

PHYS 516: Methods of Computational Physics
ASSIGNMENT 4- Molecular Dynamics Simulation

Anup V Kanale
February 22, 2017

The purpose of this assignment is to get familiar with basic concepts in ordinary differential equations using a simple molecular dynamics (MD) program, *md.c*, as an example.

1 Liouville Theorem

Here, we show that for a particle in 1-D space, the velocity Verlet algorithm exactly preserves the phase space volume for arbitrary time discretization unit, Δ .

Let x be the position of the particle, $p = mv$ the momentum where, m and v are the mass and velocity of the particle. In the dimensionless form, $p = v$. Let the coordinate and momentum of the particle at time t be (x, p) , and those at time $t + \Delta$ be (x', p')

$$x' = x + p\Delta + \frac{1}{2}a(x) + \Delta^2 \quad (1)$$

$$p' = p + \frac{a(x) + a(x')}{2}\Delta \quad (2)$$

To show this, it is enough to show that the Jacobian of the transformation $(x'(x, p), p'(x, p))$ is 1, i.e. $J = \left| \frac{\partial(x', p')}{\partial(x, p)} \right| = 1$. The derivatives are calculated from Eq (1) and (2) as follows:

$$\begin{aligned} \frac{\partial x'}{\partial x} &= 1 \\ \frac{\partial x'}{\partial p} &= \Delta \\ \frac{\partial p'}{\partial x} &= 0 \\ \frac{\partial p'}{\partial p} &= 1 \end{aligned}$$

Therefore, the Jacobian is given by

$$J = \begin{vmatrix} \frac{\partial x'}{\partial x} & \frac{\partial p'}{\partial x} \\ \frac{\partial x'}{\partial p} & \frac{\partial p'}{\partial p} \end{vmatrix} \quad (3)$$

$$= \begin{vmatrix} 1 & 0 \\ \Delta & 1 \end{vmatrix} \quad (4)$$

$$= 1 \quad (5)$$

2 Comparison of Euler and Velocity-Verlet Algorithms

Total energy of the system should be conserved. But due to numerical errors, energy conservation depends on the time-integration algorithm used. In this section, we illustrate this by comparing the Euler and Velocity-Verlet algorithms.

Euler Algorithm

Given a configuration $(\mathbf{r}_i(t), \mathbf{v}_i(t)) | i = 1 \text{ to } N_{atom}$

1. Compute the acceleration: $\mathbf{a}_i(t)$
2. Update the positions: $\mathbf{r}_i(t + \Delta) \leftarrow \mathbf{r}_i(t) + \mathbf{v}_i(t)\Delta + \frac{1}{2}\mathbf{a}_i(t)\Delta^2$
3. Update the velocities: $\mathbf{v}_i(t + \Delta) \leftarrow \mathbf{v}_i(t) + \mathbf{a}_i(t)\Delta$

Velocity-Verlet Algorithm

Given a configuration $(\mathbf{r}_i(t), \mathbf{v}_i(t)) | i = 1 \text{ to } N_{atom}$

1. Compute the acceleration: $\mathbf{a}_i(t)$
2. $\mathbf{v}_i(t + \frac{\Delta}{2}) \leftarrow \mathbf{v}_i(t) + \mathbf{a}_i(t)\frac{\Delta}{2}$
3. $\mathbf{r}_i(t + \Delta) \leftarrow \mathbf{r}_i(t) + \mathbf{v}_i(t + \frac{\Delta}{2})\Delta$
4. Compute the updated acceleration: $\mathbf{a}_i(t + \Delta)$
5. $\mathbf{v}_i(t + \Delta) \leftarrow \mathbf{v}_i(t + \frac{\Delta}{2}) + \mathbf{a}_i(t + \Delta)\frac{\Delta}{2}$

From the plot below, we can clearly see that Euler scheme blows up whereas Velocity Verlet scheme is stable and conserves total energy of the system.

