DOS extraction

Generated by Doxygen 1.8.6

Sat Oct 4 2014 18:38:04

Contents

Chapter 1

Index

1.1 Introduction

This program allows to extract the Density of States (DoS), assessed by mean capacitance-voltage measurements, in an organic semiconductor device. Simulated values are fitted to experimental data.

Source and header files are written in C++11 language.

The software is intended to be used on a Unix-like operating system.

1.2 Dependancies

The program requires the following software to be installed on your system:

- **`CMake`** (version 2.8 or above), a cross-platform configuration tool;
- **`Make`** (version 3.8.1 or above), a utility to build executables;
- **`GCC`** (version 4.8 or above), the GNU Compiler Collection;
- **`Eigen`** (version 3.2 or above), to handle with matrices, vectors and linear algebra;
- **`Gnuplot`** (version 4.6.4 or above), a graphical utility to generate plots (the package **gnuplot-x11**, a terminal for X servers, is also required for the interactive interface);
- **`Boost`** (version 1.50 or above), a set of libraries used by gnuplot-iostream;
- **`Doxygen`** (version 1.8.6 or above), a documentation generator (not compulsory).

It also uses the following libraries, shipped in the *include/* folder:

- **`GetPot`** (version 1.1.18), to parse command-line and configuration files;
- **`gnuplot-iostream`** (version 2), a C++ interface for Gnuplot.

Parallel computing capabilities are provided through the **`OpenMP`** library, shipped together with GCC.

1.3 Compile

In order to generate a test executable, simply execute one of these commands in a terminal pointing to the root directory of this package:

```
$ mkdir build
$ cd build
$ cmake ..
$ make
```

or, if you want the compiler to produce also debug symbols:

```
$ mkdir build
$ cd build
$ cmake .. -DCMAKE_BUILD_TYPE=Debug
$ make
```

You can specify the name of the test source to be compiled (without extension, for example: *simulate_dos*) by editing the variable **EXEC**, defined in the *CMakeLists.txt* file.

Repeat these instructions for each test you want to compile.

The following command will generate this documentation under the *doc/* folder (or the one specified in *CMakeLists.txt*):

```
$ make doc
```

1.4 Set up the configurations

Note

The default configuration directory is *config/*.

Before you can run an executable, you have to set up the configuration file (default: *config.pot*). Within it you can find a list of parameters, each of which is commented out to explain what modifying it will entail.

Particularly, the variables *input_params* and *input_experim* can be set, i.e. the filenames where to find input fitting parameters and experimental data respectively. It's recommended (but not compulsory) to put these files in the same directory as the configuration file (otherwise you can specify a relative or absolute path to them).

Warning

The program never checks that the input values are numeric but will always try to cast them to floating point numbers, then please pay attention while setting up the variable *skipHeaders*.

You can create multiple configuration files, each with different parameter values: the one you aim to use can be specified in the command-line before running.

1.5 Run!

Executables are placed under the *bin/* directory (or the one specified in *CMakeLists.txt*).

To run by using the default configuration filename (*config.pot*) simply move to the *bin/* directory and execute:

```
$ ./test_filename
```

To specify a different configuration file previously saved in the configuration directory:

```
$ ./test_filename -f configuration_filename
```

or:

```
$ ./test_filename --file configuration_filename
```

The variable *configuration_filename* should **not** contain the path.

Warning

Furthermore, if you run the program from a different folder than *bin/* or if you chose a different configuration directory, you have to manually specify the **full** path to the configuration directory (either absolute or relative to the current directory) by using:

```
$ ./test_filename -d configuration_directory
```

or:

```
$ ./test_filename --directory configuration_directory
```

Once complete, the results of the simulation(s) will be saved in the output directory (relative to *bin*) specified in the configuration file (default: *output*).

Gnuplot scripts are saved too for later re-use under the *gnuplot/* subdirectory; you can run them through:

```
$ gnuplot name_of_the_script
```

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 constants Namespace Reference

Numerical constants.

Variables

- const Real Q = 1.60217653000000e-19
  Electron charge $[C]$.
- const Real Q2 = Q $*$ Q
  Electron charge squared $\left[C^2\right]$.
- const Real K_B = 1.38065050000000e-23
  Boltzmann's constant $\left[J \cdot K^{-1}\right]$.
- const Real EPS0 = 8.854187817e-12
  Vacuum electrical permittivity $\left[C \cdot V^{-1} \cdot m^{-1}\right]$.
- const Real T = 300
  Reference temperature $[K]$.
- const Real V_TH = K_B $*$ T / Q
  Treshold voltage $[V]$.
- const Index PARAMS_NO = 24
  Number of parameters required in input file.
- const Real PI = M_PI
  π.
- const Real SQRT_PI = std::sqrt(PI)
  $\sqrt{\pi}$.
- const Real PI_M4 = 0.7511255444649425
  $\pi^{-\frac{1}{4}}$.
- const Real SQRT_2 = std::sqrt(2)
  $\sqrt{2}$.

6.1.1 Detailed Description

Numerical constants.

6.2 numerics Namespace Reference

Namespace for generic numeric algorithms.

Functions

- template<typename ScalarType >
  VectorX< ScalarType > sort (const VectorX< ScalarType > &vector)

  Function to sort Eigen vectors.
- template<typename ScalarType >
  VectorXpair< ScalarType > sort_pair (const VectorX< ScalarType > &vector)

  Function to sort Eigen vectors, keeping track of indexes.
- Real trapz (const VectorXr &x, const VectorXr &y)

  Function to compute approximate integral of y with spacing increment specified by x, using trapezoidal rule.
- Real trapz (const VectorXr &y)

  Compute the approximate integral of y with unit spacing, using trapezoidal rule.
- VectorXr deriv (const VectorXr &, const VectorXr &)

  Compute the numeric derivative: $\frac{dy}{dx}$.
- Real interp1 (const VectorXr &, const VectorXr &, const Real &)

  Linear 1D interpolation. Interpolate y, defined at points x, at the point xNew.
- VectorXr interp1 (const VectorXr &, const VectorXr &, const VectorXr &)

  Linear 1D interpolation. Interpolate y, defined at points x, at the points xNew.
- Real error_L2 (const VectorXr &, const VectorXr &, const VectorXr &, const Real &)

  Compute the L^2*-norm error between simulated and interpolated values, using trapz.*

6.2.1 Detailed Description

Namespace for generic numeric algorithms.

6.2.2 Function Documentation

6.2.2.1 VectorX< ScalarType > sort (const VectorX< ScalarType > & *vector*)

Function to sort Eigen vectors.

Template Parameters

ScalarType	: the scalar type.

Parameters

in	*vector*	: the vector to be sorted.

Returns

the sorted vector.

Definition at line 101 of file numerics.h.

6.2.2.2 VectorXpair< ScalarType > sort_pair (const VectorX< ScalarType > & *vector*)

Function to sort Eigen vectors, keeping track of indexes.

Template Parameters

ScalarType	: the scalar type.

Parameters

in	*vector*	: the vector to be sorted.

Returns

an Eigen vector of pairs: (sorted value, corresponding index in the unsorted vector).

Definition at line 110 of file numerics.h.

6.2.2.3 Real trapz (const VectorXr & *x*, const VectorXr & *y*)

Function to compute approximate integral of *y* with spacing increment specified by *x*, using trapezoidal rule.

Parameters

in	*x*	: the vector of the discrete domain;
in	*y*	: the vector of values to integrate.

Returns

the approximate integral value.

Definition at line 15 of file numerics.cc.

6.2.2.4 Real trapz (const VectorXr & *y*)

Compute the approximate integral of *y* with unit spacing, using trapezoidal rule.

Parameters

in	*y*	: the vector of values to integrate.

Returns

the approximate integral value.

Definition at line 31 of file numerics.cc.

6.2.2.5 VectorXr deriv (const VectorXr & *y*, const VectorXr & *x*)

Compute the numeric derivative: $\frac{dy}{dx}$.

Parameters

in	*y*	: the vector of values to differentiate;
in	*x*	: the vector of the discrete domain.

Returns

a vector of the same length as *y* containing the approximate derivative.

Definition at line 36 of file numerics.cc.

6.2.2.6 Real interp1 (const VectorXr & *x*, const VectorXr & *y*, const Real & *xNew*)

Linear 1D interpolation. Interpolate *y*, defined at points *x*, at the point *xNew*.

Parameters

in	*y*	: the vector of values to interpolate;
in	*x*	: the vector of the discrete domain;
in	*xNew*	: the point to interpolate at.

Returns

a scalar containing the interpolated value.

Definition at line 56 of file numerics.cc.

6.2.2.7 VectorXr interp1 (const VectorXr & *x*, const VectorXr & *y*, const VectorXr & *xNew*)

Linear 1D interpolation. Interpolate *y*, defined at points *x*, at the points *xNew*.

Parameters

in	*y*	: the vector of values to interpolate;
in	*x*	: the vector of the discrete domain;
in	*xNew*	: the vector of points to interpolate at.

