Ordinary Differential Equations (ODEs)

- *NRiC* Chapter 16.
- ODEs involve derivatives wrt *one* independent variable, e.g. time *t*.
- ODEs can always be reduced to a set of first-order equations (involving *only* first derivatives).

e.g. $$\frac{d^2y}{dt^2} + b(t)\frac{dy}{dt} = c(t)$$

is equivalent to the set $$\frac{dy}{dt} = z(t),\ \frac{dz}{dt} = c(t) - b(t)z(t)$$

ODE Intro, Cont'd

- Usually new variables just derivatives of old, but sometimes need additional factors of t to avoid pathologies.
- General problem is solving set of 1st order ODEs:

$$\frac{dy_i}{dt} = f_i\left(t, y_1, \quad, y_N\right), \quad i = 1, \quad, N$$

f_i' are known functions

- But also need boundary conditions: algebraic conditions on values of y_i at discrete time(s) t...

ODE Boundary Conditions (BC)

- Two categories of BC:
  - Initial Value Problem (IVP): all y_i are given at some starting point t_s, and solution is needed from t_s to t_f.
  - Two-point Boundary Value Problem (BVP): y_i are specified at two or more t, e.g. some at t_s, some at t_f (only one BC needed for each y).
- Generally, IVP much easier to solve than 2-pt BVP, so consider this first.

Finite Differences

- How do you represent derivatives with discrete number system?
- Basic idea: replace *dy*/*dt* with finite differences $\Delta y/\Delta t$. Then:

$$\lim_{\Delta t \to 0} \frac{\Delta y}{\Delta t} \to \frac{dy}{dt}$$

- How do you use this to solve ODEs?

