# DrIFtFUSION v1.0 Release Notes

Philip Calado
p.calado13@imperial.ac.uk
Department of Physics
Imperial College London
January 2018

## 1 Introduction

DrIFtFUSION is a versatile *p*-*i*-*n* solar cell simulation package. The code uses MATLAB's built-in Partial Differential Equation solver for Parabolic and Elliptic equations (PDEPE) to solve the continuity and Poissons equations for electron density $n$, hole density $p$, a positively charged mobile ionic charge density $a$, and the electrostatic potential $V$ as a function of position $x$ and time $t$. While the methodology used by the solver for discretising the equations goes beyond the scope of this work, full details can found in Skeel and Berlizns 1990.[**?** ]

## 2 Simulating open and closed circuit conditions

For closed circuit simulations, where the current output of the device is of interest (for example for current-voltage $J$-$V$ scans) a *p*-type/intrinsic/*n*-type (*p*-*i*-*n*) architecture was used. Figure 1a shows the typical structure with position labels at the interfaces and boundaries.

At open circuit the cell is disconnected from the external circuit, resulting in zero current for carriers. In conventional, fixed potential boundary condition simulations,the open circuit voltage is found by using an iterative convergence method (such as the

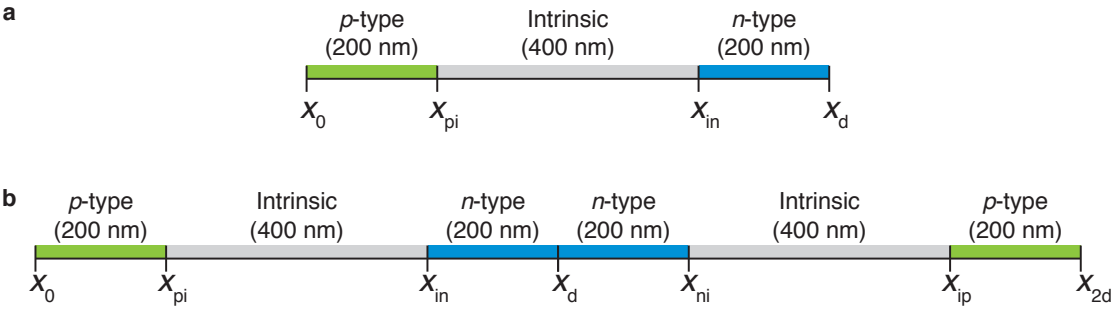


Figure 1: **Simulated device schematics.** Device schematics with position labels and layer thicknesses for (**a**) the fixed potential boundary condition *p*-*i*-*n* simulation used for JV and current transient measurements, and (**b**) the mirrored *p*-*i*-*n*-*n*-*i*-*p* cell used for simulating open circuit conditions.

Newton-Raphson method) to find the potential at which the current is zero. Simulating voltage transients in this way is computationally expensive; In order to accelerate calculation times and enable direct readout of the open circuit voltage $V_{\mathrm{OC}}$, the method of image charges was used to devise a symmetric *p-i-n-n-i-p* cell (see figure 1b).

## 3 Charge transport

Charge carrier transport in semiconductors can be well described by the superposition of:

1. *Drift currents* arising from the electrostatic response of charges to the electric field $\mathbf{E}$.
2. *Diffusion currents* arising from carrier concentration gradients $\partial n/\partial x$, $\partial p/\partial x$, and $\partial a/\partial x$.

