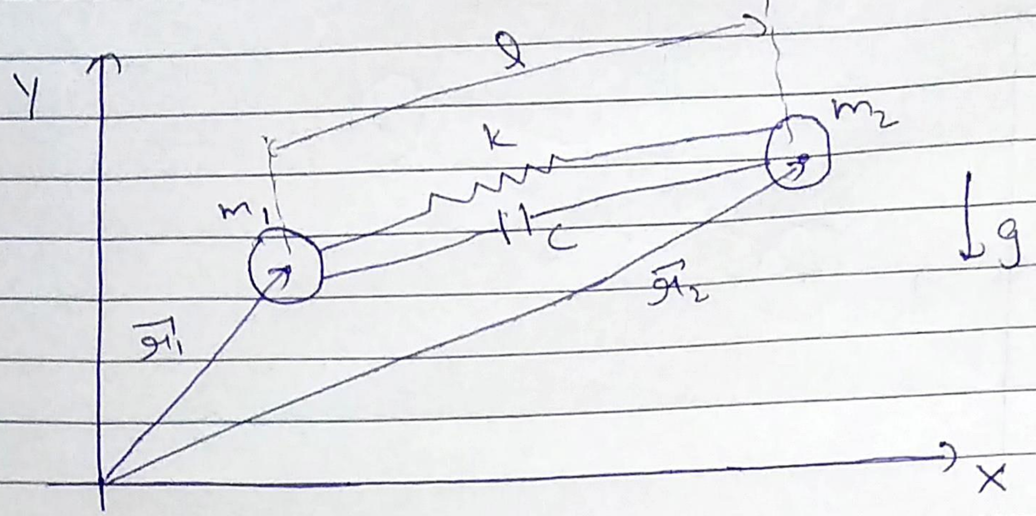
Date..............

ME 3030
Assignment - 1

Q



Spring force
$$\vec{F_s} = k\left(|\bar{r}_2-\bar{r}_1|-l\right)\left(\frac{\bar{r}_2-\bar{r}_1}{|\bar{r}_2-\bar{r}_1|}\right)$$

Damping force $\quad \bar{F_d} = c(\dot{r}_2-\dot{r}_1)$

Gravity force $\quad \vec{F_g} = -m\vec{g}$

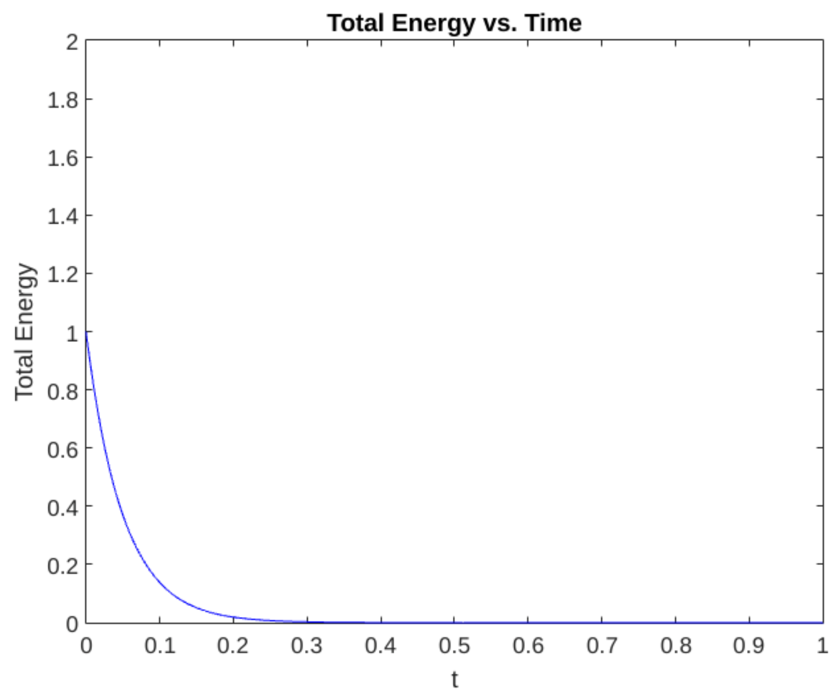
$$r_1 = \begin{Bmatrix} x_1(t) \\ y_1(t) \end{Bmatrix} \quad r_2 = \begin{Bmatrix} x_2(t) \\ y_2(t) \end{Bmatrix} \quad \dot{r}_1 = \begin{Bmatrix} \dot{x}_1(t) \\ \dot{y}_1(t) \end{Bmatrix}$$

