PHYS 141/241

Lecture 03: Numerical Integration Methods

Javier Duarte — April 7, 2023

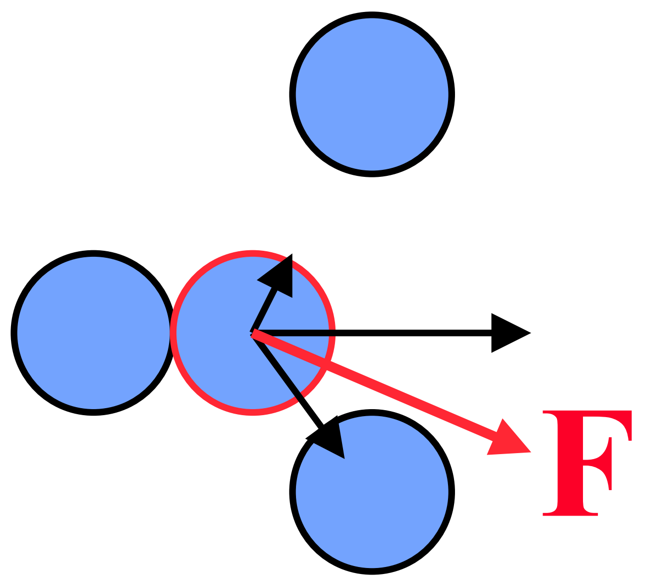
Integration algorithms

- Equations of motion in cartesian coordinates

$$\frac{d\boldsymbol{r}_j}{dt} = \frac{\boldsymbol{p}_j}{m}$$

$$\frac{d\boldsymbol{p}_j}{dt} = \boldsymbol{F}_j = \sum_{i=1,\, i \neq j}^{N} \boldsymbol{F}_{ij}$$

(pairwise additive forces)



- Desirable features of an integrator
  - Minimal need to compute forces (expensive)
  - Good ability for large time steps
  - Good accuracy
  - Conserves energy and momentum
  - Time-reversible
  - Phase space area-preserving (symplectic)

More on these later

Time integration methods

- Let's integrate a first-order ordinary differential equation (ODE):

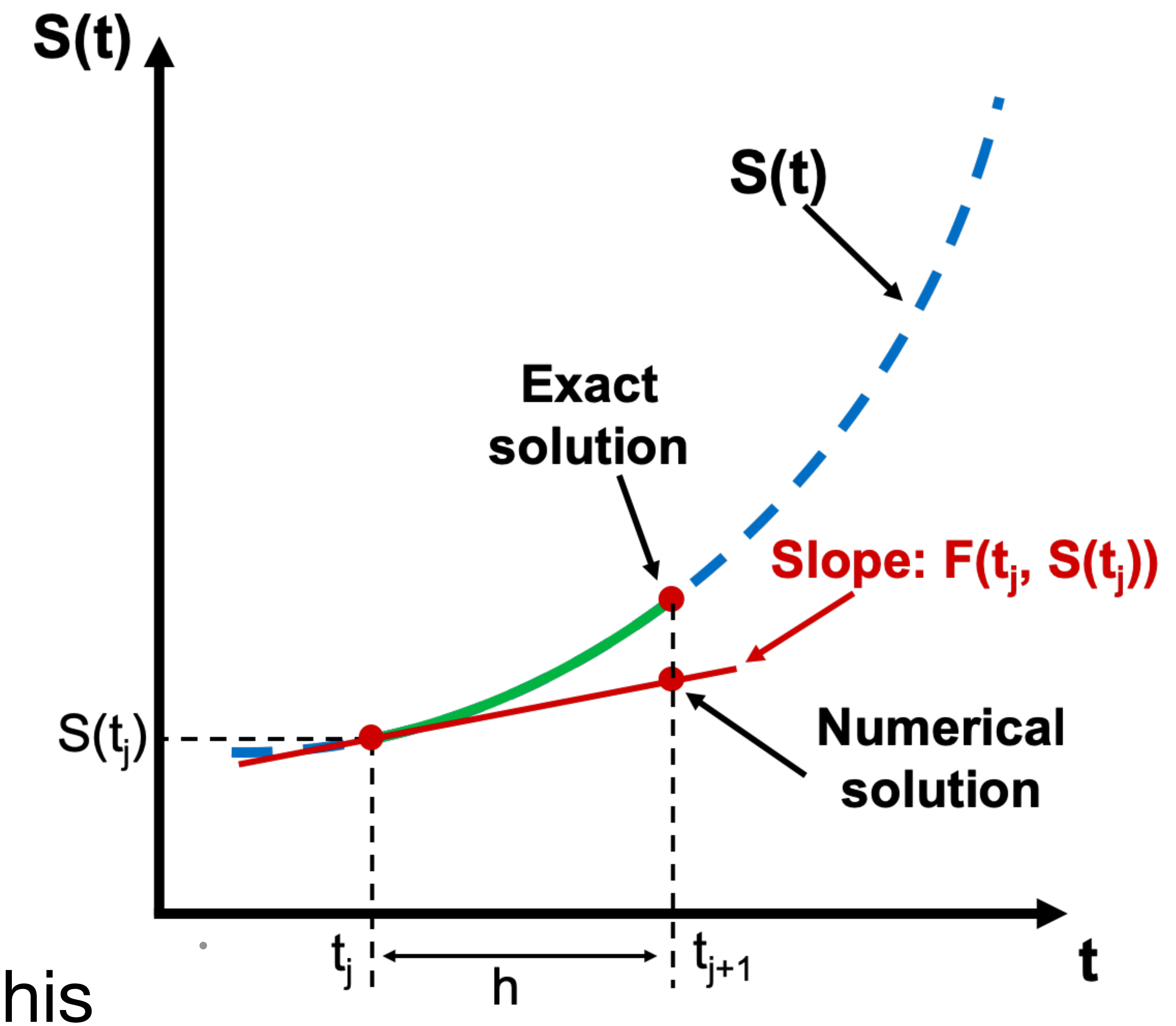
$$\frac{dS}{dt} = \dot{S}(t) = F(t, S(t))$$