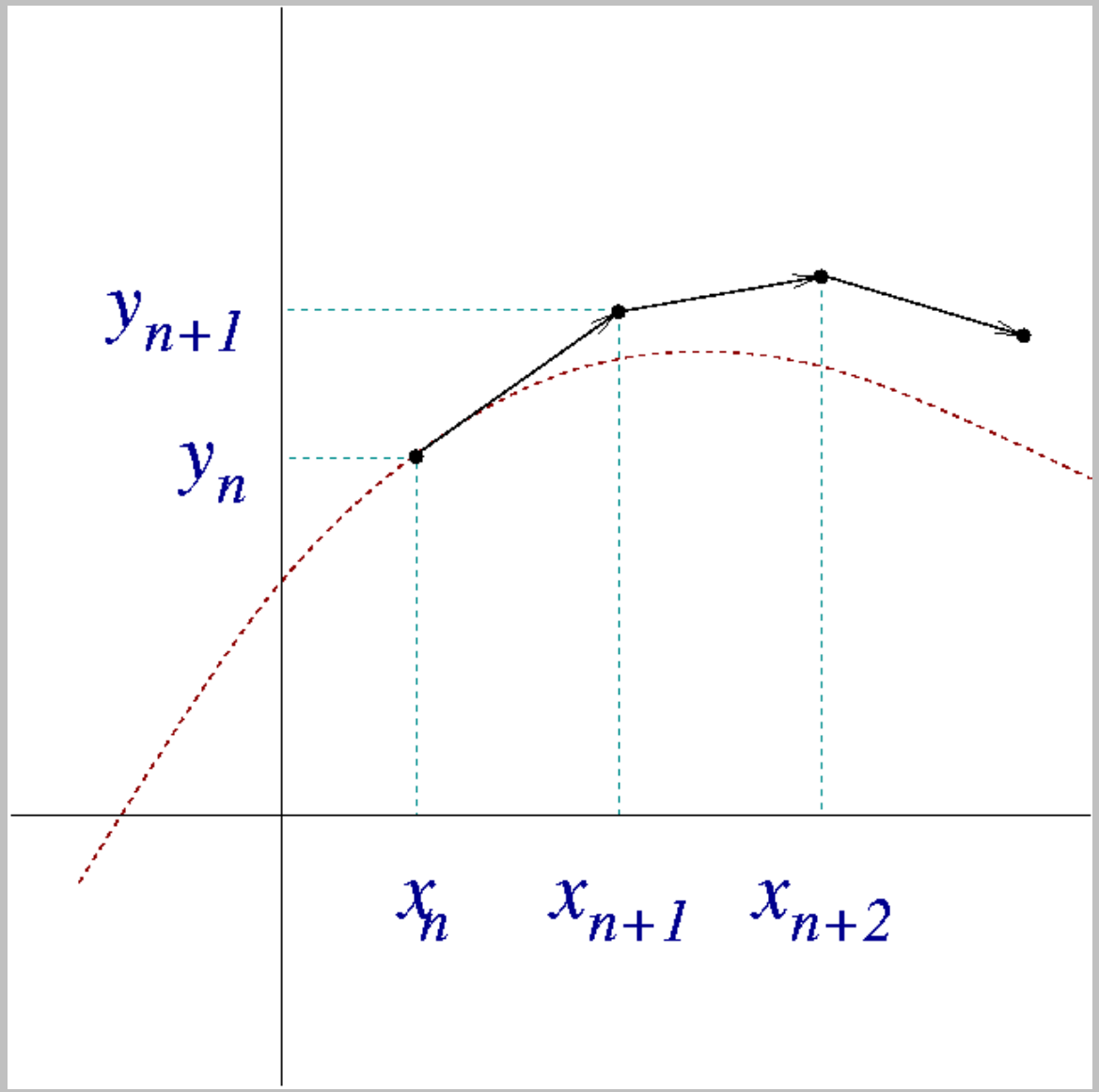
Euler's Method

- Write $\Delta\mathbf{y}/\Delta t = \mathbf{f}'(t,\mathbf{y}) \Rightarrow \Delta\mathbf{y} = \Delta t\, \mathbf{f}'(t,\mathbf{y})$.
- Start with known values $\mathbf{y}_n$ at t_n (initial values).
- Then $\mathbf{y}_{n+1}$ at $t_{n+1} = t_n + h$ is:

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h\, \mathbf{f}'(t_n,\mathbf{y}_n)$$

- h is called the *step size*.
- Integration is not symmetric: derivative evaluated only at start of step $\Rightarrow$ error term $O(h^2)$, from Taylor series. So, Euler's method is first order.

