ODE – 004

```
clear
clc
close all
%Given rates
s_in =0.2; %Non-Infected Migration Rate
i_in =0.3; %Infected Migration Rate
b =0.15; %Death chance
p =0.03; %Immunity loss chance
%Additional rates needed but not given
a = 0.0002; %Infection chance
r = 0.2; %Recovery chance
%modeling equations
% dS/dt = s_in + pR - aSI
% dI/dt = aSI + i_in - brI
% dD/dt = bI
% dR/dt = rI - pR
%Convert to dydt
dydt = @(t,y) [(s_in +p*y(3) - a*y(1)*y(2));(a*y(1)*y(2) + i_in -
b*r*y(2));(r*y(2)-p*y(3));(b*y(2))];
t =  linspace (0,10,1000);
%Define starting values
S0 = 100;
I0 = 100;
R0 = 0;
D0 = 0;
%ode 45
[~ , y] = ode45(dydt,t,[S0 I0 R0 D0]);
%Parse Outputs
S = y(:,1);
I = y(:,2);
R = y(:,3);
D = y(:,4);
%plot s i r and d vs time
figure;
plot(t, S, 'b-', t, I, 'r-', t, R, 'g-', t, D, 'k-');
xlabel('Time');
ylabel('Population');
legend(["Susceptible (S)", "Infected (I)", "Recovered (R)", "Dead (D)"],
"Position", [0.2101 0.6425 0.3161, 0.1946])
title('SIR-D Model Simulation');
grid on;
```

Interpreting the results of the population over time as the a and r values are changed gives that the following holds for the relationships of S I R and D with each “constant” rate value (a and r).

For the a constant (Infection Chance)

S – Inversely Related

I- Direct Relationship

R- Direct Relationship

D- Direct Relationship

For the r constant (Recovery Chance)

