SIMsalabim project
Manual
version 4.45

M. Koopmans
L.J.A. Koster
January 12, 2023

Contents

Chapter 1

Introduction

This project consists of two drift-diffusion simulation codes that are largely based on the same code but do have their own use: For steady-state simulations, including ions in stabilized scans, `SimSS`[1] is most suited. One can easily define a voltage range that should be used, a light intensity, etc. and it will compute a current-voltage characteristic. `SimSS` can also extract the typical performance characteristics when simulating a solar cell, or compare a computed current-voltage characteristic with an experimental one and show the error. Simulations are typically fast.

For transient simulations, `ZimT`,[2] German for cinnamon, is much more suited and it can also do more simulations, including steady-state ones, transient photovoltage, impedance spectroscopy, etc. Much of the functionality that `SimSS` has is also included in `ZimT`. However, `ZimT` reads-in a list of times, voltages and generation rates supplied by the user (see section 3.5) and then computes the resulting current density and voltage. For example, by supplying an AC voltage of varying frequency, `ZimT` can be used to simulate an impedance spectroscopy experiment. Alternatively, one can switch the light intensity from on to off and monitor how the current develops over time, just like a transient photocurrent experiment.

1.1 Why open-source?

We have been developing and using this code for a long time. At some point we decided to make it open-source in order to achieve three objectives: Firstly, to increase the number of users of the code. This can also be achieved

[1] Simulates Steady-State
[2] Zimulates Transients

by making it free-ware. However, in order to persuade others to use and thus trust the code, it is helpful that anyone can see the code. This is our second objective: transparency. All too often is the description of a numerical simulation in an academic publication no more than a few lines in the methods section. However, details matter and it should be clear what is actually being calculated in order to appreciate—or replicate!—the results. Lastly, at some point, there may even be other researchers who would like to contribute to the code itself.

Open-source does not mean that anything goes: This project is licensed under the GNU Lesser General Public License as stipulated in the license files. When in doubt, check the license.

Chapter 2

What is currently included?

`SIMsalabim` does not claim to capture *all* the physics of any specific semiconductor device. It is the sole responsibility of the user to assess whether this simulation is applicable or not. Thus, it is crucial to understand what is included and what is not. The *current* version of `SIMsalabim` includes the following:[1]

- Up to three layers can be defined
- Mobile ionic species
- Doping
- Contacts are defined by their work function. The surface recombination velocities can be either finite or infinite.
- Trapping, multiple trap levels can be defined.
- Interfaces:
  - offsets in conduction and valence bands
  - differences in effective density-of-states
  - differences in dielectric constant
  - trapping and recombination
- Generation of free charges:
  - from an optical absorption profile or uniform

[1] Future versions may include/refine/modify the implemented physics.

  - either direct or Onsager-Braun

- Recombination processes:
  - direct, band-to-band recombination
  - trap-assisted recombination
  - interface recombination
  - geminate recombination
- Series and shunt resistances
- Tracking of open-circuit voltage (`ZimT`)

2.1 Basic equations

In this section, we will briefly outline what `SimSS` and `ZimT` calculate and why. We will not put a lot of detail, but rather, we will include literature references for the interested reader.

Figure 2.1 shows the basic band diagram used throughout. The device may or may not have all the elements shown, depending on what the user specifies. `SimSS` and `ZimT` solve the drift-diffusion equations in the layer(s) between the electrodes.

2.1.1 The Potential

First of all, the Poisson equation relates the potential $V(x)$ to the charge density

$$\frac{\partial}{\partial x}\left(\varepsilon(x)\frac{\partial V(x)}{\partial x}\right) = q(n(x) - p(x) + C(x)), \tag{2.1}$$

where $\varepsilon(x)$ is the dielectric constant, n and p are the density of free electrons and holes. $C(x)$ denotes any other charges that may be present, such as doping, ions, trapped charge carriers. In detail, the term $C(x)$ follows from

$$\begin{aligned} C(x) =& p_0(x) - n_0(x) + n_{\text{ion}}(x) - p_{\text{ion}}(x)) \\ &+ \sum_{j=1}^{M}(s^e_{tb,j}(x) - f_{tb,j}(x))N_{tb,j}(x) + (s^e_{ti,j}(x) - f_{ti,j}(x))N_{ti,j}(x), \end{aligned} \tag{2.2}$$

where n_0 and p_0 are n- and p-doping densities (see `n_0` and `p_0`), n_{ion} and p_{ion} are the negative and positive ion densities. In this expression, we sum

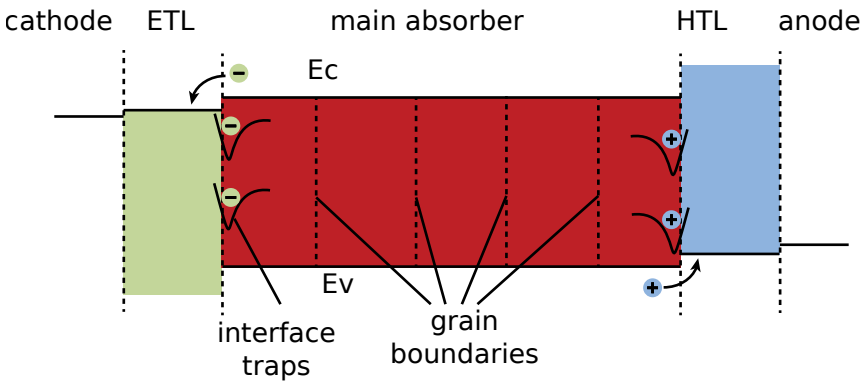

Figure 2.1: Schematic band diagram showing the main absorber, electron and hole transport layers, cathode and anode, interface traps, and grain boundaries.

over the different trap levels $j = 1 \ldots M$, where M is the number of trap levels. For each trap level, $s^e_{ti,j}$ and $s^e_{tb,j}$ is the charge-type of empty (no electron in the trap) interface / bulk traps and can have value 1 or 0, $f_{tb,j}$ and $f_{ti,j}$ are the fraction of filled bulk traps and interface traps in steady-state as defined in Refs 2 and 3, and $N_{tb,j}$ and $N_{ti,j}$ are the densities of bulk traps and interface traps. For more details on how $f_{tb,j}$ and $f_{ti,j}$ are calculated, see section 2.5.

2.1.2 The Drift-Diffusion Equations

The current continuity equations relate the electron (hole) current density $J_{n(p)}(x)$ to the generation ($G(x)$) and recombination ($R(x)$) rates. In steady-state (`SimSS`), one has

$$\frac{\partial J_n(x)}{\partial x} = -\frac{\partial J_p(x)}{\partial x} = -q(G(x) - R(x)), \tag{2.3}$$

whereas `ZimT` solves the transient equations

$$\frac{\partial n(x)}{\partial t} - \frac{1}{q}\frac{\partial J_n(x)}{\partial x} = G(x) - R(x), \tag{2.4}$$

and

$$\frac{\partial p(x)}{\partial t} + \frac{1}{q}\frac{\partial J_p(x)}{\partial x} = G(x) - R(x). \tag{2.5}$$

The flow of current is driven by gradients in the carrier densities and the potential as described by the drift-diffusion equations. Provided there

are no heterojunctions—so the conduction and valence band edges and the effective densities of states remain constant—then one has for electrons

$$J_n(x) = -qn(x)\mu_n(x)\frac{\partial V(x)}{\partial x} + kT\mu_n(x)\frac{\partial n(x)}{\partial x}, \tag{2.6}$$

and for holes, one has

$$J_p(x) = -qp(x)\mu_p(x)\frac{\partial V(x)}{\partial x} - kT\mu_p(x)\frac{\partial p(x)}{\partial x}. \tag{2.7}$$

In the case of a heterojunction, then the accompanying changes in the band edges and the effective densities of states modify the drift-diffusion equations. This can be accounted for by replacing the potential $V(x)$ in Eqs (2.6) and (2.7) with generalised potentials $V_{gn}(x)$ and $V_{gp}(x)$ for electrons and holes, respectively. This is detailed in section 2.3.

The total current density is then given by

$$J(x) = J_n(x) + J_p(x) + J_D(x) + J_{\text{nion}}(x) + J_{\text{pion}}(x), \tag{2.8}$$

where $J_D(x)$ is the displacement current, and $J_{\text{n(p)ion}}(x)$ are the negative (positive) ion currents. In steady-state, the last three terms in Eq. (2.8) are zero.[2] The movement of ions is treated in section 2.6.

Both `SimSS` and `ZimT` iteratively solve a set of discretised equations: the Poisson equation and the continuity equations are discretised and solved iteratively. These discretised equations can be found in Ref. 1: In the transient case, these take the form of Eqs (6.1-72, 73, and 74), where the transient equations correspond to Eqs (6.4-32) and (6.4.33)[3] in Ref. 1). In order to improve the convergence behaviour, these equations are linearised, see Ref. 3 for further details.

The discretization of the drift-diffusion equations requires the use of the Bernoulli function,[1]

$$B(x) = \frac{x}{e^x - 1}. \tag{2.9}$$

This function is used in every grid point and in every loop, so a fast implementation of this function is of the essence. Moreover, simply using the function as defined in Eq. (2.9) creates a problem if $x = 0$. To avoid these issues, we use Taylor expansions around $x = 0$ and only use the full expression for very large absolute x—which is quite rare. Figure 2.2 shows the exact and approximated Bernoulli function and the error of this approximation. As can be seen, the absolute error is smaller than 3.5×10^{-4}, which translates into a relative error of less than 8×10^{-4}.