Returns

a vector of the same length as *xNew* containing the interpolated values.

Definition at line 77 of file numerics.cc.

6.2.2.8 Real error_L2 (const VectorXr & *interp*, const VectorXr & *simulated*, const VectorXr & *V*, const Real & *V_shift*)

Compute the L^2-norm error between simulated and interpolated values, using *trapz*.

Parameters

in	*interp*	: the interpolated values;
in	*simulated*	: the simulated values;
in	*V*	: the vector of the electric potential;
in	*V_shift*	: shift to the electric potential.

Returns

the value of the L^2-norm error.

Definition at line 92 of file numerics.cc.

6.3 utility Namespace Reference

Namespace for utilities and auxiliary functions.

Functions

- std::string full_path (const std::string &, const std::string &)
  Auxiliary function to return the full path to a file.
- void print_block (const char *, std::ostream &=std::cout)
  Auxiliary function to print a string inside a block.
- void print_done (std::ostream &=std::cout)
  Auxiliary function to print a "DONE!" string.

6.3.1 Detailed Description

Namespace for utilities and auxiliary functions.

6.3.2 Function Documentation

6.3.2.1 std::string full_path (const std::string & *filename,* const std::string & *relative_directory*)

Auxiliary function to return the full path to a file.

Parameters

in	*filename*	: the filename;
in	*relative_directory*	: the directory for a relative path.

Returns

the variable *filename*, if it contains an absolute path; otherwise returns the concatenation of *relative_directory* and *filename* (i.e. the relative path to *filename*).

Definition at line 15 of file typedefs.cc.

6.3.2.2 void print_block (const char * *string,* std::ostream & *os* = `std::cout`)

Auxiliary function to print a string inside a block.

Parameters

in	*string*	: the string to print;
out	*os*	: output stream.

Definition at line 20 of file typedefs.cc.

6.3.2.3 void print_done (std::ostream & *os* = `std::cout`)

Auxiliary function to print a "DONE!" string.

Parameters

out	*os*	: output stream.

Definition at line 45 of file typedefs.cc.

Chapter 7

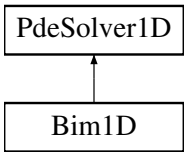
Class Documentation

7.1 Bim1D Class Reference

Class derived from PdeSolver1D, providing a finite volume Box Integration Method (BIM) solver.

`#include <solvers.h>`

Inheritance diagram for Bim1D:



Public Member Functions

- Bim1D ()=delete

  Default constructor (deleted since it is required to specify the mesh).
- Bim1D (VectorXr &)

  Constructor.
- virtual ~Bim1D ()=default

  Destructor (defaulted).
- virtual void assembleAdvDiff (const VectorXr &, const VectorXr &, const VectorXr &, const VectorXr &) override

  Assemble the matrix for an advection-diffusion term.
- virtual void assembleStiff (const VectorXr &, const VectorXr &) override

  Assemble the matrix for a diffusion term.
- virtual void assembleMass (const VectorXr &, const VectorXr &) override

  Assemble the matrix for a reaction term.

Static Public Member Functions

- static VectorXr log_mean (const VectorXr &, const VectorXr &)

  Compute the element-wise logarithmic mean of two vectors.
- static std::pair< VectorXr, VectorXr > bernoulli (const VectorXr &)

  Compute the values of the Bernoulli function.

Additional Inherited Members

7.1.1 Detailed Description

Class derived from PdeSolver1D, providing a finite volume Box Integration Method (BIM) solver.

Matrices are held in a sparse format.

Definition at line 115 of file solvers.h.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 Bim1D (VectorXr & *mesh*)

Constructor.

Parameters

in	*mesh*	: the mesh coordinates.

Definition at line 18 of file solvers.cc.

7.1.3 Member Function Documentation

7.1.3.1 VectorXr log_mean (const VectorXr & *x1*, const VectorXr & *x2*) [static]

Compute the element-wise logarithmic mean of two vectors.

$$M_{log}(x_1,x_2) = \frac{x_2 - x_1}{\log x_2 - \log x_1} = \frac{x_2 - x_1}{\log\left(\frac{x_2}{x_1}\right)} .$$

Parameters

in	*x1*	: the first vector;
in	*x2*	: the second vector.

Returns

the vector of the logarithmic means.

Definition at line 21 of file solvers.cc.

7.1.3.2 std::pair< VectorXr, VectorXr > bernoulli (const VectorXr & *x*) [static]

Compute the values of the Bernoulli function.

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

Parameters

in	*x*	: the vector of the values to compute the Bernoulli function at.

Returns

the pair $(\mathfrak{B}(x), \mathfrak{B}(-x))$.

Definition at line 52 of file solvers.cc.

7.1.3.3 void assembleAdvDiff (const VectorXr & *alpha,* const VectorXr & *gamma,* const VectorXr & *eta,* const VectorXr & *beta*) [override], [virtual]

Assemble the matrix for an advection-diffusion term.

Build the Scharfetter-Gummel stabilized stiffness matrix for: $-\nabla \cdot (\alpha \cdot \gamma(\eta \nabla u - \beta u)) = f$.

Parameters

in	*alpha*	: α, an element-wise constant function;
in	*gamma*	: γ, an element-wise linear function;
in	*eta*	: η, an element-wise linear function;
in	*beta*	: β, an element-wise constant function.

Implements PdeSolver1D.

Definition at line 111 of file solvers.cc.

7.1.3.4 void assembleStiff (const VectorXr & *eps,* const VectorXr & *kappa*) [override], [virtual]

Assemble the matrix for a diffusion term.

Build the standard finite element stiffness matrix for the diffusion problem: $-\nabla \cdot (\varepsilon \cdot \kappa \nabla u) = f$.

Parameters

in	*eps*	: ε, an element-wise constant function;
in	*kappa*	: κ, an element-wise linear function.

Implements PdeSolver1D.

Definition at line 171 of file solvers.cc.

7.1.3.5 void assembleMass (const VectorXr & *delta,* const VectorXr & *zeta*) [override], [virtual]

Assemble the matrix for a reaction term.

Build the lumped finite element mass matrix for the reaction problem: $\delta \cdot \zeta u = f$.

Parameters

in	*delta*	: δ, an element-wise constant function;
in	*zeta*	: ζ, an element-wise linear function.

Implements PdeSolver1D.

Definition at line 180 of file solvers.cc.

The documentation for this class was generated from the following files:

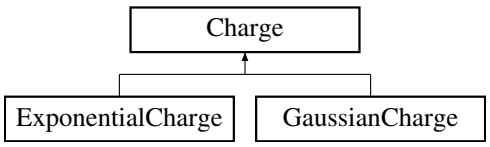
- /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/solvers.h
- /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/solvers.cc

7.2 Charge Class Reference

Abstract class providing methods to calculate total electric charge (the rhs in the Poisson equation).

`#include <charge.h>`

Inheritance diagram for Charge:



Public Member Functions

- Charge ()=delete

  Default constructor (deleted since it is required to specify a ParamList and a QuadratureRule).
- Charge (const ParamList &, const QuadratureRule &)

  Constructor.
- virtual ~Charge ()=default

  Destructor (defaulted).
- virtual VectorXr charge (const VectorXr &phi)=0

  Compute the total charge.
- virtual VectorXr dcharge (const VectorXr &phi)=0

  Compute the derivative of the total charge with respect to the electric potential.

Protected Attributes

- const ParamList & params_

  Parameter list handler.
- const QuadratureRule & rule_

  Quadrature rule handler.

7.2.1 Detailed Description

Abstract class providing methods to calculate total electric charge (the rhs in the Poisson equation).

Definition at line 28 of file charge.h.

7.2.2 Constructor & Destructor Documentation

7.2.2.1 Charge (const ParamList & *params*, const QuadratureRule & *rule*)

Constructor.

Parameters

in	*params*	: the list of simulation parameters;
in	*rule*	: a quadrature rule.

Definition at line 17 of file charge.cc.

7.2.3 Member Function Documentation

7.2.3.1 virtual VectorXr charge (const VectorXr & *phi*) [pure virtual]

Compute the total charge.

Parameters

in	*phi*	: the electric potential φ.

Returns

the total charge $q\,[C]$.

Implemented in ExponentialCharge, and GaussianCharge.

7.2.3.2 virtual VectorXr dcharge (const VectorXr & *phi*) [pure virtual]

Compute the derivative of the total charge with respect to the electric potential.

Parameters

in	*phi*	: the electric potential φ.

Returns

the derivative: $\frac{dq}{d\varphi}\,[C \cdot V^{-1}]$.

Implemented in ExponentialCharge, and GaussianCharge.

The documentation for this class was generated from the following files:

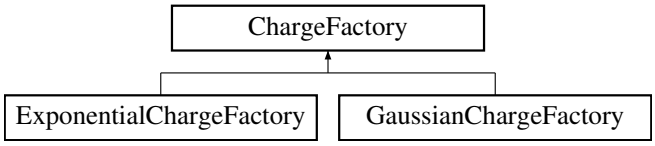
- /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/charge.h
- /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/charge.cc

7.3 ChargeFactory Class Reference

Abstract factory to handle the constitutive relation for the Density of States.

