SIMsalabim project

Manual

version 5.23

S. Heester
M. Koopmans
L.J.A. Koster

June 16, 2025

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.

1.2 Why Pascal?

Yes, this project is written in Pascal and can be compiled using the Free Pascal compiler. Pascal is highly suited to this project as it is a compiled language (so it is fast). Speed (run-time) matters as a single simulation can take anything from less than a second to several minutes or hours and a typical research project requires many such simulations. An interpreted language will result in longer run-times.

Pascal offers excellent readability: it is well-structured and most Pascal words are based on the English language. This matters, especially for an open-source project, as transparency is virtually meaningless if the code is hard to understand. Additionally, Pascal has a high degree of type safety and has a very strict syntax.

1.3 How to specify what you want to simulate?

`SIMsalabim` needs a description of the device you would like to simulate—more on that later—in addition to a specification of the experiment it should mimic. The latter can be, for example, a voltage sweep or a change in light intensity. The software will then use the drift-diffusion model to simulate the measurable outputs (current, voltage) and will also output the internal variables (for example the carrier densities and the potential within the device). Figure 1.1 outlines the basic work flow.

The device itself consists of two electrodes (left and right) and one or more semiconducting layers. So the simplest device consists of a single semiconductor sandwiched between two electrodes. More layers can be added, so as to simulate a device with multiple transport layers and/or absorber

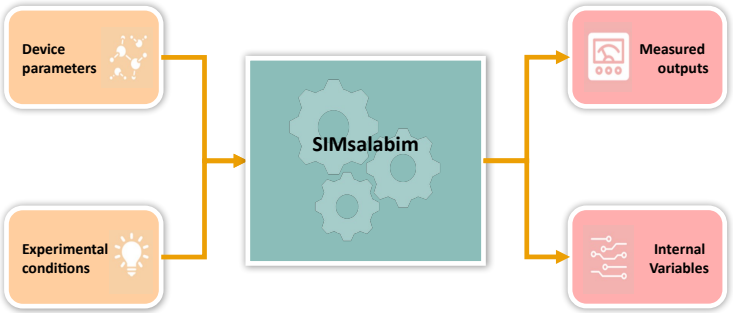

Figure 1.1: Overview of the basic simulation work flow: The input specifies the device (composed of the electrodes and as many layers as required) and the experimental conditions that should be mimicked by the simulation. SIMsalabim will then crunch the numbers and output the current-voltage-time characteristics (the measured outputs in an experiment) and the internal variables in the device. The latter is not accessible in an experiment but is highly useful to better understand the results.

layers. The interface between two semiconducting layers may or may not—depending on the input—contain interfacial traps.

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]

- Steady-state or fully transient simulations
- Arbitrarily many layers can be specified
- 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:

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

  - Different profiles
    - Constant
    - Generated optical absorption profile based on the device structure
    - User-defined optical absorption profile
  - 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

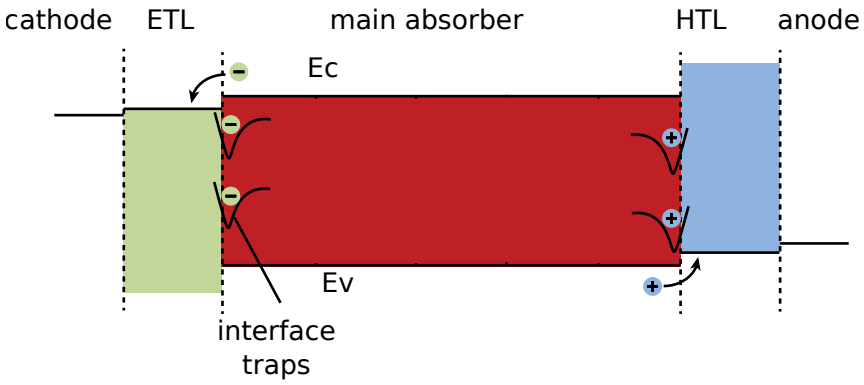

Figure 2.1: Schematic band diagram showing a device that consists of three layers (main absorber, electron and hole transport layers), cathode and anode, and interface traps.

doping, ions, trapped charge carriers. In detail, the term $C(x)$ follows from

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

where N_D and N_A are n- and p-doping densities (see `N_D` and `N_A`), n_{ion} and p_{ion} are the negative and positive ion densities. In this expression, we sum 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 2.6.

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, 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

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

where $J_D(x)$ is the displacement current, and $J_{\mathrm{n(p)ion}}(x)$ 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 2.7.

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.

[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 $p_{i,j,m}$.

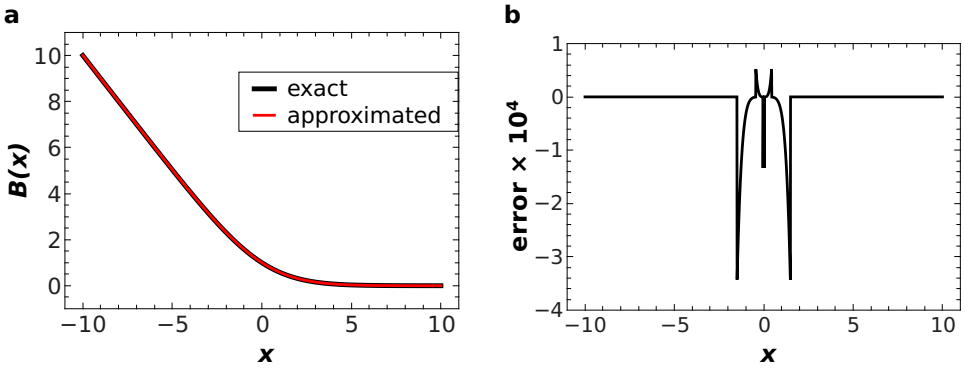

Figure 2.2: The (a) approximated and exact Bernoulli functions, and (b) the error.

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}.

2.2 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 2.8).

