PHYS 141/241

Lecture 04: Numerical Integration Methods (Continued)

Javier Duarte — April 9, 2023

2nd-order Runge-Kutta derivation

- Euler method uses the first two terms in Taylor series to approximate

$$S(t_{n+1}) = S(t_n + \Delta t) = S(t_n) + \dot{S}(t_n)\Delta t$$

- We can improve the accuracy if we keep more terms

$$S(t_{n+1}) = S(t_n + \Delta t) = S(t_n) + \dot{S}(t_n)\Delta t + \frac{1}{2!}\ddot{S}(t_n)\Delta t^2 + \cdots$$

2nd-order Runge-Kutta derivation

- Recall our ODE: $\dfrac{dS}{dt} = \dot{S}(t) = F(t, S(t))$

- This means

$$\ddot{S}(t) = \frac{d}{dt}\left[F(t, S(t))\right] = \frac{\partial F}{\partial t} + \frac{\partial F}{\partial S}\frac{dS}{dt} = \frac{\partial F}{\partial t} + \frac{\partial F}{\partial S}F(t, S(t))$$

- So we can write down two approximations to the derivative:

$$k_1 = F(t_n, S(t_n))$$
$$k_2 = F(t_n + \alpha\Delta t, S(t_n) + \beta k_1 \Delta t)$$

- And then take a weighted average: $S(t_n + \Delta t) = \dot{S}(t_n) + (ak_1 + bk_2)\Delta t$

2nd-order Runge-Kutta derivation

- What values of a, b, α, β minimize the error?
- Expanding our update equation

$$\begin{aligned} S(t_n + \Delta t) &= S(t_n) + (ak_1 + bk_2)\Delta t \\ &= S(t_n) + aF(t_n, S(t_n))\Delta t + b(F(t_n + \alpha \Delta t, S(t_n) + \beta k_1 \Delta t))\Delta t \end{aligned}$$

- Expanding the rightmost term:

$$F(t_n + \alpha\Delta t, S(t_n) + \beta k_1 \Delta t) = F(t_n, S(t_n)) + \frac{\partial F}{\partial t}\alpha\Delta t + \frac{\partial F}{\partial S}\beta F(t_n, S(t_n))\Delta t + \cdots$$

- Inserting back:

$$\begin{aligned} S(t_n + \Delta t) = S(t_n) &+ (a + b)F(t_n, S(t_n))\Delta t \\ &+ b\alpha\frac{\partial F}{\partial t}\Delta t^2 + b\beta\frac{\partial F}{\partial S}F(t_n, S(t_n))\Delta t^2 + \ldots \end{aligned}$$

2nd-order Runge-Kutta derivation

- Our proposed solution

$$S(t_n + \Delta t) = S(t_n) + (a+b)F(t_n, S(t_n))\Delta t$$
$$+ b\alpha \frac{\partial F}{\partial t}\Delta t^2 + b\beta \frac{\partial F}{\partial S} F(t_n, S(t_n))\Delta t^2 + \ldots$$

- Let's compare that to the exact solution up to $\mathcal{O}(\Delta t^3)$ from a Taylor series

$$S(t_n + \Delta t) = S(t_n) + F(t_n, S(t_n))\Delta t$$
$$+ \frac{1}{2}\frac{\partial F}{\partial t}\Delta t^2 + \frac{1}{2}\frac{\partial F}{\partial S} F(t_n, S(t_n))\Delta t^2 + \cdots$$

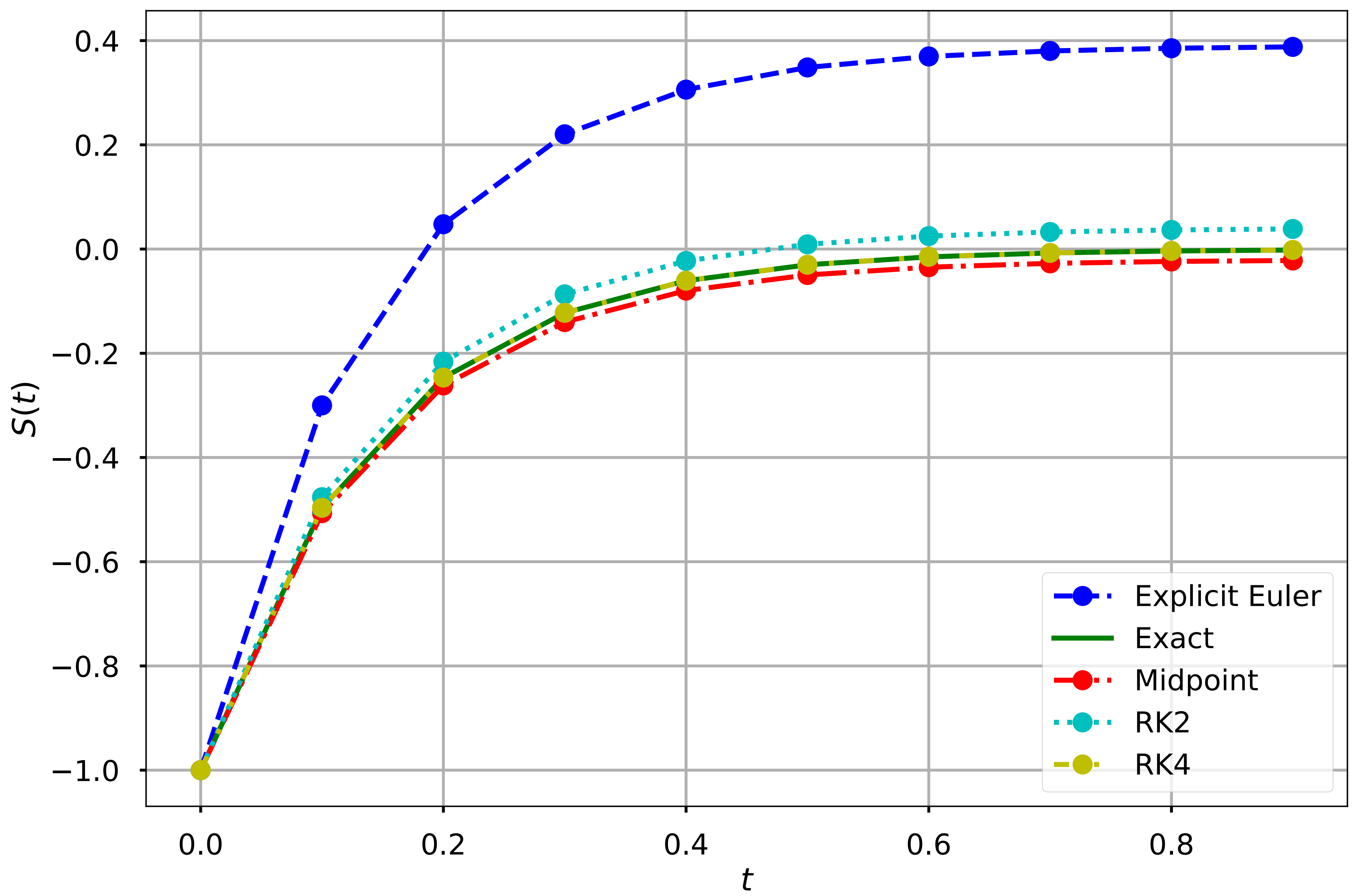
2nd-order accurate $\mathcal{O}(\Delta t^2)$

$$k_1 = F(t_n, S_n)$$
$$k_2 = F(t_n + \Delta t, S_n + k_1 \Delta t)$$
$$S_{n+1} = S_n + \left(\frac{k_1 + k_2}{2}\right)\Delta t$$

- So we find $a + b = 1$ and $b\alpha = b\beta = \frac{1}{2}$
- Infinitely many solutions! Common choice is $a = b = \frac{1}{2}$ and $\alpha = \beta = 1$

4th-order Runge-Kutta

- Can repeat same arguments to arrive at 4th-order method
- Basically “guess and check” with agreement using Taylor series



4th-order accurate $\mathcal{O}(\Delta t^4)$

$$k_1 = F(t_n, S_n)$$

$$k_2 = F(t_n + \Delta t/2, S_n + k_1 \Delta t/2)$$

$$k_3 = F(t_n + \Delta t/2, S_n + k_2 \Delta t/2)$$

$$k_4 = F(t_n + \Delta t, S_n + k_3 \Delta t)$$

$$S_{n+1} = S_n + \left(\frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6}\right)\Delta t$$

SciPy Solve IVP

- https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.solve_ivp.html
- RK4 (with some modifications

scipy.integrate.solve_ivp

`scipy.integrate.solve_ivp(fun, t_span, y0, method='RK45', t_eval=None, dense_output=False, events=None, vectorized=False, args=None, **options)` [source]

Solve an initial value problem for a system of ODEs.

This function numerically integrates a system of ordinary differential equations given an initial value:

```
dy / dt = f(t, y)
y(t0) = y0
```

Here t is a 1-D independent variable (time), y(t) is an N-D vector-valued function (state), and an N-D vector-valued function f(t, y) determines the differential equations. The goal is to find y(t) approximately satisfying the differential equations, given an initial value y(t0)=y0.

method : *string or `OdeSolver`, optional*

Integration method to use:

- 'RK45' (default): Explicit Runge-Kutta method of order 5(4) [1]. The error is controlled assuming accuracy of the fourth-order method, but steps are taken using the fifth-order accurate formula (local extrapolation is done). A quartic interpolation polynomial is used for the dense output [2]. Can be applied in the complex domain.
- 'RK23': Explicit Runge-Kutta method of order 3(2) [3]. The error is controlled assuming accuracy of the second-order method, but steps are taken using the third-order accurate formula (local extrapolation is done). A cubic Hermite polynomial is used for the dense output. Can be applied in the complex domain.

