v.1.0

A software project initiated by Ulrich W. Paetzold

at

Karlsruher Institut für Technologie

Table of contents

Energy Yield Calculator (EYCALC)

Introduction

This software aims to simulate the energy yield of single-junction and multi-junction solar cells. In contrast to the power conversion efficiency (PCE), the energy yield (EY) accounts for environmental conditions, such as constantly changing irradiation conditions or the ambient temperature.

This software allows a rapid simulation of complex architectures and was developed with the aim to handle textured perovskite-based multi-junction devices. However, it is possible to simulate any combination of thin-film architecture with incoherent photovoltaic materials (e.g., crystalline silicon).

By making use of pre-simulated textures (e.g., inverted pyramids, regular upright pyramids, random pyramids) by geometrical ray tracing, any incoherent interface within the architecture can also be textured.

The software is available as source code and as a simple to use graphical user interface (GUI), which requires either a MATLAB (>R2017a and >R2020b, respectively) installation or the MATLAB runtime.

Basic Features

The basic features of the EYcalc are:

- Spectral and angular-resolved realistic irradiance data (from 1020 locations in the USA) is used
- A simple cloud model is used to adjust the diffuse irradiation
- Fast optical simulations, by combining the transfer matrix method and geometric ray tracing
- Optics can handle arbitrary combinations of thin (coherent) and thick (incoherent) layers, which also can be textured
- Single- and multi-junction solar cells can be simulated
- No limitation on the number of absorbers
- Energy yield is computed for different electrical interconnection schemes (e.g., 2T, 3T, 4T)
- Energy yield can be derived for constant tilt (and constant rotation) angle
- Energy yield can be derived for various tracking algorithms (e.g., 1-axis, 2-axis)
- Bifacial solar cells can be simulated
- Albedo can be considered by choosing one out of 3400 spectra of natural and man-made materials from the ECOSTRESS spectral library

Modular framework

The software is divided into individual modules, which handle the irradiation, optics, electrics and energy yield simulations. Those modules can also be operated independently (e.g., calculate the reflectance, transmittance, absorptance of a solar cell architecture).

- The **Irradiation Module** calculates the spectral and angular-resolved irradiance over the course of one year with a temporal resolution of one hour by applying SMARTS to typical meteorological year (TMY3) data of locations in various climatic zones. A simple model is employed to account for cloud coverage such that realistic direct and diffuse irradiance are derived.
- The **Optics Module** rapidly calculates the spectral and angular-resolved absorptance of the non-simplified architecture of multi-junction solar cells. It is able to handle multiple planar and textured interfaces with coherent and incoherent light propagation by combining transfer matrix method (TMM) and geometrical ray-tracing.

- The **Electrical Module** determines the temperature-dependent current density-voltage (*J-V*) characteristics accounting for series and shunt resistances for a given short-circuit current density (J_{SC}) of the sub-cells forming the multi-junction in either a 2T-, 3T- or 4T-configuration. Furthermore, the maximum power point is determined to calculate the power output of the multi-junction solar module.
- The **Energy Yield Core Module** calculates the EY over the course of one year of the sub-cells depending on their orientation (rotation and/or tilt of the module) and location. The EY is computed by combining the spectral and angular resolved solar irradiation (with or without albedo), the absorptance of the multi-junction solar cell and the electrical properties.

Credits

This software project was initiated by **Ulrich W. Paetzold**. The **code development** was driven by:

- **Raphael Schmager** (energy yield core, irradiance module, optics module, electrics module, GUI)
- **Malte Langenhorst** (optics module, irradiance module)
- **Jonathan Lehr** (electrics module, albedo)
- **Fabrizio Gota** (numerical modelling on 3T interconnection, optics module)

The financial support by the following **projects and grants** is gratefully acknowledged:

- PERCISTAND (funding code: 850937), European Union's Horizon 2020 research and innovation programme
- Helmholtz Young Investigator Group of U. W. Paetzold (funding code: VH-NG-1148), Helmholtz Association
- PEROSEED (funding code: ZT-0024), Helmholtz Association
- CAPITANO (funding code: 03EE1038B), Federal Ministry for Economic Affairs and Energy
- 27Plus6 (funding code: 03EE1056B), Federal Ministry for Economic Affairs and Energy

This software uses codes and data from **other programmers and resources**:

- Parts of the transfer matrix code is taken from Steven Byrnes
- Matlab implementation of the NREL solar position algorithm by Vincent Roy
- Logarithmic Lambert W function from Michael
- The SMARTS from Dr. Christian A. Gueymard see also
- The TMY3 data from the National Solar Radiation Database
- Reference Air Mass 1.5 Spectra
- ECOSTRESS spectral library for albedo

Getting started

To use all features of the EYcalc software, you need to download and add some external files, like the SMARTS code and the TMY3 data. Please see our setup guide for help in setting up the required external files! On our wiki page you can also find a detailed description for each of the modules as well as a quick start guide.

Contributing

If you want to contribute to this project and make it better, your help is very welcome!

Contact

For any questions regarding the software, please contact Ulrich W. Paetzold.

Citing

If you use our software or parts of it in the current or a modified version, you are obliged to provide proper attribution. This can be to our paper describing the software:

- R. Schmager and M. Langenhorst et al., Methodology of energy yield modelling of perovskite-based multi-junction photovoltaics, Opt. Express. (2019). doi:10.1364/oe.27.00a507.

or to this code directly:

- EYcalc - Energy yield calculator for multi-junction solar modules with realistic irradiance data and textured interfaces. (2021). doi.org/10.5281/zenodo.4696257.

License

Interested in a sublicense agreement to use EYcalc in a non-free/restrictive environment? Contact Ulrich W. Paetzold!

Further reading

This energy yield software has been used in the following publications:

- M. De Bastiani et al., Efficient bifacial monolithic perovskite/silicon tandem solar cells via bandgap engineering, Nature Energy. (2021). doi.org/10.1038/s41560-020-00756-8.
- J. Lehr et al., Numerical study on the angular light trapping of the energy yield of organic solar cells with an optical cavity, Opt. Express. (2020) doi.org/10.1364/OE.404969.
- F. Gota et al., Energy Yield Advantages of Three-Terminal Perovskite-Silicon Tandem Photovoltaics, Joule, (2020). doi.org/10.1016/j.joule.2020.08.021.
- J. Lehr et al., Energy yield of bifacial textured perovskite/silicon tandem photovoltaic modules, Sol. Energy Mater. Sol. Cells. (2020). doi:10.1016/j.solmat.2019.110367.
- R. Schmager et al., Methodology of energy yield modelling of perovskite-based multi-junction photovoltaics, Opt. Express. (2019). doi:10.1364/oe.27.00a507.
- M. Langenhorst et al., Energy yield of all thin-film perovskite/CIGS tandem solar modules, Prog. Photovoltaics Res. Appl. (2019). doi:10.1002/pip.3091.
- J. Lehr et al., Energy yield modelling of perovskite/silicon two-terminal tandem PV modules with flat and textured interfaces, Sustain. Energy Fuels. (2018). doi:10.1039/c8se00465j.

Setup

In order to calculate the spectral and hourly resolved irradiance data for the different locations, you need to download the SMARTS (Simple Model of the Atmospheric Radiative Transfer of Sunshine) and the TMY3 (Typical Meteorological Year) dataset!

You can also download the pre-calculated irradiance data for Miami in our first release. Extract the files to `Irradiance\Spectra_722020TYA_Miami\` (see below for the exact folder structure). In this case you can skip (1) downloading the SMARTS code and the TMY3 dataset and (2) the calculation of the irradiance data, which takes (once) ~30 min for each location. However, if you want to work with different locations, you have to follow the guide below!

SMARTS

In order to download the SMARTS code, you need to register and agree their license requirements!

