An asynchronous leapfrog method II

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 leapfrog 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, wich refer to different times. 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 leapfrog method gives a unique prescription how to evolve them stepwise.

The stability properties of this modified method are the same as for the original one: the set of absolute stability is the interval $[-\mathrm{i}\,,+\mathrm{i}\,]$ on the imaginary axis. This implies exponential growth of trajectories in situations where the exact trajectory has an asymptote.

By considering new evolution steps that are composed of two consecutive old evolution steps we can average over the velocities of the sub-steps and get an integrator with a much larger set of absolute stability, which is immune to the asymptote problem.

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 system the exact solution is cheaply computable. Depending on the selected initial conditions, widely varying degrees of stiffness hold. For this system the proposed leapfrog methods are compared with the original leapfrog method, three explicit second order Runge-Kutta methods, and a Størmer-Verlet method. In all tested cases the asynchronous leapfrog method was more accurate than the original method, in some cases considerably so.

*www.ulrichmutze.de

A 'numerical interaction picture' is introduced, in which the evolution of a dynamical system and its evolution as brought about by a numerical integrator are treated in analogy to the dynamics with and without interaction in the interaction picture of quantum mechanics. The trajectories of a system in this representation are indicative for the accuracy of an integrator and provide a kind of a fingerprint of it.

For non-stiff initial conditions, the asynchronous leapfrog method turns out to be the most accurate among the methods under consideration. The averaged version behaves very robust for stiff initial conditions since it is not reversible and slightly dissipative. It is well suited as a general-purpose method for rough real-world problems.

Finally a automatic step control scheme is demonstrated. It works for both the asynchronous leapfrog methods and the Runge-Kutta methods. Here, averaging in the leapfrog method makes no difference and leapfrog methods are by a factor of about 4 more accurate than the Runge-Kutta methods.

AMS Subject Classification (MSC2010): 65L04, 65L05, 65L07, 65L12, 65P10, 65Q10

Keywords: leapfrog integrator, numerical initial value problem, Kepler oscillator, symplectic integrator, interaction picture, stiff problem, set of absolute stability

1 Preface

This work originated from an expanded form of [1] which is accessible from my homepage as [2]. A forum discussion in [3] based on [2] stressed the instability properties of leapfrog methods and convinced me that the asynchronous leapfrog integrator is unsuitable as a general-purpose workhorse method unless its stability can be improved. Most instructive in this respect were the Lotka-Volterra equations proposed to me by H.E. Lehtihet. Fortunately the densified form of the asynchronous leapfrog integrator suggests an computationally cheap averaging step which improves stability drastically. Definition of this new integrator and demonstration of its properties is the main topic of this work. Although meanwhile I worked out many applications to higher-dimensional problems and, with the spherical pendulum and the spinning top, also two with non-trivial geometric structure of its configuration space, I decided to include no new systems and to leave sections 3 to 5 of [2] virtually unchanged. The material concerning the new integrator and a study of stability in all methods under consideration is added as two following sections. Of course, Abstract, Introduction, Acknowledgments, and References are changed to fit the present form of the article.

2 Introduction

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

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

for a time-dependent quantity ψ which takes values in a real finite-dimensional vector space $\mathcal{H}$. Here F is a function $\mathbb{R} \times \mathcal{H} \rightarrow \mathcal{H}$ [1] . Equations of this kind arise in various contexts:

1. Ordinary differential equations of finite order for real or complex variables.

2. Dynamical systems on finite-dimensional real or complex manifolds; in this case $\mathcal{H}$ is a space of coordinate values (the co-domain of the charts from a suitable atlas), and F is the coordinate representation of a vector field.

3. Discrete approximations to partial differential equations like the time-dependent Schrödinger equation or Maxwell's equations.

Although the parameter t will always be referred to as 'time', it could have a different meaning as in the following examples: length of a rod as a function of temperature, position of a point on a curve as a function of the arc length separating it from some reference point on the curve.

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 $t \mapsto \psi(t)$, $\psi(t_0) = \psi_0$, 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 leapfrog algorithm, the present article develops and analyzes an economic and robust asynchronous solution of the computational initial value problem associated with (1). Section 3 recalls the leapfrog method, and Section 4 carries out the general development of the new asynchronous algorithm. Section 5 will apply this integration method and related methods to the differential equation defined by (39). Section 6 defines the averaged form of the asynchronous leapfrog integrator and analyzes the stability of it and of of the second order explicit Runge-Kutta methods. Section 7 tests the new methods by applying them to the differential equation defined by (39).

[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 form.

3 The leapfrog method

A marvelously simple synchronous solution algorithm for the computational initial value problem of (1) is the *leapfrog method* or *explicit midpoint rule*, see e.g. [10], 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 [5] and the citation of this work in [6]). It is most conveniently considered as the map

$$\begin{aligned} \mathcal{L} \,:\, (\mathbb{R}\times\mathcal{H})\times(\mathbb{R}\times\mathcal{H}) &\to (\mathbb{R}\times\mathcal{H})\times(\mathbb{R}\times\mathcal{H}) \\ ((t_0,\psi_0),(t_1,\psi_1)) &\mapsto ((t_1,\psi_1),(t_2,\psi_2))\;, \quad . \end{aligned} \tag{2}$$

where

$$\begin{aligned} t_2 &:= 2t_1 - t_0\;, \\ \psi_2 &:= \psi_0 + (t_2 - t_0)\,F(t_1,\,\psi_1)\;. \end{aligned} \tag{3}$$

The equivalent form

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

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

$$F(t_1,\,\psi(t_1)) = \dot{\psi}(t_1) = \frac{\psi(t_2)-\psi(t_0)}{t_2-t_0} + O((t_2-t_0)^3)\;. \tag{5}$$

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

$$(t_{i+1},\psi_{i+1}) = \pi_2(\mathcal{L}((t_{i-1},\psi_{i-1}),(t_i,\psi_i))) = \pi_2(\mathcal{L}^i((t_0,\psi_0),(t_1,\psi_1)))\;, \tag{6}$$

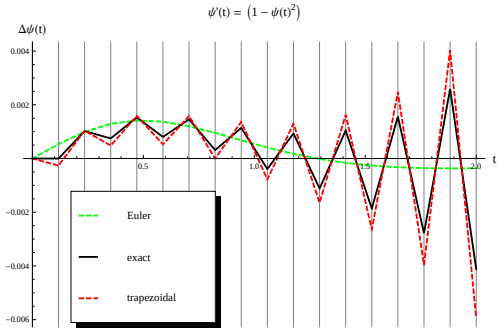
where π_2 is the canonical projection to the second component of a pair. According to the initial value problem we are given t_0, t_1 (which determines, due to the assumed synchronicity, all further t-values) and ψ_0. For starting iteration (6) we need also ψ_1. This has to be added in a way consistent with (1), e. g. by employing the *explicit Euler rule*

$$\psi_1 := \psi_0 + (t_1 - t_0)\,F(t_0,\,\psi_0)\;. \tag{7}$$

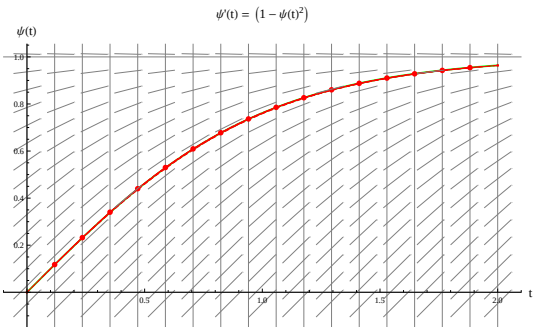
One could expect (as I did for some time) that being more accurate here would improve the accuracy of the leapfrog trajectory. The optimum definition could be expected to be

$$\psi_1 := \psi(t_1)\;, \tag{8}$$

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 (7) and (8) is given by employing the implicit *trapezoidal rule*

$$\psi_1 = \psi_0 + (t_1 - t_0)\,\frac{F(t_0,\,\psi_0) + F(t_1,\,\psi_1)}{2}\;, \tag{9}$$

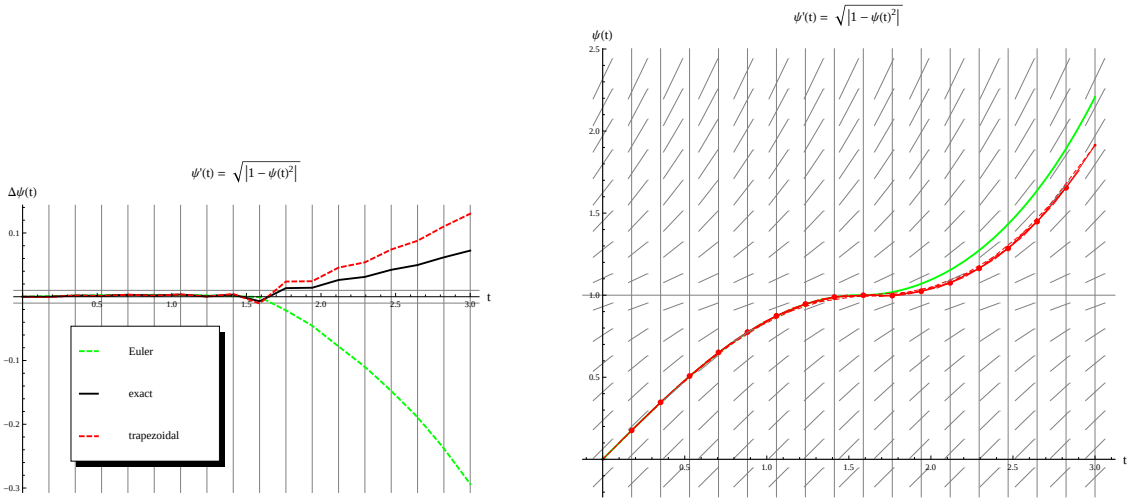
which can be efficiently solved by the iteration

$$\psi_1^{(i+1)} := \psi_0 + (t_1 - t_0)\,\frac{F(t_0,\,\psi_0) + F(t_1,\,\psi_1^{(i)})}{2}\;, \quad \psi_1^{(0)} := \psi_0\;. \tag{10}$$

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:

$$\dot\psi(t) = 1 - \psi(t)^2\;, \quad \psi(0) = 0 \quad \text{thus} \quad \psi(t) = \tanh(t)\;. \tag{11}$$

Figure 1 shows the difference between the exact trajectory and the leapfrog trajectories according to the three initialization methods introduced above. Surprisingly the less sophisticated method (7) 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 (6) can be used to define a leapfrog trajectory. This then is typically a zigzag line which tends to wiggle around a trajectory of differential equation (1) and initial data (t_0, ψ_0). See Figure 3, where the leapfrog trajectory of Figure 1(b) is modified by shifting ψ_1 considerably from its correct position. Notice the graphical manifestation of the leapfrog 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 (7) as done in Figure 1(b)) the lower the zigzag 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 (2), 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 leapfrog method to be defined as a map of states into states. Only a subset of this state space (e. g. the one given by (7)) corresponds to potential initial states of the continuous system (1).

Equation (6) defines ψ_i for arbitrarily large i, whereas equation (1) may drive a trajectory in finite time into infinity. A well-known example is

$$\dot{\psi}(t) = 1 + \psi(t)^2 \,, \quad \psi(0) = 0 \quad \text{thus} \quad \psi(t) = \tan(t) \quad \text{and} \quad \psi(\pi/2) = \infty \,. \tag{12}$$

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 leapfrog algorithm (3) to the leapfrog 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

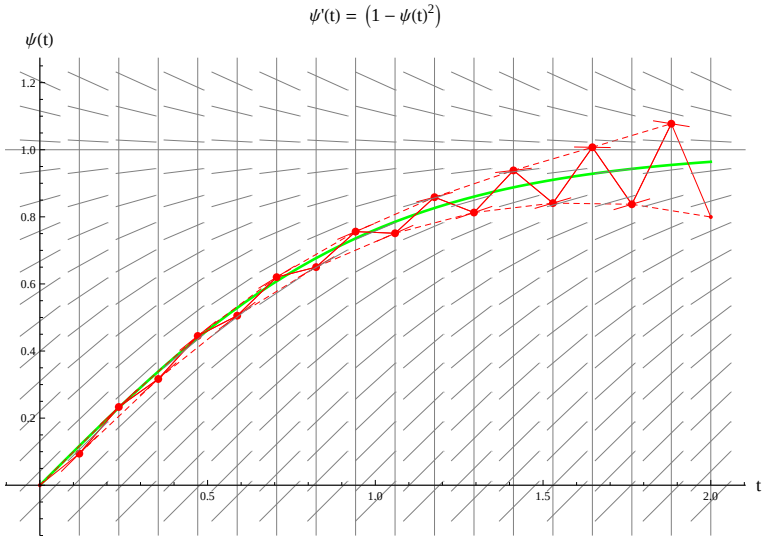


Figure 3: Zigzag trajectory resulting from intentionally spoiled initialization.

