THE UNIVERSITY OF CHICAGO

ENHANCED SAMPLING OF RARE DYNAMIC EVENTS

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCES
IN CANDIDACY FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

DEPARTMENT OF CHEMISTRY

BY
JEREMY OWEN BECKER TEMPKIN

CHICAGO, ILLINOIS
AUGUST 2017

To Sarah Matt for her love, kindness and understanding,

To Terry Tempkin for a lifetime of unconditional love and support,

And in loving memory of Alan Tempkin.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Computer simulations are an essential tool for the study of physical processes in a broad range of fields including chemistry, biology, physics, engineering, finance, and geology. Many of the most interesting events occur on timescales that are many orders of magnitude longer than is tractable to simulate directly. This challenge has motivated the development of numerous methods to enhance the sampling of rare dynamical events.

In the first part of this work, we describe a multilevel preconditioning scheme that allows one to use a less expensive model to guide the exploration of the energy landscape of a more expensive but presumably more accurate model. Our preconditioning formalism maintains the stability of the solutions to the expensive model and is robust to the relative quality of the inexpensive model. We demonstrate how this multilevel scheme can be used to accelerate the convergence of path-refinement algorithms and geometry optimizations.

In the second part, we describe a general mathematical framework for trajectory stratification for efficiently simulating rare dynamical events. Trajectory stratification involves decomposing trajectories of a given stochastic process into restricted subsets in space and time, computing averages over the subsets independently, and recombining those averages in an appropriately weighted sum yielding averages of dynamical quantities over the original unbiased process. We demonstrate how this framework leads to a novel class of rare event methods that are capable of estimating a much broader class of dynamical expectations than was previously tractable.

In the third part, we describe an open-source Python-based toolkit for rapid prototyping of enhanced sampling algorithms. The toolkit is a Python package that wraps commonly used molecular dynamics engines and provides a high-level and model agnostic programming interface for developing enhanced sampling algorithms. We illustrate how this programming framework facilitates expressive and portable implementations of enhanced sampling algorithms that are can be readily ported to HPC environments.

CHAPTER 1

INTRODUCTION

Computer simulations are a powerful tool for the study of physical processes in fields as diverse as chemistry, physics, engineering, and finance [4, 29, 25, 12]. However, many of the events of interest occur on timescales that are orders of magnitude longer than is tractable to simulate directly. For example, large conformational changes in biomolecules, protein binding and protein folding occur on timescales orders of magnitude longer than can be simulated directly with atomistic molecular models [25].

Many calculations, such as geometry optimization or reaction path refinement, can be cast in terms of a iterative procedure. In many cases, the number of steps required to converge these calculations can rapidly become computationally prohibitive. This is especially the case when the complexity of molecular models are increased to capture the necessary details of the system of interest with sufficient accuracy. This situation poses a tradeoff between accurately representing the essential features of the system of interest or reverting to a potentially inaccurate but tractable model. For example, a common practice is to first solve the problem using an inexpensive model and further refine the solution using a series of more expensive models. This procedure relies heavily on the initial model being of sufficient accuracy to provide a reasonable intermediate solution to serve as the basis for further refinement. Further, many schemes are not formally guaranteed to maintain the stability of the fixed points of the more expensive model and the solution to these schemes generally depend strongly on the relative quality of the inexpensive model.

In Chapter 2 we present a multilevel preconditioning scheme for accelerating the convergence of iterative molecular calculations. In our scheme, a less expensive model is used to guide and accelerate the exploration of the energy landscape of a more expensive model while formally guaranteeing the stability of the fixed points of the latter. Our scheme is different than previous schemes in that it maintains the stability of the fixed points of the

expensive but accurate model regardless of the relative quality of the inexpensive model. We show that convergence to these fixed points is robust to the quality of the inexpensive model. In Section 2.3, we illustrate the method for energy minimization and for the string method for reaction path refinement [20, 56, 68, 88, 19, 21, 20, 58, 44, 15].

A cornerstone rare event method in the computer simulation of condensed-phase systems is the Umbrella Sampling (US) method [83, 69, 11, 25, 49]. The central idea in US is to divide a distribution of interest along one or more reaction coordinates into subsets or strata, sample each strata independently and recombine those statistics in a weighted sum to compute an expectation against the original target distribution. When the strata are chosen appropriately, their statistics can be computed with much less overall effort than sampling the original distribution directly. In this way, US is a form of stratification sampling, which has a long history in statistics [62].

The Nonequilibrium Umbrella Sampling (NEUS) method was originally developed to extend US to compute averages against driven nonequilibrium steady states [89, 16, 13, 14] and was later extended to compute reaction rates [15]. NEUS is one of a number of "trajectory splitting" techniques [30, 36, 34, 86, 2, 42, 10, 32, 33]. These methods differ in their implementation, but they generally rely on pruning and branching states in rarely visited regions of state space and the subsequent forward evolution of the branched states to sample rarely observed trajectories. These methods were also used to compute dynamical expectations like the rate of transition [15, 86, 2].

In Chapter 3, we build on the NEUS method to develop a general mathematical framework for trajectory stratification. We show how this framework reveals the full generality of trajectory stratification for computing a wider variety of dynamic averages than was previously computed. In particular, we show how the NEUS method readily generalizes to computing expectations against non-stationary stochastic dynamics. The framework reveals a common mathematical structure shared by existing algorithms for sampling rare events.

In Section 3.5.1, we illustrate the terminology of the framework in the context of a simple Markov model. In Secton 3.5.3, we show how the framework can be used to compute finite-time hitting probabilities by stratifying state space in both space and time. In Section 3.5.4, we show how the non-stationary version of the NEUS method can be used to stratify the evolution of path-integrated variables such as the accumulated work in an non-equilibrium driven process. We show how this algorithm results in a more efficient method for computing free energy differences. The development of the trajectory stratification framework is a first step towards a formal numerical analysis of the method.

In Chapter 4, we present an overview of a software package called the Enhanced Sampling Toolkit for rapid prototyping of enhanced sampling algorithms. The Enhanced Sampling Toolkit is a Python package that provides a high-level model agnostic API that wraps popular molecular dynamics packages. The API is designed to allow developers of enhanced sampling algorithms to write algorithm-level code without having to worry about the details of the underlying dynamics engine. The API is modular and extensible for wrapping new or custom dynamics engines and developers can easily port their implementations to take advantage of HPC hardware. In addition to the API, the toolkit provides a library of Python modules that implement common enhanced sampling algorithms for developers to build and test new enhanced sampling algorithms.

CHAPTER 2

MULTISCALE PRECONDITIONING FOR ITERATIVE MOLECULAR CALCULATIONS

Iterative procedures for optimizing properties of molecular models often converge slowly owing to the computational cost of accurately representing features of interest. Here, we introduce a preconditioning scheme that allows one to use a less expensive model to guide exploration of the energy landscape of a more expensive model and thus speed the discovery of locally stable states of the latter. We illustrate our approach in the contexts of energy minimization and the string method for finding transition pathways. The relation of the method to other multilevel simulation techniques and possible extensions are discussed.

2.1 Introduction

Many tasks in computational chemistry can be cast in terms of a repeated iterative procedure. Whether the goal is to find an optimized molecular geometry or refine a reaction path, well developed computational techniques exist that are expected to progressively converge to a satisfactory solution when initiated from a reasonable starting guess (e.g., see [7] and references therein). Nevertheless, the number of steps necessary for convergence of such calculations can rapidly become computationally prohibitive as the complexity of models increases to capture details of molecular systems realistically. Most researchers are then faced with a painful choice: either represent the system with a model that is, in principle, capable of faithfully representing the essential features of the system of interest but is too costly to be practical, or revert to a computationally inexpensive but more approximate model.

A common practice is to first solve a problem using an approximate model, and then try to further improve the solution by using progressively more accurate representations.

Prototypical examples of this situation are encountered in quantum chemical geometry optimization, where a first round of calculations is carried out using an inexpensive lower-level method before switching to a higher-level method, or when a reaction path in a macromolecular system is first simulated using an approximate coarse-grained model before being simulated with an all-atom force field. While this procedure is intuitively reasonable, it relies on the chosen approximate model being sufficiently accurate that it provides a meaningful, albeit imperfect, intermediate solution that can serve as the basis for further improvements. However, sequences of calculations with models of different accuracy are not formally guaranteed to save computational effort. Indeed, such a protocol can converge to an incorrect solution if the approximate model is ill-chosen.

The goal of the present paper is to demonstrate how one can use an inexpensive and possibly inaccurate, "coarse-grained (CG)" model to accelerate calculations with an accurate and expensive "fine-grained (FG)" model. Care is taken to ensure that the approach preserves stability of the original FG solution and is robust to different choices of the CG model. The strategy of coupling multiple resolutions is well-known in equilibrium sampling [57, 52, 53, 90, 51, 70, 91], but our approach, a form of preconditioning, is fundamentally different in structure. We formulate it generally without reference to a specific computational goal. Then, we present numerical experiments that provide examples of specific applications. For the sake of clarity, the method is first illustrated in the context of energy minimization. Next, the method is used to accelerate the convergence of the string method for finding conformational transition paths in complex systems [20, 56, 68, 88, 19, 21, 20, 58, 44, 15]. The reactions that we consider are a transition between wells on a variant of the Müller-Brown potential, isomerization of the alanine dipeptide, and the conformational change of an actin monomer in going from the globular to filamentous states. We conclude with an outlook on future applications and extensions of the method.

2.2 Theory

As already mentioned, our goal is to couple two models constructed at different resolution, such that one can guide the sampling of the other. Our approach should be applicable to any iterative root-finding method, including adaptive sampling, although care will be necessary to consider how best to implement the scheme given the details of each method. For the purposes of exposition, we first motivate and detail our multiscale (or, more generally, multilevel) protocol for a general framework. We present the approach in terms of a CG model accelerating the interrogation of an FG model. The extension to more than two models is straightforward, as models can be coupled in a pairwise fashion to form a chain.

Most root-finding methods can be expressed as iterative schemes to find a fixed point which may be, for example, a molecular geometry, or a reaction path. We will assume that one has chosen a model with sufficient accuracy (the FG model), and we will let $\mathcal{S}$ represent the operation that is iteratively applied to the "state" φ to estimate the desired quantity. For example, in energy minimization, φ represents the coordinates of the atoms, and $\mathcal{S}$ represents one step of an energy minimization algorithm (e.g., steepest-descent, conjugate gradients, Newton-Raphson, etc.). In the case of the string method, φ represents a path of the system connecting two metastable states, while $\mathcal{S}$ represents the algorithmic operations necessary to update the path (evolution and reparametrization, as described later). Given this notation, the iteration is summarized as

$$\varphi_{m+1} = \mathcal{S}(\varphi_m), \tag{2.1}$$

and the goal of the calculation is to locate a fixed point:

$$\varphi_* = \mathcal{S}(\varphi_*).$$

We seek to modify the update of the state while maintaining the stability of this fixed

point. To this end, consider the change of variables $\psi = \varphi - \mathcal{S}_{\mathrm{CG}}(\varphi) = (I - \mathcal{S}_{\mathrm{CG}})(\varphi)$, where $\mathcal{S}_{\mathrm{CG}}(\varphi)$ is the state update for a CG model that has been chosen to be similar to the FG model while being relatively inexpensive to simulate. In general, $\mathcal{S}$ and $\mathcal{S}_{\mathrm{CG}}$ are defined in terms of a space of collective variables, and their combination here relies only on their having a set of common variables. In the development below, we require the sets of collective variables of the FG and CG models to have the same dimension. However, the scheme can be easily modified to precondition only a lower dimensional set of variables (see [48] for a similar modification in the context of parallel-in-time integration).

Notice that we can rewrite the original fixed point problem in the ψ variables as

$$(I - \mathcal{S})[(I - \mathcal{S}_{\mathrm{CG}})^{-1}(\psi)] = 0. \tag{2.2}$$

If we try to solve this equation by fixed point iteration we obtain

$$\psi_{m+1} = \psi_m - (I - \mathcal{S})[(I - \mathcal{S}_{\mathrm{CG}})^{-1}(\psi_m)]. \tag{2.3}$$

Transforming back into the φ variables this becomes

$$\begin{aligned} \varphi_{m+1} - \mathcal{S}_{\mathrm{CG}}(\varphi_{m+1}) &= \varphi_m - \mathcal{S}_{\mathrm{CG}}(\varphi_m) - \varphi_m + \mathcal{S}(\varphi_m) \\ &= \mathcal{S}(\varphi_m) - \mathcal{S}_{\mathrm{CG}}(\varphi_m). \end{aligned} \tag{2.4}$$

To see why Eq. (2.4) ensures stability of the original solution, we rearrange it to

$$\varphi_{m+1} = \mathcal{S}(\varphi_m) + [\mathcal{S}_{\mathrm{CG}}(\varphi_{m+1}) - \mathcal{S}_{\mathrm{CG}}(\varphi_m)]. \tag{2.5}$$

The CG part in square brackets goes to zero as the optimization converges, leaving only Eq. (2.1). Thus any fixed point of the multilevel iteration must also be a fixed point of the original iteration. In other words, the multilevel procedure converges to an FG solution.

That said, when there are multiple possible solutions, there is no guarantee that the same solution will always be reached by different string protocols.

Alternatively, we can rearrange Eq. (2.4) to

$$\varphi_{m+1} = \mathcal{S}_{\mathrm{CG}}(\varphi_{m+1}) + [\mathcal{S}(\varphi_m) - \mathcal{S}_{\mathrm{CG}}(\varphi_m)]. \tag{2.6}$$

In Eq. (2.6), φ_{m+1} appears on the righthand side only in the update of the CG model, and the term in square brackets involves only φ_m. Thus the cost of solving Eq. (2.6) for φ_{m+1} is determined by the cost of the CG model and not by the cost of the FG model. This is the key feature enabling a speedup since by design the CG model is less expensive to evaluate. Of course, the details of the CG model will affect the number of iterations of Eq. (2.6) that are required for convergence, and there is no guarantee of a speedup. Nonetheless, we show below that even CG models that differ substantially from the FG models can yield an acceleration.

In the context of optimization and root finding, changes of variables such as the one above are called preconditioners [63]. Indeed, if $\mathcal{S}_{\mathrm{CG}}$ was a linearization of $\mathcal{S}$ around φ_m, then exactly the same derivation would yield Newton's method for solving $\varphi = \mathcal{S}(\varphi)$. Nonlinear preconditioners are less common than linear preconditioners because they involve (approximate in most cases) inversion of a non-linear change of variables. Here we have used a change of variables specifically designed to take advantage of existing, inexpensive coarse grained models. The parareal method [50, 8] which speeds direct integration of evolutionary differential equations using inexpensive CG methods can be cast in a very similar framework as can nonlinear (and linear) multigrid iteration [6, 31]

We expect for certain systems that it will be advantageous to modulate the extent to which the CG model contributes to the state update. This can be done by introducing a

tuning parameter, Δ_m, that modulates the CG terms:

$$\varphi_{m+1} = \Delta_m \mathcal{S}_{\mathrm{CG}}(\varphi_{m+1}) + \mathcal{S}(\varphi_m) - \Delta_m \mathcal{S}_{\mathrm{CG}}(\varphi_m). \tag{2.7}$$

As Δ_m is made smaller, this scheme approaches a standard ("pure") FG iteration.

Each iteration of Eq. (2.7) requires solving for φ_{m+1}. This will typically require some additional (i.e., nested) iterative scheme. The simplest choice, which we employ in our tests, is to use

$$\varphi_{m+1}^{k+1} = \Delta_m \mathcal{S}_{\mathrm{CG}}(\varphi_{m+1}^{k}) + \mathcal{S}(\varphi_m) - \Delta_m \mathcal{S}_{\mathrm{CG}}(\varphi_m) \tag{2.8}$$

and set $\varphi_{m+1} = \varphi_{m+1}^{K}$ for some preset number of (inner) iterations, K.

In summary, the multilevel procedure is

1. Evaluate $\mathcal{S}(\varphi_m) - \mathcal{S}_{\mathrm{CG}}(\varphi_m)$ by performing one iteration each of the FG and CG update operations. Use these updates to obtain an initial guess for φ_{m+1}.

2. Solve Eq. (2.7) for φ_{m+1} by iteration of only the CG model as in Eq. (2.8). The extent of this iteration is tuned empirically.

3. Repeat until convergence (i.e., $\varphi_m = \varphi_{m+1} = \mathcal{S}(\varphi_m)$).

It should be noted that the iteration in Eq. (2.7) differs fundamentally from an approach that would first complete the optimization with the inexpensive CG model, and then follow through by iterations with the expensive FG model. It also differs from an approach in which one alternates between iterations with the CG model and iterations with the FG model. Such a scheme would not converge to the true FG solution. In contrast, the multilevel scheme we have outlined can converge only to a solution of the FG model.

2.3 Illustrative Applications

2.3.1 Energy Minimization

For the sake of clarity, let us illustrate the multilevel preconditioner in the case of a simple steepest descent energy minimization procedure. In its simplest form, finding the lowest energy state of an energy surface $U(r)$ involves following the dynamics prescribed by an ordinary differential equation of the form

$$\dot{r} = -\nabla U(r) \tag{2.9}$$

until a steady state ($\dot{r} = 0$) is reached. This procedure requires repeated evaluation of the force $-\nabla U(r)$, which can be very disadvantageous if the energy and its derivatives are computationally costly to evaluate.

Here, we consider the example of a simple system consisting of two particles that are strongly coupled by a stiff spring. Each particle experiences an external potential $V_{\mathrm{MB}}(r)$ corresponding to the Müller-Brown potential (see Fig. 2.1) [60]. The two-dimensional vector describing particle i is $r_i = (x_i, y_i)$, and the total FG potential energy of the system is $U(r_1, r_2) = V_{\mathrm{MB}}(r_1) + V_{\mathrm{MB}}(r_2) + K_s(r_1 - r_2)^2$ with $K_s = 10^4$.

We take the fine grain iteration operator, $\mathcal{S}$, to represent 100 steps of the discretization

$$r_i(k+1) = r_i(k) - h\nabla_{r_i} U(r_1(k), r_2(k)) \tag{2.10}$$

of Eq. (2.9) with step $h = 10^{-6}$. We begin the optimization from the configuration $r_1(0) = (-2, 2)$, $r_2(0) = (-2.2, 1.8)$ (see Fig. 2.1).

We explore the properties of the multilevel iteration in Eq. (2.7) with both a "good" CG model and a "bad" one. Both CG models are based on the two dimensional center of mass

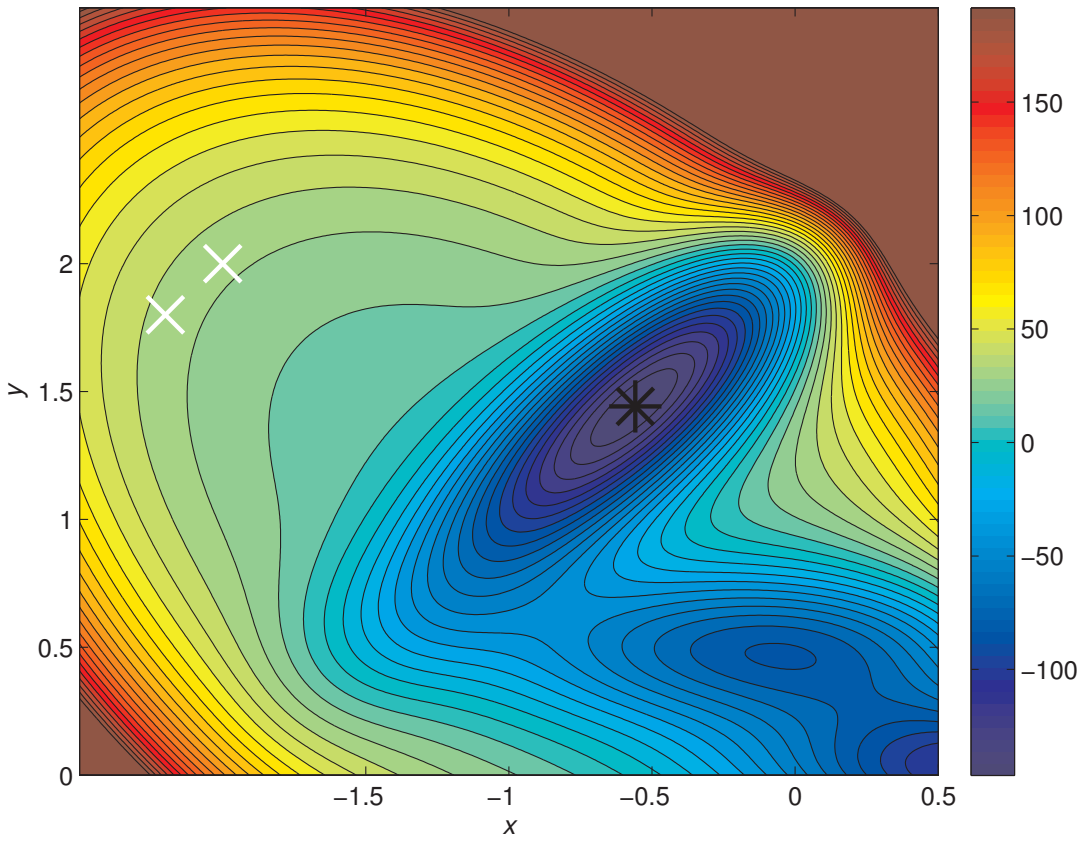


Figure 2.1: Contour plot of the Müller-Brown potential overlaid with the particle initial configurations used in the geometry optimization ($\times$) and the global minimum of the potential ($*$).

of the FG particles:

$$R(r_1, r_2) = \frac{r_1 + r_2}{2}. \tag{2.11}$$

The two models are distinguished by the external potential V_{CG} felt by the particles. In the good CG model, $V_{\mathrm{CG}} = V_{\mathrm{MB}}(R)$ is the same Müller-Brown potential that appears in the FG model described above (and pictured in Fig. 2.1). In the bad CG model, V_{CG} is the potential pictured in Fig. 2.2. The CG iteration operator, $\mathcal{S}_{\mathrm{CG}}$, represents moving the center of mass according to

$$R(k+1) = R(k) - h_{\mathrm{CG}} \nabla V_{\mathrm{CG}}(R(k)) \tag{2.12}$$

with $h_{\mathrm{CG}} = 10^{-4}$ and then setting the position of each particle to be equal to that of the center of mass. The absence of the coupling term in the center of mass update makes the larger timestep feasible. For the good CG model we use $\Delta_m = 1$ in Eq. (2.7). The iteration in Eq. (2.7) is slightly less stable with the bad CG model. Choosing $\Delta_m = 0.75$ in that case results in a stable and convergent scheme. In both cases the number of internal iterations K in Eq. (2.8) used to solve Eq. (2.7) is 50 and $\varphi^0_{m+1} = \varphi_m$.

With these choices, one iteration of the multilevel scheme is roughly a factor of 1.5 more costly than one application of $\mathcal{S}$, and we thus multiply by this factor in reporting multilevel statistics (Fig. 2.3). With the good CG model, the global minimum is found more than 20 fold faster with the multilevel scheme. With the bad CG model, the speedup is about a factor of 2. This result is remarkable given the dissimilarity between the CG and FG models (compare Figs. 2.1 and 2.2).

2.3.2 String Method

The string method seeks to discover optimal transition paths for complex reactions [20, 56, 68, 88, 19, 21, 20, 58, 44]. The paths are represented as a discretized sequence of

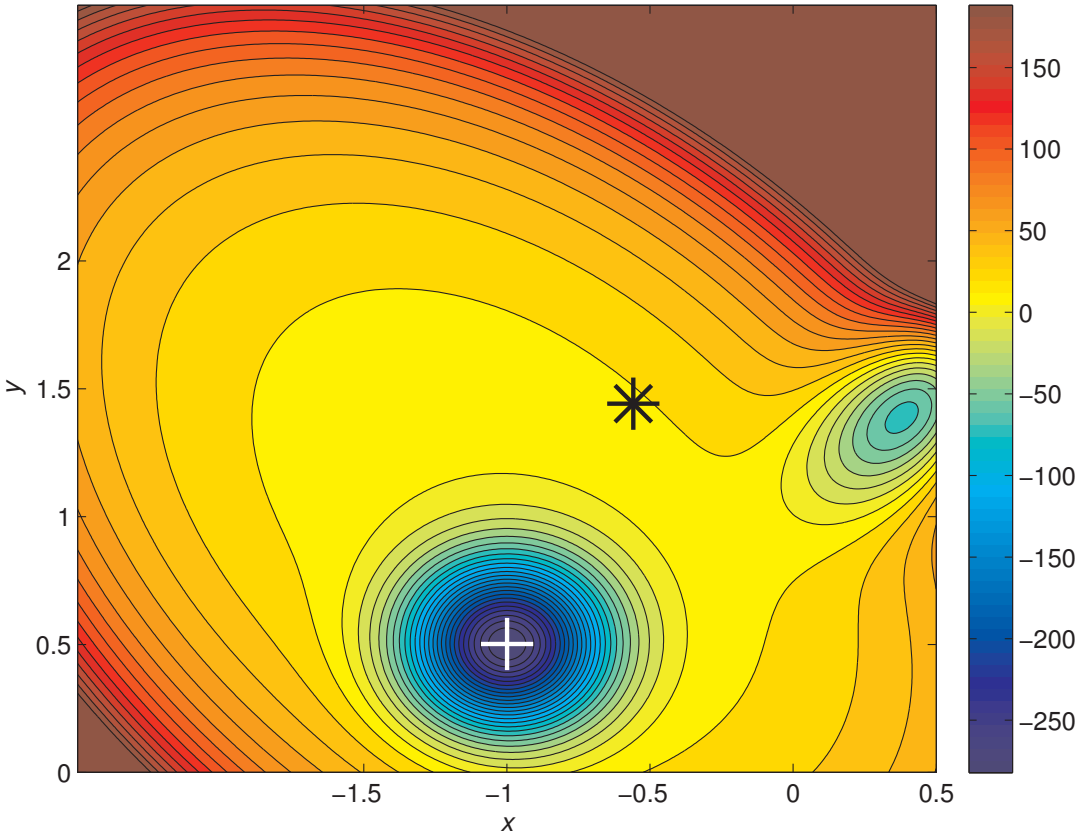


Figure 2.2: Contour plot of the bad CG model overlaid with the positions of its global minimum (+) and the Müller-Brown potential's global minimum (∗).

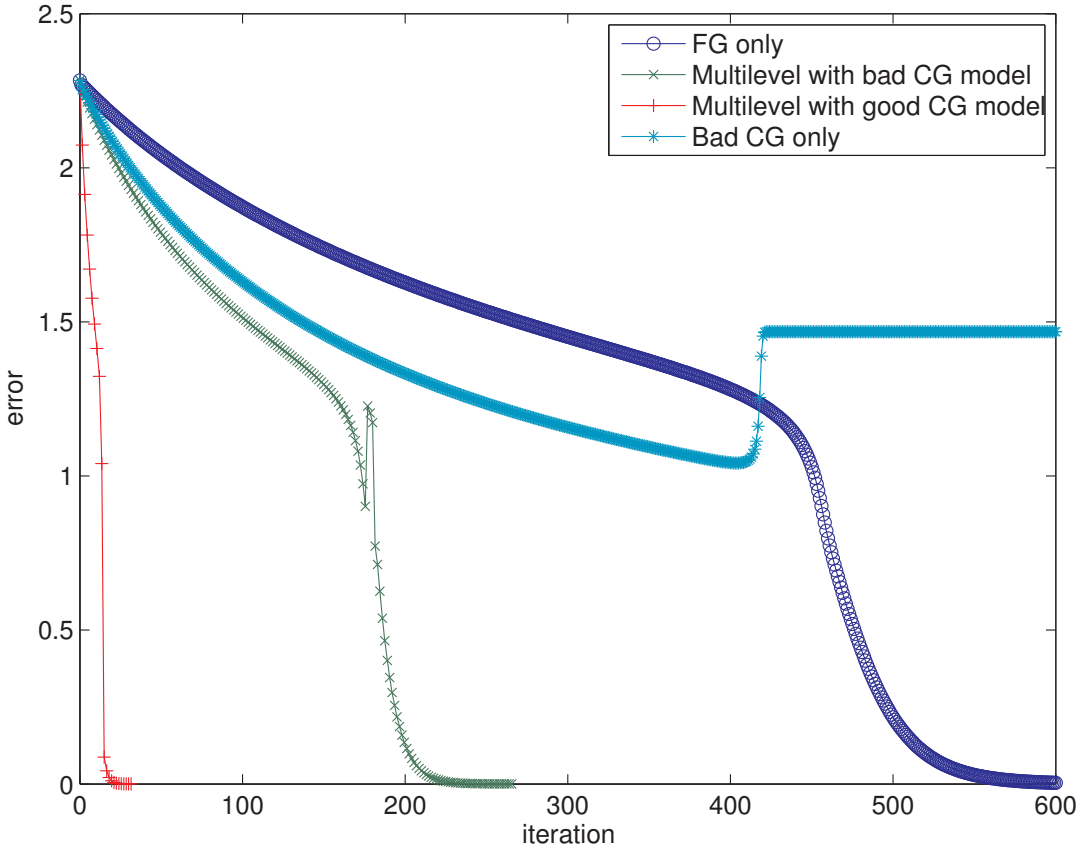


Figure 2.3: Energy minimization of the Müller-Brown model. We plot the error at each iteration of the indicated geometry optimization schemes: steepest descent with only the FG model ($\circ$), the multilevel method with bad CG model ($\times$), the multilevel method with good CG model ($+$), steepest descent Eq. (2.12) with only the bad CG model ($*$), and steepest descent Eq. (2.12) with only the good CG model ($\square$).

configurations of the system, known as "images", in a reduced dimensional space of collective variables. Starting from an initial guess for the path, the images are iteratively relaxed and redistributed until convergence. There are different schemes for evolving the string [20, 56, 68, 88, 19, 21, 20, 58, 44, 15]. When the string method is applied to finding the minimum free energy path between two metastable states, one alternates between an "evolution" step and a "reparametrization" step. In the evolution step, copies of the full system r are allowed to relax conditioned on the values of the collective variables R (fixed at the current image). The forces along the trajectories are averaged to approximate the mean force on the collective variables and the images are moved some distance in the direction of that mean force. The evolution step is followed by reparametrization of the string such that the arc length between consecutive images is uniform. In effect, determining the path amounts to solving a difficult root-finding problem in a high dimensional space.

The string method is efficient in that the computational cost grows relatively slowly with the number of collective variables and involves few global operations on the string that require communication between images. It has been used with success to shed light on the collapse of a hydrophobic chain in water [59], examine the activation of Src tyrosine kinases [28], conformational transition in myosin VI [67] and in the voltage sensor of K^+ channels [47]. Nevertheless, the method converges slowly and the large number of iterations become computationally prohibitive. It is therefore of interest to derive methods to accelerate the convergence of the current algorithms.