- Example: exponential function

$$\dot{y} = e^{-t}$$

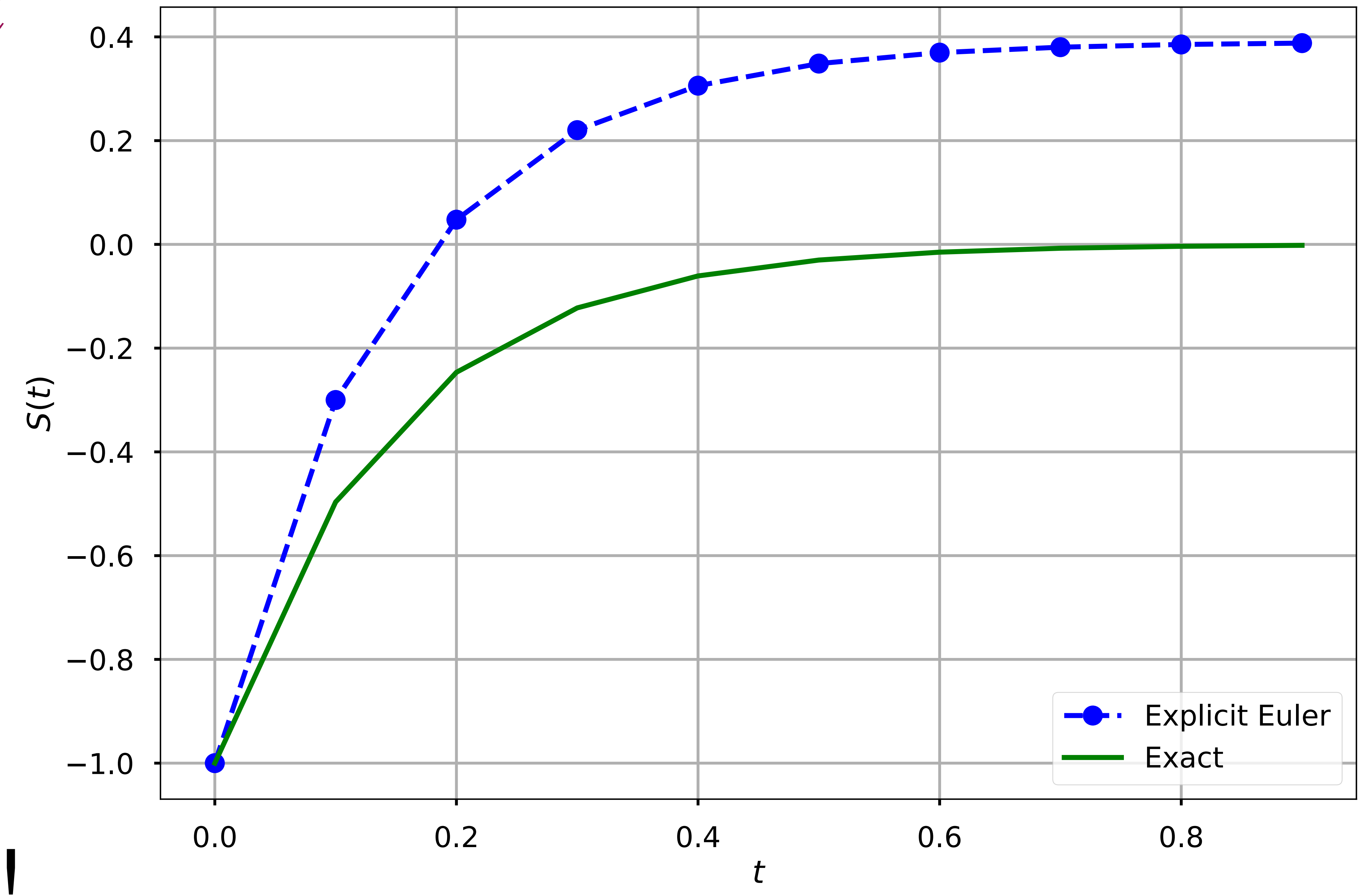
- A numerical approximation to the ODE is a set of values: $\{S_0, S_1, S_2, \ldots\}$ and $\{t_0, t_1, t_2, \ldots\}$

- There are many different ways of obtaining this



Euler method

- Explicit Euler method: $S_{n+1} = S_n + F(t_n, y_n)\Delta t$
  - Simplest of all
  - Right-hand side depends on things already known: **explicit** method
  - The error in a single step is $\mathcal{O}(\Delta t^2)$, but for N steps needed for a finite time interval, the total error scales as $\mathcal{O}(\Delta t)$!
  - Only first-order accurate, not advised to use!



- Implicit Euler method: $S_{n+1} = S_n + F(t_{n+1}, y_{n+1})\Delta t$
  - Excellent stability properties
  - Suitable for stiff ODE
  - Requires implicit solver for y_{n+1} (i.e. more computations)

Predictor-corrector methods

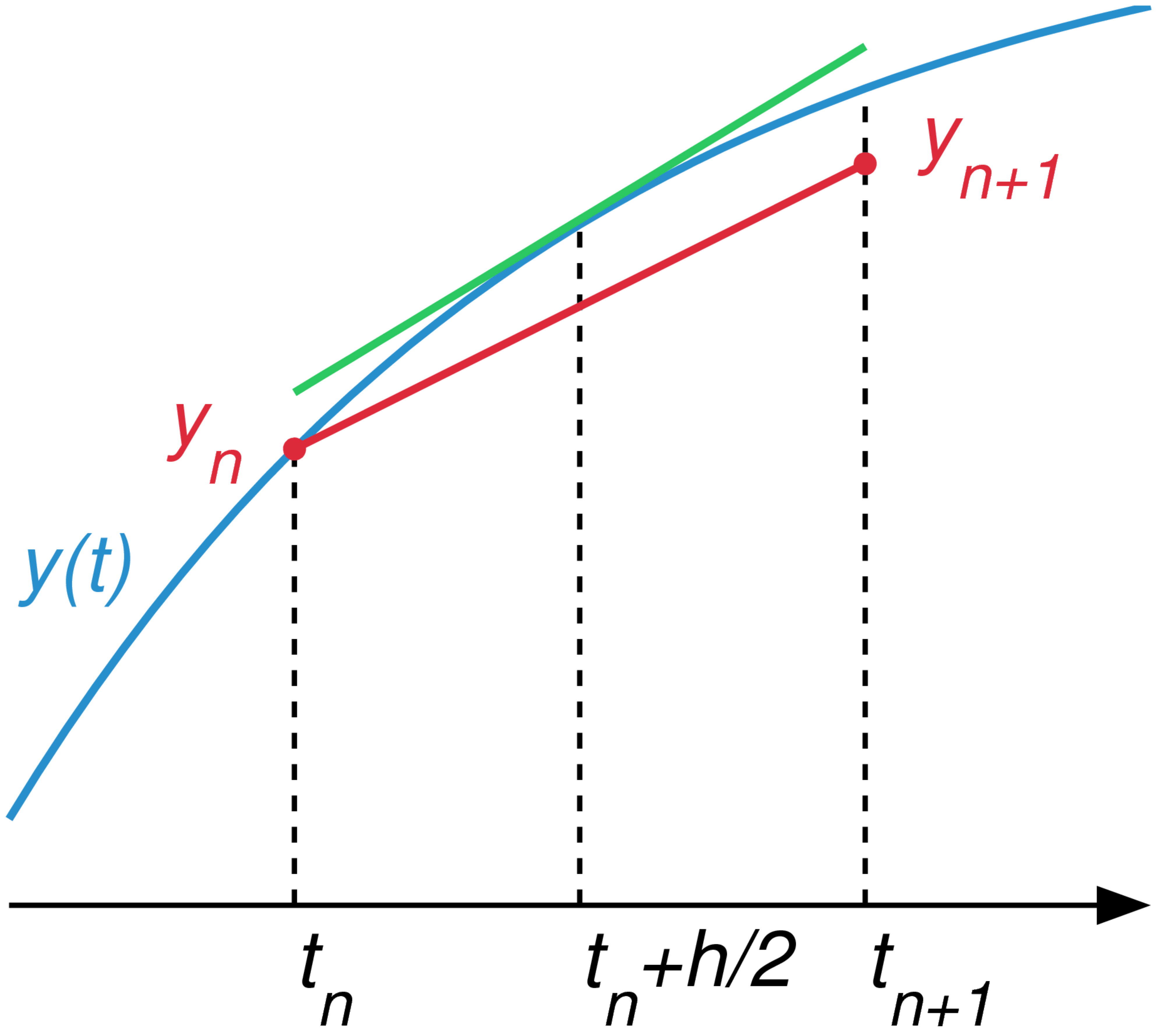
- Predictor-corrector methods of solving initial value problems improve the approximation accuracy by querying the function several times at different locations (predictions), and then using a weighted average of the results (corrections) to update the state
- Two formulas: the predictor and corrector
  - The **predictor** is an explicit formula and first estimates the solution at t, i.e. we can use Euler method or some other methods to finish this step.
  - Using the found $S(t_{n+1})$, the **corrector** can calculate a new, more accurate solution

