Variational Monte Carlo for trapped bosons

Generated by Doxygen 1.9.2

Chapter 1

Variational Monte Carlo for trapped bosons

Donwload the code from the main page of the repository by clicking the green button on top right https://github.com/Matteo294/FYS4411

Then unzip and open a terminal in the **Project1** folder

1.1 Run a simulation

\ → cd into code
```
cd code
```

and run the command
```
make
```

in the terminal.

At this point it is possible to choose to run the program in multiple ways

1. **Single run analytical** → runs a simple simulation using analytical expressions and with the pre-set parameters. The program provides an estimation for the ground state energy.
2. **Single run numerical** → runs a simple simulation as before but using a numerical approach. Much more time taking.
3. **Varying alpha** → runs multiple simulations varying the variational parameter alpha. The range and the number of alphas can be specified in **main.cpp**.
4. **Varying dt** → runs multiple simulations varying the time step-size dt used to integrate numerically the Langevin Equation. The range of **dt** together with the number of different simulations can be set in **main.cpp**
5. **Gradient descent** → performs a gradient descent to find the best variational parameter alpha
6. **One-body density function** → performs a simple simulation as in mode 1. whose purpose is to store data for the one-body density.

For further information please read the documentation provided in the file **documentation.pdf**

All the relevant parameters can be set directly in **main.cpp** before running the simulation. To run the simulation one can simply run the command $**$./main n$**$ in the terminal, where **n** must run from 0 to 6 to choose the running mode. For example to run run the simple simulation (mode 0)
```
./main 0
```

By default data saving is set to **False**. This means that data is not stored and it is not possible to carry any further analysis (e.g. Blocking variance analysis). This can be changed in the **main.cpp** file by setting the value of the constant **TO_FILE**.
In this case, before running the simulation, please make sure that the folders exists by running the **setup** command.
```
sh setup.sh
```

1.2 Variance analysis

The variance analysis of the results should be performed using the blocking method. All the instructions to run the script are written inside **blocking.py** ins the **Analysis** folder. In addition, once inside the **Analysis** folder, it is possible to display a help menu in the terminal by running the command

```
python blocking.py -h
```

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

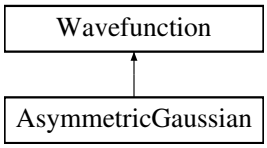
Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

Class Documentation

4.1 AsymmetricGaussian Class Reference

Inheritance diagram for AsymmetricGaussian:



Public Member Functions

- **AsymmetricGaussian** (class System *s, double alpha, double beta, double a)
- double evaluateAll ()
- double evaluateSing (int part_idx)
- double numericalSecondDerivative (int part_idx, int direction, double h)
- double psibar_psi ()
- vector< double > DriftForce (int part_idx)

Additional Inherited Members

4.1.1 Member Function Documentation

4.1.1.1 DriftForce()

```
vector< double > AsymmetricGaussian::DriftForce (
            int part_idx )  [virtual]
```

Evaluates the drift force associated to the part_idx-th particle

Implements Wavefunction.

4.1.1.2 evaluateAll()

```
double AsymmetricGaussian::evaluateAll ( )  [virtual]
```

Evaluates the wavefunction in the point in which the particle are in this moment

Implements Wavefunction.

4.1.1.3 evaluateSing()

```
double AsymmetricGaussian::evaluateSing (
            int part_idx )  [virtual]
```

Evaluates the terms of the wavefunctions that contain information relative to the part_idx-th particle.

Implements Wavefunction.

4.1.1.4 numericalSecondDerivative()

```
double AsymmetricGaussian::numericalSecondDerivative (
            int part_idx,
            int direction,
            double h )  [virtual]
```

Evaluates numerically the second derivative of the terms of the wavefunction containing information on part_idx-th particle. This is usefull when the analytic expression for the local energy is not known.

Implements Wavefunction.

4.1.1.5 psibar_psi()

```
double AsymmetricGaussian::psibar_psi ( )  [virtual]
```

Evaluates derivative of the function with respect to alpha, and divides by the wavefunction

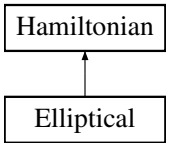
Implements Wavefunction.