[2] We assume that ions cannot enter or leave the device.

[3] This equation in Ref. 1 contains a typo: in the right-hand-side of the equation, $n_{i,j,m}$ should be $p_{i,j,m}$.

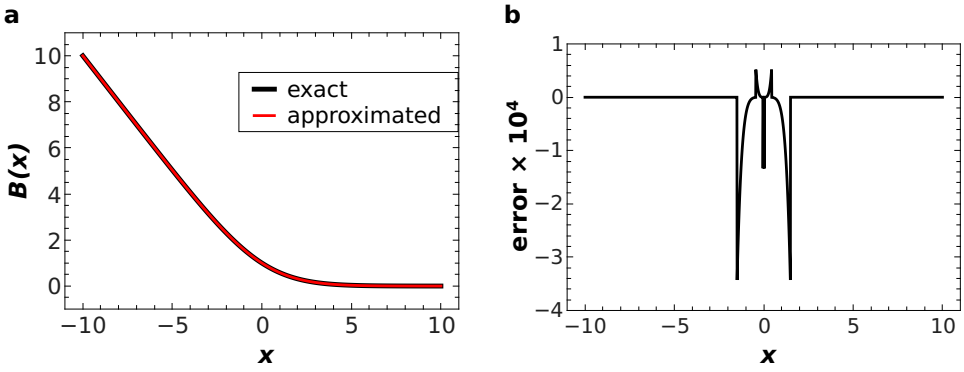

Figure 2.2: The (a) approximated and exact Bernoulli functions, and (b) the error.

2.2 Boundary conditions

The boundary condition on the potential V is given by

$$qV_R - qV_L = \texttt{W\_L} - \texttt{W\_R} + q\texttt{Vint}, \tag{2.10}$$

where $V_{R(L)}$ is the potential at the right (left) electrode, `W_R(L)` is the work function of the right (left) electrode, and `Vint` is the internally applied voltage (see section 2.7).

The boundary conditions on the carrier densities can take different forms, depending on the surface recombination velocities: If the surface recombination velocities (see `Sn/p_L/R`) are infinitely large, we simply assume that the carrier densities at the contacts follow from the difference between the local conduction or valence band edge and the work function of the electrodes. At the right electrode (anode), we have for electrons

$$n_R = n_{eq} = \texttt{Nc} \exp\left(\frac{\texttt{CB} - \texttt{W\_R}}{kT}\right), \tag{2.11}$$

where n_{eq} is the equilibrium electron density at this contact. If the surface recombination velocity is finite, then we use

$$J_n = q\texttt{Sn\_R}(n_R - n_{eq}), \tag{2.12}$$

but only if the electrons are extracted at this electrode ($J_n < 0$). If electrons are injected, we use Eq. (2.11). For holes at the right electrode (anode), one has

$$p_R = p_{eq} = \texttt{Nc} \exp\left(-\frac{\texttt{VB} - \texttt{W\_R}}{kT}\right), \tag{2.13}$$

where p_{eq} is the equilibrium hole density at this contact. Again, if the surface recombination velocity is finite, then we use

$$J_p = q\texttt{Sp\_R}(p_R - p_{eq}), \tag{2.14}$$

but only if the holes are extracted at this electrode ($J_p < 0$). If holes are injected, we use Eq. (2.13). The densities at the left electrode (cathode) are similar.

2.3 Internal interfaces

The internal interfaces between the transport layers (if any) and the main absorber layer modify the drift-diffusion equations. In order to account for differences in the conduction/valence band levels and the effective densities of states, one can replace the potential V in Eqs (2.6) and (2.7) with the generalised potentials V_{gn} and V_{gp} for electrons and holes, respectively.

If the conduction/valence band edge ($E_{c/v}(x)$) differs across the layers in the simulation volume, then the generalised potentials are defined as[4,5]

$$V_{gn}(x) = V(x) - \frac{\Delta E_c}{q}, \tag{2.15}$$

and

$$V_{gp}(x) = V(x) - \frac{\Delta E_v}{q}, \tag{2.16}$$

where $\Delta E_{c/v}$ is the change in conduction/valence band offset.

To account for changes in the effective densities of states,[4] these generalised potentials are modified by the introduction of Γ as in Eqs (14) and (15) in Ref. 6. By taking the effective density of states in the main absorber layer (N_c) as a reference,[7] we have

$$\Gamma = \frac{kT}{q} \ln \frac{N_c(x)}{N_c}. \tag{2.17}$$

This changes to generalised potentials to

$$V_{gn}(x) = V(x) - \frac{\Delta E_c}{q} + \Gamma, \tag{2.18}$$

and

$$V_{gp}(x) = V(x) - \frac{\Delta E_v}{q} - \Gamma. \tag{2.19}$$

[4]We take the effective density of states of the conduction band equal to that of the valence band.

The charge carrier mobility across such interfaces also differs from the bulk values. It is set by parameter `nu_int_LTL`, which has the units of velocity but is converted into a local mobility such that the current across the interface does not depend on the grid spacing (which would be unphysical).

2.4 Generation

`SimSS` and `ZimT` equate the generation rate of free electrons and holes either to the generation rate of electron-hole pairs (parameter `Gehp`)–common in non-excitonic materials—or they consider that such electron-hole pairs may be subject to geminate recombination losses.[8–10] If such geminate losses are taken into account (`Field_dep_G`=1), then the generation rate is reduced by a factor $P(x)$, given by

$$P(x) = \mathtt{P0} + (1 - \mathtt{P0})p(\mathtt{a}, T, F), \tag{2.20}$$

where $p(a, T, F)$ is the Onsager-Braun dissociation probability,[10] `a` is the charge separation distance, and `P0` is the fraction of electron-hole pairs that directly yield free carriers. The dissociation probability $p(a, T, F)$ can be integrated over a distribution of electron-hole pair distances, i.e.,

$$P(a, T, F) = \int_0^\infty p(y, T, F)g(a, y)dy, \tag{2.21}$$

where $g(a, y)$ is a normalized distribution function that can be specified by parameter `ThermLengDist`.

2.5 Trapping and Recombination

By their very nature, traps capture charge carriers. It is critical to understand their effects on a device in order to judge their relevance. Trapped charge carriers contribute to the space charge (see Eq. 2.1), without contributing to charge transport. Additionally, trap levels introduce another recombination pathway. Both effects are typically bad for device performance.

In transient simulations, things get a little bit more complicated as the trapping and/or detrapping can take rather long and, therefore, it might take a long time (as compared to the simulated time) before equilibrium is reached. This is especially relevant to cases where there is a distribution of traps, giving rise to multiple trapping/detrapping times.

In sum, traps can have a large impact on the simulations, depending on what is simulated and in what regime.

The non-geminate recombination of electrons and holes can proceed via different mechanisms: direct or band-to-band recombination; trap-assisted or Shockley-Read-Hall (SRH) recombination; and surface recombination.[5] Surface recombination is described in section 2.2.

2.5.1 Direct Recombination

The recombination rate of direct recombination rate is given by

$$R_{\text{direct}} = \gamma(np - n_i^2), \tag{2.22}$$

where n_i is the intrinsic carrier density, and γ is the direct recombination rate constant.

The rate constant γ can be set via different routes: It can be based on the Langevin expression (`UseLangevin`=1), in which case:

$$\gamma = \texttt{Lang\_pre}\frac{q}{\varepsilon(x)}(\mu_n(x) + \mu_p(x)), \tag{2.23}$$

where the mobilities $\mu_{n,p}$ and relative dielectric constant ε can depend on position.[6] The Langevin prefactor `Lang_pre` is motivated by the observation that the recombination rate can be (substantially) smaller than the Langevin value in organic solar cells, while the Langevin expression itself serves as an upper limit to the rate in pristine organic semiconductors. Alternatively, the rate constant can be defined directly via `kdirect`, if `UseLangevin`=0.

If the Onsager-Braun model for geminate recombination is used (see section 2.4), then the direct recombination rate is reduced by a factor of $1 - P(a, T, F)$.[9]

2.5.2 Bulk SRH Recombination

For simplicity and notational convenience, we show the recombination expressions per energy level, i.e. this is what is used if there is but a single trap level. If there are multiple trap levels, then the total SRH recombination rate is simply the sum over the individual trap levels.

[5] Auger recombination is currently not implemented.

[6] These parameters can, after all, be different for the different layers in the device. The mobilities, however, can also depend on the position if they depend on the local electric field.

In steady state, bulk SRH recombination is calculated based on the rate

$$R_{\text{SHR,bulk}} = \frac{\texttt{CnCpBulk\_tr}}{\texttt{Cn}(n+n_1)+\texttt{Cp}(p+p_1)}(np-n_i^2), \tag{2.24}$$

where `Cn,p` are the capture coefficients, `Bulk_tr` is the density of traps, and $n(p)_1$ is the electron (hole) density when the quasi-Fermi level matches the trap energy (`ETrapSingle`).

