Chemistry 380.37
Fall 2015
Dr. Jean M. Standard
November 9, 2015

The Verlet Algorithm for Molecular Dynamics Simulations

Equations of motion

For a many-body system consisting of *N* particles, Newton's classical equations of motion are

$$\vec{F}_i = m_i \vec{a}_i , \quad i = 1,2,\ldots,N \quad . \tag{1}$$

In this equation, $\vec{F}_i$ denotes the force on the *i*th particle, m_i is the mass of the particle, and $\vec{a}_i$ is the acceleration. The arrows over the force and acceleration denote the *x*, *y*, and *z* components of the vectors. In the rest of this handout, the vector notation will be dropped for convenience.

Taylor Series Approximation

The solution of Newton's equations using the Verlet algorithm is based on a Taylor series expansion. Expanding the position of the *i*th particle r_i at time *t*+*h* and *t*–*h* in Taylor series yields,

$$\begin{aligned} r_i(t+h) &= r_i(t) + h\, r_i'(t) + \frac{h^2}{2!}\, r_i''(t) + \frac{h^2}{3!}\, r_i'''(t) + O(h^4) \\ r_i(t-h) &= r_i(t) - h\, r_i'(t) + \frac{h^2}{2!}\, r_i''(t) - \frac{h^3}{3!}\, r_i'''(t) + O(h^4) \; . \end{aligned} \tag{2}$$

Adding these two equations leads to the result

$$r_i(t+h) + r_i(t-h) = 2\, r_i(t) + h^2\, r_i''(t) + O(h^4) \; . \tag{3}$$

Note that $r_i'(t)$ is equivalent to the velocity of the particle, $v_i(t)$, and $r_i''(t)$ is the acceleration of the particle, $a_i(t)$. Making these substitutions in Eq. (3), we have

$$r_i(t+h) + r_i(t-h) = 2\, r_i(t) + h^2\, a_i(t) + O(h^4) \; . \tag{4}$$

Solving for $r_i(t+h)$ and using Newton's equations for the acceleration ($a_i = F_i / m_i$) leads to

$$r_i(t+h) = 2\, r_i(t) - r_i(t-h) + \frac{h^2}{m_i}\, F_i(t) + O(h^4) \; . \tag{5}$$

Discretization of Time

The time is a continuous variable, but to implement Eq. (5) on a computer requires the solution to be generated for discrete time steps such that

$$t_{n+1} - t_n = h \; , \tag{6}$$

where *h* is the step size for the time variable.

Discretizing time in Eq. (5) leads to the **Verlet algorithm (1967),**

$$r_i(t_{n+1}) = 2r_i(t_n) - r_i(t_{n-1}) + \frac{h^2}{m_i} F_i(t_n) + O(h^4) . \tag{7}$$

Note that the Verlet Algorithm relies on two previous time steps, t_n and t_{n-1}, to advance the solution forward in time, and it is a fourth-order method. There are sometimes problems associated with the Verlet algorithm due to potential loss of accuracy when Eq. (7) is implemented. The final term in the equation tends to be small compared to the others, and this term may as a result get lost in the round-off error.

Notice that the velocity does not explicitly appear in the Verlet algorithm. This may be a problem if properties of the system that depend on velocity, such as kinetic energy, are desired. Also, the velocity at time t_n cannot be calculated until the position at time t_{n+1} is obtained. As a result the velocity is usually calculated from the positions using an approximate formula based on the derivative of the position,

$$v_i(t_n) = \frac{r_i(t_{n+1}) - r_i(t_{n-1})}{2h} . \tag{8}$$

Also notice that the Verlet algorithm is not self-starting, since it requires the positions at two previous times (t_n and t_{n-1}) to get the position at the next time, t_{n+1}. This is usually overcome by first completing one step of a less accurate method that only requires information at one previous time step, such as the Euler method, and then switching to the Verlet method for subsequent steps.

