Notes

- Assignment 1 is not out yet :-)
- http://www.cs.ubc.ca/~rbridson/courses/533d-winter-2005

Monotonicity

- Test equation with real, negative λ
  - True solution is $x(t)=x_0e^{\lambda t}$, 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: garbage!
- So monotonicity limit stricter than stability
- RK3 has the same problem
  - But the even order RK are fine for linear problems
  - TVD-RK3 designed so that it's fine when F.E. is, even for nonlinear problems!

Monotonicity and Implicit Methods

- Backward Euler is unconditionally monotone
  - No problems with oscillation, just too much damping
- Trapezoidal Rule suffers though, because of that half-step of F.E.
  - Beware: could get ugly oscillation instead of smooth damping
  - For nonlinear problems, quite possibly hit instability

Summary 1

- Particle Systems: useful for lots of stuff
- Need to move particles in velocity field
- Forward Euler
  - Simple, first choice unless problem has oscillation/rotation
- Runge-Kutta if happy to obey stability limit
  - Modified Euler may be cheapest method
  - RK4 general purpose workhorse
  - TVD-RK3 for more robustness with nonlinearity (more on this later in the course!)

Summary 2

- If stability limit is a problem, look at implicit methods
  - e.g. need to guarantee a frame-rate, or explicit time steps are way too small
- Trapezoidal Rule
  - If monotonicity isn't a problem
- Backward Euler
  - Almost always works, but may over-damp!

Second Order Motion

Second Order Motion

- If particle state is just position (and colour, size, …) then 1st order motion
  - No inertia
  - Good for very light particles that stay suspended : smoke, dust…
  - Good for some special cases (hacks)
- But most often, want inertia
  - State includes velocity, specify acceleration
  - Can then do parabolic arcs due to gravity, etc.
- This puts us in the realm of standard Newtonian physics
  - F=ma
- Alternatively put:
  - dx/dt=v
  - dv/dt=F(x,v,t)/m (i.e. a(x,v,t))
- For systems (with many masses) say $dv/dt=M^{-1}F(x,v,t)$ where M is the "mass matrix" - masses on the diagonal

What's New?

- If **x**=(x,v) this is just a special form of 1st order: d**x**/dt=**v**(x,t)
- But since we know the special structure, can we take advantage of it? (i.e. better time integration algorithms)
  - More stability for less cost?
  - Handle position and velocity differently to better control error?

Linear Analysis

- Approximate acceleration:

$$a(x,v) \approx a_0 + \frac{\partial a}{\partial x} x + \frac{\partial a}{\partial v} v$$

- Split up analysis into different cases
- Begin with first term dominating: constant acceleration
  - e.g. gravity is most important

Constant Acceleration

- Solution is

$$v(t) = v_0 + a_0 t$$
$$x(t) = x_0 + v_0 t + \tfrac{1}{2} a_0 t^2$$

- No problem to get v(t) right: just need 1st order accuracy
- But x(t) demands 2nd order accuracy
- So we can look at mixed methods:
  - 1st order in v
  - 2nd order in x

Linear Acceleration

- Dependence on x and v dominates:
  a(x,v)=-Kx-Dv
- Do the analysis as before:

$$\frac{d}{dt}\begin{pmatrix} x \\ v \end{pmatrix} = \begin{pmatrix} 0 & I \\ -K & -D \end{pmatrix}\begin{pmatrix} x \\ v \end{pmatrix}$$

- Eigenvalues of this matrix?

More Approximations…

- Typically K and D are symmetric semi-definite (there are good reasons)
  - What does this mean about their eigenvalues?
- Often, D is a linear combination of K and I ("Rayleigh damping"), or at least close to it
  - Then K and D have the same eigenvectors (but different eigenvalues)
  - Then the eigenvectors of the Jacobian are of the form $(u, \alpha u)^T$
  - [work out what α is in terms of λ_K and λ_D]

Simplification

- α is the eigenvalue of the Jacobian, and

$$\alpha = -\tfrac{1}{2}\lambda_D \pm \sqrt{\left(\tfrac{1}{2}\lambda_D\right)^2 - \lambda_K}$$

- Same as eigenvalues of $\begin{pmatrix} 0 & 1 \\ -\lambda_K & -\lambda_D \end{pmatrix}$
- Can replace K and D (matrices) with corresponding eigenvalues (scalars)
  - Just have to analyze 2x2 system

Split Into More Cases

- Still messy! Simplify further
- If D dominates (e.g. air drag, damping)

$$\alpha \approx \{-\lambda_D, 0\}$$

  - Exponential decay and constant
- If K dominates (e.g. spring force)

$$\alpha \approx \pm i\sqrt{\lambda_K}$$

Three Test Equations

- Constant acceleration (e.g. gravity)
  - a(x,v,t)=g
  - Want exact (2nd order accurate) position
- Position dependence (e.g. spring force)
  - a(x,v,t)=-Kx
  - Want stability but low damping
  - Look at imaginary axis
- Velocity dependence (e.g. damping)
  - a(x,v,t)=-Dv
  - Want stability, smooth decay
  - Look at negative real axis

Explicit methods from before

- Forward Euler
  - Constant acceleration: bad (1st order)
  - Position dependence: very bad (unstable)
  - Velocity dependence: ok (conditionally monotone/stable)
- RK3 and RK4
  - Constant acceleration: great (high order)
  - Position dependence: ok (conditionally stable, but damps out oscillation)
  - Velocity dependence: ok (conditionally monotone/stable)

