Dynamic Physics for Simulation and Game Programming

Mike Bailey
mjb@cs.oregonstate.edu

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

\[ \Delta v = a \Delta t = \text{area} \]
What is $v(t+\Delta t)$?

The velocity is off by this much!

Accumulated $v$ so far

Acceleration at the start of the time step

$\Delta v = a \Delta t$

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

This is close, but is clearly not exactly right!
What is $v(t+\Delta t)$?

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

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;
{
    float time;
    float x;
    float vx;
};

struct derivatives Derivatives;
{
    float vx;
    float ax;
};

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.

\[
\frac{dx}{dt} \Rightarrow v_x
\]
\[
\frac{dv}{dt} \Rightarrow a_x
\]
What does a First Order Solution look like in a Program?

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

The application, then, consists of:

- Initialize( );
- AdvanceOneTimeStep( );
- Finish( );
What is $v(t+\Delta t)$?

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

1. $a(t) = \frac{F(t)}{m}$
2. $a(t + \Delta t) = \frac{F(t + \Delta t)}{m}$
3. $\Delta v = a\Delta t = a_{avg}\Delta t = \frac{a(t) + a(t + \Delta t)}{2}\Delta t$
4. $v(t + \Delta t) = v(t) + \Delta v$
What does a Second Order Solution look like in a Program?

```c
void AdvanceOneTimeStep( )
{
    GetDerivs( State, Derivatives1);
    State2.t  = State.t  + Δt;
    State2.x  = State.x  + Derivatives1.vx * Δt;
    State2.vx = State.vx + Derivatives1.ax * Δt;

    GetDerivs( State2, Derivatives2 );
    float aavg = ( Derivatives1.ax + Derivatives2.ax) / 2.;
    float vavg = ( Derivatives1.vx + Derivatives2.vx) / 2.;

    State.x  = State.x  + vavg * Δt;
    State.vx = State.vx + aavg * Δt;
    State.t  = State.t  + Δt ;
}
```

The application, then, consists of:

Initialize( );

```
  AdvanceOneTimeStep( );
```

Finish( );
The Runge-Kutta Fourth Order Solution

\[
\begin{align*}
\begin{cases}
  v_1 \\ a_1
\end{cases} & = \text{GetDerivs}(t, x, v) \\
\begin{cases}
  v_2 \\ a_2
\end{cases} & = \text{GetDerivs}(t + \frac{\Delta t}{2}, x + v_1 \frac{\Delta t}{2}, v + a_1 \frac{\Delta t}{2}) \\
\begin{cases}
  v_3 \\ a_3
\end{cases} & = \text{GetDerivs}(t + \frac{\Delta t}{2}, x + v_2 \frac{\Delta t}{2}, v + a_2 \frac{\Delta t}{2}) \\
\begin{cases}
  v_4 \\ a_4
\end{cases} & = \text{GetDerivs}(t + \Delta t, x + v_3 \Delta t, v + a_3 \Delta t)
\end{align*}
\]

\[
\begin{align*}
\begin{cases}
  x(t + \Delta t) \\ v(t + \Delta t)
\end{cases} & = \begin{cases}
  x \\ v
\end{cases} + \frac{\Delta t}{6} \left( \begin{cases}
  v_1 \\ a_1
\end{cases} + 2 \begin{cases}
  v_2 \\ a_2
\end{cases} + 2 \begin{cases}
  v_3 \\ a_3
\end{cases} + \begin{cases}
  v_4 \\ a_4
\end{cases} \right)
\end{align*}
\]

