```julia
#Problem 4

#Definitions
filename = "./Exam1/aerosonde.mat"
aircraft_parameters = CustomAircraftParameters(filename)
location = "Exam1/plots"
save_plots = true
trim_Va = 25.0
trim_γ = 0.0
trim_h = 200.0

#Linear Models
lm_filename = "./Exam1/AerosondeLinearModel.mat"
data = matread(lm_filename)
Alat = data["Alat"]
Blat = data["Blat"]
Along = data["Alon"]
Blong = data["Blon"]
trim_state_vector = data["aircraft_state0"]
trim_control_vector = data["control_surfaces0"]

function CustomAircraftParameters(filename)
    file_location = joinpath(pwd(),filename)
    matlab_data = matread(file_location)
    matlab_params = matlab_data["aircraft_parameters"]
    julia_params = []
    for k in fieldnames(AircraftParameters)
        if(string(k)=="K1")
            push!(julia_params, 0.0)
        else
            push!(julia_params, matlab_params[string(k)])
        end
    end
    return AircraftParameters(julia_params...)
end

function P3dynamics!(du,u,p,t)
    Amatrix = p[1]
    x_dot = Amatrix*u
    for i in 1:length(u)
        du[i] = x_dot[i]
    end
end

function P3ODEsimulate(dynamics, initial_state, time_interval, extra_parameters, save_at_value=1.0)
    prob = ODEProblem(dynamics,initial_state,time_interval,extra_parameters)
    sol = DifferentialEquations.solve(prob,saveat=save_at_value)
    lm_ubar, lm_αbar, lm_qbar, lm_θbar, lm_hbar = [],[],[],[],[]
    for i in 1:length(sol.u)
        push!(lm_ubar,sol.u[i][1])
        push!(lm_αbar,sol.u[i][2])
        push!(lm_qbar,sol.u[i][3])
        push!(lm_θbar,sol.u[i][4])
        push!(lm_hbar,sol.u[i][5])
    end
    return [lm_ubar, lm_αbar, lm_qbar, lm_θbar, lm_hbar]
end

function plotP3(obs,location,save_plots=true)
    plots_location = joinpath(pwd(),location)
    lm_ubar, lm_αbar, lm_qbar, lm_θbar, lm_hbar = obs[1], obs[2], obs[3], obs[4], obs[5]
    t = 1:length(lm_ubar)

    p1 = plot(t, lm_ubar, xlabel="Time(s)", ylabel="ubar (m/s)", label="Linear")
    if(save_plots)
        savefig(plots_location*"/p3_1.png")
    end

    p2 = plot(t, lm_αbar, xlabel="Time(s)", ylabel="αbar (radians)", label="Linear")
    if(save_plots)
        savefig(plots_location*"/p3_2.png")
    end

    p3 = plot(t, lm_qbar, xlabel="Time(s)", ylabel="qbar (radians/s)", label="Linear")
    if(save_plots)
```

```julia
        savefig(plots_location*"/p3_3.png")
    end

    p4 = plot(t, lm_θbar, xlabel="Time(s)", ylabel="θbar (radians)", label="Linear")
    if(save_plots)
        savefig(plots_location*"/p3_4.png")
    end

    p5 = plot(t, lm_hbar, xlabel="Time(s)", ylabel="hbar (m)", label="Linear")
    if(save_plots)
        savefig(plots_location*"/p3_5.png")
    end
end

function calculate_xlon_states(aircraft_states, trim_state)
    ubar, αbar, qbar, θbar, hbar = [],[],[],[],[]
    utrim = trim_state.u
    qtrim = trim_state.q
    θtrim = trim_state.θ
    htrim = trim_state.z
    trim_wind_angles = AirRelativeVelocityVectorToWindAngles(trim_state[7:9])
    αtrim = trim_wind_angles.α
    for i in 1:length(aircraft_states)
        udev = aircraft_states[i].u - utrim
        push!(ubar,udev)
        wind_angles = AirRelativeVelocityVectorToWindAngles(aircraft_states[i][7:9])
        αdev = wind_angles.α - αtrim
        push!(αbar, αdev)
        qdev = aircraft_states[i].q - qtrim
        push!(qbar,qdev)
        θdev = aircraft_states[i].θ - θtrim
        θdev = rem(θdev, 2*pi)
        # θdev =
        push!(θbar,θdev)
        hdev = aircraft_states[i].z - htrim
        push!(hbar,hdev)
    end
    return ubar, αbar, qbar, θbar, hbar
end

function plotP4(lm_obs,nl_obs,location,problem_num,save_plots = true)
    plots_location = joinpath(pwd(),location)
    lm_ubar, lm_αbar, lm_qbar, lm_θbar, lm_hbar = lm_obs[1], lm_obs[2], lm_obs[3], lm_obs[4], lm_obs[5]
    nl_ubar, nl_αbar, nl_qbar, nl_θbar, nl_hbar = nl_obs[1], nl_obs[2], nl_obs[3], nl_obs[4], nl_obs[5]
    t = 1:length(lm_ubar)

    p1 = plot(t, lm_ubar, xlabel="Time(s)", ylabel="ubar (m/s)", label="Linear")
    plot!(t,nl_ubar, label="Non Linear")
    if(save_plots)
        savefig(plots_location*"/"*string(problem_num)*"_1.png")
    end

    p2 = plot(t, lm_αbar, xlabel="Time(s)", ylabel="αbar (radians)", label="Linear")
    plot!(t,nl_αbar,label="Non Linear")
    if(save_plots)
        savefig(plots_location*"/"*string(problem_num)*"_2.png")
    end

    p3 = plot(t, lm_qbar, xlabel="Time(s)", ylabel="qbar (radians/s)", label="Linear")
