Armageddon

Release 0.0.1

Group Dimorphos

Nov 24, 2022

CONTENTS:

CHAPTER

ONE

SYNOPSIS:

Asteroids entering Earth's atmosphere are subject to extreme drag forces that decelerate, heat and disrupt the space rocks. The fate of an asteroid is a complex function of its initial mass, speed, trajectory angle and internal strength.

Asteroids 10-100 m in diameter can penetrate deep into Earth's atmosphere and disrupt catastrophically, generating an atmospheric disturbance (airburst) that can cause damage on the ground. Such an event occurred over the city of Chelyabinsk in Russia, in 2013, releasing energy equivalent to about 520 kilotons of TNT (1 kt TNT is equivalent to 4.184×10^{12} J), and injuring thousands of people (Popova et al., 2013; Brown et al., 2013). An even larger event occurred over Tunguska, a relatively unpopulated area in Siberia, in 1908.

This simulator predicts the fate of asteroids entering Earth's atmosphere, and provides a hazard mapper for an impact over the UK.

CHAPTER

TWO

PROBLEM DEFINITION

2.1 Equations of motion for a rigid asteroid

The dynamics of an asteroid in Earth's atmosphere prior to break-up is governed by a coupled set of ordinary differential equations:

<<<<<<< HEAD

begin{aligned} frac{dv}{dt} & = frac{-C_Drho_a A v^2}{2 m} + g sin theta \ frac{dm}{dt} & = frac{-C_Hrho_a A v^3}{2 Q} \ frac{dtheta}{dt} & = frac{gcostheta}{v} - frac{C_Lrho_a A v}{2 m} - frac{vcostheta}{R_P + z} \ frac{dz}{dt} & = -vsintheta \ frac{dx}{dt} & = frac{vcostheta}{1 + z/R_P} end{aligned} end{math}

In these equations, v, m, and A are the asteroid speed (along trajectory), mass and cross-sectional area, respectively. We will assume an initially **spherical asteroid** to convert from inital radius to mass (and cross-sectional area). θ is the meteoroid trajectory angle to the horizontal (in radians), x is the downrange distance of the meteoroid from its entry position, z is the altitude and t is time; C_D is the drag coefficient, ρ_a is the atmospheric density (a function of altitude), C_H is an ablation efficiency coefficient, Q is the specific heat of ablation; C_L is a lift coefficient; and R_P is the planetary radius. All terms use MKS units.

2.2 Asteroid break-up and deformation

A commonly used criterion for the break-up of an asteroid in the atmosphere is when the ram pressure of the air interacting with the asteroid $\rho_a v^2$ first exceeds the strength of the asteroid Y.

$$\rho_a v^2 = Y$$

Should break-up occur, the asteroid deforms and spreads laterally as it continues its passage through the atmosphere. Several models for the spreading rate have been proposed. In the simplest model, the fragmented asteroid's spreading rate is related to its along trajectory speed (Hills and Goda, 1993):

$$\frac{dr}{dt} = \left[\frac{7}{2}\alpha\frac{\rho_a}{\rho_m}\right]^{1/2} v$$

Where r is the asteroid radius, ρ_m is the asteroid density (assumed constant) and α is a spreading coefficient, often taken to be 0.3. It is conventional to define the cross-sectional area of the expanding cloud of fragments as $A = \pi r^2$ (i.e., assuming a circular cross-section), for use in the above equations. Fragmentation and spreading **ceases** when the ram pressure drops back below the strength of the meteoroid $\rho_a v^2 < Y$.

2.3 Airblast damage

The rapid deposition of energy in the atmosphere is analogous to an explosion and so the environmental consequences of the airburst can be estimated using empirical data from atmospheric explosion experiments (Glasstone and Dolan, 1977).

The main cause of damage close to the impact site is a strong (pressure) blastwave in the air, known as the **airblast**. Empirical data suggest that the pressure in this wave p (in Pa) (above ambient, also known as overpressure), as a function of explosion energy E_k (in kilotons of TNT equivalent), burst altitude z_b (in m) and horizontal range r (in m), is given by:

$$p(r) = 3.14 \times 10^{11} \left(\frac{r^2 + z_b^2}{E_k^{2/3}} \right)^{-1.3} + 1.8 \times 10^7 \left(\frac{r^2 + z_b^2}{E_k^{2/3}} \right)^{-0.565}$$

For airbursts, we will take the total kinetic energy lost by the asteroid at the burst altitude as the burst energy E_k. For cratering events, we will define E_k as the **larger** of the total kinetic energy lost by the asteroid at the burst altitude or the residual kinetic energy of the asteroid when it hits the ground.

The following threshold pressures can then be used to define different degrees of damage.