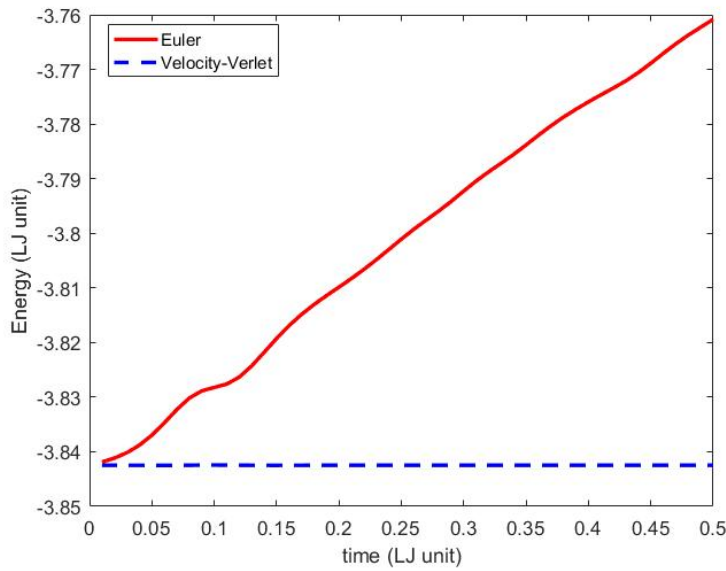


Figure 1: Comparison of total energy conservation for Euler and Velocity verlet algorithms

The code downloaded from the course website was used, which implements the Velocity Verlet algorithm by default. Euler scheme was implemented by modifying the `SingleStep()` section in the code. The time step was changed to 0.001, but for all the other parameters, the default values were used from the input file `md.in`.

Since the code is rather long and the modifications were minor, just the parts with the numerical schemes are shown below.

```

146  /*-----*/
147  void SingleStep() {
148  /*-----*/
149      r & rv are propagated by DeltaT in time using the velocity-Verlet method.
150  /*-----*/
151      int n,k;
152
153      for (n=0; n<nAtom; n++){ /* For each atom */
154          for (k=0; k<3; k++){ /*Update atomic coordinates to r(t+Dt) */
155              r[n][k] = r[n][k] + DeltaT*rv[n][k] + 0.5*DeltaT*DeltaT*ra[n][k];
156              rv[n][k] = rv[n][k] + DeltaT*ra[n][k];
157          }
158      }
159
160      ApplyBoundaryCond();
161      ComputeAccel(); /* Computes new accelerations, a(t+Dt) */
162  }
163
164  /*-----*/

```

Line 160, Column 1 Tab Size: 4 C

Figure 2: Euler scheme implementation

```

146  /*-----*/
147  void SingleStep() {
148  /*-----*/
149      r & rv are propagated by DeltaT in time using the velocity-Verlet method.
150  /*-----*/
151      int n,k;
152
153      HalfKick(); /* First half kick to obtain v(t+Dt/2) */
154      for (n=0; n<nAtom; n++) /* Update atomic coordinates to r(t+Dt) */
155          for (k=0; k<3; k++) r[n][k] = r[n][k] + DeltaT*rv[n][k];
156      ApplyBoundaryCond();
157      ComputeAccel(); /* Computes new accelerations, a(t+Dt) */
158      HalfKick(); /* Second half kick to obtain v(t+Dt) */
159  }
160
161  /*-----*/
162  void HalfKick() {
163  /*-----*/
164      Accelerates atomic velocities, rv, by half the time step.
165  /*-----*/
166      int n,k;
167      for (n=0; n<nAtom; n++)
168          for (k=0; k<3; k++) rv[n][k] = rv[n][k] + DeltaTH*ra[n][k];
169  }
170

```

Line 1, Column 1 Tab Size: 4 C

Figure 3: Euler scheme implementation

3 Velocity Autocorrelation Function

Here, we calculate the velocity autocorrelation function

$$Z(t) = \frac{\langle \vec{v}_i(t + t_0) \cdot \vec{v}_i(t_0) \rangle}{\langle \vec{v}_i(t_0) \cdot \vec{v}_i(t_0) \rangle} = \frac{\sum_{t_0} \sum_{i=1}^N \vec{v}_i(t + t_0) \cdot \vec{v}_i(t_0)}{\sum_{t_0} \sum_{i=1}^N \vec{v}_i(t_0) \cdot \vec{v}_i(t_0)} \quad (6)$$

where $\vec{v}_i(t)$ is the velocity of the i -th atom at time t . The bracket denotes averages over atoms, i , and the time origin, t_0 .

