul>

The function `tmy_irradiation_stats` is used as facility for quick calculations over solar irradiation values. Application examples are shown in Program\_Code 2-14.

### Program\_Code 2-13: function, number of days with total GHI below a preset threshold

<b>package</b>	<a href="#">solprim.tmyutility</a>
<b>function</b>	<a href="#">tmy_low_ghi_days</a>
<b>description</b>	from a TMY dataset find the total number and the maximum number of consecutive days with a total ghi insolation value below a given threshold
<b>parameters</b>	<b>ghi_list</b> : list or dataframe column with GHI hourly values [0..8759], integer or float <b>ghi_threshold</b> : ghi threshold value, integer
<b>returns</b>	tuple with - number of consecutive days with ghi total for the day < threshold, integer - total number of days with ghi total for the day < threshold, integer

At most locations in the world and for several days (or weeks) during the year the solar radiation level is not sufficient to cover loads and to charge batteries. The function [tmy\\_low\\_ghi\\_days](#) returns the total number of days in the year with total ghi below a given ghi threshold as well as the number of consecutive days with a total ghi insolation value below the threshold. These parameters provide an immediate indication over the feasibility of full or partial solar supply and give a first indication about the required capacity for power storage (→ Section 4.3).

An application example of the [tmy\\_low\\_ghi\\_days](#) function is shown in the script of Program\_Code 2-14.



### Program\_Code 2-14: script, read CSV TMY file, show aggregated parameters

<b>script</b>	<b>tmy_location_data</b>
<b>description</b>	read a .csv TMY dataset in older TMY3 format, display location metadata and relevant aggregated parameters
<b>parameters</b>	<b>filename</b> : TMY (.csv) dataset file name with directory path, string
<b>returns</b>	location metadata and relevant aggregated parameters on output terminal (Python shell)

The script `tmy_location_data` reads a file in TMY format and reports the location metadata together with some aggregated values from the functions `tmy_hdd_cdd`, `tmy_monthly_total`, `tmy_irradiation_stats`, and `tmy_low_ghi_days`.

The Berlin EPW file is available from the EnergyPlus repository (→ Online\_Resource 2-2) It has been converted to TMY with the script `convert_epw_to_csv`. An example of output for the Berlin data with the script `tmy_location_data` is the following

```
{'Longitude': 13.4, 'Location ID': '103840', 'City': 'BERLIN', 'Country': 'DEU', 'Latitude': 52.47, 'Elevation': 49, 'Time Zone': 1.0, 'Source': 'IWEK Data', 'State': '-'}
```

```
Tdry min =-9.1 max =32.8 avg =9.8 all degC
```

```
heating degree days = 2941
cooling degree days = 8
```

```
number of consecutive days with ghi < 2000 Wh = 45
number of total days with ghi < 2000 Wh = 166
```

```
specific irradiation values per m2
key indicators about GHI
    maximum hourly irrads (Wh) =870
    maximum daily irrads (Wh) =7764 on day =186
    year total insolation (kWh) =985
    day avg irradiation (Wh) =2.7
```

```
key indicators about DNI
    maximum hourly irrads (Wh) =866
    maximum daily irrads (Wh) =8764 on day =186
    year total insolation (kWh) =703
    day avg irradiation (Wh) =1.93
```

```
key indicators about DHI
    maximum hourly irrads (Wh) =459
    maximum daily irrads (Wh) =3762 on day =165
    year total insolation (kWh) =585
    day avg irradiation (Wh) =1.6
```

```
>>>
```

```
>>>
```

Resolute, Canada, is the northernmost location with TMY data available as EPW on the EnergyPlus website. After conversion to TMY and a call to the script `tmy_location_data` the output is

```
{'Source': 'WYEC2-B-17901', 'City': 'Resolute', 'Time Zone': -6.0,
'Elevation': 67, 'State': 'NU', 'Country': 'CAN', 'Longitude': -94.98,
'Location ID': '719240', 'Latitude': 74.72}
```

```
Tdry min =-43.9 max =12.8 avg =-16.4  all degC
```

```
heating degree days = 12571
```

```
cooling degree days = 0
```

```
number of consecutive days with ghi < 2000 Wh = 82
```

```
number of total days with ghi < 2000 Wh = 212
```

```
specific irradiation values per m2
```

```
key indicators about GHI
```

```
    maximum hourly irrads (Wh) =694
```

```
    maximum daily irrads (Wh) =8989 on day =173
```

```
    year total insolation (kWh) =875
```

```
    day avg irradiation (Wh) =2.4
```

```
key indicators about DNI
```

```
    maximum hourly irrads (Wh) =944
```

```
    maximum daily irrads (Wh) =18191 on day =173
```

```
    year total insolation (kWh) =968
```

```
    day avg irradiation (Wh) =2.65
```

```
key indicators about DHI
```

```
    maximum hourly irrads (Wh) =634
```

```
    maximum daily irrads (Wh) =7455 on day =156
```

```
    year total insolation (kWh) =547
```

```
    day avg irradiation (Wh) =1.5
```

```
>>>
```

```
>>>
```

The number of heating degree days is extremely high (no surprise, with an average yearly temperature of  $-16.4^{\circ}\text{C}$ ). The total number of days with low irradiation values is 212, of which 82 are consecutive days. Solar energy is definitely not an option for such location.

A place where solar energy may play a very important role is Honolulu. The dataset is TYA from the TMY3 website, converted to the “previous” TMY3 format with the `convert_tya_to_csv` script. The key parameters are

```
{'Longitude': -157.93, 'Elevation': 2, 'Country': 'USA', 'State': 'HI',  
'Latitude': 21.32, 'Location ID': '911820', 'Source': 'TMY3_new(TYA)',  
'Time Zone': -10.0, 'City': '"HONOLULU INTL ARPT"'}
```

```
Tdry min =13.3 max =33.3 avg =24.9 all degC
```

```
heating degree days = 0  
cooling degree days = 379
```

```
number of consecutive days with ghi < 2000 Wh = 1  
number of total days with ghi < 2000 Wh = 1
```

```
specific irradiation values per m2  
key indicators about GHI  
    maximum hourly irrads (Wh) =1101  
    maximum daily irrads (Wh) =7698 on day =140  
    year total insolation (kWh) =1953  
    day avg irradiation (Wh) =5.35
```

```
key indicators about DNI  
    maximum hourly irrads (Wh) =986  
    maximum daily irrads (Wh) =9457 on day =149  
    year total insolation (kWh) =1854  
    day avg irradiation (Wh) =5.08
```

```
key indicators about DHI  
    maximum hourly irrads (Wh) =609  
    maximum daily irrads (Wh) =3787 on day =129  
    year total insolation (kWh) =730  
    day avg irradiation (Wh) =2.0
```

```
>>>
```

Honolulu has a pleasant average temperature of  $24.9^{\circ}\text{C}$  with a yearly maximum of  $33.3^{\circ}\text{C}$ . As expected, no heating is necessary, while the number of cooling degree days is not particularly high. Only one day per year has a low total insolation value. The parameters indicate that the location is by all means suitable for the use of solar energy.

### Program\_Code 2-15: function, estimated noon time from GHI

<b>package</b>	<b>solprim.tmyutility</b>
<b>function</b>	<b>tmy_noontime_from_ghi</b>
<b>description</b>	from a TMY GHI dataset calculate the average time for the maximum GHI irradiation (noontime)
<b>parameters</b>	<b>ghi_list</b> : list or dataframe column with GHI hourly values [0..8759], integer or float
<b>returns</b>	list [0..364] with the noon time calculated for each day of the year, average of the noon time over the year

The function **tmy\_noontime\_from\_ghi** calculates for each day the weighed average for noontime from a TMY list with 8760 GHI values and returns it in a list [0..364]. The average over the year is also calculated and returned. Only those days for which the sum of GHI over each half day is at least 30% of the total are accounted for, it is assumed that otherwise an unbalanced GHI value between the first and second part of the day would excessively distort the result.

For each day noontime is calculated as weighed average with the following relation

$$T_{noon} = \frac{\sum_{hour=0}^{23} GHI(h) \cdot (h+0.5)}{\sum_{hour=0}^{23} GHI(h)} \quad (2-4)$$

The fraction denominator is the GHI total for each day and the numerator is the sum of the hourly GHI values for each day with the hour as weighing factor. The GHI values in the TMY file are stored for each hour beginning with  $h$  and can be considered as the average values for the midpoint of each hour. For example, the GHI reported value for  $h=11$  is in fact the average value between  $h=11$  and  $h=12$ , i.e., with the midpoint at  $h=11.5$ . For this reason the hour index is increased by 0.5.

An application of this function with graphical output is shown in → Program\_Code 2-21.

## 2.3 TMY graphical data representation

### 2.3.1 Parameter plot

Program\_Code 2-16: function, plot hourly and daily data

<b>package</b>	<a href="#">solprim.tmyutility</a>
<b>function</b>	<a href="#">tmy_yearplot</a>
<b>description</b>	plot the parameter passed as list with hourly [0..8759] or daily values [0..364] over one year
<b>parameters</b>	<b>tmy_list</b> : list or dataframe column with hourly [0..8759] or daily [0..364] values for one TMY parameter, integer or float <b>header_text</b> : output plot header text, string <b>param_text</b> : lead text for the vertical axis/ parameter name, string
<b>returns</b>	graphic plot of the selected TMY parameter over the year

The function [tmy\\_yearplot](#) produces a plot of the input variable over the year. The axes are adjusted to the input list length (8760 vs 365 items).

Program\_Code 2-17: script, plot selected variable from TMY file

<b>script</b>	<a href="#">plot_tmy_data</a>
<b>description</b>	plot one parameter from a TMY file over the year
<b>parameters</b>	<b>filename</b> : TMY (.csv) dataset file name with directory path, string <b>param</b> : parameter to plot on graph / TMY column name, string
<b>returns</b>	graphic plot of the selected TMY parameter over the year

The script [plot\\_tmy\\_data](#) reads a file in TMY format and extracts data for the column indicated by the variable **param**. The data is plotted first with 8760 hourly values, then the daily average with 365 daily values calculated with [tmy\\_daily\\_average](#). Examples for GHI and Tdry data for New York City are shown in Figure 2-1 through Figure 2-4.

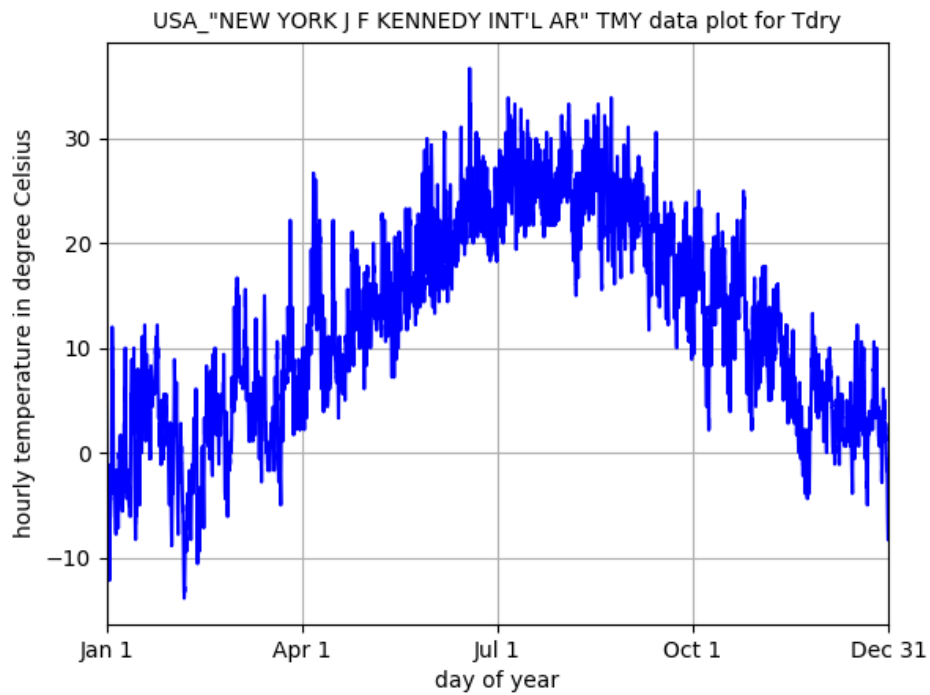


Figure 2-1: Hourly values of  $T_{dry}$  from the New York JFK Airport TMY

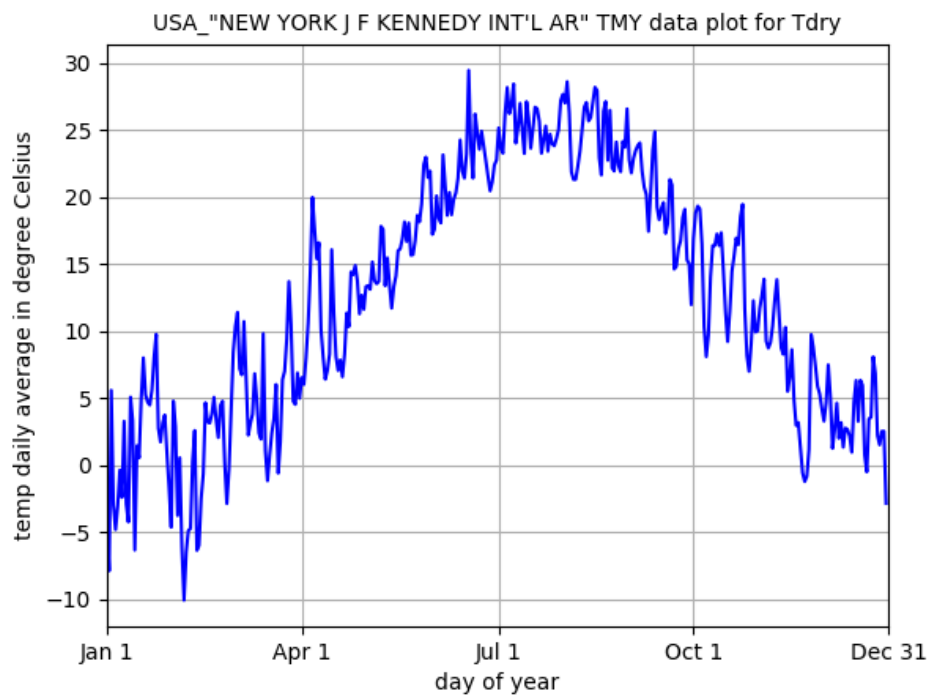


Figure 2-2: The same data as in Figure 2-1, with average daily values

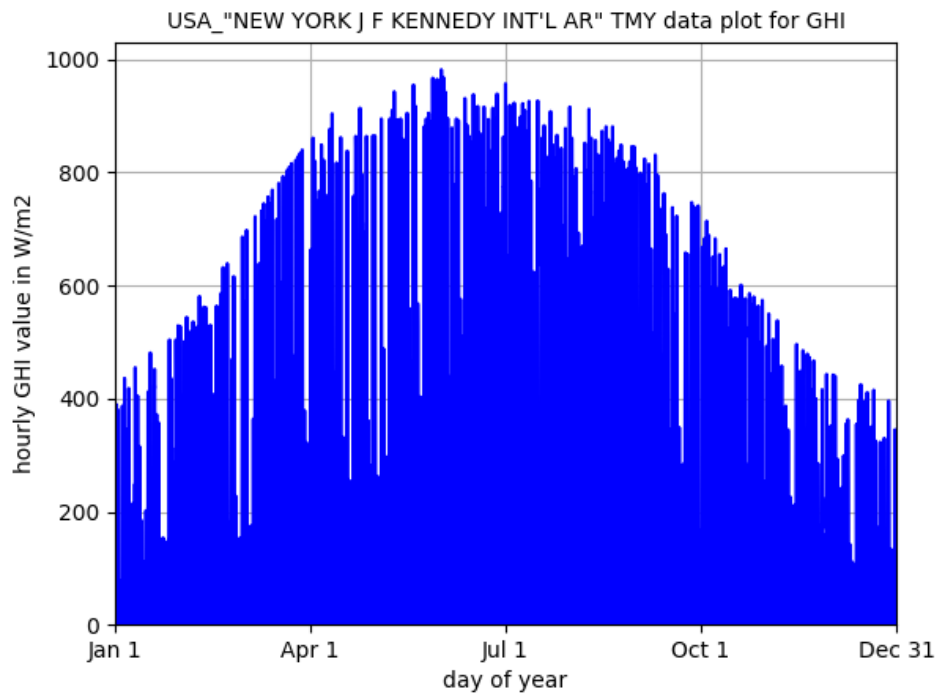


Figure 2-3: Hourly GHI values from the New York JFK Airport TMY

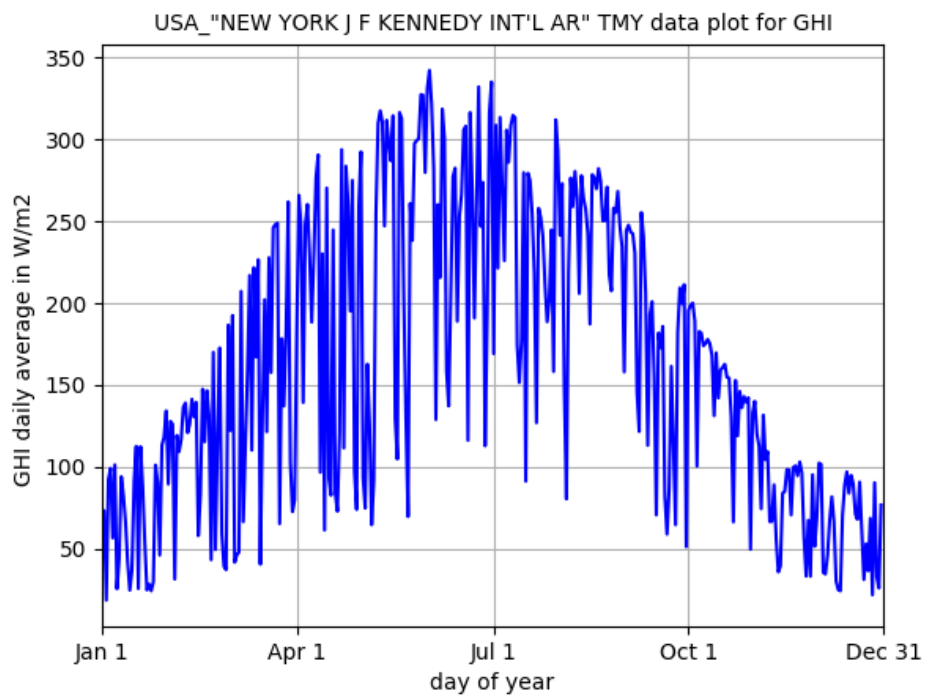


Figure 2-4: The same data as from Figure 2-3, daily averages

### Program\_Code 2-18: function, plot hourly and daily data, multiple

<b>package</b>	<b>solprim.tmyutility</b>
<b>function</b>	<b>tmy_yearplot_mult</b>
<b>description</b>	plot the parameters (two or more) passed as list tuple with hourly values [0..8759] or daily values [0..364] over one year
<b>parameters</b>	<b>tmy_tuple</b> : tuple of lists or dataframe columns with hourly [0..8759] or daily [0..364] values for one TMY parameter, integer or float <b>label_tuple</b> : tuple with the labels for the lists in <b>tmy_tuple</b> , string <b>header_text</b> : output plot header text, string <b>param_text</b> : lead text for the vertical axis/ parameter name, string <b>ypos_ref</b> : starting value for the 'y' axis, integer/float, default=None
<b>returns</b>	graphic plot of the selected TMY parameters over the year

The function **tmy\_yearplot\_mult** produces a plot of two or more input variables over the year. The function checks the length of the first input list (8760 vs 365 items) and scales the plot accordingly.

### Program\_Code 2-19: script, plot multiple variables from TMY file

<b>script</b>	<b>plot_tmy_mult</b>
<b>description</b>	from a TMY dataset plot two or more TMY parameters over the year
<b>parameters</b>	<b>filename</b> : TMY (.csv) dataset file name with directory path, string <b>param1</b> : parameter #1 to plot on graph / TMY column name, string <b>param2</b> : parameter #2 to plot on graph / TMY column name, string
<b>returns</b>	graphic plot of the selected TMY parameters over the year

The script **plot\_tmy\_mult** calls the function **tmy\_yearplot\_mult** to read a file in TMY format, extract data for one column with 8760 hourly values and plot it over the year. For data with ample variations between day and night the daily averages can be calculated before plotting. Output examples are shown in Figure 2-5 and Figure 2-6.