Midpoint method

- Implicit midpoint method:

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

  - 2nd-order accurate
  - Time symmetric and **symplectic**
  - But still implicit



- Explicit midpoint method

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

Midpoint vs Euler

- Euler uses the slope formula

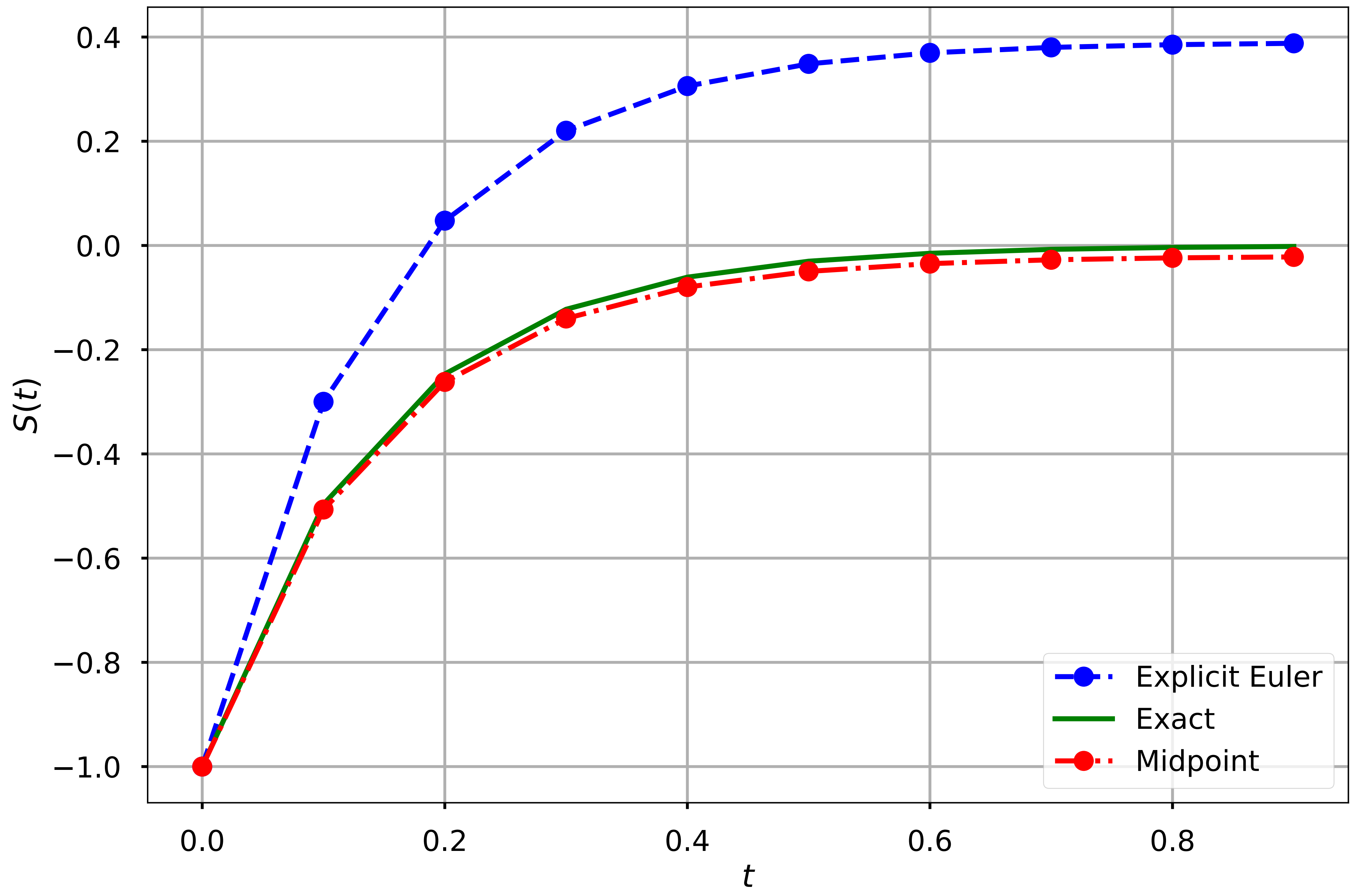
$$y'(t) \approx \frac{y(t+h) - y(t)}{h}$$

to derive $y(t+h) \approx y(t) + hf(t, y(t))$

- Midpoint replaces this with the more accurate

$$y'(t+h/2) \approx \frac{y(t+h) - y(t)}{h}$$

to derive $y(t+h) \approx y(t) + hf(t+h/2, y(t+h/2))$



Runge-Kutta motivation

- Runge-Kutta (RK) methods are one of the most widely used methods for solving ODEs
- Euler method uses the first two terms in Taylor series to approximate the numerical integration