The Verlet Leapfrog Algorithm

In an attempt to correct some of the problems associated with the original Verlet algorithm, the Verlet Leapfrog algorithm was developed. Using an approximation for the derivative, the velocity at the midpoint between times t_n and t_{n+1} can be defined as,

$$v_i(t_{n+1/2}) = \frac{r_i(t_{n+1}) - r_i(t_n)}{h} , \tag{9}$$

Eq. (8) can be solved for $r_i(t_{n+1})$ to yield

$$\boxed{r_i(t_{n+1}) = r_i(t_n) + h\, v_i(t_{n+1/2}) .} \tag{10}$$

Similarly, defining the velocity at the midpoint between times t_{n-1} and t_n gives

$$v_i(t_{n-1/2}) = \frac{r_i(t_n) - r_i(t_{n-1})}{h} . \tag{11}$$

The acceleration at time t_n can be defined using another approximate formula for the derivative,

$$a_i(t_n) = \frac{v_i(t_{n+1/2}) - v_i(t_{n-1/2})}{h} . \tag{12}$$

Using Newton's equations for the acceleration, $a_i(t_n) = F_i(t_n)/m_i$, and equating the acceleration to Eq. (12) yields

$$\frac{v_i(t_{n+1/2}) - v_i(t_{n-1/2})}{h} = \frac{F_i(t_n)}{m_i}, \tag{13}$$

which can be solved for $v_i(t_{n+1/2})$,

$$\boxed{v_i(t_{n+1/2}) = v_i(t_{n-1/2}) + \frac{h}{m_i} F_i(t_n).} \tag{14}$$

Eqs. (10) and (14) constitute the equations of the **Verlet Leapfrog algorithm**.

The Leapfrog algorithm has some advantages over the original Verlet algorithm, Eq. (7). First, the loss of accuracy due to round-off error in the original Verlet algorithm because of vastly different magnitudes of terms in the equation is corrected. Second, the velocities are included explicitly in the method, unlike the original Verlet algorithm.

There are still a few problems associated with the Leapfrog algorithm, however. First, it is still not self-starting. This can be overcome as in the original Verlet method by completing one step of the Euler method first, and then switching to the Leapfrog algorithm for subsequent steps.

Though the Leapfrog algorithm produces both positions and velocities, the velocities are obtained at 1/2-integer time steps ($t_{n+1/2}, t_{n+3/2}, t_{n+5/2}, ...$), while the positions are obtained at integer time steps ($t_n, t_{n+1}, t_{n+2}, ...$). If the velocities at integer time steps are required, an auxiliary equation may be utilized which simply averages the 1/2-integer velocities,

$$v_i(t_n) = \frac{1}{2}\left[v_i(t_{n-1/2}) + v_i(t_{n+1/2})\right]. \tag{15}$$

Example: Verlet Leapfrog Algorithm for Bond Stretching

Consider the simple one-dimensional harmonic potential as a representation of the motion of the vibrations of a bond in a molecular system,

$$V(x) = \tfrac{1}{2} k x^2. \tag{16}$$

The coordinate *x* represents the bond displacement, $x = r - r_{eq}$. The Verlet leapfrog algorithm for this system is given by

$$\begin{aligned} x(t_{n+1}) &= x(t_n) + h v_x(t_{n+1/2}) \\ v_x(t_{n+1/2}) &= v_x(t_{n-1/2}) + \frac{h}{\mu} F_x(t_n). \end{aligned} \tag{17}$$

The force F_x is obtained from the force field,

$$\begin{aligned} F_x &= -\frac{dV}{dx} \\ F_x &= -kx. \end{aligned} \tag{18}$$

Substituting, the Verlet Leapfrog equations become

$$\begin{aligned} x(t_{n+1}) &= x(t_n) + hv_x(t_{n+1/2}) \\ v_x(t_{n+1/2}) &= v_x(t_{n-1/2}) - \frac{hk}{m}x(t_n) . \end{aligned} \tag{19}$$

Example: Carbon-Carbon Bond Stretching Using the Verlet Leapfrog Algorithm

Modeling Carbon-Carbon Bonds

To apply the Verlet Leapfrog algorithm (Eq. 19) to model the motion of a carbon-carbon bond, then the mass m would be replaced by the reduced mass μ of the bond. For a bond A-B, the reduced mass μ is defined as

$$\mu = \frac{m_A m_B}{m_A + m_B} . \tag{20}$$

where m_A and m_B are the masses of atoms A and B. So, for a carbon-carbon bond, the reduced mass is 6.0 g/mol (or about 10940 atomic units).

