An asynchronous leap-frog method

Ulrich Mutze *

A second order explicit one-step numerical method for the initial value problem of the general ordinary differential equation is proposed. It is obtained by natural modifications of the well-known leap-frog method, which is a second order, two-step explicit method. According to the latter method, the input data for an integration step are two system states which refer to different times (we employ the terminology of dynamical systems). The usage of two states instead of a single one can be seen as the reason for the robustness of the method. Since the time step size thus is part of the step input data, it is complicated to change this size during the computation of a discrete trajectory. This is a serious drawback when one needs to implement automatic time step control.

The proposed modification transforms one of the two input states into a velocity and thus gets rid of the time step dependency in the step input data. For these new step input data, the leap-frog method gives a unique prescription how to evolve them stepwise.

The method is exemplified with the equation of motion of a one-dimensional non-linear oscillator describing the radial motion in the Kepler problem. For this equation the modified leap-frog method is shown to be significantly more accurate than the original method.

As a result, we have a second order explicit method that, just as the simple explicit Euler method, needs only one evaluation of the right-hand side of the differential equation per integration step, and allows to change the time step without any additional computational burden after each integration step. Unlike the Euler method and the explicit Runge-Kutta methods it is robust in the sense that it allows us to reliably model the dynamics of a wide variety of physical systems over extended periods of time.

1 Introduction

We consider the initial value problem of the general ordinary differential equation

$$\dot{\psi}(t) = F(t, \psi(t)) \tag{1}$$

*www.ulrichmutze.de

for a time-dependent quantity ψ which takes values in a real finite-dimensional vector space $\mathcal{H}$. Here F is a function $\mathbb{R} \times \mathcal{H} \to \mathcal{H}$ [1] . Equations of this kind arise from ordinary differential equations of finite order and from discrete approximations to partial differential equations like the time-dependent Schrödinger equation or Maxwell's equations. The main concrete example in the present work will be much simpler: $\mathcal{H} = \mathbb{R}^2$

$$\begin{aligned} F \; : \; \mathbb{R} \times \mathbb{R}^2 \; &\to \; \mathbb{R}^2 \\ (t, (x, v)) \; &\mapsto \; \left(v, \frac{1}{x^2} \left(\frac{1}{x} - 1 \right) \right) \end{aligned} \tag{2}$$

which describes the radial motion in the Kepler problem. This is a convenient test problem since its initial value problem can be solved directly without making use of time stepping by solving Kepler's famous transcendental equation. Since it is equivalent to a *Hamiltonian* problem, there are many integration methods available (e.g. [6], [13]) to compare with. More demanding applications of the method, in which $\mathcal{H}$ is a higher-dimensional space, are considered in [17] and presented in [18]. The additions of March 2010 deal with the elementary differential equation (12) and similar equations.

The *computational initial value problem* associated with this equation (1) asks for an algorithm which determines for each $\mathbb{R}$-valued increasing list $t_0, t_1, \ldots, t_n$ and each value $\psi_0 \in \mathcal{H}$ (the initial value) a $\mathcal{H}$-valued list $\psi_1, \ldots, \psi_n$ such that the $\mathbb{R} \times \mathcal{H}$-valued list $(t_0, \psi_0), (t_1, \psi_1), \ldots, (t_n, \psi_n)$ is a reasonable approximation to a solution curve $t \quad \mapsto \quad \psi(t)$, $\psi(t_0) = \psi_0$, of the differential equation (1) whenever the regularity properties of F suffice for determining such a curve, and the gaps between adjacent $t-$values are small enough. If such an algorithm works only for equidistant time lists (for which $t_i - t_{i-1}$ is independent of i by definition) it is said to be *synchronous* and otherwise it is said to be *asynchronous*. Asynchronous algorithms may be developed into adaptive ones, which adjust their step size $t_{i+1} - t_i$ to the size (according to a suitable notion of size in $\mathcal{H}$) of $F(t_i, \psi_i)$. For a $\mathbb{R}$-valued function F that depends on its second argument trivially, the initial value problem is simply the problem of computing the definite integral. It is straightforward and instructive to specialize the proposed algorithms to this simplified concrete situation.

Starting from the well-known leap-frog algorithm, the present article develops and analyzes an economic and robust asynchronous solution of the computational initial value problem associated with (1). Section 2 recalls the leap-frog method, and Section 3 carries out the general development of the new asynchronous algorithm. Section 4 will apply this integration method and related methods to the differential equation defined by (2).

[1] As is well-known, one may increase formal simplicity by transforming away the explicit t-dependence of the right-hand side of this equation, thus rendering it *autonomous*. However, I refrain from assuming autonomy, since the algorithms to be considered should apply to time-dependent real-world problems directly, without a need to transform them into autonomous ones.

2 The leap-frog method

A marvelously simple synchronous solution algorithm for the computational initial value problem of (1) is the *leap-frog method* or *explicit midpoint rule*, see e.g. [7], eq. (3.3.11). It seems to be the first method that has been successfully applied to the initial value problem of the time-dependent Schrödinger equation (see [2] and the citation of this work in [3]). It is most conveniently considered as the map

$$\begin{aligned} \mathcal{L} \,:\, (\mathbb{R}\times\mathcal{H})\times(\mathbb{R}\times\mathcal{H}) &\to (\mathbb{R}\times\mathcal{H})\times(\mathbb{R}\times\mathcal{H}) \\ ((t_0,\psi_0),(t_1,\psi_1)) &\mapsto ((t_1,\psi_1),(t_2,\psi_2))\;, \end{aligned} \tag{3}$$

where

$$\begin{aligned} t_2 &:= 2t_1 - t_0\;, \\ \psi_2 &:= \psi_0 + (t_2 - t_0)\,F(t_1,\,\psi_1)\;. \end{aligned} \tag{4}$$

The equivalent form

$$\begin{aligned} \frac{t_0+t_2}{2} &= t_1\;, \\ \frac{\psi_2-\psi_0}{t_2-t_0} &= F(t_1,\,\psi_1) \end{aligned} \tag{5}$$

of these equations, together with equation (1), makes the reasons for choosing them evident:

$$F(t_1,\,\psi(t_1)) = \dot{\psi}(t_1) = \frac{\psi(t_2)-\psi(t_0)}{t_2-t_0} + O((t_2-t_0)^3)\;. \tag{6}$$

Iterating the map $\mathcal{L}$ determines a *leap-frog trajectory*, $((t_j,\psi_j))_{j\in\mathbb{N}}$ if (t_0,ψ_0) and (t_1,ψ_1) are given:

$$(t_{i+1},\psi_{i+1}) = \pi_2(\mathcal{L}((t_{i-1},\psi_{i-1}),(t_i,\psi_i))) = \pi_2(\mathcal{L}^i((t_0,\psi_0),(t_1,\psi_1)))\;, \tag{7}$$

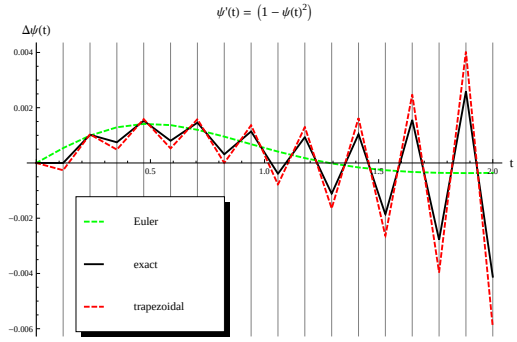
where π_2 is the canonical projection to the second component of a pair. According to the initial value problem we are given t_0, t_1 (which determines, due to the assumed synchronicity, all further t-values) and ψ_0. For starting iteration (7) we need also ψ_1. This has to be added in a way consistent with (1), e. g. by employing the *explicit Euler rule*

$$\psi_1 := \psi_0 + (t_1 - t_0)\,F(t_0,\,\psi_0)\;. \tag{8}$$

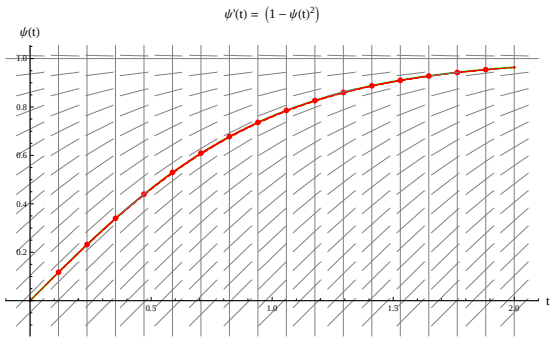
One could expect (as I did for some time) that being more accurate here would improve the accuracy of the leap-frog trajectory. The optimum definition could be expected to be

$$\psi_1 := \psi(t_1)\;, \tag{9}$$

where ψ is the *exact* trajectory determined by (1) and $\psi(t_0) = \psi_0$. In a reasonably posed problem t_1 is close to t_0 and the exact trajectory can be arbitrarily well be approximated



(a) Dependence of the accuracy on the initialization method.



