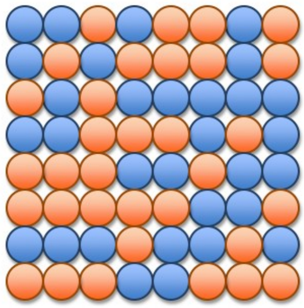


UNIT 12B
Continuous-Time Simulations

Announcement (Again)

- Exam 3 has been moved to Wednesday, November 28.

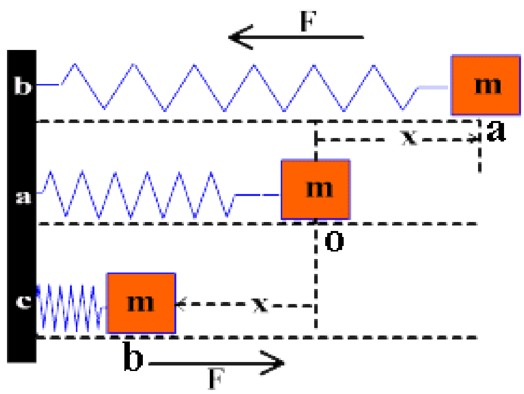
Why Do Simulations?

- To predict the behavior of a system.
  - Will this building survive an earthquake?
- To test a theory against data.
  - Do the predictions generated by these equations match what we observe in the real world?
- To explore consequences of assumptions.
  - What could you do with a Portal gun?

Continuous-Time Simulations

- Often used to model physical phenomena involving forces acting on objects.
- Is “time” really continuous?
  - Philosophical question. No one knows.
  - Just pretend it is.
- Is simulated time continuous?
  - No. It’s divided into discrete time steps.
  - But they can be as small as we like.

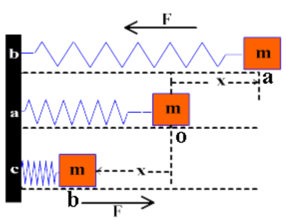
Example: A Spring Mass



Behavior of a Spring Mass

- Newton: F = ma

  Force = mass × acceleration



- Hooke's law: F = -kx

  F = force the spring applies to the mass

  k = "spring constant": kg/sec^2

  x = displacement from neutral point

Modeling the Spring Mass

- Use a variable x to model position of the mass.
- For convenience, assume x=0 at the neutral point.
- Since position x varies over time, it's actually a function x(t).
  - It's mathematically a function.
  - It doesn't have to be a function in Ruby.
  - We'll just use x and let the (t) be implicit.

7

Initial Conditions

- Let's define x(0) as the initial displacement of the spring relative to the neutral point.
  - In Ruby we'll use the variable x0.
- Let's assume that the mass starts out motionless, i.e., its initial velocity and acceleration are 0.
  - We'll relax this assumption later.

8

The Spring Force

- At any time t, the mass feels a force imposed by the spring: $F(t) = -k \cdot x(t)$
- The force causes the mass to accelerate. How?

  $F(t) = m \cdot a(t) = -k \cdot x(t)$
- Solve for the acceleration:

  $a(t) = F(t)/m = -k \cdot x(t)/m$

9

Integrating Acceleration

- When an object accelerates, its velocity v(t) changes. How can we model this?
- Divide time into tiny steps Δt.
- Re-calculate the velocity at each time step.

  $v(t+\Delta t) = v(t) + a(t) \cdot \Delta t$
- Smaller Δt brings greater accuracy.

10

Velocity Is Rate of Change of Position

- If an object has non-zero velocity, its position is changing.
- We can use the same integration trick to update the mass's position based on velocity.

  $x(t+\Delta t) = x(t) + v(t) \cdot \Delta t$

- Notice that when x changes, the spring force -kx will change, so acceleration will change.

11

Setting Up Our Simulation

```
m = 1      # mass = 1 kg
k = 1      # spring constant = 1 kg/s^2
x0 = 75    # initial displacement in mm
v = 0      # velocity
a = 0      # acceleration

dt = 0.0005     # time step for integration
```

12

The Simulation Loop

```
t = 0
x = x0
while true do
    a = -k * x / m       # accel. proportional to displacement
    v = v + a * dt       # velocity is changed by acceleration
    x = x + v * dt       # position is changed by velocity
    t = t + dt           # time marches on
    puts [t, x]
end
```

13

Graphics Help Us Understand Our Simulations

- Make a canvas:
  Canvas.init(400,400,"spring")
- Make a rectangle:
  r = Canvas::Rectangle.new(200,200,250,250)
- Make it move 10 pixels to the right:
  Canvas.move(r, 10, 0)

14

Initialize Our Graphics

```
def spring
  m = 1; k = 1; x0 = 75; v = 0; a = 0; dt = 0.005

  Canvas.init(400,400,"spring")
  Canvas::Rectangle.new(200,200,210,210,
                        :outline => "red")
  r = Canvas::Rectangle.new(200+x0,200,210+x0,210)

  x = x0
  t = 0
```

15

Loop With Graphics

```
  while t < 40 do
      a = -k * x / m
      v = v + a * dt
      x = x + v * dt
      t = t + dt
      Canvas.move(r, v*dt, 0)
      sleep(0.0001)    # force graphics to redraw
  end

end
```

16

Parameterizing The Simulation

```
def spring(*opts)
   opts = (opts[0] or {})
   x0 = (opts[:x0] or 75)
   m = (opts[:m] or 1)
   k = (opts[:k] or 1)
   dt = (opts[:dt] or 0.0005)
   v = (opts[:v0] or 0)
   maxtime = (opts[:maxtime] or 40)
```

17

Experiments

- What happens if we increase displacement?
  spring(:x0 => 75)
  spring(:x0 => 150)

- What happens if we increase the mass?
  spring(:mass => 1)
  spring(:mass => 2)
  spring(:mass => 5)

18

Simulating Gravitational Attraction

Newton's law of universal gravitation:

$$F = G \cdot m_1 \cdot m_2 / d^2$$

where G = gravitational constant,
m_1 and m_2 are the masses, and
d is the distance between them.

Since F = ma we can calculate the acceleration of each object.

N-Body Simulations

- With just two bodies, we can write a simple formula to calculate their positions at any future time, given their starting positions.
- But with 3 or more bodies, no formula exists for this, because the system is highly nonlinear, and potentially chaotic.
- Our only recourse is simulation.

Gravity Simulation In 2 Dimensions

```
include SphereLab
b = make_system(:fdemo)
view_system(b, :pendown => :track)
f1 = b[0]; f2 = f1.clone; f3 = f1.clone

500.times{update_one(f1, b[1..5], 1.0)}
f2.position.x += 1
500.times{update_one(f2, b[1..5], 1.0)}
```

21

Simulating The Solar System

```
include SphereLab
b = make_system(:solarsystem)
view_system(b[0..4], :dash => 1)
365.times {
  update_system(b, 86459); sleep(0.1) }
```

Notice that the orbits are elliptical (Kepler).

22

Simulation At Extreme Scales

- Cosmologists use simulations to study the formation of galaxies (clusters of stars), and even clusters of galaxies.
- At the other extreme, physicists simulate individual atoms and molecules, e.g., to model chemical reactions.