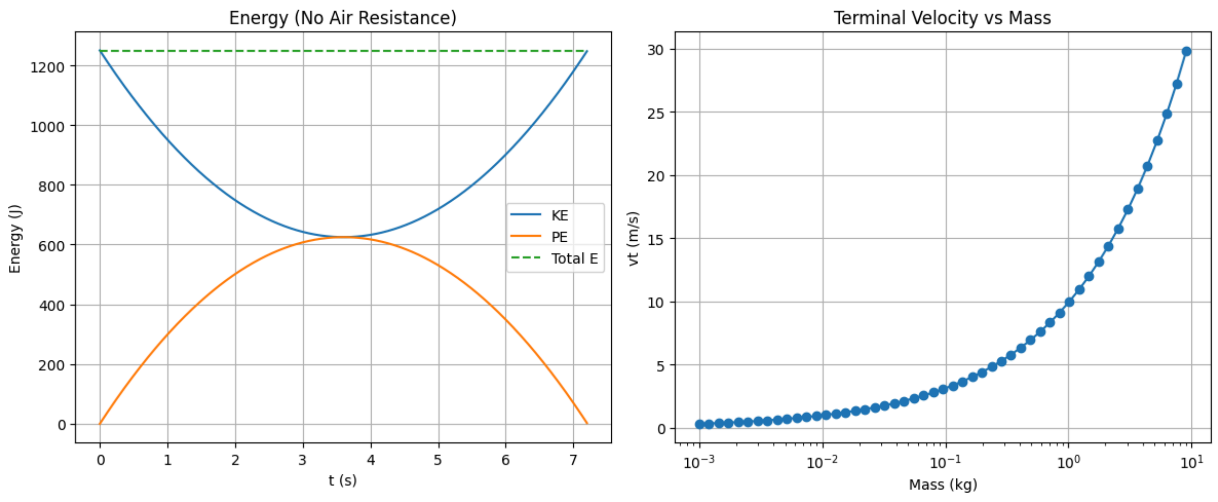
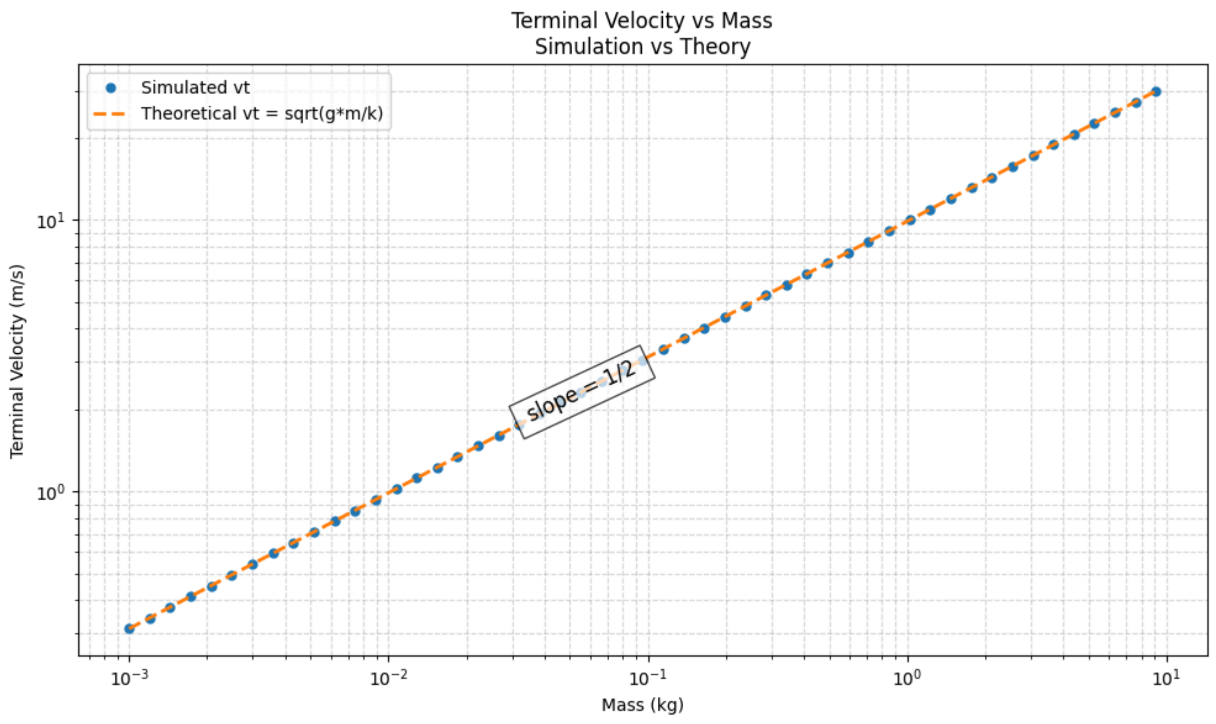
In [8]:
```
import numpy as np
import matplotlib.pyplot as plt

# Energy ---------------------------
data1 = np.loadtxt("./build/energy_no_air.txt")
t, KE, PE, E = data1.T

plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.plot(t, KE, label="KE")
plt.plot(t, PE, label="PE")
plt.plot(t, E, label="Total E", linestyle="--")
plt.title("Energy (No Air Resistance)")
plt.xlabel("t (s)")
plt.ylabel("Energy (J)")
plt.grid()
plt.legend()

# vt vs mass ----------------------
data2 = np.loadtxt("./build/vt_vs_mass.txt")
m, vt = data2.T

plt.subplot(1,2,2)
plt.plot(m, vt, marker="o")
plt.xscale("log")
plt.title("Terminal Velocity vs Mass")
plt.xlabel("Mass (kg)")
plt.ylabel("vt (m/s)")
plt.grid()

plt.tight_layout()
plt.show()
```



