

Continuous Systems

Simulation and Modeling (CSCI 3010U)

Faisal Z. Qureshi

<http://vclab.science.ontariotechu.ca>



Continuous systems simulation

- ▶ Time is treated as a continuous variable that drives the simulation
- ▶ Model is based upon differential equations, which describe how systems evolves over time, and how it responds to changes in input variables
- ▶ In this course, we will mostly deal with Ordinary Differential Equations (ODE), though Partial Differential Equations are used in some cases

Continuous system simulation: Variables

- ▶ Three set of variables:
 - ▶ State variables;
 - ▶ Input variables; and
 - ▶ Output variables.
- ▶ There is *no* overlap between input and state variables
- ▶ **Input variable** are not controlled by the simulation
 - ▶ The angle of the steering wheel in the car simulator example
- ▶ **Output variable** are things that we observe
 - ▶ The speed of the vehicle in the car simulator example
- ▶ **State Variables** are controlled by the differential equations
 - ▶ The speed of the vehicle in the car simulator example
 - ▶ The location of the vehicle in the car simulator example

Ordinary Differential Equations (ODEs)

An ODE consists of the following ingredients:

- ▶ An independent variable (usually “time” t that derivatives are taken with respect to
- ▶ A dependent variable, i.e. function of the independent variable, e.g. $x = x(t)$ “the variable x which is a function of t ”.
- ▶ A multi-variable function F that describes a relationship between the derivatives of the dependent variable (taken with respect to the independent variable)

Putting it all together, we get

$$F \left(t, x, \frac{dx}{dt}, \frac{d^2x}{dt^2}, \dots, \frac{d^{(n-1)}x}{dt^{(n-1)}} \right) = \frac{d^n x}{dt^n}$$

ODE: comments

- ▶ $\frac{dx}{dt}$ denotes derivative of x w.r.t. t
- ▶ $\frac{d^n x}{dt^n}$ denotes n -th derivative of x w.r.t. t
- ▶ $x' = \frac{dx}{dt}$
- ▶ $Dx = \frac{dx}{dt}$
- ▶ Order of a differential equation is the highest order of derivative in that equation

Examples

- ▶ $mx'' = F$ (order is 2)
- ▶ $x' + 32x'' + x''' = 0$ (order is 3)
- ▶ $x' + 34x = 32$ (order is 1)

Solving differential equations

- ▶ Solution is the dependent variable $x = x(t)$ that satisfies the equation
- ▶ Key idea: **integration**

Example: $x'' = 2$

1. Integrate once: $x' = 2t + C_1$
2. Integrate again: $x = t^2 + tC_1 + C_2$ (Solution)

How to solve for C_1 and C_2 , also called, constants of integration?

- ▶ Use **initial** or **boundary** conditions.
- ▶ Using initial conditions $x'(0) = 3$ and $x(0) = 2$, we get $C_1=3$ and $C_2 = 2$. The solution is $x(t) = t^2 + 3t + 2$.

Aside: Constant of Integration

Notice that

$$\frac{df(x)}{dx} = 3x^2$$

for both when $f(x) = x^3$ or $f(x)x^3 + C$.

This suggests constant C disappears in the process of differentiation.

Therefore, when we integrate we add the constant C for the sake of completeness.

$$\int 3x^2 dx = 3 \left(\frac{x^3}{3} \right) + C = x^3 + C$$

Furthermore, we will need other information to find the true value of C . Note also that there is nothing preventing $C = 0$.

Nth order ODEs

- ▶ General solution to an n th order ODE will contain n constants of integration
- ▶ We need n more equations
 - ▶ Use initial or boundary conditions to get these equations to solve for the constants of integration
- ▶ Initial conditions
 - ▶ The values of $x(t)$ and its first $n - 1$ derivatives for a particular value of t
 - ▶ If such values are only available at the end, run time backwards to convert problem to initial conditions
- ▶ Boundary conditions
 - ▶ The values of $x(t)$ and its derivatives for two different values of t

Nth order ODE reducibility

Any explicit differential equation of order n

$$x^{(n)} = F(t, x, x', \dots, x^{(n-1)})$$

can be written as a system of n first-order differential equations by defining a new family of unknown functions

$$x^{(i-1)} = x_i$$

Notice the abuse of notation. Here $x^{(k)}$ denote the k -th derivative of x w.r.t. t .

