FSEM
Manual

(under construction)
http://www.fsem.org

Fabian Dortu (Fabian.Dortu@imec.be),
Katholieke Universiteit Leuven,
ESAT/INSYS/IMEC,
75 Kapeldreef,
3001 Heverlee

August 16, 2006

Contents

List of Tables

Chapter 1

Introduction

FSEM is a set of freeFEM++ scripts and C++ code to solve the drift-diffusion (DD) semiconductor device equations by the finite element method (FEM). It was initially developed for the nonlinear study of semiconductors under high optical injection.

1.1 History

I started developing FSEM in the frawework of my PhD thesis in order to compute the excess carrier concentration and temperature in a homogeneous semiconductor under intense laser light illumination taking into account nonlinear recombination and diffusion mechanisms. Although the considered domain was uniform, the huge differences in the generation (laser) and diffusion characteristic lengths implied the use of an nonuniform mesh. Furthermore, I did not want to restrict FSEM to homogeneous substrate only so I envisaged to develop a full drift-diffusion solver. Since developing from scratch a full drift-diffusion solver on a noniform mesh is a rather heavy task, I decided to use FreeFEM++, a C++ based high-level programming language to solve Partial Differential Equations (PDE). This language would allow we to focus on physical (Poisson equation) and algorithmic (nonlinear solver) aspects only, putting numerical aspects (mesh generation, matrix assembling, linear solver) aside.

1.2 Installation

- Obtain the last subversion revision from SourceForge.net:

  ```
  svn co https://svn.sourceforge.net/svnroot/fsem fsem
  ```

- Go to FSEM's root directory and run 'make':

  ```
  cd fsem/fsem; make;
  ```

- The C++ modules are now compiled and the include paths are adapted to your system. If you relocate FSEM, you will have to run 'make' again. To include FSEM in a FreeFEM++ script file, use

  ```
  include "FSEMDIR/fsem_main_Include.edp"
  ```

1.3 Features

11 Aug 2006

- Carrier diffusion and drift-diffusion equations:

  - Diffusion equation solver for the excess carriers (excess electron and hole concentrations are equal, no electric field). Newton's method is used when the equation is nonlinear. The steady, time dependent, and steady-periodic equations are implemented. For the steady-periodic case (harmonic excitation), the complex equation is linearized but pseudo-nonlinear formulations are also implemented (so far only the fondamental harmonic response is implemented, but the second-harmonic should be implemented soon).
  - Drift-Diffusion equation solver using Slotboom's variables formulation and Gummel's map (Newton's method for Poisson's equation and two linearized equations for the electron and hole concentrations). It is implemented in a stand-alone FreeFEM++ script but not yet incorported in FSEM.
  - Models (coefficients appearing in the model's equation can be position dependent when it make sens)
    * Mobility models: Constant (but position dependent), Dorkel-Letruq and Philips Unified Mobility (Klaassen).
    * Band gap narrowing models: Varshini (temperature), Slotboom (doping), Shenk (doping and free carriers).
    * Recombination models: Linear, SRH (doping dependent), Auger.
    * Generation models: Optical generation by Gaussian laser source(s)
    * Absorption model: User specified absorption coefficients and Smith's model (band-to-band indirect absorption)
    * Diffusivity: Constant (but position dependent), computed from mobilities using Boltzmann (=Einstein's relation) or Fermi-Dirac statistics.
    * Ambipolar diffusivity: Constant (but position dependent) or computed from electron and hole mobilities.

- Heat equation

  - Diffusion equation solver for the temperature. The steady, time dependent, and steady-periodic equations are implemented.
  - Models (coefficients appearing in the model's equation can be position dependent when it make sens)
    * Thermal diffusivity and conductivity: Constant (but position dependent)
    * Source term: User provided, excess (relative to the bandgap) laser power density (hot-carriers), and carrier recombinations.

Chapter 2

Getting Started

2.1 Introduction

Note that FSEM is still under development. The documentation is very synthetic and some FE formulations are still under testing. If you know FreeFEM++ quite good and if you need to solve problem's that commercial software cannot and if you have the courage to dive into the code, FSEM might be a good framework.

However FSEM is designed to be used both by entry-level users and advanced-users who want to implement new models, equations or equation formulations. A typical simulation input file may look like this (note that FSEM "live" on top of FreeFEM++ such that all FreeFEM++ features remain available):

- Generate the geometry and initial mesh (2D rectangular or 3D axisymmetric coordinates)
- Define contacts (thermal and electrical)
- Define models by mean of flags.
- Solve the equation(s) (either diffusion or drift-diffusion for carriers and diffusion for heat)
- Adapt the mesh
- Solve the equation(s) again
- Plot solution, export it to gmsh or extract values along a line. Not only the solution, but all derived quantities can be plot (band gap, recombinations, ...)

Note that the geometry or the contacts can be redefined at any time.

2.2 Detailed description

- **Include fsem**. Include the main fsem include file `FSEMDIR/fsem_main_Include.edp` in your Freefem++ script. You might also want to change the verbosity of fsem by setting `fsemVERBOSITY` to a value comprised between 0 and 10:

```
include "../fsem_main_Include.edp";
fsemVERBOSITY = 10;
```

- **Geometry and mesh definition** (see Sec. 3.7). The mesh has to be assigned to the global variable `GloMesh` (See Tab. 3.13 for more details). It must be fine enough to correctly sample the a priori known optical excitations (laser radius, absorption length) and material properties (doping profile, SRH coefficients, ...). In case mesh refinement is performed, keep in mind that user defined fields (like the doping profile) should be re-interpolated on the new mesh in order to have a good sampling. A basic mesh can be defined as follow (see FreeFEM's doc for full details):

```
/* Create a square mesh with top left corner at (0,0)
 * and bottom right corner at (50, -60) m.
 */
real r1 =  50e-6;  // [m]
real y1 = -60e-6;  // [m]

/* the mesh is made adimensional by dividing by GloScaX:
 * (I admit this is annoying to make the adimensionalization by hand.
 *  This will be hiden from the user in next releases)
 */
real r1Adim = r1/GloScaX;
real y1Adim = y1/GloScaY;

int n=25, m=20;        // 25 mesh point along the x direction, 20 along the y direction

GloMesh = square(n, m, r1Adim*x, y1Adim*y]);
```

Note that the mesh has to be adimensional. The adimensionalization factors, `GloScaX` and `GloScaY` are equal by default and set to values that should be appropriate for most problems.
In this example, the top side of the sample is at y=0 and the bottom side is at y=-60 m. This in principle is not absolutely mandatory; an arbitrary shaped and oriented sample could be defined. However, in the current implementation, light will be assumed to enter the sample at (0,0) towards y negative so that the top of the domain close to r=0 should be y=0.
If non natural (in the sens of a weak form formulation) boundary conditions have to be specified, labels have to be attributed to the borders. This can be done as follow (see Freefem's doc for full details):

```
real r1=50e-6/GloScaX;
real y1=-60e-6/GloScaY;
border topRight(t=r1, 0.3*r1) { x=t; y=0; label=100;} /* we arbitrary choose 100 as label number. */
border topLeft(t=0.3*r1, 0) { x=t; y=0;}
border inner(t=0,y1) {x=0; y=t;}
border bottom(t=0,r1) {x=t; y=y1; label=100;}         /* we choose the same label as previously
                                                       * because the boundary condition
* will be identical.
*/
 border outer(t=y1, 0) {x=r1; y=t;}

 // The meshed domain is at the left hand side of the border
 GloMesh = buildmesh(topRight(22) + topLeft(22) + outer(25) + bottom(25) + inner(25));
```

- **Boundary conditions definition**. By default, the boundary conditions are natural (homogeneous Neumann). If one wants to specify Dirichet conditions, the global arrays `GloCarrContactLabel[]` and `GloCarrContactEXC[]` (Tab. 3.16) have to be set accordingly. The temperature contacts are treated similarly (Tab. 3.17). The syntax is

```
GloCarrContactLabel[ContactNumber] = BorderLabel;
GloCarrContactEXC[ContactNumber] = DimensionnalFieldValue;
```

A maximum of 10 contacts can be defined (`ContactNumber=0-9`). For instance:

```
GloCarrContactLabel[0] = 100;
GloCarrContactEXC[0] = 0; /* set ohmic contact (equilirium = 0 excess carrier) */
```

- **Model flags definition** (see Sec. 3.4). There are two type of flags. The first category of flags can be set either OFF (0) or ON (1 or any positive value). For example, the generation/recombination mechanisms (Auger, SRH, Direct) are turned ON or OFF with the following statement:

```
ModUsrRecAuger = 1;    // turn ON Auger mechanism
ModUsrRecAuger2 = 0;   // turn OFF Auger2 mechanism
ModUsrRecSRH = 1;      // turn ON SRH mechanism
ModUsrRecDirect = 0;   // turn OFF direct recombinations mechanism
```

Note that one can cumulate several gen/rec processes as long as it make sens. For instance, it is perfectly correct to have Auger, SRH and direct gen/rec occurring together. However, it makes no sens to turn on both `ModUsrRecAuger` and `ModUsrRecAuger2` because they just represent the same physical process although the model is not exactly the same.

The second category of flags can take several predefined integer values. For example, the flags that controls the mobility model can take up to (so far) three different values:

```
ModUsrMob = ModCSTMobCst;      // Constant mobility model
ModUsrMob = ModCSTMobDORLET;   // The Dorkel and Leturcq mobility model
ModUsrMob = ModCSTMobPHUNI;    // The Phillips Unified Mobility model
```

Obviously, only one mobility model can be chosen at a time.

- **External excitations definition** 3.1. So far, only optical excitations are implemented. The light beam falls orthogonally on the samples surface at r =0 and y=0 (toward y negative). The beam has a Gaussian shape and is attenuated as it penetrates into the sample. A maximum of 2 lasers is allowed (although more lasers can easily be added by changing a variable in the code).

```
ParUsrLaserN = 1;                          /* number of lasers */
ParUsrLaserE[0] = 1.24 / 0.980;            /* energy [eV] */
ParUsrLaserI[0] = 0.806e6 * ConvC2M^(-2);  /* Irradiance [W m-2] */
ParUsrLaserRefl[0] =  0.303;               /* Reflectance [adim] */
ParUsrLaserR[0] = 1.5 * ConvU2M;           /* Radius [m] */
```

Note that we made used to the convenient predefined variable `ConvC2M` which equals 10^{-2}. More variables like this are available (see Tab.).

- **Time dependent external excitations definition** Time dependent excitations (in view of a time dependent resolution, obviously) are a bit more difficult to handle. Actually, we will see that not only excitations can be make time dependent but material parameters as well. *Before* each time step calculation, a user defined function will be called. An arbitrary name can be given to the function as it will be passed as argument to the solver function. For instance, a harmonic excitation is define as follows:

```
func real MyTimeDepParFunction(real t) {
  /*     Update everything that changes with time (t [s])
  *      (and that is not automatically accounted by a model).
  *      This function will be passed as argument
  *      to the SolveCarrLinearTimeDep() macro.
  *
  *     For instance:
  *       - external excitation (laser).
  *       - superficial charges? (charging effect due to illumination).
  *       - material parameters? (Fluorine outdiffusion leading to enhanced carrier activation).
  *       - anything your mind can imagine.
  *
  */
  cout << "Time is: " << t  << "[s]" << endl;
  real omega = 2* ParCstPi * MyFreq; // [Hz]
```

```
  ParUsrLaserI[0] = 0.379e6 * ConvC2M^(-2) * 0.5*(1 - cos(omega * t)); // [W m-2] Pump laser

  return 0.0;
}
```

- **The equation resolution**. As of today, it is possible to solve for the excess carrier concentration and the temperature, in the steady state, steady periodic (harmonic excitation) and time dependent regime, with both linear and nonlinear (Newton) resolution.

```
SolveCarrNonlinearSteady();
```

  In the case of a time dependent excitation, two user functions and one vector have to be defined:

  - `func real MyTimeDepParFunction(real t)` (the function name can be arbitrary (will be passed as argument)): update time-dependent excitations (and eventually material parameters). This function is called *before* `SolveWhatheverTime` function is called. An example of `MyTimeDepParFunction(real t)` was given previously.
  - `func real MySaveFunction(real t)` (the function name can be arbitrary (will be passed as argument)): make post-processing after each time step (save or plot data). This function is called *after* `SolveWhatheverTimeStep` function is called.
  - `real[int] GloVarTimeVector(nSteps)`: time in seconds at which to make the calculations

```
func real MyTimeDepParFunction(real t) {
  // see Time dependent external excitations definition.
  ...
}

func real MySaveFunction(real t) {
  plot(GloVarSemiEXC, wait=0);  //  plot the excess carrier at each time step
}

real[int] GloVarTimeVector(3);
GloVarTimeVector[0] = 0.0;    // [s]
GloVarTimeVector[1] = 1e-6;
GloVarTimeVector[2] = 2e-6;

GloVarSemiEXC = 0;            // Set initial condition
SolveBDF1fixTime(SolveCarrNonlinearTimeStep, MyTimeDepParFunction, MySaveFunction);
```