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

- code/Wavefunctions/asymmetricGaussian.h
- code/Wavefunctions/asymmetricGaussian.cpp

4.2 Elliptical Class Reference

Inheritance diagram for Elliptical:



Public Member Functions

- **Elliptical** (class System *system, double omegaXY, double omegaZ)
- double LocalEnergyAnalytic ()
- double LocalEnergyNumeric (double h)

Additional Inherited Members

4.2.1 Member Function Documentation

4.2.1.1 LocalEnergyAnalytic()

```
double Elliptical::LocalEnergyAnalytic ( )  [virtual]
```

Evaluates the local energy of the system analytically.

Implements Hamiltonian.

4.2.1.2 LocalEnergyNumeric()

```
double Elliptical::LocalEnergyNumeric (
            double h )  [virtual]
```

Evaluates the local energy of the system analytically.

Implements Hamiltonian.

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

- code/Hamiltonians/elliptical.h
- code/Hamiltonians/elliptical.cpp

4.3 Functions Class Reference

Public Member Functions

- **Functions** (class System *system, bool parallel)
- vector< double > solve_singleRun ()
- vector< double > solve_singleRun (double h)
- vector< double > solve_singleRun (double r_max, int Nbins)
- vector< vector< double > > solve_varying_alpha (double alpha_min, double alpha_max, int Nalphas, bool alphatoFile)
- vector< vector< double > > solve_varying_dt (double dt_min, double dt_max, int Ndt, bool dttoFile)
- vector< vector< double > > solve_varying_N (vector< int > N, bool NtoFile)
- double gradientDescent (double initialAlpha, double gamma, double tolerance, int Nmax, int Nsteps)
- void printPresentation ()

  Print names of the authors of the code at the beginning of every execution.
- void printResultsSolver (vector< double > res)
- void printConfiguration (int selector, bool asymmetric, bool elliptical, bool importance)

  Print the configuration and the parameters of the system before launching the simulation.

Public Attributes

- class System * **system**

4.3.1 Member Function Documentation

4.3.1.1 gradientDescent()

```
double Functions::gradientDescent (
            double initialAlpha,
            double gamma,
            double tolerance,
            int Nmax,
            int Nsteps )
```

Finds the best alpha parameter using gradient descent approach starting from an initialAlpha. gamma specifies the learning rate tolerance specifies the condition on the derivative to stop the execution Nmax is the maximum number of iterations allowed before interrupting the execution Nsteps is the number of steps used in each Monte Carlo simulation.

See also

Solver::solve(true)

4.3.1.2 printResultsSolver()

```
void Functions::printResultsSolver (
            vector< double > res )
```

Useful function to print in a standard way the results given by the solver

See also

Solver::solve()

4.3.1.3 solve_singleRun() [1/3]

```
vector< double > Functions::solve_singleRun ( )
```

Runs a single Monte Carlo simulation using the analytic expression for the local energy.

See also

Solver::solve()

4.3.1.4 solve_singleRun() [2/3]

```
vector< double > Functions::solve_singleRun (
            double h )
```

Runs a single Monte Carlo simulation using numerical evaluation of the second derivative appearing in the local energy.

See also

Solver::solve(double h)

4.3.1.5 solve_singleRun() [3/3]

```
vector< double > Functions::solve_singleRun (
            double r_max,
            int Nbins )
```

Runs a single Monte Carlo simulation using the analytic expression for the local energy and producing the histogram for the radial one-body density.

See also

Solver::solve(double r_max, int Nbins)

4.3.1.6 solve_varying_alpha()

```
vector< vector< double > > Functions::solve_varying_alpha (
            double alpha_min,
            double alpha_max,
            int Nalphas,
            bool alphatoFile )
```

This function calls recursively the solver for Nalphas+1 different alpha values between alpha_min and alpha_max. Remember that the parameter alpha is the variational parameter of the wavefunction. Within the function the user can set the path where to generate the two .csv files containing respectively the counts for each bin and the alpha values.

