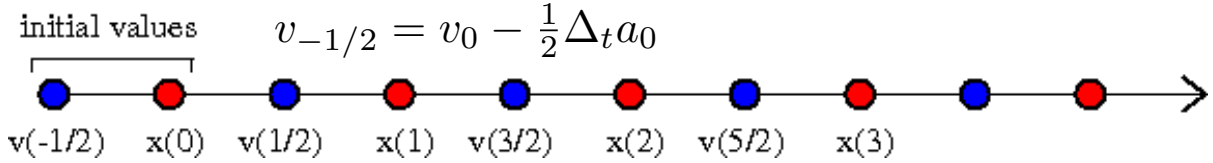
$$v_{n+1/2} = v_{n-1/2} + \Delta_t a_n$$

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




**What is the step error here?**

**- we expanded x to 2nd order. Is the step error 3rd order?**

**Let's do a different derivation to cubic order**

**- forward and backward x steps:**

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

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

add these →

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

**consider** $x_n - x_{n-1} = x(t_{n-1/2} + \Delta_t/2) - x(t_{n-1/2} - \Delta_t/2)$

$$= [x_{n-1/2} + (\Delta_t/2)v_{n-1/2} + \tfrac{1}{2}(\Delta_t/2)^2 a_{n-1/2} + \tfrac{1}{6}(\Delta_t/2)^3 \dot{a}_{n-1/2} + \ldots]$$
$$- [x_{n-1/2} - (\Delta_t/2)v_{n-1/2} + \tfrac{1}{2}(\Delta_t/2)^2 a_{n-1/2} - \tfrac{1}{6}(\Delta_t/2)^3 \dot{a}_{n-1/2} + \ldots]$$

$$= \Delta_t v_{n-1/2} + O(\Delta_t^3) \quad \rightarrow \quad x_n - x_{n-1} = v_{n-1/2}\Delta_t + O(\Delta_t^3)$$

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

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

$$x_n - x_{n-1} = v_{n-1/2}\Delta_t + O(\Delta_t^3) \quad \rightarrow \quad v_{n-1/2} = \frac{1}{\Delta_t}(x_n - x_{n-1}) + O(\Delta_t^2)$$