Nth order ODE reducibility

We can then represent the following n -th order ODE

$$x^{(n)} = F(t, x, x', \dots, x^{(n-1)})$$

with n first-order ODEs as follows

$$x'_1 = x_2$$

$$x'_2 = x_3$$

$$\vdots$$

$$x'_{(n-1)} = x_n$$

$$x'_n = F(t, x_1, x_2, \dots, x_n)$$

Nth order ODE reducibility

Example

The following 2nd order ODE

$$\frac{d^2x}{dt^2} = -g$$

can be reduced to

$$\frac{dx}{dt} = v$$

$$\frac{dv}{dt} = -g$$

We introduced a new variable v .

First order ODEs

- ▶ All of our simulations only involve first order ODEs
- ▶ What about models that involve higher order ODEs?
 - ▶ E.g., the equation of motion for particle is modelled by a second order differential equation
 - ▶ Applies reducibility, i.e., replace a higher order differential equation by a system of first order differential equations
 - ▶ We can replace an n th order ODE with n first order ODE
- ▶ **Advantages of first order ODEs**
 - ▶ First order equations are much easier to solve numerically
 - ▶ Very few numerical solvers available for higher order equations

Equation of motion

Newton's second law of motion:

"The acceleration of an object as produced by a net force is directly proportional to the magnitude of the net force, in the same direction as the net force, and inversely proportional to the mass of the object."

Mathematically:

$a \propto F$ and $a \propto \frac{1}{m}$, and combining the two we get $F = ma$. Recall $a = \frac{d^2x}{dt^2}$, so $F = ma$ is a second order equation.

$$F = ma$$
$$\implies F = m \frac{d^2x}{dt^2}$$

Equation of motion

We introduce a new variable velocity $v = \frac{dx}{dt}$, and get the following first order system of equations

$$v = \frac{dx}{dt}$$
$$F = m \frac{dv}{dt}$$

Solving ODEs numerically

General idea: given a solution $x(t)$ at time $t = t_0$, incrementally step forward in time to find $x(t + \Delta t)$

Example: lets consider the equation of motion

$$F = m \frac{dv}{dt} \implies \Delta v = \left(\frac{F}{m} \right) (\Delta t)$$

$$v = \frac{dx}{dt} \implies \Delta x = (v)(\Delta t)$$

We can use Δv and Δx to **update** the **current** values of v and x , respectively.

Solving ODEs numerically: an example

What is the value of x at time $t = 3$?

Model

$$v(t + \Delta t) = v(t) + \Delta v = v(t) + (F/m)(\Delta t)$$

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

Setup

- ▶ $m = 1$, $F = 1$ (other quantities used in the simulation)
- ▶ $v(0) = 0$, $x(0) = 0$ (initial state), $\Delta t = 1$ (time step)

Solution

- ▶ $v(1) = 1$
 - ▶ $x(1) = 1$
- ▶ $v(2) = 2$
 - ▶ $x(2) = 3$
- ▶ $v(3) = 3$
 - ▶ $x(3) = 6$

Solving ODEs numerically: practical considerations

- ▶ Have to choose Δt carefully
 - ▶ Δt too small; the simulation can become very slow
 - ▶ Δt too large; the simulation can become very inaccurate
 - ▶ Advanced techniques can change Δt when solving equations to maintain acceptable accuracy and speed
- ▶ Δt determines the exact points in time for which we have the solution
 - ▶ What if we want solution at other points in time?
 - ▶ This places constraints on how we solve the equations
- ▶ Often times Δt used for solving ODEs is much smaller than the one used to update the display

Choice of ΔT

- ▶ Molecular activity (fraction of a millisecond)
- ▶ Evolution of an ecosystem (months or years)
- ▶ Galaxy formation (millions or billions of years)

Simulation loop

- ▶ Advance the simulation
- ▶ Display current results
- ▶ Get the user response
- ▶ Repeat

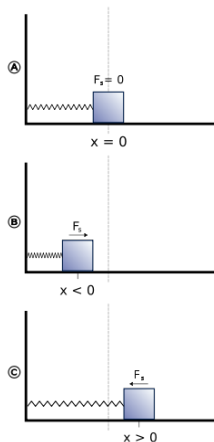
Displaying results

- ▶ Real-time or not?
- ▶ Timescale
- ▶ Response and interactivity
- ▶ Refresh rates

Mass spring system

Hook's law (1676) states, "the extension is proportional to the force."

Mathematically, $F = -kx$, where k is the spring constant and x is the displacement of the spring from rest position under the application of force F .



