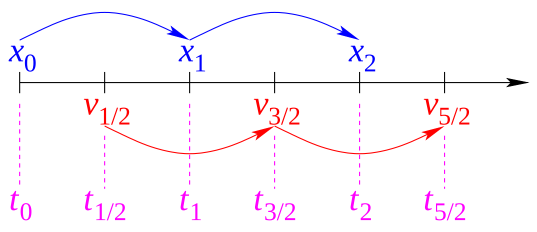
Class 15. ODEs, Part 2

The Leapfrog Integrator

- Very useful for second-order DEs in which $d^2\mathbf{x}/dt^2 = \mathbf{f}(\mathbf{x})$, e.g., SHM, N-body, etc.
  - NOTE: Now dropping the prime ($'$) from $\mathbf{f}$...
- Suppose $\mathbf{x}$ is position, so $d^2\mathbf{x}/dt^2$ is acceleration.
- Procedure: define $\mathbf{v} = d\mathbf{x}/dt$ at the *midpoints* of the steps, i.e., velocities staggered wrt positions.
  - Set $\mathbf{v}_{n+1/2} = \mathbf{v}(t_n + \frac{1}{2}h)$.
  - Then advance $\mathbf{x}_n$ to $\mathbf{x}_{n+1}$ and $\mathbf{v}_{n+1/2}$ to $\mathbf{v}_{n+3/2}$:

$$\begin{aligned}\mathbf{x}_{n+1} &= \mathbf{x}_n + h\mathbf{v}_{n+1/2},\\ \mathbf{v}_{n+3/2} &= \mathbf{v}_{n+1/2} + h\mathbf{f}(\mathbf{x}_{n+1}).\end{aligned}$$



- Complication: need to "jumpstart" and "resync"...

$$\begin{aligned}\mathbf{v}_{n+1/2} &= \mathbf{v}_n + (h/2)\mathbf{f}(\mathbf{x}_n) && \text{[opening "kick": Euler]}\\ \mathbf{x}_{n+1} &= \mathbf{x}_n + h\mathbf{v}_{n+1/2} && \text{["drift"]}\\ \mathbf{v}_{n+1} &= \mathbf{v}_{n+1/2} + (h/2)\mathbf{f}(\mathbf{x}_{n+1}) && \text{[closing "kick": resync]}\end{aligned}$$

  - Note $\mathbf{v}_{n+3/2} = \mathbf{v}_{n+1} + (h/2)\mathbf{f}(\mathbf{x}_{n+1}) = \mathbf{v}_{n+1/2} + h\mathbf{f}(\mathbf{x}_{n+1})$.
- Also have "drift-kick-drift" (DKD) scheme.
- Like midpoint method, Leapfrog is <u>second order</u>:

$$\mathbf{x}(t+h) = \mathbf{x}(t) + h\mathbf{v}(t+h/2),$$

  but

$$\mathbf{v}(t+h/2) = \mathbf{v}(t) + (h/2)\mathbf{f}(t) + \mathcal{O}(h^2).$$

  Therefore

$$\mathbf{x}(t+h) = \mathbf{x}(t) + h\mathbf{v}(t) + (h^2/2)\mathbf{f}(t) + \mathcal{O}(h^3).$$

  - This is formally equivalent to midpoint method.

- So why is Leapfrog so great?...
- Answer: Leapfrog is *time reversible.*
- Suppose we step back from $(t_{n+1}, \mathbf{x}_{n+1}, \mathbf{v}_{n+3/2})$ to $(t_n, \mathbf{x}_n, \mathbf{v}_{n+1/2})$. Applying the algorithm:

$$\begin{aligned}\mathbf{v}_{n+1/2} &= \mathbf{v}_{n+3/2} + (-h)\mathbf{f}(\mathbf{x}_{n+1}), \\ \mathbf{x}_n &= \mathbf{x}_{n+1} + (-h)\mathbf{v}_{n+1/2}.\end{aligned}$$

- These are precisely the steps (in reverse) that we took to advance the system in the first place!
- Hence if we Leapfrog forward in time, then reverse to $t = 0$, we're back to where we started, *precisely.*
- Leapfrog is time reversible because of the symmetric way in which it is defined, unlike the other schemes.
  - In Euler, forward and backward steps do not cancel since they use different derivatives at different times.
  - In Midpoint, the estimate of the derivative is based on an extrapolation from the left-hand side of the interval. On time reversal, the estimate would be based on the right-hand side, not the same.
  - Similarly, RK4 is not time reversible.
- Time reversibility is important because it guarantees conservation of energy, angular momentum, etc. (in many cases).
  - Suppose the integrator makes an error ε after one orbital period. Now reverse. Is the error $-\varepsilon$? No! The time-reversed orbit is a solution of the original ODE (with $\mathbf{v}$ replaced with $-\mathbf{v}$), so the energy error is still $+\varepsilon$. But we've returned to our starting point, so we know the final energy error is zero. Hence $\varepsilon = 0$!
- Leapfrog is only second order, but $\underline{\text{very}}$ stable.
- Leapfrog is an example of a class of "symplectic" integrators that conserve phase-space volume: exactly solves an approximate Hamiltonian system.

$$H = H_D + H_K + H_{\rm err} = \frac{1}{2}v^2 + V(\mathbf{r}) + H_{\rm err},$$

  or $D(h/2)K(h)D(h/2)$, with $H_{\rm err} \sim \mathcal{O}(h^3)$. You can also construct the usual kick-drift-kick scheme, $K(h/2)D(h)K(h/2)$, because the Hamiltonian is separable.

Adaptive Stepsize Control

- Up to now, have assumed stepsize h is constant.
- Clearly prefer choosing h small when $|\mathbf{f}'|$ is large, and h large when $|\mathbf{f}'|$ is small. (We've reintroduced the prime ($'$) notation, just to be confusing...)
- The tradeoff is extra trial steps to determine optimum h, but may achieve factor of 10 to 100 increase in stepsize, so it's often worth it.
- *NRiC* provides a routine `odeint()` for RK4 with adapative stepsize control. Complicated to use!