

MOCAT-4S Workbook

Description

MOCAT SSEM (MIT Orbital Capacity Assessment Tool-- Source Sink Model) investigates the evolution of the space objects (SOs) population in LEO by exploiting a new probabilistic source-sink model, with the objective of estimating the LEO orbital capacity. This is carried out through the long-term propagation of the proposed source-sink model, which globally takes into account different object species, such as active satellites, derelict satellites and debris.

The MOCAT-4S model contains the following species: slotted satellites (S), unslotted satellites (Su), derelict (D), and debris (N).

This file is a workbook that runs the model as well as provides notes for the code.

Table 1: Interactions among the species of the MOCAT-4S model.

	Species	S (Slotted satellites)	D (Derelicts)	N (Debris)	S _u (Unslotted satellites)
A New Launches	-	λ_s	0	0	λ_u
C _{PMD} Post- Mission Disposal	-	$-\frac{S}{\Delta t}$	$\frac{(1 - P_M)(S + S_u)}{\Delta t}$	0	$-\frac{S_u}{\Delta t}$
F Drag	-	0	$\dot{F}_{d,D}$	$\dot{F}_{d,N}$	0
C Collision Source	S	$-(1 - \zeta)\alpha_a\phi_{1,1}S^2$	$\phi_{1,2}\delta DS + \phi_{1,3}\delta NS$	$n_{f,13}\phi_{1,3}\alpha SN + n_{f,12}\phi_{1,2}\alpha SD + (1 - \zeta)n_{f,11}\alpha_a\phi_{1,1}S^2 + n_{f,14}\alpha_a\phi_{1,4}SS_u$	$-\alpha_a\phi_{1,4}\frac{SS_u^2}{S + S_u}$
	D	$-\phi_{1,2}(\delta + \alpha)SD$	$-\phi_{2,2}D^2 + \phi_{2,3}n_{f,23}DN$	$n_{f,22}\phi_{2,2}D^2 + \phi_{2,3}n_{f,23}DN$	$-\phi_{4,2}(\delta + \alpha)S_u D$
	N	$-\phi_{1,3}(\delta + \alpha)SN$	$-\phi_{2,3}DN$	$n_{f,33}\phi_{3,3}N^2$	$-\phi_{4,3}(\delta + \alpha)S_u N$
	S _u	$-\alpha_a\phi_{1,4}\frac{S_u S^2}{S + S_u}$	$\phi_{4,2}\delta DS_u + \phi_{4,3}\delta NS_u$	$n_{f,43}\phi_{4,3}\alpha S_u N + n_{f,42}\phi_{4,2}\alpha S_u D + n_{f,44}\alpha_a\phi_{4,4}S_u^2$	$-\alpha_a\phi_{4,4}S_u^2$

Species Names: S- Slotted Satellites, D- Derelict, N- Debris, Su- Unslotted Satellites

Note: S in MOCAT-3 correspond to Su in MOCAT-4S.

Parameter Definitions:

- λ : satellite launch rate
- α : probability of unsuccessful collision avoidance between an active satellite and an inactive object
- α_a : probability of unsuccessful collision avoidance between two active satellites. $\alpha_a < \alpha$
- ϕ_{XY} : probability of collision between any species X and Y (i.e., S, D, and N), approximated from the kinetic theory of gases

- δ : ratio of the density of disabling to lethal debris (this term considers the possibility that disabling collisions can generate new derelicts)
- ζ : slotting effectiveness
- P_M : probability of success of post mission disposal
- Δt : time in which satellites are active before PMD
- \dot{F}_d : time derivative of species amount in a shell due to atmospheric drag
- n_f : number of debris fragments resulting from a collision between any species X and Y
- Set up paths and clear workspace to run file.

How to use it:

This MATLAB Workbook is intended to be a straightforward, clear and interactive way to build familiarity with the MOCAT-SSEM model framework. The code is explained in comments in as much detail as possible for ease of use and clarity. **This workbook is intended to be a simple introduction and does not describe all functionality of the MOCAT-SSEM framework.**

Each model is built with a top-level simulation object. The simulation object has a `scen_properties` object, which sets global parameters, and a set of species with their own `species_properties` objects that define parameters for that species. The simulation object will automatically handle equation compilation and numerical integration of the model based on the provided parameters.

The recommended workflow begins with defining the `scen_properties` object and species before assembling and running the model.

To begin, you set the properties of the scenario you want to run such as the altitudes to be considered and atmospheric model to use, in **Define Scenario Properties**. Simply change the values that the variables are set to.

As explained previously, MOCAT-4S considers 4 different species types, S, Su, D and N. You can set the properties of these species, such as the mass of derelict objects and launch rate of satellites in **Define Species Properties**.

The sections **Collision Modeling** and **Define Initial population** require no user input but are necessary code. The first calculates the population gain/loss between species as they collide. The second, takes in the scenario properties you set earlier and a list of satellites from the file ('initialized.mat') and creates an array (`x0`) of the number of species in each shell for the model at the start of the simulation.

The **Build Model** section compiles the source sink equations, and the **Run Model** section runs the integrator.

You can select which, if any, visualizations of results you want to see in the final section, **Plots**. The plot for the evolution of total species numbers in each shell is already selected in `my_sim.total_species_evol_vis();`. To see the other visualizations, simply uncomment them (remove the % at the start of the line). You can also export numerical results for additional analysis as you desire.

Once you've finished all the inputs you'd like, simply run this workbook (F5) and it will begin.

Set up paths and clear workspace to run file.

This is necessary so that the model can find the necessary classes and functions. Clearing your workspace is not strictly necessary, but using an existing workspace may result in unintended interactions.

```
clear
clc

% The demo workbooks are in a subfolder of the main directory. We need to
% add the path recursively for the main directory.
fileName = matlab.desktop.editor.getActiveFilename;
[folder, ~, ~] = fileparts(fileName);
folder = fullfile(folder, '..');
addpath(fileparts(folder));
addpath(genpath(folder));
cd(folder);
```

Define Scenario Properties

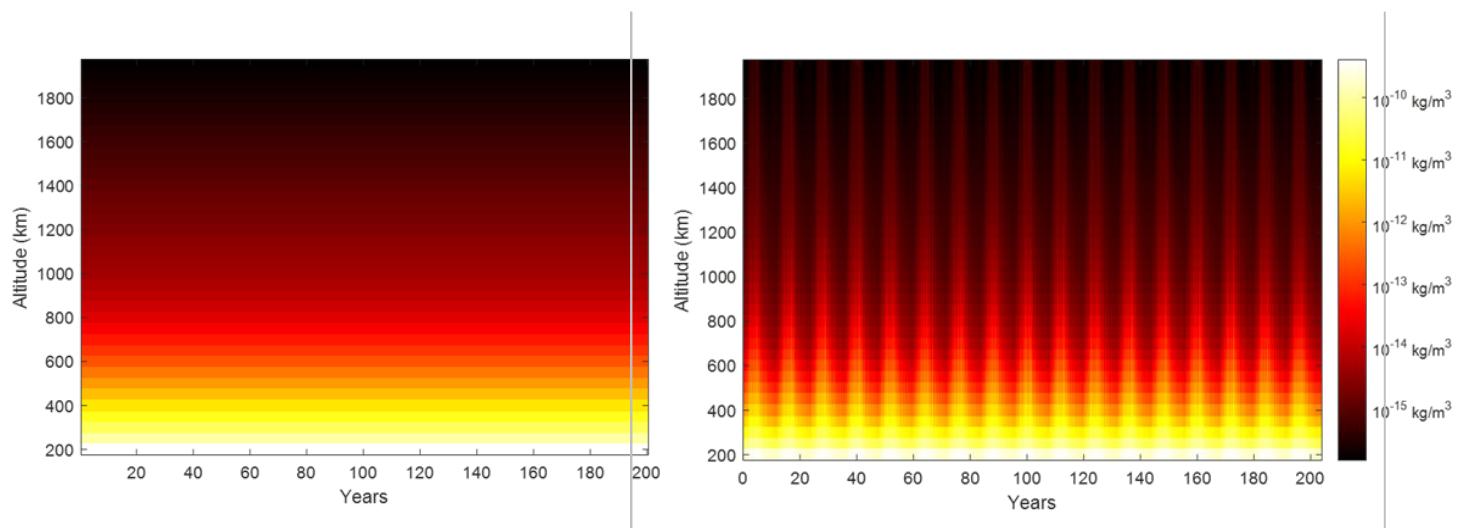
Set simulation parameters such as start time, duration, and time steps:

```
scenario_properties = MOCATSSEM_Scen_Prop_Cons( ...
    start_date = datetime(2022,1,30,9,42,28), ... # Starting date (year,
month, day, hour, minute, second)
    simulation_duration = 100.0, ... % Years of simulation to run
    steps = 1000, ... % Number of steps to run in simulation
    min_altitude = 200, ... % Maximum altitude shell in km
    max_altitude = 2000, ... % Minimum altitude shell in km
    shells = 36, ... % Number of altitude shells
    ... % We create this many equally spaced shells between the minimum and
maximum altitudes
    delta = 10, ... %ratio of the density of disabling to lethal debris
    ... % this term, delta, considers the possibility that disabling
collisions can generate new derelicts
    integrator = @ode15s); % MATLAB integrator used to run simulation
    % we typically use ode15s because often, models with many shells become
stiff diff eqs.
```

You can choose between a static exponential or JB2008-based atmospheric density models. The static exponential model is not time dependent, and will run faster than JB2008. The JB2008 model is time/space-weather dependent and more accurate, but runs slower. You can see spikes in the JB2008 graph as the density changes over time.

