# Phys 256B Project Report (Spring Quarter 2013): Artifacts of Molecular Dynamics

Nithin Dhananjayan
Grad Group:Biophysics, Department:Chemistry
University of California, Davis
ndhananj@ucdavis.edu

June 15, 2013

**Abstract**

To understand the non-linear behavior that algoritms that make time discrete add to simulations (in addition to the potentially non-linear behavior of what is being simulated) I compared Stromer-Verlet algorithm with Runge-Kutta Order 4 for both a harmonic oscillator and a harmonic oscillator bouncing against a hard wall. I wanted to do this so that I can get a better understanding of the types of issues that people doing molecular dynamics run into, since many available codes (Amber, CHARMM, Gromacs) use some version of the Stromer-Verlet Algorithm (usually the Leap-Frog algorithm). Although, I would like to study the cases where damping and noise are accounted for, there was some non-linear behavior seen in Runge-Kutta 4 near the point where it becomes unstable. I isolated some the parameters and some return maps that seem to indicate an attractor of some kind that would be interesting to characterize.

# 1 Introduction

## 1.1 Motivation

The Stromer-Verlet algorithm is used my many molecular dynamics simulation packages. This includes popular molecular simulation packages like Gromacs, Amber, and CHARMM. Although there are different implementations: velocity explicit formulation, leap-frog formulation, and position-Verlet formulations, these formulations can be proven to give the same results. They need slightly different initialization procedures with velocity explicit having the simplest initialization.

In addition, Stromer-Verlet has some nice properties given its simplicity that higher order systems, like Runge-Kuttta-4, do not have. Because of this, I wanted to make a comparison between two algorithms, Stromer-Verlet, and Ruge-Kutta-4 regarding the non-linear dynamics that they introduce (in additions to the dynamics that they are simulating). So my project is to explore what these additional dynamics are.

## 1.2 Why is it interesting?

Computational chemists make use of Molecular Dynamics to answer many questions about chemical systems that range from complicated periodic materials to a biological complex of molecules. I am, myself, working on a biological system known as NADH:Dehydrogenase, that acts as a chemical proton pump that is powered by electron transport from a higher reduction potential to lower reduction potential. I wonder what we are sacrificing by using the Stromer-Verlet algorithm, and what we are gaining as over an explicit Runge-Kutta method.

## 1.3 Project Summary

Initially, I was interested in finding out about the artifacts that simulation algorithms can inject into the simulation. Unfortunately, many of the artifacts that I had tried to track down were created by subtle bugs in my code, having to do with how I initialized the algorithm, or the order in which I updated the variables.

In this report, I give some details about one artifact I still believe to real and would like to investigate further. At particular parameters I am able to see a structured loss of energy in the phase space plot of the Runge-Kutta 4 method, while the Stromer-Verlet remained stable. Following up on this case, I was able to isolate a set of points that show up in return maps of multiple periods. I believe going through the whole 'learning channel', studying this one case (which I did not find till late), would be instructive.

# 2 Background

## 2.1 Molecular Dynamics

Molecular Dynamics software packages are used in computational biochemistry to answer basic questions about conformations and placements about particular molecules in relation to each other and to get order of magnitude estimates on the forces, energies, and free energies involved.

In order to do a simulation of this sort, molecular dynamics software packages simulate a discrete time version of Newton's Laws working in response to a variety of bonded and non-bonded forces. Although, there are packages (sometimes called ab intio molecular dynamics, or QM-MM) that take into account quantum mechanics, even these packages use quantum mechanics only to create a static

potential created by electrons for the nuclei to move in for each time step (making use of the Born-Oppenheimer approximation). Because of this, I will only go into detail regarding the Newtonian parts of the simulation.

## 2.2 Gromacs as an example molecular dynamics package

One popular, and one of the fastest molecular dynamics simulation packages, is called Gromacs– which stands for GROningen MAchine for Chemical Simulations.

The basic algorithm that Gromacs follows is outlined below:

(i) compute accelerations in accordance with forces disregarding constraints, $a_i^{(t)} = \frac{F_i^{(t)}}{m_i}$

(ii) Update and scale velocities: $v'^{(t)} = \lambda(v^{(t-\Delta t)} + a * \Delta t)$ where $\lambda = \left[1 + \frac{\Delta t}{\tau_t}\left(\frac{T_0}{T^{t-\frac{1}{2}\Delta t}} - 1\right)\right]^{\frac{1}{2}}$

(iii) Compute new unconstrained positions: $r'^{(t)} = r^{(t-\Delta t)} + v'^{(t)}\Delta t$

(iv) Apply constraint solver: $r'^{(t)} \rightarrow r''^{(t)}$

(v) Adjust velocity based on constraints: $v^{(t)} = \frac{r''^{(t)} - r^{t-\Delta t}}{\Delta t}$

(vi) Scale coordinates and size of simulation box: $r^{(t)} = \mu r''^{(t)}$, $b = \mu b$, $\mu = \left[1 + \frac{\Delta t}{\tau_p}\beta\left(P^{(t)} - P_0\right)\right]^{\frac{1}{3}}$

The scaling in step (ii) is for the purpose of applying a thermostat, to try to maintain constant temperature. I will be using a different method because it is not clear that this method creates the proper use of time steps to keep the nice properties of Stromer-Verlet that I will talk about later. Also, for a single particle, applying a background "noise" to simulate a "heat bath" and a damping factor to simulate and implicit solvent is more appropriate.