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

$$\begin{aligned} \mathcal{T} \;:\; (\mathbb{R} \times \mathcal{H}) \times (\mathbb{R} \times \mathcal{H}) \;&\to\; (\mathbb{R} \times \mathcal{H}) \times (\mathbb{R} \times \mathcal{H}) \\ ((t_0, \psi_0), (t_1, \psi_1)) \;&\mapsto\; ((t_1, \psi_1), (t_0, \psi_0)) \;. \end{aligned} \tag{13}$$

then we easily verify

$$\mathcal{L} \circ \mathcal{T} \circ \mathcal{L} = \mathcal{T} \;, \quad \mathcal{T} \circ \mathcal{T} = \mathbf{1} \tag{14}$$

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 leapfrog trajectory all previous components (t_k, ψ_k), $k < n - 1$,

$$(t_k, \psi_k) = \pi_2(\mathcal{T}\mathcal{L}^{n-k}\mathcal{T}((t_{n-1}, \psi_{n-1}), (t_n, \psi_n))) \;. \tag{15}$$

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

$$((t_p, \psi_p), (t_p + \tau, \psi_p + \tau F(t_p, \psi_p)) \tag{16}$$

or a potentially more accurate form which also involves ψ_{p-1} (which equation (16) simply forgets).

4 An asynchronous version of the leapfrog method

What I intend here, is to modify the leapfrog method in a way that no trade-offs 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 leapfrog method has been introduced in [20] and applied to time-dependent Hartree equations in [22].

We consider four consecutive components of a leapfrog trajectory of (1)

$$(t_k,\, \psi_k)\ , \quad (t_{k+1},\, \psi_{k+1})\ , \quad (t_{k+2},\, \psi_{k+2})\ , \quad (t_{k+3},\, \psi_{k+3}) \tag{17}$$

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

$$\phi_k := \frac{\psi_{k+1} - \psi_k}{\tau}\ , \quad \phi_{k+1} := F(t_{k+1},\, \psi_{k+1})\ , \quad \phi_{k+2} := \frac{\psi_{k+2} - \psi_{k+1}}{\tau}\ . \tag{18}$$

From this definition and from (4) we obtain

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

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:

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

Equation (19) allows us to give the last two equations of (20) a more symmetrical form:

$$\begin{aligned}
\phi_{k+2} &= 2\, \phi_{k+1} - \phi_k\ , \\
\psi_{k+2} &= \psi_{k+1} + \tau\, \phi_{k+2}\ .
\end{aligned} \tag{21}$$

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 leapfrog law, the iteration of $\mathcal{A}_{2\tau}$ does not exactly continue the leapfrog trajectory. Actually, we have $\chi_{k+3} = \psi_{k+3}$ only up to terms of order τ^2:

$$\begin{aligned} \psi_{k+3} &= \psi_{k+1} + 2\,\tau\, F(t_{k+2},\, \psi_{k+2})\ , \\ \chi_{k+3} &= \psi_{k+2} + \tau\,\phi_{k+2} = 2\,\psi_{k+2} - \psi_{k+1} = \psi_{k+1} + 2\,(\psi_{k+2} - \psi_{k+1}) \end{aligned} \tag{22}$$

and thus

$$(\psi_{k+3} - \chi_{k+3})/2 = \tau\, F(t_{k+2},\, \psi_{k+2}) + \psi_{k+1} - \psi_{k+2} = O(\tau^2)\ . \tag{23}$$

The reason for this behavior lies in the fact that in going from the normal leapfrog 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 leapfrog 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:

$$\phi_0 := F(t_0,\, \psi_0)\ . \tag{24}$$

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

$$((t_0, \psi_0, \phi_0), (t_1, \psi_1, \phi_1), (t_2, \psi_2, \phi_2), \ldots) \tag{25}$$

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

$$(t_{i+1}, \psi_{i+1}, \phi_{i+1}) := \mathcal{A}_{h_i}(t_i, \psi_i, \phi_i)\ , \quad h_i := t_{i+1} - t_i\ . \tag{26}$$

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

$$\begin{aligned} \mathcal{A}_h\ :\ \mathbb{R}\times\mathcal{H}\times\mathcal{H}\ &\to\ \mathbb{R}\times\mathcal{H}\times\mathcal{H} \\ (t, \psi, \phi)\ &\mapsto\ (\underline{t}, \underline{\psi}, \underline{\phi})\ . \end{aligned} \tag{27}$$

can be seen from (20) and (21) to be defined by the following chain of formulas:

$$\begin{aligned} \tau &:= \frac{h}{2}\ , \\ t' &:= t + \tau\ , \\ \psi' &:= \psi + \tau\,\phi\ , \\ \phi' &:= F(t',\, \psi')\ , \\ \underline{\phi} &:= 2\,\phi' - \phi\ , \\ \underline{\psi} &:= \psi' + \tau\,\underline{\phi} = \psi + 2\tau\,\phi'\ , \\ \underline{t} &:= t' + \tau\ . \end{aligned} \tag{28}$$

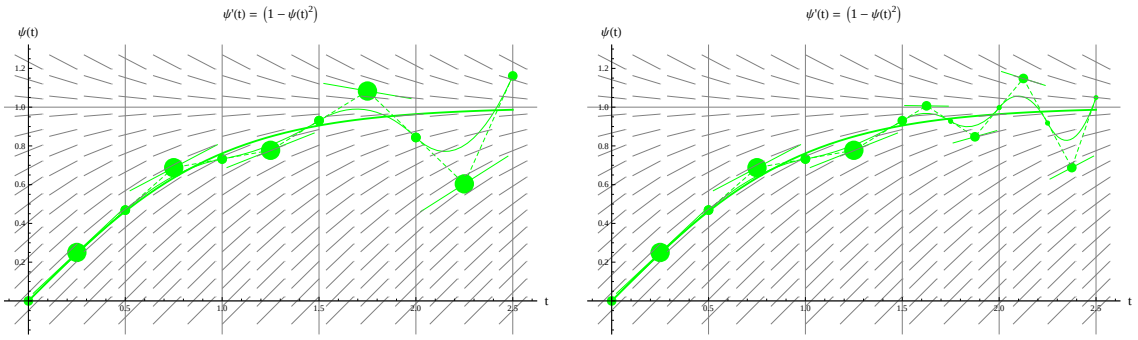
This algorithm corresponds to equation (8) in [20] but is more general since it does not assume the special form of F that was considered there. Notice that the leapfrog midpoint state data t', ϕ', ψ' appear only as intermediary quantities that help to give the algorithm an elegant form. In particular, they do not belong to the discrete trajectory (25), (26) generated by $\mathcal{A}$. Their geometrical role becomes clear from the representation of the final state as

$$\underline{\psi} = \psi + h\,\phi + \frac{h^2}{2}\frac{\phi' - \phi}{\tau}\ , \quad \underline{\phi} = \phi + h\frac{\phi' - \phi}{\tau}\ . \tag{29}$$

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 (28) 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 (28) and reduction of the time step.

and memorized:

$$\begin{aligned} t &+= \tau \,, \\ \psi &+= \tau\, \phi \,, \\ \phi &+= 2\,\lambda\,(F(t,\,\psi) - \phi) \,, \\ \psi &+= \tau\, \phi \,, \\ t &+= \tau \,, \end{aligned} \tag{30}$$

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 (30) to be equivalent to (28). 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 (11)) 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 (28) rewritten in the self-explanatory style of [15](which

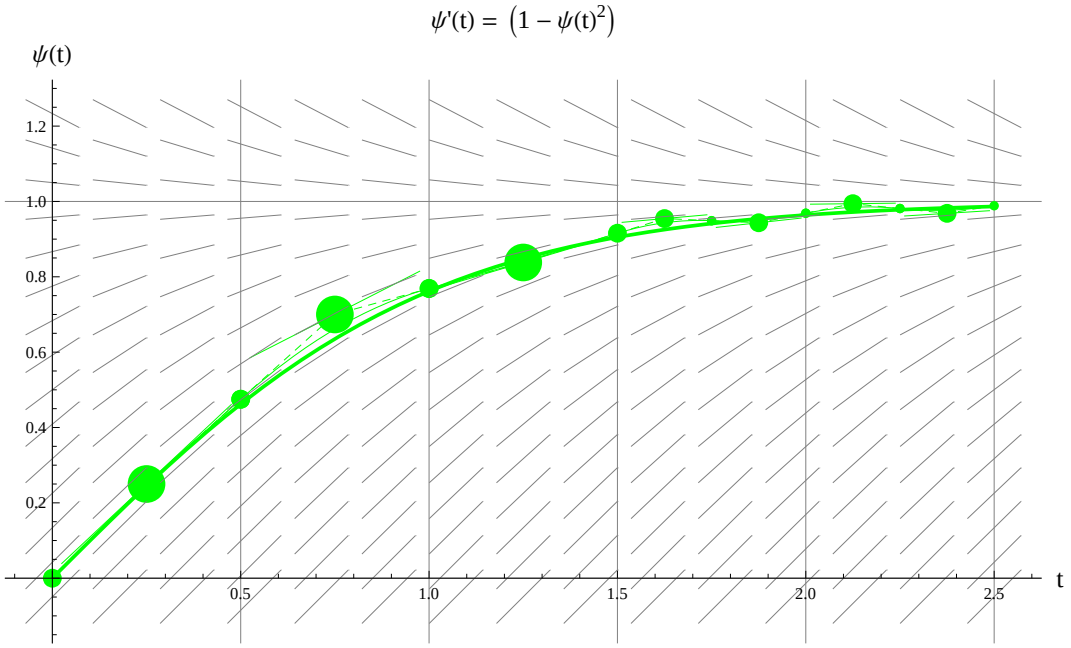
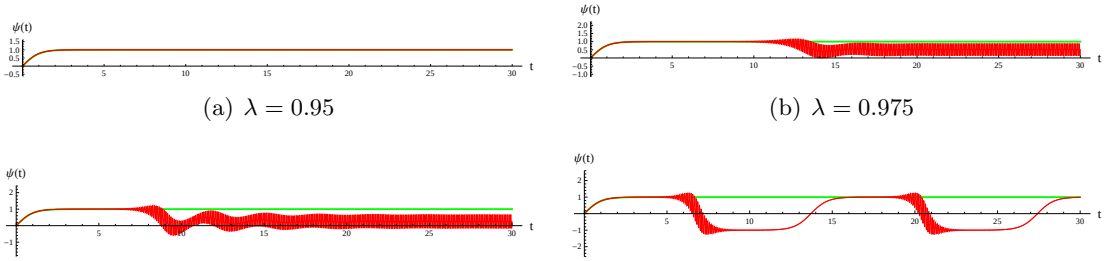


Figure 5: Situation of Figure 4(b) with relaxation ($\lambda = 0.8$).

is similar to that in [16]): We write our differential equation (1) as

$$\dot{q} = f(t, q) \tag{31}$$

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 $n+1$:

$$\boxed{\begin{aligned} &t_{n+\frac{1}{2}} = t_n + \frac{h}{2}\,, \quad q_{n+\frac{1}{2}} = q_n + \frac{h}{2}\,v_n\,, \quad v_{n+\frac{1}{2}} = f(t_{n+\frac{1}{2}},\, q_{n+\frac{1}{2}})\,, \\ &v_{n+1} = 2\,v_{n+\frac{1}{2}} - v_n\,, \quad q_{n+1} = q_{n+\frac{1}{2}} + \frac{h}{2}\,v_{n+1} = q_n + h\,v_{n+\frac{1}{2}}\,, \quad t_{n+1} = t_n + h\,. \end{aligned}} \tag{32}$$

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 (28) 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

$$\begin{aligned} B_h \,:\, \mathbb{R}\times\mathcal{H}\times\mathcal{H} \;&\to\; \mathbb{R}\times\mathcal{H}\times\mathcal{H} \\ (t,\psi,\phi) \;&\mapsto\; (t+h,\, \psi + h\,\phi,\, \phi) \end{aligned} \tag{33}$$

and the mapping

$$\begin{aligned} C \,:\, \mathbb{R}\times\mathcal{H}\times\mathcal{H} \;&\to\; \mathbb{R}\times\mathcal{H}\times\mathcal{H} \\ (t,\psi,\phi) \;&\mapsto\; (t,\psi,\, 2\,F(t,\psi) - \phi)\,. \end{aligned} \tag{34}$$

We then see immediately

$$\mathcal{A}_h = B_{h/2} \circ C \circ B_{h/2}\,. \tag{35}$$

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 (28)) 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 (35) then implies that also $\mathcal{A}_h$ is skew-symplectic. Obviously the product of two skew-symplectic maps is symplectic. *Therefore, the densified form (46) 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:

$$\begin{aligned} \dot{\psi} &= \phi\,, \\ \dot{\phi} &= \frac{\partial F}{\partial t} + \frac{\partial F}{\partial \psi}\,\phi\,. \end{aligned} \tag{36}$$

Using integrator (28) for this equation may be non-trivial since the integrator uses F whereas the differential equation (36) 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 Størmer-Verlet integrator is the path which led me to (32). I followed this path in order to simulate quantum mechanical systems [19], [20].

