The Stability of Integrators in Molecular Dynamics Simulations

Gabriel Greenstein

December 23, 2023

1 Introduction:

Molecular dynamics (MD) simulations predict how every atom in a system will move over time based on a model describing the governing physics. By capturing near atomically-precise measurements at a fine temporal resolution ($\sim$ femtoseconds), MD simulations will be critical in discovering new drugs and probing the depths of molecular biology. Since MD simulations were first performed in the 1950s, new algorithms and computational hardware have led to significant improvements in both the speed and accuracy of these simulations [1]. In fact, Anton 3 – D.E. Shaw Research's latest super-computer – can model "100 microseconds a day out to simulation sizes larger than 1 million atoms and supports simulations beyond 50 million atoms" [2]. Most biologically meaningful events can take nanoseconds, microseconds, milliseconds, or even longer [3]. Simulations of this length require millions to trillions of time-steps. Therefore, it is essential that our numerical method is stable for otherwise the results of long time MD simulation are worthless.

In this paper, I aim to motivate and investigate the *Velocity Verlet* numerical algorithm. I will then code up the algorithm in Python using *OpenMM* and test it against Euler's method on the protein 2F4K.

2 Background:

First, let's briefly review the basic idea behind an MD simulation. We are given a vector $x \in \mathbb{R}^{3N}$, where N is the number of particles in our simulation. The reason x is a vector in dimension $3N$ is because we need to specify the (x, y, x) coordinates of each particle. We also need to remember the following facts from physics: (i) $F(x) = -\nabla U(x)$, (ii) $F = ma$, and $a = d(v)/dt = d^2(x)/dt^2$ where a, v, x, m are the respective acceleration, velocity, position, and mass of a given particle, $F(x)$ are the forces acting on x and $U(x)$ is the potential energy of x. For now, we leave our potential energy function to be arbitrary, but we will later use the AMBER molecular forcefield.

We can describe the motion of the particles in our system with the following equations

$$\begin{cases} \dfrac{d}{dt}\left(x(t)\right) = v(t) \\ \dfrac{d}{dt}\left(v(t)\right) = a(t) = \dfrac{F\left(x(t)\right)}{m} \end{cases}$$

Before delving deep into the mathematics, it is important to consider the properties that make a "good" numerical approximation. First, we want to reduce the numbers of times we have to compute $F(x)$ because $F(x) = -\nabla U(x)$ and $U(x)$ has terms of $\mathcal{O}(N^2)$. We also want to make sure our numerical method has good stability for large time steps. In other words, a numerical algorithm is stable if the solution doesn't "blow-up" depending on the initial position and velocity of our particles. We also want our algorithm to accurately describe our system. Finally, we ideally want our numerical approximation to conserve energy and momentum since we are modeling a physical system.

We will now construct the Velocity Verlet method with these considerations in mind and show where Euler's method fails.

2.1 Euler's Method:

We start by briefly deriving Euler's method. Recall that Taylor expanding a function f around $y = x$ is given by

$$f(y) = \sum_{n=0}^{\infty} \frac{f^{(n)}(x)}{n!} (y - x)^n$$

Letting $y - x = \Delta t$, we then have $y = x + \Delta t$ and thus

$$f(x + \Delta t) = \sum_{n=0}^{\infty} \frac{f^{(n)}(x)}{n!} (\Delta t)^n \tag{1}$$

Applying (1) to $x(t + \Delta t)$ and $v(t + \Delta t)$, we find that

$$\begin{cases} x(t + \Delta t) = x(t) + x'(t) \cdot \Delta t + \mathcal{O}(\Delta t^2) \\ v(t + \Delta t) = v(t) + v'(t) \cdot \Delta t + \mathcal{O}(\Delta t^2) \end{cases}$$

Using the fact that $x'(t) = v(t)$, $v'(t) = \dfrac{F(x(t))}{m}$, and dropping higher order terms, we have the desired approximation

$$\begin{cases} x(t + \Delta t) = x(t) + v(t) \cdot \Delta t \\ v(t + \Delta t) = v(t) + \dfrac{F(x(t))}{m} \cdot \Delta t \end{cases} \tag{2}$$