`#include <factory.h>`

Inheritance diagram for ChargeFactory:



Public Member Functions

- ChargeFactory ()=default
  Default constructor (defaulted).
- virtual ~ChargeFactory ()=default
  Destructor (defaulted).
- virtual Charge * BuildCharge (const ParamList &params, const QuadratureRule &rule)=0
  Factory method to build an abstract Charge object.

7.3.1 Detailed Description

Abstract factory to handle the constitutive relation for the Density of States.

Definition at line 28 of file factory.h.

7.3.2 Member Function Documentation

7.3.2.1 virtual Charge* BuildCharge (const ParamList & *params*, const QuadratureRule & *rule*) [pure virtual]

Factory method to build an abstract Charge object.

Parameters

in	*params*	: the list of simulation parameters;
in	*rule*	: a quadrature rule.

Returns

a pointer to Charge.

Implemented in ExponentialChargeFactory, and GaussianChargeFactory.

The documentation for this class was generated from the following file:

- /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/factory.h

7.4 CsvParser Class Reference

Class providing methods to read **numeric** content from a .csv file and to store it in Eigen matrices or vectors.

`#include <csvParser.h>`

Public Member Functions

- CsvParser ()=delete
  Default constructor (deleted since it is required to specify at least a filename).
- CsvParser (const std::string &, const bool &=true)
  Constructor: load the input file and check its compatibility with the code.
- virtual ~CsvParser ()
  Destructor: close the input file.
- RowVectorXr importRow (const Index &)
  Method to import a row from the input file.
- MatrixXr importRows (const std::initializer_list< Index > &)
  Method to import multiple rows from the input file.
- MatrixXr importFirstRows (const Index &)
  Method to import the first nRows rows from the input file.
- VectorXr importCol (const Index &)
  Method to import a column from the input file.
- MatrixXr importCols (const std::initializer_list< Index > &)
  Method to import multiple columns from the input file.
- MatrixXr importFirstCols (const Index &)
  Method to import the first nCols columns from the input file.
- Real importCell (const Index &, const Index &)
  Method to import a single cell from the input file.
- MatrixXr importAll ()
  Method to import the whole input file.

Getter methods

- const Index & **nRows** () const
- const Index & **nCols** () const

Private Member Functions

- void reset ()

  Reset all the flags for input_ and go back to the beginning of file (possibly by ignoring headers).

Private Attributes

- bool hasHeaders_

  bool to determine if first row contains headers or not.
- Index nRows_

  Number of rows in the input file.
- Index nCols_

  Number of columns in the input file.
- std::ifstream input_

  Input stream to input_filename.
- std::string line_

  Auxiliary variable to store currently processed line.
- char separator_

  The separator character detected.

7.4.1 Detailed Description

Class providing methods to read **numeric** content from a .csv file and to store it in Eigen matrices or vectors.

Definition at line 32 of file csvParser.h.

7.4.2 Constructor & Destructor Documentation

7.4.2.1 CsvParser (const std::string & *input_filename,* const bool & *hasHeaders* = `true`)

Constructor: load the input file and check its compatibility with the code.

Parameters

`in`	*input_filename*	: the name of the input file;
`in`	*hasHeaders*	: bool to specify if first row contains headers or not; if **true**, first row is always ignored.

Definition at line 22 of file csvParser.cc.

7.4.3 Member Function Documentation

7.4.3.1 RowVectorXr importRow (const Index & *index*)

Method to import a row from the input file.

Parameters

`in`	*index*	: the row index.

Returns

a row vector containing the content read.

Definition at line 94 of file csvParser.cc.

7.4.3.2 MatrixXr importRows (const std::initializer_list< Index > & *indexes*)

Method to import multiple rows from the input file.

Parameters

in	*indexes*	: initializer list containing the row indexes (e.g. something like {1, 3, 4}).

Returns

a matrix containing the content read (row by row).

Definition at line 127 of file csvParser.cc.

7.4.3.3 MatrixXr importFirstRows (const Index & *nRows*)

Method to import the first *nRows* rows from the input file.

Parameters

in	*nRows*	: the number of rows to import.

Returns

a matrix containing the content read (row by row).

Definition at line 144 of file csvParser.cc.

7.4.3.4 VectorXr importCol (const Index & *index*)

Method to import a column from the input file.

Parameters

in	*index*	: the column index.

Returns

a column vector containing the content read.

Definition at line 158 of file csvParser.cc.

7.4.3.5 MatrixXr importCols (const std::initializer_list< Index > & *indexes*)

Method to import multiple columns from the input file.

Parameters

in	*indexes*	: initializer list containing the column indexes (e.g. something like {1, 3, 4}).

Returns

a matrix containing the content read (column by column).

Definition at line 187 of file csvParser.cc.

7.4.3.6 MatrixXr importFirstCols (const Index & *nCols*)

Method to import the first *nCols* columns from the input file.

Parameters

in	*nCols*	: the number of columns to import.

Returns

a matrix containing the content read (column by column).

Definition at line 204 of file csvParser.cc.

7.4.3.7 Real importCell (const Index & *rowIndex,* const Index & *colIndex*)

Method to import a single cell from the input file.

Parameters

in	*rowIndex*	: the cell row index.
in	*colIndex*	: the cell column index.

Returns

a scalar containing the value read.

Definition at line 218 of file csvParser.cc.

7.4.3.8 MatrixXr importAll ( )

Method to import the whole input file.

Returns

a matrix containing the content read (cell by cell).

Definition at line 226 of file csvParser.cc.

The documentation for this class was generated from the following files:

- /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/csvParser.h
- /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/csvParser.cc

7.5 DosModel Class Reference

Class providing methods to process a simulation to extract the Density of States starting from a parameter list.

`#include <dosModel.h>`

Public Member Functions

- DosModel ()

  Default constructor.
- DosModel (const ParamList &)

  Explicit conversion constructor.
- virtual ~DosModel ()=default

  Destructor (defaulted).
- const ParamList & params () const

Getter method.

- void simulate (const GetPot &, const std::string &, const std::string &, const std::string &, const std::string &) const

  Perform the simulation.

- void post_process (const GetPot &, const std::string &, std::ostream &, std::ostream &, const VectorXr &, const VectorXr &, const VectorXr &, const VectorXr &) const

  Perform post-processing.

- void gnuplot_commands (const std::string &, std::ostream &) const

  Defines commands to generate Gnuplot output files.

- void save_plot (const std::string &, const std::string &, const std::string &, const std::string &) const

  Save the Gnuplot output files.

Private Attributes

- bool initialized_

  bool to determine if DosModel param_ has been properly initialized.

- ParamList params_

  The parameter list.

7.5.1 Detailed Description

Class providing methods to process a simulation to extract the Density of States starting from a parameter list.

Definition at line 39 of file dosModel.h.

7.5.2 Constructor & Destructor Documentation

7.5.2.1 DosModel (const ParamList & *params*) [explicit]

Explicit conversion constructor.

Parameters

in	*params*	: a parameter list.

Definition at line 22 of file dosModel.cc.

7.5.3 Member Function Documentation

7.5.3.1 void simulate (const GetPot & *config*, const std::string & *input_experim*, const std::string & *output_directory*, const std::string & *output_plot_subdir*, const std::string & *output_filename*) const

Perform the simulation.

Parameters

in	*config*	: the GetPot configuration object;
in	*input_experim*	: the file containing experimental data;
in	*output_directory*	: directory where to store output files;
in	*output_plot_subdir*	: sub-directory where to store Gnuplot files;

in	*output_filename*	: prefix for the output filename.

Definition at line 25 of file dosModel.cc.

7.5.3.2 void post_process (const GetPot & *config*, const std::string & *input_experim*, std::ostream & *output_fitting*, std::ostream & *output_CV*, const VectorXr & *x_semic*, const VectorXr & *dens*, const VectorXr & *V_simulated*, const VectorXr & *C_simulated*) const

Perform post-processing.

Parameters

in	*config*	: the GetPot configuration object;
in	*input_experim*	: the file containing experimental data;
out	*output_fitting*	: output file containing infos about fitting experimental data;
out	*output_CV*	: output file containing infos about capacitance-voltage data;
in	*x_semic*	: the mesh corresponding to the semiconductor domain;
in	*dens*	: charge density;
in	*V_simulated*	: simulated voltage values;
in	*C_simulated*	: simulated capacitance values.

Definition at line 271 of file dosModel.cc.

7.5.3.3 void gnuplot_commands (const std::string & *output_CV_filename*, std::ostream & *os*) const

Defines commands to generate Gnuplot output files.

Parameters

in	*output_CV_filename*	: output CV filename;
out	*os*	: output stream.

Definition at line 364 of file dosModel.cc.

7.5.3.4 void save_plot (const std::string & *output_directory*, const std::string & *output_plot_subdir*, const std::string & *output_CV_filename*, const std::string & *output_filename*) const

Save the Gnuplot output files.