The boundary conditions on the carrier densities can take different forms, depending on the surface recombination velocities: If the surface recombination velocities (see `S_n/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

$$n_R = n_{eq} = \mathtt{N\_c} \exp\left(\frac{\mathtt{E\_c} - \mathtt{W\_R}}{kT}\right), \tag{2.11}$$

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

$$J_n = q\mathtt{S\_n\_R}(n_R - n_{eq}), \tag{2.12}$$

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, one has

$$p_R = p_{eq} = \mathtt{N\_c} \exp\left(-\frac{\mathtt{E\_v} - \mathtt{W\_R}}{kT}\right), \tag{2.13}$$

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

$$J_p = q\mathtt{S\_p\_R}(p_R - p_{eq}), \tag{2.14}$$

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 are analogous.

2.3 Internal interfaces

The internal interfaces between the different layers 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. 7. By taking the effective density of states in a layer (N_c) as a reference,[11] 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

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

The current density across interfaces requires a little care. In the thermionic emission model, the current across an interface between grid points i and $i+1$—shown for electrons—is given by[6]

$$J_{ni} = qS_T \left[n_i - n_{i+1} \exp\left(-\frac{\Delta E_c}{kT} \right) \right], \tag{2.20}$$

where S_T is the transfer velocity. Note, this expression is only valid if the effective densities of states at either side of the interface are equal.[6] Moreover, it does not take into account any electric field across the interface.

`SIMsalabim` uses a slightly different expression across the interface that does take into account any difference in effective density of states and electric field:[7]

$$J_{ni} = q\nu_{int} \left[n_i B\left(\frac{V_{gni} - V_{gni+1}}{V_t} \right) - n_{i+1} B\left(\frac{V_{gni+1} - V_{gni}}{V_t} \right) \right], \tag{2.21}$$

where ν_{int} (parameter `nu_int_n`) is the interface transfer velocity. In the absence of an electric field across the interface and assuming equal densities of states at either side of the interface, Eq. (2.21) reduces to

$$J_{ni} = q\nu_{int} B\left(-\frac{\Delta E_c}{kT} \right) \left[n_i - n_{i+1} \exp\left(-\frac{\Delta E_c}{kT} \right) \right]. \tag{2.22}$$

By comparing Eqs (2.20) and (2.22), we can convert S_T into ν_{int}:

$$S_T = \nu_{int} B\left(-\frac{\Delta E_c}{kT} \right). \tag{2.23}$$

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

2.4 Generation profile

The effects of the device structure and layer properties on the generation rate of electron-hole pairs can be taken into account by calculating the optical absorption profile, or generation profile, using the optical transfer matrix model as presented in Refs. 8 and 9. In this model a multilayer device is considered with N layers. The main properties of interest for a layer with index j are its thickness d_j and the complex index of refraction $\tilde{n}_j = n_j + ik_j$ of the layer material. Assumed is that all layers are homogeneous, isotropic and interfaces between layers are parallel. It is also assumed that every photon generates an electron-hole pair.

The generation rate G_j in layer j as a function of the position x in the device and wavelength λ is given by

$$G_j(x,\lambda) = \frac{\lambda}{hc} Q_j(x,\lambda) \tag{2.24}$$

where h is Planck's constant and c the speed of light. $Q_j(x,\lambda)$ is the time average of the energy dissipated per second in layer j for wavelength λ and is given by

$$Q_j(x,\lambda) = \frac{1}{2} c\epsilon_0 \alpha_j n_j |E_j(x)|^2 \tag{2.25}$$

where ϵ_0 is the vacuum permittivity, α_j is the absorption coefficient and $E_j(x)$ the optical electric field at a position x in layer j.

In general, the optical electric field at ambient side and substrate side of the device are related via a scattering matrix (transfer matrix) as

$$E_0^{+,-} = S E_m^{+,-} \tag{2.26}$$

where $E^{+,-}$ represents either the forward and the backward component of the optical electric field. The scattering matrix S consists of the product of interface and layer matrices.

An interface matrix describes the behaviour of a wave at an interface between two layers in the device. Using the Fresnel complex reflection r_{jk} and transmission t_{jk} coefficients, it is given by

$$I_{jk} = \frac{1}{t_{jk}} \begin{bmatrix} 1 & r_{jk} \\ r_{jk} & 1 \end{bmatrix} = \begin{bmatrix} \frac{\tilde{n}_j + \tilde{n}_k}{2\tilde{n}_j} & \frac{\tilde{n}_j - \tilde{n}_k}{2\tilde{n}_j} \\ \frac{\tilde{n}_j - \tilde{n}_k}{2\tilde{n}_j} & \frac{\tilde{n}_j + \tilde{n}_k}{2\tilde{n}_j} \end{bmatrix} \tag{2.27}$$

A layer matrix describes the propagation of a wave through a layer and is given by

$$L_j = \begin{bmatrix} \exp(-i\xi_j d_j) & 0 \\ 0 & \exp(i\xi_j d_j) \end{bmatrix} \tag{2.28}$$

where $\xi_j = \frac{2\pi\tilde{n}_j}{\lambda}$. Multiplied, $\xi_j d_j$ represents the phase change a wave experiences when travelling trough the layer.

To determine the optical electric field inside a layer j for position x, $E_j(x)$, the scattering matrix S is split into two subsets, separated by layer j

$$S = S' L_j S'' \tag{2.29}$$

where S' represents the components of the scattering matrix before L_j and S'' the components of the scattering matrix after L_j. They are defined as

$$S'_j = \begin{bmatrix} S'_{j,11} & S'_{j,12} \\ S'_{j,21} & S'_{j,22} \end{bmatrix} = \left(\prod_{v=1}^{j-1} I_{(v-1)v} L_v \right) I_{(j-1)j} \tag{2.30}$$

$$S''_j = \begin{bmatrix} S''_{j,11} & S''_{j,12} \\ S''_{j,21} & S''_{j,22} \end{bmatrix} = \left(\prod_{v=j+1}^{N} I_{(v-1)v} L_v \right) I_{N(N+1)} \tag{2.31}$$

The total optical electric field in layer j at position x is given by

$$E_j(x) = E_j^+(x) + E_j^-(x) = \left[t_j^+ \exp(i\xi_j x) + t_j^- \exp(-i\xi_j x) \right] E_0^+ \tag{2.32}$$

where t_j^+ and t_j^- are the internal transfer coefficients in the positive and negative direction.[10] This expression can be rewritten using the scattering matrices

$$E_j(x) = \frac{S''_{j,11} \exp(-i\xi_j(d_j - x)) + S''_{j,21} \exp(i\xi_j(d_j - x))}{S'_{j,11} S''_{j,11} \exp(-i\xi_j d_j) + S'_{j,12} S''_{j,21} \exp(i\xi_j d_j)} E_0^+ \tag{2.33}$$

where E_0^+ is the optical electric field at the first interface, which is related to the irradiance I_0.

