Numerical Method
Purdue University
Spring 2021

Homework 5

Jialei Wang

Problem 1

1. To apply the stretching transformation with $t = \tau b$, take the derivative of the transformation.

$$dt = d\tau \cdot b$$

Substitute the equation above to the differential equation.

$$\frac{dy}{dt} = f(t, y) \quad 0 \leq t \leq b$$

$$\frac{dy}{d\tau} \cdot \frac{1}{b} = f(\tau b, y) \quad 0 \leq \tau b \leq b$$

Since this is a new differential equation, the function will be rename from y to z.

$$\frac{dz}{d\tau} \cdot \frac{1}{b} = f(\tau b, z) \quad 0 \leq \tau \leq 1$$

The result of the both differential equation is the same.

$$\frac{dy}{dt} = \frac{dz}{d\tau}$$

Take the integral of both sides.

$$y(t) = \frac{1}{b} z(\tau)$$

2. The formula for Forward Euler method on both ODE can be written as:

$$y^+ = y + h_t f(t, y)$$

$$z^+ = z + h_\tau b f(\tau b, z)$$

In order for both method to produce the same result,

$$h_t = h_\tau b$$

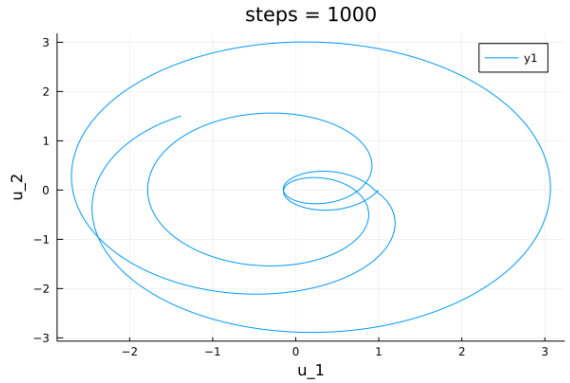
Problem 2

1. The 4th order RK method is shown below

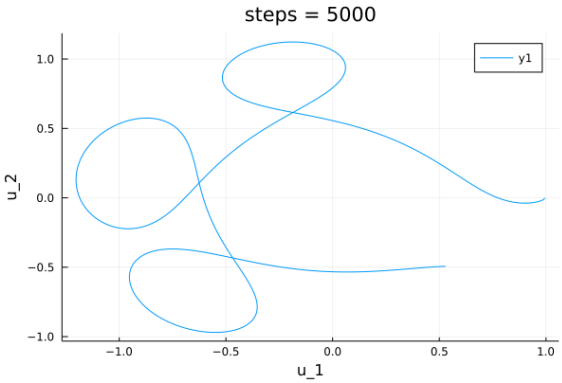
```
using LinearAlgebra
using Plots
using DelimitedFiles

function RK4(f::Function, t_start::Float64, t_end::Float64,
             y_init, steps::Int)
    y = zeros((length(y_init), steps))
    h = (t_end - t_start) / steps
    t = t_start
    y[:, 1] = y_init
    for i in 1:steps - 1
        k1 = f(t, y[:, i])
        k2 = f(t + 0.5 * h, y[:, i] .+ 0.5 .* h .* k1)
        k3 = f(t + 0.5 * h, y[:, i] .+ 0.5 .* h .* k2)
        k4 = f(t + h, y[:, i] .+ h .* k3)
        y[:, i + 1] = y[:, i] .+
                      h ./ 6. .* (k1 + 2 .* k2 + 2 .* k3 + k4)
        t += h
    end
    return y
end

function astronomy(t, y)
    mu = 0.012277471
    mu_h = 1 - mu
    result = zeros((length(y),))
    D1 = ((y[1] + mu)^2 + y[3]^2)^1.5
    D2 = ((y[1] - mu_h)^2 + y[3]^2)^1.5
    result[1] = y[2]
    result[2] = y[1] + 2 * y[4] - mu_h * (y[1] + mu) / D1 -
                mu * (y[1] - mu_h) / D2
    result[3] = y[4]
    result[4] = y[3] - 2 * y[2] - mu_h * y[3] / D1 -
                mu * y[3] / D2
    return result
end

y_init = zeros((4,))
y_init[1] = 0.994
y_init[2] = 0.0
y_init[3] = 0.0
y_init[4] = -2.00158510637908252240537862224
t_start = 0.0
t_end = 17.1
steps =
```

```
for steps in [1000, 5000, 10000, 50000, 100000]
    y = RK4(astronomy, t_start, t_end, y_init, steps)
    display(plot(y[1,:], y[3,:],
                 xlabel="u_1",
                 ylabel="u_2",
                 title=string("steps = ", steps)))
    png(string("problem2_", steps, ".png"))
end
```