In transient simulations, we use SRH trapping and detrapping rates to form net rates for electron and hole trapping, viz.

$$R_n = C_n n N_{tb}(1-f_{tb}) - C_n n_1 N_{tb} f_{tb} \tag{2.25}$$

and

$$R_p = C_p p N_{tb} f_{tb} - C_p p_1 N_{tb}(1-f_{tb}), \tag{2.26}$$

where f_{tb} is calculated using the value from the previous time step and calculating the emission and absorption rates based on the new carrier densities found in the current time step. This is done by solving the ordinary differential equation,

$$\frac{\partial f_{tb}(t)}{\partial t} = C_n n(1-f_{tb}) - C_n n_1 f_{tb} + C_p p_1(1-f_{tb}) - C_p p f_{tb}, \tag{2.27}$$

where C_n and C_p are the capture coefficients for electrons and holes and $n(p)_1$ is the electron (hole) density when the quasi-Fermi level matches the trap energy (`ETrapSingle`). We get a solution

$$\begin{aligned} f_{tb}(t) =& \frac{C_n n + C_p p_1}{C_n n + C_n n_1 + C_p p_1 + C_p p} \\ &+ c_1 \exp\big(-(C_n n + C_n n_1 + C_p p_1 + C_p p)t\big), \end{aligned} \tag{2.28}$$

where we can express c_1 in terms of the trap filling at time zero ($f_{tb}(0)$) as

$$c_1 = f_{tb}(0) - \frac{C_n n + C_p p_1}{C_n n + C_n n_1 + C_p p_1 + C_p p}. \tag{2.29}$$

Now we can calculate a trap filling at a time t with the new electron and hole densities n, p and $f_{tb}(0)$. This can be done analogously for interface traps.[3]

2.5.3 Interfacial SRH Recombination

The SRH expression needs modification if one wants to use that formalism at an interface. By interface, we mean either the interface between a transport layer and the main absorber, or a grain boundary within the main absorber. At such an interface, trapping and de-trapping can occur to either side of the interface. In order to take all these possible processes into account, we have re-derived the SRH expression. Details can be found in Koopmans *et al.* in Ref. 3.

2.6 Ionic movement

Both `SimSS` and `ZimT` include the effects of ionic species. Their motion is described by the same drift-diffusion equations that are used for electrons and holes, yet without the possibility of leaving or entering the device through the electrodes, or via generation/recombination processes. In other words, their total number is conserved throughout the simulation. The concentrations of negative and positive ions are set via `CNI` and `CPI`, respectively. Whether ions can move into the transport layers or not is set by `IonsInTLs`.

In `SimSS` one needs to specify which ionic species are mobile: the negative, positive, or both species. If a species is not mobile, then its distribution is simply uniform. Otherwise, it is solved based on the total number of ions in the volume and requirement that they cannot enter or leave the device. Whether or not the ion distributions are solved at every applied voltage is governed by parameter `ion_red_rate`: If one would like to simulate a very slow, stabilized voltage sweep, then we need to solve the ion distributions at every voltage. If, on the other hand, one wants to study what happens when the voltage sweep is much faster than the movement of ions (e.g. when using a pre-bias), then they are solved only for the first voltage and then kept fixed for the other voltages.

`ZimT` can do a full transient movement of ions. One needs to specify their mobilities `mobn/pion` and the program will solve for their motion and output their currents. If one of the ion mobilities is set to zero, then their distribution will always be uniform.

2.7 Series and shunt resistances

Figure 2.3 shows the equivalent circuit that `SimSS` and `ZimT` use.[7] However, the way this circuit is used differs for the two codes.

`SimSS` treats the applied voltages (as specified by parameters `Vmin`, `Vmax`, etc.) as the voltage across the simulation volume, i.e. as `Vint`, the internal voltage. When using finite shunt and/or series resistances, it calculates the corresponding external voltage and current density as measured by an experimenter. Both the internal (`Vint`, `Jint`) as well as the external voltages and currents (`Vext` and `Jext`) are stored.[8] Any current (`Jshunt`) flowing through the shunt resistance is also stored. The `Var_file` stores how the internal variables depend on position within the device. As these are internal to the simulation volume they do not contain any contributions from the series or shunt resistances.

`ZimT` is a little bit more sophisticated in that it treats the applied voltage (as specified by the input `tVG_file`, see section 3.5) as the *external* voltage `Vext`. Thus, if there is a (finite) series resistance, `ZimT` will solve for the voltage that is applied to the electrodes (i.e. `Vint`) by using another iteration loop. This arrangement makes it possible, for example, to use `ZimT` to simulate an RC-circuit, or transient measurements where one needs to consider the internal series resistance of the source.[9]

2.8 Iteration scheme and convergence

The system of equations formed by the Poisson and continuity equations are solved in an iterative manner (see Fig. 2.4) based on the work of Gummel.[11] First, a guess is made for the potential and the carrier densities. With this guess, a correction $\delta V(x)$ to the potential is calculated from the Poisson equation: this Poisson solver is iterative and keeps on improving the potential until the changes δV become very small (i.e. smaller than `tolPois`). This new potential is then used to update the carrier densities by solving the continuity equations. This process, the main loop, is repeated until

[7] I realise that these hand-drawn images may look a bit primative. I hope, however, that they add a personal touch. Besides, I do not want to spend too much time on stuff like this.

[8] Any comparison done with an experimental, user-supplied current-voltage characteristic is based on the external voltage and current.

[9] `ZimT` can also be used in steady-state, see section 3.5. Thus, it is possible to specify the external voltage `Vext` while still incorporating a finite series resistance by using `ZimT` instead of `SimSS`.

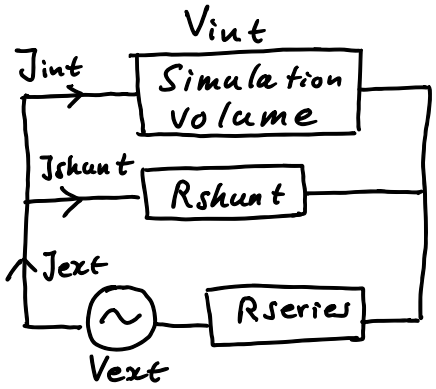

Figure 2.3: Equivalent circuit illustrating the meaning of the internal voltage and current versus the external ones.

convergence is reached.

How do we know when the main loop has converged? Of course, the Poisson solver should have converged. Additionally, the total current flowing through the device (including the displacement current in a transient simulation) must be constant, we check if the current density is sufficiently uniform: The difference between the largest and smallest current, the range, must be smaller than a preset tolerance (`tolJ`).[10] Additionally, we monitor whether the current density `Jint` converges to a definite value. Specifically, let δ_i be the relative change in `Jint` in loop i, then we estimate the error in loop i ϵ_i as

$$\epsilon_i = \frac{\delta_{i-1}\delta_i}{\delta_{i-1} - \delta_i}. \tag{2.30}$$

This expression is based on the assumption that the relative change δ_i decreases exponentially. This implies that the error should be positive,[11] decreasing with each iteration, and smaller than a preset tolerance `tolJ`.

Sometimes the error estimate based on Eq. (2.30) does not work: It supposes that the relative change (δ_i) decreases exponentially. However, there exist cases where this behaviour is simply not observed. In that case, we can check for the absolute value of δ_i. If this is very small in a number of consecutive loops (see constant `MinCountJSmall` in the `DDTypesAndConstants`

[10] In most cases, we look at the range relative to the absolute current density `Jint`. However, if the simulation is performed at open-circuit, which is possible in `ZimT`, then the ideal solution is zero current. In order to avoid dividing by zero, we then take the absolute range of the current.

[11] This implies that $\delta_{i-1} < \delta_i$.

unit) then we might also conclude that the loop has converged. This makes sense if the change is indeed very small, or possibly even oscillating between two very small values. So, if the relative change is `MinCountJSmall` times/loops smaller than `MinRelChange`, then we conclude that the loop has converged provided that the Poisson solver has converged and that the current is sufficiently uniform.

In some cases, the current density is extremely small and its exact value is irrelevant. For example, when performing a simulation in the dark (`Gehp`=0) at voltages close to zero: The diffusion and drift contributions to the current virtually cancel, but the individual components can be very large. To get an accurate solution might be difficult and is not always needed. In this case, `SimSS` and `ZimT` no longer monitor the iteration error (Eq. (2.30)) nor the uniformity of the current, but rather check if the (absolute) current `Jint` is consistently smaller than a threshold value `MinAbsJDark`.[12]

Alas, there is no guarantee that the iteration loop actually converges. Therefore, all iterative loops are limited by a maximum number of iterations. Whether and how the iteration loop converged is indicated by a parameter `convIndex` which is shown on screen and in the `JV_file`. It can take the following values:

- 0: convergence failed, for one reason or another
- 1: the Poisson solver converged, the current is sufficiently uniform, and its error is small enough
- 2: the Poisson solver converged, `Gehp` is zero (dark), and the absolute current `Jint` is smaller than `MinAbsJDark`
- 3: the Poisson solver converged, the current is sufficiently uniform and the relative change is consistently smaller than `MinRelChange`

[12] This alternative criterion can be avoided by setting `MinAbsJDark` to zero.