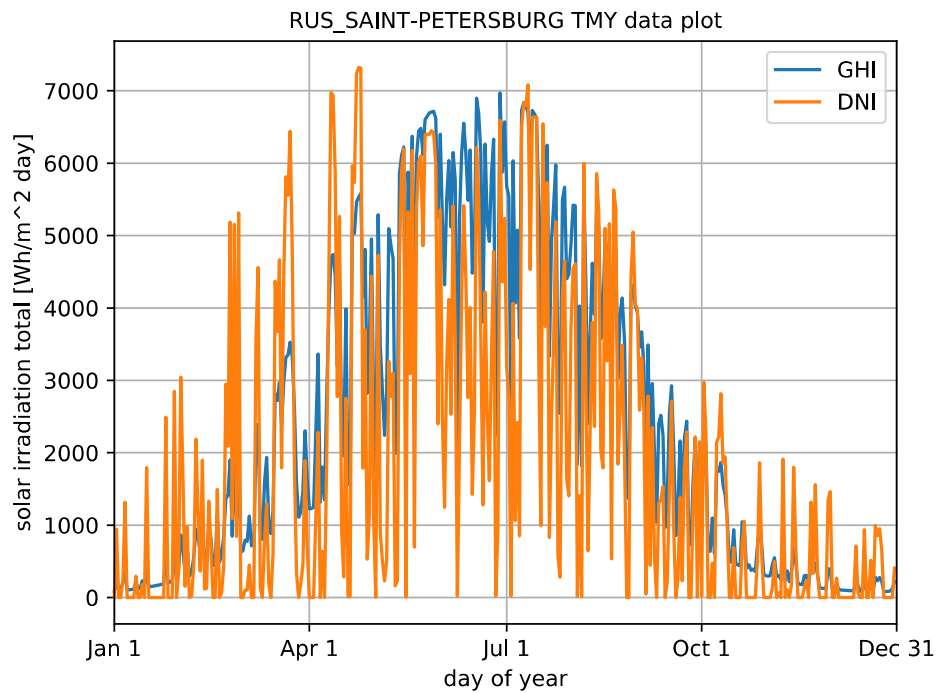


Figure 2-5: St.Petersburg, Russia, GHI and DNI daily averages over the TMY year

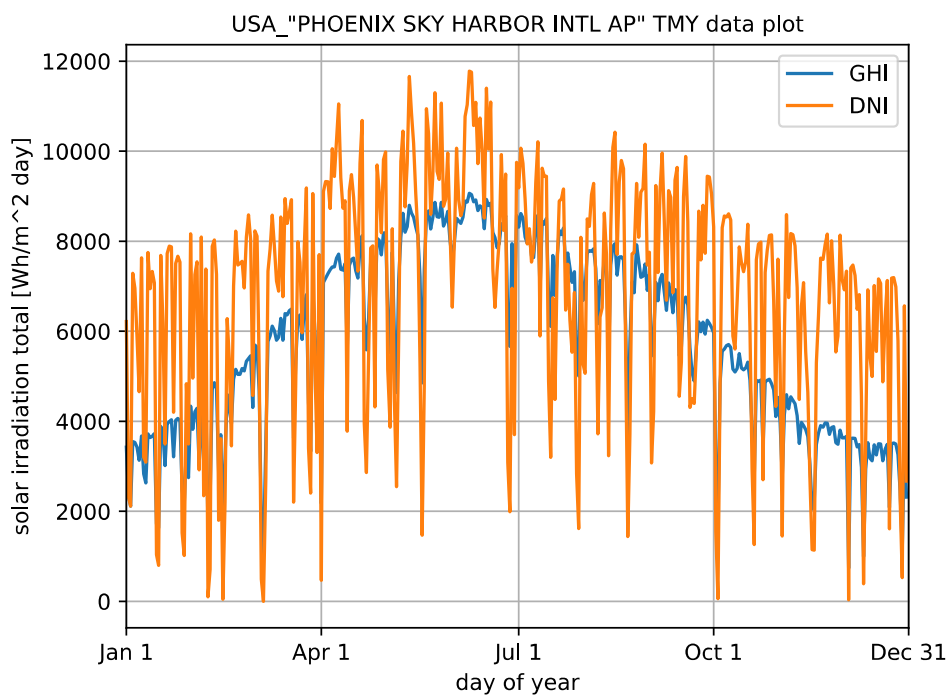


Figure 2-6: Phoenix, Arizona (USA), GHI and DNI daily averages over the TMY year

### Program\_Code 2-20: script, plot tmy data vs clear-sky radiation

<b>script</b>	<b>plot_tmy_vs_clearsky</b>
<b>description</b>	compare the GHI data from a TMY dataset with the generated clear-sky TMY
<b>parameters</b>	<b>filename</b> : TMY (.csv) dataset file name with directory path, string <b>param</b> : parameter to plot on graph / TMY column name, string
<b>returns</b>	plot of the ghi/ dni/ dhi profile for the selected location and the generated data for clear-sky conditions

The script **plot\_tmy\_vs\_clearsky** calls the function **tmy\_yearplot\_mult** to plot the GHI daily averaged profile from TMY data together with the generated GHI daily value for clear-sky conditions. The generated curve profile matches the TMY data well (Figure 2-7 to Figure 2-9). However, in some cases the curve is higher and in others lower than the TMY profile maximum. It would be interesting to determine whether this depends on the TMY input data, or the quality of the generated clear-sky irradiation profile.

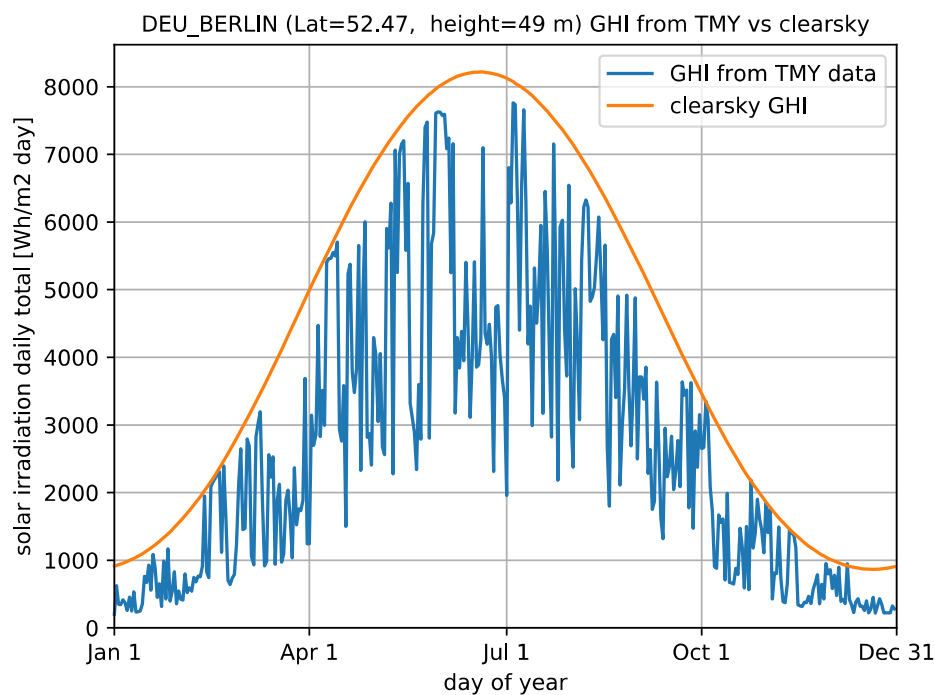


Figure 2-7: GHI for Berlin compared with the generated clear-sky GHI profile for the same latitude, height

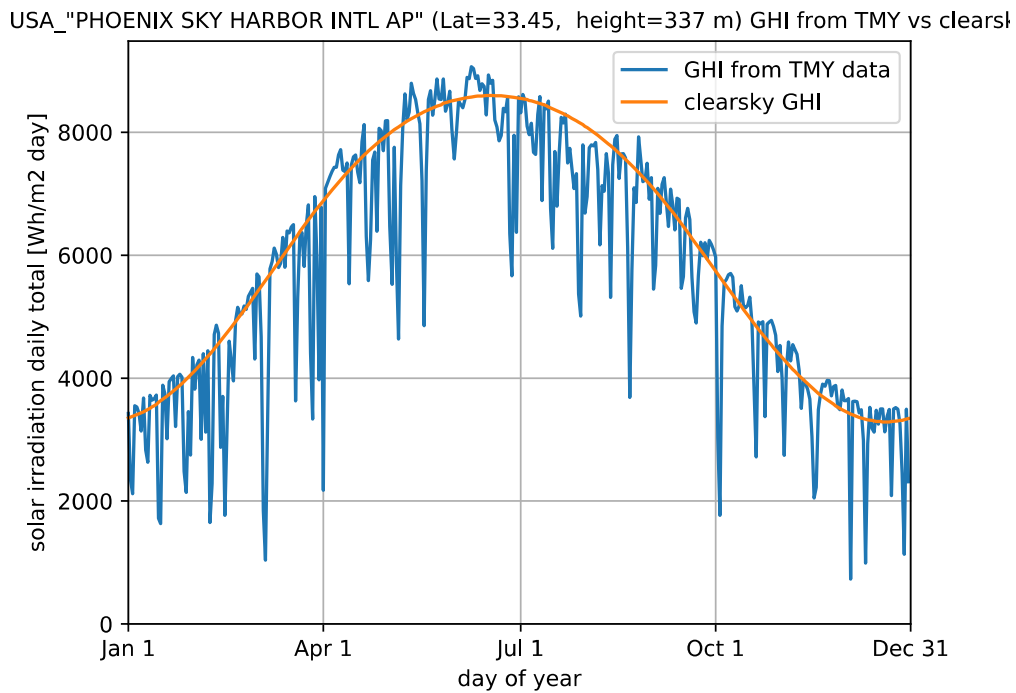


Figure 2-8: GHI for Phoenix, Arizona (USA) compared with the generated clear-sky GHI profile for the same latitude, height

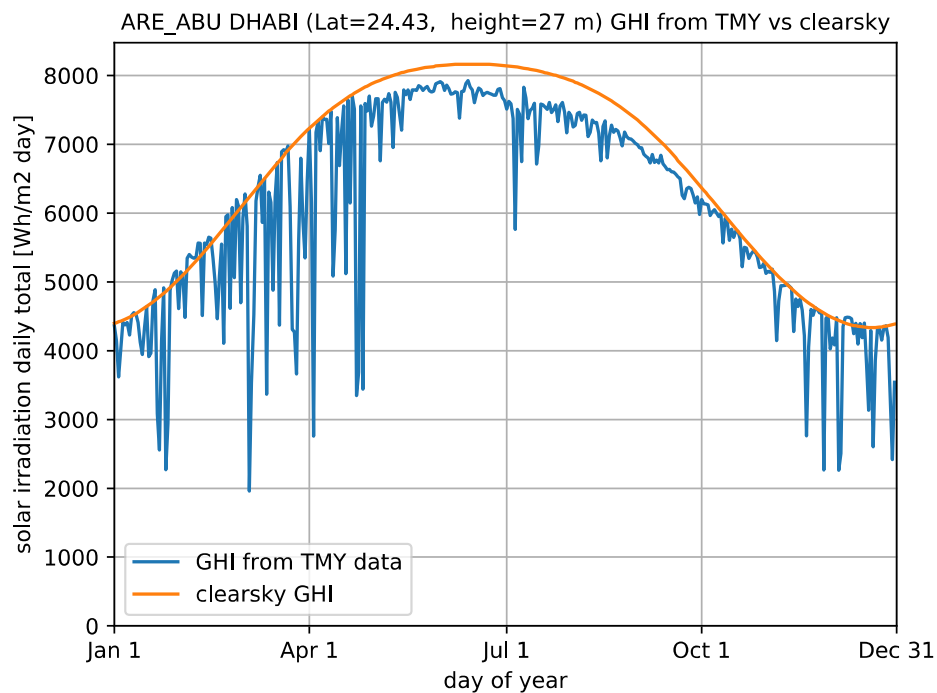


Figure 2-9: GHI for Abu Dhabi compared with the generated clear-sky GHI profile for the same latitude, height

### Program\_Code 2-21: script, plot noontime from GHI (equation of time)

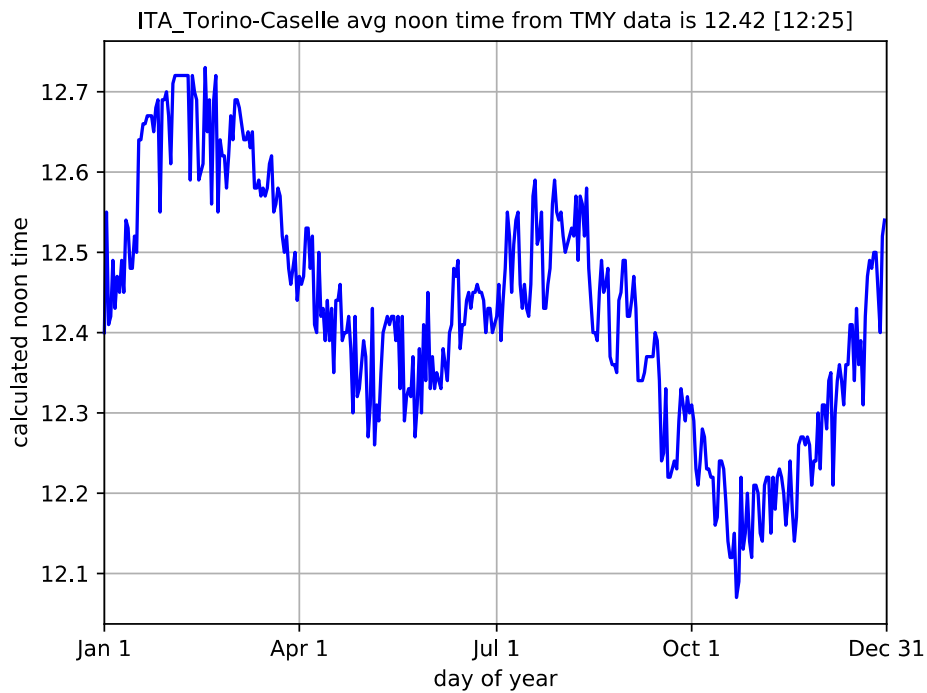
<b>script</b>	<b>plot_tmy_noontime</b>
<b>description</b>	from the GHI data in a TMY dataset calculate the average noontime in local time for each day, plot the result
<b>parameters</b>	<b>filename</b> : TMY (.csv) dataset file name with directory path, string
<b>returns</b>	plot of the calculated noontime from GHI data in the source TMY file

With the function **plot\_tmy\_noontime** is calculated the approximate noontime from the GHI column in a TMY dataset as the weighed time average of the hourly irradiation values for each day of the year (→ Section 2.2.6, Program\_Code 2-15) and the result is plotted. The effects of the equation of time and of the time shift due to the longitude appear clearly from several plots.

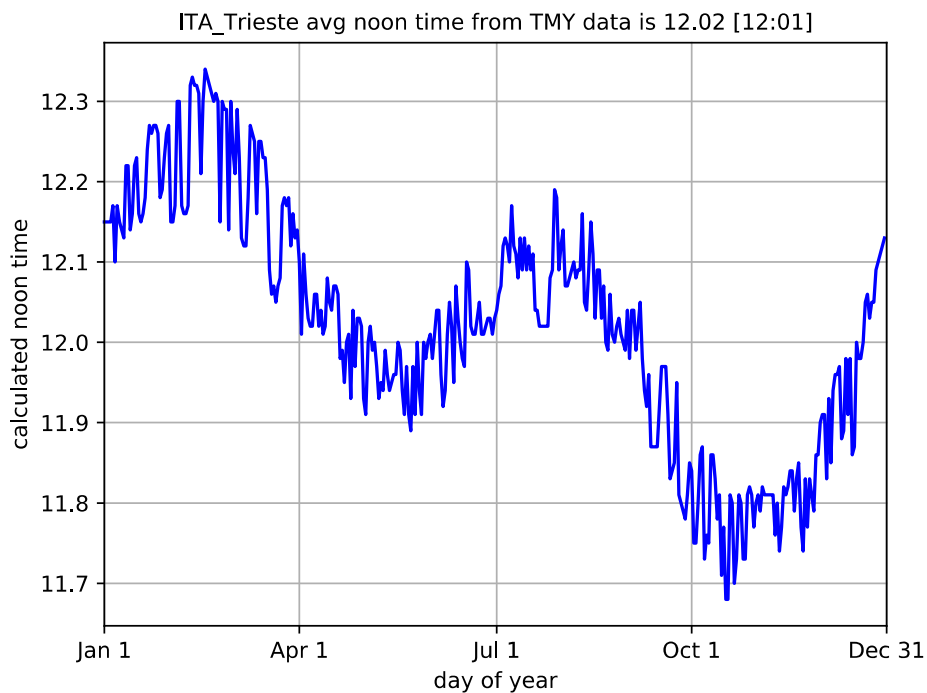
The effect of the Equation of Time (→ Section 1.2.4) and of the longitude shift (→ Section 1.2.5) are clearly visible in Figure 2-10 and Figure 2-11. Torino is at longitude 7.65 and in Time Zone UTC+1, apparent noon is therefore  $(15-65)*4$  minutes = ca. 25 minutes after local noon, which appears well from the graph. Trieste is also in Time Zone UTC+1, at lon=13.75. In this case the delay of apparent vs local noon is  $(15-13.75)*4$  minutes, that is., ca. 5 minutes after local noon. The calculated time from GHI data is 12:01, with an error of 4 minutes. In both graphs the difference between the earliest and the latest noontime is half hour, which also matches the range for the Equation of Time correction.

The data for New Orleans (Figure 2-12) is much more dispersed. However, also in this case the effect of the Equation of Time can be recognized. The average apparent solar noon is at 11:59, which agrees well with the longitude value =  $-90.25$ , i.e., exactly one minute difference to the local noon at the  $-90^\circ$  meridian.

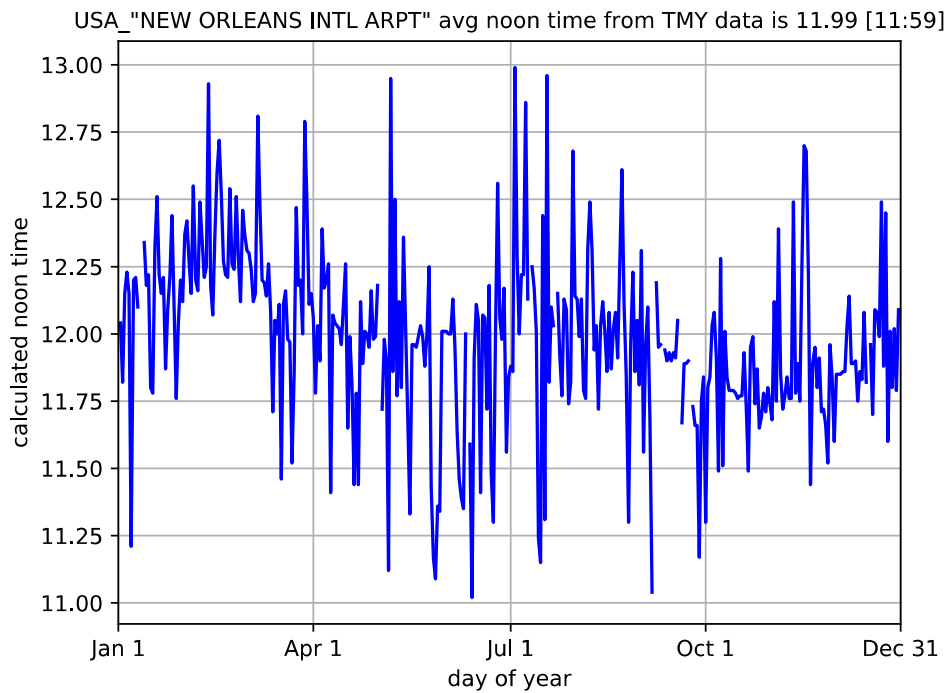
In some locations the Equation of Time and the time shift due to the longitude might not be visible, either because of too large variations in the calculated noon times or because the time in the TMY datasets was corrected with the apparent solar time taken as reference instead of the local time.



*Figure 2-10: Average solar noon time for Torino, Italy, calculated from the TMY GHI values. The deviations match the longitude time shift and the EoT correction.*



*Figure 2-11: The solar noon time calculated from the TMY GHI for Trieste, Italy, also gives a good representation of the longitude time shift and the EoT correction*



*Figure 2-12: The calculated solar noon time for New Orleans shows large variations due to variable weather. The effect of the equation of time correction is still visible.*

### 2.3.2 Barchart representation

Program\_Code 2-22: function, barchart for monthly data

<b>package</b>	<a href="#">solprim.tmyutility</a>
<b>function</b>	<a href="#">month_barchart</a>
<b>description</b>	produce a barchart for the parameter passed as list with monthly values
<b>parameters</b>	<b>month_list</b> : list with 12 monthly data values [0..11], integer or float <b>header_text</b> : output plot header text, string <b>param_text</b> : lead text for the vertical axis/ parameter name, string <b>bar_color</b> : color for the barchart in matplotlib format, string <b>box_text</b> : text to display in a box in the barchart plot, string
<b>returns</b>	barchart plot of the input data

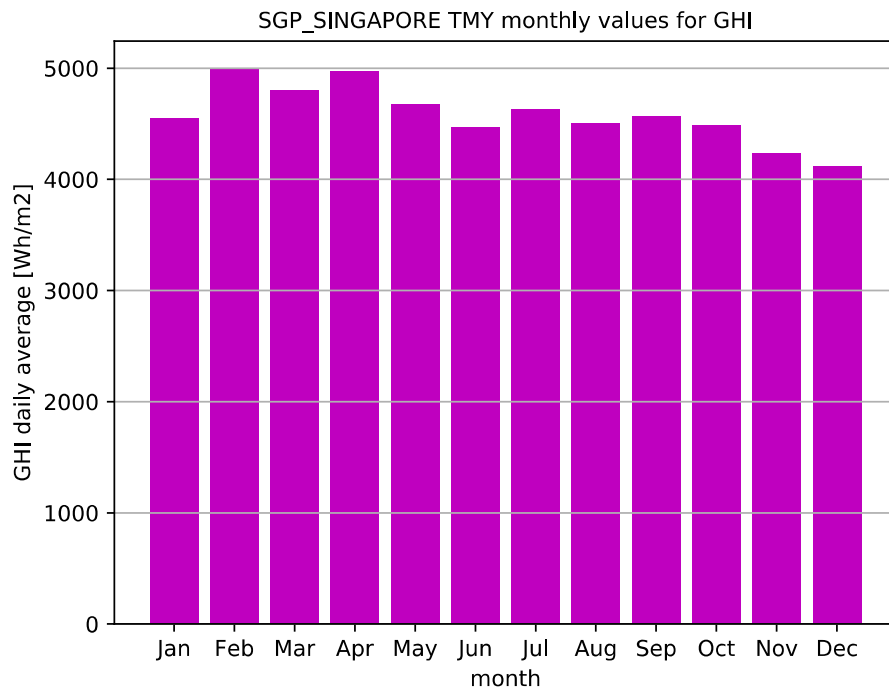
The function [month\\_barchart](#) is used to produce a barchart for 12 monthly values. Its application is shown in the following script.