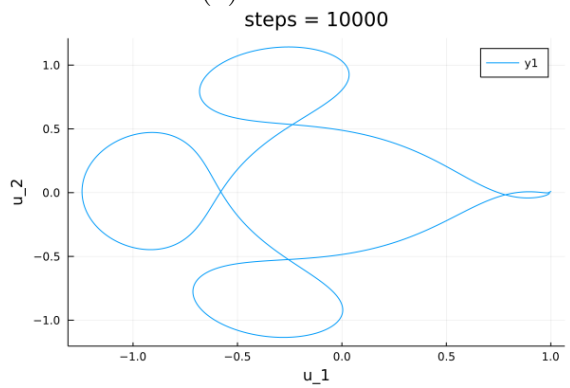
2. The result of the solution of the ODE using the above method with step size of 1000, 5000, 10000, 50000, and 100000.



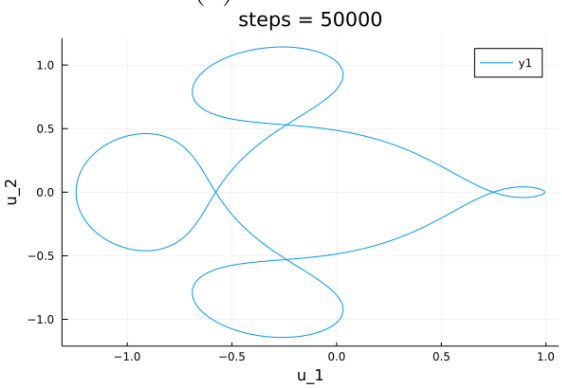
(a) $N = 1000$



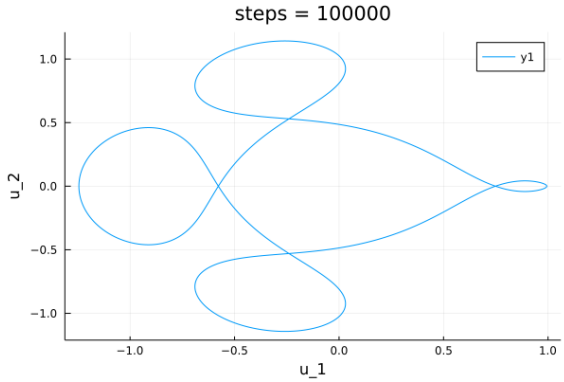
(b) $N = 5000$



(c) $N = 10000$



(d) $N = 50000$



(e) $N = 100000$

Figure 1: Result of the ODEs using 4th order RK Method with different step sizes

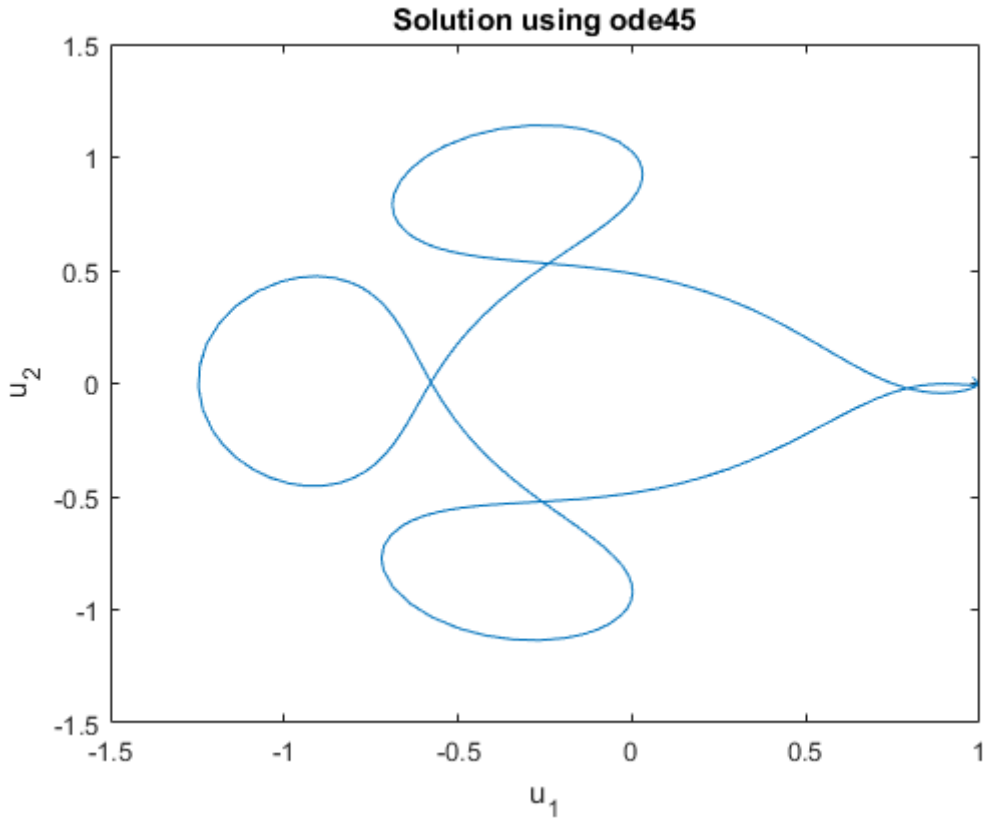
As the number of steps increases, solution approaches the exact solution. When the

steps is 1000, it does not look close to the exact solution. When the step step size is 5000 and 10000, the solution starts to take the shape of the exact solution. Lastly, there is no visible difference between the two solutions.

3. The program that uses the *ode45* function to solve the problem is given below:

```
y_init = [0.994; 0.0; 0.0; -2.00158510637908252240537862224];
tspan = [0.0, 17.1];
[t, y] = ode45(@threebody, tspan, y_init);
figure
plot(y(:,1), y(:, 3))
xlabel('u_1')
ylabel('u_2')
title('Solution using ode45')