Verlet methods

- So far methods have been very generic
- For Newton-like equations $\ddot{\boldsymbol{r}}(t) = \frac{1}{m}\boldsymbol{F}(t)$, more specialized methods
- Verlet algorithm
  - Consider expansion of coordinate forward and backward in time:

$$\boldsymbol{r}(t+\Delta t) = \boldsymbol{r}(t) + \frac{1}{m}\boldsymbol{p}(t)\Delta t + \frac{1}{2m}\boldsymbol{F}(t)\Delta t^2 + \frac{1}{3!}\dddot{\boldsymbol{r}}(t)\Delta t^3 + O(\Delta t^4)$$

$$\boldsymbol{r}(t-\Delta t) = \boldsymbol{r}(t) - \frac{1}{m}\boldsymbol{p}(t)\Delta t + \frac{1}{2m}\boldsymbol{F}(t)\Delta t^2 - \frac{1}{3!}\dddot{\boldsymbol{r}}(t)\Delta t^3 + O(\Delta t^4)$$

  - Add these together and rearrange:

$$\boldsymbol{r}(t+\Delta t) = 2\boldsymbol{r}(t) + -\boldsymbol{r}(t-\Delta t) + \frac{1}{m}\boldsymbol{F}(t)\Delta t^2 + O(\Delta t^4)$$

    - Update without ever consulting velocities!

Verlet: Issues

- Initialization
  - How do we get the position at the previous time stem when starting out?
  - Simple approximation: $\boldsymbol{r}(t_0 - \Delta t) = \boldsymbol{r}(t_0) - \boldsymbol{v}(t_0)\Delta t$
- Obtaining the velocities
  - Not evaluated during the normal course of algorithm
  - But needed to compute some properties
  - Finite difference:

$$\boldsymbol{v}(t) = \frac{1}{2\Delta t}[\boldsymbol{r}(t + \Delta t) - \boldsymbol{r}(t - \Delta t)] + O(\Delta t^2)$$

Verlet: Performance issues

- Time reversible
  - Forward time step

    $$\boldsymbol{r}(t_0 + \Delta t) = 2\boldsymbol{r}(t_0) - \boldsymbol{r}(t - \Delta t) + \frac{1}{m}\boldsymbol{F}(t)\Delta t^2$$

  - Backward time step: replace $\Delta t \rightarrow (-\Delta t)$

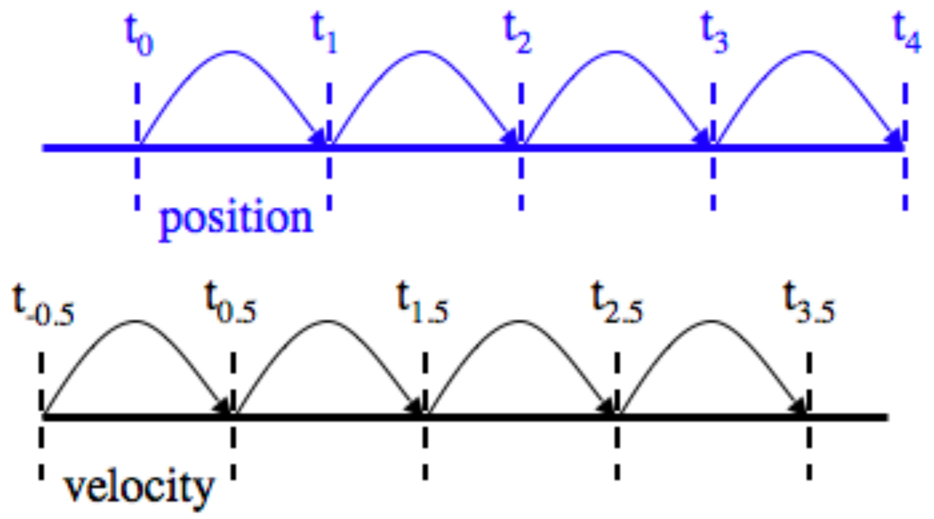
    $$\boldsymbol{r}(t_0 + (-\Delta t)) = 2\boldsymbol{r}(t_0) - \boldsymbol{r}(t - (-\Delta t))) + \frac{1}{m}\boldsymbol{F}(t)(-\Delta t)^2$$

  - Same algorithm, with same position and forces, moves system backward in time
  - If you step forward, and then backward, return to the same point!
- Numerical imprecision of adding large/small numbers

$$\underbrace{\boxed{\boldsymbol{r}(t + \Delta t) - \boldsymbol{r}(t)}}_{O(\Delta t^1)} = \underbrace{\boxed{\boldsymbol{r}(t)}}_{O(\Delta t^0)} + -\underbrace{\boxed{\boldsymbol{r}(t - \Delta t)}}_{O(\Delta t^0)} + \frac{1}{m}\boldsymbol{F}(t)\Delta t^2 .$$

Leapfrog

- Leapfrog is a variation on the so-called “velocity” Verlet
  - Eliminates addition of small numbers to differences in large ones



$$\boldsymbol{r}(t+\Delta t) = \boldsymbol{r}(t) + \boldsymbol{v}(t+\tfrac{1}{2}\Delta t)\Delta t$$

$$\boldsymbol{v}(t+\tfrac{1}{2}\Delta t) = \boxed{\boldsymbol{v}(t-\tfrac{1}{2}\Delta t)+\tfrac{1}{m}\boldsymbol{F}(t)\Delta t}$$

- Mathematically equivalent to Verlet algorithm

$$\boldsymbol{r}(t+\Delta t) = \boldsymbol{r}(t) + \left[\boldsymbol{v}(t-\tfrac{1}{2}\Delta t)+\tfrac{1}{m}\boldsymbol{F}(t)\Delta t\right]\Delta t$$

$$\boldsymbol{r}(t) = \boldsymbol{r}(t-\Delta t) + \boldsymbol{v}(t-\tfrac{1}{2}t)\Delta t$$

Leapfrog: Issues

- Initialization
  - Simple approximation to get velocity at first time step:

    $$\boldsymbol{v}(t_0 - \tfrac{1}{2}\Delta t) \equiv \boldsymbol{v}(t_0) - \frac{1}{m}\boldsymbol{F}(t_0)\tfrac{1}{2}\Delta t$$

- Obtaining the velocities
  - Interpolate
  - $$\boldsymbol{v}(t) = \frac{1}{2}\left(\boldsymbol{v}(t + \tfrac{1}{2}\Delta t) + \boldsymbol{v}(t - \tfrac{1}{2}\Delta t)\right)$$

The Leapfrog

For a second order ODE: $\ddot{\mathbf{x}} = f(\mathbf{x})$

"Drift-Kick-Drift" version

$$\begin{aligned} x_{n+\frac{1}{2}} &= x_n + v_n \frac{\Delta t}{2} \\ v_{n+1} &= v_n + f(x_{n+\frac{1}{2}}) \Delta t \\ x_{n+1} &= x_{n+\frac{1}{2}} + v_{n+1} \frac{\Delta t}{2} \end{aligned}$$