- **Results visualization** Freefem has a build in `plot` function which can be used for fast control of the result. Fsem also implements routines to output the results to `gmsh` (Tab. 3.19), a free three-dimensional (although we only use two dimensions) finite-element visualization software. It is also possible to make 1d cross section (Tab. 3.18). For example:

```
plot(GloVarSemiEXC, wait=1); // Plot the excess carrier
GloSaveGmshPosSPARSEDST(GloMesh, GloVarTempT, 1, "GloVarTempT", "filename.pos", ConvM2U, GloScaX, GloScaY);
```

Chapter 3

Examples

Real life examples, with automated data extraction, are available in `FSEMDIR/doc/examples/`. Full working examples, but simple, are printed hereafter and are available in `FSEMDIR/doc/examples-tutorial/`.:

3.1 Linear carrier diffusion equation

Freefem++ FSEMDIR/doc/examples/001a_CarrLinear.edp

```
/*
 * Solve the linear steady state diffusion equation for the excess carriers.
 * Cylindrical coordinates.
 * One laser.
 */


include "../fsem_main_Include.edp";
fsemVERBOSITY = 10;
fsemModUseCpp = true;

/*************************************************************
 * Define the mesh. The mesh must be adimentional (I will try to
 * make that automatice in the future)
 *************************************************************/
//real r0=0, r1=50e-6/GloScaX;
//real y0=0, y1=-60e-6/GloScaY;
//int n=25, m=20;
//GloMesh =square(n,m,[r0+(r1-r0)*x,y0+(y1-y0)*y]);

real a =0.08;
real r1=50e-6/GloScaX;
real y1=-60e-6/GloScaY;
border topRight(t=r1, a*r1) { x=t; y=0; }
border topLeft(t=a*r1, 0) { x=t; y=0; }
border inner(t=0,y1) {x=0; y=t;}
border inside(t=0,y1) {x=a*r1; y=t;}
border bottomLeft(t=0,a*r1) {x=t; y=y1;}
border bottomRight(t=a*r1, r1) {x=t; y=y1;}
border outer(t=y1, 0) {x=r1; y=t;}
// The meshed domain is at the left hand side of the border
GloMesh = buildmesh(topRight(20) + topLeft(20) + outer(20) + bottomLeft(20) + bottomRight(20) + inner(100) + inside(100) );



/*************************************************************
 * Define doping profile  (uniform here)
 *************************************************************/
ParUsrNDOP = 0.0;
ParUsrPDOP = 1e18 * ConvC2M^(-3);

/********************
 * Define laser(s)
 ********************/
ParUsrLaserN = 1;                       /* number of lasers */
ParUsrLaserE[0] = 1.24 / 0.980;         /* energy [eV] */
ParUsrLaserI[0] = 0.806e6 * ConvC2M^(-2);  /* Irradiance [W m-2] */
ParUsrLaserRefl[0] =  0.303;            /* Reflectance [adim] */
ParUsrLaserR[0] = 1.5 * ConvU2M;        /* Radius [m] */

/*************************************
 * Solve the steady state linear system
 *************************************/
ModUsrRecLinearMass=1;      // switch on linear gen/rec (on the Mass matrix)
ParMatxyzITAU = 1e7;        // Inverse recombination time [s-1]
ModUsrOptCarrGen=1;         // switch on optical carrier generation
//ModUsrMob=ModCSTMobCst;
ModUsrDiff=ModCSTDiffCst;//FermifromMob;
ModUsrDiffAmbi = ModCSTDiffAmbiCst; // switch on constant diffusivity
ModUsrBTBIP=1;              // Band-to-band absorption model (Smith)

/*************************************
 * Solve the steady state linear system
 *************************************/
SolveCarrLinearSteady();

/******************
 * Plots
 ******************/
plot(GloVarSemiCARRGEN[0], wait=1, value=true); // the carrier generation by laser 1

plot(GloVecR, wait=1, value=true); /* Right hand side of the system equation.
                                    * Note that the maximum is not at r=0.
                                    * This is due to the use of cylindrical coordinates
                                    * (multiplication by x)
                                    */

plot(GloVarSemiEXC, wait=1, value=true); // the excess carrier (main solution)
plot(GloVarSemiELE, wait=1, value=true); // the total electron concentration
plot(GloVarSemiHOL, wait=1, value=true); // the total hole concentration

/*********************************
 * Test (for debugging purpose)
 *********************************/

x=0;
y=0;
real GloVarSemiEXC00 = GloVarSemiEXC;
real GloVarSemiEXC00saved = 3.788188529e+23;
real eps = GloVarSemiEXC00saved * 1e-5;


cout << "GloVarSemiEXC00saved: " << GloVarSemiEXC00saved << endl;
cout << "GloVarSemiEXC00: " << GloVarSemiEXC00 << endl;

cout << "DEBUG: min, max: " << GloVarSemiEXC[].min << ", " << GloVarSemiEXC[].max <

if(GloVarSemiEXC00 > GloVarSemiEXC00saved-eps && GloVarSemiEXC00 < GloVarSemiEXC00s
  cout << endl << "*** 001a: Test successfully passed ! ***" << endl;
} else {
  cout << endl << "*** ERROR: expected value is: " << GloVarSemiEXC00saved << endl;
  exit(0);
}
```

Freefem++ FSEMDIR/doc/examples/001b_CarrLinear_OhmicContact.edp

```
/*
 * Solve the linear steady state diffusion equation for the excess carriers.
 * Cylindrical coordinates.
 * One laser.
 */

include "../fsem_main_Include.edp";
fsemVERBOSITY = 10;
fsemModUseCpp = true;

/*****************************************************************
 * Define the mesh. The mesh must be adimentional (I will try to
 * make that automatice in the future)
 *****************************************************************/
real a =0.08;
real r1=50e-6/GloScaX;
real y1=-60e-6/GloScaY;
border topRight(t=r1, a*r1) { x=t; y=0; }
border topLeft(t=a*r1, 0) { x=t; y=0; }
border inner(t=0,y1) {x=0; y=t;}
border inside(t=0,y1) {x=a*r1; y=t;}
border bottomLeft(t=0,a*r1) {x=t; y=y1; label=100;}
border bottomRight(t=a*r1, r1) {x=t; y=y1; label=100;}
border outer(t=y1, 0) {x=r1; y=t;}
// The meshed domain is at the left hand side of the border
GloMesh = buildmesh(topRight(20) + topLeft(20) + outer(20) + bottomLeft(20) + bottomRight(20) + inner(100) + inside(100) );


GloCarrContactLabel[0] = 100; GloCarrContactEXC[0] = 0; /* set ohmic contact (equilirium) */


/*****************************************************************
 * Define doping profile  (uniform here)
 *****************************************************************/
ParUsrNDOP = 0.0;
ParUsrPDOP = 1e18 * ConvC2M^(-3);

/********************
 * Define laser(s)
 ********************/
ParUsrLaserN = 1;                          /* number of lasers */
ParUsrLaserE[0] = 1.24 / 0.980;            /* energy [eV] */
ParUsrLaserI[0] = 0.806e6 * ConvC2M^(-2);  /* Irradiance [W m-2] */
ParUsrLaserRefl[0] =  0.303;               /* Reflectance [adim] */
ParUsrLaserR[0] = 1.5 * ConvU2M;           /* Radius [m] */

/**************************************
 * Solve the steady state linear system
 **************************************/
ModUsrRecLinearMass=1;      // switch on linear gen/rec (on the Mass matrix)
ParMatxyzITAU = 1e3;        // Inverse recombination time [s-1]
ModUsrOptCarrGen=1;         // switch on optical carrier generation
ModUsrDiffAmbi = ModCSTDiffAmbiCst; // switch on constant diffusivity
ModUsrBTBIP=1;              // Band-to-band absorption model (Smith)

/**************************************
 * Solve the steady state linear system
 **************************************/
SolveCarrLinearSteady();

/******************
 * Plots
 ******************/

plot(GloVarSemiEXC, wait=1, value=true); // the excess carrier (main solution)

/********************
 * PostProcessing
 ********************/

// Generate a vertical cut line at r=0
real xmin = 0.0;
real ymin = 0.0;
real xmax = 0.0;
real ymax = -50e-6; // [m]
int npts = 200;
GloCutlineSave(xmin, ymin, xmax, ymax, npts, GloVarSemiEXC, "OutCutLine.dat", true,
GloSaveGmshMeshV1(GloMesh, "OutMesh.msh", ConvM2U, GloScaX, GloScaY);
GloSaveGmshPosSPARSEDST(GloMesh, GloVarSemiEXC, ConvM2C^(-3), "GloVarSemiEXC", "Out

/*********************************
 * Test (for debugging purpose)
 *********************************/

x=0;y=0;
real GloVarSemiEXC00 = GloVarSemiEXC;
real GloVarSemiEXC00saved = 6.228002665e+23;
real eps = GloVarSemiEXC00saved * 1e-5;

cout << "GloVarSemiEXC00: " << GloVarSemiEXC00 << endl;

if(GloVarSemiEXC00 > GloVarSemiEXC00saved-eps && GloVarSemiEXC00 < GloVarSemiEXC00s
  cout << endl << "*** 001b: Test successfully passed ! ***" << endl;
} else {
  cout << endl << "*** ERROR: expected value is: " << GloVarSemiEXC00saved << endl;
  exit(0);
}
```

```
    Freefem++ FSEMDIR/doc/examples/001c_CarrLinear_TimeDep.edp

/*
 * Solve the linear steady state diffusion equation for the
 * excess carriers.
 * Cylindrical coordinates.
 * One laser.
 */


include "../fsem_main_Include.edp";
fsemVERBOSITY = 10;
fsemModUseCpp = true;

/*************************************************************
 * Define the mesh. The mesh must be adimentional (I will try to
 * make that automatice in the future)
 *************************************************************/
real a =0.08;
real r1=50e-6/GloScaX;
real y1=-60e-6/GloScaY;
border topRight(t=r1, a*r1) { x=t; y=0; }
border topLeft(t=a*r1, 0) { x=t; y=0; }
border inner(t=0,y1) {x=0; y=t;}
border inside(t=0,y1) {x=a*r1; y=t;}
border bottomLeft(t=0,a*r1) {x=t; y=y1; label=100;}
border bottomRight(t=a*r1, r1) {x=t; y=y1; label=100;}
border outer(t=y1, 0) {x=r1; y=t;}
// The meshed domain is at the left hand side of the border
GloMesh = buildmesh(topRight(20) + topLeft(20) + outer(20) + bottomLeft(20) + bottomRight(20) + inner(100) + inside(100) );


/*************************************************************
 * Define doping profile  (uniform here)
 *************************************************************/
ParUsrNDOP = 0.0;
ParUsrPDOP = 1e18 * ConvC2M^(-3);

/*******************
 * Define laser(s)
 *******************/
ParUsrLaserN = 1;                          /* number of lasers */
ParUsrLaserE[0] = 1.24 / 0.980;            /* energy [eV] */
ParUsrLaserI[0] = 0.806e6 * ConvC2M^(-2);  /* Irradiance [W m-2] */
ParUsrLaserRefl[0] =  0.303;               /* Reflectance [adim] */
ParUsrLaserR[0] = 1.5 * ConvU2M;           /* Radius [m] */

/*************************************
 * Solve the steady state linear system
 *************************************/
ModUsrRecLinearMass=1;      /* switch on linear gen/rec
                             * (on the Mass matrix) */
ParMatxyzITAU = 1e7;        // Inverse recombination time [s-1]
ModUsrOptCarrGen=1;         // switch on optical carrier generation
ModUsrDiffAmbi = ModCSTDiffAmbiCst; // switch on constant diffusivity
ModUsrBTBIP=1;              // Band-to-band absorption model (Smith)

real MyFreq = 1e9;

func real SineWave(real t) {
  /*     Update everything that changes with time (t)
   *     (and that is not automatically accounted by a model).
   *     This function will be passed as argument
   *     to the SolveCarrLinearTimeDep() macro.
   *
   *     For instance:
   *       - external excitation (laser).
   *       - superficial charges? (charging effect due to illumination).
   *       - material parameters? (Fluorine outdiffusion leading to
   *                               enhanced carrier activation).
   *       - anything your mind can imagine.
   */
  cout << "Time is: " << t  << "[s]" << endl;
  real omega = 2* ParCstPi * MyFreq; // [Hz]

  // Pump laser [W m-2]
  ParUsrLaserI[0] = 0.379e6 * ConvC2M^(-2) * 0.5*(1 - cos(omega * t));
  return 0.0;
}


/* Set the global time vector */
int nStepsByPeriod = 20;
int nPeriod = 4;
int nSteps = nPeriod * nStepsByPeriod;
real[int] GloVarTimeVector(nSteps);
for(real i=0; i<nSteps; i=i+1) {
  GloVarTimeVector[i] = i/nStepsByPeriod * 1.0 / MyFreq;  // [s]
}

/* Array to store solution at each time in view of exporting to gmsh */
GloVhP1[int] EXCTime(nSteps);  /* This is quite memory expensive to do that
* I'll try to find a cheaper method
*/
func real MySaveFunction(real t, int i) {
  /*  Save or display the solution (or cross section, ...)
   *  at current time.
   *  This function will be passed as argument
   *  to the SolveCarrLinearTimeDep() macro.
   */

  // plot solution on screen
  plot(GloVarSemiEXC, wait=0, value=true);

  // save solution in array of fesapce variable
  EXCTime[i] = GloVarSemiEXC;
}


/*************************************
 * Solve the time dependent linear problem
 *************************************/
GloVarSemiEXC = 0;            // Set initial condition
EXCTime[0] = GloVarSemiEXC;   // Store initial solution
SolveCarrBDF1fixTime(SolveCarrLinearTimeStep, SineWave, MySaveFunction);


/*****************
 * Plots
 *****************/
plot(GloVarSemiCARRGEN[0], wait=1, value=true); // the carrier generation by laser

plot(GloVecR, wait=1, value=true); /* Right hand site of the system equation.
                                    * Note that the maximum is not at r=0.
                                    * This is due to the use of cylindrical coordin
                                    * (multiplication by x)
                                    */

plot(GloVarSemiEXC, wait=1, value=true); // the excess carrier (main solution)
plot(GloVarSemiELE, wait=1, value=true); // the total electron concentration
plot(GloVarSemiHOL, wait=1, value=true); // the total hole concentration


/*******************
 * Export to gmsh
 *******************/
GloSaveGmshPosSPARSEDSTMulti(GloMesh, GloVarTimeVector, EXCTime, ConvM2C^(-3), "Glo


/*********************************
 * Test (for debugging purpose)
 *********************************/

x=0;y=0;
real GloVarSemiEXC00 = GloVarSemiEXC;
real GloVarSemiEXC00saved = 4.381706199e+22;
real eps = GloVarSemiEXC00saved * 1e-5;

cout << "GloVarSemiEXC00: " << GloVarSemiEXC00 << endl;

if(GloVarSemiEXC00 > GloVarSemiEXC00saved-eps && GloVarSemiEXC00 < GloVarSemiEXC00s
  cout << endl << "*** 001c: Test successfully passed ! ***" << endl;
} else {
  cout << endl << "*** ERROR: expected value is: " << GloVarSemiEXC00saved << endl;
  exit(0);
}
```

```
    Freefem++ FSEMDIR/doc/examples/001d_CarrLinear_Harmonic.edp

include "../fsem_main_Include.edp";
fsemVERBOSITY = 10;
fsemModUseCpp = true;

/************************************************
 * Define the mesh. The mesh must be adimentional.
 * I will try to change that in the future
 ************************************************/
real a =0.08;
real r1=50e-6/GloScaX;
real y1=-60e-6/GloScaY;
border topRight(t=r1, a*r1) { x=t; y=0; }
border topLeft(t=a*r1, 0) { x=t; y=0; }
border inner(t=0,y1) {x=0; y=t;}
border inside(t=0,y1) {x=a*r1; y=t;}
border bottomLeft(t=0,a*r1) {x=t; y=y1; label=100;}
border bottomRight(t=a*r1, r1) {x=t; y=y1; label=100;}
border outer(t=y1, 0) {x=r1; y=t;}
// The meshed domain is at the left hand side of the border
GloMesh = buildmesh(topRight(20) + topLeft(20) + outer(20) + bottomLeft(20) + bottomRight(20) + inner(100) + inside(100) );



/*************************
 *  Define doping profile
 *************************/
ParUsrNDOP = 0.0;
ParUsrPDOP = 1e18 * ConvC2M^(-3);

/***************************************
 * Solve the steady state linear system
 ***************************************/
ModUsrRecLinearMass=1;
ParMatxyzITAU = 1e7;        // Inverse recombination time [s-1]
ModUsrOptCarrGen=1; // switch on optical carrier generation
ModUsrDiffAmbi = ModCSTDiffAmbiCst;    // switch on constant diffusivity
ModUsrBTBIP=1;      // Band-to-band absorption model (Smith)


/*******************
 * Define laser(s)
 *******************/
ParUsrLaserN = 1;                       /* number of lasers */
ParUsrLaserE[0] = 1.24 / 0.980;         /* energy [eV] */
ParUsrLaserI[0] = 0.806e6 * ConvC2M^(-2);  /* Irradiance [W m-2] */
ParUsrLaserRefl[0] =  0.303;            /* Reflectance [adim] */
ParUsrLaserR[0] = 1.5 * ConvU2M;        /* Radius [m] */
ParUsrLaserModFreq = 1e7; // Hz


/***************************************
 * Solve the harmonic excitation problem
 * (use complex formulation internally)
 ***************************************/
GloVarSemiEXCc = 1e10 * ConvC2M^(-3);  // used to compute equation's coefficients
SolveCarrLinearSteadyPeriodic();


/**********
 * Plots
 **********/
plot(GloVarSemiCARRGEN[0], wait=1, value=true); // the carrier generation by laser 1

GloVhP1 GloVarSemiEXCcReal = real(GloVarSemiEXCc);
GloVhP1 GloVarSemiEXCcImag = imag(GloVarSemiEXCc);

plot(GloVarSemiEXCcReal, wait=1, value=true);
plot(GloVarSemiEXCcImag, wait=1, value=true);

GloVhP1 GloVarSemiEXCcMod = abs(GloVarSemiEXCc);
plot(GloVarSemiEXCcMod, wait=1, value=true); // the solution (excess carrier in m-3)

/********************
 * Export to gmsh
 ********************/
GloSaveGmshPosSPARSEDSTComplex(GloMesh, GloVarSemiEXCc, ConvM2C^(-3), "GloVarSemiEXCc", "OutGloVarSemiEXCc.pos", ConvM2U, GloScaX, GloScaY);
/*******************************
 * Test (for debugging purpose)
 *******************************/

x=0;y=0;
real GloVarSemiEXC00r = real(GloVarSemiEXCc);
real GloVarSemiEXC00i = imag(GloVarSemiEXCc);
real GloVarSemiEXC00savedr = 2.504108621e+23;
real GloVarSemiEXC00savedi = -9.673234312e+22;
real eps = abs(GloVarSemiEXC00savedi) * 1e-5;

cout << "GloVarSemiEXC00r: " << GloVarSemiEXC00r << endl;
cout << "GloVarSemiEXC00i: " << GloVarSemiEXC00i << endl;

if(GloVarSemiEXC00r > GloVarSemiEXC00savedr-eps && GloVarSemiEXC00r < GloVarSemiEXC
   && GloVarSemiEXC00i > GloVarSemiEXC00savedi-eps && GloVarSemiEXC00i < GloVarSemi
  cout << endl << "*** 001d: Test successfully passed ! ***" << endl;
} else {
  cout << endl << "*** ERROR: expected value is: " <<  GloVarSemiEXC00savedr << ",
  exit(0);
}
```