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

$$\mathcal{A}_{-h} \circ \mathcal{A}_h = \mathcal{A}_0 \tag{37}$$

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 leapfrog 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 [20] subsequent to equation (10). It is to be noted that there is no motion reversion operator comparable to (13). 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 (14). One should also note that the concept (37) does not assume that the differential equation (1) satisfies any reversibility condition, in particular not the one assumed in [17], 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 leapfrog state $(t - h, \psi - h\phi), (t, \psi)$ which, by successive application of $\mathcal{L}$, creates a leapfrog 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 leapfrog methods can be made even more striking: The leapfrog state

$$(t - \tau, \psi - \tau\phi), (t + \tau, \psi + \tau\phi) \ , \quad \tau := \frac{h}{2} \tag{38}$$

and the 'asynchronous leapfrog state' (t, ψ, ϕ) can easily be seen to generate nearly identical discrete trajectories by application of their respective integrators: The leapfrog configurations (t_i, ψ_i) coincide with the 'mid-configurations' $(t_{i+\frac{1}{2}}, q_{i+\frac{1}{2}})$ of the asynchronous leapfrog 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 the 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 (7) or (9) is probably considered more natural than the symmetric choice (38). 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 (45) for differential equations (44) of second order to the more general situation (1). The close relation of the arising method to the leapfrog method became apparent only later.

With the integrator (32) we have achieved for the general differential equation the status that the Størmer-Verlet method (e.g. [16] 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.

5 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 [18] 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. [4], equation (3-14)) the radial Kepler motion is governed by the differential equation

$$m\ddot{r} = -\frac{\partial}{\partial r}\left(-\frac{GMm}{r} + \frac{L^2}{2mr^2}\right) \tag{39}$$

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

$$\dot{x} = v \ , \quad \dot{v} = -\frac{\partial}{\partial x}\left(-\frac{1}{x} + \frac{1}{2x^2}\right) = \frac{1}{x^2}\left(\frac{1}{x} - 1\right) \tag{40}$$

which is the differential equation determined by (39) and also is (since, due to $m = 1$, v is the momentum) the system of canonical equations associated with the Hamiltonian

$$H(v,x) := T(v) + V(x) := \frac{1}{2}v^2 + \frac{1}{x}\left(\frac{1}{2x} - 1\right) \ . \tag{41}$$

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 [11], Section 4):

$$\begin{aligned}
a &:= -\frac{1}{2H_0} \text{ (major semi-axis)} \\
\epsilon &:= \sqrt{1 - \frac{1}{a}} \text{ (numerical eccentricity)} \\
n &:= a^{-\frac{3}{2}} \text{ (mean motion)} \\
z &:= 1 - \frac{x_0}{a} + \mathrm{i}\,\frac{x_0 v_0}{\sqrt{a}} \\
E_0 &:= \arg z \text{ (eccentric anomaly)} \\
M_0 &:= E_0 - \epsilon \sin E_0 \text{ (mean anomaly)} \\
M_1 &:= M_0 + (t_1 - t_0)n \\
E_1 &:= \text{ solution of } E_1 = M_1 + \epsilon \sin E_1 \text{ (Kepler's equation)} \\
x_1 &:= a(1 - \epsilon \cos E_1) \\
v_1 &:= \frac{\epsilon a^2 n \sin E_1}{x_1}
\end{aligned} \tag{42}$$

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 (42) 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 (42) 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 leapfrog method and the new asynchronous leapfrog method, we consider two established second order methods for further comparison: The traditional *second order Runge-Kutta method* (e.g. [8], (16.1.2)) and the more modern symplectic *position Verlet integrator*, [7], equation (2.22). For this method there are several names in use, cf. [20], above equation (13), and [15]. 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 (40) can be viewed as a single differential equation of second order

$$\ddot{x} = \frac{1}{x^2}\left(\frac{1}{x} - 1\right) \tag{43}$$

and for convenience of comparison with (28) we write this equation in a form similar to (1) as

$$\ddot{\psi}(t) = F(t,\, \psi(t)) \;, \quad \dot{\psi}(t) =: \phi(t) \;. \tag{44}$$

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 (28), has the task to promote them both. This is done by formulas very similar to (28):

$$\begin{aligned} \tau &:= \frac{h}{2} \;, \\ t' &:= t + \tau \;, \\ \psi' &:= \psi + \tau\, \phi \;, \\ \underline{\phi} &:= \phi + h\, F(t',\, \psi') \;, \\ \underline{\psi} &:= \psi' + \tau\, \underline{\phi} \;, \\ \underline{t} &:= t' + \tau \;. \end{aligned} \tag{45}$$

If one accepts to have one equation more than necessary, one may take (28) 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 [14].

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

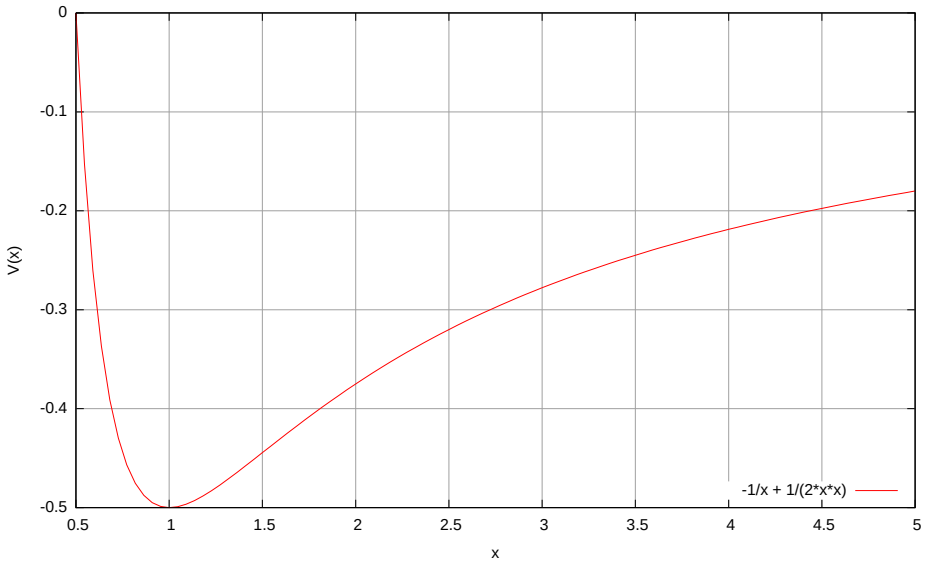


Figure 7: Potential function of the Kepler oscillator.

(42)). 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

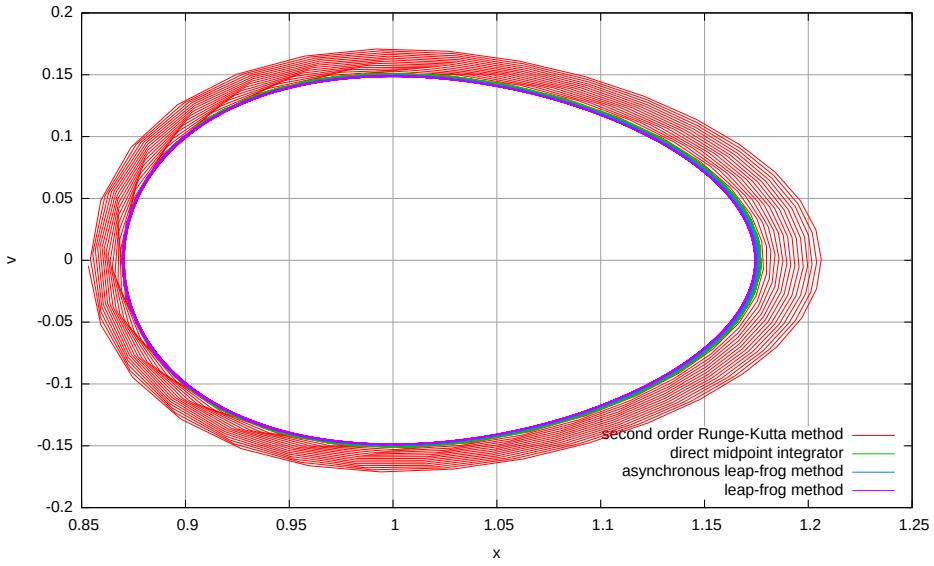


Figure 8: Computed orbits in phase space.

by $v_{\max} - v_{\min}$. As already pointed out in [12], 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 [12], 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, [9], Chapter 10 , [15], Section 4, [16], 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 [13] 'Deformation of a phase space subset by interaction picture dynamics' show features [2] which one would not easily read from any modified equation of back-

[2] If one computes numerically the area enclosed by the curves which are shown in the movies, one

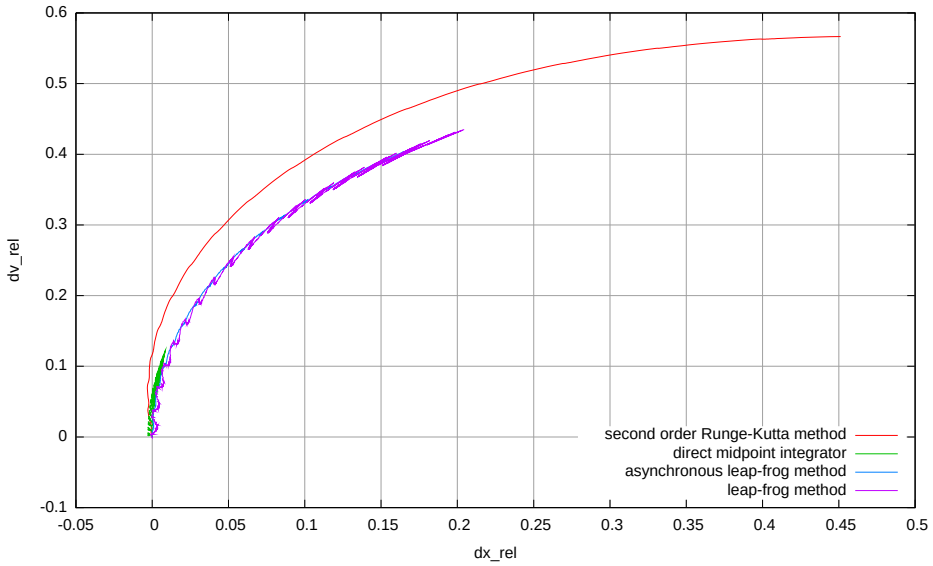


Figure 9: Integration error of four computational methods in the 'numerical interaction picture'.

ward error analysis (the converse is probably also true; it is the multitude of non-trivial observations 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. [16], 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 leapfrog integrators. We will see now that this suggestion is misleading. In the application considered in [22] it was found that the asynchronous leapfrog integrator showed similar step size requirements as the direct midpoint integrator when a actual leapfrog step was defined as consisting of two leapfrog 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* leapfrog integrators as

$$\begin{aligned} \tilde{\mathcal{L}} &:= \mathcal{L} \circ \mathcal{L} \\ \tilde{\mathcal{A}}_h &:= \mathcal{A}_{h/2} \circ \mathcal{A}_{h/2} \end{aligned} \tag{46}$$

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

finds them wiggling around a constant value for DALF-trajectories, and slowly linearly decreasing for ADALF-trajectories.