Müller-Brown potential

We now explore the use of the multilevel iteration in Eq. (2.8) in the context of finding minimum free energy paths by the string method. As above, we consider a system consisting of two (two dimensional) particles with positions r_1 and r_2 in a potential of the form $U(r_1, r_2) = V(r_1) + V(r_2) + K_s(r_1 - r_2)^2$ with collective variables representing the center of

mass of the two particles. Now, however, the potential V is the Müller-Brown potential modified by the addition of a short wavelength oscillatory term of the form $9\sin(10\pi x_i)\sin(10\pi y_i)$ (see Fig. 2.4). The sinusoidal term results in a rugged landscape with many metastable states (see Fig. 2.4). They persist at finite temperature and give rise to many locally stable paths, most of which are not of interest. To enable convergence to a result representative of the global free energy minimum path, some smoothing of the landscape is needed. In the present study, we employ a variant of the string method in which the free energy surface of the collective variables is smoothed on the fly by simulating with a weak constraint in the evolution step; instead of approximating the mean force on the collective variables, the images are moved to the mean position of the collective variables along the restrained trajectories. Ideally, the resulting reaction path is one that minimizes an effective energy that is a smooth approximation (by Gaussian convolution of the probability density) of the true free energy. While this smoothing procedure reduces the diversity of paths obtained, it can be expensive.

Another feature of this system that makes it challenging is the stiff spring connecting the two particles. As in the geometry optimization example, we choose $K_s = 10^4$, which mandates very small timesteps ($h = 10^{-6}$) in the integration of the equations of motion. The restrained simulations of the full system are carried out using the discretization

$$r_i(k+1) = r_i(k) - hK_r\nabla_{r_i}(R(k) - R_0)^2 - h\nabla_{r_i}U(r_1(k), r_2(k)) + \sqrt{2k_BTh}\,\xi_i(k) \quad (2.13)$$

where the $\xi_i(k)$ are independent normal random variables with mean 0 and variance 1, and R_0 represents the value to which the collective variables are being restrained. We choose $K_r = 5000$ and generate trajectories of length 0.1 time units; $k_BT = 9$.

For this test, we do the following in each evaluation of the string operator $\mathcal{S}$.

1. Reconstruct the positions of the full system r from the images in the collective-variable space by setting the positions of both particles equal to R.

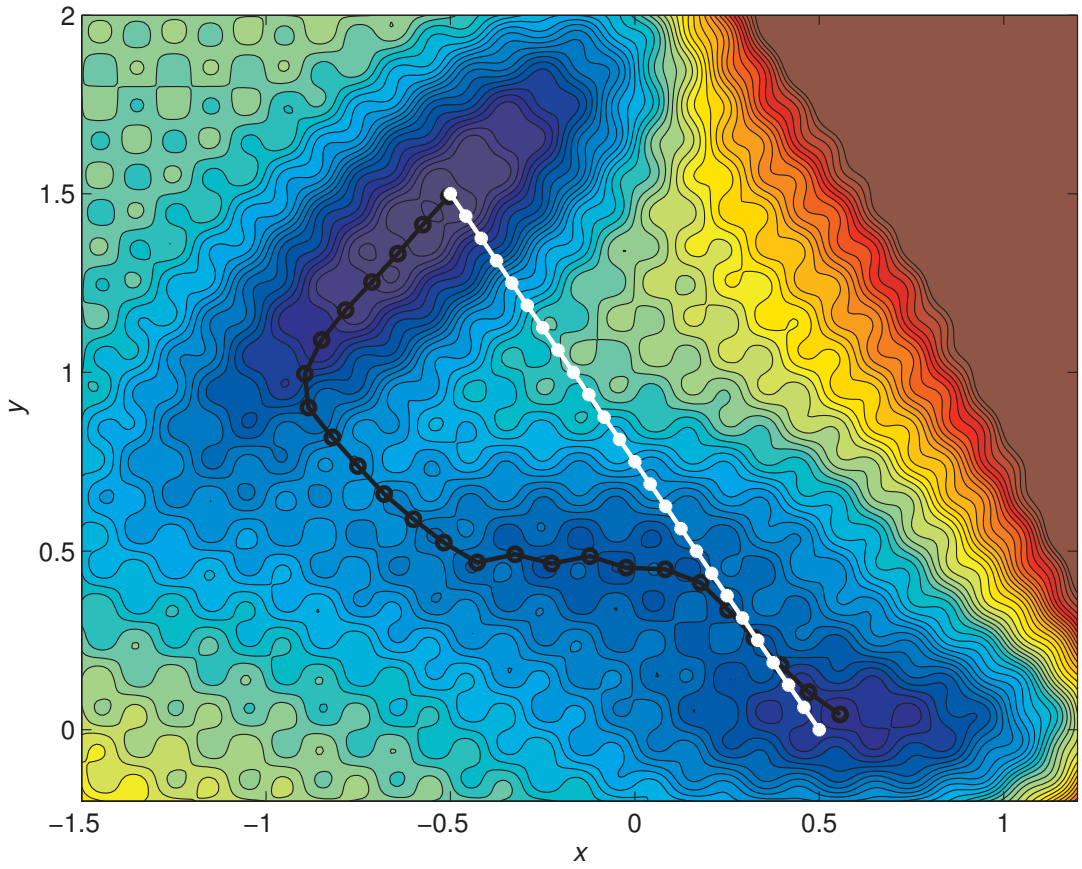


Figure 2.4: Contour plot of the potential experienced by each of the two particles comprising the FG Müller-Brown model. Overlaid are the minimum free energy path on the smoothed approximate free-energy surface, i.e. the fixed point of $\mathcal{S}$ (black circles) and the initial path used in all tests (white circles).

2. Evolve each image according to Eq. (2.13) for 0.1 time units.

3. For each image, compute the average of the collective variable R over the resulting trajectory. Set the new image position to be equal to this average.

4. Reposition the image locations along the linearly interpolated path so that they are approximately equidistant.

As in the geometry optimization problem, we test the performance of this scheme with good and bad CG models. Both represent only the collective variables. The good model corresponds to using exactly the Müller-Brown potential without any added oscillatory term

(top plot in Fig. 2.5). We expect the Müller-Brown potential to be a reasonable (though not exact) approximation of a smoothed version of the free energy landscape implied by U. The features of the bad model differ sharply from what we would expect of the true free energy landscape (bottom plot in Fig. 2.5). It has a deep well at the position of the shallow intermediate well in the Müller-Brown potential (near $(x, y) = (0, 0.5)$) and a relatively shallow well in the same position as the Müller-Brown potential's deepest well (near $(x, y) = (-0.5, 1.5)$). The Müller-Brown potential's second deepest energy well (near $(x, y) = (0.5, 0)$) is not represented at all in the bad CG model.

Given these CG models, $\mathcal{S}_{\mathrm{CG}}$ represents the result of one iteration of the string method (with no smoothing) applied directly to the smooth CG potentials pictured in Fig. 2.5. In slightly more detail, evaluating $\mathcal{S}_{\mathrm{CG}}$ entails evolving the images according to Eq. (2.12) for 10^{-3} units of time with a timestep of $h_{\mathrm{CG}} = 10^{-4}$, where V_{CG} represents either the good or the bad CG model, and then repositioning the images along the (linearly interpolated) path so that they are approximately equidistant. In summary, in this test one evaluation of the string operator $\mathcal{S}_{\mathrm{CG}}$ entails

1. Evolve each image according to Eq. (2.12) for 10^{-3} time units.

2. Set the new image position to be equal to the final value of R along this trajectory.

3. Reposition the image locations along the linearly interpolated path so that they are approximately equidistant.

In each case, we construct an initial string by linearly interpolating 25 images between the basins near $(-0.5, 1.5)$ and $(0.5, 0.0)$. To solve Eq. (2.7) for φ_{m+1} at each step, we use 10 iterations of the fixed point scheme in Eq. (2.8) with $\varphi^0_{m+1} = \varphi_m$. Implemented with the good CG model, the multilevel string iteration converges well with $\Delta_m = 1$. Implemented with the bad CG model the method is less stable: with $\Delta_m = 1$, the scheme does not seem to converge at all, but with $\Delta_m = 0.25$, it is stable and approaches the FG path. In all of our

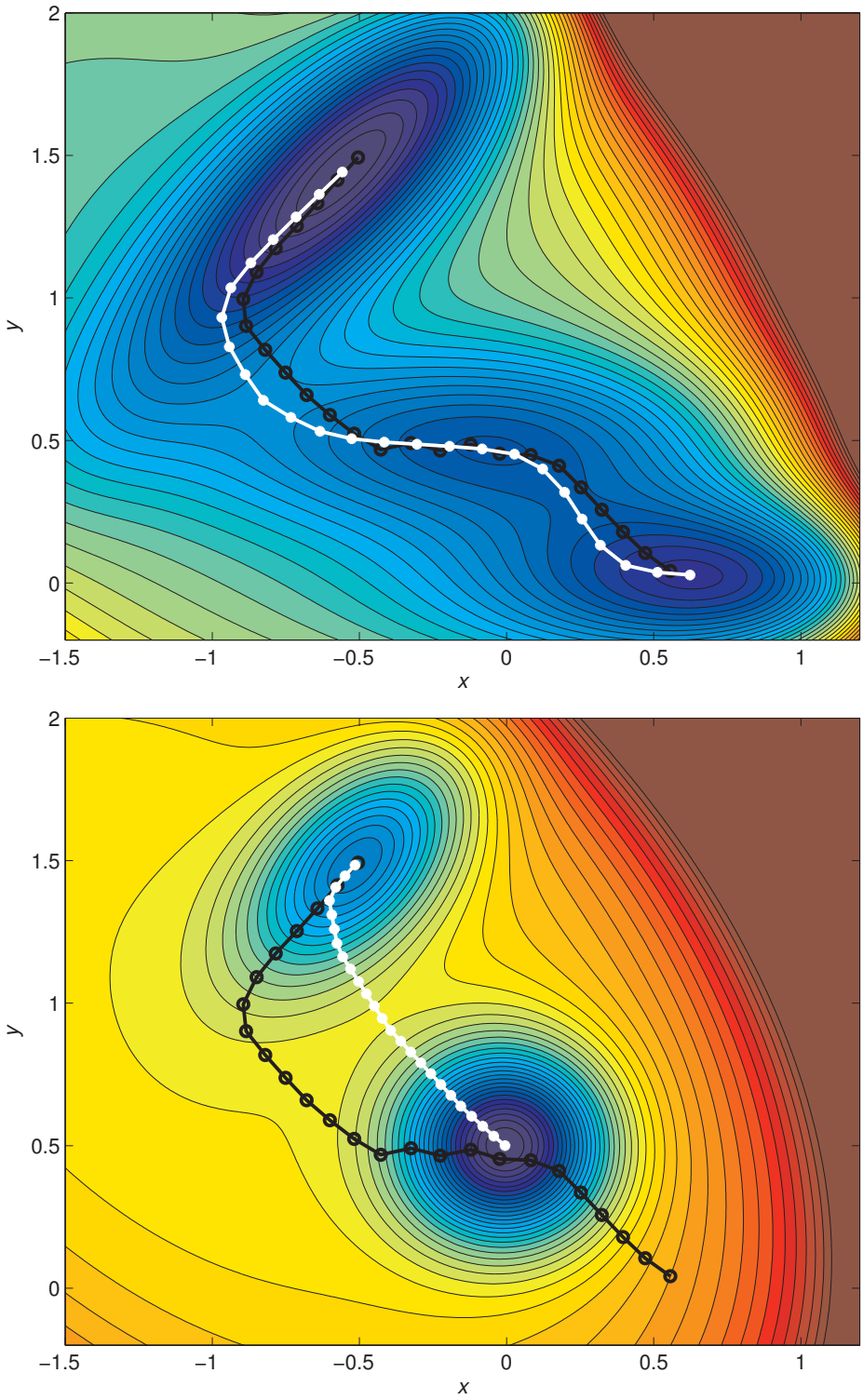


Figure 2.5: CG models used for finding reaction pathways of the Müller-Brown system. Contour plots of the (top) good and (bottom) bad CG models. The fixed point of the corresponding $\mathcal{S}_{\mathrm{CG}}$ is marked with white dots, and the fixed point of $\mathcal{S}$ is marked with black dots.

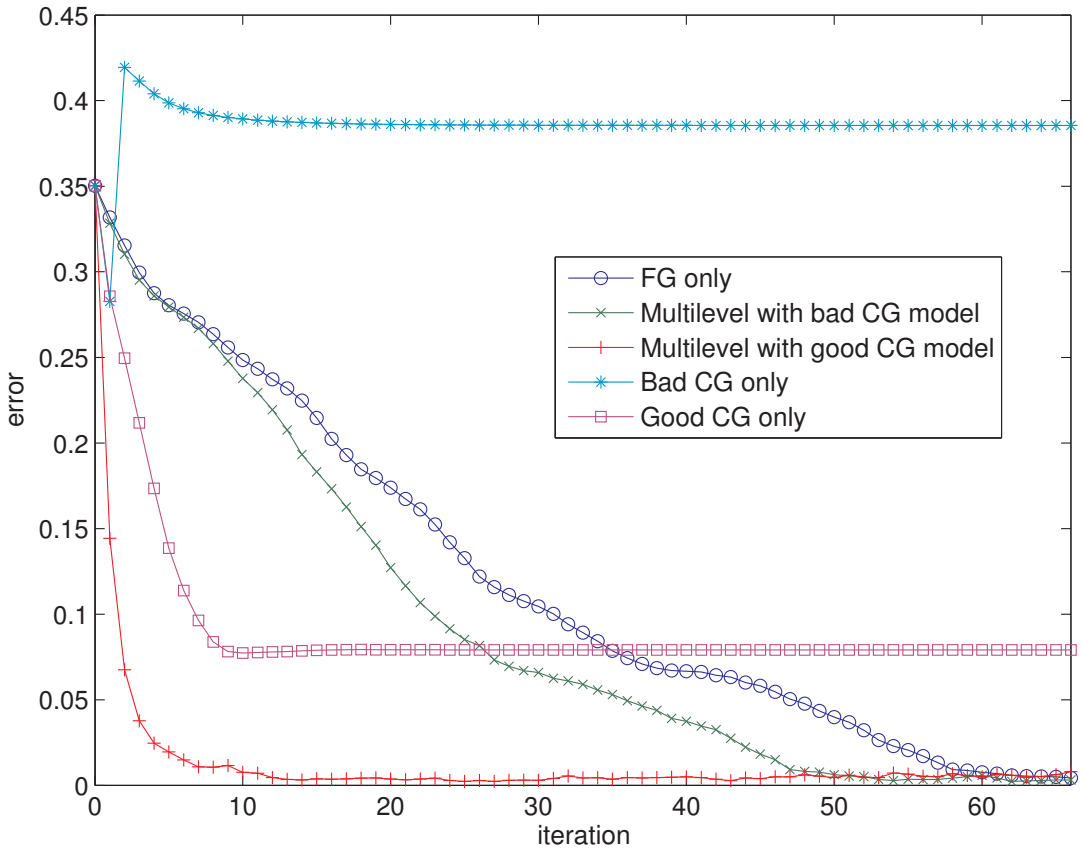


Figure 2.6: String method for the Müller-Brown model. We plot the error (as computed by Eq. (2.14)) at each iteration of the indicated string methods: the string method with only the FG model ($\circ$), the multilevel string method with bad CG model ($\times$), the multilevel string method with good CG model ($+$), the string method applied directly to the bad CG model ($*$), and the string method applied directly to the good CG model ($\square$).

tests, the additional computational cost imposed by the CG level calculations are negligible compared to evaluation of $\mathcal{S}$ and can be ignored.

In Fig. 2.6 we report the error in the path as a function of the iteration number for various schemes. The error is a discrete approximation of the average distance between the current iteration of the string, φ_m, and a final converged path computed using only the FG model, φ_*:

$$\text{error}_m = \frac{1}{N}\sum_{j=1}^{N}\sqrt{(\varphi_m^{(j,x)} - \varphi_*^{(j,x)})^2 + (\varphi_m^{(j,y)} - \varphi_*^{(j,y)})^2}, \tag{2.14}$$

where N is the number of images, which are indexed by j; x and y represent the components

of the positions of the images in the collective variable space (i.e., R).

With the good CG potential, the multilevel string method accelerates the convergence of the string by up to a factor of about 15. The reader should note that, while the minimum free energy path in the good CG model (i.e., the fixed point of $\mathcal{S}_{\mathrm{CG}}$) is qualitatively similar to the minimum free energy path in the smoothed FG model (i.e., the fixed point of $\mathcal{S}$), the agreement is not perfect. The two paths are plotted over isoenergetic contours of the good CG potential in the top panel of Fig. 2.5. The difference (as measured by Eq. (2.14)) between the fixed point of $\mathcal{S}_{\mathrm{CG}}$ with the good CG model and the fixed point of $\mathcal{S}$ is apparent from the "Good CG only" curve in Fig. 2.6. Despite the discrepancy, the form of Eq. (2.7) guarantees convergence of the multilevel string method to a fixed point of $\mathcal{S}$, as shown by the "Multilevel" curves in Fig. 2.6.

Interestingly, with the bad CG potential (and $\Delta_m = 0.25$), the multilevel string method still accelerates convergence of the path. This is surprising given that the minimum free energy path on the bad CG potential bears little resemblance to the fixed point of $\mathcal{S}$ (see bottom panel in Fig. 2.5). In fact, with the bad CG model, the fixed point of $\mathcal{S}_{\mathrm{CG}}$ is farther (as measured by the error above) from the fixed point of $\mathcal{S}$ than is the initial straight line connecting the basins (see Fig. 2.6). This makes clear the value of Eq. (2.7) and distinguishes our approach from simple alternation between CG and FG string operations. The latter would result in a path somewhere between the fixed point of $\mathcal{S}_{\mathrm{CG}}$ and $\mathcal{S}$. Analysis beyond the scope of this paper is required to determine how different properties of CG models contribute to the speedup.

Alanine dipeptide

To evaluate the multilevel string with a molecular model, we consider finding reaction pathways of the well-characterized [3, 79] c7ax to c7eq transition of the alanine dipeptide (Figure 2.7). The potential used for the FG model is the all-atom CHARMM22 force field with the

CMAP correction [54] and a Generalized-Born implicit solvation model [66]. We simulate the system with Langevin dynamics on the heavy atoms with a timestep of 1 fs and a friction coefficient of 5 ps^{-1} at 273 K unless otherwise noted. The SHAKE algorithm is used to constrain bonds involving hydrogen atoms [73]. The FG calculations are performed in NAMD2.9 [71], and the CG calculations are performed using the PNM module in CHARMM (version c35b3) [7].

The collective variables (CVs) for the string procedure are the ϕ and ψ backbone dihedrals (Figure 2.7). The string has 16 images. The initial position of the string is obtained by linearly interpolating between the points (ϕ, ψ) = (70.5, –69.5) and (ϕ, ψ) = (–82.7, 73.5). We generate initial structures (i.e., the values for remaining degrees of freedom) for the images sequentially starting from the endpoints. The structure for each image is equilibrated for 500 ps with harmonic restraints on each dihedral angle using a force constant of 200 kcal/mol/deg^2; then, the harmonic restraints were moved to the next image and the system was allowed to relax over 500 ps.

The string method is used to evaluate the operator $\mathcal{S}$. At each iteration we do the following:

1. Update the images by dragging the peptide toward the target point in the CV space. The dragging step moves the minima of harmonic restraints with force constants of 500 kcal/mol/deg^2 in 10 equal increments over a total of 100 ps.

2. Simulate the prepared images with harmonic restraints at the target point in CV space with a force constant of 25 kcal/mol/deg^2 for a total of 10 ps. The CVs are sampled every 10 steps and averaged over the trajectory to determine the new CV values. Samples that exceed a cutoff distance of 15° from the target in each CV are removed from the average.

3. Reparameterize the interpolated path to maintain approximately equidistant spacing between the images while smoothing the string as in ref. [88] with $\kappa = 0.1$. The repa-

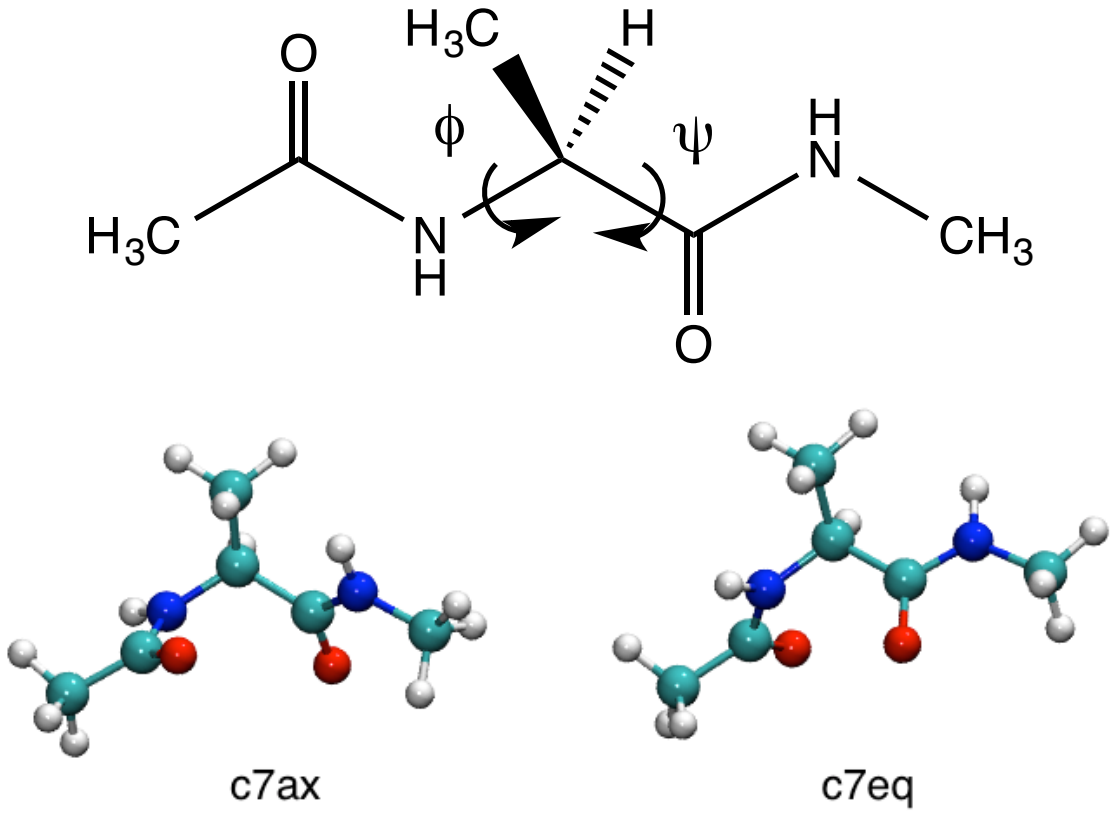


Figure 2.7: The collective variables used in the string method are the ϕ and ψ backbone dihedral angles (above). The structures used for the PNM basins are the c7ax and c7eq configurations (below).

rameterization and smoothing steps are applied iteratively 5 times for each evaluation of $\mathcal{S}$.

The CG model is a plastic network model [55] (PNM) that is supplemented with the bond, angle, dihedral angle and improper dihedral angle energy terms from the CHARMM22 force field in vacuum. The basin structures for the PNM model are prepared by minimizing the peptide structure using the full CHARMM22 force field at the c7ax and c7eq basins (Figure 2.7). As in the Müller-Brown example above, we introduce both good and bad CG models (Figure 2.8). The good CG model is the PNM [55] constructed between all heavy atoms in the alanine dipeptide. The PNM model uses spring constants of 1.0 kcal/mol, a cutoff distance of 12.0 Å, zeros of both basins set to 0.0 kcal/mol, and a mixing constant of 1.0 kcal/mol. The bad CG model uses the same terms as the good CG model but adds the electrostatic terms of the CHARMM22 force field and unphysical charges on the atoms ($+1.0e$ is added to each heavy atom charge). The good CG model converges to a pathway similar to the conventional all-atom model. In contrast, the bad CG model favors a pathway that is significantly distorted from the FG pathway as a result of the exaggerated electrostatics (Figure 2.8).

The string method is used to evaluate the operator $\mathcal{S}_{\mathrm{CG}}$. Here, the dynamics are at 100 K with a friction coefficient of 30 ps^{-1} and a timestep of 1 fs but are otherwise the same as above. At each iteration of we do the following:

1. Update the images by performing 100 ps with harmonic restraints with force constants of 500 kcal/mol/rad^2 placed at the target position in CV space.

2. Simulate the prepared images with harmonic restraints at the target point in CV space with a force constant of 50 kcal/mol/rad^2 for a total of 10 ps. The CVs are sampled every 10 steps and averaged over the trajectory to determine the new CV values.

3. Reparameterize the interpolated path to maintain approximately equidistant spacing

between the images while smoothing the string as in ref. [88] with $\kappa = 0.1$. The reparameterization and smoothing steps are applied iteratively 5 times for each evaluation of $\mathcal{S}_{\mathrm{CG}}$.

We iterate Eq. (2.8) with $K = 5$ and $\Delta_m = 1.0$ using the CVs from step 1 of the previous $\mathcal{S}$ update for φ_m and the CVs resulting from $\mathcal{S}_{\mathrm{CG}}$ for φ^k_{m+1}.

Both the conventional string method and the multilevel string method find the transition pathway connecting the c7eq basin and the c7ax basin through the α_R basin. The multilevel string method clearly enforces convergence to the FG solution regardless of the CG model used (Figure 2.8).

To evaluate the speedup in the multilevel string method, we plot the RMSD in the CVs to the final position of the FG string after 200 iterations (Figure 2.8). Five replicate calculations were performed for the conventional string method, the multilevel string method with the good CG model as preconditioner, and the multilevel string method with the bad CG model as preconditioner (Figure 2.8). The multilevel string method with the good CG model as preconditioner requires on average 4.75-fold fewer iterations to reach an RMSD less than 1°. The mutlilevel string method with the bad CG model requires on average 3.8-fold fewer iterations to reach this RMSD. Similarly to the example with the string method on the Müller-Brown potentional, both the good and bad CG models significantly reduce the number of times $\mathcal{S}$ needs to be evaluated in the string method. The multilevel string method with the good CG model as preconditioer requires an average of 16 multilevel iterations to reach an RMSD of 1° which requires a total effort of 1.76 ns of all-atom molecular dynamics with the current implementation. This is significantly reduced from the total effort of 8.36 ns required to reach an RMSD of 1° using the conventional string method with the current implementation.

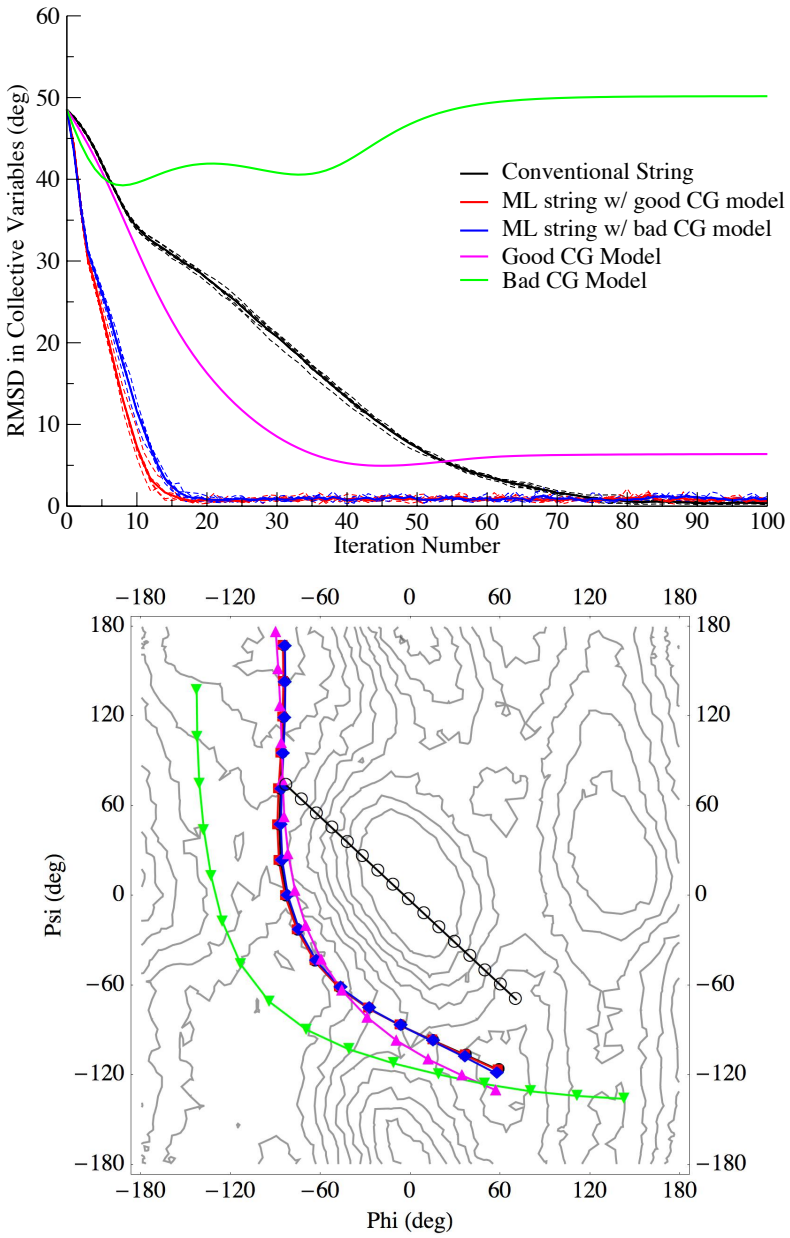


Figure 2.8: String method for finding pathways for the c7ax to c7eq transition of the alanine dipeptide. (top) Average RMSD in the CVs to the final position of the conventional string. Dotted lines indicate an individual run of the conventional or multilevel string while solid lines indicate the average RMSD. (bottom) Ramachandran plot of the converged string images for the conventional string method (black filled circles), the multilevel string with the the good CG model as preconditioner (red), the multilevel string with the bad CG model as preconditioner (blue), the good CG model (green), the bad CG model (magenta), and the initial string path (black unfilled circles). The contour lines are estimates of the free energy surface based on 400 ns of standard metadynamics simulations [39] performed in NAMD2.9. The metadynamics uses a hill height of 0.1 kcal/mol placed every 100 steps with all other parameters set to their default values in NAMD2.9.