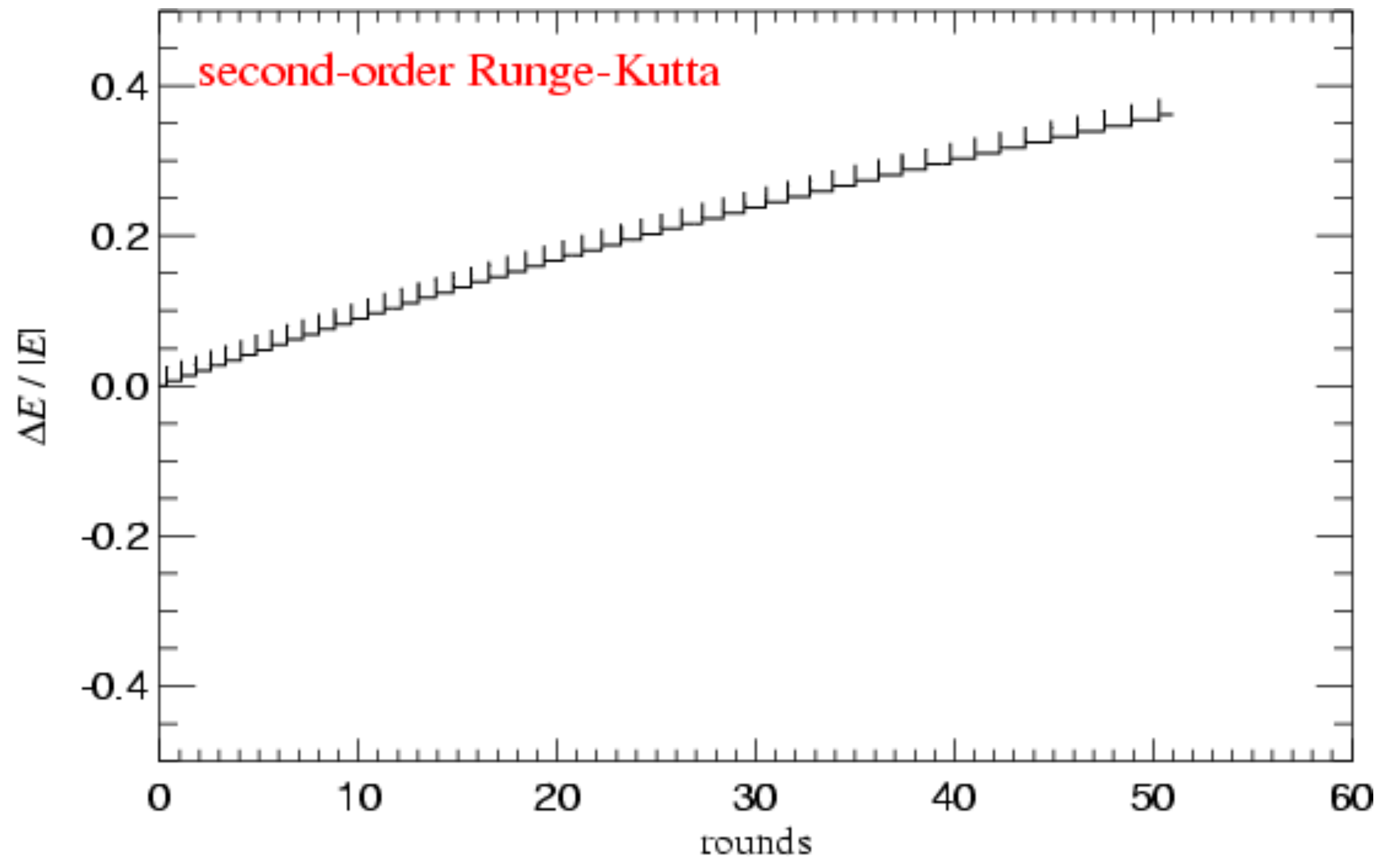
"Kick-Drift-Kick" version

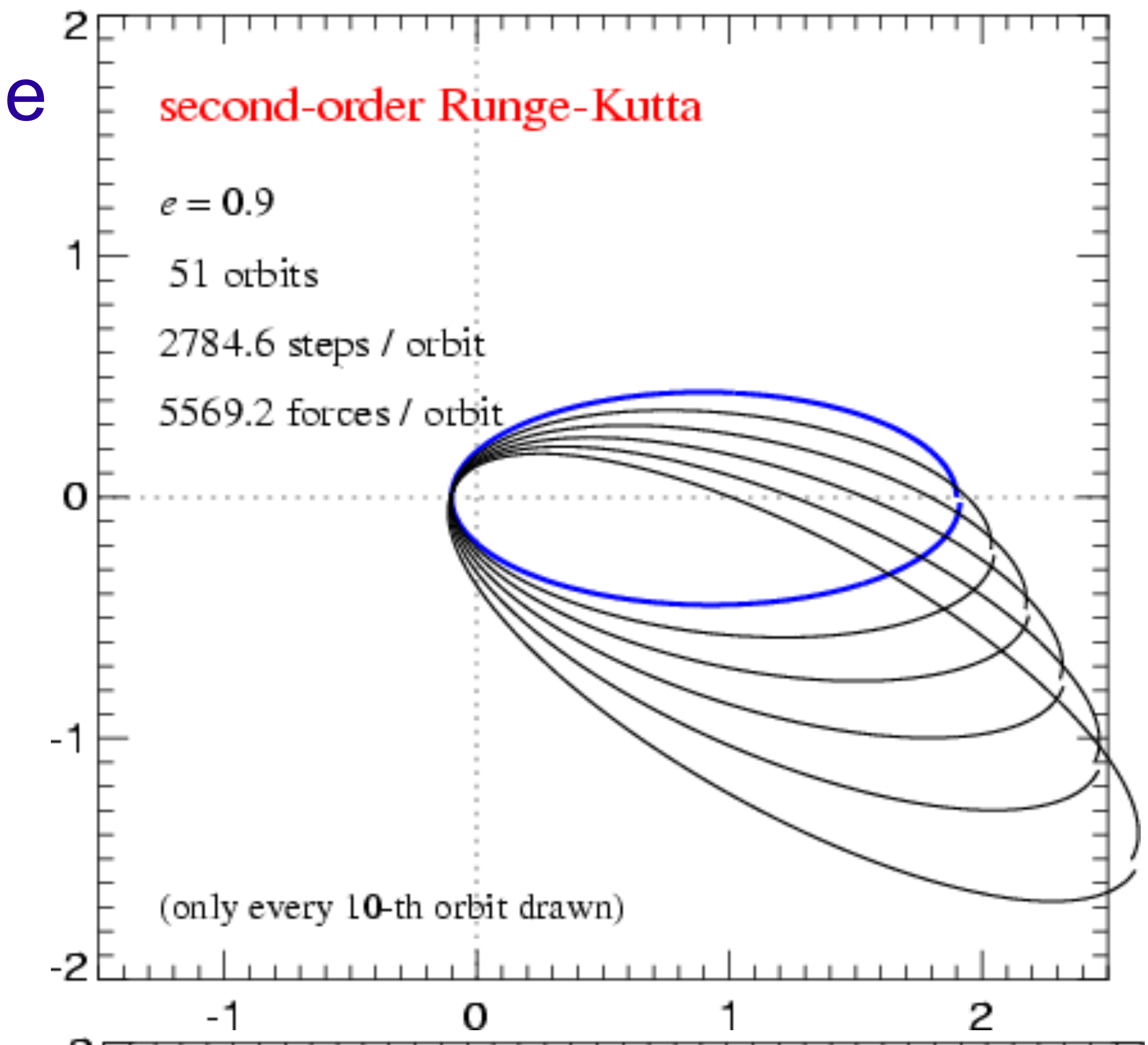
$$\begin{aligned} v_{n+\frac{1}{2}} &= v_n + f(x_n) \frac{\Delta t}{2} \\ x_{n+1} &= x_n + v_{n+\frac{1}{2}} \frac{\Delta t}{2} \\ v_{n+1} &= v_{n+\frac{1}{2}} + f(x_{n+1}) \frac{\Delta t}{2} \end{aligned}$$

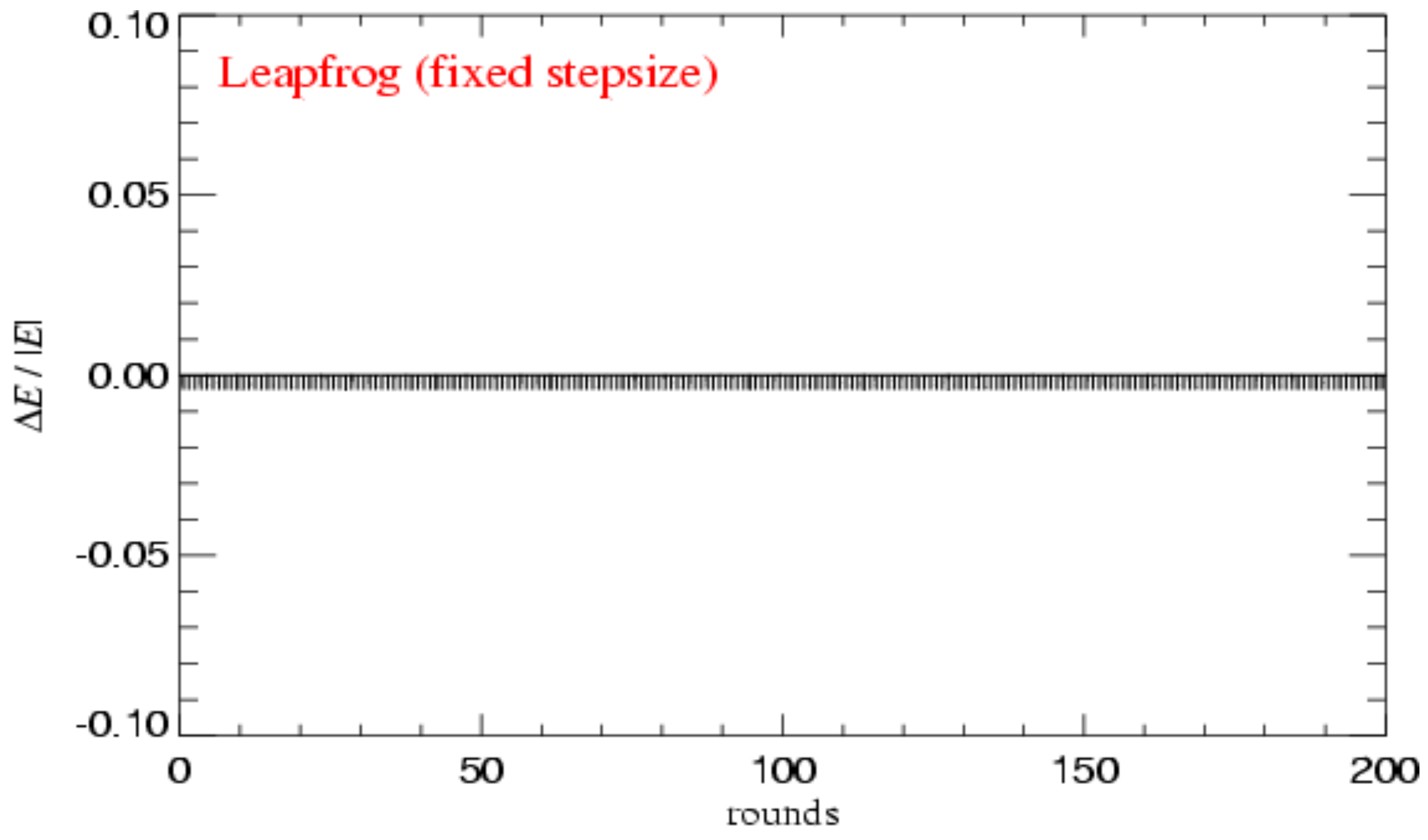
- **2nd order accurate**
- **symplectic**
- can be rewritten into time-centred formulation

