

Supplementary Information

Solcore: A multi-scale, Python-based library for modelling solar cells and semiconductor materials

D. Alonso-Álvarez T. Wilson P. Pearce M. Führer D. Farrell

N. Ekins-Daukes

Department of Physics, Imperial College London, London, United Kingdom

March 30, 2018

1 Comparison of optical solvers

This section contains the complete code needed to reproduce Figure 9, using different optical models to calculate the quantum efficiency of a thin GaAs solar cell.

```
from solcore import si, material
from solcore.structure import Junction, Layer
from solcore.solar_cell import SolarCell
from solcore.solar_cell_solver import solar_cell_solver, default_options
from solcore.light_source import LightSource
import numpy as np
import matplotlib.pyplot as plt

# user options
T = 298
wl = si(np.linspace(400, 900, 80), 'nm')
light_source = LightSource(source_type='standard', version='AM1.5g', x=wl,
output_units='photon_flux_per_m', concentration=1)
opts = default_options
opts.wavelength, opts.no_back_reflexion, opts.size, opts.light_source, opts.
    T_ambient = \
wl, False, [400, 400], light_source, T
# The size of the unit cell for the RCWA structure is 400 x 400 nm

# Defining all the materials we need
Air = material('Air')(T=T)
p_GaAs = material('GaAs')(T=T, Na=si('4e18cm-3')) # for the GaAs cell
    emitter
n_GaAs = material('GaAs')(T=T, Nd=si('2e17cm-3')) # for the GaAs cell
    base
AlAs, GaAs = material('AlAs')(T=T), material('GaAs')(T=T) # for the DBR
SiO2 = material('SiO2', sopra=True)(T=T) # for the spacer layer
TiO2 = material('TiO2', sopra=True)(T=T) # for the nanoparticles

# some parameters for the QE solver
for mat in [n_GaAs, p_GaAs]:
    mat.hole_mobility, mat.electron_mobility, mat.permittivity = 3.4e-3, 5e-2, 9
    n_GaAs.hole_diffusion_length, p_GaAs.electron_diffusion_length = si("500nm"), si(
"5um")

# Define the different parts of the structure we will use
GaAs_junction = [Junction([Layer(width=si('100nm'), material=p_GaAs, role="emitter"
),
Layer(width=si('400nm'), material=n_GaAs, role="base")], T=T, kind='DA')]
DBR = 10 * [Layer(width=si("73nm"), material=AlAs), Layer(width=si("60nm"),
    material=GaAs)]
```

```

# this creates 10 repetitions of the AlAs and GaAs layers, to make the DBR
structure
NP_layer = [Layer(si('50nm'), Air, geometry=[{'type': 'circle', 'mat': TiO2, '
center': (200, 200),
'radius': 50}])]
substrate = [Layer(width=si('50um'), material=GaAs)]
spacer = [Layer(width=si('25nm'), material=SiO2)]

solar_cell = SolarCell(spacer + GaAs_junction + substrate)      # solar cell with
SiO2 coating
opts.optics_method = 'TMM'
solar_cell_solver(solar_cell, 'qe', opts)
TMM_EQE = solar_cell[1].eqe(opts.wavelength)
opts.optics_method = 'BL'
solar_cell_solver(solar_cell, 'qe', opts)
BL_EQE = solar_cell[1].eqe(opts.wavelength)

solar_cell = SolarCell(spacer + GaAs_junction + DBR + substrate) # as above, with
a DBR on the back
opts.optics_method = 'TMM'
solar_cell_solver(solar_cell, 'qe', opts)
TMM_EQE_DBR = solar_cell[1].eqe(opts.wavelength)

solar_cell = SolarCell(NP_layer + spacer + GaAs_junction + DBR + substrate)
# cell with TiO2 nanocylinder array on the front
opts.optics_method = 'RCWA'
opts.orders = 49          # number of diffraction orders to keep in the RCWA solver
solar_cell_solver(solar_cell, 'qe', opts)
RCWA_EQE_NP = solar_cell[2].eqe(opts.wavelength)
opts.optics_method = 'TMM'
solar_cell_solver(solar_cell, 'qe', opts)
TMM_EQE_NP = solar_cell[2].eqe(opts.wavelength)
opts.optics_method = 'BL'
solar_cell_solver(solar_cell, 'qe', opts)
BL_EQE_NP = solar_cell[2].eqe(opts.wavelength)

plt.figure()
plt.plot(wl*1e9, BL_EQE_NP, wl*1e9, TMM_EQE, wl*1e9, TMM_EQE_DBR, wl*1e9,
RCWA_EQE_NP)
plt.legend(labels=['Beer-Lambert law (all structures)', 'TMM (GaAs SC)', 'TMM (GaAs
SC + DBR)',
'RCWA (GaAs SC + NP array + DBR)'])
plt.xlabel("Wavelength (nm)")
plt.ylabel("Quantum efficiency")
plt.show()

a = np.asarray([opts.wavelength*1e9, BL_EQE, TMM_EQE, TMM_EQE_DBR, RCWA_EQE_NP])
np.savetxt("optcalc_results.csv", np.transpose(a), delimiter=",")

