

DOS extraction

Generated by Doxygen 1.8.6

Wed Sep 24 2014 12:24:47

Contents

1	Index	1
1.1	Introduction	1
1.2	Dependancies	1
1.3	Compile	1
1.4	Set up the configurations	2
1.5	Run!	2
2	Namespace Index	3
2.1	Namespace List	3
3	Hierarchical Index	5
3.1	Class Hierarchy	5
4	Class Index	7
4.1	Class List	7
5	File Index	9
5.1	File List	9
6	Namespace Documentation	11
6.1	constants Namespace Reference	11
6.1.1	Detailed Description	11
6.2	numerics Namespace Reference	11
6.2.1	Detailed Description	12
6.2.2	Function Documentation	12
6.2.2.1	sort	12
6.2.2.2	sort_pair	12
6.2.2.3	trapz	13
6.2.2.4	trapz	13
6.2.2.5	deriv	13
6.2.2.6	interp1	13
6.2.2.7	interp1	14
6.2.2.8	error_L2	14

6.3	utility Namespace Reference	14
6.3.1	Detailed Description	15
6.3.2	Function Documentation	15
6.3.2.1	full_path	15
6.3.2.2	print_block	15
6.3.2.3	print_done	15
7	Class Documentation	17
7.1	Bim1D Class Reference	17
7.1.1	Detailed Description	18
7.1.2	Constructor & Destructor Documentation	18
7.1.2.1	Bim1D	18
7.1.3	Member Function Documentation	18
7.1.3.1	log_mean	18
7.1.3.2	bernoulli	18
7.1.3.3	assembleAdvDiff	19
7.1.3.4	assembleStiff	19
7.1.3.5	assembleMass	19
7.2	Charge Class Reference	19
7.2.1	Detailed Description	20
7.2.2	Constructor & Destructor Documentation	20
7.2.2.1	Charge	20
7.2.3	Member Function Documentation	20
7.2.3.1	charge	20
7.2.3.2	dcharge	21
7.3	CsvParser Class Reference	21
7.3.1	Detailed Description	22
7.3.2	Constructor & Destructor Documentation	22
7.3.2.1	CsvParser	22
7.3.3	Member Function Documentation	23
7.3.3.1	importRow	23
7.3.3.2	importRows	24
7.3.3.3	importFirstRows	24
7.3.3.4	importCol	24
7.3.3.5	importCols	24
7.3.3.6	importFirstCols	25
7.3.3.7	importCell	25
7.3.3.8	importAll	25
7.4	DosModel Class Reference	25
7.4.1	Detailed Description	26

7.4.2	Constructor & Destructor Documentation	26
7.4.2.1	DosModel	26
7.4.3	Member Function Documentation	26
7.4.3.1	simulate	26
7.4.3.2	post_process	27
7.4.3.3	gnuplot_commands	27
7.4.3.4	save_plot	27
7.5	GaussHermiteRule Class Reference	28
7.5.1	Detailed Description	28
7.5.2	Constructor & Destructor Documentation	29
7.5.2.1	GaussHermiteRule	29
7.5.3	Member Function Documentation	30
7.5.3.1	apply	30
7.6	GaussianCharge Class Reference	30
7.6.1	Detailed Description	31
7.6.2	Constructor & Destructor Documentation	31
7.6.2.1	GaussianCharge	31
7.6.3	Member Function Documentation	31
7.6.3.1	charge	31
7.6.3.2	dcharge	31
7.6.3.3	n_approx	31
7.6.3.4	dn_approx	32
7.7	NonLinearPoisson1D Class Reference	32
7.7.1	Detailed Description	33
7.7.2	Constructor & Destructor Documentation	33
7.7.2.1	NonLinearPoisson1D	33
7.7.3	Member Function Documentation	33
7.7.3.1	apply	33
7.7.3.2	computeJac	34
7.8	ParamList Class Reference	34
7.8.1	Detailed Description	36
7.8.2	Constructor & Destructor Documentation	36
7.8.2.1	ParamList	36
7.9	PdeSolver1D Class Reference	36
7.9.1	Detailed Description	37
7.9.2	Constructor & Destructor Documentation	37
7.9.2.1	PdeSolver1D	37
7.9.3	Member Function Documentation	37
7.9.3.1	assembleAdvDiff	37
7.9.3.2	assembleStiff	38

