23rd March, 2023

EXAM 1

Problem 1

1. We know that

$$\dot{\theta} = q \cos\phi - r \sin\phi$$

Given: $q = -0.2^{\circ}/s$

$r = 0^{\circ}/s$

$\phi = -12^{\circ}$

$$\therefore \dot{\theta} = -0.2 * \cos\left(-12 \times \frac{\pi}{180}\right)$$

$$= -0.196^{\circ}/s = -0.0034 \text{ radians/s}$$

∴ time rate of change of pitch angle is $-0.196^{\circ}/s$.

2. We know $v_B^E = \begin{bmatrix} 15 \\ -3 \\ 1 \end{bmatrix}$ and

$$w_E^E = \begin{bmatrix} 0 \\ 1 \\ -2 \end{bmatrix}$$

We know that $w_B^E = R_E^B \cdot w_E^E$

$\therefore \quad v_B^B = v_B^E - w_B^E$

We can use v_B^B to calculate α.

$$\alpha = \text{atan}\left(\frac{v_{B\,z}^B}{v_{B\,x}^B}\right)$$

$\therefore \quad \alpha = 0.209$ radians $= 12.024$ degrees

3. If the thrust is applied in body-x direction, then the only forces in body-z direction are due to the aerodynamics forces or gravitational forces.

We know that

$$\dot{w}_B^E = q \cdot u_B^E - p \cdot v_B^E + \frac{1}{m} f_z$$

where $f_z = Z_{aero} + Z_{gravity}$

Given: $\dot{w}_B^E = 0.05 \text{ m/s}^2$; $u_B^E = 15 \text{ m/s}$

$v_B^E = -3 \text{ m/s}$; $p = 0.08^\circ/\text{s}$; $q = -0.2^\circ/\text{s}$

$m = 10 \text{ kg}$

$\therefore f_z = 1.0236 \text{ N}$

Also, $Z_{gravity} = m * g * \cos(\theta) * \cos(\phi)$

$\therefore Z_{gravity} = 94.7749 \quad N$

$\therefore Z_{aero} = f_z - Z_{gravity}$

$= -93.7513 \quad N$

Problem 2

(1) TRUE !

TECS will maintain a constant total of KE + PE, and it uses throttle to regulate it.

From the discussion in class, it can be inferred that to speed up the aircraft, the elevator changes to decrease drag, and similarly it can increase drag to slow its speed, all while maintaining the same altitude.
As a result, it seems the statement is True.

(However, I am not entirely convinced since there is coupling between states and so change in V_a should also have an effect on h. Need to discuss more with Dr. Frew).

2. FALSE !

The linear design model can be derived for any aircraft and doesn't depend on whether it has stable modes or not.

For ex: We have shown that the Twistor airplane that we have used so far has an unstable spiral mode. However, we have still been able to develop linear models for that aircraft.

Unstable modes can cause big changes in an aircraft's state when it encounters small perturbations from the nominal trajectory, and so controling such an aircraft can be difficult.

Problem 3

From slides and book, we know that the commanded pitch angle can be obtained from commanded height using this equation:

$$\theta^c(t) = k_{p_h}\left(h^c(t) - h(t)\right) + k_{i_h}\int_{-\infty}^{t}\left(h^c(\tau) - h(\tau)\right)\cdot d\tau$$

We also know that a desired/commanded pitch angle θ^c can be obtained by changing the elevator. The corresponding equation is

$$\delta_e(t) = k_{p_\theta}\left(\theta^c(t) - \theta(t)\right) - k_{d_\theta}\, q(t)$$

$$\therefore \theta^{c}(t) = \frac{1}{k_{p_\theta}} \left(\delta_e(t) + k_{d_\theta} q(t) \right) + \theta(t)$$

$\therefore$ We have

$$\frac{1}{k_{p_\theta}} \left(\delta_e(t) + k_{d_\theta} q(t) \right) + \theta(t) = k_{p_h} \left(h^{c}(t) - h(t) \right) + k_{i_h} \int_{-\infty}^{t} \left(h^{c}(\tau) - h(\tau) \right) \cdot d\tau$$

$$\therefore \delta_e(t) = k_{p_h} k_{p_\theta} \left(h^{c}(t) - h(t) \right) + k_{i_h} k_{p_\theta} \int_{-\infty}^{t} \left(h^{c}(\tau) - h(t) \right) \cdot d\tau - k_{p_\theta} \theta(t) - k_{d_\theta} q(t)$$

This is the overall law for the elevator angle for this approach.

Problem 4

Part1:

For Short Period Mode :
Damping Ratio: 0.3517726583144173
Natural Frequency: 3.7753929083414826

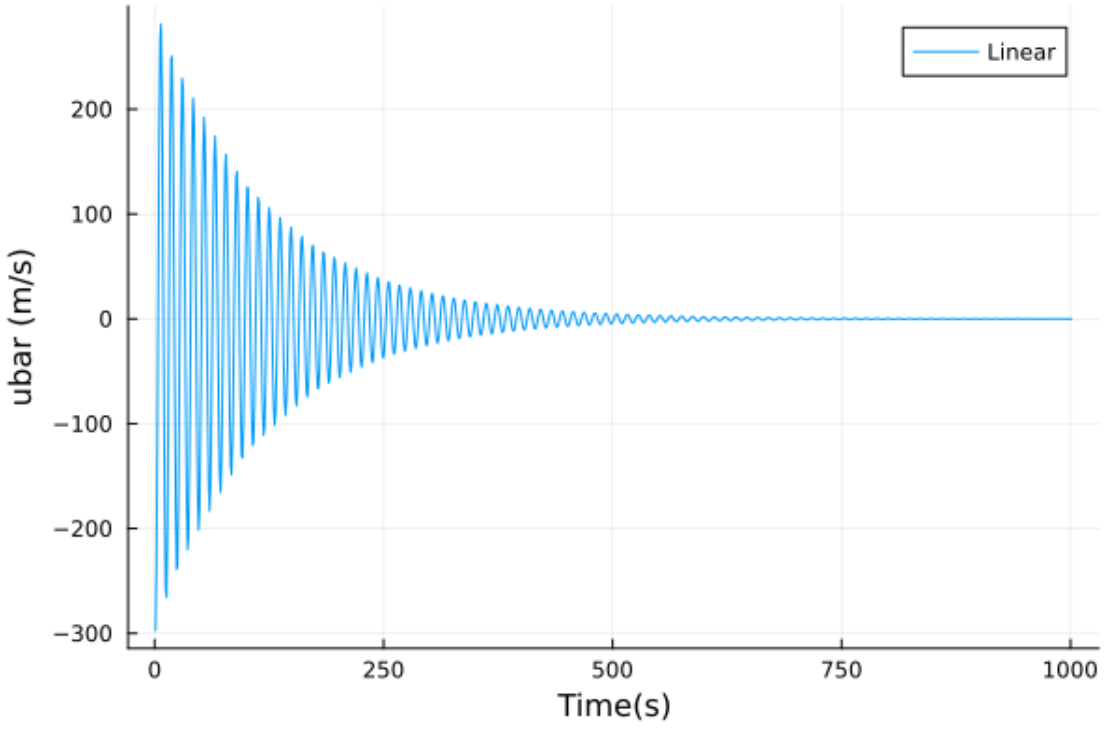
For Phugoid Mode :
Damping Ratio: 0.015662714197962942
Natural Frequency: 0.5302783415889679

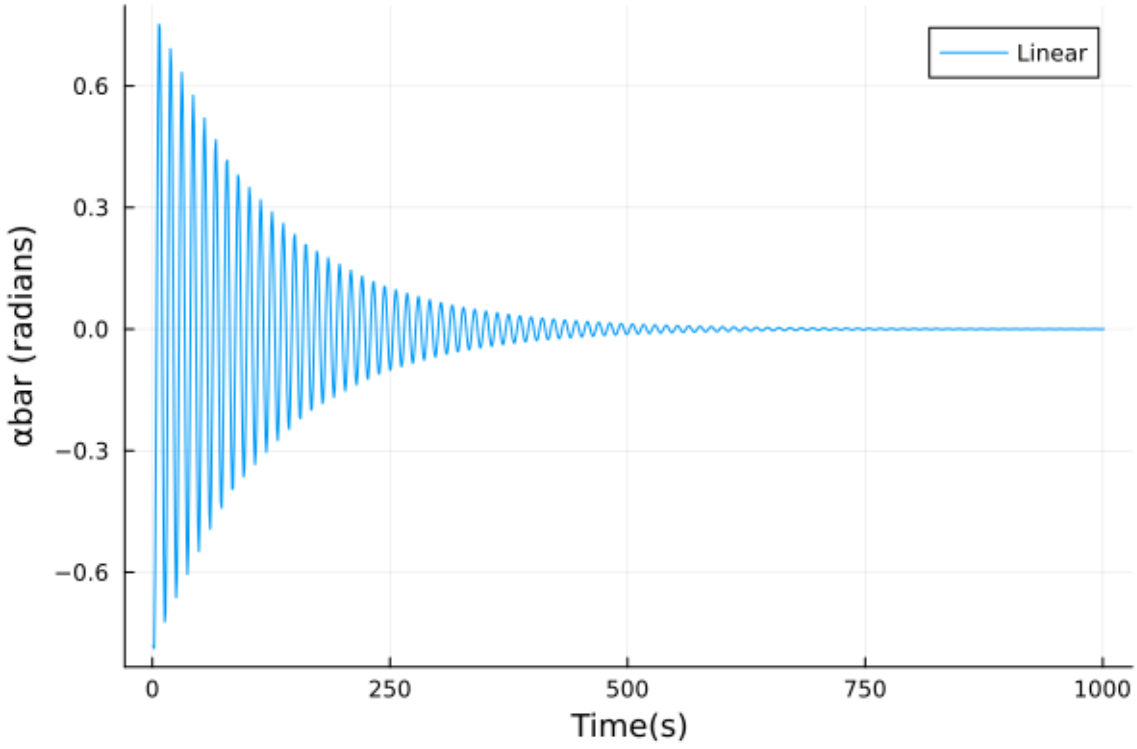
Part2:

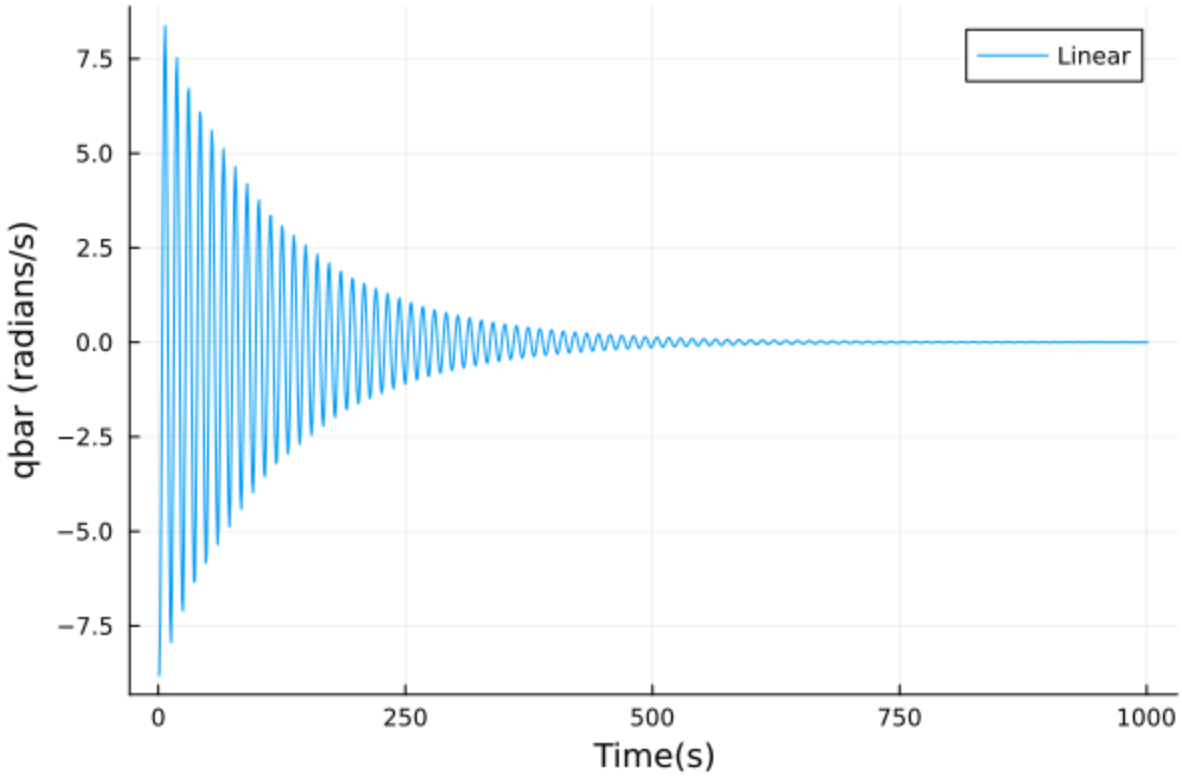
For Short Period Mode :
Eigenvalue : -1.3280799995486825 - 3.5340904243035336im
Eigenvector :
[
  0.05200873086912191 - 0.22616200256732813im
 -0.13427077743526278 + 0.11942307681502852im
   0.3053623657388001 + 0.5694776206843573im
 -0.16965064823186435 + 0.022651580265484635im
   0.6833812670502114 - 0.0im
]

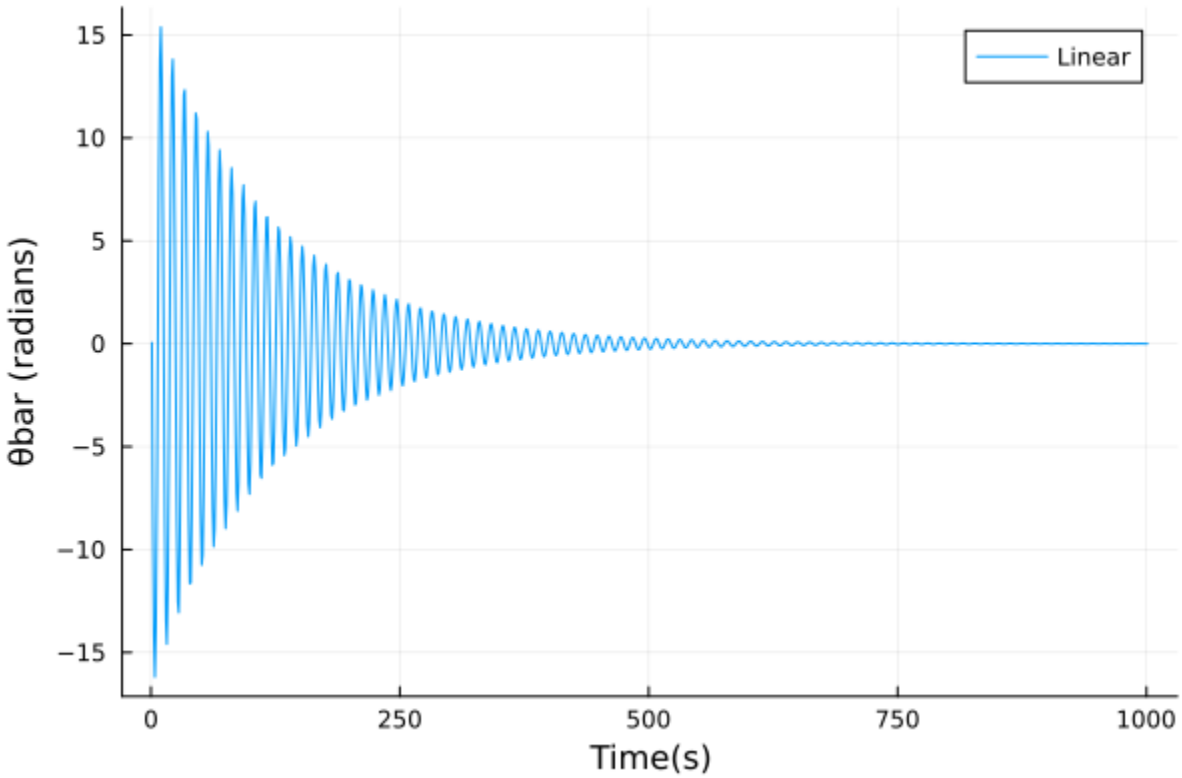
For Phugoid Mode :
Eigenvalue : -0.008305598109677771 - 0.530213293494596im
Eigenvector :
[
    -0.3612050346417226 + 0.04138233464217913im
 -0.0009502074203060539 - 0.0002885631507027 6535im
  -0.010704528067482972 + 0.00013391596586432154im
   6.366925143638189e-5 - 0.02018810050144037im
     0.9312868896617936 - 0.0im
]

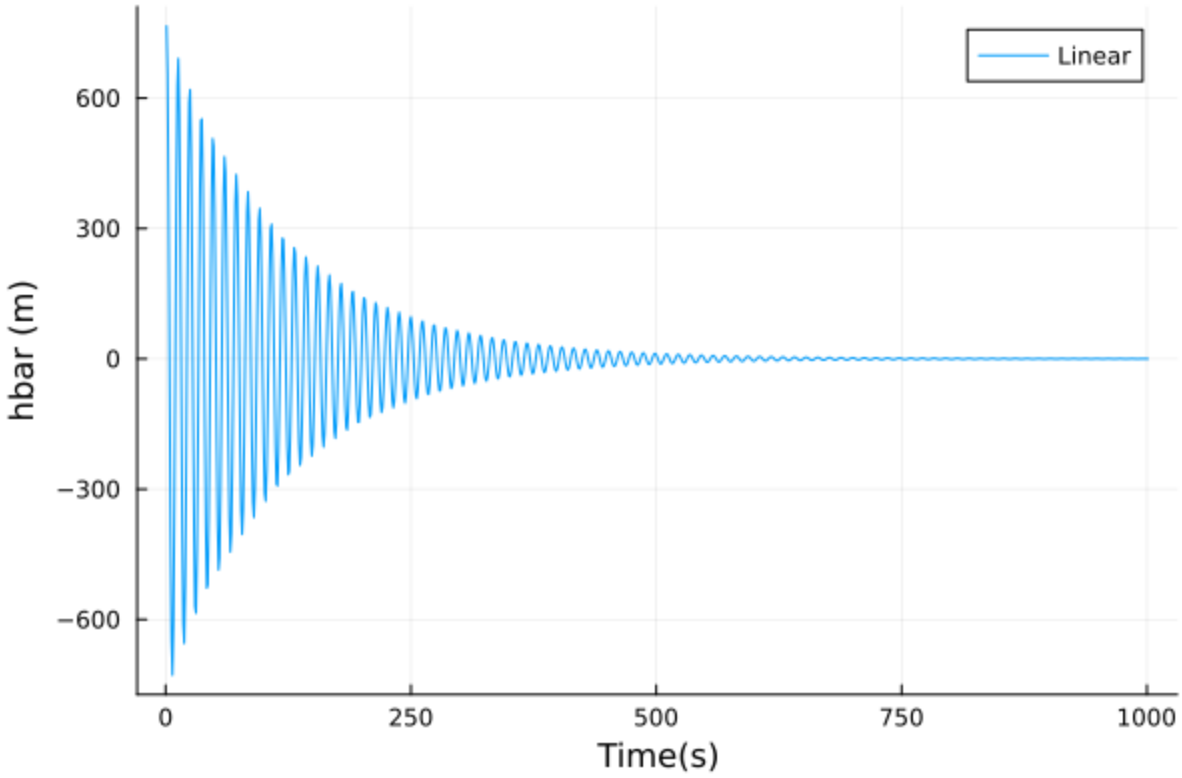
Part3:


Linear
200
100
0
−100
−200
−300
ubar (m/s)
0
250
500
750
1000
Time(s)


Linear
0.6
0.3
0.0
−0.3
−0.6
αbar (radians)
0
250
500
750
1000
Time(s)


Linear
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
qbar (radians/s)
0
250
500
750
1000
Time(s)

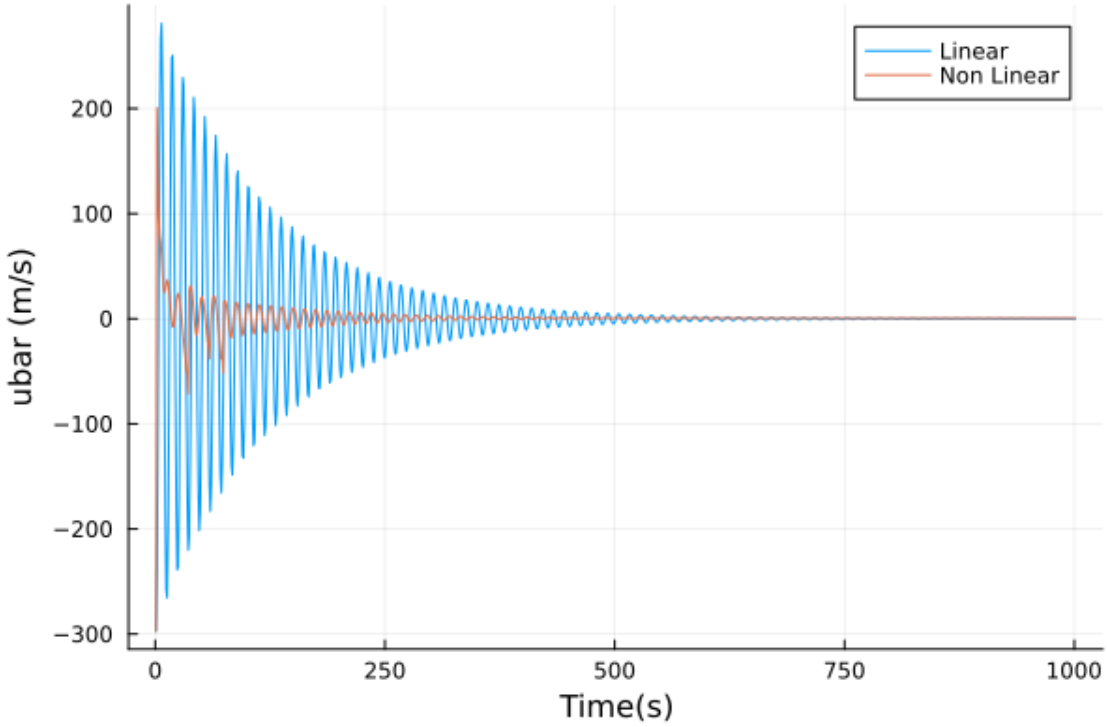

Linear
15
10
5
0
−5
−10
−15
θbar (radians)
0
250
500
750
1000
Time(s)