Program\_Code 2-23: script, monthly totals from TMY file, draw barchart

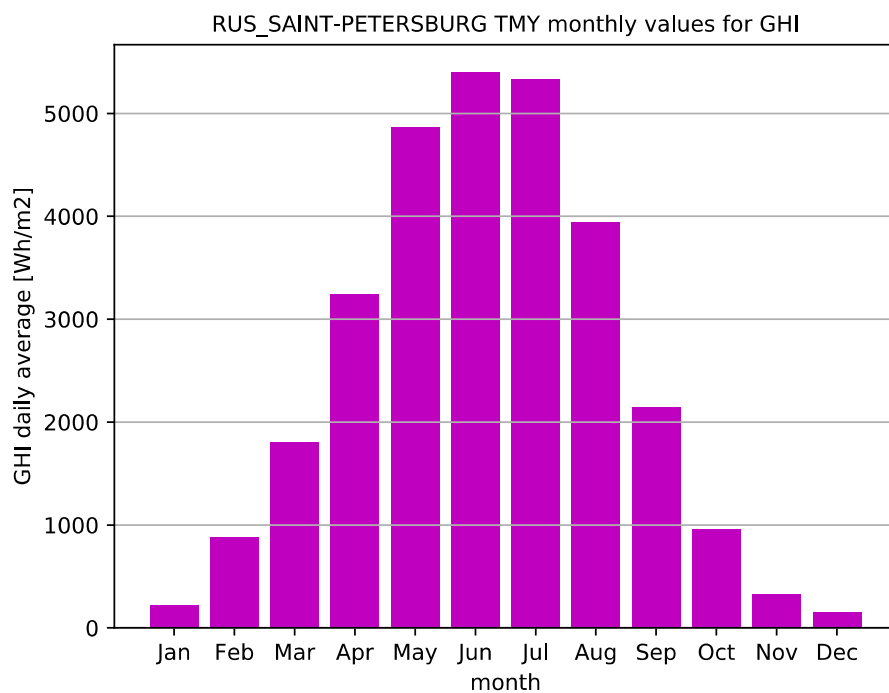
<b>script</b>	<a href="#">plot_tmy_barchart</a>
<b>description</b>	draw barchart over monthly accumulated values from a TMY dataset
<b>parameters</b>	<b>filename</b> : TMY (.csv) dataset file name with directory path, string <b>param</b> : parameter for barchart (TMY column name), string
<b>returns</b>	barchart of the selected TMY parameter for the months 1-12

The script [plot\\_tmy\\_barchart](#) reads a file in TMY format, extracts data for one column with 8760 hourly values and plots it over the year.

A function call to [tmy\\_monthly\\_total](#) returns a list [1..12] with the totals of the chosen parameter per month and a list with the monthly day averages. The list is passed to the [month\\_barchart](#) function to produce and display the graph (Figure 2-13, Figure 2-14).



*Figure 2-13: Day average of the energy yield in Singapore, near the equator. The differences between the first and second half of the year are due to the weather.*



*Figure 2-14: GHI solar irradiation in St.Petersburg (60°N). The marked difference between the summer and winter months is consequence of the high latitude.*



### 2.3.3 Heat map

One of the most interesting representations of TMY data is the heat map, which shows the variations of a parameter over two temporal dimensions, the day and the year. This representation is much more intuitive than that offered by simpler plots or barcharts.

The Python `matplotlib` package offers several tools for graphical data representation, including the generation of heat maps. The function `tmy_heatmap` makes use of this facility.

Program\_Code 2-24: function, heatmap from TMY data

<b>package</b>	<code>solprim.tmyutility</code>
<b>function</b>	<code>tmy_heatmap</code>
<b>description</b>	produce a heatmap plot for a parameter from a TMY dataset
<b>parameters</b>	<b>tmy_list</b> : list or dataframe column with hourly values [0..8759] for one TMY parameter, integer or float <b>header_text</b> : output plot header text, string
<b>returns</b>	heatmap plot with the selected parameter over the year (365d) and the day (24h)

A practical application is shown in the following script.

Program\_Code 2-25: script, plot arbitrary TMY parameter as heat map

<b>script</b>	<code>plot_tmy_heatmap</code>
<b>description</b>	plot the heatmap for a parameter from a TMY dataset
<b>parameters</b>	<b>filename</b> : TMY (.csv) dataset file name with directory path, string <b>param</b> : parameter to plot on heatmap / TMY column name, string
<b>returns</b>	heatmap plot with the selected parameter over the year (365d) and the day (24h)

The script `plot_tmy_heatmap` reads a file in TMY format, extracts data for one column with 8760 hourly values and plots it as heatmap with a call to the function `tmy_heatmap`.

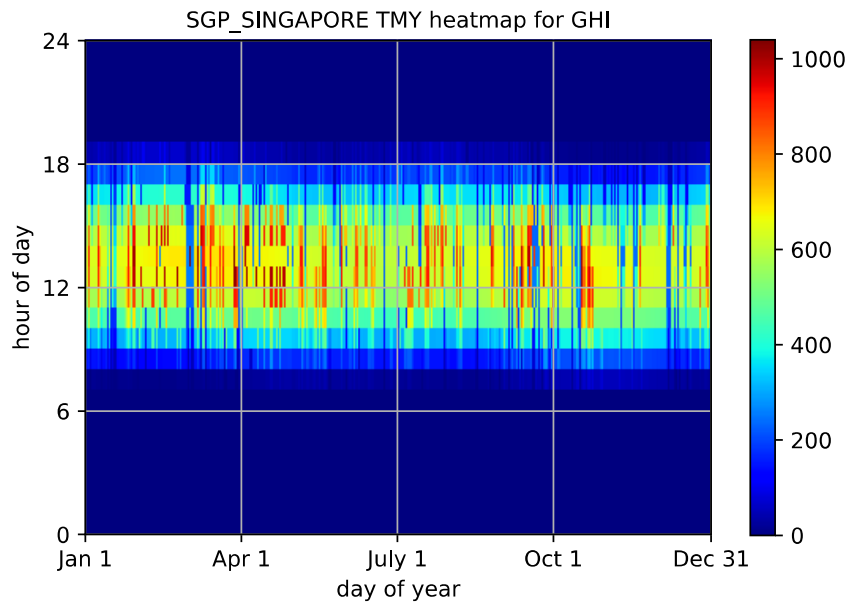
In Figure 2-15 through Figure 2-20 are shown the heat maps for Tdry and GHI in locations far apart climatically, Singapore, Sydney, Stockholm, and Kashgar/Kashi (China).

The heat maps for GHI give a good indication of the availability of the solar energy resource. Singapore is located almost on the equator and the GHI has a constant envelope over the year, with sunrise and sundown essentially at the same time every day and 12 hours average day length. The solar irradiation peak is not at noon, but between 13:00 and 13:30 and sunrise and sundown are not at 6:00 and 18:00, but postponed by 1 - 1.5 hours. The reason is that Singapore's time zone  $\tau_z=8$ , does not match the longitude  $\sim 104^\circ$ . The local time is ahead of the solar time by about one hour, with sunrise around 7:00 and sundown around 19:00. The time zone and the longitude are reported in the location metadata that is printed on the terminal display during execution of the Python script.

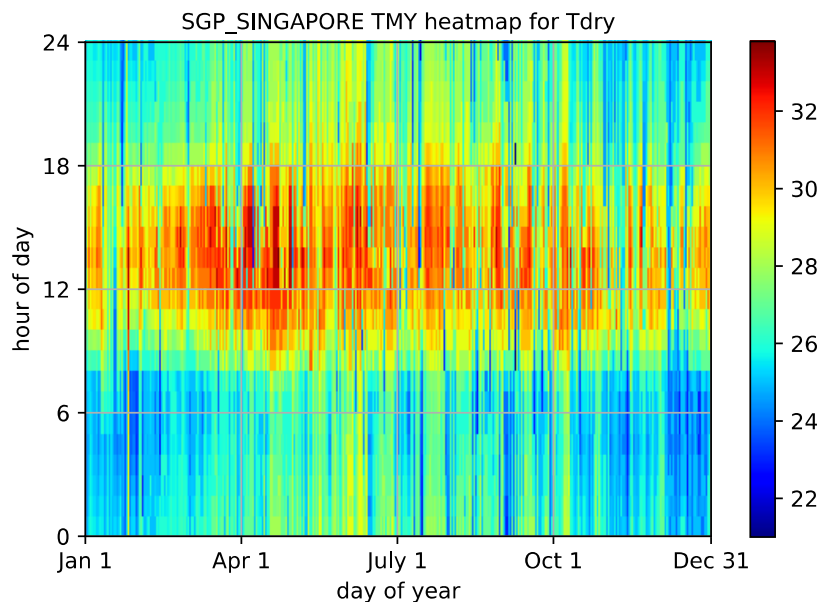
Sydney is a temperate city with good access to solar radiation. In the graphs the season effect with the hot peak around New Year is clearly visible (Figure 2-17, Figure 2-18).

The GHI heat map for Stockholm tells a different story. Due to its high latitude, at about  $60^\circ\text{N}$ , the difference between the length of the day in winter (Dec-Jan) and in summer (Jun-Jul) is clearly marked. At midsummer the first noticeable solar radiation is just past 3:00 and lasts until almost 21:00 (in winter these conditions are obviously reversed). The solar radiation envelope is quite symmetric around 12:00 noon, in fact Stockholm lies only  $3^\circ$  longitude away from the center of its meridian,  $18^\circ$  vs  $15^\circ$ . The difference between solar time and local time is here 12 minutes, too small to be noticeable in comparison to data that is sampled once per hour (Figure 2-19).

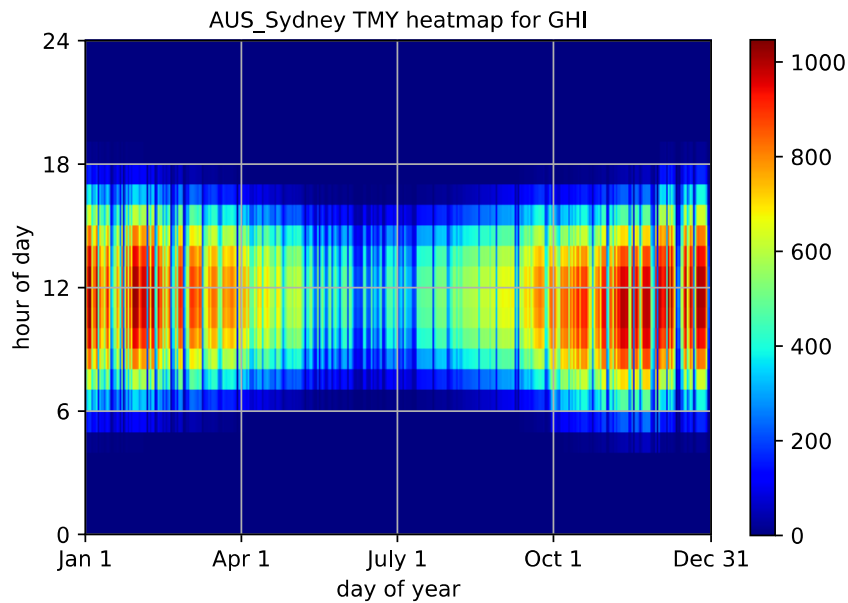
In the GHI plot for Kashi in the Xinjiang Uygur region in Western China (Figure 2-20) there is a 3-hour shift between solar time and local time. As mentioned in Section 1.2.3, China has one single time zone at  $\tau_z=8$ . Kashi is located at  $76^\circ\text{E}$ , barely away from the center of the meridian for  $\tau_z=5$  at  $75^\circ\text{E}$ . The sun is therefore highest in the Kashi sky when in Eastern China it is already mid-afternoon.



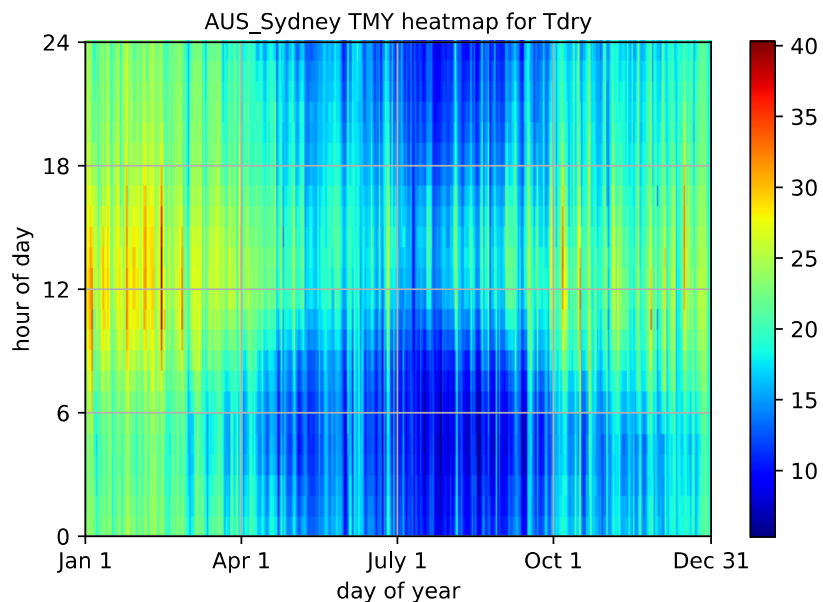
*Figure 2-15: Heatmap for GHI, Singapore (Wh/m2). Due to the equatorial location the length of the day is the same all year round.*



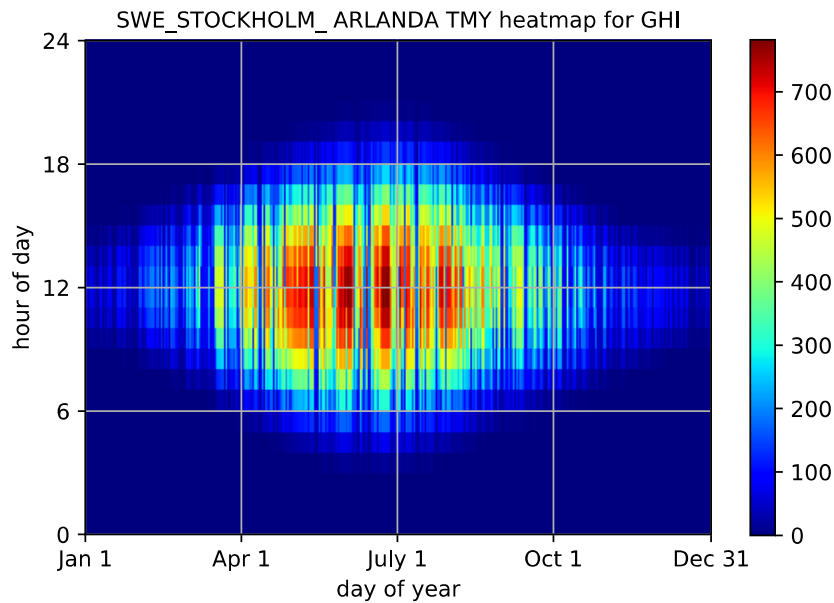
*Figure 2-16: Heatmap for Tdry, Singapore. Also here it is clearly visible that the differences between months are limited, the day/night cycle is more marked.*



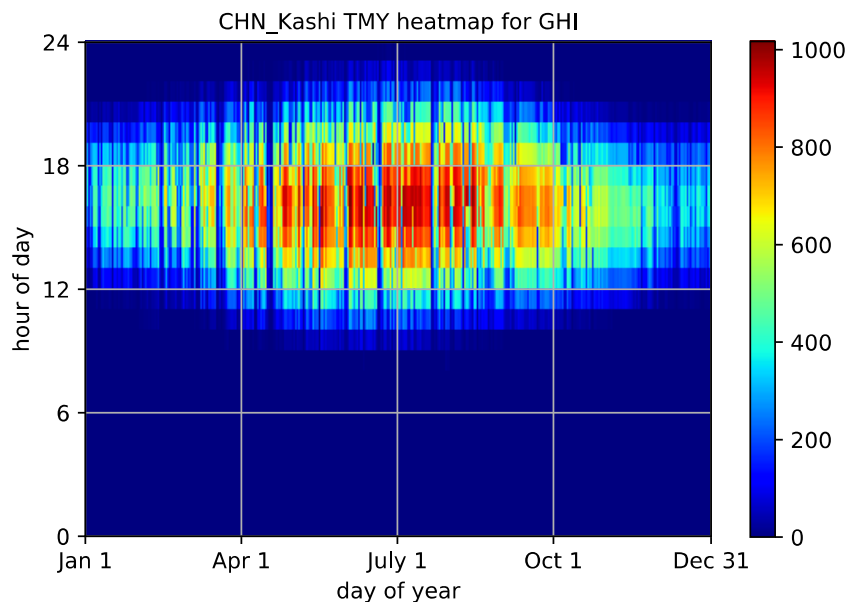
*Figure 2-17: In the GHI heatmap for Sydney the difference in solar energy availability between summer and winter is clear, as well as that summer is around New Year*



*Figure 2-18: The heatmap for Tdry in Sydney indicates marked differences between summer and winter*



*Figure 2-19: The GHI heatmap for Stockholm shows a concentration of solar energy availability in the summer months. Max irradiation is also lower than, e.g., in Sydney.*



*Figure 2-20: The most peculiar feature of the GHI heatmap for Kashi, Western China, is the timezone shift of the central hours of the day towards the mid-afternoon*

## 2.4 Solar energy collection over tilted surfaces

In most solar energy applications the solar radiation is collected on flat surfaces, such as PV panels or flat collectors for water heating. The amount of usable energy on a flat panel is the sum of the direct and the diffuse solar energy components (→ Section 1.4.2).

From TMY datasets, with knowledge of DHI and either GHI or DNI, it is possible to calculate the amount of energy collected on tilted, oriented, surfaces called **Plane-of-Array** (POA). The calculation is carried out in part with help of geometrical considerations, in part by using the direct (normal) and diffuse irradiance values DNI and DHI.

On a flat panel with given orientation and tilt the direct (beam) component of collected solar energy is given by the product of the beam radiation with the cosine of the angle of incidence (→ Section 1.5, Equation 1-21):

$$h(\theta) = h_0 \cdot \cos\theta \quad (1-21)$$

with  $h_0$  equal either the beam radiation  $B_n$  or DNI over a time interval.

The diffuse radiation collected by a tilted surface is usually calculated under the assumption that the radiation intensity is isotropic. In this case a simple relation for the diffuse radiation collected by a flat surface with the tilt angle  $\beta$  is expressed by Equation (1-24):

$$D_\beta = D_h \cdot \frac{1}{2} \cdot (1 + \cos\beta) \quad (1-24)$$

The calculation of the overall energy collected by a tilted and oriented flat surface is carried out by the function of Program\_Code 2-26.

Newer models for the estimation of diffuse radiation assume that this is not isotropic, that is, it is different for different portions of the sky. In particular, according to these models the diffuse radiation is stronger near the sun and in the direction of the equator. Python non-isotropic models for direct and diffuse radiation are presented in the Python **pvl** package and related documentation (Main Reference Sites, Software, p.9).

### Program\_Code 2-26: function, Plane-of-Array (POA) energy from TMY data

<b>package</b>	<a href="#">solprim.tmyutility</a>
<b>function</b>	<a href="#">tmy_plane_of_array</a>
<b>description</b>	from hourly TMY dni and dhi data calculate the hourly collected energy (Plane-of-Array, POA) on an unit-area (1 m <sup>2</sup> ) surface described by orientation and tilt angles
<b>parameters</b>	<b>dni_list</b> : list or dataframe column with DNI hourly values [0..8759], integer/float <b>dhi_list</b> : list or dataframe column with DHI hourly values [0..8759], integer/float <b>latitude</b> : latitude in decimal degrees (North is positive), float <b>longitude</b> : longitude in decimal degrees (East is positive), float <b>timezone</b> : timezone (East is positive), integer/float <b>plane_azim</b> : oriented surface azimuth angle, degrees, float <b>plane_tilt</b> : oriented surface tilt angle, degrees, float
<b>returns</b>	tuple with - list with hourly values [0..8759] for Plane-of-Array irradiation (W) - list with hourly values [0..8759] for direct radiation on plane, float - list with hourly values [0..8759] for diffuse radiation on plane, float - list with hourly values [0..8759] for incidence angle on plane, degrees, float

The function [tmy\\_plane\\_of\\_array](#) computes the total amount of solar energy incident on a tilted, oriented panel over the 8760 hours of the year. It returns two lists for the calculated values, on a hourly and a monthly basis. In the function hours are generated in a loop 0..8759. During daytime, when the solar irradiation components DNI and DHI are > 0, for each hour is calculated the solar position with the related incidence angle on the flat panel. The DNI component is scaled with the incidence factor and DHI is a fixed share of the overall diffuse irradiation, with the proportionality factor given by Equation (1-24).

