

Solving Classical Equations of Motion

Newton's equation of motion govern the dynamics of:

- solar systems, galaxies...
(some relativistic effects, some times large, e.g., binary star systems)
- “everyday motion”; projectiles, baseballs, mechanical machines
- non-electronic aspects of solids, liquids and gases
(some quantum effects for light atoms/molecules)
- potentials from quantum mechanics

We will discuss basic numerical algorithms
(regularly discretized time axis)

- 1D equation, properties of different integrators
- 2D motion, driving forces, dissipation...
- chaos in classical dynamic systems (may be postponed to later)

1D motion - single particle $x(t)$

velocity and acceleration:

$$\dot{x}(t) = \frac{dx(t)}{dt} = v(t), \quad \ddot{x}(t) = \frac{d^2x(t)}{dt^2} = a(t)$$

Given a force F , the equation of motion is

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

- potential depending on x or $x(t)$
- dissipation (friction) depending on v
- location independent driving (t dep)

This 2nd order diff equation
can be written as 2 coupled 1st order eqs:

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

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

Discretized time axis (constant time step):

$$t \in \{t_0, t_1, \dots, t_N\}, \quad \Delta_t = t_{i+1} - t_i$$

Start with given initial conditions at t_0 : $x=x_0$, $v=v_0$

- use approximation to diff equation to integrate forward in time

To go from time t_n to t_{n+1} , in general we can expand around t_n

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

$$v_{n+1} = v_n + \Delta_t a_n + \frac{1}{2} \Delta_t^2 \dot{a}_n + \frac{1}{6} \Delta_t^3 \ddot{a}_n + \dots$$

Simplest case: go to linear order:

Euler forward algorithm

$$\begin{aligned} x_{n+1} &= x_n + \Delta_t v_n \\ v_{n+1} &= v_n + \Delta_t a_n \end{aligned}$$

error in x_{n+1} is $O(\Delta_t^2)$

Julia implementation

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

the path $[x(t), v(t)]$ may be written to a file or processed in some other way.

Note: for consistency with the series expansion, the acceleration should be evaluated with “old” x, v, t (i-1)

The Euler algorithm is not very good in practice
- should not be used in any serious work