7.9.3.3	assembleMass	38
7.10	QuadratureRule Class Reference	38
7.10.1	Detailed Description	39
7.10.2	Constructor & Destructor Documentation	39
7.10.2.1	QuadratureRule	39
8	File Documentation	41
8.1	/home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/charge.h File Reference	41
8.1.1	Detailed Description	41
8.2	/home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/csvParser.h File Reference	41
8.2.1	Detailed Description	42
8.3	/home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/dosModel.h File Reference	42
8.3.1	Detailed Description	42
8.4	/home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/numerics.h File Reference	43
8.4.1	Detailed Description	43
8.5	/home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/paramList.h File Reference	43
8.5.1	Detailed Description	44
8.6	/home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/physicalConstants.h File Reference	44
8.6.1	Detailed Description	45
8.7	/home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/quadratureRule.h File Reference	45
8.7.1	Detailed Description	45
8.8	/home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/solvers.h File Reference	45
8.8.1	Detailed Description	46
8.9	/home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/typedefs.h File Reference	46
8.9.1	Detailed Description	47
8.9.2	Typedef Documentation	48
8.9.2.1	VectorX	48
8.9.2.2	VectorXpair	48
8.10	/home/elauksap/Dropbox/Progetto-PACS/C++/Source/test/simulate_dos.c++ File Reference	48
8.10.1	Detailed Description	48
	Index	49

Chapter 1

Index

1.1 Introduction

This program allows to extract the Density of States, 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 libraries to be installed on your system:

- **Eigen**, to handle with matrices, vectors and linear algebra;
- **Gnuplot**, a graphical utility to generate plots;
- **Boost**, a C++ library used by the **Gnuplot** interface to C++;
- **OpenMP**, for parallel computing (recommended but not compulsory).

It also uses the following packages, provided in the *include/* folder:

- **GetPot**, to parse command-line and configuration files;
- **gnuplot-iostream**, the C++ interface for **Gnuplot**.

1.3 Compile

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

```
$ make
```

or, if you want the compiler to produce also debugging informations:

```
$ make debug
```

You can specify the name of the test to be compiled (without extension, for example: *simulate_dos*) by passing the variable **NAME** through command-line:

```
$ make NAME=test_filename
```

The compiler will generate the *test_filename* executable under the *bin/* directory.

Repeat these instructions for each test you want to compile.

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).

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.

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, you can find the results of the simulation(s) in the output directory specified in the configuration file (default: *output/*) under *bin/*.

Chapter 2

Namespace Index

2.1 Namespace List

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

constants	Numerical constants	11
numerics	Namespace for generic numeric algorithms	11
utility	Namespace for utilities and auxiliary functions	14

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

Charge	19
GaussianCharge	30
CsvParser	21
DosModel	25
NonLinearPoisson1D	32
ParamList	34
PdeSolver1D	36
Bim1D	17
QuadratureRule	38
GaussHermiteRule	28

Chapter 4

Class Index

4.1 Class List

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

Bim1D	Class derived from PdeSolver1D , providing a finite volume Box Integration Method (BIM) solver	17
Charge	Abstract class providing methods to calculate total electric charge (the rhs in the Poisson equation)	19
CsvParser	Class providing methods to read numeric content from a .csv file and to store it in Eigen matrices or vectors	21
DosModel	Class providing methods to process a simulation to extract the Density of States starting from a parameter list	25
GaussHermiteRule	Class derived from QuadratureRule providing the Gauss-Hermite quadrature rule	28
GaussianCharge	Class derived from Charge , under the hypothesis that Density of States is a combination of gaussians	30
NonLinearPoisson1D	Provide a solver for a non-linear Poisson equation	32
ParamList	Class providing methods to handle a list of parameters	34
PdeSolver1D	Abstract class providing methods to assemble matrices to solve one-dimensional PDEs	36
QuadratureRule	Abstract class providing a quadrature rule	38