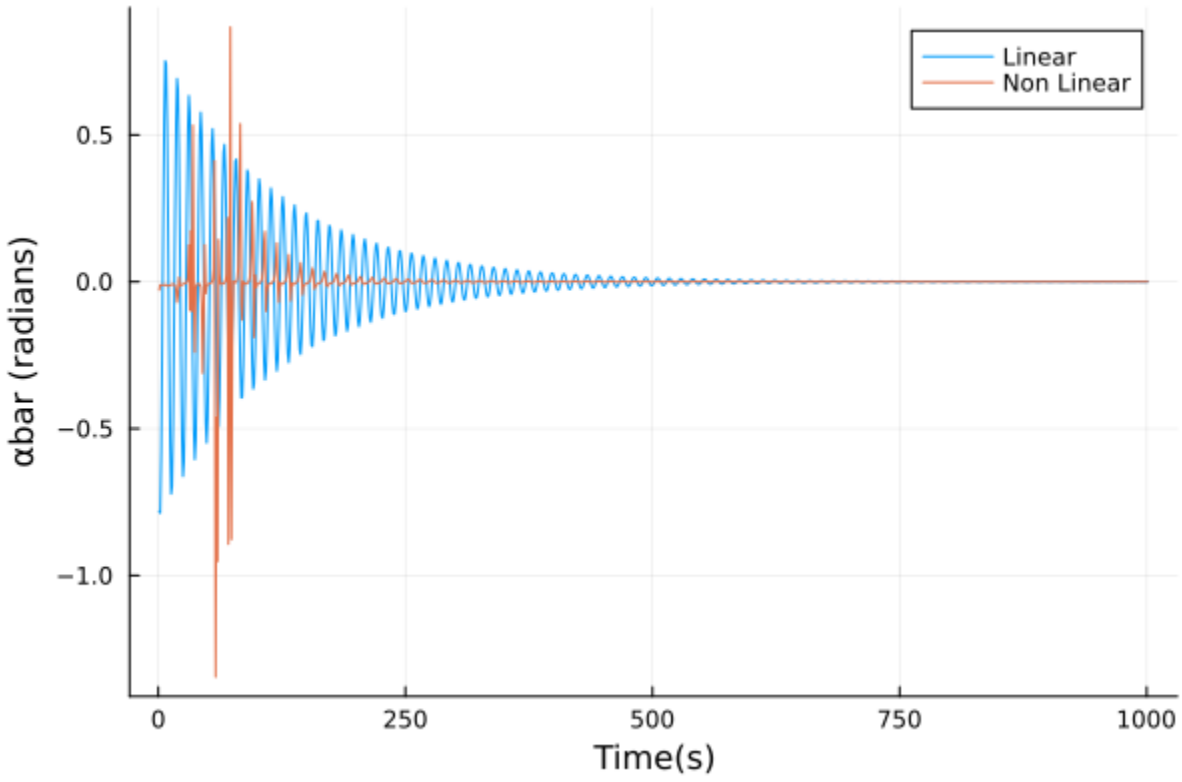


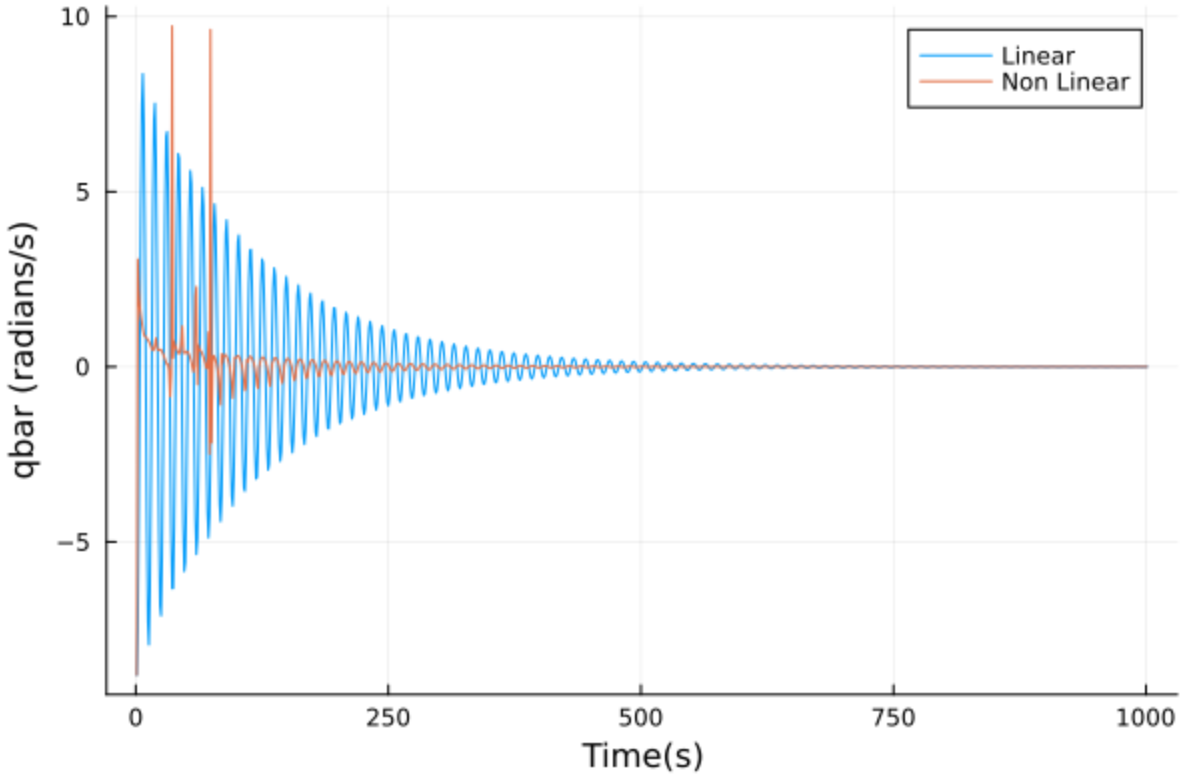
These results are for the linearized model. Since phugoid mode is a STABLE mode, the observed behavior of all the deviations converging to 0 was expected.

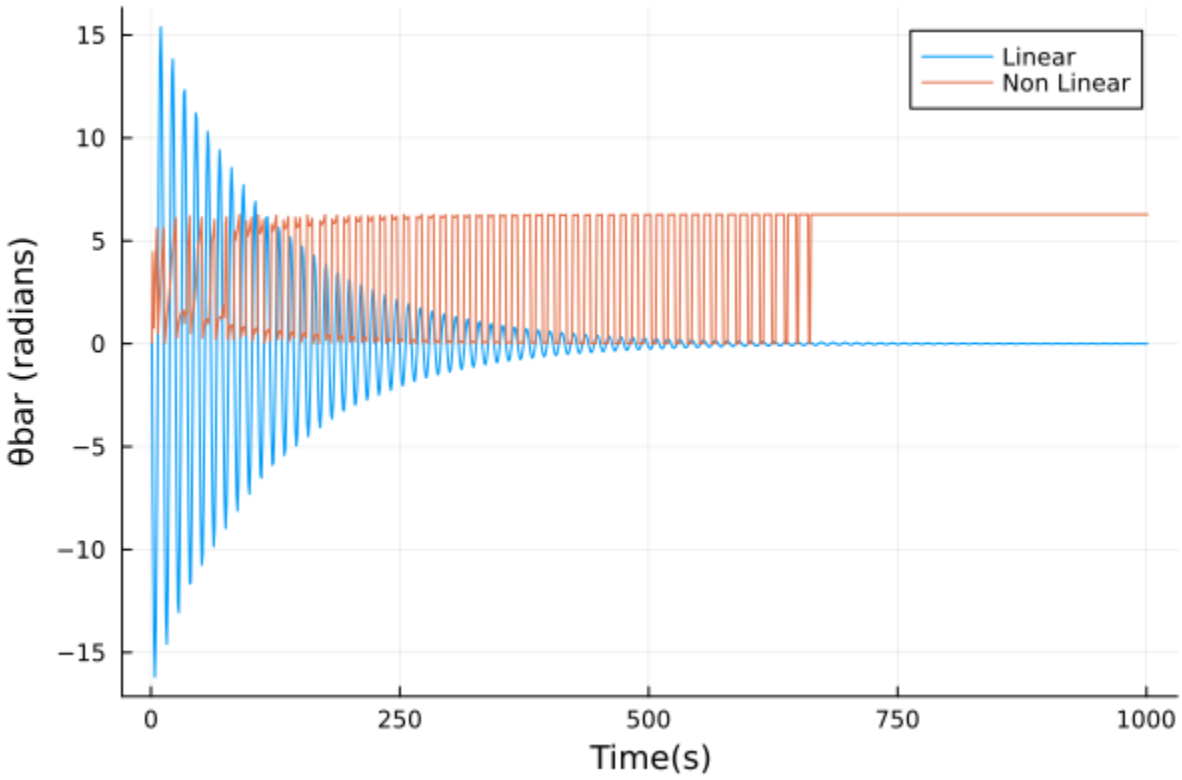
Part4:

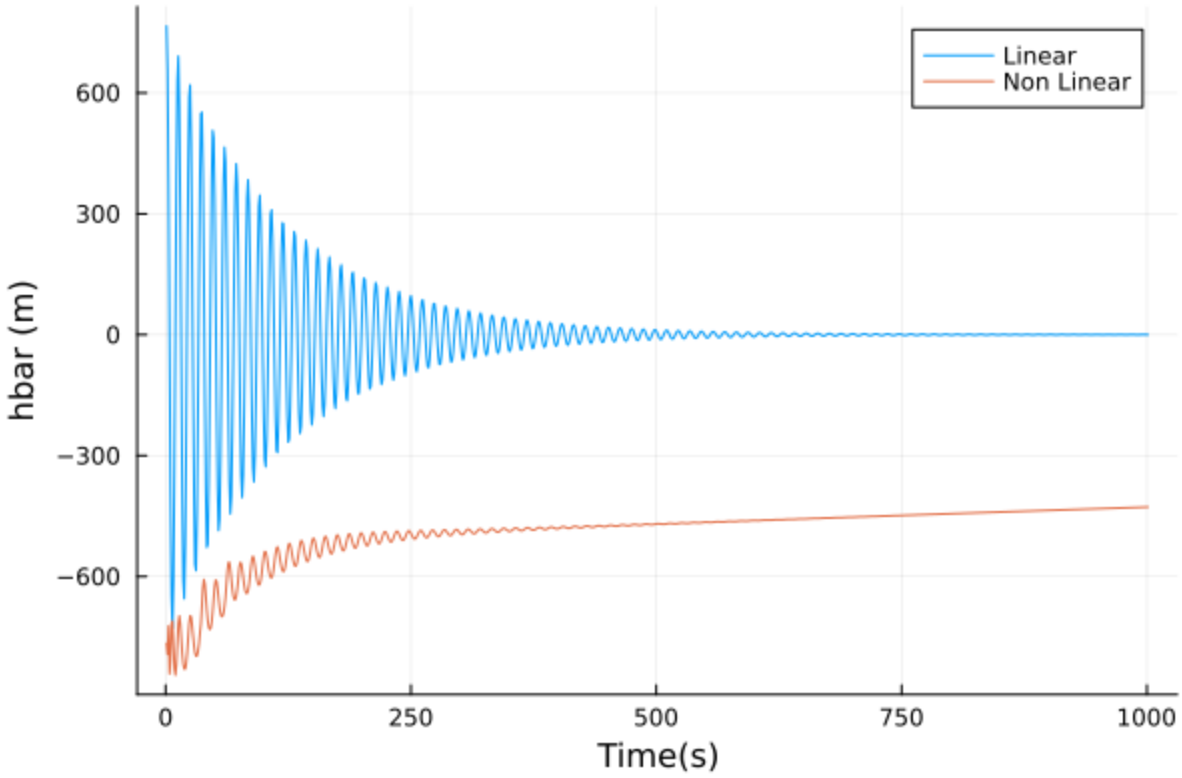
The following plots are for the time interval 0 to 1000 seconds.





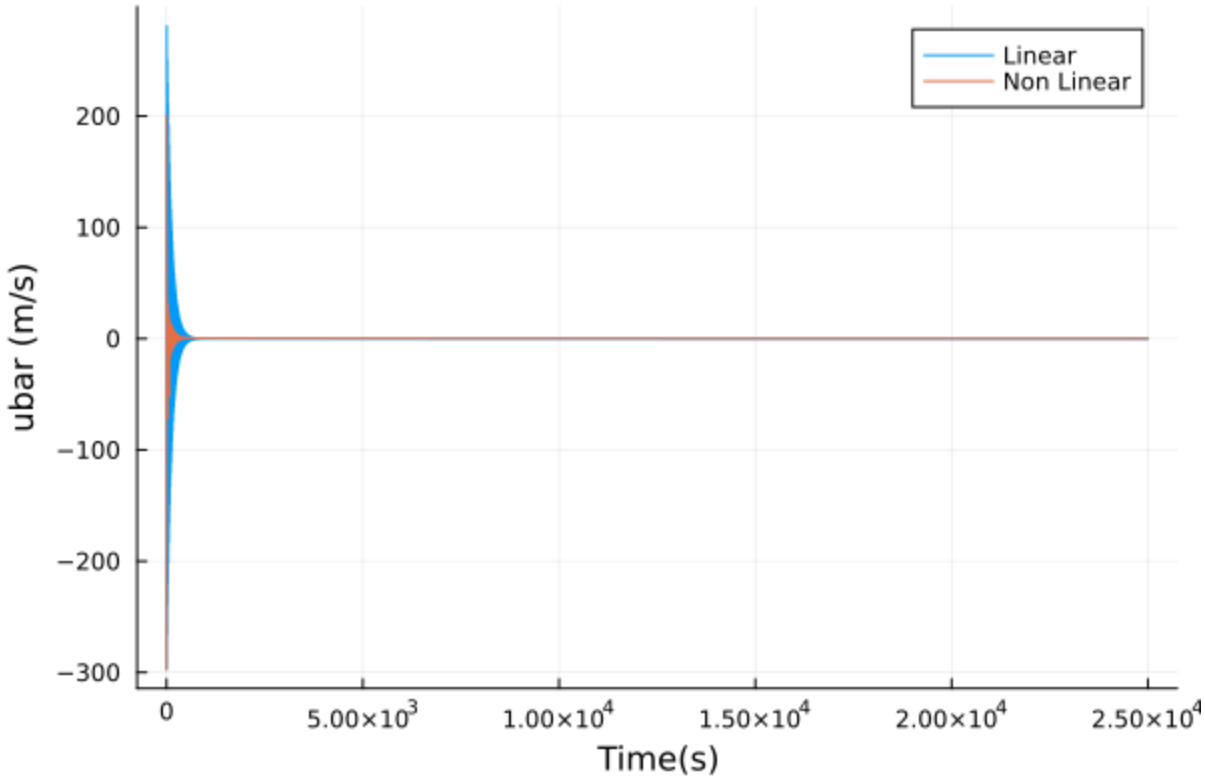

Linear
Non Linear
qbar (radians/s)
10
5
0
−5
0
250
500
750
1000
Time(s)

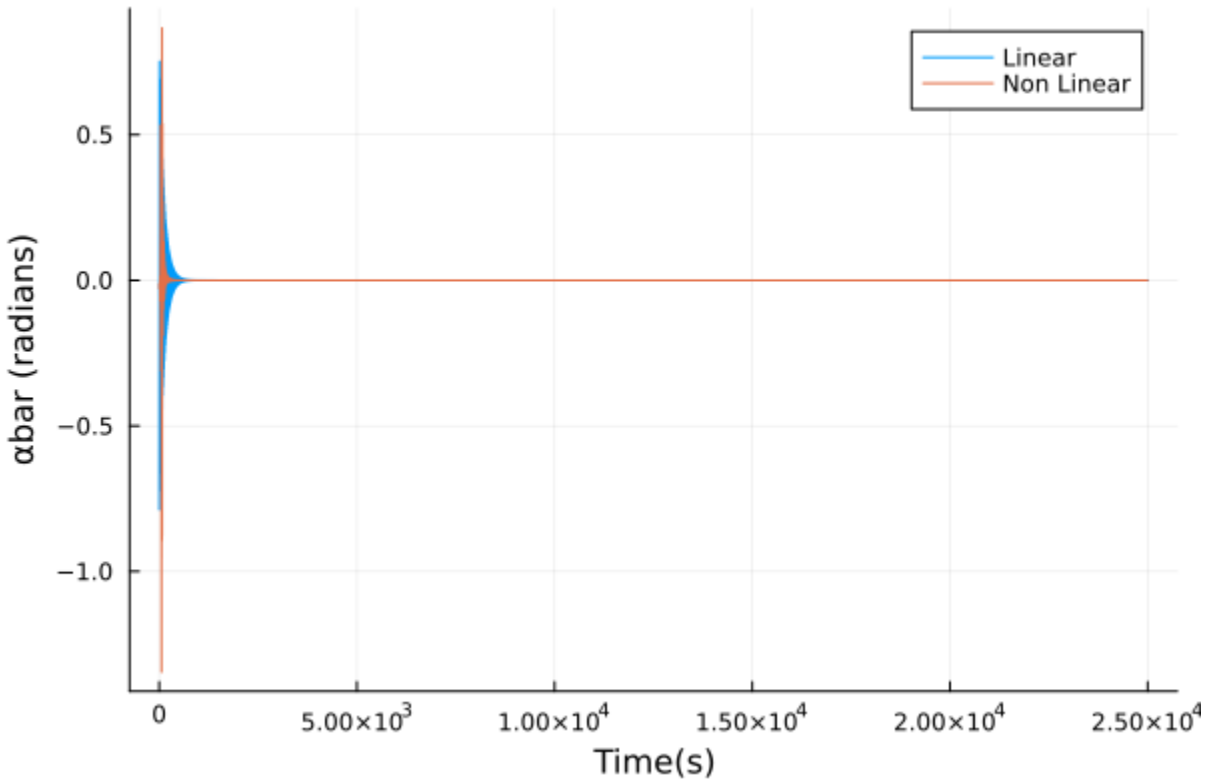

Linear
Non Linear
θbar (radians)
15
10
5
0
−5
−10
−15
0
250
500
750
1000
Time(s)