Mass spring system

Step 1: construct a model that will describe the motion of the mass over time

Hook's law: $F = -kx$

Newton's Second Law of Motion: $F = ma$

Combining the two we get

$$\begin{aligned} ma &= -kx \\ \implies m \frac{dx^2}{dt^2} &= -kx \end{aligned}$$

Mass spring system

Step2: find a way to solve the model numerically

Convert the second order $m \frac{dx^2}{dt^2} = -kx$ to a system of first order equations

$$\begin{aligned}\frac{dx}{dt} &= v \\ m \frac{dv}{dt} &= -kx\end{aligned}$$

And make the update rules

$$\begin{aligned}x(t + \Delta t) &= x(t) + v(t)\Delta t \\ v(t + \Delta t) &= v(t) - \frac{k}{m}x(t)\Delta t\end{aligned}$$

Mass spring system

- ▶ Set values for mass m and spring constant k
- ▶ Set the initial conditions
 - ▶ Values for x and v at start time t_0
- ▶ Run the simulation loop
 1. Update t to $t + \Delta t$
 2. Update values for x and v using the update rules
 3. Display results or save them to file for plotting
 4. Repeat steps 1 to 4

We just simulated a *Simple Harmonic Oscillator*

Code: 1d-mass-spring

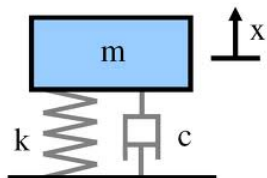
Mass spring system

```
# Mass-Spring system
class Mass:
    def __init__(self):
        self.x = 5
        self.vx = 0
        self.k = 1
        self.dt = 0.1
        self.t = 0
        self.m = 1.0

    def update(self):
        self.x += (self.vx * self.dt)
        self.vx += (- self.k * self.x * self.dt / self.m)
        self.t += self.dt
```

Mass Spring Damper

The mass experiences a damping force that is proportional to its current velocity



Mathematically

$$F = -kx - cv$$

where c is the damping constant

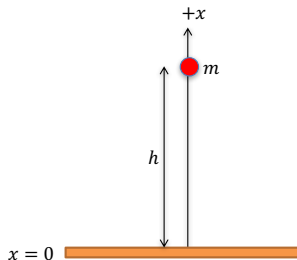
Code: modify 1d-mass-spring to add damping effect

Bouncing ball

Assumption 1: We simplify the problem by treating the ball as a particle
From Newton's Second Law of Motion

$$F = m \frac{dx^2}{dt^2}$$

where x is the height of the ball from the ground and m is the mass of the ball.



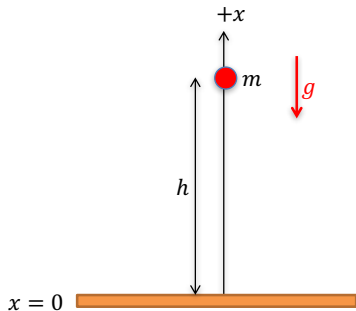
Bouncing ball

Assumption 2 Gravity is the only force acting upon this ball then

$$F = -mg$$

Putting it together we get

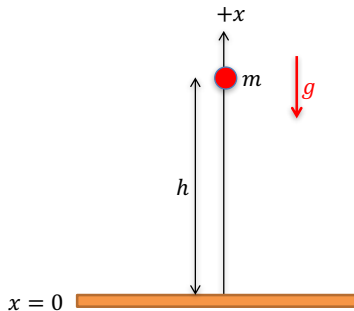
$$\frac{dx^2}{dt^2} = -g$$



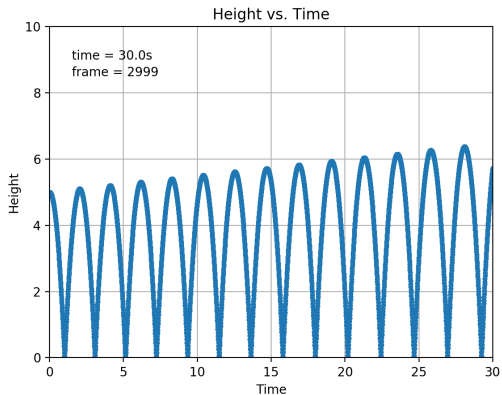
Bouncing ball

Data collection

- ▶ We need to know the value of g . For our purposes, we use $g = 9.8m/s^2$
- ▶ By using different g we can simulate bouncing ball on different planets



Bouncing ball



Did you notice something peculiar with this plot?