```

The result of using standard integration software is shown below, the result is the same as the results above.



Problem 3

1. The first step is to take the derivative of the equation:

$$u(t) - \epsilon \sin u(t) - t = 0$$

$$u'(t) - u'(t)\epsilon \cos u(t) - 1 = 0$$

$$u'(t) = \frac{1}{1 - \epsilon \cos u(t)}$$

The second derivative of the function $x(t)$ and $y(t)$ is calculated.

$$x(t) = \cos u(t) - \epsilon$$

$$x'(t) = -\sin u(t) \cdot u'(t) = \frac{-\sin u(t)}{1 - \epsilon \cos u(t)}$$

$$\begin{aligned} x''(t) &= \frac{-u'(t)\cos u(t)(1 - \epsilon \cos u(t)) + \epsilon u'(t)\sin^2 u(t)}{(1 - \epsilon \cos u(t))^2} \\ &= \frac{-\cos u(t) + \epsilon \cos^2 u(t) + \epsilon \sin^2 u(t)}{(1 - \epsilon \cos u(t))^3} \\ &= \frac{-x(t)}{(1 - \epsilon \cos u(t))^3} \end{aligned}$$

$$y(t) = \sqrt{1 - \epsilon^2} \sin u(t)$$

$$y'(t) = \sqrt{1 - \epsilon^2} \frac{\cos u(t)}{1 - \epsilon \cos u(t)}$$

$$\begin{aligned} y''(t) &= \frac{-\sqrt{1 - \epsilon^2} \sin u(t)}{(1 - \epsilon \cos u(t))^3} \\ &= \frac{-y(t)}{(1 - \epsilon \cos u(t))^3} \end{aligned}$$