In [9]:
```
import numpy as np
import matplotlib.pyplot as plt

data = np.loadtxt("./build/vt_vs_mass.txt")
m = data[:, 0]
vt_sim = data[:, 1]
```

```
# -----------------------------------------------------
# Theoretical curve:
#   v_t = sqrt(g*m/k)
# -----------------------------------------------------
g = 9.81
k = 0.1   # <- same value used in vterm.cpp

vt_theory = np.sqrt(g * m / k)

# -----------------------------------------------------
# Plotting
# -----------------------------------------------------
plt.figure(figsize=(10,6))

plt.plot(m, vt_sim, 'o', label="Simulated vt", markersize=5)
plt.plot(m, vt_theory, '--', linewidth=2, label="Theoretical vt = sqrt(g*m/k

plt.xscale('log')
plt.yscale('log')
plt.xlabel("Mass (kg)")
plt.ylabel("Terminal Velocity (m/s)")
plt.title("Terminal Velocity vs Mass\nSimulation vs Theory")

plt.grid(True, which="both", ls="--", alpha=0.5)
plt.legend()

# -----------------------------------------------------
# Annotate slope = 1/2
# -----------------------------------------------------
# choose two points on the theory curve
x1 = 0.01
x2 = 0.1
y1 = np.sqrt(g * x1 / k)
y2 = np.sqrt(g * x2 / k)

# Compute midpoints for placing the annotation
xm = np.sqrt(x1*x2)
ym = np.sqrt(y1*y2)

plt.text(xm, ym, " slope = 1/2 ", rotation=25, fontsize=12,
         bbox=dict(facecolor='white', alpha=0.7))

plt.tight_layout()
plt.show()
```



How well is energy conserved in the solution? Play with the step size to see how this affects your observation.

- The energy conservation error shows that for RK4 hits machine precision really quickly with small step sizes. Energy is very well conserved for no air resistance case.

Turn the air resistance back on. Modify the program to determine the terminal velocity (vt) of the projectile. Determine vt using the default mass and air drag parameter given in the example.