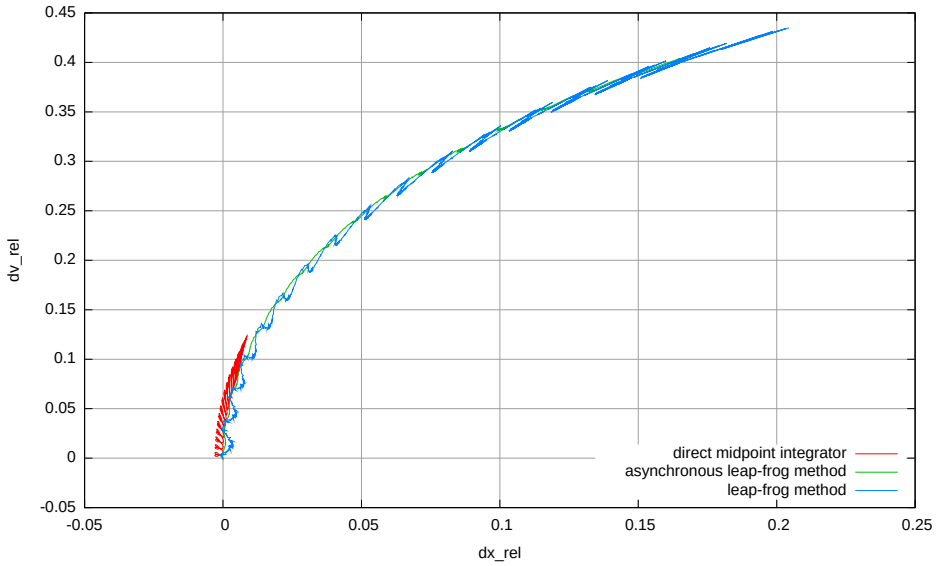


Figure 10: Integration error of the better methods in the 'numerical interaction picture'.

much better than for Runge-Kutta. It is plausible that only this densified version of the leapfrog 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 the leapfrog 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 leapfrog method has shorter and more regular error orbits than the standard leapfrog method.

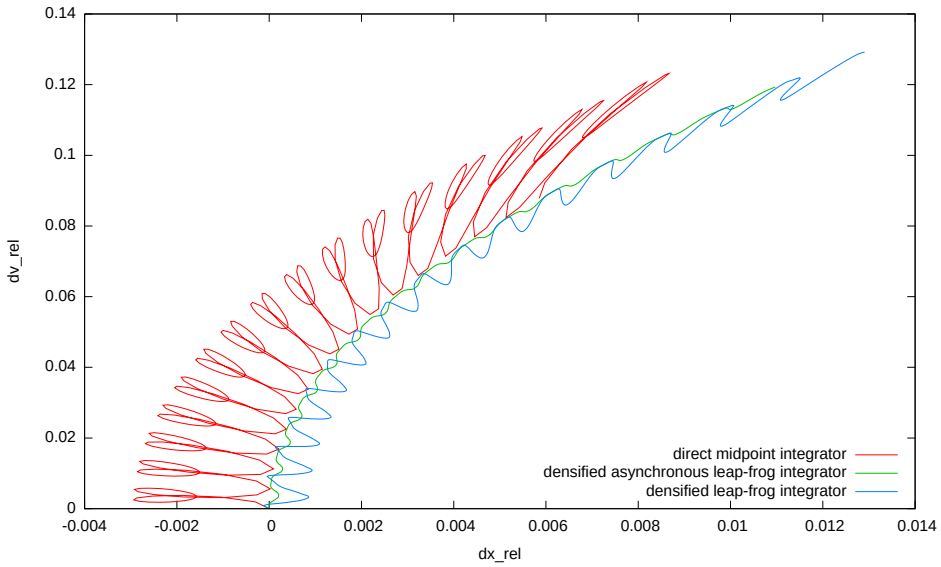


Figure 11: Integration error of the direct midpoint method together with the densified leapfrog methods in the 'numerical interaction picture' for $\epsilon = 0.15$. Sixteen periods at 32 steps per period.

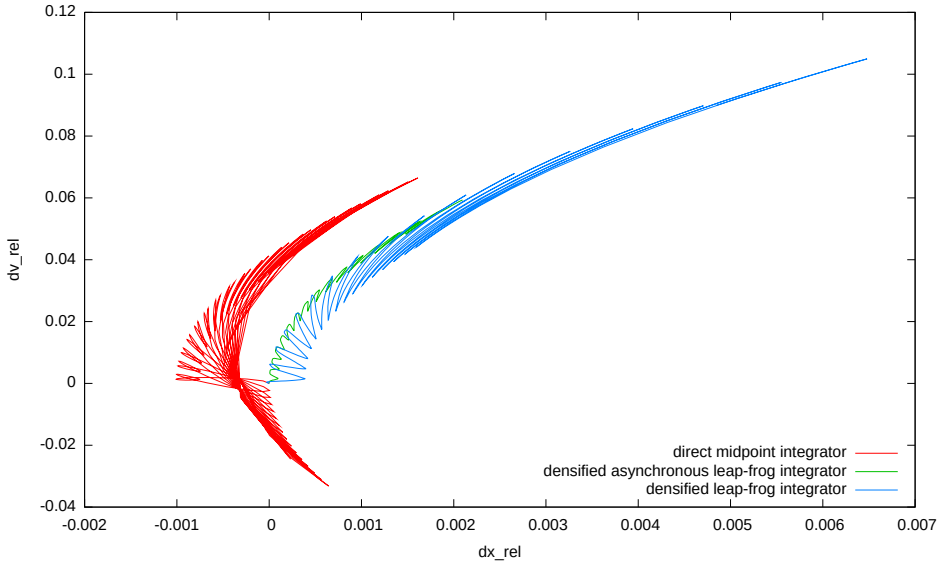


Figure 12: Integration error of the direct midpoint method together with the densified leapfrog methods in the 'numerical interaction picture' for $\epsilon = 0.30$. Sixteen periods at 64 steps per period.

6 The averaged densified asynchonous leapfrog integrator

Let us introduce short names for the integrators under consideration:

1. *ALF* asynchronous leapfrog, (28),(32)
2. *DALF* densified asynchronous leapfrog, (46)
3. *ADALF* averaged densified asynchronous leapfrog, to be defined in (48)

The problem to be faced is evident from Figure 4. Here, consecutive ALF-steps end in directions that deviate considerably from the correct direction. The deviation occurs in an alternating fashion. This suggests to define the final direction of a double step (thus of a single DALF-step) as the mean value out of the two ALF steps. Let us first write the DALF-step in the most economic form, i.e. in the style of (30) with $\lambda = 1$:

$$\begin{aligned} t \mathrel{+}= & \frac{\tau}{2} \,, \\ \psi \mathrel{+}= & \frac{\tau}{2}\,\phi \,, \\ \phi \mathrel{+}= & \, 2\,(F(t,\,\psi) - \phi) \,, \\ \psi \mathrel{+}= & \, \tau\,\phi \,, \\ t \mathrel{+}= & \, \tau \,, \\ \phi \mathrel{+}= & \, 2\,(F(t,\,\psi) - \phi) \,, \\ \psi \mathrel{+}= & \frac{\tau}{2}\,\phi \,, \\ t \mathrel{+}= & \frac{\tau}{2} \,. \end{aligned} \tag{47}$$

Here the time step of the whole DALF-step is 2τ. When considering an ALF-step as consisting of three sub steps, we have combined the last third of the first ALF-step with the first third of the second ALF-step and have saved a bit of work compared to a simple succession of two ALF-steps. This gives us the right to add a bit of computational

complexity to obtain the definition of the ADALF-step:

$$\begin{aligned} t &+= \frac{\tau}{2}\,, \\ \psi &+= \frac{\tau}{2}\,\phi\,, \\ \phi &+= 2\,(F(t,\,\psi) - \phi)\,, \\ \phi_1 &= \phi\,, \\ \psi &+= \tau\,\phi\,, \\ t &+= \tau\,, \\ \phi &+= 2\,(F(t,\,\psi) - \phi)\,, \\ \psi &+= \frac{\tau}{2}\,\phi\,, \\ \phi &= \frac{1}{2}(\phi + \phi_1)\,, \\ t &+= \frac{\tau}{2}\,, \end{aligned} \tag{48}$$

as the most obvious realization of the averaging strategy indicated above. The general structure of the strategy is as follows: If an iteration is defined by application of a function $x_{i+1} := f(x_i)$, then a more robust and fast way to approach its iteration limit is by a modified iteration scheme $x' := f(x_i)$, $x'' := f(x')$, $x_{i+1} := (x' + x'')/2$ which combines the original iteration scheme with a simple low-pass filter. Of course, the algebraic nature of the quantities x has to support formation of a mean value. This method occurred already in the code following (42) for solving Kepler's equation, and it worked perfectly in countless iteration applications for me. As we will see, ADALF adds to the success story of this method. Very probably this method is known and honored in some scientific community and I would be very interested to get notice from anybody who knows it. Finally it may be convenient to write the ADALF algorithm down for a second order system

$$\dot{x} = v\,, \quad \dot{v} = F(t, x, v) \tag{49}$$

with initial values $x(t_0) = x_0$, $v(t_0) = v_0$. This form applies directly to mechanical problems with velocity-dependent forces. We write (x, v) for ψ and (w, a) for ϕ and

translate (48) to

$$
\begin{aligned}
t &+= \frac{\tau}{2}\,, \\
x &+= \frac{\tau}{2}\,w\,, \quad v += \frac{\tau}{2}\,a\,, \\
w &+= 2\,(v-w)\,, \quad a += 2\,(F(t,x,v)-a)\,, \\
w_1 &= w\,, \quad a_1 = a\,, \\
x &+= \tau\,w\,, \quad v += \tau\,a\,, \\
t &+= \tau\,, \\
w &+= 2\,(v-w)\,, \quad a += 2\,(F(t,x,v)-a)\,, \\
x &+= \frac{\tau}{2}\,w\,, \quad v += \frac{\tau}{2}\,a\,, \\
w &= \frac{1}{2}(w+w_1)\,, \quad a = \frac{1}{2}(a+a_1)\,, \\
t &+= \frac{\tau}{2}\,.
\end{aligned}
\tag{50}
$$

The initialization of $(x,v),(w,a)$ is of course given by $x(t_0)=x_0$, $v(t_0)=v_0$, $w(t_0)=v_0$, and $a(t_0)=F(t_0,x_0,v_0)$. The versions without an averaging step (i.e. DALF) and without a separation in two half-steps (i.e. ALF) are obvious from this pattern.

6.1 Stability

As is well known the behavior of integrator-created trajectories of the simple linear test equation

$$\dot{\psi}(t) = \omega\,\psi(t)\,, \quad \omega\in\mathbb{C}\,, \quad |\omega|=1 \tag{51}$$

for a complex-valued function ψ helps to analyze the complex phenomenon of *stability* of integrators [10], Def 3.6.1, [21]. Due to the simple nature of (51) the integrators of present interest can be represented by the following complex propagation matrices:

$$\begin{pmatrix}\psi_{n+1}\\ \phi_{n+1}\end{pmatrix} = \begin{pmatrix}\alpha(h,\omega) & \beta(h,\omega)\\ \gamma(h,\omega) & \delta(h,\omega)\end{pmatrix}\begin{pmatrix}\psi_n\\ \phi_n\end{pmatrix}\,, \tag{52}$$

where h is the time step. The matrix elements $\alpha,\beta,\gamma,\delta$ are given for ALF:

$$\alpha(h,\omega) = 1+h\omega \qquad \beta(h,\omega) = \frac{h^2\omega}{2} \tag{53}$$

$$\gamma(h,\omega) = 2\omega \qquad \delta(h,\omega) = -1+h\omega\,, \tag{54}$$

and for DALF:

$$\alpha(h,\omega) = 1+h\omega+\frac{h^2\omega^2}{2} \qquad \beta(h,\omega) = \frac{h^3\omega^2}{8} \tag{55}$$

$$\gamma(h,\omega) = 2h\omega^2 \qquad \delta(h,\omega) = 1-h\omega+\frac{h^2\omega^2}{2}\,, \tag{56}$$

and for ADALF:

$$\alpha(h,\omega) = 1 + h\omega + \frac{h^2\omega^2}{2} \qquad \beta(h,\omega) = \frac{h^2\omega(1+h\omega)}{16} \tag{57}$$

$$\gamma(h,\omega) = 2h\omega^2 \qquad \delta(h,\omega) = \frac{h\omega(-1+h\omega)}{4} \ . \tag{58}$$

For reasons of comparison we present the propagation matrix of the general second order explicit Runge-Kutta method. It depends on a parameter a_1. Giving this parameter the value $0, 1/3, 1/2$ defines the midpoint method, Ralfton's method, and Heun's method respectively. The only Runge-Kutta method considered in Section 5 is the midpoint method, $a_1 = 0$. In order to allow Runge-Kutta methods to be used interchangeably with leapfrog methods, their state space also gets augmented by a velocity. This velocity is initialized at the beginning of a trajectory by the same formula as in the leapfrog case, and the evolution step is reformulated in a way that it takes the velocity as given at the beginning (saving one evaluation of the f) and has to be updated at the end of the step (consuming one evaluation of the f). It is clear that this does not influence the states along a trajectory in any way, with the exception of the last state: In a trajectory of the modified style it determines not only a position but also a velocity. The computational work that generated this additional information was done with the initialization step. As we will see, the circumstance that we have a value for the velocity at the beginning and at the end of an evolution step allows us to implement a cheap and effective automatic step control. With the modification described here, this auto-step method works also with Runge-Kutta and an instructive direct comparison becomes possible.