Static Exponential Model

JB2008 Model



Set atmospheric density model:

```
atm_density_choice = "Static Exponential"
```

```
atm_density_choice =
"Static Exponential"
```

```
switch atm_density_choice
    case "JB2008"
        scenario_properties.density_filepath =
fullfile(".", "Drag", "Atmosphere Model", "JB2008", "Precomputed",
"dens_highvar_2000.mat");
        scenario_properties.dens_model = @JB2008_dens_func;
        scenario_properties.time_dep_density = true;
    case "Static Exponential"
        scenario_properties.dens_model = @static_exp_dens_func;
        scenario_properties.time_dep_density = false;
end
```

Processes and saves the scenario properties you just set to be used in the simulation:

```
scen_properties = scen_properties_class(scenario_properties);
```

To access and edit ODE options, e.g.:

- `scen_properties.options.AbsTol = 1e-2;`
- `scen_properties.options.RelTol = 1e-2;`

```
% scen_properties.options
```

Define Species Properties

This is similiar to the MOCAT-3 model, but features an additional species S, which represents slotted satellites that are assumed to use orbits that are physically coordinated and thus subject to a further reduction in collision propbability. Define launch function:

- Null: Launches no new satellites ever. May be helpful in debugging.
- Constant: Launches a constant number of satellites, with that number set by species_properties.lambda_constant in the Su species properties below.

Choose your launch function to determine the rate of new active satellites (species Su are inserted into orbit)

```
launch_func_choice = "Constant"

launch_func_choice =
"Constant"

switch launch_func_choice
  case "Null"
    launch_func_sat = @launch_func_null;
  case "Constant"
    launch_func_sat = @launch_func_constant;
    lambda_constant = num2cell(500 * rand(scen_properties.N_shell,1));
end
```

Define drag function:

```
my_drag_func = @drag_func;
```

species_properties is a struct object that takes in your inputs for the properties of a species. Once all the properties of a certain species have been set, they are saved and processed for the model by the species function. Then we clear species_properties at the end of each section so we can use it again for the next species (and we dont get properties left over from the last species).

```
species_properties = struct;
```

List of Species Properties

- *sym_name* - Symbolic name for species
- *Cd* - Coefficient of drag, unitless
- *mass* - Mass of species [kg]
- *radius* - Radius of species [m]
- *A* - Area of species in [m^2]
- *amr* - Area/mass ratio in [m^2/kg]
- *beta* - Ballistic coefficient
- *slotted* - Whether species is slotted or not [bool]
- *drag_effected* - Whether object decreases altitude due to drag or not [bool]

Only for unslotted satellites (Su)

- *lambda_constant* - Determines launch rate if launch function is Constant
- *maneuverable* - Whether species is maneuverable or not [bool]
- *deltat* - Lifetime of spacecraft in years

- P_m - Post-mission disposal efficacy, if post-mission disposal is successful, the satellite is removed from the simulation. If it fails, the satellite transitions to a derelict [Float [0,1]]
- α - Efficacy of Collision Avoidance vs. inactive
- $\alpha_{active} = 0.01$; % Efficacy of Collision Avoidance vs. other active
- $\alpha = 2e-3$; % Efficacy of Collision Avoidance vs. inactive
- α_{active} - Efficacy of Collision Avoidance vs. other active
- $RBflag$ - No Rocket Bodies yet.

Only for slotted satellites (S)

- $slotting_effectiveness$ - proportion of conjunctions between pairs of slotted spacecraft that are mitigated by the slotting system

Define species properties for unslotted satellites

```
% Su
species_properties.sym_name = "Su";
species_properties.Cd = 2.2; % unitless
species_properties.mass = 223; % kg
species_properties.radius = 1.490/2; % m
species_properties.A = 1.741; % m^2
species_properties.amr = species_properties.A/species_properties.mass; %
m^2/kg
species_properties.amr = species_properties.A./species_properties.mass; %
m^2/kg
species_properties.beta = species_properties.Cd*species_properties.amr; %
ballistic coefficient

% Orbit Properties
species_properties.slotted = false; % bool

% Capabilities
species_properties.drag_effected = false; % bool, does object decrease
altitude due to drag
species_properties.active = true; % bool, use alpha_active for collisions
vs. trackable
species_properties.maneuverable = true; % bool, use alpha_active for
collisions vs. trackable
species_properties.trackable = true; % bool, others can avoid this with
alpha.
species_properties.deltat = 5.0; % lifetime of spacecraft in years
species_properties.Pm = .90; % Post-mission disposal efficacy Float [0,1]
species_properties.alpha = 2e-3; % Efficacy of Collision Avoidance vs.
inactive
species_properties.alpha_active = 1e-3; % Efficacy of Collision Avoidance
vs. other active
species_properties.RBflag = 0; % No Rocket Bodies yet.
if launch_func_choice == "Constant"
    species_properties.lambda_constant = lambda_constant;
```

```
end
```

Create species instance with a launch function, post-mission disposable function, drag function, species properties which are specific to satellites, and scenario properties which are specific to the simulation. If post-mission disposable is successful (based on probability Pm), the satellite is removed from the simulation. If it fails, the satellite transitions to a derelict.

```
Su_species = species(launch_func_sat, @pmd_func_sat, my_drag_func,  
species_properties, scen_properties);  
species_properties = struct;
```

Define species properties for slotted satellites

```
% S  
species_properties.sym_name = "S";  
species_properties.Cd = 2.2; % unitless  
species_properties.mass = 223; % kg  
species_properties.radius = 1.490/2; % m  
species_properties.A = 1.741; % m^2  
species_properties.amr = species_properties.A/species_properties.mass; %  
m^2/kg  
species_properties.amr = species_properties.A./species_properties.mass; %  
m^2/kg  
species_properties.beta = species_properties.Cd*species_properties.amr; %  
ballistic coefficient  
  
% Orbit Properties  
species_properties.slotted = true; % bool  
species_properties.slotting_effectiveness = 1.0; % Float [0,1],  
  
% Capabilities  
species_properties.drag_effected = false; % bool, does object decrease  
altitude due to drag  
species_properties.active = true; % bool, use alpha_active for collisions  
vs. trackable  
species_properties.maneuverable = true; % bool, use alpha_active for  
collisions vs. trackable  
species_properties.trackable = true; % bool, others can avoid this with  
alpha.  
species_properties.deltat = 5.0; % lifetime of spacecraft in years  
species_properties.Pm = .90; % Post-mission disposal efficacy Float [0,1]  
species_properties.alpha = 2e-3; % Efficacy of Collision Avoidance vs.  
inactive  
species_properties.alpha_active = 1e-3; % Efficacy of Collision Avoidance  
vs. other active  
species_properties.RBflag = 0; % No Rocket Bodies yet.  
if launch_func_choice == "Constant"  
    species_properties.lambda_constant = lambda_constant;  
end
```

```
S_species = species(launch_func_sat, @pmd_func_sat, my_drag_func,
species_properties, scen_properties);
species_properties = struct;
```

Define species properties for derelict

```
% D
species_properties.sym_name = "D";
species_properties.Cd = Su_species.species_properties.Cd;
species_properties.mass = Su_species.species_properties.mass;
species_properties.radius = Su_species.species_properties.radius;
species_properties.A = Su_species.species_properties.A;
species_properties.amr = Su_species.species_properties.A/
Su_species.species_properties.mass(1); % m^2/kg
species_properties.beta =
Su_species.species_properties.Cd*Su_species.species_properties.amr; %
ballistic coefficient
species_properties.drag_effected = true; % bool, does object decrease
altitude due to drag
species_properties.maneuverable = false; % bool, use alpha_active for
collisions vs. trackable
species_properties.active = false;

D_species = species(@launch_func_null, @pmd_func_derelict, my_drag_func,
species_properties, scen_properties);
D_species.species_properties.pmd_linked_species = [S_species, Su_species];
species_properties = struct;
```

Define species properties for debris.

```
% N
species_properties.sym_name = "N";
species_properties.Cd = 2.2; % unitless
species_properties.mass = 0.640;
species_properties.radius = 0.180/2; % m
species_properties.A = 0.020;
species_properties.amr = species_properties.A/species_properties.mass; %
m^2/kg
species_properties.beta = species_properties.Cd*species_properties.amr; %
ballistic coefficient
species_properties.drag_effected = true; % bool, does object decrease
altitude due to drag
species_properties.maneuverable = false; % bool, use alpha_active for
collisions vs. trackable
species_properties.active = false;

N_species = species(@launch_func_null, @pmd_func_none, my_drag_func,
species_properties, scen_properties);
species_properties = struct;
```

Add all species to scenario properties.

```
scen_properties.species = [Su_species, S_species, D_species, N_species];
scen_properties.species_cell = struct('Su', Su_species, 'S', S_species, 'D',
D_species, 'N', N_species);
```

Collision Modeling

The `make_collision_pairs_MOCAT4.m` function calculates collision coefficients for the table above, based on the species properties and NASA's standard break-up model, and adds the collision terms to the source sink equations. The `make_collision_pairs_MOCAT4.m` function is specific to MOCAT-4. The generalized function for collision modelling for other MOCAT versions is `make_collision_pairs_SBM.m`.

```
scen_properties = make_collision_pairs_MOCAT4(scen_properties, Su_species,
S_species, D_species, N_species);
```

Done

Define Initial Population

This section creates an initial population for satellites, derelict and debris. The `MC2SSEM_population` function takes a list of space objects from `initialized.mat` and categorizes them into satellites, derelict and debris. `x0` is a total count of each species in each shell that we will start the simulation from.