The `md.c` program from the course website was modified to calculate autocorrelation function. In summary, the `calc_vac` function was added to calculate the numerator term of Eqn (6), and some lines were added in the main function— variable `denm` to accumulate the sums in the denominator term and an outer loop to run the simulation `NSAMPLE` times and take averages. The required variable were added in the `md.h` header file, but are not shown here as these additions were not significant. The modified part of the `md.c` is shown below.

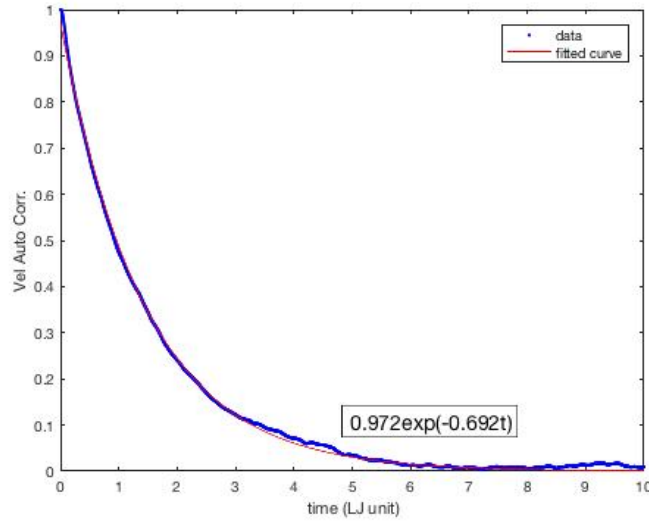
```

1  |/*****
2  |Molecular dynamics (MD) simulation with the Lennard-Jones potential.
3  |
4  |USAGE
5  |
6  |% cc -o md md.c -lm
7  |% ./md < md.in (see md.h for the input-file format)
8  |*****/
9  |#include <stdio.h>
10 |#include <math.h>
11 |#include "md.h"
12 |#include <time.h>
13 |
14 |int main(int argc, char **argv) {
15 |    double cpu, cpu1, cpu2; //for measuring running time of the program
16 |
17 |    double denm=0;
18 |    int outer,n,k;
19 |
20 |    InitParams();
21 |    InitConf();
22 |    ComputeAccel(); /* Computes initial accelerations */
23 |    cpu1 = ((double) clock())/CLOCKS_PER_SEC;
24 |
25 |    for(outer=1; outer<=NSAMPLE; outer++){
26 |        for(stepCount=1; stepCount<=StepLimit; stepCount++) {
27 |
28 |            SingleStep();
29 |
30 |            // VAC MODIFICATION
31 |            if(stepCount==1){
32 |                for (n=0; n<nAtom; n++){
33 |                    for (k=0; k<3; k++){
34 |                        v0[n][k] = rv[n][k];
35 |                        denm += v0[n][k]*v0[n][k];
36 |                    }
37 |                }
38 |            }
39 |
40 |            calc_vac(stepCount);
41 |            if (stepCount%StepAvg == 0) EvalProps();
42 |        }
43 |    }
44 |
45 |    /* Normalize autocorrelation by the square of v0 */
46 |    for(stepCount=1; stepCount<=StepLimit; stepCount++) {
47 |        vac[stepCount] = vac[stepCount]/denm;
48 |        printf("%9.6f %9.6f\n", stepCount*DeltaT, vac[stepCount]);
49 |    }
50 |
51 |    cpu2 = ((double) clock())/CLOCKS_PER_SEC;
52 |    cpu = cpu2-cpu1;
53 |    //printf("%le %le %le\n",cpu1, cpu2, cpu);
54 |    return 0;
55 |}
56 |
57 |// VAC MODIFICATION
58 |void calc_vac(int step){
59 |    int n,k;
60 |
61 |    for (n=0; n<nAtom; n++) //time summation over atoms
62 |        for (k=0; k<3; k++) //dot product
63 |            vac[step] += rv[n][k] * v0[n][k];
64 |}
65 |

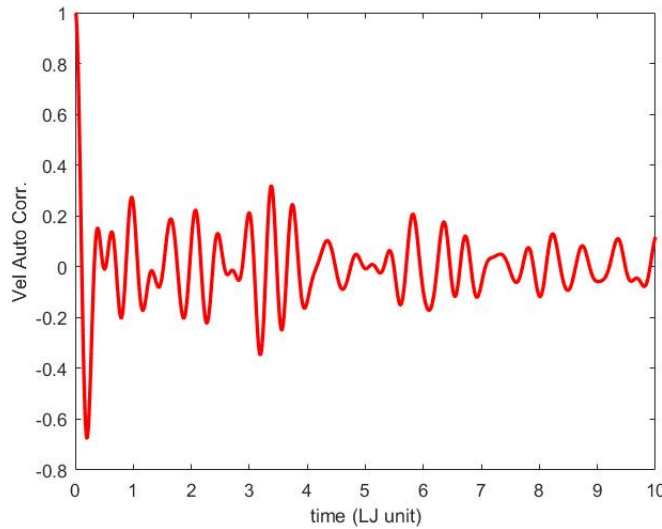
```