Attention

The alpha values are printed to a .csv only if bool alphatofile=True.

Note

The .csv file with the counts for each bin is always generated and filled.

4.3.1.7 solve_varying_dt()

```
vector< vector< double > > Functions::solve_varying_dt (
            double dt_min,
            double dt_max,
            int Ndt,
            bool dttoFile )
```

This function calls recursively the solver (ImportanceSampling only!) for Ndt+1 different values of dt between dt_↩ min and dt_max. Remember that the parameter dt is the one used to integrate the Langevin's equation numerically. Within the function the user can set the path where to generate the .csv file containing the values of dt and also the .dat files with the local energy values saved at every step for each dt value.

Attention

The dt values are printed to a .csv only if bool dttofile=True.

Note

The .dat files with the local energy value at every step are generated only if the toFile flag in Solver is active.

See also

Solver

Attention

This function works only with the importance sampling solver.

See also

ImportanceSampling::ImportanceSampling()

4.3.1.8 solve_varying_N()

```
vector< vector< double > > Functions::solve_varying_N (
            vector< int > N,
            bool NtoFile )
```

This function progressively adds particles to the system for the values specified in the argument vector and calls recursively the solver. Within the function the user can set the path where to generate the .csv file containing the values of N and also the .dat files with the local energy values saved at every step for each N value.

Attention

The N values are printed to a .csv only if bool Ntofile=True.

Note

The .dat files with the local energy value at every step are generated only if the toFile flag in Solver is active.

See also

Solver

Attention

This function works only if the N values specified in the vector are increasing and if the minimum among them (the first in the vector) is smaller or equal to the number of particle that we set in the system at the moment of its creation.

See also

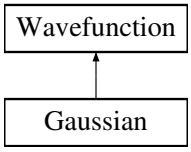
System::system()

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

- code/Others/functions.h
- code/Others/functions.cpp

4.4 Gaussian Class Reference

Inheritance diagram for Gaussian:



Public Member Functions

- **Gaussian** (class System *s, double alpha)
- double evaluateAll ()
- double evaluateSing (int part_idx)
- double numericalSecondDerivative (int part_idx, int direction, double h)
- double psibar_psi ()
- vector< double > DriftForce (int part_idx)

Additional Inherited Members

4.4.1 Member Function Documentation

4.4.1.1 DriftForce()

```
vector< double > Gaussian::DriftForce (
            int part_idx )  [virtual]
```

Evaluates the drift force associated to the part_idx-th particle

Implements Wavefunction.

4.4.1.2 evaluateAll()

```
double Gaussian::evaluateAll ( )  [virtual]
```

Evaluates the wavefunction in the point in which the particle are in this moment

Implements Wavefunction.

4.4.1.3 evaluateSing()

```
double Gaussian::evaluateSing (
            int part_idx )  [virtual]
```

Evaluates the terms of the wavefunctions that contain information relative to the part_idx-th particle.

Implements Wavefunction.

4.4.1.4 numericalSecondDerivative()

```
double Gaussian::numericalSecondDerivative (
            int part_idx,
            int direction,
            double h )  [virtual]
```

Evaluates numerically the second derivative of the terms of the wavefunction containing information on part_idx-th particle. This is usefull when the analytic expression for the local energy is not known.

Implements Wavefunction.

4.4.1.5 psibar_psi()

```
double Gaussian::psibar_psi ( )  [virtual]
```

Evaluates derivative of the function with respect to alpha, and divides by the wavefunction

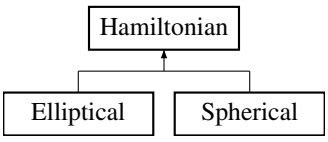
Implements Wavefunction.

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

- code/Wavefunctions/gaussian.h
- code/Wavefunctions/gaussian.cpp

4.5 Hamiltonian Class Reference

Inheritance diagram for Hamiltonian:



Public Member Functions

- **Hamiltonian** (class System *systemm, int nparams)
- void setParameter (int idx, double value)
- double getParameter (int idx)
- int getnparameter ()
- virtual double LocalEnergyAnalytic ()=0
- virtual double LocalEnergyNumeric (double h)=0

Public Attributes

- class System * **system**

Protected Attributes

- int **nparams**
- vector< double > **params**

4.5.1 Member Function Documentation

4.5.1.1 getnparameter()

```
int Hamiltonian::getnparameter ( )
```

Returns the number of parameters characterizing the selected Hamiltonian.