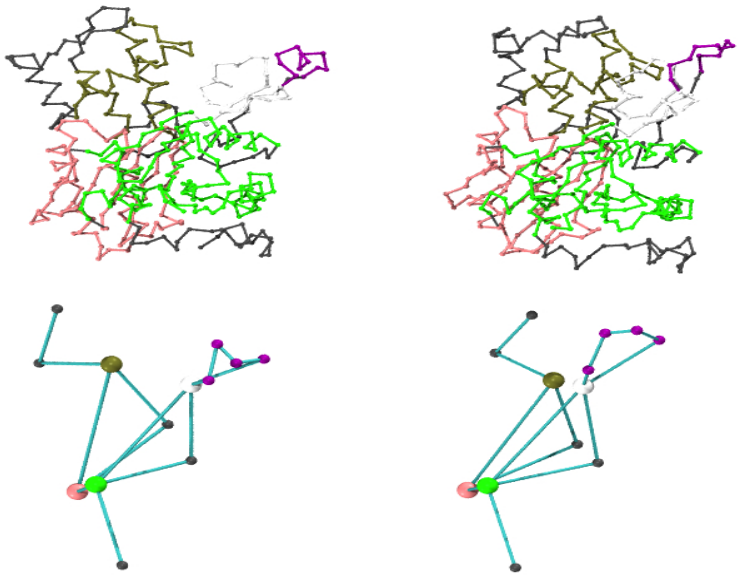


Figure 2.9: G-actin (left) and F-actin (right) endpoint structures in the (top) elastic network model and (bottom) projected representation used for the CVs. The subdomains are colored as follows: green=1, white=2, pink=3, tan=4, and purple=5-8 (D-loop).

G to F transition of an actin monomer

Actin is a highly abundant cytoskeletal protein that is important in many dynamic cellular processes. This protein exists in two forms in the cell: the globular (G-actin) and filamentous (F-actin) forms. Individual actin subunits change conformation upon polymerization; this conformational change is generally described as involving the relative rotation of the two main domains and the extension of the D-loop to facilitate intersubunit interactions [65, 26]. Here, as an illustration of our multilevel approach, we study the G- to F- transition in the isolated subunit (Fig. 2.9).

To ensure that there is a well-defined path for the string method to find for this process, we construct a two-state elastic network model following the procedure in ref [68]. The stable states are taken from the positions of the C^{α} atoms of crystal structures for G-actin (1J6Z) and F-actin (2ZWH) (Fig. 2.9). In constructing the model, we neglected unresolved residues in the N- and C-terminal regions of the 1J6Z structure and the methylated histidine

Table 2.1: Actin CV groupings

Site	Structural Feature	Residues
1	SD1	5-33; 80-147; 334-349
2	SD2	34-39; 52-69
3	SD3	148-179; 273-333
4	SD4	180-219; 252-262
5	D-loop	40-43
6	D-loop	44-45
7	D-loop	46-47
8	D-loop	48-51
9	Flap	236-251
10	H-loop	263-272
11	C-terminus	350-375
12	S-loop	70-79
13	SD4 hinge	220-235

at position 73; the resulting elastic network model consists of 368 C^{α} atoms.

The potential for a two-state elastic network model contains attractive terms centered on the input endpoints (Eq. (6) in ref. [68]) and a repulsive term (Eq. (9) in ref. [68]), both of which are functions of pairwise distances (all interactions are truncated at 11.5 Å). In order to make the problem more challenging for the conventional string method, we add a tunable "roughening" term to the pairwise repulsive term to yield

$$U^R(\mathbf{X}) = \epsilon \sum_{ij}^{N} \left[\left(\frac{\sigma}{\Delta x_{ij}} \right)^{12} + A_r \cos\left(\frac{2\pi \Delta x_{ij}}{L_r} \right) \right]. \tag{2.15}$$

The amplitude and period of the roughening term are $A_r = 0.0596$ kcal/mol and $L_r = 0.03$ Å. Otherwise, we use the same parameter values as [68], except the maximum value for the site-dependent force parameter ($k_{ij} = 2.0$ kcal/mol/Å^2), the energy parameter for the variable argument to that force parameter ($\varepsilon_k = 5.0$ kcal/mol), and the exponential mixing coefficient ($\beta_m = 0.015$), which modulates the height of the barrier between the two stable states.

The collective variables (CVs) for the string procedure comprise the Cartesian coordinates

of 13 sites at the centers of mass of 13 groups of C^{α} atoms as shown in Table 2.1 and Fig. 2.9. The grouping scheme builds on an existing 10-site CG model of the actin system that has been extensively studied [74, 75] but expands the D-loop from 1 to 4 sites. This added detail enables the CVs to better capture the transition of interest, which involves folding/unfolding of the D-loop helix (see the elastic network structures in Fig. 2.9).

The string has 32 images. We construct an initial path by first linearly interpolating the CV sites at the basins. Each CV site of image 16 was then perturbed by 3.0 Å in a random direction. Starting from each basin, a sequence of points in CV space was produced by linearly interpolating each basin and the perturbed middle image. The elastic network structures at the basins were then dragged along the pertubred path by moving the minima of harmonic potentials with a force constant of 100 kcal/mol/Å^2. Each step from image to image moved the harmonic potentials in 20 increments for a total of 20,000 steps with a timestep of 0.5 fs. This perturbed initial string was constructed in order to provide a more challenging problem for the optimization than a simple linearly interpolated initial path.

We use the string method with swarms of trajectories [68] with 250 copies of the system for each image to evaluate the string operator $\mathcal{S}$. All molecular dynamics is performed as Langevin dynamics with a friction coefficient of 30 ps^{-1} at 300 K unless otherwise noted. At each iteration, we do the following.

1. Update the images by dragging the elastic network structures toward the starting point in the CV space. The dragging step moves the minima of harmonic restraints with force constants of 100 kcal/mol/Å^2 towards the position of the starting point in CV space in 20 equal increments over a total of 10^4 steps of molecular dynamics with a timestep of 2.5 fs.

2. Evaluate $\mathcal{S}(\varphi_m)$. Each swarm copy is prepared by running 100 steps of molecular dynamics from the final structure from step 1 with a harmonic restraint with a force constant of 100 kcal/mol/Å^2 applied to the CV sites and a timestep of 2.5 fs. Another

100 steps of molecular dynamics with loose harmonic restraints with force constants of 1.0 kcal/mol/Å^2 applied to the CV sites and a timestep of 50.0 fs. The average CVs over the last 100 steps of each swarm trajectory are computed and then averaged between swarm trajectories to determine the new CVs.

3. Reparameterize the interpolated path to maintain approximately equidistant spacing between the images while smoothing the string as in ref. [88] with $\kappa = 0.1$.

To evaluate $\mathcal{S}_{\mathrm{CG}}$, we use a single trajectory of steepest descent (zero-temperature dynamics) on the unroughened elastic network surface with loose harmonic restraints with force constants of 1.0 kcal/mol/Å^2 applied to the CV sites. At each iteration we do the following.

1. Update the image positions by stepwise dragging the elastic network structures toward the starting point in the CV space. The dragging moves the minima of harmonic restraints with force constants of 100 kcal/mol/Å^2 in 20 increments over a total of 10^3 steps of steepest descent with a timestep of 2.5 fs.

2. Evaluate $\mathcal{S}_{\mathrm{CG}}(\varphi_m)$ with 10 steps of steepest descent and a timestep of 50.0 fs. The new CVs are taken from the final position of the trajectory.

3. Reparameterize the interpolated path to maintain approximately equidistant spacing between the images while smoothing the string as in ref. [88] with $\kappa = 0.1$.

We iterate Eq. (2.8) with $K = 5$ and $\Delta_m = 1.0$ using the CVs from step 1 of the previous $\mathcal{S}$ update for φ_m and the CVs resulting from $\mathcal{S}_{\mathrm{CG}}$ for φ^k_{m+1}.

To compare the paths obtained from the conventional and multilevel string methods qualitatively, we projected their outputs onto a variable space that captures the major structural differences between the endpoints: the relative rotation of the two main domains as described by the dihedral angle between CV sites 2, 1, 3 and 4 and the extension of the D-loop as described by the distance between CV site 2 and the CV site extended the farthest from SD

2 in the F-actin structure which is CV site 6. As shown in Fig. 2.10, in this representation the two paths are very similar—both show a clear separation between the early rotation of the two domains and the later extension of the D-loop. Experimental evidence suggests that F-actin is highly polymorphic in structure and that much of that polymorphism arises from the positioning of the D-loop [27, 18]. Consistent with this notion, the path obtained indicates that the flattened subunit can indeed adopt a variety of different D-loop configurations despite the simplicity of the two-state elastic network potential. This result suggests that the polymorphism arises mainly from the topology of molecular interactions, rather than a delicate balance of energetics.

To characterize the speedup in the multilevel string method over the conventional string method, we used the RMSD in CVs. The target structures for the RMSD calculation are determined from averaging the CVs over the final 200 iterations of a conventional string on an unroughened elastic network surface. The RMSD is calculated after aligning each image to the corresponding target image from the averaged unroughened string. Fig. 2.11 shows that the multilevel string method significantly accelerates the convergence of the path compared to the conventional string method; on average the preconditioning results in roughly a 3.5-fold decrease in computer time required to achieve an RMSD less than 0.3 Å (Fig. 2.11). This speedup is substantial given the relative speed with which the FG calculation itself converges; we expect greater accelerations for computationally costlier models.

2.4 Discussion

In the present paper, we introduce the idea of multilevel preconditioning for iterative molecular optimization calculations. The approach put forward here is mathematically similar to the parareal parallel-in-time integration algorithm first introduced in [50]. As described in [8], one can view the parareal algorithm as a fixed point iteration preconditioned by a less expensive model, and it can be derived by a change in variables analogous to that in Section

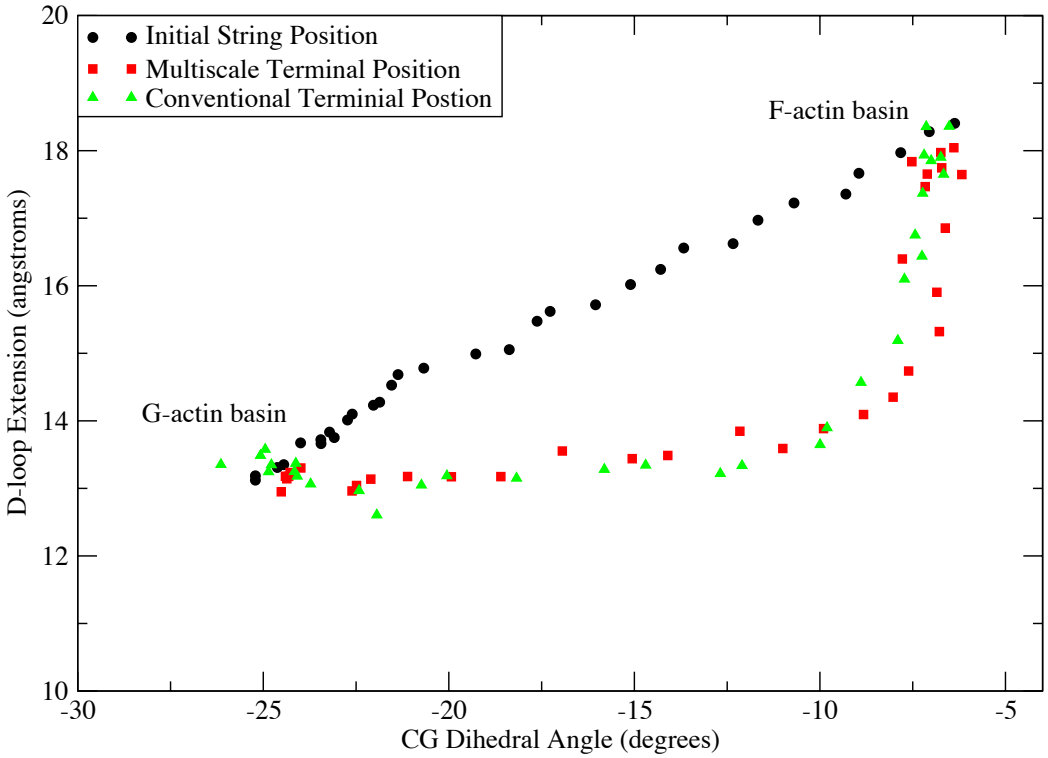


Figure 2.10: Comparison of paths obtained from the conventional and multilevel string methods. Projection of the paths onto variables that capture the major structural differences between the endpoints: the relative rotation of the two main domains as described by the dihedral angle between CV sites 2, 1, 3 and 4 and the extension of the D-loop as described by the distance between CV sites 2 and 6.

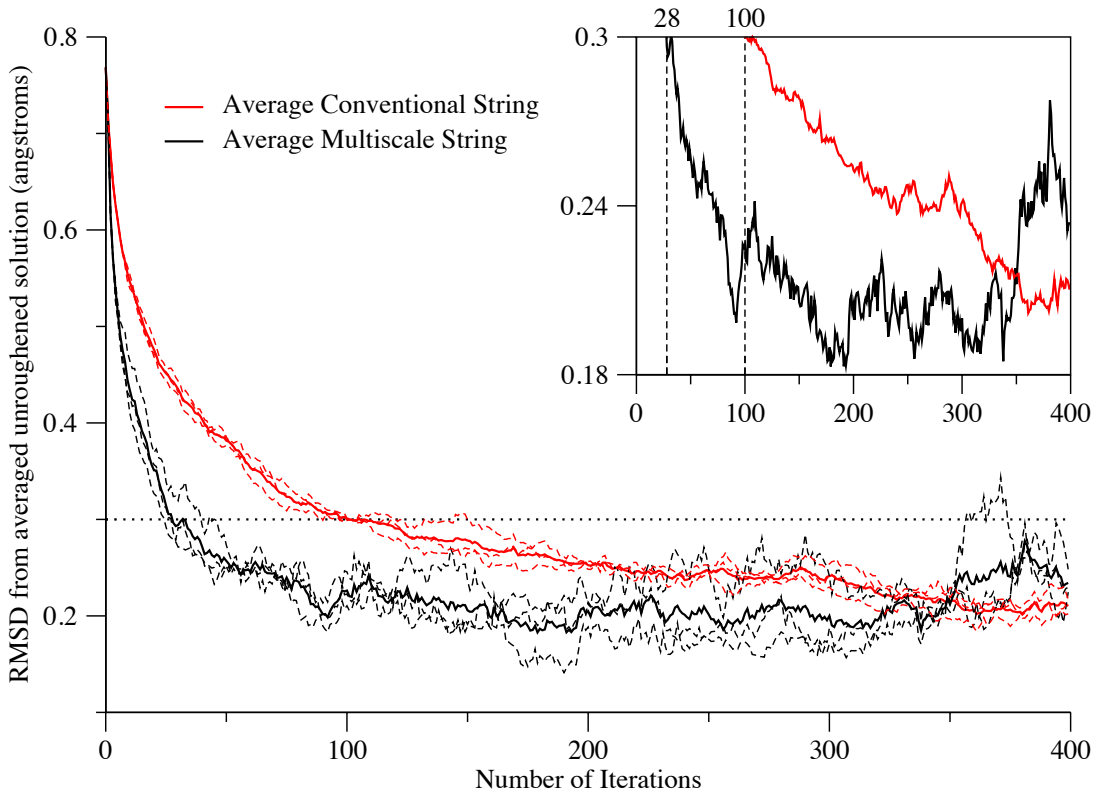


Figure 2.11: Average RMSD to the average string solution on an unroughened elastic network surface of the conventional string method (solid red) and the multilevel string method (solid black) for the actin system averaged over three independent runs of each method (dashed red and black lines).

2.2. Similar manipulations can be used to derive multigrid schemes [6, 31].

We demonstrate the application of our protocol to energy minimization and determination of transition paths. In the examples that we consider, a CG model is used to accelerate exploration of the landscape of an FG model. Two common aspects of CG models underlie the observed speedups: (1) CG potentials are less computationally costly than FG ones, and (2) CG potentials are generally smoother, allowing large changes in state at each iteration. The actual performance realized will depend on how these aspects are balanced with maintaining the correspondence to the FG model.

That said, a key feature of our approach is that it robustly converges to an FG solution for different choices of CG models and update operators. This feature is important as CG models can stabilize states different than the motivating FG model for many reasons, ranging from the choice of approximations to practical limitations (e.g., when sampling is required to estimate the parameters of the CG model). The flexibility with respect to CG models and update operators should also allow input of phenomenological information to focus sampling on states relevant to experimental observations when competing optima are available. By the same token, in the context of the string method, it could be advantageous to use an alternative method for proposing CG transition paths. For example, with modifications it may be possible to use steered dynamics [76, 35, 32] to generate candidate paths that were driven from the reactant to the product and restrained to be close to the existing state. Such a scheme could exploit the speedup associated with steered dynamics while correcting for its biases [35].

The basic philosophy of using a less computationally costly model to speed calculations with a more costly one is not new to molecular simulations. In particular, resolution replica exchange simulations [52, 53, 90, 51] build on the ideas of [81] to speed equilibrium sampling. Again, the essential idea is that CG models are smoother, allowing more rapid exploration of the state space. Our approach is slightly closer in spirit to [70], which uses a CG model to

drag the FG model as a means for generating candidate moves in a Monte Carlo simulation. With regard to sampling, it is important to note that the derivation of our approach is general and not specific to geometry optimization or the string method. With appropriate modifications, the approach of this paper could be used to accelerate other iterative schemes such as metadynamics [39] or nonequilibrium umbrella sampling [89, 16, 15, 13, 14].

A well-known challenge in coupling FG and CG models is that one must repeatedly reconstruct FG configurations from CG ones [53, 90, 51, 6]. When the models differ significantly in the numbers of degrees of freedom, this step can be computationally demanding. Given this issue, we expect that a particularly promising application of multilevel preconditioning strategies similar to the one described here will be in quantum chemical calculations. In that context one would have CG and FG models at the same resolution (number of atoms) but at two different levels of theory for the electronic structure calculations: an accurate but computationally expensive theory and an inaccurate but computationally inexpensive one [43]. We are investigating extensions of this nature presently.

CHAPTER 3

TRAJECTORY STRATIFICATION OF STOCHASTIC DYNAMICS

We present a general mathematical framework for trajectory stratification for simulating rare events. Trajectory stratification involves decomposing trajectories of the underlying process into fragments limited to restricted regions of state space (strata), computing averages over the distributions of the trajectory fragments within the strata with minimal communication between them, and combining those averages with appropriate weights to yield averages with respect to the original underlying process. Our framework reveals the full generality and flexibility of trajectory stratification, and it illuminates a common mathematical structure shared by existing algorithms for sampling rare events. We demonstrate the power of the framework by defining strata in terms of both points in time and path-dependent variables for efficiently estimating averages that were not previously tractable.

3.1 Introduction

Computer simulation is a powerful tool for the study of physical processes. Specifically, stochastic simulation methods have broad applicability in modeling physical systems in a variety of fields including chemistry, physics, economics, engineering and climate science [4, 29]. In many practical applications, the statistical properties of the process of interest are approximated by averages over many independent realizations of trajectories of the process, or, in the case of ergodic properties, by averages taken over a single very long trajectory of the process. However, for many systems, the most interesting events are those that occur most infrequently and are therefore very difficult to observe by direct numerical integration of the equations governing the dynamics. For example, in chemistry, the conformational changes responsible for the function of many molecules and, in climate science, extreme

events like severe droughts and violent hurricanes, occur on timescales orders of magnitude longer than the timestep for numerical integration. This basic observation has motivated the development of numerous techniques aimed at enhancing the sampling of rare events of interest without sacrificing statistical fidelity (see [25] for an account within the context of molecular simulation).

In this article, we depart from standard enhanced sampling approaches and develop a general mathematical and computational framework for the estimation of statistical averages involving rare trajectories of stochastic processes. Our approach can be viewed as a form of stratified sampling, long a cornerstone of experimental design in statistics (e.g., [62]). In stratified sampling, a population is divided into subgroups (strata), averages within those strata are computed separately, and then averages over the entire state space are assembled as weighted sums of the strata averages. Stratification also has a long history in computer simulations of condensed-phase systems as umbrella sampling (US) [83, 69, 11, 25, 49]. The key idea behind any stratified sampling strategy is that, when the strata are chosen appropriately, their statistics can be obtained accurately with relatively low effort and combined to estimate the average of interest with (much) less overall effort than directly sampling the stochastic process to the same statistical precision. Here we show that the trajectories of an arbitrary discrete-time Markov process (including many dynamics with memory, so long as they can be written as a suitable mapping) can also be stratified: they can be decomposed into fragments restricted to regions of trajectory space (strata), averages over the distributions of trajectory fragments within the strata can be computed with limited communication between them, and those averages can be combined in a weighted fashion to yield a very broad range of statistics that characterize the dynamics.

These basic features are at the core of the existing nonequilibrium umbrella sampling (NEUS) method [89, 15, 88], which forms the starting point for our development. NEUS was originally introduced to estimate stationary averages with respect to a given, possibly

irreversible, stochastic process [89]. Starting in [15, 88] it was observed that the general NEUS approach was applicable to certain dynamical averages as well. At its most basic level, NEUS relies on duplication of states in rarely visited regions of space and subsequent forward evolution of the duplicated states. In this way it is similar to a long list of so-called "trajectory splitting" techniques [30, 36, 34, 86, 2, 42, 10, 32, 33] that are also able to compute averages of dynamic quantities. Like NEUS, splitting techniques also often involve a decomposition of state space into regions. Unlike NEUS however, in most splitting techniques bias is removed through the use of a separate weight factor for each individual sample (rather than for an entire region), and the computational effort expended in each region is not controlled exactly. What makes the NEUS method unique among splitting techniques is that it is also a trajectory stratification strategy.

Our goal in this article is to provide a clear and general mathematical framework for trajectory stratification that builds upon the NEUS method. In the process we clearly delineate the range of statistics that can be estimated by NEUS, including more general quantities than previously computed. Our analysis of the underlying mathematical structure of US [82, 17] has already facilitated the derivation of a central limit theorem for US and a detailed understanding of its error properties. Here, our framework reveals unanticipated connections between the equilibrium and nonequilibrium US methods and places the nonequilibrium algorithm within the well studied family of stochastic approximation methods [46]. The analysis leads to a practical scheme that departs dramatically from currently available alternatives. We demonstrate the use of trajectory stratification to compute a hitting time distribution as well as to compute the expectation of a path-dependent functional that gives the relative normalization constants for two arbitrary, user-specified unnormalized probability densities.

3.2 A Unified Framework

In this section we present a framework that reveals the unified structure underlying umbrella sampling in both the equilibrium and nonequilibrium case. In 3.2.1, we review the equilibrium approach [82, 17] to introduce terminology and the central eigenproblem in a context where the analogies to traditional umbrella sampling descriptions [83, 69, 11, 25, 49] are readily apparent. In 3.2.2, we present the nonequilibrium version of the algorithm and show how this interpretation results in a flexible scheme for computing dynamic averages. As for its equilibrium counterpart, an eigenproblem lies at the core of the nonequilibrium method. This eigenproblem however, involves a matrix that depends on the desired eigenvector, introducing the need for a self-consistent iteration. In 3.3, we give a precise description of the fixed-point problem solved by this iteration and show that the algorithm is an example of a stochastic approximation strategy [46]. In 3.4 we specialize our development to the context of steady-state averages that motivated the original development of NEUS [89].

3.2.1 Averages with Respect to a Specified Density

Our presentation in this section follows [82]. We view umbrella sampling as a method to compute averages of the form

$$\int_{x\in\mathbb{R}^d} f(x)\pi(dx), \tag{3.1}$$

where π is a known probability distribution and d is the dimension of the underlying system (e.g., the total number of position coordinates for all atoms in a molecular system). For example, π might be the canonical distribution, $\pi(dx) \propto e^{-\beta V(x)}dx$ where V is a potential energy function, β is an inverse temperature, and f might be 1 on some set A and 0 elsewhere. In this case, $-\beta^{-1}\log\int f(x)\pi(dx)$ can be regarded as the free energy of the set A.

Note that in our notation π is a probability measure on $\mathbb{R}^d$ and dx is an infinitesimal volume element in $\mathbb{R}^d$. If the distribution π has a density function $p(x)$ then $\pi(A) = \int_{x\in A} p(x)dx$

and, in particular, $\pi(dx) = p(x)dx$. This more general notation is useful when we move to our description of the nonequilibrium umbrella sampling scheme. As an aid to the reader, we choose to introduce it in the simpler setting of this section.

Consistent with traditional implementations of US [69, 25], we divide the computation of the average in (3.1) into a series of averages over local subsets of space. More precisely, instead of directly computing averages with respect to π, we will compute averages with respect to n probability distributions, π_j, each of which concentrates probability in a restricted region of space (relative to π itself) with the goal of eliminating or reducing barriers to efficient sampling associated with π. So that general averages with respect to π can be assembled, the π_j will satisfy $\pi = \sum_{j=1}^{n} z_j \, \pi_j$ for a set of weights z_j to be defined in a moment.

To obtain the restricted distributions π_j we can set

$$\pi_j(dx) = \frac{\psi_j(x)\pi(dx)}{\int_{y\in\mathbb{R}^d} \psi_j(y)\pi(dy)}, \tag{3.2}$$

where the ψ_j are non-negative user defined functions satisfying $\sum_{j=1}^{n} \psi_j(x) = 1$ for all x (this last requirement is relaxed in [82]). For example, one might choose $\psi_j(x) = \mathbf{1}_{A_j}(x)/\sum_{\ell=1}^{n} \mathbf{1}_{A_\ell}(x)$, where the A_j are a collection of sets covering the space to be sampled, and, for any set A_j, the function $\mathbf{1}_{A_j}(x)$ is 1 if $x \in A_j$ and 0 otherwise.

Note that $\pi = \sum_{j=1}^{n} z_j \, \pi_j$ is satisfied with

$$z_j = \int_{x\in\mathbb{R}^d} \psi_j(x)\pi(dx) \tag{3.3}$$

and that the average (3.1) with respect to π can be reconstructed using the equation

$$\int_{x\in\mathbb{R}^d} f(x)\pi(dx) = \sum_{j=1}^{n} z_j \, \langle f \rangle_j, \tag{3.4}$$

with

$$\langle f \rangle_j = \int_{x \in \mathbb{R}^d} f(x) \pi_j(dx). \tag{3.5}$$

Here z_j is the statistical weight associated with each distribution π_j and $\langle f \rangle_j$ are the averages of the observable f against π_j. From (3.4) we see that if we can sample from the π_j and compute the z_j then we can compute averages with respect to π. Since π_j is known explicitly in this case, it can be sampled by standard means (e.g., Langevin dynamics or Metropolis Monte Carlo [25]).

Our key observation underpinning the equilibrium umbrella sampling method is that the z_j themselves are functions of averages with respect to the local distributions π_j:

$$z_j = \sum_{i=1}^{n} z_i F_{ij} \quad \text{and} \quad \sum_{j=1}^{n} z_j = 1, \tag{3.6}$$

where

$$F_{ij} = \int_{x \in \mathbb{R}^d} \psi_j(x) \pi_i(dx). \tag{3.7}$$

The matrix F is stochastic (i.e., has non-negative entries with rows that sum to 1) and (3.6), which is written in matrix-vector form as

$$z^{\mathrm{T}} F = z^{\mathrm{T}} \quad \text{and} \quad \sum_{j=1}^{n} z_j = 1, \tag{3.8}$$

is an eigenproblem that can be solved easily for the vector z.

We now have a stratification scheme for computing the target average in (3.1) by sampling from the distributions π_j. Operationally, the main steps are as follows.

1. Assemble F defined in (3.7) (or the alternative in A below) and $\langle f \rangle_j$ defined in (3.5) by sampling from π_j defined in (3.2).

2. Solve eigenvector equation (3.8) for z defined in (3.3).

3. Compute the desired expectation via (3.4).

The efficiency of this *equilibrium* US scheme has been analyzed in detail elsewhere [82, 17]. Roughly, the benefit of US is due to the facts that averages with respect to the π_j are often sufficient to solve for all desired quantities, and one can choose ψ_j so that averages with respect to the π_j converge much more quickly than averages with respect to π itself. It is this basic philosophy that we extend in 3.2.2 to the computation of dynamic averages.

3.2.2 Averages with Respect to a Given Markov Process

The mathematical description of the nonequilibrium umbrella sampling scheme that follows reveals how the stratification strategy developed for the equilibrium case in 3.2.1 can be extended to compute nearly arbitrary dynamic statistics. Our interest in this section is computing averages over trajectories of some specified Markov process, $X^{(t)}$. This process can be time-inhomogenous, i.e., given the value of $X^{(t)}$, the distribution of $X^{(t+1)}$ can depend on the value of t. We compute averages of trajectories evolved up to a first exit time of the process, $(t, X^{(t)})$, from a user specified set of times and positions, D—i.e., trajectories terminate when they first leave the set D. We consider averages over trajectories of $X^{(t)}$ run until time

$$\tau = \min\{t > 0 : (t, X^{(t)}) \notin D\} \tag{3.9}$$

for a set $D \in \mathbb{N} \times \mathbb{R}^d$. In the first of our numerical examples in 3.5, D is a set of times and positions for which we would like to compute an escape probability. In our second numerical

example D restricts only the times over which we simulate. The averages are of the form

$$\mathbf{E}\left[\sum_{t=0}^{\tau-1} f(t, X^{(t)})\right]. \tag{3.10}$$

We note that the average in (3.10) is not completely general, in order to streamline the developments below. Without any modification, we can compute averages similar to (3.10) but with the argument $(t, X^{(t)})$ in the definitions of τ and f replaced by $(t, X^{(t-1)}, X^{(t)})$. On the other hand, expectations with $(t, X^{(t)})$ replaced by $(t, X^{(t-m)}, \ldots, X^{(t-1)}, X^{(t)})$ for $m \geq 2$ cannot be obtained immediately. These and many more general expectations can, however, be accommodated by applying the algorithm to an enlarged process (e.g., $(t, X^{(t-m)}, \ldots, X^{(t-1)}, X^{(t)})$) at the cost of storing copies (as described below) of the enlarged process. For many expectations, this cost is quite manageable. Finally, we require that $\mathbf{E}[\tau] < \infty$. The limit $\tau \to \infty$ will be considered in 3.4.

Below we show that expectations of time-dependent functions can be decomposed as a weighted sum of expectations computed over restricted subsets of the full space and, in turn, how the statistical weights can be computed as expectations over these subsets, mirroring the basic structure of the equilibrium scheme described in 3.2.1. However, as we discuss in 3.3, the algorithm for computing these local expectations departs significantly from the equilibrium case because their form is not known *a priori* in the nonequilibrium setting.