(b) Exact (green) and discrete trajectory (red, Euler initialization).

Figure 1: Case in which Euler initialization is superior.

by any numerical method which is short term accurate irrespective of potentially unfavorable long-term behavior (e.g. fourth order Runge-Kutta). An intermediate position between methods (8) and (9) is given by employing the implicit *trapezoidal rule*

$$\psi_1 = \psi_0 + (t_1 - t_0)\,\frac{F(t_0,\,\psi_0) + F(t_1,\,\psi_1)}{2}\;, \tag{10}$$

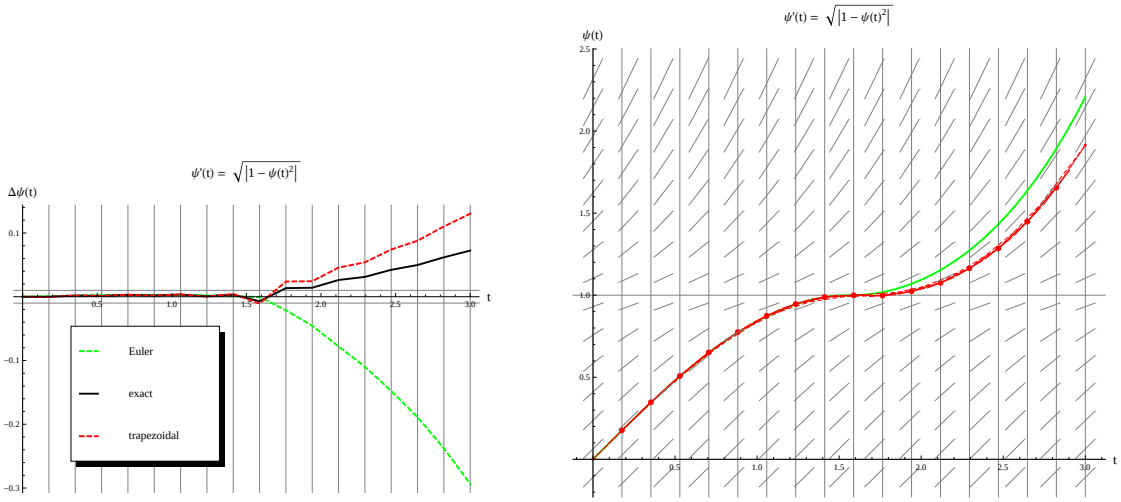
which can be efficiently solved by the iteration

$$\psi_1^{(i+1)} := \psi_0 + (t_1 - t_0)\,\frac{F(t_0,\,\psi_0) + F(t_1,\,\psi_1^{(i)})}{2}\;, \quad \psi_1^{(0)} := \psi_0\;. \tag{11}$$

Let us study the behavior of these methods for a well-known non-linear differential equation for which the exact solution can be expressed in terms of exponential functions and thus is exactly known over any range:

$$\dot{\psi}(t) = 1 - \psi(t)^2\;, \quad \psi(0) = 0 \quad \text{thus} \quad \psi(t) = \tanh(t)\;. \tag{12}$$

Figure 1 shows the difference between the exact trajectory and the leap-frog trajectories according to the three initialization methods introduced above. Surprisingly the less sophisticated method (8) works best in this case. Unfortunately this is not the case in all interesting applications. An example for this is shown in Figure 2 for a different differential equation. We thus have an interesting situation: Even for arbitrary ψ_1, the iteration (7) can be used to define a leap-frog trajectory. This then is typically a zig-zag line which tends to wiggle around a trajectory of differential equation (1) and initial data (t_0, ψ_0). See Figure 3, where the leap-frog trajectory of Figure 1(b) is modified by shifting ψ_1 considerably from its correct position. Notice the graphical manifestation of the leap-frog algorithm: The red dots make up the discrete trajectory. In all but the first and the last trajectory points — let an arbitrary such point be designated (t_i, ψ_i) — there is a short solid red line which indicates the slope required by the differential equation



(a) Dependence of the accuracy on the initialization method.

(b) Exact (green) and discrete trajectory (red, Euler initialization).

Figure 2: Case in which Euler initialization is inferior.

at this position. To each such short solid line, there is a parallel broken line which connects an already constructed trajectory point (t_{i-1}, ψ_{i-1}) with the point (t_{i+1}, ψ_{i+1}) to be constructed next. The closer ψ_1 is set to this trajectory (e.g. by (8) as done in Figure 1(b)) the lower the zig-zag amplitude will be. The auxiliary lines, which are evident in Figure 3 are generated also for Figure 1(b) but they collapse here to a single polygon.

The map (3), considered as a discrete dynamical system, is thus related to the continuous dynamical system associated with (1) in a more sophisticated manner than usual: The state space of the discrete system is $(\mathbb{R} \times \mathcal{H}) \times (\mathbb{R} \times \mathcal{H})$ since only this set allows the leap-frog method to be defined as a map of states into states. Only a subset of this state space (e. g. the one given by (8)) corresponds to potential initial states of the continuous system (1).

Equation (7) defines ψ_i for arbitrarily large i, whereas equation (1) may drive a trajectory in finite time into infinity. A well-known example is

$$\dot{\psi}(t) = 1 + \psi(t)^2 \,, \quad \psi(0) = 0 \quad \text{thus} \quad \psi(t) = \tan(t) \quad \text{and} \quad \psi(\pi/2) = \infty \,. \tag{13}$$

In such cases the size of the numbers involved in applying mapping $\mathcal{L}$ repeatedly will grow above the size which can be handled with realistic computational resources (which include computation time). Such exploding situations also occur if the time step size is too large for the differential equation under consideration.

It might be instructive to discuss the close correspondence of the leap-frog algorithm (4) to the leap-frog game (Bockspringen in German). In the variant which is of interest here, there are two participants A and B in this nice dynamical sportive exercise. There

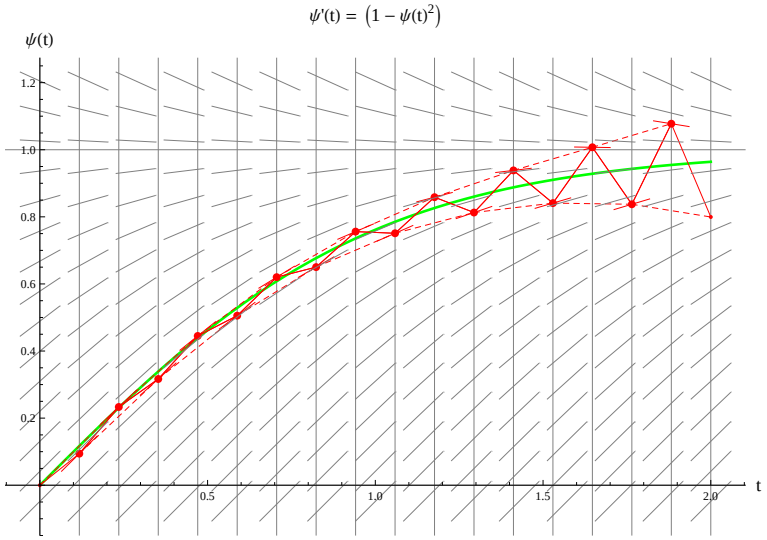


Figure 3: Zig-zag trajectory resulting from intentionally spoiled initialization.

is an intended direction of motion and the participants line up in this direction, B standing a few meters in front of A. This is the initial condition, which corresponds in the algorithm to the ordered input $((t_0, \psi_0) \cong A, (t_1, \psi_1) \cong B)$. After two or three energetic steps, A jumps over B, supporting himself with both palms on the shoulders of B, thereby receiving from B a smooth kick which makes A fly to a position sufficiently far in front of B that now the action can continue with the roles of A and B reversed, then reversed again, and so forth. The kick which A receives from B corresponds to adding the term $(t_2 - t_0)\, F(t_1,\, \psi_1)$ (associated with B) to the term ψ_0, which is associated with A. The result of this addition is ψ_2, which corresponds again to A, but at a new position. By continuation we create terms $\psi_3, \psi_4 \ldots$. All terms with even index correspond to A, and those with odd index to B. The index grows with the progress along the intended direction of motion.