Download the files and extract them under:

```
Irradiance\Code_SMARTS_295_PC\
|--- Albedo
|--- CIE_data
|--- Documentation
|--- Examples
    ...
|--- smarts.295.exe
|--- smarts295.xls
|--- smarts295bat.exe
```

TMY3

You can find the TMY3 datasets on their website. Download the National Solar Radiation Database 1991-2005 and extract the files here:

```
Irradiance\Dataset_TMY3\
|--- 690150TYA.CSV
|--- 690190TYA.CSV
|--- 690230TYA.CSV
    ...
|--- 912850TYA.CSV
|--- User Manual TMY3.pdf
```

Calculate the irradiance files

Now everything should be there and you can start to simulate the spectral and hourly resolved realistic irradiance files.

You can do this by the example code provided in the `main.m`. First select a location from the `User Manual TMY3.pdf` (p 23) and define the *code* and *alias*:

```
CodeLocation  = '722020TYA';
AliasLocation = 'Miami';
```

Next, simulate the irradiance data by calling the irradiance function:

```
Irradiance(CodeLocation, AliasLocation);
```

The irradiance function then calls three sub functions, which

1. loads the TMY3 dataset for the specified location
2. calculates the clear sky irradiance
3. enhances the irradiance by a simple cloud model

```
% Extract the data from the TMY3 datasets
extractTMY3(Code_location,Alias_location)

% Calculate the irradiance spectra with help of the SMARTS code
calcSMARTS(Code_location,Alias_location)

% Enhance irradiance by a simply clouds model
simpleclouds(Code_location,Alias_location)
```

Note: Step 2, `calcSMARTS()` might fail on slower systems! In this case, you can increase some delays. See the debugging details below!

Once everything is done, the data is stored in the irradiance folder and will look like:

```
Irradiance\
|--- Spectra_722020TYA_Miami
    |--- Irr_spectra_clear_sky.mat
    |--- Irr_spectra_clouds.mat
    |--- TMY3_722020TYA_Miami.mat
|--- Spectra_722780TYA_Phoenix
    ...
|--- Spectra_726060TYA_Portland
    ....
```

ECOSTRESS spectral library

In order to use albedo, you need to download and extract the ECOSTRESS spectral data to the following folder:

```
EnergyYield\LibraryEcospec\
|--- artificialblack.txt
|--- artificialwhite.txt
|--- manmade.concrete.constructionconcrete.solid.all.0598uuucnc.jhu.becknic.spectrum.txt
|--- rock.sedimentary.sandstone.coarse.all.greywacke_1.jhu.becknic.spectrum.txt
|--- rock2.sedimentary.sandstone.fine.all.sandstone_1.jhu.becknic.spectrum.txt
    ....
```

Initially, only `artificial black` and `artificial white` are included. If you like add other or define your own grounds, search in the ECOSTRESS library or consult the ECOSTRESS documentation.

Debugging

On slower systems, the calculation of the irradiance data might fail due to a too short delay! You can simply increase the delay pause(xx) in the calcSMARTS() function:

```
[...]
while exist([location_SmartS,'\smarts295.out.txt'], 'file') ~= 2
   pause(0.02);  % you might need to increase this value a bit
end
while true
   pause(0.02);  % you might need to increase this value a bit
   try
      Data_import = importdata([location_SmartS,'\smarts295.ext.txt'],' ',1);
      break
   catch
      continue
   end
end
[...]
```

Quick Start Guide

Download and extract the **EYcalc** project. Open the main.m, which contains all definitions and settings to calculate the energy yield (EY) of an exemplary perovskite/c-Si multi-junction solar cell.

Note: The EYcalc software will not run without additional files. You either need to add pre-calculated irradiance data or calculate the irradiance data for locations of your choice. Before you continue, see the setup guide for details.

General settings

First, we add all sub folders to the current working path. Nest, you need to set the paths to store data. If StoreInDatabase is true, the Optics and Energy Yield core module auto-saves the results and loads them in case the data exists.

Note: you need to define valid paths, even if you use StoreInDatabase = false.

```
% ### PATH ###
addpath(genpath(pwd));

% ### DATABASE ###
% Use database to store simulations and load already simulated data
StoreInDatabase = false;
PathOpticsResults = 'path/to/optics/data';
PathEYResults = 'path/to/ey/data';
```

Next, we need to specify the complex refractive indices for all layers of interest. This is done by loading them from e.g., an Excel (*.xlsx) database:

```
[IndRefr.nkdata,IndRefr.names]=xlsread('_RefractiveIndexLib.xlsx');
```

Note: the refractive index data is defined in 1 nm steps. The first column contains the wavelengths, the next columns contain the real and imaginary part of the complex refractive indices. If you add new data, use the same unique name to identify the material and add the suffix: _n and _k.

Irradiance

The energy yield will be calculated for a specific location, covered by the TMY3 dataset. The code you downloaded does not contain this dataset (~1.7GB). However, it can be obtained for free here. Please check out the setup guide for this!

Note: If you want to skip the (first) calculation, you can download the pre-calculated irradiance data for Miami in our first release. Extract the files to Irradiance\Spectra_722020TYA_Miami\ (see the setup guide for details).

In the main.m, we first define the location alias and the corresponding location code. You'll find the alias and code in the TMY3 user manual (p23).

```
AliasLocation = 'Miami';          % To be specified
CodeLocation = '722020TYA';       % Code to be looked up from User Manual TMY3.pdf
```

Next, the Irradiance data is calculated and saved e.g., in Irradiance\Spectra_722020TYA_Miami\.

```
Irradiance(CodeLocation, AliasLocation);
```

The calculation might take some time (~20 min) for one location. However, once performed for any of the possible locations, the stored data is simply loaded.

Note: it is also possible to load the AM1.5G reference spectrum for testing. For this, you need to use:

```
AliasLocation = 'Spectra_AM1.5G';
CodeLocation = 'spectrum';
```

Optics

To calculate the absorptance in the absorber layers, the optics module is called. Here, we first define the stack by the names specified in the refractive index database, its layer thicknesses, and its morphology.

Note: the boundary layers need to be incoherent and their thicknesses should be Infinite.

For each incoherent layer (except the first one) a morphology needs to be defined. The morphology works upwards. In this example the cSi layer is double-sided textured - the front air/glass interface is Flat.

```
Stack = {'Air','MgF2','Glass1.5','ITOfront','SnO2','Pero1.62','SpiroOMeTAD',...
   'ITOfront','aSi(n)','aSi(i)','cSi','aSi(i)','aSi(p)','ITOfront','Air'};
LayerThickness = [inf,100,1E4,100,10,450,20,25,5,5,250E3,5,5,100,inf];
Morphology = {'Flat','RandomUpright','RandomUpright'};
Polarization   = 'mixed';
lambdaTMM      = 300:5:1200;
AngleResolution = 5;
IncoherentLayers = {'Air','Glass','EVA','Encapsulation','PDMS','cSi'};
bifacial = false;
Absorbers = {'Perovskite','cSi'};
```

In addition, the polarization for the transfer matrix simulations can be defined. The wavelengths (lambdaTMM) for the transfer matrix simulations needs to be provided in nanometers. The AngleResolution defines the spacing for the angular depended transfer matrix calculations and should not be larger than 5°.

It is possible to define which layers should be treated as incoherent. If not specified, the optics code automatically treats layers with thicknesses > 5µm incoherent. This threshold could be modified in OpticsModule.m.

If the EY of a bifacial solar module should be calculated, the bifacial option needs to be true. Then, the optics code simulates the stack from both sides.

For proper indexing, it's best to define the absorbers by their names in the stack. Then the energy yield core module takes the right layers to calculate the short-circuit current densities. If the absorbers are not defined Absorbers = {}, the optics code auto-detects them. Usually, this works fine as well.