3.2 Nonlinear carrier diffusion equation

Freefem++ FSEMDIR/doc/examples/003a_CarrNonlinear.edp

```
include "../fsem_main_Include.edp";
fsemVERBOSITY = 10;
fsemModUseCpp = true;

/************************************************
 * Define the mesh. The mesh must be adimentional.
 * I will try to change that in the future.
 ************************************************/
real a =0.08;
real r1=50e-6/GloScaX;
real y1=-60e-6/GloScaY;
border topRight(t=r1, a*r1) { x=t; y=0; }
border topLeft(t=a*r1, 0) { x=t; y=0; }
border inner(t=0,y1) {x=0; y=t;}
border inside(t=0,y1) {x=a*r1; y=t;}
border bottomLeft(t=0,a*r1) {x=t; y=y1; label=100;}
border bottomRight(t=a*r1, r1) {x=t; y=y1; label=100;}
border outer(t=y1, 0) {x=r1; y=t;}
// The meshed domain is at the left hand side of the border
GloMesh = buildmesh(topRight(20) + topLeft(20) + outer(20) + bottomLeft(20) + bottomRight(20) + inner(100) + inside(100) );


/*************************
 * Define doping profile
 *************************/
ParUsrNDOP = 0.0;
ParUsrPDOP = 1e18 * ConvC2M^(-3);

/*******************
 * Define laser(s)
 *******************/
ParUsrLaserN = 1;                          /* number of lasers */
ParUsrLaserE[0] = 1.24 / 0.980;            /* energy [eV] */
ParUsrLaserI[0] = 0.806e6 * ConvC2M^(-2);  /* Irradiance [W m-2] */
ParUsrLaserRefl[0] =  0.303;               /* Reflectance [adim] */
ParUsrLaserR[0] = 1.5 * ConvU2M;           /* Radius [m] */
ParUsrLaserModFreq = 1e5; // Hz

/***********************************
 * Solve the linear steady problem
 ***********************************/
ModUsrOptCarrGen=1;            // switch on optical carrier generation
ModUsrDiffAmbi = ModCSTDiffAmbiCst;    // switch on constant diffusivity
ModUsrBTBIP=1;                 // Band-to-band absorption model (Smith)
ModUsrRecLinearMass=1;         // Linear recombination
ParMatxyzITAU = 1e7;        // Inverse recombination time [s-1]

SolveCarrLinearSteady();
plot(GloVarSemiEXC, wait=1, value=true);

/***************************************
 *  Solve the nonlinear steady problem
 ***************************************/
ModUsrRecAuger=1;
ModUsrRecSRH=1;
ModUsrRecSRHConcDepLifeTime=1;
ModUsrRecLinearMass=0;

ParMatxyzTAUN0 = 2e-3;
ParMatxyzTAUP0 = 2e-4;
ParMatxyzNSRH = 5e16 * ConvC2M^(-3);
ParMatxyzPSRH = 5e16 * ConvC2M^(-3);

SolveCarrNonlinearSteady();


/************
 * Plots
 ************/
plot(GloVarSemiCARRGEN[0], wait=1, value=true); // the carrier generation by laser 1

plot(GloVecR, wait=1, value=true); /* Right hand site of the system equation.
                                    * Note that the maximum is not at r=0.
                                    * This is due to the use of cylindrical coordinates
                                    * (multiplication by x)
                                    */

plot(GloVecX, wait=1, value=true); // the solution (excess carrier [adim])
plot(GloVarSemiEXC, wait=1, value=true); // the solution (excess carrier [m-3])
```

```
/*******************************
 * Test (for debugging purpose)
 *******************************/

x=0;y=0;
real GloVarSemiEXC00 = GloVarSemiEXC;
real GloVarSemiEXC00saved = 8.693800235e+23;
real eps = GloVarSemiEXC00saved * 1e-5;

cout << "GloVarSemiEXC00: " << GloVarSemiEXC00 << endl;

if(GloVarSemiEXC00 > GloVarSemiEXC00saved-eps && GloVarSemiEXC00 < GloVarSemiEXC00s
  cout << endl << "*** 003a: Test successfully passed ! ***" << endl;
} else {
  cout << endl << "*** ERROR 003a: expected value is: " <<  GloVarSemiEXC00saved <<
  exit(0);
}
```

```
    Freefem++ FSEMDIR/doc/examples/003b_CarrNonlinear_Diffusionless.edp

/*
 * Solve the diffusion less equation
 * This is like a zero dimentional problem (Generations = Recombinations)
 */

include "../fsem_main_Include.edp";
fsemVERBOSITY = 10;
fsemModUseCpp = true;

/************************************************
 * Define the mesh. The mesh must be adimentional
 * (I will try to change that in the future).
 * This is the coarsest mesh that is
 * possible to make (2 elements, 4 nodes).
 ************************************************/
real r0=0, r1=1e-9/GloScaX;   /* very small dimentions (1 nm) relative to the
                               * the generation (see later)
                               * in order to have uniform generation
                               */
real y0=0, y1=-1e-9/GloScaY;
int n=1, m=1;
GloMesh =square(n,m,[r0+(r1-r0)*x,y0+(y1-y0)*y]);



/*************************************
 * Define doping profile (uniform here)
 *************************************/
ParUsrNDOP = 0.0;                   /* Donnors (n-type) */
ParUsrPDOP = 1e18 * ConvC2M^(-3);   /* Acceptors (p-type)*/

/********************
 * Define laser(s)
 ********************/
ParUsrLaserN = 1;                          /* number of lasers */
ParUsrLaserE[0] = 1.24 / 0.980;            /* energy [eV] */
ParUsrLaserI[0] = 0.806e6 * ConvC2M^(-2);  /* Irradiance [W m-2] */
ParUsrLaserRefl[0] =  0.303;               /* Reflectance [adim] */
ParUsrLaserR[0] = 1.5 * ConvU2M;           /* Radius [m] */

/*********************************
 * Solve the LINEAR steady problem
 *********************************/
ModUsrOptCarrGen=1;          /* switch on optical carrier generation */
ModUsrDiffAmbi = ModCSTDiffAmbiCst;  /* switch on constant diffusivity */
ModUsrBTBIP=1;               /* Band-to-band absorption model (Smith) */
ModUsrRecLinearMass=1;       /* Linear recombination in the Mass matrix */
ParMatxyzITAU = 1e7;         // Inverse recombination time [s-1]
ParMatxyzD = 0;      /* Set diffusion coefficient to zero (diffusionless) */

SolveCarrLinearSteady();  /* Solve the linear steady state equation for the excess carriers */

plot(GloVarSemiEXC, wait=1, value=true);  /* plot the excess carrier concentration [m^(-3)] */

/************************************
 * Solve the LINEAR steady problem as if
 * it was a NON-LINEAR problem
 ************************************/
ModUsrRecLinear=1;        /* Linear recombination in the Right Hand Side */
ParMatxyzCLIN=1e7;        // Inverse recombination time [s-1]
ModUsrRecLinearMass=0;

SolveCarrNonlinearSteady();
                      /* Solve the nonlinear steady state equation for the excess carrier
                   * The solution should be identical to the previous one
                          * and Newton iteration should converge in 1 iteration
                          */
plot(GloVarSemiEXC, wait=1, value=true);  /* plot the excess carrier concentration [m^(-3)] */

/************************************
 * Solve the NON-LINEAR steady problem
 ************************************/
ModUsrRecAuger = 1;
ModUsrRecSRH = 1;
ModUsrRecLinear= 0;

ParMatxyzTAUN0 = 2e-3;
ParMatxyzTAUP0 = 2e-4;
ParMatxyzNSRH = 5e16 * ConvC2M^(-3);
ParMatxyzPSRH = 5e16 * ConvC2M^(-3);

SolveCarrNonlinearSteady();
plot(GloVarSemiEXC, wait=1, value=true);  /* plot the excess carrier concentration

// Plot some more information
plot(GloVarSemiCARRGEN[0], wait=1, value=true); // the carrier generation by laser
plot(GloVecR, wait=1, value=true); /* Right hand site of the system equation.
                                    * Note that the maximum is not at r=0.
                                    * This is due to the use of cylindrical coordin
                                    * (multiplication by x)
                                    */



/*********************************
 * Test (for debugging purpose)
 *********************************/
real GloVarSemiEXC00 = GloVarSemiEXC[][0];
real GloVarSemiEXC00saved = 7.649390799e+24;
real eps = GloVarSemiEXC00saved * 1e-5;

cout << "GloVarSemiEXC00: " << GloVarSemiEXC00 << endl;

if(GloVarSemiEXC00 > GloVarSemiEXC00saved-eps && GloVarSemiEXC00 < GloVarSemiEXC00s
  cout << endl << "*** 003b: Test successfully passed ! ***" << endl;
} else {
  cout << endl << "*** ERROR: expected value is: " <<  GloVarSemiEXC00saved << endl
  exit(0);
}
```

```
    Freefem++ FSEMDIR/doc/examples/003c_CarrNonlinear_TimeDep.edp
/*
 * Solve the linear steady state diffusion equation for the excess carriers.
 * Cylindrical coordinates.
 * One laser.
 */


include "../fsem_main_Include.edp";
fsemVERBOSITY = 3;
fsemModUseCpp = true;

/*****************************************************************
 * Define the mesh. The mesh must be adimentional (I will try to
 * make that automatice in the future)
 *****************************************************************/
real a =0.08;
real r1=50e-6/GloScaX;
real y1=-60e-6/GloScaY;
border topRight(t=r1, a*r1) { x=t; y=0; }
border topLeft(t=a*r1, 0) { x=t; y=0; }
border inner(t=0,y1) {x=0; y=t;}
border inside(t=0,y1) {x=a*r1; y=t;}
border bottomLeft(t=0,a*r1) {x=t; y=y1; label=100;}
border bottomRight(t=a*r1, r1) {x=t; y=y1; label=100;}
border outer(t=y1, 0) {x=r1; y=t;}
// The meshed domain is at the left hand side of the border
GloMesh = buildmesh(topRight(20) + topLeft(20) + outer(20) + bottomLeft(20) + bottomRight(20) + inner(100) + inside(100) );



/*****************************************************************
 * Define doping profile  (uniform here)
 *****************************************************************/
ParUsrNDOP = 0.0;
ParUsrPDOP = 1e18 * ConvC2M^(-3);

/********************
 * Define laser(s)
 ********************/
ParUsrLaserN = 1;                          /* number of lasers */
ParUsrLaserE[0] = 1.24 / 0.980;            /* energy [eV] */
ParUsrLaserI[0] = 0.806e6 * ConvC2M^(-2);  /* Irradiance [W m-2] */
ParUsrLaserRefl[0] =  0.303;               /* Reflectance [adim] */
ParUsrLaserR[0] = 1.5 * ConvU2M;           /* Radius [m] */

/*************************************
 * Solve the steady state linear system
 *************************************/
ModUsrRecAuger=1;
ModUsrRecSRH=1;
ModUsrRecSRHConcDepLifeTime=1;
ModUsrOptCarrGen=1;         // switch on optical carrier generation
ModUsrDiffAmbi = ModCSTDiffAmbiCst; // switch on constant diffusivity
ModUsrBTBIP=1;              // Band-to-band absorption model (Smith)

ParMatxyzTAUN0 = 2e-3;
ParMatxyzTAUP0 = 2e-4;
ParMatxyzNSRH = 5e16 * ConvC2M^(-3);
ParMatxyzPSRH = 5e16 * ConvC2M^(-3);

real MyFreq = 1e7;

/* Set the global time vector */
int nStepsByPeriod = 20;
