An Introductory Tutorial to the SEEKR2 (Simulation Enabled Estimation of Kinetic Rates v. 2) Multiscale Milestoning Software [Article v1.0]

Anupam Anand Ojha[1*], **Lane William Votapka**[1], **Gary Alexander Huber**[1], **Shang Gao**[1], **Rommie Elizabeth Amaro**[2*]

[1]Department of Chemistry and Biochemistry, University of California San Diego, La Jolla, California, 92093, United States; [2]Department of Molecular Biology, University of California San Diego, La Jolla, California, 92093, United States

This LiveCoMS document is maintained online on GitHub at https://github.com/anandojha/SEEKR_tutorials; to provide feedback, suggestions, or help improve it, please visit the GitHub repository and participate via the issue tracker.

This version dated February 15, 2024

Abstract

SEEKR2 (Simulation enabled estimation of kinetic rates v. 2) is a powerful and versatile software tool designed to computationally estimate the kinetics and thermodynamics of complex molecular processes, particularly emphasizing the process of receptor-ligand binding and unbinding. We present a suite of tutorials for the SEEKR2 (Simulation enabled estimation of kinetic rates v. 2) multiscale milestoning software. This tutorial presents a comprehensive guide for users offering the best practices for preparing, executing, and analyzing molecular dynamics (MD) and Brownian dynamics (BD) simulations using SEEKR2. This tutorial highlights the advancements presented in SEEKR2 - the latest iteration within the SEEKR programs, including significant improvements in speed and capabilities compared to its earlier versions. SEEKR2 now supports both NAMD and OpenMM simulation engines, providing users with more flexibility in their simulation setups. Additionally, the BD component has been upgraded to the Browndye2 engine, enhancing the accuracy and efficiency of simulations. This tutorial aims to guide users to install SEEKR2, run MD and BD simulations within the framework of the SEEKR2 program, and analyze and interpret the kinetics and thermodynamics of binding and unbinding of model host-guest systems, thereby demonstrating its ease of usability and extensible features that allow for future expansions of the method. This tutorial equips users with the necessary knowledge to effectively prepare, execute, and analyze simulations using SEEKR2. By following the best practices outlined in the tutorial, users can leverage the power of the SEEKR2 program to gain insights into complex molecular processes and accelerate their understanding of key biomolecular interactions.

***For correspondence:**
aaojha@ucsd.edu (AAO); ramaro@ucsd.edu (REA)

Received: August 31, 2023
Accepted: January 27, 2024

https://doi.org/https://doi.org/10.33011/livecoms.5.1.2359
Living J. Comp. Mol. Sci. 2024, 5(1), 2359

TUTORIAL OUTLINE

Section 1: Introduction

☐ Scope of tutorials
☐ Learning outcomes

Section 2: Prerequisites

☐ Background knowledge and experience
☐ Software and hardware requirements

Section 3: Background and Theory

☐ Brownian dynamics
☐ Markovian milestoning with Voronoi tessellations
☐ The SEEKR2 framework

Section 4: SEEKR2 Installation

☐ Creating a new conda environment
☐ Installing SEEKR2 dependencies
☐ Installing Browndye
☐ Conda installation of SEEKR2
☐ Installing SEEKR2 from source

Section 5: SEEKR2 tutorials

☐ Basic Tutorial: β-cyclodextrin (host)-guest complexes
☐ Advanced Tutorial: Trypsin-benzamidine complex

Section 6: SEEKR2 benchmarking

☐ Benchmarking on host-guest and trypsin benzamidine complexes

Section 7: Conclusion

1 Introduction

Significant progress has been made in computational biophysics since the 1970s with the advent of powerful computers and the development of molecular dynamics (MD) simulations and other computational techniques, enabling researchers to understand complex biological processes at the atomic level. Capitalizing on large-scale MD simulations and advanced computational approaches, this field provides useful perspectives into the molecular complexities of biological entities. From visualizing MD in real-time to predicting protein structures and drug-receptor interactions, the tools at our disposal have expanded precipitously. Techniques such as enhanced MD dynamics sampling and multiscale modeling emphasize the extent of its capabilities [1–8]. Such advancements not only elucidate kinetic and thermodynamic properties of interest in complex systems but also pave the way for transformative drug design and molecular biology breakthroughs [1, 9]. Among the recent advances, SEEKR2 (Simulation enabled estimation of kinetic rates v. 2) has emerged as a powerful tool to compute the kinetics and thermodynamics of complex biological processes, such as receptor-ligand binding and unbinding [10–14]. SEEKR2 is a multiscale simulation method that combines MD and Brownian dynamics (BD) simulations to compute the receptor-ligand binding (k_{on}) and unbinding (k_{off}) rates while providing valuable insights into the underlying mechanisms. SEEKR2 has numerous capabilities for predicting the kinetics and thermodynamics of molecular processes, even beyond predicting binding/unbinding kinetics. The SEEKR framework could be used to characterize membrane permeability or protein-protein association/dissociation. In short, the SEEKR program is capable of modeling any process that can be modeled with either MD or BD simulations in a reasonable amount of time and where a collective variable can adequately represent that process. SEEKR2 cannot model processes that are beyond the scope of MD or BD simulations. This includes processes like the breaking or forming of covalent bonds, as Quantum Mechanics/Molecular Mechanics (QM/MM) simulations are not currently integrated into OpenMM, which SEEKR relies on. This program requires a suitable collective variable based on atomic positions to describe the process it models. Therefore, it may not be useful for modeling certain phenomena, such as solvation energies, where such a collective variable is not applicable. While SEEKR2 offers relatively accurate kinetic estimates, this comes at the cost of computational efficiency. For example, modeling the binding/unbinding kinetics of a drug-like molecule to a target protein can take several days, even with access to advanced GPU clusters.

1.1 Scope of tutorials

This tutorial aims to provide a comprehensive understanding of the SEEKR2 multiscale milestoning software with detailed instructions to install the software, step-by-step instructions to set up SEEKR2 simulations, analyze the results, and interpret the kinetic and thermodynamic quantities obtained from simulations. Clear instructions and examples ensure users quickly adapt the tutorial and apply the SEEKR2 framework to their specific systems of interest. The authors expect this tutorial to serve as a practical guide for researchers interested in using this method to investigate receptor-ligand binding and unbinding kinetics. This tutorial is designed to be thorough and engaging, with an estimated completion time of approximately 4-6 hours. We encourage users to plan their learning sessions according to their availability and pace, as completing the tutorial in a single session may be challenging.

https://doi.org/https://doi.org/10.33011/livecoms.5.1.2359
Living J. Comp. Mol. Sci. 2024, 5(1), 2359

1.2 Learning outcomes

Upon completion of the tutorial, readers should be able to:

- **Conceptualize SEEKR2 framework**
  - Understand the underlying theory behind the SEEKR2 framework.
  - Understand the multiscale nature of SEEKR2 simulations, i.e., MD and BD simulations within the SEEKR2 framework.
- **Setup and Installation**
  - Install necessary dependencies for SEEKR2 installation.
  - Install the SEEKR2 package either by conda installation or directly from source.
  - Identify key configuration files and their purposes.
- **Run SEEKR2 simulations**
  - Hands-on experience with SEEKR2 simulations for the host-guest and trypsin-benzamidine complexes.
  - Walk through the three stages of SEEKR2 calculations for each complex, i.e., prepare, run, and analyze.
  - Outline the key steps and procedures involved in each stage.
  - Use SEEKR2 commands to initiate and monitor simulations.
- **Post-SEEKR2 simulation analysis and troubleshooting**
  - Analyze and interpret free energy profiles generated from SEEKR2 simulations.
  - Troubleshoot common issues that may arise during SEEKR2 simulations, utilizing the documentation and community resources effectively.

2 Prerequisites

2.1 Background knowledge and experience

To ensure a successful installation and effective utilization of the SEEKR2 software, users are recommended to possess foundational knowledge in several key areas. Familiarity with the following topics prior to proceeding with the installation and utilization of SEEKR2 will greatly facilitate the process and enhance the ability of users to harness the full potential of SEEKR2:

- **Linux**: SEEKR2 runs on Linux machines, necessitating a working knowledge of Linux commands. Users should be comfortable navigating the Linux file systems, executing commands with the Linux environment, and managing files using the terminal interface.
- **Anaconda**: Experience with Anaconda, a popular package manager and environment management system, is required for installing SEEKR2. Users should be familiar with creating new conda environments, managing and installing conda packages, and activating or deactivating environments as necessary.
- **MD simulations using OpenMM**: Prior experience in running MD simulations using the OpenMM engine is crucial for effectively utilizing SEEKR2. Users should have a substantial understanding of MD simulation principles, such as force fields, integrators, and simulation parameters and their implementation in the OpenMM simulation engine [15].
- **Force field parameterization using Amber**: It is beneficial for users to be familiar with AmberTools, especially with Amber's antechamber and LEaP programs for force field parameterization and system preparation, respectively. LEaP is essential for system preparation, including solvation, ion addition, and force field assignment, while the antechamber program is employed for parameterizing small molecules. Familiarity with these tools ensures smooth integration into the SEEKR2 workflow [16, 17].
- **Research collaboratory for structural bioinformatics (RCSB) protein data bank**: To initiate a SEEKR2 simulation, users need a Protein Data Bank (PDB) file, which can be obtained from the RCSB Protein Data Bank (https://www.rcsb.org). Users should be familiar with searching for specific PDB files and downloading them from the database for use in SEEKR2 simulations. Alternatively, one may obtain structures or systems already prepared for simulation from collaborators or a previous study.

2.2 Software and hardware requirements

Using Anaconda or Miniconda to install SEEKR2 is recommended because their package and environment management capabilities ensure easy installation and compatibility. Users can either install from the conda-forge channel or manually install SEEKR2 from the source code, but the latter approach can be more time-consuming and error-prone. MD simulations in the SEEKR2 framework are run using the OpenMM or the Nanoscale Molecular Dynamics (NAMD) simulation engines. The OpenMM engine can be installed simultaneously during the conda installation of the SEEKR2 OpenMM plugin when users select to install the SEEKR2 package through conda-forge. OpenMM must be installed from source (See section 4.5.1) when users opt to install

https://doi.org/https://doi.org/10.33011/livecoms.5.1.2359
Living J. Comp. Mol. Sci. 2024, 5(1), 2359

SEEKR2 from the source code. Alternatively, if OpenMM is not available or preferred, users can opt for the NAMD simulation engine. NAMD is a parallel MD code commonly accessible on computing clusters and supercomputers. It should be noted that although NAMD is a viable option for basic applications, not all SEEKR2 functionalities may be supported. Instructions for NAMD installation can be found at https://www.ks.uiuc.edu/Research/namd/.

Users may want to use a GPU-enabled machine to achieve higher-speed OpenMM simulations through the SEEKR2 framework. Following the instructions at https://developer.nvidia.com/cuda-toolkit or https://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html is highly recommended to download and install CUDA. Browndye2 must be installed for k_{on} calculations using BD simulations. Please refer to section 4.3 for instructions to install Browndye2.

To execute SEEKR2 simulations, users must have access to a Linux-based machine. Simulations can be carried out with just one processor, though this will be very slow for systems of interest in biomedical research and drug discovery. A GPU device or cluster is recommended for optimal performance of MD engines. When working with large receptor-ligand complexes, faster processors and greater memory capacity may be required to implement SEEKR2 calculations successfully. Users are advised to run the simulations while monitoring the resource usage to estimate the necessary storage and memory. Employing multiple processors for SEEKR2 simulations is recommended for BD simulations, where multiple processors can be utilized. When the NAMD simulation engine is used, employing multiple CPU processors will also accelerate the calculations. However, when using OpenMM, increasing the number of CPU processors may not lead to faster simulation speeds because the majority of the calculation is performed on the GPU, so the CPU is not likely to be a bottleneck.

3 Background and Theory

3.1 Brownian dynamics

To compute the association rate constant (k_{on}) for receptor-ligand complexes, we must use BD. BD simulations are computationally less expensive as compared to MD simulations due to implicit solvent and approximations such as the often-used rigid body approximation. Moreover, the encounter-based approach focuses on the initial stages of the binding process when the ligand is far from the receptor-binding site, and the approximations in BD suffice as a physical description. Explicit electrostatic interactions are considered between the protein and the ligand, which is particularly important for receptor-ligand complexes where electrostatic forces play a significant role in molecular recognition and binding. Browndye [18] is a software package that runs the BD simulations to compute the second-order rate constants for encountering two molecules using a simplified physical model.

