Numerical solutions of classical equations of motion

Newton's laws govern the dynamics of

- Solar systems, galaxies,...
- Molecules in liquids, gases; often good approximation
  - quantum mechanics gives potentials
  - large (and even rather small) molecules move almost classically if the density is not too high
- "Everything that moves"

Almost no "real" systems can be solved analytically

- **Numerical integration of equations of motion**

One-dimensional motion

➢ A single "point particle" at x(t)

Equation of motion

$$\ddot{x}(t) = \frac{1}{m} F[x(t), \dot{x}(t), t]$$

Notation: velocity: $v(t) = \dot{x}(t)$

acceleration: $a(t) = \ddot{x}(t)$

Forces from: potential, damping (friction), driving (can be mix)

Rewrite second-order diff. eqv. as coupled first-order:

$$\dot{x}(t) = v(t)$$

$$\dot{v}(t) = a[x(t), v(t), t)]$$

Discretized time axis

$$t = t_0, t_1, \ldots, t_N, \quad t_{n+1} - t_n = \Delta_t$$

Start from given initial conditions: $v = v_0, \; x = x_0$

Simplest integration method: **Euler forward algorithm**

$$x_{n+1} = x_n + \Delta_t v_n$$
$$v_{n+1} = v_n + \Delta_t a_n$$

Step error: $O(\Delta_t^2)$

Fortran 90 implementations:

```
do i=1,nt
   t0=dt*(i-1)
   x1=x0+dt*v0
   v1=v0+dt*acc(x0,v0,t0)
   x0=x1; v0=v1
enddo
```

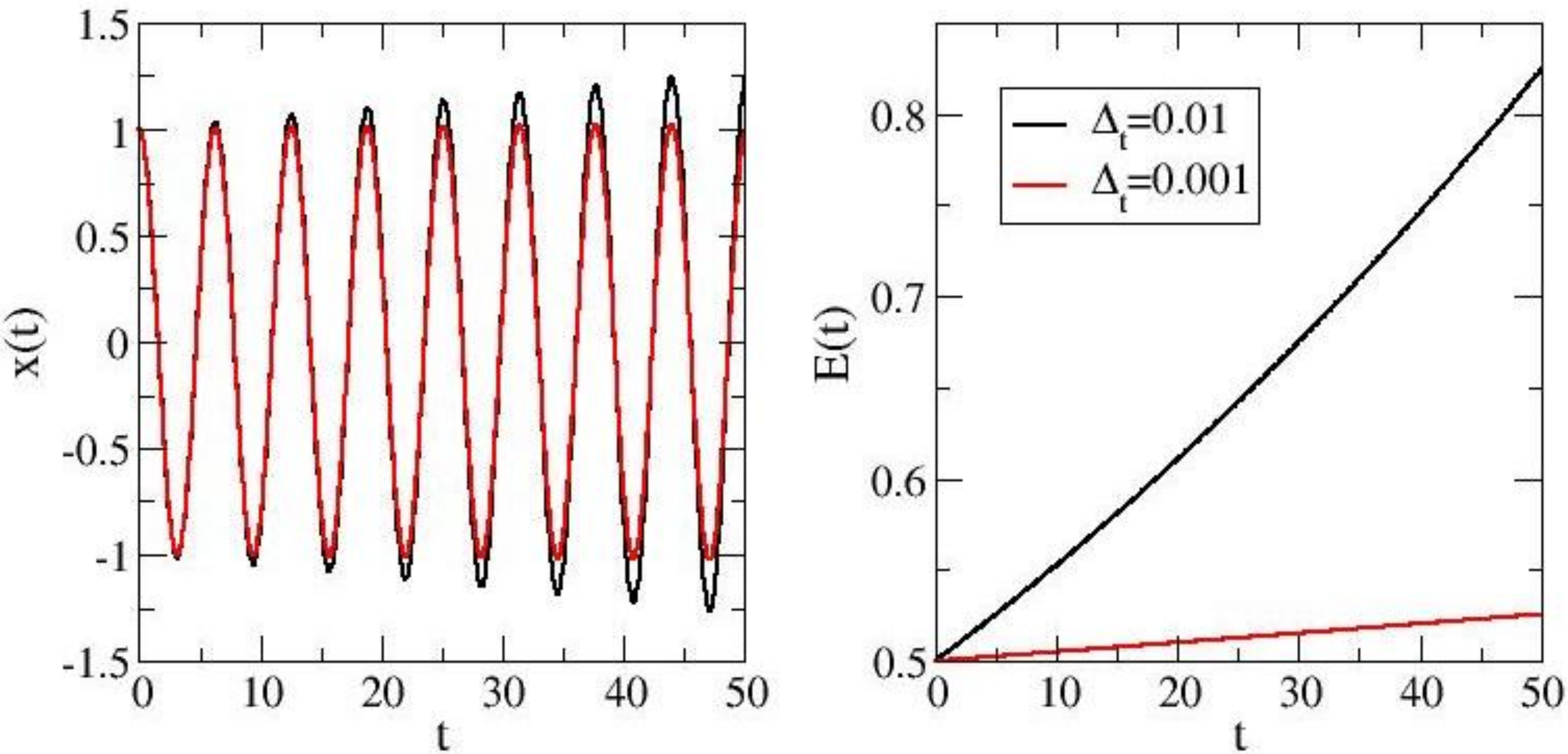
```
do i=1,nt
   t=dt*(i-1)
   a=acc(x,v,t)
   x=x+dt*v
   v=v+dt*a
enddo
```