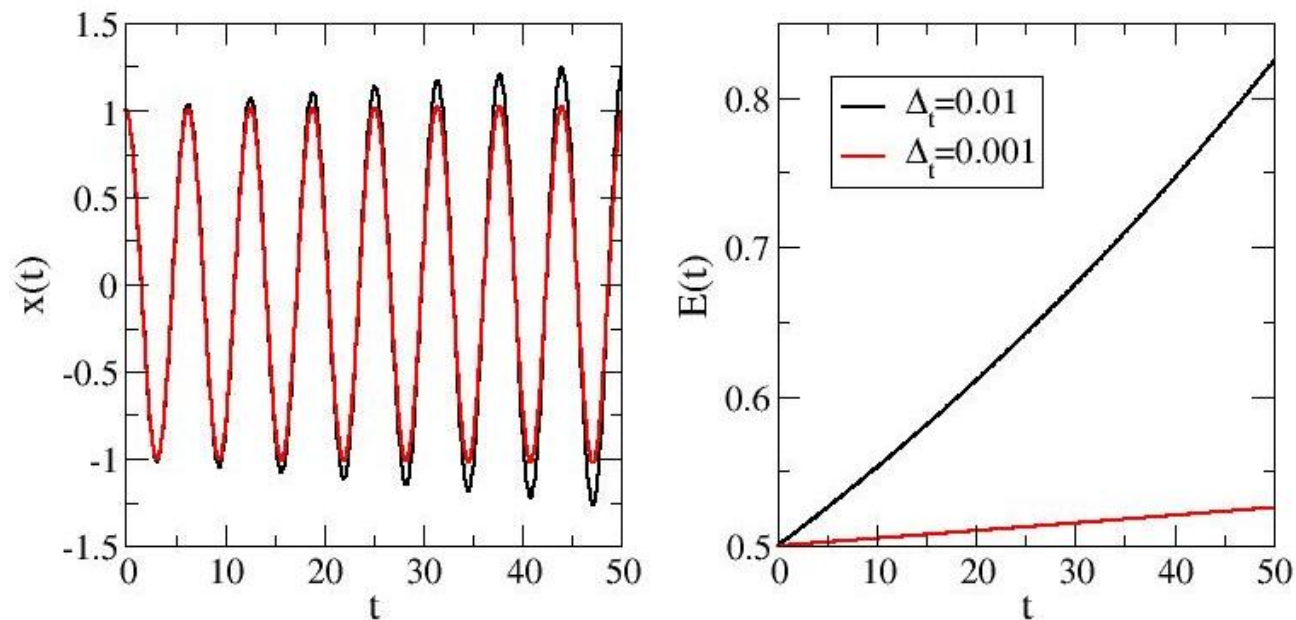
Illustration of the Euler method

Harmonic oscillator

$$E = \frac{1}{2}kx^2 + \frac{1}{2}mv^2 \quad (F = -kx) \quad \text{Periodic motion; } \omega = \sqrt{\frac{k}{m}}$$

Test case: $k=m=1$, $x_0=1$, $v_0=0$

Comparing two different time steps: $\Delta t=0.01$, 0.001



Amplitude increases with time, unbounded energy error

- there are algorithms with bounded energy error for periodic motion

Leapfrog (Verlet) Algorithm

Use second-order form of x:

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

and recall first-order form of v:

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

Re-write x formula as

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

where we can identify “half-step” velocity $v_{n+1/2} = v_n + \frac{1}{2} \Delta_t a_n + O(\Delta_t^2)$

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

We will only have v on the half-step, use $v_{n-1/2}$ to obtain $v_{n+1/2}$

(cubic step error remains intact)

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

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

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

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

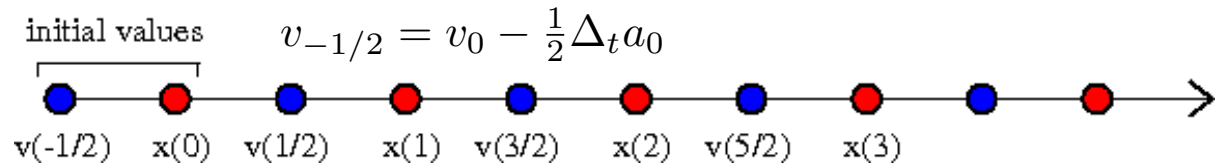
$$\rightarrow v_{n+1/2} = v_{n-1/2} + a_n \Delta_t + O(\Delta_t^3)$$

Note: The acceleration (force) at the integer step n is required

- we do not have v_n
- force cannot be v-dependent here; $a_n = a_n(x_n, t)$

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

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



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:

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

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

add these →

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

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

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

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

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

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

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

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

we use this only if we want to extract v from the x path

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

However, earlier we saw that

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

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

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

[step error in v is $O(\Delta_t^3)$] and so the error in x is still $O(\Delta_t^4)$ when used in x_{n+1}

We conclude that the algorithm is identical to the Leapfrog algorithm

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

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

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

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: $\mathbf{x}_0, \mathbf{x}_1$