$$\dot{r}_2 = \begin{Bmatrix} \dot{x}_2(t) \\ \dot{y}_2(t) \end{Bmatrix}$$

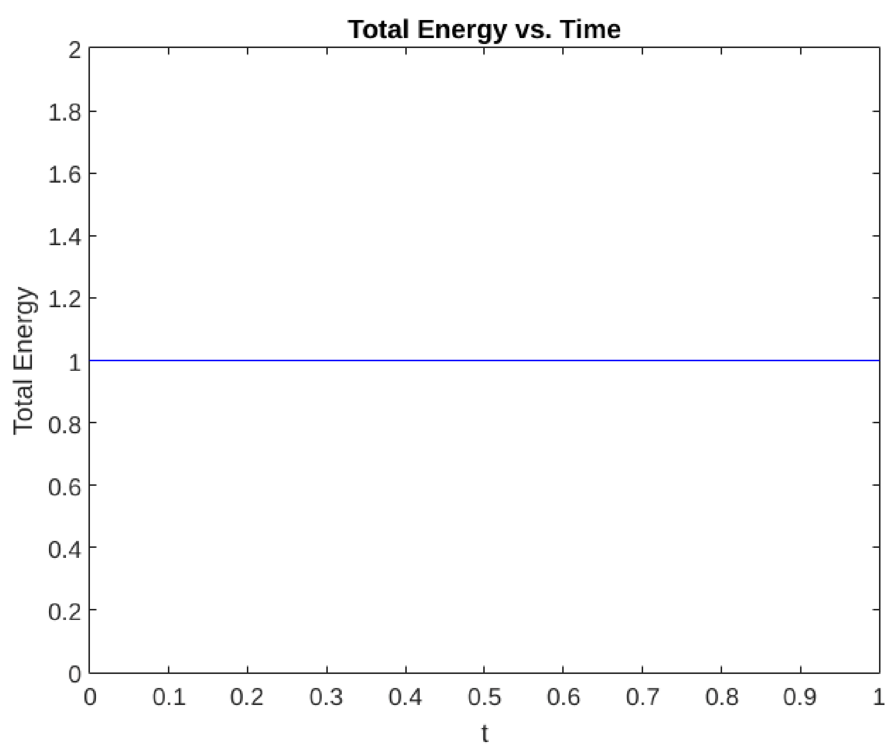
Writing equations for m_1:
$$m_1\begin{Bmatrix} \ddot{x}_1(t) \\ \ddot{y}_1(t) \end{Bmatrix} = \frac{k\left(|\bar{r}_2-\bar{r}_1|-l\right)}{|\bar{r}_2-\bar{r}_1|}\begin{Bmatrix} x_2-x_1 \\ y_2-y_1 \end{Bmatrix} + c\begin{Bmatrix} \dot{x}_2-\dot{x}_1 \\ \dot{y}_2-\dot{y}_1 \end{Bmatrix} - m_1\begin{Bmatrix} 0 \\ g \end{Bmatrix}$$

Writing equations for m_2:
$$m_2\begin{Bmatrix} \ddot{x}_2(t) \\ \ddot{y}_2(t) \end{Bmatrix} = k\left(\frac{|\bar{r}_2-\bar{r}_1|-l}{|\bar{r}_2-\bar{r}_1|}\right)\begin{Bmatrix} x_1-x_2 \\ y_1-y_2 \end{Bmatrix} - c\begin{Bmatrix} \dot{x}_2-\dot{x}_1 \\ \dot{y}_2-\dot{y}_1 \end{Bmatrix} - m_2\begin{Bmatrix} 0 \\ g \end{Bmatrix}$$

