HOMEWORK: EXERCISE 3 - Space Debris

One of our retired satellites was not properly passivized at end of life, and has suffered an internal explosion that has generated space debris. The orbit of the satellite at the time of explosion was given by: ζp = 350 km; e = 0.1; Ω = 195 deg; i = 54 deg; ω = 235 deg; θ = 20 deg.

In [ ]:
```
# Data:
zeta_p = 350.0 # km
e_sat = 0.1
Omega_sat = deg2rad(195.0) # rad
inc_sat = deg2rad(54.0) # rad
omega_sat = deg2rad(235.0) # rad
theta_sat = deg2rad(20.0) # rad
```

```
0.3490658503988659
```

After the explosion, 30 pieces of debris were created with the same initial position r0 as the satellite, but with a dispersion2 in initial velocity v0 that follows a normal distribution with mean equal to the velocity of the satellite, and standard deviation 100 m/s in each direction. The ballistic coefficient of each fragment follows lognormal distribution, with μ = ln(20 kg/m2) and σ = 0.5.

In [ ]:
```
using Pkg
# Pkg.activate(".")
Pkg.activate("C:\\Users\\alici\\Desktop\\Julia\\EXERCISE3")
Pkg.instantiate()
Pkg.add(url="https://github.com/AliciaSBa/AstrodynamicsEdu.jl.git")
Pkg.update()

using AstrodynamicsEdu
using LinearAlgebra
using DifferentialEquations
```

In [ ]:
```julia
N = 30
# Calculate the initial position and velocity vectors
r_p = zeta_p + R_Earth
a_sat = r_p/(1-e_sat)
coe_sat = MyCOE(RAAN = Omega_sat, inc = inc_sat, omega = omega_sat, a = a_sat,
                                                    e = e_sat, theta = theta_sat)
state_sat = COE_to_stateVector(coe_sat,mu_Earth)
# Debris position and velocity
r_0 = state_sat.r
mu_v = state_sat.v
sigma_v = 0.1 # km/s (in each direction)
# Ballistic coefficient - lognormal distribution
mu_bc = log(20.0)
sigma_bc = 0.5
```

```
0.5
```

We want to study how the orbits of the cloud of debris evolve in time, ignoring the possibility that they may subsequently collide among themselves.

(a) Code a function `[v,bc] = explosion(mu_v,sigma_v,mu_bc,sigma_bc)` that generates the (3 x N) array v and the (1 x N) array bc with the initial velocities (in km/s) and the ballistic coefficients (in kg/m2) of N pieces of debris, with distribution parameters `mu_v` and `sigma_v` for the velocity and `mu_bc`, `sigma_bc` for the mass as described above.

In [ ]:
```julia
using Random
# Set the random number generator's seed to 0, to ensure that subsequent runs of the
    # code produce the same random numbers
Random.seed!(0)
rng = Random.MersenneTwister(1234)

# Function to calculate velocity and Bc of each piece
function explosion(mu_v, sigma_v, mu_bc, sigma_bc)
    N = 30
    v = zeros(3, N)
    bc = zeros(1, N)

    # Generate 3 normal distributions (1 per velocity component)
    for i in 1:3
        for j in 1:N
            u1 = rand()
            u2 = rand()
            z1 = sqrt(-2*log(u1))*cos(2*pi*u2)
            v[i, j] = mu_v[i] + z1*sigma_v
        end
    end

    # Obtain the Ballistic coefficients lognormal distribution
    # First calculate the mean and variance of the underlying normal distribution
    mu = log(mu_bc^2/sqrt(sigma_bc^2 + mu_bc^2))
    sigma = sqrt(log(1 + (sigma_bc/mu_bc)^2))

    # Generate N random samples from the log-normal distribution with the
        #given mean and standard deviation
    bc[1, :] = (exp.(mu .+ sigma .* randn(N,1))).*10^6 # kg/km^2

    return v, bc
end

v0,Bc = explosion(mu_v,sigma_v,mu_bc,sigma_bc)
```

```
println("Velocity of each piece: ",v0, " km/s")
println("Ballistic coefficient of each piece: ",Bc, " kg/km^2")
```

```
Velocity of each piece: [-7.6629393269444925 -7.738283858976623 -7.7173204909554585 -7.8474809
35733069 -7.630070452467509 -7.696254727298384 -7.87221879201388 -7.676887194957834 -7.7420593
119813965 -7.885801796923632 -8.070934180456844 -7.703701705334888 -7.901924028889331 -7.87760
7295483539 -7.838986571520096 -7.736517678641874 -7.916577725083215 -7.650507326615872 -7.8256
71641091026 -7.723903408710842 -7.765523239448662 -7.800280797947573 -7.940712435740529 -7.898
833678652749 -7.711711070725511 -7.859423467230367 -7.764918705014169 -7.823957734116187 -7.78
2450204588986 -8.027895560280653; -0.5734555253366086 -0.6595586156075784 -0.7092307321905793
-0.5822622184572072 -0.7054408539185247 -0.6944872946529371 -0.8006808170375868 -0.69404460114
82864 -0.6406423646571583 -0.7696529728107184 -0.7592531668273151 -0.7473968976855457 -0.84392
77347424528 -0.5720234645409044 -0.6265175158026375 -0.8091882834222964 -0.7070338923337296 -
0.7579143749824253 -0.8648002902894655 -0.5984475371268543 -0.7279945827555802 -0.649923291587
6587 -0.6183411034893357 -0.7638676950220178 -0.6000259826109311 -0.5528931755814932 -0.718297
0222515797 -0.7241943206951555 -0.8415146278523714 -0.6207993915686888; -1.7048073120622609 -
1.8047212975460851 -1.9357039700353258 -1.8576705596746446 -1.9155328087527639 -1.957638280276
7247 -1.838560129117264 -1.848795356509331 -1.940239090013048 -1.7461330133421722 -1.963524314
1912864 -2.046343227780515 -1.8383746478186571 -2.0104188673213814 -1.9467989258383283 -2.1059
601432223443 -1.7671448500154905 -1.7376846674679445 -1.7424697917613199 -1.908404271681895 -
1.8705664377017575 -1.72227449239178 -2.0755429087637385 -1.9088693093331681 -1.92692934683782
7 -1.8576700871589729 -1.9265515390024979 -1.8764807178934508 -1.9452058074777285 -1.808632145
8379022] km/s
Ballistic coefficient of each piece: [2.430723586335961e6 2.9054543372566802e6 3.258220425114
033e6 2.538867165270325e6 2.071980201351082e6 3.2028641552824564e6 2.7802652379787965e6 2.32590
69977419465e6 2.378518664015541e6 2.189976456388036e6 2.642568213486311e6 2.489284098402383e6
3.035514739874384e6 2.607242997399797e6 3.394098500040479e6 2.7357543505942053e6 2.76407808369
72333e6 2.711509853021118e6 3.8004484831795366e6 2.349247636981357e6 2.4882897989726495e6 2.84
0668652866057e6 2.6552409417419103e6 2.723477406032103e6 3.10251207839299e6 4.353135078049892e
6 3.733188007308912e6 2.1615818428855366e6 3.2582293192650992e6 3.2873022034945018e6] kg/km^2
```