Adapted from: http://en.wikipedia.org/wiki/Runge-Kutta
The Runge-Kutta Fourth Order Solution

```c
void
AdvanceOneTimeStep( )
{
    GetDerivs( State, Derivatives1 );
    State2.t   = State.t + Δt/2.;
    State2.x   = State.x + Derivatives1.vx * (Δt/2.);
    State2.vx  = State.vx + Derivatives1.ax * (Δt/2.);

    GetDerivs( State2, Derivatives2 );
    State3.t   = State.t + Δt/2.;
    State3.x   = State.x + Derivatives2.vx * (Δt/2);
    State3.vx  = State.vx + Derivatives2.ax * (Δt/2.);

    GetDerivs( State3, Derivatives3 );
    State4.t   = State.t + Δt;
    State4.x   = State.x + Derivatives3.vx * Δt;
    State4.vx  = State.vx + Derivatives3.ax * Δt;

    GetDerivs( State4, Derivatives4);

    State.x   = State.x + (Δt/6.) * ( Derivatives1.vx + 2.*Derivatives2.vx + 2.*Derivatives3.vx + Derivatives4.vx );
    State.vx  = State.vx + (Δt/6.) * (Derivatives1.ax + 2.*Derivatives2.ax + 2.*Derivatives3.ax + Derivatives4.ax );
}```
Solving Motion where there is a Spring

\[ F_{spring} = -ky \]

This is known as Hooke's law

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

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

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

- \( F_{\text{drag}} \): Air Resistance Force
- \( \rho \): Fluid density
- \( v \): Y Velocity
- \( A \): Cross-sectional area
- \( C_d \): Drag Coefficient

Air Resistance always acts in a direction opposite to the velocity of the object.
## Some Drag Coefficients

<table>
<thead>
<tr>
<th>$C_d$</th>
<th>Item</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>a smooth brick</td>
</tr>
<tr>
<td>0.9</td>
<td>a typical bicycle plus cyclist</td>
</tr>
<tr>
<td>0.4</td>
<td>rough sphere</td>
</tr>
<tr>
<td>0.1</td>
<td>smooth sphere</td>
</tr>
<tr>
<td>0.001</td>
<td>laminar flat plate</td>
</tr>
<tr>
<td>0.005</td>
<td>turbulent flat plate</td>
</tr>
<tr>
<td>0.295</td>
<td>bullet</td>
</tr>
<tr>
<td>1.0-1.3</td>
<td>person (upright position)</td>
</tr>
<tr>
<td>1.28</td>
<td>flat plate perpendicular to flow</td>
</tr>
<tr>
<td>1.0-1.1</td>
<td>skier</td>
</tr>
<tr>
<td>1.0-1.3</td>
<td>wires and cables</td>
</tr>
<tr>
<td>1.3-1.5</td>
<td>Empire State Building</td>
</tr>
<tr>
<td>1.8-2.0</td>
<td>Eiffel Tower</td>
</tr>
</tbody>
</table>

Solving Motion where there is Air Resistance

\[
\Delta v = \frac{\sum F}{m} \Delta t = \frac{-W - \text{Sign}(v_y) \frac{1}{2} \rho v_y^2 AC_d}{m} \Delta t
\]

\[
\rho_{\text{air}} = 1.293 \ \frac{kg}{m^3}
\]

The \( \text{Sign}(\cdot) \) function returns +1 or -1, depending on the sign of argument.

```c
void GetDerivs( State, Derivatives )
{
    Derivatives.vy = State.vy;
    Derivatives.ay = (-W - .5*Sign(State.vy)*DENSITY* State.vy * State.vy * AREA*DRAG ) / MASS;
}
```
Terminal Velocity

When a body is in free fall, it is being accelerated by the force of gravity. However, as it accelerates, it is encountering more and more air resistance force. At some velocity, these two forces balance each other out and the velocity becomes constant, that is, $\Delta v=0$.

This is known as the **terminal velocity**.

\[
\Delta v = \frac{-W + \frac{1}{2} \rho v_y^2 A C_d}{m} \Delta t
\]

The velocity becomes constant when $\Delta v = 0$:

\[
W - \frac{1}{2} \rho v_y^2 A C_d = 0
\]

\[
v_t = \sqrt{\frac{2W}{\rho A C_d}}
\]
Human Terminal Velocity

Assume:

Weight = 200 pounds = 890 Newtons

\[ C_d = 1.28 \]

\[ A = 6 \text{ ft}^2 = 0.558 \text{ m}^2 \]

\[ \rho_{\text{air}} = 1.293 \frac{\text{kg}}{\text{m}^3} \]

\[
\nu_t = \sqrt{\frac{2W}{\rho AC_d}} = 43.90 \frac{m}{\text{sec}} \approx 98 \text{mph}
\]
How about a Cliff Jumper on a Bungee Cord?

\[ F_{spring} = -ky \]

\[ F_{drag} = -\text{Sign}(v_y) \frac{1}{2} \rho v_y^2 C_d A \]

\[ \Delta v = \sum_{m} F \Delta t = \frac{-W - ky - \text{Sign}(v_y) \frac{1}{2} \rho v_y^2 AC_d}{m} \Delta t \]

