LangevinSimulator

Generated by Doxygen 1.9.1

Chapter 1

Main Page

1.1 LangevinSimulator

LangevinSimulator is licensed, open-source software. LangevinSimulator comes with verbatim copies of the software "parameter_handler" as well as "TensorUtils", which is licensed, open-source software that is NOT part of LangevinSimulator. Please see sec. **Licenses** at the end of this document for details.

1.1.1 Cite us

Please cite our work in scientific publications. Citation entries as `RIS` and `BibTeX` are given in `MyParent↩Folder/LangevinSimulator/citation`. The software LangevinSimulator is based on the following articles:

- 2022: C. Widder, F. Glatzel & T. Schilling

  Generalized Langevin dynamics simulation with non-stationary memory kernels: How to make noise (`preprint`)

- 2020: H. Meyer, S. Wolf, G. Stock, T. Schilling

  A numerical procedure to evaluate memory effects in non-equilibrium coarse-grained models (`article`)

1.1.2 Introduction

LangevinSimulator offers methods to compute a data-driven coarse-grained model of Hamiltonian systems out of equilibrium. Formally, the non-stationary, generalized Langevin equation (nsGLE) is derived within a Mori-type projection operator formalism. The nsGLE is the equation of motion for the coarse-grained observable, which is a function of phase space. LangevinSimulator is capable of simulating new, self-consistent trajectories by integrating the equations of motion.

Based on a given set of original trajectories, LangevinSimulator allows to extract the so called memory kernel, that is needed to integrate the equations of motion. Further, the drift term and fluctuating forces are needed to simulate new trajectories. While the memory kernel and drift term are ensemble-averaged quantities, the fluctuating forces are themselves trajectories depending on the initial state in phase space. LangevinSimulator offers routines to compute the fluctuating forces for each trajectory as well as the drift term. From the fluctuating forces corresponding to each original trajectory, their mean value and auto-correlation function is estimated. As proven in the article "Generalized Langevin dynamics simulation with non-stationary memory kernels: How to make noise", this is enough to simulate self-consistent trajectories that reproduce the mean and auto-correlation function and consequently reproduce the

drift term as well as the memory kernel. LangevinSimulator implements this technique by drawing the fluctuating forces from a multivariate Gaussian distribution with proper mean and covariance matrix. Before the numerical evaluation, the mean initial value is subtracted from the trajectories. After the numerical integration, the mean initial value is added onto the simulations. This way, correlations between the fluctuating forces and initial values are taken into account.

General instructions on the usage of LangevinSimulator are given on this mainpage. For further reading, please see the namespaces InputOutput and KernelMethods as well as the classes RandomForceGenerator and RK4.

1.1.3 Build instructions

1.1.3.1 Compile with Codeblocks

Open and compile the Release/Debug targets of the following CodeBlocks projects in `MyParentFolder/LangevinSimulator`:

- `main_correlation.cbp`
- `main_kernel.cbp`
- `main_fluctuating_forces.cbp`
- `main_simulator.cbp`

1.1.3.2 Compile with make

To compile all executables open a terminal in the folder `MyParentFolder/LangevinSimulator` and type:

```
./make_all.sh
```

This will `make` the following makefiles at `MyParentFolder/LangevinSimulator`:

- `makefile_correlation`
- `makefile_kernel`
- `makefile_fluctuating_forces`
- `makefile_simulator`

The binaries are located in `LangevinSimulator/bin/Release` and `LangevinSimulator/bin/Debug`.

1.1.3.3 Customize your build

If you desire to customize your build options, it is recommended to open the CodeBlocks projects and modify the build settings within CodeBlocks. After the project is saved you may use `cbp2make` in order to generate your desired makefile, e.g.

```
sudo apt install cbp2make
cd MyParentFolder/LangevinSimulator
cbp2make -in main_correlation.cbp -out makefile_correlation
```

You are now ready to compile using `make`, e.g.

```
make -f makefile_correlation
```

1.1.4 Usage

You may execute the code using **CodeBlocks**, the built-in **C++ Interface** or the **Python Interface**. A description of all available parameters is given in sec. **Parameters**.