(b) Implement a function `ap = drag_acceleration(zeta, v, bc)` that returns the perturbation acceleration vector (in km/s2) due to drag on an object at altitude zeta, with velocity vector v (in km/s) and ballistic coefficient BC. Assume a simple isothermal atmosphere model with scale height H = 50 km and ρ0 = 10^-8 kg/m3 at ζ0 = 100 km. To avoid issues with the drag acceleration, set it to NaNs when the altitude of the object is below 100 km (this will result in NaNs in the subsequent time integration below, and remove those points from the plots).

The equivalent equation in the AstrodynamicsEdu library would be: `ap_drag = drag_acceleration(alt,v,Bc)`

(c) Implement a function `ap = J2_acceleration(r)` that returns the perturbation acceleration vector (in km/s2) due to the J2 coefficient of the non-sphericity of the Earth on an object with position vector r in ECI (in km).

The equivalent equation in the AstrodynamicsEdu library would be: `ap_J2 = J2_acceleration(r)`

(d) Establish a Cowell propagator `[r,v] = cowell(r0,v0,bc,t,flag_drag,flag_J2)` that integrates the trajectory of multiple objects, with (3 x N) arrays r0 and v0 giving their initial positions and velocities. Use ode45 (with the right settings!) to perform the integration for each object. The vector t, of length M, is the vector of time instants at which the outputs must be provided. The outputs r and v are (3 x N x M) arrays with the integration results for each object. flag drag and flag J2 are Booleans that activate the corresponding perturbations when true.

This function exists in the AstrodynamicsEdu library by the same name: `r,v = cowell(r0, v0, bc, t, flag_drag, flag_J2)`, the only difference is that it does not use the ode45, but instead the

DifferentialEquations and the OrdinaryDiffEq package to integrate the ODE. The flags must be equalt to either `true` or `false`.

(e) Simulate the trajectories of the cloud of debris for 6 months (1) without perturbations (2) with the drag perturbation and (3) with both perturbations. For each case, generate plots of (i) perigee and apogee altitudes ζp, ζa vs orbital period τ (Gabbard plot), (ii) e vs a, and (iii) Ω vs a for the state of the cloud of debris on t = 0. Using different colors, overlay the state of the cloud of debris every 7 days afterward, up to 6 months. Discuss your results.

We have 3 different cases for which we have to simulate the cloud of debris:

Case 1: Without perturbations

Case 2: With the drag pertubation

Case 3: With both the drag and the J2 perturbation

This will be done by using the cowell method to propagate the trajectories of the cloud of debris of each case.

In [ ]:
```
# Time span (6 months)
M = 26
t_final = 6*30*24*60*60 # s
t = range(0.0,t_final,M) # s
t = collect(t)

# Case 1: Without perturbations
flag_drag = false
flag_J2 = false
r_c1 = zeros(3,M,N)
v_c1 = zeros(3,M,N)
for i=1:N
    v_0 = v0[:,i]
    r_c1[:,:,i], v_c1[:,:,i] = cowell(r_0.c0, v_0, Bc[i], t, flag_drag, flag_J2)
end

# Case 2: With Drag perturbation
flag_drag = true
flag_J2 = false
r_c2 = zeros(3,M,N)
v_c2 = zeros(3,M,N)
for i=1:N
    v_0 = v0[:,i]
    r_c2[:,:,i], v_c2[:,:,i] = cowell(r_0.c0, v_0, Bc[i], t, flag_drag, flag_J2)
end

# Case 3: With both perturbations
flag_drag = true
flag_J2 = true
r_c3 = zeros(3,M,N)
v_c3 = zeros(3,M,N)
for i=1:N
    v_0 = v0[:,i]
    r_c3[:,:,i], v_c3[:,:,i] = cowell(r_0.c0, v_0, Bc[i], t, flag_drag, flag_J2)
end
```