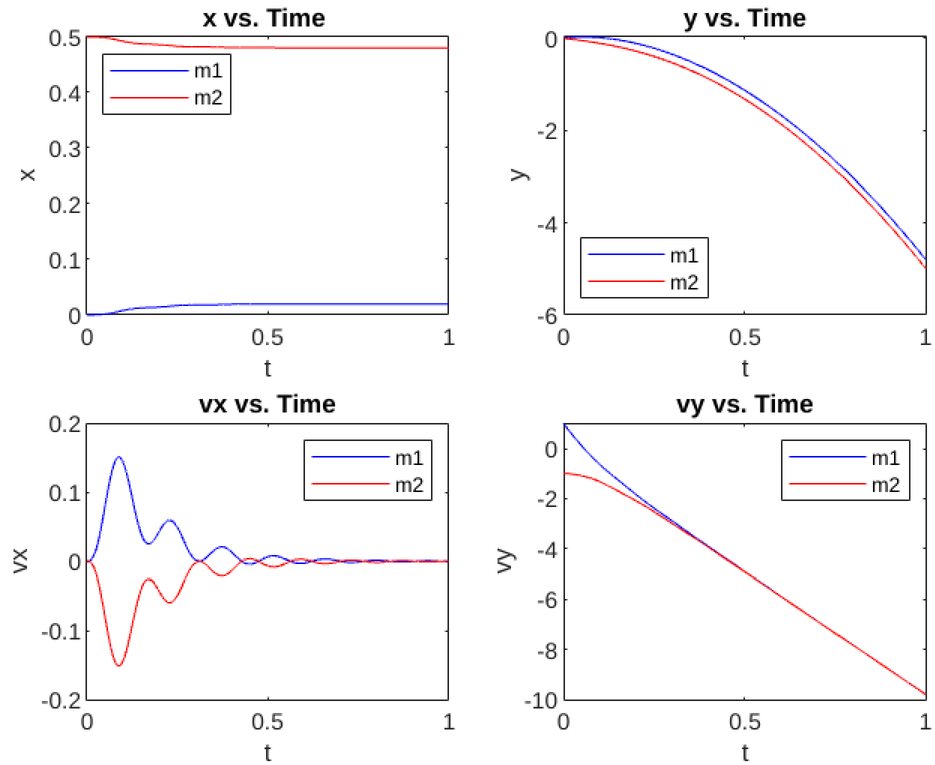
1. Plot of total energy vs time when damping is non zero:



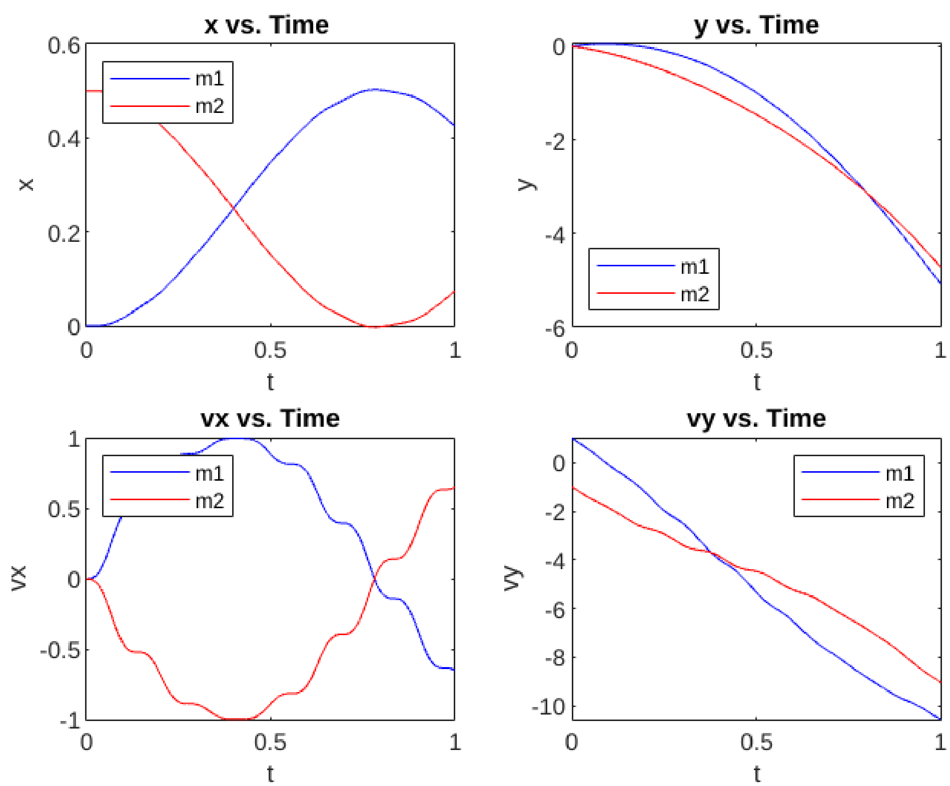
2. Plot of total energy vs time when damping is zero:



3. Plot of x, y, vx and vy when damping is non zero



4. Plot of x, y, vx and vy when damping is zero



```
% Define system parameters
m1 = 1.0;       % Mass of m1 in kg
m2 = 1.0;       % Mass of m2 in kg
k = 1000.0;     % Spring stiffness in N/m
c = 5.0;        % Damping coefficient in Ns/m
l = 0.5;      % Free length of the spring in m
g = 9.81;     % Acceleration due to gravity in m/s^2

% Initial conditions
x1_initial = 0.0;
y1_initial = 0.0;
vx1_initial = 0.0;
vy1_initial = 1.0;

x2_initial = 0.5;
y2_initial = 0.0;
vx2_initial = 0.0;
vy2_initial = -1.0;

% Time step
dt = 1.0e-5;

% Number of time steps
num_steps = 100000;

% Time array
time = (0:num_steps-1) * dt;

% Initialize arrays to store positions and velocities
x1 = zeros(1, num_steps);
y1 = zeros(1, num_steps);
vx1 = zeros(1, num_steps);
vy1 = zeros(1, num_steps);

x2 = zeros(1, num_steps);
y2 = zeros(1, num_steps);
vx2 = zeros(1, num_steps);
vy2 = zeros(1, num_steps);

% Initialize distance between masses array
distance_between_masses = zeros(1, num_steps);

% Initialize kinetic energy, spring potential energy, gravitational potential
 energy and total energy array
kinetic_energy = zeros(1, num_steps);
spring_pot_energy = zeros(1, num_steps);
grav_pot_energy = zeros(1, num_steps);
total_energy = zeros(1, num_steps);

% Set initial conditions
x1(1) = x1_initial;
y1(1) = y1_initial;
```