The scaling in step (vi) is to apply a barostat to maintain a desired pressure. I will not be making any adjustments to pressure, since this becomes quite confusing for a single particle case, and again, I am not sure how to maintain the nice properties of Stromer-Verlet when applying a barostat.

Constrains are another complication in Gromacs that I will be avoiding. The purpose of constraints in a molecular dynmics simulation is to create infinitely stiff bonds without having forces that have very steep curves. The time step required for a simulation is usually limited by how steep the potential being simulated are. Also, very stiff bonds, like Carbon-Hydrogen bonds are usually not activated by the temperatures typical in biology, and since our simulations don't account for quantum mechanics, we would want to treat bonds like this as constraints rather than extremely stiff springs.

The forces that are usually involved in molecular dynamics are: (1) Harmonic forces use to simulate bonds, angles between two bonds, and dihedral angles. (2) long range electrostatic forces modeled by Coulomb forces. (3) $r^{-12}$ potentials to model strong repulsive interactions without discontinuities, and (4) $r^{-6}$ potentials to model dipole-dipole interactions.

## 2.3 Stability of Stromer-Verlet and Runge-Kutta

One nice think about Stromer-Verlet is that the stability of Stromer-Verlet Algorithm (above) depends only on the relation of time step to maximum frequency, while the stability of Runge-Kutta (below) has many factors. See Figure 1.

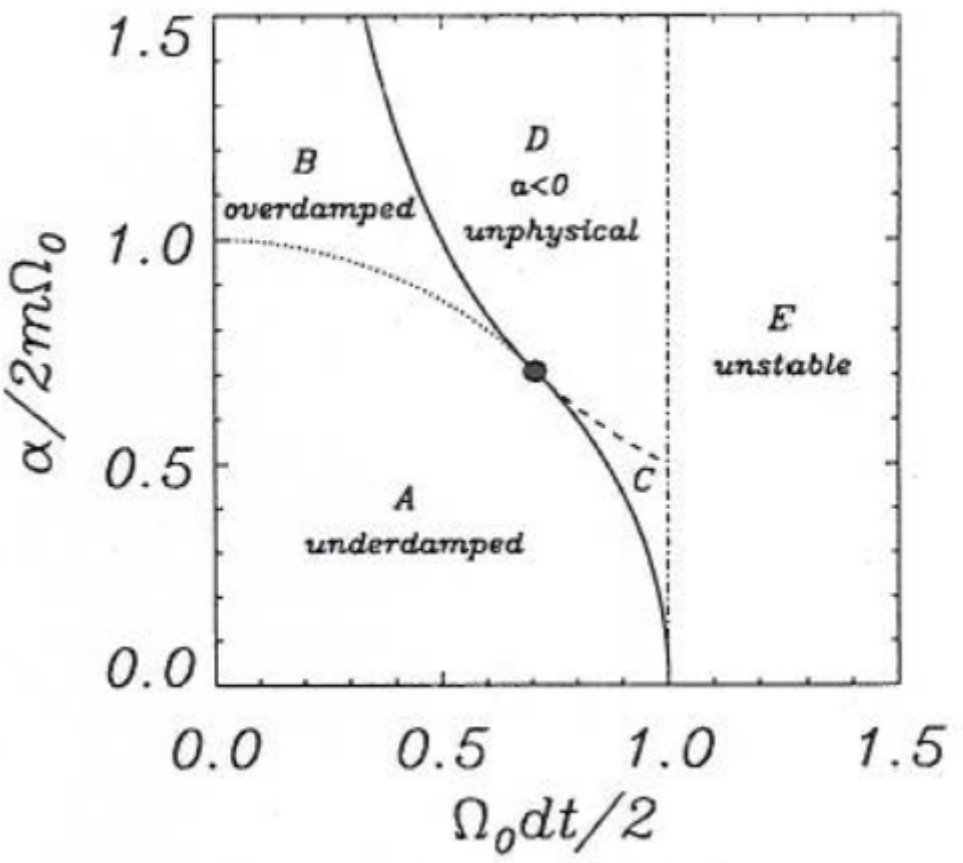


Figure 1: Stability of Stromer-Verlet Algorithm (above[2]) depends only on the relation of time step to maximum frequency, while the stability of Runge-Kutta (below[3]) has many factors.

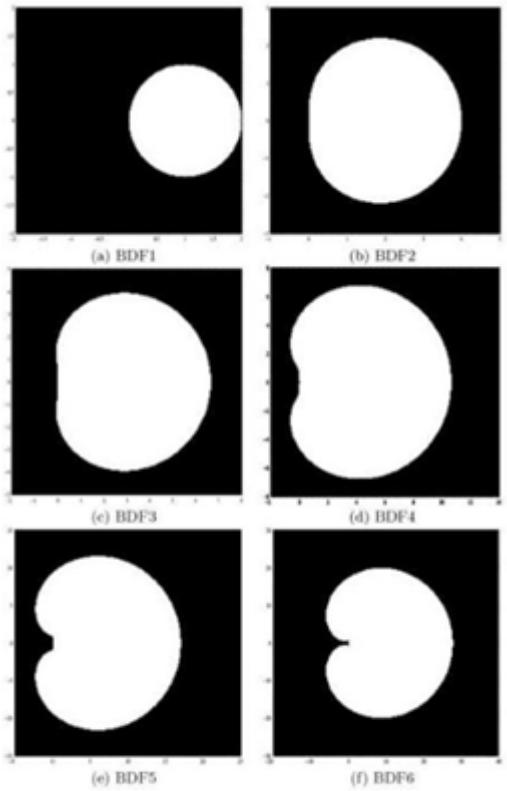


Fig. 12.5. Absolute stability regions in the complex plane for $k$-step Backward Differentiation Formulae, $k = 1, 2, \ldots, 6$. In each case the region of absolute stability is the set of points in the complex plane outside the white region. In each case, the region of absolute stability contains the whole of the negative real axis.