Also, the equation for $x(t)$ and $y(t)$ can be substituted into $r^2 = x^2 + y^2$.

$$\begin{aligned} r^2 = x^2 + y^2 &= (\cos u(t) - \epsilon)^2 + (\sqrt{1 - \epsilon^2} \sin u(t))^2 \\ &= \cos^2 u(t) - 2\epsilon \cos u(t) + \epsilon^2 + \sin^2 u(t) - \epsilon^2 \sin^2 u(t) \\ &= 1 - 2\cos u(t)\epsilon - \epsilon^2 - \epsilon \sin^2 u(t) \\ &= 1 - 2\cos u(t)\epsilon + \epsilon^2 \cos^2 u(t) \\ &= (1 - \epsilon \cos u(t))^2 \end{aligned}$$

$$r = 1 - \epsilon \cos u(t)$$

By substitute $r(t)$ into $x''(t)$ and $y''(t)$,

$$x''(t) = \frac{x(t)}{r^3}$$

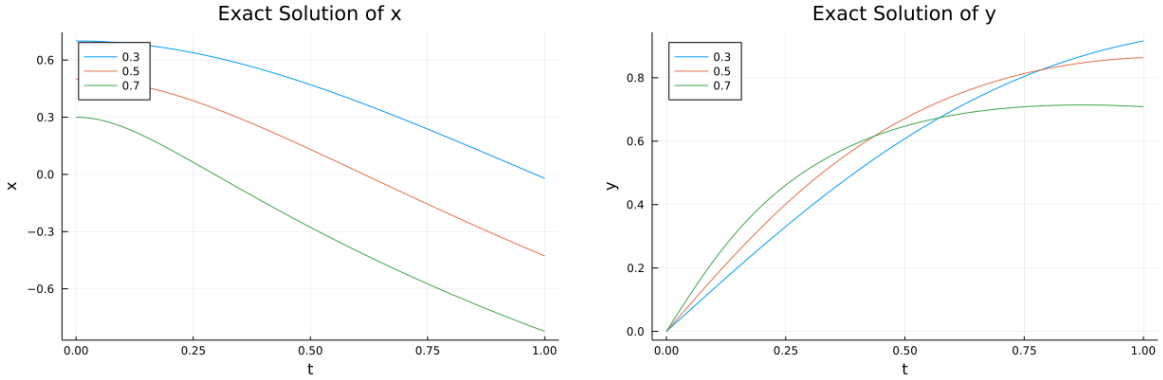
$$y''(t) = \frac{y(t)}{r^3}$$

2. The program that solves for the exact solution is:

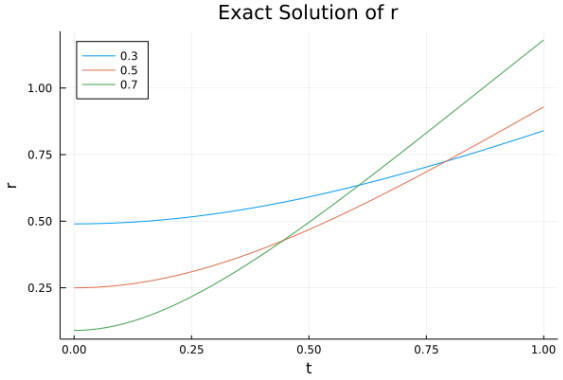
```
function solve_u(e::BigFloat, t::BigFloat, eps::BigFloat)
    u_n = t
    u_n_next = u_n - (u_n - e * sin(u_n) - t) *
            (1 - e * cos(u_n))
    while (abs(u_n - u_n_next) > eps)
        u_n = u_n_next
        u_n_next = u_n - (u_n - e * sin(u_n) - t) *
                (1 - e * cos(u_n))
    end
    return float(u_n_next)
end

function solve_solution(e::Float64, tspan, N)
    t = collect(LinRange(tspan[1], tspan[2], N))
    x = zeros((N,))
    y = zeros((N,))
    r = zeros((N,))
    for i in 1:N
        u = solve_u(BigFloat(e), BigFloat(t[i]),
                    BigFloat(eps(Float64)))
        x[i] = cos(u) - e
        y[i] = sqrt(1 - e ^ 2.) * sin(u)
        r[i] = x[i] ^ 2. + y[i] ^ 2.
    end
    return x, y, r
end

tspan = (0.0, 1.0)
N = 1000
t = collect(LinRange(tspan[1], tspan[2], N))
x = zeros((N, 3))
y = zeros((N, 3))
r = zeros((N, 3))
for (i, e) in enumerate([0.3, 0.5, 0.7])
    x[:,i], y[:,i], r[:,i] = solve_solution(e, tspan, N)
end
display(plot(t, x,
            label=["0.3" "0.5" "0.7"],
            xlabel="t",
            ylabel="x",
            title="Exact Solution of x",
            legend=:topleft))
png("exact_x.png")

display(plot(t, y,
            label=["0.3" "0.5" "0.7"],
            xlabel="t",
```

```
                ylabel="y",
                title="Exact Solution of y",
                legend=:topleft))
png("exact_y.png")

display(plot(t, r,
                label=["0.3" "0.5" "0.7"],
                xlabel="t",
                ylabel="r",
```