The algorithm can easily be shown to be *reversible*: Let the operator of motion reversal be defined as

$$\begin{aligned} \mathcal{T} \;:\; (\mathbb{R} \times \mathcal{H}) \times (\mathbb{R} \times \mathcal{H}) \;&\to\; (\mathbb{R} \times \mathcal{H}) \times (\mathbb{R} \times \mathcal{H}) \\ ((t_0, \psi_0), (t_1, \psi_1)) \;&\mapsto\; ((t_1, \psi_1), (t_0, \psi_0)) \end{aligned} \tag{14}$$

then we easily verify

$$\mathcal{L} \circ \mathcal{T} \circ \mathcal{L} = \mathcal{T}\,, \quad \mathcal{T} \circ \mathcal{T} = \mathbf{1} \tag{15}$$

from which one concludes that $\mathcal{L}$ is invertible, with the inverse given by $\mathcal{T} \circ \mathcal{L} \circ \mathcal{T}$. This allows us to reconstruct from the last two data $((t_{n-1}, \psi_{n-1}), (t_n, \psi_n))$ of a leap-frog trajectory all previous components (t_k, ψ_k), $k < n - 1$,

$$(t_k, \psi_k) = \pi_2(\mathcal{T}\mathcal{L}^{n-k}\mathcal{T}((t_{n-1}, \psi_{n-1}), (t_n, \psi_n)))\,. \tag{16}$$

If we would like to change time step size after having arrived at some state $((t_{p-1}, \psi_{p-1}), (t_p, \psi_p)$ to value τ, we may start a new synchronous trajectory with the state

$$((t_p, \psi_p), (t_p + \tau, \psi_p + \tau F(t_p, \psi_p)) \tag{17}$$

or a potentially more accurate form which also involves ψ_{p-1} (which equation (17) simply forgets).

3 An asynchronous version of the leap-frog method

What I intend here, is to modify the leap-frog method in a way that no tradeoffs between simplicity and accuracy are involved when we start a trajectory or when we change the time step size. A further aim is to preserve the computational simplicity of the algorithm. In a narrower framework than (1) this modified leap-frog method has been introduced in [17] and applied to time-dependent Hartree equations in [18].

We consider four consecutive components of a leap-frog trajectory of (1)

$$(t_k,\, \psi_k)\ , \quad (t_{k+1},\, \psi_{k+1})\ , \quad (t_{k+2},\, \psi_{k+2})\ , \quad (t_{k+3},\, \psi_{k+3}) \tag{18}$$

and let τ be the time step. Then we define velocity-like quantities ϕ as follows:

$$\phi_k := \frac{\psi_{k+1} - \psi_k}{\tau}\ , \quad \phi_{k+1} := F(t_{k+1},\, \psi_{k+1})\ , \quad \phi_{k+2} := \frac{\psi_{k+2} - \psi_{k+1}}{\tau}\ . \tag{19}$$

From this definition and from (5) we obtain

$$\frac{\phi_k + \phi_{k+2}}{2} = \frac{\psi_{k+2} - \psi_k}{2\tau} = F(t_{k+1},\, \psi_{k+1}) = \phi_{k+1}\ . \tag{20}$$

These equations allow us to compute $(t_{k+2},\, \psi_{k+2},\, \phi_{k+2})$ if $(t_k,\, \psi_k,\, \phi_k)$ and τ are given:

$$\begin{aligned}
t_{k+1} &= t_k + \tau\ , \quad \psi_{k+1} = \psi_k + \tau\, \phi_k\ , \quad \phi_{k+1} = F(t_{k+1},\, \psi_{k+1})\ , \\
t_{k+2} &= t_{k+1} + \tau\ , \\
\psi_{k+2} &= \psi_k + 2\,\tau\, \phi_{k+1}\ , \\
\phi_{k+2} &= \frac{\psi_{k+2} - \psi_{k+1}}{\tau}\ .
\end{aligned} \tag{21}$$

Equation (20) allows us to give the last two equations of (21) a more symmetrical form:

$$\begin{aligned}
\phi_{k+2} &= 2\, \phi_{k+1} - \phi_k\ , \\
\psi_{k+2} &= \psi_{k+1} + \tau\, \phi_{k+2}\ .
\end{aligned} \tag{22}$$

The association $(t_k,\, \psi_k,\, \phi_k) \quad \mapsto \quad (t_{k+2},\, \psi_{k+2},\, \phi_{k+2})$ can now be considered a mapping $\mathbb{R} \times \mathcal{H} \times \mathcal{H} \ \to \ \mathbb{R} \times \mathcal{H} \times \mathcal{H}$ which depends on the total time step $2\,\tau$ and thus will be denoted $\mathcal{A}_{2\tau}$. The data $(t_{k+1},\, \psi_{k+1},\, \phi_{k+1})$ are intermediary (or temporary) with respect to this mapping. This mapping may be iterated: Let us apply $\mathcal{A}_{2\tau}$ to $(t_{k+2},\, \psi_{k+2},\, \phi_{k+2})$. The result may be denoted $(t_{k+4},\, \chi_{k+4},\, \kappa_{k+4})$ and the intermediary

data as $(t_{k+3},\, \chi_{k+3},\, \kappa_{k+3})$. Although the formulas which define $\mathcal{A}_{2\tau}$ are all consequences of the leap-frog law, the iteration of $\mathcal{A}_{2\tau}$ does not exactly continue the leap-frog trajectory. Actually, we have $\chi_{k+3} = \psi_{k+3}$ only up to terms of order τ^2:

$$\begin{aligned} \psi_{k+3} &= \psi_{k+1} + 2\,\tau\, F(t_{k+2},\, \psi_{k+2})\ , \\ \chi_{k+3} &= \psi_{k+2} + \tau\,\phi_{k+2} = 2\,\psi_{k+2} - \psi_{k+1} = \psi_{k+1} + 2\,(\psi_{k+2} - \psi_{k+1}) \end{aligned} \tag{23}$$

and thus

$$(\psi_{k+3} - \chi_{k+3})/2 = \tau\, F(t_{k+2},\, \psi_{k+2}) + \psi_{k+1} - \psi_{k+2} = O(\tau^2)\ . \tag{24}$$

The reason for this behavior lies in the fact that in going from the normal leap-frog algorithm to the asynchronous one we change the notion of system state. The new state notion is more conventional in so far as it refers to a single point in time, whereas the normal leap-frog state consists of data that refer to two points in time. If we are given, according to the initial value problem of (1), the initial values t_0 and ψ_0, the augmentation to a full state according to the new state notion is straightforward and does not depend on the next time value t_1. It is simply given by:

$$\phi_0 := F(t_0,\, \psi_0)\ . \tag{25}$$

From (t_0, ψ_0, ϕ_0) and a list $(t_1, t_2, \ldots)$ we generate a discrete trajectory

$$((t_0, \psi_0, \phi_0), (t_1, \psi_1, \phi_1), (t_2, \psi_2, \phi_2), \ldots) \tag{26}$$

by the iteration of mappings $\mathcal{A}_{2\tau}$ defined earlier

$$(t_{i+1}, \psi_{i+1}, \phi_{i+1}) := \mathcal{A}_{h_i}(t_i, \psi_i, \phi_i)\ , \quad h_i := t_{i+1} - t_i\ . \tag{27}$$

For convenience, let us rewrite the definition of $\mathcal{A}$ in fully explicit terms: For each $h \in \mathbb{R}$ the mapping

$$\begin{aligned} \mathcal{A}_h\ :\ \mathbb{R} \times \mathcal{H} \times \mathcal{H}\ &\to\ \mathbb{R} \times \mathcal{H} \times \mathcal{H} \\ (t, \psi, \phi)\ &\mapsto\ (\underline{t}, \underline{\psi}, \underline{\phi}) \end{aligned} \tag{28}$$

can be seen from (21) and (22) to be defined by the following chain of formulas:

$$\begin{aligned} \tau &:= \frac{h}{2}\ , \\ t' &:= t + \tau\ , \\ \psi' &:= \psi + \tau\,\phi\ , \\ \phi' &:= F(t',\, \psi')\ , \\ \underline{\phi} &:= 2\,\phi' - \phi\ , \\ \underline{\psi} &:= \psi' + \tau\,\underline{\phi} = \psi + 2\tau\,\phi'\ , \\ \underline{t} &:= t' + \tau\ . \end{aligned} \tag{29}$$

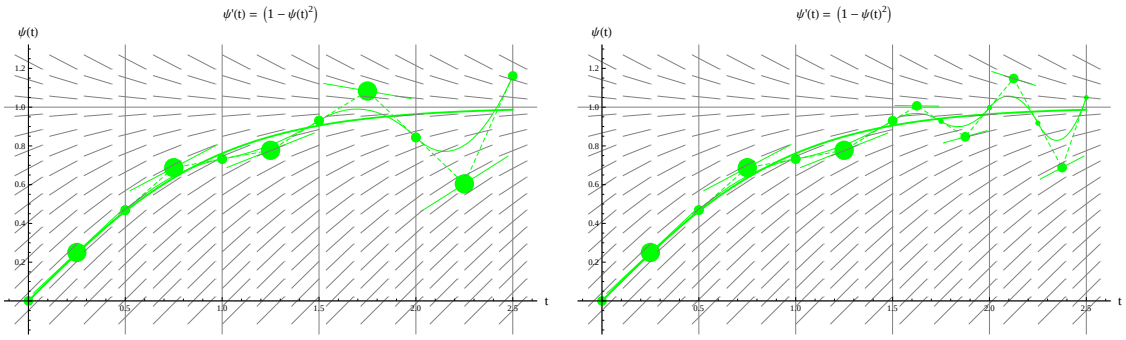
This algorithm corresponds to equation (8) in [17] but is more general since it does not assume the special form of F that was considered there. Notice that the leap-frog midpoint state data t', ϕ', ψ' appear only as intermediary quantities that help to give the algorithm an elegant form. In particular, they do not belong to the discrete trajectory (26), (27) generated by $\mathcal{A}$. Their geometrical role becomes clear from the representation of the final state as

$$\underline{\psi} = \psi + h\,\phi + \frac{h^2}{2}\frac{\phi' - \phi}{\tau}\ , \quad \underline{\phi} = \phi + h\frac{\phi' - \phi}{\tau}\ . \tag{30}$$