Beyond this, Stromer-Verlet is symplectic. In the following simulation that I ran where a particle is bouncing off of two hard walls (a situation that will artificially change the energy involved), the symplectic naure of Strome-Verlet shows a distortion in phase space, but a closed loop. See Figure 2. The Rune-Kutta integration for the same parameters was not stable despite being a higher order integrator.

I wanted to see how a higher order algorithm, known not to be symplectic, like Runge-Kutta would do in various situations in comparison to Stromer-Verlet. I wanted to characterize also how these integrators fared as I changed the time-step.

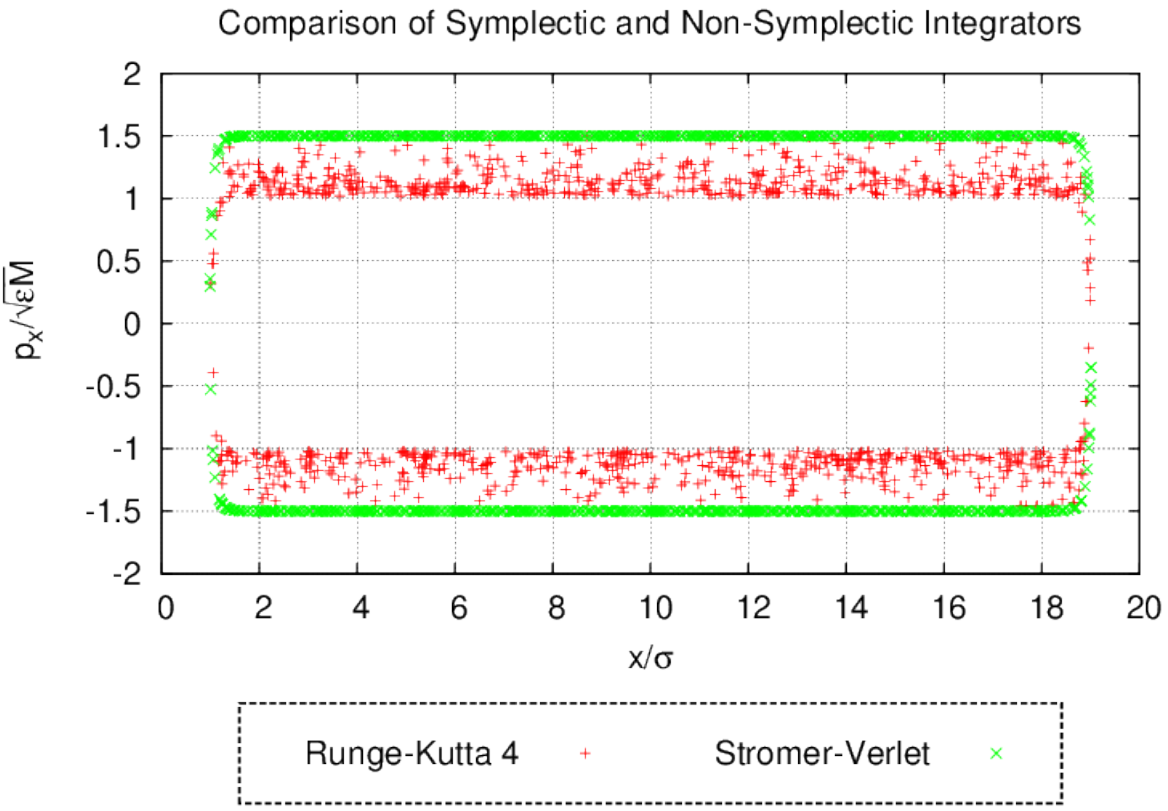


Figure 2: Distortion of phase space by Stromer-Verlet Still keeps a closed loop. However, Runge-Kutta diverges despite having higher order.

# 3 Dynamical System

## 3.1 Underlying Langevin Dynamics

The basic system will evolve according to the following equation:

$$m\ddot{x} = f(x,t) - \alpha\dot{x} + \frac{1}{\tau}\beta(t) \tag{1}$$

Here, $\alpha$ is a damping factor used to represent an implicit viscous fluid through which objects move, and the $\beta(t)$ is meant to represent random perturbations that come from an implicit heat bath. To get the appropriate representation for a heat bath, we need $\langle\beta(t)\rangle = 0$, and $\langle\beta(t)\beta(t')\rangle = 2\alpha k_b T\tau\delta(t-t')$. $\tau$ would be the characteristic time of action by the heat bath. I'll assume it to be small enough that I can use my simulation time step as $\tau$ as crude approximation no matter how small my time step.

### 3.2 Model Systems

The basic systems I want to look at are a harmonic oscillator and a harmonic oscillator bouncing against a hard-wall. To this end, I place the harmonic oscillator inside a box in such a way that I can move the equilibrium position or change the initial velocity so that the system can easily be adjusted to be one of the two model systems I want to explore. The "force field" for this underlying system is:

$$f = k(x - x_{eq}) + \frac{12\epsilon}{\sigma}\left[\left(\frac{\sigma}{x}\right)^{13} + \left(\frac{\sigma}{L_{box} - x}\right)^{13}\right] \tag{2}$$

The first term is the harmonic oscillator, with a k that can be set to zero to get rid of the spring, while the second term represents the walls. I made the simulator use natural units where $\epsilon$ and $\sigma$ are used as units of energy and distance. So to get rid of the walls, I would need to comment that part out of my code.