The propagation matrix for second order Runge-Kutta methods is

$$\alpha(h,\omega,a_1) = 1 + h\omega(1-a_1) \qquad \beta(h,\omega,a_1) = h(a_1 + \frac{h\omega}{2}) \tag{59}$$

$$\gamma(h,\omega,a_1) = \omega(1 + h\omega(1-a_1)) \qquad \delta(h,\omega,a_1) = h\omega(a_1 + \frac{h\omega}{2}) \ . \tag{60}$$

Now we consider the two complex eigenvalues of the propagation matrices of the various methods. We have for ALF:

$$\lambda_{1,2}(h\omega) = h\omega \pm \sqrt{1 + h^2\omega^2} \ , \tag{61}$$

and for DALF:

$$\lambda_{1,2}(h\omega) = \frac{1}{2}\left(2 + h^2\omega^2 \pm h\omega\sqrt{4 + h^2\omega^2}\right) \ , \tag{62}$$

and for ADALF:

$$\lambda_{1,2}(h\omega) = \frac{1}{8}\left(4 + 3h\omega + 3h^2\omega^2 \pm \sqrt{16h\omega + (4 + 3h\omega + 3h^2\omega^2)^2}\right) \tag{63}$$

and for second order Runge-Kutta:

$$\lambda_1(h\omega) = 0 \ , \quad \lambda_2(h\omega) = 1 + h\omega + \frac{h^2\omega^2}{2} \ . \tag{64}$$

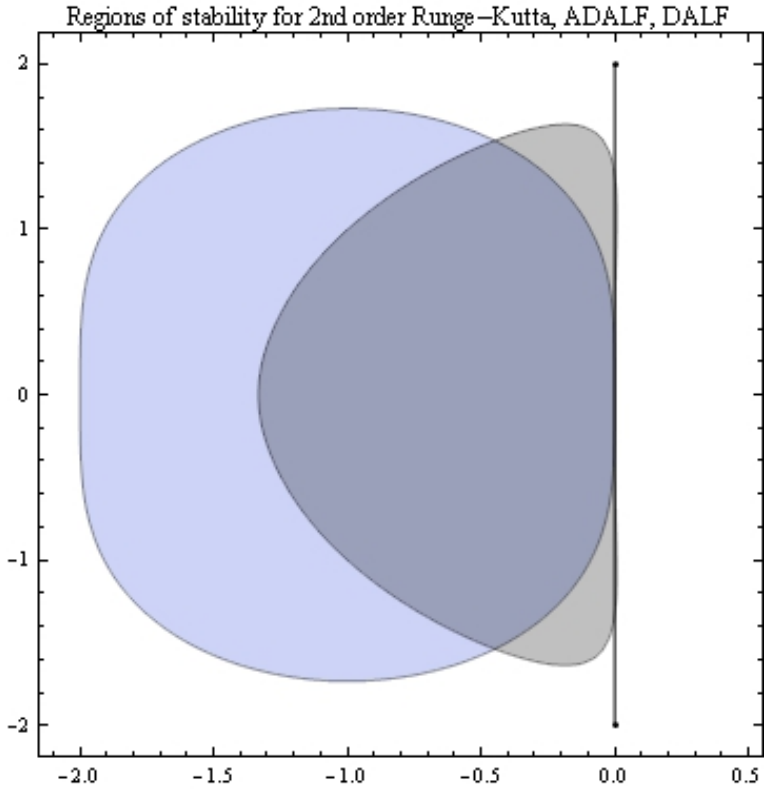


Figure 13: Regions of stability for 2nd order Runge-Kutta, ADALF, and DALF

Notice that the eigenvalues are independent of the parameter a_1. The number $h\omega \in \mathbb{C}$ belongs to the *set of absolute stability* of an integrator of the kind considered here, iff $|\lambda_1(h\omega)| \leq 1$ and $|\lambda_2(h\omega)| \leq 1$. Given the explicit formulas for λ_1, λ_2 it is straightforward to work out the regions of stability as done here by a commercial graphics function [3] that created Figure 13. The two points where the ADALF stability region disengages from the imaginary axis have coordinates $(0, \pm 4/3)$.

The striking difference between the pure leapfrog methods, which give only an interval on the imaginary axis [4] , and the averaged version, which gives a region that extends into the left half-plane (as is also the case for the Runge-Kutta methods), asks for an explanation. Thereto we consider the solutions of (51) with initial condition $\psi(0) = 1$ for various values of ω. For $\omega = \mathrm{i}$ we have the purely oscillatory solution $\psi(t) = \mathrm{e}^{\mathrm{i}t}$ for which leapfrog works well if the time step is not too large. If, however, ω gets a non-vanishing negative real part, the solution is a damped oscillation that finally dies out and becomes horizontal and nearly straight.

This, together with the reversibility of the leapfrog method causes problems that motivated the relaxation parameter in (30) and were discussed together with the relaxation

[3] RegionPlot of Mathematica

[4] The ALF method, of course has stability set $[-\mathrm{i}, +\mathrm{i}]$

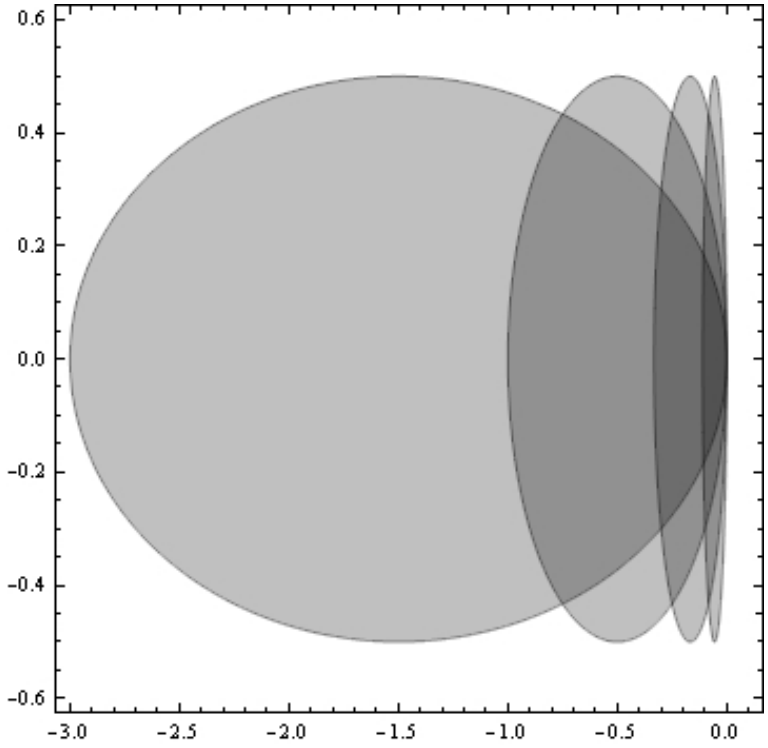


Figure 14: Regions of stability for ALF with relaxation. The values of the relaxation parameter λ from left to right are 0.25, 0.5, 0.75, 0.9.

approach. This approach is now obsolete and the ADALF method is the proper replacement. The most obvious disadvantage of the relaxation method is that there is no single preferable value for λ. This can be understood on the basis of the stability regions, which for the integrator (30) are shown in Figure 14. Obviously the origin is the only stable point on the imaginary axis and so for $\lambda < 1$ no harmonic oscillation is stable. The common feature of the relaxation method and the ADALF method is that they break reversibility. Let us now leave the relaxation method once and for all.

For the purely oscillatory case, i. e. ω on the imaginary axis, ADALF needs a by a factor 1.5 ($= 2 : 4/3$) shorter integration steps than DALF in order to avoid exponential growth ('1.5 π steps per period' for ADALF instead of 'π steps per period' for DALF). Since each step does 2 function evaluations, we need 6.3 or 9.4 function evaluations per period. This roughly places the evaluations at the corners of a regular hexagon or a regular nonagon. It is plausible that for less evaluations it is hard to follow a circle.

7 The Kepler oscillator as a test example, continued

Compared to the treatment in Section 5 we consider in some cases also trajectories with trajectories of numerical eccentricity up to 0.99. Figure 15 shows the phase space

portrait up to eccentricities $\epsilon = 0.9$. As is obvious from this portrait and even more so from considering the corresponding Kepler orbit, for strongly eccentric orbits the velocity varies over a wide range so that we then have an example of a *stiff system*. When we give for two time-stepped trajectories the same value for steps per revolution, although the eccentricities differ considerably, this is not a fair comparison. Since the velocities vary more strongly for the more eccentric trajectory, one should use more steps for the more eccentric trajectory, if generating a discrete approximation with fixed time step should be an equally demanding task for both trajectories. To find the proper enhancement factor it is helpful to consider the full Kepler trajectory from which one gets our oscillator trajectory as the projection onto the radial direction. Using the distance r from the central body we have for the angular velocity n (see (42)) the formula $L = r^2 n = 1$ (known as Kepler's second law, see (39) and following for L). From $r_{\min} = x_{\min} = 1/(1+\epsilon)$ we get $n_{\max} = r_{\min}^{-2} = (1+\epsilon)^2$ and

$$n_{\text{mean}} = a^{-3/2} = (1-\epsilon^2)^{3/2} \, .$$

Then, the appropriate number of steps per revolution depends on the eccentricity as follows:

$$N_{\text{perRev}}(\epsilon_1) : N_{\text{perRev}}(\epsilon_2) = \frac{n_{\max}(\epsilon_1)}{n_{\text{mean}}(\epsilon_1)} : \frac{n_{\max}(\epsilon_2)}{n_{\text{mean}}(\epsilon_2)} \, .$$

For $\epsilon_1 := 0$ and $\epsilon_2 := \epsilon$ we get

$$N_{\text{perRev}}(\epsilon) = N_{\text{perRev}}(0) \, \frac{n_{\max}(\epsilon)}{n_{\text{mean}}(\epsilon)} = N_{\text{perRev}}(0) \, \sqrt{\frac{1+\epsilon}{1-\epsilon}} \, \frac{1}{1-\epsilon} \, . \tag{65}$$

In legends of graphics the quantity nPerRev always means $N_{\text{perRev}}(0)$ and the number of integration steps per revolution that actually is being used depends on ϵ and is given as $N_{\text{perRev}}(\epsilon)$ from (65). For an estimate of the stability limit it is plausible to take nPerRev as the 'steps per period' of the surrogate equation from Subsection 6.1 and to infer from the analysis there the limits π for DALF and $\frac{3}{2}\pi$ for ADALF.

