DOS extraction

Generated by Doxygen 1.8.6

Thu Sep 18 2014 10:47:11

Contents

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:

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 double Q = 1.60217653000000e-19
  Electron charge $[C]$.
- const double Q2 = Q $*$ Q
  Electron charge squared $\left[C^2\right]$.
- const double K_B = 1.38065050000000e-23
  Boltzmann's constant $\left[J \cdot K^{-1}\right]$.
- const double EPS0 = 8.854187817e-12
  Vacuum electrical permittivity $\left[C \cdot V^{-1} \cdot m^{-1}\right]$.
- const double T = 300
  Reference temperature $[K]$.
- const double V_TH = K_B $*$ T / Q
  Treshold voltage $[V]$.
- const unsigned PARAMS_NO = 22
  Number of parameters required in input file.
- const double PI = M_PI
  π.
- const double SQRT_PI = std::sqrt(PI)
  $\sqrt{\pi}$.
- const double PI_M4 = 0.7511255444649425
  $\pi^{-\frac{1}{4}}$.
- const double 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.
- double trapz (const VectorXd &x, const VectorXd &y)

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

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

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

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

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

  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 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

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 107 of file numerics.h.

6.2.2.3 double trapz (const VectorXd & *x*, const VectorXd & *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 3 of file numerics.c++.

6.2.2.4 double trapz (const VectorXd & *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 18 of file numerics.c++.

6.2.2.5 VectorXd deriv (const VectorXd & *y*, const VectorXd & *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 23 of file numerics.c++.

6.2.2.6 double interp1 (const VectorXd & *x*, const VectorXd & *y*, const double & *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 42 of file numerics.c++.

6.2.2.7 VectorXd interp1 (const VectorXd & *x*, const VectorXd & *y*, const VectorXd & *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 61 of file numerics.c++.

6.2.2.8 double error_L2 (const VectorXd & *interp*, const VectorXd & *simulated*, const VectorXd & *V*, const double & *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 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	*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 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	*string*	: the string to print;
out	*os*	: 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	*os*	: output stream.

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

Chapter 7

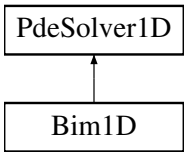
Class Documentation

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 (VectorXd &)

  Constructor.
- virtual ~Bim1D ()=default

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

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

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

  Assemble the matrix for a reaction term.

Static Public Member Functions

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

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

  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 (VectorXd & *mesh*)

Constructor.

Parameters

in	*mesh*	: the mesh coordinates.

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

7.1.3 Member Function Documentation

7.1.3.1 VectorXd log_mean (const VectorXd & *x1*, const VectorXd & *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 9 of file solvers.c++.

7.1.3.2 std::pair< VectorXd, VectorXd > bernoulli (const VectorXd & *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 32 of file solvers.c++.

7.1.3.3 void assembleAdvDiff (const VectorXd & *alpha,* const VectorXd & *gamma,* const VectorXd & *eta,* const VectorXd & *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 79 of file solvers.c++.

7.1.3.4 void assembleStiff (const VectorXd & *eps,* const VectorXd & *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 132 of file solvers.c++.

7.1.3.5 void assembleMass (const VectorXd & *delta,* const VectorXd & *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 141 of file solvers.c++.

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

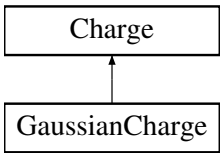
- /home/Data/Dropbox/Progetto-PACS/C++/Source/src/solvers.h
- /home/Data/Dropbox/Progetto-PACS/C++/Source/src/solvers.c++

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 VectorXd charge (const VectorXd &phi)=0

  Compute the total charge.
- virtual VectorXd dcharge (const VectorXd &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	*params*	: the list of simulation parameters;
in	*rule*	: a quadrature rule.

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

7.2.3 Member Function Documentation

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

Compute the total charge.

