Computational Methods - Lesson 4 (Luxembourg, 07.10.2020)

Differential equations II

Matteo Barborini*,[a], Valentin Vassilev Galindo*,[b], Prof. Alexandre Tkatchenko*

Theoretical Chemical Physics group

Faculté des Sciences, des Technologies et de Médecine - Département Physique et sciences des matériaux

*Campus Limpertsberg, Université du Luxembourg - 162 A, avenue de la Faïencerie, L-1511 Luxembourg

[a]matteo.barborini@uni.lu, [b]valentin.vassilev@uni.lu

1. Higher order algorithms

In this second lesson we will discuss about more accurate algorithms to solve ordinary differential equations. In particular, we will consider the Verlet and the velocity-Verlet algorithms, followed by the predictor-corrected algorithm and the Runge-Kutta one of the second and fourth order.

1.1. Verlet and Velocity Verlet.

As seen in the previous lesson, the time evolution at time t_{n+1}, through Taylor's expansion of the displacement function (around the previous step), leads to the equation

$$x(t_{n+1}) = x(t_n) + v(t_n)\Delta t + \frac{1}{2}a(t_n)\Delta t^2 + O(\Delta t^3). \tag{1}$$

At the same time we can define the evolution *backwards* in time (through Taylor expansion still around t_n) as:

$$x(t_{n-1}) = x(t_n) - v(t_n)\Delta t + \frac{1}{2}a(t_n)\Delta t^2 + O(\Delta t^3). \tag{2}$$

Summing, and subtracting, the two equations we get the coupled equations

$$\begin{cases} x(t_{n+1}) + x(t_{n-1}) = 2x(t_n) + a(t_n)\Delta t^2 + O(\Delta t^4) \\ x(t_{n+1}) - x(t_{n-1}) = 2v(t_n)\Delta t + O(\Delta t^3) \end{cases} \tag{3}$$

that can be rewritten as

$$\begin{cases} x(t_{n+1}) = 2x(t_n) - x(t_{n-1}) + a(t_n)\Delta t^2 + O(\Delta t^4) \\ v(t_n) = \frac{x(t_{n+1}) - x(t_{n-1})}{2\Delta t} + O(\Delta t^3) \end{cases} . \tag{4}$$

Notice that the expression of the velocity is not used directly to compute the variation of the displacement. Moreover the algorithm is not *self-starting*, as the leapfrog algorithm since with the initial conditions x_0, v_0, but we need to obtain the value of $x(t_1)$ from for example a first order step with the Euler algorithm. Finally, from the two equations we can see that the accuracty of the Verlet algorithm is of order $O(\Delta t^3)$ in the displacement and of order $O(\Delta t^2)$ in the velocity.

It is important to underline that, since the velocity is computed as the difference of two positions, as the value Δt becomes too small, the all ratio might become large, leading to computational rounding errors that when accumulating can invalidate the full integration.

To partially attenuate this issue, we can construct the velocity-Verlet algorithm. To construct this algorithm, let us start from the Verlet algorithm adding and subtracting the quantity $\frac{x(t_{n+1})}{2}$ from the right side of the equation of the displacement:

$$\begin{aligned} x(t_{n+1}) &= 2x(t_n) - x(t_{n-1}) + a(t_n)\Delta t^2 + \frac{x(t_{n+1})}{2} - \frac{x(t_{n+1})}{2} = \\ &= x(t_n) + \frac{x(t_{n+1}) - x(t_{n-1})}{2} - \frac{1}{2}\left[x(t_{n+1}) - 2x(t_n) + x(t_{n-1})\right] + a(t_n)\Delta t^2. \end{aligned} \tag{5}$$

Considering the second Verlet equation, related to the velocity, we have that the second term in the right side of this last equation is actually equal to $v(t_n)\Delta t = \frac{x(t_{n+1}) - x(t_{n-1})}{2}$; moreover, the third term is actually related to the approximation of the second order derivative of the position with respect to time[1], $a(t_n)\Delta t^2 =$

[1]We did not discuss second order derivatives in the first lesson regarding differentiation. Yet we can understand that if we can approximate the calculation of the velocity as the first order derivative $v(t_n) = \frac{x(t_{n+1}) - x(t_n)}{\Delta t}$ then we can compute the acceleration as the first order derivative of the velocity $a(t_n) = \frac{v(t_n) - v(t_{n-1})}{\Delta t}$. If we substitute in this last equation, the approximate values of the velocities in the two times t_n and t_{n+1} we obtain $a(t_n) = \frac{\frac{x(t_{n+1}) - x(t_n)}{\Delta t} - \frac{x(t_n) - x(t_{n-1})}{\Delta t}}{\Delta t}$ which is equal to $a(t_n) = \frac{x(t_{n+1}) - 2x(t_n) + x(t_{n-1})}{\Delta t^2}$