In [ ]:

```
#Predifine the arrays
a1 = zeros(M,N)
e1 = zeros(M,N)
RAAN1 = zeros(M,N)
hp1 = zeros(M,N)
ha1 = zeros(M,N)
tau1 = zeros(M,N)
a2 = zeros(M,N)
e2 = zeros(M,N)
RAAN2 = zeros(M,N)
hp2 = zeros(M,N)
ha2 = zeros(M,N)
tau2 = zeros(M,N)
a3 = zeros(M,N)
e3 = zeros(M,N)
RAAN3 = zeros(M,N)
hp3 = zeros(M,N)
ha3 = zeros(M,N)
tau3 = zeros(M,N)

for j=1:N
    for i=1:M
        # Case 1 (no perturbations)
        coe1 = stateVector_to_COE(MyStateVector(MyVector(r_c1[:,i,j]),
                                  MyVector(v_c1[:,i,j])),mu_Earth)
        a1[i,j] = coe1.a
        e1[i,j] = coe1.e
        RAAN1[i,j] = coe1.RAAN
        hp1[i,j] = coe1.a * (1 - coe1.e) - R_Earth
        ha1[i,j] = coe1.a * (1 + coe1.e) - R_Earth
        tau1[i,j] = 2*pi*sqrt(coe1.a^3/mu_Earth)
        # Case 2 (drag perturbation)
        coe2 = stateVector_to_COE(MyStateVector(MyVector(r_c2[:,i,j]),
                                  MyVector(v_c2[:,i,j])),mu_Earth)
        a2[i,j] = coe2.a
        e2[i,j] = coe2.e
        RAAN2[i,j] = coe2.RAAN
        if coe2.a < 0
            tau2[i,j] = NaN
            hp2[i,j] = NaN
            ha2[i,j] = NaN
        else
            tau2[i,j] = 2*pi*sqrt(coe2.a^3/mu_Earth)
            hp2[i,j] = coe2.a * (1 - coe2.e) - R_Earth
            ha2[i,j] = coe2.a * (1 + coe2.e) - R_Earth
        end
        # Case 3 (drag and J2 perturbations)
        coe3 = stateVector_to_COE(MyStateVector(MyVector(r_c3[:,i,j]),
                                  MyVector(v_c3[:,i,j])),mu_Earth)
        a3[i,j] = coe3.a
        e3[i,j] = coe3.e
        RAAN3[i,j] = coe3.RAAN

        if coe3.a < 0
            tau3[i,j] = NaN
            hp3[i,j] = NaN
            ha3[i,j] = NaN
        else
            tau3[i,j] = 2*pi*sqrt(coe3.a^3/mu_Earth)
            hp3[i,j] = a3[i,j] * (1 - e3[i,j]) - R_Earth
            ha3[i,j] = a3[i,j] * (1 + e3[i,j]) - R_Earth
        end
    end
```

```
end

n0 = sqrt(mu_Earth/a_sat^3)
tau = 2*pi/n0
t_period = t/tau
```

```
26-element Vector{Float64}:
    0.0
   96.860828501153
  193.721657002306
  290.582485503459
  387.443314004612
  484.30414250576496
  581.164971006918
  678.025799508071
  774.886628009224
  871.747456510377
    ⋮
 1646.634084519601
 1743.494913020754
 1840.355741521907
 1937.2165700230598
 2034.0773985242129
 2130.938227025366
 2227.7990555265187
 2324.659884027672
 2421.520712528825
```

Now, in order to be able to intepret the results we are going to generate the following plots for each case:

(i) perigee and apogee altitudes ζp, ζa vs orbital period τ (Gabbard plot)

(ii) e vs a

(iii) Ω vs a

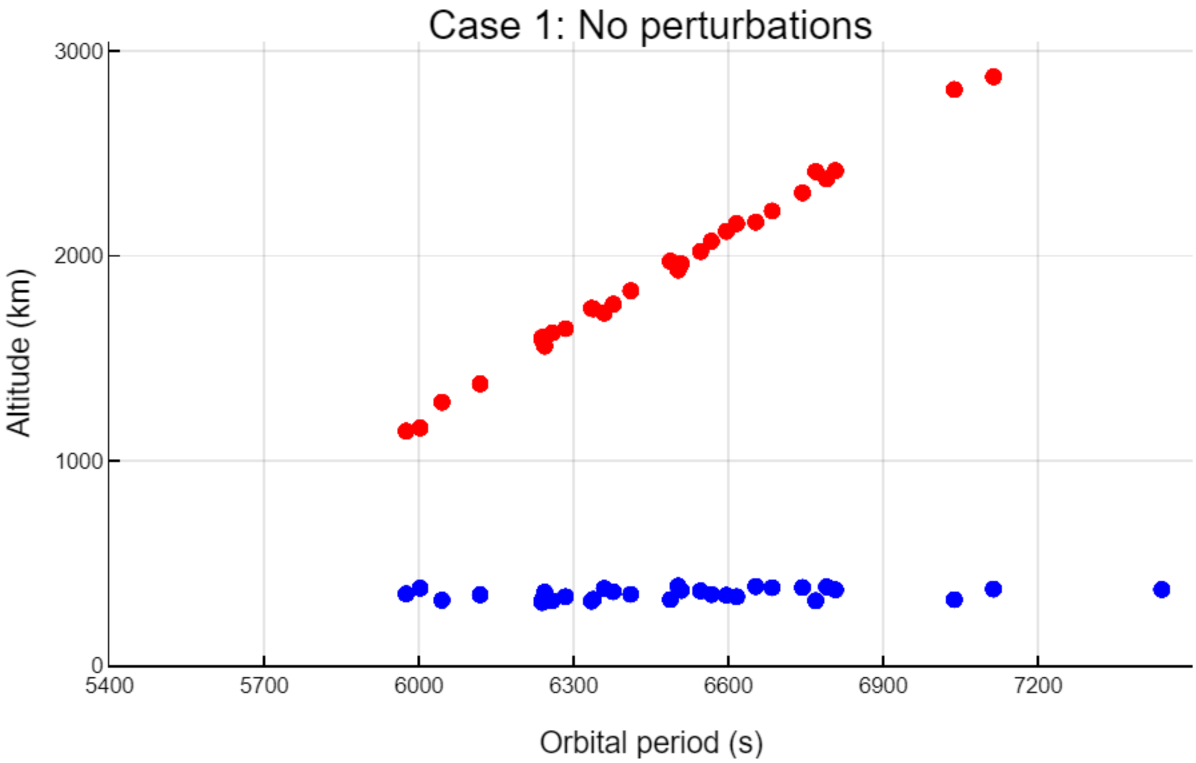
For (i), the apogee is in red and the perigee altitude in blue, and the scatter points become more faded with time. Where for (ii) and (iii), each color represents a different piece of debris, and they are more faded as more time has passed. This way, the later states of space debris are the more faded points, and we can see their evolution.