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

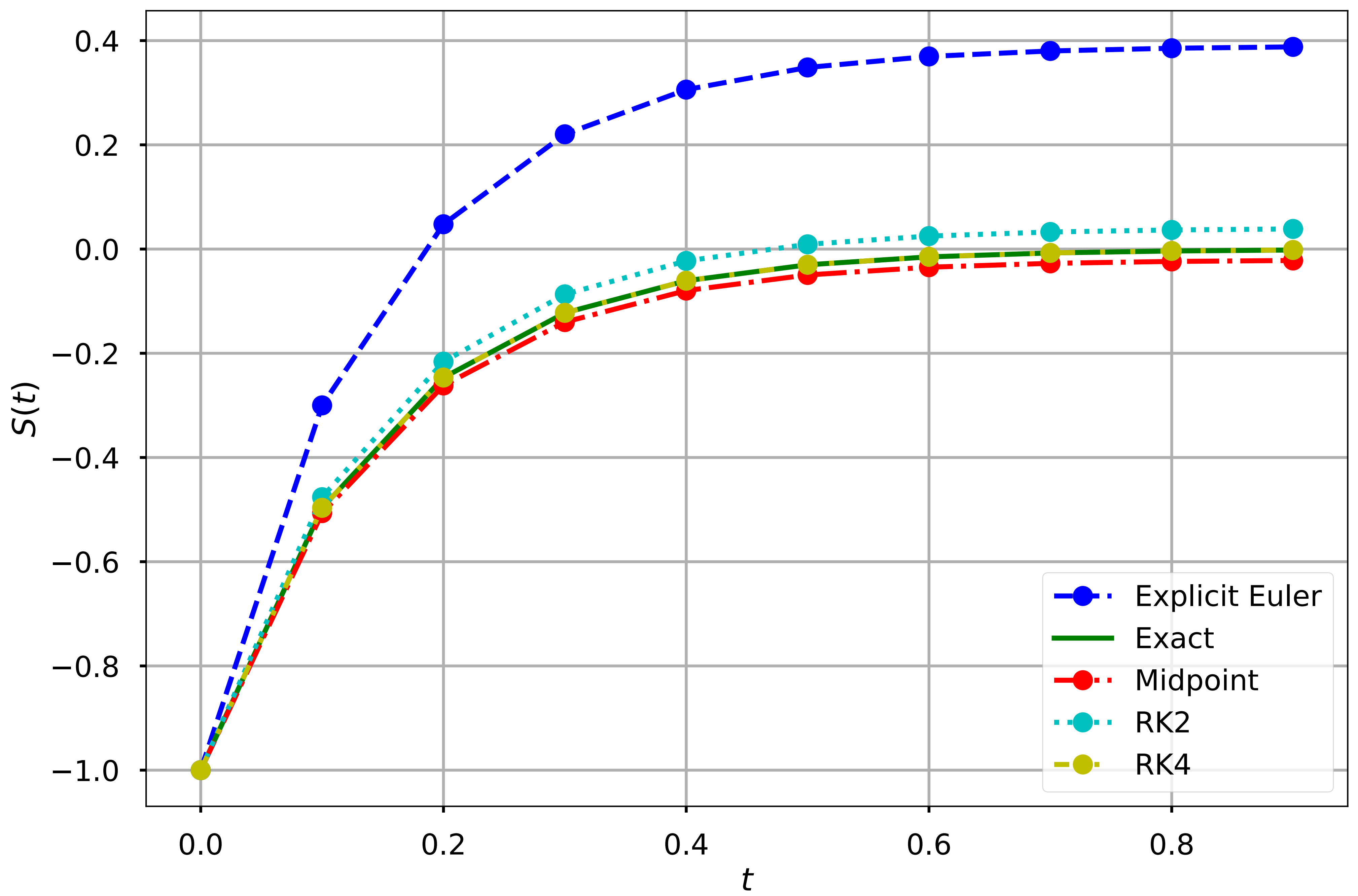
- We can improve the accuracy of the numerical integration if we keep more terms!

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

- In order to get this more accurate solution, we need to derive expressions for the higher order derivatives

Runge-Kutta methods

- Runge-Kutta methods:
  - Whole class of integration methods



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

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

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

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

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

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

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

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

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

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

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

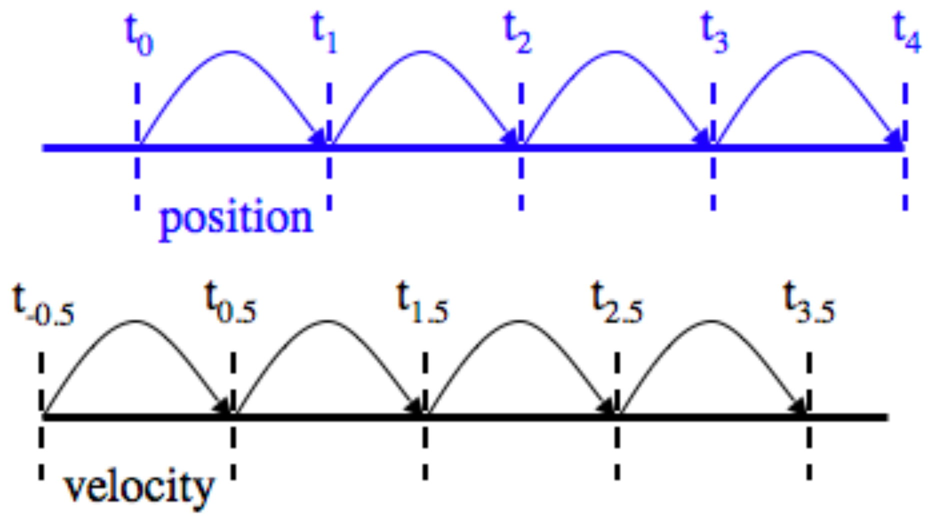
$$\underbrace{\boldsymbol{r}(t + \Delta t) - \boldsymbol{r}(t)}_{O(\Delta t^1)} = \underbrace{\boldsymbol{r}(t)}_{O(\Delta t^0)} + -\underbrace{\boldsymbol{r}(t - \Delta t)}_{O(\Delta t^0)} + \frac{1}{m}\boldsymbol{F}(t)\Delta t^2 .$$