The force constants for carbon-carbon bonds from the on-line CRC (Table 9-75) are listed in Table 1 in units of kcal $mol^{-1}Å^{-2}$, N/cm, N/m (or J/m^2), and atomic units ($hartree/bohr^2$).

Table 1. Force constants for typical carbon-carbon bonds.

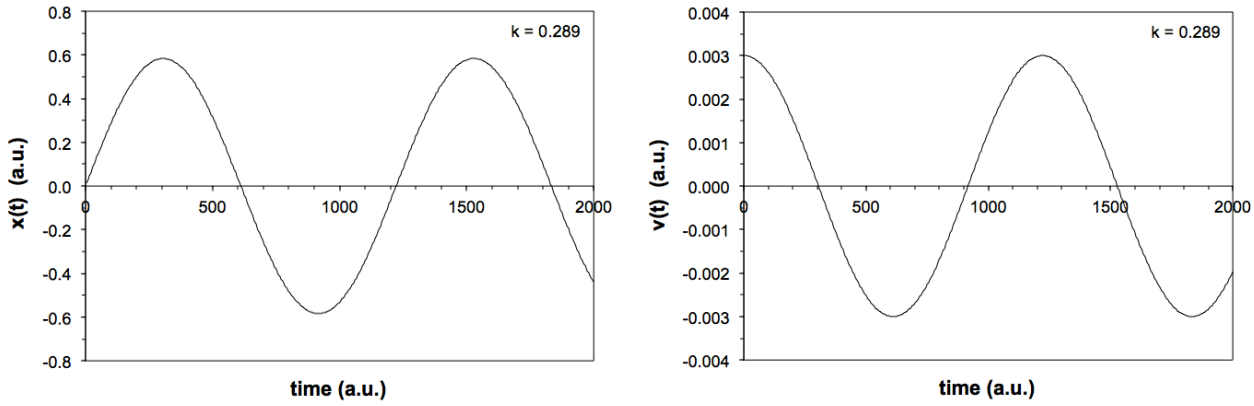
Bond	Force Constant (kcal $mol^{-1}Å^{-2}$)	Force Constant (N/cm)	Force Constant (N/m)	Force Constant (a.u.)
C−C (from C_2H_6)	648	4.50	450	0.289
C=C (from CCl_2CH_2)	1213	8.43	843	0.541
C≡C (from C_2)	1750	12.16	1216	0.781

Results: Variation of Force Constants

Calculations were performed using the three force constants given in Table 1 (k=0.289, 0.541, and 0.781 a.u.), representing carbon-carbon single, double, and triple bonds, respectively. The Verlet Leapfrog Algorithm was implemented using a value of 5 a.u. for h and 500 total time steps with an initial position of 0 a.u. and an initial velocity of 0.003 a.u. [Note that an atomic unit of time is 2.419×10^{-17} s, or 0.024 fs. An atomic unit of distance (one bohr) equals 0.5291771 Å. An atomic unit of velocity equals 2.188×10^{5} m/s.]