Parameters

in	*phi*	: the electric potential φ.

Returns

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

Implemented in GaussianCharge.

7.2.3.2 virtual VectorXd dcharge (const VectorXd & *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 GaussianCharge.

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

- /home/Data/Dropbox/Progetto-PACS/C++/Source/src/charge.h
- /home/Data/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.
- RowVectorXd importRow (const unsigned &)

  Method to import a row from the input file.
- MatrixXd importRows (const std::initializer_list< unsigned > &)

  Method to import multiple rows from the input file.
- MatrixXd importFirstRows (const unsigned &)

  Method to import the first nRows rows from the input file.
- VectorXd importCol (const unsigned &)

  Method to import a column from the input file.
- MatrixXd importCols (const std::initializer_list< unsigned > &)

  Method to import multiple columns from the input file.
- MatrixXd importFirstCols (const unsigned &)

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

- double importCell (const unsigned &, const unsigned &)

  Method to import a single cell from the input file.

- MatrixXd importAll ()

  Method to import the whole input file.

Getter methods

- const unsigned & **nRows** () const
- const unsigned & **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.

- unsigned nRows_

  Number of rows in the input file.

- unsigned 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`	*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 10 of file csvParser.c++.

7.3.3 Member Function Documentation

7.3.3.1 RowVectorXd importRow (const unsigned & *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 63 of file csvParser.c++.

7.3.3.2 MatrixXd importRows (const std::initializer_list< unsigned > & *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 92 of file csvParser.c++.

7.3.3.3 MatrixXd importFirstRows (const unsigned & *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 108 of file csvParser.c++.

7.3.3.4 VectorXd importCol (const unsigned & *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 121 of file csvParser.c++.

7.3.3.5 MatrixXd importCols (const std::initializer_list< unsigned > & *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 146 of file csvParser.c++.

7.3.3.6 MatrixXd importFirstCols (const unsigned & *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 162 of file csvParser.c++.

7.3.3.7 double importCell (const unsigned & *rowIndex*, const unsigned & *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 175 of file csvParser.c++.

7.3.3.8 MatrixXd importAll ( )

Method to import the whole input file.

Returns

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

Definition at line 183 of file csvParser.c++.

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

- /home/Data/Dropbox/Progetto-PACS/C++/Source/src/csvParser.h
- /home/Data/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 VectorXd &, const VectorXd &, const VectorXd &, const VectorXd &) 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

in	*params*	: 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	*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 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 VectorXd & *x_semic*, const VectorXd & *dens*, const VectorXd & *V_simulated*, const VectorXd & *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 172 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	*output_CV_filename*	: output CV filename;
out	*os*	: output stream.

Definition at line 257 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	*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 286 of file dosModel.c++.

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

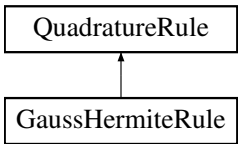
- /home/Data/Dropbox/Progetto-PACS/C++/Source/src/dosModel.h
- /home/Data/Dropbox/Progetto-PACS/C++/Source/src/dosModel.c++

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 unsigned &)

  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 unsigned &=1000, const double &=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

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

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 unsigned & *nNodes*)

Constructor.

Parameters

in	*nNodes*	: 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

in	*config*	: the GetPot configuration object.

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

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

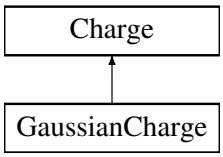
- /home/Data/Dropbox/Progetto-PACS/C++/Source/src/quadratureRule.h
- /home/Data/Dropbox/Progetto-PACS/C++/Source/src/quadratureRule.c++

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 VectorXd charge (const VectorXd &) override

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

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

Private Member Functions

- double n_approx (const double &, const double &, const double &) const

  Compute electrons density (per unit volume).
- double dn_approx (const double &, const double &, const double &) 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	*params*	: the list of simulation parameters;
in	*rule*	: a quadrature rule.

