TCAD-Informed Surrogate Models of Semiconductor Devices

by

Samuel B. Chinnery

S.B. Electrical Science and Engineering
Massachusetts Institute of Technology, 2021

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2022

Author . . . .
Department of Electrical Engineering and Computer Science
May 2, 2022

Certified by . . . .
Alan Edelman
Professor of Applied Mathematics
Thesis Supervisor

Certified by . . . .
Christopher Rackauckas
Research Affiliate
Thesis Supervisor

Accepted by . . . .
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

TCAD-Informed Surrogate Models of Semiconductor Devices

by

Samuel B. Chinnery

Submitted to the Department of Electrical Engineering and Computer Science
on May 2, 2022, in partial fulfillment of the requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Extensive research has been conducted over the last half-century to develop models of semiconductor devices for use in circuit analysis and simulation. Such models typically fall into one of two categories: "Cheap" analytical models that can be solved quickly but introduce significant error, and "expensive" physics-based models that achieve high accuracy at the price of prohibitive computation time. As electronic circuits grow to contain billions of active devices, there is a pressing need for new models that are both accurate and fast to compute.

In this thesis, we introduce `Semiconductors.jl`, a new semiconductor simulation tool written in the Julia programming language. We use `Semiconductors.jl` to implement performant surrogate models that approximate the behavior of fine-grained technology computer-aided design (TCAD) device models using a coarsified grid. The resulting surrogate models are shown to approximate the current-voltage characteristics of the fine-grained models to within a maximum error of 0.1% while using less than one tenth as many discretization nodes as the fine-grained baseline model.

Thesis Supervisor: Alan Edelman
Title: Professor of Applied Mathematics

Thesis Supervisor: Christopher Rackauckas
Title: Research Affiliate

Contents

List of Figures

Chapter 1

Introduction

There is a growing need for performant, accurate models of semiconductor devices. Simulation is an integral part of the engineering design process, and many widely used simulators like SPICE [104] use outdated models for transistors and diodes that have little use in modern electronics. The prevalence of complex semiconductor devices like heterojunction bipolar transistors (HBTs) [58, 98], FinFETs [65] and vertical nanowire FETs [50], along with aggressive downscaling of device dimensions into the 1 nm regime [29], has driven a shift toward physics-based device simulators. Such simulators generally rely on technology computer-aided design (TCAD) models of devices to accurately predict their physical characteristics [11]. This is accomplished by solving a system of partial differential equations (PDEs) that govern the transport of electrons and holes in the device.

Physics-based simulators provide superior accuracy to other simulation methods at the expense of significantly increased runtime. While SPICE models are designed to give fast convergence in a wide variety of operating conditions, the PDE systems arising in semiconductor physics are often *stiff*, a term broadly used to describe difficult numerical problems. Circuit level simulations based on TCAD models have thus historically been intractable due to computational expense and convergence difficulties. One approach to circumvent this difficulty has been to apply traditional machine learning techniques to electronic device simulation. Simply training a data-driven model on the characteristics of a semiconductor device often lacks the predictive capability demanded by high-volume engineering applications [69]. Any model used to simulate semiconductor devices must be linked to the underlying physics when explainable, repeatable results are desired.

The emerging field of *scientific machine learning* seeks to establish a connection between computational physics and machine learning. A wide range of scientific machine learning tools have been developed that promise to accelerate many domains within scientific computing [126]. Universal differential equations (UDEs), which use machine learning techniques to augment models based on differential equations, have gained particular attention in recent years. The advent of automatic differentiation (AD), a class of techniques used to evaluate gradients of computer programs, has allowed many computationally expensive models to be surrogatized in a resource-efficient manner through the use of gradient-based optimization.

In this work, we leverage automatic differentiation to create surrogate models of semiconductor devices in a physics-based simulation environment. We introduce `Semiconductors.jl`, a new simulation tool built with the Julia programming language that contains a fully AD-compatible device simulator. We propose several TCAD models of common diodes, bipolar junction transistors (BJTs) and MOSFETs to serve as candidates for surrogatization. Using `Semiconductors.jl` and other scientific machine learning packages, we create performant, coarse-grained surrogate models of 1D and 2D silicon diodes and train the surrogates to minimize error in their I-V curves. The trained surrogates are shown to be accurate to within a maximum error of 0.1% with respect to the fine-grained baseline model.

The remainder of this thesis is structured as follows: Chapter 2 reviews the physics underlying device simulation, introduces the key numerical and computational methods used by `Semiconductors.jl` and discusses related work in surrogatization and inverse design. Chapter 3 describes the implementation of `Semiconductors.jl` and provides documentation for its public methods. Chapter 4 presents experimental results on the characteristics and the surrogatization of the device models implemented in `Semiconductors.jl`. Chapter 5 concludes the thesis and outlines several directions for future work. Appendix A contains derivations related to the discretized semiconductor equations. Finally, Appendix B contains additional results on the character of solutions to the semiconductor problem in the models implemented by `Semiconductors.jl`.

Chapter 2

Theory

2.1 Semiconductor physics

2.1.1 Compact models

Circuit-level simulation of semiconductor devices has traditionally relied on *compact models*, which relate terminal voltages and currents directly through a set of equations. The model equations may be derived empirically through curve-fitting, or the model may be a collection of simpler circuit elements such as resistors and diodes whose behavior closely approximates that of the device of interest. The most common models combine both approaches, using complex networks of passive components whose values vary nonlinearly with operating conditions.

Perhaps the earliest example of a compact semiconductor model is the ubiquitous Shockley diode equation, introduced in 1949 by physicist William Shockley [140]. Succinctly, the model gives the current through a semiconductor diode as

$$I = I_S\left(e^{\frac{V_D}{V_T}} - 1\right),$$

where I_S is a parameter, V_D is the forward bias voltage, and $V_T = kT/q$ is the thermal voltage. This model was instrumental in the development of early solid-state circuits and is widely used in hand analysis to this day. The physics introduced in [140] naturally extended to bipolar junction transistors (BJTs).

The modeling of BJTs began in 1954 with the Ebers-Moll model [34]. This model used identical assumptions to the Shockley diode equation, leading to the following equations for collector current (I_C) and emitter current (I_E):

$$I_C = I_S\left(e^{\frac{V_{BE}}{V_T}} - 1\right) - \frac{I_S}{\alpha_R}\left(e^{\frac{V_{BC}}{V_T}} - 1\right), \quad I_E = \frac{I_S}{\alpha_F}\left(e^{\frac{V_{BE}}{V_T}} - 1\right) - I_S\left(e^{\frac{V_{BC}}{V_T}} - 1\right),$$

where V_{BE} is the base-emitter voltage, V_{BC} is the base-collector voltage, I_S is a device parameter, and $\beta_F = \alpha_F/(1 - \alpha_F)$ and $\beta_R = \alpha_R/(1 - \alpha_R)$ are the forward and reverse current gains, which were further assumed to be constant. This model was found to be overly simplistic in [56], in which the authors derived the Gummel-Poon bipolar model, which has remained in widespread use since its introduction in 1970. Most notably, the Gummel-Poon model allows β_F to vary with operating conditions, closely approximating the behavior of actual BJTs. Additionally, this model incorporated the Early effect, which models the increase in collector current (I_C) of a BJT as collector-emitter voltage (V_{CE}) is increased.

Table 2.1: Comparison of analytical BJT and MOS models.

Model	Device type	Year introduced	Number of parameters	Reference
Ebers-Moll	BJT	1954	3	[34]
Gummel-Poon	BJT	1970	21	[56]
SPICE Gummel-Poon	BJT	1973	39	[104]
VBIC	BJT	1995	85	[97]
Mextram 505	BJT	2017	121	[108]
Schichman-Hodges	MOS	1968	14	[138]
SPICE empirical	MOS	1973	22	[112]
BSIM3 v3.2	MOS	1998	144	[88]
BSIM4 v4.8.2	MOS	2020	349	[25]

The original Gummel-Poon model was considerably extended in the implementation of the SPICE circuit simulator in 1973, resulting in a model today known as the "SPICE Gummel-Poon" model, or SGP [104]. The SGP accounts for temperature dependence in numerous device characteristics, in addition to the improvements offered by the standard Gummel-Poon model [141]. Further improvements to bipolar device modeling include the Vertical Bipolar Inter-Company Model (VBIC) [97] and the Most EXquisite TRAnsistor Model (Mextram) [108], which require on the order of 100 parameters to model complex effects such as parasitic substrate transistors and avalanche breakdown of the base-collector junction. Table 2.1 shows the growth in model complexity over time, both in these BJT models and in common MOS models.

2.1.2 Drift-diffusion model

Accurate, physics-based simulation of electronic devices requires a mathematical model for the transport of charged particles in a semiconductor lattice. The primary charge carriers are *electrons* and *holes*, which carry negative and positive charge, respectively. The most basic physical model for carrier transport is the *drift-diffusion* model. This model accounts for both *drift*, carrier motion due to excitation by an electric field, and *diffusion*, the tendency of carriers to move from areas of high concentration to areas of lower concentration [43]. In other fields, the drift-diffusion model is referred to as the *convection-diffusion* model, which has been used in applications ranging from fluid flow models in physics to options valuation in finance [132].

The drift-diffusion model is usually formulated as a system of three coupled, nonlinear partial differential equations (PDEs). The system results from coupling Poisson's equation with the *continuity equations*, which ensure conservation of charge for electrons and holes [146]. The unknown quantities are the electrostatic potential ψ, and two quantities relating to the carrier concentrations. These may be either the electron and hole densities n and p, or the electron and hole quasi-Fermi potentials ϕ_n and ϕ_p. Generally, the system is given as

$$\begin{cases} \nabla \cdot (\varepsilon \mathbf{E}) + q(n - p - \Gamma) = 0 \\ G - U + \frac{1}{q}\nabla \cdot \mathbf{J_n} = 0 \\ G - U - \frac{1}{q}\nabla \cdot \mathbf{J_p} = 0 \end{cases}, \tag{2.1}$$

where ε is the permittivity at each point in space, Γ is the net ionized impurity concentration, $\mathbf{E} = -\nabla\psi$ is the electric field, q is the electron charge, G and U are the net generation and recombination rates, and $\mathbf{J_n}$ and $\mathbf{J_p}$ are the electron and hole current densities. The current densities may be expressed in terms of ϕ_n and ϕ_p as

$$\mathbf{J_n} = -qn\mu_n\nabla\phi_n, \quad \mathbf{J_p} = -qp\mu_p\nabla\phi_p, \tag{2.2}$$

where μ_n and μ_p are the electron and hole mobilities [39].

With typical doping concentrations (on the order of $1 \times 10^7\,\mu\text{m}^{-3}$ or less), a semiconductor is said to be *nondegenerate*, in which case the Boltzmann approximation for carrier distributions holds [90, 121]. In this case, the quasi-Fermi potentials are approximated as

$$\phi_n = \psi - V_T \ln\left(\frac{n}{n_i}\right), \quad \phi_p = \psi + V_T \ln\left(\frac{p}{n_i}\right), \tag{2.3}$$

where $V_T = kT/q$ is the thermal voltage ($\approx 25.85\,\text{mV}$ at room temperature) and n_i is the intrinsic carrier concentration [39]. Substituting (2.3) into (2.2) gives the current densities as

$$\mathbf{J_n} = -qn\mu_n\nabla\left(\psi - V_T\ln\left(\frac{n}{n_i}\right)\right) = -qn\mu_n\nabla\psi + q\mu_n V_T\nabla n, \tag{2.4}$$

$$\mathbf{J_p} = -qp\mu_p\nabla\left(\psi + V_T\ln\left(\frac{p}{n_i}\right)\right) = -qp\mu_p\nabla\psi - q\mu_p V_T\nabla p. \tag{2.5}$$

Substituting (2.4) and (2.5) into (2.1) and recalling that $\mathbf{E} = -\nabla\psi$ gives the full system in terms of only ψ, n and p:

$$\begin{cases} \nabla\cdot(\varepsilon\nabla\psi) - q(n - p - \Gamma) = 0 \\ G - U + \nabla\cdot(\mu_n(V_T\nabla n - n\nabla\psi)) = 0 \\ G - U + \nabla\cdot\left(\mu_p(V_T\nabla p + p\nabla\psi)\right) = 0 \end{cases}. \tag{2.6}$$

This system is commonly referred to as the *Van Roosbroeck system* and was first formulated by Van Roosbroeck in 1950 [156].

2.1.3 Equilibrium conditions

With appropriate boundary conditions, the Van Roosbroeck system can be solved for any operating conditions to yield ψ, n and p when current flows through the device. In some cases, it is useful to study a special case of this system where no current flows. This is referred to as the *thermal equilibrium* state. In thermal equilibrium, the law of mass action requires $np = n_i^2$ [43]. This assumption, along with the Boltzmann approximation, allows the carrier densities to be expressed directly in terms of ψ as

$$n = n_i \exp\left(\frac{\psi}{V_T}\right), \quad p = n_i \exp\left(-\frac{\psi}{V_T}\right). \tag{2.7}$$

Clearly, $np = ni^2$ in this case. Substituting the expressions from (2.7) into the first equation in (2.6) allows the potential to be determined as the solution of only a single PDE:

$$\nabla\cdot(\varepsilon\nabla\psi) - q\left(2n_i\sinh\left(\frac{\psi}{V_T}\right) - \Gamma\right) = 0. \tag{2.8}$$

Equation (2.8) is known as the *Poisson-Boltzmann equation* [43]. Unlike the Van Roosbroeck system, it allows an entire semiconductor device to be simulated by solving only one differential equation for ψ. When solved numerically by discretization, this reduces the number of coupled equations by a factor of 3, leading to quicker nonlinear solves, and computational experience shows that the Newton process is less likely to diverge with poor initial conditions when only this equation is considered.

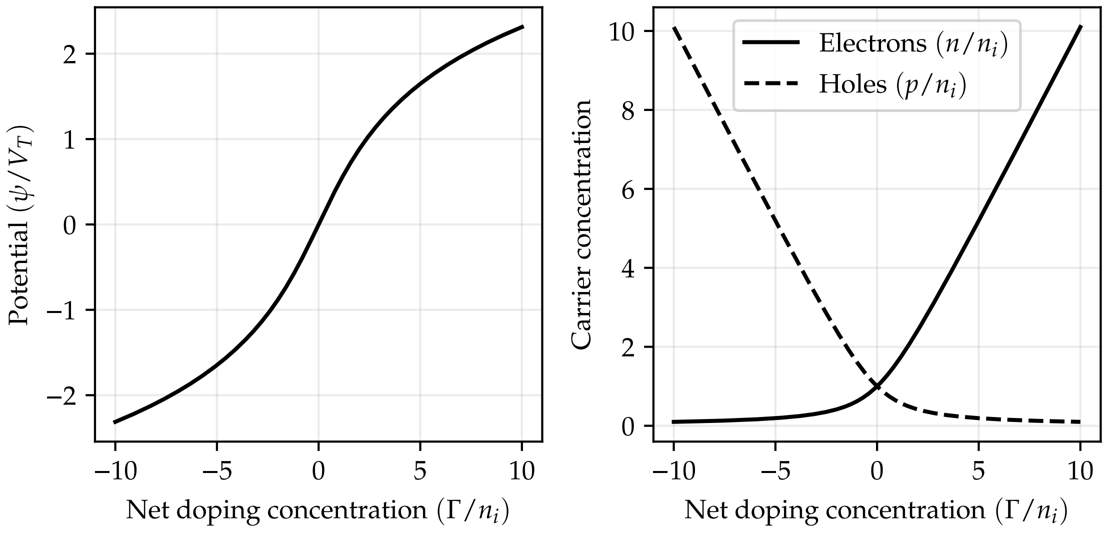


Figure 2.1: Electrostatic potential and electron and hole concentrations as a function of net doping concentration under charge-neutrality conditions.

2.1.4 Charge neutrality

A further simplification can be made to the Van Roosbroeck system by imposing *charge neutrality*. This means that the charge carried by mobile electrons and holes in a region is exactly cancelled by the fixed charge of the ionized dopant atoms. Equivalently, this requires that $n - p = \Gamma$ in charge-neutral regions. Substituting the equilibrium carrier concentrations from (2.7) gives the potential in closed-form as a function of the doping concentration:

$$2n_i \sinh\left(\frac{\psi}{V_T}\right) = \Gamma \implies \psi = V_T \sinh^{-1}\left(\frac{\Gamma}{2n_i}\right). \tag{2.9}$$

From (2.9) we may also determine n and p:

$$\psi = V_T \sinh^{-1}\left(\frac{\Gamma}{2n_i}\right) = V_T \ln\left(\frac{\Gamma}{2n_i} + \sqrt{1 + \left(\frac{\Gamma}{2n_i}\right)^2}\right)$$

$$\implies n = n_i \exp\left(\frac{\psi}{V_T}\right) = \frac{\Gamma}{2} + \sqrt{n_i^2 + \left(\frac{\Gamma}{2}\right)^2}, \quad p = n - \Gamma = -\frac{\Gamma}{2} + \sqrt{n_i^2 + \left(\frac{\Gamma}{2}\right)^2}. \tag{2.10}$$

Equations (2.9) and (2.10) are imposed at Dirichlet boundary conditions in device simulation [42, 105, 120]. Together, they describe an *ohmic contact*, an idealized metal-semiconductor interface at which excess electrons and holes immediately recombine to achieve charge neutrality [43]. The behavior of the potential and carrier concentrations as a function of doping concentration is illustrated in Figure 2.1. When $|\Gamma| \gg 0$, the potential varies logarithmically with the doping concentration. Additionally, the majority carrier concentration approaches $|\Gamma|$ at high doping concentrations, and the minority carrier concentration decreases to maintain $np = ni^2$.

The charge-neutrality conditions are further useful as a method to provide initial guesses for the coupled solution of the Van Roosbroeck system under non-equilibrium conditions [38]. Computational experience shows that charge-neutrality initial conditions are necessary for convergence

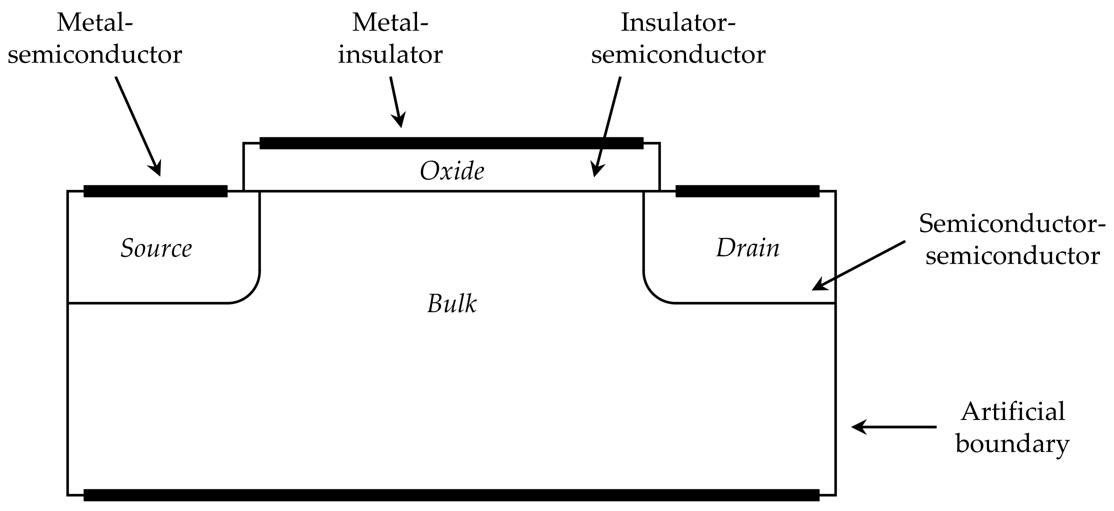


Figure 2.2: Illustration of boundary condition interface types in a simplified 2D MOSFET model. Thickened regions of the device outline represent Ohmic contacts.

in many situations, particularly when large doping gradients are involved (as in the base-emitter junction of a BJT).

2.1.5 Boundary conditions

In a semiconductor device, there are five types of interfaces along which boundary conditions must be imposed:

1. **Metal-semiconductor interface:** Represents an ideal Ohmic contact.

2. **Metal-insulator interface:** Represents a metal or polysilicon contact bordering an insulator region. Typically used to model gate contacts in field-effect transistors.

3. **Semiconductor-semiconductor interface:** Represents a junction between two semiconductor materials with discontinuous parameters, such as permittivity, carrier lifetime, or intrinsic carrier concentration. Does not apply to discontinuous doping profiles, which are naturally handled through the PDE formulation. This enables simulation of heterojunction devices.

4. **Semiconductor-insulator interface:** Represents a semiconductor region bordered by an insulating material. Typically used in gate regions of field-effect transistors or in other 2D devices where a boundary mesh is necessary.

5. **Artificial boundary:** Represents a nonphysical exterior boundary. This is required so that field lines in $\mathbf{E}$, $\mathbf{J_n}$ and $\mathbf{J_p}$ are contained within the simulated region.

Figure 2.2 shows an example of each type of interface in a typical 2D MOSFET model. The presence of semiconductor-semiconductor interfaces at the drain and source regions depends on some discontinuous device parameter. The most common example in a MOSFET would be carrier lifetimes, assuming the n-type and p-type silicon have different properties.

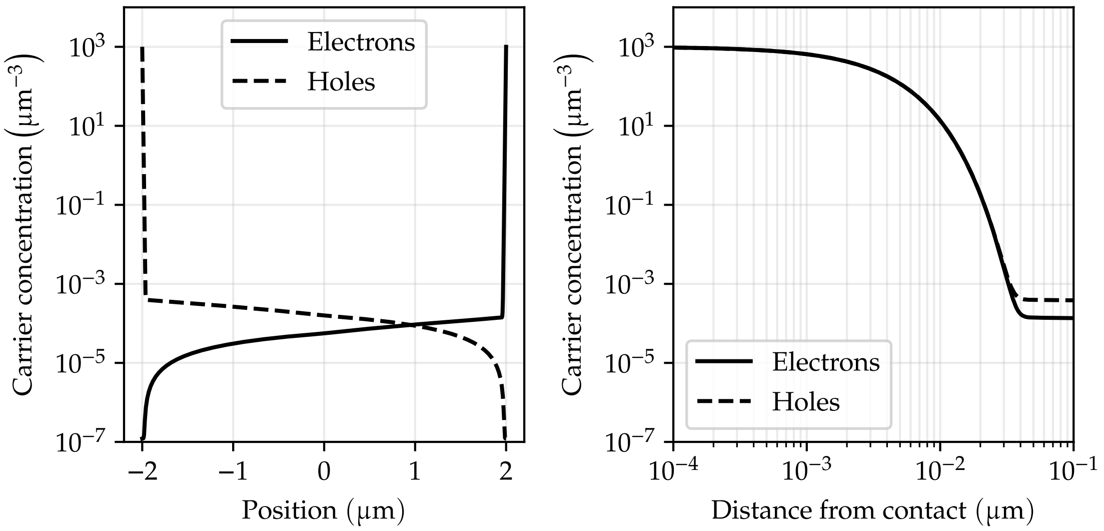


Figure 2.3: Example simulation of 1D diode under heavy reverse bias ($V_R = 50\,\mathrm{V}$) showing boundary layers in carrier concentrations: along entire device length (left), and near the Ohmic contacts, with distance plotted in log-scale.

Dirichlet boundary conditions are imposed at Ohmic contacts to enforce charge neutrality. Specifically, the potential is constrained by (2.9) and the carrier concentrations are constrained by (2.10). Under non-equilibrium conditions, (2.9) may be modified to include an external bias voltage V:

$$\psi = V + V_T \sinh^{-1}\left(\frac{\Gamma}{2n_i}\right), \tag{2.11}$$

where V should be constant along each contact, but Γ may vary depending on the device of interest. Importantly, this means that the Dirichlet boundary condition for the potential may have many different values along a single contact; in fact, this is usually the case for 2D devices with contacts in multiple orientations such as BJTs and MOSFETs.

This method of setting Dirichlet boundary conditions amounts to defining a *voltage-controlled* contact, in that the contact voltage is an independent variable, and the current through it is determined by the simulator. This method can occasionally create *boundary layers* near the contacts, where the carrier concentrations abruptly increase over an unphysically small distance [93, 113]. Such boundary layers can lead to slow convergence with coarse meshes, and numerical error can result near the contacts if appropriate discretization schemes are not used. The boundary layers are less abrupt and can typically be resolved with uniform meshing if carrier concentrations are used in the PDE system, but the boundary layer width may be extremely small (on the order of 1×10^{-30} m under heavy reverse bias) if quasi-Fermi potentials are used [37].

Figure 2.3 shows an example of boundary layers in the carrier concentrations for a 1D diode. In this example, the donor and acceptor doping concentrations are $N_D = N_A = 1 \times 10^3\ \mu\mathrm{m}^{-3}$. The junction is abrupt, and the applied reverse bias voltage is $V_R = 50\,\mathrm{V}$. Under these non-equilibrium conditions, boundary layers of approximately 1 nm width form near the Ohmic contacts in both the electron and hole concentrations. This width would be much smaller if quasi-Fermi potentials were used in the discretization instead of carrier concentrations—as the authors of [37] note, "it is both physically questionable and numerically hopeless to resolve this length scale explicitly."

Dirichlet boundary conditions are also enforced at metal-insulator interfaces to simulate gate contacts. Typically, space charge inside the insulator regions is assumed to be negligible, so only the Poisson equation is solved in those regions [120]. The potential at the surface of the insulator region is then constrained by

$$\psi = V + \phi_{ms}, \tag{2.12}$$

where ϕ_{ms} is the metal workfunction difference potential [109]. A typical value for an $n+$ polysilicon contact with $N_D \approx 1 \times 10^8\,\mu\text{m}^{-3}$ is $\phi_{ms} = 550\,\text{mV}$ [144]. More sophisticated models exist for polysilicon contacts, but it is common to assume that any voltage dropped within the polysilicon is negligible due to its high conductivity [170].

Neumann boundary conditions are enforced at all remaining interfaces [38, 105, 109, 120, 160]. Specifically, we require

$$\mathbf{E} \cdot \hat{\mathbf{n}} = \mathbf{J_n} \cdot \hat{\mathbf{n}} = \mathbf{J_p} \cdot \hat{\mathbf{n}} = 0,$$

where $\hat{\mathbf{n}}$ is the outward-facing normal to the interface. This method of confining the electric field and current density streamlines within the device can sometimes lead to unphysical results when interactions between the device and its environment are important. In this case, a *background mesh* can be used, where a large insulating region is added around the device of interest. The permittivity of this insulating region can be changed to, for example, that of a vacuum or that of air. This can also allow simultaneous solution of the Maxwell equations in 3D if a time-dependent simulation is desired [32].

2.1.6 Recombination models

The quantity U in (2.6) is the net recombination rate. Accurate modeling of carrier recombination is essential to non-equilibrium simulation of electronic devices. The three main recombination processes in semiconductor devices are band-to-band radiative recombination, the Auger process and Shockley-Read-Hall recombination [35, 38, 120, 146]. Radiative recombination occurs when a conduction band electron spontaneously transitions to the valence band, emitting a photon in the process. The radiative recombination rate is

$$U_R = c_r \left(np - n_i^2 \right), \tag{2.13}$$

where c_r is typically $1.8 \times 10^{-3}\,\mu\text{m}^3/\text{s}$ in silicon [109].

The Auger effect was first observed in 1922 and results when an electron and hole recombine directly, transferring energy to a third carrier in the process [101]. The Auger recombination rate is

$$U_A = (c_n n + c_p p) \left(np - n_i^2 \right), \tag{2.14}$$

where c_n is typically $1.0 \times 10^{-7}\,\mu\text{m}^6/\text{s}$ and c_p is typically $2.3 \times 10^{-7}\,\mu\text{m}^6/\text{s}$ in silicon [109].

Shockley-Read-Hall recombination is the primary recombination process in most silicon semiconductors [42, 146]. The effect was first published in 1952 by William Shockley and William Thornton Read and described a process by which carriers exchange energy with the semiconductor lattice in intermediate energy states called *traps* [139]. Traps are caused by impurities in the semiconductor lattice, and the majority of Shockley-Read-Hall recombination is caused by traps near the middle of the band gap [109]. The Shockley-Read-Hall recombination rate is

$$U_{SRH} = \frac{np - n_i^2}{(n + n_i)\tau_p + (p + n_i)\tau_n}, \tag{2.15}$$

where τ_n and τ_p are the *carrier recombination lifetimes*, which may vary significantly depending on doping levels and fabrication processes [109].

In theory, the net recombination rate may be found by combining (2.13), (2.14) and (2.15) to yield $U = U_R + U_A + U_{SRH}$. In practice, either $U = U_A + U_{SRH}$ [120] or just $U = U_{SRH}$ [42, 89] is used in numerical simulation due to the relative insignificance of the other mechanisms in most silicon devices.

2.1.7 Scattering phenomena

Three quantities of great interest in the system in (2.6) are the generation rate G and the mobilities μ_n and μ_p. These quantities are typically used to model *scattering* phenomena—interactions between mobile charge carriers and the semiconductor lattice. In semiconductor physics coursework, it is typically assumed that $G = 0$ and that the carrier mobilities are constant. While this simplifies hand calculations and allows for intuitive understanding of electronic devices, these parameters can vary over several orders of magnitude with numerous properties of the device [121].

It is well known that carrier mobility is significantly degraded in the presence of high electric fields [43, 117, 146]. Under low-field conditions, the drift velocity of electrons and holes is proportional to $\|\mathbf{E}\|$, the magnitude of the electric field. As the electric field is increased, mobile carriers suffer an increasing number of collisions with the semiconductor lattice until they cannot be accelerated any further. The velocity of the carriers at this point is known as the *saturation velocity*, and numerous analytical models exist to simulate this effect computationally [43]. A widely used model is the Caughey-Thomas model, first introduced by Caughey and Thomas in 1967 [16]. This model gives the mobilites as

$$\mu_n = \frac{\mu_{n0}}{\left(1 + (\|\mathbf{E}\|/E_{cn})^{\beta_n}\right)^{1/\beta_n}}, \quad \mu_p = \frac{\mu_{p0}}{\left(1 + \left(\|\mathbf{E}\|/E_{cp}\right)^{\beta_p}\right)^{1/\beta_p}}, \tag{2.16}$$

where μ_{n0} and μ_{p0} are the low-field mobilities, E_{cn} and E_{cp} are the critical electric fields, and β_n and β_p are fitting parameters. The parameter values given in [16] are repeated below:

$$\mu_{n0} = 4.870 \times 10^{10}\ \mu\text{m}^2\text{V}^{-1}\text{s}^{-1}, \quad \mu_{p0} = 1.375 \times 10^{11}\ \mu\text{m}^2\text{V}^{-1}\text{s}^{-1},$$
$$E_{cn} = 1.95\,\text{V}/\mu\text{m}, \quad E_{cp} = 8.00 \times 10^{-1}\,\text{V}/\mu\text{m}, \quad \beta_n = 2.0, \quad \beta_p = 1.0. \tag{2.17}$$

Other values for the parameters have been proposed, but the original values given in (2.17) remain common in the literature [35]. This model can also be extended to account for mobility degradation at high doping levels; however, this effect can be negligible in non-degenerate semiconductors with low doping levels [16].

Impact ionization is the process by which electrically excited charge carriers transfer energy to atoms in the semiconductor lattice and generate additional charged carriers [146]. The effect of impact ionization on the Van Roosbroeck system is similar to that of recombination, except that carriers are added to the system instead of removed. Under high-field conditions, impact ionization can be the dominant cause of current flow in a device, particularly in reverse-biased PN junctions. The process by which current multiplies rapidly under heavy reverse bias in a PN junction is known as *avalanche breakdown* and can be destructive at high current levels [31, 67].

In general, the rate of carrier generation due to impact ionization is

$$G = \frac{1}{q}\left(\alpha_n\|\mathbf{J_n}\| + \alpha_p\left\|\mathbf{J_p}\right\|\right), \tag{2.18}$$

where α_n and α_p are the *ionization coefficients* of electrons and holes [146]. Several expressions exist for the ionization coefficients; their defining characteristic is that the coefficients increase abruptly with $\|\mathbf{E}\|$ for at least some values of $\|\mathbf{E}\|$. This general form was proposed by Chynoweth in 1958, who conjectured that the ionization coefficients could be written as $a\exp(-b/\|\mathbf{E}\|)$ for some constants a and b [19]. This conjecture was experimentally validated over the next three decades, eventually leading to the model

$$\alpha_n = \alpha_n^\infty \exp\left(-\frac{E_{in}}{\|\mathbf{E}\|}\right), \quad \alpha_p = \alpha_p^\infty \exp\left(-\frac{E_{ip}}{\|\mathbf{E}\|}\right), \tag{2.19}$$

where α_n^∞ and α_p^∞ are the limiting values of α_n and α_p as $\|\mathbf{E}\| \to \infty$, and E_{in} and E_{ip} are constants. Several values for these parameters are summarized by Selberherr in [136].

Other models for the ionization coefficients include the model proposed by Thornber in 1981, which has the form

$$\alpha = \frac{q\|\mathbf{E}\|}{E_i}\exp\left(-\frac{F_i}{\|\mathbf{E}\|(1+\|\mathbf{E}\|/F_r)+F_{kT}}\right),$$

where E_i, F_i, F_r and F_{kT} are parameters with distinct values for electrons and holes [149]. More recently, *nonlocal models* such as the *lucky electron model* have emerged that rely on values along the entire device to compute the generation rate at a single point in space [72]. While all three models have proven to be physically accurate in some situations, the expressions in (2.19) are often used in device simulation for simplicity [35].

2.1.8 Extensions to drift-diffusion

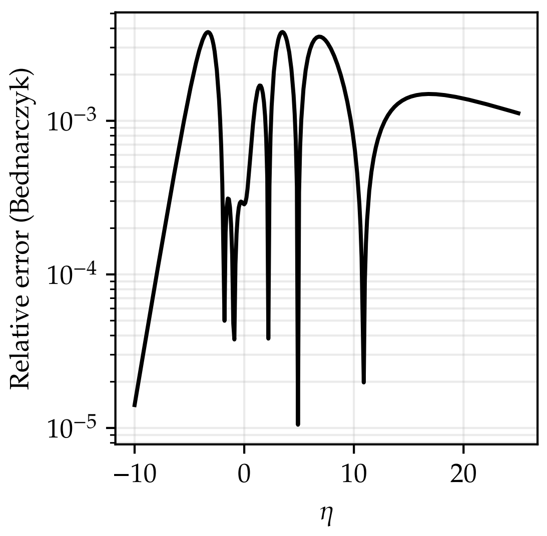
As device dimensions have decreased, it has become necessary to revisit many of the classical assumptions made in device physics. Modern device simulators must account for an increasing number of nanometer-scale phenomena to produce reliable results [85]. Perhaps the most limiting assumption of the system in (2.6) is the Boltzmann approximation, which only holds for nondegenerate semiconductors. Modern silicon devices frequently make use of degenerate doping concentrations, with values on the order of $1 \times 10^8\ \mu\text{m}^{-3}$ or higher being common in 10 nm CMOS [87, 121]. The Boltzmann approximation can overestimate the carrier concentrations in degenerate semiconductor regions and underestimate the electrostatic potential at Ohmic contacts [38].

To address this limit, the Fermi-Dirac statistics may instead be used to model carrier concentrations in device simulation. As a consequence of this choice, the quasi-Fermi potentials ϕ_n and ϕ_p cannot generally be written in closed-form as a function of the carrier concentrations n and p. Thus, the charge neutrality relationships in (2.10) and (2.9) no longer hold and must be determined by a nonlinear solver [38]. The use of Fermi-Dirac statistics is further complicated by its use of the *Fermi-Dirac integral* of order 1/2:

$$F_{1/2}(\eta) = \frac{2}{\sqrt{\pi}}\int_0^\infty \frac{\sqrt{x}}{e^{x-\eta}+1}\,\mathrm{d}x\,. \tag{2.20}$$

The function $F_{1/2}(\eta)$ is analytic for $\eta \in \mathbb{R}$, but evaluating it numerically poses considerable difficulty [77]. Several approximations have been proposed to allow accurate computation of (2.20) with minimal overhead. An elegant approximation is given by Bednarczyk in [9]:

$$F_{1/2}(\eta) \approx \frac{4}{3\sqrt{\pi}}\left(\frac{1}{a(\eta)^{-3/8}+\exp(-\eta)}\right), \quad \text{where}$$
$$a(\eta) = \eta^4 + 33.6\eta\left(1-0.68\exp\left(-0.17(\eta+1)^2\right)\right)+50. \tag{2.21}$$



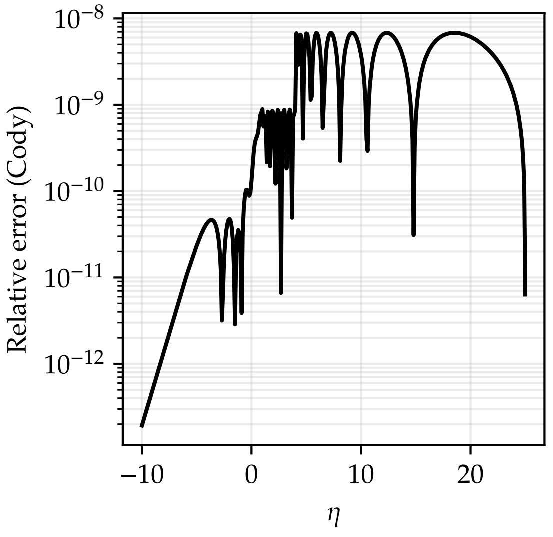


Figure 2.4: Relative error of Fermi-Dirac approximations given in [9] (left) and in [20] (right).

The error of this approximation is below 0.4% for all $\eta \in \mathbb{R}$. Other approximations for $F_{1/2}(\eta)$ include that given in [20], which maintains an error below 1×10^{-8} for all $\eta \in \mathbb{R}$ but is more computationally expensive to evaluate. The relative errors of these approximations are shown in Figure 2.4. The exact values of $F_{1/2}(\eta)$ were computed using Mathematica 12.1, which gives the integral as

$$F_{1/2}(\eta) = -\mathrm{Li}_{3/2}(-e^{\eta}) = -\sum_{k=1}^{\infty} \frac{e^{-k\eta}}{k^{3/2}},$$

where $\mathrm{Li}_{3/2}(z)$ is the polylogarithm function of order 3/2.

Beyond carrier statistics, other physical models can replace or augment drift-diffusion if sub-micron accuracy is desired. The *hydrodynamic model* allows the temperature of the charge carriers to be different from the lattice temperature and accounts for energy conservation in both electrons and holes [162]. Each additional temperature adds an equation to the Van Roosbroeck system, resulting in a total of 5 coupled PDEs. The lattice temperature can additionally be modeled as non-constant to account for self-heating effects. In this case, the heat equation is solved self-consistently with the transport equations, resulting in a system of 6 equations if the hydrodynamic model is used, or 4 equations if the drift-diffusion model is used [109]. More complex models include the Boltzmann Transport Equation and quantum models based on the Schrödinger equation. Such models often rely on a Monte Carlo approach to model carrier transport at the level of individual electrons and holes and are considerably more expensive to evaluate than deterministic models [8, 130, 160, 161].

2.2 Numerical methods

2.2.1 Finite difference methods

Ordinary differential equations (ODEs) with only one independent variable can often be solved using generic differential equation solvers such as those provided in `DifferentialEquations.jl` [124]. For many partial differential equations (PDEs), however, it is difficult or impossible to successfully apply general solution methods. The Van Roosbroeck system is no exception, and many approaches have been developed to reliably and accurately provide convergence for semiconductor problems.

At the core of any PDE solver is the *discretization*: a method for decoupling a PDE or a system of PDEs into a system of algebraic equations. The most fundamental discretization is the *finite difference* approach, which relies on the definition of the derivative as a limit. Given a grid $x = x[i]$ and a function $f[k] \triangleq f(x[k])$, the *forward difference* of f at $x[k]$ is

$$f'[k] = \frac{f[k+1] - f[k]}{x[k+1] - x[k]}. \tag{2.22}$$

Similarly, the *reverse difference* of f at $x[k]$ is

$$f'[k] = \frac{f[k] - f[k-1]}{x[k] - x[k-1]}. \tag{2.23}$$

The approximations in (2.22) and (2.23) rely on the behavior of f on either side of the point $x[k]$, suggesting that an approximation considering both $f[k+1]$ and $f[k-1]$ could be more accurate. In particular, we can average (2.22) and (2.23) to yield

$$\begin{aligned} f'[k] &= \frac{1}{2}\left(\frac{f[k+1] - f[k]}{x[k+1] - x[k]} + \frac{f[k] - f[k-1]}{x[k] - x[k-1]}\right) \\ &= \frac{1}{2}\left(\frac{f[k+1]}{x[k+1] - x[k]} - \frac{f[k-1]}{x[k] - x[k-1]} + \left(\frac{x[k+1] - 2x[k] + x[k-1]}{(x[k+1] - x[k])\,(x[k] - x[k-1])}\right) f[k]\right). \end{aligned} \tag{2.24}$$

The approximation in (2.24) is the *central difference*, which gives the derivative to a higher accuracy than either (2.22) or (2.23) but requires the value of f at 3 grid points instead of 2. The central difference can also be used to evaluate the second derivative of f:

$$\begin{aligned} f''[k] &= \frac{1}{2}\left(\frac{f'[k+1] - f'[k]}{x[k+1] - x[k]} + \frac{f'[k] - f'[k-1]}{x[k] - x[k-1]}\right) \\ &= \frac{1}{4}\left(c_1 f[k-2] + c_2 f[k-1] + c_3 f[k] + c_4 f[k+1] + c_5 f[k+2]\right), \end{aligned} \tag{2.25}$$

where

$$\begin{aligned} c_1 &= \frac{1}{(x[k] - x[k-1])\,(x[k-1] - x[k-2])}, \\ c_2 &= \frac{1}{x[k] - x[k-1]}\left(\frac{1}{x[k+1] - x[k]} - \frac{1}{x[k-1] - x[k-2]}\right), \\ c_3 &= -\frac{2}{(x[k+1] - x[k])\,(x[k] - x[k-1])}, \\ c_4 &= \frac{1}{x[k+1] - x[k]}\left(\frac{1}{x[k+2] - x[k+1]} - \frac{1}{x[k] - x[k-1]}\right), \\ c_5 &= \frac{1}{(x[k+2] - x[k+1])\,(x[k+1] - x[k])}. \end{aligned}$$

The complexity of (2.25), along with its requirement of 5 values of f for each computation, motivates consideration of a simpler approach.

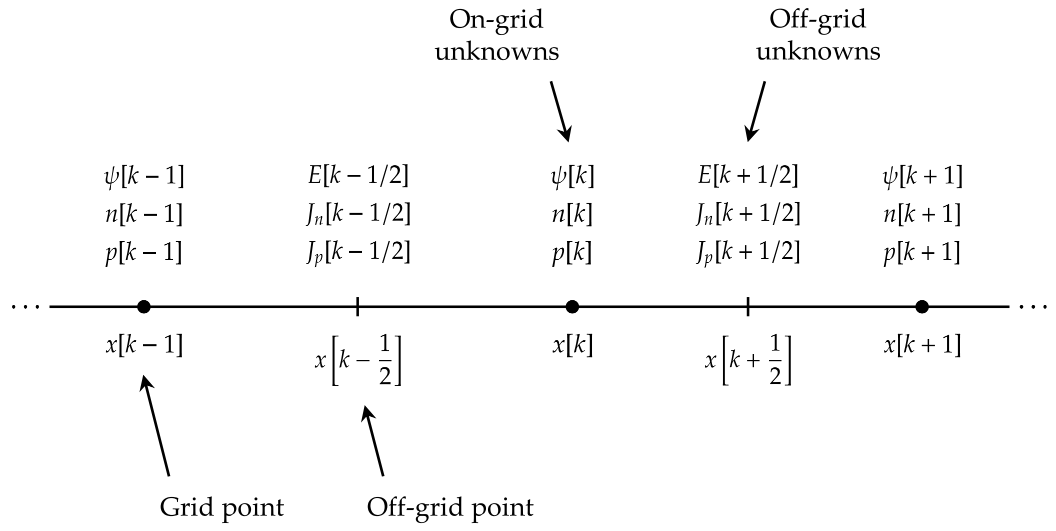


Figure 2.5: One-dimensional discretization grid for Van Roosbroeck system, showing quantities defined at grid points ("on-grid unknowns") and quantities defined between grid points ("off-grid unknowns").

Since the central difference provides an accurate approximation to f' between the points $x[k-1]$ and $x[k+1]$, it is natural to assume that the forward difference is accurate between the points $x[k]$ and $x[k+1]$, and similarly for the reverse difference. This is indeed the case, and we may denote the central difference of f at the midpoint of $x[k]$ and $x[k+1]$ as

$$f'\left[k+\frac{1}{2}\right] \triangleq f'\left(x\left[k+\frac{1}{2}\right]\right) = \frac{f[k+1]-f[k]}{x[k+1]-x[k]}, \quad \text{where} \quad x\left[k+\frac{1}{2}\right] \triangleq \frac{x[k+1]+x[k]}{2}. \tag{2.26}$$

Using (2.26), we may evaluate the second derivative of f at $x[k]$:

$$f''[k] = \frac{f'[k+1/2]-f'[k-1/2]}{x[k+1/2]-x[k-1/2]} = \frac{2}{x[k+1]-x[k-1]}\left(\frac{f[k+1]-f[k]}{x[k+1]-x[k]} - \frac{f[k]-f[k-1]}{x[k]-x[k-1]}\right). \tag{2.27}$$

The expression in (2.27) is considerably simpler than that in (2.25) and requires only 3 values of f per computation. This approach implies the use of a two-level grid, where functions are defined at grid points $x[i]$ and their derivatives are defined at the midpoints of grid intervals $x[i+1/2]$, where i is an integer [48, 158].

We will now apply this approach to the system in (2.6), using only one spatial dimension for simplicity. The grid used in this discretization is shown in Figure 2.5. The potential ψ and carrier concentrations n and p are defined at grid points, and the electric field $E=-\psi'$ and current densities J_n and J_p are defined between grid points. The Poisson equation in 1D reads

$$(-\varepsilon\psi')' + q(n-p-\Gamma) = 0. \tag{2.28}$$

Assuming ε is constant between grid points, (2.28) can be discretized at $x = x[k]$ by substituting the central difference approximation for $(-\varepsilon\psi')'$:

$$-\frac{2}{x[k+1]-x[k-1]}\left(\varepsilon\left[k+\frac{1}{2}\right]\frac{\psi[k+1]-\psi[k]}{x[k+1]-x[k]} - \varepsilon\left[k-\frac{1}{2}\right]\frac{\psi[k]-\psi[k-1]}{x[k]-x[k-1]}\right) + q(n[k]-p[k]-\Gamma[k]) = 0. \tag{2.29}$$

The electron and hole continuity equations in 1D are

$$G - U + \frac{J_n'}{q} = 0, \quad G - U - \frac{J_p'}{q} = 0, \tag{2.30}$$

where

$$J_n = -q\mu_n(n\psi' - V_T n'), \quad J_p = -q\mu_p(n\psi' + V_T p').$$

At each node, the electron continuity equation in (2.30) gives

$$G[k] - U[k] + \frac{J_n'[k]}{q} = 0, \tag{2.31}$$

and a similar discretization holds for the hole continuity equation. We apply the central difference approximation to $J_n'[k]/q$ in (2.31) to yield

$$\frac{J_n'[k]}{q} = \frac{2}{x[k+1] - x[k-1]}\left(\frac{J_n[k+1/2] - J_n[k-1/2]}{q}\right), \tag{2.32}$$

where

$$\begin{aligned}\frac{J_n[k+1/2]}{q} &= \mu_n\left[k+\frac{1}{2}\right]\left(n\left[k+\frac{1}{2}\right]\frac{\psi[k]-\psi[k+1]}{x[k+1]-x[k]} + V_T\frac{n[k+1]-n[k]}{x[k+1]-x[k]}\right)\\ &= \frac{\mu_n[k+1/2]V_T}{x[k+1]-x[k]}\left(n\left[k+\frac{1}{2}\right]\frac{\psi[k]-\psi[k+1]}{V_T} + n[k+1]-n[k]\right)\end{aligned} \tag{2.33}$$

The term $n[k+1/2]$ in (2.33) is problematic since we have assumed that the electron concentration n is only defined *at* grid points, not *between* grid points. An intuitive approach would set

$$n\left[k+\frac{1}{2}\right] = \frac{n[k+1]+n[k]}{2},$$

using the arithmetic mean of n between $x[k]$ and $x[k+1]$. This approach implicitly assumes that n varies linearly between grid points, which is true only of the potential ψ [3, 159]. By contrast, n and p can vary over several orders of magnitude along the length of a device, rendering the assumption of linear variation inaccurate.

Many approaches exist to estimate n and p between grid points. The most well-known of these approaches was introduced as an appendix to a 1969 paper by Scharfetter and Gummel [135]. The derivation of $n[k+1/2]$ is shown here for electrons; analogous expressions hold for holes. We assume that the current density J_n and the electric field $E = -\psi'$ are constant from $x[k]$ to $x[k+1]$, which is reasonable in most physical simulations [44]. We further assume that the electron and hole mobilities are only field-dependent; that is, $\mu_n = \mu_n(E)$ and $\mu_p = \mu_p(E)$. Let $x_0 \in (x[k], x[k+1])$ be the point at which the central difference approximation for n' holds. From (2.4), the current density at $x = x_0$ is

$$J_n(x_0) = q\mu_n\left[k+\frac{1}{2}\right]\left(n(x_0)E\left[k+\frac{1}{2}\right] + V_T n'(x_0)\right).$$

Substituting the central difference approximation of $n'(x_0)$,

$$J_n(x_0) = q\mu_n\left[k+\frac{1}{2}\right]\left(n(x_0)E\left[k+\frac{1}{2}\right] + V_T\left(\frac{n[k+1]-n[k]}{x[k+1]-x[k]}\right)\right). \tag{2.34}$$

Since J_n is constant between $x[k]$ and $x[k+1]$ by assumption, (2.34) must also hold for any $n(x)$ and $n'(x)$ where $x \in (x[k], x[k+1])$. We may equate this value of J_n with the value of J_n at the point $x = x_0$:

$$n(x)E\left[k+\frac{1}{2}\right] + V_T n'(x) = n(x_0)E\left[k+\frac{1}{2}\right] + V_T\left(\frac{n[k+1]-n[k]}{x[k+1]-x[k]}\right).$$

This expression can be rearranged to yield a first-order ODE in $n(x)$:

$$n'(x) + \frac{E[k+1/2]}{V_T}n(x) - \left(\frac{E[k+1/2]}{V_T}n(x_0) + \frac{n[k+1]-n[k]}{x[k+1]-x[k]}\right) = 0. \tag{2.35}$$

Solving the ODE in (2.35) gives

$$\begin{aligned} n(x) &= n(x_0) + \frac{V_T}{E[k+1/2]}\left(\frac{n[k+1]-n[k]}{x[k+1]-x[k]}\right) + c\exp\left(-\frac{E[k+1/2]}{V_T}x\right) \\ &= n(x_0) - V_T\left(\frac{n[k+1]-n[k]}{\psi[k+1]-\psi[k]}\right) + c\exp\left(\frac{\psi[k+1]-\psi[k]}{V_T}\frac{x}{x[k+1]-x[k]}\right), \end{aligned} \tag{2.36}$$

where c is a constant. Since (2.36) also holds at the endpoints of the interval $x = x[k]$ and $x[k+1]$, we must have $n(x[k]) = n[k]$ and $n(x[k+1]) = n[k+1]$, which gives a system of equations in the unknown quantities c and $n(x_0)$:

$$\begin{cases} n[k] & = n(x_0) - V_T\left(\frac{n[k+1]-n[k]}{\psi[k+1]-\psi[k]}\right) + c\exp\left(\frac{\psi[k+1]-\psi[k]}{V_T}\frac{x[k]}{x[k+1]-x[k]}\right) \\ n[k+1] & = n(x_0) - V_T\left(\frac{n[k+1]-n[k]}{\psi[k+1]-\psi[k]}\right) + c\exp\left(\frac{\psi[k+1]-\psi[k]}{V_T}\frac{x[k+1]}{x[k+1]-x[k]}\right) \end{cases}. \tag{2.37}$$

Solving the system in (2.37) gives

$$c = \frac{n[k+1]-n[k]}{\exp\left(\frac{\psi[k+1]-\psi[k]}{V_T}\frac{x[k+1]}{x[k+1]-x[k]}\right) - \exp\left(\frac{\psi[k+1]-\psi[k]}{V_T}\frac{x[k]}{x[k+1]-x[k]}\right)} \tag{2.38}$$

and

$$\begin{aligned} n(x_0) = n[k]&\left(1 - \frac{V_T}{\psi[k+1]-\psi[k]} + \frac{1}{\exp\left(\frac{\psi[k+1]-\psi[k]}{V_T}\right)-1}\right) + \\ &n[k+1]\left(\frac{V_T}{\psi[k+1]-\psi[k]} - \frac{1}{\exp\left(\frac{\psi[k+1]-\psi[k]}{V_T}\right)-1}\right). \end{aligned} \tag{2.39}$$

At this point, two approaches exist to define $n[k+1/2]$. We could set $n[k+1/2] = n(x[k+1/2])$, choosing the value of n that lies halfway between $x[k]$ and $x[k+1]$. While intuitive, this approach would be inconsistent with our earlier assumption that the finite difference approximation for n' holds only at $x = x_0$. Another approach would be to accept the value given in (2.39) as $n[k+1/2]$, with the knowledge that this value is generally different from the value at the midpoint. The latter approach is favored in modern simulators, and indeed the definition $n[k+1/2] \triangleq n(x_0)$ has been definitively accepted as *the* Scharfetter-Gummel discretization [44, 82]. This derivation will be presented below; an alternate derivation using the "midpoint rule" $n[k+1/2] = n(x[k+1/2])$ is presented in Appendix A.1.

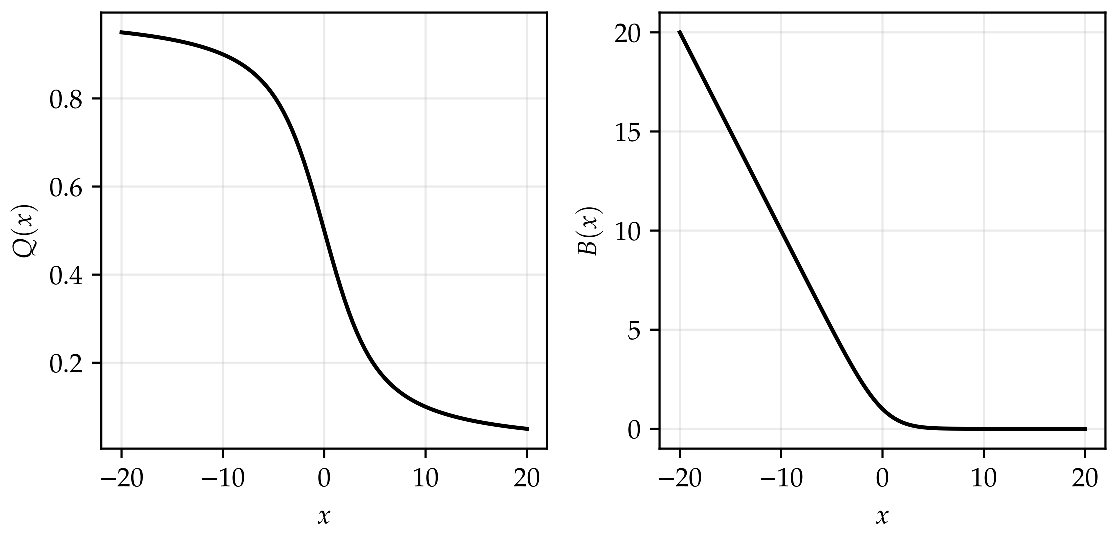


Figure 2.6: Plot of $Q(x)$ and Bernoulli function $B(x)$ for $x \in (-20, 20)$.

Using (2.39) and applying a similar approach for $n[k-1/2]$, we can write

$$n\left[k+\frac{1}{2}\right] \triangleq n[k]\left(1-Q(\delta_r)\right) + n[k+1]Q(\delta_r), \tag{2.40}$$

$$n\left[k-\frac{1}{2}\right] \triangleq n[k-1]\left(1-Q(\delta_l)\right) + n[k]Q(\delta_l), \tag{2.41}$$

where

$$\delta_r \triangleq \frac{\psi[k+1]-\psi[k]}{V_T}, \quad \delta_l \triangleq \frac{\psi[k]-\psi[k-1]}{V_T} \tag{2.42}$$

and

$$Q(x) \triangleq \frac{1}{x} - \frac{1}{e^x - 1}. \tag{2.43}$$

An analogous process for the hole continuity equation gives

$$p\left[k+\frac{1}{2}\right] \triangleq p[k]Q(\delta_r) + p[k+1]\left(1-Q(\delta_r)\right), \tag{2.44}$$

$$p\left[k-\frac{1}{2}\right] \triangleq p[k-1]Q(\delta_l) + p[k]\left(1-Q(\delta_l)\right). \tag{2.45}$$

The function $Q(x)$ as defined in (2.43) smoothly constructs a weighted average of, for example, $n[k]$ and $n[k+1]$ to produce $n[k+1/2]$, where the weights depend on the potential difference δ_r. This function is analytic, but the singularity at $x = 0$ makes numerical evaluation difficult when $|x| \approx 0$. This difficulty can be circumvented by switching from direct evaluation to a Taylor expansion of $Q(x)$ for small x, namely:

$$Q(x) = \frac{1}{2} - \frac{x}{12} + \frac{x^3}{720} - \frac{x^5}{30\,240} + \frac{x^7}{1\,209\,600} - \frac{x^9}{47\,900\,160} + \cdots. \tag{2.46}$$

This function is shown for $x \in (-20, 20)$ in Figure 2.6.

We are now in a position to fully discretize the continuity equations. Using (2.33) and applying a similar approach for the left midpoint, we can determine $J_n[k+1/2]$ and $J_n[k-1/2]$:

$$J_n\left[k+\frac{1}{2}\right] = \frac{\mu_n[k+1/2]V_T}{x[k+1]-x[k]}\left(-n\left[k+\frac{1}{2}\right]\delta_r + n[k+1] - n[k]\right),$$
$$J_n\left[k-\frac{1}{2}\right] = \frac{\mu_n[k-1/2]V_T}{x[k]-x[k-1]}\left(-n\left[k-\frac{1}{2}\right]\delta_l + n[k] - n[k-1]\right).$$

An analogous process for the hole continuity equation gives

$$J_p\left[k+\frac{1}{2}\right] = -\frac{\mu_p[k+1/2]V_T}{x[k+1]-x[k]}\left(p\left[k+\frac{1}{2}\right]\delta_r + p[k+1] - p[k]\right),$$
$$J_p\left[k-\frac{1}{2}\right] = -\frac{\mu_p[k-1/2]V_T}{x[k]-x[k-1]}\left(p\left[k-\frac{1}{2}\right]\delta_l + p[k] - p[k-1]\right).$$

We now compute $J_n'[k]/q$ from (2.32). Substituting $n[k+1/2]$ and $n[k-1/2]$ from (2.40) and (2.41),

$$\begin{aligned}\frac{J_n'[k]}{q} = \frac{2}{x[k+1]-x[k-1]}\Bigg(&\frac{\mu_n[k-1/2]V_T}{x[k]-x[k-1]}\left(1+\delta_l-\delta_l Q(\delta_l)\right)n[k-1] \\ &- \frac{\mu_n[k-1/2]V_T}{x[k]-x[k-1]}\left(1-\delta_l Q(\delta_l)\right)n[k] - \frac{\mu_n[k+1/2]V_T}{x[k+1]-x[k]}\left(1+\delta_r-\delta_r Q(\delta_r)\right)n[k] \\ &- \frac{\mu_n[k+1/2]V_T}{x[k+1]-x[k]}\left(1-\delta_r Q(\delta_r)\right)n[k+1]\Bigg).\end{aligned} \tag{2.47}$$

An analogous process for the hole continuity equation gives $J_p'[k]/q$. Substituting $p[k+1/2]$ and $p[k-1/2]$ from (2.44) and (2.45),

$$\begin{aligned}\frac{J_p'[k]}{q} = -\frac{2}{x[k+1]-x[k-1]}\Bigg(&\frac{\mu_p[k-1/2]V_T}{x[k]-x[k-1]}\left(1-\delta_l Q(\delta_l)\right)p[k-1] \\ &- \frac{\mu_p[k-1/2]V_T}{x[k]-x[k-1]}\left(1+\delta_l-\delta_l Q(\delta_l)\right)p[k] - \frac{\mu_p[k+1/2]V_T}{x[k+1]-x[k]}\left(1-\delta_r Q(\delta_r)\right)p[k] \\ &- \frac{\mu_p[k+1/2]V_T}{x[k+1]-x[k]}\left(1+\delta_r-\delta_r Q(\delta_r)\right)p[k+1]\Bigg).\end{aligned} \tag{2.48}$$

Next, we define the *Bernoulli function*

$$B(x) = \frac{x}{e^x-1}. \tag{2.49}$$

The Bernoulli function is similar to the function $Q(x)$ in that it has a removable singularity at the origin, which poses numerical difficulty for small x. For such x, we may use the Taylor expansion

$$B(x) = 1 - \frac{x}{2} + \frac{x^2}{12} - \frac{x^4}{720} + \frac{x^6}{30\,240} - \frac{x^8}{1\,209\,600} + \frac{x^{10}}{47\,900\,160} - \cdots. \tag{2.50}$$

This function is shown for $x \in (-20, 20)$ in Figure 2.6.