The plot of exact solution is shown below:



(a) Exact solution of x (b) Exact solution of y



(c) Exact solution of r

Figure 2: Exact solution of the problem for $\epsilon = 0.3, 0.5, 0.7$

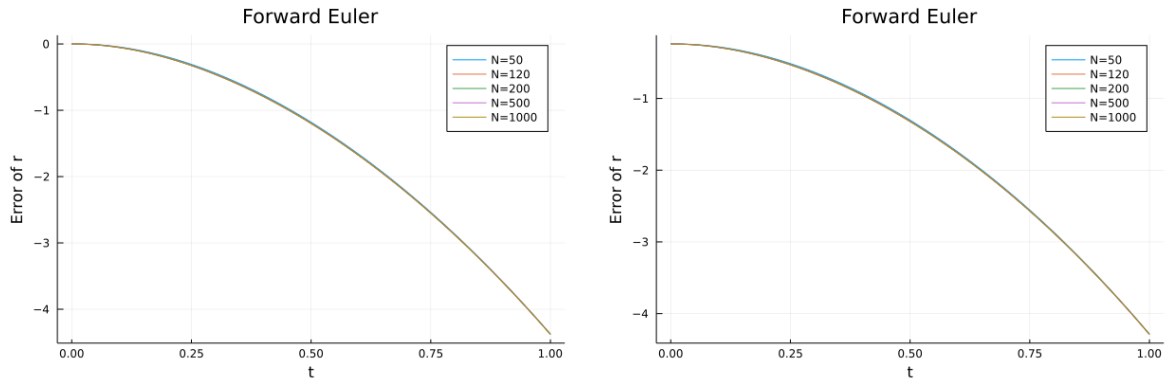
3. The function that calculates Forward and Backward Euler solution of the problem.

```
function jacobian_g(y, h)
    J = convert(Matrix{BigFloat},Matrix(1.0I, 4, 4))
    D = (y[1] ^ 2 + y[3] ^ 2) ^ 4
    J[1, 2] = -h
    J[3, 4] = -h
    J[2, 1] = -(y[3] ^ 2 - 5 * y[1] ^ 2) / D * h
    J[2, 3] = 6 * y[3] * y[1] / D * h
    J[4, 1] = 6 * y[3] * y[1] / D * h
    J[4, 3] = -(y[1] ^ 2 - 5 * y[3] ^ 2) / D * h
    return J
```

```
end


function f(t, y)
    y_out = zeros((4,))
    r = y[1] ^ 2 + y[3] ^ 2
    y_out[1] = y[2]
    y_out[2] = y[1] / r ^ 3
    y_out[3] = y[4]
    y_out[4] = y[3] / r ^ 3
    return y_out
end

function g(y_plus, y, h)
    output = convert(Vector{BigFloat}, zeros((4,)))
    r = (y_plus[1] ^ 2 + y_plus[3] ^ 2) ^ 3
    output[1] = y_plus[1] - h * y_plus[2] - y[1]
    output[2] = y_plus[2] - h * y_plus[1] / r - y[2]
    output[3] = y_plus[3] - h * y_plus[4] - y[3]
    output[4] = y_plus[4] - h * y_plus[3] / r - y[4]
    return output
end

function forward_euler(f::Function, t_start::Float64,
                       t_end::Float64, y_init, steps::Int)
    y = zeros((length(y_init), steps))
    h = (t_end - t_start) / steps
    t = t_start
    y[:, 1] = y_init
    for i in 1:steps-1
        y[:, i + 1] = y[:, i] + h .* f(t, y[:, i])
        t += h
    end
    return y
end

function backward_euler(f::Function, t_start::Float64,
                       t_end::Float64, y_init, steps::Int)
    y = convert(Matrix{BigFloat}, zeros((length(y_init), steps)))
    h = BigFloat((t_end - t_start) / steps)
    t = t_start
    y[:, 1] = convert(Vector{BigFloat}, y_init)
    threshold = BigFloat(eps(Float64))
    for i in 1:steps-1
        # use Newton's method to solve the system of
        # non-linear equations from the Backward Euler
        # method formula.
        y_cur = y[:, i]
```

