Effect of Integrator Choice on the Simulation of Molecular Dynamics of Alanine Dipeptide

Nimit Sohoni *(nims)* and Debnil Sur *(debnil)*

***Abstract*—Molecular dynamics (MD) simulation requires numerical approaches to integrate a many-body system over a force field and calculate velocities, accelerations, and atomic trajectories. The choice of numerical algorithm thus affects both the correctness and speed of the simulation. We test three significant integrators – the Velocity Verlet, Langevin, and Beeman's integrators – on alanine dipeptide, a simple molecular system. All three methods were found to have similar correctness and speed on a small molecule. This provides an interesting benchmark to test the scalability of these methods on larger systems. However, while all were successful at sampling lower-energy states, none adequately sampled from less probable landscapes–signaling a possible area of further research.**

***Keywords*—Molecular dynamics, MD simulation, integration**

I. Introduction

Biomolecular dynamics exist over a wide range of both time and space, and methods to study them depend heavily on the questions asked. Experimental techniques are often the best choice; for instance, spectroscopy for bond vibrations and electrophysiology for ion channel activity. Due to theoretical advances over recent decades, modeling and simulation can be more detailed and/or efficient than setting up a new experiment.

Variations exist within these theoretical approaches as well. For instance, prediction of protein structure and/or function is often best approached through bioinformatic algorithmic analysis that detects related proteins through similarity in amino acid. Similarly, computational drug design has made advances through sacrificing some accuracy via exceptionally fast statistical methods. Traditionally, simulations have tested simple theoretical models' ability to predict experimental observations. Today, they are increasingly being used to predict properties like binding or folding that are later experimentally confirmed. Continuing advances in computational power will only accelerate this trend in coming years.

Specifically, understanding mechanisms of protein folding has challenged many working in protein chemistry for decades. Advances in this area can impact many fields, such as protein structure prediction and, in turn, drug discovery. Despite both experimental and theoretical advances, generating detailed descriptions of folding process and mechanism at an atomic level has been difficult. Molecular dynamics (MD) simulation with full atomic representation of the protein and implicit representation of the solvent provides atomic level resolution and accuracy.

These MD simulations must model various physical and chemical processes. This requires integrating the equations of a many-body system of interacting particles. Many numerical algorithms have been devised and implemented over the years for this purpose. Early investigations applied the traditional high-order explicit Runge-Kutta (RK) and implicit predictor-corrector (PC) schemes. However, on MD scales of time, the extra orders obtained in these schemes accumulated too rapidly. This instability restricts the use of these conventional integrators to very small time steps and thus significantly reduces computational efficiency. They also produce solutions which are neither symplectic nor time reversible, unlike exact phase trajectories [1].

Thus, subsequent work at the intersection of scientific computing and molecular dynamics has focused on improving the stability of integration algorithms, particularly through the preservation of symplecticity and reversibility. Increased stability bounds numerical errors, even for large time steps, and thus improves long-duration evaluations in MD. In this work, we compare the performance of the most common numerical methods through a number of criteria, chiefly rate of convergence and accuracy. Through this comparison, we can evaluate the observed performance of a common, open-source implementation of these techniques and compare this with theoretical performance. The difference between expectation and observation reveals continued necessary progress for scientific computing research.

As a case study, we will use alanine dipeptide. This molecular system serves as an excellent benchmark for many different computational methods because of its moderate size and complexity [2]. Computational work has used various levels of methods, from empirical potentials and classical MD simulations to ab initio dynamics in implicit solvent. It has also been extensively studied experimentally [3]. Assuming the rigidity of peptide bonds and neglecting rotation of the methyl groups, all conformers can be fully characterized through two independent dihedrals. This lends itself particularly well to our metrics.

II. Theory

A. Introduction

MD simulation is based on Newton's second law, $F = ma$. From the knowledge of the force on each atom, one can determine the acceleration of each atom in a system. Integration of the equations of motion then yield a trajectory describing the positions, velocities, and accelerations of particles as they vary

N. Sohoni is with the Institute of Computational and Mathematical Engineering at Stanford University. D. Sur is with the Department of Computer Science at Stanford.