DNI and DHI are recorded with reference to the local time (as shown in → Section 2.2.3). Due to the fact that they need to be processed together with the solar position, the local time must be calculated with consideration of the effects of longitude, timezone, and Equation of Time. The code of [tmy\\_plane\\_of\\_array](#) includes these corrections.

### Program\_Code 2-27: script, POA solar energy collection, graph

<b>script</b>	<b>plot_poa_graph</b>
<b>description</b>	from a TMY dataset for a given location, the azimuth and one or more tilt angles for an oriented flat panel the Plane-of-Array (POA) irradiation is calculated on a hourly basis and displayed in a graph
<b>parameters</b>	<b>filename</b> : TMY (.csv) dataset file name with directory path, string <b>plane_azim</b> : oriented surface azimuth angle, degrees, float <b>plane_tilt_list</b> : list of oriented surface tilt angles, degrees, float
<b>returns</b>	graph with the POA yearly irradiation at the location for one or more tilt angles

The script **plot\_poa\_graph** calculates a hourly list of POA generation [ $\text{W m}^{-2}$ ] based on the DNI, DHI for the selected location.

In Figure 2-21 and Figure 2-22 is shown the outcome of solar irradiation on a daily basis during the year in Abu Dhabi and Berlin. In Abu Dhabi a tilt angle =  $20^\circ$  (for an azimuth orientation =  $180^\circ$ ) gives the smoothest result over the year. In Berlin the seasonal differences in solar energy availability are larger and no tilt angle of the capturing surface can smooth them. Moreover, the optimal tilt angle does not only depend on the latitude but also on the share between direct and diffuse radiation that is available at a location, i.e., from the weather. This will be shown by the example of Program\_Code 2-30, Figure 2-24 and Figure 2-25



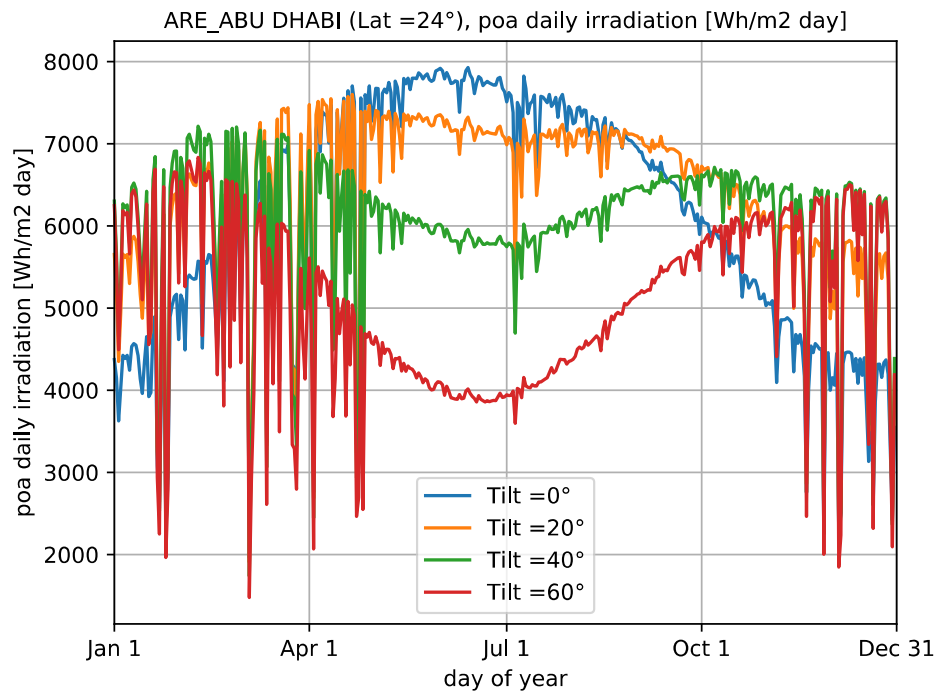


Figure 2-21: Daily irradiation [Wh/m<sup>2</sup>] on a solar flat panel at azimuth = 180° and for different tilt angles during one year in Abu Dhabi

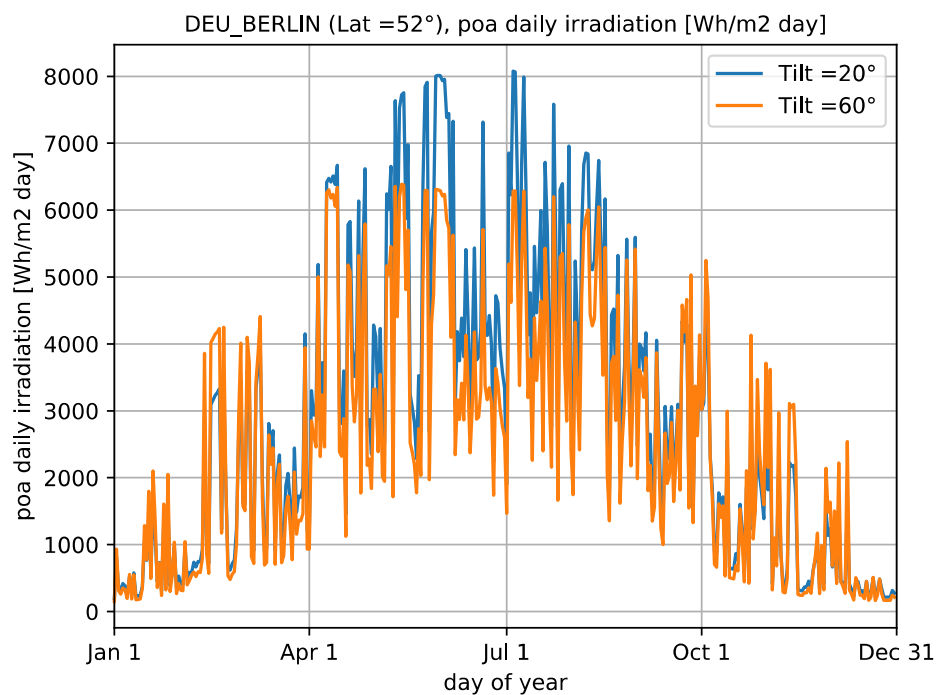


Figure 2-22: Daily irradiation on a solar flat panel at azimuth = 180° and for different tilt angles for the TMY year in Berlin

### Program\_Code 2-28: script, POA result verification

<b>script</b>	<b>test_poa_result</b>
<b>description</b>	utility script to test the outcome of the function 'tmy_plane_of_array'. For an input TMY dataset and a plane azimuth and tilt is generated the specific POA radiation collection for the 8760 hours of the year. The outcome can be easily compared with the result of other programs or online services.
<b>parameters</b>	<b>filename</b> : TMY (.csv) dataset file name with directory path, string <b>plane_azim</b> : oriented surface azimuth angle, degrees, float <b>plane_tilt</b> : oriented surface tilt angle, degrees, float
<b>returns</b>	file in .csv format with the parameters calculated on hourly basis by the 'tmy_plane_of_array' function: plane-of-array radiation, direct radiation over the tilted surface, diffuse radiation, incidence angle

The script **test\_poa\_result** is used to verify the operation and the quality of the result of the **tmy\_plane\_of\_array** function (Program\_Code 2-26). It produces a table with the most important original and calculated parameters to be used for comparison with the result of other programs or online resources for the solar position calculation.

An example of output of **test\_poa\_result** for the TMY dataset of the Honolulu International Airport is the following

Year	Month	Day	Hour	Incld	GHI	DNI	DHI	POA_dir	POA_dif	POA_total
1987	1	1	0	0	0	0	0	0.00	0.00	0.00
1987	1	1	1	0	0	0	0	0.00	0.00	0.00
1987	1	1	2	0	0	0	0	0.00	0.00	0.00
1987	1	1	3	0	0	0	0	0.00	0.00	0.00
1987	1	1	4	0	0	0	0	0.00	0.00	0.00
1987	1	1	5	0	0	0	0	0.00	0.00	0.00
1987	1	1	6	84.8	58	315	23	28.53	20.56	49.09
1987	1	1	7	71.36	207	587	47	187.59	42.02	229.61
1987	1	1	8	57.7	413	834	36	445.71	32.18	477.90
1987	1	1	9	43.91	603	938	48	675.80	42.91	718.72
1987	1	1	10	30.11	710	964	53	833.90	47.38	881.28

Several comparison testruns with the SAM program (Main Reference Sites, Software, p.9) indicate maximum differences in the outcome for the POA total in the order of 1–1.5%.

### Program\_Code 2-29: script, POA solar energy collection, barchart

<b>script</b>	<b>plot_poa_barchart</b>
<b>description</b>	from a TMY dataset for a given location and the azimuth, tilt angles for an oriented flat panel calculate the plane-of-array (POA) irradiation on a hourly basis, calculate the monthly totals and display on a barchart
<b>parameters</b>	<b>filename</b> : TMY (.csv) dataset file name with directory path, string <b>plane_azim</b> : oriented surface azimuth angle, degrees, float <b>plane_tilt</b> : oriented surface tilt angle, degrees, float
<b>returns</b>	barchart with the POA irradiation totals over the tilted plane for the 12 months of the year

The script `plot_poa_barchart` produces a barchart of POA generation totals for the 12 months for given azimuth and tilt of a collecting surface. An example for Bangkok is shown in Figure 2-23.

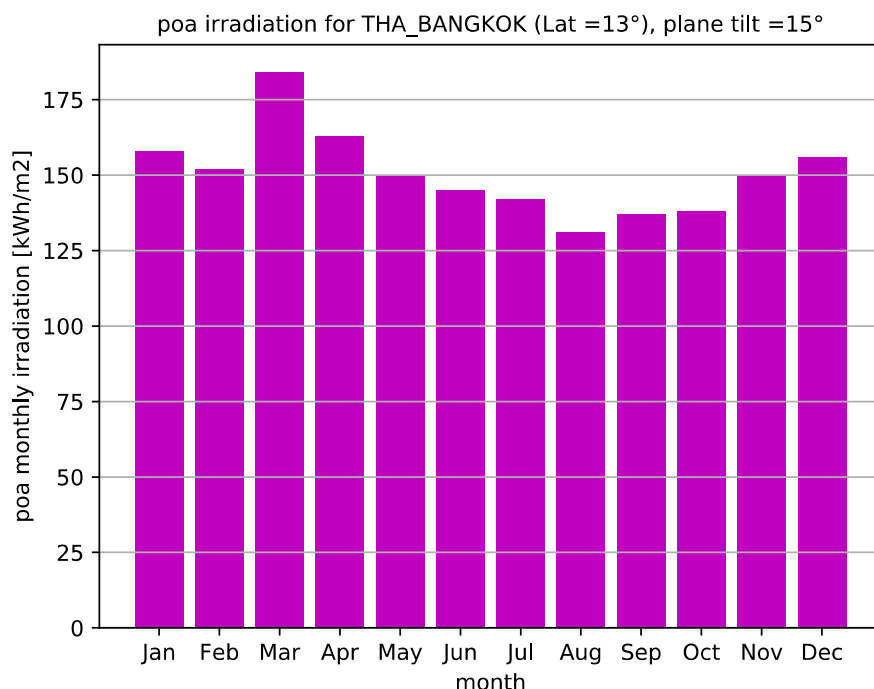


Figure 2-23: POA total energy yield per month in Bangkok, Thailand, for a flat surface with tilt =15°

### Program\_Code 2-30: script, POA solar energy collection, heatmap

<b>script</b>	<a href="#"><code>plot_poa_heatmap</code></a>
<b>description</b>	from a TMY dataset for a given location and the azimuth, tilt angles for an oriented flat panel calculate the Plane-of-Array (POA) irradiation on a hourly and monthly basis and display as heatmap
<b>parameters</b>	<b>filename</b> : TMY (.csv) dataset file name with directory path, string <b>plane_azim_from</b> : oriented surface azimuth angle start, integer <b>plane_azim_to</b> : oriented surface azimuth angle end, integer <b>plane_azim_step</b> : azimuth angle incremental step, integer <b>plane_tilt_from</b> : oriented surface tilt angle start, integer <b>plane_tilt_to</b> : oriented surface tilt angle end, integer <b>plane_tilt_step</b> : tilt angle incremental step, integer
<b>returns</b>	heatmap graph with the POA irradiation for location and oriented panel in the course of the year

A heatmap is an intuitive representation of the variations of a parameter, represented by a changing color, over two dimensions (→ Section 2.3.3). The heatmap is useful also to assess the best orientation angles for a solar panel, calculating the energy yield from a TMY dataset and different combinations of azimuth and tilt angles. Most energy is usually collected for panels pointing to the equator, but this may differ for repeated weather patterns concentrated in the morning or afternoon hours.

The script [`plot\_poa\_heatmap`](#) produces a heatmap for a TMY dataset and a given range of azimuth and tilt angles. The script is very processing-intensive, as for any combination of azimuth and tilt must be calculated the yearly POA yield over the 8760 TMY hours. For this reason the script makes a full calculation only at major steps (e.g., every 10 or 15 degrees) and interpolates the result for the intermediate values.

In Figure 2-24 and Figure 2-25 is shown the POA yield calculated by [`plot\_poa\_heatmap`](#) with and without the contribution of the solar diffuse radiation component.

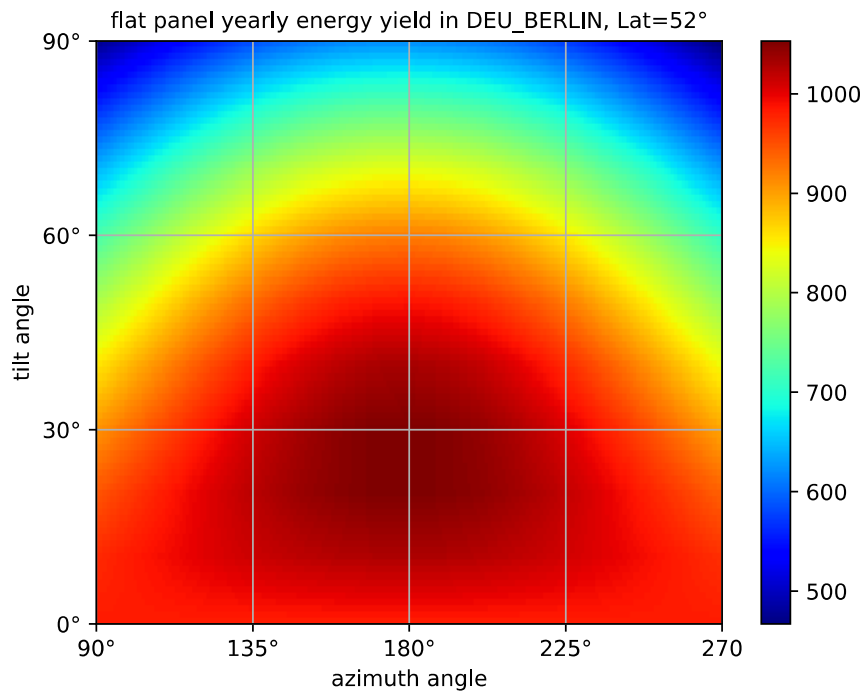


Figure 2-24: Energy yield [ $\text{Wh m}^{-2}$ ] for a flat surface in Berlin, at 52N°. Due to the strong diffuse energy component, a tilt angle  $\sim 25^\circ$  gives the maximum energy return.

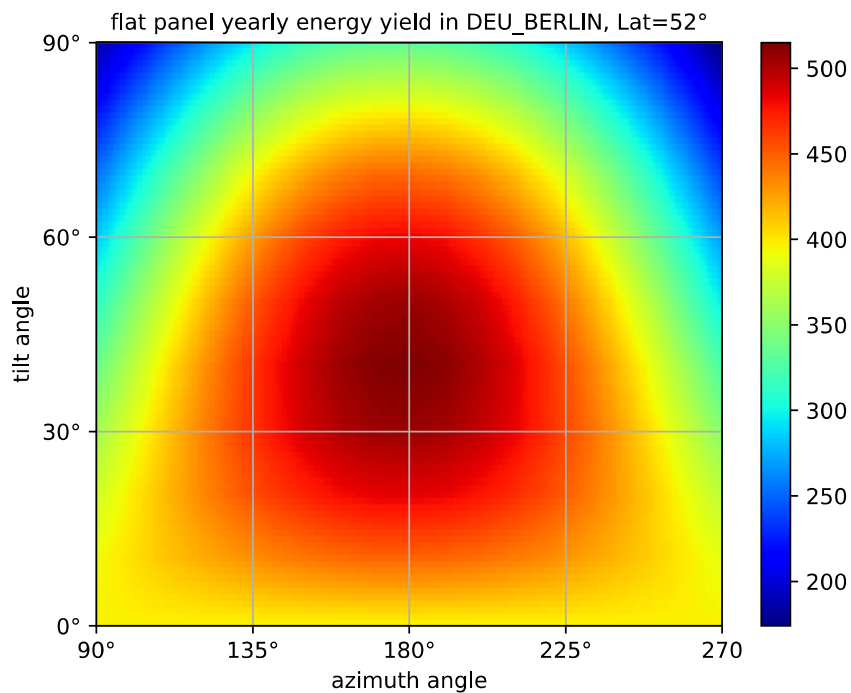


Figure 2-25: Energy yield [ $\text{Wh m}^{-2}$ ] for a flat surface in Berlin, direct solar energy component only. The maximum energy yield is achieved at an higher tilt angle.

## 2.5 TMY data representation on the psychrometric chart

### 2.5.1 Moist air and the psychrometric chart

An essential tool, both conceptual and practical, in the dimensioning of the heating, cooling, and air conditioning (HVAC) functions in buildings is the psychrometric chart, on which are conveniently represented the thermodynamic variables for moist air. On a psychrometric chart the description of state changes, for example, the determination of the amount of energy that is required to raise or lower the air temperature at a certain humidity percentage is quite simple. At a first look a psychrometric chart may seem complex and difficult to use, however, it just reflects basic physical principles and helps in carrying out calculations that would otherwise present a certain complexity.

Moist air is the mixture of dry air and water vapor ( $H_2O$ ) and psychrometry is the study of their thermodynamic properties. The analysis of processes related to moist air, such as air conditioning and drying, is very important both because of the role they play in human comfort as well as their high energy demand.

Dry air is a mixture of several gases (nitrogen, oxygen, argon, carbon dioxide, and other gases) and as such it does not exist in nature – in the real world air always contains some moisture. Dry air and water vapor do not combine physically or chemically, though they share the same volume and have the same pressure and temperature. The capacity of dry air to absorb water vapor depends on the pressure and on the temperature, in particular, it rapidly increases with the temperature. At ambient temperature the quantity of water vapor contained in air is in the order of up to 20-25 grams per kg dry air. Despite such little amount the properties of humid air, in particular its heat content, depend in large part on the water fraction rather than on the dry mass of air.

TMY data for temperature and relative humidity can be represented on a psychrometric chart. This provides an immediate indication of the climate of a location and its potential impact, for example, on the design of a building envelope and HVAC equipment. The psychrometric chart representation is particularly important in passive and energy-efficient design, including the use of solar energy, because it allows the detailed analysis of the dynamic behavior of a building and the assessment of the contribution of renewable resources.

### 2.5.2 Basic psychrometric relations

A psychrometric chart can have different variables on its main axes, while other physical data is indicated by lines. A practical representation of temperature and humidity values, which is useful also for the assessment of the energy requirements for heating, cooling, and dehumidification, shows the temperature on the  $x$  axis and the water content on the  $y$  axis. This value is not explicitly contained in TMY datasets, which instead present the moisture content either as relative humidity (RH) or as dew point temperature. Here we shall see how to convert RH values into the absolute water content.

Dry air at a certain temperature can only contain a certain amount of water in gaseous form, i.e., water vapor. The maximum pressure that the water vapor can reach at a certain temperature before it begins to condensate is called the **saturation pressure**,  $p_{ws}$ . Values for the saturation pressure are obtained from physical measurements and tabulated.

Two concepts are central to the analysis of moist air, the relative humidity and the humidity ratio. The **relative humidity**  $\phi$  (%) is the ratio of the partial water vapor pressure  $p_w$  at a certain temperature to the saturation pressure  $p_{ws}$  of air at the same temperature, expressed as a percentage:

$$\phi = \frac{p_w}{p_{ws}} \cdot 100\% \quad (2-5)$$

The relative humidity  $\phi$  indicates therefore the water content in air as fraction of its possible maximum at a certain temperature.  $\phi$  is always  $\leq 1$ , for saturated air  $\phi = 1$  (or, in percentage terms, 100%). The relative humidity is very important in the analysis of ambient comfort because the moisture exchange between the body and the air depends on it. It is also important for the analysis of drying processes.

The **humidity ratio**  $x$  is the ratio of the actual mass of the water vapor present in moist air to the mass of the dry air. In SI-units the humidity ratio is expressed in gram water vapor per kilogram dry air, or also kg water vapor per kg dry air:

$$x = \frac{m_w}{m_a} \quad (2-6)$$

with humidity ratio  $x$  (g water or kg water/kg air),  $m_w$  the mass of water vapor (g or kg) and  $m_a$  the mass of dry air (kg), not including the mass of the moisture. At 20°C 1 kg very damp air contains ca. 20-25 g water.

The total mass of moist air is composed of the masses of dry air and of the water vapor:

$$m_t = m_a + m_w = (1+x) \cdot m_a \quad (2-7)$$

and

$$p_t = p_a + p_w \quad (2-8)$$

that is, the total pressure of moist air is the sum of the partial pressure of the dry air and the partial pressure of water vapor.