Figure 4: Modified Code block

The program was run using $\text{InitUcell} = 3, 3, 3$, $\text{DeltaT} = 0.005$ and $\text{StepAvg} = 10$, for $\text{StepLimit} = 2000$ steps to calculate VAC in the time range of $[0, \text{tmax} = \text{DeltaT} * \text{StepLimit} = 0.005 * 2000 = 10.0]$ for both gas phase ($\text{Density} = 0.1$, $\text{InitTemp} = 1.0$) and solid phase ($\text{Density} = 1.0$, $\text{InitTemp} = 0.1$). As The results are shown below:



(a) Velocity autocorrelation vs time for gas phase



(b) Velocity autocorrelation vs. time for solid phase

As expected, vac for gas phase dies down from 1 to 0 exponentially while for the the solid phase it just fluctuates about zero.

4 Split Operator Formalism- Deriving the Velocity Verlet algorithm

Consider a conservative system of N atoms which can translate along x , y and z axes. Hence, the total degrees of freedom of the system are

$$f = 3N \quad (7)$$

Let x denote the position, p momentum, and V the potential energy of the system. Then, the Hamiltonian of the system is given by

$$H = \sum_{j=1}^N \frac{p_j^2}{2m} + V(x_1, x_2, \dots, x_f) \quad (8)$$

Differentiating the above equation to get \dot{x}_j and \dot{p}_j , and noting that the system is conservative,

$$\dot{x}_j = \frac{\partial H}{\partial p_j} = \frac{p_j}{m} \quad (9)$$

$$\dot{p}_j = -\frac{\partial H}{\partial x_j} = -\frac{\partial V}{\partial x_j} = F_j \quad (10)$$

Now, for any function of the form $\phi(x_1, x_2, \dots, x_f, p_1, p_2, \dots, p_f)$,

$$\phi = \left(\sum_{j=1}^f \dot{x}_j \frac{\partial}{\partial x_j} + \dot{p}_j \frac{\partial}{\partial p_j} \right) \phi \quad (11)$$

$$= \underbrace{\left(\sum_{j=1}^f \frac{\partial H}{\partial p_j} \frac{\partial}{\partial x_j} + \frac{\partial H}{\partial x_j} \frac{\partial}{\partial p_j} \right)}_{iL} \phi \quad (12)$$

$$= iL\phi \quad (13)$$

where L is defined as the Liouville operator.