We now motivate the Velocity Verlet method by showing that (2) is unstable for long-time dynamics. Consider the following

$$\begin{cases} x(t + \Delta t) = x(t) + v(t) \cdot \Delta t + \dfrac{1}{2} \cdot \dfrac{F(x(t))}{m} \cdot (\Delta t)^2 + \mathcal{O}(\Delta t^3) \\ v(t + \Delta t) = v(t) + \dfrac{F(x(t))}{m} \cdot \Delta t + \mathcal{O}(\Delta t^2) \end{cases}$$

where we have expanded $x(t + \Delta t)$ by an additional term and used the fact that $x''(t) = a(t) = F(x(t))/m$. Suppose that we started at $t = t_0$ and we have moved one time step to $t_1 = t_0 + \Delta t$. Now we want to go back to t_0. By Euler's method, we would have

$$\begin{cases} x(t_1 - \Delta t) = x(t_1) + v(t_1) \cdot (-\Delta t) + \dfrac{1}{2} \cdot \dfrac{F(x(t_1))}{m} \cdot (-\Delta t)^2 \\ v(t_1 - \Delta t) = v(t_1) + \dfrac{F(x(t_1))}{m} \cdot (-\Delta t) \end{cases}$$

where we have neglected higher order terms. Since $t_1 = t_0 + \Delta t$, we have that $x(t_1) =$

$x(t_0) + v(t_0) \cdot \Delta t + \frac{1}{2} \cdot \frac{F(x(t_0))}{m} \cdot (\Delta t)^2$ and $v(t_1) = v(t_0) + \frac{F(x(t_0))}{m} \cdot \Delta t$ from the first iteration of Euler's method. Substituting in $x(t_1)$ and $v(t_1)$, and simplifying, we find that

$$\begin{cases} x(t_1 - \Delta t) = x(t_0) + \frac{1}{2m} \cdot \Big(F(x(t_1)) - F(x(t_0)) \Big) \cdot (\Delta t)^2 \\ v(t_1 - \Delta t) = v(t_0) + \frac{1}{m} \cdot \Big(F(x(t_0)) - F(x(t_1)) \Big) \cdot \Delta t \end{cases}$$

meaning that we can only return to the same place (i.e. $x(t_1 - \Delta t) = x(t_0)$ and $v(t_1 - \Delta t) = v(t_0)$) with Euler's method if we never moved anywhere in the first place (i.e., $\Delta t = 0$).

While the importance of "time-reversal" sounds trivial, it is imperative that our numerical algorithm has this property. This is because if it didn't, the energy of our system would be changing by small amounts after each iteration. Since MD simulations require millions to trillions of these time steps, this energy change compounds to an unstable solution that would result in an unrealistic structure.

2.2 Deriving Velocity Verlet [4, 5, 7]:

In order to construct a time-reversible numerical algorithm, we need to consider an arbitrary function over a phase space, i.e., $f(x(t), p(t))$, where $x(t), p(t)$ is the position and momentum of a particle at time t. We see that by the chain-rule

$$\frac{\partial f}{\partial t}(x(t), p(t)) = \frac{\partial f}{\partial x} \cdot \frac{dx}{dt} + \frac{\partial f}{\partial p} \cdot \frac{dp}{dt} \tag{3}$$

Swapping terms around and factoring out f in (3), we obtain

$$\frac{\partial f}{\partial t} = (\dot{x}(t) \cdot \partial_x + \dot{p}(t) \cdot \partial_p) f, \tag{4}$$

where $\dot{f}(t) = \frac{df}{dt}$ and $\partial_x = \frac{\partial}{\partial x}$. We can therefore think of $(\dot{x} \cdot \partial_x + \dot{p} \cdot \partial_p)$ as an operator acting on functions.

Define $iL = \dot{x} \cdot \partial_x + F(x) \cdot \partial_p$, where we have replaced $\dot{p}$ with $F(x)$ since $p = mv$ and thus $\dot{p} = F$. Note that L is a self-adjoint operator meaning that $L = L^\dagger$ (this is equivalent to saying $< Lf, g > = < f, Lg >$, where f, g are continuous functions and $< -, - >$ is the inner product) [4]. It is important to note that $\dagger$ acts on non-operators by performing a conjugate transpose. We will need this in later computations.