Parameters

in	*output_directory*	: directory where to store output files;
in	*output_plot_subdir*	: sub-directory where to store Gnuplot files;
in	*output_CV_filename*	: output CV filename;
in	*output_filename*	: prefix for the output filename.

Definition at line 393 of file dosModel.cc.

The documentation for this class was generated from the following files:

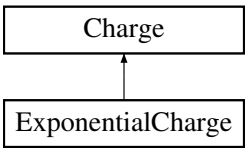
- /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/dosModel.h
- /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/dosModel.cc

7.6 ExponentialCharge Class Reference

Class derived from Charge, under the hypothesis that Density of States is a single exponential.

`#include <charge.h>`

Inheritance diagram for ExponentialCharge:



Public Member Functions

- ExponentialCharge ()=delete

  Default constructor (deleted since it is required to specify a ParamList and a QuadratureRule).
- ExponentialCharge (const ParamList &, const QuadratureRule &)

  Constructor.
- virtual ~ExponentialCharge ()=default

  Destructor (defaulted).
- virtual VectorXr charge (const VectorXr &) override

  Compute the total charge.
- virtual VectorXr dcharge (const VectorXr &) override

  Compute the derivative of the total charge with respect to the electric potential.

Private Member Functions

- Real n_approx (const Real &, const Real &, const Real &) const

  Compute electrons density (per unit volume).
- Real dn_approx (const Real &, const Real &, const Real &) const

  Compute the approximate derivative of electrons density (per unit volume) with respect to the electric potential.

Private Attributes

- Real N0_

  Parameter of the exponential density.

Additional Inherited Members

7.6.1 Detailed Description

Class derived from Charge, under the hypothesis that Density of States is a single exponential.

Provide methods to compute total electric charge and its derivative under the hypothesis that Density of States is a single exponential, whose parameter is got by the constructor, of the form:

$$\frac{N_0}{\lambda} \exp\left(-\frac{(\cdot)}{\lambda}\right) .$$

Definition at line 125 of file charge.h.

7.6.2 Constructor & Destructor Documentation

7.6.2.1 ExponentialCharge (const ParamList & *params,* const QuadratureRule & *rule*)

Constructor.

Parameters

in	*params*	: the list of simulation parameters;
in	*rule*	: a quadrature rule.

Definition at line 110 of file charge.cc.

7.6.3 Member Function Documentation

7.6.3.1 VectorXr charge (const VectorXr & *phi*) [override], [virtual]

Compute the total charge.

Parameters

in	*phi*	: the electric potential φ.

Returns

the total charge $q\,[C]$.

Implements Charge.

Definition at line 141 of file charge.cc.

7.6.3.2 VectorXr dcharge (const VectorXr & *phi*) [override], [virtual]

Compute the derivative of the total charge with respect to the electric potential.

Parameters

in	*phi*	: the electric potential φ.

Returns

the derivative: $\frac{dq}{d\varphi}\left[C \cdot V^{-1}\right]$.

Implements Charge.

Definition at line 153 of file charge.cc.

7.6.3.3 Real n_approx (const Real & *phi*, const Real & *N0*, const Real & *lambda*) const [private]

Compute electrons density (per unit volume).

Parameters

in	*phi*	: the electric potential φ;
in	*N0*	: the exponential N_0;
in	*lambda*	: the exponential λ.

Returns

the electrons density $n(\varphi)\left[m^{-3}\right]$.

Definition at line 113 of file charge.cc.

7.6.3.4 Real dn_approx (const Real & *phi*, const Real & *N0*, const Real & *lambda*) const [private]

Compute the approximate derivative of electrons density (per unit volume) with respect to the electric potential.

Parameters

in	*phi*	: the electric potential φ;
in	*N0*	: the exponential N_0;
in	*lambda*	: the exponential λ.

Returns

the derivative: $\frac{\mathrm{d}n}{\mathrm{d}\varphi}\left[m^{-3}\cdot V^{-1}\right]$.

Definition at line 127 of file charge.cc.

The documentation for this class was generated from the following files:

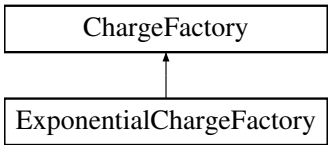
- /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/charge.h
- /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/charge.cc

7.7 ExponentialChargeFactory Class Reference

Concrete factory to handle a single exponential DoS constitutive relation.

`#include <factory.h>`

Inheritance diagram for ExponentialChargeFactory:



Public Member Functions

- ExponentialChargeFactory ()=default

  Default constructor (defaulted).
- virtual ~ExponentialChargeFactory ()=default

  Destructor (defaulted).
- virtual Charge * BuildCharge (const ParamList &, const QuadratureRule &) override

  Factory method to build a concrete Charge object.

7.7.1 Detailed Description

Concrete factory to handle a single exponential DoS constitutive relation.

Definition at line 82 of file factory.h.

7.7.2 Member Function Documentation

7.7.2.1 Charge * BuildCharge (const ParamList & *params*, const QuadratureRule & *rule*) [override], [virtual]

Factory method to build a concrete Charge object.

Parameters

in	*params*	: the list of simulation parameters;
in	*rule*	: a quadrature rule.

Returns

a pointer to ExponentialCharge.

Implements ChargeFactory.

Definition at line 20 of file factory.cc.

The documentation for this class was generated from the following files:

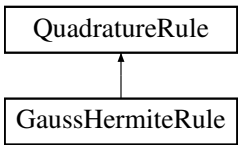
- /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/factory.h
- /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/factory.cc

7.8 GaussHermiteRule Class Reference

Class derived from QuadratureRule providing the Gauss-Hermite quadrature rule.

`#include <quadratureRule.h>`

Inheritance diagram for GaussHermiteRule:



Public Member Functions

- GaussHermiteRule ()=delete

  Default constructor (deleted since it is required to specify the number of nodes).
- GaussHermiteRule (const Index &)

  Constructor.
- virtual ~GaussHermiteRule ()=default

  Destructor (defaulted).
- virtual void apply () override

  Apply the quadrature rule in order to compute the nodes and weights.
- virtual void apply (const GetPot &) override

  Apply the quadrature rule reading parameters from a configuration file.
- void apply_iterative_algorithm (const Index &=1000, const Real &=1.0e-14)

  Compute nodes and weights using an adapted version of the algorithm presented in:
  William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. 2007.
  Numerical Recipes: The Art of Scientific Computing (3rd edition).
  Cambridge University Press, New York, NY, USA.
- void apply_using_eigendecomposition ()

  Compute nodes and weights using an eigendecomposition-based algorithm.

Additional Inherited Members

7.8.1 Detailed Description

Class derived from QuadratureRule providing the Gauss-Hermite quadrature rule.

Compute nodes and weights for the *nNodes_*-points approximation of:

$$\int_{-\infty}^{+\infty} w(x) f(x)\, \mathrm{d}x$$

where $w(x) = e^{-x^2}$.

Definition at line 92 of file quadratureRule.h.

7.8.2 Constructor & Destructor Documentation

7.8.2.1 GaussHermiteRule (const Index & *nNodes*)

Constructor.

Parameters

in	*nNodes*	: the number of nodes to be used for the quadrature rule.

Definition at line 26 of file quadratureRule.cc.

7.8.3 Member Function Documentation

7.8.3.1 void apply (const GetPot & *config*) [override], [virtual]

Apply the quadrature rule reading parameters from a configuration file.

Parameters

in	*config*	: the GetPot configuration object.

Implements QuadratureRule.

Definition at line 34 of file quadratureRule.cc.

The documentation for this class was generated from the following files:

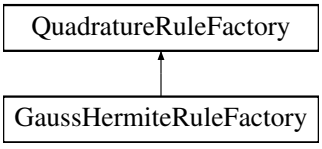
- /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/quadratureRule.h
- /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/quadratureRule.cc

7.9 GaussHermiteRuleFactory Class Reference

Concrete factory to handle a Gauss-Hermite quadrature rule.

`#include <factory.h>`

Inheritance diagram for GaussHermiteRuleFactory:



Public Member Functions

- GaussHermiteRuleFactory ()=default

  Default constructor (defaulted).
- virtual ~GaussHermiteRuleFactory ()=default

  Destructor (defaulted).
- virtual QuadratureRule * BuildRule (const Index &) override

  Factory method to build a concrete QuadratureRule object.

7.9.1 Detailed Description

Concrete factory to handle a Gauss-Hermite quadrature rule.

Definition at line 135 of file factory.h.

7.9.2 Member Function Documentation

7.9.2.1 QuadratureRule * BuildRule (const Index & *nNodes*) [override], [virtual]

Factory method to build a concrete QuadratureRule object.

Parameters

in	*nNodes*	: the number of nodes to be used for the quadrature rule.

Returns

a pointer to GaussHermiteRule.

Implements QuadratureRuleFactory.

Definition at line 25 of file factory.cc.