The Index Process

The US scheme in 3.2.1 used the basis functions ψ_j to stratify the sampling of the distribution π by decomposing averages with respect to π into averages with respect to the more easily sampled π_j. To arrive at an analogous partitioning of state space for the nonequilibrium case, we first need to introduce an *index process* $J^{(t)}$ that takes values in $\{1, 2, \ldots, n\}$ and (roughly) labels the point $(t, X^{(t)})$ in time and space, $\mathbb{N} \times \mathbb{R}^d$. Our objective is to generate

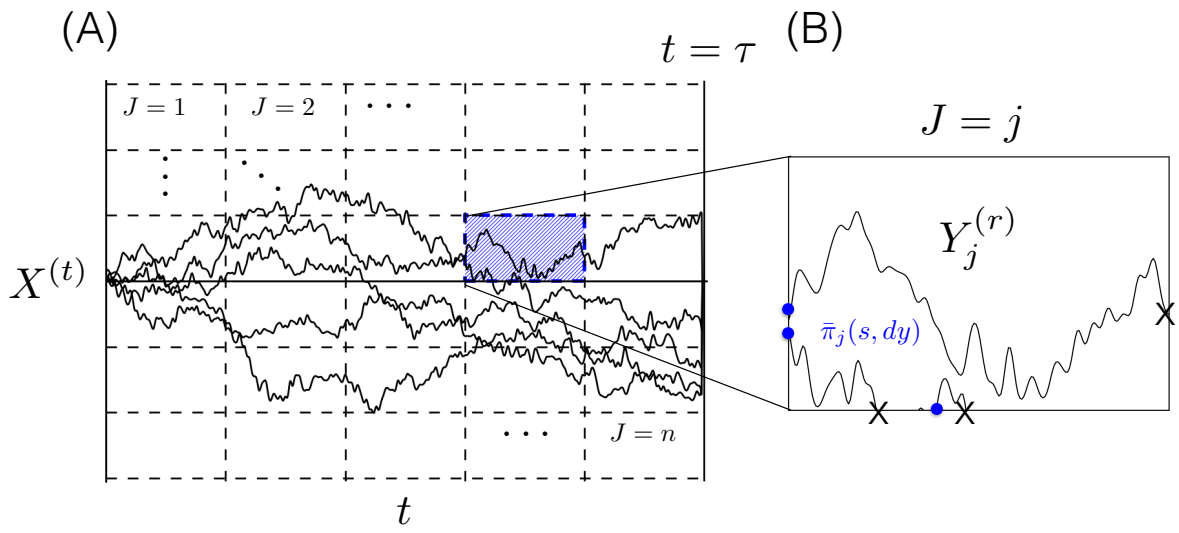


Figure 3.1: Illustration of the stratification of a process $(X^{(t)}, J^{(t)})$ (black lines, left panel) via the scheme outlined in 3.2.2. (A) The restricted distributions corresponding to each value of the index process $J^{(t)}$ are outlined as discrete regions of the $(t, X^{(t)})$ space (left panel, black dashed lines). In this depiction, the value of $J^{(t)}$ corresponds to the current cell containing $(t, X^{(t)})$ within a rectangular grid of time and position. (B) Each of the restricted distributions $\pi_j(t, dx)$ are sampled by integrating a locally restricted dynamics $Y_j^{(r)}$ (right panel, black lines). The $Y_j^{(r)}$ process is generated by integrating sequential fragments of the unbiased process $(X^{(t)}, J^{(t)})$ corresponding to a particular fixed value of $J = j$ (left panel). As each fragment transitions from $J = i$ to $J = j$ with $j \neq i$, the dynamics are stopped and restarted at a time and point (s, y) (left panel, blue dots) drawn from the flux distribution $\bar{\pi}_j(s, dy)$.

fragments of trajectories of $X^{(t)}$ consistent with specific values of $J^{(t)}$ thereby breaking the coupled process $(X^{(t)}, J^{(t)})$ into separate regions corresponding to a given value of $J^{(t)}$ (see panel A of Figure 3.1).

The idea of discretizing a process $X^{(t)}$ according to the value of some user-specified index process is not new in computational statistical mechanics. For example, in our notation, given a partition of state space $A_1, A_2, \ldots, A_n$, the Milestoning procedure [23] and some Markov State Modeling procedures [78] correspond to an index process that marks the pairs of sets (A_i, A_j) for $i \neq j$ between which $X^{(t)}$ last transitioned. In the Milestoning method, the pairs

of sets are considered unordered, so that a transition from A_j to A_i immediately following a transition from A_i to A_j does not correspond to a change in $J^{(t)}$ and $J^{(t)}$ can assume $n = \binom{m}{2}$ distinct values. The original presentation of NEUS on the other hand corresponds to a process $J^{(t)}$ which marks the index of the set A_j containing $X^{(t)}$. For accurate results, the Milestoning procedure requires that the index process $J^{(t)}$ itself be Markovian. Even under the best circumstances, that assumption is only expected to hold approximately. It is not required by the NEUS algorithm. Our presentation below reveals the full flexibility in the choice of $J^{(t)}$ within NEUS. That flexibility is essential in the generalized setting of this article.

In the developments below we require that $J^{(t)}$ is chosen so that the *joint process* $(X^{(t)}, J^{(t)})$ is Markovian. This assumption allows that trajectories can be continued beyond a single transition event (before τ) without additional information about the history of $X^{(t)}$ or $J^{(t)}$. We do not assume that $J^{(t)}$ alone is Markovian and in general it will not be. Our assumption implies no practical restriction on the underlying Markov process $X^{(t)}$. When $X^{(t)}$ is non-Markovian, additional variables can often be appended to $X^{(t)}$ to yield a new Markov process to which the developments below can be applied. A version of this idea is applied in 3.5.4 where we append a variable representing a nonequilibrium work to an underlying Markov process.

The Eigenproblem

Given a specific choice of index process $J^{(t)}$, the nonequilibrium umbrella sampling algorithm stratifies trajectories of $X^{(t)}$ according to their corresponding values of $J^{(t)}$. That is, for each possible value of the index process, NEUS generates segments of trajectories of $X^{(t)}$ between the times that $J^{(t)}$ transitions to and from $J = i$. To make this idea more precise,

we need to carefully describe the distribution sampled by these trajectory fragments:

$$\pi_j(t, dx) = \frac{\mathbf{P}\left[t < \tau,\, X^{(t)} \in dx,\, J^{(t)} = j\right]}{z_j}, \tag{3.11}$$

where

$$z_j = \sum_{t=0}^{\infty} \mathbf{P}\left[t < \tau,\, J^{(t)} = j\right]. \tag{3.12}$$

For each j, π_j is the distribution of time and position pairs $(t, X^{(t)})$ conditioned on $J^{(t)} = j$ and $t < \tau$. We call the π_j *restricted distributions*. We have reused the notations π_j and z_j from our account of the equilibrium umbrella sampling scheme to emphasize the analogous roles played by those objects in both sections. Note that here we are treating time as an additional random variable. Also note that in these definitions as well as in the formulas below, $\mathbf{P}$ and $\mathbf{E}$ represent probabilities and expectations with respect to the original, unbiased $X^{(t)}$ and $J^{(t)}$. We assume that $z_j > 0$ for all j since we can remove the index j from consideration if $z_j = 0$. The z_j are all finite because $\sum_{j=1}^{n} z_j = \mathbf{E}[\tau]$, which we assume is finite.

Observe that

$$\begin{aligned}
\mathbf{E}\left[\sum_{t=0}^{\tau-1} f(t, X^{(t)})\right] &= \sum_{t=0}^{\infty} \mathbf{E}\left[f\left(t, X^{(t)}\right),\, t < \tau\right] \\
&= \sum_{j=1}^{n} \sum_{t=0}^{\infty} \int_{x \in \mathbb{R}^d} f(t, x) \mathbf{P}\left[t < \tau,\, X^{(t)} \in dx,\, J^{(t)} = j\right] \\
&= \sum_{j=1}^{n} z_j \langle f \rangle_j,
\end{aligned}$$

where

$$\langle f \rangle_j = \sum_{t=0}^{\infty} \int_{x \in \mathbb{R}^d} f(t, x) \pi_j(t, dx). \tag{3.13}$$

Thus we have a decomposition of (3.10) analogous to the decomposition of (3.1) in (3.4).

Also as in the equilibrium case, the z_j can be computed from averages with respect to the π_j. To see this, observe that for any t we can write

$$\sum_{i=1}^{n} \mathbf{P}\left[t+1<\tau,\, J^{(t+1)}=j,\, J^{(t)}=i\right] = \mathbf{P}\left[t+1<\tau,\, J^{(t+1)}=j\right]. \tag{3.14}$$

Summing this expression over t we obtain

$$\sum_{i=1}^{n}\sum_{t=0}^{\infty} \mathbf{P}\left[t+1<\tau,\, J^{(t+1)}=j,\, J^{(t)}=i\right] = \sum_{t=0}^{\infty} \mathbf{P}\left[t<\tau,\, J^{(t)}=j\right] - \mathbf{P}\left[J^{(0)}=j\right]. \tag{3.15}$$

These expressions are all bounded by $\mathbf{E}[\tau]$ and are therefore finite. Expression (3.15) can be rewritten as an affine eigenequation:

$$z^{\mathrm{T}} G + a^{\mathrm{T}} = z^{\mathrm{T}}, \tag{3.16}$$

where z is defined in (3.12),

$$G_{ij} = \frac{\sum_{t=0}^{\infty} \mathbf{P}\left[t+1<\tau,\, J^{(t+1)}=j,\, J^{(t)}=i\right]}{z_i}, \tag{3.17}$$

and

$$a_j = \mathbf{P}\left[J^{(0)}=j\right]. \tag{3.18}$$

Equation 3.16 is the analog of (3.8) in 3.2.1. Here, the matrix element G_{ij} stores the expected number of transitions between the values of $J^{(t)}$, normalized by the expected number of time steps with $J=i$. Note that the matrix G is substochastic; that is, it has non-negative entries and rows that sum to a number less than or equal to one.

To complete the analogy with the umbrella sampling schemes described in 3.2.1, we need

to show that the elements of the matrix G are expressible as expectations over the π_j. Indeed,

$$\begin{aligned} G_{ij} &= \frac{1}{z_i} \int_{x \in \mathbb{R}^d} \sum_{t=0}^{\infty} \mathbf{P}_{t,x,i} \left[t+1 < \tau,\ J^{(t+1)} = j \right] \mathbf{P} \left[t < \tau,\ X^{(t)} \in dx,\ J^{(t)} = i \right] \\ &= \sum_{t=0}^{\infty} \int_{x \in \mathbb{R}^d} \mathbf{P}_{t,x,i} \left[t+1 < \tau,\ J^{(t+1)} = j \right] \pi_i(t, dx) \end{aligned} \tag{3.19}$$

where $\mathbf{P}_{t,x,i}$ is used to denote probabilities with respect to X initialized at time and position (t, x) and conditioned on $J^{(t)} = i$ and $t < \tau$. Note that in the first line we have appealed to the Markovian assumption on $(X^{(t)}, J^{(t)})$. Had we instead assumed that $J^{(t)}$ alone was Markovian, we could have ignored the x dependence in (3.19).

Just as for the umbrella sampling algorithm described in 3.2.1, we arrive at a procedure for computing (3.10) via stratification:

1. Assemble G_{ij} defined in (3.17) and $\langle f \rangle_j$ defined in (3.13) by sampling from the π_j defined in (3.11).

2. Solve the affine eigenvector equation (3.16) for z defined in (3.12).

3. Compute
$$\mathbf{E} \left[\sum_{t=0}^{\tau-1} f(t, X^{(t)}) \right] = \sum_{j=1}^{n} z_j \langle f \rangle_j$$
as in (3.13).

Relative to the scheme in 3.2.1, sampling the restricted distributions π_j requires a more complicated procedure. This is the subject of 3.3.

Rapid convergence of the scheme rests on the choice of $J^{(t)}$. Roughly, one should choose the index process so that the variation in estimates of the required averages with respect to the π_j (e.g., estimates of the G_{ij}) are small. In practice, this requires that transitions between values of $J^{(t)}$ are frequent, which is the analog of selecting the biases in equilibrium US to limit the range of the free energy over each subset of state space (see [82, 17]). We defer further specification of $J^{(t)}$ to 3.5, where we describe this and other important implementation details in the context of particular applications.

3.3 A General NEUS Fixed-Point Iteration

In this section we present a detailed algorithm for computing (3.10) by the stratification approach outlined in 3.2.2. To accomplish this one must be able to generate samples from the restricted distributions $\pi_j(t, dx)$. In NEUS, the restricted distributions are sampled by introducing a set of Markov processes $Y_j^{(r)}$ whose trajectories consist of sequential segments of the process $(t, X^{(t)})$ such that $J = j$. We use the time variable r to distinguish between the time associated with the underlying process $(t, X^{(t)})$ and the time r associated with the process $Y_j^{(r)}$. The trajectories of $Y_j^{(r)}$ are generated by integrating the underlying process $(X^{(t)}, J^{(t)})$ forward in time until a segment leaves the state $J = j$ or reaches the stopping time τ at which point the dynamics are stopped and restarted at a random time s and position y drawn from a distribution $\bar{\pi}_j(s, dy)$. For $Y_j^{(r)}$ to preserve the restricted distribution $\pi_j(t, dx)$, $\bar{\pi}_j(s, dy)$ must be the distributions of times s and positions y at which the process $(X^{(t)}, J^{(t)})$ transitions from a state $J^{(s-1)} = i$ with $i \neq j$ to state $J^{(s)} = j$ (see B). We call these distributions the *flux distributions*. The $Y_j^{(r)}$ process is illustrated in 3.1 for a particular choice of index process.

In general, the flux distributions $\bar{\pi}_j(s, dy)$ for which $Y_j^{(r)}$ preserves the restricted distribution $\pi_j(t, dx)$ are not known *a priori* and must be computed approximately. In 3.3.1, we define the flux distributions carefully and express them as a weighted sum of *conditional*

flux distributions $\gamma_{ij}(s, dy)$ which are a set of time and position distributions conditioned on the process $(X^{(t)}, J^{(t)})$ switching from a specific $J = i$ to $J = j$. In order to construct an estimate of $\bar{\pi}_j(s, dy)$ from estimates of the $\gamma_{ij}(s, dy)$ it is necessary to compute the flux $z_i\, G_{ij}$ of the process $(X^{(t)}, J^{(t)})$ from each restriction with $J = i$, $i \neq j$. The elements of the desired matrix G (and the corresponding weights z computed from (3.16)) are computed from expectations over the restricted distribution $\pi_j(t, dx)$ and sampled by integrating the process $Y_j^{(r)}$. This relationship between the weights z_j and the flux distributions $\bar{\pi}_j(s, dy)$ motivates a self-consistent iteration in which we solve for the flux distributions $\bar{\pi}_j(s, dy)$ and the matrix G, yielding successively better approximations of the flux distributions and the matrix G simultaneously. In 3.3.2, we derive a fixed-point equation that expresses the relationship between the flux distributions and the transition matrix G that motivates this self-consistent iteration. In 3.3.3, we describe the complete NEUS algorithm in detail and interpret it as a stochastic approximation algorithm [46] for solving the fixed-point equation derived in 3.3.2. In the Supplementary Material, we analyze a simple four-site Markov model to clearly illustrate the structure of this self-consistent iteration and the terminology of the framework.

3.3.1 The Flux Distributions

Before deriving the fixed-point problem and the corresponding stochastic approximation algorithm, we define the flux distributions $\bar{\pi}_j(s, dy)$ precisely. First, define the time of the ℓth change in the value of $J^{(t)}$ for a given realization of the coupled process $(X^{(t)}, J^{(t)})$ as

$$S^{(\ell)} = \min\{s > S^{(\ell-1)} : J^{(s)} \neq J^{(S^{(\ell-1)})}\} \tag{3.20}$$

for $\ell > 0$, and $S^{(0)} = 0$. Note that for $i \neq j$,

$$z_i \, G_{ij} = \sum_{\ell=0}^{\infty} \mathbf{P}\left[S^{(\ell+1)} < \tau,\, J^{(S^{(\ell+1)})} = j,\, J^{(S^{(\ell)})} = i\right] \tag{3.21}$$

is the net probability flux between values of the process $J^{(t)}$ before time τ.

Now let $\bar{\pi}_j(s, dy)$ be the distribution of time and position pairs $(S^{(\ell)}, X^{(S^{(\ell)})})$ conditioned on $J^{(S^{(\ell)})} = j$. Specifically, define

$$\bar{\pi}_j(s, dy) \propto \sum_{\ell=0}^{\infty} \mathbf{P}\left[S^{(\ell)} = s,\, s < \tau,\, X^{(s)} \in dy,\, J^{(s)} = j\right]. \tag{3.22}$$

We can then write

$$\pi_j(t, dx) \propto \sum_{s=0}^{t} \int_{y \in \mathbb{R}^d} \mathbf{P}_{s,y,j}\left[t < \sigma(s) \vee \tau, X^{(t)} \in dx\right] \bar{\pi}_j(s, dy) \tag{3.23}$$

where

$$\sigma(s) = \min\{r > s :\, J^{(r)} \neq J^{(s)}\}$$

and $x \vee y = \min\{x, y\}$. Instead of working directly with the flux distributions, we find it convenient to express both the fixed-point problem and the algorithm in terms of the probability distribution of time and position pairs $(t, X^{(t)})$ conditioned on observing a transition from $J = i$ to $J = j$ at time t, i.e., in terms of

$$\gamma_{ij}(s, dy) \propto \int_{x \in \mathbb{R}^d} \mathbf{P}_{s-1,x,i}[s < \tau,\, X^{(s)} \in dy,\, J^{(s)} = j]\, \pi_i(s-1, dx) \tag{3.24}$$

which is defined only for $s > 0$. To simplify notation, we let γ denote the set of all distributions γ_{ij}. Note that $z_j(1 - G_{jj}) = \sum_{\ell=0}^{\infty} \mathbf{P}[J(S^{(\ell)}), S^{(\ell)} < \tau]$ is the total flux into $J = j$ and that $z_i G_{ij}$ is the conditional flux from $J = i$ to $J = j$. The following simple but crucial

identity relates γ to the flux distributions $\bar{\pi}_j$:

$$\bar{\pi}_j(s, dy) = \frac{1}{z_j(1 - G_{jj})} \begin{cases} \sum_{i \neq j} z_i \, G_{ij} \, \gamma_{ij}(s, dy), & \text{if } s > 0 \\ a_j \, \mathbf{P}[X^{(0)} \in dy \mid J^{(0)} = j] & \text{if } s = 0. \end{cases} \tag{3.25}$$

The $s > 0$ term is the contribution from transitions into state $J = j$ from the neighboring state $J = i$, and the $s = 0$ term accounts for the initial $t = 0$ contribution of the underlying process when $J = j$. We emphasize that both the fixed-point problem and the iteration that we define below could be expressed in terms of the flux distributions $\bar{\pi}_j$ instead of γ. We choose to express them in terms of γ because the resulting formalism more naturally captures the implementation of the method used to generate our numerical results in 3.5.

3.3.2 The Fixed-Point Problem

We now derive the fixed-point problem. Our goal is to find an expression of the form

$$(\mathcal{G}(G, \gamma), \Gamma(G, \gamma)) = (G, \gamma) \tag{3.26}$$

that characterizes the desired matrix G and collection of probability measures γ as the fixed-point of a pair of maps $\mathcal{G}(\tilde{G}, \tilde{\gamma})$ and $\Gamma(\tilde{G}, \tilde{\gamma})$ that take as arguments approximations $\tilde{G}$ of G and $\tilde{\gamma}$ of γ and return, respectively, a new substochastic matrix and a new collection of probability measures.

To this end, we define functions mapping $\tilde{G}$ and $\tilde{\gamma}$ to approximations of the flux distributions $\bar{\pi}_j$ and the restricted distributions π_j. We denote these functions by the corresponding capital letters $\bar{\Pi}_j$ and Π_j. Based on (3.25) and (3.16), we define

$$\bar{\Pi}_j(s, dy; \tilde{G}, \tilde{\gamma}) = \frac{1}{\tilde{z}_j \, (1 - \tilde{G}_{jj})} \begin{cases} \sum_{i \neq j} \tilde{z}_i \, \tilde{G}_{ij} \, \tilde{\gamma}_{ij}(s, dy) & \text{if } s > 0, \\ a_j \, \mathbf{P}[X^{(0)} \in dy \mid J^{(0)} = j] & \text{if } s = 0, \end{cases} \tag{3.27}$$

where $\tilde{z}$ solves the equation $\tilde{z}^{\mathrm{T}} = \tilde{z}^{\mathrm{T}}\tilde{G} + a^{\mathrm{T}}$. The matrix $\tilde{G}$ is strictly substochastic. We assume that $\tilde{G}$ is also irreducible in which case the solution $\tilde{z}$ exists and is unique. To motivate the definition above, we observe that for the exact values G and γ, $\bar{\pi}_j(s, dy) = \bar{\Pi}_j(s, dy; G, \gamma)$ by (3.25). Moreover, given $\tilde{G}$ and samples from $\tilde{\gamma}$, one can generate samples from $\bar{\Pi}_j(s, dy; \tilde{G}, \tilde{\gamma})$; see 3.3.3. This is crucial in developing a practical algorithm to solve the fixed-point problem.

Similarly, we define

$$\Pi_j(t, dx; \tilde{G}, \tilde{\gamma}) \propto \sum_{s=0}^{t} \int_{y \in \mathbb{R}^d} \mathbf{P}_{s,y,j} \left[t < \sigma(s) \vee \tau,\, X^{(t)} \in dx \right] \bar{\Pi}_j(s, dy; \tilde{G}, \tilde{\gamma}). \tag{3.28}$$

Definition (3.28) is motivated by (3.23) and the same two considerations as (3.27): First,

$$\Pi_j(t, dx; G, \gamma) = \pi_j(t, dx);$$

this follows directly from the definition of Π_j using that $\bar{\Pi}_j(s, dy; G, \gamma) = \bar{\pi}_j(s, dy)$. Second, the distribution $\Pi_j(t, dx; \tilde{G}, \tilde{\gamma})$ is invariant for a process similar to the process $Y_j^{(r)}$ mentioned above, except with times and positions drawn from $\bar{\Pi}_j(s, dy; \tilde{G}, \tilde{\gamma})$ instead of $\bar{\pi}_j(s, dy)$ on leaving the state $J = j$; see 3.3.3. Thus, given $\tilde{G}$ and samples from $\tilde{\gamma}$, one can generate samples from $\Pi_j(t, dx; \tilde{G}, \tilde{\gamma})$.

At this point we are ready to define the functions $\mathcal{G}$ and Γ appearing in (3.26) above. For a substochastic matrix $\tilde{G}$ and a collection of probability distributions $\tilde{\gamma} = \{\tilde{\gamma}_{ij}\}$, define the substochastic matrix

$$\begin{aligned} \mathcal{G}_{ij}(\tilde{G}, \tilde{\gamma}) = \sum_{t=0}^{\infty} \int_{x \in \mathbb{R}^d} \mathbf{P}_{t,x,i}[t+1 < \tau,\, J^{(t+1)} = j] \\ \times \Pi_i(t, dx; \tilde{G}, \tilde{\gamma}) \end{aligned} \tag{3.29}$$

and the collection of probability distributions

$$\begin{aligned}
&\Gamma_{ij}(s, dy; \tilde{G}, \tilde{\gamma}) \\
&\quad \propto \int_{x \in \mathbb{R}^d} \mathbf{P}_{s-1,x,i}[s < \tau,\, X^{(s)} \in dy,\, J^{(s)} = j] \\
&\quad \times \Pi_i(s-1, dx; \tilde{G}, \tilde{\gamma}).
\end{aligned} \tag{3.30}$$

Because $\Pi_j(G, \gamma) = \pi_j$, expressions (3.19) and (3.24) imply that $\mathcal{G}(G, \gamma) = G$ and $\Gamma_{ij}(G, \gamma) = \gamma_{ij}$, establishing our fixed-point relation (3.26).

Having fully specified the fixed-point problem, we can now consider iterative methods for its solution. One approach would be to fix some $\varepsilon \in (0, 1]$ and compute the deterministic fixed-point iteration

$$\begin{aligned}
\tilde{G}(m+1) &= \tilde{G}(m) + \varepsilon \left(\mathcal{G}(\tilde{G}(m), \tilde{\gamma}(m)) - \tilde{G}(m) \right), \text{ and} \\
\tilde{\gamma}(m+1) &= \tilde{\gamma}(m) + \varepsilon \left(\Gamma(\tilde{G}(m), \tilde{\gamma}(m)) - \tilde{\gamma}(m) \right),
\end{aligned} \tag{3.31}$$

given initial guesses $\tilde{G}(0)$ and $\tilde{\gamma}(0)$ for G and γ, respectively. One would choose $\varepsilon = 1$ in this deterministic iteration; we consider arbitrary $\varepsilon \in (0, 1]$ to motivate the stochastic approximation algorithm developed in 3.3.3.

In practice, computing $\mathcal{G}$ and Γ in the right hand side of (3.31) requires computing averages with respect to $\Pi_j(\tilde{G}(m), \tilde{\gamma}(m))$. While we cannot hope to compute these integrals exactly, we can construct a stochastic algorithm that samples drawn (asymptotically) from $\Pi_j(\tilde{G}(m), \tilde{\gamma}(m))$ and can be used to approximate the iteration in (3.31). The resulting scheme, which we detail in 3.3.3, fits within the basic stochastic approximation framework.

3.3.3 A Stochastic Approximation

In this section, we present the full NEUS algorithm and we interpret it as a stochastic approximation algorithm analogous to the deterministic fixed-point iteration (3.31). In NEUS,

as in the fixed-point iteration, we generate a sequence of approximations $\tilde{G}(m)$ and $\tilde{\gamma}(m)$, converging to G and γ, respectively. During the mth iteration of the NEUS algorithm, we update the current approximations $\tilde{G}(m)$ and $\tilde{\gamma}(m)$ based on statistics gathered from trajectories of a Markov process $Y_j^{(r)}(\tilde{G}(m), \tilde{\gamma}(m)) = \left(T_j^{(r)}, X_j^{(r)}\right)$ whose law depends on $\tilde{G}(m)$ and $\tilde{\gamma}(m)$. Recall that τ is the first time at which the process $(t, X^{(t)})$ is not contained in the set $D \in \mathbb{N} \times \mathbb{R}^d$. We define a single step in the evolution of $Y_j^{(r)}(\tilde{G}(m), \tilde{\gamma}(m))$ as follows:

1. Evolve the process $\left(X^{(t)}, J^{(t)}\right)$ one step from time $T_j^{(r)}$ and initial position $(X_j^{(r)}, j)$ to generate $\left(\tilde{X}_j^{(r+1)}, \tilde{J}_j^{(r+1)}\right)$ and time $\tilde{T}_j^{(r+1)} = T_j^{(r)} + 1$. Set $\tilde{Y}_j^{(r+1)} = (\tilde{T}_j^{(r+1)}, \tilde{X}_j^{(r+1)})$.

2. If $\tilde{J}_j^{(r+1)} = j$ and $\tilde{Y}_j^{(r+1)} \in D$, set $Y_j^{(r+1)}(\tilde{G}(m), \tilde{\gamma}(m)) = \tilde{Y}_j^{(r+1)}$.

3. Otherwise, if $\tilde{J}_j^{(r+1)} \neq j$ or $\tilde{Y}_j^{(r+1)} \notin D$, draw $Y_j^{(r+1)}(\tilde{G}(m), \tilde{\gamma}(m))$ from $\bar{\Pi}_j(s, dy; \tilde{G}(m), \tilde{\gamma}(m))$.

The process $Y_j^{(r)}(\tilde{G}, \tilde{\gamma})$ preserves the distribution $\Pi_j(t, dx; \tilde{G}, \tilde{\gamma})$ in (3.28); see B for proof. It can also be shown that, subject to mild restrictions, the average of a function over a trajectory of $Y_j^{(r)}(\tilde{G}, \tilde{\gamma})$ converges to the average of that function over (3.28) as the length of the trajectory increases. The implementation details for drawing $Y_j^{(r+1)}(\tilde{G}(m), \tilde{\gamma}(m))$ from $\bar{\Pi}_j(s, dy; \tilde{G}(m), \tilde{\gamma}(m))$ will be discussed in more detail in 3.5.

We now state the NEUS algorithm. To simplify the expressions below, we sometimes omit the iteration number m. The algorithm proceeds as follows:

1. Choose initial approximations $\tilde{G}(0)$ and $\tilde{\gamma}(0)$ of G and γ, respectively. Fix the number of steps, K, of the processes $Y_j^{(r)}(\tilde{G}(m), \tilde{\gamma}(m))$, and the maximum number of new

points, L, included in the update to the empirical approximations of the distributions $\tilde{\gamma}_{ij}(m)$.

2. If $m > 0$, for each $j = 1, 2, \ldots, n$ evolve the process $Y_j^{(r)}(\tilde{G}(m), \tilde{\gamma}(m))$ for K time steps, starting from initial condition

$$Y_j^{(0)}(m) = Y_j^{(K)}(m-1).$$

Label the output $Y_j^{(r)}(m)$ for $r = 1, 2, \ldots, K$. Similarly, record the steps $\tilde{Y}_j^{(r)}$ and $\tilde{J}_j^{(r)}$ proposed during the first stage of the algorithm for simulating $Y_j^{(r)}(\tilde{G}(m), \tilde{\gamma}(m))$, and label them $\tilde{Y}_j^{(r)}(m)$ and $\tilde{J}_j^{(r)}(m)$, respectively. For $m = 0$, $Y_j^{(0)}(0)$ is drawn from $\bar{\Pi}_j(s, dy; \tilde{G}(0), \tilde{\gamma}(0))$.

3. Let

$$M_{ij}(m) = \sum_{r=1}^{K} \mathbf{1}_{\{j\}}(\tilde{J}_i^{(r)})\, \mathbf{1}_D(\tilde{Y}_i^{(r)})$$

be the number of i to j transitions of the index process observed while generating $Y_j^{(r)}(m)$ and let $\left\{\tilde{X}_{ij}^{(\ell)}\right\}_{\ell=1}^{M_{ij}(m)}$ and $\left\{\tilde{T}_{ij}^{(\ell)}\right\}_{\ell=1}^{M_{ij}(m)}$ be the positions $\tilde{X}_i^{(r)}$ and times $\tilde{T}_i^{(r)}$ for which $\tilde{J}_i^{(r)} = j$ and $\tilde{Y}_i^{(r)} \in D$.

4. Compute

$$\hat{G}_{ij}(m) = \frac{M_{ij}(m)}{K}, \tag{3.32}$$

$$\hat{\gamma}_{ij}(s, dy; m) = \begin{cases} \frac{1}{L \wedge M_{ij}(m)} \sum_{\ell=1}^{L \wedge M_{ij}(m)} \mathbf{1}_{\tilde{T}_{ij}^{(\ell)}}(s)\, \delta_{\tilde{X}_{ij}^{(\ell)}}(dy) & \text{if } M_{ij}(m) > 0, \\ 0 & \text{if } M_{ij}(m) = 0, \end{cases} \tag{3.33}$$

and

$$\langle \hat{f} \rangle_i(m) = \frac{1}{K} \sum_{r=0}^{K-1} f\left(Y_i^{(r)}(m)\right), \tag{3.34}$$

where $L \wedge M_{ij}(m) = \min\{L, M_{ij}(m)\}$. In Equation (3.33), δ_x represents the Dirac delta function centered at position x.

