Appendix A

RK2 Integration

Sometimes it is not adequately accurate to assume -- as was done in the 11/29/2012 version of the paper* -- that acceleration is constant during the integration time step dt.

This might happen if, for example, dt is too large or acceleration is a very strong function of velocity.

One simple way to fix this is to use a 2nd order integration method such as Runge-Kutta 2.

RK2, also known as the mid-point method, works by calculating an Euler step, then using half that step value to find the slope at the midpoint of the step. That midpoint slope is then used to calculate the full step.

Pseudo-code for this is shown on the following page.

* http://www.chiefdelphi.com/media/papers/download/3518

Pseudo-code for 2nd-order Runge-Kutta numerical integration

```
f(Vx,Vy) := -(sqrt(Vx^2+Vy^2)/M)*(ud*Vx+um*Vy)    // function for acceleration in the X direction

g(Vx,Vy) := +(sqrt(Vx^2+Vy^2)/M)*(um*Vx-ud*Vy)+g  // function for acceleration in the Y direction


// initialize t, x, y, Vx, Vy, ax=f(Vx,Vy), ay=g(Vx,Vy) here.
// dVx1, dVy1, dVx2, dVy2  are temporary variables.

while (y>=0) {

dVx1=ax*dt;  dVy1=ay*dt;

dVx2=dt*f(dVx1/2+Vx,dVy1/2+Vy);
dVy2=dt*g(dVx1/2+Vx,dVy1/2+Vy);

x+=dt*(dVx2/2+Vx);
y+=dt*(dVy2/2+Vy);

Vx+=dVx2;  Vy+=dVy2;

ax=f(Vx,Vy);  ay=g(Vx,Vy);

t+=dt;

// output t, x, & y here

}
```