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

The Leapfrog

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

"Drift-Kick-Drift" version

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

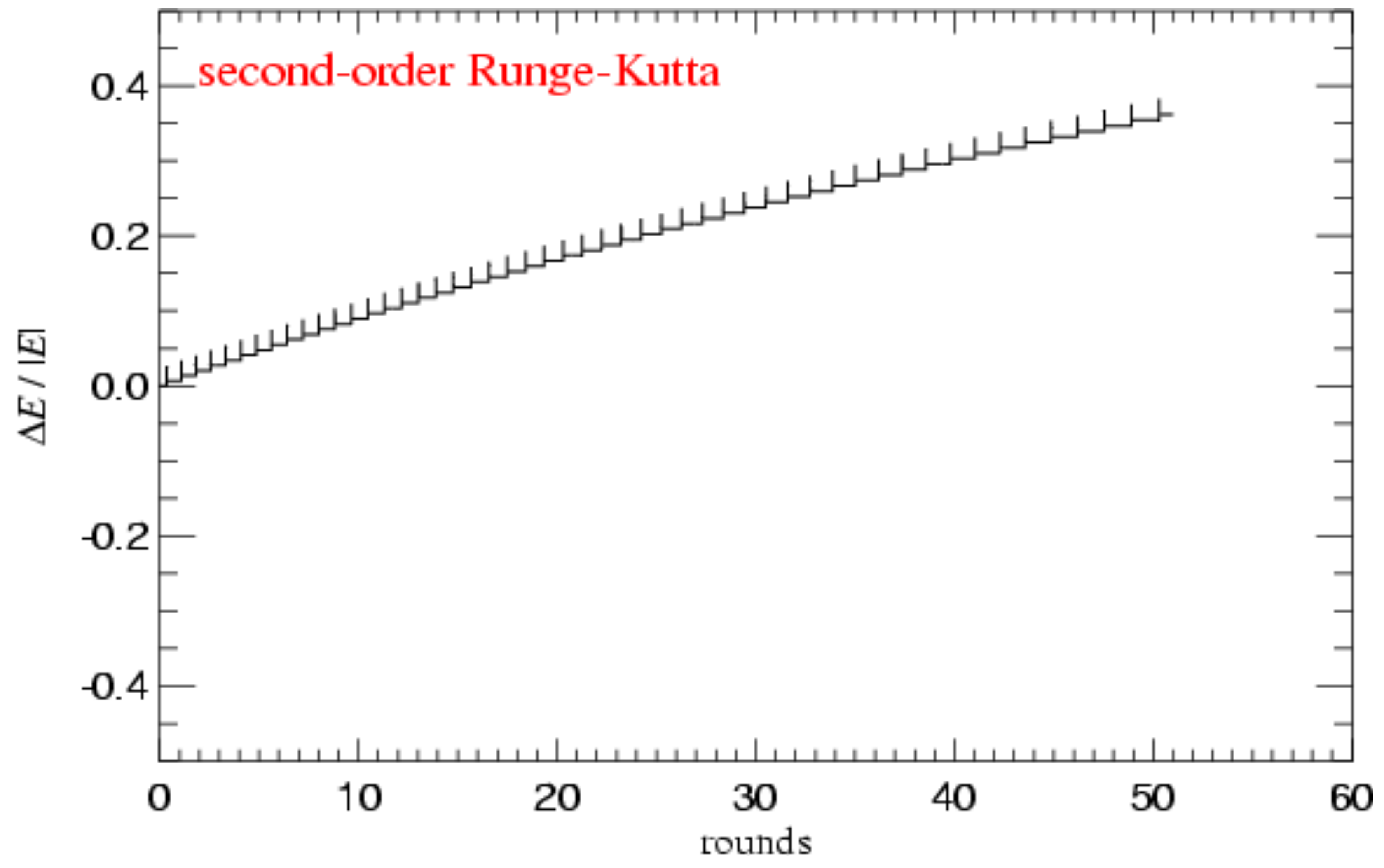
"Kick-Drift-Kick" version

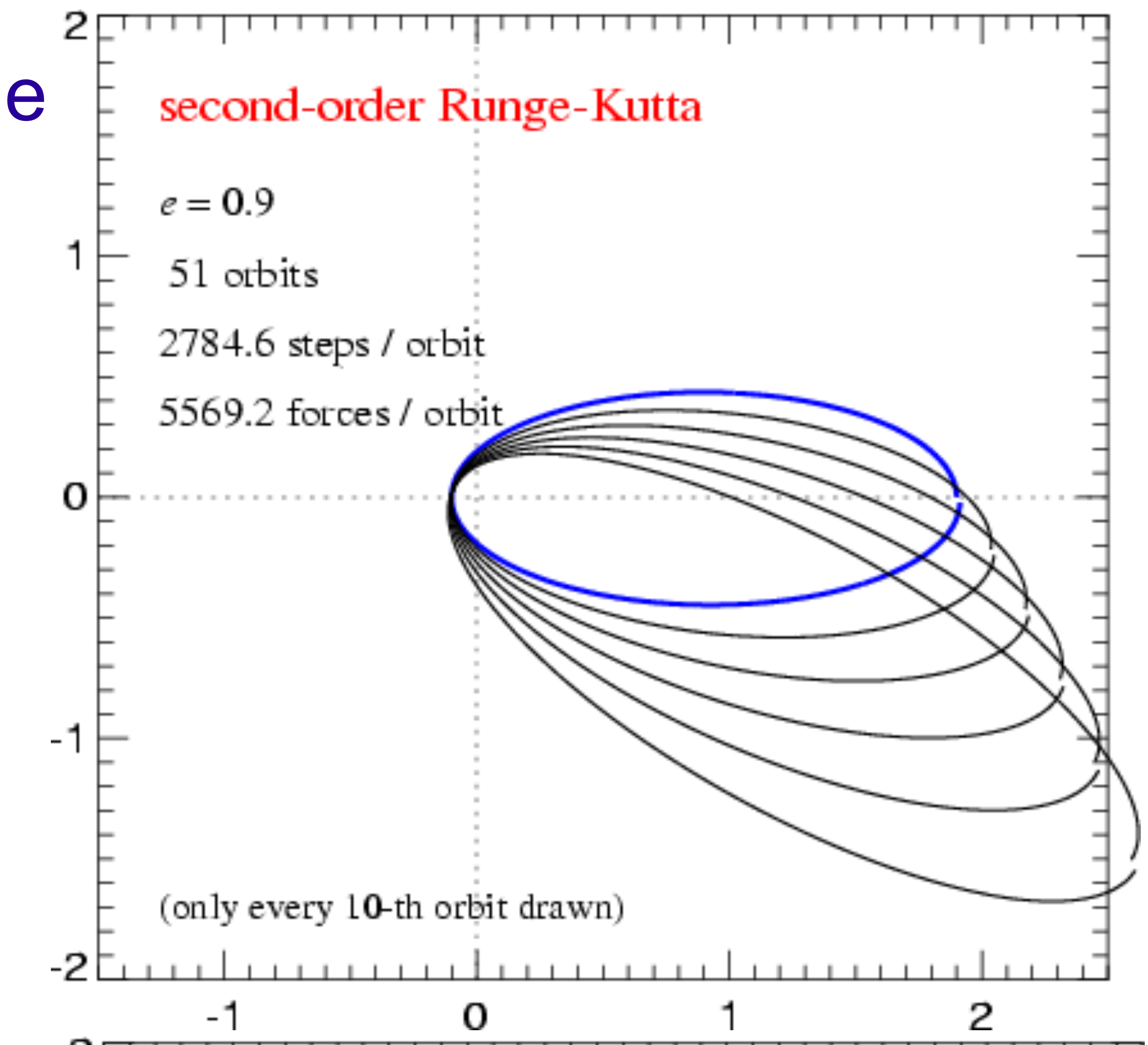
$$
\begin{aligned}
v_{n+\frac{1}{2}} &= v_n + f(x_n) \frac{\Delta t}{2} \\
x_{n+1} &= x_n + v_{n+\frac{1}{2}} \frac{\Delta t}{2} \\
v_{n+1} &= v_{n+\frac{1}{2}} + f(x_{n+1}) \frac{\Delta t}{2}
\end{aligned}
$$