Finally, we call the OpticsModule() :

```
optics = OpticsModule(IndRefr, Stack, LayerThickness, AngleResolution, Morphology, bifacial, Polarization, lambdaTMM, PathOpticsResults, StoreInDatabase, IncoherentLayers, Absorbers);
```

Electrics

The electrics module is called by the energy yield core module. However, we need to predefine all the electrical parameters. For each absorber within the stack, one needs to define the properties below. This means, for n absorbers, the properties need to be [1 x *n*] in size.

```
electrics.configuration = '2T';          % 2T, 3T, 4T, 2T exp, 3T exp, 4T exp
electrics.shunt = 'with';                % with, without
electrics.RshTandem = 1000;              % shunt resistance of tandem device
electrics.RsTandem = 3;                  % serial resistance of tandem device
electrics.Rsh = [1300, 800];             % shunt resistance of n-th cell
electrics.Rs = [2, 1];                   % serial resistance of n-th cell
electrics.CE = [1, 1];                   % collection efficiency of n-th cell
electrics.[2.7e-18, 1e-12];              % reverse-blocking current of n-th cell
electrics.n = [1.1, 1];                  % ideality factor of n-th cell
electrics.Temp = [25, 25];               % temperature of cells (can also be n vectors)
electrics.NOCT = [48, 48];               % nominal temperature of n-th cell, if a number, Temp is overwritten
electrics.tcJsc = [0.0002, 0.00032];     % temperature coefficient of Jsc in K^-1 of n-th cell
electrics.tcVoc = [-0.002, -0.0041];     % temperature coefficient of Voc in K^-1 of n-th cell
```

Energy Yield

The energy yield calculations rely on the previous calculations and definitions. Further, we need to define the tilt and rotation angle of the solar module. They can both be 0, which means that the solar panel is lying flat on the ground.

Note: Tilting the module facing the southern hemisphere can be achieved by a rotation angle of 180° and a tilt angle >0°. If the rotation angle is 0°, a tilt angle >0°, tilts the module facing north.

```
SolarCellRotationAngle = 180;
SolarCellTiltAngle = 20;
```

Next, we define, if we would like to use one of the tracking methods (see main.m for details) and, if albedo should be taken into account. The albedo string needs to match an existing file in the EnergyYield\LibraryEcospec\ folder, whereas the *.txt file extension is missing in the string. By default, only artificial black and artificial white are included. If you like add other or define your own grounds, check out the ECOSTRESS library or consult the ECOSTRESS documentation.

Note: In a tilted module configuration, the albedo can also reach the front of a solar module. In case of a bifacial simulation, additional contributions from the semi-transparent rear due to albedo and due to direct and diffuse irradiation is used.

```
tracking = 0;
groundtype = 'artificialblack';
```

Finally, the energy yield is calculated by calling the EnergyYield() function:

```
EY = EnergyYield(irradiance, optics, electrics, SolarCellRotationAngle, SolarCellTiltAngle, tra
cking, albedo, groundtype, PathEYResults, StoreInDatabase);
```

An example output of the EY struct, is documented here.

Irradiance Module

This module calculates realistic irradiation data, which is used as input for the energy yield calculations. For this the TMY3 data sets, available from the National Renewable Energy Laboratory (NREL), are used. They contain statistically representative and hourly resolved meteorological / irradiation data of many locations spread over the USA, representing all relevant climatic zones. For each location, they contain measured atmospheric properties like dry-bulb temperature, pressure, precipitable water, aerosol optical depth and albedo. Based on all these properties, the irradiance and the sun's position, the spectrally resolved (280 nm - 4000 nm) clear sky irradiance is calculated with SMARTS. In SMARTS, we use the Shettle and Fenn's urban aerosol model and the US standard reference atmosphere.

The obtained clear sky irradiance is further enhanced by a simple cloud model. For this, the cloud cover, which is available in the TMY3 dataset is used. This simple cloud model assumes no spectral change for the direct irradiation. The diffuse irradiation however, is assumed to be composed by the direct and diffuse clear sky irradiance weighted by the cloud cover. The normalized spectral data is finally scaled to the measured diffuse irradiance of the TMY3 data.

$$I_{\text{clouds,dir}}(\lambda) = \frac{I_{\text{clear,dir}}(\lambda)}{\int I_{\text{clear,dir}}(\lambda)d\lambda} \cdot I_{\text{meas,dir}} \cdot$$

$$\mathrm{I}_{\text{clouds,diff}}(\lambda) = \frac{\mathrm{I}_{\text{clear,diff}}(\lambda) \cdot (1 - \mathrm{CC}) + \mathrm{I}_{\text{clear,dir}}(\lambda) \cdot \mathrm{CC}}{\int \left[\mathrm{I}_{\text{clear,diff}}(\lambda) \cdot (1 - \mathrm{CC}) + \mathrm{I}_{\text{clear,dir}}(\lambda) \cdot \mathrm{CC}\right] \mathrm{d}\lambda} \cdot \mathrm{I}_{\text{meas,diff}} \cdot$$

Usage

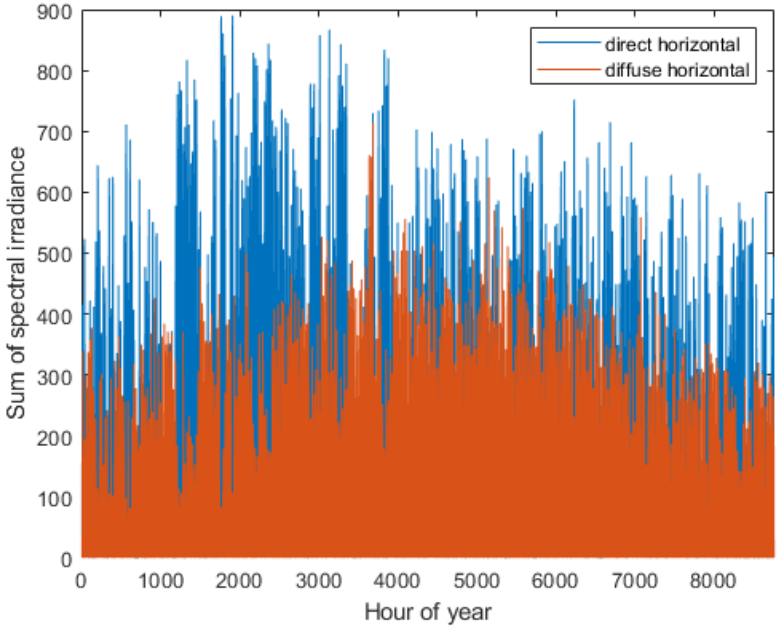
Loading the irradiance data according to the example in the `main.m` file, we can analyze the spectral and hourly resolved data.

```
CodeLocation = '722020TYA';
AliasLocation = 'Miami';

load(['Irradiance/Spectra_',num2str(CodeLocation),'_',num2str(AliasLocation),'/TMY3_',nu
m2str(CodeLocation),'_',num2str(AliasLocation),'.mat']);
```

For example, the direct and diffuse horizontal irradiance can be plotted. To show the irradiance for the full exemplary year for each hour of the day, we take for simplicity the sum over the spectral data.