7.1 The main difference between DALF and ADALF

We want to see how DALF's stability deficit manifests itself in a typical situation and how ADALF remedies the deficit. The figures 16, 17, and 18 show such a situation. All these figures refer to a trajectory with the extreme eccentricity $\epsilon = 0.99$. For the first of these figures the time step is by a factor $10 : \pi = 3.2$ below the stability limit of DALF, so that we are sure that the solution will never grow exponentially. A certain amount detail rendition should not be expected unless the time step is by a factor of about 10 below the stability limit. In Figure 16 we see a dramatic deviation of the DALF trajectory from expected behavior. Long before having reached its correct turning point $x \approx 0.5$ the particle stops, makes a feeble leap and then a strong bolt which releases all the energy that the particle had prior to the first stop. How can the particle be forced to bounce back against the direction of the force field? This is caused by a phenomenon in the working of the DALF integrator that one may describe as an excitation of an internal

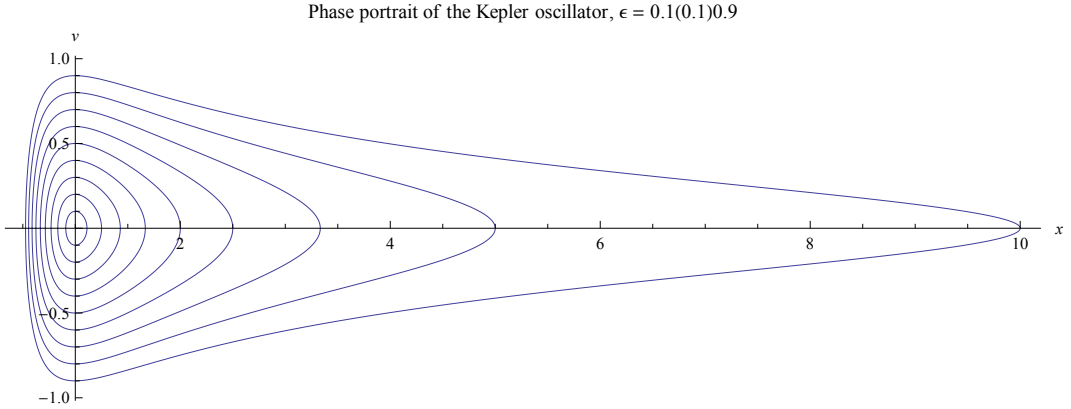


Figure 15: Phase space portrait of the Kepler oscillator

degree of freedom. If this excitation is very large it lets the trajectory attain a zigzag character. But long before a zigzag becomes visible in a typical graphical representation of a trajectory unexpected things like our bouncing back phenomenon may happen. It is useful to have a quantity which indicates the 'degree of internal excitation'. Such a quantity can be defined as follows: In the algorithm (47) the terms $a := F(t, \psi)$ and $b := \phi$ are intended to be approximately equal so that their small difference, in the third and the sixth step of the algorithm causes a fine tuning of the direction of motion in accordance with the vector field F. To detect deviations from this normal behavior we compute the quantity

$$\kappa(a, b) := \frac{\| a - b \|}{\| a \| + \| b \| + \text{tiny}} \tag{66}$$

which takes values in $[0, 1]$, where values larger than, say, 0.1 indicate significant deviation from $a \approx b$. In one DALF step we compute two κ-values, and their mean value gives the quantity we wish to define. I refer to this particular quantity as *jerk*. For ADALF the same definition makes sense and will be used for comparison. It is clear that the averaging step in ADALF smooths away any zigzag and keeps jerk down. The jerk values for DALF- and for ADALF-trajectories are shown in the lower part of the Figures under consideration. The first of these figures clearly shows that for the DALF trajectory the integrator works fully in the 'jerky mode' and thus is no longer under the only control of the field F but also reacts to the accumulated value of the now virtually autonomous quantity ϕ. The ADALF-trajectory shows considerable jerk only near the return points. It absorbs kinetic energy to the effect that the trajectory deviates strongly from the exact one, in a predictable manner, though. The next figures reduce the time step by a factor 2 each and show no longer the unphysical bouncing-back and the trajectories and oscillation frequency and amplitude come closer to those of the exact solution. The jerk graphs demonstrate that the jerk level goes down systematically. In the jerk graph of DALF of Figure 17 there looms a plateau which is fully developed in Figure 18. Notice that the automatically drawn t-axis follows just this plateau and reduces its visibility.

One should recall that DALF is reversible so that the trajectory, even if it has dissolved

into a felt of interwoven zigzag lines, can be followed back to the stage where it was an innocent smooth curve. This suggests to consider the 'excited state of a leapfrog trajectory' an interesting object to study, possibly as a model for a quantum-mechanical wave function. See also Figure 6, case $\lambda = 1$, where the excited trajectory mutates back to a smooth one in a periodic manner.

7.2 Order

The order of an integrator is best demonstrated by analyzing how the mean error along a trajectory over a given time span depends on the number of integration steps. Here, we vary the integration time step over two octaves in 4 stages. For Figure 19 we consider a situation that is computationally not demanding: the eccentricity is small ($\epsilon = 0.01$) and and the time step is many times below the stability limit for all integrators under consideration. These methods are: DALF, ADALF, the three Runge-Kutta methods defined in Subsection 6.1, and the Størmer-Verlet integrator which in Section 5 was referred to as direct midpoint integrator. The result from this figure is that all these integrators are of order 2 and that the leapfrog integrators (to which also Størmer-Verlet belongs) are four times more accurate as the Runge-Kutta integrators. For Fig 20 the eccentricity is increased by a factor of 20 ($\epsilon = 0.2$) and we see some differentiation in the data of the methods. Fig 21 shows the mean error and the order for the range $\epsilon = 0.5(0.5)0.95$. Here we find variations of the curves from integrator to integrator, especially among the Runge-Kutta integrators, which may come as a surprise if one recalls that the stability region is the same for the Runge-Kutta methods under consideration. One more observation: The mean error grows with ϵ although with (65) measures were taken to equalize the accuracy of the result with respect to ϵ.

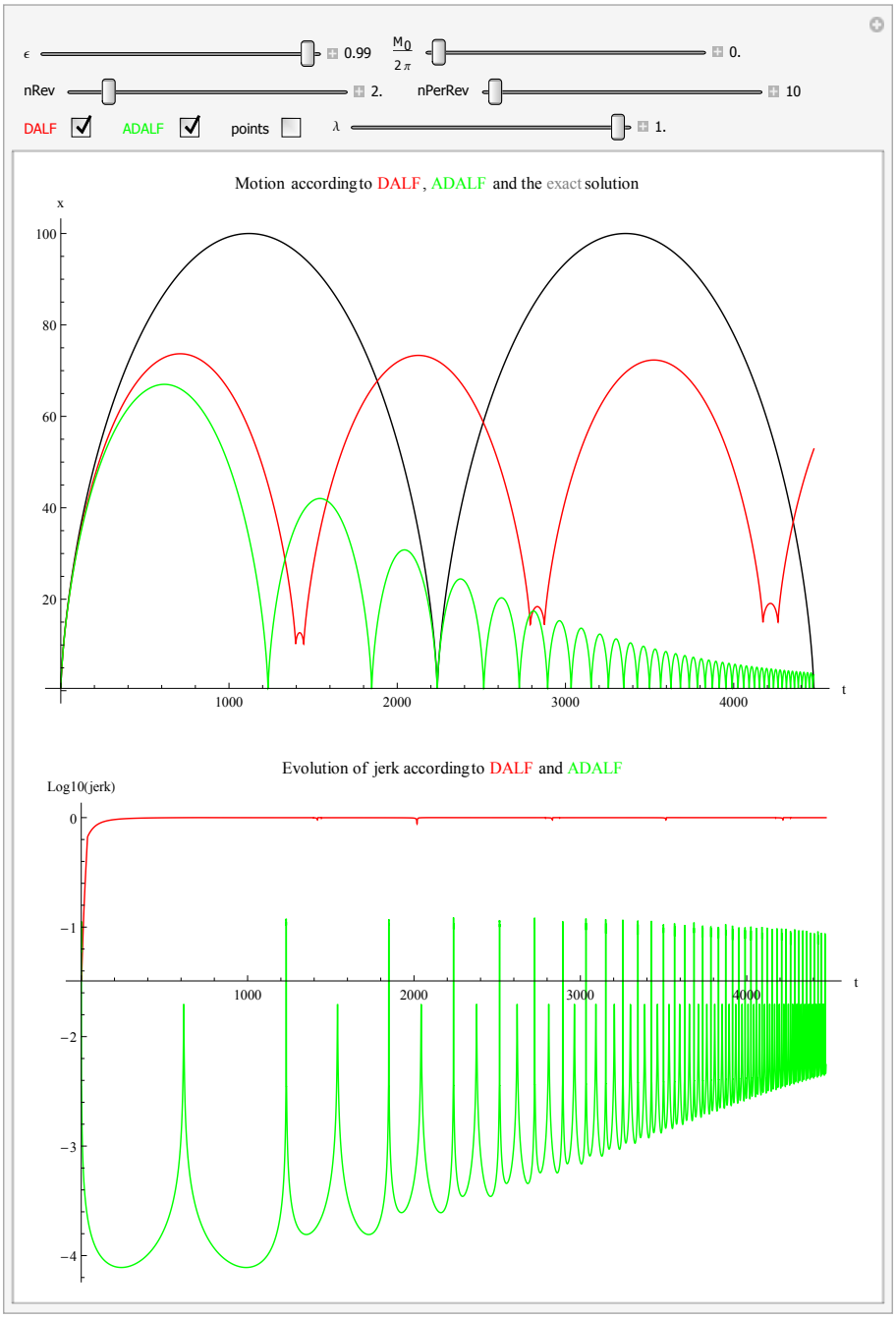


Figure 16: Position and jerk as functions of time

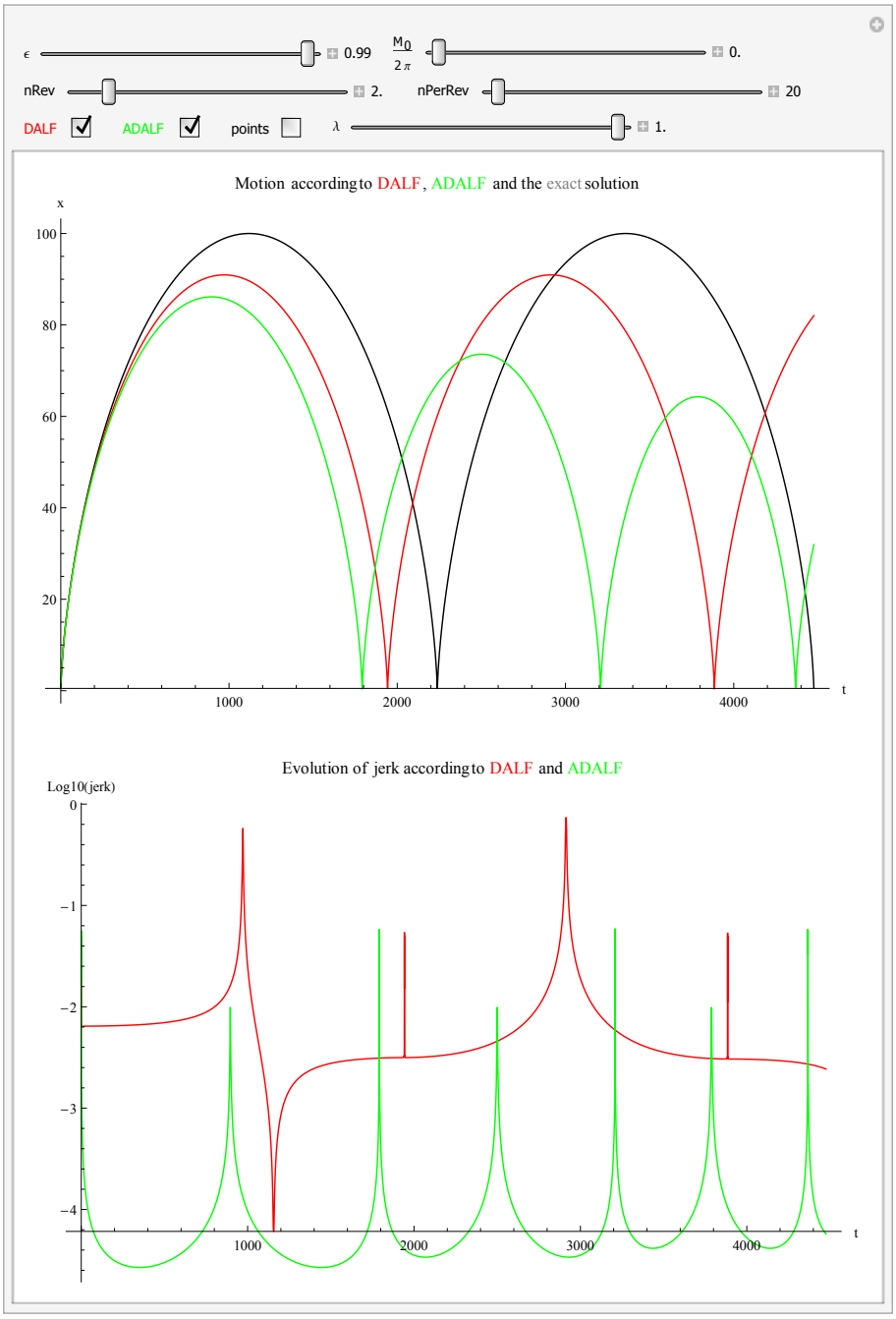


Figure 17: Position and jerk as functions of time for half of the original time step