$x(t_{n+1}) - 2x(t_n) + x(t_{n-1})$, so that the final equation reduces to the second order Taylor expansion of the position around t_n:

$$x(t_{n+1}) = x(t_n) + v(t_n)\Delta t + \frac{1}{2}a(t_n)\Delta t^2. \tag{6}$$

To obtain the new equation for the evolution of the velocity, let us consider the evolution of the velocity of the original Verlet algorithm at time t_{n+1}:

$$v(t_{n+1}) = \frac{x(t_{n+2}) - x(t_n)}{2\Delta t}. \tag{7}$$

If we substitute in this equation the value of $x(t_{n+2})$ derived by the first equation of the original Verlet algorithm, $x(t_{n+2}) = 2x(t_{n+1}) - x(t_n) + a(t_{n+1})\Delta t^2$, we have that

$$v(t_{n+1}) = \frac{2x(t_{n+1}) - x(t_n) + a(t_{n+1})\Delta t^2 - x(t_n)}{2\Delta t} = \frac{x(t_{n+1}) - x(t_n)}{\Delta t} + \frac{a(t_{n+1})}{2}\Delta t.$$

Still, since we want to write the value of $v(t_{n+1})$ as a function of $v(t_n)$, we again consider the first equation of the Verlet algorithm, and explicit the value of $x(t_n) = \frac{x(t_{n+1})+x(t_{n-1})}{2} - \frac{a(t_n)}{2}\Delta t^2$.
By substituting this equation into the one of the velocity above, we have

$$\begin{aligned} v(t_{n+1}) = \frac{x(t_{n+1}) - \frac{x(t_{n+1})+x(t_{n-1})}{2} + \frac{a(t_n)}{2}\Delta t^2}{\Delta t} + \frac{a(t_{n+1})}{2}\Delta t = \\ = \frac{x(t_{n+1}) - x(t_{n-1})}{2\Delta t} + \frac{a(t_n)}{2}\Delta t + \frac{a(t_{n+1})}{2}\Delta t \end{aligned} \tag{8}$$

where the first term is the velocity $v(t_n)$ computed with the central difference method so that we finally obtain

$$v(t_{n+1}) = v(t_n) + \frac{a(t_n) + a(t_{n+1})}{2}\Delta t. \tag{9}$$

The final Velocity-Verlet algorithm will thus be given by

$$\begin{cases} x(t_{n+1}) = x(t_n) + v(t_n)\Delta t + \frac{1}{2}a(t_n)\Delta t^2 \\ v(t_{n+1}) = v(t_n) + \frac{a(t_n)+a(t_{n+1})}{2}\Delta t \end{cases} \tag{10}$$

which, differently from the simple Verlet, is self-starting, in fact (x_0, v_0) are sufficient to start the dynamical evolution of the system.

1.2. Predictor-corrector. The predictor-corrector alogorithms are a family of algorithms for which a first estimate of the quantities that have to be integrated, is used to build the true numerical integration step. In the simplest version of these methods the prediction of the displacement from x_{n-1} is written as

$$x^{(p)}(t_{n+1}) = x(t_{n-1}) + 2v(t_n)\Delta t. \tag{11}$$

From this predicted position, we can compute the predicted acceleration $\phi^{(p)}(t_{n+1}) = \phi(x^{(p)}(t_{n+1}))$ at time t_{n+1} which is different from the actual one $\phi(x(t_{n+1}))$, since $x^{(p)}(t_{n+1}) \neq x(t_{n-1})$. Using this predicted value, it is possible to compute the velocity at time t_{n+1} as

$$v(t_{n+1}) = v(t_n) + \frac{\phi^{(p)}(t_{n+1}) + \phi(t_n)}{2}\Delta t \tag{12}$$

and the position will be given for example through the midpoint estimator

$$x(t_{n+1}) = x(t_n) + \frac{v(t_{n+1}) + v(t_n)}{2}\Delta t. \tag{13}$$

Thus, the final predictor-corrector approach can be summarised with the following system of equations

$$\begin{cases} x^{(p)}(t_{n+1}) = x(t_{n-1}) + 2v(t_n)\Delta t \\ v(t_{n+1}) = v(t_n) + \frac{\phi^{(p)}(t_{n+1})+\phi(t_n)}{2}\Delta t \\ x(t_{n+1}) = x(t_n) + \frac{v(t_{n+1})+v(t_n)}{2}\Delta t \end{cases}, \tag{14}$$