From (4), we have the first order ordinary differential equation

$$\frac{\partial f}{\partial t} = iLf,$$

which yields the solution

$$f(t) = e^{iLt} f(0) \tag{5}$$

Note that while e^{iLt} is an exponential, it is also an operator so normal exponential rules don't apply. Moreover, we have to think of a matrix exponential operator in its Taylor series expansion to understand how and what it is operating on.

In what follows, we split up our operator into $iL = iL_x + iL_p$. We now cite Trotter's Theorem: For non-commuting matrices A, B (i.e., $[A, B] = AB - BA \neq 0$),

$$e^{A+B} = \lim_{p\to\infty} \left(e^{A/2P}\, e^{B/P}\, e^{A/2P}\right)^P$$

For a moment, suppose $P = 2$. Then

$$\begin{aligned} e^{A+B} &= \left(e^{A/4}\, e^{B/2}\, e^{A/4}\right)\left(e^{A/4}\, e^{B/2}\, e^{A/4}\right) + \text{correction terms} \\ &= e^{A/4}\, e^{B/2}\, e^{A/2}\, e^{B/2}\, e^{A/4} + \text{correction terms} \end{aligned}$$

Thus we see for large P,

$$e^{A+B} = \left(e^{A/2P}\, e^{B/P}\, e^{A/2P}\right)^P + \mathcal{O}(P^{-2}) \tag{6}$$

Fix $t = T$ and let $A = iL_pT$ and $B = iL_xT$. Then for large P, we have

$$e^{iLT} = e^{iL_pT + iL_xT} \approx \left(e^{iL_pT/2P}\, e^{iL_xT/P}\, e^{iL_pT/2P}\right)^P$$

Letting $T/P = \Delta t$, we obtain

$$e^{iLT} \approx \left(e^{iL_p\frac{\Delta t}{2}}\, e^{iL_x\Delta t}\, e^{iL_p\frac{\Delta t}{2}}\right)^P$$

Substituting this into our solution (5), we have

$$f(t) \approx \left(e^{iL_p\frac{\Delta t}{2}}\, e^{iL_x\Delta t}\, e^{iL_p\frac{\Delta t}{2}}\right)^P f(0), \tag{7}$$

meaning that if we apply $\left(e^{iL_p\frac{\Delta t}{2}}\, e^{iL_x\Delta t}\, e^{iL_p\frac{\Delta t}{2}}\right)$ P times to $f(0)$, we evolve our system (roughly) by time T.

From (7) we can easily construct the Velocity Verlet numerical scheme. First, let's substitute $iL_x = \dot{x}\cdot\partial_x$ and $iL_p = F(x)\cdot\partial_p$ into $e^{(iL_x + iL_p)T}$, which yields

$$e^{iLT} \approx e^{F(x)\cdot\partial_p\cdot\frac{\Delta t}{2}}\, e^{\dot{x}\cdot\partial_x\cdot\Delta t}\, e^{F(x)\cdot\partial_p\cdot\frac{\Delta t}{2}}$$

Since derivatives are linear operators, we can rearrange constants in the exponents (think about expanding $e^{\partial_x\cdot c}$ in a Taylor series were c is some constant) so that we get

$$e^{iLT} \approx e^{\left(F(x)\cdot\frac{\Delta t}{2}\right)\cdot\partial_p}\, e^{(\dot{x}\cdot\Delta t)\partial_x}\, e^{\left(F(x)\cdot\frac{\Delta t}{2}\right)\partial_p}$$

In order to get the Verlet algorithm, we need to show the following: For some constant $c \in \mathbb{R}$ and variable x, $e^{c\cdot\partial_x}[f(x)] = f(x + c)$. This follows almost immediately from (1) replacing $\Delta t = c$. Moreover,

$$f(x+c) = \sum_{n=0}^{\infty} \frac{f^{(n)}(x)}{n!}c^n = \sum_{n=0}^{\infty} c^n\,\frac{(\partial_x)^n\,[f(x)]}{n!} = \left(\sum_{n=0}^{\infty}\frac{(c\cdot\partial_x)^n}{n!}\right)[f(x)]\,. \tag{8}$$