int nPeriod = 3;
int nSteps = nPeriod * nStepsByPeriod;
real[int] GloVarTimeVector(nSteps);
for(real i=0; i<nSteps; i=i+1) {
  GloVarTimeVector[i] = i/nStepsByPeriod * 1.0 / MyFreq;  // [s]
}

/* Array to store solution at each time in view of exporting to gmsh */
GloVhP1[int] EXCTime(nSteps);  /* This is quite memory expensive to do that
* I'll try to find a cheaper method
*/

func real MyTimeDepParFunction(real t) {
  /*     Update everything that changes with time (t)
   *     (and that is not automatically accounted by a model).
   *     This function will be passed as argument
   *     to the SolveCarrLinearTimeDep() macro.
   *
   *     For instance:
   *       - external excitation (laser).
   *       - superficial charges? (charging effect due to illumination).
   *       - material parameters? (Fluorine outdiffusion leading to enhanced carrie
   *       - anything your mind can imagine.
   */
  cout << "Time is: " << t  << "[s]" << endl;
  real omega = 2* ParCstPi * MyFreq; // [Hz]
  ParUsrLaserI[0] = 0.379e6 * ConvC2M^(-2) * 0.5*(1 - cos(omega * t)); // [W m-2] P

  return 0.0;
}

func real MySaveFunction(real t, int i) {
  /*  Save or display the solution (or cross section, ...) at current time.
   *     This function will be passed as argument
   *     to the SolveCarrLinearTimeDep() macro.
   */
  plot(GloVarSemiEXC, wait=0, value=true);

  // save solution in array of fesapce variable
  EXCTime[i] = GloVarSemiEXC;
}


/*****************************************
 * Solve the time dependent nonlinear problem
 *****************************************/
GloVarSemiEXC = 0;            // Set initial condition
SolveCarrBDF1fixTime(SolveCarrNonlinearTimeStep, MyTimeDepParFunction, MySaveFuncti


/******************
 * Plots
 ******************/
plot(GloVarSemiCARRGEN[0], wait=1, value=true); // the carrier generation by laser

plot(GloVecR, wait=1, value=true); /* Right hand site of the system equation.
                                    * Note that the maximum is not at r=0.
                                    * This is due to the use of cylindrical coordin
                                    * (multiplication by x)
                                    */

plot(GloVarSemiEXC, wait=1, value=true); // the excess carrier (main solution)
plot(GloVarSemiELE, wait=1, value=true); // the total electron concentration
plot(GloVarSemiHOL, wait=1, value=true); // the total hole concentration


/********************
 * Export to gmsh
 ********************/
GloSaveGmshPosSPARSEDSTMulti(GloMesh, GloVarTimeVector, EXCTime, ConvM2C^(-3), "Glo



/*********************************
 * Test (for debugging purpose)
 *********************************/

x=0;y=0;
real GloVarSemiEXC00 = GloVarSemiEXC;
real GloVarSemiEXC00saved = 6.528592424e+22;
real eps = GloVarSemiEXC00saved * 1e-5;

cout << "GloVarSemiEXC00: " << GloVarSemiEXC00 << endl;

if(GloVarSemiEXC00 > GloVarSemiEXC00saved-eps && GloVarSemiEXC00 < GloVarSemiEXC00s
  cout << endl << "*** 003c: Test successfully passed ! ***" << endl;
} else {
  cout << endl << "*** ERROR: expected value is: " <<  GloVarSemiEXC00saved << endl
  exit(0);
}
```

Freefem++ FSEMDIR/doc/examples/003d_CarrNonlinear_Harmonic.edp

```
/*
 * Solve:
 *   - the nonlinear steady state (probe) diffusion equation for the excess carriers.
 *   - the nonlinear steady periodic (pump) diffusion equation for the excess carriers.
 *
 * Cylindrical coordinates.
 * Two lasers (steady probe and harmonic pump).
 */


include "../fsem_main_Include.edp";
fsemVERBOSITY = 10;
fsemModUseCpp = true;

UsrNewtonNbrMaxIterCarr = 40;

/*************************************************************
 * Define the mesh. The mesh must be adimentional (I will try to
 * make that automatice in the future)
 *************************************************************/
real a =0.08;
real r1=50e-6/GloScaX;
real y1=-60e-6/GloScaY;
border topRight(t=r1, a*r1) { x=t; y=0; }
border topLeft(t=a*r1, 0) { x=t; y=0; }
border inner(t=0,y1) {x=0; y=t;}
border inside(t=0,y1) {x=a*r1; y=t;}
border bottomLeft(t=0,a*r1) {x=t; y=y1; label=100;}
border bottomRight(t=a*r1, r1) {x=t; y=y1; label=100;}
border outer(t=y1, 0) {x=r1; y=t;}
// The meshed domain is at the left hand side of the border
GloMesh = buildmesh(topRight(20) + topLeft(20) + outer(20) + bottomLeft(20) + bottomRight(20) + inner(100) + inside(100) );

/*************************************************************
 * Define doping profile  (uniform here)
 *************************************************************/
ParUsrNDOP = 0.0;
ParUsrPDOP = 1e18 * ConvC2M^(-3);

/*************************************************************
 * Solve the non linear steady state diff equation (probe only)
 *************************************************************/

ParMatxyzTAUN0 = 2e-3;
ParMatxyzTAUP0 = 2e-4;
ParMatxyzNSRH = 5e16 * ConvC2M^(-3);
ParMatxyzPSRH = 5e16 * ConvC2M^(-3);

bool ProbeIsOn = true; // turn the probe laser ON or OFF

if(ProbeIsOn) {
  // Define PROBE laser
  ParUsrLaserN = 1;                          /* number of lasers */
  ParUsrLaserE[0] = 1.24 / 0.980;            /* energy [eV] */
  ParUsrLaserI[0] = 0.806e6 * ConvC2M^(-2);  /* Irradiance [W m-2] */
  ParUsrLaserRefl[0] =  0.303;               /* Reflectance [adim] */
  ParUsrLaserR[0] = 1.5 * ConvU2M;           /* Radius [m] */

  // Set models
  ModUsrRecLinearMass=0;
  ModUsrOptCarrGen=1;   // Switch on optical carrier generation
  ModUsrBTBIP=1;        // Band-to-band absorption model (Smith)

  ModUsrRecAuger=1;
  ModUsrRecSRH=1;
  ModUsrRecSRHConcDepLifeTime=1;
  ModUsrMob = ModCSTMobPHUNI;
  ModUsrDiff = ModCSTDiffBoltzfromMob;
  ModUsrDiffAmbi = ModCSTDiffAmbifromDiff;

  // Solve nonlinear steady state
  SolveCarrNonlinearSteady();
  plot(GloVarSemiEXC, wait=1, value=true);

  GloVarSemiEXCbias = GloVarSemiEXC;   // use probe-only steady state solution as bias
} else {
  GloVarSemiEXCbias = 0.0;  // no bias (probe laser is OFF)
}

x=0; y=0;
real ProbeOnlySol = GloVarSemiEXC;
/*****************************************************
 * Solve the pseudo nonlinear steady periodic equation
 * (pump only but considering the probe-only
 *  solution as a bias -> use GloVarSemiEXCbias)
 *****************************************************/
// Define PUMP laser
ParUsrLaserN = 1;                          /* number of lasers */
ParUsrLaserE[0] = 1.24 / 0.830;            /* energy [eV] */
ParUsrLaserI[0] = 0.379e6 * ConvC2M^(-2);  /* Irradiance [W m-2] */
ParUsrLaserRefl[0] =  0.303;               /* Reflectance [adim] */
ParUsrLaserR[0] = 2.2 * ConvU2M;           /* Radius [m] */
ParUsrLaserModFreq = 1e6; // Hz
GloXFlagHighFreqJac = false;

// Set models
ModUsrRecLinearMass=1;
ParMatxyzITAU=0;      // We do not want any linear recombination term
ModUsrOptCarrGen=1;   // Switch on optical carrier generation
ModUsrBTBIP=1;        // Band-to-band absorption model (Smith)

ModUsrRecAuger=1;
ModUsrRecSRH=1;
ModUsrRecSRHConcDepLifeTime=1;
ModUsrMob = ModCSTMobPHUNI;
//ModUsrMob = ModCSTMobCst;
ModUsrDiff = ModCSTDiffBoltzfromMob;

  // Solve nonlinear steady state

// Solve

GloVarSemiEXCc = 1e15 * ConvC2M^(-3); // Initial guess
SolveCarrNonlinearSteadyPeriodic();

x=0; y=0;
complex HarmonicSol = GloVarSemiEXCc;


/*
 * Probe + Pump steady periodic.
 * This should give the same answer the harmonic case with
 * probe bias and zero frequency pump
 */

// Define PUMP laser
ParUsrLaserN = 2;                          /* number of lasers */
ParUsrLaserE[0] = 1.24 / 0.980;            /* energy [eV] */
ParUsrLaserE[1] = 1.24 / 0.830;
ParUsrLaserI[0] = 0.806e6 * ConvC2M^(-2);  /* Irradiance [W m-2] */
ParUsrLaserI[1] = 0.379e6 * ConvC2M^(-2);
ParUsrLaserRefl[0] =  0.303;               /* Reflectance [adim] */
ParUsrLaserRefl[1] =  0.303;
ParUsrLaserR[0] = 1.5 * ConvU2M;           /* Radius [m] */
ParUsrLaserR[1] = 2.2 * ConvU2M;

// Set models
ModUsrRecLinearMass=0;
ModUsrOptCarrGen=1;   // Switch on optical carrier generation
ModUsrBTBIP=1;        // Band-to-band absorption model (Smith)

ModUsrRecAuger=1;
ModUsrRecSRH=1;
ModUsrRecSRHConcDepLifeTime=1;
ModUsrMob = ModCSTMobPHUNI;
//ModUsrDiff = ModCSTDiffAmbifromMob;
ModUsrDiff = ModCSTDiffBoltzfromMob;
ModUsrDiffAmbi = ModCSTDiffAmbifromDiff;

// Solve
GloVarSemiEXC = 1e15 * ConvC2M^(-3); // Initial guess
SolveCarrNonlinearSteady();
x=0; y=0;
real ProbePumpSol = GloVarSemiEXC;
```

```
/******************
 * Plots
 ******************/

// Variables used to draw real and imaginary part of the solution
GloVhP1 RealEXC;
GloVhP1 ImagEXC;
RealEXC = real(GloVarSemiEXCc);
ImagEXC = imag(GloVarSemiEXCc);
plot(RealEXC, wait=1, value=true); // the excess carrier (main solution)
plot(ImagEXC, wait=1, value=true); // the excess carrier (main solution)

cout << endl;
cout << "GloVarSemiEXCbias: " << GloVarSemiEXCbias << endl;
cout << "ParUsrLaserModFreq: " << ParUsrLaserModFreq << endl;
cout << endl;
cout << "ProbeOnlySol: " << ProbeOnlySol << endl;
cout << "ProbePumpSol: " << ProbePumpSol << endl;
cout << "HarmonicSol : " << HarmonicSol << endl;

cout << "ProbePumpSol - ProbeOnlySol: " << ProbePumpSol - ProbeOnlySol << endl;
cout << endl;
cout << endl;


/*********************************
 * Test (for debugging purpose)
 *********************************/

x=0;y=0;
real GloVarSemiEXC00r = real(GloVarSemiEXCc);
real GloVarSemiEXC00i = imag(GloVarSemiEXCc);
real GloVarSemiEXC00savedr = 2.825302548e+24;   // ParUsrLaserModFreq was 1e6
real GloVarSemiEXC00savedi = -6.983955682e+23;
real eps = abs(GloVarSemiEXC00savedi) * 1e-5;

cout << "GloVarSemiEXC00r: " << GloVarSemiEXC00r << endl;
cout << "GloVarSemiEXC00i: " << GloVarSemiEXC00i << endl;


cout << "max de RealExc " << RealEXC[].max << endl;
cout << "min de ImagExc: " << ImagEXC[].min << endl;


if(GloVarSemiEXC00r > GloVarSemiEXC00savedr-eps && GloVarSemiEXC00r < GloVarSemiEXC00savedr+eps
   && GloVarSemiEXC00i > GloVarSemiEXC00savedi-eps && GloVarSemiEXC00i < GloVarSemiEXC00savedi+eps) {
  cout << endl << "*** 003d: Test successfully passed ! ***" << endl;
} else {
  cout << endl << "*** ERROR: expected value is: " <<  GloVarSemiEXC00savedr << ", " << GloVarSemiEXC00savedi << endl;
  exit(0);
}
```