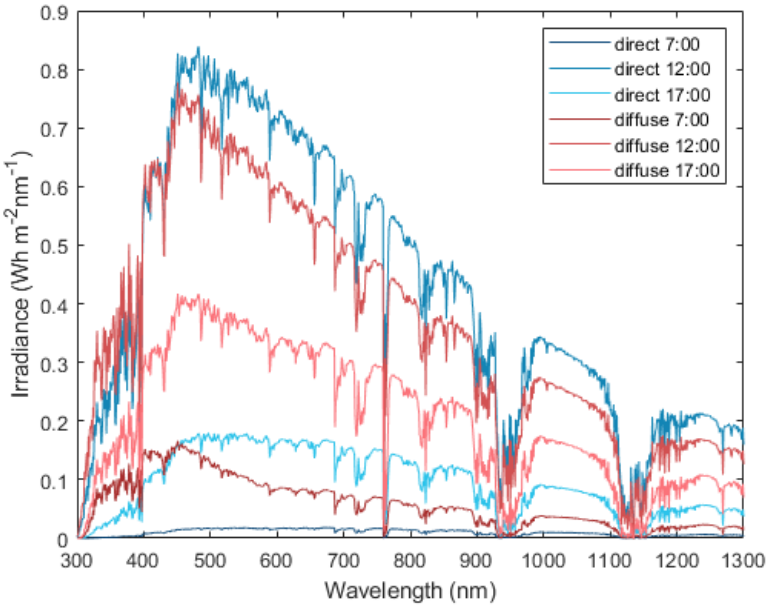
```
t = 1:8760;
lambda = irradiance.Irr_spectra_clouds_wavelength;
Idir = irradiance.Irr_spectra_clouds_direct_horizontal;
Idiff = irradiance.Irr_spectra_clouds_diffuse_horizontal;

figure;
plot(t,sum(Idir,2),'DisplayName','direct horizontal'); hold on
plot(t,sum(Idiff,2),'DisplayName','diffuse horizontal'); hold off
xlim([0 8760])
xlabel('Hour of year'); ylabel('Sum of spectral irradiance');
legend('show')
```

This brings us:



Next, we have a look to the spectral shape of the direct and diffuse irradiance for 3 different hours on an exemplary day. Here we choose 12. May, at 7:00, 12:00 and 17:00. These hours are indexed as follows: 3151:5:3161.

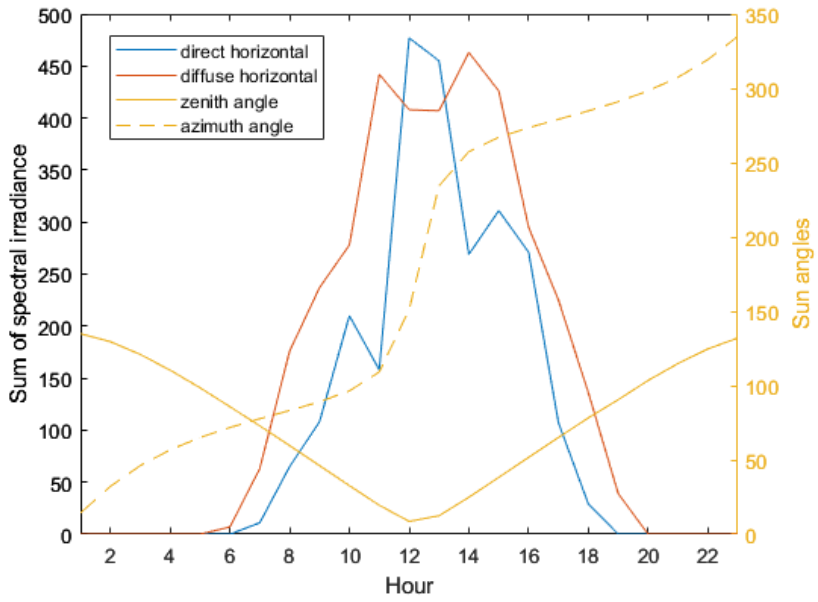
```
figure;
ax = axes;
plot(lambda,Idir(3151:5:3161,:)); hold on
plot(lambda,Idiff(3151:5:3161,:)); hold off
ax.ColorOrder = [ 0.0667 0.2980 0.4824; 0.0745 0.5216 0.7098; 0.1725 0.7647 0.9176;...
   0.6588 0.1843 0.1804; 0.8235 0.3176 0.3216; 0.9843 0.4510 0.4784];
xlim([300 1300])
legend({'direct 7:00', 'direct 12:00', 'direct 17:00',...
   'diffuse 7:00', 'diffuse 12:00', 'diffuse 17:00'})
xlabel('Wavelength (nm)'); ylabel('Irradiance (Wh m^{-2}nm^{-1})');
```



The influence of time and clouds can be clearly observed.

A more detailed analysis is also possible. Using again the sum over the spectral data, we can display the sun's azimuth and zenith angles together with the diffuse and direct horizontal irradiance over the course of one day.

```
thetasun = irradiance.Data_TMY3(:,7);
phisun = irradiance.Data_TMY3(:,8);

figure;
ax = axes;
plot(1:23,sum(Idir(3145:3167,:),2),'DisplayName','direct horizontal'); hold on
plot(1:23,sum(Idiff(3145:3167,:),2),'DisplayName','diffuse horizontal');
ylabel('Sum of spectral irradiance');
yyaxis right
plot(1:23, thetasun(3145:3167),'DisplayName','zenith angle')
plot(1:23, phisun(3145:3167),'DisplayName','azimuth angle'); hold off
xlabel('Hour'); ylabel('Sun angles'); legend('show')
xlim([1 23])
```



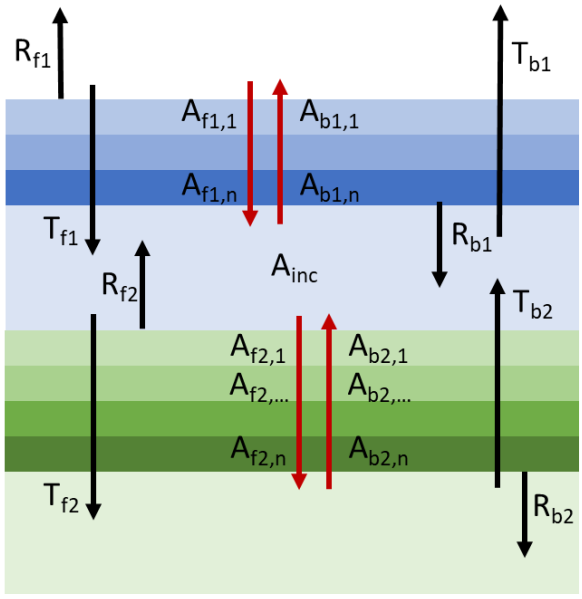
Optics Module

The optics module calculates the spectrally and angular-resolved absorptance for a variety of different architectures of single or multi-junction solar cells. It is able to handle multiple planar and textured interfaces with coherent and incoherent light propagation by combining the transfer matrix method (TMM) and geometrical ray-tracing.

First, the defined architecture is analyzed and divided into optically thin (coherent) partial stacks and optically thick (incoherent) layers. In this regard, a coherent layer stack is defined by two incoherent boundary layers. The light propagation and absorption in any incoherent layer separating two of such partial stacks is treated with the Beer-Lambert law, with which the absorption is calculated. For multiple adjacent incoherent layers, the transmittance and reflectance at their interfaces are calculated with the Fresnel equations.

For each partial stack and for each incoherent layer the reflectance, transmittance and absorption in forward and in backward direction are determined. All these quantities are spectrally resolved.

In order to calculate the total reflectance, transmittance and the absorptance in each layer, these quantities are connected by subsequently adding partial stacks or incoherent layers. A simple example is illustrated in the figure. Here, two partial stacks consisting of 3 and 4 coherent layers are connected. The properties: reflectance R and transmittance T for the coherent layers and the absorptance A for the coherent layers for light propagating in forward and backward direction are shown. In addition, the absorptance A_{inc} in the incoherent layer is shown, at which the stacks are merged.