- Euler is not a very good algorithm in practice
- Energy error unbounded (can diverge)
- Algorithms with better precision almost as simple

Illustration of Euler algorithm: Harmonic oscillator

$$E = \frac{1}{2}kx^2 + \frac{1}{2}mv^2 \qquad (F = -kx)$$

Integrated equations of motion for k=m=1; $\Delta_t = 0.01,\ 0.001$



Leapfrog algorithm (no damping)

Taylor expand x(t) to second order in time step

$$x(t_n + \Delta_t) = x(t_n) + \Delta_t v(t_n) + \frac{1}{2}\Delta_t^2 a(x_n, t_n) + O(\Delta_t^3).$$

Contains velocity at "half step": $v(t_n) + \frac{1}{2}\Delta_t a(x_n, t_n) = v(t_{n+1/2})$

Substituting this gives

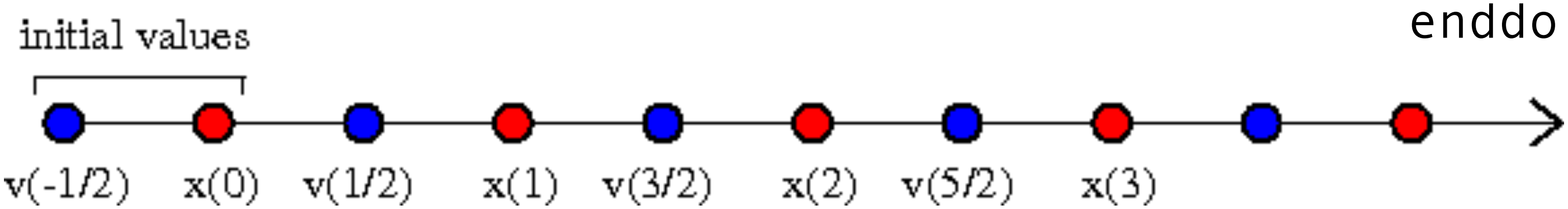
$$x(t_n + \Delta_t) = x(t_n) + \Delta_t v(t_n + \Delta_t/2) + O(\Delta_t^3)$$

Use similar form for v propagation: Leapfrog algorithm

$$v_{n+1/2} = v_{n-1/2} + \Delta_t a_n$$

$$x_{n+1} = x_n + \Delta_t v_{n+1/2}$$

```
do  i=1,nt
    t=dt*(i-1)
    a=acc(x,t)
    v=v+dt*a
    x=x+dt*v
enddo
```



Starting velocity from: $v_{-1/2} = v_0 - a_0\Delta_t/2$

What is the step error in the leapfrog algorithm?

