Runga Kutta:

Created By:
Casey Cartwright
May 2015

```
ClearAll["Global`*"]
```

The essential idea in runga kutta schemes is to sample the system at multiple inervals, rather then just at the end points of the motion. This offers the ability to have error terms of order $O\left(\Delta^{n+1}\right)$. The Newton method we just used to solve the probe problem is $O\left(\Delta^{2}\right)$, so this would be a first order method. Turns out that the Newton method is also unstable, so it would be in our best interest to develop a slightly more powerful numerical scheme for solving ODE’s. For us this will be the Runga Kutta scheme. Of course there are many other methods, many faster then Rugna Kutta, and each of them are good for their own territory. But in a first (week long) class Runga Kutta will be more then sufficient. If we feel we need more flexibity we can always create an adaptive stepsize algorithm to further the strength of this method.

The most common Runga Kutta scheme is the fourth order method. So how do we do this? We must calcuate coeffiecents for the expansion of the interated solution in terms of the step

$$y_{n+1} = y_n + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6}$$

Suppose that we have some differential equation we want to solve.

α(x)y’’+β(x)y’+γ(x)y=f(x)

There are two things we could do, one is to put the differential equation into first oder form,

z’=f(x,z) ,   z’=y’’,

or we just rearrange to get to,

$$y'' = f(x, y, y') = \frac{f(x)}{\alpha(x)} - \frac{\beta(x)}{\alpha(x)} y' - \frac{\gamma(x)}{\alpha(x)} y$$

with initial conditions of course.

Fixing a step size of Δ the RK4 scheme is as follows,

$$k_1 = \Delta \mathrm{f}(x_n, y_n)$$
$$k_2 = \Delta \mathrm{f}\left(x_n + \frac{\Delta}{2}, y_n + \frac{k_1}{2}\right)$$
$$k_3 = \Delta \mathrm{f}\left(x_n + \frac{\Delta}{2}, y_n + \frac{k_2}{2}\right)$$
$$k_4 = \Delta \mathrm{f}(x_n + \Delta, y_n + k_3)$$
$$y_{n+1} = y_n + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6}$$

First Order Equations

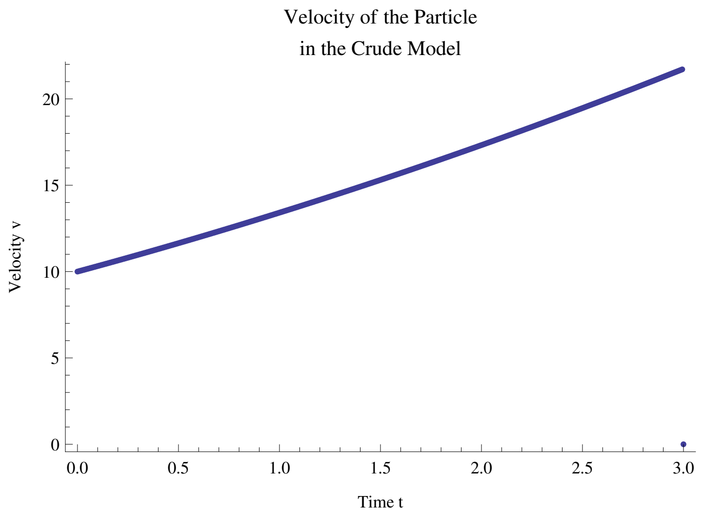
So lets see how we can implement this with a simple example. We are going to redo the "quick and dirty model" with the RK4 method.

$$a = v^{1/2}, \quad \frac{\mathrm{dv}}{\mathrm{dt}} = a, \qquad \frac{\mathrm{dv}}{\mathrm{dt}} = f(t, v) = v^{1/2} \qquad \mathrm{v(0)=10}$$