Chapter 5

File Index

5.1 File List

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

/home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/ charge.c++	??
/home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/ charge.h	
Classes for computing total electric charge	41
/home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/ csvParser.c++	??
/home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/ csvParser.h	
Tools to store content from a .csv file in matrices or vectors	41
/home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/ dosModel.c++	??
/home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/ dosModel.h	
Mathematical model for Density of States extraction	42
/home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/ numerics.c++	??
/home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/ numerics.h	
Generic numeric algorithms	43
/home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/ paramList.c++	??
/home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/ paramList.h	
List of simulation parameters	43
/home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/ physicalConstants.h	
Physical constants	44
/home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/ quadratureRule.c++	??
/home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/ quadratureRule.h	
Quadrature rules	45
/home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/ solvers.c++	??
/home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/ solvers.h	
Generic solvers for PDEs	45
/home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/ typedefs.c++	??
/home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/ typedefs.h	
Typedefs and utility functions	46
/home/elauksap/Dropbox/Progetto-PACS/C++/Source/test/ simulate_dos.c++	
A test file	48

Chapter 6

Namespace Documentation

6.1 constants Namespace Reference

Numerical constants.

Variables

- const [Real Q](#) = 1.602176530000000e-19
Electron charge [C].
- const [Real Q2](#) = [Q](#) * [Q](#)
Electron charge squared [C²].
- const [Real K_B](#) = 1.380650500000000e-23
Boltzmann's constant [J · K⁻¹].
- const [Real EPS0](#) = 8.854187817e-12
Vacuum electrical permittivity [C · V⁻¹ · m⁻¹].
- const [Real T](#) = 300
Reference temperature [K].
- const [Real V_TH](#) = [K_B](#) * [T](#) / [Q](#)
Threshold voltage [V].
- const [Index PARAMS_NO](#) = 22
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

<i>ScalarType</i>	: the scalar type.
-------------------	--------------------

Parameters

<i>in</i>	<i>vector</i>	: the vector to be sorted.
-----------	---------------	----------------------------

Returns

the sorted vector.

Definition at line 98 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

<i>ScalarType</i>	: the scalar type.
-------------------	--------------------

Parameters

<i>in</i>	<i>vector</i>	: the vector to be sorted.
-----------	---------------	----------------------------

Returns

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

Definition at line 107 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

<i>in</i>	<i>x</i>	: the vector of the discrete domain;
<i>in</i>	<i>y</i>	: the vector of values to integrate.

Returns

the approximate integral value.

Definition at line 3 of file numerics.c++.

6.2.2.4 Real trapz (const VectorXr & y)

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

Parameters

<i>in</i>	<i>y</i>	: the vector of values to integrate.
-----------	----------	--------------------------------------

Returns

the approximate integral value.

Definition at line 18 of file numerics.c++.

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

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

Parameters

<i>in</i>	<i>y</i>	: the vector of values to differentiate;
<i>in</i>	<i>x</i>	: the vector of the discrete domain.

Returns

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

Definition at line 23 of file numerics.c++.

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	<i>y</i>	: the vector of values to interpolate;
in	<i>x</i>	: the vector of the discrete domain;
in	<i>xNew</i>	: the point to interpolate at.

Returns

a scalar containing the interpolated value.

Definition at line 42 of file numerics.c++.

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	<i>y</i>	: the vector of values to interpolate;
in	<i>x</i>	: the vector of the discrete domain;
in	<i>xNew</i>	: the vector of points to interpolate at.

Returns

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

Definition at line 61 of file numerics.c++.

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	<i>interp</i>	: the interpolated values;
in	<i>simulated</i>	: the simulated values;
in	<i>V</i>	: the vector of the electric potential;
in	<i>V_shift</i>	: shift to the electric potential.

Returns

the value of the L^2 -norm error.

Definition at line 75 of file numerics.c++.

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	<i>filename</i>	: the filename;
in	<i>relative_ - directory</i>	: 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 3 of file typedefs.c++.

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	<i>string</i>	: the string to print;
out	<i>os</i>	: output stream.

Definition at line 8 of file typedefs.c++.

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

Auxiliary function to print a "DONE!" string.

Parameters

out	<i>os</i>	: output stream.
-----	-----------	------------------

Definition at line 31 of file typedefs.c++.