Mobile ionic charge has been assigned a positive charge in the simulation and therefore follows the same direction as holes. Using the Einstein relation $D = \mu k_{\mathrm{B}} T/q$ to express the diffusion coefficient in terms of mobility, the electron $\mathbf{J}_n$, hole $\mathbf{J}_p$, and ion $\mathbf{J}_a$ current densities can be written as:

$$\mathbf{J}_n = q\mu_n n\mathbf{E} + \mu_n k_{\mathrm{B}} T \frac{\partial n}{\partial x} \tag{1}$$

$$\mathbf{J}_p = q\mu_p p\mathbf{E} - \mu_p k_{\mathrm{B}} T \frac{\partial p}{\partial x} \tag{2}$$

$$\mathbf{J}_a = q\mu_a a\mathbf{E} - \mu_a k_{\mathrm{B}} T \frac{\partial a}{\partial x} \tag{3}$$

where $k_{\mathrm{B}}$ is the Boltzmann constant, $T$ is temperature and $q$ is the electronic charge. This work takes the convention that electrons flowing right and holes flowing left produce a negative current.

## 4 The Continuity equations

The continuity equations are a set of 'book keeping' equations that ensure conservation of charge in the system. They describe how the density of charge carriers changes as a function of position and time. In the most general form, for a concentration $\varphi$ with flux $\mathbf{j}_\varphi$, and source $\Lambda$:

$$\frac{\partial \varphi}{\partial t} - \nabla \mathbf{j}_{\varphi} - \Lambda = 0 \tag{4}$$

For solar cells, in the simplest case, the source term $\Lambda$ is composed of two parts:

1. Generation $G$ of electronic carriers by photoexcitation.
2. Recombination $U$ of electronic carriers through radiative and non-radiative pathways.

In DrIFtFUSION, mobile ions are considered inert and as such the source term for ions is zero. In one dimension the continuity equations for electrons, holes and ions are:

$$\frac{\partial n}{\partial t} = \frac{1}{q}\frac{\partial \mathbf{J}_n}{\partial x} + G - U \tag{5}$$

$$\frac{\partial p}{\partial t} = -\frac{1}{q}\frac{\partial \mathbf{J}_p}{\partial x} + G - U \tag{6}$$

$$\frac{\partial a}{\partial t} = -\frac{1}{q}\frac{\partial \mathbf{J}_a}{\partial x} \tag{7}$$

## 5 Poisson's equation

Poisson's equation relates the electrostatic potential to the space charge density $\rho$ as a function of position. In the simulation the space charge density is defined by the sum of the mobile and static charge densities. Doping is achieved via the inclusion of fixed charge densities terms for donors, $N_{\mathrm{D}}$ and acceptors, $N_{\mathrm{A}}$, which effectively generate mobile counter charges. In this work ionic defects are modelled as Schottky defects in which each charged vacancy has an oppositely charged counterpart (see subsection **??**). In the simplest approximation, the counter ions are described as being static with uniform charge density of $N_a$. Together these terms yield the following expression:

$$\frac{\partial V^2}{\partial x^2} = -\frac{\rho}{\varepsilon_0 \varepsilon_r} = -\frac{q}{\varepsilon_0 \varepsilon_r}(p - n + a + N_{\mathrm{A}} - N_{\mathrm{D}} - N_a) \tag{8}$$

where $\varepsilon_0$ is permittivity of free space and $\varepsilon_{\mathrm{r}}$ is the relative permittivity of the medium. The electric field $\mathbf{E}$ can easily be obtained from the negative gradient of the electrostatic potential:

$$\mathbf{E} = -\frac{\partial V}{\partial x} \tag{9}$$

# 6 Getting started

## 6.1 DrIFtFUSION procedures

There are a number of procedures in the DrIFtFUSION package- here is a brief overview of what they do:

1. *pinParams*: Here, the user defines the parameters for the simulation e.g. layer thickness, mobilities etc. The output is a parameters structure.
2. *pindrift*: This is the core procedure which calls the MATLAB PDEPE. The continuity, transport equations, boundary conditions, and initial conditions can all be accessed and modified here.
3. *pinAna*: Takes the solution from pindrift, or a user input solution, and outputs calculated values for $V_{\mathrm{OC}}$, currents etc.
4. *meshgen_x*: Generates the position mesh
5. *meshgen_t*: Generates the time mesh
6. *symmetricise*: Generates a mirrored cell solution for open circuit. conditions used the equilibrium solution from closed circuit boundary conditions (BCs)
7. *mobsetfun*: A function for quickly switching on and off mobilities
8. *equilibrate*: Runs the solver multiple times from analytical solutions to generate open and closed equilibrium solutions.
9. *doJV*: Uses the closed circuit equilibrium solutions to run current voltage scans
10. *doTPV*: Uses the open circuit equilibrium solutions to perform a transient photovoltage measurement

## 6.2 Installation

Make sure all of the above procedures are in the active MATLAB folder.

