Tutorial:
Rare Event Sampling with FRESHS and FFS using the example of polymer translocation through a nanopore simulated with ESPResSo

Kai Kratzer, Joshua T. Berryman and Axel Arnold

December 18, 2013

Contents

1 Introduction and background

In this tutorial, we present the software package FRESHS [**?**] for parallel simulation of rare events using sampling techniques from the 'splitting' family of methods which are suited to simulate equilibrium and non-equilibrium systems. FRESHS provides a plugin system for software implementing the underlying physics of the system of interest. At present, example plugins exist for our framework to steer the popular MD packages GROMACS, LAMMPS and ESPResSo, but due to the simple interface of our plugin system, it is also easy to attach other simulation software or self-written code. Use of our framework does not require recompilation of the simulation program.

System states are managed using standard database technology[1] so as to allow checkpointing, scaling and flexible analysis. The communication within the framework uses standard TCP/IP networking and is therefore suited to high-performance parallel hardware as well as to distributed or even heterogeneous networks of inexpensive machines. For FFS we implemented an automatic interface placement [**?**] that ensures optimal, nearly constant flux through the interfaces. We introduce 'ghost' (or 'look-ahead') runs that remedy the bottleneck which occurs when progressing to the next interface.

FRESHS is open-source, providing a publicly available parallelized rare-event sampling system.

We focus now on the FFS simulation technique, which we use here to simulate the polymer translocation through a nanopore.

1.1 Forward Flux Sampling (FFS)

FFS is designed to sample the transition rate k_{AB} and transition trajectories from an initial state A to a final state B. The transition rate is split into two contributions $k_{AB} = \Phi P_B$, where Φ is the so-called *escape flux*, the flux of trajectories leaving the initial state A, and P_B is the probability that a trajectory that manages to leave the initial state A arrives at the final state B, instead of returning to A.

The initial and final states are defined in terms of an order parameter λ. If $\lambda <= \lambda_A$, the system is in the initial state, if $\lambda_A < \lambda < \lambda_B$, it is in the "barrier" region, and if $\lambda > \lambda_B$, the system is in the final state.

Usually, A is defined such that its boundary is still relatively frequently visited, so that Φ can be determined from a conventional brute-force simulation. P_B however is usually very small and therefore cannot be sampled directly. Therefore, the "barrier" region is partitioned by a set of n interfaces λ_i, where $\lambda_i < \lambda_{i+1}$ and $\lambda_0 \equiv \lambda_A$ and $\lambda_n \equiv \lambda_B$ (fig. **??**). The probability P_B is then given by

$$P_B = \prod_{i=0}^{n-1} p_i \tag{1}$$

where p_i is the probability to reach interface $i+1$ coming from interface i, before returning back to A. The advantage of this splitting is that the probabilities p_i are much higher than P_B itself, and therefore easier to sample. By tracking back the transition paths from interface to interface, FFS also generates transition trajectories, which can be used for investigations of the transition process.

[1] Here, we use sqlite3 databases.

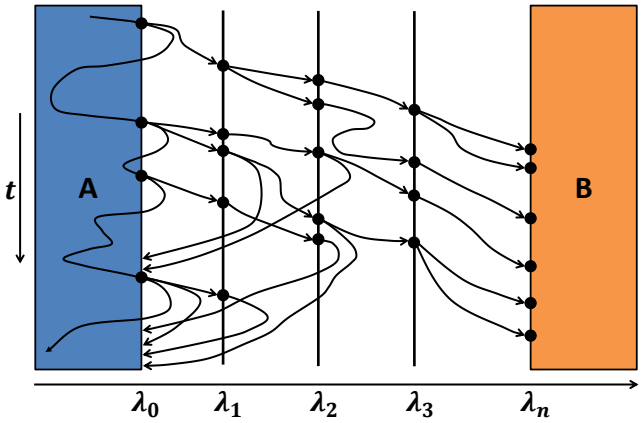

Figure 1: Schematic illustration of the FFS algorithm. The "barrier" region between the initial state A and the final state B is partitioned by a set of interfaces. The initial MD run in A is used to determine the *escape flux* Φ, the other lines depict the successful and the unsuccessful runs which are used to determine P_B. The dots represent configurations on the particular interface.