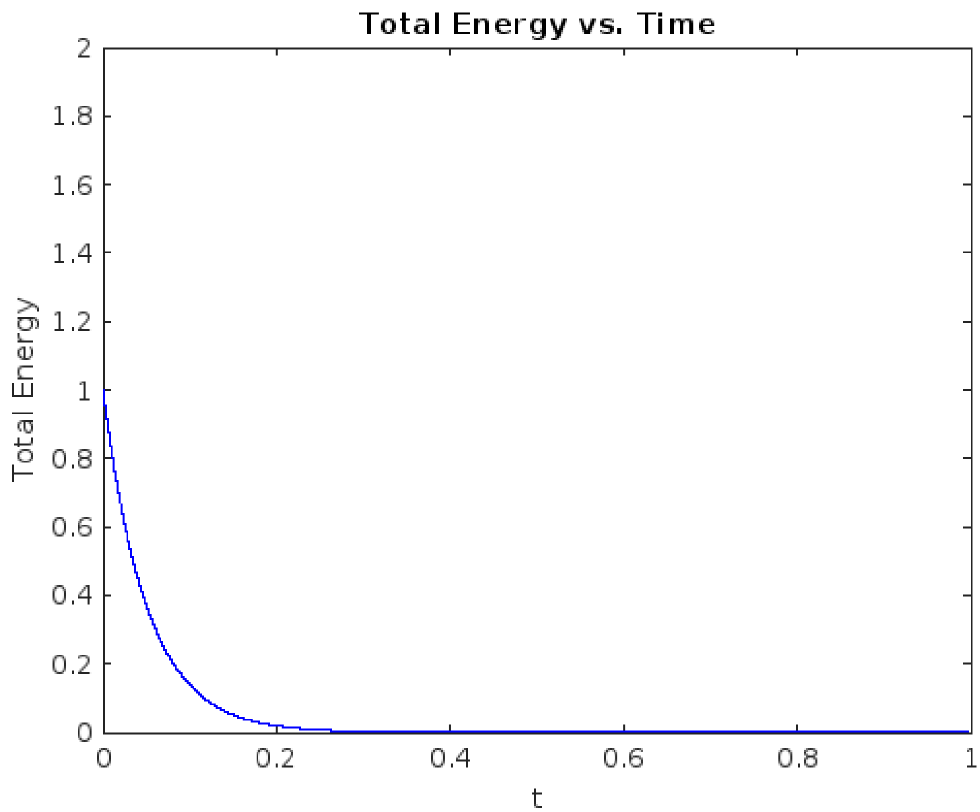
```
vx1(1) = vx1_initial;
vy1(1) = vy1_initial;

x2(1) = x2_initial;
y2(1) = y2_initial;
vx2(1) = vx2_initial;
vy2(1) = vy2_initial;

distance_between_masses(1) = sqrt((x2(1) - x1(1))^2 + (y2(1) - y1(1))^2);
kinetic_energy(1) = 0.5*m1*((vx1(1))^2 + (vy1(1))^2) + 0.5*m2*((vx2(1))^2 +
 (vy2(1))^2);
spring_pot_energy(1) = 0.5*k*((distance_between_masses(1) - l)^2);
grav_pot_energy(1) = m1*g*y1(1) + m2*g*y2(1);
total_energy(1) =  kinetic_energy(1) + spring_pot_energy(1) +
 grav_pot_energy(1);

% disp(total_energy(1));

% Using Euler's explicit scheme
for i = 1:num_steps-1
    % Forces
    spring_force = k * (distance_between_masses(i) - l);
    damper_force_x = c * (vx2(i) - vx1(i));
    damper_force_y = c * (vy2(i) - vy1(i));

    % Accelerations
    ax1 = (spring_force * (x2(i) - x1(i))) / (m1 * distance_between_masses(i))
 + damper_force_x / m1;
    ay1 = (spring_force * (y2(i) - y1(i))) / (m1 * distance_between_masses(i))
 + damper_force_y / m1 - g;

    ax2 = (spring_force * (x1(i) - x2(i))) / (m2 * distance_between_masses(i))
 - damper_force_x / m2;
    ay2 = (spring_force * (y1(i) - y2(i))) / (m2 * distance_between_masses(i))
 - damper_force_y / m2 - g;

    % Calculate k1 values
    k1x1 = vx1(i);
    k1x2 = vx2(i);
    k1y1 = vy1(i);
    k1y2 = vy2(i);
    k1vx1 = ax1;
    k1vx2 = ax2;
    k1vy1 = ay1;
    k1vy2 = ay2;

    % To calculate, k2 values, first calculate some temporary values of x1,
    % x2, y1, y2, etc...
    temp_x1 = x1(i) + 0.5*dt*k1x1;
    temp_x2 = x2(i) + 0.5*dt*k1x2;
    temp_y1 = y1(i) + 0.5*dt*k1y1;
    temp_y2 = y2(i) + 0.5*dt*k1y2;
    temp_vx1 = vx1(i) + 0.5*dt*k1vx1;
    temp_vx2 = vx2(i) + 0.5*dt*k1vx2;
```

```
    temp_vy1 = vy1(i) + 0.5*dt*k1vy1;
    temp_vy2 = vy2(i) + 0.5*dt*k1vy2;

    temp_distance = sqrt((temp_x2 - temp_x1)^2 + (temp_y2 - temp_y1)^2);
    temp_spring_force = k * (temp_distance - l);
    temp_damper_force_x = c * (temp_vx2 - temp_vx1);
    temp_damper_force_y = c * (temp_vy2 - temp_vy1);

    temp_ax1 = (temp_spring_force * (temp_x2 - temp_x1)) / (m1 *
temp_distance) + temp_damper_force_x / m1;
    temp_ay1 = (temp_spring_force * (temp_y2 - temp_y1)) / (m1 *
temp_distance) + temp_damper_force_y / m1 - g;

    temp_ax2 = (temp_spring_force * (temp_x1 - temp_x2)) / (m2 *
temp_distance) - temp_damper_force_x / m2;
    temp_ay2 = (temp_spring_force * (temp_y1 - temp_y2)) / (m2 *
temp_distance) - temp_damper_force_y / m2 - g;

    k2x1 = temp_vx1;
    k2x2 = temp_vx2;