Chapter 7

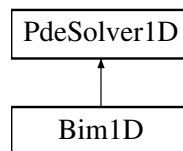
Class Documentation

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 111 of file solvers.h.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 Bim1D (VectorXr & mesh)

Constructor.

Parameters

<code>in</code>	<code>mesh</code>	: the mesh coordinates.
-----------------	-------------------	-------------------------

Definition at line 6 of file solvers.c++.

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

<code>in</code>	<code>x1</code>	: the first vector;
<code>in</code>	<code>x2</code>	: the second vector.

Returns

the vector of the logarithmic means.

Definition at line 9 of file solvers.c++.

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

<code>in</code>	<code>x</code>	: the vector of the values to compute the Bernoulli function at.
-----------------	----------------	--

Returns

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

Definition at line 32 of file solvers.c++.

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

Implements [PdeSolver1D](#).

Definition at line 79 of file solvers.c++.

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	<i>eps</i>	: ε , an element-wise constant function;
in	<i>kappa</i>	: κ , an element-wise linear function.

Implements [PdeSolver1D](#).

Definition at line 132 of file solvers.c++.

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	<i>delta</i>	: δ , an element-wise constant function;
in	<i>zeta</i>	: ζ , an element-wise linear function.

Implements [PdeSolver1D](#).

Definition at line 141 of file solvers.c++.

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

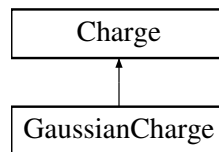
- /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/[solvers.h](#)
- /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/solvers.c++

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 25 of file `charge.h`.

7.2.2 Constructor & Destructor Documentation

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

Constructor.

Parameters

in	<i>params</i>	: the list of simulation parameters;
in	<i>rule</i>	: a quadrature rule.

Definition at line 5 of file `charge.c++`.

7.2.3 Member Function Documentation

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

Compute the total charge.

Parameters

<code>in</code>	<code>phi</code>	: the electric potential φ .
-----------------	------------------	--------------------------------------

Returns

the total charge q [C].

Implemented in [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

<code>in</code>	<code>phi</code>	: the electric potential φ .
-----------------	------------------	--------------------------------------

Returns

the derivative: $\frac{dq}{d\varphi}$ [C · V⁻¹].

Implemented in [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.c++](#)

7.3 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.

7.3.1 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 29 of file csvParser.h.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 CsvParser (const std::string & input_filename, const bool & hasHeaders = true)

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

Parameters

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

Definition at line 10 of file csvParser.c++.

7.3.3 Member Function Documentation

7.3.3.1 RowVectorXr importRow (const Index & *index*)

Method to import a row from the input file.

Parameters

<i>in</i>	<i>index</i>	: the row index.
-----------	--------------	------------------

Returns

a row vector containing the content read.

Definition at line 67 of file csvParser.c++.

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

Method to import multiple rows from the input file.

Parameters

<i>in</i>	<i>indexes</i>	: 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 96 of file csvParser.c++.

7.3.3.3 MatrixXr importFirstRows (const Index & nRows)

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

Parameters

<i>in</i>	<i>nRows</i>	: the number of rows to import.
-----------	--------------	---------------------------------

Returns

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

Definition at line 112 of file csvParser.c++.

7.3.3.4 VectorXr importCol (const Index & index)

Method to import a column from the input file.

Parameters

<i>in</i>	<i>index</i>	: the column index.
-----------	--------------	---------------------

Returns

a column vector containing the content read.

Definition at line 125 of file csvParser.c++.

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

Method to import multiple columns from the input file.

Parameters

<i>in</i>	<i>indexes</i>	: 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 150 of file csvParser.c++.

7.3.3.6 MatrixXr importFirstCols (const Index & nCols)

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

Parameters

<i>in</i>	<i>nCols</i>	: the number of columns to import.
-----------	--------------	------------------------------------

Returns

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

Definition at line 166 of file csvParser.c++.

7.3.3.7 Real importCell (const Index & rowIndex, const Index & colIndex)

Method to import a single cell from the input file.

Parameters

<i>in</i>	<i>rowIndex</i>	: the cell row index.
<i>in</i>	<i>colIndex</i>	: the cell column index.