Please ensure that the input trajectories are formatted properly. Example trajectories are located at `MyParentFolder/LangevinSimulator/TEST_DATA`. The first column contains the times and the following columns contain the corresponding values for the observables. The first few lines from the example trajectory `dipoleMoment0000.txt` are:

```
# System Size:  12.000000000000000  12.000000000000000  12.000000000000000
# Number of Particles:  125
# Timestep: 0.010000000000000
# WCA-Parameters:
#   Epsilon:    1.000000000000000
#   Sigma:  2.000000000000000
# Data Format:  time    mu_x    mu_y    mu_z
0.000000000000000   125.000000000000000 0.000000000000000   0.000000000000000
0.100000000000000   124.999746318223842 0.000039394462241   -0.000022892471865
0.200000000000000   124.998984698373746 0.000156246425438   -0.000088155065376
0.300000000000000   124.997713835234592 0.000348563439825   -0.000180648094650
0.400000000000000   124.995931782722295 0.000626572571891   -0.000293840621097
0.500000000000000   124.993636080296639 0.000990302342154   -0.000415916202714
0.600000000000000   124.990823678351745 0.001436710617749   -0.000538403967656
0.700000000000000   124.987490854766818 0.001965428796087   -0.000653031331692
0.800000000000000   124.983633337966822 0.002578609901494   -0.000755257410848
0.900000000000000   124.979246078958454 0.003275395739426   -0.000836776042212
```

The following subsections explain how to calculate the following quantities: 1) correlation matrix 2) memory kernel 3) fluctuating forces (and drift term) 4) (self-consistent) simulations

Please mind that these quantities must be computed in consecutive order.

1.1.4.1 CodeBlocks

Open and run the CodeBlocks projects at `MyParentFolder/LangevinSimulator`: 1) `main_correlation.cbp` 2) `main_kernel.cbp` 3) `main_fluctuating_forces.cbp` 4) `main_simulator.cbp`

The parameters are set in `MyParentFolder/LangevinSimulator/parameter.txt`, see next subsection for details.

1.1.4.2 C++ Interface

The C++ Interface uses the software "parameter_handler" to read parameters from a text file. A verbatim copy of "parameter_handler" is located in `MyParentFolder/ParameterHandlder`. The parameter file for the given test data in `MyParentFolder/LangevinSimulator/TEST_DATA` could look like

```
out_folder TEST_OUT
in_folder TEST_DATA
in_prefix dipoleMoment
file_range 0-99
num_obs 2
t_min 50
t_max 90
increment 5
shift 1
num_sim 1000
txt_out True
```

This parameter file is stored at `MyParentFolder/LangevinSimulator/parameter.txt`.

Navigate to `MyParentFolder/LangevinSimulator` and

- calculate the correlation function
  `./bin/Release/main_correlation -f parameter.txt`
- calculate the memory kernel
  `./bin/Release/main_kernel -f parameter.txt`
- calculate the fluctuating forces for each trajectory as well as the drift term
  `./bin/Release/main_fluctuating_forces -f parameter.txt`
- simulate trajectories
  `./bin/Release/main_simulator -f parameter.txt`

1.1.4.3 Python Interface

The python interface consists of two python scripts `ParameterHandler.py` and `run.py` in `MyParent↩ Folder/LangevinSimulator/`. You may test all cpp executables with the test data from `MyParent↩ Folder/LangevinSimulator/TEST_DATA` as follows:

```
cd MyParentFolder/LangevinSimulator
python3 run.py
```

Please note that the script is supposed to be located and executed in `MyParentFolder/Langevin↩ Simulator`.

1.1.4.3.1 <**tt**>**ParameterHandler.py**</**tt**> The file `ParameterHandler.py` contains the actual interface, which is the `run(param_handler)` function that writes the parameter file and passes it to the cpp executables for you. The variable `param_handler` is an instantiation from the class `ParameterHandler`. It contains all available parameters as attributes. Please have a look at the implementation at `MyParentFolder/↩ LangevinSimulator/ParameterHandler.py` for details.