### 3.3 Algorithms

In some sense, what the algorithms themselves are (was included in my presentation) matter less than what my actual implementation was.

One complication was that I wanted to make my implementations extensible to multiple particles, and have tested basic functionality as an actual, albeit small, MD simulator with multiple particles in a "heat bath" and viscous fluids, with the possibility of including periodic boundary conditions. So include here by listing of the the functions that do the Stromer-Verlet step and the Runge-Kutta step (with some reformatting to fit on the page).

```
// do one step of verlet algorithm
inline void verlet_step(part_data* d, coord dt, coord a, coord b,
  coord T, coord alpha){
  int i,part;
  force tempf[Particles]; position tempr[Particles]; momentum beta[Particles];
  for(part=0; part<Particles; ++part){
    beta[part]=random_momentum(T,dt,alpha);
    //get new position
    for(i=0;i<Dimensions; ++i){
      tempr[part].r[i]=
       d->data[part].r.r[i]+
        (d->data[part].v.r[i]+beta[part].r[i]+(d->data[part].half_f.r[i])*dt)*
         dt*b;
      //d->data[part].r.r[i]+=
      //   (d->data[part].v.r[i]+(d->data[part].half_f.r[i])*dt)*dt;
    }
  }
  for(part=0; part<Particles; ++part){
    tempf[part]=half_force_field(part,d,&(tempr[part])); // get new half_force
    // update velocity
    for(i=0;i<Dimensions; ++i){
      d->data[part].v.r[i]+=
       (d->data[part].half_f.r[i]+tempf[part].r[i])*dt-
        alpha*(tempr[part].r[i]-d->data[part].r.r[i])+beta[part].r[i];
      //d->data[part].v.r[i]+=(d->data[part].half_f.r[i]+tempf.r[i])*dt;
    }
  }
  for(part=0; part<Particles; ++part){
    d->data[part].r=tempr[part]; //update position
```

```
    pbc_correct(&(d->data[part].r));
    d->data[part].half_f=tempf[part]; //update force
  }
}
```

That was the Stromer-Verlet implementation. Next is the Runge-Kutta, I know that there is a simpler way to implement Runge-Kutta 4 (namely optimizing the velocity calculation), but in preparation for including noise, I setup this more robust version. In doing so, I discovered a bug in the initialization of my simpler version, and found that this new version can use much larger time steps.

```
// doing a time deriviative of sorts, ignore the poition slot
inline step_data rk4_update(int part, part_data* d, position* r, velocity* v, coord
  step_data to_ret;
  force int_hf=half_force_field(part,d,r), retVal;
  momentum beta=random_momentum(T,dt,alpha);
  scal_mult_vec(v, half_alpha, &retVal);
  diff_vec(&int_hf,&retVal,&retVal);
  scal_mult_vec(&beta,inv_two_dt,&int_hf);
  add_vec(&int_hf,&retVal,&retVal);
  to_ret.v=(*v);
  to_ret.half_f=retVal;
  return to_ret;
}

// do one step of the RK4 algorithm
inline void rk4_step(part_data* d, coord half_dt, coord dt,
  coord two_dt, coord alpha, coord half_alpha, coord inv_two_dt, coord T){
  int i, part;
  part_data k1,k2,k3,k4,k2_in,k3_in,k4_in;
  vec_coord vec1[Particles],vec2[Particles];
  for(part=0; part<Particles; ++part){
    k1.data[part]=
      rk4_update(part,d,&(d->data[part].r),
        &(d->data[part].v),alpha,half_alpha,dt,inv_two_dt,T);
    scal_mult_vec(&(k1.data[part].v),half_dt,&vec1[part]);
    add_vec(&(d->data[part].r),&vec1[part],&(k2_in.data[part].r));
    pbc_correct(&(k2_in.data[part].r));
    scal_mult_vec(&(k1.data[part].half_f),dt,&vec2[part]);
    add_vec(&(d->data[part].v),&vec2[part],&(k2_in.data[part].v));
    k2.data[part]=
      rk4_update(part,d,&(k2_in.data[part].r),
        &(k2_in.data[part].v),alpha,half_alpha,dt,inv_two_dt,T);
    scal_mult_vec(&(k2.data[part].v),half_dt,&vec1[part]);
    add_vec(&(d->data[part].r),&vec1[part],&(k3_in.data[part].r));
    pbc_correct(&(k3_in.data[part].r));
    scal_mult_vec(&(k2.data[part].half_f),dt,&vec2[part]);
    add_vec(&(d->data[part].v),&vec2[part],&(k3_in.data[part].v));
    k3.data[part]=
      rk4_update(part,d,&(k3_in.data[part].r),
        &(k3_in.data[part].v),alpha,half_alpha,dt,inv_two_dt,T);
    scal_mult_vec(&(k3.data[part].v),dt,&vec1[part]);
    add_vec(&(d->data[part].r),&vec1[part],&(k4_in.data[part].r));
    pbc_correct(&(k4_in.data[part].r));
    scal_mult_vec(&(k3.data[part].half_f),two_dt,&vec2[part]);
```

```
      add_vec(&(d->data[part].v),&vec2[part],&(k4_in.data[part].v));
      k4.data[part]=
        rk4_update(part,d,&(k4_in.data[part].r),
          &(k4_in.data[part].v),alpha,half_alpha,dt,inv_two_dt,T);
    }
    for(i=0;i<Dimensions; ++i){
      for(part=0; part<Particles; ++part){
        d->data[part].half_f.r[i]=
          (k1.data[part].half_f.r[i]+2.0*k2.data[part].half_f.r[i]+
            2.0*k3.data[part].half_f.r[i]+k4.data[part].half_f.r[i])/6.0;
      }
      for(part=0; part<Particles; ++part){
        d->data[part].r.r[i]+=
          dt*(k1.data[part].v.r[i]+2.0*k2.data[part].v.r[i]+
            2.0*k3.data[part].v.r[i]+k4.data[part].v.r[i])/6.0;
        pbc_correct(&(d->data[part].r));
      }
      for(part=0; part<Particles; ++part){
        d->data[part].v.r[i]+=two_dt*d->data[part].half_f.r[i];
      }
    }
}
```

