```julia
#Problem 1

function GetLateralEOMCoefficients(Γ::Array{Float64},ap::AircraftParameters)
    #For roll rate p
    Cp0 = Γ[3]*ap.Cl0 + Γ[4]*ap.Cn0
    Cpβ = Γ[3]*ap.Clbeta + Γ[4]*ap.Cnbeta
    Cpp = Γ[3]*ap.Clp + Γ[4]*ap.Cnp
    Cpr = Γ[3]*ap.Clr + Γ[4]*ap.Cnr
    Cpδa = Γ[3]*ap.Clda + Γ[4]*ap.Cnda
    Cpδr = Γ[3]*ap.Cldr + Γ[4]*ap.Cndr

    #For yaw rate r
    Cr0 = Γ[4]*ap.Cl0 + Γ[8]*ap.Cn0
    Crβ = Γ[4]*ap.Clbeta + Γ[8]*ap.Cnbeta
    Crp = Γ[4]*ap.Clp + Γ[8]*ap.Cnp
    Crr = Γ[4]*ap.Clr + Γ[8]*ap.Cnr
    Crδa = Γ[4]*ap.Clda + Γ[8]*ap.Cnda
    Crδr = Γ[4]*ap.Cldr + Γ[8]*ap.Cndr

    CoefficientsDict = Dict(
                            "Cp0" => Cp0,
                            "Cpβ" => Cpβ,
                            "Cpp" => Cpp,
                            "Cpr" => Cpr,
                            "Cpδa" => Cpδa,
                            "Cpδr" => Cpδr,
                            "Cr0" => Cr0,
                            "Crβ" => Crβ,
                            "Crp" => Crp,
                            "Crr" => Crr,
                            "Crδa" => Crδa,
                            "Crδr" => Crδr
                        )

    return CoefficientsDict
end

function GetLongitudinalEOMCoefficients(tc::AircraftControl, α::Float64,ap::AircraftParameters)

    CLtrim = ap.CL0 + (ap.CLalpha*α) + (ap.CLde*tc.δe)
    CDtrim = ap.CDmin + (ap.K * (CLtrim - ap.CLmin)^2)
    CXtrim = ( CLtrim*sin(α) ) - ( CDtrim*cos(α) )
    CZtrim = ( -CLtrim*cos(α) ) - ( CDtrim*sin(α) )

    δCDδCL = 2*ap.K*(CLtrim-ap.CLmin)
    CDα = ap.CLalpha*δCDδCL
    CDq = ap.CLq*δCDδCL
    CDδe = ap.CLde*δCDδCL

    CXα = -CDα*cos(α) + CDtrim*sin(α) + ap.CLalpha*sin(α) + CLtrim*cos(α)
    CZα = -CDα*sin(α) - CDtrim*cos(α) - ap.CLalpha*cos(α) + CLtrim*sin(α)

    CXq = -CDq*cos(α) + ap.CLq*sin(α)
    CZq = -CDq*sin(α) - ap.CLq*cos(α)

    CXδe = -CDδe*cos(α) + ap.CLde*sin(α)
    CZδe = -CDδe*sin(α) - ap.CLde*cos(α)

    CoefficientsDict = Dict(
                            "CLtrim" => CLtrim,
                            "CDtrim" => CDtrim,
                            "CXtrim" => CXtrim,
                            "CZtrim" => CZtrim,
                            "CXα" => CXα,
                            "CZα" => CZα,
                            "CXq" => CXq,
                            "CZq" => CZq,
                            "CXδe" => CXδe,
                            "CZδe" => CZδe
                        )

    return CoefficientsDict
end

function GetLinearizedModel(::Type{LateralAircraftState},ts::AircraftState,tc::AircraftControl,ap::AircraftParameters)

    ρ = stdatmo(-ts.z)
    S = ap.S
    b = ap.b
    m = ap.m
    AirSpeedVector = [ts.u, ts.v, ts.w]  #because it has been given that wind speed is zero.