From the definition of $B(x)$ in (2.49), we note the identities

$$B(x) = 1 - xQ(x), \quad B(-x) = 1 + x - xQ(x). \tag{2.51}$$

Using (2.51), we may rewrite (2.47) and (2.48) in terms of $B(x)$. Combined with (2.29), this gives the final system of three equations to be solved at each interior node in the simulation:

$$F_\psi[k] = -q(n[k] - p[k] - \Gamma[k]) + \frac{2}{x[k+1] - x[k-1]} \Bigg(\varepsilon\left[k - \frac{1}{2}\right] \frac{\psi[k-1] - \psi[k]}{x[k] - x[k-1]} - \varepsilon\left[k + \frac{1}{2}\right] \frac{\psi[k] - \psi[k+1]}{x[k+1] - x[k]} \Bigg) = 0, \quad (2.52)$$

$$F_n[k] = G[k] - U[k] + \frac{2}{x[k+1] - x[k-1]} \Bigg(\frac{\mu_n[k-1/2]V_T}{x[k] - x[k-1]} \Big(B(-\delta_l)n[k-1] - B(\delta_l)n[k] \Big) - \frac{\mu_n[k+1/2]V_T}{x[k+1] - x[k]} \Big(B(-\delta_r)n[k] - B(\delta_r)n[k+1] \Big) \Bigg) = 0, \quad (2.53)$$

$$F_p[k] = G[k] - U[k] + \frac{2}{x[k+1] - x[k-1]} \Bigg(\frac{\mu_p[k-1/2]V_T}{x[k] - x[k-1]} \Big(B(\delta_l)p[k-1] - B(-\delta_l)p[k] \Big) - \frac{\mu_p[k+1/2]V_T}{x[k+1] - x[k]} \Big(B(\delta_r)p[k] - B(-\delta_r)p[k+1] \Big) \Bigg) = 0. \quad (2.54)$$

In a 1D grid with N nodes, the non-equilibrium simulation can thus be completed by solving equations (2.52) through (2.54) for nodes $k = 2$ through $k = N - 1$ and applying Dirichlet boundary conditions at nodes $k = 1$ and $k = N$. The result is a system of $3N$ equations in $3N$ unknowns which yields ψ, n and p at each grid point when solved.

2.2.2 Finite volume methods

With the 1D discretization complete, it is natural to wonder if the finite difference scheme might generalize to higher dimensions. The answer is complicated; finite difference methods do exist in 2D and 3D, but their applications are increasingly limited since many assumptions made in the 1D case do not easily generalize to higher dimensions. Some problems such as the Laplace problem with Dirichlet boundary conditions

$$\nabla^2 f(\mathbf{x}) = 0, \quad \text{where} \quad \mathbf{x} \in \Omega$$

are easily discretized using finite differences in 2D and 3D [158]. Others, particularly those requiring Neumann, Robin or mixed boundary conditions, are awkward to solve using finite differences and often require the use of *ghost nodes*, fictitious grid nodes outside the device boundary that must be included in the discretization [41, 160].

The difficulty of solving nonlinear PDE systems in 2D and 3D has motivated several discretization methods that are used in physics simulators today. The most common of these methods are the *finite element method* and the *finite volume method*. Instead of solving for the unknowns directly

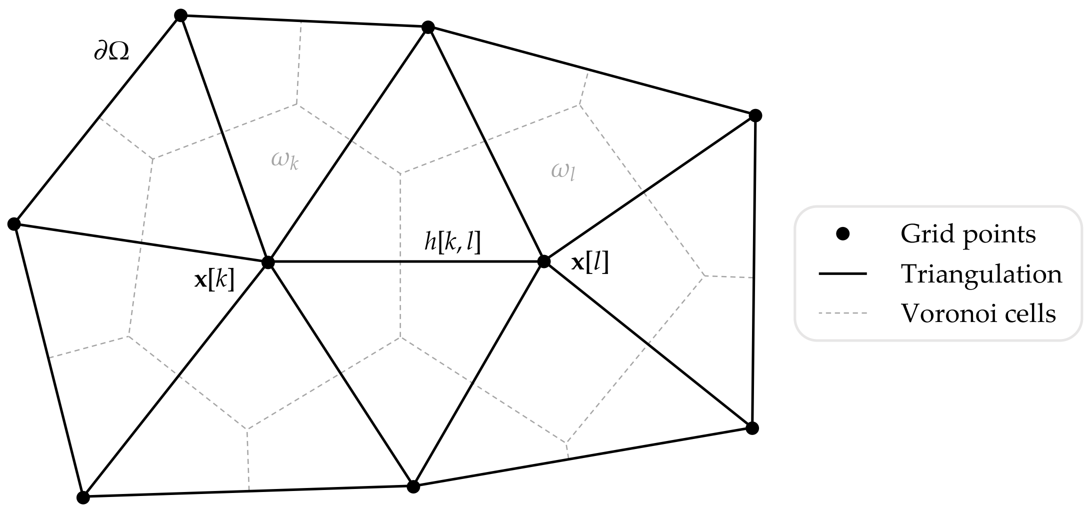


Figure 2.7: Example two-dimensional discretization grid for finite volume method with two interior nodes, $\mathbf{x}[k]$ and $\mathbf{x}[l]$. The boundary of the discretization is $\partial\Omega$.

as in a finite difference scheme, the finite element method treats the unknown functions as a linear combination of *shape functions* with unknown weights. The problem then is to determine the weights themselves. The details of the finite element method will not be discussed here; while many physics simulators have historically used the finite element method, modern semiconductor simulators often avoid this method due to its complexity and sensitivity to obtuse angles in meshes [5, 38].

The finite volume method, occasionally referred to as the *box method* in the literature, circumvents many of the difficulties associated with implementing finite element methods [3, 14, 45, 75]. For the Van Roosbroeck system in particular, the finite volume method guarantees the positivity of carrier concentrations and the enforcement of the maximum principle [38]. Additionally, the finite volume method can be more intuitive than the finite element method and in many cases produces identical discretizations to a finite difference scheme. The finite volume method can also be used in time-dependent simulations. A brief introduction to the method will be given here, following many of the details from [38] and [45].

Consider the discretization grid shown in Figure 2.7. This grid contains 2 *interior points* $\mathbf{x}[k]$ and $\mathbf{x}[l]$ which do not lie on the boundary of the discretization ($\partial\Omega$) and 7 *exterior points* which lie on $\partial\Omega$. The grid points are connected to form triangles such that no grid point lies in the circumcircle of any other triangle, forming what is known as a *Delaunay triangulation*. The regions bounded by the perpendicular bisectors of the edges of each triangle are called *Voronoi cells*, and the resulting tessellation is called a *Voronoi diagram*. When the edge of a triangle lies on $\partial\Omega$, the Voronoi cells containing that edge will intersect with $\partial\Omega$. The Voronoi diagram has the property that the Voronoi cell around each node contains all points closer to that node than to any other node. The Delaunay triangulation can be generated using methods such as those described in [133] and [137]. In 3D, the triangulation is replaced by a *tetrahedralization*, which can be generated according to, for example, [59].

The formulation of the finite volume method from [45] is repeated here, using the notation of this thesis for clarity. Let $\nabla \cdot \mathbf{j}(u) + f(u) = 0$ be a (generally) nonlinear PDE to be solved for

$\mathbf{x} \in \Omega$, where $u(\mathbf{x})$ is an unknown function. The function $\mathbf{j}(u)$ describes the flux of u through the control volumes, and $f(u)$ is the *reaction term*, so called due to its use in chemical reaction systems. Integrating the PDE over a control volume ω_k,

$$\oiint_{\omega_k} (\nabla \cdot \mathbf{j}(u) - f(u)) \, \mathrm{d}\omega = 0 \implies \oiint_{\omega_k} \nabla \cdot \mathbf{j}(u) \, \mathrm{d}\omega + \oiint_{\omega_k} f(u) \, \mathrm{d}\omega = 0. \tag{2.55}$$

Applying Gauss's theorem to the first term in (2.55) gives

$$\oiint_{\omega_k} \nabla \cdot \mathbf{j}(u) \, \mathrm{d}\omega = \oint_{\partial\omega_k} \mathbf{j}(u) \cdot \hat{\mathbf{n}} \, \mathrm{d}s \,, \tag{2.56}$$

where $\hat{\mathbf{n}}$ is the outward-facing normal to $\partial\omega_k$. Since the line integral runs along a number of straight line segments forming $\partial\omega_k$, we can rewrite (2.56) as

$$\oiint_{\omega_k} \nabla \cdot \mathbf{j}(u) \, \mathrm{d}\omega = \sum_{l \in \mathcal{N}[k]} \int_{\partial\omega_k \cap \partial\omega_l} \mathbf{j}(u) \cdot \hat{\mathbf{n}}[k,l] \, \mathrm{d}s \,, \quad \text{where} \quad \hat{\mathbf{n}}[k,l] \triangleq \frac{\mathbf{x}[l] - \mathbf{x}[k]}{\|\mathbf{x}[l] - \mathbf{x}[k]\|}, \tag{2.57}$$

and $\mathcal{N}[k]$ is the set of neighboring control volumes to ω_k. We have assumed in (2.57) that there is no outward flux of u through $\partial\Omega$; that is, that

$$\int_{\partial\omega_k \cap \partial\Omega} \mathbf{j}(u) \cdot \hat{\mathbf{n}} \, \mathrm{d}s = 0.$$

This is typical in semiconductor problems, since we enforce the homogeneous Neumann boundary conditions $\mathbf{E} \cdot \hat{\mathbf{n}} = \mathbf{J_n} \cdot \hat{\mathbf{n}} = \mathbf{J_p} \cdot \hat{\mathbf{n}} = 0$ along $\partial\Omega$. Next, for each neighboring control volume ω_l, we define the *flux approximation* $g_j[k,l]$ such that

$$g_j[k,l] \approx (\mathbf{j}[k,l] \cdot \hat{\mathbf{n}}[k,l]) \left\| \mathbf{x}[l] - \mathbf{x}[k] \right\|, \tag{2.58}$$

where $\mathbf{j}[k,l]$ is the value of $\mathbf{j}(u)$ along the boundary segment $\partial\omega_k \cap \partial\omega_l$. Consequently, we can rewrite

$$\int_{\partial\omega_k \cap \partial\omega_l} \mathbf{j}(u) \cdot \hat{\mathbf{n}}[k,l] \, \mathrm{d}s \approx \left(\frac{|\partial\omega_k \cap \partial\omega_l|}{\|\mathbf{x}[l] - \mathbf{x}[k]\|} \right) g_j[k,l], \tag{2.59}$$

where $|\partial\omega_k \cap \partial\omega_l|$ is the length of the boundary segment. Substituting (2.59) into (2.57) gives

$$\oiint_{\omega_k} \nabla \cdot \mathbf{j}(u) \, \mathrm{d}\omega \approx \sum_{l \in \mathcal{N}[k]} \left(\frac{|\partial\omega_k \cap \partial\omega_l|}{\|\mathbf{x}[l] - \mathbf{x}[k]\|} \right) g_j[k,l]. \tag{2.60}$$

This definition of $g_j[k,l]$ is convenient for PDEs of the form $\nabla \cdot (D\nabla u) - f(u) = 0$, where D is a diffusion coefficient [45]. In this case $g_j[k,l] = D(u[l] - u[k])$, which is only possible since the Voronoi diagram guarantees that $\mathbf{x}[l] - \mathbf{x}[k]$ is perpendicular to the boundary segment $\partial\omega_k \cap \partial\omega_l$, allowing a finite difference approximation of the gradient. Additionally, this definition guarantees flux conservation across each boundary segment when $g_j[k,l] = -g_j[l,k]$.

Next, we seek to approximate the second term in (2.55). Assuming $f(u)$ is constant in each control volume,

$$\oiint_{\omega_k} f(u) \, \mathrm{d}\omega = |\omega_k| f(u[k]) = |\omega_k| f[k], \tag{2.61}$$

where $|\omega_k|$ is the area of the control volume ω_k. Combining (2.55), (2.60) and (2.61) gives the algebraic equation

$$\sum_{l \in \mathcal{N}[k]} \left(\frac{|\partial\omega_k \cap \partial\omega_l|}{\|\mathbf{x}[l] - \mathbf{x}[k]\|} \right) g_j[k,l] + |\omega_k| f[k] = 0. \tag{2.62}$$

Equation (2.62) is solved at every node except those where Dirichlet boundary conditions are specified. At nodes where Dirichlet boundary conditions are specified, we simply solve $u[k]-b[k] = 0$, where $b[k]$ is the desired value of $u[k]$. This completes the discretization. In a 2D grid with N nodes, this procedure yields a system of N nonlinear equations in N unknowns which yields u at each grid point when solved. This procedure easily generalizes to coupled systems of PDEs, where the discretization is repeated for each unknown function, resulting in a correspondingly larger system of equations.

We now derive the finite volume discretization for the Van Roosbroeck system in 2D. This discretization requires a flux discretization g and a reaction discretization f for each of the unknown functions ψ, n and p. We will denote the flux discretizations by g_ψ, g_n and g_p, and the reaction discretizations by f_ψ, f_n and f_p. Similarly to the 1D finite difference scheme, the flux discretization requires access to the values of $\mathbf{E} = -\nabla\psi$, $\mathbf{J_n}$ and $\mathbf{J_p}$ along the boundary segments $\partial\omega_k \cap \partial\omega_l$, which are generally between grid points. For the Poisson equation, we again assume a linear variation in ψ between grid points, which implies an electric field that is constant on each triangle of the discretization (but not generally on each control volume). As with the finite difference scheme, this assumption is reasonable for most semiconductor problems [123]. We may thus use the discretizations

$$g_\psi[k,l] = \varepsilon[k,l]\left(\psi[k] - \psi[l]\right), \quad f_\psi[k] = q\left(n[k] - p[k] - \Gamma[k]\right), \tag{2.63}$$

where $\varepsilon[k,l]$ is the permittivity assigned to the triangle containing the edge between $\mathbf{x}[k]$ and $\mathbf{x}[l]$. Note that this quantity can be different depending on which triangle the discretization is invoked from, since a single edge typically belongs to two triangles. This allows the permittivity to be piecewise constant, which can be useful in the simulation of heterojunction devices [109, 121].

As with the finite difference scheme, the discretization of the electron and hole continuity equations is more involved. Similar numerical difficulties occur in 2D if simple arithmetic means are used to estimate n and p between grid points. This would motivate a 2D scheme similar to the 1D Scharfetter-Gummel scheme discussed earlier. Such a scheme would seek to find n and p such that $\mathbf{J_n}$ and $\mathbf{J_p}$ were constant on each triangle in the discretization. However, it was shown in [123] that this requires

$$\frac{E_y}{E_x} = \frac{J_{ny}}{J_{nx}}$$

for the electron continuity equation, where E_x, E_y, J_{nx} and J_{ny} are the x and y components of $\mathbf{E}$ and $\mathbf{J_n}$, respectively. In other words, the Scharfetter-Gummel discretization does not exist in two dimensions unless $\mathbf{E}$ and $\mathbf{J_n}$ are collinear. Thus, we must relax the assumption that the current densities are constant on each triangle. A suitable assumption is instead that the current densities are constant on each *edge* of the discretization but are allowed to vary within each triangle. Under this assumption, the estimation of n and p between grid points is identical to the Scharfetter-Gummel scheme in 1D, leading to the discretizations

$$g_n[k,l] = \mu_n[k,l] V_T \left(B\left(-\frac{\psi[l] - \psi[k]}{V_T}\right) n[k] - B\left(\frac{\psi[l] - \psi[k]}{V_T}\right) n[l] \right), \quad f_n[k] = U[k] - G[k], \tag{2.64}$$

where μ_n has the value $\mu_n[k,l]$ in the triangle from which the discretization is invoked. Similar discretizations result for the hole continuity equation:

$$g_p[k,l] = \mu_p[k,l]V_T\left(B\left(\frac{\psi[l]-\psi[k]}{V_T}\right)p[k] - B\left(-\frac{\psi[l]-\psi[k]}{V_T}\right)p[l]\right), \quad f_p[k] = U[k] - G[k], \tag{2.65}$$

where $\mu_p[k,l]$ is defined similarly to $\mu_n[k,l]$. This approach allows μ_n and μ_p to vary along the device, which is necessary for simulating a field-dependent mobility [81].

We will now show that the finite volume discretization defined by (2.63) through (2.65) in 1D yields identical residuals to the finite difference residuals in (2.52) through (2.54). We first note that in 1D, the Voronoi boundary segments are zero-dimensional, so $\partial\omega_k \cap \partial\omega_l = 1$ for all k, l. Each node $x[k]$ has an associated Voronoi cell bounded by the midpoints of the grid segments to its left and right, such that

$$|\omega_k| = \frac{x[k+1]-x[k-1]}{2}.$$

Additionally, the term $\varepsilon[k,l]$ reduces to $\varepsilon[k+1/2]$ or $\varepsilon[k-1/2]$, since each node has only two neighbors and $\mathcal{N}[k] = \{k-1, k+1\}$. Similar notation applies for $\mu_n[k,l]$ and $\mu_p[k,l]$. We also note that $g_\psi[k,l] = -g_\psi[l,k]$, $g_n[k,l] = -g_n[l,k]$ and $g_p[k,l] = -g_p[l,k]$ due to charge conservation. Substituting (2.63), (2.64) and (2.65) into (2.62) gives the residuals

$$\begin{aligned}F_\psi[k] = &-\frac{\varepsilon[k-1/2]}{x[k]-x[k-1]}\left(\psi[k-1]-\psi[k]\right) + \frac{\varepsilon[k+1/2]}{x[k+1]-x[k]}\left(\psi[k]-\psi[k+1]\right) + \\ &\frac{x[k+1]-x[k-1]}{2}\left(q\left(n[k]-p[k]-\Gamma[k]\right)\right) = 0,\end{aligned} \tag{2.66}$$

$$\begin{aligned}F_n[k] = &-\frac{\mu_n[k-1/2]V_T}{x[k]-x[k-1]}\Big(B(-\delta_l)n[k-1] - B(\delta_l)n[k]\Big) + \\ &\frac{\mu_n[k+1/2]V_T}{x[k+1]-x[k]}\Big(B(-\delta_r)n[k] - B(\delta_r)n[k+1]\Big) + \\ &\frac{x[k+1]-x[k-1]}{2}\left(U[k]-G[k]\right) = 0,\end{aligned} \tag{2.67}$$

$$\begin{aligned}F_p[k] = &-\frac{\mu_p[k-1/2]V_T}{x[k]-x[k-1]}\Big(B(\delta_l)p[k-1] - B(-\delta_l)p[k]\Big) + \\ &\frac{\mu_p[k+1/2]V_T}{x[k+1]-x[k]}\Big(B(\delta_r)p[k] - B(-\delta_r)p[k+1]\Big) + \\ &\frac{x[k+1]-x[k-1]}{2}\left(U[k]-G[k]\right) = 0.\end{aligned} \tag{2.68}$$

Equations (2.66) through (2.68) are identical to (2.52) through (2.54) up to a scale factor of $-2/(x[k+1]-x[k-1])$.

Beyond the classical Scharfetter-Gummel scheme, other flux discretizations exist for the carrier concentrations n and p. The midpoint scheme discussed in Appendix A.1 can be generalized to higher dimensions using the flux discretizations

$$\begin{aligned}g_n[k,l] = \mu_n[k,l]V_T\Bigg(&\left(1+\frac{\psi[l]-\psi[k]}{V_T}\left(1-\sigma\left(\frac{\psi[l]-\psi[k]}{V_T}\right)\right)\right)n[k] - \\ &\left(1-\frac{\psi[l]-\psi[k]}{V_T}\sigma\left(\frac{\psi[l]-\psi[k]}{V_T}\right)\right)n[l]\Bigg),\end{aligned}$$

$$g_p[k,l] = \mu_p[k,l]V_T\left(\left(1 - \frac{\psi[l]-\psi[k]}{V_T}\sigma\left(\frac{\psi[l]-\psi[k]}{V_T}\right)\right)p[k] - \left(1 + \frac{\psi[l]-\psi[k]}{V_T}\left(1-\sigma\left(\frac{\psi[l]-\psi[k]}{V_T}\right)\right)\right)p[l]\right),$$

where $\sigma(x)$ is the sigmoid function given in (A.3). Another discretization was introduced by Slotboom in a 1973 paper on 2D simulation of BJTs [142]. The *Slotboom discretization* uses the flux functions

$$g_n[k,l] = \mu_n[k,l]V_T\left(S\left(-\frac{\psi[l]-\psi[k]}{V_T}\right)n[k] - S\left(\frac{\psi[l]-\psi[k]}{V_T}\right)n[l]\right),$$
$$g_p[k,l] = \mu_p[k,l]V_T\left(S\left(\frac{\psi[l]-\psi[k]}{V_T}\right)p[k] - S\left(-\frac{\psi[l]-\psi[k]}{V_T}\right)p[l]\right),$$

where $S(x) = e^{-x/2}$. This discretization is identical to the Scharfetter-Gummel discretization in (2.64) and (2.65), except the Bernoulli function is replaced by $S(x)$. The Slotboom discretization is derived similarly to the Scharfetter-Gummel discretization, except $J_n[x+1/2]$ is estimated using an integrating factor. This discretization has been shown to yield more accurate I-V curves in 1D semiconductors compared to the Scharfetter-Gummel scheme [47]. Further extensions to the Scharfetter-Gummel scheme include discretizations using non-Boltzmann statistics [36] and discretizations designed to reduce error due to the *crosswind effect*, which arises when the current densities are not parallel to grid segments [61, 148].

2.2.3 Nonlinear solvers

The output of any PDE discretization is a large system of nonlinear algebraic equations that must be solved to yield an approximate solution of the PDE system. Several *nonlinear solvers* exist for this purpose. A nonlinear solver considers equations of the form $\mathbf{F}(\mathbf{z}) = 0$, where $\mathbf{F}$ is a vector-valued function of the vector-valued unknown $\mathbf{z}$. In the case of semiconductor problems, we typically write

$$\mathbf{z} = \begin{bmatrix}\psi[1] & n[1] & p[1] & \psi[2] & n[2] & p[2] & \cdots & \psi[N] & n[N] & p[N]\end{bmatrix}^T, \tag{2.69}$$

where N is the number of nodes in the discretization. The function $\mathbf{F}$ is the residual

$$\mathbf{F}(\mathbf{z}) = \begin{bmatrix}F_\psi[1] & F_n[1] & F_p[1] & F_\psi[2] & F_n[2] & F_p[2] & \cdots & F_\psi[N] & F_n[N] & F_p[N]\end{bmatrix}^T, \tag{2.70}$$

where $F_\psi[k]$, $F_n[k]$ and $F_p[k]$ are given by, for example, (2.66) through (2.68) for a 1D discretization. Nonlinear solvers typically require access to the *Jacobian* $\mathbf{J} = \partial\mathbf{F}(\mathbf{z})/\partial\mathbf{z}$. The Jacobian is an $N \times N$ matrix that can be computed either manually (as in Appendix A.2 for 1D problems) or by using automatic differentiation or (less commonly) finite differences. Since the Jacobian is typically sparse, techniques such as *graph coloring* are often used for computational efficiency [46].

Newton's method is one of the simplest and most widely used linear solvers, particularly for systems arising from discretized PDEs. Beginning with an initial guess $\mathbf{z} = \mathbf{z}_0$, each step of Newton's method computes

$$\mathbf{z}_{n+1} = \mathbf{z}_n - (\mathbf{J}_n)^{-1}\mathbf{F}_n, \tag{2.71}$$

where $\mathbf{J}_n$ and $\mathbf{F}_n$ are the Jacobian and residual computed using the values of $\mathbf{z}$ from the previous iteration. In practice, the Jacobian inverse $(\mathbf{J}_n)^{-1}$ is not computed explicitly; we instead rewrite (2.71) as

$$\mathbf{z}_{n+1} = \mathbf{z}_n + \mathbf{x}_n, \tag{2.72}$$

where $\mathbf{x}_n$ is the *Newton update*, which is obtained by solving the linear system

$$-\mathbf{J}_n \mathbf{x}_n = \mathbf{F}_n. \tag{2.73}$$

This is typically much more efficient than matrix inversion and avoids much of the numerical instability introduced by iterative inversion methods like Gauss-Jordan elimination [116].

When the initial guess $\mathbf{z}_0$ is sufficiently close to the solution, Newton's method provides *quadratic convergence*. This means that the residual $\mathbf{F}(\mathbf{z})$ at each iteration is roughly the square root of the residual at the previous iteration. Equivalently, the number of correct decimal digits in the solution roughly doubles each iteration in a neighborhood of the actual solution [6]. The phrase *sufficiently close* is key; Newton's method can converge slowly or even diverge if $\mathbf{z}_0$ is not in a neighborhood of the actual solution. Two techniques can be used to prevent slow convergence. First, $\mathbf{z}_0$ can be chosen based on some approximation of the actual solution to increase the probability of convergence. In the case of semiconductor problems, a good starting point is the charge neutrality conditions in (2.9) and (2.10) [38]. Alternatively, a different nonlinear solver can be used with a high tolerance to give $\mathbf{z}_0$ from an imprecise initial condition. One choice is *Gummel's method*, which decouples the solution of ψ, n and p rather than solving all three variables simultaneously [153]. The resulting solution can be improved by applying a few iterations of Newton's method until convergence.

Occasionally, Newton's method may fail to converge even given an accurate initial condition. This can occur when $\mathbf{J}$ is nearly singular, which may result in devices with multiple stable operating points or when large doping gradients are present. In this case, it is often beneficial to apply an *approximate Newton method* to yield convergence. An approximate Newton method modifies definition of the Newton update $\mathbf{x}_n$ in (2.73) such that

$$-\mathbf{M}_n \mathbf{x}_n = \mathbf{F}_n,$$

where $\mathbf{M}_n$ is related to $\mathbf{J}_n$. A common choice is $\mathbf{M}_n = \mathbf{J}_n / d_n$, where $d_n \in (0, 1]$ is the *damping ratio*. The Newton step in (2.72) can then be written as

$$\mathbf{z}_{n+1} = \mathbf{z}_n - d_n \left(\mathbf{J}_n\right)^{-1} \mathbf{F}_n. \tag{2.74}$$

The iterative process described by (2.74) is known as a *damped Newton method*. At each step, the damped Newton method applies an update to $\mathbf{z}$ in the same direction as the exact Newton method, except the update is scaled by the damping ratio d_n. When $d_n < 1$, the updates are smaller than those performed by the exact Newton method, which can often help convergence by preventing large updates resulting from an ill-conditioned Jacobian.

The choice of damping ratio d_n is critical to ensuring convergence in a reasonable number of iterations. If the values of d_n are too small, the sequence $\mathbf{z}_n$ will converge slowly. If the values of d_n are too large, the Newton process may diverge. We must also have $d_n = 1$ for some point in the damped Newton process, since quadratic convergence is only guaranteed when $d_n = 1$. One algorithm for selecting d_n is as follows:

1. Fix the initial damping ratio $d_1 \in (0, 1]$ according to the problem structure. Smaller initial damping ratios are appropriate when difficult convergence is expected.

2. At each iteration, set $d_{n+1} = \max\left(\mathcal{K} d_n, 1\right)$, where $\mathcal{K} > 1$ is the damping *growth factor*.

3. Apply the damped Newton update given in (2.74).

4. Repeat steps 1 through 3 until convergence.

This approach gives a simple method for choosing d_n, but the parameter $\mathcal{K}$ must be set manually depending on the device being simulated. Typical values might be $\mathcal{K} = 1.1$ through $\mathcal{K} = 1.5$, corresponding to less aggressive and more aggressive increases in d_n. More sophisticated algorithms exist for choosing the damping ratio, and some can provide convergence for any choice of $\mathbf{z}_0$ if certain conditions on $\mathbf{F}$ are met [6, 24]. However, such algorithms still rely on parameters that must be manually chosen, and it is unclear if the necessary conditions on $\mathbf{F}$ hold for the discretized van Roosbroeck system.

Several criteria may be used to declare convergence in a Newton-like process. The most obvious of these is a condition on the norm of the residual:

$$||\mathbf{F}_n|| < \epsilon_1, \tag{2.75}$$

where ϵ_1 is a small value indicating the largest possible residual necessary to accept a candidate solution. The criterion in (2.75) can be a good measure of convergence when all unknown quantities are on roughly the same scale. In semiconductor problems, this is typically not the case, since ψ may be on the order of a few volts, while n and p are typically at least $1 \times 10^3\ \mu\text{m}^{-3}$. As a result, large values of the residuals F_n and F_p can mask smaller values of F_ψ, and convergence may be erroneously declared when portions of ψ are undergoing significant Newton updates. To circumvent this issue, the residuals themselves can be scaled arbitrarily without modifying the solution to the system. Some Newton solvers exploit this to use the residual-norm criterion for convergence by scaling the system such that all elements on the diagonal of $\mathbf{J}$ are equal to 1 [24].

Rather than considering the residual, it can be beneficial to examine how much the solution changes with each Newton step. Intuitively, the Newton process is likely to have converged if no points in the solution change appreciably between steps. Specifically, we can require

$$||\mathbf{z}_n - \mathbf{z}_{n-1}|| < \epsilon_2, \tag{2.76}$$

where ϵ_2 is the largest possible update such that $\mathbf{z}_n$ is accepted as the solution. The criterion in (2.76) has the advantage of only considering changes in the unknown variables themselves, which is more physically connected to the solution than are the residuals. The disadvantage of this approach is that it requires an *a priori* estimate of the maximum allowable Newton update, which can vary widely depending on the problem size, doping concentrations and device geometry.

Another related convergence criterion is similar to (2.76) but does not require a problem-dependent tolerance. This is achieved by requiring

$$\frac{||\mathbf{z}_n - \mathbf{z}_{n-1}||}{||\mathbf{z}_1 - \mathbf{z}_0||} < \epsilon_3, \tag{2.77}$$

where ϵ_3 is the largest possible update relative to the first update such that $\mathbf{z}_n$ is accepted as the solution. This criterion avoids the need for a tolerance set by the scale of the unknowns, but it may still declare convergence erroneously in some cases, particularly when the concentrations n and p vary widely along the device. In this case, proportionally large updates may be made in the areas of the device with lower carrier concentrations, but these updates may be smaller than the smallest updates in areas with high concentration. This can occasionally lead to unphysical solutions where carrier concentrations become negative in regions where their magnitudes are relatively small. This difficulty suggests a relative approach, where the updates in each unknown are measured relative to the size of the unknowns. Such an approach can be difficult when the unknowns are near or equal to zero, in which case large errors may be inaccurately computed in an otherwise acceptable solution.

Due to the difficulty in choosing convergence criteria for Newton-like solvers, there is no standard implementation. Many solvers use a combination of (2.76) and (2.77), where ϵ_2 and ϵ_3 are referred to as the *absolute tolerance* and *relative tolerance*, respectively [45]. When multiple criteria are used, the solver may terminate when any of the criteria are met, or it may require multiple or all criteria to be met to ensure accurate convergence [24]. Extensions to the Newton method presented here include the use of techniques to improve the accuracy of the linear solve used to compute $\mathbf{x}_n$. One technique involves Dirichlet boundary conditions at Ohmic contacts. Instead of directly solving, for example, $\psi[k] - b_\psi[k] = 0$ for some desired boundary value $b_\psi[k]$, we can solve $P(\psi[k] - b_\psi[k]) = 0$, where $P \gg 1$ is a large *penalty parameter*. This admits the same solution $\psi[k] = b_\psi[k]$ but has been found to decrease the condition number of the Jacobian matrices arising in certain discretizations [155]. Another technique involves the use of a preconditioner in the linear solve, which can improve performance when an iterative linear solver is used [107].

2.2.4 Test functions

In semiconductor simulation, it is typically useful to know the voltage and current at each contact in a device. In the Van Roosbroeck system, the voltage is set by the user in the form of Dirichlet boundary conditions on the potential ψ. The current is more complicated to determine. The total current through a contact is given by the flux integral

$$I = \int_{c_i} \mathbf{J} \cdot \hat{\mathbf{n}} \, \mathrm{d}s \,, \tag{2.78}$$

where $c_i \subset \partial\Omega$ is the segment of the device boundary belonging to the ith contact and $\hat{\mathbf{n}}$ is the outward-facing normal to $\partial\Omega$. Direct evaluation of the integral in (2.78) can be difficult, since the contact may contain only a few grid points at which the carrier concentrations n and p are known. This section describes the method of *test functions*, which is used to accurately evaluate boundary fluxes and compute terminal currents. This method was proposed by Markowich in 1985 [94] and by Nanz in 1992 [106], and it has since been applied to finite volume simulations [38, 45].

Let $T_i(\mathbf{x})$ be a smooth *test function* such that

$$\int_{c_i} \mathbf{J} \cdot \hat{\mathbf{n}} \, \mathrm{d}s = \oint_{\partial\Omega} (T_i \mathbf{J}) \cdot \hat{\mathbf{n}} \, \mathrm{d}s \,. \tag{2.79}$$

For (2.79) to hold, we must have $T_i = 1$ along c_i and $T_i = 0$ along the remainder of $\partial\Omega$. For such a T_i, Gauss's theorem then gives

$$I = \oint_{\partial\Omega} (T_i \mathbf{J}) \cdot \hat{\mathbf{n}} \, \mathrm{d}s = \oiint_{\Omega} \boldsymbol{\nabla} \cdot (T_i \mathbf{J}) \, \mathrm{d}\omega \,. \tag{2.80}$$

Applying the product rule to (2.80) gives

$$I = \oiint_{\Omega} (\boldsymbol{\nabla} T_i) \cdot \mathbf{J} \, \mathrm{d}\omega + \oiint_{\Omega} T_i \, (\boldsymbol{\nabla} \cdot \mathbf{J}) \, \mathrm{d}\omega \,. \tag{2.81}$$

From the continuity equations in (2.1),

$$\boldsymbol{\nabla} \cdot \mathbf{J_n} = -q\,(G - U) \,, \quad \boldsymbol{\nabla} \cdot \mathbf{J_p} = q\,(G - U) \,.$$

Thus, since $\mathbf{J} = \mathbf{J_n} + \mathbf{J_p}$,

$$\boldsymbol{\nabla} \cdot \mathbf{J} = \boldsymbol{\nabla} \cdot \left(\mathbf{J_n} + \mathbf{J_p}\right) = \boldsymbol{\nabla} \cdot \mathbf{J_n} + \boldsymbol{\nabla} \cdot \mathbf{J_p} = 0,$$

verifying the conservation of charge. The second term in (2.81) vanishes, leaving

$$I = \oiint_{\Omega} (\nabla T_i) \cdot \left(\mathbf{J_n} + \mathbf{J_p}\right) \mathrm{d}\omega\,. \tag{2.82}$$

In a finite volume discretization, the integral in (2.82) can be approximated as [38, 45]

$$I = q \sum_{k=1}^{N} \sum_{\substack{l \in \mathcal{N}[k] \\ l > k}} \frac{|\partial\omega_k \cap \partial\omega_l|}{\|\mathbf{x}[l] - \mathbf{x}[k]\|} \left(g_n[k,l] - g_p[k,l]\right)(T_i[k] - T_i[l])\,, \tag{2.83}$$

where g_n and g_p are the electron and hole fluxes in (2.64) and (2.65) and N is the number of nodes in the discretization. This approximation is possible since the electron and hole flux discretizations can also be viewed as projections of the current densities onto the midpoint of the edge bounded by $\mathbf{x}[k]$ and $\mathbf{x}[l]$. The factor of q is necessary since the flux discretizations only account for the flux of charge *carriers*, while the current densities refer to the flux of *charge*.

A few methods exist for determining the test function $T_i(\mathbf{x})$. The test function must be smooth in order for Gauss's theorem to hold, and it must assume the appropriate boundary values along $\partial\Omega$. Determining such a function generally requires solving a separate PDE along the device. The simplest choice is the *Laplace equation*

$$\nabla^2 T_i = 0. \tag{2.84}$$

In 1D, (2.84) simplifies to $T_i''(x) = 0$, which implies a linear test function along the device. In this case, the test function can be determined analytically. Let x_1 and x_2 be the leftmost and rightmost extents of the device, and let c_1 and c_2 be the left and right contacts. The test function giving the current through c_1 has the boundary conditions $T_1(x_1) = 1$ and $T_1(x_2) = 0$, which implies the system

$$\begin{cases} ax_1 + b = 1 \\ ax_2 + b = 0 \end{cases}$$

for some a, b. Solving the system gives

$$a = -\frac{1}{x_2 - x_1}, \quad b = \frac{x_2}{x_2 - x_1}.$$

The test function is thus

$$T_1(x) = ax + b = \frac{x_2 - x}{x_2 - x_1}. \tag{2.85}$$

Similarly, the test function giving the current through c_2 is

$$T_2(x) = 1 - T_1(x) = \frac{x - x_1}{x_2 - x_1}. \tag{2.86}$$

The expressions in (2.85) and (2.86) can be used for any 1D device, since such a device can have only two contacts.

As an instructive example of the test function method, we can derive the expression for terminal current in a 1D semiconductor with constant mobilities using the typical Laplace test function. Since each node in this case has at most two neighbors, we can rewrite the sum in (2.83) as

$$I = \pm q \sum_{k=1}^{N-1} \frac{1}{x[k+1] - x[k]} \left(g_n[k,k+1] - g_p[k,k+1]\right)(T_1[k] - T_1[k+1])\,, \tag{2.87}$$

where the sign of the current is determined by the choice of contact, since all the current flowing into c_1 must flow out of c_2. The test function flux in (2.87) can be rewritten using (2.85) as

$$T_1[k] - T_1[k+1] = \frac{x_2 - x[k]}{x_2 - x_1} - \frac{x_2 - x[k+1]}{x_2 - x_1} = \frac{x_[k+1] - x[k]}{x_2 - x_1}. \tag{2.88}$$

Substituting (2.88) into (2.87),

$$I = \pm \frac{q}{x_2 - x_1} \sum_{k=1}^{N-1} \Big(g_n[k, k+1] - g_p[k, k+1] \Big).$$

Substituting the carrier fluxes from (2.64) and (2.65),

$$I = \pm \frac{qV_T}{x_2 - x_1} \sum_{k=1}^{N-1} \mu_n \left(B(-\delta_r) n[k] - B(\delta_r) n[k+1] \right) - \mu_p \left(B(\delta_r) p[k] - B(-\delta_r) p[k+1] \right). \tag{2.89}$$

The expression in (2.89) is an average of the projections of the current onto the midpoints of grid segments. This illustrates how the test function method can accurately estimate terminal currents: even in difficult situations where computing the current from one or two projections would be inaccurate, the test function considers all nodes in the discretization, which has a mitigating effect on local numerical errors.

In higher dimensions, the PDE in (2.84) is easy to solve since the resulting finite volume discretization is a linear system of equations. Solving the Laplace equation requires only one linear solve, as opposed to more complex equations which require one linear solve for each Newton iteration. In some simulations, however, the Laplace test function has been shown to yield inaccurate estimates of the current, particularly in devices where the doping profile varies abruptly over several orders of magnitude [106]. In such situations, it can be beneficial to define a test function such that "In regions with large carrier concentrations the gradient of the weight function should be small" [106]. Nanz proposed the function

$$\nabla \cdot \left(\frac{n}{n_i} \nabla T_{n,i} \right) = 0 \tag{2.90}$$

for the electron current and

$$\nabla \cdot \left(\frac{p}{n_i} \nabla T_{p,i} \right) = 0 \tag{2.91}$$

for the hole current, with the Dirichlet boundary conditions $T_{n,i}(\mathbf{x}) = T_{p,i}(\mathbf{x}) = 1.1$ for $x \in c_i$ and $T_{n,i}(\mathbf{x}) = T_{p,i}(\mathbf{x}) = -0.1$ at all other contacts. The solutions to the PDEs (2.90) and (2.91) were then clamped to the range $[0, 1]$ and smoothed with the polynomial

$$s(x) = -2x^3 + 3x^2. \tag{2.92}$$

Due to the modified boundary conditions, the smoothed test functions have a "plateau" near each contact, which was shown to circumvent numerical issues typically associated with the Laplace test function near the contact edges. This effect is shown in Figure 2.8, which shows the test functions generated using (2.90) and (2.91) for a 1D diode under forward and reverse bias. The reverse-biased diode has a large depletion region near $x = 0$, which causes large gradients in n and p and leads to a more pronounced plateau effect. The forward-biased diode has a very small depletion region, so the test functions appear smoother near $x = 0$.

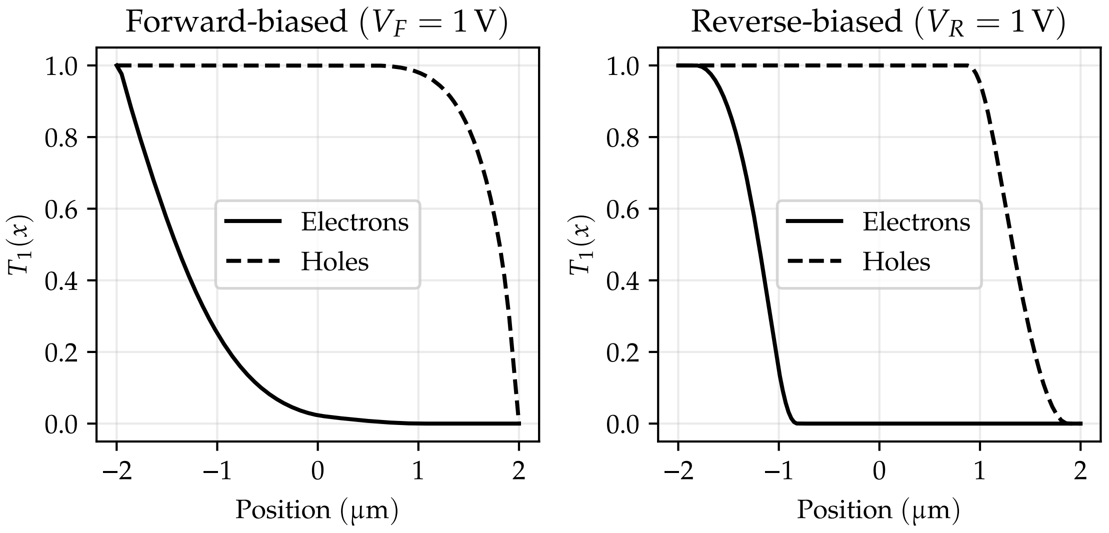


Figure 2.8: Concentration-weighted test functions for 1D diode under forward and reverse bias.

The *concentration-weighted* test functions described by (2.90) and (2.91), while occasionally more accurate than the Laplace test function, are significantly more expensive to evaluate. First, they require two separate test functions—one for electrons and one for holes—for each grid. Second, due to their dependence on carrier concentrations, they must be re-evaluated for every operating point of the device, unlike the Laplace test function which must only be evaluated once per grid and can be reused for any operating point.

In addition to the test function method, other methods have been proposed to evaluate terminal currents faster, more accurately, or both. Since the current field is conservative, Palm and Van de Wiele noted that the current density can be derived from a *current potential*, which can be used to evaluate terminal currents and the current field itself along the device [110]. Gusmeroli and Spinelli proposed the *residue method*, which does not require the solution of a PDE to evaluate the current but relies on the use of the continuous Galerkin discretization. The residue method was additionally noted to have lower accuracy than the test function method at low currents [57]. The residue method was extended to 3D simulations including impact ionization in [96]. The test function was applied to Monte Carlo simulations using the Boltzmann transport equation in [167].

2.2.5 Current boundary conditions

In device simulation, it is often useful to set terminal currents explicitly rather than determining them during postprocessing. In this case, we can add one equation to the typical discretized system to enforce the current [33, 55]. The method of test functions can be used to formulate the extra equation as long as the test function is not solution-dependent like the concentration-weighted test functions described by (2.90) and (2.91). The extra equation is then

$$F_I(\mathbf{z}) = I - q \sum_{k=1}^{N} \sum_{\substack{l \in \mathcal{N}[k] \\ l > k}} \frac{|\partial\omega_k \cap \partial\omega_l|}{\|\mathbf{x}[l] - \mathbf{x}[k]\|} \left(g_n[k,l] - g_p[k,l]\right)(T_i[k] - T_i[l]) = 0, \tag{2.93}$$

where $\mathbf{z}$ is the vector of unknowns and I is the desired current value. We can solve the augmented system using a damped Newton process similar to (2.74). The Newton update at each iteration is given by solving

$$\begin{bmatrix} \mathbf{J}_n & \frac{\partial \mathbf{F}_n}{\partial V} \\ \frac{\partial F_{I,n}}{\partial \mathbf{z}_n} & 0 \end{bmatrix} \mathbf{x}_n = - \begin{bmatrix} \mathbf{F}_n \\ F_{I,n} \end{bmatrix}, \tag{2.94}$$

where $x \in \mathbb{R}^{N+1}$, the unknown vector $\mathbf{z}$ is defined in (2.69), the residual $\mathbf{F}$ is defined in (2.70) and $\mathbf{J}_n = \partial \mathbf{F}_n / \partial \mathbf{z}_n$. Then, the updated solution and bias voltage are given by

$$\begin{bmatrix} \mathbf{z}_{n+1} \\ V_{n+1} \end{bmatrix} = \begin{bmatrix} \mathbf{z}_n \\ V_n \end{bmatrix} + \mathbf{x}_n. \tag{2.95}$$

The gradient $\partial F_I / \partial \mathbf{Z}$ in (2.94) can be determined explicitly or via automatic differentiation. The entries of the derivative $\partial \mathbf{F} / \partial V$ are 1 at indices corresponding to the nodes in the current-controlled contact and are 0 at all other indices, assuming Dirichlet boundary conditions are enforced without penalty. The process in (2.94) and (2.95) can be generalized to an arbitrary number of current-controlled contacts by allowing the residual F_I to be vector-valued. Each additional current-controlled contact adds a row and column to the block Jacobian in (2.95).

As noted in [105], this method of imposing a fixed current through a set of contacts can lead to convergence problems. Computational experience shows that many current-controlled simulations diverge unless aggressive damping is used in the Newton process to keep bias voltage variation small between iterations. While this method allows either current or voltage to be set at a contact, it can be extended to enforce arbitrary combinations of the two, allowing the inclusion of simple circuits connected to the device terminals [73].

2.2.6 I-V curve tracing

The *current-voltage characteristic*, sometimes called an I-V curve, is a key indicator of the performance and character of many semiconductor devices. It is therefore of interest in device simulation to accurately generate I-V curves so that devices may be analyzed and comparisons may be made among them. To generate an I-V curve, the voltage or current at one contact is varied while the voltage and current at all other contacts is held constant. This requires many simulations; one simulation must be performed for each set of bias conditions along the I-V curve. Since the Newton method is highly sensitive to initial conditions, it is critical that initial conditions be chosen appropriately for each point on an I-V curve.

A general requirement for convergence using Newton-like nonlinear solvers is that the initial guess be sufficiently close to the solution. Consider a simple device with two contacts, for which it is desired to simulate the bias voltages $(V_1, V_2, \cdots, V_L)$, at which points the unknown currents $(I_1, I_2, \cdots, I_L)$ will be recorded. At each point on the curve, the vector-valued unknowns in the discretization are $(\mathbf{z}_1, \mathbf{z}_2, \cdots, \mathbf{z}_L)$, where $\mathbf{z}$ is defined in (2.69). The naïve algorithm for tracing this I-V curve is

1. Begin with an initial guess $\mathbf{z}_0$.
2. Solve $\mathbf{F}(\mathbf{z}_n) = \mathbf{0}$, using $\mathbf{z} = \mathbf{z}_{n-1}$ as an initial guess for the nonlinear solver.
3. Compute the current I_n using the solution $\mathbf{z}_n$.
4. Repeat steps 2 and 3 for $n \in [1, L]$.

A problem arises when determining the initial guess $\mathbf{z}_0$. Without any *a priori* knowledge of the character of $\mathbf{z}_1$, it is generally difficult to provide an initial guess such that the first nonlinear solve converges. One solution is to choose $V_1 = 0$, such that the device is in thermal equilibrium at the first point on the I-V curve. Then, the Van Roosbroeck system reduces to the equilibrium case of a single PDE, which is given in (2.8). This PDE can be solved for ψ, and (2.7) can be used to compute n and p, thus giving $\mathbf{z}_0$. Computational experience shows that this uncoupled PDE is less sensitive to initial conditions than the non-equilibrium Van Roosbroeck system, but some initial guess is still needed for the thermal equilibrium problem. In some cases, we can guess $\mathbf{z} = \mathbf{1}_N$ or $\mathbf{z} = \mathbf{0}_N$ and get convergence when large doping gradients are not involved. Alternatively, we can estimate ψ from the charge neutrality condition in (2.9), which gives convergence for a wide range of geometries and doping profiles [38].

For some devices, the naïve curve tracing scheme described above performs poorly or may fail altogether. To understand why, we must analyze the character of solutions to the semiconductor problem. Several theoretical results have been proven regarding the *existence* and *uniqueness* of solutions to Van Roosbroeck system. Markowich showed in 1985 that, when solutions to the Van Roosbroeck system exist, the I-V characteristic varies continuously with applied voltage. This is due in part to the implicit value theorem [94]. In other words, the I-V characteristic is a smooth curve in the current-voltage plane. Existence of a solution is guaranteed in the thermal equilibrium case when impact ionization is negligible. Few results exist for non-negligible impact ionization rates, except in very specific situations. In particular, existence has been proven for all bias conditions in a 1D diode with a symmetric, piecewise constant doping profile with constant mobilities [92].

Uniqueness is only guaranteed under specific conditions. For instance, it has been shown that a unique solution exists in the thermal equilibrium case where $V = 0$ for all contacts, when impact ionization is negligible [153]. Moreover, a unique solution exists even if V is nonzero for some contacts, so long as the vector of bias voltages

$$\mathbf{V} = \begin{bmatrix} V_1 & V_2 & \cdots & V_M \end{bmatrix}^T,$$

where M is the number of contacts, is within a sphere of sufficiently small radius. The maximum radius for which uniqueness holds is problem-dependent and can vary with the doping profile, the mobilities μ_n and μ_p and any parameters of the geometry and discretization [94]. In general, however, uniqueness does not hold, and the Van Roosbroeck system can admit one, several or no solutions. The most common example of a device admitting multiple solutions is a diode in avalanche breakdown, which can exhibit *snap-back* behavior near the breakdown voltage, leading to an S-shaped I-V curve [33, 95]. Similar behavior has been observed in the simulation of thyristors [153], bipolar transistors [7, 51] and CMOS devices [23].

The simulation of multivalued behavior of I-V curves poses considerable numerical difficulty for two main reasons. First, it is typically necessary to include impact ionization in simulation to produce a multivalued I-V curve. The dependence of the ionization rate G on the vector quantities $\mathbf{J_n}$, $\mathbf{J_p}$ and $\mathbf{E}$ is difficult to model in a finite-volume discretization, and several schemes have been proposed to incorporate this phenomenon [80, 81, 151, 152]. Beyond the discretization, impact ionization causes convergence problems for most nonlinear solvers. Kumashiro found that a major source of this difficulty is a "positive feedback" effect between the ionization rate and the current densities: An increase in G in one iteration of the nonlinear solve causes $\|\mathbf{J_n}\|$ and $\left\|\mathbf{J_p}\right\|$ to increase, which causes G to increase in the next iteration, and so on [80]. Second, the Jacobian of the discretized PDE system is singular at *turning points* where the I-V curve becomes multivalued [74, 153]. Thus, while it is often possible to approach turning points in simulation, it can be difficult or impossible to trace the I-V curve beyond the turning point without encountering convergence issues.

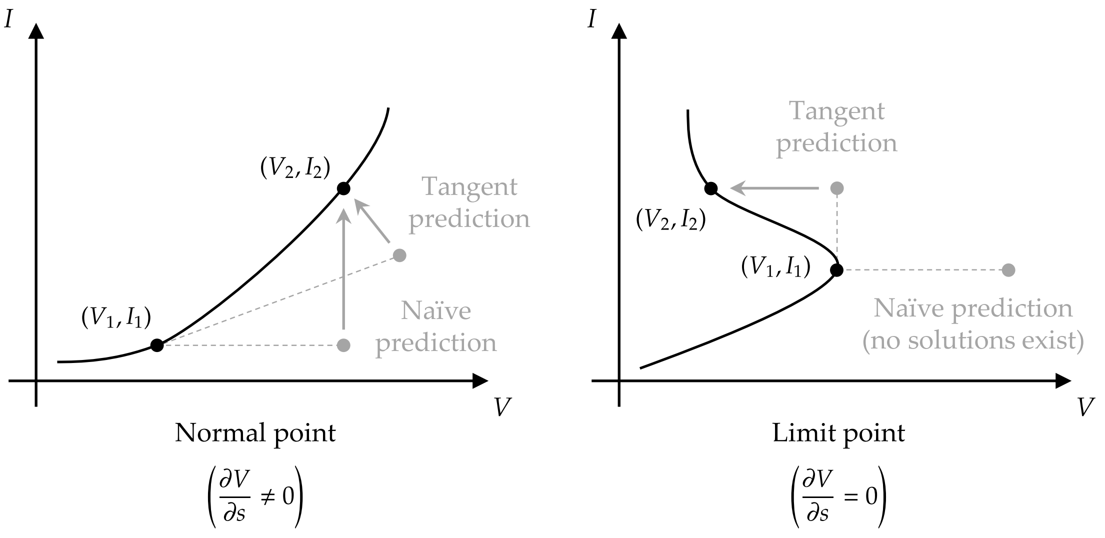
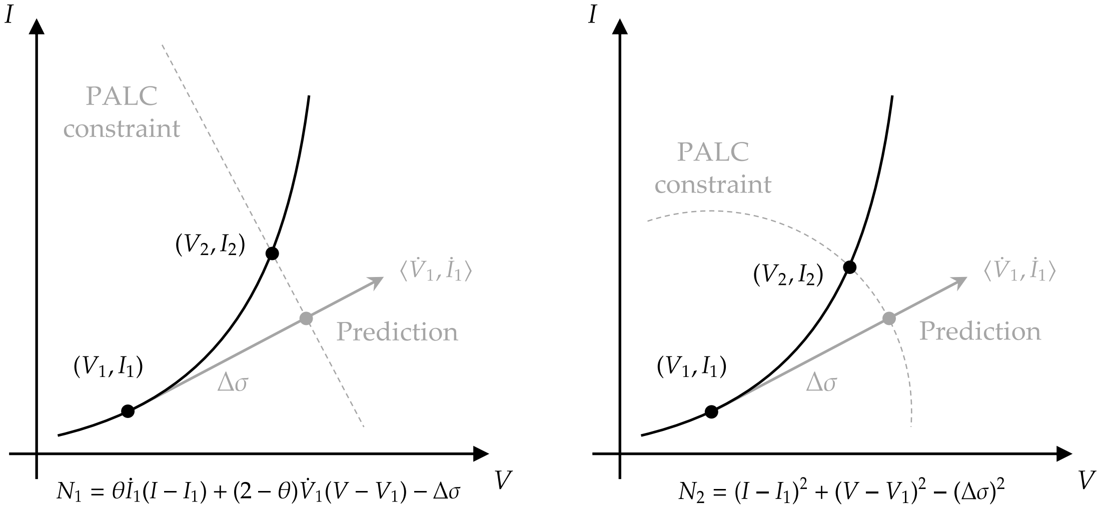

I
(V_2, I_2)
Tangent prediction
(V_1, I_1)
Naïve prediction
V
Normal point
$\left(\frac{\partial V}{\partial s} \neq 0\right)$
I
Tangent prediction
(V_2, I_2)
(V_1, I_1)
Naïve prediction (no solutions exist)
V
Limit point
$\left(\frac{\partial V}{\partial s} = 0\right)$

Figure 2.9: Illustration of naïve prediction and tangent prediction algorithms at a normal point (left) and at a limit point (right). Figure adapted from [95].

Several techniques have been developed to overcome the difficulty of tracing multivalued I-V curves. The naïve algorithm above is part of a more general class of algorithms called *predictor-corrector* algorithms. Such algorithms can be used to trace I-V curves by using a *predictor* to guess the next point on the curve. Since the guess will not generally be a solution of the PDE system, a *corrector* is then used to find a point on the curve of solutions near the initial guess. When tracing I-V curves, the corrector is a nonlinear solver, i.e. the Newton method. The predictor algorithm can be one of several choices. One of the simplest and most widely-used predictors is the *tangent predictor*, which will be discussed below.

To understand the tangent predictor, it is useful to first consider the naïve approach described above as a simple predictor algorithm [74]. Consider the situation where it is desired to trace an I-V curve from a point (V_1, I_1) to (V_2, I_2). This approach makes the prediction $\mathbf{z}_2 = \mathbf{z}_1$ and fixes the bias voltage at some value V_2, which is typically set such that adjacent points on the I-V curve are separated by a fixed distance, i.e. $V_2 = V_1 + \Delta V$. A Newton-like solver is then applied to correct the predicted solution, yielding the solution $\mathbf{z}_2$. This solution is consistent with the bias voltage $V = V_2$, since that bias voltage was set externally via Dirichlet boundary conditions. This situation is illustrated in Figure 2.9. On the left, the I-V curve is single-valued, and a solution exists for the predicted voltage $V = V_1 + \Delta V$. The Newton solver converges to the desired solution, and the step is successful. On the right, however, the I-V curve is multivalued for a range of bias voltages. In this case, the naïve predictor attempts to set $V = V_1 + \Delta V$, but such a solution does not exist due to the snapback behavior. The Newton solver diverges, and the step is unsuccessful.

The tangent predictor is able to overcome such *limit points* where the I-V curve becomes multivalued. Consider the parametric I-V curve $(V(s), I(s))$, where I is computed from a high-dimensional curve $\mathbf{z}(s)$ in the solution space $\mathbb{R}^N$. Instead of fixing the bias voltage, the tangent predictor treats the bias voltage as an unknown variable in the discretized PDE system. The resulting augmented system of equations has $N + 1$ unknowns, similar to the system used to set current boundary conditions described in Section 2.2.5. Consider again the situation where it is



I
PALC constraint
(V_2, I_2)
$\langle \dot{V}_1, \dot{I}_1 \rangle$
Prediction
(V_1, I_1)
$\Delta\sigma$
V
$N_1 = \theta \dot{I}_1(I - I_1) + (2 - \theta)\dot{V}_1(V - V_1) - \Delta\sigma$
I
PALC constraint
(V_2, I_2)
$\langle \dot{V}_1, \dot{I}_1 \rangle$
Prediction
(V_1, I_1)
$\Delta\sigma$
V
$N_2 = (I - I_1)^2 + (V - V_1)^2 - (\Delta\sigma)^2$

Figure 2.10: Illustration of the two PALC constraints N_1 and N_2 and their effect on the correction step of the continuation algorithm.

desired to trace an I-V curve from a point (V_1, I_1) to (V_2, I_2). This approach makes the prediction

$$\begin{bmatrix} \mathbf{z}_2 \\ V_2 \end{bmatrix} = \begin{bmatrix} \mathbf{z}_1 \\ V_1 \end{bmatrix} + \Delta\sigma \begin{bmatrix} \dot{\mathbf{z}}_1 \\ \dot{V}_1 \end{bmatrix},$$

where $\dot{\mathbf{z}} = \partial \mathbf{z}/\partial s$, $\dot{V} = \partial V/\partial s$, and $\Delta\sigma$ is the desired length of the curve segment from (V_1, I_1) to (V_2, I_2) [24]. Since the bias voltage is now unknown, it is acceptable for the prediction to lie outside of the curve, even if no solutions exist with the given bias voltage. The second step of the curve-tracing algorithm, the corrector, attempts to find a solution point near the prediction, possibly with a different bias voltage. To correct the prediction and find a point on the I-V curve that lies a distance of approximately $\Delta\sigma$ away from (V_1, I_1), we need a constraint to add to the augmented system of equations.