```

Listing 1: Script which defines a thin GaAs solar cell and calculates the QE performance (using the depletion approximation) of a bare cell, and the same cell with a distributed Bragg reflector (DBR) on the rear and a layer of TiO₂ nanocylinders on the front. The optical calculations are carried out using the three optical methods available: BL, TMM and RCWA, in order to compare their suitability for investigating different types of structures.

2 Example of a complete solar cell simulation

This tutorial is a step-by-step creation of a solar cell in Solcore and the calculation of their properties using Solcore. The solar cell will include:

- Dual junction GaInP/GaAs solar cell, lattice matched to GaAs.
- The bottom cell has 30 strained-balanced quantum wells (QW), made of GaAsP/InGaAs.

- There is a tunnel junction in between the subcells.
- There is a dual layer anti-reflecting coating on the front made of MgF-ZnS

2.1 Defining the structure

First we need to create the solar cell structure. It is made of several bits and pieces: the QWs, 2xJunctions, 1x tunnel junction and the ARC. We start with the first.

2.1.1 Defining the QWs

```
from solcore import material
from solcore.structure import Layer
import solcore.poisson_drift_diffusion as PDD
import numpy as np

T = 300
wl = np.linspace(350, 1050, 1001) * 1e-9

# First, we create the materials of the QW
QWmat = material('InGaAs')(T=T, In=0.2, strained=True)
Bmat = material('GaAsP')(T=T, P=0.1, strained=True)
i_GaAs = material('GaAs')(T=T)

# The QW is 7 nm wide, with GaAs interlayers 2 nm thick at each side and GaAsP
# barriers 10 nm thick.
# The final device will have 30 of these QWs.
QW = PDD.CreateDeviceStructure('QW', T=T, repeat=30, substrate=i_GaAs, layers=[
    Layer(width=10e-9, material=Bmat, role="barrier"),
    Layer(width=2e-9, material=i_GaAs, role="interlayer"),
    Layer(width=7e-9, material=QWmat, role="well"),
    Layer(width=2e-9, material=i_GaAs, role="interlayer"),
    Layer(width=10e-9, material=Bmat, role="barrier") ])

# We solve the quantum properties of the QW, leaving the default values of all
# parameters
QW_list = PDD.SolveQWproperties(QW, wavelengths=wl)
```

Listing 2: Initial definitions of the QW structure

The first few lines in Listing 2 import the Solcore utilities needed to define the structure, including the *material* function, the *Layer* class and the Poisson-Drift-Diffusion solver. After that, we create the materials that will made the QWs and create a *Device structure*. We will use these structure just to be able to solve the QWs afterwards and transform it into a sequence of layers with effective properties that the PDD solver understands. The call to *CreateDeviceStructure* has several inputs, including the temperature, the substrate, the number of repetition of the QWs and the structure of the layers. The call to *SolveQWproperties* will, indeed, use the utilities within the quantum mechanics module to calculate the band structure of the QWs, their absorption coefficient and, finally, will calculate effective bandgap, density of states, etc. that the PDD solver will use. Although the device will have 30 quantum wells, only one unit (the one indicated in *Layers*) will be modelled as an isolated QW.