This representation suggests an interpretation in which h is replaced by a parameter which varies from 0 to h and thus connects the states ψ and $\underline{\psi}$ by a parabolic curve in the linear space $\mathcal{H}$ (and the quantities ϕ and $\underline{\phi}$ by a linear curve). Everywhere along this connecting curve, *ϕ is the time derivative of ψ*. The connecting parabola is easily seen to be the Bézier curve generated by the *control points* $(t, \psi), (t', \psi'), (\underline{t}, \underline{\psi})$. In this way, the inherently time-discrete method proposes its own time-continuous representation. This is very convenient if one needs to compare trajectories from simulations with different time steps. This time continuous representation is by mere interpolation; if one needs true detail about the history between ψ and $\underline{\psi}$ one has to reduce the time step in the simulation. It is interesting to observe that the parabolas of adjacent time steps fit together in a differentiable manner so that a sequence of time steps gives rise to a quadratic *Bézier spline* as a differentiable representation of the discrete trajectory. Figure 4 shows this spline curve together with the control points. The larger disks mark the intermediary configurations (t', ψ') and the smaller ones mark the configurations (t, ψ) (or $(\underline{t}, \underline{\psi})$) which belong to the discrete trajectory. The short solid line attached to the larger disks indicates the direction given by the direction field of the differential equation. It coincides with the direction determined by connecting the two neighboring smaller disks. In Figure 4(b), instead of the two final steps of sub-figure (a) we have four final steps of half the step size. Notice that the size of the marking disks is coupled to the step size so that the large disks belonging to the small steps equal in size just the small disks belonging to the large steps. The disk at $t = 1.5$ marks the final discrete configuration reached by a large step and also is the first discrete configuration from which a small step starts (so it has also to be marked with a disk half this size; the data structure of the graphics contains such a disk, it is hidden by the larger disk since it is not given a different color).

In the example of Figure 4 the horizontal course of the exact trajectory and the direction field provided by the differential equation in its neighborhood enforce the formation of a wave. The significance of this phenomenon is not clear. It is tempting to speculate that classical particle trajectories could be transformed to wave-like processes by discretization. Some form of discretization should be expected to happen, since the 'computational resources of Nature' available for the evolution of any particle should be expected to be limited.

The evolution equations (29) can be given a form where no quantity needs to be copied



(a) Constant large time step.

(b) Half this time step in the second half of the trajectory.

Figure 4: Geometry of (29) and reduction of the time step.

and memorized:

$$\begin{aligned} t &+= \tau \,, \\ \psi &+= \tau\, \phi \,, \\ \phi &+= 2\,\lambda\,(F(t,\,\psi) - \phi) \,, \\ \psi &+= \tau\, \phi \,, \\ t &+= \tau \,, \end{aligned} \tag{31}$$

This property is obviously advantageous if ψ and ϕ are large arrays of data as they are in simulations of systems with many degrees of freedom. I tend to favor this property also at a conceptual physical level. The *relaxation parameter* λ introduced here has to be 1 for (31) to be equivalent to (29). Values slightly less than 1 let the method work in some cases well where otherwise large deviations from the exact trajectory would occur. Figure 5 shows an example which demonstrates drastic reduction of the deviations seen in Figure 4. If we continue the trajectory to larger values of t, excessive oscillations seem to built up unless relaxation is in place to prevent them. Figure 6 illustrates this. Here the discrete trajectory is shown in red color by rendering only the corresponding spline curve. The exact solution (tanh, see (12)) is shown in green color. The phenomenon of oscillation is related to the property of reversibility: The sequence of (ψ, ϕ)- states is not allowed to have two equal components, since going back from two equal states a suitably selected number of steps one would get two different points with $\psi = 0$, although inspection of the discrete trajectory shows that there is only *one* such point. If the exact trajectory would not be effectively constant, the occurrence of equal ψ-values along the trajectory could easily be avoided. But in the case under consideration reversibility forces the trajectory to make use of ever new values of ψ and ϕ which is in conflict with the aim to render the exact trajectory with good accuracy. The mechanism which brings about this kind of 'self-avoidance' seems to be poorly understood.

It might be convenient to see (29) rewritten in the self-explanatory style of [12](which

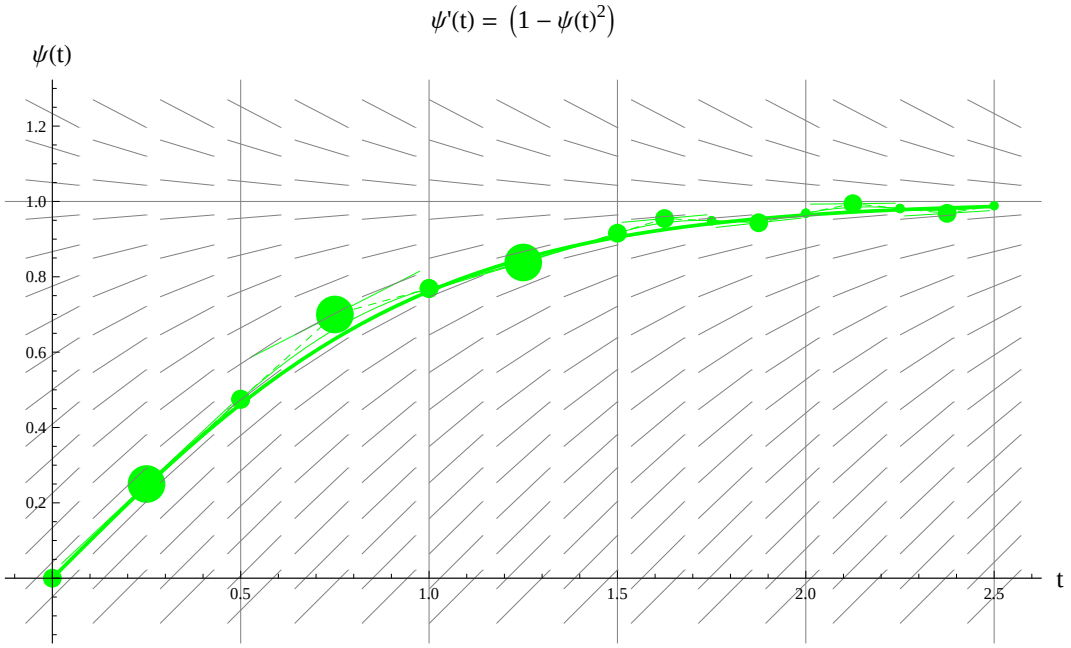
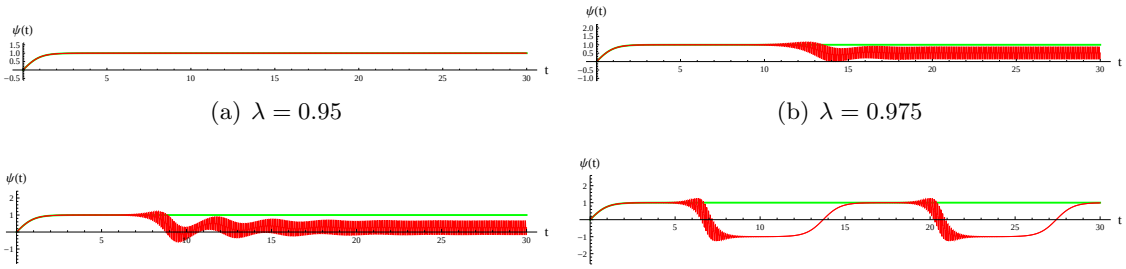


Figure 5: Situation of Figure 4(b) with relaxation ($\lambda = 0.8$).

is similar to that in [13]): We write our differential equation (1) as

$$\dot{q} = f(t, q) \tag{32}$$

and the initial data as t_0, q_0. We complement them by setting $v_0 := f(t_0, q_0)$ and are in a position to define a fully explicit time step which promotes data indexed by n to data



(a) $\lambda = 0.95$ (b) $\lambda = 0.975$

(c) $\lambda = 0.9875$ (d) $\lambda = 1.0$

Figure 6: How relaxation works for a virtually horizontal exact trajectory.

indexed by $n+1$:

$$\boxed{\begin{aligned} &t_{n+\frac{1}{2}} = t_n + \frac{h}{2}\,, \quad q_{n+\frac{1}{2}} = q_n + \frac{h}{2}\,v_n\,, \quad v_{n+\frac{1}{2}} = f(t_{n+\frac{1}{2}},\, q_{n+\frac{1}{2}})\,, \\ &v_{n+1} = 2\,v_{n+\frac{1}{2}} - v_n\,, \quad q_{n+1} = q_{n+\frac{1}{2}} + \frac{h}{2}\,v_{n+1} = q_n + h\,v_{n+\frac{1}{2}}\,, \quad t_{n+1} = t_n + h\,. \end{aligned}} \tag{33}$$

Notice that the initial condition plays a slightly exceptional role: For $n > 0$ the quantities v_n and $f(t_n, q_n)$ equal only approximately, whereas for $n = 0$ they equal exactly . The small quantity $\delta_n := v_n - f(t_n, q_n)$ is an interesting one to monitor in simulations. Notice $\delta_0 = 0$.