Brownian dynamics describes motion at a mesoscopic level, in which models of large molecules are usually simplified to smaller collections of rigid bodies, and the solvent is treated using continuum theories [19] instead of explicitly with individual molecules. A recent review summarizes theoretical details, common methods and software, and some of the most recent applications of BD [20]. Considering Newton's equations of motion and assuming a separation between the time scales of the macromolecules and the solvent molecules, one can derive the following stochastic differential equation of motion [21]:

$$d\mathbf{x} = -(k_B T)^{-1}\mathbf{D}\cdot\nabla V dt + \sqrt{2dt\,\mathbf{D}}\cdot\mathbf{W} \quad (1)$$

where V is the potential energy of the system and the vector, $\mathbf{x}$ represents the state variables, such as positions and orientations of the solute molecules (solvent molecules are not typically explicitly included). The matrix, $\mathbf{D}$ (generalized diffusivity), represents the dynamic effects of the solvent, i.e., the hydrodynamic damping caused by motion through a fluid and the addition of thermal energy. The vector, $\mathbf{W}$ of uncorrelated random numbers following a unit Gaussian distribution gives stochastic effects. It is worth noting that the presence of the temperature, along with Boltzmann's constant, affects the thermal fluctuations. Like the MD models, there exists a force term (the gradient of the potential energy) in the BD model with components that include intermolecular electrostatics and also the averaged effects of the solvent, such as hydrophobic and solvent dielectric effects. As an example of the latter, the Browndye2 engine uses the software package APBS (Adaptive Poisson-Boltzmann Solver) to compute the electric field around a macromolecule, given the bulk properties of the solvent and dissolved ions [22]. Because BD simulations encompass several approximations from MD simulations, the number of variables is significantly reduced, and the time step dt can be in orders of magnitude larger than that used in MD simulations. In the case of the receptor-ligand complexes presented in this tutorial, both the receptor and the ligand are treated as rigid bodies at the BD level.

Given the simplifications and speedup of using BD simulations, it is possible to generate elongated trajectories of the ligand in the space around the receptor to estimate the association rate. Browndye2 uses the Luty-McCammon-Zhou algorithm (Figure 1) [23] (a variation of the earlier Northrup-Allison-McCammon algorithm [24]), which generates multiple trajectories, each of which ends either in an

https://doi.org/https://doi.org/10.33011/livecoms.5.1.2359
Living J. Comp. Mol. Sci. 2024, 5(1), 2359

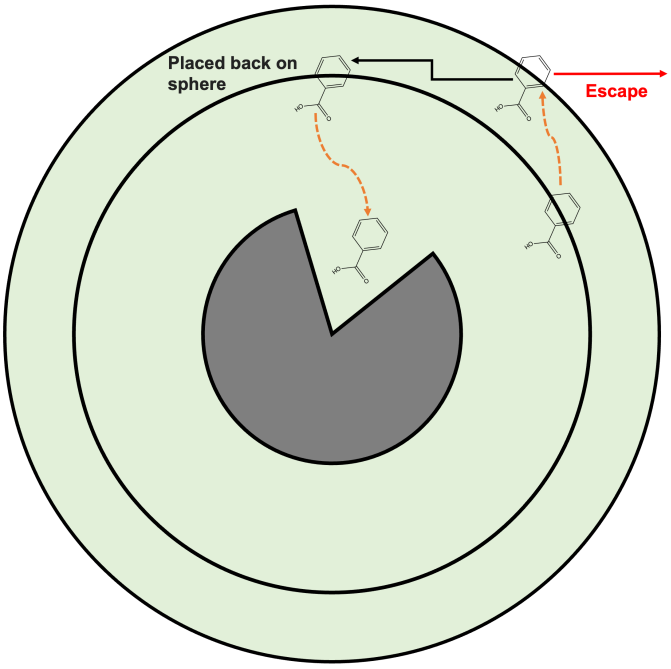


Figure 1. Computation of association rate using the Luty-McCammon-Zhou method

encounter with the receptor, or an escape. The ligand is started on a sphere surrounding the receptor and moves until it either reaches a final encounter with the receptor or reaches another sphere, larger and concentric with the first. If it reaches this second sphere, it either escapes, ending the trajectory, or it is placed back on the first sphere to continue the trajectory. The probability of the escape versus continuing from the inner sphere depends on several factors, such as the diameters of the spheres, the total charge on each system (ligand and the receptor), and the ionic strength of the solvent. Ultimately, the second-order association rate constant is computed from the proportion of encounters to escapes, and therefore a large number of trajectories are required to estimate that probability precisely. When used with SEEKR2, the encounter complex is defined as the ligand reaching the outermost milestone that encompasses the MD region. The milestoning calculations, with their parameters computed from the MD simulations, convert this initial rate constant into the final rate constant estimate, which accounts for the atomic details in the binding site, as described in section 3.2 and 3.3.

3.2 Markovian milestoning with Voronoi tessellations

Milestoning is a phase-space splitting strategy for enhanced MD sampling. For a thorough understanding of the milestoning theory, readers are advised to refer to the key papers by Elber and coworkers [25–27]. A series of mathematical equations are employed in the SEEKR2 milestoning methodology for calculating the mean first passage time (MFPT) and free energy related to binding for each milestone (ΔG_i). This tutorial recognizes the value of revisiting essential equations from our earlier works. Previous SEEKR2 publications employ the following mathematical representations [2, 3, 12, 28]. For the sake of comprehensiveness and to ensure this tutorial remains a self-contained resource, we have chosen to present these equations again.

Let us begin by segmenting the phase configuration of a bimolecular complex into N distinct milestones. The transitions between milestones is described in the transition rate matrix, $\mathbf{Q}$, of size $N \times N$, which represents the fluxes across milestones, as shown in equation 2. The matrix $\mathbf{Q}$ consists of diagonal elements, q_{ii}, and off-diagonal elements, q_{ij}.

$$\mathbf{Q} = \begin{pmatrix} q_{1,1} & q_{1,2} & \cdots & q_{1,N} \\ q_{2,1} & q_{2,2} & \cdots & q_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ q_{N,1} & q_{N,2} & \cdots & q_{N,N} \end{pmatrix} \tag{2}$$

Given the number of transitions between milestones i and j as N_{ij}, with the i^{th} milestone as the last milestone the trajectory has interacted with, and R_i represents the duration the trajectory has spent in the particular milestone, the expressions for q_{ii} and q_{ij} are given by equations 3 and 4 respectively.

$$q_{ii} = -\sum_{j \neq i} q_{ij} \tag{3}$$

$$q_{ij} = \begin{cases} \frac{N_{ij}}{R_i} & \text{if } R_i \neq 0 \\ 0 & \text{if } R_i = 0 \end{cases} \tag{4}$$

For a trajectory within the Voronoi cell, V_α, its position and velocity vectors are denoted as x_α and v_α respectively. The position and velocity of this trajectory at time, $t + \Delta t$, as estimated by the integrator algorithm (Langevin integrator, at most times), are given by ${x_\alpha}^*$ and ${v_\alpha}^*$. Equations 5 and 6 enforce reflective boundaries to ensure trajectories are confined to their respective Voronoi cells.

$$x_\alpha(t + \Delta t) = \begin{cases} {x_\alpha}^* & \text{if } {x_\alpha}^* \in V_\alpha \\ x_\alpha(t) & \text{otherwise} \end{cases} \tag{5}$$

$$v_\alpha(t + \Delta t) = \begin{cases} {v_\alpha}^* & \text{if } {x_\alpha}^* \in V_\alpha \\ -v_\alpha(t) & \text{otherwise} \end{cases} \tag{6}$$

The equilibrium probabilities of Voronoi cells V_α and V_β are represented by π_α and π_β respectively, with the total simulation time within these cells given by T_α and T_β. A dimension-consistent normalization factor, T, is given by equation 7.

https://doi.org/https://doi.org/10.33011/livecoms.5.1.2359
Living J. Comp. Mol. Sci. 2024, 5(1), 2359

$$T = \left(\sum_{\alpha=1}^{n} \frac{\pi_\alpha}{T_\alpha} \right)^{-1} \quad (7)$$

If N_{ij}^{α} denotes the number of collisions with the j^{th} milestone after last visiting the i^{th} milestone (in this context, both the i^{th} and j^{th} milestone are edges to anchor α), the total transitions between the i^{th} and j^{th} milestones, N_{ij}, is given by equation 8, as formulated in the original Markovian milestoning with Voronoi tesselations (MMVT) theory. [25]

$$N_{ij} = T \sum_{\alpha=1}^{n} \pi_\alpha \frac{N_{ij}^{\alpha}}{T_\alpha} \quad (8)$$

Given R_i^{α} represents the time a trajectory spends in Voronoi cell V_α after its last interaction with the i^{th} milestone, the aggregate time post the last interaction with the i^{th} milestone, R_i, is given by equation 9.

$$R_i = T \sum_{\alpha=1}^{n} \pi_\alpha \frac{R_i^{\alpha}}{T_\alpha} \quad (9)$$

To compute the stationary probabilities π, one must consider both $N_{\alpha,\beta}$ (number of collisions in Voronoi cell V_α that occur at its boundary shared with V_β) and $N_{\beta,\alpha}$ (number of collisions in V_β at its shared boundary with V_α), as given by equations 10 and 11.

$$\sum_{\beta=1,\beta\neq\alpha}^{n} \pi_\beta \frac{N_{\beta,\alpha}}{T_\beta} = \sum_{\beta=1,\beta\neq\alpha}^{n} \pi_\alpha \frac{N_{\alpha,\beta}}{T_\alpha} \quad (10)$$

$$\sum_{\alpha=1}^{n} \pi_\alpha = 1 \quad (11)$$

Consider $\hat{\mathbf{Q}}$ as the matrix of size N-1 by N-1 extracted from the upper left portion of **Q**. The rationale behind selecting the upper left section of the matrix **Q** while omitting the last column and row is to establish a “sink state" required to compute a kinetic quantity. By excluding the final column and row from **Q**, we effectively designate the state denoted by that particular row and column as the sink state. The vector **1** represents a vector of ones, the mean free passage times from each milestone, $\mathbf{T}^N$, can be determined by solving equation 12.

$$\hat{\mathbf{Q}}\mathbf{T}^N = -\mathbf{1} \quad (12)$$

The stationary probabilities of the milestones, **p**, are obtained by solving equation 13.

$$\mathbf{Qp} = \mathbf{p} \quad (13)$$

The stationary probabilities for the i^{th} milestone and the reference milestone are represented by p_i and p_{ref}, respectively. Using these parameters, the free energy landscape for the i^{th} milestone, ΔG_i, is given by equation 14, where k_B stands for Boltzmann’s constant and T designates the temperature.

$$\Delta G_i = -k_B T \ln \left(\frac{p_i}{p_{ref}} \right) \quad (14)$$

3.3 The SEEKR2 workflow

Let us consider the case of a model receptor-ligand complex to understand the steps involved in the SEEKR2 workflow. Initially, a structural (PDB) file containing the receptor protein and the ligand in an explicit solvent is provided as an input file (Figure 2a).