Euler's Method, Cont'd



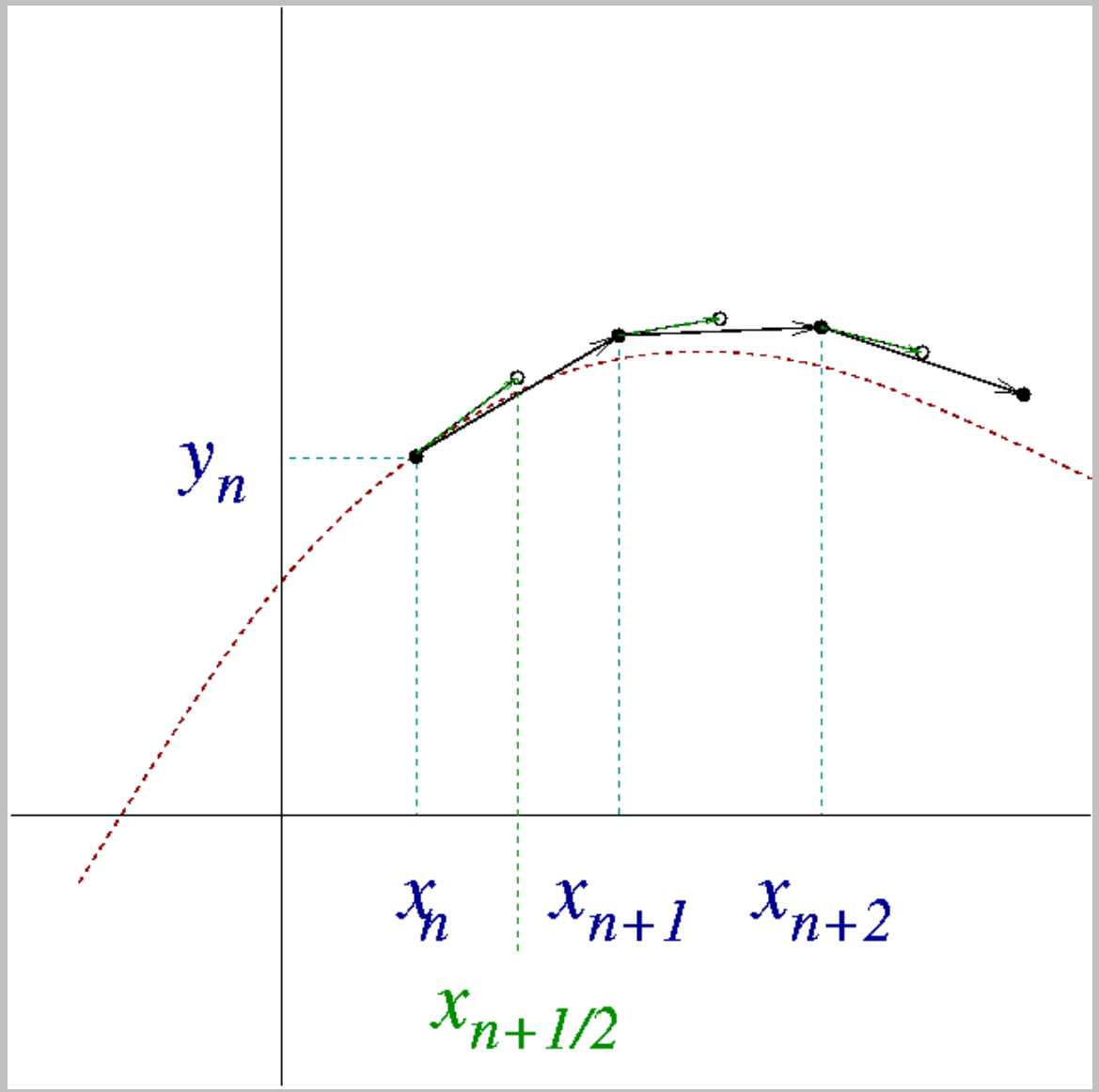
Runge-Kutta Methods

- We can do better by symmetrizing derivative:
  - Take a trial step to midpoint, evaluate $\mathbf{y}_{n+1/2}$ and $t_{n+1/2}$.
  - Use these to evaluate derivative $\mathbf{f}'(t_{n+1/2},\mathbf{y}_{n+1/2})$.
  - Then use this to go back and take a full step.
- Thus:

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h\,\mathbf{f}'(t_n + \tfrac{1}{2}\,h,\mathbf{y}_n + \tfrac{1}{2}\,h\,\mathbf{f}'(t_n,\mathbf{y}_n)) + O(h^3)$$

- Can show that $O(h^2)$ terms "cancel," so leading error term is $O(h^3) \Rightarrow$ 2nd-order Runge-Kutta.

Runga-Kutta, Cont'd



Fourth-Order Runge-Kutta

- Actually, there are many ways to evaluate $\mathbf{f}'$ at midpoints, which add higher order error terms with different coefficients. Can add these together in ways such that higher order error terms cancel.

  e.g. can build fourth-order Runge-Kutta (RK4):