4.5.1.2 getParameter()

```
double Hamiltonian::getParameter (
            int idx )
```

Returns the idx-th parameter characterizing the chosen Hamiltonian.

Note

The meaning of each element of the vector depends on the chosen Hamiltonian.

See also

Gaussian

AsymmeGaussian

4.5.1.3 LocalEnergyAnalytic()

```
virtual double Hamiltonian::LocalEnergyAnalytic ( )  [pure virtual]
```

Evaluates the local energy of the system analytically.

Implemented in Elliptical, and Spherical.

4.5.1.4 LocalEnergyNumeric()

```
virtual double Hamiltonian::LocalEnergyNumeric (
            double h )  [pure virtual]
```

Evaluates the local energy of the system analytically.

Implemented in Elliptical, and Spherical.

4.5.1.5 setParameter()

```
void Hamiltonian::setParameter (
            int idx,
            double value )
```

Assigns the value of the idx-th element of the vector of parameters.

Note

The meaning of each element of the vector depends on the chosen Hamiltonian.

See also

Gaussian

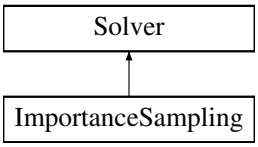
AsymmeGaussian

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

- code/Hamiltonians/hamiltonian.h
- code/Hamiltonians/hamiltonian.cpp

4.6 ImportanceSampling Class Reference

Inheritance diagram for ImportanceSampling:



Public Member Functions

- **ImportanceSampling** (class System *system, int Nsteps, double initialFraction, double dt, double D, bool tofile)
- vector< double > solve (bool allAverages)
- vector< double > solve (double r_max, int N_bins)
- vector< double > solve (double h)
- void thermalize ()

Additional Inherited Members

4.6.1 Member Function Documentation

4.6.1.1 solve() [1/3]

```
vector< double > ImportanceSampling::solve (
            bool allAverages )  [virtual]
```

Performs a MC simulation to evaluate the energy of the ground state of the system if allAverages = 1 --> evaluates the derivative of the mean value of the energy with respect to alpha (used in Functions::gradient_descent)

Implements Solver.

4.6.1.2 solve() [2/3]

```
vector< double > ImportanceSampling::solve (
            double h )  [virtual]
```

Performs a MC simulation to evaluate the energy of the ground state of the system This function overrides solve(bool allAverages) fro numerical evaluations. The parameter h is the steplength used to evaluat numerical derivatives

Implements Solver.

4.6.1.3 solve() [3/3]

```
vector< double > ImportanceSampling::solve (
            double r_max,
            int N_bins )  [virtual]
```

Performs a MC simulation to evaluate the energy of the system and the radial onebody density first argument specifies the r_max to sample and second argument is the number of bins to fill

Note

The .csv file with the counts for each bin is always generated. The path can be specified into Functions::solve_singleRun(double r_max, int N_bins)

Implements Solver.

4.6.1.4 thermalize()

```
void ImportanceSampling::thermalize ( )  [virtual]
```

Thermalizes the system. The number of thermalization steps can be set through the proper setter

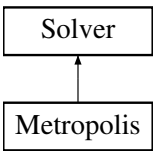
Implements Solver.

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

- code/Solvers/importanceSampling.h
- code/Solvers/importanceSampling.cpp

4.7 Metropolis Class Reference

Inheritance diagram for Metropolis:



Public Member Functions

- **Metropolis** (class System *system, int Nsteps, int NstepsThermal, double step, bool tofile)
- vector< double > solve (bool allAverages)
- vector< double > solve (double r_max, int N_bins)
- vector< double > solve (double h)
- void thermalize ()

Additional Inherited Members

4.7.1 Member Function Documentation

4.7.1.1 solve() [1/3]

```
vector< double > Metropolis::solve (
            bool allAverages )  [virtual]
```

Performs a MC simulation to evaluate the energy of the ground state of the system if allAverages = 1 --> evaluates the derivative of the mean value of the energy with respect to alpha (used in Functions::gradient_descent)

Implements Solver.