Before starting you will need to install the *v2struct* MATLAB toolbox, available from:

https://uk.mathworks.com/matlabcentral/fileexchange/31532-pack---unpack-variables-to---from-structures-with-enhanced-functionality?focused=3847342&tab=example

Click the 'Download' drop-down menu and select toolbox. Download and then install it in MATLAB. v2struct allows variables to be easily packed and unpacked into structures.

## 6.3 pindrift input arguments

*pindrift* uses adaptable input arguments to allow the user to both easily control the simulation from the command line in combination with adjusting parameters in *pinParams*, as well as coding long multi-step procedures (see *doJV* and *doTPV* for examples). Details of how the arguments are dealt with by *pindrift* can be found in the code but are reproduced here for clarity.

1. If no input argument is given, the code uses the parameters described in pinParams and an analytical approximation to intial conditions. In this case the first solution must always be made with mobilities switched off.

2. If a previous solution is used as an input parameter, e.g.:

   `sol = pindrift(sol_eq)`

   then *pindrift* will use the previous solution as initial conditions and the parameters are taken from the *pinParams* function.

3. If a previous solution and a string are used as input arguments e.g.
   `sol = pindrift(sol_eq, 'p')`

   then *pindrift* will use the previous solution as initial conditions and the parameters from the same solution. Clearly in this case changing parameters in *pinParams* will not have any affect on the simulation.

4. Finally, if a previous solution followed by the name of a parameters structure are used as input arguments e.g.
   `sol = pindrift(sol_eq, params)`

   then *pindrift* will use the previous solution as initial conditions and the parameters the named parameters structure. This allows multi-step functions to be designed and parameters to be adjusted throughout the procedure.

## 6.4 Quick start guide

1. Set the desired parameters for the device in *pinparams*. All the parameters are commented with their meaning in the script. It is highly recommended that you start with the current parameter set as all the functionality has been tested using these parameters. In particular the boundary conditions rely on the depletion region of the contact materials being thinner that their width. Less doped contacts or higher dielectric constants in the contact may lead to inconsistent boundary conditions (BCs). Future versions of the code will seek to address self-consistency of the BCs by using an adaptive built in potential.

2. Type:

   `[sol_eq, sol_i_eq, ssol_eq, ssol_i_eq] = equilibrate`

   The *equilibrate* procedure will run and produce 4 equilibrium solutions:

   i. `sol_eq` Closed circuit equilibrium without ion migration

   ii. `sol_i_eq` Closed circuit equilibrium with ion migration

   iii. `ssol_eq` Open circuit equilibrium without ion migration

   iv. `ssol_i_eq` Open circuit equilibrium with ion migration

   During the procedure various figures will appear. The user should verify that the current and voltage are reasonably low (e.g. $\mathbf{J} \approx$ nA and $V \approx \mu$V). If the current is not zero, either increase the doping density or reduce the dielectric constant of the contact regions. This will reduce the depletion with and restore the BCs to a self-consistent state.