Now we return to our ψ, ϕ-notation. The symmetric grouping of the formulas in (29) suggests that the time step map can be written as a product of three maps. For each $h \in \mathbb{R}$ we define the mapping

$$\begin{aligned} B_h \,:\, \mathbb{R}\times\mathcal{H}\times\mathcal{H} \;&\to\; \mathbb{R}\times\mathcal{H}\times\mathcal{H} \\ (t,\psi,\phi) \;&\mapsto\; (t+h,\, \psi + h\,\phi,\, \phi) \end{aligned} \tag{34}$$

and the mapping

$$\begin{aligned} C \,:\, \mathbb{R}\times\mathcal{H}\times\mathcal{H} \;&\to\; \mathbb{R}\times\mathcal{H}\times\mathcal{H} \\ (t,\psi,\phi) \;&\mapsto\; (t,\psi,\, 2\,F(t,\psi) - \phi)\,. \end{aligned} \tag{35}$$

We than see immediately

$$\mathcal{A}_h = B_{h/2} \circ C \circ B_{h/2}\,. \tag{36}$$

Obviously B_h is *symplectic*: Writing $B_h(t,\psi,\phi)$ as (t',ψ',ϕ') (where now the apostrophe is used again as a normal diacritical mark and not in the special meaning of (29)) we get $\mathrm{d}\psi' \wedge \mathrm{d}\phi' = (\,\mathrm{d}\psi + h\,\mathrm{d}\phi) \wedge \mathrm{d}\phi = \mathrm{d}\psi \wedge \mathrm{d}\phi$. Surprisingly C is not symplectic but *skew-symplectic*: Writing $C(t,\psi,\phi)$ as (t,ψ',ϕ') we have $\mathrm{d}\psi' \wedge \mathrm{d}\phi' = \mathrm{d}\psi \wedge (2\,\mathrm{d}F - \mathrm{d}\phi) = -\,\mathrm{d}\psi \wedge \mathrm{d}\phi$, since $\mathrm{d}F$ is proportional to $\mathrm{d}\psi$ (notice $\mathrm{d}t = 0$) which implies $\mathrm{d}F \wedge \mathrm{d}\psi = 0$. The product representation (36) then implies that also $\mathcal{A}_h$ is skew-symplectic. Obviously the product of two skew-symplectic maps is symplectic. *Therefore, the densified form (47) of the asynchronous leapfrog integrator is a symplectic explicit integrator.* Symplecticity plays here an unusual role, however, since associating the 'dynamical states' (ψ,ϕ) with a system trajectory is a peculiarity of the method; the usual description uses ψ alone. In a conventional framework we encounter the variable ϕ only if our original differential equation is the one which results from (1) by differentiation with respect to time:

$$\begin{aligned} \dot{\psi} &= \phi\,, \\ \dot{\phi} &= \frac{\partial F}{\partial t} + \frac{\partial F}{\partial \psi}\,\phi\,. \end{aligned} \tag{37}$$

Using integrator (29) for this equation may be non-trivial since the integrator uses F whereas the differential equation (37) gives only $\frac{\partial F}{\partial t}$ and $\frac{\partial F}{\partial \psi}$, so that one obtains F by

solving a (probably partial) differential equation. The method to convert a first-order differential equation into a second order one by differentiation with respect to time and then to apply an explicit Störmer-Verlet integrator is the path which led me to (33). I followed this path in order to simulate quantum mechanical systems [16], [17].

Obviously $\mathcal{A}_0$ is the identity map and $\mathcal{A}$ is *reversible* in the sense that for all $h \in \mathbb{R}$ we have — by a non-trivial cancellation of terms — the equation

$$\mathcal{A}_{-h} \circ \mathcal{A}_h = \mathcal{A}_0 \tag{38}$$

which implies that each of the maps $\mathcal{A}_h$ is invertible, as is the product of arbitrarily many such mappings. As mentioned already for the corresponding situation of the normal leap-frog method, this invertibility of all discrete evolution maps does not imply that the dynamical system defined by (1) has invertible evolution maps. The reversion of discrete trajectories is discussed in [17] subsequent to equation (10). It is to be noted that there is no motion reversion operator comparable to (14). One may be tempted to try $\mathcal{T} : (t, \psi, \phi) \mapsto (-t, \psi, -\phi)$, but this fails to satisfy $\mathcal{A}_h \circ \mathcal{T} \circ \mathcal{A}_h = \mathcal{T}$ which would correspond to (15). One should also note that the concept (38) does not assume that the differential equation (1) satisfies any reversibility condition, in particular not the one assumed in [14], after equation (1).

For a given state (t, ψ, ϕ), which determines a discrete trajectory by successive application of $\mathcal{A}_h$, one may consider the leap-frog state $(t - h, \psi - h\phi), (t, \psi)$ which, by successive application of $\mathcal{L}$, creates a leap-frog trajectory which is very similar to the trajectory considered before. Therefore, in a sense, the role of ϕ is to memorize information from the foregoing integration step in addition to state data ψ. Also multi-step methods and predictor-corrector methods improve computational economy by memorizing results from antecedent integration steps. But they do so directly, by memorizing a list of previous states, each associated with the time of its validity. Letting information from antecedent integration steps propagate in the form of derived quantities, such as our ϕ-data, is the clue of the present method.

The connection between trajectories generated by the two leap-frog methods can be made even more striking: The leap-frog state

$$(t - \tau, \psi - \tau\phi), (t + \tau, \psi + \tau\phi) \ , \quad \tau := \frac{h}{2} \tag{39}$$

and the 'asynchronous leap-frog state' (t, ψ, ϕ) can easily be seen to generate nearly identical discrete trajectories by application of their respective integrators: The leap-frog configurations (t_i, ψ_i) coincide with the 'mid-configurations' $(t_{i+\frac{1}{2}}, q_{i+\frac{1}{2}})$ of the asynchronous leap-frog trajectory. So, the two trajectories differ essentially only by subtleties of interpretation. They always remain closely together. Actually, apart from the endpoints of the trajectories, the situation is simple: the midpoint of two adjacent points of one kind of trajectory is just a point of the other kind of trajectory. So the trajectories of he two methods are woven into each other. It is to be noticed, however, that for a given initial value $(t_0, \psi(t_0))$ starting the normal leapfrog trajectory according to (8) or (10) is probably considered more natural than the symmetric choice (39). Then there is no simple exact relation between the two kinds of trajectories.

To consider the usage of derived quantities such as ϕ is most directly motivated by the desire to obtain an asynchronous method, as the matter is presented here. Actually, my motivation was a different one, namely to extend the well-behaved integrator (46) for differential equations (45) of second order to the more general situation (1). The close relation of the arising method to the leap-frog method became apparent only later.

With the integrator (33) we have achieved for the general differential equation the status that the Störmer-Verlet method (e.g. [13] equations (2.16-18)) achieves for the equations of motion of Newtonian mechanics: it is second order, reversible, robust ('geometrical integrator'). Actually we have an improvement at least over this method as cited above: we need only one evaluation of the force term per integration step. The property of symplecticity, which commonly is considered to be the source for robustness can be claimed only at the risk of misunderstandings for our integrator, since the concept of a phase space is present only in a degenerate form. As pointed out earlier, my understanding is that just this rudimentary phase space structure, brought about by the the introduction of quantity ϕ, is the source for the robustness of the present method. The reversibility of an integrator for the general equation (1) implies more miracles than it implies for the reversible equations of Newtonian mechanics. This is pointed out in a discussion of the 'leaking bucket equation' to be found on my homepage. For this equation the final part of the exact trajectory is exactly horizontal so that we have similar but more transparent conditions as those discussed above in connection with Figure 6.