Therefore, we have our desired property $e^{c\cdot\partial_x}[f(x)] = f(x+c)$

This whole time we had our function f be arbitrary. We now set $f(t) = (x(t), p(t))^\intercal$ so that what we are evolving is the (position, momentum) vector of each particle. Then by

(8), we have

$$f(t) \approx \underbrace{e^{\left(F(x)\cdot\frac{\Delta t}{2}\right)\cdot\partial_p}\ \underbrace{e^{(\dot{x}\cdot\Delta t)\partial_x}\ \underbrace{e^{\left(F(x)\cdot\frac{\Delta t}{2}\right)\partial_p}\ f(0)}_{(1)}}_{(2)}}_{(3)}$$

where

(1) $p\left(\frac{\Delta t}{2}\right) = p(0) + \frac{\Delta t}{2}\cdot F(x(0))$

(2) $x(\Delta t) = x(0) + \dot{x}\left(\frac{\Delta t}{2}\right)\cdot\Delta t$

(3) $p(\Delta t) = p\left(\frac{\Delta t}{2}\right) + \frac{\Delta t}{2}\cdot F(x(\Delta t))$

Recall that $p(t) = m\cdot v(t)$ and so we can replace $p(t)$ with $v(t)/m$. We also know $\dot{x}(t) = v(t)$. Making these simplifications yields the following steps so that we can compute one time-step:

(1) $v\left(\frac{\Delta t}{2}\right) = v(0) + \frac{\Delta t}{2m}\cdot F(x(0))$

(2) $x(\Delta t) = x(0) + v\left(\frac{\Delta t}{2}\right)\cdot\Delta t$

(3) $v(\Delta t) = v\left(\frac{\Delta t}{2}\right) + \frac{\Delta t}{2m}\cdot F(x(\Delta t))$

Alternatively, we can substitute (1) into (2) and (3), yielding

$$\begin{cases} x(\Delta t) = x(0) + v(0)\cdot\Delta t + \frac{(\Delta t)^2}{2m}\cdot F(x(0)) \\ v(\Delta t) = v(0) + \frac{\Delta t}{2m}\cdot\Big(F(x(\Delta t)) + F(x(0))\Big) \end{cases}$$

This numerical scheme is often referred to as the Velocity Verlet integrator.

2.3 Time-Reversal Property of Velocity Verlet:

We now show that the Velocity Verlet numerical scheme is time-reversible, implying that it is stable.

We start by returning to equation (5),

$$f(t) = e^{iLt}\,f(0)$$

where L is a self-adjoint operator (i.e., $L = L^{\dagger}$). Let $G(t) = e^{iLx}$. Imagine we time evolve our particle $f(0)$ by some arbitrary time t. We then have $f(t) = G(t)f(0)$. If we can show

$G(-t)\,f(t) = f(0)$, then this means our operator $G(t)$ reverses time. Thus it suffices to show $G(-t) = \big(G(t)\big)^{-1}$. Observe that

$$\Big(G(t)\Big)^{\dagger} = \big(e^{iLt}\big)^{\dagger} = \left(\sum_{n=0}^{\infty} \frac{(iLt)^n}{n!}\right)^{\dagger} = \sum_{n=0}^{\infty} \frac{(-it)^n}{n!}(L^{\dagger})^n = \sum_{n=0}^{\infty} \frac{(-it)^n}{n!}(L)^n = \sum_{n=0}^{\infty} \frac{(-itL)^n}{n!},$$

where we have used properties of the adjoint and the fact that $L = L^{\dagger}$.

Then $e^{-iLt} = e^{iL(-t)} = G(-t)$ by definition, but we also know that $e^{-iLt} = \big(e^{iLt}\big)^{-1} = \big(G(t)\big)^{-1}$. Therefore, $G(-t) = \big(G(t)\big)^{-1}$.