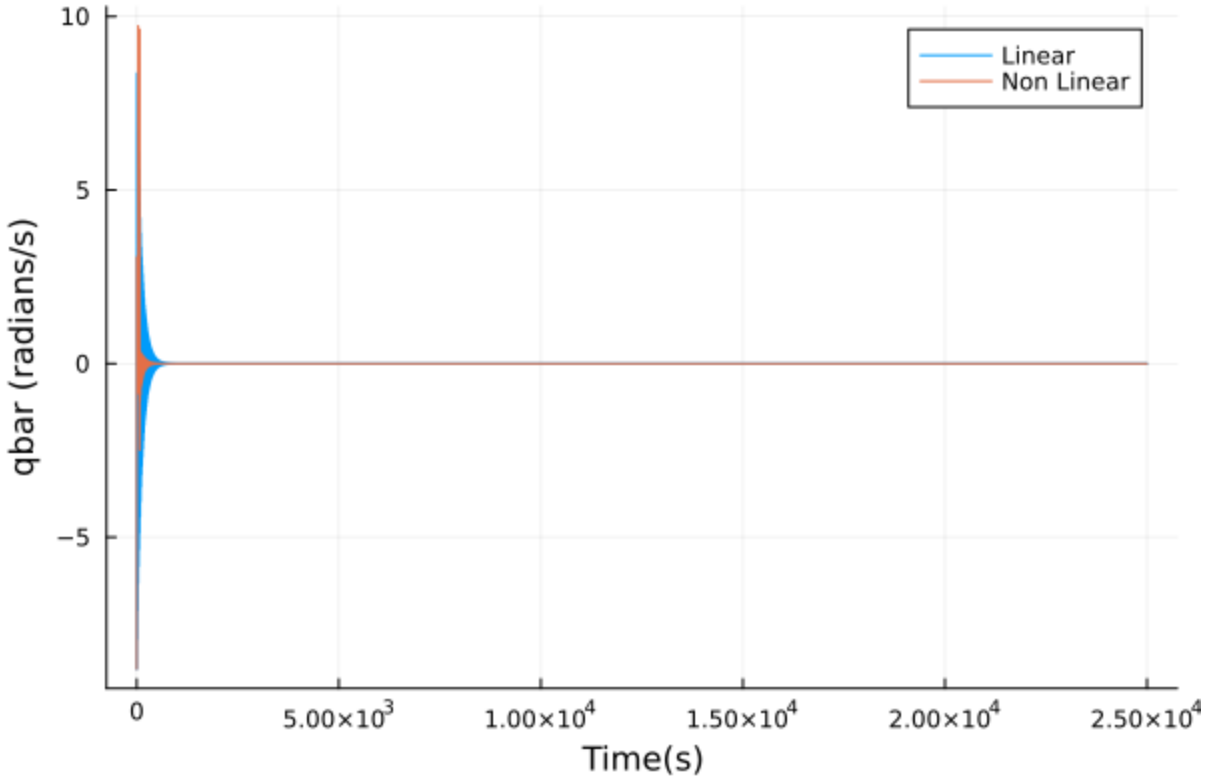


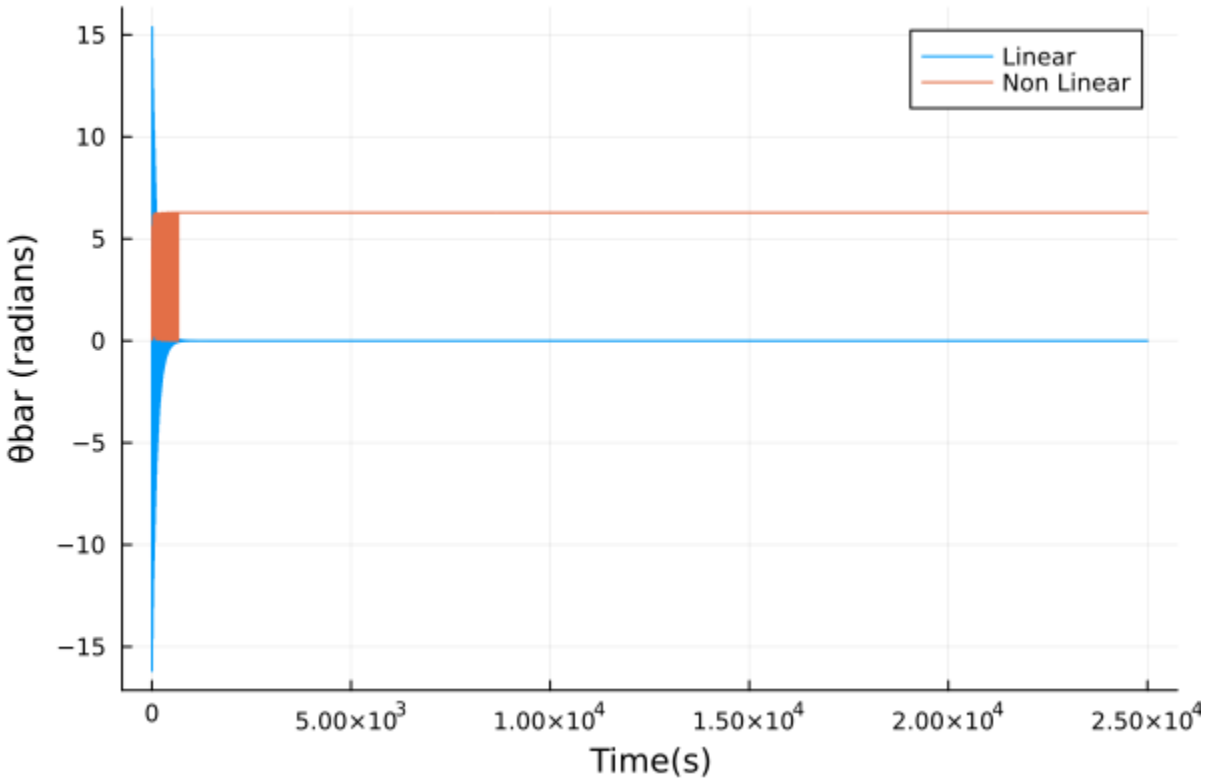
The plots show that the perturbations in u, alpha, and q go to zero for both the linearized and nonlinear models. It is a bit misleading from the plot, but the same happens for perturbation in theta. The converged perturbation value for pitch shown in the plot is close to 2pi (which, when wrapped, is close to 0). The scaling required to get the initial perturbation of 3 degrees in pitch resulted in a very high perturbation value for height. As a result, it takes a lot of time for the height perturbation to go to 0, but it eventually does go to 0, as seen in the plots below. So, yes, the linear model is a good enough approximation of the nonlinear model.

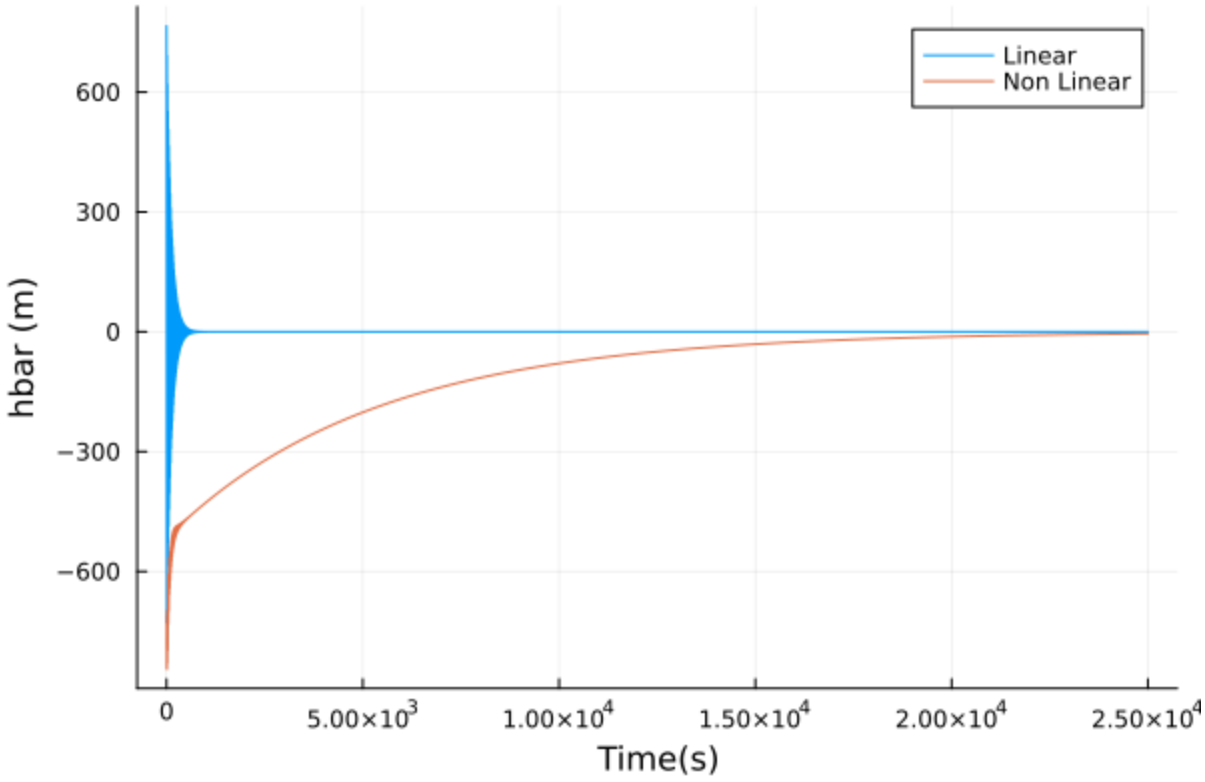
The following plots are for the time interval 0 to 25000 seconds.


Linear
Non Linear
200
100
0
−100
−200
−300
ubar (m/s)
0
5.00×10^3
1.00×10^4
1.50×10^4
2.00×10^4
2.50×10^4
Time(s)

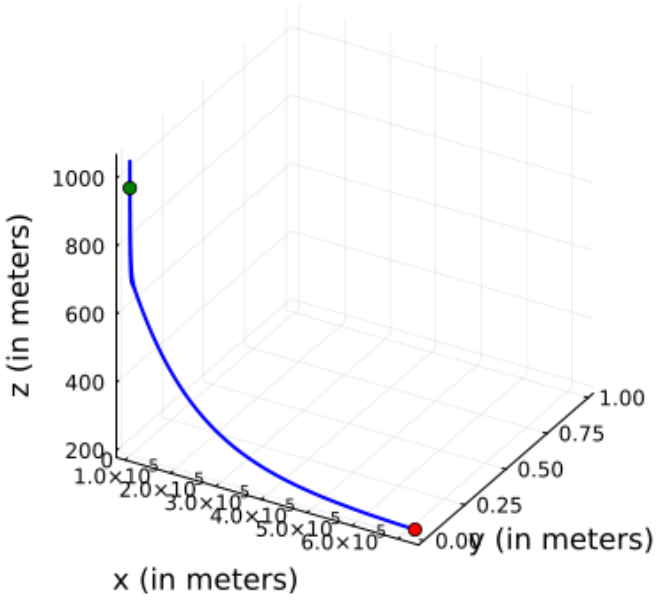

Linear
Non Linear
0.5
0.0
−0.5
−1.0
αbar (radians)
0
5.00×10^3
1.00×10^4
1.50×10^4
2.00×10^4
2.50×10^4
Time(s)


Linear
Non Linear
qbar (radians/s)
10
5
0
−5
0
5.00×10^3
1.00×10^4
1.50×10^4
2.00×10^4
2.50×10^4
Time(s)


Linear
Non Linear
θbar (radians)
15
10
5
0
−5
−10
−15
0
5.00×10^3
1.00×10^4
1.50×10^4
2.00×10^4
2.50×10^4
Time(s)