- Might expect: $O(\Delta_t^3)$
- Actually: $O(\Delta_t^4)$
- Can be easily seen in a different derivation

The Verlet algorithm

Start from two Taylor expansions: $x(t_n \pm \Delta_t)$

$$x_{n+1} = x_n + \Delta_t v_n + \frac{1}{2}\Delta_t^2 a_n + \frac{1}{6}\Delta_t^3 \dot{a}_n + O(\Delta_t^4)$$

$$x_{n-1} = x_n - \Delta_t v_n + \frac{1}{2}\Delta_t^2 a_n - \frac{1}{6}\Delta_t^3 \dot{a}_n + O(\Delta_t^4)$$

Adding these gives the so-called Verlet algorithm

$$x_{n+1} = 2x_n - x_{n-1} + \Delta_t^2 a_n + O(\Delta_t^4)$$

Velocity defined by: $v_{n-1/2} = (x_n - x_{n-1})/\Delta_t$

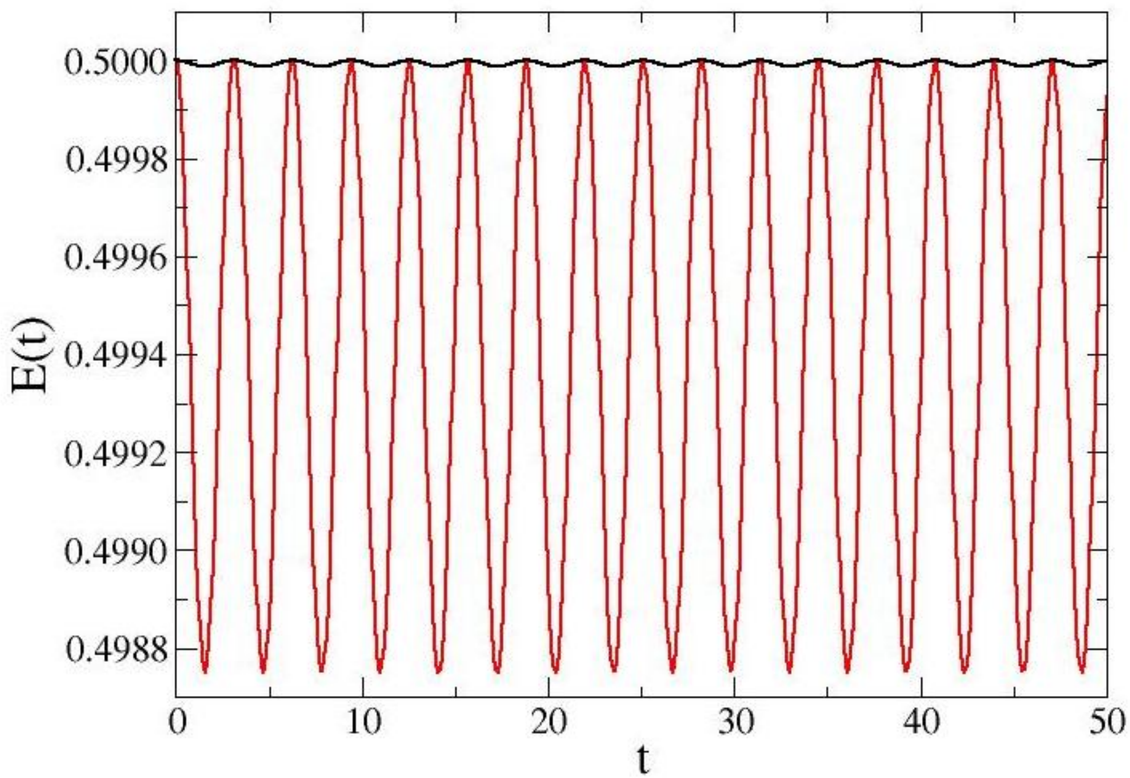
$$x_{n+1} = x_n + \Delta_t(v_{n-1/2} + \Delta_t a_n) + O(\Delta_t^4)$$

Same as leapfrog, since $v_{n-1/2} + \Delta_t a_n = v_{n+1/2}$

Properties of Verlet/leapfrog algorithm

- Time reversal symmetry (check for round-off errors)
- Errors bounded for periodic motion (time-reversal)
- High accuracy with little computational effort

Illustration: Harmonic oscillator (k=m=1), $\Delta_t = 0.1, 0.01$



Code almost identical to Euler (swicth 2lines!)

```
do i=1,nt
    t=dt*(i-1)
    a=acc(x,t)
    v=v+dt*a
    x=x+dt*v
enddo
```

Remember, initialize v at the half-step -dt/2!