Finally, to take into account the full solar spectrum, we need to integrate the generation rate $G(x, \lambda)$ over λ

$$G(x) = \int_{\lambda_{min}}^{\lambda_{max}} G(x, \lambda)\, d\lambda \tag{2.34}$$

where $\lambda_m in$ and λ_{max} are the lower and upper bound of the spectrum in terms of wavelength.

2.5 Generation

SimSS and ZimT equate the generation rate of free electrons and holes either to the generation rate of electron-hole pairs (parameter G_ehp)–common in non-excitonic materials—or they consider that such electron-hole pairs may be subject to geminate recombination losses.[12–14] If such geminate losses are taken into account (fieldDepG=1), then the generation rate is reduced by a factor $P(x)$, given by

$$P(x) = \mathtt{P0} + (1 - \mathtt{P0})p(\mathtt{a}, T, F), \tag{2.35}$$

where $p(a, T, F)$ is the Onsager-Braun dissociation probability,[14] 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.,

$$P(a, T, F) = \int_0^\infty p(y, T, F)g(a, y)dy, \tag{2.36}$$

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

2.6 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.

[5] Auger recombination is currently not implemented.

2.6.1 Direct Recombination

The recombination rate of direct recombination rate is given by

$$R_{\text{direct}} = \gamma(np - n_i^2), \tag{2.37}$$

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:

$$\gamma = \texttt{preLangevin}\frac{q}{\varepsilon(x)}(\mu_n(x) + \mu_p(x)), \tag{2.38}$$

where the mobilities $\mu_{n,p}$ and relative dielectric constant ε can depend on position.[6] The Langevin prefactor `preLangevin` 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 `k_direct`, if `useLangevin`=0.

If the Onsager-Braun model for geminate recombination is used (see section 2.5), then the direct recombination rate is reduced by a factor of $1 - P(a, T, F)$.[13]

2.6.2 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.

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

$$R_{\text{SHR,bulk}} = \frac{\texttt{C\_n\_bulkC\_p\_bulkN\_t\_bulk}}{\texttt{C\_n\_bulk}(n + n_1) + \texttt{C\_p\_bulk}(p + p_1)}(np - n_i^2), \tag{2.39}$$

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

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

$$R_n = C_n n N_{tb}(1 - f_{tb}) - C_n n_1 N_{tb} f_{tb} \tag{2.40}$$

and

$$R_p = C_p p N_{tb} f_{tb} - C_p p_1 N_{tb}(1 - f_{tb}), \tag{2.41}$$

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,

$$\frac{\partial f_{tb}(t)}{\partial t} = C_n n(1 - f_{tb}) - C_n n_1 f_{tb} + C_p p_1(1 - f_{tb}) - C_p p f_{tb}, \tag{2.42}$$

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 (`E_t_bulk`). We get a solution

$$\begin{aligned} f_{tb}(t) = &\frac{C_n n + C_p p_1}{C_n n + C_n n_1 + C_p p_1 + C_p p} \\ &+ c_1 \exp\big(-(C_n n + C_n n_1 + C_p p_1 + C_p p)t\big), \end{aligned} \tag{2.43}$$

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

$$c_1 = f_{tb}(0) - \frac{C_n n + C_p p_1}{C_n n + C_n n_1 + C_p p_1 + C_p p}. \tag{2.44}$$

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]

2.6.3 Interfacial SRH Recombination

The SRH expression needs modification if one wants to use that formalism at an interface. By interface, we mean the interface between the selected layer and the layer to the right. At such an interface, trapping and de-trapping can occur to either side of the interface. This is valid for every layer except for the rightmost layer, where there is an interface with the electrode. There we do not take interfacial SRH recombination into account. 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.

2.7 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 properties that define ionic movement are specified per layer. The concentrations and mobilities of anions and cations are set via `N_anion`, `N_cation` and `mu_anion`, `mu_cation`, respectively. Whether ions can move into adjacent layers or not is set by `IonsMayEnter`. As there can be multiple (adjacent) layers where ions can or cannot enter, we define ionic regions. This is a group of adjacent layers that can accept ions. Different ionic regions are separated by layers that cannot accept ions or by the contacts. An example is illustrated in figure 2.3.

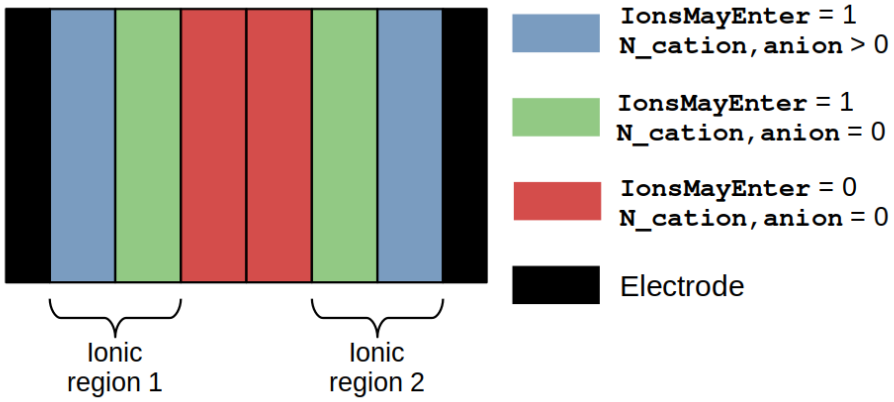

Figure 2.3: Schematic drawing of ionic regions. Suppose we have 6 adjacent layers, the two layers next to the electrodes contain ions, the layers next to them have IonsMayEnter=1, but the other layers cannot accept ions (IonsMayEnter=0). Then we end up with ions in 4 layers, but they form two regions with ions that are not connected.