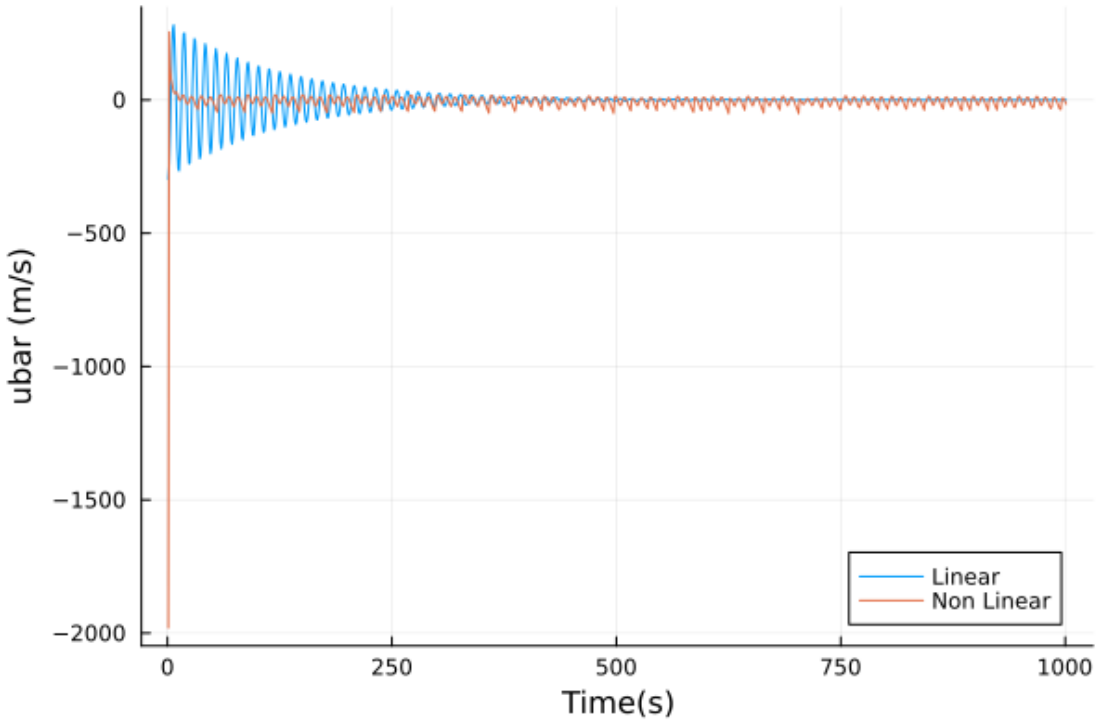


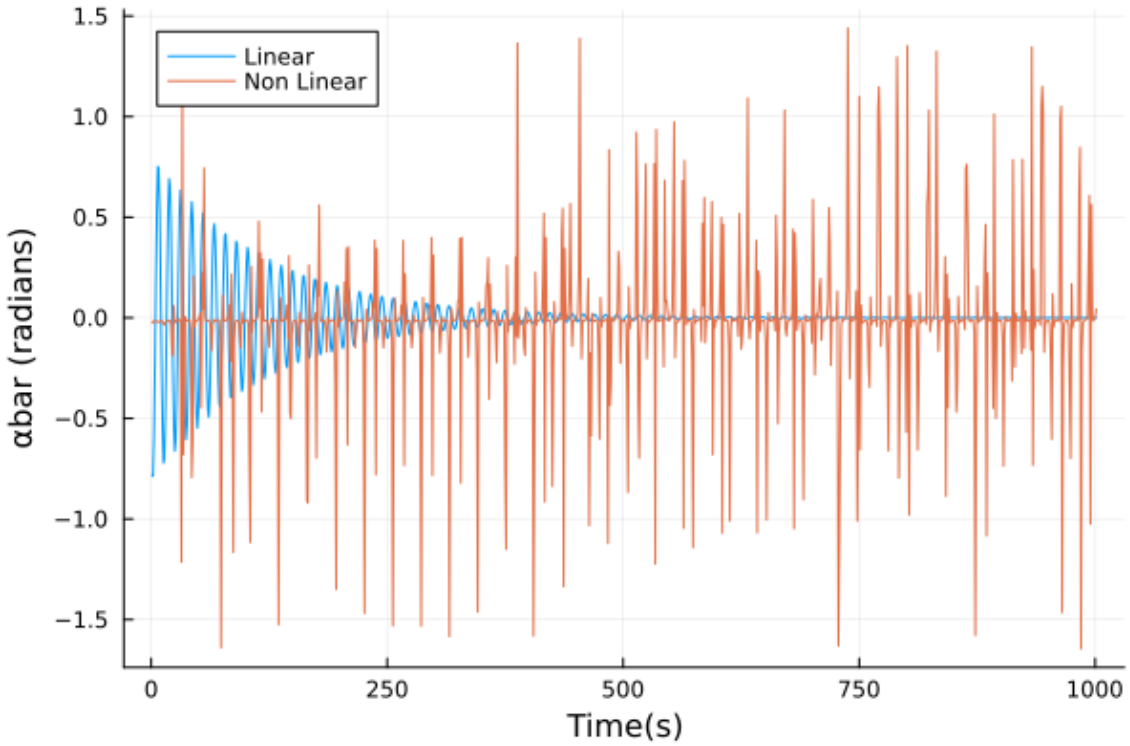
As visible, the perturbation in h eventually goes to 0.

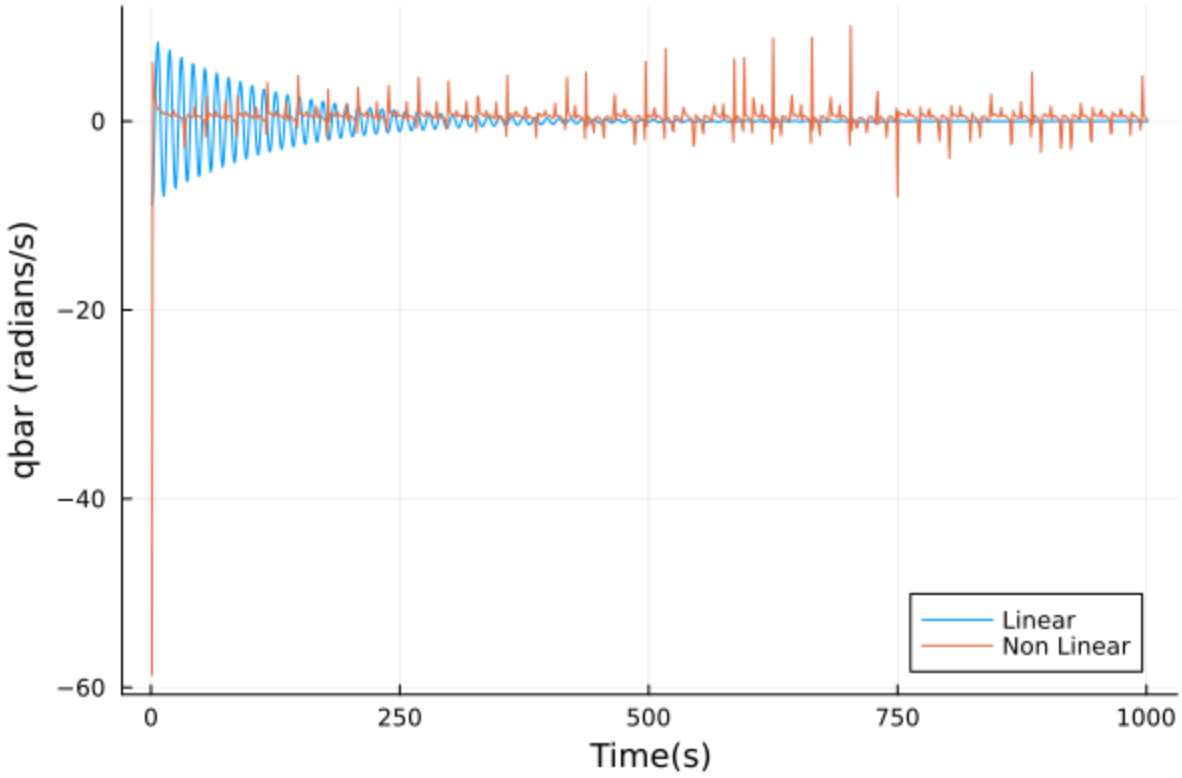


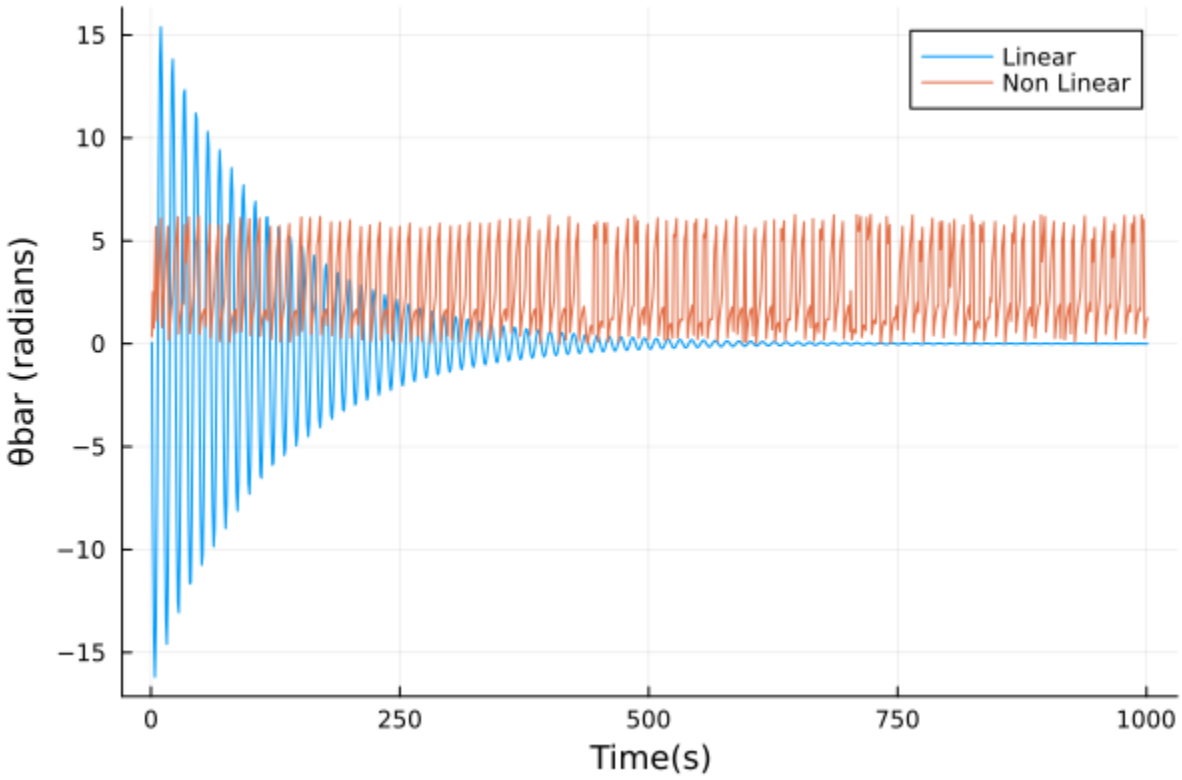
Part5:

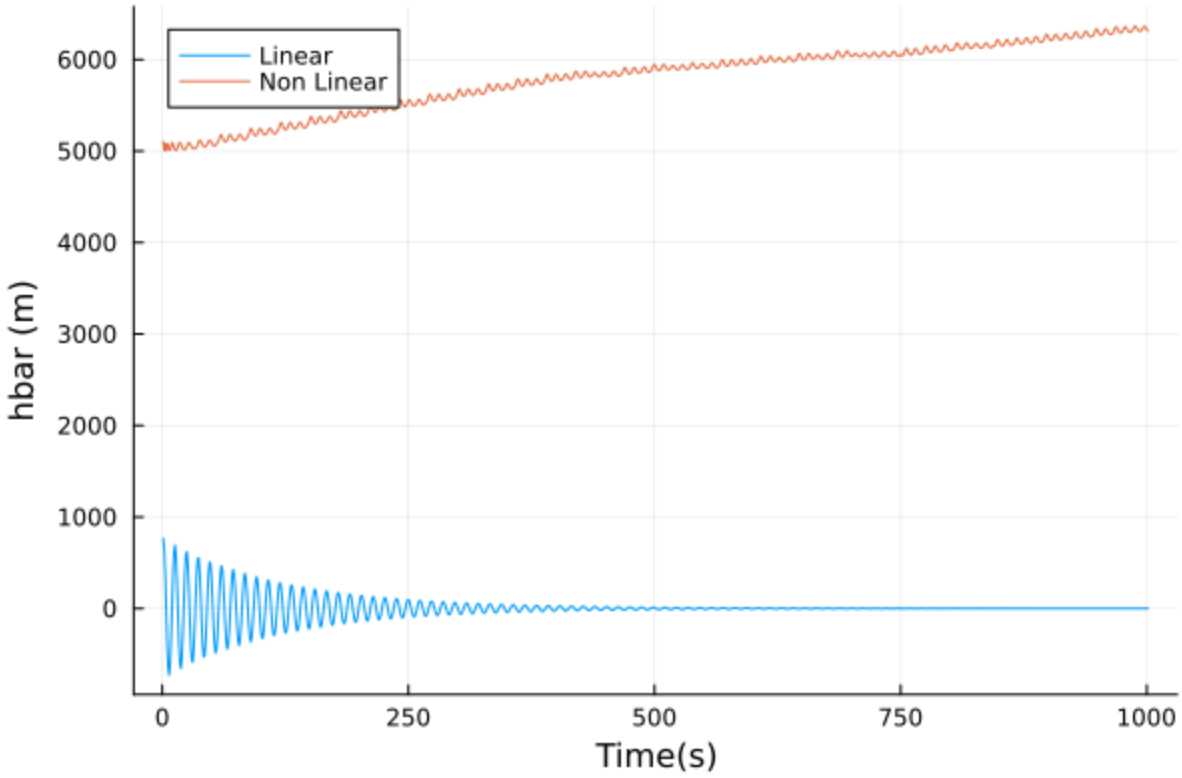
The following plots are for the time interval 0 to 1000 seconds.