Two equivalent Verlet/leapfrog methods

Verlet:

$$x_{n+1} = 2x_n - x_{n-1} + \Delta_t^2 a_n + O(\Delta_t^4)$$

Leapfrog:

$$v_{n+1/2} = v_{n-1/2} + \Delta_t a_n$$

$$x_{n+1} = x_n + \Delta_t v_{n+1/2}$$

Error build-up in Verlet/leapfrog method

Error in x after N steps, time $T = t_N - t_0 = N\Delta_t$

Difference between numerical and exact solution: $x_n = x_n^{\text{ex}} + \delta_n$

Inserting this in Verlet equation

$$x_{n+1} = 2x_n - x_{n-1} + \Delta_t^2 a_n + O(\Delta_t^4)$$

gives

$$\delta_{n-1} - 2\delta_n + \delta_{n+1} = -(x_{n-1}^{\text{ex}} - 2x_n^{\text{ex}} + x_{n+1}^{\text{ex}}) + \Delta_t^2 a_n + O(\Delta_t^4).$$

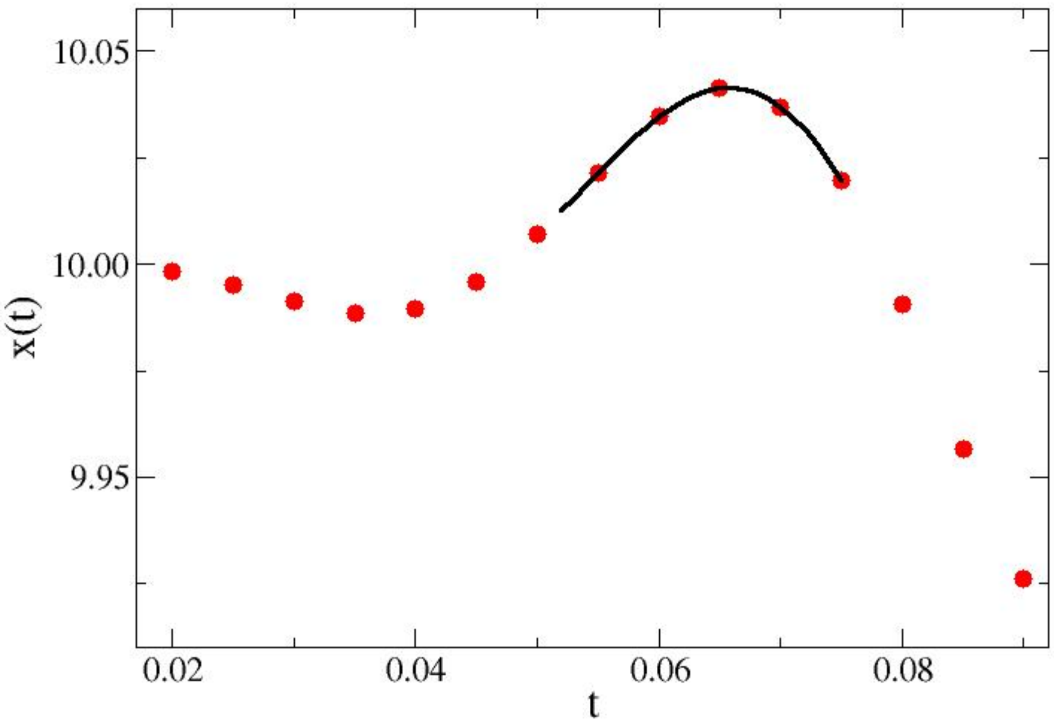
Discretized second derivative:

$$\frac{d^2 f(t_n)}{dt^2} = \frac{d}{dt}\frac{1}{\Delta_t}[f(t_{n+1/2}) - f(t_{n-1/2})] = \frac{1}{\Delta_t^2}(f_{n-1} - 2f_n + f_{n+1})$$

The equation of motion for the error is thus:

$$\ddot{\delta}(t_n) = -\ddot{x}^{\text{ex}}(t_n) + a(t_n) + O(\Delta_t^2)$$

$$\ddot{\delta}(t_n) = -\ddot{x}^{\text{ex}}(t_n) + a(t_n) + O(\Delta_t^2)$$



Assume smoothness on scale of time-step, use continuum derivative (imagine a high-order interpolating polynomial between points)