Returns

a scalar containing the value read.

Definition at line 179 of file csvParser.c++.

7.3.3.8 MatrixXr importAll ()

Method to import the whole input file.

Returns

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

Definition at line 187 of file csvParser.c++.

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.c++](#)

7.4 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
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.

7.4.1 Detailed Description

Class providing methods to process a simulation to extract the Density of States starting from a parameter list.
Definition at line 35 of file `dosModel.h`.

7.4.2 Constructor & Destructor Documentation

7.4.2.1 [DosModel](#) (const [ParamList](#) & *params*) [`explicit`]

Explicit conversion constructor.

Parameters

<code>in</code>	<code>params</code>	: a parameter list.
-----------------	---------------------	---------------------

Definition at line 10 of file `dosModel.c++`.

7.4.3 Member Function Documentation

7.4.3.1 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	<i>config</i>	: the GetPot configuration object;
in	<i>input_experim</i>	: the file containing experimental data;
in	<i>output_directory</i>	: directory where to store output files;
in	<i>output_plot_ - subdir</i>	: sub-directory where to store Gnuplot files;
in	<i>output_filename</i>	: prefix for the output filename.

Definition at line 13 of file dosModel.c++.

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

Definition at line 177 of file dosModel.c++.

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

Defines commands to generate [Gnuplot](#) output files.

Parameters

in	<i>output_CV_ - filename</i>	: output CV filename;
out	<i>os</i>	: output stream.

Definition at line 262 of file dosModel.c++.

7.4.3.4 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	<i>output_directory</i>	: directory where to store output files;
in	<i>output_plot_ - subdir</i>	: sub-directory where to store Gnuplot files;
in	<i>output_CV_ - filename</i>	: output CV filename;

<code>in</code>	<code>output_filename</code>	: prefix for the output filename.
-----------------	------------------------------	-----------------------------------

Definition at line 291 of file dosModel.c++.

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

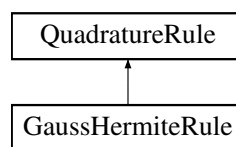
- /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/dosModel.h
- /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/dosModel.c++

7.5 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.
- void [apply](#) (const [GetPot](#) &)
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

7.5.1 Detailed Description

Class derived from [QuadratureRule](#) providing the Gauss-Hermite quadrature rule.

Compute nodes and weights for the $nNodes_$ -points approximation of

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

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

Definition at line 82 of file quadratureRule.h.

7.5.2 Constructor & Destructor Documentation

7.5.2.1 GaussHermiteRule (const Index & *nNodes*)

Constructor.

Parameters

<i>in</i>	<i>nNodes</i>	: the number of nodes to be used for the quadrature rule.
-----------	---------------	---

Definition at line 14 of file quadratureRule.c++.

7.5.3 Member Function Documentation

7.5.3.1 void apply (const GetPot & *config*)

Apply the quadrature rule reading parameters from a configuration file.

Parameters

<i>in</i>	<i>config</i>	: the GetPot configuration object.
-----------	---------------	------------------------------------

Definition at line 22 of file quadratureRule.c++.

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

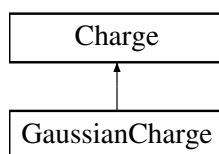
- /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/[quadratureRule.h](#)
- /home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/quadratureRule.c++

7.6 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

7.6.1 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.

Definition at line 70 of file charge.h.

7.6.2 Constructor & Destructor Documentation

7.6.2.1 GaussianCharge (const ParamList & params, const QuadratureRule & rule)

Constructor.

Parameters

in	<i>params</i>	: the list of simulation parameters;
in	<i>rule</i>	: a quadrature rule.

Definition at line 8 of file charge.c++.

7.6.3 Member Function Documentation

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

Compute the total charge.

Parameters

in	<i>phi</i>	: the electric potential φ .
----	------------	--------------------------------------

Returns

the total charge q [C].

Implements [Charge](#).

Definition at line 37 of file charge.c++.

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	<i>phi</i>	: the electric potential φ .
----	------------	--------------------------------------

Returns

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

Implements [Charge](#).