3.3 Linear heat equation

```
    Freefem++ FSEMDIR/doc/examples/005a_TempLinear.edp

/*
 * Solve the linear steady state diffusion equation for the excess carriers.
 * Cylindrical coordinates.
 * One laser.
 */


include "../fsem_main_Include.edp";
fsemVERBOSITY = 10;
fsemModUseCpp = true;

/*****************************************************************
 * Define the mesh. The mesh must be adimentional (I will try to
 * make that automatice in the future)
 *****************************************************************/
real a =0.08;
real r1=50e-6/GloScaX;
real y1=-60e-6/GloScaY;
border topRight(t=r1, a*r1) { x=t; y=0; label=100;}
border topLeft(t=a*r1, 0) { x=t; y=0; }
border inner(t=0,y1) {x=0; y=t;}
border inside(t=0,y1) {x=a*r1; y=t;}
border bottomLeft(t=0,a*r1) {x=t; y=y1; label=101;}
border bottomRight(t=a*r1, r1) {x=t; y=y1; label=101;}
border outer(t=y1, 0) {x=r1; y=t;}
// The meshed domain is at the left hand side of the border
GloMesh = buildmesh(topRight(20) + topLeft(20) + outer(20) + bottomLeft(20) + bottomRight(20) + inner(100) + inside(100) );


/* At least one contact needed for steady state calculation
 * otherwize solution diverges */
//GloTempContactLabel[0] = 100; GloTempContactT[0] = 310; /* set temperature of thermal contact */
GloTempContactLabel[1] = 101; GloTempContactT[1] = 300; /* set temperature of thermal contact */

/*****************************
 * Set models
 *****************************/
ModUsrHeatConductivityCst = 1;
ModUsrHeatRhoCpCst = 1;

/* Specify heat source by a user defined function */
ModUsrHeatUserSpec = 1;
GloVarHeatGen = 1e13 * exp(-(x*GloScaX/20e-6)^2) * exp(y*GloScaY/20e-7);  // W/m3

/****************************************
 * Solve the steady state linear system
 ****************************************/

SolveTempLinearSteady();

/******************
 * Plots
 ******************/
plot(GloVarHeatGen, wait=1, value=true);
plot(GloVarTempT, wait=1, value=true); // the temperature


/*********************************
 * Test (for debugging purpose)
 *********************************/

x=0;y=0;
real GloVarTempT00 = GloVarTempT;
real GloVarTempT00saved = 302.3483161;
real eps = abs(GloVarTempT00saved) * 1e-5;

cout << "GloVarTempT00: " << GloVarTempT00 << endl;

if(GloVarTempT00 > GloVarTempT00saved-eps && GloVarTempT00 < GloVarTempT00saved+eps) {
  cout << endl << "*** 005a: Test successfully passed ! ***" << endl;
} else {
  cout << endl << "*** ERROR: expected value is: " <<  GloVarTempT00saved << endl;
  exit(0);
}
```

```
    Freefem++ FSEMDIR/doc/examples/005b_TempLinear_Harmonic.edp
/*
 * Solve the linear steady state diffusion equation for the excess carriers
 * Cylindrical coordinates.
 * One laser.
 */


include "../fsem_main_Include.edp";
fsemVERBOSITY = 10;
fsemModUseCpp = true;

/*****************************************************************
 * Define the mesh. The mesh must be adimentional (I will try to
 * make that automatice in the future)
 *****************************************************************/
real a =0.08;
real r1=50e-6/GloScaX;
real y1=-60e-6/GloScaY;
border topRight(t=r1, a*r1) { x=t; y=0; label=100;}
border topLeft(t=a*r1, 0) { x=t; y=0; }
border inner(t=0,y1) {x=0; y=t;}
border inside(t=0,y1) {x=a*r1; y=t;}
border bottomLeft(t=0,a*r1) {x=t; y=y1; label=101;}
border bottomRight(t=a*r1, r1) {x=t; y=y1; label=101;}
border outer(t=y1, 0) {x=r1; y=t;}
// The meshed domain is at the left hand side of the border
GloMesh = buildmesh(topRight(20) + topLeft(20) + outer(20) + bottomLeft(20) + bottomRight(20) + inner(100) + inside(100) );


/* Contact is not mandatory for harmonic calculation. */
//GloTempContactLabel[0] = 100; GloTempContactTc[0] = 310; /* set temperature of thermal contact */
GloTempContactLabel[1] = 101; GloTempContactTc[1] = 0; /* set temperature of thermal contact */

/*****************************
 * Set models
 *****************************/
ModUsrHeatConductivityCst = 1;
ModUsrHeatRhoCpCst = 1;

/* Specify heat source by a user defined function */
ModUsrHeatUserSpec = 1;
GloVarHeatGen= 1e13 * exp(-(x*GloScaX/20e-6)^2) * exp(y*GloScaY/20e-6);  // W/m3
// Note: alternatively, one can set GloVarHeatGenc instead of GloVarHeatGen,
//       but be aware that they will be summed.
ParUsrLaserModFreq = 1e5; // Hz

/***************************************
 * Solve the steady state linear system
 ***************************************/

SolveTempLinearSteadyPeriodic();

/******************
 * Plots
 ******************/
plot(GloVarHeatGen, wait=1, value=true);

GloVhP1 TReal = real(GloVarTempTc);
GloVhP1 TImag = imag(GloVarTempTc);
GloVhP1 TAbs = abs(GloVarTempTc);

plot(TReal, wait=1, value=true); // the temperature
plot(TImag, wait=1, value=true); // the temperature
plot(TAbs, wait=1, value=true);

GloVhP1 TPhase = atan(TImag / TReal) / pi * 180;
plot(TPhase, wait=1, value=true);


/*******************
 * Export to gmsh
 *******************/
GloSaveGmshPosSPARSEDSTComplex(GloMesh, GloVarTempTc, 1, "GloVarSemiTempTc", "OutGloVarSemiTempTc.pos", ConvM2U, GloScaX, GloScaY);

/*********************************
 * Test (for debugging purpose)
 *********************************/

x=0;y=0;
real GloVarTempT00r = real(GloVarTempTc);
real GloVarTempT00i = imag(GloVarTempTc);
real GloVarTempT00savedr = 2.976558049;
real GloVarTempT00savedi = -2.955961579;
real eps = abs(GloVarTempT00savedi) * 1e-5;

cout << "GloVarTempT00r: " << GloVarTempT00r << endl;
cout << "GloVarTempT00i: " << GloVarTempT00i << endl;


if(GloVarTempT00r > GloVarTempT00savedr-eps && GloVarTempT00r < GloVarTempT00savedr
   && GloVarTempT00i > GloVarTempT00savedi-eps && GloVarTempT00i < GloVarTempT00sav
  cout << endl << "*** 005b: Test successfully passed ! ***" << endl;
} else {
  cout << endl << "*** ERROR: expected value is: " <<  GloVarTempT00savedr << ", "
  exit(0);
}
```

Chapter 4

Syntax

4.1 External excitations

Table 4.1: User variables: Lasers

Type	Name	Default	Description	Units
real	ParUsrLaserModFreq	0	Laser's modulation frequency when solving linear steady periodic equations. All lasers are assumed to have the same excitation frequency. Othermize, use superposition principle.	[s-1]
int	ParUsrLaserNMAX	2	DEV	
int	ParUsrLaserN	0	Number of laser	[adim]
real[int]	ParUsrLaserE(ParUsrLaserNMAX)		Photon energy of each laser	[eV]
real[int]	ParUsrLaserI(ParUsrLaserNMAX)		Peak intensity of each Gaussian shaped laser	[W m-2]
real[int]	ParUsrLaserRefl(ParUsrLaserNMAX)		Reflectivity coefficient at each wavelength	[adim]
real[int]	ParUsrLaserR(ParUsrLaserNMAX)		Gauss Radius of the Gaussian shaped laser	[m]
real[int]	ParUsrLaserA(ParUsrLaserNMAX)		The absorption coefficient for each laser (optional)	[m-1]