While Solcore can solve the Schrdinger equation in a structure with any number of layers, the absorption calculator for QWs can only deal properly with single QWs. That is the reason of modelling only 1 QW despite having 30 in the structure. This will clearly represent a limitation when modelling the absorption of superlattices, where there is a strong coupling between neighbouring QWs.

In the code above, we have used the *PDD.SolveQWproperties* function with the default values, but it can have a number of optional input parameters to define the number of confined states to calculate, the energy of quasiconfined states, electric field, boundary conditions, etc.

2.1.2 Defining the junctions

In order to calculate the properties of a solar junction using the PDD solver, we need to give all the layers and materials the junction is made of, in a similar way we have done for the QWs. One thing

to note is that if Solcore cannot find a property it needs to solve the PDD equations, it will take the corresponding property for GaAs as a default value.

The different code snippets are additive in order to get a final, complete script. Normally, all the import statements would be packed together at the beginning.

```
from solcore.structure import Junction

## Materials for the BOTTOM junction
window_bottom = material('GaInP')(T=T, Nd=5e24, In=0.49)
n_GaAs = material('GaAs')(T=T, Nd=1e24)
p_GaAs = material('GaAs')(T=T, Na=8e22)
bsf_bottom = material('GaInP')(T=T, Na=5e24, In=0.49)

GaAs_junction = Junction([Layer(width=10e-9, material=window_bottom, role="Window"),
                           Layer(width=150e-9, material=n_GaAs, role="Emitter")] +
                           QW_list +
                           [Layer(width=2000e-9, material=p_GaAs, role="Base"),
                            Layer(width=200e-9, material=bsf_bottom, role="BSF")], sn=1
                           e6, sp=1e6, T=T, kind='PDD')

## Materials for the TOP junction
window_top = material('AlInP')(T=T, Nd=5e23, Al=0.53)
n_GaInP = material('GaInP')(T=T, Nd=5e23, In=0.49)
p_GaInP = material('GaInP')(T=T, Na=8e22, In=0.49)
bsf_top = material('AlInP')(T=T, Na=5e24, Al=0.53)

GaInP_junction = Junction([Layer(width=120e-9, material=n_GaInP, role="Emitter"),
                             Layer(width=800e-9, material=p_GaInP, role="Base")], sn=1
                             e3, sp=1e3, T=T, kind='PDD')
```

Listing 3: Definition of the junctions

As it can be seen in Listing 3, while we have defined the window and back surface field layer (BSF) for the TOP junction, we have not included it into the *Junction* definition. The reason for this is that very wide bandgap materials cause convergence problems when doing calculations under illumination, specially when working as the front-most window layers or in distributed Bragg reflectors, where there are very abrupt changes of properties in very short distances. In order to account for their presence, two things are done: (1) the surface recombination velocity of the top junction is set to a low value to mimic the passivating effect of the window and BSF layers, and (2) the missing layers are added outside the Junction object when creating the full solar cell in order to consider their optical properties (see below).

The only tunnel junction of this solar cell will be defined according to the parametric model and we will assume it is made of GaInP layers, 40 nm-thick in total, that will block part of the light reaching the bottom junction. The code is in Listing 4. Since the top junction is also made of GaInP, most of the light should already be absorbed and therefore it should not represent a very important loss. We will use a relatively low peak current to demonstrate the effect of tunnel junction breakdown when working at high concentration.

```
from solcore.structure import TunnelJunction

tunnel = TunnelJunction([Layer(width=40e-9, material=n_GaInP, role="TJ")],
                        v_peak=0.2, j_peak=7.5e4, v_valley=1, j_valley=4e4, prefactor
                        =5, j01=1e-23, kind='parametric', pn=True)
```

Listing 4: Definition of the tunnel junction

2.1.3 Defining the AR coating

The AR coating will reduce the front surface reflection and, therefore, increase the photocurrent of the solar cell. We use a simple dual layer coating made of MgF2 and ZnS. Both materials are available in the SOPRA database of optical constants ("MgF2" and "ZnScub", respectively). The data for MgF2 only extends to 900 nm, but we will assume its optical properties will be similar at longer wavelengths,

extrapolating the available data. The code simply consists on importing those materials, as shown in Listing 5.

```
MgF2 = material('MgF2')()
ZnS = material('ZnScub')()
```