Manuscript completed December 9, 2016.

with time. The trajectory can be used to find the average values of properties. This method is deterministic; once positions and velocities are known, the state of the system can be predicted at any point in the future or past. These simulations are expensive in time, cost, and computation, so simulations of solvated proteins are typically calculated up to the nanosecond time scale.

To see the importance of integration techniques, we will first consider the simple case of $F = ma$ where acceleration $a = \frac{dv}{dt}$ is constant. After integration, we find $v = at + v_0$ and, in turn, $x = vt + x_0$. This gives the following relationship $x(t) = \frac{at^2}{2} + v_0 t + x_0$. Acceleration is given as the derivative of the potential energy with respect to position, r, $a = -\frac{1}{m}\frac{dE}{dr}$. Thus, to calculate trajectory, we only need the initial positions of the atoms; an initial distribution of velocities; and the acceleration, determined by the gradient of the potential energy function. Initial positions can be determined from experimental structures, such as x-ray crystal structure of the protein or a solution structure determined by NMR spectroscopy.

The initial distribution of velocities are usually determined from a random distribution with correction so there is no overall momentum, i.e., $P = \sum_{i=1}^{N} m_i v_i = 0$. Velocities are often chosen randomly from a Maxwell-Boltzmann or Gaussian distribution at a given temperature. This gives the probability that an atom i has velocity v_x in the x direction at a temperature T: $p(v_{ix} = (\frac{m_i}{2\pi k_B T})^{1/2} \exp(-\frac{1}{2}\frac{m_i v_{ix}^2}{k_B T})$ The temperature can be calculated from velocities via $T = \frac{1}{3N}\sum_{i=1}^{N} \frac{|p_i|}{2m_i}$, where N is the number of atoms in the system.

The potential energy is thus a function of the atomic positions $(3N)$ of all atoms in the system. This function's complicated nature means that there is no analytical solution to the equations. Therefore, solving these equations requires a numerical approach. Experimentation with these techniques (in general theory, other applications, and MD simulations) has been an enormous area of study. Now, we will delve into the most popular techniques.

B. *Integration Techniques*

The most commonly used time integration algorithm is the Verlet algorithm [4]. This writes two third-order Taylor expansions for the positions $r(t)$, one forward and one backward in time. Let v be the velocity, a the acceleration, and b the third derivatives of r with respect to t. We then have:

$r(t+\Delta t) = r(t) + v(t)\Delta t + (1/2)a(t)\Delta t^2 + (1/6)b(t)\Delta t^3 + O(\Delta t^4)$ and $r(t + \Delta t) = r(t) - v(t)\Delta t + (1/2)a(t)\Delta t^2 - (1/6)b(t)\Delta t^3 + O(\Delta t^4)$.

Combining these gives $r(t + \Delta t) = 2r(t) - r(t - \Delta t) + a(t)\Delta t^2 + O(\delta t^4)$, the basic form of the Verlet algorithm. Since we are integrating Newton's equations, $a(t)$ is just force over mass. Force, in turn, is a function of the positions $r(t)$: $a(t) = -(1/m)\nabla V(r(t))$.

One can see that the truncation error of the algorithm when evolving the system by Δt is order Δt^4, even if the third derivatives do not appear explicitly. The algorithm is also simple to implement, accurate, and stable. These characteristics have made it popular in MD simulation.

A problem with this version of the Verlet algorithm is that velocities are not directly generated. While not necessary in time evolution, this knowledge can be necessary. They are required to compute the kinetic energy K, which is needed to test conservation of total energy $E = K + V$. To overcome this, variants of the Verlet algorithm have been develop. They generate the same trajectory and differ in what variables are stored in memory and when.

