Supporting information for: Two Relations to Estimate Membrane Permeability Using Milestoning

Lane W. Votapka,[†,‡] Christopher T. Lee,[†,‡] and Rommie E. Amaro[*,†]

†*Department of Chemistry and Biochemistry, University of California, San Diego, 9500 Gilman Drive, La Jolla, California 92093-0340*

‡*Contributed equally to this work*

E-mail: ramaro@ucsd.edu

Langevin Dynamics Engine

We have developed a software package which performs Langevin dynamics calculations on a user specified PMF and diffusivity profile. The engine is implemented using the Python and C programming languages. To run a simulation the user specifies a PMF and viscosity profile as well as the hydrodynamic radius and mass.

The PMF profile is generated using a piecewise cubic Hermite polynomial (PCHIP) from the following user-specified values: `dz`, the distance from the center to the edge of the membrane; `W1`, `W2`, and `W3`, the PMF at the interface, between the interface/core, and at the core of the membrane respectively; as well as `a` and `b` which correspond to the location of the water bilayer interface and interface/core. We interpolate the points using `scipy.interpolate`. Conveniently the PCHIP algorithm allows for both fast computation of the value at any point. Furthermore we can precalculate the first derivative, the negative value of which is the force, as another spline for fast evaluation. The viscosity profile is computed in a similar fashion though requiring additional inputs: `dz`, the distance from the center to edge of the membrane; `d1`, `d2`, `d3`, and `d4`, the viscocities in bulk, the interface, interface/core, and core respectively; and `a`, `b`, `c` which specify the location of the interface, interface/core, and core respectively.

The Grønbach-Jensen–Farago integrator is implemented in both Python and C. To generate Gaussian random number in C, we employ the use of the PCG random number generator (RNG)[S1] to sample random integers. These random integers are converted to random uniform floating point numbers by using a bit stream algorithm by Taylor R. Campbell. Uniform floats are then transformed using the Box-Muller algorithm to yield a gaussian random number. The qualities of our C random number generator are compared to that of python library `numpy` in fig. S1. The numpy and C RNGs are seeded off entropy from `/dev/urandom`. The C integrator is wrapped using the Python/C API allowing it to be called from Python which handles the system setup, configuration, and analysis.