1.1.4.3.2 <**tt**>**run.py**</**tt**> The purpose of this script is to set all parameters which are passed to the interface on execution. Please have a look at the following implementation:

```
from ParameterHandler import ParameterHandler, run
import os
#######################################################################################################################
# COMPUTE CORRELATION FUNCTION, MEMORY KERNEL, DRIFT TERM, FLUCTUATING FORCES AND SIMULATE TRAJECTORIES
#######################################################################################################################
# set parameters (see 'ParameterHandler.py' for details)
params = ParameterHandler()
params.out_folder = "./TEST_OUT"
params.in_folder = "./TEST_DATA"
params.in_prefix = "dipoleMoment"
params.file_range = "0-99"
params.num_obs = 2;
params.t_min = 50
params.t_max = 90
params.increment = 5
params.shift = True
params.fluctuating_force = True
params.num_sim = 1000
params.txt_out = True
params.corr_exe = "./bin/Release/main_correlation"
params.kernel_exe = "./bin/Release/main_kernel"
params.ff_exe = "./bin/Release/main_fluctuating_forces"
params.sim_exe = "./bin/Release/main_simulator"
# RUN CPP EXECUTABLES
run(params)
# save this script
os.system("cp "+os.path.basename(__file__)+" "+params.out_folder+'/'+"run.py")
```

Note that this script will save itself to `<out_folder>/<name_of_run_script>`, so you do not need to store your parameters yourself. Further, it is possible to execute several scripts in parallel. To this end, simply create a copy `my_run.py` within `MyParentFolder/LangevinSimulator`, set a new `<out_folder>` and proceed as usual.

1.1.4.4 Parameters

The following parameters are used for the cpp interface as well as the python interface.

- `<out_folder>`:

  Directory in which all output files are stored. Default: `"./OUT"`. Type: string.

- `<in_folder>`:

  Directory where the input trajectories are located. Default: `"./TEST_DATA"`. Type: string.

- <in_prefix>:

  Input trajectories must be stored as <in_folder>/<in_prefix><n>.txt, where <n> is a positive integer. Default "". Type: string.

- <file_range>:

  Specify the input trajectories to be used. Takes the form "0-99" or "0-99,200-299". Default: "0-999". Type: string.

- <num_obs>:

  Number of observables to be used. Default 1. Type: positive integer.

- <t_min>:

  Initial time. Default 0.0. Type: floating point.

- <t_max>:

  End of desired time interval. Default 1000.0. Type: floating point.

- <increment>:

  Only use every <increment>-th time step. Default 1. Type: positive integer.

- <shift>:

  Enable (recommended) or disable the shift. The shift will subtract the average initial value from your trajectories before numerical evaluation. The shift will be added again after the simulations have been carried out. Note that in general you will only obtain self-consistent simulations if <shift> is set to true. Default True. Type: boolean.

- <num_sim>:

  Number of simulations to be carried out. Default 0. Type: unsigned integer.

- <text_out>:

  Enable/disable output files as text files. Default True. Type: boolean.

The follwing parameters are only used by the python interface.

- <fluctuating_force>:

  Enable/disable the calculation of the fluctuating forces. Default: False. Type: boolean.

- <corr_exe>:

  Set the path to the executable main_correlation. Default: "./bin/Release/main_correlation". Type: string.

- <kernel_exe>:

  Set the path to the executable main_kernel. Default: "./bin/Release/main_kernel". Type: string.

- <ff_exe>:

  Set the path to the executable main_fluctuating_forces. Default: "./bin/Release/main_fluctuating_forces". Type: string.

- <sim_exe>:

  Set the path to the executable main_simulator. Default: "./bin/Release/main_simulator". Type: string.

1.1.5 Utilities

Please have a look at MyParentFolder/LangevinSimulator/utilities for some additional python utilities in order to read/write and plot your data.

1.1.6 Trouble shooting

In the unfortunate case that an error occurs, please send a bug report to <christoph.widder[at]merkur.uni-freiburg.de>. You may want to try the debug builds in `MyParentFolder/LangevinSimulator/bin/Debug`. When using the python interface, simply set the executable paths in your `run.py` accordingly. The debug builds support debugging symbols and additional error-handling from within TensorUtils.