Damage Level	Description	Pressure (kPa)
1	~10% glass windows shatter	1.0
2	~90% glass windows shatter	3.5
3	Wood frame buildings collapse	27
4	Multistory brick buildings collapse	43

Table 1: Pressure thresholds (in kPa) for airblast damage

2.4 Additional sections

You should expand this documentation to include explanatory text for all components of your tool.

CHAPTER

THREE

FUNCTION API

Module dealing with postcode information.

class locator.**PostcodeLocator**(*postcode_file='/home/runner/work/acs-armageddon-Dimorphos/acs-armageddon-Dimorphos/armageddon/../resources/full_postcodes.csv'*, *census_file='/home/runner/work/acs-armageddon-Dimorphos/acs-armageddon-Dimorphos/armageddon/../resources/population_by_postcode_sector.csv'*, *norm=<function great_circle_distance>*)

Class to interact with a postcode database file.

Parameters

- **postcode_file** (*str, optional*) – Filename of a .csv file containing geographic location data for postcodes.
- **census_file** (*str, optional*) – Filename of a .csv file containing census data by postcode sector.
- **norm** (*function*) – Python function defining the distance between points in latitude-longitude space.

get_population_of_postcode(*postcodes*, *sector=False*)

Return populations of a list of postcode units or sectors. :param postcodes: list of postcode units or postcode sectors :type postcodes: list of lists :param sector: if true return populations for postcode sectors,

otherwise returns populations for postcode units

Returns
Contains the populations of input postcode units or sectors

Return type
list of lists

Examples

```
>>> locator = PostcodeLocator('resources/full_postcodes.csv', 'resources/
↪population_by_postcode_sector.csv')
>>> pop1 = locator.get_population_of_postcode([['SW7 2AZ', 'SW7 2BT', 'SW7 2BU',
↪ 'SW7 2DD']])
>>> pop1
[[19, 19, 19, 19]]
>>> pop2 = locator.get_population_of_postcode([['SW7  2']], True)
```

(continues on next page)

(continued from previous page)

```
>>> pop2
[[2283]]
```

get_postcodes_by_radius(*X*, *radii*, *sector=False*)

Return (unit or sector) postcodes within specific distances of input location. :param X: Latitude-longitude pair of centre location :type X: arraylike :param radii: array of radial distances from X :type radii: arraylike :param sector: if true return postcode sectors, otherwise postcode units :type sector: bool, optional

Returns
Contains the lists of postcodes closer than the elements of radii to the location X.

Return type
list of lists

Examples

```
>>> locator = PostcodeLocator('resources/full_postcodes.csv', 'resources/
↪population_by_postcode_sector.csv')
>>> postcodes = locator.get_postcodes_by_radius((51.4981, -0.1773), [0.13e3])
>>> postcode_dictionaries = [dict.fromkeys(postcodes[i], "risk") for i in␣
↪range(len(postcodes))]
>>> ans1 = [{'SW7 5HG': 'risk','SW7 2BU': 'risk','SW7 5HQ': 'risk',␣
↪       'SW7 2BT': 'risk','SW7 5HF': 'risk','SW7 2DD': 'risk',␣
↪   'SW7 2AZ': 'risk'}]
>>> postcode_dictionaries == ans1
True
>>> postcodes = locator.get_postcodes_by_radius((51.4981, -0.1773), [0.4e3, 0.
↪2e3], True)
>>> postcode_dictionaries = [dict.fromkeys(postcodes[i], "risk") for i in␣
↪range(len(postcodes))]
>>> ans2 = [{'SW7 4': 'risk','SW7 5': 'risk','SW7 3': 'risk',␣
↪ 'SW7 1': 'risk','SW7 9': 'risk', 'SW7 2': 'risk'},                    {'SW7 4
↪': 'risk','SW7 5': 'risk','SW7 3': 'risk',                    'SW7 1': 'risk',
↪'SW7 9': 'risk','SW7 2': 'risk'}]
>>> postcode_dictionaries == ans2
True
```

locator.**great_circle_distance**(*latlon1*, *latlon2*)

Calculate the great circle distance (in metres) between pairs of points specified as latitude and longitude on a spherical Earth (with radius 6371 km). :param latlon1: latitudes and longitudes of first point (as [n, 2] array for n points) :type latlon1: arraylike :param latlon2: latitudes and longitudes of second point (as [m, 2] array for m points) :type latlon2: arraylike

Returns
Distance in metres between each pair of points (as an n x m array)

Return type
numpy.ndarray

Examples

```
>>> import numpy
>>> fmt = lambda x: numpy.format_float_scientific(x, precision=3)
>>> with numpy.printoptions(formatter={'all': fmt}):    print(great_circle_
↪distance([[54.0, 0.0], [55, 0.0]], [55, 1.0]))
[[1.286e+05]
 [6.378e+04]]
```

class solver.**Planet**(*atmos_func='exponential'*, *atmos_filename='/home/runner/work/acs-armageddon-Dimorphos/acs-armageddon-Dimorphos/armageddon/../resources/AltitudeDensityTable.csv'*, *Cd=1.0*, *Ch=0.1*, *Q=10000000.0*, *Cl=0.001*, *alpha=0.3*, *Rp=6371000.0*, *g=9.81*, *H=8000.0*, *rho0=1.2*)