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

7.6.3 Member Function Documentation

7.6.3.1 VectorXd charge (const VectorXd & *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 37 of file charge.c++.

7.6.3.2 VectorXd dcharge (const VectorXd & *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 60 of file charge.c++.

7.6.3.3 double n_approx (const double & *phi*, const double & *N0*, const double & *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 11 of file charge.c++.

7.6.3.4 double dn_approx (const double & *phi*, const double & *N0*, const double & *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 24 of file charge.c++.

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

- /home/Data/Dropbox/Progetto-PACS/C++/Source/src/charge.h
- /home/Data/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 unsigned &=100, const double &=1.0e-6)

  Constructor.
- virtual ~NonLinearPoisson1D ()=default

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

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

Getter methods

- const VectorXd & **phi** () const
- const VectorXd & **norm** () const
- const double & **qTot** () const
- const double & **cTot** () const

Private Member Functions

- SparseXd computeJac (const VectorXd &) const

  Compute the Jacobi matrix.

Private Attributes

- const PdeSolver1D & solver_

  Solver handler.
- unsigned maxIterationsNo_

  Maximum number of iterations.
- double tolerance_

  Tolerance.
- VectorXd phi_

  The electric potential.
- VectorXd norm_

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

  Total charge.
- double 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:

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

Definition at line 190 of file solvers.h.

7.7.2 Constructor & Destructor Documentation

7.7.2.1 NonLinearPoisson1D (const PdeSolver1D & *solver*, const unsigned & *maxIterationsNo* = `100`, const double & *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 162 of file solvers.c++.

7.7.3 Member Function Documentation

7.7.3.1 void apply (const VectorXd & *mesh*, const VectorXd & *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 165 of file solvers.c++.

7.7.3.2 SparseXd computeJac (const VectorXd & *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 265 of file solvers.c++.

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

- /home/Data/Dropbox/Progetto-PACS/C++/Source/src/solvers.h
- /home/Data/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 RowVectorXd &)

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

  Destructor (defaulted).

Getter methods

- const unsigned & **simulationNo** () const
- const double & **t_semic** () const
- const double & **t_ins** () const
- const double & **eps_semic** () const
- const double & **eps_ins** () const
- const double & **Wf** () const
- const double & **Ea** () const
- const double & **N0** () const
- const double & **sigma** () const
- const double & **N0_2** () const
- const double & **sigma_2** () const
- const double & **shift_2** () const
- const double & **N0_3** () const
- const double & **sigma_3** () const
- const double & **shift_3** () const
- const double & **N0_4** () const
- const double & **sigma_4** () const
- const double & **shift_4** () const
- const unsigned & **nNodes** () const
- const unsigned & **nSteps** () const
- const double & **V_min** () const
- const double & **V_max** () const

Private Attributes

- unsigned simulationNo_

  Index of the simulation.
- double t_semic_

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

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

  Semiconductor layer relative electrical permittivity [ ].
- double eps_ins_

  Insulator layer relative electrical permittivity [ ].
- double Wf_

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

  Electron affinity $[V]$.
- double N0_

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

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

  2nd gaussian mean.
- double sigma_2_

  2nd gaussian standard deviation.
- double shift_2_

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

  3rd gaussian mean.
- double sigma_3_

  3rd gaussian standard deviation.
- double shift_3_

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

- double N0_4_

  4th gaussian mean.
- double sigma_4_

  4th gaussian standard deviation.
- double shift_4_

  4th gaussian shift with respect to the 1st gaussian electric potential.
- unsigned nNodes_

  Number of nodes that form the mesh.
- unsigned nSteps_

  Number of steps to simulate.
- double V_min_

  Minimum voltage $[V]$.
- double 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 RowVectorXd & *list*) [explicit]

Explicit conversion constructor.