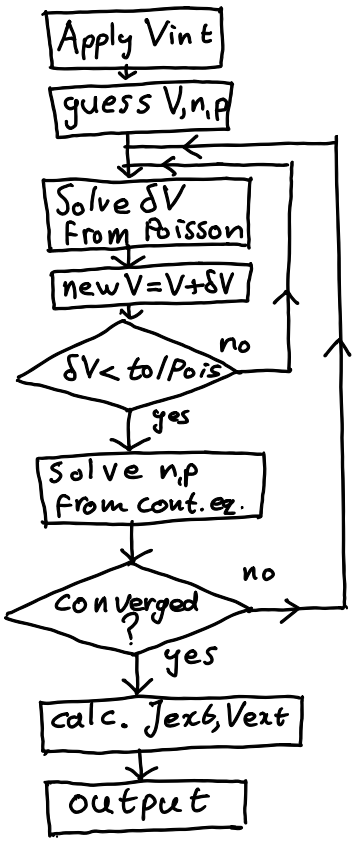

Figure 2.4: Simplified flow diagram of the simulation program. To solve the basic equations, Gummel iteration is used. First, the internal voltage `Vint` is applied to the electrodes (see Eq. (2.10)) and a guess is made for the potential and carrier densities. Subsequently, a correction δV to the potential is calculated from the Poisson equation. This correction is added to the potential V and this is repeated until convergence is reached. Next, the carrier densities are calculated from the new potential by solving the continuity equations. This entire procedure is repeated until converged is reached. The external current and voltage, `Jext` and `Vext` are calculated and the output is shown.

Chapter 3

Input files

In order to run SimSS or ZimT a number of input files must be used.

3.1 Device and simulations parameters

The most important input file is the one that specifies all properties of the device, where input and output files can be found, etc. By default, the SimSS and ZimT look for a file called device_parameters.txt. All input and output files are simple text files and are human-readable. For input files and the log_file, we use the extension .txt, while for output files that are used to store or plot data, we prefer the .dat extension.

The device_parameters.txt file starts like this:[1]

```
** SimSS Device data:
** version: 4.00

**General*********************************************
T = 300                  * K, absolute temperature
L = 400e-9               * m, device length/thickness
eps_r = 24.1             * relative dielectric constant
CB = 3.5                 * eV, conduction band edge
...
```

The order of the parameters is fixed and cannot be changed. However, comments may be added by simply inserting an asterisk at the end of a line (like shown above). If the comment is to be left-justified, use two asterisks. Section 5 lists and defines all parameters.

[1] ZimT's parameter file is similar.

It is also possible to specify a different device parameter file via the command line. This must be the *first* parameter that is passed and should simply state the name of the file:

```
./zimt different_par_file.txt
```

Now `ZimT` will look for file `different_par_file.txt`. Of course, such a parameter file must have the same structure and parameters as the default one. Again, one can change other parameters values via the command line like

```
./zimt different_par_file.txt -T 350
```

3.2 Generation profile

Light absorption in the layer can either be uniform (i.e. constant versus position) or follow a user-defined profile. The latter is specified by parameter `Gen_profile`. The definition of the file format is pretty basic. No header is permitted and it simply consists of a row of x-coordinates (measured from the left electrode) and corresponding generation rate. There can be no more than `maxGenProfPoints` rows.[2] Note: neither need be in real units as the last x-coordinate is re-scaled to the device thickness `L`. This means that the generation profile includes the transport layers (if any, of course). Parameter `TLsAbsorb` can be used to ignore the part of the generation profile that corresponds to the transport layers.

The generation rate only sets the shape of the profile, not the magnitude. The actual magnitude of the profile is controlled by `Gehp` and is set such that the average of the profile over the thickness of the device. In other words, the maximum short-circuit current density equals qL_{eff} `Gehp`, where L_{eff} is the thickness of the absorber.

An example:

```
0 1
2 1.1
3 1.5
4 2
```

This yields a profile that has its maximum absorption at the right electrode (x-coordinate 4 will be scaled to `L`).

[2] See unit `DDTypesAndConstants`.

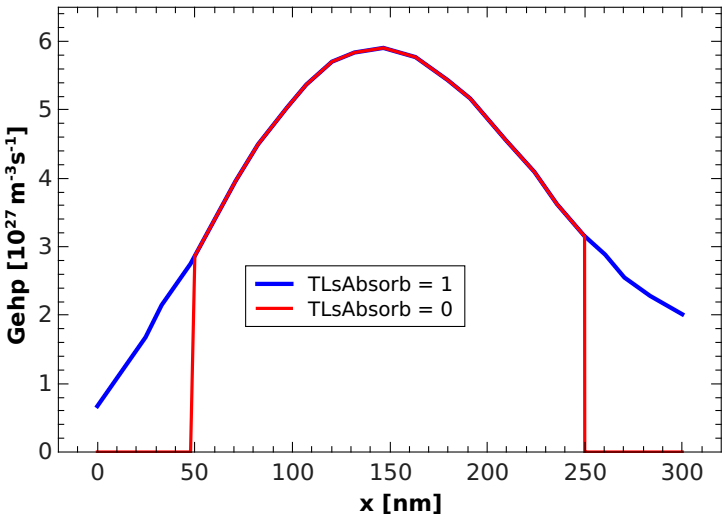

Figure 3.1: This example shows a user-supplied generation profile: it should specify the generation rate of electron-hole pairs (`Gehp`) in the entire simulation volume, so including the transport layers (if any). If the transport layers (TLs) do not absorb, then either the generation profile can be set to zero in the TLs, or one can set `TLsAbsorb` to zero: this will override the profile and ensures that the generation rate in the TLs is zero (see the red line).

3.3 Multiple trap levels

In order to specify multiple trap levels (either in the bulk, or at interfaces), the user can supply a file for either bulk or interface traps.[3] The name of these files is specified by parameter `Bulk/IntTrapFile`. The total density of traps (bulk, interface, or grain boundary) is taken from the respective input parameter, so either `Bulk_tr`, `St_L`, `St_R`, or `GB_tr`. The input file(s) only specify their distribution in energy.

Please note, the first line is ignored and it is strongly suggested this be used to indicate some kind of header. The rest of the file should list the trap levels as energy (in eV, below vacuum, so positive) and the relative number of traps. There can be a maximum of `Max_NEtr` trap levels (see unit TypesAndConstants). The order of the trap levels is of no consequence.

An example of `BulkTrapFile`:

```
Energy  Relative number of traps
4   1     comments may be added
4.2 0.8
4.4 0.2
```

This introduces three trap levels in the bulk. If `Bulk_tr` is set to 2×10^{20} m^{-3}, then the trap level at 4.2 eV will have a density of 8×10^{19} m^{-3}.

After reading the trap energies, a number of checks are performed to ensure that the energies make sense given the conduction and valence band energies of the layer that contains the traps. The occupancy of the trap levels is shown in the `Var_file`. The number of trap levels that is used in the code is the maximum of the those in the bulk or at grain boundaries. So, if using `IntTrapFile` to specify 5 trap levels at the interfaces, while using a single trap in the bulk (simply specified by `ETrapSingle`), then the number of bulk traps shown in the `Var_file` would still be 5. However, only the first such level would actually have a finite number of traps; the other levels all have zero associated traps.

3.4 Experimental current-voltage characteristics

`SimSS` can read an experimental current-voltage curve and compare it with its simulated result. If `UseExpData` = 1 then `SimSS` will try to read the curve from file `ExpJV`. This file should contain a header ('V J' for example)

[3] Traps at grain boundaries get their levels from interface traps.

and then a list of voltages (in V) and current densities (in A/m^2). Note: the short-circuit current density is *negative*.

There should be at least `minExpData` and no more than `maxExpData` points,[4] and there should be a sweep over voltages, either up or down. Using an experimental curve overrides all other specifications of voltages (both in the parameter file and values parsed via the command line). Once the current-voltage curve has been simulated the two are compared and deviations are shown on screen.

An example:

```
V   J   A bad solar cell I measured a long time ago.
0   -10.1
0.2 -5.2
0.3 -1
0.4 0.1
0.5 3.0
```

3.5 Specifying voltage and generation rates in transient simulations

`ZimT` requires a file (`tVG_file`) to specify which times, external voltages[5] and light intensities (generation rates of electron-hole pairs `Gehp`) should be simulated. This file may contain a header and comments. For example:

```
This is the header.
You could put a brief explanation about what this file does.
Fully optional though.
t       Vext   Gehp       comments
0       0       1e26    no comments!
1e-3    0.1     1e26    more voltage!
2e-3    0.1     1e26
3e-3    0.1     1.1e26
4e-3    oc      2e27    now try to find Voc.
```

The comments are optional and `ZimT` ignores them but the `t Vext Gehp` part is mandatory and this should not be preceded by any other characters on that line. If `ZimT` cannot find these characters (white space is ignored,

[4] These constants are provided by unit `DDTypesAndConstants`.

[5] The maximum (absolute) voltage that is accepted depends on the size of the floating point type used (type `myReal` defined in unit `TypesAndConstants`).

case-sensitive) it stops. The first time should be 0 as this corresponds to a steady-state condition at the start of the simulation. Time, voltage and generation rate are all in SI units, so seconds, volts, and m^{-3} s^{-1}.

In order to simulate a transient photovoltage experiment, or just an open-circuit, the voltage may be specified as `oc`—for open-circuit. `ZimT` will then solve for the correct applied voltage such that the current density is close to zero.

`ZimT` reads this file, calculates a time step, reads another line, etc. `ZimT` can also be used to do a steady-state voltage sweep, just like `SimSS` by simply putting all times to 0.[6] Obviously, it is not possible to go back in time.

[6] `ZimT` interprets 0 time as an infinite time-step, which results in steady-state.