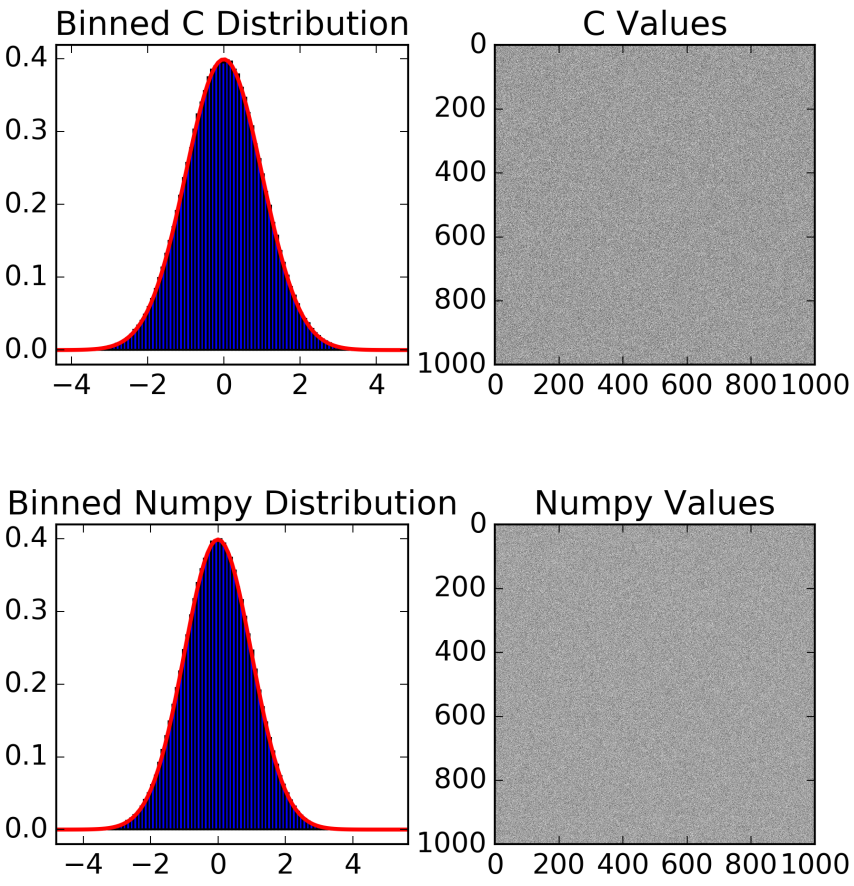

Figure S1: The binned probability distribution of both C and `numpy` generated gaussian random numbers are shown in comparison with an ideal Gaussian with zero mean and unit variance. On the right, a greyscale plot of the sequence of random numbers indicates, by visual inspection, that they are not correlated and there are no repeat sequences.

Milestoning Sampling and Efficiency

The total number of statistics aggregated during milestoning for each system is shown in fig. S2. Notably, the statistics consist of only the trajectories which were members of the first hitting point distribution (FHPD). To quantify the efficiency of identifying FHPD members we ran a quick test of the accept rate of the milestone at $z = 0$ with adjacent milestones at $z = -1, 1$ for 10000 random members with a timestep of 1×10^{-15} s. We find that the typical rejection rate is upwards of ≈90% (9000) during the reverse phase. While most rejections come about from self-crossing events occurring in the first several timesteps, and thus contribute little to the total computer time, the number of trajectories required to generate a well-explored FHPD is very large, making the compute time of rejections non-trivial. None of the reported statistics account for the number of rejected trajectories.