Parameters

in	*list*	: a row vector containing a parameters list (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:

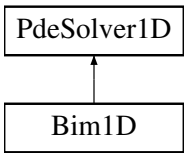
- /home/Data/Dropbox/Progetto-PACS/C++/Source/src/paramList.h
- /home/Data/Dropbox/Progetto-PACS/C++/Source/src/paramList.c++

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 (VectorXd &)

  Constructor.
- virtual ~PdeSolver1D ()=default

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

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

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

  Assemble the matrix for a reaction term.

Getter methods

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

Protected Attributes

- VectorXd mesh_

  The mesh.
- unsigned nNodes_

  Number of nodes that form the mesh.
- SparseXd AdvDiff_

  Matrix for an advection-diffusion term.
- SparseXd Stiff_

  Stiffness matrix.
- SparseXd 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 (VectorXd & *mesh*)

Constructor.

Parameters

in	*mesh*	: the mesh.

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

7.9.3 Member Function Documentation

7.9.3.1 virtual void assembleAdvDiff (const VectorXd & *alpha,* const VectorXd & *gamma,* const VectorXd & *eta,* const VectorXd & *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.

7.9.3.2 virtual void assembleStiff (const VectorXd & *eps,* const VectorXd & *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.

7.9.3.3 virtual void assembleMass (const VectorXd & *delta,* const VectorXd & *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:

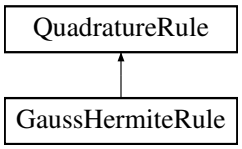
- /home/Data/Dropbox/Progetto-PACS/C++/Source/src/solvers.h
- /home/Data/Dropbox/Progetto-PACS/C++/Source/src/solvers.c++

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 unsigned &)

  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 unsigned & **nNodes** () const
- const VectorXd & **nodes** () const
- const VectorXd & **weights** () const

Protected Attributes

- unsigned nNodes_

  Number of nodes of the quadrature rule.
- VectorXd nodes_

  Vector containing the computed nodes coordinates.
- VectorXd weights_

  Vector containing the computed weights.

Friends

- class **GaussianCharge**

7.10.1 Detailed Description

Abstract class providing a quadrature rule.

Approximate the integral:

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

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 29 of file quadratureRule.h.

7.10.2 Constructor & Destructor Documentation

7.10.2.1 QuadratureRule (const unsigned & *nNodes*)

Constructor.

Parameters

in	*nNodes*	: 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/Data/Dropbox/Progetto-PACS/C++/Source/src/quadratureRule.h
- /home/Data/Dropbox/Progetto-PACS/C++/Source/src/quadratureRule.c++

Chapter 8

File Documentation

8.1 /home/Data/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/Data/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/Data/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/Data/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.
- double trapz (const VectorXd &x, const VectorXd &y)

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

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

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

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

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

  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/Data/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/Data/Dropbox/Progetto-PACS/C++/Source/src/physicalConstants.h File Reference

Physical constants.

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

Namespaces

- constants

  Numerical constants.

Variables

- const double Q = 1.60217653000000e-19

  Electron charge $[C]$.
- const double Q2 = Q * Q

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

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

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

  Reference temperature $[K]$.
- const double V_TH = K_B * T / Q

  Treshold 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/Data/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/Data/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/Data/Dropbox/Progetto-PACS/C++/Source/src/typedefs.h File Reference

Typedefs and utility functions.

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

Namespaces

- constants

  Numerical constants.
- utility

  Namespace for utilities and auxiliary functions.

Typedefs

- typedef SparseMatrix< double > SparseXd

  Typedef for sparse 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, unsigned > >

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

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 unsigned PARAMS_NO = 22

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

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

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

  $\pi^{-\frac{1}{4}}$.
- const double 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

ScalarType	: the scalar type.

Definition at line 34 of file typedefs.h.

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

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

Template Parameters

ScalarType	: the scalar type.

Definition at line 41 of file typedefs.h.

8.10 /home/Data/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