5. Replace the deterministic iteration (3.31) by the approximation

$$\tilde{G}_{ij}(m+1) = \tilde{G}_{ij}(m) + \varepsilon_m \left(\hat{G}_{ij}(m) - \tilde{G}_{ij}(m)\right) \tag{3.35}$$

and

$$\tilde{\gamma}_{ij}(m+1) = \tilde{\gamma}_{ij}(m) + \varepsilon_m \left(\hat{\gamma}_{ij}(m) - \tilde{\gamma}_{ij}(m)\right) \left(\frac{\mathbf{1}_{\{M_{ij}(m)>0\}}}{I_{ij}(m)}\right) \tag{3.36}$$

where

$$I_{ij}(m) = \frac{1}{m+1} \sum_{\ell=0}^{m} \mathbf{1}_{\{M_{ij}(\ell)>0\}}$$

and $\varepsilon_m > 0$ satisfies

$$\sum_{m=1}^{\infty} \varepsilon_m = \infty \quad \text{and} \quad \sum_{m=1}^{\infty} \varepsilon_m^2 < \infty. \tag{3.37}$$

6. Update the expectations

$$\langle \tilde{f} \rangle_i(m+1) = \langle \tilde{f} \rangle_i(m) + \varepsilon_m \left(\langle \hat{f} \rangle_i(m) - \langle \tilde{f} \rangle_i(m)\right). \tag{3.38}$$

7. Once the desired level of convergence has been reached, compute

$$\mathbf{E}\left[\sum_{t=0}^{\tau-1} f(t, X^{(t)})\right] \approx \sum_{j=1}^{n} \tilde{z}_j(m)\, \langle \tilde{f} \rangle_i(m),$$

where the vector $\tilde{z}(m)$ solves $\tilde{z}^{\mathrm{T}}(m) = \tilde{z}^{\mathrm{T}}(m)\, \tilde{G}(m) + a^{\mathrm{T}}$.

We now interpret NEUS as a stochastic approximation algorithm analogous to the deterministic fixed-point iteration (3.31). First, we observe that $\hat{G}(m)$ approximates $\mathcal{G}(\tilde{G}(m), \tilde{\gamma}(m))$ in the following sense. Suppose we were to compute a sequence

$$\hat{G}(n), \hat{G}(n+1), \ldots, \hat{G}(n+k)$$

as in NEUS, except holding the values of $\tilde{G}(n)$ and $\tilde{\gamma}(n)$ fixed and computing $\hat{G}(n+i)$ from a trajectory of $Y_i^{(t)}(\tilde{G}(n), \tilde{\gamma}(n))$. We would then have

$$\lim_{k\to\infty} \frac{1}{k} \sum_{i=0}^{k-1} \hat{G}(n+i) = \mathcal{G}(\tilde{G}(n), \tilde{\gamma}(n)) \tag{3.39}$$

by (3.29) and since the left hand side is a trajectory average of $Y_i^{(t)}(\tilde{G}(m), \tilde{\gamma}(m))$. The distribution $\gamma_{ij}(m)$ approximates $\Gamma_{ij}(\tilde{G}(m), \tilde{\gamma}(m))$ in a similar sense. Therefore, the NEUS iteration (3.35) is a version of the deterministic fixed-point iteration (3.31) but with a shrinking sequence ε_m instead of a fixed ε and with random approximations instead of the exact values of $\mathcal{G}$ and Γ. The conditions (3.37) on the sequence ε_m are common to most stochastic approximation algorithms [46]; they ensure convergence of the iteration when $\mathcal{G}$ and Γ can only be approximated up to random errors.

We remark that in practice the empirical measures $\tilde{\gamma}(m)$ are stored as lists of time and position pairs. The update in (3.35) allows the number of pairs stored in these lists to grow

with each iteration. This can lead to impractical memory requirements for the method. We therefore limit the size of each list $\tilde{\gamma}_{ij}(m)$ to a fixed maximum value by implementing a selection step in which the points that have been stored for the most iterations are removed to make room for the points in the updates of $\tilde{\gamma}_{ij}(m)$ when this maximum is exceeded. Also, in our numerical experiments in 3.5, we use $\varepsilon_m = 1/(m+1)$ in which case,

$$\tilde{G}_{ij}(m) = \frac{1}{m+1} \sum_{\ell=0}^{m} \hat{G}_{ij}(\ell)$$

and

$$\tilde{\gamma}_{ij}(m) = \frac{1}{\sum_{\ell=0}^{m} \mathbf{1}_{\{M_{ij}(\ell)>0\}}} \sum_{\ell=0}^{m} \hat{\gamma}_{ij}(\ell).$$

This and other details of our implementation are explained in 3.5.

3.4 Ergodic Averages

In this section we consider the calculation of ergodic averages with respect to a general (not necessarily time-homogenous) Markov process. We also describe the simplifications that occur when the target Markov process is time-homogenous as in the original NEUS algorithm.

In order to ensure that the definitions in this section are sensible, we require that

$$\lim_{\tau\to\infty} \frac{1}{\tau} \sum_{t=0}^{\tau-1} \mathbf{P}\left[X^{(t)} \in dx,\, J^{(t)} = i\right]$$

exists as a probability distribution on $\mathbb{R}^d \times \{1, 2, \ldots, n\}$ and let

$$\pi(dx) = \lim_{\tau\to\infty} \frac{1}{\tau} \sum_{t=0}^{\tau-1} \mathbf{P}\left[X^{(t)} \in dx\right].$$

This general ergodicity requirement allows processes $X^{(t)}$ with periodicities or time depen-

dent forcing.

Our goal is to compute ergodic averages of the form

$$\lim_{\tau\to\infty}\frac{1}{\tau}\sum_{t=0}^{\tau-1}\mathbf{E}\left[f(X^{(t)})\right]=\int_{x\in\mathbb{R}^d}f(x)\pi(dx).$$

To that end, we fix a deterministic time horizon $\tau>0$ in (3.12) and (3.17). Note that the restriction $t<\tau$ inside the probability in those formulae can now be written as an upper bound of $\tau-1$ on the summation index. If we divide both sides of (3.16) by τ and take the limit $\tau\to\infty$, we obtain the equation

$$z^{\mathrm{T}}G=z^{\mathrm{T}} \tag{3.40}$$

where now

$$z_j=\lim_{\tau\to\infty}\frac{1}{\tau}\sum_{t=0}^{\tau-1}\mathbf{P}\left[J^{(t)}=j\right]. \tag{3.41}$$

and

$$G_{ij}=\lim_{\tau\to\infty}\frac{\sum_{t=0}^{\tau-2}\mathbf{P}\left[J^{(t+1)}=j,\,J^{(t)}=i\right]}{\sum_{t=0}^{\tau-1}\mathbf{P}\left[J^{(t)}=i\right]}. \tag{3.42}$$

Note that the matrix G is now stochastic and that $\sum_{j=1}^{n}z_j=1$. We can rewrite the ergodic average of f as

$$\int_{x\in\mathbb{R}^d}f(x)\pi(dx)=\sum_{j=1}^{n}z_j\,\langle f\rangle_j,$$

where

$$\langle f\rangle_j=\int_{x\in\mathbb{R}^d}f(x)\pi_j(dx) \tag{3.43}$$

and we represent the large τ limit of the position marginal distribution of π_j defined in (3.11) as

$$\pi_j(dx)=\lim_{\tau\to\infty}\sum_{t=0}^{\tau-1}\pi_j(t,dx). \tag{3.44}$$

These formulas indicate that the only modification of the algorithm in 3.3 that is required to compute a long-time average is to set $\tau = \infty$ in the definition of the processes $Y_i(\tilde{G}, \tilde{\gamma})$, to set $a = 0$ in (3.27), and let $\tilde{z}$ solve $\tilde{z}^{\mathrm{T}} = \tilde{z}^{\mathrm{T}}\tilde{G}$ with $\sum_{j=1}^{n} \tilde{z}_j = 1$. In other words, the algorithm seamlessly transitions from solving the initial value problem to solving the infinite time problem as τ becomes large.

When the joint process $(X^{(t)}, J^{(t)})$ is time-homogenous and stationary and our goal is to compute the average of a position dependent observable $f(x)$ with respect to the stationary distribution π of $X^{(t)}$, the above relations can be further simplified. In this case,

$$\pi_j(dx) = \frac{1}{z_j} \lim_{t\to\infty} \mathbf{P}\left[X^{(t)} \in dx,\, J^{(t)} = j\right],$$

where z_j defined in (3.41) becomes

$$z_j = \lim_{t\to\infty} \mathbf{P}\left[J^{(t)} = j\right].$$

The matrix G in (3.42) can now be written

$$G_{ij} = \lim_{t\to\infty} \mathbf{P}\left[J^{(t+1)} = j \mid J^{(t)} = i\right]$$

and the vector $\langle f \rangle_j$ defined in (3.43) becomes

$$\langle f \rangle_j = \int_{x\in\mathbb{R}^d} f(x)\pi_j(dx).$$

These simplifications lead to a version of the original NEUS method [89] that employs a direct method for solving for the weights similar to the scheme in [88].

In [88] and [16] the basic NEUS approach was extended to the estimation of transition rates between sets for a stationary Markov process. Implicit in this extension was the observation that any algorithm that can efficiently compute averages with respect to the stationary

distribution of a time-homogenous Markov process can be applied to computing dynamic averages more generally by an enlargement of the state space, i.e., by applying the scheme to computing stationary averages for a higher dimensional time-homogenous Markov process. This idea is also central to Exact Milestoning [5], which extends the original Milestoning procedure [23] to compute steady-state averages with respect to a time-homogenous Markov process and is very similar in structure to steady-state versions of NEUS. For an alternative implementation of NEUS inspired by Exact Milestoning but presented in the more general context of 3.2.2 and 3.3, see C.

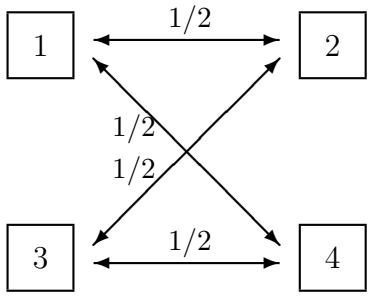
3.5 Numerical Examples

Here we illustrate the flexibility of the generalized algorithm with respect to both the means of restricting the trajectories (the choice of the $J^{(t)}$ process) and the averages that can be calculated. Specifically, in 3.5.1 we present a minimal and analytically tractable model to illustrate the terminology of the framework. In 3.5.2 we discuss an alternative choice of the $J^{(t)}$ process. In 3.5.3 we show how finite-time hitting probabilities can be calculated by discretizing the state space according to both time and space. In 3.5.4 we show how free energies can be obtained by discretizing the state space according to time and the irreversible work.

3.5.1 Illustrative Markov Model

In this supplementary document we introduce a minimal and analytically tractable Markov model where the terminology and notation of the trajectory stratification framework can be expressed to clearly illustrate the method. This model does not address the utility of trajectory stratification in practice.

The model is a discrete Markov process $X^{(t)} \in \{1, 2, 3, 4\}$ with the structure



The chain has the transition matrix,

$$T = \begin{bmatrix} 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \end{bmatrix} \tag{3.45}$$

and initial condition

$$\mathbf{P}[X^{(0)} = y] = \begin{cases} \frac{1}{2} & \text{if } y = 1 \\ \\ \frac{1}{2} & \text{if } y = 3 \end{cases}. \tag{3.46}$$

We consider the case where $\tau = 2$. In general, the aim is to compute expectations of the form

$$\mathbf{E}\left[\sum_{t=0}^{\tau-1} f(t, X^{(t)})\right] = \sum_{i=1}^{n} z_i \langle f \rangle_i \tag{3.47}$$

over a domain, D, of time-space pairs. For this model,

$$D = (t \in \{0, 1\} \times x \in \{1, 2, 3, 4\}).$$

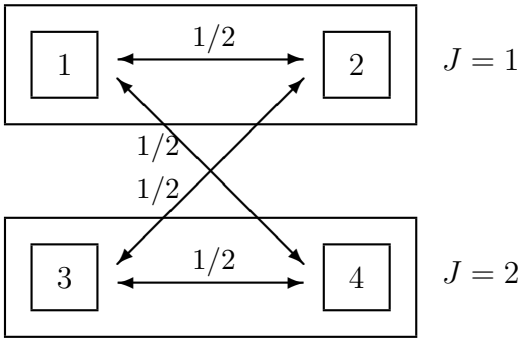
For now, there is no need to choose a particular form of $f(t, x)$, however the choice of this function is determined from the context of a particular application. In this example, we outline how an expectation of the form in (3.47) can be computed by stratification.

We first define the index process $J^{(t)}$ over which we stratify the process $X^{(t)}$. For this

model we define the index process

$$J^{(t)} = \begin{cases} 1 & \text{if } X^{(t)} \in \{1, 2\} \\ 2 & \text{if } X^{(t)} \in \{3, 4\}, \end{cases} \tag{3.48}$$

where we have grouped the states as:



For $\tau = 2$, the process begins at $X^{(0)} = 1$ or $X^{(0)} = 3$ each with probability one-half and transitions to site 2 or 4 each with probability one-half at time $t = 1$. At time $t = 2$, the process leaves D.

To compute (3.47) by stratification, we define the restricted distributions, $\pi_i(t, x)$, against which the terms in the left hand side of (3.47) are computed. The restricted distributions are

$$\pi_i(t, x) = \frac{1}{z_i} \mathbf{P}\left[t < 2, X^{(t)} = x, J^{(t)} = i\right] \tag{3.49}$$

with normalization constants

$$\begin{aligned} z_1 &= \sum_{t=0}^{\infty} \mathbf{P}[t < 2, J^{(t)} = 1] \\ &= \mathbf{P}[J^{(0)} = 1] + \mathbf{P}[J^{(1)} = 1] \\ &= \frac{1}{2} + \frac{1}{2} = 1 \end{aligned}$$

and similarly, $z_2 = 1$. We then write the expectation in (3.47) as

$$\mathbf{E}\left[\sum_{t=0}^{\tau-1} f(t, X^{(t)})\right] = \langle f \rangle_1 + \langle f \rangle_2 \tag{3.50}$$

where

$$\langle f \rangle_i = \sum_{t=0}^{\infty} \sum_{x=1}^{4} f(t,x)\pi_i(t,x). \tag{3.51}$$

The z_i can be expressed as expectations over the π_i. Recall that,

$$G_{ij} = \frac{\sum_{t=0}^{\infty} \mathbf{P}\left[t+1 < \tau, J^{(t+1)} = j, J^{(t)} = i\right]}{z_i}. \tag{3.52}$$

From the definition of G_{ij}, the exact transition matrix for $\tau = 2$ is

$$G = \begin{bmatrix} 1/4 & 1/4 \\ 1/4 & 1/4 \end{bmatrix} \tag{3.53}$$

and the initial conditions are

$$a = \begin{bmatrix} \mathbf{P}[J^{(0)} = 1] \\ \mathbf{P}[J^{(0)} = 2] \end{bmatrix} = \begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix} \tag{3.54}$$

by definition. The related affine eigenequation, $z^{\mathrm{T}} = z^{\mathrm{T}} G + a^{\mathrm{T}}$, can be solved to verify that $z_1 = z_2 = 1$.

So far, we have outlined how (3.47) can be computed from stratification in this four-site model by computing expectations against the two restricted distributions, $\pi_1(t,x)$ and $\pi_2(t,x)$. For this simple model, these terms can be evaluated exactly and no sampling is required to compute (3.47). In general, these terms need to be computed by solving the self-consistent iteration at the heart of the NEUS algorithm. In this section we interpret the self-consistent procedure in the context of the four-site Markov model.

The flux distributions, $\bar{\pi}_j(s,y)$, can also be derived exactly for the four site Markov model. Recall that the flux distributions are defined as

$$\bar{\pi}_j(s,y) = \frac{1}{z_j(1-G_{jj})} \begin{cases} \sum_{i \neq j} z_i G_{ij} \gamma_{ij}(s,y) & \text{if } s > 0 \\ a_j \mathbf{P}[X^{(0)} \in y \mid J^{(0)} = j] & \text{if } s = 0 \end{cases} \tag{3.55}$$

where γ_{ij} are the conditional flux distributions for transitions from $J = i$ to $J = j$. In general, the conditional flux distributions are

$$\gamma_{ij}(s,y) \propto \sum_{x \in \mathbb{R}^d} \mathbf{P}_{s-1,x,i}[s < \tau, X^{(s)} = y, J^{(s)} = j] \pi_i(s-1,x) \tag{3.56}$$

or, for this simple model,

$$\gamma_{21}(s,y) = \begin{cases} 1 & \text{if } (s,y) = (1,2) \\ 0 & \text{otherwise} \end{cases} \tag{3.57}$$

and

$$\gamma_{12}(s,y) = \begin{cases} 1 & \text{if } (s,y) = (1,4) \\ 0 & \text{otherwise.} \end{cases} \tag{3.58}$$

Therefore, noting that

$$\frac{z_2 G_{21}}{z_1(1-G_{11})} = \frac{1}{3} \tag{3.59}$$

and

$$\frac{a_1}{z_1(1-G_{11})} = \frac{2}{3}, \tag{3.60}$$

the exact flux distributions are

$$\bar{\pi}_1(t,x) = \begin{cases} \frac{1}{3} & \text{if } (t,x) = (1,2) \\ \frac{2}{3} & \text{if } (t,x) = (0,1) \\ 0 & \text{otherwise} \end{cases} \tag{3.61}$$

and

$$\bar{\pi}_2(t,x) = \begin{cases} \frac{1}{3} & \text{if } (t,x) = (1,4) \\ \frac{2}{3} & \text{if } (t,x) = (0,3) \\ 0 & \text{otherwise.} \end{cases} \tag{3.62}$$

As discussed in Section 3, we interpret NEUS as a stochastic approximation algorithm to solve the deterministic fixed point equation

$$(\mathcal{G}(G,\gamma), \Gamma(G,\gamma)) = (G,\gamma), \tag{3.63}$$

where the matrix G and conditional flux distributions γ are the fixed points of a pair of maps, $\mathcal{G}$ and Γ, that take $\tilde{G}$ and $\tilde{\gamma}$ as arguments and return a new approximation to G and γ respectively. Here, however, we interpret the deterministic fixed point iteration

$$(\mathcal{G}(\tilde{G}(m),\tilde{\gamma}(m)), \Gamma(\tilde{G}(m),\tilde{\gamma}(m))) = (\tilde{G}(m+1),\tilde{\gamma}(m+1)),$$

for the four-site model and show that, for this model, this iteration can be expressed in a single variable representing the relative contribution of the fluxes $\tilde{z}_i(m)\tilde{G}_{ij}(m)$ where $\tilde{z}(m)$ solves $\tilde{z}^{\mathrm{T}}(m) = \tilde{z}^{\mathrm{T}}(m)\tilde{G}(m) + a^{\mathrm{T}}$.

We define the Markov process $Y_j^{(r)}(\tilde{G}(m),\tilde{\gamma}(m))$ that samples each approximate restricted distribution $\Pi_j(t,x;\tilde{G}(m),\tilde{\gamma}(m))$. The process $Y_j^{(r)}(\tilde{G}(m),\tilde{\gamma}(m))$ evolves with the underlying process $(X^{(t)}, J^{(t)})$ forward in time until the process transitions from $J = j$ to

$J = i$ or hits $t = \tau$, at which point $Y_j^{(r+1)}$ is drawn from $\bar{\Pi}_j(t, x; \tilde{G}(m), \tilde{\gamma}(m))$. For clarity, we use the notation $Y_j^{(r)}(m)$ to denote the chain $Y_j^{(r)}(\tilde{G}(m), \tilde{\gamma}(m))$.

For the four-site model, the conditional flux distributions are determined exactly by sampling $Y_j^{(r)}(m)$, i.e., $\tilde{\gamma}_{21}(1,2) = \gamma_{21}(1,2) = 1$ and $\tilde{\gamma}_{12}(1,4) = \gamma_{21}(1,4) = 1$. Therefore, the function $\bar{\Pi}_j(t, x; \tilde{G}(m), \tilde{\gamma}(m))$ depends only on determining the relative fluxes, $\tilde{z}_i(m)\tilde{G}_{ij}(m)$. Specifically let

$$p(m) = \frac{a_1}{\tilde{z}_1(m)(1 - \tilde{G}_{11}(m))} \tag{3.64}$$

be the relative weight of the $s = 0$ time contribution to $\bar{\Pi}_1(t, x; \tilde{G}(m), \tilde{\gamma}(m))$ at the mth iteration. From (3.60) we see that $p(m)$ should converge to $2/3$ as m increases. Because the underlying dynamics are symmetric for transitions between the index process J, we derive this fixed point equation only for the single number $p(m)$ described above for $J = 1$ since the analogous ratio for $J = 2$ is equivalent at each iteration. In the following we express (3.63) as a fixed point expression in $p(m)$.

For our example, $Y_1^{(r)}(m)$ is equivalent to a Markov chain over sites 1 and 2. In order to compute the expectation of $\tilde{G}_{1j}(m)$ from $Y_1^{(r)}(m)$, we first derive the effective transition matrix, $T(m)$, for $Y_1^{(r)}(m)$. We assume here that at each iteration m we compute expectations from the process $Y_1^{(r)}(m)$ exactly.

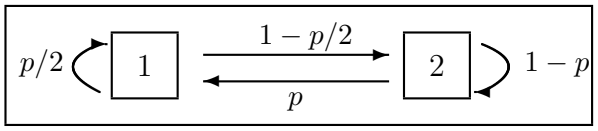
To compute $T(m)$, we enumerate the transitions of $Y_1^{(r)}(m)$. For brevity, we momentarily drop the iteration index m from our notation. The possible transition for $Y_1^{(r)}$ are as follows:

- $Y_1^{(r)}$ transitions from $1 \to 2$ by proposing $X^{(0)} = 1 \to X^{(1)} = 2$ with probability $1/2$.
- $Y_1^{(r)}$ transitions from $1 \to 2$ by proposing a transition of $X^{(0)} = 1 \to X^{(1)} = 4$ with probability $1/2$ and drawing $Y_1^{(r+1)} = 2$ from $\bar{\Pi}_j$ with probability $(1 - p)$. This transition has overall probability $(1 - p)/2$.
- $Y_1^{(r)}$ transitions from $1 \to 1$ by proposing $X^{(0)} = 1 \to X^{(1)} = 4$ with probability $1/2$ and drawing $Y_1^{(r+1)} = 1$ from $\bar{\Pi}_j$ with probability p. This transition has overall

probability $p/2$.

- $Y_1^{(r)}$ transitions from $2 \to 1$ by drawing $Y_1^{(r+1)} = 1$ from $\bar{\Pi}_j$ with probability p.
- $Y_1^{(r)}$ transitions from $2 \to 2$ by drawing $Y_1^{(r+1)} = 2$ from $\bar{\Pi}_j$ with probability $1 - p$.

Summing over the possible transition probabilities for this chain results in an effective chain with the structure



The effective transition matrix for $Y_1^{(r)}$ is

$$T(m) = \begin{bmatrix} p(m)/2 & 1 - p(m)/2 \\ p(m) & 1 - p(m) \end{bmatrix}. \tag{3.65}$$

By symmetry, the effective transition matrix for $Y_2^{(r)}$ has the same form.

The entries of the matrix $\tilde{G}_{1j}(m+1)$ are computed as expectations from the effective chain defined by $T(m)$. In the $\tau = 2$ case, only the transitions from site 1 or site 2 at time $t = 0$ contribute to $\tilde{G}_{ij}(m+1)$. For $J = 1$, the process $(J^{(t)}, X^{(t)})$ conditioned on $(J^{(0)}, X^{(0)}) = (1, 1)$ transitions to $(J^{(1)}, X^{(1)}) = (2, 4)$ with probability one-half and transitions to $(J^{(1)}, X^{(1)}) = (1, 2)$ with probability one-half. Therefore, each element of the transition matrix

$$\tilde{G}_{1j}(m+1) = \lim_{K\to\infty} \frac{1}{K} \sum_{r=0}^{K-1} \mathbf{P}[J^{(1)} = j | Y_1^{(r)} = 1] \mathbf{P}[Y_1^{(r)} = 1] \tag{3.66}$$

$$= \frac{1}{2} \lim_{K\to\infty} \frac{1}{K} \sum_{r=0}^{K-1} \mathbf{P}[Y_1^{(r)} = 1] \tag{3.67}$$

$$= \frac{1}{2} \left(\frac{p(m)}{1 + p(m)/2} \right) = \frac{p(m)}{p(m) + 2} \tag{3.68}$$

A similar argument is made for the second row of $\tilde{G}(m+1)$. Therefore the matrix

$$\tilde{G}(m+1) = \begin{bmatrix} p(m)/(p(m)+2) & p(m)/(p(m)+2) \\ p(m)/(p(m)+2) & p(m)/(p(m)+2) \end{bmatrix}. \tag{3.69}$$

The weights $\tilde{z}(m+1)$ are computed by solving $\tilde{z}(m+1)^T = \tilde{z}(m+1)^T \tilde{G}(m+1)^T + a^T$ which gives the solution $\tilde{z}_1(m+1) = \tilde{z}_2(m+1) = (p(m)+2)/(4-2p(m))$.

Finally, the fixed point relationship is expressed for this model in $p(m)$ by expressing $p(m+1)$ in (3.64) in terms of $p(m)$ which gives

$$p^{(m+1)} = \frac{2 - p^{(m)}}{2}, \tag{3.70}$$

an iteration that converges to the value 2/3.

For the four-site model, the NEUS fixed point equation can be reduced to a fixed point equation in a single variable that represents the relative weight of the contributions to $\bar{\pi}_i$. In general, the terms in the fixed point expression cannot be computed exactly and a stochastic approximation strategy like NEUS is required.

3.5.2 One Choice of the $J^{(t)}$ Process

Rapid convergence of the scheme outlined in 3.3 rests on the choice of $J^{(t)}$. Perhaps the most intuitive choice is

$$J^{(t)} = \sum_{j=1}^{n} j \mathbf{1}_{A_j}(t, X^{(t)}) \tag{3.71}$$

where the subsets $A_1, A_2, \ldots, A_n$ partition $\mathbb{N} \times \mathbb{R}^d$. Indeed, earlier steady-state NEUS implementations [89, 16, 15, 88, 14] employed an analogous rule using a partition of the space variable (the time variable was not stored or partitioned). However, even with an optimal choice of the subsets $A_1, A_2, \ldots, A_n$, (3.71) has an important disadvantage: in many situa-

tions, $X^{(t)}$ frequently recrosses the boundary between neighboring subsets A_i and A_j, which slows convergence. Fortunately, there are many alternative choices of $J^{(t)}$ that approximate the choice in (3.71) while mitigating this issue. We give one simple and intuitive alternative which we use in the numerical examples that follow.

Let ψ_j be a set of non-negative functions on $\mathbb{N} \times \mathbb{R}^d$ for which $\sum_{j=1}^n \psi_j = 1$. The ψ_j are generalizations of the functions $\mathbf{1}_{A_j}$ in that they serve to restrict trajectories to regions of state space. In practice, given a partition of space $A_1, A_2, \ldots, A_n$, the ψ_j can be chosen to be smoothed approximations of the functions $\mathbf{1}_{A_j}$. Given a trajectory of $X^{(t)}$, the rule defining $J^{(t)}$ is as follows. Initially, choose $J^{(0)} \in \{1, 2, \ldots, n\}$ with probabilities proportional to $\{\psi_1(0, X^{(0)}), \psi_2(0, X^{(0)}), \ldots, \psi_n(0, X^{(0)})\}$. At later times $J^{(t)}$ evolves according to the rule

1. If $\psi_{J^{(t-1)}}(t, X^{(t)}) > 0$ then $J^{(t)} = J^{(t-1)}$.

2. Otherwise sample $J^{(t)}$ independently from $\{1, 2, \ldots, n\}$ according to probabilities $\{\psi_1(t, X^{(t)}), \psi_2(t, X^{(t)}), \ldots, \psi_n(t, X^{(t)})\}$.

While transitions out of $J^{(t)} = i$ occur when $X^{(t)}$ leaves the support of ψ_i, transitions back into $J^{(t)} = i$ can only occur outside of the support of ψ_j. Thus, this transition rule allows one to separate in space the values of $X^{(t)}$ at which $J^{(t)}$ transitions away from i from those where $J^{(t)}$ transitions into i, mitigating the recrossing issues mentioned above.

In our examples, we discretize time and only one additional "collective variable" (a dihedral angle in 3.5.3 and the nonequilibrium work in 3.5.4). Here we denote the collective variable by ϕ, and we discretize it within some interval of values $[a, b]$ (though it may take values outside this interval). In both examples $[a, b]$ is evenly discretized into a set of points $\{a + k(b-a)/m_\phi\}_{k=0}^{m_\phi}$ for some integer m_ϕ. Letting ϕ_j be any of the points in that dis-

cretization, we set

$$\psi_j(t,x) \propto \begin{cases} \left[1 - \frac{1}{\Delta_\phi}|\phi(x)\phi_j|\right] \mathbf{1}_{[a,b]} & \text{if } |\phi(x) - \phi_j| \leq \Delta_\phi \text{ and } t \in [t^j_{start}, t^j_{end}) \\ 0 & \text{otherwise} \end{cases} \tag{3.72}$$

where Δ_ϕ is some fixed value controlling the width of the support of ψ_j, and the indicator $\mathbf{1}_{[a,b]}$ restricts the terminal functions. Recall that the ψ_j are required to sum to 1. We choose t^j_{start} and t^j_{end} to equally divide the interval $[0,\tau)$, where, in our examples, τ is a fixed time horizon. The function ψ_j is largest when $t \in [t^j_{start}, t^j_{end})$ and $\phi(x) = \phi_j$. The supports of the various ψ_j correspond to products of overlapping intervals in the ϕ variable, but non-overlapping intervals in time. The fact that ψ_j depends on time is essential in our examples.