1.1.7 Licenses

1.1.7.1 parameter_handler: Redistribution of licensed, open-source software.

Please find the licenses for all files from the software "parameter_handler" in its verbatim copy at `MyParentFolder/ParameterHandler` or visit https://github.com/andreashaertel/parameter_handler.

1.1.7.2 TensorUtils: Redistribution of licensed, open-source software.

Please find the license for all files from the software "TensorUtils" in its verbatim copy at `MyParentFolder/TensorUtils` or visit https://github.com/TensorUtils/TensorUtils.

1.1.7.3 LangevinSimulator: Licensed, open-source software.

Please find the license "LICENSE" for the software "LangevinSimulator" in its source directory `MyParentFolder/LangevinSimulator`. All files within the source directory `MyParentFolder/LangevinSimulator` (and subfolders) are part of the software "LangevinSimulator" and are licensed under the GNU GPL3, if not stated otherwise.

LangevinSimulator Version 1.0

Copyright 2020-2022 Christoph Widder, Fabian Koch and Tanja Schilling

- Christoph Widder <christoph.widder[at]merkur.uni-freiburg.de>
- Fabian Koch <fabian.glatzel[at]physik.uni-freiburg.de>
- Tanja Schilling <tanja.schilling[at]physik.uni-freiburg.de>

This file is part of LangevinSimulator.

LangevinSimulator is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

LangevinSimulator is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with LangevinSimulator. If not, see https://www.gnu.org/licenses/.

Chapter 2

Namespace Index

2.1 Namespace List

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

Namespace Documentation

4.1 InputOutput Namespace Reference

This namespace provides functions to read/write trajectories from/to text files. Square matrices as a function of one or two times may be written as text files.

Functions

- std::vector< std::string > getDataFilenames (std::string file_range, std::string in_folder, std::string in_prefix)

  Get paths to the trajectories `<in_folder>/<in_prefix><n>.txt` *for each* `n` *within the given ranges.*
- TensorUtils::tensor< double, 3 > readTrajectories (std::vector< std::string > &data_files, double t_min, double t_max, size_t increment, size_t num_obs)

  Read trajectories for each path in ´data_files´.
- TensorUtils::tensor< double, 1 > popTimes (TensorUtils::tensor< double, 3 > &trajectories)

  Erase the times from the trajectories and return them as a one dimensional tensor instead.
- void write (TensorUtils::tensor< double, 1 > &times, TensorUtils::tensor< double, 2 > &traj, std::filesystem↩ ::path out_path)

  Write a trajectory to a text file. Directory must exist.
- void write (TensorUtils::tensor< double, 1 > &times, TensorUtils::tensor< double, 4 > &corr, std::filesystem↩ ::path out_path)

  Write square matrices as a function of two times to a text file. Directory must exist.
- void write (TensorUtils::tensor< double, 1 > &times, TensorUtils::tensor< double, 3 > &corr, std::filesystem↩ ::path out_path)

  Write square matrices as a function of time to a text file. Directory must exist.

4.1.1 Detailed Description

This namespace provides functions to read/write trajectories from/to text files. Square matrices as a function of one or two times may be written as text files.

LangevinSimulator Version 1.0

Christoph Widder <christoph.widder[at]merkur.uni-freiburg.de> Fabian Glatzel <fabian.glatzel[at]physik.uni-freiburg.de>

This file is part of LangevinSimulator.

LangevinSimulator is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

LangevinSimulator is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with LangevinSimulator. If not, see https://www.gnu.org/licenses/.

4.1.2 Function Documentation

4.1.2.1 getDataFilenames()

```
std::vector<std::string> InputOutput::getDataFilenames (
            std::string file_range,
            std::string in_folder,
            std::string in_prefix )
```

Get paths to the trajectories `<in_folder>/<in_prefix><n>.txt` for each `n` within the given ranges.

Parameters

file_range	Takes the form `0-999` or `0-99,200-299`.
in_folder	Path that contains the trajectories to be loaded.
in_prefix	Prefix of the desired trajectories.

Returns

`vector<string>` with all paths to the desired trajectories.