    k2y1 = temp_vy1;
    k2y2 = temp_vy2;
    k2vx1 = temp_ax1;
    k2vx2 = temp_ax2;
    k2vy1 = temp_ay1;
    k2vy2 = temp_ay2;

    % To calculate, k3 values, first calculate some temporary values of x1,
    % x2, y1, y2, etc...
    temp_x1 = x1(i) + 0.5*dt*k2x1;
    temp_x2 = x2(i) + 0.5*dt*k2x2;
    temp_y1 = y1(i) + 0.5*dt*k2y1;
    temp_y2 = y2(i) + 0.5*dt*k2y2;
    temp_vx1 = vx1(i) + 0.5*dt*k2vx1;
    temp_vx2 = vx2(i) + 0.5*dt*k2vx2;
    temp_vy1 = vy1(i) + 0.5*dt*k2vy1;
    temp_vy2 = vy2(i) + 0.5*dt*k2vy2;

    temp_distance = sqrt((temp_x2 - temp_x1)^2 + (temp_y2 - temp_y1)^2);
    temp_spring_force = k * (temp_distance - l);
    temp_damper_force_x = c * (temp_vx2 - temp_vx1);
    temp_damper_force_y = c * (temp_vy2 - temp_vy1);

    temp_ax1 = (temp_spring_force * (temp_x2 - temp_x1)) / (m1 *
temp_distance) + temp_damper_force_x / m1;
    temp_ay1 = (temp_spring_force * (temp_y2 - temp_y1)) / (m1 *
temp_distance) + temp_damper_force_y / m1 - g;

    temp_ax2 = (temp_spring_force * (temp_x1 - temp_x2)) / (m2 *
temp_distance) - temp_damper_force_x / m2;
    temp_ay2 = (temp_spring_force * (temp_y1 - temp_y2)) / (m2 *
temp_distance) - temp_damper_force_y / m2 - g;

    k3x1 = temp_vx1;
```

```
    k3x2 = temp_vx2;
    k3y1 = temp_vy1;
    k3y2 = temp_vy2;
    k3vx1 = temp_ax1;
    k3vx2 = temp_ax2;
    k3vy1 = temp_ay1;
    k3vy2 = temp_ay2;

    % To calculate, k4 values, first calculate some temporary values of x1,
    % x2, y1, y2, etc...
    temp_x1 = x1(i) + dt*k3x1;
    temp_x2 = x2(i) + dt*k3x2;
    temp_y1 = y1(i) + dt*k3y1;
    temp_y2 = y2(i) + dt*k3y2;
    temp_vx1 = vx1(i) + dt*k3vx1;
    temp_vx2 = vx2(i) + dt*k3vx2;
    temp_vy1 = vy1(i) + dt*k3vy1;
    temp_vy2 = vy2(i) + dt*k3vy2;

    temp_distance = sqrt((temp_x2 - temp_x1)^2 + (temp_y2 - temp_y1)^2);
    temp_spring_force = k * (temp_distance - l);
    temp_damper_force_x = c * (temp_vx2 - temp_vx1);
    temp_damper_force_y = c * (temp_vy2 - temp_vy1);

    temp_ax1 = (temp_spring_force * (temp_x2 - temp_x1)) / (m1 *
temp_distance) + temp_damper_force_x / m1;
    temp_ay1 = (temp_spring_force * (temp_y2 - temp_y1)) / (m1 *
temp_distance) + temp_damper_force_y / m1 - g;

    temp_ax2 = (temp_spring_force * (temp_x1 - temp_x2)) / (m2 *
temp_distance) - temp_damper_force_x / m2;
    temp_ay2 = (temp_spring_force * (temp_y1 - temp_y2)) / (m2 *
temp_distance) - temp_damper_force_y / m2 - g;

    k4x1 = temp_vx1;
    k4x2 = temp_vx2;
    k4y1 = temp_vy1;
    k4y2 = temp_vy2;