4.7.1.2 solve() [2/3]

```
vector< double > Metropolis::solve (
            double h )  [virtual]
```

Performs a MC simulation to evaluate the energy of the ground state of the system This function overrides solve(bool allAverages) fro numerical evaluations. The parameter h is the steplength used to evaluat numerical derivatives

Implements Solver.

4.7.1.3 solve() [3/3]

```
vector< double > Metropolis::solve (
            double r_max,
            int N_bins )  [virtual]
```

Performs a MC simulation to evaluate the energy of the system and the radial onebody density first argument specifies the r_max to sample and second argument is the number of bins to fill

Note

The .csv file with the counts for each bin is always generated. The path can be specified into Functions::solve_singleRun(double r_max, int N_bins)

Implements Solver.

4.7.1.4 thermalize()

```
void Metropolis::thermalize ( )  [virtual]
```

Thermalizes the system. The number of thermalization steps can be set through the proper setter

Implements Solver.

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

- code/Solvers/metropolis.h
- code/Solvers/metropolis.cpp

4.8 Particle Class Reference

Public Member Functions

- **Particle** (class System *system, double mass, vector< double > pos)
- void setMass (double m)

  Sets the mass of the particle.
- void setPosition (vector< double > new_pos)

  Sets the position of the particle to the selected vector.
- vector< double > getPosition ()

  Returns a vector describing the position of the particle.
- double getMass ()

  Returns the mass of the particle.
- void move (vector< double > delta_pos)

  This function varies the position of the particle of the vector delta_pos.

Public Attributes

- class System * **system**

4.8.1 Member Function Documentation

4.8.1.1 setPosition()

```
void Particle::setPosition (
            vector< double > new_pos )
```

Sets the position of the particle to the selected vector.

Set the position of a particle

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

- code/Particles/particle.h
- code/Particles/particle.cpp

4.9 RandomGenerator Class Reference

Public Attributes

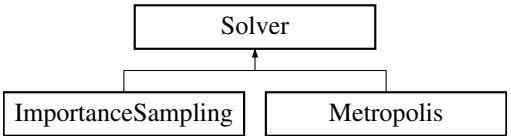
- uniform_real_distribution< double > **uniform**
- normal_distribution< double > **normal**

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

- code/Others/random_generator.h
- code/Others/random_generator.cpp

4.10 Solver Class Reference

Inheritance diagram for Solver:



Public Member Functions

- **Solver** (class System *system, int Nsteps, int NstepsThermal, int nparams, bool tofile)
- int getNsteps ()
- double getNstepsThermal ()
- double getParameter (int idx)
- int getnparameter ()
- bool getToFile ()
- void setNsteps (int Nsteps)
- void setNstepsThermal (int NstepsThermal)
- void setParameter (int idx, double value)
- void setPrintFile (string new_file)
- void setOneBodyFile (string new_file)
- void setToFile (bool tofile)
- virtual vector< double > solve (bool allAverages)=0
- virtual vector< double > solve (double h)=0
- virtual vector< double > solve (double r_max, int N_bins)=0
- virtual void thermalize ()=0

Public Attributes

- class System * system
- FILE * **energytofile**
- FILE * **onebodyFile**

Protected Attributes

- int **Nsteps**
- int **NstepsThermal**
- int **nparams**
- bool **tofile**
- vector< double > **params**

4.10.1 Member Function Documentation

4.10.1.1 getnparameter()

```
int Solver::getnparameter ( )
```

Returns the number of parameters

4.10.1.2 getNsteps()

```
int Solver::getNsteps ( )
```

Returns the number of MC steps

4.10.1.3 getNstepsThermal()

```
double Solver::getNstepsThermal ( )
```

Returns the number of steps used for thermalizing the system

4.10.1.4 getParameter()

```
double Solver::getParameter (
            int idx )
```

Returns the needed parameters for Importance sampling or Metropolis.