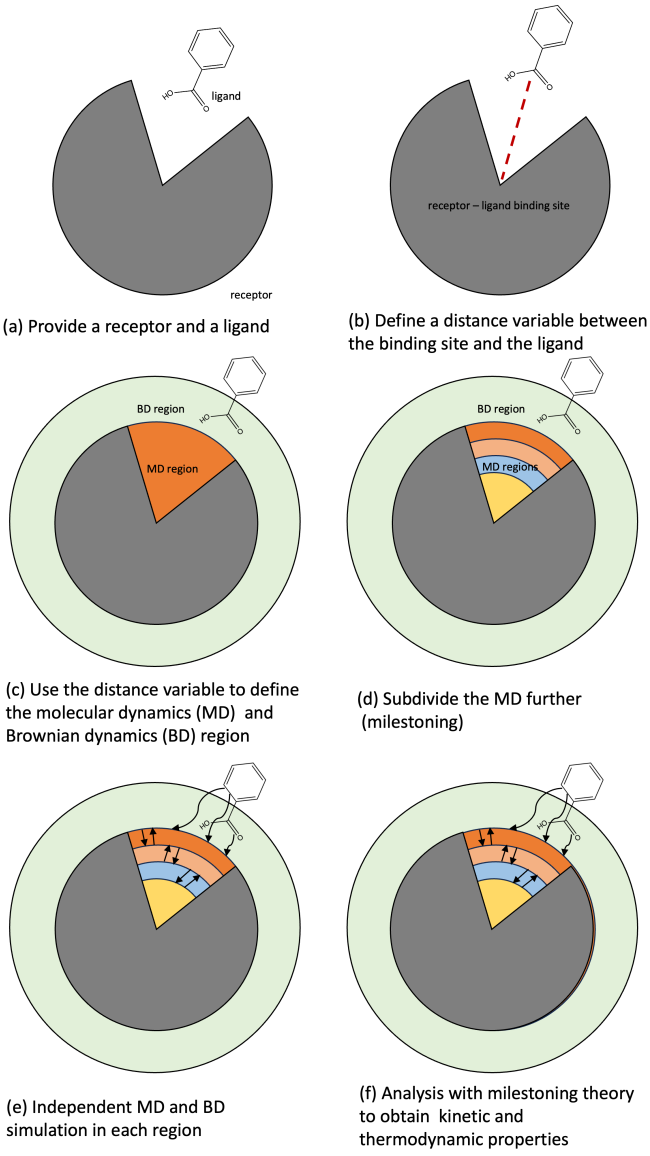
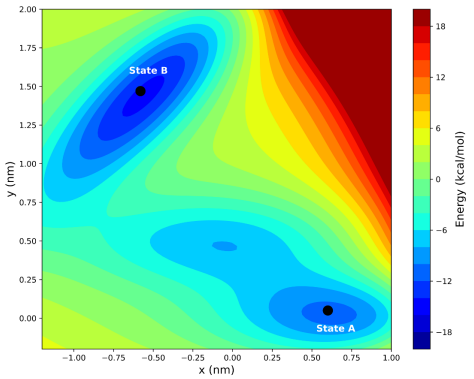
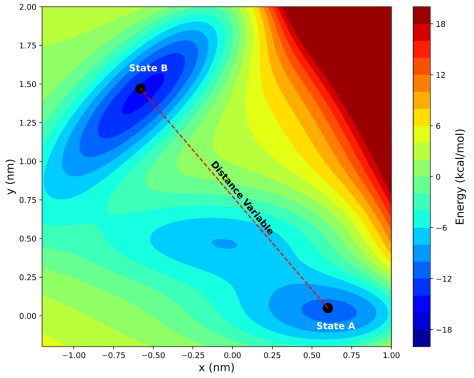


Figure 2. Different stages of the SEEKR2 simulation framework

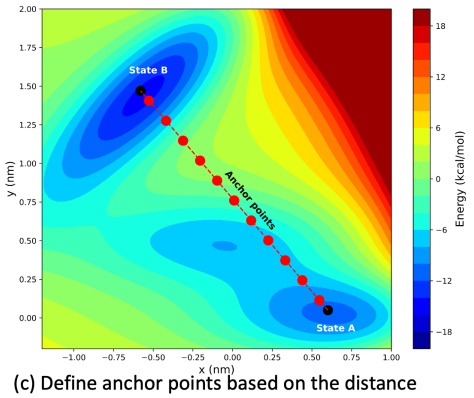
https://doi.org/https://doi.org/10.33011/livecoms.5.1.2359
Living J. Comp. Mol. Sci. 2024, 5(1), 2359



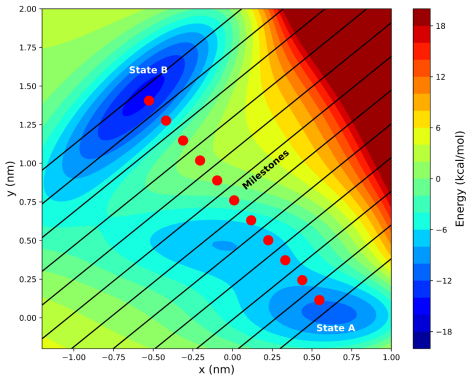
(a) Two states of a Muller potential energy system



(b) Define a distance variable between the state A and B



(c) Define anchor points based on the distance variable



(d) Define milestones based on the anchor points

Figure 3. A MMVT model of a simple Muller potential system within the SEEKR2 framework

We then identify the ligand in the PDB file through its atom indices. Once the ligand and the receptor protein are identified, we define a collective variable (CV). In most cases, for receptor-ligand binding and unbinding processes, we choose the CV to be the distance between the center of mass (COM) of α-carbon atoms at the binding site of the receptor and the COM of the heavy atoms of the ligand (Figure 2b).

Atomistic simulation details are required for receptor-ligand complexes when the ligand is close to the binding site, and conformational changes in the receptor are observed as the ligand slowly unbinds. In cases where the ligand is far from the receptor-ligand binding site, these two entities can be treated as point charges. Hence, BD is employed beyond a cut-off distance defined by the user. Therefore, the phase space of the receptor-ligand complex is comprised of the MD region and the BD region (Figure 2c). We further subdivide the MD region into milestones based on a pre-defined CV. In one dimension, these milestones are concentric spheres with radii based on the increasing distance between the COM of the receptor-ligand binding site and the ligand (Figure 2d). Independent MD and BD simulations are performed with reflective boundary conditions imposed within the MD Voronoi cells (Figure 2e). Once the simulation finishes, we can use milestoning theory to obtain the receptor-ligand unbinding rate (k_{off}) and the addition of BD theory to obtain the receptor-ligand binding rate (k_{on}) (Figure 2f). Thermodynamic parameters can then be obtained from the kinetic rates, k_{on} and k_{off}.

To further understand the process of milestoning employed in the SEEKR2 framework, let us consider a system exhibiting a simple Muller potential, where an energy barrier exists between the two equilibrium states, A and B (Figure 3a). Let us define a CV as a distance variable that approximately describes the transition from state A to state B (Figure 3b). Anchor points are placed equidistant along the distance variable, constituting the points of a one-dimensional Voronoi tesselation (Figure 3c). Once the anchor points are defined, milestones are placed between the anchor points (Figure 3d). Independent and parallel MD simulations are run in the region defined by two successive milestones with reflective boundary conditions.

4 SEEKR2 Installation

Before proceeding with the SEEKR2 installation, we assume that the user has a working knowledge of Linux and has successfully installed the Anaconda or Miniconda distribution in the system. This tutorial also assumes the user possesses a computer with one or more graphical processing units (GPUs) capable of installing GPU-enabled software and running MD simulations on GPUs.

https://doi.org/https://doi.org/10.33011/livecoms.5.1.2359
Living J. Comp. Mol. Sci. 2024, 5(1), 2359

4.1 Creating a new conda environment

- To verify the installation of conda in our system, we can run the following command in the terminal:

```
which conda
```

  If conda is properly installed, it will display the path to the conda executable.
- We will create a dedicated conda environment for the SEEKR2 package. We execute the following command in the terminal to create a new environment named SEEKR2 with Python version 3.9:

```
conda create --name SEEKR2 python=3.9
```

- To use the SEEKR2 environment, we need to activate it by executing the following command in the terminal:

```
conda activate SEEKR2
```

4.2 Installing SEEKR2 dependencies

- We will ensure that the SEEKR2 conda environment is activated.

```
conda activate SEEKR2
```

- To ensure that the Cython and git packages are installed, we execute the following commands in the terminal:

```
pip install --upgrade cython
```

```
conda install git
```

- For the proper execution of the tutorial scripts, we install the Ambertools and mdtraj packages via conda.

```
conda install -c conda-forge ambertools
```

```
conda install -c conda-forge mdtraj
```

- When installing SEEKR2 from source, we will also need to install the following additional packages (otherwise, not required for conda installation of SEEKR2):

```
conda install numpy
```

```
conda install scipy
```

```
conda install netcdf4
```

```
conda install mpi4py
```

```
conda install swig
```

```
conda install -c conda-forge doxygen
```

  To install the ccmake package, sudo privileges are required (otherwise, not required for conda installation of SEEKR2). We execute the following command in the terminal:

```
sudo apt-get install cmake-curses-gui
```

  To ensure the successful installation of ccmake, execute the following command:

```
which ccmake
```

4.3 Installing Browndye

Browndye2 (the latest version of the Browndye software) is compatible with various operating systems, including Linux, BSD, MacOS, and MS Windows. This tutorial will only provide instructions to install the Browndye package on Linux distributions. The recommended method for installing Browndye is to download the source code and compile it. Browndye relies on Ocaml and C++ compilers, with the C++17 version being the minimum requirement.

- We install the necessary package dependencies based on the Linux distribution:
  For Ubuntu (20.04 and 22.04), we run the following command in the terminal:

```
apt-get install make gcc g++ ocaml \
libexpat-dev liblapack-dev apbs
```

  For CentOS 7 distribution, we run the following commands to update the compilers and install the required packages:

```
yum install centos-release-scl epel-release
```

```
yum install devtoolset-9 ocaml expat-devel \
lapack-devel apbs
```

```
scl enable devtoolset-9 bash
```

- We will download the Browndye2 source code from https://browndye.ucsd.edu/downloads/browndye2.tar.gz. We recommend the user, though not required, to install the software in the home directory. We then extract the downloaded source code archive and install BrownDye2 by executing the following commands:

https://doi.org/https://doi.org/10.33011/livecoms.5.1.2359
Living J. Comp. Mol. Sci. 2024, 5(1), 2359

```
wget https://browndye.ucsd.edu/downloads/browndye2.tar.gz
```

```
tar xvfz browndye2.tar.gz
```

```
cd browndye2
```

```
make -j 4 all
```

- Once we have finished unpacking the files, it is important to ensure that we include the installation location of the Browndye2 software in the system's PATH variable. This will allow us to run the software from any directory without specifying the full path each time. If the software has been installed in the home directory, we can add the path by using the following command:

```
export PATH=/home/USERNAME/browndye2/bin:\
${PATH}
```

- To remove the downloaded file that is no longer necessary, move to the folder where Browndye2 is installed and execute the following command:

```
rm browndye2.tar.gz
```

Once the compilation process is complete, Browndye2 will be installed and ready to use on the Linux distribution.

4.4 Conda installation of SEEKR2

4.4.1 Installing SEEKR2-OpenMM plugin

- If not already activated, we first activate the SEEKR2 conda environment.

```
conda activate SEEKR2
```

- We will install the SEEKR2 plugin, which installs the OpenMM MD engine (version 7.7) and the CUDA Toolkit version 10.2 (compatible with SEEKR2), along with installing other dependencies.

```
conda install -c conda-forge \
seekr2_openmm_plugin
```

- To test the successful installation of SEEKR2, open a Python terminal and enter the following command:

```
python
>>> import seekr2plugin
```

4.4.2 Installing SEEKR2

- If not already activated, we first activate the SEEKR2 conda environment.

```
conda activate SEEKR2
```

- It is recommended, though not required, to install the SEEKR2 program in the home directory. So, we navigate to the home directory.

```
cd ~
```

- We will clone the SEEKR2 Python API repository by running the following command in the terminal:

```
git clone https://github.com/seekrcentral/seekr2.git
```

- We then proceed with the installation of SEEKR2 Python API in the cloned *seekr2* directory:

```
cd seekr2
```

```
python -m pip install .
```

- Once the installation is complete, it is recommended to run tests to ensure the proper functioning of SEEKR2. From within the *seekr2* directory, we execute the following command:

```
pytest
```

Running the tests may generate one or two failures depending on the availability of NAMD and Browndye2 software, which can safely be ignored if these programs are not required for your specific use case. SEEKR2 is now successfully installed on our system, and we can begin utilizing its features and functionalities.