Species order and definition (NOTE: do not change the order):

- S = Active slotted satellites
- D = Derelict objects
- N = Debris objects
- Su = Active unslotted satellites
- B = Rocket bodies
- U = Lethal non-trackable debris objects (AKA N with trackable = False)

```
scen_properties.species_types = [isfield(scen_properties.species_cell,'S')
&& ~isempty(scen_properties.species_cell.S), ...
isfield(scen_properties.species_cell,'D')
&& ~isempty(scen_properties.species_cell.D), ...
isfield(scen_properties.species_cell,'N')
&& ~isempty(scen_properties.species_cell.N), ...
isfield(scen_properties.species_cell,'Su')
&& ~isempty(scen_properties.species_cell.Su), ...
isfield(scen_properties.species_cell,'B')
&& ~isempty(scen_properties.species_cell.B), ...
isfield(scen_properties.species_cell,'U')
&& ~isempty(scen_properties.species_cell.U)];
load('initialized.mat', 'sats');
x0 = MC2SSEM_population(sats,scen_properties);

% Display starting population
```

```

for spec_index = 1:length(scen_properties.species)
    spec_names(spec_index) =
scen_properties.species(spec_index).species_properties.sym_name;
end

T = array2table(x0, 'VariableNames', spec_names);
disp(array2table([[1:scen_properties.N_shell]',x0], 'VariableNames',
['Shell', spec_names]))

```

Shell	Su	S	D	N
1	0	0	8	23
2	7	0	3	15
3	14	24	10	32
4	42	2	31	91
5	141	32	114	186
6	446	10	91	227
7	643	2334	117	194
8	278	101	98	309
9	157	0	129	340
10	75	4	120	519
11	44	15	150	829
12	48	96	215	1224
13	42	1	144	1381
14	17	0	80	1160
15	12	3	116	695
16	9	1	288	488
17	1	0	56	353
18	43	0	73	256
19	2	1	35	207
20	3	98	39	145
21	5	272	11	104
22	0	0	9	244
23	2	0	18	86
24	0	0	40	106
25	5	32	275	107
26	50	0	227	174
27	3	1	56	192
28	0	0	20	144
29	0	0	18	93
30	0	0	18	58
31	0	1	6	20
32	0	0	3	26
33	0	0	3	23
34	0	0	5	19
35	0	0	3	14
36	0	0	0	0

Build Model

`simulation_class` is responsible for building and running the model. The `build_model()` function compiles the equations. It adds together all of the terms (regarding launch, collisions, post-mission disposal, and atmospheric drag).

Building the model is only required once in order to run the model if the user doesn't want to change any input parameters.

```

x0 = reshape(table2array(T), [], 1);% Pad initial conditions with the
indicator variables
my_sim = simulation_class(scen_properties.species, scen_properties);
my_sim.build_model();

Building model...
ans =
simulation_class with properties:

    species_list: [1×4 species]
    scen_properties: [1×1 scen_properties_class]
        equations: [36×4 sym]
        results: []
    xdot_eqs: []
    xdot_fun: []
    var_col: []
    drag_term_upper: []
    drag_term_cur: []

Now processing Su
Now processing S
Now processing D
Now processing N
Done building model.

```

Run Simulation

`run_model()` is where the simulation model is actually run. It takes a list of initial conditions (`x0`).

The function also accepts many optional parameters:

- `progressBar` - Displays percentage of simulation run in another window - bool
- `disp_times` - Displays time steps in console - bool
- `start_date` - Start date of simulation - MATLAB datetime object
- `simulation_duration` - Years of simulation to run - double
- `N_step` - Number of simulations steps to run - int

```

x0 = reshape(x0, [], 1);
my_sim.run_model(x0, 'progressBar', true, 'disp_times', false)

```

Plots

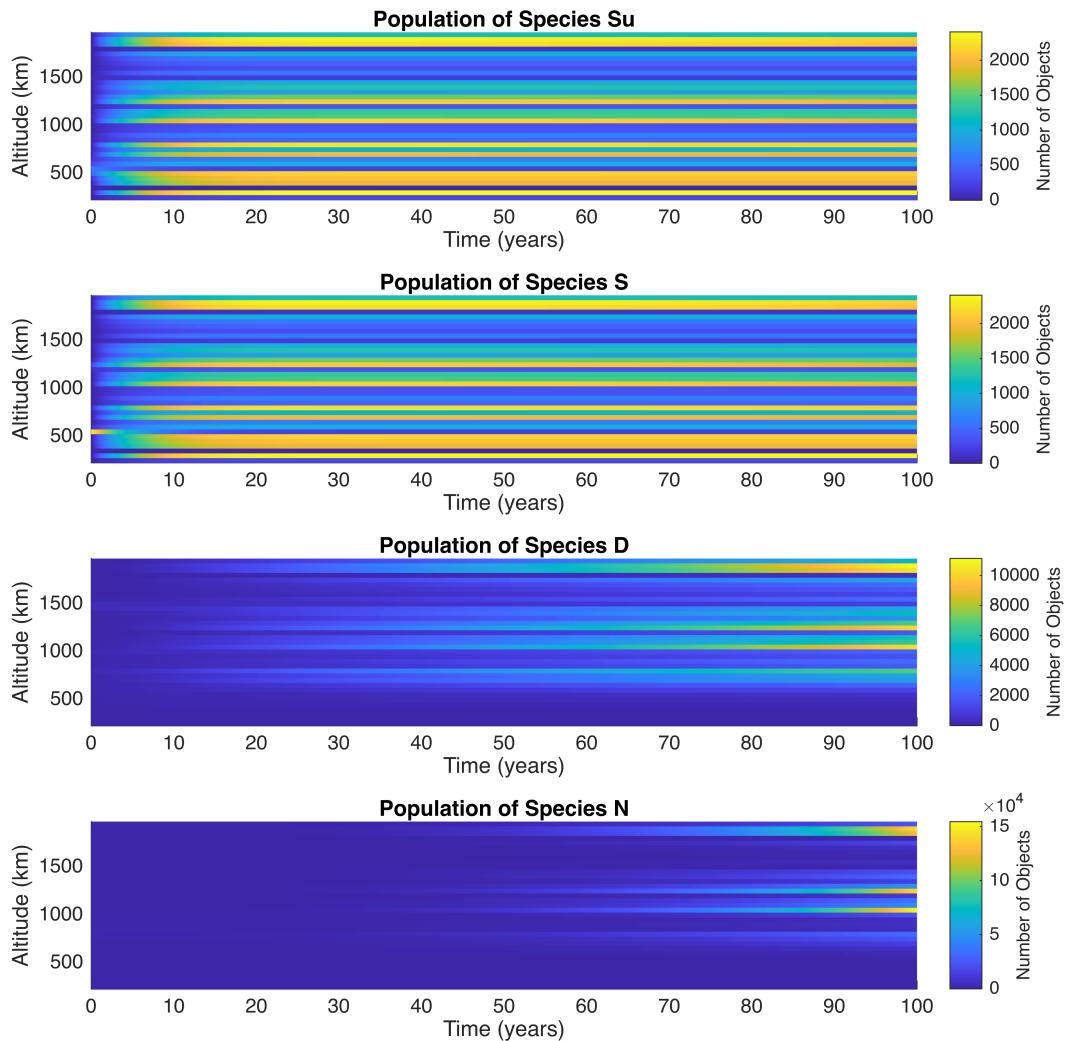
Produces a Figure with subplots for each species population evolution over time

```

my_sim.total_species_evol_vis();