# 4 Methods

I implemented a Stromer-Verlet and Runge-Kutta simulator using c. Then created time evolution plots, phase plots, return maps, and bifurcation diagrams to understand the non-linear behavior of failure modes.

I would like to calculate Lyapanov Characteristic Exponents of something that looks like an attractor to me. I would like to map out the generating partitions for the system that maximize entropy rate. But the implementation of the simulators had more subtleties than I expected. I have incorporated damping and noise into my simulator but the parameters are set to zero for now. I would like to explore these cases further.

# 5 Results

## 5.1 Confirmation that simulator works

In order to test that the simulator works, I ran both methods for a small time-step with no damping and no noise, and saw the expected behavior in both a low velocity and high velocity case (bouncing off walls). See Figure 3. It turns out, however, that checking for agreement and functionality at small times steps is not adequate to ensure bugs are eliminated. Nevertheless, it is a good sanity check.

## 5.2 Some Divergence of Between Algorithms seen even without Damping and Noise

To explicitly see an interesting case, I moved the equilibrium position near the wall. Here, I could see the Stromer-Verlet algorithm was still well behaved, while the Runge-Kutta had an interestingly structured energy loss in the simulation at the same time step. See Figure 4.

I thought that using the angle (or more easily, the tangent of the angle) from point (20,0) on the phase plot, could serve to be something close to a bifurcation diagram with the the parameter being

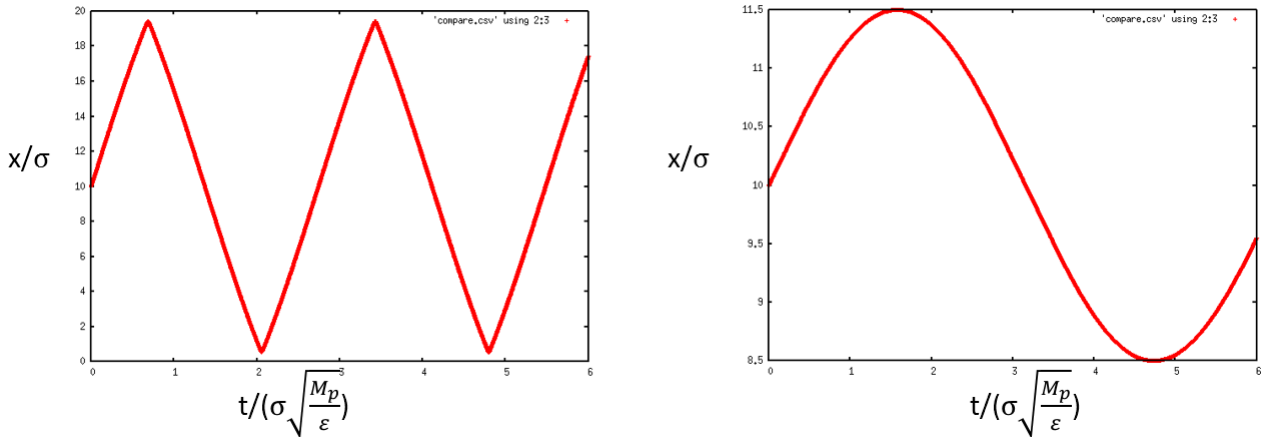


Figure 3: For small time step both Stromer-Verlet and Runge-Kutta work as expected. On the left is the time evolution of a high-velocity case where the oscillator bounces off of walls, and on the right is a low velocity case where the the initial velocity is lower and the oscillator is allowed to go its full distance. The time step used was $dt = 6 * 10^{-7}\left(\sigma\sqrt{\frac{M_p}{\epsilon}}\right)$