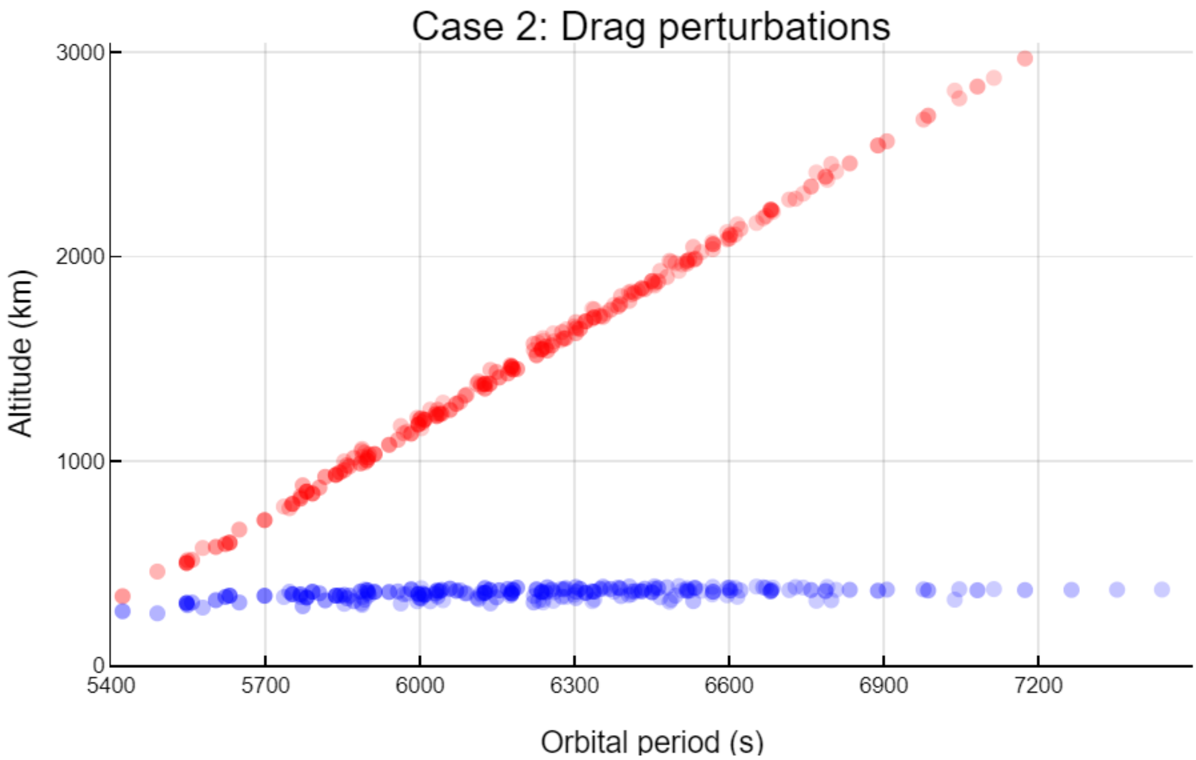
In [ ]:

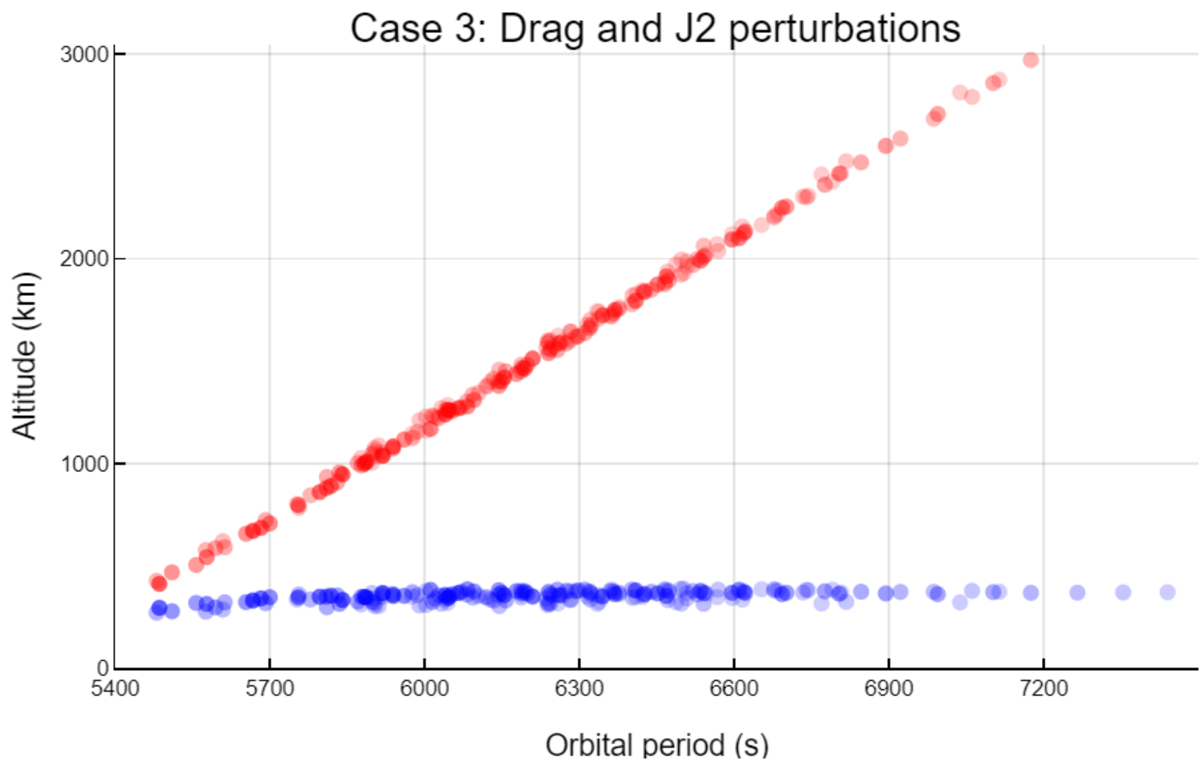
```
using Plots
plotly()

# 1. Gavvard plots

function plot_Gavvard(tau,ha,hp)
    # Define the color gradient
    function fade_colorBlue(alpha)
        RGBA(0, 0, 1, alpha)
    end

    function fade_colorRed(alpha)
        RGBA(1, 0, 0, alpha)
    end

    # Calculate the alpha values for each point
    alphas = LinRange(0.2, 1, length(tau))
    # Apogee altitude
    scatter!(tau,ha,label="ha",color=fade_colorRed.(alphas), markerstrokewidth=0,
                legend=false, xlims=(5400,7500), ylims=(0,3050))
```