$$\mathbf{k}_1 = h\,\mathbf{f}'(t_n, \mathbf{y}_n)$$

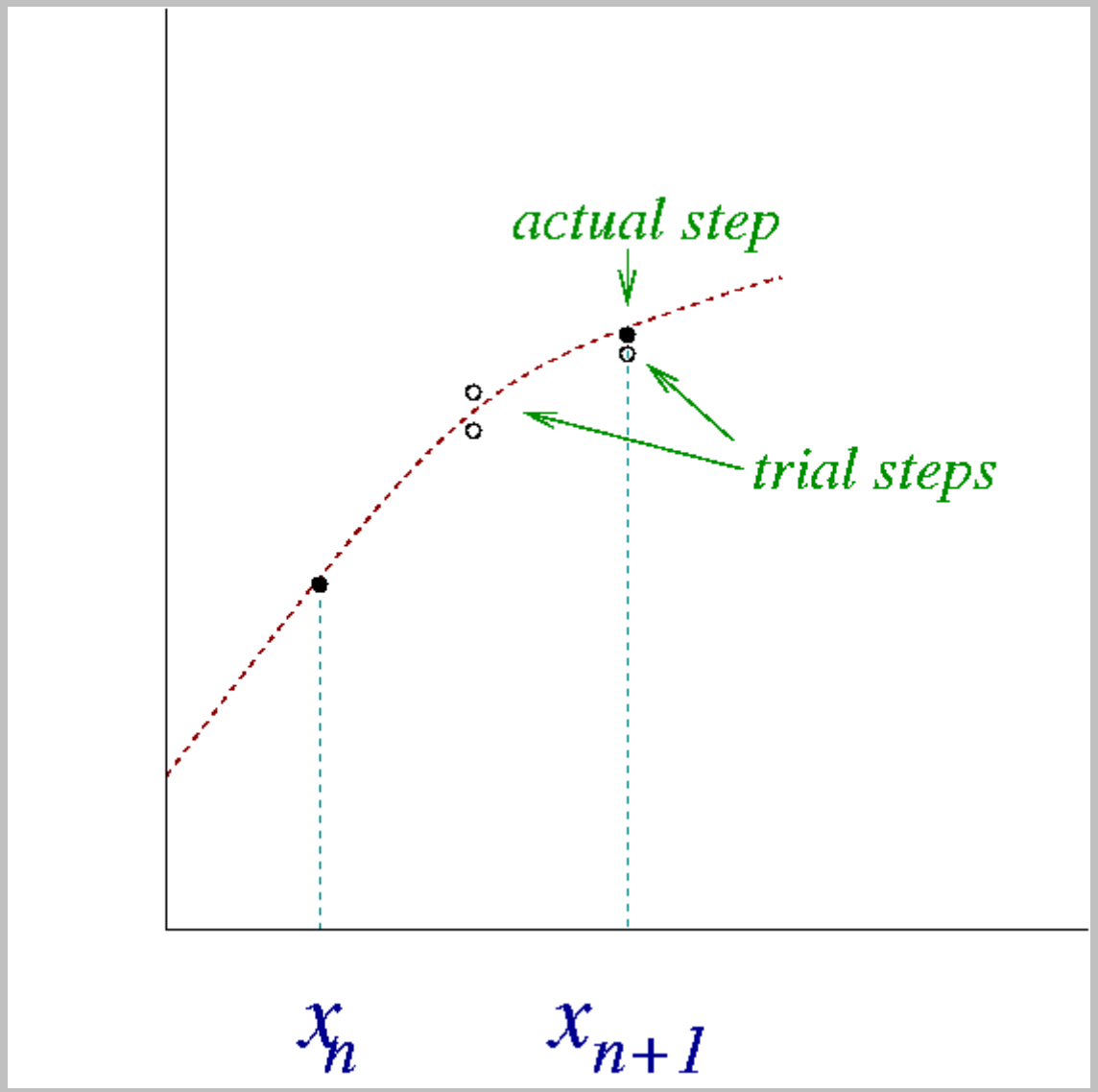
$$\mathbf{k}_2 = h\,\mathbf{f}'(t_n + h/2, \mathbf{y}_n + \mathbf{k}_1/2)$$

$$\mathbf{k}_3 = h\,\mathbf{f}'(t_n + h/2, \mathbf{y}_n + \mathbf{k}_2/2)$$

$$\mathbf{k}_4 = h\,\mathbf{f}'(t_n + h, \mathbf{y}_n + \mathbf{k}_3)$$

Then $\mathbf{y}_{n+1} = \mathbf{y}_n + \mathbf{k}_1/6 + \mathbf{k}_2/3 + \mathbf{k}_3/3 + \mathbf{k}_4/6 + O(h^5)$

Fourth-Order Runge-Kutta, Cont'd



Fourth-Order Runge-Kutta, Cont'd

- Disadvantage of RK4: requires $\mathbf{f}'$ to be evaluated 4 times per step.
- But, can still be cost effective if larger steps OK.
- RK4 is workhorse method. Higher-order RK4 takes too much effort for increased accuracy.
- Other methods (e.g. Bulirsch-Stoer, *NRiC* §16.4) are more accurate for smooth functions.
- But RK4 often "good enough".