```c
void GetDerivs( State, Derivatives )
{
    Derivatives.vy = State.vy;
    Derivatives.ay = ( -W - K*State.y - .5*Sign(State.vy)*DENSITY*State.vy*State.vy*AREA*DRAG ) / MASS;
}
```
Coulomb Damping

This is very much like drag force, but it is the resistance of a fluid being squeezed through a small opening. The resisting force is proportional to the velocity:

\[ F_{damping} = -cv \]
Lift – Another Good Force to Know About

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

Coefficient of Lift, for a given angle of attack

Air density  Airspeed  Platform area

http://en.wikipedia.org/wiki/Lift_%28force%29
Coefficient of Lift vs. Angle of Attack

http://en.wikipedia.org/wiki/Lift_coefficient
Lift and Drag Dramatically Working Together – Flight of a Frisbee
Friction Force – Another Good Force to Know About

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

- **Normal force** (i.e., amount of force that is perpendicular to the surface)
- **Coefficient of Friction**

Diagram showing a body sliding down an inclined plane with forces labeled as \( \mu N \), \( N \), and \( W \).
### Some Coefficients of Friction

<table>
<thead>
<tr>
<th>Materials</th>
<th>μ</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Dry &amp; clean</td>
</tr>
<tr>
<td>Aluminum Steel</td>
<td>0.61</td>
</tr>
<tr>
<td>Copper Steel</td>
<td>0.53</td>
</tr>
<tr>
<td>Brass Steel</td>
<td>0.51</td>
</tr>
<tr>
<td>Cast iron Copper</td>
<td>1.05</td>
</tr>
<tr>
<td>Cast iron Zinc</td>
<td>0.85</td>
</tr>
<tr>
<td>Concrete (wet) Rubber</td>
<td>0.30</td>
</tr>
<tr>
<td>Concrete (dry) Rubber</td>
<td>1.0</td>
</tr>
<tr>
<td>Concrete Wood</td>
<td>0.62</td>
</tr>
<tr>
<td>Copper Glass</td>
<td>0.68</td>
</tr>
<tr>
<td>Glass Glass</td>
<td>0.94</td>
</tr>
<tr>
<td>Metal Wood</td>
<td>0.2–0.6</td>
</tr>
<tr>
<td>Polythene Steel</td>
<td>0.2</td>
</tr>
<tr>
<td>Steel Steel</td>
<td>0.80</td>
</tr>
<tr>
<td>Steel Teflon</td>
<td>0.04</td>
</tr>
<tr>
<td>Teflon Teflon</td>
<td>0.04</td>
</tr>
<tr>
<td>Wood Wood</td>
<td>0.25–0.5</td>
</tr>
</tbody>
</table>

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.

\[
\rho_{\text{air}} = 4.66 \times 10^{-5} \text{ pounds / in}^3 \\
\rho_{\text{helium}} = 0.65 \times 10^{-5} \text{ pounds / in}^3
\]  

Densities

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:

\[
F_{\text{buoyancy}} = V_{\text{balloon}} \rho_{\text{helium}} - V_{\text{balloon}} \rho_{\text{air}} = V_{\text{balloon}} (\rho_{\text{helium}} - \rho_{\text{air}})
\]

\[
V_{\text{balloon}} = \frac{4}{3} \pi r^3 = 904.78 \text{ in}^3
\]

\[
F_{\text{buoyancy}} = 904.78 \text{ in}^3 (4.01 \times 10^{-5} \text{ pounds / in}^3) = 0.036 \text{ pounds}
\]

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

\[ T = I \alpha \]

Moment of Inertia \( \approx \) an angular “mass”
(newton-meters-sec\(^2\)=kg-meters\(^3\))

Torque \( \approx \) an angular “force”
(newton-meters)

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

\[ \Delta \omega = \alpha \Delta t = \frac{T}{I} \Delta t \]
Spinning Motion:
What does this look like in a Program?

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

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

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

The state and derivative vectors now include angular components