4.4.3 Installing Seekrtools

Seekrtools is a suite of software utilities designed to work with SEEKR2 to streamline the preparation process and execution of multiscale milestoning simulations. For more comprehensive instructions and tutorials, please refer to the official documentation at https://seekrtools.readthedocs.io/en/latest. Ensuring that SEEKR2 and OpenMM packages are installed before installing Seekrtools is crucial, as most programs within Seekrtools rely on these packages.

- If not already activated, we first activate the SEEKR2 conda environment.

```
conda activate SEEKR2
```


https://doi.org/https://doi.org/10.33011/livecoms.5.1.2359
Living J. Comp. Mol. Sci. 2024, 5(1), 2359

- It is recommended, though not required, to install the seekrtools program in the home directory. So, we navigate to the home directory.

```
cd ~
```

- We will clone the seekrtools Python API repository by running the following command in the terminal:

```
git clone https://github.com/seekrcentral/seekrtools.git
```

- We then proceed with the installation of seekrtools Python API in the cloned *seekrtools* directory:

```
cd seekrtools
```

```
python -m pip install .
```

- Once the installation is complete, it is recommended to run tests to ensure the proper functioning of seekrtools. From within the *seekrtools* directory, we execute the following command:

```
pytest
```

4.5 Installing SEEKR2 from source

4.5.1 Installing OpenMM from source

Sometimes, a SEEKR2 installation from conda-forge will not be possible or desirable. In those cases, one will need to perform the more arduous and difficult process of installing OpenMM and the SEEKR2 OpenMM Plugin from source. Before installing the OpenMM package, it is essential to install CUDA. Please follow NVIDIA's CUDA toolkit installation manual instructions and refer to the documentation at https://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html. Carefully read and follow the steps outlined in the guide to ensure a successful CUDA installation. Once CUDA is installed, we can proceed with installing the OpenMM package.

- If not already activated, we first activate the SEEKR2 conda environment.

```
conda activate SEEKR2
```

- It is recommended, though not required, to install the OpenMM repository in the home directory. So, we navigate to the home directory.

```
cd ~
```

- To clone the OpenMM repository:

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

- We then navigate to the *openmm* directory:

```
cd openmm
```

- Create a *build* directory:

```
mkdir build
```

- To navigate to the build directory:

```
cd build
```

- To configure the build using ccmake:

```
ccmake ..
```

The above command opens the ccmake GUI. Press 'c' to configure and 't' to toggle advanced mode. We now modify the necessary variables in the ccmake GUI. We will set CMAKE_INSTALL_PREFIX to a local directory where we want to install the OpenMM engine (e.g., */home/USERNAME/bin/openmm*). If the directory does not exist, we will create it. Alternatively, we can leave this variable the default if we have sudo privileges and want to install OpenMM globally. We will then review and modify any other variables if needed. After modifying the variables, we press 'c' to configure. If any issues arise, ccmake will notify us. Once the configuration is successful, we press 'g' to generate the build files. The ccmake GUI will then close automatically.

- To install OpenMM, execute the following commands in the *build* directory:

```
make
```

```
make install
```

```
make PythonInstall
```

- To test the successful installation of OpenMM:

```
python -m openmm.testInstallation
```

The above command will run a series of tests to ensure OpenMM is installed correctly on our system.

We have now installed OpenMM and its plugins from the source on our local machine.

https://doi.org/https://doi.org/10.33011/livecoms.5.1.2359
Living J. Comp. Mol. Sci. 2024, 5(1), 2359

4.5.2 Installing SEEKR2-OpenMM plugin from source

Once the OpenMM engine is installed, we will install the SEEKR2 plugin on top of the installed version of OpenMM. It is recommended, though not required, to install the plugin in the home directory.

- If not already activated, we first activate the SEEKR2 conda environment.

```
conda activate SEEKR2
```

- It is recommended, though not required, to install the SEEKR2 plugin in the home directory. So, we navigate to the home directory.

```
cd ~
```

- To clone the SEEKR2-OpenMM plugin repository:

```
git clone https://github.com/seekrcentral/seekr2_openmm_plugin.git
```

- We then navigate to into the *seekr2_openmm_plugin* directory:

```
cd seekr2_openmm_plugin/seekr2plugin
```

- Create a *build* directory:

```
mkdir build
```

- To navigate to the build directory:

```
cd build
```

- To configure the build using ccmake:

```
ccmake ..
```

The above command opens the ccmake GUI. Press 'c' to configure. We now modify the necessary variables in the ccmake GUI. We will set CMAKE_INSTALL_PREFIX to the directory similar to the CMAKE_INSTALL_PREFIX as set during the OpenMM installation (e.g., */home/USERNAME/bin/openmm*). We will also set the OPENMM_DIR to the directory similar to the CMAKE_INSTALL_PREFIX as set during the installation of OpenMM engine (e.g., */home/USERNAME/bin/openmm*). We will set the SEEKR2_BUILD_OPENCL_LIB to OFF. After modifying the variables, we press 'c' to configure. If any issues arise, ccmake will notify us. Once the configuration is successful, we then press 'g' to generate the build files. The ccmake GUI will then close automatically.

- To install the SEEKR2 plugin, we execute the following commands in the *build* directory:

```
make
```

```
make install
```

```
make PythonInstall
```

- To ensure proper installation of the SEEKR2 plugin, we execute the following command in the terminal:

```
make test
```

4.5.3 Installing SEEKR2

Please refer to section 4.4.2 for detailed instructions to install SEEKR2. However, for the sake of completion, we execute the following commands to install SEEKR2.

```
conda activate SEEKR2
# Activate SEEKR2 environment, if not activated
```

```
cd ~
# Navigate to home directory (recommended)
```

```
git clone https://github.com/seekrcentral/seekr2.git
# Clone the SEEKR2 repository
```

```
cd seekr2
# Navigate to the seekr2 directory
```

```
python -m pip install .
# Install SEEKR2
```

```
pytest
# Run tests to check successful installation
```

4.5.4 Installing Seekrtools

Seekrtools is a suite of software utilities designed to work with SEEKR2 applications, primarily SEEKR2, to streamline the preparation process and execution of multiscale milestoning simulations. For more comprehensive instructions and tutorials, please refer to the official documentation at https://seekrtools.readthedocs.io/en/latest. Ensuring that SEEKR2 and OpenMM packages are installed before installing Seekrtools is crucial, as most programs within Seekrtools rely on these packages.

- If not already activated, we first activate the SEEKR2 conda environment.

```
conda activate SEEKR2
```


https://doi.org/https://doi.org/10.33011/livecoms.5.1.2359
Living J. Comp. Mol. Sci. 2024, 5(1), 2359

- It is recommended, though not required, to install the seekrtools program in the home directory. So, we navigate to the home directory.

```
cd ~
```

- We will clone the seekrtools Python API repository by running the following command in the terminal:

```
git clone https://github.com/seekrcentral/seekrtools.git
```

- We then proceed with the installation of seekrtools Python API in the cloned *seekrtools* directory:

```
cd seekrtools
```

```
python -m pip install .
```

5 SEEKR2 tutorials

We will explore the three distinct stages of a SEEKR2 calculation: prepare, run, and analyze. Every stage is required to obtain estimates of the kinetics and thermodynamics of a system. This tutorial assumes that we use a computer with one or more graphical processing units (GPUs). If the computer does not have a GPU, we ought to transfer all files to a computer equipped with a GPU (and OpenMM, with the SEEKR2 OpenMM Plugin and SEEKR2 Python API installed) to run MD simulations.

To begin with the SEEKR2 tutorials, we first activate the SEEKR2 conda environment, if not already activated.

```
conda activate SEEKR2
```

We download the SEEKR_tutorials repository by executing the following command in the terminal:

```
git clone https://github.com/anandojha/SEEKR_tutorials
```

Within the *working_examples* directory of the *SEEKR_tutorials* repository, there exist three subdirectories, i.e., *beta_cyclodextrin_guest_complexes*, *trypsin_benzamidine_complex*, and *simulation_data* (Outline 1). The *beta_cyclodextrin_guest_complexes* and the *trypsin_benzamidine_complex* directories contain all the necessary files for running SEEKR2 simulations for receptor-ligand complexes. The *simulation_data* directory contains reference SEEKR2 simulations offering users to perform post-simulation analyses. To start with the tutorials, we now navigate to the *working_examples* directory:

```
cd ~/SEEKR_tutorials/SEEKR_tutorials/
cd working_examples
```

The *working_examples* directory contains the SEEKR2 tutorials on biomolecular complexes, i.e., the seven β-cyclodextrin-guest complexes (Figure 4) and the trypsin-benzamidine complex (Figure 6). One of the subdirectories, *beta_cyclodextrin_guest_complexes*, contains the seven host-guest complexes where necessary files and scripts are located for each of the complexes to set up the SEEKR2 calculation, run MD and BD simulations, and analyze SEEKR2 simulations to calculate thermodynamic and kinetic quantities of interest. The tutorial will go through one of the host-guest complexes, i.e., the BCD-1-butanol complex. The user may follow the same instructions to get started with the other six host-guest complexes within the *beta_cyclodextrin_guest_complexes* directory. Similarly, the other subdirectory, *trypsin_benzamidine_complex* contains the files and scripts to parameterize the receptor-ligand complex, set up the SEEKR2 simulations, and perform the post-SEEKR2 analysis.

Outline 1: Overview of the working_examples directory within SEEKR_tutorials directory

```
working_examples
├─ beta_cyclodextrin_guest_complexes
│   ├─ BCD_1-butanol
│   ├─ BCD_1-napthylethanol
│   ├─ BCD_1-propanol
│   ├─ BCD_2-napthylethanol
│   ├─ BCD_aspirin
│   ├─ BCD_methyl_butyrate
│   └─ BCD_terbutanol
├─ simulation_data
│   ├─ beta_cyclodextrin_guest_complexes
│   └─ trypsin_benzamidine_complex
└─ trypsin_benzamidine_complex
```

5.1 Basic Tutorial: β-cyclodextrin (host)-guest complexes

https://doi.org/https://doi.org/10.33011/livecoms.5.1.2359
Living J. Comp. Mol. Sci. 2024, 5(1), 2359

Stage 1: Prepare

The first stage of a SEEKR2 calculation is the preparation phase. This stage involves setting up the necessary input files and defining parameters for the simulation. Let us subdivide the preparation phase further into three stages:

1. Obtaining the structure and the parameter file for the host-guest complex

If not already activated, we first activate the SEEKR2 conda environment.

```
conda activate SEEKR2
```

We start with a structure (PDB) file for the host-guest complex. We then prepare a force field parameter file specific to the complex. In this tutorial, we have taken the initiative to provide the parameter file for the receptor-ligand complex. We assume that the users have prior experience with force field files for receptor complexes. Let us begin with the *hostguest.pdb* and *hostguest.parm7* files.

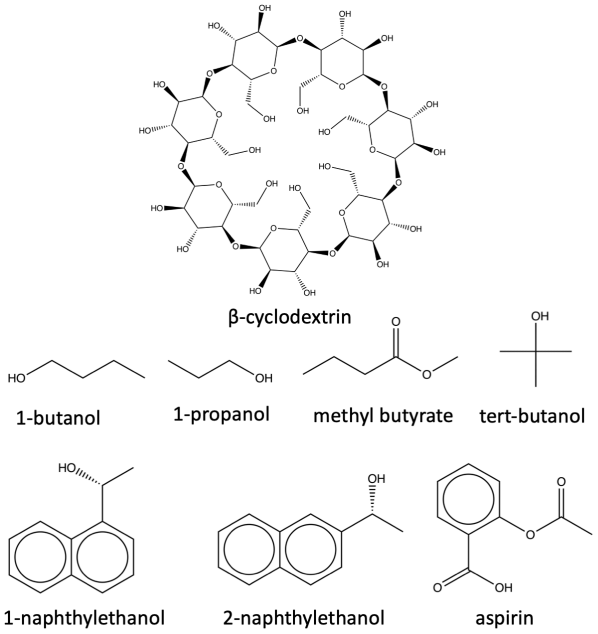


Figure 4. β-cyclodextrin (host) and the seven ligand (guest) molecules

Outline 2 shows the files we will require for the successful completion of the SEEKR2 tutorial on the host-guest complexes.