An implementation of the DFFS algorithm consists of two stages. First, the flux Φ across the first interface λ_0 is computed by setting up the system in the initial state and monitoring the frequency of trajectory crossings on λ_0. On occurence of such a crossing, the configuration of the system is stored if the crossing happened in the direction of increasing λ. This generates N_0 configurations corresponding to states of the system at λ_0. If the system reaches the final state during this run, it is reset to the initial state and re-equilibrated.

In the second stage of the DFFS algorithm, the partial probabilities p_i are computed step-by-step (fig. **??**). To compute p_1, one chooses configurations at random from the configurations stored at the first interface λ_0 and uses them to fire new "trial" trajectories. These new trajectories are continued until they reach either the next interface λ_1 or return to the previous interface λ_0. Again, the successful configurations at λ_1 are stored in a new set. After M_0 trial runs have been fired, p_1 is computed by dividing the number of successful runs by M_0. This procedure is repeated using the configurations at λ_1 as starting points for M_1 trial runs that are continued until they reach the next interface λ_2 or return to λ_0, and so on, until the final interface at λ_B is reached. This procedure results in a complete set of estimated partial probabilities p_i. Transition trajectories from the initial state A to the final state B can then be reconstructed from the collection of successful trajectories between the two states. For further information, refer to [**?**, **?**, **?**, **?**, **?**].

1.2 Polymer translocation through a nanopore

In this tutorial, we will use the software package ESPResSo [**?**] together with the Forward Flux Sampling method using ghost runs and automatic, optimized

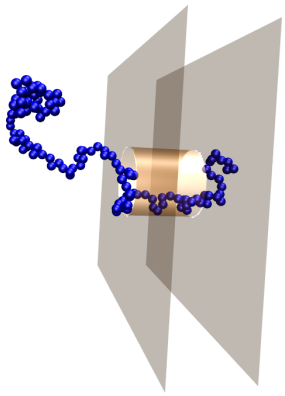

Figure 2: Polymer in a nanopore, set up and simulated using ESPResSo and visualized with VMD.

interface placement [**?**] to push a polymer through a nanopore (see also fig. **??**).

1.2.1 Reaction coordinate

As reaction coordinate, we use the z coordinate in direction of the pore of the center of mass of the polymer chain,

$$\lambda = \frac{1}{N} \sum_{i=0}^{N-1} z_i, \tag{2}$$

where N is the number of monomers in the polymer chain. Note, that this is probably not the best reaction coordinate, but an easy one which increases in direction to the final state B. As an advanced task you could think about other reaction coordinates and try to implement them.

1.2.2 Initial state A

For the setup in the initial state A, we set up a polymer with the first bead located at the entry of the pore. The rest of the polymer is set at random, using the capabilities of ESPResSo to set up random polymers. Then, we check the reaction coordinate. If it is in our basin of state A, we begin the initial MD run in A, if not, we delete the polymer and set a new one.

1.2.3 Final state B

The final state is located at a position λ_B, where most of the polymer has translocated through the pore. This is the case, when the reaction coordinate is larger than the z-coordinate of the center of the pore. To make sure, that we are really through, we set the border of state B a little bit behind.

1.2.4 Simulation scenario

For this simulation, we use a box with dimensions $30 \times 30 \times 60$ and periodic boundary conditions. The pore's center is in the middle of the box, with a

length of $l = 10.0$ in z-direction. The diameter of the pore can be specific in the configuration file (see sec. **??**), this is advantageous to investigate the effect of different radii of the pore. We recommend to start with a larger radius at the beginning, e.g. $r = 5$. The entry of the pore is located at $\lambda = 25$ and the center of the pore is at $\lambda = 30$. This means, the polymer is through the pore, when the reaction coordinate is larger than $\lambda = 30$. For the border of the initial domain A we choose $\lambda_A = 17$, which should be at the beginning of the increase of the free energy curve towards the entry of the pore.

The number of steps of an initial MD run is limited. However, if the polymer diffuses too much away from the pore, we have to interrupt the simulation before the center of mass is at the other side of the simulation box because of the periodic boundary conditions. Therefore, we initialize our system again in A, if the center of mass is lower than e.g. $\lambda = 5$.