The humidity ratio  $x$ , i.e., the actual water content in unsaturated moist air, can be calculated from the relative humidity  $\phi$  with help of the state equation of the ideal gases, and of the definition of  $\phi$  and  $x$  (2-5), (2-6) in the following way.

The masses of the dry air and the water vapor at a temperature  $T$  in a volume  $V$  ( $T$  and  $V$  are the same for both gases) are:

$$m_a = \frac{p_a \cdot V}{R_a \cdot T} = \frac{(p_t - p_w) \cdot V}{R_a \cdot T} \quad (2-9)$$

$$m_w = \frac{p_w \cdot V}{R_w \cdot T} \quad (2-10)$$

By referring to the humidity ratio  $x$  defined in Equation (2-6) it is possible to eliminate  $V$  and  $T$  from the ratio of the two masses:

$$x = \frac{m_w}{m_a} = \frac{R_a}{R_w} \cdot \frac{p_w}{(p_t - p_w)} = 0.622 \cdot \frac{p_w}{(p_t - p_w)} \quad (2-11)$$

the humidity ratio  $x$  can be expressed as function of the relative humidity  $\phi$  of Equation (2-5):

$$x = \frac{R_a}{R_w} \cdot \frac{\phi \cdot p_{ws}}{(p_t - \phi \cdot p_{ws})} = 0.622 \cdot \frac{\phi \cdot p_{ws}}{(p_t - \phi \cdot p_{ws})} \quad (2-12)$$

with  $p_t$  the total pressure of the moist air (dry air together with water vapor),  $p_w$  the partial pressure of water vapor in the moist air,  $p_{ws}$  the saturation pressure and  $(p_t - p_w)$  the partial pressure of dry air. The coefficient 0.622 is the ratio of the individual gas constants of water and



dry air respectively, that is, of their molecular weights. Equations that return  $x$  in gram instead of kg per kg dry air use the coefficient = 622 to include the  $\times 1000$  scaling factor.

Equation (2-12) is one of the most important relations of psychrometrics. At a given total pressure, which depends on the temperature and the volume, it defines the relation between the humidity ratio  $x$  and the relative humidity  $\phi$ . At the saturation pressure ( $\phi = 1$ ) the equation becomes

$$x_s = 0.622 \cdot \frac{p_{ws}}{p_t - p_{ws}} \quad (2-13)$$

with the index  $s$  referring to the saturation values for the humidity ratio and the saturation pressure of water vapor.  $p_t$  is the pressure of the moist air, which in open systems is the same as the atmospheric pressure. The standard atmospheric pressure is defined as 1013.25 mbar or 101.325 kPa, it can vary depending on temperature, height over sea level, weather.

### 2.5.3 Psychrometric chart representation

The relations presented in Section 2.5.2 are sufficient to build a psychrometric chart to plot TMY values, as shown in the following script.

Program\_Code 2-31: script, TMY psychrometric chart data representation

<b>script</b>	<b>psychro_display</b>
<b>description</b>	plot RH vs Tdry values as scattered points, show the psychrometric diagram with comfort zone, count the hourly values falling into the comfort zone and in other clusters identified by limit temperatures
<b>parameters</b>	<b>filename</b> : TMY (.csv) dataset file name with directory path, string. The dataset must contain values for Tdry, RH, pressure. <b>T_freeze, T_cold, T_cool, T_warm, T_hot</b> : temperature limits for the clusters, integer/float <b>RH_low, RH_high</b> : relative humidity limits for the comfort cluster, integer/float
<b>returns</b>	psychrometric diagram with scattered points from a TMY dataset represented as humidity ratio vs dry temperature and with different colors depending on their temperature and relative humidity values

The script **psychro\_display** displays on a psychrometric diagram the dry temperature ( $T_{dry}$ ) and relative humidity (RH) values from a TMY dataset. At the beginning the script sorts the input data into different bins as function of  $T_{dry}$  and RH. The temperature limits are selected according to the different principles and technologies for heating, cooling, and dehumidification for the respective temperature ranges (the clusters are briefly described in → Section 2.5.4). The number of points located in each cluster is then counted, this provides an immediate indication of a location's climatic conditions.

The script is structured in one main part and some local functions.

The function **saturation\_pressure** accepts a input temperature value and returns the water vapor saturation pressure (RH=100%) in dry air. The range for the input temperature is [-40..+50] degrees °C, corresponding to output saturation pressures in range [0.013..12.35] kPa. The function refers to tables of corresponding values for temperature and saturation pressure sourced by thermodynamic property tables and stored for each

degree °C. A tuple contains the lists of equal length for the temperature and pressure values. After having found the index corresponding or nearest the input value, the same index points to the output value in the other list. Intermediate values are interpolated.

The function `x_from_rh` calculates the water content  $x$  in g H<sub>2</sub>O per kg dry air for a given temperature (°C), relative humidity (RH%), and air pressure (mbar). The function is based on the relation of Equation (2-12).

The function `psychro_diagram` draws the psychrometric diagram background. It accepts as input a minimum and a maximum temperature to scale the  $x$  axis and a reference pressure value to calculate and draw the curves for the relative humidity. These values are obtained from the TMY `Tdry` and pressure data. The year averages of these parameters are passed to the function `psychro_diagram` to provide for the right axes scaling. This is particular important for high locations with a lower average pressure than at sea level. Default input values for the functions are `t_min=-20`, `t_max=50`, `pressure=1013.25`.

The main `psychro_display` script performs the following steps:

- read TMY data from the indicated directory and filename
- assign `Tdry`, `RH` values to clusters (implemented as Python lists) depending on the `Tdry` value in range: freeze, cold, cool, comfort, warm, hot. For inclusion in the “comfort” cluster also the `RH` value must be within a preset range.
- call the function `psychro_diagram` to draw the plot background. The limit temperatures for the  $x$  axis are defined from the `Tdry` data range and the reference pressure is the average of the location pressure over the year.
- for each hour in the TMY dataset calculate the humidity ratio  $x$  from the `Tdry`, `RH`, and pressure values
- plot scattered points as function of `Tdry`, `RH` with colors depending on the clusters they belong to. Each dot on the plot represents the temperature and humidity of one of the 8760 hours of the year.
- count the number of points in each cluster, prepare description text to be placed on the graph and displayed on the output device.

The script output is the TMY data representation on the psychrometric diagram and a list of the point count in the different clusters, presented both on the graph and on the Python shell terminal. This data supports a basic thermal analysis of buildings at the TMY location, as described in the following Section.

#### 2.5.4 Ambient conditions and building thermal analysis

Buildings must provide a comfortable indoor environment, defined by a narrow range of inhouse temperature and relative humidity values, under the expected and unfavorable outdoor temperature and humidity conditions. The available technologies to provide indoor comfort are either passive, i.e., they do not need any active energy input, or active, that is, to operate they require an external energy input in form of electricity or heat. Examples of passive technologies are the building shell, the building's thermal mass, even its design and orientation, as well as solar collectors (which do not require electricity or fuels). Active technologies are, for example, gas heaters, heat exchangers for district heating, air conditioners.

In the script `psychro_display` the TMY data is organized in **clusters** depending on the temperature and relative humidity of each data point. The clusters are divided as 'freeze', 'cold', 'cool', 'comfort', 'warm', and 'hot', whereby the limit temperatures between them are not fixed but can be set manually. Each cluster corresponds to a particular passive or active heating or cooling technology, as briefly described in the following.

For ambient temperatures in the freeze range (the upper default limit value is the freezing temperature  $=0^{\circ}\text{C}$ ) active heating must in most cases be provided, which requires the provision of energy from an external source.

In the cold ambient temperature range (default limits  $0^{\circ}$ – $12^{\circ}\text{C}$ ) active heating might or not need to be provided, depending on design solutions such as thermal insulation and solar heating.

In the cool temperature range ( $12^{\circ}$ – $20^{\circ}\text{C}$ ) passive solutions, such as insulation and thermal storage, are usually sufficient to provide indoor comfort.

The comfort range is defined for temperatures between 20°–24°C and relative humidity  $40\% < RH < 60\%$ . In such conditions there isn't any need to do anything to provide indoor comfort. However, for most locations an analysis of the datasets indicates that natural comfort conditions are verified only for some percent of the time, which means at most one or two weeks per year. When RH is outside the comfort range humidification or dehumidification must be provided, and this requires much energy.

The 'warm' cluster contains the datapoints at temperatures in the range 24°–30°C and those in range 20°–24°C, but with RH values outside the comfort zone. Here some passive solution, such as shading, or some active solution with moderate energy requirements, for example, ventilation, should be sufficient to bring indoor temperature values to the comfort zone.

In the hot temperature range (temperatures at 30°C and above) active cooling is required. The energy demand is, as a rule, high.

Clusters on a psychrometric chart provide an immediate feeling for the climate of any particular location and for the energy demand of a building. Different solutions can also be compared. For example, with better insulation or higher thermal mass it is possible to lower the limit between the 'cold' and the 'cool' clusters. The different count in the two clusters would then provide an indication of how much energy is saved with the passive measure compared to active heating. By considering the initial investments and the variable costs it can be assessed whether active or passive design solutions are more suitable.

In Figure 2-26 is shown a data example for New York. The climate is temperate with the coldest temperature at -13.9°C and not too many hot spells in summer. Humidity is also not too high. 48.3% of the time the temperature is below 12°C and only for 325 hours during the year, 3.7% of the time (13 days), it is >30°C. From the psychrometric plot it is evident that the most important measure for ambient comfort is heating. A plant would need to be designed for an average temperature of 3.9°C and operate in the range -14°C ... +12°C for 4233 hours, or about six months in the year.

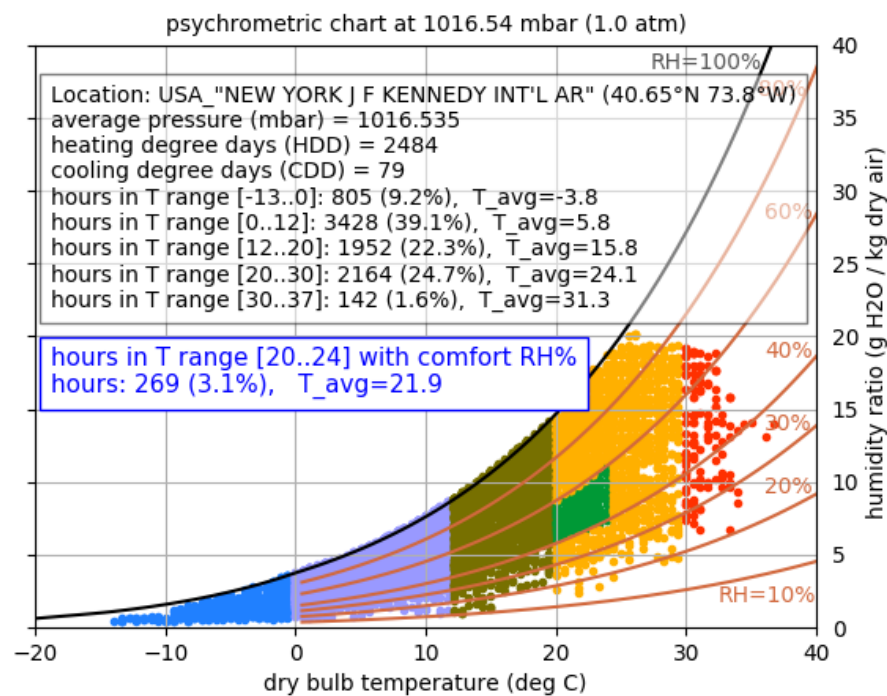


Figure 2-26: Psychrometric chart for New York

In Singapore (Figure 2-27) the climate is hot and humid all year round. The demand for cooling and conditioning is permanent. In Moscow the highest demand is for heating during the winter, while in summer really discomfort ambient conditions are rarely reached (Figure 2-28).

In Figure 2-29 is shown the psychrometric plot for Berlin with the limit temperature between 'cold' and 'cool'  $T_{cool} = +12^{\circ}\text{C}$  and in Figure 2-30 with  $T_{cool} = +5^{\circ}\text{C}$ , which could be achieved, for example, with better insulation. This measure helps to shift 2325 hours, more than 25% of the total year length, from the cold zone requiring active heating to a zone where passive measures would be sufficient. This is not a new measure for energy efficiency, just a way of showing the effect of insulation and passive solutions with help of TMY data and the psychrometric chart.

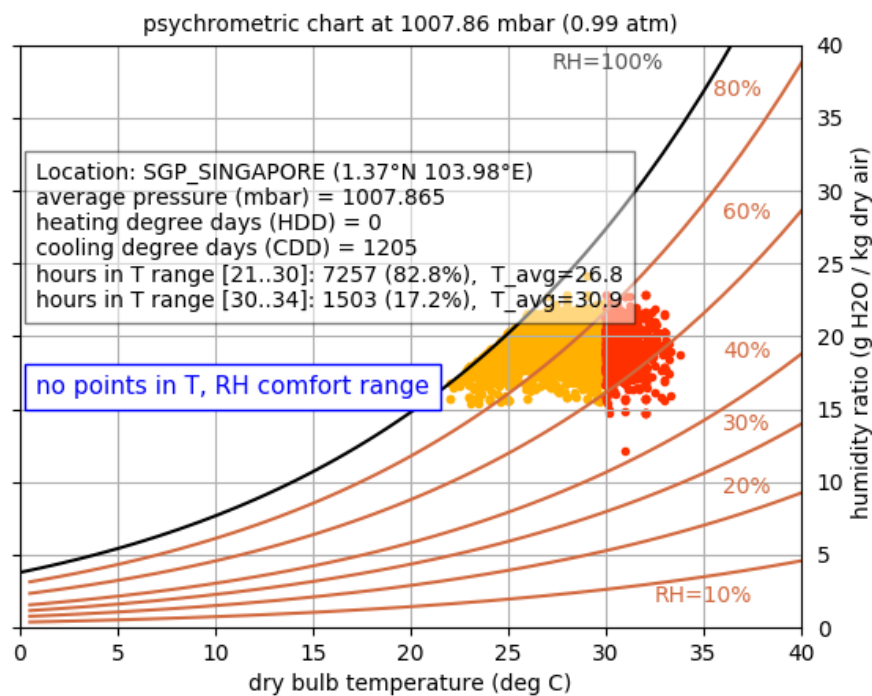


Figure 2-27: Psychrometric chart for Singapore TMY data. The climate is hot and humid all year round.

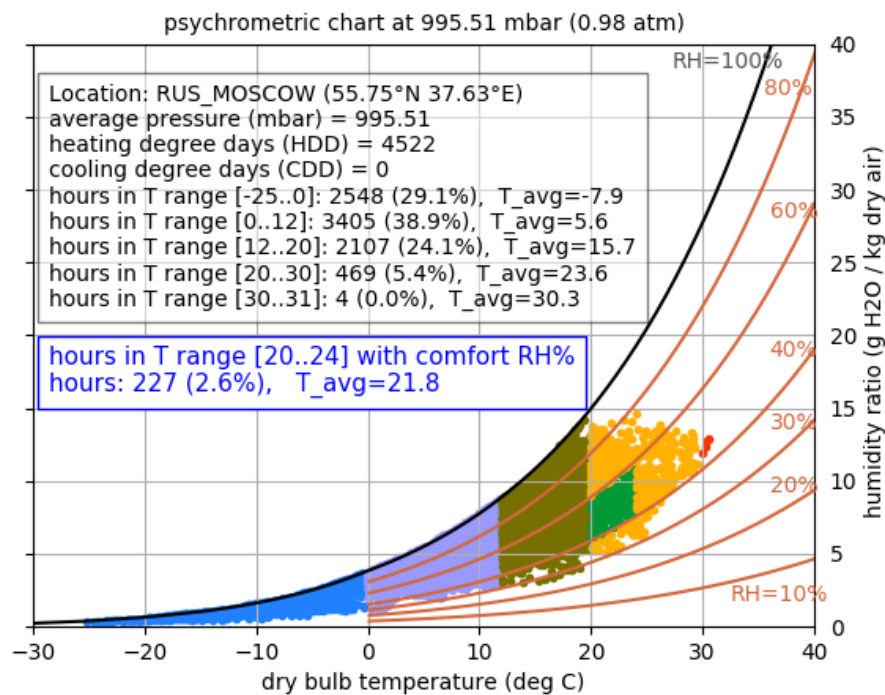


Figure 2-28: Psychrometric chart for Moscow TMY data, with a prevalently cold climate



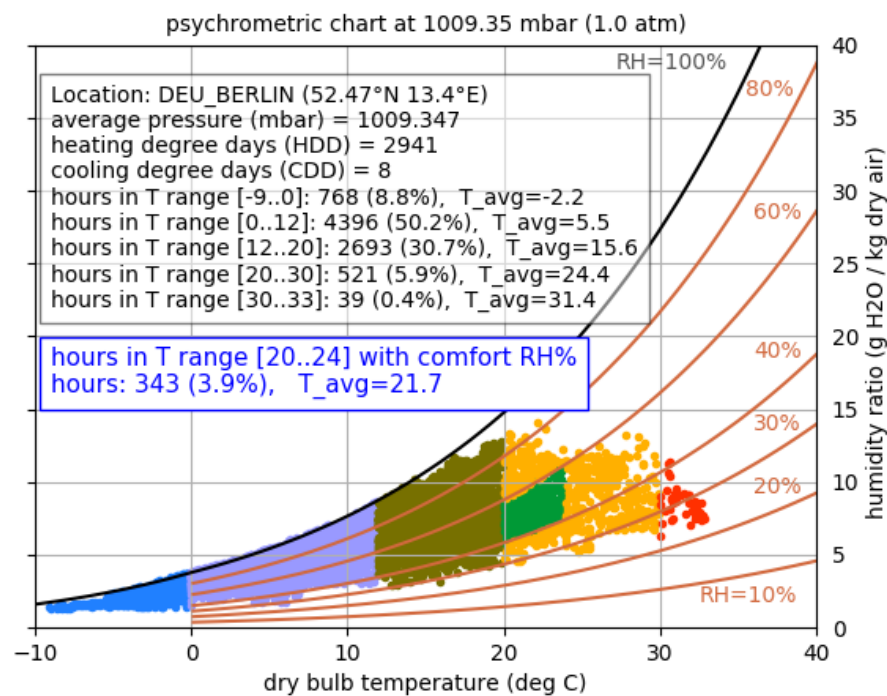


Figure 2-29: Psychrometric chart for Berlin TMY data (1). For a standard building the border between 'cool' and 'cold' ambient temperatures is taken as +12°C.

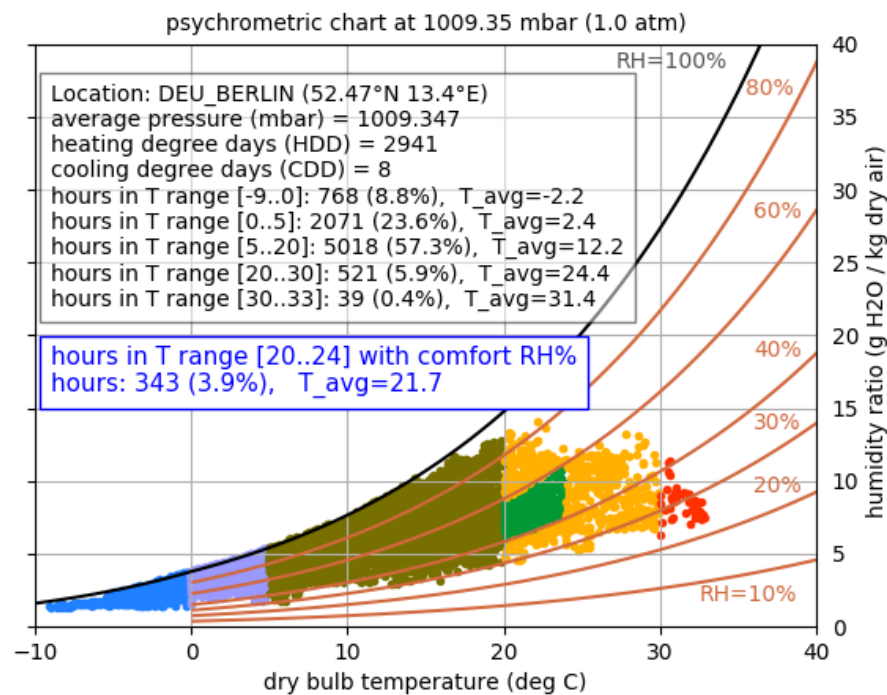


Figure 2-30: Psychrometric chart for Berlin (2). With an outside 'cool' temperature limit = +5°C the number of points in the cool range almost doubles.



### Online\_Resource 2-6: ClimateConsultant

Climate Consultant is a simple to use, graphic-based program for the presentation and analysis of climate data in TMY / EPW format. From the tabulated data Climate Consultant produces meaningful graphic displays. Climate Consultant has been developed by the University of California in Los Angeles, Department of Architecture and Urban Design.

The Psychrometric Chart is probably the most interesting feature of Climate Consultant. Dots for the temperature and humidity values of each hour of the year are plotted on the chart, which is divided in different design strategies. The percentage of hours that the original TMY data falls into each design strategy area provides a preliminary idea of the most effective HVAC strategies. With this background Climate Consultant produces a list of design guidelines for any particular climate dataset.

An additional feature of Climate Consultant is the conversion of EPW files into tabulated TMY files via the function "Export Weather Data" in the File menu.