4.1.2.2 popTimes()

```
TensorUtils::tensor<double,1> InputOutput::popTimes (
            TensorUtils::tensor< double, 3 > & trajectories )
```

Erase the times from the trajectories and return them as a one dimensional tensor instead.

Parameters

trajectories	Contains the trajectories obtained from readTrajectories(std::vector<std::string> &data_files,double t_min,double t_max,size_t increment,size_t num_obs).

Returns

Times for corresponding trajectories.

4.1.2.3 readTrajectories()

```
TensorUtils::tensor<double,3> InputOutput::readTrajectories (
            std::vector< std::string > & data_files,
            double t_min,
            double t_max,
            size_t increment,
            size_t num_obs )
```

Read trajectories for each path in ´data_files´.

Parameters

data_files	Contains the paths obtained from getDataFilenames(std::string file_range, std::string in_folder, std::string in_prefix).
t_min	Initial time.
t_max	End of desired time interval.
increment	Load every `increment`-th time step only.
num_obs	Number of observables to be used.

Returns

Stores all trajectories in a tensor, i.e. `return_value(n,t,o)`. Here, `(n,t,o)` enumerates the trajectories, time-steps and observables, respectively, where `o=0` gives the time of the `t`-th time-step.

4.1.2.4 write() [1/3]

```
void InputOutput::write (
            TensorUtils::tensor< double, 1 > & times,
            TensorUtils::tensor< double, 2 > & traj,
            std::filesystem::path out_path )
```

Write a trajectory to a text file. Directory must exist.

The first column stores the times, the following columns contain the observables.

4.1.2.5 write() [2/3]

```
void InputOutput::write (
            TensorUtils::tensor< double, 1 > & times,
            TensorUtils::tensor< double, 3 > & corr,
            std::filesystem::path out_path )
```

Write square matrices as a function of time to a text file. Directory must exist.

This function is used to write the drift term to text files. The first column contains the times, the following columns contain the matrix elements in lexicographical order.

4.1.2.6 write() [3/3]

```
void InputOutput::write (
            TensorUtils::tensor< double, 1 > & times,
            TensorUtils::tensor< double, 4 > & corr,
            std::filesystem::path out_path )
```

Write square matrices as a function of two times to a text file. Directory must exist.

This function is used to write the memory kernel and correlation function to text files. The first two columns contain the two times, the following columns contain the matrix elements in lexicographical order.

4.2 KernelMethods Namespace Reference

This namespace provides functions to compute the memory kernel, drift and fluctuating forces.

Functions

- TensorUtils::tensor< double, 1 > shiftTrajectories (TensorUtils::tensor< double, 3 > &traj)

  Subtracts and returns the average of the trajectories.
- void mollifyTrajectories (TensorUtils::tensor< double, 1 > &times, TensorUtils::tensor< double, 3 > &traj, size_t mollifier_width)

  Fast convolution with a mollifier of sample-width `2*mollifier_width-1`.
- TensorUtils::tensor< double, 4 > getCorrelationFunction (TensorUtils::tensor< double, 3 > &traj, bool unbiased=false)

  Returns the cross-correlation matrix of the observable `C(t,i,s,j)=E[traj(n,t,i)*traj(n,s,j)]` *for all times* `t` *and* `s`.
- TensorUtils::tensor< double, 3 > getStationaryCorrelation (TensorUtils::tensor< double, 3 > &traj, bool unbiased=false)

  Returns the cross-correlation matrix of a stationary process `C(num_ts-1+t-s,i,j)=E[traj(n,t,i)*traj(n,s,j)]` *for all time differences* `t-s=-(N-1),...,+(N-1)`.
- TensorUtils::tensor< double, 2 > subAverage (TensorUtils::tensor< double, 3 > &traj)

  Subtracts and returns the average trajectory.
- TensorUtils::tensor< double, 4 > calcLowerBlockTriangularInverse (TensorUtils::tensor< double, 4 > &src)

  Returns the inverse of a lower block triangular matrix.
- void getMemoryKernel (gsl_matrix *kernel, gsl_matrix *corr, size_t const num_ts, size_t const num_obs, double const dt)

  Computes the non-stationary memory kernel for a given correlation function.