3.5.3 Finite-Time Hitting Probability

In this section we compute the probability, $P_{BA}(\tau_{\max})$, of hitting a set B before a separate set A and before a fixed time $\tau_{\max} > 0$ given that the system is at a point $X^{(0)} \notin A \cup B$ at time $t = 0$. In the case where $X^{(0)}$ and B are separated by a large free energy barrier while $X^{(0)}$ and A are not, computing $P_{BA}(\tau_{\max})$ can be challenging since trajectories that contribute to $P_{BA}(\tau_{\max})$ are rare in direct simulations. To compute $P_{BA}(\tau_{\max})$ via the scheme in 3.3.3, we let the stopping time τ be the minimum of $\tau_{\max}$ and the first time, t, at which $X^{(t-1)}$ is in either A or B, i.e., $\tau - 1 = \min\{\tau_A, \tau_B, \tau_{\max} - 1\}$ where τ_A and τ_B are the first times that $X^{(t)}$ enters the sets A and B respectively. Strictly speaking, to write τ in the form in (3.9), we need to replace $(t, X^{(t)})$ in that equation by $(t, X^{(t-1)}, X^{(t)})$. The set D corresponding to our choice of τ is then $D = \{(t,x,y) : t < \tau_{\max}, x \notin (A \cup B)\}$. As we have already mentioned, this can be done without further modification of the scheme. Then $f(t, X^{(t)})$ in (3.10) is

$$f(t, X^{(t)}) = \mathbf{1}_B(X^{(t)}). \tag{3.73}$$

The system that we simulate is the alanine dipeptide (CH_3-CONH-$C^{\alpha}H(C^{\beta}H_3)$-CONH-$CH_3$) in vacuum modeled by the CHARMM 22 force field [54]. We use the default Langevin integrator [77] implemented in LAMMPS [72], with a temperature of 310 K, a timestep of 1 fs and a damping coefficient of $30\,\mathrm{ps}^{-1}$. The SHAKE algorithm is used to constrain all bonds to hydrogens [73]. We consider the system to be in set A if $-150^\circ < \phi < -100^\circ$ and in set B if $30^\circ < \phi < 100^\circ$ (3.2). We discretize time into intervals of $t_{end} - t_{start} = 10^3$ time steps with a terminal time of $\tau_{\max} = 10^4$ time steps. We use the rule outlined in 3.5.2 for the evolution of $J^{(t)}$ with the ψ_j of the form in (3.72). The ϕ_j in (3.72) are chosen from the set $\{-100^\circ, -74^\circ, -48^\circ, -22^\circ, 4^\circ, 30^\circ\}$ with $[a, b] = [-100^\circ, 30^\circ]$ and $\Delta_\phi = 20^\circ$.

We generate the initial point $X^{(0)}$ by running an unbiased simulation at 310 K and choosing a single point $X^{(0)}$ between the sets A and B. The vector a defined in (3.18) is

$$a_j = \frac{\psi_j(0, X^{(0)})}{\sum_{i=1}^{n} \psi_i(0, X^{(0)})}. \tag{3.74}$$

Note that the initial condition at $J^{(0)}$ can be drawn from an ensemble of configurations with minimal changes to the algorithm, but we restrict our attention to the initial condition consisting of a single point. To evaluate the performance of the algorithm in 3.3.3, we choose two points from our direct simulation, one at $\phi = -58.0^\circ$ and one at $\phi = -91.0^\circ$. The former is chosen to allow the NEUS results to be compared with results from unbiased direct simulations, while the latter provides a more challenging test because P_{BA} becomes small when $X^{(0)}$ is close to A.

We set $K = 1000$ and $L = 1$ and perform a total of 10^4 iterations (about 5.9 μs of dynamics) of the scheme in 3.3.3 for each starting point. Each step of the process $Y_j^{(r)}(\tilde{G}(m), \tilde{\gamma}(m))$ corresponds to 10 time steps of the physical model. The $\tilde{\gamma}_{ij}$ are represented as lists of time and position pairs with associated weights. We cap the maximum size of those lists at 250 entries. If $\tilde{\gamma}_{ij}$ reaches this maximum size, each new entry overwrites the oldest previous entry. Because the lists are empty at the start of the calculation, we restrict sampling in Step 2 of the

algorithm in 3.3.3 to regions with at least one stored entry point ("progressive initialization" in [14]). When required, a sample (S, Y) is drawn from $\bar{\Pi}_j(s, dy; \tilde{G}(m), \tilde{\gamma}(m))$ by the following. With probability $a_j/z_j(1 - G_{jj})$, set $S = 0$ and select Y from $\mathbf{P}[X^{(0)} \in dy | J^{(0)} = j]$, or with the remaining probability select an index I proportional to the flux $z_i\, G_{ij}$ and then select (S, Y) from the list of weighted samples comprising $\tilde{\gamma}_{Ij}(m)$. For each j we compute $\langle f \rangle_j = P^j_{BA} = M_{jB}/(mK)$ where M_{jB} is the total number of transition events of $X^{(r)}_j$ into B observed after m iterations (mK is the total computational effort in state j after m iterations). The estimate of $P_{BA}(\tau_{\max})$ after m iterations is then computed as $P_{BA}(\tau_{\max}) = \sum_{j=1}^{n} P^j_{BA}\, \tilde{z}_j(m)$.

To assess the efficiency of the trajectory stratification, we also estimate $P_{BA}(\tau_{\max})$ by integrating an ensemble of $n = 10^6$ unbiased dynamics trajectories for $\tau_{\max}$ time steps from the initial point $X^{(0)}$. In this case, $P_{BA}(\tau_{\max}) \approx N_B/N$, where N_B is the number of trajectories that hit set B before set A. To assess the accuracy of the NEUS result, we perform 10 independent NEUS calculations. In each NEUS simulation, we estimate the value of P_{BA} as the average over the final 1000 iterations of each simulation and compute the mean of this estimate over 10 independent NEUS simulations. We obtain $P_{BA}(\tau_{\max}) \approx 4.33 \times 10^{-4}$ from NEUS and $P_{BA}(\tau_{\max}) \approx 4.12 \times 10^{-4}$ from direct simulation for the starting point at $\phi = -58.0°$ (3.3). In this case, the NEUS result is within the 95% confidence interval $[3.72 \times 10^{-4}, 4.52 \times 10^{-4}]$ (estimated as $\pm 1.96\sqrt{p(1-p)/n}$, where p is the estimate of P_{BA} from the direct simulation) for the direct simulation estimate given the number of samples. We obtain $P_{BA}(\tau_{\max}) \approx 2.94 \times 10^{-8}$ from NEUS for the starting point at $\phi = -91.0°$, consistent with the fact that none of the unbiased trajectories reached B before A in this case. From the same data (for either NEUS or direct simulation), one can easily assemble estimates of $P_{BA}(t)$ for any $t \leq \tau_{\max}$ by counting only those transitions into B that occur before t time steps. Up to a normalization, $P_{BA}(t)$ is the cumulative distribution function for the time that it takes $X^{(t)}$ to enter B conditioned on not entering A. Estimates of this

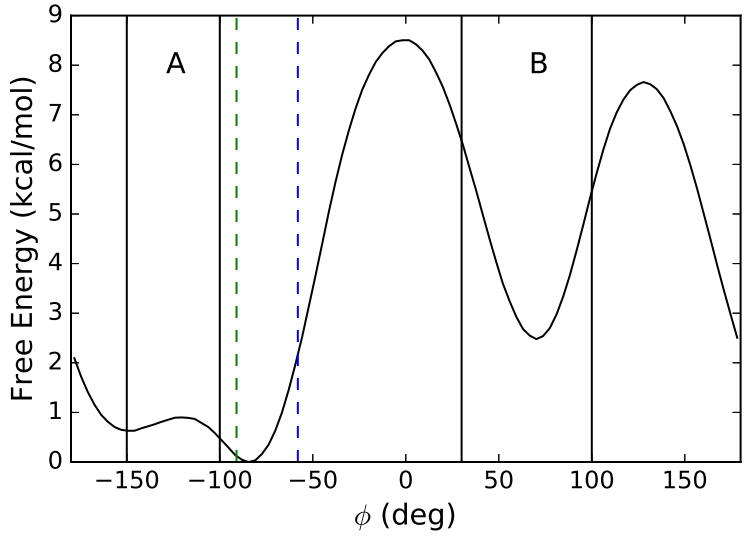


Figure 3.2: Free energy (black curve) of the alanine dipeptide projected onto the ϕ dihedral angle, with sets A and B indicated. The initial positions of $X^{(0)}$ at $\phi = -58.0°$ (blue) and $\phi = -91.0°$ (green) are shown as vertical dashed lines. The free energy is computed from the method presented in 3.2.1 as implemented in [82]

.

cumulative distribution function compiled from the NEUS and direct simulation data are plotted in 3.4. The NEUS results show excellent agreement with the results from the direct simulation.

Spatiotemporal plots of the weights computed from the converged NEUS calculations and the direct simulations are shown in 3.5. For both starting points, the stratification scheme is able to efficiently sample events with weights spanning 12 orders of magnitude. When $X^{(0)}$ is close to the boundary of set A, accurate estimation of the very small probability $P_{BA}(\tau_{\max})$ depends sensitively on the ability to realize a set of very rare trajectories, ruling out the use of direct simulation.

3.5.4 Free Energy Differences via the Jarzynski Equation

In this section, we show how a specific choice of the $J^{(t)}$ process enables us to stratify a path-dependent variable, specifically, the accumulated work appearing in the Jarzynski equation

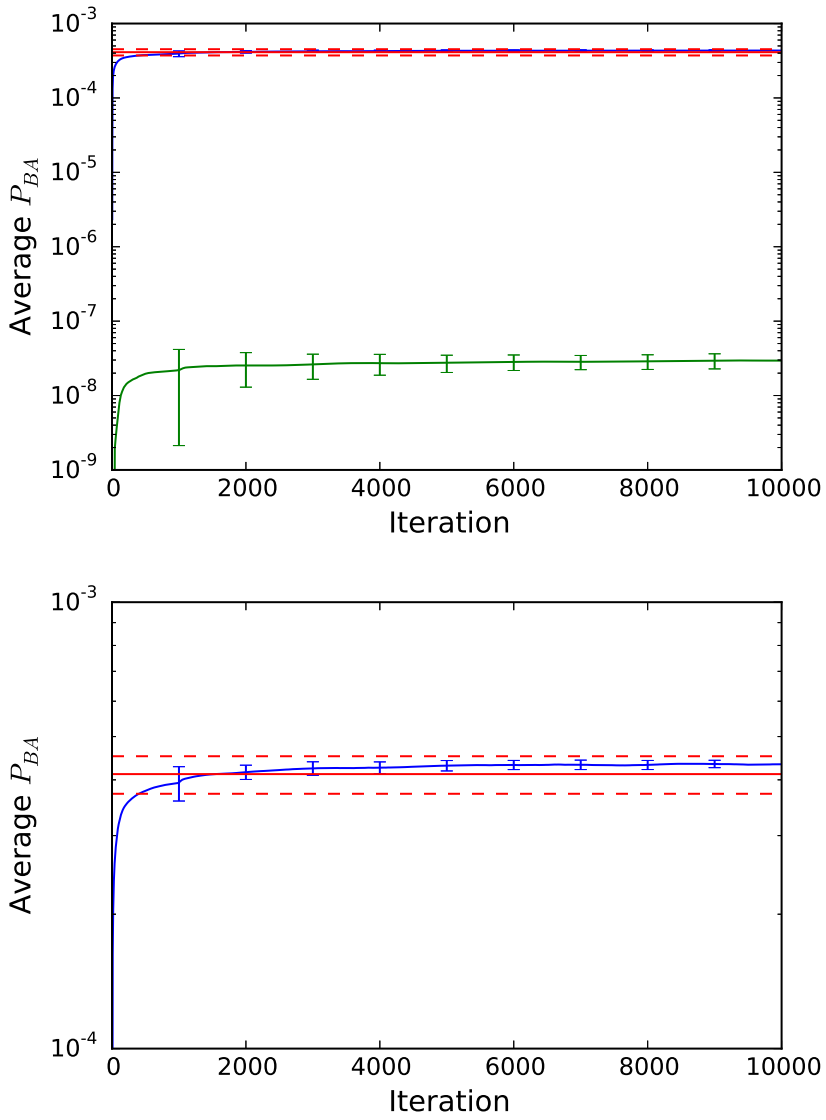


Figure 3.3: Running estimate of P_{BA} from NEUS for dynamics starting at $\phi = -58.0°$ (blue, upper curve; error bars are computed every 1000 iterations and indicate $\pm 2.262 s/\sqrt{n}$ where s is the standard error estimated from $n = 10$ independent NEUS simulations) compared to the final result from direct simulation (red solid line; dashed lines indicate $\pm 1.96\sqrt{p(1-p)/n}$, where $n = 10^6$ is the number of physically weighted trajectories generated and p is the estimate of P_{BA} from the direct simulation). Also shown is the estimate from NEUS for dynamics starting from $\phi = -91.0°$ (green, lower curve; error bars computed similarly as the blue curve). The estimate at each iteration is computed as the average of the previous 1000 iterations. Lower panel is a magnification of the upper panel.

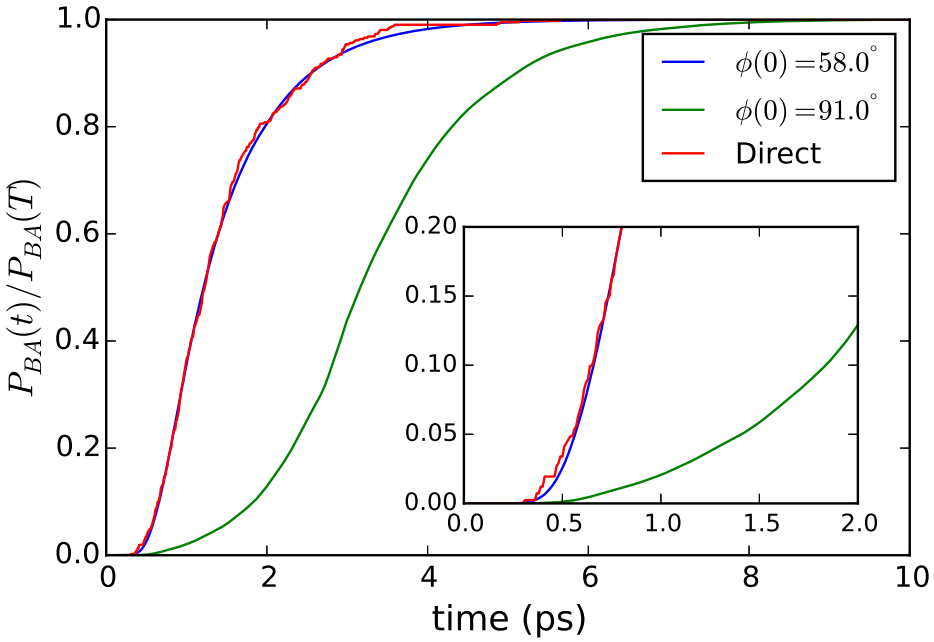


Figure 3.4: Estimate of the cumulative distribution function of the time to enter set B conditioned on not entering A from NEUS for the dynamics starting at $\phi = -58.0°$ (blue) and $\phi = -91.0°$ (green) compared to the result from the direct simulation (red). (Inset) The early time portion is shown. The estimate from each NEUS simulation at each time is computed as an average over the last 1000 iterations of the calculation and then averaged over 10 independent NEUS simulaitons.

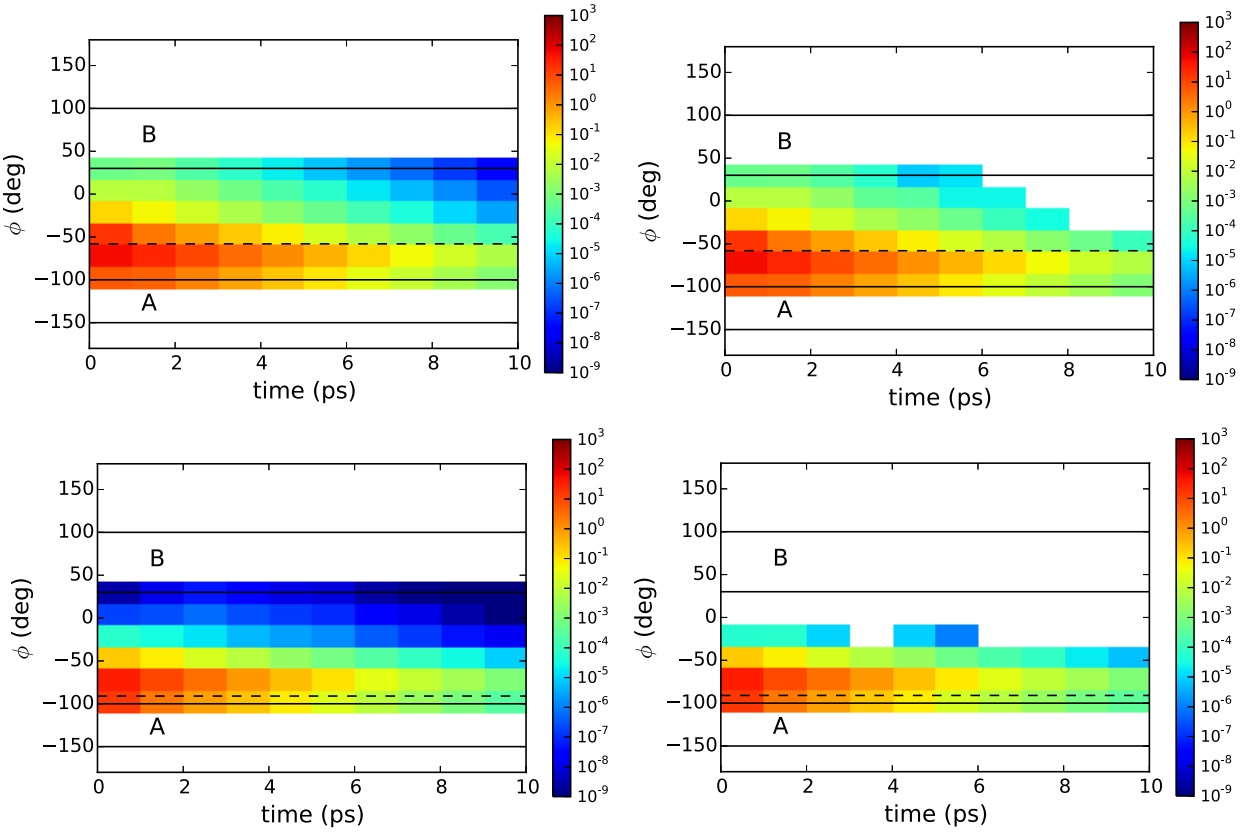


Figure 3.5: Estimates of the subset weights from NEUS (left) and direct simulations (right). Upper panels show the dynamics starting from $\phi = -58.0°$ (dashed line) and lower panels show the dynamics starting from $\phi = -91.0°$ (dashed line). White space represents subsets which were not sampled.

[40, 49]. For a statistical model defined by a density proportional to $\exp[-V(x)]$ (e.g., $V(x)$ is a potential function or a log-likelihood), the normalization constant is $Q = \int e^{-V(x)} dx$. In fields ranging from statistics to chemistry, a ratio of normalization constants is often used to compare models [12, 45]. Subject to certain conditions [40, 61], the Jarzynski equation relates the ratio of normalization constants to an average over paths of a time-dependent process, $X^{(t)}$:

$$\frac{Q_t}{Q_0} = \mathbf{E}\left[\exp(-W^{(t)})\right] \tag{3.75}$$

where

$$W^{(t)} = \sum_{\ell=0}^{t-1} V(\ell+1, X^{(\ell)}) - V(\ell, X^{(\ell)}), \qquad W^{(0)} = 0$$

and we refer to $\Delta F = -\log(Q_t/Q_0)$ as the free energy difference. For example, for a small time discretization parameter, dt, a suitable choice of dynamics is

$$X^{(t+1)} = X^{(t)} - \frac{\partial V(t+1, X^{(t)})}{\partial x}\, dt + \sqrt{2\, dt}\, \xi_t \tag{3.76}$$

where ξ_t is a standard Gaussian random variable and $X^{(0)}$ is drawn from $p_0 \propto \exp[-V(0,x)]$.

Formula (3.75) suggests a numerical procedure for estimating free energy differences in which one simulates many trajectories of $X^{(t)}$, evaluates the work $W^{(t)}$ for each, and then uses this sample to compute the expectation on the right hand side of (3.75) approximately. This approach has been particularly useful in the context of single-molecule laboratory experiments [37, 38]. A well-known weakness of this strategy in the fast-switching (small t) regime is large statistical errors result from the fact that low-work trajectories contribute significantly to the expectation but are infrequently sampled [37, 92, 64, 41, 85].

The quantity that we seek to compute is the free energy difference between a particle in a double-well potential that is additionally harmonically restrained with spring constant $k = 20$ near $x = -1$ and a particle in the same potential restrained near $x = 1$. The model

is adapted from the one presented in [12]. Setting $\tau = 1001$, for $t < \tau$ we define

$$V(t,x) = 5\left(x^2-1\right)^2 + 3x + k\left(x - (2t\,dt - 1)\right)^2 \tag{3.77}$$

where $dt = 0.001$. We show $V(0,x)$, $V(\tau-1,x)$, and $V(x;k=0)$ in 3.6. The process $X^{(t)}$ evolves according to (3.76).

The reader may be concerned that the expectation in (3.75) is not immediately of the general form in (3.10) suitable for an application of NEUS. We will apply NEUS as described in 3.2.2 to the augmented process $Z^{(t)} = (X^{(t)}, W^{(t)})$. To compute the expectation of the left hand side of (3.75) via NEUS, we compute the expectation in (3.10) with

$$f(t, Z^{(t)}) = \begin{cases} \exp(-W^{(t)}) & \text{if } t = \tau - 1 \\ 0 & \text{if } t \neq \tau - 1. \end{cases} \tag{3.78}$$

The index process $J^{(t)}$ will mark transitions between regions of the time t and accumulated work $W^{(t)}$ variables. We discretize the work space in overlapping subsets using the pyramid form in (3.72). We use 86 subsets with centers evenly spaced on the interval $[-25.0, 30.0]$ with a width of $\Delta_\phi = 0.6$. We discretize time into 5 discrete nonoverlaping subsets every 200 time steps for a total of 430 subsets. We again cap the maximum size of the list representation of $\{\tilde{\gamma}_{ij}\}$ at 250 entries using the same scheme as in 3.5.3.

For both NEUS and direct simulations, we prepare an ensemble of 1000 starting states $X^{(0)}$ by performing an unbiased simulation with fixed potential $V(0,x)$ for 10^6 steps, saving every 1000 steps. The direct fast-switching simulations start from each of these points and comprise 1000 steps of integration forward in time; each trajectory contributes equally to the left hand side of (3.75). For the NEUS simulations, the vector a is constructed as in (3.74), and trajectories are initialized at $J^{(0)}$ by drawing uniformly from this ensemble. We set $K = 1000$ and $L = 1$, and we perform 250 iterations. Each step in K corresponds to a

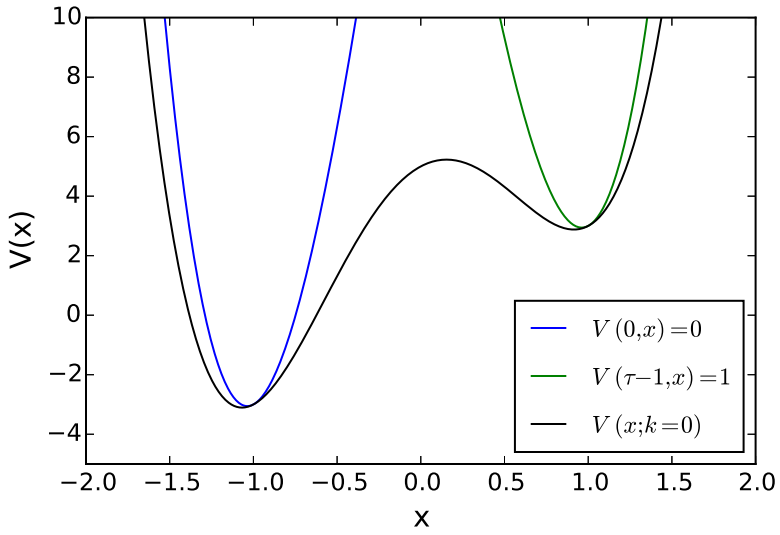


Figure 3.6: $V(0,x)$ (blue) and $V(\tau-1,x)$ (green) for the switching process used to compute Jarzynski's equality. For reference, the potential with $k=0$ (black) is also shown.

single step of (3.76). As in 3.5.3, we sample only in the restricted distributions where there is at least one point stored in $\tilde{\gamma}$ from which to restart the dynamics.

The estimated ΔF produced from data generated in the last 50 iterations of NEUS is 5.87 (the units are chosen to absorb temperature factors above), which is in excellent agreement with the reference value of 5.94, in contrast to the estimate from direct simulation (3.7). The left panel of 3.8 shows the weights along the time and work axes. In the right panel of 3.8 we plot histogram approximations of the density $P_W(w)$ of $W^{(\tau-1)}$ along with the weighted density proportional to $P_W(w)\exp(-w)$. The separation of the peaks of this distribution highlight how NEUS is able to effectively sample the low work tails that contribute significantly to the expectation in the Jarzynski relation in (3.75) but are rarely accessed by the switching procedure in the unbiased simulations.

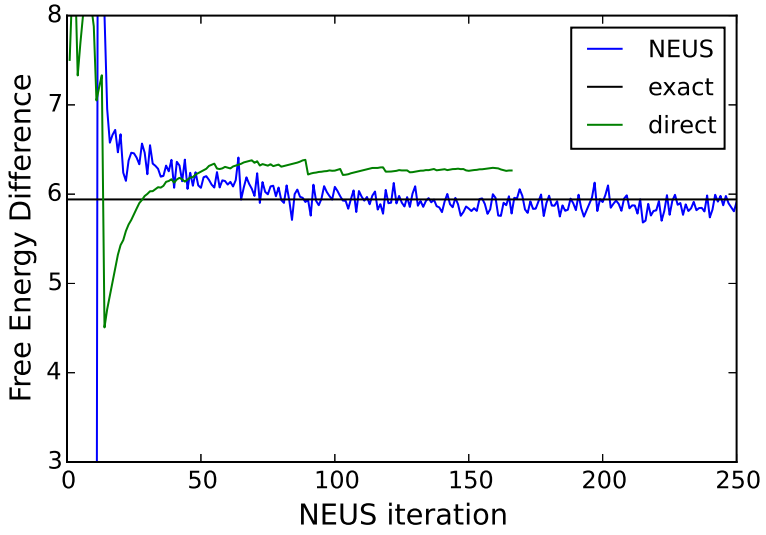


Figure 3.7: Estimate of the free energy computed from NEUS (blue) and from conventional fast-switching simulations (green). The value computed from numerically integrating the potentials is shown as a black line. For the direct fast-switching simulations, we scale the number of repetitions to the number of NEUS iterations that are equivalent in computational effort.

3.6 Conclusions

We describe a trajectory stratification framework for the estimation of expectations with respect to arbitrary Markov processes. The basis for this framework is the nonequilibrium umbrella sampling method (NEUS) originally introduced to compute steady state averages. Our development highlights the structural similarities between the nonequilibrium and equilibrium US algorithms and places the NEUS method within the general context of stochastic approximation. These connections have practical implications for further optimizing the procedure and point the way to a more in depth convergence analysis that will be the subject of future work.

Our development reveals that the basic trajectory stratification approach can be useful well beyond the estimation of stationary averages for time-homogenous Markov processes. This flexibility is demonstrated in two examples, both involving an expectation over trajectories of finite duration. In the first example, we show that the probability of first hitting a

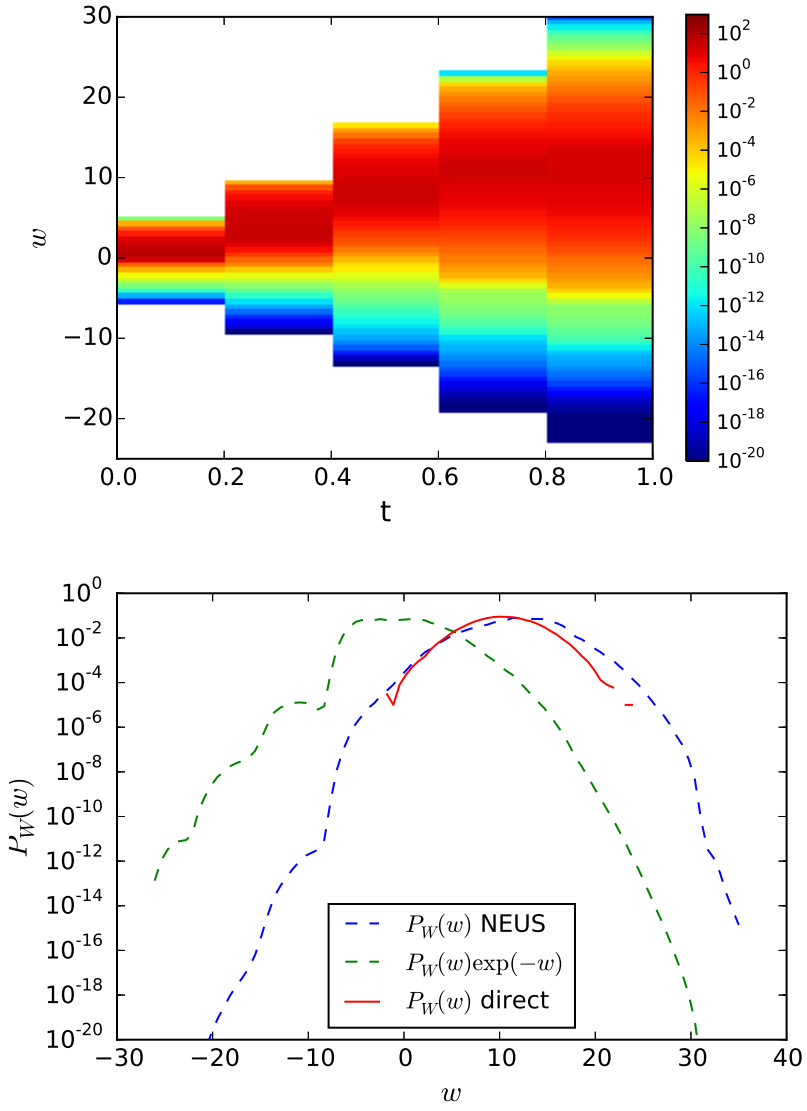


Figure 3.8: Sampling the work with NEUS. (top) The estimate of the dynamic weights, z_j, from the final iteration of the NEUS calculation. White space represents subsets that are not visited in the NEUS calculation. (bottom) The probability density $P_W(w)$ of the accumulated work $W^{(\tau-1)}$ estimated from NEUS (blue dashed line), from direct integration (red solid line) and the exponentially scaled probability density proportional to $P_W(w)\exp(-w)$ estimated from the NEUS calculation (green dashed line).

set within a finite time can be efficiently computed via stratification even when the dynamics start close to a competing absorbing state. In our second example, we use NEUS to stratify a process according to a path-dependent variable, the accumulated work in a nonequilibrium process appearing in the Jarzynski equation. The result is a novel and effective scheme for estimating free energy differences by enhancing sampling of the tails of the accumulated work distribution.