that is not self-starting since it needs the position at time t_{n-1}. Various ways to initialize this first step can be proposed. Again, one possible way is to use Euler at t_0 to compute an estimation of the initial predicted position as:

$$x^{(p)}(t_1) = x(t_0) + v(t_0)\Delta t. \tag{15}$$

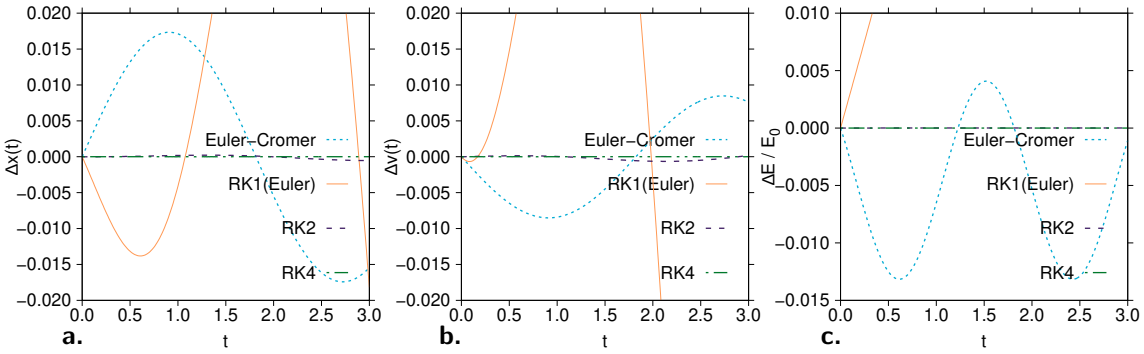

Figure 1: Error between the exact solution and the algorithmic propagation of the position **a** and of the velocity **b**, obtained with the Euler-Cromer, Euler (RK1), RK2 (Midpoint) and RK4 algorithms. **c.** Relative error of the energies $\Delta E/E_0$ for the same four algorithms with respect to the exact initial value that should be conserved.

1.3. Runge-Kutta methods of the second and fourth order. The Runge-Kutta methods are a family of algorithms used to solve ordinary differential equations of the first order (or second order in a system of dependent first order differential equations), with a chosen order of accuracy with respect to the discretization step. In order to derive the simplest of these algorithms, with an accuracy of the second order in the discretization step, let us consider for now an ordinary differential equation of the first order

$$\dot{x}(t) = f(x(t), t) \qquad x(t_0) = x_0. \tag{16}$$

In order to algorithmically integrate the solution $x(t)$ with respect to t, we define a time step Δt in which we divide our time evolution. We now define the function at the new time step $x(t + \Delta t)$ through the Taylor expansion of the solution at the second order in Δt:

$$x(t + \Delta t) = x(t) + \dot{x}(t)\Delta t + \frac{1}{2}\ddot{x}(t)\Delta t^2 + O(\Delta t^3). \tag{17}$$

From the differential equation, we know that the derivatives that appear in this expansion can be written in terms of the function $f(x(t), t)$ and of its derivatives, in fact we have that

$$\dot{x}(t) = f(x(t), t) \qquad \text{and} \qquad \ddot{x}(t) = \frac{\partial f(x(t), t)}{\partial t} + \frac{\partial f(x(t), t)}{\partial x(t)}\dot{x}(t), \tag{18}$$

that substituted in the expansion will give:

$$x(t + \Delta t) = x(t) + f(x(t), t)\Delta t + \frac{1}{2}\left[\frac{\partial f(x(t), t)}{\partial t} + \frac{\partial f(x(t), t)}{\partial x(t)} f(x(t), t)\right]\Delta t^2 + O(\Delta t^3). \tag{19}$$

The first two terms on the right side of this last equation represent Euler's propagation of the first order.
The idea behind the **Runge-Kutta** method is to find a general approach to rewrite this expansion (at various orders) not as a function of the derivatives of $f(x(t), t)$ but as a function of the variations of $f(x(t), t)$ in time. This is like saying that we want to rewrite the expansion as the combination of the components that come from the algorithmic differentiation of the function $f(x(t), t)$ with a time step Δt, with an accuracy that is of the same order of the truncation in the expansion (in this case $O(\Delta t^2)$).
Thus, in the **second order Runge-Kutta algorithm** this expansion will be rewritten as a combination of two terms