There are a few constraints that can be added to enforce the step size requirement. One choice is to force the tangent vector to have unit length, i.e. to require that

$$\dot{V}(s)^2 + \dot{I}(s)^2 = 1. \tag{2.96}$$

The use of the constraint in (2.97) along with the tangent predictor algorithm is known as *arc-length continuation*. Typically, this constraint is not enforced exactly due to its nonlinearity. Instead, *pseudo arc-length continuation* (PALC) is often used as a substitute, where (2.97) is enforced approximately [74]. Two PALC constraints were introduced in [24]:

$$N_1 = \theta \dot{I}_n(I - I_n) + (2 - \theta)\dot{V}_n(V - V_n) - \Delta\sigma = 0, \tag{2.97}$$

and

$$N_2 = (I - I_n)^2 + (V - V_n)^2 - (\Delta\sigma)^2 = 0. \tag{2.98}$$

The behavior of these two constraints is illustrated in Figure 2.10. The constraint N_1 defines a straight line that intersects with the tangent vector a distance $\Delta\sigma$ away from the point (V_1, I_1). The slope of the line is determined by the parameter θ, which ranges from 0 to 2. When $\theta = 1$, the constraint line is perpendicular to the tangent vector [154]. The constraint N_2 defines a circle centered at (V_1, I_1) with radius $\Delta\sigma$. This constraint will generally intersect with the I-V curve at two points, but the Newton solver will likely converge to the solution nearest the prediction [24].

The constraints in (2.97) and (2.98) depend on appropriate scaling for the voltage and current. For N_1, a perpendicular line to the tangent vector is only desirable if V and I are on similar scales. Typically this is not the case, and V can be on the order of a few volts, while I may be on the order of microamps or less, depending on the device. In this case, the parameter θ should be very close to 2, i.e. $\theta = 2 - 1 \times 10^{-6}$ or so. Setting $\theta = 2$ amounts to setting a current boundary condition, where a fixed current step is taken at each point on the curve. Conversely, setting $\theta = 0$ amounts to setting a voltage boundary condition at the active contact. For N_2, this scaling problem suggests an alternate definition such as

$$N_2 = \theta^2(I - I_n)^2 + (2 - \theta)^2(V - V_n)^2 - (\Delta\sigma)^2,$$

which defines an ellipse centered at (V_1, I_1). The parameter θ has a similar effect. Alternatively, if modification to the constraints is not desired, the voltages or currents can be scaled after simulation, before beginning the continuation algorithm.

As evidenced by the complexity of these continuation algorithms, tracing multivalued I-V curves is highly nontrivial. The study of continuation methods led to the development of *bifurcation analysis*, a field of mathematics concerned with related differential equations that exhibit branching or admit multiple solutions. Similar continuation methods to the ones described here have been compiled in software packages aimed at general elliptic PDE systems [4, 154]. Arc-length continuation has also proven useful in modeling substrate currents and latchup phenomena in CMOS circuits [23]. More generally, continuation has been applied to circuit-level simulators where multivalued transfer characteristics may be present [22]. Continuation can also be useful to iteratively find DC operating points in large circuits when stepping directly to the desired bias conditions causes convergence problems [131]. When solution branches exist that cannot be reached with conventional continuation methods, *deflated continuation* can be used to find all possible solutions to a system at a given operating point [40]. This can be useful to avoid the need to trace near turning points where the system Jacobian becomes singular.

Several methods have also been proposed to trace I-V curves without requiring complex continuation algorithms. One such algorithm was used to trace snapback behavior in bipolar transistors and does not require adding equations to the PDE discretization. Instead of treating the bias voltages as unknowns, this approach tries to guess points on either side of the I-V curve and linearly interpolates between them to estimate the shape of the curve [7]. Another approach uses *automatic biasing*, where a combination of a voltage source, a current source, and a resistor are used to load the device such that only one operating point is possible for each load combination [51]. This scheme requires one or more equations to be added to the discretization, making its use in simulation difficult. Unlike the other methods, however, it could easily be applied in a physical curve tracer machine to give accurate measurements of breakdown and snapback parameters.

2.3 Computational methods

2.3.1 Automatic differentiation

Evaluating derivatives of computer programs is essential to many tasks in computational physics and machine learning. Nonlinear solvers such as Newton's method, for example, require the Jacobian of a vector-valued function that may be prohibitively difficult to compute by hand. Many optimization methods require the gradient of a loss function which may depend on thousands or millions of model parameters. Historically, numerical derivatives were computed using finite differencing. While this approach was suitable for simple problems, the resulting truncation error often led iterative solvers to far worse performance than if the derivatives were computed by hand [145]. *Symbolic differentiation* can achieve lower error than finite differencing by manipulating equations algebraically. This approach, however, is seldom appropriate for scientific computing where thousands of derivatives are computed in rapid succession. Another approach is *automatic differentiation* (AD), which has similarities to both symbolic differentiation and numerical finite differencing. It does not, however, incur truncation error, and in many cases provides comparable accuracy to symbolic differentiation at significantly lower computational expense [54].

Automatic differentiation methods compute derivatives of computer programs by exploiting the structure of the programs, and in many cases, of the programming language itself. The two main AD methods are *forward-mode* and *reverse-mode* AD. Forward-mode AD operates in an intuitive manner, treating each mathematical operation in a program as a separate step whose derivative can be evaluated. The complete list of operations in a program and the dependencies between different operations is known as a *Wengert list* [165]. Forward-mode AD works by considering the value and derivative of each expression in a Wengert list, which are often called *primal* and *dual* numbers, respectively. The derivatives of all the expressions in the Wengert list are composed by applying the chain rule. Forward-mode AD has been implemented in several software packages, most notably in the Julia package `ForwardDiff.jl` [129]. It is typically implemented using operator overloading or by defining a special *dual type*, for which typical arithmetic operations are defined, that holds both a primal value and its dual.

Rather than computing the derivative of each operation with respect to its inputs, reverse-mode AD relies on the *sensitivity* of an input variable to each of the outputs of a function [54]. Reverse-mode AD also relies on the use of a Wengert list and composes intermediate results using the chain rule. It is typically implemented using *adjoint methods* to compute the sensitivities [70] or using *source-to-source transformation*, where a second program is generated from the code of the first that computes the first program's derivative [145]. Notable implementations of this method include the Tapenade AD tool [60], which works with Fortran or C code, and `Zygote.jl`, which works with Julia code [68].

Compared to forward-mode AD, reverse-mode AD is more efficient at computing gradients of functions with fewer outputs than inputs. Such instances include many machine learning models, when it is desired to compute the gradient of a scalar-valued loss function. Forward-mode AD is more efficient at computing derivatives of vector-valued functions with more outputs than inputs [54]. Both forward-mode and reverse-mode AD require a library of matrix derivative results to compute derivatives and sensitivities of intermediate quantities. A collection of such results is given in [49].

2.3.2 Scientific machine learning

Nearly every physical phenomena can be described by a differential equation or by a system of differential equations. As illustrated in Section 2.1, a simple system of 3 PDEs can give an accurate model of most semiconductor devices in 1D, 2D or 3D. Despite the simplicity of the Van Roosbroeck system, the number of possible models for a given system grows exponentially as increasingly many physical effects are considered. Impact ionization and field-dependent mobilities are the most common extensions to the drift-diffusion model; many others exist when simulation of nanoscale devices is desired [90]. Eventually, physical intuition must be replaced with experimental data, as many parameters like ionization coefficients rely on empirical curve-fitting to measured devices. This effect is pervasive in other fields of science and engineering. As increasingly complex models become necessary, simple physics-based models must be augmented by observational data. *Scientific machine learning* is a broad name given to a class of methods that seeks to bridge the divide between physical models and data-driven optimization. This section will briefly describe some of the common methods in scientific machine learning and the computational tools that have been developed to implement them.

One of the earliest-developed techniques related to scientific machine learning is the *neural ordinary differential equation* (neural ODE) [17]. The neural ODE relies on the ability of neural networks (NNs) to act as universal approximators which can learn and accurately model the behavior of a wide class of functions. This ability is enhanced by the use of nonlinear activation functions like the Gaussian Error Linear Unit (GELU) [62]. NNs have classically been used for *supervised* and *unsupervised* tasks, where they have been highly effective at detecting patterns in structured and unstructured data, respectively. *Recurrent neural networks* can additionally be used to construct discrete models of time-series data. It is often of interest to model continuous-time data, in which a quantity does not change value at a well-defined frequency. Neural ODEs fill this need by using a NN to parameterize the forcing function of an ODE. The resulting equation can be solved on any timescale and can be trained on a wide variety of data. Neural ODEs have been extended to stiff systems [78] and to problems requiring Bayesian inference [27]. Recently, the *Fourier neural operator* was introduced to allow neural ODEs to learn an entire family of problems, rather than a single equation [86].

The *universal differential equation* has been introduced as an extension to the philosophy underlying neural ODEs [126]. Neural ODEs use NNs to parameterize an entire differential equation and are thus a purely data-driven approach. Many models call for a mix of physical and data-based modeling, particularly when the model is used in a mission-critical application (as in much of semiconductor simulation, where TCAD models are often used to make design decisions). Universal differential equations fill this need by allowing for a differential equation model to be partially parameterized by a NN while retaining some description of the underlying physics [125]. This approach extends on earlier physics related approaches like Physics-Informed Neural Networks (PINNs) [127] by leveraging AD capabilities to allow data-driven model creation without requiring domain specific code. Universal differential equations have been applied in a wide range of domains, from predicting the COVID-19 pandemic [26] to discovering missing physics in climate models [128]. Beyond universal differential equations, other scientific machine learning techniques such as continuous time echo state networks (CTESNs) have been shown to yield performant surrogates of systems with dynamics on multiple timescales [1].

2.4 Related work

Much of the groundwork for efficient surrogate modeling was laid by *inverse design* approaches in disparate fields of engineering. Historically, engineering design has been accomplished using sets of best practices and "rules of thumb," and designers have often relied on intuition and experience to meet a set of specifications. Inverse design, by contrast, begins with a set of specifications and uses mathematical optimization to generate a design without prior knowledge of the character of such a solution [21]. Much of the early applications of inverse design were directed toward photonics [103], particularly in problems such as nanolenses and mode converters [2], photovoltaic materials [169], wavelength demultiplexers [118] and grating couplers [134]. Recently, *optical metasurfaces* have been constructed using inverse design, and some approaches have leveraged automatic differentiation for faster simulation and optimization [21, 114].

The inverse design of metasurfaces prompted the development of a new surrogate approach: physics-enhanced deep surrogates (PEDSs) [115]. The PEDS approach considers the simulation of a metasurface using both a fine-grained geometry and a coarse-grained geometry. The fine-grained geometry provides a more realistic simulation of the metasurface at the expense of higher computational cost. The coarse-grained geometry is faster to simulate but introduced unacceptable error compared to the fine-grained simulation. To develop a surrogate model, the PEDS approach creates a *residual geometry* that is added to the downsampled geometry used in the coarse-grained simulation. The residual geometry is generated by a neural network that can be trained to minimize the error between the output of the coarse-grained simulation and the output of the fine-grained simulation. Additionally, the neural network can be trained to generate residuals for a wide range of input parameters, providing an efficient way to quickly create a surrogate without repeating the lengthy training process.

A number of inverse design and surrogate methods have been applied in semiconductor-related problems. Geometric programming has been used to optimize doping profiles in bipolar transistors using an analytical model of the base transit time [71]. We note that this optimization did not entail a physics-based simulation of the entire device and relied on minimization of a single parameter. Geometric programming has also allowed inverse design of CMOS op-amps [63, 91, 157], multistage RF amplifiers [28], planar spiral inductors [102] and LC oscillators [64]. Traditional surrogate methods including Latin hypercube sampling have been applied to model the power consumption and bandwidth of a bipolar op-amps [168] and the drain current of 65 nm NMOS and PMOS devices [166]. Various adaptive sampling techniques have also been used to train surrogates of SPICE MOSFET models [143]. Forward-mode AD was used to perform sensitivity analysis on systems derived from Maxwell's equations in [66].

More recent works have focused on applying powerful machine learning methods to inverse design problems. A neural network architecture was proposed in [100] to avoid overfitting in TCAD-augmented machine learning. The architecture used an autoencoder to estimate the thicknesses of p-doped, intrinsic and n-doped regions in a p-i-n diode. This method also allowed inverse design of the layer thicknesses to match simulation outputs to a hand-drawn I-V curve. Similar autoencoder-based models were used to recover the metal workfunction of a Schottky diode from its I-V curve in [30] and to predict FinFET I-V curves from device dimensions [99]. Differential evolution techniques were used to design semiconducting boron sheets for 2D FETs in [171]. A multilayer perceptron (MLP) model was trained to predict drain current, effective mobility and electron density in a 3D FinFET in [76]. The same model was used in inverse design; the width, thickness and backgate voltages of a device were designed from a desired subthreshold swing, threshold voltage and mobility degradation. Deep neural networks have also been applied in inverse design of tunnel field-effect transistors (TFETs) [164] and FinFET SRAM cells [172].

Considerable theoretical and experimental attention has been devoted to the optimization and inverse design of doping profiles. Early theoretical work showed that doping profile recovery was possible from stationary simulations in the limited *unipolar* case where $p = 0$ everywhere, given an arbitrarily large set of I-V or C-V curve data [13]. This work supported earlier experimental results that used black-box optimization to extract doping profiles from CMOS devices [83, 122]. It was later shown that the quality of the doping reconstruction depended on the amount of data available, and it was conjectured that some doping profiles may be uniquely identifiable from I-V data [84]. A stronger statement was proven in [12], which showed that the *location* of a P-N junction could be uniquely identified using I-V and C-V data if the number of junctions is less than or equal to two. Later experimental work showed that doping profile recovery was possible in 2D and was robust to measurement noise [18]. Similar doping reconstructions were achieved in a 1D finite-difference discretization of the Van Roosbroeck system using optimal control [147].

Chapter 3

Implementation

3.1 Computing environment

3.1.1 Packages used

This work was implemented using the Julia programming language [10], version 1.7.1. Much of the framework for finite volume discretizations was provided or influenced by `VoronoiFVM.jl` [45]. Finite volume grids were implemented using `ExtendableGrids.jl` and visualized using `GridVisualize.jl` and `PyPlot.jl`. The `Triangulate.jl` package was used to generate boundary-conforming Delaunay triangulations in all simulation models [137]. The `SimplexGridFactory.jl` package was used to construct device geometries. Automatic differentiation was primarily performed using `Zygote.jl` [68], and `ForwardDiff.jl` was used when nested AD was required [129]. Sparse matrix functionality was provided by `SparseArrays.jl`. Automatic sparsity detection was implemented using `Symbolics.jl` [52]. Our implementation of arc-length continuation was influenced by `BifurcationKit.jl` [163]. Deflated continuation was also performed using `BifurcationKit.jl`. Surrogate models were trained using `DiffEqFlux.jl` [126] and `GalacticOptim.jl`.

3.1.2 Semiconductor simulation package

This section describes `Semiconductors.jl`, the simulation package created as part of this work. The package acts as an interface to `VoronoiFVM.jl` by defining key physics functions and postprocessing methods related to semiconductor simulation. It also provides an interface to quickly define and modify TCAD models and their discretization grids. The core structure in `Semiconductors.jl` is `Semiconductors.Device`. This structure defines the device model and contains information about its geometry. The definition of this structure is given below:

```
struct Device
        model::Model                            # Device model and parameters
        grid::ExtendableGrids.ExtendableGrid    # FVM discretization grid
        b_types::Dict{Int64,String}             # Boundary region types
        r_types::Dict{Int64,String}             # Cell region types
end
```

The first field, `model::Model`, contains the device model and simulation parameters and will be described below. The second field, `grid::ExtendableGrids.ExtendableGrid`, contains the discretization grid. The grid object contains several *adjacencies* generated by `ExtendableGrids.jl` that describe the relationships among nodes, edges, and faces in the discretization. We refer the reader to the `ExtendableGrids.jl` documentation for further details on the grid object. The third field, `b_types::Dict{Int64,String}` maps boundary regions in the grid to their *boundary region types*. The boundary region type is a string which can take one of four values:

- `"contact"`: Describes an ideal Ohmic contact. Dirichlet boundary conditions are enforced in these regions.
- `"exterior"`: Describes an artificial simulation boundary. Homogeneous Neumann boundary conditions are enforced in these regions.
- `"interior"`: Describes a semiconductor-semiconductor interface. Typically used to separate regions in a device with discontinuous doping, permittivity or carrier lifetimes.
- `"gate"`: Describes a gate contact. Dirichlet boundary conditions are enforced for the potential in these regions.

The fourth field, `r_types::Dict{Int64,String}` maps regions in the grid to their *cell region types*. The cell region type is a string which can take one of two values:

- `"semiconductor"`: Describes a semiconductor region. The Poisson equation and both continuity equations are solved in these regions.
- `"insulator"`: Describes an insulator region. Typically used in gate oxide layers and in geometries containing a boundary mesh. Only the Poisson equation is solved in these regions. Space charge is neglected in these regions, i.e. $n = p = 0$.

For `b_types` and `r_types`, the region numbers correspond to the boundary region numbers and cell region numbers defined during the creation of `grid`.

Within each `Semiconductors.Device` is a `Semiconductors.Model`. The `Model` contains several physical constants and parameters necessary to simulate a device. The definition of this structure is given below:

```
mutable struct Model

        e0::Float64                     # Vacuum permittivity (constant)
        q::Float64                      # Electron charge (constant)
        vt::Float64                     # Thermal voltage at 300 K (constant)

        er::Dict{Int64,Float64}         # Relative permittivity by cell region
        ni::Dict{Int64,Float64}         # Intrinsic concentration by cell region
        tn::Dict{Int64,Float64}         # Electron lifetime by cell region
        tp::Dict{Int64,Float64}         # Hole lifetime by cell region
        ew::Dict{Int64,Float64}         # Metal-insulator workfunction by region

        # Intrinsic carrier concentration by boundary region
        ni_boundary::Dict{Int64,Float64}

        doping                          # Doping profile model (function)
        recomb                          # Recombination model (function)
        generation                      # Generation model (function)
        mobility_n                      # Electron mobility model (function)
        mobility_p                      # Hole mobility model (function)

        fldmob::Bool                    # Toggle field-dependent mobility
        impact::Bool                    # Toggle impact ionization

end
```

The structure is mutable to allow some parameters to be changed without recreating the `Model`. The first three fields give the values of the vacuum permittivity ε_0, the electron charge q and the thermal voltage V_T. The following values were used, according to the 2018 CODATA recommended values [150]:

$$\begin{aligned}
\varepsilon_0 &= 8.854\,187\,812\,8 \times 10^{-18}\,\mathrm{F\,\mu m^{-1}},\\
q &= 1.602\,176\,634 \times 10^{-19}\,\mathrm{C},\\
V_T &= \frac{kT}{q} = 25.851\,999\,786\,435\,5\,\mathrm{mV},
\end{aligned}$$

where we have assumed $T = 300\,\mathrm{K}$. The device temperature can be altered by changing the value of V_T. The temperature is assumed to be constant along the length of the device; self-heating and other thermal effects are not currently supported.

The next six fields define mappings between cell region numbers and material properties, allowing those properties to be piecewise constant. The field `er::Dict{Int64,Float64}` gives the unitless relative permittivity, which is typically 12.7 for silicon and 1 for air or vacuum, in each cell region. The field `ni::Dict{Int64,Float64}` gives the intrinsic carrier concentration n_i, which is typically $1.1 \times 10^{-2}\,\mathrm{\mu m^{-3}}$ [146], in each cell region. The fields `tn::Dict{Int64,Float64}` and `tp::Dict{Int64,Float64}` give the electron and hole lifetimes in each cell region. These quantities are related to the density of crystal defects in the silicon lattice and can vary significantly across fabrication processes [121]. We use the values $\tau_n = \tau_p = 1 \times 10^{-10}\,\mathrm{s}$ for most devices in this work.

The field `ew::Dict{Int64,Float64}` gives the metal-insulator workfunction potential E_w/q in each boundary region. This should only be defined for boundary regions of type `"gate"`. We use the value $E_w/q = 550\,\mathrm{mV}$ in this work, which is typical for a degenerate polysilicon contact on SiO_2 [144]. The field `ni_boundary::Dict{Int64,Float64}` gives the intrinsic carrier concentration n_i in each boundary region. This field is used to compute charge-neutrality carrier concentrations at Ohmic contacts. This field should only be defined for boundary regions of type `"contact"`, in which case it should take the value of n_i for whichever semiconductor region it contacts. If this value is required to vary along a contact, the contact should be split into multiple boundary regions to allow a piecewise constant n_i along the contact.

The next five fields define various physical models used by the simulator. The field `doping` defines the doping profile of the device. It should be defined using the template shown below:

```
function doping(x,y,reg;boundary=false)
        ...
        return (gamma)
end
```

The arguments x and y give the x and y-coordinate for which the doping should be evaluated. The argument reg gives the cell region number if boundary==false or the boundary region if boundary==true. The return value gamma should give the value of the doping at the point specified by x, y and reg. This interface allows a wide range of doping profiles to be specified. The doping can be a continuous function of x and y, a piecewise constant function of reg or a combination of both.

The field recomb defines the device recombination model. In this work, only the Shockley-Read-Hall recombination is used, which is defined as shown below:

```
function u_srh(n,p,ni,tn,tp)
        u = n*p - ni^2
        u /= (n+ni)*tp + (p+ni)*tn
        return (u)
end
```

The arguments n, p, ni, tn and tp give the electron and hole concentrations, intrinsic carrier concentration, and electron and hole lifetimes at the node at which the recombination should be evaluated, respectively. Additional recombination models such as the Auger model and the spontaneous radiative recombination rate can be added by following this function template. The field generation defines the device generation model. It should be defined using the template shown below:

```
function generation(e_norm,jn_norm,jp_norm,device)
        ...
        return (g)
end
```

The arguments e_norm, jn_norm and jp_norm give the least-squares approximations to the electric field, electron current density and hole current density in the cell containing the node at which the generation rate should be evaluated. This approximation is described in detail in Section 3.2.3. The argument device gives the Semiconductors.Device instance for the device to be simulated. This is included so that physical constants like q can be passed to the generation model. When impact ionization is enabled in this work, the Selberherr model given in (2.19) is used with the first set of parameters from Table 1 and Table 2 of [136], namely

$$\alpha_n^\infty = 1.0 \times 10^2\,\mu\text{m}^{-1}, \quad \alpha_p^\infty = 2.0 \times 10^2\,\mu\text{m}^{-1}, \quad E_{in} = 1.66 \times 10^2\,\text{V}\,\mu\text{m}^{-1}, \quad E_{ip} = 1.98 \times 10^2\,\text{V}\,\mu\text{m}^{-1},$$

where we have converted length units from cm to μm as appropriate.

The fields mobility_n and mobility_p define the device mobility models. They should be defined using the template shown below:

```
function mobility(n,p,e_norm,reg,fldmob)
        ...
        return (u)
end
```

The arguments n and p give the average electron and hole concentrations along the edge at which the mobility should be evaluated. These arguments are unused in the current implementation but may be used in a future implementation including concentration-dependent mobilities. Constructing these averages can be nontrivial; see [81] for details of such an implementation. The argument `e_norm` gives the least-squares approximation to the electric field in the cell containing the edge at which the mobility should be evaluated. This approximation is described in detail in Section 3.2.3. The argument `reg` gives the cell region number of the cell containing the edge at which the mobility should be evaluated. This can be used if a piecewise constant mobility is desired. The argument `fldmob` is set to `true` if a field-dependent mobility is to be simulated and `false` if not. The return value u should give the value of the mobility at the point specified by `reg`. The electron and hole mobility functions have identical argument and return value structures.

The remaining two fields in the `Semiconductors.Model` structure are `fldmob::Bool` and `impact::Bool`. These are set by the user and indicate whether field-dependent mobility and/or impact ionization should be included in the simulation, respectively.

3.2 Semiconductor simulator

3.2.1 Equilibrium solver

Two solvers were implemented using `VoronoiFVM.jl` to simulate instances of `Semiconductors.Device`. An equilibrium solver was implemented using the Poisson-Boltzmann equation given in (2.8), and a non-equilibrium solver was implemented using the full Van Roosbroeck system given in (2.6). The equilibrium solver was mainly used to provide an initial condition for the non-equilibrium solver when convergence problems were encountered. For details of the `VoronoiFVM` API, we refer the reader to [45]. Initial conditions for the equilibrium solver were computed using the charge-neutrality condition given in (2.9). In this reduced system, only the potential ψ is unknown.

The function `ic_equilib` was implemented to generate the charge-neutrality initial condition. The function definition is shown below:

```
function ic_equilib(
        d::Semiconductors.Device,
        sys::VoronoiFVM.AbstractSystem{Tv,Ti}
) where {Tv,Ti}

        ...
        return (ic)

end
```

The argument `d::Semiconductors.Device` is the `Device` instance to be simulated. The argument `sys::VoronoiFVM.AbstractSystem{Tv,Ti}` is the `VoronoiFVM` system containing the device physics functions, the current solution and the system Jacobian. The types `Tv` and `Ti` are used internally by `VoronoiFVM` to store values and integers, respectively. Typically, `Tv==Float64` and `Ti==Int32`. The return value `ic` is a $1 \times N$ `VoronoiFVM.SparseSolutionArray` if sparse unknown storage is enabled, or a $1 \times N$ `VoronoiFVM.DenseSolutionArray` if not, where N is the number of nodes in the discretization. The assembly loop for `ic` follows a similar elementwise algorithm to the `VoronoiFVM` PDE assembly loop `eval_and_assemble`, which is defined in `vfvm_solver.jl`. The algorithm is given in pseudocode below:

```
# n_cells is the number of cells (segments/triangles/tetrahedra) in the
# discretization
for icell in 1:n_cells

        # nodes_per_cell is the number of nodes per cell (2 per segment, 3 per
        # triangle, 4 per tetrahedron)
        for inode in 1:nodes_per_cell

                # Compute charge-neutral potential
                ...

                # Update the initial condition, or average it with whatever is
                # already there
                if ic[1,node.index]==0.0
                        ic[1,node.index] = v
                else
                        ic[1,node.index] = 0.5 * (ic[1,node.index]+v)
                end
        end
end
```

The algorithm loops over each cell in the discretization, and loops over each node within a cell. This means that some nodes may be visited multiple times if they are on a boundary between different cell regions. If the doping is region-dependent, this can lead to multiple potential values being computed for the same node. To work around this, the algorithm averages the potential with whatever is currently in the initial condition vector in the case that the node is visited multiple times. This value will likely be different than the actual charge-neutral potential, but some error is acceptable as long as the initial condition remains close enough to the solution to give convergence.

The function `equilib` was implemented to perform the thermal-equilibrium simulation. The function definition is shown below:

```
function equilib(
        device::Semiconductors.Device;
        Plotter=nothing,
        verbose=false,
        unknown_storage=:sparse,
        damp_initial=0.1
)

        # Solve system
        ...

        return (solution)

end
```

The argument `device::Semiconductors.Device` is the `Device` instance to be simulated. The keyword argument `Plotter` can be set to a `GridVisualize`-compatible plotter, typically `PyPlot`, to allow plotting of the equilibrium potential. The remaining keyword arguments `verbose`, `unknown_storage` and `damp_initial` are passed to `VoronoiFVM.solve!` in a `VoronoiFVM.SolverControl` instance. They dictate whether the solver will print debug information, whether to use sparse or dense unknown storage and the initial damping ratio for the Newton solver, respectively. This function is responsible for creating the physics callbacks defining the discretization. The flux, source and reaction callbacks for this system are given below:

```
function flux!(f,u,edge)
        eps = m.e0 * m.er[edge.region]
        f[1] = eps * (u[1,1]-u[1,2])
end

function source!(f,node)
        if device.r_types[node.region]=="semiconductor"
                f[1] = m.q * m.doping(node[1],node[2],node.region)
        end
end

function reaction!(f,u,node)
        if device.r_types[node.region]=="semiconductor"
                ni = m.ni[node.region]
                f[1] = 2*m.q*ni * sinh(u[1]/m.vt)
        end
end
```

Here, `m` is the `Semiconductors.Model` instance belonging to the device to be simulated. The source and reaction terms are only defined in the semiconductor regions, i.e. the regions where `device.r_types[node.region]=="semiconductor"`. In all other regions, the `source!` and `reaction!` callbacks leave the source and reaction terms unmodified at their default values of `0.0`.

The boundary condition physics callback is more involved. Dirichlet boundary conditions must be imposed at all Ohmic contacts using the charge-neutrality condition given in (2.9). This is simple in 1D, since each contact is a single point and can have only one doping value. In higher dimensions, the doping concentration can vary along the contact, requiring a multivalued Dirichlet boundary condition. This functionality can be implemented with `VoronoiFVM` by using the `bcondition` physics callback. The callback for this system is given in pseudocode below:

```
function bcond!(f,u,bnode)

        # Neumann BCs are assumed at non-contact boundaries
        if device.b_types[bnode.region]=="contact"

                # Compute equilibrium carrier densities for each
                # boundary node
                ...

                # Apply Dirichlet BCs for potential and concentrations
                boundary_dirichlet!(f,u,bnode,1,reg,v)
        # At gate contacts, potential is constrained by the
        # metal-insulator workfunction
        elseif device.b_types[bnode.region]=="gate"

                # Look up workfunction and set Dirichlet condition for
                # potential only
                boundary_dirichlet!(f,u,bnode,1,reg,m.ew[reg])

        end
end
```

Here, the function `VoronoiFVM.boundary_dirichlet!` is used to apply a separate Dirichlet boundary condition at each node, which amounts to setting a multivalued boundary condition along the contact. No boundary conditions are set at non-contact boundary regions, and homogeneous Neumann boundary conditions are imposed by default. Dirichlet boundary conditions are also imposed at gate contacts, where $\psi = E_w/q$ since no external bias is applied in the thermal equilibrium condition.

3.2.2 Non-equilibrium solver

The non-equilibrium solver must solve the full Van Roosbroeck system given in (2.6) for ψ, n and p. Its initial condition is either provided by the equilibrium solver or by a previous non-equilibrium solution. To generate the initial condition from the output of the equilibrium solver, the carrier concentrations are first computed using the Boltzmann approximations given in (2.7). The initial condition for the non-equilibrium solver should be a $3 \times N$ `AbstractArray`, where the three rows give the values of ψ, n and p. The array can be constructing by stacking the arrays for ψ, n and p vertically, i.e. by using `vcat`.

The function `non_equilib_sys` was implemented to generate the `VoronoiFVM.System` defining the physics callbacks for the non-equilibrium simulation. The function definition is shown below:

```
function non_equilib_sys(
        device::Semiconductors.Device,
        n_contacts::Int64;
        unknown_storage=:sparse,
        qfp=false,
        auto_sparsity=false
)

        # Generate system
        ...

        return (sys)

end
```

The argument `device::Semiconductors.Device` is the `Device` instance to be simulated. The argument `n_contacts::Int64` gives the number of Ohmic contacts in the device to be simulated. This value is required by the bias loop that sets Dirichlet boundary conditions. The keyword argument unknown_storage dictates whether the Newton solver should use sparse or dense unknown storage. The keyword argument qfp dictates whether the system should be discretized using quasi-Fermi potentials. This feature is not currently implemented; early computational experiments showed that the discretization using quasi-Fermi potentials led to slower solves and caused convergence problems for some devices. The argument auto_sparsity dictates whether the solver should use automatic Jacobian sparsity detection if either field-dependent mobility or impact ionization is requested.

The physics callbacks used by non_equilib_sys are generated one of two ways. If a field-dependent mobility or impact ionization is requested in simulation, the generic callback is used to define a *generic operator* which performs the vector discretizations used to estimate $\mathbf{E}$, $\mathbf{J_n}$ and $\mathbf{J_p}$. This case is described in detail in Section 3.2.3. If neither a field-dependent mobility nor impact ionization is requested in simulation, the callbacks are defined by the function physics_noneq_boltz. The definition of this function is shown below:

```
function physics_noneq_boltz(device::Semiconductors.Device)
        ...
        return (flux!,source!,reaction!,bcond!)
end
```

The argument device::Semiconductors.Device is the Device instance to be simulated. The return values flux!, source!, reaction! and bcond! are the physics callbacks. The flux callback is given below:

```
function flux!(f,u,edge)

        # Poisson equation flux discretization via finite differencing
        eps = m.e0*m.er[edge.region]
        f[1] = eps*(u[1,1]-u[1,2])

        # Continuity equation flux discretizations via Scharfetter-Gummel
        if device.r_types[edge.region]=="semiconductor"

                # Compute B(deltaV) for exponentially-fitted upwinding
                bp,bm = fbernoulli_pm((u[1,2]-u[1,1])/m.vt)

                # Get electron and hole mobilities
                un = m.mobility_n(0,0,0,edge.region,m.fldmob)
                up = m.mobility_p(0,0,0,edge.region,m.fldmob)

                # Compute electron and hole fluxes
                f[2] = un*m.vt * (u[2,1]*bm-u[2,2]*bp)
                f[3] = up*m.vt * (u[3,1]*bp-u[3,2]*bm)

        end
end
```

As before, m is the Semiconductors.Model instance belonging to the device to be simulated. This callback is similar to the flux callback used in the equilibrium solver, with the addition of the two continuity equations. The continuity equations are only solved in semiconductor regions; elsewhere, it is assumed that $n = p = 0$. The arguments for `n`, `p` and `e_norm` to `m.mobility_n` and `m.mobility_p` are set to the dummy value of `0.0` since a field-dependent mobility was not requested. The source and reaction callbacks are given below:

```
function source!(f,node)
        if device.r_types[node.region]=="semiconductor"
                f[1] = m.q*m.doping(node[1],node[2],node.region)
        end
end

function reaction!(f,u,node)
        if device.r_types[node.region]=="semiconductor"
                ni = m.ni[node.region]
                tn = m.tn[node.region]
                tp = m.tp[node.region]
                recomb = m.recomb(u[2],u[3],ni,tn,tp)
                f[1] = m.q*(u[2]-u[3])
                f[2] = recomb
                f[3] = recomb
        end
end
```

These callbacks are identical to the ones used in the equilibrium solver, except that the recombination term is added to the reaction callback for the continuity equations. The boundary condition callback for this system is identical to the one used in the equilibrium solver, except a nonzero bias voltage is set at each contact, and Dirichlet boundary conditions are added for the carrier concentrations:

```
function bcond!(f,u,bnode)

        # Get bias voltage of the current contact
        bias = parameters(u)[reg]

        # Neumann BCs are assumed at non-contact boundaries
        if device.b_types[bnode.region]=="contact"

                # Compute equilibrium carrier densities for each
                # boundary node
                ...

                # Apply Dirichlet BCs for potential and concentrations
                boundary_dirichlet!(f,u,bnode,1,reg,v+bias)
                boundary_dirichlet!(f,u,bnode,2,reg,n0)
                boundary_dirichlet!(f,u,bnode,3,reg,p0)

        # At gate contacts, potential is constrained by the
        # metal-insulator workfunction
        elseif device.b_types[bnode.region]=="gate"

                # Look up workfunction and set Dirichlet condition for
                # potential only
                boundary_dirichlet!(f,u,bnode,1,reg,m.ew[reg]+bias)

        end
end
```

Here, the bias voltage at each contact is stored in the solution array `u` using the `VoronoiFVM.parameters` interface. This interface allows the bias voltages to be passed directly to `VoronoiFVM.solve!` using the keyword argument `params`, which avoids the need to re-create or modify the `VoronoiFVM.System` instance for each bias point.

The function `non_equilib` was implemented to simulate the `VoronoiFVM.System` created by `non_equilib_sys` at multiple bias points to form an I-V curve. The function definition is shown below:

```
function non_equilib(
        device::Semiconductors.Device,
        bias_list::AbstractArray,
        ic::AbstractArray;
        Plotter=nothing,
        verbose=false,
        damp_initial=1.0,
        int_contacts=nothing,
        tf_conc=false,
        max_round=1000,
        max_iterations=100,
        catch_conv=false,
        tol_absolute=1e-10,
        tol_relative=1e-10,
        return_tfs=false
)

        ...

        if return_tfs
                if tf_conc
                        return (j_list,sys,solution,tfs_n,tfs_p)
                else
                        return (j_list,sys,solution,tfs)
                end
        else
                return (j_list,sys,solution)
        end

end
```

The argument `device::Semiconductors.Device` is the `Device` instance to be simulated. The argument `bias_list::AbstractArray` is the list of bias points at which the device should be simulated. This argument should be an $L \times M$ array, where L is the number of bias points to simulate and M is the number of Ohmic contacts in `device`. The argument `ic::AbstractArray` is the initial condition to be used for the damped Newton solver. This argument should be a $3 \times N$

array, where N is the number of nodes in the discretization. The three rows of ic should contain the values of ψ, n and p, respectively, for each node in the discretization. The ordering of the nodes is determined by the implementation of the discretization grid device.model.grid.

The keyword argument Plotter can be set to a GridVisualize-compatible plotter, typically PyPlot, to allow plotting of the non-equilibrium potential. The keyword arguments verbose and damp_initial are passed to VoronoiFVM.solve! in a VoronoiFVM.SolverControl instance. They dictate whether the solver will print debug information and the initial damping ratio for the Newton solver, respectively. The keyword argument int_contacts dictates the contacts at which boundary integration should be performed to evaluate terminal currents. This argument should be an AbstractVector containing the boundary region numbers of the contacts at which terminal currents should be evaluated. The ordering of the columns in the return value j_list is determined by the ordering of int_contacts.

The keyword argument tf_conc dictates whether the concentration-weighted test functions defined in (2.90) and (2.91) should be used for boundary integration instead of the Laplace test function defined in (2.84). Setting this argument to true will substantially increase simulation time but can potentially improve the accuracy of terminal currents. The keyword argument max_roud dictates the number of consecutive iterations within roundoff tolerance required to accept a solution. This argument is passed directly to VoronoiFVM.solve! as a VoronoiFVM.SolverControl instance; we refer the reader to the VoronoiFVM.jl documentation for more details on this argument.

The keyword argument max_iterations dictates the maximum number of Newton iterations that can be performed before a VoronoiFVM.ConvergenceError is thrown. The keyword argument catch_conv dictates whether such convergence errors should be caught or allowed to terminate the program. If catch_conv==true, the function non_equilib will return without error at the current bias point and print the debug message "Caught ConvergenceError" to the console. This is typically used when non_equilib is used as part of a larger curve-tracing algorithm in which convergence errors are expected. The keyword arguments tol_absolute and tol_relative give the absolute and relative tolerances for the Newton solver; these tolerances are defined as ϵ_2 and ϵ_3 in (2.76) and (2.77), respectively. The keyword argument return_tfs dictates whether the test functions used in boundary integration should be returned.

The return values of non_equilib can take one of three forms depending on the function arguments. The first three return values are always j_list, sys and solution. The return value j_list is a length(int_contacts) by size(bias_list,1) Matrix containing the terminal currents at each of the desired Ohmic contacts, for each bias point in bias_list. The return value sys is the VoronoiFVM.System generated by non_equilib_sys. This is typically used to avoid the need to re-generate sys when other simulations are run using the same device. The return value solution is a $3 \times N$ VoronoiFVM.SparseSolutionArray containing the values of ψ, n and p at the last bias point in bias_list. The user must call non_equilib multiple times if the full solution is desired at multiple bias points.

The structure of the remaining return values is dictated by the keyword argument return_tfs. If return_tfs==true, non_equilib returns five values, the last two of which are tfs_n and tfs_p. Each of these return values is a length(int_contacts)-element Vector of Vector{Float64}s, each element of which contains the concentration-weighted test function used for boundary integration of electrons and holes, respectively, at the corresponding contact. The ordering of these return values is determined by the ordering of the keyword argument int_contacts. If return_tfs==false, non_equilib returns four values, the last of which is tfs. This has an identical structure to tfs_n and tfs_p and contains the Laplace test functions used for boundary integration at the contacts specified by the keyword argument int_contacts.

Test functions for boundary integration are typically generated using the `VoronoiFVM` method `testfunction` with a `VoronoiFVM.TestFunctionFactory` instance. This method creates a Laplace test function by creating a simple `VoronoiFVM.System` describing a finite volume discretization of the Laplace equation. The resulting system of equations is linear and can be solved exactly with a single Newton iteration. If a concentration-weighted test function is desired, a different system must be created to discretize the concentration-weighted PDEs described in Section 2.2.4. The function `tf_sys` was implemented in `Semiconductors.jl` to generate the discretization used in such a test function. The function definition is shown below:

```
function tf_sys(
        system::VoronoiFVM.AbstractSystem{Tv},
        sol::VoronoiFVM.SparseSolutionArray,
        device::Semiconductors.Device,
        bc0::Vector,
        bc1::Vector;
        jp=false
) where Tv

        # Generate test function system
        ...

        return (factory)

end
```

The argument `system::VoronoiFVM.AbstractSystem{Tv}` is the `VoronoiFVM` system containing the device physics functions, the current solution and the system Jacobian. The argument `sol::VoronoiFVM.SparseSolutionArray` contains the current solution at all nodes in the discretization. This argument has size $3 \times N$, where the three rows give the values of ψ, n and p. The argument `device::Semiconductors.Device` is the `Device` instance to be simulated.

The arguments `bc0::Vector` and `bc1::Vector` give the boundary region numbers of the inactive contacts and the active contacts, respectively. The *active contacts* are the Ohmic contacts through which boundary integration should be performed. The *inactive contacts* are all other contacts. Typically, `bc1` has length 1, corresponding to a single active contact. If multiple active contacts are specified, the current through all active contacts will be summed together in the integration. The keyword argument `jp` dictates whether the test function will be used to compute hole current. The function `tf_sys` is typically called twice, once with `jp==false` and once with `jp==true`, to generate electron and hole current test functions, respectively.

The function `tf_sys` generates the callbacks used in the finite-volume discretization of the concentration-weighted test function system. Specifically, we seek to discretize the PDEs given by (2.90) and (2.91) for electrons and holes, respectively. By inspection of the PDEs, these discretizations should only have a flux part; their source and reaction terms are both zero. Using the Scharfetter-Gummel discretization given in (2.64) and (2.65), we can determine the values of n and p between grid segments in terms of the function $Q(x)$, which is defined in (2.43). The resulting fluxes for the electron and hole test functions are

$$g_{tn}[k,l] = \left(Q\left(-\frac{\psi[l]-\psi[k]}{V_T}\right)n[k] + Q\left(\frac{\psi[l]-\psi[k]}{V_T}\right)n[l]\right)\frac{T_n[l]-T_n[k]}{n_i}, \tag{3.1}$$

and

$$g_{tp}[k,l] = \left(Q\left(\frac{\psi[l]-\psi[k]}{V_T}\right)p[k] + Q\left(-\frac{\psi[l]-\psi[k]}{V_T}\right)p[l]\right)\frac{T_p[l]-T_p[k]}{n_i}. \tag{3.2}$$

The fluxes given in (3.1) and (3.2) are implemented in the `flux` physics callback passed to `VoronoiFVM.System` by `tf_sys`. As described in Section 2.2.4, we impose the Dirichlet boundary conditions $T_n = T_p = 1.1$ at active contacts and $T_n = T_p = -0.1$ at inactive contacts. These boundary conditions are implemented in the `bcondition` physics callback.

The test function systems generated by `tf_sys` must be solved at each bias point to perform terminal integration. The function `tf_solve` was implemented to solve these systems and generate the electron and hole current test functions for boundary integration. The function definition is shown below:

```julia
function tf_solve(
        factory::VoronoiFVM.TestFunctionFactory{Tv},
        ic::Vector{Float64}
) where Tv

        # Solve test function system
        ...

        return (sol_smooth)

end
```

The argument `factory::VoronoiFVM.TestFunctionFactory{Tv}` is the `TestFunctionFactory` instance containing the test function system to be solved. The argument `ic::Vector{Float64}` contains an initial condition to be used by the Newton solver when solving the test function system. This initial condition is passed by `Semiconductors.non_equilib` at each bias point. It is initialized to a vector of zeros before the first simulation, and it is updated to hold the test function used at the previous bias point in subsequent simulations. No method is currently implemented to generate an initial condition for the first simulation; consequently, if concentration-weighted test functions are used, it is often best to start any I-V curve simulation with all applied bias voltages set to 0 V and gradually ramp up to higher voltages to help convergence of the test function system.

The return value `sol_smooth` is a `Vector{Float64}` containing the smoothed test function. As described in Section 2.2.4, the test function is smoothed by clamping the solution to the range $[0, 1]$ and smoothing the clamped solution using the polynomial $s(x)$ defined in (2.92). Another choice of smoothing function is $s_1(x) = \sin^2(\pi x/2)$, which has similar properties to $s(x)$. Computational experience shows that these two smoothing functions produce nearly identical currents when used in boundary integration.

3.2.3 Vector discretization methods

The non-equilibrium solver included in `Semiconductors.jl` is capable of simulating a field-dependent mobility, impact ionization, or both. These effects require modifications to the standard `VoronoiFVM` finite volume discretization due to their dependence on the vector quantities $\mathbf{E}$, $\mathbf{J_n}$ and $\mathbf{J_p}$. Several *vector discretization* methods for estimating these quantities in a finite volume discretization are analyzed in [151] and [152]. The vector discretization used by `Semiconductors.jl` is an implementation of the "element-based method" described in [151]. Our implementation finds the

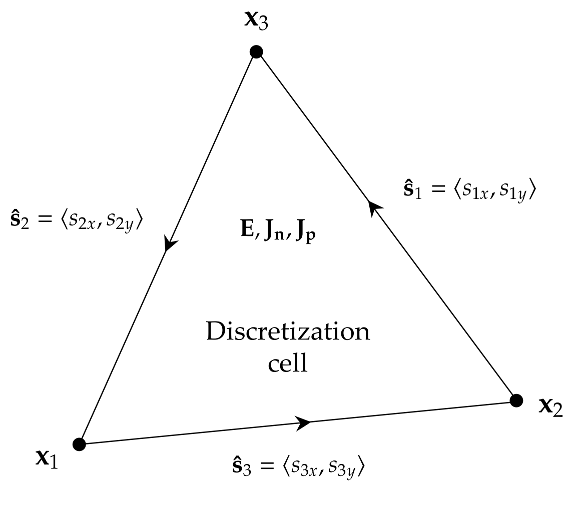

$\mathbf{x}_3$
$\hat{\mathbf{s}}_1 = \langle s_{1x}, s_{1y} \rangle$
$\hat{\mathbf{s}}_2 = \langle s_{2x}, s_{2y} \rangle$
$\mathbf{E}, \mathbf{J_n}, \mathbf{J_p}$
Discretization cell
$\mathbf{x}_2$
$\mathbf{x}_1$
$\hat{\mathbf{s}}_3 = \langle s_{3x}, s_{3y} \rangle$

Figure 3.1: Notation used in 2D vector discretization method. The tangent vectors $\hat{\mathbf{s}}_1$ through $\hat{\mathbf{s}}_3$ point along the edges of the triangle as shown.

least-squares approximation to the electric field and current densities on each discretization cell (triangles in 2D, segments in 1D).

The vector discretization used by `Semiconductors.jl` will be described here for a 2D geometry. The generalization to 1D or 3D is straightforward. Consider an arbitrary discretization cell as notated in Figure 3.1. We first seek to estimate the electric field $\mathbf{E} = -\nabla\psi$, which is assumed to be constant on the triangle. The tangent vectors $\hat{\mathbf{s}}_1$ through $\hat{\mathbf{s}}_3$ are

$$\hat{\mathbf{s}}_1 = \frac{\mathbf{x}_3 - \mathbf{x}_2}{||\mathbf{x}_3 - \mathbf{x}_2||}, \quad \hat{\mathbf{s}}_2 = \frac{\mathbf{x}_1 - \mathbf{x}_3}{||\mathbf{x}_1 - \mathbf{x}_3||}, \quad \hat{\mathbf{s}}_3 = \frac{\mathbf{x}_2 - \mathbf{x}_1}{||\mathbf{x}_2 - \mathbf{x}_1||}, \tag{3.3}$$

where $\mathbf{x}_1$ through $\mathbf{x}_3$ are the points at each corner of the triangle. Using an identical approach to the Voronoi finite volume flux discretization, we can write the projections of $\mathbf{E}$ onto $\hat{\mathbf{s}}_1$ through $\hat{\mathbf{s}}_3$ as E_1 through E_3, where

$$E_1 = \mathbf{E} \cdot \hat{\mathbf{s}}_1 = \frac{\psi[2] - \psi[3]}{||\mathbf{x}_3 - \mathbf{x}_2||}, \quad E_2 = \mathbf{E} \cdot \hat{\mathbf{s}}_2 = \frac{\psi[3] - \psi[1]}{||\mathbf{x}_1 - \mathbf{x}_3||}, \quad E_3 = \mathbf{E} \cdot \hat{\mathbf{s}}_3 = \frac{\psi[1] - \psi[2]}{||\mathbf{x}_2 - \mathbf{x}_1||}. \tag{3.4}$$

We can combine (3.3) and (3.4) into an overdetermined linear system $\mathbf{SE} = \mathbf{p}$, where

$$\mathbf{S} = \begin{bmatrix} s_{1x} & s_{1y} \\ s_{2x} & s_{2y} \\ s_{3x} & s_{3y} \end{bmatrix}, \quad \mathbf{E} = \begin{bmatrix} E_x \\ E_y \end{bmatrix}, \quad \mathbf{p} = \begin{bmatrix} E_1 \\ E_2 \\ E_3 \end{bmatrix}. \tag{3.5}$$

The system in (3.5) can be solved by a least-squares approximation, which gives

$$\mathbf{E} = \begin{bmatrix} E_x \\ E_y \end{bmatrix} = \left(\mathbf{S}^T\mathbf{S}\right)^{-1} \mathbf{S}^T\mathbf{p}. \tag{3.6}$$

The solution in (3.6) can be computed in Julia using the backslash operator, which computes the least-squares approximation of an overdetermined system.

For the electric field, the solution given in (3.6) is exact by construction; we should have $||\mathbf{SE} - \mathbf{p}|| = 0$ to within machine precision since the finite-volume discretization assumes $\mathbf{E}$ is constant on each triangle. For the current densities, however, the 2D Scharfetter-Gummel discretization implies a different current density along each edge, as described in Section 2.2.2 and in [123]. In this case, the least-squares solution will not be exact but can still give a good approximation for $\mathbf{J_n}$ and $\mathbf{J_p}$. Using a similar approach to (3.4), we can compute the projections of $\mathbf{J_n}$ and $\mathbf{J_p}$ onto the triangle edges:

$$J_{n1} = \mathbf{J_n} \cdot \hat{\mathbf{s}}_1 = \frac{q\mu_n[2,3]V_T}{||\mathbf{x}_3 - \mathbf{x}_2||}\left(B\left(-\frac{\psi[3]-\psi[2]}{V_T}\right)n[2] - B\left(\frac{\psi[3]-\psi[2]}{V_T}\right)n[3]\right), \tag{3.7}$$

$$J_{p1} = \mathbf{J_p} \cdot \hat{\mathbf{s}}_1 = \frac{q\mu_p[2,3]V_T}{||\mathbf{x}_3 - \mathbf{x}_2||}\left(B\left(\frac{\psi[3]-\psi[2]}{V_T}\right)p[2] - B\left(-\frac{\psi[3]-\psi[2]}{V_T}\right)p[3]\right). \tag{3.8}$$

The remaining projections J_{n2}, J_{p2}, J_{n3} and J_{p3} are defined analogously to (3.7) and (3.8). We have here assumed that the mobilities $\mu_n[2,3]$ and $\mu_p[2,3]$ have already been estimated from the projections of $\mathbf{E}$; in practice, this means that the electric field must be fully estimated before either of the current densities are computed if a field-dependent mobility is desired.

The function `cell_field_norms` was implemented to estimate the magnitude of the electric field and the electron and hole current densities in a discretization cell. The function definition is shown below:

```julia
function cell_field_norms(
        u::VoronoiFVM.SparseSolutionArray,
        sys::VoronoiFVM.System,
        device::Semiconductors.Device,
        idx::VoronoiFVM.SparseSolutionIndices,
        icell::Int32,
        edge::VoronoiFVM.Edge,
        edges_per_cell::Int32
)

        # Compute field magnitudes
        ...

        return (e_norm,jn_norm,jp_norm)

end
```

The argument `u::VoronoiFVM.SparseSolutionArray` contains the current solution at all nodes in the discretization. The argument `sys::VoronoiFVM.System` is the `VoronoiFVM` system containing the device physics functions, the current solution and the system Jacobian. The argument `device::Semiconductors.Device` is the `Device` instance to be simulated. The argument `idx::VoronoiFVM.SparseSolutionIndices` is used to determine the indices of specific values of ψ, n and p in the solution u. The argument `icell::Int32` is the index of the discretization cell in which the fields should be estimated. The argument `edge::VoronoiFVM.Edge` is used by `VoronoiFVM._fill!` to get the indices of the nodes that lie on each edge of the discretization cell. The argument `edges_per_cell::Int32` contains the number of edges in each discretization cell. In 1D, `edges_per_cell==1`; in 2D, `edges_per_cell==3`; in 3D, `edges_per_cell==6` if `device.grid` uses a tetrahedral tessellation.

The standard `VoronoiFVM` callback interface cannot be used to define the physics of the simulation if a field-dependent mobility or impact ionization is to be simulated. In the standard interface, the flux callback can only access ψ, n and p at the two nodes on an edge. Similarly, the reaction discretization can only access ψ, n and p at a single node. The vector discretization scheme above depends on unknowns at all nodes in each discretization cell. In 2D, this means that the flux discretization requires 3 values of ψ if a field-dependent mobility is desired, and the reaction discretization requires 3 values of ψ, n and p if impact ionization is desired. In this case, the `generic` callback in `VoronoiFVM.Physics` is used to define a *generic operator* that performs the finite volume discretization.

The function `_generic_noneq` was implemented to perform the finite volume discretization when vector discretization is required. The function definition is shown below:

```
function _generic_noneq!(
        f,
        u::AbstractArray{Tu},
        sys::VoronoiFVM.AbstractSystem{Tv,Ti},
        device::Semiconductors.Device;
        tf=nothing
) where {Tu,Tv,Ti}

        # Compute residual, and compute boundary integral if requested
        ...

        if tf==nothing
                return (nothing)
        else
                return (integral)
        end
end
```

The argument `f` contains the current residual at all nodes in the discretization. The argument `u::AbstractArray{Tu}` contains the current solution at all nodes in the discretization. Both `f` and `u` are passed by the `VoronoiFVM` assembly routine `VoronoiFVM.eval_and_assemble`. The type `Tu` is used internally by VoronoiFVM to store the solution and to compute the system Jacobian. Typically, `Tu==Float64`, but `Tu==ForwardDiff.Dual` is also possible during Jacobian evaluation. The argument `sys::VoronoiFVM.AbstractSystem{Tv,Ti}` is the `VoronoiFVM` system containing the device physics functions, the current solution and the system Jacobian. The types `Tv` and `Ti` are used internally by `VoronoiFVM` to store values and integers, respectively. Typically, `Tv==Float64` and `Ti==Int32`. The argument `d::Semiconductors.Device` is the `Device` instance to be simulated.

The keyword argument `tf` can be used to provide a test function for boundary integration. If `tf!=nothing`, the function `_generic_noneq` computes the flux part of the boundary integrals of n and p and returns it. If `tf==nothing`, the function `_generic_noneq` returns `nothing`. The return value is used when boundary currents are requested in a simulation using a field-dependent mobility, impact ionization or both. In this case, the standard `VoronoiFVM.integrate` method will not work properly due to the use of the generic operator. The function `integrate_generic` calls `_generic_noneq` to compute the flux part of the boundary integral and adds the source and reaction terms in a separate loop. This function will be described in more detail below.

The function `_generic_noneq` implements an assembly loop similar to the elementwise algorithm used in the `VoronoiFVM` PDE assembly loop `eval_and_assemble`, which is defined in `vfvm_solver.jl`. The algorithm is given in pseudocode below:

```
# n_cells is the number of cells (segments/triangles/tetrahedra) in the
# discretization
for icell in 1:n_cells

        # Call cell_field_norms to estimate E, Jn, Jp if cell is in a
        # semiconductor region
        ...

        # Only do reaction discretization if impact ionization is enabled
        if m.impact
                for inode in 1:nodes_per_cell

                        # Add reaction part of residual if cell is in a
                        # semiconductor region
                        ...

                end
        end

        # Only do flux discretization if field-dependent mobility is enabled
        if m.fldmob
                for iedge in 1: edges_per_cell

                        # Add flux part of Poisson equation residual in all
                        # regions
                        ...

                        # Add flux part of continuity equation residuals if cell
                        # is in a semiconductor region
                        ...

                end
        end

end
```

Here, the generic operator is only used to provide the residuals that cannot be defined using the standard callback interface. Since impact ionization modifies the reaction terms of the residual, the function `_generic_noneq` should only perform the reaction discretization if impact ionization is enabled. Similarly, field-dependent mobilities only affect the carrier flux terms, so `_generic_noneq` should only perform the flux discretization if field-dependent mobility is enabled. When either of these terms is not defined by the generic operator, `non_equilib` uses the standard `reaction!` or `flux!` callback defined by `physics_noneq_boltz` in creating the `VoronoiFVM.System` for the simulation. These terms could also be defined by the generic operator, but the Jacobian evaluation is generally slower if only the generic operator is used. This is because the assembly routine `VoronoiFVM.eval_and_assemble` differentiates each callback separately during Jacobian evaluation, which allows the Jacobian to be assembled block-by-block instead of in a single pass.

When a generic operator is needed in the discretization, `VoronoiFVM` uses `SparseDiffTools.jl` to accelerate Jacobian evaluation by exploiting the sparsity of the residual function. This package uses graph coloring to create a compressed representation of the Jacobian that is more easily evaluated than the full Jacobian [46]. This method requires knowledge of the *sparsity pattern* of the Jacobian, which defines the location of nonzero entries. Sparsity detection can be performed automatically using `Symbolics.jl` [52]. This approach uses symbolic variables as arguments to the generic operator to determine dependencies between its outputs and inputs. Computational experience shows that this method is typically slow and can require several seconds to generate a sparsity pattern when the generic operator is complex. It also requires the generic operator to accept a vector of symbolic variables as its input, which can be nontrivial due to the number of external functions called by `_generic_noneq`.

An alternative to automatic sparsity detection is to manually define a sparse matrix containing the location of nonzero entries in the Jacobian. This is implemented in `Semiconductors.jl` by the function `_generic_noneq_sparsity`. The function definition is shown below:

```julia
function _generic_noneq_sparsity(
        sys::VoronoiFVM.AbstractSystem{Tv,Ti},
        device::Semiconductors.Device
) where {Tv,Ti}

        # Generate sparsity pattern
        ...

        return (sparsity)

end
```

The argument `sys::VoronoiFVM.AbstractSystem{Tv,Ti}` is the `VoronoiFVM` system containing the device physics functions, the current solution and the system Jacobian. The argument `d:: Semiconductors.Device` is the `Device` instance to be simulated. The types `Tv` and `Ti` are used internally by `VoronoiFVM` to store values and integers, respectively. Typically, `Tv==Float64` and `Ti==Int32`. The return value `sparsity` is a $N \times N$ `SparseArrays.SparseMatrixCSC` whose entries are 1 where the corresponding Jacobian entry is nonzero and 0 otherwise. The implementation of `_generic_noneq_sparsity` is identical to that of `_generic_noneq`, except it writes ones to the sparsity pattern matrix instead of computing and modifying values in the residual.

