In [1]:
```
import sympy as sp
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
sp.init_printing()
```

Solve the differential equation

$$\frac{dy}{dt} = ty$$

using RK1, Rk2, and RK4 methods over the interval $[0, 2]$ with the initial condition $y(0) = 1$

In [2]:
```
def rk1(f,time,initial,h):    #define the function
    points=np.arange(time[0],time[1]+h,h)   #divide the interval
    n=len(points)   # how many points here
    sol=np.zeros(n)   #where we store the solution for y
    sol[0]=initial        #if y is solution then the first entry of sol is y(t_0)
    for i in range(1,n):
        sol[i]=sol[i-1]+h*f.subs({t:points[i-1],y:sol[i-1]})     #y_n=y_(n-1)+h(f(x
    return sol
```

In [3]:
```
def rk2(f,time,initial,h):
    points=np.arange(time[0],time[1]+h,h)
    n=len(points)
    sol=np.zeros(n)
    sol[0]=initial
    for i in range(1,n):
        k1=f.subs({t:points[i-1],y:sol[i-1]})
        k2=f.subs({t:(points[i-1]+h),y:(sol[i-1]+h*k1)})
        sol[i]=sol[i-1]+h/2*(k1+k2)     #y_n=y_(n-1)+h(f(x_(n-1),y_(n-1)))
    return sol
```

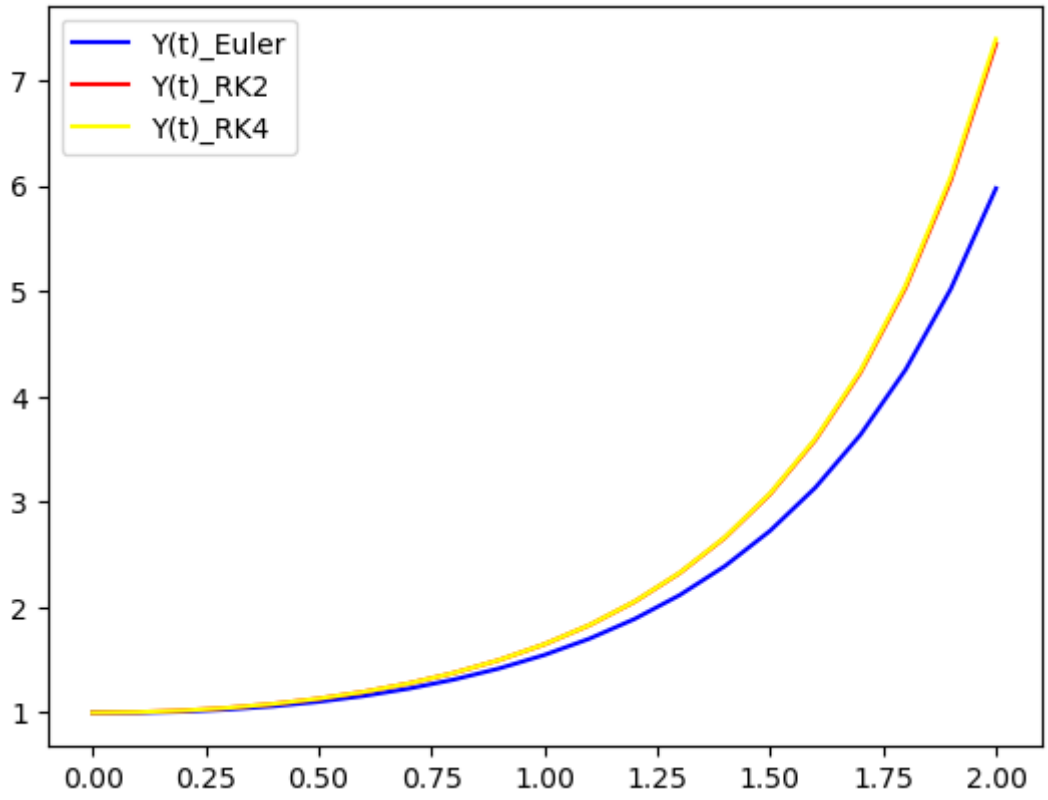
In [4]:
```
def rk4(f,time,initial,h):
    points=np.arange(time[0],time[1]+h,h)
    n=len(points)
    sol=np.zeros(n)
    sol[0]=initial
    for i in range(1,n):
        k1=f.subs({t:points[i-1],y:sol[i-1]})
        k2=f.subs({t:(points[i-1]+h/2),y:(sol[i-1]+h/2*k1)})
        k3=f.subs({t:(points[i-1]+h/2),y:(sol[i-1]+h/2*k2)})
        k4=f.subs({t:(points[i-1]+h),y:(sol[i-1]+h*k3)})
        sol[i]=sol[i-1]+h*(k1/6+k2/3+k3/3+k4/6)     #y_n=y_(n-1)+h(f(x_(n-1),y_(n-1
    return sol
```