Definition at line 60 of file charge.c++.

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

Compute electrons density (per unit volume).

Parameters

in	<i>phi</i>	: the electric potential φ ;
in	<i>N0</i>	: the gaussian mean N_0 ;
in	<i>sigma</i>	: the gaussian standard deviation σ .

Returns

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

Definition at line 11 of file charge.c++.

7.6.3.4 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	<i>phi</i>	: the electric potential φ ;
in	<i>N0</i>	: the gaussian mean N_0 ;
in	<i>sigma</i>	: the gaussian standard deviation σ .

Returns

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

Definition at line 24 of file charge.c++.

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.c++

7.7 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.

7.7.1 Detailed Description

Provide a solver for a non-linear Poisson equation.

A Newton method is applied in order to solve:

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

Definition at line 190 of file solvers.h.

7.7.2 Constructor & Destructor Documentation

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

Constructor.

Parameters

in	<i>solver</i>	: the solver to be used;
in	<i>maxIterationsNo</i>	: maximum number of iterations desired;
in	<i>tolerance</i>	: tolerance desired.

Definition at line 162 of file solvers.c++.

7.7.3 Member Function Documentation**7.7.3.1 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	<i>mesh</i>	: the mesh;
in	<i>init_guess</i>	: initial guess for the Newton algorithm;
in	<i>charge_fun</i>	: an object of class Charge specifying how to compute total electric charge.

Definition at line 165 of file solvers.c++.

7.7.3.2 SparseXr computeJac (const VectorXr & x) const [private]

Compute the Jacobi matrix.

Parameters

in	<i>x</i>	: the vector where to start from.
----	----------	-----------------------------------

Returns

the Jacobi matrix in a sparse format.

Definition at line 273 of file solvers.c++.

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.c++

7.8 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 [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 mean [m⁻³].
- [Real](#) **sigma_**
1st gaussian standard deviation (normalized by $K_B \cdot T$) [].
- [Real](#) **N0_2_**
2nd gaussian mean.
- [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 mean.
- [Real](#) **sigma_3_**
3rd gaussian standard deviation.
- [Real](#) **shift_3_**
3rd gaussian shift with respect to the 1st gaussian electric potential.

- [Real NO_4_](#)
4th gaussian mean.
- [Real sigma_4_](#)
4th gaussian standard deviation.
- [Real shift_4_](#)
4th gaussian shift with respect to the 1st gaussian electric potential.
- [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 **DosModel**

7.8.1 Detailed Description

Class providing methods to handle a list of parameters.

It can include up to 4 gaussians, later combined to compute total charge.

Definition at line 25 of file paramList.h.

7.8.2 Constructor & Destructor Documentation

7.8.2.1 ParamList (const RowVectorXr & list) [explicit]

Explicit conversion constructor.

Parameters

<code>in</code>	<code>list</code>	: 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 5 of file paramList.c++.

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

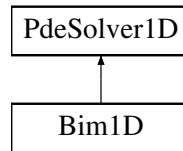
- `/home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/paramList.h`
- `/home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/paramList.c++`

7.9 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**

7.9.1 Detailed Description

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

Matrices are held in a sparse format.

Definition at line 31 of file solvers.h.

7.9.2 Constructor & Destructor Documentation

7.9.2.1 PdeSolver1D (VectorXr & mesh)

Constructor.

Parameters

<code>in</code>	<code>mesh</code>	: the mesh.
-----------------	-------------------	-------------

Definition at line 3 of file solvers.c++.

7.9.3 Member Function Documentation

7.9.3.1 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

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

Implemented in [Bim1D](#).

7.9.3.2 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: $-\nabla \cdot (\epsilon \cdot \kappa \nabla u) = f$.

Parameters

<code>in</code>	<code>eps</code>	: ϵ , an element-wise constant function;
<code>in</code>	<code>kappa</code>	: κ , an element-wise linear function.

Implemented in [Bim1D](#).

7.9.3.3 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

<code>in</code>	<code>delta</code>	: δ , an element-wise constant function;
<code>in</code>	<code>zeta</code>	: ζ , an element-wise linear function.

Implemented in [Bim1D](#).