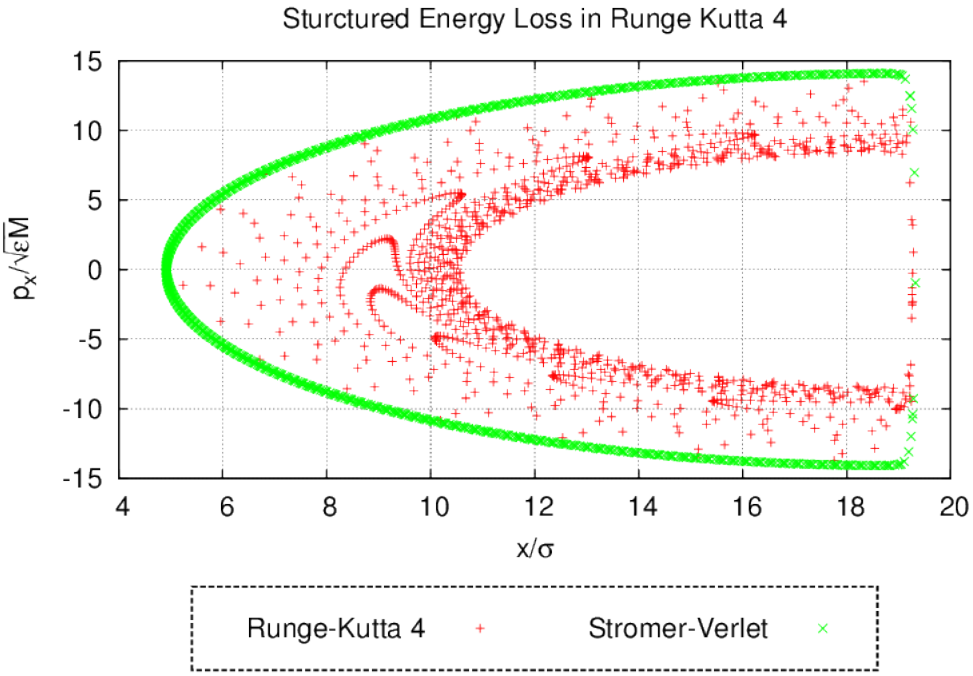


Figure 4: Runge-Kutta and Stromer Verlet diverge with $dt = 5 * 10^{-3}\left(\sigma\sqrt{\frac{M}{\epsilon}}\right)$

the tangent in phase space. The result of this is shown in Figure 5. It is not clear that these points in phase space represent fixed points or limit cycles or anything of that sort. But I thought, it would present a good way to look at what might be interesting to explore further.

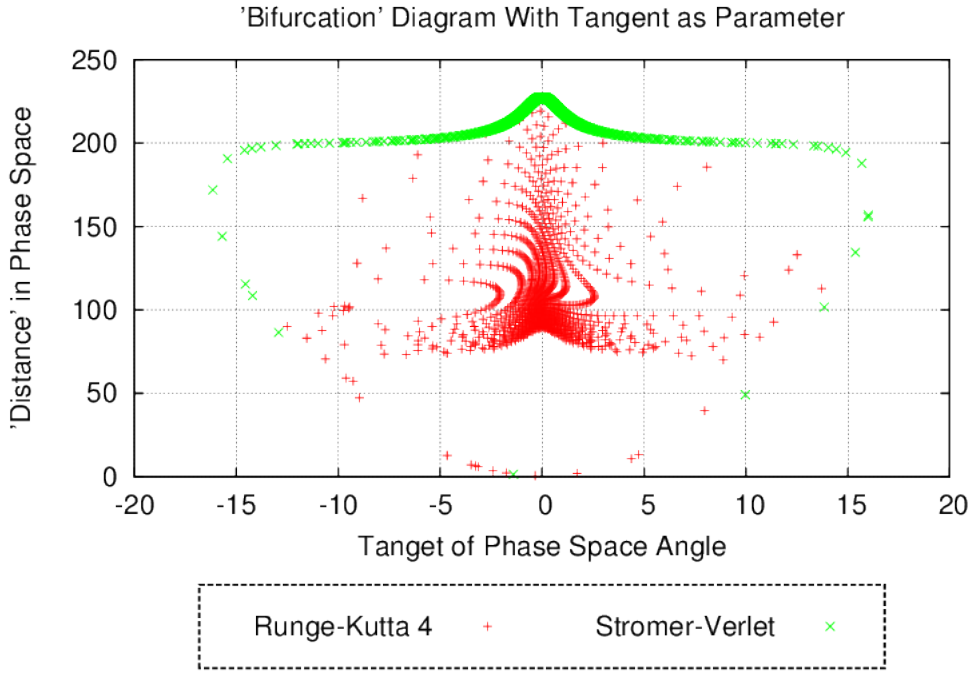


Figure 5: Runge-Kutta and Stromer Verlet with $dt = 5 * 10^{-3}$ $\left(\sigma\sqrt{\frac{M}{\epsilon}}\right)$

Now, in the case given above,an angle of 0 may be an interesting point. If we look at only the points where the velocity (or momentum) is effectively zero, we get a fairly ordered plot over time. See Figure 6.