    wind_angles = AirRelativeVelocityVectorToWindAngles(AirSpeedVector)
    Va = wind_angles.Va
    β = wind_angles.β
    af_term = Va*cos(β)
    Q = 0.25*ρ*Va*S*b/m
    Γ = GetGammaValues(ap)
    lc = GetLateralEOMCoefficients(Γ,ap)

    A11 = 0.25*ρ*S*b*ts.v*( (ap.CYp*ts.p) + (ap.CYr*ts.r) )/(m*Va)
    A11 += ρ*S*ts.v*(ap.CY0 + ap.CYbeta*β + ap.CYda*tc.δa + ap.CYdr*tc.δr)/(m)
    A11 += 0.5*ρ*S*ap.CYbeta*sqrt(ts.u^2 + ts.w^2)/m
    A12 = ts.w + Q*ap.CYp
    A12 = A12/af_term
    A13 = -ts.u + Q*ap.CYr
    A13 = A13/af_term
    A14 = ap.g*cos(ts.θ)*cos(ts.ϕ)
```

```julia
    A14 = A14/af_term
    A15 = 0.0

    A21 = 0.25*ρ*S*b*b*ts.v*( (lc["Cpp"]*ts.p) + (lc["Cpr"]*ts.r) )/(Va)
    A21 += ρ*S*b*ts.v*(lc["Cp0"] + lc["Cpβ"]*β + lc["Cpδa"]*tc.δa + lc["Cpδr"]*tc.δr)
    A21 += 0.5*ρ*S*b*lc["Cpβ"]*sqrt(ts.u^2 + ts.w^2)
    A21 = A21*af_term
    A22 = (Γ[1]*ts.q) + (Q*b*m*lc["Cpp"])
    A23 = (-Γ[2]*ts.q) + (Q*b*m*lc["Cpr"])
    A24 = 0.0
    A25 = 0.0

    A31 = 0.25*ρ*S*b*b*ts.v*( (lc["Crp"]*ts.p) + (lc["Crr"]*ts.r) )/(Va)
    A31 += ρ*S*b*ts.v*(lc["Cr0"] + lc["Crβ"]*β + lc["Crδa"]*tc.δa + lc["Crδr"]*tc.δr)
    A31 += 0.5*ρ*S*b*lc["Crβ"]*sqrt(ts.u^2 + ts.w^2)
    A31 = A31*af_term
    A32 = (Γ[7]*ts.q) + (Q*b*m*lc["Crp"])
    A33 = (-Γ[1]*ts.q) + (Q*b*m*lc["Crr"])
    A34 = 0.0
    A35 = 0.0

    A41 = 0.0
    A42 = 1.0
    A43 = cos(ts.ϕ)*tan(ts.θ)
    A44 = (ts.q*cos(ts.ϕ)*tan(ts.θ)) - (ts.r*sin(ts.ϕ)*tan(ts.θ))
    A45 = 0.0

    A51 = 0.0
    A52 = 0.0
    A53 = cos(ts.ϕ)*sec(ts.θ)
    A54 = (ts.p*cos(ts.ϕ)*sec(ts.θ)) - (ts.r*sin(ts.ϕ)*sec(ts.θ))
    A55 = 0.0

    A = [A11 A12 A13 A14 A15
        A21 A22 A23 A24 A25
        A31 A32 A33 A34 A35
        A41 A42 A43 A44 A45
        A51 A52 A53 A54 A55
        ]

    B11 = (1/b)*2*Va*Q*ap.CYda
    B11 = B11/af_term
    B12 = (1/b)*2*Va*Q*ap.CYdr
    B12 = B12/af_term

    B21 = 2*Va*m*Q*lc["Cpδa"]
    B22 = 2*Va*m*Q*lc["Cpδr"]

    B31 = 2*Va*m*Q*lc["Crδa"]
    B32 = 2*Va*m*Q*lc["Crδr"]

    B41 = 0.0
    B42 = 0.0

    B51 = 0.0
    B52 = 0.0

    B = [B11 B12
        B21 B22
        B31 B32
        B41 B42
        B51 B52
        ]

    return A,B
end

function GetLinearizedModel(::Type{LongitudinalAircraftState},ts::AircraftState,tc::AircraftControl,ap::AircraftParameters)

    ρ = stdatmo(-ts.z)
    S = ap.S
    c = ap.c
    m = ap.m
    km = ap.kmotor
    AirSpeedVector = [ts.u, ts.v, ts.w]  #because it has been given that wind speed is zero.