Listing 5: Definition of the anti reflecting coating

2.1.4 Creating the solar cell

With all the materials and structures defined, we just need to put everything together, including the front window layer and the BSF layer of the top junction that we had left outside (Listing 6).

```
from solcore.solar_cell import SolarCell

my_solar_cell = SolarCell([Layer(width=110e-9, material=MgF2, role="ARC1"),
                             Layer(width=60e-9, material=ZnS, role="ARC2"),
                             Layer(width=30e-9, material=window_top, role="window"),
                             GaInP_junction,
                             Layer(width=100e-9, material=bsf_top, role="BSF"),
                             tunnel,
                             GaAs_junction],
                           T=T, substrate=n_GaAs)
```

Listing 6: Creation of the MJ solar cell

2.2 Calculating the external quantum efficiency

With the structure fully defined, now it is possible to calculate its behavior as solar cell. We first calculate the external quantum efficiency at 1 Sun of intensity using the AM1.5g solar spectrum. The code that defines the spectrum to use and launch the calculation will be:

```
from solcore.solar_cell_solver import solar_cell_solver
from solcore.light_source import LightSource

light_source = LightSource(source_type='standard', version='AM1.5g', x=wl,
                           output_units='photon_flux_per_m', concentration=1)

solar_cell_solver(my_solar_cell, 'qe',
                  user_options={'light_source': light_source, 'wavelength': wl, '
                                optics_method': 'TMM'})
```

Listing 7: Calculation of the EQE

The calculation will proceed in several stages: (1) The structure is processed and discretized, (2) the problem is solved under equilibrium, (2) the problem is solved under short circuit conditions increasing in small steps the intensity of the illumination, and (3), the quantum efficiency is calculated, one wavelength at a time. This is done for the two junctions. Figure 1 shows the electrons and holes densities at equilibrium (dashed line) and at short circuit (continuous line), as well as the EQE of the two junctions and the total absorption in the layers.

As it can be seen, the minority carrier population increases significantly under illumination, specially in the QW region of the bottom cell, which is a relatively thick, undoped region. The EQE of the bottom junction, in comparison with the total absorbed light, indicates excellent carrier collection with only some missing carriers in the region of the spectrum where light is absorbed by the QWs. The top junction is not performing that well, the reason being a combination of low mobility of carriers in GaInP (between 3 and 6 times smaller than that of p-GaAs), insufficient thickness to absorb all light (part of it reaching the GaAs junction), parasitic absorption of the AlGaP window layer and surface recombination.