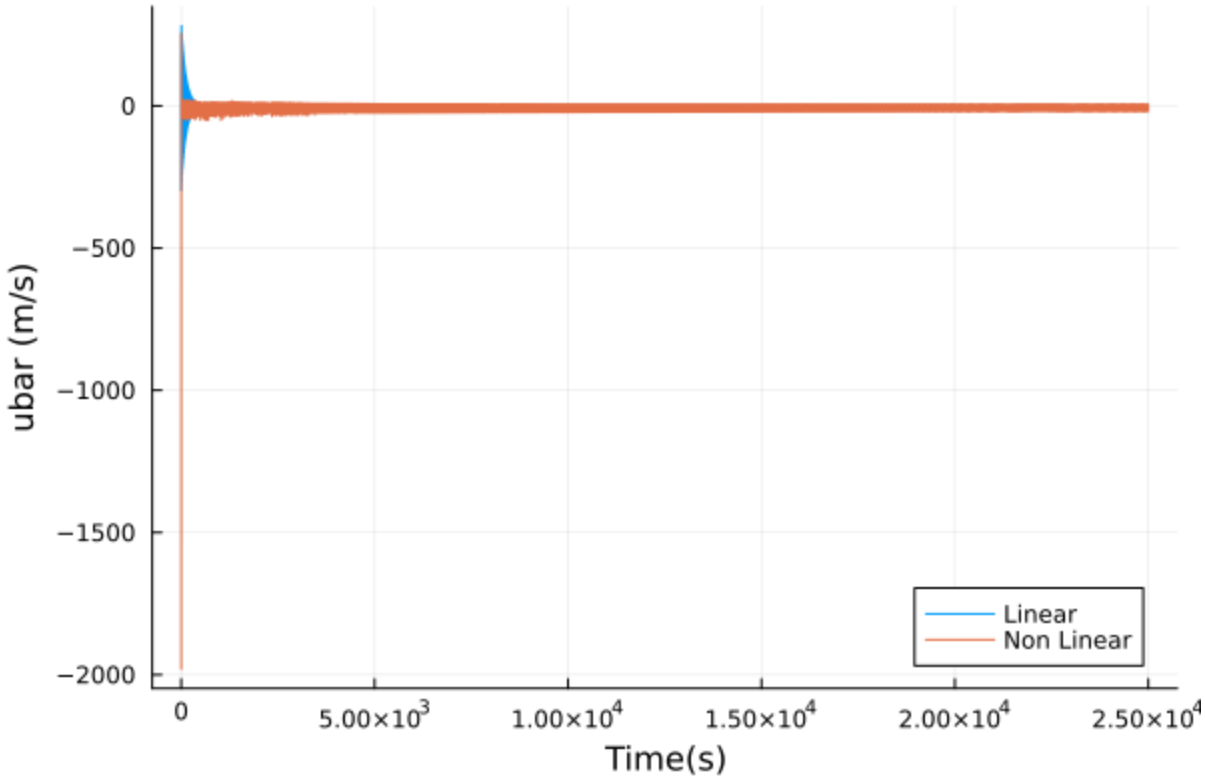

qbar (radians/s)
0
−20
−40
−60
0
250
500
750
1000
Time(s)
Linear
Non Linear

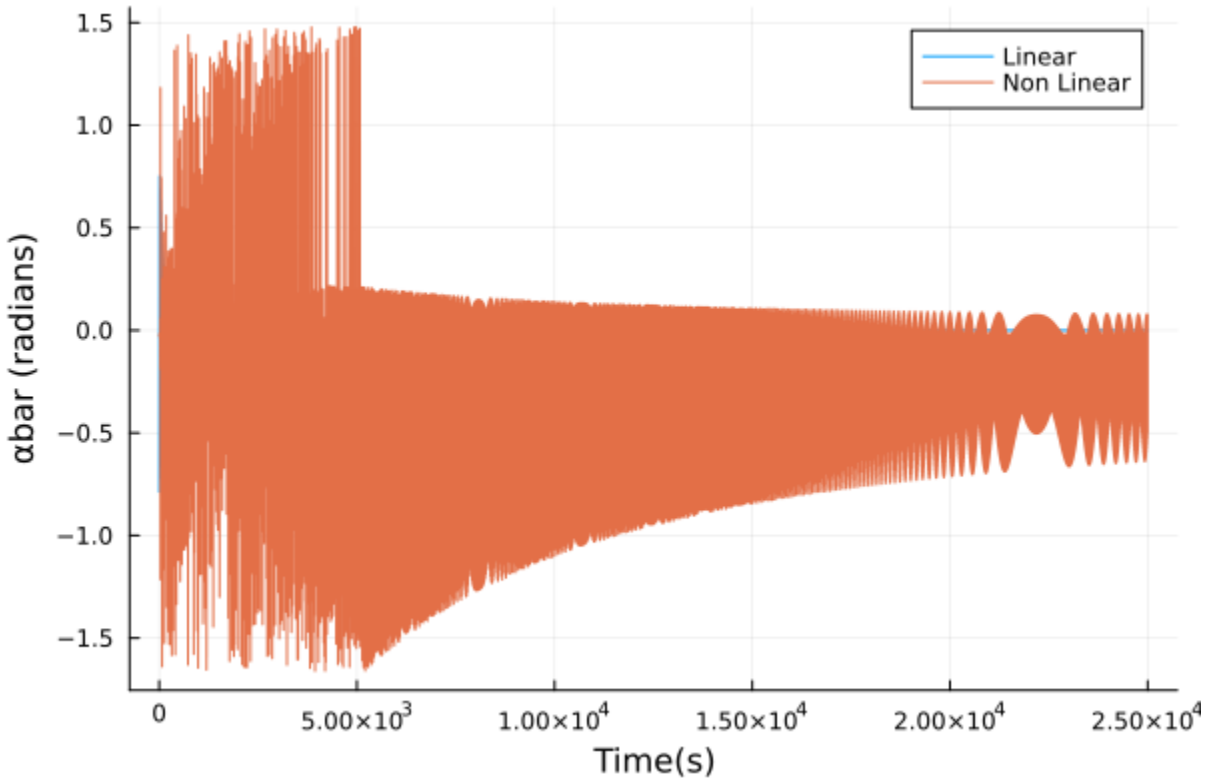

θbar (radians)
15
10
5
0
−5
−10
−15
0
250
500
750
1000
Time(s)
Linear
Non Linear

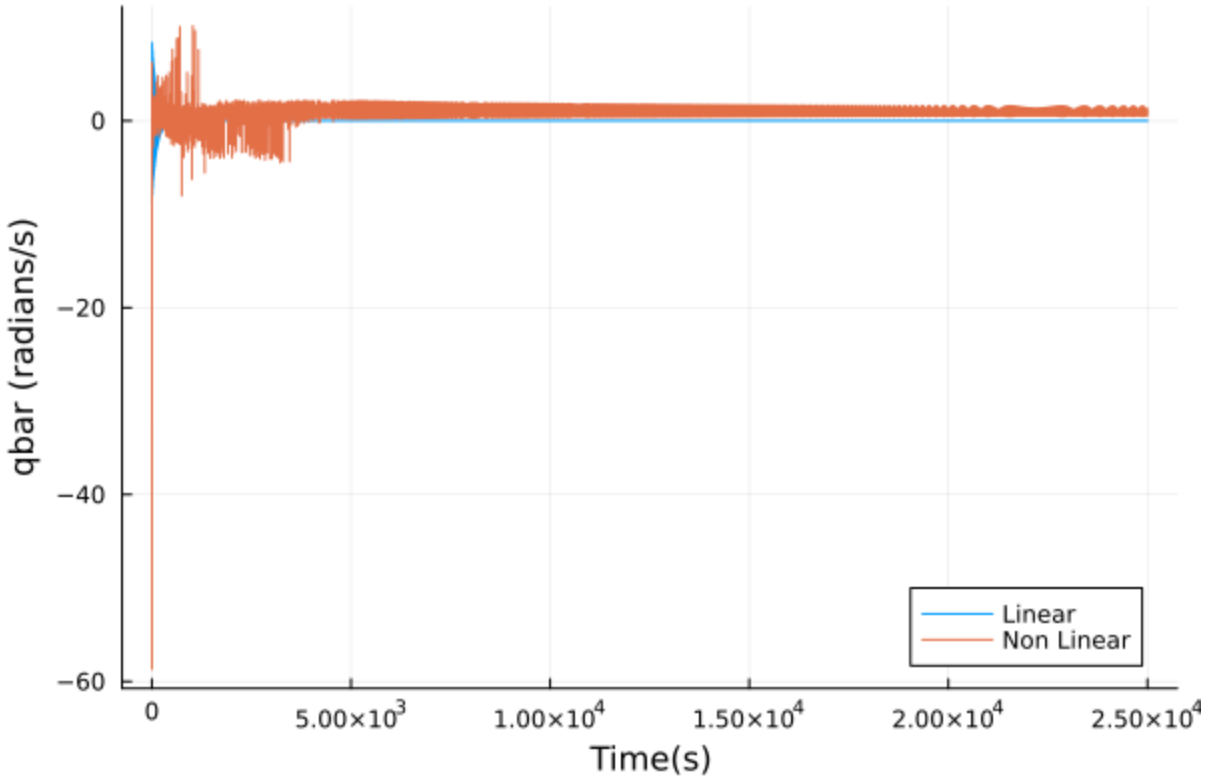


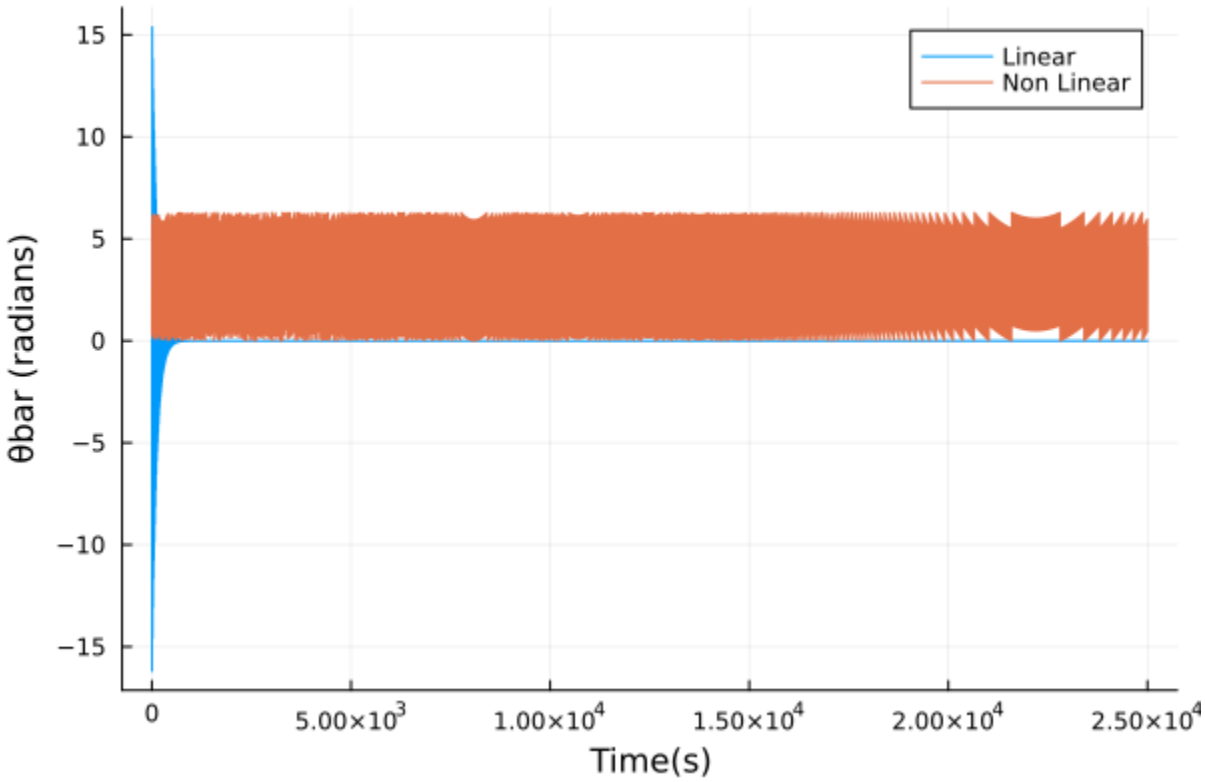
The plots show that the perturbations in u, and q go to zero for both the linearized and nonlinear models. However, there is a lot of noise. The converged perturbation value for pitch shown in the plot is close to 2pi (which, when wrapped, is close to 0). There is a lot of noise in the perturbation of the angle of attack as well. The scaling required to get the initial perturbation of 20 degrees in pitch resulted in a very high perturbation value for all the variables. Since the perturbation is no longer small, our approximated linear model doesn't behave like the more accurate nonlinear model. So, the linear model is no longer a good approximation. This is also visible from the plots when the simulation is run for a really long period of time.

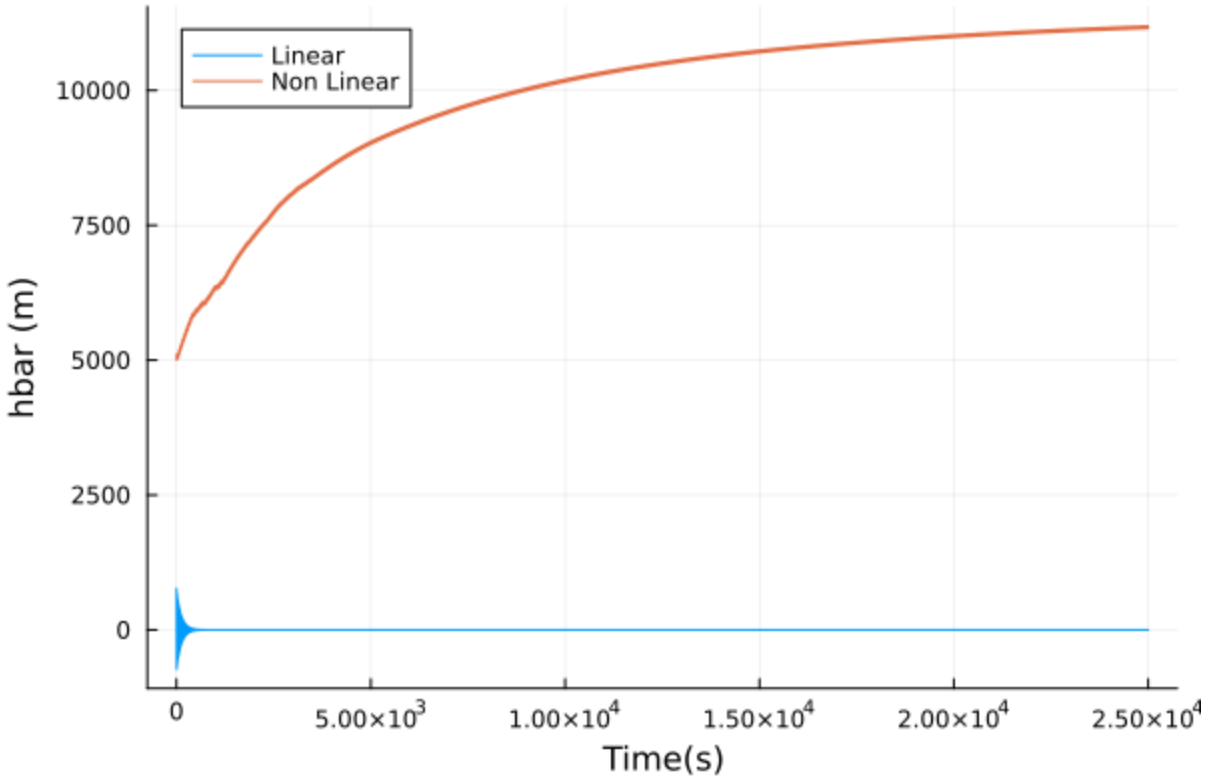
The following plots are for the time interval 0 to 25000 seconds.


ubar (m/s)
0
−500
−1000
−1500
−2000
0
5.00×10^3
1.00×10^4
1.50×10^4
2.00×10^4
2.50×10^4
Time(s)
Linear
Non Linear

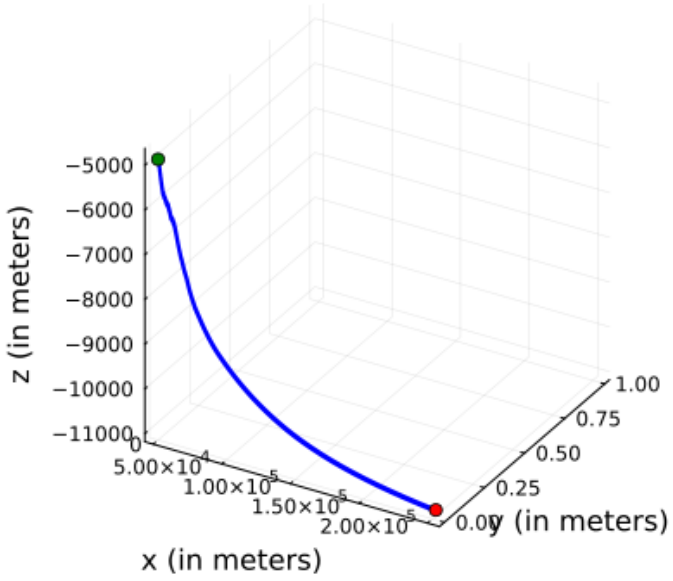

αbar (radians)
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0
5.00×10^3
1.00×10^4
1.50×10^4
2.00×10^4
2.50×10^4
Time(s)
Linear
Non Linear


qbar (radians/s)
0
−20
−40
−60
0
5.00×10^3
1.00×10^4
1.50×10^4
2.00×10^4
2.50×10^4
Time(s)
Linear
Non Linear


θbar (radians)
15
10
5
0
−5
−10
−15
0
5.00×10^3
1.00×10^4
1.50×10^4
2.00×10^4
2.50×10^4
Time(s)
Linear
Non Linear