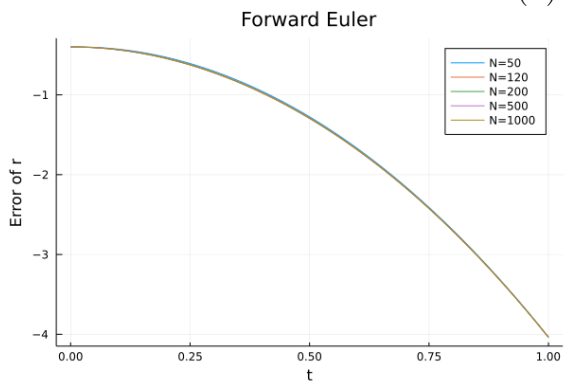
```
            y_next = y_cur - inv(jacobian_g(y_cur, h)) *
                     g(y_cur, y[:, i], h)
            error = abs.(y_cur .- y_next)
            while (any(x->x>threshold, error))
                y_cur = y_next
                y1 = inv(jacobian_g(y_cur, h))
                y2 = g(y_cur, y[:, i], h)
                y_diff = y1 * y2
                y_next = y_cur - y_diff
                error = abs.(y_cur .- y_next)
            end
            y[:, i+1] = y_next
        end
        return float(y)
    end
```