4 The Kepler oscillator as a test example

Computing the motion of a point mass in the gravitation field of a stationary point mass is what the *Kepler problem* is about. The radial motion in elliptic Kepler orbits (as opposed to parabolic and hyperbolic ones) is oscillatory and can be viewed as the motion of a one-dimensional oscillator which deserves interest as a mechanical example system. Unlike other non-linear example oscillators such as the *Duffing oscillator* and the *Van der Pol oscillator* this system seems to be anonymous. The self-suggesting name *Kepler oscillator* can be found in [15] for this system and will be used in the present article. In the literature this name is, however, more often used for the harmonic oscillator which is related to the Kepler problem by a regularizing transformation, known as the *KS transformation*.

As is well known (e.g. [1], equation (3-14)) the radial Kepler motion is governed by the differential equation

$$m\ddot{r} = -\frac{\partial}{\partial r}\left(-\frac{GMm}{r} + \frac{L^2}{2mr^2}\right) \tag{40}$$

in which L is the constant angular momentum of mass m relative to the position of the space-fixed mass M. Of course, r is the distance between these two masses and G is the constant of gravity. Restricting ourselves to orbits with non-vanishing L and by selecting suitable units of time, mass, and length, we get for the quantities m, GM, and L the common numerical value 1. Writing x for the numerical value of r and v for the

numerical value of $\dot{r}$ we get

$$\dot{x} = v \ , \quad \dot{v} = -\frac{\partial}{\partial x}\left(-\frac{1}{x} + \frac{1}{2x^2}\right) = \frac{1}{x^2}\left(\frac{1}{x} - 1\right) \tag{41}$$

which is the differential equation determined by (2) and also is (since, due to $m = 1$, v is the momentum) the system of canonical equations associated with the Hamiltonian

$$H(v,x) := T(v) + V(x) := \frac{1}{2}v^2 + \frac{1}{x}\left(\frac{1}{2x} - 1\right) \ . \tag{42}$$

This quantity is known to be constant on each orbit. Since, as Figure 7 shows, V attains an absolute minimum at $x = 1$: $V(1) = -\frac{1}{2}$ we have $H(v,x) \geq H(0,1) = -\frac{1}{2}$. We consider only states for which $H(v,x) < 0$ and thus $x > \frac{1}{2}$. These correspond to the elliptical orbits in the Kepler problem; for them the radial motion has an oscillatory character. Kepler's ingenious method for computing the system path for given initial state (not simply the orbit, a subject to which surprisingly many physics texts restrict their interest) can be formulated as a simple algorithm: Given (t_0, x_0, v_0) such that $H_0 := H(v_0, x_0) < 0$ and t_1 we have to go through the following chain of formulas (see also [8], Section 4):

$$\begin{aligned}
a &:= -\frac{1}{2H_0} \text{ (major semi-axis)} \\
\epsilon &:= \sqrt{1 - \frac{1}{a}} \text{ (numerical eccentricity)} \\
n &:= a^{-\frac{3}{2}} \text{ (mean motion)} \\
z &:= 1 - \frac{x_0}{a} + \mathrm{i}\,\frac{x_0 v_0}{\sqrt{a}} \\
E_0 &:= \arg z \text{ (eccentric anomaly)} \\
M_0 &:= E_0 - \epsilon \sin E_0 \text{ (mean anomaly)} \\
M_1 &:= M_0 + (t_1 - t_0)n \\
E_1 &:= \text{ solution of } E_1 = M_1 + \epsilon \sin E_1 \text{ (Kepler's equation)} \\
x_1 &:= a(1 - \epsilon \cos E_1) \\
v_1 &:= \frac{\epsilon a^2 n \sin E_1}{x_1}
\end{aligned} \tag{43}$$

to get the exactly evolved state (t_1, x_1, v_1). Here the solution E of $E = M + \epsilon \sin E$ is given by the algorithm (C++ syntax, `R` is the type for representing real numbers, i.e. `typedef double R;`)

```
R solKepEqu(R M, R eps, R acc)
// M: mean anomaly, eps: numerical eccentricity, acc: accuracy e.g. 1e-8
{
   R xOld=M+1000, xNew=M;
   while ( abs(xOld-xNew) > acc ){
      xOld=xNew;
```

```
      R x1=M+eps*sin(xNew);
      R x2=M+eps*sin(x1);
      xNew=(x1+x2)*0.5;   // My standard provision against oscillations.
         // Works extremely well
   }
   return xNew;
}
```

The computational burden for (43) is independent of the time span $t_1 - t_0$, and it does not matter whether this span is positive (prediction) or negative (retro-diction). Hence, there is no relevant distinction between solution (43) and what normally is referred to as a *closed form solution*. So, in assessing the accuracy of numerical integrators, we have the exact solution always available. In addition to the original leap-frog method and the new asynchronous leap-frog method, we consider two established second order methods for further comparison: The traditional *second order Runge-Kutta method* (e.g. [5], (16.1.2)) and the more modern symplectic *position Verlet integrator*, [4], equation (2.22). For this method there are several names in use, cf. [17], above equation (13), and [12]. The present article refers to it as the *direct midpoint integrator* and recalls its definition for the present simple situation that the forces don't depend on the velocity. Equation (41) can be viewed as a single differential equation of second order

$$\ddot{x} = \frac{1}{x^2}\left(\frac{1}{x} - 1\right) \tag{44}$$

and for convenience of comparison with (29) we write this equation in a form similar to (1) as

$$\ddot{\psi}(t) = F(t,\, \psi(t)) \;, \quad \dot{\psi}(t) =: \phi(t) \;. \tag{45}$$

Since the differential equation is second order, the initial values for ψ *and* ϕ have to come from the problem and the integrator, just as in (29), has the task to promote them both. This is done by formulas very similar to (29):

$$\begin{aligned} \tau &:= \frac{h}{2} \;, \\ t' &:= t + \tau \;, \\ \psi' &:= \psi + \tau\, \phi \;, \\ \underline{\phi} &:= \phi + h\, F(t',\, \psi') \;, \\ \underline{\psi} &:= \psi' + \tau\, \underline{\phi} \;, \\ \underline{t} &:= t' + \tau \;. \end{aligned} \tag{46}$$

If one accepts to have one equation more than necessary, one may take (29) as it stands, and replace the defining equation for ϕ' by the definition $\phi' := \phi + \tau\, F(t',\, \psi')$.

The implementation code for the integrators under consideration is contained in `class KepOsc` in file `tut2.cpp` which is listed in [11].

The orbits of the system can be uniquely parametrized by the values $0 \le \epsilon < 1$ of the numerical eccentricity, which is related to the total energy H_0 through $\epsilon^2 = 1 + 2H_0$ (see

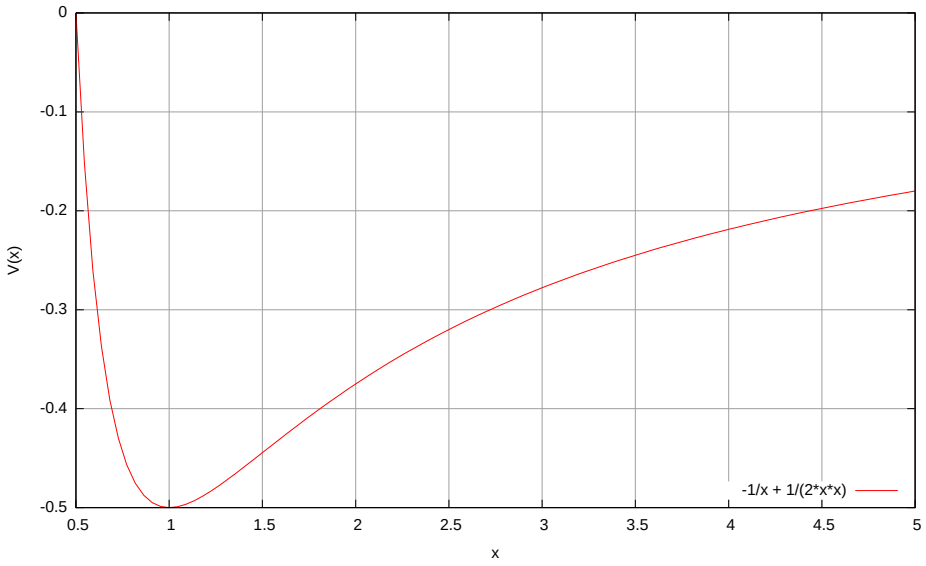


Figure 7: Potential function of the Kepler oscillator.

(43)). The x-values along an orbit range between the solutions $x_{\min}, x_{\max}$ of $V(x) = H_0$ and the v-values range between the solutions $v_{\min}, v_{\max}$ of $T(v) - \frac{1}{2} = H_0$.