One popular improvement is the Verlet Leapfrog Algorithm [4], which is written as: $r(t + \Delta) = r(t) + v(t + \Delta/2)\Delta$ and $v(t+\Delta/2) = v(t-\Delta/2) + a(t)\Delta t$. Here, the velocities are first calculated at time $t+\Delta/2$, then used to calculate the position r at time $t+\Delta$. Thus, the velocities leap over the positions, then the positions leap over the velocities. This explicitly calculates the velocities but does not calculate them at the same time as position. The velocity at time t, $v(t)$, can be approximated as $v(t) = \frac{1}{2}[v(t - \Delta/2) + v(t + \Delta/2)]$.

A better implementation is the velocity Verlet scheme [4], where positions and velocity at time $t + \Delta$ are derived as follows.

$r(t + \Delta) = r(t) + v(t)\Delta + (1/2)a(t)\Delta^2$

$v(t + \Delta t) = v(t) + \frac{1}{2}[a(t) + a(t + \Delta)]\Delta$

This requires $9N$ memory locations to save the $3N$ positions, velocities, and accelerations, but we never need to simultaneously store the values at two different times for any single quantity. There is no compromise on precision.

Alternate solution methods also exist for this classical mechanics formulation of a force field. A popular one is Beeman's algorithm [4], which is closely related to the Verlet algorithm and has the following formulations for position and velocity.

$r(t + \Delta) = r(t) + v(t)\Delta + \frac{2}{3}a(t)\Delta^2 - \frac{1}{6}a(t - \Delta)\Delta^2$

$v(t+\Delta) = v(t) + v(t)\Delta + \frac{1}{3}a(t)\Delta + \frac{5}{6}a(t)\Delta - \frac{1}{6}a(t-\Delta)\Delta$

While the simplification to Newtonian mechanics eases calculation of force fields, it may not always be the most realistic approach to molecular dynamics simulation. As a result, further work in simulation explored the addition of different forces to classical equations to better account for different effects. Langevin dynamics utilizes one such theoretical advance to attempt to more accuately model molecular dynamics.

The Langevin equation is a stochastic differential equation in which two force terms have been added to Newton's second law to approximate the effects of neglected degrees of freedom [4]. One term represents friction, the other a random force R. Since friction opposes motion, the first additional force is proportional to the particle's velocity and oppositionally directed. For example, the effects of solvent molecules not explicitly present in the simulated system would be approximated via a frictional drag on the solute and random kicks associated with thermal motions. It is also of note that the Langevin method is thermostatted. It thus approximates the canonical ensemble, which represents the possible states of a mechanical system in thermal equilibrium with a fixed temperature heat bath.

Then, Langevin's equation for the motion of atom i is $F_i - \gamma_i v_i + R_i(t) = m_i a_i$, where F_i is still the sum of all forces exerted on atom i by all other atoms. The dynamics of a simulation can be modeled accordingly. We omit the exact solution of these equations for purposes of space, but closed form expressions of positions and velocities can be derived and computed through numerical methods.

C. Criteria

In testing and comparing optimal integration techniques, we must define and parameterize our criteria. Broadly, we wish to rank these popular integration methods via speed and correctness.

For the speed, we will simply measure how long it takes our script to run in real time with each integrator.

To check correctness, for each integrator, we will create a Ramachandran plot of the protein by sampling the ϕ, ψ angles every n timesteps of the simulation, look at the generated histogram, and compare it to the theoretical probability distribution at 300 K. The theoretical probability distribution is the Boltzmann distribution generated from the ”theoretically calculated” free energy function on the Ramachandran plot, shown in Figure 1.

A Ramachandran plot visualizes energetically allowed regions for backbone dihedral angles ψ against ϕ of amino acid residues in a protein structure. This can be used to show, theoretically, which conformations of the angles are possible for an amino-acid residue in a protein. It can also show the empirical distribution of datapoints observed in a single structure and thus help validate it. As alanine dipeptide has only two dihedral angles, this method yields complete understanding of the distribution over its possible states.