The `integrate` method provided by `VoronoiFVM` for boundary current integration assumes that the standard callback interface is used to define the discretization. When a generic operator is used in the discretization, the currents given by `integrate` will generally be incorrect since the residual terms contributed by the generic operator are not considered in the integration. To resolve this, the function `integrate_generic` was implemented in `Semiconductors.jl` to allow boundary current integration when a generic operator is used. This function contains code from `VoronoiFVM.integrate`, which has been modified to improve performance and accommodate the use of a generic operator. The function definition is shown below:

```julia
function integrate_generic(
        system::VoronoiFVM.AbstractSystem{Tv,Ti},
        tf::Vector{Tv},
        U::AbstractArray{Tu,2},
        device::Semiconductors.Device,
        idx::VoronoiFVM.SparseSolutionIndices
) where {Tu,Tv,Ti}

        # Perform boundary integration for electron and hole current
        ...

        return (integral)

end
```

The argument `sys::VoronoiFVM.AbstractSystem{Tv,Ti}` is the `VoronoiFVM` system containing the device physics functions, the current solution and the system Jacobian. The argument `tf::Vector{Tv}` is the test function to be used for terminal current integration. The argument `U::AbstractArray{Tu,2}` is the current solution. The argument `device::Semiconductors.Device` is the `Device` instance to be simulated. The argument `idx::VoronoiFVM.SparseSolutionIndices` is used to determine the indices of specific values of ψ, n and p in the solution. The return value `integral` is a 3-element `Vector{Tu}` containing the values of the boundary integration. In `VoronoiFVM.integrate`, `integral[1]` would contain the flux of the electric field through the active contact. Since this quantity is not physically meaningful, we do not compute it in `integrate_generic`. The elements `integral[2]` and `integral[3]` contain the electron and hole currents through the active contact. The currents must be multiplied by q to give a value in amps.

3.2.4 Continuation

Pseudo arc-length continuation was implemented to trace multivalued I-V curves. Our implementation uses `VoronoiFVM` data structures and assembly routines with a custom Newton solver and tangent predictor to perform the continuation algorithm. Specifically, a `VoronoiFVM.System` is used to store the device physics functions and the current Jacobian of the residual. An interface was created to allow the residual and Jacobian to be computed on-demand using `VoronoiFVM.eval_and_assemble`. This interface can be used by `Semiconductors.jl`, or the residual and Jacobian functions it provides can be passed to `BifurcationKit.jl` to perform any of the continuation algorithms in that package. The data structures `Semiconductors.IVState` and `Semiconductors.IVCurve` were created to hold the state of the nonlinear solver and of the continuation algorithm. This section describes the implementation of the `VoronoiFVM` continuation interface, the `IVState` and `IVCurve` data structures and the arc-length continuation algorithm.

The `Semiconductors.jl` continuation interface is similar to the `DifferentialEquations.jl` interface provided by `VoronoiFVM.jl`. The interface consists of three functions: `_bk_res_jac!`, `bk_residual` and `bk_jacobian`. The function `_bk_res_jac!` is used internally by `Semiconductors.jl` and should not be directly called by the user. This function uses `VoronoiFVM.eval_and_assemble` to assemble both the residual and the Jacobian into a `VoronoiFVM.System` instance. It then stores the hash of the current solution and the current parameter set in the `System` to ensure the same residual and Jacobian are not evaluated multiple times. The parameter set is the set of bias voltages applied to the contacts of the device to be simulated. The functions `bk_residual` and `bk_jacobian` are defined as shown below:

```
function bk_residual(sys,u,p)
        _bk_res_jac!(sys,u,p)
        sys.history.nf += 1
        f = copy(vec(sys.residual))
        return (f)
end

function bk_jacobian(sys,u,p)
        _bk_res_jac!(sys,u,p)
        sys.history.njac += 1
        j = sparse(sys.matrix)
        return (j)
end
```

The arguments sys, u and p should contain the VoronoiFVM.System belonging to the device to be simulated, the current solution, and the current parameter set, respectively.

The residual interface bk_residual must return a copy of sys.residual to avoid external modification of the residual. This is necessary for proper operation of the continuation algorithm: In the event that a continuation step fails, the algorithm must revert the residual stored in the IVState instance to the residual from before the continuation step failed (i.e. before any Newton updates took place). If sys.residual is modified, the algorithm will revert to an erroneous state and convergence will likely fail. The same is true of the Jacobian interface bk_jacobian. This function must also convert the Jacobian stored in sys to a SparseArrays.SparseMatrixCSC before returning it. The Jacobian stored in sys is typically a ExtendableSparse.ExtendableSparseMatrix, which is provided by ExtendableSparse.jl. This data structure is used to allow faster assembly of the Jacobian, but it is not currently compatible with vcat or hcat, which are both necessary to assemble the augmented system used in continuation.

The IVState and IVCurve structures were created to hold data used by the arc-length continuation algorithm. The IVState structure contains the current state of the algorithm and several parameters used by the Semiconductors.jl damped Newton solver. An abbreviated definition of this structure is given below:

```
mutable struct IVState{Tz,Tv,Ti}
        z::AbstractVector{Tz}           # New solution (updated inplace)
        v::Tv                           # New voltage (updated inplace)
        i::Tv                           # New current (updated inplace)
        zpred::AbstractVector{Tz}       # Predicted solution
        vpred::Tv                       # Predicted voltage
        ipred::Tv                       # Predicted current
        z0::AbstractVector{Tz}          # Old solution
        v0::Tv                          # Old voltage
        i0::Tv                          # Old current
        N::Tv                           # Arc length residual
        bias::AbstractVector{Tv}        # Bias vector
        c_active::Ti                    # Active contact
        dfdv::AbstractVector{Tv}        # Vector derivative dF/dV
        didz::AbstractVector{Tv}        # Gradient vector dI/dz
        dids::Tv                        # Scalar derivative dI/ds
        dvds::Tv                        # Scalar derivative dV/ds
        dzds::AbstractVector{Tv}        # Vector derivative dz/ds
        dndz::AbstractVector{Tv}        # Gradient vector dN/dz
        dndv::Tv                        # Scalar derivative dN/dV
end
```

The field `z::AbstractVector{Tz}` is used to store the current solution at all nodes in the discretization. The damped Newton solver updates this field inplace each iteration. The field `v::Tv` is used to store the current bias voltage at the active contact. Only one active contact is currently supported by the arc-length continuation algorithm. The field `i::Tv` is used to store the terminal current through the active contact. This field is updated by boundary integration at the end of each Newton iteration to compute the current residual in the arc-length constraint. The fields `zpred::AbstractVector{Tz}`, `vpred::Tv` and `ipred::Tv` are the solution, bias voltage and terminal current predicted by the tangent predictor. The fields `z0::AbstractVector{Tz}`, `v0::Tv` and `i0::Tv` are used to hold the solution, bias voltage and terminal current from the last successful continuation step. If a continuation step fails, the fields `z`, `v` and `i` are reverted to these values before retrying the step with a smaller step size $\Delta\sigma$.

The field `N::Tv` contains the current arc-length residual; this is computed using either of the constraints N_1 or N_2 defined in (2.97) and (2.98). The field `bias::AbstractVector{Tv}` holds the current bias voltages at all active contacts in the device. This should be set to an initial value by the user before running the continuation algorithm; the entry of `bias` corresponding to the active contact is modified by the continuation algorithm, and all other entries are left unchanged. The field `c_active::Ti` holds the number of the boundary region corresponding to the active contact.

The remaining seven fields hold derivatives used in the tangent vector computation and in the assembly of the block Jacobian for the augmented system. The derivatives $\partial I/\partial s$, $\partial V/\partial s$ and $\partial \mathbf{z}/\partial s$ are computed by `Semiconductors.get_tangent!`, which implements the tangent expressions given in [24]. The vector derivative $\partial \mathbf{F}/\partial V$ is sparse, as described in Section 2.2.6. It is assembled manually using the function `Semiconductors.assemble_dfdv`. The gradient vector $\partial I/\partial \mathbf{z}$ is computed by `Semiconductors.assemble_didz!`, which uses `ReverseDiff.gradient` to take the gradient of the boundary integration function. The gradient vector $\partial N/\partial \mathbf{z}$ and the scalar derivative $\partial N/\partial V$ are computed by `Semiconductors.assemble_dndz_dndv!`, which uses `ReverseDiff.gradient` to take the gradient of the arc-length constraint and `ForwardDiff.derivative` to compute the scalar derivative of the arc-length constraint.

A constructor was created for `IVState` to allow initialization of an `IVState` instance. The constructor is defined as shown below:

```julia
function IVState(
        ic::AbstractVector{Tz},
        bias::AbstractVector{Tv},
        c_active::Ti,
        ds::Tv
) where {Tz,Tv,Ti}

        ...
        return (s)

end
```

The argument `ic::AbstractVector{Tz}` gives the initial condition to be used by the continuation algorithm. This should be a $3N \times 1$ `Vector` giving the values of ψ, n and p at each node in the discretization. The initial condition can be generated from a solution produced by `non_equilib` by passing the $N \times 3$ solution array to `VoronoiFVM.values`, which will reshape the solution to a vector. The argument `bias::AbstractVector{Tv}` gives the initial bias voltages at each Ohmic contact in the device to be simulated. These voltages should be equal to the bias voltages used to generate the initial condition `ic`. If `ic` was generated by `equilib`, `bias` should be a vector of zeros. The argument `c_active::Ti` gives the boundary region number corresponding to the active contact. The argument `ds::Tv` sets the initial step size to be used by the continuation algorithm.

In addition to the values given by the arguments to the constructor, default values are set for the following parameters:

- `use_n1==true`: The constraint N_1, which defines a line emanating from the tangent vector at a distance $\Delta\sigma$, is used for arc-length continuation. Setting this field to `false` dictates that N_2 should be used instead.
- `log_j==false`: The raw value of the terminal current is used in the parameterization of the I-V curve. Setting this field to `true` dictates that the continuation algorithm should parameterize the I-V curve by the base-10 logarithm of the current, which can be useful if the current is expected to vary over several orders of magnitude.
- `scale_j==1.0`: No scaling is used for the current. Setting this field to a value other than `1.0` scales the current axis in the parameterization of the I-V curve.
- `tol_abs==1e-10`: The absolute tolerance of the damped Newton solver is set to $\epsilon_2 = 1 \times 10^{-10}$ using the convergence criterion given in (2.76).
- `tol_rel==1e-10`: The relative tolerance of the damped Newton solver is set to $\epsilon_3 = 1 \times 10^{-10}$ using the convergence criterion given in (2.77).
- `tol_mono==1e-3`: The damped Newton solver will terminate with an error if the residual increases by a factor of more than `1/tol_mono` between two iterations.
- `damp_initial==1.0`: No damping is used in the damped Newton solver by default. Setting this field to a value less than `1.0` enables damping in the solver if `n_damp>0`.
- `n_damp==0`: No damping is used in the damped Newton solver by default. Setting this field to a value greater than `0` dictates the number of damped Newton iterations that will be performed by the solver if `damp_initial<1.0`.
- `max_iters==10`: A maximum of 10 Newton iterations will be performed at each continuation step before retrying the step with a smaller step size.

The return value `s` is the `IVState` instance generated with the specified parameters.

The `IVCurve` structure contains the values on the I-V curve traced by the continuation algorithm and several parameters used in curve tracing. The definition of this structure is given below:

```
mutable struct IVCurve{Tv,Ti}
        v::AbstractVector{Tv}           # Holds voltage
        i::AbstractVector{Tv}           # Holds current
        tol_trunc::Tv                   # Truncation error tolerance
        max_steps::Ti                   # Maximum continuation steps
        step::Ti                        # Current step number
        ds_decrease_factor::Tv          # Factor to decrease ds by
        ds_max_increase::Tv             # Maximum increase of ds
        trunc_error_norm::Tv            # Norm of truncation error
        dsmin::Tv                       # Minimum ds
        n_solve::Ti                     # Number of Newton solves
        n_solve_max::Ti                 # Maximum Newton solves
        skip_restart::Bool              # Whether to skip restart
end
```

The fields `v::AbstractVector{Tv}` and `i::AbstractVector{Tv}` are used to hold the bias voltage and terminal current at each point on the I-V curve. Values are appended to `v` and `i` using `push!` at the end of each continuation step so that they appear in the order the curve was traced. The field `tol_trunc::Tv` dictates the maximum truncation error between two continuation steps such that a restart is not required. The field `max_steps::Ti` contains the maximum number of continuation steps that can be performed before the algorithm terminates with an error. The field `step::Ti` holds the current continuation step number. The fields `ds_decrease_factor::Tv` and `ds_max_increase::Tv` dictate the amount by which $\Delta\sigma$ decreases following an unsuccessful step and the maximum amount by which $\Delta\sigma$ can increase following a successful step. The default values are `ds_decrease_factor==0.5` and `ds_max_increase==2.0`.

The field `trunc_error_norm::Tv` is used to store the norm of the truncation error following a successful continuation step. Our implementation uses the step size selection algorithm described in [24] which attempts to maintain `trunc_error_norm<tol_trunc` at each continuation step. If this criterion is not met, the current continuation step is restarted and the step size is decreased by `ds_decrease_factor`. The field `dsmin::Tv` dictates the minimum step size allowed by the continuation algorithm. If the step size falls below `dsmin`, the algorithm will terminate with an error. The field `n_solve::Ti` holds the number of Newton solves performed by the continuation algorithm in the current run. The field `n_solve_max::Ti` dictates the maximum number of Newton solves allowed by the continuation algorithm. This is typically used to prevent long runtimes of the continuation algorithm when the step size becomes vanishingly small. The field `skip_restart::Bool` is used internally by the continuation algorithm to indicate that a restart has already been attempted in the current continuation step. This field should not be modified by the user.

A constructor was created for `IVCurve` to allow initialization of an `IVCurve` instance. The constructor is defined as shown below:

```
function IVCurve(
        state::IVState{Tz,Tv,Ti},
        vmin::Tv,
        vmax::Tv,
        dsmin::Tv
) where {Tz,Tv,Ti}

        ...
        return (s)

end
```

The argument `state::IVState{Tz,Tv,Ti}` is the `IVState` instance which will be used to trace the I-V curve. This argument is necessary in the constructor to ensure type consistency. The arguments `vmin::Tv` and `vmax::Tv` are currently unused but may be implemented in a future revision to allow lower and upper bounds to be set for the active contact bias voltage. The argument `dsmin::Tv` sets the minimum step length allowed by the continuation algorithm. In addition to the values given by the arguments to the constructor, default values are set for the following parameters:

- `tol_trunc==0.1`: A maximum truncation error of 0.1 is allowed between continuation steps; truncation error exceeding this value will trigger a restart of the current continuation step. This field should be set relative to the scaling of the voltage and current at the active contact.

- `max_steps==10`: A maximum of 10 continuation steps will be performed before the continuation algorithm terminates with an error.

- `ds_decrease_factor==0.5`: The step length $\Delta\sigma$ is decreased by this factor if a restart is required due to the truncation error criterion or a convergence error in the damped Newton solver. Setting this field to a value closer to 1 decreases the step size less aggressively in such situations.

- `ds_max_increase==2.0`: The step length $\Delta\sigma$ is allowed to increase by a factor between 1 and this value following a successful continuation step. Setting this field to a larger number allows more aggressive increases; setting this closer to 1 prevents excessive step size growth if multiple continuation steps are successful.

- `n_solve_max==20`: A maximum of 20 damped Newton solves are allowed by the continuation algorithm. This field should have a value of at least `max_steps` since at least one damped Newton solve is required for each continuation step.

The return value `s` is the `IVCurve` instance generated with the specified parameters.

The function `continuation!` was implemented to perform the arc-length continuation algorithm for tracing I-V curves. The function is defined as shown below:

```julia
function continuation!(
        state::IVState{Tz,Tv,Ti},
        curve::IVCurve{Tv},
        d::Semiconductors.Device,
        sys::VoronoiFVM.AbstractSystem{Tv},
        tf::AbstractArray{Tv}
) where {Tz,Tv,Ti}

        while curve.step<=curve.max_steps
                check_cont_terminate(state,curve)
                initialize!(state,d,sys,tf)
                predict!(state)
                newton!(state,d,sys,tf)
                print_cont_1(state,curve)
                ds_update!(state,curve,d,sys,tf)
                print_cont_2(state,curve)
                push!(curve.v,state.v)
                push!(curve.i,state.i)
        end
        return (nothing)

end
```

The argument `state::IVState{Tz,Tv,Ti}` is the `IVState` instance to be used by the continuation algorithm. The user-defined parameters in the `IVState` should be set before calling `continuation!`.

The argument `curve::IVCurve{Tv}` is the `IVCurve` instance to be used by the continuation algorithm. The user-defined parameters in the `IVCurve` should be set before calling `continuation!`. The argument `d::Semiconductors.Device` is the `Device` instance to be simulated.

The continuation algorithm relies on several other `Semiconductors.jl` functions that mutate the arguments `state` and `curve` to implement curve tracing. During each continuation step, the algorithm first calls `check_cont_terminate`, which terminates the continuation process with an error if the maximum number of Newton solves specified by `curve.n_solve_max` has been reached or if the minimum step size specified by `curve.dsmin` has been reached. Then, `initialize!` is called to prepare `state` for the continuation step. Specifically, this function resets the Newton solver convergence status and damping ratio, clears the Newton update set by the previous continuation step, sets `state.z0 = state.z` and `state.v0 = state.v`, and updates the terminal current and all the derivatives stored in `state`.

After this, `predict!` is called to populate the tangent vectors in `state` and predict the next point on the I-V curve. Then, `newton!` is called to correct the tangent prediction with the damped Newton solver. Debug information is then printed by `print_cont_1` and `print_cont_2`. The debug information is printed in two parts because some of the information depends on the step length $\Delta\sigma$, which is updated by calling `ds_update!`. Finally, the resulting bias voltage and terminal current are pushed to `curve.v` and `curve.i` if the continuation step was successful. This process repeats until the maximum number of continuation steps specified by `curve.max_steps` is reached, at which point the algorithm terminates without error and returns `nothing`.

3.2.5 Current boundary conditions

Current boundary condition functionality was implemented in `Semiconductors.jl`, which allows the user to fix the current through any Ohmic contact in a device. This is typically used in the simulation of current-controlled devices like bipolar transistors, where a constant base current is necessary to trace many common I-V characteristics. This functionality is supported by the data structures `Semiconductors.NewtonState` and `Semiconductors.NewtonParams`, which hold the state and parameters of the damped Newton solver used to solve the augmented system described in Section 2.2.5. These structures are similar to the `IVState` structure used in arc-length continuation. The constructor for `NewtonState` is defined as shown below:

```julia
function NewtonState(
        ic::AbstractVector{Tz},
        bias::AbstractVector{Tv}
) where {Tz,Tv}

        ...
        return (s)

end
```

The argument `ic::AbstractVector{Tz}` gives the initial condition to be used by the damped Newton solver. This should be a $3N \times 1$ `Vector` giving the values of ψ, n and p at each node in the discretization. The argument `bias::AbstractVector{Tv}` gives the initial bias voltages at each Ohmic contact in the device to be simulated. These voltages should be equal to the bias voltages used to generate the initial condition `ic`. If `ic` was generated by `equilib`, `bias` should be a vector of zeros. The return value `s` is the `NewtonState` instance generated with the specified initial condition and initial bias vector. The NewtonState structure contains no user-modifiable fields.

The constructor for NewtonParams is defined as shown below:

```
function NewtonParams(
        set_current::Tv,
        c_active::Ti;
        tol_abs=1e-10,
        tol_rel=1e-10,
        tol_mono=1e-3,
        max_iters=25,
        do_damp_search=false,
        damp_initial=0.1,
        damp_growth=1.5,
        damp_search_iters=10,
        damp_search_decrease=0.5,
        dirichlet_scale=1.0,
        verbose=verbose_noverbose
) where {Tv,Ti}

        ...
        return (p)

end
```

The argument set_current::Tv gives the desired current through the active contact. The argument c_active::Ti gives the boundary region number corresponding to the active contact. The keyword arguments tol_absolute and tol_relative give the absolute and relative tolerances for the Newton solver; these tolerances are defined as ϵ_2 and ϵ_3 in (2.76) and (2.77), respectively. The keyword argument tol_mono gives the monotonicity tolerance for the Newton solver. The damped Newton solver will terminate with an error if the residual increases by a factor of more than 1/tol_mono between two iterations. The keyword argument max_iters dictates the maximum number of iterations allowed by the damped Newton solver. The solver will terminate with an error if max_iters is exceeded.

The keyword argument do_damp_search dictates whether a search should be performed to determine the initial damping value used by the damped Newton solver. The keyword argument damp_initial gives the initial damping ratio for the damped Newton solver. If do_damp_search= =true, this value is the damping ratio at which the damping ratio search will begin. The keyword argument damp_growth gives the amount by which the damping ratio should grow between iterations. Higher values of this argument correspond to more aggressive damping growth. The keyword argument damp_search_iters dictates the number of iterations to perform in the damping ratio search. The keyword argument damp_search_decrease dictates the amount by which the damping ratio should be decreased during the damping ratio search if the chosen damping ratio did not give convergence. The damping search is typically used to determine an initial damping value when using current boundary conditions, since the resulting nonlinear systems often require very small damping ratios for convergence.

The keyword argument dirichlet_scale is used by the fully-differentiable simulator described below as a penalty factor when imposing Dirichlet boundary conditions. The keyword argument verbose dictates the verbosity level of the damped Newton solver. This may be set to a value of verbose_noverbose, in which case no debug information is printed to the console, or verbose_print_iters, in which case the Newton solver convergence log is printed to the console. The return value p is the NewtonParams instance generated with the specified parameters.

The function newton_current! was implemented to perform a non-equilibrium simulation of a device with a single current-controlled contact. Multiple current-controlled contacts are not currently supported. The function is defined as shown below:

```
function newton_current!(
        state::NewtonState{Tz,Tv,Ti},
        params::NewtonParams{Tv,Ti},
        d::Semiconductors.Device,
        sys::VoronoiFVM.AbstractSystem{Tv},
        tf::AbstractArray{Tv}
) where {Tz,Tv,Ti}

        initialize!(state,params,d,sys)
        while state.converged==status_not_converged
                assemble_derivs!(state,params,d,sys,tf)
                newton_step_current!(state,params,d,sys,tf)
                check_convergence!(state,params)
        end

        current!(state,d,sys,tf)
        return (nothing)

end
```

The arguments state::NewtonState{Tz,Tv,Ti} and params::NewtonParams{Tv,Ti} store the state and parameters of the damped Newton solver. The argument d::Semiconductors.Device is the Device instance to be simulated. The argument sys::VoronoiFVM.AbstractSystem{Tv} is the VoronoiFVM system containing the device physics functions. This argument is used by VoronoiFVM.eval_and_assemble when assembling the system residual and Jacobian. The argument tf::AbstractArray{Tv} is the test function to be used for boundary current integration. It should be created as a Laplace test function by using a VoronoiFVM.TestFunctionFactory instance. Concentration-weighted test functions are not appropriate when setting current boundary conditions since the current must be computed each Newton iteration. The test function must be created using the same active contact as specified by params.c_active.

The function newton_current! relies on several other Semiconductors.jl functions that mutate the argument state to implement current boundary conditions. The function first calls initialize!, which prepares state for the Newton solver. Specifically, this function resets the Newton solver convergence status and damping ratio and clears the Newton update set by the previous solve. Then, for each Newton iteration, the function performs the following steps:

1. Call assemble_derivs! to assemble the system residual and Jacobian using update_res_jac!, assemble the derivative $\partial \mathbf{F}/\partial V$ and the gradient $\partial \mathbf{I}/\partial \mathbf{z}$, and update the current stored in state.

2. Call newton_step_current! to perform a single damped Newton iteration using the current residual and Jacobian.

3. Call `check_convergence!` to see if any convergence criteria have been met. Convergence is met when either the absolute or relative tolerance criteria given in (2.76) and (2.77) is met. Convergence terminates with an error when the residual increases by more than `1/params.tol_mono` between iterations or when the maximum number of iterations specified by `params.max_iters` is exceeded.

4. Repeat steps 1 through 3 until convergence.

Once the solver has converged, the function calls `current!` once to update the terminal current using the solution from the last Newton step. The function returns `nothing` since all updates to the solution and to the bias vector are performed on `state` inplace. The final solution and bias vector are stored in `state.z` and `state.bias`.

3.2.6 Modifications for AD

A fully-differentiable simulator was implemented in `Semiconductors.jl` for use in surrogate models. The differentiable simulator is compatible with the automatic differentiation package `Zygote.jl`, which uses source-to-source transformation to implement reverse-mode AD [68]. The differentiable simulator was designed to be used with the surrogate training methods implemented in `DiffEqFlux.jl` [126]. At a high level, the differentiable simulator uses similar assembly and solution routines to the solver used with current boundary conditions described in Section 3.2.5. That solver, however, is not AD-compatible, and several modifications were made to that solver to allow differentiation of the simulator. This section describes the implementation of the fully-differentiable semiconductor simulator and the modifications required to existing code to allow differentiation of the simulator.

The `NewtonState` and `NewtonParams` structures are used extensively by the differentiable simulator. The core function of the differentiable simulator is `newton!`, which solves the discretized PDE system in an AD compatible manner. The function is defined as shown below:

```julia
function newton!(
        state::NewtonState,
        params::NewtonParams,
        d::Semiconductors.Device,
        sys::VoronoiFVM.AbstractSystem
)

        initialize!(state,params,d,sys)

        while state.converged==status_not_converged
                F,J = assemble_res_jac(state,params,d,sys)
                newton_step!(F,J,state,params,d)
                check_convergence_flux!(state,params)
        end

        return (nothing)

end
```

The arguments `state::NewtonState{Tz,Tv,Ti}` and `params::NewtonParams{Tv,Ti}` store the state and parameters of the damped Newton solver. The argument `d::Semiconductors.Device` is the Device instance to be simulated. The argument sys::VoronoiFVM.AbstractSystem{Tv} is the VoronoiFVM system containing the device physics functions.

The algorithm used by the differentiable Newton solver is similar to that used by the solver described in Section 3.2.5. The key difference in this implementation is in the assembly of the residual and Jacobian. Currently, Zygote.jl does not support operations that mutate arrays. The VoronoiFVM assembly routines rely heavily on array mutations to construct the residual and Jacobian stored in sys. Consequently, new assembly routines were written to allow AD compatibility. The function assemble_res_jac was implemented to assemble the residual and Jacobian without requiring array mutations. The function is defined as shown below:

```
function assemble_res_jac(
        state::NewtonState,
        params::NewtonParams,
        d::Semiconductors.Device,
        sys::VoronoiFVM.AbstractSystem
)

        F = Zygote.Buffer(zero(state.z),false)
        J = Zygote.Buffer(zeros(num_dof(sys),num_dof(sys)),false)

        assemble_flux!(F,J,state,d,sys)
        assemble_reaction!(F,J,state,d,sys)
        assemble_bcs!(F,J,state,params,d,sys)

        F_vec = copy(F)
        J_mat = sparse(copy(J))
        return (F_vec,J_mat)

end
```

The arguments state::NewtonState, params::NewtonParams, d::Semiconductors.Device and sys::VoronoiFVM.AbstractSystem are the same arguments passed to newton!.

A first key difference between newton_current! and the differentiable newton! is the data structure used for residual and Jacobian assembly. In newton!, the residual and Jacobian are defined using Zygote.Buffer structures, which provide an AD-compatible, array-like object to hold intermediate values during assembly. The Buffer can be mutated like an array but does not support many array operations like broadcasting. A Buffer cannot be used in any computations, but its contents must be *frozen* once assembly is complete. This is accomplished by the lines F_vec = copy(F) and J_mat = sparse(copy(J)), which create a Vector with the same contents as F and a SparseMatrixCSC with the same contents as J. These values can be used in subsequent computations, and they are used to compute the Newton update each iteration. The Buffer must be re-created each Newton iteration, since its contents cannot be modified after the Buffer has been frozen.

A second key difference between newton_current! and newton! is the method used for Jacobian evaluation. Typically, VoronoiFVM.eval_and_assemble is used to evaluate and assemble the system Jacobian using ForwardDiff. However, nested AD is difficult to implement using Zygote and can negatively impact performance even when used properly. Consequently, newton! uses a manual Jacobian assembly routine that does not rely on ForwardDiff.jacobian. The Jacobian implemented in this solver was computed for the simple case where field-dependent mobility

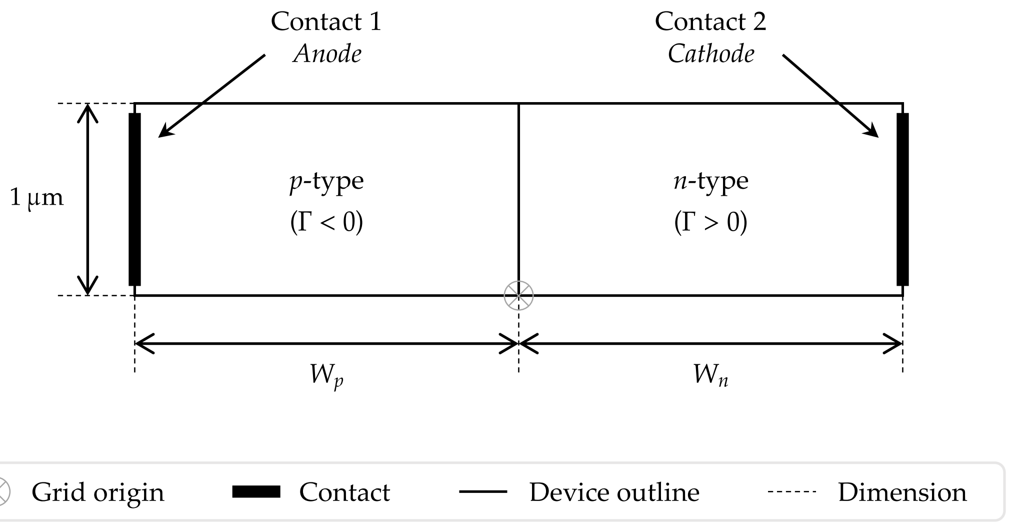


Figure 3.2: TCAD model of 1D diode generated by `diode1d`. The anode and cathode contacts span the entire height of the device.

and impact ionization are both disabled. Neither of these features are currently supported in the differentiable simulator; the sparsity patterns arising from their vector discretizations and the need to differentiate the mobility and generation terms would make the partial derivative expressions prohibitively complex. The manual Jacobian assembly is implemented by the functions `assemble_flux!`, `assemble_reaction!` and `assemble_bcs!`. The expressions used in manual Jacobian assembly are derived in Appendix A.3.

3.3 The model zoo

Several common semiconductor devices were modeled using `Semiconductors.jl`. In particular, planar diodes, bipolar transistors and MOSFETs were implemented to test the functionality of the simulator. The device models were created using the `Semiconductors.Device` structure, which holds the discretization grid and various physical parameters of the model. This section describes some of the models implemented in `Semiconductors.jl`.

3.3.1 Diodes

One-dimensional and two-dimensional diode models were implemented in `Semiconductors.jl`. The 1D diode model is shown in Figure 3.2. This model is equivalent to a 2D planar diode with a height of 1 μm, contacts that span the entire height of the device, and a doping profile that is constant along the y axis. This device is parameterized by the widths of the p-type and n-type regions, which we denote by W_p and W_n. The grid origin of this device is located at the junction between the p and n regions. The 1D diode model is generated by the function `diode1d`. The function definition is shown below:

```
function diode1d(
        wp::Float64,wn::Float64;
```

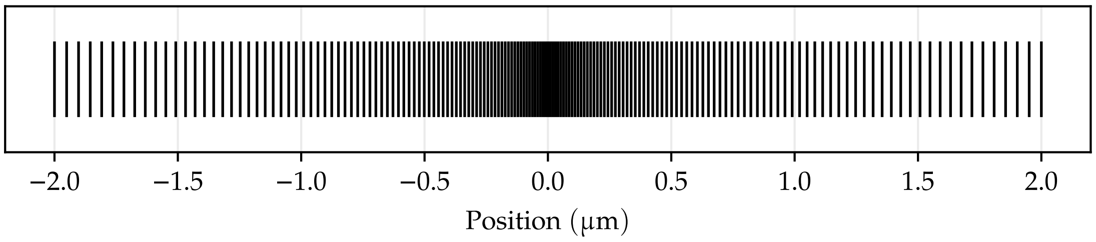


Figure 3.3: Default discretization grid for 1D diode generated by `diode1d(2.0,2.0)`, showing refinement about $x = 0$. The grid has 163 nodes and 162 cells (segments).

```
        ha=0.05,hb=0.01,hc=0.05,
        na=1e3,nd=1e3,
        doping="abrupt"
)

        # Generate grid and define model parameters
        ...

        return (d)

end
```

The arguments `wp::Float64` and `wn::Float64` give the values of W_p and W_n, respectively. The keyword arguments `ha`, `hb` and `hc` dictate the spacing between grid points at the anode contact, at the junction and at the cathode contact, respectively. The grid will be described in detail below. The keyword arguments `na` and `nd` dictate the acceptor and donor doping concentrations N_A and N_D, respectively. When a non-abrupt doping profile is used, these values give the doping at the anode and cathode, respectively. The keyword argument `doping` dictates which doping profile will be used in the model. The doping profile will be described in detail below. The return value `d` is the `Device` instance generated with the specified parameters.

The grid used by the 1D diode model is shown in Figure 3.3. This grid is generated using `ExtendableGrids.geomspace`. This function creates an array of values such that the ratio between each pair of adjacent points is a multiple q of the ratio between the previous pair of adjacent points, where $q \in (0, 1]$. This property allows for local grid refinement about the semiconductor junction at $x = 0$. This refinement is often critical to simulation accuracy, since the potential and carrier concentrations change most abruptly near the junction. The function `ExtendableGrids.geomspace` is called twice to generate two arrays: one containing the points from $x = -W_p$ and $x = 0$, and once containing the points from $x = 0$ to $x = W_n$. The two arrays are spliced together after removing the duplicate value $x = 0$, which appears in both arrays.

The doping profile used by the 1D diode model is determined using the `doping` keyword argument. This argument allows the user to select one of three profiles: abrupt, smooth or linear. The abrupt doping profile is specified by the default value `doping="abrupt"`. This option defines

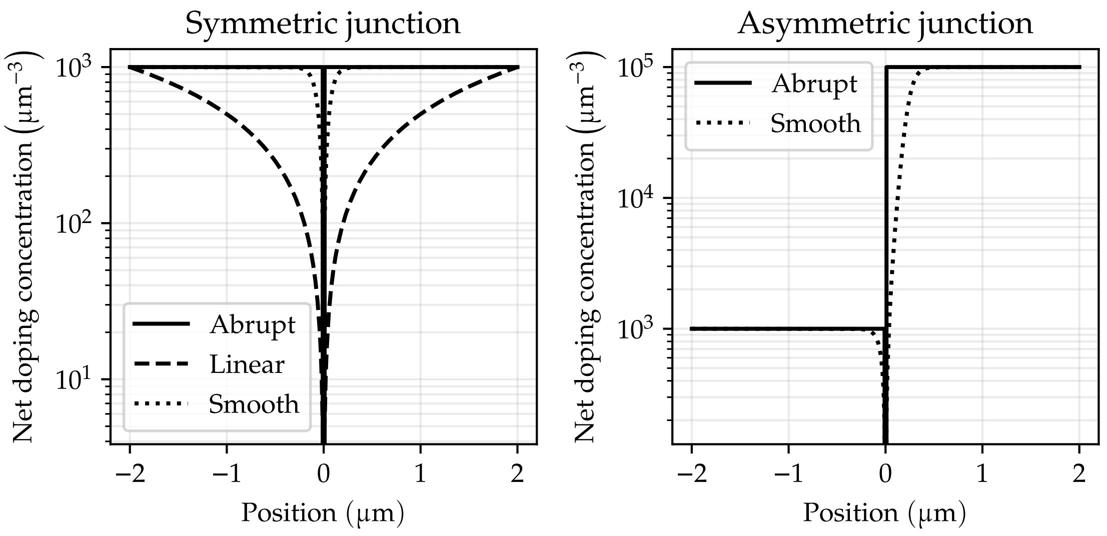


Figure 3.4: Doping profiles generated by `diode1d(2.0,2.0)`, for a symmetric junction (left) and an asymmetric junction (right).

a piecewise doping:

$$\Gamma(x) = \begin{cases} -N_A, & x \in [-W_p, 0) \\ 0, & x = 0 \\ N_D, & x \in (0, W_n] \end{cases} . \tag{3.9}$$

The smooth doping profile is specified by the value `doping="smooth"`. This profile uses a hyperbolic tangent approximation to the abrupt doping profile, which can help convergence when N_A and N_D are separated by several orders of magnitude. This doping profile is given below:

$$\Gamma(x) = \frac{N_D + N_A}{2}\left(\tanh\left(kx - \tanh^{-1}(r)\right) + r\right), \quad \text{where} \quad r = \frac{N_D - N_A}{N_D + N_A}, \tag{3.10}$$

and where k is a parameter that dictates the steepness of the doping profile. The default value implemented in `diode1d` is $k = 10$, which works well for the typical doping concentrations $N_A = N_D = 1 \times 10^3\ \mu\text{m}^{-3}$. With this profile, the doping is defined such that $\Gamma \to -N_A$ as $x \to -\infty$ and $\Gamma \to N_D$ as $x \to \infty$. Typically, this means $\Gamma(-W_p) \approx -N_A$ and $\Gamma(W_n) \approx N_D$ if k is sufficiently large.

The linear doping profile is specified by the value `doping="lin"`. This profile defines a linearly-graded junction where $\Gamma(-W_p) = -N_A$ and $\Gamma(W_n) = N_D$. This doping profile is given below:

$$\Gamma(x) = \frac{N_D(x + W_p) + N_A(x - W_n)}{W_p + W_n}. \tag{3.11}$$

Due to the constraints on $\Gamma(-W_p)$ and $\Gamma(W_n)$, the PN junction does not necessarily lie at $x = 0$ when this profile is used, except in the symmetric case where $N_A = N_D$. The doping profiles given by (3.9) through (3.11) are shown in Figure 3.4. All profiles shown in Figure 3.4 were generated using `diode1d(2.0,2.0)`, which sets $W_p = W_n = 2\ \mu\text{m}$ and uses the default doping concentrations $N_A = N_D = 1 \times 10^3\ \mu\text{m}^{-3}$. The linear doping profile is not shown for the asymmetric junction since it results in a PN junction at $x \neq 0$.

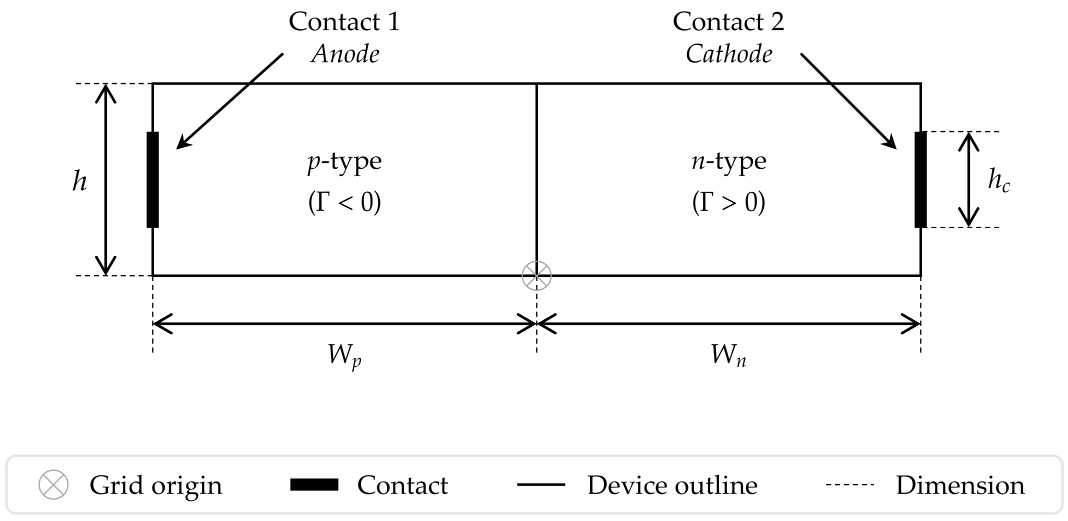


Figure 3.5: TCAD model of 2D diode implemented by `diode2d`. The contact heights are equal and are determined by the parameter h_c.

The 2D diode model is shown in Figure 3.5. This device is parameterized by the widths of the p-type and n-type regions, which we denote by W_p and W_n, and the height of the device and of the contacts, which we denote by h and h_c, respectively. The grid origin of this device is located at the junction between the p and n regions. The 2D diode model is generated by the function `diode2d`. The function definition is shown below:

```
function diode2d(
        wn:Float64,wp:Float64,
        h:Float64,hc:Float64;
        na=1e3,nd=1e3,
        lin=false,
        max_vol=0.8,
        refine_param=1e-4,
        yplane=nothing,
        bgmesh=false,
        x_bg=1.0,y_bg=1.0,
        max_vol_bg=2.0,
        dx1=0.1,dx2=0.05,dx3=0.1,
        ny=5,rectgrid=false
)

        # Generate grid and define model parameters
        ...

        return (d)

end
```

The arguments `wp::Float64` and `wn::Float64` give the values of W_p and W_n, respectively. The arguments `h::Float64` and `hc::Float64` give the values of h and h_c, respectively. The keyword arguments `na` and `nd` dictate the acceptor and donor doping concentrations N_A and N_D, respectively. When a linear doping profile is used, these values give the doping at the anode and cathode, respectively. The keyword argument `lin` dictates whether to use a linear doping profile. The linear doping profile for the 2D diode is the same as that used in the 1D diode, which is given in (3.11). The keyword arguments `max_vol` and `refine_param` control the grid refinement if a triangular grid is used. The keyword argument `yplane` adds an optional cut plane to the grid to allow cross-sectional plotting in 1D when `rectgrid==false`.

The keyword argument `bgmesh` adds an optional background mesh to the grid to model interactions between the device and its environment. The keyword arguments `x_bg` and `y_bg` dictate the thickness of the background mesh in the x and y directions, respectively, if `bgmesh==true`. The keyword argument `max_vol_bg` controls the refinement of the background mesh. The keyword arguments `dx1`, `dx2` and `dx3` set the spacing between grid points at the anode contact, at the junction and at the cathode contact, respectively, if a rectangular grid is used. The keyword argument `ny` dictates the number of grid points along the y direction if a rectangular grid is used. The keyword argument `rectgrid` dictates whether a rectangular grid should be used. If `rectgrid==false`, a triangular grid is used.

The grid used by the 2D diode model can be either triangular or rectangular, and it may have a background mesh or no background mesh. The default triangular grid is shown without and with a background mesh in Figures 3.6 and 3.7, respectively. Local refinement is visible in both cases about $x = 0$. This is accomplished by using the `unsuitable` callback in a `SimplexGridFactory` instance. This callback accepts the coordinates of each point in a triangle and the area of a triangle as arguments and returns a `Bool`. The callback returns `true` if the given triangle requires further refinement, or `false` if the triangle should be accepted by the mesh generator. The callback used by `diode2d` sets a constraint on the square of the distance between $x = 0$ and the barycenter of each triangle. The amount of local refinement is dictated by the keyword argument `refine_param`. The default value is `refine_param=1e-4`; smaller values produce more refinement about the junction and larger values produce less refinement.

3.3.2 Bipolar transistors

A planar NPN transistor model was implemented in `Semiconductors.jl`. The transistor model is shown in Figure 3.8. This model is a 2D cross-section of the active portion of a vertical NPN transistor. The device has emitter, base and collector contacts. The emitter and collector contacts are centered vertically along the device, and the base contact is centered in the base region. This device is parameterized by the widths of the emitter, base and collector regions, which we denote by W_e, W_b and W_c; the height of the device h; the heights of the emitter and collector contacts, which we denote by h_{ce} and h_{cc}; and the width of the base contact W_{cb}. The grid origin of this device is located at the bottom-left corner of the emitter region. The planar NPN model is generated by the function `npn1`. The function definition is shown below:

```julia
function npn1(
        we::Float64,wb::Float64,wc::Float64,
        h::Float64,hce::Float64,hcc::Float64,wcb::Float64;
        ne=1e6,nb=1e4,nc=1e3,
        doping=nothing,k1=2.0,k2=3.0,
        max_vol=2.0,plot_doping=false,
        refine_param=0.3,
```

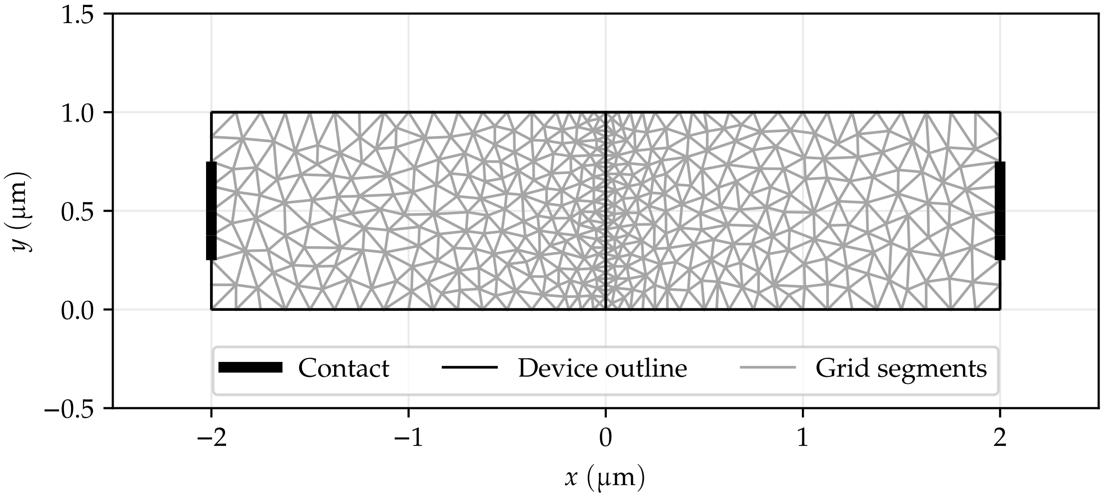


Figure 3.6: Default discretization grid for 2D diode generated by `diode2d(2.0,2.0,1.0,0.5)`, showing refinement about $x = 0$. The grid has 459 nodes and 825 cells (triangles).

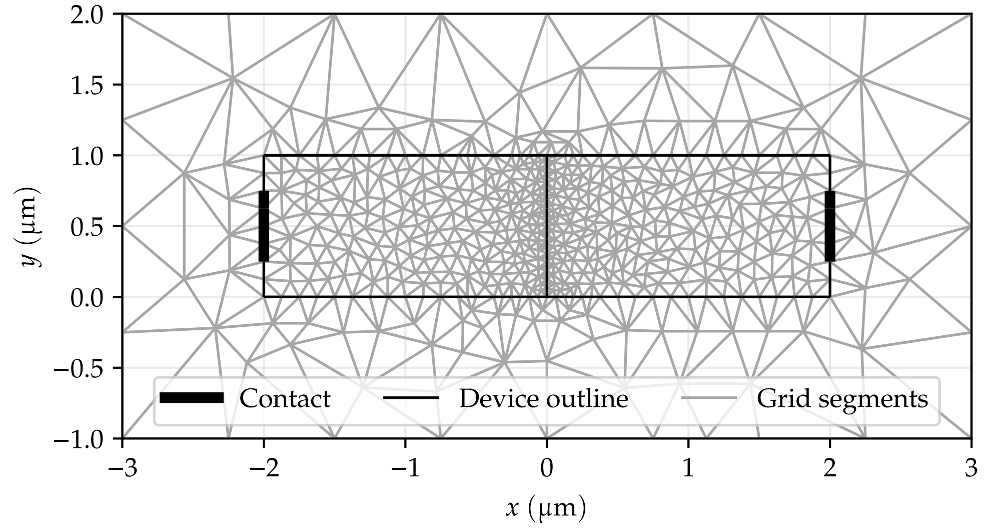


Figure 3.7: Discretization grid for 2D diode generated by `diode2d(2.0,2.0,1.0,0.5,bgmesh=true)`. The grid has 561 nodes and 1095 cells (triangles).

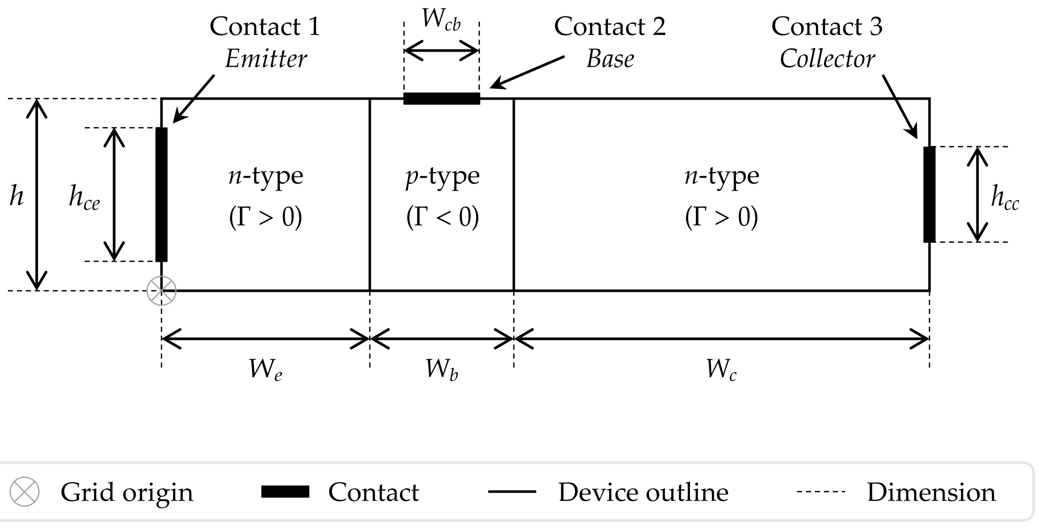


Figure 3.8: TCAD model of planar NPN transistor implemented by `npn1`. The emitter and collector contacts are centered vertically along the device, and the base contact is centered in the base region.

```
            yplane=nothing,
            dx1=1.0,dx2=0.1,dx3=0.1,dx4=2.0,
            ny=6,rectgrid=false
)

            # Generate grid and define model parameters
            ...

            return (d)

end
```

The arguments `we::Float64`, `wb::Float64` and `wc::Float64` give the values of W_e, W_b and W_c, respectively. The arguments `h::Float64`, `hce::Float64` and `hcc::Float64` give the values of h, h_{ce} and h_{cc}, respectively. The argument `wcb::Float64` gives the value of W_{cb}. The keyword arguments `ne`, `nb` and `nc` dictate the doping concentrations in the emitter, base and collector regions, which we denote as N_E, N_B and N_C, respectively. If `ne`, `nb` and `nc` are positive, the doping profile is that of an NPN transistor. If `ne`, `nb` and `nc` are negative, the doping profile is that of a PNP transistor.

The keyword argument `doping` dictates which doping profile will be used in the model. The doping profile will be described in detail below. The keyword arguments `k1` and `k2` dictate the steepness of the smooth doping approximation if `doping=="smooth"`. The parameter `max_vol` dictates the maximum area of a triangle in the grid when `rectgrid==false`. The keyword argument `plot_doping` causes `npn1` to plot the specified doping profile along the x axis if `plot_doping==true`. The keyword argument `refine_param` controls the grid refinement if a triangular grid is used. The keyword argument `yplane` adds an optional cut plane to the grid to allow cross-sectional plotting in 1D when `rectgrid==false`.

The keyword arguments `dx1`, `dx2`, `dx3` and `dx4` set the spacing between grid points at the emitter contact, at the base-emitter junction, at the base-collector junction and at the collector contact, respectively, if a rectangular grid is used. The keyword argument `ny` dictates the number of grid points along the y direction if a rectangular grid is used. The keyword argument `rectgrid` dictates whether a rectangular grid should be used. If `rectgrid==false`, a triangular grid is used.

The grid used by the planar NPN model can be either triangular or rectangular. The default triangular and rectangular grids are shown in Figures 3.9 and 3.10, respectively. Local refinement is visible about the two junctions in both cases. When the triangular grid is used, this refinement is accomplished using a similar `unsuitable` callback to `SimplexGridFactory` as is used with the 2D diode model. When the rectangular grid is used, this refinement is accomplished using `ExtendableGrids.geomspace` as is used with the 1D diode model. A high level of refinement about the junctions is typically necessary for convergence, particularly when the emitter and base doping values are separated by several orders of magnitude.

The doping profile used by the planar NPN model is determined using the `doping` keyword argument. This argument allows the user to select one of four profiles: abrupt, smooth, exponentially-graded base and linearly-graded base. The abrupt doping profile is selected by the default value `doping=nothing`. This option defines a piecewise doping:

$$\Gamma(x) = \begin{cases} N_E, & x \in [0, W_e) \\ -N_B, & x \in (W_e, W_e + W_b) \\ N_C, & x \in (W_e + W_b, W_e + W_b + W_c] \\ 0, & \text{otherwise} \end{cases} . \tag{3.12}$$

The smooth doping profile is specified by the value `doping="smooth"`. This profile uses a hyperbolic tangent approximation to the abrupt doping profile, which can help convergence when N_E and N_B are separated by several orders of magnitude. This doping profile is given below:

$$\begin{aligned}\Gamma(x) = \frac{1}{2}\left(N_E - (N_E + N_B)\tanh\left(k_1 (x - W_e) + \tanh^{-1}\left(\frac{N_E - N_B}{N_E + N_B}\right)\right)\right) + \\ \frac{1}{2}\left(N_C + (N_C + N_B)\tanh\left(k_2 (x - (W_e + W_b)) - \tanh^{-1}\left(\frac{N_C - N_B}{N_C + N_B}\right)\right)\right),\end{aligned} \tag{3.13}$$

where k_1 and k_2 are parameters that dictate the steepness of the doping profile at the base-emitter junction and at the base-collector junction, respectively. The default value implemented in `npn1` is $k_1 = k_2 = 10$, which works well for the typical doping concentrations $N_E = 1 \times 10^6\ \mu\text{m}^{-3}$, $N_B = 1 \times 10^4\ \mu\text{m}^{-3}$ and $N_C = 1 \times 10^3\ \mu\text{m}^{-3}$.

The exponentially-graded base doping profile is specified by the value `doping="expbase"`. This profile defines a constant doping in the emitter and collector regions and a doping that varies exponentially from $-N_B$ to $-N_C$ in the base region. This doping profile is given below:

$$\Gamma(x) = \begin{cases} N_E, & x \in [0, W_e) \\ -N_B^{1-\frac{x-W_e}{W_b}} N_C^{\frac{x-W_e}{W_b}}, & x \in (W_e, W_e + W_b) \\ N_C, & x \in (W_e + W_b, W_e + W_b + W_c] \\ 0, & \text{otherwise} \end{cases} . \tag{3.14}$$

The linearly-graded base doping profile is specified by the value `doping="linbase"`. This profile defines a constant doping in the emitter and collector regions and a doping that varies linearly

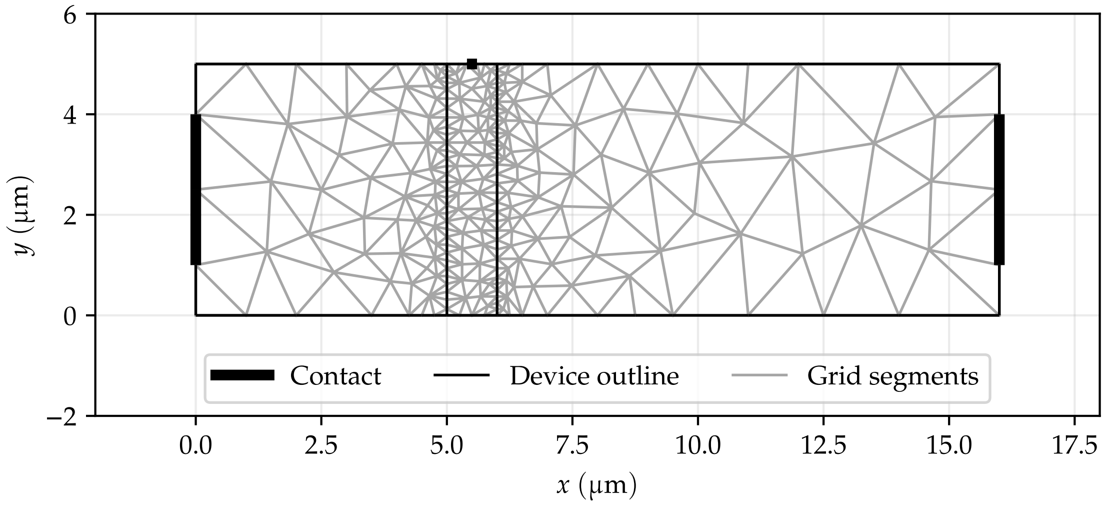


Figure 3.9: Default discretization grid for planar NPN transistor generated by `npn1(5,1,10,5,3,3,0.2)`, showing refinement about base-emitter and base-collector junctions. The grid has 257 nodes and 461 cells (triangles).

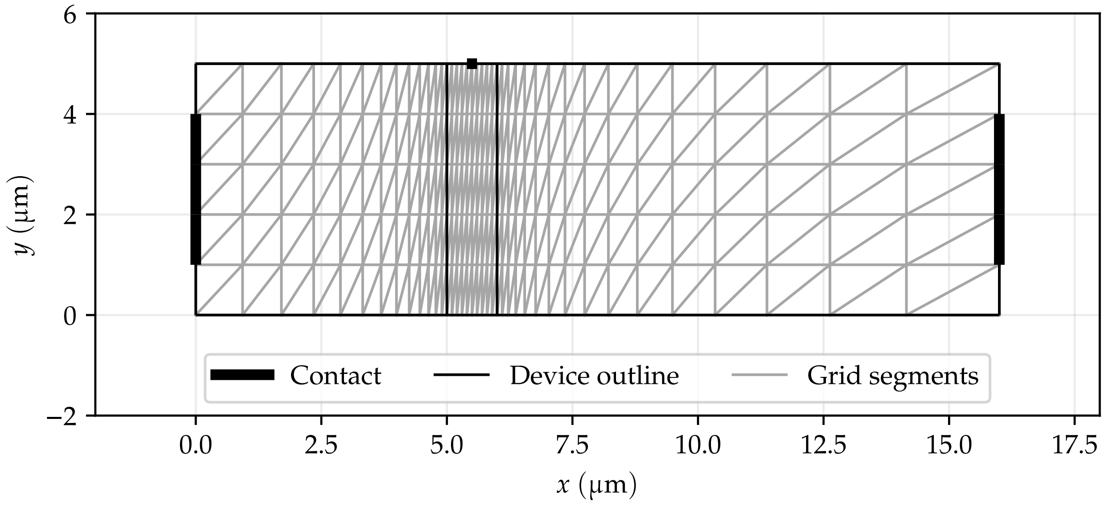


Figure 3.10: Discretization grid for planar NPN transistor generated by `npn1(5,1,10,5,3,3,0.2,rectgrid=true)`. The grid has 240 nodes and 390 cells (triangles).