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

**Here it looks like the error in $x_{n+1}$ should be $O(\Delta_t^3)$**

**However, earlier we saw that**

$$v_{n+1/2} = v_{n-1/2} + a_n\Delta_t + O(\Delta_t^3)$$

$$v_{n+1/2} = v_n + (\Delta_t/2)a_n + (\Delta_t/2)^2\dot{a}_n + O(\Delta_t^3)$$

$$v_{n-1/2} = v_n - (\Delta_t/2)a_n + (\Delta_t/2)^2\dot{a}_n - O(\Delta_t^3)$$

**and so the error in x is actually $O(\Delta_t^4)$**

**We arrive at an algorithm identical to the Leapfrog algorithm**

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

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

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

**The Leapfrog step error is actually $O(\Delta_t^4)$**

**The form without explicit v is called the Verlet algorithm**

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

**Initial conditions: $x_0$, $x_1$**

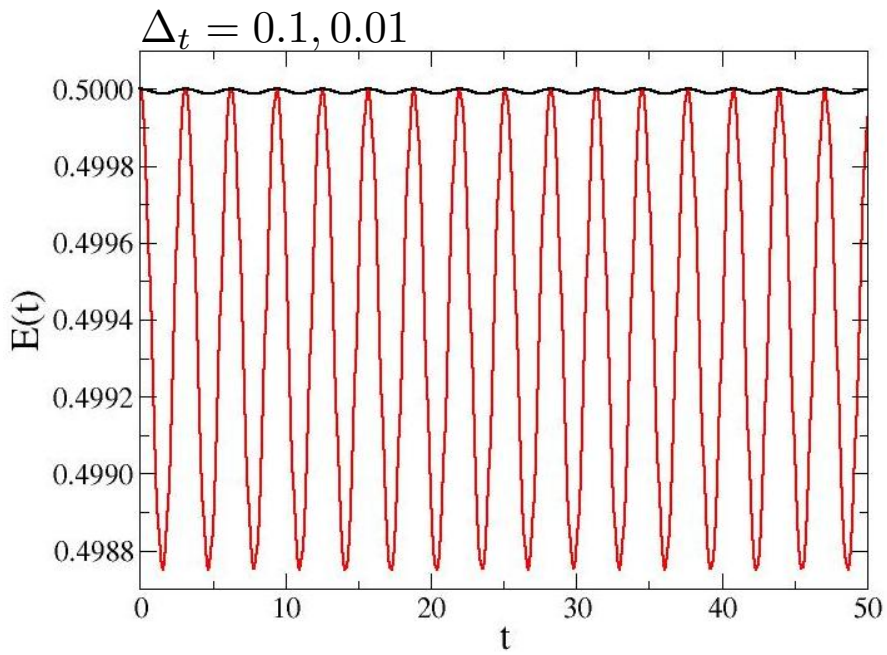
**If needed, the velocity obtained as** $$v_{n+1/2} = \frac{1}{\Delta_t}(x_{n+1} - x_n) + O(\Delta_t^2)$$

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

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

**Julia Leapfrog implementation**

```
for i=1:nt
    t=dt*(i-1)
    a=acc(x,t)
    v=v+dt*a
    x=x+dt*v
end
```

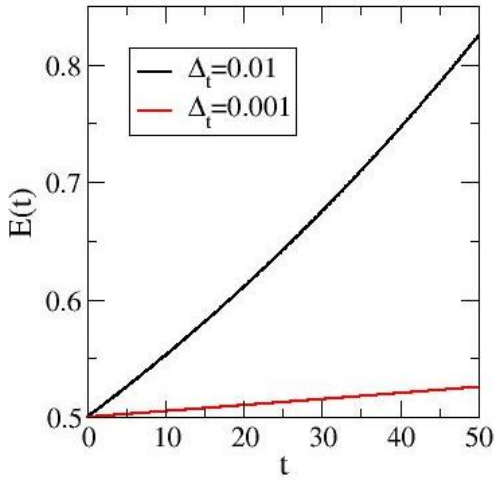



**Properties of the method**

- time-reversal symmetric
- errors bounded for periodic motion
- small step error at low effort

**Test:** same oscillator as before

```
for i=1:nt
    t=dt*(i-1)
    a=acc(x,v,t)
    x=x+dt*v
    v=v+dt*a
end
```




**Note: Implementation almost identical to Euler**
**- just swap two lines!**

## Accumulated errors in the Leapfrog/Verlet algorithm

The number of time steps N can be large; N=T/$\Delta_t$

How does the error at time T depend on T and $\Delta_t$?

Write the position at time step n as the exact value plus a deviation

$$x_n = x_n^{\mathrm{ex}} + \delta_n$$

and use in the Verlet algorithm

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

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

Here we see discrete versions of second derivatives

Recall: For any function f(t) defined on the time grid:

$$f_{n+1} = f_n + \Delta_t \dot{f}_n + \tfrac{1}{2}\Delta_t^2 \ddot{f}_n + \tfrac{1}{6}\Delta_t^3 \dddot{f}_n + O(\Delta_t^4)$$

$$f_{n-1} = f_n - \Delta_t \dot{f}_n + \tfrac{1}{2}\Delta_t^2 \ddot{f}_n - \tfrac{1}{6}\Delta_t^3 \dddot{f}_n + O(\Delta_t^4)$$

add these:

$$f_{n+1} - 2f_n + f_{n-1} = \Delta_t^2 \ddot{f}_n + O(\Delta_t^4)$$

$$(x^{\rm ex}_{n+1} - 2x^{\rm ex}_n + x^{\rm ex}_{n-1})/\Delta_t^2 = -(\delta_{n+1} - 2\delta_n + \delta_{n-1})/\Delta_t^2 + a_n + O(\Delta_t^2)$$