Chapter 4

Output

4.1 Screen output

SimSS shows its progress by listing the voltage and current (Vint, Jint)[1] it has found and shows how it concluded that convergence was achieved (convIndex, see section 2.8). If the simulation at some voltage fails to converge, then the user is warned and the current is not shown. If SimSS suspects that the user is simulating a solar cell under illumination, then it will try to obtain and display the main PV parameters (fill-factor, short-circuit current density, etc.). These parameters are calculated based on the external current and voltage, see section 2.7. So, if there is a finite series resistance (Rseries), and Vint = 0 is one of the simulated voltages, then SimSS will not necessarily have simulated the short-circuit condition (Vext = 0) and will, therefore, use interpolation to estimate the short-circuit current density. The error margins that SimSS shows are purely based on interpolation errors and their propagation: they do not take into account any inaccuracies due to grid spacing, or tolerances of the iterative solvers.

If an external current-voltage curve is used,[2] then SimSS will also calculate and show the PV parameters for the external curve so the two can be compared: both the simulated and the experimental parameters are shown as well as their differences. Additionally, SimSS will calculate a root-mean-square error such that simulated and experimental current-voltage curves can be compared directly: this error compares the curves, not just the main PV parameters as defined by parameter rms_mode. Note, this comparison is also done if the simulated device is not a solar cell.

[1] See section 2.7

[2] See UseExpData.

ZimT shows its progress as it reads which times, voltages, and generation rates should be simulated. As there can be very many such time steps, this output (on screen) is shown only every OutputRatio instances. ZimT shows the external current and voltage, see section 2.7. Again, the convIndex is shown to indicate how convergence was achieved, see section 2.8. If one such combination of time, voltage, and generation rate fails to converge, then ZimT switches to red font to warn the user.

4.2 Current-voltage characteristic

The current-voltage characteristic is stored in the JV_file (SimSS) or tj_file (ZimT). It lists the external voltage and current density (see section 2.7), the convergence (convIndex), the Onsager-Braun dissociation rate (see section 2.4). Additionally, the total generation and recombination rates are stored as their equivalent current densities:[3]

- Jphoto is the photogenerated current density,
- Jdir is the direct (band-to-band) recombination current,
- JBulkSRH is the recombination current due to bulk traps,
- JIntLeft is the recombination current due to traps at the left interface,
- JIntRight is the recombination current due to traps at the right interface,
- JminLeft is the minority current at the left electrode,
- JminRight is the minority current at the right electrode,
- JShunt is the current flowing through the shunt resistance.

In transient simulations (like ZimT), the recombination currents of holes and electrons can be different, so ZimT splits these currents into the hole and electron contributions. ZimT will additionally show the time (first column) and the ionic currents (Jnion, Jpion) as well as the displacement current density JD.

[3] All are listed in A/m^2.

4.3 Internal variables

The internal variables (x, V, n, p, J_n, J_p, etc.) are stored in `Var_file` so they may be plotted to further analyse the results. A band diagram can readily be plotted as the vacuum level, conduction and valence bands, and quasi-Fermi levels are all included. The parameter `OutputRatio` determines at which voltages or times these variables are stored. Note, that writing many such variables to file—for example, at every voltage—can slow down the simulation simply due to I/O operations. The number of digits in the output is set by constant `nd` (number of digits) in unit `DDTypesAndConstants`: If the number of digits is not limited, and depending on the size of the floating point type used,[4] the number of digits can be too large for some graphing software to read.

Most of the internal variables are self-explanatory. A few, however, may not be so obvious: `ftb` lists the filling of a bulk trap level (at the respective grid point), while `fti` lists filling of an interface trap. If there are multiple trap levels (see Section 3.3), then the occupancy of the individual trap levels are listed. In steady-state, when electrons and holes recombine via a bulk trap (SHR recombination in the bulk), then the recombination rates for electrons and holes are equal in every grid point (`BulkSRHn/p`). In transient simulations, this is no longer true and they can have different values. Recombination via interface traps (see section 2.5) can also lead to different rates for electrons and holes: the overall rates (`JIntSRH` in `JV_file` or `tJ_file`) should be the same, but they need not be the same in every grid point. Hence, the rates for electrons (`IntSRHn`) and holes (`IntSRHp`) are also shown separately. The generation rates of electron-hole pairs (`Gehp`) and of free electrons and holes (`Gfree`) are also listed. These will be the same unless the Onsager-Braun model is used, see section 2.4.

4.4 Solar cell parameters

`SimSS` will try to calculate the basic solar cell parameters (J_{sc}, V_{oc}, etc.) of the current-voltage characteristic it has calculated *if* the user is simulating something that could be a solar cell under illumination. So this is only done if there is a non-zero generation rate of carriers and if the work functions of the electrodes are different. The errors that are shown are solely based on the interpolation algorithm used, so they do not include the numerical accuracy of the actual simulation. The propagation of errors is included in,

[4] See type `myReal` as defined in unit `TypesAndConstants`.

for example, calculating the maximum-power-point and the fill-factor. The parameters are shown on screen and stored in the `scPars_file`. The program will compare the simulated solar cell parameters with the experimental ones if an experimental current-voltage curve was read (see `UseExpData` and section 3.4). `ZimT` does not have this functionality.

4.5 The log file

Important messages are stored in a separate file, the log (see `log_file`). It stores the version number, the size of the floating point type that was used, and any values from the command lines. The latter is especially relevant as it make it possible for the user to check which values were used. For example, if the user wants to change some parameter via the command line, but makes a typo then the log file will simply not show this parameter. Taken together, the log file and the parameter file specify all parameters that were used in a simulation. `SimSS` also lists which voltages it will simulate and whether the ions are allowed to redistribute at those voltages.

If `ZimT` or `SimSS` fail to find an acceptable solution at some simulated voltage or time, then it will write a short message in the log file to help the user to understand what went wrong: For example, perhaps the main loop was fine but the Poisson solver never converged. Or the Poisson loop was happy, but the current density throughout the simulation volume was not sufficiently uniform. See section 2.8 for more details on the iteration scheme.

4.6 Exit codes

When `ZimT` or `SimSS` run normally, then they return exit code 0. However, when something happens, something unexpected, the exit code will be non-zero, see Table 4.1. There are three cases: If the program finishes without problem, but it did not simulate anything (for example, we used the option `-tidy`), then the exit code is 3. Secondly, if the program encounters something strange, but it detects it in time (so it halts, but it does not crash!), the exit code ranges from 90 –99. The free pascal compiler also has its own list of exit codes. These will be passed on to the operating system in case the program actually crashes, see Table 4.1 for a few typical cases.

Table 4.1: If the program cannot continu due to invalid input, a numerical problem, or even a programming error, it will return a value ranging from 90 –99. If it crashes, then it will return an exit code that is larger than 99.

Code	Meaning
3	Warning-like exit.
Error	
90	Device parameter file corrupted
91	Invalid input (physics, or voltage in `tVG_file` too large)
92	Invalid input from command line
93	Numerical failure
94	Failed to converge, halt (`FailureMode` = 0)
95	Failed to converge at least 1 point, not halt (`FailureMode` $\neq$ 0)
96	Missing input file.
99	Programming error (i.e. not due to the user!)
Fatal (crash)	
106	Invalid numeric format.
200	Division by zero.
201	Range check error.
202	Stack overflow error (only reported when stack checking is enabled).
205	Floating point overflow.
206	Floating point underflow.

Chapter 5

Description of all parameters

5.1 Introductory remarks

All parameters can be set in the file `device_parameters.txt` or a similar file with a different name, see section 3.1. We have tried to break up the parameters into coherent blocks: general, mobilities, etc. The order of the parameters cannot be changed as `SimSS` and `ZimT` simply rely on the order of the input parameters to know which value is what. Comments may be added after an asterisk.

Another way of setting individual parameters is to specify them in the command line. This is incredibly useful if you want to run many simulations, for example on a cluster:

```
./simss -L 200E-9
```

This will run `SimSS` with a thickness (`L`) of 200 nm and overrides the value that is specified in the file `device_parameters.txt`. Note, although this is not case-sensitive, it is probably best simply to stick to the exact spelling as defined in the parameter file.

There are two options that can be used but that are not in the parameter file:

```
./simss -h
```

will make `SimSS` show a short help message and exit (even if more parameters are specified).

```
./simss -tidy
```

will clean-up the parameter file (also see `AutoTidy`) and exit.

A note on the units: `SimSS` and `ZimT` use SI units only, except for eV instead of J for work functions and other energies. So no cm^2 or mA, but only m^2 or A.

5.2 Description of simulation parameters

Some parameters are specific to either `SimSS` or `ZimT` as indicated. The parameters are listed as they appear in the file `device_parameters.txt`.

`version`
The code verifies that the version number of the parameter file matches that of the program. If not, it exits.

`T`
Temperature in K.

`L`
Total thickness (m) of the device, so including the transport layers (if any).

`eps_r`
Relative dielectric constant. The relative dielectric constant of the transport layers (if any) may be specified separately.

`CB`
Conduction band edge (eV, positive). The conduction band edge of the transport layers (if any) may be specified separately.

`VB`
Valence band edge (eV, positive). The valence band edge of the transport layers (if any) may be specified separately.

`Nc`
Effective density of states (m^{-3}) of the conduction and valence bands. The transport layers (if any) may have different values.

`n_0`
Ionised n-doping (m^{-3}). The transport layers (if any) may have different values.

`p_0`
Ionised p-doping (m^{-3}). The transport layers (if any) may have different values.

`mun_0`
Electron mobility (m^2/Vs) at zero field. The transport layers (if any) may have different values.

`mup_0`
Hole mobility (m^2/Vs) at zero field. The transport layers (if any) may have different values.

