

## Lec3 - Leapfrog, many particles

- dt errors. Leapfrog algorithm
- Many particles - balls in boxes

## Recap lecture 2

- Discussed (simplest) method for solving Newton's laws for motion of single particle under external force
- Write as two first order equations:

$$\begin{aligned}\frac{dr}{dt} &= v \\ \frac{dv}{dt} &= \frac{1}{m}F\end{aligned}$$

- Use Euler algorithm to evolve (update) in discrete time
- Harmonic oscillators, projectiles
- Discussed python code to implement this algorithm

## Time step errors

- The Euler method we have used so far has its limitations – solution accurate to  $O(dt)$  only. Why ? error per step  $O(dt^2)$ . Total steps  $T/dt$ . Total error  $dt^2 T/dt = Tdt$
- See this by computing *energy*  
$$E = 1/2mv^2 + 1/2kx^2$$
- Compute error as function of dt ...
- Note: calculate mean error over some finite period of time - see code.
- Can do better ...

## More accurate integrators

Using Taylor ....

$$x(t + dt) = x(t) + v(t)dt + a(t)dt^2/2 + O(dt^3)$$

leading to

$$x_{n+1} = x_n + v_n dt + a_n dt^2/2$$

Subtracting  $x(t - dt)$  from  $x(t + dt)$  find

$$v(t) = \frac{x(t + dt) - x(t - dt)}{2dt} + O(dt^3)$$

leading to

$$v_{n+1} = \frac{x_{n+2} - x_n}{2dt}$$

Substituting for  $x_{n+2}$  using previous equation:

$$v_{n+1} = v_n + \frac{dt}{2}(a_n + a_{n+1})$$

**leapfrog algorithm**

Accurate to  $O(dt^2)$

# Leapfrog algorithm

Almost as simple to implement in code.

- Compute and store new variable `oldforce` to keep initial acceleration.
- `force()` function same as before.
- See *much* smaller errors in energy. Consistent with  $\frac{\Delta E}{E} \sim dt^2$
- *Symplectic integrator*. Time reversible. Exactly conserves some “nearby” Hamiltonian.

## Many particles

- Consider two masses a and b interacting via some mutual force
- Denote force on a due to b as  $F_{ab}$ .
- Likewise force on b due to a as  $F_{ba}$
- By Newton's third law  $F_{ab} = -F_{ba}$  – *vector statement*
- Given a specific force law can we solve Newton's 2nd law for both particles numerically – *simulate* the system ?

## Newton's 2nd law

If many mutually interacting particles:

$$F_a = \sum_b F_{ab}$$

where label  $b$  runs over all particles in system

$$\begin{aligned}\frac{dx_a}{dt} &= v_a \\ \frac{dv_a}{dt} &= F_a/m_a\end{aligned}$$

Get a pair of (vector) equations for each particle. Simulating system requires summing over all particles.

## Example - balls in box

- Consider a force of form

$$F_{ab} = Ae^{-\left(\frac{r_a-r_b}{r_0}\right)^2} \mathbf{n}$$

- $\mathbf{n}$  is unit vector from ball<sub>b</sub> to ball<sub>a</sub>.
- Take  $r_0$  to be 1/10 radius of ball say. Simulates *hard sphere* repulsion.
- Leapfrog error  $O(dt^2)$ . Dimensionless error  $O((\omega dt)^2)$ . where  $\omega$  is characteristic frequency of system (i.e inverse characteristic timescale  $\tau$ ). Stable integration requires  $\omega dt < 1$ . Here  $\omega \sim \frac{\langle v \rangle}{r_0}$

## Python lists and for

Useful to introduce a `list` to store the objects which are interacting

```
system=[balla,ballb]
```

In general lists can comprise numbers or arbitrary abstract objects enclosed in square brackets eg.

```
a=[1,2,3]
```

```
b=[4,5,6]
```

```
c=range(1,10)
```

The statement `c=a+b` concatenates the lists.

## More for loops

To carry out a sum over all balls use `for` command eg.

```
for ball in system:
```

```
    ball.pos=ball.pos+ball.vel*dt+...
```

- Note indentation – all statements at same level are executed before passing to next element of list
- Note give each ball a position, velocity and (2) accelerations.
- Use leapfrog to update.
- Add elastic force for walls of container - reverse velocity if hits wall.

## Integrate module

- Change `force` function to `newforce`. Latter is passed the labels to two individual balls and computes mutual force vector.
- Function `totalforce` gets total force on a single ball.

# Comments

- Notice that in absence of repulsion force trajectories of balls are regular and repeating. But fill all of space when interaction included.
- Note code works just as well for *any* number of balls. Just need to initialize all their positions and velocities.
- Straightforward to model other force laws eg inverse square. Just change definition of `newforce`.
- Can use to model molecules in a gas ...  
Study freezing transition ?

# Summary

- Discussed improved interaction algorithms
- Developed code to study multiparticle systems – here hard spheres
- Hard sphere repulsion leads to *chaotic* trajectories