$$(f_{n+1} - 2f_n + f_{n-1})/\Delta_t^2 = \ddot{f}_n + O(\Delta_t^2)$$

**Replace discrete time derivatives by continuum versions:**

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

**The exact solution satisfies**

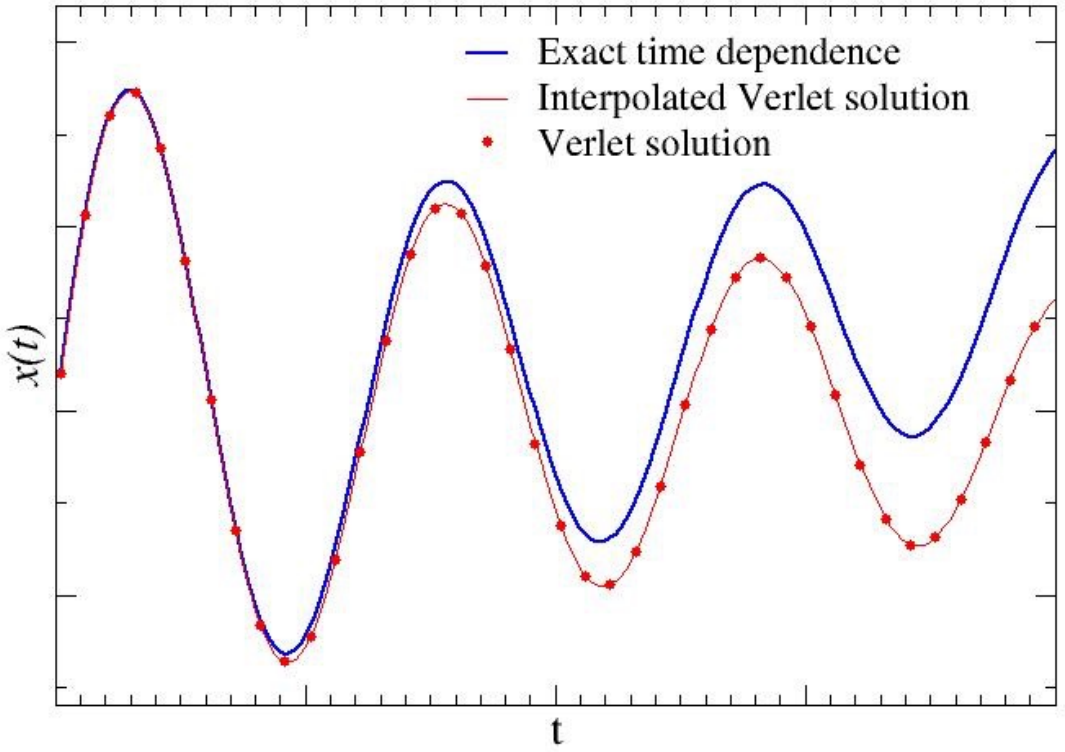
$$\ddot{x}^{\rm ex}(t) = a(t)$$

**and we are left with**

$$\ddot{\delta}(t) = O(\Delta_t^2)$$

**Why is it OK to get rid of the discretization here?**

**We can always construct a smooth interpolation between the discrete values $x_n$**
**- e.g., a high-order polynomial**



The equation for the error is rather incomplete

$$\ddot{\delta}(t) = O(\Delta_t^2)$$

We can also write it as an actual equation

$$\ddot{\delta}(t) = \Delta_t^2 g(t)$$

though we do not know anything about the function g(t)

For the accumulated error at time t=T we have to integrate

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

We cannot go any further without making assumptions for g(t)

- g(t) could oscillate around 0, leading to cancelations and small errors
  - this is the case for periodic motion
- g(t) could increase rapidly as t increases, causing large errors
  - the solution then likely goes completely bad after some time
- g(t) could be roughly time independent, leading to: $\delta(T) = O(\Delta_t^2 T^2)$
  - may be common for “well behaved” regular, non-periodic motion

Note: $\Delta_t$ must be sufficiently small for the above arguments to be valid

## Leapfrog/Verlet method including damping

We assumed velocity-independent force (acceleration) in

$$v_{n+1/2} = v_{n-1/2} + \Delta_t a_n$$ we do not have $v_n$ for $a_n = a(x_n, v_n, t)$