Code: ball-floor turn off RK4

Bouncing ball

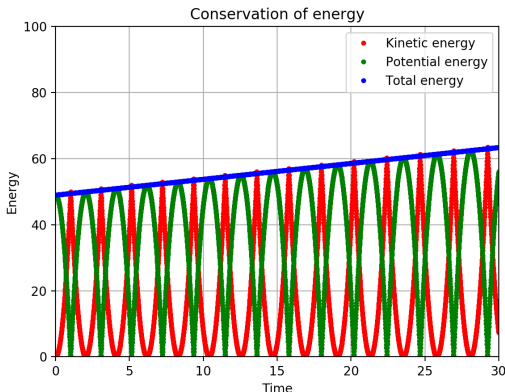
- ▶ The ball goes higher with each bounce, which is unexpected.
- ▶ The error doesn't go away even if we make timestep really small. It does, however, minimize the effect.
- ▶ It seems we are imparting energy to the ball with each bounce. This breaks the *the law of conservation of energy*, which states that “the total energy of an isolated system remains constant.”

Bouncing ball total energy

Total energy of the ball is the sum of its *kinetic* and *potential* energies.

$$\text{Kinetic energy} = \frac{1}{2}mv^2$$

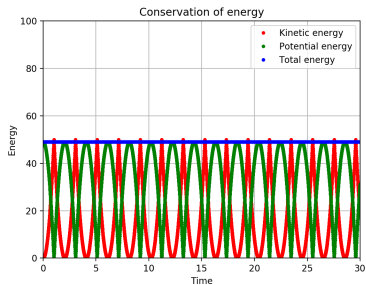
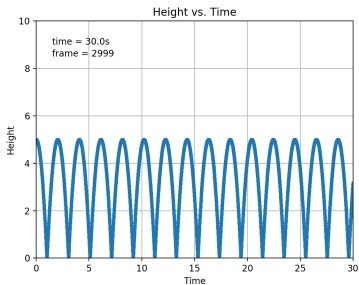
$$\text{Potential energy} = mgy$$



This behavior is due to incorrect assumptions of Euler method.

Bouncing Ball

Total energy is conserved when using Runga-Kutta or RK4 solver.



Code: ball-floor turn on RK4

Euler method

- ▶ A numerical solver for first order ODEs
- ▶ First order numerical procedure for solving ODEs (initial value problems)
- ▶ It is an *explicit method*
 - ▶ Calculates the state of the system at a later time given its current state by using the *update equations*
 - ▶ $y(t + \Delta t) = F(y(t))$

Aside: implicit methods

- ▶ Calculates the state of the system at a later time given by solving an equation that includes both the future state and the current state
- ▶ $G(y(t + \Delta t), y(t)) = 0$

Euler method

- ▶ Numerically unstable
 - ▶ Adversally effects accuracy
 - ▶ Exhibits error growth over time
 - ▶ Error is proportional to Δt
- ▶ Particularly unsuited for stiff equations
 - ▶ Equations containing terms that lead to rapid changes
 - ▶ E.g., a mass spring system with large spring constant
- ▶ Use *extremely* small time steps
 - ▶ Infeasible in practice

Runga-Kutta method

- ▶ An other numerical solver for first order ODEs
- ▶ An alternate to Euler method
- ▶ A family of explicit and implicit methods
- ▶ Often RK4 is used
 - ▶ Error is proportional to Δt^4
 - ▶ Makes a huge difference for small values of Δt

Takeaway: whenever possible use RK4 method

Numerical solvers in Python

```
from scipy.integrate import ode
def f(self, t, y, arg1):
    """Solves  $y' = f(t, y)$ 
    Arguments:
    - y is the state of the system. In our case
      y[0] is the position and y[1] is the velocity.
    - arg1 is 9.8, as set by set_f_params() method.

    Returns vector  $dy/dt$ . In our case,  $dx/dt = v$  and
     $dv/dt = -g$ .
    """
    return [y[1], -arg1]

r = ode(f).set_integrator('dop853')
r.set_initial_value([y0, vy0], t0)
r.set_f_params(9.8)
r.integrate(dt)
print r.t, r.y
```

Bouncing ball: takeaways

- ▶ Exploit your knowledge of physics to determine if simulation is behaving as expected
- ▶ Use several strategies
- ▶ Compare outputs of several strategies
 - ▶ If outputs differ, you must have a way to explain the differences
 - ▶ If outputs are the same, the simulation *may* be correct

