Tutorial 4: Runge-Kutta 4th order method solving ordinary differenital equations differential equations

Version 2, BRW, 1/31/07

Lets solve the differential equation found for the y direction of velocity with air resistance that is proportional to v.

vy'[t]==-k vy[t]-g

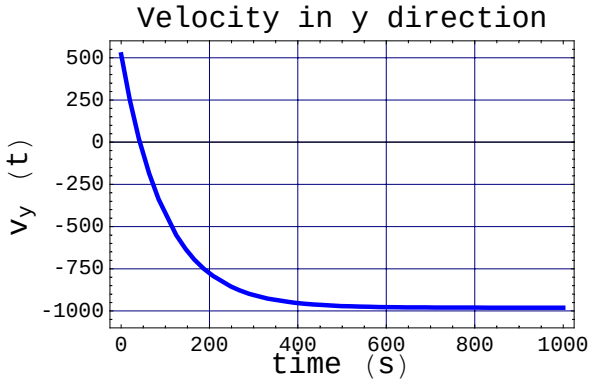
We will solve this differential equation numerically with NDSolve and using the 4th order Runge-Kutta method. Define the initial conditions.

```
In[1]:= g = 9.81;    (* in m/s^2 *)
        v0 = 600.0; (* in m/s *)
        k = 0.01;
        φ0 = 60.0 * π/180;  (* 20 degrees in radians *)
        vxo = v0 Cos[φ0];   (* define initial velocites at t=0.*)
        vyo = v0 Sin[φ0];
```

Solve the DE with NDSolve and plot the result.

```
In[5]:= sol = NDSolve[{vy'[t] == -k vy[t] - g, vy[0] == vyo}, vy, {t, 0, 2000}];
        vysol[t_] := vy[t] /. sol[[1]]
```

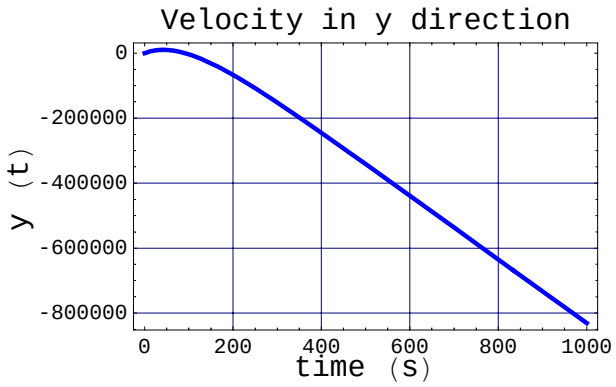
```
In[7]:= Plot[vysol[t], {t, 0, 1000}, Frame → True, PlotRange -> {All, {-1100, 600}},
          PlotStyle → {{RGBColor[0, 0, 1], Thickness[0.01]}}, GridLines → Automatic,
          PlotLabel → StyleForm["Velocity in y direction", FontSize → 14, FontWeight → "Bold"],
            FrameLabel -> {StyleForm["time (s)", FontSize → 14],
            StyleForm["vy (t)", FontSize → 14] }];
```



Solve for the position as a function of time

```
In[8]:= sol = NDSolve[{y''[t] == -k y'[t] - g, y'[0] == vyo, y[0] == 10}, y, {t, 0, 2000}];
        ysol[t_] := y[t] /. sol[[1]]
```

```
In[11]:= Plot[ysol[t], {t, 0, 1000}, Frame → True, PlotRange -> {All, All},
           PlotStyle → {{RGBColor[0, 0, 1], Thickness[0.01]}}, GridLines → Automatic,
           PlotLabel → StyleForm["Velocity in y direction", FontSize → 14, FontWeight → "Bold"],
             FrameLabel -> {StyleForm["time (s)", FontSize → 14],
             StyleForm["y (t)", FontSize → 14] }];
```



■ Solving with 4th order runge kutta

Runge-Kutta is a useful method for solving 1st order ordinary differential equations. Lets solve this differential equation using the 4th order Runge-Kutta method with n segments. The more segments, the better the solutions. The solution of the differential equation will be a lists of velocity values (vt[[i]]) for a list of time values (t[[i]]). REMEMBER that vt[[i]] and t[[i]] each are an array of values!
Define the differential equation as a function.

```
Remove[t, vy]

f[t_, vy_] := -k vy - g;
```

The function f[vy,t] is used to find the derivative at the initial condition. The first values used are the boundary conditions where vy(0)=v_{yo}. We want to solve for the times t=0 till t=2000 seconds

```
t0 = 0;
tf = 2000;
```

Using 4th order Runge Kutta method, the solution of the differential equation vy[[i]] at time t[[i]] is determined by

t[[i+1]]=t[[i]]+h;
vy[[i+1]]=vy[[i]]+(k1+2*k2+2*k3+k4)*h /6

where h is the step size,
h=(tf-t0)/n

n is the number of steps, tf is the final time, and t0 is the initial time, and remembering the function f[t, vy] we can write
k1=f[t[[i]],vy[[i]]]
k2=f[t[[i]]+0.5*h,vy[[i]]+0.5*k1*h]
k3=f[t[[i]]+0.5*h,vy[[i]]+0.5*k2*h]

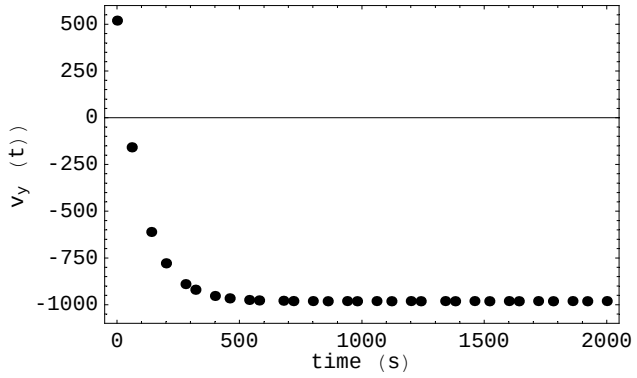
k4=f[t[[i]]+h,vy[[i]]+k3*h];

