Time Stepping, Integration, and Events

Peter Eastman

SimTK Workshop, September 25, 2008

Time Stepping

- *Time stepping* = evolving a system through time
- It evolves in two ways
  - Continuous evolution
    - Done by integrating equations of motion
  - Discrete events
    - Change the system discontinuously at specific times
  - Physical processes are mostly continuous
  - Models of them may be discrete or hybrid

Discrete Systems

- Discrete time steps and/or event driven
- Inherently discrete
  - Population dynamics
  - Cellular automata
  - Queuing systems
- Discrete models of continuous systems
  - "instantaneous" state changes, e.g. impulsive collisions, stochastic activity
  - Model changes (e.g. bond forms, bone breaks, add/remove molecule)
  - Sampling, other data reduction

Continuous Systems

- Smooth changes
- Modeled with differential equations, e.g. Newton's 2nd law:

$$\mathbf{M}(q)\ddot{q} = \mathbf{f}(t, q, \dot{q})$$

- Inherently continuous
  - flow, mechanics, deformation
- Continuous models of discrete systems
  - Population dynamics, traffic flow
  - Implicit solvation models

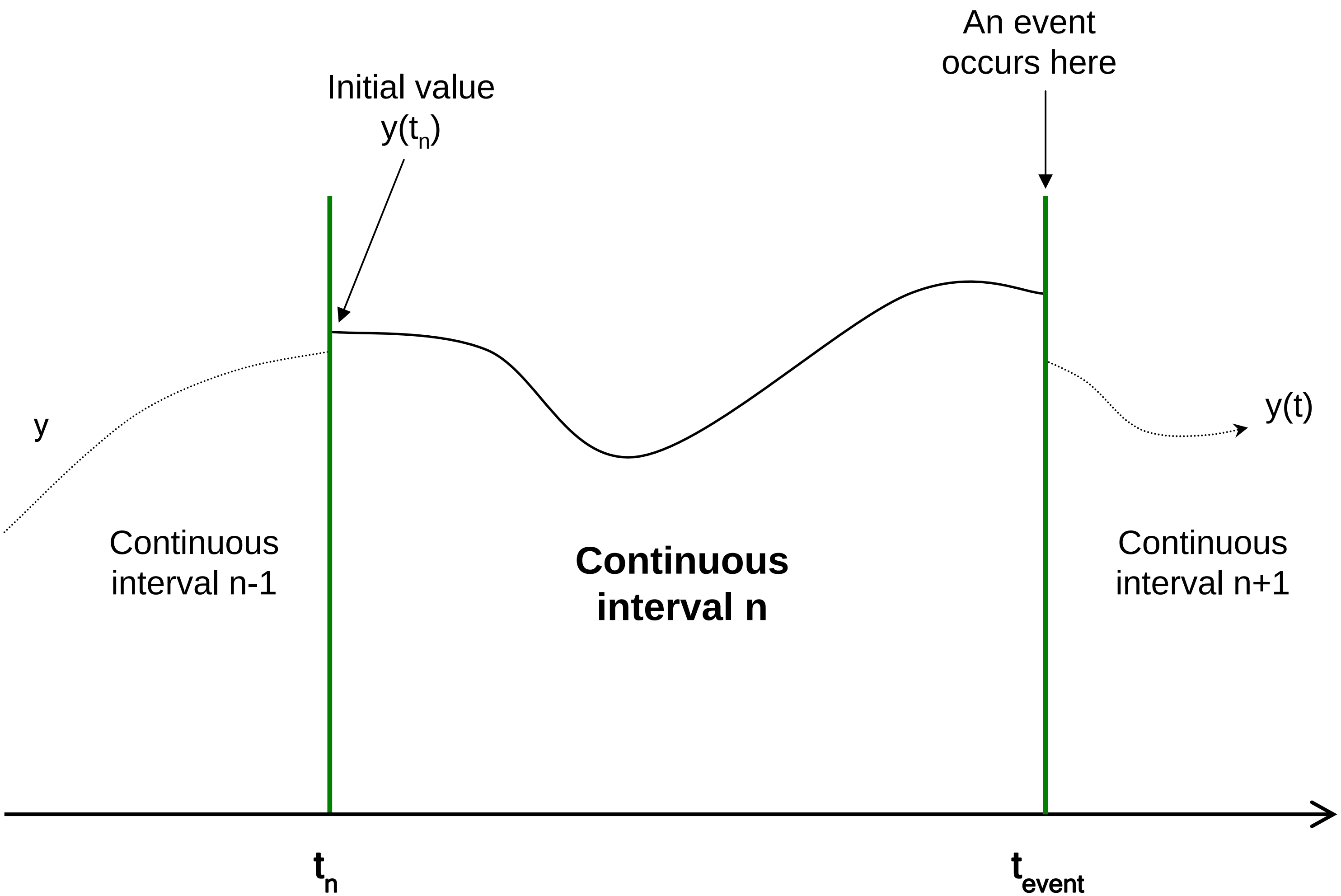
Hybrid Systems

- Arise from inherently discrete and continuous system elements
- And from discrete & continuous modeling choices
- So, interesting systems are often hybrid

Time Stepper

- Numerical method for advancing a hybrid dynamic system model through time
- Repeat until done:
  1. Advance continuous system until discrete update required
  2. Make discrete changes
  3. Set up new initial conditions for continuous system
- Uses a numerical integrator to advance continuous system

Time Stepping



Numerical Integrator

- “Inner loop” of time stepper
- Numerical method to advance *continuous* dynamic system model through time
- Time stepper provides initial state
- Job is twofold:
  1. Solve continuous equations for trajectory y(t)
  2. Detect when a discontinuous event occurs; return control to time stepper
- This is *much* harder than advancing the discrete system!

Integrators in SimTK

- RungeKuttaMersonIntegrator
  – Recommended for most mechanical systems
- VerletIntegrator
  – Recommended for most molecular simulations
- CPodesIntegrator
  – An implicit integrator (good for stiff systems)
  – Not very mature yet, still needs some work