```

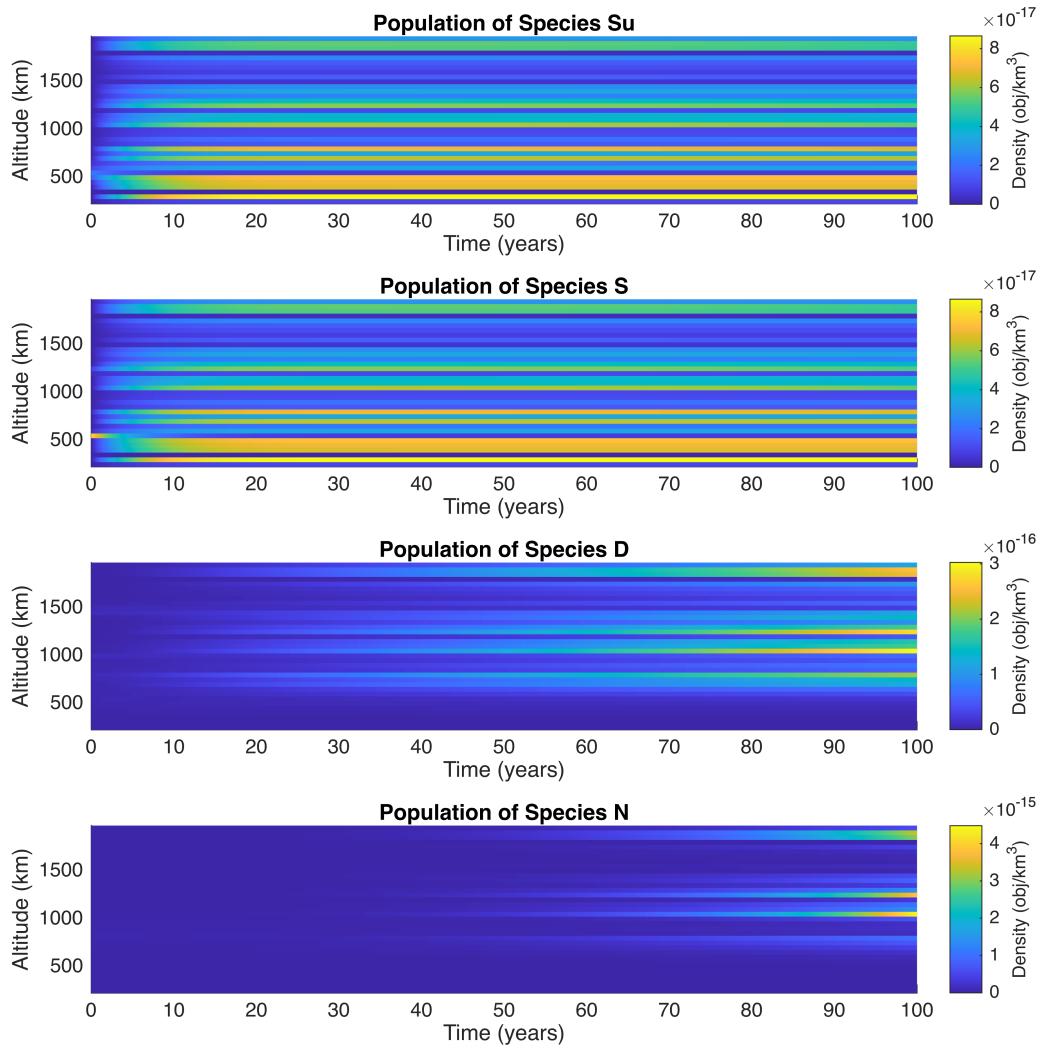
Producing visuals for the evolution of each species.



Produces a Figure with subplots for each species population density evolution over time

```
my_sim.density_species_evol_vis();
```

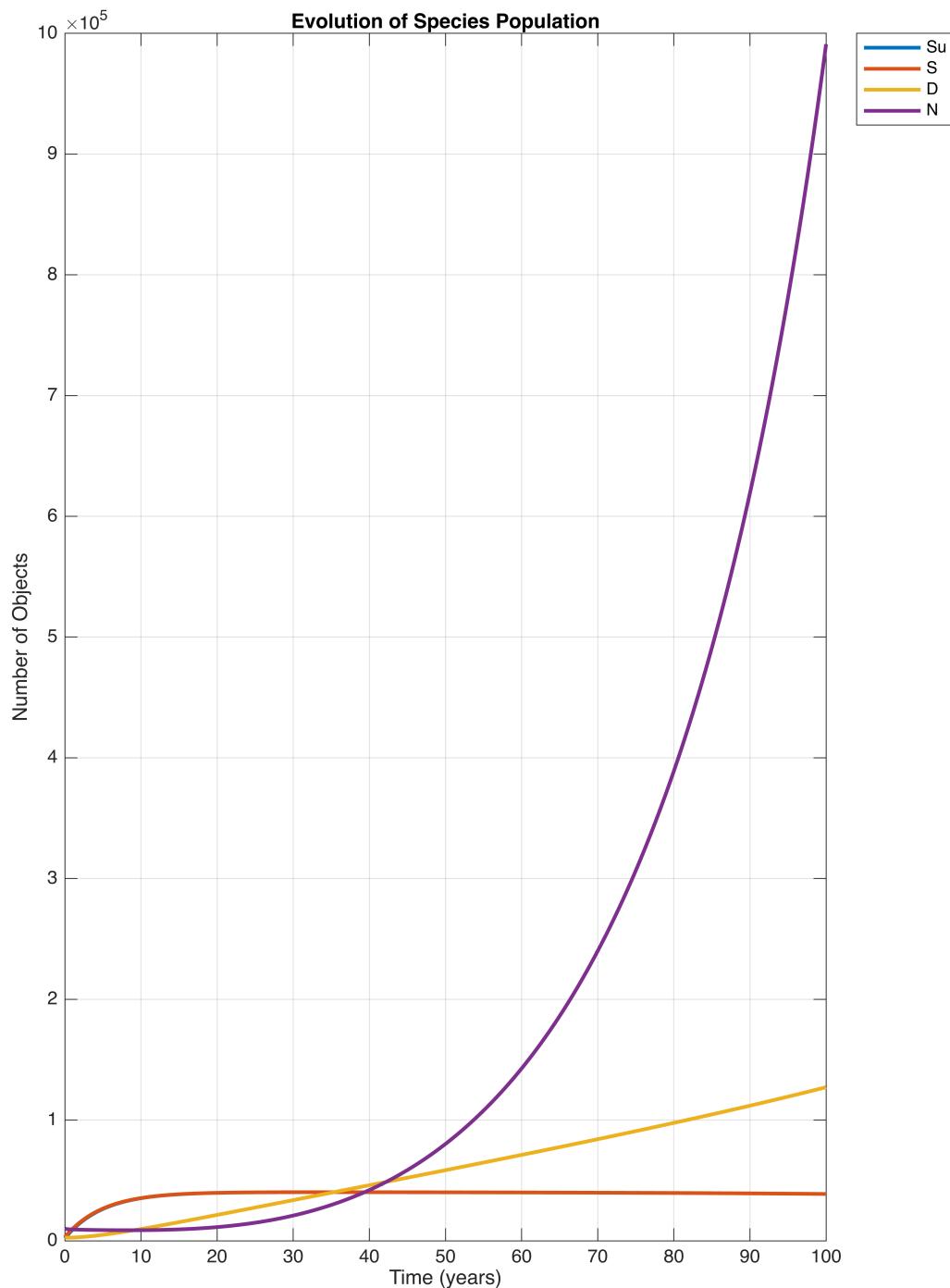
Producing visuals for the evolution of each species' density.



Produces a Figure for the evolution of the total population of each species over time across all shells

```
my_sim.pop_vs_time_vis();
```

Producing visual of the change in populations across all altitudes vs time.



