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:

\[
\begin{align*}
\dot{x}(t) &= v(t) \\
\dot{v}(t) &= a[x(t), v(t), t]
\end{align*}
\]
Discretized time axis
\[ t = t_0, t_1, \ldots, t_N, \quad t_{n+1} - t_n = \Delta t \]
Start from given initial conditions: \[ v = v_0, \quad x = x_0 \]

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:
\[
\begin{align*}
    &\textbf{do } i=1,nt \\
    &\quad t0=dt*(i-1) \\
    &\quad x1=x0+dt*v0 \\
    &\quad v1=v0+dt*acc(x0,v0,t0) \\
    &\quad x0=x1; \; v0=v1 \\
\textbf{enddo} \\
\end{align*}
\]
\[
\begin{align*}
    &\textbf{do } i=1,nt \\
    &\quad t=dt*(i-1) \\
    &\quad a=acc(x,v,t) \\
    &\quad x=x+dt*v \\
    &\quad v=v+dt*a \\
\textbf{enddo} \\
\end{align*}
\]

- 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

\[ E = \frac{1}{2} k x^2 + \frac{1}{2} m v^2 \]  \quad (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**

\[
\begin{align*}
    v_{n+1/2} &= v_{n-1/2} + \Delta_t a_n \\
    x_{n+1} &= x_n + \Delta_t v_{n+1/2}
\end{align*}
\]

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 (switch 2 lines!)

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

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: \( x_n = x_n^{ex} + \delta_n \)

Inserting this in Verlet equation

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

gives

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

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:

\[ \ddot{\delta}(t_n) = -\ddot{x}^{ex}(t_n) + a(t_n) + O(\Delta_t^2) \]
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 \( n \)):

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