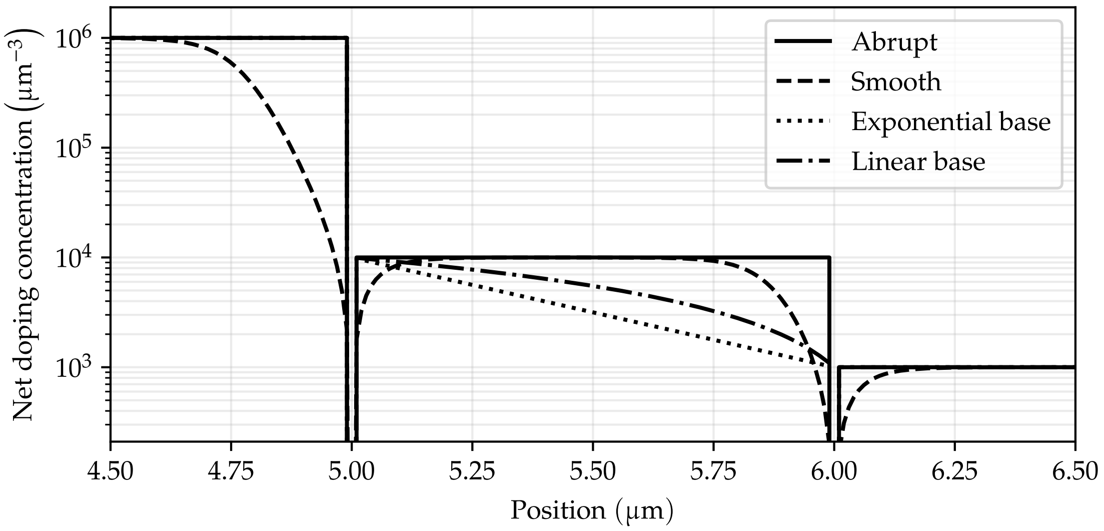


Figure 3.11: Doping profiles generated by `npn1(5,1,10,5,3,3,0.2)`, plotted along the x axis only. The region where $x \in (5\,\mu\text{m}, 6\,\mu\text{m})$ is the base region.

from $-N_B$ to $-N_C$ in the base region. This doping profile is given below:

$$\Gamma(x) = \begin{cases} N_E, & x \in [0, W_e) \\ -N_B\left(1 - \dfrac{x - W_e}{W_b}\right) - N_C\left(\dfrac{x - W_e}{W_b}\right), & x \in (W_e, W_e + W_b) \\ N_C, & x \in (W_e + W_b, W_e + W_b + W_c] \\ 0, & \text{otherwise} \end{cases} . \tag{3.15}$$

The doping profiles given by (3.12) through (3.15) are shown in Figure 3.11. All profiles shown in Figure 3.11 were generated using `npn1(5,1,10,5,3,3,0.2)`, which sets $W_e = 5\,\mu\text{m}$, $W_b = 1\,\mu\text{m}$, $W_c = 10\,\mu\text{m}$, $h = 5\,\mu\text{m}$, $h_{ce} = h_{cc} = 3\,\mu\text{m}$ and $W_b = 200\,\text{nm}$. The default doping concentrations $N_E = 1 \times 10^6\,\mu\text{m}^{-3}$, $N_B = 1 \times 10^4\,\mu\text{m}^{-3}$ and $N_C = 1 \times 10^3\,\mu\text{m}^{-3}$ were used for the profiles shown in Figure 3.11.

3.3.3 MOSFETs

A planar MOSFET model was implemented in `Semiconductors.jl`. The transistor model is shown in Figure 3.12. This model is a 2D cross-section of a typical bulk MOSFET, such as those described in [87]. The device has gate, source, drain and bulk contacts, which are assigned to boundary regions 1 through 4, respectively. The size and position of all contacts except the bulk contact are configurable by the user. The bulk contact spans the entire bottom face of the device. This device is parameterized by the quantities labeled in Figure 3.12. The grid origin of this device is located at the bottom-left corner of the bulk region. The planar MOSFET model is generated by the function `mos1`. The function definition is shown below:

```
function mos1(
        l::Float64,ls::Float64,h::Float64,hs::Float64,
        tox::Float64,c1::Float64,c2::Float64;
```

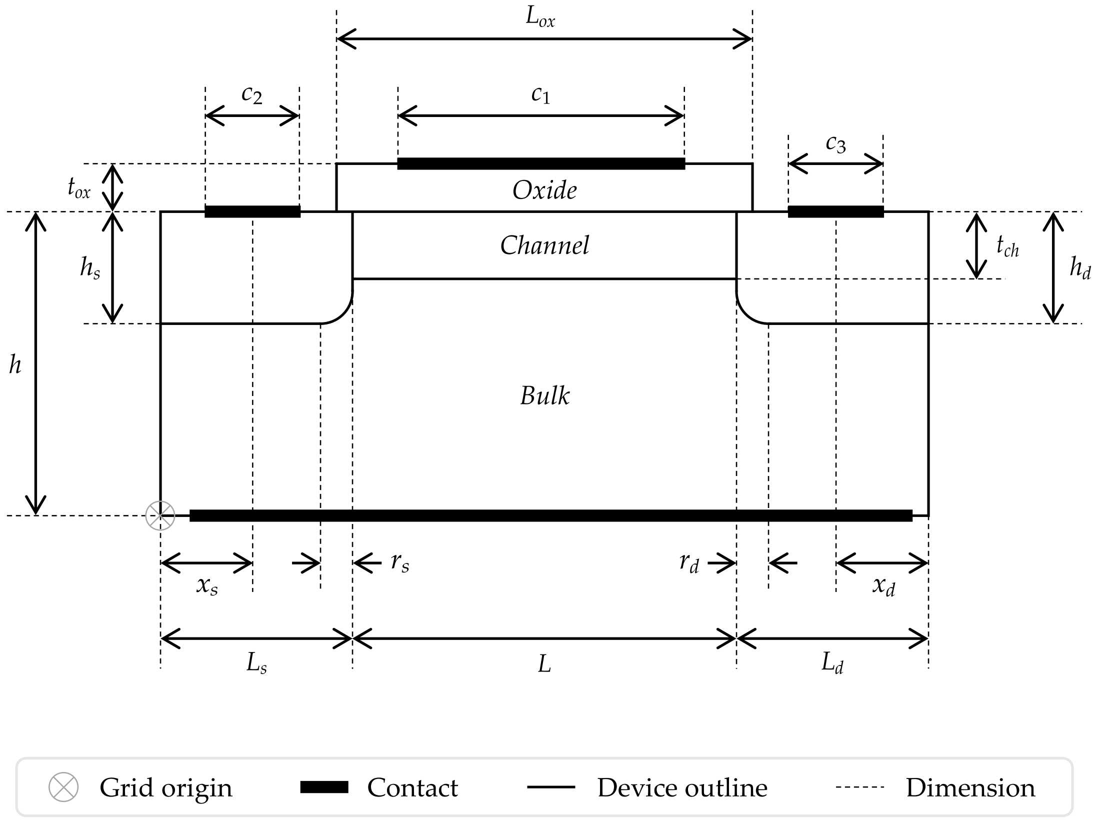


Figure 3.12: TCAD model of planar MOSFET implemented by `mos1`. The contacts along the top of the device are the source, gate and drain, respectively. The bottom contact is the bulk contact.

```
        N_arc=10,max_vol=1e-4,
        ld=nothing,hd=nothing,rs=nothing,rd=nothing,
        xs=nothing,xd=nothing,tch=nothing,c3=nothing,lox=nothing,
        er_si=nothing,er_ox=nothing,
        ni_si=nothing,tn_si=nothing,tp_si=nothing,ew_ox=nothing,
        n_s=nothing,n_d=nothing,n_ch=nothing,n_b=nothing,
        max_vol_ox=nothing,max_vol_s=nothing,max_vol_d=nothing,
        max_vol_b=nothing,max_vol_ch=nothing,refine_param=nothing
)

        # Generate grid and define model parameters
        ...

        return (d)

end
```

The first seven arguments give the values of L, L_s, h, h_s, t_{ox}, c_1 and c_2, respectively. The keyword argument `N_arc` dictates the number of straight line segments in the arcs in the boundary of the source and drain diffusion. The keyword argument `max_vol` dictates the maximum area of a triangle in the grid. The next nine keyword arguments give the values of L_d, H_d, r_s, r_d, x_s, x_d,

t_{ch}, c_3 and L_{ox}, respectively. The keywords `er_si` and `er_ox` give the relative permittivities of the silicon and oxide regions, respectively. The keyword argument `ni_si` gives the intrinsic carrier concentration of the silicon regions. The keyword arguments `tn_si` and `tp_si` give the electron and hole lifetimes in the silicon regions, respectively. The keyword argument `ew_ox` gives the gate-oxide workfunction potential.

The keyword arguments `n_s`, `n_d`, `n_ch` and `n_b` give the source, drain, channel and bulk doping concentrations, which we denote as N_s, N_d, N_{ch} and N_b, respectively. The keyword arguments `max_vol_ox`, `max_vol_s`, `max_vol_d`, `max_vol_b` and `max_vol_ch`, if set, override the maximum volume constraint given by `max_vol` for the oxide, source, drain, bulk and channel regions, respectively. The keyword argument `refine_param` controls the grid refinement near the channel-oxide interface and around the source and drain diffusion. Smaller values produce more refinement about the interfaces and larger values produce less refinement.

Default values for the keyword arguments are set inside the definition of `mos1`. We assume typical values for the physical parameters. The default permittivities are `er_si=11.7` and `er_ox=3.9`. The carrier lifetimes are `tn_si=1e-7` and `tp_si=1e-7`. The gate-oxide workfunction potential is `ew=5.5e-1`, which is the value $E_w/q = 550\,\text{mV}$ given in [144]. The doping concentrations are $N_s = N_d = 2 \times 10^8\,\mu\text{m}^{-3}$, $N_{ch} = -1 \times 10^6\,\mu\text{m}^{-3}$ and $N_b = -5 \times 10^4\,\mu\text{m}^{-3}$. These concentrations define an *n*-channel MOSFET; their signs can be reversed to define a *p*-channel MOSFET instead. Only an abrupt doping profile is currently implemented due to the complexity of the model geometry.

The grid used by the planar MOSFET model is shown in Figure 3.13. A zoomed inset showing the gate oxide region is shown in Figure 3.14. This grid was generated using a 100 nm channel, a 2 nm oxide thickness and a 200 nm substrate thickness. Local refinement is visible near the top of the channel and around the source and drain diffusion. This refinement is accomplished using a similar `unsuitable` callback to `SimplexGridFactory` as is used with the 2D diode model. A high level of refinement is typically necessary for convergence since the default doping concentrations vary from $5 \times 10^4\,\mu\text{m}^{-3}$ to $2 \times 10^8\,\mu\text{m}^{-3}$.

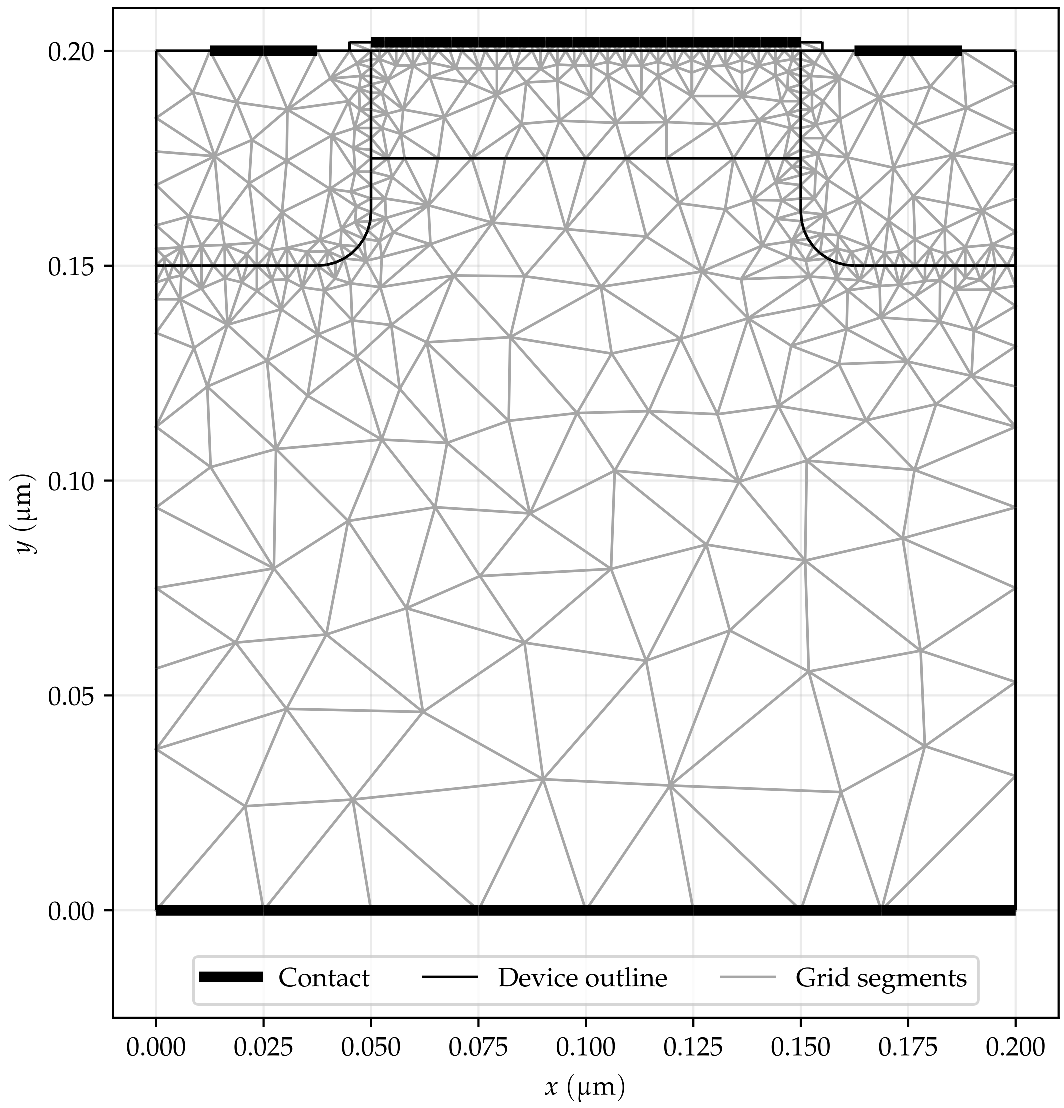


Figure 3.13: Default discretization grid for planar MOSFET generated by `mos1(0.1,0.05,0.2,0.05,0.002,0.1,0.025)`, showing refinement about source and drain diffusion. The grid has 478 nodes and 870 cells (triangles).

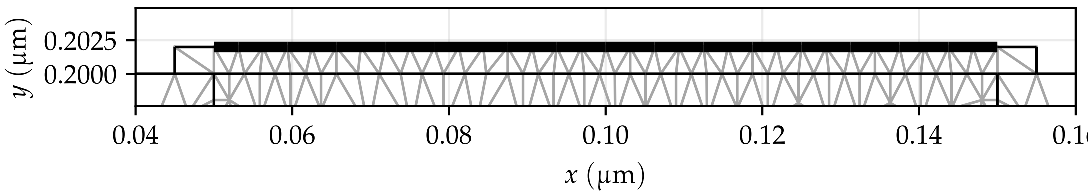


Figure 3.14: Zoomed plot of gate oxide region in default MOSFET grid.

Chapter 4

Results

4.1 Numerical analysis

This section presents experimental results on the implementation of the Bernoulli function $B(x)$, the interpolation function $Q(x)$ and the derivative of the Bernoulli function $B'(x)$. These functions are defined in (2.49), (2.43) and (A.10), respectively.

4.1.1 Error comparison

The functions $B(x)$, $Q(x)$ and $B'(x)$ are difficult to evaluate numerically, particularly when x is near zero. There are two main sources of error in the implementation of these functions. The first is *rounding error*, which typically results from the subtraction of two nearly equal values. The result of such a computation must be rounded to the nearest floating point value, which may be quite far from the exact result. The second is *truncation error*, which is introduced by an approximation to a function that becomes inaccurate outside of a certain range. For example, if a Taylor series expansion is used to approximate $B(x)$ at $x = x_0$, the series will only be close to the exact value of $B(x)$ in a small neighborhood of x_0.

Figures 4.1, 4.2 and 4.3 compare these sources of error for $B(x)$, $Q(x)$ and $B'(x)$, respectively. The "Direct" error in these plots shows the relative error between a direct computation of the functions using `Float64` values and a direct computation using `BigFloat` values. Specifically, we use the functions shown below:

```
q(x) = 1/x - 1/(exp(x)-1)
b(x) = x/(exp(x)-1)
db(x) = 1/(exp(x)-1) - x*exp(x)/(exp(x)-1)^2
```

The direct error was computed by evaluating `q(x)`, `b(x)` and `db(x)` at 1001 evenly spaced points between $x = 1 \times 10^{-2}$ and $x = 1 \times 10^{2}$. The direct error in all three functions shows an asymptote for small x due to cancellation in the denominator. For $B(x)$, this lower asymptote begins around $x = \pm 0.3$; for $Q(x)$, the lower asymptote begins around $x = 1$ and $x = -2$; for $B'(x)$, the lower asymptote begins around $x = \pm 2$. The direct error in $B(x)$ and $B'(x)$ also shows an asymptote for large x due to floating-point overflow in the denominator. For $B(x)$, this upper asymptote begins around $x = 3$; for $B'(x)$, the upper asymptote begins around $x = 4$.

The "Series" error in Figures 4.1, 4.2 and 4.3 shows the truncation error of a Taylor series

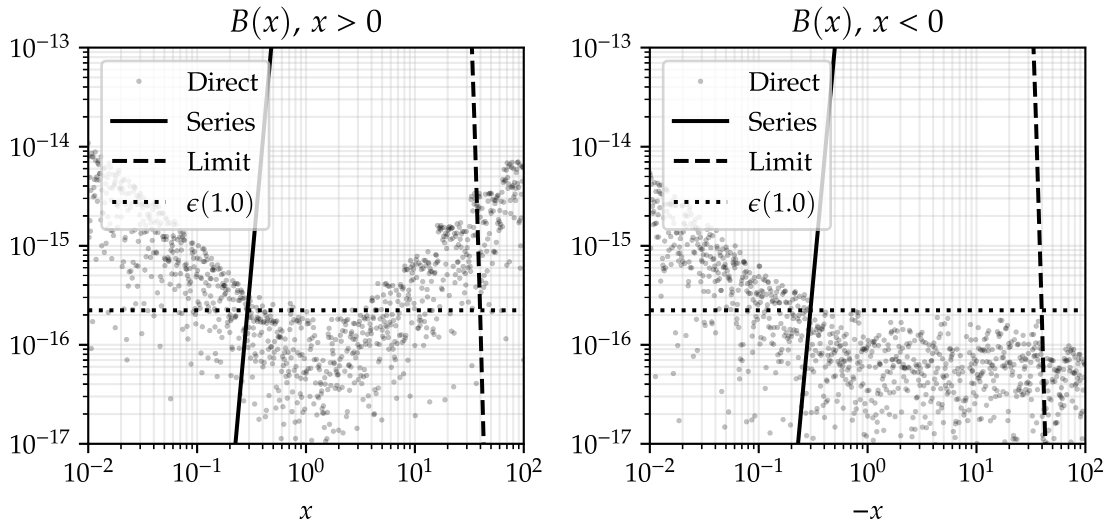


Figure 4.1: Relative error in direct, series and limit approximations of $B(x)$. The "Limit" error gives $B(x)$ for $x > 0$ and $x + B(x)$ for $x < 0$. The line labeled $\epsilon(1.0)$ is the `Float64` machine epsilon $\epsilon(1.0) = 2^{-52} \approx 2.220 \times 10^{-16}$.

expansion at $x = 0$. Specifically, we use the expansions

$$B(x) = 1 - \frac{x}{2} + \frac{x^2}{12} - \frac{x^4}{720} + \frac{x^6}{30\,240} - \frac{x^8}{1\,209\,600} + \frac{x^{10}}{47\,900\,160} + O\left(x^{12}\right), \tag{4.1}$$

$$Q(x) = \frac{1}{2} - \frac{x}{12} + \frac{x^3}{720} - \frac{x^5}{30\,240} + \frac{x^7}{1\,209\,600} - \frac{x^9}{47\,900\,160} + \frac{691x^{11}}{1\,307\,674\,368\,000} - \frac{x^{13}}{74\,724\,249\,600} + O\left(x^{15}\right), \tag{4.2}$$

and

$$B'(x) = -\frac{1}{2} + \frac{x}{6} - \frac{x^3}{180} + \frac{x^5}{5040} - \frac{x^7}{151\,200} + \frac{x^9}{4\,790\,016} - \frac{691x^{11}}{108\,972\,864\,000} + \frac{x^{13}}{5\,337\,446\,400} - \frac{3617x^{15}}{666\,913\,927\,680\,000} + \frac{43\,867x^{17}}{283\,838\,567\,620\,608\,000} - \frac{174\,611x^{19}}{40\,142\,883\,134\,914\,560\,000} + O\left(x^{21}\right). \tag{4.3}$$

The order of the expansions given in (4.1) through (4.3) were chosen to give an overall implementation error on the order of 1×10^{-15}.

The "Limit" error in Figures 4.1, 4.2 and 4.3 shows the absolute error between $B(x)$, $Q(x)$ and $B'(x)$ and their limiting values as $x \to \infty$ and $x \to -\infty$. For $B(x)$, the limiting values are $B(x) \to 0$ as $x \to \infty$ and $B(x) \to -x$ as $x \to -\infty$. For $Q(x)$, the limiting values are $Q(x) \to 1/x$ as $x \to \infty$ and $Q(x) \to 1 + 1/x$ as $x \to -\infty$.

4.1.2 Optimal threshold search

Figures 4.1, 4.2 and 4.3 show that different approximations to $B(x)$, $Q(x)$ and $B'(x)$ have different regions of accuracy. When $|x|$ is small, the Taylor expansions give the lowest error. When $|x|$

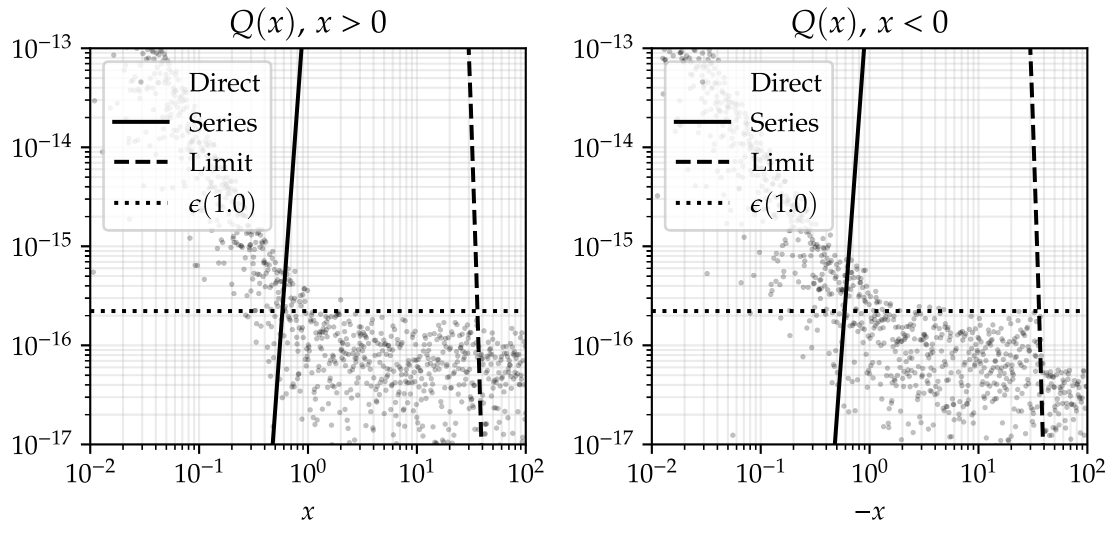


Figure 4.2: Relative error in direct, series and limit approximations of $Q(x)$. The "Limit" error gives $1/x - Q(x)$ for $x > 0$ and $Q(x) - 1/x - 1$ for $x < 0$. The line labeled $\epsilon(1.0)$ is the `Float64` machine epsilon $\epsilon(1.0) = 2^{-52} \approx 2.220 \times 10^{-16}$.

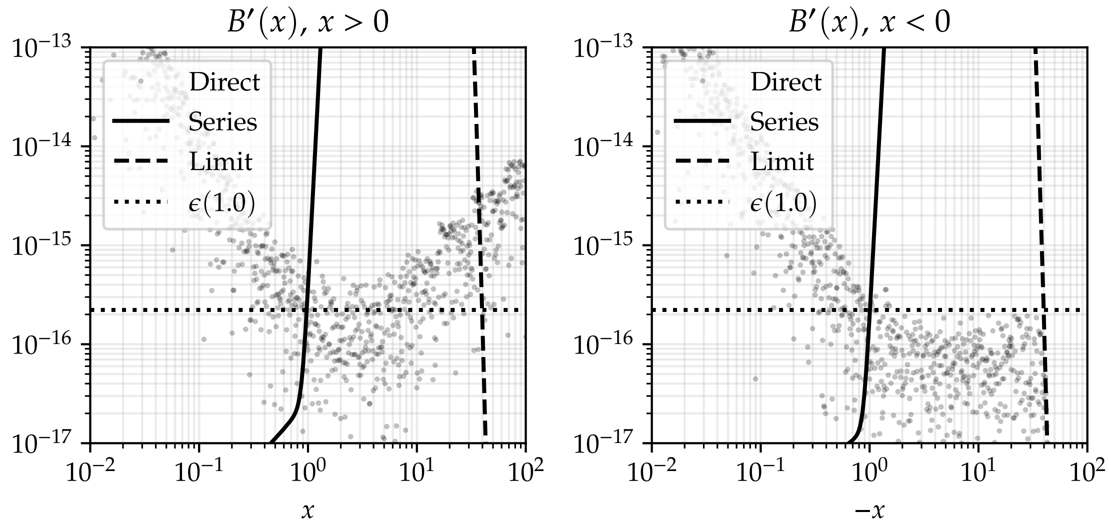


Figure 4.3: Relative error in direct, series and limit approximations of $B'(x)$. The "Limit" error gives $-B'(x)$ for $x > 0$ and $1 + B'(x)$ for $x < 0$. The line labeled $\epsilon(1.0)$ is the `Float64` machine epsilon $\epsilon(1.0) = 2^{-52} \approx 2.220 \times 10^{-16}$.

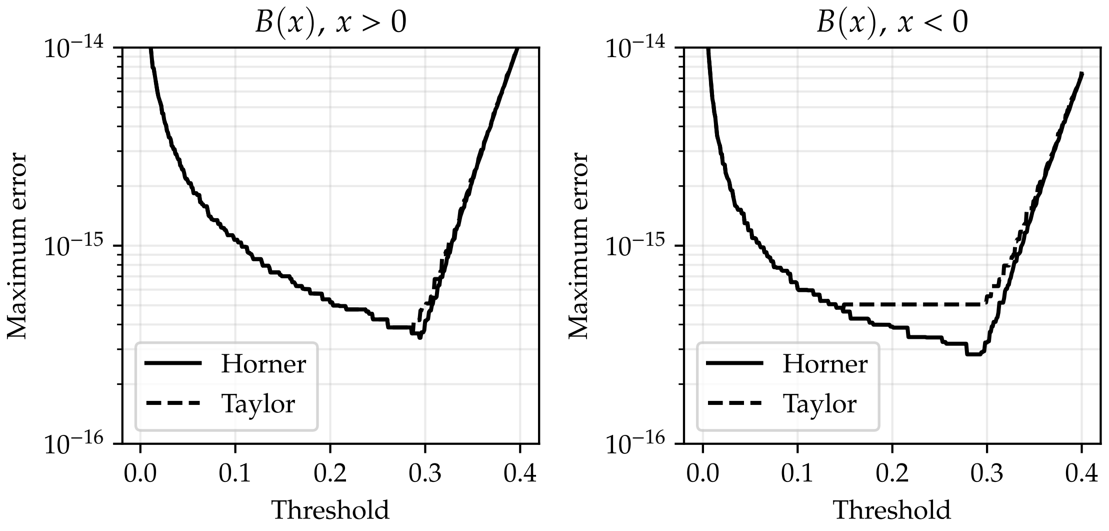


Figure 4.4: Optimal small-value threshold search for $B(x)$. For positive x, a minimum error of roughly 3.4×10^{-16} was achieved using a small-value threshold of 0.295; for negative x, a minimum error of roughly 2.8×10^{-16} was achieved using a small-value threshold of 0.279.

is large, the limiting values give the lowest error. For intermediate values of x, the functions may be evaluated directly. This property suggests that an adaptive algorithm could be used to implement these functions, where the function implementation is determined by the absolute value of x. To implement such an algorithm, two thresholds are required: A *small-value* threshold, which determines the value of $|x|$ below which Taylor expansions will be used, and a *large-value* threshold, which determines the value of $|x|$ above which the limiting values will be used.

We perform a simple search to determine the optimal small-value threshold for $B(x)$, $Q(x)$ and $B'(x)$. The optimal small-value threshold minimizes the maximum error of the approximation for all values of x. For each function, a list of candidate threshold values was generated with a step of 1×10^{-3} between each pair of values. At each threshold value, $10^5 + 1$ logarithmically spaced values between 1×10^{-16} and 2 were evaluated using an approximation to $B(x)$, $Q(x)$ and $B'(x)$ that switched between direct evaluation and a series expansion at the threshold value. The maximum relative error among the $10^5 + 1$ logarithmically spaced points was computed, and the threshold giving the lowest maximum error was chosen as the optimal threshold. The results of this search are shown in Figures 4.4, 4.5 and 4.6.

In Figures 4.4, 4.5 and 4.6, the curves labeled "Taylor" were computed using a direct evaluation of the Taylor series approximations for $B(x)$, $Q(x)$ and $B'(x)$ with the polynomial expressions in (4.1), (4.2) and (4.3). The curves labeled "Horner" were computed using Horner's method for the Taylor series approximations [15]. This method minimizes the number of multiplications required to evaluate the series approximations and can reduce numerical error when high-order approximations are necessary.

From Figures 4.4, 4.5 and 4.6, we observe that optimal thresholds appear to exist for all three functions. An increase in the maximum error is visible for smaller thresholds in all three functions. For such thresholds, the cancellation error due to the singularity at $x = 0$ dominates, and the Taylor expansions are not able to provide accuracy over a large range of values. An increase in the maximum error is also visible for larger thresholds in all three functions. For such thresholds,

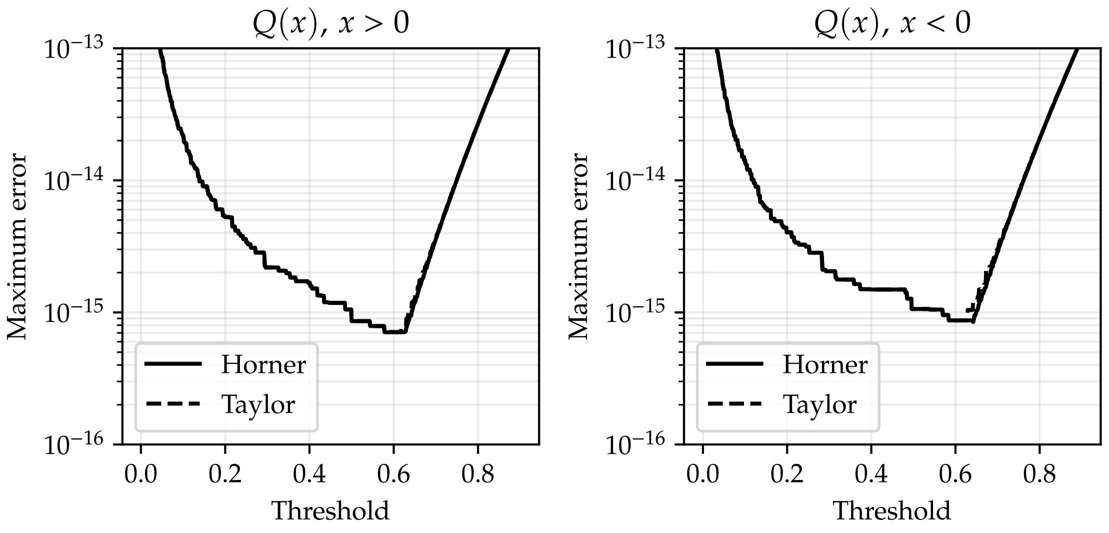


Figure 4.5: Optimal small-value threshold search for $Q(x)$. For positive x, a minimum error of roughly 7.1×10^{-16} was achieved using a small-value threshold of 0.578; for negative x, a minimum error of roughly 8.7×10^{-16} was achieved using a small-value threshold of 0.584.

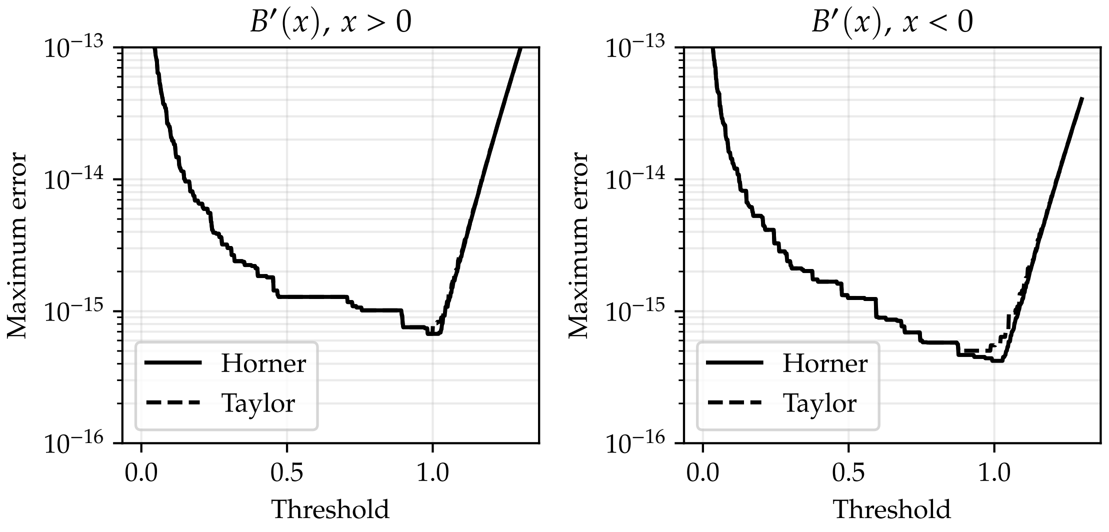


Figure 4.6: Optimal small-value threshold search for $B'(x)$. For positive x, a minimum error of roughly 6.7×10^{-16} was achieved using a small-value threshold of 0.982; for negative x, a minimum error of roughly 4.2×10^{-16} was achieved using a small-value threshold of 0.992.

the Taylor series themselves begin to diverge rapidly from the exact values of $B(x)$, $Q(x)$ and $B'(x)$. Larger thresholds cause the Taylor series approximations to be used outside their range of accuracy, leading to the increase in maximum error.

Figures 4.4, 4.5 and 4.6 show that different optimal thresholds exist for positive and negative x in $B(x)$, $Q(x)$ and $B'(x)$. The optimal thresholds for positive and negative x were 0.295 and 0.279 for $B(x)$, 0.578 and 0.584 for $Q(x)$, and 0.982 and 0.992 for $B'(x)$. An implementation using this algorithm could either choose different thresholds depending on the sign of x or conservatively choose the smaller threshold.

4.2 Device characteristics

This section presents experimental results on the characteristics of the device models implemented in `Semiconductors.jl`. The characteristics are organized by device type, and different methods are presented for each type of device as applicable.

4.2.1 Diodes

One-dimensional silicon diodes were simulated using the model generated by `diode1d(2.0,2.0)`. The output of any non-equilibrium simulation performed by `Semiconductors.jl` is a $3 \times N$ `Matrix{Float64}` giving the values of ψ, n and p at each node in the discretization. These values are shown in thermal equilibrium, with a forward bias of 0.6 V and with a reverse bias of 10 V in Figure 4.7. The thermal equilibrium and reverse-biased cases exhibit depletion regions near the center of the device, where the high electric field forces nearly all mobile carriers away from the junction. The reverse-biased case additionally exhibits boundary layers near the contacts as described in Section 2.1.5 and as illustrated in Figure 2.2.

The forward-biased case shows an excess of mobile charge carriers near the junction due to the increased electron and hole currents. This case also shows the degradation of the charge-neutral regions at high forward bias. Here, the increased terminal currents cause a potential drop on either side of the junction, which implies a nonzero space charge. As the forward bias voltage is further increased, the current through the diode increases exponentially, leading to higher voltage drops on either side of the junction and further increased electron and hole concentrations.

Figure 4.8 shows the I-V characteristics of the 1D diode generated by `diode1d(2.0,2.0)` under forward and reverse bias. These I-V curves were generated using the naïve predictor described in Section 2.2.6. In particular, the bias voltage was increased in fixed steps, using the solution from the previous step as the initial condition for each step. The forward bias voltage was varied from 0 V to 1 V in 10 mV increments; the reverse bias voltage was varied from 0 V to 125 V in 50 mV increments. Both the forward and reverse bias simulations were performed with impact ionization (II), with field-dependent mobility (FDM), and with both II and FDM. The simulations were performed using `non_equilib`, with the keyword argument `catch_conv=true` set to catch convergence errors without terminating in an error.

When the diode was reverse-biased, convergence failed before reaching the maximum bias voltage of 125 V when impact ionization was enabled. The convergence error is likely due to the positive feedback effect in the carrier residuals when impact ionization is enabled, as described in [80]. The points at which convergence failed thus give an estimate of the breakdown voltages of the devices. In particular, convergence failed at V_R = 113.85 V for the II only case and at V_R = 108.40 V for the II and FDM case. The curves in Figure 4.8 indicate that the II and FDM case would likely have a higher breakdown voltage if convergence was possible at higher bias voltages.

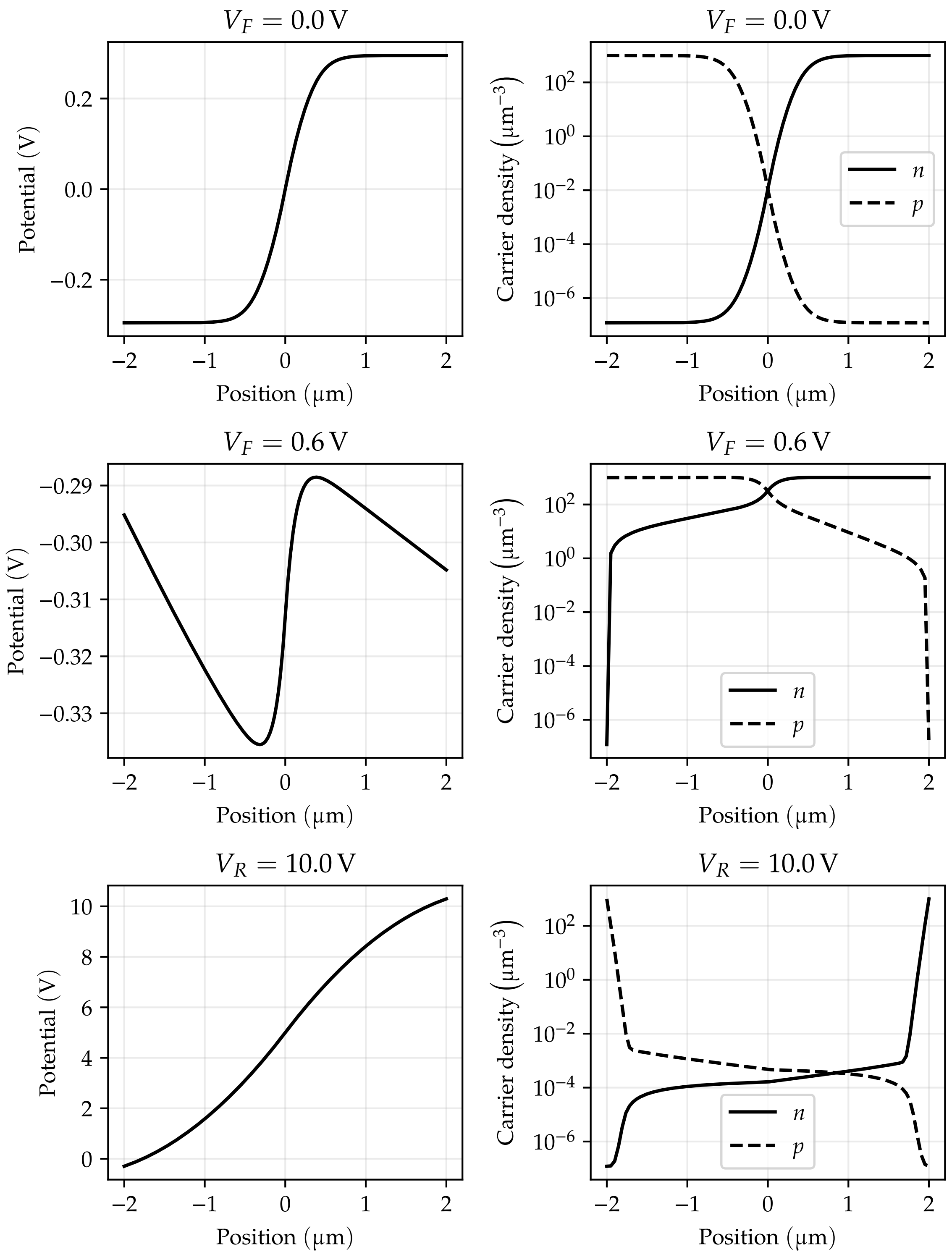


Figure 4.7: Potential and carrier density in 1D diode generated by `diode1d(2.0,2.0)`. Simulations were performed at equilibrium (top), with a forward bias of 0.6 V (middle) and with a reverse bias of 10.0 V (bottom).

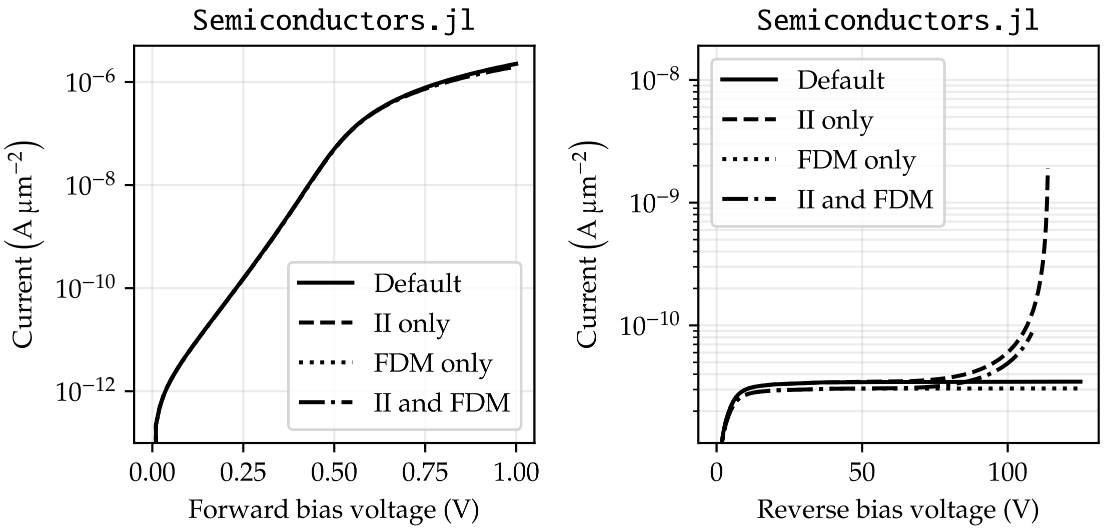


Figure 4.8: Current-voltage characteristic of 1D diode generated by `diode1d(2.0,2.0)`. Simulations were performed with impact ionization (II), with field-dependent mobility (FDM), and with both II and FDM. Convergence failed at $V_R = 113.85\,\text{V}$ for the II only case and at $V_R = 108.40\,\text{V}$ for the II and FDM case.

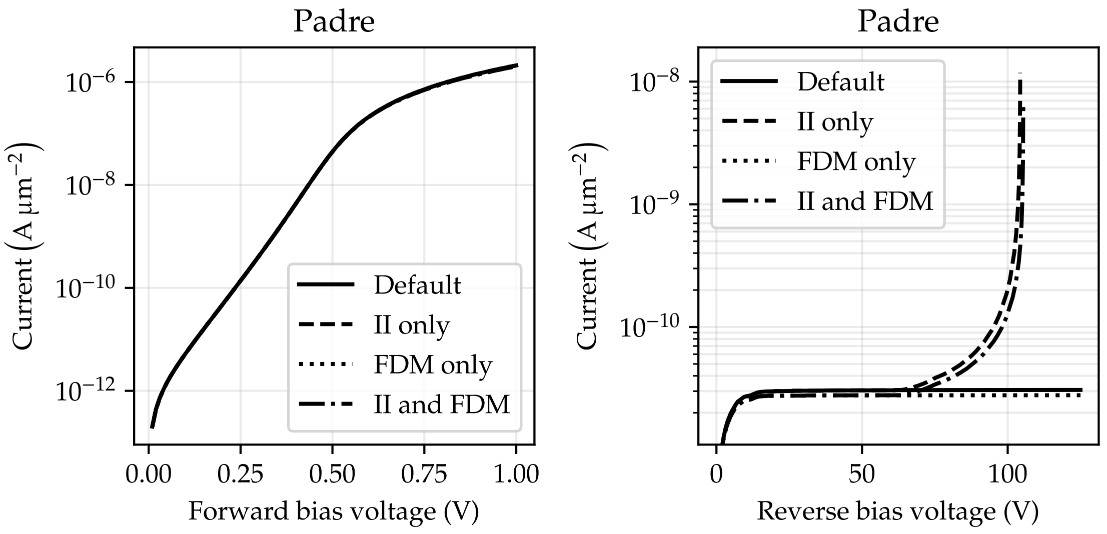


Figure 4.9: Current-voltage characteristic of 1D diode generated by Padre TCAD. Simulations were performed with impact ionization (II), with field-dependent mobility (FDM), and with both II and FDM. Convergence failed at $V_R = 104.35\,\text{V}$ for the II only case and at $V_R = 105.35\,\text{V}$ for the II and FDM case.

To verify the results shown in Figure 4.8, a similar diode was simulated using the Padre TCAD package [119]. The resulting I-V curves are shown in Figure 4.9. The curves are qualitatively similar to the curves produced by `Semiconductors.jl`. The forward-biased characteristic shows a similar *knee voltage* of roughly 0.6 V, where the exponential increase in forward current gives way to a linear voltage drop across the semiconductor regions. The reverse-biased characteristic shows similar breakdown voltages when impact ionization is enabled. A small multiplicative offset is present between the `Semiconductors.jl` curves and the Padre curves for forward and reverse bias voltages. This is likely due to Padre's device physics models; values of physical constants like ε, carrier statistics models and the values of scattering parameters like μ_{n0}, μ_{p0} and the ionization coefficients α_n and α_p could all affect the exact transition points on the I-V curves.

It is known that the Jacobian of the discretized Van Roosbroeck system can become ill-conditioned, which can cause convergence problems near turning points in I-V curves [74, 153]. One measure of the degree to which the Jacobian can impact convergence is the *condition number* of the Jacobian. The condition number of a matrix is the ratio of the largest singular value to the smallest singular value of that matrix. Accordingly, the spectrum of the singular values of the Jacobian can be helpful in determining numerical stability and debugging convergence errors.

Four such spectra are shown in Figure 4.10. These spectra show the relative singular values of the Jacobian, defined as the ratio of each singular value to the maximum singular value. Figure 4.10 shows that the Jacobian of the PDE system for the 1D diode has a condition number on the order of 10^{37} near equilibrium, which decreases to roughly 10^{29} near the device breakdown voltage of 108.3 V. When the diode was heavily reverse-biased, the Jacobian appeared to have a larger number of significant singular values compared to when the diode was biased near equilibrium. This indicates that the linear solve itself was likely not the cause of the convergence error near breakdown; the structure of the residual function or the basins of attraction introduced by Newton's method may have influenced convergence to a greater degree.

To further investigate the source of convergence errors, the condition number of the Jacobian was computed at each bias voltage in an I-V curve simulation. These condition numbers are shown in Figure 4.11. The simulations used to generate Figure 4.11 were performed with field-dependent mobility and impact ionization enabled. Similar results were obtained when field-dependent mobility and/or impact ionization was disabled. In these cases, there was less variation in the condition number between adjacent bias points, but the overall trend remained similar. Figure 4.11 confirms the observation that the condition number of the Jacobian decreases under heavy reverse bias and additionally shows that the condition number increases further when the diode is forward-biased into conduction. A slight increase in the condition number is apparent near the reverse breakdown voltage, but this may be due more to a larger residual than to an ill-conditioned Jacobian.

Two-dimensional silicon diodes were simulated using the model generated by `diode2d(2.0,2.0,1.0,0.5)`. This model is shown in Figure 3.5, with $W_n = W_p = 2\,\mu\text{m}$, $h = 1\,\mu\text{m}$ and $h_c = 500\,\text{nm}$. This model uses the triangular discretization grid shown in Figure 3.6. This model is similar to the 1D diode, except the contacts no longer span the entire height of the device. The values of ψ, n and p are shown for a forward bias voltage of 0.6 V in Figure 4.12. These values are nearly identical to those plotted in Figure 4.7 for $V_F = 0.6\,\text{V}$, except the Dirichlet boundary conditions on n and p now only constrain part of the left and right boundaries. Figure 4.12 was plotted using a linear interpolation on a recursively-refined copy of the discretization grid; see the documentation of `PyPlot.tricontourf` for details.

Figure 4.13 shows the potential for the 2D diode model generated by `diode2d(2.0,2.0,1.0,0.5,bgmesh=true)` at a forward bias voltage of 0.6 V. This model uses the discretization grid with background mesh shown in Figure 3.7. The background mesh allows the potential to be solved

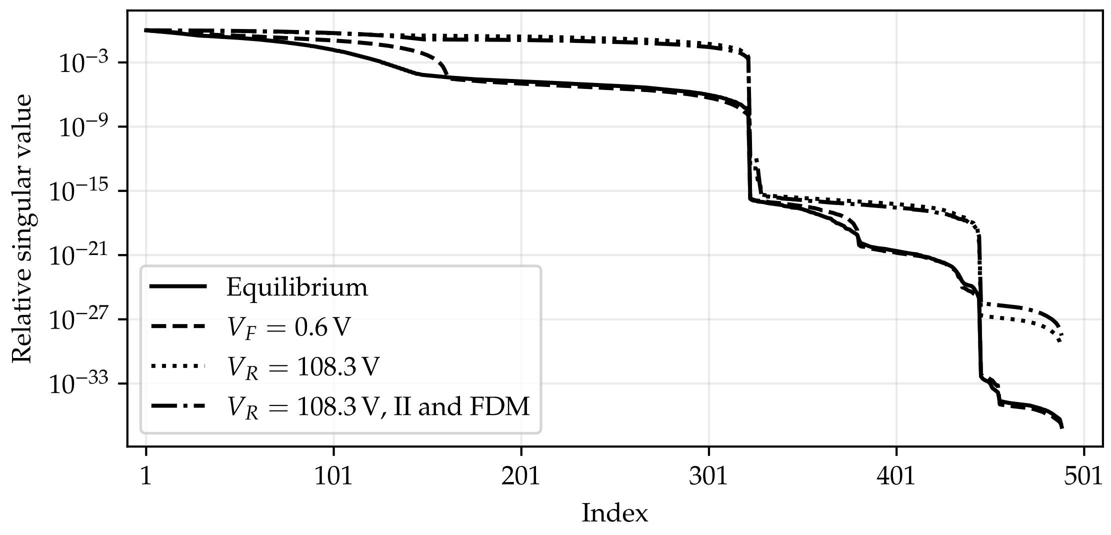


Figure 4.10: Spectra of Jacobian matrices generated by `diode1d(2.0,2.0)`. The system Jacobian has size 489×489. The condition numbers of the matrices are 1.090×10^{37}, 7.556×10^{36}, 4.970×10^{29} and 3.279×10^{28}, from top to bottom in the legend.

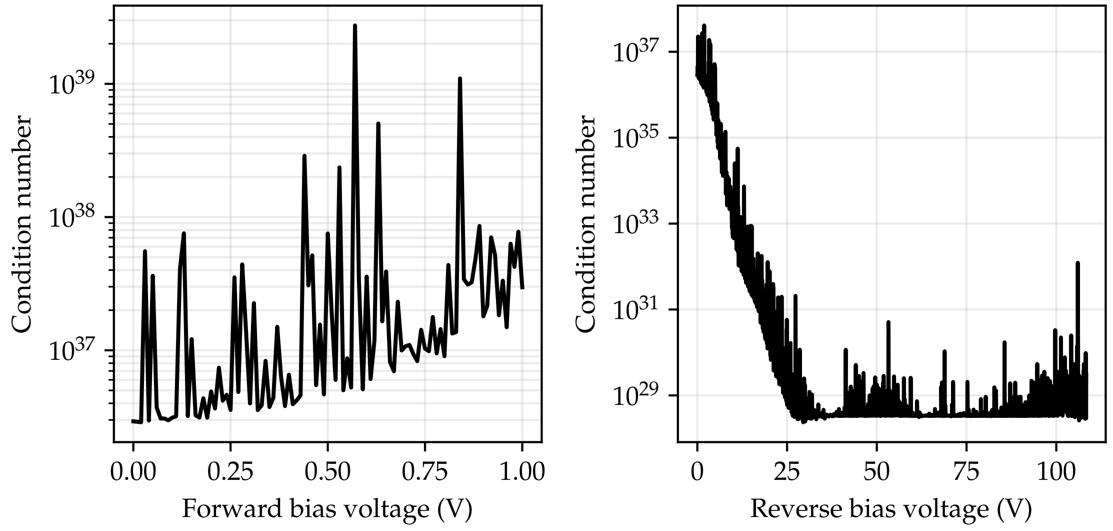


Figure 4.11: Condition number of Jacobian matrices generated by `diode1d(2.0,2.0)` for forward and reverse bias voltages. The forward bias voltages are `0.0:0.01:1.0`, and the reverse bias voltages are `0.0:0.05:108.35`.

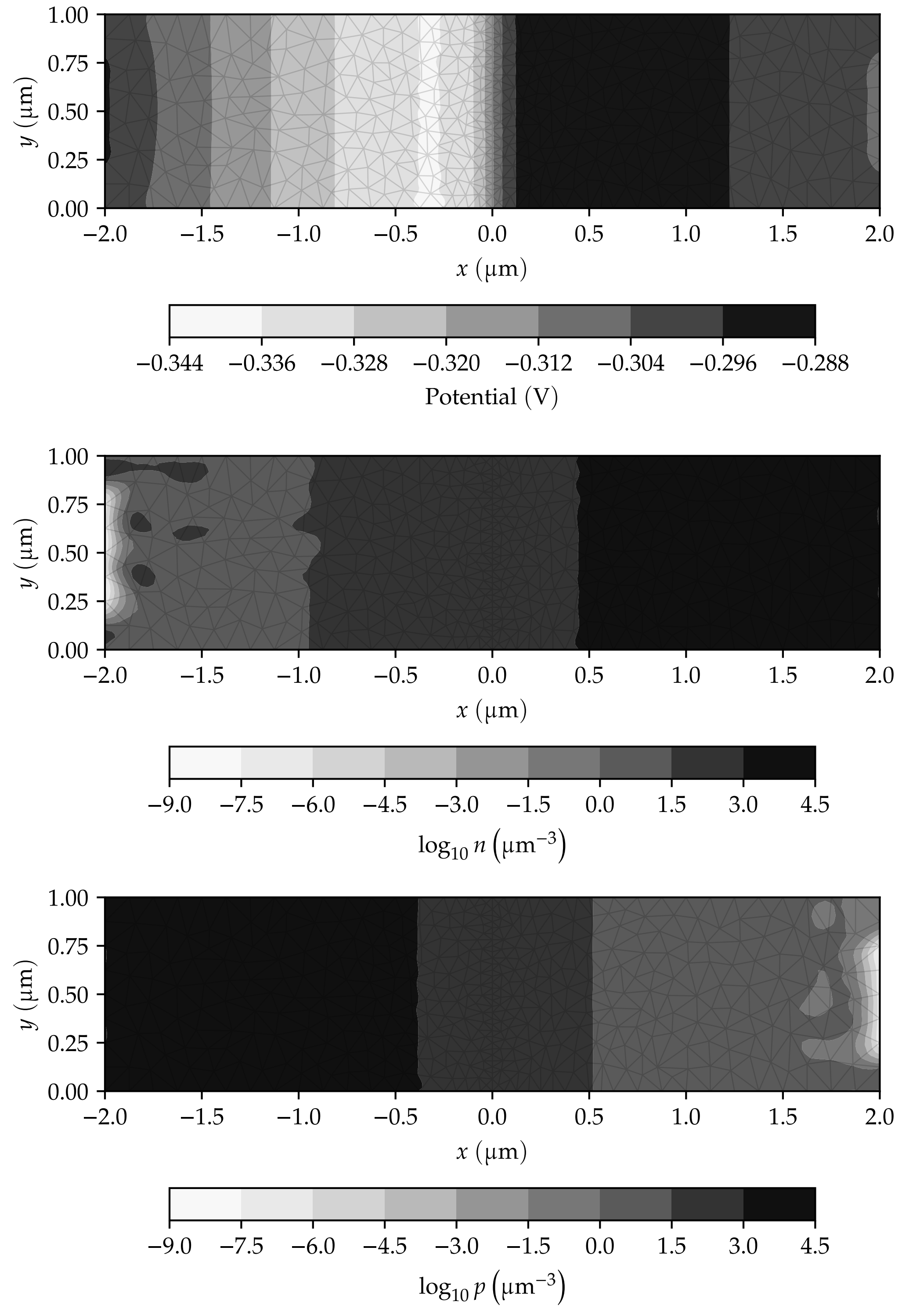


Figure 4.12: Potential (top), electron density (middle) and hole density (bottom) in 2D diode generated by `diode2d(2.0,2.0,1.0,0.5)`. Simulation was performed with a forward bias of 0.6 V.

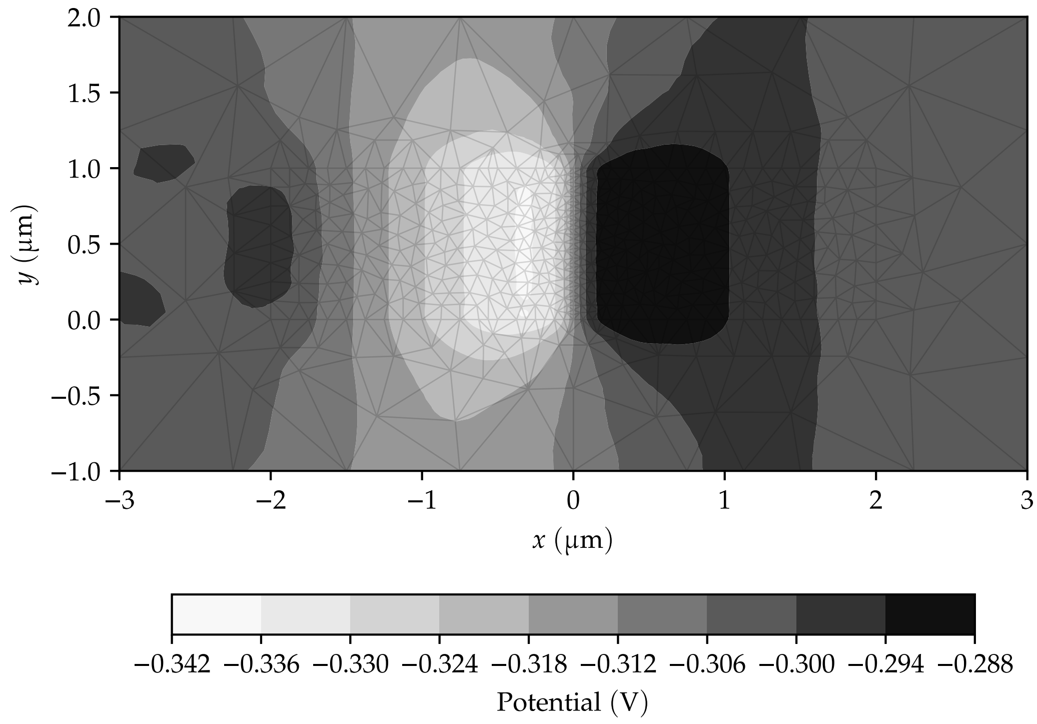


Figure 4.13: Potential in 2D diode with background mesh generated by `diode2d(2.0,2.0,1.0, 0.5,bgmesh=true)`. Simulation was performed with a forward bias of 0.6 V.

outside the device border, which simulates the interaction between the diode and its environment. The environment in this case is either air or a vacuum, which both have $\varepsilon = \varepsilon_0$.