Implicit methods from before

- Backward Euler
  - Constant acceleration: bad (1st order)
  - Position dependence: ok (stable, but damps)
  - Velocity dependence: great (monotone)
- Trapezoidal Rule
  - Constant acceleration: great (2nd order)
  - Position dependence: great (stable, no damping)
  - Velocity dependence: good (stable but only conditionally monotone)

Setting Up Implicit Solves

- Let's take a look at actually using Backwards Euler, for example

$$x_{n+1} = x_n + \Delta t\, v_{n+1}$$
$$v_{n+1} = v_n + \Delta t\, M^{-1} F(x_{n+1}, v_{n+1})$$

- Eliminate position, solve for velocity:

$$v_{n+1} = v_n + \Delta t\, M^{-1} F(x_n + \Delta t\, v_{n+1}, v_{n+1})$$

- Linearize at guess v^k, solving for $v_{n+1} \approx v^k + \Delta v$

$$v^k + \Delta v = v_n + \Delta t\, M^{-1}\left(F\left(x_n + \Delta t\, v^k, v^k\right) + \Delta t \frac{\partial F}{\partial x}\Delta v + \frac{\partial F}{\partial v}\Delta v\right)$$

- Collect terms, multiply by M

$$\left(M - \Delta t \frac{\partial F}{\partial v} - \Delta t^2 \frac{\partial F}{\partial x}\right)\Delta v = M\left(v_n - v^k\right) + \Delta t\, F\left(x_n + \Delta t\, v^k, v^k\right)$$

Symmetry

- Why multiply by M?
- Physics often demands that $\frac{\partial F_{position}}{\partial x}$ and $\frac{\partial F_{velocity}}{\partial v}$ are symmetric
  - And M is symmetric, so this means matrix is symmetric, hence easier to solve
  - (Actually, physics generally says matrix is SPD - even better)
  - If the masses are not equal, the acceleration form of the equations results in an unsymmetric matrix - bad.
- Unfortunately the matrix $\frac{\partial F_{velocity}}{\partial x}$ is usually unsymmetric
  - Makes solving with it considerably less efficient
  - See Baraff & Witkin, “Large steps in cloth simulation”, SIGGRAPH ‘98 for one solution: throw out bad part

Specialized 2nd Order Methods

- This is again a big subject
- Again look at explicit methods, implicit methods
- Also can treat position and velocity dependence differently: mixed implicit-explicit methods

Symplectic Euler

- Like Forward Euler, but updated velocity used for position

$$v_{n+1} = v_n + \Delta t\, a(x_n, v_n)$$
$$x_{n+1} = x_n + \Delta t\, v_{n+1}$$

- Some people flip the steps (= relabel v_n)
- (Symplectic means certain qualities are preserved in discretization; useful in science, not necessarily in graphics)
- [work out test cases]

Symplectic Euler performance

- Constant acceleration: bad
  - Velocity right, position off by O(Δt)
- Position dependence: good
  - Stability limit $\Delta t < \frac{2}{\sqrt{K}}$
  - No damping!
- Velocity dependence: ok
  - Monotone limit $\Delta t < 1/D$
  - Stability limit $\Delta t < 2/D$

Tweaking Symplectic Euler

- [sketch algorithms]
- Stagger the velocity to improve x
- Start off with

$$v_{1/2} = v_0 + \tfrac{1}{2}\Delta t\, a(x_0, v_0)$$

- Then proceed with

$$v_{n+1/2} = v_{n-1/2} + \tfrac{1}{2}(t_{n+1} - t_{n-1})\, a\left(x_n, v_{n-1/2}\right)$$
$$x_{n+1} = x_n + \Delta t\, v_{n+1/2}$$

- Finish off with

$$v_N = v_{N-1/2} + \tfrac{1}{2}\Delta t\, a\left(x_N, v_{N-1/2}\right)$$

Staggered Symplectic Euler

- Constant acceleration: great!
  - Position is exact now
- Other cases not effected
  - Was that magic? Main part of algorithm unchanged (apart from relabeling) yet now it’s more accurate!
- Only downside to staggering
  - At intermediate times, position and velocity not known together
  - May need to think a bit more about collisions and other interactions with outside algorithms…

A common explicit method

- May see this one pop up:

$$v_{n+1} = v_n + \Delta t\, a(x_n, v_n)$$
$$x_{n+1} = x_n + \Delta t\left(\tfrac{1}{2} v_n + \tfrac{1}{2} v_{n+1}\right) = x_n + \Delta t\, v_n + \tfrac{1}{2} \Delta t^2 a_n$$

- Constant acceleration: great
- Velocity dependence: ok
  - Conditionally stable/monotone
- Position dependence: **BAD**
  - Unconditionally unstable!

An Implicit Compromise

- Backward Euler is nice due to unconditional monotonicity
  - Although only 1st order accurate, it has the right characteristics for damping
- Trapezoidal Rule is great for everything except damping with large time steps
  - 2nd order accurate, doesn't damp pure oscillation/rotation
- How can we combine the two?

Implicit Compromise

- Use Backward Euler for velocity dependence, Trapezoidal Rule for the rest:

$$x_{n+1} = x_n + \Delta t\left(\tfrac{1}{2} v_n + \tfrac{1}{2} v_{n+1}\right)$$
$$v_{n+1} = v_n + \Delta t\, a\left(\tfrac{1}{2} x_n + \tfrac{1}{2} x_{n+1}, v_{n+1}, t_{n+1/2}\right)$$

- Constant acceleration: great (2nd order)
- Position dependence: great (2nd order, no damping)
- Velocity dependence: great (unconditionally monotone)