However, we approximated $G(t) = e^{iLt} \approx \Big(e^{iL_p \frac{\Delta t}{2}}\, e^{iL_x \Delta t}\, e^{iL_p \frac{\Delta t}{2}}\Big)^P$ for large P, where $\Delta t = T/P$. But from (6), this approximation had $\mathcal{O}(P^{-2}) \ll \epsilon$ for some small $\epsilon > 0$ when P is very large (i.e., we take many time steps). Since in MD simulations, one time step $\sim$ femtosecond (which means P is very large), the energy in the system is approximately conserved.

3 Methods:

Now that we have proved in theory that the Velocity Verlet algorithm should be time reversible (whereas Euler's method should not), we aim to demonstrate this. We will attempt to do this by running an MD simulation on 2F4K, a small (35 residues), ultra-fast folding protein [6].

When attempting to run this protein using *OpenMM* – a Python library for performing MD simulations – we encountered multiple errors as the AMBER molecular forcefield could not recognize several of the residues. To resolve this, we downloaded *OpenMM Setup* which is a Graphical User Interface (GUI) for OpenMM. Included in this GUI is *PDB Fixer*, which enables us to replace these erroneous residues by their most similar residue. OpenMM Setup also allows us to choose which parts of the original PDB file we want to include in our simulation (e.g. protein, ligands, other molecules present in the system etc.), and add hydrogen atoms, solvents (specifying the pH of the solution and if we want to include ions), and any atoms that are missing in the residues. We can also the run simulations through the GUI. However, it doesn't allow us to create and implement our own integrator. So we will write our own code in a separate script.

After uploading 2F4K.pdb into OpenMM Setup, we began to modify the PDB file so that it would be compatible with the OpenMM simulation library. To make our simulation run faster, we removed molecules not part of the protein structure. We then replaced the difficult residues (NLE at position 65 and 70) with their most similar residue compatible with the AMBER forcefield – LEU. We also had to add missing heavy atoms (CD1, CD2) to both of these residues. We then added hydrogen (pH = 7.0) because the simulation wouldn't run without it. However, we did not include water in our system to speed up our simulations. We then exported the new 2F4K structure as a PDB file and began to program our simulation in a Jupyter Notebook using the OpenMM library.

We began by coding both methods using the *openmm.CustomIntegrator* object. We then implemented a function that would compute a simulation at 300° Kelvin consisting of $1,000,000$ time-steps, where each time-step was 0.01 femtoseconds. It took a while to decide on this number. We observed that time-steps slightly larger than 0.1 caused our protein to rotate uncontrollably, and time-steps larger than 1 femtoseconds "blew-up" within a few thousand iterations (i.e., the numerical method was unstable!). We speculate that this was because we were simulating in free space (i.e, no solvent) and that our protein wasn't fixed to anything. However, in simulations ~ 0.01, the protein didn't have this problem.

Now that our simulation could ran and appeared stable, we time-evolved 2F4K and at every $5,000$ steps added a PDB frame to our simulation. At the end of the simulation, we also outputted a separate PDB. We did this using both the Euler and Velocity Verlet integrators. The reason we returned a separate file when the simulation ended was due to a somewhat insignificant error in the *PDBReporter* object. When the PDBReporter object adds a new frame to the simulation, it only includes the position of the residues and not the secondary structure formed by them. This will present a discrepancy in the RMSD data below. We therefore used the *PDBFile* object to write the final PDB structure at the end since we couldn't figure out how to create a simulation with it.

In an effort to see if energy was conserved, we also ran a time reversed simulation on both methods. We used the final PDB structure after time-evolving 2F4K as the starting position, and set the initial velocities to be negative the final velocities of the time-evolution simulation. This reverses time in our simulation: Assume a particle starts at any point (x_0, v_0). Over time t_f, our particle moves to the point (x_f, v_f) by Newton's equations. Now, invert the velocity at this point, i.e. $(x_f, -v_f)$. If we now start at (x_f, v_f) and follow our particle over time t, we will eventually arrive at (x_0, v_0).

