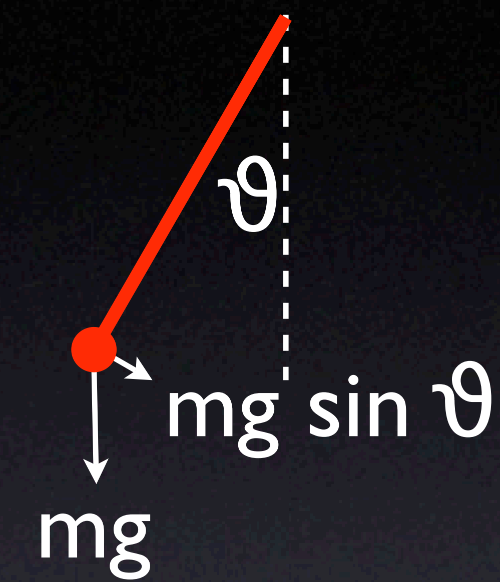
Pendulum & the Walker: solving ODEs numerically

Phys 682 / CIS 629: Computational Methods for Nonlinear Systems

$$d\vec{y}/dt = f(\vec{y}, t)$$



- Simple pendulum

$$\vec{F} = m\vec{a} \implies a = \frac{d^2\theta}{dt^2} = -\frac{g}{L}\sin\theta$$

- Can split second-order ODE into pair of first-order ODEs

$$\frac{d\theta}{dt} = v$$
$$\frac{dv}{dt} = -\frac{g}{L}\sin\theta$$

$\vec{y} = (\theta, v)$ Solve for y(t) given y_0

Discretization: accuracy, fidelity & stability

- Consider simple exponential decay: $\frac{dy}{dt} = -Ay$

- Euler step:

$$\frac{y_{n+1} - y_n}{\Delta t} = -Ay_n \implies y_{n+1} = (1 - A\Delta t)y_n$$

Accuracy:

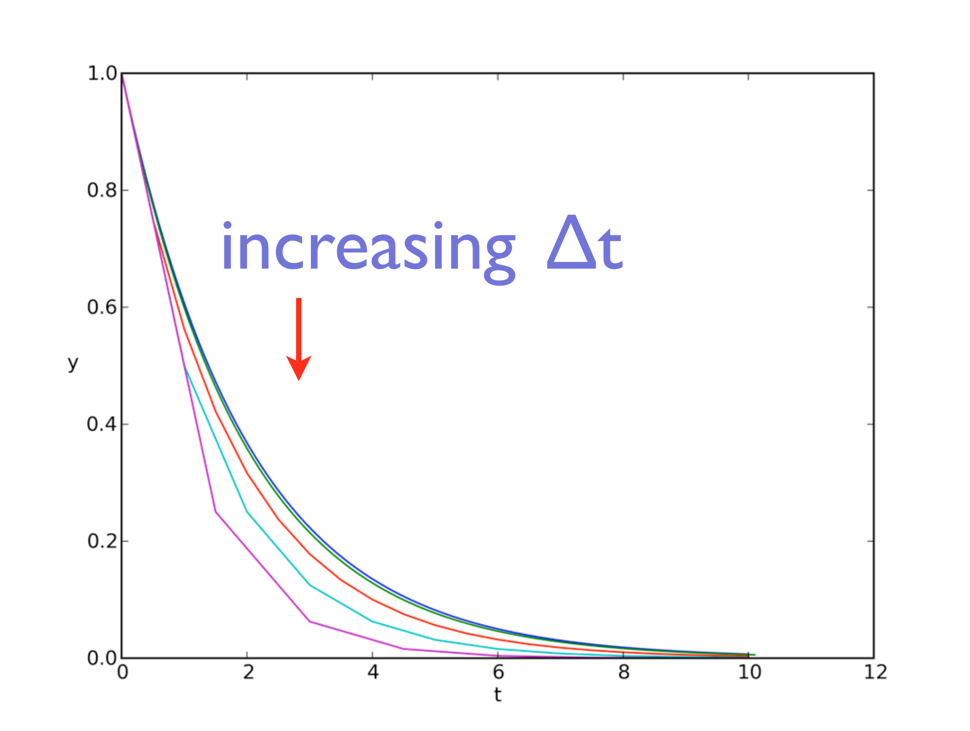
$$e^{(-A\Delta t)}y_n \approx (1 - A\Delta t)y_n + \frac{(A\Delta t)^2}{2!}y_n + \ldots$$