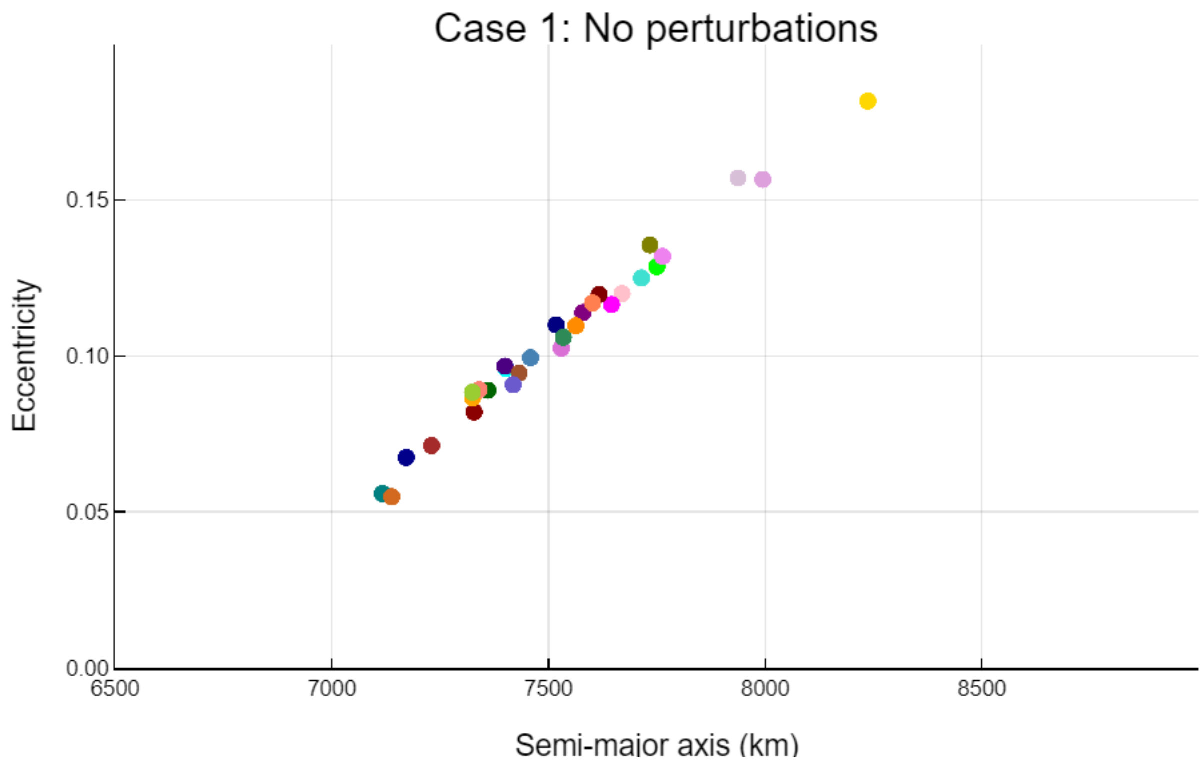
```
    # Perigee altitude
    scatter!(tau,hp,label="hp",color = fade_colorBlue.(alphas), markerstrokewidth=0,
                        legend = false, xlims=(5400,7500), ylims=(0,3050))
    xlabel!("Orbital period (s)")
    ylabel!("Altitude (km)")
end

figure1 = plot()
for i = 1:N
    title!(figure1, "Case 1: No perturbations")
    plot_Gavvard(tau1[:,i],ha1[:,i],hp1[:,i])
end
display(figure1)

figure2 = plot()
for i = 1:N
    title!(figure2, "Case 2: Drag perturbations")
    plot_Gavvard(tau2[:,i],ha2[:,i],hp2[:,i])
end
display(figure2)

figure3 = plot()
for i = 1:N
    title!(figure3, "Case 3: Drag and J2 perturbations")
    plot_Gavvard(tau3[:,i],ha3[:,i],hp3[:,i])
end
display(figure3)

# Define N different colors
colors = [:teal, :darkred, :darkgreen, :purple, :darkblue, :orange, :pink, :brown, :cyan,
          :magenta, :gold, :indigo, :lime, :olive, :maroon, :navy, :turquoise, :chocolate,
          :orchid, :salmon, :sienna, :slateblue, :thistle, :violet, :yellowgreen, :coral,
          :steelblue, :darkorange, :seagreen, :plum]
rgb_colors = [ColorTypes.color(string(name)) for name in colors]

# 2. e vs. a plot
function plot_e_vs_a(a,e,colorVal)

    # Define the color gradient
    function fade_color(colorVal,alpha)
        RGBA(colorVal, alpha)
    end

    # Calculate the alpha values for each point
    alphas = LinRange(0.2, 1, length(a))

    scatter!(a,e, legend=false, xlims=(6500,9000), ylims=(0,0.2),
                    color = fade_color.(colorVal, alphas), markerstrokewidth=0)
    xlabel!("Semi-major axis (km)")
    ylabel!("Eccentricity")
end

figure4 = plot()
for i = 1:N
    title!(figure4, "Case 1: No perturbations")
    plot_e_vs_a(a1[:,i],e1[:,i], rgb_colors[i])
end
display(figure4)

figure5 = plot()
for i = 1:N
    title!(figure5, "Case 2: Drag perturbations")
    plot_e_vs_a(a2[:,i],e2[:,i], rgb_colors[i])
```