Exact solution satisfies: $\ddot{x}^{\text{ex}}(t_n) = a(t_n)$

We are thus left with: $\ddot{\delta}(t_n) \sim \Delta_t^2$

Integrate to obtain error after time T:

Worst case: no error cancellations (same sign for all n):

$$\delta(T) = \int_0^T dt \dot{\delta}(t) = \int_0^T dt \int_0^t dt' \ddot{\delta}(t') \sim T^2 \Delta_t^2$$

Verlet/leapfrog methods for damped systems

We assumed velocity-independent forces in leapfrog method;

$$v_{n+1/2} = v_{n-1/2} + \Delta_t a_n$$
$$x_{n+1} = x_n + \Delta_t v_{n+1/2}$$

With velocity dependent $a(x,v,t) = F(x,v,x)/m$

we need v_n but have only $v_{n+1/2}$

To study this problem, separate damping from rest of force

$$a(x,v,t) = \frac{1}{m}[F(x,t) - G(v)]$$

Consider approximation: $a(x_n, v_n, t_n) \approx [F(x_n, t_n) - G(v_{n-1/2})]/m$

$$\hat{v}_{n+1/2} = v_{n-1/2} + \Delta_t[F(x_n, t_n) - G(v_{n-1/2})]/m$$
$$\hat{x}_{n+1} = x_n + \Delta_t \hat{v}_{n+1/2}$$

The error made in a is $\sim \Delta_t$ which gives x-error $\sim \Delta_t^3$

We can do a second step using $v_n = (\hat{x}_{n+1} - x_{n-1})/(2\Delta_t)$

This renders the error in x $\sim O(\Delta_t^4)$

Summary; leapfrog algorithm with damping:

$$\hat{v}_{n+1/2} = v_{n-1/2} + \Delta_t[F(x_n, t_n) - G(v_{n-1/2})]/m$$

$$\hat{x}_{n+1} = x_n + \Delta_t \hat{v}_{n+1/2}$$

$$v_n = (\hat{x}_{n+1} - x_{n-1})/(2\Delta_t)$$

$$v_{n+1/2} = v_{n-1/2} + \Delta_t a_n$$

v_n used here in a_n

$$x_{n+1} = x_n + \Delta_t v_{n+1/2}$$

Requires more work than standard leapfrog:

$$v_{n+1/2} = v_{n-1/2} + \Delta_t a_n$$

$$x_{n+1} = x_n + \Delta_t v_{n+1/2}$$

Runge-Kutta method

Classic high-order scheme; error $O(\Delta_t^5)$ (4th order)

Consider first single first-order equation: $\dot{x}(t) = f[x(t), t]$

Warm-up: 2nd order Runge-Kutta

Use mid-point rule:

$$\int_{t_n}^{t_{n+1}} f[x(t), t]dt = \Delta_t f[x(t_{n+1/2}), t_{n+1/2}] + O(\Delta_t^3)$$

But we don't know $x(t_{n+1/2}) = x_{n+1/2}$

Approximate it using Euler formula;

$$\hat{x}_{n+1/2} = x_n + \frac{\Delta_t}{2} f(x_n, t_n) + O(\Delta_t^2)$$

Sufficient accuracy for $O(\Delta_t^3)$ formula:

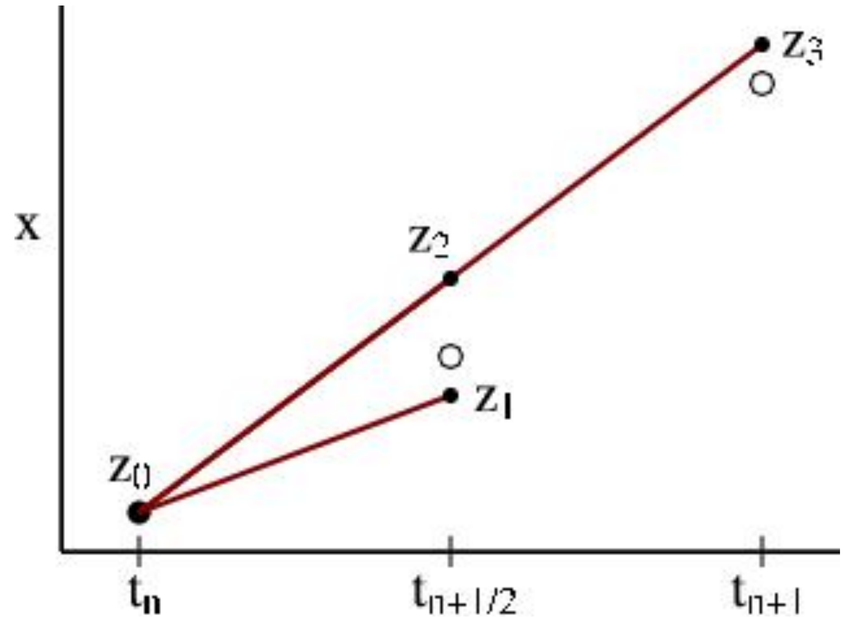
$$x_{n+1} = x_n + \Delta_t f(\hat{x}_{n+1/2}, t_{n+1/2}) + O(\Delta_t^3)$$