2. Preparing PQR files for Brownian dynamics simulations

Given the *hostguest.pdb* file containing the receptor (β-cyclodextrin) and the ligand molecule, we will obtain separate PQR files for the receptor and ligand molecule, respectively. These files are required to run BD simulations to calculate the receptor-ligand binding rates (k_{on}). To obtain the PQR files, we will use the ambpdb program integrated into the Ambertools. The ambpdb program requires a topology file (*hostguest.parm7*) and a coordinate file (*hostguest.inpcrd*) to create a PQR file (*hostguest.pqr*). The PQR file is almost identical to a PDB file, except the charge and radius columns in the PQR files substitute the beta and occupancy columns in the PDB file. Once a *hostguest.pqr* file is created using the ambpdb tool, we want to create two separate PQR files from the *hostguest.pqr* file, i.e., the *receptor.pqr* file containing the coordinate, charge, and radius information of the receptor atoms and the *ligand.pqr* file containing the coordinate, charge, and radius information of the ligand atoms.

The python script, *create_BD_files.py* generates a coordinate file, i.e., *hostguest.inpcrd* using the CPPTRAJ module with the structure and topology files as input. The script then further creates *receptor.pqr* and *ligand.pqr* files by reading into the *hostguest.pqr* file. To achieve this, we execute the following command in the terminal:

```
python create_BD_files.py
```

Outline 2: Overview of the beta_cyclodextrin_guest_complexes directory within working_examples directory

```
beta_cyclodextrin_guest_complexes
├─ BCD_1-butanol
│  ├─ hostguest.pdb
│  ├─ hostguest.parm7
│  ├─ create_BD_files.py
│  └─ input_SMD_HIDR.xml
├─ BCD_1-napthylethanol
├─ BCD_1-propanol
├─ BCD_2-napthylethanol
├─ BCD_aspirin
├─ BCD_methyl_butyrate
└─ BCD_terbutanol
```

Once the coordinate file is generated, the script then creates a PQR file, i.e., *hostguest.pqr* by executing the following command in the terminal:

https://doi.org/https://doi.org/10.33011/livecoms.5.1.2359
Living J. Comp. Mol. Sci. 2024, 5(1), 2359

```
ambpdb -p hostguest.parm7 -c hostguest.inpcrd \
-pqr > hostguest.pqr
```

The Browndye2 software lumps all the charges of the residue into one point charge, which may be acceptable for a protein, as in the case of the *receptor.pqr* file where all the atoms of a particular residue have the same numbering. But for a small ligand, the accuracy can be further improved if we consider each atom as a point charge. This is accomplished by renumbering the atoms of the ligand, each with a different number. To achieve this, we execute the following command in the terminal:

```
python ~/$PWD/scripts/pqr_resid_for_each_atom.py \
ligand.pqr ligand.pqr
# where $PWD is seekrtools/seekrtools/scripts
```

Now, we have the required files to proceed to the next step.

3. Running Steered MD simulations to obtain starting structures for SEEKR2 simulations

Determining the anchor points for the host-guest complex is the first step towards milestoning simulations in the SEEKR2 framework. We first choose an appropriate collective variable (CV) to determine the anchor points that could describe the receptor-ligand binding and unbinding dynamics. In the case of the host-guest complex, we choose the distance between the center of mass (COM) of the receptor (β-cyclodextrin) and the ligand as the CV to determine the anchor points. The anchor points, therefore, lie on a line composed of the COM-COM distance. Once the anchor points are determined, we define concentric spherical milestones around the receptor-ligand binding site where the concentric radii are midpoints of any two consecutive anchor points. These milestones will act as reference points during the simulation. We then use steered MD (SMD) to slowly pull the ligand away from the binding site with a harmonic restraint and save the trajectory snapshots as it crosses the anchor point while slowly moving out of the binding pocket. Once the ligand is pulled out completely, we have the saved structure files, which will be used to create a SEEKR2 file tree where SEEKR2 simulations will occur.

Holo insertion by directed restraints (HIDR), a computational method employed in the SEEKR2 framework, utilizes one or more initial configurations alongside a SEEKR2 input XML file, pulls the system towards all the anchors present in the SEEKR2 framework until starting structures exist in all of them. The HIDR algorithm employs SMD simulations to accomplish this.

The HIDR program needs a *model.xml* file to run the SMD simulations, so we run the *prepare.py* script on a model input file. The model input XML file for SEEKR2 calculations contains various parameters and settings that define the configuration and behavior of the calculation (Summary 1). It should be noted that the <md_steps_per_anchor> tag defines the number of MD simulation steps per anchor, and for tutorial purposes, the simulation steps in each anchor are reduced compared to our original study. The <root_directory> tag has to be explicitly defined by the user in the *input_SMD_HIDR.xml* file. To run the prepare.py script on the model input file, we execute the following command in the terminal:

```
python ~/seekr2/seekr2/prepare.py \
input_SMD_HIDR.xml
```

Summary 1: Overview of sections and tags within the XML file

- **<calculation_type>**: Specifies the type of milestoning model to employ. SEEKR2 either employs the "mmvt" (Markovian milestoning with Voronoi Tesselations) or the "elber" model for the original Elber milestoning method.
- **<calculation_settings>**: A block of settings specific to the chosen calculation type. The tags within this section vary depending on the calculation type.
- **<md_output_interval>**: Interval between outputs of simulation state information, trajectory frames, and restart checkpoints (in MD timesteps).
- **<md_steps_per_anchor>**: Total number of MD timesteps to be run per anchor.
- **<temperature>**: Temperature (in Kelvin) to be used for all stages of the calculation.
- **<pressure>**: Pressure (in bar) for simulations if the ensemble is set to npt. Ignored if the ensemble is set to nvt.
- **<ensemble>**: Defines the ensemble of MD simulations. Options include nvt (constant volume and temperature) and npt (constant pressure and temperature).
- **<root_directory>**: Filesystem path to the directory where the calculation files will be written.
- **<md_program>**: Specifies the MD simulation engine to use. The user has the option to choose either openmm or namd.
- **<constraints>**: Specifies the type of bond and angle constraints in the MD simulation.
- **<rigid_water>**: Specifies whether water molecules will have a rigid angle.

https://doi.org/https://doi.org/10.33011/livecoms.5.1.2359
Living J. Comp. Mol. Sci. 2024, 5(1), 2359

- **<hydrogen_mass>**: Mass (in AMU) to use for hydrogen mass repartitioning (HMR).
- **<integrator_type>**: Type of integrator to be used for simulation dynamics.
- **<timestep>**: MD timestep (in ps).
- **nonbonded_cutoff**: Nonbonded cutoff (in nm) for MD simulations.
- **<cv_inputs>**: Settings defining the collective variables (CVs) and their associated anchors and milestones.
- **<cv_input>**: Input structure for a CV, with the CV type defined by the class attribute.
- **<group>**: Lists of atom indices for CV functions involving centers of masses of groups of molecules.
- **<bd_group>**: Atoms within PQR files used in BD simulations to define the CV.
- **<input_anchors>**: Block of input anchors used to construct model anchors.
- **<input_anchor>**: Input for an anchor, with the class attribute matching a particular CV. Attributes depend on the anchor type, such as <starting_amber_params>, <radius>, <lower_milestone_radius>, <upper_milestone_radius>, etc.
- **<browndye_settings_input>** (optional): Settings for Browndye simulations and k_{on} calculations.
- **<binary_directory>**: Directory containing Browndye2 programs
- **<receptor_pqr_filename>**: Path to the PQR file representing the receptor.
- **<ligand_pqr_filename>**: Path to the PQR file representing the ligand.
- **<apbs_grid_spacing>**: Grid spacing (in Å) for APBS calculations.
- **<receptor_indices**>: Atom indices defining the binding site in the receptor PQR file.
- **<ligand_indices>**: Atom indices defining the center of the ligand molecule in the ligand PQR file.
- **<ions>**: Block of ion objects used in APBS and BD calculations.
- **<ion>**: Object representing an ion with attributes like <radius>, <charge>, and <conc>.
- **<num_b_surface_trajectories>**: Total number of trajectories for b-surface simulations.
- **<n_threads**>: Number of CPUs to use in Browndye2 calculations.

Now the model XML file and the SEEKR2 file tree have been generated in a separate *SEEKR_simulation* directory as specified explicitly by the <root_directory> tag in the model input XML file, i.e., the *input_SMD_HIDR.xml* file. HIDR will now employ SMD simulations to gradually pull the system into every anchor and save the structures for subsequent SEEKR2 calculations. It is important to note that HIDR offers alternative approaches, such as random acceleration MD (RAMD) and ratcheting, to populate starting structures. For detailed instructions on utilizing these alternative methods, it is recommended to refer to the HIDR documentation and seekrtools tutorials. To run SMD simulations with the HIDR algorithm, execute the following command in the terminal:

```
python ~/seekrtools/seekrtools/hidr/hidr.py \
any SEEKR_simulation/model.xml -M SMD -p \
hostguest.pdb
```

This command is likely to run for hours or days, depending on the speed of the GPU. We can obtain a comprehensive overview of HIDR arguments by executing HIDR with the -h argument using the following python command:

```
python ~/seekrtools/seekrtools/hidr/hidr.py -h
```

Several important options are available, such as specifying the number of equilibration steps to be executed before SMD simulations. For example, including the argument below will instruct HIDR to perform 5,000,000 equilibration steps (10 ns) before initiating any SMD simulations:

```
python ~/seekrtools/seekrtools/hidr/hidr.py \
any SEEKR_simulation/model.xml -M SMD -p \
hostguest.pdb -e 5000000
```

Additionally, we can allow for some equilibration steps within each anchor after the SMD simulation has reached that particular anchor. These equilibration steps are referred to as settling steps in HIDR. To specify the number of settling steps, the -S argument is used. To allow for 200,000 settling steps (0.2 ns), we execute the following command in the terminal:

```
python ~/seekrtools/seekrtools/hidr/hidr.py \
any SEEKR_simulation/model.xml -M SMD -p \
hostguest.pdb -S 100000
```

By default, HIDR SMD simulations move the system towards each anchor at an approximate 0.01 nm/ns speed. This speed is designated to let the system reach each anchor within a reasonable time frame while avoiding excessive perturbations to the system. However, the speed can be adjusted using the -v argument. For instance, if the user desires to perform SMD simulations ten times faster (thus

https://doi.org/https://doi.org/10.33011/livecoms.5.1.2359
Living J. Comp. Mol. Sci. 2024, 5(1), 2359

completing in one-tenth of the time), the speed can be set to 0.1 nm/ns:

```
python ~/seekrtools/seekrtools/hidr/hidr.py \
any SEEKR_simulation/model.xml -M SMD -p \
hostguest.pdb -v 0.1
```

To keep the directory clean, we can optionally choose to delete the intermediate files:

```
rm hostguest.inpcrd hostguest.pqr ligand.pqr \
receptor.pqr
```

Stage 2: Run

During the prepare stage of a SEEKR2 calculation, a file tree containing all the necessary files and directories is generated at the specified location indicated by the <root_directory> tag in the model input file. Once the preparation stage is complete, we move on to the run stage.

The run stage involves executing the simulations based on the files and directories from the prepare stage. Inside this directory, i.e., *SEEKR_simulation*, a *model.xml* file exists. In the subsequent stages of SEEKR2, especially the run stage and beyond, the path to the *model.xml* file is used as an argument in most SEEKR2 programs. It is important to note that modifying the *model.xml* file directly without re-running the *prepare.py* script is not recommended. To start the run stage, we will use the *run.py* script. The following command launches the script to run MD simulations within the SEEKR2 milestones:

```
python ~/seekr2/seekr2/run.py any \
SEEKR_simulation/model.xml
```

In the above command, the word "any" is the instruction argument for the *run.py* script. It instructs the script to run any unfinished MD or BD simulations. We can use "any_md" or "any_bd" as an instruction input to run only unfinished MD or BD simulations. Please refer to the SEEKR2 documentation for a comprehensive list of available instruction inputs as *run.py* arguments.

Once we initiate the *run.py* script, simulations will run until completion or interruption. The SEEKR2 framework saves checkpoints for both MD and BD simulations, allowing us to resume the calculation from where it was interrupted. To track the progress and convergence of the simulations, we use the *converge.py* script:

```
python ~/seekr2/seekr2/converge.py any \
SEEKR_simulation/model.xml
```

Running the *converge.py* script generates convergence plots, and images are saved in the *plots_and_images* subfolder within the <root_directory>. For additional arguments that can be used with both *run.py* and *converge.py*, we can run either script with the -h argument (Summary 2).