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

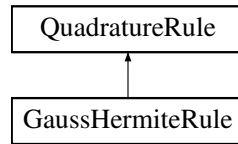
- [/home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/solvers.h](#)
- [/home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/solvers.c++](#)

7.10 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.

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**

7.10.1 Detailed Description

Abstract class providing a quadrature rule.

Approximate the integral:

$$\int_a^b f(x) \, dx$$

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 29 of file quadratureRule.h.

7.10.2 Constructor & Destructor Documentation

7.10.2.1 QuadratureRule (const Index & *nNodes*)

Constructor.

Parameters

<i>in</i>	<i>nNodes</i>	: the number of nodes to be used for the quadrature rule.
-----------	---------------	---

Definition at line 5 of file quadratureRule.c++.

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.c++

Chapter 8

File Documentation

8.1 /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.

8.1.1 Detailed Description

Classes for computing total electric charge.

Author

Pasquale Claudio Africa pasquale.africa@gmail.com

Date

2014

Definition in file [charge.h](#).

8.2 /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 <string>
#include <fstream>
#include <sstream>
#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.*

8.2.1 Detailed Description

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

Author

Pasquale Claudio Africa pasquale.africa@gmail.com

Date

2014

Definition in file [csvParser.h](#).

8.3 /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 "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.

8.3.1 Detailed Description

Mathematical model for Density of States extraction.

Author

Pasquale Claudio Africa pasquale.africa@gmail.com

Date

2014

Definition in file [dosModel.h](#).

8.4 /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.

8.4.1 Detailed Description

Generic numeric algorithms.

Author

Pasquale Claudio Africa pasquale.africa@gmail.com

Date

2014

Definition in file [numerics.h](#).

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

List of simulation parameters.

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

Classes

- class [ParamList](#)

Class providing methods to handle a list of parameters.

8.5.1 Detailed Description

List of simulation parameters.

Author

Pasquale Claudio Africa pasquale.africa@gmail.com

Date

2014

Definition in file [paramList.h](#).

8.6 /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 = 1.602176530000000e-19

Electron charge [C].

- const [Real](#) Q2 = Q * Q

Electron charge squared [C²].

- const [Real](#) K_B = 1.380650500000000e-23

Boltzmann's constant [J · K⁻¹].

- const [Real](#) EPS0 = 8.854187817e-12

Vacuum electrical permittivity [C · V⁻¹ · m⁻¹].

- const [Real](#) T = 300

Reference temperature [K].

- const [Real](#) V_TH = K_B * T / Q

Threshold voltage [V].

8.6.1 Detailed Description

Physical constants.

Author

Pasquale Claudio Africa pasquale.africa@gmail.com

Date

2014

Definition in file [physicalConstants.h](#).

8.7 /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.

8.7.1 Detailed Description

Quadrature rules.

Author

Pasquale Claudio Africa pasquale.africa@gmail.com

Date

2014

Definition in file [quadratureRule.h](#).

8.8 /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.

8.8.1 Detailed Description

Generic solvers for PDEs.

Author

Pasquale Claudio Africa pasquale.africa@gmail.com

Date

2014

Definition in file [solvers.h](#).

8.9 /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](#) = 22
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)
 $\sqrt{2}$.

8.9.1 Detailed Description

Typedefs and utility functions.

Author

Pasquale Claudio Africa pasquale.africa@gmail.com

Date

2014

Definition in file [typedefs.h](#).

8.9.2 Typedef Documentation

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

Template alias for [Eigen](#) vectors.

Template Parameters

<i>ScalarType</i>	: the scalar type.
-------------------	--------------------

Definition at line 41 of file typedefs.h.

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

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

Template Parameters

<i>ScalarType</i>	: the scalar type.
-------------------	--------------------

Definition at line 48 of file typedefs.h.

8.10 /home/elauksap/Dropbox/Progetto-PACS/C++/Source/test/simulate_dos.c++ 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.

8.10.1 Detailed Description

A test file.

Author

Pasquale Claudio Africa pasquale.africa@gmail.com

Date

2014

Definition in file [simulate_dos.c++](#).