4th-order Runga-Kutta (the real thing)

Uses Simpson's formula: $x_{n+1} = x_n + \frac{\Delta_t}{6}(f_n + 4f_{n+1/2} + f_{n+1})$

Need to find $O(\Delta_t^4)$ approximations for $f_{n+1/2}, f_{n+1}$

Somewhat obscure scheme accomplishes this (can be proven correct using Taylor expansion)



$$\hat{x}_{n+1/2} = x_n + \Delta_t f(x_n, t_n)/2$$

$$\hat{x}'_{n+1/2} = x_n + \Delta_t f(\hat{x}_{n+1/2}, t_{n+1/2})/2$$

$$k_1 = \Delta_t f(x_n, t_n)$$
$$k_2 = \Delta_t f(x_n + k_1/2, t_{n+1/2}),$$
$$k_3 = \Delta_t f(x_n + k_2/2, t_{n+1/2})$$
$$k_4 = \Delta_t f(x_n + k_3, t_{n+1})$$

$$x_{n+1} = x_n + \frac{1}{6}(k1 + 2k_2 + 2k_3 + k_4)$$

Runge-Kutta for two coupled equations

$$\dot{x}(t) = f(x,y,t) \quad \dot{y}(t) = g(x,y,t)$$

$$
\begin{aligned}
k_1 &= \Delta_t f(x_n, y_n, t_n), \\
l_1 &= \Delta_t g(x_n, y_n, t_n), \\
k_2 &= \Delta_t f(x_n + k_1/2, y_n + l_1/2, t_{n+1/2}), \\
j_2 &= \Delta_t g(x_n + k_1/2, y_n + l_1/2, t_{n+1/2}), \\
k_3 &= \Delta_t f(x_n + k_2/2, y_n + l_2/2, t_{n+1/2}), \\
l_3 &= \Delta_t g(x_n + k_2/2, y_n + l_2/2, t_{n+1/2}), \\
k_4 &= \Delta_t f(x_n + k_3, y_n + l_3, t_{n+1}), \\
l_4 &= \Delta_t g(x_n + k_3, y_n + l_3, t_{n+1}), \\
x_{n+1} &= x_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4), \\
y_{n+1} &= y_n + \frac{1}{6}(l_1 + 2l_2 + 2l_3 + l_4),
\end{aligned}
$$