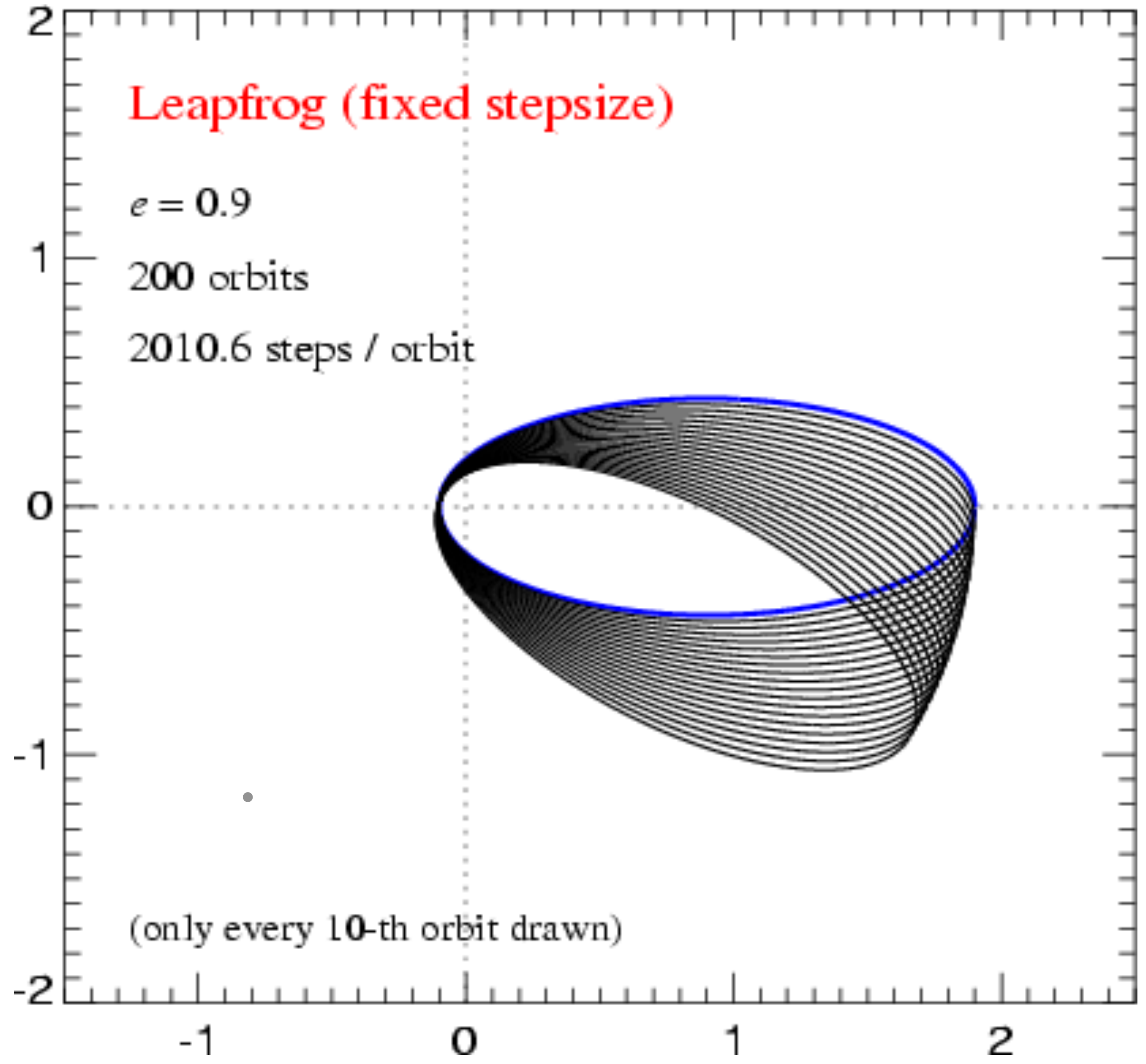
When compared with an integrator of the same order, the leapfrog is highly superior