- TensorUtils::tensor< double, 3 > getMemoryKernel (TensorUtils::tensor< double, 3 > &correlation, double dt)

  Computes the stationary memory kernel for a given correlation function.
- TensorUtils::tensor< double, 3 > diffTrajectories (TensorUtils::tensor< double, 3 > &trajectories, double dt, bool darboux_sum)

  Computes the derivative of all trajectories using the symmetric difference quotient.
- TensorUtils::tensor< double, 3 > matInverse (TensorUtils::tensor< double, 3 > &mat)

  Block-wise matrix inversion.
- TensorUtils::tensor< double, 3 > diffFront (TensorUtils::tensor< double, 3 > &correlation, double dt)

  Differentiate with respect to the first index using the symmetric difference quotient. For the first and last index, the one-sided difference quotient is used.
- TensorUtils::tensor< double, 3 > getDrift (TensorUtils::tensor< double, 4 > &correlation, double dt)

  Returns the drift term for a given non-stationary correlation function.
- TensorUtils::tensor< double, 2 > getDrift (TensorUtils::tensor< double, 3 > &correlation, double dt)

  Returns the drift term for a given stationary correlation function.
- TensorUtils::tensor< double, 3 > getFluctuatingForce (TensorUtils::tensor< double, 4 > &kernel, Tensor↩ Utils::tensor< double, 3 > &drift, TensorUtils::tensor< double, 3 > &trajectories, TensorUtils::tensor< double, 1 > &times, bool darboux_sum)

  Returns the fluctuating forces for each trajectory for the non-stationary case.
- TensorUtils::tensor< double, 3 > getFluctuatingForce (TensorUtils::tensor< double, 3 > &kernel, Tensor↩ Utils::tensor< double, 2 > &drift, TensorUtils::tensor< double, 3 > &trajectories, TensorUtils::tensor< double, 1 > &times, bool darboux_sum)

  Returns the fluctuating forces for each trajectory for the stationary case.
- void writeCovarianceMatrix (TensorUtils::tensor< double, 3 > &ff, std::filesystem::path out_path, bool stationary=false)

  Computes the average and covariance matrix of the fluctuating forces as required from RandomForceGenerator.
- void writeExtendedCovarianceMatrix (TensorUtils::tensor< double, 3 > &traj, TensorUtils::tensor< double, 3 > &ff, std::filesystem::path out_path, bool stationary=false)

  Computes the average and covariance matrix of the initial values and fluctuating forces as required from RandomForceGenerator when drawing the initial values and fluctuating forces from a joint distribution.
- TensorUtils::tensor< double, 3 > simulateTrajectories (TensorUtils::tensor< double, 3 > &traj, Tensor↩ Utils::tensor< double, 3 > &drift, TensorUtils::tensor< double, 4 > &kernel, TensorUtils::tensor< double, 1 > &mean_initial_value, RandomForceGenerator &rfg, double dt, bool shift, bool gaussian_init_val, bool darboux_sum, size_t num_sim, std::filesystem::path out_path)

  Draws the fluctuating forces and simulates new trajectories for the non-stationary case.
- TensorUtils::tensor< double, 3 > simulateTrajectories (TensorUtils::tensor< double, 3 > &traj, Tensor↩ Utils::tensor< double, 2 > &drift, TensorUtils::tensor< double, 3 > &kernel, TensorUtils::tensor< double, 1 > &mean_initial_value, RandomForceGenerator &rfg, double dt, bool shift, bool gaussian_init_val, bool darboux_sum, size_t num_sim, std::filesystem::path out_path)

  Draws the fluctuating forces and simulates new trajectories for the stationary case.

4.2.1 Detailed Description

This namespace provides functions to compute the memory kernel, drift and fluctuating forces.

LangevinSimulator Version 1.0

Christoph Widder <christoph.widder[at]merkur.uni-freiburg.de> Fabian Glatzel <fabian.glatzel[at]physik.uni-freiburg.de>

This file is part of LangevinSimulator.