In [5]:
```
x,y,t=sp.symbols("x y t")
```

In [6]:
```
f=t*y #y'=f(t,y)
time=[0,2] #intravl(time)
initial=1 #y(t_0)
h=0.1  #step size
Euler_sol=rk1(f,time,initial,h) #here, we call the function
rk2_sol=rk2(f,time,initial,h)
rk4_sol=rk4(f,time,initial,h)
```

In [7]:
```
points=np.arange(time[0],time[1]+h,h)
fig,ax=plt.subplots()
ax.plot(points,Euler_sol,color="blue",label="Y(t)_Euler")
ax.plot(points,rk2_sol,color="red",label="Y(t)_RK2")
ax.plot(points,rk4_sol,color="Yellow",label="Y(t)_RK4")
ax.legend()
```

Out[7]:
```
<matplotlib.legend.Legend at 0x16cbb69fbd0>
```



In [8]:
```
import sympy as sp
import numpy as np
import matplotlib.pyplot as plt
sp.init_printing()
from scipy.integrate import solve_ivp
```

Integrating numerically using RK45 method

$$\frac{dy}{dt} = \sin(t) + 3 * \cos(2 * t) - y$$

from $[0, 10]$ whose exact solution is

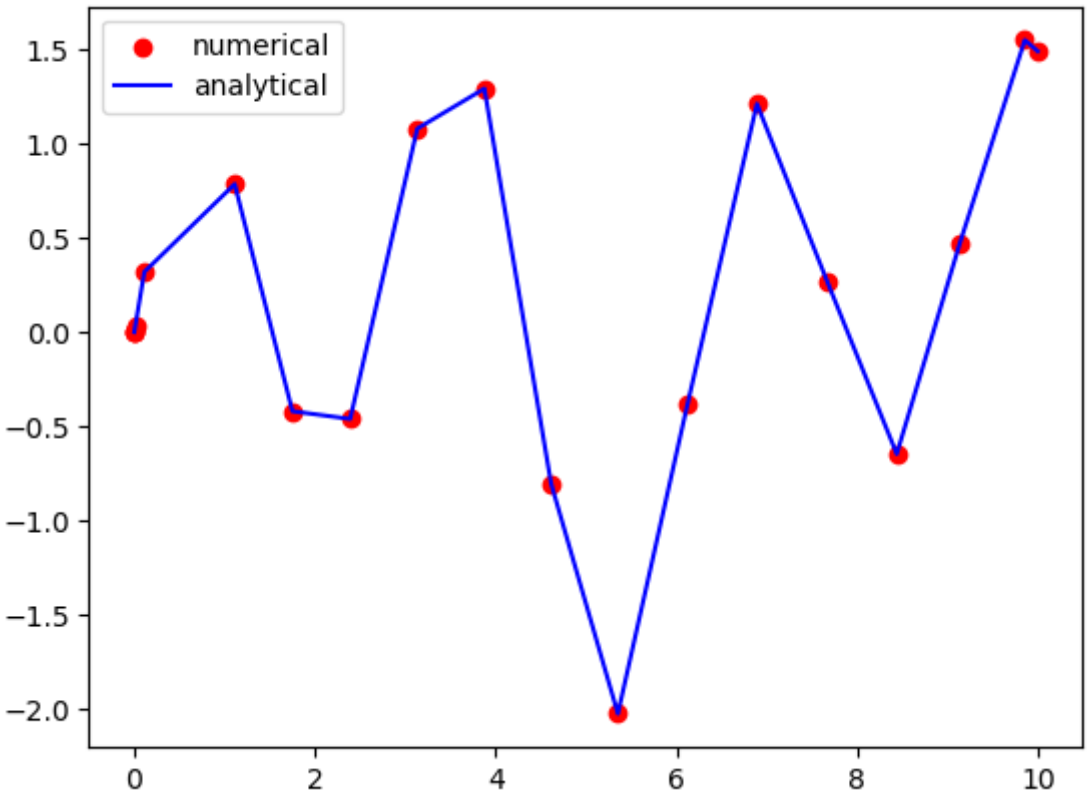
$y(t) = 1/2 * \sin(t) - 1/2 * cos(t) + (3/5) * cos(2 * t) + 6/5 *$