    plot!(t,nl_qbar,label="Non Linear")
    if(save_plots)
        savefig(plots_location*"/"*string(problem_num)*"_3.png")
    end

    p4 = plot(t, lm_θbar, xlabel="Time(s)", ylabel="θbar (radians)", label="Linear")
    plot!(t,nl_θbar,label="Non Linear")
    if(save_plots)
        savefig(plots_location*"/"*string(problem_num)*"_4.png")
    end

    p5 = plot(t, lm_hbar, xlabel="Time(s)", ylabel="hbar (m)", label="Linear")
    plot!(t,nl_hbar,label="Non Linear")
    if(save_plots)
        savefig(plots_location*"/"*string(problem_num)*"_5.png")
    end
```

```julia
end



#=
Part 1)
=#
reduced_Along = Along#[1:4,1:4]
long_eigenvals, long_eigenvectors = eigen(reduced_Along)

reduced_Alat = Alat#[1:4,1:4]
lat_eigenvals, lat_eigenvectors = eigen(reduced_Alat)

#Short Period Mode
short_natural_frequency = get_natural_frequency(long_eigenvals[1])
short_damping_ratio = get_damping_ratio(long_eigenvals[1])
#Phugoid Mode
phugoid_natural_frequency = get_natural_frequency(long_eigenvals[3])
phugoid_damping_ratio = get_damping_ratio(long_eigenvals[3])
#Roll Mode
roll_natural_frequency = get_natural_frequency(lat_eigenvals[1])
roll_damping_ratio = get_damping_ratio(lat_eigenvals[1])
#Dutch Roll Mode
dutch_roll_natural_frequency = get_natural_frequency(lat_eigenvals[2])
dutch_roll_damping_ratio = get_damping_ratio(lat_eigenvals[2])
#Spiral Mode
spiral_natural_frequency = get_natural_frequency(lat_eigenvals[4])
spiral_damping_ratio = get_damping_ratio(lat_eigenvals[4])

println("")
println("For Short Period Mode : ")
println("Damping Ratio : ", short_damping_ratio)
println("Natural Frequency : ", short_natural_frequency)
println("")
println("For Phugoid Mode : ")
println("Damping Ratio : ", phugoid_damping_ratio)
println("Natural Frequency : ", phugoid_natural_frequency)
println("")

#=
Part 2)
=#
short_period_vector = long_eigenvectors[:,1]
phugoid_vector = long_eigenvectors[:,3]
println("For Short Period Mode : ")
println("Eigenvalue : ", long_eigenvals[1])
println("Eigenvector : ", short_period_vector)
println("")
println("For Phugoid Mode : ")
println("Eigenvalue : ", long_eigenvals[3])
println("Eigenvector : ", phugoid_vector)
println("")


#=
Part 3)
=#
initial_perturbation_vector = [x.re for x in phugoid_vector]
curr_pitch_per = initial_perturbation_vector[4]
initial_vector = initial_perturbation_vector*(3*pi/curr_pitch_per/180)

TI = [0.0,25000.0]
extra_params = [Along]
lm_state_values = P3ODEsimulate(P3dynamics!, initial_vector, TI, extra_params)
plotP3(lm_state_values, location)


#=
Part 4)
=#
start_state = vec(trim_state_vector)
wind_angles = AirRelativeVelocityVectorToWindAngles(start_state[7:9])
α = wind_angles.α
perturbation_vec = initial_vector
udev = perturbation_vec[1]
```

```
wdev = trim_Va*cos(α)*perturbation_vec[2]
qdev = perturbation_vec[3]
θdev = perturbation_vec[4]
hdev = -perturbation_vec[5]

perturbed_state = start_state + [0.0,0.0,hdev,0.0,θdev,0.0,udev,0.0,wdev,0.0,qdev,0.0]
control_input = vec(trim_control_vector)
wind_inertial = [0.0,0.0,0.0]
time_values = [i for i in 0:TI[2]]
extra_params = [control_input, wind_inertial, aircraft_parameters]
trajectory_states = simulate(aircraft_dynamics!, perturbed_state, TI, extra_params)
nl_state_values = calculate_xlon_states(trajectory_states, AircraftState(start_state...))
plotP4(lm_state_values,nl_state_values,location,"p4")

#=
Part 5)
=#
start_state = vec(trim_state_vector)
wind_angles = AirRelativeVelocityVectorToWindAngles(start_state[7:9])
α = wind_angles.α
perturbation_vec = initial_vector*20/3
udev = perturbation_vec[1]
wdev = trim_Va*cos(α)*perturbation_vec[2]
qdev = perturbation_vec[3]
θdev = perturbation_vec[4]
hdev = perturbation_vec[5]

perturbed_state = start_state + [0.0,0.0,hdev,0.0,θdev,0.0,udev,0.0,wdev,0.0,qdev,0.0]
control_input = vec(trim_control_vector)
wind_inertial = [0.0,0.0,0.0]
time_values = [i for i in 0:TI[2]]
extra_params = [control_input, wind_inertial, aircraft_parameters]
trajectory_states = simulate(aircraft_dynamics!, perturbed_state, TI, extra_params)
nl_state_values = calculate_xlon_states(trajectory_states, AircraftState(start_state...))
plotP4(lm_state_values,nl_state_values,location,"p5")
control_array = [AircraftControl(control_input...) for i in 1:length(trajectory_states)]
PlotSimulation(time_values, trajectory_states, control_array , location, save_plots)
```