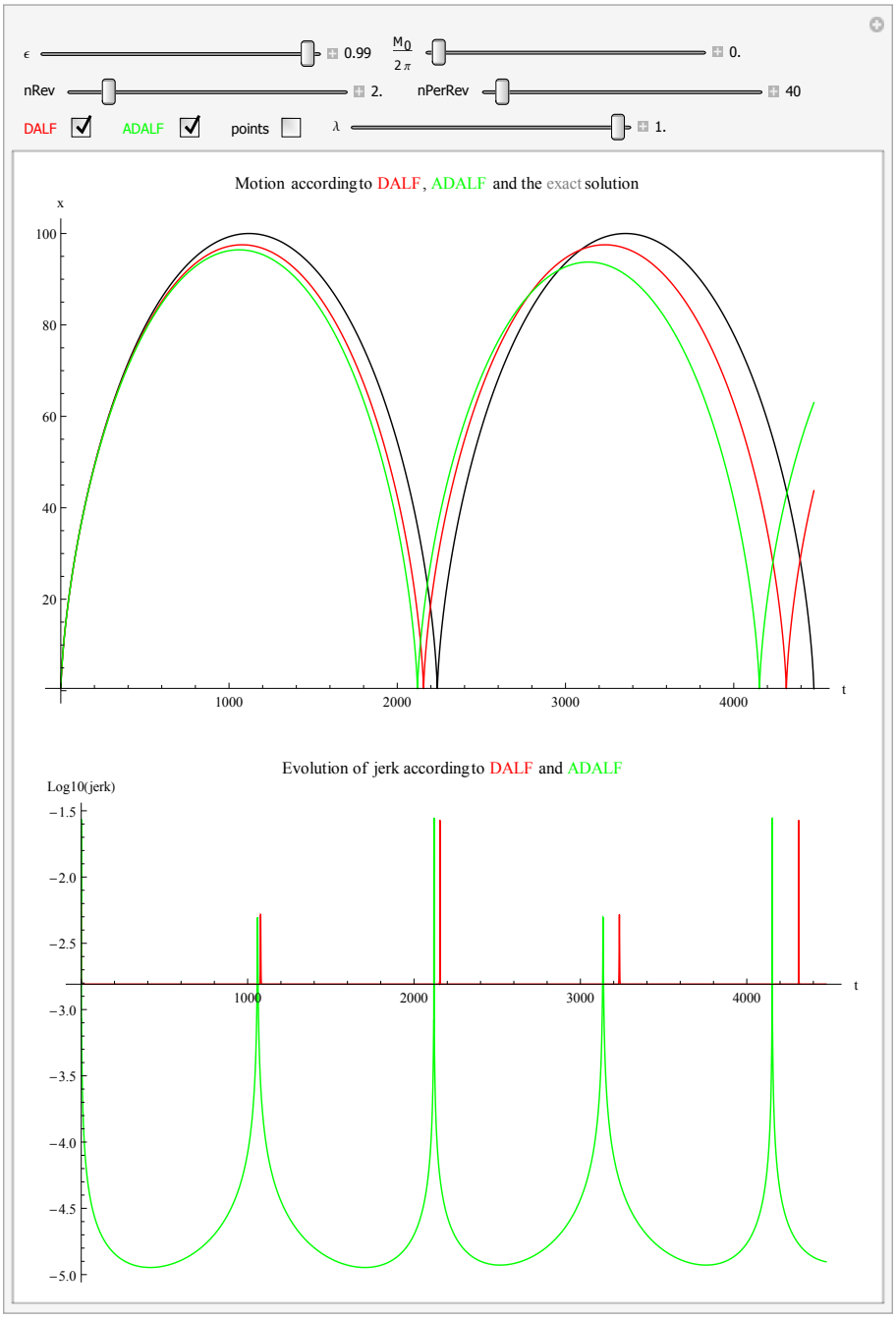


Figure 18: Position and jerk as functions of time for a quarter of the original time step

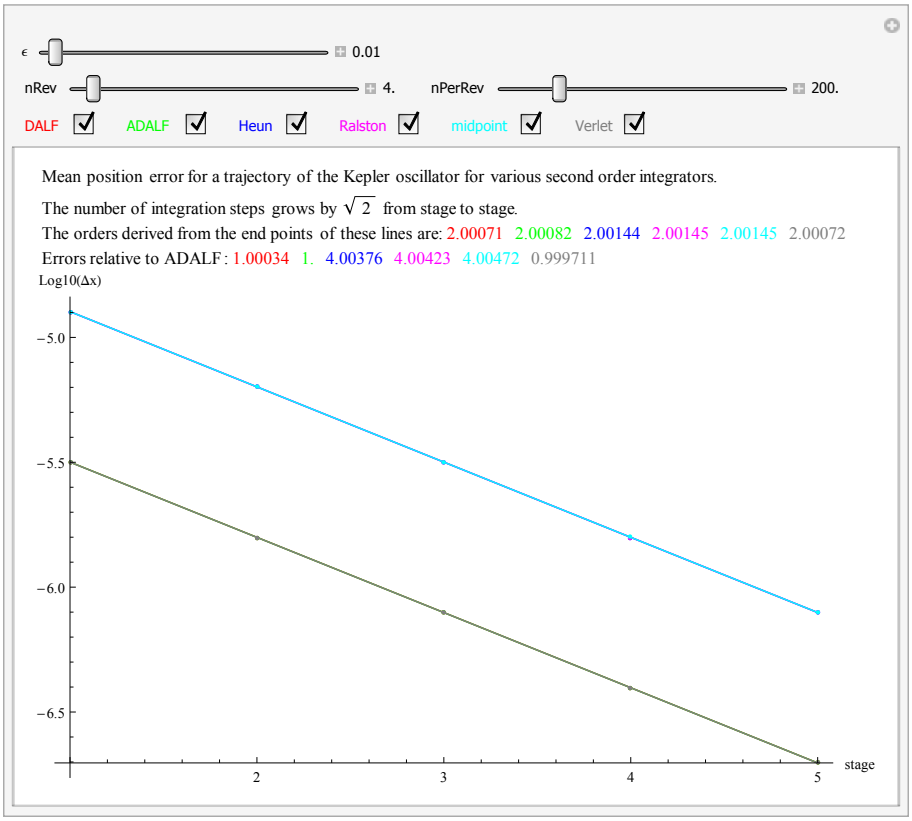


Figure 19: Order and accuracy of integrators for $\epsilon = 0.01$

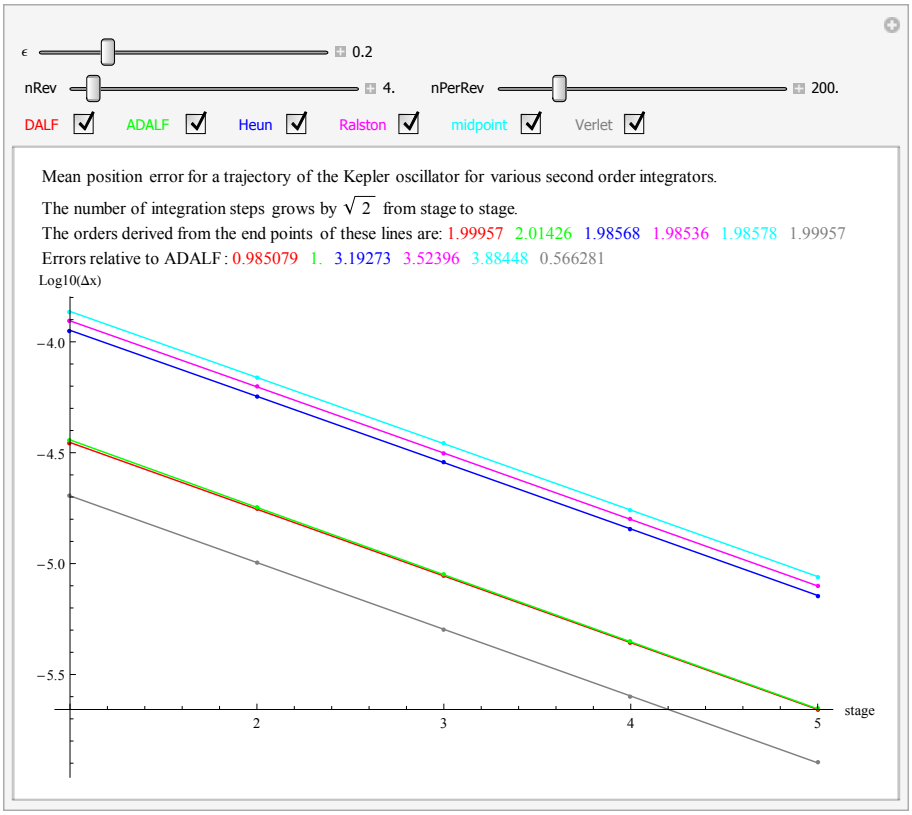


Figure 20: Order and accuracy of integrators for $\epsilon = 0.2$

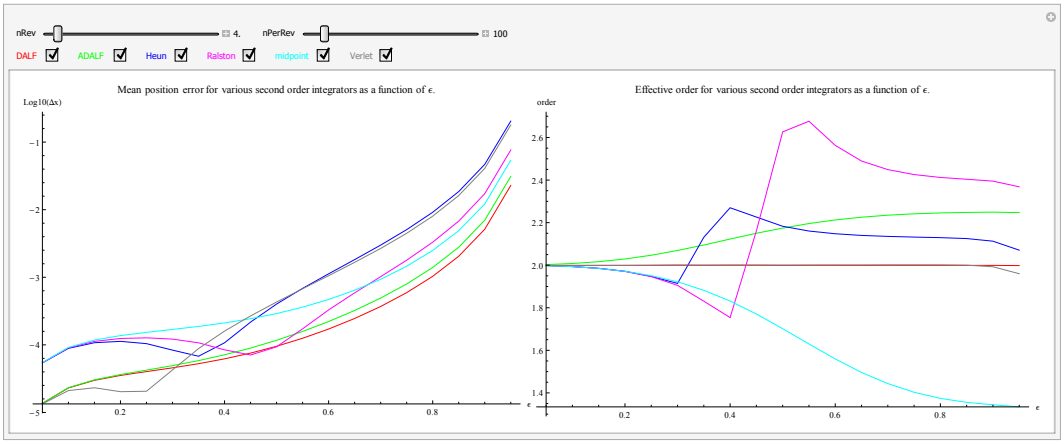


Figure 21: Accuracy and order of integrators over an ϵ-range from 0.05 to 0.95

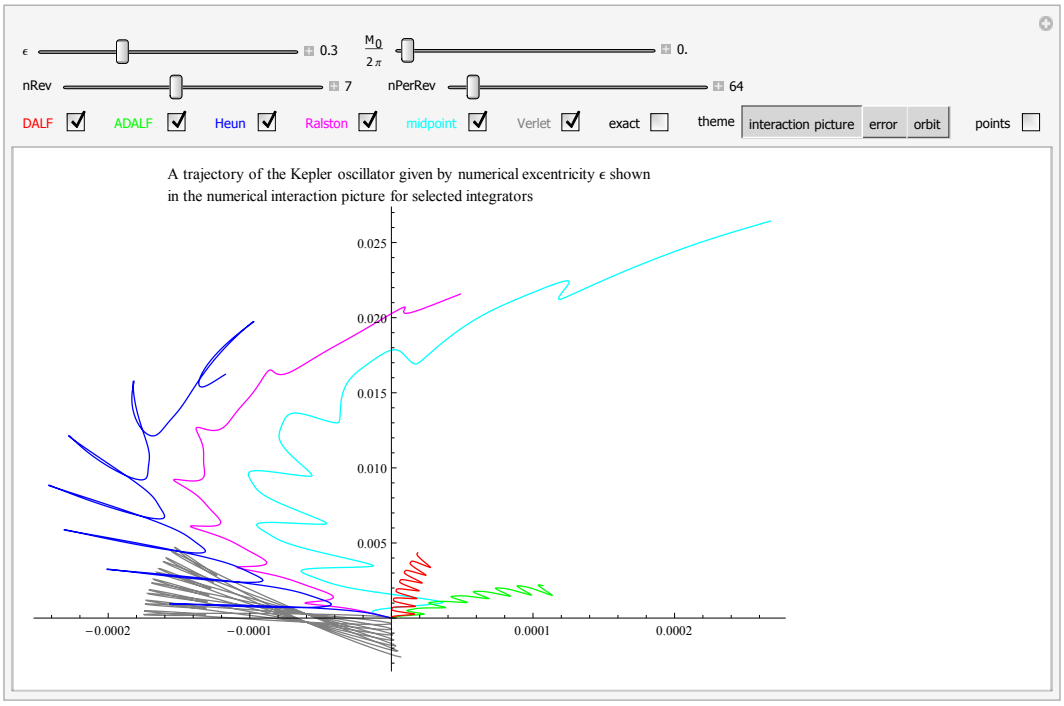


Figure 22: Interaction picture of Runge-Kutta and leapfrog trajectories

7.3 Numerical interaction picture

The trajectories from the numerical interaction picture as introduced in Section 5 show in Figure 22, in agreement with the results of the previous subsection, that the Runge-Kutta methods are considerably less accurate than the leapfrog methods. Recall that short trajectories indicate high accuracy of the integrator.