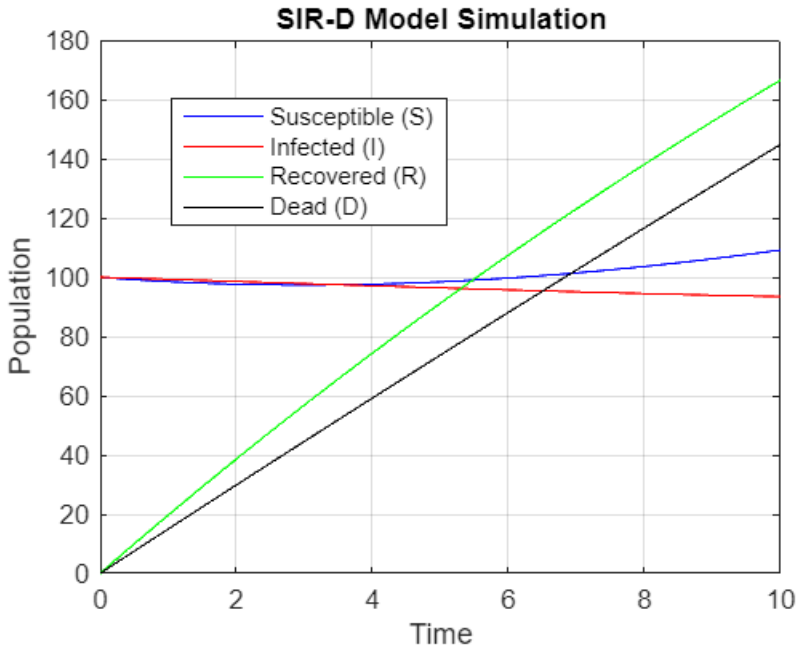
S – Direct Relationship

I- Direct Relationship

R- Direct Relationship

D- Inversely Related

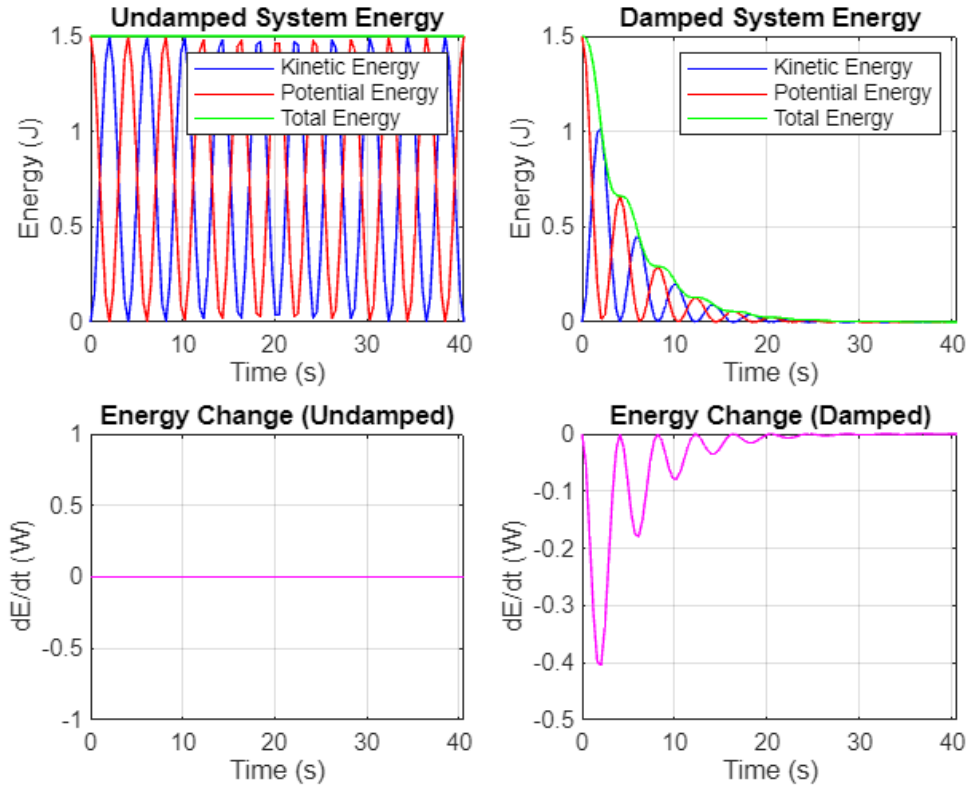
Below is one of the graphs generated that allowed me to observe these relationships



ODE – 005

```
clear
clc
close all
%Given params
m = 5;
c_damped = 1;
c_undamped = 0;
k = 3;
x0 = 1;
dxdt0 = 0;
% Time span
omega_n = sqrt(k/m);
T = 2*pi/omega_n;
tspan = linspace(0, 5*T,100);
% Absolute and relative tolerances
options = odeset('AbsTol', 1e-6, 'RelTol', 1e-6);
%ode 45 undamped
odefun_undamped = @(t, y) [y(2); -k/m*y(1)];
[t_undamped, y_undamped] = ode45(odefun_undamped, tspan, [x0; dxdt0],options);
x_undamped = y_undamped(:, 1);
dxdt_undamped = y_undamped(:, 2);
%energies for undamped system
KE_undamped = 0.5 * m * dxdt_undamped.^2;
PE_undamped = 0.5 * k * x_undamped.^2;
E_undamped = KE_undamped + PE_undamped;
dEdt_undamped = -c_undamped * dxdt_undamped.^2;
%ode45 damped
odefun_damped = @(t, y) [y(2); -k/m*y(1) - c_damped/m*y(2)];
[t_damped, y_damped] = ode45(odefun_damped, tspan, [x0; dxdt0],options);
x_damped = y_damped(:, 1);
dxdt_damped = y_damped(:, 2);
%energies for damped system
KE_damped = 0.5 * m * dxdt_damped.^2;
PE_damped = 0.5 * k * x_damped.^2;
E_damped = KE_damped + PE_damped;
dEdt_damped = -c_damped * dxdt_damped.^2;
% plots
figure;
subplot(2, 2, 1);
plot(t_undamped, KE_undamped, 'b', t_undamped, PE_undamped, 'r', t_undamped,
E_undamped, 'g');
xlabel('Time (s)');
ylabel('Energy (J)');
title('Undamped System Energy');
legend('Kinetic Energy', 'Potential Energy', 'Total Energy');
grid on;
```

```
subplot(2, 2, 3);
plot(t_undamped, dEdt_undamped, 'm');
xlabel('Time (s)');
ylabel('dE/dt (W)');
title('Energy Change (Undamped)');
grid on;
subplot(2, 2, 2);
plot(t_damped, KE_damped, 'b', t_damped, PE_damped, 'r', t_damped, E_damped,
'g');
xlabel('Time (s)');
ylabel('Energy (J)');
title('Damped System Energy');
legend('Kinetic Energy', 'Potential Energy', 'Total Energy');
grid on;
subplot(2, 2, 4);
plot(t_damped, dEdt_damped, 'm');
xlabel('Time (s)');
ylabel('dE/dt (W)');
title('Energy Change (Damped)');
grid on;
```