First, reflectance is derived by following the ray interactions (see above figure) within the joined stack. The total reflectance is then the sum over the individual paths:

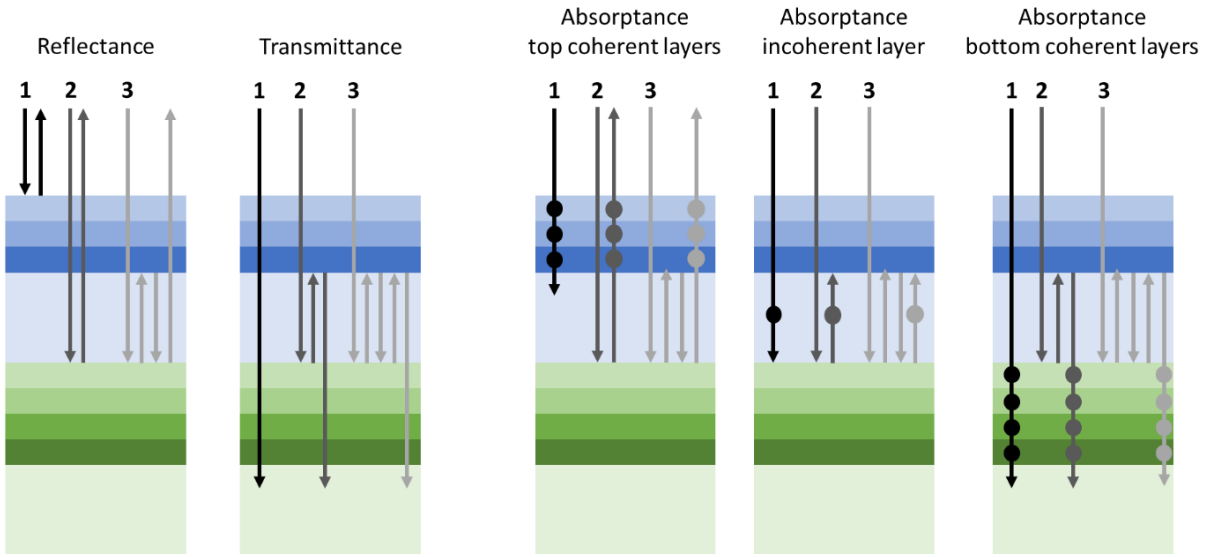
$$R_f = R_{f1} + T_{f1}[\dots]T_{b1} + \sum_j T_{f1}([\dots]R_{b1})^j([\dots]T_{b1})$$

Where we use the following abbreviation:

$$[\dots] = (1 - A_{inc})R_{f2}(1 - A_{inc})$$

Next, the transmittance is derived in an analogous way, which leads to:

$$T_f = T_{f1}(1 - A_{inc})T_{f2} + \sum_j T_{f1} \cdot ([\dots]R_{b1})^j (1 - A_{inc})T_{f2}$$



Finally, the absorptance is calculated. Here, three cases need to be distinguished. The absorptance in the coherent layers of the first partial stack (=top):

$$A_f(f_1, 1..n) = A_{f1} + T_{f1}[\dots]A_{b1} + \sum_j T_{f1}([\dots]R_{b1})^j([\dots]A_{b1}),$$

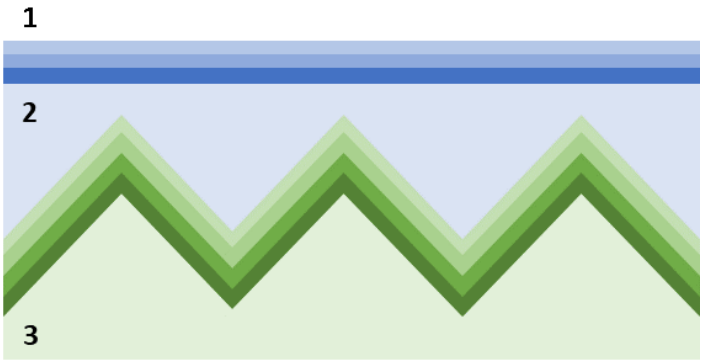
the absorptance of the incoherent layer connecting the two partial stacks:

$$A_f(i) = T_{f1}A_{inc} + T_{f1}(1 - A_{inc})R_{f2}A_{inc}$$
$$+ \sum_j T_{f1}([\dots]R_{b1})^j A_{inc} + T_{f1}([\dots]R_{b1})^j (1 - A_{inc})R_{f2}A_{inc},$$

and the absorptance in the coherent layers of the second partial stack (=bottom):

$$A_f(f_2, 1..n) = T_{f1}(1 - A_{inc})A_{f2} + \sum_j T_{f1}([\dots]R_{b1})^j (1 - A_{inc})A_{f2}\,.$$

The advantage of this approach is, that for each of the incoherent layers, defined within the full stack, it is possible to add a texture. The texturing can be done by setting the morphology for each incoherent layer. The texturing then is added at the interface between this incoherent layer and the above layer(s) / layer stack. In the above architecture with 3 incoherent layers, texturing the last (3) incoherent layer will lead to the following scenario:



Note: since texturing of the first (1) incoherent layer facing upwards makes no sense, the input of the morphologies start at the second (2) layer: e.g. {'Flat', 'Texture'}.

In order to use a specific texture, either the redistribution matrices for R, A and T need to be a known property. Or they will be calculated by the use of path data extracted from OPAL2.

The geometrical ray-tracing as described by Baker-Finch and McIntosh is an elegant approach that allows fast calculation of the characteristic paths for textures (e.g., pyramids). This gives us a limited set of paths with its probabilities and their intersection angles.

Usage

First, we define the input parameters and call the optics analogous to the description in the quick start guide.

```
Stack = {'Air','MgF2','Glass1.5','ITOfront','SnO2','Pero1.62','SpiroOMeTAD',...
   'ITOfront','aSi(n)','aSi(i)','cSi','aSi(i)','aSi(p)','ITOfront','Air'};
LayerThickness = [inf,100,1E4,100,10,450,20,25,5,5,250E3,5,5,100,inf];
Morphology = {'Flat','RandomUpright','RandomUpright'};
Polarization = 'mixed';
lambdaTMM = 300:5:1200;
AngleResolution = 5;
IncoherentLayers = {'Air','Glass','cSi'};
bifacial = false;
Absorbers = {'Perovskite','cSi'};

optics = OpticsModule(IndRefr, Stack, LayerThickness, AngleResolution, Morphology, bifacia
l, Polarization, lambdaTMM, PathOpticsResults, StoreInDatabase, IncoherentLayers, Absorbe
rs);
```

This gives us a struct called optics, which contains the interesting quantities:

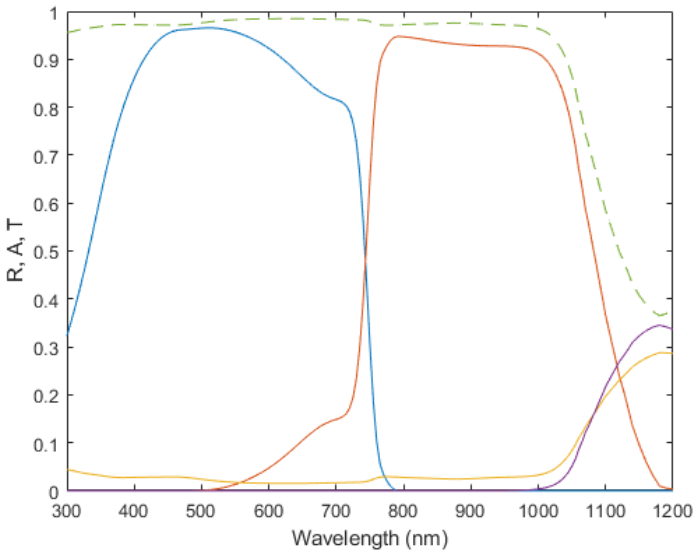
```
>> optics

optics =

  struct with fields:

       Reflectance: [90×90×181 single]
      Absorptance: [13×90×181 single]
    Transmittance: [90×181 single]
           lambda: [1×181 double]
    AbsorberIndex: [5 10]
                A: [181×2 single]
                R: [181×1 single]
                T: [181×1 single]
             hash: "DCCC4EDAAA22C57CFDEA5578ADA460A744DBE13F"
```

Reflectance and Transmittance are given as tensors of dimension: [angle of incidence x angle of escape x wavelength]. The dimension of the Absorptance matrix is [number of layer x angle of incidence x wavelength]. In addition, those quantities are also provided for normal incidence only and summed up for all escape angles: A, R, and T. Moreover, the absorption is only given for the absorbers.