Discussion

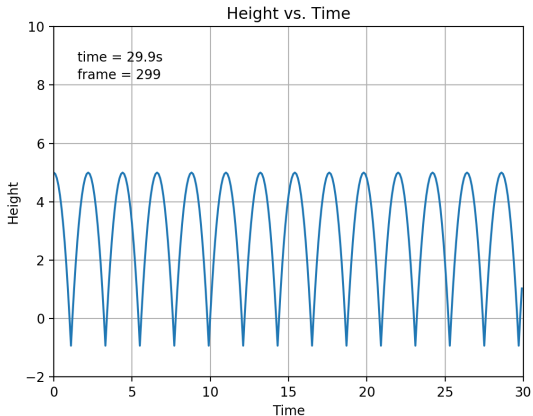
Q. Why does Euler method performing so poorly for our bouncing ball example?

A. Euler method assumes that the acceleration remains constant between two time steps. Notice that this assumption is generally false, but especially so when the ball “hits” the ground at $x = 0$. The velocity is flipped, changing the sign of the derivative and causing a discontinuity.

RK4 method is much better at handling discontinuities (as long as there aren't too many of these).

This is why RK4 is able to get good results even for large time steps.

Bouncing ball



For this simulation, the floor sits at height 0. **The ball pierces through the floor, which is incorrect.**

Code: `ball-floor` increase timestep to see the ball penetrating the floor

Bouncing ball

Need a better way to detect collisions with the floor

Scheme 1

- ▶ Use smaller time steps
- ▶ The ball will travel less distance between two time steps, and there is a greater chance of catching the collision instant
- ▶ In any case, the ball will penetrate less into the floor

Scheme 2

- ▶ Try to find the exact time of collision using $x = vt$ relationship
- ▶ Adjust time step accordingly

Bouncing ball

Collision detection

1. Approximate time to collide $t_c = \frac{x}{v}$
2. Set $x = 0$ and $v = -v(t + t_c)$
 - ▶ Flip v to indicate that the ball is now going back up again

Problem

v is larger than had we calculated t_c exactly right (that's because the particle is under constant acceleration). Consequently energy is not conserved.

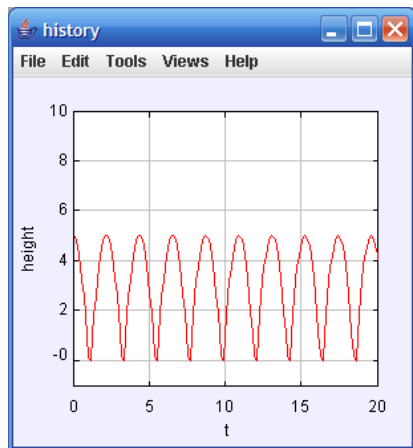
Bouncing ball

- ▶ Use the Law of Conservation of Energy to compute the velocity of the ball when it touches ground.
- ▶ The ball was released at height h . We know the total energy of the system, which is mgh . At the start the kinetic energy is 0.
- ▶ When the ball touches the ground, its potential energy reduces to 0. Since the total energy remains the same, all of its energy is now kinetic energy.

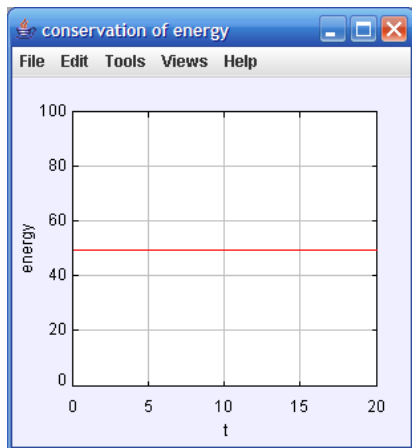
$$\frac{1}{2}mv^2 = mgh$$
$$v = \sqrt{2gh}$$

i.e., set $x = 0$ and $v = -\sqrt{2gh}$ at collision time.

Bouncing ball



Ball doesn't enter the floor



Energy is conserved

2D elastic band

Simulate a ball (point mass) attached to the origin via an elastic band (or a *spring sitting in a plane*).

- ▶ We assume that the rest length of the band is 0.
- ▶ Hook's law describes the relationship between the extension of the band and the force it applies on the attached ball

Hook's law in 1D

$$F = -kx,$$

where x is the displacement from the rest length (in this case 0), and k is the spring constant for the elastic band. F is the force on the ball.