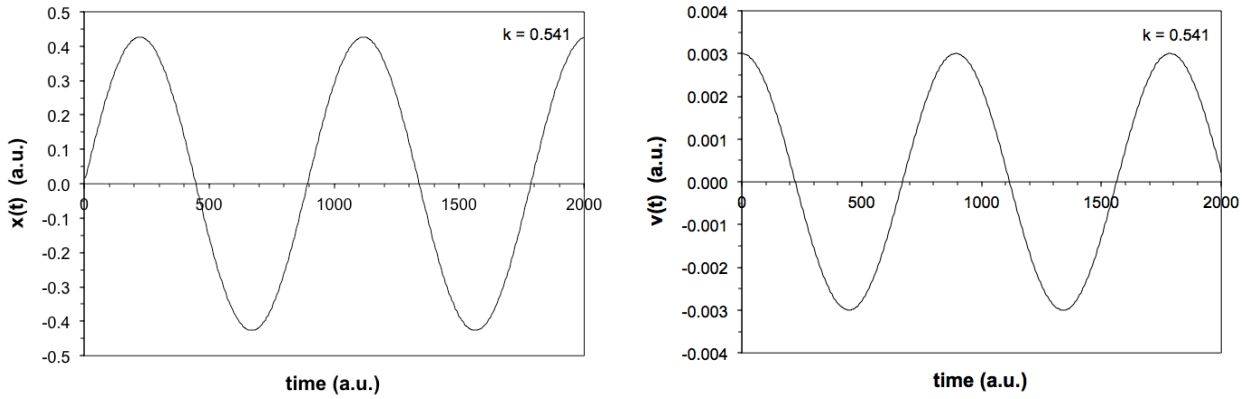
The position and velocity values are plotted as functions of time to demonstrate the dynamics of a carbon-carbon bond modeled by each of the force constants in Figures 1-3. Note that in each case the position oscillates symmetrically about x=0, a characteristic of the harmonic oscillator. In addition, we see that the velocity is out of phase with the position (and it also oscillates symmetrically about v=0).

The figures also show that as the force constant increases, the frequency of oscillation of the C-C bond increases, as would be expected as a result of the relationship between the harmonic frequency and force constant. The period of oscillation therefore decreases as the force constant increases, from 0.030 ps to 0.022 ps to 0.018 ps for force constants of 0.289, 0.541, and 0.781 a.u., respectively.



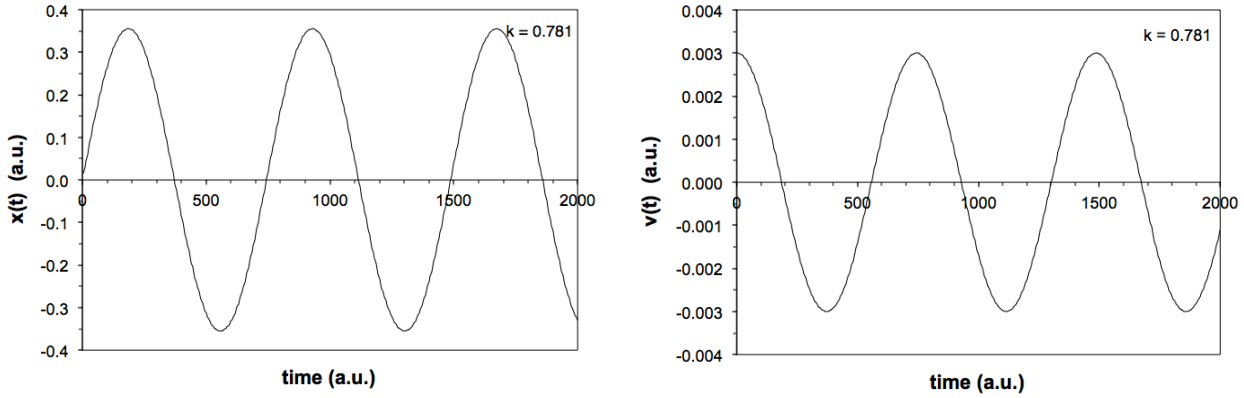
(a) **(b)**

Figure 1. (a) Bond displacement and (b) velocity as functions of time for a model carbon-carbon single bond with k=0.289 a.u.



(a) **(b)**

Figure 2. (a) Bond displacement and (b) velocity as functions of time for a model carbon-carbon double bond with k=0.541 a.u.



(a) **(b)**

Figure 3. (a) Bond displacement and (b) velocity as functions of time for a model carbon-carbon triple bond with k=0.781 a.u.

Testing the Accuracy of the Verlet Algorithm

The exact solution for the position at each time step for the harmonic oscillator may be compared to the numerical solution from the Verlet Algorithm.