INTEGRATING THE KEPLER PROBLEM



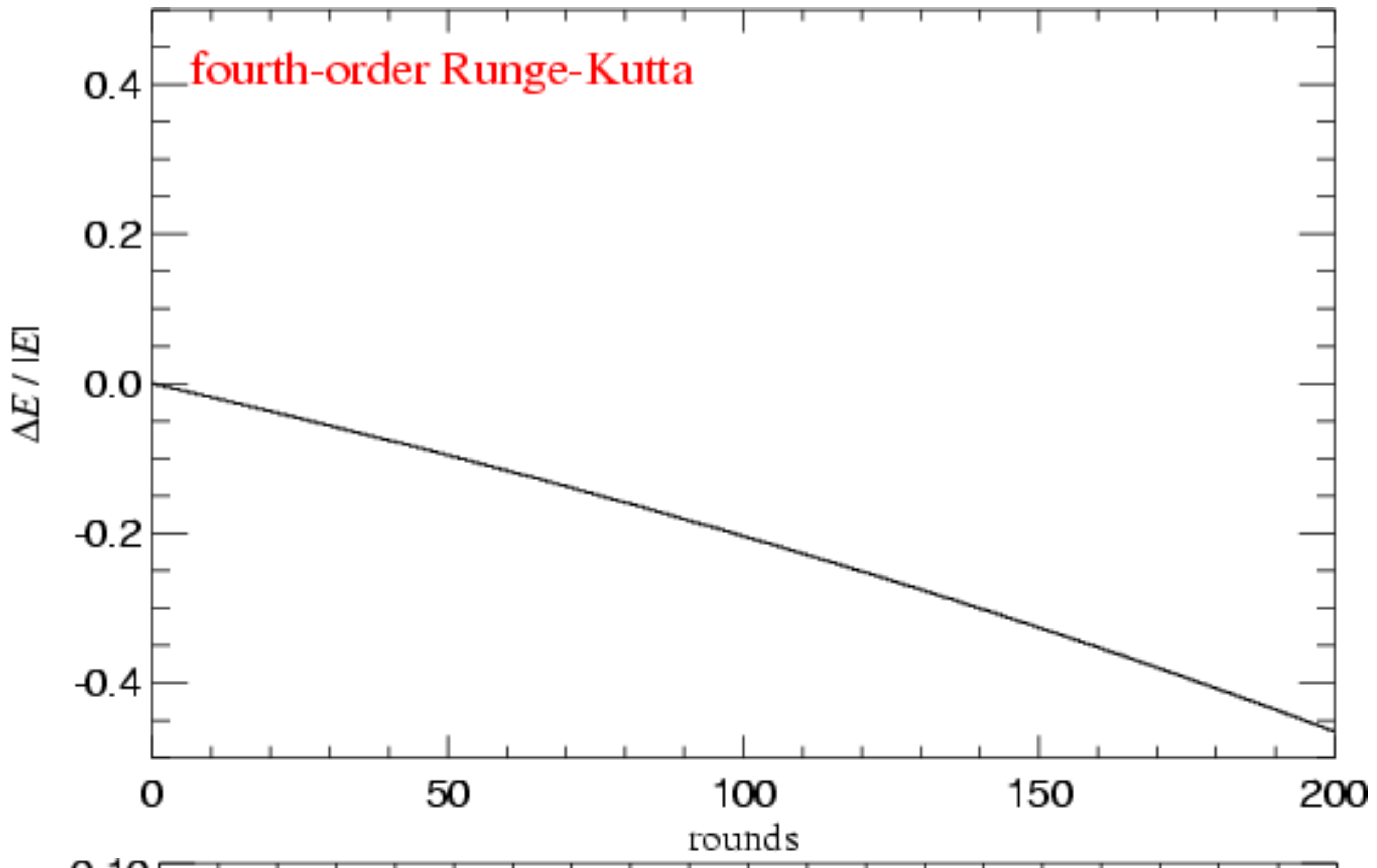


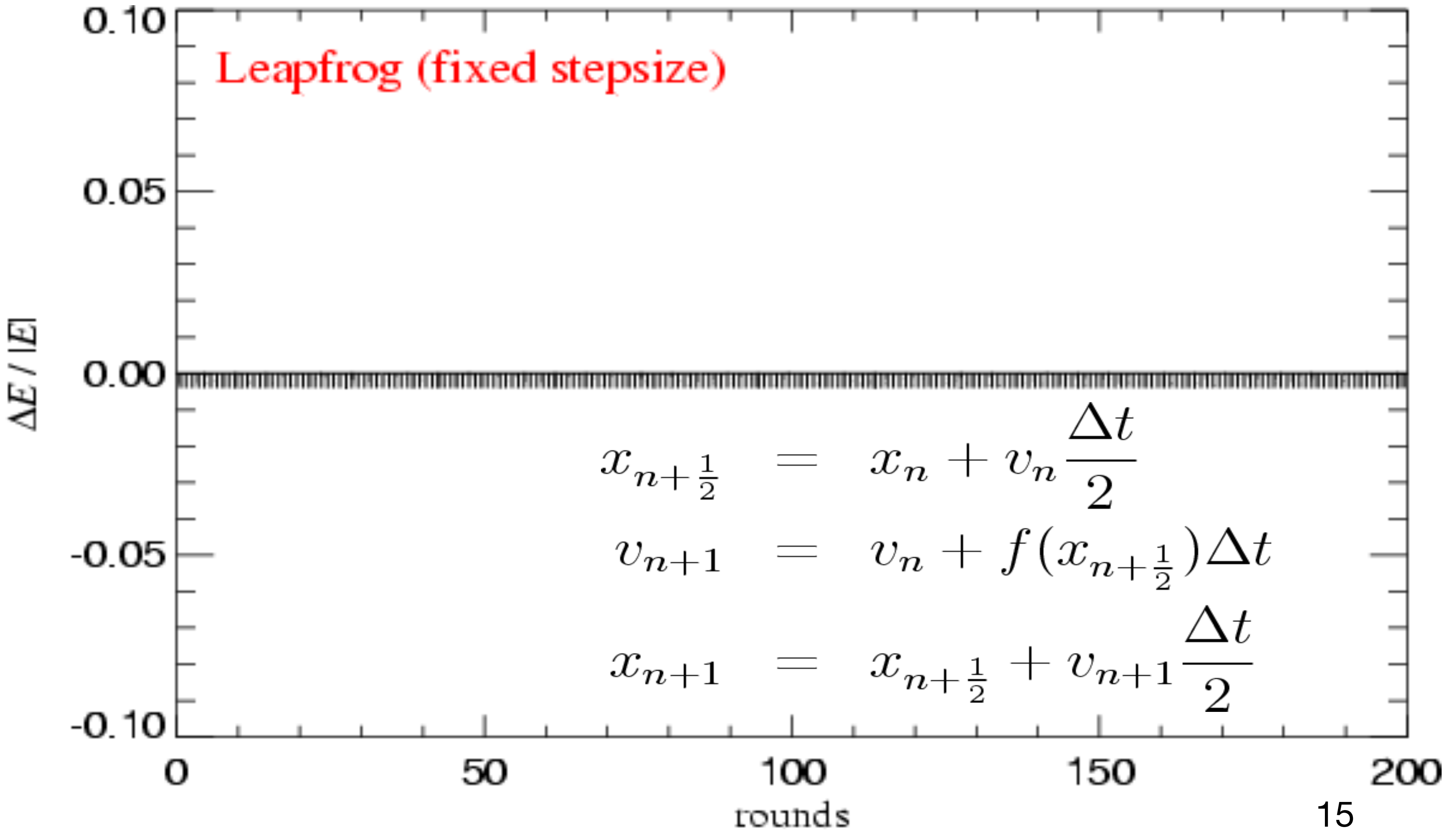




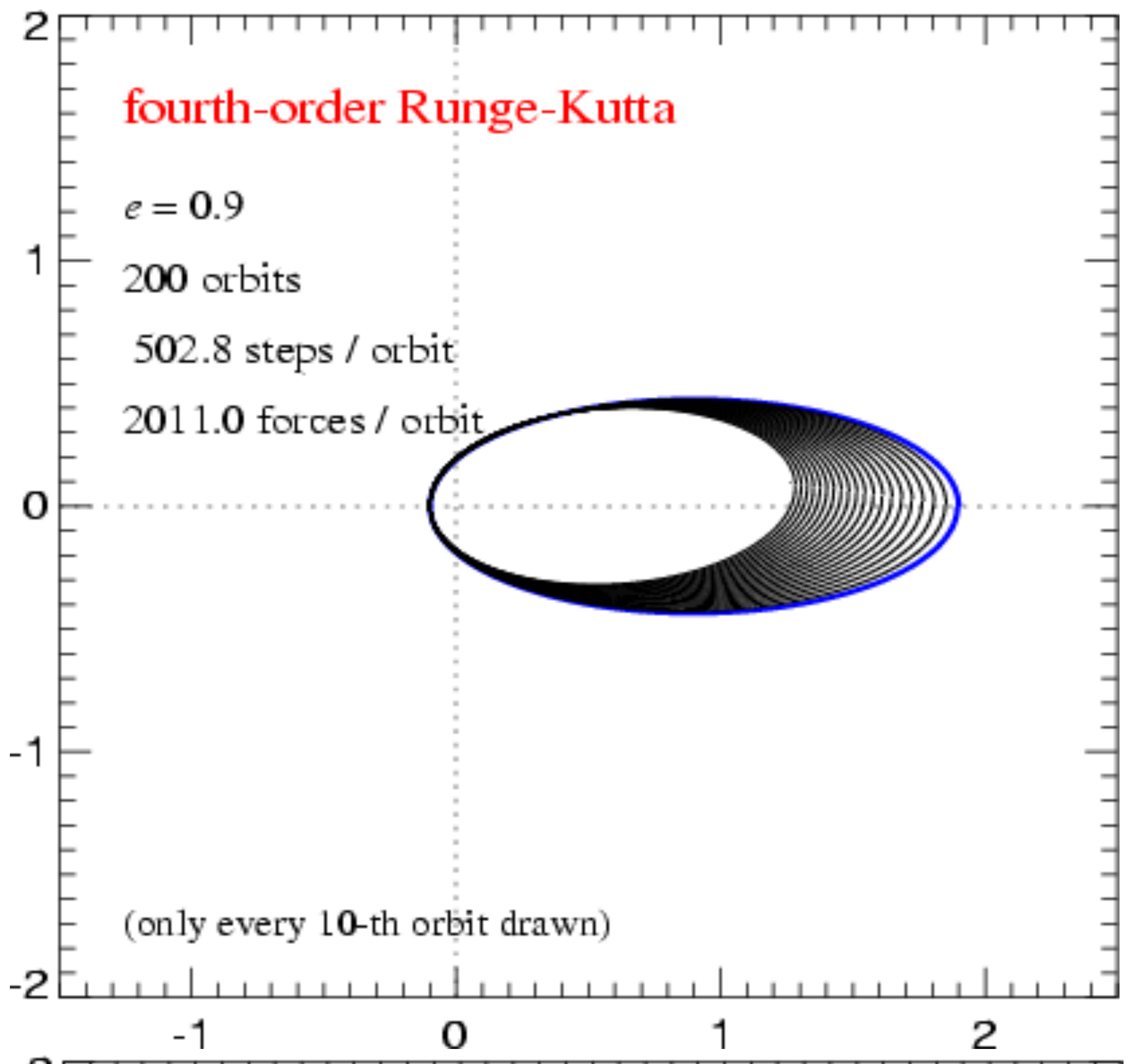
The leapfrog is behaving much better than one might expect...

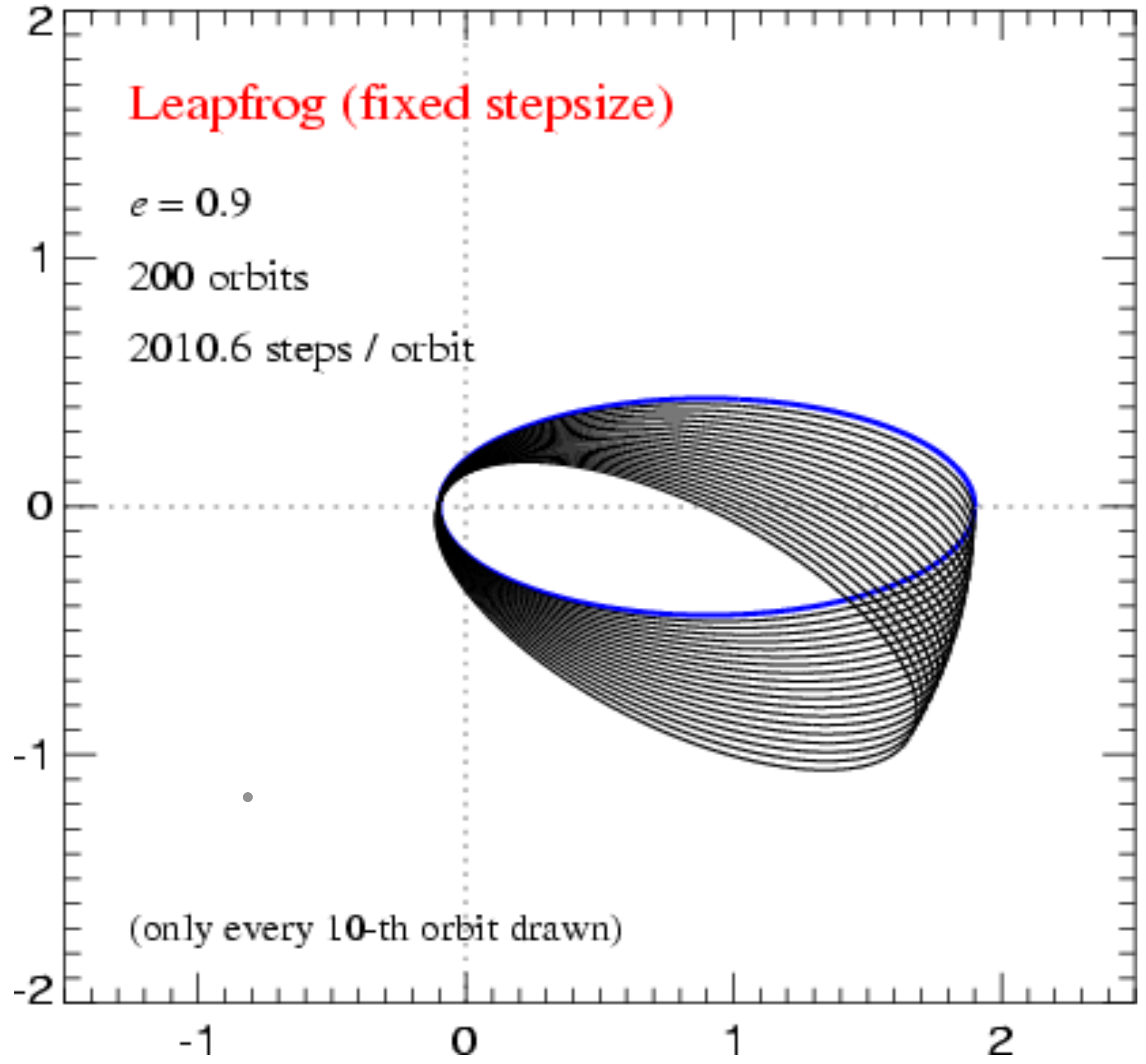
INTEGRATING THE KEPLER PROBLEM





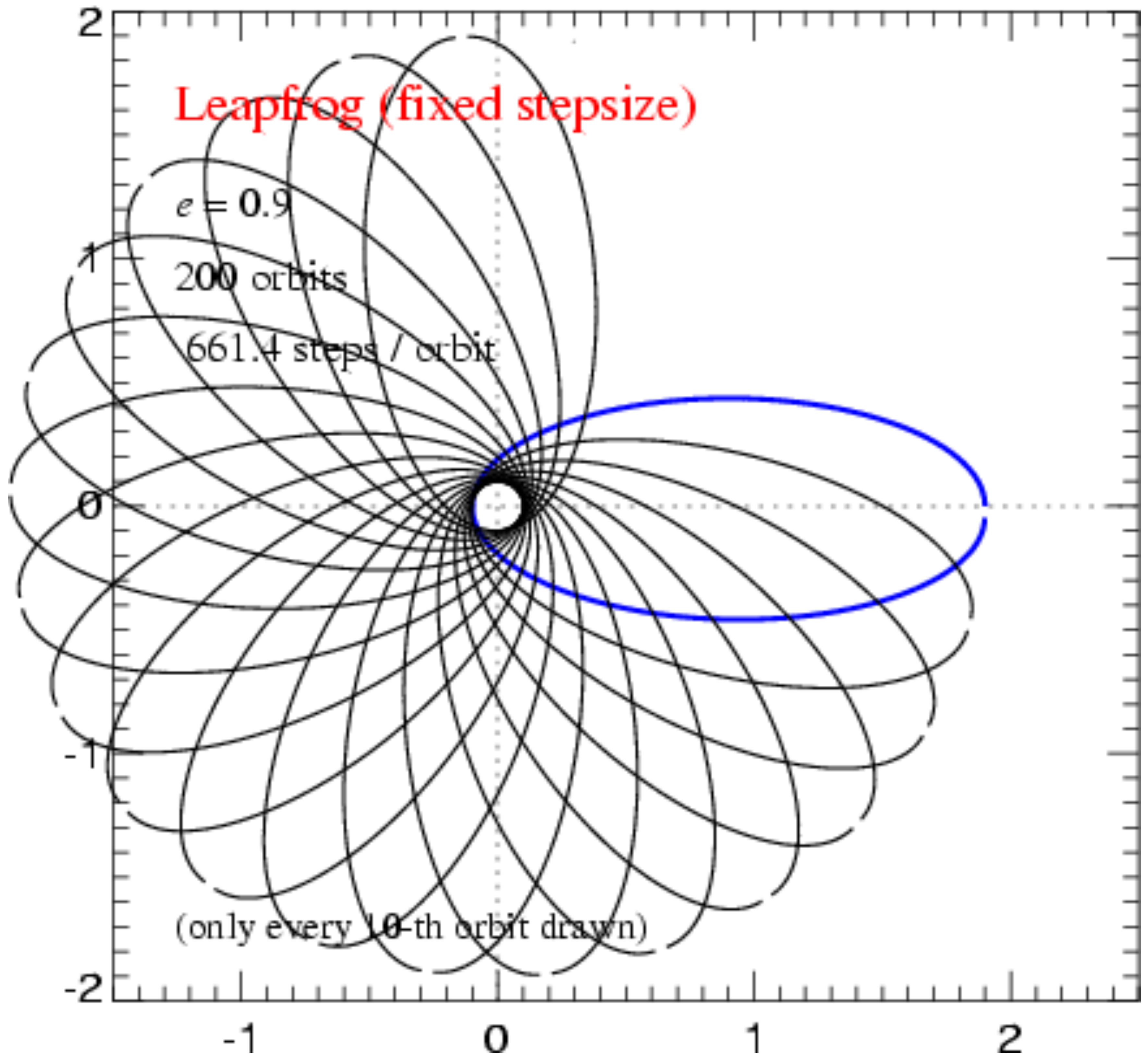
$$
\begin{aligned}
x_{n+\frac{1}{2}} &= x_n + v_n \frac{\Delta t}{2} \\
v_{n+1} &= v_n + f(x_{n+\frac{1}{2}})\Delta t \\
x_{n+1} &= x_{n+\frac{1}{2}} + v_{n+1} \frac{\Delta t}{2}
\end{aligned}
$$