Summary 2: Arguments for converge.py

Positional Arguments

- **MODEL_FILE**: This argument specifies the name of the model file for the SEEKR2 calculation. We need to replace `MODEL_FILE` with the model file name, i.e., *model.xml* used in the calculation.

Optional Arguments

- **-s K_ON_STATE**: This argument allows us to define the bound state used to compute the k_{on} value. If we want to specify a particular bound state, we include the -s option followed by the state name.
- **-d IMAGE_DIRECTORY**: Using this argument, we can define the directory where plots and images will be saved. If we want to specify a different directory, we include the -d option followed by the desired directory path. By default, all the plots will be saved to the *images_and_plots* directory.
- **-c CUTOFF**: This argument sets the minimum convergence value required to conclude that the calculations have converged for a given anchor. The default value is 0.1, but we can specify a different cutoff value by including the -c option followed by the desired value.
- **-m MINIMUM_ANCHOR_TRANSITIONS**: Using this argument, we can set the minimum number of transitions that must be observed per milestone in a given anchor as a criterion for SEEKR2 simulations. The default value is 100, but we can specify a different value by including the -m option followed by the desired number.
- **-l, –long_converge**: This argument determines whether to run a complete convergence analysis. Including the -l flag in the command will enable the extended convergence analysis. By default, this is set to False.

Stage 3: Analyze

The final stage within the SEEKR2 framework involves analyzing the results obtained from the simulations. This stage enables the construction of kinetics and thermodynamics profiles for the studied process.

https://doi.org/https://doi.org/10.33011/livecoms.5.1.2359
Living J. Comp. Mol. Sci. 2024, 5(1), 2359

Summary 3: Arguments for analyze.py

Positional Arguments

- **MODEL_FILE**: This argument specifies the name of the model file for the SEEKR2 calculation. We need to replace `MODEL_FILE` with the model file name, i.e., *model.xml* used in the calculation.

Optional Arguments

- **-f, –force_warning**: By default, missing statistics for any anchors will generate fatal errors. This option will instead raise a warning and attempt the calculation anyway.
- **-n NUM_ERROR_SAMPLES**: This argument specifies the number of error samples to generate for estimating the error/uncertainty of computed values. The default value is 100.
- **-S STRIDE_ERROR_SAMPLES**: This argument specifies the number of strides between saved error samples. An argument of *None* automatically assigns the quantity at the number of milestones in the model squared. The default value is None.
- **-K SKIP_ERROR_SAMPLES, –skip_error_samples SKIP_ERROR_SAMPLES**: This argument specifies the number of error samples to skip before using them. An argument of None automatically assigns the quantity at ten times the number of milestones in the model squared. The default value is None.
- **-d IMAGE_DIRECTORY**: By using this argument, we can define the directory where plots and images will be saved. If we want to specify a different directory, we include the -d option followed by the desired directory path. By default, all the plots will be saved to the *images_and_plots* directory.
- **-s, –skip_checks**: By default, post-simulation checks will be run before the analysis is started, and if the checks fail, the analysis will not proceed. This argument bypasses those checks and allows the analysis to proceed anyways.
- **-t MINIMUM_TIME**: A user may wish to skip a simulation time for each anchor before counting the transitions for milestoning analysis. When performing analysis, we enter the time (in ps) to skip a portion of the production simulations.
- **-T MAXIMUM_TIME, –maximum_time MAXIMUM_TIME**: A user may wish to stop the analysis of simulation time for each anchor at a particular time. We enter the time (in ps) to end the analysis at a given anchor if the simulation time exceeds it.

To execute the post-simulation SEEKR2 analysis:

```
python ~/seekr2/seekr2/analyze.py \
SEEKR_simulation/model.xml
```

The *analyze.py* script takes the *model.xml* file as an argument. It constructs the milestoning model, populates it with transition probabilities and simulation times within each milestone, and computes error margins. For a list of arguments that can be used with *analyze.py*, we can run the script with the -h argument (Summary 3). Figure 5 shows the free energy profile per milestone for the BCD-1-butanol complex, obtained by executing the *analyze.py* script.

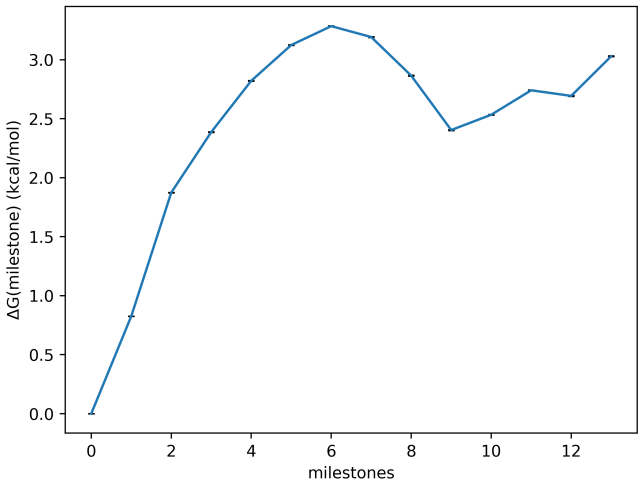


Figure 5. Free energy profile per milestone (ΔG_i) obtained from the SEEKR2 milestoning method for the β-cyclodextrin (host) and the 1-butanol (guest) complex.

5.2 Advanced Tutorial: Trypsin-benzamidine complex

Stage 1: Prepare

1. Obtaining the structure and the parameter file for the trypsin-benzamidine complex

This section will prepare input files for the SEEKR2 simulation to determine the (un)binding kinetics between the receptor protein, trypsin, and the ligand molecule, benzamidine. The Amber molecular mechanics forcefield is used to parameterize the receptor-ligand complex. This tutorial assumes the user has some familiarity with AmberTools. If the user is new to AmberTools, we recommend visiting the Amber tutorials page (https://ambermd.org/tutorials/) or reviewing the Amber manual (https://ambermd.org/Manuals.php) for a comprehensive understanding. Outline 3 shows the files we will require

https://doi.org/https://doi.org/10.33011/livecoms.5.1.2359
Living J. Comp. Mol. Sci. 2024, 5(1), 2359

for the successful completion of the SEEKR2 tutorial on the trypsin-benzamidine complex.

(i) Identify the ligand molecule and the receptor protein in the receptor-ligand complex

The first step is to identify the molecules in the receptor-ligand complex. In this case, we have trypsin as the target molecule and benzamidine as the ligand molecule. The trypsin-benzamidine complex is a standard complex for studying binding and unbinding kinetics.

(ii) Parameterization of the receptor-ligand complex

In the SEEKR2 framework, to estimate the binding and unbinding of receptor-ligand bimolecular complexes, one of the molecules is the ligand that needs to be parameterized separately. One method to accomplish this is to use the Antechamber program in AmberTools. Antechamber requires a structure file specific to the small molecule to perform the parameterization. In our case, we will utilize a PDB file that contains only the benzamidine structure, which has been parametrized using a semi-empirical method and the generalized AMBER force field. Although semi-empirical methods for assigning charges are quick and convenient, they may not provide the most accurate results for assigning partial charges to a molecule. It is advisable to explore more precise levels of quantum calculations to obtain partial charges, such as Hartree Fock with Density Function Theory (HF-DFT) or Møller-Plesset 2 (MP2) calculations within a charge model such as RESP or CHELPG. However, these calculations involving higher levels of quantum theory require dedicated quantum calculation software like Gaussian, GAMESS, ORCA, etc. Incorporating parameters from quantum calculation software is a complex topic that is beyond the scope of this tutorial.

Outline 3: Overview of the trypsin_benzamidine_complex directory within working_examples directory

```
trypsin_benzamidine_complex
├─ hostguest.pdb
├─ minimize_equilibriate.py
├─ extract_benzamidine.py
├─ create_BD_files.py
├─ parameterize_trypsin_benzamidine.tleap
├─ save_benzamidine_lib.tleap
└─ input_SMD_HDR.xml
```

To save a separate PDB structure of the ligand molecule (*benzamidine.pdb*) from the given receptor-ligand complex (*trypsin_benzamidine_init.pdb*), we execute the following command in the terminal.

```
python extract_benzamidine.py
```

We use the Antechamber program, part of AmberTools to parameterize the benzamidine molecule. The antechamber command takes several arguments to specify input and output files, formats, and parameters (Summary 4). We will execute the following command in the terminal to parameterize the benzamidine molecule.

```
antechamber -i benzamidine.pdb -fi pdb -bk BEN \
-o benzamidine.mol2 -fo mol2 -c bcc -nc 1
```

We will now generate a parameter modification (frcmod) file containing the molecular force field parameters for the benzamidine molecule. This file is later used in the LEAP functionality of the Ambertools to parameterize the receptor-ligand complex. The ParmChk2 program generates the frcmod file for the benzamidine molecule. We will execute the following command in the terminal to generate the frcmod file.

```
parmchk2 -i benzamidine.mol2 -f mol2 -o \
benzamidine.frcmod
```

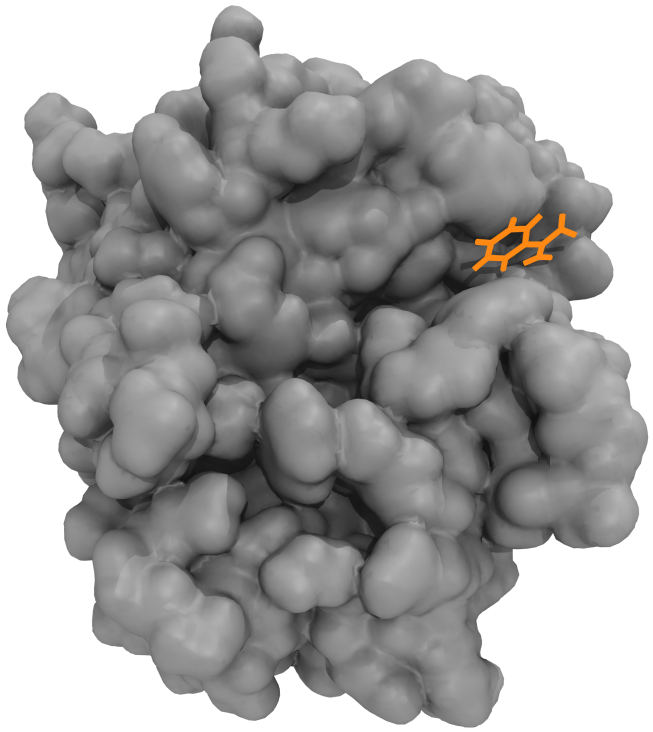


Figure 6. Trypsin - benzamidine complex

https://doi.org/https://doi.org/10.33011/livecoms.5.1.2359
Living J. Comp. Mol. Sci. 2024, 5(1), 2359

We generate a .lib file, a library file of forcefield parameters for the benzamidine molecule, by executing the following command in the terminal:

```
tleap -f save_benzamidine_lib.tleap
```

The final step in parameterizing the trypsin-benzamidine complex involves solvating the system with water molecules and applying periodic boundary conditions. This ensures that the simulation accurately represents the behavior of the complex in an aqueous environment while accounting for long-range interactions and avoiding edge effects. Additionally, counterions are added to neutralize any net charge in the system, creating an electrically neutral simulation system. We execute the following command in the terminal to parameterize the trypsin-benzamidine complex:

```
tleap -f parameterize_trypsin_benzamidine.tleap
```

Summary 4: Arguments used in the antechamber parameterization of the ligand

- `-i benzamidine.pdb`: Take the `ben.pdb` file as input.
- `-fi pdb`: The format of `benzamidine.pdb` is in PDB format.
- `-bk BEN`: The component/block ID for benzamidine in the PDB file is `BEN`.
- `-o benzamidine.mol2`: Specifies the output file name of the benzamidine molecule.
- `-fo mol2`: Outputs the `benzamidine.mol2` file in MOL2 format.
- `-c bcc`: Uses the AM1-BCC semi-empirical method to assign partial charges of the atoms.
- `-nc 1`: The molecule has a net molecular charge of +1 due to its protonation state in aqueous environments at pH = 7.