The class called Planet is initialised with constants appropriate for the given target planet, including the atmospheric density profile and other constants

Set up the initial parameters and constants for the target planet

Parameters

- **atmos_func** (*string, optional*) – Function which computes atmospheric density, rho, at altitude, z. Default is the exponential function rho = rho0 exp(-z/H). Options are 'exponential', 'tabular' and 'constant'
- **atmos_filename** (*string, optional*) – Name of the filename to use with the tabular atmos_func option
- **Cd** (*float, optional*) – The drag coefficient
- **Ch** (*float, optional*) – The heat transfer coefficient
- **Q** (*float, optional*) – The heat of ablation (J/kg)
- **Cl** (*float, optional*) – Lift coefficient
- **alpha** (*float, optional*) – Dispersion coefficient
- **Rp** (*float, optional*) – Planet radius (m)
- **rho0** (*float, optional*) – Air density at zero altitude (kg/m^3)
- **g** (*float, optional*) – Surface gravity (m/s^2)
- **H** (*float, optional*) – Atmospheric scale height (m)

RK4_helper(*timestep*)

Helper function for RK4 method

Parameters
timestep (*float*) – The stepsize of iteration

Returns
change – A numpy array containing the change of each variable. Includes the following variables: 'angle', 'radius', 'altitude', 'velocity', 'mass', 'distance'

Return type
ndarray

analyse_outcome(*result*)

Inspect a pre-found solution to calculate the impact and airburst stats

Parameters
result (*DataFrame*) – pandas dataframe with velocity, mass, angle, altitude, horizontal distance, radius and dedz as a function of time

Returns

outcome – dictionary with details of the impact event, which should contain the key:

`outcome` (which should contain one of the following strings: `Airburst` or `Cratering`),

as well as the following 4 keys:
`burst_peak_dedz`, `burst_altitude`, `burst_distance`, `burst_energy`

Return type
Dict

`calculate_energy`(*result*)

Function to calculate the kinetic energy lost per unit altitude in kilotons TNT per km, for a given solution.

Parameters

- **result** (*DataFrame*) – A pandas dataframe with columns for the velocity, mass, angle, altitude, horizontal distance and radius as a function of time
- **Returns** (*DataFrame*) – Returns the dataframe with additional column `dedz` which is the kinetic energy lost per unit altitude

`calculator_rk4`(*variables*)

Calculate the change of variables at given point

Parameters
variables (*float*) – Angle, radius, altitude, velocity, mass, distance at currenty step

Returns
result – A numpy array containing the change of each variable. Includes the following variables: 'angle', 'radius', 'altitude', 'velocity', 'mass', 'distance'

Return type
ndarray

`create_tabular_density`(*filename='./resources/AltitudeDensityTable.csv'*)

Create a function given altitude return the density of atomosphere using tabulated value

Parameters
filename (*str, optional*) – Path to the tabular. default="./resources/AltitudeDensityTable.csv"

Returns
tabular_density – A function that takes altitude as input and return the density of atomosphere density at given altitude.

Return type
function

`solve_atmospheric_entry`(*radius*, *velocity*, *density*, *strength*, *angle*, *init_altitude=100000.0*, *dt=0.05*, *radians=False*, *backend='RK4'*)

Solve the system of differential equations for a given impact scenario

Parameters

- **radius** (*float*) – The radius of the asteroid in meters

- **velocity** (*float*) – The entery speed of the asteroid in meters/second
- **density** (*float*) – The density of the asteroid in kg/m^3
- **strength** (*float*) – The strength of the asteroid (i.e. the maximum pressure it can take before fragmenting) in N/m^2
- **angle** (*float*) – The initial trajectory angle of the asteroid to the horizontal By default, input is in degrees. If 'radians' is set to True, the input should be in radians
- **init_altitude** (*float, optional*) – Initial altitude in m
- **dt** (*float, optional*) – The output timestep, in s
- **radians** (*logical, optional*) – Whether angles should be given in degrees or radians. Default=False Angles returned in the dataframe will have the same units as the input
- **backend** (*str, optional*) – Which solving method to use. Default='FE'

Returns
Result – A pandas dataframe containing the solution to the system. Includes the following columns: 'velocity', 'mass', 'angle', 'altitude', 'distance', 'radius', 'time'

Return type
DataFrame

solve_atmospheric_entry_FE(*radius*, *velocity*, *angle*, *init_altitude*, *dt*)

Solve the system of differential equations for a given impact scenario using forward Eular method