blabla

4.2 Material parameters

Table 4.2: User variables: Semiconductor material coefficients

Type	Name	Default	Description	Units
GloVhP1	ParMatxyzD	ParMatSiD	Ambipolar diffusivity (see ModCSTDiffAmbiCst)	[m^2 s^{-1}]
GloVhP1	ParMatxyzDn	ParMatSiDn	Electron diffusivity (see ModCSTDiffCst)	[m^2 s^{-1}]
GloVhP1	ParMatxyzDp	ParMatSiDp	Hole diffusivity (see ModCSTDiffCst)	[m^2 s^{-1}]
GloVhP1	ParMatxyzMobn	ParMatSiMobn	Electron mobility (see ModCSTMobCst)	[m^2 V^{-1} s^{-1}]
GloVhP1	ParMatxyzMobp	ParMatSiMobp	Hole mobility (see ModCSTMobCst)	[m^2 V^{-1} s^{-1}]
GloVhP1	ParMatxyzCLIN	ParMatSiITAU	Linear recombination coefficient (inverse lifetime)	[s^{-1}]
GloVhP1	ParMatxyzCDIR	ParMatSiCDIR	Direct recombination coefficient (see ModUsrRecDirect)	[m^3 s^{-1}]
GloVhP1	ParMatxyzAUGN	ParMatSiAUGN	Auger electron coefficient (see ModUsrRecAuger)	[m^6 s^{-1}]
GloVhP1	ParMatxyzAUGP	ParMatSiAUGP	Auger hole coefficient (see ModUsrRecAuger)	[m^6 s^{-1}]
GloVhP1	ParMatxyzTAUN	ParMatSiTAUN	Electron SRH lifetime (not concentration dependent)	[s]
GloVhP1	ParMatxyzTAUP	ParMatSiTAUP	Hole SRH lifetime (not concentration dependent)	[s]
GloVhP1	ParMatxyzTAUN0	ParMatSiTAUN0	Electron SRH lifetime (ModUsrRecSRHConcDepLifeTime)	[s]
GloVhP1	ParMatxyzTAUP0	ParMatSiTAUP0	Hole SRH lifetime (ModUsrRecSRHConcDepLifeTime)	[s]
GloVhP1	ParMatxyzNSRH	ParMatSiNSRH	Electron SRH concentration (ModUsrRecSRHConcDepLifeTime)	[m^{-3}]
GloVhP1	ParMatxyzPSRH	ParMatSiPSRH	Hole SRH concentration (ModUsrRecSRHConcDepLifeTime)	[m^{-3}]
GloVhP1	ParMatxyzITAU	ParMatSiITAU	Inverse linear recombination lifetime	[s^{-1}]
GloVhP1	ParMatxyzEG300	ParMatSiEG300	Bandgap at 300	[eV]
GloVhP1	ParMatxyzEGALPH	ParMatSiEGALPH	Bandgap Varshini temperature coefficient	[TODO]
GloVhP1	ParMatxyzEGBETA	ParMatSiEGBETA	Bandgap Varshini temperature coefficient	[TODO]
GloVhP1	ParMatxyzBGNV0	ParMatSiBGNV0	Slotboom BGN coef.	[V]
GloVhP1	ParMatxyzBGNN0	ParMatSiBGNN0	Slotboom BGN coef.	[cm^{-3}]
GloVhP1	ParMatxyzBGNCON	ParMatSiBGNCON	Slotboom BGN coef.	[adim]
GloVhP1	ParMatxyzNC300	ParMatSiNC300	Effective conduction band density of state at 300 K	[cm^{-3}]
GloVhP1	ParMatxyzNV300	ParMatSiNV300	Effective valence band density of state at 300 K	[cm^{-3}]
GloVhP1	ParMatxyzNCF	ParMatSiNCF	Effective conduction band density of state temperature coef.	[adim]
GloVhP1	ParMatxyzNVF	ParMatSiNVF	Effective valence band density of state temperature coef.	[adim]
real[int]	ParMatBBEPhonons(ParMatSiNPH0)		Smith band-to-band absorption coeff. param.	TODO
real[int]	ParMatBBE1(ParMatSiNPH0)		Smith band-to-band absorption coeff. param.	TODO
real[int]	ParMatBBB(ParMatSiNPH0)		Smith band-to-band absorption coeff. param.	TODO
real[int]	ParMatBBEXP(ParMatSiNPH0)		Smith band-to-band absorption coeff. param.	TODO
	ParMatBBEPhonons	ParMatSiBBEPhonons		
	ParMatBBE1	ParMatSiBBE1		
	ParMatBBB	ParMatSiBBB		
	ParMatBBEXP	ParMatSiBBEXP		
real	ParMatBBEta	ParMatSiBBEta	Smith band-to-band absorption coeff. param.	TODO

Table 4.3: User variables: Silicon (Si) material electronic coefficients

Type	Name	Default	Description	Unit
real	ParMatSiD	18 * ConvC2M^2	Ambipolar diffusion coefficient	[$m^2 s^{-1}$]
real	ParMatSiDn	25.8529 * ConvC2M^2	electron diffusion coefficient	[$m^2 s^{-1}$]
real	ParMatSiDp	12.92645 * ConvC2M^2	hole diffusion coefficient	[$m^2 s^{-1}$]
real	ParMatSiMobn	1e3 * ConvC2M^2	electron mobility	[$m^2 V^{-1}$ s
real	ParMatSiMobp	5e2 * ConvC2M^2	hole mobility	[$m^2 V^{-1}$ s
real	ParMatSiNC300	2.8e19 * ConvC2M^(-3)	Conduction band effective density of state at 300K	[m^{-3}]
real	ParMatSiNV300	1.04e19 * ConvC2M^(-3)	Valenc band effective density of state at 300K	[m^{-3}]
real	ParMatSiNCF	1.5	Coef. to compute eff. density of state	[adim]
real	ParMatSiNVF	1.5	Coef. to compute eff. density of state	[adim]
real	ParMatSiCDIR	0	Direct recombination coefficient	[$m^3 s^{-1}$]
real	ParMatSiAUGN	2.8e-31 * ConvC2M^6	Auger coef. for electrons	[$m^6 s^{-1}$]
real	ParMatSiAUGP	9.9e-32 * ConvC2M^6	Auger coef. for holes	[m^6 s^{-1}]
real	ParMatSiAUGNLLI	2.2e-31 * ConvC2M^6	Low Level Injection coeff for electrons	[m^6 s^{-1}]
real	ParMatSiAUGPLLI	9.9e-32 * ConvC2M^6	Low Level Injection coeff for hole	[m^6 s^{-1}]
real	ParMatSiAUGHLI	1.66e-30 * ConvC2M^6	High Level Injection	[m^6 s^{-1}]
real	ParMatSiTAUN	1e-7	Cst SRH rec. lifetime for electrons	[s]
real	ParMatSiTAUP	1e-7	Cst SRH rec. lifetime for electrons	[s]
real	ParMatSiTAUN0	3.94e-4	Conc. dep. SRH rec. lifetime for electrons P. C. Dhanasekaran Solid-State Elec. 25(8) 719 (1982)	[s]
real	ParMatSiTAUP0	3.94e-5		
real	ParMatSiNSRH	7.1e15 * ConvC2M^(-3)	Conc. dep. SRH rec. P. C. Dhanasekaran Solid-State Elec. 25(8) 719 (1982)	[m^{-3}]
real	ParMatSiPSRH	7.1e15 * ConvC2M^(-3)	Conc. dep. SRH rec. P. C. Dhanasekaran Solid-State Elec. 25(8) 719 (1982)	[m^{-3}]
real	ParMatSiSREC	1e2 * ConvC2M	Surface recombination velocity	[m s^{-1}]
real	ParMatSiITAU	0.0	inverse of rec. lifetime (used for linear problem)	[s^{-1}]
real	ParMatSiEG300	1.08	Band gap at 300 K (pure Si)	[eV]
real	ParMatSiEGALPH	4.73e-4	Band gap temperature coefficient	[eV/K]
real	ParMatSiEGBETA	636	Band gap temperature coefficient	[K]
real	ParMatSiBGNV0	6.92e-3	Slotboom BGN coef.	[eV]
real	ParMatSiBGNN0	1.3e17 * ConvC2M^(-3)	Slotboom BGN coef.	[m^{-3}]
real	ParMatSiBGNCON	0.5	Slotboom BGN coef.	[adim]
int	ParMatSiNPHO	4	Number of phonons participating to the transition	[adim]
real[int]	ParMatSiBBEPhonons(ParMatSiNPHO)		Participating phonon energies	[eV]
	ParMatSiBBEPhonons[0]	55.3e-3		
	ParMatSiBBEPhonons[1]	55.3e-3		
	ParMatSiBBEPhonons[2]	55.3e-3		
	ParMatSiBBEPhonons[3]	55.3e-3		
real[int]	ParMatSiBBB(ParMatSiNPHO)		TODO	[m-1]
	ParMatSiBBB[0]	7.5859e3 * ConvC2M^(-1)		
	ParMatSiBBB[1]	7.5859e3 * ConvC2M^(-1)		
	ParMatSiBBB[2]	7.5859e3 * ConvC2M^(-1)		
	ParMatSiBBB[3]	7.5859e3 * ConvC2M^(-1)		
real[int]	ParMatSiBBE1(ParMatSiNPHO)		TODO	[eV]
	ParMatSiBBE1[0]	3.4		
	ParMatSiBBE1[1]	4.4		
	ParMatSiBBE1[2]	5.2		
	ParMatSiBBE1[3]	5.4		
real[int]	ParMatSiBBEXP(ParMatSiNPHO)		TODO	[eV]
	ParMatSiBBEXP[0]	2		

	ParMatSiBBEXP[1]	2		
	ParMatSiBBEXP[2]	2		
	ParMatSiBBEXP[3]	2		
real	ParMatSiBBEta	3.45	Real part of refractive index	[adim]

Table 4.4: User variables: Silicon (Si) material thermal coefficients

Type	Name	Default	Description	Units
`real`	`ParMatSiK`	`1.5 * ConvC2M^(-1)`	Thermal conductivity	[W m^{-1} K^{-1}]
`real`	`ParMatSiRho`	`2.329e3`	Mass density	[kg m^{-3}]
`real`	`ParMatSiCp`	`7e2`	Mass calorific capacity	[J kg^{-1} K^{-1}]

4.3 Fields

Table 4.5: Field variables: Semiconductor related fields

Type	Name	Default	Description	Units
GloVhP1	GloVarSemiEXC	0.0	Excess carrier concentration	$[m^{-3}]$
GloVhP1	GloVarSemiEXCp	0.0	Excess carrier concentration at previous time	$[m^{-3}]$
GloVhP1	GloVarSemiEXCbias	0.0	Bias concentration. Used in conjunction with nonlinear steady periodic equation.	$[m^{-3}]$
GloVhP1<complex>	GloVarSemiEXCc	0.0	Complex excess carrier concentration (generic usage)	$[m^{-3}]$
GloVhP1<complex>	GloVarSemiEXCH0	0.0	DC excess carrier. Used in conjunction with nonlinear steady periodic equation.	$[m^{-3}]$
GloVhP1<complex>	GloVarSemiEXCH1	0.0	Fondamental harmonic excess carrier. Used in conjunction with nonlinear steady periodic equation.	$[m^{-3}]$
GloVhP1<complex>	GloVarSemiEXCH2	0.0	Second harmonic excess carrier. Used in conjunction with nonlinear steady periodic equation.	$[m^{-3}]$
GloVhP1	GloVarSemiELE	0.0	Total free electron concentration	[m−3]
GloVhP1	GloVarSemiHOL	0.0	Total free hole concentration	$[m^{-3}]$
GloVhP1	GloVarSemiDOPTOT		Total doping concentration (Donnor + Acceptors)	$[m^{-3}]$
GloVhP1	GloVarSemiELE0		Equilibrium electron concentration	$[m^{-3}]$
GloVhP1	GloVarSemiHOL0		Equilibrium hole concentration	$[m^{-3}]$
GloVhP1	GloVarSemidELEdFEn		dGloVarSemiELE/dGloVarSemiFEn	$[m^{-3}\ eV^{-1}]$
GloVhP1	GloVarSemidHOLdFEp		dGloVarSemiHOL/dGloVarSemiFEp	$[m^{-3}\ eV^{-1}]$
GloVhP1	GloVarSemiGenRecLinear		Debugging purpose linear (in the excess carrier) recombination By convention, it is positive when the recombination rate is larger than the (thermal-)generation rate.	$[m^{-3}\ s^{-1}]$
GloVhP1	GloVarSemidGenRecLineardELE		dGloVarSemiGenRecLinear/dGloVarSemiELE	$[s^{-1}]$
GloVhP1	GloVarSemidGenRecLineardHOL		dGloVarSemiGenRecLinear/dGloVarSemiHOL	$[s^{-1}]$
GloVhP1	GloVarSemiGenRecDirect		Net direct (band-to-band, radiative) generation/recombination rate. By convention, it is positive when the recombination rate is larger than the (thermal-)generation rate.	$[m^{-3}\ s^{-1}]$
GloVhP1	GloVarSemidGenRecDirectdELE		dGloVarSemiGenRecDirect/dGloVarSemiELE	$[s^{-1}]$
GloVhP1	GloVarSemidGenRecDirectdHOL		dGloVarSemiGenRecDirect/dGloVarSemiHOL	$[s^{-1}]$
GloVhP1	GloVarSemiGenRecSRH		Net SRH generation/recombination rate. By convention, it is positive when the recombination rate is larger than the (thermal-)generation rate.	$[m^{-3}\ s^{-1}]$
GloVhP1	GloVarSemidGenRecSRHdELE		dGloVarSemiGenRecSRH/dGloVarSemiELE	$[s^{-1}]$
GloVhP1	GloVarSemidGenRecSRHdHOL		dGloVarSemiGenRecSRH/dGloVarSemiHOL	$[s^{-1}]$
GloVhP1	GloVarSemiGenRecAuger		Net Auger generation/recombination rate. By convention, it is positive when the recombination rate is larger than the (thermal-)generation rate.	$[m^{-3}\ s^{-1}]$
GloVhP1	GloVarSemidGenRecAugerdELE		dGloVarSemiGenRecAuger/dGloVarSemiELE	$[s^{-1}]$
GloVhP1	GloVarSemidGenRecAugerdHOL		dGloVarSemiGenRecAuger/dGloVarSemiHOL	$[s^{-1}]$
GloVhP1	GloVarSemiGenRecTOT		Total (thermal-)generation/recombination rate.	$[m^{-3}\ s^{-1}]$
GloVhP1	GloVarSemidGenRecTOTdELE		dGloVarSemiGenRecTOT/dGloVarSemiELE	$[s^{-1}]$
GloVhP1	GloVarSemidGenRecTOTdHOL		dGloVarSemiGenRecTOT/dGloVarSemiHOL	$[s^{-1}]$
GloVhP1	GloVarSemiDIFFELE		Electron diffusion coefficient	$[m^2\ s^{-1}]$
GloVhP1	GloVarSemiDIFFHOL		Hole diffusion coefficient	$[m^2\ s^{-1}]$
GloVhP1	GloVarSemidDIFFXdY		dGloVarSemiDIFFX/dY where X,Y are electron and/or hole concentrations.	$[m^2\ s^{-1}]$