4. The error of using both Forward and Backward Euler with time step of 50, 120, 200, 500, and 1000 is shown below



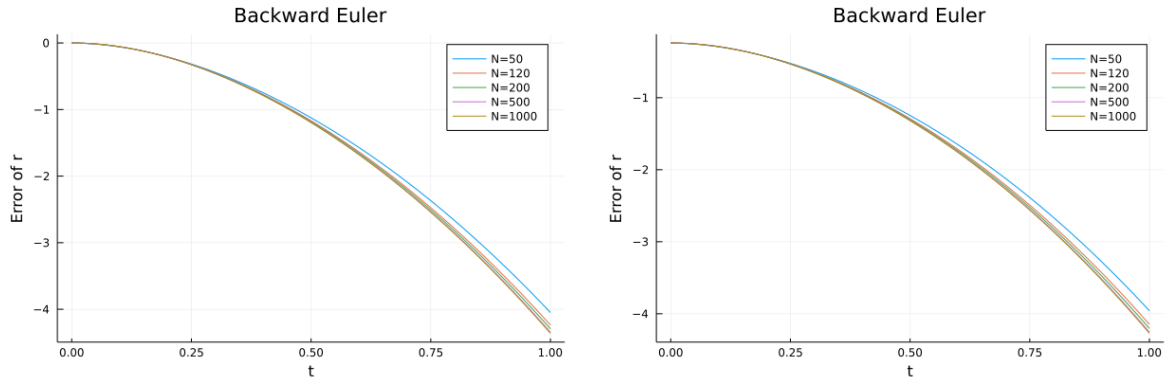
(a) $\epsilon = 0.3$

(b) $\epsilon = 0.5$



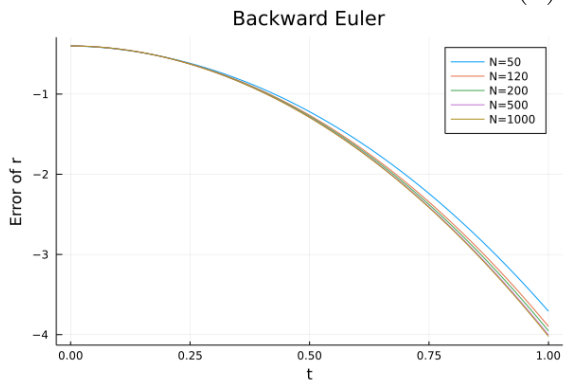
(c) $\epsilon = 0.7$

Figure 3: The error of using Forward Euler method with different time steps and ϵ



(a) $\epsilon = 0.3$

(b) $\epsilon = 0.5$



(c) $\epsilon = 0.7$

Figure 4: The error of using Backward Euler method with different time steps and ϵ

From the error plots above, both method shows the error increases at time increases. At various step sizes, there is no significant differences between the error. In conclusion, both Forward and Backward Euler is not fit for solving this set of ODEs.

Problem 4

The modified code is shown below.

```
using LinearAlgebra

"""
This function will use Hune's method to simulate the Raptors problem
"""
function simulate_raptors_Hune(angle)
    vhuman=6.0
    vraptor0=10.0 # the slow raptor velocity in m/s
    vraptor=15.0 #

    raptor_distance = 20.0

    raptor_min_distance = 0.2 # a raptor within 20 cm can attack
    tmax=10.0 # the maximum time in seconds
    nsteps=1000

    # initial positions
    h = [0.0,0.0]
    r0 = [1.0,0.0]*raptor_distance
    r1 = [-0.5,sqrt(3.)/2.]*raptor_distance
    r2 = [-0.5,-sqrt(3.)/2.]*raptor_distance

    # how much time el
    dt = tmax/nsteps
    t = 0.0

    hhist = zeros(2,nsteps+1)
    r0hist = zeros(2,nsteps+1)
    r1hist = zeros(2,nsteps+2)
    r2hist = zeros(2,nsteps+2)

    hhist[:,1] = h
    r0hist[:,1] = r0
    r1hist[:,1] = r1
    r2hist[:,1] = r2

    tend = tmax

    """
    This function will compute the derivatives of the
    positions of the human and the raptors
    """
    function compute_derivatives(angle,h,r0,r1,r2)
        dh = [cos(angle),sin(angle)]*vhuman
        dr0 = (h-r0)/norm(h-r0)*vraptor0
```

```
        dr1 = (h-r1)/norm(h-r1)*vraptor
        dr2 = (h-r2)/norm(h-r2)*vraptor
        return dh, dr0, dr1, dr2
    end

    for i=1:nsteps
        dh_1, dr0_1, dr1_1, dr2_1 = compute_derivatives(angle,h,r0,r1,r2)
        dh_2, dr0_2, dr1_2, dr2_2 = compute_derivatives(angle,
                                                        h + dt * dh_1,
                                                        r0 + dt * dr0_1,
                                                        r1 + dt * dr1_1,
                                                        r2 + dt * dr2_1)

        h += 0.5 * dt * (dh_1 + dh_2)
        r0 += 0.5 * dt * (dr0_1 + dr0_2)
        r1 += 0.5 * dt * (dr1_1 + dr1_2)
        r2 += 0.5 * dt * (dr2_1 + dr2_2)
        t += dt

        hhist[:,i+1] = h
        r0hist[:,i+1] = r0
        r1hist[:,i+1] = r1
        r2hist[:,i+1] = r2

        if norm(r0-h) <= raptor_min_distance ||
            norm(r1-h) <= raptor_min_distance ||
            norm(r2-h) <= raptor_min_distance

            # truncate the history
            hhist = hhist[:,1:i+1]
            r0hist = r0hist[:,1:i+1]
            r1hist = r1hist[:,1:i+1]
            r2hist = r2hist[:,1:i+1]
            tend = t
            break
        end
    end
    return tend
end
```