Most graphs to be presented here refer to a single path: The one which as an orbit is characterized by $\epsilon = 0.15$, and the initial state of which is the 'perihelion' i.e. $v_0 = 0$ and that $x_0 = x_{\min}$. The oscillation period of this path turns out to be $t_P = 6.501$. Further we easily find $v_{\max} = -v_{\min} = 0.157$ and $x_{\min} = 0.870$, $x_{\max} = 1.176$ which agrees with the location of the oval shape in Figure 8. As time step for stepwise integration we use $h = t_P/32$ to the effect that 32 computed steps cover the whole period and thus would lead back to the initial position if there would be no integration errors. Each computation yields a discrete trajectory of 512 steps, which corresponds to 16 full 'revolutions'. Figure 8 shows that for these data the Runge-Kutta method does not create a periodic orbit and that the orbit in phase space spirals into the outer space. At this graphical resolution, orbits created by the other methods are hard to distinguish. Figure 9 represents the deviation of the computed position from the exact one for the four methods under consideration. What is displayed here is not simply the difference in phase space location but the phase space position that occurs if the state at time t is back-evolved via the exact dynamics to the initial time $t = 0$. If the stepwise integration would not introduce an error, the point to be displayed would come out as $(0, 0)$ in all cases. The errors express themselves as curves (paths) with parameter t and a longer curve indicates a larger total error after the whole integration. Although the dependence on the curve parameter t is not shown in the curves (only the orbit is represented), the 16 approximately repeated substructures in these curves show how the error evolves from revolution to revolution. The coordinates in these diagrams are indexed 'relative' which means that x-differences are divided by $x_{\max} - x_{\min}$ and v-differences are divided

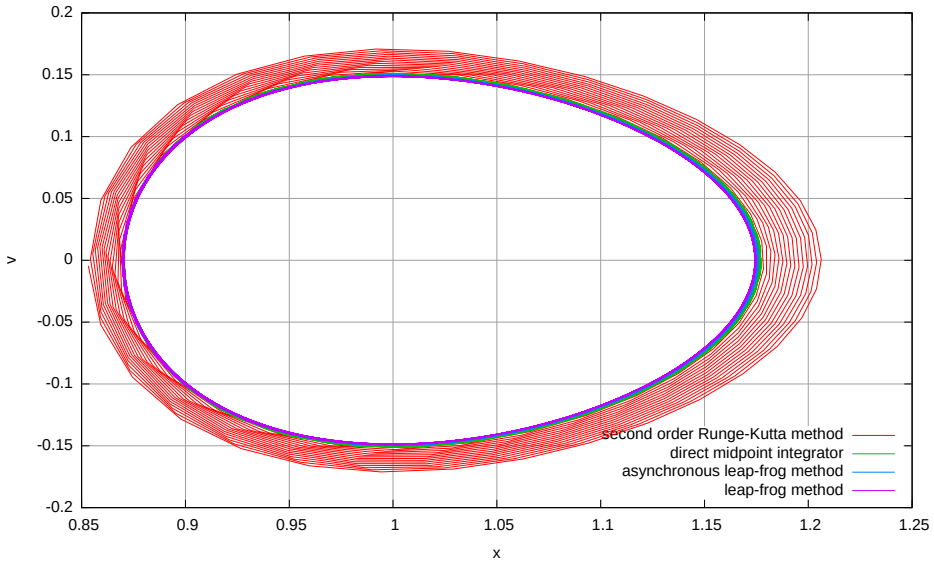


Figure 8: Computed orbits in phase space.

by $v_{\max} - v_{\min}$. As already pointed out in [9], near equation (92), these curves can be interpreted as paths in an *interaction picture* dynamics, which again is a dynamical system. This kind of interaction picture considers the stepwise integration as the combined action of the exact evolution and a 'discretization interaction' (in analogy to considering a digitized signal as a superposition of the original analog signal and 'digitization noise'). It may therefore be fittingly referred to as *numerical interaction picture*. The more accurate the stepwise integration method, the weaker is the interaction and the slower is the motion seen in the numerical interaction picture. As mentioned in [9], this diagnostics based on the numerical interaction picture is not restricted to systems for which the exact solution is directly accessible; an access through stepwise back-evolution methods is sufficient if these are, say, two orders of magnitude more accurate than the method under investigation.

This numerical interaction picture dynamics is related to the usage of evolution operators $\mathrm{e}^{\mathrm{i}Ht}\,\mathrm{e}^{-\mathrm{i}H_0 t}$ in quantum mechanical scattering theory and with backward error analysis in numerical analysis of differential equations, [6], Chapter 10 , [12], Section 4, [13], Chapter 5. The aim of *backward error analysis* is to represent the discretization interaction by additional terms to the right-hand side of the differential equation; in case of Hamiltonian systems by an addition to the Hamiltonian, which is the way for introducing interaction physicists are most familiar with. The idea of the presently proposed method is to work with the dynamical systems directly without being forced to construct an equivalent Hamiltonian, or — more generally — a modified equation. The two movies [10] 'Deformation of a phase space subset by interaction picture dynamics' show features which one would not easily read from any modified equation of backward error analysis (the converse is probably also true; it is the multitude of non-trivial ob-

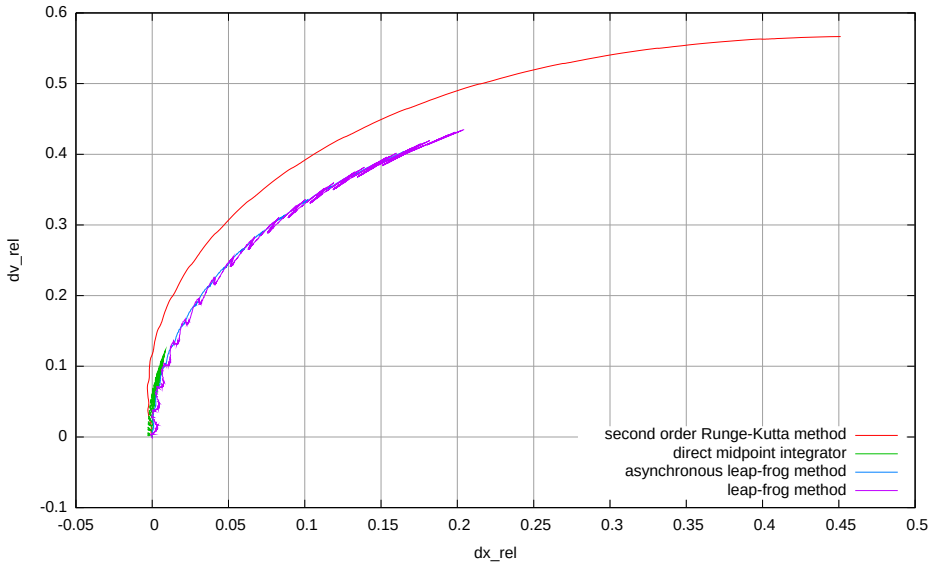


Figure 9: Integration error of four computational methods in the 'numerical interaction picture'.

servations which enhances our understanding). Already our trajectory representation in Figures 9, 10, 11, 12 shows more morphologic features than the energy-error curves that are normally generated as a kind of fingerprint of an integrator (e.g. [13], Fig 4.1).

Since, as already clear from Figure 8, the error of the Runge-Kutta method is much larger than the error of the other methods, the following Figure 10 gives the corresponding representation for the more accurate methods only. This figure suggests that the direct midpoint integrator is by far more accurate than the two leap-frog integrators. We will see now that this suggestion is misleading. In the application considered in [18] it was found that the asynchronous leap-frog integrator showed similar step size requirements as the direct midpoint integrator when a actual leap-frog step was defined as consisting of two leap-frog steps of half the step size. Also in the present context it makes sense to consider such a subdivision of a step. We thus define the *densified* leap-frog integrators as

$$\begin{aligned} \tilde{\mathcal{L}} &:= \mathcal{L} \circ \mathcal{L} \\ \tilde{\mathcal{A}}_h &:= \mathcal{A}_{h/2} \circ \mathcal{A}_{h/2} \end{aligned} \tag{47}$$

and display the resulting error orbits in Figure 11. Note that again there are 32 integration steps per orbit, but these are made from substeps so that only every second computed step results in a graphical point. For such a combined step, the computational burden is the same as for one second order Runge-Kutta step, but the accuracy is much better than for Runge-Kutta. It is plausible that only this densified version of the leap-frog methods turns out to come close to the accuracy of the direct midpoint method: The latter has direct access to the second derivative of the solution, whereas