2D elastic band

Option 1

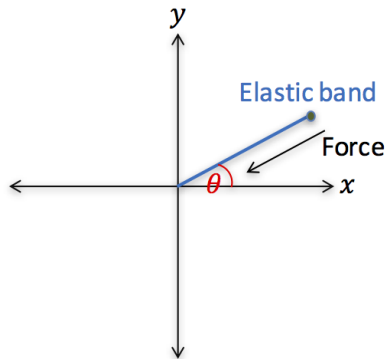
- ▶ Use Hook's Law in 2D

$$F = -k \left(\sqrt{x^2 + y^2} \right)$$

So

$$F_x = -k \left(\sqrt{x^2 + y^2} \right) \cos(\theta)$$

$$F_y = -k \left(\sqrt{x^2 + y^2} \right) \sin(\theta)$$



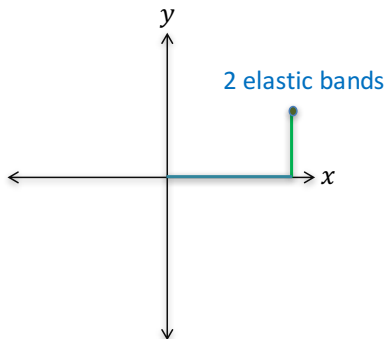
2D elastic band

Option 2

Replace 1 2D elastic band with 2 1D elastic bands. The first band sits along the x -axis; whereas, the second band sits along the y -axis.

$$F_x = -kx$$

$$F_y = -ky$$



2D elastic band

Add a damping force that is proportional to the velocity of the ball

Model

$$F_x = -kx - cv_x$$

$$F_y = -ky - cv_y$$

How many state variables?

5 = 2 for positions, 2 for velocity and 1 for time

Notice that we are consider t as a state variable as well. This is not exactly right, but it makes for easier implementations.

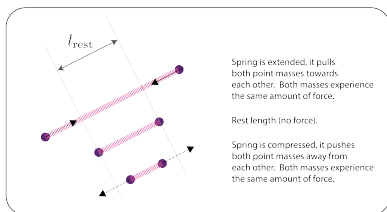
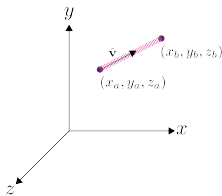
2D elastic band

Interaction

- ▶ User interacts with the ball by dragging it to a new location.
- ▶ Dragging to the new location changes the x and y extensions of the elastic band, effectively changing the forces acting on the ball.
- ▶ This is similar to grasping a real ball attached to a spring, and then letting go of the ball.

Mass-Spring systems in 3D

Consider a spring with rest length l and spring constant k . The spring is connected at two point masses located at $\mathbf{p}_a = (x_a, y_a, z_a)$ and $\mathbf{p}_b = (x_b, y_b, z_b)$, respectively. Our goal is to estimate the spring force exerted on these masses.



Spring Deformation and Axis

Spring axis vector:

$$\mathbf{v} = (x_b - x_a, y_b - y_a, z_b - z_a)$$

Current length of the spring:

$$l_{\text{current}} = \|\mathbf{v}\| = \sqrt{(x_b - x_a)^2 + (y_b - y_a)^2 + (z_b - z_a)^2}$$

Deformation:

$$d = l_{\text{current}} - l_{\text{rest}}$$

The deformation is positive if the spring is extended, and it is negative if the spring is compressed.

Hook's Law

Unit-vector along spring axis:

$$\hat{\mathbf{v}} = \frac{1}{l_{\text{current}}} (x_b - x_a, y_b - y_a, z_b - z_a)$$

Use the unit vector $\hat{\mathbf{v}}$ to compute force direction. Recall that this vector points towards \mathbf{p}_b . If the spring is extended, the mass at location \mathbf{p}_b will experience a force in the direction of the mass at location \mathbf{p}_a . Therefore, the spring exerts the following force on the mass at location \mathbf{p} :

$$f_{\text{on point mass at } \mathbf{p}_b} = -kd\hat{\mathbf{v}}$$

This expression also works when the spring is compressed. When the spring is compressed, d is negative. Therefore, the force on mass at \mathbf{p}_b is along $\hat{\mathbf{v}}$.