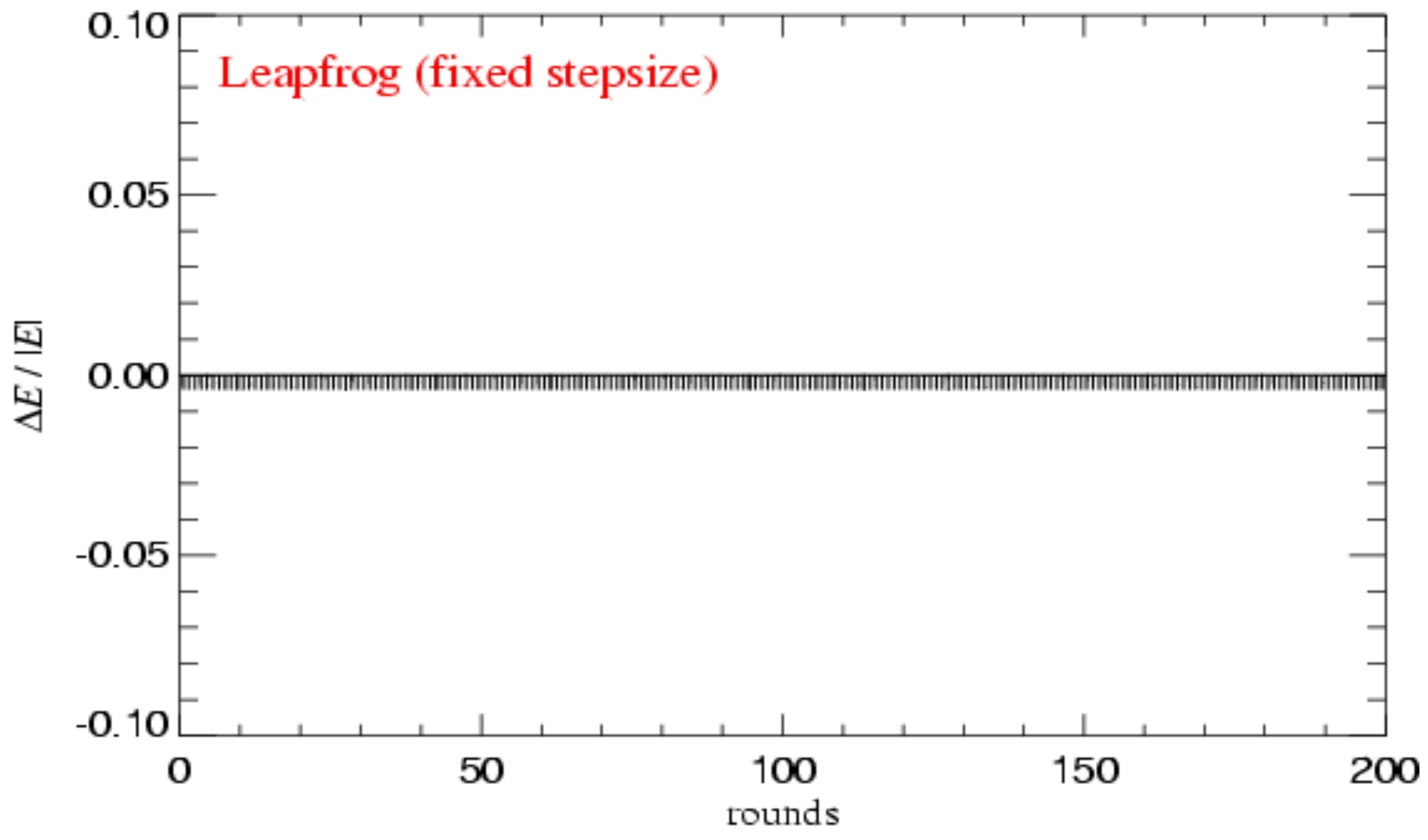
- **2nd order accurate**
- **symplectic**
- can be rewritten into time-centred formulation

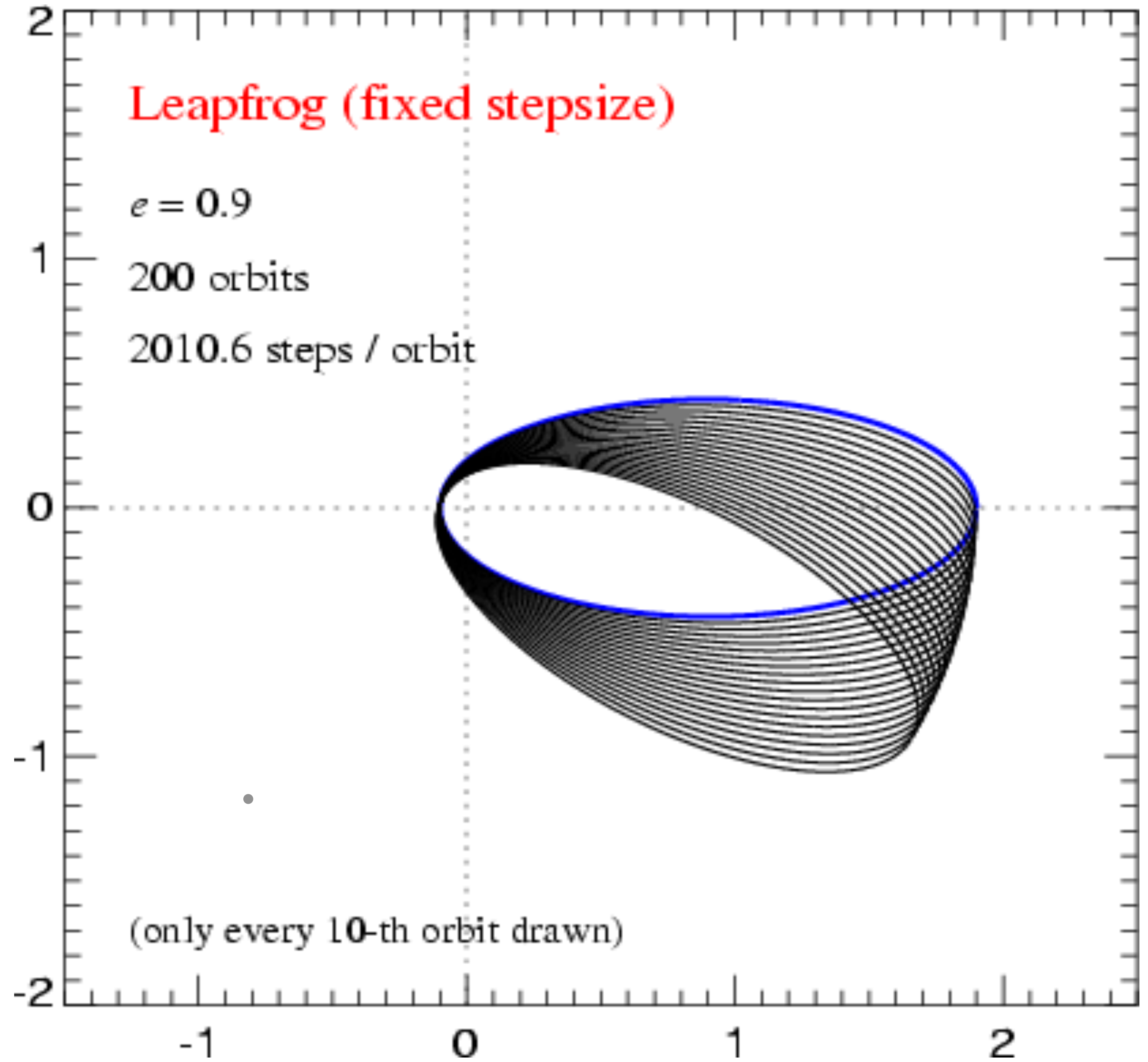
When compared with an integrator of the same order, the leapfrog is highly superior