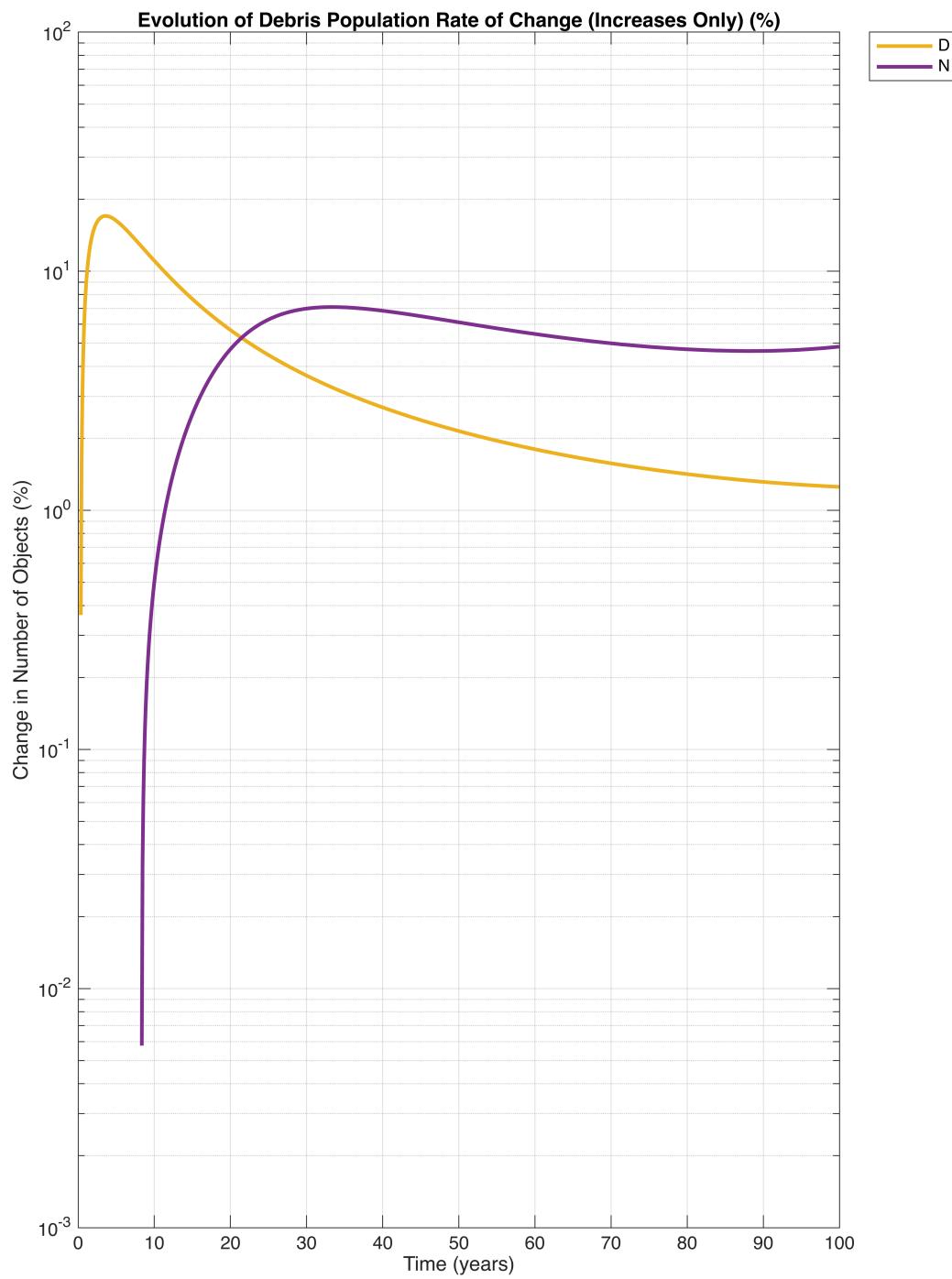
Graphs the rate of change for the total inactive population added across all altitudes at each time.

percentage = (bool, defaults to false) that expresses rate of change in percentage terms.

```
my_sim.total_deb_pop_time_deriv('percentage', true)
```

Producing visual of the change in inactive populations across all altitudes vs time.

D
N



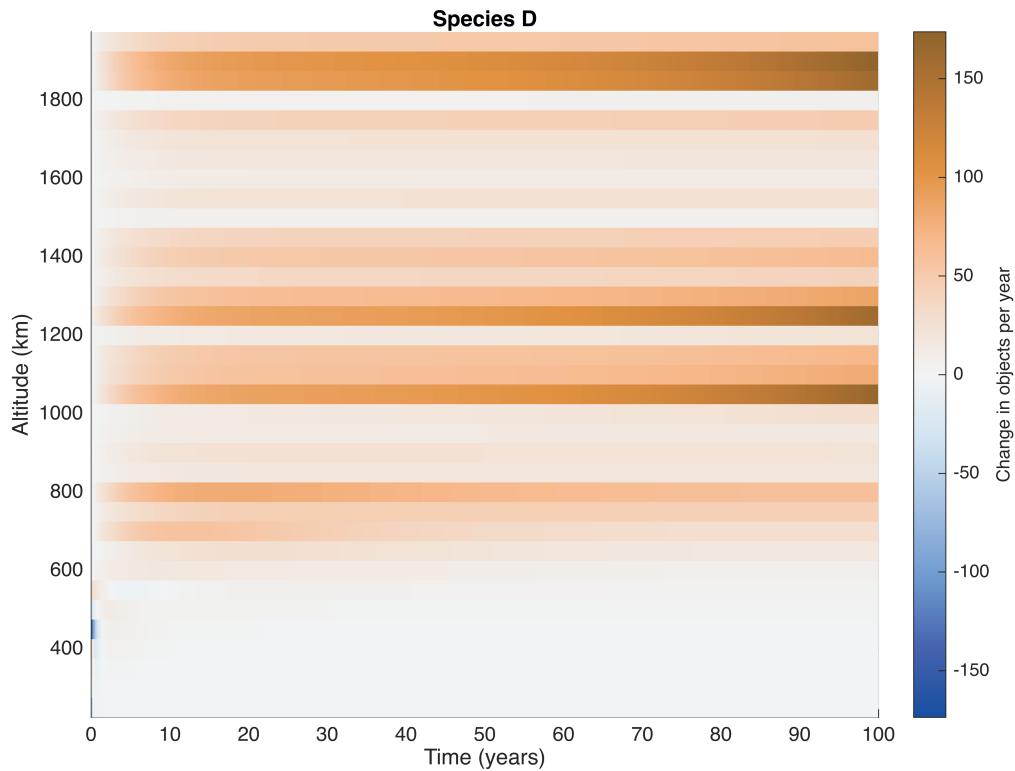
Produces a Figure with subplots for numerical derivative of each inactive species population evolution over time constraint is the value used in the color scale.

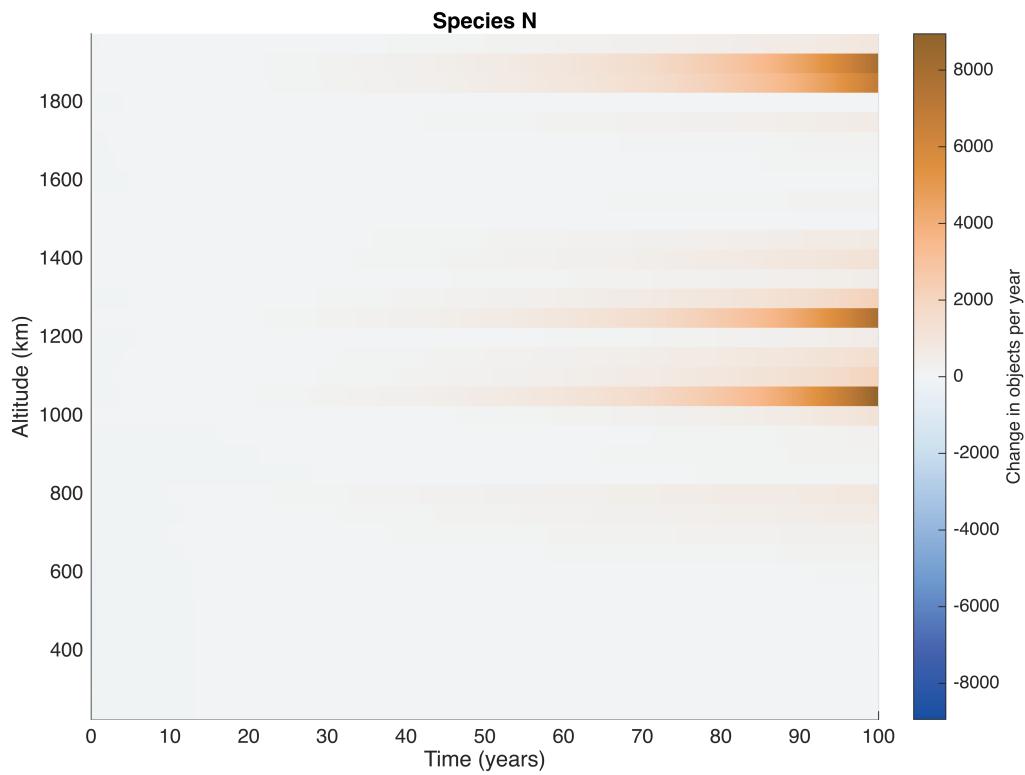
percentage is bool (default false) for whether to use percentage or absolute.

color_scale_crop_percentiles is pair (default [0, 100]). Color scale will be set based on the inclusive range from the lower value to the higher value.

```
my_sim.total_deb_species_deriv_evolv_vis();
```

Producing visuals for the rate change of each debris species.