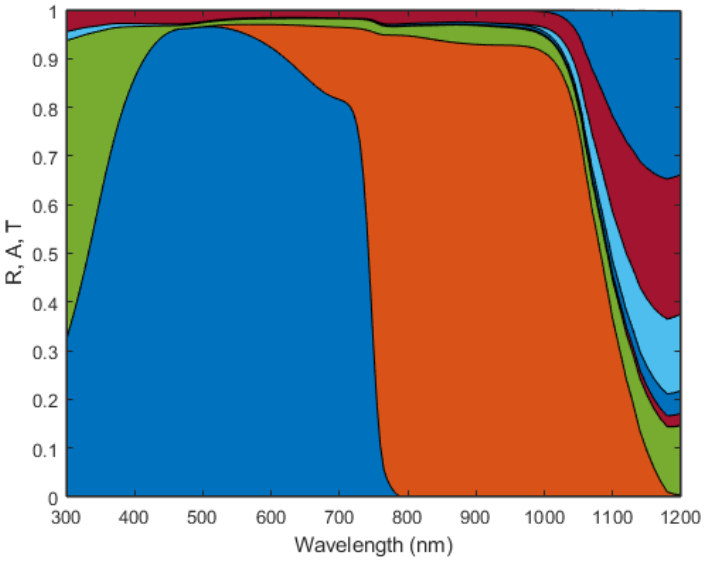
We can plot this:

```
Aall = squeeze(sum(optics.Absorptance(:,1,:))); % get total absorptance
```

```
figure;
plot(lambdaTMM, optics.A ); hold on
plot(lambdaTMM, optics.R );
plot(lambdaTMM, optics.T );
plot(lambdaTMM, Aall,'--');hold off
xlabel('Wavelength (nm)');
ylabel('R, A, T')
```




```
A = squeeze(optics.Absorptance(optics.AbsorberIndex,1,:))';
Aparasitic = squeeze(optics.Absorptance(setdiff(1:size(optics.Absorptance,1), optics.Absorb
erIndex),1,:))';

figure;
area(lambdaTMM, [A, Aparasitic, optics.R, optics.T] );
xlabel('Wavelength (nm)'); ylabel('R, A, T')
```



Electrics Module

The electrics module calculates the power output of the single junctions and of the multi-junction solar cell. For this, the Shockley diode equation including series R_S and shunt R_{SH} resistance is used:

$$J(V) = J_{SC} - J_0\left(e^{\frac{V+JR_s}{nkT_{module}}} - 1\right) - \frac{V + JR_s}{R_{sh}}$$

Where J_0 is the dark saturation current density, n the ideality factor, k the Boltzmann constant, and T_{module} the module temperature.

For each hour of the year the J-V characteristics is calculated. In this regard, each point in time is characterized by the short-circuit current density (see Energy Yield Core Module) and the module temperature. The module temperature is derived by a simple empirical model based on the nominal operating cell temperature (NOCT), which can be set in the configuration. The ambient temperature is then scaled depending on the insolation S heating up the solar module.

$$T_{module} = T_{ambient} + \frac{\mathrm{NOCT} - 20°C}{800\mathrm{W/m^2}} \cdot S.$$

Different temperatures also influence the *V*OC and *J*SC of the solar cell. This influence is modelled by temperature coefficients (expressed in ppm K-1) for each parameter:

$$V_{OC} = V_{OC,0}\left(1 + \frac{t_{V_{OC}}}{10^6}(T - T_0)\right)$$

$$J_{SC} = J_{SC,0}\left(1 + \frac{t_{J_{SC}}}{10^6}(T - T_0)\right)$$

Solving the transcendental one-diode equation for the open-circuit voltage *V*OC, one can use the Lambert *W* function:

$$V_{OC,0} = (J_{SC} + J_0)R_{sh} - nV_{th}W\left(\frac{J_0R_{sh}}{nV_{th}}\exp\left(\frac{(J_{SC} + J_0)R_{sh}}{nV_{th}}\right)\right)$$

With the same approach, the voltage can be expressed depend on the current as follows:

$$V(J) = -JR_s + (J_{SC} + J_0 - J)R_{sh} - nV_{th}W\left(\frac{J_0R_{sh}}{nV_{th}}\exp\left(\frac{(J_{SC} + J_0 - J)R_{sh}}{nV_{th}}\right)\right) - V_{OC}^{RT} + V_{OC}\,.$$

Usage

The electrics module is called by the energy yield core module. Once the total short-circuit current density is calculated, the electrics module can be called. We can define the electrical parameters as described in the main.m example.

```
electrics.configuration = '2T';          % 2T, 3T, 4T, 2T exp, 3T exp, 4T exp
electrics.shunt = 'with';                % with, without
electrics.RshTandem = 1300;              % shunt resistance of tandem device
electrics.RsTandem = 3;                  % serial resistance of tandem device
electrics.Rsh = [1300, 1000];            % shunt resistance of n-th cell
electrics.Rs = [2, 1];                   % serial resistance of n-th cell
electrics.CE = [1, 1];                   % collection efficiency of n-th cell
electrics.j0 = [2.7e-18, 1e-12];         % reverse-blocking current of n-th cell
electrics.n = [1.1, 1];                  % ideality factor of n-th cell
electrics.Temp = [25, 25];               % temperature of cells (can also be n vectors)
electrics.NOCT = [48, 48];               % nominal temperature of n-th cell, if a number, Temp is o
verwritten
```

```
electrics.tcJsc = [0.0002, 0.00032];  % temperature coefficient of Jsc in K^-1 of n-th cell
electrics.tcVoc = [-0.002, -0.0041];  % temperature coefficient of Voc in K^-1 of n-th cell
```

For testing, we define the short-circuit current densities for the top peroskite and bottom silicon solar cell as follows:

```
Jsc_top = 19.5; % mAcm-2
Jsc_bot = 18.2; % mAcm-2
```

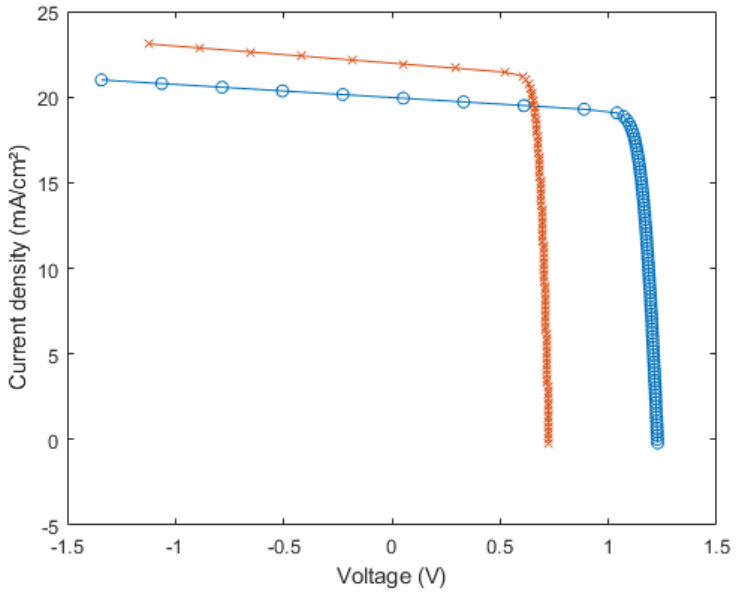
And then we call the electrics module:

```
[el.Voc_Tandem, el.FF_Tandem, el.Power_Tandem, el.JMPP_Tandem, el.VMPP_Tandem] = ...
    calctandemelectrics(electrics, min(Jsc_top,Jsc_bot) );
```

```
el =

  struct with fields:

      Voc_Tandem: 2.0072
       FF_Tandem: 0.7728
    Power_Tandem: 282.3224
     JMPP_Tandem: 16.9000
     VMPP_Tandem: 1.6705
```