Figure 4.14 shows the I-V characteristics of the 2D diode generated by `diode2d(2.0,2.0,1.0, 0.5)` under forward and reverse bias. These I-V curves were generated using the naïve predictor described in Section 2.2.6. The bias voltages were varied identically to the simulations performed to generate Figure 4.8. Since the 2D diode model has limited variation along the y axis and is similar to the 1D problem, its I-V curves are also similar to those of the 1D model. A slight error in the currents is apparent for moderate forward and reverse bias voltages; these errors are plotted in Figure 4.15.

When the 2D diode is forward biased, Figure 4.15 shows that a modest error near 6% exists for small bias voltages. This error decreases as the bias voltage is increased, eventually becoming negative near the knee voltage of 0.5 V. Beyond this point, a significant negative error exists. This is likely caused by current crowding at the contacts; since the contacts now only span a fraction of the height of the device, the current field lines must group together in a smaller area near the contacts. This effect is not modeled in the 1D simulation. When the 2D diode is reverse biased, a modest error also exists for small bias voltages. This error increases rapidly as V_R approaches 100 V due to the difference in breakdown voltages between the two devices. This could be caused either by the relatively coarser discretization grid along the x axis or by the shorter contacts which provide fewer sites at which avalanche breakdown can occur.

The vector-valued quantities **E** and **J** can illustrate behaviors of the 2D diode not present in the 1D simulation. Figure 4.16 shows these quantities in the 2D diode generated by `diode2d(2.0,2.0,1.0,0.5)` under a forward bias of 0.6 V. A large electric field exists near the center of the device;

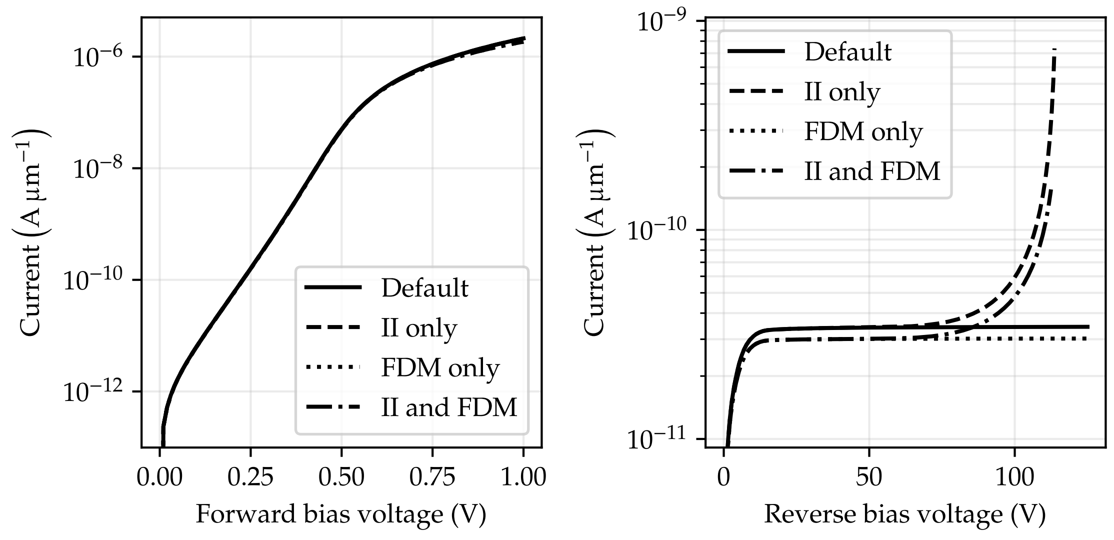


Figure 4.14: Current-voltage characteristic of 2D diode generated by `diode2d(2.0,2.0,1.0,0.5)`. Simulations were performed with impact ionization (II), with field-dependent mobility (FDM), and with both II and FDM. Convergence failed at $V_R = 113.70\,\mathrm{V}$ for the II only case and at $V_R = 112.50\,\mathrm{V}$ for the II and FDM case.

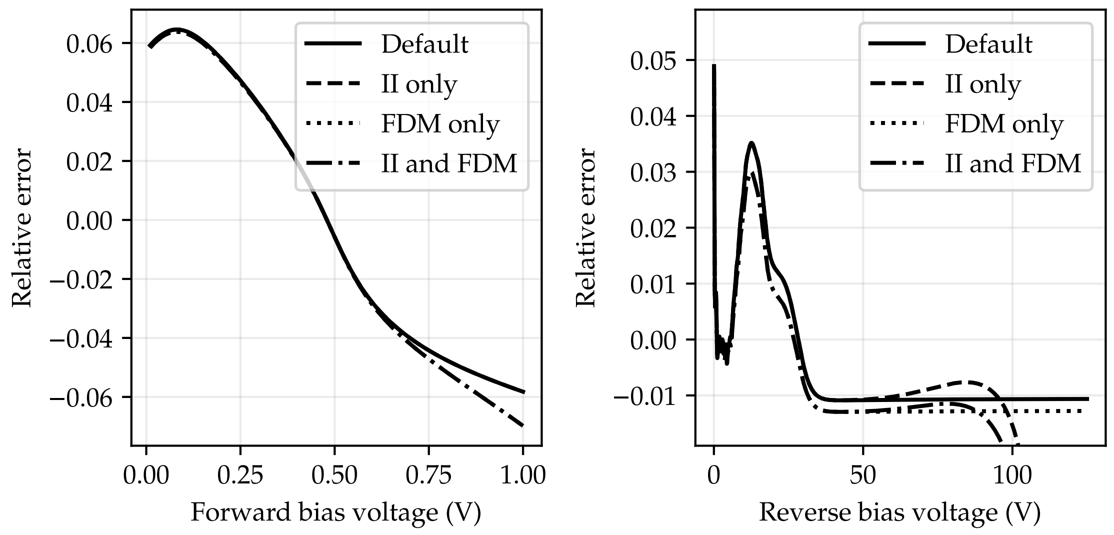


Figure 4.15: Relative error between I-V curves of 2D diode and 1D diode, for forward (left) and reverse (right) bias voltages. An increase in error near $V_R = 100\,\mathrm{V}$ for the II cases is apparent due to the difference in breakdown voltages.

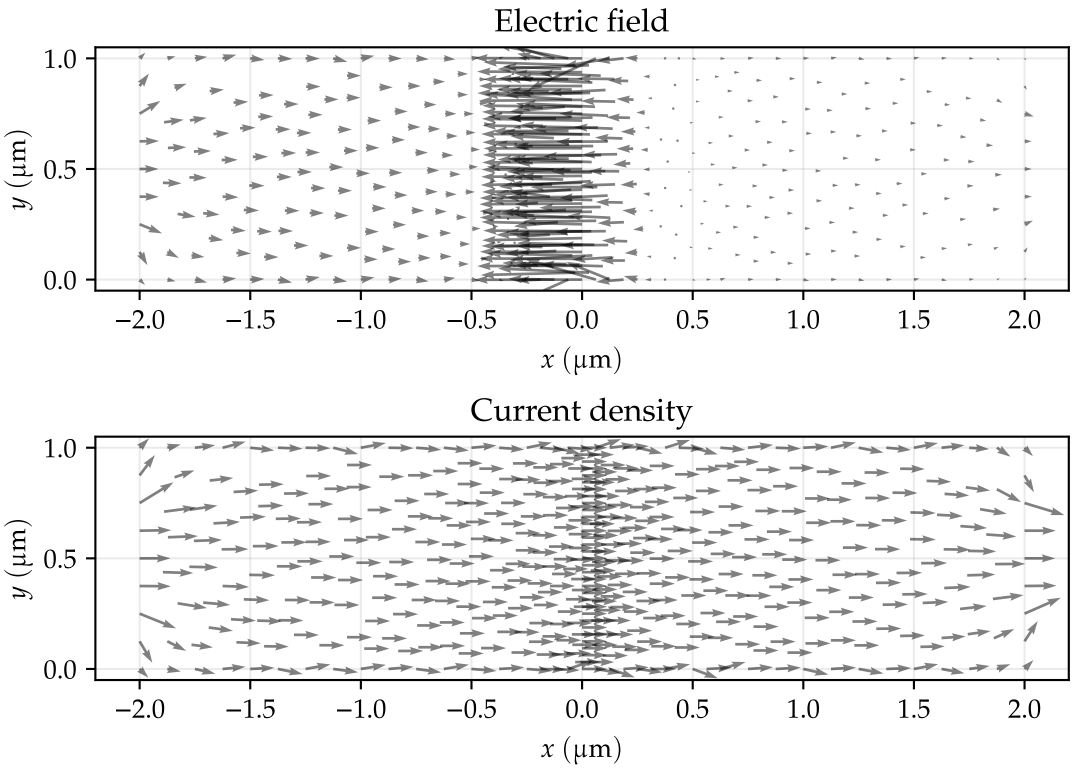


Figure 4.16: Electric field (top) and current density (bottom) in 2D diode model generated by `diode2d(2.0,2.0,1.0,0.5)` with a forward bias of 0.6 V. Arrow length is proportional to field magnitude; one arrow is plotted for each discretization node.

this is the depletion region formed by the doping gradient. A smaller electric field exists on either side of the depletion region due to the applied bias. This electric field causes the drift current that carries electrons to the junction before they are swept across the depletion region by diffusion. The current density is roughly constant in magnitude along the device due to conservation of charge. Slight increases in the magnitude of the current density field are visible near the contacts due to current crowding.

4.2.2 Bipolar transistors

Two-dimensional silicon BJTs were simulated using the model generated by `npn1(5,1,10,5,3,3,0.2)`. This model is shown in Figure 3.8, with $W_e = 5\,\mu\text{m}$, $W_b = 1\,\mu\text{m}$, $W_c = 10\,\mu\text{m}$, $h_{ce} = h_{cc} = 3\,\mu\text{m}$ and $W_{cb} = 200\,\text{nm}$. This model uses the triangular discretization grid shown in Figure 3.9. The default doping values are $N_E = 1 \times 10^6\,\mu\text{m}^{-3}$, $N_B = 1 \times 10^4\,\mu\text{m}^{-3}$ and $N_C = 1 \times 10^3\,\mu\text{m}^{-3}$, which are used in the default abrupt doping profile given in (3.12). The values of ψ, n and p in this device are shown in Figure 4.17.

Figure 4.18 shows the I-V characteristic of the NPN BJT generated by `npn1(5,1,10,5,3,3,0.2)`. This characteristic measures the collector current I_C as a function of collector-emitter voltage V_{CE}. The resulting curve is parametric in the base current I_B. To establish a fixed current through the base contact, `Semiconductors.newton_current!` was used to create the augmented nonlinear system described in Section 2.2.5. The solver parameters used were `damp_initial=0.1`, `damp_growth=1.5`, `max_iters=50` and `damp_search=true`. A high damping value is typically

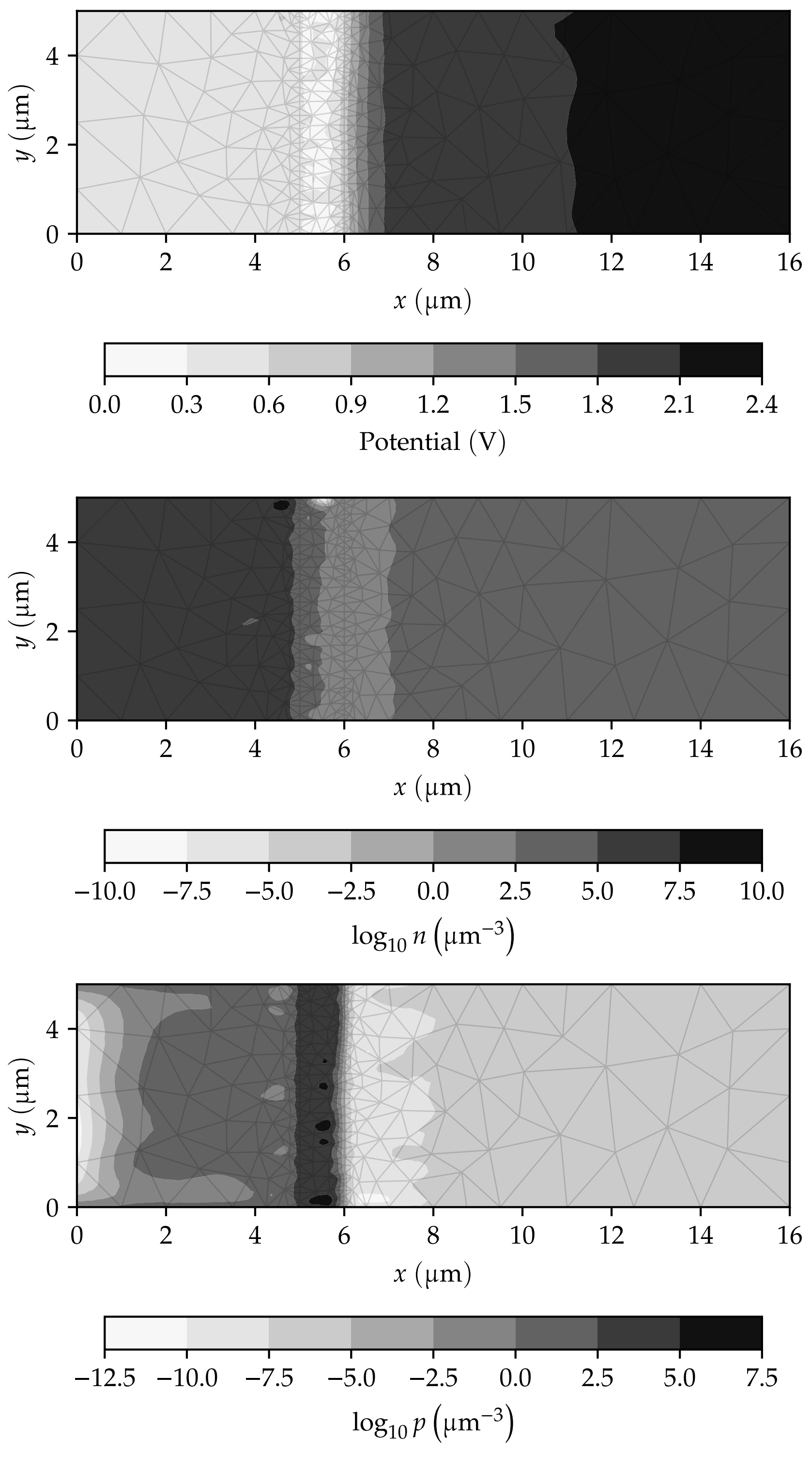


Figure 4.17: Potential (top), electron density (middle) and hole density (bottom) in NPN BJT generated by `npn1(5,1,10,5,3,3,0.2)`. Simulation was performed with $I_B = 5 \times 10^{-7}\,\text{A}\,\mu\text{m}^{-1}$ and $V_{CE} = 2\,\text{V}$.

necessary to avoid large variations in bias voltages when using current boundary conditions. This simulation was performed with base currents ranging from $I_B = 1 \times 10^{-7}\,\mathrm{A\,\mu m^{-1}}$ to $I_B = 9 \times 10^{-7}\,\mathrm{A\,\mu m^{-1}}$ in five equal steps.

The key features of BJT operation are clearly visible in Figure 4.18. The device exhibits well-defined forward and reverse active regions. The current gain in the forward active region is much greater than in the reverse active region since $N_E \gg N_C$. A saturation voltage in the range of 100 mV to 1 V is visible, and limited quasi-saturation is exhibited at the highest base current. The Early effect is also visible. This effect describes the slight increase in collector current as V_{CE} is increased due to widening of the depletion region of the base-collector junction.

Figure 4.19 shows the large signal current gain β_F of the NPN BJT generated by `npn1(5,1,10,5,3,3,0.2)` as a function of collector current. This characteristic is typically included in the datasheets of commercial BJTs to describe the performance of the device over a wide range of operating currents. In physical devices, β_F is roughly constant for moderate operating currents. At very low collector currents, reverse leakage current through the base-collector junction dominates I_B, which causes the gain to roll off as $I_C \to 0$. At very high collector currents, voltage drop across the collector and emitter regions introduces a resistive effect, which causes the gain to roll off as $I_C \to \infty$.

As shown in Figure 4.19, the doping profile has a significant effect on the current gain of a BJT. In this figure, the default abrupt doping profile given in (3.12) gives the lowest gain for most operating points. The smooth doping profile given in (3.13) gives the second lowest gain, since it is nearly identical to the abrupt profile except at the junctions. The exponentially-graded base and linearly-graded base doping profiles given in (3.14) and in (3.15) give higher gains, with the exponentially-graded base providing a gain near 100 at extremely low bias currents. Many other factors such as the ratio N_E/N_C can affect the current gain; for a comprehensive analysis, see [53].

4.2.3 MOSFETs

Two-dimensional silicon MOSFETs were simulated using the model generated by `mos1(0.1,0.05,0.2,0.05,0.002,0.1,0.025)`. This model is shown in Figure 3.12, with $L = 100\,\mathrm{nm}$, $L_s = L_d = 50\,\mathrm{nm}$, $h = 200\,\mathrm{nm}$, $h_s = h_d = 50\,\mathrm{nm}$, $t_{ox} = 2\,\mathrm{nm}$, $c_1 = 100\,\mathrm{nm}$ and $c_2 = c_3 = 25\,\mathrm{nm}$. This model uses the triangular discretization grid shown in Figure 3.13. The default doping values are $N_s = N_d = 2 \times 10^8\,\mathrm{\mu m^{-3}}$, $N_{ch} = -1 \times 10^6\,\mathrm{\mu m^{-3}}$ and $N_b = -5 \times 10^4\,\mathrm{\mu m^{-3}}$, which specifies an n-channel MOSFET with a heavily p-doped channel region and a lightly p-doped bulk region. The default abrupt doping profile was used. The electrostatic potential in this device is shown in Figure 4.20 for $V_{DS} = 1\,\mathrm{V}$ and $V_{GS} = 0.5\,\mathrm{V}$.

Figures 4.21 and 4.22 show the I_D-V_{DS} and I_D-V_{GS} characteristics of the n-channel MOSFET generated by `mos1(0.1,0.05,0.2,0.05,0.002,0.1,0.025)`. The I_D-V_{DS} characteristic shows the effect of channel length modulation as V_{DS} is increased. This effect causes I_D to increase when V_{GS} is held constant due to the widening of the depletion region around the drain diffusion. This characteristic also shows the importance of the field-dependent mobility model in MOSFET simulations. The currents simulated using a constant mobility are on the order of 10 times higher than the currents simulated using a field-dependent mobility. This is mainly due to the magnitude of the electric field in the channel region, which increases the rate of carrier scattering, decreasing the effective mobility.

The I_D-V_{GS} characteristic shown in Figure 4.22 shows the threshold voltage and leakage current of the MOSFET generated by `mos1(0.1,0.05,0.2,0.05,0.002,0.1,0.025)`. The threshold voltage is the point at which the exponential subthreshold current gives way to a sub-exponential saturation current. For this MOSFET, the threshold voltage is roughly 2 V for the simulated values

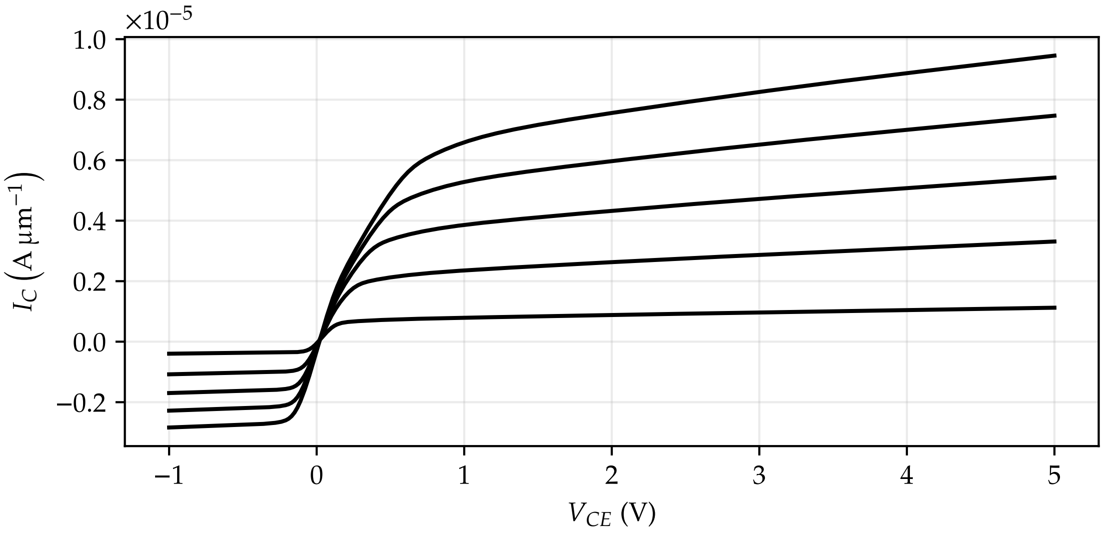


Figure 4.18: Current-voltage characteristic of NPN BJT generated by `npn1(5,1,10,5,3,3,0.2)` showing operation in forward and reverse active regions. Base current was varied from $I_B = 1 \times 10^{-7}$ A µm^{-1} to $I_B = 9 \times 10^{-7}$ A µm^{-1} in five equal steps.

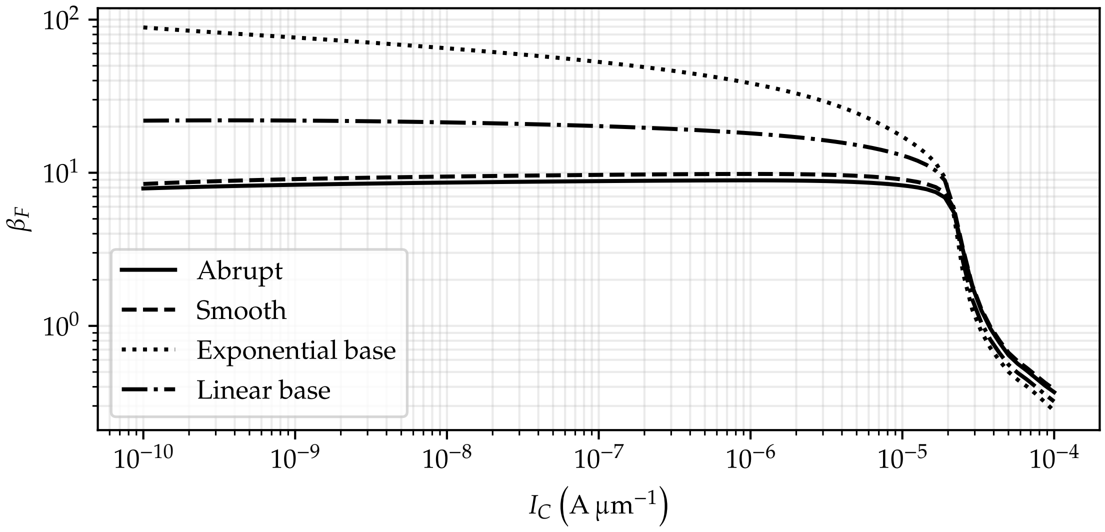


Figure 4.19: Large-signal forward current gain (β_F) versus collector current for NPN BJT generated by `npn1(5,1,10,5,3,3,0.2)`. Results are shown for the four built-in doping profiles in `npn1`.

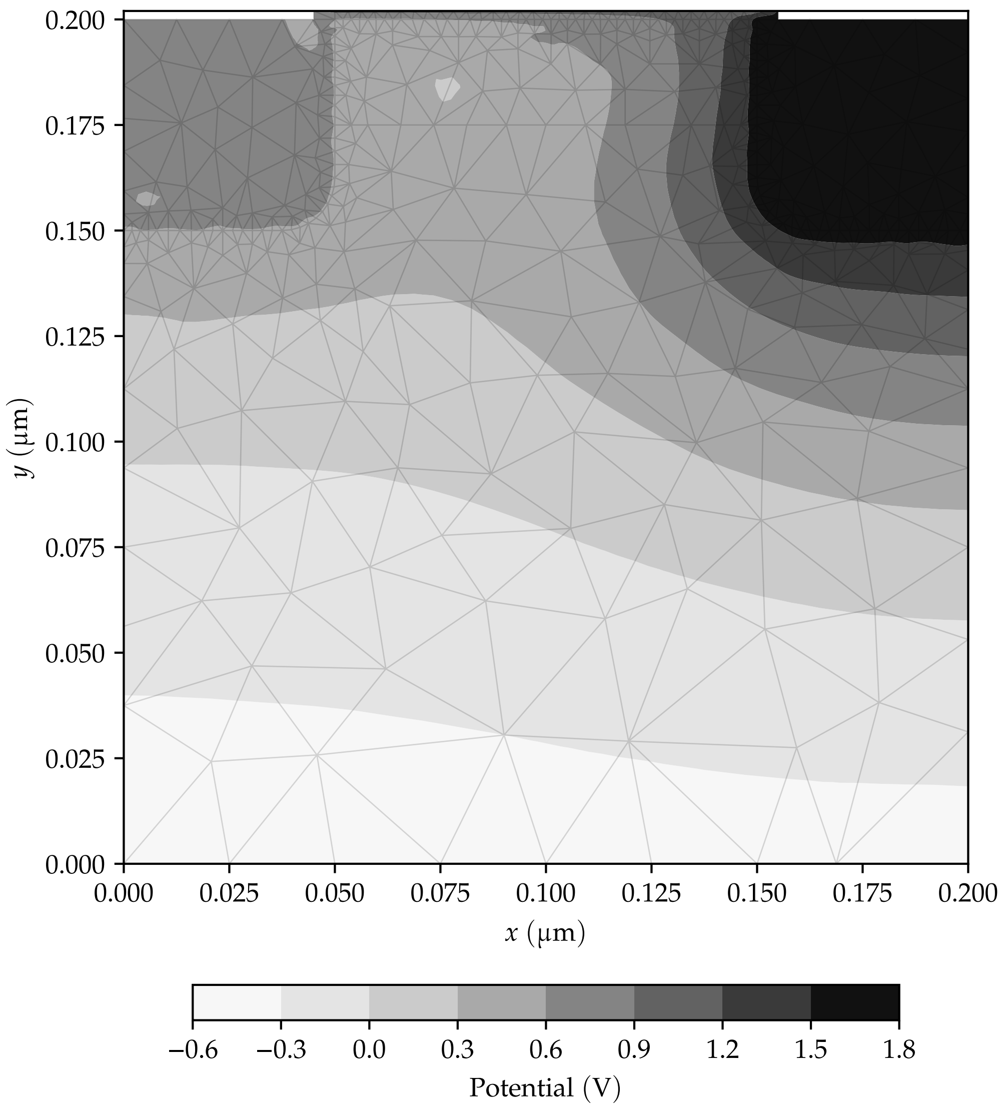


Figure 4.20: Electrostatic potential in *n*-channel MOSFET generated by `mos1(0.1,0.05,0.2,0.05,0.002,0.1,0.025)`. Simulation was performed with $V_{DS} = 1\,\text{V}$ and $V_{GS} = 0.5\,\text{V}$.

of V_{DS}. When V_{DS} is high, the exponential subthreshold current region can be small or nonexistent. This is due to substrate leakage currents, which flow through the reverse-biased bulk-drain junction and can cause power loss in digital circuits. This I_D-V_{GS} characteristic also shows the magnitude of the difference in currents between the simulation using constant mobility and the simulation using field-dependent mobility.

The vector-valued quantities **E** and **J** can provide insight into the behavior exhibited by the I-V characteristics of a MOSFET. Figure 4.23 shows these quantities in the *n*-channel MOSFET generated by `mos1(0.1,0.05,0.2,0.05,0.002,0.1,0.025)`, with $V_{DS} = 1\,\mathrm{V}$ and $V_{GS} = 0.5\,\mathrm{V}$. A significant vertical electric field exists at the bulk-oxide interface and within the gate oxide. This electric field draws mobile electrons to the interface, creating an *inversion layer* that carries the drain current when the MOSFET is on. This is reflected in the vector plot of the current density, which shows that current enters through the drain contact, is swept across the channel by the lateral electric field at the drain and exits through the source contact. The behavior of these vector-valued quantities in *accumulation* ($V_{GS} < 0$) is also instructive; this behavior is discussed further in Section B.3.

4.3 Continuation

Figure 4.24 shows I-V curves for a reverse-biased 1D diode generated by Padre TCAD. This diode is similar to the model generated by `diode1d(2.0,2.0)`; it has 162 nodes in its mesh (as opposed to 163), and it has identical dopings of $N_A = N_D = 1 \times 10^3\ \mu\mathrm{m}^{-3}$. These simulations were performed using the `contin` command in Padre. Since Padre does not support continuation in the logarithm of the terminal current, the continuation process had to be invoked multiple times along the curve to allow proper scaling of the voltage and current. These curves show two distinct breakdown voltages for the II and FDM case and for the II case. The first and second breakdown voltages were 105.4 V and 26.7 V for the II and FDM case, and 104.3 V and 84.2 V for the II only case.

4.4 Surrogatization

This section presents experimental results on the surrogatization of the device models implemented in `Semiconductors.jl`.

4.4.1 Methods

Given a `Semiconductors.Device` instance and the associated discretization grid, the surrogatization process seeks to create a similar device with a coarser grid that closely approximates the I-V characteristic of the first device. This is accomplished by optimization of the device parameters, which are trained to minimize the *root-mean-square* relative error (RMSRE) between the I-V curve of the surrogate model and of the original model. Specifically, we optimize the doping profile of the surrogate model by defining this quantity as a piecewise constant function along the device. The doping profile is part of the reaction discretization of the Poisson equation, so one value should be defined at each node in the grid.

Given two sets of currents $I_1[k]$ and $I_2[k]$, where $k \in [1, M]$, the RMSRE of I_2 with respect to I_1 is defined as

$$\mathrm{RMSRE} = \sqrt{\frac{1}{M}\sum_{k=1}^{M}\left(\frac{I_2[k]}{I_1[k]} - 1\right)^2}.$$

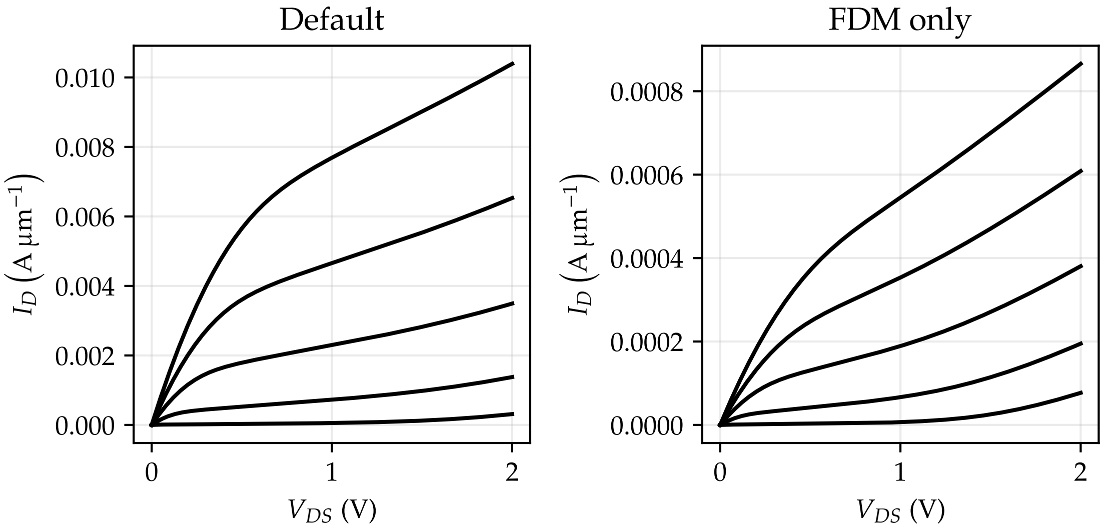


Figure 4.21: Drain-source voltage I-V characteristic of n-channel MOSFET generated by `mos1(0.1, 0.05,0.2,0.05,0.002,0.1,0.025)`. Gate-source voltage was varied from V_{GS} = 0.1 V to V_{GS} = 0.9 V in five equal steps. Simulations were performed with constant mobility (left) and with field-dependent mobility (right).

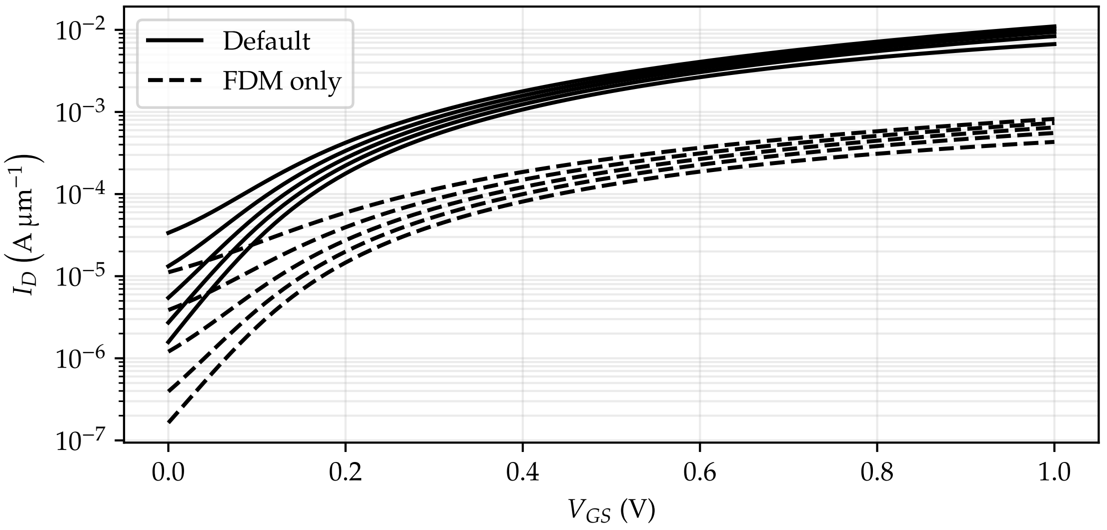
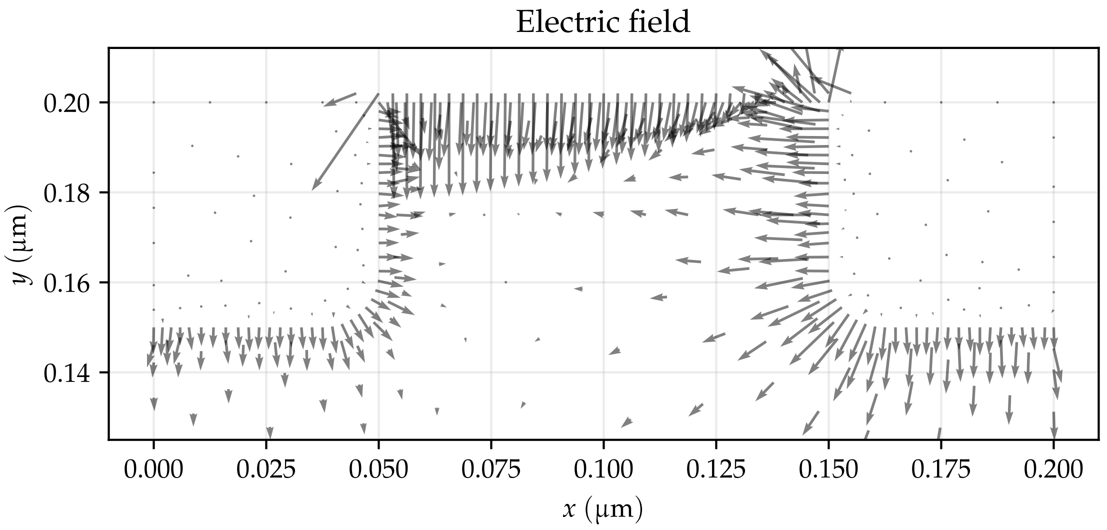


Figure 4.22: Gate-source voltage I-V characteristic of n-channel MOSFET generated by `mos1(0.1,0.05,0.2,0.05,0.002,0.1,0.025)`. The drain-source voltage was varied from V_{DS} = 0.5 V to V_{DS} = 1.5 V in five equal steps. Simulations were performed with constant mobility ("Default") and with field-dependent mobility ("FDM only").



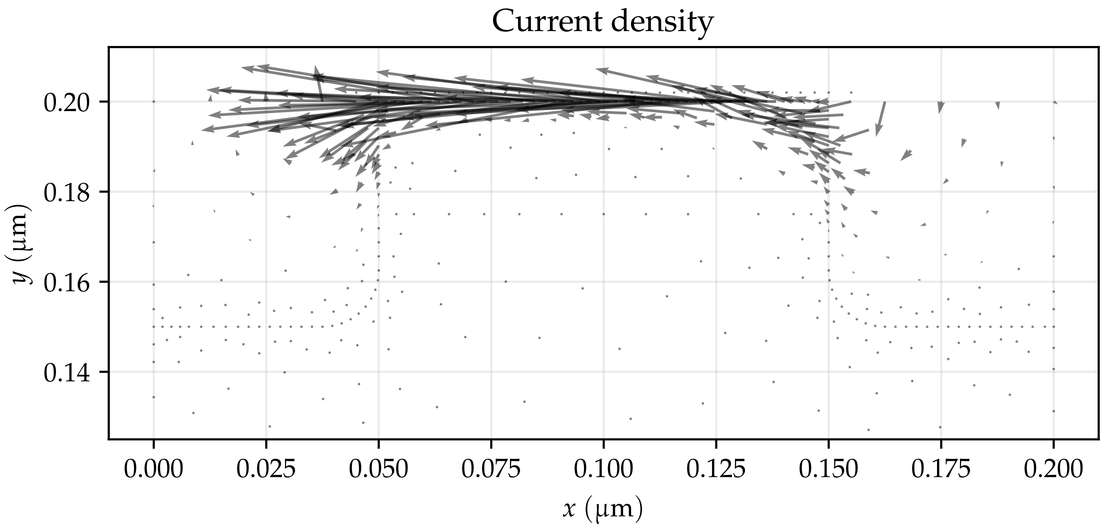


Figure 4.23: Electric field (top) and current density (bottom) in *n*-channel MOSFET generated by `mos1(0.1,0.05,0.2,0.05,0.002,0.1,0.025)` with $V_{DS} = 1\,\mathrm{V}$ and $V_{GS} = 0.5\,\mathrm{V}$. Arrow length is proportional to field magnitude; one arrow is plotted for each discretization node.

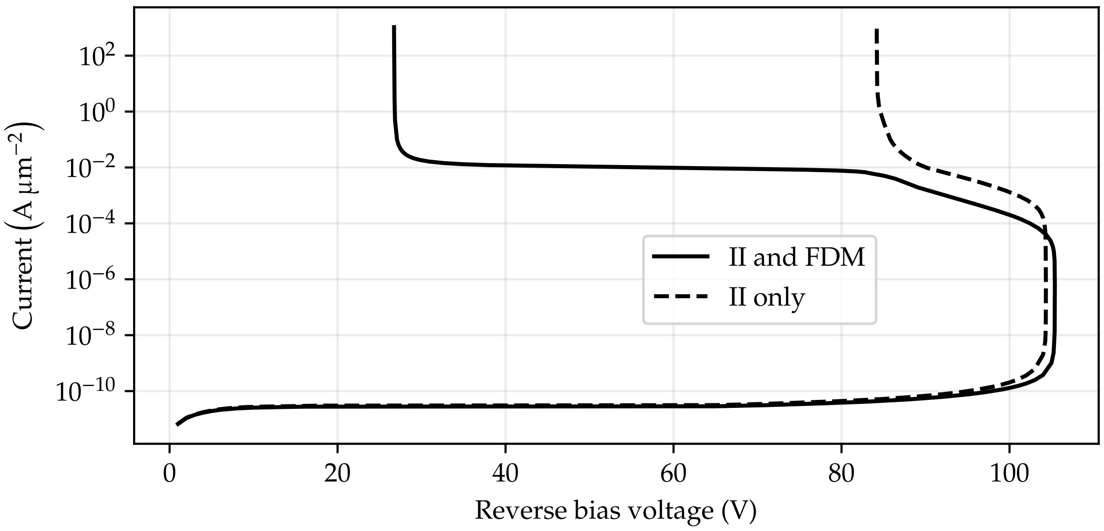


Figure 4.24: I-V curves of reverse-biased 1D diode generated by Padre TCAD using arc-length continuation. The first and second breakdown voltages were 105.4 V and 26.7 V for the II and FDM case, and 104.3 V and 84.2 V for the II only case.

This metric was chosen since it measures relative error instead of absolute error, which is necessary when comparing I-V curves since the underlying data vary over several orders of magnitude. The normalization by M makes the loss length-independent so that I-V curves traced with different values of M give a roughly equivalent RMSRE for identical data.

To minimize the RMSRE, we use the Broyden-Fletcher-Goldfarb-Shanno (BFGS) optimizer implemented by Optim.jl. This optimizer belongs to a class of algorithms known as *secant optimizers*, which uses the gradient to approximate a step of Newton's method [111]. This is generally more efficient than using Newton's method directly, since Newton's method requires the full $3N \times 3N$ Hessian matrix, while secant optimizers only require the $3N \times 1$ gradient vector. Other gradient-based optimizers like ADAM [79] were tested in the surrogatization approach but were found to have inferior performance with respect to quickly reducing the loss function. The optimization of the surrogate models was performed using `DiffEqFlux.sciml_train`, which implements unconstrained optimization by using `Zygote.jl` for AD.

Preliminary experiments were conducted to analyze the unoptimized surrogate model and the gradients of the RMSRE loss function. Figure 4.25 compares the I-V curves of the fine-grained 1D diode model generated by `diode1d(2.0,2.0)` and the unoptimized surrogate model generated by `diode1d(2.0,2.0,ha=0.8,hb=0.16,hc=0.8)`. Both models were simulated using a forward bias voltage of $V_F \in [0.01, 1]$V, which was increased in 10 mV steps. The equilibrium case $V_F = 0$ V was excluded since it results in a current of zero; numerical error in the Newton solve may thus give extremely small currents at that bias point which distort the RMSRE metric. The fine-grained model has 163 discretization nodes, and the coarse-grained model has 11 discretization nodes. The relative error of the coarse-grained model peaks at over 50% for low bias voltages and remains relatively high along the entire I-V curve. The RMSRE of this unoptimized model is 0.294 17.

Optimization of the surrogate model requires the gradient of the RMSRE loss function with respect to the doping profile. The loss function should be defined such that a surrogate model with similar characteristics to the fine-grained model gives a low loss value. The interpretation of

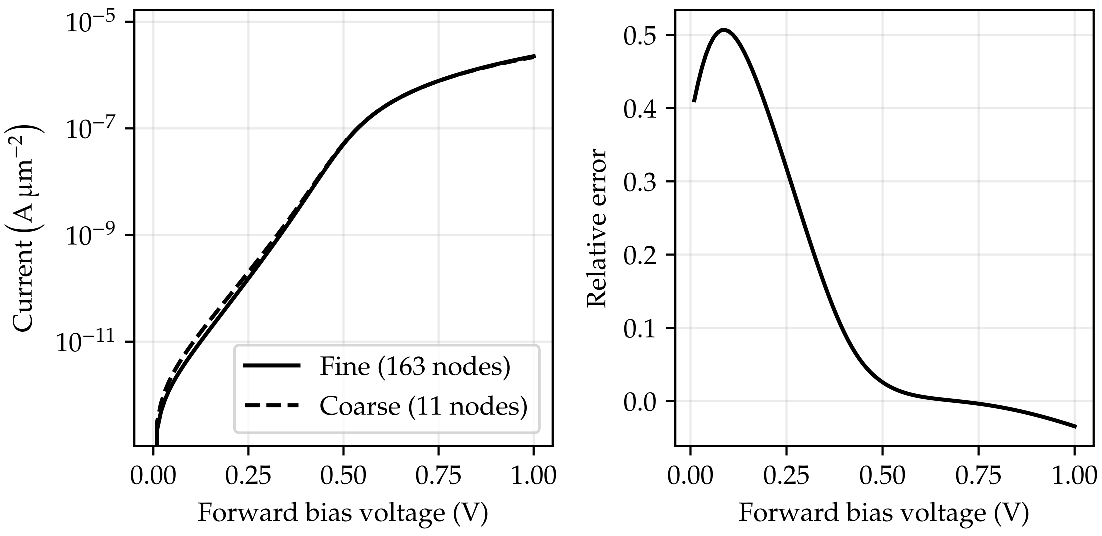


Figure 4.25: Comparison of I-V curves for 1D diode using fine and coarse discretization grids. The fine-grained model was generated by `diode1d(2.0,2.0)`, and the coarse-grained model was generated by `diode1d(2.0,2.0,ha=0.8,hb=0.16,hc=0.8)`.

"similar" here is difficult; we interpret this as meaning that the I-V curves of the two devices are close to within a small tolerance over some range of bias voltages. The particular range of bias voltages over which the RMSRE is computed can affect the optimized doping profile. Figure 4.26 shows the gradient of the RMSRE loss function for three ranges of bias voltages.

The top plot in Figure 4.26 shows $V_F \in (0\,\mathrm{V}, 0.5\,\mathrm{V}]$. For this range of bias voltages, the device is only weakly forward-biased, and currents flow by diffusion across the depletion region. The doping values adjacent to the junction thus have the most significant effect on the loss function; a positive gradient in the *p* region means that less *p*-doping would decrease the loss, and a negative gradient in the *n* region means that less *n*-doping would decrease the loss. The middle plot in Figure 4.26 shows $V_F \in (0.5\,\mathrm{V}, 1\,\mathrm{V}]$. For this range of bias voltages, the device is strongly forward-biased, and currents flow due to the large electric field that exists in the semiconductor regions. The doping values at the contacts have the most significant effect on the loss function for these bias voltages.

The bottom plot in Figure 4.26 shows $V_F \in (0\,\mathrm{V}, 1\,\mathrm{V}]$. This range includes both of the previous ranges, but the gradient closely resembles the gradient shown in the top plot. This is because there is a much higher relative error at lower voltages, as indicated by Figure 4.25. The overall loss gradient acts mainly to minimize this error as opposed to the relatively smaller error present at higher bias voltages.

The trends illustrated in Figure 4.26 are also explained by the results in Figure 4.27, which shows the gradients of the current with respect to doping profile in the fine-grained 1D diode model generated by `diode1d(2.0,2.0)`. In the simulations where the device is only weakly forward biased, the doping values on either side of the junction have the most significant effect on the total current. This behavior changes between $V_F = 0.4\,\mathrm{V}$ and $V_F = 0.5\,\mathrm{V}$, at which point the doping values at the contacts begin to dominate the gradient. This effect continues for the remaining bias voltages. The gradients are the opposite sign of those shown in the middle plot of Figure 4.26 since the relative error at high bias voltages is negative in the unoptimized surrogate.

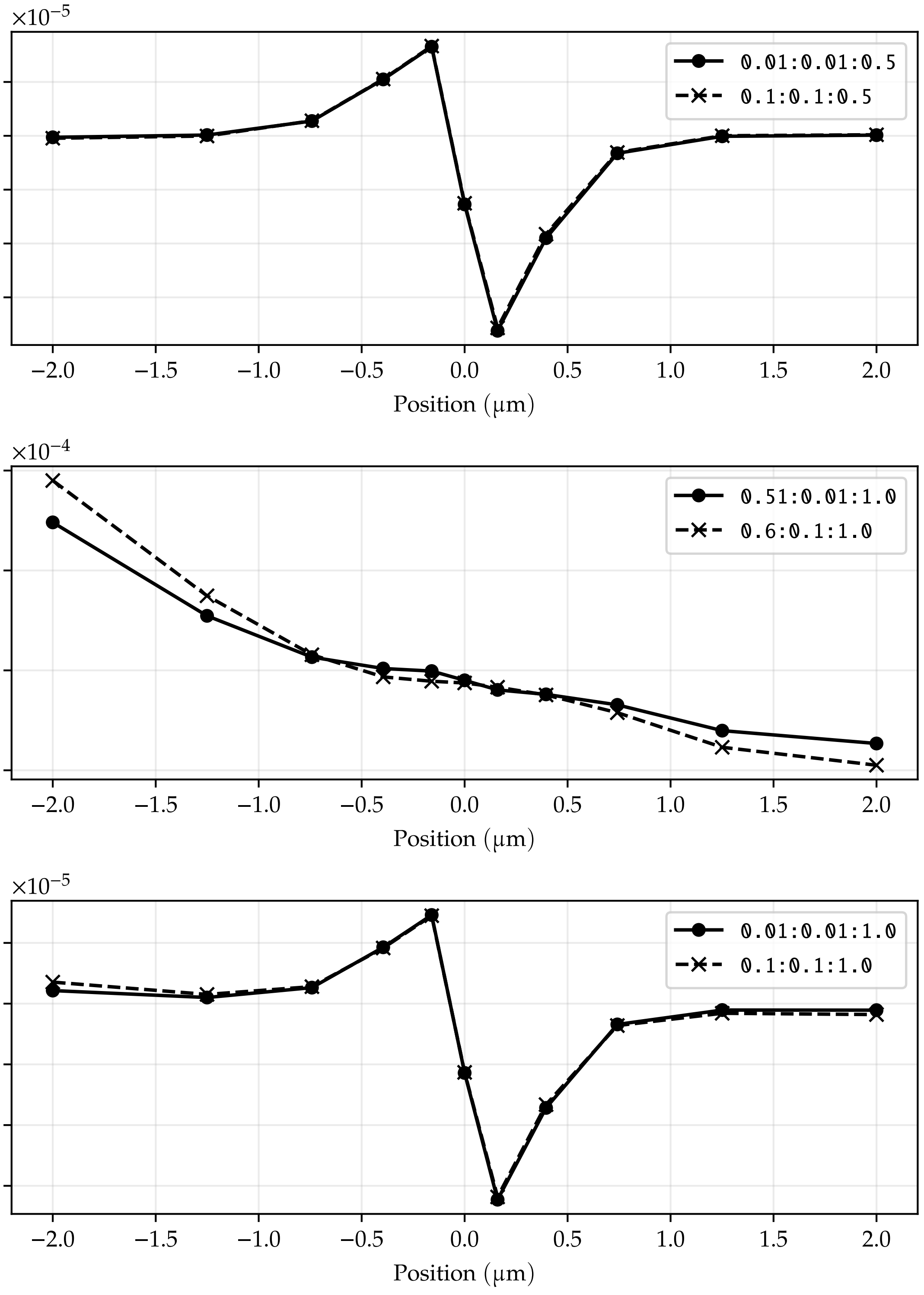


Figure 4.26: Loss gradients in surrogate model using three different bias voltage ranges. Plot legends show the range of V_F used to generate each gradient. Solid lines were computed using 10 mV bias steps, and dashed lines were computed using 100 mV bias steps.

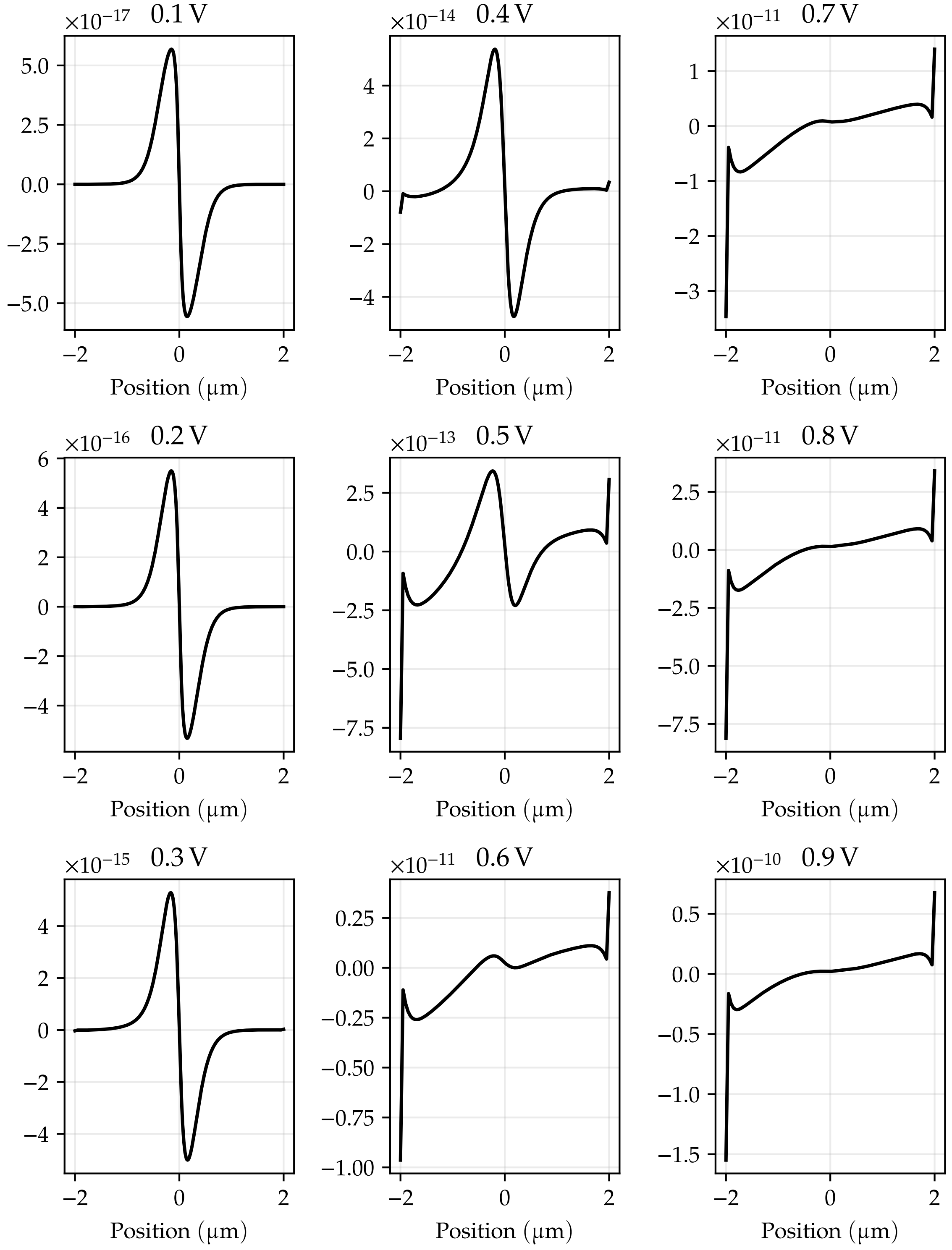


Figure 4.27: Gradients of current with respect to doping profile in 1D diode model generated by `diode1d(2.0,2.0)`. Forward bias voltages from $V_F = 0.1\,\text{V}$ to $V_F = 0.9\,\text{V}$ are shown.

4.4.2 One-dimensional diodes

Figures 4.28 through 4.31 show the results of the surrogatization process applied to 1D diodes. Figures 4.28 and 4.29 were generated using 100 mV bias steps to compute loss, and Figures 4.30 and 4.31 were generated using 10 mV bias steps to compute loss. Both bias steps were effective in training surrogates; using 100 mV bias steps resulted in a maximum relative error of roughly 1%, while using 10 mV bias steps resulted in a much lower maximum error of roughly 0.1%. Both bias steps produced qualitatively similar optimal doping profiles, which both specify a more strongly *n*-doped cathode contact.

The difference in performance between the two bias steps is likely due to the optimizer overfitting to the sampled points on the I-V curve. In Figure 4.28, the minima of the relative error have been shifted to align with the 100 mV bias increments: the minima occur at roughly 0.1 V, 0.2 V, 0.3 V and so on. This artificially decreases the loss without decreasing the relative error for much of the I-V curve between 0 V and 1 V. This behavior was avoided by using the finer 10 mV bias increments shown in Figure 4.30. The relative error using this finer bias increment is more smooth since this loss function considers a larger number of points on the I-V curve. The disadvantage of using this finer bias increment is the required simulation time—the simulation using 10 mV bias increments had a runtime roughly ten times longer than the simulation using 100 mV bias increments.

This overfitting hypothesis is validated by the results in Figures 4.32 and 4.33. Here, the BFGS optimizer was run with a maximum of 1000 iterations. The optimizer terminated after 862 iterations with a loss of 3.5339×10^{-12}, indicating that the doping profile was a zero of the RMSRE loss function. The relative error plot in Figure 4.32 shows that the optimizer successfully positioned the nulls in the relative error to lie exactly on the 100 mV bias increments used in the loss function. The resulting doping profile performed relatively poorly away from these bias increments, reaching a maximum error of nearly 1.4% at $V_F = 0.01$ V. To prevent this overfitting, the bias points could be sampled randomly instead of deterministically, or a larger number of bias points could be used in the loss function.

4.4.3 Two-dimensional diodes

Figures 4.34 through 4.37 show the results of the surrogatization process applied to 2D diodes. The fine-grained model `d_fine` and the coarse-grained model `d_coarse` were generated as shown below:

```
d_fine = diode2d(2.0,2.0,1.0,0.5,dx1=0.1,dx2=0.02,dx3=0.1,ny=9,rectgrid=true)
d_coarse = diode2d(2.0,2.0,1.0,0.5,dx1=0.8,dx2=0.16,dx3=0.8,rectgrid=true)
```

Rectangular grids were used so that `d_coarse.grid` would have a similar geometric structure to `d_fine.grid`. The resulting fine-grained grid has 747 nodes and 1312 cells; the coarse-grained grid has 55 nodes and 80 cells. The relative error of this unoptimized, coarse-grained model is similar to that of the 1D coarse-grained model described in Section 4.4.2. The peak error of the unoptimized 2D surrogate was 50.388% at $V_F = 0.09$ V.

Figures 4.34 and 4.35 were generated using 100 mV bias steps to compute loss. The resulting doping profile gives a peak error of 6.5107×10^{-3} at $V_F = 0.01$ V. This behavior is similar to the overfitting observed in the surrogatization of the 1D diode. Since no bias voltages below $V_F = 0.1$ V were considered in this loss function, the optimizer found a solution that gives low error at all

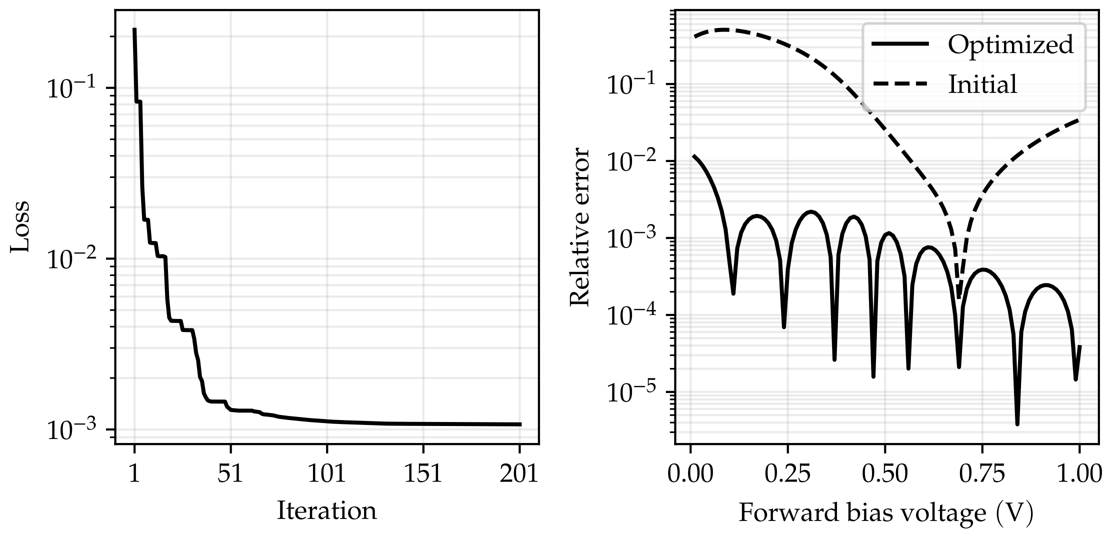


Figure 4.28: Loss progression showing training process for surrogate model of 1D diode (left); relative error between surrogate model and fine-grained model (right). Loss was computed using 100 mV bias steps.