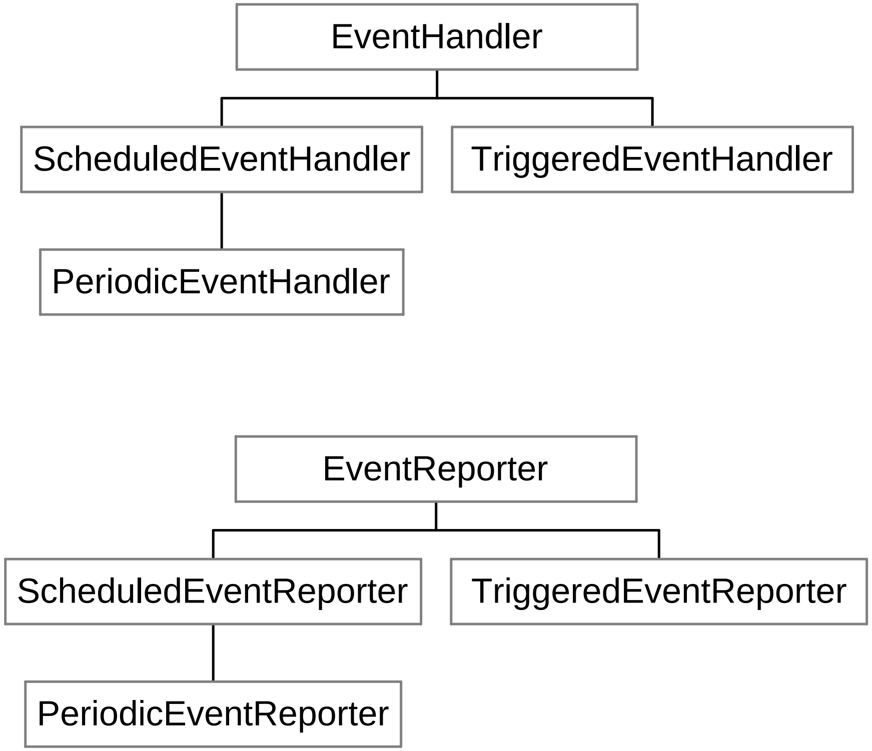
Events in SimTK

- An event has two pieces
  - A *trigger* to determine when it occurs
    - a function f(t,y) of the state variables
    - the event occurs when f(t,y) = 0
    - can be restricted to only rising transitions or only falling transitions
  - A *handler*
    - a piece of code that is invoked when the event occurs
    - may modify the state in discontinuous ways

Special Cases of Events

- Scheduled events
  - The event occurs at specific times that are known in advance, e.g. $f(t,y) = t\text{-}t_0$
  - These can be handled more efficiently than general event trigger functions
- Reports
  - The handler does not modify the state
  - Used to report information about intermediate states during a simulation

Class Hierarchy



An Example

(ExampleEventReporter.cpp)

```
class PositionReporter : public PeriodicEventReporter {
public:
    PositionReporter(const MultibodySystem& system, const MobilizedBody& body, Real interval) :
        PeriodicEventReporter(interval), system(system), body(body) {
    }
    void handleEvent(const State& state) const {
        system.realize(state, Stage::Position);
        Vec3 pos = body.getBodyOriginLocation(state);
        std::cout<<state.getTime()<<"\t"<<pos[0]<<"\t"<<pos[1]<<std::endl;
    }
private:
    const MultibodySystem& system;
    const MobilizedBody& body;
};

...

system.updDefaultSubsystem().addEventReporter(new PositionReporter(system, pendulum, 0.1));
```

The Realization Cache

- A state stores
  - State variables (coordinates, speeds, etc.)
  - Calculated values (positions, forces, etc.)
- Calculated values are stored in the *realization cache*
- Calculating these values is known as “realizing the state”

Cache Stages

- Realization must happen in a particular order
  - e.g. first positions, then forces, then accelerations
- Calculations can be expensive
  - Don’t waste time calculating forces if you only want positions
- Solution: realize the cache in *stages*

Stage	Available Information
1. Empty 2. Topology 3. Model 4. Instance	(Construction and initialization of the System and State)
5. Time	All state variables
6. Position	Cartesian positions of bodies
7. Velocity	Cartesian velocities of bodies, constraint errors
8. Dynamics	Forces, kinetic and potential energy
9. Acceleration	Time derivatives of state variables, event trigger functions
10. Report	System-specific values not needed for time integration

Realizing the State

- Always realize the State before requesting information
  - e.g. system.realize(state, Stage::Position)
  - If the State is already realized, realize() returns immediately
  - Asking for information not available at the current stage produces an exception
- Modifying a state variable automatically invalidates later cache stages

A Triggered Event Handler

(ExampleEventHandler.cpp)

```
class BounceHandler : public TriggeredEventHandler {
public:
    BounceHandler() : TriggeredEventHandler(Stage::Position) {
        getTriggerInfo().setTriggerOnRisingSignTransition(false);
    }
    Real getValue(const State& state) const {
        return state.getQ()[0];
    }
    void handleEvent(State& state, Real accuracy, const Vector& yWeights, const Vector& ooConstraintTols,
            Stage& lowestModified, bool& shouldTerminate) const {
        state.updU()[0] *= -1;
        lowestModified = Stage::Velocity;
    }
};

...

system.updDefaultSubsystem().addEventHandler(new BounceHandler());
```

Exercises

- Write an event handler that doubles the speed of the pendulum at time t=10
- Write an event reporter that prints the time whenever the pendulum reaches the end of its swing to the left