    wind_angles = AirRelativeVelocityVectorToWindAngles(AirSpeedVector)
    Va = wind_angles.Va
    α = wind_angles.α
    af_term = Va*cos(α)
    lc = GetLongitudinalEOMCoefficients(tc,α,ap)

    A11 = ts.u*ρ*S*lc["CXtrim"]/m
    A11 -= 0.5*ρ*S*ts.w*lc["CXα"]/m
    A11 += 0.25*ρ*S*c*lc["CXq"]*ts.u*ts.q/(m*Va)
    A11 += ρ*ap.Sprop*ap.Cprop*tc.δt*( km*ts.u*(1-2*tc.δt)/Va + 2*ts.u*(tc.δt-1) )/m
    A12 = -ts.q + ts.w*ρ*S*lc["CXtrim"]/m
    A12 += 0.25*ρ*S*c*lc["CXq"]*ts.w*ts.q/(m*Va)
    A12 += 0.5*ρ*S*ts.u*lc["CXα"]/m
    A12 += ρ*ap.Sprop*ap.Cprop*tc.δt*( km*ts.w*(1-2*tc.δt)/Va + 2*ts.w*(tc.δt-1) )/m
    A12 = A12*af_term
    A13 = -ts.w + (0.25*ρ*Va*S*lc["CXq"]*c)/m
    A14 = -ap.g*cos(ts.θ)
    A15 = 0.0

    A21 = ts.q + ts.u*ρ*S*lc["CZtrim"]/m
    A21 -= 0.5*ρ*S*lc["CZα"]*ts.w/m
    A21 += 0.25*ts.u*ρ*S*lc["CZq"]*c*ts.q/(m*Va)
```

```julia
    A21 = A21/af_term
    A22 = ts.w*ρ*S*lc["CZtrim"]/m
    A22 += 0.5*ρ*S*lc["CZα"]*ts.u/m
    A22 += 0.25*ts.w*ρ*S*lc["CZq"]*c*ts.q/(m*Va)
    A23 = ts.u + (0.25*ρ*Va*S*lc["CZq"]*c)/m
    A23 = A23/af_term
    A24 = -ap.g*sin(ts.θ)
    A24 = A24/af_term
    A25 = 0.0

    A31 = ts.u*ρ*S*c*(ap.Cm0 + ap.Cmalpha*α + ap.Cmde*tc.δe)
    A31 -=  0.5*ρ*S*c*ap.Cmalpha*ts.w/ap.Iy
    A31 += 0.25*ρ*S*c*c*ap.Cmq*ts.q*ts.u/(ap.Iy*Va)
    A32 = ts.w*ρ*S*c*(ap.Cm0 + ap.Cmalpha*α + ap.Cmde*tc.δe)
    A32 += 0.5*ρ*S*c*ap.Cmalpha*ts.u/ap.Iy
    A32 += 0.25*ρ*S*c*c*ap.Cmq*ts.q*ts.w/(ap.Iy*Va)
    A32 = A32*af_term
    A33 = (0.25*ρ*Va*S*c*c*ap.Cmq)/ap.Iy
    A34 = 0.0
    A35 = 0.0

    A41 = 0.0
    A42 = 0.0
    A43 = 1.0
    A44 = 0.0
    A45 = 0.0

    A51 = sin(ts.θ)
    A52 = -cos(ts.θ)
    A52 = A52*af_term
    A53 = 0.0
    A54 = (ts.u*cos(ts.θ)) + (ts.w*sin(ts.θ))
    A55 = 0.0

    A = [A11 A12 A13 A14 A15
        A21 A22 A23 A24 A25
        A31 A32 A33 A34 A35
        A41 A42 A43 A44 A45
        A51 A52 A53 A54 A55
        ]

    B11 = 0.5*ρ*Va*Va*S*lc["CXδe"]/m
    B12 = ρ*ap.Sprop*ap.Cprop*( Va*(km-Va) + 2*tc.δt*(km-Va)*(km-Va) )/m

    B21 = 0.5*ρ*Va*Va*S*lc["CZδe"]/m
    B21 = B21/af_term
    B22 = 0.0

    B31 = 0.5*ρ*Va*Va*S*c*ap.Cmde/ap.Iy
    B32 = 0.0

    B41 = 0.0
    B42 = 0.0

    B51 = 0.0
    B52 = 0.0

    B = [B11 B12
        B21 B22
        B31 B32
        B41 B42
        B51 B52
        ]

    return A,B
end


trim_definition = TrimDefinitionSL(18.0,0.0,1800.0)
state, control, results = GetTrimConditions(trim_definition, aircraft_parameters)
wind_inertial = [0.0,0.0,0.0]
Alat,Blat = GetLinearizedModel(LateralAircraftState,state,control,aircraft_parameters)
# Alat,Blat = @enter GetLinearizedModel(LateralAircraftState,state,control,aircraft_parameters)
Along,Blong = GetLinearizedModel(LongitudinalAircraftState,state,control,aircraft_parameters)
# Alat,Blat = @enter GetLinearizedModel(LateralAircraftState,state,control,aircraft_parameters)

println("A_lat")
display(Alat)
println("B_lat")
display(Blat)
println("A_long")
display(Along)
println("B_long")
display(Blong)


#=
Problem 1 - Part 2)
=#

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
```

```julia
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("")
println("For Dutch Roll Mode : ")
println("Damping Ratio : ", dutch_roll_damping_ratio)
println("Natural Frequency : ", dutch_roll_natural_frequency)


#=
Problem 1 - Part 2)
=#

println("")
println("For Roll Mode : ")
println("Damping Ratio : ", roll_damping_ratio)
println("Natural Frequency : ", roll_natural_frequency)
println("Time Constant : ", 1/roll_natural_frequency)
println("The roll mode is stable.")