Climate Consultant is available for the Windows and Mac operating systems and needs to be installed as program. Download at

<http://www.energy-design-tools.aud.ucla.edu/climate-consultant/request-climate-consultant.php>

### Online\_Resource 2-7: DView utility to display tmy, epw data

DView is an utility developed by NREL to visualize TMY data in advanced, meaningful forms. At the moment it is available only for the Windows operating system:

<https://beopt.nrel.gov/downloadDView>

To a large extent, the functionality of DView is the same as that included in the SAM program (Main Reference Sites, Software, p.9).

### 3 PV Technology

#### 3.1 Solar modules

##### 3.1.1 Basic technical parameters

**Flat-plate photovoltaic (PV)** modules, also called **solar modules**, are the smallest field-installable solar generating unit. They generate dc power in rough proportion to the total incident solar radiation (direct and diffuse) at an efficiency that depends strongly on the operating temperature.

Solar cells and solar modules under illumination show a characteristic relation between output voltage and current. The curve can be described by a few basic parameters: the short-circuit current  $I_{SC}$ , the open-circuit voltage  $V_{OC}$ , the peak power  $P_{max}$  and the fill factor  $FF$ . The conversion efficiency  $\eta$  can be derived from these parameters.

The **short-circuit current**  $I_{SC}$  is the current that – at least in theory – flows through the external circuit of the module when this is short circuited.  $I_{SC}$  represents the upper limit to the current generated by the solar module. It is proportional to the irradiation on the solar cell, the spectrum of the incident light, and the area of the cell; for cells of 100-150 cm<sup>2</sup> under nominal irradiation 1000 W·m<sup>-2</sup>  $I_{SC}$  reaches some Amps

The **open-circuit voltage**  $V_{OC}$  is the voltage at which no current flows through the external circuit, i.e., the cell electrodes are disconnected. This is the maximum voltage that a solar module can deliver.  $V_{OC}$  for commercial solar cells typically lies in the range 0.4–0.6 V.

The **maximum power point** is the particular pair of output voltage  $V_{mp}$  and corresponding current  $I_{mp}$  at which the solar module delivers its maximum power output, indicated as **peak power** ( $P_{max}$ ). The external circuitry connected to the module must be designed so that this can operate at or near the maximum power point under different conditions.

The **fill factor** is the ratio between the maximum power generated by a solar cell and the product of  $V_{OC}$  with  $I_{SC}$ ; the theoretical maximum power output

$$FF = \frac{I_{mp} \cdot V_{mp}}{I_{SC} \cdot V_{OC}} \quad (3-1)$$

Typical values for the fill factor are in the range 0.75–0.80.

The **conversion efficiency** is calculated as the ratio between the maximum generated power and the incident power under STC conditions (→ 3.1.2):

$$\eta = \frac{P_{max}}{P_{in}} = \frac{J_{mp} \cdot V_{mp}}{P_{in}} = \frac{J_{SC} \cdot V_{OC} \cdot FF}{P_{in}} \quad (3-2)$$

The conversion efficiency lies typically in the range of 12 to 20%.

### 3.1.2 Solar module testing

Solar modules are tested under Standard Testing Conditions (STC). These are an **irradiance** value  $P_{in} = 1000 \text{ W} \cdot \text{m}^{-2}$  with a direct radiation component  $= 835 \text{ W} \cdot \text{m}^{-2}$ , and the absorption profile for an **air mass**  $AM=1.5$  as defined in the ASTM G173-03 standard spectrum (→ Section 1.1.1 and Online\_Resource 1-1), and cell temperature  $= 25^\circ\text{C}$ .

A solar irradiance of  $1000 \text{ W} \cdot \text{m}^{-2}$  is known in the PV community as one **equivalent sun**.

AM1.5 is the solar radiation spectrum after crossing an air amount equal 1.5 times the vertical layer AM1. The corresponding elevation angle is  $\alpha = \arcsin(1/1.5) = 41.83^\circ$ .

Under optimal weather conditions and at high solar elevation angles the global irradiation can reach values up to  $1000 \text{ W} \cdot \text{m}^{-2}$ . In case of high cloud reflectance even values of up to  $1200 \text{ W} \cdot \text{m}^{-2}$  can be reached for a short time at some locations, though such conditions are rare. In most cases solar modules will not operate under the very strong irradiance indicated by the STC but under overcast conditions with a large share of diffuse radiation. It is therefore important to consider not only the module peak power, but also that at lower irradiation values.

For irradiance values below STC the power output of a module decreases proportionally, though at low input energy levels the efficiency also becomes lower than nominal. For example, at  $200 \text{ W} \cdot \text{m}^{-2}$  the efficiency may be about 95% of the STC efficiency. Though still uncommon, some data sheets now report the reduction in efficiency under partial load conditions.

Besides the irradiation the solar module output is dependent on the temperature. Under STC the module temperature is assumed to be the same as the ambient temperature. Under real operating conditions the solar cells in the module heat up, thus lowering their performance and, consequently, the overall module efficiency. The two temperature coefficients TC ( $I_{sc}$ ) and TC ( $V_{oc}$ ) indicate how much the short-circuit current and the open-circuit voltage change for each degree deviation from the STC 25°C reference. Typical values are 0.05%/K for TC ( $I_{sc}$ ) and -0.4%/K for TC ( $V_{oc}$ ). These values can be combined in a single output power change %factor. Increasing temperatures have influence over the output voltage much more than the current, so that the temperature coefficient for the power is also negative, that is, for increasing temperatures the power output decreases.

STC testing procedures were designed in first place for quality control in the production of solar modules, and not for operative considerations. When solar modules began to be widely used for power generation it turned out that STC laboratory tests did not deliver predictable results for field operations. For this reason in the late 1980s engineers of the Pacific Gas & Electric Company developed the PVUSA Test Conditions (PTC) to address real operating conditions.<sup>19</sup> The PVUSA testing requirements are closer to the expected actual operating conditions in a majority of locations: solar irradiance 1000 W·m<sup>-2</sup>, 20°C ambient (not cell) temperature, and wind speed of 1 meter per second at 10 meters above ground level, which provides for some cooling of the modules. In general, the PTC rating gives lower results than the STC rating and is now generally recognized as a more realistic measure of PV output.<sup>20</sup>

Solar cells and modules perform worse under temperatures higher than at the standard testing conditions, but also heat up during operation, thus lowering their performance. The degree of self-heating for a solar module is indicated by the **nominal operating cell temperature (NOCT)**.

---

19 See, for example, <http://solarprofessional.com/articles/products-equipment/modules/rating-pv-modules-for-field-performance#.WS8p5DexV5d>

20 These aspects are treated in the Sandia Lab Report "Photovoltaic Array Performance Model" (2004), <http://prod.sandia.gov/techlib/access-control.cgi/2004/043535.pdf>

NOCT is defined as the temperature reached by the cells in a solar module under the PVUSA reference conditions and is approximately proportional to the irradiance according to the relation

$$T_{\text{cell}} = 20^{\circ}\text{C} + (T_{\text{NOCT}} - 20^{\circ}\text{C}) \cdot \frac{E}{800 \text{ W}\cdot\text{m}^{-2}} \quad (3-3)$$

Typical NOCT values are in the order of 45–50°C, that is, under an irradiance 800 W·m<sup>-2</sup> the cells operate at temperatures ca. 25–30°C above the 20°C reference temperature. As a consequence, at mild-warm ambient temperatures and for strong irradiation values the overall efficiency of solar modules can be 15-20% below nominal.

The calculation of the operating cell temperature is a bit tricky because it refers to two different reference temperature and irradiation values, those for the STC power output conditions and for the NOCT evaluation, as the following example will show.

Let us consider the module type 'A10Green\_Technology\_A10J\_S72\_185', whose parameters are presented in Section → 3.2.3. The NOCT temperature is 49.9°C, nominal output power under STC conditions  $P_{\text{max}}$  is 184.7 W (Equation 3-4, p.152) and the temperature-dependent output power coefficient TC is = -0.936 W/K (Equation 3-7, p.152). We want to calculate the power output under ambient conditions irradiance = 850 W·m<sup>-2</sup> and temperature = 27°C. Two different relations need to be used, first to estimate the operating cell temperature from the NOCT parameters and the irradiance, then to estimate the power output from the irradiance and the cell temperature.

The cell temperature according to the NOCT relation is

$$T_{\text{cell}} = 27^{\circ}\text{C} + (49.9^{\circ}\text{C} - 20^{\circ}\text{C}) \cdot \frac{850}{800} = 58.8^{\circ}\text{C}$$

The power change is calculated with respect to the STC temperature reference = 25°C:

$$P = P_{\text{max}} \cdot (1 + \text{TC}_{\text{mpp}} \cdot (T_{\text{cell}} - 25^{\circ}\text{C}))$$

$\text{TC}_{\text{mpp}}$  is negative, so the power decreases for increasing temperatures. By substituting the values in the equation we obtain

$$P = 184.7 \cdot (1 - 0.936 \cdot (58.8^{\circ}\text{C} - 25^{\circ}\text{C})) = 126.3 \text{ W}$$

this value is much lower than the nominal STC power  $P_{\text{max}} = 184.7 \text{ W}$ .

The relations presented here will be used to calculate the power output of solar modules under ambient conditions from TMY data in the examples of Program\_Code 4-1 and Program\_Code 4-3.

## 3.2 Solar component databases

### 3.2.1 Access to datasets in csv format via pandas dataframes

`pandas` is a powerful software library for data processing and analysis written for Python. Pandas methods are used in this primer to easily access structured data, such as TMY data. In particular, pandas methods can be used to read `.csv` or spreadsheet data into pandas dataframes or from these to save as `.csv`, as well as for other functions, such as finding the maximum of the values contained in a column.

The pandas website is <http://pandas.pydata.org/>, where complete information, documentation, and examples can be found. Due to its importance in processing large amounts of scientific data, pandas applications are described and discussed at length in several websites.

In the following will be presented some examples about reading and saving structured data and about dataframe processing.

Many datasets, such as TMY data, are stored as `.csv`. The US research center NREL and other institutions have produced datasets in `.csv` format about the most important solar components, these datasets are accessible online (→ Section 3.2.2). An example of a few lines of the solar module `.csv` dataset as it appears in a text editor is the following

```
Name,BIPV,Date,T_NOCT,A_c,N_s,I_sc_ref,V_oc_ref,I_mp_ref,V_mp_ref,alpha_sc,
beta_oc,a_ref,I_L_ref,I_o_ref,R_s,R_sh_ref,Adjust,gamma_r,Version,PTC,Techn
ology
Units,,,C,m2,,A,V,A,V,A/K,V/K,V,A,A,Ohm,Ohm,%,%/K,,,
[0],,,cec_t_noct,cec_area,cec_n_s,cec_i_sc_ref,cec_v_oc_ref,cec_i_mp_ref,ce
c_v_mp_ref,cec_alpha_sc,cec_beta_oc,cec_a_ref,cec_i_l_ref,cec_i_o_ref,cec_r
_s,cec_r_sh_ref,cec_adjust,cec_gamma_r,,,cec_material
1Soltech 1STH-215-P,N,10/7/2010,47.4,1.567,60,7.84,36.3,7.35,29,0.007997,-
0.13104,1.6413,7.843,1.936E-09,0.359,839.4,16.5,-0.495,MM107,189.4,Multi-c-
Si
1Soltech 1STH-220-
P,N,10/4/2010,47.4,1.567,60,7.97,36.6,7.47,29.3,0.008129,-
0.13213,1.6572,7.974,2.03E-09,0.346,751.03,16.8,-0.495,MM107,194,Multi-c-Si
```

```

1Soltech 1STH-225-
P,N,10/4/2010,47.4,1.567,60,8.09,36.9,7.58,29.6,0.008252,-
0.13321,1.6732,8.094,2.126E-09,0.334,670.65,17.1,-0.495,MM107,198.5,Multi-
c-Si
1Soltech 1STH-230-
P,N,10/4/2010,47.4,1.567,60,8.18,37.1,7.65,29.9,0.008344,-
0.13393,1.6888,8.185,2.332E-09,0.311,462.56,17.9,-0.495,MM107,203.1,Multi-
c-Si
1Soltech 1STH-235-WH,N,3/4/2010,49.9,1.635,60,8.54,37,8.02,29.3,0.00743,-
0.13653,1.6292,8.543,1.166E-09,0.383,1257.84,8.7,-0.482,MM107,205.1,Mono-c-
Si

```

In a spreadsheet the contents become easier to read. This is a selection of the first rows and columns

Name	BIPV	Date	T_NOCT	A_c	N_s	I_sc_ref	V_oc_ref
Units			C	m2		A	V
[0]			cec_t_noct	cec_area	cec_n_s	cec_i_sc_ref	cec_v_oc_ref
1Soltech 1STH-215-P	N	10/7/2010	47.4	1.567	60	7.84	36.3
1Soltech 1STH-220-P	N	10/4/2010	47.4	1.567	60	7.97	36.6
1Soltech 1STH-225-P	N	10/4/2010	47.4	1.567	60	8.09	36.9
1Soltech 1STH-230-P	N	10/4/2010	47.4	1.567	60	8.18	37.1
1Soltech 1STH-235-WH	N	3/4/2010	49.9	1.635	60	8.54	37
1Soltech 1STH-240-P	N	4/2/2014	43	1.561	60	8.62	36.9

A pandas dataframe can be produced by a dataset of PV components (PV modules, inverters) and loaded in the Python shell terminal (for details see → Program\_Code 3-1, Program\_Code 3-2). A loaded pandas dataframe is here named **df** and can be called directly. The following examples shall illustrate the basic commands.

The shell input by dataframe name, that is

```
>>> df
```

returns the list with the component names (which serve as index) per row and attributes per column, for example:

```

          BIPV      Date  T_NOCT  A_c  N_s  \
1Soltech_1STH_215_P      N  10/7/2010   47.4  1.567  60
1Soltech_1STH_220_P      N  10/4/2010   47.4  1.567  60
1Soltech_1STH_225_P      N  10/4/2010   47.4  1.567  60
1Soltech_1STH_230_P      N  10/4/2010   47.4  1.567  60
1Soltech_1STH_235_WH      N   3/4/2010   49.9  1.635  60

```

The listing is limited to 30 rows and the columns are shown only within the window width (a method to override this limit is described in the code of [csvdata\\_component\\_list](#) (Program\_Code 3-3)).

The last line gives the size of the dataframe

```
[18102 rows x 21 columns]
>>>
>>>
```

The module names are used as indexes, this is shown by the dot attribute `index`:

```
>>> df.index
Index(['1Soltech_1STH_215_P', '1Soltech_1STH_220_P', '1Soltech_1STH_225_P',
      '1Soltech_1STH_230_P', '1Soltech_1STH_235_WH',
      '1Soltech_1STH_240_P',
      '1Soltech_1STH_240_WH', '1Soltech_1STH_245_P',
      '1Soltech_1STH_245_WH',
      '1Soltech_1STH_250_P',
      ...,
      'iTek_iT_240', 'iTek_iT_260_HE', 'iTek_iT_265_HE', 'iTek_iT_270_HE',
      'iTek_iT_275_HE', 'iTek_iT_280_HE', 'iTek_iT_285_HE',
      'iTek_iT_290_HE',
      'iTek_iT_295_HE', 'iTek_iT_300_HE'],
      dtype='object', length=18102)
>>>
```

The types for the data content are accessed with the attribute `dtypes`:

```
>>> df.dtypes
BIPV          object
Date          object
T_NOCT       float64
A_c          float64
N_s          int64
I_sc_ref     float64
V_oc_ref     float64
I_mp_ref     float64
V_mp_ref     float64
alpha_sc     float64
beta_oc      float64
a_ref        float64
I_L_ref      float64
I_o_ref      float64
R_s          float64
R_sh_ref     float64
Adjust       float64
gamma_r      float64
Version      object
PTC          float64
Technology   object
dtype: object
>>>
```



It is also possible to call `dtypes.index`:

```
>>> df.dtypes.index
Index(['BIPV', 'Date', 'T_NOCT', 'A_c', 'N_s', 'I_sc_ref', 'V_oc_ref',
      'I_mp_ref', 'V_mp_ref', 'alpha_sc', 'beta_oc', 'a_ref', 'I_L_ref',
      'I_o_ref', 'R_s', 'R_sh_ref', 'Adjust', 'gamma_r', 'Version', 'PTC',
      'Technology'],
      dtype='object')
>>>
```

The full dataset under a particular index is accessed with the `.loc` method:

```
>>> df.loc['iTek_iT_280_HE']
BIPV      N
Date      2/2/2015
T_NOCT    44.5
A_c       1.634
N_s       60
I_sc_ref   9.3
V_oc_ref  39.2
I_mp_ref   8.6
V_mp_ref  32.3
alpha_sc   0.00372
beta_oc   -0.10976
a_ref      1.5503
I_L_ref    9.32
I_o_ref    9.39e-11
R_s        0.254
R_sh_ref   118.83
Adjust     17.15
gamma_r    -0.4
Version    NRELv1
PTC        257.8
Technology Mono-c-Si
Name: iTek_iT_280_HE, dtype: object
>>>
```

Pandas makes use of an implicit numerical index that is accessed with `.iloc` and the starting index is =0. For example, `index=1` returns the second item in the full list:

```
>>> df.iloc[1]
BIPV      N
Date      10/4/2010
T_NOCT    47.4
A_c       1.567
N_s       60
I_sc_ref   7.97
V_oc_ref  36.6
I_mp_ref   7.47
```

```

V_mp_ref          29.3
alpha_sc          0.008129
beta_oc           -0.13213
a_ref             1.6572
I_L_ref           7.974
I_o_ref           2.03e-09
R_s               0.346
R_sh_ref          751.03
Adjust            16.8
gamma_r           -0.495
Version           MM107
PTC               194
Technology        Multi-c-Si
Name: 1Soltech_1STH_220_P, dtype: object
>>>

```

The contents of a full column are called with the following notation

```

>>> df.T_NOCT
1Soltech_1STH_215_P          47.4
1Soltech_1STH_220_P          47.4
1Soltech_1STH_225_P          47.4
1Soltech_1STH_230_P          47.4
[. . .]
>>>

```

This is the same as using the **getattr()** function:

```

>>> getattr(df, 'PTC')
1Soltech_1STH_215_P          189.4
1Soltech_1STH_220_P          194.0
1Soltech_1STH_225_P          198.5
1Soltech_1STH_230_P          203.1
1Soltech_1STH_235_WH          205.1
[. . .]
>>>

```

A single parameter in a module is accessed with a combination of index and attribute

```

>>> df.loc['iTek iT_280_HE'].PTC
257.80000000000001
>>>

```

or the equivalent formulation

```

>>> df.loc['iTek iT_280_HE']['PTC']
257.80000000000001
>>>

```

The item can be accessed also by selecting the column first, followed by the index. The index can either be passed as string in square brackets or indicated as `.dot`. All the following notations are therefore equivalent:

```
>>> df.PTC['ecoSolargy_EC0355T156M4_72']
323.69999999999999
>>> df.PTC.ecoSolargy_EC0355T156M4_72
323.69999999999999
>>> df['PTC']['ecoSolargy_EC0355T156M4_72']
323.69999999999999
>>> df['PTC'].ecoSolargy_EC0355T156M4_72
323.69999999999999
>>>
```

An index beginning with a figure is not accepted under the `.dot` notation. Trying to access the first module of the list with the following instruction returns an error

```
>>> df.PTC.1Soltech_1STH_215_P
SyntaxError: invalid syntax
>>>
```

but the following notation works correctly

```
>>> df.PTC['1Soltech_1STH_215_P']
189.40000000000001
>>>
```

Pandas work well for TMY data, where single columns contain homogeneous data for a location. In a TMY dataset after loading into a pandas dataframe a full column is simply accessed with (in `.dot` format):

```
>>> tmy_data.GHI
```

or also (in index format):

```
>>> tmy_data['GHI']
```

The names of the columns in a pandas dataframe can be displayed from the Python shell window with the command

```
>>> tmy_data.columns
```

The following scripts shall better illustrate the use of pandas dataframes to read and process datasets from `.csv` formats.

### Program\_Code 3-1: function, read dataset in CSV format, return pandas dataframe

<b>package</b>	<b><code>solprim.readcsvfile</code></b>
<b>function</b>	<b><code>readcsvfile</code></b>
<b>description</b>	read a datafile in <code>.csv</code> format, return data as pandas dataframe
<b>parameters</b>	<b>filename</b> : <code>.csv</code> dataset file name including directory path, string
<b>returns</b>	pandas dataframe

The function `readcsvfile` reads a datafile in `.csv` format and returns a pandas dataframe. The function only input is a directory path with a filename.

In the generated pandas dataframe the names of the components, solar modules or inverters, are used as index. In order to avoid inconsistencies for the index and column names, `readcsvfile` substitutes all characters that are not alphanumeric, such as blanks, dots, slashes, brackets, and others with underscores “\_”.

The function `readcsvfile` operates only on the component datasets from SAM or similar libraries (→ Section 3.2.2). It takes `row=0` as header, drops `rows=1, 2` and loads the data starting from `row=3`. Without changes `readcsvfile` does not work on, e.g., TMY files because these have a different header structure. The necessary modifications to read other `.csv` files, however, are quite simple once the principle is understood and the file structure is known.

### Program\_Code 3-2: script, read dataset in CSV format, return pandas dataframe

<b>script</b>	<a href="#">csvdata_pandas_read</a>
<b>description</b>	read a datafile in .csv format, return data as pandas dataframe, print data on default output device
<b>parameters</b>	<b>filename</b> : .csv dataset file name including directory path, string
<b>returns</b>	pandas dataframe

This script is a transparent interface to the function [readcsvfile](#). From a directory and name path the script reads the specified file and returns a pandas dataframe with correct index and column names.