The documentation for this class was generated from the following files:

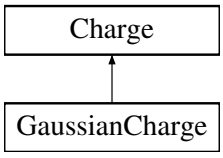
- /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/factory.h
- /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/factory.cc

7.10 GaussianCharge Class Reference

Class derived from Charge, under the hypothesis that Density of States is a combination of gaussians.

`#include <charge.h>`

Inheritance diagram for GaussianCharge:



Public Member Functions

- GaussianCharge ()=delete

Default constructor (deleted since it is required to specify a ParamList and a QuadratureRule).

- GaussianCharge (const ParamList &, const QuadratureRule &)

  Constructor.
- virtual ~GaussianCharge ()=default

  Destructor (defaulted).
- virtual VectorXr charge (const VectorXr &) override

  Compute the total charge.
- virtual VectorXr dcharge (const VectorXr &) override

  Compute the derivative of the total charge with respect to the electric potential.

Private Member Functions

- Real n_approx (const Real &, const Real &, const Real &) const

  Compute electrons density (per unit volume).
- Real dn_approx (const Real &, const Real &, const Real &) const

  Compute the approximate derivative of electrons density (per unit volume) with respect to the electric potential.

Additional Inherited Members

7.10.1 Detailed Description

Class derived from Charge, under the hypothesis that Density of States is a combination of gaussians.

Provide methods to compute total electric charge and its derivative under the hypothesis that Density of States is a linear combination of multiple gaussians, whose parameters are read from a ParamList object, of the form:

$$\frac{N_0}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(\cdot)^2}{2\sigma^2}\right) .$$

Definition at line 74 of file charge.h.

7.10.2 Constructor & Destructor Documentation

7.10.2.1 GaussianCharge (const ParamList & *params*, const QuadratureRule & *rule*)

Constructor.

Parameters

in	*params*	: the list of simulation parameters;
in	*rule*	: a quadrature rule.

Definition at line 20 of file charge.cc.

7.10.3 Member Function Documentation

7.10.3.1 VectorXr charge (const VectorXr & *phi*) [override], [virtual]

Compute the total charge.

Parameters

in	*phi*	: the electric potential φ.

Returns

the total charge $q\,[C]$.

Implements Charge.

Definition at line 51 of file charge.cc.

7.10.3.2 VectorXr dcharge (const VectorXr & *phi*) [override],[virtual]

Compute the derivative of the total charge with respect to the electric potential.

Parameters

in	*phi*	: the electric potential φ.

Returns

the derivative: $\frac{\mathrm{d}q}{\mathrm{d}\varphi}\,\left[C\cdot V^{-1}\right]$.

Implements Charge.

Definition at line 78 of file charge.cc.

7.10.3.3 Real n_approx (const Real & *phi,* const Real & *N0,* const Real & *sigma*) const [private]

Compute electrons density (per unit volume).

Parameters

in	*phi*	: the electric potential φ;
in	*N0*	: the gaussian mean N_0;
in	*sigma*	: the gaussian standard deviation σ.

Returns

the electrons density $n(\varphi)\,\left[m^{-3}\right]$.

Definition at line 23 of file charge.cc.

7.10.3.4 Real dn_approx (const Real & *phi,* const Real & *N0,* const Real & *sigma*) const [private]

Compute the approximate derivative of electrons density (per unit volume) with respect to the electric potential.

Parameters

in	*phi*	: the electric potential φ;
in	*N0*	: the gaussian mean N_0;
in	*sigma*	: the gaussian standard deviation σ.

Returns

the derivative: $\frac{\mathrm{d}n}{\mathrm{d}\varphi}\,\left[m^{-3}\cdot V^{-1}\right]$.

Definition at line 37 of file charge.cc.

The documentation for this class was generated from the following files:

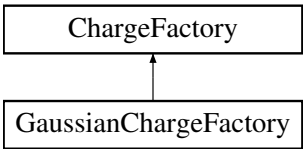
- /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/charge.h
- /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/charge.cc

7.11 GaussianChargeFactory Class Reference

Concrete factory to handle a multiple gaussians DoS constitutive relation.

`#include <factory.h>`

Inheritance diagram for GaussianChargeFactory:



Public Member Functions

- GaussianChargeFactory ()=default

  Default constructor (defaulted).
- virtual ~GaussianChargeFactory ()=default

  Destructor (defaulted).
- virtual Charge * BuildCharge (const ParamList &, const QuadratureRule &) override

  Factory method to build a concrete Charge object.

7.11.1 Detailed Description

Concrete factory to handle a multiple gaussians DoS constitutive relation.

Definition at line 55 of file factory.h.

7.11.2 Member Function Documentation

7.11.2.1 Charge * BuildCharge (const ParamList & *params*, const QuadratureRule & *rule*) [override], [virtual]

Factory method to build a concrete Charge object.

Parameters

in	*params*	: the list of simulation parameters;
in	*rule*	: a quadrature rule.

Returns

a pointer to GaussianCharge.

Implements ChargeFactory.

Definition at line 15 of file factory.cc.

The documentation for this class was generated from the following files:

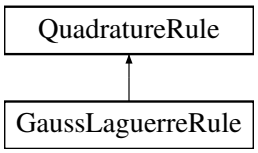
- /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/factory.h
- /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/factory.cc

7.12 GaussLaguerreRule Class Reference

Class derived from QuadratureRule providing the Gauss-Laguerre quadrature rule.

`#include <quadratureRule.h>`

Inheritance diagram for GaussLaguerreRule:



Public Member Functions

- GaussLaguerreRule ()=delete

  Default constructor (deleted since it is required to specify the number of nodes).
- GaussLaguerreRule (const Index &)

  Constructor.
- virtual ~GaussLaguerreRule ()=default

  Destructor (defaulted).
- virtual void apply () override

  Apply the quadrature rule in order to compute the nodes and weights.
- virtual void apply (const GetPot &) override

  Apply the quadrature rule reading parameters from a configuration file.
- void apply_iterative_algorithm (const Index &=1000, const Real &=1.0e-14)

  Compute nodes and weights using an adapted version of the algorithm presented in:
  William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. 2007.
  Numerical Recipes: The Art of Scientific Computing (3rd edition).
  Cambridge University Press, New York, NY, USA.
- void apply_using_eigendecomposition ()

  Compute nodes and weights using an eigendecomposition-based algorithm.

Static Public Member Functions

- static Real log_gamma (const Real &)

  Auxiliary function to compute $\log\Gamma(x)$.

Additional Inherited Members

7.12.1 Detailed Description

Class derived from QuadratureRule providing the Gauss-Laguerre quadrature rule.

Compute nodes and weights for the *nNodes_*-points approximation of:

$$\int_0^{+\infty} w(x) f(x)\, \mathrm{d}x$$

where $w(x) = e^{-x}$.

Definition at line 135 of file quadratureRule.h.

7.12.2 Constructor & Destructor Documentation

7.12.2.1 GaussLaguerreRule (const Index & *nNodes*)

Constructor.

Parameters

in	*nNodes*	: the number of nodes to be used for the quadrature rule.

Definition at line 146 of file quadratureRule.cc.

7.12.3 Member Function Documentation

7.12.3.1 Real log_gamma (const Real & *x*) [static]

Auxiliary function to compute $\log\Gamma(x)$.

Parameters

in	*x*	: the point to compute the function at.

Returns

the natural logarithm of the gamma function evaluated at *x*.

Definition at line 149 of file quadratureRule.cc.

7.12.3.2 void apply (const GetPot & *config*) [override], [virtual]

Apply the quadrature rule reading parameters from a configuration file.

Parameters

in	*config*	: the GetPot configuration object.

Implements QuadratureRule.

Definition at line 182 of file quadratureRule.cc.

The documentation for this class was generated from the following files:

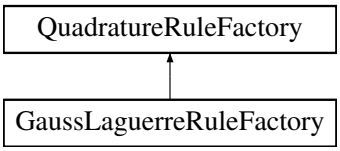
- /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/quadratureRule.h
- /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/quadratureRule.cc

7.13 GaussLaguerreRuleFactory Class Reference

Concrete factory to handle a Gauss-Laguerre quadrature rule.

`#include <factory.h>`

Inheritance diagram for GaussLaguerreRuleFactory:



Public Member Functions

- GaussLaguerreRuleFactory ()=default

  Default constructor (defaulted).

- virtual ~GaussLaguerreRuleFactory ()=default

  Destructor (defaulted).
- virtual QuadratureRule * BuildRule (const Index &) override

  Factory method to build a concrete QuadratureRule object.

7.13.1 Detailed Description

Concrete factory to handle a Gauss-Laguerre quadrature rule.

Definition at line 161 of file factory.h.

7.13.2 Member Function Documentation

7.13.2.1 QuadratureRule * BuildRule (const Index & *nNodes*) [override], [virtual]

Factory method to build a concrete QuadratureRule object.

Parameters

in	*nNodes*	: the number of nodes to be used for the quadrature rule.

Returns

a pointer to GaussLaguerreRule.