2.8 Series and shunt resistances

Figure 2.4 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 `varFile` 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 `tVGFile`, 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]

2.9 Iteration scheme and convergence

The system of equations formed by the Poisson and continuity equations are solved in an iterative manner (see Fig. 2.5) based on the work of Gummel.[15] 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`.

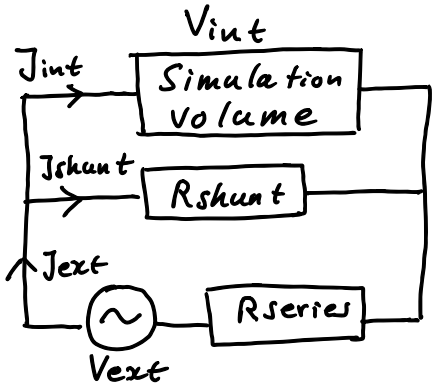

Figure 2.4: Equivalent circuit illustrating the meaning of the internal voltage and current versus the external ones.

convergence is reached.

How do we know when the main loop has converged? Of course, the Poisson solver should have converged. Additionally, we monitor the relative change to the carrier densities in each main loop. If this change is smaller than a preset tolerance (`tolDens`), then the main loop stops. Next, the program computes the internal and external current densities and their error. The error (in A/m^2, so it is absolute rather than relative) is based on the root-mean-square deviation of the current from its average value.

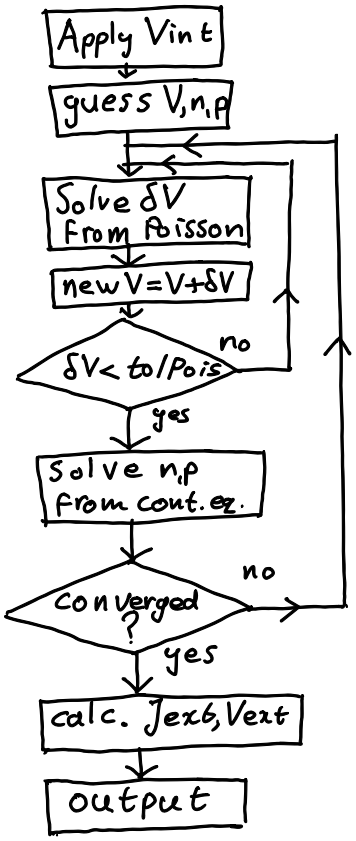

Figure 2.5: Simplified flow diagram of the simulation program. To solve the basic equations, Gummel iteration is used. First, the internal voltage `Vint` is applied to the electrodes (see Eq. (2.10)) and a guess is made for the potential and carrier densities. Subsequently, a correction δV to the potential 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 files are the ones that specify all properties of the device and layers, where input and output files can be found, etc. By default, the SimSS and ZimT look for a file called simulation_setup.txt for the device and simulation parameters. The parameters that define each layer are in separate files as specified in simulation_setup.txt. All input and output files are simple text files and are human-readable. For input files and the logFile, we use the extension .txt, while for output files that are used to store or plot data, we prefer the .dat extension.

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

```
** SimSS Simulation Setup:
** version: 5.23

**General************************************************
T = 295                   * K, absolute temperature

**Layers*************************************************
l1 = L1_parameters.txt    * parameter file for layer 1, mandatory
l2 = L2_parameters.txt    * parameter file for layer 2
...
```

[1] ZimT's parameter file is similar.

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. For a detailed description of these files and the parameters, refer to chapter 5.

An important remark, an interface defined in a layer parameter file always refers to the interface with the current layer and the adjacent layer to the right. This holds for all layers except for the interface between the rightmost layer and the contact, which we do not treat as a relevant interface. The parameters that define the interface are specified in the block *Interface-layer-to-right*. Each layer file must have such a block, even if it is the last (rightmost) layer in the device stack.

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 general parameters values via the command line like

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

or layer specific parameters like

```
./zimt different_par_file.txt -l1.L 150E-9
```

3.2 Generation profile

Light absorption in the layer can be uniform (i.e. constant versus position) or follow either an user-defined profile or a generated profile as described in section 2.4. Which profile is used, is set by the parameter `genProfile`. If set to 'none', a uniform generation profile is used. If set to 'calc', the generation profile is calculated using the transfer matrices. In any other case, the value of `genProfile` is treated as a filename. Parameter `layerGen` can be used to indicate whether a layer absorbs / generates electron-hole pairs or not.

The case of an user-defined profile is specified by setting the parameter `genProfile` to the name of the file with the generation profile. The definition of the file format is pretty basic. There *must* be a header, but comments

may be added before or after the header (or even by simply adding text at the end of the line that contains the header). The header must look as shown below, but it is not case-sensitive. Note: `x` need be in real units as the last x-coordinate is re-scaled to the full device thickness, i.e. the sum of each layer thickness `L`. This means that the generation profile includes all layers that absorbs / generates electron-hole pairs. The generation rate `G_ehp`, however must be absolute and in m^{-3} s^{-1}. The actual magnitude of the profile is multiplied by `G_frac`. An example:

```
* we can start with comments, but it's fully optional
* there must be a line that starts with x G_ehp.
x G_ehp * Note: G_ehp is in m-3 s-1.
0 1e27
2 1.1e27
3 1.5e27
4 2e27
```

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

The case of a calculated profile is controlled by parameter `genProfile`. In this case, the generation profile is calculated based on the device structure, which for example with three layers takes on the following structure, where layers with a * must always be defined:

- Substrate*
- ITO
- Layer 1
- Layer 2
- Layer 3
- Back layer*

For each relevant layer there must be a thickness defined and a file containing the refractive index n and extinction coefficient k per wavelength λ.[2] The shortest wavelength defined must be equal or smaller than `lambda_min` and similar, the longest wavelength defined must be equal or larger than `lambda_max`. This can be extended with as many layers as needed.