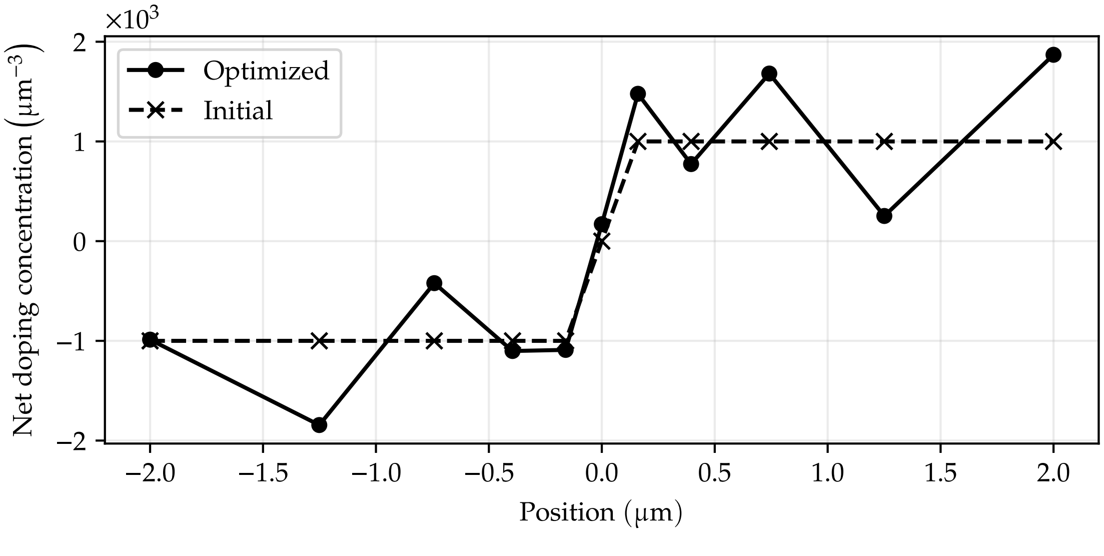


Figure 4.29: Optimal doping profile after 200 iterations of BFGS, using 100 mV bias steps to compute loss. The dashed line gives the doping profile of the unoptimized surrogate for reference.

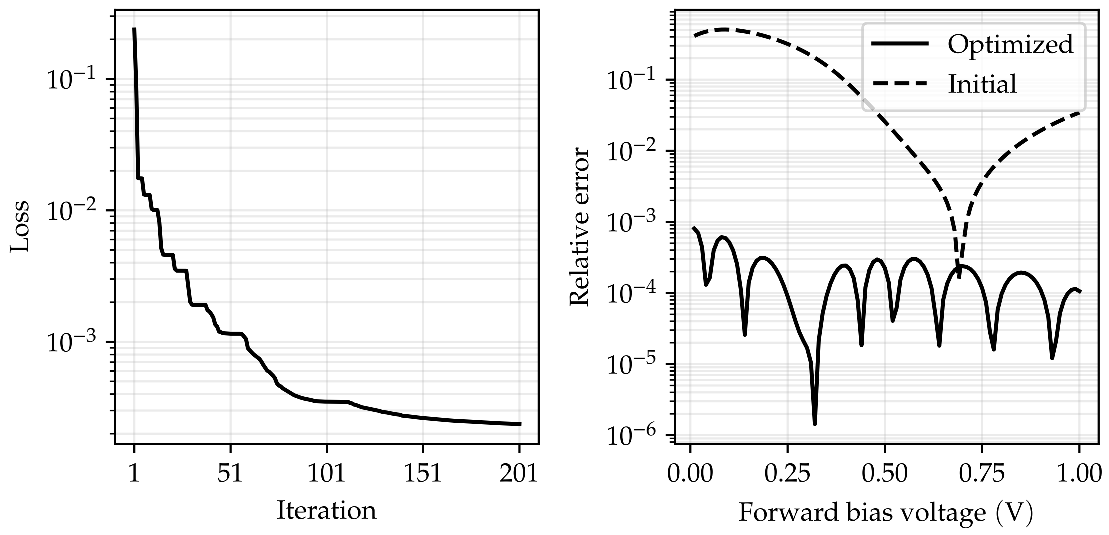


Figure 4.30: Loss progression showing training process for surrogate model of 1D diode (left); relative error between surrogate model and fine-grained model (right). Loss was computed using 10 mV bias steps.

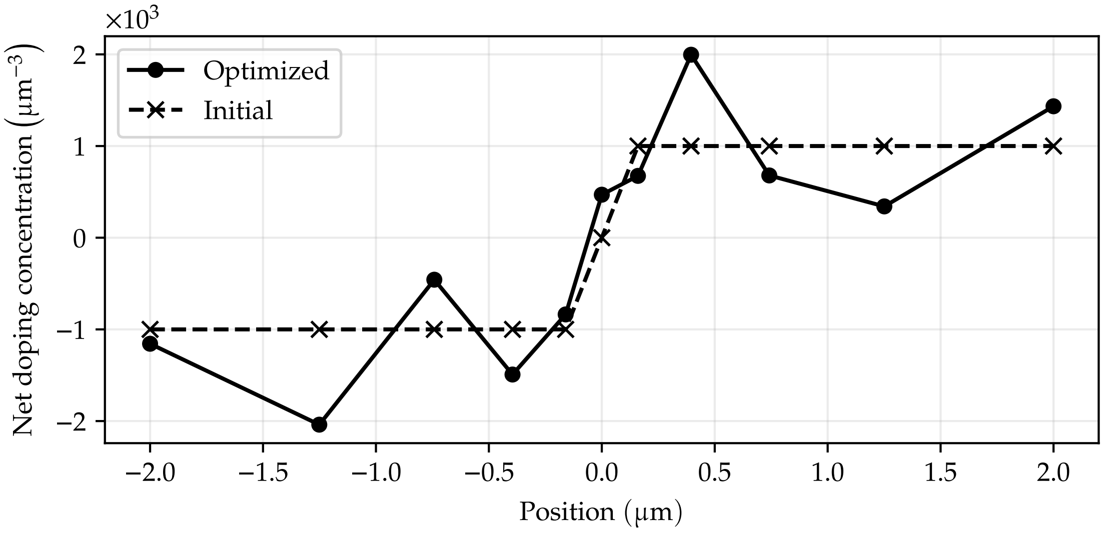


Figure 4.31: Optimal doping profile after 200 iterations of BFGS, using 10 mV bias steps to compute loss. The dashed line gives the doping profile of the unoptimized surrogate for reference.

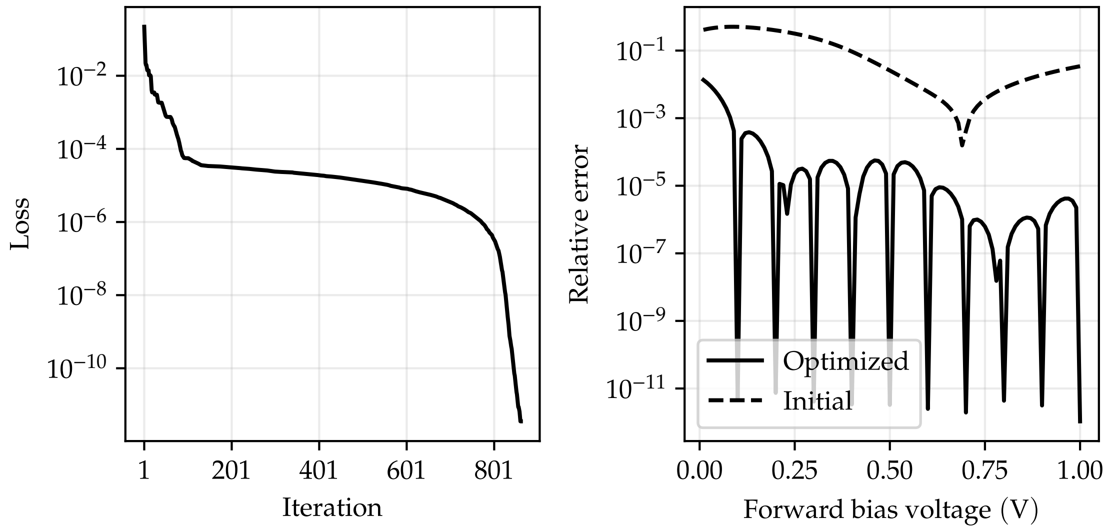


Figure 4.32: Loss progression showing overfitting in surrogate model of 1D diode after 862 iterations of BFGS (left); relative error between surrogate model and fine-grained model (right). Loss was computed using 10 mV bias steps.

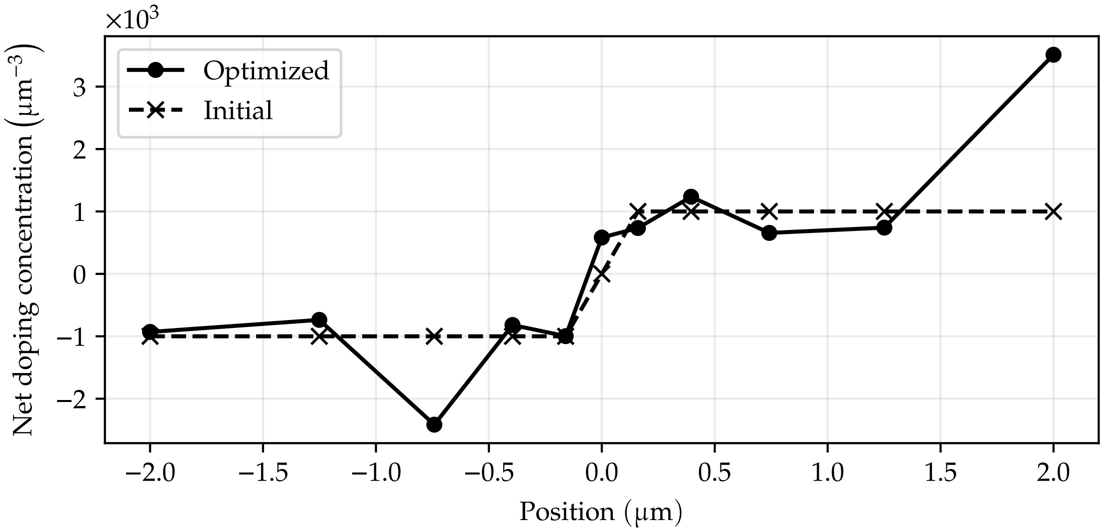


Figure 4.33: Optimal doping profile after 862 iterations of BFGS, using 10 mV bias steps to compute loss. The dashed line gives the doping profile of the unoptimized surrogate for reference.

bias voltages except those near $V_F = 0$. To resolve this, Figures 4.36 and 4.37 were generated using the same 100 mV bias steps, but the additional bias step $V_F = 0.01$ V was added to the bias vector to eliminate peaking in the error at low bias voltages. The resulting doping profile gives a peak error of 7.9327×10^{-4} at $V_F = 0.06$ V, which is nearly a factor of ten better than the doping profile generated without including $V_F = 0.01$ V.

The doping profiles shown in Figures 4.35 and 4.37 are qualitatively similar to the optimized 1D doping profile shown in Figure 4.31. All three profiles show a slightly increased p-doping near the anode contact, a highly increased n-doping to the right of the junction and an increased n-doping near the cathode contact. The 2D surrogate provides additional degrees of freedom for the optimizer, and the resulting 2D profiles vary slightly along the y axis. The doping profile shown in Figure 4.35 shows a larger heavily doped p-type region near the anode contact and a relatively more constant value in the y direction near the junction when compared to the profile shown in Figure 4.37. These changes result in the decreased error at low bias voltages shown in Figure 4.36.

In both bias step strategies discussed here, the BFGS optimizer was run with a maximum of 200 iterations. The optimizer terminated early in both cases due to insufficient decrease in the loss function between iterations. If lower error is desired, this early termination could potentially be avoided by choosing bias steps randomly to avoid convergence to a local minimum, or by using an optimization algorithm that attempts to find the global minimum of the objective function.

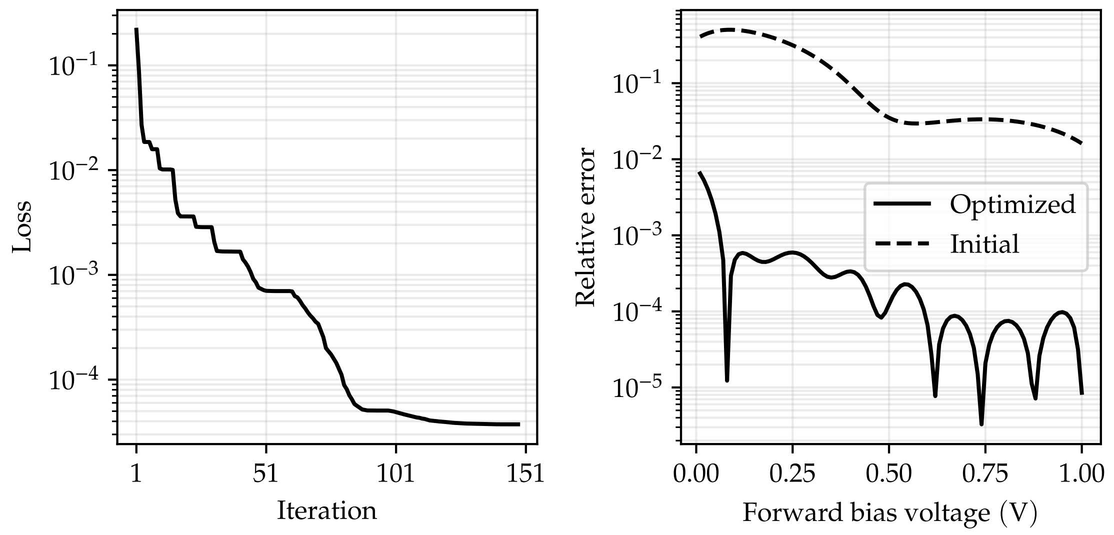


Figure 4.34: Loss progression showing training process for surrogate model of 2D diode (left); relative error between surrogate model and fine-grained model (right). Loss was computed using 100 mV bias steps.

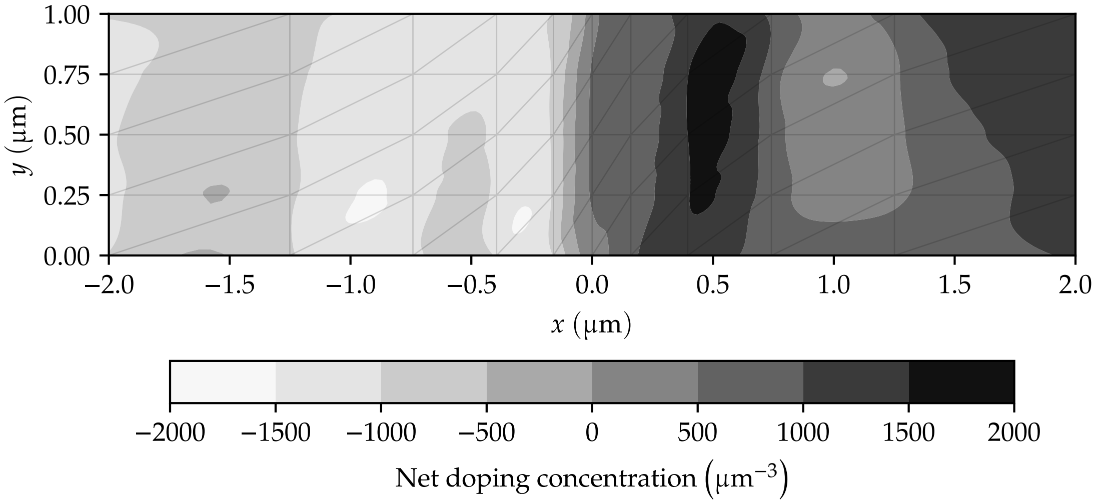


Figure 4.35: Optimal doping profile after 148 iterations of BFGS, using 100 mV bias steps to compute loss. A local minimum loss of 3.7437×10^{-5} was reached at termination.

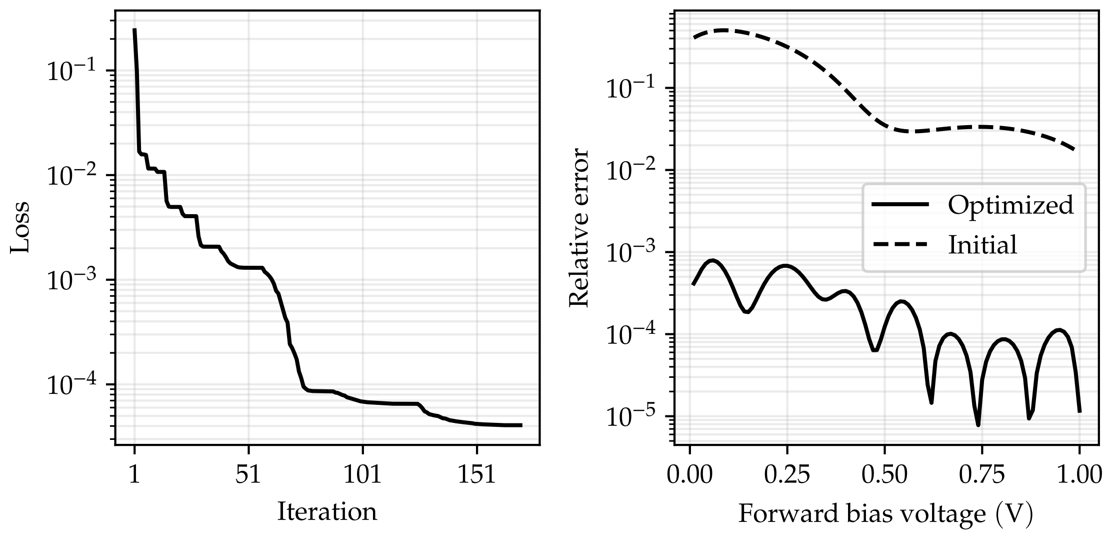


Figure 4.36: Loss progression showing training process for surrogate model of 2D diode (left); relative error between surrogate model and fine-grained model (right). Loss was computed using 100 mV bias steps, with the addition of $V_F = 0.01\,\text{V}$.

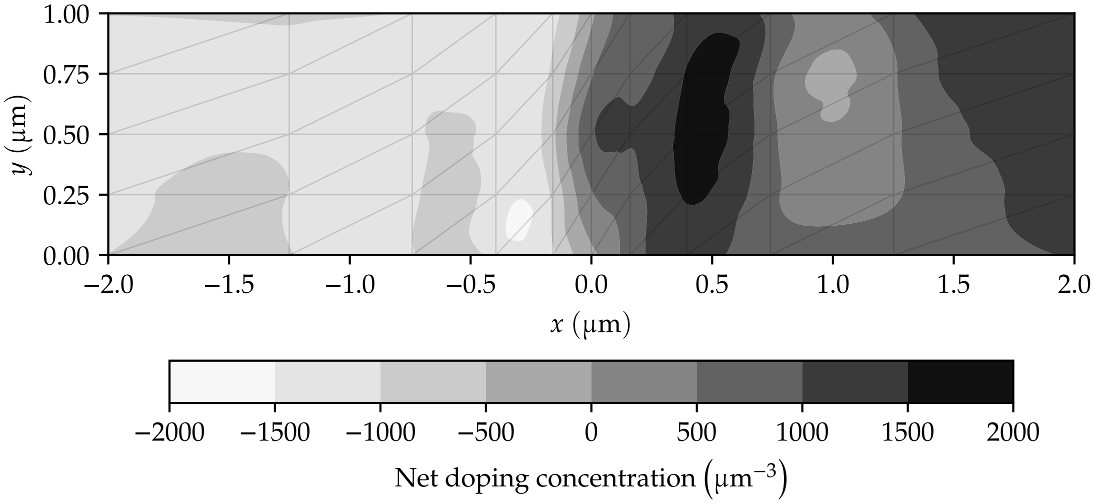


Figure 4.37: Optimal doping profile after 170 iterations of BFGS, using 100 mV bias steps to compute loss, with the addition of $V_F = 0.01\,\text{V}$. A local minimum loss of 4.0720×10^{-5} was reached at termination.

Chapter 5

Conclusion

In this work, we described the design, implementation and performance of surrogate models of silicon semiconductor devices. Chapter 2 discussed the Van Roosbroeck system, the PDE system governing carrier transport in semiconductors, and its finite difference and finite volume discretizations. Numerical methods for solving the discretized PDE system were discussed, including the damped Newton method. Automatic differentiation and scientific machine learning were discussed as key computational methods for training surrogates. A literature review was presented to survey related work in inverse design, surrogate modeling and doping profile optimization.

Chapter 3 introduced `Semiconductors.jl`, a new semiconductor simulation tool capable of performing 1D and 2D non-equilibrium simulations of semiconductor devices using the drift-diffusion model. The features of `Semiconductors.jl` were discussed, and documentation was provided for its public methods. A new vector discretization scheme was introduced to allow estimation of field magnitudes in a finite volume discretization via the least-squares approximation. A fully differentiable simulator was introduced that allows automatic differentiation of the entire PDE solver via `Zygote.jl`. TCAD models of 1D and 2D diodes, 2D BJTs and 2D MOSFETs were presented, and their implementation in `Semiconductors.jl` was discussed.

Chapter 4 presented experimental results on the characteristics and the surrogatization of device models implemented in `Semiconductors.jl`. Current-voltage characteristics were presented for various diode, BJT and MOSFET models. Scalar- and vector-valued quantities within the devices were analyzed to verify device operation. The condition number of the Jacobian generated by the 1D diode discretization was analyzed to determine the root cause of convergence errors near junction breakdown. Surrogate models were created and trained for the 1D and 2D diode models. The coarse-grained surrogate models for both devices achieved a maximum relative error below 1×10^{-3} using less than 10% as many nodes as the fine-grained baseline.

Future work on `Semiconductors.jl` could improve the physics models, the numerical methods and the computational performance of the simulator. This work assumed a simple drift-diffusion model and used the Boltzmann approximation for carrier statistics. A more general approach would allow arbitrary carrier distributions such as the Fermi-Dirac distribution, which more accurately models electron and hole concentrations in degenerate semiconductors. This would require the use of quasi-Fermi potentials in the discretization, since n and p could no longer be expressed in closed form with respect to the potential. The drift-diffusion model could also be extended to include lattice and carrier temperatures to allow modeling of effects like self heating and hot carrier injection. Particle-based or Monte Carlo methods could be implemented to improve simulation accuracy at deep sub-micron scales.

More device models could be implemented to test the performance of the surrogatization process. In particular, the finite volume discretization discussed here can be generalized to a boundary conforming Delaunay tetrahedralization of a 3D geometry. This generalization would allow simulation of complex 3D devices like FinFETs, nanowire FETs and other emerging transistor structures. Larger, circuit-level models could also be considered, allowing surrogatization of multiple active devices at once. A 3D simulation would also allow the simultaneous solution of Maxwell's equations for the magnetic and electric fields. This would allow accurate time-dependent simulation and surrogatization of devices operating at high frequency. Time dependence can be included in the finite volume discretization using the `storage` callback to `VoronoiFVM.System`.

This work relied heavily on Newton's method as a nonlinear solver for the discretized PDE systems. While Newton's method can be quadratically convergent with suitable initial conditions, it often fails to converge when the initial condition is far from the solution. Specifically, convergence may fail if the bias step between two points on an I-V curve is too large, or if the desired bias point is near a turning point. Some TCAD packages use Gummel's method, an uncoupled nonlinear solver, to generate a rough solution before applying Newton's method. Since Gummel's method is less sensitive to initial conditions, this scheme can give convergence in situations where Newton's method is difficult or impossible to apply. Homotopy embedding could also be implemented to aid convergence in numerically difficult situations [131].

Additional work is necessary to enable simulation of multivalued I-V curves using arc-length continuation. The continuation algorithm implemented in `Semiconductors.jl` performs well for single-valued I-V curves but fails to converge when a turning point is reached. This may be due to the scaling of the current and voltage variables or to the nonlinear solver itself. A scaling scheme similar to that used in Padre TCAD could be implemented, where minimum and maximum values for the current and voltage are specified for each run, and the bias voltages and terminal currents are normalized by those minimum and maximum values. An alternative nonlinear solver such as Gummel's method could be implemented to provide initial conditions for the damped Newton solver near turning points. In the unlikely case that the linear solve computing the Newton step is preventing convergence, a preconditioner method could be implemented to increase the accuracy of the linear solve.

The differentiable simulator implemented in this work uses `Zygote.Buffer` to accumulate gradients during assembly of the discretized PDE system. Performance profiling shows that this accumulation comprises most of the runtime of the surrogate training process. This performance could likely be improved using a custom pullback defined by, for example, `ChainRulesCore.jl`. Performance could be further improved through parallelization. Currently, the I-V curve tracing routines and all surrogate optimization loops are executed on a single thread. It may be possible to accelerate the curve tracing by computing multiple bias points at once, provided a suitable initial condition can be generated for each concurrent simulation.

Appendix A

Derivations

A.1 Scharfetter-Gummel midpoint scheme

This appendix contains details regarding the Scharfetter-Gummel midpoint scheme. In this work, the *midpoint scheme* refers to the use of the values of n and p at the midpoint of grid segments, as opposed to the traditional scheme that uses the value of x for which the finite difference approximations of n' and p' hold.

A.1.1 Derivation

Substituting (2.38) and (2.39) into (2.36) gives

$$\begin{aligned} n(x) &= n[k]\left(1 - \frac{1}{1-\exp(\delta_r)} - \frac{\exp\left(\delta_r \frac{x}{x[k+1]-x[k]}\right)}{\exp\left(\delta_r \frac{x[k+1]}{x[k+1]-x[k]}\right) - \exp\left(\delta_r \frac{x[k]}{x[k+1]-x[k]}\right)}\right) + \\ &\qquad n[k+1]\left(\frac{1}{1-\exp(\delta_r)} - \frac{\exp\left(\delta_r \frac{x}{x[k+1]-x[k]}\right)}{\exp\left(\delta_r \frac{x[k+1]}{x[k+1]-x[k]}\right) - \exp\left(\delta_r \frac{x[k]}{x[k+1]-x[k]}\right)}\right) \\ &= n[k]\left(1 - \frac{1-\exp\left(\delta_r \frac{x-x[k]}{x[k+1]-x[k]}\right)}{1-\exp(\delta_r)}\right) + n[k+1]\left(\frac{1-\exp\left(\delta_r \frac{x-x[k]}{x[k+1]-x[k]}\right)}{1-\exp(\delta_r)}\right), \end{aligned} \tag{A.1}$$

where δ_r is given in (2.42). We may now obtain $n[k+1/2]$ by substituting $x = (x[k] + x[k+1])/2$ in (A.1), which gives

$$\begin{aligned} n\left[k+\frac{1}{2}\right] &= n[k]\left(1 - \frac{1-\exp(\delta_r/2)}{1-\exp(\delta_r)}\right) + n[k+1]\left(\frac{1-\exp(\delta_r/2)}{1-\exp(\delta_r)}\right) \\ &= (1-\sigma(\delta_r))\, n[k] + \sigma(\delta_r) n[k+1], \end{aligned} \tag{A.2}$$

where $\sigma(x)$ is the sigmoid function

$$\sigma(x) = \frac{1-e^{x/2}}{1-e^{x}} = \frac{1}{1+e^{x/2}}. \tag{A.3}$$

Similar reasoning gives the value of n at the midpoint of $x[k-1]$ and $x[k]$:

$$n\left[k-\frac{1}{2}\right] = (1-\sigma(\delta_l))\, n[k-1] + \sigma(\delta_l) n[k], \tag{A.4}$$

where δ_l is given in (2.42). Substituting (A.2) into (2.33),

$$\frac{J_n[k+1/2]}{q} = \\ \frac{\mu_n[k+1/2]V_T}{x[k+1]-x[k]}\left(((1-\sigma(\delta_r))n[k]+\sigma(\delta_r)n[k+1])\,\frac{\psi[k]-\psi[k+1]}{V_T}+n[k+1]-n[k]\right). \quad \text{(A.5)}$$

Using a similar approach to compute $J_n[k-1/2]$ from (A.4),

$$\frac{J_n[k-1/2]}{q} = \\ \frac{\mu_n[k-1/2]V_T}{x[k]-x[k-1]}\left(((1-\sigma(\delta_l))n[k-1]+\sigma(\delta_l)n[k])\,\frac{\psi[k-1]-\psi[k]}{V_T}+n[k]-n[k-1]\right). \quad \text{(A.6)}$$

Substituting (A.5) and (A.6) into (2.32),

$$\frac{J_n'[k]}{q} = \frac{2}{x[k+1]-x[k-1]}\left(c_1 n[k-1]+c_2 n[k]+c_3 n[k+1]\right), \quad \text{(A.7)}$$

where

$$\begin{aligned} c_1 &= \frac{\mu_n[k-1/2]V_T}{x[k]-x[k-1]}\left(\delta_l(1-\sigma(\delta_l))+1\right), \\ c_2 &= -\frac{\mu_n[k-1/2]V_T}{x[k]-x[k-1]}\left(\delta_l\sigma(\delta_l)-1\right)-\frac{\mu_n[k+1/2]V_T}{x[k+1]-x[k]}\left(\delta_r(1-\sigma(\delta_r))+1\right), \\ c_3 &= -\frac{\mu_n[k+1/2]V_T}{x[k+1]-x[k]}\left(\delta_r\sigma(\delta_r)-1\right). \end{aligned}$$

Equation (A.7) can be used to compute the electron current residual $F_n[k]$, and an analogous process can be used for holes:

$$\begin{aligned} F_n[k] = G[k]-U[k]+\frac{2}{x[k+1]-x[k-1]}\Bigg(& \\ & \frac{\mu_n[k-1/2]V_T}{x[k]-x[k-1]}\Big(\,(1+\delta_l(1-\sigma(\delta_l)))n[k-1]-(1-\delta_l\sigma(\delta_l))n[k]\,\Big)- \\ & \quad \frac{\mu_n[k+1/2]V_T}{x[k+1]-x[k]}\Big(\,(1+\delta_r(1-\sigma(\delta_r)))n[k]-(1-\delta_r\sigma(\delta_r))n[k+1]\,\Big)\Bigg)=0, \quad \text{(A.8)} \end{aligned}$$

$$\begin{aligned} F_p[k] = G[k]-U[k]+\frac{2}{x[k+1]-x[k-1]}\Bigg(& \\ & \frac{\mu_p[k-1/2]V_T}{x[k]-x[k-1]}\Big(\,(1-\delta_l\sigma(\delta_l))p[k-1]-(1+\delta_l(1-\sigma(\delta_l)))p[k]\,\Big)- \\ & \quad \frac{\mu_p[k+1/2]V_T}{x[k+1]-x[k]}\Big(\,(1-\delta_r\sigma(\delta_r))p[k]-(1+\delta_r(1-\sigma(\delta_r)))p[k+1]\,\Big)\Bigg)=0. \quad \text{(A.9)} \end{aligned}$$

The Poisson residual in (2.52) can be used along with (A.8) and (A.9) to provide a complete discretization.

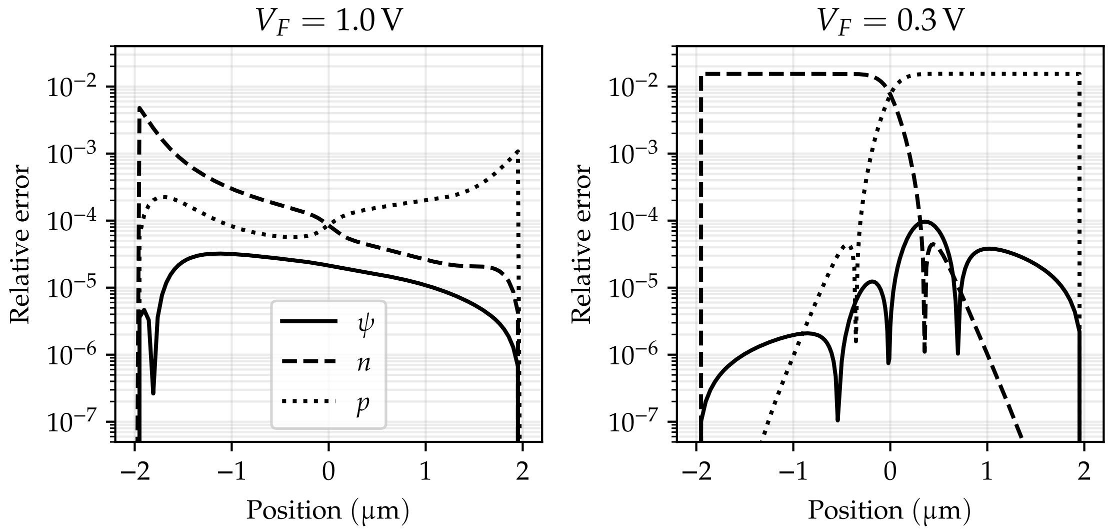


Figure A.1: Relative error between midpoint scheme and Scharfetter-Gummel discretization in a 1D diode simulation under heavy forward bias ($V_F = 1.0\,\mathrm{V}$, left) and light forward bias ($V_F = 0.3\,\mathrm{V}$, right).

A.1.2 Comparison

The residuals given in (A.8) and (A.9) are similar to those in (2.53) and (2.54). In the traditional Scharfetter-Gummel discretization, the Bernoulli function $B(x)$ is used to determine the weights of, say, $n[k-1]$ and $n[k]$ in F_n. In the midpoint scheme, the function $1 - x\sigma(x)$ is used instead, which has identical behavior for small x but deviates from the Bernoulli function for large x. Similarly, the traditional Scharfetter-Gummel discretization uses $Q(x)$ to interpolate between $n[k-1]$ and $n[k]$ in the determination of $n[k-1/2]$, while the midpoint scheme uses $\sigma(x)$. The behaviors of these four functions are compared in Figure A.2. The difference between the interpolation functions is

$$\sigma(x) - Q(x) = \frac{1}{1+e^{x/2}} - \frac{1}{x} + \frac{1}{e^x - 1} = \frac{1}{2}\operatorname{csch}\left(\frac{x}{2}\right) - \frac{1}{x}.$$

The difference between the weight functions is

$$1 - x\sigma(x) - B(x) = 1 - \frac{x}{1+e^{x/2}} - \frac{x}{e^x - 1} = 1 - \frac{x}{2}\operatorname{csch}\left(\frac{x}{2}\right).$$

These differences are shown in Figure A.3.

Since the two weight functions are nearly identical for small x, the impact of the midpoint scheme on the overall simulation is typically negligible. Figure A.1 shows the relative error between a simulation conducted using the midpoint scheme and a simulation conducted using the traditional Scharfetter-Gummel discretization. The simulation was run using the default 1D diode model, resulting in a grid with 163 nodes. The diode was simulated with a forward bias of $V_F = 1.0\,\mathrm{V}$ and $V_F = 0.3\,\mathrm{V}$, corresponding to drift-dominant and diffusion-dominant currents, respectively. In the drift-dominant scenario, the error in ψ is considerably smaller than the error in the carrier concentrations and is relatively constant along the device. The error in n and p is larger, peaking at nearly 1% near the left contact. In the diffusion-dominant scenario, the error in n and p is large along the device, with the error in the majority carrier concentration reaching nearly 2% in each doped region.

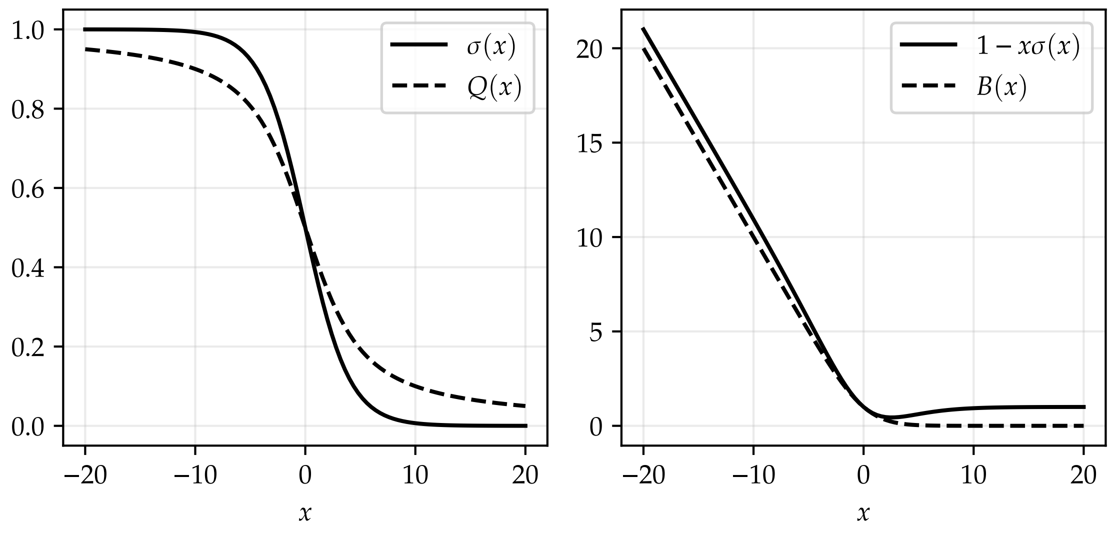


Figure A.2: Comparison of interpolation functions $Q(x)$ and $\sigma(x)$ (left); comparison of weight functions $B(x)$ and $1 - x\sigma(x)$ (right), for $x \in (-20, 20)$.

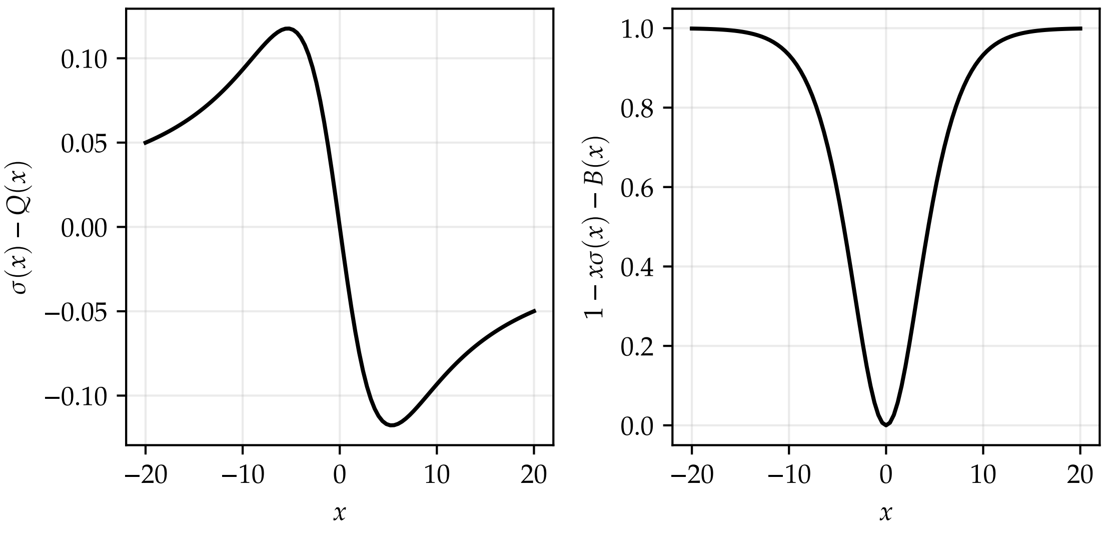


Figure A.3: Difference between interpolation functions (left) and weight functions (right) in classical Scharfetter-Gummel discretization versus midpoint scheme discretization.

A.2 Jacobian of 1D finite difference discretization

In this section, we derive the system Jacobian for the residuals given in (2.52) through (2.54). We assume that $G[k] = 0$, that $U[k]$ is only a function of $n[k]$ and $p[k]$, that μ_n and μ_p are piecewise constant and that Dirichlet boundary conditions are enforced without the use of penalty methods. We further assume that only the Shockley-Read-Hall recombination model is used and that τ_n and τ_p are constant along the device, i.e. $U[k] = U_{SRH}[k]$, where U_{SRH} is given in (2.15). Nonzero generation rates and field-dependent mobilities create additional nonzero entries in the Jacobian and are not considered here.

A.2.1 Derivation

The nonzero derivatives of $F_\psi[k]$ are

$$\begin{aligned}
\frac{\partial F_\psi[k]}{\partial \psi[k-1]} &= \frac{2\varepsilon[k-1/2]}{(x[k+1]-x[k-1])\,(x[k]-x[k-1])},\\
\frac{\partial F_\psi[k]}{\partial \psi[k]} &= -\frac{2}{x[k+1]-x[k-1]}\left(\frac{\varepsilon[k-1/2]}{x[k]-x[k-1]}+\frac{\varepsilon[k+1/2]}{x[k+1]-x[k]}\right),\\
\frac{\partial F\psi[k]}{\partial \psi[k+1]} &= \frac{2\varepsilon[k+1/2]}{(x[k+1]-x[k-1])\,(x[k+1]-x[k])},\\
\frac{\partial f\psi[k]}{\partial n[k]} &= -q,\\
\frac{\partial f\psi[k]}{\partial p[k]} &= q.
\end{aligned}$$

The nonzero derivatives of $F_n[k]$ are

$$\begin{aligned}
\frac{\partial F_n[k]}{\partial \psi[k-1]} &= \frac{2\mu_n[k-1/2]}{(x[k+1]-x[k-1])\,(x[k]-x[k-1])}\left(B'(-\delta_l)n[k-1]+B'(\delta_l)n[k]\right),\\
\frac{\partial F_n[k]}{\partial \psi[k]} &= -\frac{2\mu_n[k-1/2]}{(x[k+1]-x[k-1])\,(x[k]-x[k-1])}\left(B'(-\delta_l)n[k-1]+B'(\delta_l)n[k]\right)-\\
&\qquad \frac{2\mu_n[k+1/2]}{(x[k+1]-x[k-1])\,(x[k+1]-x[k])}\left(B'(-\delta_r)n[k]+B'(\delta_r)n[k+1]\right),\\
\frac{\partial F_n[k]}{\partial \psi[k+1]} &= \frac{2\mu_n[k+1/2]}{(x[k+1]-x[k-1])\,(x[k+1]-x[k])}\left(B'(-\delta_r)n[k]+B'(\delta_r)n[k+1]\right),
\end{aligned}$$

$$\begin{aligned}
\frac{\partial F_n[k]}{\partial n[k-1]} &= \frac{2\mu_n[k-1/2]V_T B(-\delta_l)}{(x[k+1]-x[k-1])\,(x[k]-x[k-1])},\\
\frac{\partial F_n[k]}{\partial n[k]} &= -\frac{2V_T}{x[k+1]-x[k-1]}\left(\frac{\mu_n[k-1/2]}{x[k]-x[k-1]}B(\delta_l)+\frac{\mu_n[k+1/2]}{x[k+1]-x[k]}B(-\delta_r)\right)-\frac{\partial U[k]}{\partial n[k]},\\
\frac{\partial F_n[k]}{\partial n[k+1]} &= \frac{2\mu_n[k+1/2]V_T B(\delta_r)}{(x[k+1]-x[k-1])\,(x[k+1]-x[k])},\\
\frac{\partial F_n[k]}{\partial p[k]} &= -\frac{\partial U[k]}{\partial p[k]}.
\end{aligned}$$

The nonzero derivatives of $F_p[k]$ are

$$\begin{aligned}
\frac{\partial F_p[k]}{\partial\psi[k-1]} &= -\frac{2\mu_p[k-1/2]}{(x[k+1]-x[k-1])\,(x[k]-x[k-1])}\left(\,B'(\delta_l)p[k-1]+B'(-\delta_l)p[k]\,\right),\\
\frac{\partial F_p[k]}{\partial\psi[k]} &= \frac{2\mu_p[k-1/2]}{(x[k+1]-x[k-1])\,(x[k]-x[k-1])}\left(\,B'(\delta_l)p[k-1]+B'(-\delta_l)p[k]\,\right)+\\
&\qquad \frac{2\mu_p[k+1/2]}{(x[k+1]-x[k-1])\,(x[k+1]-x[k])}\left(\,B'(\delta_r)p[k]+B'(-\delta_r)p[k+1]\,\right),\\
\frac{\partial F_p[k]}{\partial\psi[k+1]} &= -\frac{2\mu_p[k+1/2]}{(x[k+1]-x[k-1])\,(x[k+1]-x[k])}\left(\,B'(\delta_r)p[k]+B'(-\delta_r)p[k+1]\,\right),
\end{aligned}$$

$$\begin{aligned}
\frac{\partial F_p[k]}{\partial n[k]} &= -\frac{\partial U[k]}{\partial n[k]},\\
\frac{\partial F_p[k]}{\partial p[k-1]} &= \frac{2\mu_p[k-1/2]V_T B(\delta_l)}{(x[k+1]-x[k-1])\,(x[k]-x[k-1])},\\
\frac{\partial F_p[k]}{\partial p[k]} &= -\frac{2V_T}{x[k+1]-x[k-1]}\left(\frac{\mu_p[k-1/2]}{x[k]-x[k-1]}B(-\delta_l)+\frac{\mu_p[k+1/2]}{x[k+1]-x[k]}B(\delta_r)\right)-\frac{\partial U[k]}{\partial p[k]},\\
\frac{\partial F_p[k]}{\partial p[k+1]} &= \frac{2\mu_p[k+1/2]V_T B(-\delta_r)}{(x[k+1]-x[k-1])\,(x[k+1]-x[k])}.
\end{aligned}$$

In the expressions above, the function $B'(x)$ is the derivative of the Bernoulli function,

$$B'(x)=\frac{e^x-1-xe^x}{(e^x-1)^2}=\frac{1}{e^x-1}-\frac{xe^x}{(e^x-1)^2}. \tag{A.10}$$

The Taylor expansion of $B'(x)$ at $x=0$ can be found by differentiation of (2.50):

$$B'(x)=-\frac{1}{2}+\frac{x}{6}-\frac{x^3}{180}+\frac{x^5}{5040}-\frac{x^7}{151\,200}+\frac{x^9}{4\,790\,016}-\cdots.$$

The derivatives of the recombination function are

$$\frac{\partial U[k]}{\partial n[k]}=\frac{(p[k]+n_i)\left(n_i\tau_p+p[k]\tau_n\right)}{\left((n[k]+n_i)\,\tau_p+(p[k]+n_i)\,\tau_n\right)^2},\quad \frac{\partial U[k]}{\partial p[k]}=\frac{(n[k]+n_i)\left(n[k]\tau_p+n_i\tau_n\right)}{\left((n[k]+n_i)\,\tau_p+(p[k]+n_i)\,\tau_n\right)^2}. \tag{A.11}$$

A.2.2 Sparsity

The system Jacobian is a block diagonal matrix comprised of entries $\mathbf{J}[k]\in\mathbb{R}^{3\times 9}$, where

$$\mathbf{J}[k]=\begin{bmatrix}
\frac{\partial F_\psi[k]}{\partial\psi[k-1]} & 0 & 0 & \frac{\partial F_\psi[k]}{\partial\psi[k]} & \frac{\partial F_\psi[k]}{\partial n[k]} & \frac{\partial F_\psi[k]}{\partial p[k]} & \frac{\partial F_\psi[k]}{\partial\psi[k+1]} & 0 & 0\\
\frac{\partial F_n[k]}{\partial\psi[k-1]} & \frac{\partial F_n[k]}{\partial n[k-1]} & 0 & \frac{\partial F_n[k]}{\partial\psi[k]} & \frac{\partial F_n[k]}{\partial n[k]} & \frac{\partial F_n[k]}{\partial p[k]} & \frac{\partial F_n[k]}{\partial\psi[k+1]} & \frac{\partial F_n[k]}{\partial n[k+1]} & 0\\
\frac{\partial F_p[k]}{\partial\psi[k-1]} & 0 & \frac{\partial F_p[k]}{\partial p[k-1]} & \frac{\partial F_p[k]}{\partial\psi[k]} & \frac{\partial F_p[k]}{\partial n[k]} & \frac{\partial F_p[k]}{\partial p[k]} & \frac{\partial F_p[k]}{\partial\psi[k+1]} & 0 & \frac{\partial F_p[k]}{\partial p[k+1]}
\end{bmatrix}$$

and $k\in[2,N-1]$. Dirichlet boundary conditions hold when $k=1$ or $k=N$, and

$$\mathbf{J}[1]=\mathbf{J}[N]=\begin{bmatrix}\mathbf{I}_3 & \mathbf{0}_{3,6}\end{bmatrix}.$$

The overall Jacobian has the form

$$\mathbf{J}=\operatorname{diag}\left(\,\mathbf{J}[1],\mathbf{J}[2],\cdots,\mathbf{J}[N-1],\mathbf{J}[N]\,\right),$$

where each entry is offset from the previous entry by 3 columns.

A.3 Jacobian of finite volume discretization

In this section, we derive the system Jacobian for the finite volume residual given in (2.62) using the discretizations given in (2.63) through (2.65). This Jacobian is manually assembled by `assemble_res_jac!` in the differentiable simulator.

We assume that $G[k] = 0$, that $U[k]$ is only a function of $n[k]$ and $p[k]$, that μ_n and μ_p are piecewise constant and that Dirichlet boundary conditions are enforced without the use of penalty methods. We further assume that only the Shockley-Read-Hall recombination model is used and that τ_n and τ_p are constant along the device, i.e. $U[k] = U_{SRH}[k]$, where U_{SRH} is given in (2.15). Nonzero generation rates and field-dependent mobilities create additional nonzero entries in the Jacobian and are not considered here.

A.3.1 Residuals

The finite volume residuals are

$$\begin{aligned}
F_\psi[k] &= \sum_{l\in\mathcal{N}[k]} \varepsilon[k,l]\left(\frac{|\partial\omega_k \cap \partial\omega_l|}{\|\mathbf{x}[l]-\mathbf{x}[k]\|}\right)\Big(\,\psi[k]-\psi[l]\,\Big) + q|\omega_k|\Big(\,n[k]-p[k]-\Gamma[k]\,\Big),\\
F_n[k] &= \sum_{l\in\mathcal{N}[k]} \mu_n[k,l]V_T\left(\frac{|\partial\omega_k \cap \partial\omega_l|}{\|\mathbf{x}[l]-\mathbf{x}[k]\|}\right)\left(B\left(-\frac{\psi[l]-\psi[k]}{V_T}\right)n[k] - B\left(\frac{\psi[l]-\psi[k]}{V_T}\right)n[l]\right)+\\
&\qquad |\omega_k|\Big(\,U[k]-G[k]\,\Big),\\
F_p[k] &= \sum_{l\in\mathcal{N}[k]} \mu_p[k,l]V_T\left(\frac{|\partial\omega_k \cap \partial\omega_l|}{\|\mathbf{x}[l]-\mathbf{x}[k]\|}\right)\left(B\left(\frac{\psi[l]-\psi[k]}{V_T}\right)p[k] - B\left(-\frac{\psi[l]-\psi[k]}{V_T}\right)p[l]\right)+\\
&\qquad |\omega_k|\Big(\,U[k]-G[k]\,\Big).
\end{aligned}$$

A.3.2 Derivation

The nonzero derivatives of $F_\psi[k]$ are

$$\begin{aligned}
\frac{\partial F_\psi[k]}{\partial\psi[k]} &= \sum_{l\in\mathcal{N}[k]} \varepsilon[k,l]\left(\frac{|\partial\omega_k \cap \partial\omega_l|}{\|\mathbf{x}[l]-\mathbf{x}[k]\|}\right),\\
\frac{\partial F_\psi[k]}{\partial\psi[l]} &= -\varepsilon[k,l]\left(\frac{|\partial\omega_k \cap \partial\omega_l|}{\|\mathbf{x}[l]-\mathbf{x}[k]\|}\right), \quad \text{where} \quad l\in\mathcal{N}[k],\\
\frac{\partial F_\psi[k]}{\partial n[k]} &= q|\omega_k|,\\
\frac{\partial F_\psi[k]}{\partial p[k]} &= -q|\omega_k|.
\end{aligned}$$

The nonzero derivatives of $F_n[k]$ are

$$\begin{aligned}
\frac{\partial F_n[k]}{\partial\psi[k]} &= \sum_{l\in\mathcal{N}[k]} \mu_n[k,l]\left(\frac{|\partial\omega_k \cap \partial\omega_l|}{\|\mathbf{x}[l]-\mathbf{x}[k]\|}\right)\left(B'\left(-\frac{\psi[l]-\psi[k]}{V_T}\right)n[k] + B'\left(\frac{\psi[l]-\psi[k]}{V_T}\right)n[l]\right),\\
\frac{\partial F_n[k]}{\partial\psi[l]} &= -\mu_n[k,l]\left(\frac{|\partial\omega_k \cap \partial\omega_l|}{\|\mathbf{x}[l]-\mathbf{x}[k]\|}\right)\left(B'\left(-\frac{\psi[l]-\psi[k]}{V_T}\right)n[k] + B'\left(\frac{\psi[l]-\psi[k]}{V_T}\right)n[l]\right), \quad \text{where} \quad l\in\mathcal{N}[k],
\end{aligned}$$

$$
\begin{aligned}
\frac{\partial F_n[k]}{\partial n[k]} &= \sum_{l\in\mathcal{N}[k]} \mu_n[k,l]V_T\left(\frac{|\partial\omega_k\cap\partial\omega_l|}{\|\mathbf{x}[l]-\mathbf{x}[k]\|}\right)B\left(-\frac{\psi[l]-\psi[k]}{V_T}\right)+|\omega_k|\left(\frac{\partial U[k]}{\partial n[k]}\right),\\
\frac{\partial F_n[k]}{\partial n[l]} &= -\mu_n[k,l]V_T\left(\frac{|\partial\omega_k\cap\partial\omega_l|}{\|\mathbf{x}[l]-\mathbf{x}[k]\|}\right)B\left(\frac{\psi[l]-\psi[k]}{V_T}\right), \quad \text{where} \quad l\in\mathcal{N}[k],\\
\frac{\partial F_n[k]}{\partial p[k]} &= |\omega_k|\left(\frac{\partial U[k]}{\partial p[k]}\right).
\end{aligned}
$$

The nonzero derivatives of $F_p[k]$ are

$$
\begin{aligned}
\frac{\partial F_p[k]}{\partial \psi[k]} &= -\sum_{l\in\mathcal{N}[k]} \mu_p[k,l]\left(\frac{|\partial\omega_k\cap\partial\omega_l|}{\|\mathbf{x}[l]-\mathbf{x}[k]\|}\right)\left(B'\left(\frac{\psi[l]-\psi[k]}{V_T}\right)p[k]+B'\left(-\frac{\psi[l]-\psi[k]}{V_T}\right)p[l]\right),\\
\frac{\partial F_p[k]}{\partial \psi[l]} &= \mu_p[k,l]\left(\frac{|\partial\omega_k\cap\partial\omega_l|}{\|\mathbf{x}[l]-\mathbf{x}[k]\|}\right)\left(B'\left(\frac{\psi[l]-\psi[k]}{V_T}\right)p[k]+B'\left(-\frac{\psi[l]-\psi[k]}{V_T}\right)p[l]\right), \quad \text{where} \quad l\in\mathcal{N}[k],
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial F_p[k]}{\partial n[k]} &= |\omega_k|\left(\frac{\partial U[k]}{\partial n[k]}\right),\\
\frac{\partial F_p[k]}{\partial p[k]} &= \sum_{l\in\mathcal{N}[k]} \mu_p[k,l]V_T\left(\frac{|\partial\omega_k\cap\partial\omega_l|}{\|\mathbf{x}[l]-\mathbf{x}[k]\|}\right)B\left(\frac{\psi[l]-\psi[k]}{V_T}\right)+|\omega_k|\left(\frac{\partial U[k]}{\partial p[k]}\right),\\
\frac{\partial F_p[k]}{\partial p[l]} &= -\mu_p[k,l]V_T\left(\frac{|\partial\omega_k\cap\partial\omega_l|}{\|\mathbf{x}[l]-\mathbf{x}[k]\|}\right)B\left(-\frac{\psi[l]-\psi[k]}{V_T}\right), \quad \text{where} \quad l\in\mathcal{N}[k].
\end{aligned}
$$

The derivative of the Bernoulli function, $B'(x)$, is given in (A.10), and the derivatives of the recombination function are given in (A.11).

The above expressions hold at all nodes in the discretization except the nodes belonging to boundary regions corresponding to ideal Ohmic contacts. For such boundary nodes, we have

$$
\frac{\partial F_\psi[k]}{\partial\psi[k]} = \frac{\partial F_n[k]}{\partial n[k]} = \frac{\partial F_p[k]}{\partial p[k]} = 1
$$

assuming Dirichlet boundary conditions are imposed without penalty. All other partial derivatives in these rows of the Jacobian are 0.

Appendix B

Additional results

This appendix presents additional experimental results on the characteristics of the device models implemented in `Semiconductors.jl` and of their solutions. The results in this appendix are mainly illustrative and serve to reinforce the key results presented in Chapter 4.

B.1 Diodes

Figure B.1 shows the log-magnitude of Jacobian matrices of the one-dimensional diode generated by `diode1d`. The diode used in this figure uses an 11-point discretization grid to allow visualization of the sparsity pattern of the matrix. The matrices assembled using Dirichlet boundary conditions with a penalty factor of unity. The sparsity pattern described in Section A.2.2 is clearly visible in this figure. The resulting Jacobian has size 33×33 since each of the 11 nodes is associated with unknown values of ψ, n and p.

In Figure B.1, the partial derivatives of F_ψ are several orders of magnitude smaller than those of F_n or F_p, in agreement with the results presented in Section A.2.2. The derivatives of F_ψ are proportional to the quantities q and ε_0, which both have values on the order of 1×10^{-19} to 1×10^{-18} in the units used in `Semiconductors.jl`. The residuals of $n[k]$ and $p[k]$ are strongly coupled to $\psi[k]$ when the device is forward biased due to the nonlinear upwinding introduced by the Scharfetter-Gummel discretization.

Figure B.2 shows the potential, electron density and hole density in the two-dimensional diode generated by `diode2d(2.0,2.0,1.0,0.5,rectgrid=true)`. This simulation is identical to the one shown in Figure 4.12, except a rectangular discretization grid is used instead of a triangular grid. This rectangular grid has 285 nodes and 448 cells (triangles), compared to 459 nodes and 825 cells in the triangular grid shown in Figure 4.12. This rectangular grid can yield faster simulations when limited variation along the y axis is expected; in this case, the diode model is nearly 1D, so the rectangular grid using only 5 nodes in the y direction is sufficient to simulate ψ, n and p in both dimensions.