We can also illustrate the single junction IV curves, by calling calcsingleelectrics().

```
shunt = 'with';
RshTandem = 1000;
RsTandem = 3;
Rsh = [1300, 1000];
Rs = [2, 1];
CE = [1, 1];
j0 = [2.7e-18, 1e-12];
n = [1.1, 1];
Temp = [25, 25];
NOCT = [48, 48];
tcJsc = [0.0002, 0.00032];
tcVoc = [-0.002, -0.0041];

jsc_RT = [19.5, 18.2];

[~, ~, ~, ~, ~, j_top, V_top] = calcsingleelectrics(jsc_RT(1), Rs(1), Rsh(1), j0(1), n(1), tcJsc(1),
tcVoc(1), Temp(1), shunt);
[~, ~, ~, ~, ~, j_bot, V_bot] = calcsingleelectrics(jsc_RT(2), Rs(2), Rsh(2), j0(2), n(2), tcJsc(2),
tcVoc(2), Temp(2), shunt);

plot(V_top,j_top,'o-'); hold on
plot(V_bot,j_bot,'x-');
xlabel('Voltage (V)');
ylabel('Current density (mA/cm^2)')
```

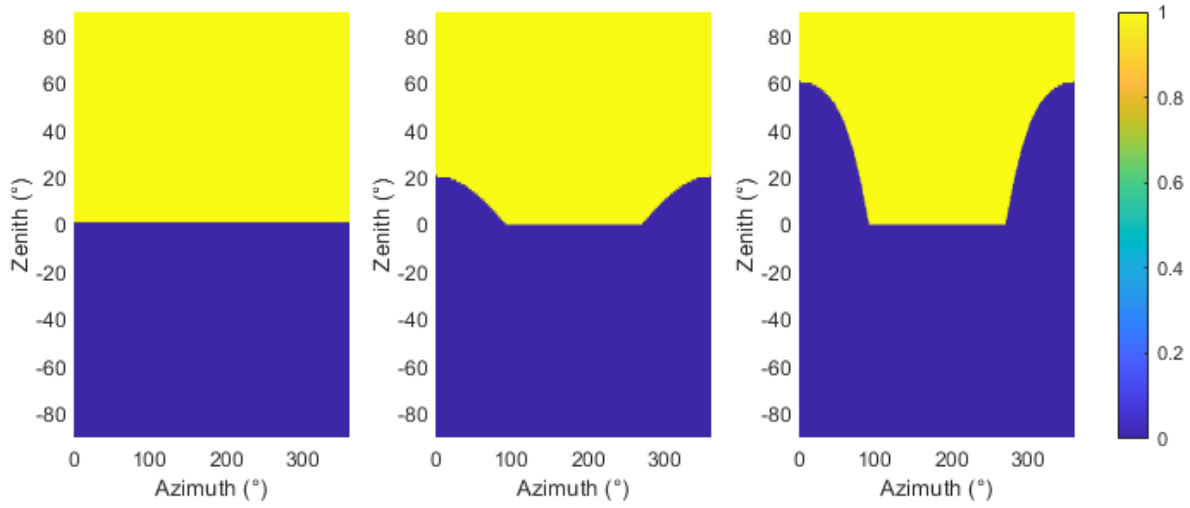

Current density (mA/cm²)
Voltage (V)
25
20
15
10
5
0
-5
-1.5
-1
-0.5
0
0.5
1
1.5

Energy Yield Core Module

The energy yield core module calculates the energy yield (EY) over the course of one year of the sub-cells depending on their orientation (rotation and/or tilt of the module) and location. The EY is computed by combining the spectral and angular resolved solar irradiation (with or without albedo), the absorptance of the multi-junction solar cell and the electrical properties.

First, the spectral and temporal resolved irradiance data is extracted from the irradiance module. Furthermore, the ambient temperature, the sun's azimuth and elevation angles are loaded from the TMY3 data within the irradiance dataset.

Next, a matrix is defined, covering the allowed angles of incidence (spherical coordinates: theta, phi) on the solar cell in its local coordinate system. In a non-rotated solar module, which is lying flat on the ground, all elements in this matrix describing the upper hemisphere are one. Once the solar module is titled and/or rotated, distinct angles become zero. For example, a tilt angle of 0°, 20° and 60° is shown in the following figure:



Besides a static tilt, there is also the option to use different tracking algorithms. In case of tracking, the above matrix is calculated for each hour of the year dependent on the corresponding tilt and rotation angles. These angles are calculated by the various tracking algorithms (options: see below).

For each hour of the year, the short-circuit current-density is calculated for each absorber within the defined stack. The short-circuit current density is calculated separately for the direct and diffuse irradiance.

$$J_{\mathrm{SC}}^{\mathrm{dir}} = \frac{q}{hc}\int \chi(\lambda)\mathrm{A}(\lambda,\theta'_{\mathrm{sun}})I_{\mathrm{dir}}(\lambda)\lambda\cos(\theta'_{\mathrm{sun}})\,d\lambda$$
$$\text{with} \quad \theta'_{\mathrm{sun}} < 90°,\ \Gamma(\theta'_{\mathrm{sun}},\varphi'_{\mathrm{sun}}) = 1$$

$$J_{\mathrm{SC}}^{\mathrm{diff}} = \frac{q}{hc}\iiint \chi(\lambda)\mathrm{A}(\lambda,\theta'_{\mathrm{sun}})I_{\mathrm{diff}}(\lambda)\cos(\theta')\lambda\Gamma(\theta',\varphi')\sin(\theta')\,d\varphi'\,d\theta' d\lambda.$$

In case of a bifacial simulation and/or if albedo is enabled further contributions are taken into account. The total short-circuit current density is then given by the sum of all individual contributions:

$$J_{\mathrm{SC}}^{\mathrm{tot}} = J_{\mathrm{SC}}^{\mathrm{dir}} + J_{\mathrm{SC}}^{\mathrm{diff}} + J_{\mathrm{SC}}^{\mathrm{diff,albedo\text{-}front}} + J_{\mathrm{SC}}^{\mathrm{diff,albedo\text{-}back}} + J_{\mathrm{SC}}^{\mathrm{dir,back}} + J_{\mathrm{SC}}^{\mathrm{diff,back}}.$$

Usage

In order to use the energy yield core module, you need to load the irradiance data, calculate the optics of your stack and define the electrics. An example you can find in the main.m and in the Quick start guide.

Assuming everything is loaded and simulated, you can define the tilt and rotation angle of the solar cell.

```
SolarCellRotationAngle = 180;
SolarCellTiltAngle = 20;
```

Next, you define, if you want to use a tracking algorithm and if albedo should be considered.

```
% 0: disable tracking
% 1: 1-axis non-tilted east-west
% 2: 2-axis tracking
% 3: 1-axis latitude-titled zenith rotation
% 4: 1-axis latitude-tiled seesaw rotation; limitation: rotation needs to be = 180°
tracking = 0;

albedo = 0;
groundtype = 'artificialblack';
```

Finally, you can call the energy yield core module:

```
EY = EnergyYield(irradiance, optics, electrics, SolarCellRotationAngle, SolarCellTiltAngle, tra
cking, albedo, groundtype, PathEYResults, StoreInDatabase);
```

Note: irradiance, optics, and electrics are structures containing the results/definitions of/for the other modules.

The output of the calculation is another structure, here called EY. It contains the hourly resolved results for both absorbers, and some additional useful data! This includes the individual (JscDirect, JscDiffuse, JscAlbedo) and total (Jsc) short-circuit current densities. Next, it includes the electrical properties like open circuit voltage (Voc), fill factor (FF), current and voltage at the maximum power point (JMPP,VMPP), and the power. These quantities are calculated:

- for the multi-junction solar cell in the tandem configuration (*_Tandem),
- for each sub cell considered as a single cell (*_SJ),
- and for the sub cells in the tandem configuration (no appendix)

Moreover, the power conversion efficiency is calculated, which only makes sense, when the AM1.5G spectrum is loaded as irradiance data.