The length of the polymer can also be tuned in the configuration file. We start with a shorter polymer, e.g. $N = 64$. The polymer uses FENE interactions for the bonds. The other interactions (e.g. polymer-pore) are set to Weeks-Chandler-Andersen interactions.

Now, we proceed with the software requirements and the setup of the environment.

2 Requirements

To follow this tutorial you will need:

1. A POSIX-compatible operating system (e.g. Linux, Mac OS X)
2. GIT[2]: Version control system to checkout the latest code
3. FRESHS[3]: The Flexible Rare Event Sampling Harness System
4. ESPResSo[4]: MD simulation tool which we will couple to FRESHS
5. Python with sqlite support and numpy to execute the code and analysis tools
6. Gnuplot[5]: Plotting tool for visualizations
7. SQL viewer (optional), e.g. SQLite Manager addon for firefox[6] or sqlitebrowser[7]
8. VMD[8] (optional)

Some package dependencies for a typical unix system might include:

1. **For ESPResSo:** tcl-dev, automake, libtool, g++

[2] http://git-scm.com/
[3] http://www.freshs.org
[4] http://espressomd.org
[5] http://www.gnuplot.info/
[6] https://code.google.com/p/sqlite-manager/
[7] http://sqlitebrowser.sourceforge.net/
[8] http://www.ks.uiuc.edu/Research/vmd/

2. **For FRESHS itself:** libsqlite3, numpy
3. **General:** csh (may be needed by vmd), texlive & bibtex (may be used to build documentation)

3 Preparing the environment

In this section we will get the required tools and prepare the directory structure, configuration files and simulation tools.

3.1 ESPResSo

Checkout the latest ESPResSo code (alternatively, you can use your already existing code snapshot):

```
git clone git://github.com/espressomd/espresso.git
```

Make sure that you have the following enabled in your myconfig.hpp:

```
#define CONSTRAINTS
#define LENNARD_JONES
```

Compile ESPResSo with these settings and note down the absolute path, where your 'Espresso' executable is located. You will need that information later in the client's configuration file.

3.2 Getting FRESHS and the tutorial files

The recommended way is to checkout the latest FRESHS development code from github:

```
git clone https://github.com/freshs/freshs.git
```

Alternatively, you can also download the .tar.bz2-package from the webpage, but the development code should have the latest updates included.

After checking out the code from github, several subdirectories should be located in the freshs-directory, here is an overview:

- client: The code of the client which connects to the server and launches the harness scripts and the simulation tool.
- doc: Documentation of the framework.
- harnesses: Example harness scripts.
- scripts: Analysis scripts to analyze data during and after the simulation. These scripts can also serve as templates for own analysis scripts.
- server: The code of the server which implements all methods and distributes jobs to the simulation clients.
- test: Directory containing some test codes. Not used in this tutorial.
- tutorial: The tutorial files.

3.3 Preparing the configuration files

3.3.1 Server

Navigate to the 'server' folder of the 'freshs' directory. Copy the ffs example configuration file to a new configuration file for our simulation:

```
cp server-sample-ffs.conf server-espresso-ffs_tutorial.conf
```

Open the file with an editor of your choice and change the following sections:

```
[general]
# Change this to something individual (do not use 42, 1337, ...),
# should be above 1000 and less than 65535
listenport =
...
# turn on ghosts
use_ghosts = 1
...
# adapt user_msg for easy configuration of the data directory,
# pore diameter and polymer length.
# Think about a directory, where the snapshots can be stored
user_msg = "servername": "polytranslocsrv", "storedir":
"/insert/path/to/data/directory", "p_length": 64, "pore_rad": 5

[ffs_control]
# enable parallel escape runs
parallel_escape = 1
...
# use 100000 integration steps for each MD escape trace
escape_steps = 100000

[auto_interfaces]
# use automatic interface placement
auto_interfaces = 1
# minimal distance between interfaces
auto_mindist = 0.001
# maximum number of calculation steps for exploring runs
auto_max_steps = 100000
# minimum acceptable estimated flux
auto_flux_min = 0.4
# maximum acceptable estimated flux
auto_flux_max = 0.6
# use exploring scouts method. Clients must support this.
auto_histo = 1

[hypersurfaces]
# borderA = lambda_A = lambda_0
borderA = 17.0
# borderB = lambda_B = lambda_n
borderB = 33.0

[runs_per_interface]
# borderA = lambda_A = lambda_0
borderA = 200
# borderB = lambda_B = lambda_n
borderB = 100
```