We can write a for loop to complete the Runge Kutta over n segments

```
n = 100; (* number of steps *)
t = Table[j, {j, 1, n + 1}]; (* initialize the arrays t and vy *)
vy = Table[j, {j, 1, n + 1}];

t[[1]] = t0; (* initial conditions *)
vy[[1]] = vyo;

h = (tf - t0) / n; (* step size *)
For[i = 1, i < n + 1, i++,
 {
  k1 = f[t[[i]], vy[[i]]];
  k2 = f[t[[i]] + 0.5 * h, vy[[i]] + 0.5 * k1 * h];
  k3 = f[t[[i]] + 0.5 * h, vy[[i]] + 0.5 * k2 * h];
  k4 = f[t[[i]] + h, vy[[i]] + k3 * h];
  t[[i + 1]] = t[[i]] + h;
  vy[[i + 1]] = vy[[i]] + (k1 + 2 * k2 + 2 * k3 + k4) * h / 6;
 }]
```

Let us plot the solution list {t[[i]],vy[[i]]} and compare with what was computed earlier. We get the same plot.

```
vyt = Table[{t[[j]], vy[[j]]}, {j, 1, Length[t]}];

ListPlot[vyt, Frame → True, PlotJoined → False, PlotRange -> {All, {-1100, 600}},
  PlotStyle → PointSize[0.02], FrameLabel -> {"time (s)", "vy (t))"}];
```



■ Runge Kutta for two coupled 1st order differential equations

Can we use Runge Kutta for a 2nd order differential equation? We can if we can write the 2nd order DE as a coupled set of 1st order equations. Let us solve for the position using Runge Kutta. The 2nd order differential equation is

y''[t]==-k y'[t]-g

Solve for y(t). Rewrite the DE as two coupled first order DE's. Let y_1=y[t] and y_2= y'[t]. The two coupled DE are

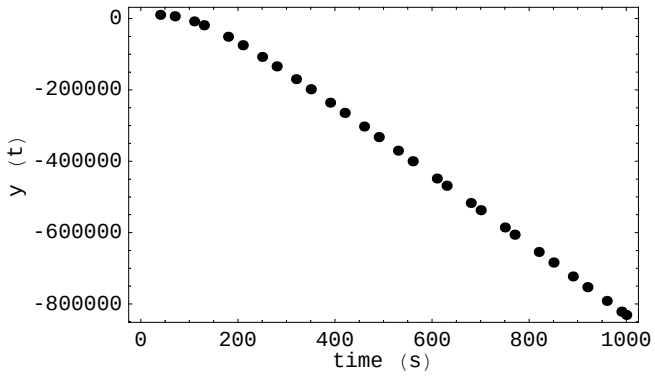
$$\frac{dy_1}{dt}=y_2 \text{ and } \frac{dy_2}{dt}=\text{-k } y_2 - g$$

Solve the DE for y_1 and y_2. We can used a modified form of the Runge kutta routine for the coupled equations.

```
In[26]:= v0y = v0 Sin[φ0];
         y0 = 10;
         t0 = 0;
         tf = 1000;

        n = 100; (* number of steps *)
        t = Table[j, {j, 1, n + 1}]; (* initialize the arrays t and y *)
        y1 = Table[j, {j, 1, n + 1}];
        y2 = Table[j, {j, 1, n + 1}];

        (* initial conditions *)
        t[[1]] = t0;
        y1[[1]] = y0;
        y2[[1]] = v0y;

        (* define the functions from the two coupled  DE *)
        f1[t_, y1_, y2_] := y2
        f2[t_, y1_, y2_] := - k y2 - g
        h = (tf - t0) / n; (* step size *)
        For[i = 1, i < n + 1, i++,
         {
          k1 = f1[t[[i]], y1[[i]], y2[[i]]];
          m1 = f2[t[[i]], y1[[i]], y2[[i]]];
          k2 = f1[t[[i]] + 0.5 * h, y1[[i]] + 0.5 * k1 * h, y2[[i]] + 0.5 * m1 * h];
          m2 = f2[t[[i]] + 0.5 * h, y1[[i]] + 0.5 * k1 * h, y2[[i]] + 0.5 * m1 * h];
          k3 = f1[t[[i]] + 0.5 * h, y1[[i]] + 0.5 * k2 * h, y2[[i]] + 0.5 * m2 * h];
          m3 = f2[t[[i]] + 0.5 * h, y1[[i]] + 0.5 * k2 * h, y2[[i]] + 0.5 * m2 * h];
          k4 = f1[t[[i]] + h, y1[[i]] + k3 * h, y2[[i]] + m3 * h];
          m4 = f2[t[[i]] + h, y1[[i]] + k3 * h, y2[[i]] + m3 * h];

          (* find the iterative solution *)
          t[[i + 1]] = t[[i]] + h;
          y1[[i + 1]] = y1[[i]] + (k1 + 2 * k2 + 2 * k3 + k4) * h / 6;
          y2[[i + 1]] = y2[[i]] + (m1 + 2 * m2 + 2 * m3 + m4) * h / 6;
         }]
```

Remember that y1 is the solution we want since y1=y(t).

```
In[40]:= yt = Table[{t[[j]], y1[[j]]}, {j, 1, Length[t]}];
         ListPlot[yt, Frame → True, PlotJoined → False, PlotRange -> {All, All},
           PlotStyle → PointSize[0.02], FrameLabel -> {"time (s)", "y (t)"}];
```



Which is the same answer we got using NDSolve