In the above line the first equation is the differential equation we want to solve. The second is the definition of acceleration in terms of velocity. Using this with the differential equation we now have the form of our function. Lets analyze this system for 3 units of time.

```
max = 500;
ttmax = 3;
Δ = ttmax / (max - 1);
vsoltab = Table[{Δ (i - 1), 0}, {i, 1, max}];
vsoltab[[1, 2]] = 10;
ff[v_] := v^(1/2) // N;


For[i = 1, i < max - 1, i++,
 ti = Δ (i - 1);
 k1 = Δ ff[vsoltab[[i, 2]]];
 k2 = Δ ff[vsoltab[[i, 2]] + k1/2];
 k3 = Δ ff[vsoltab[[i, 2]] + k2/2];
 k4 = Δ ff[vsoltab[[i, 2]] + k3];
 vsoltab[[i + 1, 2]] = vsoltab[[i, 2]] + 1/6 (k1 + 2 k2 + 2 k3 + k4);
]
```

```
ListPlot[vsoltab, PlotLabel → "Velocity of the Particle
in the Crude Model", Frame → {{True, False}, {True, False}},
 FrameLabel → {"Time t", "Velocity v"}]
```



How can we get the position?
How do we put in a condition to stop the code rather then run it for only units of time?

Second Order Equations

This time we will leave the equations in second order form

$a = v^{1/2}$, $\frac{dx}{dt} = v$, $\frac{d^2 x}{dt^2} = f(t, x, x') = \left(\frac{dx}{dt}\right)^{1/2}$,

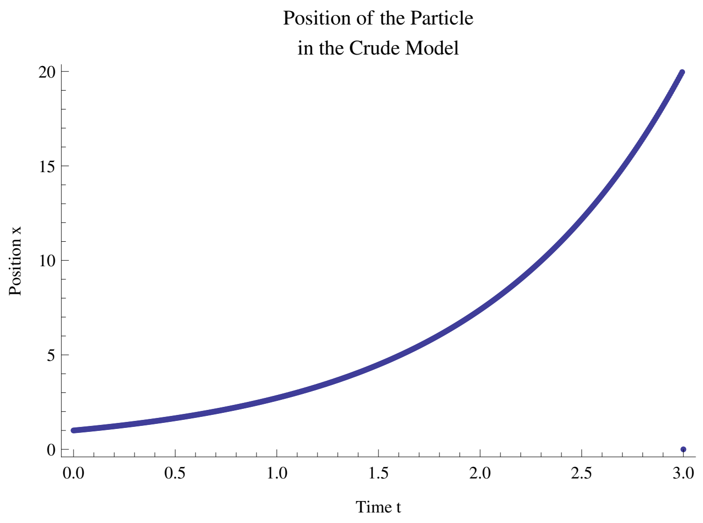
v(0)=10, x(0)=1

```
max = 500;
ttmax = 3;
Δ = ttmax / (max - 1);
xsoltab = Table[{Δ (i - 1), 0, 0}, {i, 1, max}];
xsoltab[[1, 2]] = 10;
xsoltab[[1, 3]] = 1;
fff[v_, u_] := {(v)^(1/2), u} // N;
```