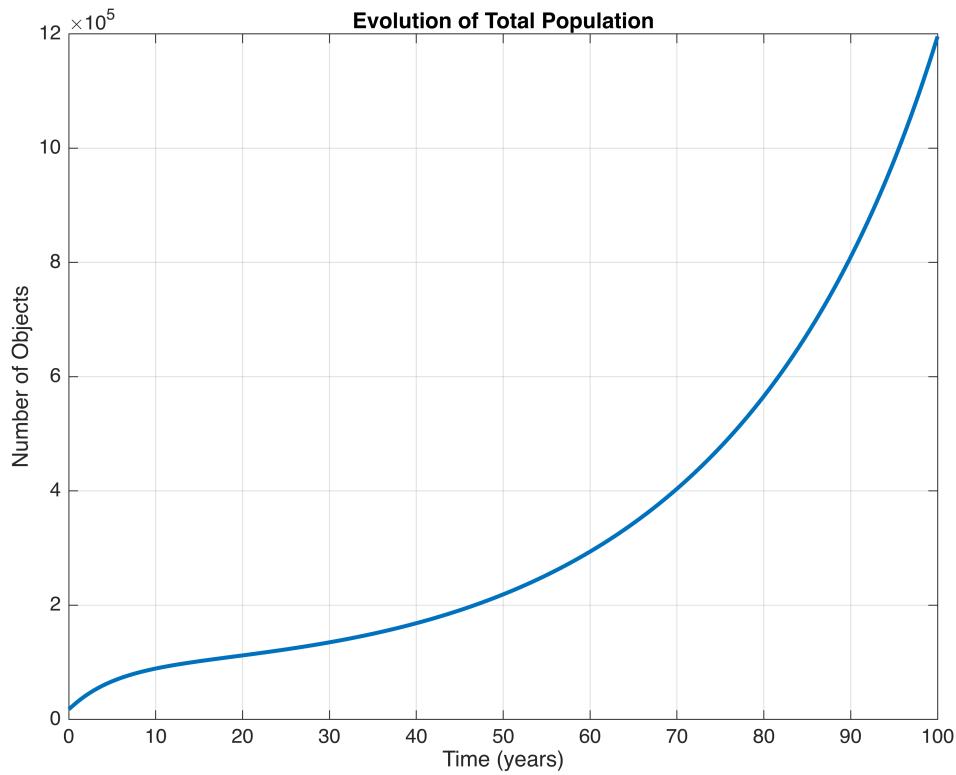




Produces a Figure for the evolution of the total population of all objects over time across all shells

```
my_sim.tot_vs_time_vis();
```

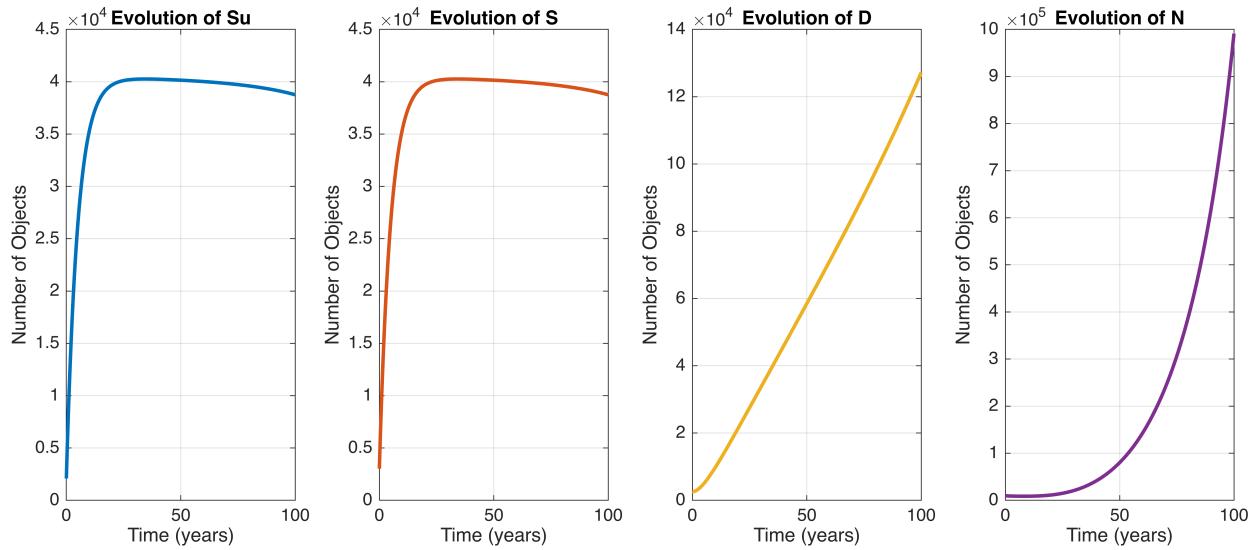
Producing visual of the change in the total population across all altitudes vs time.



Produces a Figure per species for the evolution of the total population of each species over time across all shells

```
my_sim.species_time_vis();
```

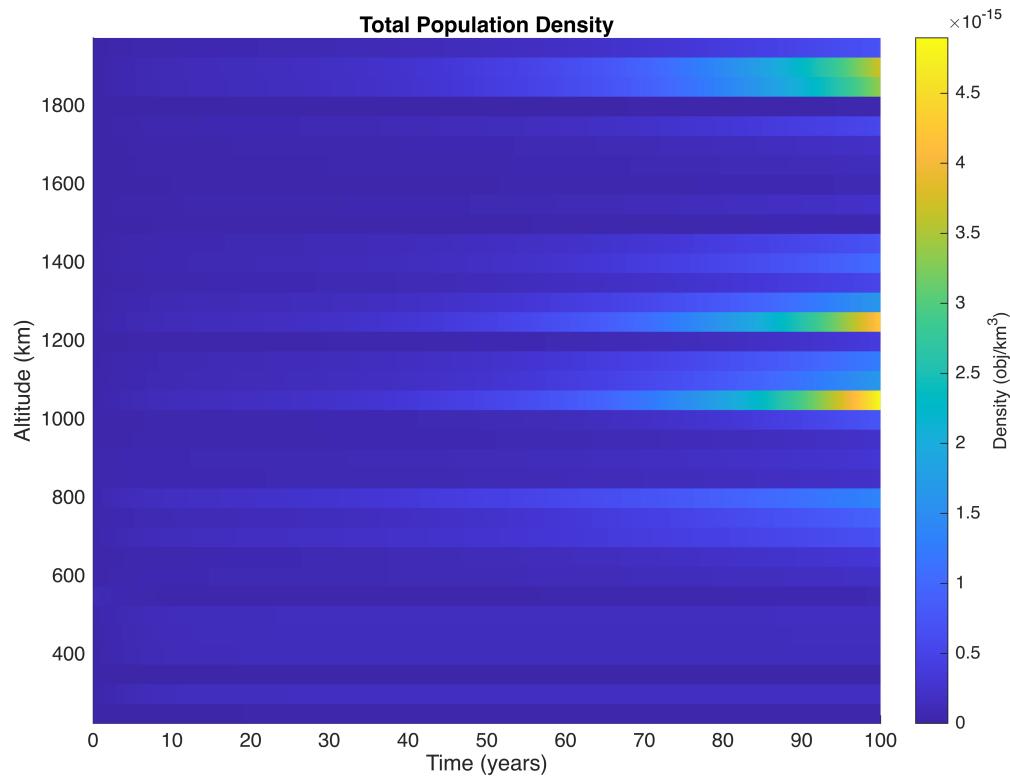
Producing visuals of the change in each species across all altitudes vs time.



Displays total spatial density in each shell over time

```
my_sim.density_vs_time();
```

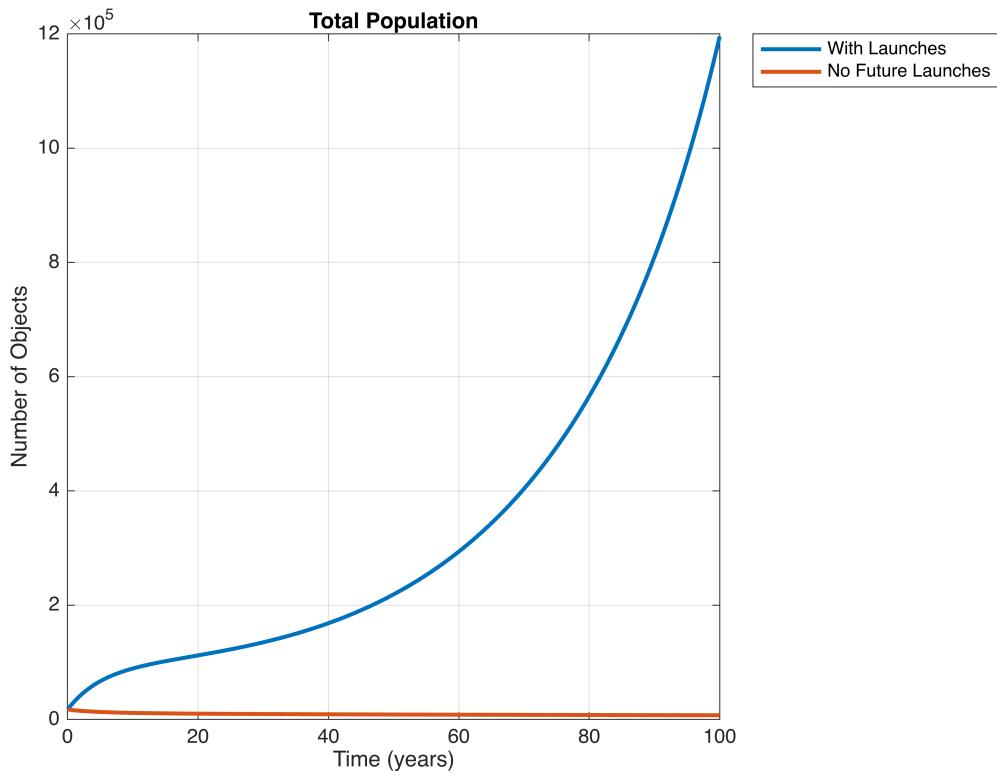
Producing visuals for the evolution of total population density.