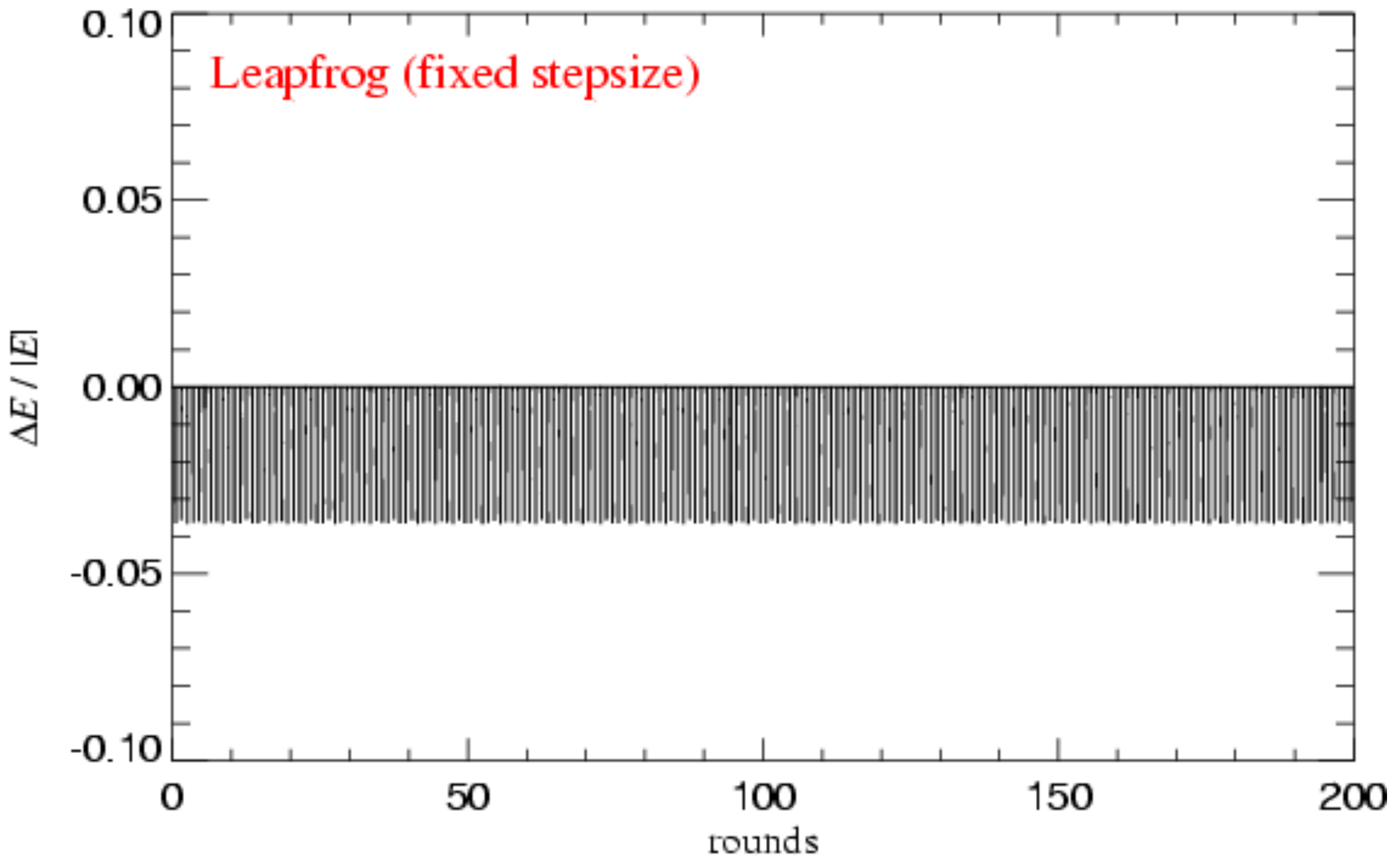




Even for rather large timesteps, the leapfrog maintains qualitatively correct behaviour without long-term secular trends

INTEGRATING THE KEPLER PROBLEM

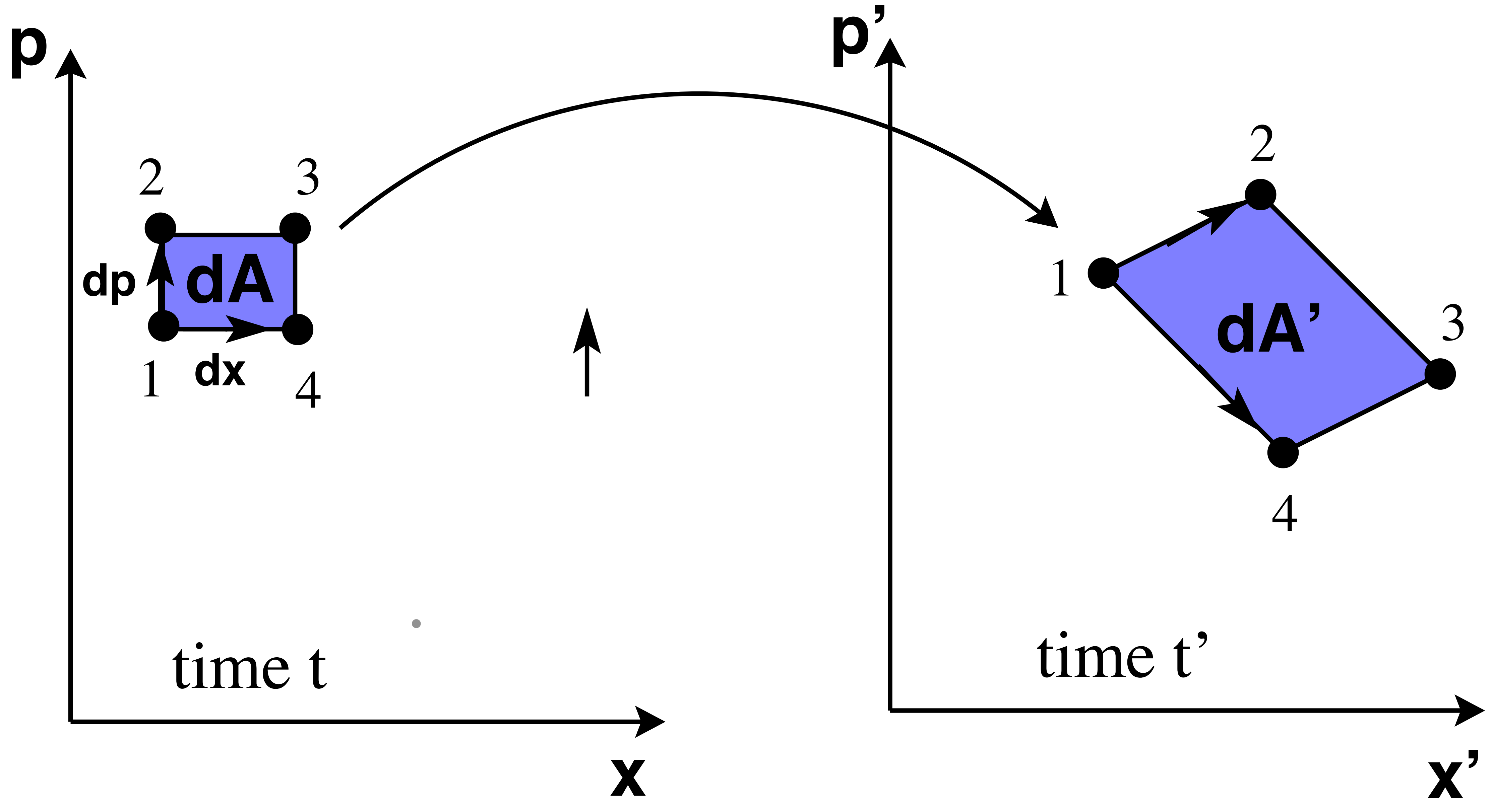




Advantages

- Advances of leapfrog and verlet algorithms
  - Time-reversal invariant
  - Conserves angular momentum
  - **Symplectic** (i.e. phase-space area preserving)
    - Euler, RK2, and RK4 are not!