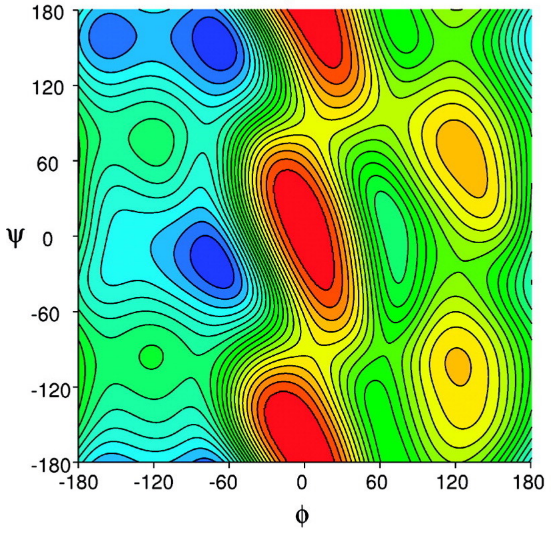

Fig. 1. Theoretical free energy landscape for this system, from Vymetal (2010) [2].

III. Experimental Design

Thus, in our experiment, we began with an alanine dipeptide in its native structure. We originally ”denatured” it first by running a simulation at high temperature. However, we decided that the dynamics of ’renaturation’ were not particularly interesting, and all algorithms seemed to take a roughly similar amount of time (though it varies between simulations) to get the alanine dipeptide back to a physically reasonable conformation. Thus, we simply took the initial condition to be alanine dipeptide in its native state. The randomness introduced by MD integration methods (at least, as commonly implemented), coupled with numerical errors, ensures that separate simulations will not be overtly correlated, so we do not really need to vary the initial conditions between simulations with the same integrator.

We then set up a simulation using the open-source Python package OpenMM [6]. As hyperparameters, we used a step size of 1 femtosecond, temperature of 300 K, and 1 million timesteps (to simulate 1 ns total). We also used the Amber99SB-ILDN forcefield to generate the dynamics [5]. Finally, as an implicit solvent, we used the water model OBC [8]. The alanine dipeptide structure is available as part of the default OpenMM package. We also used the MDTraj package to help with generating some of the plots [9].

We ran three 1-nanosecond simulations each using the Velocity Verlet, Beeman’s, and Langevin integrators. The ”stock” OpenMM implementations of Verlet and Langevin were used, while we implemented Beeman’s ourselves using the OpenMM custom integrator implementation functionality. We captured the structures every 1000 timesteps (1 picosecond), generated their Ramachandran plots by sampling ϕ, ψ angles every 500 timesteps (0.5 picoseconds), and recorded the total computational time taken by the various simulations, as well as measuring some other data about the simulations. The results are below.

The theoretical free energy landscape for alanine dipeptide at 300 K as a function of ϕ and ψ has been calculated using Molecular Dynamics by Vymetal and Vondrasek [2], using several different force fields and parameters. Unfortunately, they only provided a plot for the free energy landscape using the Amber FF03 force field with the TIP3P explicit solvent model. For computational ease and efficiency, we used an implicit solvent model, and we chose to the Amber FF99SB-ILDN force field rather than FF03 because it is known to agree more accurately with NMR data in many cases.

IV. Results

A. Correctness

Let us begin with our first criterion, correctness.

As described before, the theoretical probability distribution was discretized, and the empirical probability distribution was also discretized and then subtracted from this (to easily calculate the difference of the two distributions). Then, the sum of the squares of the discretized difference was taken. We will refer to this quantity as Q from now on. It can be thought of as the deviation of the distribution of the samples from the true probability distribution. Of course, the lower the value of Q, the better the agreement of the samples and the true distribution.

For the Langevin integrator, the three values of Q obtained were $382.1, 393.1, 370.7$.

For the Verlet integrator, we obtained $Q = 349.5, 356.6,$ and 363.1.

Finally, for the Beeman integrator, the values of Q were $412.7, 351.9, 375.4$.

Figs. 2-4 show the Ramachandran plots generated by the simulations. (ϕ, ψ) angles were sampled every 500 time steps (0.5 picoseconds), for a total of 2,000 samples, which are then used to generate the Ramachandran plots. (The Ramachandran plots were only generated for one representative run of each simulation. They are not appreciably qualitatively different between simulations.)