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

We can still use this form, with $a_n \rightarrow a(x_n, v_{n-1/2}, t)$, where error is $O(\Delta_t)$

- x error is then $O(\Delta_t^3)$ instead of $O(\Delta_t^4)$ [see by expanding a(v) in v]

To do better, first separate out dissipative part of force:

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

Consider the approximation

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

and use this for intermediate (^) velocity and position:

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

$$\hat{x}_{n+1} = x_n + \Delta_t \hat{v}_{n+1/2}$$ has $O(\Delta_t^3)$ error

Then we can obtain $v_n$ with $O(\Delta_t^2)$ error: $v_n = (\hat{x}_{n+1} - x_{n-1})/(2\Delta_t)$

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

Now we can use this in the acceleration $a_n(x_n, v_n, t)$; $O(\Delta_t^2)$ error

Summary of procedure:

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

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

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

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

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

$v_n$ used here in $a_n$

More than twice as much work as the standard Leapfrog method

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

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

but the $O(\Delta_t^4)$ error is now maintained (work pays off)

Test by running friction.ipynb on the web site (tomorrow's discussion)

## Runge-Kutta (RK) Method

A classic method with very small step error; O($\Delta_t^5$)

Let's first apply it to a single 1st-order equation:

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

It's instructive to first look at a simpler method with O($\Delta_t^3$) error

2nd-order RK method

Apply the mid-point rule (recall from numerical integration)

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

But here we do not have $x(t_{n+1/2}) = x_{n+1/2}$

We can approximate it using the first-order form

$$\hat{x}_{n+1/2} = x_n + \frac{\Delta_t}{2} f(x_n, t_n)$$ ^ indicates an intermediate value of x. Error is $O(\Delta_t^2)$

- series expand f() in the integration formula to see that

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

Illustrates the use of intermediate values with larger error

## 4th-order RK method

Use Simpson's formula:

$$x_{n+1} = x_n + \int_{t_n}^{t_{n+1}} f[x(t), t]dt = x_n + \frac{\Delta_t}{6}(f_n + 4f_{n+1/2} + f_{n+1}) + O(\Delta_t^5)$$

where we need to find an approximation to $f_{n+1/2}$ and $f_{n+1}$ with $O(\Delta_t^4)$ errors

The way to do this is a bit involved/obscure…

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

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

similar for $x_{n+1}$. Scheme boils down to evaluating these:

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

and then

$$x_{n+1} = x_n + \frac{1}{6}(k1 + 2k_2 + 2k_3 + k_4)$$ with $O(\Delta_t^5)$ step error

## RK method for two coupled 1st-order equations

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

**Simple generalization of the previous case:**

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

$$\begin{aligned}
k_1 &= \Delta_t f(x_n, t_n) \\
k_2 &= \Delta_t f(x_n + k_1/2, t_{n+1/2}), \\
k_3 &= \Delta_t f(x_n + k_2/2, t_{n+1/2}) \\
k_4 &= \Delta_t f(x_n + k_3, t_{n+1}) \\
x_{n+1} &= x_n + \frac{1}{6}(k1 + 2k_2 + 2k_3 + k_4)
\end{aligned}$$