-



What is the underlying mathematical reason for the very good long-term behaviour of the leapfrog ?

HAMILTONIAN SYSTEMS AND SYMPLECTIC INTEGRATION

$$H(\mathbf{p}_1, \ldots, \mathbf{p}_n, \mathbf{x}_1, \ldots, \mathbf{x}_n) = \sum_i \frac{\mathbf{p}_i^2}{2m_i} + \frac{1}{2}\sum_{ij} m_i m_j \phi(\mathbf{x}_i - \mathbf{x}_j)$$

If the integration scheme introduces non-Hamiltonian perturbations, a completely different long-term behaviour results.

The Hamiltonian structure of the system can be preserved in the integration if each step is formulated as a *canoncial transformation*. Such integration schemes are called *symplectic*.

Poisson bracket:

$$\{A, B\} \equiv \sum_i \left(\frac{\partial A}{\partial \mathbf{x}_i}\frac{\partial B}{\partial \mathbf{p}_i} - \frac{\partial A}{\partial \mathbf{p}_i}\frac{\partial B}{\partial \mathbf{x}_i} \right)$$

Hamilton's equations

$$\frac{\mathrm{d}\mathbf{x}_i}{\mathrm{d}t} = \{\mathbf{x}_i, H\}$$

$$\frac{\mathrm{d}\mathbf{p}_i}{\mathrm{d}t} = \{\mathbf{p}_i, H\}$$

Hamilton operator

$$\mathbf{H} f \equiv \{f, H\}$$

System state vector

$$|t\rangle \equiv |\mathbf{x}_1(t), \ldots, \mathbf{x}_n(t), \mathbf{p}_1(t), \ldots, \mathbf{p}_n(t), t\rangle$$

Time evolution operator

$$|t_1\rangle = \mathbf{U}(t_1, t_0)\ |t_0\rangle$$

$$\mathbf{U}(t+\Delta t, t) = \exp\left(\int_t^{t+\Delta t} \mathbf{H}\, \mathrm{d}t\right)$$

The time evolution of the system is a continuous canonical transformation generated by the Hamiltonian.

Symplectic integration schemes can be generated by applying the idea of operating splitting to the Hamiltonian

THE LEAPFROG AS A SYMPLECTIC INTEGRATOR

Separable Hamiltonian

$$H = H_{\rm kin} + H_{\rm pot}$$

Drift- and Kick-Operators

$$\mathbf{D}(\Delta t) \equiv \exp\left(\int_t^{t+\Delta t} {\rm d}t\, \mathbf{H}_{\rm kin}\right) = \begin{cases} \mathbf{p}_i & \mapsto \quad \mathbf{p}_i \\ \mathbf{x}_i & \mapsto \quad \mathbf{x}_i + \frac{\mathbf{p}_i}{m_i}\Delta t \end{cases}$$

$$\mathbf{K}(\Delta t) = \exp\left(\int_t^{t+\Delta t} {\rm d}t\, \mathbf{H}_{\rm pot}\right) = \begin{cases} \mathbf{x}_i & \mapsto \quad \mathbf{x}_i \\ \mathbf{p}_i & \mapsto \quad \mathbf{p}_i - \sum_j m_i m_j \frac{\partial\phi(\mathbf{x}_{ij})}{\partial \mathbf{x}_i}\Delta t \end{cases}$$

The drift and kick operators are symplectic transformations of phase-space !

The Leapfrog

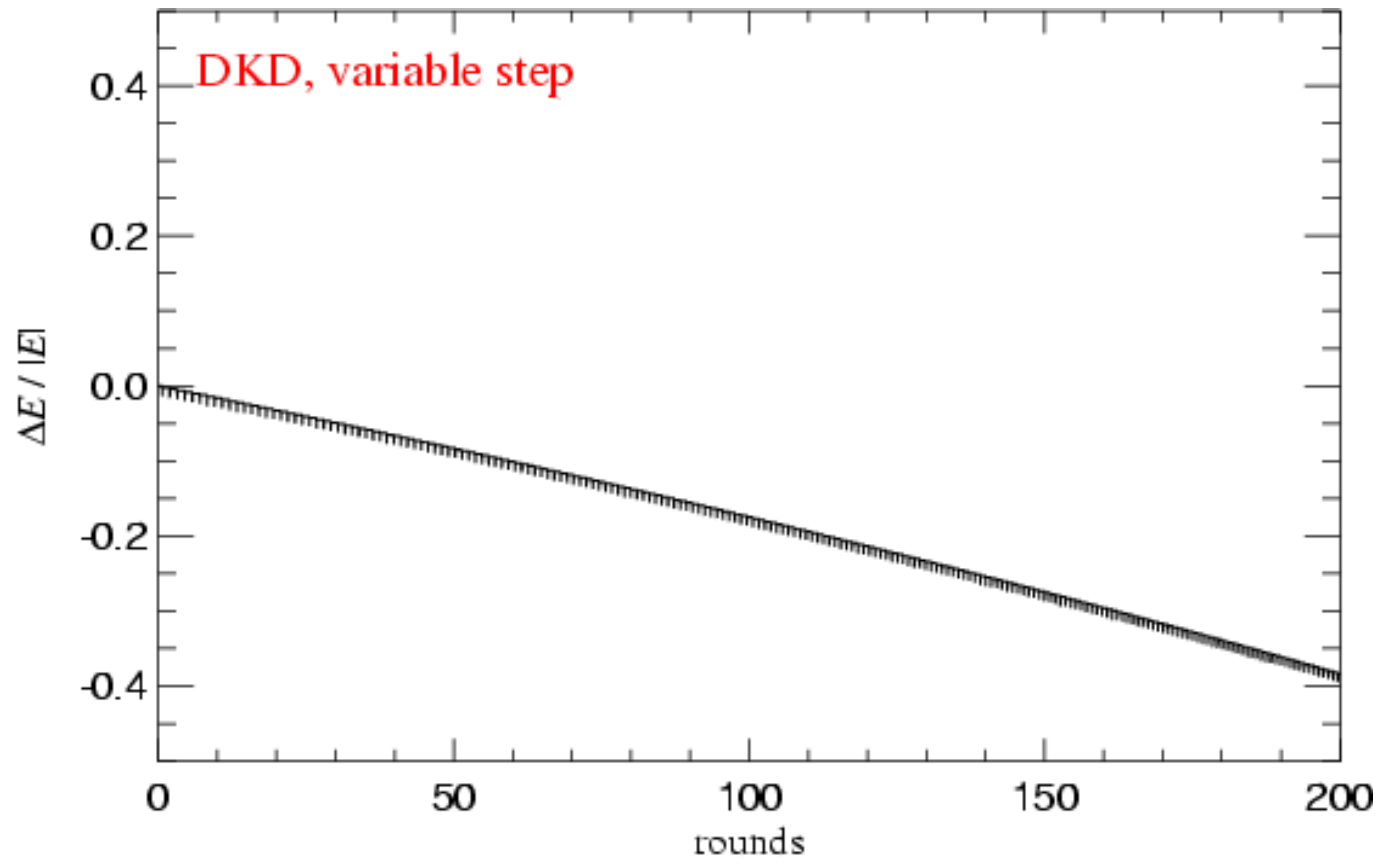
Drift-Kick-Drift:
$$\tilde{\mathbf{U}}(\Delta t) = \mathbf{D}\left(\frac{\Delta t}{2}\right)\mathbf{K}(\Delta t)\,\mathbf{D}\left(\frac{\Delta t}{2}\right)$$

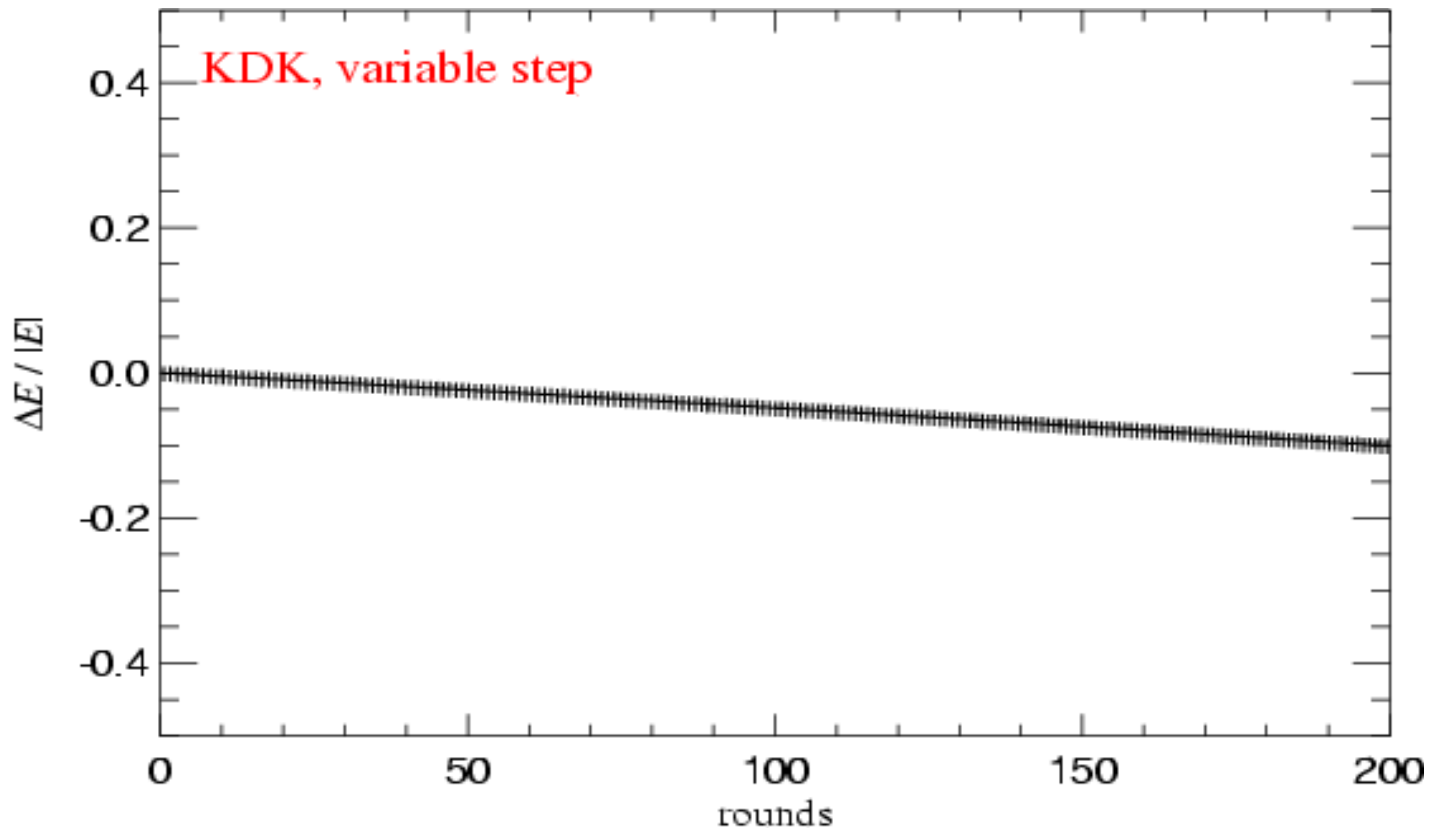
Kick-Drift-Kick:
$$\tilde{\mathbf{U}}(\Delta t) = \mathbf{K}\left(\frac{\Delta t}{2}\right)\mathbf{D}(\Delta t)\,\mathbf{K}\left(\frac{\Delta t}{2}\right)$$

