**Notes**

- Most of assignment 1 hasn’t been covered in class yet, but after today you should be able to do a lot of it.
- Forgot to include instructions about `view_obj`:
  - To navigate, hold down shift and click/drag with left, right, or middle mouse buttons (same navigation model as Maya).

**Trapezoidal Rule Again**

- The method:
  \[ x_{n+1} = x_n + \Delta t \left( \frac{1}{2} v(x_n, t_n) + \frac{1}{2} v(x_{n+1}, t_{n+1}) \right) \]
- Let’s work out stability:
  \[ x_{n+1} = x_n + \Delta t \left( \frac{1}{2} \lambda x_n + \frac{1}{2} \lambda x_{n+1} \right) \]
  \[ (1 - \frac{1}{2} \lambda \Delta t) x_{n+1} = (1 + \frac{1}{2} \lambda \Delta t) x_n \]
  \[ x_{n+1} = \frac{1 + \frac{1}{2} \lambda \Delta t}{1 - \frac{1}{2} \lambda \Delta t} x_n \]

**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: garbage!
- So monotonicity limit stricter than stability in this case.
- 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.
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

- 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:
  - \( \frac{dx}{dt} = v \)
  - \( \frac{dv}{dt} = F(x,v,t)/m \) (i.e. \( a(x,v,t) \))
- For systems (with many masses) say \( \frac{dv}{dt} = M^{-1}F(x,v,t) \) 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: \( \frac{dx}{dt}=v(x,t) \)
- 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
  \[
  v(t) = v_0 + a_0 t \\
  x(t) = x_0 + v_0 t + \frac{1}{2} a_0 t^2
  \]
- 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 \(\alpha\) is in terms of \(\lambda_K\) and \(\lambda_D\)]

Simplification

- \(\alpha\) is the eigenvalue of the Jacobian, and
  \[
  \alpha = -\frac{1}{2} \lambda_D \pm \sqrt{\left(\frac{1}{2} \lambda_D\right)^2 - \lambda_K}
  \]
- 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)
  \[
  \alpha \approx \pm \sqrt{-1} \sqrt{\lambda_K}
  \]

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 or zero damping
  - Look at imaginary axis
- Velocity dependence (e.g. damping)
  - \(a(x,v,t)=-Dv\)
  - Want stability, monotone 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 \( v_{n+1} \approx v^k + \Delta v \)
  \[
  v^k + \Delta v = v_n + \Delta t M^{-1} F(x_n + \Delta t v^k, v^k) + \Delta t \frac{\partial F}{\partial x} \Delta x + \frac{\partial F}{\partial v} \Delta v
  \]
- Collect terms, multiply by \( M \)
  \[
  \left( M - \Delta t \frac{\partial F}{\partial v} - \Delta t^2 \frac{\partial^2 F}{\partial x^2} \right) \Delta v = M \left( v_n - v^k \right) + \Delta t F(x_n + \Delta t v^k, v^k)
  \]

Symmetry

- Why multiply by \( M \)?
- Physics often demands that \( \frac{\partial F_{\text{position}}}{\partial x} \) and \( \frac{\partial F_{\text{velocity}}}{\partial v} \) are symmetric
  - And \( M \) is symmetric, so this means matrix is symmetric, hence easier to solve
  - (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 \( \frac{\partial F_{\text{velocity}}}{\partial x} \) 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 of the underlying physics are preserved in discretization - quite desirable visually!
- [work out test cases]

Symplectic Euler performance

- Constant acceleration: bad
  - Velocity right, position off by \(O(\Delta t)\)
- Position dependence: good
  - Stability limit \(\Delta t < \frac{2}{\sqrt{K}}\)
  - No damping! (symplectic)
- 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
  \[ v_{\frac{N}{2}} = v_0 + \frac{1}{2} \Delta t a(x_0, v_0) \]
- Then proceed with
  \[ v_{n+\frac{1}{2}} = v_{n-\frac{1}{2}} + \frac{1}{2} (t_{n+1} - t_{n-1}) a(x_n, v_{n-\frac{1}{2}}) \]
  \[ x_{n+1} = x_n + \Delta t v_{n+\frac{1}{2}} \]
- Finish off with
  \[ v_N = v_{N-\frac{1}{2}} + \frac{1}{2} \Delta t a(x_N, v_{N-\frac{1}{2}}) \]
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:
  
  \[ v_{n+1} = v_n + \Delta t a(x_n, v_n) \]
  
  \[ x_{n+1} = x_n + \Delta t \left( \frac{1}{2} v_n + \frac{1}{2} v_{n+1} \right) = x_n + \Delta t v_n + \frac{1}{2} \Delta t^2 a_n \]

- 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:
  
  \[ x_{n+1} = x_n + \Delta t \left( \frac{1}{2} v_n + \frac{1}{2} v_{n+1} \right) \]
  
  \[ v_{n+1} = v_n + \Delta t a\left( \frac{1}{2} x_n + \frac{1}{2} x_{n+1}, v_{n+1}, t_{n+\frac{1}{2}} \right) \]

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