The result of the plot is shown below

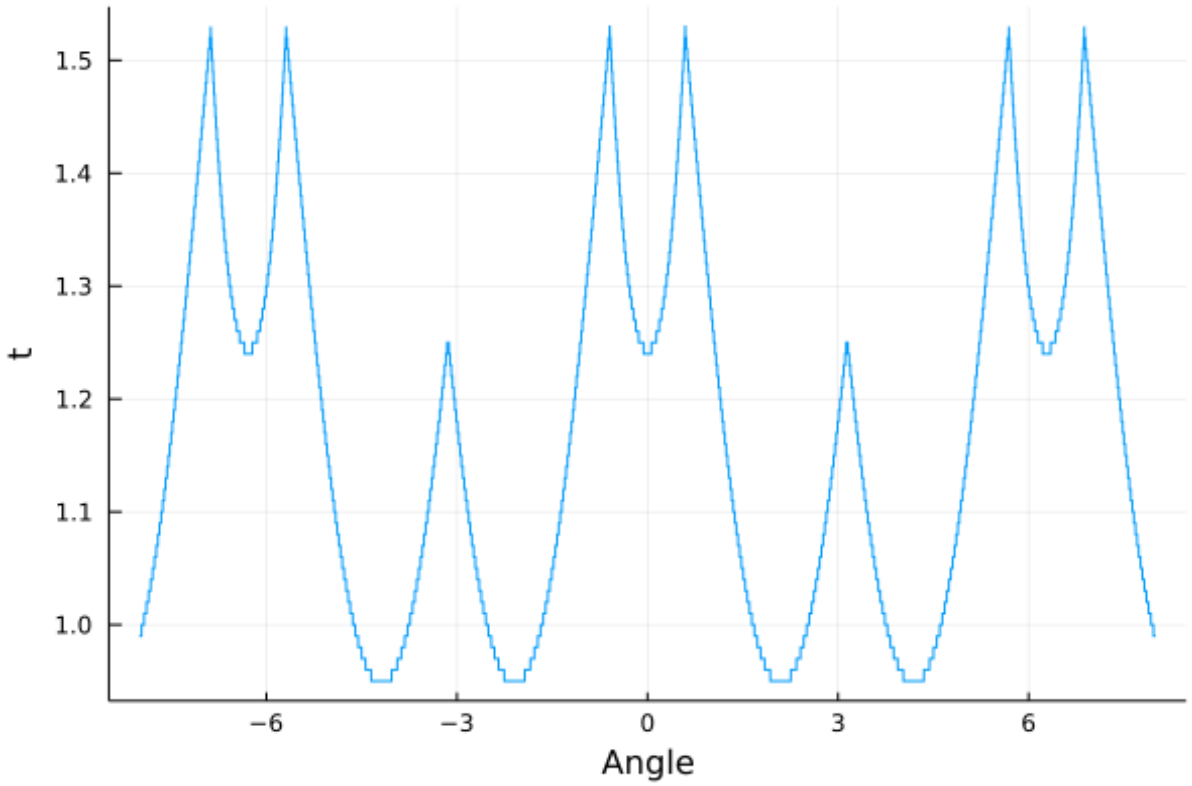


Figure 5: The result plot of the Raptor Problem

From the result of the plot, we can observe that the result of periodic with respect to the angle which is in radian. The period of the result is 2π and centered at 0 rad. At 0 rad, the result is around 1.25, and it increases to the global maximum at around -0.6 and 0.6 rad where the global maximum is around 1.53. The result then decrease to global minimum at -2.6 and 2.6 rad where the global minimum is around 0.93. The result then finally increase to 1.25 at π rad.

Problem 0

I have worked on this homework on my own.