An example of a file with n, k values with the required header line:

[2] Please note, λ is in metres.

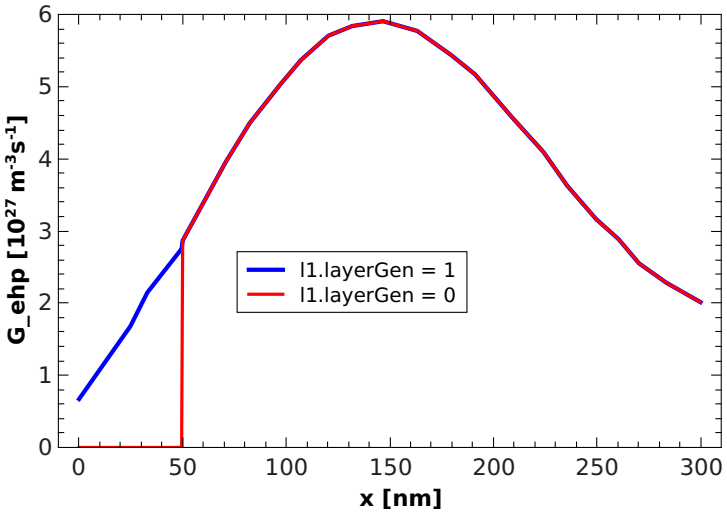

Figure 3.1: This example shows a user-supplied generation profile: it should specify the generation rate of electron-hole pairs (`G_ehp`) in the entire simulation volume, so all defined layers. If any layer does not absorb, then either the generation profile can be set to zero in the layer, or one can set `layerGen` to zero: this will override the profile and ensures that the generation rate in the layer is zero (see the red line, where `layerGen` = 0 in the first layer).

```
lambda n k
300E-9 1.75 0.1888075
305E-9 1.75 0.176253
310E-9 1.73 0.16639
```

The substrate layer (e.g. glass or an anti reflective coating) is an exception to this, because it does not require the definition of a thickness, only a n,k file. The reason for this is that the substrate layer is only used to calculate the initial transmission/reflection at the first interface.

The solar spectrum used (e.g. AM1.5) is controlled by the parameter `spectrum`. The same conditions on the wavelength range as for the n,k files apply here. An example of a spectrum file with the required header line:

```
lambda I
300.0E-9 1.0205E+06
305.0E-9 1.6463E+07
310.5E-9 6.5540E+07
```

The generated profile contains the actual magnitude of the generation rate, the value for `G_ehp` is ignored in this case. Please note, λ is in metres, I is in W/m^3.

For both the spectrum and the files with n,k values the minimal required spacing between consecutive wavelengths is defined by `minDeltaLambda` in unit `DDTypesAndConstants`. The maximum supported number of decimal digits of precision for the wavelength is 6, anything beyond this is ignored.

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. The name of these files is specified by parameter `bulk/intTrapFile`. The input parameters `N_t_bulk` and `N_t_bulk` are ignored if using the input files. The input file(s) specifies the energy (in eV below vacuum, so positive) and the corresponding density of traps. For bulk traps, the latter is in m^{-3}, while for interface traps this is in m^{-2}. The order of the trap levels is of no consequence. An example of `bulkTrapFile`:

```
E   Ntrap  *header, must be there!
4   1e15     comments may be added
4.2 8e14
4.4 2e14
```

This introduces three trap levels in the bulk.

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. Columns `ntb` and `nti` in the `varFile` list the total density of electrons trapped in bulk, resp. interface 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 ('Vext Jext', not case-sensitive) 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`,[3] 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:

```
Vext   Jext * mandatory header
*comments are optional!
0   -10.1
0.2 -5.2
0.3 -1
0.4 0.1
0.5 3.0
```

3.5 Voltage and generation rates in transient simulations

`ZimT` requires a file (`tVGFile`) to specify which times, external voltages[4] and relative generation rates `G_frac`) should be simulated.

This file must contain the correct header and may contain comments. For example:

```
*You could put a brief explanation about what this file does.
*Fully optional though.
t       Vext   G_frac       comments
0       0       1       no comments!
1e-3    0.1     1       more voltage!
2e-3    0.1     1
3e-3    0.1     1.1     a bit more light
4e-3    oc      2       now try to find Voc.
```

The comments are optional and `ZimT` ignores them but the `t Vext G_frac` 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, not 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 and voltage are in

[3] This constant is provided by unit `DDTypesAndConstants`.

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

SI units, so seconds and volts. `G_frac` is the fractional generation rate. The total rate of electron hole pairs is the product of the calculated profile—if using the transfer matrix algorithm—or the `G_ehp` of the respective layers.

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.[5] While it is not possible to go back in time, it *is* possible to go back to the steady-state condition (t=0).

[5] `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`, `Jext`)[1] it has found, including an estimate of the error on the current (see section 2.9). 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 2.8. So, if there is a finite series resistance (`R_series`), 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 fit 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 `fit_mode`. Note, this comparison is also done if the simulated device is not a solar cell. The fit error is defined as the area

[1] See section 2.8

[2] See `useExpData`.