$$x(t + \Delta t) = x(t) + a_1 k_1 + a_2 k_2 + O(\Delta t^3), \tag{20}$$

where a_1 and a_2 are coefficients defining the linear combination and k_1 and k_2 are 'slopes' of the function $f(x(t), t)$ at different times and positions. The general form that is used to define this slopes is at the second order

$$\begin{aligned} k_1 &= f(x(t), t)\Delta t \\ k_2 &= f(x(t) + q_x k_1, t + q_t \Delta t)\Delta t \end{aligned} \tag{21}$$

where q_x and q_t are coefficients that we need to find.
To find these parameters, together with the linear coefficients a_1 and a_2, we expand the function k_2 in Taylor series at the second order in Δt (accuracy that we want to reach), around the equilibrium $f(x(t),t)$, for which we have

$$k_2 = \Delta t \left[f(x(t),t) + q_x k_1 \frac{\partial f(x(t),t)}{\partial x(t)} + q_t \Delta t \frac{\partial f(x(t),t)}{\partial t} \right] + O(\Delta t^3). \tag{22}$$

By combining this expression with that of k_1 we can write the update as:

$$x(t+\Delta t) = x(t) + a_1 f(x(t),t)\Delta t + a_2 \Delta t \left[f(x(t),t) + \right. \\ \left. + q_x k_1 \frac{\partial f(x(t),t)}{\partial x(t)} + q_t \Delta t \frac{\partial f(x(t),t)}{\partial t} \right] + O(\Delta t^3), \tag{23}$$

that is

$$x(t+\Delta t) = x(t) + (a_1 + a_2)\, f(x(t),t)\Delta t + a_2 \Delta t^2 \left[q_x f(x(t),t) \frac{\partial f(x(t),t)}{\partial x(t)} + q_t \frac{\partial f(x(t),t)}{\partial t} \right] + O(\Delta t^3). \tag{24}$$

If we compare this last equation with the previous second order Taylor expansion of $x(t+\Delta t)$, we have that the parameters will have to satisfy the conditions

$$\begin{cases} a_1 + a_2 = 1 \\ a_2 q_x = \frac{1}{2} \\ a_2 q_t = \frac{1}{2} \end{cases} \tag{25}$$

that is a system without a unique solution.
In general three values of a_2 are used, $a_2 = \frac{1}{2}$, $a_2 = 1$ and $a_2 = \frac{2}{3}$, which correspond respectively to Heun's Method, the Midpoint Method and Ralston's method.
For *Heun's Method*, we have that $a_2 = \frac{1}{2}$ and $a_1 = \frac{1}{2}$ so $q_x = 1$ and $q_t = 1$ and the algorithm is given by:

$$\begin{cases} k_1 = f(x(t),t)\Delta t \\ k_2 = f(x(t)+k_1, t+\Delta t)\Delta t \\ x(t+\Delta t) = x(t) + \frac{1}{2}(k_1 + k_2) \end{cases} \quad \textbf{Heun's Method.} \tag{26}$$

For the *Midpoint Method* we have that $a_2 = 1$ and $a_1 = 0$ so that $q_x = \frac{1}{2}$ and $q_t = \frac{1}{2}$ and the algorithm will have the form

$$\begin{cases} k_1 = f(x(t),t)\Delta t \\ k_2 = f(x(t) + \frac{1}{2}k_1, t + \frac{1}{2}\Delta t)\Delta t \\ x(t+\Delta t) = x(t) + k_2 \end{cases} \quad \textbf{Midpoint Method.} \tag{27}$$

Finally, for *Ralston's Method* we have that $a_2 = \frac{2}{3}$ and $a_1 = \frac{1}{3}$ so $q_x = \frac{3}{4}$ and $q_t = \frac{3}{4}$ so that the algorithm will be

$$\begin{cases} k_1 = f(x(t),t)\Delta t \\ k_2 = f(x(t) + \frac{3}{4}k_1, t + \frac{3}{4}\Delta t)\Delta t \\ x(t+\Delta t) = x(t) + \frac{1}{3}k_1 + \frac{2}{3}k_2 \end{cases} \quad \textbf{Ralston's Method.} \tag{28}$$