Index

/home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/chargeCharge, 20
h, 41 GaussianCharge, 31
/home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/csmcomputeJac
Parser.h, 41 NonLinearPoisson1D, 34
/home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/dosconstants, 11
Model.h, 42 CsvParser, 21
/home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/numecCsvParser, 22
h, 43 CsvParser, 22
/home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/paramimportAll, 25
List.h, 43 importCell, 25
/home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/physicalimportCol, 24
Constants.h, 44 importCols, 24
/home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/quadratureimportFirstCols, 25
Rule.h, 45 importFirstRows, 24
/home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/solverimportRow, 23
h, 45 importRows, 24
/home/elauksap/Dropbox/Progetto-PACS/C++/Source/src/typedefs.-
h, 46 dcharge
/home/elauksap/Dropbox/Progetto-PACS/C++/Source/test/simulateCharge, 21
_dos.c++, 48 GaussianCharge, 31
deriv
numerics, 13
dn_approx
GaussianCharge, 32
DosModel, 25
DosModel, 26
DosModel, 26
gnuplot_commands, 27
post_process, 27
save_plot, 27
simulate, 26
error_L2
numerics, 14
full_path
utility, 15
GaussHermiteRule, 28
apply, 30
GaussHermiteRule, 29
GaussHermiteRule, 29
GaussianCharge, 30
charge, 31
dcharge, 31
dn_approx, 32
GaussianCharge, 31
GaussianCharge, 31
n_approx, 31
gnuplot_commands
apply
GaussHermiteRule, 30
NonLinearPoisson1D, 33
assembleAdvDiff
Bim1D, 18
PdeSolver1D, 37
assembleMass
Bim1D, 19
PdeSolver1D, 38
assembleStiff
Bim1D, 19
PdeSolver1D, 38
bernoulli
Bim1D, 18
Bim1D, 17
assembleAdvDiff, 18
assembleMass, 19
assembleStiff, 19
bernoulli, 18
Bim1D, 18
Bim1D, 18
log_mean, 18
Charge, 19
Charge, 20
charge, 20
dcharge, 21
charge

- DosModel, [27](#)
- importAll
 - CsvParser, [25](#)
- importCell
 - CsvParser, [25](#)
- importCol
 - CsvParser, [24](#)
- importCols
 - CsvParser, [24](#)
- importFirstCols
 - CsvParser, [25](#)
- importFirstRows
 - CsvParser, [24](#)
- importRow
 - CsvParser, [23](#)
- importRows
 - CsvParser, [24](#)
- interp1
 - numerics, [13](#), [14](#)
- log_mean
 - Bim1D, [18](#)
- n_approx
 - GaussianCharge, [31](#)
- NonLinearPoisson1D, [32](#)
 - apply, [33](#)
 - computeJac, [34](#)
 - NonLinearPoisson1D, [33](#)
 - NonLinearPoisson1D, [33](#)
- numerics, [11](#)
 - deriv, [13](#)
 - error_L2, [14](#)
 - interp1, [13](#), [14](#)
 - sort, [12](#)
 - sort_pair, [12](#)
 - trapz, [13](#)
- ParamList, [34](#)
 - ParamList, [36](#)
 - ParamList, [36](#)
- PdeSolver1D, [36](#)
 - assembleAdvDiff, [37](#)
 - assembleMass, [38](#)
 - assembleStiff, [38](#)
 - PdeSolver1D, [37](#)
 - PdeSolver1D, [37](#)
- post_process
 - DosModel, [27](#)
- print_block
 - utility, [15](#)
- print_done
 - utility, [15](#)
- QuadratureRule, [38](#)
 - QuadratureRule, [39](#)
 - QuadratureRule, [39](#)
- save_plot
 - DosModel, [27](#)
- simulate
 - DosModel, [26](#)
- sort
 - numerics, [12](#)
- sort_pair
 - numerics, [12](#)
- trapz
 - numerics, [13](#)
- typedefs.h
 - VectorX, [48](#)
 - VectorXpair, [48](#)
- utility, [14](#)
 - full_path, [15](#)
 - print_block, [15](#)
 - print_done, [15](#)
- VectorX
 - typedefs.h, [48](#)
- VectorXpair
 - typedefs.h, [48](#)