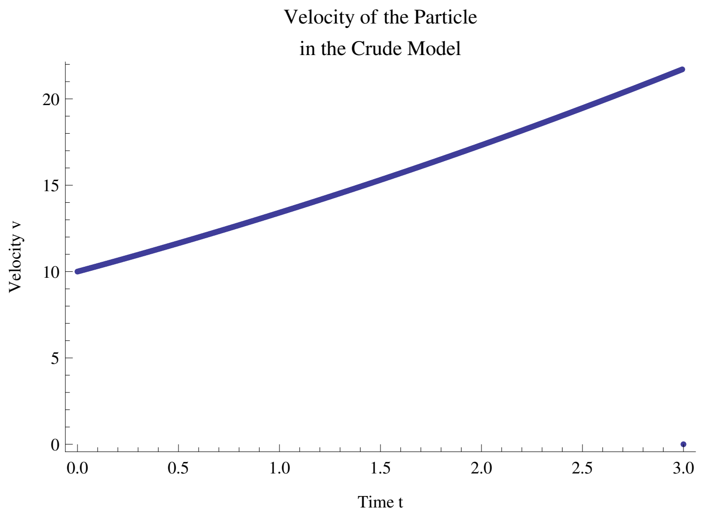
```
For[i = 1, i < max - 1, i ++,
 ti = Δ (i - 1);
 kk1 = Δ fff[xsoltab[[i, 2]], xsoltab[[i, 3]]];
 kk2 = Δ fff[xsoltab[[i, 2]] + kk1[[1]]/2, xsoltab[[i, 3]] + kk1[[2]]/2];
 kk3 = Δ fff[xsoltab[[i, 2]] + kk2[[1]]/2, xsoltab[[i, 3]] + kk2[[2]]/2];
 kk4 = Δ fff[xsoltab[[i, 2]] + kk3[[1]], xsoltab[[i, 3]] + kk3[[2]]];
 xsoltab[[i + 1, 2]] =
  xsoltab[[i, 2]] + 1/6 (kk1[[1]] + 2 kk2[[1]] + 2 kk3[[1]] + kk4[[1]]);
 xsoltab[[i + 1, 3]] = xsoltab[[i, 3]] + 1/6 (kk1[[2]] + 2 kk2[[2]] + 2 kk3[[2]] + kk4[[2]]);
]

xtab = Table[{Δ (i - 1), xsoltab[[i, 3]]}, {i, 1, max}];
vtab = Table[{Δ (i - 1), xsoltab[[i, 2]]}, {i, 1, max}];

xsoltab[[5, 2]]
```

```
10.0762
```

```
ListPlot[xtab, PlotLabel → "Position of the Particle
in the Crude Model", Frame → {{True, False}, {True, False}},
 FrameLabel → {"Time t", "Position x"}]
```



```
ListPlot[vtab, PlotLabel → "Velocity of the Particle
in the Crude Model", Frame → {{True, False}, {True, False}},
 FrameLabel → {"Time t", "Velocity v"}]
```



Your Task:

Your task is to repeat the more realistic trajectory, quoted here for ease,

Exercise: a more realistic trajectory

$$f_{\text{Grav}}(r) = G\frac{m_{\text{probe}}\ m_{\text{planet}}}{(r_{\text{planet}}+y)^2}$$

Evidently y is the distance above the Earth' s surface; next we include the drag force due to the atmo-spheric, which falls (if we assume fixed temperature) as the altitude rises because the atmosphere' s density falls approximately exponentially with height.

$$f_{\text{atmo}}(y, v) = \alpha\, e^{-y/y_0}\, v^2; \quad \alpha = 0.15708$$

Next come the parameters that we will use. Assume that all values are given in standard MKS units. You should recognize the gravitational constant. mProbe is the mass of the prob, mPlanet is the mass of the planet. y0 is a constant that defines the length scale of the atmospheric force which can be seen in the definition of forceAtmo. rPlanet is the radius of the planet. And finally h is the initial height of the Probe above the planet' s surface.

```
parameters = {gravConst = 6.67 * 10^-11, mProbe = 10,
   mPlanet = 6.419 * 10^23, y0 = 11 100, rPlanet = 3390 * 10^3, h = 10^6};
```

With the information above, create a module which does/has the following,

A function for the instantaneous velocity of the probe.
a For, While, or some other loop to,

    Create one array with (time, position) data pairs
    Create one array with (time, velocity) data pairs
    Create one array with (time, acceleration) data pairs

Your For loop should exit out once the probe hits the planet.
Plot the position, velocity, and acceleration as functions of time.

Your module should do all of this. Make sure that your module has enough inputs so that it is sufficiently general enough to be used to vary parameters with ease and compare plots. Also make the plot as nice as you can. (Use the code I give above for plotting. This means turning frames on and using frame labels rather then axes labels. This makes a much better looking plot. If you look I use True,False,True,-False in the code for the frames. Try turning one of those False values to True and see what happens.)

This Notebook use’s the text Numerical Recipes: The Art of Scientific Computing, 3rdEdition as a reference.