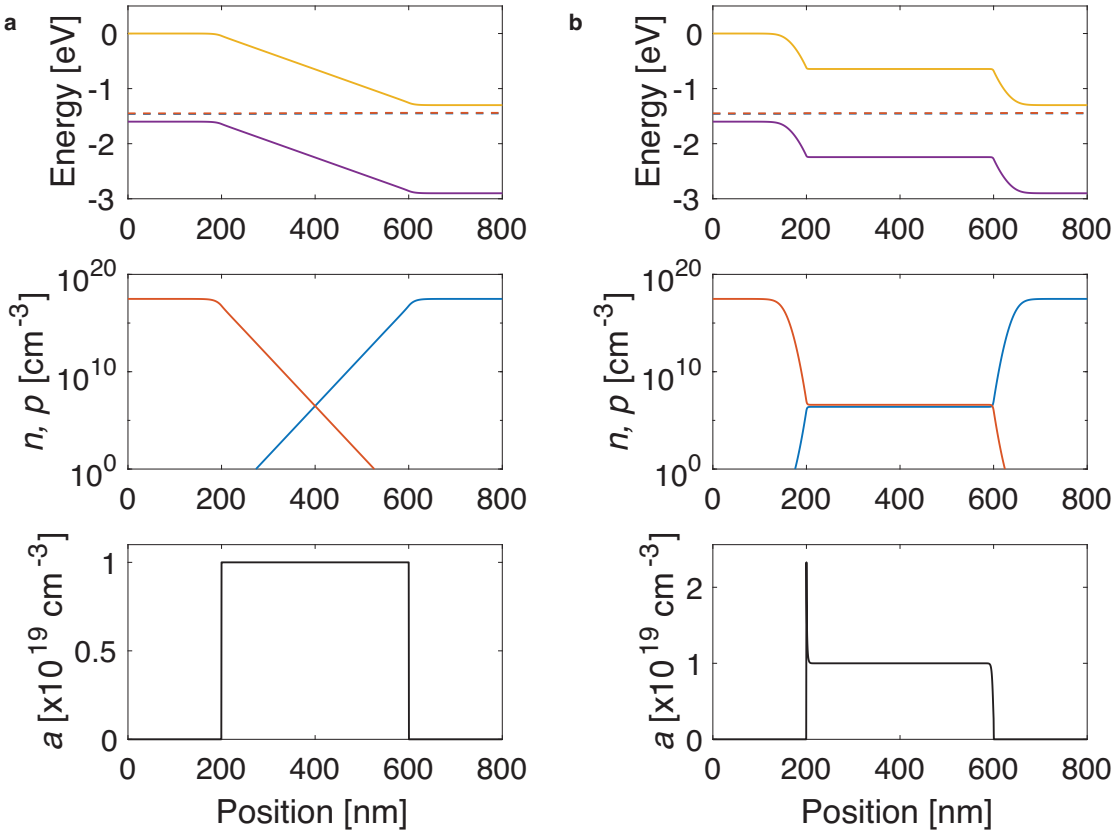


Figure 2: **Equilibrium energy level diagrams and charge densities.** Equilibrium energy level diagrams (top), electron $n$ and hole $p$ densities (middle), and mobile ionic charge $a$ density (bottom) for (**a**) a device with static ions, and (**b**) mobile ions.

Figure 2a shows the energy level diagrams and charge densities for electrons, holes and mobile ionic charge that should appear in multi-panel figure (MATLAB Figure 1) for the device before ions have migrated. Figure 2b shows the device at

equilibrium after the ion mobility has been switched on. In this instance mobile ionic charge has accumulated at the $p$-type/intrinsic interface ($x = 200$ nm) and is depleted from the intrinsic/$n$-type interface ($x = 600$ nm). The energy level diagram shows that the field has been screened out in the intrinsic layer by the ionic charge.