`mob_n_dep`
This integer value (0 or 1) specifies whether the electron mobility is constant (0) or is field-dependent (1). The field-dependence is of the form

$$\mu(F) = \mu_0 \exp\left(\gamma\sqrt{F}\right), \tag{5.1}$$

where μ_0 is the zero-field mobility (`mun_0`), F is the absolute electric field, and γ is the field activation factor for electrons `gamma_n`. The transport layers (if any) can only have constant mobilities.

`mob_p_dep`
This integer value (0 or 1) specifies whether the hole mobility is constant (0) or is field-dependent (1). The field-dependence is of the form

$$\mu(F) = \mu_0 \exp\left(\gamma\sqrt{F}\right), \tag{5.2}$$

where μ_0 is the zero-field mobility (`mup_0`), F is the absolute electric field, and γ is the field activation factor for holes `gamma_p`. The transport layers (if any) can only have constant mobilities.

`gamma_n`
Field-activation factor ($\sqrt{m/V}$) of the electron mobility. Only relevant if `mob_n_dep` is set to 1.

`gamma_p`
Field-activation factor ($\sqrt{m/V}$) of the hole mobility. This is only relevant if `mob_p_dep` is set to 1.

`W_L`
Work function (eV) of the left electrode. This is typically the cathode.

`W_R`
Work function (eV) of the right electrode. This is typically the anode.

`Sn_L`
Surface recombination velocity (m/s) of electrons at the left electrode. If you would like to use an infinitely large surface recombination velocity, simply use a negative value: in this case, the electron density follows from the difference between the work function and the conduction band (if there is a transport layer adjacent the left electrode, then the conduction band of the transport layer is used). See section 2.2.

`Sp_L`
Surface recombination velocity (m/s) of holes at the left electrode. If you would like to use an infinitely large surface recombination velocity, simply use a negative value: in this case, the hole density follows from the difference between the work function and the valence band (if there is a transport layer adjacent the left electrode, then the valence band of the transport layer is used). See section 2.2.

`Sn_R`
Surface recombination velocity (m/s) of electrons at the right electrode. If you would like to use an infinitely large surface recombination velocity, simply use a negative value: in this case, the electron density at the right electrode (n_R) follows from the difference between the work function and the conduction band (if there is a transport layer adjacent the right electrode, then the conduction band of the transport layer is used). See section 2.2.

`Sp_R`
Surface recombination velocity (m/s) of holes at the right electrode. If you would like to use an infinitely large surface recombination velocity, simply use a negative value: in this case, the hole density at the right electrode (p_R) follows from the difference between the work function and the valence band (if there is a transport layer adjacent the right electrode, then the valence band of the transport layer is used). See section 2.2.

`Rshunt`
This is the shunt resistance of the device (Ω m^2). Use a negative value

to indicate an infinitely large shunt resistance (i.e. no shunt). If `Rshunt` is finite, then the external current density J_{ext} (the current density that is measured) equals

$$J_{\text{ext}} = J_{\text{int}} + V_{\text{int}}/\texttt{Rshunt}, \tag{5.3}$$

where J_{int} is the internal current density—the current in the device flowing between anode and cathode—and V_{int} is the applied voltage on the electrodes. See section 2.7.

`Rseries`
This is a resistance that is placed in series with the devices (Ω m^2). Ideally, this is zero and it cannot be negative. If `Rseries` is positive, then the external voltage (V_{ext}) is modified according to

$$V_{\text{ext}} = V_{\text{int}} + J_{\text{ext}}\texttt{Rseries}, \tag{5.4}$$

where J_{ext} is the external current density and V_{int} is the applied voltage on the electrodes. See section 2.7.

`L_LTL`
Thickness (m) of the left transport layer (TL). Setting `L_LTL` to zero implies no transport layer is present adjacent the left electrode.

`L_RTL`
Thickness (m) of the left transport layer (TL). Setting `L_RTL` to zero implies no transport layer is present adjacent the right electrode.

`Nc_LTL`
Effective density of states of left TL (m^{-3}).

`Nc_RTL`
Effective density of states of right TL (m^{-3}).

`doping_LTL`
Density of ionised dopants (m^{-3}) of left TL. This overrides `n_0` and `p_0`. If `doping_LTL` >0, then this is p-type doping, if `doping_LTL` <0 this corresponds to n-type doping.

`doping_RTL`
Density of ionised dopants (m^{-3}) of right TL. This overrides `n_0` and `p_0`.

If `doping_RTL` >0, then this is p-type doping, if `doping_RTL` <0 this corresponds to n-type doping.

`mob_LTL`
Mobility (m^2/Vs) of electrons and holes in the left TL. This mobility is taken as a constant (not field-dependent) and overrides `mun_0` and `mup_0`

`mob_RTL`
Mobility (m^2/Vs) of electrons and holes in the right TL. This mobility is taken as a constant (not field-dependent) and overrides `mun_0` and `mup_0`

`nu_int_LTL`
Interface transfer velocity (m/s) between main layer and left TL. Internally, `nu_int_LTL` is converted to a mobility μ_{int} at the interface such that the current-voltage curve does not depend on the grid spacing:

$$\mu_{\text{int}} = (q/kT)\Delta x \texttt{nu\_int\_LTL}, \tag{5.5}$$

where Δx is the (local) grid spacing. Had we defined the interfacial mobility μ_{int} simply as an input parameter, then the simulated current would be dependent on the grid spacing which is not physical as an interface does not have a thickness.

`nu_int_RTL`
Interface transfer velocity (m/s) between main layer and right TL. Internally, `nu_int_RTL` is converted to a mobility μ_{int} at the interface such that the current-voltage curve does not depend on the grid spacing:

$$\mu_{\text{int}} = (q/kT)\Delta x \texttt{nu\_int\_RTL}, \tag{5.6}$$

where Δx is the (local) grid spacing. Had we defined the interfacial mobility μ_{int} simply as an input parameter, then the simulated current would be dependent on the grid spacing which is not physical as an interface does not have a thickness.

`eps_r_LTL`
Relative dielectric constant of the left TL.

`eps_r_RTL`
Relative dielectric constant of the right TL.

`CB_LTL`
Conduction band edge (eV, positive) of the left TL.

`CB_RTL`
Conduction band edge (eV, positive) of the right TL.

`VB_LTL`
Valence band edge (eV, positive) of the left TL.

`VB_RTL`
Valence band edge (eV, positive) of the right TL.

`TLsAbsorb`
Integer value to indicate whether the TLs absorb (yes=1, no≠1). This overrides a user-defined generation profile in case this is specified (as defined by parameter `Gen_profile`).

`TLsTraps`
Integer value to indicate whether the TLs can contain bulk traps (yes=1, no≠1). Note, this is only relevant if there are bulk traps (see `Bulk_tr`).

`IonsInTLs`
Integer value to indicate whether ions can move from the bulk into the TLs (yes=1, no≠1).

`CNI`
Concentration of negative ions (m^{-3}). Note, it does not matter whether these are ions, dopants, or vacancies. It is simply a singly negatively charged species that may move (depending on `mob_ion_spec`) or not.

`CPI`
Concentration of positive ions (m^{-3}). Note, it does not matter whether these are ions, dopants, or vacancies. It is simply a singly positively charged species that may move (depending on `mob_ion_spec`) or not.

`mob_ion_spec` (`SimSS`)
Integer value to indicate which ionic species can move. If they cannot move, they will have a uniform distribution (unless they cannot move into the transport layers, then they are restricted to the middle layer). -1: negative ions can move and positive ions are stationary, 0: both species can move,

1: positive ions can move, negative are stationary. In `SimSS`, we cannot set the mobility of the ions as that is only relevant in transient simulations (like `ZimT`).

`ion_red_rate` (`SimSS`)
Integer value (non-negative). If zero, then the mobile ions species (set by `mob_ion_spec`) only move at the first voltage that is simulated (so either `Vmin` or `Vmax`) and the ion distributions are kept fixed for the other voltages. This mimics a very fast up or down scan, i.e. faster than the movement of the ions. If the value is larger than zero, then this is the number of voltage steps after which the ion distributions are updated. So, if 1, then the distributions are updated for every voltage: this corresponds to a very slow, stabilized scan.

If larger than 1, then the ion distributions kept fixed for `ion_red_rate`-1 voltages. This mimics, in a primitive way, a voltage scan where the ions have some time to re-distribute but not enough to fully stabilize at every voltage.[1] The voltages at which the ion distributions are solved and updated are stored in the `log_file`.

`mobnion` (`ZimT`)
Mobility of negative ions (m^{-2}/Vs). Take 0 if they are not supposed to move, this will result in a uniform profile with the possible exception of the transport layers.

`mobpion` (`ZimT`)
Mobility of positive ions (m^{-2}/Vs). Take 0 if they are not supposed to move, this will result in a uniform profile with the possible exception of the transport layers.

`Gehp` (`SimSS`)
Average generation rate (m^{-3} s^{-1}) of electron-hole pairs in the absorbing layer.[2] Note: the generation rate of free electrons and holes can be smaller than this if the Onsager-Braun model is employed.

`Gfrac` (`SimSS`)
In `SimSS`, the actual average generation rate is set as a fraction of `Gehp`.

[1] For a proper treatment of hysteresis and its dependence on scan speed, one needs to use `ZimT`.

[2] This includes the transport layers—if any—provided that they absorb (see `TLsAbsorb`).

We do this, as this makes it easier to do global fitting of a set of solar cells as a function of light intensity where one knows the relative intensities (for example, when using neutral density filters to attenuate the light). So, the effective generation rate equals `Gfrac` × `Gehp`.