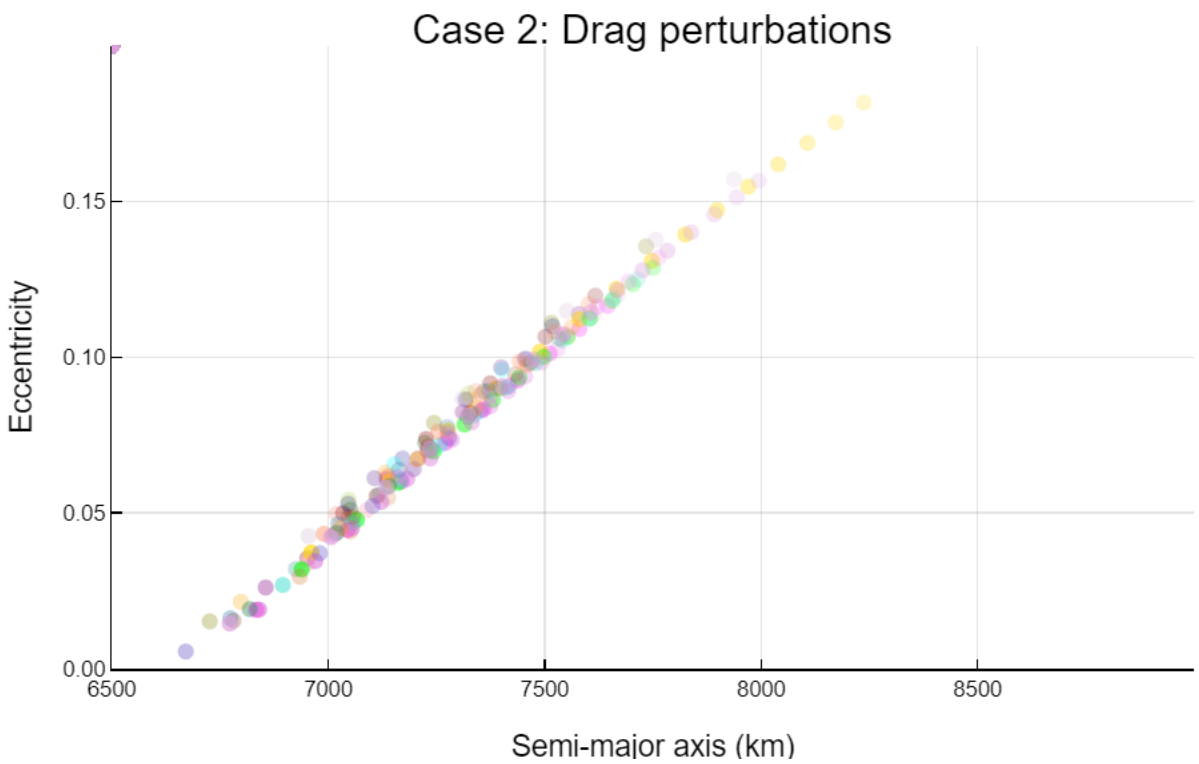
```
end
display(figure5)

figure6 = plot()
for i = 1:N
    title!(figure6, "Case 3: Drag and J2 perturbations")
    plot_e_vs_a(a3[:,i],e3[:,i], rgb_colors[i])
end
display(figure6)

# 3. RAAN vs. a plot
function plot_RAAN_vs_a(a,RAAN,colorVal)
    # Define the color gradient
    function fade_color(colorVal,alpha)
        RGBA(colorVal, alpha)
    end

    # Calculate the alpha values for each point
    alphas = LinRange(0.2, 1, length(a))
    scatter!(a,RAAN, legend=false, xlims = (6500,8000), markerstrokewidth=0,
                                    color = fade_color.(colorVal, alphas))
    xlabel!("Semi-major axis (km)")
    ylabel!("RAAN (rad)")
end

figure7 = plot()
for i = 1:N
    title!(figure7, "Case 1: No perturbations")
    plot_RAAN_vs_a(a1[:,i],RAAN1[:,i],rgb_colors[i])
end
display(figure7)

figure8 = plot()
for i = 1:N
    title!(figure8, "Case 2: Drag perturbations")
    plot_RAAN_vs_a(a2[:,i],RAAN2[:,i],rgb_colors[i])
end
display(figure8)

figure9 = plot()
for i = 1:N
    title!(figure9, "Case 3: Drag and J2 perturbations")
    plot_RAAN_vs_a(a3[:,i],RAAN3[:,i],rgb_colors[i])
end
display(figure9)
```


Case 1: No perturbations
Altitude (km)
3000
2000
1000
0
5400
5700
6000
6300
6600
6900
7200
Orbital period (s)

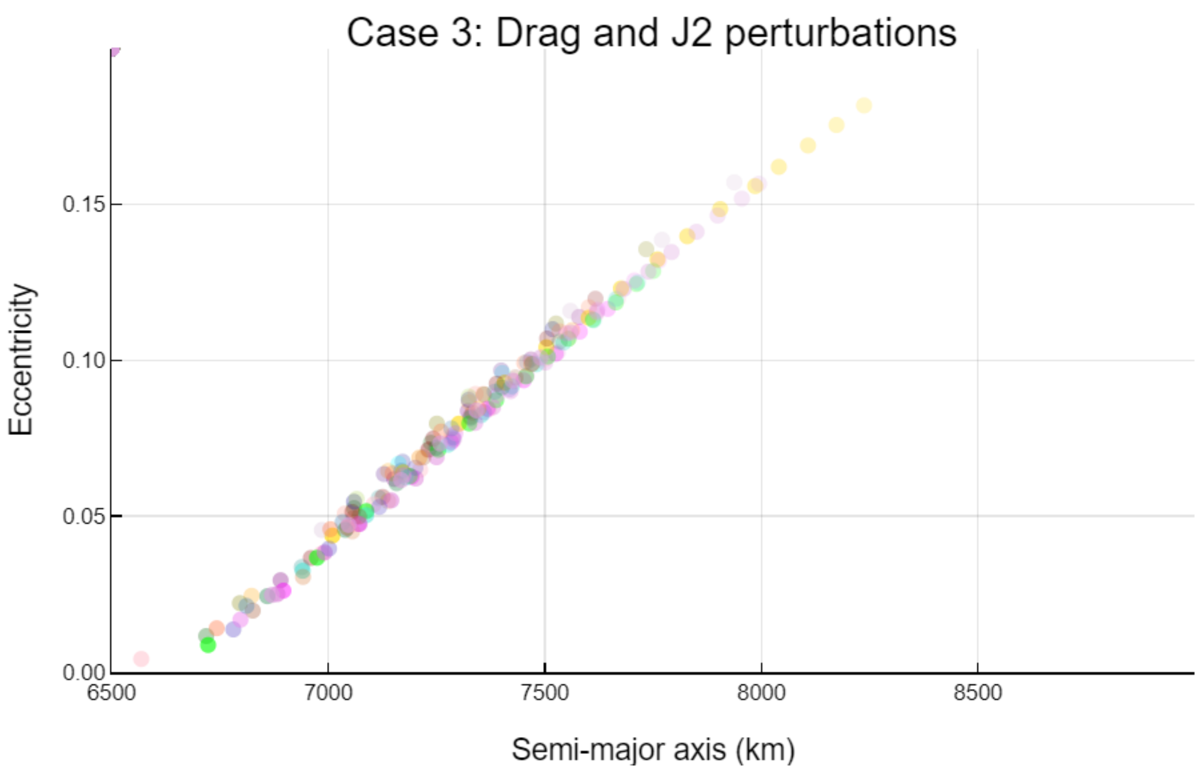

Case 2: Drag perturbations
Altitude (km)
3000
2000
1000
0
5400
5700
6000
6300
6600
6900
7200
Orbital period (s)