B.2 Bipolar transistors

Figure B.3 shows the potential, electron density and hole density in the NPN BJT generated by `npn1(5,1,10,5,3,3,0.2,rectgrid=true)`. This simulation is identical to the one shown in Figure 4.17, except a rectangular discretization grid is used instead of a triangular grid. This rectangular grid has 240 nodes and 390 cells (triangles), compared to 257 nodes and 461 cells in

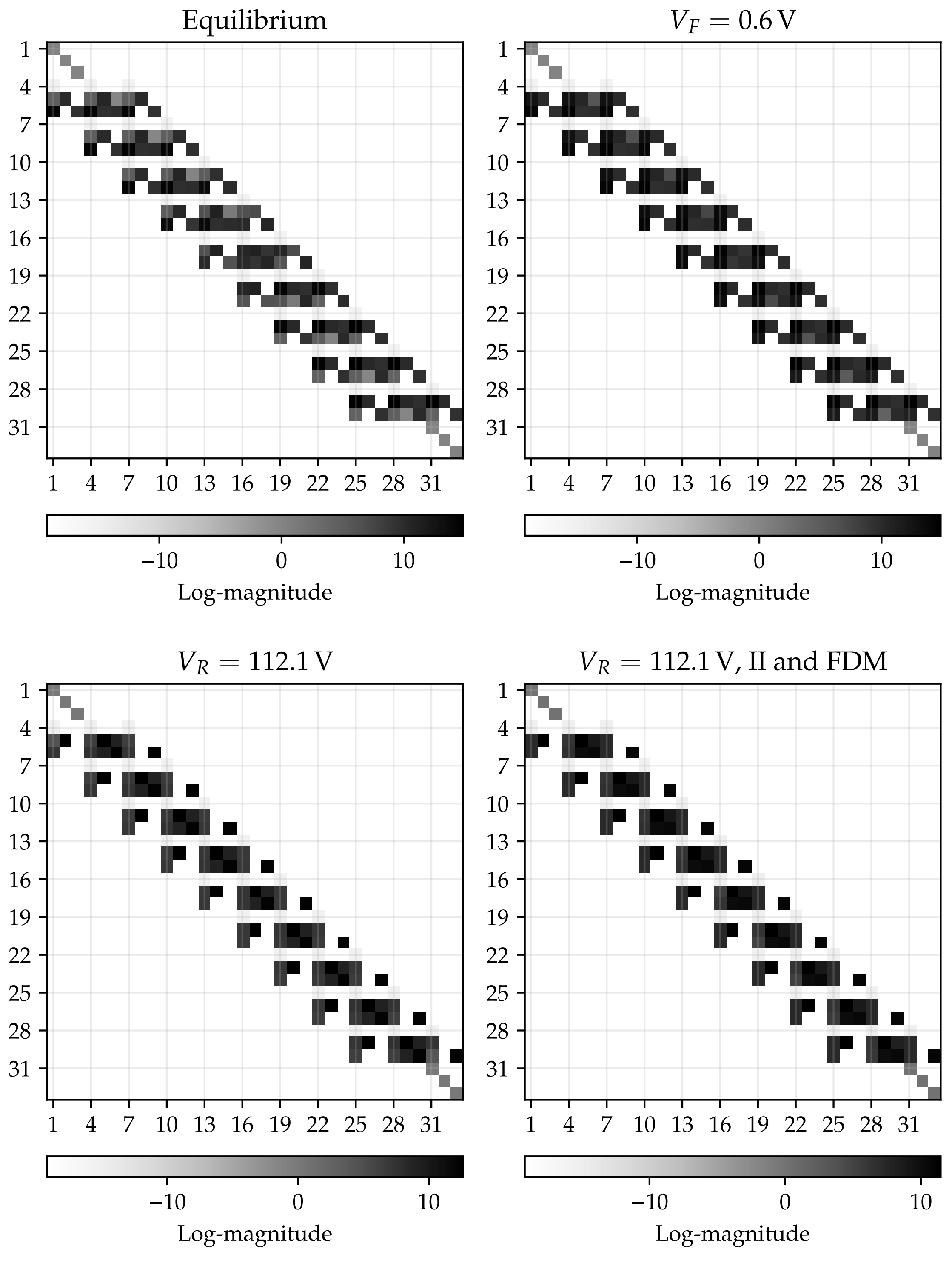


Figure B.1: Jacobian matrices of 1D diode generated by `diode1d(2.0,2.0,ha=0.04,hb=0.04, hc=0.04)`. The resulting model has an 11-point discretization grid with uniform spacing. Bias conditions are labeled in the plot titles.

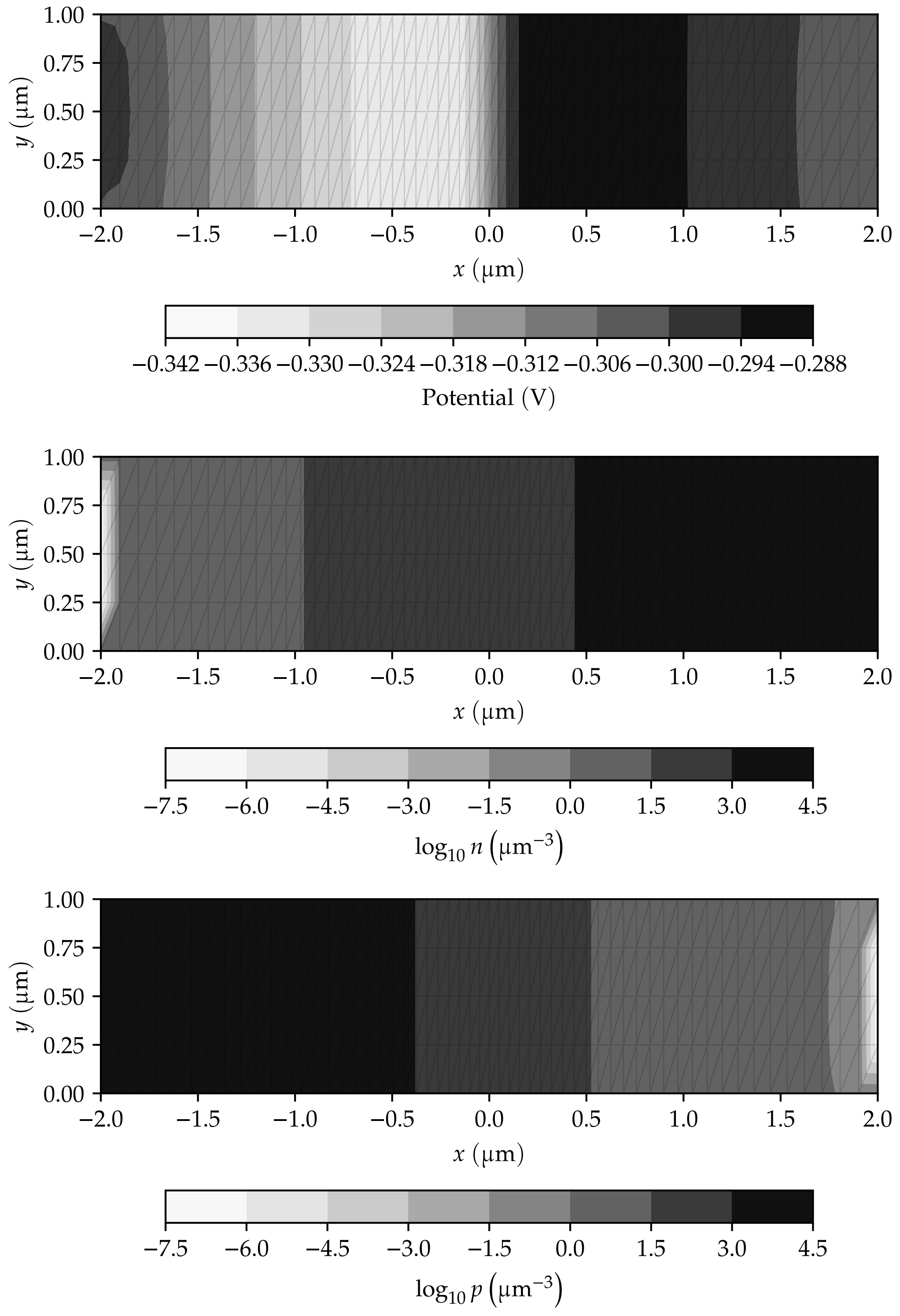


Figure B.2: Potential (top), electron density (middle) and hole density (bottom) in 2D diode generated by `diode2d(2.0,2.0,1.0,0.5,rectgrid=true)`. Simulation was performed with a forward bias of 0.6 V.

the triangular grid shown in Figure 4.17. This grid has limited resolution in the y direction, but it provides a more regular discretization along the x direction as opposed to the unstructured triangular discretization. Consequently, local refinement is only possible in the x direction; any refinement applied in the y direction creates additional cells along the entire device, which is not generally desirable.

Figure B.4 shows the vector-valued electric field and current density in the NPN BJT generated by `npn1(5,1,10,5,3,3,0.2)` with $I_B = 5 \times 10^{-7}\,\mathrm{A\,\mu m^{-1}}$ and $V_{CE} = 2\,\mathrm{V}$. Large electric fields exist near the base-emitter junction and base-collector junction due to their depletion regions. The current density is roughly constant along the device due to conservation of charge. A small current is visible pointing inwards from the base contact due to the base current, which is relatively small compared to the collector current. Slight increases in the magnitude of the current density field are also visible near the collector and emitter contacts due to current crowding.

B.3 MOSFETs

Figure B.5 shows the vector-valued electric field and current density in the n-channel MOSFET generated by `mos1(0.1,0.05,0.2,0.05,0.002,0.1,0.025)` with $V_{DS} = 1\,\mathrm{V}$ and $V_{GS} = -0.5\,\mathrm{V}$. This figure shows the path of undesirable leakage currents through the substrate when $V_{GS} < 0$. In this region, the device is driven into accumulation, a condition in which electrons are strongly repelled from the bulk-oxide interface. Current can no longer flow through the channel region in accumulation, and the current must flow through the reverse-biased bulk-drain junction and into the substrate. This effect is similar to the off-state leakage in a BJT and is often modeled as a separate *parasitic transistor* in the substrate. The parasitic substrate BJT can cause latch-up in CMOS devices, which can be highly disruptive in digital logic circuits [23].

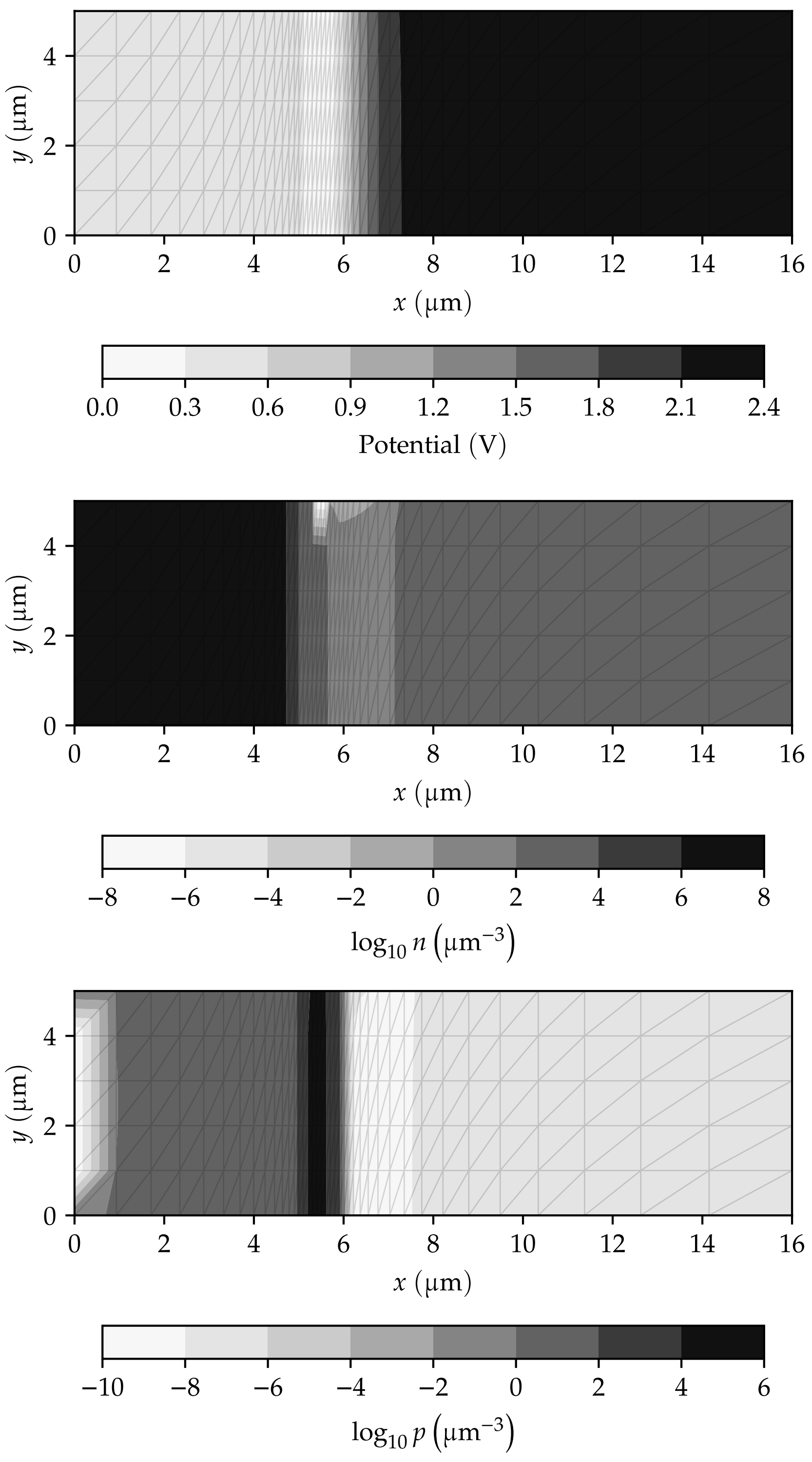


Figure B.3: Potential (top), electron density (middle) and hole density (bottom) in NPN BJT generated by `npn1(5,1,10,5,3,3,0.2,rectgrid=true)`. Simulation was performed with $I_B = 5 \times 10^{-7}\,\mathrm{A\,\mu m^{-1}}$ and $V_{CE} = 2\,\mathrm{V}$.

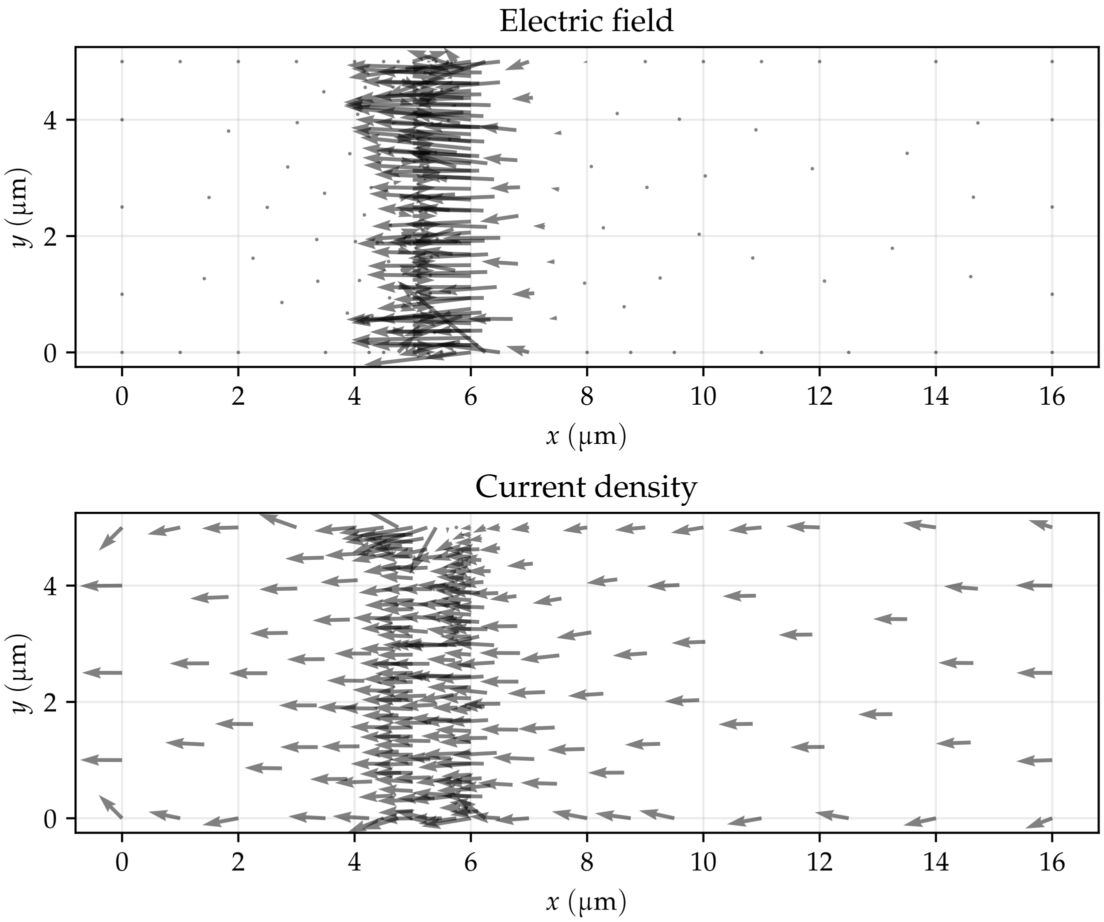
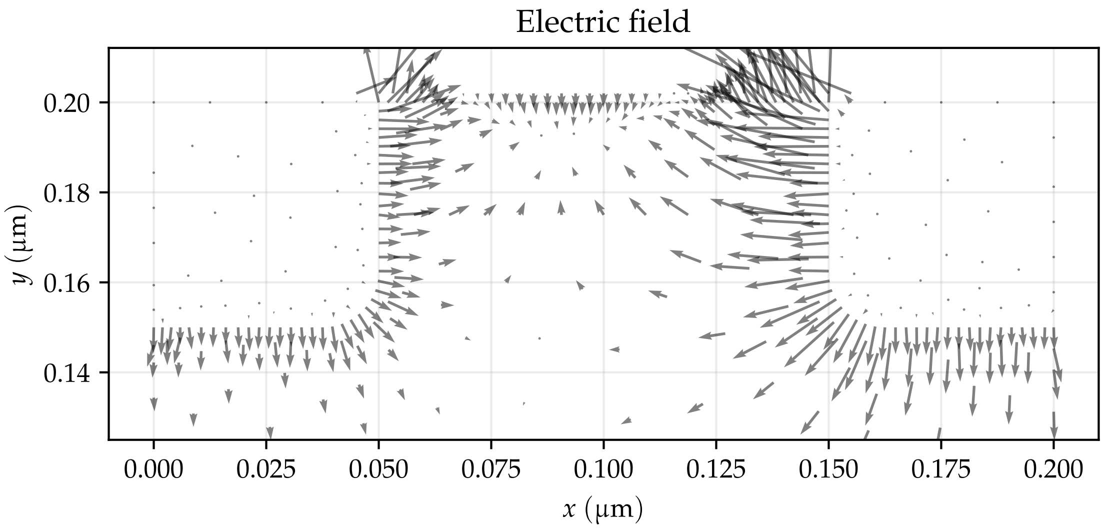


Figure B.4: Electric field (top) and current density (bottom) in NPN BJT generated by `npn1(5, 1,10,5,3,3,0.2)`. Simulation was performed with $I_B = 5 \times 10^{-7}\,\mathrm{A\,\mu m^{-1}}$ and $V_{CE} = 2\,\mathrm{V}$. Arrow length is proportional to field magnitude; one arrow is plotted for each discretization node.



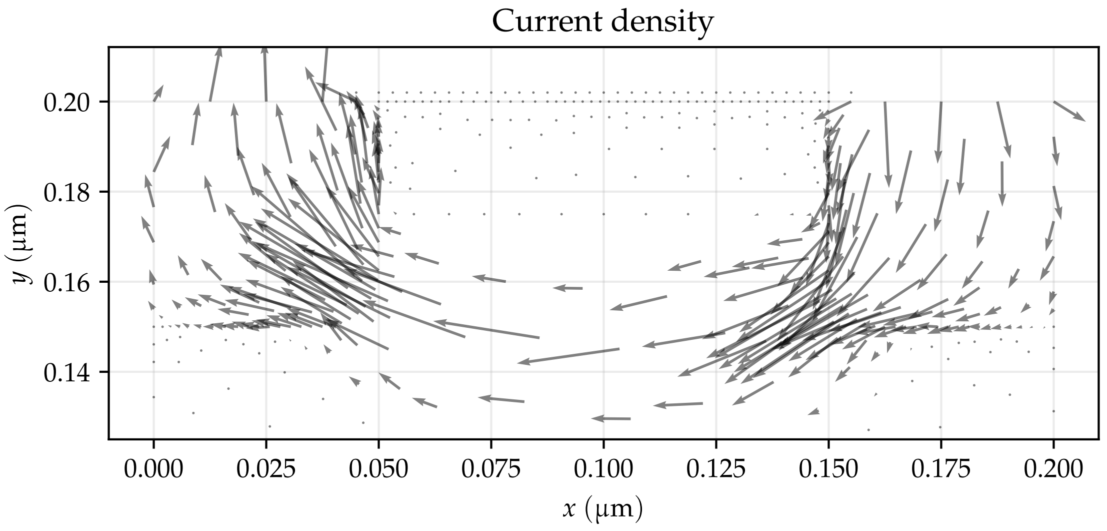


Figure B.5: Electric field (top) and current density (bottom) in *n*-channel MOSFET generated by `mos1(0.1,0.05,0.2,0.05,0.002,0.1,0.025)` with $V_{DS} = 1\,\text{V}$ and $V_{GS} = -0.5\,\text{V}$. Arrow length is proportional to field magnitude; one arrow is plotted for each discretization node.

Bibliography

[1] R. Anantharaman et al. *Accelerating Simulation of Stiff Nonlinear Systems using Continuous-Time Echo State Networks*. 2021. arXiv: `2010.04004 [cs.LG]`.

[2] Y. Augenstein and C. Rockstuhl. "Inverse Design of Nanophotonic Devices With Structural Integrity". In: *ACS Photonics* 7.8 (2020), pp. 2190–2196. DOI: `10.1021/acsphotonics.0c00699`.

[3] R. E. Bank, D. J. Rose, and W. Fichtner. "Numerical Methods for Semiconductor Device Simulation". In: *IEEE Transactions on Electron Devices* 30.9 (1983), pp. 1031–1041. DOI: `10.1109/T-ED.1983.21257`.

[4] R. E. Bank and T. F. Chan. "PLTMGC: A Multigrid Continuation Program for Parameterized Nonlinear Elliptic Systems". In: *SIAM Journal on Scientific and Statistical Computing* 7.2 (1986), pp. 540–559. DOI: `10.1137/0907036`.

[5] R. E. Bank, W. M. Coughran Jr., and L. C. Cowsar. "The Finite Volume Scharfetter-Gummel Method for Steady Convection Diffusion Equations". In: *Computing and Visualization in Science* 1.3 (1998), pp. 123–136. DOI: `10.1007/s007910050012`.

[6] R. E. Bank and D. J. Rose. "Global Approximate Newton Methods". In: *Numerische Mathematik* 37.2 (1981), pp. 279–295. DOI: `10.1007/BF01398257`.

[7] M. Bartels et al. "A Robust Curve Tracing Scheme for the Simulation of Bipolar Breakdown Characteristics With Nonlocal Impact Ionization Models". In: *29th European Solid-State Device Research Conference*. Vol. 1. 1999, pp. 492–495. URL: `https://ieeexplore.ieee.org/abstract/document/1505547`.

[8] M. K. Basak and M. A.-A. Joarder. "Simulation of a 2D P-N Junction in Silicon Thin Film Incorporating Quantum Transport For Carriers". B.S. thesis. Daffodil International University, 2017. URL: `http://dspace.daffodilvarsity.edu.bd:8080/handle/20.500.11948/1850`.

[9] D. Bednarczyk and J. Bednarczyk. "The approximation of the Fermi-Dirac integral $F_{1/2}(\eta)$". In: *Physics Letters A* 64.4 (1978), pp. 409–410. DOI: `10.1016/0375-9601(78)90283-9`.

[10] J. Bezanson et al. *Julia: A Fast Dynamic Language for Technical Computing*. 2012. arXiv: `1209.5145 [cs.PL]`.

[11] P. Blakey. "Transistor Modeling and TCAD". In: *IEEE Microwave Magazine* 13.7 (2012), pp. 28–35. DOI: `10.1109/MMM.2012.2216103`.

[12] M. Burger et al. "On Inverse Problems for Semiconductor Equations". In: *Milan Journal of Mathematics* 72.1 (2004), pp. 273–313. DOI: `10.1007/s00032-004-0025-6`.

[13] M. Burger, H. W. Engl, and P. A. Markowich. “Inverse Doping Problems for Semiconductor Devices”. In: *Recent Progress in Computational and Applied PDEs*. Ed. by T. F. Chan et al. Boston, MA: Springer US, 2002, pp. 39–53. ISBN: 9781461501138. DOI: 10.1007/978-1-4615-0113-8_3.

[14] J. F. Burgler et al. “A New Discretization Scheme for the Semiconductor Current Continuity Equations”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 8.5 (1989), pp. 479–489. DOI: 10.1109/43.24876.

[15] C. S. Burrus et al. “Horner’s Method for Evaluating and Deflating Polynomials”. In: *DSP Software Notes* 26 (2003). URL: https://www.ece.rice.edu/dsp/software/FVHDP/horner2.pdf.

[16] D. M. Caughey and R. E. Thomas. “Carrier Mobilities in Silicon Empirically Related to Doping and Field”. In: *Proceedings of the IEEE* 55.12 (1967), pp. 2192–2193. DOI: 10.1109/PROC.1967.6123.

[17] R. T. Q. Chen et al. *Neural Ordinary Differential Equations*. 2019. arXiv: 1806.07366 [cs.LG].

[18] Y. Cheng, I. M. Gamba, and K. Ren. “Recovering Doping Profiles in Semiconductor Devices With the Boltzmann-Poisson Model”. In: *Journal of Computational Physics* 230.9 (2011), pp. 3391–3412. DOI: 10.1016/j.jcp.2011.01.034.

[19] A. G. Chynoweth. “Ionization Rates for Electrons and Holes in Silicon”. In: *Physical Review* 109.5 (1958), pp. 1537–1540. DOI: 10.1103/PhysRev.109.1537.

[20] W. J. Cody and H. C. Thacher. “Rational Chebyshev Approximations for Fermi-Dirac Integrals of Orders −1/2, 1/2 and 3/2”. In: *Mathematics of Computation* 21.97 (1967), pp. 30–40. ISSN: 00255718, 10886842. URL: http://www.jstor.org/stable/2003468.

[21] S. Colburn and A. Majumdar. “Inverse Design and Flexible Parameterization of Meta-Optics Using Algorithmic Differentiation”. In: *Communications Physics* 4.1 (2021), pp. 1–11. DOI: 10.1038/s42005-021-00568-6.

[22] W. M. Coughran, E. Grosse, and D. J. Rose. “CAzM: A Circuit Analyzer With Macromodeling”. In: *IEEE Transactions on Electron Devices* 30.9 (1983), pp. 1207–1213. DOI: 10.1109/T-ED.1983.21276.

[23] W. M. Coughran, M. R. Pinto, and R. K. Smith. “Computation of Steady-State CMOS Latchup Characteristics”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 7.2 (1988), pp. 307–323. DOI: 10.1109/43.3162.

[24] W. M. Coughran, M. R. Pinto, and R. K. Smith. “Continuation Methods in Semiconductor Device Simulation”. In: *Continuation Techniques and Bifurcation Problems*. Ed. by H. D. Mittelmann and D. Roose. Springer, 1990, pp. 47–65. ISBN: 9783764323974.

[25] C. K. Dabhi et al. *BSIM4 v4.8.2 MOSFET Model*. 2020. URL: http://bsim.berkeley.edu/BSIM4/BSIM4_4.8.2_20200101.tar.gz.

[26] R. Dandekar, C. Rackauckas, and G. Barbastathis. “A Machine Learning-Aided Global Diagnostic and Comparative Tool to Assess Effect of Quarantine Control in COVID-19 Spread”. In: *Patterns* 1.9 (2020), p. 100145. ISSN: 2666-3899. DOI: 10.1016/j.patter.2020.100145.

[27] R. Dandekar et al. *Bayesian Neural Ordinary Differential Equations*. 2022. arXiv: 2012.07244 [cs.LG].

[28] J. L. Dawson et al. "Optimal Allocation of Local Feedback in Multistage Amplifiers via Geometric Programming". In: *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications* 48.1 (2001), pp. 1–11. DOI: 10.1109/81.903183.

[29] S. B. Desai et al. "MoS_2 Transistors With 1-Nanometer Gate Lengths". In: *Science* 354.6308 (2016), pp. 99–102. DOI: 10.1126/science.aah4698.

[30] H. Dhillon et al. "TCAD-Augmented Machine Learning With and Without Domain Expertise". In: *IEEE Transactions on Electron Devices* 68.11 (2021), pp. 5498–5503. DOI: 10.1109/TED.2021.3073378.

[31] J. Dickerson, R. J. Kaplar, and G. Pickrell. *Modeling of Avalanche Breakdown in Silicon and Gallium Nitride High-Voltage Diodes Using COMSOL*. 2018. URL: https://www.osti.gov/servlets/purl/1594284.

[32] G. Ding. *A Software Package for Numerical Simulation of Semiconductor Devices Under HPM Environment*. 2006. URL: https://www.yumpu.com/en/document/read/6664225/a-software-package-for-numerical-simulation-of-semiconductor-gss.

[33] D. H. Doan et al. "Drift-Diffusion Simulation of S-Shaped Current-Voltage Relations for Organic Semiconductor Devices". In: *Journal of Computational Electronics* 19.3 (2020), pp. 1164–1174. DOI: 10.1007/s10825-020-01505-6.

[34] J. J. Ebers and J. L. Moll. "Large-Signal Behavior of Junction Transistors". In: *Proceedings of the IRE* 42.12 (1954), pp. 1761–1772. DOI: 10.1109/JRPROC.1954.274797.

[35] H. Fardi. "Numerical Analysis of Semiconductor PN Junctions Using MATLAB". In: *Journal of Scientific Research and Reports* 6.2 (2015), pp. 84–98. URL: https://www.journaljsrr.com/index.php/JSRR/article/view/21668.

[36] P. Farrell, T. Koprucki, and J. Fuhrmann. "Computational and Analytical Comparison of Flux Discretizations for the Semiconductor Device Equations Beyond Boltzmann Statistics". In: *Journal of Computational Physics* 346 (2017), pp. 497–513. DOI: 10.1016/j.jcp.2017.06.023.

[37] P. Farrell and D. Peschka. *Challenges for Drift-Diffusion Simulations of Semiconductors: A Comparative Study of Different Discretization Philosophies*. Mar. 2018. DOI: 10.20347/WIAS.PREPRINT.2486.

[38] P. Farrell et al. *Numerical Methods for Drift-Diffusion Models*. 2016. DOI: 10.20347/WIAS.PREPRINT.2263.

[39] P. E. Farrell, C. H. L. Beentjes, and Á. Birkisson. *The Computation of Disconnected Bifurcation Diagrams*. 2016. arXiv: 1603.00809 [math.NA].

[40] P. E. Farrell, Á. Birkisson, and S. W. Funke. *Deflation Techniques for Finding Distinct Solutions of Nonlinear Partial Differential Equations*. 2015. arXiv: 1410.5620 [math.NA].

[41] H. Feng and S. Zhao. "FFT-Based High Order Central Difference Schemes for Three-Dimensional Poisson's Equation with Various Types of Boundary Conditions". In: *Journal of Computational Physics* 410.109391 (2020). ISSN: 0021-9991. DOI: 10.1016/j.jcp.2020.109391.

[42] W. Fichtner, D. J. Rose, and R. E. Bank. "Semiconductor Device Simulation". In: *SIAM Journal on Scientific and Statistical Computing* 4.3 (1983), pp. 391–415. DOI: 10.1137/0904031.

[43] C. G. Fonstad. *Microelectronic Devices and Circuits*. McGraw-Hill Series in Electrical and Computer Engineering: Electronics and VLSI Circuits. New York, NY: McGraw-Hill, 1994. ISBN: 0070214964.

[44] W. R. Frensley. *Scharfetter-Gummel Discretization Scheme for Drift-Diffusion Equations*. 2004. URL: `https://personal.utdallas.edu/~frensley/minitech/ScharfGum.pdf`.

[45] J. Fuhrmann and contributors. *VoronoiFVM.jl: Finite Volume Solver for Coupled Nonlinear Partial Differential Equations*. 2022. DOI: `10.5281/zenodo.3529808`. URL: `https://github.com/j-fu/VoronoiFVM.jl`.

[46] A. H. Gebremedhin, F. Manne, and A. Pothen. "What Color Is Your Jacobian? Graph Coloring for Computing Derivatives". In: *SIAM Review* 47.4 (2005), pp. 629–705. DOI: `10.1137/S0036144504444711`.

[47] A. Ghazarians. "A Numerical Study of the van Roosbroeck System for Semiconductors". M.S. thesis. San Jose State University, 2018. DOI: `10.31979/etd.k2yb-6c32`.

[48] N. L. Gibson. *Numerical Methods for Maxwell's Equations*. 2015. URL: `http://sites.science.oregonstate.edu/~gibsonn/553Talk.pdf`.

[49] M. B. Giles. *An Extended Collection of Matrix Derivative Results for Forward and Reverse Mode Algorithmic Differentiation*. 2008. URL: `https://people.maths.ox.ac.uk/gilesm/files/NA-08-01.pdf`.

[50] J. Goldberger et al. "Silicon Vertically Integrated Nanowire Field Effect Transistors". In: *Nano Letters* 6.5 (2006), pp. 973–977. DOI: `10.1021/nl060166j`.

[51] R. J. G. Goossens et al. "An Automatic Biasing Scheme for Tracing Arbitrarily Shaped I-V Curves". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 13.3 (1994), pp. 310–317. DOI: `10.1109/43.265673`.

[52] S. Gowda et al. *High-Performance Symbolic-Numerics via Multiple Dispatch*. 2022. arXiv: `2105.03949 [cs.CL]`.

[53] P. R. Gray et al. *Analysis and Design of Analog Integrated Circuits*. 5th ed. New York, NY: John Wiley & Sons, 2009. ISBN: 9780470245996.

[54] A. Griewank and A. Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. SIAM, 2008. ISBN: 9780898716597. DOI: `10.1137/1.9780898717761`.

[55] B. M. Grossman and M. J. Hargrove. "Numerical Solution of the Semiconductor Transport Equations With Current Boundary Conditions". In: *IEEE Transactions on Electron Devices* 30.9 (1983), pp. 1092–1096. DOI: `10.1109/T-ED.1983.21263`.

[56] H. K. Gummel and H. C. Poon. "An Integral Charge Control Model of Bipolar Transistors". In: *The Bell System Technical Journal* 49.5 (1970), pp. 827–852. DOI: `10.1002/j.1538-7305.1970.tb01803.x`.

[57] R. Gusmeroli and A. S. Spinelli. "Accurate Boundary Integral Calculation in Semiconductor Device Simulation". In: *IEEE Transactions on Electron Devices* 53.7 (2006), pp. 1730–1733. DOI: `10.1109/TED.2006.875806`.

[58] R. Han et al. "A SiGe Terahertz Heterodyne Imaging Transmitter With 3.3 mW Radiated Power and Fully-Integrated Phase-Locked Loop". In: *IEEE Journal of Solid-State Circuits* 50.12 (2015), pp. 2935–2947. DOI: `10.1109/JSSC.2015.2471847`.

[59] S. Hang. "TetGen, a Delaunay-Based Quality Tetrahedral Mesh Generator". In: *ACM Transactions on Mathematical Software* 41.2 (2015), pp. 11.2–11.36. DOI: `10.1145/2629697`.

[60] L. Hascoet and V. Pascual. "The Tapenade Automatic Differentiation Tool: Principles, Model, and Specification". In: *ACM Transactions on Mathematical Software* 39.3 (2013), pp. 1–43. DOI: `10.1145/2450153.2450158`.

[61] Y. He and Z. Teng. "Nonoscillatory Streamline Upwind Formulations for Drift-Diffusion Equation". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 12.10 (1993), pp. 1535–1541. DOI: 10.1109/43.256928.

[62] D. Hendrycks and K. Gimpel. *Gaussian Error Linear Units (GELUs)*. 2020. arXiv: 1606.08415 [cs.LG].

[63] M. d. M. Hershenson, S. P. Boyd, and T. H. Lee. "Optimal Design of a CMOS Op-Amp via Geometric Programming". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 20.1 (2001), pp. 1–21. DOI: 10.1109/43.905671.

[64] M. d. M. Hershenson et al. "Design and Optimization of LC Oscillators". In: *1999 IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers*. 1999, pp. 65–69. DOI: 10.1109/ICCAD.1999.810623.

[65] D. Hisamoto et al. "A Folded-Channel MOSFET for Deep-Sub-Tenth Micron Era". In: *International Electron Devices Meeting 1998*. 1998, pp. 1032–1034. DOI: 10.1109/IEDM.1998.746531.

[66] T. W. Hughes et al. "Forward-Mode Differentiation of Maxwell's Equations". In: *ACS Photonics* 6.11 (2019), pp. 3010–3016. DOI: 10.1021/acsphotonics.9b01238.

[67] G. A. M. Hurkx et al. "A New Analytical Diode Model Including Tunneling and Avalanche Breakdown". In: *IEEE Transactions on Electron Devices* 39.9 (1992), pp. 2090–2098. DOI: 10.1109/16.155882.

[68] M. Innes. *Don't Unroll Adjoint: Differentiating SSA-Form Programs*. 2018. arXiv: 1810.07951 [cs.PL].

[69] C. Jeong et al. "Bridging TCAD and AI: Its Application to Semiconductor Design". In: *IEEE Transactions on Electron Devices* 68.11 (2021), pp. 5364–5371. DOI: 10.1109/TED.2021.3093844.

[70] S. G. Johnson. *Notes on Adjoint Methods for 18.335*. 2012. URL: https://math.mit.edu/~stevenj/18.336/adjoint.pdf.

[71] S. Joshi, S. Boyd, and R. W. Dutton. "Optimal Doping Profiles via Geometric Programming". In: *IEEE Transactions on Electron Devices* 52.12 (2005), pp. 2660–2675. DOI: 10.1109/TED.2005.859649.

[72] C. Jungemann et al. "Is Physically Sound and Predictive Modeling of NMOS Substrate Currents Possible?" In: *Solid-State Electronics* 42.4 (1998), pp. 647–655. DOI: 10.1016/S0038-1101(97)00298-0.

[73] W. Kausel et al. "A New Boundary Condition for Device Simulation Considering Outer Components". In: *Simulation of Semiconductor Devices and Processes, Vol. 3*. Ed. by G. Baccarani and M. Rudan. Bologna, Italy: Tecnoprint, 1988.

[74] H. B. Keller. "Lectures on Numerical Methods in Bifurcation Problems". In: (1988). URL: http://www.math.tifr.res.in/~publ/ln/tifr79.pdf.

[75] T. Kerkhoven. "On the Scharfetter-Gummel Box-Method". In: *Simulation of Semiconductor Devices and Processes, Vol. 5*. Ed. by S. Selberherr, H. Stippel, and E. Strasser. Vienna, Austria: Springer, 1993, pp. 417–420. ISBN: 9783709173725.

[76] I. Kim et al. "Simulator Acceleration and Inverse Design of Fin Field-Effect Transistors Using Machine Learning". In: *Scientific Reports* 12.1 (2022), pp. 1–9. DOI: 10.1038/s41598-022-05111-3.

[77] R. Kim, X. Wang, and M. Lundstrom. *Notes on Fermi-Dirac Integrals*. 2019. arXiv: 0811.0116 [cond-mat.mes-hall].

[78] S. Kim et al. "Stiff Neural Ordinary Differential Equations". In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 31.9 (2021), p. 093122. DOI: 10.1063/5.0060697.

[79] D. P. Kingma and J. Ba. *Adam: A Method for Stochastic Optimization*. 2014. arXiv: 1412.6980 [cs.LG].

[80] S. Kumashiro. *Method of Simulating Impact Ionization Phenomenon in Semiconductor Device*. US Patent 6,144,929. Nov. 2000.

[81] S. E. Laux and B. M. Grossman. "A General Control-Volume Formulation for Modeling Impact Ionization in Semiconductor Transport". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 4.4 (1985), pp. 520–526. DOI: 10.1109/TCAD.1985.1270151.

[82] S. E. Laux and R. G. Byrnes. "Semiconductor Device Simulation Using Generalized Mobility Models". In: *IBM Journal of Research and Development* 29.3 (1985), pp. 289–301. DOI: 10.1147/rd.293.0289.

[83] Z. K. Lee, M. B. McIlrath, and D. A. Antoniadis. "Two-Dimensional Doping Profile Characterization of MOSFETs by Inverse Modeling Using I-V Characteristics in the Subthreshold Region". In: *IEEE Transactions on Electron Devices* 46.8 (1999), pp. 1640–1649. DOI: 10.1109/16.777152.

[84] A. Leitão, P. A. Markowich, and J. P. Zubelli. "On Inverse Doping Profile Problems for the Stationary Voltage-Current Map". In: *Inverse Problems* 22.3 (2006), p. 1071. DOI: 10.1088/0266-5611/22/3/021.

[85] R. G. Leventhal. *Semiconductor Modeling for Simulating Signal, Power, and Electromagnetic Integrity*. New York, NY: Springer, 2006. ISBN: 9780387241593.

[86] Z. Li et al. *Fourier Neural Operator for Parametric Partial Differential Equations*. 2021. arXiv: 2010.08895 [cs.LG].

[87] M. Liu. "10-nm CMOS: A Design Study on Technology Requirement With Power and Performance Assessment". PhD thesis. UC San Diego, 2007. URL: https://escholarship.org/uc/item/64g181j0.

[88] W. Liu et al. *BSIM3 v3.2 MOSFET Model*. Tech. rep. UCB/ERL M98/51. EECS Department, University of California, Berkeley, Aug. 1998. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/1998/3486.html.

[89] M. Lundstrom. *A Primer on Semiconductor Device Simulation*. Jan. 2006. URL: https://nanohub.org/resources/980.

[90] M. Lundstrom. *Nanoscale Transistors: Device Physics, Modeling and Simulation*. 1st ed. New York, NY: Springer US, 2006. ISBN: 1280612215.

[91] P. Mandal and V. Visvanathan. "CMOS Op-Amp Sizing Using a Geometric Programming Formulation". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 20.1 (2001), pp. 22–38. DOI: 10.1109/43.905672.

[92] P. A. Markowich. "A Nonlinear Eigenvalue Problem Modelling the Avalanche Effect in Semiconductor Diodes". In: *SIAM Journal on Mathematical Analysis* 16.6 (1985), pp. 1268–1283. DOI: 10.1137/0516091.

[93] P. A. Markowich. "A Singular Perturbation Analysis of the Fundamental Semiconductor Device Equations". In: *SIAM Journal on Applied Mathematics* 44.5 (1984), pp. 896–928. DOI: 10.1137/0144064.

[94] P. A. Markowich. *The Stationary Semiconductor Device Equations*. Springer Science & Business Media, 1985. ISBN: 9783211818923.

[95] P. A. Markowich, C. A. Ringhofer, and A. Steindl. "Computation of Current-Voltage Characteristics in a Semiconductor Device Using Arc-Length Continuation". In: *IMA Journal of Applied Mathematics* 33.2 (1984), pp. 175–187. URL: https://apps.dtic.mil/sti/citations/ADA137919.

[96] A. Mauri et al. "3D Finite Element Modeling and Simulation of Industrial Semiconductor Devices Including Impact Ionization". In: *Journal of Mathematics in Industry* 5.1 (2015), pp. 1–18. DOI: 10.1186/s13362-015-0015-z.

[97] C. C. McAndrew et al. "VBIC95, the Vertical Bipolar Inter-Company Model". In: *IEEE Journal of Solid-State Circuits* 31.10 (1996), pp. 1476–1483. DOI: 10.1109/4.540058.

[98] L. S. McCarthy et al. "AlGaN/GaN Heterojunction Bipolar Transistor". In: *IEEE Electron Device Letters* 20.6 (1999), pp. 277–279. DOI: 10.1109/55.767097.

[99] K. Mehta and H.-Y. Wong. "Prediction of FinFET Current-Voltage and Capacitance-Voltage Curves Using Machine Learning With Autoencoder". In: *IEEE Electron Device Letters* 42.2 (2021), pp. 136–139. DOI: 10.1109/LED.2020.3045064.

[100] K. Mehta et al. "Improvement of TCAD Augmented Machine Learning Using Autoencoder for Semiconductor Variation Identification and Inverse Design". In: *IEEE Access* 8 (2020), pp. 143519–143529. DOI: 10.1109/ACCESS.2020.3014470.

[101] L. Meitner. "Über die Entstehung der β-Strahl-Spektren Radioaktiver Substanzen". In: *Zeitschrift für Physik* 9.1 (1922), pp. 131–144. DOI: 10.1007/BF01326962.

[102] S. S. Mohan et al. "Simple Accurate Expressions for Planar Spiral Inductances". In: *IEEE Journal of Solid-State Circuits* 34.10 (1999), pp. 1419–1424. DOI: 10.1109/4.792620.

[103] S. Molesky et al. "Inverse Design in Nanophotonics". In: *Nature Photonics* 12.11 (2018), pp. 659–670. DOI: 10.1038/s41566-018-0246-9.

[104] L. W. Nagel and D. O. Pederson. *SPICE (Simulation Program with Integrated Circuit Emphasis)*. Tech. rep. UCB/ERL M382. EECS Department, University of California, Berkeley, Apr. 1973. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/1973/22871.html.

[105] G. Nanz. "A Critical Study of Boundary Conditions in Device Simulation". In: *Simulation of Semiconductor Devices and Processes, Vol. 4*. Ed. by W. Fichtner and D. Aemmer. Konstanz, Germany: Hartung-Gorre Verlag, 1991. ISBN: 3891914768.

[106] G. Nanz, P. Dickinger, and S. Selberherr. "Calculation of Contact Currents in Device Simulation". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 11.1 (1992), pp. 128–136. DOI: 10.1109/43.108625.

[107] S. G. Nash. "Preconditioning of Truncated-Newton Methods". In: *SIAM Journal on Scientific and Statistical Computing* 6.3 (1985), pp. 599–616. DOI: 10.1137/0906042.

[108] G. Niu et al. *The Mextram Bipolar Transistor Model, Version 505.1.0*. 2017. URL: https://www.eng.auburn.edu/~niuguof/mextram/_downloads/MextramDefinition.pdf.

[109] V. Palankovski and R. Quay. *Analysis and Simulation of Heterostructure Devices*. Ed. by S. Selberherr. Computational Microelectronics. Vienna, Austria: Springer, 2004. ISBN: 3211405372.

[110] E. Palm and F. Van de Wiele. "Current Lines and Accurate Contact Current Evaluation in 2-D Numerical Simulation of Semiconductor Devices". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 4.4 (1985), pp. 496–503. DOI: 10.1109/TCAD.1985.1270148.

[111] J. M. Papakonstantinou. "Historical Development of the BFGS Secant Method and its Characterization Properties". PhD thesis. Rice University, 2009. URL: https://hdl.handle.net/1911/61898.

[112] T. Patel. *Comparison of Level 1, 2 and 3 MOSFET's*. 2014. DOI: 10.13140/RG.2.1.1616.3442.

[113] Y.-J. Peng. "Boundary Layer Analysis and Quasi-Neutral Limits in the Drift-Diffusion Equations". In: *ESAIM: Mathematical Modelling and Numerical Analysis* 35.2 (2001), pp. 295–312. DOI: 10.1051/m2an:2001116.

[114] R. Pestourie et al. "Inverse Design of Large-Area Metasurfaces". In: *Optics Express* 26.26 (2018), pp. 33732–33747. DOI: 10.1364/OE.26.033732.

[115] R. Pestourie et al. "Physics-Enhanced Deep Surrogates for PDEs". In: (2021). arXiv: 2111.05841 [cs.LG].

[116] G. Peters and J. H. Wilkinson. "On the Stability of Gauss-Jordan Elimination With Pivoting". In: *Communications of the ACM* 18.1 (Jan. 1975), pp. 20–24. ISSN: 0001-0782. DOI: 10.1145/360569.360653.

[117] R. F. Pierret. *Semiconductor Device Fundamentals*. Reading, MA: Addison-Wesley, 1996. ISBN: 0201543931.

[118] A. Y. Piggott et al. "Inverse Design and Demonstration of a Compact and Broadband On-Chip Wavelength Demultiplexer". In: *Nature Photonics* 9.6 (2015), pp. 374–377. DOI: 10.1038/nphoton.2015.69.

[119] M. Pinto, K. R. Smith, and A. Alam. *Padre 2.4E Users Manual*. 1994. URL: https://nanohub.org/resources/3943/download/padre_manual.pdf.

[120] M. R. Pinto, C. S. Rafferty, and R. W. Dutton. *PISCES II: Poisson and Continuity Equation Solver*. Stanford, CA, 1984. URL: https://web.archive.org/web/20151210025620/http://www-tcad.stanford.edu/tcad/reports/piscesII.pdf.

[121] J. Piprek. *Semiconductor Optoelectronic Devices: Introduction to Physics and Simulation*. Elsevier, 2013. ISBN: 9780125571906.

[122] E. Pop. "CMOS Inverse Doping Profile Extraction and Substrate Current Modeling". M.Eng. thesis. Massachusetts Institute of Technology, 1999. URL: https://dspace.mit.edu/handle/1721.1/80565.

[123] C. H. Price. "Two-Dimensional Numerical Simulation of Semiconductor Devices". PhD thesis. Stanford University, 1982. URL: https://apps.dtic.mil/sti/pdfs/ADA119110.pdf.

[124] C. Rackauckas and Q. Nie. "DifferentialEquations.jl—A Performant and Feature Rich Ecosystem for Solving Differential Equations in Julia". In: *Journal of Open Research Software* 5.1 (2017), p. 15. DOI: 10.5334/jors.151.

[125] C. Rackauckas et al. "Generalized Physics-Informed Learning through Language-Wide Differentiable Programming". In: *Proceedings of the AAAI 2020 Spring Symposium on Combining Artificial Intelligence and Machine Learning with Physical Sciences*. Ed. by J. Lee et al. Vol. 2587. CEUR Workshop Proceedings. CEUR-WS.org, 2020. URL: https://dspace.mit.edu/bitstream/handle/1721.1/137320/%20generalized-physics.pdf?sequence=2&isAllowed=y.

[126] C. Rackauckas et al. *Universal Differential Equations for Scientific Machine Learning*. 2021. arXiv: `2001.04385 [cs.LG]`.

[127] M. Raissi, P. Perdikaris, and G. E. Karniadakis. "Physics-Informed Neural Networks: A Deep Learning Framework for Solving Forward and Inverse Problems Involving Nonlinear Partial Differential Equations". In: *Journal of Computational Physics* 378 (2019), pp. 686–707. ISSN: 0021-9991. DOI: `10.1016/j.jcp.2018.10.045`.

[128] A. Ramadhan et al. *Capturing Missing Physics in Climate Model Parameterizations Using Neural Differential Equations*. 2020. arXiv: `2010.12559 [physics.ao-ph]`.

[129] J. Revels, M. Lubin, and T. Papamarkou. *Forward-Mode Automatic Differentiation in Julia*. 2016. arXiv: `1607.07892 [cs.MS]`.

[130] F. E. Reyes Aspé. "Simulation Tool Development for Semiconductor Devices Based on Drift-Diffusion and Monte Carlo". M.S. thesis. University of Chile, 2015. URL: `https://repositorio.uchile.cl/handle/2250/137791`.

[131] J. Roychowdhury and R. Melville. "Delivering Global DC Convergence for Large Mixed-Signal Circuits via Homotopy/Continuation Methods". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 25.1 (2006), pp. 66–78. DOI: `10.1109/TCAD.2005.852461`.

[132] A. Safdari-Vaighani, A. Heryudono, and E. Larsson. "A Radial Basis Function Partition of Unity Collocation Method for Convection-Diffusion Equations Arising in Financial Applications". In: *Journal of Scientific Computing* 64.2 (2015), pp. 341–367. DOI: `10.1007/s10915-014-9935-9`.

[133] Z. H. Sahul, R. W. Dutton, and M. Noell. "Grid and Geometry Techniques for Multi-Layer Process Simulation". In: *Simulation of Semiconductor Devices and Processes, Vol. 5*. Ed. by S. Selberherr, H. Stippel, and E. Strasser. Vienna, Austria: Springer, 1993, pp. 417–420. ISBN: 9783709173725.

[134] N. V. Sapra et al. "Inverse Design and Demonstration of Broadband Grating Couplers". In: *IEEE Journal of Selected Topics in Quantum Electronics* 25.3 (2019), pp. 1–7. DOI: `10.1109/JSTQE.2019.2891402`.

[135] D. L. Scharfetter and H. K. Gummel. "Large-Signal Analysis of a Silicon Read Diode Oscillator". In: *IEEE Transactions on Electron Devices* 16.1 (1969), pp. 64–77. DOI: `10.1109/T-ED.1969.16566`.

[136] S. Selberherr. "Process and Device Modeling". In: *Process and Device Modeling*. Ed. by W. L. Engl. Advances in CAD for VLSI. North-Holland, 1986. Chap. 8. ISBN: 0444878912.

[137] J. R. Shewchuk. "Delaunay Refinement Algorithms for Triangular Mesh Generation". In: *Computational Geometry* 22.1-3 (2002), pp. 21–74. DOI: `10.1016/S0925-7721(01)00047-5`.

[138] H. Shichman and D. A. Hodges. "Modeling and simulation of insulated-gate field-effect transistor switching circuits". In: *IEEE Journal of Solid-State Circuits* 3.3 (1968), pp. 285–289. DOI: `10.1109/JSSC.1968.1049902`.

[139] W. Shockley and W. T. Read Jr. "Statistics of the Recombinations of Holes and Electrons". In: *Physical Review* 87.5 (1952), pp. 835–842. DOI: `10.1103/PhysRev.87.835`.

[140] W. Shockley. "The Theory of P-N Junctions in Semiconductors and P-N Junction Transistors". In: *The Bell System Technical Journal* 28.3 (1949), pp. 435–489. DOI: `10.1002/j.1538-7305.1949.tb03645.x`.

[141] F. Sischka. *Gummel-Poon Bipolar Model: Model Description and Parameter Extraction*. 2017. URL: `https://www.franz-sischka.de/Downloads;focus=CMTOI_de_dtag_hosting_hpcreator_widget_Download_17426528&path=download.action&frame=CMTOI_de_dtag_hosting_hpcreator_widget_Download_17426528?id=267927`.

[142] J. W. Slotboom. "Computer-Aided Two-Dimensional Analysis of Bipolar Transistors". In: *IEEE Transactions on Electron Devices* 20.8 (1973), pp. 669–679. DOI: `10.1109/T-ED.1973.17727`.

[143] T. N. Smith. "Data Driven Surrogate Models for Faster SPICE Simulation of Power Supply Circuits". M.Eng. thesis. Massachusetts Institute of Technology, 2021. URL: `https://dspace.mit.edu/handle/1721.1/139114`.

[144] C. Sodini. *The MOS Capacitor in Thermal Equilibrium*. 1998. URL: `http://web.mit.edu/course/6/6.012/SPR98/www/lectures/S98_Lecture6.pdf`.

[145] B. Speelpenning. "Compiling Fast Partial Derivatives of Functions Given by Algorithms". PhD thesis. University of Illinois at Urbana-Champaign, 1980. URL: `https://www.osti.gov/servlets/purl/5254402`.

[146] S. M. Sze, Y. Li, and K. K. Ng. *Physics of Semiconductor Devices*. 4th ed. John Wiley & Sons, 2021. ISBN: 9781119429111.

[147] T. A. Takhtaganov et al. "Optimization Under Uncertainty for the Shockley and the Drift-Diffusion Models of a Diode". In: (2014). Ed. by D. P. Kouri and M. L. Parks, pp. 165–174. URL: `https://www.sandia.gov/app/uploads/sites/136/2021/11/CCR2014.pdf`.

[148] Z. Teng, Y. He, and Q. Tong. "Generalized Scharfetter-Gummel Scheme Reducing the Cross-wind Effect for the Current Continuity Equation Including Energy Balance". In: *Computer Physics Communications* 79.2 (1994), pp. 190–200. DOI: `10.1016/0010-4655(94)90067-1`.

[149] K. K. Thornber. "Applications of Scaling to Problems in High-Field Electronic Transport". In: *Journal of Applied Physics* 52.1 (1981), pp. 279–290. DOI: `10.1063/1.328490`.

[150] E. Tiesinga et al. "CODATA Recommended Values of the Fundamental Physical Constants: 2018". In: *Reviews of Modern Physics* 93 (2 June 2021), p. 025010. DOI: `10.1103/RevModPhys.93.025010`.

[151] O. Triebl and T. Grasser. "Investigation of Vector Discretization Schemes for Box Volume Methods". In: *Technical Proceedings of the 2007 NSTI Nanotechnology Conference and Trade Show*. Vol. 3. 2007, pp. 61–64.

[152] O. Triebl and T. Grasser. "Vector Discretization Schemes Based on Unstructured Neighborhood Information". In: *2006 International Semiconductor Conference*. Vol. 2. 2006, pp. 337–340. DOI: `10.1109/SMICND.2006.284013`.

[153] T.-Y. Tso. "Pseudo Arc-Length Continuation Method for Multiple Solutions in One Dimensional Steady State Semiconductor Device Simulation". PhD thesis. Iowa State University, 1991. DOI: `10.31274/rtd-180813-9519`.

[154] H. Uecker, D. Wetzel, and J. D. M. Rademacher. *pde2path—A Matlab Package for Continuation and Bifurcation in 2D Elliptic Systems*. 2012. arXiv: `1208.3112 [math.AP]`.

[155] M. Utku and G. F. Carey. "Boundary Penalty Techniques". In: *Computer Methods in Applied Mechanics and Engineering* 30.1 (1982), pp. 103–118. ISSN: 0045-7825. DOI: `10.1016/0045-7825(82)90057-3`.

[156] W. Van Roosbroeck. "Theory of the Flow of Electrons and Holes in Germanium and Other Semiconductors". In: *The Bell System Technical Journal* 29.4 (1950), pp. 560–607. DOI: 10.1002/j.1538-7305.1950.tb03653.x.

[157] J. P. Vanderhaegen and R. W. Brodersen. "Automated Design of Operational Transconductance Amplifiers Using Reversed Geometric Programming". In: *Proceedings of the 41st Design Automation Conference, 2004*. 2004, pp. 133–138. DOI: 10.1145/996566.996608.

[158] R. S. Varga. *Matrix Iterative Analysis*. Springer, 2000. ISBN: 9783642051548.

[159] D. Vasileska. *Drift-Diffusion Model: Time-Dependent Simulations, Sharfetter-Gummel Discretization*. 2006. URL: https://nanohub.org/resources/1575/download/ddmodel_sg_tds_word.pdf.

[160] D. Vasileska. *Particle-Based Device Simulator Description*. 2008. URL: https://nanohub.org/resources/4551/download/emc_device_simulator_part_2.pdf.

[161] D. Vasileska and S. M. Goodnick. "Computational Electronics". In: *Materials Science and Engineering R: Reports* 38.5 (July 2002), pp. 181–236. ISSN: 0927-796X. DOI: 10.1016/S0927-796X(02)00039-6.

[162] D. Vasileska and S. M. Goodnick, eds. *Nano-Electronic Devices: Semiclassical and Quantum Transport Modeling*. 1st ed. New York, NY: Springer New York, 2011. ISBN: 1441988408.

[163] R. Veltz. *BifurcationKit.jl*. July 2020. URL: https://hal.archives-ouvertes.fr/hal-02902346.

[164] G. Wang et al. "Optimization and Performance Prediction of Tunnel Field-Effect Transistors Based on Deep Learning". In: *Advanced Materials Technologies* (2021), p. 2100682. DOI: 10.1002/admt.202100682.

[165] R. E. Wengert. "A Simple Automatic Derivative Evaluation Program". In: *Communications of the ACM* 7.8 (1964), pp. 463–464. DOI: 10.1145/355586.364791.

[166] M. B. Yelten, P. D. Franzon, and M. B. Steer. "Surrogate-Model-Based Analysis of Analog Circuits—Part I: Variability Analysis". In: *IEEE Transactions on Device and Materials Reliability* 11.3 (2011), pp. 458–465. DOI: 10.1109/TDMR.2011.2160062.

[167] P. D. Yoder et al. "Optimized Terminal Current Calculation for Monte Carlo Device Simulation". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 16.10 (1997), pp. 1082–1087. DOI: 10.1109/43.662672.

[168] H. You et al. "Kriging Model Combined With Latin Hypercube Sampling for Surrogate Modeling of Analog Integrated Circuit Performance". In: *2009 10th International Symposium on Quality Electronic Design*. 2009, pp. 554–558. DOI: 10.1109/ISQED.2009.4810354.

[169] L. Yu et al. "Inverse Design of High Absorption Thin-Film Photovoltaic Materials". In: *Advanced Energy Materials* 3.1 (2013), pp. 43–48. DOI: 10.1002/aenm.201200538.

[170] Z. Yu, B. Ricco, and R. W. Dutton. "A Comprehensive Analytical and Numerical Model of Polysilicon Emitter Contacts in Bipolar Transistors". In: *IEEE Transactions on Electron Devices* 31.6 (1984), pp. 773–784. DOI: 10.1109/T-ED.1984.21606.

[171] Y.-L. Zhang et al. "Fully Boron-Sheet-Based Field Effect Transistors from First-Principles: Inverse Design of Semiconducting Boron Sheets". In: *The Journal of Physical Chemistry Letters* 12.1 (2020), pp. 576–584. DOI: 10.1021/acs.jpclett.0c03333.

[172] R. Zhang et al. "Inverse Design of FinFET SRAM Cells". In: *2020 IEEE International Reliability Physics Symposium (IRPS)*. 2020, pp. 1–6. DOI: 10.1109/IRPS45951.2020.9129530.