### 3.2.2 PV components datasets

The US research center NREL manages and publishes comprehensive libraries for PV solar modules, inverters, and solar thermal modules. These databases are available as part of the SAM software package (Main Reference Sites, Software, p.9) under the “library” directory. The following datasets are available:

<code>CEC_Modules.csv</code>	PV solar module database by the California Energy Commission. It includes 18,102 components with 21 technical parameters.
<code>Sandia_Modules.csv</code>	PV solar module database by the Sandia National Laboratories. It includes 523 components with 42 technical parameters.
<code>CEC_Inverters.csv</code>	inverter database published by the California Energy Commission (CEC). It includes 3772 devices with 14 technical parameters.

The data in the .csv datasets after conversion to a pandas dataframe must be accessed either with an index or by full columns. In a spreadsheet the component names will appear in the first column.

### Program\_Code 3-3: script, read component database as CSV dataset, return index list

<b>script</b>	<b>csvdata_component_list</b>
<b>description</b>	read a datafile in <code>.csv</code> format, return component list on default output device and as file
<b>parameters</b>	<b>filename</b> : <code>.csv</code> dataset file name including directory path, string
<b>returns</b>	list (index) of the items contained in the datafile that can be used as index in a <code>.txt</code> file

The script `csvdata_component_list` reads a component database and produces a text file listing all components names, which can be later be used as indexes to single components.

The name of the produced index file is the same as the input file, with the addition `_index` and extension `.txt`. This file can be easily read with a simple text editor.

### Online\_Resource 3-1: Repositories for PV module and inverter data

Indications about databases and libraries for solar energy components, including `CEC_Inverters`, `CEC_Modules`, and `Sandia_Modules`, are found at <https://sam.nrel.gov/libraries>

Other libraries are found at

[http://www.gosolarcalifornia.ca.gov/equipment/pv\\_modules.php](http://www.gosolarcalifornia.ca.gov/equipment/pv_modules.php)

The conversion from spreadsheet format into `.csv` is quite simple, after opening a dataset it is sufficient to save it again as `.csv` file.

### 3.2.3 CEC PV module database

The SAM system (→ Main Reference Sites, Software, p.9) contains a large database for PV modules and inverters. The datasets are in .csv format and can be easily accessed.

Actually, the datasets for PV modules available in the SAM system are two, one with more than 18,000 items and 21 parameters, another by Sandia labs with 523 items and a 42-items parameter set.

The CEC\_Modules database contains the following parameters, shown on the example of the module type 'A10Green\_Technology\_A10J\_S72\_185'.<sup>21</sup>

parameter	value	explanation
BIPV	N	building-integrated PV (prevents effective cooling)
Date	6/9/2010	Testing/ reporting date
T_NOCT	49.9	Nominal operating cell temperature
A_c	1.3	Module area in m2
N_s	72	Number of cells in series
I_sc_ref	5.43	Short-circuit current
V_oc_ref	44.14	Open-circuit voltage
I_mp_ref	5.03	Current at maximum power point
V_mp_ref	36.72	Voltage at maximum power point
alpha_sc	0.002253	temperature coeff. for short-circuit current
beta_oc	-0.15961	temperature coeff. for open-circuit voltage
a_ref	1.986	Modified ideality factor at reference conditions
I_L_ref	5.436	Photocurrent at reference conditions
I_o_ref	1.177e-09	Reverse saturation current at reference conditions
R_s	0.311	Series resistance (constant)
R_sh_ref	298.8	Shunt resistance at reference conditions
Adjust	15.8	
gamma_r	-0.507	temperature coeff. for output power
Version	MM107	
PTC	160.2	nominal power according to the PVUSA testing conditions
Technology	Mono-c-Si	Mono-cristalline silicium
Name:	A10Green_Technology_A10J_S72_185	

<sup>21</sup> These parameters are described at detail in the Sandia Labs report "Photovoltaic Array Performance Model" (2004),  
<http://prod.sandia.gov/techlib/access-control.cgi/2004/043535.pdf>

The most significant parameters for solar modules can be either read directly or easily calculated from this data.

The maximum (nominal) power  $P_{\max}$  is the product of the voltage and current at the maximum power point:

$$P_{\max} = I_{\text{mp\_ref}} \cdot V_{\text{mp\_ref}} \quad (3-4)$$

For the module,  $P_{\max}$  is then

$$P_{\max} = 5.03 \cdot 36.72 = 184.702 \text{ W}$$

The nominal power under STC conditions is 184 W and is higher than the PTC = 160 W (→ Section 3.1.2). Under real operating conditions, considering that the cell temperature in most cases will be higher than that under reference conditions, the power output will be even lower.

The nominal efficiency is equal  $P_{\max}$  divided by the module area and scaled to percent units

$$\eta = \frac{P_{\max}}{A_c} \cdot 100 \quad (3-5)$$

For a module area = 1.3 m<sup>2</sup> the nominal efficiency becomes 14.2078%.

The fill factor can also be calculated from the reported parameters:

$$FF = \frac{I_{\text{mp\_ref}} \cdot V_{\text{mp\_ref}}}{I_{\text{sc\_ref}} \cdot V_{\text{oc\_ref}}} = \frac{5.03 \cdot 36.72}{5.43 \cdot 44.14} = 0.7597 \quad (3-6)$$

In practical operation the temperature coefficient related to the NOCT temperature (→ Section 3.1.2) is particularly important. The power temperature coefficient from the table is  $\gamma_r$  and represents the percentage power difference for temperatures different than STC. The percentage value can be transformed in an absolute power indication by multiplying it with the peak power:

$$-0.507\%/K \cdot 184.702 \text{ W} = -0.936 \text{ W/K} \quad (3-7)$$



### Program\_Code 3-4: script, read CEC PV module dataset as CSV, return module data

<b>script</b>	<a href="#"><code>csvdata_CEC_module_data</code></a>
<b>description</b>	read a datafile with the CEC module database in <code>.csv</code> format, return all parameters for a component identified by a name/ index
<b>parameters</b>	<b>filename</b> : <code>.csv</code> dataset file name including directory path, string the database has name <code>CEC_Modules.csv</code> in the SAM directory <b>module</b> : module name, string
<b>returns</b>	list for all parameters associated with the selected module

The `CEC_module` database contains data for 18,102 PV solar module types. The code of the [`csvdata\_CEC\_module\_data`](#) script consists only in few statements to read the dataset, select a module, print the data. The table has the aspect shown at the beginning of this Section and in the examples of → Section 3.2.1.

### Program\_Code 3-5: script, read CEC PV module dataset, add columns, sort data

<b>script</b>	<a href="#"><code>csvdata_CEC_module_sort</code></a>
<b>description</b>	read the <code>CEC_Module</code> dataset in <code>.csv</code> format, add columns for power, efficiency. Sort data for efficiency and save result.
<b>parameters</b>	<b>filename</b> : <code>.csv</code> dataset file name including directory path, string the database has name <code>CEC_Modules.csv</code> in the SAM directory
<b>returns</b>	file with the modules from the original dataset sorted by selected criterion, e.g., for power or efficiency

With several thousands data records it may be difficult to find a component of interest in the `CEC_module` database. This script provides a data processing example on `pandas` data structures that can solve the problem. In [`csvdata\_CEC\_module\_sort`](#) are first defined two new columns for 'power' and 'efficiency', which are filled with data via relations on the other parameters as described earlier in this Section. Further, the data

is sorted by decreasing efficiency, so that the most efficient modules are listed first. At the end the data is stored in a new `.csv` file that can be opened either with a spreadsheet or a text editor. An output example is

	eta	P_max	PTC	Technology
SunPower_SPR_X22_360	22.07	359.96	334.4	Mono-c-Si
SunPower_SPR_X22_360_COM	22.07	359.96	334.4	Mono-c-Si
SunPower_SPR_X22_475_COM	22.04	476.5	442.8	Mono-c-Si
SunPower_SPR_X22_359	22.03	359.34	333.4	Mono-c-Si
Integrated_Solar_Technology_STS_105M_B4U	22.01	105	91.8	Mono-c-Si
SunPower_SPR_X22_470_COM	21.73	469.74	438	Mono-c-Si
SunPower_SPR_X21_255	21.54	254.66	241.7	Mono-c-Si

### 3.2.4 Sandia PV module database

The US Sandia labs have prepared a PV module dataset that is available under the SAM package. The `sandia_Modules.csv` database lists 523 PV solar module types.

For each module are stored 42 parameters.<sup>22</sup> Parameter naming is inconsistent with the CEC database, so that a direct merge of the two datasets without additional processing is not possible.

Vintage	2006 (E)	
Area	1.018	Module area in m2
Material	mc-Si	
Cells_in_Series	36	Number of cells in series
Parallel_Strings	1	
Isco	8.2	Short-circuit current
Voco	22	Open-circuit voltage
Impo	7.5	Current at maximum power point
Vmpo	17.4	Voltage at maximum power point
Aisc	0.00065	
Aimp	-0.0002	
C0	0.991	
C1	0.009	
Bvoco	-0.08	
Mbvoc	0	
Bvmpo	-0.082	
Mbvmp	0	
N	1.32	
C2	0.12669	
C3	-6.05171	

<sup>22</sup> Sandia National Laboratory, Photovoltaic Array Performance Model (2004):  
<http://prod.sandia.gov/techlib/access-control.cgi/2004/043535.pdf>

```

A0          0.9415
A1          0.052728
A2         -0.0095876
A3          0.00067629
A4         -1.8111e-05
B0          1
B1         -0.002438
B2          0.0003103
B3         -1.246e-05
B4          2.11e-07
B5         -1.36e-09
dT          3
FD          1
a          -3.537
b          -0.0721
C4          0.9866
C5          0.0134
IXO         8.1
IXXO        5.25
C6          1.1183
C7         -0.1183
Notes       Source: Sandia National Laboratories Updated 9...
Name:       BP_Solar_BP3125__2006__E__, dtype: object

```

#### Program\_Code 3-6: script, read Sandia PV module dataset as CSV, return module data

<b>script</b>	<a href="#"><code>csvdata_Sandia_module_data</code></a>
<b>description</b>	read a datafile with the Sandia module database in <code>.csv</code> format, return all parameters for a component identified by a name/ index
<b>parameters</b>	<b>filename</b> : <code>.csv</code> dataset file name including directory path, string the database has name <code>Sandia_Modules.csv</code> in the SAM directory <b>module</b> : module name, string
<b>returns</b>	list for all parameters associated with the selected module

The Python script [`csvdata\_Sandia\_module\_data`](#) reads the Sandia module database and returns the data for a module identified by its index.

### 3.2.5 CEC Inverter dataset

The SAM software makes use of load-dependent efficiency models for the power inverters that convert dc from solar modules into ac either to supply ac power loads or for direct grid in-feed. These inverter models are described by different coefficients.

The following script reads and displays inverter parameters from the SAM dataset.

Program\_Code 3-7: script, read CEC inverter dataset as CSV, return inverter data

<b>script</b>	<code>csvdata_CEC_inverter_data</code>
<b>description</b>	read a datafile with the SAM inverter database in <code>.csv</code> format, return all parameters for one component identified by a name/ index
<b>parameters</b>	<b>filename</b> : <code>.csv</code> dataset file name including directory path, string the database has name <code>CEC_inverter.csv</code> in the SAM directory <b>inverter</b> : inverter name, string
<b>returns</b>	list for all parameters associated with the selected inverter

The script `csvdata_CEC_inverter_data` is similar to those of Program\_Code 3-4 and Program\_Code 3-6. It reads a `.csv` dataset into a `pandas dataframe`, then with the name of the inverter as index it identifies and displays all related parameters.

## 4 PV system operation

### 4.1 PV solar generator without storage

Large-scale PV solar generators without storage are widespread in those locations where the direct power in-feed in public grids is allowed and financially remunerated.

The following script simulates the full generation chain for a solar array based on solar panel type and array size.

Program\_Code 4-1: script, PV solar generator simulation without storage

<b>script</b>	<b>power_pvgen_direct</b>
<b>description</b>	From basic parameters for PV modules and size, azimuth and elevation of a solar array calculate the generated solar power on a hourly basis in the course of the year
<b>parameters</b>	<b>filename</b> : TMY (.csv) dataset file name with directory path, string <b>csv_filename</b> : PV module (.csv) dataset file name including directory path, string <b>module</b> : PV module name, string <b>parameters</b> for PV module, array, and system as described in the script source code
<b>returns</b>	<ul style="list-style-type: none"> <li>- .csv file with 8760 hourly values including Tdry,GHI, DNI, DHI, POA, Tcell, generated power</li> <li>- barchart for the POA totals per month [<math>\text{W}\cdot\text{m}^{-2}</math>]</li> <li>- barchart for generated power totals per month [<math>\text{W}\cdot\text{m}^{-2}</math>]</li> </ul>

The **power\_pvgen\_direct** script includes all necessary steps for the calculation of the PV generator production for a PV module array and TMY data. All detail information is contained in the source code. The inverters and ancillary equipment are described by a lumped parameter for efficiency losses. The most important solar module parameters are defined in the input (alternatively these can be read from a component file, see → Section 3.2.3, 3.2.4), the climate is described by a TMY file for the location. The output is a .csv file with hourly simulation values for the power generation, a barchart with monthly Plane-of\_Array values in [ $\text{kWh}/\text{m}^2\cdot\text{day}$ ] and a barchart for the total generated power per month [ $\text{MWh}$ ], as shown in Figure 4-1 and Figure 4-2.

An example of display output for the `power_pvgen_direct` script is the following:

```
{'Source': 'TMY3_new(TYA)', 'Location ID': '744860', 'Time Zone': -5.0,
'City': '"NEW YORK J F KENNEDY INT\'L AR"', 'Elevation': 5, 'Longitude':
-73.8, 'Latitude': 40.65, 'State': 'NY', 'Country': 'USA'}
```

```
===== system data
```

```
array azimuth=180  array tilt=38
```

```
PV module type = SunPower_SPR_X21_335_BLK
```

```
P_STC (W) = 335
```

```
PTC (W) = 313.7
```

```
NOCT =46.4
```

```
pvmmod_Tcoeff =-1.0385
```

```
no of modules = 12
```

```
installed power/ nameplate dc rating (W) =4020
```

```
module area (m2) = 1.631  array total area (m2) = 19.6
```

```
===== location-specific TMY insolation data
```

```
GHI year total per m2 =1430.4 kWh/m2      per day =3.92 kWh/m2
```

```
DNI year total per m2 =1347.5 kWh/m2      per day =3.69 kWh/m2
```

```
DHI year total per m2 =645.5 kWh/m2      per day =1.77 kWh/m2
```

```
POA year total per m2 =1538.5 kWh/m2      per day =4.22 kWh/m2
```

```
===== main simulation results
```

```
Total yearly generation (MWh) = 5.37
```

```
Total yearly POA insolation (MWh) = 30.15
```

```
PV generation coeff. kWh/kWp =1335.57
```

```
PV generation / total POA insol =0.18
```

```
file saved: data-output/Pvgen_simulation_001.csv
```

```
>>>
```

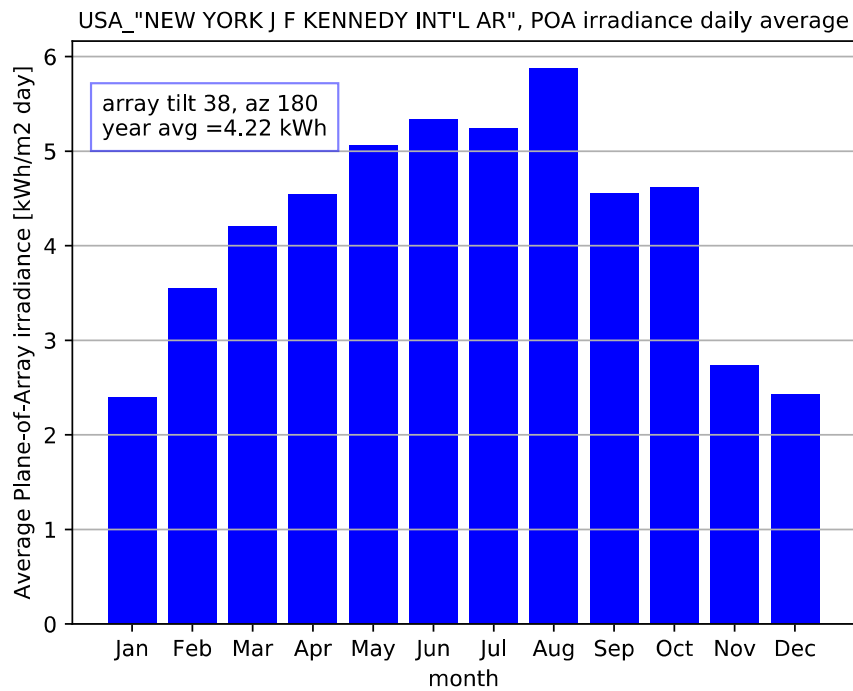


Figure 4-1: Day average of the monthly Plane-of-Array (POA) irradiance per  $m^2$ . The result is calculated from TMY data and the plane orientation.

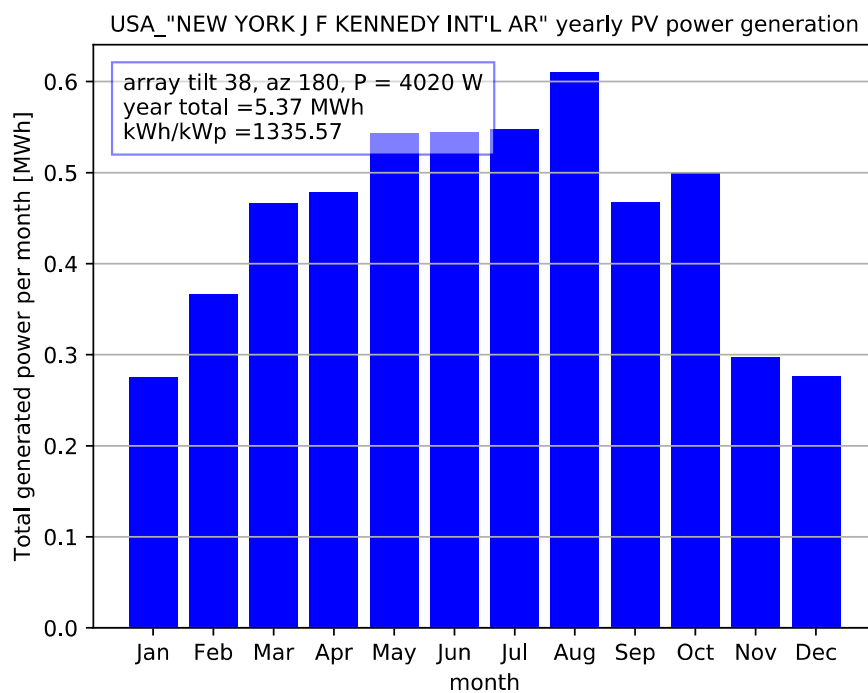


Figure 4-2: The same system of Figure 2-30, total array power generation per month in MWh

## Result comparison with solar calculation software

In order to check the quality of the script of Program\_Code 4-1 its operation has been tested on five different locations worldwide with different climatic conditions and access to the solar resource and compared with four established solutions: SAM (Main Reference Sites, Software, p.9), the ESMAP Global Solar Atlas (Online\_Resource 4-1), the EU PVGIS Solar Database (Online\_Resource 4-2), and the NREL PVWatts calculator (Online\_Resource 4-3).

The tested system is fairly simple, a 4 kWp PV array for direct connection to the power grid (no storage or load simulation), and different orientation azimuth and tilt angles. The outcome of the tests is shown in Table 4-1.

The numeric data shown in black is the outcome of Program\_Code 4-1. The parameters in blue are given explicitly as output from the different resources, those in reddish color had to be inferred. For example, PVWatts does not indicate the yearly GHI at a location, but this can be inferred by calculating the POA on a surface with tilt=0°. Similar byways need to be used with other programs and parameters.

A real challenge is represented by the use of the language, as different programs call the same thing by different names, or use different conventions to describe the data. For all programs “due South” is 180°, but for PVGIS this is 0°. The plane tilt is called “inclination” in ESMAP and “slope” in PVGIS. The “Plane-of\_Array” irradiation is called “Annual Solar Radiation” in PVWatts, “Global Tilted Irradiation (GTI)” in ESMAP and “Yearly in-plane irradiation” in PVGIS.

Another aspect is more annoying. Different programs show different specific performance parameters. The only outcome that all agree on is the overall yearly generation of a PV array in kWh or MWh. Otherwise a simple parameter such as the POA insolation can be indicated for the whole array size or per m<sup>2</sup>, as well as monthly total, or monthly day average, or hour average etc.

The parameters reported in the table help to get an immediate idea over the performance of a PV array. The POA compared with the GHI indicates how well the tilted panel “captures” the solar radiation. The kWh/kWp indicator shows both the quality of the equipment and the resource availability.



