Discrete Dynamics

2. Force equals mass times acceleration (F=ma)

\[ a = \frac{F}{m} \]

\[ \frac{\Delta v}{\Delta t} = a \quad \Rightarrow \quad \Delta v = a \Delta t \quad \Rightarrow \quad v = \sum \Delta v = \sum a \Delta t \]

\[ \frac{\Delta x}{\Delta t} = v \quad \Rightarrow \quad \Delta x = v \Delta t \quad \Rightarrow \quad x = \sum \Delta x = \sum v \Delta t \]

These can be thought of as summing areas under curves.

Integrating the Physics Equations if Acceleration is Constant

\[ \Delta v = a \Delta t = \text{area} \]

What if Acceleration is not Constant?

What is \( v(t + \Delta t) \)?

The velocity is off by this much!

\[ v(t + \Delta t) = v(t) + \Delta v = v(t) + a \Delta t \]

This is close, but it is clearly not exactly right!

The problem is that we are treating all of the quantities as if they always have the value that they had at the start of the time step, even though they don’t.

This is known as a First Order solution.
What does a First Order Solution look like in a Program?

You need a way to hold the entire state of the system. This will be the input to our numerical integrator.

You also need a way to return the derivatives once you determine them.

```c
struct state State;

void GetDerivs( State, Derivatives )
{
  . . .
}
```

The inputs are the state, which consists of all variables necessary to completely describe the state of the physical system. The outputs are the derivatives of each state variable.

```c
struct derivatives Derivatives;

void AdvanceOneTimeStep( )
{
  GetDerivs( State, Derivatives ); // get derivatives
  State.x = State.x + Derivatives.vx * ∆t; // use derivatives
  State.vx = State.vx + Derivatives.ax * ∆t; // use derivatives
  State.t = State.t + ∆t;
}
```

The application, then, consists of:

```
Initialize( );
AdvanceOneTimeStep( );
Finish( );
```

What is v(t+∆t) ?

A Second Order solution is obtained by doing the First Order solution, determining all quantities at time = t and then averaging them with the quantities at time = t + ∆t and then treating them as constant throughout the interval.

The Runge-Kutta Fourth Order Solution

```
void AdvanceOneTimeStep( )
{
  GetDerivs( State, Derivatives1 );
  State2.t = State.t + ∆t/2.;
  State2.x = State.x +  Derivatives1.vx * (∆t/2.);
  State2.vx = State.vx + Derivatives1.ax * (∆t/2.);
  GetDerivs( State2, Derivatives2 );
  aavg = ( Derivatives1.ax + Derivatives2.ax ) / 2.;
  vavg = ( Derivatives1.vx + Derivatives2.vx ) / 2.;
  State.x = State.x + vavg * (∆t/2.);
  State.vx = State.vx + aavg * (∆t/2.);
  State.t = State.t + ∆t/2.;
}
```

The application, then, consists of:

```
Initialize( );
AdvanceOneTimeStep( );
Finish( );
```

Adapted from: http://en.wikipedia.org/wiki/Runge-Kutta

The Runge-Kutta Fourth Order Solution

```
void AdvanceOneTimeStep( )
{
  GetDerivs( State, Derivatives1 );
  State2.t = State.t + ∆t/2.;
  State2.x = State.x +  Derivatives1.vx * (∆t/2.);
  State2.vx = State.vx + Derivatives1.ax * (∆t/2.);
  GetDerivs( State2, Derivatives2 );
  aavg = ( Derivatives1.ax + Derivatives2.ax ) / 2.;
  vavg = ( Derivatives1.vx + Derivatives2.vx ) / 2.;
  State.x = State.x + vavg * ∆t;
  State.vx = State.vx + aavg * ∆t;
  State.t = State.t + ∆t;
}
```

The application, then, consists of:

```
Initialize( );
AdvanceOneTimeStep( );
Finish( );
```

Adapted from: http://en.wikipedia.org/wiki/Runge-Kutta
Solving Motion where there is a Spring

\[ F_{spring} = -ky \]

This is known as Hooke’s law

\[ \Delta v = \sum \frac{F}{m} \Delta t = \frac{-W - ky}{m} \Delta t \]

void GetDerivs( State, Derivatives )
{
    Derivatives.vy = State.vy;
    Derivatives.ay = (-W - K*State.y) / MASS;
}

Air Resistance Force

\[ F_{drag} = \frac{1}{2} \rho v^2 AC_d \]

Drag Coefficient

Terminal Velocity

\[ \Delta v = \frac{-W}{m} \left( 1 + \frac{1}{2} \rho v^2 AC_d \right) \Delta t \]

This is known as the terminal velocity.

The velocity becomes constant when \( \Delta v = 0 \):
\[ W \left( 1 - \frac{1}{2} \rho v^2 AC_d \right) = 0 \]
\[ v = \frac{2W}{\rho AC_d} \]

Some Drag Coefficients

<table>
<thead>
<tr>
<th>Item</th>
<th>( C_d )</th>
</tr>
</thead>
<tbody>
<tr>
<td>a smooth brick</td>
<td>0.1</td>
</tr>
<tr>
<td>a typical bicycle plus cyclist</td>
<td>0.9</td>
</tr>
<tr>
<td>rough sphere</td>
<td>0.4</td>
</tr>
<tr>
<td>smooth sphere</td>
<td>0.1</td>
</tr>
<tr>
<td>laminar flat plate</td>
<td>0.001</td>
</tr>
<tr>
<td>turbulent flat plate</td>
<td>0.005</td>
</tr>
<tr>
<td>bullet</td>
<td>0.295</td>
</tr>
<tr>
<td>person (upright position)</td>
<td>1.0-1.3</td>
</tr>
<tr>
<td>flat plate perpendicular to flow</td>
<td>1.28</td>
</tr>
<tr>
<td>skier</td>
<td>1.0-1.1</td>
</tr>
<tr>
<td>wires and cables</td>
<td>1.0-1.3</td>
</tr>
<tr>
<td>Empire State Building</td>
<td>1.0-2.0</td>
</tr>
<tr>
<td>Empire State Building</td>
<td>1.0-2.0</td>
</tr>
<tr>
<td>Eiffel Tower</td>
<td>1.8-2.0</td>
</tr>
</tbody>
</table>