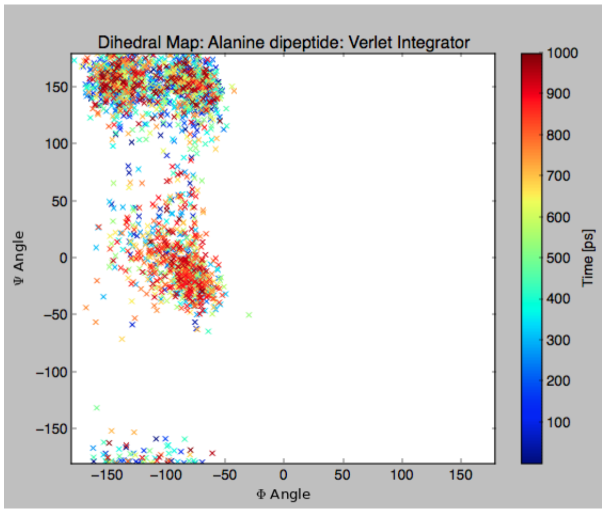

Fig. 2: Ramachandron plot for Verlet integrator.

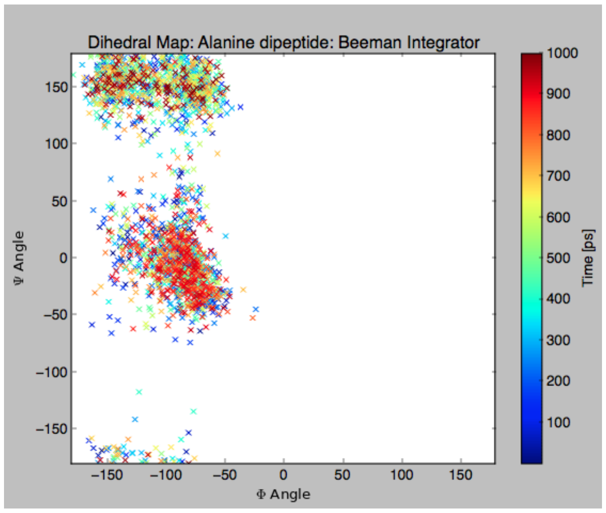

Fig. 3: Ramachandran plot for Beeman's integrator.

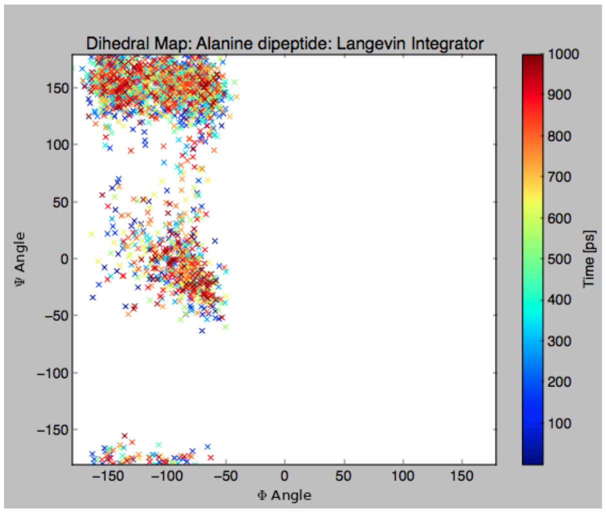

Fig. 4: Ramachandran plot for Langevin integrator.

Due to limited computational resources, we were not able to complete more runs. Still, it appears that the Verlet integrator performs the best of the three, although not by a large amount. This is somewhat surprising, since Beeman's algorithm is a third-order method as opposed to Velocity Verlet, a second-order method. However, the difference is not necessarily paradoxical: it is possible that, while Beeman's method is more numerically accurate at each time step, the numerical roundoff errors introduced during each step of Velocity Verlet actually fortuitously ”conspire” in some way to do a better job of moving around the free energy landscape; in other words, that the simulation is biased, and the numerical errors actually serve to counteract this bias somewhat by introducing more variation.