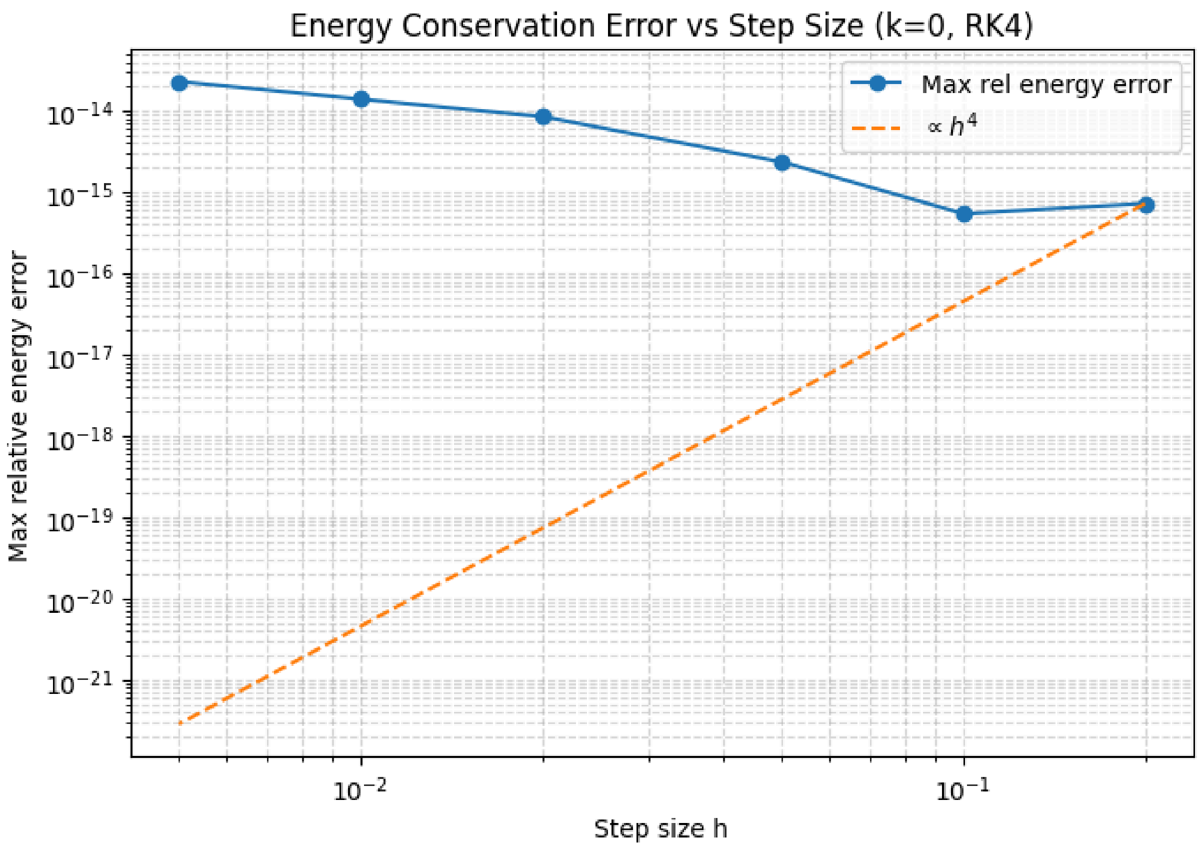
- Default mass = 1kg, and 0.1 for k, we get vterm is equal to 9.9 m/s

Think about how you might evaluate the accuracy of your solution. Discuss any cross checks or evidence that the solutions with air resistance are reasonably accurate?

- The drag formula gives a theory of sqrt(gm/k), and from the terminal velocity graph we get the same results.

In [14]:
```
import numpy as np
import matplotlib.pyplot as plt

data = np.loadtxt("./build/energy_error_vs_h.txt")
h = data[:,0]
err = data[:,1]

plt.figure(figsize=(7,5))
plt.loglog(h, err, 'o-', label='Max rel energy error')

# optional: reference h^4 line (scaled)
```

```
ref = (h/h[0])**4 * err[0]
plt.loglog(h, ref, '--', label=r'$\propto h^4$')

plt.xlabel("Step size h")
plt.ylabel("Max relative energy error")
plt.title("Energy Conservation Error vs Step Size (k=0, RK4)")
plt.grid(True, which="both", ls="--", alpha=0.5)
plt.legend()
plt.tight_layout()
plt.show()

# pick the "clean" part of the curve: h=0.05, 0.02, 0.01
mask = (h >= 0.05)
slope, _ = np.polyfit(np.log(h[mask]), np.log(err[mask]), 1)
print("Measured order:", slope)
```



```
Measured order: -0.8502207742196536
```