GloVhP1	GloVarSemiDIFFA	Ambipolar diffusion coefficient	[m^2 s^{-1}]
GloVhP1	GloVarSemidDIFFAdEXC	dGloVarSemiDIFFA/dGloVarSemiEXC.	[m^5 s^{-1}]
GloVhP1	GloVarSemiMOBELE	The electron mobility	[m^2 V^{-1} s^{-1}]
GloVhP1	GloVarSemiMOBHOL	The hole mobility	[m^2 V^{-1} s^{-1}]
GloVhP1	GloVarSemidMOBELEdELE	dGloVarSemiMOBELE/dGloVarSemiELE	[m^5 V^{-1} s^{-1}]
GloVhP1	GloVarSemidMOBELEdHOL	dGloVarSemiMOBELE/dGloVarSemiHOL	[m^5 V^{-1} s^{-1}]
GloVhP1	GloVarSemidMOBHOLdELE	dGloVarSemiMOBHOL/dGloVarSemiELE	[m^5 V^{-1} s^{-1}]
GloVhP1	GloVarSemidMOBHOLdHOL	dGloVarSemiMOBHOL/dGloVarSemiHOL	[m^5 V^{-1} s^{-1}]
GloVhP1	GloVarSemiEGAP	The band gap energy	[eV]
GloVhP1	GloVarSemiRECITAU	Inverse of the recombination time (linear term)	[s^{-1}]
GloVhP1	GloVarSemiNIE	Intrinsic effective carrier concentration	[m^{-3}]
GloVhP1	GloVarSemiNCT	Conduction band effective carrier density	[m^{-3}]
GloVhP1	GloVarSemiNVT	Valence band effective carrier density	[m^{-3}]
GloVhP1	GloVarSemiFEn	Electron quasi Fermi level relative to the conduction band (= Ef_n - E_c).	[eV]
GloVhP1	GloVarSemiFEp	Hole quasi Fermi level relative to the valence band (= E_v - Ef_p)	[eV]
GloVhP1	GloVarSemidFEndELE	dGloVarSemiFEn/dGloVarSemiELE	
GloVhP1	GloVarSemidFEpdHOL	dGloVarSemiFEp/dGloVarSemiHOL	
GloVhP1	GloVarSemiPROBAELE	Dummy control variable: probabiliy of finding an electron at the bottom edge of the conduction band	[adim]
GloVhP1	GloVarSemiPROBAHOL	Dummy control variable: probabiliy of finding a hole at the top edge of the valence band	[adim]
GloVhP1[int]	GloVarSemiALPHA(ParUsrLaserNMAX)	Absorption coefficient of each laser.	[m^{-1}];
GloVhP1[int]	GloVarSemiCARRGEN(ParUsrLaserNMAX)	Carrier generation of each laser	[m^{-3} s^{-1}]
GloVhP1	GloVarSemiCARRGENTOT	Total carrier generation	[m^{-3} s^{-1}]
GloVhP1<complex>	GloVarSemiCARRGENTOTc	Total carrier generation	[m^{-3} s^{-1}]
GloVhP1[int]	GloVarSemiHOTHEATGEN(ParUsrLaserNMAX)	Heat generated by hot carriers of each laser	[W m^{-3}]
GloVhP1	GloVarSemiRECHEATGEN	Heat generated by carrier recombination	[W m^{-3}]

Table 4.6: Field variables: Temperature related fields

Type	Name	Default	Description	Units
GloVhP1	GloVarTempT	300.0	Lattice temperature	[K]
GloVhP1	GloVarTempTp	300.0	Lattice temperature at previous time	[K]
GloVhP1<complex>	GloVarTempTc	300.0	Complex lattice temperature	[K]
GloVhP1<complex>	GloVarTempTH1	0	Lattice temperature first harmonic.	[K]
GloVhP1<complex>	GloVarTempTH2	0	Lattice temperature second harmonic.	[K]
GloVhP1	GloVarTempK		Thermal conductivity	[W m^{-1} K^{-1}]
GloVhP1	GloVarTempRhoCp		Volume calorific capacity	[J m^{-3} K^{-1}]
GloVhP1	GloVarHeatGen		Heat generation	[W m^{-3}]
GloVhP1<complex>	GloVarHeatGenc		Heat generation (complex)	[W m^{-3}]

4.4 Model flags

Table 4.7: Flags: Semiconductor related Flags

Type	Name	Default	Description	Value
int	ModUsrNP	0	Chose between different strategies to compute excess carrier concentration.	See next lines
int	ModCSTNPBasic	0	Uniform doping. GloVarSemiELE = GloVarSemiELE0 + GloVarSemiEXC; GloVarSemiHOL = GloVarSemiHOL0 + GloVarSemiEXC;	
int	ModCSTNPCorrected	1	Uniform doping. Total electron and hole concentrations are ensured to be positive. GloVarSemiELE = ((GloVarSemiEXC)¡0)?GloVarSemiELE0:(GloVarSemiELE0 + GloVarSemiEXC); GloVarSemiHOL = ((GloVarSemiEXC)¡0)?GloVarSemiHOL0:(GloVarSemiHOL0 + GloVarSemiEXC);	
int	ModCSTNPMFC	2	Non-Uniform doping. The diffusion equation is actually solved for the Mean Free Carrier Concentration (MFC=GloVarSemiEXC). GloVarSemiELE = GloVarSemiEXC + (GloVarSemiELE0 - GloVarSemiHOL0) / 2.0; GloVarSemiHOL = GloVarSemiEXC + (GloVarSemiHOL0 - GloVarSemiELE0) / 2.0;	
int	ModUsrBGNSlotboom	0	Slotboom(doping) Band Gap Narrowing	0—1
int	ModUsrBGNSchenk	0	Schenk(doping+carrier) Band Gap Narrowing	0—1
int	ModUsrBGNTemp	0	Varshini(temperature) Bang Gap Narrowing	0—1
int	ModUsrRecLinear	0	Linear gen/rec (mainly for debug purpose)	0—1
int	ModUsrRecDirect	0	Direct gen/rec (band-to-band radiative recombination)	0—1
int	ModUsrRecAuger	0	Auger gen/rec	0—1
int	ModUsrRecAuger2	0	Auger2 gen/rec	0—1
int	ModUsrRecSRH	0	SRH gen/rec	0—1
int	ModUsrRecSRHConcDepLifeTime	0	Concentration dependent lifetimes. ModUsrRecSRH must be 1.	0—1
int	ModUsrRecLinearMass	0	Linear recombination term (mass matrix for linear solver).	0—1
int	ModUsrRecSRHMassH	0	SRH Linear recombination term (on the mass matrix) for harmonic calculations.	0—1
int	ModUsrRecAugerMassH	0	Auger Linear recombination term (on the mass matrix) for harmonic calculations.	0—1
int	ModUsrBTBTable	0	Band-To-Band absorption coefficient from table.	0—1
int	ModUsrBTBIP	0	Band-To-Band absorption coefficient from Smith.	0—1
int	ModUsrOptCarrGen	0	Band-to-band optical carrier generation	0—1
int	ModUsrGenH2	0	Pseudo generation term for second harmonic calculations.	0—1
int	ModUsrGenSRHH2	0	SRH pseudo generation term for second harmonic calculations.	0—1
int	ModUsrGenAugerH2	0	Auger pseudo generation term for second harmonic calculations.	0—1
int	ModUsrMob	0	Carrier mobility	See next lines
int	ModCSTMobCst	0	-Constant mobility	
int	ModCSTMobDORLET	1	-Dorkel and Leturcq mobility model	
int	ModCSTMobPHUNI	2	-Phillips Unified mobility model	
int	ModUsrDiff	0	Carrier diffusivity	See next lines
int	ModCSTDiffCst	1	-Constant carrier diffusivity	
int	ModCSTDiffFermifromMob	2	-Carrier diffusivity from mobility considering Fermi-Dirac statistics	

int	ModCSTDiffBoltzfromMob	3	-Carrier diffusivity from mobility considering Boltzmann statisitics	
int	ModUsrDiffAmbi	0	Ambipolar diffusivity	See next lines
int	ModCSTDiffAmbiCst	0	-Constant ambipolar diffusivity	
int	ModCSTDiffAmbifromDiff	1	-Ambipolar diffusivity calculated from previously defined carrier diffusivities	

Table 4.8: Flags: Tmperature related Flags

Type	Name	Default	Description	Value
int	ModUsrHeatConductivityCst	0	Constant conductivity model	0—1
int	ModUsrHeatRhoCpCst	0	Constant volume calorific capacity (Rho*Cp) model	0—1
int	ModUsrHeatUserSpec	0	User specified volume heat source	0 —1
int	ModUsrHeatHotCarrier	0	Heat generation by hot carriers	0—1
int	ModUsrHeatRec	0	Heat generation by recombination	0—1
int	ModUsrHeatGenH1	0	Heat generation for first harmonic	0—1
int	ModUsrHeatGenH2	0	Heat generation for second harmonic	0—1

4.5 Functions

Table 4.9: Functions: Semiconductor related functions

Type	Name	Description	Outputs
real	ComputeEquilNP()	Compute equilibrium electron and hole concentration.	GloVarSemiELE0, GloVarSemiHOL0
real	ComputeNP()	Compute non-equilibrium electron and hole concentrations. Concentration is ensured larger than the equilibrium ones (that could be a wrong assumption is some cases!)	
real	ComputeEGap()	Compute energy gap	GloVarSemiEGAP
real	ComputeMobility()	Compute mobilities	GloVarSemiMOBELE, GloVarSemiMOBHOL, GloVarSemidMOBELEdELE, GloVarSemidMOBELEdHOL, GloVarSemidMOBHOLdELE, GloVarSemidMOBHOLdHOL
real	ComputeDiff()	Compute diffusivity	GloVarSemiDIFF, GloVarSemidDIFFdEXC
real	ComputeDiffAmbi()	Compute ambipolar diffusivity	GloVarSemiDIFFA, GloVarSemidDIFFAdEXC
real	ComputeFermiNXT()	Compute effective density of states	GloVarSemiNCT, GloVarSemiNVT
real	ComputeFermiNie()	Compute the intrinsic effective carrier density	GloVarSemiNIE
real	ComputeRecVol()	Compute the volume carrier generation/recombination rates	GloVarSemiGenRecAuger, GloVarSemidGenRecAugerdELE, GloVarSemidGenRecAugerdHOL, GloVarSemiGenRecSRH, GloVarSemidGenRecSRHdELE, GloVarSemidGenRecSRHdHOL, GloVarSemiRECITAU
real	ComputeFEfromN()	Compute the quasi-fermi levels from the carrier concentration	GloVarSemiFEn, GloVarSemiFEp, GloVarSemidFEndELE, GloVarSemidFEpdHOL
real	ComputeNfromFE()	Compute the carrier concentrations from the quasi-fermi levels	GloVarSemiELE, GloVarSemiHOL, GloVarSemidELEdFEn, GloVarSemidHOLdFEp
real	ComputeAbsCoef()	Compute optical absorption coefficients	GloVarSemiALPHA[ParUsrLaserN]
real	ComputeGenVol()	Compute optical volume generation rates	GloVarSemiCARRGEN[ParUsrLaserN]

Table 4.10: Functions: Temperature related functions

Type	Name	Description	Outputs
real	ComputeThermalConductivity()	Compute thermal conductivity	GloVarTempK
real	ComputeThermalRhoCp()	Compute thermal Rho * Cp	GloVarTempRhoCp
real	ComputeHeatGen()	Compute heat generation	GloVarSemiHOTHEATGEN[ParUsrLaserN], GloVarSemiRECHEATGEN
real	ComputeTotalHeatGen()	Compute total heat generation	GloVarHeatGen