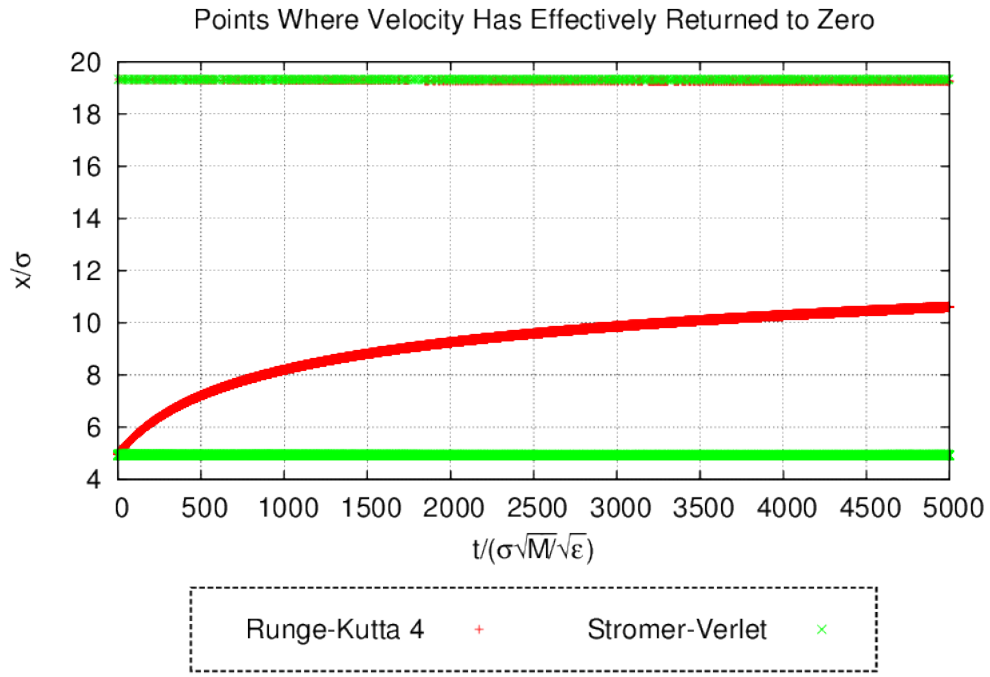


Figure 6: Runge-Kutta and Stromer Verlet with $dt = 5 * 10^{-3}$ $\left(\sigma\sqrt{\frac{M}{\epsilon}}\right)$

Another point to look at might be a phase angle tangent of 1. If we look at when we have the desired phase within a tolerance in our simulations, we see the the Stromer-Verlet algorithm takes on continuous range of values, but the Runge-Kutta only some. See Figure 7. Our oscillator has a small period compared to the time scale plotted.

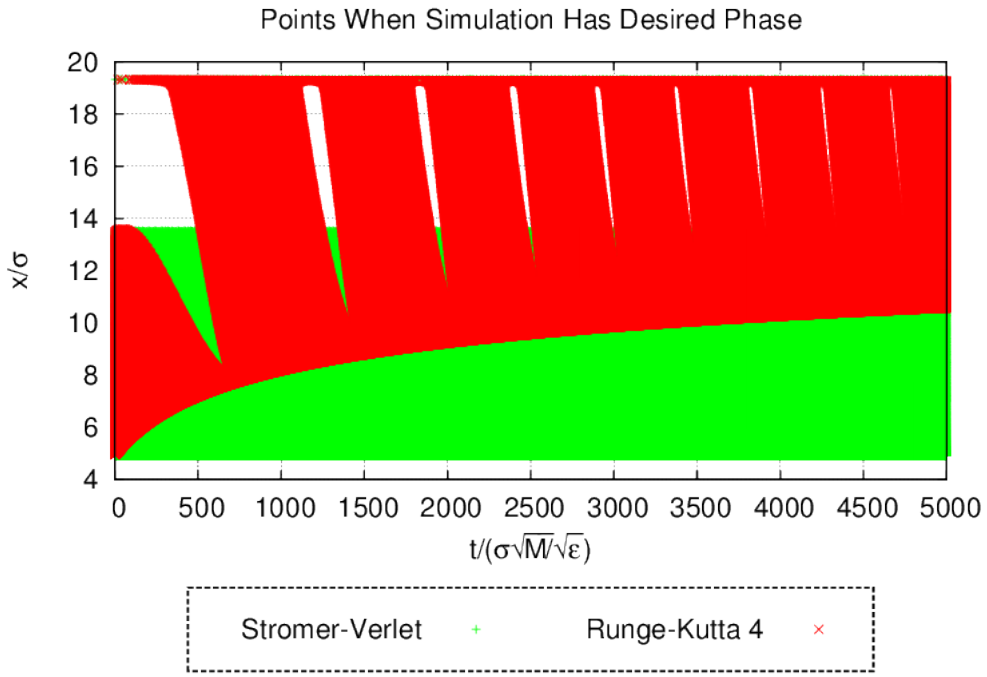


Figure 7: Runge-Kutta and Stromer Verlet with $dt = 5 * 10^{-3}\left(\sigma\sqrt{\frac{M}{\epsilon}}\right)$. The targeted phase angle tangent was 1

For this case, it may be interesting to look at the return map to see what it could tell us. See Figure 8. Here we see, that there are some components at this phase space angle that are not fixed points. Further examination, looking at larger periods show them these regions to expand when we look at higher period return maps. See Figure 9.

I find it interesting that the very same points in the return map show up at multiple period lengths. To me, this seems to indicate that these points in phase space are an attractor of some sort, and I want to investigate this attractor further.

## 6 Conclusions

Even though, in principle, molecular dynamics is just an implementation of a differential equation solver, there are many issues that need to be addressed to get stable results in reasonable time frames. I attempted to look at the artifacts that can occur if we use an integrator that doesn't properly account for conjugate variables. However, many of the things I thought we issues with Runge-Kutta or Stromer-Verlet turned out to be bugs in my own code.

Once (I believe) I sorted these issues out. I was able to isolate non-linear behavior in the phase map at particular control parameters. Using return maps, I believe, I've isolated some attractors in the flow. I believe this to be the case because the same points show up on return maps at multiple periods without being limit cycles of the given length. I would like to explore this further by making small perturbations at these phase space points to see how the disturbances grow.