Human Terminal Velocity

Assume:

\[ \text{Weight} = 200 \text{ pounds} = 890 \text{ Newtons} \]
\[ A = 6 \text{ ft}^2 = 0.558 \text{ m}^2 \]
\[ \rho_{Air} = 1.293 \text{ kg/m}^3 \]

\[ v_t = \frac{2W}{\rho AC_d} = 43.30 \text{ m/sec} = 98 \text{mph} \]
How about a Cliff Jumper on a Bungee Cord?

\[ F_{\text{spring}} = -ky \]
\[ F_{\text{drag}} = -\text{Sign}(v_y) \frac{1}{2} \rho v_y^2 C_d A \]
\[ \Delta \vec{V} = \sum_{m} \frac{\vec{F}}{m} \Delta t = \frac{-W - ky - \text{Sign}(v_y) \frac{1}{2} \rho v_y^2 C_d A}{m} \Delta t \]

Lift – Another Good Force to Know About

\[ F_{\text{lift}} = \frac{1}{2} \rho v^2 A C_l \]

Coefficient of Lift vs. Angle of Attack

Friction Force – Another Good Force to Know About

\[ F_{\text{friction}} = \mu N \]

Some Coefficients of Friction

<table>
<thead>
<tr>
<th>Materials</th>
<th>( \mu ) Dry</th>
<th>( \mu ) Clean</th>
<th>( \mu ) Lubricated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aluminum</td>
<td>0.65</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Copper</td>
<td>0.03</td>
<td>0.03</td>
<td></td>
</tr>
<tr>
<td>Brass</td>
<td>0.01</td>
<td>0.01</td>
<td></td>
</tr>
<tr>
<td>Cast Iron</td>
<td>0.15</td>
<td>0.05</td>
<td></td>
</tr>
<tr>
<td>Cement Block</td>
<td>0.05</td>
<td>0.05</td>
<td></td>
</tr>
<tr>
<td>Concrete</td>
<td>0.10</td>
<td>0.20</td>
<td></td>
</tr>
<tr>
<td>Copper</td>
<td>0.62</td>
<td>0.62</td>
<td></td>
</tr>
<tr>
<td>Glass</td>
<td>0.48</td>
<td>0.48</td>
<td></td>
</tr>
<tr>
<td>Stone</td>
<td>0.94</td>
<td>0.94</td>
<td></td>
</tr>
<tr>
<td>Metal</td>
<td>0.25–0.6</td>
<td>0.25–0.6</td>
<td></td>
</tr>
<tr>
<td>Polyethylene</td>
<td>0.2</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>Steel</td>
<td>0.88</td>
<td>0.70</td>
<td></td>
</tr>
<tr>
<td>Steel</td>
<td>0.04</td>
<td>0.04</td>
<td></td>
</tr>
<tr>
<td>Teflon</td>
<td>0.04</td>
<td>0.04</td>
<td></td>
</tr>
<tr>
<td>Wood</td>
<td>0.25–0.5</td>
<td>0.25–0.5</td>
<td></td>
</tr>
</tbody>
</table>

http://en.wikipedia.org/wiki/Lift_coefficient

http://en.wikipedia.org/wiki/Lift_%28force%29

http://en.wikipedia.org/wiki/Friction
Buoyancy – Another Good Force to Know About

Archimedes’ Principle says that the buoyancy force on an object in a fluid is the weight of the fluid that is being displaced by the object.

\[ \rho_{water} = 6.24 \times 10^{-2} \text{ pounds/in}^3 \]
\[ \rho_{air} = 0.076 \times 10^{-2} \text{ pounds/in}^3 \]

So, for a helium balloon that is one foot in diameter (i.e., radius=6 inches), it has its weight pulling it down and a buoyancy force pushing it up. The net force pushing it up because of the gas inside the balloon is:

\[ F_{buoyancy} = \rho_{air} V - \rho_{helium} V = \rho_{air}(V_{helium} - V_{air}) \]
\[ V_{helium} = \frac{4}{3} \pi r^3 = 904.78 \text{in}^3 \]
\[ F_{buoyancy} = 904.78 \text{in}^3 \left( 4.01 \times 10^{-2} \text{ pounds/in}^3 \right) = 0.036 \text{ pounds} \]

Note that this must still counterbalance the weight of the balloon material, or the balloon will not fly.

Spinning Motion:

\[ T = \frac{1}{2} I \alpha \]

\[ \alpha = \frac{\Delta \omega}{\Delta t} \]

\[ \Delta \omega = \omega_{M} = \frac{T}{I} \]

Densities

The state and derivative vectors now include angular components

```c
struct state {
    float t;
    float x;
    float vx;
    float theta;
    float omega;
};

struct derivatives {
    float vx;
    float ax;
    float omega;
    float alpha;
};
```

void GetDerivs( State, Derivatives )
{
    Derivatives.vx = State.vx;
    Derivatives.x = SomeOfAllForces / MASS;
    Derivatives.omega = State.omega;
    Derivatives.alpha = SomeOfAllTorques / INERTIA
}
```