Hook's Law

Similarly, the force on the mass at location \mathbf{p}_a is

$$f_{\text{on point mass at } \mathbf{p}_a} = +kd\hat{\mathbf{v}}.$$

This is just the opposite of the force on mass at \mathbf{p}

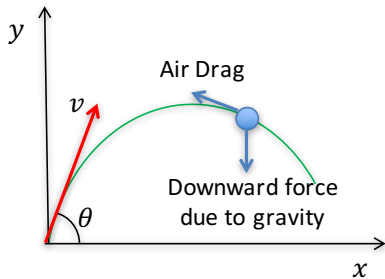
Projectile motion

$$mx'' = -\gamma x'$$

$$my'' = -\gamma y' - mg$$

Here γ is the friction constant, m is the mass of the particle, and g is the acceleration due to gravity.

This model doesn't take into account the effects of earth's gravitational field.



Projectile motion

Gravitational force between two masses M and m with is given by Newton's Law of Universal Gravitation

$$F = \frac{GM}{(R + y)^2},$$

where $R + y$ is the distance between their centres. G is the gravitational constant.

$$G = 6.674 \times 10^{-11} N(m/kg)^2$$

The value of g is merely a simplification given by

$$g = \frac{GM}{R^2}.$$

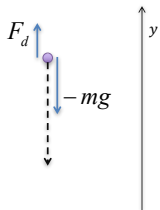
Projectile motion

To model a projectile near the surface of the earth we use

$$mx'' = -\gamma x'$$
$$my'' = -\gamma y' - \frac{GM}{(R + y)^2}$$

Unlike previous models that you have seen in this course, the above equations have no *analytical solution*. You'll have to solve them numerically.

Free-falling particle



- ▶ Force acting on a particle of mass m falling under gravity is

$$F = -mg + F_d,$$

where F_d is the drag force experienced by the particle as it moves through the air.

- ▶ F_d is a velocity dependent drag force. It increases with velocity and at some point, it will become equal to the mg , i.e., $F_d = mg$. The velocity at which this occurs is referred to the **terminal velocity** of the particle.
- ▶ Once terminal velocity is achieved the particle experiences 0 net force. The particle still continues to fall at a constant velocity. Why is that?

Free-falling particle

Terminal velocity

The velocity at which the motion of an object through a fluid is constant due to the drag force exerted by that fluid.

Terminal velocity depends upon both the particle and the medium through which it is moving.

Example: Falling pebble

Consider the fall of a pebble of mass 10^{-2} kg . The terminal velocity of this pebble is 30 m/s .

- ▶ How long will it take for this pebble to achieve terminal velocity?
- ▶ How much distance will this pebble cover before it achieves terminal velocity?

Observation: The pebble will cover around 50 m to achieve the terminal velocity. This will take around 3 s .

So if we are dealing with a pebble simulation across these distances (or times), we need to take into account terminal velocity.

Example: Falling pebble

Takeaway: even when modeling simple systems, such as a free falling particle, we need to carefully evaluate the conditions so as not to miss important effects.

Describing drag in terms of terminal velocity

- ▶ Linear drag

$$F_{1,d} = C_1 v = mg \frac{v}{v_{1,t}}$$

- ▶ Quadratic drag

$$F_{2,d} = C_2 v^2 = mg \left(\frac{v}{v_{1,t}} \right)^2$$

Modeling a falling coffee filter

Sketch

- ▶ Observe a falling coffee filter and record positions vs. times.
- ▶ Estimate velocities and accelerations via finite differences.
- ▶ Estimate terminal velocity. Recall that the object falls with constant velocity once terminal velocity is achieved.
- ▶ Identify the relationship between acceleration and velocity. Is it linear or quadratic?
- ▶ Right down the equations taking into account your findings.
- ▶ Run the simulation and see if it matches your observations.

Modeling a falling coffee filter

- ▶ Observe a falling coffee filter and record positions vs. times.
- ▶ Estimate velocities and accelerations via finite differences.
- ▶ Estimate terminal velocity. Recall that the object falls with constant velocity once terminal velocity is achieved.
- ▶ Identify the relationship between acceleration and velocity. Is it linear or quadratic?
- ▶ Right down the equations taking into account your findings.
- ▶ Run the simulation and see if it matches your observations.

```
//Falling coffee filter
//Time ( s )      Position ( m )
0.2055 0.4188
0.2302 0.4164
0.255 0.4128
0.2797 0.4082
0.3045 0.4026
0.3292 0.3958
0.3539 0.3878
0.3786 0.3802
0.4033 0.3708
0.428 0.3609
0.4526 0.3505
0.4773 0.34
0.502 0.3297
0.5266 0.3181
0.5513 0.3051
0.5759 0.2913
0.6005 0.2788
0.6252 0.2667
0.6498 0.2497
0.6744 0.2337
0.699 0.2175
0.7236 0.2008
0.7482 0.1846
0.7728 0.1696
0.7974 0.1566
0.822 0.1393
0.8466 0.1263
```