Comparisons were made with the following time steps and number of points:

h (a.u.):	10.	5.	2.
# steps:	200	400	1000

The initial conditions used for each run were $x_0 = 0.0$ a.u. and $v_0 = 0.003$ a.u. The absolute error between the Verlet and exact results for the bond displacement coordinate *x* were plotted as functions of time. Results for each of the force constants are shown in Figures 4-6.

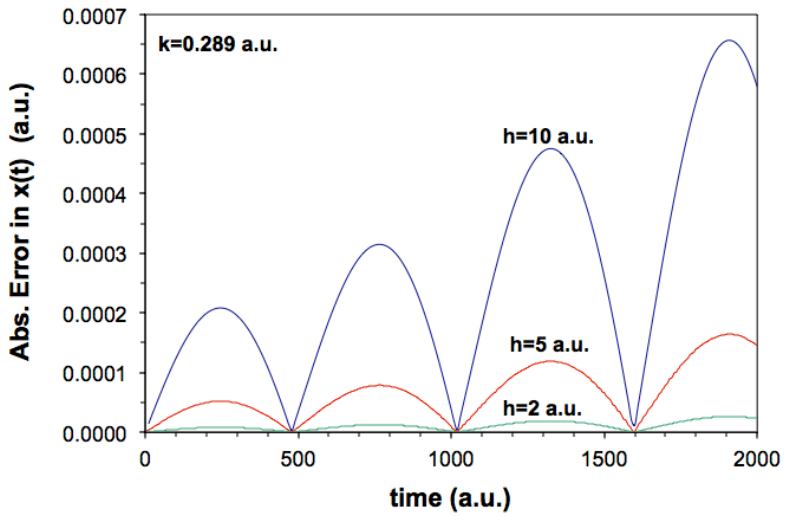


Figure 4. Absolute error in position as a function of time for a model carbon-carbon double bond with *k*=0.289 a.u.

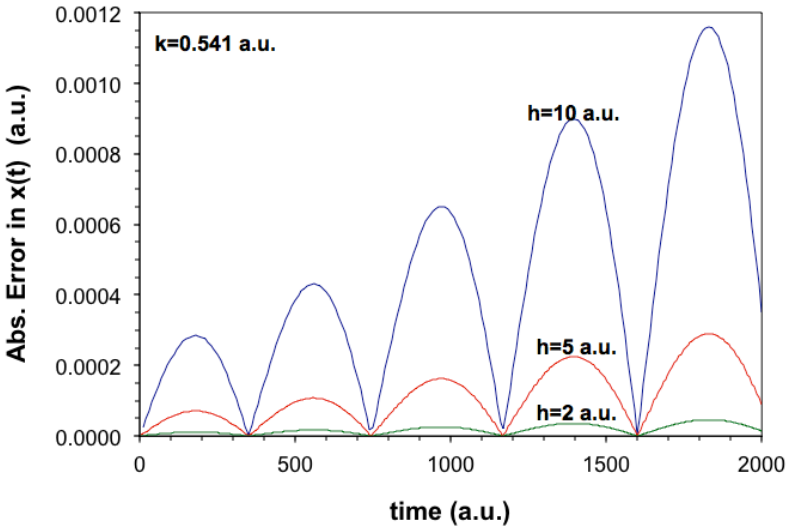


Figure 5. Absolute error in position as a function of time for a model carbon-carbon double bond with *k*=0.541 a.u.