The solutions produced by *equilibrate* can now be used as the starting conditions for *pindrift*. The solution structure contains:

i. A matrix called sol with 4 layers. The layers contain the solutions as follows:
   Layer 1: $n$ electron charge density
   Layer 2: $p$ hole charge density
   Layer 3: $a$ ion charge density
   Layer 4: $V$ electrostatic potential

ii. The position mesh, $x$

iii. The time mesh, $t$

iv. The parameters structure *params*

v. Other variable outputs such as $V_{\mathrm{OC}}$ and **J**

3. First let's look at closed circuit conditions. Let's try running a dark $JV$ scan without any mobile ions. In *pinparams*, under the `General Parameters` section, change the $JV$ parameter to:

   ```
   JV = 1;
   ```

4. At the command line type:

   ```
   JV_dk = pindrift(sol_eq);
   ```

   Note that solving JV scans requires the current to be calculated at every point and is therefore much slower than

5. Now let's try illuminating the device. First switch off the $JV$ function:

   ```
   JV = 0;
   ```

   In *pinparams*, under the `General Parameters` section, change the light intensity parameter:

   ```
   Int = 1;
   ```

   This will introduce a uniform generation profile of approximately equivalent to 1 Sun intensity.

6. At the command line type:

   ```
   sol_1Sun = pindrift(sol_eq);
   ```

   The energy level diagram should now show quasi Fermi level splitting and there should be a current density of approx. 16 mAcm$^{-2}$ provided that the default active layer thickness of 400 nm has been used.

7. Now let's use the illuminated solution to run a light $J$-$V$ curve. Switch on the $J$-$V$ option again:

   In pinParams:
   ```
   JV = 1;
   ```

8. At the command prompt type:

   ```
   JV_1sun = pindrift(sol_1Sun);
   ```

9. To obtain a hysteresis curve, first switch off the light and the JV scan:

   In pinParams:
   ```
   Int = 1;
   JV = 0;
   ```

10. Now switch on interfacial recombination by reducing the SRH time constants:

    In pinParams:
    ```
    taun_etl = 1e-11;
    taup_etl = 1e-11;
    taun_htl = 1e-11;
    taup_htl = 1e-11;
    ```

11. Create a new equilibrium solution for this new parameter set as follows:

    ```
    sol_i_eq_SR = pindrift(sol_i_eq);
    ```

12. Illuminate the device and record the new solution:

    In pinParams:
    ```
    Int = 1;
    ```

    Command line:
    ```
    sol_i_1sun_SR = pindrift(sol_i_eq_SR);
    ```

    You should obtain an energy level diagram identical to the one shown in Figure 3 and an open circuit voltage of 0.47 V.

13. Now run a forward $J$-$V$ scan using the surface recombination solution. By default the ion mobility is set to $10^{-8}$ $\mathrm{cm^2V^{-1}s^{-1}}$ and the scan rate is set to 130 $\mathrm{Vs^{-1}}$. Slower ion mobilities may lead to instabilities in the solver so it is recommended that the user experiments using the default mobility value and transforms the time mesh appropriately.