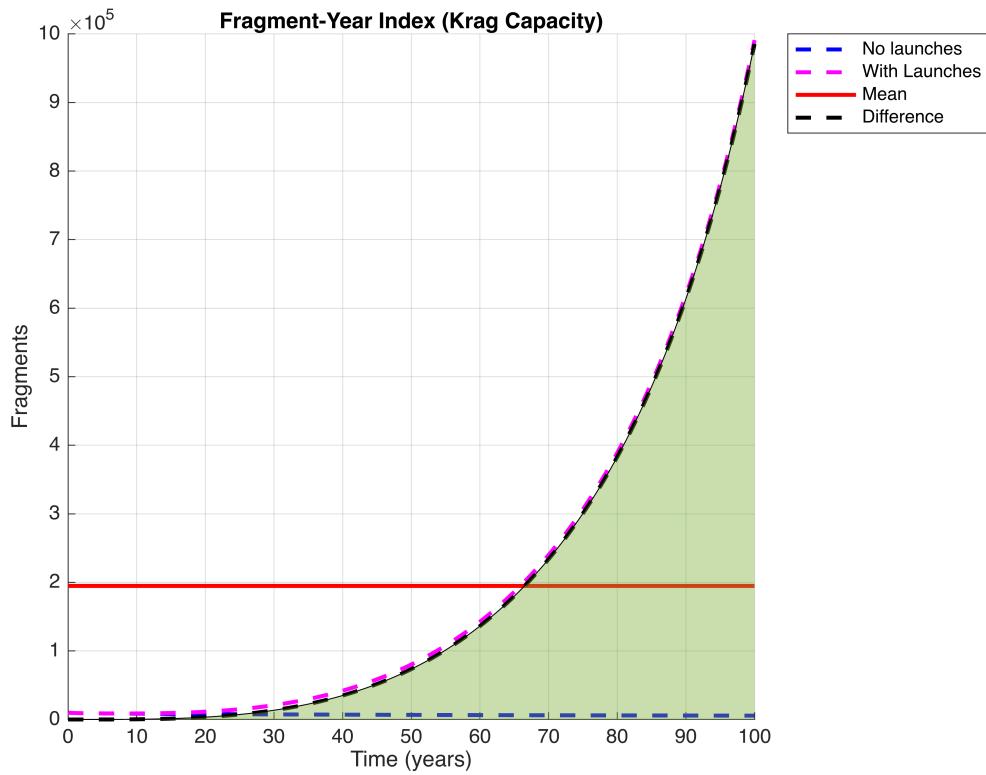
Computes and displays Number_Time_Product

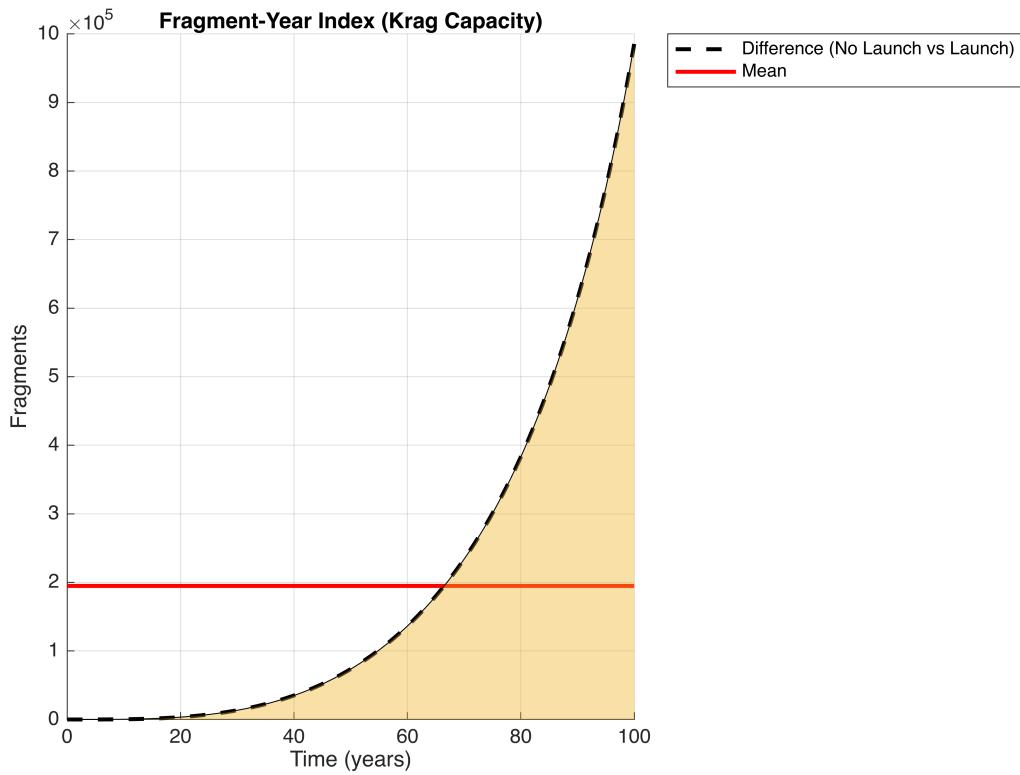
```
my_sim.num_time_prod(x0);
```

Warning: Negative data ignored



Producing visuals for Number-Time Product metric for Capacity.

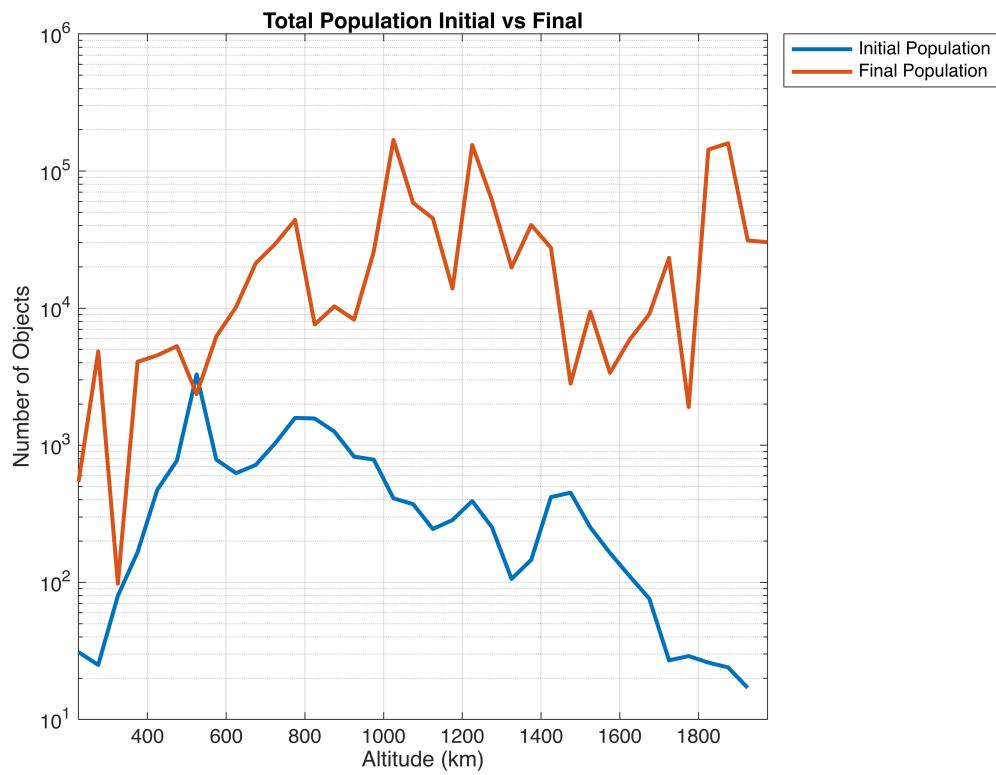




Produces a Figure comparing the total initial number of objects and the total final number of objects

```
my_sim.init_vs_final_vis();
```

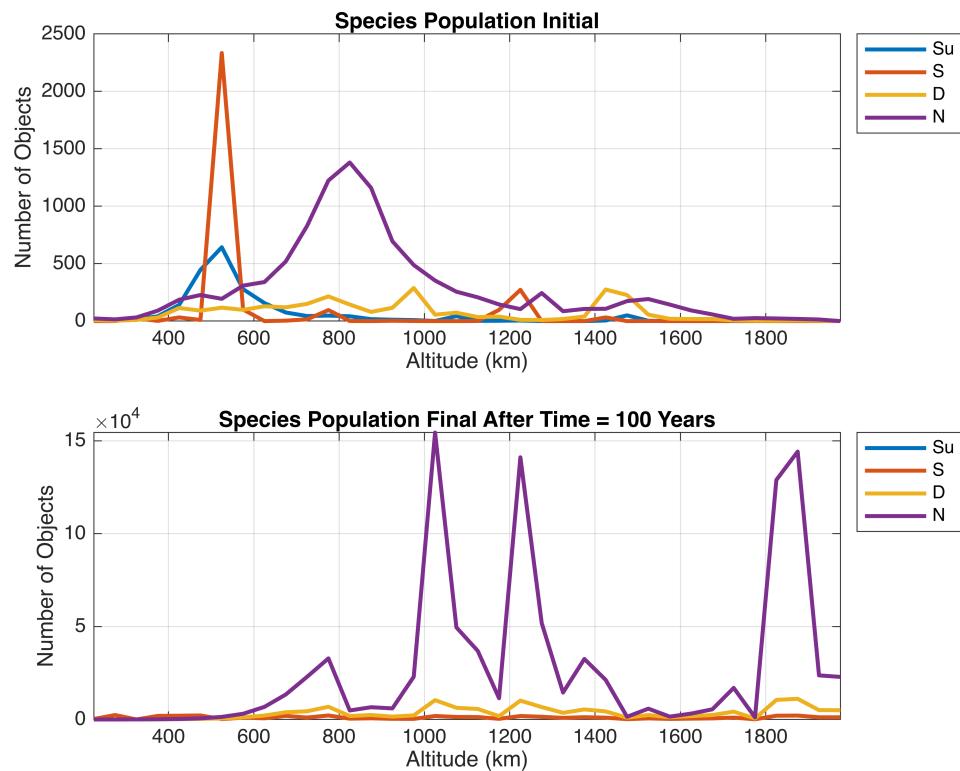
Producing visual for the final vs initial populations.