Note

Their actual meaning depends on the specific choice for the solver.

4.10.1.5 getToFile()

```
bool Solver::getToFile ( )
```

Returns True if printing the local energy at every step to file is enabled.

4.10.1.6 setNsteps()

```
void Solver::setNsteps (
            int Nsteps )
```

Sets the number of MC steps

4.10.1.7 setNstepsThermal()

```
void Solver::setNstepsThermal (
            int NstepsThermal )
```

Sets the number of steps for thermalizing the system

4.10.1.8 setOneBodyFile()

```
void Solver::setOneBodyFile (
            string new_file )
```

Sets the file path where print the local energy

4.10.1.9 setParameter()

```
void Solver::setParameter (
            int idx,
            double value )
```

Sets the needed parameters for Importance sampling or Metropolis

4.10.1.10 setPrintFile()

```
void Solver::setPrintFile (
            string new_file )
```

Sets the file path where to print the local energy

4.10.1.11 setToFile()

```
void Solver::setToFile (
            bool tofile )
```

Sets whether to print to file the local energy at every step.

4.10.1.12 solve() [1/3]

```
virtual vector<double> Solver::solve (
            bool allAverages )  [pure virtual]
```

Performs a MC simulation to evaluate the energy of the ground state of the system if allAverages = 1 --> evaluates the derivative of the mean value of the energy with respect to alpha (used in Functions::gradient_descent)

Implemented in ImportanceSampling, and Metropolis.

4.10.1.13 solve() [2/3]

```
virtual vector<double> Solver::solve (
            double h )  [pure virtual]
```

Performs a MC simulation to evaluate the energy of the ground state of the system This function overrides solve(bool allAverages) fro numerical evaluations. The parameter h is the steplength used to evaluat numerical derivatives

Implemented in ImportanceSampling, and Metropolis.

4.10.1.14 solve() [3/3]

```
virtual vector<double> Solver::solve (
            double r_max,
            int N_bins )  [pure virtual]
```

Performs a MC simulation to evaluate the energy of the system and the radial onebody density first argument specifies the r_max to sample and second argument is the number of bins to fill

Note

The .csv file with the counts for each bin is always generated. The path can be specified into Functions::solve_singleRun(double r_max, int N_bins)

Implemented in ImportanceSampling, and Metropolis.

4.10.1.15 thermalize()

```
virtual void Solver::thermalize ( )  [pure virtual]
```

Thermalizes the system. The number of thermalization steps can be set through the proper setter

Implemented in ImportanceSampling, and Metropolis.

4.10.2 Member Data Documentation

4.10.2.1 system

```
class System* Solver::system
```

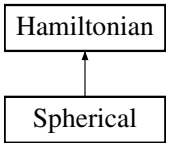
System pointer

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

- code/Solvers/solver.h
- code/Solvers/solver.cpp

4.11 Spherical Class Reference

Inheritance diagram for Spherical:



Public Member Functions

- **Spherical** (class System *system, double omega)
- double LocalEnergyAnalytic ()
- double LocalEnergyNumeric (double h)

Additional Inherited Members

4.11.1 Member Function Documentation

4.11.1.1 LocalEnergyAnalytic()

```
double Spherical::LocalEnergyAnalytic ( )  [virtual]
```

Evaluates the local energy of the system analytically.

Implements Hamiltonian.

4.11.1.2 LocalEnergyNumeric()

```
double Spherical::LocalEnergyNumeric (
            double h )  [virtual]
```

Evaluates the local energy of the system analytically.

Implements Hamiltonian.