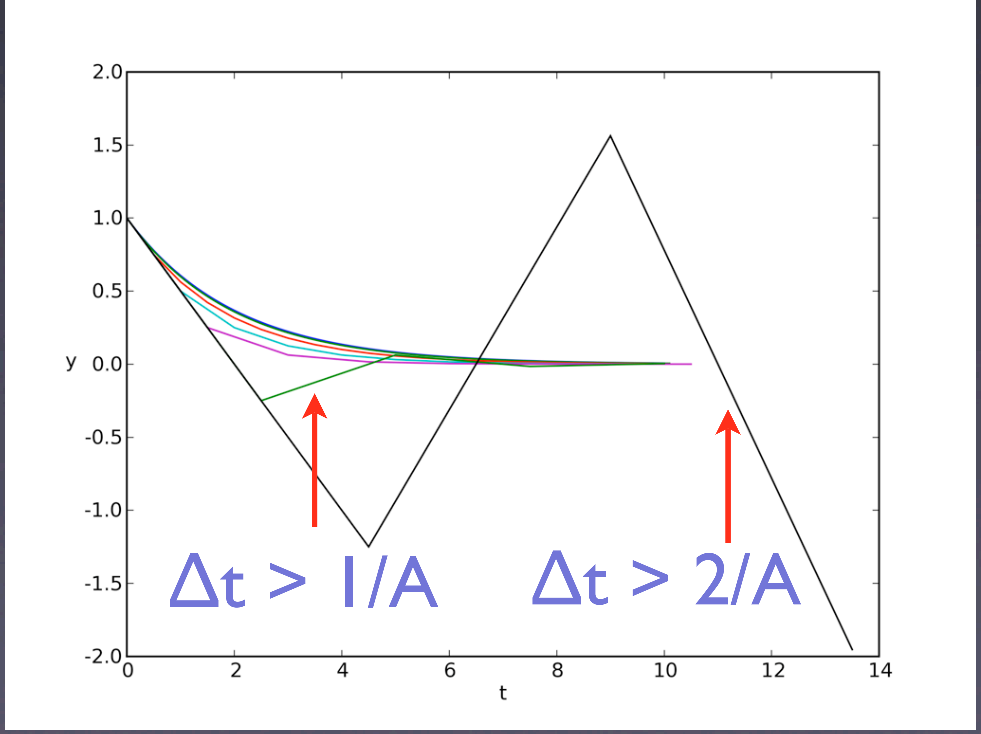
Error in one step = $\frac{(A\Delta t)^2}{2}y_n \sim O(\Delta t^2)$

Error in interval $T = N\Delta t \sim N\Delta t^2 \sim \frac{T}{\Delta t}\Delta t^2 \sim O(\Delta t)$

Fidelity: If Δt > 1/A, y(t) goes negative (unphysical)