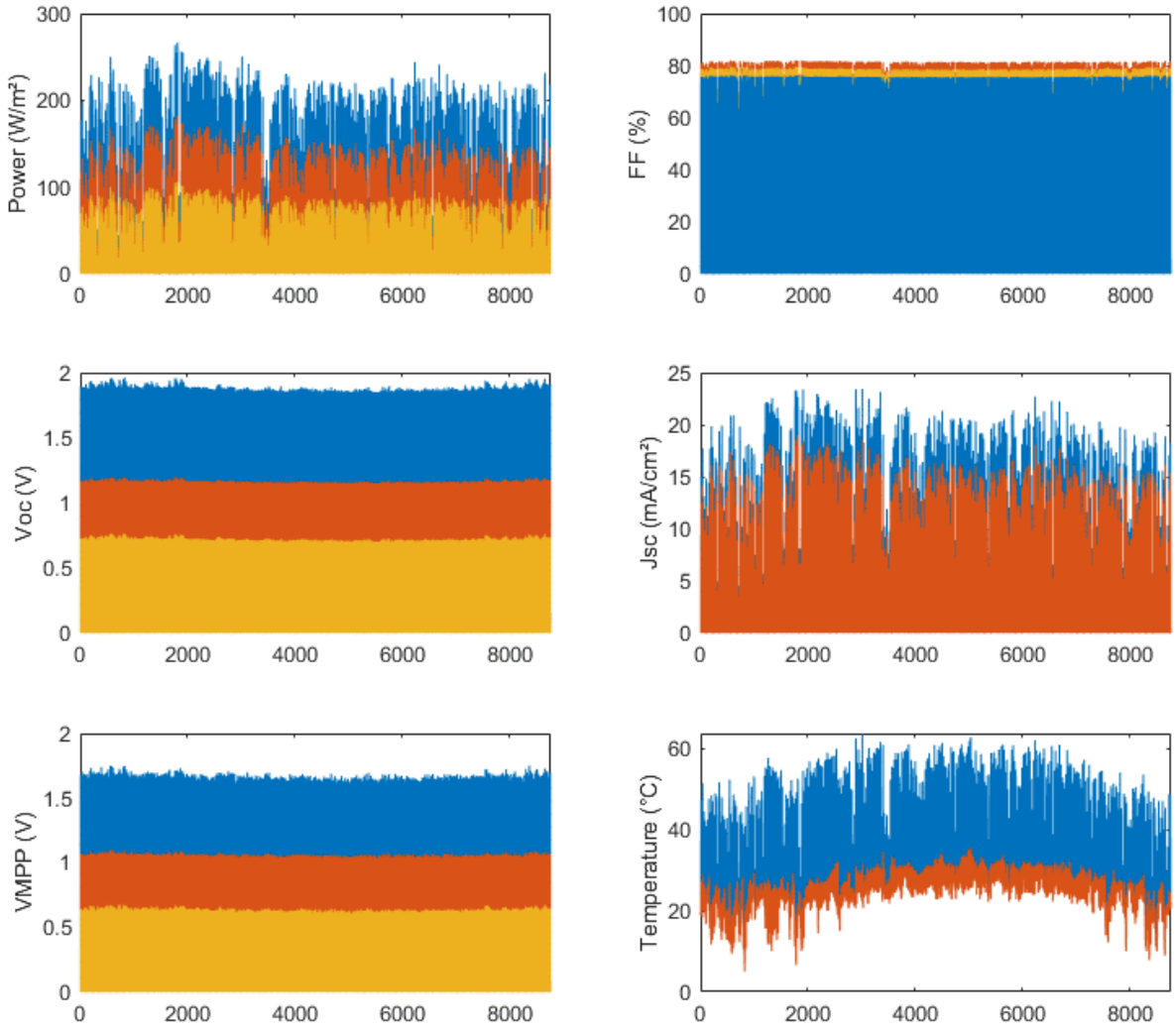
```
CodeLocation = 'AM1.5G';
AliasLocation = 'spectrum');
```

Finally, the irradiance data of the current location, the absorptance of the absorbers in the stack, the total power of the tandem are provided. The total power equals to the annual energy yield given in $kWhm^{-2}a^{-1}$.

```
>> EY

EY =

  struct with fields:

                JscDirect: [8760×2 double]
               JscDiffuse: [8760×2 double]
               JscAlbedo: [8760×2 double]
                     Jsc: [8760×2 double]
                  Voc_SJ: [8760×2 double]
                   FF_SJ: [8760×2 double]
                 JMPP_SJ: [8760×2 double]
                VMPP_SJ: [8760×2 double]
               Power_SJ: [8760×2 double]
                     Voc: [8760×2 double]
                      FF: [8760×2 double]
                    JMPP: [8760×2 double]
                    VMPP: [8760×2 double]
                   Power: [8760×2 double]
              Voc_Tandem: [8760×1 double]
               FF_Tandem: [8760×1 double]
             JMPP_Tandem: [8760×1 double]
             VMPP_Tandem: [8760×1 double]
            Power_Tandem: [8760×1 double]
             TempAmbient: [8760×1 double]
              TempModule: [8760×2 double]
               TandemPCE: [8760×1 double]
                     PCE: [8760×2 double]
            IrradianceDifH: [8760×1 double]
            IrradianceDifN: [8760×1 double]
            IrradianceDirH: [8760×1 double]
            IrradianceDirN: [8760×1 double]
                       A: {2×1 cell}
    TandemPowerTotal: 426.4966
       TandemPCEmean: 0.0806
              albedo: 0
```

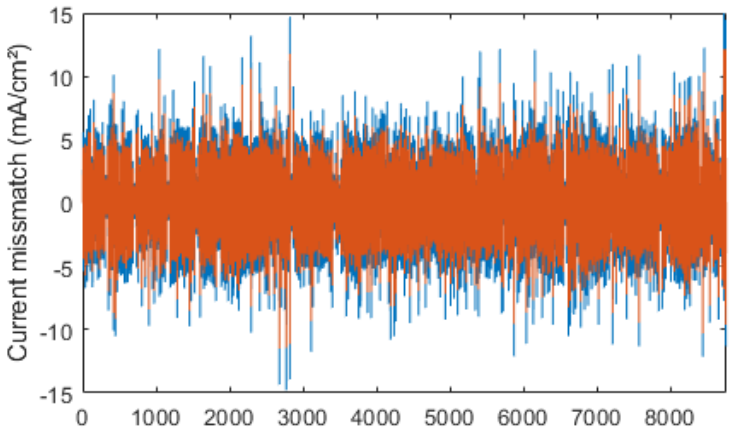
We can illustrate and compare some of the results:

```
figure;
subplot(3,2,1); plot(EY.Power_Tandem); hold on; plot(EY.Power);
subplot(3,2,2); plot(100*EY.FF_Tandem); hold on; plot(100*EY.FF);
subplot(3,2,3); plot(EY.Voc_Tandem); hold on; plot(EY.Voc);
subplot(3,2,4); plot(EY.Jsc);
subplot(3,2,5); plot(EY.VMPP_Tandem); hold on; plot(EY.VMPP);
subplot(3,2,6); plot(EY.TempModule(:,1)); hold on; plot(EY.TempAmbient);
```



Note: the data is available in hourly resolution. Summing up the tandem power over all 8760 h of the representative year, will lead to the annual energy yield. In this example, we get ~ 425.17 $kWhm^{-2}a^{-1}$.

In order to increase this energy yield, an obvious improvement would be to improve current matching between the two sub cells.



We can also search the optimal tilt angle for this architecture and location (Miami):

```
SolarCellRotationAngle = 180;
SolarCellTiltAngle = 0:5:50;
[EYaoi, TandemPowerTotal] = sweepEY(irradiance, optics, electrics, SolarCellRotationAngle,
SolarCellTiltAngle, tracking, albedo, groundtype, PathEYResults, StoreInDatabase);

figure;
plot(SolarCellTiltAngle,TandemPowerTotal);
xlabel('Tilt angle (°)')
ylabel('Anual Energy Yield (kWhm^{-2}a^{-1})')
```