INTEGRATING THE KEPLER PROBLEM



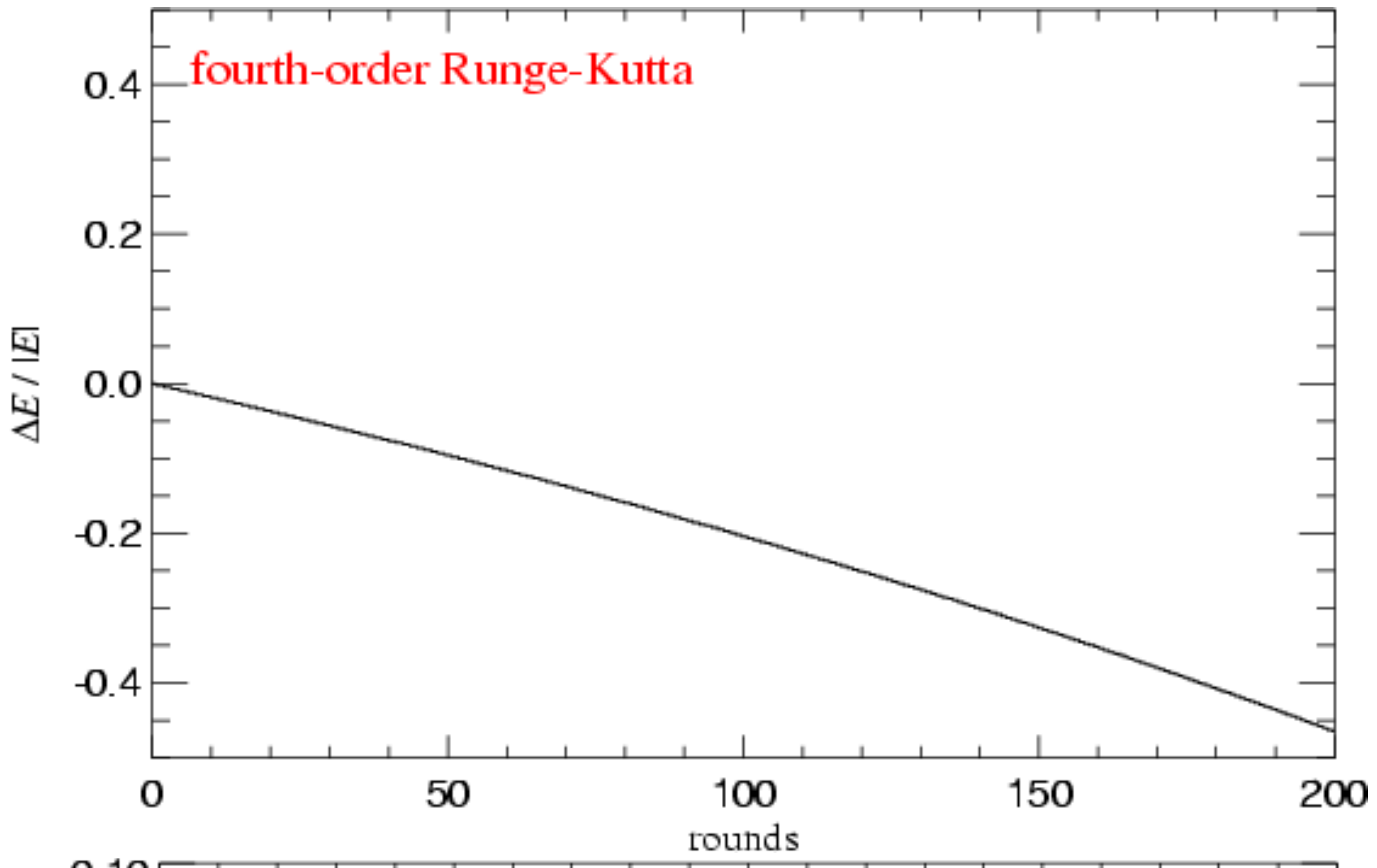


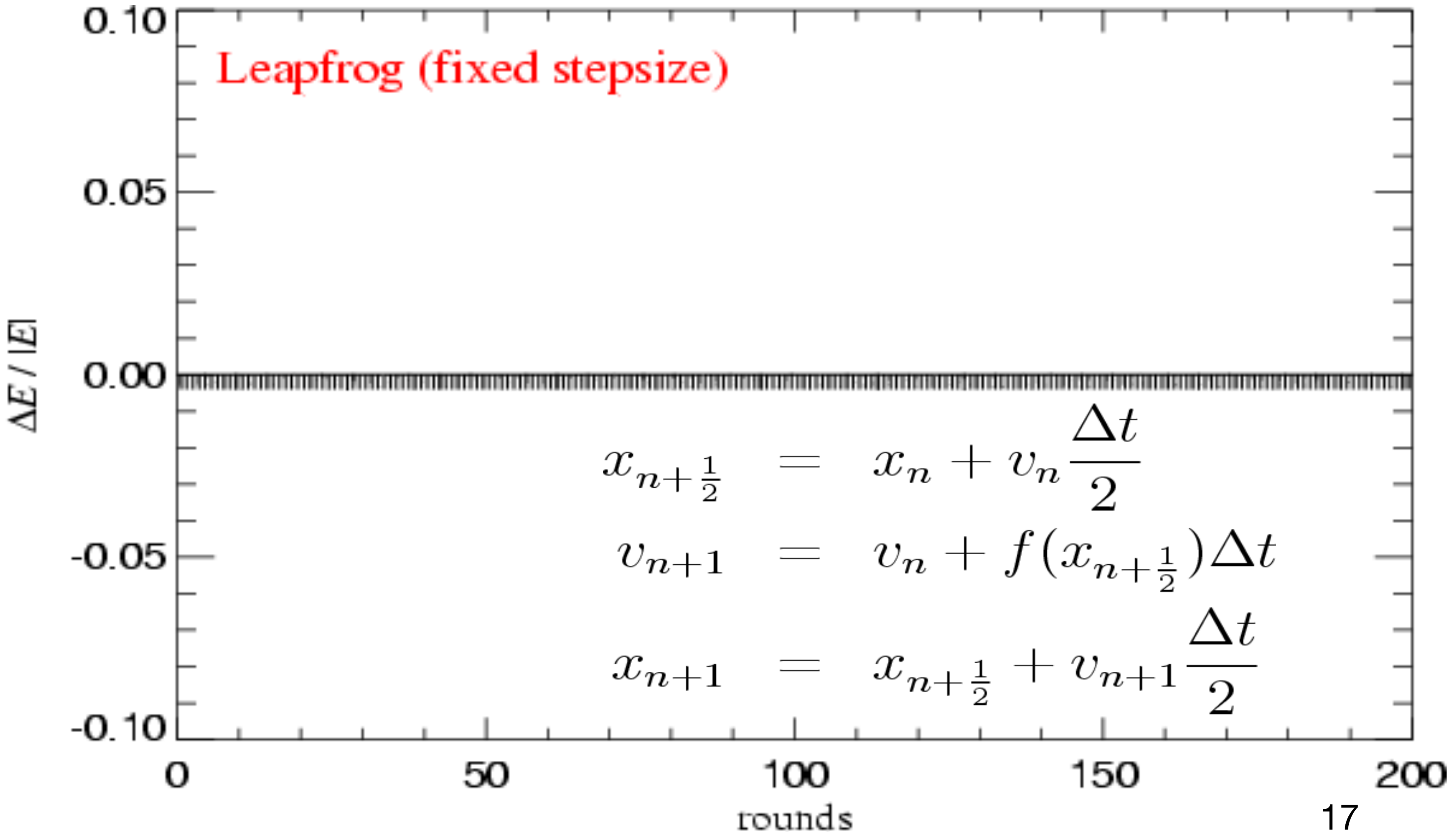




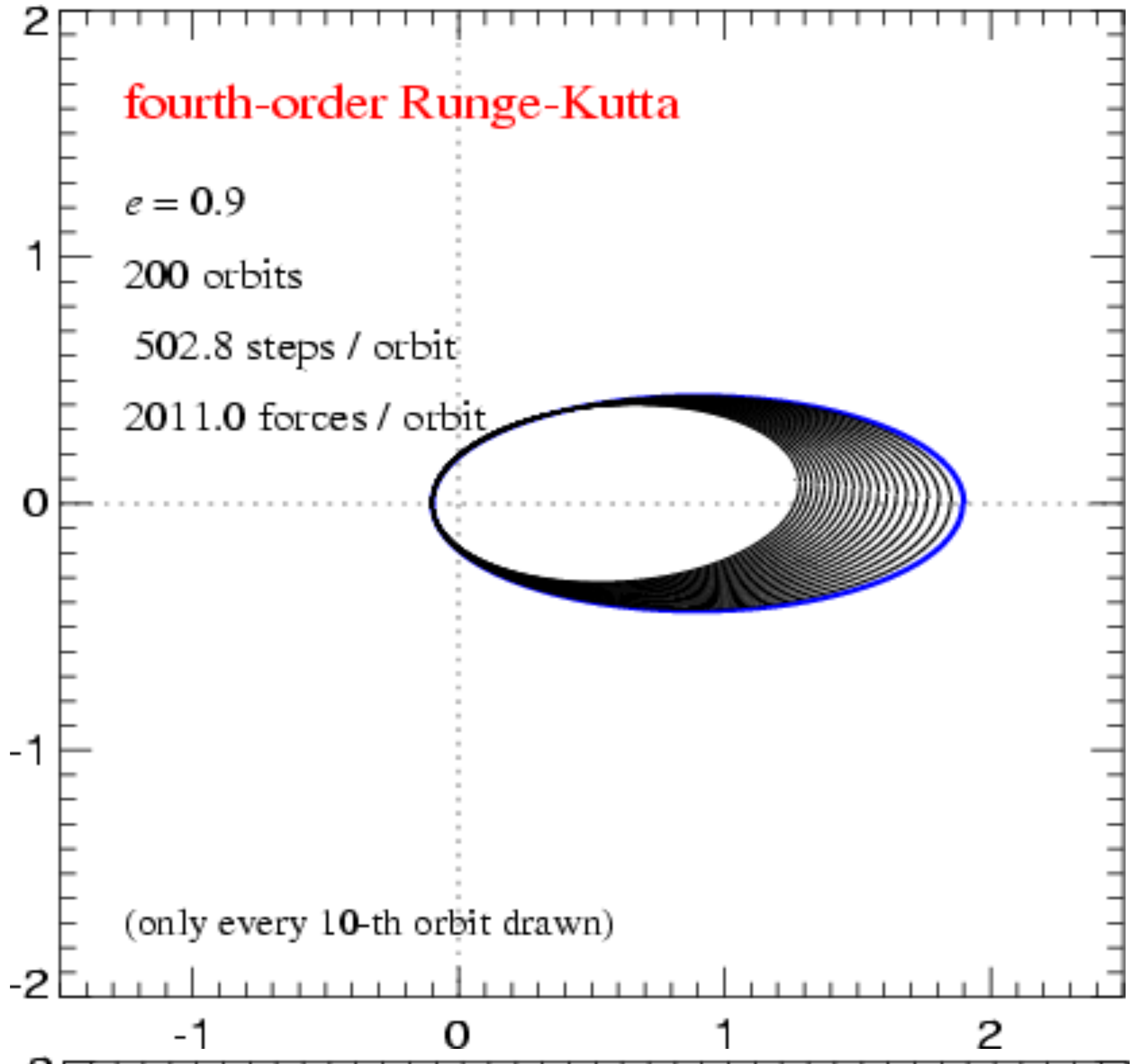
The leapfrog is behaving much better than one might expect...