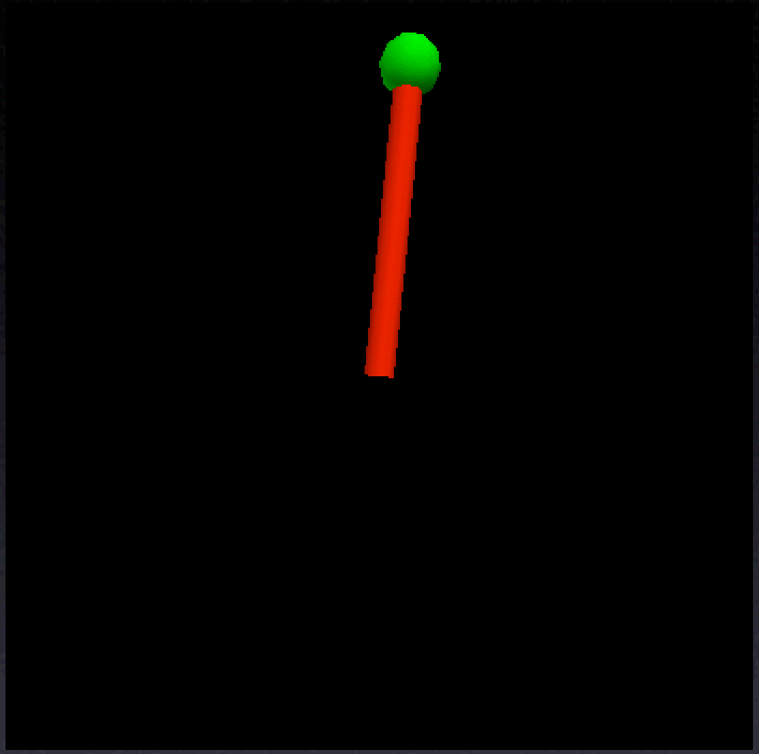
Stability: If Δt > 2/A, y(t) blows up

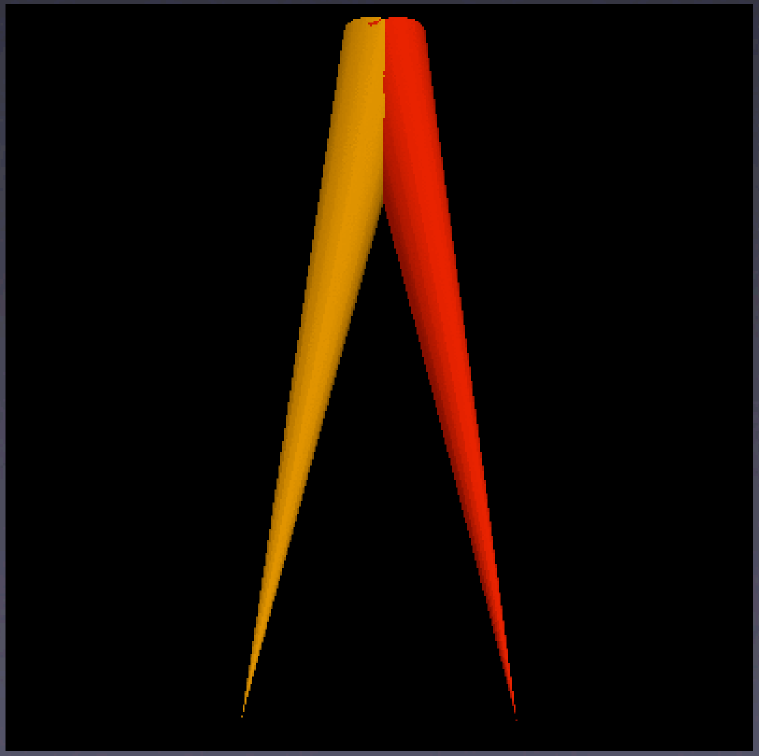




Pendulum & Walker

- Pendulum
  - prerequisite for the Walker
  - not the usual hints-plus-fill-in-the-blanks
  - explore accuracy, fidelity & stability for Hamiltonian system
  - use time-stepping algorithm that conserves an approximate energy (fidelity)
- Walker
  - simple model of bipedal walker (Ruina and coworkers)
  - double pendulum, fixed at stance foot
  - event detection (heel strikes)
    - integrating after change of variables
  - period-doubling bifurcations, leading to chaos
  - use of third-party ODE solver
  - new graphics module (not visual/vpython)





scipy.integrate.odeint

$$d\vec{y}/dt = f(\vec{y}, t)$$

$$\frac{d\theta}{dt} = v$$
$$\frac{dv}{dt} = -\frac{g}{L}\sin\theta$$

$$\vec{y} = (\theta, v)$$

```
import scipy, scipy.integrate, pylab

def dydt(y,t,g,L):
    """return a list or array representing dy/dt, for vector y,
    current time t, and parameters g and L"""
    theta, v = y
    return [v, -(g/L)*scipy.sin(theta)]

g = 9.8; L = 1.
times = scipy.arange(0., 10., 0.1) # times for which y(t) is needed
y0 = [scipy.pi/4., 0.]              # initial condition

y_trajectory = scipy.integrate.odeint(dydt, y0, times), args=(g,L))

# args includes any additional arguments beyond required y and t

pylab.plot(times, y_trajectory[:,0])                 # plot theta vs t
pylab.plot(y_trajectory[:,0], y_trajectory[:,1])   # plot v vs theta
```