Installed power kW Array surface m2			4.02 19.6	Sol_primer	SAM	PVWatts	ESMAP	PVGIS	
New York			GHI unit total	kWh/m^2	1430	1430	1427	1504	1470
Lat	40.65								
Lon	-73.8		POA unit total	kWh/m^2	1538	1538	1650	1770	1700
azim	180								
tilt	38		Gener total	MWh	5.37	5.44	5.56	5.52	5.79
eff	10.00%								
			kWh/kWp		1336	1351	1383	1372	1440
			eff.electrical		0.18	0.19	0.20	0.19	0.20
Bangkok			GHI unit total	kWh/m^2	1791	1791	1792	1823	1950
Lat	13.92								
Lon	100.6		POA unit total	kWh/m^2	1811	1809	1843	1877	2010
azim	180								
tilt	15		Gener total	MWh	6.03	5.99	5.76	5.48	5.99
eff	10.00%								
			kWh/kWp		1499	1489	1432	1362	1490
			eff.electrical		0.17	0.17	0.16	0.15	0.16
Shanghai			GHI unit total	kWh/m^2	1271	1271	1420	1341	no data
Lat	31.4								
Lon	121.45		POA unit total	kWh/m^2	1239	1238	1497	1438	no data
azim	180								
tilt	30		Gener total	MWh	4.31	4.25	4.97	4.43	no data
eff	10.00%								
			kWh/kWp		1071	1056	1237	1101	no data
			eff.electrical		0.18	0.17	0.18	0.17	
Muenchen			GHI unit total	kWh/m^2	1123	1123	1121	1170	1170
Lat	48.13								
Lon	11.7		POA unit total	kWh/m^2	1163	1162	1252	1354	1330
azim	200								
tilt	40		Gener total	MWh	4.11	4.07	4.23	4.27	4.45
eff	10.00%								
			kWh/kWp		1023	1013	1053	1062	1107
			eff.electrical		0.18	0.18	0.19	0.19	0.19
Rio de Janeiro			GHI unit total	kWh/m^2	1843	1843	1847	1762	no data
Lat	-22.9								
Lon	-43.17		POA unit total	kWh/m^2	1906	1904	1964	1871	no data
azim	350								
tilt	20		Gener total	MWh	6.43	6.43	6.22	5.50	no data
eff	10.00%								
			kWh/kWp		1598	1599	1547	1367	no data
			eff.electrical		0.17	0.18	0.17	0.16	

*Table 4-1: Comparison of the 'solar primer' Python solution with established software for PV performance simulation and evaluation*

The results indicate a very good agreement of the results by the Python code with the SAM software. There is also agreement with the PVWatts results, at least when the reference TMY datasets are the same for both SAM and for the Python code (in which were used the Energy Plus datasets). In some cases PVWatts refers to different datasets, the results are predictably different.

ESMAP and PVGIS do not allow the selection of a particular PV model type and make use of average, representative parameters instead. The most notable difference in comparison to the outcome by SAM are higher overall values for the reference GHI, which lead to an higher calculated PV generation. Considering that the EnergyPlus TMY datasets used by SAM and the Python code result from real field observations, it is an open question whether EPW data is no longer representative, or to what extent the ESMAP and PVGIS simulations are realistic. Here we can only indicate this situation and take it as a fact.

On the other hand, considering that each year there are wide natural variations in the availability of the solar resource, all these programs offer a good starting point to evaluate the overall feasibility of a PV plant.

### Online\_Resource 4-1: ESMAP Global solar atlas

The Global Solar Atlas provides a summary of global solar power resources and potential. It is offered as a free service and allows users to quickly obtain data and carry out a simple electricity output calculation for any location covered by the solar resource database.

The database is queried by pointing and clicking over the map. Different solar mapping layers can be selected. Detail information is available online under Knowledge Base.

<http://globalsolaratlas.info/>

### Online\_Resource 4-2: EU PVGIS tool for PV generation simulation

EU PVGIS (Photovoltaic Geographical Information System) is a map-based data repository for solar radiation and a tool to simulate the performance of PV solar generation systems. The beta version is online at

<http://re.jrc.ec.europa.eu/PVGIS5-beta.html>

EU PVGIS simulates the operation of grid-connected PV generators at locations identified by their geographical coordinates

[http://re.jrc.ec.europa.eu/pvg\\_tools/en/tools.html#TMY](http://re.jrc.ec.europa.eu/pvg_tools/en/tools.html#TMY)

### Online\_Resource 4-3: PVWatts® online calculator for electricity PV production estimation

PVWatts® Calculator is a web application by the National Renewable Energy Laboratory (NREL) of the US department of energy. With this tool it is possible to estimate the electricity production of a simple photovoltaic system on the basis of a few input parameters.

<http://pvwatts.nrel.gov/>

## 4.2 Load analysis

The most efficient use of solar electricity is to feed power loads directly. Here lies also the most important obstacle to the use of solar energy: in most cases the load profiles do not match the availability of solar power. Some storage is necessary, and storage is inherently both inefficient and costly.

The analysis of the loads to be covered totally or in part by PV solar energy is necessary in order to estimate the required and most economically feasible storage option.

Load profiles have different shapes. The simplest loads draw a constant amount of power over time. However, the majority of loads, both commercial and in households, typically vary in time as electrical equipment is switched on or off.

Power loads typically depend on

- the day of the week (workday/ weekend/ holiday)
- the hour of the day. Commercial and industrial loads are higher during the mid of the day, household loads in the evenings.
- the outside temperature. At temperatures below comfort levels the overall demand for electric power increases. At high temperature and humidity values the demand for air conditioning, and, consequently, for electric power, rises sharply.
- the season. During winter the demand for illumination is higher than in the summer.
- the production or service demand (activity index). Industrial equipment consume more power in relation to production levels, a restaurant or an hotel in proportion to the number of guests.

These relations can be described with a overall equation (4-1).

$$L_{\text{total}} = L_{\text{base}} + L_{\text{time}}(\text{day}, \text{hour}) + L_{\text{illum}}(\text{GHI}) + L_{\text{activity}}(\text{activity index}) + k_{\text{heat}} \cdot (T_{\text{set, heat}} - T_{\text{ambient}}) + k_{\text{cool}} \cdot (T_{\text{ambient}} - T_{\text{set, cool}}) \quad (4-1)$$

The coefficients may depend on the day of the week and the hour of the day. For example, in an office building the HVAC load will depend in first place on the outside temperature  $T_{\text{ambient}}$ , but the operational setpoints usually differ in relation to building occupancy and cause different power loads. Some productions are also continuous and do not depend on external factors.

Power utilities make use of relations similar to Equation (4-1) with the coefficients determined by long-term observations in order to forecast energy demand. One of the most important information for power utilities to plan their generation schedule is a good weather forecast.

An example for the energy consumption analysis of an airport indicates that<sup>23</sup>

- 3.3% of the loads (for 30.5% of energy consumption in 2015) depend on the outside temperature, all related to HVAC systems.
- 65% of the loads (25% of energy consumption) are related to inhouse and and airfield lighting systems.
- 12.5% of the loads (9% of energy consumption) depend on the number of aircraft operations
- 3.5% of the loads (11% of energy consumption) in 2015 are influenced by the passenger traffic.

In this listing it is easy to recognize the loads depending on temperature, illumination, and activity. A striking aspect is that HVAC is responsible for a very large share of energy consumption with a small overall nominal load, while illumination, on the contrary, represents two thirds of the overall load but only one quarter of the energy consumption.

While loads considered singularly are mostly random – there isn't a way to determine beforehand at what exact time an air conditioning device will switch on or off – on the long-term loads can be treated statistically. The energy demand for air conditioning, for example, can be determined over a full day as function of the ambient air temperature.

---

23 Ortega Alba Sergio, Manana Mario, "Characterization and Analysis of Energy Demand Patterns in Airports", *Energies* 2017, 10, 119; doi:10.3390/en10010119

A customer-level load profile described by the coefficients in Equation (4-1) is determined with a multiple approach, for which spreadsheets are the perfect tool. On the one hand single pieces of equipment will be listed with their demand profiles and dependencies (on weather, product demand, etc.) On the other hand the model needs to be verified with actual measurements. In simple situations power bills with overall consumption figures may be sufficient, in more complex situations the total or partial load must be logged, e.g., on a 10- or 15-minute basis for some months and under different seasons. The data is then used to validate the initial model and to determine the contribution of each piece of equipment or complex function, such as illumination or HVAC, to the overall load. At this point the relation coefficients of power demand to ambient temperature, activity, etc. are also determined. A potential positive side-benefit of this analysis is that it may help identify losses and inefficient operation (e.g., unreasonably large differences between low loads and peaks) and thus find relevant solutions for more efficient energy consumption. In more general terms, at least up to a certain point reducing energy consumption is economically more convenient than feeding loads with renewable energy sources.

#### Program\_Code 4-2: script, load profile generation

<b>script</b>	<b>power_load_profile</b>
<b>description</b>	generate a load profile for equipment connected with different types of behavior
<b>parameters</b>	<b>filename</b> : TMY (.csv) dataset file name with directory path, string <b>parameters</b> for work hours, work days and work-off days, baseload, illumination, time-dependent loads and loads depending on the ambient temperature as described in the script code
<b>returns</b>	.csv file with 8760 hourly values for the load (W)

The **power\_load\_profile** script contains a full description of all parameters. Its operation is illustrated here with a practical example about a shopping center, a type of facility found almost everywhere. Shopping

centers are heavily dependent on electric power because of illumination, refrigeration (in particular, for food storage), and HVAC functions.

Several studies indicate that for a retail store with food handling an indicative load share is 50% for refrigeration, 20% illumination, 20% HVAC, 10% other loads. If refrigeration is not required the share becomes 50% illumination, 40% HVAC and 10% other loads.

In food retail points a rule-of-thumb for the total yearly energy consumption is 1 MWh per square meter. In the assumption of a 400 m<sup>2</sup> store the electric power consumption is therefore 400 MWh/year. On a hourly basis this means 45 kW average load, but due to the peaks the contractual power would need to be at least twice as high, in the range 100-120 kW. 50% of the load is for refrigeration, which is part of the baseload that is always present.

Because of their design, stores use much artificial and little natural illumination. In this example it is assumed that illumination is used only during work hours 8-20. The load is 80% of maximum during work hours, 100% when  $GHI < 600 \text{ W} \cdot \text{m}^{-2}$ , that is, when natural illumination cannot be used directly. HVAC equipment also draws much energy. The assumption here is that some HVAC load is connected all the time, for example, for ventilation, while a temperature-variable load increases from the hot/cold threshold temperatures up to reaching 100% at the limit temperatures for heating or cooling.

A graph for a load profile generated on the above-mentioned assumptions with its different components is shown in Figure 4-3. From the profile it appears that the most marked differences are not seasonal (the climate in Singapore is fairly constant, hot and humid) but reflect the weekly workday/work-off cycle.

For load profiles without actual, measured data much guesswork is required. The script [power\\_load\\_profile](#) provides a framework, while the user must be in control of the input parameters until a plausible result is reached. In case a large storage capacity is planned, the overall simulation will be less sensitive to short-term variations.

The column `total_load` of the load profile generated by the script can be used as input to the SAM PV 'electric load' function. This menu item is activated and appears on the screen on selection of the 'Photovoltaic (detailed)' and 'Residential (distributed)' financial model.

An example of load profile generated by the `power_load_profile` script is shown here (loads are indicated in Watt for each hour).

month	day	hour	base_load	work_load	GHI	illum_load	Tdry	heat_load	cool_load	total_load
2	5	8	27400	10800	50	29500	26	0	6250	73950
2	5	9	27400	10800	116	29500	27	0	12500	80200
2	5	10	27400	10800	176	29500	27	0	12500	80200
2	5	11	27400	10800	584	29500	29	0	25000	92700
2	5	12	27400	10800	655	0	30	0	31250	69450
2	5	13	27400	10800	672	0	29.5	0	28125	66325
2	5	14	27400	10800	634	0	29	0	25000	63200
2	5	15	27400	10800	541	29500	29	0	25000	92700
2	5	16	27400	10800	404	29500	28.5	0	21875	89575
2	5	17	27400	10800	236	29500	28.2	0	19999	87699
2	5	18	27400	10800	69	29500	28	0	18750	86450
2	5	19	27400	10800	2	29500	27	0	12500	80200
2	5	20	27400	10800	0	29500	26	0	6250	73950
2	5	21	27400	2160	0	0	25.7	0	4374	33934
2	5	22	27400	1080	0	0	25.5	0	3125	31605

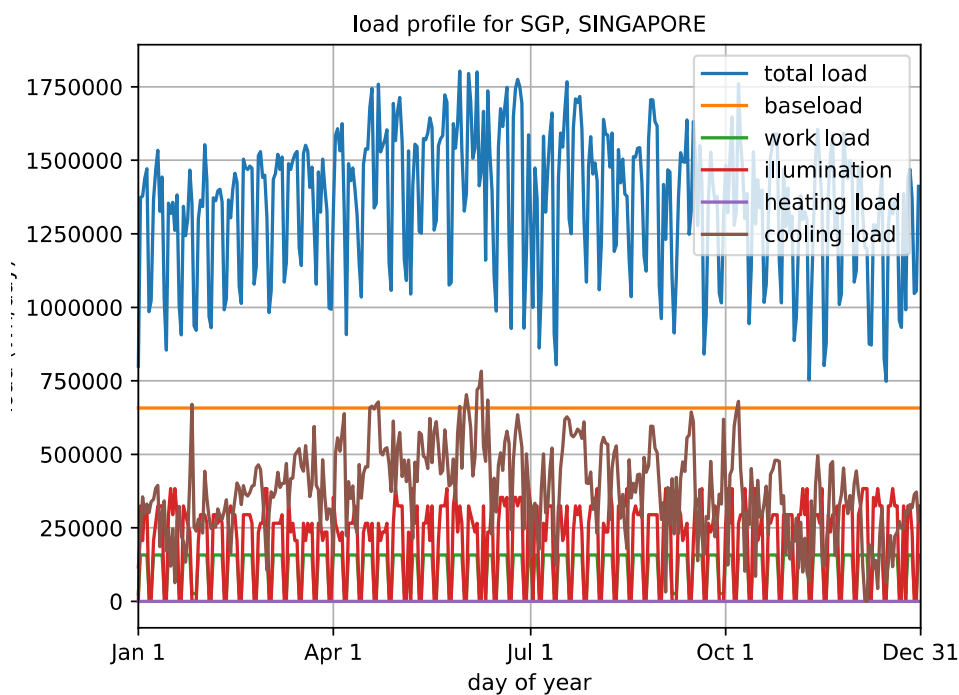


Figure 4-3: Load profile (Wh/day) generated on the basis of the Singapore TMY data. The difference in load between workdays and work-off days is clearly visible.



### 4.3 PV solar generator with storage and load

The last script of this primer (Program\_Code 4-3) combines different functions in order to provide a full simulation of a solar power system. Input data is a TMY file for a location, solar module parameters (either given manually or read from a component dataset, → Section 3.2.2), the number and azimuth/ tilt of solar modules and a load profile in TMY format as generated from the script of Program\_Code 4-2. The output is a .csv table with the most important input values, the calculated POA, the simulated system behavior for storage charge, external power demand to cover the load when PV generation is not sufficient, as well as the extra generation that can be fed into an external grid, when PV power fully covers the load and storage is already at its maximum.

The operation of any “regular” load requires in almost all cases the connection to a public grid to receive power when needed or feed in extra generated power. Island operation is extremely difficult, and will not work in locations with marked differences in the times and periods for the bulk of generation and highest demand, in particular in the temperate and the polar zones.

A PV simulation system is used to find answers to the basic questions for system sizing:

- what is the required size of a solar array at a location in order to cover a particular load?
- what is the required active capacity for electricity storage (that is, not considering minimum charge and other operational aspects)?
- what is the required external power exchange (demand from grid when the solar source is insufficient, feed-in into the grid when the solar source is in excess of load and storage requirements) in terms of connection load and purchase/sale of electricity over the year?

These questions can be answered quite rapidly by simulations with different parameters. They give indications to equipment sizing and, when necessary, support the definition of contractual conditions with the power company.

### Program\_Code 4-3: script, PV solar system simulation with load and storage

<b>script</b>	<b>power_pvgen_load</b>
<b>description</b>	full simulation of PV power generation connected to a load
<b>parameters</b>	<b>filename</b> : TMY (.csv) dataset file name with directory path, string <b>loadfile</b> : .csv file with a load profile on hourly basis [0..8759] as generated by the script <b>power_load_profile</b> (Program_Code 4-2), dataset file name with directory path, string <b>csv_filename</b> : PV module (.csv) dataset file name including directory path, string <b>module</b> : PV module name, string <b>parameters</b> for PV module, array, and system as described in the script source code
<b>returns</b>	- .csv file with 8760 hourly values including Tdry, GHI, DNI, DHI, POA, Tcell, generated power, load, storage charge status, power to get from the public grid and to feed to the grid. - average and total values of the output parameters

The script **power\_pvgen\_load** simulates the operation of a complete system for PV power generation, load, and storage. Thanks to its quick response and the presentation of the results on the output display, it supports the interactive definition of main system parameters, such as the PV array size and the storage capacity. The user can choose whether to limit some aspect, e.g., the array size, or to find the array size that provides for maximum load coverage.

An output example is the following:

```
{'State': '-', 'City': 'SINGAPORE', 'Location ID': '486980', 'Elevation': 16, 'Source': 'IWECC Data', 'Latitude': 1.37, 'Longitude': 103.98, 'Time Zone': 8.0, 'Country': 'SGP'}
```

```
===== system data
array azimuth=180  array tilt=1
```

```
PV module type = SunPower_SPR_X21_335_BLK
P_STC (W) = 335
PTC (W) = 313.7
NOCT =46.4
pvmod_Tcoeff =-1.0385
no of modules = 1080
installed power/ nameplate dc rating (W) =361800
module area (m2) = 1.631  array total area (m2) = 1761.5
```

```
Storage size (MWh) = 2.8
Storage time at avg load (days) = 2.06

===== location-specific TMY insolation data
GHI year total per m2 =1671.4 kWh/m2      per day =4.58 kWh/m2
DNI year total per m2 =687.7 kWh/m2      per day =1.88 kWh/m2
DHI year total per m2 =1145.8 kWh/m2      per day =3.14 kWh/m2
POA year total per m2 =1670.7 kWh/m2      per day =4.58 kWh/m2

===== main simulation results
Total yearly generation (MWh) = 500.89
Total load (MWh) = 496.54
Total external demand (MWh) = 12.16
Total extra generation (MWh) = 16.87

file saved data/data-output/PVgen_load_simulation_001.csv
>>>
```

In a practical dimensioning exercise it is easy to make several runs in order to identify, for example, the minimum array size and storage capacity that bring the total external demand to zero, i.e., to make a system independent of any external grid. However, in most cases the extra generation, i.e., the energy amount that is neither used by the load nor can be stored because the upper battery capacity limit has been reached, is quite high. In particular, in the temperate zones with large differences in solar energy availability between summer and winter this is an important problem and finding the right match between generation and storage to cover a given load is by far no trivial task. In the tropics, with more regular availability of solar energy and little influence of the seasons, this aspect is somewhat less important.

Another important aspect is that the calculated PV array area will almost always be larger, or much larger, of the building or the plant that represents the load. The storage devices will also take up a large volume. This is a different way to say that renewable energy does not come for free, and that the most rational way to use it in most cases is not alone, but integrated in a grid.

That said, the presented script is not a simulation program, but rather a support for the analysis of a technical system that includes resource availability and load. If necessary, it can be adapted to match specific situations, for which Python is a very versatile tool.

## Appendix. Online Sources for Solar Irradiation Data

Service	Website
ESMAP	<a href="http://www.esmap.org/re_mapping">http://www.esmap.org/re_mapping</a> <a href="http://globalsolaratlas.info/knowledge-base/data-description">http://globalsolaratlas.info/knowledge-base/data-description</a>
NREL USA	<a href="https://maps.nrel.gov/nsrdb-viewer/#/">https://maps.nrel.gov/nsrdb-viewer/#/</a> <a href="http://www.nrel.gov/gis/solar.html">http://www.nrel.gov/gis/solar.html</a>
NREL India maps	<a href="http://www.nrel.gov/international/ra_india.html">http://www.nrel.gov/international/ra_india.html</a>
NREL International maps	<a href="http://www.nrel.gov/international/">http://www.nrel.gov/international/</a> <a href="http://www.nrel.gov/gis/international.html">http://www.nrel.gov/gis/international.html</a>
Open Energy Information	<a href="http://en.openei.org/wiki/Main_Page">http://en.openei.org/wiki/Main_Page</a>
SWERA	<a href="http://en.openei.org/apps/SWERA/">http://en.openei.org/apps/SWERA/</a>
EU PVGIS (Photovoltaic Geographical Information System)	<a href="http://re.jrc.ec.europa.eu/pvgis.html">http://re.jrc.ec.europa.eu/pvgis.html</a> <a href="http://re.jrc.ec.europa.eu/pvgis/cmmaps/eur.htm">http://re.jrc.ec.europa.eu/pvgis/cmmaps/eur.htm</a>
Solargis	<a href="http://solargis.com/products/maps-and-gis-data/free/overview/">http://solargis.com/products/maps-and-gis-data/free/overview/</a>
NASA Atmospheric Science Data Center	<a href="https://eosweb.larc.nasa.gov/sse/">https://eosweb.larc.nasa.gov/sse/</a>
IRENA	<a href="http://globalatlas.irena.org/Default.aspx?tid=-1">http://globalatlas.irena.org/Default.aspx?tid=-1</a>

*Table 1: Principal online map sources, calculation tools for solar irradiation data*

Besides the solar irradiation values, other parameters may be reported, for example the temperature maxima, minima, and their averages over different periods of time. Several services now allow an estimation of PV electricity production under some simplified assumptions, such as standard efficiency coefficients for the solar modules and for the ancillary equipment. Many solar irradiation databases also indicate the optimal tilt angle for solar flat plane collectors at any given location in order to maximize the solar energy yield over the year.