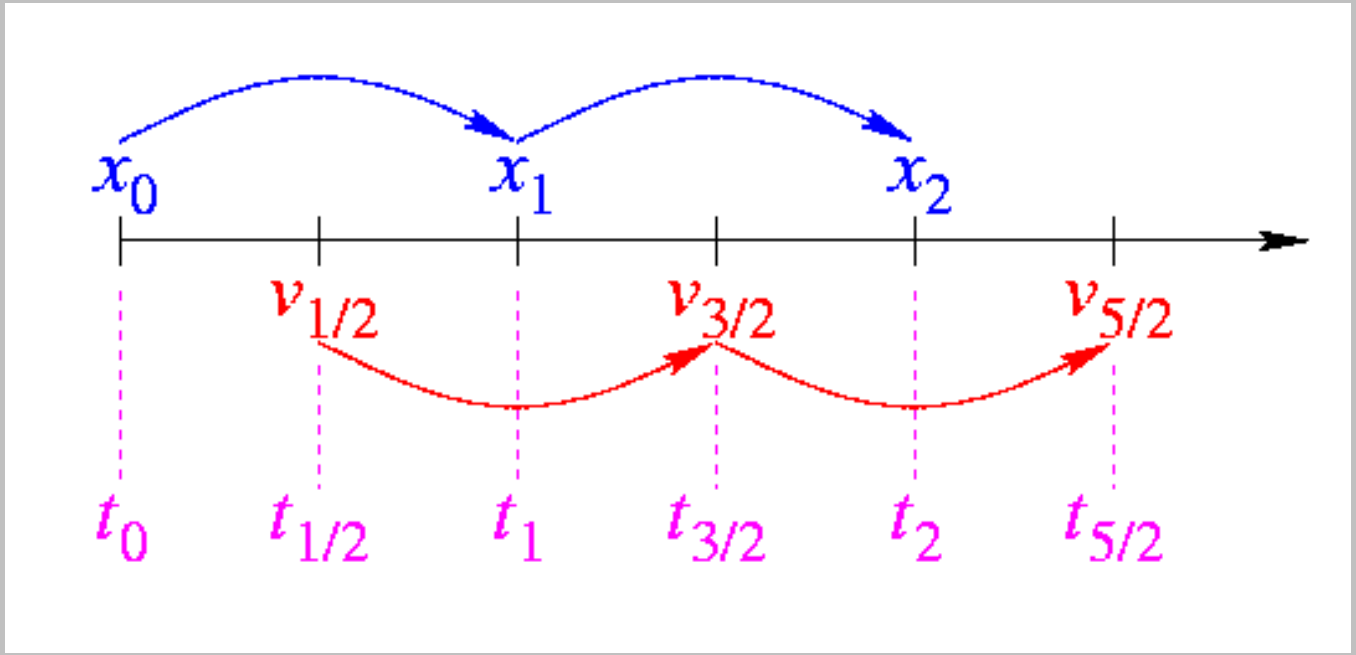
The Leapfrog Integrator

- Extremely useful for second-order DEs in which $d^2x/dt^2 = f(x)$, e.g. SHM, N-body, etc.
- Suppose x is position, so d^2x/dt^2 is acceleration.
- Procedure: define $v = dx/dt$ at the *midpoints* of the steps, i.e. velocities staggered wrt positions.
  - Define $v_{i+1/2} = v(t + \frac{1}{2}\,\delta t)$, $i = 0, 1, 2, \ldots$
  - Then advance x_i to x_{i+1} and $v_{i+1/2}$ to $v_{i+3/2}$:

$$x_{i+1} = x_i + v_{i+1/2}\,\delta t$$

$$v_{i+3/2} = v_{i+1/2} + f(x_{i+1})\,\delta t$$

Leapfrog, Cont'd



Leapfrog, Cont'd

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

$v_{i+1/2} = v_i + \frac{1}{2}\,\delta t\, f(x_i)$ [opening "kick": Euler]

$x_{i+1} = x_i + v_{i+1/2}\,\delta t$ ["drift"]

$v_{i+1} = v_{i+1/2} + \frac{1}{2}\,\delta t\, f(x_{i+1})$ [closing "kick": resync]

➔ Note $v_{i+3/2} = v_{i+1} + \frac{1}{2}\,\delta t\, f(x_{i+1}) = v_{i+1/2} + \delta t\, f(x_{i+1})$.

- Also have "drift-kick-drift" (DKD) scheme.
- Like midpoint method, Leapfrog is second order.
- So why is Leapfrog so great?...

Leapfrog, Cont'd

- Answer: Leapfrog is *time reversible*.
- Suppose we step back from $(t_{i+1},x_{i+1},v_{i+3/2})$ to $(t_i,x_i,v_{i+1/2})$. Applying the algorithm:

$$v_{i+1/2} = v_{i+3/2} + f(x_{i+1})(-\delta t)$$

$$x_i = x_{i+1} + v_{i+1/2}(-\delta t)$$

  - ➔ 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, Cont'd

- 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.

Leapfrog, Cont'd

- 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 v replaced with $-v$), 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 very stable.

Adaptive Stepsize Control

- Up to now, have assumed stepsize h is constant.
- Clearly prefer choosing h small when f' is large, and h large when f' is small.
- 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!