3.3.2 Client

Navigate to the 'client' folder of the 'freshs' directory. Copy the client example configuration file to a new configuration file for our simulation:

```
cp client-sample.conf client-espresso-ffs_tutorial.conf
```

Open the file with an editor of your choice and change the following sections:

```
[general]
# the host to connect to, if you only use your machine, set this to localhost
host = localhost
```

```
# port of the server which you entered in the server's config
port =
...
# the path to the Espresso executable (no quotes, absolute path!)
executable = /home/myusername/espresso/Espresso
# location of the harness dir where the 'job_script' is located (no quotes!)
# (will be created in the next section)
harness = ../harnesses/espresso-ffs_tutorial
...
# set unix niceness to 19 to not disturb other processes...
nice_job = 19

[ffs_control]
# ESPResSo is capable of checking the order parameter
checking_script = 1
```

3.4 Preparing the harness script

The harness script for this tutorial is based on the espresso_plain example from the harness directory and is located at

```
harnesses/espresso-ffs_tutorial
```

Open the file job_script and get a general idea what this script does (the comments in the file should help a lot). Then, look for comments containing a 'TODO' and fix the particular parts. Hints:

- calc_rc: `[lindex [part <id>] 4]` gives the z-coordinate of the particle with index id. Loop over all particles with ids 0 to $p_length.
- The command to add something to a TCL list is 'lappend'.

The harness script contains sections to visualize the polymer using VMD. Therefore, VMD must be installed. However, this is only reasonable for the first run, to ensure that the system is set up correctly. For the productive run, this sections should be commented, you'll see why when you start the client.

4 Launching FRESHS

FRESHS consists of two separate python programs, the *server* and the *client*. Parallelism is achieved by running multiple copies of the client; which are controlled centrally by the server. In addition, ESPResSo can calculate the physics in parallel, too. Note, that this makes only sense for larger systems, not for a polymer of 64 beads like in our example. Here, it should be faster to use multiple clients.

Open a new terminal, navigate to the server directory and start the server with

```
python main_server.py -c server-espresso-ffs_tutorial.conf
```

This should start the server which listens now on the specified port. Open a new terminal (or a new tab in the current session for better overview), navigate to the client directory and start a client using

```
python main_client.py -c client-espresso-ffs_tutorial.conf
```

which starts a client connecting to the server.

If things aren't working, then check the error messages and make sure that you set the right paths to the harness and to the various executables. If things are hanging you can e.g. kill all clients using the following command:

lambda	configpoint	origin_po...	calcsteps	ctime	runtime	myid	seed	lpos	uuid
0	[['0/client0001_time137...	escape	1030	10.3	17.74117398262024	client0001_2	1492836801	15	b2f56d6d-c509-4646-ae32-762556fb06cf
0	[['0/client0002_time137...	escape	870	8.7	16.096243143081665	client0002_2	121666977	15	602d4515-fba4-4827-8ae6-0279f25df693
0	[['0/client0003_time137...	escape	900	9	15.849206924438477	client0003_2	1352147194	15	cfbf85b1-b2ab-4494-9bee-79e1a8051d2e
0	[['0/client0001_time137...	client0001_2	250	2.5	4.069807052612305	client0001_3	518982348	15	d4fe6e2e-2732-4b72-807e-f479d900f6f7
0	[['0/client0001_time137...	client0001_3	100	1	1.6469919681549072	client0001_4	1630601806	15	8e67cc32-3906-478f-8749-a399d386e038
0	[['0/client0001_time137...	client0001_4	60	0.6	2.256424903869629	client0001_5	679866450	15	b47883bc-527c-4233-99b6-9328459c7b41
0	[['0/client0002_time137...	client0002_2	1050	10.5	14.326093912124634	client0002_3	229621548	15	b33ca76c-a853-4f1c-a9fe-b0c6cbcef7eb
0	[['0/client0001_time137...	client0001_5	990	9.9	12.76273798942566	client0001_6	428070566	15	dd4e277f-3921-47f0-825f-e20446492737
0	[['0/client0001_time137...	client0001_6	50	0.5	1.2672169208526611	client0001_7	1083197038	15	c704b2f5-0027-462b-969d-a00135528a89
0	[['0/client0003_time137...	client0003_2	1950	19.5	25.18477201461792	client0003_3	1853633291	15	866cd4ee-b692-43c9-bdf1-1e5bbafde477
0	[['0/client0001_time137...	client0001_7	2230	22.3	31.511761903762817	client0001_8	1541147800	15	1c9675ec-1a31-4d36-b24c-f68eeae9ae6d
0	[['0/client0002_time137...	client0002_3	3860	38.6	49.49374294281006	client0002_4	1940435785	15	8d786103-da5f-4b57-9ed4-a0d6f6b57add
0	[['0/client0002_time137...	client0002_4	180	1.8	2.651998996734619	client0002_5	618307292	15	949bd0a0-d6d3-49e8-9bff-97fe359c24f5
0	[['0/client0000_time137...	escape	6350	63.5	91.6383900642395	client0000_2	196022417	15	766f3189-ccd4-4ab7-9834-fb0a5778ed98

Figure 3: View of the database with SQLite Manager.

```
kill `ps -ef | grep main_client.py | grep client-espresso-ffs | awk '{print $2}'`
```

If everything is running, have a look at the command line output of the server and if you see numbers counting up, lean back and relax, grab a coffee or go to the toilet. Then proceed with the next section.

5 Analysis tools

FRESHS saves a comprehensive description of everything it has done into an SQL database, which by default goes into the directory 'DB'. This database is deeply informative, but is not immediately readable, so a number of scripts are provided to construct graphs based on the information within. At the end of a simulation, a summary of the simulation is printed and the interface information and transition rates are saved in the 'OUTPUT' directory. In addition to that, several post-processing tools which read out the database can be applied.

5.1 Database view

If you have the 'sqlitebrowser' or the firefox 'SQLite Manager' plugin installed, then use them to have a look at the raw data which comprises the output database, really it is just a big table of simulation information with one row for each trajectory fragment (fig. **??**), which contains the state at the end of the fragment, as well as enough information to re-generate it (i.e. a pointer to the parent fragment, and an RNG seed). The database can be already analyzed and viewed during the simulation. Note, that you should not write to the database during the simulation, handle with care. If the simulation fails at a certain stage, data can be modified/deleted, and the simulation can be resumed from the last state in the database using the '-r' flag of the server.

5.2 Analysis scripts

The folder 'scripts' in your FRESHS directory contains analysis scripts which can be used directly to e.g. backtrack successful runs or to serve as templates for own analysis scripts.

5.2.1 Backtracking successful runs

Change to the 'scripts' folder and run the following script:

```
./ffs_buildTree_success.py ../server/DB/<timestamp>_configpoints.sqlite
```

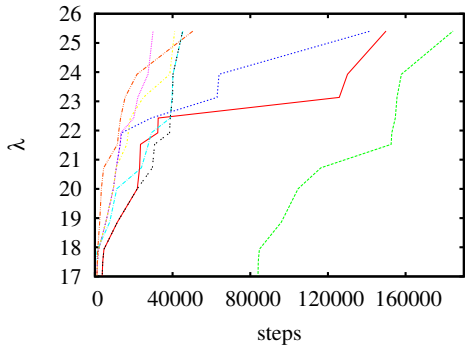

Figure 4: Backtracking successful runs is already possible during the simulation. Then, the runs are backtracked from the last known interface.