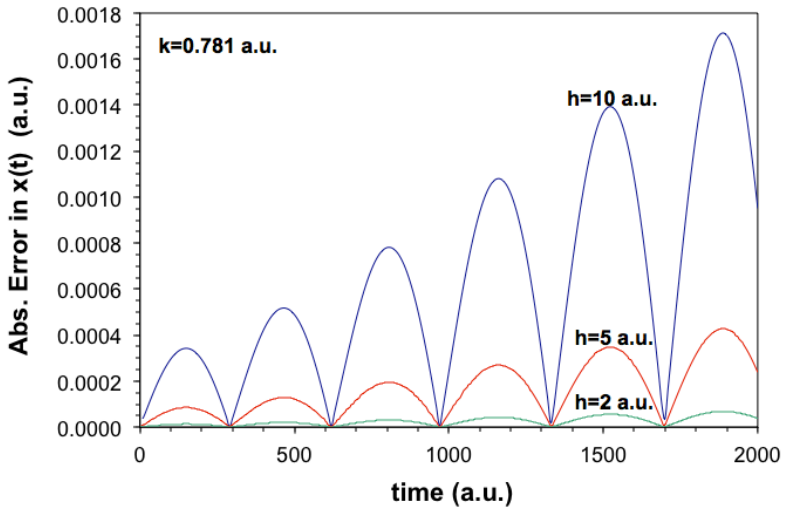


Figure 6. Absolute error in position as a function of time for a model carbon-carbon double bond with k=0.789 a.u.

The maximum absolute error in the position shown in each of Figures 4-6 clearly increases over time. It is also cyclic, since we can see that the largest error occurs at the maximum or minimum extension of the bond. For each time step, the maximum absolute error over one oscillation cycle can be compared. Results are given in Table 2.

Table 2. Maximum absolute error for different time steps and force constants.

	Maximum Absolute Error (a.u.)		
h (a.u.)	k = 0.289 a.u.	k = 0.541 a.u.	k = 0.781 a.u.
10	3.148×10^{-4}	4.310×10^{-4}	5.176×10^{-4}
5	7.869×10^{-5}	1.077×10^{-4}	1.294×10^{-4}
2	1.259×10^{-5}	1.723×10^{-5}	2.070×10^{-5}

The results show that the absolute error decreases as the time step decreases. Similar trends are observed for each of the three force constants. Notice, however, that the decrease in the error is not linear. A graph showing the maximum absolute error plotted against the time step is shown in Figure 7.

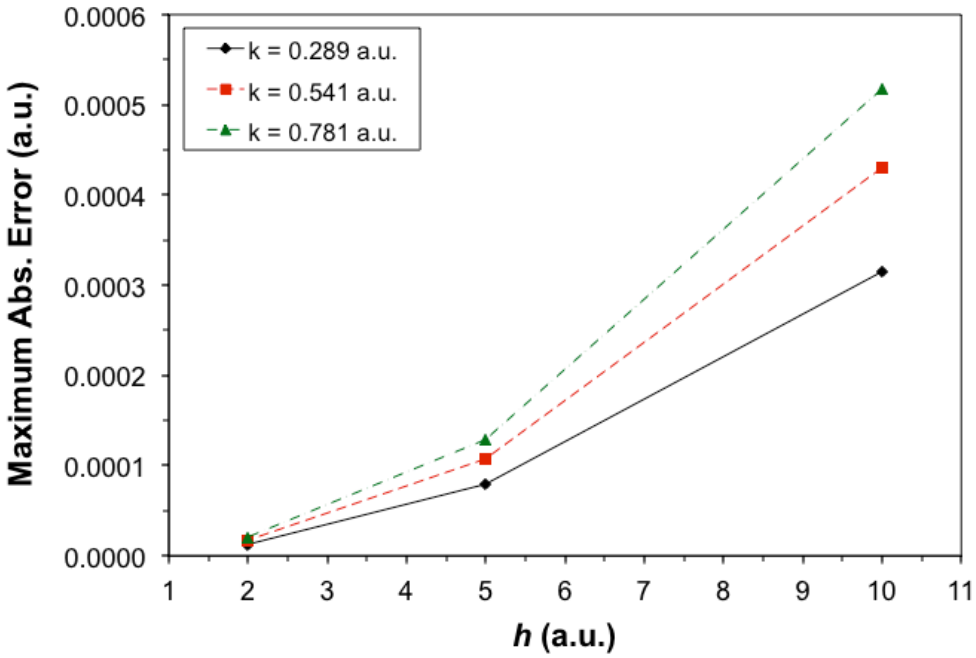


Figure 7. Maximum absolute error in position as a function of step size in time for model carbon-carbon bonds.

