Particle Systems & Time Integration
But first…

Office hours Wed @ 3pm, Smith Hall 232
Some dynamical systems can be naturally described as many small particles.
Others are more difficult to get a handle on, but may still be approximated through particle systems.
Particle Basics

- Each particle has a position
  - perhaps other attributes too: orientation, age, color, velocity, temperature, radius, …
  - Call the state $\mathbf{q}$ (aka generalized coordinates)
- Seed randomly somewhere at start
  - Maybe some created each frame
- Move (evolve state $\mathbf{q}$) each frame according to some formula
- Eventually “die” when some condition met
Particle Basics

But let’s start with a 1D particle…

\[ q(t) \]
An ordinary differential equation (ODE) describes how the particle’s state changes through time:

\[ \dot{q}(t) = \frac{dq}{dt}(t) = v(q, t) \]

- ODE defines a vector field in state-space. The function we seek, \( q(t) \), must be solved for
- In general, analytic solutions hopeless
- Need to solve numerically starting from an initial configuration \( q_0 (= q(t=0)) \)
- Main idea: approximate the derivative and discretize in time
Forward Euler

- Simplest method:
  \[ q_{n+1} = q_n + \Delta t v(q_n, t_n) \]

- First order accurate:
  - Global error accumulated over fixed time interval is \( O(\Delta t) \)
  - Thus it converges to the right answer

- But want error to be small
Forward Euler Accuracy

- Obvious approach: make $\Delta t$ small

- But then need more time steps - expensive
  - also note - $O(1)$ error made in modeling
  - even if numerical error was 0, still wrong!
  - need to validate against experiments

- Smaller time steps == better, but if we wanted fastest sim possible, how large could we go?
Forward Euler Stability

The Test Equation

\[ \frac{dq}{dt} = aq, \; a \in \mathbb{C} \]

Linear ODE, known analytic solution \( q(t) = e^{at} \)

- What does it look like?
The Test Equation

- Gives a rough picture of the stability of a method
  - ‘a’ will in general represent eigenvalues of Jacobian (first-order approximation to nonlinear ODEs)

\[ v(q,t) \approx v(q^*,t^*) + \frac{\partial v}{\partial q} \cdot (q - q^*) + \frac{\partial v}{\partial t} \cdot (t - t^*) \]

- Nonlinear effects can definitely cause problems
- Even with linear problems, what follows assumes constant time steps - varying (but supposedly stable) steps can induce instability
  - see J. P. Wright, “Numerical instability due to varying time steps…”, JCP 1998
Using the Test Equation

◆ Forward Euler on test equation is

\[ q_{n+1} = q_n + \Delta t a q_n \]

◆ Solving gives

\[ q_n = (1 + a\Delta t)^n q_0 \]

◆ So for stability, need

\[ |1 + a\Delta t| < 1 \]
Stability Region

- Can plot all the values of $a\Delta t$ on the complex plane where F.E. is stable:

- Big problem with Forward Euler: not very stable
Real Eigenvalue

- Say eigenvalue is real (and negative)
  - Corresponds to a damping motion, smoothly coming to a halt
- Then need:
  \[ \Delta t < \frac{2}{|a|} \]
- Is this bad?
  - If ‘a’ is big, could mean small time steps needed for stability (aka stiff problem)
Imaginary Eigenvalue

- If eigenvalue is pure imaginary (oscillatory or rotational motion), cannot make $\Delta t$ small enough

- Forward Euler unconditionally unstable for these kinds of problems!

- Need to look at other methods
Runge-Kutta Methods

- Also “explicit”
  - next q is an explicit function of previous
- But evaluate \( v \) at a few locations to get a better estimate of next \( q \)
- E.g. midpoint method (RK2)

\[
q_{n+1/2} = q_n + \frac{1}{2} \Delta t v(q_n, t_n) \\
q_{n+1} = q_n + \Delta t v(q_{n+1/2}, t_{n+1/2})
\]
Midpoint RK2

- Second order: error is $O(\Delta t^2)$
- Larger stability region:

- But still not stable on imaginary axis
RK4

- Often most bang for the buck
- Combination of Forward Euler steps and averaging

\[ v_1 = v(q_n, t_n) \]
\[ v_2 = v(q_n + \frac{1}{2} \Delta t v_1, t_{n+\frac{1}{2}}) \]
\[ v_3 = v(q_n + \frac{1}{2} \Delta t v_2, t_{n+\frac{1}{2}}) \]
\[ v_4 = v(q_n + \Delta t v_3, t_{n+1}) \]
\[ q_{n+1} = q_n + \Delta t \left( \frac{1}{6} v_1 + \frac{2}{6} v_2 + \frac{2}{6} v_3 + \frac{1}{6} v_4 \right) \]
Higher Order Runge-Kutta

- RK3 and up naturally include part of the imaginary axis
Implicit Methods
Backward Euler

- The simplest implicit method:

\[ q_{n+1} = q_n + \Delta t v(q_{n+1}, t_{n+1}) \]

- the next x \textit{implicitly} defined since it appears in derivative
  - Need to solve equations to figure it out

- First order accurate

- We’ll come back to it later for a general formulation!
Test equation shows stable when \( |1 - a\Delta t| > 1 \)

This includes everything except a circle in the positive real-part half-plane. *Unconditionally stable* for linear ODEs.

It’s stable even when the physics is unstable!

This is the biggest problem: damps out motion unrealistically
Trapezoidal Rule

- Can improve by going to second order:
  \[
  q_{n+1} = q_n + \Delta t \left( \frac{1}{2} v(q_n, t_n) + \frac{1}{2} v(q_{n+1}, t_{n+1}) \right)
  \]

- This is actually just a half step of F.E., followed by a half step of B.E.
  - F.E. is under-stable, B.E. is over-stable, the combination is **just right**

- Stability region is the left half of the plane: **exactly** the same as the underlying ODE!

- Really good for pure rotation (doesn’t amplify or damp)
What to ask for from a numerical integrator?

- No one “best” integrator – pick the right tool for the job!
- Many different integrators because there are many notions of “good”
  - Convergence/accuracy
  - Stability
  - Computational Efficiency
  - Monotonicity
  - …
Monotonicity

- Test equation with real, negative \( \lambda \)
  - True solution is \( x(t) = x_0 e^{\lambda t} \), which smoothly decays to zero, doesn’t change sign (monotone)

- Forward Euler at stability limit:
  - \( x = x_0, -x_0, x_0, -x_0, \ldots \)

- Not smooth, oscillating sign, no good!

- So monotonicity limit stricter than stability
Monotonicity

◆ Backward Euler is unconditionally monotone
  • No problems with oscillation, just too much damping

◆ Trapezoidal Rule can suffer though, because of that half-step of F.E.
  • could get ugly oscillation instead of smooth damping
  • for some nonlinear problems, possible to hit instability
Need to move particles in velocity field according to underlying ODE

Forward Euler
• Simple, first choice unless problem has oscillations/rotations

Runge-Kutta is better, but requires more evaluations
• RK4 general purpose workhorse
Summary 2

- If stability limit is a problem, look at implicit methods
  - e.g. explicit time steps are way too small

- Trapezoidal Rule
  - If monotonicity isn’t a problem

- Backward Euler
  - Almost always works, but may over-damp!