`Gen_profile`
This specifies the name of the file that contains the generation profile (see section 3.2 for details). If set to 'none', then a uniform generation profile is assumed (unless the transport layers do not absorb).

`Field_dep_G`
Integer value to indicate whether field-dependent splitting of electron-hole pairs should be used (yes=0, no≠1). See section 2.4.

`P0`
Fraction of electron-hole pairs that directly yield free charge carriers. Only relevant if the Onsager-Braun model of charge generation is used. See section 2.4.

`a`
Charge separation distance (m) in the Onsager-Braun model.

`ThermLengDist`
Integer value that selects which distribution of thermalisation lengths should be used in the Onsager-Braun model. (1) specifies a delta function, the other distributions are:
2: Gaussian

$$g(a,r) = \frac{4}{\mathtt{a}^3\sqrt{\pi}} r^2 \exp\left[-\left(\frac{r}{\mathtt{a}}\right)^2\right] \tag{5.7}$$

3: Exponential

$$g(a,r) = \frac{1}{\mathtt{a}} \exp\left[-\frac{r}{\mathtt{a}}\right] \tag{5.8}$$

4: r^2 exponential

$$g(a,r) = \frac{r^2}{2\mathtt{a}^3} \exp\left[-\frac{r}{\mathtt{a}}\right] \tag{5.9}$$

5: r^4 Gaussian

$$g(a,r) = \frac{8r^4}{3\mathtt{a}^5\sqrt{\pi}} r^2 \exp\left[-\left(\frac{r}{\mathtt{a}}\right)^2\right]. \tag{5.10}$$

`kf`
Decay rate (1/s) of charge-transfer states as used in the Onsager-Braun model.[8–10]

`kdirect`
Rate constant (m^3/s) of direct (band-to-band, or bimolecular) recombination.

`Lang_pre`
Prefactor of the Langevin expression, should be positive but not larger than 1.

`UseLangevin`
Integer value to specify whether the rate constant of direct recombination is calculated from the Langevin expression (1) or not ($\neq$1). In the former case, `Lang_pre` is used in conjunction with the Langevin expression. In the latter, `kdirect` is used.

`Bulk_tr`
Defines the density (m^-3) of traps in the bulk: this can include the transport layers as defined by `TLsTrap`.

`St_L`
Number of traps per area (m^-2) at the left interface between the left TL (if any) and the main absorber.

`St_R`
Number of traps per area (m^-2) at the left interface between the left TL (if any) and the main absorber.

`num_GBs`
Non-negative integer to specify the number of grain boundaries in the main absorber. The grain boundaries are implemented as interface traps put at regular intervals in the main absorber.

`GB_tr`
Number of traps per area (m^-2) at a grain boundary.

`Cn`
Capture coefficient (m^3) for electrons (for all traps). Set to 0 in order to

exclude capture from and emission to the conduction band.

`Cp`
Capture coefficient (m^3) for holes (for all traps). Set to 0 in order to exclude capture from and emission to the valence band.

`ETrapSingle`
Energy level (relative to vacuum, eV) of all traps.

`BulkTrapFile`
This specifies the name of the file that is used to specify multiple bulk trap levels, see section 3.3 for details). Using such a file means that `ETrapSingle` is ignored for bulk traps. If set to 'none', then this file is not read and the trap level (if any) is taken from `ETrapSingle`.

`IntTrapFile`
This specifies the name of the file that is used to specify multiple interface trap levels, see section 3.3 for details). Using such a file means that `ETrapSingle` is ignored for interface traps. If set to 'none', then this file is not read and the trap level (if any) is taken from `ETrapSingle`.

`Tr_type_L`
Integer value (-1, 0, 1) to specify the type of trap at the left interface: -1: acceptor, 0: neutral, 1: donor.

`Tr_type_R`
Integer value (-1, 0, 1) to specify the type of trap at the right interface: -1: acceptor, 0: neutral, 1: donor.

`Tr_type_B`
Integer value (-1, 0, 1) to specify the type of trap at grain boundaries and in the bulk: -1: acceptor, 0: neutral, 1: donor.

`NP`
Integer value to specify the number of grid points. The maximum number of grid points (`Max_NP`) is set in the unit `TypesAndConstants`.

`tolPois`
Absolute tolerance of the Poisson solver (in V): if the largest change δV in a grid point is smaller than this value, then the Poisson solver stops.

`maxDelV`
Maximum change (in terms of the thermal voltage) of the potential per loop of the Poisson solver. This helps to limit the changes in the potential per iteration and can help with convergence.

`MaxItPois`
Maximum number (integer) of iterations of the Poisson solver. Typically a few 100 iterations is plenty.

`MaxItSS`
Maximum number of iterations of the main loop (so Poisson and continuity equations) in steady-state. Note: `SimSS` is always a steady-state simulation.

`MaxItTrans` (`ZimT`)
Maximum number of iterations of the main loop (so Poisson and continuity equations) in transient simulations.

`CurrDiffInt`
This integer value (1 or 2) specifies how the electron and hole currents are calculated. The standard way (1) of doing this is by using Eqs (2.6) and (2.7); this means that we obtain the currents from the differentials of the of the potential and the densities. If the integral form (2) is chosen, then the current is calculated by integrating Eq. (2.3), or its transient cousin (for `ZimT`) Eqs (2.4) and (2.5).

`tolJ`
Tolerance of the current density in the main iteration loop. For details, see section 2.8.

`MinRelChange`
If the relative change of the main loop is smaller than this value, then the loop converges as long as the current is sufficiently uniform and the Poisson solver has converged. For details, see section 2.8.

`MinAbsJDark`
If `SimSS` finds (repeatedly) that the current density is below this limit (A/m^2), then it stops and sets the current equal to zero. The motivation for being able to do this, is that it is very difficult (numerically) to get a sufficiently uniform current in, for example, the dark with low applied

voltage and ohmic contacts: The current is very small and drift & diffusion nearly cancel. However, the gradients in the concentrations are huge and the individual drift & diffusion components are very large. Typically, one is not so interested in exceedingly low currents any way and one does not want `SimSS` to spend a lot of time on this. See section 2.8 for details.

`couplePC`
This non-negative floating point number sets the coupling between the Poisson solver and the continuity equations. The Poisson solver changes the carrier densities (n and p, but also the ionic densities) to reflect any changes to the potential. This helps (quite a lot) in finding a solution. In order to improve the stability of the code, however, it can be helpful to reduce this coupling (smaller `couplePC`). Setting `couplePC` $= 0$ means that the densities are not changed by the Poisson solver.

`accDens`
This floating point value (ranging from 0 to 2) is the acceleration parameter for the solver of the densities. It can be used to enforce successive under- or over-relaxation (SUR/SOR). Let n_i^k be the electron density in grid point i and loop k and δn_i^k the calculated change, then

$$n_i^{k+1} = \texttt{accDens}\delta n_i^k + n_i^k. \tag{5.11}$$

So, if `accDens` < 1 then the new solution is mostly based on the old one (SUR). This is sometimes useful if convergence is difficult as it limits the change per loop. If `accDens` > 1 then the change is larger (SOR) and the convergence can be fast in simple cases, but impossible in others.

`IgnoreNegDens`
Integer value that indicates what to do if the continuity solver finds a negative carrier density (including an ionic density) in a grid point. If not 1 and there is a density on some grid point that is negative, then the program quits. If 1, then we simply ignore this and set that density to some small value. This can be really helpful if the density becomes very small so the difference between zero, small&positive, and small&negative becomes problematic due to the finite number of digits.

`FailureMode`
Integer value that specifies what should be done if the main loop does not converge for some voltage/time. 0: a warning message is shown and the program exits, 1: the solution is accepted, so failure is ignored. 2: the current

voltage/time is skipped. In `SimSS` there is no real difference between 1 and 2.

In `ZimT`, however, this is very different. If the simulation at some time does not converge, then this time point is simply skipped and we move on to the next point in the `tVG_file`. The time step is thus enlarged as the last time point that converged is used to calculate the time step and displacement current. This is useful as there are, from time to time, points that simply do not converge.

`grad`
This parameter determines that gradient of the grid used. `grad` = 0 corresponds to uniform grid spacing, a positive value will make the grid spacing finer near the electrodes and internal interfaces. Setting `grad` to values larger than about 4 will yield extremely small—bordering on the ridiculous—spacing near the electrodes.

`TolVint` (`ZimT`)
Tolerance (V) that determines how accurately the internal voltage (`Vint`) is solved for. In `ZimT`, one specifies the external voltage (`Vext`) and the internal voltage need not be known. If not, then `Vint` is solved for via an iterative procedure (bisection).

`Vdistribution` (`SimSS`)
Integer that is either 1 or 2. This specifies the distribution of voltages that will be simulated. If 1, then this distribution is uniform (specified by `Vstep`). If it is 2, then a logarithmic distribution is used (specified by `Vacc` and `NJV`). The latter is useful when the results will be plotted on a logarithmic voltage axis. Note: ignored if `UseExpData` is 1.

`PreCond` (`SimSS`)
Integer value. If 1, then a pre-bias will be applied (pre-conditioning). This can be used if there are ions in the device and we would like to see if pre-biasing affects the current-voltage curve: First, this bias (`Vpre`) is applied and the ionic densities are calculated. They then either remain fixed for all voltage (if `ion_red_rate`=0) or updated after `ion_red_rate` voltages. The exact voltages at which the ion distributions are solved and updated are stored in the `log_file`. Note, this cannot be used if `UseExpData` or `until_Voc` is 1.