4 Results and Analysis:

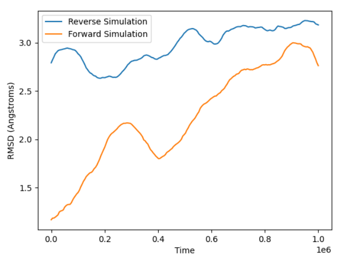
Now that we have simulations and structures (both forward in time and reversed in time), we have lots of data to analyze. Let's first see how the RMSD of $E_{t_0 \to T \to t_0}$ and $V_{t_0 \to T \to t_0}$ to the original structure. Here, $t_0 \to T \to t_0$ means that we time-evolved and then time-reversed and the first letter refers to the integrator (Euler or Velocity Verlet). Opening the appropriate files in PyMOL and aligning structures, we find that:

	$E_{t_0 \to T \to t_0}$	$V_{t_0 \to T \to t_0}$
RMSD	2.957	0.986

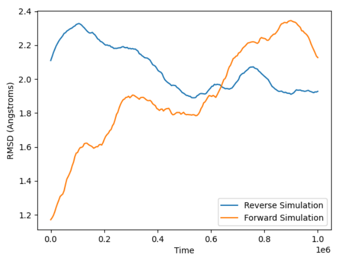
While Velocity Verlet didn't return to the exact structure it originated from, we do notice that it outperformed Euler's method substantially. It's also important to note that we only ran our simulation for 10 picoseconds, which is a small fraction of the time that would typically be run. Therefore, we would expect this error to compound the longer this simulation was run.

We also can compare the RMSD at every $5,000$ steps to the original structure. This yields the following graphs 1.

As alluded to in the Methods section, the ending RMSD value of the reversed simulations do not match the data in the table above. We speculate that this is because of the fact that the simulation had no secondary structure, but the final PDB structures did (we compared the final PDB structures to the original structure to get the RMSD values).
The graph also illustrates the approximate energy conservation of the Velocity Verlet integrator as seen by the rough symmetry between the blue and orange lines.



((a)) Euler



((b)) Velocity Verlet

Figure 1: Forward and Reverse Simulations for Euler and Velocity Verelet Integrators

5 Future Work:

There is still much work to be done. First off, I would change the simulation parameters to include a solvent to get a more realistic result. This would also allow me to figure out why the protein rotates uncontrollably when time-steps exceed ~ 0.1 femtoseconds. Then I could ascertain whether or not the short time-step is directly because of the algorithm. I would also run longer simulations so that I could get a better sense of the long-time stability of the numerical algorithms since that it is topic of consideration. Lastly, I would consider more proteins (including ligand-bound proteins) and numerical algorithms (like RESPA and NAPA) so that I could get a more complete understanding as to how certain integrators affect the stability and accuracy of MD simulations.

References

[1] Hollingsworth, Scott A., and Ron O. Dror. "Molecular Dynamics Simulation for All." *Neuron*, vol. 99, no. 6, Sept. 2018, pp. 1129–43. *ScienceDirect*, https://doi.org/10.1016/j.neuron.2018.08.011.

[2] Russell, John. "Anton 3 Is a 'Fire-Breathing' Molecular Simulation Beast." *HPCwire*, 2 Sept. 2021, https://www.hpcwire.com/2021/09/01/anton-3-is-a-fire-breathing-molecular-simulation-beast/.

[3] Dror, Ron. "CS 279: Molecular Dynamics Simulation" 06 Oct. 2022, Stanford University. Lecture.

[4] Tuckerman, M., et al. "Reversible Multiple Time Scale Molecular Dynamics." *The Journal of Chemical Physics*, vol. 97, no. 3, Aug. 1992, pp. 1990–2001. *DOI.org*, https://doi.org/10.1063/1.463137.

[5] De Raedt, Hans, and Bart De Raedt. "Applications of the Generalized Trotter Formula." *Physical Review A*, vol. 28, no. 6, Dec. 1983, pp. 3575–80. *DOI.org*, https://doi.org/10.1103/PhysRevA.28.3575.

[6] Lindorff-Larsen, K., et al. "How Fast-Folding Proteins Fold." *Science*, vol. 334, no. 6055, Oct. 2011, pp. 517–20. *DOI.org*, https://doi.org/10.1126/science.1208351.

[7] Kofke, David. "CE 530 Molecular Simulation: Symmetric MD Integrators" 13 April. 200, SUNY Buffalo. Lecture.