Case 3: Drag and J2 perturbations
Altitude (km)
3000
2000
1000
0
5400
5700
6000
6300
6600
6900
7200
Orbital period (s)

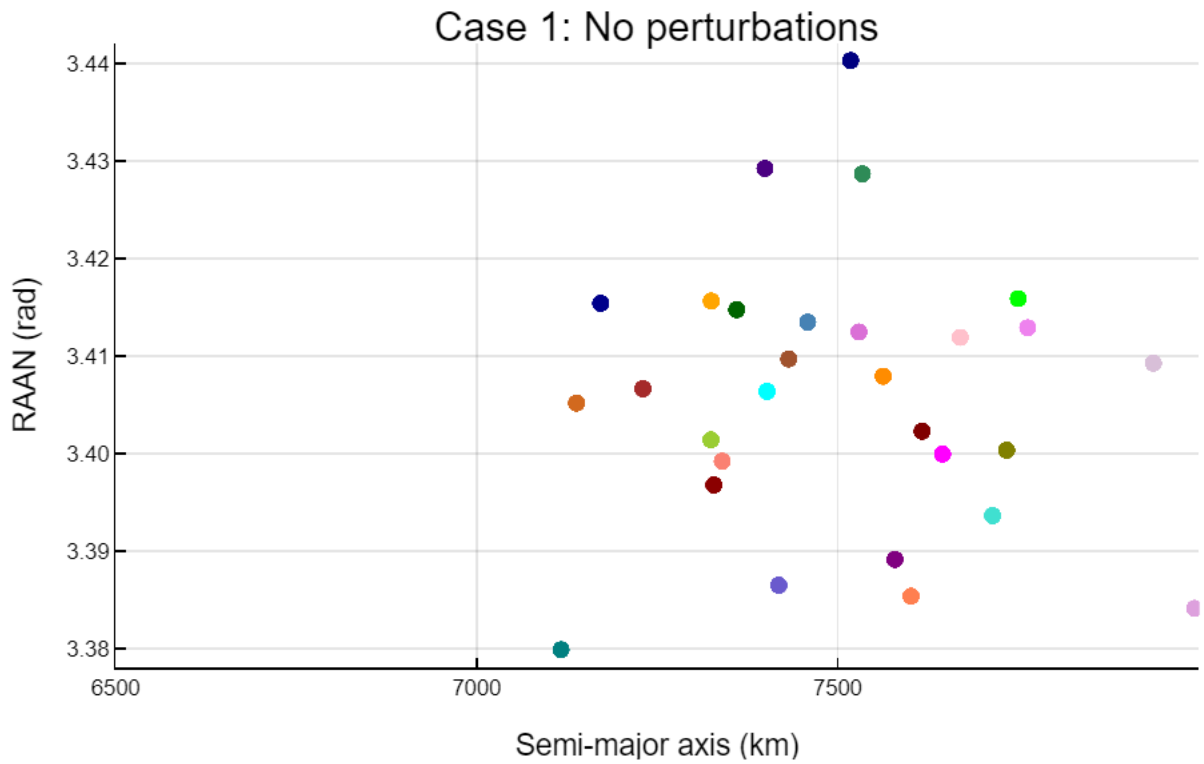

Case 1: No perturbations
Eccentricity
0.15
0.10
0.05
0.00
6500
7000
7500
8000
8500
Semi-major axis (km)