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

- code/Hamiltonians/spherical.h
- code/Hamiltonians/spherical.cpp

4.12 System Class Reference

Public Member Functions

- System (int dim, int Npart, bool parallel)
- class Hamiltonian ∗ getHamiltonian ()
- class Wavefunction ∗ getWavefunction ()
- class Solver ∗ getSolver ()
- class RandomGenerator ∗ getRandomGenerator ()
- int getDimension ()
- int getNParticles ()
- bool getUseMatrix ()
- bool getParallel ()
- vector< class Particle ∗ > getParticles ()
- void setHamiltonian (class Hamiltonian ∗hamiltonian)
- void setSolver (class Solver ∗solver)
- void setWavefunction (class Wavefunction ∗wavefunction)
- void setRandomGenerator (class RandomGenerator ∗randomgenerator)
- void setUseMatrix (bool usematrix)
- void addParticle (double mass, vector< double > pos)
- double r2 (double parameter)
- double **r2** (vector< double > vect, double parameter)
- double **cdot** (vector< double > v1, vector< double > v2)
- void EvaluateRelativeDistance ()

  Updates the relative_distance matrix.
- void EvaluateRelativePosition ()

  Updates the relative position matrix.
- void EvaluateRelativePosition (int idx)

  Updates a special row of the relative position matrix.
- void EvaluateRelativeDistance (int idx)

  Updates a special row of the relative distance matrix.

Public Attributes

- vector< vector< vector< double > > > relative_position

  NxN matrix of 3d vectors. The ij-th element is a 3D vector containing posi - posj.
- vector< vector< double > > relative_distance

  NxN matrix of doubles. The ij-th element is the distance between particle i and particle j.

4.12.1 Constructor & Destructor Documentation

4.12.1.1 System()

```
System::System (
            int dim,
            int Npart,
            bool parallel )
```

Initializes the system with a certain dimensionality and a certain number of particles. If the simulations are run in parallel, bool parallel is true. Particles are initialized in the origin, they will be moved while calling the termalizer.

See also

Solver::thermalize()

4.12.2 Member Function Documentation

4.12.2.1 addParticle()

```
void System::addParticle (
            double mass,
            vector< double > pos )
```

Adds a particle to the system, placing it in the position specified by pos.

4.12.2.2 EvaluateRelativeDistance() [1/2]

```
void System::EvaluateRelativeDistance ( )
```

Updates the relative_distance matrix.

See also

relative_distance

4.12.2.3 EvaluateRelativeDistance() [2/2]

```
void System::EvaluateRelativeDistance (
            int idx )
```

Updates a special row of the relative distance matrix.

See also

relative_distance

4.12.2.4 EvaluateRelativePosition() [1/2]

```
void System::EvaluateRelativePosition ( )
```

Updates the relative position matrix.

See also

relative_position

4.12.2.5 EvaluateRelativePosition() [2/2]

```
void System::EvaluateRelativePosition (
            int idx )
```

Updates a special row of the relative position matrix.

See also

relative_position

4.12.2.6 getDimension()

```
int System::getDimension ( )
```

Returns the dimensionality of the system

4.12.2.7 getHamiltonian()

```
class Hamiltonian * System::getHamiltonian ( )
```

Returns the pointer to the selected Hamiltonian

4.12.2.8 getNParticles()

```
int System::getNParticles ( )
```

Returns the number of particles present in the system

4.12.2.9 getParallel()

```
bool System::getParallel ( )
```

Returns True if the simulation is run in parallel.

4.12.2.10 getParticles()

```
vector< class Particle * > System::getParticles ( )
```

Vector of pointers to Particles object

See also

class Particle)

4.12.2.11 getRandomGenerator()

```
class RandomGenerator * System::getRandomGenerator ( )
```

Returns the pointer to the random generator

4.12.2.12 getSolver()

```
class Solver * System::getSolver ( )
```

Returns the pointer to the selected Solver

4.12.2.13 getUseMatrix()

```
bool System::getUseMatrix ( )
```

Returns True if the evaluation of the matrices for relative distance and position is enabled for the current simulation.