    In pinParams:
    ```
    JV = 1;
    ```

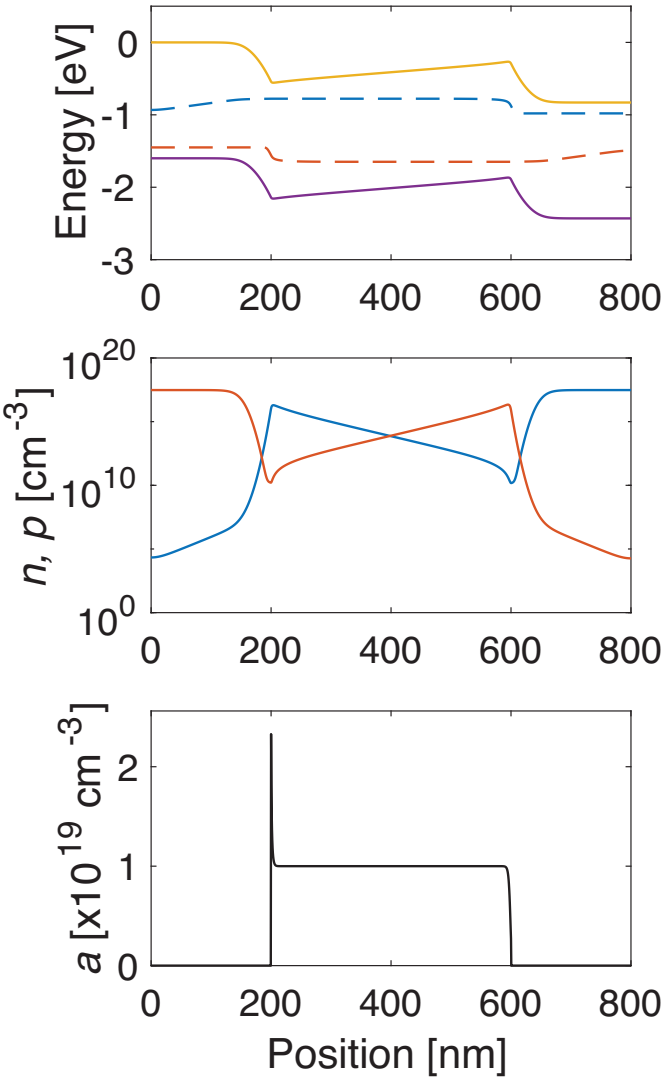


Figure 3: **Equilibrium energy level diagrams and charge densities for illuminated device.** Energy level diagrams (top), electron $n$ and hole $p$ densities (middle), and mobile ionic charge $a$ density (bottom) for a device with mobile ions and surface recombination under 1 Sun eq. illumination.

Command line:

```
JV_i_f = pindrift(sol_i_1sun_SR);
```

Select Figure(11) and type:

```
hold on
```

at the command prompt.

14. Now run the reverse scan. First in the $J$-$V$ scan settings change the start and end voltages:

    In pinParams:

    ```
    Vstart = 1.3;
    ```

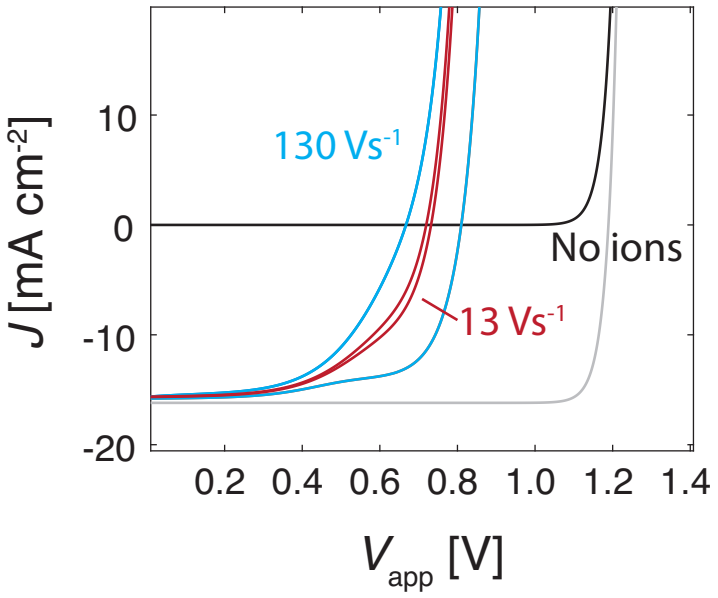


Figure 4: **Example current-voltage scans.** Scans without ions or surface recombination (black and grey curves) and hysteresis curves for devices with mobile ions at 13 $Vs^{-1}$ (red) and 130 $Vs^{-1}$ (blue).

```
Vend = 0;
```

Command line:

```
JV_i_r = pindrift(JV_i_f);
```

15. It is recommended to reset the current-voltage settings to default to avoid problems later:

    In pinParams:

    ```
    Vstart = 0;
    Vend = 1.3;
    ```

    You should obtain $J$-$V$ curves identical to those shown in Figure 4.

    As an exercise the user is encouraged to change the scan rate to 13 $Vs^{-1}$. and repeat steps $13 - 15$.

## 7 Example procedures

In addition to *equilibrate* two procedures are included in the package to help the user understand how to easily code their own routines by dynamically adjusting the simulation parameters.