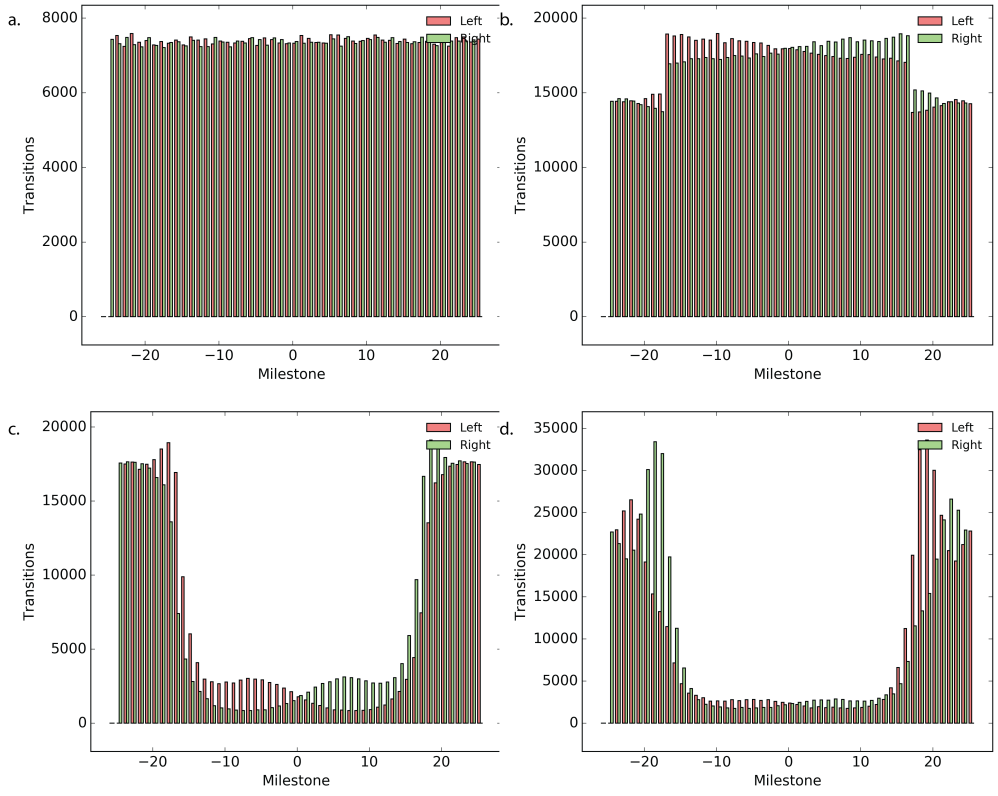

Figure S2: Shown here are the total number of forward and reverse trajectories collected for systems (a) flat, (b) small-barrier, (c) urea, and (d) codeine.

Sampling Convergence of Crossing Probabilities

To quantify the impact of sampling on our confidence in crossing probability, we compute the 95% confidence of $\rho = 0.5$ with respect to a total number of observations. Confidence intervals are estimated using the percentile boostrap over 10,000 resamples. The results are shown in fig. S3. Even at $N = 200$ the confidence interval is still quite large at 0.07. The calculation of ρ is a binomial sampling problem, thus the 95% confidence interval should follow the following solution,

$$CI(95) = 2\rho/\sqrt{N-1}. \tag{1}$$

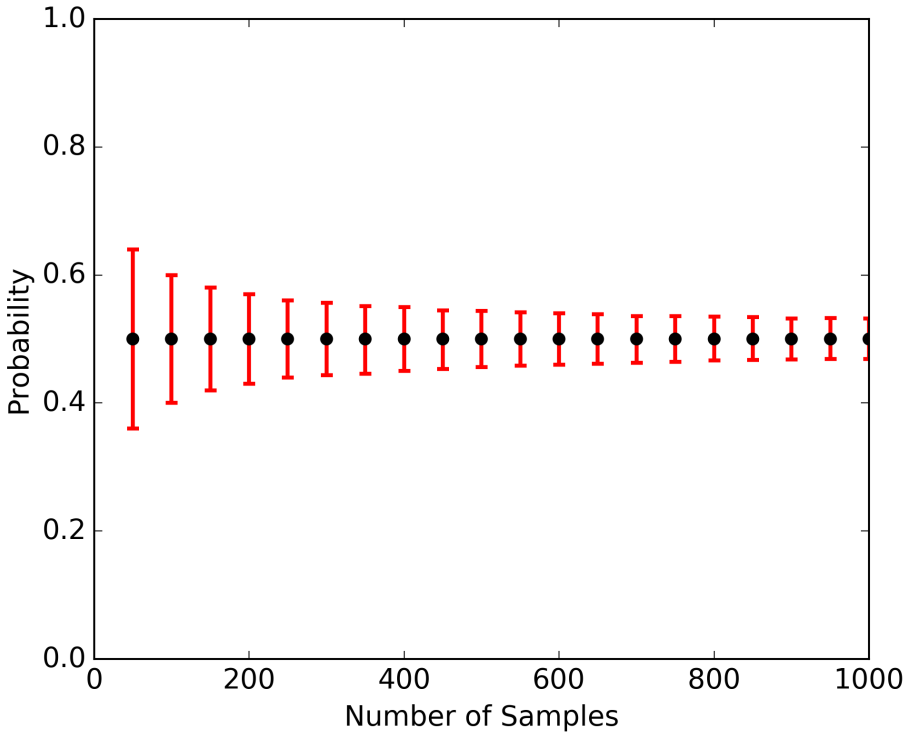

Figure S3: The 95% confidence intervals of a 50-50 crossing probability with respect to the total number of observations.

References

(S1) O'Neill, M. E. PCG: A Family of Simple Fast Space-Efficient Statistically Good Algorithms for Random Number Generation. *ACM Trans. Math. Softw.. In Review.*