This procedure, used to obtain the second order Runge-Kutta algorithm can be extended progressivally up to the chosen order. In particular an algorithm which is considered to be one of the most efficient is the **fourth order Runge-Kutta method**, that can be obtained by comparing the fourth order Taylor expansion with the combination of four progressive displacements of the function $f(x(t),t)$. Thus, following the same identical considerations used for the second order method, we have that the slopes used in the fourth order one are

$$\begin{cases} k_1 = f(x(t),t)\Delta t \\ k_2 = f(x(t) + \frac{1}{2}k_1, t + \frac{1}{2}\Delta t)\Delta t \\ k_3 = f(x(t) + \frac{1}{2}k_2, t + \frac{1}{2}\Delta t)\Delta t \\ k_4 = f(x(t) + k_3, t + \Delta t)\Delta t \end{cases}, \tag{29}$$

and the final update of the displacement will be their linear combination

$$x(t+\Delta t) = x(t) + \frac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right) + O(\Delta t^5). \tag{30}$$

To understand how we can apply the Runge-Kutta algorithms to a system of first order equations, let us consider the simple case of the harmonic oscillator, defined as

$$\left\{\begin{array}{l} \dot{v}(t) = f_1(x(t), v(t), t) \\ \dot{x}(t) = f_2(x(t), v(t), t) \end{array}\right. \rightarrow \left\{\begin{array}{l} \dot{v}(t) = \phi(t) \\ \dot{x}(t) = v(t) \end{array}\right. \rightarrow \left\{\begin{array}{l} \dot{v}(t) = -\omega x(t)^2 \\ \dot{x}(t) = v(t) \end{array}\right. \tag{31}$$

First, if we consider the first order **Runge-Kutta** method, we just need to define one slope for both the position and the velocity:

$$k_1^x = v(t_n)\Delta t \qquad k_1^v = \phi(t_n)\Delta t. \tag{32}$$

Using these slopes, the updates of the position and of the velocity will be written as

$$\left\{\begin{array}{l} v(t_{n+1}) = v(t_n) + k_1^v \\ x(t_{n+1}) = x(t_n) + k_1^x \end{array}\right. \rightarrow \left\{\begin{array}{l} v(t_{n+1}) = v(t_n) + \phi(t)\Delta t \\ x(t_{n+1}) = x(t_n) + v(t_n)\Delta t \end{array}\right. ; \tag{33}$$

which corresponds to Euler's algorithm, which has an accuracy in fact of $O(\Delta t)$.
In order to define the second order Runge-Kutta (the Midpoint method) we need to compute the second slopes

$$\left\{\begin{array}{l} k_2^x = \Delta t\left[v(t_n) + \frac{1}{2}k_1^v\right] = \Delta t\left[v(t_n) + \frac{1}{2}\phi(t_n)\Delta t\right] \\ k_2^v = -\omega^2\Delta t\left[x(t_n) + \frac{1}{2}k_1^x\right] = -\omega^2\Delta t\left[x(t_n) + \frac{1}{2}v(t_n)\Delta t\right] \end{array}\right. \tag{34}$$

so that in this case we have

$$\left\{\begin{array}{l} v(t_{n+1}) = v(t_n) + k_2^v \\ x(t_{n+1}) = x(t_n) + k_2^x \end{array}\right. \rightarrow \left\{\begin{array}{l} v(t_{n+1}) = v(t_n) + \phi(t)\Delta t - \frac{\omega^2}{2}v(t)\Delta t^2 \\ x(t_{n+1}) = x(t_n) + v(t_n)\Delta t + \frac{1}{2}\phi(t_n)\Delta t^2 \end{array}\right. . \tag{35}$$

Finally, for the fourth order Runge-Kutta algorithm we can progress the computation of the slopes for the position and for the velocity up to the fourth order

$$\left\{\begin{array}{ll} k_1^v = -\omega^2 x(t_n)\Delta t & k_1^x = v(t_n)\Delta t \\ k_2^v = -\omega^2(x(t_n) + \frac{1}{2}k_1^x)\Delta t & k_2^x = \left(v(t_n) + \frac{1}{2}k_1^v\right)\Delta t \\ k_3^v = -\omega^2(x(t_n) + \frac{1}{2}k_2^x)\Delta t & k_3^x = \left(v(t_n) + \frac{1}{2}k_2^v\right)\Delta t \\ k_4^v = -\omega^2(x(t_n) + k_3^x)\Delta t & k_4^x = \left(v(t_n) + k_3^v\right)\Delta t \end{array}\right. ; \tag{36}$$

so that the full updates will be given by:

$$\left\{\begin{array}{l} x(t_{n+1}) = x(t_n) + \frac{1}{6}\left(k_1^x + 2k_2^x + 2k_3^x + k_4^x\right) \\ v(t_{n+1}) = v(t_n) + \frac{1}{6}\left(k_1^v + 2k_2^v + 2k_3^v + k_4^v\right) \end{array}\right. . \tag{37}$$