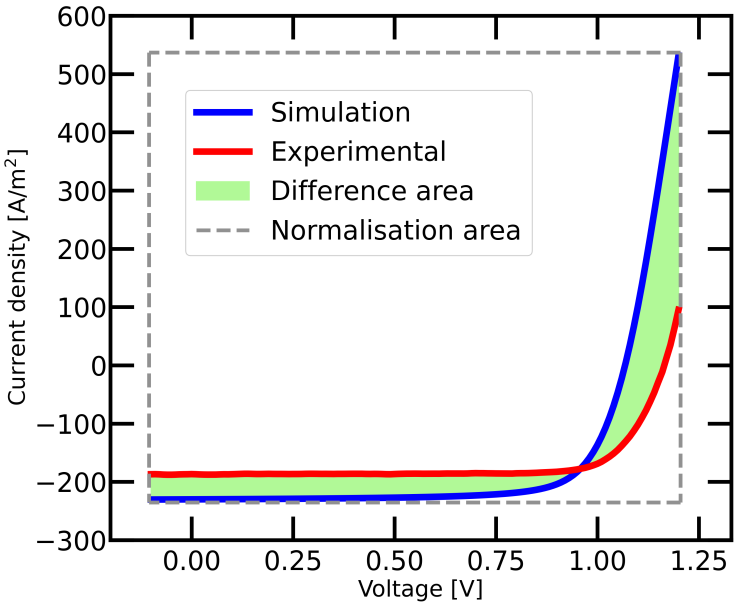

Figure 4.1: An illustration of the definition of the fit error. The fit error is calculated by dividing the difference area between the simulated and experimental curves by the normalisation area.

between the simulated and experimental current-voltage curves normalised with the maximum spanned area, as illustrated in Fig. 4.1.

If `fit_mode` is 'lin' then the fit error ϵ is calculated as

$$\epsilon = \frac{1}{(J_{\max} - J_{\min})\,(V_{\max} - V_{\min})} \int_{V_{\min}}^{V_{\max}} |J_{\exp} - J_{\sim}|\,\mathrm{d}V. \tag{4.1}$$

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 fit error as

$$\epsilon = \frac{1}{\ln\left(\frac{J_{\max}}{J_{\min}}\right)(V_{\max} - V_{\min})} \int_{V_{\min}}^{V_{\max}} \ln\left(\frac{J_{\exp}}{J_{\sim}}\right)\mathrm{d}V. \tag{4.2}$$

Note, we only use voltages (V) that converged.

`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 2.8, including an error estimate of the current. 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 `JVFile` (`SimSS`) or `tjFile` (`ZimT`). It lists the external voltage and current density (see section 2.8), the estimated error on the current, and the internal current density. The next column shows the quasi-Fermi level splitting (in eV), averaged over each layer, followed by a breakdown of the generation and recombination rates (stored as their equivalent current densities), split into the contribution of each layer.[3] So for layer `n` one has:

- `JphotoLn` is the photogenerated current density,
- `JdirLn` is the direct (band-to-band) recombination current,
- `JBulk` is the recombination current due to bulk traps,
- `JintLnLn+1` is the recombination current due to traps the interface between layers `n` and `n+1`,
- `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`.

4.3 Internal variables

The internal variables (x, V, n, p, J_n, J_p, etc.) are stored in `varFile` so they may be plotted to further analyse the results. A band diagram can readily be

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

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 can be limited (which is usually desirable) by setting parameter `limitDigits` to 1. 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: `ntb` lists the density of electrons trapped by bulk traps (summed over all levels) in m^{-3}. The density of electrons trapped in interface traps (so in m^{-2}) is denoted by `nti`. 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 2.6) can also lead to different rates for electrons and holes: the overall rates (`JIntSRH` in `JVFile` or `tJFile`) 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 (`G_ehp`) and of free electrons and holes (`Gfree`) are also listed. These will be the same unless the Onsager-Braun model is used, see section 2.5.

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, for example, calculating the maximum-power-point and the fill-factor. The parameters are shown on screen and stored in the `scParsFile`. The program will compare the simulated solar cell parameters with the experimental ones if an experimental current-voltage curve was read (see `useExpData` and

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

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 `logFile`). 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 2.9 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 continue 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 `tVGFile` too large)
92	Invalid input from command line
93	Numerical failure
94	Failed to converge, halt (`failureMode` = 0)
95	Failed to converge $\geq$ 1 point, not halt (`failureMode` $\neq$ 0)
96	Missing input file.
97	Runtime exceeds limit set by `timeout`.
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 if stack checking enabled).
205	Floating point overflow.
206	Floating point underflow.
207	Invalid floating point operation.
217	Unhandled exception occurred.

Chapter 5

Description of all parameters

5.1 Introductory remarks

All parameters can be set in the `simulation_setup.txt` file and layer files or similar files with different names, see section 3.1. We have tried to break up the parameters into coherent blocks: general, layers, Optics, mobilities etc and split them between two types of files. One which contains the more general parameters that define the device (`simulation_setup.txt`), of which there can only be one, but this is specific for either `SimSS` or `ZimT`. The other type of file contains the parameters that define a layer, where each layer must have it's own parameter file, but they are shared among `SimSS` and `ZimT`. 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 (except for the number of layers) is to specify them in the command line. This is incredibly useful if you want to run many simulations, for example on a cluster. In case of a general parameter this would be like:

```
./simss -T 300
```

This will run `SimSS` with a temperature (`T`) of 300 K and overrides the value that is specified in the file `simulation_setup.txt`. When changing a parameter for a specific layer, the parameter name must be preceded by the layer index parameter, see `l1`, separated by a dot, for example:[1]

```
./simss -l1.L 200E-9
```

[1] Note, although this is not case-sensitive, it is probably best simply to stick to the exact spelling as defined in the parameter files.

This will run `SimSS` with a thickness (`L`) of 200 nm for the first layer and overrides the value that is specified in the parameter file for the first layer.

There one option 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).

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 files.

5.2.1 Simulation setup

The more general parameters like the number of layers, contacts, optics and numerical parameters as defined in the `simulation_setup.txt` file.

`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.

`l1`

Names of parameter files for layer 1 (mandatory).

```
l2
..
..
lN
```

Names of parameter files for rest of layers, from 2 to `N` (optional). Define each layer on a separate line. Layer indices must be consecutive.

`leftElec`

Indicates whether the left electrode is the cathode (-1) or the anode (1). By convention (in semiconductor simulation software), the left electrode is the cathode and this has been the default. However, sometimes one wants to flip the device layout and the cathode is on the right. This also impacts the sign of the current, as it flows in the opposite direction in the device. We take the sign of the current such that injected (e.g. dark) current is positive if the applied voltage is positive, regardless of the choice of cathode and anode.