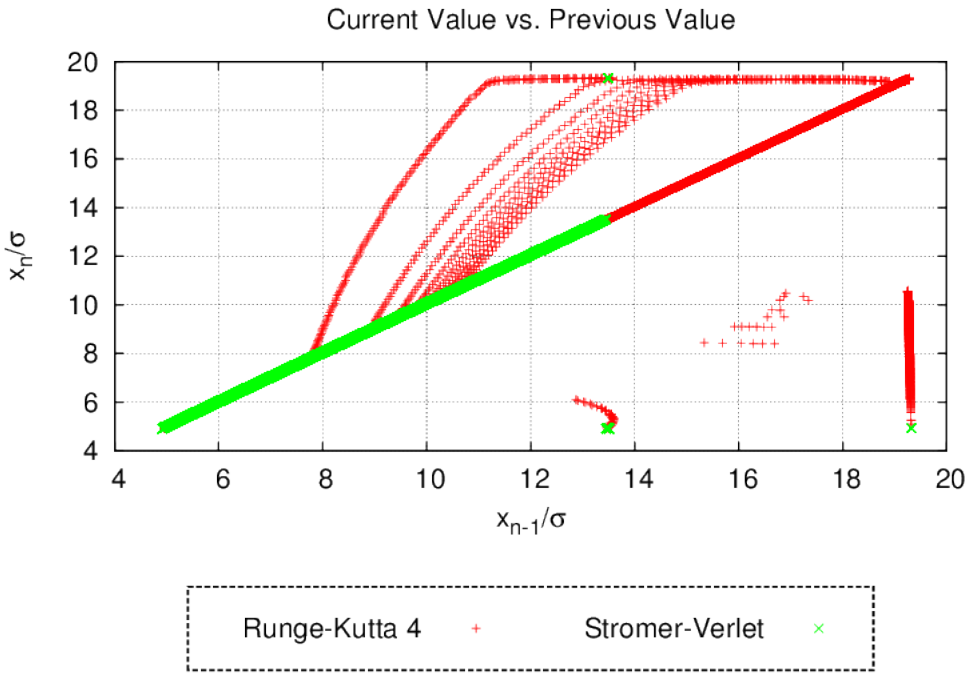


Figure 8: Return map for Runge-Kutta and Stromer Verlet with $dt = 5 * 10^{-3}\left(\sigma\sqrt{\frac{M}{\epsilon}}\right)$. The targeted phase angle tangent was 1

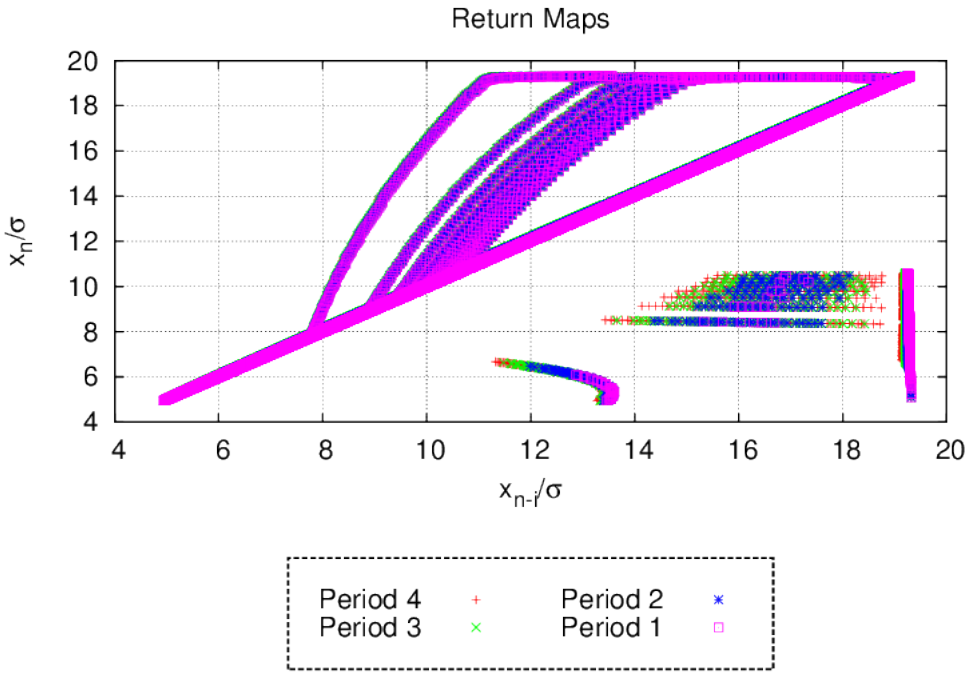


Figure 9: Multiple length period return maps for Runge-Kutta with $dt = 5 * 10^{-3}\left(\sigma\sqrt{\frac{M}{\epsilon}}\right)$. The targeted phase angle tangent was 1

# 7 Bibliography

1. Crutchfield, J.P. “Lecture Notes” Natural Computation and Self-Organization Winter/Spring 2013

2. C. C. Strelioff and J. P. Crutchfield, ”Optimal Instruments and Models for Noisy Chaos”, CHAOS 17 (2007) 043127. Santa Fe Institute Working Paper 06-11-042. arxiv.org e-print cs.LG/0611054.

3. GROMACS: A message-passing parallel molecular dynamics implementation, H.J.C. Berendsen, D. van der Spoel, R. van Drunen, Computer Physics Communications 91 (1995) 43-56

4. A simple and effective Verlet-type algorithm for simulating Langevin Dynamics, Neils Gronbach-Jensen, Oded Farago, Molecular Physics 2013

5. Süli, Endre; Mayers, David (2003), An Introduction to Numerical Analysis, Cambridge University Press, ISBN 0-521-00794-1.

6. http://en.wikipedia.org/wiki/Runge%E2%80%93Kutta_methods