Modeling a falling coffee filter

Takeaway: it is sometimes possible to infer dynamics from empirical data

Simulating multiple objects

- ▶ So far we have simulated single objects
- ▶ Now we discuss how to simulate multiple objects?
 - ▶ The number of objects is a parameter for the simulation.

Simulating a collection of balls in a square

- ▶ Balls move in 2D
- ▶ Random initial positions and velocities
- ▶ Balls move under the influence of gravity
- ▶ Balls bounce off the walls
- ▶ Balls pass through each other (i.e., no collisions between balls)
- ▶ No friction

Simulating a collection of balls in a square

Question 1

Say we are interested in simulating n balls in a square. What is the state size of our simulation?

Answer 1

$4n$, (x, y) locations and (v_x, v_y) velocities for each ball.

Question 2

How do we set up the initial state for our simulation, i.e., the initial locations and initial velocities for each ball?

Answer 2

Random positions and velocities.

Simulating a collection of balls in a square

What are we missing in this simulation?

- ▶ Not handling collisions between balls
 - ▶ If only a few balls in a very large square, ball-ball collisions may be rare event.
 - ▶ If a lot of balls crammed in a small space, we can't really ignore ball-ball collisions.

Ball-ball collisions

- ▶ Ball-ball collisions are difficult to do efficiently.
 - ▶ Unlike ball-wall collisions, where only one object is moving, in ball-ball collision, both objects are moving.

Naive approach

- ▶ At each time step, inspect each pair for possible collision.
- ▶ For n balls this leads to n^2 inspections.

Other things to consider

What if three balls collide with each other at the same instant?

What if n balls collide with each other at the same instant?

Object-object collisions

- ▶ Efficient collisions between multiple objects is very challenging
- ▶ Most simulations only consider these when absolutely necessary
 - ▶ Gas molecules are small, so when simulating low-density gases in large volumes, inter-molecules collisions are sometimes ignored.

What other things have we ignored in our simulation containing multiple balls in a square?

Brainstorm

What other things have we ignored in our simulation containing multiple balls in a square?

- ▶ Ball-wall collisions ignore the effects of impact on the wall (and the balls)
 - ▶ If the balls were ball bearings, and the walls were made of thin aluminum then each collision would dent the wall.
 - ▶ The walls will get bent out of shape over time.
 - ▶ How would you model walls that bends overtime? This require some very complicated physics, large computational power, and sophisticated numerical techniques.
- ▶ We also didn't model the color of the balls
 - ▶ This would be of interest if we are intrested in light bounces or heat transfer.

Guidelines

- ▶ We need carefully identify what really needs to be modeled and simulated.
- ▶ We can make simulations arbitrarily complex by considering more things.
 - ▶ This makes it harder to produce simulations.
 - ▶ Simulations will be less efficient.
 - ▶ Simulations might become less useful.
- ▶ We need to know where to draw the line.

Correctly determining the applications of the simulation is an important first step in getting the model right

Summary

- ▶ Input, output and state variables
- ▶ Differential equations are used to model the behavior of state variables
- ▶ Numerical solvers for solving differential equations
 - ▶ Good numerical solvers really only exist for degree 1 differential equations.
- ▶ Transform higher order differential equations to multiple first-order differential equations.
 - ▶ This introduces extra state variables

Summary

- ▶ Use of indirect means to determine whether or not our simulation is correct.
 - ▶ We used our knowledge of the law of conservation of energy to identify the problem with our simulation
- ▶ Interactions
- ▶ Projectile motion
 - ▶ Our first encounter with an ODE that has no *analytical* solution
- ▶ Drag
 - ▶ Terminal velocity
- ▶ First exposure to inferring dynamics from empirical data

Summary

Simulating multiple objects

- ▶ How to model the system?
- ▶ How to manage state space?
- ▶ Performance
- ▶ Problem set up or initialization

Copyright and License

©Faisal Z. Qureshi



This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.