A Simple Integration Scheme

Newton's second law of motion for a particle of mass m moving under the influence of a force $\mathbf{F}$ is

$$m\,\frac{d^2\mathbf{x}}{dt^2} \;=\; \mathbf{F}(\mathbf{x},\mathbf{v},t),$$

where $\mathbf{v} = d\mathbf{x}/dt$. This equation is easy to solve if $\mathbf{F}$ has some simple form—that's pretty much what Physics I was all about! In general, however, finding an analytic solution to such an equation is not at all an easy matter.

We have seen how to handle the case of uniformly accelerated motion. Now consider what happens if the acceleration is non-uniform, and possibly time-dependent too. There is, in general, no longer an analytic solution to the motion. But nature presents us with many such problems—projectile motion with a variable gravitational field or with air resistance, for example. In such cases we must resort to numerical computation of the trajectory.

The basic idea behind any numerical algorithm is exactly the same as we saw in the uniformly accelerated case. We break the motion into many small segments, or *time steps* and write down an algorithm, or integration scheme, to advance the system from time t_i to time t_{i+1}. Clearly, applying the scheme repeatedly will accomplish the desired goal of advancing the system arbitrarily far forward in time. The result of each step becomes the initial state for the next.

There is an enormous literature devoted to just this problem. However, one of the simplest schemes requires no knowledge beyond the equations for uniformly accelerated motion already presented.

Let's imagine that we describe the motion as a series of constant time steps, each of duration δt. The key (which also seems intuitively reasonable) is to choose δt sufficiently short that the force $\mathbf{F}$ may be regarded as *constant* during the interval. You can quickly convince yourself that, if $\mathbf{F}$ is sufficiently "well behaved"—differentiable, or even just continuous—that this requirement on $\mathbf{F}$ can always be met by choosing a sufficiently short timestep δt. The proper choice of time step is in fact a very subtle issue, often something of an art, but let's defer such problems until we have seen the basic integration scheme in action.

Having made the assumption that $\mathbf{F}$ may be regarded as constant, we can easily connect the positions and velocities at the start and end of the step. More precisely, if we let t_i $(= i\delta t)$, $\mathbf{x}_i = \mathbf{x}(t_i)$, and $\mathbf{v}_i = \mathbf{v}(t_i)$ represent the state of the system at the end if the i-th step, and we define $\mathbf{a}_i = \mathbf{F}(\mathbf{x}_i,\mathbf{v}_i,t_i)/m$, we can write

$$\begin{aligned}
\mathbf{x}_{i+1} &= \mathbf{x}_i + \mathbf{v}_i\delta t + \tfrac{1}{2}\mathbf{a}_i\delta t^2\,,\\
\mathbf{v}_{i+1} &= \mathbf{v}_i + \mathbf{a}_i\delta t\,,\\
t_{i+1} &= t_i + \delta t\,.
\end{aligned}$$

That's it!

Note that the map is *explicit*—the procedure for getting from state i to state $i+1$ depends only on quantities determinable at time t_i. At time t_{i+1}, a new $\mathbf{a}_{i+1} = \mathbf{F}(\mathbf{x}_{i+1},\mathbf{v}_{i+1},t_{i+1})/m$ is computed, and the process repeats. In this way the variation of the acceleration in time and/or in space is taken into account. There are many variations on this basic scheme that can greatly improve its performance, but we will stick with this one for now.

Let's move to a one-dimensional problem, for simplicity of notation. (The generalization to higher dimensions should be fairly obvious.) A schematic C++ program to carry out a numerical integration is as follows.

```
// (...assume that the function acc is already defined...)

main()
{
    real x, v, t, dt, t_max;

    // Initialization

    t = 0;
    x = ...;
    v = ...;

    // Parameters

    dt = ...;
    t_max = ...;

    while (t < t_max) {

        real a = acc(x, v, t);

        x = x + v*dt + 0.5*a*dt*dt;
        v = v + a*dt;
        t = t + dt;

    }
}
```

(Obviously, you will have to include the appropriate libraries, choose your own initial conditions, and provide statements to print out or otherwise use the results.) You are strongly encouraged to rewrite this program in more modular form, separating the initialization, termination test, and integration scheme into separate functions for ease of programming later. Ultimately, the top level of your program should take on the schematic form:

```
main()
{
    real x, v, t, dt, t_max;

    initialize(x, v, t, dt, t_max);

    while (!time_to_stop(x, v, t, t_max))
        step(x, v, t, dt);
}
```