4.12.2.14 getWavefunction()

```
class Wavefunction * System::getWavefunction ( )
```

Returns the pointer to the selected Wavefunction

4.12.2.15 r2()

```
double System::r2 (
            double parameter )
```

Evaluates the squared distance of particles and sums. The parameter can be set to give a different coefficient to the last squared coordinate of each particle (useful for asymmetric wavefunction and elliptical potential)

4.12.2.16 setHamiltonian()

```
void System::setHamiltonian (
            class Hamiltonian * hamiltonian )
```

Sets the Hamiltonian for the system,

See also

class Hamiltonian

4.12.2.17 setRandomGenerator()

```
void System::setRandomGenerator (
            class RandomGenerator * randomgenerator )
```

Sets the Random generator for the system,

See also

class RandomGenerator

4.12.2.18 setSolver()

```
void System::setSolver (
            class Solver * solver )
```

Sets the Solver for the system,

See also

class Solver

4.12.2.19 setUseMatrix()

```
void System::setUseMatrix (
            bool usematrix )
```

If set to True, imposes the update of the relative_positions and relative_distances matrices every time that a particle is moved This flag is automatically assigned when we set the wavefunction to the system and the choice is made depending on the number of paramerers characterizing the assigned wavefunction: Gaussian does not require the mentioned matrices and has 1 parameter, while AsymmGaussian gaussian requires it and has 2 parameters.

See also

System::setWavefunction()

4.12.2.20 setWavefunction()

```
void System::setWavefunction (
            class Wavefunction * wavefunction )
```

Sets the Wavefunvtion for the system,

See also

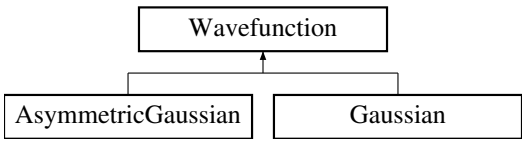
class Wavefunction

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

- code/System/system.h
- code/System/system.cpp

4.13 Wavefunction Class Reference

Inheritance diagram for Wavefunction:



Public Member Functions

- **Wavefunction** (class System *system, int nparams)
- virtual double evaluateAll ()=0
- virtual double evaluateSing (int part_idx)=0
- virtual double numericalSecondDerivative (int part_idx, int direction, double h)=0
- virtual vector< double > DriftForce (int part_idx)=0
- virtual double psibar_psi ()=0
- void setParameter (int idx, double value)
- double getParameter (int idx)
- int getNparams ()

Public Attributes

- class System * **s**

Protected Attributes

- int **nparams**
- vector< double > **params**

4.13.1 Member Function Documentation

4.13.1.1 DriftForce()

```
virtual vector<double> Wavefunction::DriftForce (
            int part_idx )  [pure virtual]
```

Evaluates the drift force associated to the part_idx-th particle

Implemented in AsymmetricGaussian, and Gaussian.

4.13.1.2 evaluateAll()

```
virtual double Wavefunction::evaluateAll ( )  [pure virtual]
```

Evaluates the wavefunction in the point in which the particle are in this moment

Implemented in AsymmetricGaussian, and Gaussian.

4.13.1.3 evaluateSing()

```
virtual double Wavefunction::evaluateSing (
            int part_idx )  [pure virtual]
```

Evaluates the terms of the wavefunctions that contain information relative to the part_idx-th particle.

Implemented in AsymmetricGaussian, and Gaussian.

4.13.1.4 getNparams()

```
int Wavefunction::getNparams ( )
```

Get number of parameters in the chosen wavefunction

4.13.1.5 getParameter()

```
double Wavefunction::getParameter (
            int idx )
```

Get wavefunction parameters in an array

4.13.1.6 numericalSecondDerivative()

```
virtual double Wavefunction::numericalSecondDerivative (
            int part_idx,
            int direction,
            double h )  [pure virtual]
```

Evaluates numerically the second derivative of the terms of the wavefunction containing information on part_idx-th particle. This is usefull when the analytic expression for the local energy is not known.

Implemented in AsymmetricGaussian, and Gaussian.

4.13.1.7 psibar_psi()

```
virtual double Wavefunction::psibar_psi ( )  [pure virtual]
```

Evaluates derivative of the function with respect to alpha, and divides by the wavefunction

Implemented in AsymmetricGaussian, and Gaussian.

4.13.1.8 setParameter()

```
void Wavefunction::setParameter (
            int idx,
            double value )
```

Set wavefunction parameters in an array

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

- code/Wavefunctions/wavefunction.h
- code/Wavefunctions/wavefunction.cpp

Index