4.2.2 Function Documentation

4.2.2.1 diffTrajectories()

```
TensorUtils::tensor<double,3> KernelMethods::diffTrajectories (
            TensorUtils::tensor< double, 3 > & trajectories,
            double dt,
            bool darboux_sum )
```

Computes the derivative of all trajectories using the symmetric difference quotient.

At the boundary the one-sided difference quotient is used.

4.2.2.2 matInverse()

```
TensorUtils::tensor<double,3> KernelMethods::matInverse (
            TensorUtils::tensor< double, 3 > & mat )
```

Block-wise matrix inversion.

Returns

Returns a tensor defined by `return_val(n,:,:)=mat(n,:,:)^{-1}`.

4.2.2.3 writeCovarianceMatrix()

```
void KernelMethods::writeCovarianceMatrix (
            TensorUtils::tensor< double, 3 > & ff,
            std::filesystem::path out_path,
            bool stationary = false )
```

Computes the average and covariance matrix of the fluctuating forces as required from RandomForceGenerator.

The average fluctuating forces are written to `out_path/ff_average.f64`. The covariance matrix is written to `out_path/ff_cov.f64` or `out_path/ff_cov_stationary.f64` in the stationary case.

4.2.2.4 writeExtendedCovarianceMatrix()

```
void KernelMethods::writeExtendedCovarianceMatrix (
            TensorUtils::tensor< double, 3 > & traj,
            TensorUtils::tensor< double, 3 > & ff,
            std::filesystem::path out_path,
            bool stationary = false )
```

Computes the average and covariance matrix of the initial values and fluctuating forces as required from RandomForceGenerator when drawing the initial values and fluctuating forces from a joint distribution.

The average initial values and fluctuating forces are written to `out_path/ff_average.f64`. The covariance matrix of initial values and fluctuating forces is written to `out_path/ff_cov.f64`. In the stationary case, the covariance matrix of the fluctuating forces is written to `out_path/ff_cov_stationary.f64` and the covariance between the initial values and fluctuating forces is written to `out_path/ff_cov_extended.f64`.

Chapter 5

Class Documentation

5.1 RandomForceGenerator Class Reference

This class offers routines to generate multivariate normal distributed fluctuating forces.

`#include <RandomForceGenerator.hpp>`

Public Member Functions

- RandomForceGenerator ()

  Returns an uninitialized instantiation, but sets up the random number generator.
- void init_cov (TensorUtils::tensor< double, 2 > &ff_average, TensorUtils::tensor< double, 4 > &ff_cov, std::filesystem::path out_path)

  Computes and writes the rotation matrix for a given covariance matrix. On exit, the RandomForceGenerator is initialized.
- void init_decomp (TensorUtils::tensor< double, 2 > &ff_average, TensorUtils::tensor< double, 4 > &ff_decomp)

  Initializes the RandomForceGenerator with a previously computed rotation matrix.
- TensorUtils::tensor< double, 2 > pull_multivariate_gaussian ()

  Draw multivariate normal distributed fluctuating forces.

Protected Member Functions

- void set_decomp (TensorUtils::tensor< double, 4 > &source, gsl_matrix *dest)

  Computes `M=UD^{1/2}`, where `C=UDU^{-1}` is the spectral decomposition of the covariance matrix of the fluctuating forces.

5.1.1 Detailed Description

This class offers routines to generate multivariate normal distributed fluctuating forces.

LangevinSimulator Version 1.0

Christoph Widder <christoph.widder[at]merkur.uni-freiburg.de> Fabian Glatzel <fabian.glatzel[at]physik.uni-freiburg.de>

5.1.2 Member Function Documentation

5.1.2.1 pull_multivariate_gaussian()

```
TensorUtils::tensor<double,2> RandomForceGenerator::pull_multivariate_gaussian ( )
```

Draw multivariate normal distributed fluctuating forces.

Returns

Returns the fluctuating forces `rand_ff = Mz`. The rotation matrix `M` is given by `M=UD^{1/2}`, where `C=UDU^{-1}` is the spectral decomposition of the covariance matrix and `z` contains independent, standard normal distributed random numbers.