In the *parameterize_trypsin_benzamidine.tleap* file, explanations for the commands are as under:

- `source leaprc.protein.ff14SB`: Sourcing the ff14SB forcefield, a recently generated force field parameter set known for its good performance in molecular simulations of proteins, provides accurate descriptions of protein atoms and their interactions.
- `source leaprc.water.tip4pew`: By sourcing the leaprc.water.tip4pew file, the TIP4Pew water model is employed. This water model is considered more accurate than the commonly used TIP3P model.
- `solvateoct receptor_ligand_complex TIP4PEWBOX 8`: This command solvates the receptor-ligand complex, represented by the variable receptor_ligand_complex, in a truncated octahedral box using TIP4Pew water molecules. The truncated octahedral shape is preferred over a simple cubic box due to its more efficient use of space, reducing the number of water molecules required to solvate the system. The specified padding of 8 Å creates a buffer region around the protein to prevent any artifacts from periodic boundary conditions.
- `addIons2 mol Cl- 7`: Since the system has a net positive charge, adding chloride ions (Cl-) neutralizes the overall charge. When setting up simulations, it is generally recommended to carefully consider the ions present in the experimental or physiological conditions to mimic the real environment closely. However, in this specific scenario, the decision to use only chloride ions is based on the information available and the challenges associated with including other ions.

To remove multiple output files generated by the antechamber and leap commands, which are no longer required for subsequent parameterization steps, we can execute the following command in the terminal:

```
rm *ANTECHAMBER* ATOMTYPE.INF *sqm* leap.log \
benzamidine.frcmod benzamidine.lib \
benzamidine.mol2 benzamidine.pdb
```

The next step involves the energy minimization of the trypsin-benzamidine complex to its local minima, followed by equilibration in the NVT ensemble. System minimization aims to find an energetically stable conformation of the system by iteratively adjusting the atomic positions to minimize the potential energy, therefore beginning with a favorable starting structure. Equilibration prepares the biomolecular complexes for MD production runs by resolving initial structural and energetic irregularities, ensuring stable and reliable simulations under target conditions. To perform the energy minimization followed by system equilibration, we execute the following command in the terminal:

```
python minimize_equilibriate.py
```

2. Preparing PQR files for Brownian dynamics simulations

Given the *trypsin_benzamidine.pdb* file containing the receptor (trypsin) protein and the ligand (benzamidine) molecule, we will obtain separate PQR files for the receptor and ligand molecule, respectively. The ambpdb program requires a topology file (*trypsin_benzamidine.prmtop*) and a coordinate file (*trypsin_benzamidine.inpcrd*) to create a PQR file (*trypsin_benzamidine.pqr*). Once a *trypsin_benzamidine.pqr*

https://doi.org/https://doi.org/10.33011/livecoms.5.1.2359
Living J. Comp. Mol. Sci. 2024, 5(1), 2359

file is created using the ambpdb tool, we would want to create two separate PQR files from the *trypsin_benzamidine.pqr* file, i.e., the *trypsin.pqr* file containing the coordinate, charge, and radius information of the receptor atoms and the *benzamidine.pqr* file containing the coordinate, charge, and radius information of the ligand atoms.

The python script, create_BD_files.py generates a coordinate file, i.e., *trypsin_benzamidine.inpcrd* using the CPPTRAJ module with the structure and topology files as input. Once the coordinate file is generated, the script then creates a PQR file, i.e., *trypsin_benzamidine.pqr* by executing the following command:

```
ambpdb -p trypsin_benzamidine.prmtop -c \
trypsin_benzamidine.inpcrd -pqr > \
trypsin_benzamidine.pqr
```

The script then further creates *trypsin.pqr* and *benzamidine.pqr* files by reading into the *trypsin_benzamidine.pqr* file. To achieve this, we execute the following command in the terminal:

```
python create_BD_files.py
```

While the Browndye2 software consolidates all residue charges into a single point charge, suitable for proteins like in the case of *trypsin.pqr* file with uniformly numbered atoms within a residue. But for small ligands, accuracy is enhanced by treating each atom as a distinct point charge and renumbering them individually. To achieve this, we execute the following command in the terminal:

```
python ~/$PWD/scripts/pqr_resid_for_each_atom.py \
benzamidine.pqr benzamidine.pqr
# where $PWD is seekrtools/seekrtools/scripts
```

Now, we have the required files to proceed to the next step.

3. Running Steered MD simulations to obtain starting structures for SEEKR2 simulations

To determine the anchor points for milestoning simulations in the SEEKR2 framework, we choose the appropriate CV for the trypsin-benzamidine complex. Here, the distance between the center of mass (COM) of the binding site of the trypsin protein and the ligand serves the CV, followed by determining the anchor points (Figure 7). Subsequently, we establish concentric spherical milestones around the binding site, using midpoints between successive anchor points as their radii. Through SMD simulations, the benzamidine ligand is slowly extracted from the binding site, capturing trajectory snapshots at each anchor point. Once the ligand is pulled out entirely and we have the saved structure files, a SEEKR2 file tree is created where SEEKR2 simulations will occur.

As mentioned previously, the HIDR program requires a *model.xml* file to run the SMD simulations, so we run the *prepare.py* script on a model input file. It should be noted that the <md_steps_per_anchor> tag defines the number of MD simulation steps per anchor, and for tutorial purposes, the simulation steps in each anchor are reduced. The <root_directory> tag has to be explicitly defined by the user in the *input_SMD_HIDR.xml* file. To run the *prepare.py* script on the model input file, we execute the following command in the terminal:

```
python ~/seekr2/seekr2/prepare.py \
input_SMD_HIDR.xml
```

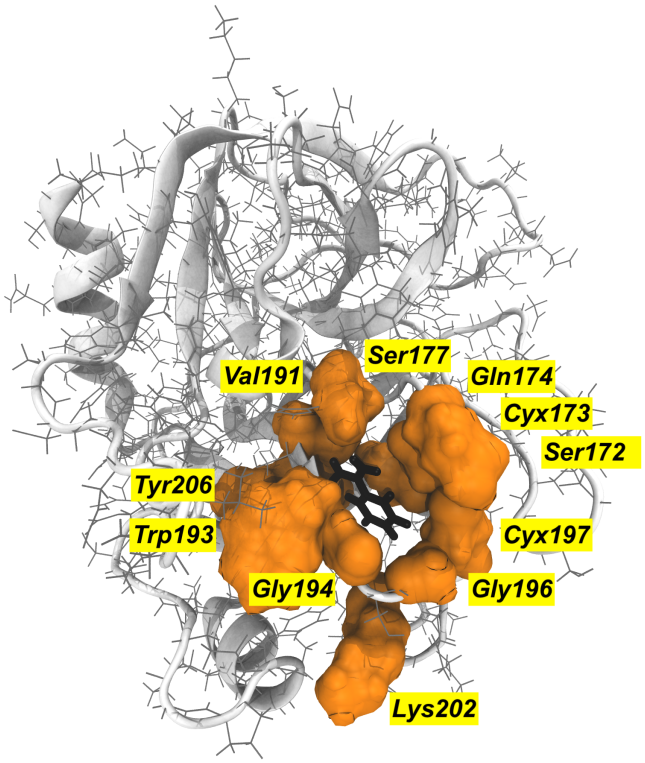


Figure 7. Trypsin-benzamidine complex with the benzamidine molecule (in black) and surrounding residues constituting the binding site (in orange). The distance between the center of mass of the heavy atoms of the benzamidine molecule and the α-carbons of the binding site residues constitute the collective variable.

Now the model XML file and the SEEKR2 file tree have been generated in a separate *SEEKR_simulation* directory as specified explicitly by the <root_directory> tag in the model input XML file, i.e., *the input_SMD_HIDR.xml file*. HIDR will now employ SMD simulations to gradually pull the system into every anchor and save the structures for subsequent SEEKR2 cal-

https://doi.org/https://doi.org/10.33011/livecoms.5.1.2359
Living J. Comp. Mol. Sci. 2024, 5(1), 2359

culations. To run SMD simulations with the HIDR algorithm, execute the following command in the terminal:

```
python ~/seekrtools/seekrtools/hidr/hidr.py \
any SEEKR_simulation/model.xml -M SMD -p \
trypsin_benzamidine.pdb
```

This command is likely to run for hours or days, depending on the speed of the GPU. We can obtain a comprehensive overview of HIDR arguments by executing HIDR with the -h argument using the following python command:

```
python ~/seekrtools/seekrtools/hidr/hidr.py -h
```

Several options in the HIDR program can be specified (described earlier in section 5.1), such as determining the equilibration steps before SMD simulations, allowing for some equilibration steps (settling steps) within each anchor after the SMD simulation has reached that particular anchor, and adjusting the speed of HIDR simulations.
To keep the directory clean, we can optionally choose to delete the intermediate files:

```
rm trypsin_benzamidine.inpcrd trypsin.pqr \
trypsin_benzamidine.pqr benzamidine.pqr
```

Stage 2: Run

During the prepare stage of a SEEKR2 calculation, a file tree with essential files and directories is created at the location specified by the <root_directory> tag in the model input file. The run phase begins, executing SEEKR2 simulations based on the files and directories from the prepare stage. Within the main directory (*SEEKR_simulation*), a *model.xml* file exists. The path to the *model.xml* file is frequently used as an argument in many subsequent SEEKR2 processes, especially during the run phase. It is to be noted that directly altering the *model.xml* without reinitiating *prepare.py* is not recommended. The *run.py* script initiates the run phase, and the following command launches the script to run MD simulations within the SEEKR2 milestones.

```
python ~/seekr2/seekr2/run.py any \
SEEKR_simulation/model.xml
```

In the above command, "any" serves as the instruction argument for the *run.py* script, prompting it to complete any pending MD or BD simulations. One might use "any_md" or "any_bd" to address unfinished MD or BD simulations. Please refer to the SEEKR2 documentation for a detailed set of available directives for the *run.py* script.

Once we initiate the *run.py* script, simulations will run until completion or interruption. The SEEKR2 framework saves checkpoints for both MD and BD simulations, allowing us to resume the calculation from where it was interrupted. To track the progress and convergence of the simulations, we use the *converge.py* script:

```
python ~/seekr2/seekr2/converge.py any \
SEEKR_simulation/model.xml -l
```

Running the *converge.py* script generates convergence plots, and images are saved in the *plots_and_images* subfolder within the <root_directory>. For additional arguments that can be used with both *run.py* and *converge.py*, we can run either script with the -h argument.

Stage 3: Analyze

The final stage within the SEEKR2 framework involves analyzing the results obtained from the simulations. This stage enables the construction of kinetics and thermodynamics profiles for the studied process. To execute the post-simulation SEEKR2 analysis:

```
python ~/seekr2/seekr2/analyze.py \
SEEKR_simulation/model.xml
```

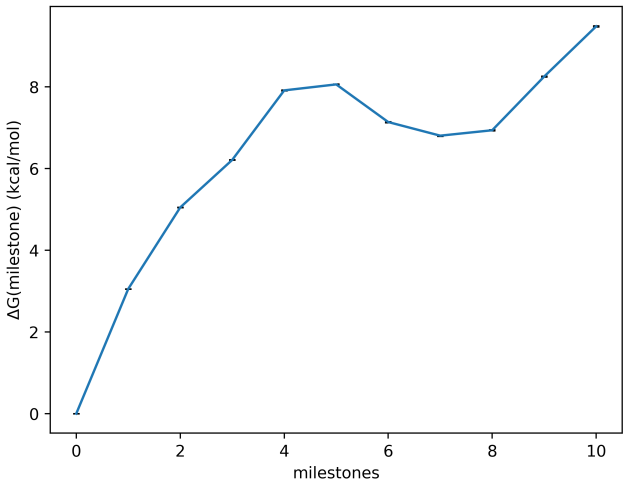


Figure 8. Free energy profile per milestone (ΔG_i) obtained from the SEEKR2 milestoning method for the trypsin-benzamidine complex.