    k4vx1 = temp_ax1;
    k4vx2 = temp_ax2;
    k4vy1 = temp_ay1;
    k4vy2 = temp_ay2;

    % Update velocities and positions of m1
    vx1(i+1) = vx1(i) + dt*(k1vx1 + 2.0*k2vx1 + 2.0*k3vx1 + k4vx1)/6.0;
    vy1(i+1) = vy1(i) + dt*(k1vy1 + 2.0*k2vy1 + 2.0*k3vy1 + k4vy1)/6.0;

    x1(i+1) = x1(i) + dt*(k1x1 + 2.0*k2x1 + 2.0*k3x1 + k4x1)/6.0;
    y1(i+1) = y1(i) + dt*(k1y1 + 2.0*k2y1 + 2.0*k3y1 + k4y1)/6.0;

    % Update velocities and positions of m2
    vx2(i+1) = vx2(i) + dt*(k1vx2 + 2.0*k2vx2 + 2.0*k3vx2 + k4vx2)/6.0;
    vy2(i+1) = vy2(i) + dt*(k1vy2 + 2.0*k2vy2 + 2.0*k3vy2 + k4vy2)/6.0;
```

```
    x2(i+1) = x2(i) + dt*(k1x2 + 2.0*k2x2 + 2.0*k3x2 + k4x2)/6.0;
    y2(i+1) = y2(i) + dt*(k1y2 + 2.0*k2y2 + 2.0*k3y2 + k4y2)/6.0;

    % Update distance
    distance_between_masses(i+1) = sqrt((x2(i+1) - x1(i+1))^2 + (y2(i+1) -
 y1(i+1))^2);

    % Update energy
    kinetic_energy(i+1) = 0.5*m1*((vx1(i+1))^2 + (vy1(i+1))^2) +
 0.5*m2*((vx2(i+1))^2 + (vy2(i+1))^2);
    spring_pot_energy(i+1) = 0.5*k*((distance_between_masses(i+1) - l)^2);
    grav_pot_energy(i+1) = m1*g*y1(i+1) + m2*g*y2(i+1);
    total_energy(i+1) =  kinetic_energy(i+1) + spring_pot_energy(i+1) +
 grav_pot_energy(i+1);
end

% Plot total energy vs time graph
plot_energy(time, total_energy);

% Uncomment below line to plot x1, y1, x2, y2, vx1, vy1, vx2, vy2 in one plot
% plot_x_y(time, x1, y1, x2, y2, vx1, vy1, vx2, vy2);

function plot_energy(time, total_energy)
    figure;
    plot(time, total_energy, 'b');
    xlabel('t');
    ylabel('Total Energy');
    title('Total Energy vs. Time');
    ylim([0 2]);
end

function plot_x_y(time, x1, y1, x2, y2, vx1, vy1, vx2, vy2)
    figure;

    % x vs time plot
    subplot(2,2,1);
    plot(time, x1, 'b', time, x2, 'r');
    xlabel('t');
    ylabel('x');
    legend('m1', 'm2', 'Location','northwest');
    title('x vs. Time ');

    % y vs time plot
    subplot(2,2,2);
    plot(time, y1, 'b', time, y2, 'r');
    xlabel('t');
    ylabel('y');
    legend('m1', 'm2', 'Location','southwest');
    title('y vs. Time');

    % vx vs time plot
    subplot(2,2,3);
    plot(time, vx1, 'b', time, vx2, 'r');
    xlabel('t');
```

```
    ylabel('vx');
    legend('m1', 'm2', 'Location','northeast');
    title('vx vs. Time');

    % vy vs time plot
    subplot(2,2,4);
    plot(time, vy1, 'b', time, vy2, 'r');
    xlabel('t');
    ylabel('vy');
    legend('m1', 'm2', 'Location','northeast');
    title('vy vs. Time');
end
```



Published with MATLAB® R2023a