`W_L`

Work function (eV) of the left electrode. Instead of inputting a fixed value, one can also pass the string 'sfb', which stands for 'semi-flat band', which will result in a work function equal to the equilibrium Fermi level of the adjacent layer, without considering any ions and/or traps that might also contribute space charge at the contact.

`W_R`

Work function (eV) of the right electrode. Instead of inputting a fixed value, one can also pass the string 'sfb', which stands for 'semi-flat band', which will result in a work function equal to the equilibrium Fermi level of the adjacent layer, without considering any ions and/or traps that might also contribute space charge at the contact.

`S_n_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 at the left electrode (n_L) follows from the difference between the work function and the conduction band. See section 2.2.

`S_p_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 at the left electrode (p_L) follows from the difference between the work function and the valence band. See section 2.2.

`S_n_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. See section 2.2.

`S_p_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. See section 2.2.

`R_shunt`
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 `R_shunt` is finite, then the external current density J_{ext} (the current density that is measured) equals

$$J_{\text{ext}} = J_{\text{int}} + V_{\text{int}}/\texttt{R\_shunt}, \tag{5.1}$$

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 2.8.

`R_series`
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 `R_series` is positive, then the external voltage (V_{ext}) is modified according to

$$V_{\text{ext}} = V_{\text{int}} + J_{\text{ext}}\texttt{R\_series}, \tag{5.2}$$

where J_{ext} is the external current density and V_{int} is the applied voltage on the electrodes. See section 2.8.

`G_frac` (`SimSS`)
In `SimSS`, the actual average generation rate is set as a fraction of `G_ehp`. 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 `G_frac` $\times$ `G_ehp`.

`genProfile`
This specifies the name of the file that contains the generation profile (see

section 3.2 for details). If set to 'none', then an uniform generation profile is assumed over the absorbing layers. If set to 'calc' a calculated generation profile based on the device structure and transfer matrices is used.

`L_TCO`
Thickness (m) of the TCO layer. When set to 0, no TCO layer is used.

`L_BE`
Thickness (m) of the back layer/electrode, must be ¿ 0.

`nkSubstrate`
Name of file with n,k values of substrate.

`nkTCO`
Name of file with n,k values of TCO layer. Use none if no TCO layer is defined.

`nkBE`
Name of file with n,k values of the back electrode.

`spectrum`
Name of file that contains the spectrum.

`lambda_min`
Minimum wavelength (m), or the lower bound of the spectrum for the calculated generation profile.

`lambda_max`
Maximum wavelength (m) or the upper bound of the spectrum for the calculated generation profile.

`NP`
Integer value to specify the number of grid points. Must be at least 5 per layer. 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).

`tolDens`
Relative tolerance of the density solver. See section 2.9.

`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.

`minAcc`
Minimum acceleration parameter (must be positive, yet smaller than `maxAcc`). To enable successive over- and under relaxation, we use an acceleration pa-

rameter for the solver of the charge densities. This acceleration parameter (r) depends on the number of performed iterations and varies from `maxAcc` initially downto `minAcc` in the last iteration loop. So, in loop k we have

$$r(k) = \texttt{maxAcc} - (\texttt{maxAcc} - \texttt{minAcc})k/\texttt{maxIt}, \tag{5.3}$$

where `maxIt` is either `maxItSS` or `maxItTrans`, depending on whether this is a steady-state or transient simulation. The acceleration parameter is then used to dampen (if $r < 1$) or accelerate (if $r > 1$) the main loop. For example, let n_i^k be the electron density in grid point i and loop k and δn_i^k the calculated change, then

$$n_i^k = r(k)\delta n_i^k + n_i^{k-1}. \tag{5.4}$$

`maxAcc`
Maximum acceleration parameter (see `minAcc` for details and use). Must be smaller than 2 but larger than `minAcc`.

`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 `tVGFile`. 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).

`Vdist` (`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. The exact voltages at which the ion distributions are solved and updated are stored in the `logFile`. Note, this cannot be used if `useExpData` or `untilVoc` is 1.

`Vpre` (`SimSS`)
The pre-conditioning voltage (V), see `preCond`.

`fixIons` (`SimSS`)
Integer (0 or 1). If 1, the ions are fixed at the first applied voltage.

`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 (`Vdist` 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 `usExpData` is 1.

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

`untilVoc` (`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 (`G_ehp` non-zero). Cannot be used if `useExpData` is 1.

`timeout`
Maximum runtime in seconds. Specifying a negative value implies unlimited runtime.

`pauseAtEnd`
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 en-

sures 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 `fitMode`). 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.

`fit_mode` (`SimSS`)
This indicates how the fit error should be calculated when comparing a simulated with an experimental current-voltage curve. Must be either 'lin' or 'log'. If the fraction of voltages that converged is small than `fit_threshold` then no fit error is calculated.

`fit_threshold` (`SimSS`)
See `fit_mode`. If fewer than this fraction of voltages were used in computing the fit error then no fit error is shown.

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

`JVFile` (`SimSS`)
`SimSS` will store the simulated current-voltage characteristics in this file.

`tjFile` (`ZimT`)

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

`varFile`
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).

`limitDigits`
An integer value. If 1, then the number of digits in the output is limited to a sensible value (depends on the number of digits used in the floating point type `myReal` (see unit TypesAndConstants). If not 0, then the full length of the floating point type is stored (which, again, depends on the size of `myReal`).

`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 `varFile` 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 `varFile` every `outputRatio` time-steps, unless `varFile` 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.

`scParsFile` (`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.

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

5.2.2 Layer specific parameters

The parameters that define the properties of a layer such as generation, recombination, mobilities, trapping and ions. Each layer has its own file and the structure of the file is the same for each layer.

`version`

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

`L`

Thickness (m) of the layer.

`eps_r`

Relative dielectric constant.

`E_c`

Conduction band edge (eV, positive).

`E_v`

Valence band edge (eV, positive).

`N_c`

Effective density of states (m^{-3}) of the conduction and valence bands.

`N_D`

Ionised n-doping (m^{-3}).

`N_A`

Ionised p-doping (m^{-3}).

`mu_n`

Electron mobility (m^2/Vs) at zero field.