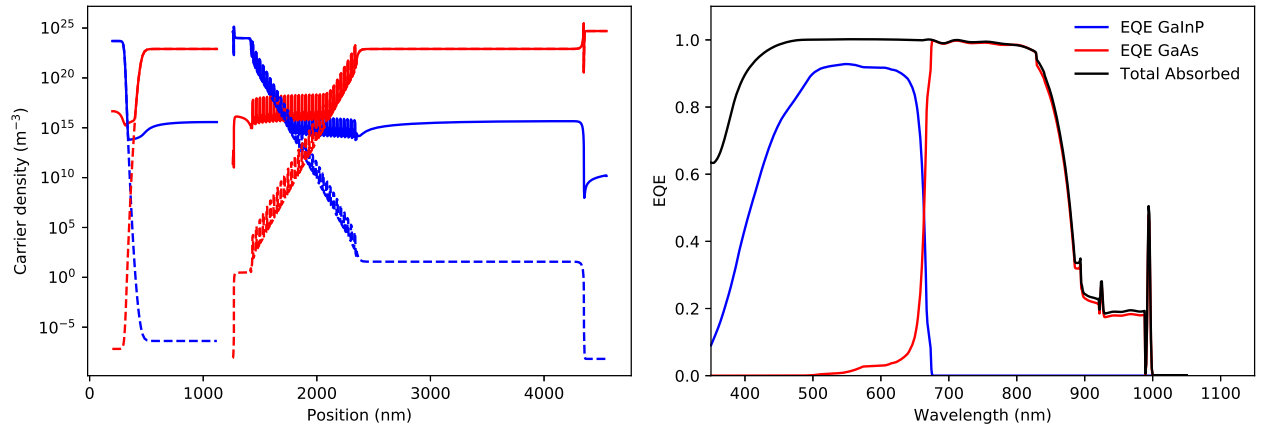


Figure 1: (a) Carrier densities in equilibrium (dashed lines) and at short circuit (continuous lines) as a function of the position inside the cell. The left part is the GaInP junction, separated from the GaAs subcell on the right by a gap left by the tunnel junction. (b) EQE of the two junctions and the total absorption in the layers.

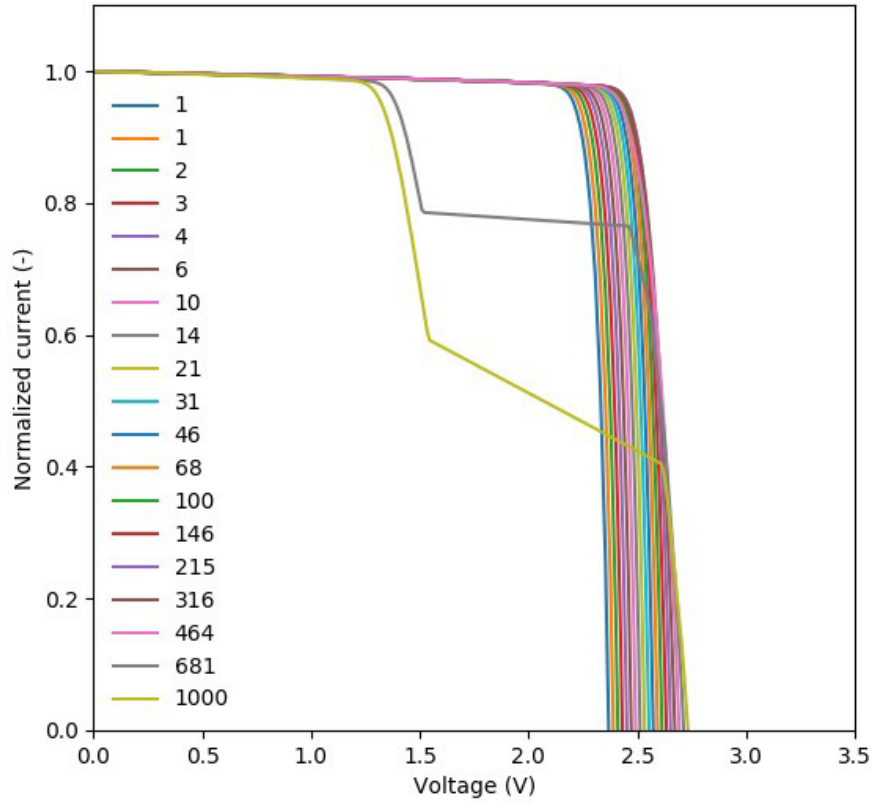


Figure 2: All IV curves normalized by their corresponding I_{SC} .

2.3 Calculating the IV characteristics

The calculation of the current-voltage characteristics proceed in a similar way. In this case, care must be taken to use the correct polarity for the (internal) voltage sweep, and the polarity of the tunnel junction, already defined above as *pn*. As the optical properties have already been calculated, we set the optics method to None, so they are not calculated again. Listing 8 shows the definition of the voltages, the sunlight concentration factor (from 1 to 1000 suns) and the execution of the calculation within a for loop.