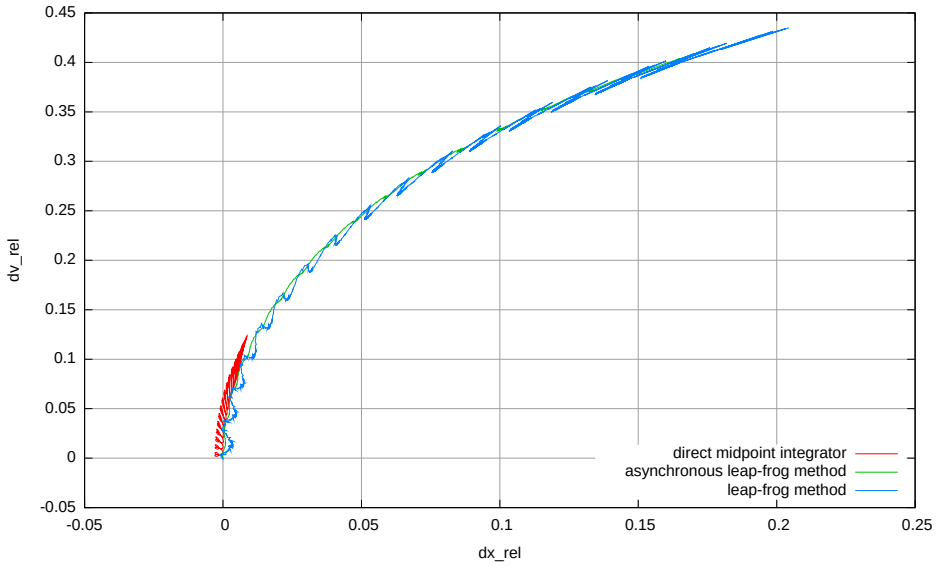


Figure 10: Integration error of the better methods in the 'numerical interaction picture'.

the leap-frog methods only accesses the first derivative and thus can be viewed as simulating access to the second derivative by evaluating the first derivative at two different points. One may generate corresponding diagrams for different values of eccentricity and step size and will experience a surprising morphological stability of the curves and their relative length. Figure 12 is an example for this. Here, the number of points per revolution is increased to 64 in response to the increased value of the eccentricity ϵ.

It is rather evident from all such graphs is that the asynchronous leap-frog method has shorter and more regular error orbits than the standard leap-frog method.

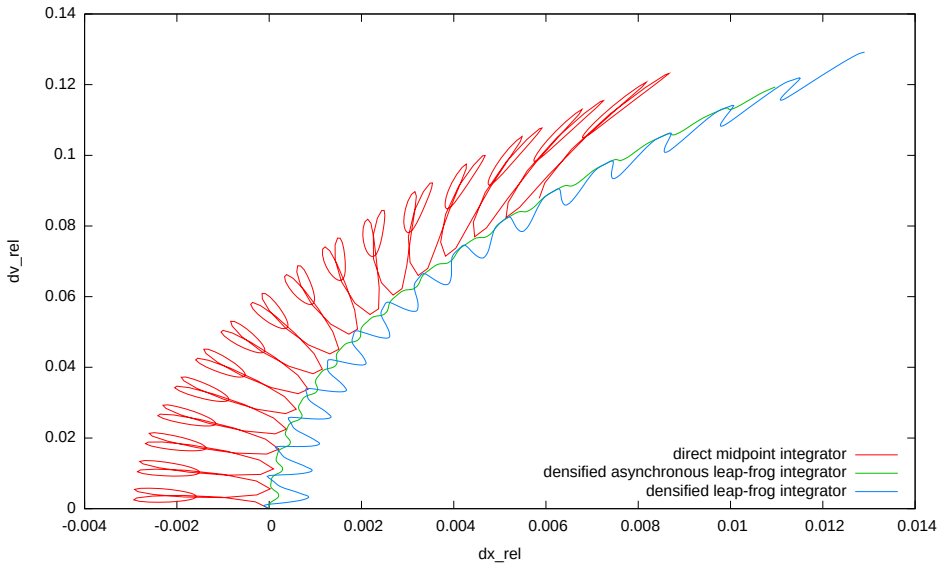


Figure 11: Integration error of the direct midpoint method together with the densified leap-frog methods in the 'numerical interaction picture' for $\epsilon = 0.15$. Sixteen periods at 32 steps per period.

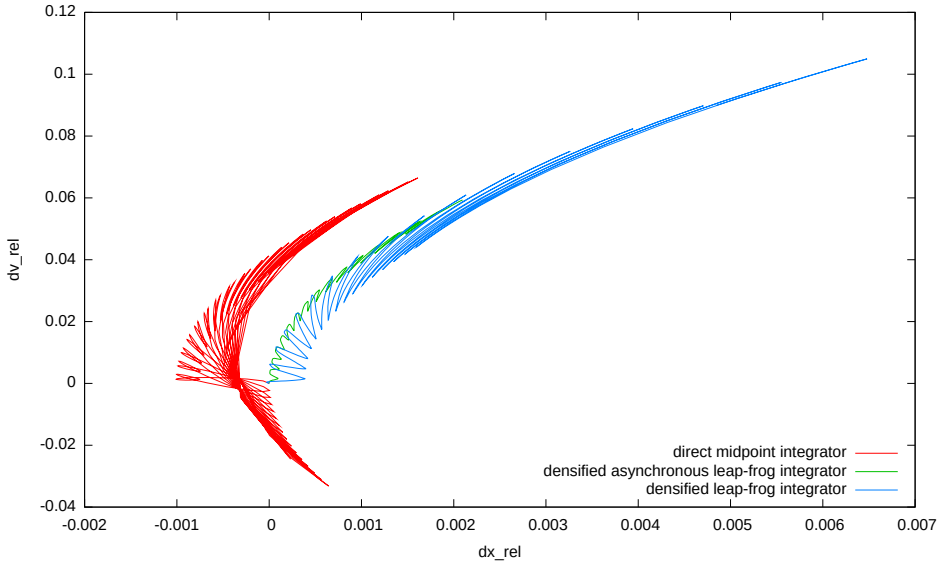


Figure 12: Integration error of the direct midpoint method together with the densified leap-frog methods in the 'numerical interaction picture' for $\epsilon = 0.30$. Sixteen periods at 64 steps per period.

Acknowledgment

I am grateful to Domenico Castrigiano for many discussions on the relation of discrete mathematics to classical analysis and to Ernst Hairer for a valuable comment.

References

[1] Herbert Goldstein: Klassische Mechanik, Akademische Verlagsgesellschaft, 1963

[2] A. Askar and S. Cakmak, J. Chem. Phys. 68, 2794 (1978)

[3] H. Tal-Ezer and R. Kosloff: An accurate and efficient scheme for propagating the time dependent Schrödinger equation, J. Chem. Phys. 81 (9) 3967-3971 (1984)

[4] M. Tuckerman, B.J. Berne, G.J. Martyna: Reversible multiple time scale dynamics, J. Chem. Phys. Vol. 97(3) pp. 1990 - 2001, 1992, Equation (2.22)

[5] William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery: Numerical Recipes in C, The Art of Scientific Computing, Second Edition, Cambridge University Press 1992

[6] J.M. Sanz-Serna and M.P. Calvo: Numerical Hamiltonian Problems, Applied Mathematics and Mathematical Computation 7, Chapman & Hall 1994

[7] A.M. Stuart and A.R. Humphries: Dynamical Systems and Numerical Analysis, Cambridge University Press, 1996

[8] Ulrich Mutze: Predicting Classical Motion Directly from the Action Principle II Mathematical Physics Preprint Archive 1999–271
www.ma.utexas.edu/mp_arc/c/99/99-271.pdf

[9] Ulrich Mutze: A Simple Variational Integrator for General Holonomic Mechanical Systems, Mathematical Physics Preprint Archive 2003–491
www.ma.utexas.edu/mp_arc/c/03/03-491.pdf

[10] Ulrich Mutze: homepage
http://www.ulrichmutze.de/interactionpicturemovie1/intact1.html
http://www.ulrichmutze.de/interactionpicturemovie2/intact2.html

[11] Ulrich Mutze: homepage
http://www.ulrichmutze.de/softwaredescriptions/tut.pdf

[12] Ernst Hairer, Christian Lubich, Gerhard Wanner: Geometric numerical integration illustrated by the Störmer/Verlet method, Acta Numerica (2003) pp. 1-51 Cambridge University Press, 2003

[13] Benedict Leimkuhler and Sebastian Reich: Simulating Hamiltonian Dynamics, Cambridge Monographs on Applied and Computational Mathematics, Cambridge University Press 2004

[14] T. Holder, B. Leimkuhler, and S. Reich: Explicit, time-reversible and variable step size integration. Appl. Numer. Math., 39:367-377,2001.
http://opus.kobv.de/zib/volltexte/1998/361/pdf/SC-98-17.pdf

[15] G. Gallavotti: Classical Mechanics
http://ipparco.roma1.infn.it/pagine/deposito/2005/MC.ps.gz

[16] Ulrich Mutze: The direct midpoint method as a quantum mechanical integrator, Mathematical Physics Preprint Archive 2006–356
www.ma.utexas.edu/mp_arc/c/06/06-356.pdf (2006)

[17] Ulrich Mutze: The direct midpoint method as a quantum mechanical integrator II, Mathematical Physics Preprint Archive 2007–176
www.ma.utexas.edu/mp_arc/c/07/07-176.pdf

[18] Ulrich Mutze: Separated quantum dynamics
Mathematical Physics Preprint Archive 2008–69
www.ma.utexas.edu/mp_arc/c/08/08-69.pdf

Last modification: 2014-06-07