Figure 23 shows only the leapfrog methods and thus allows a more detailed rendition of their trajectories. Here parameters are selected such that in the gray Størmer-Verlet trajectory a strange 'two focal points' feature appears, the nature of which is not understood so far. These focal points are shown again with slightly different parameters in Figure 24 in a format that better allows to appreciate the interesting geometry and the aesthetic qualities of the trajectory.

7.4 Automatic step control

One of the motivations for inventing the asynchronous version of the leapfrog integrator was to facilitate automatic time step control. This subsection will show that this works in a simple and quite effective manner. This method seems to have been first demonstrated in [23]. It assumes that the state to be propagated is of the form (ψ, ϕ), where ϕ was initialized by (24). All integrators which as the first operation in an evolution step do such an evaluation of F for values t and ϕ valid for the beginning of the step can be

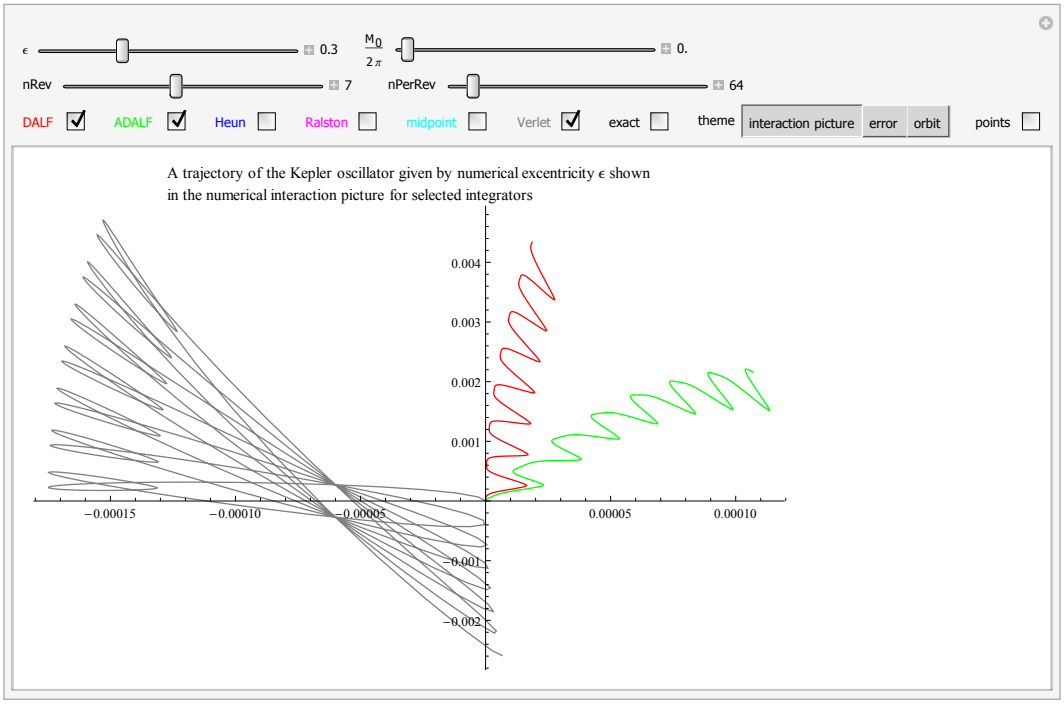


Figure 23: Interaction picture of leapfrog trajectories

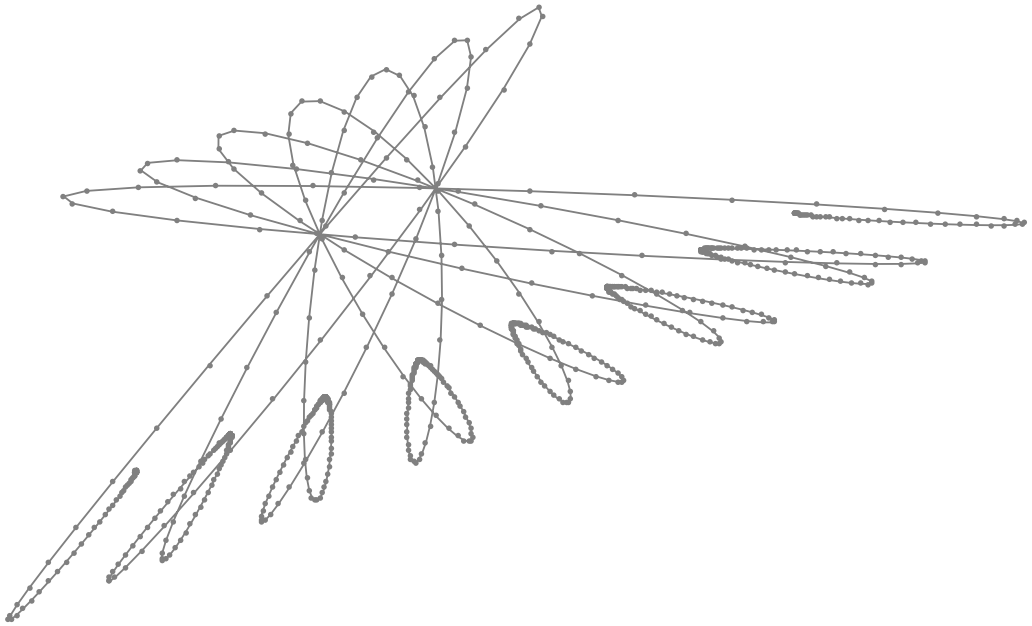


Figure 24: Interaction picture of a Størmer-Verlet trajectory

transformed to this assumed form: We leave out this first operation, replace their then no longer available result by the available quantity ϕ and add at the end of the step an operation $\phi = F(t, \psi(t))$ to have ϕ available for the step to follow. As already indicated in Subsection 6.1 this can be done for all second order explicit Runge-Kutta methods so that we can use those for comparison without any change in the driver logic of the program that created Figure 25. For the Størmer-Verlet method, ϕ is not an auxiliary quantity that can be given a value by evaluation of F. So the method does not apply verbally. So it is left out as a comparison method here.

So, how time step control works? After having done an evolution step we know an initial value ϕ_i and a final value ϕ_f of ϕ. Using (66) we compute $\kappa(\phi_\mathrm{i}, \phi_\mathrm{f})$ (this is a simple computational step, which does not require an evaluation of F). If this dimensionless quantity exceeds some agreed critical level a_1 we conclude that this step was not acceptable. As a consequence we re-initialize ϕ and re-do the step with a shorter time step. Let us call f_1 the factor with which we multiply the old time step. If κ is smaller than some agreed comfortable level a_2, we increase the time step to be used for the next evolution step by multiplication with some some agreed factor f_2. In the program of Figure 25 there are input quantities kinkCrit = 0.001 and frac = 0.2. These determine the quantities just mentioned as follows:

$$a_1 = \mathrm{kinkCrit}\ , \quad a_2 = 0.5a_1\ , \quad f_1 = 1 - \mathrm{frac}\ , \quad f_2 = 1 + \mathrm{frac}\ .$$

We see that the results are by a factor of about 200 more accurate than in the fixed step integration Figure 21. However we need 25 times more function evaluations. For a second order integrator this translates into a factor 625 for accuracy which has to be compared to the factor 200 that we actually got. So the method is by a factor 3 less efficient than one would expect for an optimally selected constant time step. Of course, all integrators need nearly the same number of function evaluations since curvature of the exact trajectory determines, where a re-initialization is needed.

Over the ϵ-range in Figure 21 the time step varies by a factor of about 125 and only of about 3 in Figure 25.

The error curves in Figure 21 and Figure 25 show surprising similarity. Especially apparent are the minima of the curves of Ralfton's and Heun's methods for $\epsilon \approx 0.4$.

For serious applications in which both accuracy and computation time matter, modified values of a_1, a_2, f_1, f_2 or even variations of the algorithm may be useful. My experience with large applications is restricted to constant time step.

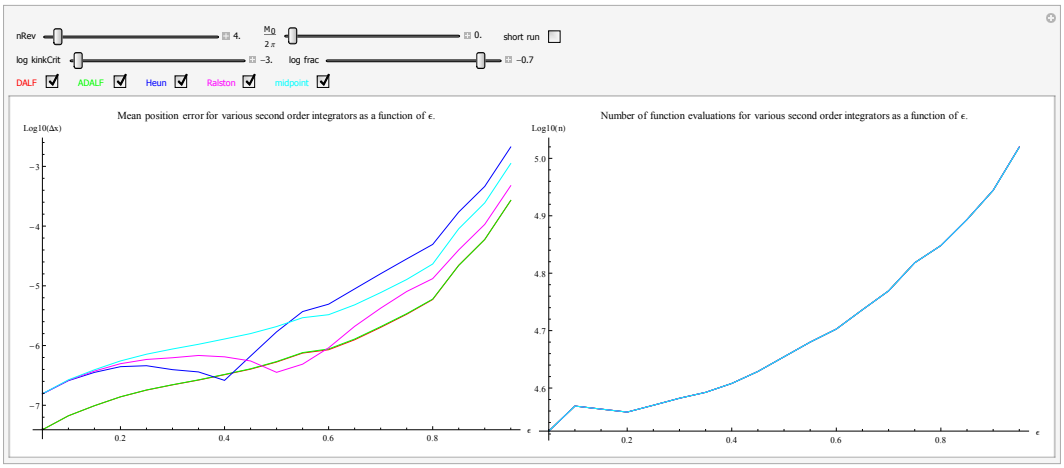


Figure 25: Run of all integrators, except of Størmer-Verlet, under automated step control for $\epsilon = 0.05(0.05)0.95$

Acknowledgment

I am grateful to Domenico Castrigiano for many discussions on the relation of discrete mathematics to classical analysis and to Ernst Hairer , Blair Perot, and David Seal for valuable comments. H.E. Lehtihet has provided instructive test examples and provided ideas for interpreting the obtained results. Sajad Jafari shared his very interesting and computationally demanding equations dealing with 'romantic relationships' prior to publication.

References

[1] Ulrich Mutze: An asynchronous leapfrog method (2008)
Mathematical Physics Preprint Archive 2008–197
http://www.ma.utexas.edu/mp_arc/c/08/08-197.pdf

[2] Ulrich Mutze: homepage
http://www.ulrichmutze.de/articles/leapfrog4.pdf

[3] Ulrich Mutze: Post at researchgate.net
https://www.researchgate.net/post/Who_knows_of_an_integrator_for_the_general_first_order_ODE_which_is_second_order_and_needs_only_one_evaluation_of_the_equations_rhs_per_time_step

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

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

[6] 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)

[7] 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)

[8] 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

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

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

[11] 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

[12] 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

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

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

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

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

[17] 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

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

[19] 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)

[20] 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

[21] L.F. Shampine: Stability of the Leapfrog/Midpoint Method
www.faculty.smu.edu/shampine/stablemidpoint.pdf

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

[23] The Asynchronous Leapfrog Method as a Stiff ODE Solver
from The Wolfram Demonstrations Project.

http://demonstrations.wolfram.com/TheAsynchronousLeapfrogMethodAsAStiffODESolver/

Contributed by: Ulrich Mutze (published 2013-05-22)

last modification 2013-10-29