Parameters

- **radius** (*float*) – The radius of the asteroid in meters
- **velocity** (*float*) – The entery speed of the asteroid in meters/second
- **angle** (*float*) – The initial trajectory angle of the asteroid to the horizontal By default, input is in degrees. If 'radians' is set to True, the input should be in radians
- **init_altitude** (*float, optional*) – Initial altitude in m
- **dt** (*float, optional*) – The output timestep, in s

Return type
None

solve_atmospheric_entry_RK4(*radius*, *velocity*, *angle*, *init_altitude*, *dt*)

Solve the system of differential equations for a given impact scenario using RK4 method

Parameters

- **radius** (*float*) – The radius of the asteroid in meters
- **velocity** (*float*) – The entery speed of the asteroid in meters/second
- **angle** (*float*) – The initial trajectory angle of the asteroid to the horizontal By default, input is in degrees. If 'radians' is set to True, the input should be in radians
- **init_altitude** (*float, optional*) – Initial altitude in m
- **dt** (*float, optional*) – The output timestep, in s

Return type
None

damage.**damage_zones**(*outcome*, *lat*, *lon*, *bearing*, *pressures*, *map=False*)

Calculate the latitude and longitude of the surface zero location and the list of airblast damage radii (m) for a given impact scenario. :param outcome: the outcome dictionary from an impact scenario :type outcome: Dict :param lat: latitude of the meteoroid entry point (degrees) :type lat: float :param lon: longitude of the meteoroid entry point (degrees) :type lon: float :param bearing: Bearing (azimuth) relative to north of meteoroid trajectory (degrees) :type bearing: float :param pressures: List of threshold pressures to define airblast damage levels :type pressures: float, arraylike :param plot: Boolean value to decide plotting :type plot: bool

Returns

- **blat** (*float*) – latitude of the surface zero point (degrees)
- **blon** (*float*) – longitude of the surface zero point (degrees)
- **damrad** (*arraylike, float*) – List of distances specifying the blast radii for the input damage levels
- **plot** (*plot object*) – The plot specifying the areas effected by each damage level

Examples

```
>>> import armageddon
>>> outcome = {'burst_altitude': 8e3, 'burst_energy': 7e3,                    'burst_
↪distance': 90e3, 'burst_peak_dedz': 1e3,                    'outcome': 'Airburst'}
>>> armageddon.damage_zones(outcome, 52.79, -2.95, 135,                              ␣
↪   pressures=[1e3, 3.5e3, 27e3, 43e3])
(51.98371949678459, -2.8338706741653983, [115971.31673025587, 42628.36651535611,␣
↪9575.214234120964, 5835.9834520793875])
```

damage.**impact_risk**(*planet*, *means={'angle': 45, 'bearing': 115.0, 'density': 3000, 'lat': 53.0, 'lon': -2.5, 'radius': 35, 'strength': 10000000.0, 'velocity': 19000.0}*, *stdevs={'angle': 1, 'bearing': 0.5, 'density': 500, 'lat': 0.025, 'lon': 0.025, 'radius': 1, 'strength': 5000000.0, 'velocity': 1000.0}*, *pressure=27000.0*, *nsamples=10*, *sector=True*)

Perform an uncertainty analysis to calculate the risk for each affected UK postcode or postcode sector

Parameters

- **planet** (*armageddon.Planet instance*) – The Planet instance from which to solve the atmospheric entry
- **means** (*dict*) – A dictionary of mean input values for the uncertainty analysis. This should include values for `radius`, `angle`, `strength`, `density`, `velocity`, `lat`, `lon` and `bearing`
- **stdevs** (*dict*) – A dictionary of standard deviations for each input value. This should include values for `radius`, `angle`, `strength`, `density`, `velocity`, `lat`, `lon` and `bearing`
- **pressure** (*float*) – A single pressure at which to calculate the damage zone for each impact
- **nsamples** (*int*) – The number of iterations to perform in the uncertainty analysis
- **sector** (*logical, optional*) – If True (default) calculate the risk for postcode sectors, otherwise calculate the risk for postcodes

Returns

risk – A pandas DataFrame with columns for postcode (or postcode sector) and the associated risk. These should be called `postcode` or `sector`, and `risk`.

Return type

DataFrame

mapping.**plot_circle**(*lat*, *lon*, *radius*, *map=None*, ***kwargs*)

Plot a circle on a map (creating a new folium map instance if necessary).

Parameters

- **lat** (*float*) – latitude of circle to plot (degrees)
- **lon** (*float*) – longitude of circle to plot (degrees)
- **radius** (*float*) – radius of circle to plot (m)
- **map** (*folium.Map*) – existing map object

Return type

Folium map object

Examples

```
>>> import folium
>>> armageddon.plot_circle(52.79, -2.95, 1e3, map=None)
```

<<<<<<< HEAD

PYTHON MODULE INDEX