Now, consider a special function $\Gamma(x_j, p_j)$ such that,

$$\frac{d\Gamma}{dt} = iL\Gamma \quad (14)$$

This equation integrated to get Γ .

$$\Gamma(t) = \Gamma(0)e^{iLt} \quad (15)$$

This can be proved as follows. Take the Taylor expansion of the exponential term as follows:

$$e^{iLt} = 1 + iLt - \frac{L^2 t^2}{2!} - \frac{iL^3 t^3}{3!} + \dots \quad (16)$$

Taking the derivative w.r.t t ,

$$\frac{de^{iLt}}{dt} = iL - L^2t - \frac{iL^3t^2}{2} + \dots \quad (17)$$

$$= iL \left(1 + iLt - \frac{L^2t^2}{2!} + \dots \right) \quad (18)$$

$$= iLe^{iLt} \quad (19)$$

$$\times \Gamma(0), \quad \frac{d(\Gamma(0)e^{iLt})}{dt} = \Gamma(0)iLe^{iLt} \quad (20)$$

$$\therefore \frac{d\Gamma}{dt} = iL\Gamma \quad (21)$$

Let us define the classical propagator as

$$U(t) = e^{iLt} \quad (22)$$

Note that $U(t)$ is unitary, i.e., $U(-t) = U(t)$.

Let us now decompose the Liouville operator as follows,

$$iL = iL_1 + iL_2 \quad (23)$$

$$= \underbrace{\sum_j F_j(x_j) \frac{\partial}{\partial p_j}}_{iL_1} + \underbrace{\sum_j \frac{p_j}{m} \frac{\partial}{\partial x_j}}_{iL_2} \quad (24)$$

Now, we can write the exponential of the above decomposition using Trotter expansion as

$$e^{i(L_1+L_2)t} = (e^{\frac{iL_1\Delta t}{2}} e^{iL_2\Delta t} e^{\frac{iL_1\Delta t}{2}})^P \quad (25)$$

where $\Delta t = t/P$. This allows us to define a discrete time propagator as

$$G(\Delta t) = U_1\left(\frac{\Delta t}{2}\right) U_2(\Delta t) U_1\left(\frac{\Delta t}{2}\right) \quad (26)$$

$$= e^{\frac{iL_1\Delta t}{2}} e^{iL_2\Delta t} e^{\frac{iL_1\Delta t}{2}} \quad (27)$$

Since we know that $U(t)$ is unitary, it means that $G(\Delta t)$ will also be unitary. Now, let us substitute the definition of the split operators from Eqn (24).

$$G(\Delta t) = e^{\Delta t/2 F(x) \frac{\partial}{\partial p}} e^{\Delta t \dot{x} \frac{\partial}{\partial x}} e^{\Delta t/2 F(x) \frac{\partial}{\partial p}} \quad (28)$$

Operating this on the $\{x(0), p(0)\}$ and using the property for any operator of the form $e^{c \frac{\partial}{\partial q}}$

$$e^{c \frac{\partial}{\partial q}} f(q) = f(q + c) \quad (29)$$

where c is independent of q (*I don't know how to derive this!*).

Using this, we can derive that

$$x(\Delta t) = x(0) + \Delta t \dot{x}(0) + \frac{(\Delta t)^2}{2} \frac{F[x(0)]}{m} \quad (30)$$

$$\dot{x}(\Delta t) = \dot{x}(0) + \frac{\Delta t}{2m} \{F[x(0)] + F[x(\Delta t)]\} \quad (31)$$

which can be written in the known form on the velocity-Verlet algorithm

$$\dot{x}\left(\frac{\Delta t}{2}\right) = \dot{x}(0) + \frac{\Delta t}{2m}F[x(0)] \quad (32)$$

$$x(\Delta t) = x(0) + \Delta t\dot{x}\left(\frac{\Delta t}{2}\right) \quad (33)$$

$$\dot{x}(\Delta t) = \dot{x}\left(\frac{\Delta t}{2}\right) + \frac{\Delta t}{2m}F[x(\Delta t)] \quad (34)$$