As can be seen with the energy vs time graphs above the damped spring equation does not hold all energy within the system, therefore it can be said that it does not obey the law of

conservation of energy (with the predication that the non-obeyance of the law is only true if just looking at the system and not when accounting for heat losses). The verification that the undamped case follows the law of CoE is best seen by looking at the dE/dt graph as it can be difficult to read the graph of Energy vs time given the multiple sine/cosine waves present on the graph.

ODE – 006

```
clear
clc
close all
%Given params
m = 25;
k = 10;
F0 = 30;
t_pulse_end = 1.5; % Renamed t1 for clarity in ODE function
x0 = 0;
dxdt0 = 0;
%Time vector
t = 0:0.01:10;
%Tolerances for ode45
options = odeset('AbsTol', 1e-6, 'RelTol', 1e-6);
% Critical damping coefficient
cc = 2*sqrt(m*k);
% Range of c values to test
num_c = 1000;
c_values = linspace(0, cc, num_c);
valid_c_values = [];
% Define the ode function
function dzdt = massode(t_c, z, m, k, c, F0, t_pulse_end)
    F_t = 0;
    if (t_c >= 0) && (t_c <= t_pulse_end)
        F_t = F0;
    end
    dzdt = [z(2); (F_t - k*z(1) - c*z(2))/m];
end
for i = 1:length(c_values)
    c = c_values(i);
    %ode45
    odefun = @(t_c, z) massode(t_c, z, m, k, c, F0, t_pulse_end);
    [~, y] = ode45(odefun, t, [x0; dxdt0], options);
    x = y(:, 1);
    dxdt = y(:, 2);
    % Check the requirements
    min_displacement = min(x);
    max_velocity = max(dxdt(dxdt > 0));
    condition1 = min_displacement > -0.5;
```

```
    condition2 = ~isempty(max_velocity) && max(max_velocity) > 0.72;
    condition3 = c < cc;
    if condition1 && condition2 && condition3
        valid_c_values = [valid_c_values, c];
    end
end
    c_min_valid = min(valid_c_values);
    c_max_valid = max(valid_c_values);
    fprintf('The range of c values that satisfy the conditions\nis approximately:
%g < c < %g', c_min_valid, c_max_valid);
```

```
The range of c values that satisfy the conditions
is approximately: 11.5539 < c < 30.0401
```

ODE – 007

```
clear all
clc
close
%Given params
m = 1.8;
L = 1.2;
rho = 0.02;
g = 3.72;
h = 18;
lift_constant = 2 * m * sqrt(g) / (rho * pi * L^2);
c_drag = 0.0012;
% PID controller parameters for each case
pid_params = [3 0 0;3 0.6 0;3 0.6 0.2];
% Time span and initial conditions
tspan = 0:0.1:100;
z0 = [0; 0; 0];
function dzdt = helicopter_motion(t, z, kp, kd, ki, m, A, c, g, h)
    y = z(1);
    v = z(2);
    Q = z(3);
    % Calculate the motor power P based on the PID controller
    P = kp * (h - y) - kd * v + ki * Q;
    % Define the system of ODEs
    dydt = v;
    dvdt = (8 * P) / (v * m + A) - (c * v^2) / m - g;
    dQdt = h - y;
    dzdt = [dydt; dvdt; dQdt];
end
%plot
figure;
```

```
hold on;
for i = 1:size(pid_params, 1)
    kp = pid_params(i, 1);
    kd = pid_params(i, 2);
    ki = pid_params(i, 3);
    %ode for each pid param
    odefun = @(t, z) helicopter_motion(t, z, kp, kd, ki, m, lift_constant,
c_drag, g, h);
    %ode45
    [t, z_sol] = ode45(odefun, tspan, z0);
    plot(t, z_sol(:, 1));
end
xlabel('Time (s)');
ylabel('Vertical Position (y) (m)');
title('Vertical Motion of Ingenuity by Case');
legend(["Case 1", "Case 2", "Case 3"], "Position", [0.6390 0.5085 0.2053,
0.1429])
grid on;
hold off;
```