Looking at the Ramachandran plots, it does seem that there is a clear bias towards low-free-energy regions, even more so than that would be expected from the theoretical free energy landscape. It is possible that the simulation time we are using is too short to adequately sample this probability distribution, but it is still telling that not one sample ever ”crossed over,” for instance, to a structure near one of the local energy minima with a positive ϕ angle.

Figs. 5-7 show the plot of total energy over time for each method. Total energy was output every 1000 time steps using OpenMM's built-in functionality. Note that we do not necessarily expect the Langevin integrator to conserve energy, since it is a thermostatted method. The Beeman's and Verlet methods should, however. However, it is also not unexpected that the energy plots fluctuate wildly, as this is a standard characteristic of most kinds of MD simulations; as long as the average energy over a large enough time window does not change as time increases, energy is ”effectively” conserved. In fact, all three integrators seem to conserve energy fairly well, by this definition, which is a desirable property in general for molecular dynamics.

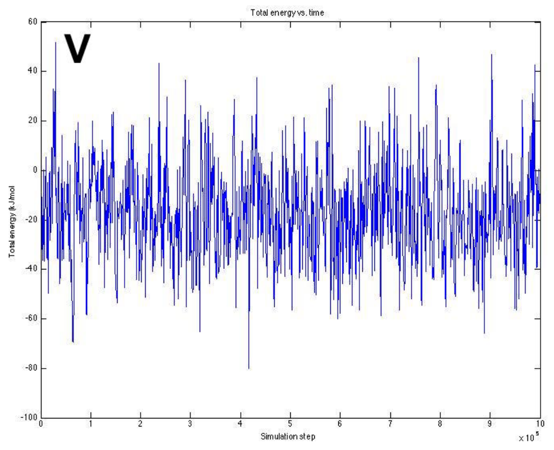

Fig. 5: Energy over time for Verlet integrator.

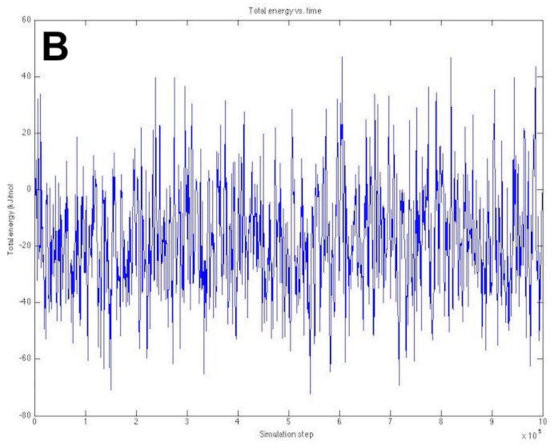

Fig. 6: Energy over time for Beeman's integrator.

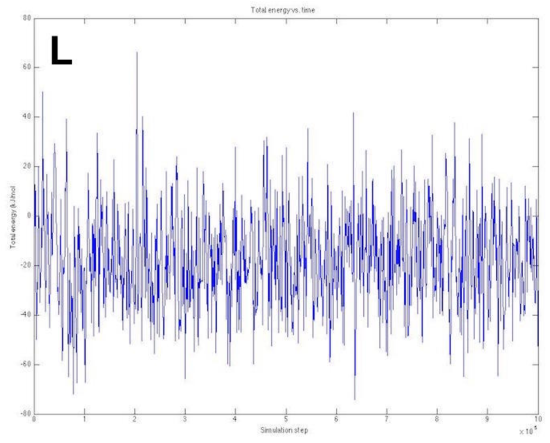

Fig. 7: Energy over time for Langevin integrator.

B. Speed

Now, we will consider our second criterion, computational speed. Since all simulations were run for the same number of time steps and with the same step size, we can directly compare the total amount of time taken by each one.

The Langevin simulations took $203.6, 204.1$, and 203.3 seconds, respectively. The Verlet simulations took 200.3, 204.2, and 204.6 seconds. Finally, the Beeman's simulations took $201.1, 202.9$, and 204.2 seconds.

So, we can see that the time taken for each method is relatively similar.