Implements QuadratureRuleFactory.

Definition at line 30 of file factory.cc.

The documentation for this class was generated from the following files:

- /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/factory.h
- /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/factory.cc

7.14 NonLinearPoisson1D Class Reference

Provide a solver for a non-linear Poisson equation.

`#include <solvers.h>`

Public Member Functions

- NonLinearPoisson1D ()=delete

  Default constructor (deleted since it is required to specify the solver to be used).
- NonLinearPoisson1D (const PdeSolver1D &, const Index &=100, const Real &=1.0e-6)

  Constructor.
- virtual ~NonLinearPoisson1D ()=default

  Destructor (defaulted).
- void apply (const VectorXr &, const VectorXr &, Charge &)

  Apply a Newton method to the equation and then discretize it using the solver specified.

Getter methods

- const VectorXr & **phi** () const
- const VectorXr & **norm** () const
- const Real & **qTot** () const
- const Real & **cTot** () const

Private Member Functions

- SparseXr computeJac (const VectorXr &) const

  Compute the Jacobi matrix.

Private Attributes

- const PdeSolver1D & solver_

  Solver handler.
- Index maxIterationsNo_

  Maximum number of iterations.
- Real tolerance_

  Tolerance.
- VectorXr phi_

  The electric potential.
- VectorXr norm_

  Vector holding L^∞-norm errors for each iteration.
- Real qTot_

  Total charge.
- Real cTot_

  Total capacitance.

7.14.1 Detailed Description

Provide a solver for a non-linear Poisson equation.

A Newton method is applied in order to solve:

$$-\frac{\mathrm{d}}{\mathrm{d}z}\left(\varepsilon(z)\cdot\frac{\mathrm{d}\varphi}{\mathrm{d}z}(z)\right)=-q\cdot\frac{N_0}{\sqrt{\pi}}\int_{-\infty}^{+\infty}\exp\left(-\alpha^2\right)\left(1+\exp\left(\frac{\sqrt{2}\sigma\alpha-q\varphi(z)}{K_B\cdot T}\right)\right)^{-1}\mathrm{d}\alpha\ .$$

Definition at line 192 of file solvers.h.

7.14.2 Constructor & Destructor Documentation

7.14.2.1 NonLinearPoisson1D (const PdeSolver1D & *solver*, const Index & *maxIterationsNo* = `100`, const Real & *tolerance* = `1.0e-6`)

Constructor.

Parameters

`in`	*solver*	: the solver to be used;
`in`	*maxIterationsNo*	: maximum number of iterations desired;
`in`	*tolerance*	: tolerance desired.

Definition at line 202 of file solvers.cc.

7.14.3 Member Function Documentation

7.14.3.1 void apply (const VectorXr & *mesh*, const VectorXr & *init_guess*, Charge & *charge_fun*)

Apply a Newton method to the equation and then discretize it using the solver specified.

Parameters

in	*mesh*	: the mesh;
in	*init_guess*	: initial guess for the Newton algorithm;
in	*charge_fun*	: an object of class Charge specifying how to compute total electric charge.

Definition at line 209 of file solvers.cc.

7.14.3.2 SparseXr computeJac (const VectorXr & *x*) const [private]

Compute the Jacobi matrix.

Parameters

in	*x*	: the vector where to start from.

Returns

the Jacobi matrix in a sparse format.

Definition at line 330 of file solvers.cc.

The documentation for this class was generated from the following files:

- /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/solvers.h
- /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/solvers.cc

7.15 ParamList Class Reference

Class providing methods to handle a list of parameters.

`#include <paramList.h>`

Public Member Functions

- ParamList ()=default

  Default constructor (defaulted).
- ParamList (const RowVectorXr &)

  Explicit conversion constructor.
- virtual ~ParamList ()=default

  Destructor (defaulted).

Getter methods

- const Index & **simulationNo** () const
- const Real & **t_semic** () const
- const Real & **t_ins** () const
- const Real & **eps_semic** () const
- const Real & **eps_ins** () const
- const Real & **Wf** () const
- const Real & **Ea** () const
- const Real & **N0** () const
- const Real & **sigma** () const
- const Real & **N0_2** () const
- const Real & **sigma_2** () const
- const Real & **shift_2** () const
- const Real & **N0_3** () const
- const Real & **sigma_3** () const

- const Real & **shift_3** () const
- const Real & **N0_4** () const
- const Real & **sigma_4** () const
- const Real & **shift_4** () const
- const Real & **N0_exp** () const
- const Real & **lambda_exp** () const
- const Index & **nNodes** () const
- const Index & **nSteps** () const
- const Real & **V_min** () const
- const Real & **V_max** () const

Private Attributes

- Index simulationNo_

  Simulation number index.
- Real t_semic_

  Semiconductor layer thickness $[m]$.
- Real t_ins_

  Insulator layer thickness $[m]$.
- Real eps_semic_

  Semiconductor layer relative electrical permittivity $[\,]$.
- Real eps_ins_

  Insulator layer relative electrical permittivity $[\,]$.
- Real Wf_

  Work-function $[V]$.
- Real Ea_

  Electron affinity $[V]$.
- Real N0_

  1st gaussian N_0 $\left[m^{-3}\right]$.
- Real sigma_

  1st gaussian standard deviation (normalized by $K_B \cdot T$*)* $[\,]$.
- Real N0_2_

  2nd gaussian N_0.
- Real sigma_2_

  2nd gaussian standard deviation.
- Real shift_2_

  2nd gaussian shift with respect to the 1st gaussian electric potential.
- Real N0_3_

  3rd gaussian N_0.
- Real sigma_3_

  3rd gaussian standard deviation.
- Real shift_3_

  3rd gaussian shift with respect to the 1st gaussian electric potential.
- Real N0_4_

  4th gaussian N_0.
- Real sigma_4_

  4th gaussian standard deviation.
- Real shift_4_

  4th gaussian shift with respect to the 1st gaussian electric potential.
- Real N0_exp_

  Exponential N_0.
- Real lambda_exp_

Exponential λ.

- Index nNodes_

  Number of nodes that form the mesh.

- Index nSteps_

  Number of steps to simulate.

- Real V_min_

  Minimum voltage $[V]$.

- Real V_max_

  Maximum voltage $[V]$.

Friends

- class **GaussianCharge**
- class **ExponentialCharge**
- class **DosModel**

7.15.1 Detailed Description

Class providing methods to handle a list of parameters.

It can include up to 4 gaussians (later combined to compute total charge) and an exponential.

Definition at line 28 of file paramList.h.

7.15.2 Constructor & Destructor Documentation

7.15.2.1 ParamList (const RowVectorXr & *list*) [explicit]

Explicit conversion constructor.

Parameters

in	*list*	: a row vector containing a list of parameters (for example got by a CsvParser object). Parameters should be sorted in the same order as specified above.

Definition at line 17 of file paramList.cc.

The documentation for this class was generated from the following files:

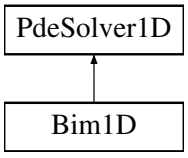
- /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/paramList.h
- /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/paramList.cc

7.16 PdeSolver1D Class Reference

Abstract class providing methods to assemble matrices to solve one-dimensional PDEs.

`#include <solvers.h>`

Inheritance diagram for PdeSolver1D:



Public Member Functions

- PdeSolver1D ()=delete

  Default constructor (deleted since it is required to specify the mesh).
- PdeSolver1D (VectorXr &)

  Constructor.
- virtual ~PdeSolver1D ()=default

  Destructor (defaulted).
- virtual void assembleAdvDiff (const VectorXr &alpha, const VectorXr &gamma, const VectorXr &eta, const VectorXr &beta)=0

  Assemble the matrix for an advection-diffusion term.
- virtual void assembleStiff (const VectorXr &eps, const VectorXr &kappa)=0

  Assemble the matrix for a diffusion term.
- virtual void assembleMass (const VectorXr &delta, const VectorXr &zeta)=0

  Assemble the matrix for a reaction term.

Getter methods

- const SparseXr & **AdvDiff** () const
- const SparseXr & **Stiff** () const
- const SparseXr & **Mass** () const

Protected Attributes

- VectorXr mesh_

  The mesh.
- Index nNodes_

  Number of nodes that form the mesh.
- SparseXr AdvDiff_

  Matrix for an advection-diffusion term.
- SparseXr Stiff_

  Stiffness matrix.
- SparseXr Mass_

  Mass matrix.

Friends

- class **NonLinearPoisson1D**

7.16.1 Detailed Description

Abstract class providing methods to assemble matrices to solve one-dimensional PDEs.

Matrices are held in a sparse format.

Definition at line 34 of file solvers.h.

7.16.2 Constructor & Destructor Documentation

7.16.2.1 PdeSolver1D (VectorXr & *mesh*)

Constructor.

Parameters

in	*mesh*	: the mesh.

Definition at line 15 of file solvers.cc.