5.1.2.2 set_decomp()

```
void RandomForceGenerator::set_decomp (
            TensorUtils::tensor< double, 4 > & source,
            gsl_matrix * dest )  [protected]
```

Computes `M=UD^{1/2}`, where `C=UDU^{-1}` is the spectral decomposition of the covariance matrix of the fluctuating forces.

This function is called within the constructor and stores the matrix `M` in an internal buffer to be used by pull_multivariate_gaussian().

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

- RandomForceGenerator.hpp

5.2 RK4 Class Reference

This class offers routines to integrate the nsGLE using the classical Runge-Kutta method (RK4).

`#include <RK4.hpp>`

Public Member Functions

- void integrate (double dt, TensorUtils::tensor< double, 3 > &drift, TensorUtils::tensor< double, 4 > &kernel, TensorUtils::tensor< double, 2 > &traj, TensorUtils::tensor< double, 2 > &rand_ff)

  Integrates the equation of motions for a given realization of the fluctuating force `rand_ff` using the classical Runge-Kutta method.
- void integrate (double dt, TensorUtils::tensor< double, 2 > &drift, TensorUtils::tensor< double, 3 > &kernel, TensorUtils::tensor< double, 2 > &traj, TensorUtils::tensor< double, 2 > &rand_ff)

  Overloaded function for the stationary case.

Protected Member Functions

- TensorUtils::tensor< double, 1 > f (size_t n, double dt, TensorUtils::tensor< double, 3 > &drift, TensorUtils::tensor< double, 4 > &kernel, TensorUtils::tensor< double, 2 > &traj, TensorUtils::tensor< double, 2 > &rand_ff)

  *Returns the time-derivative of the observable 'A'(t)=:f(t,A(t)), `where`t=n*dt`.*
- TensorUtils::tensor< double, 1 > f (size_t n, double dt, TensorUtils::tensor< double, 2 > &drift, TensorUtils::tensor< double, 3 > &kernel, TensorUtils::tensor< double, 2 > &traj, TensorUtils::tensor< double, 2 > &rand_ff)

  Overloaded function for the stationary case.
- void process (size_t n, double dt, TensorUtils::tensor< double, 3 > &drift, TensorUtils::tensor< double, 4 > &kernel, TensorUtils::tensor< double, 2 > &traj, TensorUtils::tensor< double, 2 > &rand_ff)

  Executes a classcal Runge-Kutta step with step-size `2dt`.
- void process (size_t n, double dt, TensorUtils::tensor< double, 2 > &drift, TensorUtils::tensor< double, 3 > &kernel, TensorUtils::tensor< double, 2 > &traj, TensorUtils::tensor< double, 2 > &rand_ff)

  Overloaded function for the stationary case.

5.2.1 Detailed Description

This class offers routines to integrate the nsGLE using the classical Runge-Kutta method (RK4).

LangevinSimulator Version 1.0

Christoph Widder <christoph.widder[at]merkur.uni-freiburg.de> Fabian Glatzel <fabian.glatzel[at]physik.uni-freiburg.de>

5.2.2 Member Function Documentation

5.2.2.1 integrate()

```
void RK4::integrate (
            double dt,
            TensorUtils::tensor< double, 3 > & drift,
            TensorUtils::tensor< double, 4 > & kernel,
            TensorUtils::tensor< double, 2 > & traj,
            TensorUtils::tensor< double, 2 > & rand_ff )
```

Integrates the equation of motions for a given realization of the fluctuating force `rand_ff` using the classical Runge-Kutta method.

The initial value must be stored in `traj[0]`. The times must be equally spaced. The length of `traj` determines the length of the simulated trajectory.

5.2.2.2 process()

```
void RK4::process (
            size_t n,
            double dt,
            TensorUtils::tensor< double, 3 > & drift,
            TensorUtils::tensor< double, 4 > & kernel,
            TensorUtils::tensor< double, 2 > & traj,
            TensorUtils::tensor< double, 2 > & rand_ff )  [protected]
```

Executes a classcal Runge-Kutta step with step-size `2dt`.

Odd times are restored using the Verlet scheme. Local error is in `O(h**4)` instead of `O(h**5)`.

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

- RK4.hpp

Index