The results shown in Figure 7 appear roughly quadratic in form. Looking at the data, we see that when the step size decreases from 10 to 5 a.u. (a factor of 2), the error decreases by a factor of 4. Similarly, when the step size decreases from 10 to 2 a.u. (a factor of 5), the error decreases by a factor of 25.

The relationship between step size and error can be explored further. Assume that the error has the following form,

$$Error = C\,h^{M}\,, \tag{21}$$

where C is a constant and M is an integer. Taking the log of both sides yields,

$$\log\left(Error\right) = \log C + M\log\left(h\right)\,. \tag{22}$$

This equation indicates that a plot of log(*Error*) on the *y*-axis and log(*h*) on the x-axis should be linear with a slope of M. If the error scales quadratically with the step size, then the slope should be equal to 2. The log plot is shown in Figure 8.

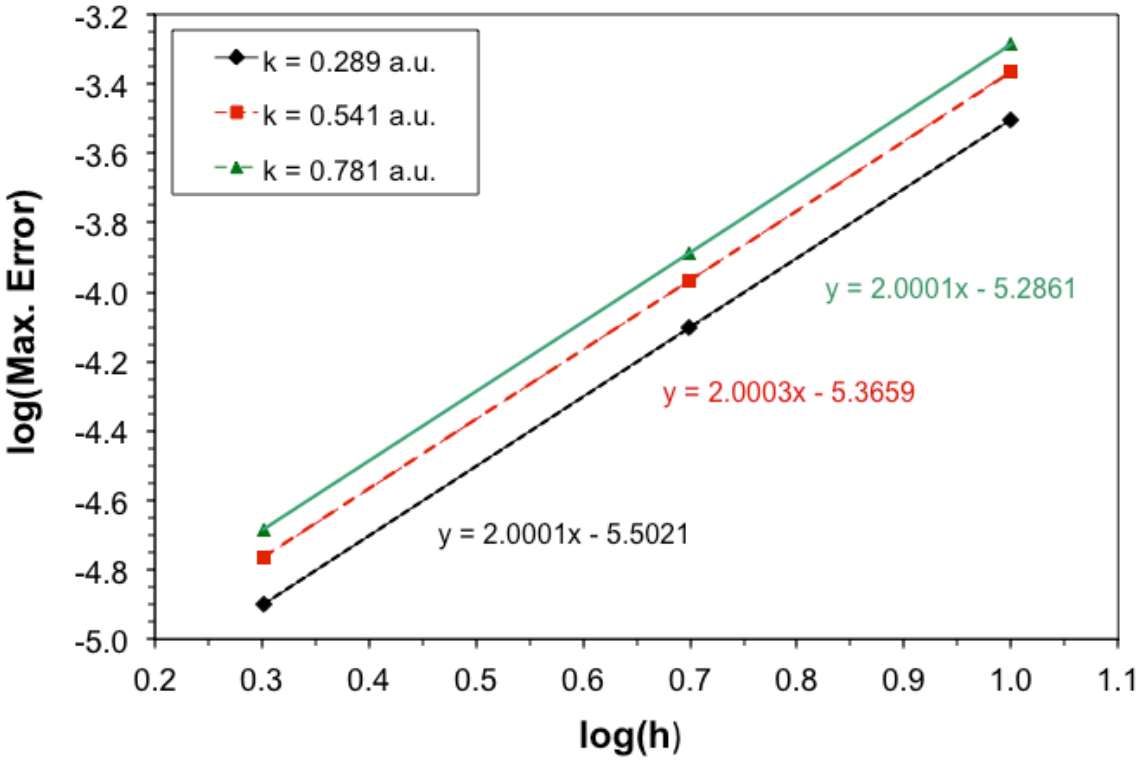


Figure 8. Log-log plot of maximum error in bond displacement and step size for Verlet Leapfrog algorithm.

As illustrated in Figure 8, the slope for all the k values does indeed equal 2. This verifies that when the step size is decreases by some factor S, then the error decreases by S^2.