It is interesting that Beeman's algorithm did not seem to take a materially longer time than either of the others, despite its more expensive computations. It is possible that the size of our system is so small that the increased cost owing to its slightly more complicated functional form is negligible compared to the general overhead of the simulations.

C. Analysis

Figs. 8-10 shows the plot of the RMS from the starting structure versus time step for one run of each simulation. This allows us, in a crude manner, to see the timescales on which the molecule moves around on the free energy landscape. We can clearly see jumps in the RMS at a few different times. Our "native" structure starts in the top region. This corresponds to moving between the top cluster of points and the middle cluster of points (also with occasional movements to the bottom cluster of points, with ψ near -180 and ϕ). These are relatively large structural changes because the dihedral angle ψ changes by a fairly large amount, so it makes sense that the RMS distance would change by a lot too. Note that the jumps occur quite infrequently in the simulations. This is because the energy barrier between the two states is high enough to significantly slow interconversion between the states at 300 K.

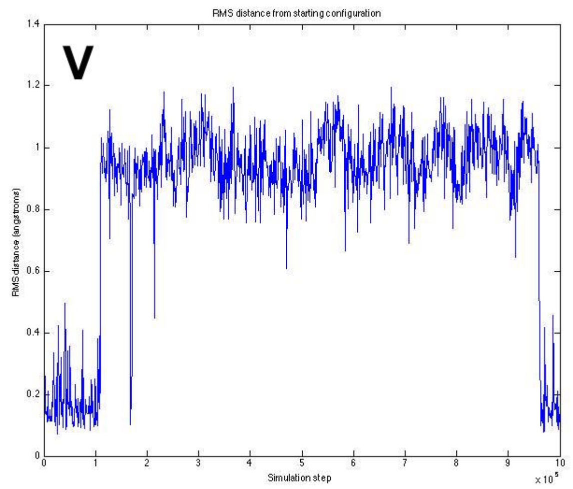

Fig. 8: RMS distance from native structure over time for Verlet integrator.

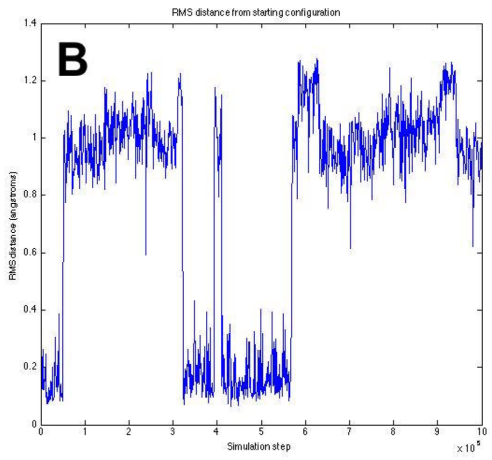

Fig. 9: RMS distance from native structure over time for Beeman's integrator.

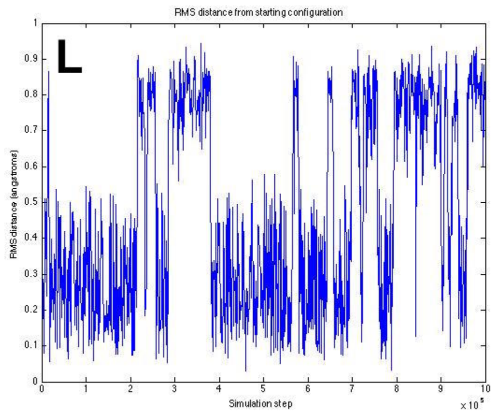

Fig. 10: RMS distance from native structure over time for Langevin integrator.

D. Discussion and Conclusions

V. Future Work and Acknowledgments

Due to the central nature of our experiment in MD simulations, future work can be taken in a number of different directions. We will outline the most prominent options here. First, while we tested the most commonly used techniques, recent optimizations have been made to their performance that have not yet been implemented in many open-source packages. Implementing and testing higher-order versions of these procedures could demonstrate whether or not they make a tangible difference in simulation. Second, we used a small, easily foldable molecule due to constraints on computational power. Running these experiments on a larger biomolecule can help test whether these results scale and apply to larger systems. Third, we can also try different metrics. The broader criteria of speed and correctness are likely the best ways to test integration performance. But correctness, in particular, could be tested through different, more quantitative schemes. For example, we could model the outputs via a Boltzmann distribution and using some form of likelihood testing to find the probability of our conformation.