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
```

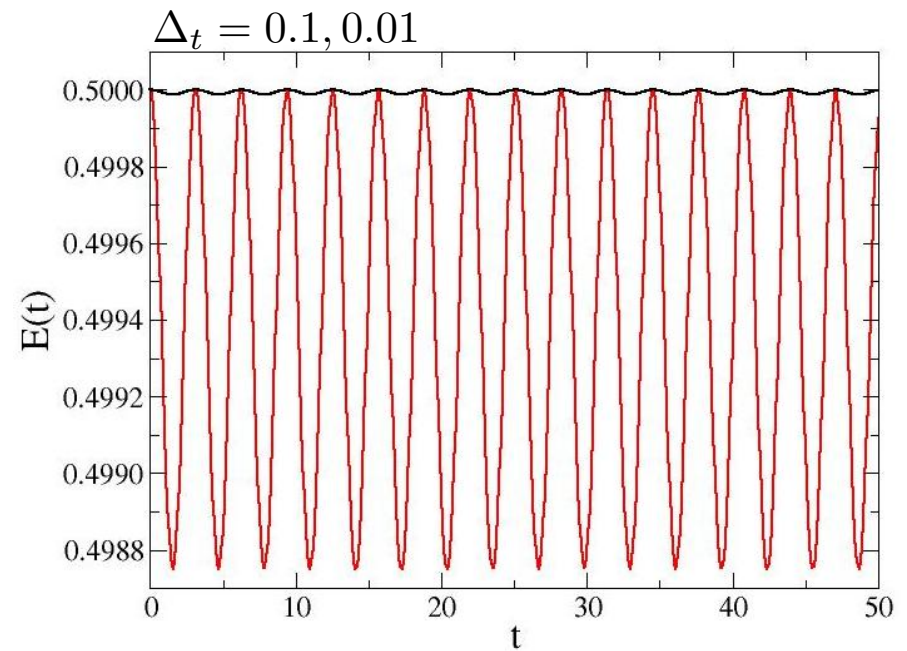
Properties of the method

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

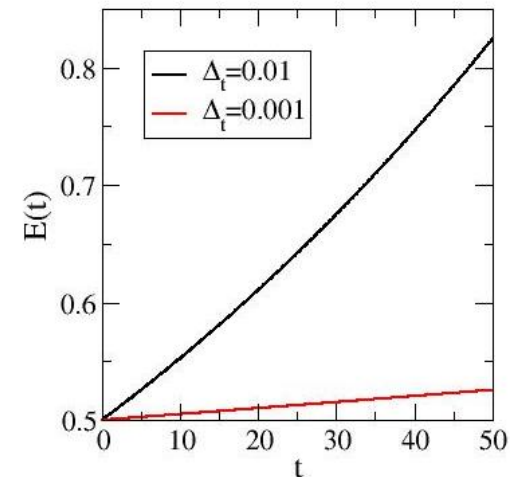
Test: same oscillator as before

Note: Implementation almost identical to Euler

- just swap two lines!



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



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 Δ_t ?

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

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

and use in the Verlet algorithm

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

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

Here we see discrete versions of second derivatives

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

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

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

add these:

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

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

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

Replace discrete time derivatives by continuum versions:

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

The exact solution satisfies

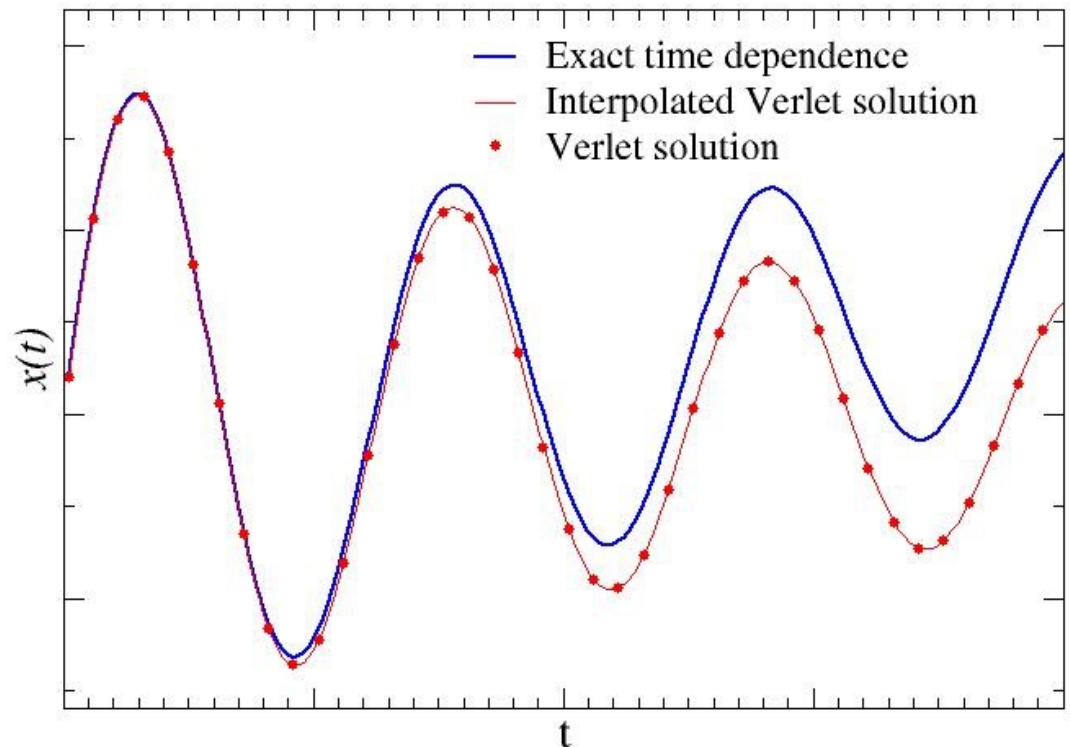
$$\ddot{x}^{\text{ex}}(t) = a(t)$$

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: Δ_t must be sufficiently small for the above arguments to be valid