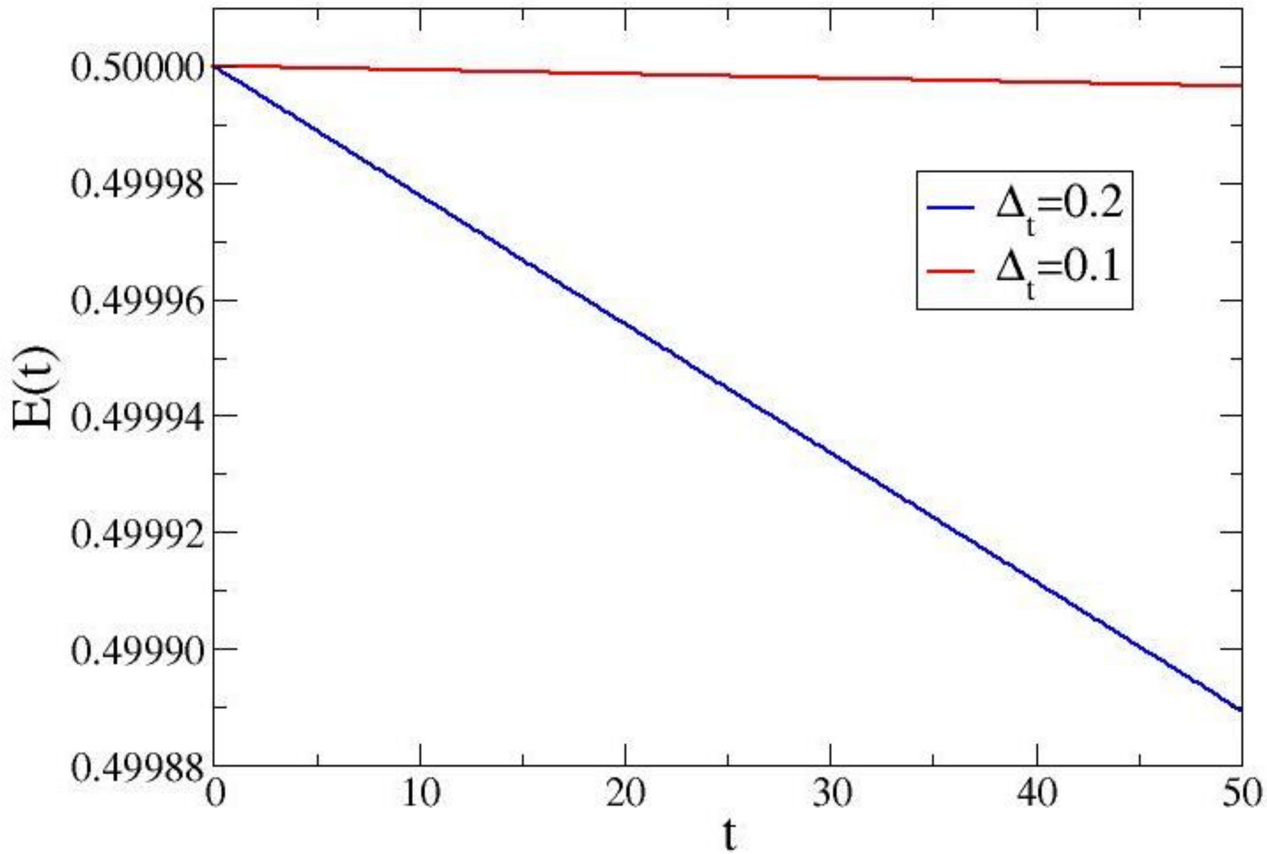
Equations of motion, Runge-Kutta algorithm

$$
\begin{aligned}
k_1 &= \Delta_t a(x_n, v_n, t_n), \\
l_1 &= \Delta_t v_n, \\
k_2 &= \Delta_t a(x_n + l_1/2, v_n + k_1/2, t_{n+1/2}), \\
l_2 &= \Delta_t (v_n + k_1/2), \\
k_3 &= \Delta_t a(x_n + l_2/2, v_n + k_2/2, t_{n+1/2}), \\
l_3 &= \Delta_t (v_n + k_2/2), \\
k_4 &= \Delta_t a(x_n + l_3, v_n + k_3, t_{n+1}), \\
l_4 &= \Delta_t (v_n + k_3), \\
v_{n+1} &= y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4), \\
x_{n+1} &= x_n + \frac{1}{6}(l_1 + 2l_2 + 2l_3 + l_4).
\end{aligned}
$$

Including damping is no problem here

The RK method does not have time-reversal symmetry

- Errors not bounded for periodic motion
- Time-reversibility important in some applications



harmonic oscillator (k=m=1)

Advantages of RK relative to leapfrog:

- Variable time-step can be used (uses only n-data for n+1)
- Better error scaling (but more computations for each step)

What algorithm to use?

Recommendation

In the case of energy-conserving systems
(no damping or external driving forces)

- **Use the Verlet/leapfrog algorithm**
  - **good energy-conserving property (no divergence)**

In the case of non-energy-conserving systems
(including damping and/or external driving forces)

- **Energy is not conserved, so no advantage for Verlet**
- **Use the Runge-Kutta algorithm**
  - **smaller discretization error for given integration time T**