In [9]:
```
t=sp.symbols("t")
ode_fn = lambda t, y: sp.sin(t) + 3. * sp.cos(2. * t) -y
exact_sol=(1./2.) * sp.sin(t) - (1./2.) * sp.cos(t) + (3./5.) * sp.cos(2.*t) + (6./
exact_sol=sp.lambdify(t,exact_sol,modules="numpy")
t_begin=0
t_end=10
#h=0.1
#t_space = np.arange(t_begin, t_end+h, h)
y_init = [0]
```

```
#y_an_sol = exact_sol(t_space)
num_sol = solve_ivp(ode_fn, [t_begin, t_end], y_init, method='RK45', t_eval=None)
y_an_sol = exact_sol(num_sol.t)
#num_sol.t,num_sol.y
```

In [10]:
```
fig,ax=plt.subplots()
ax.scatter(num_sol.t,num_sol.y[0],color="red",label="numerical")
ax.plot(num_sol.t,y_an_sol,color="blue",label="analytical")
ax.legend()
```

Out[10]:
```
<matplotlib.legend.Legend at 0x16cbd826010>
```



Integrating numerically using RK45 method

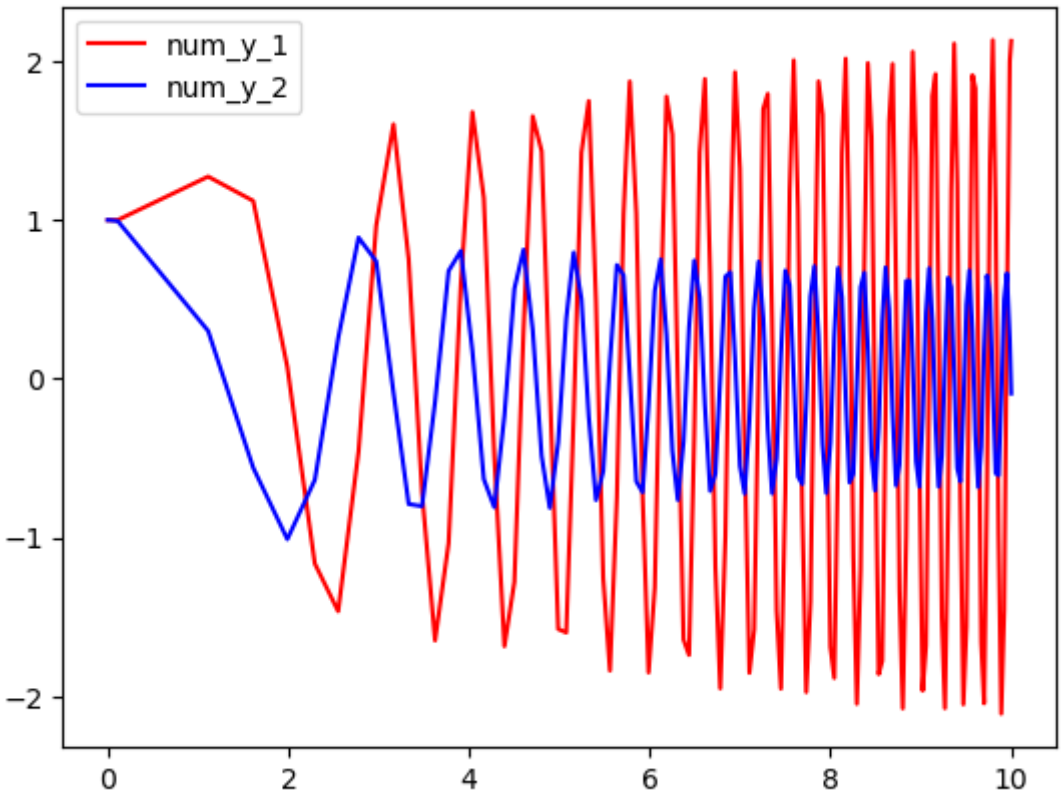
$$\frac{dy_1}{dt} = t^2 y_2, \frac{dy_2}{dt} = -t y_1$$

from $[0, 10]$ with initial conditions $y_1(0) = 1$ and $y_2(0) = 1$.

In [11]:
```
ode_fn_sys=lambda t, y:[t**2*y[1],-t*y[0]]
y_init=[1,1]
num_sol_sys = solve_ivp(ode_fn_sys, [t_begin, t_end], y_init, method='RK45', t_eval
#num_sol_sys.t,num_sol_sys.y[0],num_sol_sys.y[1]
```

In [12]:
```
fig,ax1=plt.subplots()
ax1.plot(num_sol_sys.t,num_sol_sys.y[0],color="red",label="num_y_1")
ax1.plot(num_sol_sys.t,num_sol_sys.y[1],color="blue",label="num_y_2")
ax1.legend()
```

Out[12]:
```
<matplotlib.legend.Legend at 0x16cbb7573d0>
```


num_y_1
num_y_2
2
1
0
−1
−2
0
2
4
6
8
10