println("")
println("For Spiral Mode : ")
println("Damping Ratio : ", spiral_damping_ratio)
println("Natural Frequency : ", spiral_natural_frequency)
println("Time Constant : ", 1/spiral_natural_frequency)
println("The spiral mode is unstable.")


#Problem 2

function PulseControl(t::Float64,true_control::Union{AircraftControl,Array{Float64,1}},pulse_control::Array{Float64,1},pulse_time::Array{Float64,1})

    if(t==0.0)
        return true_control
    end
    if( length(pulse_time) == 1 )
        if(t<=pulse_time[1])
            control = true_control + pulse_control
        else
            control = true_control
        end
    elseif( length(pulse_time) == 2)
        if(t<=pulse_time[1])
            control = true_control + pulse_control
        elseif(t>pulse_time[1] && t<=pulse_time[2])
            control = true_control - pulse_control
        else
            control = true_control
        end
    end

    return control
end

function AircraftEOMPulsed(t::Float64,aircraft_state::AircraftState,controls::AircraftControl,control_function::Function,
            pulse_control::Array{Float64,1},pulse_time::Array{Float64,1},wind_inertial::Array{Float64,1},aircraft_parameters::AircraftParameters)

    aircraft_surfaces = control_function(t,controls,pulse_control,pulse_time)
    return AircraftEOM(t,aircraft_state,aircraft_surfaces,wind_inertial,aircraft_parameters)
end

function aircraft_dynamics_pulsed!(du,u,p,t)
    aircraft_state = AircraftState(u...)
    control_function = p[1]
    control_inputs = AircraftControl(p[2]...)
    pulse_control = p[3]
    pulse_time = p[4]
    wind_inertial = p[5]
    aircraft_parameters = p[6]
    x_dot = AircraftEOMPulsed(t,aircraft_state,control_inputs,control_function,pulse_control,pulse_time,wind_inertial,aircraft_parameters)
    for i in 1:length(u)
        du[i] = x_dot[i]
    end
end

filename = "ttwistor.mat"
aircraft_parameters = AircraftParameters(filename)
```

```
location = "/HW5/plots"
save_plots = true
case_num = 3


trim_definition = TrimDefinitionSL(18.0,0.0,1800.0)
state, control, results = GetTrimConditions(trim_definition, aircraft_parameters)
initial_state = collect(values(state))
control_input = collect(values(control))
wind_inertial = [0.0,0.0,0.0]

if(case_num == 1)
    pulse_control = [pi/10,0.0,0.0,0.0]
    pulse_time = [1.0]
end

if(case_num == 2)
    pulse_control = [0.0,pi/10,0.0,0.0]
    pulse_time = [1.0,2.0]
end

if(case_num == 3)
    pulse_control = [0.0,0.0,pi/10,0.0]
    pulse_time = [1.0,2.0]
end


time_interval = [0.0,200.0]
extra_parameters = [PulseControl,control_input,pulse_control,pulse_time,wind_inertial,aircraft_parameters]
trajectory_states = simulate(aircraft_dynamics_pulsed!,initial_state,time_interval,extra_parameters,1.0)
time_values = [i for i in 0:length(trajectory_states)-1]
control_array = [AircraftControl(PulseControl(1.0*(i-1),control_input,pulse_control,pulse_time)...) for i in 1:length(trajectory_states)]
PlotSimulation(time_values, trajectory_states, control_array , location, save_plots)
```