The *analyze.py* script takes the *model.xml* file as an argument. It constructs the milestoning model, populates it with transition probabilities and simulation times within each milestone, and computes error margins. For a detailed understanding of the MMVT error margins, please refer to

https://doi.org/https://doi.org/10.33011/livecoms.5.1.2359
Living J. Comp. Mol. Sci. 2024, 5(1), 2359

the SEEKR2 manuscript [12]. Figure 8 shows the free energy profile per milestone for the trypsin-benzamidine complex, obtained by executing the *analyze.py* script.

6 SEEKR2 benchmarking

Benchmarking simulations were performed on a Linux machine with an NVIDIA Quadro RTX 5000 GPU with 16 GB RAM and an Intel Xeon W-10885M CPU. SEEKR2 milestoning simulations were run using the OpenMM version 7.7 for all the seven host-guest complexes and the trypsin benzamidine complexes, using 1 GPU with CUDA version 10.2. Table 1 describes the benchmarking results.

Receptor-ligand complex	Benchmark (ns/day)
BCD-1-butanol (5358 atoms)	569.60
BCD-1-napthylethanol (5362 atoms)	544.92
BCD-1-propanol (5361 atoms)	540.00
BCD-2-napthylethanol (5368 atoms)	526.27
BCD-aspirin (5361 atoms)	559.68
BCD-methyl butyrate (5360 atoms)	556.87
BCD-terbutanol (5358 atoms)	564.95
Trypsin-benzamidine (23036 atoms)	171.11

Table 1. Benchmarking rates for SEEKR2 simulations performed on the seven host-guest complexes and the trypsin-benzamidine complex performed on a single NVIDIA Quadro RTX 5000 GPU.

7 Conclusion

SEEKR2 has emerged as a multiscale simulation tool designed to increase the speed and efficiency of calculating kinetic and thermodynamic properties of receptor-ligand binding and unbinding. This tutorial covers the underlying theory behind SEEKR2, describing how kinetic and thermodynamic properties are calculated. Beginning with the installation and setup, this manuscript guides the readers through the processes of preparing SEEKR2 simulations, highlighting essential options such as equilibration steps and adjusting the speed of HIDR simulations. The subsequent sections then provide a detailed walkthrough of the run and analysis stages, emphasizing the significance of files like *model.xml* and scripts like *run.py* and *analyze.py*. Having achieved the learning outcomes outlined at the beginning, this tutorial is a comprehensive guide to installing and running SEEKR2 simulations, ensuring that even those unfamiliar with SEEKR2 can navigate its various features. We expect researchers are now well-positioned to apply SEEKR2 to their systems of interest.

Abbreviations

SEEKR: Simulation-enabled estimation of kinetic rates
SEEKR2: Simulation-enabled estimation of kinetic rates version 2
BD: Brownian dynamics
MD: Molecular dynamics
NAMD: Nanoscale molecular dynamics
VMD: Visual molecular dynamics
k_{on}: Association rate constant
k_{off}: Dissociation rate constant
CUDA: Compute unified device architecture
GPU: Graphics processing unit
CPU: Central processing unit
RCSB: Research collaboratory for structural bioinformatics
APBS: Adaptive Poisson-Boltzmann solver
MFPT: Mean first-passage time
MMVT: Markovian milestoning with Voronoi tessellations
PDB: Protein data bank
CV: Collective variable
COM: Center of mass
BSD: Berkeley software distribution
API: Application programming interface
SMD: Steered molecular dynamics
HIDR: Holo insertion by directed restraints
RAMD: Random acceleration molecular dynamics
HF-DFT: Hartree-Fock with density functional theory
GAMESS: General atomic & molecular electronic structure system
MP2: Møller-Plesset 2
AM1-BCC: Austin model 1 with bond charge correction
HMR: Hydrogen mass repartitioning
AMU: Atomic mass unit
frcmod: Parameter modification file
TIP3P: Transferable intermolecular potential with 3 points
TIP4Pew: Transferable intermolecular potential with 4 points with Ewald techniques
RAM: Random-access memory
RTX: Real-time ray tracing

Author Contributions

AA Ojha conceptualized and wrote the tutorial journal and is a developer of the SEEKR2 package. LW Votapka is the lead developer of the SEEKR2 package. GA Huber contributed to the Brownian dynamics section in the tutorial. RE Amaro provided support with computing resources. LW Votapka and RE Amaro provided guidance for tutorial development. S Gao extensively tested the SEEKR2 installation by following the tutorial and helped debug the instructions for installing, running, and analyzing SEEKR2 simulations.

Other Contributions

For a more detailed description of contributions from the community and others, see the GitHub issue tracking and changelog at https://github.com/anandojha/SEEKR_tutorials.

https://doi.org/https://doi.org/10.33011/livecoms.5.1.2359
Living J. Comp. Mol. Sci. 2024, 5(1), 2359

Author Information

ORCID:
Anupam Anand Ojha: 0000-0001-6588-3092
Lane William Votapka: 0000-0002-0865-5867
Gary Alexander Huber: 0000-0002-5936-6184
Shang Gao: 0009-0002-3961-5064
Rommie Amaro: 0000-0002-9275-9553

Potentially Conflicting Interests

The authors declare no conflicting interests.

Funding Information

AA Ojha acknowledges the support of the Molecular Sciences Software Institute (MolSSI) fellowship under the National Science Foundation (NSF) grant OAC-1547580. GA Huber acknowledges support from National Institutes of Health (NIH) GM31749 and University of California San Diego. RE Amaro acknowledges support from NSF Extreme Science and Engineering Discovery Environment (XSEDE) CHE060063 and NIH GM132826.

References

[1] **Amaro RE**, Mulholland AJ. Multiscale Methods in Drug Design Bridge Chemical and Biological Complexity in the Search for Cures. Nature Reviews Chemistry. 2018; 2(4):0148.

[2] **Jagger BR**, Ojha AA, Amaro RE. Predicting Ligand Binding Kinetics Using a Markovian Milestoning with Voronoi Tessellations Multiscale Approach. Journal of Chemical Theory and Computation. 2020; 16(8):5348–5357.

[3] **Ojha AA**, Srivastava A, Votapka LW, Amaro RE. Selectivity and Ranking of Tight-Binding JAK-STAT Inhibitors Using Markovian Milestoning with Voronoi Tessellations. Journal of Chemical Theory and Computation. 2023; 63(8):2469–2482.

[4] **Zuckerman DM**, Chong LT. Weighted Ensemble Simulation: Review of Methodology, Applications, and Software. Annual Review of Biophysics. 2017; 46:43–57.

[5] **Miao Y**, Feher VA, McCammon JA. Gaussian Accelerated Molecular Dynamics: Unconstrained Enhanced Sampling and Free Energy Calculation. Journal of Chemical Theory and Computation. 2015; 11(8):3584–3595.

[6] **Kokh DB**, Doser B, Richter S, Ormersbach F, Cheng X, Wade RC. A Workflow for Exploring Ligand Dissociation from a Macromolecule: Efficient Random Acceleration Molecular Dynamics Simulation and Interaction Fingerprint Analysis of Ligand Trajectories. The Journal of Chemical Physics. 2020; 153(12):125102.

[7] **Ojha AA**, Thakur S, Ahn SH, Amaro RE. DeepWEST: Deep Learning of Kinetic Models with the Weighted Ensemble Simulation Toolkit for Enhanced Sampling. Journal of Chemical Theory and Computation. 2023; 19(4):1342–1359.

[8] **Ahn SH**, Ojha AA, Amaro RE, McCammon JA. Gaussian-Accelerated Molecular Dynamics with the Weighted Ensemble Method: A Hybrid Method Improves Thermodynamic and Kinetic Sampling. Journal of Chemical Theory and Computation. 2021; 17(12):7938–7951.

[9] **Lee CT**, Amaro RE. Exascale Computing: A New Dawn for Computational Biology. Computing in Science & Engineering. 2018; 20(5):18–25.

[10] **Votapka LW**, Jagger BR, Heyneman AL, Amaro RE. SEEKR: Simulation Enabled Estimation of Kinetic Rates, a Computational Tool to Estimate Molecular Kinetics and Its Application to Trypsin–Benzamidine Binding. The Journal of Physical Chemistry B. 2017; 121(15):3597–3606.

[11] **Jagger BR**, Votapka LW, Amaro RE. SEEKR: Simulation Enabled Estimation of Kinetic Rates, A Multiscale Approach for the Calculation of Protein-Ligand Association and Dissociation Kinetics. Biophysical Journal. 2018; 114(3):42a.

[12] **Votapka LW**, Stokely AM, Ojha AA, Amaro RE. SEEKR2: Versatile Multiscale Milestoning Utilizing the OpenMM Molecular Dynamics Engine. Journal of Chemical Information and Modeling. 2022; 62(13):3253–3262.

[13] **Jagger BR**, Lee CT, Amaro RE. Quantitative Ranking of Ligand Binding Kinetics with a Multiscale Milestoning Simulation Approach. The Journal of Physical Chemistry Letters. 2018; 9(17):4941–4948.

[14] **Ahn SH**, Jagger BR, Amaro RE. Ranking of Ligand Binding Kinetics Using a Weighted Ensemble Approach and Comparison with a Multiscale Milestoning Approach. Journal of Chemical Information and Modeling. 2020; 60(11):5340–5352.

[15] **Eastman P**, Swails J, Chodera JD, McGibbon RT, Zhao Y, Beauchamp KA, Wang LP, Simmonett AC, Harrigan MP, Stern CD, et al. OpenMM 7: Rapid Development of High Performance Algorithms for Molecular Dynamics. PLoS Computational Biology. 2017; 13(7):e1005659.

[16] **Wang J**, Wang W, Kollman PA, Case DA. Antechamber: An Accessory Software Package for Molecular Mechanical Calculations. J Am Chem Soc. 2001; 222(1).

[17] **Salomon-Ferrer R**, Case DA, Walker RC. An Overview of the Amber Biomolecular Simulation Package. Wiley Interdisciplinary Reviews: Computational Molecular Science. 2013; 3(2):198–210.

[18] **Huber GA**, McCammon JA. Browndye: A Software Package for Brownian Dynamics. Computer Physics Communications. 2010; 181(11):1896–1905.

[19] **Huber GA**, McCammon JA. Brownian Dynamics Simulations of Biological Molecules. Trends in Chemistry. 2019; 1(8):727–738.

[20] **Muñiz-Chicharro A**, Votapka LW, Amaro RE, Wade RC. Brownian Dynamics Simulations of Biomolecular Diffusional Association Processes. Wiley Interdisciplinary Reviews: Computational Molecular Science. 2023; 13(3):e1649.

[21] **Ermak DL**, McCammon JA. Brownian Dynamics with Hydrodynamic Interactions. The Journal of Chemical Physics. 1978; 69(4):1352–1360.

[22] **Jurrus E**, Engel D, Star K, Monson K, Brandi J, Felberg LE, Brookes DH, Wilson L, Chen J, Liles K, et al. Improvements to the APBS Biomolecular Solvation Software Suite. Protein Science. 2018; 27(1):112–128.

[23] **Luty BA**, McCammon JA, Zhou HX. Diffusive Reaction Rates from Brownian Dynamics Simulations: Replacing the Outer Cutoff Surface by an Analytical Treatment. The Journal of Chemical Physics. 1992; 97(8):5682–5686.

[24] **Northrup SH**, Allison SA, McCammon JA. Brownian Dynamics Simulation of Diffusion-Influenced Bimolecular Reactions. The Journal of Chemical Physics. 1984; 80(4):1517–1524.

[25] **Vanden-Eijnden E**, Venturoli M, Ciccotti G, Elber R. On the Assumptions Underlying Milestoning. The Journal of Chemical Physics. 2008; 129(17).

[26] **Bello-Rivas JM**, Elber R. Exact Milestoning. The Journal of Chemical Physics. 2015; 142(9).

[27] **Elber R**. Milestoning: An Efficient Approach for Atomically Detailed Simulations of Kinetics in Biophysics. Annual Review of Biophysics. 2020; 49:69–85.

[28] **Ojha AA**, Votapka LW, Amaro RE. QMrebind: Incorporating Quantum Mechanical Force Field Reparameterization at the Ligand Binding Site for Improved Drug-Target Kinetics Through Milestoning Simulations. Chemical Science. 2023; 14(45):13159–13175.