The output is written to a folder called 'OUTPUT/timestamp'. Change to this directory, start gnuplot and plot the tree graph to visualize the traces which lead to the points on the last known interface (fig. **??**):

```
load "tree_success.gnuplot"
```

5.2.2 Interface statistics

To plot statistical information about the simulation, e.g. a histogram of the runtime of the clients or a histogram of the calculation steps, run

```
./ffs_interfaces_statistics.py ../server/DB/<timestamp>_configpoints.sqlite
```

The output is written again to the folder called 'OUTPUT/timestamp'. Change to this directory, start gnuplot and plot the histograms:

```
load "histo_runtime.gnuplot"
load "histo_calcsteps.gnuplot"
```

5.2.3 Analyzing custom data

Use the template 'scripts/ffs_customdata.py' to write a script which plots the reaction coordinate distribution from the field 'customdata' of the database for each interface. This can already be applied when the simulation is still running like the analysis examples before.

```
#!/usr/bin/python

import sys
sys.path.append('../server/modules')
sys.path.append('../server/modules/ffs')

# custom
import configpoints

def extract_customdata(interface,dbhandle):
    customdata = dbhandle.return_customdata(interface)
    floatdata = []
    for el in customdata:
        for el2 in el.split():
            floatdata.append(float(el2))
    return floatdata

if len(sys.argv) < 2:
```

```
    print "Usage:", sys.argv[0], "<../server/DB/configpoint-DB-file>"
    exit(1)

cfph = configpoints.configpoints('none', sys.argv[1])

maxlam = cfph.biggest_lambda()

for i in range(maxlam+1):

    print "Interface", i
    print extract_customdata(i,cfph)
    # TODO: build histograms
```

6 FAQ and troubleshooting

- Clients do not find the executable:
  Please give the absolute path to the executable and check, that it has the right permissions.

- Killing clients:
  Sometimes, clients launched in a thread can not be killed by pressing CTRL+C. Therefore, it is useful to use e.g.

  ```
  kill `ps -ef | grep main_client.py | grep client-espresso-ffs | awk '{print $2}'`
  ```

  for killing all the clients which are executed with the espresso-ffs config. Please be careful doing this, you might e.g. first have a look at the output of the ps and grep combination.

- Speeding up the server:
  If the server's database is located e.g. on a network file system, the access can be slow. Change the location of the database folder in the server's configuration file to a faster location (or create a symbolic link), e.g. a local hard drive (/tmp directory should be always writeable, but remember the size limitation), an SSD or a temporary RAM filesystem (tmpfs).

- If a bond of the polymer breaks, consider reducing the timestep of the simulation (and increase the integration steps per cycle accordingly).

References

[1] K. Kratzer, J. T. Berryman, A. Taudt, and A. Arnold. The Flexible Rare Event Sampling Harness System (FRESHS). *Submitted*, 2013.

[2] K. Kratzer, A. Arnold, and R. J. Allen. Automatic, optimized interface placement in forward flux sampling simulations. *J. Chem. Phys.*, 138(16):164112, 2013.

[3] R. J. Allen, P. B. Warren, and P. R. ten Wolde. Sampling rare switching events in biochemical networks. *Phys. Rev. Lett.*, 94:018104, 2005.

[4] Rosalind J. Allen, Daan Frenkel, and Pieter Rein ten Wolde. Simulating rare events in equilibrium or nonequilibrium stochastic systems. *J. Chem. Phys.*, 124:024102, 2006.

[5] Rosalind J. Allen, Daan Frenkel, and Pieter Rein ten Wolde. Forward flux sampling-type schemes for simulating rare events: Efficiency analysis. *J. Chem. Phys.*, 124(19):194111, 2006.

[6] Rosalind J Allen, Chantal Valeriani, and Pieter Rein ten Wolde. Forward flux sampling for rare event simulations. *Journal of Physics: Condensed Matter*, 21(46):463102, 2009.

[7] C. Valeriani, R. J. Allen, M. J. Morelli, D. Frenkel, and P. R. ten Wolde. Computing stationary distributions in equilibrium and nonequilibrium systems with forward flux sampling. *J. Chem. Phys.*, 127:114109, 2007.

[8] Bernward A. Mann Hans-Jörg Limbach, Axel Arnold and Christian Holm. Espresso – an extensible simulation package for research on soft matter systems. *Comput. Phys. Commun. 174*, 9:704–727, 2006.