Our general framework also suggests new and exciting applications of trajectory stratification. For example, with little modification, these methods can be applied to sequential data assimilation applications where the goal is to approximate averages with respect to the conditional distribution of a hidden signal $X^{(t)}$ given sequentially arriving observations (i.e., with respect to the posterior distribution). In high-dimensional settings (e.g., weather forecasting) the only practical alternatives are limited to providing information about only the mode of the posterior distribution (i.e., variational methods) or involve uncontrolled and often unjustified approximations (i.e., Kalman-type schemes). The approach that we present here opens the door to efficient data assimilation, machine learning, and, more generally, new forms of analysis of complex dynamics.

CHAPTER 4

ENHANCED SAMPLING TOOLKIT

We introduce a Python-based software package called the Enhanced Sampling Toolkit for the rapid prototyping and development of enhanced sampling algorithms. The Enhanced Sampling Toolkit is a Python package that wraps molecular dynamics software and provides a high-level Python library for rapidly developing enhanced sampling algorithms. The toolkit is designed to facilitate rapid development cycles of enhanced sampling algorithms for testing and deployment in high-performance computing (HPC) environments.

4.1 Introduction

Computer simulation is an essential tool for the study of physical systems. However, many of the most interesting phenomena occur ofn timescales that are many orders of magnitude longer than we can tractably simulate. To address this challenge, a wide variety of algorithms have been developed to enhance the sampling of these rare events [25].

Effective application of rare event algorithms requires software implementations that can be developed and deployed in high-performance computing (HPC) environments. A common strategy in developing these implementations is to incorporate the enhanced sampling routines directly into popular molecular dynamics packages. For example, CHARMM [7], NAMD [71], GROMACS [1], AMBER [9], OpenMM [22] and LAMMPS [72] all natively support a collection of enhanced sampling algorithms. These implementations are easily applied in HPC environments but are limited to use with the underlying dynamics code and are not portable to models or dynamics routines that are not supported by the original package. This situation frequently requires that application developers laboriously port implementations of the enhanced sampling algorithm code into their package of choice.

Recently, software packages such as COLVARS [24], PLUMED [84], WESTPA [93], Open-

PathSampling, and SSAGES were developed to provide portable implementations of popular enhanced sampling algorithms. These codes interface with many of the most popular molecular dynamics codes either through the shell or by providing hooks that can be compiled into existing dynamics routines. These codes are typically written in the C/C++ or Python languages and provide a scriptable front end to drive the underlying dynamics engine. Although highly portable, these solutions do not readily allow developers to rapidly prototype novel sampling algorithms that are not already supported by these packages.

There is a great need for software tools that facilitate rapid prototyping and testing of novel enhanced sampling implementations. Researchers frequently need to iterate quickly on the implementation of an enhanced sampling algorithm over the course of the development of a project. Software tools that support rapid testing and implementation cycles in a fast and effective manner can significantly reduce the overall time required to develop novel algorithms.

To address the need for a fast and portable prototyping environment, we developed a Python-based toolkit called the Enhanced Sampling Toolkit to accelerate the development and prototyping of novel enhanced sampling algorithms in HPC environments. The specific objectives of the toolkit are:

1. To enable rapid development of novel enhanced sampling algorithms in the Python language.

2. To facilitate easy porting of enhanced sampling implementations between commonly used or custom dynamics engines.

3. To provide a prototyping environment suitable for deployment in HPC environments.

We chose to write the Enhanced Sampling Toolkit in the Python language owing to its inherent flexibility and widely supported scientific software environment. An increasing portion of the scientific community has adopted Python as their base language for its flexibility

and relative ease of use. The Python language was chosen as the core language of the Enhanced Sampling Toolkit to take advantage of the deep and well-developed scientific software stack and to exploit the inherent flexibility of the Python language to further the goals of the Enhanced Sampling Toolkit.

4.2 Overview of Features and Organization

The Enhanced Sampling Toolkit consists of two parts. The core of the toolkit is the Walker Application Programing Interface (API). The role of the Walker API is to provide a layer of abstraction between the routines that implement the enhanced sampling algorithm and the routines that implement the dynamics routines. The Walker API defines a set of core interactions that an enhanced sampling algorithm requires of an instance of a simulation. In addition to providing implementations of the Walker API for common dynamics engines, the Enhanced Sampling Toolkit provides a library of high-level Python-based software tools that allow developers to rapidly implement novel applications of common enhanced sampling algorithms.

4.2.1 The Walker Application Programming Interface

We consider a "Walker" to represent a single instance of a simulation. The Walker API is a specification that defines a library of interactions that an algorithm code has with a dynamics engine in order to drive the sampling of the dynamics. The API provides a standard and model-agnostic interface with single or multiple instances of a dynamics engine. A list of the currently available API routines are provided in Table 4.1.

The Walker API specification defines routines that perform one of two types of interactions. The first of type of routine passes information from the dynamics engine to the algorithm code. For example, the algorithm code can retrieve state arrays such as the position or velocities of particles in a molecular dynamics system or state arrays of collective

Table 4.1: Core Walker API Routines

API call	Description
get_position()	Return the coordinates of the system
set_position()	Set the coordinates of the system
get_velocitiy()	Return the velocities of the system
set_velocity()	Set the velocities of the system
draw_velocity()	Draw new velocities
reverse_velocity()	Scale the velocities
equilibrate()	Prepare the system at a particular value of the collective variables
get_colvars()	Return the state array of collective variables
add_colvars()	Add / modify new collective variables
propagate()	Integrate the equations of motion
close()	Destroy the walker instance
set_temperature()	Set temperature of the system
set_timestep()	Set time step for integration routine
get_time()	Return the physical time of the model
set_time()	Set the physical time of the model

variables. The second type of routine alters the state of the dynamics engine. For example, the algorithm-facing routines can pass state arrays back to the dynamics code, integrate the equations of motion or set the physical time of the dynamics engine. These interactions provide a flexible programming interface through which implementations of enhanced sampling algorithms can drive the underlying dynamics engines.

The Walker API provides a layer of abstraction between the dynamics engine and the algorithm code. An illustration is provided in Figure 4.1. This layer of abstraction provides several advantages that support rapid development of algorithms. First, the algorithm level implementation does not depend directly on the specifics of the underlying model. This allows algorithm implementations to be ported between different dynamics engines with minimal changes to the algorithm code. To date, the Enhanced Sampling Toolkit provides Walker API backends for the LAMMPS and OpenMM molecular dynamics codes.

Second, the Walker API is easily extensible to new or custom dynamics engines. The only new code required to support a new dynamics engine is an implementation of the Walker

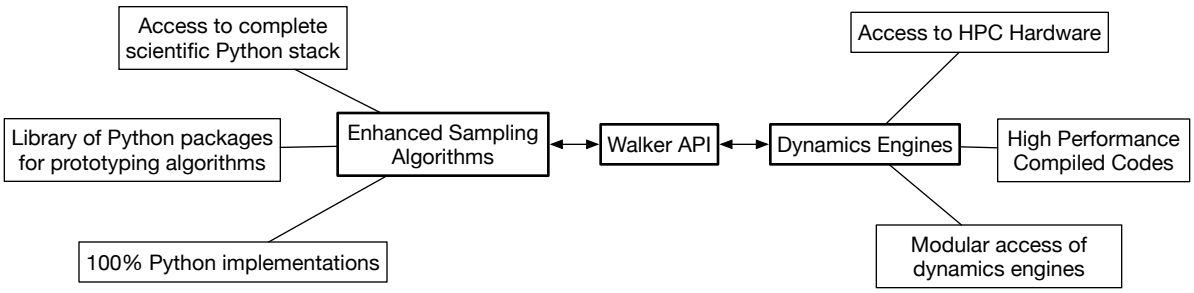


Figure 4.1: Illustration of the structure of the Enhanced Sampling Toolkit

API routines for that engine. If the algorithm-level code requires only a subset of the Walker API specification, developers can choose to implement only the subset of routines that are relevant to their specific dynamics engine. This flexibility allows the Enhanced Sampling Toolkit to be implemented for models at multiple physical scales, i.e. the Walker API can be implemented for atomic-resolution models, coarse-grained particle-based models, and stochastic field models. The flexibility of the Walker API further provides a simple way to implement simultaneous use of different dynamics engines at multiple scales.

An illustration of the type of model-agnostic scripting that the Walker API provides is provided by the example Monte Carlo script in Figure 4.2.1. This example illustrates how the Walker API can be used to quickly express the core elements of a simple Monte Carlo program in an efficient and expressive interface. In this implementation, the main body of the Monte Carlo script is written in 15 lines of Python. The Walker API also readily transitions to developing more sophisticated sampling applications such as the NEUS algorithm.

4.2.2 Library of Enhanced Sampling Tools

In addition to the Walker API, the Enhanced Sampling Toolkit provides a library of Python-based tools for rapidly developing enhanced sampling algorithms. The library is a repository of common enhanced sampling algorithms provided as modular Python packages. To date, the toolkit includes an application library for the NEUS method as described in Chapter 3, the conventional replica exchange molecular method [80] and the string method with

Figure 4.2: Example Monte Carlo program implemented using the Walker API.

```
import numpy as np
from est.walker import lammps_walker

#  Set some parameters, list for samples
nsamples = 100
samples = []

#  initialize a walker object from file "input.lammps"
wlkr = lammps_walker.Lammps("/path/to/input.lammps",
                            log=None,  #  set no lammps log file
                            index=0,  #  internal indexing
                            debug=False)  #  turn off logging

#  set initial state of the walker
initial_pos = np.loadtxt("/path/to/corodinate_array")
wlkr.set_position(initial_pos)
wlkr.draw_velocity()

#  set first state
old_config = wlkr.get_position()

#  enter sampling loop
for i in range(nsamples):

        #  propagate dynamics
        wlkr.propagate(100)

        #  get new position vector
        new_config = wlkr.get_position()

        #  implement custom metropolis criteria in Python routine
        if not metropolisMove(old_config, new_config):
                wlkr.set_position(old_config)

        samples.append(wlkr.get_position())
```

swarms of trajectories [68]. All of the results in Chapter 3 are developed using the NEUS module and the Walker API of the Enhanced Sampling Toolkit. The goal of this library is to accelerate the development of Python-based enhanced sampling implementations by providing a high-level toolkit that developers can use in their own implementations.

The library is written in Python and is implemented on top of the Walker API. Like the Walker API, the implementations provided by the toolset are model-agnostic and benefit from the portability and flexibility provided by the API. The library is easily used alongside other parts of the Python scientific software ecosystem and supports simultaneous use of different library modules. This feature extends its flexibility to develop hybrid enhanced sampling schemes and exploit other scientific Python libraries with relative ease. Because the Enhanced Sampling Toolkit is 100% pure Python, the broad and widely-supported software stack in Python is available out-of-the-box for use in developing algorithms.

4.2.3 HPC features

Although the primary focus of the toolkit is to enable rapid prototyping and testing, the toolkit can effectively take advantage of HPC environments. Implementations of enhanced sampling algorithms exploit two levels of parallelism to accelerate application performance. The first is algorithm-level parallelism where multiple copies of the system are simulated in parallel to accelerate sampling. The second level is dynamics-level parallelism where the routines that execute the integration of the underlying equations of motion of the system exploit multiple cores to accelerate performance.

By abstracting the algorithm-level code away from the dynamics engine code, the Walker API allows developers to effectively take advantage of both levels of parallelism. Algorithm-level parallelism is supported in the Enhanced Sampling Toolkit using standard Python-based MPI bindings, such as the *mpi4py* package. This package allows data structures to be communicated between walker instances using standard high-performance MPI implemen-

tations.

The dynamics engines supported by the Walker API, such as the OpenMM and LAMMPS packages, provide native support for dynamics-level parallelism through MPI and OpenMP as well as support for compute accelerators such as NVIDIA GPUs or Intel Phi cards. Using these dynamics engines with the Walker API allows developers to use the optimized dynamics-level parallelism already available in popular molecular dynamics packages to accelerate the computationally intensive integration routines. The Enhanced Sampling Toolkit allows developers to port and test toolkit implementations that use standard parallel programming paradigms such as MPI and rapidly port these implementations to take advantage of HPC architectures.

4.2.4 Developer Resources

The Enhanced Sampling Toolkit is publicly available on github (`https://github.com/jtempkin/enhanced_sampling_toolkit`). The public release contains several reference implementations of the NEUS algorithm including tutorials contained in Jupyter notebooks where users can explore the toolkit features in a hands-on environment. Installation and quick-start instructions are provided in the online documentation provided with the Github repository.

The Enhanced Sampling Toolkit is developed and hosted through the Github website which enables distributed contributions to the package. Unit testing frameworks are provided through the *pytest* package. Examples of developing unit testing modules are provided in the documentation.

4.3 Summary

We present a Python-based toolkit for rapidly prototyping enhanced sampling algorithms. The toolkit provides two central features, the Walker API and a library of Python modules.

The Walker API is a standard specification that allows algorithm-level code to one or more instances of a dynamics engine. The Walker API provides a layer of abstraction between the algorithm-level code and the dynamics engine that enables rapid prototyping of high-level algorithmic code using a model-agnostic framework. This allows the Enhanced Sampling Toolkit implementations to be highly flexible, easy to program and readily portable to HPC systems.

The Enhanced Sampling toolkit also provides a library of tools and reference implementations of common enhanced sampling algorithms. These libraries are provided as modular Python packages and are written to be highly expressive and flexible tools for developing custom algorithm implementations.

Finally, the Enhanced Sampling Toolkit exploits the Walker API to facilitate the use of HPC features. Implementations using the Enhanced Sampling Toolkit are readily ported to HPC architectures using native support for dynamics-level parallelism and standard Python-tools for algorithm-level parallelism. The Enhanced Sampling Toolkit allows developers to implement algorithm level code in a fast and flexible toolkit while executing the most computationally demanding dynamics routines in fast compiled codes that have native support for HPC architectures.

The flexibility and portability of the Enhanced Sampling Toolkit enables programmers to develop, test and deploy novel applications of enhanced sampling algorithms in a rapid development cycle. Rapid development and iteration of software implementations significantly decreases the investment of programming time required to address pressing scientific problems in simulation.

CHAPTER 5

OUTLOOK

In this work we develop several methods for enhancing the sampling of rare dynamical events. In Chapter 2, we introduce a multilevel preconditioning scheme for accelerating iterative molecular calculations. The preconditioning scheme provides a framework for using a less expensive model to guide the exploration of an energy landscape of a more expensive but more accurate model. Our method departs from similar multilevel strategies in that it preserves the stability of the fixed points of the more accurate model and formally guarantees that solutions to the multilevel fixed point equation are also solutions to the expensive but accurate model. We demonstrate how this multilevel preconditioning method can be used to accelerate the convergence of geometry optimizations and the string method for reaction path refinement. One potential application of the preconditioning method is to combine the results of Chapter 2 and Chapter 3 and develop a suitable modification of the preconditioning scheme to accelerate convergence of the self-consistent iteration at the heart of the NEUS method. In principle, a suitable choice of coarse grained model could potentially be used to precondition the update of the approximate conditional flux distributions, $\tilde{\gamma}_{ij}$, and the weights, $\tilde{z}_i$.

In Chapter 3, we develop a mathematical framework for trajectory stratification and demonstrate how this framework leads to the development of a novel class of rare event sampling methods suitable for computing averages taken with respect to arbitrary non-stationary stochastic dynamics. In particular, we show how this novel class of methods can compute a much broader class of dynamical expectations than was previously tractable.

In particular, trajectory stratification methods have potentially useful applications in sampling the conformational dynamics of intrinsically disordered proteins. One such example could be in sampling the conformational changes in molecular dynamics models of proteins that are probed by various spectroscopy experiments. Recent developments in NMR and IR

spectroscopy have provided powerful new experimental methods for studying intrinsically disordered proteins. However, sampling the molecular dynamics of these conformational dynamics often involves sampling potentially rare trajectories. These kinds of computational challenges can be addressed by the trajectory stratification methods developed here.

The development of the non-stationary trajectory stratification method opens up potential new avenues into how this novel class of algorithms can be applied to sample non-stationary stochastic processes. In particular, the application of trajectory stratification to the sampling of path-integrated variables such as the work accumulated in a driven process introduces a novel and previously unexplored class of rare event methods. There are a few potentially powerful applications to be explored in this direction. As a first step, it is useful to develop a more quantitative understanding of how stratifying the accumulated work in a driven process performs in comparison to more traditional free energy methods like umbrella sampling. In addition, the use of stratification to compute exponentials of path-integrated variables may have important applications in efficiently computing parameter sensitivities and in efficiently computing solutions to stochastic optimal control problems. The development of more efficient algorithms for solving these problems has the potential to be particularly useful, since stochastic optimal control problems arise in a diverse range of fields such as physics, machine learning, finance and engineering.

In developing the trajectory stratification framework, we derive a fixed point equation describing the relationship between the flux distributions and the normalizations constants. We interpret the NEUS method as a self-consistent iterative procedure that solves this fixed point equation in the context of stochastic approximations. This description lays the groundwork for a more formal numerical analysis of the method. One direction to pursue from a formal analysis of the NEUS method is more general understanding of how the structure of the index process, $J^{(t)}$, effects convergence and error of the algorithm. It is clear from our work that convergence of the algorithm depends sensitively on the choice of the $J^{(t)}$ process.

However, there is not a general understanding of how to optimally construct $J^{(t)}$. Results in this direction are potentially very useful in applying the NEUS method to complex systems.

APPENDIX A

AN ALTERNATIVE F

Here we present an alternative construction of the stochastic matrix F (3.2.1) that more closely aligns with the nonequilibrium version of the algorithm presented in 3.2.2. Suppose that one has available a transition distribution $p(dy \,|\, x)$ for a Markov chain that preserves (or nearly preserves) the target density, π, in the sense that

$$\pi(dy) = \int_{x \in \mathbb{R}^d} p(dy \,|\, x) \pi(dx).$$

For example, $p(dy \,|\, x)$ might be the transition density for a number of steps of a Langevin dynamics integrator. We can again express the z_i as the solution to an eigenproblem (3.8) where now

$$F_{ij} = \int_{y \in \mathbb{R}^d} \int_{x \in \mathbb{R}^d} \psi_j(y) p(dy \,|\, x) \pi_i(dx). \tag{A.1}$$

Note that when $\psi_i(x) = \mathbf{1}_{A_i}$ for some partition of space $\{A_i\}$, and $p(dy \,|\, x)$ is reversible with respect to π, the entry F_{ij} can be estimated by evolving samples according to $p(dy \,|\, x)$, rejecting any proposed samples that lie outside of A_i (so that π_i is preserved), and then counting the number of times the chain attempts transitions from set A_i to set A_j. For a closely related approach to approximating certain nonequilibrium quantities see [87].

APPENDIX B

INVARIANCE OF Π_I FOR $Y_I^{(R)}$

Lemma B.0.1. *The measure $\Pi_i(t, dx; \tilde{G}, \tilde{\gamma})$ defined in (3.27) is invariant for the process $Y_i^{(r)}(\tilde{G}, \tilde{\gamma})$ defined in 3.3.3.*

Proof. To simplify notation, we write $\Pi_j(s, dy)$ for $\Pi_j(s, dy; \tilde{G}, \tilde{\gamma})$, and we do the same for $\bar{\Pi}_j$ and $Y_j^{(r)}$. The kernel of the process $Y_j^{(r)}$ is

$$Q(t, x; s, dy) = \mathbf{P}^{\text{acc}}_{t,x,j}(dy)\mathbf{1}_{\{t+1\}}(s) + \mathbf{P}^{\text{rej}}_{t,x,j}\bar{\Pi}_j(s, dy),$$

where

$$\mathbf{P}^{\text{acc}}_{t,x,j}(dy) = \mathbf{P}_{t,x,j}\left[t+1 < \tau,\, X^{(t+1)} \in dy,\, J^{(t+1)} = j\right],$$

and

$$\mathbf{P}^{\text{rej}}_{t,x,j} = \mathbf{P}_{t,x,j}\left[t+1 = \tau \text{ or } J^{(t+1)} \neq j\right].$$

We will assume at first that $s > 0$. Applying the kernel Q to Π_j yields

$$\begin{aligned}
&\sum_{t=0}^{\infty}\int_x Q(t, x; s, dy)\Pi_j(t, dx) \\
&\quad\propto \sum_{t=0}^{\infty}\mathbf{1}_{\{t+1\}}(s)\sum_{r=0}^{t}\int_z\left(\int_x \mathbf{P}^{\text{acc}}_{t,x,j}(dy)\mathbf{P}_{r,z,j}\left[t < \sigma(r)\vee\tau,\, X^{(t)} \in dx\right]\right)\bar{\Pi}_j(r, dz) \\
&\qquad + \bar{\Pi}_j(s, dy)\sum_{t=0}^{\infty}\sum_{r=0}^{t}\int_z\left(\int_x \mathbf{P}^{\text{rej}}_{t,x,j}\mathbf{P}_{r,z,j}\left[t < \sigma(r)\vee\tau,\, X^{(t)} \in dx\right]\right)\bar{\Pi}_j(r, dz) \\
&\quad= \sum_{t=0}^{\infty}\mathbf{1}_{\{t+1\}}(s)\sum_{r=0}^{t}\int_z \mathbf{P}_{r,z,j}\left[t+1 < \sigma(r)\vee\tau,\, X^{(t+1)} \in dy\right]\bar{\Pi}_j(r, dz) \\
&\qquad + \bar{\Pi}_j(s, dy)\sum_{t=0}^{\infty}\sum_{r=0}^{t}\int_z \mathbf{P}_{r,z,j}\left[t+1 = \sigma(r)\vee\tau\right]\bar{\Pi}_j(r, dz) \\
&\quad= \sum_{r=0}^{s-1}\int_z \mathbf{P}_{r,z,j}\left[s < \sigma(r)\vee\tau,\, X^{(s)} \in dy\right]\bar{\Pi}_j(r, dz)
\end{aligned}$$

$$
\begin{aligned}
&\quad + \bar{\Pi}_j(s, dy) \sum_{r=0}^{\infty} \int_z \left(\sum_{t=r}^{\infty} \mathbf{P}_{r,z,j} \left[t + 1 = \sigma(r) \vee \tau \right] \right) \bar{\Pi}_j(r, dz) \\
&= \Pi_j(s, dy) - \bar{\Pi}_j(s, dy) \\
&\quad + \bar{\Pi}_j(s, dy) \left(\sum_{r=0}^{\infty} \int_z \mathbf{P}_{r,z,j} \left[\sigma(r) \vee \tau < \infty \right] \bar{\Pi}_j(r, dz) \right).
\end{aligned}
$$

In the second to last equality, the notation $\mathbf{P}_{r,z,t}$ encodes a condition on $r < \tau$ and the probability in the summand becomes a delta mass at dy yielding $\bar{\Pi}_j(s, dy)$. In the final expression, the condition $\mathbf{E}[\tau] < \infty$ implies $\sigma(r) \vee \tau < \infty$ with probability one. Thus, for $s > 0$,

$$
\sum_{t=0}^{\infty} \int_x Q(t, x; s, dy) \Pi_j(t, dx) \propto \Pi_j(s, dy).
$$

A very similar argument shows that the same expression holds when $s = 0$. Thus, Π_j is invariant for $Y_j^{(r)}$. □

APPENDIX C

AN ALTERNATIVE IMPLEMENTATION

The Exact Milestoning method recently proposed for computing certain dynamic averages with respect to a stationary time-homogenous Markov process [5] has a structure very similar to the steady-state version of NEUS that we have described in 3.4. The differences in implementation between the two methods suggest an alternative implementation of NEUS in the more general context of 3.2.2 and 3.3.

To describe this alternative implementation, it is convenient to work with a slightly different set of variables that reflects the interface-to-interface emphasis in Milestoning. Here we show how these new variables are equivalent to the variables that we used to describe NEUS. In particular, we define the substochastic matrix

$$\begin{aligned} H_{ij} &= \frac{\sum_{\ell=0}^{\infty} \mathbf{P}\left[S^{\ell+1} < \tau,\, J^{(S^{\ell+1})} = j,\, J^{(S^{(\ell)})} = i\right]}{\sum_{\ell=0}^{\infty} \mathbf{P}\left[J^{(S^{(\ell)})} = i,\, S^{\ell} < \tau\right]} \\ &= \sum_{t=0}^{\infty} \int_{x\in\mathbb{R}^d} \mathbf{P}_{t,x,i}\left[\sigma(t) < \tau,\, J^{(\sigma(t))} = j\right]\, \bar{\pi}_i(t, dx) \end{aligned}$$

which can be expressed in terms of G by

$$H_{ij} = \begin{cases} G_{ij}/(1 - G_{ii}), & i \neq j \\ 0, & i = j. \end{cases}$$

We also define

$$v_j = \sum_{\ell=0}^{\infty} \mathbf{P}\left[S^{\ell} < \tau,\, J^{(S^{(\ell)})} = i\right] = z_j(1 - G_{jj}),$$

which satisfies

$$v^{\mathrm{T}} = v^{\mathrm{T}} H + a^{\mathrm{T}},$$

and we define the matrix of subprobability distributions (they do not integrate to 1)

$$\xi_{ij}(s, dy) = \sum_{t=0}^{\infty} \int_{x \in \mathbb{R}^d} \mathbf{P}_{t,x,i} \left[\sigma(t) = s,\, s < \tau,\, X^{(s)} \in dy,\, J^{(s)} = j \right] \bar{\pi}_i(t, dx)$$

which are related to the conditional flux distributions γ_{ij} as defined in (3.24) by

$$\xi_{ij} = \begin{cases} H_{ij}\, \gamma_{ij}, & i \neq j \\ 0, & i = j. \end{cases}$$

Note that $\sum_j \gamma_{ij}$ is the distribution of transition points leaving $J = i$. We have already seen in (3.25) that

$$v_j\, \bar{\pi}_j(s, dy) = \begin{cases} \sum_i v_i\, \xi_{ij}(s, dy), & \text{if } s > 0, \\ a_j\, \mathbf{P} \left[X^{(0)} \in dy \,|\, J^{(0)} = j \right], & \text{if } s = 0. \end{cases}$$

We will construct an algorithm that finds the unnormalized distribution $\theta_j = v_j \bar{\pi}_j$.

In light of these observations, and following the development in 3.3, for any vector of subprobability distributions $\{\tilde{\theta}_j\}$ approximating $\{\theta_j\}$, we define the functions

$$\Xi_{ij}(s, dy; \tilde{\theta}) = \frac{1}{\tilde{v}_i} \sum_{t=0}^{\infty} \int_{x \in \mathbb{R}^d} \mathbf{P}_{t,x,i} \left[\sigma(t) = s,\, s < \tau,\, X^{(s)} \in dy,\, J^{(s)} = j \right] \tilde{\theta}_i(t, dx),$$

and

$$\Theta_j(\tilde{\theta}) = \begin{cases} \sum_{i=1}^{n} \tilde{v}_i\, \Xi_{ij}(\tilde{\theta}), & \text{if } s > 0, \\ a_j\, \mathbf{P} \left[X^{(0)} \in dy \,|\, J^{(0)} = j \right], & \text{if } s = 0. \end{cases}$$

for $\tilde{v}_j = \sum_{t=0}^{\infty} \int_y \tilde{\theta}_j(s, dy)$. The fixed-point problem describing this alternative implementation finds a vector of time and position subprobability distributions $\{\theta_j\}$ satisfying

$$\Theta_j(\theta) = \theta.$$

Given the current iterate $\tilde{\theta}(m)$ with the vector of normalizations $\tilde{v}(m)$, the stochastic approximation algorithm corresponding to this fixed-point problem fixes a non-negative integer N and, for each i, initiates N independent trajectories of $(t, X^{(t)})$ from $\tilde{\theta}_i(m)/\tilde{v}_i(m)$, all of which are run until τ or the next transition of the index process. Suppose that N_i of these N trajectories end upon a transition of the index process (rather than upon reaching time τ). Let $\{(J_i^\ell(m), T_i^\ell(m), X_i^\ell(m))\}$, be the corresponding values of the index process, time, and position upon those transition events. Then we can define the empirical subprobability measures

$$\hat{\xi}_{ij}(s, dy; m) = \frac{1}{N} \sum_{\ell=1}^{N_i} \mathbf{1}_{\{j\}}(J_\ell^i(m))\, \mathbf{1}_{\{s\}}(T_i^\ell(m))\, \delta_{X_\ell^i(m)}(dy)$$

and set

$$\hat{\theta}_j(s, dy; m) = \begin{cases} \sum_{i=1}^n \tilde{v}_i(m)\, \hat{\xi}_{ij}(m), & \text{if } s > 0, \\ a_j\, \mathbf{P}\left[X^{(0)} \in dy \,|\, J^{(0)} = j\right], & \text{if } s = 0. \end{cases}$$

Note that $\mathbf{E}\left[\hat{\theta}_j(m)\right] = \Theta_j(\tilde{\theta}(m))$.

The alternative stochastic approximation algorithm is then

$$\tilde{\theta}_j(m+1) = \tilde{\theta}_j(m) + \varepsilon_m \left(\hat{\theta}_j(m) - \tilde{\theta}_j(m)\right). \tag{C.1}$$

An approximation of $\bar{\pi}_j$ is obtained at any time by normalizing $\tilde{\theta}_j(m)$.

Both the implementation described in this section and the one described in 3.3 store an empirical estimate of $\bar{\pi}$ (in 3.3 the estimate of γ directly corresponds through (3.27) to an estimate of $\bar{\pi}$). The core difference between the two implementations is that in this section one generates a fixed number of trajectory fragments for each value of the index process, while in the implementation in 3.3 the number of trajectory fragments generated is random but the total computational expense for each value of the index process is fixed. This alternative iteration has the advantage that the update in (C.1) is completely uncorrelated between iterations (conditioned on the current iterate). On the other hand, a potential disadvantage

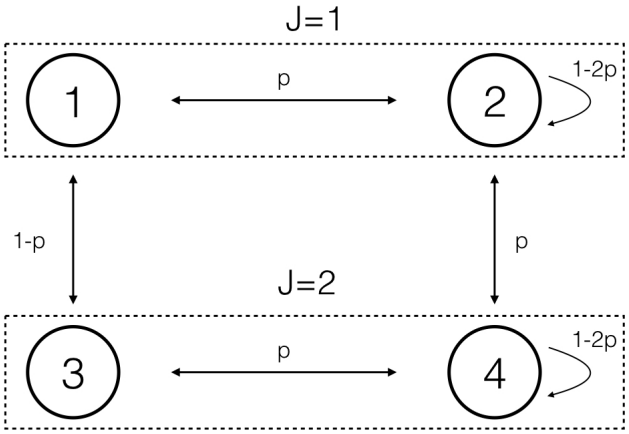
is that the random and varied computational effort for each value of the index process could make synchronization difficult in a parallel implementation.

APPENDIX D

ANALYSIS OF A SIMPLE MODEL IN THE SMALL MEMORY LIMIT

In this section, we analyze a simple tractable model in the limit where a single point is stored in each conditional flux distribution $\tilde{\gamma}_{ij}(m)$. We will show in this analysis that for a simple tractable model, the distribution of the states of $\tilde{\gamma}_{ij}(m)$ preserves the correct distribution of γ_{ij} on average as $m \to \infty$.