### 7.1 doJV

The function *doJV* is included as an example procedure to illustrate how to automate transient measurements by breaking them into individual parts and using appropriate initial conditions for each solution. To run the function at the command line type:

```
[JV_dk_f, JV_dk_r, JV_1S_f, JV_1S_r] = DoJV(sol_ini, JVscan_rate, mui,
Vstart, Vend)
```

`sol_ini`, `JV_scan_rate`, `mui`, `Vstart`, and `Vend` are user defined arguments that the user must define- the arguments are described at the beginning of the code but should be self-explanatory. As an example:

```
[JV_dk_f, JV_dk_r, JV_1S_f, JV_1S_r] = DoJV(sol_i_eq_SR, 130, 1e-8, 0, 1.3)
```

would produce the same set of results as in the introductory exercise. Note here that the solution with surface recombination is used as the input argument. Using `sol_i_eq` would yield a *J-V* without hysteresis. Please refer to the reference given in Section 10 for a discussion of the underlying principles.

### 7.2 doTPV

*doTPV* uses the mirrored cell open circuit equilibrium solution to perform a transient photovoltage (TPV) measurement on the simulated device. This example shows how different steps can be separated out. In particular the cell is first illuminated, after which the ion mobility is switched on.

## 8 Common problems

### 8.1 Unable to meet integration tolerances... error

One of the most common errors that occurs with the PDEPE is the following error:

```
Warning:  Failure at t=4.060671e-03.  Unable to meet integration
tolerances without reducing the step size below the smallest value
allowed (1.387779e-17) at time t.
```

This is a general error that can occur due to incorrect boundary conditions, non-physical transport equations, bad initial conditions etc.

*If the simulation without modification then the most common cause for this error is a time step (i.e. the value of* `tmax`*) that is too large.* Try breaking the solution down into smaller parts with a very short initial time step. An example of this can be found at

the end of equilibrate function where two solutions are used to reach equilibrium with mobile ions.

It is important to remember when writing procedures to change the initial time value `t0` in addition to `tmax`.

### 8.2 Large file sizes

The solution structure can become quite large and use up a lot of space. It is recommended that when users become familiar with coding the simulation that only the solutions that are required are saved with procedures recorded that enable reproduction of the results at a later date.

V2struct can cause large file sizes if solutions accidentally become stored in the parameters structure. Be very wary when using this function and check the solution params structure for instances of variables or structures that should not be present within it.

## 9 Known issues

### 9.1 Charging at open circuit

Due to the large mesh size, numerical errors can cause a charge build up at the $p$-type boundary. An error check for this will be included in future versions.

Alternatively, this problem can also arise if mesh points do not exist at interfaces. The PDEPE solver can deal with discontinuities provided that a mesh point exists at the position of the discontinuity. *When defining new meshes ensures that mesh points exist at interfaces.*

### 9.2 Total current calculation

Total/net current calculations are inaccurate in space charge regions. At present the median value of the current is used. Where space charge regions are particularly large this could lead to errors in the current calculations.

### 9.3 Fast JV scans

Very fast JV scans with mobile ions currently cause numerical problems.

## 10 Citing this code

Please cite the following reference when publishing work that results from this code:

Philip Calado, Andrew M Telford, Daniel Bryant, Xiaoe Li, Jenny Nelson, Brian C. O'Regan, and Piers R.F. Barnes. Evidence for ion migration in hybrid perovskite solar cells with minimal hysteresis. *Nature Communications*, 7, 2016.

## 11 Version history

v1.1 Removed dependence on V2Struct - Packing and repacking the parameters structure can lead to accidental packing of solutions and therefore it is safer to use structure syntax. Unfortunately the code does not look as nice!

$$F = \frac{{}_1Q_2}{\epsilon_0 \epsilon_r d^2}$$

$$j_{drift} = n\mu \frac{\partial V}{\partial x}$$

$j_{drift}$ = Flux due to drift
$n$ = Density of electrons
$V$ = Electrostatic potential
$x$ = Position