4.6 Solvers

Table 4.11: Solve functions

Type	Name	Description
real	SolveCarrLinearTimeStep(real[int] & PrevSol, real InvDt)	Solve the excess carrier linear time dependent diffusion equation
real	SolveCarrLinearSteady(bool isUniform)	Solve the excess carrier linear steady state diffusion equation. The previous solution, GloVarSemiEXC, is used for coefficient calculation.
real	SolveCarrLinearSteady()	
real	SolveCarrLinearSteadyPeriodic()	Solve the excess carrier linear steady periodic (harmonic) diffusion equation
real	SolveCarrNonlinearSteady(bool isUniform)	Solve the excess carrier nonlinear steady state diffusion equation
real	SolveCarrNonlinearSteady()	
real	SolveCarrNonlinearTimeStep(real[int] & PrevSol, real InvDt)	Solve the excess carrier nonlinear time dependent diffusion equation
real	SolveCarrNonlinearSteadyPeriodic()	Solve the excess carrier (pseudo) nonlinear steady periodic (harmonic) diffusion equation
real	SolveCarrNonlinearSteadyPeriodicV2(bool SolveH1, bool SolveH2)	Solve the excess carrier (pseudo) nonlinear steady periodic (harmonic) diffusion equation. Version 2. The probe and the pump are assumed to be laser 0 and 1 respectively. The peak (and not the continuous component) irradiance value have to be specified. The model flags must be set as if solving the nonlinear system.
real	SolveCarrNonlinearSteadyPeriodicV2()	SolveCarrNonlinearSteadyPeriodicV2(true, true)
real	SolveTempLinearSteady()	Solve the temperature linear steady state diffusion equation
real	SolveTempLinearSteadyPeriodic()	Solve the excess carrier linear steady periodic (harmonic) diffusion equation
real	SolveTempLinearTimeStep(real[int] & PrevSol, real InvDt)	Solve the excess carrier linear time dependent diffusion equation
real	SolveTempCarrNonlinearSteadyPeriodic(bool SolveH1, bool SolveH2)	Solve the excess temperature (pseudo) linear steady periodic (harmonic) diffusion equation. Version 2. The probe and the pump are assumed to be laser 0 and 1 respectively. The peak (and not the continuous component) irradiance value have to be specified.
real	SolveTempCarrNonlinearSteadyPeriodic()	SolveTempLinearSteadyPeriodicV2(true, true)

Table 4.12: User variables: Solver parameters

Type	Name	Default	Description	Units
int	UsrNewtonNbrMaxIter		Maximum number of Newton iterations	[adim]
int	UsrNewtonNbrIter		Number of Newton iterations at the end the Newton loop	[adim]
real	UsrNewtonRelax		Newton relaxation coefficient	[adim]
real	UsrNewtonEpsIncr		Maximum increment convergence criterium	depends
real	UsrNewtonEpsRes		Maximum residue convergence criterium	depends
real	UsrLinearEps		Convergence criterium in linear solver	[adim]
int	UsrNewtonNbrMaxIterCarr	20	Maximum number of Newton iterations (carrier equation)	[adim]
real	UsrNewtonRelaxCarr	1	Newton relaxation coefficient (carrier equation)	[adim]
real	UsrNewtonEpsIncrCarr	1e14	Maximum increment convergence criterium (carrier equation)	[cm-3]

real	UsrNewtonEpsResCarr	1e12	Maximum residue convergence criterium (carrier equation)	[cm-3]
real	UsrLinearEpsCarr	1e-12	Convergence criterium in linear solver (carrier equation)	[adim]
int	UsrNewtonNbrMaxIterTemp	20	Maximum number of Newton iterations (temperature equation)	[adim]
real	UsrNewtonRelaxTemp	1	Newton relaxation coefficient (temperature equation)	[adim]
real	UsrNewtonEpsIncrTemp	1e-3	Maximum increment convergence criterium (temperature equation)	[K]
real	UsrNewtonEpsResTemp	1e-3	Maximum residue convergence criterium (temperature equation)	[K]
real	UsrLinearEpsTemp	1e-12	Convergence criterium in linear solver (temperature equation)	[adim]

4.7 Mesh

Table 4.13: Mesh variables

Type	Name	Default	Description	Units
real	GloScaX	10000 * ConvU2M	x (or r) coordinate scaling factor	[m]
real	GloScaY	10000 * ConvU2M	y coordinate scaling factor	[m]
mesh	GloMesh	square(10, 10)		
int	GloMeshP1ndof			
	GloMeshP1ndof	VhDummy.ndof		
fespace	GloVhP1(GloMesh	, P1)	fespace on GloMesh	NA
GloVhP1	GloMeshCOORDX	x	x coordinate of each node	[adim]
GloVhP1	GloMeshCOORDY	y	y coordinate of each node	[adim]
	GloMeshCOORDX	x		
	GloMeshCOORDY	y		

Table 4.14: Mesh functions

Type	Name	Description	Default
real	GloMeshP1setndof()	DEV	
real	GloMeshCoordReset()	DEV	
	GloMeshFieldInterpol(var1)	DEV	
	GloMeshFieldReset(var1)	DEV	

Table 4.15: Mesh adaptation functions

Type	Name	Description	Default
real	GloMeshGetAdaptFunc(real[int] &AdaptFunc, real[int] &a)	Compute adaptation function based on a.	
real	GloMeshGetAdaptFunc(real[int] &AdaptFunc, real[int] &a, real[int] &b)	Compute adaptation function based on a and b	
real	GloMeshGetAdaptFunc(real[int] &AdaptFunc, real[int] &a, real[int] &b, real[int] &c)	Compute adaptation function based on a, b and c	
real	GloMeshGetAdaptFunc(real[int] &AdaptFunc, real[int] &a, real[int] &b, real[int] &c, real[int] &d)	Compute adaptation function based on a, b, c and d	
real	GloMeshGetAdaptFunc(real[int] &AdaptFunc, real[int] &a, real[int] &b, real[int] &c, real[int] &d, real[int] &e)	Compute adaptation function based on a, b, c, d and e	
real	GloMeshGetAdaptFunc(real[int] &AdaptFunc, real[int] &a, real[int] &b, real[int] &c, real[int] &d, real[int] &e, real[int]& f)	Compute adaptation function based on a, b, c, d, e and f	
	GloMeshGetPreAdapt(AdaptFunc)	Compute adaptation function based on ParUsrNDOP, ParUsrPDOP and an estimation of GloVarSemiCARRGENTOT. • real[int] AdaptFunc: the output adapt function	

`GloMeshAdapt(Mesh, Solution, error, minFeature, maxFeature, NNodes, NSmooth, OnExc, OnSol)`

Compute adaptation function based on ParUsrNDOP, ParUsrPDOP, laser radius and Solution,

- mesh Mesh: the mesh
- Vh Solution: the solution
- real error: error (0.01)
- real minFeature: minimum mesh size dimension [adim]
- real maxFeature: maximum mesh size dimension [adim]
- int NNoded: number of nodes.
- int NSmooth: number of smoothing steps
- bool OnExc: Refines on exctitaions (laser)
- bool OnSol: Refines on Solution

Table 4.16: Excess carrier contact definition

Type	Name	Default	Description	Units
int[int]	GloCarrContactLabel(10)		Label of the border that will be a Ohmic contact.	[adim]
	GloCarrContactLabel	0		
real[int]	GloCarrContactEXC(10)		Excess carrier value at Ohmic contact. Should be 0 m^{-3} but other values are allowed (for fun?).	[m-3]
	GloCarrContactEXC	0		

Table 4.17: Temperature contact definition

Type	Name	Default	Description	Units
int[int]	GloTempContactLabel(10)		Label of the border that will be a isothermal contact.	[adim]
	GloTempContactLabel	0		
int[int]	GloTempContactLabelc(10)		Label of the border that will be a isothermal contact.	[adim]
	GloTempContactLabelc	0		
real[int]	GloTempContactT(10)		Temperature value at isothermal contact.	[K]
	GloTempContactT	300		
complex[int]	GloTempContactTc(10)		Temperature value at isothermal contact.	[K]
	GloTempContactTc	0		

4.8 Visualization

Table 4.18: Post-processing

Type	Name	Description
	`GloCutlineSave(xmin, ymin, xmax, ymax, npts, Var, filename, show, ScaX, ScaY)`	Make a 1d cross section (cutline) and save data in a file. • real xmin, ymin: line origin coordinate [m]. • real xman, ymax: line end coordinate [m]. • int npts: number of points in cut line. • Var: variable defined on a FE-space. • boolean show: draw a plot on screen. • real ScaX, ScaY: x and y scaling factors [m].
	`GloCutline(llvar, ll, xmin, ymin, xmax, ymax, npts, Var, ScaX, ScaY)`	Make a 1D cross section (cutline) • real[int] llvar(npts)(O): field values along the cutline • real[int] ll(npts)(O): coordinate along the cutline [m] • real xmin, ymin: line origin coordinate [m]. • real xman, ymax: line end coordinate [m]. • int npts: number of points in cut line. • Var: variable defined on a FE-space. • real ScaX, ScaY: x and y scaling factors [m].
	`GloSaveMatVert(filename, nlines, format)`	Save series of vectors vertically • string filename: output filename • int nlines: number of lines • format: output format Example: GloSaveMatVert(”outfile.dat”, 25, a[i] << ” ” << b[i] << endl)
	`GloSaveVec(file, vector)`	Append vector horizontaly in file. • ofstream file(filename): file descriptor • real[int] vector: vector to save

	`GloSetLogScale(PlotMin, PlotNIso, MyLogScale)`	Set logarithmic scale. • real PlotMin: Scale minimum value. • int PlotNIso: Number of isovalue. • real[int] MyLogScale(PlotNIso):(O) Output scale.

Table 4.19: Export to gmsh

Type	Name	Description
	`GloSaveGmshMeshV1(Th, filename, ConvFactor, ScaX, ScaY)`	Save triangulation (mesh) Th in Gmsh mesh v1.0. • mesh Th: the mesh • string filename: mesh filename • real ConvFactor: units factor • real ScaX, ScaY: scaling factor
	`GloSaveGmshPosSPARSED(Th, Quantity, QuantityName, filename, ConvFactor, ScaX, ScaY)`	obsolete (do work but only point are shown (no smooth interpolation))
	`GloSaveGmshPosSPARSEDST(Th, Quantity, QuantityConvFactor, QuantityName, filename, MeshConvFactor, ScaX, ScaY)`	Save variable in Gmsh sparsed mesh . • mesh Th: the mesh • Quantity: FEsV variable • real QuantityConvFactor: units conversion factor • string QuantityName: variable output name • string filename: mesh filename • real MeshConvFactor: units factor • real ScaX, ScaY: scaling factor

`GloSaveGmshPosSPARSEDSTComplex(Th, QuantityComplex, QuantityConvFactor, QuantityName, filename, MeshConvFactor, ScaX, ScaY)`	Save complex variable in Gmsh sparsed mesh. • mesh Th: the mesh • QuantityComplex: FEsV¡complex¿ variable • real QuantityConvFactor: units factor • string QuantityName: variable output name • string filename: mesh filename • real MeshConvFactor: units factor • real ScaX, ScaY: scaling factor
`GloSaveGmshPosSPARSEDSTMulti(Th, TimeVector, QuantityVector, QuantityConvFactor, QuantityName, filename, MeshConvFactor, ScaX, ScaY)`	Save variable in Gmsh sparsed mesh for multiple time steps. • mesh Th: the mesh • real[int] TimeVector: output time values • QuantityVector: FEsV[int] variable • real QunatityConvFactor: units factor • string QuantityName: variable output name • string filename: mesh filename • real MeshConvFactor: units factor • real ScaX, ScaY: scaling factor
`GloSaveGmshPosASCII(Th, Quantity, QuantityName, filename)`	DO NOT WORK
`GloSaveGmshPosASCIITriangle(Th, Quantity, QuantityName, filename)`	DO NOT WORK ?????

4.9 Miscellaneous

fsemVERBOSITY

- 0 Show nothing except the usual Freefem outputs.
- 1 Show when entering Solve functions.
- 2 Show when entering linear or Newton solver
- 3 Show preassembling, assembling, residue calculations inside the solver.
- 4 Show when entering coefficient computation functions & detail on assembling.
- 5 Show model flags and min/max value of coefficients.
- 6 Show more on coefficient calculation (interpolation when needed)
- 7 Show details on matrix after assemling (inside a Newton iteration for example)

Chapter 5

Developper's Corner

Adding a new model for the coefficient calculation of an existing partial differential equation is pretty easy with FSEM. This is achieved is two steps. For example, the model for bandgap narrowing (Slotboom) was added as follow:

- **Coefficient calculation.** A new function `ModBandGapDEgSlotBoom()` has to be added to the file /models/modules/Semi/fsem. In this function (go and check the file, the syntax is self-explanatory), the bandgap difference is implemented in Freefem++'s language or alternatively a C++ function is called. The user can provide either or both a Freefem++ or a C++ implementation. The choice of which implementation is actually used is done through the variable `fsemModUseCpp`. If a C++ function is called, a function `CppBandGapDEgSlotBoom()` has to be added to the file /models/modules/Semi/fsem_Mod (go and check the file, the syntax is self-explanatory). Then the C++ file has to be compiled by invoking `make` in the file's directory.
- **Coefficient incorporation in assembling** Now that the coefficient calculation is implemented, FSEM has to know how to use it. This is done by modifying the function `ComputeEGap()` in /models/fsem_Mod_ComputeSemi.edp. Usually, the coefficient calculation will be triggered by a model flag, which must be declared in /models/fsem_Mod_FlagsSemi.edp.

Bibliography