Produces a Figure comparing the initial population of each species alongside a plot of the final population of each species

```
my_sim.init_and_final_vis();
```

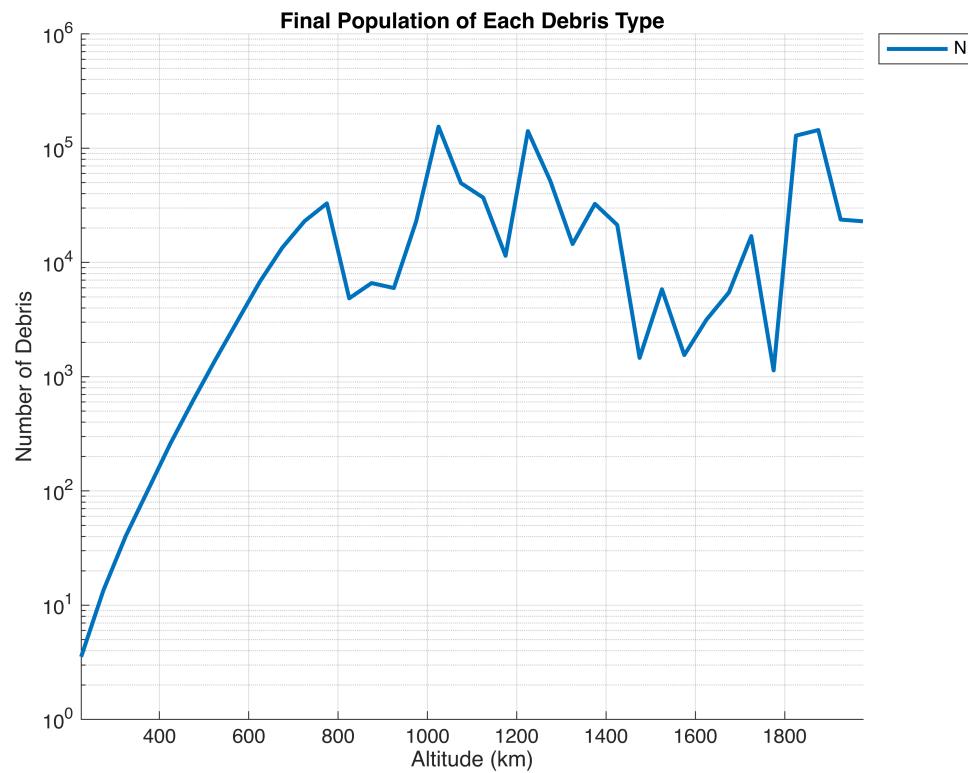
Producing visual for the initial and final species populations.



Produces a Figure showing number of debris across altitudes by debris type

```
my_sim.final_debris_vis();
```

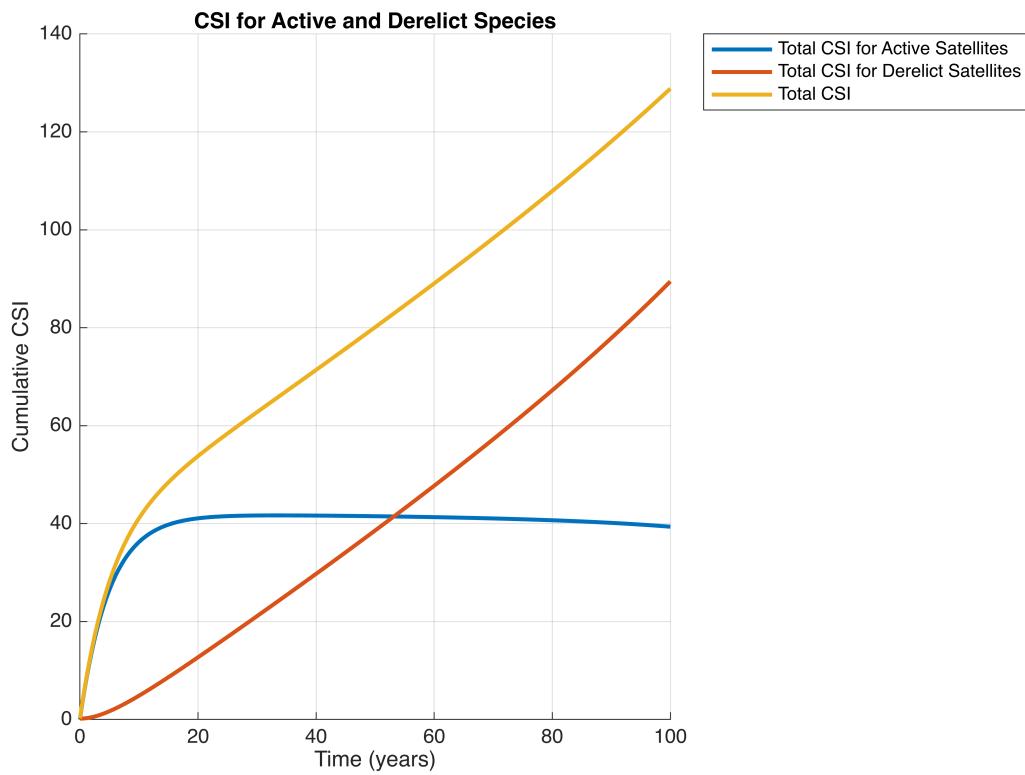
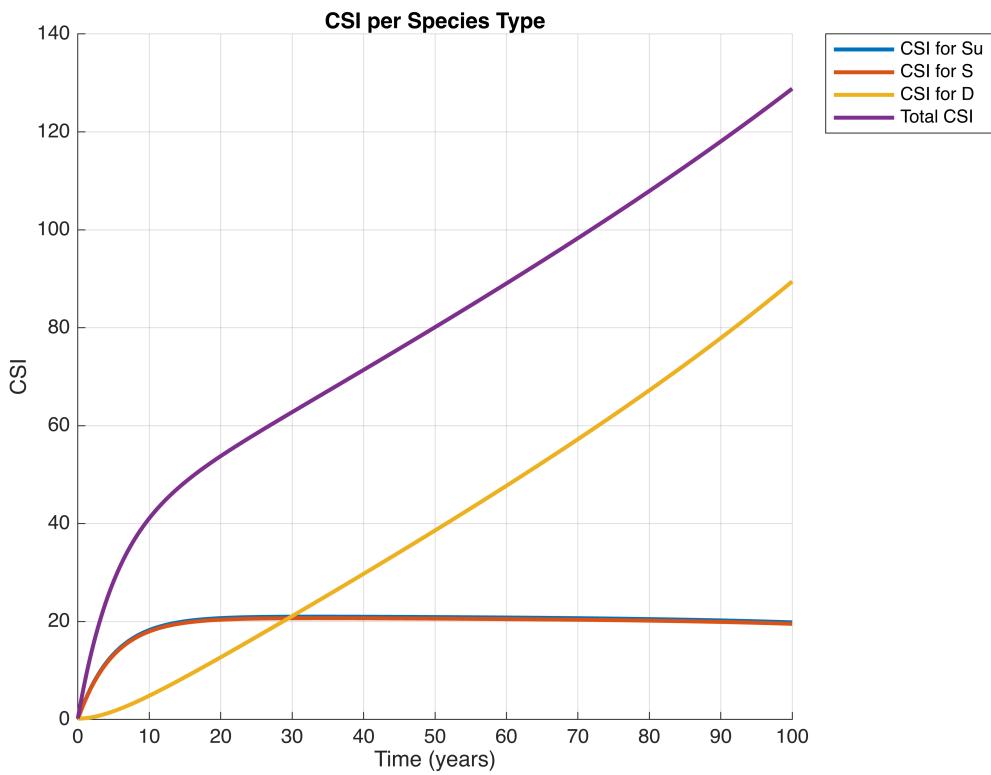
Producing visual for the final debris populations per type of debris.



Computes and displays cumulative Criticality of Space Index - CSI

```
my_sim.cum_CSI()
```

Producing two visuals of CSI.



Warning: Negative data ignored

Close all figures created (or run this line in your command line)

```
% close all
```