`mu_p`

Hole mobility (m^2/Vs) at zero field.

`mobnDep`

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

$$\mu(F) = \mu_n \exp\left(\gamma\sqrt{F}\right), \tag{5.5}$$

where μ_n is the zero-field mobility (`mu_n`), F is the absolute electric field, and γ is the field activation factor for electrons `gamma_n`.

`mobpDep`
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

$$\mu(F) = \mu_p \exp\left(\gamma\sqrt{F}\right), \tag{5.6}$$

where μ_p is the zero-field mobility (`mu_p`), F is the absolute electric field, and γ is the field activation factor for holes `gamma_p`.

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

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

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

$$\mu_{\mathrm{int}} = (q/kT)\Delta x\texttt{nu\_int\_n}, \tag{5.7}$$

where Δx is the (local) grid spacing. See section 2.3 for details.

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

$$\mu_{\mathrm{int}} = (q/kT)\Delta x\texttt{nu\_int\_p}, \tag{5.8}$$

where Δx is the (local) grid spacing. See section 2.3 for details.

`N_t_int`
Number of traps per area (m^-2) at the interface with the layer to the right.

`E_t_int`
Energy level (relative to vacuum, eV) of traps at the interface with the layer to the right.

`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 `E_t_int` is ignored for traps at the interface with the layer to the right. If set to 'none', then this file is not read and the trap level (if any) is taken from `E_t_int`.

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

`C_n_int`
Capture coefficient (m^3 s^{-1}) for electrons for traps at the interface with the layer to the right. Set to 0 in order to exclude capture from and emission to the conduction band.

`C_p_int`
Capture coefficient (m^3 s^{-1}) for holes for traps at the interface with the layer to the right. Set to 0 in order to exclude capture from and emission to the valence band.

`N_anion`
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 or not.

`N_cation`
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 or not.

`mu_anion`
Mobility of negative ions (m^{-2}/Vs). Take 0 if they are not supposed to move, this will result in a uniform profile for this layer.

`mu_cation`
Mobility of positive ions (m^{-2}/Vs). Take 0 if they are not supposed to move, this will result in a uniform profile for this layer.

`ionsMayEnter`
Integer value to indicate whether ions can enter from other layers (yes=1, no≠1).

`G_ehp`
The generation rate (m^{-3} s^{-1}) of electron-hole pairs in this layer. Only used if there is no generation profile (either supplied by the user or calculated using the transfer matrix algorithm). Parameter `G_frac` rescales this value. Note: the generation rate of free electrons and holes can be smaller than this if the Onsager-Braun model is employed. When the value of the generation rate is defined in the generation profile, set to 1.

`layerGen`
Integer value to indicate whether this layer absorbs / generates electron-hole pairs (yes=1, no≠1). This overrides a user-defined or calculated generation profile in case this is specified (as defined by parameter `genProfile`).

`nkLayer`
Name of file with n,k values of this layer.

`fieldDepG`
Integer value to indicate whether field-dependent splitting of electron-hole pairs should be used (yes=0, no≠1). See section 2.5.

`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 2.5.

`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

$$g(a,r) = \frac{4}{\mathtt{a}^3\sqrt{\pi}} r^2 \exp\left[-\left(\frac{r}{\mathtt{a}}\right)^2\right] \tag{5.9}$$

3: Exponential

$$g(a,r) = \frac{1}{\mathtt{a}} \exp\left[-\frac{r}{\mathtt{a}}\right] \tag{5.10}$$

4: r^2 exponential

$$g(a,r) = \frac{r^2}{2\mathtt{a}^3} \exp\left[-\frac{r}{\mathtt{a}}\right] \tag{5.11}$$

5: r^4 Gaussian

$$g(a,r) = \frac{8r^4}{3\mathtt{a}^5\sqrt{\pi}} r^2 \exp\left[-\left(\frac{r}{\mathtt{a}}\right)^2\right]. \tag{5.12}$$

`k_f`

Decay rate (1/s) of charge-transfer states as used in the Onsager-Braun model.[12–14]

`k_direct`

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

`preLangevin`

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, `preLangevin` is used in conjunction with the Langevin expression. In the latter, `k_direct` is used.

`N_t_bulk`

Defines the density (m^{-3}) of traps in the bulk of the layer.

`E_t_bulk`

Energy level (relative to vacuum, eV) of traps in the bulk.

`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 `E_t_bulk` is ignored for bulk traps. If set to 'none', then this file is not read and the trap level (if any) is taken from `E_t_bulk`.

`bulkTrapType`
Integer value (-1, 0, 1) to specify the type of bulk trap: -1: acceptor, 0: neutral, 1: donor.

`C_n_bulk`
Capture coefficient (m^3 s^{-1}) for electrons for traps in the bulk. Set to 0 in order to exclude capture from and emission to the valence band.

`C_p_bulk`
Capture coefficient (m^3 s^{-1}) for holes for traps in the bulk. Set to 0 in order to exclude capture from and emission to the valence band.

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 2.9 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`, `tolDens`, `minAcc`, `maxAcc` and `grad`. While `NP` and `tolDens` greatly influence the speed and convergence they also govern the accuracy of the solution. On the other hand, `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),

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
`tolDens`	1e-7	reduce if high accuracy is needed, increase if convergence is difficult
`minAcc`	0.1	reduce if convergence is difficult
`maxAcc`	0.95	reduce if program crashes
`grad`	0–4	
`tolVint`	1e-6	`ZimT` only

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.

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, Sol. RRL 2200560 (2022).

[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] L. Krückemeier, B. Krogmeier, Z. Liu, U. Rau, and T. Kirchartz, Adv. Energy Mater. **11**, 2003489 (2021).

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

[8] G.F. Burkhard, E.T. Hoke and M.D. McGehee, Adv. Mater. **22**, 3293 (2010).

[9] L.A. Petterson, L.A. Roman and O. Inganäs, J. Appl. Phys. **86**, 487 (1999).

[10] P. Peumans, A. Yakimov and S.R. Forrest, J. Appl. Phys. **93**, 3693 (2003).

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

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

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

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

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