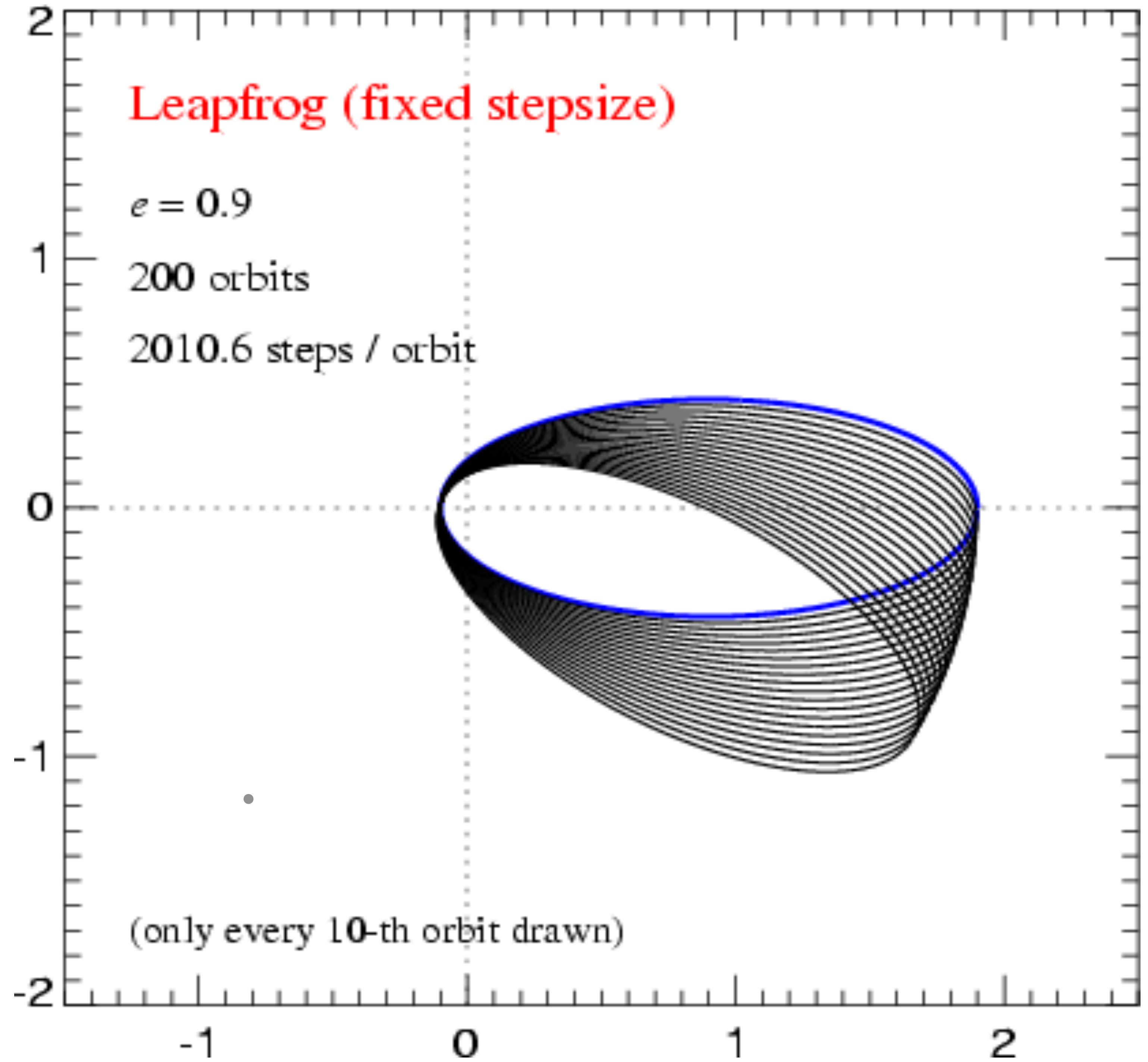
INTEGRATING THE KEPLER PROBLEM





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





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