7.16.3 Member Function Documentation

7.16.3.1 virtual void assembleAdvDiff (const VectorXr & *alpha,* const VectorXr & *gamma,* const VectorXr & *eta,* const VectorXr & *beta*) [pure virtual]

Assemble the matrix for an advection-diffusion term.

Build the matrix for the advection-diffusion problem: $-\nabla \cdot (\alpha \cdot \gamma(\eta \nabla u - \beta u)) = f$.

Parameters

in	*alpha*	: α, an element-wise constant function;
in	*gamma*	: γ, an element-wise linear function;
in	*eta*	: η, an element-wise linear function;
in	*beta*	: β, an element-wise constant function.

Implemented in Bim1D.

7.16.3.2 virtual void assembleStiff (const VectorXr & *eps,* const VectorXr & *kappa*) [pure virtual]

Assemble the matrix for a diffusion term.

Build the matrix for the diffusion problem: $-\nabla \cdot (\varepsilon \cdot \kappa \nabla u) = f$.

Parameters

in	*eps*	: ε, an element-wise constant function;
in	*kappa*	: κ, an element-wise linear function.

Implemented in Bim1D.

7.16.3.3 virtual void assembleMass (const VectorXr & *delta,* const VectorXr & *zeta*) [pure virtual]

Assemble the matrix for a reaction term.

Build the mass matrix for the reaction problem: $\delta \cdot \zeta u = f$.

Parameters

in	*delta*	: δ, an element-wise constant function;
in	*zeta*	: ζ, an element-wise linear function.

Implemented in Bim1D.

The documentation for this class was generated from the following files:

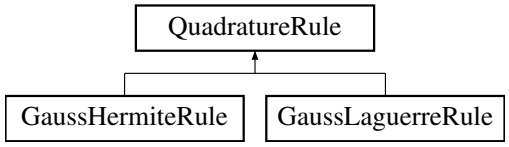
- /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/solvers.h
- /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/solvers.cc

7.17 QuadratureRule Class Reference

Abstract class providing a quadrature rule.

`#include <quadratureRule.h>`

Inheritance diagram for QuadratureRule:



Public Member Functions

- QuadratureRule ()=delete

  Default constructor (deleted since it is required to specify the number of nodes).
- QuadratureRule (const Index &)

  Constructor.
- virtual ~QuadratureRule ()=default

  Destructor (defaulted).
- virtual void apply ()=0

  Apply the quadrature rule in order to compute the nodes and weights.
- virtual void apply (const GetPot &config)=0

  Apply the quadrature rule reading parameters from a configuration file.

Getter methods

- const Index & **nNodes** () const
- const VectorXr & **nodes** () const
- const VectorXr & **weights** () const

Protected Attributes

- Index nNodes_

  Number of nodes of quadrature.
- VectorXr nodes_

  Vector containing the computed nodes coordinates.
- VectorXr weights_

  Vector containing the computed weights.

Friends

- class **GaussianCharge**
- class **ExponentialCharge**

7.17.1 Detailed Description

Abstract class providing a quadrature rule.

Approximate the integral:

$$\int_a^b f(x)\,\mathrm{d}x$$

with the finite sum:

$$\sum_{i=1}^{nNodes\_} w_i \cdot f(x_i)$$

where $\{x_i\}_{i=1}^{nNodes\_}$ and $\{w_i\}_{i=1}^{nNodes\_}$ are called respectively nodes and weights.

Definition at line 32 of file quadratureRule.h.

7.17.2 Constructor & Destructor Documentation

7.17.2.1 QuadratureRule (const Index & *nNodes*)

Constructor.

Parameters

in	*nNodes*	: the number of nodes to be used for the quadrature rule.

Definition at line 17 of file quadratureRule.cc.

7.17.3 Member Function Documentation

7.17.3.1 virtual void apply (const GetPot & *config*) [pure virtual]

Apply the quadrature rule reading parameters from a configuration file.

Parameters

in	*config*	: the GetPot configuration object.

Implemented in GaussLaguerreRule, and GaussHermiteRule.

The documentation for this class was generated from the following files:

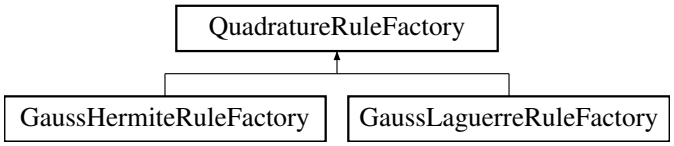
- /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/quadratureRule.h
- /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/quadratureRule.cc

7.18 QuadratureRuleFactory Class Reference

Abstract factory to handle a quadrature rule.

`#include <factory.h>`

Inheritance diagram for QuadratureRuleFactory:



Public Member Functions

- QuadratureRuleFactory ()=default

  Default constructor (defaulted).
- virtual ~QuadratureRuleFactory ()=default

  Destructor (defaulted).
- virtual QuadratureRule * BuildRule (const Index &nNodes)=0

  Factory method to build an abstract QuadratureRule object.

7.18.1 Detailed Description

Abstract factory to handle a quadrature rule.

Definition at line 109 of file factory.h.

7.18.2 Member Function Documentation

7.18.2.1 virtual QuadratureRule* BuildRule (const Index & *nNodes*) [pure virtual]

Factory method to build an abstract QuadratureRule object.

Parameters

in	*nNodes*	: the number of nodes to be used for the quadrature rule.

Returns

a pointer to QuadratureRule.

Implemented in GaussLaguerreRuleFactory, and GaussHermiteRuleFactory.

The documentation for this class was generated from the following file:

- /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/factory.h

Chapter 8

File Documentation

8.1 /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/charge.cc File Reference

```
#include "charge.h"
```

8.1.1 Detailed Description

Author

Pasquale Claudio Africa pasquale.africa@gmail.com

Date

2014

Copyright

Definition in file charge.cc.

8.2 /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/charge.h File Reference

Classes for computing total electric charge.

```
#include "paramList.h"
#include "quadratureRule.h"
#include "typedefs.h"
```

Classes

- class Charge

  Abstract class providing methods to calculate total electric charge (the rhs in the Poisson equation).
- class GaussianCharge

  Class derived from Charge, under the hypothesis that Density of States is a combination of gaussians.
- class ExponentialCharge

  Class derived from Charge, under the hypothesis that Density of States is a single exponential.

8.2.1 Detailed Description

Classes for computing total electric charge.

Author

Pasquale Claudio Africa pasquale.africa@gmail.com

Date

2014

Copyright

Definition in file charge.h.

8.3 /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/csvParser.cc File Reference

```
#include "csvParser.h"
```

8.3.1 Detailed Description

Author

Pasquale Claudio Africa pasquale.africa@gmail.com

Date

2014

Copyright

Definition in file csvParser.cc.

8.4 /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/csvParser.h File Reference

Tools to store content from a .csv file in matrices or vectors.

```
#include "typedefs.h"
#include <fstream>
#include <sstream>
#include <string>
#include <utility>
```

Classes

- class CsvParser

  *Class providing methods to read **numeric** content from a .csv file and to store it in Eigen matrices or vectors.*

8.4.1 Detailed Description

Tools to store content from a .csv file in matrices or vectors.

Author

Pasquale Claudio Africa pasquale.africa@gmail.com

Date

2014

Copyright

Definition in file csvParser.h.

8.5 /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/dosModel.cc File Reference

```
#include "dosModel.h"
```

8.5.1 Detailed Description

Author

Pasquale Claudio Africa pasquale.africa@gmail.com

Date

2014

Copyright

Definition in file dosModel.cc.

8.6 /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/dosModel.h File Reference

Mathematical model for Density of States extraction.

```
#include "charge.h"
#include "csvParser.h"
#include "factory.h"
#include "numerics.h"
#include "paramList.h"
#include "quadratureRule.h"
#include "solvers.h"
#include "typedefs.h"
#include "gnuplot-iostream.h"
#include <chrono>
#include <limits>
```

Classes

- class DosModel

  Class providing methods to process a simulation to extract the Density of States starting from a parameter list.

8.6.1 Detailed Description

Mathematical model for Density of States extraction.

Author

Pasquale Claudio Africa pasquale.africa@gmail.com

Date

2014

Copyright

Definition in file dosModel.h.

8.7 /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/factory.cc File Reference

```
#include "factory.h"
```

8.7.1 Detailed Description

Author

Pasquale Claudio Africa pasquale.africa@gmail.com

Date

2014

Copyright

Definition in file factory.cc.

8.8 /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/factory.h File Reference

Abstract factory design patterns.

```
#include "charge.h"
#include "paramList.h"
#include "quadratureRule.h"
```

Classes

- class ChargeFactory
  Abstract factory to handle the constitutive relation for the Density of States.
- class GaussianChargeFactory
  Concrete factory to handle a multiple gaussians DoS constitutive relation.
- class ExponentialChargeFactory
  Concrete factory to handle a single exponential DoS constitutive relation.
- class QuadratureRuleFactory
  Abstract factory to handle a quadrature rule.
- class GaussHermiteRuleFactory
  Concrete factory to handle a Gauss-Hermite quadrature rule.
- class GaussLaguerreRuleFactory
  Concrete factory to handle a Gauss-Laguerre quadrature rule.