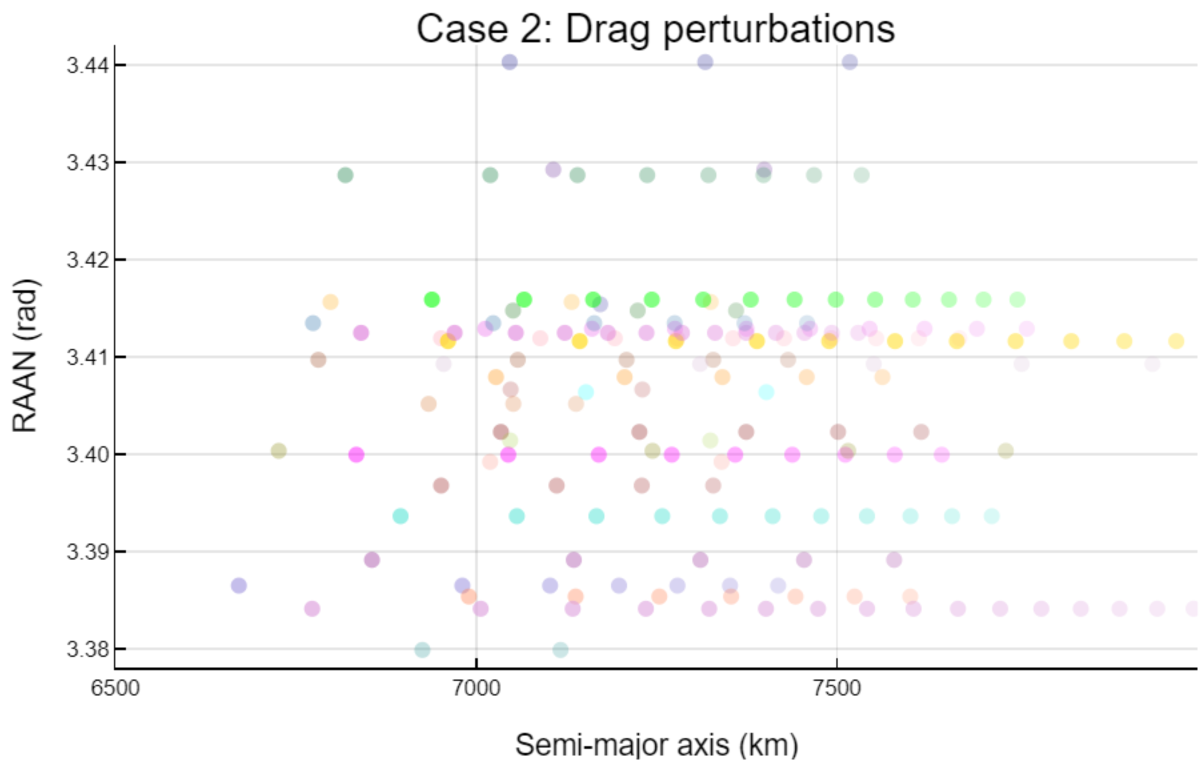


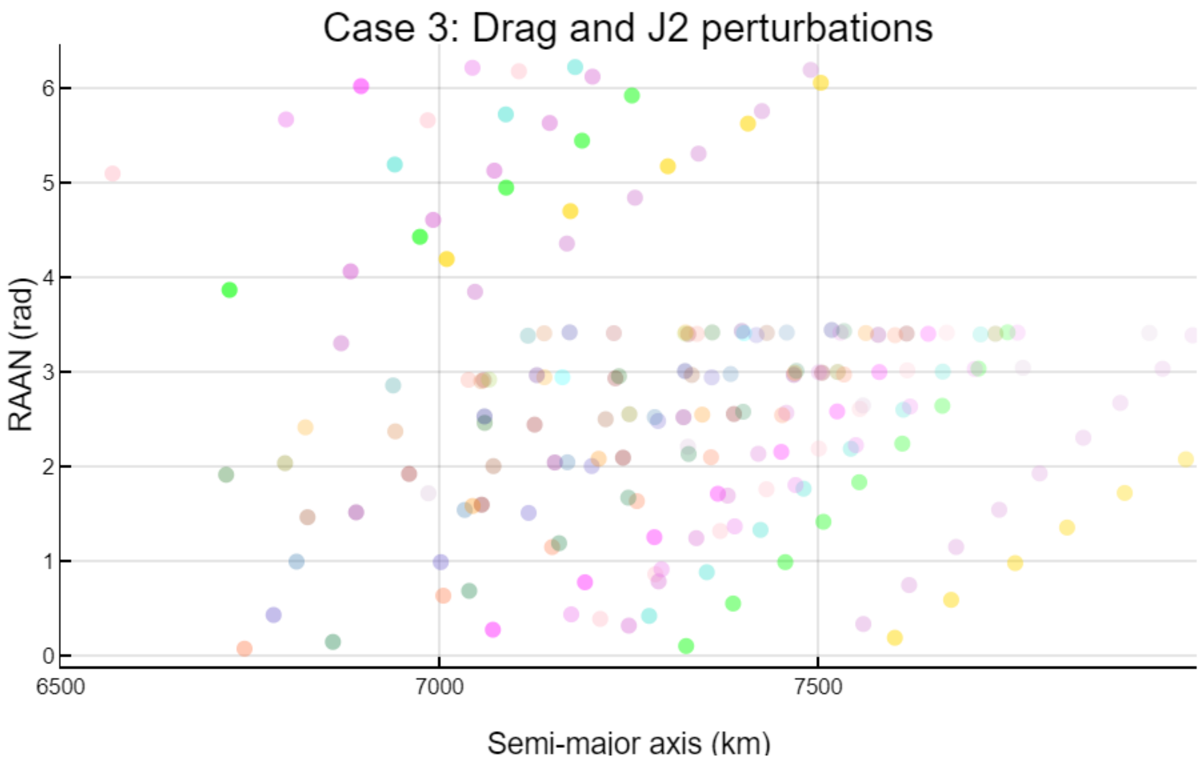
Case 2: Drag perturbations
Eccentricity
0.00
0.05
0.10
0.15
6500
7000
7500
8000
8500
Semi-major axis (km)



Case 3: Drag and J2 perturbations
Eccentricity
0.00
0.05
0.10
0.15
6500
7000
7500
8000
8500
Semi-major axis (km)


Case 1: No perturbations
RAAN (rad)
3.44
3.43
3.42
3.41
3.40
3.39
3.38
6500
7000
7500
Semi-major axis (km)


Case 2: Drag perturbations
RAAN (rad)
3.44
3.43
3.42
3.41
3.40
3.39
3.38
6500
7000
7500
Semi-major axis (km)



As we can see from the plots:

Case 1: No perturbations result in constant orbital altitudes and period for each fragment. The eccentricity and semi-major axis remain constant. It also results in constant values for both RAAN and semi-major axis.

Case 2: Atmospheric drag causes the orbital period to decrease over time. Perigee altitudes remain constant, while apogee altitudes decrease, leading to the reduction in the period.The rate of loss of eccentricity per kilometer of semi-major axis is constant due to atmospheric drag. The relationship between eccentricity and semi-major axis is not linear but inversely proportional. The drag circularizes the orbit. Atmospheric drag causes variations in the semi-major axis while keeping the RAAN constant. Only parameters within the orbital plane are affected by time variations.

Case 3: Drag and Earth's oblateness perturbations cause both apogee and perigee altitudes to decrease over time, resulting in a decrease in the orbital period. The decrease in apogee altitude is more abrupt due to the combined perturbations, while perigee altitude is affected only by Earth's oblateness. The relationship between eccentricity and semi-major axis remains inversely proportional. The effect of perturbations (drag and oblateness) is primarily on the semi-major axis, reducing it faster. Both RAAN and semi-major axis decrease over time due to drag and oblateness perturbations. The oblateness perturbation affects the RAAN, leading to its change over time. The rate of change in RAAN is influenced by J2 and (Re / a)^2, with stronger effects for lower altitudes and higher inclinations.

Therefore, the main effect of the drag perturbation is that it decreases the apocenter and circularizes the orbit. Whereas, the main effect of the J2 perturbation is that the RAAN changes, and the eccentricity therefore decreases.