VI. Conclusion

Advances in hardware and software have made molecular dynamics simulations an essential part of the toolbox of modern computational biology. Integration techniques are necessary to efficiently compute the effects of the force fields that underly these systems. In this paper, we tested the relative performance of three common integration schemes–Velocity Verlet, Beeman's, and Langevin–in simulating the dynamics of alanine dipeptide. Our results can be understood through correctness (via Ramachandran plot) and speed (through computational time). Interestingly, we discovered that the Velocity Verlet scheme generated slightly more ”correct” output. Though this can potentially be attributed to numerical errors, it still represents a non-intuitive result that should be explored with greater computational power and on larger biomolecules. In terms of energy, all three methods fluctuated quite a bit but generally conserved energy, a desirable aspect in simulation. We also found that all three systems have generally the same speed. Though somewhat surprising as well, this can be chalked up to the relatively smaller size of the simulated molecule. Testing whether or not this result scales could also be a fruitful avenue for future experiments. Finally, all these methods do a good job of sampling the landscape around low free-energy regions. However, on this short time-scale, none sample the lower probability regions well. This points to a potential avenue of improvement for further work in scientific computing: generating a method of integration that better samples the complete distribution of states. Such work can help create a more ”complete” and effective method of simulation.

Acknowledgments

The authors would like to thank Prof. Ron Dror for his guidance and intuition throughout the project, and the course in general. We would also like to thank Osama el-Gabalawy and Anthony Ma for their helpful assistance and intuition regarding MD simulation over the quarter.

References

[1] Omelyan, I. P., Mryglod, I. M., & Folk, R. (2002). New optimized algorithms for molecular dynamics simulations. Condens. Matter Phys, 5, 369.

[2] Vymetal, J.,& Vondrasek, J. (2010). Metadynamics As a Tool for Mapping the Conformational and Free-Energy Space of Peptides - The Alanine Dipeptide Case Study. The Journal of Physical Chemistry B, 114(16), 5632-5642.

[3] Parchansky, V., Kapitan J., Kaminsky J., Sebestik J., & Bour, P. (2013). Ramachandran Plot for Alanine Dipeptide as Determined from Raman Optical Activity. The Journal of Physical Chemistry Letters, 4(16), 27632768.

[4] Grubmller, H., Heller, H., Windemuth, A., & Schulten, K. (1991). Generalized Verlet algorithm for efficient molecular dynamics simulations with long-range interactions. Molecular Simulation, 6(1-3), 121-142.

[5] Larsen, K., et. al. (2010). Improved side-chain torsion potentials for the Amber ff99SB protein force field. Proteins, 78(8): 19501958.

[6] Eastman, P., Friedrichs, M. S., Chodera, J. D., Radmer, R. J., Bruns, C. M., Ku, J. P., ... & Tye, T. (2012). OpenMM 4: a reusable, extensible, hardware independent library for high performance molecular simulation. Journal of chemical theory and computation, 9(1), 461-469.

[7] Best, R. B., & Hummer, G. (2009). Optimized molecular dynamics force fields applied to the helix coil transition of polypeptides. The Journal of Physical Chemistry B, 113(26), 9004-9015.

[8] Bjelkmar, P., Larsson, P., Cuendet, M. A., Hess, B., & Lindahl, E. (2010). Implementation of the CHARMM force field in GROMACS: Analysis of protein stability effects from correction maps, virtual interaction sites, and water models. Journal of Chemical Theory and Computation, 6(2), 459-466.

[9] McGibbon, R., et. al. (2015). MDTraj: A Modern Open Library for the Analysis of Molecular Dynamics Trajectories. Biophysical Journal, 109(1), 1528-1532.