Hamiltonian of the numerical system:
$$\tilde{H} = H + H_{\rm err} \qquad H_{\rm err} = \frac{\Delta t^2}{12}\left\{\{H_{\rm kin}, H_{\rm pot}\}, H_{\rm kin} + \frac{1}{2}H_{\rm pot}\right\} + \mathcal{O}(\Delta t^3)$$

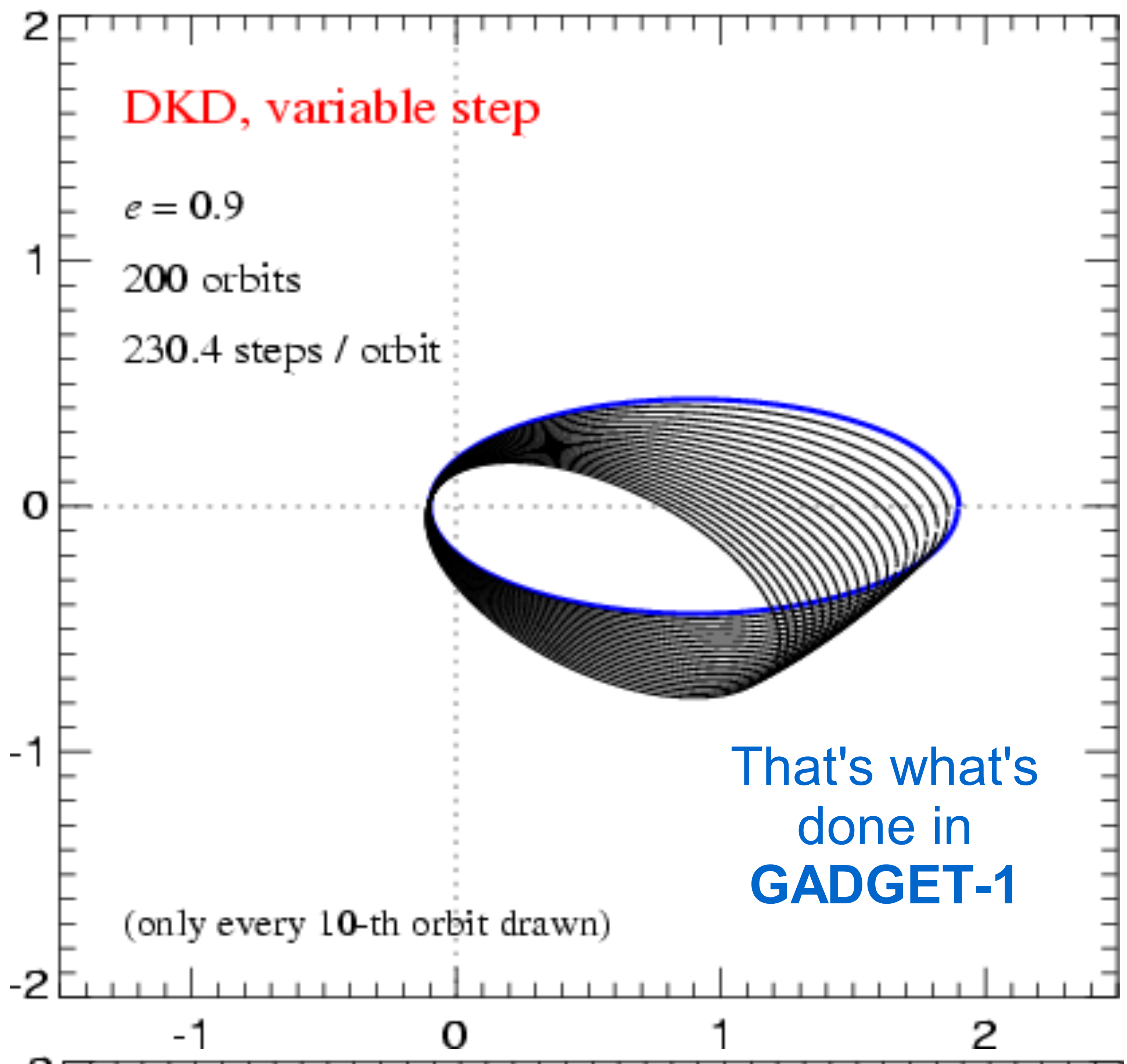
When an adaptive timestep is used, much of the symplectic advantage is lost

INTEGRATING THE KEPLER PROBLEM

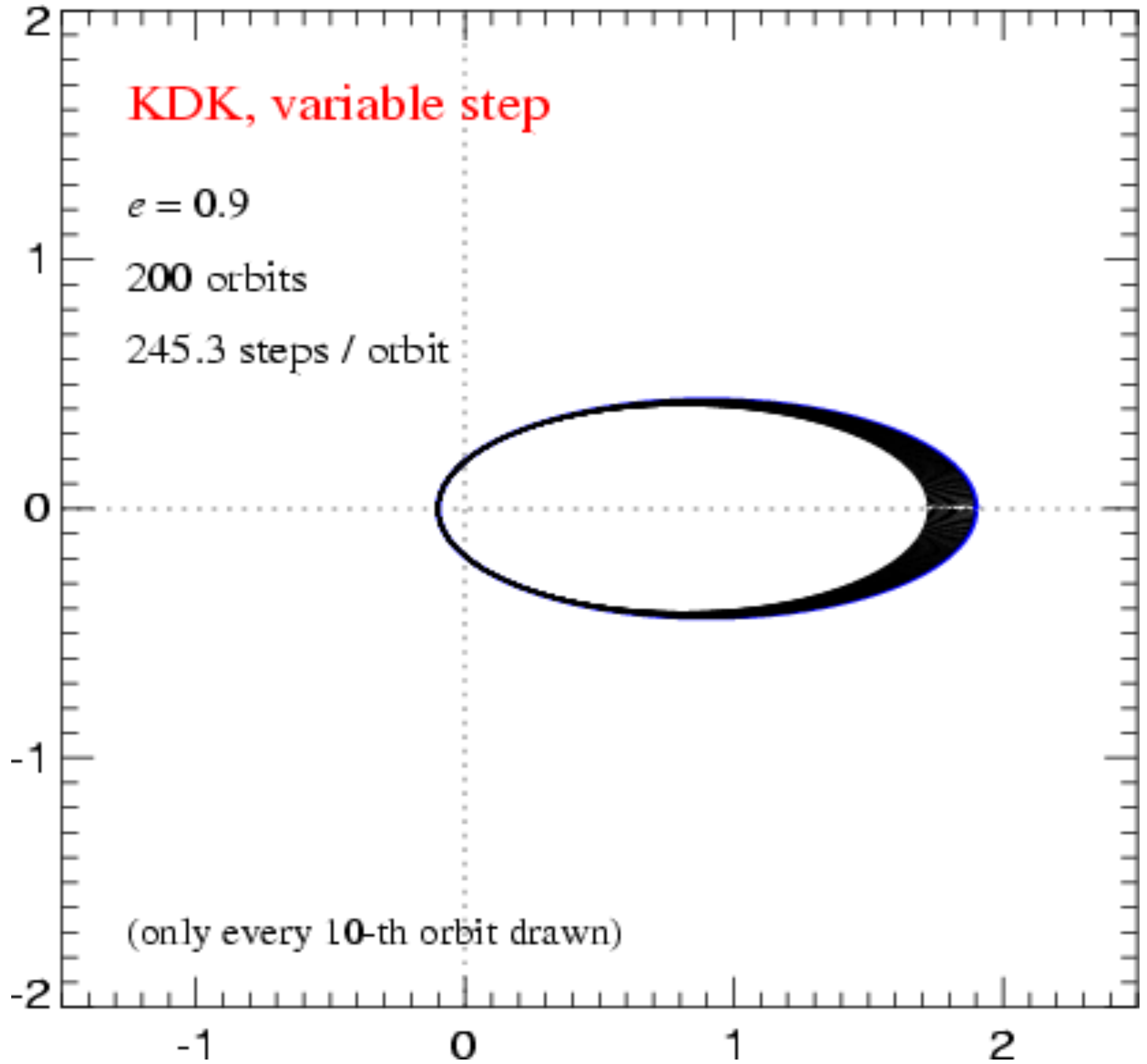




→ Going to KDK reduces the error by a factor 4, at the same cost !



That's what's done in **GADGET-1**



For periodic motion with adaptive timesteps, the DKD leapfrog shows more time-asymmetry than the KDK variant

LEAPFROG WITH ADAPTIVE TIMESTEP