8.8.1 Detailed Description

Abstract factory design patterns.

Author

Pasquale Claudio Africa pasquale.africa@gmail.com

Date

2014

Copyright

Definition in file factory.h.

8.9 /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/numerics.cc File Reference

```
#include "numerics.h"
```

8.9.1 Detailed Description

Author

Pasquale Claudio Africa pasquale.africa@gmail.com

Date

2014

Copyright

Definition in file numerics.cc.

8.10 /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/numerics.h File Reference

Generic numeric algorithms.

```
#include "typedefs.h"
#include <limits>
```

Namespaces

- numerics

  Namespace for generic numeric algorithms.

Functions

- template<typename ScalarType >
  VectorX< ScalarType > sort (const VectorX< ScalarType > &vector)

  Function to sort Eigen vectors.
- template<typename ScalarType >
  VectorXpair< ScalarType > sort_pair (const VectorX< ScalarType > &vector)

  Function to sort Eigen vectors, keeping track of indexes.
- Real trapz (const VectorXr &x, const VectorXr &y)

  Function to compute approximate integral of y with spacing increment specified by x, using trapezoidal rule.
- Real trapz (const VectorXr &y)

  Compute the approximate integral of y with unit spacing, using trapezoidal rule.
- VectorXr deriv (const VectorXr &, const VectorXr &)

  Compute the numeric derivative: $\frac{dy}{dx}$.
- Real interp1 (const VectorXr &, const VectorXr &, const Real &)

  Linear 1D interpolation. Interpolate y, defined at points x, at the point xNew.
- VectorXr interp1 (const VectorXr &, const VectorXr &, const VectorXr &)

  Linear 1D interpolation. Interpolate y, defined at points x, at the points xNew.
- Real error_L2 (const VectorXr &, const VectorXr &, const VectorXr &, const Real &)

  Compute the L^2*-norm error between simulated and interpolated values, using trapz.*

8.10.1 Detailed Description

Generic numeric algorithms.

Author

Pasquale Claudio Africa pasquale.africa@gmail.com

Date

2014

Copyright

Definition in file numerics.h.

8.11 /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/paramList.cc File Reference

```
#include "paramList.h"
```

8.11.1 Detailed Description

Author

Pasquale Claudio Africa pasquale.africa@gmail.com

Date

2014

Copyright

Definition in file paramList.cc.

8.12 /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/paramList.h File Reference

Interface to process a list of simulation parameters.

```
#include "typedefs.h"
```

Classes

- class ParamList

  Class providing methods to handle a list of parameters.

8.12.1 Detailed Description

Interface to process a list of simulation parameters.

Author

Pasquale Claudio Africa pasquale.africa@gmail.com

Date

2014

Copyright

Definition in file paramList.h.

8.13 /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/physicalConstants.h File Reference

Physical constants.

```
#include "typedefs.h"
```

Namespaces

- constants

  Numerical constants.

Variables

- const Real Q = 1.60217653000000e-19

  Electron charge $[C]$.
- const Real Q2 = Q ∗ Q

  Electron charge squared $\left[C^2\right]$.
- const Real K_B = 1.38065050000000e-23

  Boltzmann's constant $\left[J \cdot K^{-1}\right]$.
- const Real EPS0 = 8.854187817e-12

  Vacuum electrical permittivity $\left[C \cdot V^{-1} \cdot m^{-1}\right]$.
- const Real T = 300

  Reference temperature $[K]$.
- const Real V_TH = K_B ∗ T / Q

  Treshold voltage $[V]$.

8.13.1 Detailed Description

Physical constants.

Author

Pasquale Claudio Africa pasquale.africa@gmail.com

Date

2014

Copyright

Definition in file physicalConstants.h.

8.14 /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/quadratureRule.cc File Reference

```
#include "quadratureRule.h"
```

8.14.1 Detailed Description

Author

Pasquale Claudio Africa pasquale.africa@gmail.com

Date

2014

Copyright

Definition in file quadratureRule.cc.

8.15 /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/quadratureRule.h File Reference

Quadrature rules.

```
#include "typedefs.h"
```

Classes

- class QuadratureRule

  Abstract class providing a quadrature rule.
- class GaussHermiteRule

  Class derived from QuadratureRule providing the Gauss-Hermite quadrature rule.
- class GaussLaguerreRule

  Class derived from QuadratureRule providing the Gauss-Laguerre quadrature rule.

8.15.1 Detailed Description

Quadrature rules.

Author

Pasquale Claudio Africa pasquale.africa@gmail.com

Date

2014

Copyright

Definition in file quadratureRule.h.

8.16 /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/solvers.cc File Reference

```
#include "solvers.h"
```

8.16.1 Detailed Description

Author

Pasquale Claudio Africa pasquale.africa@gmail.com

Date

2014

Copyright

Definition in file solvers.cc.

8.17 /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/solvers.h File Reference

Generic solvers for PDEs.

```
#include "charge.h"
#include "typedefs.h"
#include <utility>
#include <limits>
```

Classes

- class PdeSolver1D

  Abstract class providing methods to assemble matrices to solve one-dimensional PDEs.
- class Bim1D

  Class derived from PdeSolver1D, providing a finite volume Box Integration Method (BIM) solver.
- class NonLinearPoisson1D

  Provide a solver for a non-linear Poisson equation.

8.17.1 Detailed Description

Generic solvers for PDEs.

Author

Pasquale Claudio Africa pasquale.africa@gmail.com

Date

2014

Copyright

Definition in file solvers.h.

8.18 /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/typedefs.cc File Reference

```
#include "typedefs.h"
```

8.18.1 Detailed Description

Author

Pasquale Claudio Africa pasquale.africa@gmail.com

Date

2014

Copyright

Definition in file typedefs.cc.

8.19 /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/typedefs.h File Reference

Typedefs and utility functions.

```
#include <Eigen/Dense>
#include <Eigen/Sparse>
#include "GetPot"
#include <iostream>
#include <fstream>
#include "physicalConstants.h"
```

Namespaces

- constants

  Numerical constants.
- utility

  Namespace for utilities and auxiliary functions.

Macros

- #define Real double

  Pre-processor macro for real numbers.
- #define Index ptrdiff_t

  Pre-processor macro for indexing variables.

Typedefs

- typedef Matrix< Real, Dynamic, Dynamic > MatrixXr

  Typedef for dense real-valued dynamic-sized matrices.
- typedef Matrix< Real, Dynamic, 1 > VectorXr

  Typedef for dense real-valued dynamic-sized column vectors.
- typedef Matrix< Real, 1, Dynamic > RowVectorXr

  Typedef for dense real-valued dynamic-sized row vectors.
- typedef SparseMatrix< Real > SparseXr

  Typedef for sparse real-valued dynamic-sized matrices.
- template<typename ScalarType >
  using VectorX = Matrix< ScalarType, Dynamic, 1 >

  Template alias for Eigen vectors.
- template<typename T >
  using VectorXpair = VectorX< std::pair< T, Index > >

  Template alias for an Eigen vector of pairs: (ScalarType, Index).

Functions

- std::string full_path (const std::string &, const std::string &)

  Auxiliary function to return the full path to a file.
- void print_block (const char *, std::ostream &=std::cout)

  Auxiliary function to print a string inside a block.
- void print_done (std::ostream &=std::cout)

  Auxiliary function to print a "DONE!" string.

Variables

- const Index PARAMS_NO = 24

  Number of parameters required in input file.
- const Real PI = M_PI

  π.
- const Real SQRT_PI = std::sqrt(PI)

  $\sqrt{\pi}$.
- const Real PI_M4 = 0.7511255444649425

  $\pi^{-\frac{1}{4}}$.
- const Real SQRT_2 = std::sqrt(2)

  $\sqrt{2}$.

8.19.1 Detailed Description

Typedefs and utility functions.

Author

Pasquale Claudio Africa pasquale.africa@gmail.com

Date

2014

Copyright

Definition in file typedefs.h.

8.19.2 Typedef Documentation

8.19.2.1 using VectorX = Matrix<ScalarType, Dynamic, 1>

Template alias for Eigen vectors.

Template Parameters

ScalarType	: the scalar type.

Definition at line 44 of file typedefs.h.

8.19.2.2 using VectorXpair = VectorX<std::pair<T, Index> >

Template alias for an Eigen vector of pairs: (*ScalarType*, *Index*).

Template Parameters

ScalarType	: the scalar type.

Definition at line 51 of file typedefs.h.

8.20 /home/elauksap/Dropbox/Progetto-PACS/C++/Source/test/simulate_dos.cc File Reference

A test file.

```
#include "src/dosModel.h"
```

Functions

- int main (const int argc, const char *const *argv, const char *const *envp)
  *The **main** function.*

8.20.1 Detailed Description

A test file.

Author

Pasquale Claudio Africa pasquale.africa@gmail.com

Date

2014

Copyright

Definition in file simulate_dos.cc.

Index