As visible, the perturbation in h is not going to 0 and is increasing exponentially.



```julia
#Problem 1

function FlightPathAnglesFromState(aircraft_state)

    euler_angles = EulerAngles(aircraft_state[4:6])
    wind_angles = AirRelativeVelocityVectorToWindAngles(aircraft_state[7:9])
    Vg = TransformFromBodyToInertial(aircraft_state[7:9], euler_angles)
    #=
    We know γ_a = γ when there is no wind.
    Thus, γ = γ_a = θ - α
    =#
    # γ = aircraft_state.θ - wind_angles.α
    γ = atan(Vg[3], sqrt(Vg[1]^2 + Vg[2]^2))
    χ = atan(Vg[2], Vg[1])

    return (Vg,χ,γ)
end


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

trim_definition = TrimDefinitionCT(18.0,0.0,1655,500.0)
state, control, results = GetTrimConditions(trim_definition, aircraft_parameters)
trim_variables = TrimVariablesCT(results.minimizer)
Vg, χ, γ = FlightPathAnglesFromState(state)
=#
#Definitions
x = AircraftState(
    100.0,
    200.0,
    -1655.0,
    -12*pi/180,
    9*pi/180,
    140*pi/180,
    15.0,
    -3.0,
    1.0,
    0.08*pi/180,
    -0.2*pi/180,
    0.0*pi/180
    )

wind_inertial = [0, 1, -2]

#P1
ϕ = x.ϕ
q = x.q
r = x.r
θ_dot = q*cos(ϕ) - r*sin(ϕ)

#P2
euler_angles = EulerAngles(x.ϕ, x.θ, x.ψ)
wind_body_frame = TransformFromInertialToBody(wind_inertial, euler_angles)
Va = x[7:9] - wind_body_frame   #Va vector in body frame
wind_angles = AirRelativeVelocityVectorToWindAngles(Va)
α = wind_angles.α

#P3
m = 10
w_dot = 0.05
θ = x.θ
u = x.u
p = x.p
v = x.p
g = 9.81
fZ = m*(w_dot -q*u + p*v)
fZ_gravity = m*g*cos(θ)*cos(ϕ)
fZ_aero = fZ - fZ_gravity
```

```
#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)
```

```
        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
```

```
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)
```