**This is more general than Newton's equations**

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

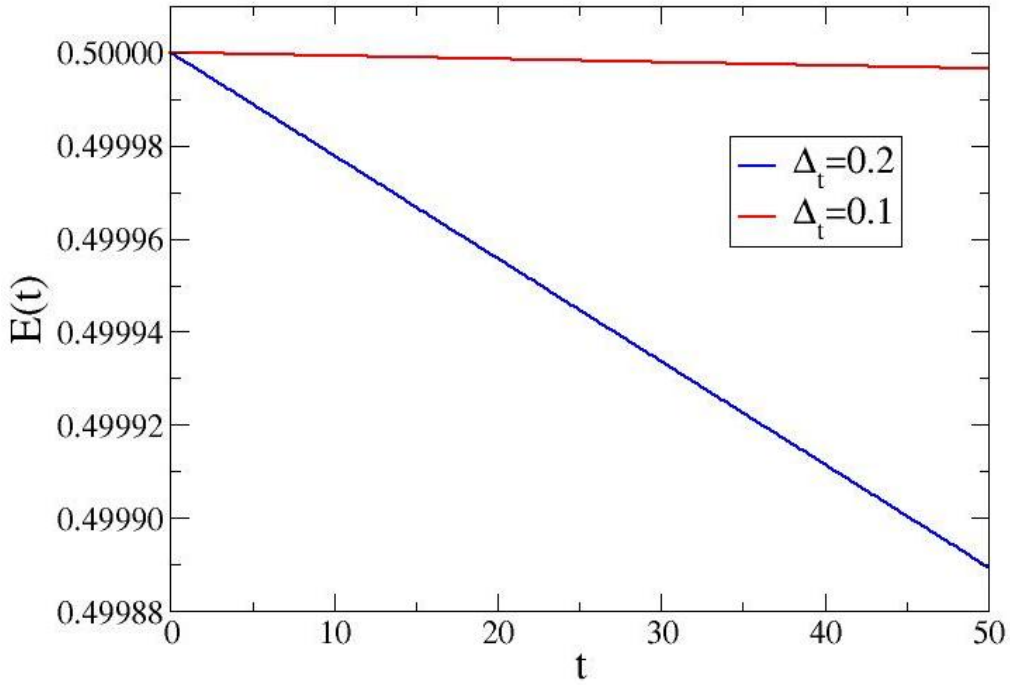
## RK formulas for equations of motion

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

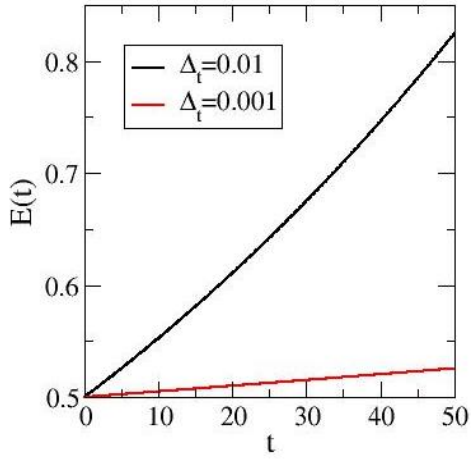
Friction (dissipative, v-dependent forces) can be included directly here

## Test of the 4th-order RK method

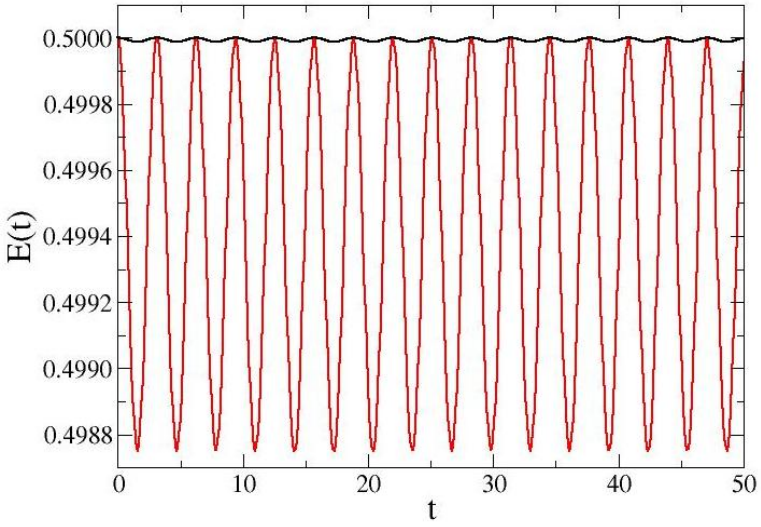
Same harmonic oscillator as before




Euler




Leapfrog $\Delta_t = 0.1, 0.01$




The energy error is not bounded
- not even for periodic motion

No time-reversal symmetry
- can be important in some applications

May be better than Leapfrog when E is not conserved (damping, driving)