The model is a discrete Markov process $X^{(t)} \in \{1, 2, 3, 4\}$ with the structure



The chain has the transition matrix

$$T = \begin{bmatrix} 0 & p & 1-p & 0 \\ p & 1-2p & 0 & p \\ 1-p & 0 & 0 & p \\ 0 & p & p & 1-2p \end{bmatrix}. \tag{D.1}$$

and stationary distribution

$$v_{stat} = \begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix}. \tag{D.2}$$

For the purpose of this analysis, we compute the expected occupation of site $X^{(t)} = 2$, i.e. $f(x) = \mathbf{1}_2(x)$. Specifically, the expectation we compute is

$$\langle f \rangle = \sum_{x=1}^{4} \mathbf{1}_2(x)\pi(x) = 1/4. \tag{D.3}$$

The $J^{(t)}$ process over which we stratify $X^{(t)}$ is the same as the definition in (3.48). To compute (D.3) by stratification, we define the conditional expectations, $\pi_i(x)$, against which the stratifications terms are computed in the long-time limit. In general, the coniditional expectations, $\pi_i(x)$, are

$$\pi_i(x) = \lim_{\tau\to\infty} \frac{1}{\tau} \sum_{t=0}^{\tau-1} \mathbf{P}\left[X^{(t)} = x | J^{(t)} = i\right]. \tag{D.4}$$

In this simple model, for $J = 1$

$$\pi_1(x) = \begin{cases} 1/2 & \text{if } x = 1 \\ 1/2 & \text{if } x = 2 \\ 0 & \text{otherwise} \end{cases} \tag{D.5}$$

and for $J = 2$

$$\pi_2(x) = \begin{cases} 1/2 & \text{if } x = 3 \\ 1/2 & \text{if } x = 4 \\ 0 & \text{otherwise.} \end{cases} \tag{D.6}$$

The normalization constants are

$$z_i = \lim_{\tau \to \infty} \frac{1}{\tau} \sum_{t=0}^{\tau-1} \mathbf{P}\left[J^{(t)} = i\right] \tag{D.7}$$

and for this model, $z_1 = z_2 = 1/2$. The definition of G is

$$G_{ij} = \lim_{t \to \infty} \mathbf{P}\left[J^{(t+1)} = j | J^{(t)} = i\right]. \tag{D.8}$$

and the exact matrix for this model is

$$G = \begin{bmatrix} 1/2 & 1/2 \\ 1/2 & 1/2 \end{bmatrix}. \tag{D.9}$$

The solution to eigenproblem $z^T G = z^T$ is easily verified to be $z_1 = z_2 = 1/2$. We compute (D.3) by stratification as

$$\langle f \rangle = z_1 \langle f \rangle_1 + z_2 \langle f \rangle_2 = 1/2 * 1/2 + 0 = 1/4. \tag{D.10}$$

Having outlined the stratification terms for this simple model, we now consider the NEUS scheme in the limit where only a single copy of the system is stored in each conditional flux distribution $\tilde{\gamma}_{ij}(m)$. For the purpose of this analysis, assume that expectations are computed exactly at each NEUS iteration.

In the small buffer limit, each conditional flux distribution $\tilde{\gamma}_{ij}(m)$ consists of a single point. For $J = 1$, $\tilde{\gamma}_{21}(m)$ stores either state $x = 1$ or state $x = 2$. For $J = 2$, $\tilde{\gamma}_{12}(m)$ stores

either state $x = 3$ or state $x = 4$. There are four possible global states for the buffers,

$$\tilde{\gamma}(m) = \begin{cases} (1,3) \\ (1,4) \\ (2,3) \\ (2,4) \end{cases} . \tag{D.11}$$

In NEUS, the state of the conditional flux distributions is updated at each iteration. In the following analysis we compute an effective transition matrix, T_{buff}, for the transitions between the possible states of $\tilde{\gamma}$.

In order to compute T_{buff}, we first compute the effective chains for the restricted processes, $Y_i^{(r)}(m)$. The dynamics of the process $Y_i^{(r)}(m)$ depend on the of the current state of $\tilde{\gamma}_{ji}(m)$. The notation we use is $Y_i(k)$ which represents the process $Y^{(r)}$ that samples the restricted distribution corresponding to $J = i$ but with the buffer $\tilde{\gamma}_{ji}(m)$ containing only the state $X^{(t)} = k$. For example, $Y_1(1)$ is the process that samples $J = 1$ with the $\tilde{\gamma}_{21}(m)$ containing only state $X^{(t)} = 1$.

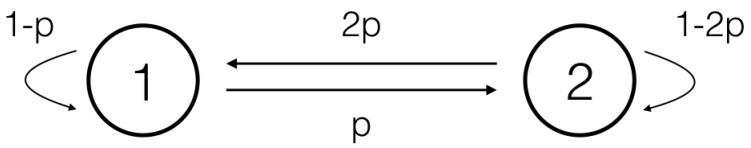
There are only four restricted processes that are possible in the small-buffer limit, $Y_1(1)$, $Y_1(2)$, $Y_2(3)$ and $Y_2(4)$. We compute the effective chain for each process. The entries of T_{buff} for each restricted process is computed by enumerating each possible transition. For $Y_1(1)$, the possible steps are:

- $Y_1^{(r)}(1)$ transitions from $1 \to 2$ by proposing a transition from $X^{(t)} = 1 \to X^{(t+1)} = 1$ with probability p.
- $Y_1^{(r)}(1)$ transitions from $1 \to 1$ by proposing a transition from $X^{(t)} = 1 \to X^{(t+1)} = 3$ with probability $1 - p$ and drawing $Y_1^{(r+1)}(1) = 1$ from $\tilde{\gamma}_{21}(m)$ with probability one.
- $Y_1^{(r)}(1)$ transitions from $2 \to 1$ by proposing a transition from $X^{(t)} = 2 \to X^{(t+1)} = 1$ with probability p or by proposing a transition from $X^{(t)} = 2 \to X^{(t)} = 4$ with

probability p and drawing $Y_1^{(r+1)}(1) = 1$ from the buffer with probability one. This transition has overall probability p.

- $Y_1^{(r)}(1)$ transitions from $2 \to 2$ by proposing a transition from $X^{(t)} = 2 \to X^{(t+1)} = 2$ with probability $1 - 2p$.

The effective chain for $Y_1(1)$ is



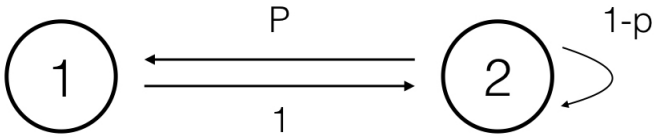
which has transition matrix

$$T^{Y_1(1)} = \begin{bmatrix} 1-p & p \\ 2p & 1-2p \end{bmatrix} \tag{D.12}$$

and stationary distribution

$$v^{Y_1(1)} = \begin{bmatrix} 2/3 \\ 1/3 \end{bmatrix}. \tag{D.13}$$

Similarly, the effective chain for $Y_1(2)$ is



which has transition matrix

$$T^{Y_1(2)} = \begin{bmatrix} 0 & 1 \\ p & 1-p \end{bmatrix} \tag{D.14}$$

and stationary vector

$$v^{Y_1(2)} = \begin{bmatrix} \frac{p}{p+1} \\ \\ \frac{1}{p+1} \end{bmatrix}. \tag{D.15}$$

Enumerating the effective chains of the remaining processes gives

$$v^{Y_2(3)} = \begin{bmatrix} 2/3 \\ 1/3 \end{bmatrix} \tag{D.16}$$

and

$$v^{Y_2(4)} = \begin{bmatrix} \frac{p}{p+1} \\ \\ \frac{1}{p+1} \end{bmatrix}. \tag{D.17}$$

At each iteration of the NEUS algorithm, the new state of each $\tilde{\gamma}_{ij}(m)$ is chosen by drawing the new state proportional to the probability that those states were generated by the chain $Y_i^{(r)}$. We compute the entries in T_{buff} by enumerating the possible transitions. For example, to compute the $(1,3) \to (1,3)$ entry in T_{buff} we first compute the probability that the process $Y_2(3)$ proposes transitions to site $x = 1$. This probability is the product of the first entry of $v^{Y_2(3)}$ and the probability of proposing a transition from $X^{(t)} = 3 \to X^{(t)} = 1$. The product of these probabilities is $2(1-p)/3$. Similarly, the probability of $Y_2(3)$ generating a transition point at $X^{(t)} = 2$ is $p/3$.

Therefore the probability of $\tilde{\gamma}_{21}(m)$ containing the site $X^{(t)} = 1$ at iteration $m+1$ is $2(1-p)/(2-p)$. The probability the buffer $\tilde{\gamma}_{12}(m)$ contains the state $X^{(t)} = 3$ at iteration $m+1$ is $2(1-p)/(2-p)$. Therefore the probability that we start at the buffer state $(1,3)$ and transition at the next NEUS iteration to the state $(1,3)$ is

$$\mathbf{P}\left[(1,3) \to (1,3)\right] = \left(\frac{2(1-p)}{(2-p)}\right)^2 = \frac{4(p-1)^2}{(2-p)^2} \tag{D.18}$$

Enumerating each entry as above, the full T_{buff} is

$$T_{buff} = \frac{1}{(p-2)^2} \begin{bmatrix} 4(p-1)^2 & 2p(1-p) & 2p(1-p) & p^2 \\ 2(p-1)^2 & p(1-p) & 2(1-p) & p \\ 2(p-1)^2 & 2(1-p) & p(1-p) & p \\ (p-1)^2 & 1-p & 1-p & 1 \end{bmatrix}. \tag{D.19}$$

The stationary vector for T_{buff} is

$$v_{buff} = \begin{bmatrix} (1,3) \\ (1,4) \\ (2,3) \\ (2,4) \end{bmatrix} = \begin{bmatrix} (1-p)^2 \\ p(1-p) \\ p(1-p) \\ p^2 \end{bmatrix}. \tag{D.20}$$

The entries to this matrix are the limiting probability of the state of the conditional flux distributions $\tilde{\gamma}_{ij}(m)$ as $m \to \infty$.

It is easy to verify that in the low-memory limit the NEUS scheme yields the correct proportion of each state of $\tilde{\gamma}_{ij}(m)$ on average as $m \to \infty$. The exact proportion of entries in γ_{21} is

$$\gamma_{21}(y) = \begin{cases} 1-p & \text{if } y = 1 \\ p & \text{if } y = 2 \end{cases} \tag{D.21}$$

From (D.20), the probability that $\tilde{\gamma}_{21}$ contains $X^{(t)} = 1$ is $(1-p)^2 + p(1-p) = 1-p$ and the remaining states can be readily checked as well.

BIBLIOGRAPHY

[1] Mark James Abraham, Teemu Murtola, Roland Schulz, Szilárd Páll, Jeremy C. Smith, Berk Hess, and Erik Lindahl. GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. *SoftwareX*, 1:19–25, 2015.

[2] Rosalind J Allen, Patrick B Warren, and Pieter Rein Ten Wolde. Sampling Rare Switching Events in Biochemical Networks. *Phys. Rev. Lett.*, 94:018104, jan 2005.

[3] Joannis Apostolakis, Philippe Ferrara, and Amedeo Caflisch. Calculation of conformational transitions and barriers in solvated systems: Application to the alanine dipeptide in water. *J. Chem. Phys.*, 110:2099, 1999.

[4] Søren Asmussen and Peter W Glynn. *Stochastic Simulation: Algorithms and Analysis*. Springer, 2007.

[5] Juan M Bello-Rivas and Ron Elber. Exact milestoning. *J. Chem. Phys.*, 142:94102, 2015.

[6] William L. Briggs, Van Emden Henson, and Steve F. McCormick. *A multigrid tutorial*. SIAM, 2nd edition, jan 2000.

[7] B. R. Brooks, C. L. Brooks, A. D. Mackerell, L. Nilsson, R. J. Petrella, B. Roux, Y. Won, G. Archontis, C. Bartels, S. Boresch, A. Caflisch, L. Caves, Q. Cui, A. R. Dinner, M. Feig, S. Fischer, J. Gao, M. Hodoscek, W. Im, K. Kuczera, T. Lazaridis, J. Ma, V. Ovchinnikov, E. Paci, R. W. Pastor, C. B. Post, J. Z. Pu, M. Schaefer, B. Tidor, R. M. Venable, H. L. Woodcock, X. Wu, W. Yang, D. M. York, and M. Karplus. CHARMM: The biomolecular simulation program. *J. Comp. Chem.*, 30(10):1545–1614, 2009.

[8] Eric J. Bylaska, Jonathan Q. Weare, and John H. Weare. Extending molecular simulation time scales: Parallel in time integrations for high-level quantum chemistry and complex force representations. *J. Chem. Phys.*, 139:074114, 2013.

[9] David A. Case, Thomas E. Cheatham, Tom Darden, Holger Gohlke, Ray Luo, Kenneth M. Merz, Alexey Onufriev, Carlos Simmerling, Bing Wang, and Robert J. Woods. The Amber biomolecular simulation programs. *J. Comp. Chem.*, 26(16):1668–1688, dec 2005.

[10] Frédéric Cérou and Arnaud Guyader. Adaptive Multilevel Splitting for Rare Event Analysis. *Stochastic Analysis and Applications*, 25(2):417–443, 2007.

[11] David Chandler. *Introduction to Modern Statistical Mechanics*. Oxford University Press, 1987.

[12] Christophe Chipot and Andrew Pohorille. *Free Energy Simulations*. Springer, 2007.

[13] Alex Dickson and Aaron R Dinner. Enhanced Sampling of Nonequilibrium Steady States. *Annu. Rev. Phys. Chem.*, 61:441–459, jan 2010.

[14] Alex Dickson, Mark Maienschein-Cline, Allison Tovo-Dwyer, Jeff R. Hammond, and Aaron R. Dinner. Flow-Dependent Unfolding and Refolding of an RNA by Nonequilibrium Umbrella Sampling. *J. Chem. Theory Comput.*, 7(9):2710–2720, sep 2011.

[15] Alex Dickson, Aryeh Warmflash, and Aaron R Dinner. Nonequilibrium umbrella sampling in spaces of many order parameters. *J. Chem. Phys.*, 130:074104, feb 2009.

[16] Alex Dickson, Aryeh Warmflash, and Aaron R Dinner. Separating forward and backward pathways in nonequilibrium umbrella sampling. *J. Chem. Phys.*, 131:154104, oct 2009.

[17] A R Dinner, E Thiede, B Van Koten, and J Weare. Stratification of Markov chain Monte Carlo sampling. *in prep.*

[18] Zeynep A Oztug Durer, Dmitri S. Kudryashov, Michael R. Sawaya, Christian Altenbach, Wayne Hubbell, and Emil Reisler. Structural states and dynamics of the D-Loop in actin. *Biophys. J.*, 103(5):930–939, 2012.

[19] Weinan E, Weiqing Ren, and Eric Vanden-Eijnden. String method for the study of rare events. *Phys. Rev. B*, 66:052301, 2002.

[20] Weinan E, Weiqing Ren, and Eric Vanden-Eijnden. Finite Temperature String Method for the Study of Rare Events. *J. Phys. Chem. B*, 109(14):6688–6693, 2005.

[21] Weinan E, Weiqing Ren, and Eric Vanden-Eijnden. Simplified and improved string method for computing the minimum energy paths in barrier-crossing events. *J. Chem. Phys.*, 126(16), 2007.

[22] Peter Eastman, Mark S. Friedrichs, John D. Chodera, Randall J. Radmer, Christopher M. Bruns, Joy P. Ku, Kyle A. Beauchamp, Thomas J. Lane, Lee-Ping Wang, Diwakar Shukla, Tony Tye, Mike Houston, Timo Stich, Christoph Klein, Michael R. Shirts, and Vijay S. Pande. OpenMM 4: A Reusable, Extensible, Hardware Independent Library for High Performance Molecular Simulation. *J. Chem. Theory Comput.*, 9(1):461–469, jan 2013.

[23] Anton K Faradjian and Ron Elber. Computing time scales from reaction coordinates by milestoning. *J. Chem. Phys.*, 120:10880, jun 2004.

[24] Giacomo Fiorin, Michael L. Klein, and Jérôme Hénin. Using collective variables to drive molecular dynamics simulations. *Mol. Phys.*, 111(22-23):3345–3362, dec 2013.

[25] Daan Frenkel and Berend Smit. *Understanding Molecular Simulation*. Academic Press, 2nd edition, 2002.

[26] Takashi Fujii, Atsuko H. Iwane, Toshio Yanagida, and Keiichi Namba. Direct visualization of secondary structures of F-actin by electron cryomicroscopy. *Nature*, 467(7316):724–728, 2010.

[27] Vitold E Galkin, Albina Orlova, Gunnar F Schröder, and Edward H Egelman. Structural polymorphism in F-actin. *Nat. Struct. Mol. Biol.*, 17:1318–1323, 2010.

[28] Wenxun Gan, Sichun Yang, and Benoît Roux. Atomistic View of the Conformational Activation of Src Kinase Using the String Method with Swarms-of-Trajectories. *Biophys. J.*, 97(4):L8–L10, 2009.

[29] Crispin W Gardiner. *Stochastic Methods: A Handbook for the Natural and Social Sciences*. Springer, 4th edition, 2009.

[30] Paul Glasserman, Philip Heidelberger, Perwez Shahabuddin, and Tim Zajic. *A Look At Multilevel Splitting*, pages 98–108. Springer New York, New York, NY, 1998.

[31] Jonathan Goodman and Alan D. Sokal. Multigrid Monte Carlo method. Conceptual foundations. *Phys. Rev. D*, 40:2035, 1989.

[32] Nicholas Guttenberg, Aaron R Dinner, and Jonathan Weare. Steered transition path sampling. *J. Chem. Phys.*, 136(23):234103, 2012.

[33] Martin Hairer and Jonathan Weare. Improved diffusion Monte Carlo. *Commun. Pure Appl. Math.*, 67(12):1995–2021, 2014.

[34] Zsolt Haraszti and J Keith Townsend. The Theory of Direct Probability Redistribution and Its Application to Rare Event Simulation. *ACM Trans. Model. Comput. Simul.*, 9(2):105–140, apr 1999.

[35] Jie Hu, Ao Ma, and Aaron R. Dinner. Bias annealing: A method for obtaining transition paths de novo. *J. Chem. Phys.*, 125:114101, 2006.

[36] G A Huber and S Kim. Weighted-Ensemble Brownian dynamics simulations for protein association reactions. *Biophys. J.*, 70:97–110, 1996.

[37] Gerhard Hummer. Fast-growth thermodynamic Integration: Error and efficiency analysis. *J. Chem. Phys.*, 114:7330–7337, 2001.

[38] Gerhard Hummer and Attila Szabo. Kinetics from Nonequilibrium Single-Molecule Pulling Experiments. *Biophys. J.*, 85(1):5–15, 2003.

[39] Marcella Iannuzzi, Alessandro Laio, and Michele Parrinello. Efficient Exploration of Reactive Potential Energy Surfaces Using Car-Parrinello Molecular Dynamics. *Phys. Rev. Lett.*, 90:238302, 2003.

[40] Christopher Jarzynski. Nonequilibrium Equality for Free Energy Differences. *Phys. Rev. Lett.*, 78:2690–2693, 1997.

[41] Christopher Jarzynski. Rare events and the convergence of exponentially averaged work values. *Phys. Rev. E*, 73:046105, apr 2006.

[42] A M Johansen, P Del Moral, and A Doucet. Sequential Monte Carlo samplers for rare events. In *Proceedings of the 6th International Workshop on Rare Event Simulation*, Bramberg, 2006.

[43] Seyit Kale, Olaseni Sode, Jonathan Weare, and Aaron R. Dinner. Finding chemical reaction paths with a multilevel preconditioning protocol. *J. Chem. Theory Comput.*, 10(12):5467–5475, 2014.

[44] Y Kanai, A Tilocca, A Selloni, and R Car. First-principles string molecular dynamics: An efficient approach for finding chemical reaction pathways. *J. Chem. Phys.*, 121:3359, 2004.

[45] Robert E Kass and Adrian E Raftery. Bayes Factors. *Journal of the American Statistical Association*, 90(430):773–795, 1995.

[46] Harold J Kushner and G George Yin. *Stochastic Approximations and Recursive Algorithms and Applications*. Springer, 2nd edition, 2003.

[47] Jérôme J. Lacroix, Stephan A. Pless, Luca Maragliano, Fabiana V. Campos, Jason D. Galpin, Christopher A. Ahern, Benoît Roux, and Francisco Bezanilla. Intermediate state trapping of a voltage sensor. *J. Gen. Physiol.*, 140(6):635–652, 2012.

[48] Frédéric Legoll, Tony Lelièvre, and Giovanni Samaey. A Micro-Macro Parareal Algorithm: Application to Singularly Perturbed Ordinary Differential Equations. *SIAM J. Sci. Comput.*, 35(4):A1951–A1986, jan 2013.

[49] Tony Leliévre, Mathias Rousset, and Gabriel Stoltz. *Free Energy Computations: A Mathematical Perspective*. Imperial College Press, 2010.

[50] Jacques-Louis Lions, Yvon Maday, and Gabriel Turinici. A parareal in time disctratization of PDEs. *C. R. Acad. Sci. Paris, Serie I*, 332:661–668, apr 2001.

[51] Pu Liu and Gregory A. Voth. Smart resolution replica exchange: An efficient algorithm for exploring complex energy landscapes. *J. Chem. Phys.*, 126:045106, 2007.

[52] Edward Lyman, F Marty Ytreberg, and Daniel M Zuckerman. Resolution Exchange Simulation. *Phys. Rev. Lett.*, 96:028105, 2006.

[53] Edward Lyman and Daniel M. Zuckerman. Resolution Exchange Simulation with Incremental Coarsening. *J. Chem. Theory Comput.*, 2(3):656–666, 2006.

[54] A. D. MacKerell, D. Bashford, M. Bellott, R. L. Dunbrack, J. D. Evanseck, M. J. Field, S. Fischer, J. Gao, H. Guo, S. Ha, D. Joseph-McCarthy, L. Kuchnir, K. Kuczera, F. T. K. Lau, C. Mattos, S. Michnick, T. Ngo, D. T. Nguyen, B. Prodhom, W. E. Reiher,

B. Roux, M. Schlenkrich, J. C. Smith, R. Stote, J. Straub, M. Watanabe, J. Wiórkiewicz-Kuczera, D. Yin, and M. Karplus. All-Atom Empirical Potential for Molecular Modeling and Dynamics Studies of Proteins. *J. Phys. Chem. B*, 102(18):3586–3616, apr 1998.

[55] Paul Maragakis and Martin Karplus. Large Amplitude Conformational Change in Proteins Explored with a Plastic Network Model: Adenylate Kinase. *J. Mol. Biol.*, 352(4):807–822, 2005.

[56] Luca Maragliano, Alexander Fischer, Eric Vanden-Eijnden, and Giovanni Ciccotti. String method in collective variables: Minimum free energy paths and isocommittor surfaces. *J. Chem. Phys.*, 125:024106, 2006.

[57] Luca Maragliano and Eric Vanden-Eijnden. A temperature accelerated method for sampling free energy and determining reaction pathways in rare events simulations. *Chem. Phys. Lett.*, 426(1-3):168–175, 2006.

[58] Luca Maragliano and Eric Vanden-Eijnden. On-the-fly string method for minimum free energy paths calculation. *Chem. Phys. Lett.*, 446(1-3):182–190, 2007.

[59] Thomas F Miller III, Eric Vanden-Eijnden, and David Chandler. Solvent coarse-graining and the string method applied to the hydrophobic collapse of a hydrated chain. *Proc. Natl. Acad. Sci. USA*, 104(37):14559–14564, sep 2007.

[60] Klaus Müller and Leo D. Brown. Location of saddle points and minimum energy paths by a constrained simplex optimization procedure. *Theor. Chim. Acta*, 53(1):75–93, 1979.

[61] Radford M Neal. Annealed Importance Sampling. *Stat. Comput.*, 11(2):125–139, 2001.

[62] Jerzy Neyman. On the Two Different Aspects of the Representative Method: The Method of Stratified Sampling and the Method of Purposive Selection. *Journal of the Royal Statistical Society*, 97:558–625, 1934.

[63] Jorge Nocedal and Stephen J Wright. *Numerical Optimization*. New York, 2nd edition, 2006.

[64] Harald Oberhofer, Christoph Dellago, and Phillip L Geissler. Biased Sampling of Nonequilibrium Trajectorties: Can Fast Switching Simulations Outperform Conventional Free Energy Calculation Methods? *J. Phys. Chem. B*, 109:6902–6915, 2005.

[65] Toshiro Oda, Mitsusada Iwasa, Tomoki Aihara, Yuichiro Maéda, and Akihiro Narita. The nature of the globular- to fibrous-actin transition. *Nature*, 457(7228):441–445, 2009.

[66] Alexey Onufriev, Donald Bashford, and David A. Case. Modification of the Generalized Born Model Suitable for Macromolecules. *J. Phys. Chem. B*, 104(15):3712–3720, 2000.

[67] Victor Ovchinnikov, Martin Karplus, and Eric Vanden-Eijnden. Free energy of conformational transition paths in biomolecules: The string method and its application to myosin VI. *J. Chem. Phys.*, 134:085103, 2011.

[68] Albert C Pan, Deniz Sezer, and Benoît Roux. Finding Transition Pathways Using the String Method with Swarms of Trajectories. *J. Phys. Chem. B*, 112(11):3432–3440, mar 2008.

[69] C Pangali, M Rao, and B J Berne. A Monte Carlo simulation of the hydrophobic interaction. *J. Chem. Phys.*, 71:2975–2981, 1979.

[70] Elias Alphonsus Jozef Franciscus Peters and Gijsbertus De With. Efficient sampling of a dual-resolution ensemble by means of dragging. *J. Chem. Theory Comput.*, 7(9):2699–2709, 2011.

[71] James C. Phillips, Rosemary Braun, Wei Wang, James Gumbart, Emad Tajkhorshid, Elizabeth Villa, Christophe Chipot, Robert D. Skeel, Laxmikant Kalé, and Klaus Schulten. Scalable molecular dynamics with NAMD, 2005.

[72] Steve Plimpton. Fast Parallel Algorithms for Short-Range Molecular Dynamics. *J. Comp. Phys.*, 117(1):1–19, mar 1995.

[73] Jean Paul Ryckaert, Giovanni Ciccotti, and Herman J.C. Berendsen. Numerical integration of the cartesian equations of motion of a system with constraints: molecular dynamics of n-alkanes. *J. Comp. Phys.*, 23(3):327–341, 1977.

[74] Marissa G. Saunders and Gregory A. Voth. Water molecules in the nucleotide binding cleft of actin: Effects on subunit conformation and implications for ATP hydrolysis. *J. Mol. Biol.*, 413(1):279–291, 2011.

[75] Marissa G. Saunders and Gregory A. Voth. Comparison between actin filament models: Coarse-graining reveals essential differences. *Structure*, 20(4):641–653, 2012.

[76] J. Schlitter, M. Engels, P. Krüger, E. Jacoby, and A. Wollmer. Targeted Molecular Dynamics Simulation of Conformational Change-Application to the T R Transition in Insulin. *Mol. Simulat.*, 10(2-6):291–308, 1993.

[77] T Schneider and E Stoll. Molecular-dynamics study of a three-dimensional one-component model for distortive phase transitions. *Phys. Rev. B*, 17:1302, 1978.

[78] C Schütte, F Noé, J Lu, M Sarich, and E Vanden-Eijnden. Markov state models based on milestoning. *J. Chem. Phys.*, 134:204105, 2011.

[79] Paul E. Smith. The alanine dipeptide free energy surface in solution. *J. Chem. Phys.*, 111:5568, 1999.

[80] Yuji Sugita and Yuko Okamoto. Replica-exchange molecular dynamics method for protein folding. *Chem. Phys. Lett.*, 314(1-2):141–151, nov 1999.

[81] Robert H. Swendsen and Jian Sheng Wang. Replica Monte Carlo Simulation of Spin-Glasses. *Phys. Rev. Lett.*, 57:2607, 1986.

[82] Erik H. Thiede, Brian Van Koten, Jonathan Weare, and Aaron R. Dinner. Eigenvector method for umbrella sampling enables error analysis. *J. Chem. Phys.*, 145(8):084115, 2016.

[83] G M Torrie and J P Valleau. Nonphysical Sampling Distributions in Monte Carlo Free-Energy Estimation: Umbrella Sampling. *J. Comp. Phys.*, 23:187–199, 1977.

[84] Gareth A. Tribello, Massimiliano Bonomi, Davide Branduardi, Carlo Camilloni, and Giovanni Bussi. PLUMED 2: New feathers for an old bird. *Comput. Phys. Commun.*, 185(2):604–613, feb 2014.

[85] Suriyanarayanan Vaikuntanathan and Christopher Jarzynski. Escorted free energy simulations. *J. Chem. Phys.*, 134:54107, 2011.

[86] Titus S van Erp, Daniele Moroni, and Peter G Bolhuis. A novel path sampling method for the calculation of rate constants. *J. Chem. Phys.*, 118:7762–7774, 2003.

[87] Eric Vanden-Eijnden and Maddalena Venturoli. Markovian milestoning with Voronoi tessellations. *J. Chem. Phys.*, 130:194101, 2009.

[88] Eric Vanden-Eijnden and Maddalena Venturoli. Revisiting the finite temperature string method for the calculation of reaction tubes and free energies. *J. Chem. Phys.*, 130:194103, may 2009.

[89] Aryeh Warmflash, Prabhakar Bhimalapuram, and Aaron R Dinner. Umbrella sampling for nonequilibrium processes. *J. Chem. Phys*, 127:154112, oct 2007.

[90] Jonathan Weare. Efficient Monte Carlo sampling by parallel marginalization. *Proc. Natl. Acad. Sci. USA*, 104(31):12657–12662, 2007.

[91] Yu Yamamori and Akio Kitao. MuSTAR MD: Multi-scale sampling using temperature accelerated and replica exchange molecular dynamics. *J. Chem. Phys.*, 139(14):145105, 2013.

[92] F Marty Ytreberg and Daniel M Zuckerman. Single-ensemble nonequilibrium path-sampling estimates of free energy differences. *J. Chem. Phys.*, 120:10876, 2004.

[93] Matthew C. Zwier, Joshua Adelman, Joseph W. Kaus, Adam J. Pratt, Kim F. Wong, Nicholas B. Rego, Ernesto Suárez, Steven Lettieri, David W. Wang, Michael Grabe, Daniel M Zuckerman, and Lillian T. Chong. WESTPA: An interoperable, highly scalable software package for weighted ensemble simulation and analysis. *J. Chem. Theory Comput.*, page 150113180903008, jan 2015.