Note that this example will take several hours to run due to the presence of QWs, which increase the number of mesh points in the bottom cell to several thousand. To have an example of the results faster, reduce the number of concentrations (19 in the example) and comment the QWs related line in the definition of the GaAs junction in Listing 3.

```
con = np.logspace(0, 3, 19)
vint = np.linspace(-3.5, 4, 600)
V = np.linspace(-3.5, 0, 300)

allI = []
isc = []
voc = []
FF = []
pmpp = []

for c in con:
    light_source.options['concentration'] = c

    solar_cell_solver(my_solar_cell, 'iv',
                      user_options={'light_source': light_source, 'wavelength': wl, '
optics_method': None,
                                'light_iv': True, 'mpp': True, 'voltages': V, '
internal_voltages': vint})

    isc.append(my_solar_cell.iv['Isc'])
    voc.append(my_solar_cell.iv['Voc'])
    FF.append(my_solar_cell.iv['FF'])
    pmpp.append(my_solar_cell.iv['Pmpp'])
    allI.append(my_solar_cell.iv['IV'][1])
```

Listing 8: Calculation of the IV curves

Figure 3 shows the summary of the efficiency, I_{SC} , V_{OC} and fill factor (FF) versus concentration. The efficiency of the cell is just above 30% at 1 sun, and increases with concentration, peaking at around 200-300 suns. Further increasing the concentration rapidly decreases the efficiency. Both, I_{SC} and V_{OC} follow the expected linear trend in log-log and semilog plots, respectively, while the fill factor increases for low concentrations and then continuously drops from 20 suns and, specially, from 200 suns. The initial smooth drop of the fill factor can be related with a limited carrier transport across the QW region, that under concentration, suffers from charge accumulation and damping of the electric field. However, the abrupt drop at higher concentrations is related to the failure of the tunnel junction which, by design, had a low peak current.

This effect can be better seen in Figure 2 where, for the two highest concentrations, the IV curve exhibit steps due to the tunnel junction limiting the current.

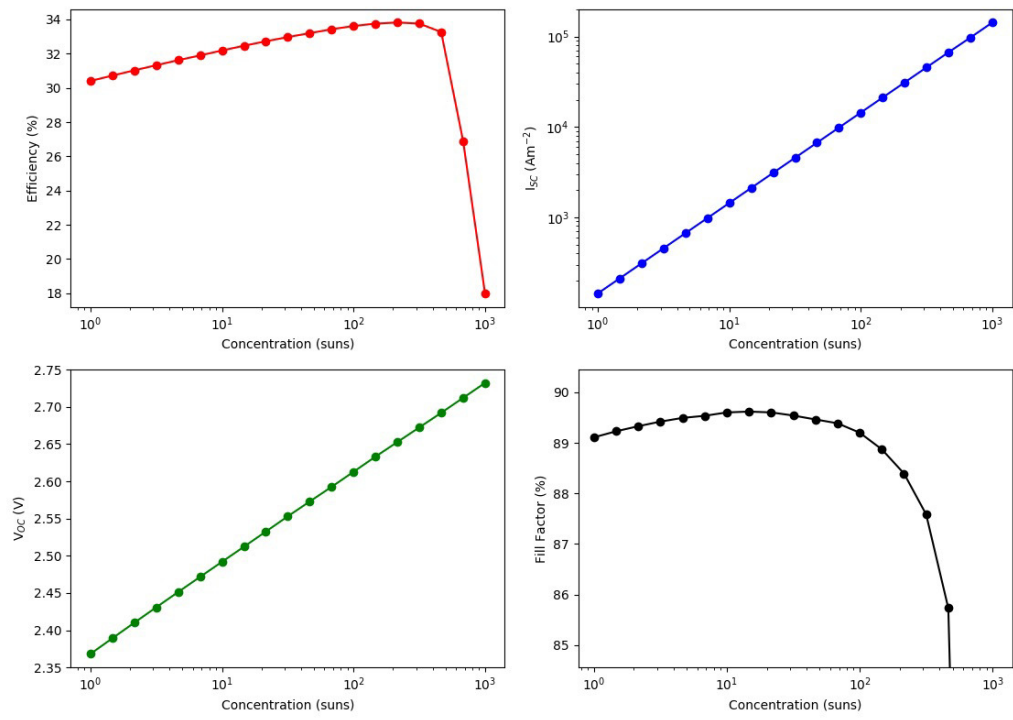


Figure 3: Summary of the efficiency, I_{sc} , V_{oc} and fill factor versus concentration.