`Vpre` (`SimSS`)

The pre-conditioning voltage (V), see `PreCond`.

`Vscan` (`SimSS`)
Integer (-1 or 1). This indicates whether the voltage should be swept up (1) or down (-1). Note: ignored if `UseExpData` is 1.

`Vmin` (`SimSS`)
Minimum voltage (V) that will be simulated, unless `UseExpData` is 1. The maximum (absolute) value that is accepted depends on the size of the floating point type used (type `myReal` defined in unit `TypesAndConstants`).

`Vmax` (`SimSS`)
Minimum voltage (V) that will be simulated, unless `UseExpData` is 1. The maximum value that is accepted depends on the size of the floating point type used (type `myReal` defined in unit `TypesAndConstants`).

`Vstep` (`SimSS`)
Voltage step (V) used if `Vdistribution` is 1, unless `UseExpData` is 1.

`Vacc` (`SimSS`)
If a logarithmic distribution of voltages is used (`Vdistribution` is 2), then this parameter (V) specifies the accumulation point of a row of voltages. The step size (difference between consecutive voltages) becomes zero at `Vacc`, so it should lie outside the interval should lie outside [`Vmin`, `Vmax`]. Moving `Vacc` closer to either boundary, will result in smaller voltage steps at either boundary. Note: ignored if `UseExpData` is 1.

`NJV` (`SimSS`)
Number (so integer) of voltage points in the logarithmic voltage distribution. Note: ignored if `UseExpData` is 1.

`until_Voc` (`SimSS`)
Integer value. If 1, then the simulation will stop if the simulated current is positive (i.e. $V > V_{oc}$) provided there is light (`Gehp` non-zero). Cannot be used if `UseExpData` is 1.

`Pause_at_end`
If 1, then the program will wait for the user to press Enter once it is done.

`AutoTidy`

Integer value. If 1, then the device parameter file will be tidied up when running `SimSS` or `ZimT` and the simulation will proceed as usual. This ensures that the parameter file stays neat with all comments nicely aligned.

`AutoStop` (`ZimT`)
Integer value. If 1, then `ZimT` tries to identify if it has reached a steady-state, which means it would be pointless to keep on simulating. This depends on the voltage and light intensity in the `tVG_file` but also on the current or voltage that `ZimT` calculates. If the changes are very small and this parameter is 1, then `ZimT` stops.

`UseExpData` (`SimSS`)
Integer value. If 1, then the program should read an experimental current-voltage (JV) curve: `SimSS` will simulate the same voltages that occur in the file (overriding any other specification of voltages). Once it is done, `SimSS` will output a comparison between the simulated and experimental JV curves, including an r.m.s. error (see `rms_mode`). The experimental JV curve should be stored in `ExpJV`.

`ExpJV` (`SimSS`)
Name of the file with the experimental current-voltage (JV) curve. The program will try to read this file is `UseExpData` is 1. For a definition of the file format see section 3.4.

`rms_mode` (`SimSS`)
This indicates how the normalised root-mean-square (r.m.s.) error should be calculated when comparing a simulated with an experimental current-voltage curve. First, `SimSS` finds the smallest (J_{min}) and largest (J_{max}) current density in either the simulated or in the experimental JV-curve. If 'lin' then the r.m.s. error ϵ is calculated as

$$\epsilon^2 = \frac{1}{N(J_{\mathrm{max}} - J_{\mathrm{min}})^2} \sum_{i=1}^{N} \left(J_{\mathrm{exp}}(V_i) - J_{\mathrm{sim}}(V_i)\right)^2 . \tag{5.12}$$

In the case of 'log' we only sum over currents for which the experimental and simulated values have the same sign, and we calculate the r.m.s. error as

$$\epsilon^2 = \frac{1}{N\left[\ln(J_{\mathrm{max}}/J_{\mathrm{min}})\right]^2} \sum_{i=1}^{N} \left[\ln\left(\frac{J_{\mathrm{exp}}(V_i)}{J_{\mathrm{sim}}(V_i)}\right)\right]^2 . \tag{5.13}$$

Note, we only use voltages (V_i) that converged. If the fraction of voltages that converged is small than `rms_threshold` then no r.m.s. error is calculated.

`rms_threshold` (`SimSS`)
See `rms_mode`. If fewer than this fraction of voltages were used in computing the r.m.s. error then no r.m.s. error is shown.

`log_file` (`SimSS`)
`ZimT` and `SimSS` generate some output in a log file (see section 4.5). This parameter sets its name.

`tVG_file` (`ZimT`)
This file contains a list of times, voltages and generation rates. See section 3.5.

`Var_file`
Name of the file where the internal variables (x, V, n, p, Jn, Jp, etc.) are stored. If no such output is required, then simply put 'none' (`ZimT` only).

`OutputRatio` (`SimSS`)
Non-negative integer value. If zero, the internal variables (see section 4.3) will not be stored at all. If positive, the internal variables will be stored in `Var_file` every `OutputRatio` voltages. Note, that writing many such variables to file—for example, at every voltage—can slow down the simulation simply due to I/O operations.

`OutputRatio` (`ZimT`)
Positive integer value. The internal variables will be stored in `Var_file` every `OutputRatio` time-steps, unless `Var_file` is set to 'none'. Note, that writing many such variables to file—for example, at every voltage—can slow down the simulation simply due to I/O operations. The same ratio is applied to the screen output so it cannot be zero as that would mean there would be no screen output.

`tj_file` (`ZimT`)
Name of the output file with the calculated time, voltages, current density. `ZimT` outputs every time step (if successful) to this file.

`JV_file` (`SimSS`)

SimSS will store the simulated current-voltage characteristics in this file.

scPars_file (SimSS)
SimSS will try to figure out if the user wants to simulate a device that could be a solar cell—based on the work functions and whether there is light. If so, it will attempt to calculate the main performance characteristics (open-circuit voltage, short-circuit current, fill-factor, maximum power point) and will store these in this file.

5.3 How to choose the numerical parameters

Whether or not the simulation converges depends on the physical parameters: large densities of traps or ions, widely different electron and hole mobilities, strong illumination conditions, large steps in time, light intensity or voltage all add to the challenge of finding a solution. If SimSS or ZimT struggle, or even fail, to find a solution, then it might help to modify the numerical parameters. Table 5.1 shows the most relevant numerical parameters and their typical values. Refer to section 2.8 to make the best use of tweaks to these parameters.

The numerical parameters impact the speed, convergence, accuracy, or any combination of these. The main ones being NP, tolJ, accDens and grad. While NP and tolJ greatly influence the speed and convergence they also govern the accuracy of the solution. On the other hand, accDens and grad can—in some cases—really improve the speed without sacrificing the accuracy of the solution.

For transient simulations, one has to define an input file with times, voltages, etc. Obviously, the resulting time steps impact the accuracy of the simulation, with larger time steps typically being less accurate than smaller ones. It might, therefore, be tempting to use very small time steps, especially if ZimT struggles to converge. However, extremely small time steps can be the *cause* of the problem: If the time step is very small (for example 1 ps), then any noise in the potential or charge density is amplified which leads to a large displacement current. As a result, the total current might not be sufficiently uniform and ZimT does not converge. Thus, while reducing the time step can help in obtaining a stable and accurate solution, care must be taken not to overdo it.

Table 5.1: Typical value of the most important numerical parameters: they affect the accuracy, speed, and convergence behaviour of the simulations. These values are merely intended as a starting point and may require tailoring in order to get the best results.

parameter	typical value	comments
`NP`	100–400	increase if high accuracy is required
`tolPois`	1e-3–1e-4	
`maxDelV`	1–10	reduce if convergence is difficult
`MaxItPois`	100	
`MaxItSS`	300	larger values can make sense in `ZimT`
`MaxItTrans`	100	`ZimT` only
`tolJ`	0.01–0.001	reduce if high accuracy is needed
`MinRelChange`	1e-8	
`MinAbsJDark`	1e-8	
`accDens`	0.1–1	reduce if convergence is difficult
`grad`	0–4	
`TolVint`	1e-6	`ZimT` only

Bibliography

[1] S. Selberherr, *Analysis and Simulation of Semiconductor Devices* (Springer-Verlag, Wien, 1984).

[2] W. Shockley and W.T. Read, Jr., Phys. Rev. **87**, 835 (1952).

[3] M. Koopmans and L.J.A. Koster (submitted).

[4] C. Snowden, *Introduction to Semiconductor Device Modelling* (World Scientific, Singapore, 1986).

[5] M. Gruber, B.A. Stickler, G. Trimmel, F. Schürrer, and K. Zojer, Org. Electron. **11**, 1999 (2010).

[6] O.W. Purbo, D.T. Cassidy, and S.H. Chrisholm, J. Appl. Phys. **66**, 5078 (1989).

[7] A.H. Marshak, Solid State Electron. **30**, 1089 (1987).

[8] V. D. Mihailetchi, L. J. A. Koster, J. C. Hummelen, and P. W. M. Blom, Phys. Rev. Lett. **93**, 216601 (2004).

[9] L. J. A. Koster, E. C. P. Smits, V. D. Mihailetchi, and P. W. M. Blom, Phys. Rev. B **72**, 085205 (2005).

[10] C. L. Braun, J. Chem. Phys. **80**, 4157 (1984).

[11] H. K. Gummel, IEEE Trans. Electron Devices **11**, 455 (1964).